Block Vs Object storage

Alessandro Costantini

alessandro.costantini@cnaf.infn.it

Summary

- **Introduction**
  - Scientific data needs and new challenges
- POSIX Standard
- Object Storage
- CEPH
- MinIO

I/O data

- I/O is commonly used by different applications to achieve goals like:
  - Storing output from simulations for later analysis
  - loading initial conditions or datasets for processing
  - checkpointing to files that save the state of an application in case of system failure
  - Implement 'out-of-core' techniques for algorithms that process more data than can fit in system memory

The I/O Challenge

- Data access is a huge challenge
- Data stored, moved and analyzed (also in real time)
- Findable Accesible Interoperable Reusable (FAIR principles)

- Using parallelism to obtain performance
  - O(n) n=TB
  - Critical to handle parallelism in all phases
  - Adding more compute nodes increases aggregate memory bandwidth and flops/s, but not I/O

Factors which affect I/O

- I/O is interaction with data Memory <> Disk
  - simply data migration
  - very expensive operation
- How is I/O performed
  - I/O Pattern
    - Number of processes and files
    - Characteristics of file access
- Where is I/O performed
  - Characteristics of the computational system
  - Characteristics of the Operating System and the File System

Object Storage vs. Block Storage

Point of Comparison	Object storage	Block storage
Data storage	Unique, identifiable, and distinct units called objects store data in a flat-file system.	Fixed-sized blocks store portions of the data in a hierarchical system and reassemble when needed.
Metadata	Unlimited, customizable contextual information.	Limited, basic information.
Cost	More cost-effective.	More expensive.
Scalability	Unlimited scalability.	Limited scalability.
Performance	Suitable for high volumes of unstructured data. Performs best with large files.	Best for transactional data and database storage. Performs best with files small in size.
Location	A centralized or geographically dispersed system that stores data on-premise, private, hybrid, or public cloud.	A centralized system that stores data on-premise or in private cloud. Latency may become an issue if the application and the storage are geographically far apart.

Summary #2

- Introduction
- **POSIX Standard**
  - Features
  - Architectures
  - Limits
- Object Storage
- CEPH
- MinIO

POSIX Standard

- POSIX (Portable Operating System Interface for Unix)
  - set of standards that define the Application Programming Interface (API) as well as some shell and utility interfaces
  - developed primarily for *nix operating systems
  - actually any operating system can utilize the standards
- The standards emerged from a project that began in 1985
- The family of POSIX standards is formally designated as IEEE 1003 and the international standard name is ISO/IEC 9945

POSIX IO Standard

- POSIX IO (not an official name) is the portion of the standard that defines the I/O interface for POSIX compliant applications.
- Functions
  - read(), write(), open(), close(), lseek(), fwrite(), fread(), …

- open
- read
- write
- close
- lseek
- llseek
- _llseek
- lseek64
- stat
- fstat
- stat64
- chmod
- fchmod
- access
- rename
- mkdir
- getdents
- fcntl
- unlink
- fseek
- rewind
- ftell
- fgetpos
- fsetpos
- fclose
- fsync
- creat
- readdir
- opendir
- fopendir
- rewinddir
- scandir
- seekdir
- telldir
- flock
- lockf
- lseekm
- lstat
- fstatat
- fopen
- fdopen
- freopen
- remove
- chown
- fchown
- fchmodat
- fchownat
- faccessat
- utime
- futimes
- lutimes
- futimesat
- link
- linkat
- unlinkat
- symlink
- symlinkat
- rmdir
- mkdirat
- getxattr
- lgetxattr
- fgetxattr
- xetxattr
- lsetxattr
- fsetxattr
- listxattr
- llistxattr
- flistxattr
- removexattr

POSIX-oriented operating systems

- Depending upon the degree of compliance with the standards, operating systems can be classified as
  - **POSIX-certified**
    - operating systems have been certified to conform to one or more of the various POSIX standards. This means that they passed the automated conformance tests
    - AIX, Solaris, macOS (since 10.5 Leopard)
  - **POSIX-compliant**
    - while not officially certified as POSIX compatible, comply in large part
    - Android (Available through Android NDK), *BSD, OpenSolaris

POSIX-oriented operating systems

- Depending upon the degree of compliance with the standards, operating systems can be classified as
  - **POSIX for Microsoft Windows**
    - Cygwin, Microsoft POSIX subsystem Windows Subsystem for Linux
  - **POSIX for OS/2**
    - POSIX compliant environments for OS/2
  - **Compliant via compatibility feature**
    - not officially certified as POSIX compatible, but they conform in large part to the standards by implementing POSIX support via some compatibility feature
    - SymbianOS, Windows NT Kernel

POSIX IO: Metadata

- POSIX I/O prescribes a specific set of metadata that all files must possess (POSIX I/O calls)
  - **user** and **group** which owns the file
  - the **permission** that user and group has to read and modify the file
  - **attributes** such as the time the file was created and last modified
- Calls manipulate the metadata that POSIX dictates all files must possess
  - such as chmod() and stat() or other shell commands that provide command-line interfaces for these calls

POSIX IO: Metadata

- POSIX style of metadata certainly works but…
- Prescriptive
  - all files must possess metadata such as the user and group which owns the file, the permission that user and group has to read and modify the file, and attributes such as the time the file was created and last modified
    - Eg. “ls –l” in a directory with a 1M of file
- Inflexible
  - the ownership and access permissions for files are often identical within directories containing scientific data (for example, file-per-process checkpoints), but POSIX file systems must track each of these files independently.
- Not descriptive enough for many data sets
  - often resulting in README files effectively storing the richer metadata that POSIX does not provide.

POSIX IO: stateful

- POSIX I/O is stateful
  - rely on a state in time to perform an action: e.g to change the output given the determined inputs and state.
    - "**state**" is simply the condition or quality of an entity at an instant in time
  - A typical application might
    - open() a file
    - read() the data from it
    - seek() to a new position
    - write() some data
    - close() the file

