GPFS-OpenStack Integration

Vladimir Sapunenko, INFN-CNAF

Tutorial Days di CCR,

18 dicembre 2014

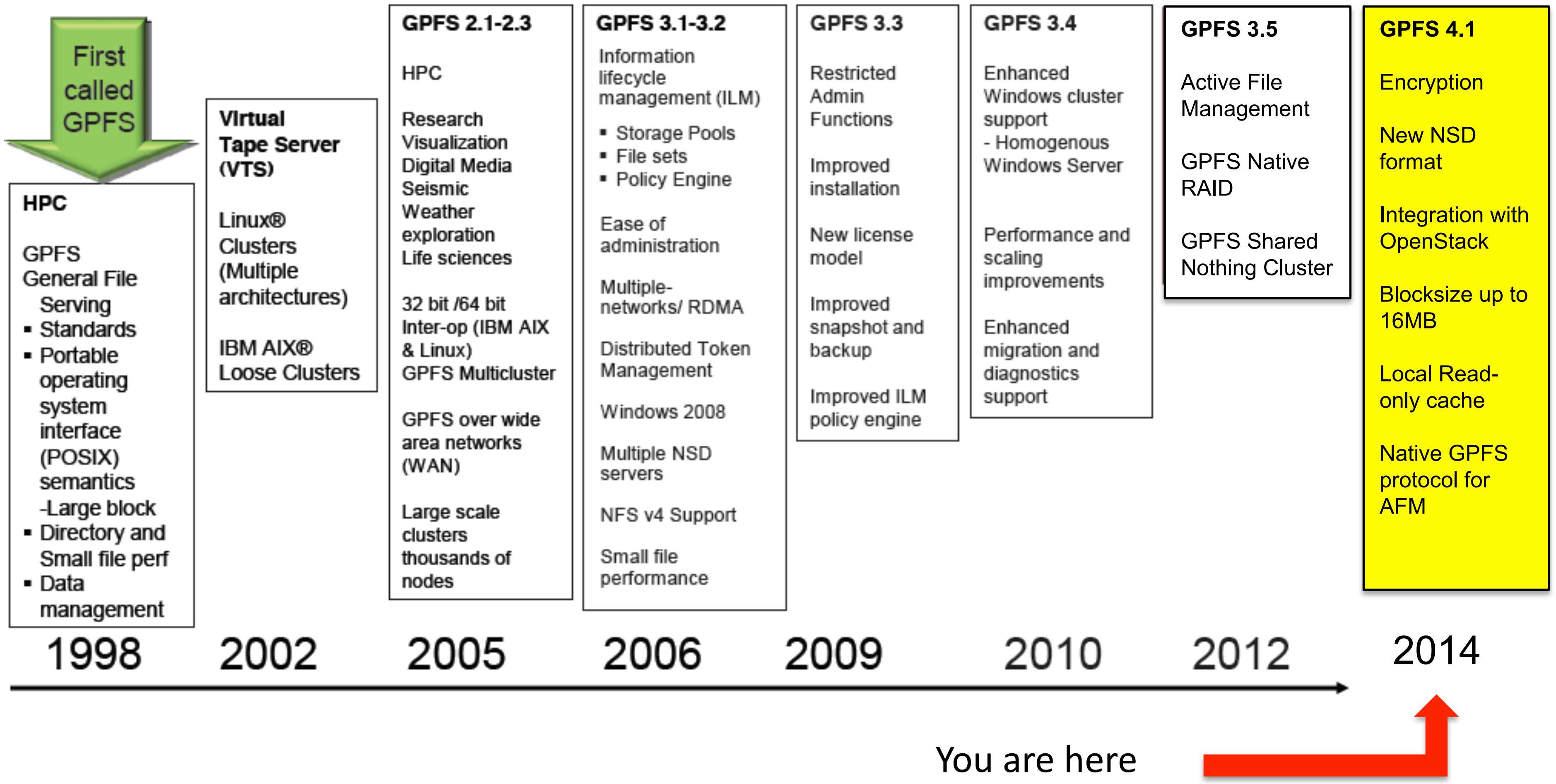
Outline

- GPFS features as they relate to cloud scenarios
- GPFS integration with OpenStack components
  - Glance
  - Cinder
  - Swift
- Demo

GPFS

- IBM General Parallel File System (GPFS) is a cluster file system that provides concurrent access to file systems from multiple nodes.
- The storage provided by these nodes can be direct attached, network attached, SAN attached, or a combination of these methods.
- GPFS provides many features beyond common data access, including data replication, policy based storage management, and space efficient file snapshot and clone operations.
- Licensed and widely used in INFN
  - Current support contract till October 2017

GPFS history and evolution



First called GPFS

HPC

GPFS
General File Serving
- Standards
- Portable operating system interface (POSIX) semantics
  -Large block
- Directory and Small file perf
- Data management

Virtual Tape Server (VTS)

Linux® Clusters (Multiple architectures)

IBM AIX® Loose Clusters

GPFS 2.1-2.3

HPC

Research
Visualization
Digital Media
Seismic
Weather exploration
Life sciences

32 bit /64 bit
Inter-op (IBM AIX & Linux)
GPFS Multicluster

GPFS over wide area networks (WAN)

Large scale clusters thousands of nodes

GPFS 3.1-3.2

Information lifecycle management (ILM)
- Storage Pools
- File sets
- Policy Engine

Ease of administration

Multiple-networks/ RDMA

Distributed Token Management

Windows 2008

Multiple NSD servers

NFS v4 Support

Small file performance

GPFS 3.3

Restricted Admin Functions

Improved installation

New license model

Improved snapshot and backup

Improved ILM policy engine

GPFS 3.4

Enhanced Windows cluster support
- Homogenous Windows Server

Performance and scaling improvements

Enhanced migration and diagnostics support

GPFS 3.5

Active File Management

GPFS Native RAID

GPFS Shared Nothing Cluster

GPFS 4.1

Encryption

New NSD format

Integration with OpenStack

Blocksize up to 16MB

Local Read-only cache

Native GPFS protocol for AFM

1998 2002 2005 2006 2009 2010 2012 2014

You are here

Conceptual understanding

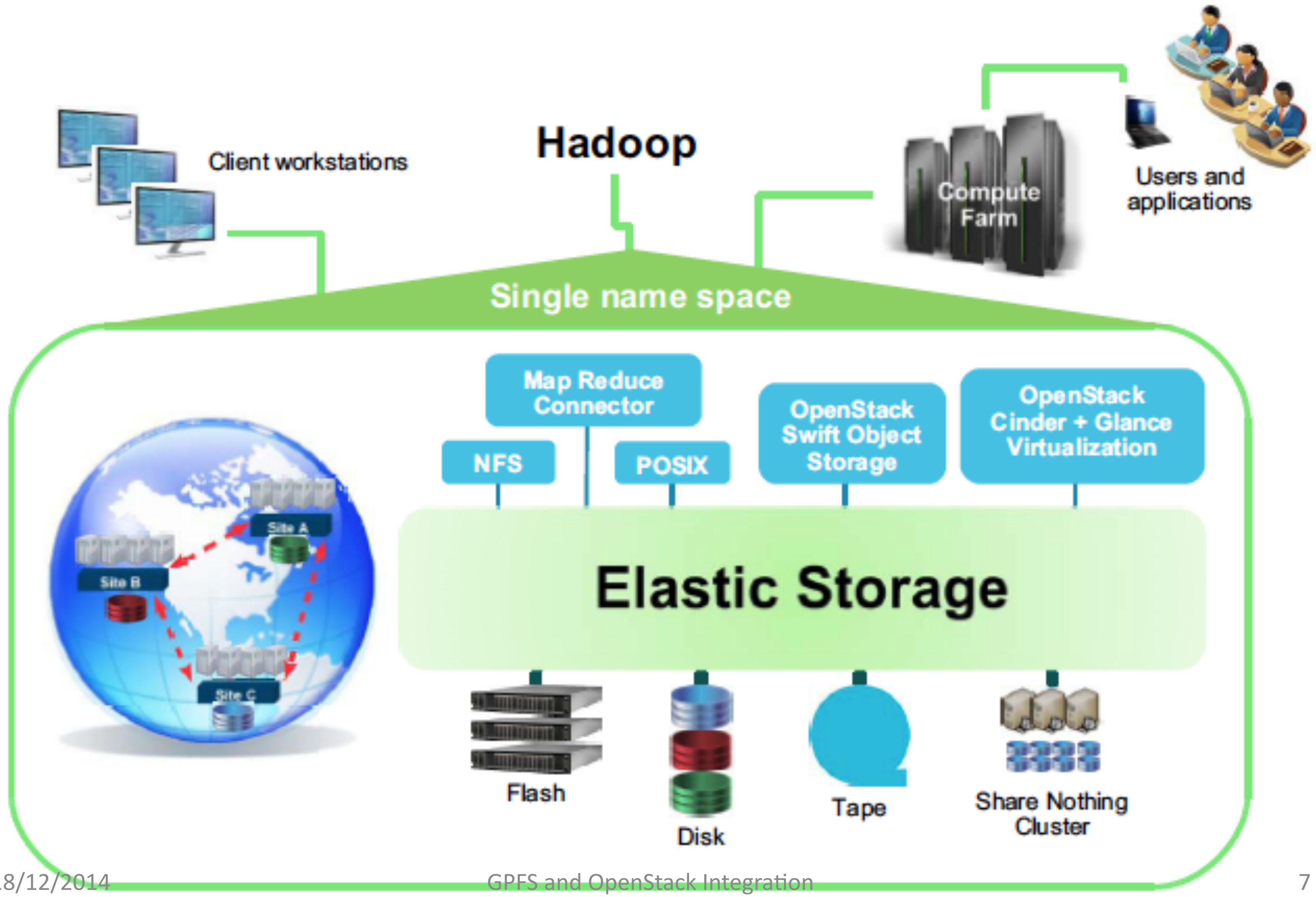
What GPFS does:

