OpenStack Swift

OpenStack Summit
Atlanta 2014

Martin Lanner & Hugo Kuo
May 15, 2014

Agenda

- **Object Storage & Swift**
- **How Swift Works**
- **Installing Swift: Hands-on Lab**
  - Manual Swift installation (20 minutes)
  - SwiftStack installation (15 minutes)
- **Operating, Managing and Monitoring Swift**
- **Failure Handling**

Why Object Storage?

- Data grows at ~50% per year
- 50%-75% of all data is unstructured or of archival nature
- Modern application design, using RESTful API (HTTP)
- High availability
- Agile data centers

Why Swift?

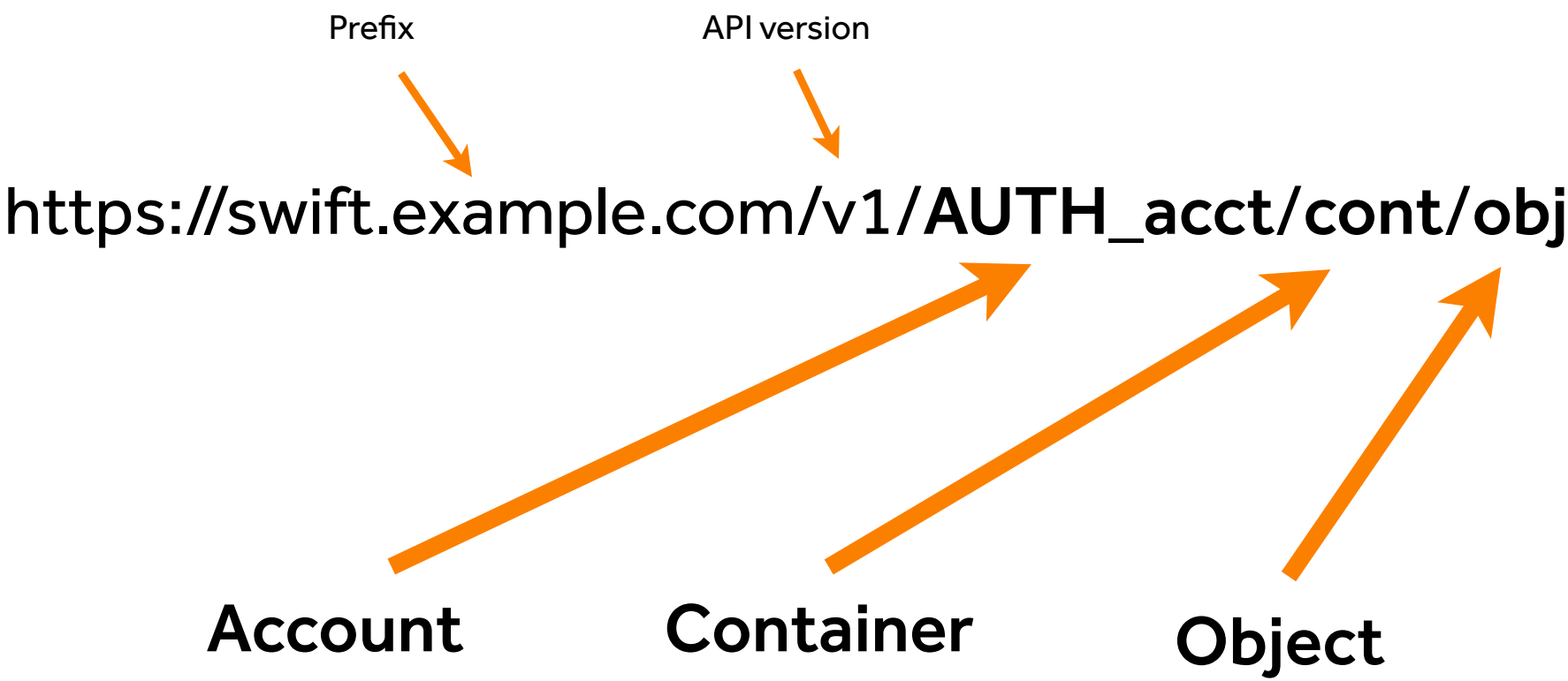
- Open source
- Proven at scale: > 100PB
- Actively developed by ~15 core devs and > 150 committers
- Deploy in your own data center
- Unique features:
  - Multi-region cluster, geographic distribution of data
  - Storage policies
  - Erasure Coding

How Swift Works

Swift Design Goals

Reliable	Configurable replica model with zones & regions Easy to use HTTP API – Developers don't shard High Concurrency (Supports lots of users)
Highly scalable	Multi-tenant – each account has its own namespace Tier & scale any component in the system
Hardware proof	No Single Point of Failure (High Availability) Assumes unreliable hardware Mix & match hardware vendors

The Swift API



The Swift API

Write a new object:

```
PUT /v1/account/container/object_name
```

Read an object:

```
GET /v1/account/container/object_name
```

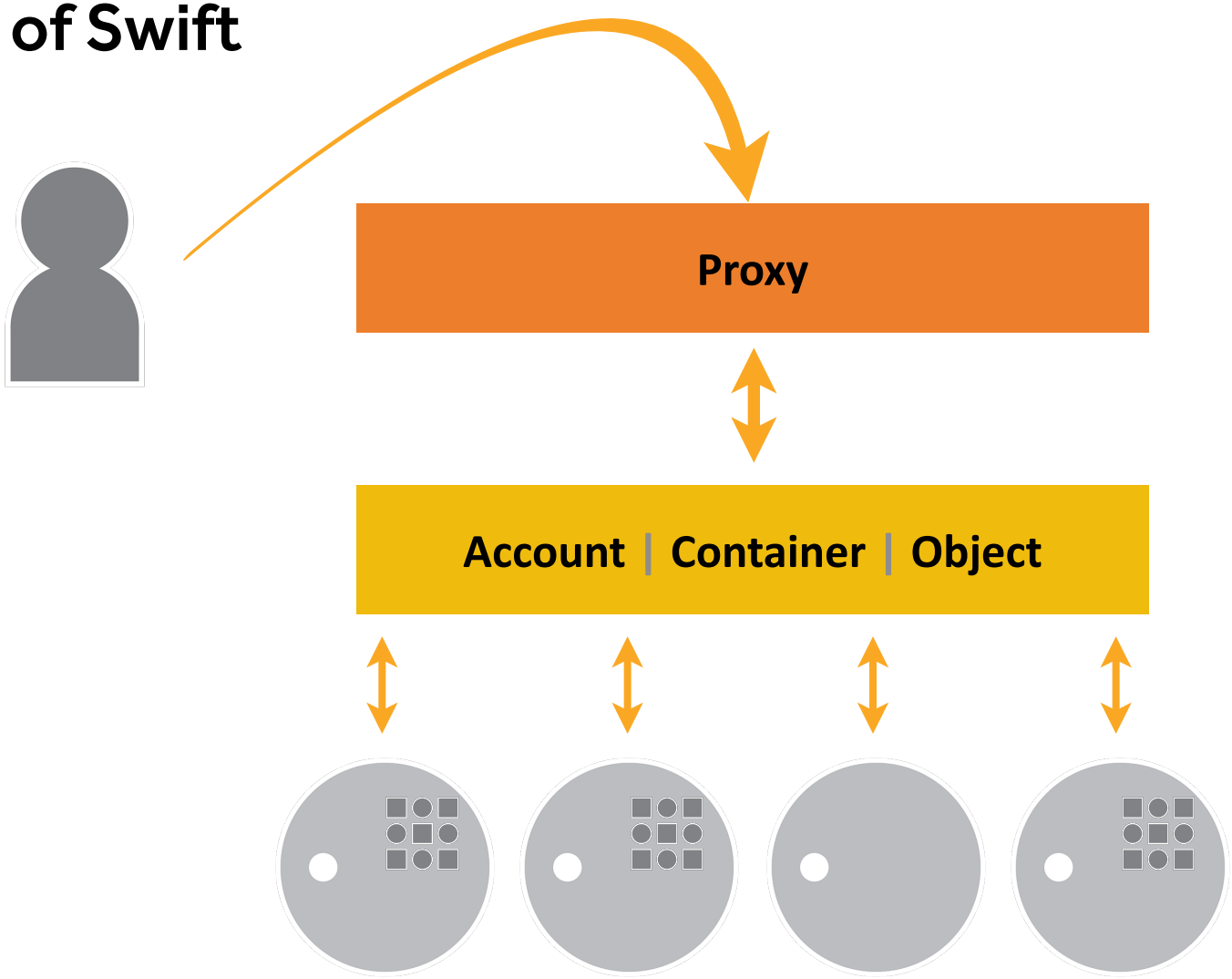
Swift Overview

- **RESTful API**

- **Swift Components**
  - Proxy Server
  - Account Server
  - Container Server
  - Object Server