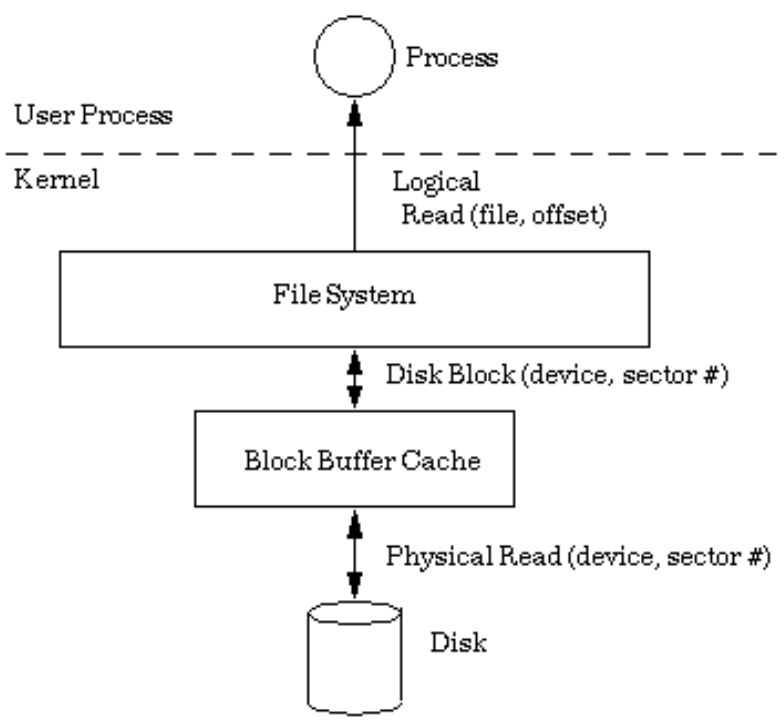
POSIX IO: file descriptor

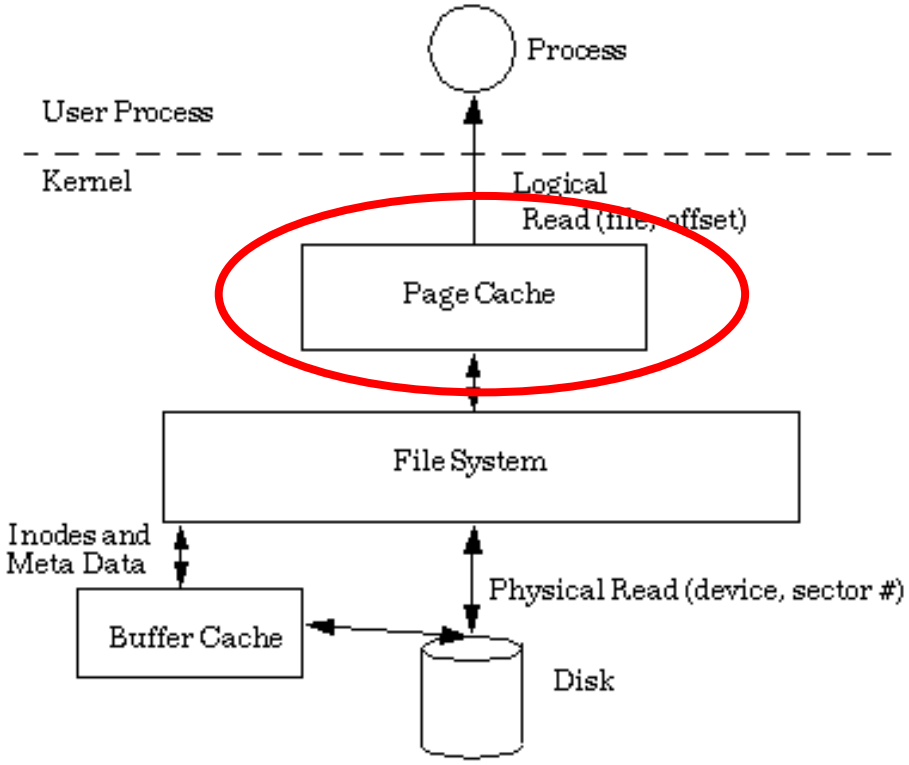
- POSIX I/O is stateful
  - reading and writing data is governed by some persistent state that is maintained by the operating system in the form of **file descriptors**.
  - File descriptors are central to this process
    - applications cannot read or write a file without first open()ing it to get a file descriptor
    - the position where the next read or write will place its data is generated by where the last read, write, or seek call ended
- **Writes must be strongly consistent:**
  - a write() is required to block application execution until the system can guarantee that any other read() call will see the data that was just written.

A “dirty” solution to stateful

- Instead of blocking the application until the data is physically stored on a slow (but nonvolatile) storage device, write()s are allowed to return control back to the application only after the data is written to a memory page.
- The operating system tracks “dirty pages” which contain data that hasn’t been flushed to disk and writes those dirty pages from memory, back into their intended files asynchronously: **Page caching**.
- The POSIX consistency guarantee is still satisfied because the OS tracks cached pages and will serve read() calls directly out of memory if a page is already there.

A “dirty” solution to stateful





Images from http://sunsite.uakom.sk

Page cache on distributed Nodes

- Unfortunately, page cache becomes much more difficult to manage on networked file systems because different client nodes who want to read and write to the same file will not share a common page cache. For example, consider the following I/O pattern:
  - Node0 writes some data to a file, and a page in Node0's page cache becomes dirty
  - Node1 reads some data before node0's dirty pages are flushed to the parallel file system storage servers
  - Node1 doesn't get node0's changes from the storage servers because node0 hasn't told anyone that it's holding dirty pages
  - The read() call from Node1 returns data that is inconsistent from that data on Node0

- **POSIX consistency was just violated**

POSIX Consistency problem

- To maintain the POSIX's strong consistency, parallel or networked file systems must **ensure that no nodes are able to read data until dirty pages have been flushed** from page cache to the back-end storage servers. This means that **parallel file systems must either:**

1. **Not use page cache at all**, which drives up the I/O latency for small writes since applications block until their data traverses the network and is driven all the way down to a hard drive
2. **Relax POSIX consistency** in a way that provides consistency for most reasonable I/O workloads (e.g., when different nodes modify non-overlapping parts of a file) but provide no consistency guarantees if two nodes try to modify the same part of the same file
3. **Implement complex locking mechanisms** to ensure that a node cannot read from a file (or part of a file) that may be being modified by another node

POSIX Consistency problem

- **NFS employs approach #2**
  - guarantees “close-to-open”consistency
  - the data in a file is only consistent between the time a file is closed and the time it is opened. If two nodes open the same file at the same time, it is possible for them to hold dirty pages, and it is then up to client applications to ensure that no two nodes are attempting to modify the same page.

POSIX Consistency problem

- **Parallel file systems like Lustre and GPFS use approach #3**
  - implement sophisticated distributed locking mechanisms to ensure that dirty pages are always flushed before another node is allowed to perform overlapping I/O.
  - Locking mechanisms can suffer from scalability limitations when application I/O is not judiciously crafted to explicitly avoid scenarios where multiple nodes are fighting over locks for overlapping extents.

Summary #3

- Introduction
  - Scientific data needs and I/ Challenges
- POSIX Standard
- **Object Storage**
  - Features
  - Architectures
  - Applications
- CEPH
- MinIO

What is object storage?