- allows a group of computers concurrent access to a common set of data
- allows shared access to file systems from Remote GPFS Clusters providing a Global namespace
- Supports Shared-disk and from v.3.5 Shared-Nothing mode
- is not just a **Clustered File System** software – it's a full featured set of File management tools

New major release: GPFS 4.1

- Code name “Elastic Storage”
  - the Next Generation of Software Defined Storage for cloud, big data and analytics
- Elastic Storage features:
  - Enhanced security
    - native encryption and secure erase, NIST SP 800-131A encryption compliance
  - Increased performance
    - Server-side Flash caches increase IO performance up to 6X
  - Improved usability
    - data migration; AFM, FPO, and backup/restore enhancements; reliability, availability and serviceability enhancements
  - integrated with IBM Tivoli Storage Manager (TSM) or IBM Linear Tape File System (LTFS),
    - Can manage the full data life cycle, delivering geometrically lower cost savings through policy driven automation and tiered storage management.

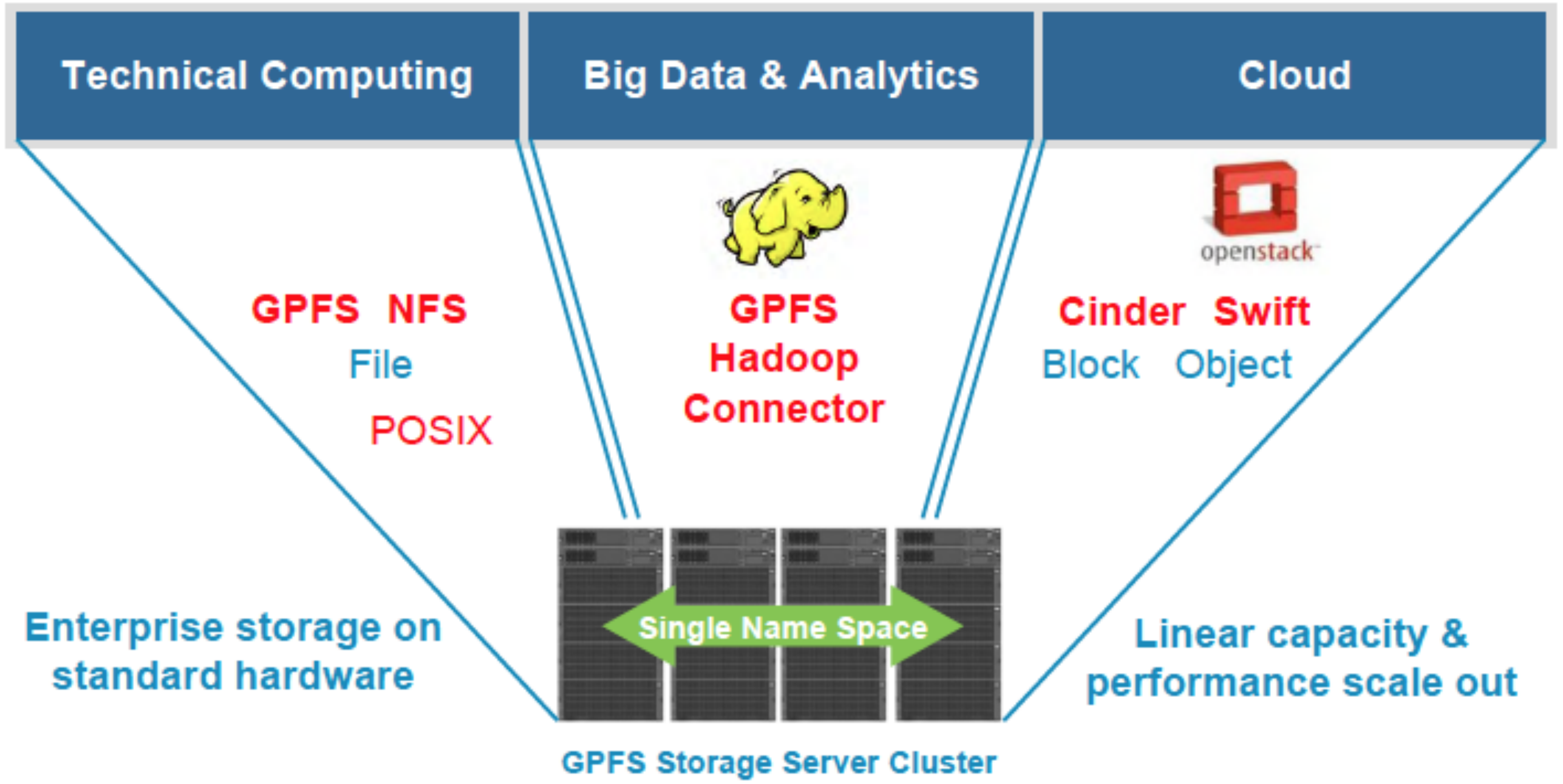
Elastic storage overview



GPFS features useful for clouds

- Storage pools
- Independent Filesets
- File cloning
- Shared Nothing Cluster - File Placement Optimizer (FPO)
  - Metablocks (blockGroupFactor)
  - Extended failure groups
  - Local read-only cache (**LROC**)
- GPFS = Software Defined Storage !

GPFS = Software Defined Storage (SDS)



