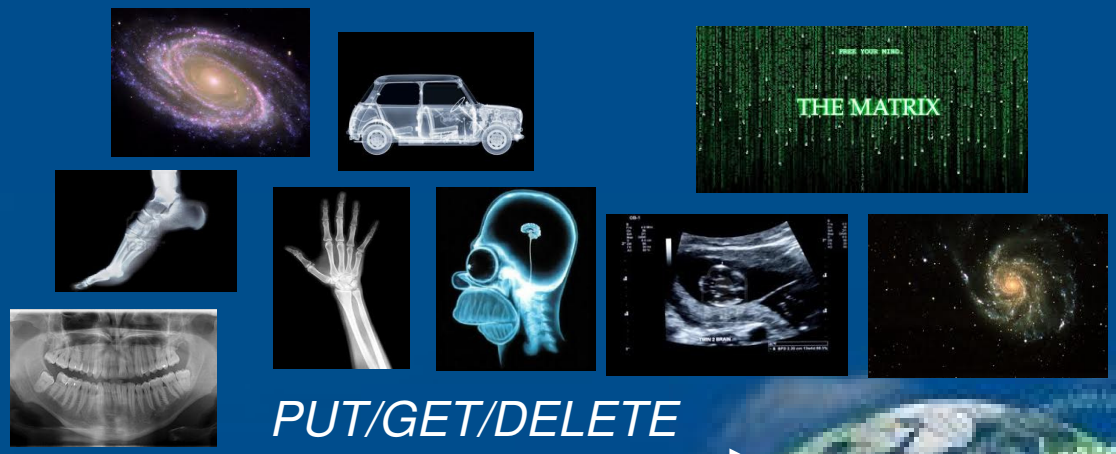

Openstack Swift
Object Store Cloud
built from the grounds up
David Hadas
Swift ATC
HRL
davidh@il.ibm.com

Object Store Cloud Services



PUT/GET/DELETE

Expectations:

- *Huge Capacity (Scale)*
- *Always Available + Safe*
- *High Performance*
- *More PUTs than GETs*
- *Low Cost per TB*

High Level: Object Store vs. File System

Object Store

- ***Interface:***
  - Web based: GET/PUT/DELETE
  - RESTful: Stateless
  - Metadata
- ***Synchronization***
  - Eventual Consistency
  - No Distributed Locking
- ***Software Defined Storage***
  - Commodity hardware
  - Designed to Fault but never Fail
  - Built to auto-recover by design
- ***Features***
  - Basic services that scale (KISS)
  - SW extendible with web interfaces

