Loss of Storage Truth in Virtual Distributed Filesystems:

A Formal Critique of CAP and Coordination-Centric Storage

Alexey A. Nekludoff

AstraVerge Research

E-mail: an@astraverge.org
ORCID: 0009-0002-7724-5762

21 January 2026

Report type: Zenodo Report (technical case study)
Disclaimer: The case is anonymized; identifiers are illustrative and do not disclose internal infrastructure details.

Abstract

Modern storage systems increasingly rely on virtualization and distributed coordination to achieve scalability and fault tolerance. While these approaches offer operational flexibility, they also introduce failure modes not captured by traditional performance or availability metrics. This paper introduces the concept of *storage truth* as a foundational property of storage systems, defined as the authoritative existence and addressability of data independent of transient access conditions.

We show that virtual, network-centric storage architectures transform storage truth from a localized physical fact into a model-derived property dependent on continuous coordination. This shift enables a distinct failure mode, termed *loss of storage truth*, in which data remains physically intact yet becomes operationally inaccessible due to degradation in auxiliary subsystems, most notably network coordination.

Using a real-world production incident involving Ceph Software-Defined Storage, the paper demonstrates how minor network degradation can escalate into prolonged loss of service without data loss. Ceph is analyzed as a structurally faithful realization of CAP-style reasoning applied to storage, illustrating how embedding consensus into storage correctness amplifies partial failures into system-wide outages.

The paper critically examines CAP-based design principles in storage systems, showing that defining correctness through coordination rather than physical persistence subordinates storage truth to network state. This design choice is shown to be catastrophic: transient failures outside the storage medium itself can render data operationally nonexistent. Such systems do not fail gracefully; they fail ontologically, as demonstrated by the analyzed production incident.

Finally, the paper formulates an alternative principle of storage design based on locality of truth, separation of truth and access, and eviction over suspension. Practical observations indicate that truth-preserving architectures, such as ZFS combined with iSCSI for distributed access, currently provide superior operational resilience for storage serving as a substrate for virtualized infrastructure.

The central implication of this analysis is that architectural reasoning which treats CAP-style trade-offs as fundamental reflects an insufficient level of professional qualification for the design of distributed storage systems, where correctness is defined by preservation of truth rather than coordination symmetry.

Keywords: storage truth, virtual storage, distributed storage systems, network-centric storage, Ceph, CAP theorem, fault tolerance, virtualized infrastructure, storage architecture, failure semantics.

Contents

1 Introduction **1**

2 What Is Storage Truth **2**
2.1 Storage as a System of Facts . . . 2
2.2 Truth vs. Access . . . 2
2.3 Control Plane and Execution Plane in Storage . . . 2
2.4 Why Storage Truth Is Not Consensus Truth . . . 3
2.5 Failure Semantics of Storage Truth . . . 3
2.6 Implications . . . 3

3 From Storage Truth to Virtual Storage Truth **4**
3.1 Virtualization of Storage . . . 4
3.2 From Physical Facts to Logical Models . . . 4
3.3 Control-Plane Mediation of Storage Truth . . . 4
3.4 Virtual Storage and the Expansion of the Failure Domain . . . 5
3.5 Truth Preservation vs. Truth Reconstruction . . . 5
3.6 Implications . . . 5

4 Ceph Architecture as a Network-Centric Virtual Filesystem **5**
4.1 Ceph as a Virtual Filesystem . . . 6
4.2 Placement as a Function of Network State . . . 6
4.3 Peering and the Suspension of Action . . . 6
4.4 Network as a Constitutive Element of Storage Truth . . . 6
4.5 Truth Reconstruction Instead of Truth Preservation . . . 7
4.6 Ceph as a Case of Network-Centric Virtual Storage Truth . . . 7

5 Incident Description **7**
5.1 Environment . . . 7
5.2 Timeline Overview . . . 7
5.3 Resolution . . . 8
5.4 Observed Impact . . . 8

6 From Degradation to Loss of Storage Truth **8**
6.1 Degradation as an Expected Operating Mode . . . 8
6.2 Escalation Beyond Performance Degradation . . . 9
6.3 Operational Symptoms of Truth Loss . . . 9
6.4 Loss of Truth Without Loss of Data . . . 9
6.5 From Degradation to Ontological Failure . . . 9
6.6 Summary . . . 10

7 Ceph as a Worst-Case Realization of CAP-Style Reasoning **10**
7.1 CAP as an Architectural Justification . . . 10
7.2 From Trade-Off to Structural Amplification . . . 10
7.3 CAP at Its Most Restrictive Point . . . 10
7.4 Implications . . . 11

8 Why the CAP Theorem Does Not Apply to IT Systems, with Storage as a Case Study 11
8.1 Scope and Original Assumptions of CAP . . . 11
8.2 Why IT Systems Do Not Satisfy CAP Preconditions . . . 11
8.3 CAP as a Category Error . . . 12
8.4 Storage as a Counterexample . . . 14
8.5 Partition Tolerance as a Modeling Choice . . . 14
8.6 From Theorem to Misapplied Doctrine . . . 14
8.7 Implications . . . 15

9 Comparative Counterexamples 15
9.1 RAID Arrays: Distribution Without Consensus . . . 15
9.2 ZFS: Localized Truth with Distributed Access . . . 15
9.3 Oracle RAC: Eviction Over Consensus . . . 15
9.4 Kafka: Truth as Order, Not Location . . . 16
9.5 Synthesis . . . 16

10 An Alternative Principle of Storage Design 16
10.1 Locality of Truth . . . 16
10.2 Separation of Truth and Access . . . 17
10.3 Eviction Over Suspension . . . 17
10.4 Truth Preservation vs. Truth Reconstruction . . . 17
10.5 Design Implications . . . 17
10.6 Summary . . . 18

11 Implications for Virtualized Infrastructure 18
11.1 Amplification Through Virtualization Layers . . . 18
11.2 Ineffectiveness of Automation and Self-Healing . . . 18
11.3 Disaster Recovery as the Only Viable Escape . . . 18
11.4 Observability Limits in Virtualized Storage . . . 19
11.5 Architectural Consequences . . . 19
11.6 Summary . . . 19

12 Practical Recommendations 19
12.1 Technology Choice as an Architectural Responsibility . . . 19
12.2 Observed Operational Robustness . . . 20
12.3 ZFS with iSCSI as a Truth-Preserving Baseline . . . 20
12.4 Scope and Limitations . . . 20
12.5 Summary . . . 20

13 Conclusions 21

A Formal Analysis of the CAP Claim 23
A.1 Formal Preconditions of Impossibility Claims . . . 23
A.2 The Domain of the Gilbert–Lynch Result . . . 23
A.3 The Illicit Generalization . . . 23
A.4 Tautological Character of the Result . . . 24
A.5 Conclusion of the Formal Analysis . . . 24

B Formal Counterexample: Sharding **24**
B.1 Definition of the Sharded Model . . . 24
B.2 Local Correctness and Availability . . . 24
B.3 Partition as a Structural Property . . . 25
B.4 System-Level Properties . . . 25
B.5 Interpretation . . . 25
B.6 Conclusion of the Counterexample . . . 25

C Eviction as a Formal Resolution Strategy **26**
C.1 Ambiguity Under Partial Failure . . . 26
C.2 Eviction Operator . . . 26
C.3 Correctness Preservation . . . 26
C.4 Availability Preservation . . . 26
C.5 Comparison with Coordination-Based Suspension . . . 27
C.6 Reintegration . . . 27
C.7 Conclusion of the Resolution Strategy . . . 27

D Suspension as an Anti-Pattern **27**
D.1 Definition of Suspension . . . 27
D.2 Effect on Availability . . . 28
D.3 Effect on Correctness . . . 28
D.4 Dominance by Eviction . . . 28
D.5 Formal Anti-Pattern Characterization . . . 28
D.6 Conclusion of the Anti-Pattern Analysis . . . 28

E Mapping Eviction to Real Systems **29**
E.1 Oracle RAC and Node Eviction . . . 29
E.2 ZFS, Storage Controllers, and Ownership . . . 29
E.3 Failover Controllers and SAN Architectures . . . 29
E.4 Contrast with Coordination-Centric Systems . . . 29
E.5 Conclusion of the Mapping . . . 29

F Formal Taxonomy of Failure Types **30**
F.1 Preliminaries . . . 30
F.2 Partition (Communication Failure) . . . 30
F.3 Ambiguity (Ownership Uncertainty) . . . 30
F.4 Degradation (Performance Failure) . . . 30
F.5 Suspension (Progress Failure) . . . 31
F.6 Corruption (Truth Failure) . . . 31
F.7 Loss of Storage Truth . . . 31
F.8 Summary of Failure Classes . . . 31
F.9 Implications of the Taxonomy . . . 31

1 Introduction

Modern IT infrastructures increasingly rely on deep virtualization and distributed coordination across nearly all layers of the stack. Compute, networking, and storage are commonly abstracted into elastic, software-defined services whose internal structure is hidden behind control planes, orchestration systems, and automated recovery mechanisms. While this approach has enabled unprecedented scalability and operational flexibility, it has also introduced new classes of failure that are not adequately described by traditional notions of fault tolerance, availability, or performance.

In particular, recent large-scale incidents reveal a recurring pattern in which systems remain nominally intact yet lose the ability to operate meaningfully. Control planes continue to function, automation proceeds as designed, and observability reports internally consistent states, while the underlying system fails to provide its essential service. Such failures are neither simple outages nor data loss events; they represent a deeper mismatch between system models and execution reality.

Previous work has identified this phenomenon at the level of compute and orchestration as a loss of execution truth: a condition in which control-plane state diverges from the actual ability of the system to execute work. Storage systems, however, occupy a more foundational position in the infrastructure stack. They serve as substrates for filesystems, databases, virtual machines, and stateful services, and their failure semantics propagate upward into all dependent layers.

This paper argues that contemporary virtualized storage systems introduce an analogous but distinct failure mode: the loss of storage truth. Unlike performance degradation or data corruption, loss of storage truth occurs when a storage system can no longer act authoritatively on stored data, even though the data itself remains physically intact. In such cases, the system loses not the data, but the certainty required to safely access it.

