Cloud Storage

Cloud Data Centers….

Early days…

…today

Observations….

- Pervasive failure of system components
  - Epically if you want to use cheap commodity stuff
- Emerging workloads on very large files
  - “Big data,”
  - Multi-gigabyte files, terabyte data sets, much larger then traditional many small file design points
- Random writes are rare
  - Updates are frequently appends
  - More reads then writes, often sequential
- Co-designing application and file system interfaces can have significant benefits
  - E.g. alternative consistency models

Design points

- Use inexpensive commodity parts
  - No expensive disks or controllers
- Small number of large files
  - A few million files, but 100Mb or larger
- Workload consisting of
  - mostly of large streaming reads (100s of KB) and small random reads (few KBs).
  - Large, sequential writes that append
  - Random writes rare
- Many concurrent operations
  - E.g. parallel processing
- Bandwidth is more important then latency

Amazon S3

- Storage on the “cloud”
- Object store with web service interfaces hosted by Amazon

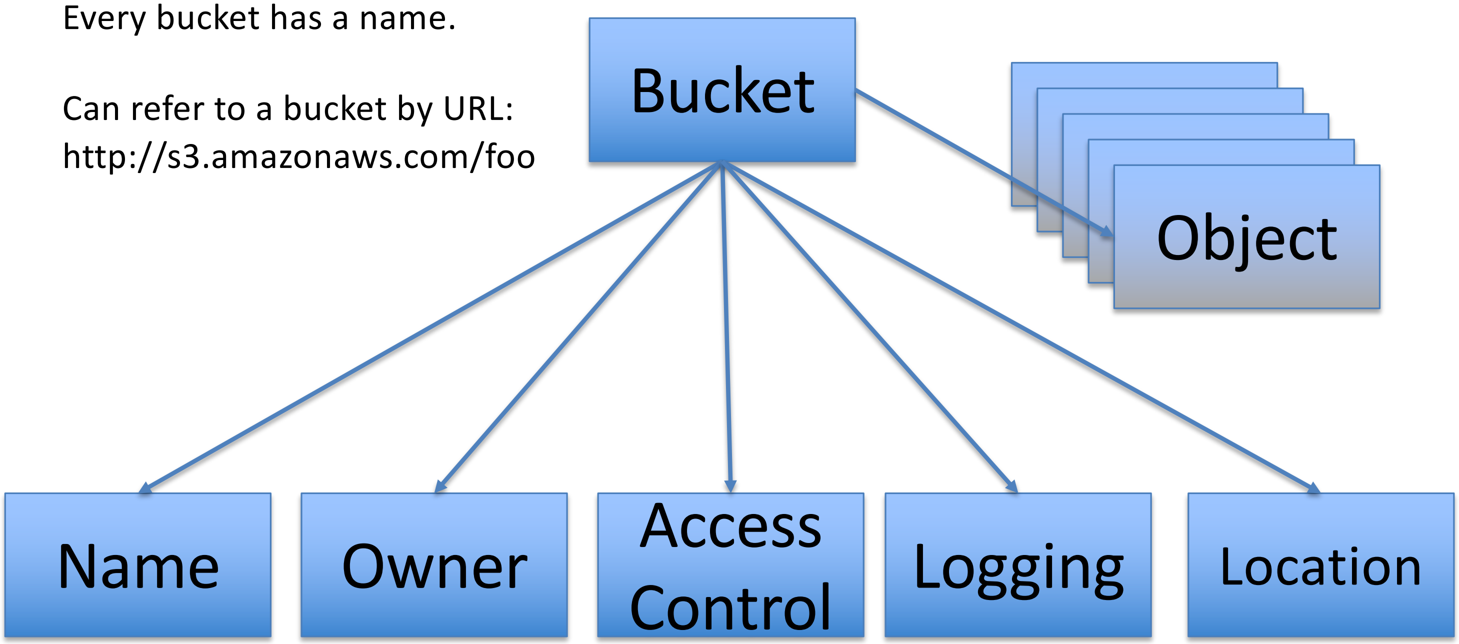
The Simple Storage Service

- **Buckets**
  - Container for objects stored in Amazon S3
- **Objects**
  - Data + Metadata
  - Metadata consists of name/value pairs
- **Keys**
  - A key is the unique identifier for an object within a bucket

