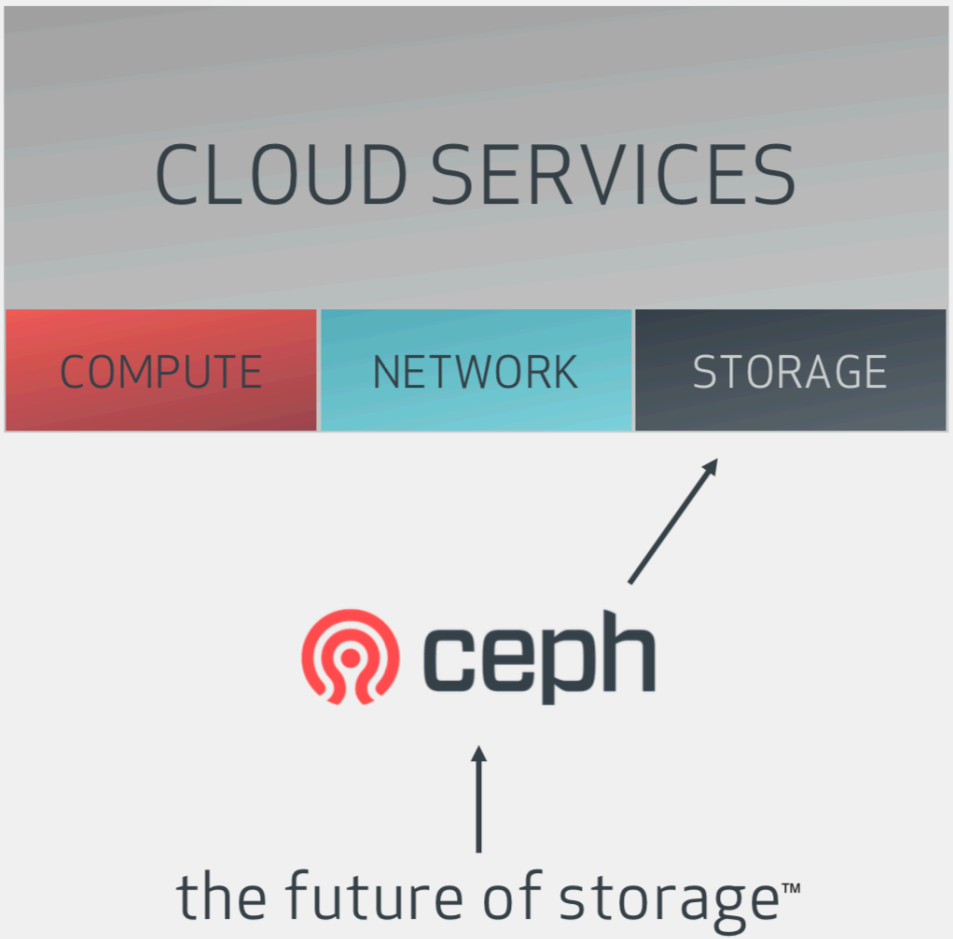
Introduction To Ceph

Houmin Wei

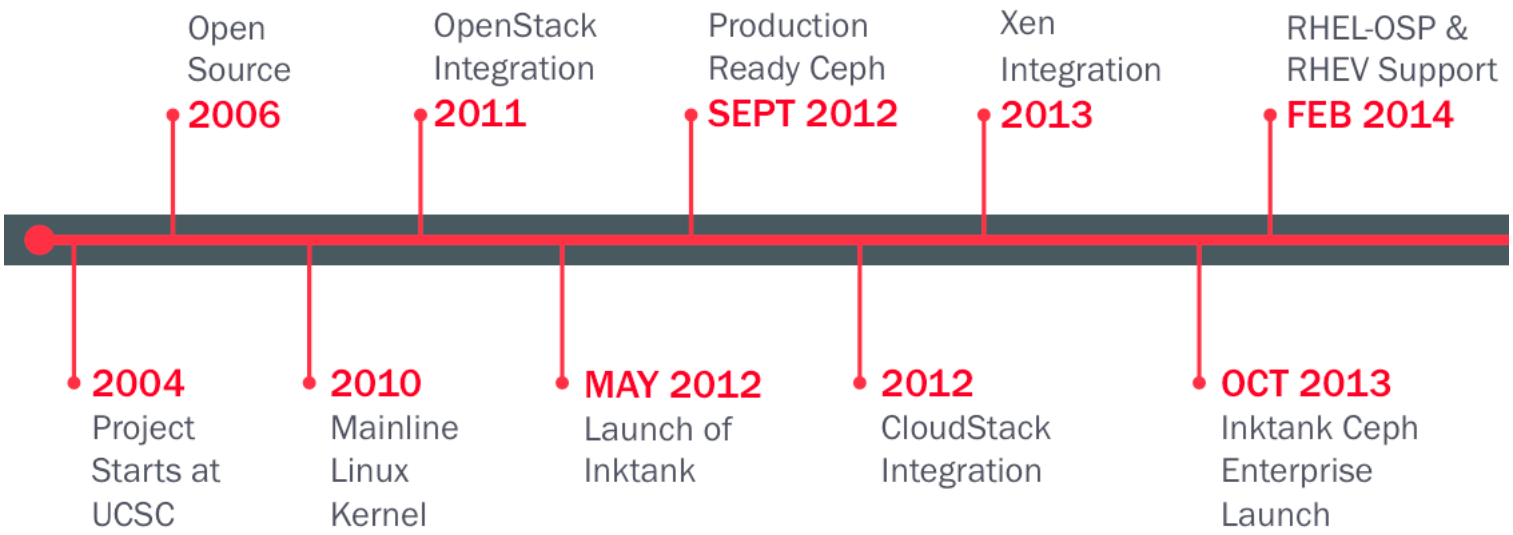
Infrastructure @ Shannon

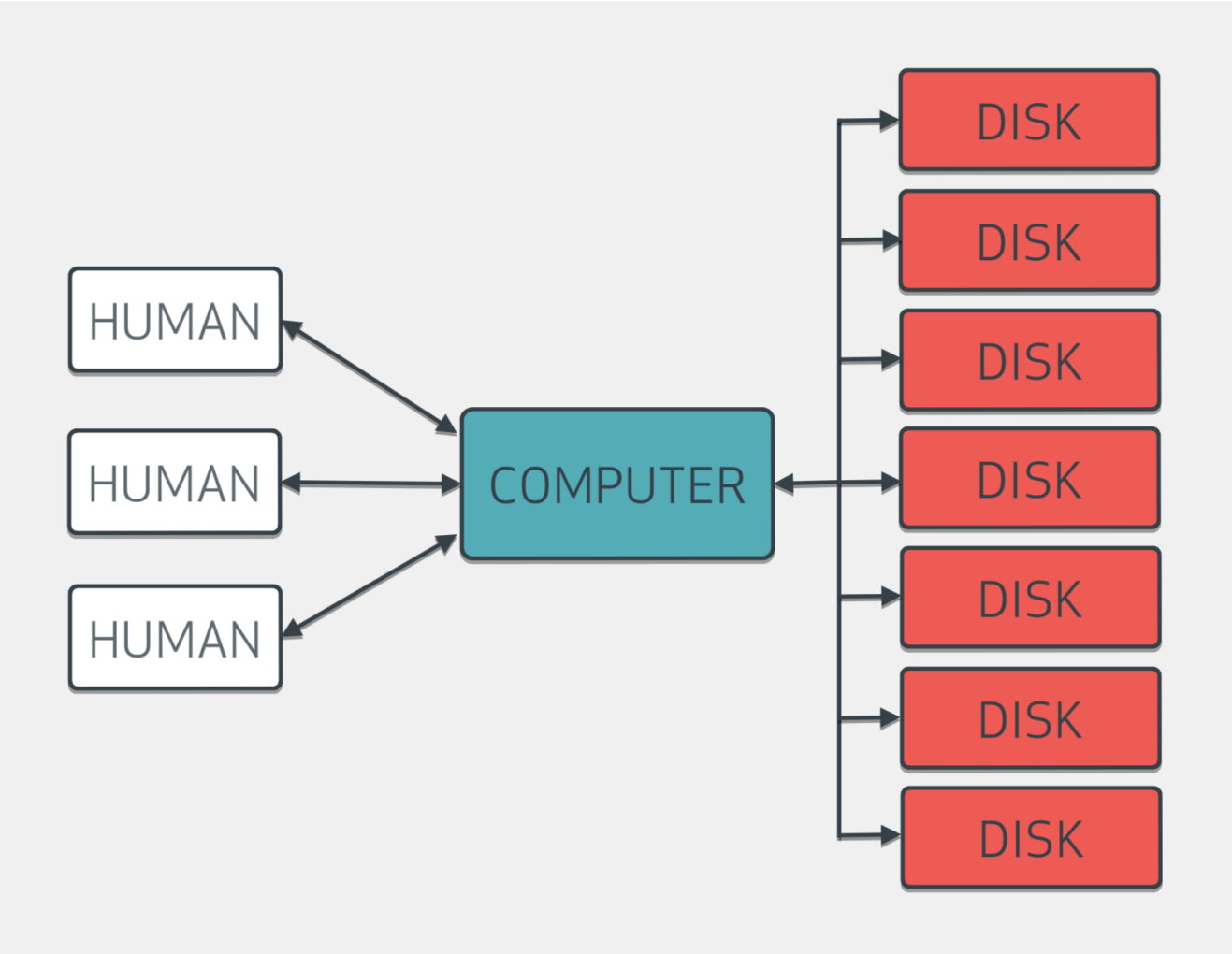
January 3, 2019

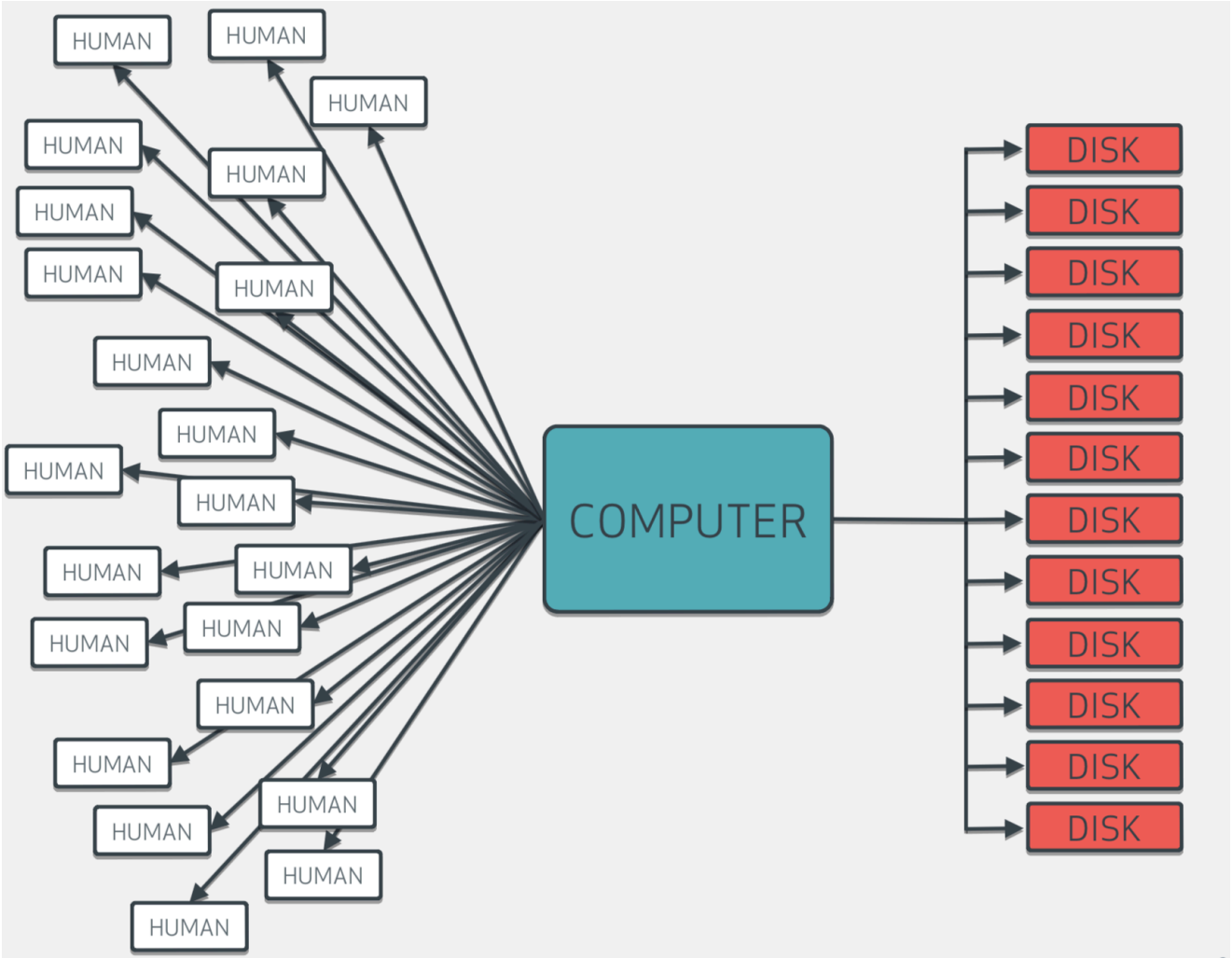

CLOUD SERVICES
COMPUTE
NETWORK
STORAGE
ceph
the future of storage™

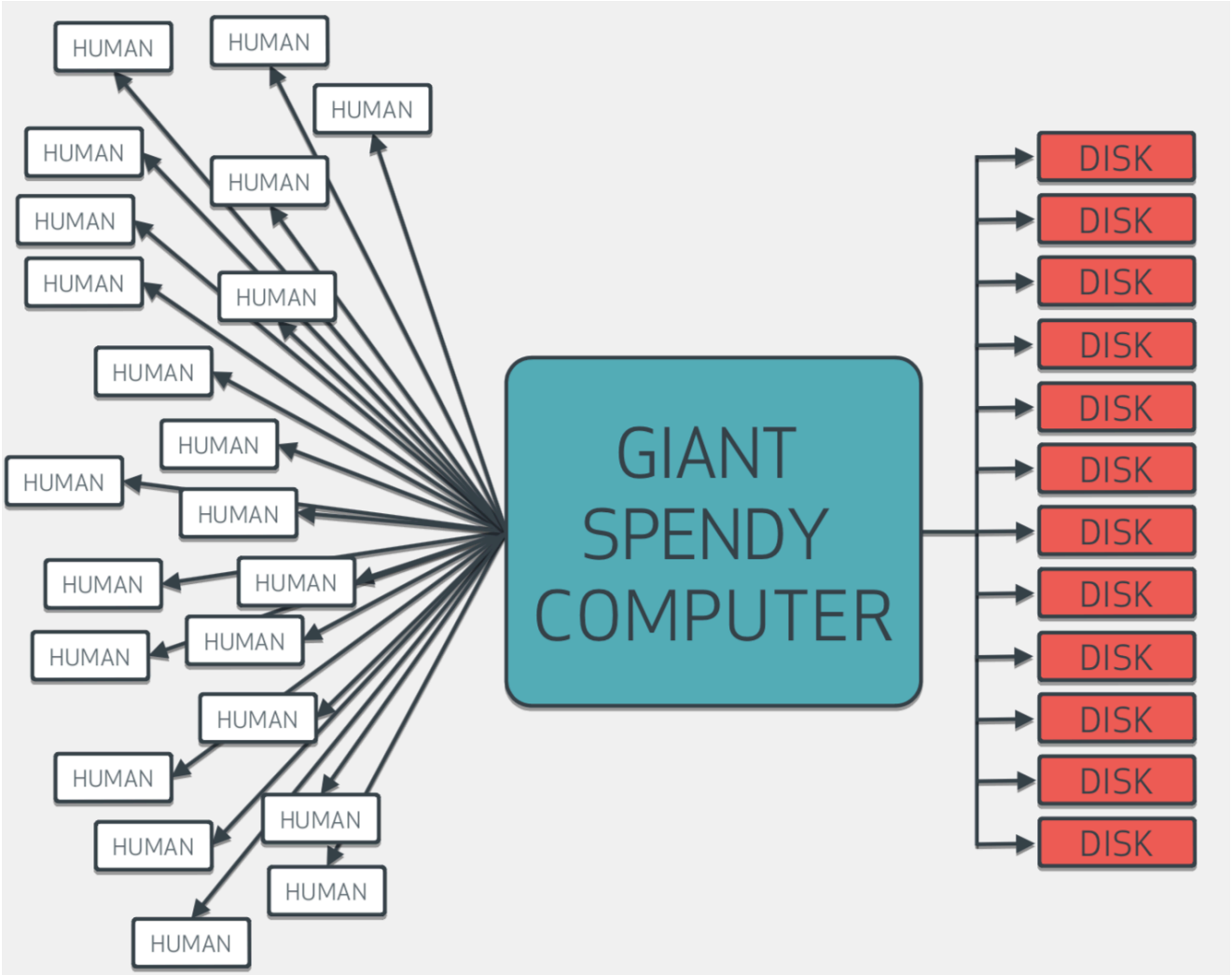
Timeline of Ceph

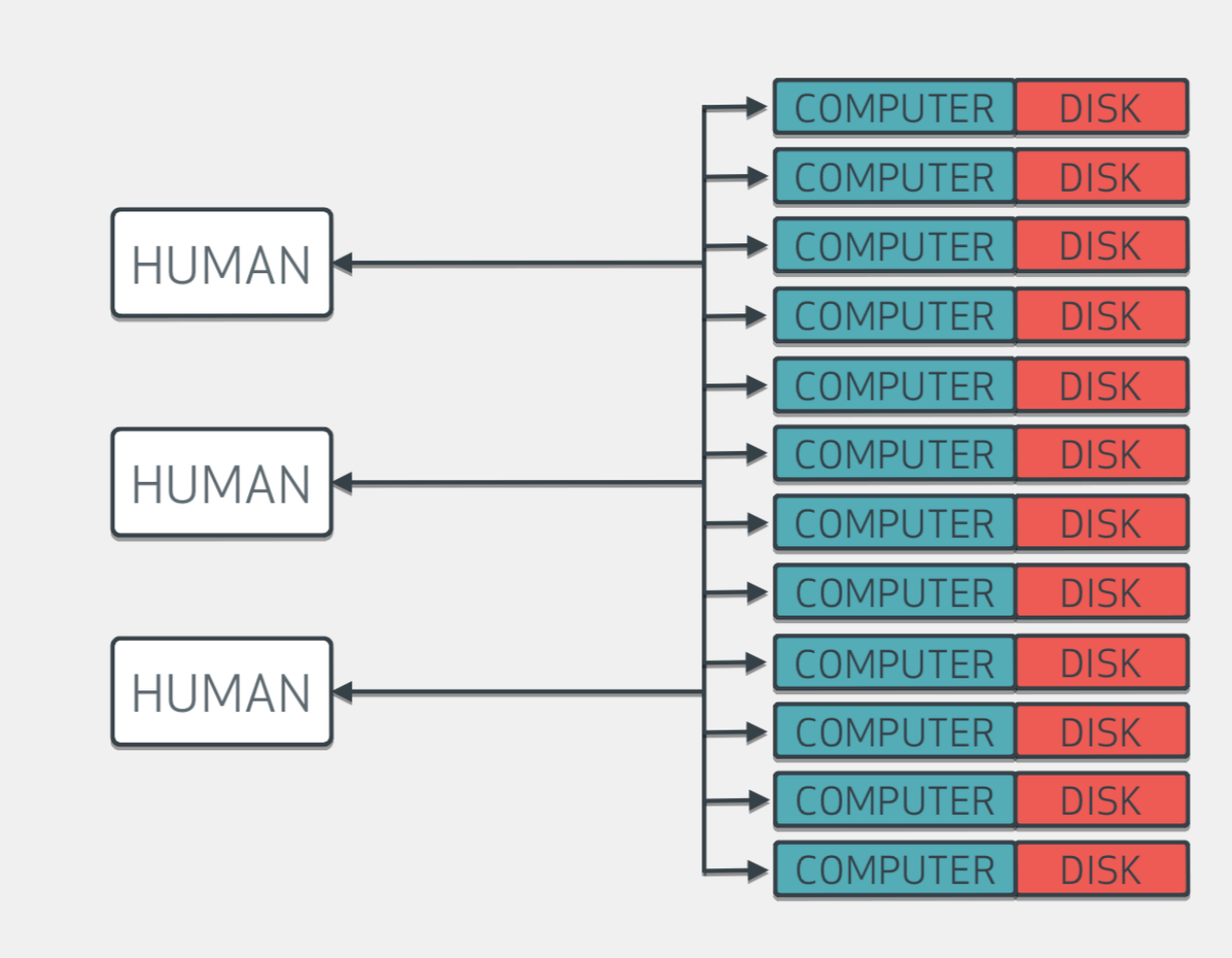
Original author: **sage weil**