Single software defined storage solution across all these application types

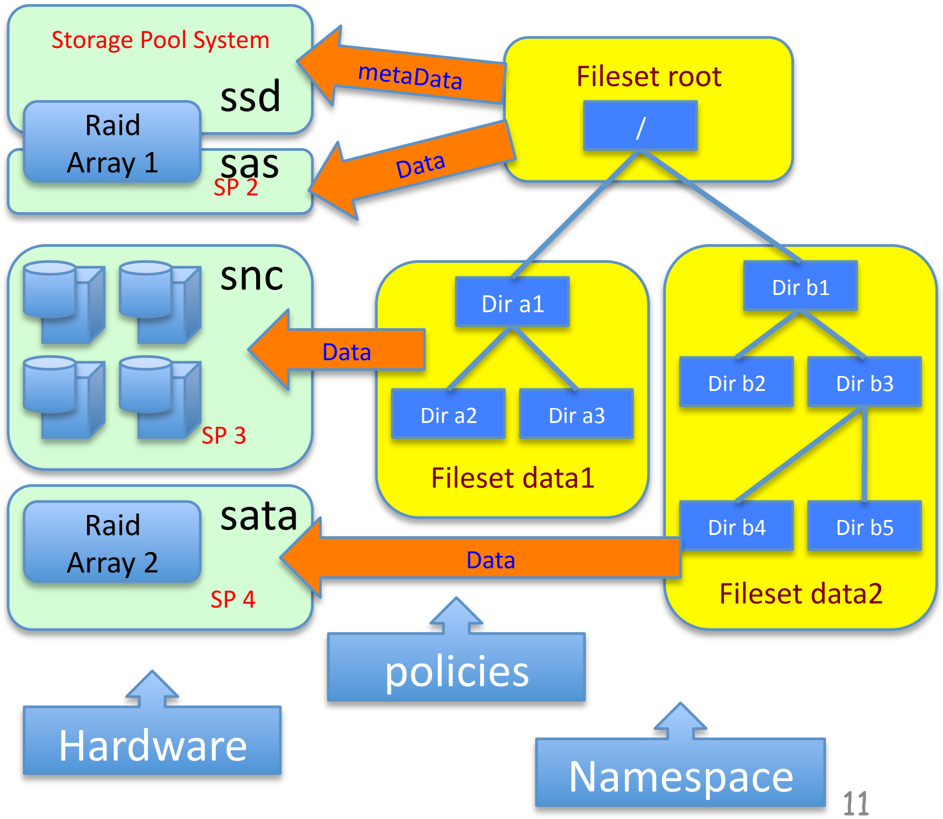
Some details about

- Storage Pools (SP)
- Independent FileSets
- Intelligent Life Circle Management (ILM)
- File clones
- Extended Failure Groups
- File Placement Optimizer (FPO)

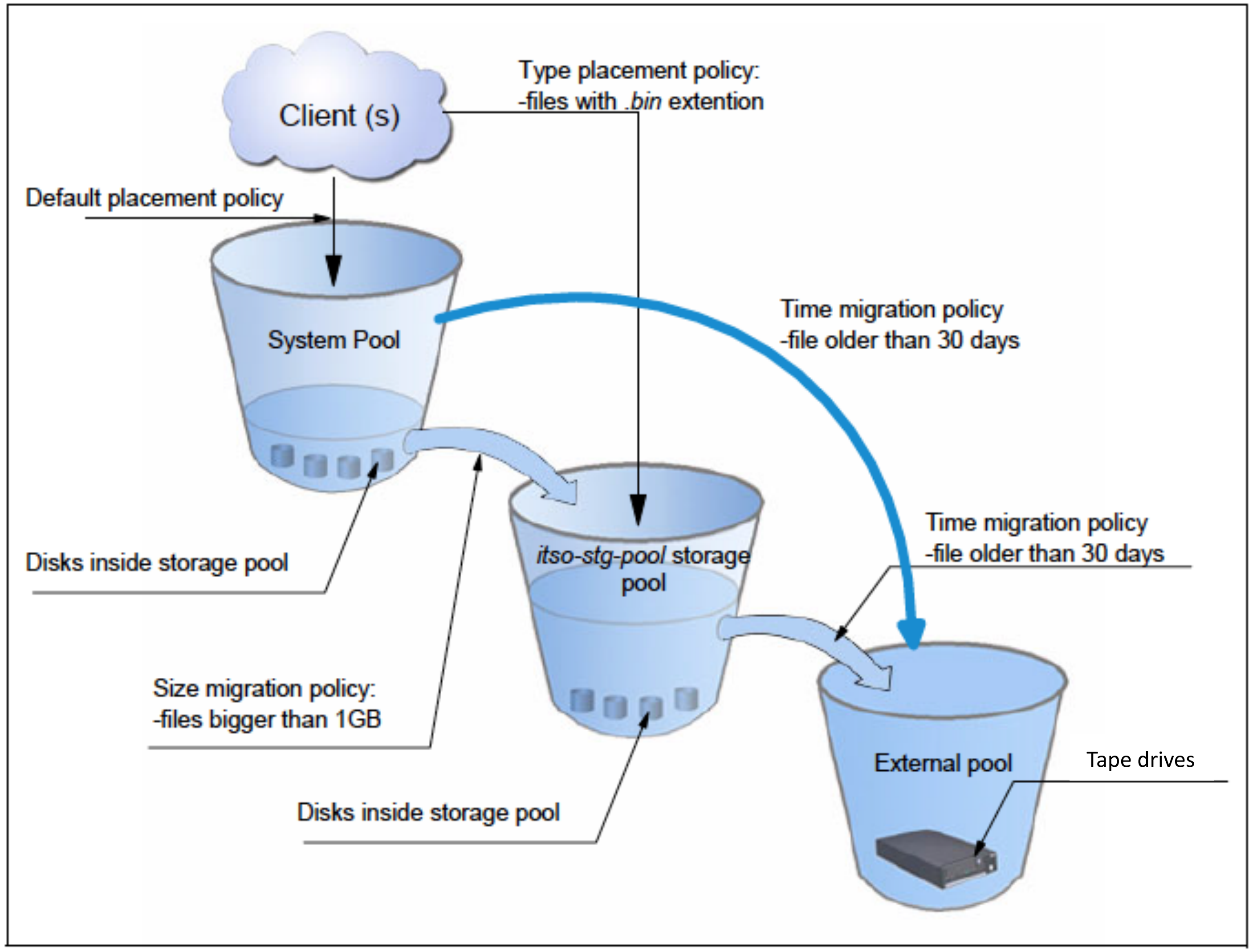
Filesets and Storage pools

- Storage pools (SP) allow the creation of disk groups within a file system (hardware partitioning)

- Filesets is a sub-tree of the file system namespace (Namespace partitioning).
  - Behave like separate file systems
  - can be used as administrative boundaries to set quotas
  - Independent Filesets
    Using separate i-node space
- Use Policy to connect SP and Filesets
  - Default policy writes everything to system SP



