Introduction to Ceph

Or Building an Object Store in 5 Parts

http://openwest.dev-zero.net/intro-to-ceph.odp
http://openwest.dev-zero.net/intro-to-ceph.pdf

Let’s build a hypothetical object store

First, a definition

Object

Sequence of Bytes

Has a Name

Optionally some extra attributes

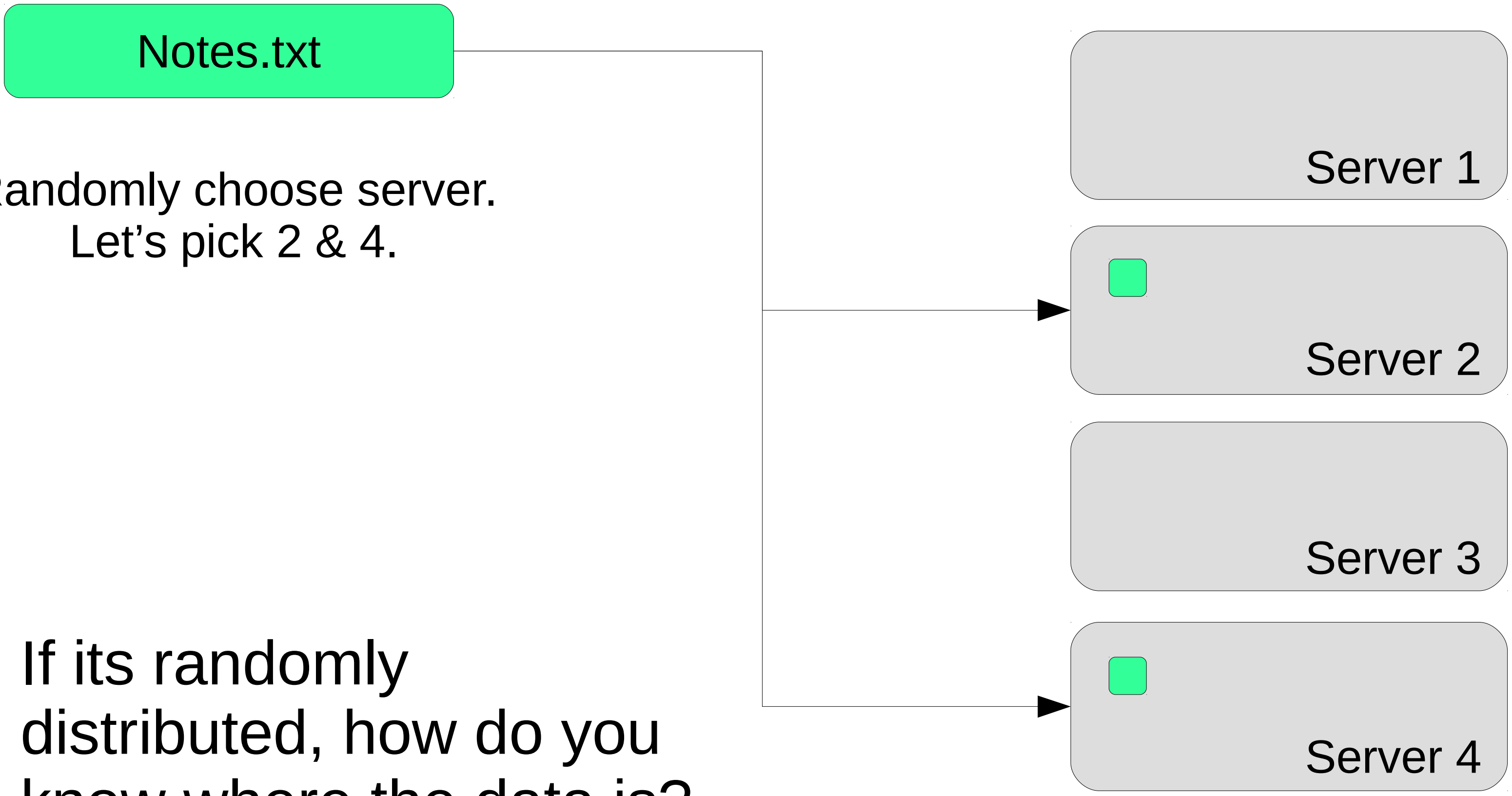
Very much like a File

No directory hierarchy

Requirements

- Data can be written
- Data can be read
- Data should be distributed
  - Spread across multiple storage locations
- Data should be fault tolerant
  - Handle failures and automatically recover
- Data should be consistent
  - Different data is a bad thing