- **The Rings**
  - Regions
  - Zones
  - Devices
  - Partitions

Parts of Swift

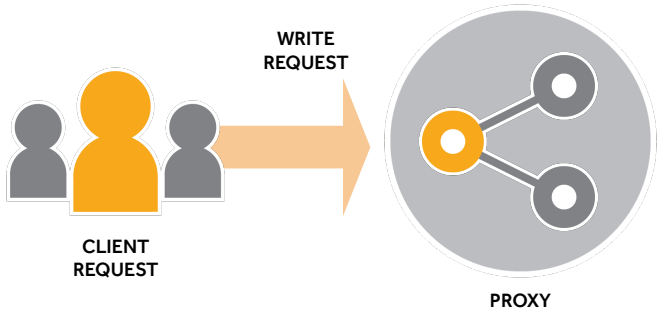


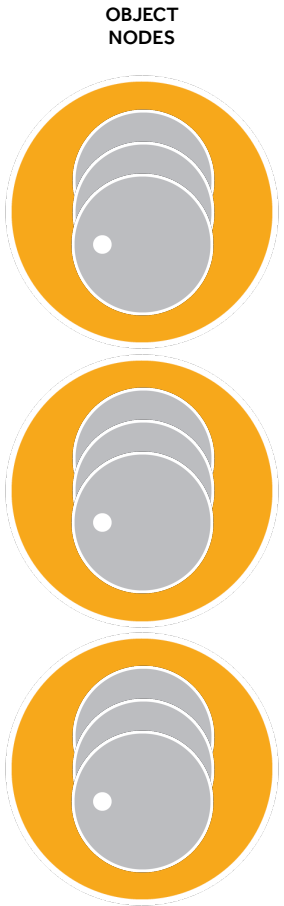
System Components: Proxy

Write Request

Client sends a write request to the cluster

One proxy is randomly selected to serve request



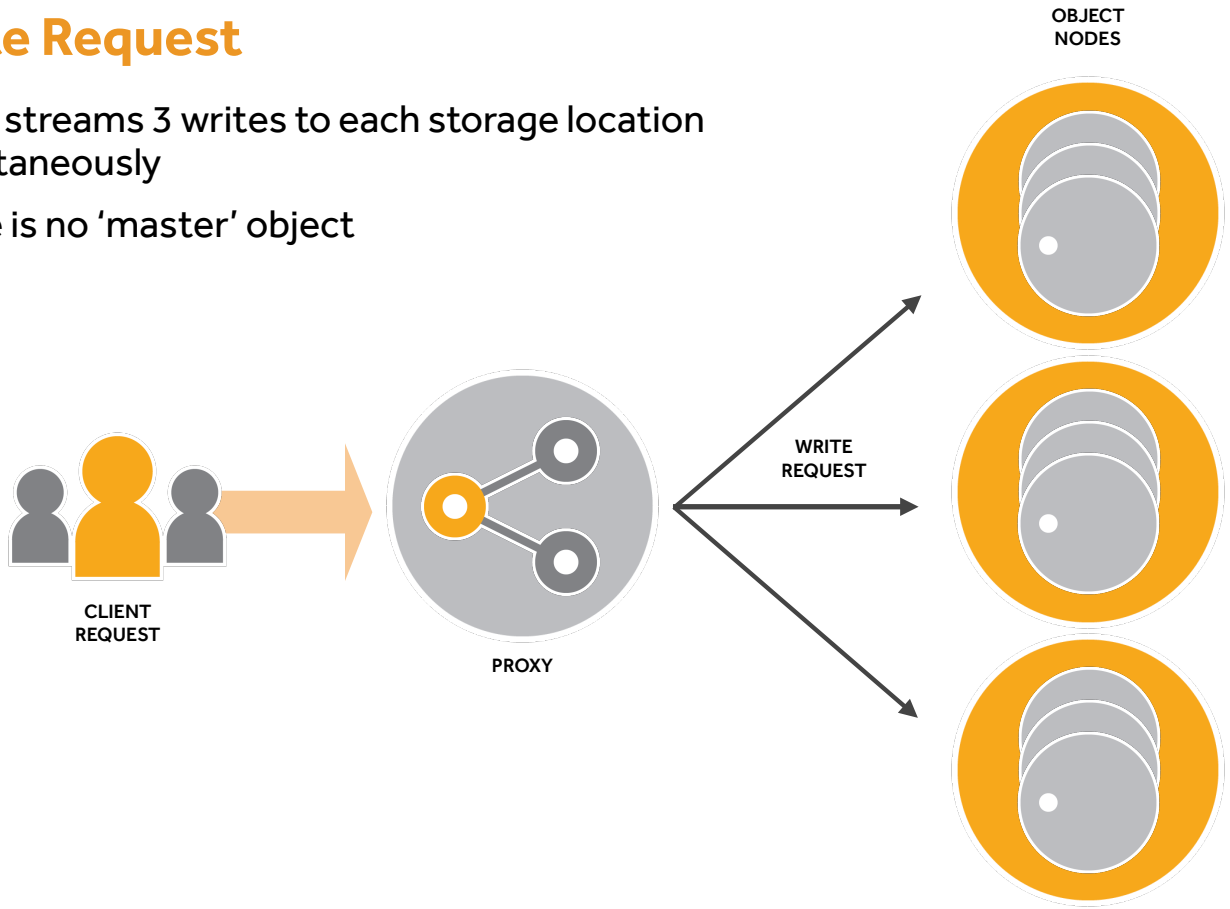


System Components: Proxy

Write Request

Proxy streams 3 writes to each storage location simultaneously

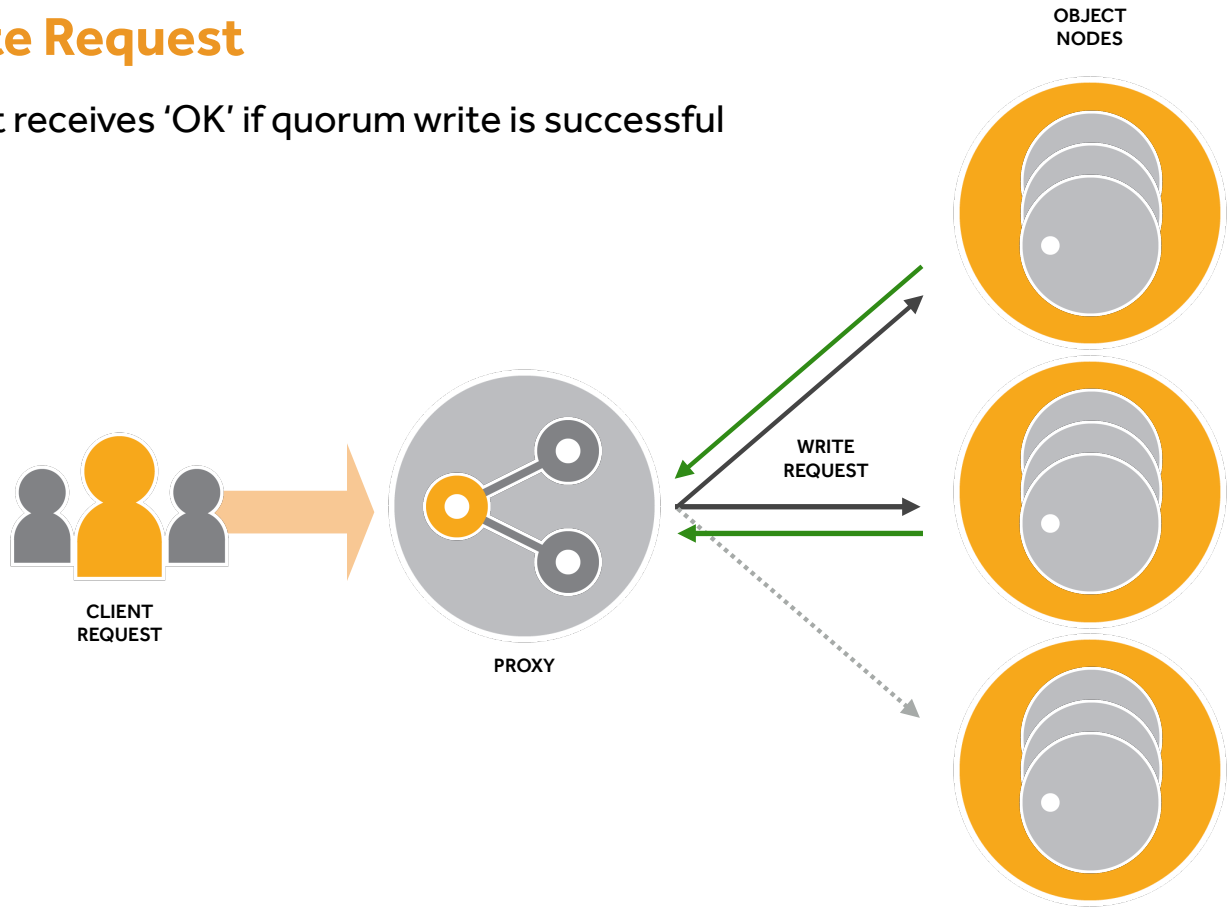
There is no ‘master’ object



System Components: Proxy

Write Request

Client receives ‘OK’ if quorum write is successful



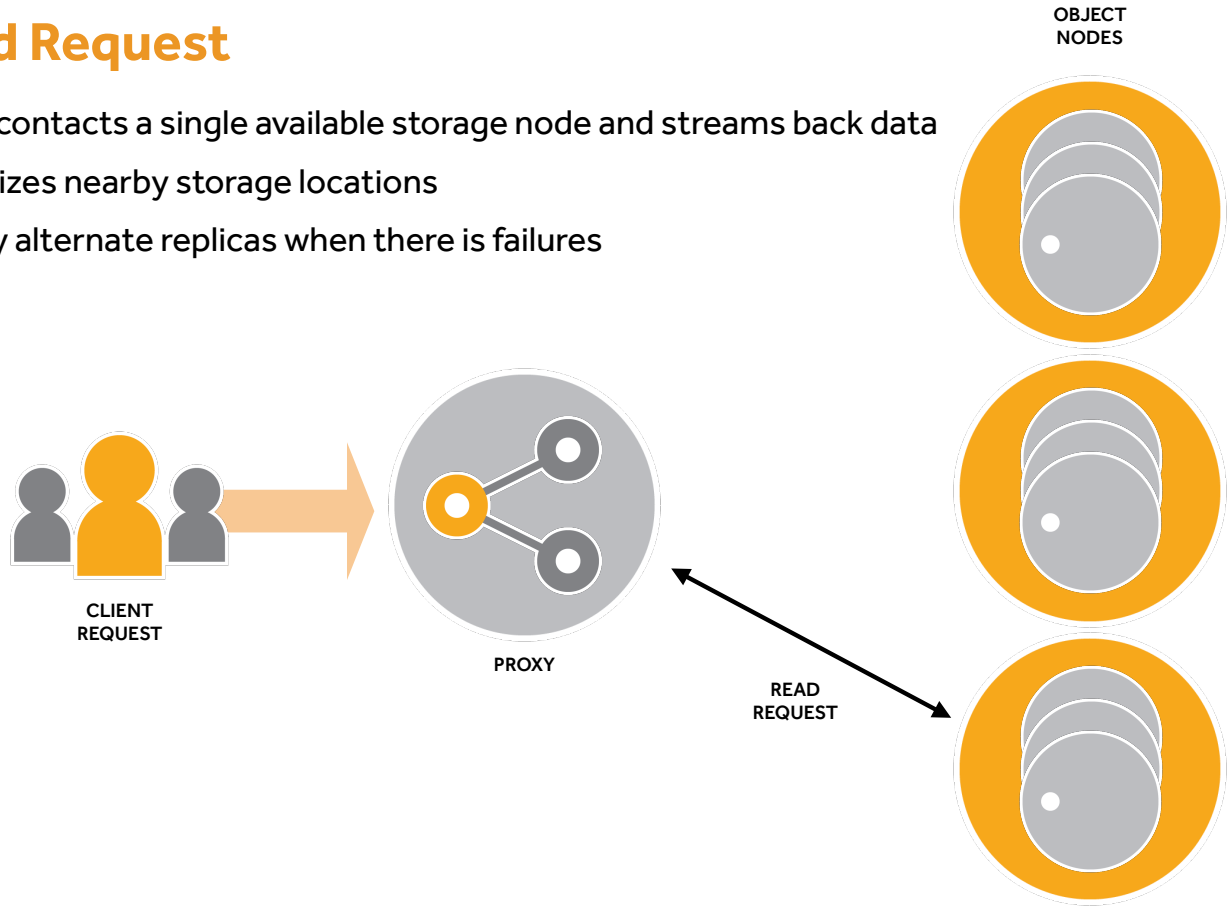
System Components: Proxy

Read Request

Proxy contacts a single available storage node and streams back data

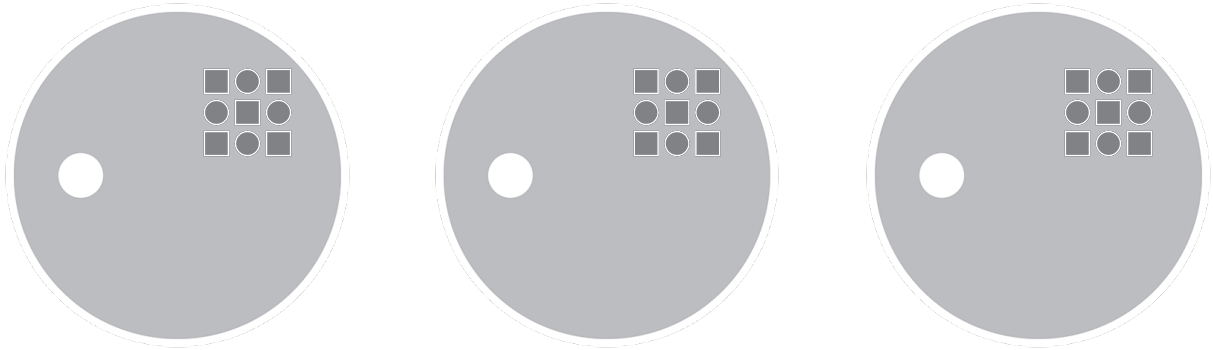
Prioritizes nearby storage locations

Will try alternate replicas when there is failures



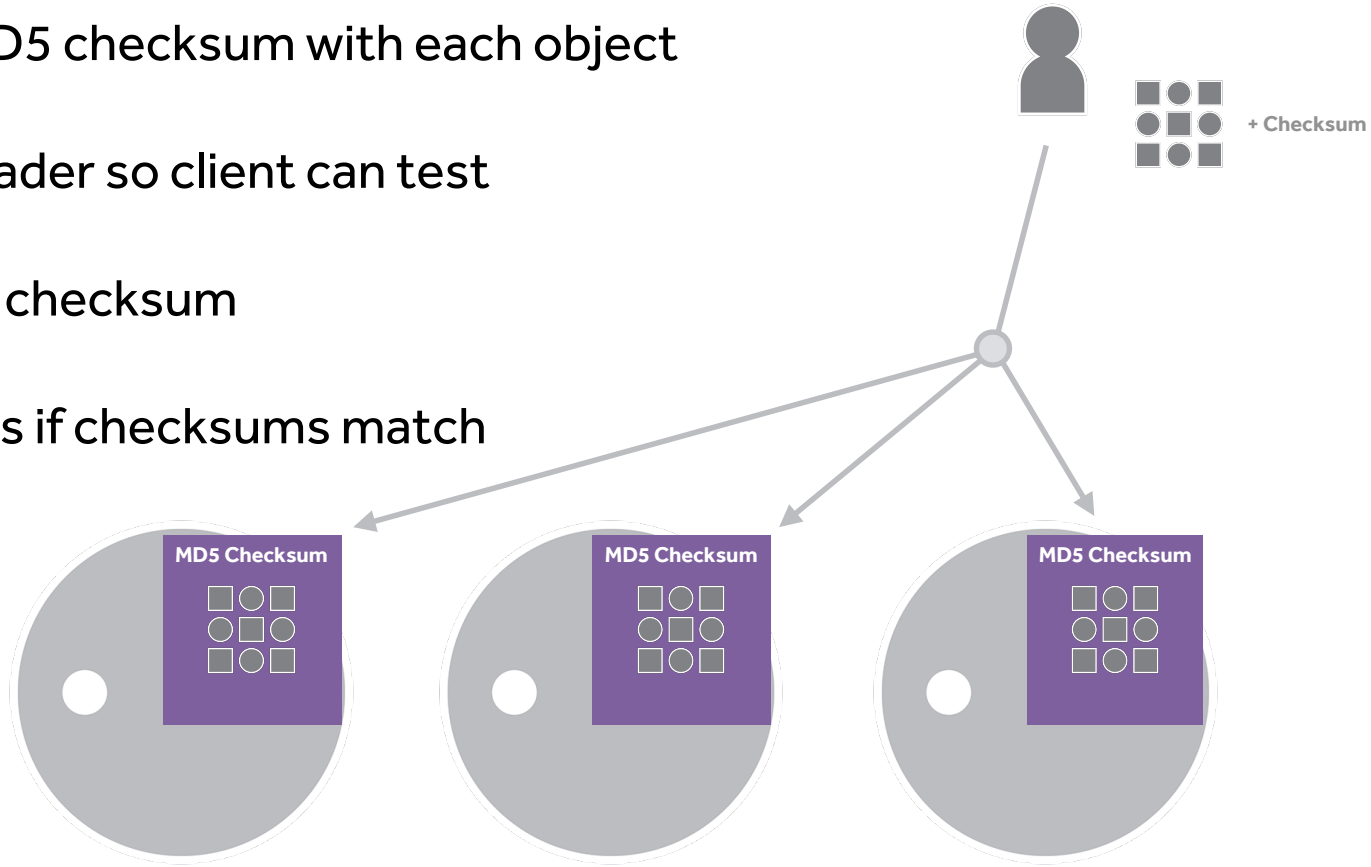