Object storage (also known as object-based storage) is a computer data storage architecture that **manages data as objects**, as opposed to other storage architectures like file systems which manages data as a file hierarchy (Posix), and block storage (RBD) which manages data as blocks within sectors and tracks.

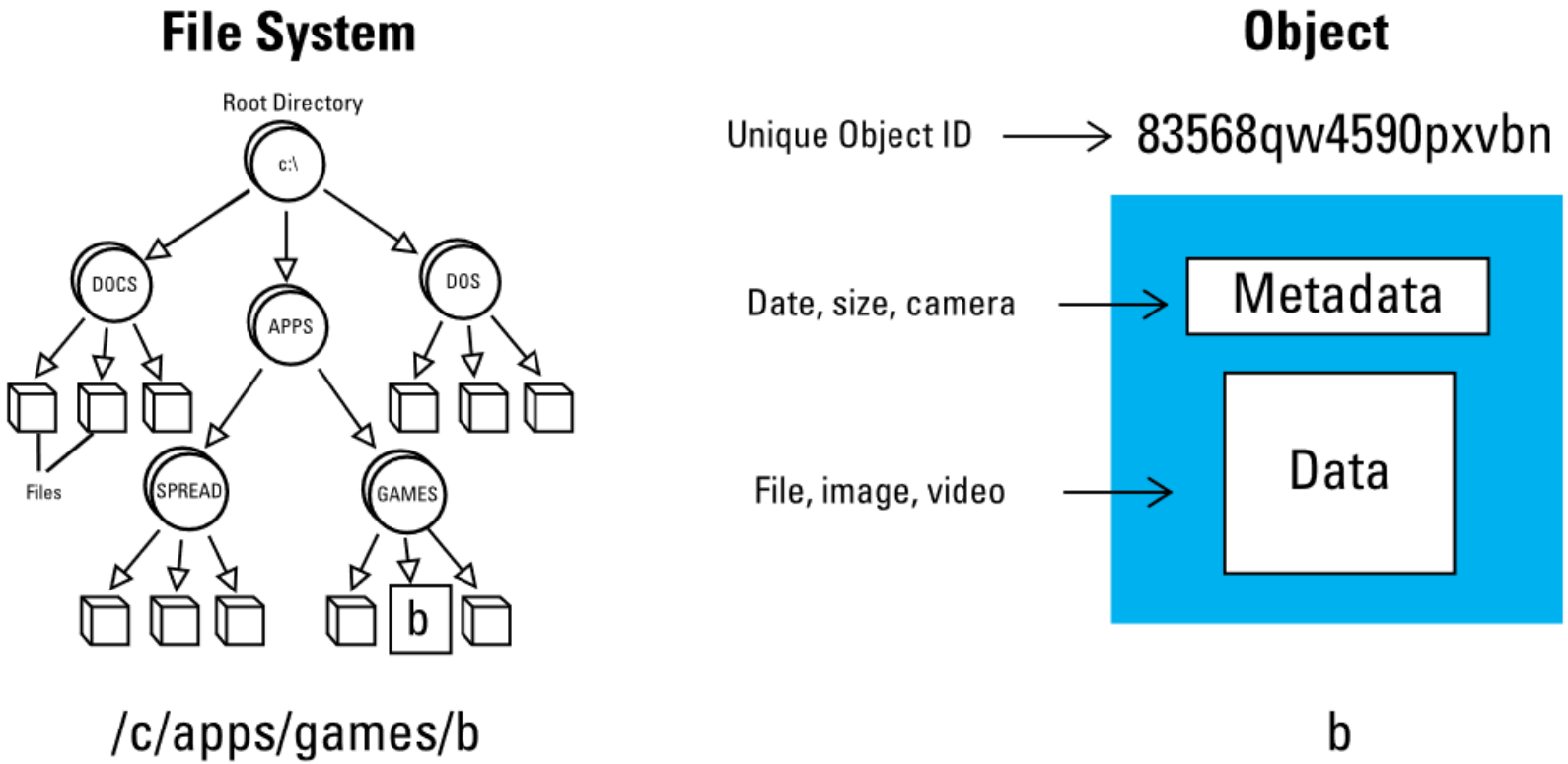
Each object typically includes the data itself, a variable amount of metadata, and a globally unique identifier.

What is Object Storage used for?

Istituto Nazionale di Fisica Nucleare

- **Storage for Unstructured Data** such as music, media files, or text documents. Any type of data that doesn't have a distinct structure to it, has metadata (ex. a song's artist, album title, etc.), and likely won't be manipulaed often is a great fit for Object Storage. Some popular products that use object storage in this category that you might recognize are Netflix and Spotify who use is to store their media files.
- **Backup and Recovery** of critical business applications and workloads. With the rise of digital products, mobile devices and the internet, consumers and enterprises expect applications to always be on and functioning. Due to the highly resilient nature and low cost of Object Storage, many businesses use it to backup their data and workloads to ensure business continuity and to prevent data loss in the event of a disaster.
- **Archived data for long term retention**. Sometimes customers specifically in the financial services industry and healthcare industry have requirements to keep data under retention or records for a certain time period (x number of years). Since this data will persist and not be manipulated frequently, object storage is a perfect cost-effective solution for this use case.
- **Cloud Native Applications** for a persistent data store. As businesses look to modernize their approach to application development in an effort to minimize the time to bring their solutions to market, they need a data store that will scale and not cause costs to sky rocket. Object Storage is a great solution for this as applications can connect directly to the object store and will allow for data to scale simply effectively as the business grows with its number of users and locations.
- **Data Lake for Analytics**. With the acceleration of the number of devices generating data (Smartphones, smart devices, IOT sensors etc.), there will be lots of data circulating around that can be processed for intelligent insights. Current storage solutions such as NAS and others are just not effective enough to support this vast growth of data being produced through these various sources. Object Storage can be a great solution for storing all types of data (structured, semi-structured, and unstructured data) that will give businesses a place to dump data before processing and analyzing in order to enable critical insights.