Buckets

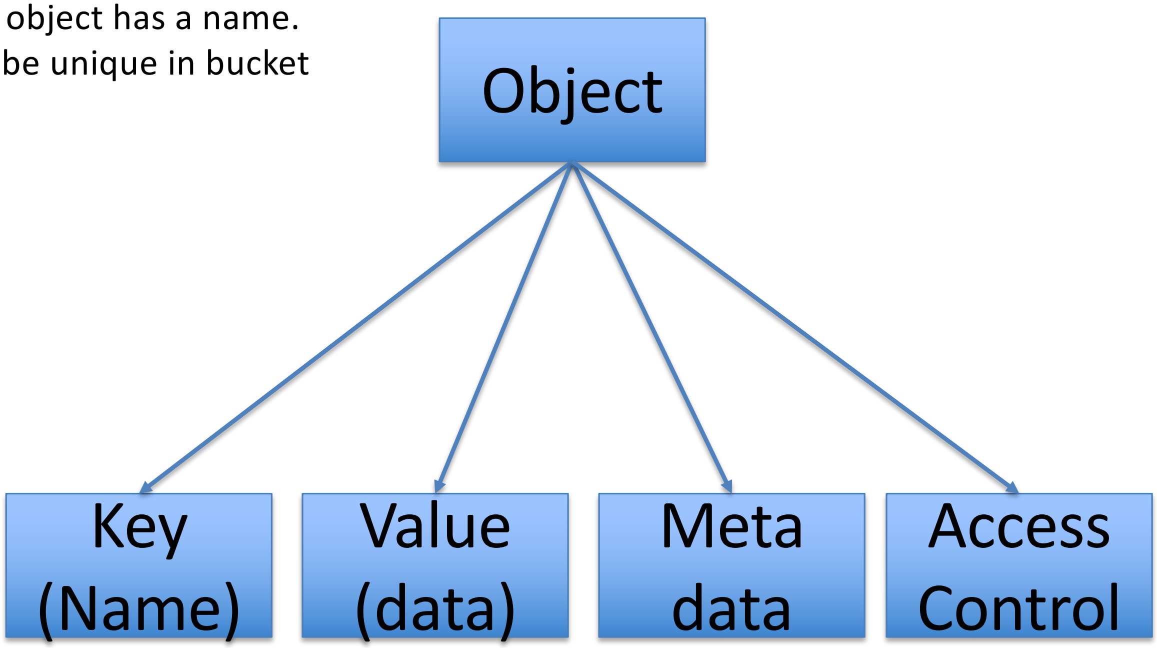
Every bucket has a name.

Can refer to a bucket by URL:
http://s3.amazonaws.com/foo



Objects

Every object has a name.
Must be unique in bucket



Operations on Objects

- Object consists of
  - Key: the object name used to retrieve the object
  - Version ID
  - Value
  - Metadata
- Can store, retrieve and delete objects
  - Retrieve done on whole object
- Objects can be versioned

Representational State Transfer (REST)

- Client-Server
  - a pull-based interaction style: consuming components pull representations.
- Stateless
  - each request from client to server must contain all the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- Cachable
  - to improve network efficiency responses must be capable of being labeled as cacheable or non-cacheable.
- Uniform interface: all resources are accessed with a generic interface
  - (e.g., HTTP GET, POST, PUT, DELETE).
- Layered components
  - intermediaries, such as proxy servers, cache servers, gateways, etc, can be inserted between clients and resources to support performance, security, etc.

CRUD to HTTP

- Create → PUT with a new URI or POST with a base URI returning a new URI
- Read → GET
- Update → PUT,
- Delete → DELETE.

Retrieving and Object

- Object may be retrieved in whole or in parts
- Example:

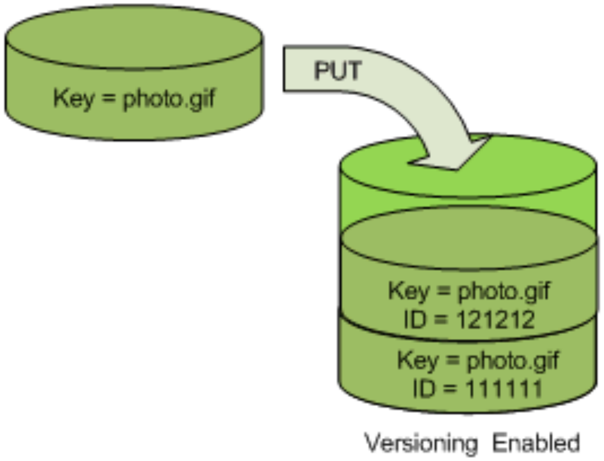
```
GET /my-image.jpg HTTP/1.1
Host: bucket.s3.amazonaws.com
Date: Wed, 28 Oct 2009 22:32:00 GMT
Authorization: authorization string
```