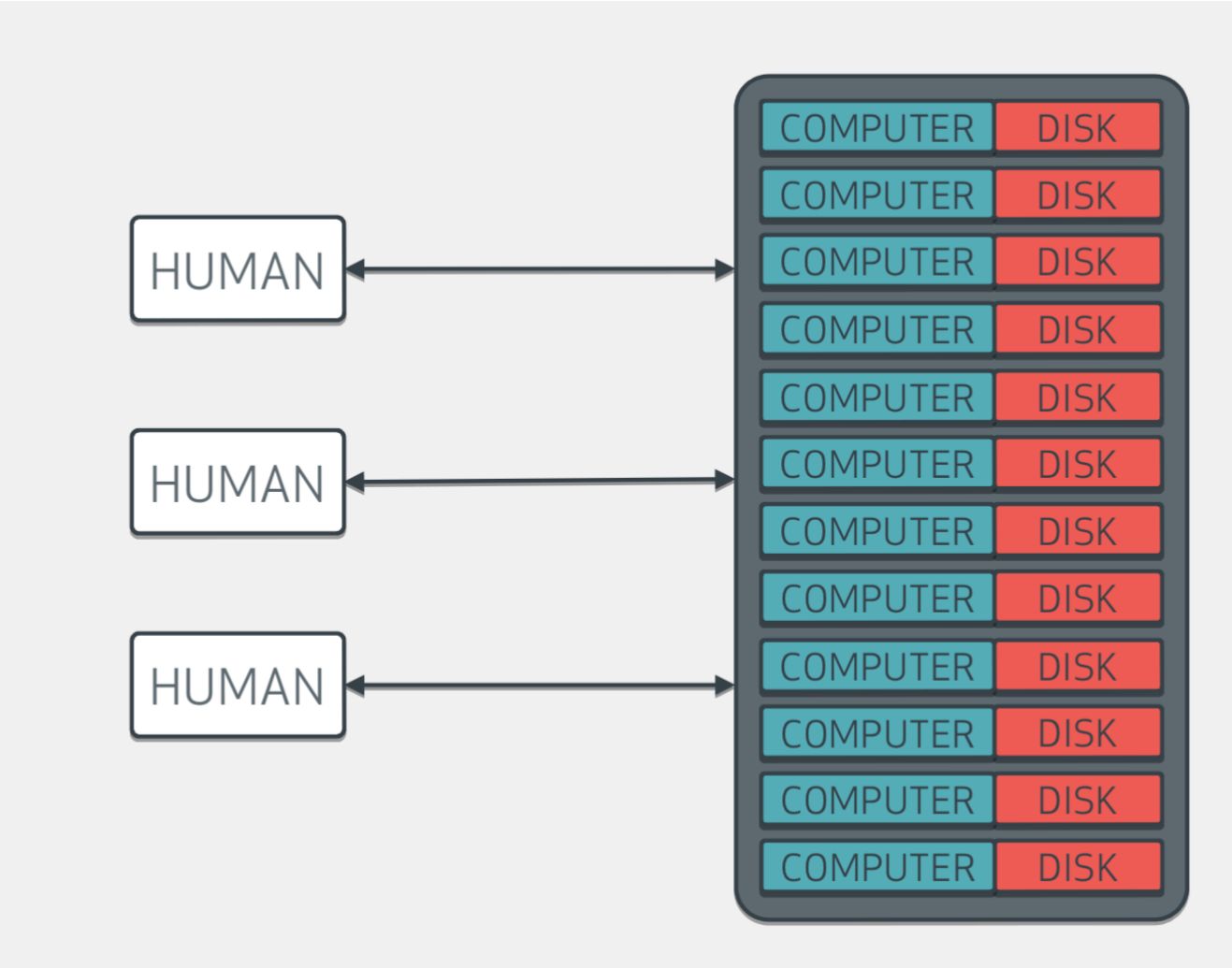



HUMAN
HUMAN
HUMAN
COMPUTER
DISK
DISK
DISK
DISK
DISK
DISK
DISK

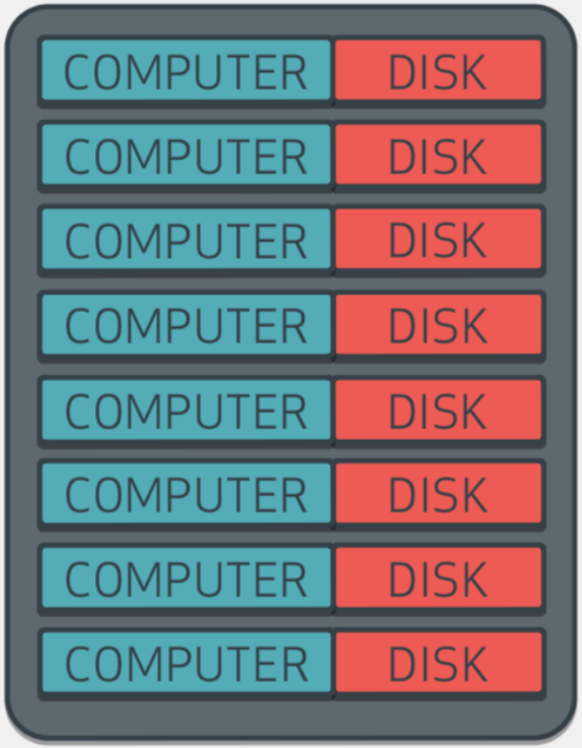

HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
COMPUTER
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK


HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
HUMAN
GIANT SPENDY COMPUTER
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK
DISK