FS vs Object


File System
Root Directory
c:\
DOCS
APPS
DOS
Files
SPREAD
GAMES
b
/c/apps/games/b
Object
Unique Object ID
83568qw4590pxvbn
Date, size, camera
Metadata
File, image, video
Data
b

Object Metadata

- The flat organizational structure also enables object storage to provide a much richer metadata component for the object.
  - **Non-editable metadata**
    - Some metadata cannot be edited directly. This metadata is set at the time of object creation or rewrite
  - **Editable metadata**
    - **Fixed-key metadata**: Metadata whose keys are set, but for which you can specify a value.
    - **Custom metadata**: Metadata that you add by specifying both a key and a value associated with the key

Fixed-key metadata

- **Access control metadata**
  - Object Storage uses Identity and Access Management (IAM) and Access Control Lists (ACLs) to control access to objects.
- **Content Type**
  - Also known as MIME type, allows browsers to render the object properly
- **Content Disposition**
  - Content-Disposition allows to control presentation style of the content, for example determining whether an attachment should be automatically displayed or whether some form of action from the user should be required to open it

Fixed-key metadata

- **Content Encoding**
  - The Content-Encoding metadata can be used to indicate that an object is compressed
- **Content Language**
  - The Content-Language metadata indicates the lanuage(s) that the object is intended for
- **Cache Control**
  - Can control whether and for how long browser and Internet caches are allowed to cache your objects

Custom Metadata

- **Custom Metadata**
  - Custom metadata is metadata that can be added and removed.
  - Custom metadata are specified in the form of *key:value*
    - *Limited (10MB; 100 istances)*

Key Features of Object Storage

- One of the big draws of object oriented storage is its simplicity.
- There are only a few commands for object based file systems:
  - PUT (basically a "write" — PUT the object into the storage)
  - GET (basically a "read" — GET the object from the storage)
  - DELETE (delete the object)
  - HEAD (returns an object's metadata but not the data itself)

Write-once…

- Data is write-once
  - Editing an object means creating a completely new copy of it with the necessary changes
  - It is up to the user of the object store to keep track of which object IDs correspond to more meaningful information like a file name.
  - there is no need for a node to obtain a lock an object before reading its contents
  - There is no risk of another node writing to that object while its is being read
- The only reference to an object is its unique object ID
  - a simple hash of the object ID can be used to determine where an object will physically reside (which disk of which storage node)

…Read many

- Objects are comprised of
  - an object ID
  - Data
- Any metadata for an object (such as a logical file name, creation time, owner, access permissions) can be managed separately
- Objects' immutability restricts them to write-once, read-many workloads
  - Object storage cannot be used for scratch space or hot storage
  - applications are limited to data archival.

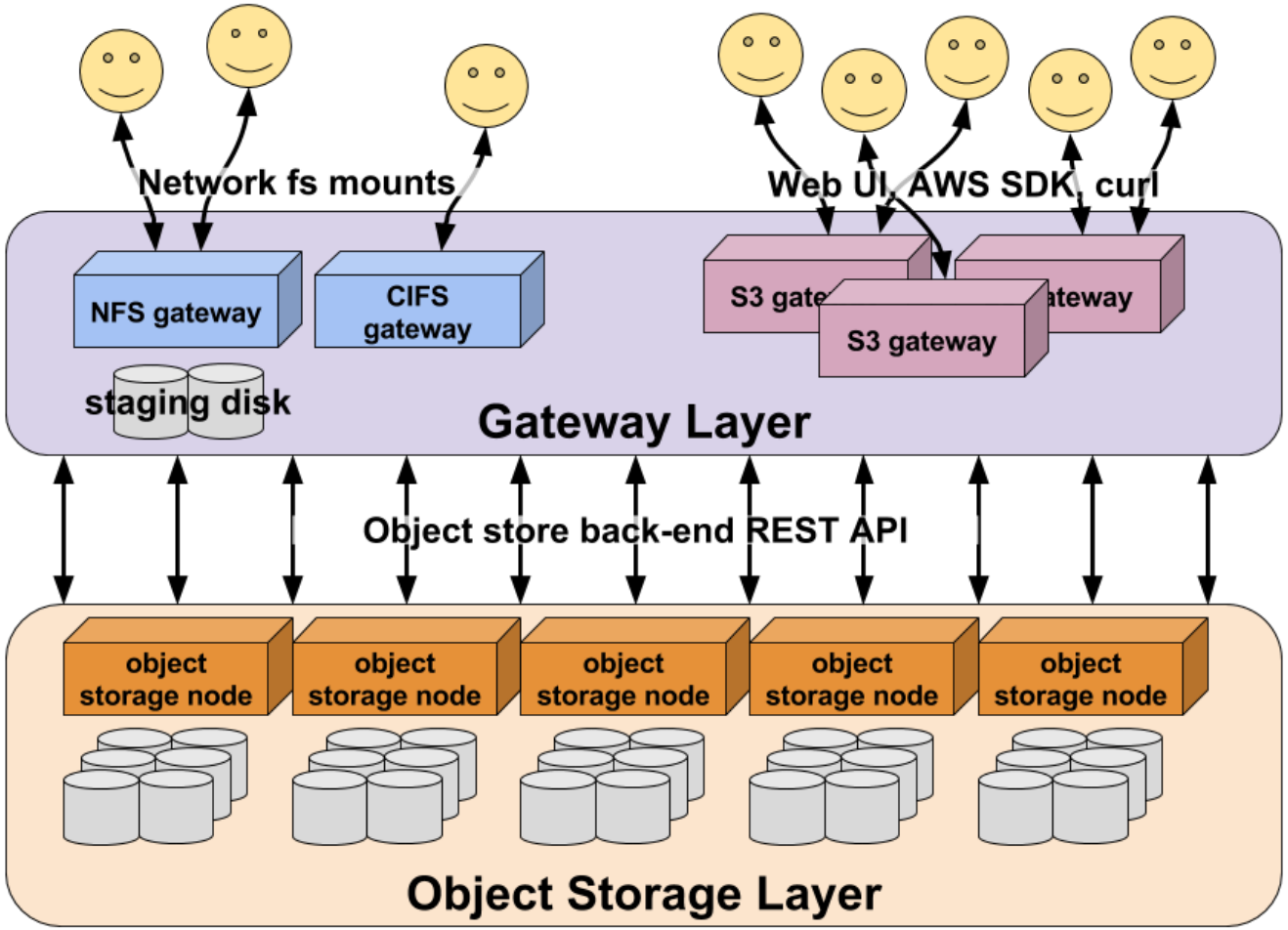
Applications

- Unique ID provides **greater scalability**, enabling an object storage system **to support faster access** to a much higher quantity of objects or files
- Object-storage systems allow retention of massive amounts of unstructured data.
- Object storage is used for purposes such as storing photos on Facebook, songs on Spotify, or files in online collaboration services, such as Dropbox.

Architecture and Structure

- Object storage systems are software defined scale-out architectures that can leverage commodity servers and storage.
  - IT planners can add additional storage nodes as their capacity demands grow.
  - Ideal for a use case where a lot of data needs to be stored for a long period.