The emergence of this failure mode is closely tied to architectural choices that treat storage as a fully virtualized, network-centric service whose correctness depends on continuous global coordination. While such designs are often motivated by scalability and fault tolerance goals, they implicitly shift the definition of storage correctness away from physical data locality and toward model-derived agreement.

Using a real-world production incident involving Ceph Software-Defined Storage, this paper examines how these design assumptions manifest under realistic failure conditions. The incident demonstrates how minor degradation in an auxiliary subsystem can escalate into prolonged loss of storage availability without physical data loss, affecting a broad range of unrelated services.

Beyond the specific case study, the paper challenges the widespread application of the CAP theorem as a guiding principle for IT system and storage design. It is argued that CAP does not apply to IT systems as a whole and is particularly ill-suited for storage systems whose correctness is grounded in factual data existence rather than negotiated agreement.

The contributions of this paper are threefold. First, it introduces a precise concept of storage truth and distinguishes it from access availability and coordination state. Second, it demonstrates, through both analysis and empirical evidence, how virtual, network-centric storage architectures are structurally vulnerable to loss of storage truth. Third, it formulates an alternative principle of storage design based on localized truth, separation of truth and access, and eviction rather than global suspension.

The remainder of the paper is organized as follows. Section 2 defines storage truth and establishes its role as a foundational property of storage systems. Section 3 examines how virtualization transforms storage truth into a model-dependent construct. Section 4 analyzes Ceph as a network-centric virtual filesystem. Sections 5 and 6 describe and analyze a real-world incident leading to loss of storage truth. Section 7 discusses the limitations of the CAP theorem in the context of IT systems, with storage as a case study. Section 8 presents comparative counterexamples, followed by alternative design principles and practical recommendations. The paper concludes with implications for virtualized infrastructure design.

Previous work has identified this phenomenon at the level of compute and orchestration as a loss of

execution truth [9]: a condition in which control-plane state diverges from the actual ability of the system to execute work.

2 What Is Storage Truth

Before analyzing failure modes of virtual and distributed storage systems, it is necessary to clarify what is meant by *storage truth*. Many contemporary discussions implicitly conflate storage with generic distributed data systems, thereby importing assumptions that are valid for logs, key–value stores, or event streams, but invalid for storage as such. This section introduces a precise notion of storage truth and distinguishes it from related but ontologically weaker concepts.

2.1 Storage as a System of Facts

A storage system is not merely a data transport or replication mechanism. Its primary function is to maintain *facts*: durable, addressable data items whose existence and location must be unambiguous at any given time. For block and file storage, these facts take the form of blocks, extents, or files that are identified by stable logical addresses and are expected to persist independently of transient access conditions.

Crucially, a storage system is required to answer the question: *"Where does this data exist?"* rather than merely *"Can some replica respond?"*. The former is a question of fact; the latter is a question of availability. While availability may degrade, the factual status of stored data must remain well-defined.

We therefore define storage truth as follows.

Definition 1 (Storage truth). Storage truth is the property of a storage system by which the existence, identity, and authoritative location of stored data are uniquely determined and invariant under transient failures of access paths or auxiliary subsystems.

This definition deliberately excludes performance characteristics such as latency, throughput, or replication lag. A system may be slow or temporarily inaccessible and yet preserve storage truth, provided that the factual state of stored data remains well-defined.

2.2 Truth vs. Access

An essential distinction must be made between *truth* and *access*. Access mechanisms—networks, protocols, clients, and front-end services—are means by which data is reached, but they are not themselves constitutive of the data's existence.

In traditional storage architectures (e.g., RAID arrays, local filesystems, or centralized storage appliances), this separation is explicit. If an access path fails, the storage system does not lose its identity or internal coherence; it merely becomes unreachable through that path. When access is restored, the same factual state is observed.

This separation can be summarized as:

Access may fail without invalidating truth.

Any storage architecture that violates this principle risks transforming transient transport or coordination failures into existential failures of the storage system itself.

2.3 Control Plane and Execution Plane in Storage

As with compute orchestration, storage systems can be analyzed in terms of control and execution planes.

The *storage control plane* consists of metadata services, placement algorithms, cluster maps, and coordination protocols that describe how data should be laid out and accessed. The *storage execution plane* consists of the actual devices and processes that perform reads and writes on physical media.

Under normal operation, the control plane provides an abstract representation of storage truth. However, this representation is only valid insofar as it remains aligned with execution-plane reality. When control-plane assumptions become dependent on subsystems that do not directly store data (most notably the network), this alignment becomes fragile.

2.4 Why Storage Truth Is Not Consensus Truth

A common error in the design of modern distributed storage systems is the implicit identification of storage truth with consensus truth. In this view, data is considered valid if a sufficient number of participants agree on its state.

This identification is appropriate for systems whose primary abstraction is an ordered log or a stream of events, where truth is defined by sequence and can tolerate temporary divergence. Storage systems, however, operate on addressable state rather than append-only history. A block or file cannot be "mostly correct" or "eventually located"; it either exists at a specific address or it does not.

Consensus mechanisms answer the question: *"Do participants agree?"*

Storage systems must answer the question: *"Where is the authoritative copy?"*

These are not equivalent. Agreement does not create factual locality; it merely asserts a shared belief. When belief becomes a prerequisite for access, storage truth is no longer intrinsic to the data, but contingent on coordination success.

2.5 Failure Semantics of Storage Truth

The defining test of a storage system is not how it behaves under nominal conditions, but how it degrades under partial failure.

A storage system that preserves storage truth exhibits the following failure semantics:

- Access paths may degrade or fail independently.
- Performance may collapse without invalidating data identity.
- Individual participants may be evicted or fenced without global loss of service.
- Recovery restores access to the same factual state.

Conversely, loss of storage truth occurs when a system can no longer determine, or refuses to act upon, the authoritative location of data due to failures in coordination, communication, or agreement mechanisms that are external to the data itself.

Definition 2 (Loss of storage truth). Loss of storage truth is a condition in which a storage system retains internal control-plane consistency yet loses the ability to determine or act upon the authoritative existence or location of stored data, despite the absence of physical data destruction.

This definition intentionally separates loss of storage truth from data loss. In the former case, data may remain intact on physical media, yet be rendered operationally unavailable by architectural design choices.

2.6 Implications

The concept of storage truth establishes a clear criterion for evaluating storage architectures: any system in which the determination of data existence depends on continuous global coordination, rather than on localized factual ownership, is structurally vulnerable to loss of storage truth.

The following sections will show that this vulnerability is not incidental but systematic in virtualized, network-centric storage designs, and that it manifests predictably under realistic failure conditions.

3 From Storage Truth to Virtual Storage Truth

The notion of storage truth introduced in the previous section presupposes a direct relationship between stored data and its physical realization. Classical storage systems, even when network-accessible, retain a clear locality of truth: data exists at specific physical locations, and access mechanisms merely mediate interaction with that locality. Virtualized storage systems alter this relationship in a fundamental way.

This section introduces the concept of *virtual storage truth* and shows how virtualization transforms storage from a system of facts into a system of models. We argue that this transformation, while operationally convenient, introduces new failure modes in which storage truth is no longer grounded in physical reality but becomes contingent on coordination and representation.

3.1 Virtualization of Storage

Virtualization in storage systems abstracts physical media into logical entities: volumes, objects, pools, or virtual disks. These abstractions are designed to decouple consumers of storage from the underlying hardware, enabling mobility, elasticity, and multi-tenancy.

In a virtualized storage system, clients do not interact with physical devices directly. Instead, they operate on logical identifiers whose resolution into physical locations is performed dynamically by the system. As a result, the answer to the question "where does this data exist?" is no longer a static fact but a computed outcome derived from metadata, placement rules, and system state.

This indirection is the defining feature of virtual storage.

3.2 From Physical Facts to Logical Models

In non-virtualized storage, the mapping between logical addresses and physical media is stable and largely implicit. In virtual storage systems, this mapping becomes explicit, mutable, and algorithmic. Placement decisions may depend on cluster topology, failure domains, replication policies, and current system conditions.

Consequently, storage truth is no longer embedded in the medium itself but is represented by a *model* maintained by the system's control plane. The existence and location of data are inferred rather than directly known.

We refer to this inferred notion as *virtual storage truth*.

Definition 3 (Virtual storage truth). Virtual storage truth is the property of a storage system by which the existence and authoritative location of data are determined by a maintained logical model rather than by a fixed physical binding.

Virtual storage truth is not inherently incorrect or unsafe. However, it introduces a critical dependency: the validity of storage truth now depends on the correctness, availability, and coherence of the model that represents it.

3.3 Control-Plane Mediation of Storage Truth

Because virtual storage truth is model-based, the storage control plane becomes the mediator of truth. Metadata services, placement algorithms, and coordination protocols collectively determine what is considered authoritative.

This mediation has two immediate consequences.

First, storage truth becomes temporally sensitive. Changes in system state (network partitions, node failures, or load-induced delays) can alter the model even when the underlying data remains physically unchanged.

Second, storage truth becomes observationally fragile. Different components may observe different subsets of system state, leading to transient disagreement about data placement or availability.

Unlike classical storage systems, virtualized storage cannot always fall back to a purely local notion of truth when coordination degrades.

3.4 Virtual Storage and the Expansion of the Failure Domain

Virtualization expands the failure domain of storage systems. Failures that were previously confined to access paths or individual devices can now affect the entire system's ability to reason about data.

In particular, failures in auxiliary subsystems such as networks or coordination services can invalidate the logical model on which virtual storage truth depends. When this occurs, the system may lose the ability to determine where data is authoritatively located, even if the data itself remains intact on physical media.

This represents a qualitative shift in failure semantics:

Physical storage systems lose access under failure; virtual storage systems may lose truth.

3.5 Truth Preservation vs. Truth Reconstruction

A further distinction arises between systems that *preserve* storage truth and those that attempt to *reconstruct* it.

In truth-preserving systems, authoritative data ownership remains localized, and recovery restores access to an already well-defined state. In truth-reconstructing systems, the authoritative state is recomputed through coordination and agreement after failures.

While reconstruction is acceptable for certain classes of data (e.g., replicated logs or caches), it is fundamentally problematic for storage systems that serve as substrates for filesystems, databases, or virtual machine disks. In these contexts, the inability to determine truth without reconstruction constitutes an operational hazard.