Durability with Replicas

- Swift stores multiple replicas to protect data
- 3 replicas provide a good balance between cost and durability guarantees
- The number of replicas is determined by the Ring



Durability with Checksums

- Swift stores MD5 checksum with each object
- Returned in header so client can test
- Uploads with a checksum
- Swift only saves if checksums match

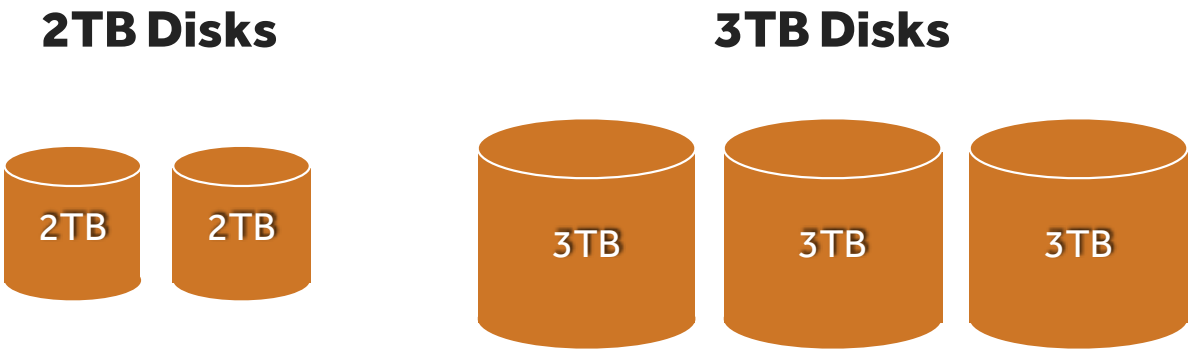


Swift Background Processes: Auditor & Replicator

- Active auditing and replication
- Asks other servers if they have a copy of this object
- Replicator will push object where missing

Disk Weights

Any size disk can be used in a cluster.



Weight: 2000

Weight: 3000

A weight is assigned to each disk.
Every individual disk's weight is relative to all other disks in the cluster.

Swift Partitions

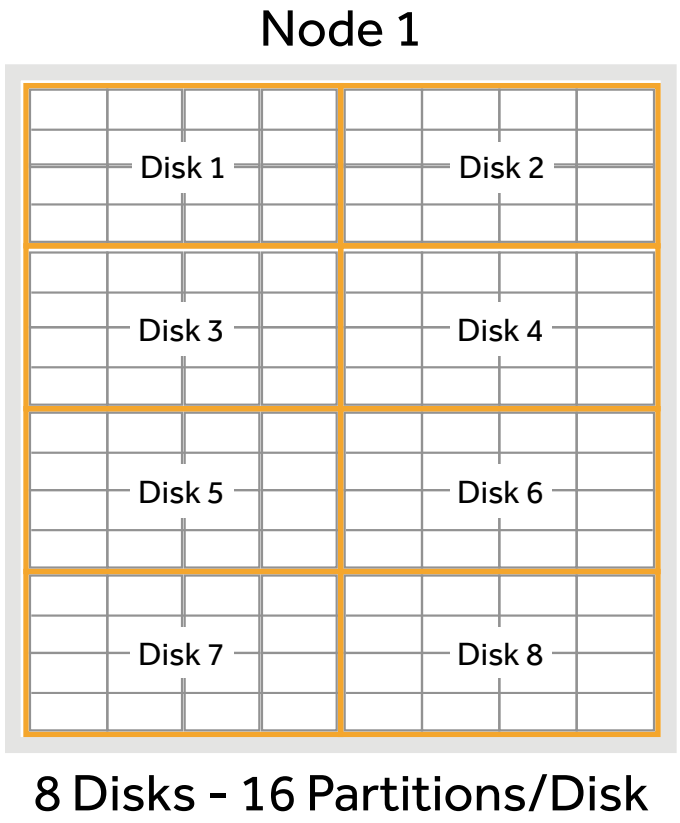
Swift partitions are **NOT**
your regular Linux disk partitions.