ILM: Intelligent Lifecycle Management

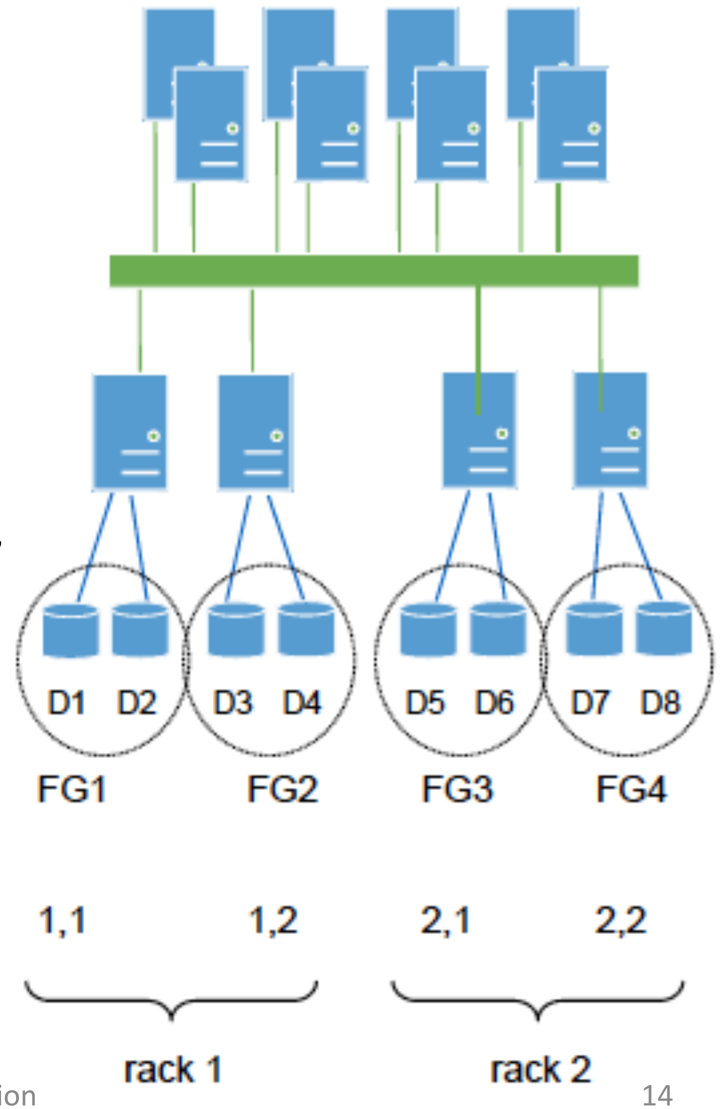


File clones

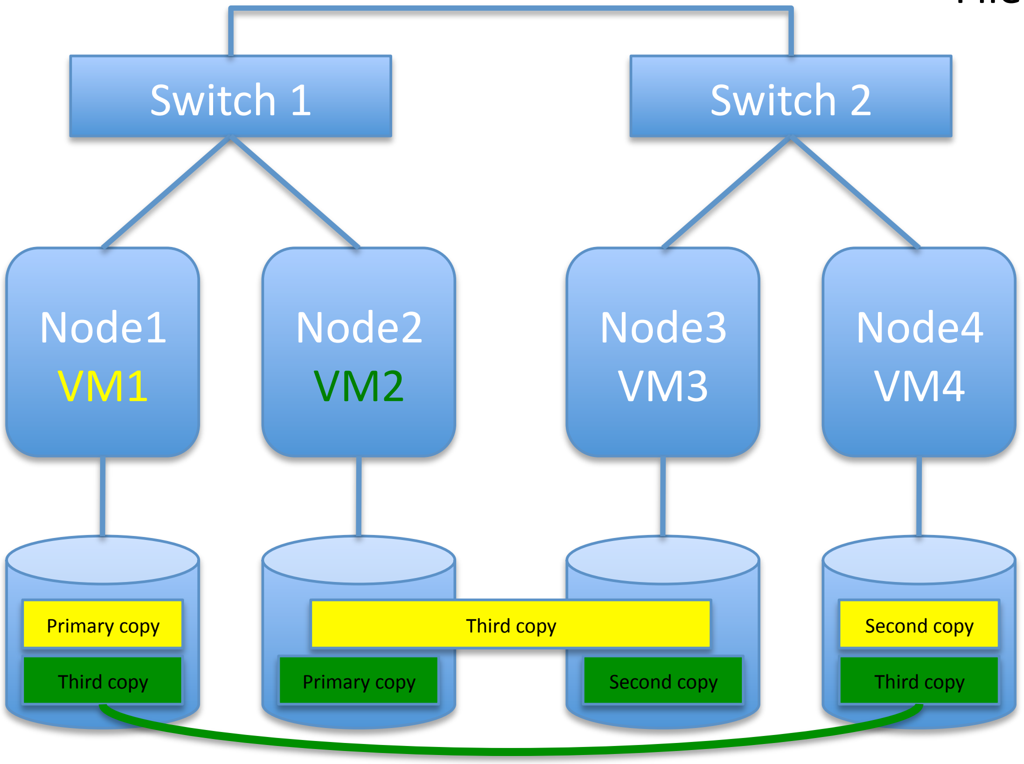
- A file clone is a writable snapshot of an individual file.
- Cloning a file is similar to creating a copy of a file
  - but the creation process is faster and more space-efficient because no additional disk space is consumed until the clone or the original file is modified.
- Multiple clones of the same file can be created with no additional space overhead.

Extended Failure groups

- Failure Group: collection of disks that could become unavailable simultaneously, e.g.,
  - Disks attached to the same storage controlle
  - Disks served by the same NSD server
- Used for two purposes:
  - Replication: replicas of the same block must be on disks in two different failure groups
  - Striping: stripe across failure groups, then across disks within failure group: D1, D3, D5, D7, D2, D4, D6, D8
- GPFS-FPO: “extended failure group”
  - (conveys additional location information)
  - Example: r,n = rack, node within rack with replication 3:
  - second copy placed in a different rack
  - third copy: same rack, but different node



GPFS FPO file system example

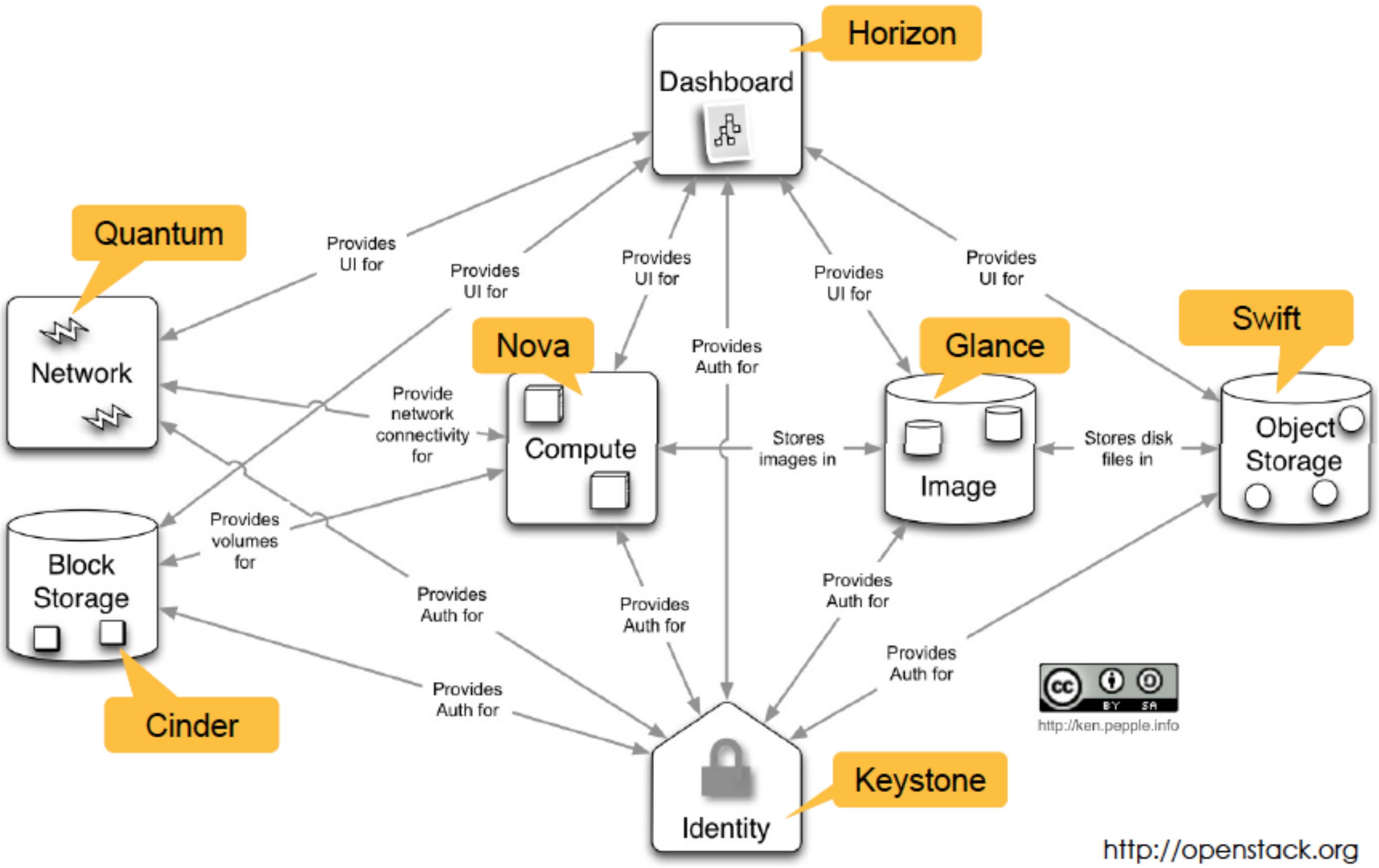


Cluster Configuration parameters:
    readReplicaPolicy local
    restripeOnDiskFailure yes
File System (storage pool):
    allowWriteAffinity=yes

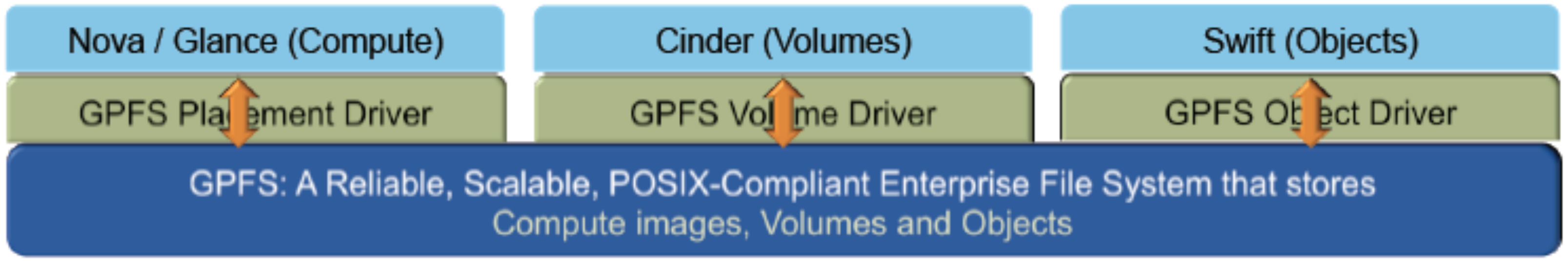
Replication factor:
data and metadata = 3