3.6 Implications

The transition from storage truth to virtual storage truth marks a critical architectural boundary. Virtualization does not merely add flexibility; it changes the ontological status of stored data from a physical fact to a model-derived assertion.

Systems that adopt virtual storage truth must therefore be evaluated not only on performance and scalability, but on their ability to maintain truth under partial failure. When truth becomes dependent on continuous global coordination, the storage system inherits the fragility of that coordination.

The next section examines a prominent example of this design choice and shows how a network-centric interpretation of virtual storage truth leads to systematic loss of storage truth under realistic failure conditions.

4 Ceph Architecture as a Network-Centric Virtual Filesystem

Ceph is commonly described as a distributed storage system capable of providing block, object, and file interfaces. From an architectural perspective, however, Ceph should be understood more precisely as a *network-centric virtual filesystem*: a system in which data placement, identity, and accessibility are defined not by physical locality but by continuous coordination over a networked control plane.

This section analyzes Ceph through the lens of virtual storage truth and shows that its core design choices make network coherence a constitutive element of storage truth itself.

4.1 Ceph as a Virtual Filesystem

Despite supporting multiple access modalities (RBD, object storage, and CephFS), Ceph presents a unified abstraction: all data is represented as objects managed by a distributed placement and metadata layer. Physical disks and nodes do not own data in an authoritative sense; instead, ownership is virtualized and expressed through cluster state.

In this sense, Ceph is not merely a storage backend but a virtual filesystem whose namespace and block addressing are resolved dynamically. Clients never interact with a stable physical location. Every read or write operation is mediated by placement logic that maps logical identifiers to a set of object storage daemons based on current cluster topology.

As a result, the question "where does this data exist?" has no static answer in Ceph. It is resolved algorithmically at access time.

4.2 Placement as a Function of Network State

At the heart of Ceph lies a placement function that determines which storage daemons are responsible for a given object. While this function is deterministic in form, its inputs are not constant. They depend on the current cluster map, which reflects node availability, failure domains, and perceived liveness.

Crucially, the cluster map is maintained and propagated through network-based coordination. Changes in network connectivity, latency, or message loss directly affect the system's perception of which components are alive and therefore which placements are valid.

Thus, object placement in Ceph is not purely a function of data identity but a function of *network-observed system state*. The authoritative location of data is inseparable from the system's ability to maintain coherent communication among its participants.

4.3 Peering and the Suspension of Action

Ceph introduces the notion of peering to reconcile object state among replicas. Before a placement group can accept I/O, its participating daemons must agree on their collective state and history. This agreement is established through network exchanges and time-bounded coordination.

When network conditions degrade, peering may fail to complete. In such cases, Ceph deliberately suspends read and write operations for the affected placement groups. This behavior is often described as conservative or correctness-preserving. From the perspective of storage truth, however, it reveals a deeper property: Ceph cannot act without network-mediated agreement.

The system does not merely lose performance under degraded coordination; it loses the ability to assert authoritative data ownership. Action is suspended not because data is damaged, but because truth can no longer be confirmed through coordination.

4.4 Network as a Constitutive Element of Storage Truth

In classical storage systems, the network is an access path. Its failure limits reachability but does not invalidate the factual existence of data. In Ceph, by contrast, the network is part of the mechanism by which storage truth is defined.

Storage truth in Ceph depends on:

- timely dissemination of cluster state;
- consistent observation of daemon liveness;
- successful completion of peering protocols;
- agreement on placement group histories.

Each of these conditions is contingent on network behavior. As a result, storage truth in Ceph is not localized to disks or nodes but distributed across a network of observers whose agreement is continuously required.

This represents a qualitative departure from truth-preserving storage designs. Ceph does not merely virtualize access to storage; it virtualizes storage truth itself.

4.5 Truth Reconstruction Instead of Truth Preservation

Ceph is designed to reconstruct authoritative state after failures through recovery and rebalancing. When components fail or rejoin, data ownership is recomputed and, if necessary, redistributed.

This approach implicitly treats storage truth as something that can be reconstructed through coordination, rather than preserved through localized ownership. While this is viable for systems whose data model tolerates temporary ambiguity, it is problematic for storage substrates that serve as the foundation for filesystems, databases, and virtual machine disks.

In these contexts, suspension of I/O due to uncertainty is operationally equivalent to loss of storage truth, even if the underlying data remains intact.

4.6 Ceph as a Case of Network-Centric Virtual Storage Truth

The preceding analysis shows that Ceph exemplifies a broader class of systems in which virtual storage truth is defined by network-centric coordination. In such systems, failures of communication are elevated from access impairments to existential threats to storage operation.

This design choice is not an implementation flaw but a direct consequence of treating consensus and coordination as foundational to storage correctness. As the following sections will demonstrate, this choice has predictable and severe consequences under realistic failure conditions and motivates a reconsideration of the formal assumptions underlying distributed storage design.

5 Incident Description

This section provides a concise factual description of a production incident involving a Ceph-based virtual storage cluster. The purpose of this section is not analysis or attribution, but the establishment of an empirical reference case that will be examined in the following sections.

5.1 Environment

The incident occurred in a virtualized infrastructure environment relying on Ceph Software-Defined Storage as a shared storage backend for multiple critical services. The cluster under consideration provided storage to a heterogeneous set of applications, including collaboration platforms, payment-related services, mediation systems, monitoring infrastructure, and internal file services.

The storage cluster was deployed on NVMe-backed object storage daemons and connected via a leaf–spine network fabric with high-speed (100 GbE) links. The environment was designed to tolerate individual component failures and to support online recovery and rebalancing operations.

5.2 Timeline Overview

The incident unfolded in several stages.

An initial pre-incident phase was characterized by elevated disk latency and performance degradation observed by multiple consumers of the storage platform. At this stage, the system remained operational,

but service owners were advised to reduce load, migrate workloads to other clusters, or activate alternative instances.

As the situation progressed, the degradation escalated into a storage-level incident. Multiple applications experienced sustained impact, while others were forced to switch to disaster recovery configurations. Operational teams investigated a range of hypotheses, including internal Ceph throttling effects, CPU-level contention, and locking-related pathologies, and undertook various mitigation actions such as host replacement, data relocation, and metadata maintenance operations.

The incident persisted for several days, affecting different services at different times, until normal operation was eventually restored.

5.3 Resolution

After extended investigation and mitigation efforts, the root cause of the incident was identified as degradation on a specific network link within the storage fabric. Replacement of a single optical transceiver module restored network stability, after which the Ceph storage service returned to normal operation.

Notably, recovery did not require disk replacement, data reconstruction, or restoration from backup. Stored data remained physically intact throughout the incident period.

5.4 Observed Impact

The total duration of service degradation and partial unavailability spanned multiple days. During this time, critical production services experienced performance collapse, forced failover, or operational suspension. The impact was not confined to a single application domain but propagated across unrelated services sharing the same virtual storage substrate.

This incident thus provides a representative case in which prolonged loss of storage availability occurred in the absence of physical data loss, triggered by degradation in an auxiliary subsystem external to the storage media itself.

6 From Degradation to Loss of Storage Truth

The incident described in the previous section did not begin as a sudden failure. Instead, it evolved through a sequence of increasingly severe degradations, each of which was initially interpreted as a performance or capacity issue rather than as a threat to the integrity of storage operation. This section analyzes the transition from ordinary degradation to a qualitative loss of storage truth and identifies the point at which the storage system ceased to function as a system of facts. This marks the transition from performance failure to loss of storage truth as defined in Definition 2.

6.1 Degradation as an Expected Operating Mode

In large-scale virtualized infrastructures, transient degradation is a familiar and often tolerated condition. Elevated latency, reduced throughput, and temporary resource contention are commonly managed through load shedding, horizontal scaling, or workload relocation. Such responses implicitly assume that the underlying storage system continues to preserve storage truth: data is still assumed to exist at well-defined locations, and corrective actions are expected to restore acceptable access.

During the initial phase of the incident, this assumption guided operational response. Consumers were advised to reduce load or migrate workloads, and additional capacity was provisioned in other clusters. These actions are appropriate when degradation is confined to performance and does not undermine the system's ability to determine authoritative data ownership.

6.2 Escalation Beyond Performance Degradation

As the incident progressed, the effectiveness of performance-oriented mitigations diminished. Despite redistribution of load and partial evacuation of affected hosts, storage latency remained elevated and service impact persisted across multiple, otherwise unrelated applications.

At this stage, the incident could no longer be explained solely in terms of resource saturation or local contention. Mitigation efforts increasingly targeted internal storage mechanisms, including throttling behavior, locking pathologies, and metadata maintenance. These actions implicitly acknowledged that the storage system's internal coordination mechanisms had become part of the failure surface.

The persistence of impact despite these measures indicates that the system had entered a regime in which performance degradation alone could not account for observed behavior.

6.3 Operational Symptoms of Truth Loss

A critical transition occurred when standard storage-level interventions ceased to produce meaningful improvement, while the physical data remained intact. Applications experienced sustained inability to perform I/O, yet no data loss, disk failure, or corruption was observed.

This combination of symptoms is characteristic of loss of storage truth. The system did not merely slow down; it lost the ability to act authoritatively on stored data. Although objects and blocks continued to exist on physical media, the storage system could no longer reliably determine or assert their authoritative placement in a way that permitted safe operation.

From an operational perspective, this manifested as follows:

- prolonged unavailability of storage-backed services without data loss;
- ineffectiveness of internal recovery and rebalancing mechanisms;
- reliance on external actions such as workload migration and disaster recovery failover;
- eventual resolution through restoration of an auxiliary subsystem rather than through storage-level repair.

These symptoms distinguish loss of storage truth from conventional storage degradation or failure.

6.4 Loss of Truth Without Loss of Data

A defining feature of the incident is that recovery did not involve data reconstruction, disk replacement, or restoration from backup. Instead, service was restored immediately after network stability was reestablished. This demonstrates that the failure mode was not rooted in the physical storage layer but in the system's inability to maintain a coherent model of storage truth under degraded coordination.

In other words, the system behaved as if data had become unavailable or unsafe to access, even though the data itself was neither lost nor damaged. Storage truth was not destroyed; it was rendered operationally inaccessible by the system's own architecture.