Part I : Objects and Storage Servers



Randomly choose server.
Let’s pick 2 & 4.

If its randomly distributed, how do you know where the data is?

Problem 1

- Need a way to know where the data is stored so we can read it

Otherwise, our storage system is not better than

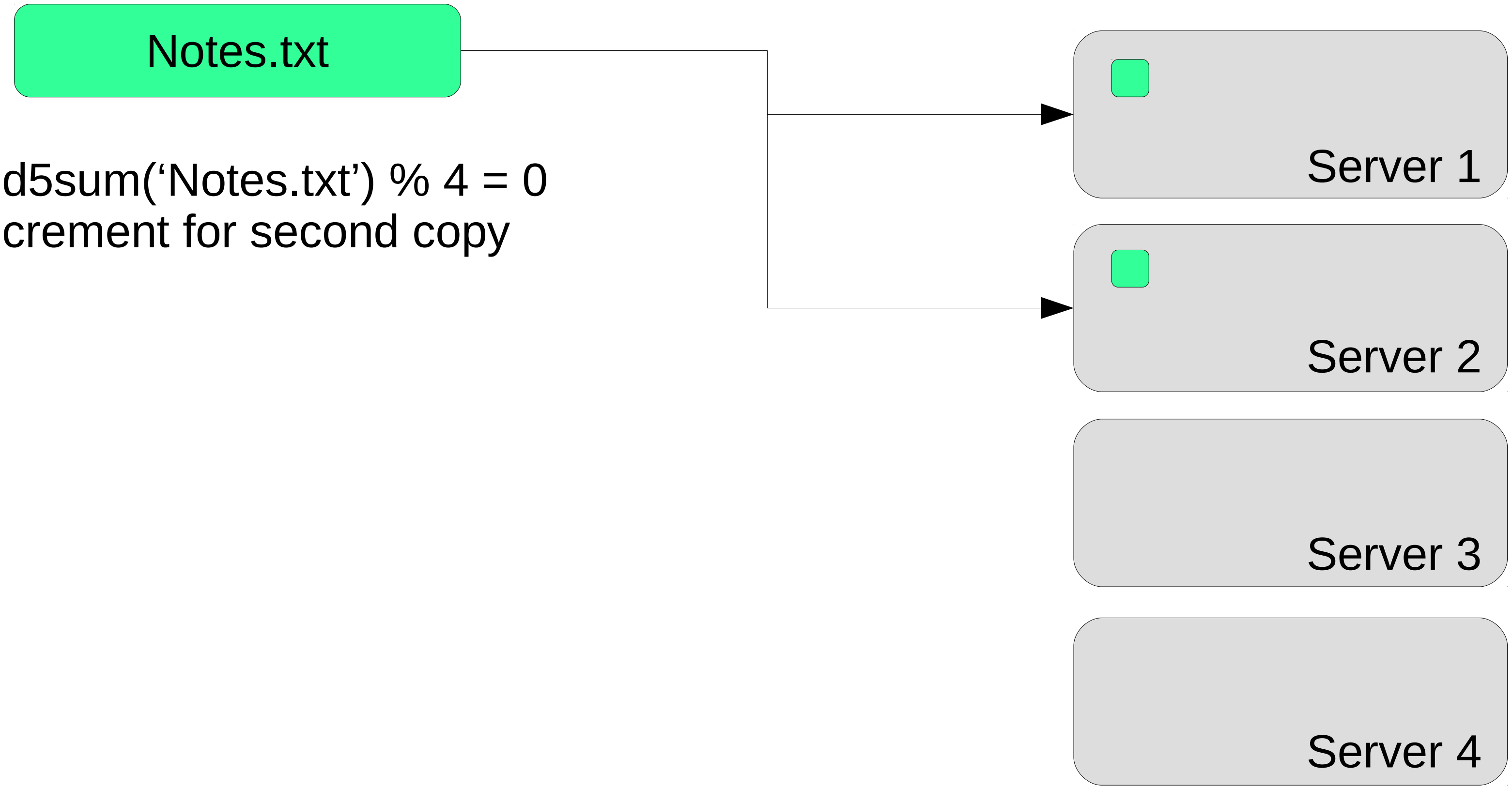
cat Notes.txt > /dev/null && rm Notes.txt