Object Storage Cluster



Object-based file systems

- Some distributed file systems use an object-based architecture
  - metadata is stored in metadata servers
  - data is stored in object storage servers.
- Abstraction of the distinct servers to present a full file system to users and applications.
  - IBM Spectrum Scale (also known as GPFS),
  - Dell EMC Elastic Cloud Storage
  - Ceph
  - Lustre

Cloud storage

- The vast majority of cloud storage available in the market leverages an object-storage architecture. Some notable examples are
  - Amazon Web Services S3, which debuted in March 2006,
  - Google Coud Storage released on May 2010
  - Rackspace Files (whose code was donated in 2010 to Openstack project and released as OpenStack Swift)
  - MinIO Multi-Cloud Object Storage

Side-by-side comparison

	OBJECT STORAGE	BLOCK STORAGE
PERFORMANCE	Performs best for big content and high stream throughput	Strong performance with database and transactional data
GEOGRAPHY	Data can be stored across multiple regions	The greater the distance between storage and application, the higher the latency
SCALABILITY	Can scale infinitely to petabytes and beyond	Addressing requirements limit scalability
ANALYTICS	Customizable metadata allows data to be easily organized and retrieved	No metadata

Let’s quiz #1

- Large data sets. Whether you’re storing pharmaceutical or financial data, or multimedia files such as photos and videos

Object storage

Block storage

Let’s quiz #1

- Large data sets. Whether you’re storing pharmaceutical or financial data, or multimedia files such as photos and videos

Object storage

Block storage

Let’s quiz #2

- RAID Volumes, where you combine multiple disks organized through stripping or mirroring.

Object storage

Block storage

Let’s quiz #2

- RAID Volumes, where you combine multiple disks organized through stripping or mirroring.

Object storage

Block storage

Let’s quiz #3

- Storage for backup files, database dumps, and log files.

Object storage

Block storage

Let’s quiz #3

- Storage for backup files, database dumps, and log files.

Object storage

Block storage

Use case

Object storage

- Storage of **unstructured data** like music, image, and video files.
- Storage for backup files, database dumps, and log files.
- Large data sets. Whether you're storing pharmaceutical or financial data, or multimedia files such as photos and videos, storage can be used as your **big data** object store.
- Archive files in place of local tape drives.

Block storage

- **Ideal for databases**, since a DB requires consistent I/O performance and low-latency connectivity.
- Use block storage for **RAID Volumes**, where you combine multiple disks organized through stripping or mirroring.
- Any application which requires **service side processing**, like Java, PHP, and .Net will require block storage.
- Running **mission-critical** applications

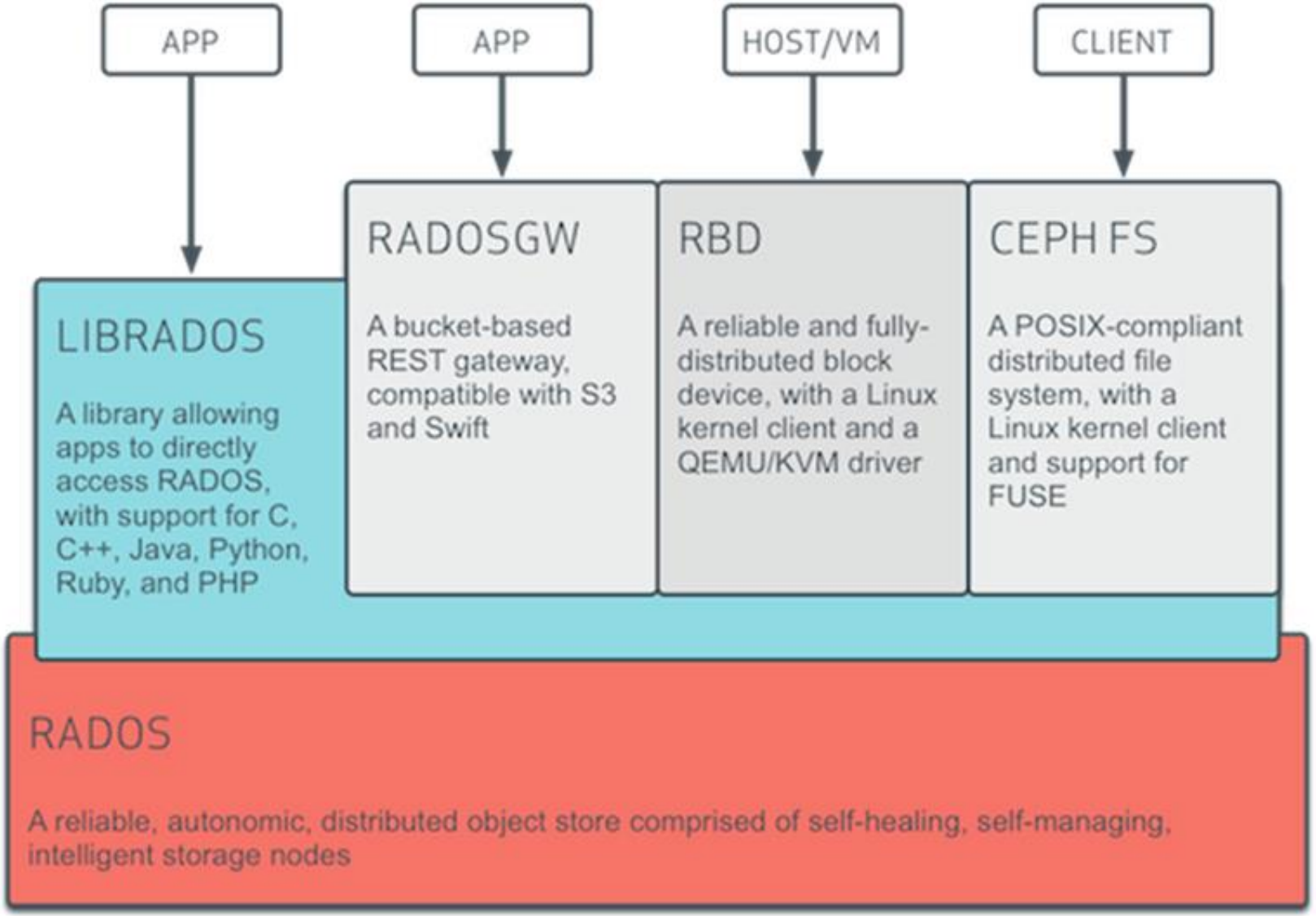
Summary #4

- Introduction
  - Scientific data needs and I/ Challenges
- POSIX Standard
- Object Storage
- **CEPH**
- MinIO

CEPH Features

- In CEPH everything is an object
- No database for object position on the cluster
- There is a “rule” to place where store data on the cluster:
  - Each node of the cluster can calculate the object position