Swift Partitions: Directories on Disk

```
swiftstack@node1:/srv/node/d14/objects$ ls

100005  107626  115455  130228  140520  148705  157620  166997  176011  193357  202156
100009  10763   115459  130238  140533  148724  157636  167     176027  193370  202161
100014  107634  115471  130240  140535  148734  157642  167003  176044  193377  202165
100020  107636  115476  130268  140536  148743  15765   167005  176045  193382  202167
100028  107648  115479  13029   140541  148747  157664  167006  176051  193388  202169
100056  107656  11550   130296  140544  148749  157666  167008  176054  193390  202170
100071  107659  115504  130297  14055   148751  157671  167026  176055  193391  202177
100072  107663  115512  130302  140560  148752  157675  167033  176061  193395  202197
100084  107666  115513  13033   140593  148761  157678  167041  176065  193397  202199
100095  107667  115522  130330  140617  148762  15768   167042  176070  19340   202219
100103  10767   115528  130335  140622  148766  157691  167054  17608   193402  202223
100105  107675  115535  130336  140629  148785  157702  167074  176084  193428  202224
100114  107685  115537  130340  140631  148788  157707  167076  17609   193435  202228
100115  107692  11554   130348  140637  14879   157709  167086  176092  19344   202236
```

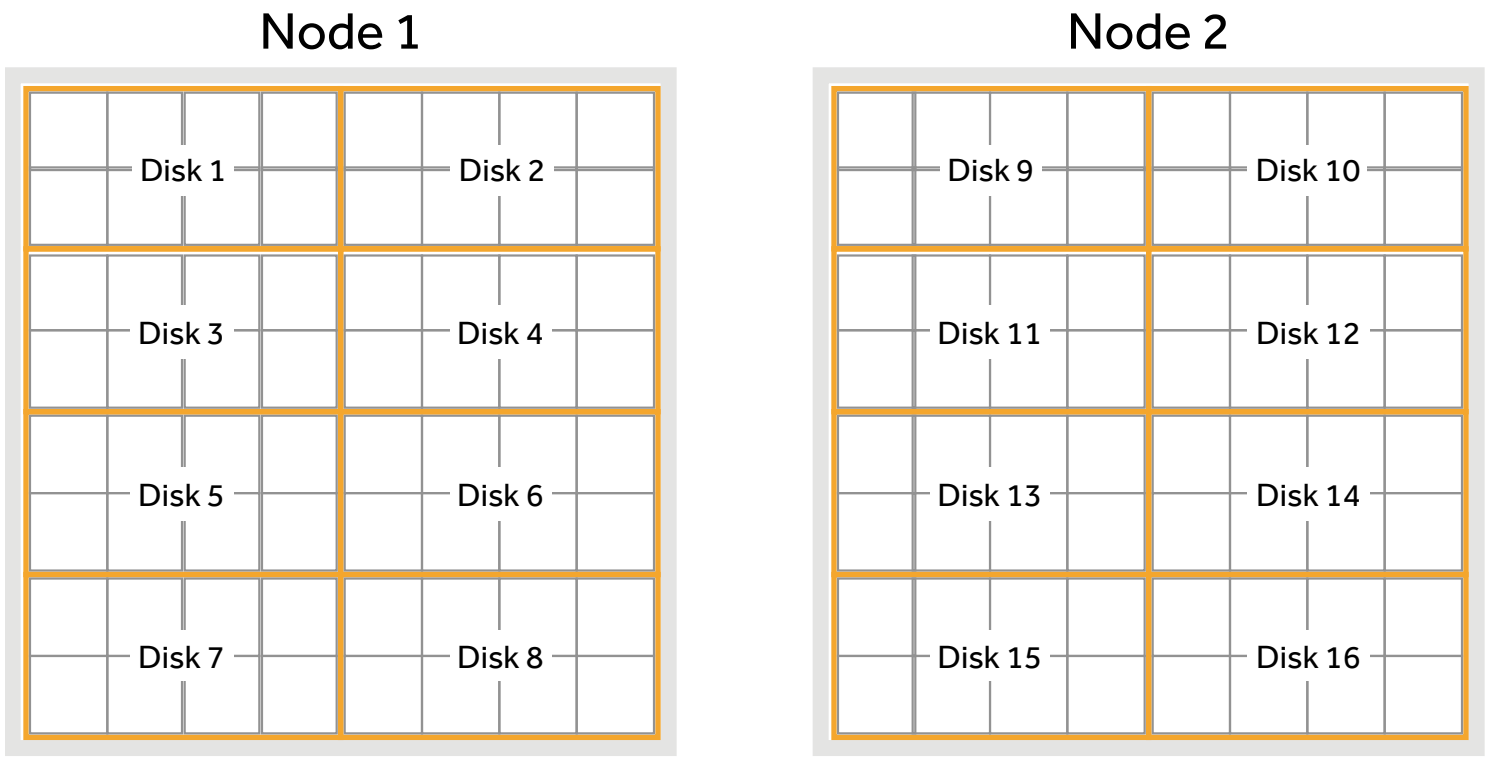
Swift Partitions - 1 Node



Example:
Assuming equally weighted disks.

8 * 16 = 128 partitions

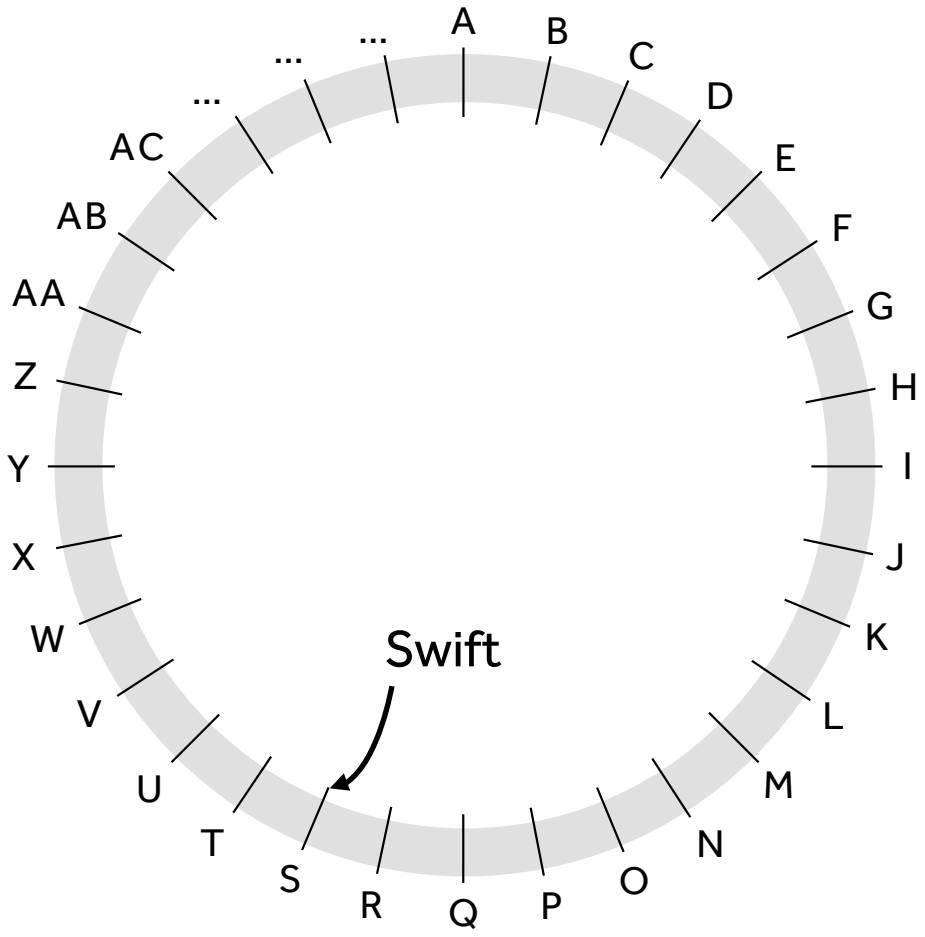
Swift Partitions - Adding A Node: Partitions Are Reassigned



