SDC
STORAGE DEVELOPER CONFERENCE
SNIA SANTA CLARA, 2015

OpenStack SwiftOnFile: User Identity for Cross Protocol Access Demystified

Dean Hildebrand, Sasikanth Eda

Sandeep Patil, Bill Owen

IBM

Overview

- OpenStack Swift Architecture
- Swift-on-File Architecture
- Spectrum Scale for Object Storage Architecture
- File and Object Use Cases
- OpenStack Swift ACL Vs. File ACL semantics
- Modes to Support Compatibility
  - Non-Unified Identity Between Object and File Interface
  - Unified Identity Between Object and File Interface
- Implementation approaches

OpenStack Swift Architecture: Overview

OpenStack Swift is a highly available, distributed, eventually consistent object/blob store[1].

- Wide range of usecases including web / mobile applications, backups, active archiving
- One of 3 OpenStack storage services (Cinder block & Manila file)
- 100% python
- 35 kloc with over 70 kloc more in unit, function & EI test code
- Vibrant community, top contributing companies for Juno include:
  - SwiftStack, Rackspace, Redhat, HP, Intel, IBM, Box

[1] http://docs.openstack.org/developer/swift/

OpenStack Swift Features

Multi-tenancy
ACLs
Role-based Auth
HTTP/HTTPS

Storage automation

Cost Savings through simplified management and use of cheapest and densest storage

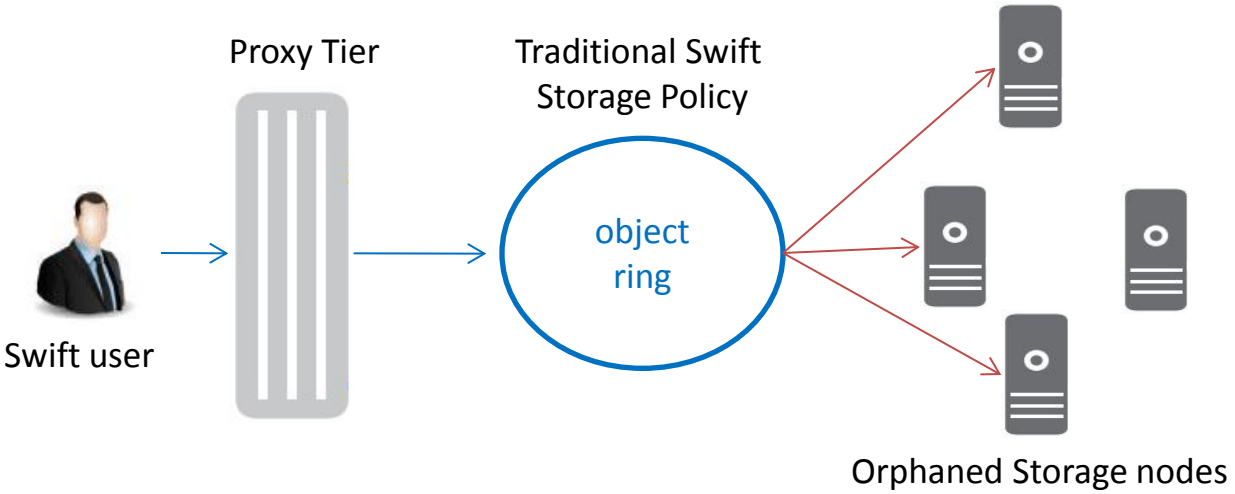
Mac/Windows/Linux
Swift and S3 API support
SDKs
User-defined metadata

Geo-replication
High-Availability
Flat namespace

Extensible Middleware
Versioning
Quotas
Expiration
Rate Limiting
Rolling upgrades

4

OpenStack Swift Architecture: Overview



Swift storage path:

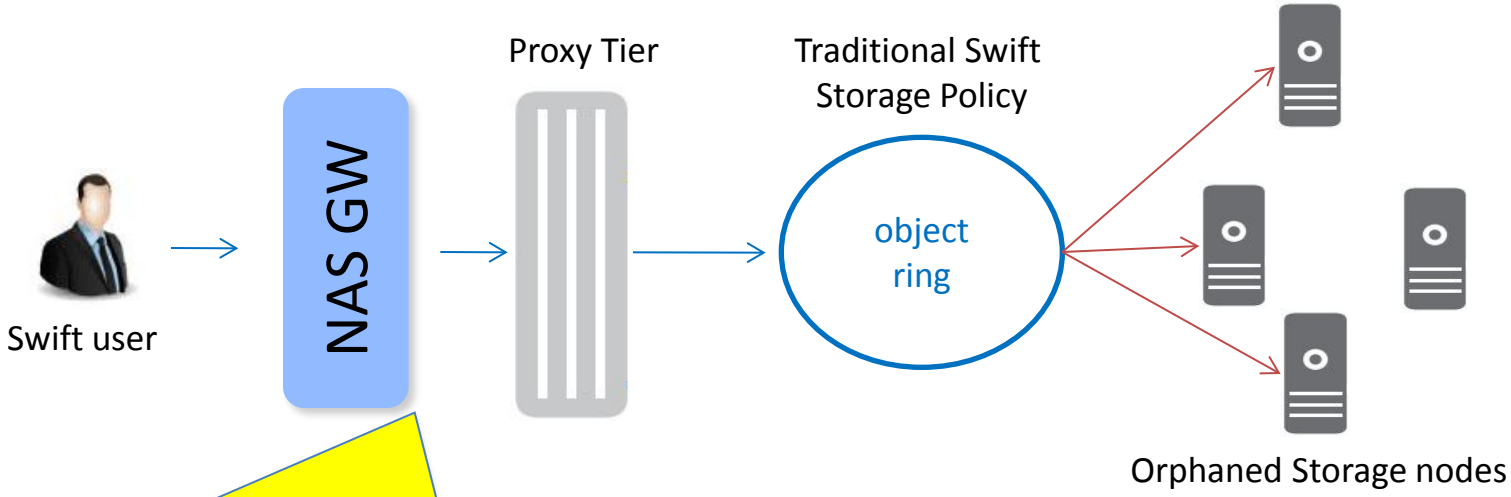
-rwxr-xr-x 1 swift swift 29 Aug 22 09:25
/mnt/sdb1/2/node/sdb2/objects/981/f79/
d7b843bf79/1401254393.89313.data