HUMAN
HUMAN
HUMAN
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK


HUMAN
HUMAN
HUMAN
COMPUTER
DISK

"STORAGE APPLIANCE"



Storage Challenges

- **Must Support Current Infrastructures**
  - Block storage with snapshots, cloning
  - File storage with POSIX support
- **Must plan for integration of emerging technologies**
  - Cloud infrastructures and SDNs
- **Must Support New Challenges**
  - Object storage to support massive influx of unstructured data
- All this while supporting:
  - **Massive data growth -> scalability**
  - **Mixed hardware that must work heterogeneously**
  - **Maintain reliability and fault tolerance**
  - **Performance!**

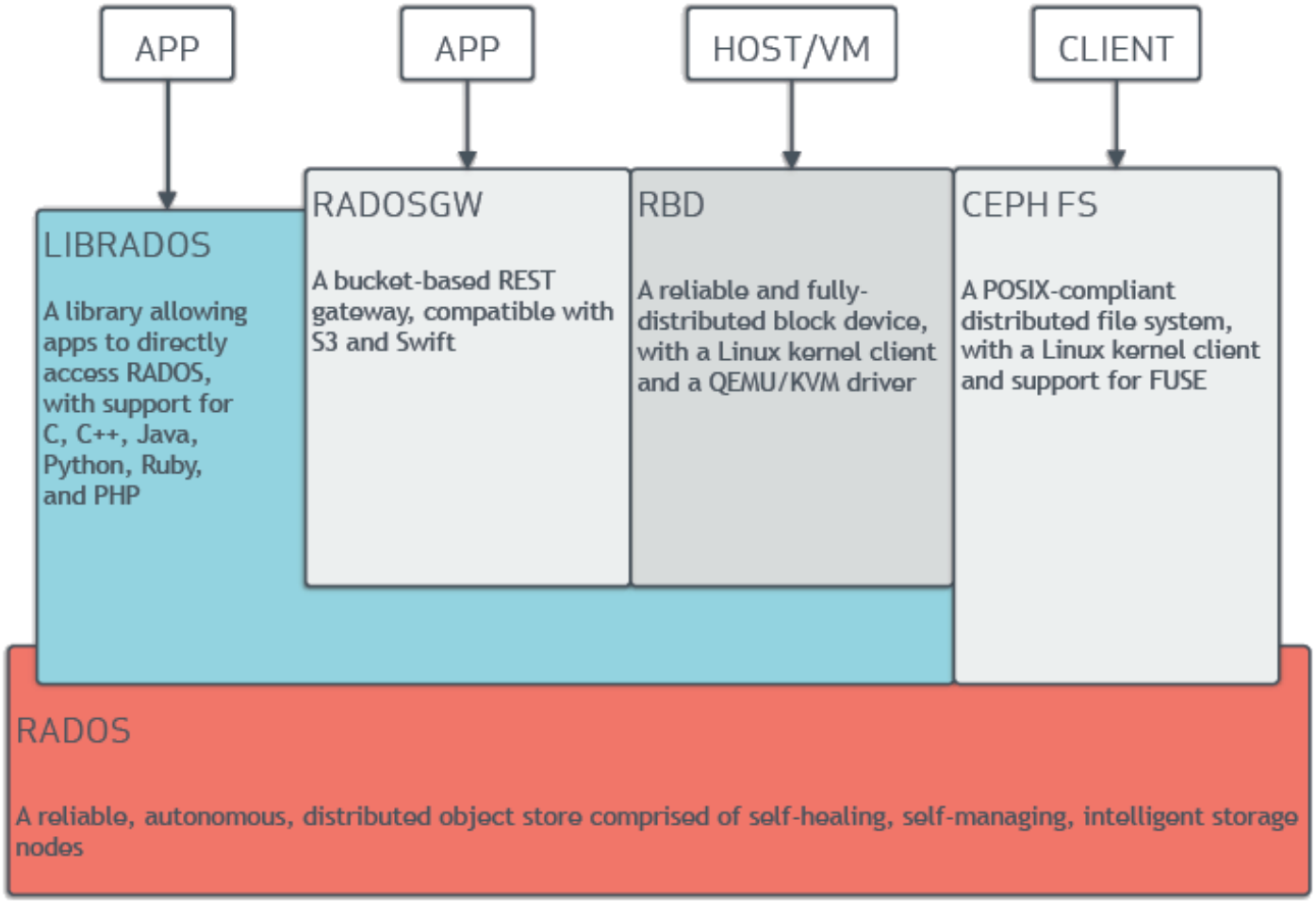
Ceph Features

- **Hardware Agnostic**
  Runs on commodity servers with directly attached storage. Not locked in a specified hardware platform.
- **Flexible**
  Can define pools of storage with differenct redundancy rules, disk types, geographic placement, depending on user requirements.
- **Scalable**
  In both bandwidth and capacity. No metadata servers. Silent clients do not generate network traffic/cluster load.
- **Self-Healing**
  Recovers automatically after disk or server failures.

Ceph: Unified File Storage

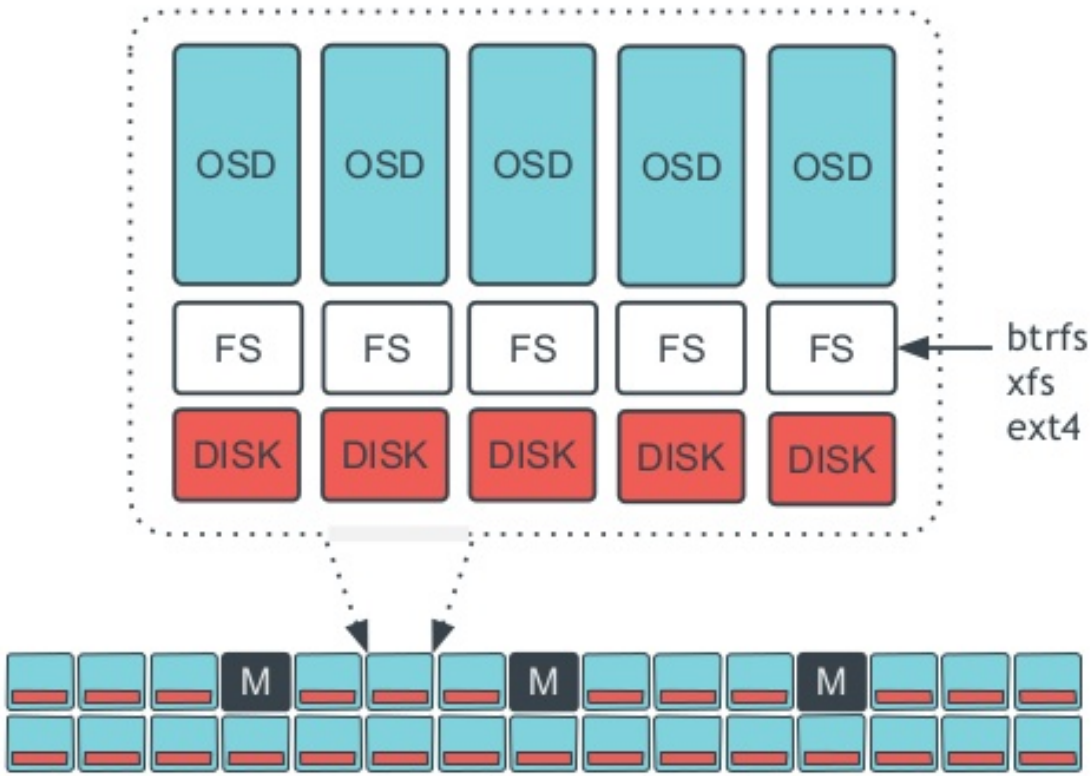
- **Object**
  - Native LIBRADOS (Reliable Autonomic Distributed Object Store)
  - RADOS Gateway provides S3/Swift REST API compatibility
- **Block**
  - Linux kernel (krbd) and KVM(librbd) support
  - provides snapshotting and cloning capabilities
- **File**
  - CephFS support file system

Ceph Stack


APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
RADOS
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

Outline

Ceph Cluster



Monitors

- **Maintain the cluster map**
  - MON map
  - OSD map
  - MDS map
  - PG map
  - CRUSH map
- **Provide consensus for distributed decision making**
  - Each MON knows about all the other MONs in the cluster
  - First establish a quorum of more than half of MONs so an odd number of MONs is needed in the cluster
  - MONs in a quorum can distribute maps to OSDs and clients. Clients request cluster maps from a MON only when the primary OSD fails
- **Monitors are not in data path**

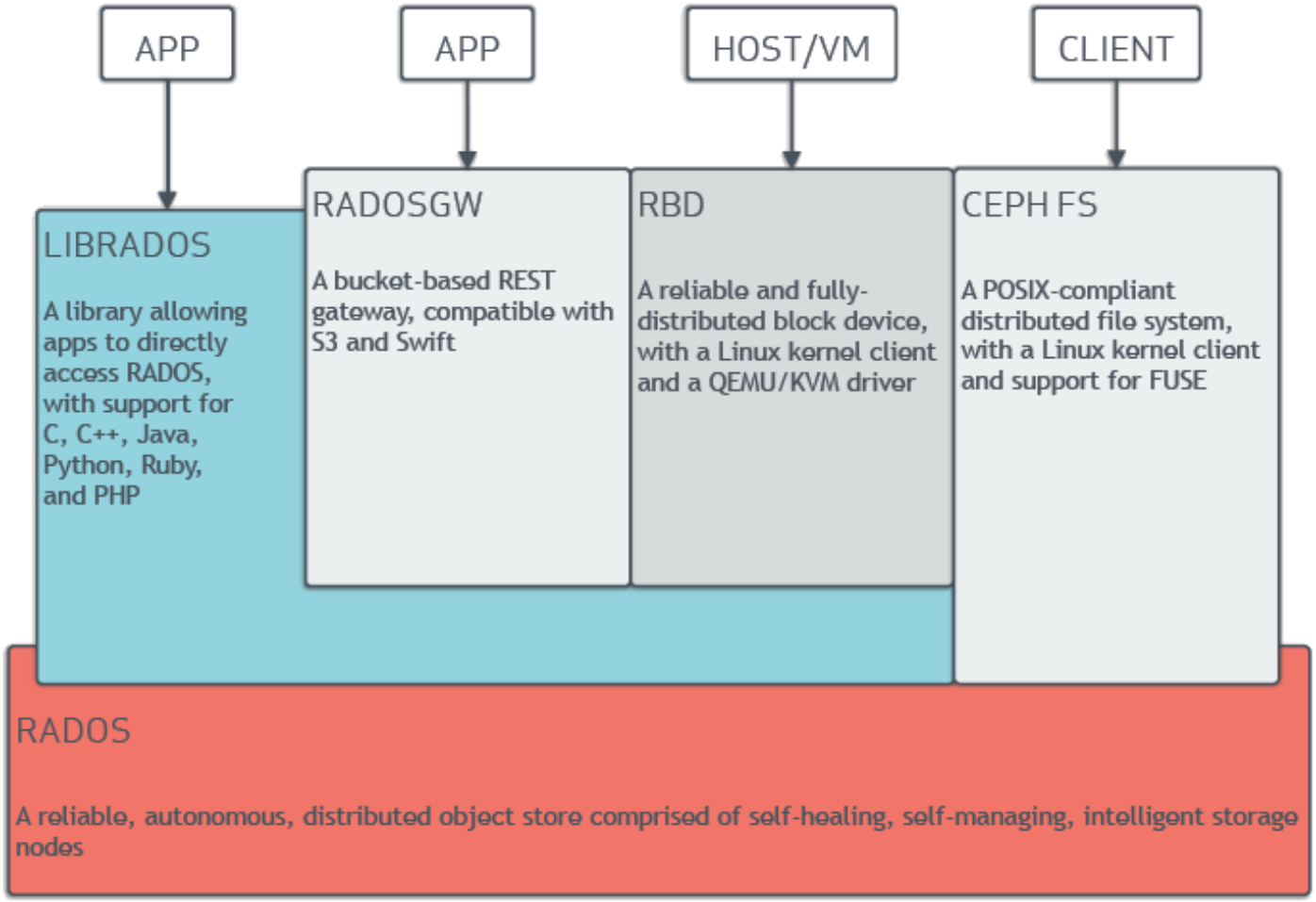
Object Storage Node

- **Object Storage Device (OSD)**
  - Building block of Ceph Storage Cluster
  - One hard disk
  - One Linux File System
  - One Ceph OSD Daemon
- **File System:**
  - File system must be btrfs, xfs or ext4
  - Have the XATTRs enabled

Ceph OSD Daemon

- **Ceph Object Storage Device Daemon**
  - Serve stored objects to clients
- **OSD is primary for some objects**
  - Responsible for replication
  - Responsible for coherency
  - Responsible for re-balancing
  - Responsible for recovery
- **OSD is secondary for some objects**
  - Under the control of the primary
  - Capable of becoming primary
- **Supports extended objects classes**
  - Atomic transactions
  - Synchronization and notifications