Updating an Object

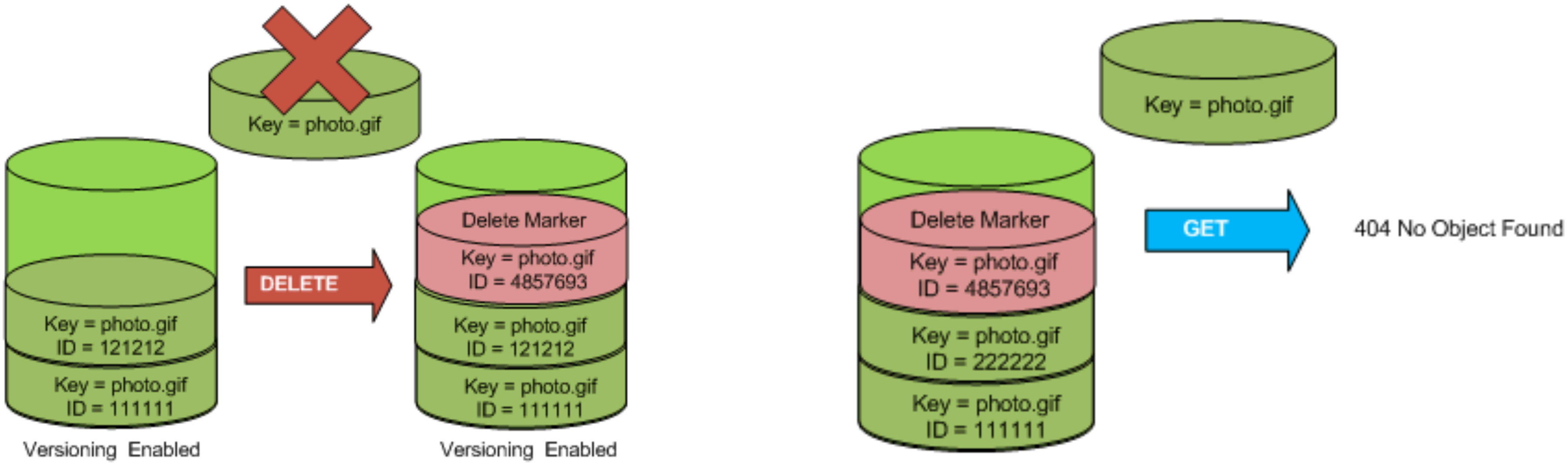
- Only full objects are updated into S3
- Example

```
PUT /my-image.jpg HTTP/1.1
Host: myBucket.s3.amazonaws.com
Date: Wed, 12 Oct 2009 17:50:00 GMT
Authorization: authorization string
Content-Type: text/plain
Content-Length: 11434
Expect: 100-continue
[11434 bytes of object data]
```