8 Disks - 8 Partitions/Disk + 8 Disks - 8 Partitions/Disk

16 * 8 = 128 partitions

The Ring - Like An Encyclopedia



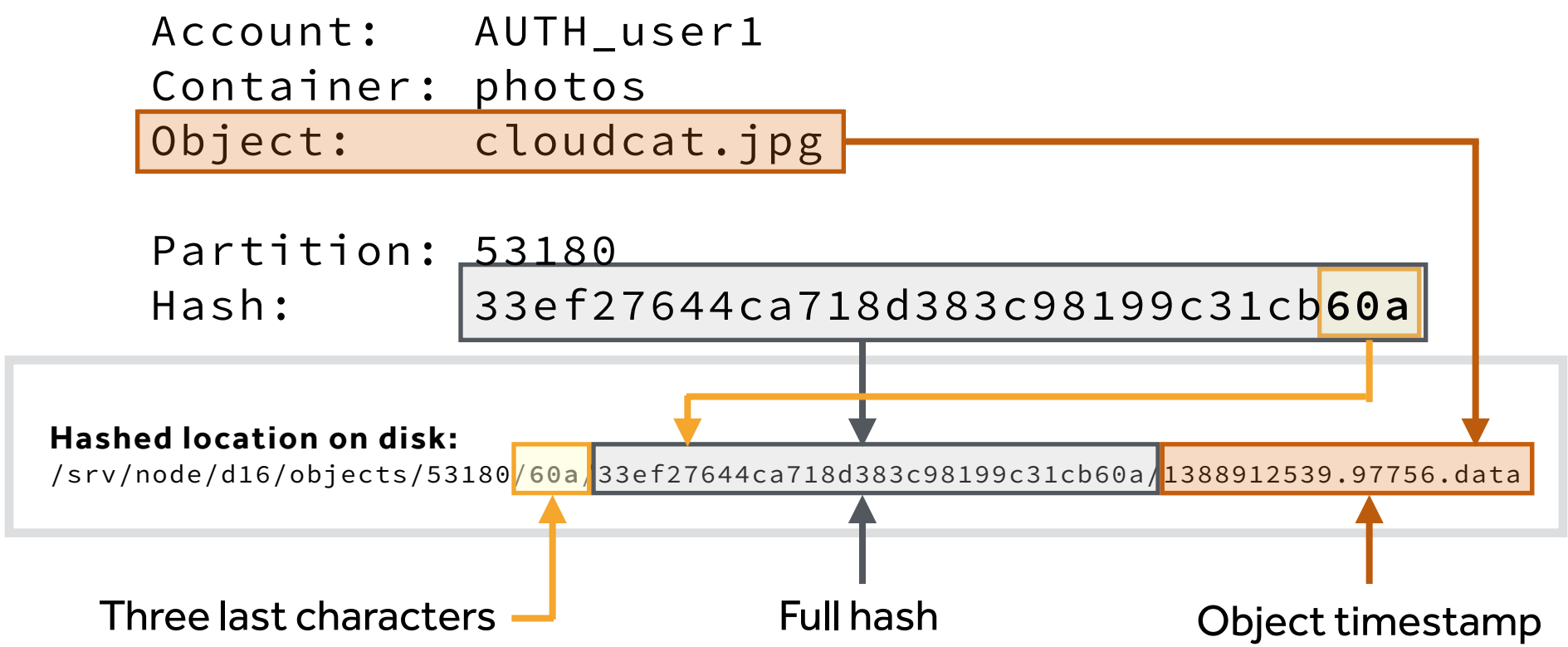
The Ring - Like An Encyclopedia

But instead of letters,

Swift uses hashes

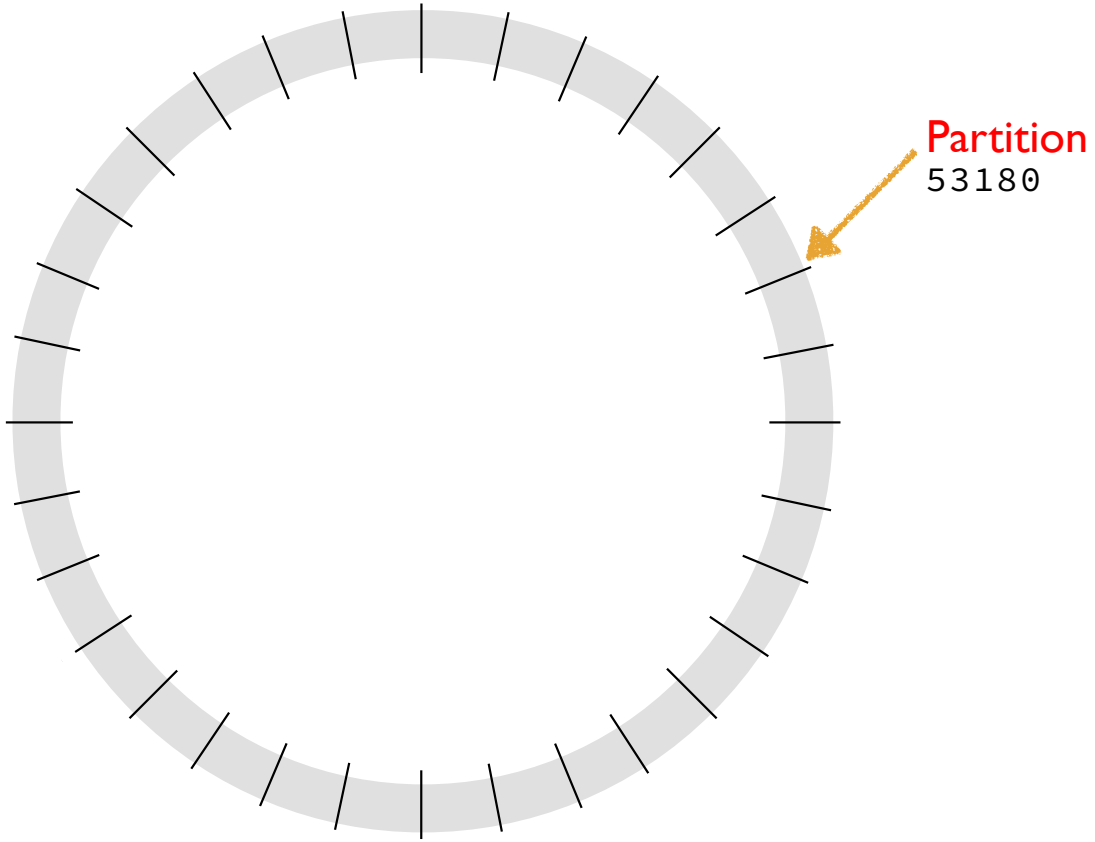
for each:

The Ring - Object Location Mapping



The Ring - Partition

```
/53180/60a/33ef27644ca718d383c98199c31cb60a/1388912539.97756.data
```



Ring Building Process

1 Number of replicas

Single Region Cluster Recommendation: 3

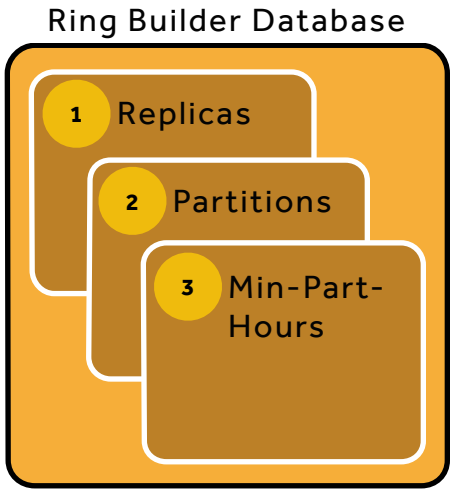
2 Number of partitions

How big will the cluster be?

Rule of thumb: 100 partitions * the number of drives that you think you will ever have, rounded up to the nearest power of 2

3 How quickly partitions can move

Min-Part-Hours: 24 hours default setting

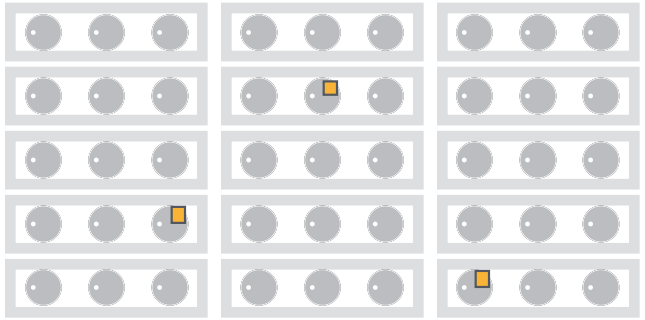


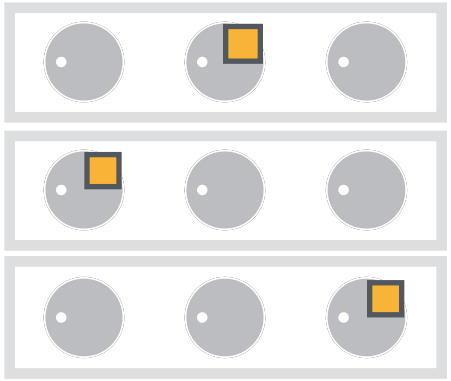
Ensures that only one replica is in flight

As Unique As Possible

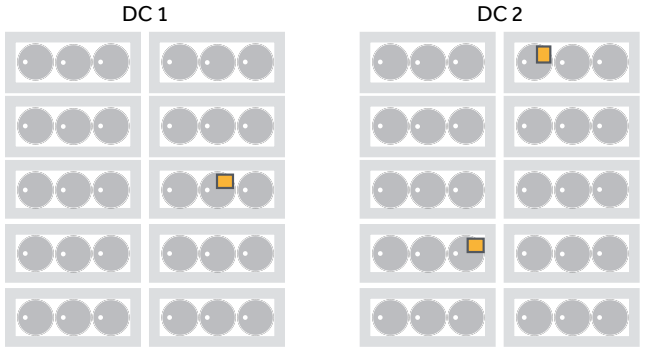
Single Node
Three Disks

Multiple Zones





Three Nodes



Multi-Region Cluster

Installing Swift - Lab

Manual Swift Installation (20 min)

Log in to Your Server

```
$ ssh demo@<your-vm-ip>
```

- See *** **Swift from CLI** *** card for IP address
- Password: `password`

Step 2: Format Devices

```
$ df -h
```

```
$ blkid -o list
```

```
$ sudo su -
# mkfs.xfs -f -i size=512 -L d1 /dev/mapper/v-xvdd
# mkfs.xfs -f -i size=512 -L d2 /dev/mapper/v-xvde
# mkfs.xfs -f -i size=512 -L d3 /dev/mapper/v-xvdf
# mkfs.xfs -f -i size=512 -L d4 /dev/mapper/v-xvdg
# mkfs.xfs -f -i size=512 -L d5 /dev/mapper/v-xvdh
```

```
$ blkid -o list
```

Step 3: Mount Drives

```
# mkdir -p /srv/node/d1
# mkdir -p /srv/node/d2
# mkdir -p /srv/node/d3
# mkdir -p /srv/node/d4
# mkdir -p /srv/node/d5
```

```
# mount -t xfs -L d1 /srv/node/d1
# mount -t xfs -L d2 /srv/node/d2
# mount -t xfs -L d3 /srv/node/d3
# mount -t xfs -L d4 /srv/node/d4
# mount -t xfs -L d5 /srv/node/d5
```

```
# chown -R swift:swift /srv/node
```

Step 4: Create The Builder Files

```
# cd /etc/swift
# swift-ring-builder account.builder create 14 3 1
# swift-ring-builder container.builder create 14 3 1
# swift-ring-builder object.builder create 14 3 1
```

```
# cd /etc/swift
# p=0
# for t in object container account; do
> for i in 1 2 3 4 5; do
> swift-ring-builder $t.builder add z$i-127.0.0.1:600$p/d$i 100
> done
> let p++
> done
```

Step 4: Continued ...

```
# swift-ring-builder account.builder
# swift-ring-builder container.builder
# swift-ring-builder object.builder
```

Step: Create The Rings

```
# cd /etc/swift
# swift-ring-builder account.builder rebalance
# swift-ring-builder container.builder rebalance
# swift-ring-builder object.builder rebalance
```

```
# ls *.ring.gz
```

Output should be:

```
account.ring.gz container.ring.gz object.ring.gz
```

Step 6: Start Swift

```
# swift-init main restart
```

```
# tail -f /var/log/swift/all.log
```

Step 7: Use The Swift CLI Client

```
# cd /home/demo
```

Upload the object `cloudcats.jpg` into the cats container:

```
# swift -U admin:admin -K admin \
  -A http://127.0.0.1/auth/v1.0 upload cats cloudcat.jpg
```

List files in the cats container:

```
# swift -U admin:admin -K admin \
  -A http://127.0.0.1/auth/v1.0 list cats
```

Download `cloudcats.jpg` from the cats container:

```
# swift -U admin:admin -K admin \
  -A http://127.0.0.1/auth/v1.0 download cats cloudcat.jpg
```

Title text

Step 8: Serving Data out of Swift

Make the cats container world readable:

```
# swift -U admin:admin -K admin \
  -A http://127.0.0.1/auth/v1.0/ post -r '.r:*' cats
```

To view the image, open your web browser and go to:

`http://<your-vm-ip>/v1/AUTH_admin/cats/cloudcat.jpg`

Installing Swift - Lab

SwiftStack Swift Installation (15 min)

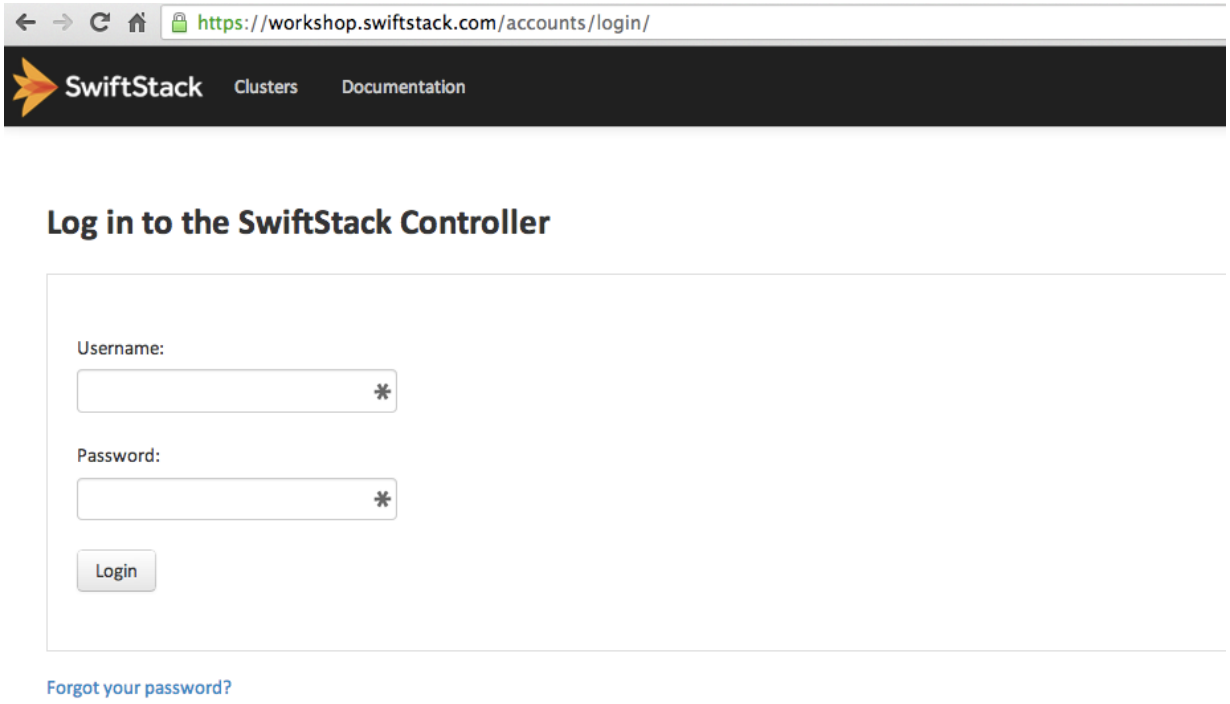
Log in to Your SwiftStack Node

```
$ ssh demo@<your-vm-ip>
```