User request -> Proxy tier -> Storage policy -> Ring -> storage location / path

File Access to Objects Use Cases

1. Use file API as transition to object API
2. Create and Share
    - Create data via file interface and share globally using object interface
    - Example use cases include HPC, video, legacy API access to objects
3. Sync/Archive and Analyze
    - Running Analytics directly through Swift/S3 API limits functionality
    - While HDFS connectors exist for Swift/S3, they have
        - Limited functionality since Hive and HBase (among others) are not supported due to file 'append' requirement
        - Poor performance due to loss of data location on writes, load imbalances etc.
4. Simplified management plane
    - Manage file and object within single system

OpenStack Swift Architecture: Problem
Not Designed for File Access to Objects



- Poor performance due to inefficient NAS “copy and change” gateways
- Users mistakenly think they can do POSIX

Swift-on-File Architecture: Overview

✓ Swift Object Server implementation (Diskfile) [2]

- Data can be stored and retrieved through ***Swift and S3 and from NAS***

✓ Enabled as a Swift Storage Policy

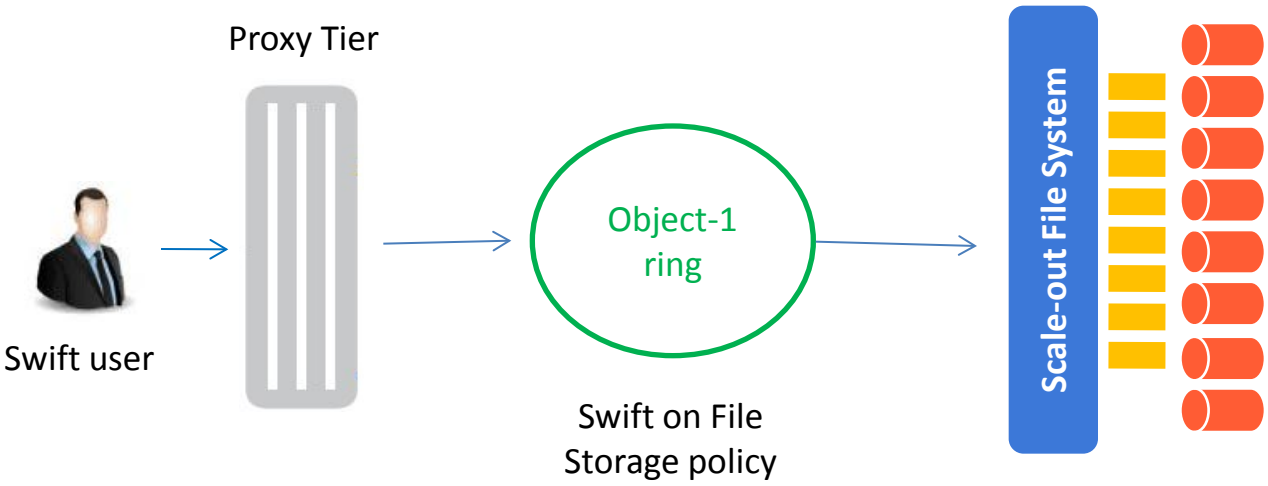
✓ Stores objects on any scale-out file system

- Stores objects ***once*** in scale-out file system
- Leverages Scale-out File System data protection

✓ ***Stores objects following the same path as the object's URL***

[2] https://github.com/stackforge/swiftonfile/blob/master/README.md#swift-on-file

Swift-on-File Architecture: Overview



SwiftOnFile storage path:

-rwxr-xr-x 1 swift swift 29 Aug 22 09:25
/mnt/swiftonfile/cont/obj1

Path accessible via
NFS / CIFS / POSIX

User request -> Proxy tier -> Swift on File policy -> Swift on File Ring -> storage location / path

Swift-on-File Architecture: Overview

This object:
http://swift.example.com/v1/acct/cont/obj

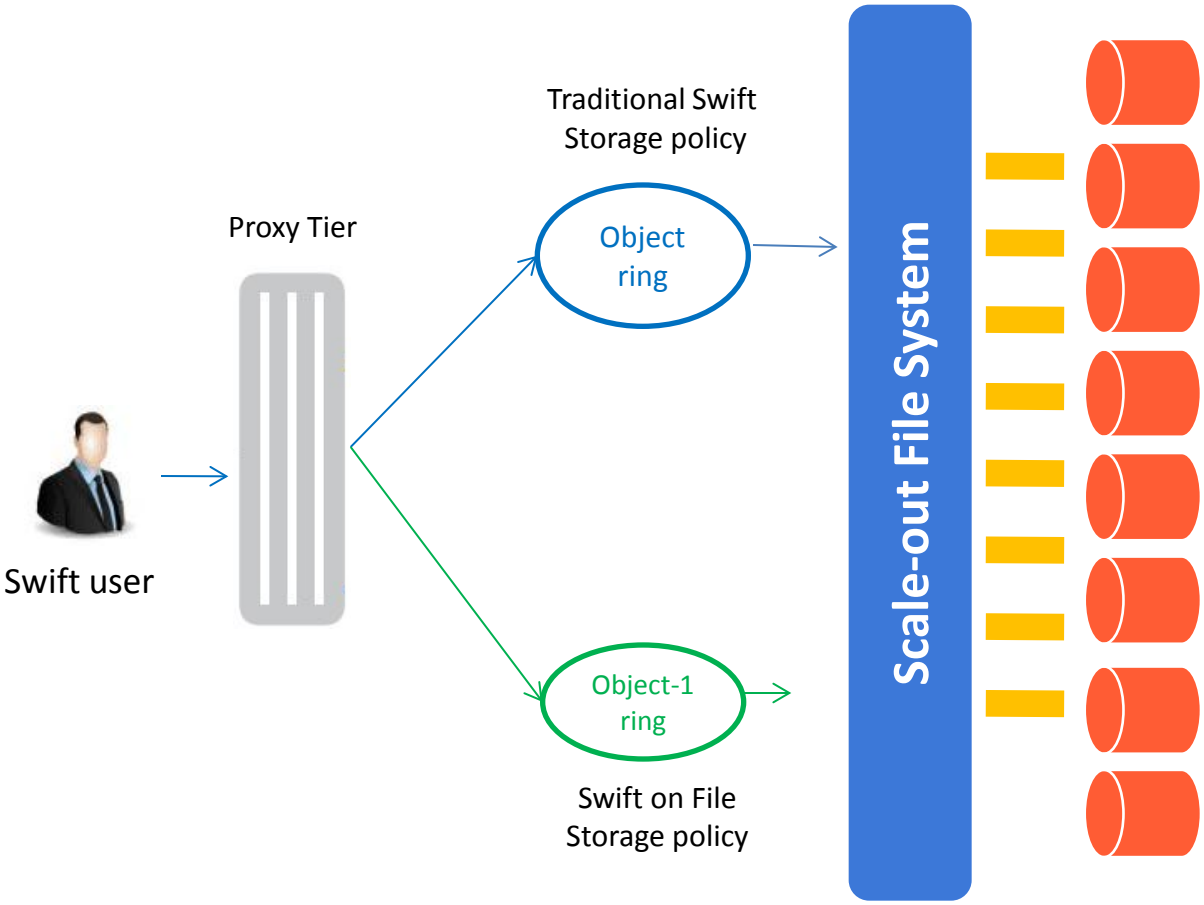
was stored with Swift here:

/mnt/sdb1/2/node/sdb2/objects/981/f79/f566bd022b9285b05e665fd7b843bf79/1401254393.89313.data

But is now stored with SwiftonFile here:

/mnt/scaleoutFS/acct/cont/obj

Co-Existence of Traditional and Swift-On-File Object Placement



Not useful for Analytic / File workloads

Swift storage path:

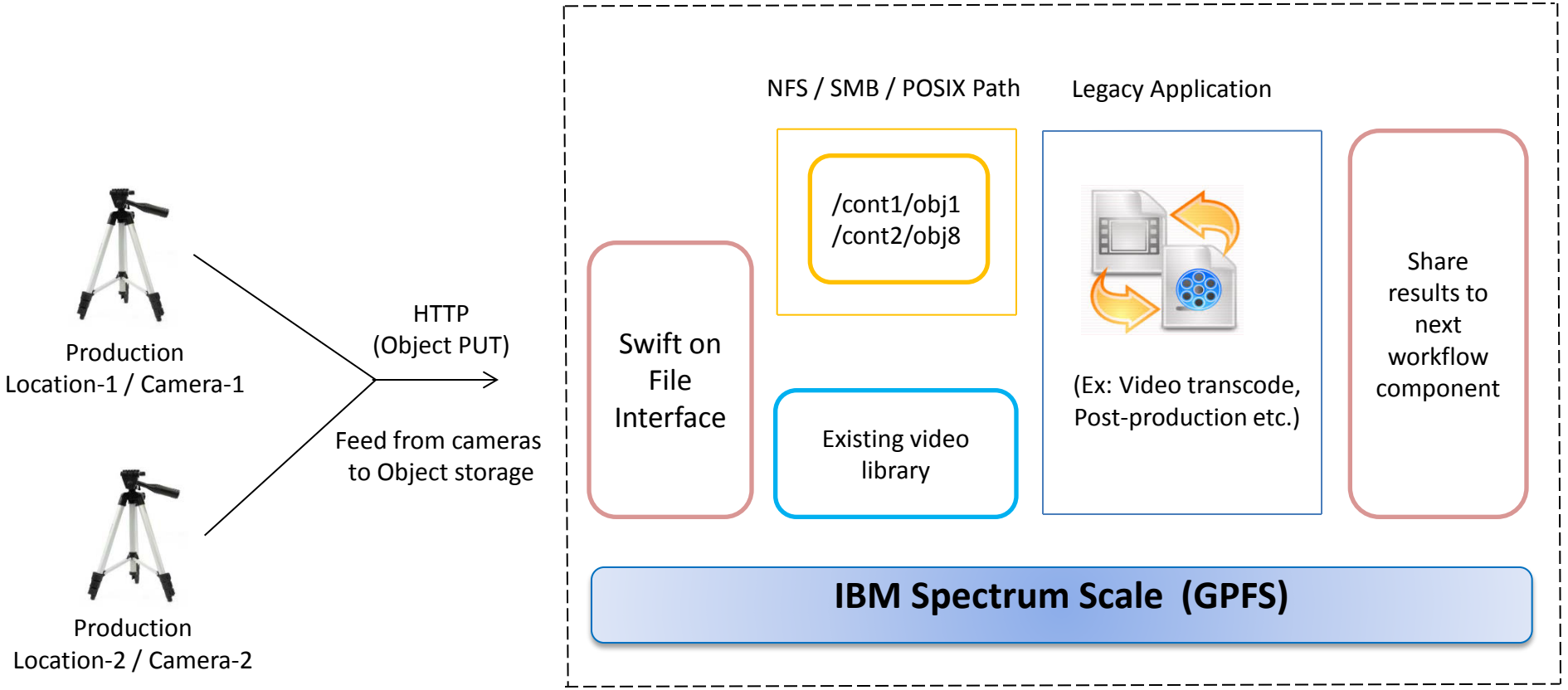
-rwxr-xr-x 1 swift swift 29 Aug 22 09:25
/mnt/sdb1/2/node/sdb2/objects/981/f79/d7b843bf79/1401254393.89313.data