Problem 1

- Multiple possible solutions
  1) Use something else to store Object→Server mappings. Used by some distributed systems. Adds an extra operation to each read and write.
  2) Use an attribute of the Object and do some math. Can use Object name, Object contents, or combination of both. Probably going to use some kind of hash function.

S = md5sum(name) % NumServers

Part II : Objects mapped using hashes



md5sum(‘Notes.txt’) % 4 = 0
Increment for second copy

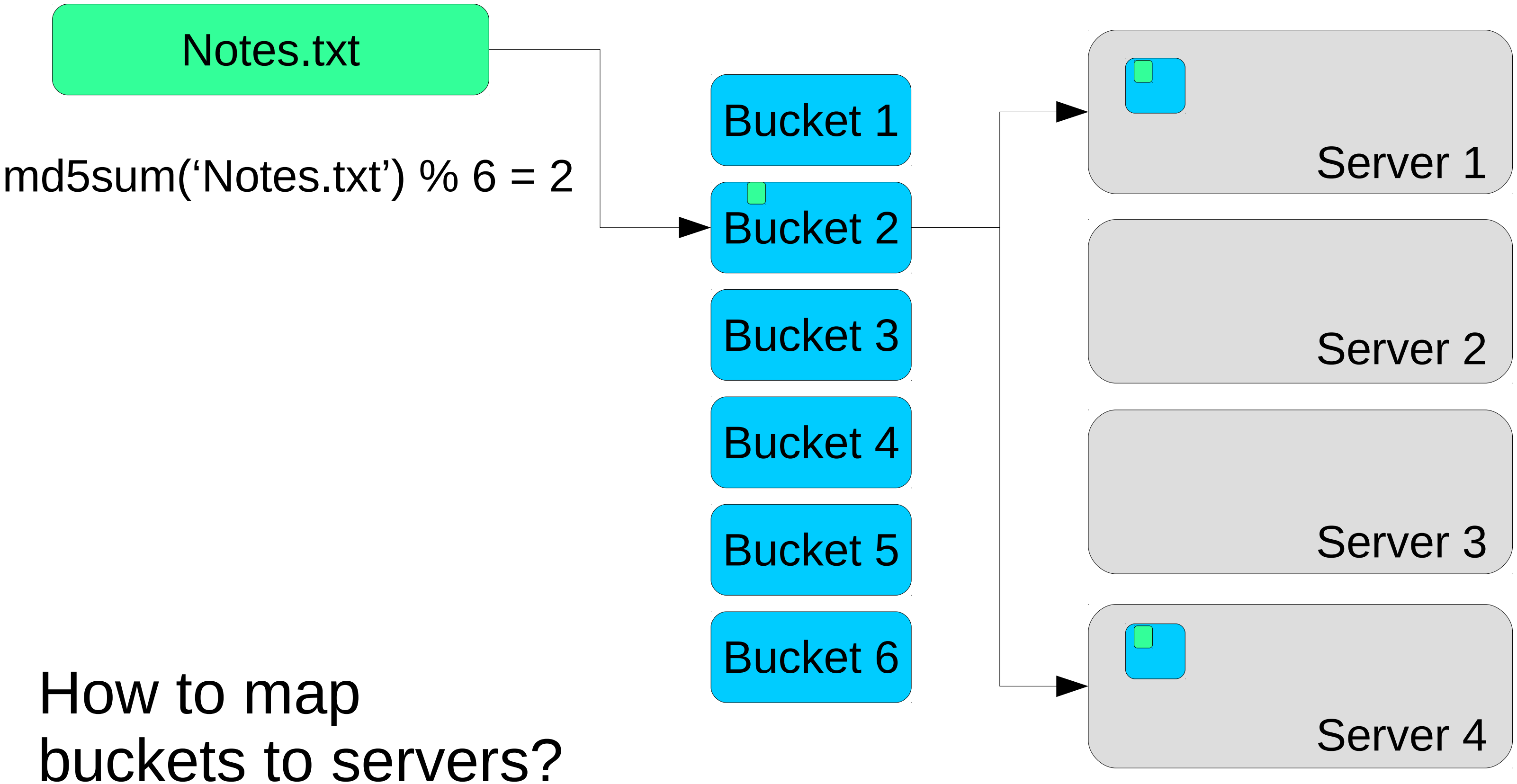
Problem 2