Failure groups:
Nsd1:failure group 1,0
Nsd2:failure group 1,1
Nsd3:failure group 2,0
Nsd4:failure group 2,1

OpenStack Architecture



GPFS as the Enterprise Storage Layer for OpenStack


Nova / Glance (Compute)
Cinder (Volumes)
Swift (Objects)
GPFS Placement Driver
GPFS Volume Driver
GPFS Object Driver
GPFS: A Reliable, Scalable, POSIX-Compliant Enterprise File System that stores Compute images, Volumes and Objects

- A common storage layer for images, volumes and objects
  - Avoids data copy
  - Local access to data
- Adds enterprise storage management features to OpenStack

Enabling GPFS in Cinder

- To use the Block Storage service with the GPFS driver, set the volume_driver in **cinder.conf**:

`volume_driver = cinder.volume.drivers.ibm.gpfs.GPFSDriver`

- Parameters:

`gpfs_images_dir` <path of the Image service repository>

`gpfs_images_share_mode=copy_on_write`

`gpfs_max_clone_depth=0`

`gpfs_mount_point_base` <path of the GPFS directory where Block Storage volume and snapshot files are stored>

`gpfs_sparse_volumes=True`

`gpfs_storage_pool` <storage pool that volumes are assigned to. By default, the system storage pool is used>

- Volume creation options:

`data_pool_name`

`replicas` <number of replicas>

`write_affinity_depth` <allocation policy for the volume file if `allow-write-affinity` is set in GPFS file system>

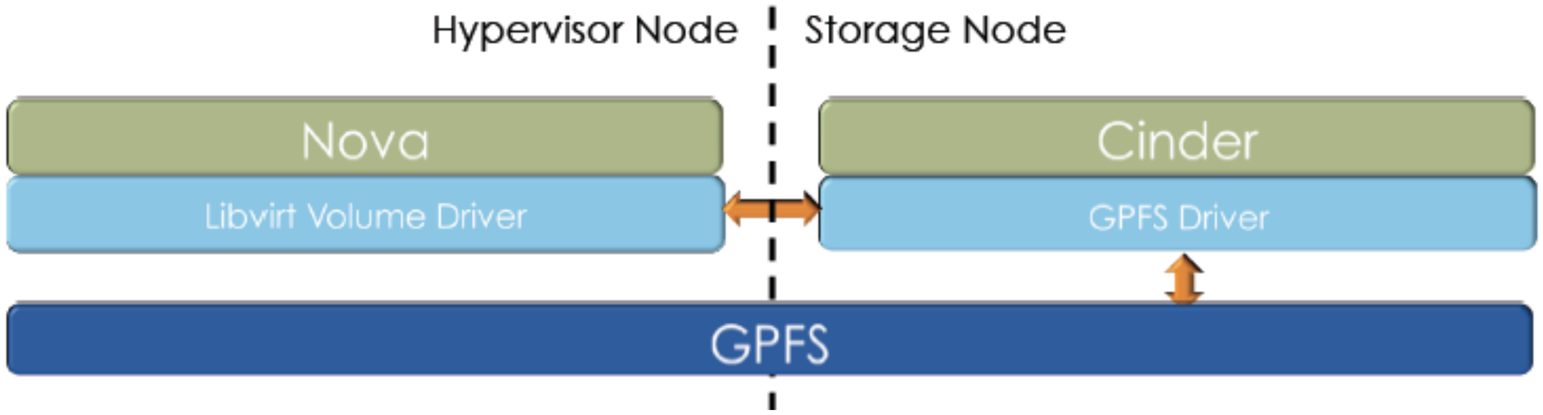
`block_group_factor`

`write_affinity_failure_group`

Cinder: How GPFS driver works

- Similar to NFS driver
- Instances do not actually access a storage device at the block level
- Volume files are created in a GPFS file system and mapped to instances, which emulate a block device
- Optionally, the Image Service can be configured to store images on a GPFS file system.
  - if both image data and volume data reside in the same GPFS file system, the data from image file is moved efficiently to the volume file using copy-on-write (COW) optimization strategy.

Cinder: Volume Services

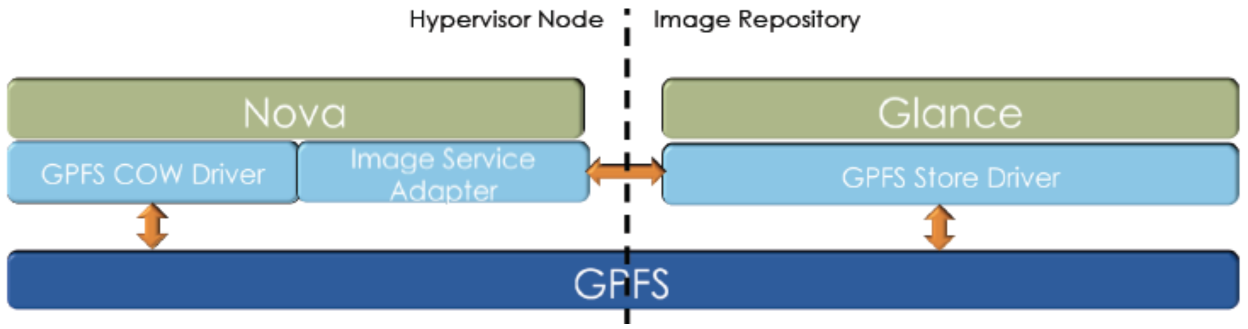


- Cinder driver interface points: Create, Delete, Attach and Detach Volumes, Create Snapshot, Create volume from snapshot, Clone a volume

Create Volume
1. User initiates volume creation through Horizon or command line
2. The request ends up as a call to the GPFS driver
3. GPFS driver creates a (sparse) file and sets the right placement based on user parameters and policy. This operation is "instantaneous".
4. Create from snapshot is similar, except it uses GPFS COW mechanism

Attach Volume
1. User initiates volume attach by selecting the volume and the virtual machine
2. The virtual machine and the volume objects are passed to the Nova Driver which passes them to Libvirt volume driver
3. The Libvrt volume driver invokes Libvirt interface to attach the volume file on GPFS mount point to the specified virtual machine

Glance: Instance Deployment

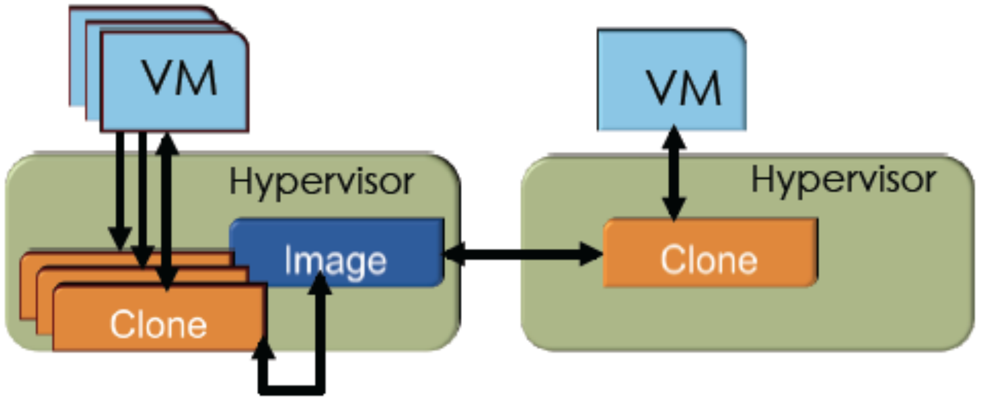


- **GPFS COW Driver**: Implements image creation and caching functions. Interfaces with GPFS to create instance images for provisioning.
- **Image Service Adapter**: Implements the interface with the image repository including image transport.
- **GPFS Store Driver**: Extends the Store base class to implement GPFS specific functions.