Appropriate for Analytic / File workloads

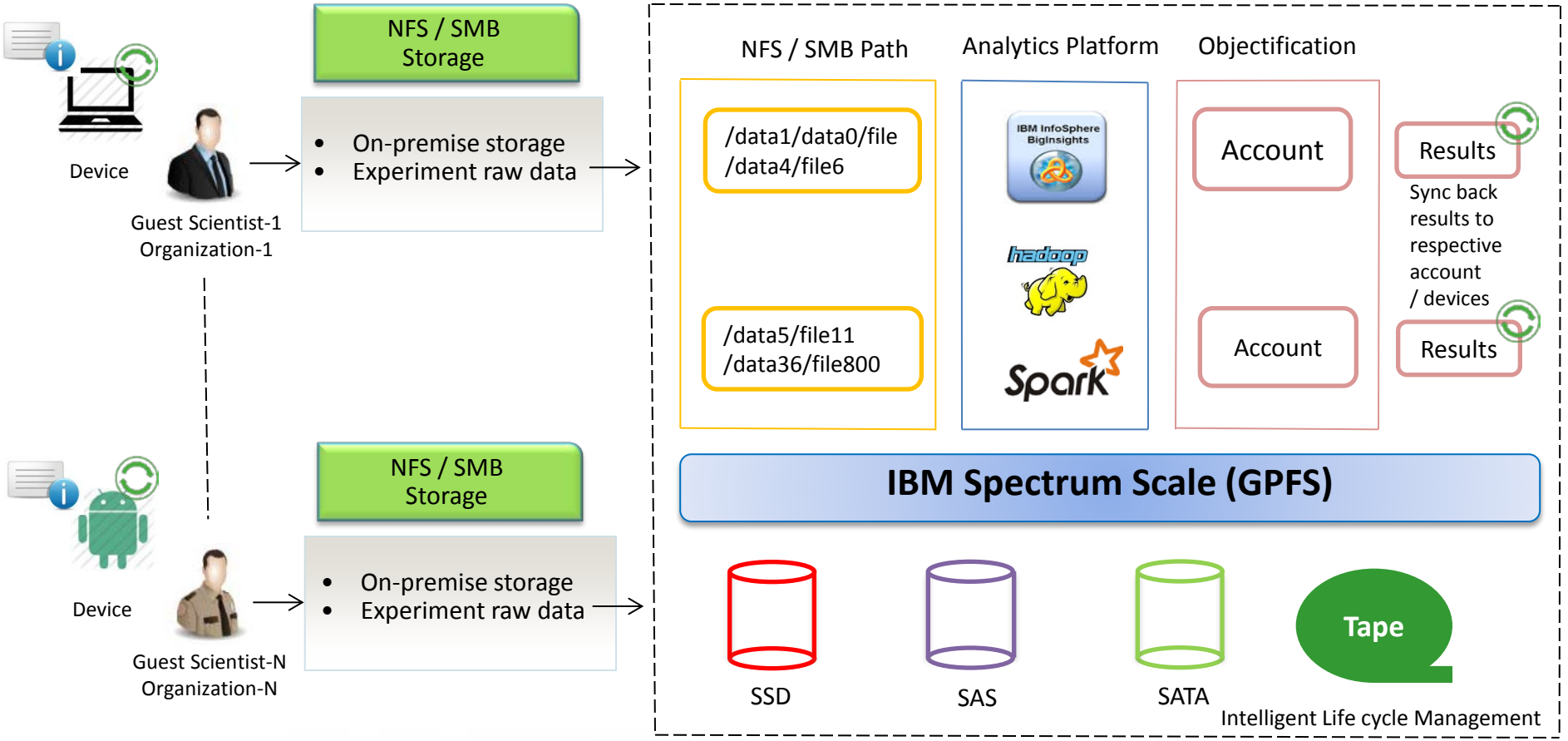
SwiftOnFile storage path:

-rwxr-xr-x 1 swift swift 29 Aug 22 09:25
/mnt/swiftonfile/container/object1

Swift-on-File Usecase 1:
Video Capture and Analysis

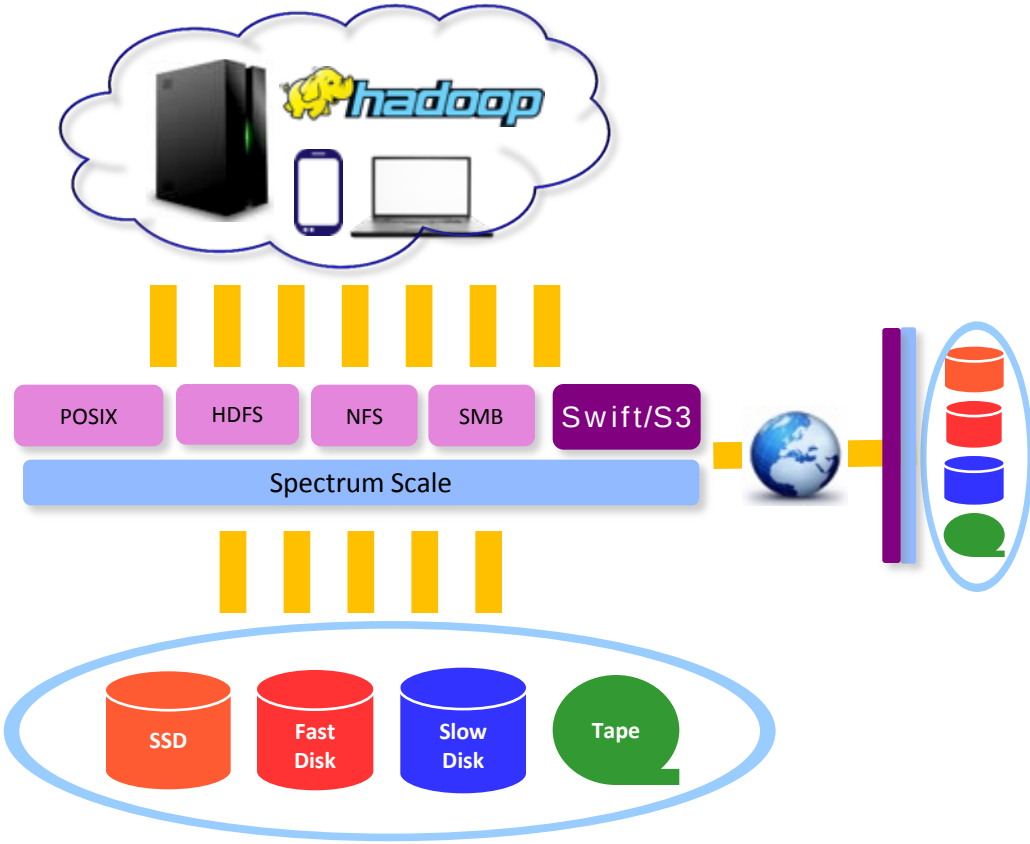


Swift-on-File Usecase 2:
Secure Analytics (End-to-End Life Cycle Management)