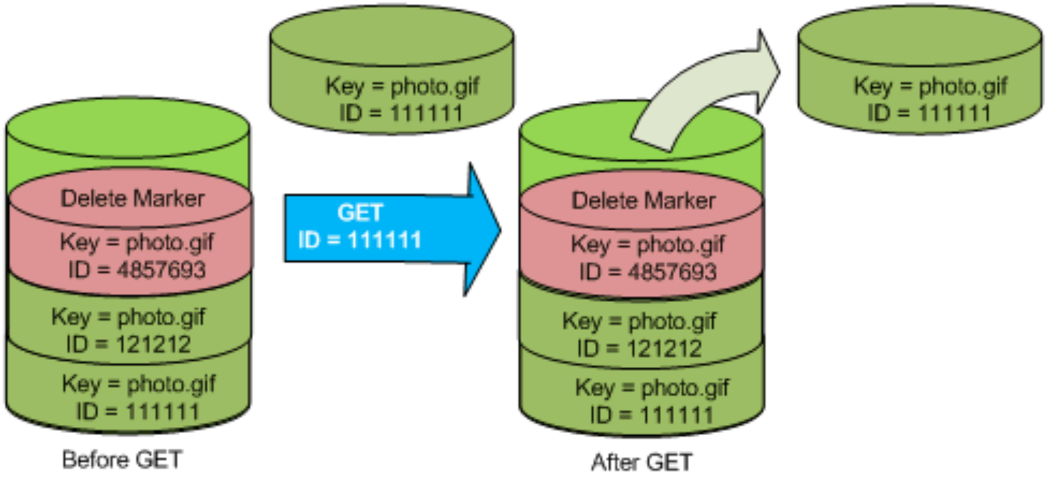
Object Versioning



Deleting an Object



Retrieving a Version



Storage Classes

- Standard
- Infrequent Access
- Glacier
- Use cases driven by pricing for storage and access
  - https://aws.amazon.com/s3/pricing/

S3 Scalability and Availability

- To achieve reliability, there are many copies of the data kept
  - Just like mirroring in RAID
- However, because of network latency, waiting to update all of the copies is not practical

What is consistency?

- If you have data that may be in multiple locations, and accessed by multiple users, what happens when you update the data?
  - When is a write seen by a read?
- For example:
  - Assume a file is replicated on nodes M and N
  - The client A writes to the to node N
  - After a period of time t, client B reads the file from node M
  - The consistency model has to determine whether client B sees the write from client A or not.

Consistency Models

- *Strong consistency.*
  - After the update completes, any subsequent access (by A, B, or C) will return the updated value.
- *Weak consistency.*
  - The system does not guarantee that subsequent accesses will return the updated value.
- *Eventual consistency.*
  - This is a specific form of weak consistency;
  - the storage system guarantees that if no new updates are made to the object, eventually all accesses will return the last updated value.

Creating New Objects

- Read after write consistancy for new objects

```
PUT /key-prefix/cool-file.jpg 200
GET /key-prefix/cool-file.jpg 200
```

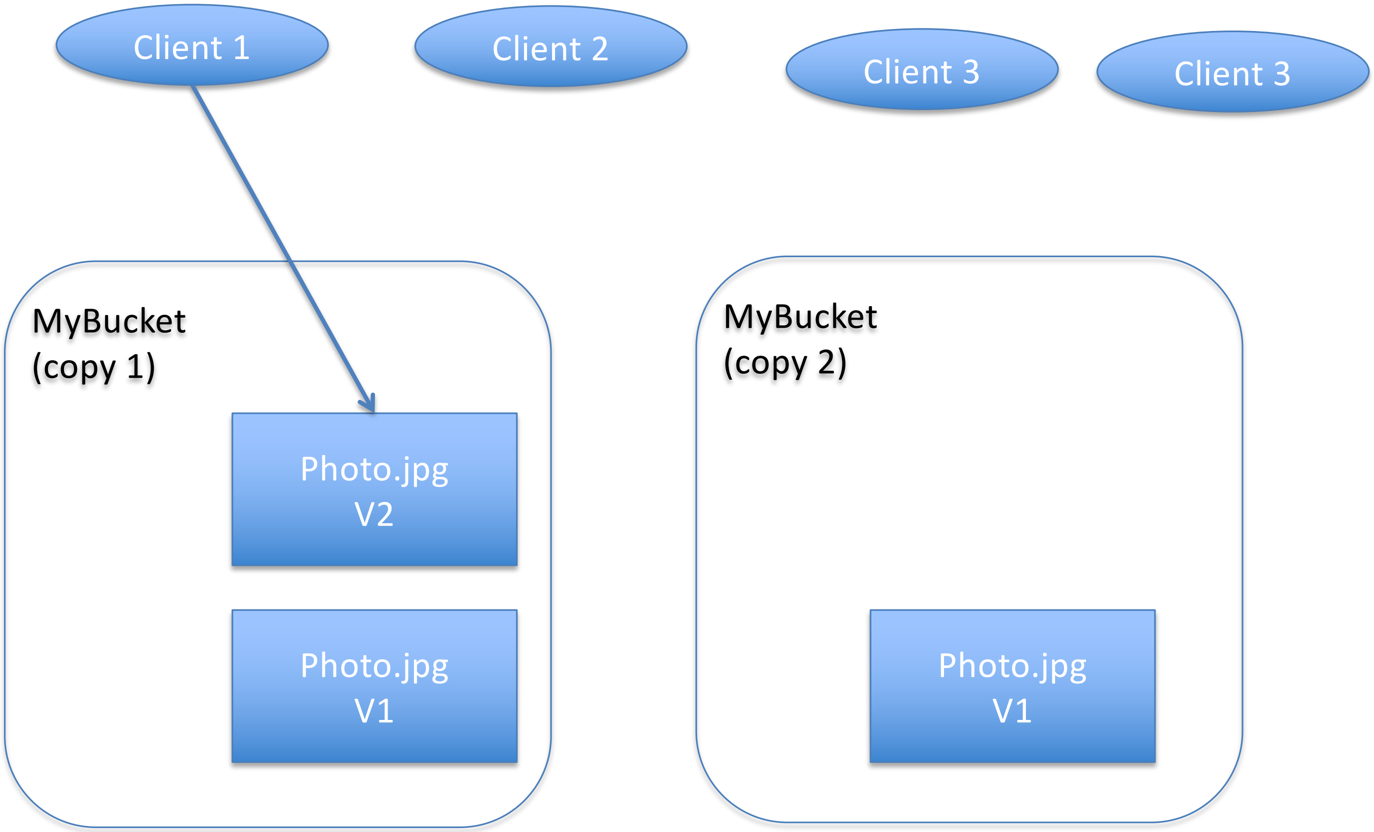
- But in some cases, not quite…

```
GET /key-prefix/cool-file.jpg 404
PUT /key-prefix/cool-file.jpg 200
GET /key-prefix/cool-file.jpg 404
```