Instance Deployment

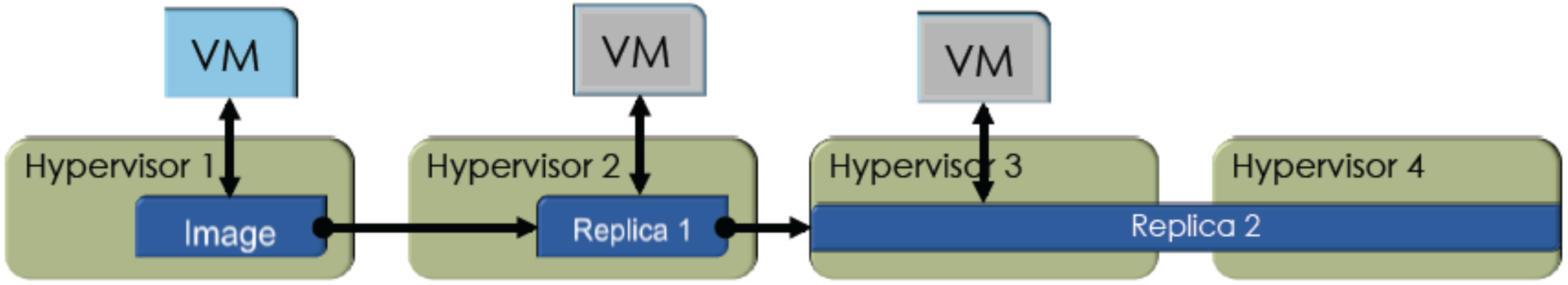
1. User selects an image for instance creation – this refers to an image via the **GPFS Store Driver**
2. The **Nova Image Service adapter** creates a <u>link</u> from image to HyperVisor cache <u>in the same file system</u>
3. **Nova** <u>clones</u> the image using the **COW Driver for GPFS** and creates a VM image instance in a per-HyperVisor cache <u>in the same file system</u>.
4. Libvirt uses the VM image instance to deploy the VM

VM provisioning via File Clones



- the placement of master images and their Clones is independent
- Clone can be on a different node respect to master
  - Writes remain local
  - Reads are streamed
- Different replication level for the cloned instances
- Automatic dedup – both storage and in-memory
- Format and hypervisor agnostic

Transparent Block Level Replication and Recovery



- Pipelined replication
- Configurable file-level replication
  - Store first replica on Hypervisor 2,
  - stripe replica 2 on Hypervisors 3 and 4
- Transparent failover
- Distributed restripe

Enabling GPFS for Glance and Nova

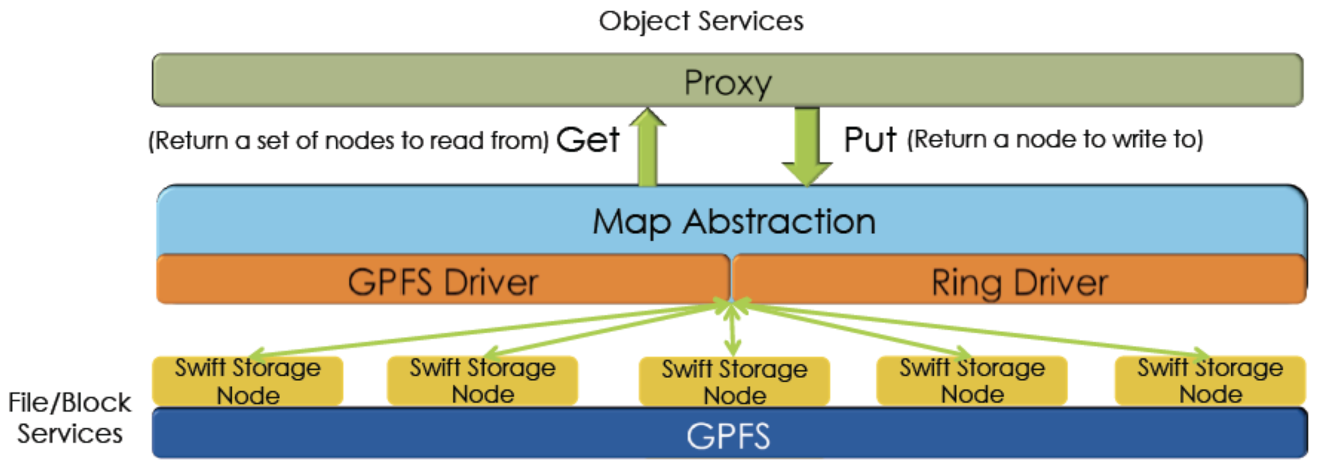
- The initial set of configuration flags for Glance (glance-api.conf and glance-cache.conf):

```
filesystem_store_datadir =<images path in gpfs
filesystem, same as gpfs_images_dir in cinder.conf
>
```

- The initial set of configuration flags for Nova:

```
instances_path=<instances path in gpfs filesystem>
disk_cachemodes=file=writeback
```

OpenStack Integration: Object Store



Two scenarios:

- shared Disks
  - data protection based on RAID, no need for replication
- Local Disks
  - Up to 3-way data replication managed by GPFS

OpenStack Swift and GPFS

- Combination of GPFS and Swift consolidate File and Object under a single shared storage infrastructure.
- The global namespace eliminates the physical client-to-server mappings
  - ideal platform to perform common storage management tasks, such as automated storage tiering and user transparent data migration.

OpenStack Swift and GPFS: Outline

- How to take advantage of the benefits of GPFS when building an object storage solution
- How to install OpenStack Swift on GPFS
  - preferred practices for configuring and tuning both Swift and GPFS
- The Swift and GPFS features that have been tested

Use cases

- Sites that already use GPFS and are seeking to support Swift in the same data plane as their existing data.
- Sites that are seeking an enterprise-ready, cost-efficient, and high-performing object solution.
  – For this use case, GPFS initially provides data management for the object store only, but gives users the option to extend to further types of applications as their requirements grow.

Benefits

- Data protection
  - Delegating the responsibility of protecting data to GPFS (and not using Swift three-way replication) increases both the efficiency and performance of the system
  - GPFS gives every node in the Swift cluster access to every object. If any single node fails, object data is still available from all of the remaining nodes.
  - For objects, there is no need to use Swift replication to make data available from other nodes
- Integration of file and object in a single system
- Very large object support:
  - The maximum single object size that has been tested is 5 TB. This is much larger than the typical Swift object size limit (5 GB)
- Enterprise storage management features:
  - global namespace, encryption, backup, disaster recovery, ILM (auto-tiering), tape integration, and remote caching

Definitions

- **GPFS Object Node**:
  - The physical access points that applications use to store and access objects. These servers run the OpenStack Swift software and the GPFS software.
- **Network Shared Disk (NSD)**:
  - A logical unit number (LUN) that is provided by a storage subsystem for use in a GPFS file system.
- **Storage pool**:
  - A collection of NSDs with similar characteristics.
- **Fileset**:
  - A sub-tree of a file system namespace
- **Replication**:
  - multiple copies of data and metadata for failure containment and disaster recovery.

Placement of Object Store components

- Placing object store data in a separate GPFS management entity called an "*independent fileset*" permits GPFS information lifecycle management (*ILM*) (including snapshots and backup) to uniquely identify and manage the object store data.
- Objects
  – Most GPFS deployments have a single *storage pool* for data, but the *fileset* that contains the object data can be mapped to a separate *storage pool* to provide performance isolation.
- Account and container
  – Place account and container information in the same *fileset* as the object data, which will ease the management of the data by having all the data in a single data management entity.

Swift rings

- Every GPFS Object Node can access the shared file system, so all of the rings are constructed using virtual devices rather than physical devices.
- The virtual devices are subdirectories in the GPFS fileset created for Swift data. When constructing the object ring, virtual devices are added to the “localhost” node. This has the effect of giving every proxy service local access to every virtual device.
- Note:
  – Do not use a single virtual device to contain all object data to avoid access contention between nodes.
    It’s advisable to use a number of virtual devices.
    One approach is to create a reasonable number of virtual devices (for example, 10) for each GPFS Object Node that currently exists, or is expected to exist, in the configuration.