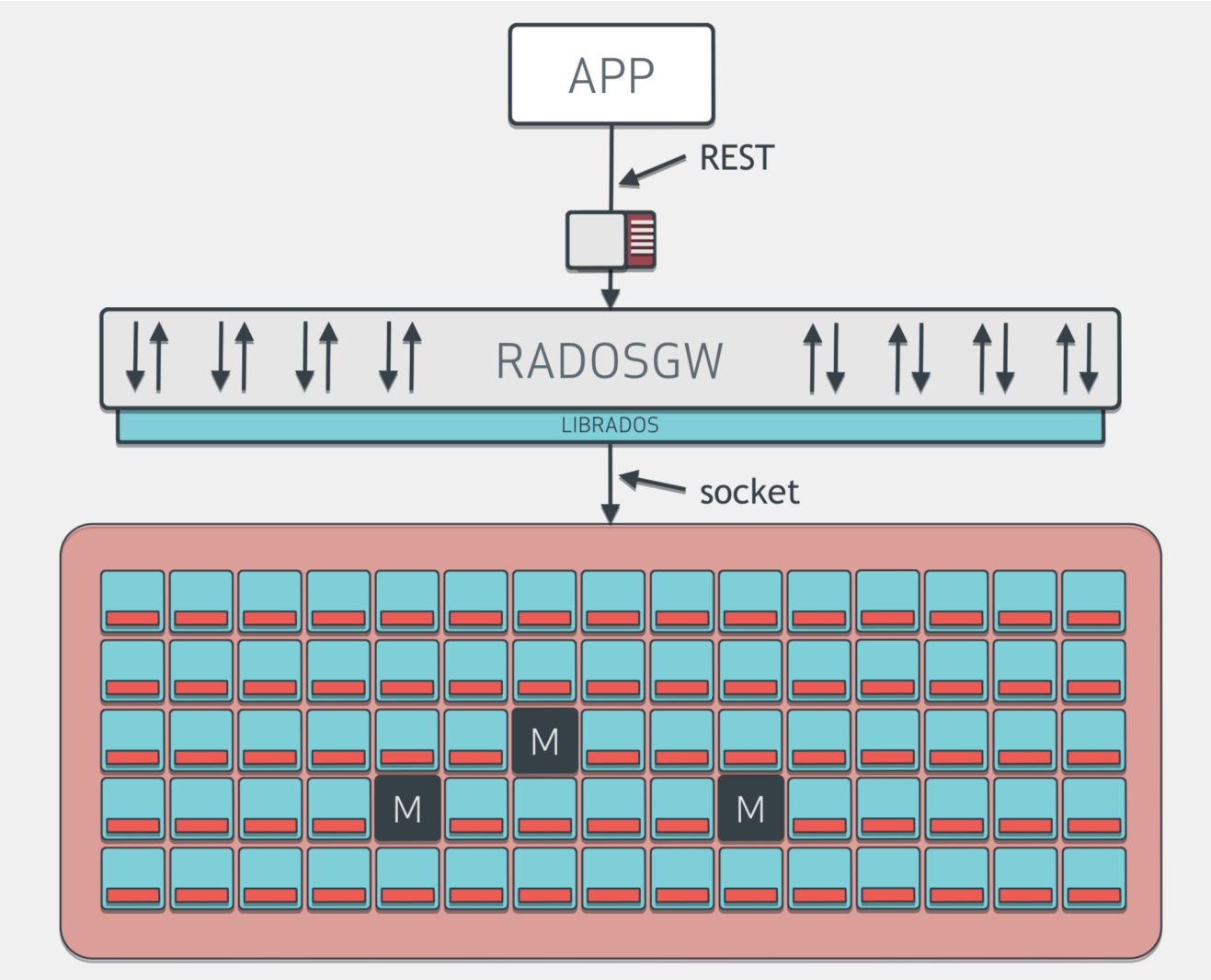
Ceph Stack


APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
RADOS
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

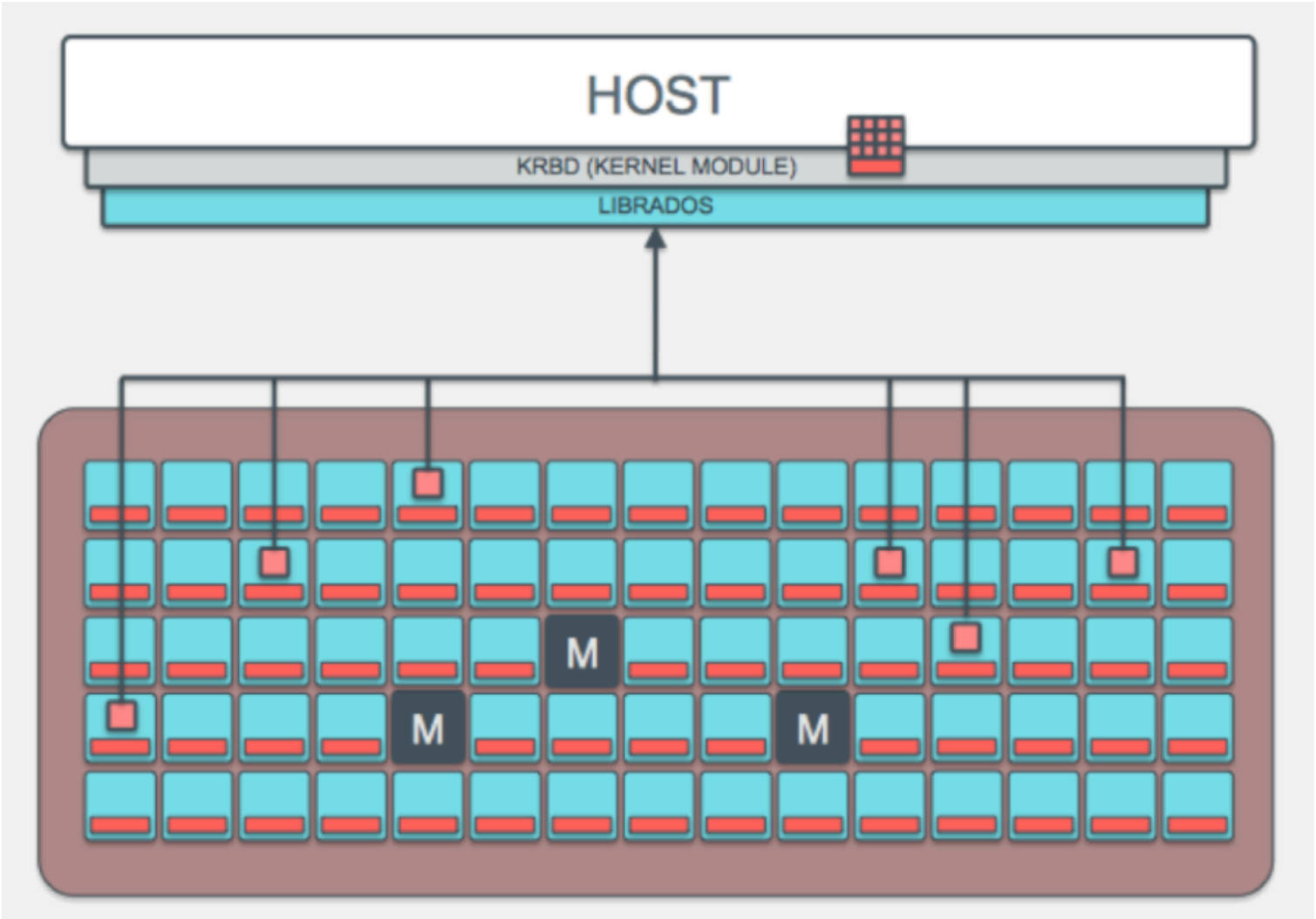
Communication Methods

Ceph Service	Method	Description
N/A	*librados*	Library that provides direct access to RADOS for applications
Ceph Object Gateway	*radosgw*	RESTful APIs for S3 and Swift compatibility
Ceph File System	*libcephfs*	Library that provides access to Ceph Cluster via a POSIX-like interface
Ceph Block Device	*librbd*	Python module, provides file-like access to RBD images

RADOSGW



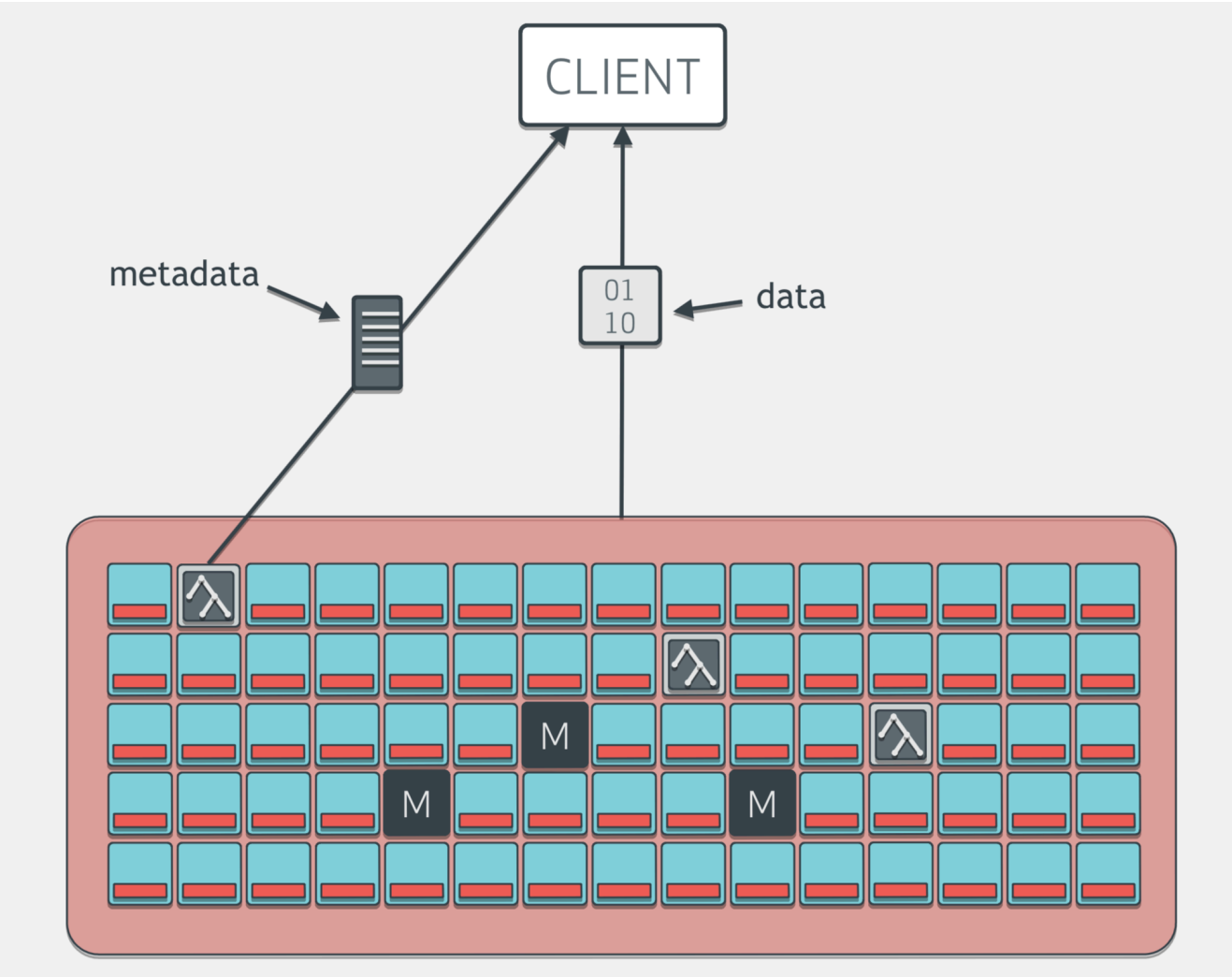
RBD



Metadata Server(MDS)

- **Manages metadata for a POSIX-compliant shared file system**
  - Directory Hierarchy
  - File Metadata(owner, timestamps, mode, etc)
  - Stores metadata in Rados
  - Does not access file content
  - Only required for shared file system
- **The Ceph Metadata Server daemon**
  - Provides the POSIX information needed by file systems that enables CephFS to interact with the Ceph Object Store
  - It remembers where data lives within a tree
  - Client accessing CephFS data first make a request to an MDS, which provides what they need to get files from the right OSDs
- If you aren't running CephFS, MDS daemons do not need to be deployed.