- Recovery operations are painful
  - What objects were on a failed server?
  - Which objects are degraded?
- Adding or removing servers makes locations for previously stored objects invalid
  - Have to potentially move a lot of data

Solution: Create a logical ‘bucket,’ Map objects to buckets, then map buckets to Servers

Part III : Object Buckets

Buckets Map to Servers



How to map buckets to servers?

http://www.reactiongifs.com/magic-3/

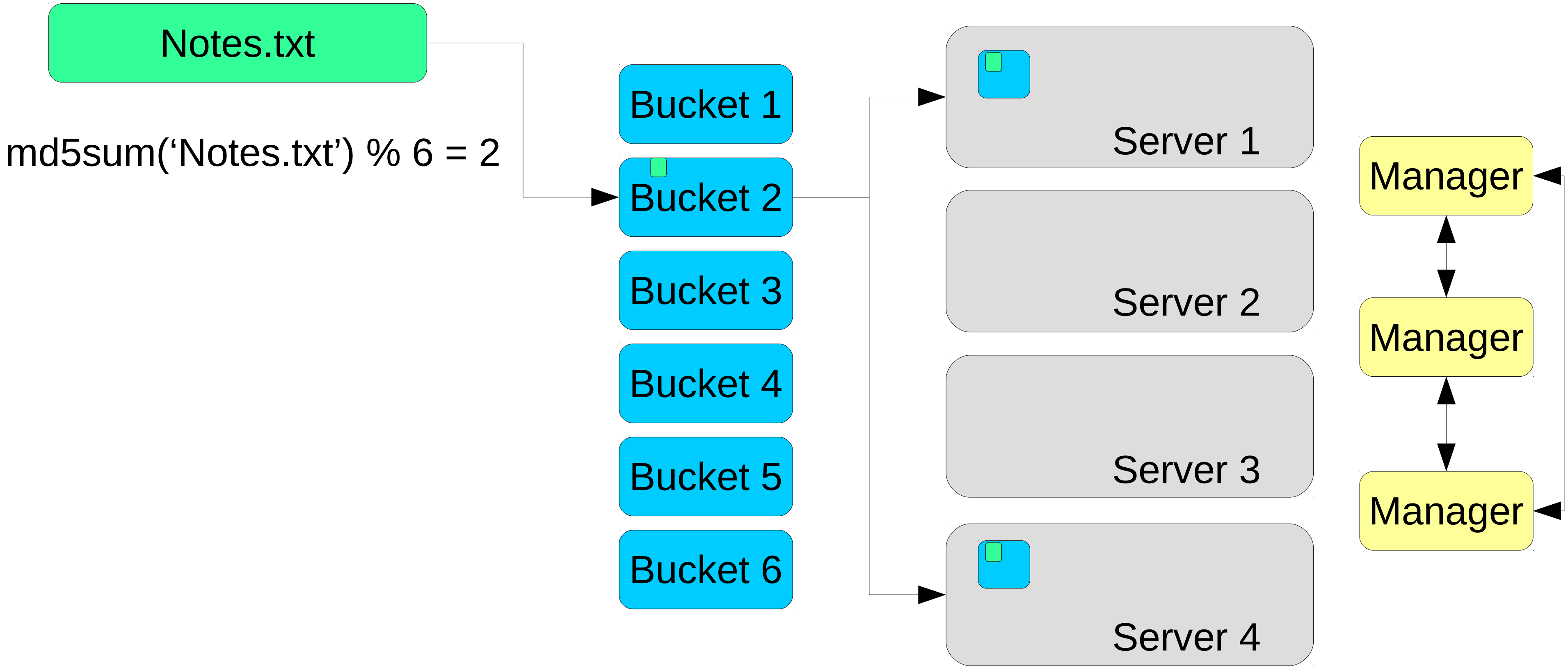
Several potential ways but we’ll put that on hold for a minute