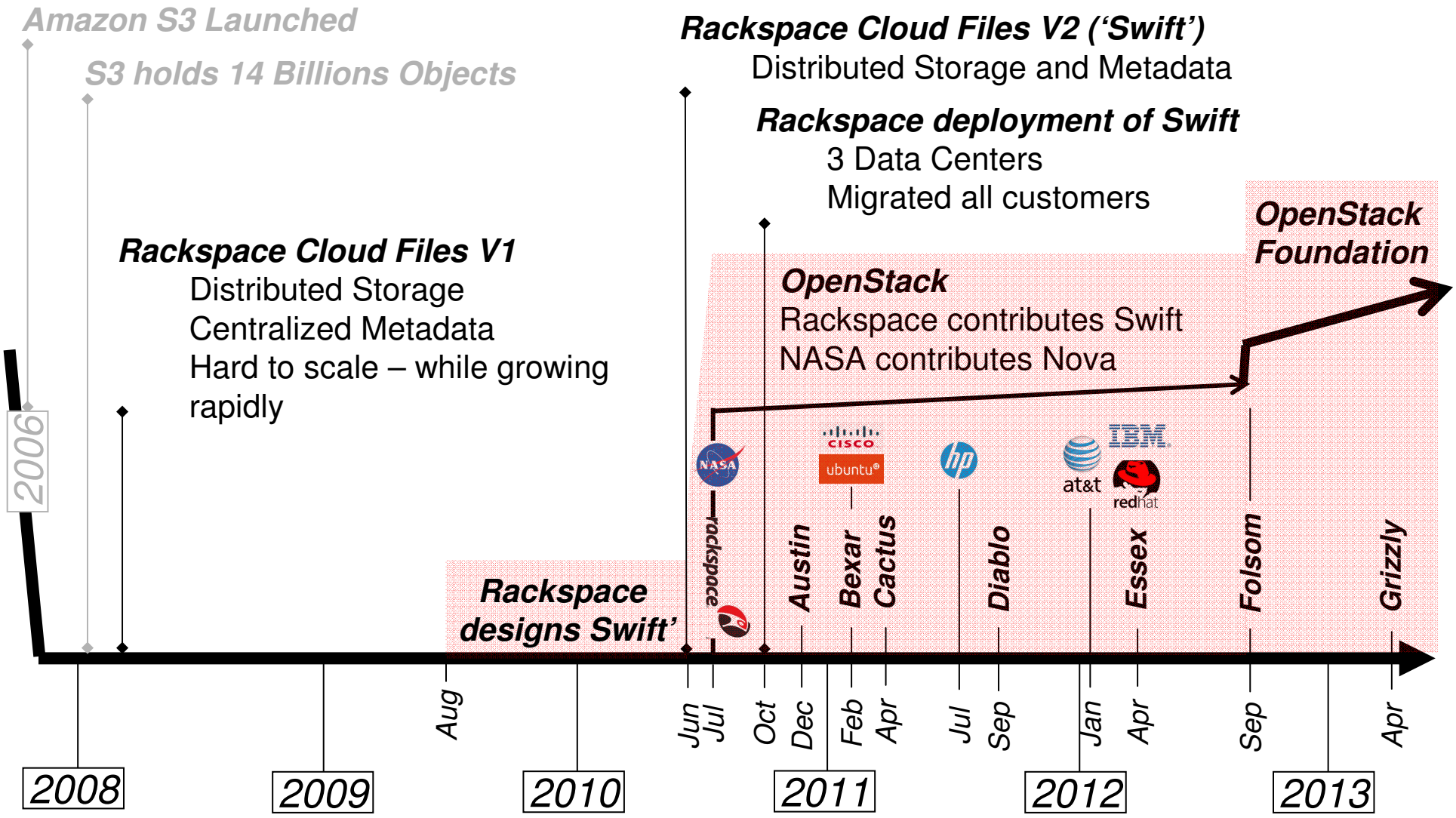
File System

- ***Interface:***
  - Posix:
    Open,Seek/Read/Write,Close
  - Stateful
- ***Synchronization***
  - Always consistent
  - Uses Distributed Locking
- ***Hardware and Software***
  - Best of breed hardware
  - Designed not to fault
  - Admin controlled recovery
- ***Features***
  - Abundant enterprise features built into the products

Object Store Cloud Options

- ***Amazon S3***
  - Proprietary API
- ***Rackspace Cloud Files (Swift)***
  - Swift API
- ***Microsoft Blob Storage***
  - Proprietary API
- ***Google Cloud Storage***
  - Proprietary API
- ***HP Cloud Object Store (Swift)***
  - Swift API
- ***OpenStack Object Services***
  - Written in Python
    - Apache license
  - Interfaces:
    - Native Swift 1.0
    - S3, CDMI interfaces

Historical Perspective



OpenStack - The Cloud operating System

- *550K lines of code*
- *300K Downloads*
- *550 Developers*
- *7000 Individuals from 100 Countries*
- *850 Organizations*
- *10M$ Funding*
- *3000 participants in the last summit*