Swift services

- Every GPFS Object Node can access every virtual device in the shared file system, and some Swift object services can be optimized to take advantage of this by running from a single GPFS Object Node:
  - **swift-object-replicator** runs to periodically clean up tombstone files from deleted objects. It is run on a single GPFS Object Node and manages cleanup for all of the virtual devices.
  - **swift-object-updater** is responsible for updating container listings with objects that were not successfully added to the container when they were initially created, updated, or deleted. Like the object replicator, it is run on a single GPFS Object Node.
- **swift-object-auditor** service is not run on any node
  - swift-object-auditor is responsible for comparing the checksum in the object file's extended attributes with the checksum of the object data on disk. If a discrepancy is discovered, the object file is moved to Swift's quarantine directory, with the expectation that the object-replicator will eventually replace the quarantined object with a replica object instance. With GPFS Storage Object, it is better to use storage controller capabilities, such as GPFS Native RAID checksums and disk scrubbing/auditing capabilities.

Swift services on GPFS Object Nodes

Service name	Executes on GPFS Object Nodes
openstack-swift-account	all
openstack-swift-account-auditor	all
openstack-swift-account-reaper	all
openstack-swift-account-replicator	all
openstack-swift-container	all
openstack-swift-container-auditor	all
openstack-swift-container-updater	all
openstack-swift-container-replicator	all
openstack-swift-object	all
openstack-swift-object-auditor	none
openstack-swift-object-expirer	All (started if object expiration is enabled)
openstack-swift-object-replicator	single
openstack-swift-object-updater	single
openstack-swift-proxy	all

GPFS installation and configuration

- The GPFS software is not required on the controller node where Keystone identity service running if it's not a Object Node.
- After the GPFS is installed and a GPFS cluster is created, the following steps must be completed:
  - Create a GPFS file system and an *independent fileset*.
  - Mount the GPFS file system on all Object Nodes and ensure that the fileset is linked.
    - The *independent fileset* will be used only for **Swift object storage**.
  - Create the base directory where all object data will be stored inside the newly created fileset.
    - here this base directory is referred to as gpfs_object_base .

GPFS tuning for Swift data

- Setting the number of *i-nodes*
  - The suggested *Swift Partition Power to use is 8*. With this value, the maximum number of *i-nodes* needs to be set to <u>twice the maximum number of expected objects</u>
  - To set the maximum *i-node* limit for a *fileset*, use the following command:

```
mmchfileset FileSystem Fileset --inode-limit MaxNumInodes[:NumInodesToPreallocate]
```

- Data and metadata cache
  - Use *mmchconfig* to set this GPFS *pagepool* to be at least 30% - 50% of memory on each GPFS Object Node.

GPFS tuning (cont.)

- maxFilesToCache
  - This parameter limits the total number of different files that can be cached at one time. This needs to be set using mmchconfig to be large enough to handle the number of concurrently open files (objects), plus allow caching of recently used files.
- maxStatCache
  - This parameter sets aside additional pageable memory to cache attributes of files that are not currently in the regular file cache. Increasing this value can improve the performance of background Swift services that scan directories.
- seqDiscardThreshold
  - The default for this value is 1 MB, which means that if you have a file that is sequentially read and is greater than 1 MB, GPFS does not keep the data in cache after consumption. Because everything in Swift is read sequentially, unless this value is increased, GPFS will not cache anything; so this value needs to be increased to the largest expected object size that needs to be cached in memory.

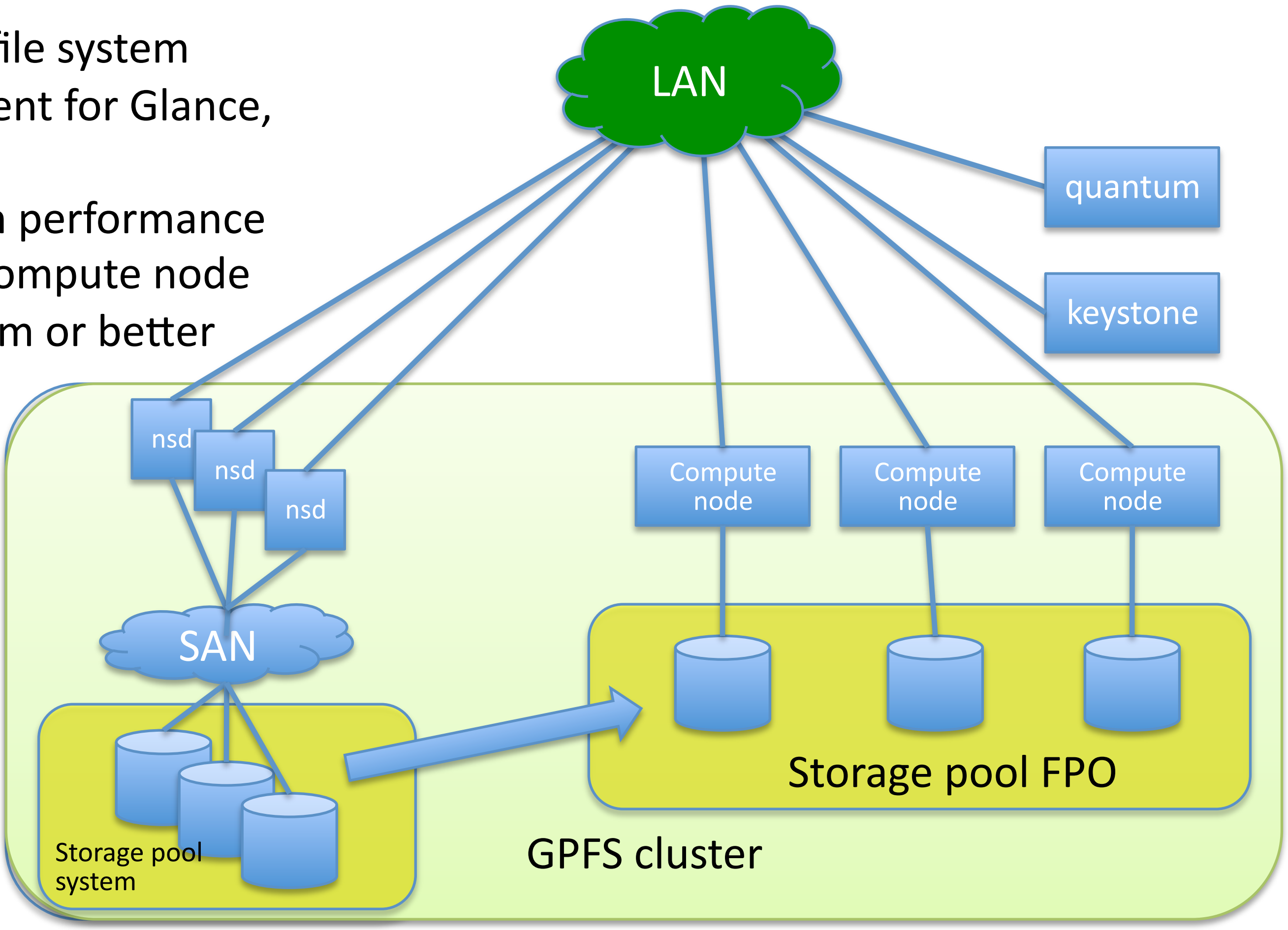
System administration considerations

- Adding a GPFS Object Node
  - Add new node to GPFS cluster
  - copy the existing *ring files* from any existing *GPFS Object Node* to the new node
  - Update load balancer configuration
  - move selected *virtual devices* onto the new node by using the *swift-ring-builder set_info* command
- Note:
  - <u>There is no movement of your data </u>required when you perform this operation. *The ring structure* is all that is changed, and then the *proxy services* can send *account* and *container* requests to the new node in addition to the existing nodes

OpenStack Deployment example on existing GPFS file system

1. Use existing GPFS file system for initial deployment for Glance, Cinder and Swift
2. If not satisfied with performance move to FPO adding Compute node disks to GPFS file system or better (faster) shared disks