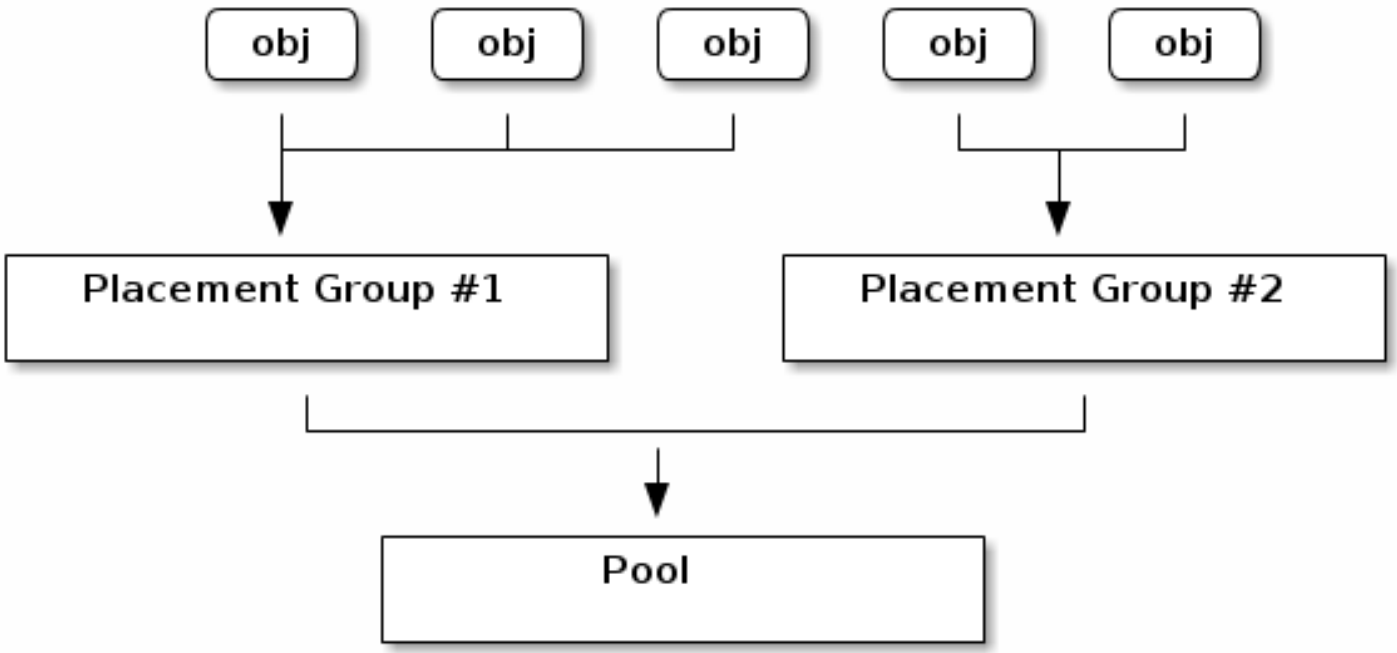
CephFS



Ceph Cluster

```
ubuntu@ip-172-31-58-195:~# ceph -s
  cluster:
    id:     ccb00027-ca55-48cc-bf5a-b6cbf79c9b99
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum ip-172-31-56-105,ip-172-31-57-232,ip-172-31-58-195
    mgr: ip-172-31-56-105(active), standbys: ip-172-31-57-232, ip-172-31-58-195
    mds: cephfs-1/1/1 up  {0=ip-172-31-58-195=up:active}, 2 up:standby
    osd: 4 osds: 3 up, 3 in
    rgw: 1 daemon active

  data:
    pools:   8 pools, 184 pgs
    objects: 268  objects, 37 MiB
    usage:   3.1 GiB used, 87 GiB / 90 GiB avail
    pgs:     184 active+clean
```

Pools and Placement Groups



Pools

- **What are pools?**
  - Pools are logical partitions for storing object data
- **Pools have the following attributes:**
  - Set ownership/access
  - Set number of object replicas
  - Set number of placement groups
  - Set the CRUSH rule set to use
- **The PGs within a pool are dynamically mapped to OSDs**

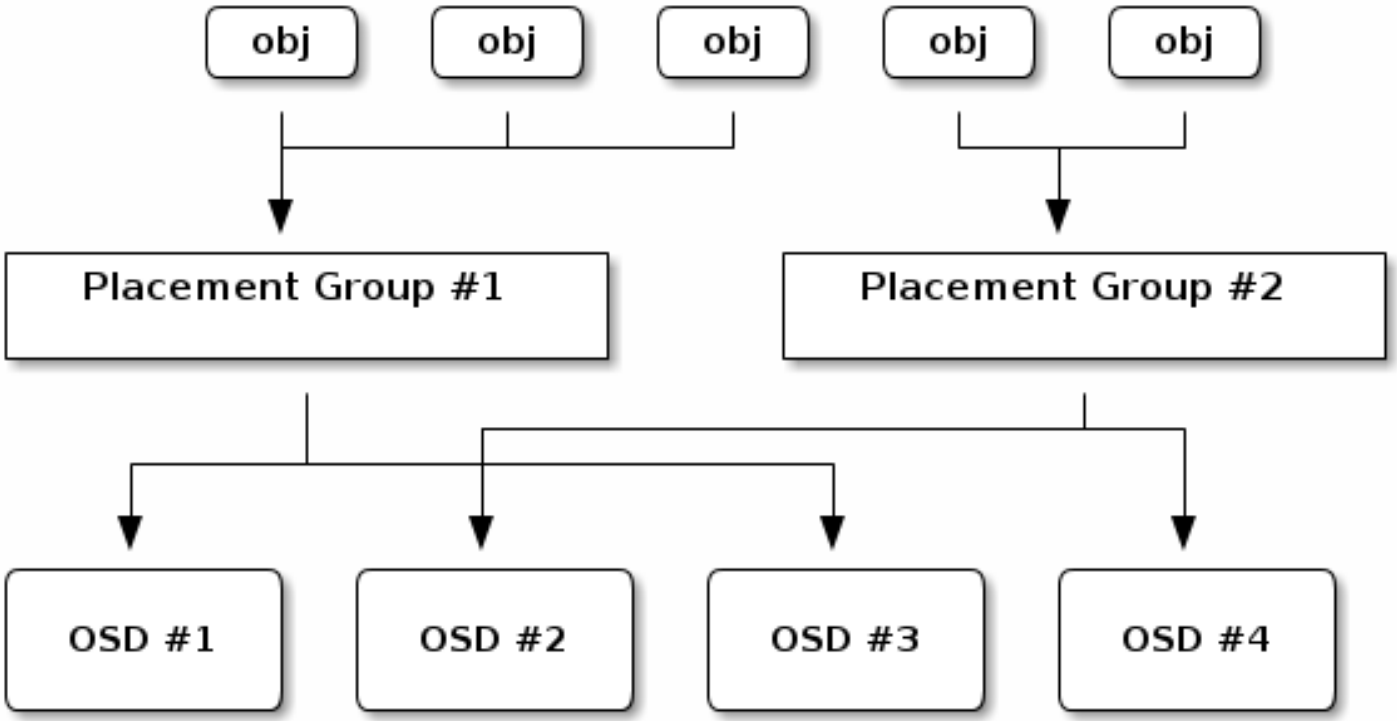
Pools

```
ubuntu@ip-172-31-58-195:~# ceph osd lspools
1 .rgw.root
2 default.rgw.control
3 default.rgw.meta
4 default.rgw.log
5 cephfs_data
6 cephfs_metadata
7 rbd
8 default.rgw.buckets.index
ubuntu@ip-172-31-58-195:~# rados df
POOL_NAME                  USED OBJECTS CLONES COPIES ... RD_OPS       RD WR_OPS      WR
cephfs_data                 28 B       2      0      6 ...      2    1 KiB      2  2 KiB
cephfs_metadata            13 KiB      22      0     66 ...      5    5 KiB     91 46 KiB
...
rbd                        37 MiB      19      0     57 ...   2522 5.3 MiB   1313 81 MiB

total_objects    268
total_used       3.1 GiB
total_avail      87 GiB
total_space      90 GiB
ubuntu@ip-172-31-58-195:~#  ceph osd pool create {pool-name} {pg-num} [{pgp-num}]
```

Placement Groups (PGs)

- **What is a PG?**
  - A PG is a logical collection of objects
  - Objects within the PG are replicated by the same set of OSDs
- **How to choose the PG?**
  - An object's PG is determined by hashing the object name against the number of PGs in the pool
  - A PG's location is determined by CRUSH according to desired protection and placement strategies

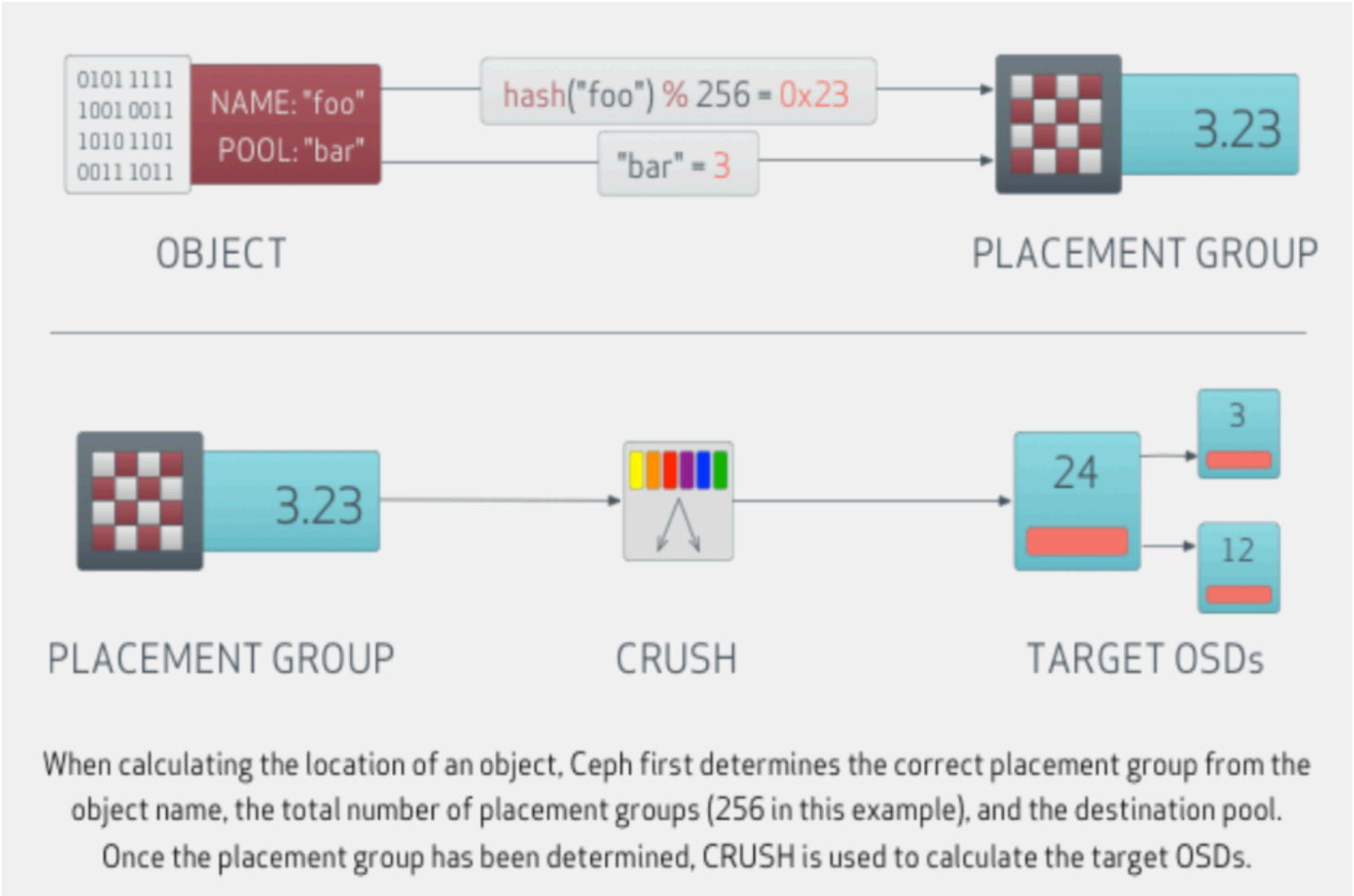
Placement Groups (PGs)



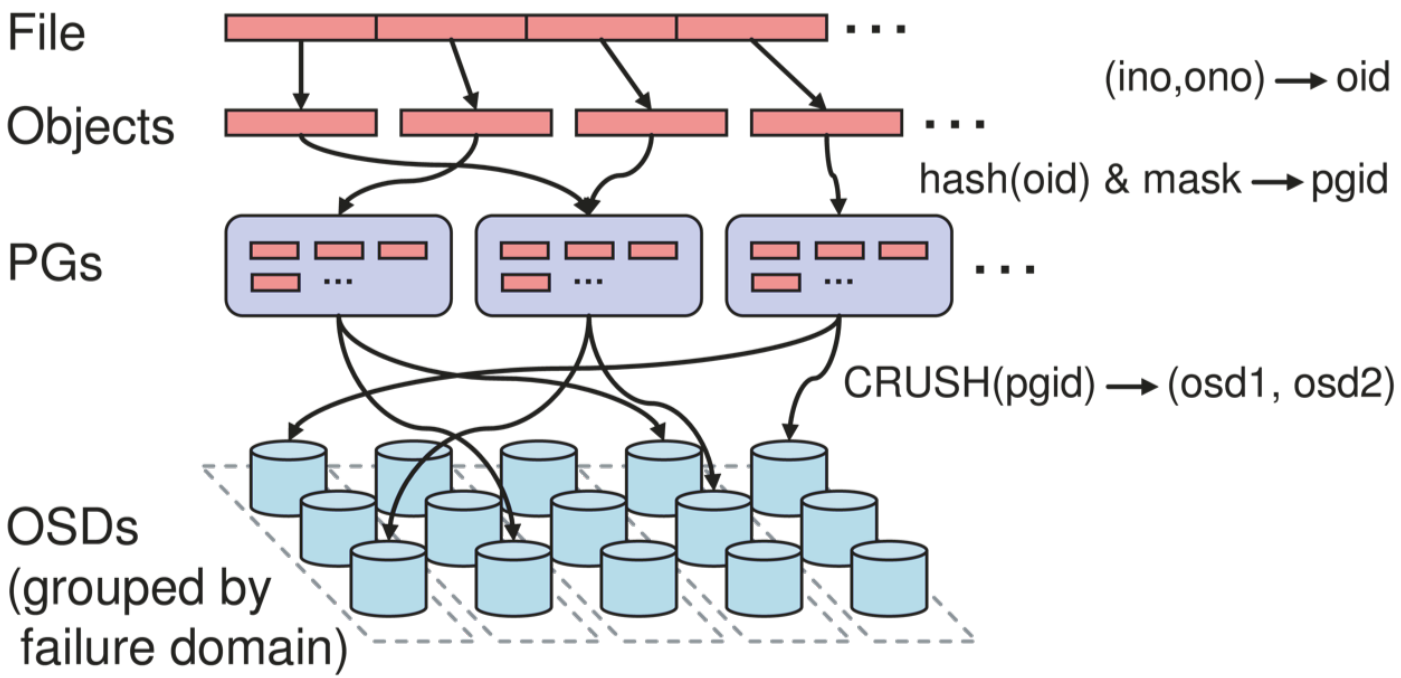
Placement Groups (PGs)