Swift

A massively scalable redundant storage system

- Replicate objects to disk drives spread as far apart as possible

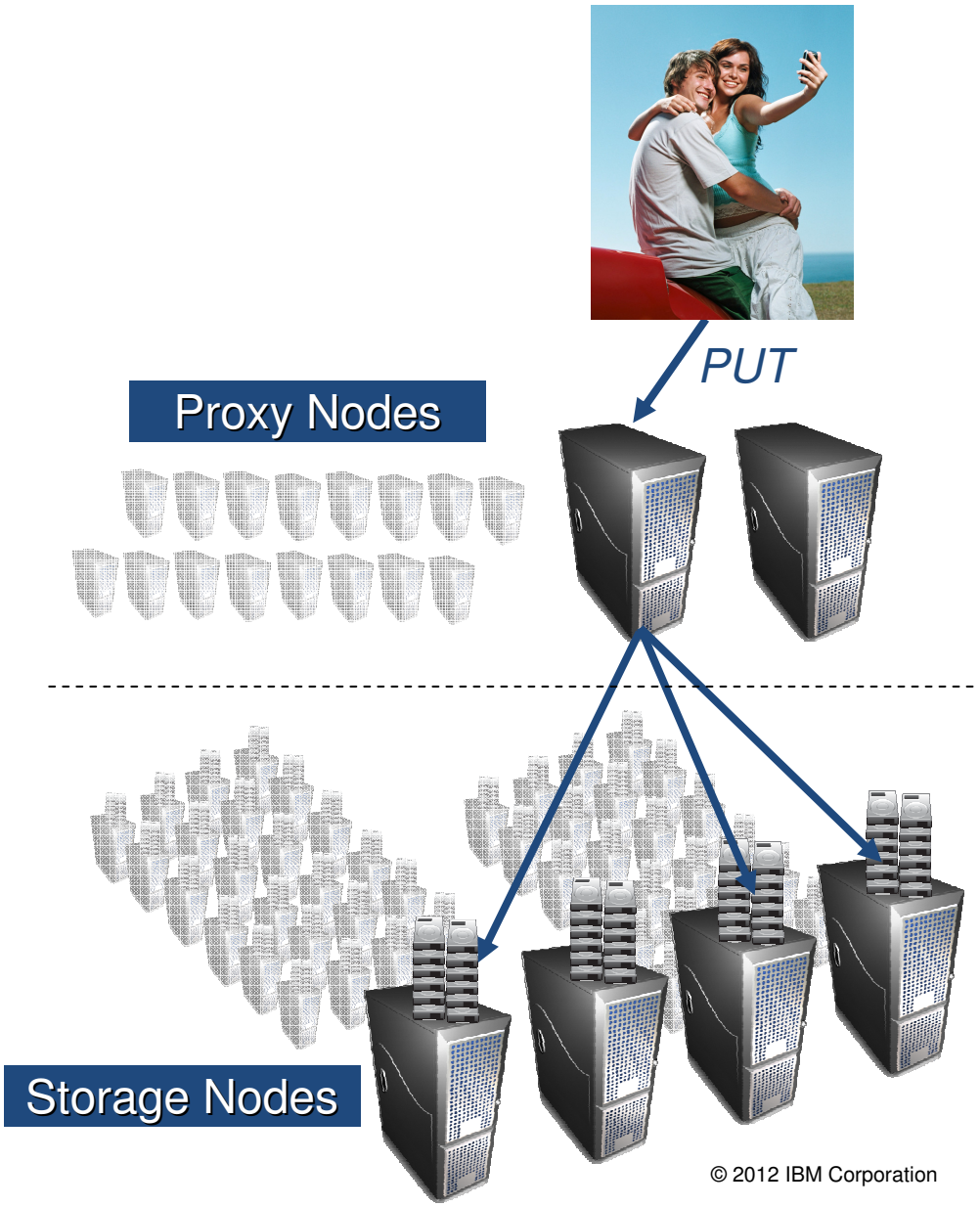
- ***Scales horizontally (adding new servers)***
  - Known to work with
    - 'Thousands of servers'
    - 'Petabytes of data'

- ***Designed to contain high rate of failures***
  - Inexpensive commodity hard drives and servers can be used
  - Maintains replication level following faults

- ***An Open Source Software***

Swift General Layout

- ***Swift Cluster***
  - Proxy Nodes
    - Smart
  - Storage Nodes
    - Simple
  - (Can be the same node)
- ***Object GET/PUT***
  - Web Clients use HTTP
  - Approach a Proxy Node
  - The Proxy node maps to storage devices
    - Located on Storage Nodes
  - Request is propagated to Storage Node
- ***A Cut-through technology***
  - Proxy and Storage Nodes never store the entire object in memory



Swift API Examples

- ***GET MyAccount/MyContainer/MyObject***
  - Responder provides the object in the body + metadata in http headers
- ***PUT MyAccount/MyContainer***
  - Requester indicates container metadata in http headers => Responder indicates 201
- ***POST MyAccount/MyContainer***
  - Requester indicates container metadata in http headers => Responder indicates 200
- ***PUT MyAccount/MyContainer/MyObject***
  - Requester indicates object content in the message body + metadata in http headers
  - Responder indicates 201
- ***GET MyAccount/MyContainer***
  - Responder provides the list of objects in the body + metadata in http headers
- ***DELETE MyAccount/MyContainer/MyObject***
  - Responder indicates 200