Spectrum Scale for Object Storage

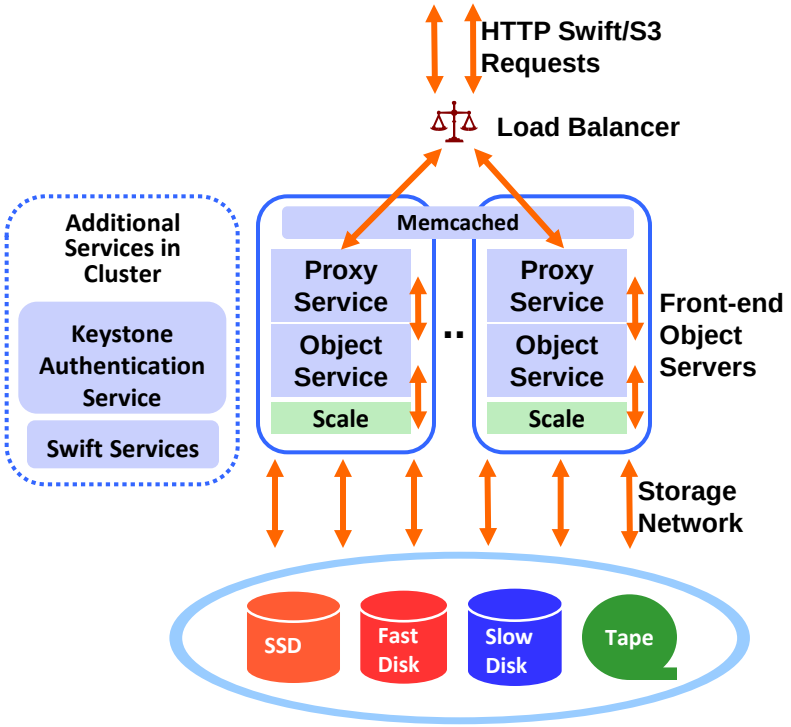
- **Combine strengths of Spectrum Scale and OpenStack Swift**
- Eliminate data migration through native File and Object integration
  - POSIX/NFS/SMB/S3/Swift
- High performance and scalability
- Authentication integration (LDAP/AD)
- Data protection
  - Snapshots, Backup, Disaster Recovery
- Encryption
- Compression
- Integrated or software-only solutions
- External storage integration
  - TSM, LTFS, Optical



14

Spectrum Scale for Object Storage
Detailed Architecture

- Run all Swift and Scale processes on all front-end object servers
- Front-end servers access data directly from storage system
- Objects are erasure coded on storage cluster using Spectrum Scale Native RAID
- Use Swift policies to map containers to Spectrum Scale Filesets with specific features e.g. encryption, compression.



15

OpenStack Swift ACL Semantics

- ✓ Swift ACL enables owner to set READ/WRITE access rights
- ✓ ACL's are set and stored at container level (db)
- ✓ SWIFT ACL's are not associated with the user's User ID or Group ID, unlike the File world
- ✓ Controllable via headers
  - X-Container-Read, X-Container-Write, X-Remove-Container-Read, X-Remove-Controllable-Write

Example (Read ACL):

```
$ swift stat container1
Account: AUTH_test
Container: container1
Objects: 17
Bytes: 29
Read ACL: test:tester2
Write ACL:
Sync To:
Sync Key:
Accept-Ranges: bytes
X-Storage-Policy: SwiftOnFile
X-Timestamp: 1440323340.30419
X-Trans-Id: txd285be9a47d940018e1fd-0055e26f81
```

Example (Write ACL):

```
$ swift stat container2
Account: AUTH_test
Container: container2
Objects: 9
Bytes: 261
Read ACL:
Write ACL: test:tester2
Sync To:
Sync Key:
Accept-Ranges: bytes
X-Storage-Policy: SwiftOnFile
X-Timestamp: 1439524393.73931
X-Trans-Id: tx6e2d8962e3e5431bbf7e6-0055cdf823
```

File ACL Semantics

In file world, there exits predominantly two kinds of ACL support;

✓POSIX ACLs - Associated with three sets of permissions that define access for the owner, the owning group, and for others. Each set may contain Read (r), Write (w), and Execute (x) permissions

✓NFSv4 ACLs - Provides finer granularity than typical POSIX read/write/execute permissions and are similar to SMB ACLs.

Both these ACL's are based on User's User ID and Group ID

• Other features that are tied with User ID properties include Quota, Backup, ILM functionalities.

In contrast to Object ACL's, File ACL's are:

✓Granular and comprehensive

✓Supports Inheritance (file inherits its parent directory permissions at the time of creation)