- **Without them**
  - Track placement and metadata on per-object basis
  - Not realistic nor scalable with a million++ objects
- **Extra Benefits**
  - Reduce the number of processes
  - Reduce amount of per object metadata Ceph must track
- **Handling the cluster life cycle**
  - The total number of PGs must be adjusted when growing the cluster
  - As devices leave or join the Ceph cluster, most PGs remain where they are
  - Approximately 50-200 PGs per OSD is recommended -> to balance out memory and CPU requirements and per-OSD load

From Object to OSD



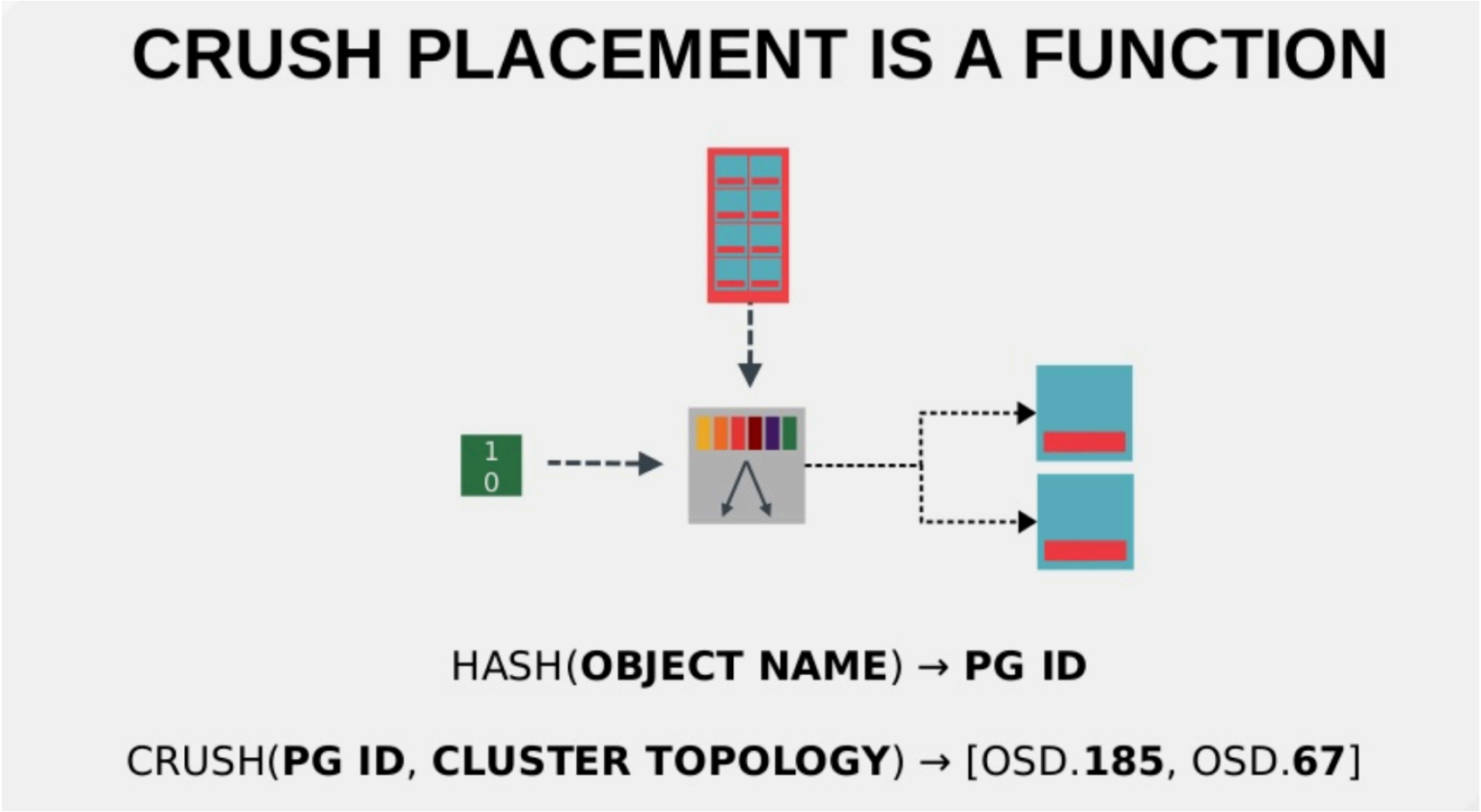
Object Addressing



寻址流程

- **File -> Object**
  - 将用户要操作的 File，映射成 RADOS 能够处理的 Object
  - 本质上是按照 Object 的最大 Size 对 File 进行切分
  - 切分可以使大小不限的 File 变成最大 Size 一致、可以被 RADOS 高效管理的 Object
  - 切分同时也可以让对单一 File 实施的串行处理变成对多个 Object 实施的并行化处理
- **Object -> PG**
  - 将每个 Object 独立的映射到一个 PG 中去
  - 固定的 Hash 算法
- **PG -> OSD**
  - 将作为 Object 的逻辑组织单元的 PG 映射到数据的实际存储单元 OSD
  - 采用 CRUSH 算法

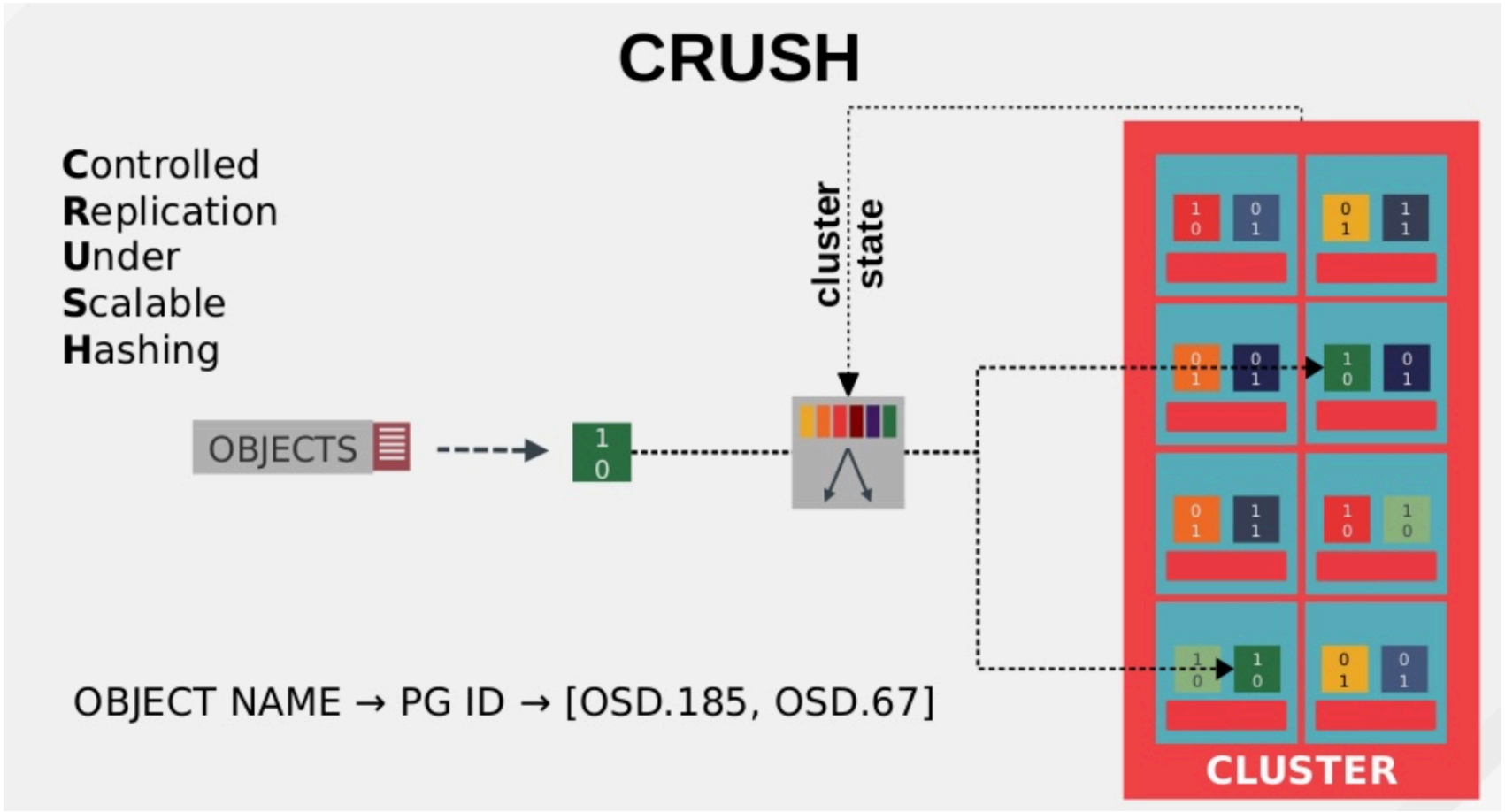
CRUSH Function



CRUSH

- CRUSH(x) -> $(osd_{n1}, osd_{n2}, osd_{n3})$
  - Inputs
    - x is the placement group
    - hierarchical cluster map
    - placement rules
  - Outputs: a list of OSDs
- Advantages
  - Anyone can calculate object location
  - Cluster map infrequently updated

CRUSH


CRUSH
Controlled
Replication
Under
Scalable
Hashing
cluster
state
OBJECTS
1
0
OBJECT NAME → PG ID → [OSD.185, OSD.67]
CLUSTER

CRUSH

CRUSH(Controlled Replication Under Scalable Hashing)

- **Ceph's data distribution mechanism**
- **Pseudo-random placement algorithm**
  - Deterministic function of inputs
  - Clients can compute data location
- **Rule-based configuration**
  - Desired/required replica count
  - Affinity/distribution rules
  - Infrastructure topology
  - Weighting
- **Excellent data distribution**
  - De-clustered placement
  - Excellent data-re-distribution

Key CRUSH Properties

- **No Storage** - Only need to know the cluster topology
- **Fast** - microseconds, even for very large clusters
- **Stable** - very little data movement when topology changes
- **Reliable** - placement is constrained by ***failure domains***
- **Flexible** - replication, erasure codes, complex placement schemes

Hierarchical Cluster Map

- **The cluster map is composed of *device* and *bucket***
  - **device**: basic storage device, OSD
  - **bucket**: container, and contain any number of devices or other buckets
  - **item**: member of bucket, can be device or lower level bucket
  - both device and bucket have numerical identifiers and weight values associated with them
- bucket type:
  - root, region, datacenter, room, row, rack, host
  - you can define your own bucket type and hierarchy relation

Hierarchical Cluster Map

```
# devices
device 0 osd.0 class hdd
device 1 osd.1 class hdd
device 2 osd.2 class hdd
device 3 osd.3 class hdd
```