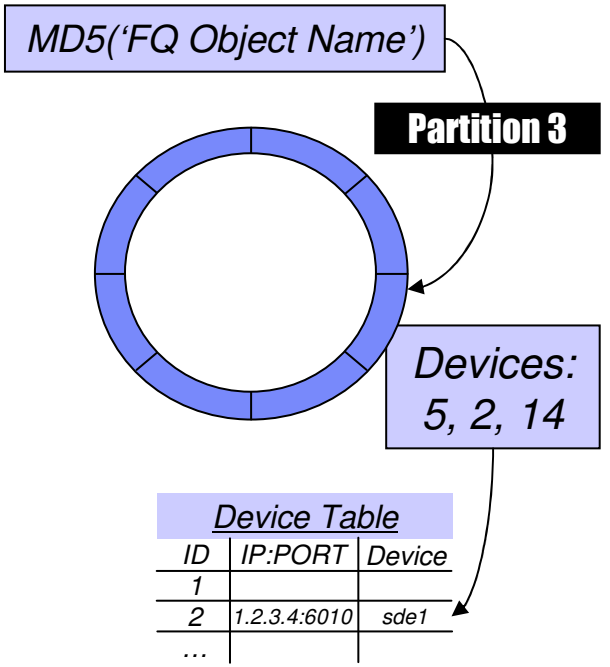
Principles of Operation - Mapping

- ***Mapping must allow us to stay:***
  - Distributed
  - Scalable
  - Load Balanced
  - Highly Available
- ***Fortunately, "All problems in computer science can be solved by another level of indirection (David Wheeler)"***
- ***Double mapping to allow virtualization of Disk space***
  - Objects are stored in virtual partitions
  - Partitions are stored in "devices"
    - Typically a device is a local hard drive in a server
    - Device can be anything
      - Allowing another level of indirection

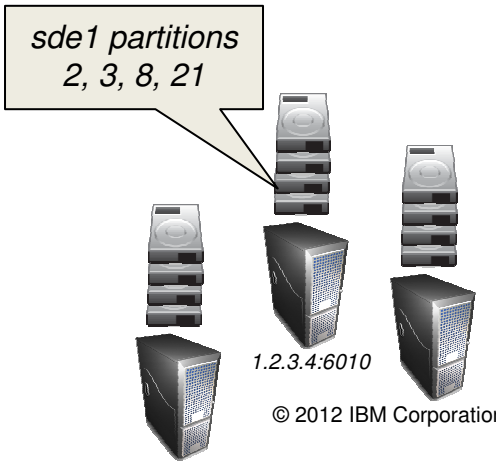
Mapping Implementation

- ***Consistent Hashing concepts***
  - Only K/n keys are remapped
  - Evenly distributed load during changes

- ***Mapping of Objects to Partitions***
  - Partition determined by MD5(Fully Qualified Object Name)
  - Uses Partition-Power highest bits of the MD5 result
  - Can elastically grow and shrink



- ***Mapping Partitions to devices***
  - Using a 'Ring' which maps each partition to D devices (i.e. D replications)
  - The Ring is consistent allowing addition of removal of devices
    - Used mainly to grow the cluster



Principles of Operation - Eventual Consistency

- ***Faults are assumed***
  - Disk faults
  - Server faults
- ***Service continues normally during faults***
  - Continue servicing both PUTs and GETs
  - Continue offering the same level of replication
- ***Distributed, on-going recovery***
  - Handoff replicas are created following lost of replica
- ***Always strive for consistency***
  - Replicas look for the way ‘back home’

Eventual Consistency Implementation

- ***PUT/POST/DELETE Object***
  Add/Update an object at D replicas
  - Respond after RoundDown(D/2 +1)
    - On failure:
      place on a handoff device
    - Any device may serve as a handoff for any object
  - Update the container
    - On failure: Queue in async_pending for later update
  - …Delete marks an object for deletion
- ***GET Object***
  Get one of the replicas
  - Would only try the D replicas at the Ring
- ***Replicator Daemon***
  - Crawl local devices
  - Replicate locally found replicas to its designated place
    - Delete successful and handoff
    - Signatures for objects + partitions used for efficiency and speed
- ***Updater Daemon***
  - Crawl local devices looking for async_pending
  - Updates container about an object
- ***Auditor Daemon***
  - Crawl local devices
  - Audit the object signatures
- ***Expirer Daemon***
  - Deletes cold objects

Principles of Operation - Containers and Accounts

- ***Containers and Accounts are Objects***
  - The content of the object is the list of items it contains
  - GET /myaccount/mycontainer retrieves the list of objects
  - PUT /myaccount/mycontainer creates a container
  - PUT /myaccount/mycontainer/myobject adds an object to a container