This can be done using GPFS policy and without service interrupting



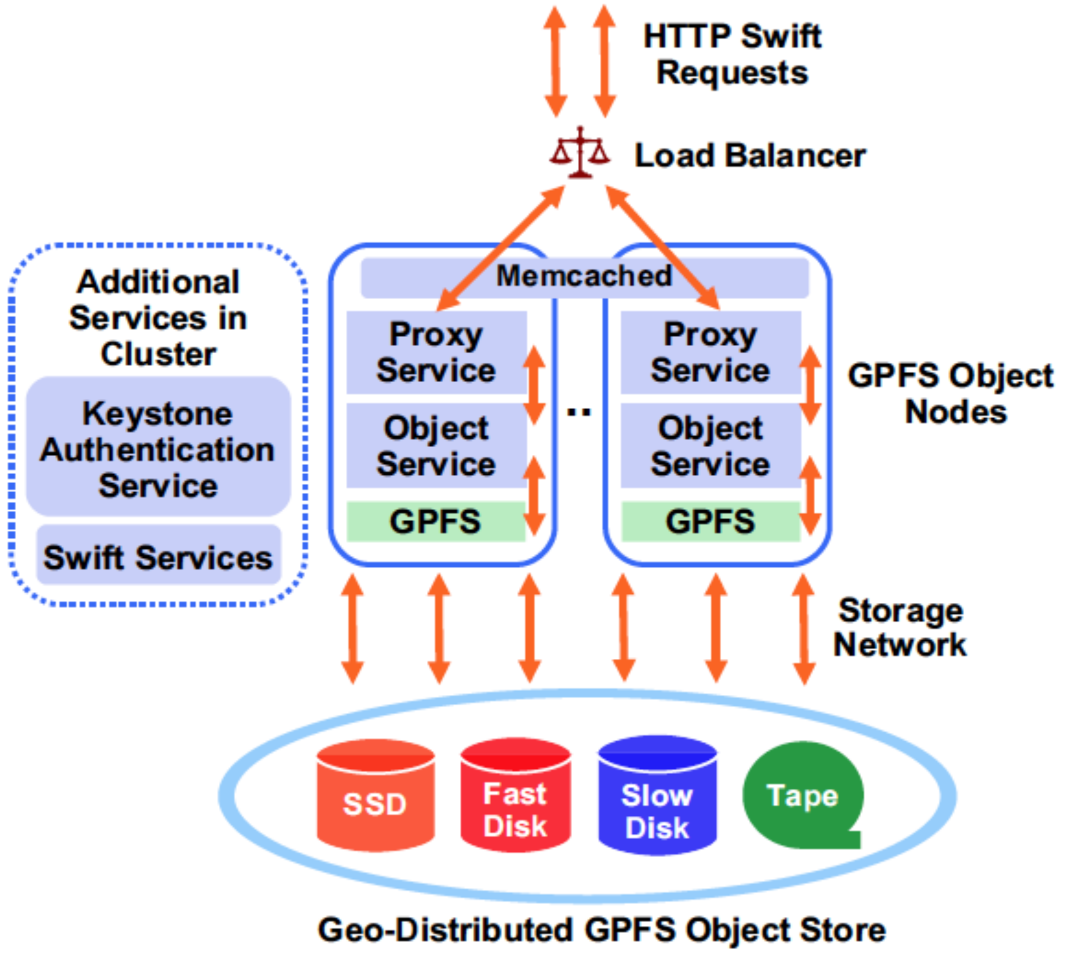
References

- IBM RedBook: A Deployment Guide for Elastic Storage Object http://www.redbooks.ibm.com/abstracts/redp5113.html
- GPFS-OpenStack Integration http://www.gpfsug.org/wp-content/uploads/2013/05/24-04-13_GPFS-OpenStack_DS.pdf
- Video demo *OpenStack*® Cinder driver support for GPFS https://www.youtube.com/watch?v=q-O-VPs0K8w
- OpenStack Documentation portal http://docs.openstack.org/juno/config-reference/content/GPFS-driver-background.html

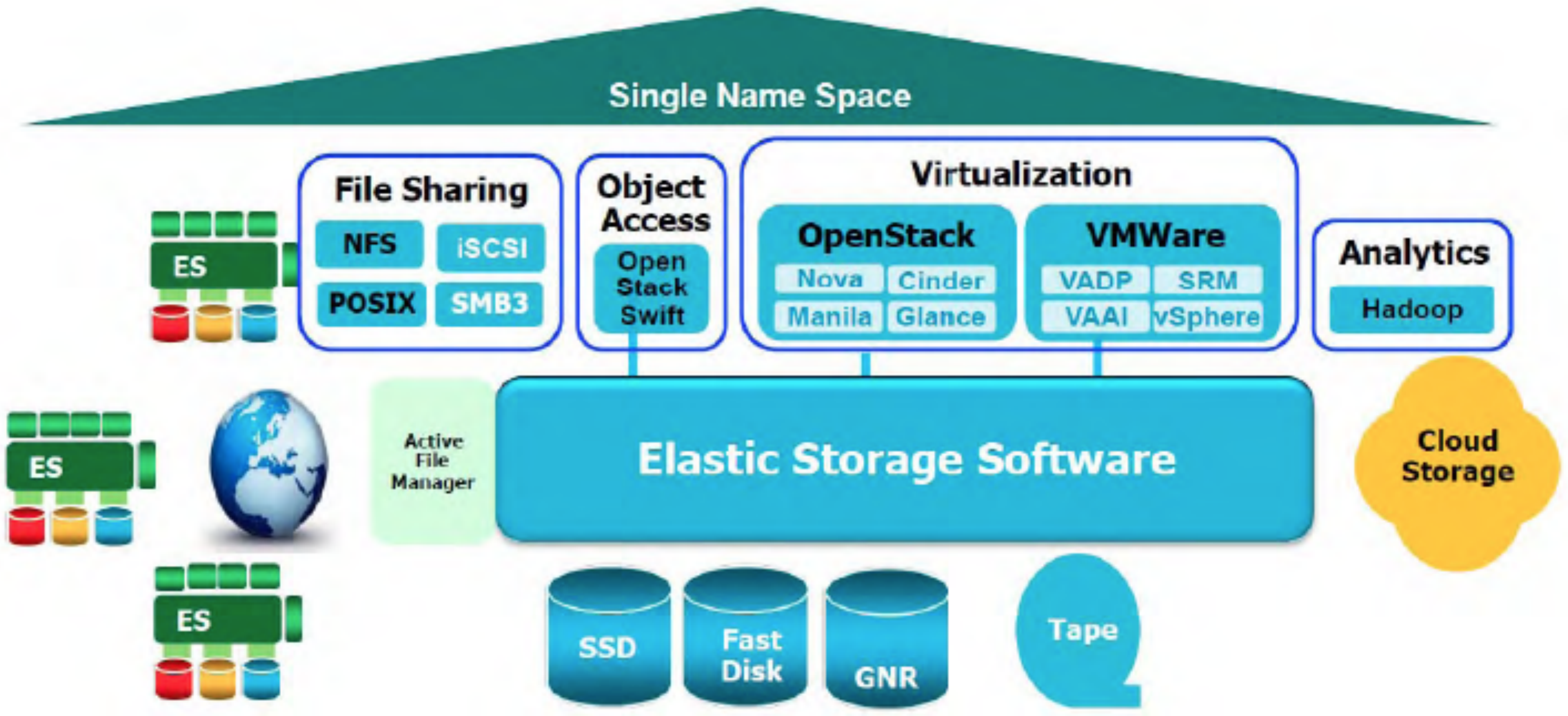
Questions?

Backup slides

Geo distributed GFPS Object Store



GPFS Elastic Storage Vision



GPFS 4.1 new features

- Clustered NFS (**cNFS**)
  - NFSv4 and IPv6 support
- **Quota** management
  - enable and disable on-line
  - quota files are metadata now (no user.quota, group.quota files in FS)
- Local read-only cache (**LROC**)
  - Overflow pagepool to local storage (SSD)
  - more RAM for application
- **FPO** - file placement optimizer (Hadoop-like)
  - data locality when restriping
  - AIO performance
  - Mixed storage pools in one file system

GPFS 4.2: NSD V.2 format

provides the following benefits:

- Includes a partition table so that the disk is recognized as a GPFS device
- Adjusts data alignment to support disks with a 4 KB physical block size
- Adds backup copies of some key GPFS data structures
- Expands some reserved areas to allow for future growth