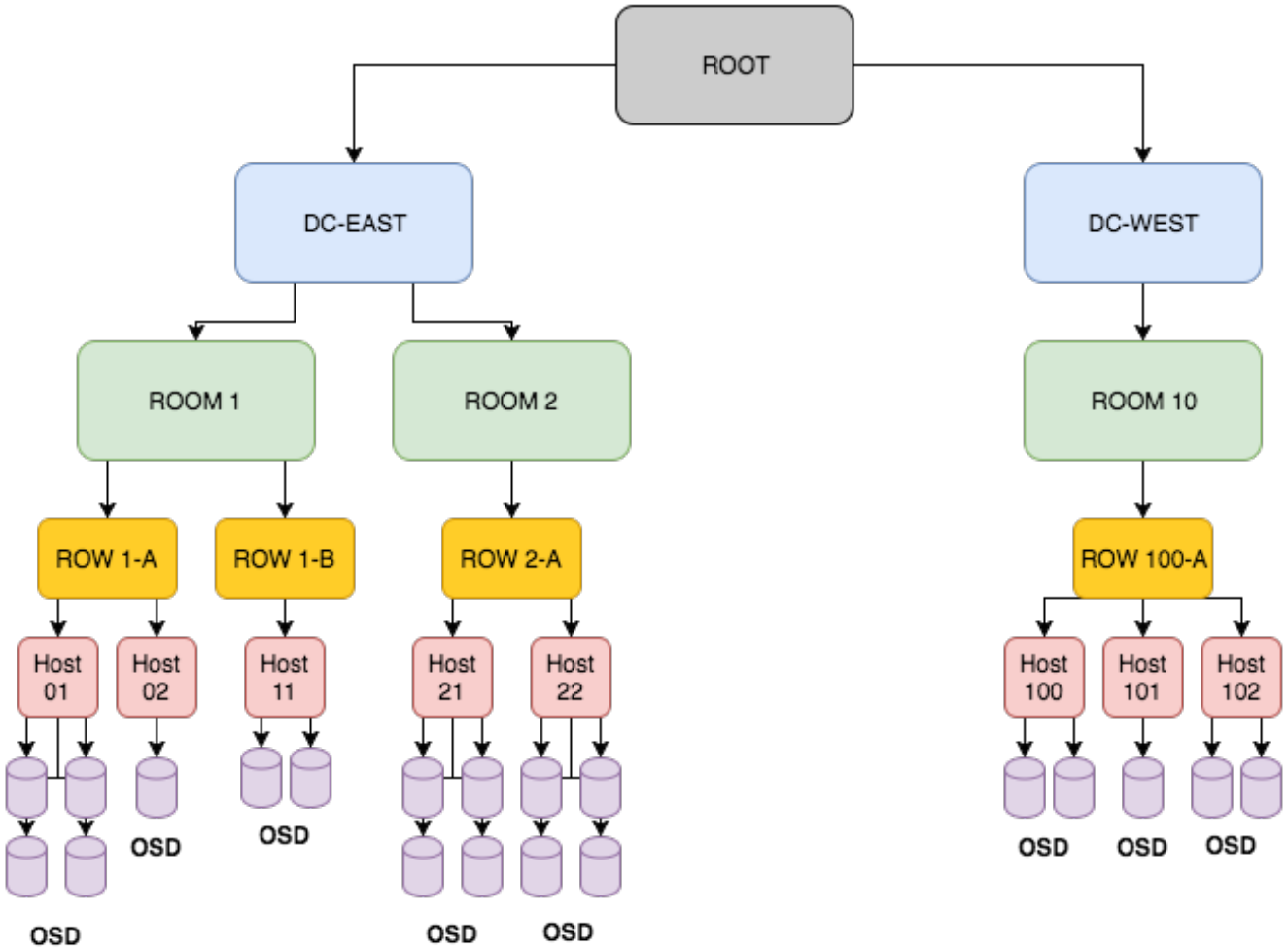
```
# buckets
host node1 {
    id -2
    id -3 class hdd
    # weight 1.000
    alg straw2
    hash 0    # rjenkins1
    item osd.0 weight 1.000
}
```

```
host node2 {
    id -5
    id -6 class hdd
    # weight 1.000
    alg straw2
    hash 0    # rjenkins1
    item osd.1 weight 1.000
}
```

```
host node3 {
    id -7
    id -8 class hdd
    # weight 2.000
    alg straw2
    hash 0    # rjenkins1
    item osd.2 weight 1.000
    item osd.3 weight 1.000
}
```

```
root default {
    id -1    # bucket id, 一般为负数
    # weight 4.000 # sum of item's weight
    alg straw2 # algorithm for random select
    hash 0    # rjenkins1 # hash function type
    item node1 weight 1.000
    item node2 weight 1.000
    item node3 weight 2.000
}
```

Cluster Map Example



Failure Domains

- **CRUSH generates *n* distinct target devices(OSDs)**
  - may be replicas or erasure coding shards
- **Separate replicas across failure domains**
  - single failure should only compromise one replica
  - size of failure domain depends on cluster size
    - disk
    - host(NIC, RAM, PS)
    - rack(ToR switch)
    - row(distribution switch, ...)
    - data center(eletricity...)
  - based on types in CRUSH hierarchy

CRUSH Rules

- **Policy**
  - where to place replicas
  - the failure domain
- **Trivial program**
  - short sequence of imperative commands
  - flexible, extensible
  - not particularly nice for humans

```
1 rule flat {
2     ruleset 0    # ruleset id
3     type replicated
4     min_size 1   # 副本最小数量
5     max_size 10 # 副本最大数量
6     step take root
7     step choose fistn 0 type osd
8     step emit
9 }
```

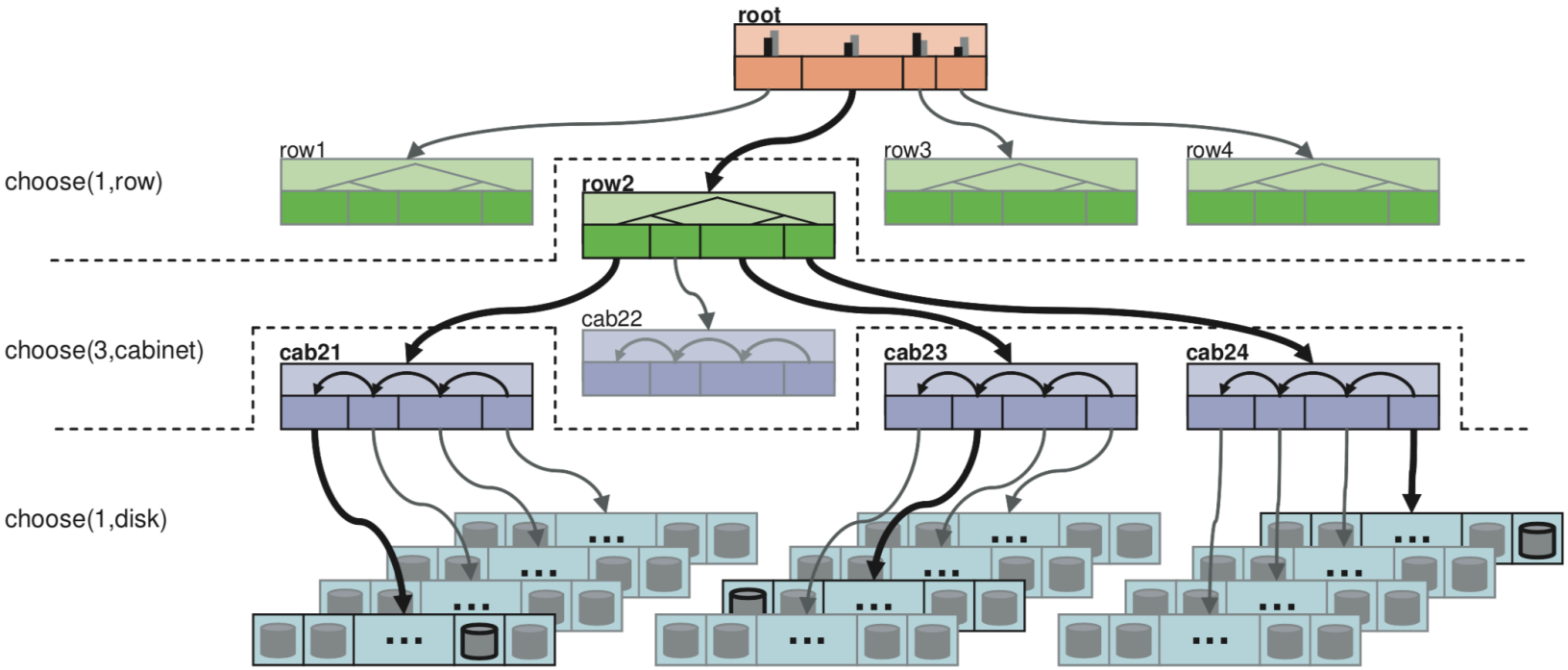
Basic Operation: Step

- **take**: choose a bucket(often root bucket)
- **choose**
  - **choose firstn {num} type {bucket-type}**
    - 选择 num 个 bucket-type 的 item
  - **chooseleaf firstn {num} type {bucket-type}**
    - 先选择 bucket-type 类型的 item，再递归选择叶子节点的 OSD
  - meaning of num parameter
    - if $num == 0$, choose pool-num-replicas buckets(all available)
    - if $num > 0 and < pool - num - replicas$, choose that many buckets
    - if $num < 0$, it means $pool - num - replicas - num$
- **emit**

RuleSet

```
1  rule replicated_ruleset {
2      ruleset 0        # ruleset id
3      type replicated # type replicated or erasure code
4
5      min_size 1
6      max_size 10
7
8      step take root
9      step choose firstn 1 type row
10     step choose firstn 3 type cabinet
11     step choose firstn 1 type osd
12     step emit
13 }
```

CRUSH Rules


root
row1
row2
row3
row4
choose(1,row)
cab22
cab21
cab23
cab24
choose(3,cabinet)
choose(1,disk)

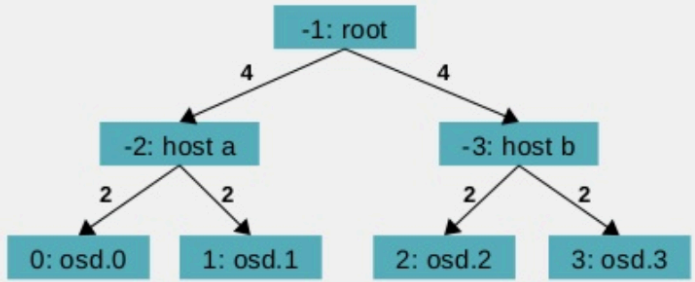
RuleSet Example

```
1  rule ssd-primary {
2      ruleset 5          # ruleset id
3      type replicated # type replicated or erasure code
4
5      min_size 5
6      max_size 10
7
8      step take ssd    # 选择ssd这个root bucket作为输入
9      step chooseleaf firstn 1 type host    # 选择一个host，再递归选择叶子节点的osd
10     step emit
11
12     step take hdd    # 选择hdd这个root bucket作为输入
13     step chooseleaf firstn -1 type host    # 选择总副本数减一个host，再递归选择叶子节点的osd
14     step emit
15 }
```

Bucket Algorithms

- **Goal of CRUSH algorithm**
  - efficiency and scalability of the mapping algorithm
  - minimum data migration to restore a balanced distribution when the cluster changes due to the addition or removal of devices.
- **CRUSH define 4 kinds of buckets to represent internal nodes in the cluster hierarchy**
  - uniform buckets
  - list buckets
  - tree buckets
  - straw buckets
- Each bucket type is based on a **different internal data structure** and utilizes a **different function c(r, x) for pseudo-randomly choosing nested items** during the replica placement process
- Each bucket type represents a different **tradeoff between computation and reorganization efficiency**

How does it work?



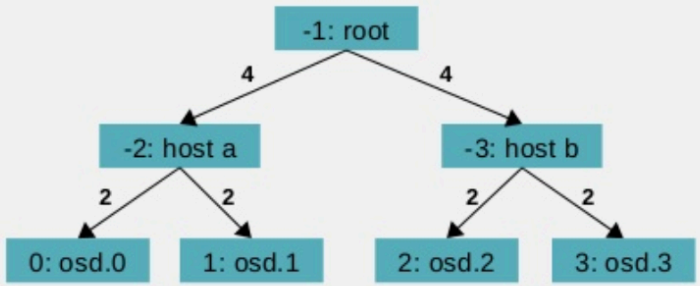
- Weighted tree
- Each node has a unique id

- While CRUSH is executing, it has a "working value" vector → → →

```
rule by-host {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take root
        step choose firstn 0 type host
        step choose firstn 1 type osd
        step emit
}
```

[ ]

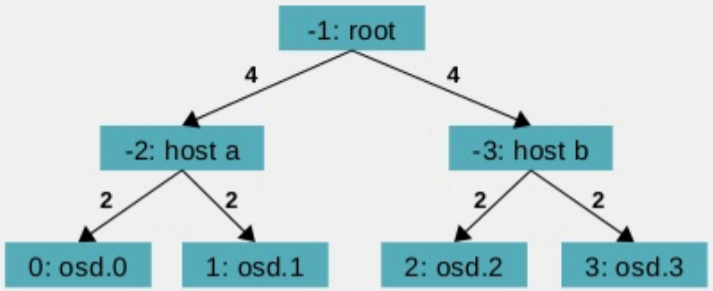
How does it work?



- take root

```
rule by-host {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take root
        step choose firstn 0 type host
        step choose firstn 1 type osd
        step emit
}


[-1]
```

How does it work?

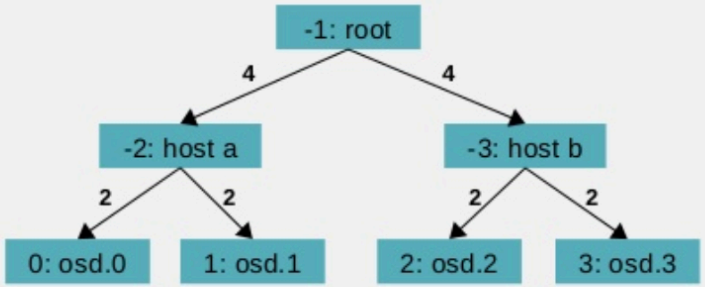


- choose firstn 0 type host
  - nrep=2
- descend from [-1] with
  - x=<whatever>
  - r=0
  - hash(-1, x, 0)
    → -3

```
rule by-host {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take root
        step choose firstn 0 type host
        step choose firstn 1 type osd
        step emit
}
```

```
[-1]
[-3]
```

How does it work?