6.5 From Degradation to Ontological Failure

The transition observed in this incident marks a boundary between quantitative and qualitative failure modes. Performance degradation represents a quantitative change in system behavior. Loss of storage truth represents a qualitative change in the nature of the system itself.

Once storage truth is lost, further optimization or mitigation within the same architectural framework becomes ineffective. The system can no longer rely on its internal mechanisms to recover, because those mechanisms presuppose the very truth that has been lost. At this point, only external intervention—such as failover to an independent storage instance or restoration of coordination infrastructure—can reestablish operation.

This transition explains why the incident persisted for multiple days despite ongoing mitigation efforts and why resolution occurred abruptly once the underlying coordination impairment was removed.

6.6 Summary

The analyzed incident illustrates a clear progression from tolerable degradation to loss of storage truth. The decisive factor in this progression was not the severity of load or the failure of storage media, but the erosion of the system's ability to maintain authoritative knowledge of data placement.

This observation motivates a deeper examination of the architectural assumptions that allow auxiliary subsystems, particularly networks, to participate in the definition of storage truth. The following sections examine these assumptions formally and show why they lead to structurally fragile storage designs.

7 Ceph as a Worst-Case Realization of CAP-Style Reasoning

The behavior observed in the analyzed incident is often informally justified using references to the CAP theorem, with the argument that the storage system sacrificed availability in order to preserve consistency under network degradation. While this explanation is common, it obscures a more important point: Ceph does not merely exhibit a CAP trade-off, but realizes its most pathological form when applied to storage systems.

In this sense, Ceph can be understood as a worst-case embodiment of CAP-style reasoning in the domain of storage.

7.1 CAP as an Architectural Justification

Within the CAP framework, a system experiencing network partition must choose between consistency and availability. Ceph is frequently characterized as a CP-oriented system: when coordination is impaired, it prefers to suspend I/O rather than risk inconsistent state.

Taken at face value, this choice appears principled. However, this framing implicitly assumes that the CAP model is appropriate for the class of systems under consideration. The incident described earlier demonstrates that, when applied to storage substrates, this assumption leads to failure modes that are operationally disproportionate to the underlying fault.

7.2 From Trade-Off to Structural Amplification

In the analyzed case, a localized degradation in a single network component resulted in a multi-day outage of storage-backed services. No data was lost, no disks failed, and no irrecoverable corruption occurred. Nevertheless, the system entered a state in which it could not safely perform I/O.

This outcome is not a marginal effect of CAP trade-offs but an amplification of them. By embedding network-mediated agreement into the definition of storage correctness, Ceph elevates partial communication failure from an access problem to an existential condition for storage operation.

Availability is not merely reduced; it is categorically withdrawn.

7.3 CAP at Its Most Restrictive Point

Ceph's design places it at the most restrictive corner of the CAP space. Any loss of confidence in global coordination triggers suspension of action, even when authoritative data continues to exist on physical media. In effect, the system treats uncertainty about agreement as equivalent to uncertainty about data existence.

This behavior represents a worst-case realization of CAP reasoning: the system voluntarily relinquishes its primary function—serving stored data—in order to maintain an internally consistent model of state.

7.4 Implications

The significance of this observation is not that Ceph chose consistency over availability, but that the choice itself was made at an inappropriate level of abstraction. By applying CAP-style consensus logic to a storage substrate, Ceph transforms auxiliary failures into prolonged loss of service without physical data loss.

This section establishes Ceph as a concrete, operational example of how CAP reasoning, when uncritically applied to storage systems, leads to pathological outcomes. The following section examines the formal assumptions underlying the CAP theorem and shows why these assumptions do not hold for storage systems as a class.

8 Why the CAP Theorem Does Not Apply to IT Systems, with Storage as a Case Study

The CAP theorem is frequently presented as a fundamental and formally established limitation of distributed systems. In reality, CAP is not a universal formal result but a modeling artifact whose conclusions depend entirely on specific and often unspoken assumptions. Treating CAP as a universal law of IT system design is a categorical mistake. Storage systems, where data existence is a physical fact rather than a negotiated property, provide a clear counterexample to CAP-style reasoning.

8.1 Scope and Original Assumptions of CAP

The CAP theorem was formulated for a specific class of systems: distributed systems in which state is replicated across multiple nodes, communication is asynchronous, and network partitions are treated as a normal and unavoidable condition. Within this model, operations are defined in terms of client-visible responses to requests, and correctness is evaluated with respect to abstract state consistency.

These assumptions implicitly constrain the theorem's domain of applicability. In particular, CAP presupposes that:

- system state is inherently distributed;
- no single node possesses authoritative ownership of the state;
- agreement among replicas defines correctness;
- partitions are modeled as first-class failure modes.

Systems that violate any of these assumptions fall outside the scope for which CAP provides meaningful guidance.

Similar category errors arising from control-plane abstractions have been previously analyzed in the context of compute orchestration and scheduling systems [9]. This mismatch between formal models and operational reality reflects a broader epistemic issue discussed in the context of coherent observational frameworks [8].

8.2 Why IT Systems Do Not Satisfy CAP Preconditions

General IT systems—compute platforms, storage infrastructures, and networked services—do not constitute the class of systems to which CAP-style abstractions meaningfully apply, because their fundamental state is not inherently distributed. In many such systems, state is not fundamentally distributed but localized, with clear ownership and well-defined authority boundaries.

For example, in storage systems based on localized media, data exists as a physical fact on specific devices. Replication, mirroring, or backup mechanisms do not distribute ownership of that fact; they merely create copies. The failure of communication between components does not erase or invalidate the existence of data, but only affects access to it.

Similarly, in compute and orchestration systems, execution state ultimately resides in physical processors and memory. Control planes may lose visibility or control, but the execution substrate remains authoritative regardless of network conditions, exposing coordination failure rather than loss of state.

These characteristics place such systems outside the CAP abstraction, which treats data existence as a distributed quantity to be reconstructed via consensus.

On Formal Status A necessary precondition for any statement to assert constraints on a class of systems is formalization. For a claim to possess mathematical or theoretical status, it must define its object, specify the domain of quantification, and be expressed within an explicit system of assumptions and inference rules. What is not formalized asserts nothing and constrains nothing.

The construct commonly referred to as the "CAP theorem" does not satisfy these conditions. It has never been formulated as a theorem within a formal axiomatic framework, nor has it been presented as a theory with a well-defined object and scope. As a result, CAP cannot impose necessary limits on the design of IT systems; it can at most describe properties of particular abstract models once their assumptions are explicitly fixed.

This distinction is not semantic but foundational. Without formal status, claims of impossibility cannot be elevated from heuristic observations to normative principles of system design.

8.3 CAP as a Category Error

The persistent influence of the CAP theorem in the design and justification of IT systems rests on a fundamental category error. CAP is routinely treated as a general law of distributed reality, whereas in fact it is a statement about a narrow class of abstract models that do not provide an ontology of system state, existence, or authority.

At the core of CAP lies an implicit identification of *truth* with *agreement*. System correctness is defined in terms of consistent responses observed across replicas, and state is assumed to exist only insofar as it is collectively acknowledged. Within this abstraction, network partitions become existential events: without agreement, truth itself is considered undefined.

This abstraction is categorically incompatible with most IT systems.

In storage systems, execution environments, and physical infrastructure, correctness is grounded in *factual existence*, not in negotiated consensus. Data exists on specific media regardless of whether other components can observe or agree upon its state. Computation executes on physical processors regardless of control-plane visibility. These facts do not disappear under partition; only access and observation are impaired.

By conflating agreement with existence, CAP commits a category error: it applies a model of consensus-defined truth to systems whose truth is independent of consensus. This error is not merely philosophical but operational. When adopted as a design principle, it leads to architectures that suspend action whenever coordination is degraded, even in the presence of intact and authoritative state.

Partition tolerance provides a particularly clear example of this misclassification. In CAP-style reasoning, partition is elevated to a defining condition of system correctness. In real IT systems, partition is a failure of communication or observation. Treating it as an ontological rupture rather than as an access impairment transforms recoverable faults into global outages.

The incident analyzed in this paper demonstrates this effect empirically. Authoritative data persisted throughout the failure, yet the storage system refused to act due to loss of coordination. This behavior does not follow from any physical necessity; it follows from the adoption of a model in which truth must be continuously revalidated through agreement. The resulting outage is not an unavoidable trade-off but an artifact of the chosen abstraction.

For these reasons, CAP cannot be regarded as a true statement about IT systems. It is false as a general theory because its premises do not describe the ontological structure of the systems to which it is applied. When used outside its narrow domain of applicability, CAP does not illuminate design constraints; it obscures them and legitimizes architectures that are structurally fragile.

Recognizing CAP as a category error clarifies why alternative designs—those that localize truth, separate truth from access, and resolve ambiguity through eviction rather than suspension—exhibit more predictable and resilient behavior under failure. It also explains why invoking CAP to justify loss of service in the presence of intact data is not an explanation, but a post hoc rationalization.

A necessary precondition for any statement to assert constraints on a class of systems is formalization. What is not formalized asserts nothing and constrains nothing. In the absence of a formally defined object, axioms, and inference rules, claims of necessity or impossibility have no mathematical status and cannot serve as normative limits on system design.

Statement (CAP is false). The CAP theorem is false as a theory of IT systems. Its premises do not correspond to the ontology of real infrastructure, where truth exists independently of agreement and partitions impair access rather than existence. Moreover, CAP has never existed as a mathematical theorem: it has never been formulated within a formal axiomatic system, nor proven as a theorem by any author. What is commonly referred to as the "CAP theorem" is an informal conjecture about a narrow class of abstract coordination models, later elevated to a design doctrine. Treating CAP as a fundamental law transforms this abstraction into normative engineering guidance and legitimizes avoidable loss of service in the presence of intact and authoritative state.

In summary, CAP is neither a theorem nor a theory. It is an informal conjecture about a narrowly defined coordination model, later generalized without formal justification to systems whose ontological structure it does not describe.

On the Gilbert–Lynch result. The result commonly attributed to Gilbert and Lynch is frequently presented as a formal proof of the CAP theorem. This characterization is misleading. Their work does not establish a general theorem about IT systems, nor does it reveal a previously unknown constraint of distributed reality. Instead, it proves a conditional statement within a narrowly specified abstract model whose premises already encode the desired conclusion.