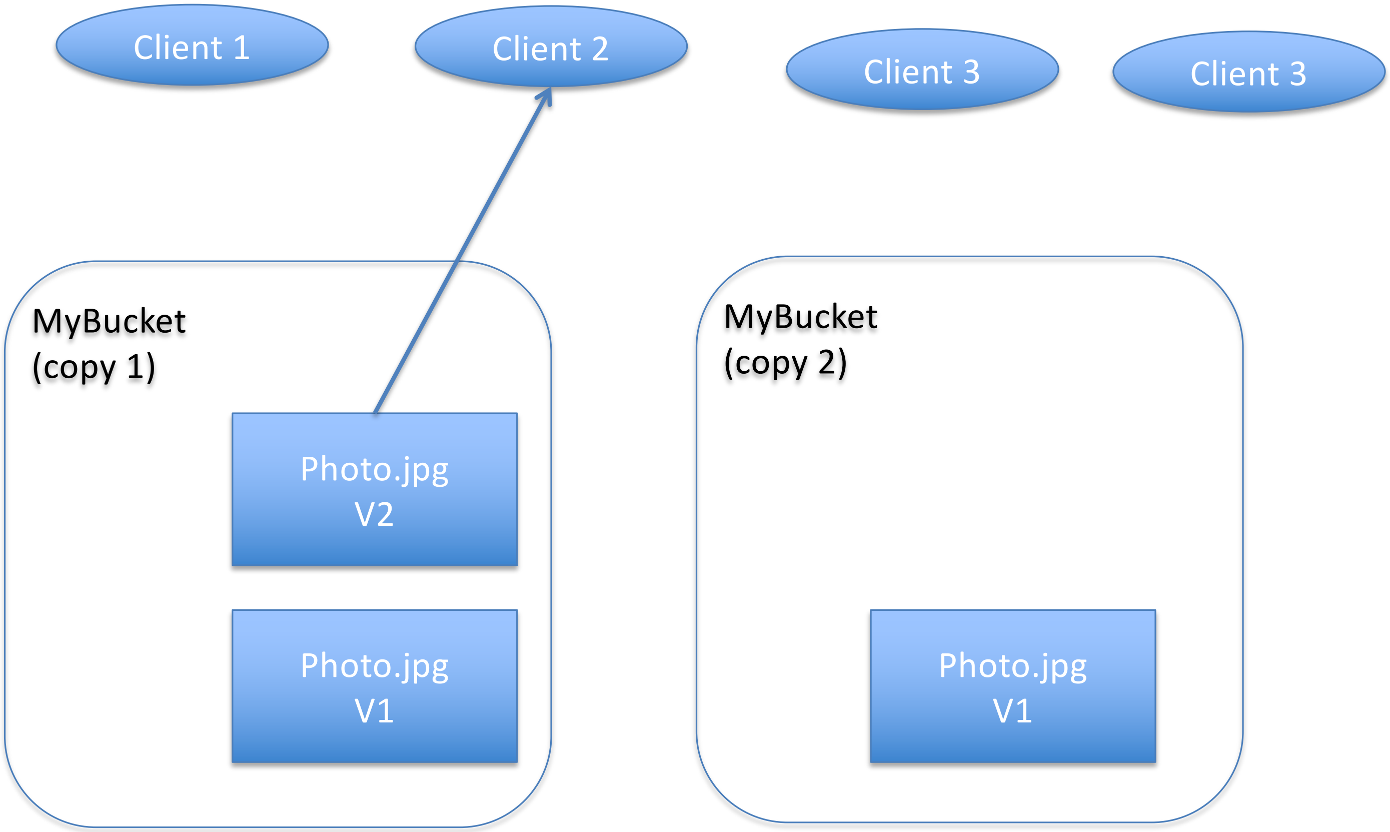
What about updates?