Problem 3

- How is the state of the environment known?
- What servers are running and available?
- Which buckets are consistent? Which are missing a member?

Solution: Use a distributed consensus algorithm and some additional servers to keep state

Part IV : Distributed Consensus

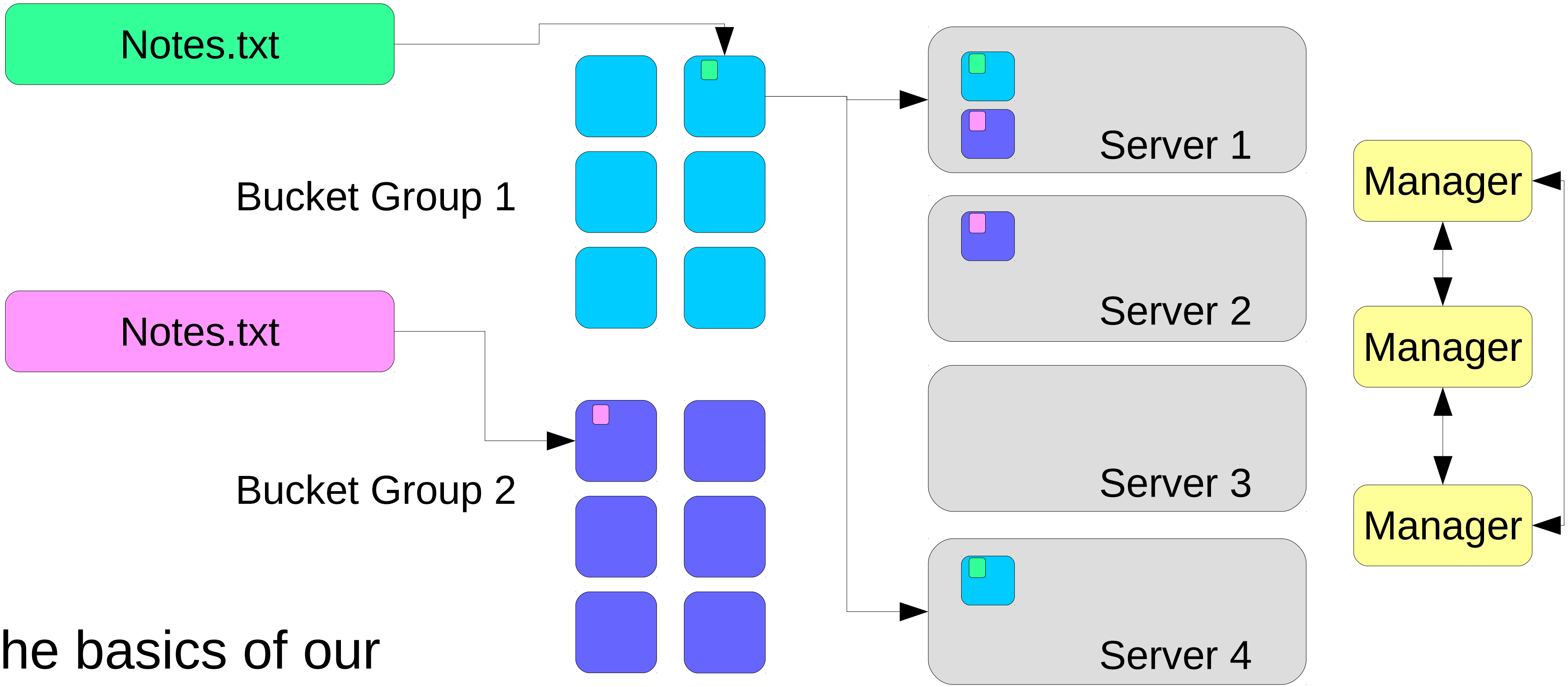

Notes.txt
md5sum('Notes.txt') % 6 = 2
Bucket 1
Bucket 2
Bucket 3
Bucket 4
Bucket 5
Bucket 6
Server 1
Server 2
Server 3
Server 4
Manager
Manager
Manager

Problem 4

- 1 global name space quickly leads to object collisions
- What if different storage policies are needed for different types of data?

Solution: Use multiple groups of buckets. Objects are stored in a group specified by the client.

Part V : Bucket Groups



The basics of our hypothetical object store are now in place

Translation to Ceph

Hypothetical Object Store	Ceph
Object	Object
Server	Object Storage Daemon (OSD)
Bucket	Placement Group (PG)
Manager	Monitor (Mon)
Bucket Group	Pool

Translation to OpenStack Swift?

Hypothetical Object Store	Ceph	Swift
Object	Object	Object
Server	Object Storage Daemon (OSD)	Object Server
Bucket	Placement Group (PG)	Partitions
Manager	Monitor (Mon)	The Ring
Bucket Group	Pool	Storage Policies

Not exactly direct translations to Swift and crude approximation. I haven’t looked at it seriously in 7 years and could be wrong on current architecture. Author assumes no responsibility for incorrectness.