✓ACL's can be set at both file as well as directory level

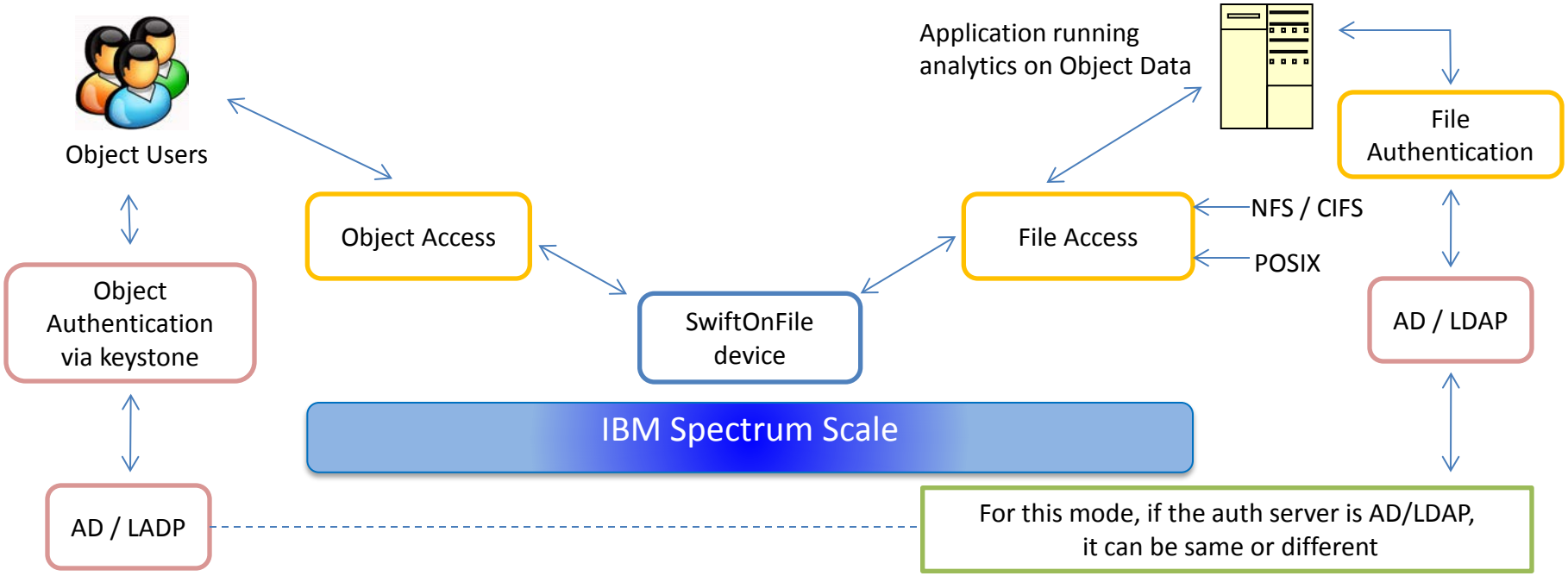
Modes to Support Compatibility

In order to achieve compatibility, the first step is to provide an ability to have File User ID mapping with the SWIFT users.

Two proposed modes:

- Mode 1: Non-Unified Identity
- Mode 2: Unified Identity

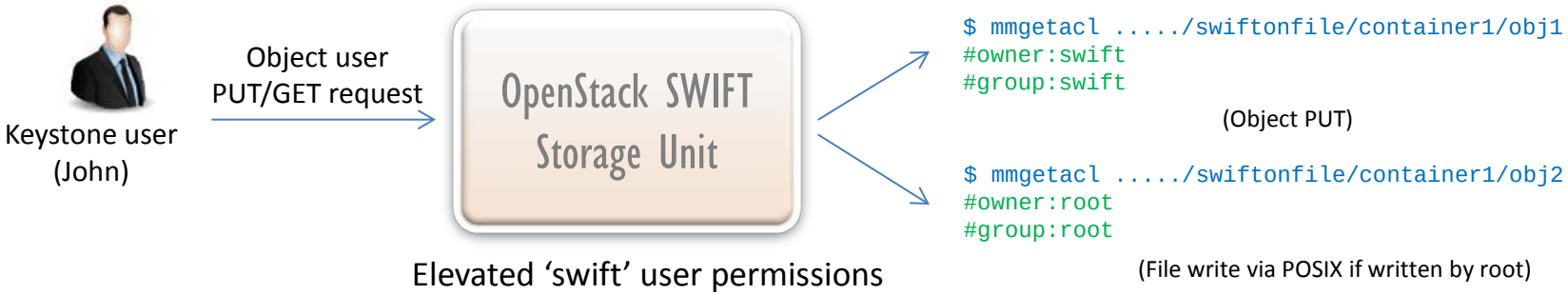
Mode 1:
Non-Unified Identity Between Object and File



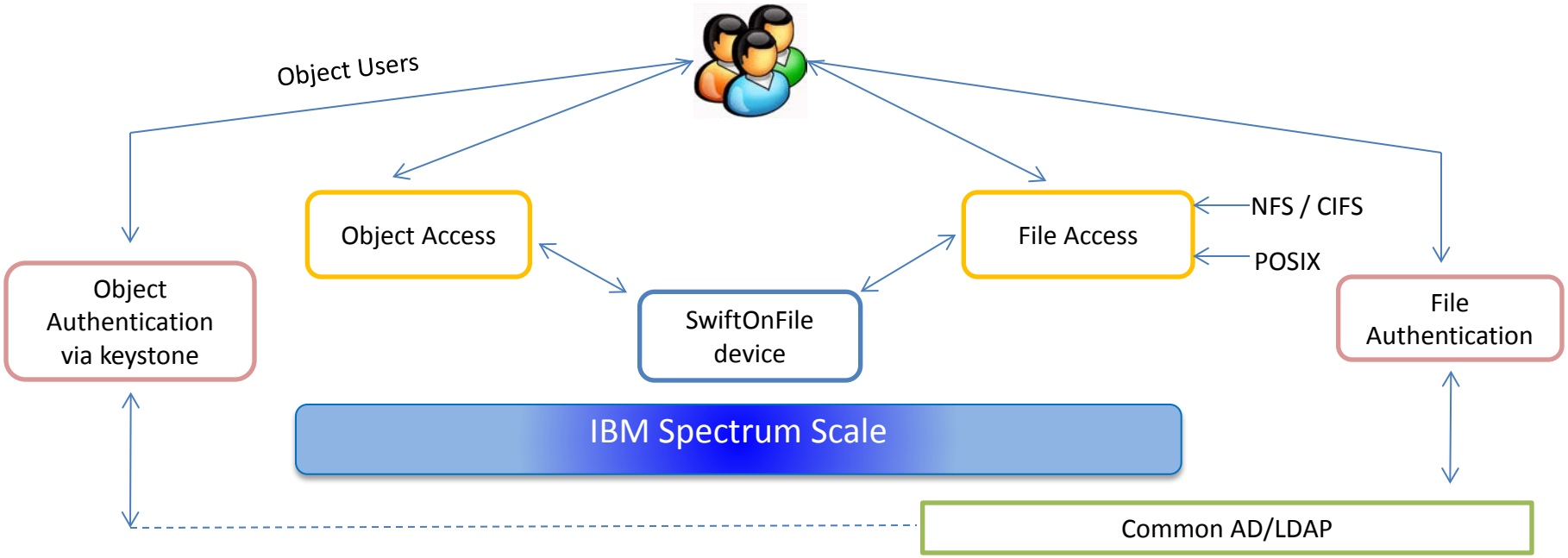
- Data created via object API will be available for application via file API using;
  - Root
  - Newly defined special user
  - User given explicit ACLs
- Data created via file API will be accessible via object API
  - Must elevate ‘swift’ user permissions