- S3 has “eventual consistency” for updates
- Reads may return old values for an object
- Eventually all reads will return the same value

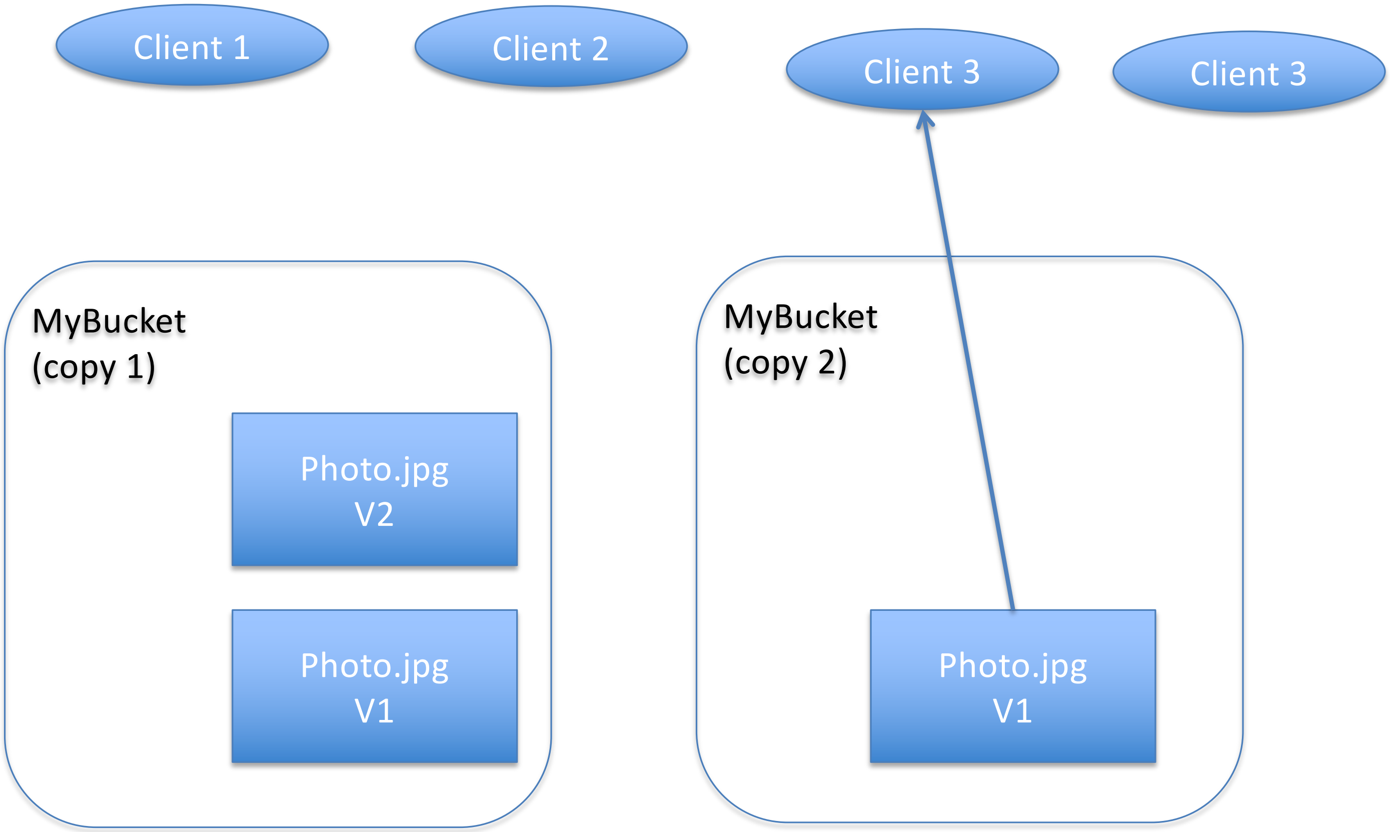