- See SwiftStack handout for your specific IP address
- Password: `password`

Log in to SwiftStack Controller

https://try.swiftstack.com



Install Swift Using SwiftStack

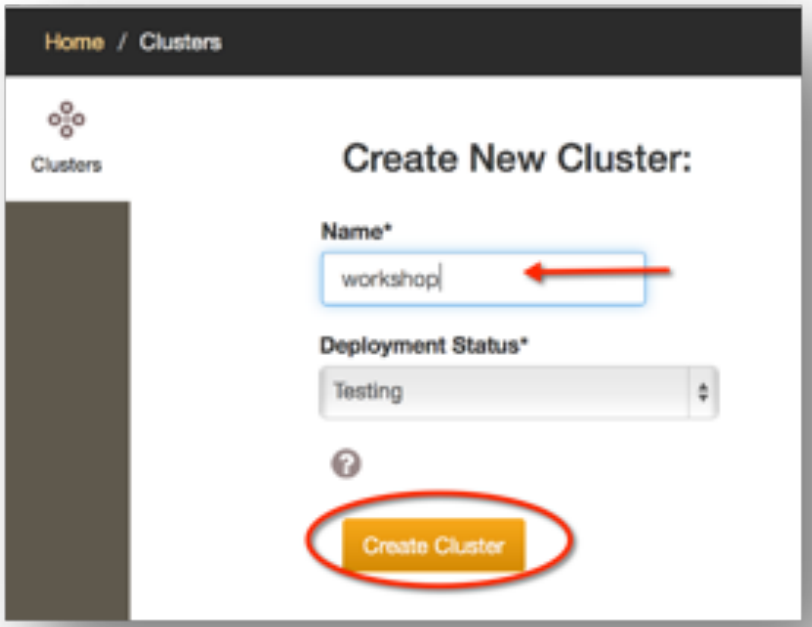
SwiftStack installation command:

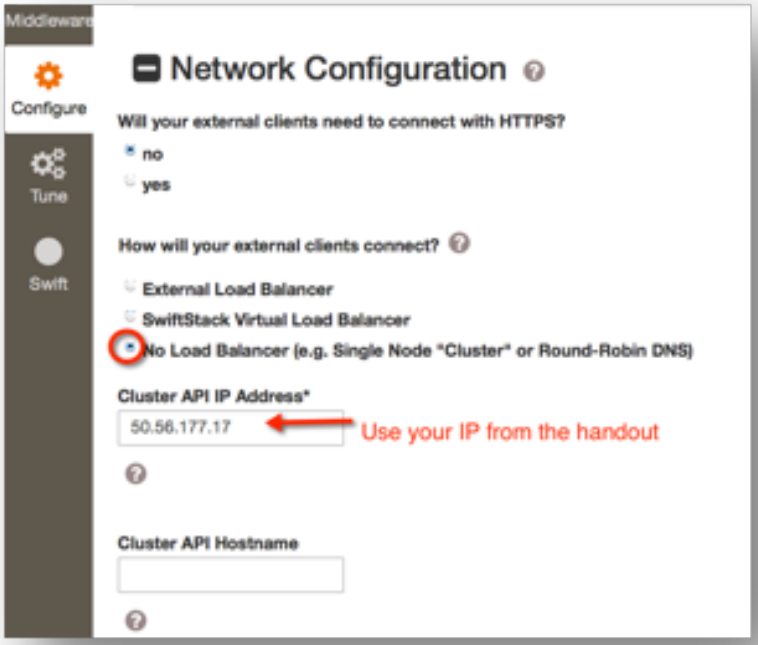
```
$ curl https://try.swiftstack.com/install_ubuntu | bash
```

Output should look similar to:

```
+-------------------------------------------------------------------------+
|                                                                         |
| Your claim URL is:                                                      |
| https://try.swiftstack.com/claim/09f7d921-4756-11e3-8016-bc764e04efd3 |
|                                                                         |
+-------------------------------------------------------------------------+
```

Create New Cluster & Configure





Deploying your single node cluster

Authentication

- SwiftStack Auth
- LDAP
- Keystone
- Active Directory

Integrations

- Load Balancing
- SSL
- CDN Integration
- Billing / Utilization
- Quotas
- CIFS / NFS Gateways
- File Managers & File System Adapters
- Backup

Upload An Object

Using the Swift Command Line Client, upload an object to the cluster:

```
$ cd /home/demo
$ swift -A localhost/auth/v1.0 -U user1 \
  -K password photos cloudcat.jpg
```

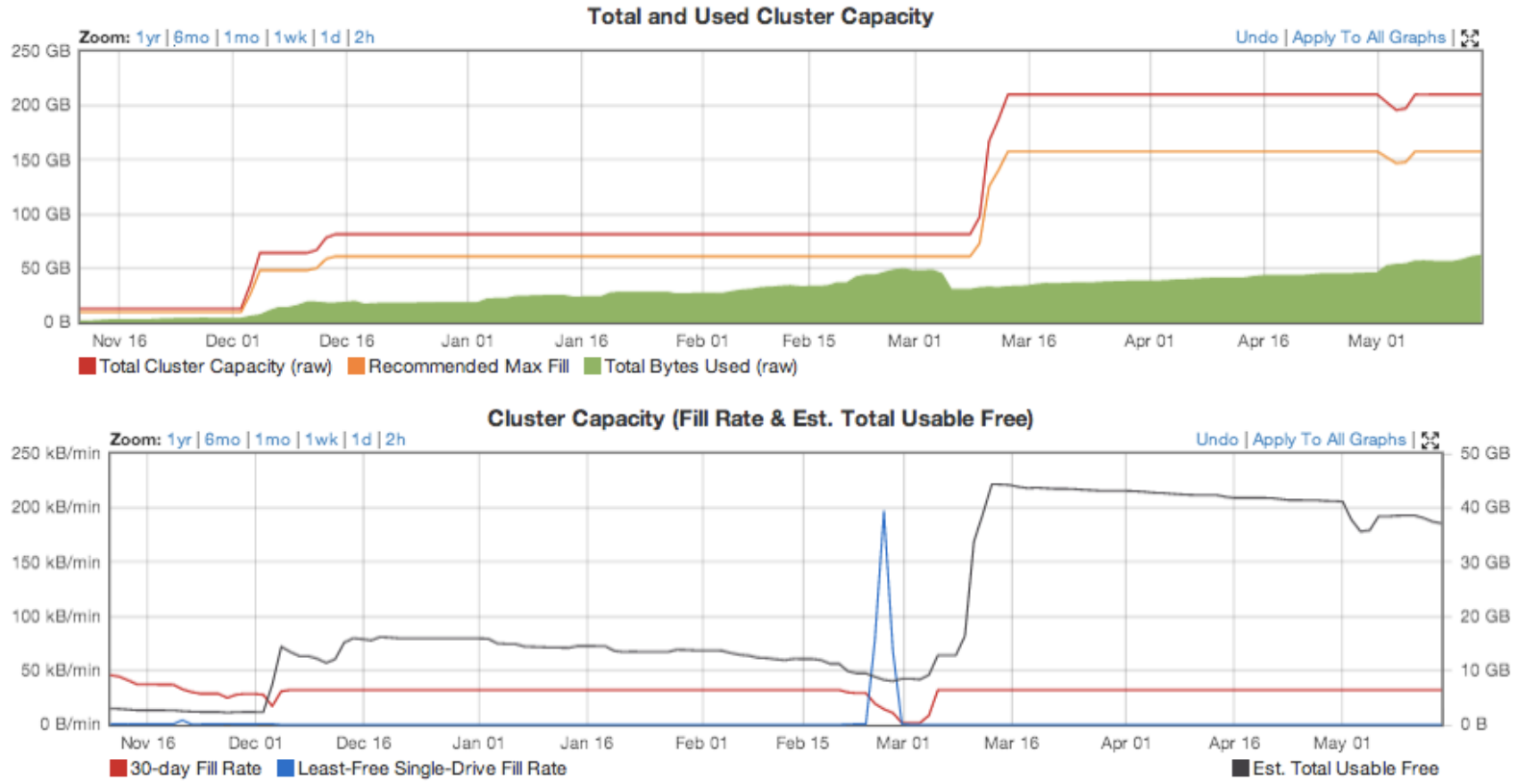
Find Where Objects Are

Using the swift-get-nodes command, find on which disks objects are located:

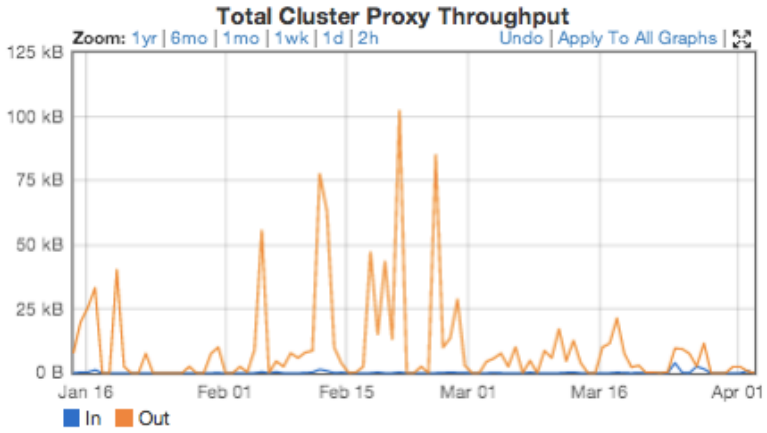
```
$ sudo /opt/ss/bin/swift-get-nodes \
  /etc/swift/object.ring.gz \
  AUTH_demo/photos/cloudcat.jpg
```