Mode 1:
Non-Unified Identity Between Object and File

- ✓ **Object authentication setup is independent of File Authentication** setup
  - End user can make use the common authentication server incase of AD/LDAP
- ✓ Data created by **Object API owned by "swift" user or must have full access to "swift" user data**
- ✓ **Application processing object data from file API needs the required file ACL to access the data**
- ✓ File access should ensure that object data always retains full access to "swift" user (can be achieved using DAC_OVERRIDE CAPABILITIES).



Mode 2:
Unified Identity Between Object and File



- ✓ Common set of Object and File users using same directory service (AD+RFC 2307 or LDAP)
- ✓ Objects created using Swift API will be owned by the user performing the Object operation (PUT)
  - **Design Decision**: If object already exists, existing ownership of File will be retained

Unified Identity Between Object and File

- ✓ **Design Decision (Authorization)**:
  - Object access will follow Object ACL semantics
  - File access will follow File ACL semantics

- ✓ Retaining ACL and XATTR
  - If an object update is performed then existing “file ACL” and “XATTR” will be retained
- ✓ For an object update operation
  - ✓ No explicit “file ACL” will be set for that user
- ✓ For initial PUT operation of an object over a nested directory
  - ✓ Object owned by user but no explicit ACL will be set for that user over the nested directories

Quick Swift Terminology Aside

- ✓ Middleware[3] is used to add additional functionality to Proxy, Object, and Container/Account WSGI servers
  - ✓ Inject code on both the request and response paths
  - ✓ Easy way to customize a Swift deployment
  - ✓ Numerous supported Middleware features as well as several other middleware modules available
  - ✓ E.g., Swift3, Rate Limiting

- ✓ Diskfile[4] forms a disk abstraction layer for the object server
  - ✓ Customizes the API and layout of how objects are stored
  - ✓ Example APIs: POSIX, Kinetic
  - ✓ Example layout: Standard Swift, Swift on File

[3] http://docs.openstack.org/developer/swift/development_ondisk_backends.html

[4] http://docs.openstack.org/developer/swift/development_middleware.html

Accessing Objects via File without Ownership: Implementation

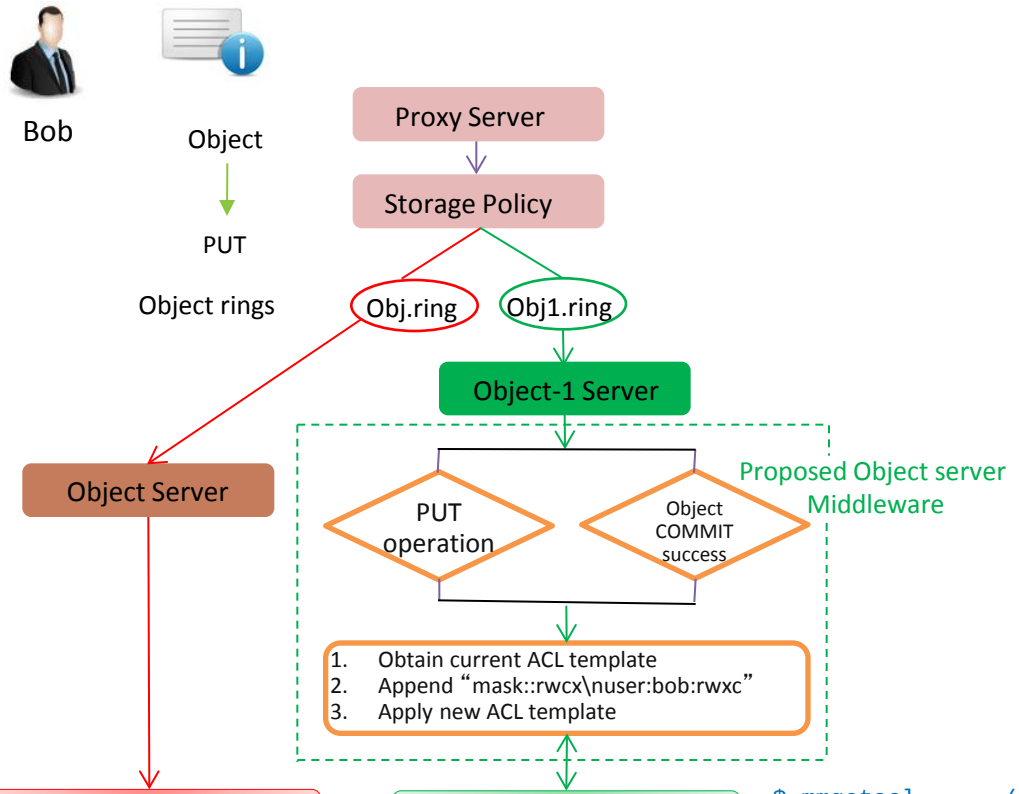
1. Proxy Server Middleware
   - Collect username from request (or obtain it using Auth token) and pass to object server