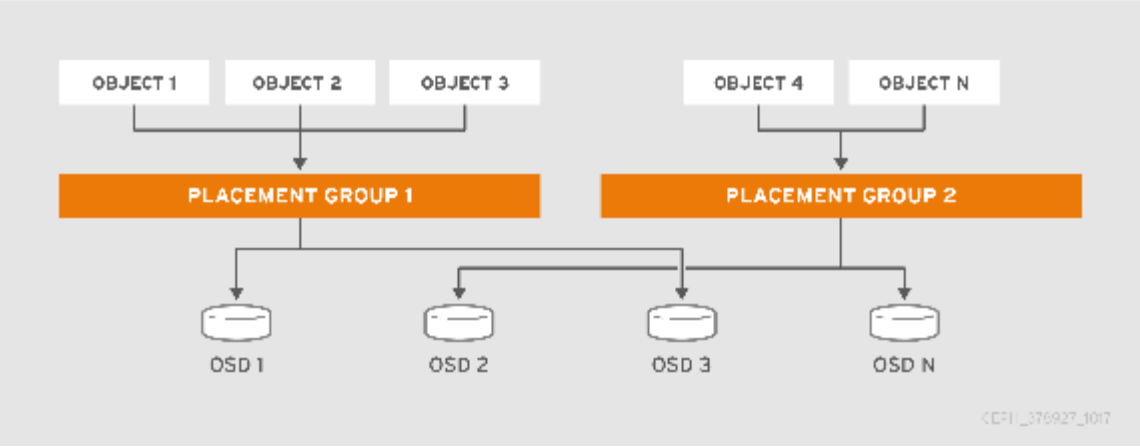
CEPH Architecture



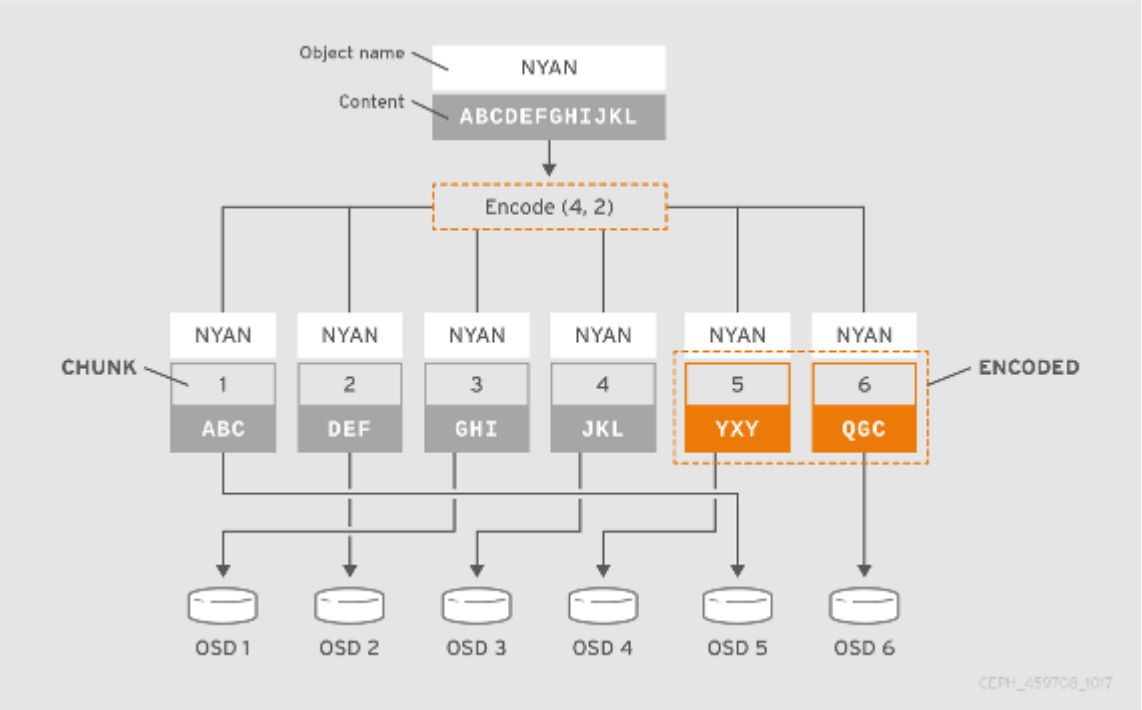
CEPH Replication strategy

Istituto Nazionale di Fisica Nucleare

Replicated



Erasure coding

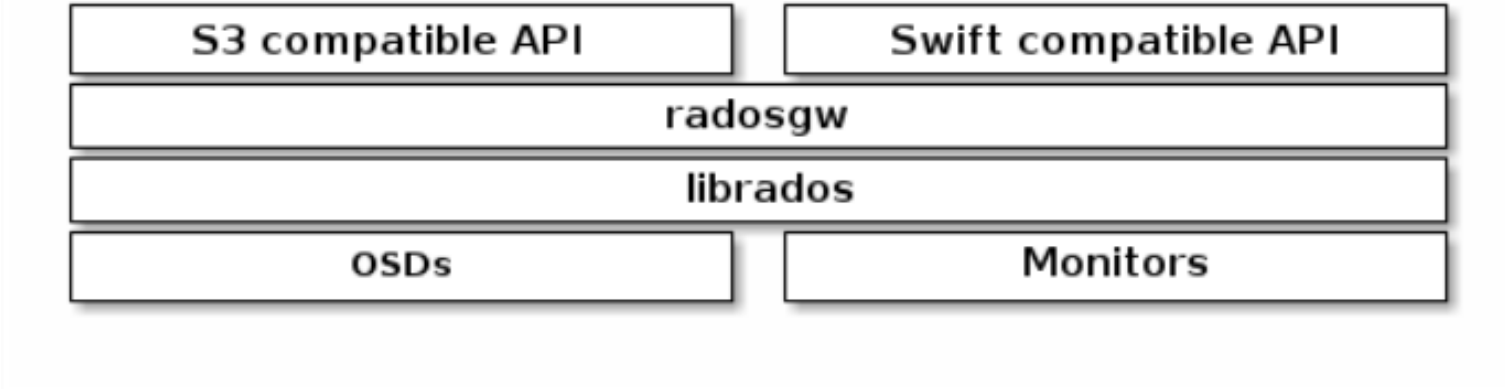


CEPH Object Gateway

Ceph Object Gateway is an object storage interface built on top of librados to provide applications with a RESTful gateway to Ceph Storage Clusters. Ceph Object Storage supports two interfaces:

1.S3-compatible: Provides object storage functionality with an interface that is compatible with a large subset of the Amazon S3 RESTful API.