Formally, the result of Gilbert and Lynch establishes an implication within a restricted class of models rather than a universal constraint on IT systems. Let $\mathcal{M}$ denote the class of models defined by the following premises: (i) system state is inherently distributed across replicas, (ii) correctness is defined exclusively in terms of replica agreement, and (iii) network partitions are treated as first-class, unavoidable conditions.

Within $\mathcal{M}$, the authors prove that certain response properties cannot be simultaneously guaranteed. Symbolically, the result has the form

$$\forall M \in \mathcal{M},\ \neg(C(M) \wedge A(M) \wedge P(M)).$$

However, this statement does not extend beyond $\mathcal{M}$. In particular, it does not imply

$$\forall S \in \mathcal{S}_{IT},\ \neg(C(S) \wedge A(S) \wedge P(S)),$$

where $\mathcal{S}_{IT}$ denotes the class of real IT systems.

The impossibility result follows directly from the defining assumptions of $\mathcal{M}$ and therefore proves only what is already presupposed: that systems whose notion of truth is entirely coordination-defined cannot operate when coordination is impaired. This is a tautological consequence of the chosen formalization, not a discovery about the ontology of real systems.

Specifically, the model assumes that system state is inherently distributed, that correctness is defined exclusively in terms of replica agreement, and that network partitions are treated as first-class, unavoidable conditions. Under these assumptions, the impossibility of simultaneously guaranteeing certain response properties follows trivially. What is demonstrated, therefore, is not a fundamental trade-off of real systems, but the internal consistency of a self-imposed abstraction.

In other words, the result proves precisely what it presupposes: that systems designed such that truth exists only through coordination cannot operate when coordination is impaired. This is not a discovery about IT systems, but a tautological consequence of the chosen formalization. Elevating this result to a universal theorem about storage, execution, or infrastructure correctness constitutes a category error rather than a scientific insight.

8.4 Storage as a Counterexample

Storage systems provide a particularly clear counterexample to CAP reasoning. The primary obligation of a storage system is not to respond to requests under all conditions, but to preserve the identity, integrity, and authoritative location of data.

In storage, correctness is not defined by majority agreement but by factual existence. A block either exists at a given address or it does not. This fact remains true regardless of whether other components can currently observe or confirm it.

Applying CAP-style trade-offs to storage therefore conflates two distinct concerns:

- the availability of access paths to data;
- the existence and correctness of the data itself.

While availability may be legitimately sacrificed under certain failure modes, suspending operation due to uncertainty about agreement does not improve storage correctness. Instead, it converts auxiliary failures into storage-level outages without preventing data loss.

8.5 Partition Tolerance as a Modeling Choice

A critical but often overlooked aspect of CAP is that partition tolerance is not a universal requirement but a modeling choice. Treating network partitions as defining conditions of correctness presupposes that the system has no authoritative local state to fall back on.

In many IT systems, partitions are better understood as failures of access or coordination rather than as existential threats. Systems such as RAID arrays, local filesystems, and centralized databases continue to preserve correctness under partition by isolating or evicting affected components rather than by suspending operation globally.

When partition tolerance is elevated from an access concern to a truth-defining constraint, the system inherits the full fragility of its coordination layer.

8.6 From Theorem to Misapplied Doctrine

The widespread invocation of CAP in the design and justification of IT systems has transformed a narrow theoretical result into a generalized doctrine. This shift obscures the fact that CAP does not describe an unavoidable law of distributed reality, but a trade-off within a particular abstraction.

When applied outside its intended domain, CAP becomes less a theorem and more a rationalization for architectural choices that embed coordination into the definition of correctness. The resulting systems may be internally consistent yet operationally brittle.

8.7 Implications

The analysis above shows that CAP is inapplicable to real-world IT systems. Practical system design must be grounded in an explicit ontology of what exists and where authority resides, not in abstract coordination fantasies. Storage systems demand preservation of factual truth rather than coherence constructed through coordination protocols. Treating consensus as a prerequisite for correctness transforms recoverable access impairments into prolonged loss of service.

The storage incident analyzed in this work illustrates how adherence to CAP-style reasoning can produce failure modes that are disproportionate to the underlying fault. This observation motivates the need for alternative design principles that distinguish between truth, access, and coordination.

The following section formulates such principles and outlines a storage design approach that preserves truth under failure without relying on global consensus.

9 Comparative Counterexamples

To demonstrate that loss of storage truth is not an inherent property of large, complex, or distributed systems, this section examines several well-established architectures that operate under comparable failure conditions yet do not exhibit the failure mode observed in Ceph. These systems differ in scale, complexity, and implementation, but share a common architectural property: they preserve localized storage or execution truth and treat coordination as auxiliary rather than constitutive.

9.1 RAID Arrays: Distribution Without Consensus

Redundant Array of Independent Disks (RAID) systems distribute data across multiple physical devices for performance and fault tolerance. Despite this distribution, RAID arrays do not require consensus among components to define correctness.

If a disk fails or an access path is lost, the array degrades locally. Data remains addressable and authoritative on the remaining devices, and the system continues to operate within reduced redundancy constraints. Importantly, RAID controllers do not suspend operation due to uncertainty about global agreement; they act based on locally available facts.

This demonstrates that data distribution alone does not necessitate coordination-driven correctness or global suspension of action.

9.2 ZFS: Localized Truth with Distributed Access

ZFS combines local storage management with strong integrity guarantees, including end-to-end checksumming and copy-on-write semantics. While ZFS is frequently deployed behind network protocols such as iSCSI or NFS, the network serves strictly as an access mechanism.

Under network degradation, ZFS pools do not lose internal coherence or authoritative knowledge of stored data. Access may stall or fail, but the storage system itself remains intact and truthful. When connectivity is restored, access resumes to the same factual state.

ZFS thus exemplifies a design in which truth is localized to storage media, while access is distributed and potentially unreliable.

9.3 Oracle RAC: Eviction Over Consensus

Oracle Real Application Clusters (RAC) operate with multiple database instances coordinated over a private interconnect. Despite this reliance on internal communication, RAC does not treat network agreement as a prerequisite for data existence.

When coordination degrades or a node becomes unresponsive, RAC employs eviction mechanisms to remove the offending participant. Remaining nodes continue to operate against the same authoritative storage substrate. The system resolves ambiguity by reducing participation, not by suspending operation.

This approach highlights a critical alternative to CAP-style reasoning: preserving truth by narrowing the set of actors, rather than halting action until agreement is restored.

9.4 Kafka: Truth as Order, Not Location

Apache Kafka represents a different class of system in which truth is defined by ordered append-only logs rather than by addressable state. In this model, data correctness depends on sequence, not on physical placement.

Kafka tolerates temporary replica divergence and degraded coordination by allowing leaders to continue accepting writes while followers lag behind. Agreement is eventually restored through log replication, but availability is maintained without requiring immediate global consensus.

While Kafka is not a storage substrate in the classical sense, it demonstrates that even systems with strong consistency guarantees can decouple truth from continuous coordination, provided their data model permits monotonic progress.

9.5 Synthesis

These counterexamples illustrate that loss of storage truth is not an inevitable consequence of scale, distribution, or fault tolerance. Instead, it arises from a specific architectural choice: embedding coordination and agreement into the definition of correctness.

Systems that localize truth and treat coordination as an optimization or access concern degrade gracefully under partial failure. Systems that distribute truth and require continuous agreement risk transforming minor impairments into system-wide outages.

Ceph stands apart from the examined counterexamples in that it elevates network coherence from a performance concern to a condition of existence. This choice, rather than distribution itself, explains the observed loss of storage truth under realistic failure conditions.

10 An Alternative Principle of Storage Design

The preceding analysis shows that loss of storage truth is not an accidental failure mode but a predictable consequence of specific architectural choices. In particular, designs that embed coordination and consensus into the definition of storage correctness systematically amplify auxiliary failures into prolonged service outages. This section formulates an alternative principle of storage design that avoids these pathologies while remaining compatible with fault-tolerant and scalable systems.

10.1 Locality of Truth

The foundational requirement of any storage system is the preservation of storage truth. This requires that authoritative data ownership be localized and independent of transient coordination state.

Principle 1 (Locality of storage truth). At any moment, the authoritative existence and location of stored data must be determined by a localized execution context, not by global agreement among participants.

This principle does not prohibit replication or redundancy. Rather, it distinguishes between the existence of a fact and the existence of its copies. Replication distributes availability; it must not distribute authority over the fact itself.

Systems that satisfy this principle retain the ability to act decisively under partial failure, because truth does not depend on the continued participation of all components.

10.2 Separation of Truth and Access

A second principle follows directly from the first.

Principle 2 (Separation of truth and access). Failures of access paths, coordination channels, or auxiliary services must not invalidate storage truth; they may only limit reachability or performance.

Under this principle, networks, control planes, and coordination mechanisms are treated as access infrastructure rather than as constituents of correctness. Their failure degrades service but does not suspend the storage system's ability to determine authoritative state.

This separation ensures that recovery restores access to an already well-defined state rather than attempting to reconstruct truth through coordination.

10.3 Eviction Over Suspension

When ambiguity arises due to partial failure, systems must choose between two responses: suspending action until agreement is restored, or reducing the set of participants until ambiguity is eliminated.

Principle 3 (Eviction over suspension). When coordination fails, a storage system should resolve ambiguity by evicting or fencing uncertain participants rather than by suspending operation globally.

This principle is evident in classical storage and database systems, which prefer to isolate or remove malfunctioning components while allowing remaining participants to continue operating against the same authoritative data.

Eviction preserves service continuity and limits failure propagation. Suspension, by contrast, transforms uncertainty into system-wide unavailability.

10.4 Truth Preservation vs. Truth Reconstruction

The principles above imply a clear distinction between truth preservation and truth reconstruction.

Truth-preserving systems maintain authoritative state through localized control and recover by restoring access. Truth-reconstructing systems recompute authoritative state through coordination after failures.

While reconstruction may be acceptable for caches, logs, or derived data, it is inappropriate for storage substrates that underpin filesystems, databases, or virtual machine disks. In these contexts, reconstruction delays are operationally indistinguishable from data unavailability.