Eventual Consistency



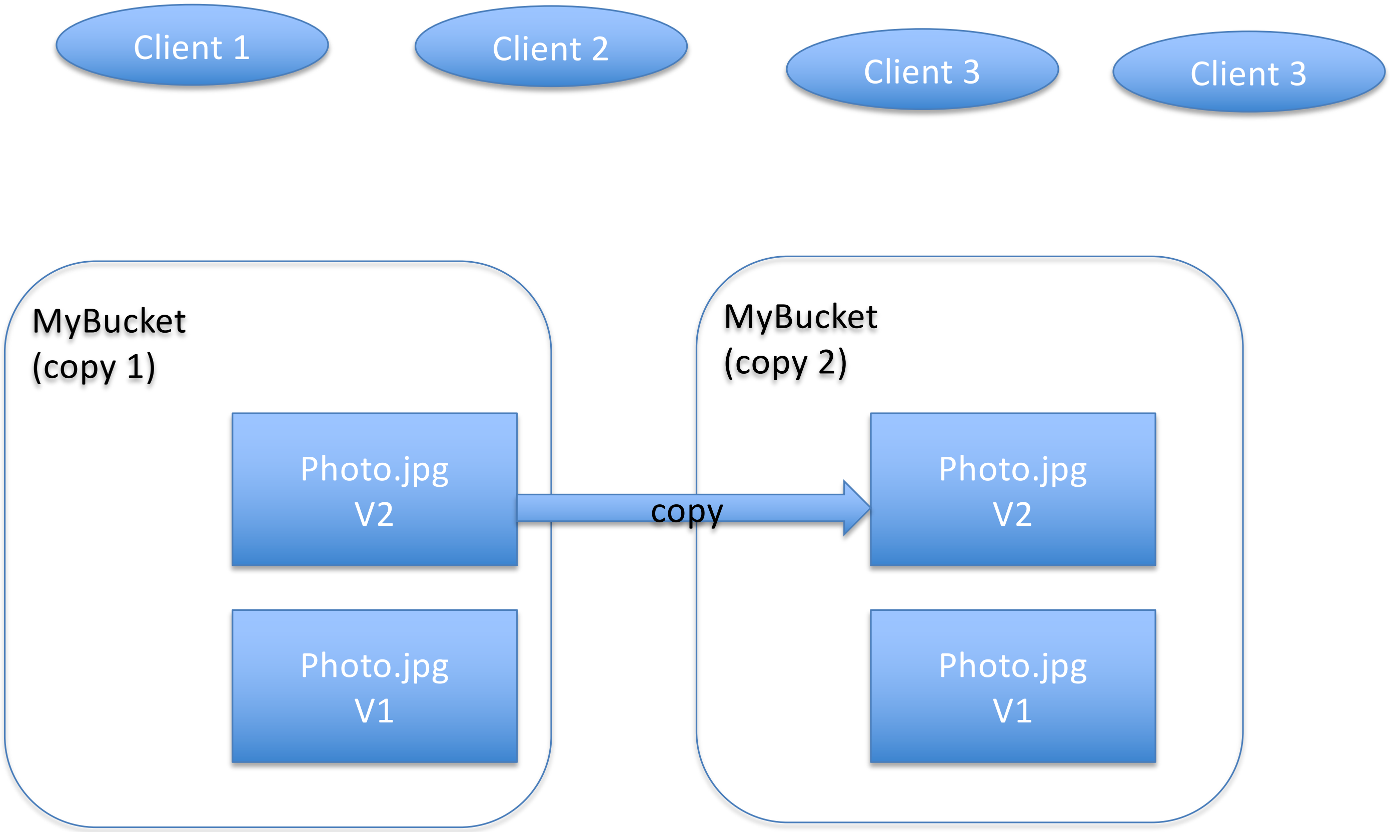
Eventual Consistency



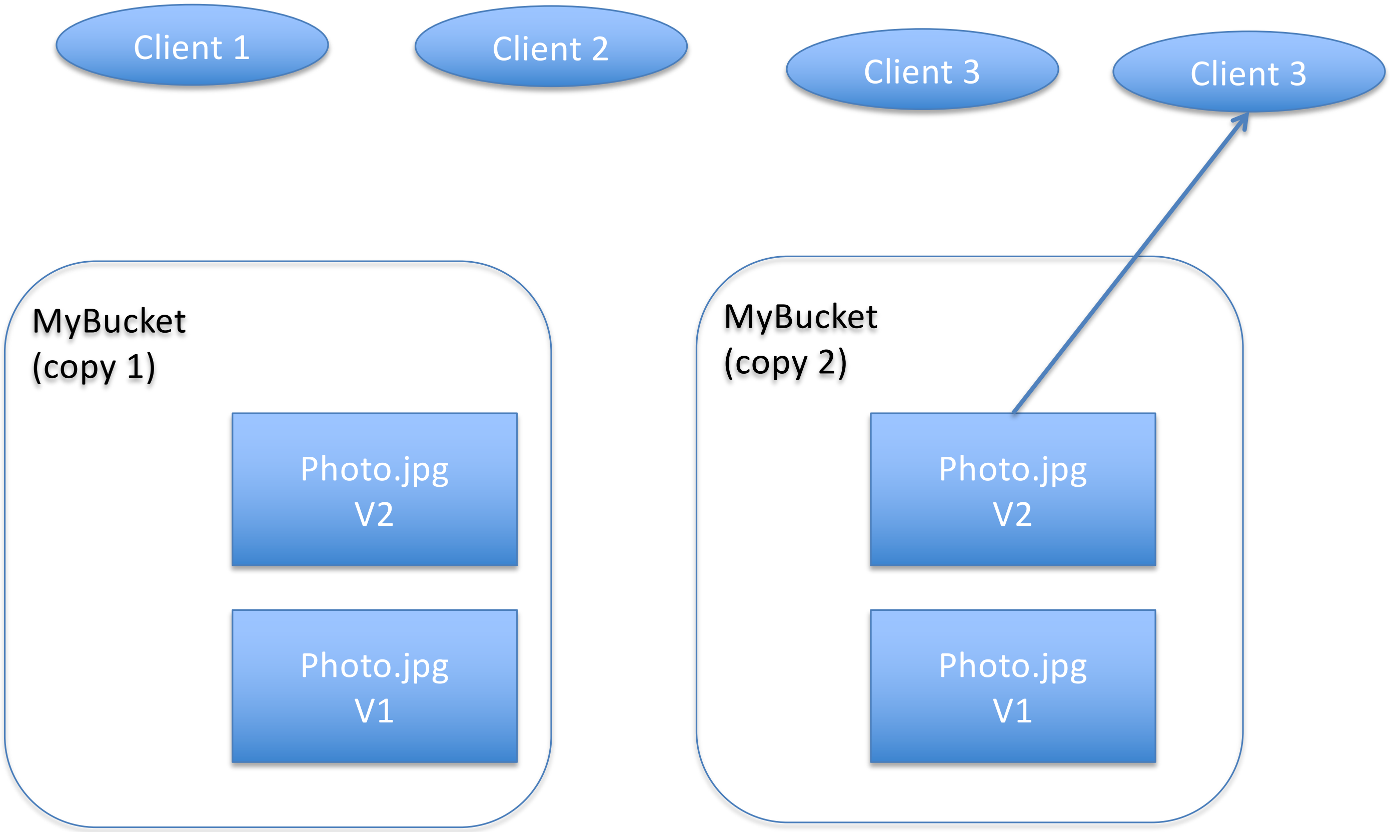
Eventual Consistency



Eventual Consistancy



Eventual Consistency


Client 1
Client 2
Client 3
Client 3
MyBucket
(copy 1)
Photo.jpg
V2
Photo.jpg
V1
MyBucket
(copy 2)
Photo.jpg
V2
Photo.jpg
V1