2.Swift-compatible: Provides object storage functionality with an interface that is compatible with a large subset of the OpenStack Swift API.



API support

Ceph Object Gateway S3 API

Ceph supports a RESTful API that is compatible with the basic data access model of the Amazon S3 API.

Feature	Status	Remarks
List Buckets	Supported	
Delete Bucket	Supported	
Create Bucket	Supported	Different set of canned ACLs
Bucket Lifecycle	Supported	
Policy (Buckets, Objects)	Supported	ACLs & bucket policies are supported
Bucket Website	Supported	
Bucket ACLs (Get, Put)	Supported	Different set of canned ACLs
Bucket Location	Supported	
Bucket Notification	Supported	See S3 Notification Compatibility
Bucket Object Versions	Supported	
Get Bucket Info (HEAD)	Supported	
Bucket Request Payment	Supported	
Put Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Object ACLs (Get, Put)	Supported	
Get Object Info (HEAD)	Supported	
POST Object	Supported	
Copy Object	Supported	
Multipart Uploads	Supported	
Object Tagging	Supported	See Object Related Operations for Policy verbs
Bucket Tagging	Supported	
Storage Class	Supported	See Storage Classes

Ceph Object Gateway Sfift API

Ceph supports a RESTful API that is compatible with the basic data access model of the Swift API.

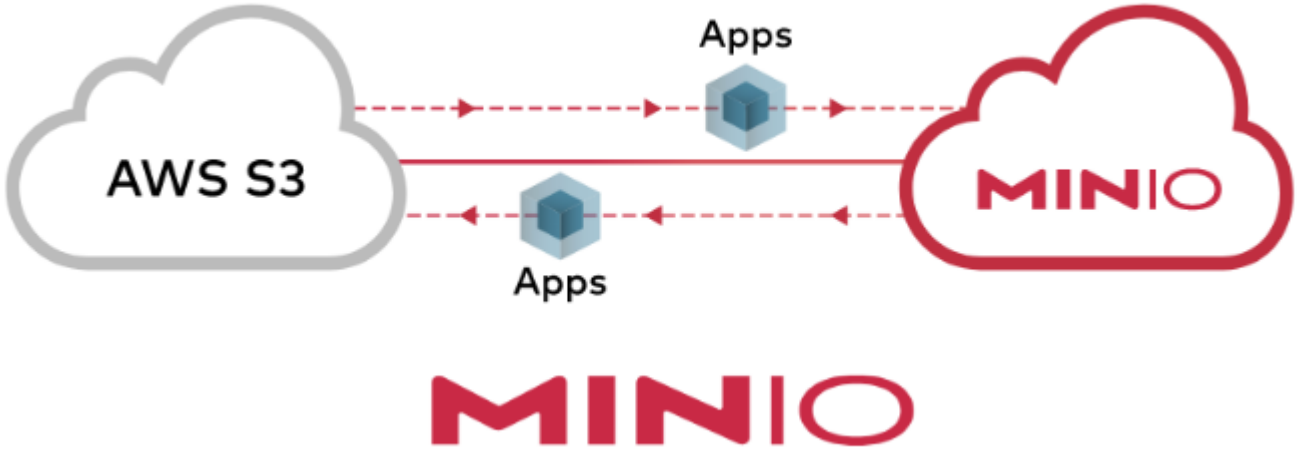
Feature	Status	Remarks
Authentication	Supported	
Get Account Metadata	Supported	
Swift ACLs	Supported	Supports a subset of Swift ACLs
List Containers	Supported	
Delete Container	Supported	
Create Container	Supported	
Get Container Metadata	Supported	
Update Container Metadata	Supported	
Delete Container Metadata	Supported	
List Objects	Supported	
Static Website	Supported	
Create Object	Supported	
Create Large Object	Supported	
Delete Object	Supported	
Get Object	Supported	
Copy Object	Supported	
Get Object Metadata	Supported	
Update Object Metadata	Supported	
Expiring Objects	Supported	
Temporary URLs	Partial Support	No support for container-level keys
Object Versioning	Partial Support	No support for `X-History-Location`
CORS	Not Supported	

Summary #5

- Introduction
  - Scientific data needs and I/ Challenges
- POSIX Standard
- Object Storage
- CEPH
- **MinIO**

MinIO Features

- MinIO is adistributed object storage server written in Go, designed for Private Cloud infrastructure
- Providing S3 storage functionality. Suited for storing unstructured data such as photos, videos, log files, backups, and container.



MINIO

How does MinIO works?

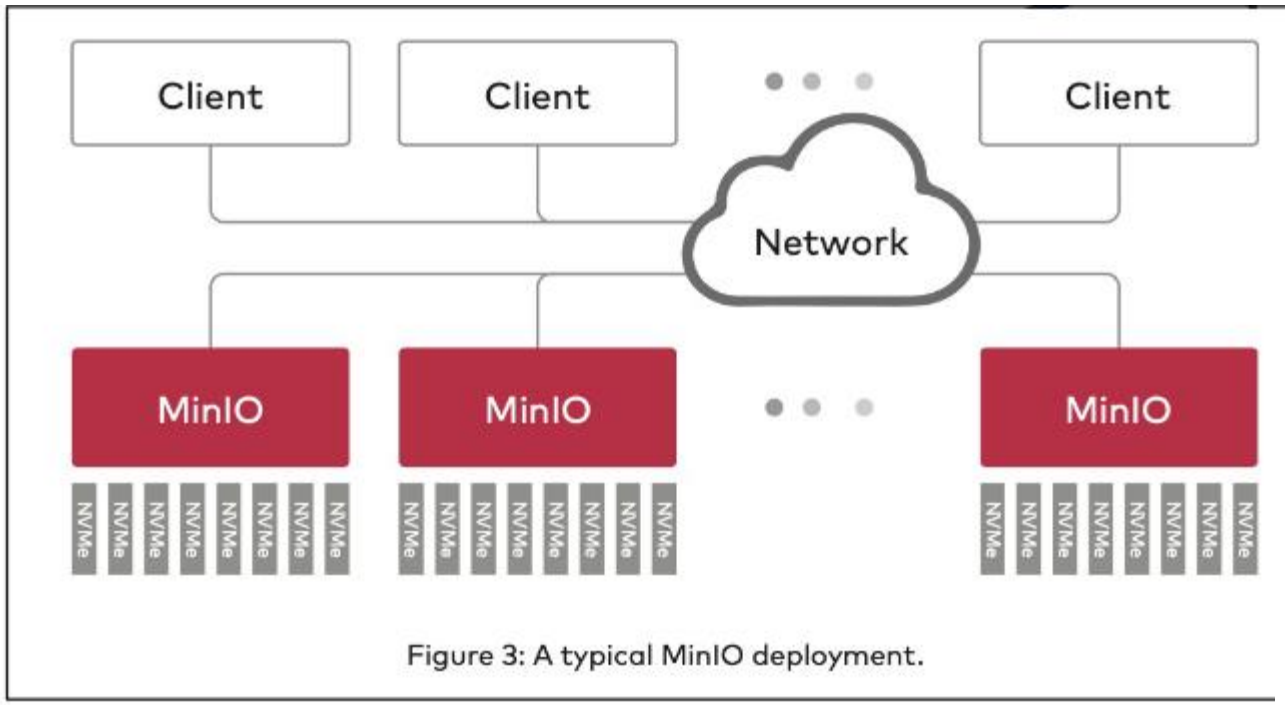


Figure 3: A typical MinIO deployment.