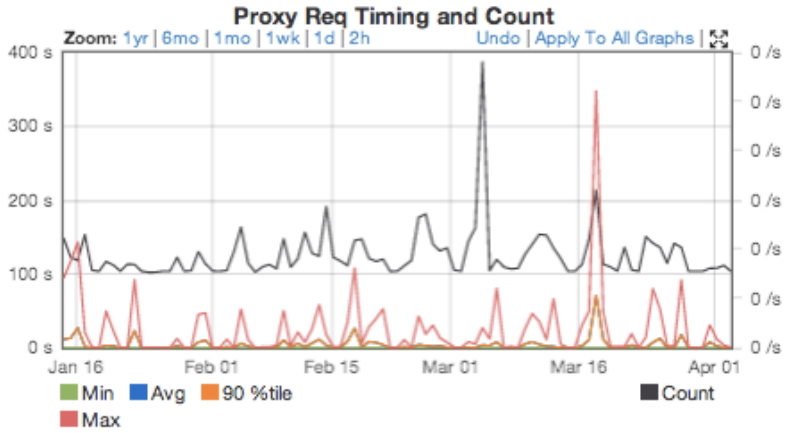
Operating, Managing and Monitoring Swift

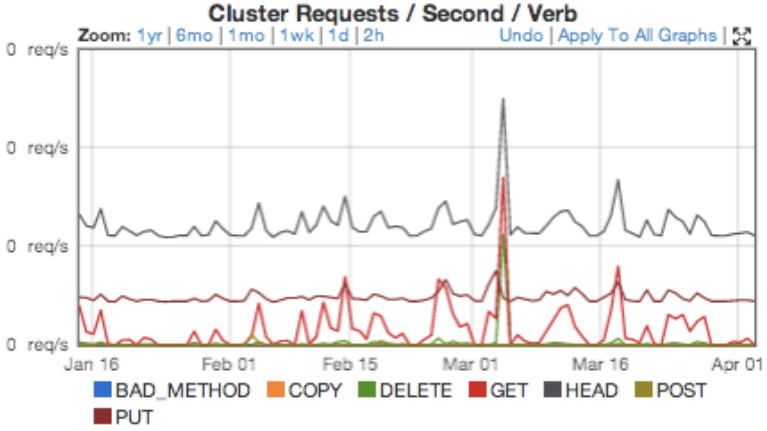
Capacity Adjustments & Monitoring

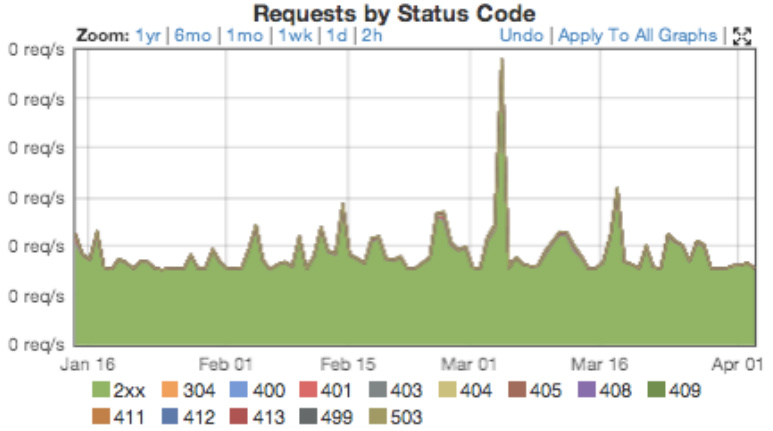


Swift Stats

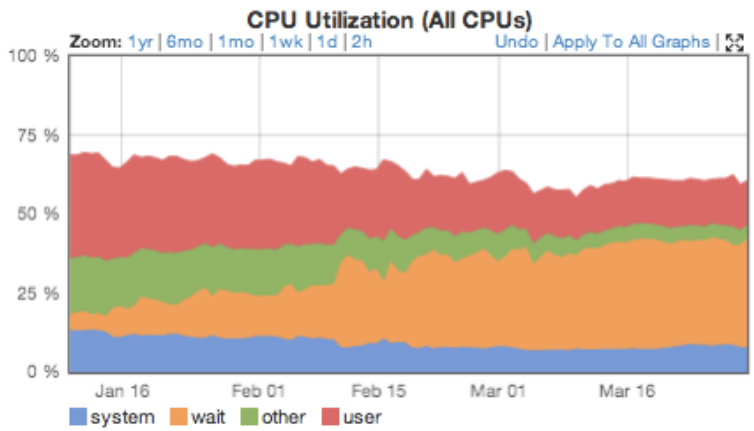


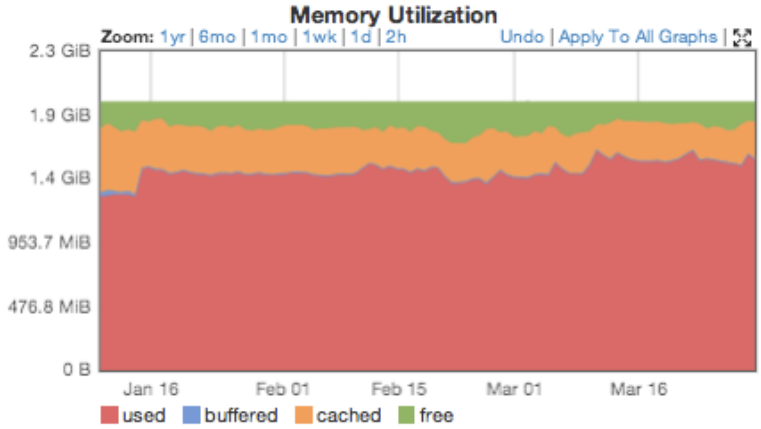


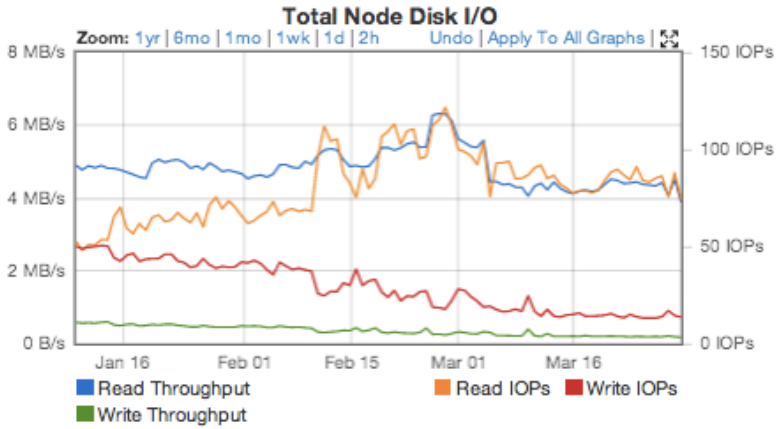


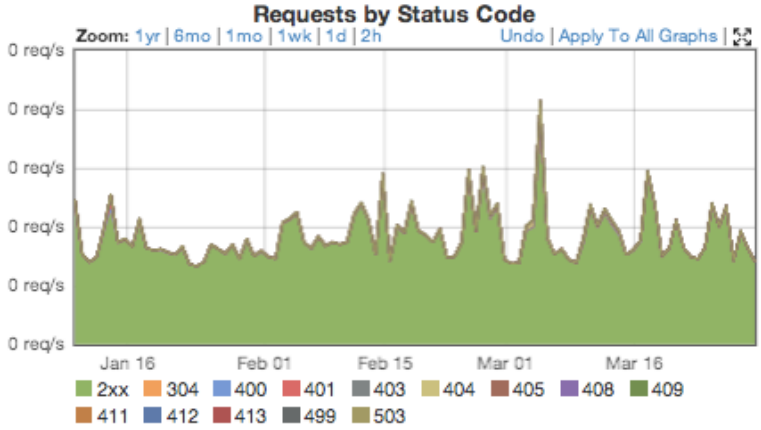


Node Stats



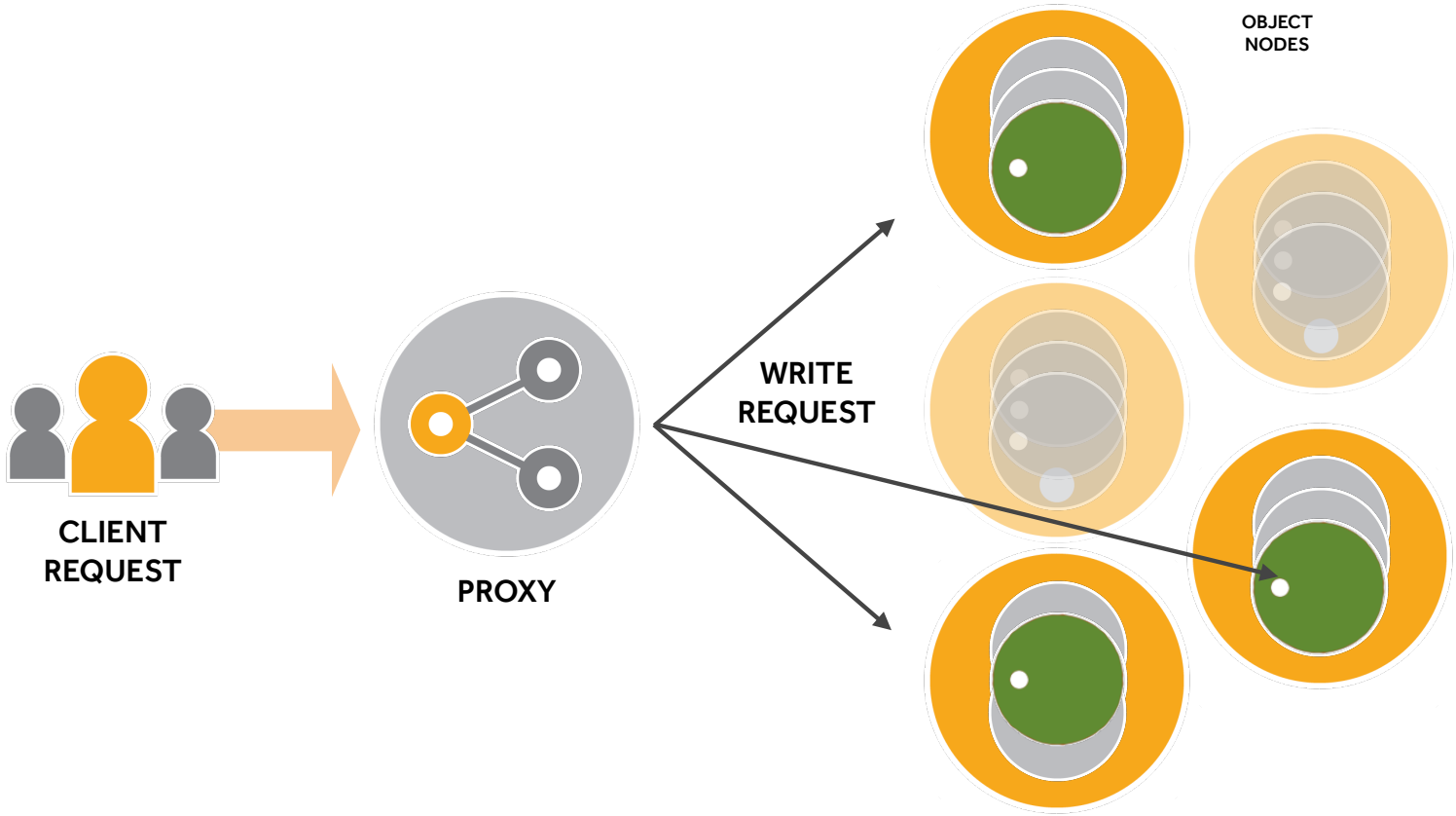






Failure Handing

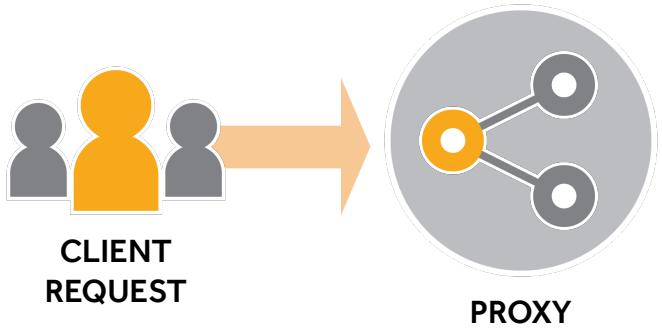
3 Replicas: Successfully written

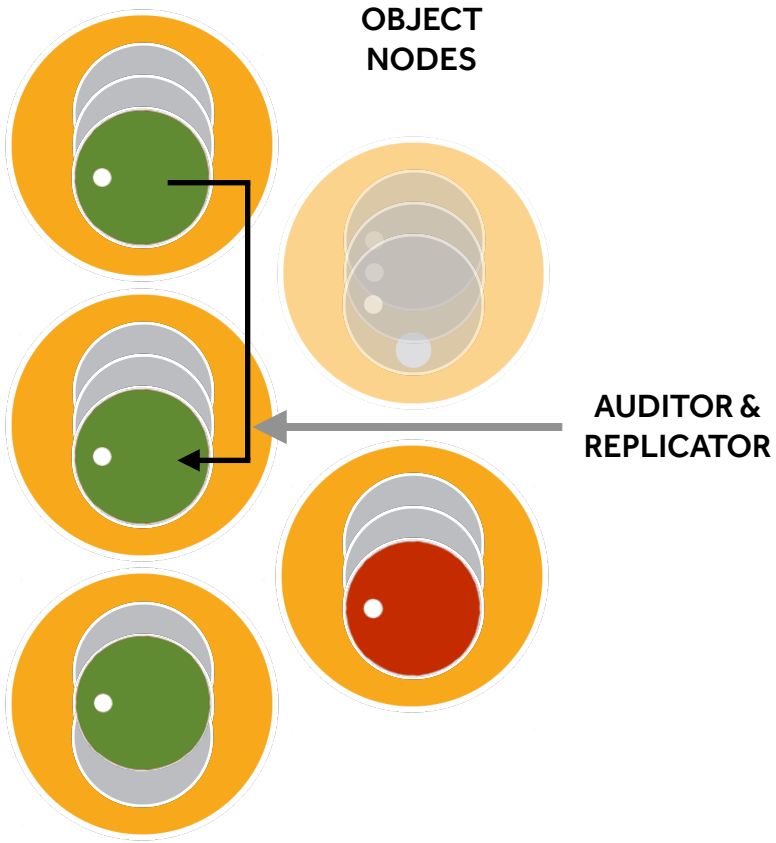


Failure Handling: Bad Disk