10.5 Design Implications

Adopting these principles leads to several concrete design implications:

- Coordination mechanisms should optimize performance and availability, not define correctness.
- Network partitions should be treated as access failures, not as existential threats to storage operation.
- Recovery procedures should assume the persistence of storage truth and focus on reestablishing reachability.
- Storage systems should degrade locally and predictably under failure, rather than globally and abruptly.

These implications are consistent with the behavior of established truth-preserving systems such as RAID arrays, local filesystems, and eviction-based clustered databases.

10.6 Summary

The alternative principle of storage design presented here rejects the notion that correctness must be negotiated continuously through global coordination. Instead, it asserts that storage systems must preserve localized truth and treat coordination as a secondary concern.

By separating truth from access and preferring eviction to suspension, storage systems can remain operationally coherent under realistic failure conditions. The incident analyzed in this work demonstrates the consequences of violating these principles and motivates their adoption in the design of future storage infrastructures.

11 Implications for Virtualized Infrastructure

The loss of storage truth analyzed in this work has implications that extend beyond storage systems themselves. In modern infrastructures, storage is rarely consumed directly by applications; instead, it is embedded within layers of virtualization, orchestration, and automation. When storage truth becomes fragile, its failure propagates across these layers, producing compound failure modes that cannot be resolved within any single control plane.

This section examines the implications of storage truth loss for virtualized infrastructure environments and highlights why such failures are particularly disruptive in layered cloud architectures.

11.1 Amplification Through Virtualization Layers

Virtualized infrastructures typically stack multiple abstraction layers: storage virtualization, hypervisors, virtual machines, container platforms, and application-level orchestration. Each layer assumes that lower layers provide stable and authoritative resources.

When a virtual storage system loses storage truth, this assumption is violated. Hypervisors may continue to schedule virtual machines whose disks are operationally unavailable. Container platforms may reschedule workloads that remain bound to inaccessible volumes. From the perspective of higher layers, resources appear logically present while being physically unusable.

This mirrors the phenomenon of loss of execution truth observed in layered compute orchestration. In both cases, control-plane state remains internally consistent while becoming decoupled from execution reality.

11.2 Ineffectiveness of Automation and Self-Healing

Automation mechanisms in virtualized environments presuppose that corrective actions can be executed reliably. Live migration, rescheduling, replica recreation, and autoscaling all require functioning storage access.

When storage truth is lost, these mechanisms become ineffective or even counterproductive. Migration attempts fail because disks cannot be accessed. Rescheduling creates additional load without restoring functionality. Automated recovery loops consume resources while reinforcing the illusion of control.

As a result, systems designed to improve resilience instead prolong outages by acting on misleading control-plane information.

11.3 Disaster Recovery as the Only Viable Escape

The incident analyzed in this work demonstrates that, once storage truth is lost, recovery often depends on mechanisms external to the affected storage system. In practice, this takes the form of disaster recovery failover to an independent storage instance or cluster.

While DR can restore service, it does so by abandoning the compromised storage truth and substituting an alternative one. This approach is inherently coarse, expensive, and disruptive. Its necessity in response to localized infrastructure faults indicates a structural weakness rather than an exceptional scenario.

In effect, DR becomes a compensatory mechanism for architectural fragility at the storage layer.

11.4 Observability Limits in Virtualized Storage

Observability systems in virtualized infrastructures are typically aligned with control planes: they report logical states, health checks, and availability metrics. When storage truth is lost, these signals remain internally consistent yet misleading.

Latency metrics, I/O error rates, and cluster health indicators may fluctuate, but none directly expose the ontological failure that has occurred. Engineers are left diagnosing symptoms without visibility into the underlying loss of authoritative state.

This limitation parallels the observability failure modes described in loss of execution truth: when truth itself is compromised, observation becomes structurally unreliable.

11.5 Architectural Consequences

The implications of storage truth loss for virtualized infrastructure can be summarized as follows:

- Storage failures propagate upward, disabling recovery mechanisms at higher layers.
- Layered control planes amplify rather than contain failures.
- Automation relies on assumptions that collapse when truth is no longer localized.
- Recovery shifts from fine-grained remediation to coarse-grained failover.

These consequences suggest that storage architecture is not an interchangeable component within virtualized environments, but a foundational determinant of system-wide reliability.

11.6 Summary

Virtualized infrastructures magnify the impact of storage truth loss by embedding storage into multiple dependent control planes. When storage ceases to function as a system of facts, higher layers lose their ability to reason, act, and recover.

The incident examined in this work illustrates that such failures are not isolated storage problems but systemic infrastructure failures. Designing virtualized environments for resilience therefore requires storage systems that preserve localized truth and degrade predictably under failure, rather than systems whose correctness depends on continuous global coordination.

12 Practical Recommendations

The analysis presented in this work is not intended to prescribe a universal choice of software or architecture. The selection of storage technologies necessarily depends on organizational constraints, operational maturity, and specific workload requirements. However, empirical evidence from production environments allows certain practical conclusions to be drawn.

12.1 Technology Choice as an Architectural Responsibility

The choice of a storage system is an architectural decision rather than a purely technical preference. Different systems embody different assumptions about failure semantics, coordination, and recovery. Teams must therefore be explicit about which failure modes are acceptable and which are not.

In particular, systems that embed coordination and consensus into the definition of storage correctness must be evaluated not only for their nominal performance and scalability, but for their behavior under partial and degraded failure conditions.

12.2 Observed Operational Robustness

The incident analyzed in this work demonstrates that virtual, network-centric storage systems may exhibit disproportionate failure amplification in response to localized infrastructure faults. Prolonged loss of service occurred without physical data loss, driven by degradation in auxiliary subsystems.

By contrast, storage architectures based on localized truth and distributed access have repeatedly demonstrated more predictable and contained failure behavior in comparable environments. In such architectures, access degradation does not invalidate storage truth, and recovery restores reachability to an already well-defined state.

12.3 ZFS with iSCSI as a Truth-Preserving Baseline

Based on accumulated operational experience, including the case examined in this study, configurations based on ZFS as a local storage substrate combined with iSCSI for distributed access represent a particularly robust baseline for distributed storage.

This approach exhibits several practical advantages:

- authoritative data ownership remains localized to the storage host;
- storage truth is preserved independently of network coordination state;
- network failures degrade access rather than suspend storage operation;
- recovery restores access to an unchanged factual state;
- failure impact remains localized and predictable.

Importantly, this architecture scales operationally rather than algorithmically. Increasing scale does not introduce new global coordination dependencies, but extends a pattern of localized truth with distributed reach.

12.4 Scope and Limitations

The recommendation above does not imply that ZFS with iSCSI is optimal for all use cases. Workloads that prioritize object semantics, eventual consistency, or large-scale content distribution may benefit from alternative designs.

However, for storage serving as a substrate for virtual machine disks, filesystems, databases, and other systems that depend on strict addressable state, truth-preserving architectures offer a level of operational resilience that network-centric virtual storage systems have not consistently matched in practice.

12.5 Summary

Ultimately, the choice of storage software remains the responsibility of each engineering organization. Nevertheless, real-world incidents demonstrate that architectures preserving localized storage truth degrade more gracefully and recover more reliably than those that require continuous global coordination.

At the present stage of technology and operational practice, ZFS combined with iSCSI represents a conservative but resilient approach to distributed storage at virtually any scale, particularly in environments where predictability and recoverability are more critical than theoretical elasticity.

13 Conclusions

This work introduced the notion of *storage truth* as a foundational concept for analyzing storage systems and demonstrated how its loss constitutes a qualitatively distinct failure mode. Unlike conventional performance degradation or data loss, loss of storage truth occurs when a system can no longer act authoritatively on stored data despite the physical integrity of the data itself.

Through a conceptual analysis and a real-world incident study, the paper showed that virtual, network-centric storage designs are particularly susceptible to this failure mode. By embedding coordination mechanisms into the definition of correctness, such systems transform auxiliary failures—especially network degradation—into prolonged storage outages without physical data damage.

Ceph was examined as a representative and structurally faithful realization of this design approach. Its behavior under failure illustrates not an implementation defect, but a predictable consequence of treating storage truth as a distributed, model-derived property rather than as a localized physical fact.

The paper further argued that CAP-style reasoning has no applicability as a general design principle for IT systems and is fundamentally incompatible with storage system design. CAP provides no ontology of data existence, authority, or truth, and replaces these notions with abstract coordination constraints. Applying such reasoning to storage conflates access availability with truth itself and leads to architectures that fail by denying access to data that unambiguously exists.

Comparative counterexamples—including RAID, ZFS, Oracle RAC, and log-based systems—demonstrated that distribution and fault tolerance do not inherently require global coherence over truth. Systems that localize authoritative state and employ eviction rather than suspension degrade more predictably and recover more reliably under partial failure conditions.

Finally, the paper formulated an alternative principle of storage design based on locality of truth, separation of truth and access, and eviction over suspension. Practical observations indicate that architectures adhering to these principles—such as ZFS with iSCSI for distributed access—offer superior operational resilience when storage serves as a substrate for virtualized infrastructure.

The broader implication is that storage architecture is not a neutral component within virtualized environments. Decisions about how storage truth is defined and preserved determine the failure semantics of the entire infrastructure stack. Designing reliable systems therefore requires an explicit ontological account of what exists, where authority resides, and how truth is preserved, rather than reliance on abstract coordination models.

Architectural reasoning that treats CAP-style trade-offs as fundamental reflects an insufficient level of professional qualification for the design of distributed storage systems, where correctness is defined by preservation of truth rather than coordination symmetry.

References

[1] B. Beyer, C. Jones, J. Petoff, and N. R. Murphy, Eds., *Site Reliability Engineering: How Google Runs Production Systems*. O'Reilly Media, 2016.

[2] J. Bonwick and B. Moore, "The zettabyte file system," in *Proceedings of the First USENIX Conference on File and Storage Technologies (FAST)*, 2007.

[3] E. A. Brewer, "Towards robust distributed systems," *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2000, Invited talk.

[4] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *ACM SIGACT News*, vol. 33, no. 2, pp. 51–59, 2002.

[5] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," *Proceedings of the NetDB Workshop*, 2011.