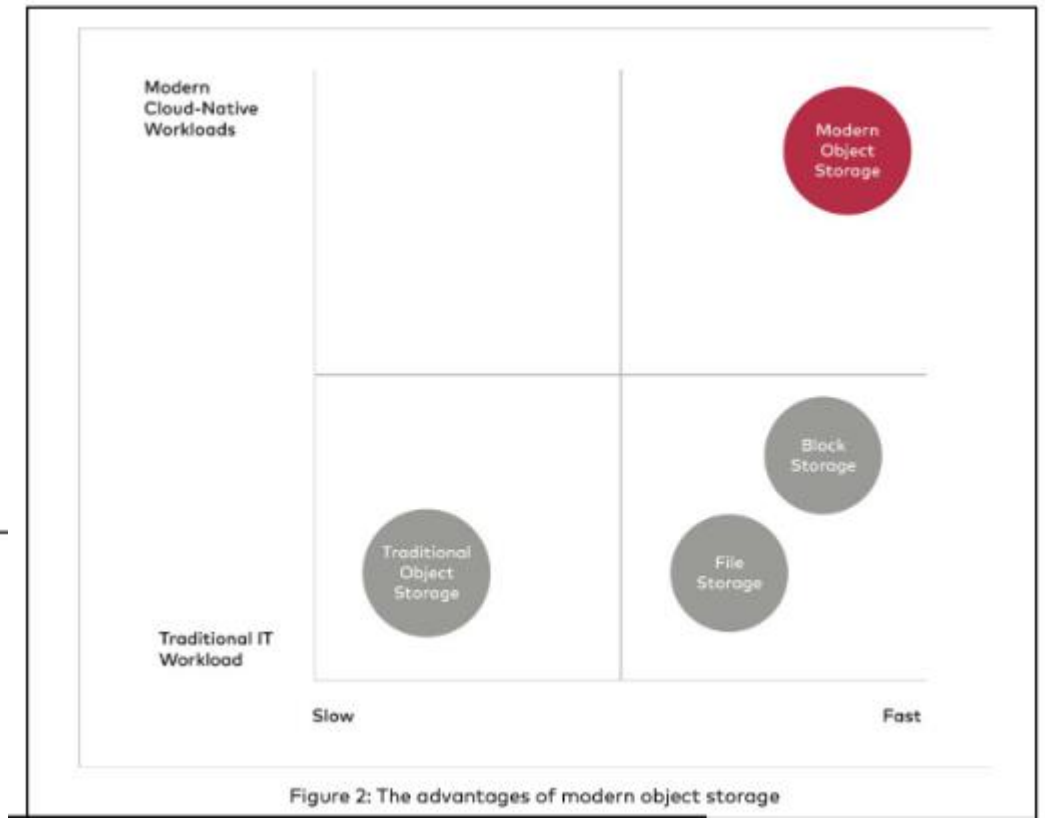
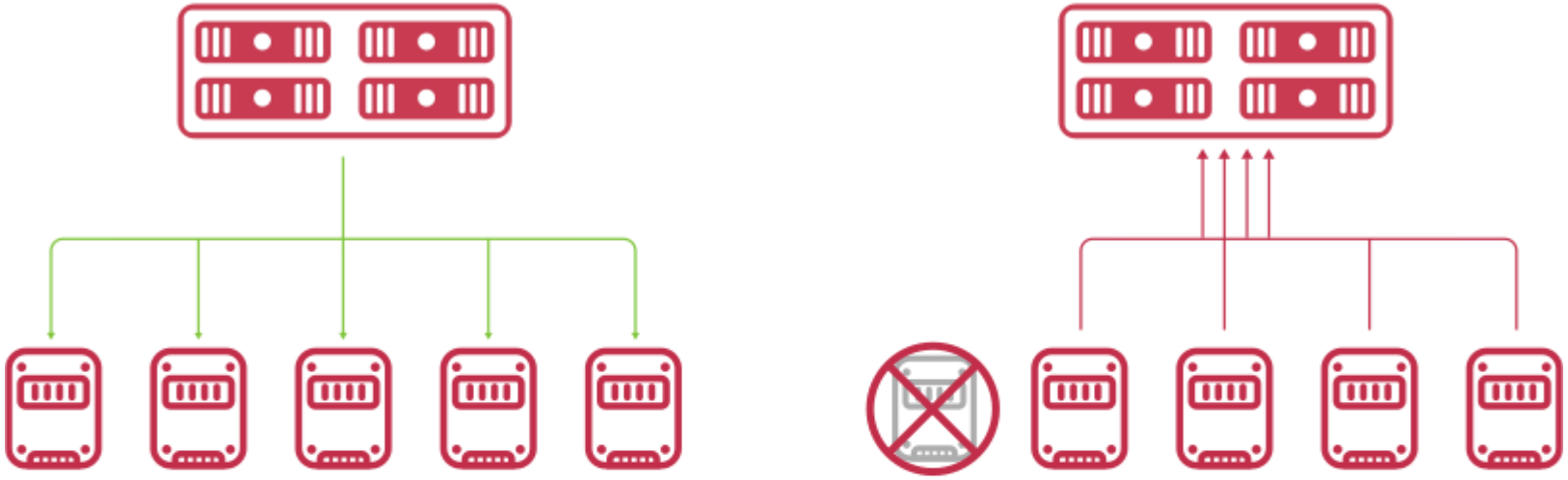


Figure 2: The advantages of modern object storage

MinIO replica strategy

ERASURE CODING (3+2)



- Green = **The original data comes into the system, is encoded into data and parity blocks and written to 5 drives.**
- Red = **When 1 drive fails, data on any other 3 drives can be decoded into the original data.**

MinIO User interface