Handoff Locations

The replicators will proactively push replicas to handoff locations

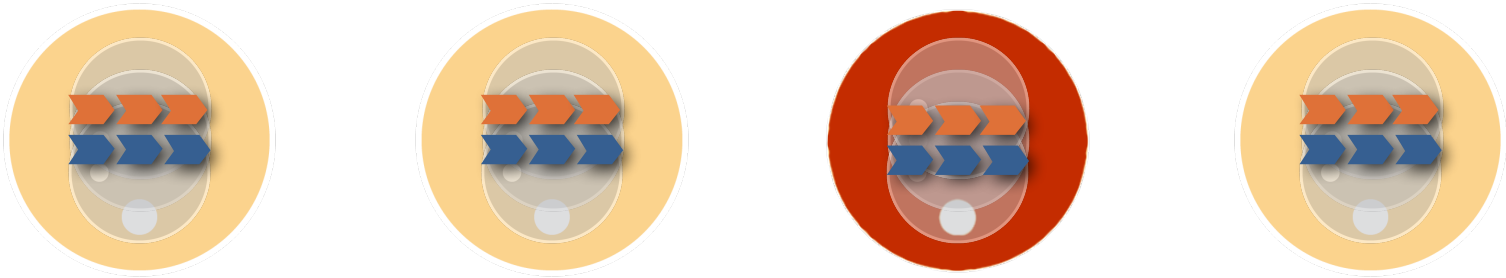




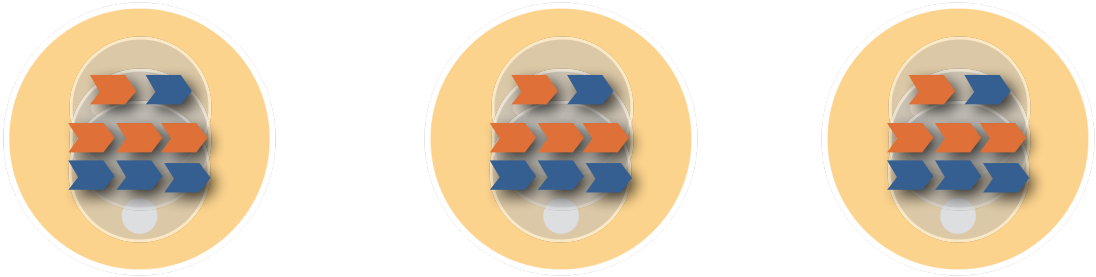
Failure Handling: Node Down

Replication works on Swift partitions on disks

Partitions, not drives or files are replicated during a recovery



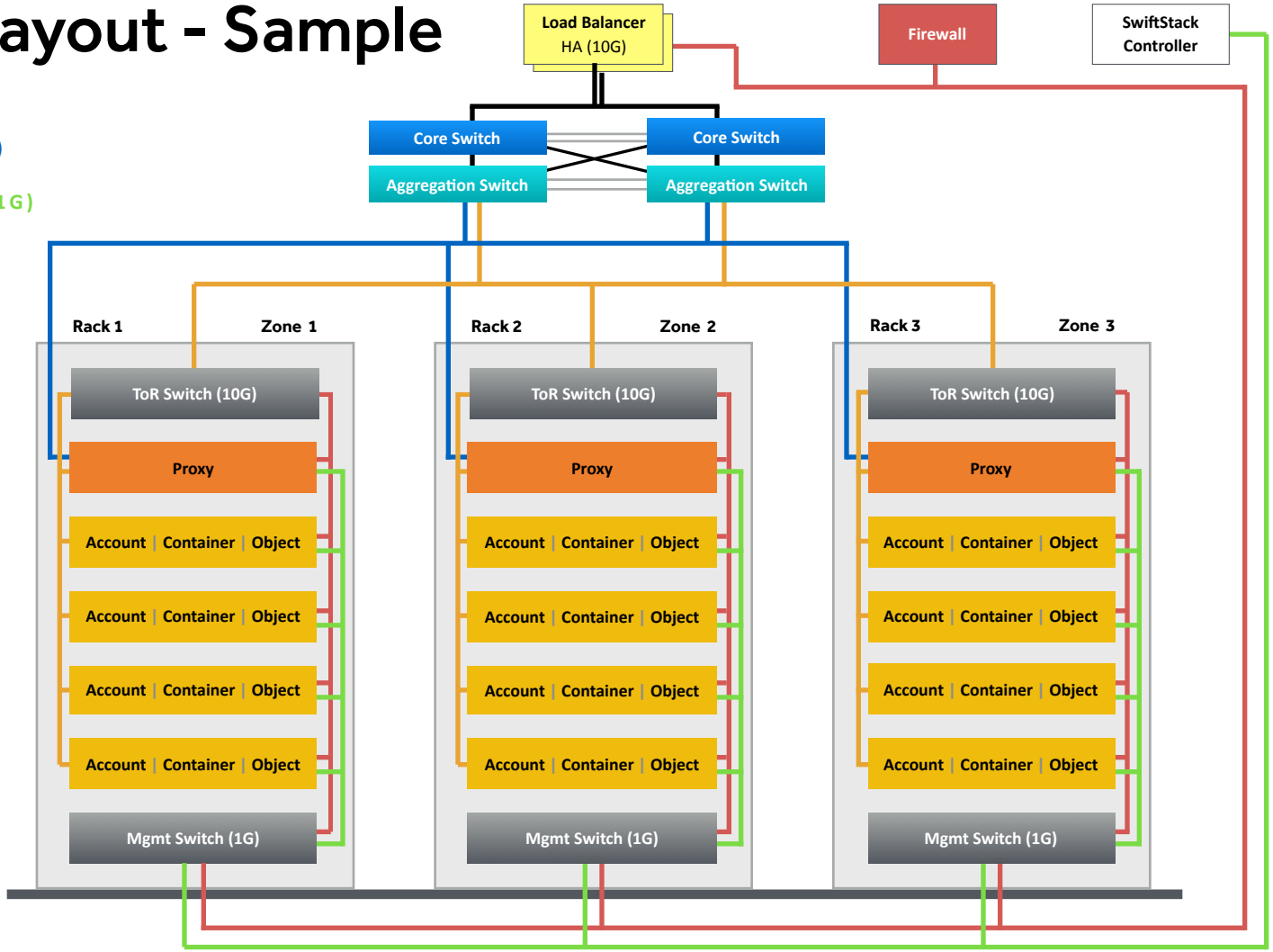
Replication will move partitions to other nodes



Cluster Topology Example

Cluster Layout - Sample

LEGEND
- Outward-facing (10G)
- Cluster-facing (10G)
- Controller network (1G)
- Out-of-band (1G)



Thank You!

SwiftStack