[6] A. Lakshman and P. Malik, “Cassandra: A decentralized structured storage system,” *ACM SIGOPS Operating Systems Review*, vol. 44, no. 2, pp. 35–40, 2010.

[7] L. Lamport, “Time, clocks, and the ordering of events in a distributed system,” *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[8] A. A. Nekludoff, *Coherent observational epistemology: Foundational principles, secondary principles, and axiomatic system*, Zenodo Report, 2025. DOI: 10.5281/zenodo.17632756 [Online]. Available: https://doi.org/10.5281/zenodo.17632756

[9] A. A. Nekludoff, *Loss of execution truth in layered cloud orchestration: A case study of openstack and kubernetes under cpu oversubscription*, Zenodo Report, 2025. DOI: 10.5281/zenodo.18252976 [Online]. Available: https://doi.org/10.5281/zenodo.18252976

[10] Oracle Corporation, *Oracle Real Application Clusters Administration and Deployment Guide*. Oracle, 2023.

[11] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn, “Ceph: A scalable, high-performance distributed file system,” *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pp. 307–320, 2006.

A Formal Analysis of the CAP Claim

This appendix provides a formal clarification of the logical status of the claims commonly associated with the CAP theorem. The purpose is not to dispute the correctness of any specific formal result, but to make explicit the scope, quantifiers, and ontological assumptions involved, and to show why extrapolating these results to IT systems constitutes a logical error.

A.1 Formal Preconditions of Impossibility Claims

Any claim of impossibility must be evaluated relative to a formally specified domain. Formally, an impossibility statement has the structure:

$$\forall x \in \mathcal{D},\ \neg P(x),$$

where $\mathcal{D}$ is a well-defined domain of objects and P is a precisely specified predicate.

Without explicit definition of $\mathcal{D}$ and P, such a statement has no mathematical meaning. Informal or underspecified claims cannot impose necessary constraints on classes of systems beyond the domain in which they are formally defined.

A.2 The Domain of the Gilbert–Lynch Result

Let $\mathcal{M}_{CAP}$ denote the class of abstract models considered in the formalization by Gilbert and Lynch. By construction, models in $\mathcal{M}_{CAP}$ satisfy the following premises:

1. system state is inherently distributed across replicas;
2. correctness is defined exclusively in terms of agreement among replicas;
3. communication is asynchronous and may be indefinitely delayed;
4. network partitions are treated as first-class, unavoidable conditions.

Within this domain, predicates $C(M)$, $A(M)$, and $P(M)$ are defined in terms of client-visible response properties. The formal result established in this setting can be expressed as:

$$\forall M \in \mathcal{M}_{CAP},\ \neg\big(C(M) \wedge A(M) \wedge P(M)\big).$$

This statement is mathematically correct within the domain $\mathcal{M}_{CAP}$.

A.3 The Illicit Generalization

Let $\mathcal{S}_{IT}$ denote the class of real IT systems, including storage systems, execution environments, and physical infrastructure. Systems in $\mathcal{S}_{IT}$ are characterized by the existence of authoritative local state, physical execution substrates, and failure modes that impair access or observation without negating existence.

The claim commonly attributed to CAP implicitly asserts:

$$\forall S \in \mathcal{S}_{IT},\ \neg\big(C(S) \wedge A(S) \wedge P(S)\big).$$

However, no formal argument establishes this statement. The implication

$$\forall M \in \mathcal{M}_{CAP} \Rightarrow \forall S \in \mathcal{S}_{IT}$$

is invalid, as $\mathcal{M}_{CAP}$ is a strict subset defined by assumptions that do not hold for $\mathcal{S}_{IT}$ in general.

This constitutes a quantifier shift error: a property proven for a restricted class of abstract models is illegitimately generalized to a broader class of systems with different ontological structure.

A.4 Tautological Character of the Result

Within $\mathcal{M}_{CAP}$, the impossibility result follows directly from the defining premises. If truth is defined solely through coordination and agreement, then impairment of coordination necessarily prevents correctness as defined.

Formally, this can be expressed as:

$$\forall M \in \mathcal{M}_{CAP},\ \big(\neg\text{Coordination}(M)\big) \Rightarrow \neg\text{Correctness}(M).$$

This implication is not an empirical discovery but a tautological consequence of the model's definitions. It does not describe an unavoidable property of real IT systems, but the internal consistency of an abstraction in which truth is coordination-defined.

A.5 Conclusion of the Formal Analysis

The formal result attributed to CAP is valid only within the narrowly defined domain $\mathcal{M}_{CAP}$. No formal reasoning justifies extending this result to the class $\mathcal{S}_{IT}$ of real IT systems. Consequently, CAP lacks the logical status required to function as a general theorem or theory of IT system design.

Any use of CAP as a normative constraint on storage, execution, or infrastructure architectures therefore rests not on mathematical necessity, but on an unjustified generalization of a restricted abstract model.

B Formal Counterexample: Sharding

This appendix presents sharding as a formal counterexample to the generalized CAP claim. The purpose is not to advocate sharding as a universal solution, but to demonstrate that the existence of partitions does not, in itself, imply an unavoidable trade-off between consistency and availability when truth is localized rather than consensus-defined.

B.1 Definition of the Sharded Model

Let a system S consist of a finite set of shards

$$\mathcal{H} = \{h_1, h_2, \ldots, h_n\},$$

where each shard h_i maintains authoritative ownership over a disjoint subset of the global data domain:

$$D = \bigsqcup_{i=1}^{n} D_i, \quad D_i \cap D_j = \varnothing \ \text{ for } i \neq j.$$

For each shard h_i, all operations on elements of D_i are processed locally. No shard requires agreement with any other shard to determine correctness of operations within its domain.

B.2 Local Correctness and Availability

Define correctness for shard h_i as:

$$C(h_i) := \text{all operations on } D_i \text{ are linearizable locally.}$$

Define availability for shard h_i as:

$$A(h_i) := \text{all requests targeting } D_i \text{ eventually receive a response.}$$

Both properties are evaluated with respect to the local execution context of h_i and do not depend on the state of communication with other shards.

B.3 Partition as a Structural Property

Consider a partition event P that disrupts communication between shards:

$$P : h_i \not\leftrightarrow h_j \quad \text{for some } i \neq j.$$

Under this condition, each shard h_i remains fully capable of processing operations on D_i. Neither correctness nor availability within D_i is affected by the absence of communication with other shards.

Formally, for all shards h_i:

$$P \Rightarrow \big(C(h_i) \wedge A(h_i)\big).$$

B.4 System-Level Properties

Define system-level correctness and availability as the conjunction of local properties:

$$C(S) := \bigwedge_{i=1}^{n} C(h_i), \quad A(S) := \bigwedge_{i=1}^{n} A(h_i).$$

Then, under partition P, the system satisfies:

$$P \Rightarrow \big(C(S) \wedge A(S)\big).$$

This directly contradicts the generalized CAP claim when interpreted as a statement about real IT systems:

$$P \Rightarrow \neg\big(C(S) \wedge A(S)\big).$$

B.5 Interpretation

The existence of the sharded model demonstrates that partition alone does not necessitate a loss of correctness or availability. What CAP-style reasoning treats as an unavoidable trade-off arises only when correctness is defined globally through coordination rather than locally through ownership.

Sharding constitutes a constructive counterexample: it provides a concrete system model in which partition is present, yet consistency and availability are preserved relative to localized truth.

B.6 Conclusion of the Counterexample

The sharded model falsifies any universal claim of the form:

$$\forall S \in \mathcal{S}_{IT},\ P(S) \Rightarrow \neg\big(C(S) \wedge A(S)\big).$$

Therefore, CAP has no status as a general impossibility theorem for IT systems. Its conclusions are artifacts of architectures that embed global coordination into correctness. Systems that localize truth by construction, such as sharded designs, fall outside this abstraction and routinely achieve consistency, availability, and partition tolerance simultaneously.

C Eviction as a Formal Resolution Strategy

This appendix formalizes eviction as a resolution strategy for ambiguity under partial failure. The goal is to show that eviction constitutes a logically sound and constructive alternative to coordination-based suspension, preserving correctness and availability relative to localized truth.

C.1 Ambiguity Under Partial Failure

Let a system S consist of a finite set of participants

$$\mathcal{N} = \{n_1, n_2, \ldots, n_k\},$$

each capable of performing operations on a shared or partially shared state.

Define an ambiguity condition $\mathcal{A}$ as a situation in which, due to communication degradation or observation loss, the system cannot determine whether multiple participants are concurrently acting on overlapping state.

In coordination-centric designs, ambiguity $\mathcal{A}$ is treated as a blocking condition, leading to suspension of operations until agreement is restored.

C.2 Eviction Operator

Define an eviction operator

$$\mathcal{E} : \mathcal{N} \to \mathcal{N}', \quad \mathcal{N}' \subset \mathcal{N},$$

which removes one or more participants from active operation.

Eviction is applied such that the remaining set $\mathcal{N}'$ satisfies the property:

$$\neg\mathcal{A}(\mathcal{N}'),$$

i.e., ambiguity is eliminated by construction.

Importantly, eviction reduces the set of actors rather than attempting to resolve ambiguity through additional coordination.

C.3 Correctness Preservation

Assume that authoritative state ownership is localized to specific participants. Correctness for the system S is defined with respect to the active set $\mathcal{N}'$ as:

$$C(S, \mathcal{N}') := \bigwedge_{n_i \in \mathcal{N}'} C(n_i),$$

where $C(n_i)$ denotes local correctness of participant n_i relative to its owned state.

Since eviction does not alter state ownership or state content, correctness is preserved for all non-evicted participants:

$$\mathcal{E}(\mathcal{N}) \Rightarrow C(S, \mathcal{N}').$$

C.4 Availability Preservation

Define availability relative to the active participant set as:

$$A(S, \mathcal{N}') := \forall \text{ requests } r \text{ targeting state owned by } \mathcal{N}',\ r \text{ eventually receives a response.}$$

Eviction preserves availability for requests whose target state remains within $\mathcal{N}'$. Requests targeting evicted participants may fail or be rejected, but this constitutes a localized loss of access rather than a global suspension of service.

Thus:

$$\mathcal{E}(\mathcal{N}) \Rightarrow A(S, \mathcal{N}'),$$

for the remaining operational domain.