2. Object Server DiskFile module
   - Perform "ACL inheritance / append" operation on object with obtained username

[3] http://docs.openstack.org/developer/swift/development_ondisk_backends.html

[4] http://docs.openstack.org/developer/swift/development_middleware.html

Accessing Objects via File without Ownership: Architecture




```
$ mmgetacl /objects/c24/18.72.data
#owner:swift
#group:swift
user::rw-c
group::----
other::----
```

```
$ mmgetacl ...../sof/container1/obj1
#owner:swift
#group:swift
user::rw-c
group::----
other::----
mask::rwxc
user:bob:rwxc
```

Accessing Objects via File WITH Ownership
Implementation

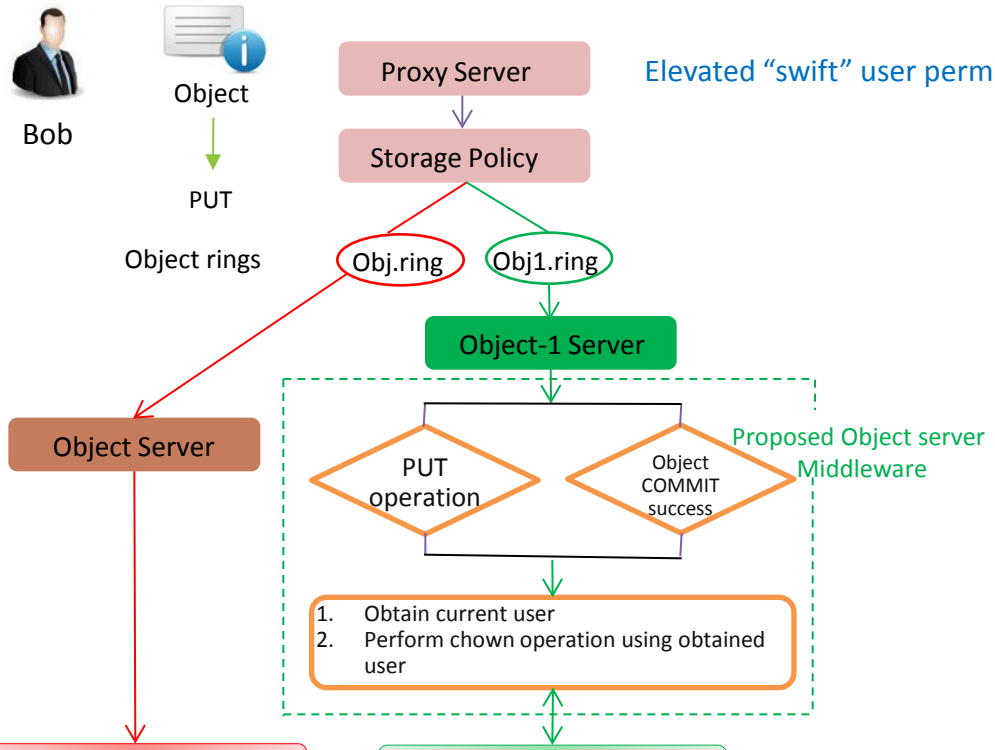
1. Elevate permissions to user configured with Swift
   - Typically “swift” user

2. Proxy Server Middleware
   - Collect username from request (or obtain it using Auth token) and pass it to object server

3. Object Server DiskFile module
   - Perform “chown” operation on object with the obtained username

Accessing Objects via File WITH Ownership

Architecture

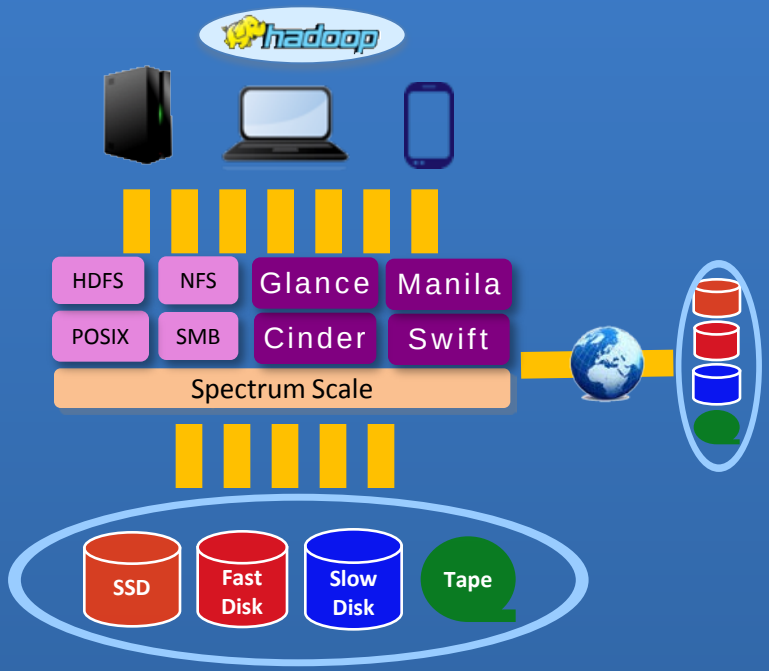



```
$ mmgetacl /objects/c24/18.72.data
#owner:swift
#group:swift
user::rw-c
group::----
other::----
```

```
$ mmgetacl ...../sof/container1/obj1
#owner:bob
#group:bob
user::rw-c
group::----
other::----
```

IBM Spectrum Scale

Data management at scale



- Avoid vendor lock-in with true Software Defined Storage and Open Standards
- Seamless performance & capacity scaling
- Automate data management at scale
- Enable global collaboration

OpenStack and Spectrum Scale helps clients manage data at scale

Business: I need virtually unlimited storage

An open & scalable cloud platform

Operations: I need a flexible infrastructure that supports both object and file based storage

A single data plane that supports Cinder, Glance, Swift, Manila as well as NFS, et. al.

Operations: I need to minimize the time it takes to perform common storage management tasks

A fully automated policy based data placement and migration tool

Collaboration: I need to share data between people, departments and sites with low latency.

Sharing with a variety of WAN caching modes

Results

- Converge File and Object based storage under one roof
- Employ enterprise features to protect data, e.g. Snapshots, Backup, and Disaster Recovery
- Support native file, block and object sharing to data [coming in 2015]

Thank you