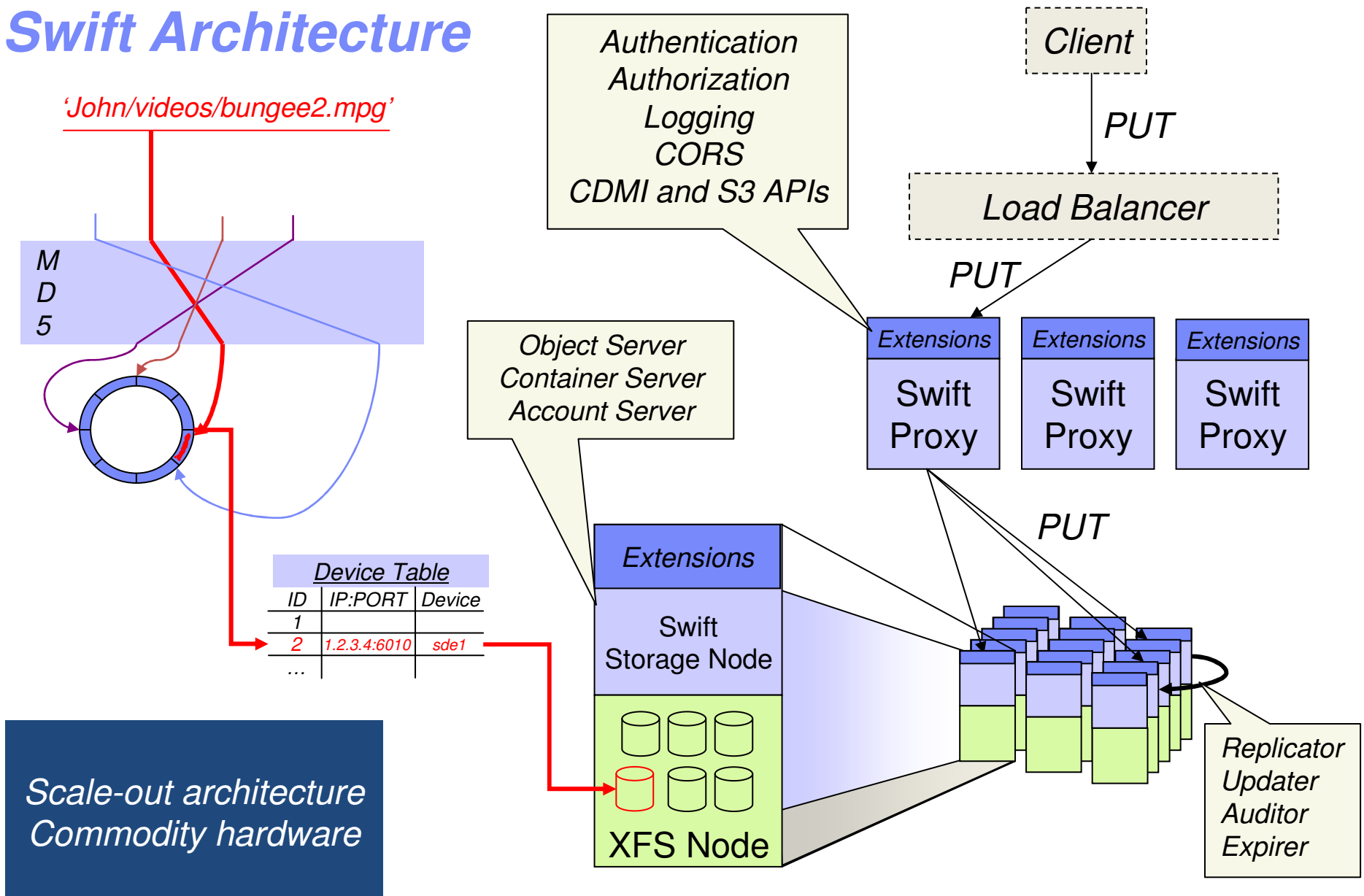
Containers and Accounts - Implementation

- ***The Container/Account is an SQLite3 DB.***
  - Today used for

- ***Separate Rings***
  - Allow store on separate devices (SSDs for containers/accounts):
    - Objects
    - Containers
    - Accounts

- ***Containers and Accounts have their own set of eventual consistency daemons:***
  - Replicators
  - Updaters
  - Auditors
  - Expirer