C.5 Comparison with Coordination-Based Suspension

Let $\mathcal{S}$ denote a suspension strategy that halts all operations under ambiguity $\mathcal{A}$:

$$\mathcal{A} \Rightarrow \neg A(S).$$

By contrast, eviction yields:

$$\mathcal{A} \Rightarrow \exists \mathcal{N}' \subset \mathcal{N} \text{ such that } C(S, \mathcal{N}') \wedge A(S, \mathcal{N}').$$

Therefore, eviction strictly dominates suspension with respect to availability, while preserving correctness relative to localized truth.

C.6 Reintegration

Evicted participants may be reintegrated once ambiguity is resolved. Reintegration is performed as a recovery operation and does not affect the correctness of the system during the eviction period.

This further distinguishes eviction from coordination-based strategies, which require global agreement before progress can resume.

C.7 Conclusion of the Resolution Strategy

Eviction provides a formal mechanism for resolving ambiguity without global coordination. By reducing the active participant set until ambiguity is eliminated, systems preserve correctness and availability for the remaining domain.

This strategy underlies the behavior of truth-preserving systems such as eviction-based databases and classical storage architectures. Its formal properties demonstrate that ambiguity does not necessitate global suspension, and that coordination-free progress is possible when truth is localized by design.

D Suspension as an Anti-Pattern

This appendix formalizes suspension as an architectural anti-pattern. The goal is to show that global suspension under ambiguity is neither a necessary nor a sound resolution strategy, and that its adoption introduces strictly worse system properties compared to eviction-based approaches.

D.1 Definition of Suspension

Let a system S consist of a set of participants $\mathcal{N}$ operating on state D. Define an ambiguity condition $\mathcal{A}$ as in Appendix C.

A suspension strategy Σ is defined as:

$$\Sigma(\mathcal{A}) \Rightarrow \forall n_i \in \mathcal{N},\ \text{halt}(n_i),$$

i.e., upon detection of ambiguity, all participants are prevented from executing operations until coordination is restored.

D.2 Effect on Availability

Define availability as the ability of the system to respond to requests within a finite time. Under suspension:

$$\mathcal{A} \Rightarrow \neg A(S).$$

Suspension therefore maps a localized ambiguity into a global loss of availability, regardless of whether authoritative state remains intact and accessible.

D.3 Effect on Correctness

Suspension does not improve correctness. Since no state transformation occurs during suspension, correctness properties are neither restored nor strengthened. Formally:

$$\Sigma(\mathcal{A}) \Rightarrow \text{state}(S) = \text{state}(S).$$

Thus, suspension provides no corrective action; it merely prevents progress.

D.4 Dominance by Eviction

From Appendix C, eviction yields:

$$\mathcal{A} \Rightarrow \exists \mathcal{N}' \subset \mathcal{N} \text{ such that } C(S, \mathcal{N}') \wedge A(S, \mathcal{N}').$$

Comparing strategies:

$$\Sigma(\mathcal{A}) \Rightarrow \neg A(S),$$

$$\mathcal{E}(\mathcal{A}) \Rightarrow A(S, \mathcal{N}').$$

Eviction strictly dominates suspension with respect to availability while preserving correctness relative to the remaining operational domain.

D.5 Formal Anti-Pattern Characterization

A strategy constitutes an architectural anti-pattern if:

1. it transforms localized uncertainty into global failure;
2. it reduces system availability without improving correctness;
3. a strictly dominating alternative exists.

Suspension satisfies all three conditions and therefore qualifies as a formal anti-pattern.

D.6 Conclusion of the Anti-Pattern Analysis

Suspension under ambiguity is not a necessary consequence of distributed operation but an avoidable design choice. Its persistence in modern systems results from treating coordination failure as an ontological rupture rather than as an access impairment. Formal analysis shows that suspension is inferior to eviction-based strategies and should not be used as a default resolution mechanism.

E Mapping Eviction to Real Systems

This appendix maps the formal eviction strategy introduced in Appendix C to established real-world systems. The purpose is to demonstrate that eviction is not a theoretical construct but a widely deployed and empirically validated resolution mechanism.

E.1 Oracle RAC and Node Eviction

In Oracle Real Application Clusters (RAC), ambiguity regarding node liveness or state ownership is resolved through node eviction. When communication failures or split-brain conditions are detected, one or more nodes are forcibly removed from the cluster.

Formally, RAC implements an eviction operator:

$$\mathcal{E}_{RAC} : \mathcal{N} \to \mathcal{N}',$$

where $\mathcal{N}'$ retains exclusive ownership of shared resources. Progress continues within $\mathcal{N}'$ without requiring global agreement across all nodes.

E.2 ZFS, Storage Controllers, and Ownership

In classical storage systems and ZFS-based architectures, correctness is grounded in physical ownership of disks and pools. When ambiguity arises (e.g., dual attachment or controller failure), one controller is fenced or powered off.

This corresponds to eviction at the hardware or firmware level. The remaining controller continues operation without coordination with the evicted component, preserving both correctness and availability for the surviving domain.

E.3 Failover Controllers and SAN Architectures

Enterprise SAN systems resolve ambiguity through controller failover and fencing. Rather than suspending all I/O during uncertainty, systems remove one execution path and continue operating with the remaining authoritative controller.

Formally, these systems implement eviction implicitly, eliminating ambiguity by reducing the active actor set until ownership is unambiguous.

E.4 Contrast with Coordination-Centric Systems

In coordination-centric systems, ambiguity leads to suspension until consensus is re-established. In eviction-based systems, ambiguity leads to exclusion of participants until correctness can be guaranteed locally.

This distinction explains the empirical resilience of classical storage and RAC systems compared to coordination-heavy virtualized storage platforms.

E.5 Conclusion of the Mapping

Eviction is not an exotic or experimental strategy. It is the foundational mechanism underlying decades of resilient system design in databases, storage controllers, and clustered execution environments.

The formal properties of eviction align with empirical practice: ambiguity is resolved by exclusion rather than suspension, truth is localized rather than negotiated, and availability is preserved for the remaining operational domain.

F Formal Taxonomy of Failure Types

This appendix introduces a formal taxonomy of failure types in IT systems. The objective is to separate conceptually distinct classes of failures that are frequently conflated in informal reasoning, particularly in CAP-style analyses. By distinguishing failures by their ontological effect, the taxonomy clarifies which events affect truth, which affect access or observation, and which affect coordination without negating correctness.

F.1 Preliminaries

Let a system S operate on state D with a defined notion of correctness $C(S)$ and availability $A(S)$. Let $\mathcal{N}$ denote the set of active participants, and let communication relations between participants be represented by a graph G.

Failures are classified by their effect on:

1. existence of authoritative state;
2. ability to observe or access state;
3. ability to coordinate among participants;
4. correctness of state transformations.

F.2 Partition (Communication Failure)

A *partition* is defined as a failure of communication between subsets of participants:

$$P := \exists \mathcal{N}_1, \mathcal{N}_2 \subset \mathcal{N},\ \mathcal{N}_1 \cap \mathcal{N}_2 = \varnothing\ \wedge\ \neg\text{Comm}(\mathcal{N}_1, \mathcal{N}_2).$$

Partition affects the communication graph G but does not alter state D. Formally:

$$P \Rightarrow \text{state}(S) = \text{state}(S).$$

Partition is therefore an access or observation failure, not a truth failure.

F.3 Ambiguity (Ownership Uncertainty)

An *ambiguity* failure arises when the system cannot determine whether multiple participants are concurrently acting on overlapping authoritative state:

$$\mathcal{A} := \exists n_i, n_j \in \mathcal{N},\ i \neq j\ \wedge\ \text{Overlap}(n_i, n_j)\ \wedge\ \neg\text{Observable}(n_i, n_j).$$

Ambiguity concerns uncertainty about ownership or authority, not corruption of state. It may occur with or without partition and is resolved by eviction rather than by suspension (Appendix C).

F.4 Degradation (Performance Failure)

A *degradation* is defined as a reduction in performance characteristics such as latency or throughput:

$$D_g := \text{Latency}(S) > \tau\ \vee\ \text{Throughput}(S) < \rho.$$

Degradation does not imply incorrectness or loss of availability in the logical sense. It is a quantitative failure that may precede qualitative failures but does not, by itself, negate correctness:

$$D_g \Rightarrow C(S).$$

F.5 Suspension (Progress Failure)

A *suspension* is a system-induced halt of operations in response to another failure:

$$S_p := \forall n_i \in \mathcal{N},\ \text{halt}(n_i).$$

Suspension is not a primary failure but a secondary effect caused by design choices. It transforms localized uncertainty into global unavailability:

$$S_p \Rightarrow \neg A(S).$$

As shown in Appendix D, suspension constitutes an architectural anti-pattern.

F.6 Corruption (Truth Failure)

A *corruption* failure is defined as a violation of correctness:

$$C_r := \neg C(S).$$

Corruption directly affects state D and invalidates authoritative truth. It is ontologically distinct from partition, ambiguity, or degradation, none of which necessarily imply corruption.

F.7 Loss of Storage Truth

A *loss of storage truth* occurs when authoritative state exists but the system refuses or is unable to act upon it due to coordination-centric design:

$$L_{ST} := C(S) \wedge \neg A(S).$$

This failure mode is not reducible to corruption or partition. It arises when access or coordination failures are misclassified as truth failures.

F.8 Summary of Failure Classes

The failure types introduced above are mutually distinct:

$$P \not\Rightarrow C_r, \quad D_g \not\Rightarrow C_r, \quad \mathcal{A} \not\Rightarrow C_r.$$

Only corruption constitutes a genuine truth failure. Other failure types affect access, observation, or progress and should be addressed through localized resolution strategies such as eviction rather than global suspension.

F.9 Implications of the Taxonomy

This taxonomy exposes a central error in CAP-style reasoning: the treatment of partition and ambiguity as ontological failures rather than as access or coordination impairments. By conflating these categories, CAP-style models justify suspension and loss of service in situations where truth remains intact.

A correct classification of failure types enables architectures that preserve truth, limit the scope of failure, and avoid unnecessary global outages.

Reasoning in CAP terms reflects an insufficient architectural qualification for systems where truth preservation, rather than coordination symmetry, defines correctness.