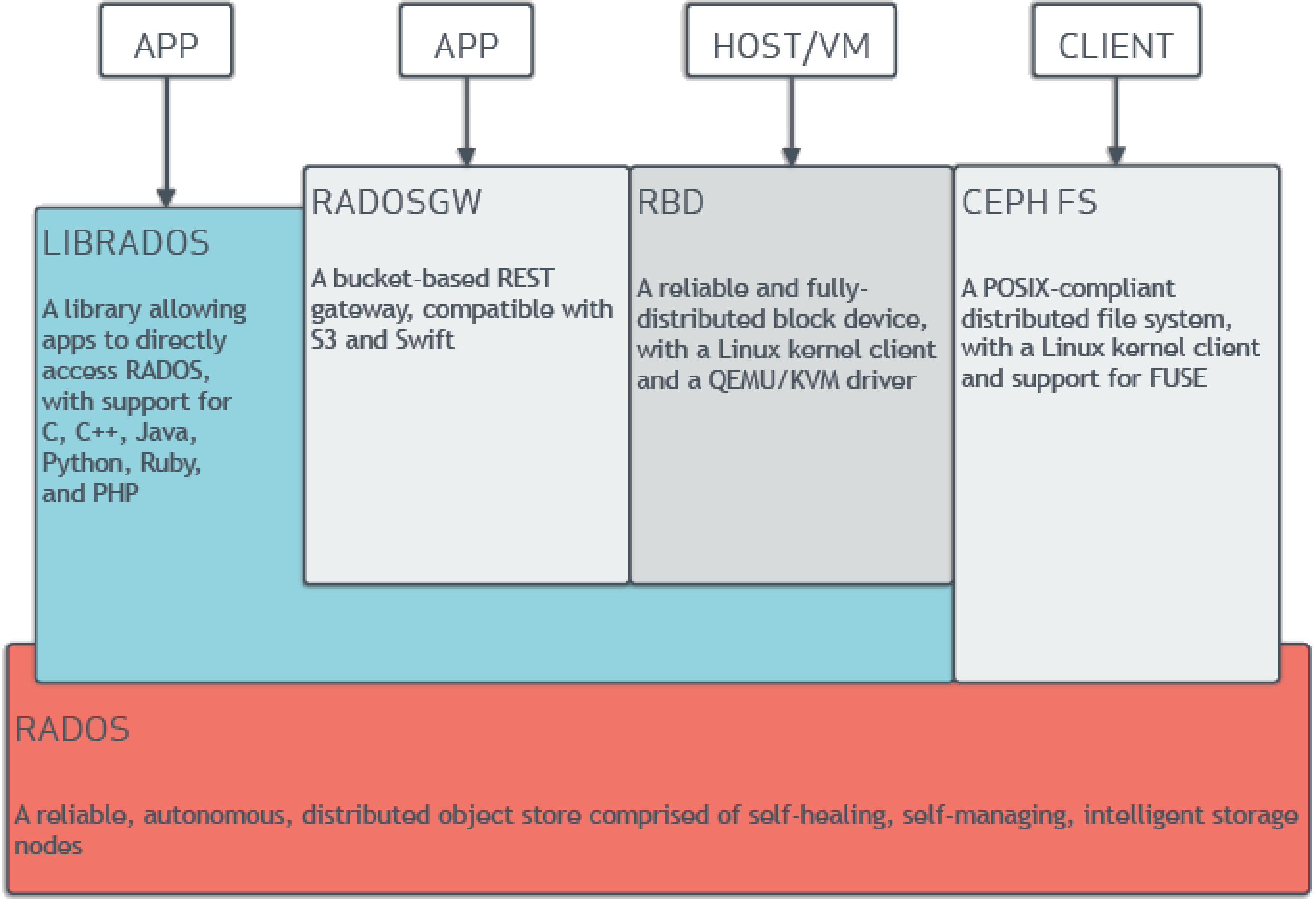
Ceph Specifics

- RADOS – Reliable, Autonomic, Distributed Object Store
  - The Ceph equivalent of the Hypothetical Object Store we built
  - http://ceph.com/papers/weil-rados-pdsw07.pdf
- CRUSH – Controlled Replication Under Scalable Hashing
  - The magic for mapping to a Placement Group to a set of OSDs
  - Allows for OSDs of different size and is location aware
    - Region, Datacenter, Room, Row, Rack, Chassis, Host
  - Rulesets define storage requirements
    - i.e. 3 copies where each copy is in a different rack
  - http://ceph.com/papers/weil-crush-sc06.pdf

Ceph Architecture

Clients

Object Store



http://docs.ceph.com/docs/jewel/architecture/

Ceph Clients

- Using the object store requires a client that can talk the protocol and semantics of the object store.
- Couple of helper libraries and applications to make that simpler
- librados – C Library that talks native Ceph details. Bindings for other languages.
- Radosgw – S3 and Swift like REST based API
  - See http://docs.ceph.com/docs/jewel/radosgw/s3/ and http://docs.ceph.com/docs/jewel/radosgw/swift/ for API compatibility

Ceph Clients