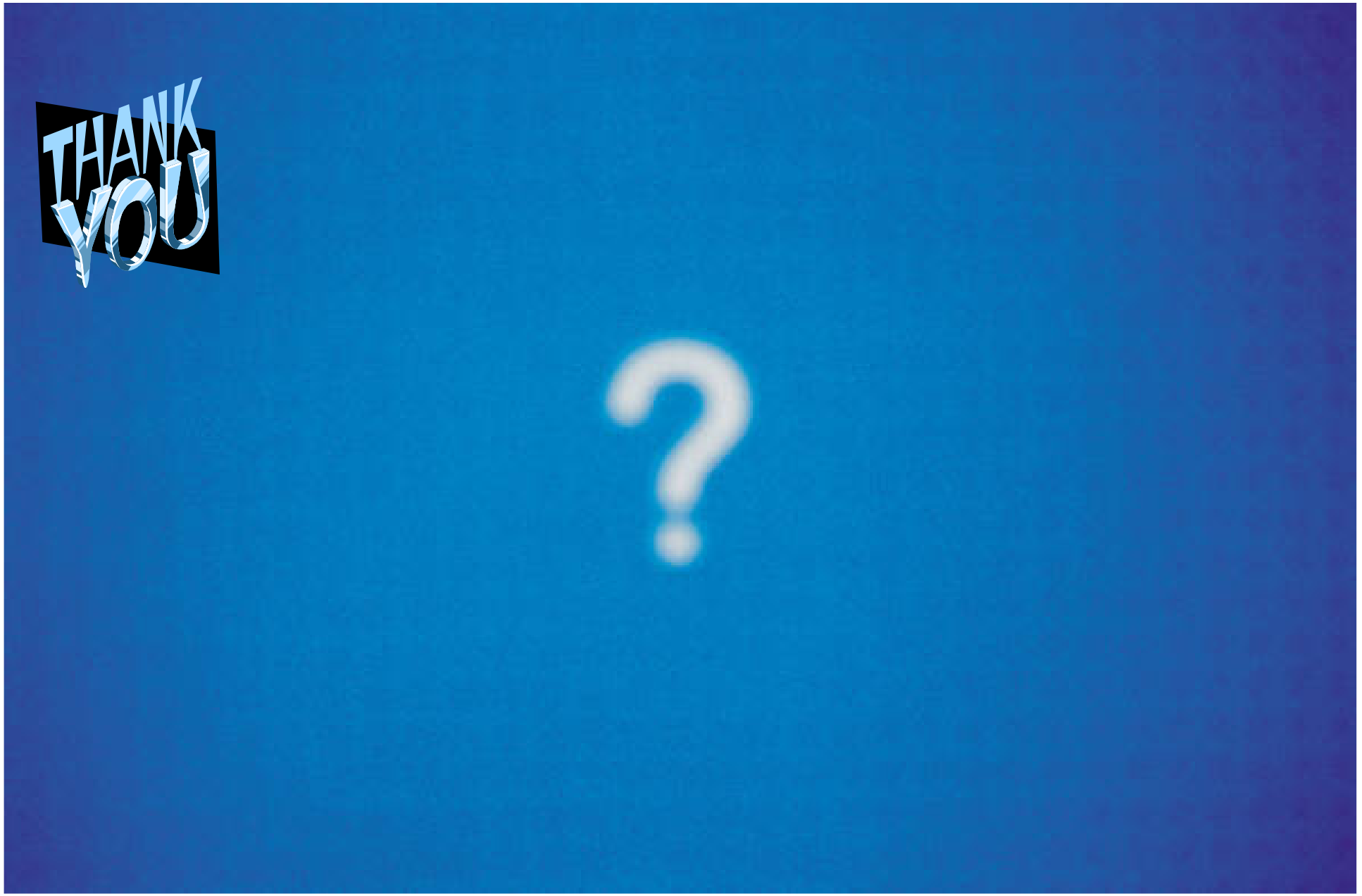
Swift Architecture



Scale-out architecture
Commodity hardware

IBM Participation – Create Enterprise Grade Swift

- ***Contributions to Open Source***
  - Done:
    - CDMI Support
    - Swift as a request processor of Apache
  - On going:
    - Account ACLs
    - Account Listing
    - Enable Ring Size to Grow
    - Swift Crawler
  - Future
    - Object ACLs
    - Metadata Extendibility
    - Undelete


THANK YOU