- RBD – RADOS Block Device. Virtual disk abstraction that’s useful for Virtual Machines and other things.
  - Comes in 2 forms. krbd and librbd.
  - krbd is the mainline Linux kernel client. Allows for access to rbds are block devices on any system that can talk to the cluster. Slower development pace and doesn’t support all RBD features.
  - librbd is used by user space applications such as qemu, rbd-nbd, or LIO-TCM.
- CephFS – POSIX Filesystem abstraction. Requires additional Metadata Server.
  - Finally considered production ready with limitations

Demo

Status and Information (Jewel and older)

```
mike@ceph1:~$ sudo ceph status
    cluster 31073bfc-9a63-4141-a6b0-50d50f8b33a0
     health HEALTH_OK  ← Overall Cluster Health
     monmap e3: 3 mons at
{ceph1=172.16.0.11:6789/0,ceph2=172.16.0.12:6789/0,ceph3=172.16.0.
13:6789/0} ← List of monitors
            election epoch 10, quorum 0,1,2 ceph1,ceph2,ceph3
     osdmap e109: 8 osds: 8 up, 8 in ← Number of OSDs and states
            flags sortbitwise
      pgmap v8982: 128 pgs, 9 pools, 6544 bytes data, 183 objects
            337 MB used, 119 GB / 119 GB avail
                 128 active+clean ← Status of Placement Groups
```

Status and Information (Luminous and newer)

```
mike@ceph1:~$ sudo ceph status
  cluster:
    id:     8a5fa553-f2f3-4181-8675-b617e982e259
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum ceph5,ceph6,ceph7
    mgr: ceph5(active), standbys: ceph7, ceph6
    osd: 4 osds: 4 up, 4 in

  data:
    pools:   3 pools, 96 pgs
    objects: 39.21k objects, 151GiB
    usage:   450GiB used, 3.16TiB / 3.60TiB avail
    pgs:     96 active+clean
```

Status and Information

- `ceph health [detail]` #Additional health information
- `ceph osd tree` #Locations of OSDs in the CRUSH maps
- `ceph osd find <osd number>` #Show location of OSD
- `ceph osd map <pool> <object name>` #Show Placement Group and OSDs that object maps to

Direct Object Access - CLI

- `rados put <object name> <infile>` #Write an Object.
- `rados get <object name> <outfile>` #Read an Object.
- `rados ls` #List all Objects in a Pool. Can take a long time.
- `rados listomapvals` <object name> #List attributes and values associated with object

RBD

- `rbd create --size <size> <image name>` #Create RBD
- `rbd map <image name>` #Connect RBD image to host kernel
- `rbd unmap /dev/rbd<number>` #Disconnect RBD

- Qemu: `-drive file=rbd:<pool>/<rbdname>`
- libvirt

```
<disk type='network' device='disk'>
  <driver name='qemu' type='raw' cache='writeback'/>
  <source protocol='rbd' name='$POOL/$RBD_NAME'>
    <host name='172.16.0.11' port='6789'/>
    <host name='172.16.0.12' port='6789'/>
    <host name='172.16.0.13' port='6789'/>
  </source>
  <target dev='vda' bus='virtio'/>
</disk>
```

RBD

- rbd-nbd
  - Allows for using the user-space RBD libraries that have more features than the kernel RBD client
  - Translates between Network Block Device protocol and Ceph RBD
  - `rbd-nbd map <pool>/<rbd name>`
  - `rbd-nbd unmap /dev/nbd#`
- iSCSI gateway
  - Uses Linux-LIO + TCM Userspace app
  - 2nd implementation in STGT. Older and probably deprecated.

Radosgw

- s3cmd
  - Set access_key and secret_key in s3cmd.conf
  - Set host_base and host_bucket to name or IP of RGW
  - `` `s3cmd -c /path/to/s3cmd.conf ls` ``
  - `` `s3cmd -c /path/to/s3cmd.conf put /path/to/file s3://BUCKET` ``
  - `` `s3cmd -c /path/to/s3cmd.conf get s3://BUCKET/file /path/to/file` ``
- Duplicity
  - Put S3 credentials in /etc/boto.cfg
  - `` `duplicity /path/to/save s3://radosgw.ip.fqdn/BUCKET` ``
  - `` `duplicity list-current-files s3://radowgw.ip.fqdn/BUCKET` ``

Questions?

Extras

- Building a Cluster
  - Use reliable and known hardware. Do some math on sizing and capacity in advance.
  - Don’t bother with hardware RAID for OSDs. Run one daemon per individual hard disk. Ceph handles the redundancy.
  - Know your use case. While some things fit nicely into Ceph, some workloads might be better suited by another solution.

Resources

- http://docs.ceph.com ← Good once you know your way around
- https://www.youtube.com/watch?v=I2aTsugXHEQ ← LCA 2014 talk by Sage. I find his example of using Ceph as a email storage system to be intriguing.
- https://www.sebastien-han.fr/blog/ ← Blog with lots of useful information especially in the context of Ceph with OpenStack