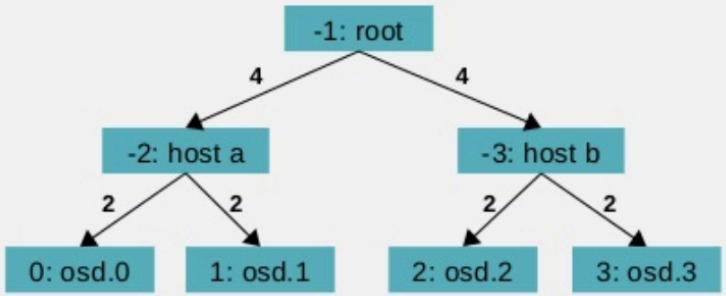


- choose firstn 0 type host
  - nrep=2
- descend from [-1] with
  - x=<whatever>
  - r=**1**
  - hash(-1, x, 1)
    → -3 → dup, reject

```
rule by-host {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take root
        step choose firstn 0 type host
        step choose firstn 1 type osd
        step emit
}
```

```
[-1]
[-3]
```

How does it work?

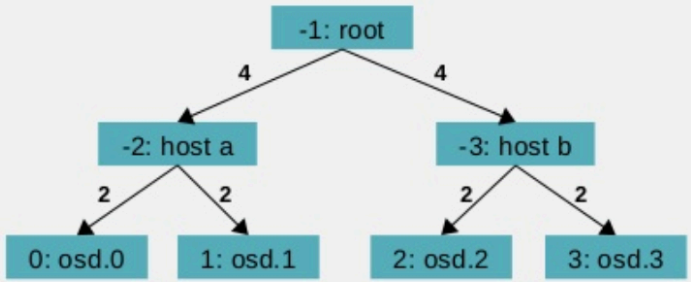


- choose firstn 0 type host
  - nrep=2
- descend from [-1] with
  - x=<whatever>
  - r=2
  - hash(-1, x, 2)
    → -2

```
rule by-host {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take root
        step choose firstn 0 type ho
        step choose firstn 1 type os
        step emit
}
```

```
[-1]
[-3,-2]
```

How does it work?

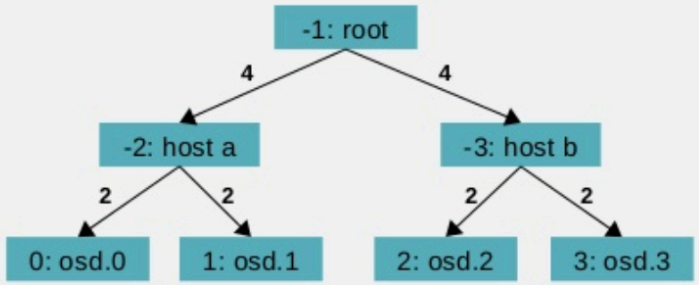


- choose firstn 1 type osd
  - nrep=1
- descend from [**-3**,-2] with
  - x=<whatever>
  - r=0
  - hash(-3, x, 0)
    → 2

```
rule by-host {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take root
        step choose firstn 0 type host
        step choose firstn 1 type osd
        step emit
}
```

```
[-1]
[-3,-2]
[2]
```

How does it work?



- choose firstn 1 type osd
  - nrep=1
- descend from [-3,**-2**] with
  - x=<whatever>
  - r=1
  - hash(-2, x, 1)
    → 1

```
rule by-host {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take root
        step choose firstn 0 type host
        step choose firstn 1 type osd
        step emit
}
```

```
[-1]
[-3,-2]
[2,1]
```

Bucket Algorithms

- **Many algorithms for selecting a child**
  - every internal tree node has a type
  - tradeof between time/computation and rebalancing behavior
  - can mix types within a tree
- **uniform**
  - $hash(nodeid, x, r)\%num_children$
  - fiexed $O(1)$ time
  - addint child shuffles everything
- **straw2**
  - $hash(nodeid, x, r, child)$ for every child
  - scale based on child weight
  - pick the biggest value
  - $O(n)$ time
- **adding or removing child**
  - only moves values to or from that child
  - still fast enough for small n

Cluster Aware

Where should the object be placed

- **Traditional**
  - Centralized interface provides services to the client through a double dispatch
  - Huge bottleneck at petabyte-to-exabyte scale.
- **Ceph**
  - Ceph OSD and Client are *cluster aware*
  - Each OSD knows about other OSD in the cluster
  - OSD can interact directly with other OSD and monitors
  - Client can interact directly with OSD

Cluster Map

Cluster Map indicates the knowledge of the cluster topology.

- **The Monitor Map**
- **The OSD Map**
- **The PG Map**
- **The CRUSH Map**
- **The MDS Map**

Cluster Map: Monoitor Map

```
1 ubuntu@ip-172-31-58-195:~# ceph mon dump
2 dumped monmap epoch 2
3 epoch 2
4 fsid ccb00027-ca55-48cc-bf5a-b6cbf79c9b99
5 last_changed 2018-12-26 18:17:17.540294
6 created 2018-12-26 18:16:44.542161
7 0: 172.31.56.105:6789/0 mon.ip-172-31-56-105
8 1: 172.31.57.232:6789/0 mon.ip-172-31-57-232
9 2: 172.31.58.195:6789/0 mon.ip-172-31-58-195
```

Cluster Map: OSD Map

```
ubuntu@ip-172-31-58-195:~# ceph osd dump
epoch 245
fsid ccb00027-ca55-48cc-bf5a-b6cbf79c9b99
created 2018-12-26 18:17:10.141095
modified 2018-12-29 12:02:43.764006
crush_version 26
...
pool 5 'cephfs_data' replicated size 3 min_size 2 crush_rule 0 object_hash rjenkins pg_num 64
      pgp_num 64 last_change 48 flags hashpspool stripe_width 0 application cephfs
pool 6 'cephfs_metadata' replicated size 3 min_size 2 crush_rule 0 object_hash rjenkins pg_num
      64 pgp_num 64 last_change 48 flags hashpspool stripe_width 0 application cephfs
max_osd 4
osd.0 up   in  weight 1 up_from 98 up_thru 244 down_at 94 last_clean_interval [90,93)
      172.31.56.105:6801/2052 172.31.56.105:6802/2052 172.31.56.105:6803/2052
      172.31.56.105:6804/2052 exists,up 2ee23b61-eca4-41e5-a6b0-277bfd6c48a1
...
```

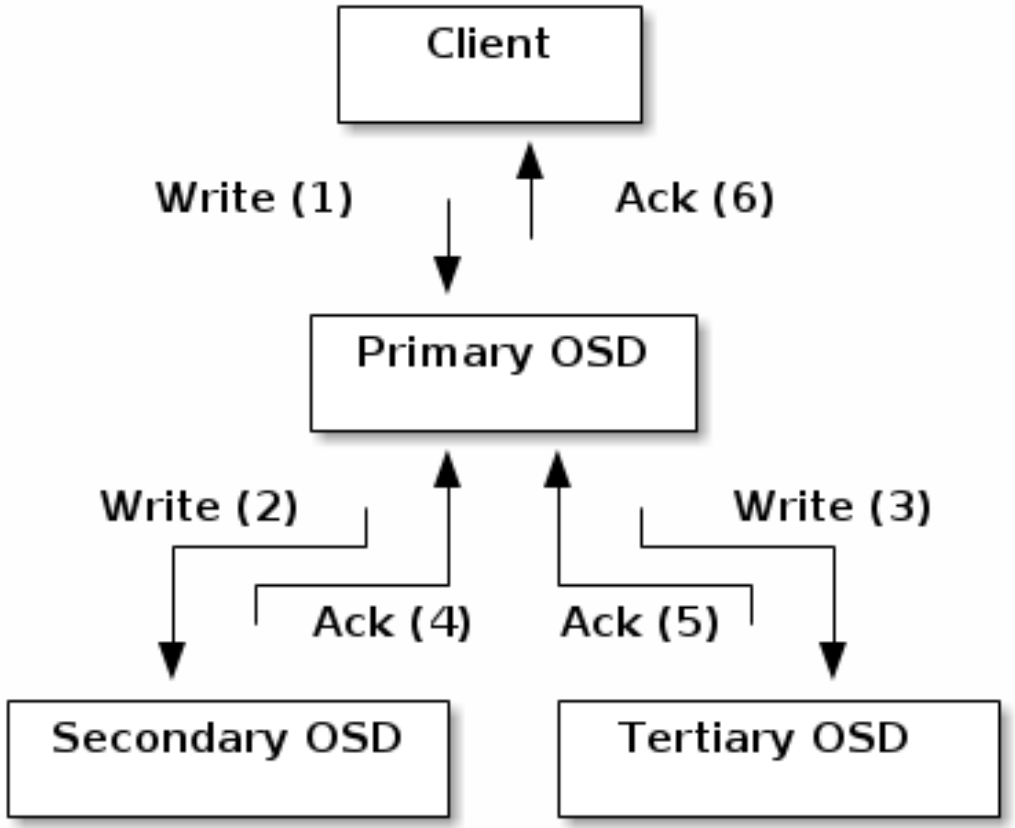
Cluster Map: CRUSH Map

```
ubuntu@ip-172-31-58-195:~# ceph osd tree
ID CLASS WEIGHT  TYPE NAME                 STATUS REWEIGHT PRI-AFF
-1       0.11597 root default
-3       0.02899     host ip-172-31-56-105
 0   ssd 0.02899         osd.0                 up  1.00000 1.00000
-5       0.02899     host ip-172-31-57-232
 1   ssd 0.02899         osd.1                 up  1.00000 1.00000
-7       0.02899     host ip-172-31-58-195
 2   ssd 0.02899         osd.2                 up  1.00000 1.00000
-9       0.02899     host ip-172-31-61-170
 3   ssd 0.02899         osd.3               down        0 1.00000
```

Cluster Map

- 由若干个 monitor 共同负责整个 Ceph 集群中所有的 OSD 状态的发现与记录，并且共同形成 cluster map 的 master 版本，然后扩散到整体 OSD 以及 client
- OSD 使用 cluster map 进行数据的维护，client 使用 cluster 进行数据的寻址
- monitor 并不主动轮询各个 OSD 的当前状态。正相反，OSD 需要向 monitor 上报状态信息。常见的上报有两种情况
  - 新的 OSD 被加入集群
  - 某个 OSD 发现自身或者其他 OSD 发生异常
- 在收到这些上报信息后，monitor 将更新 cluster map 信息并加以扩散

Data Operation Procedure

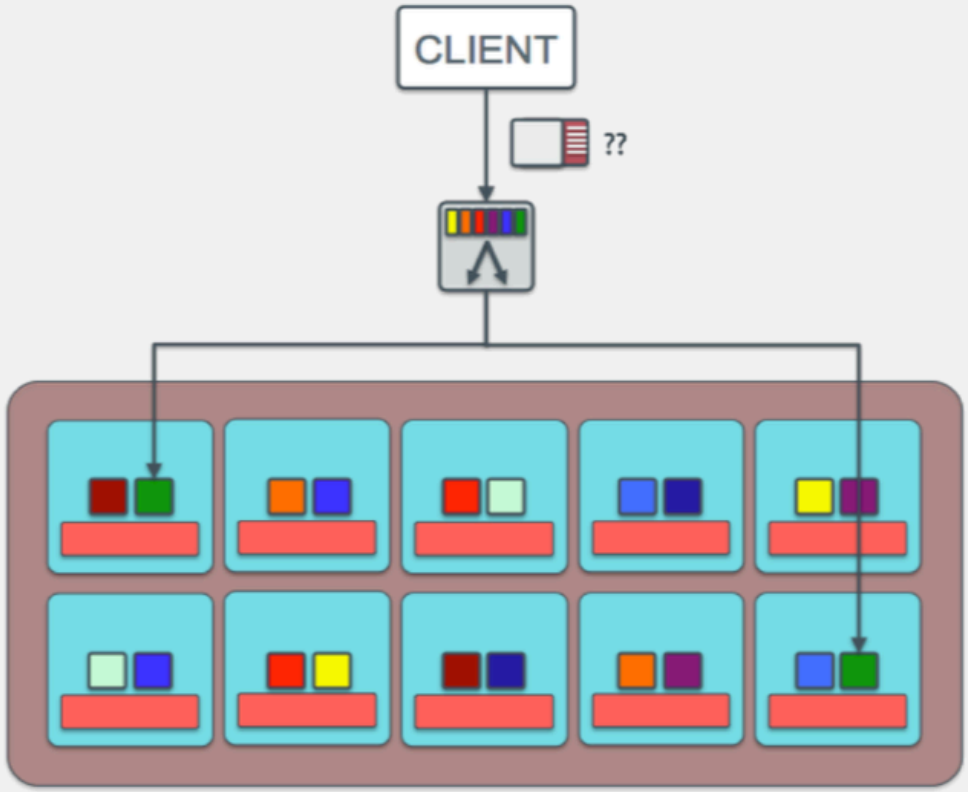


Peering and Recovery

The Ceph Object Store is dynamic

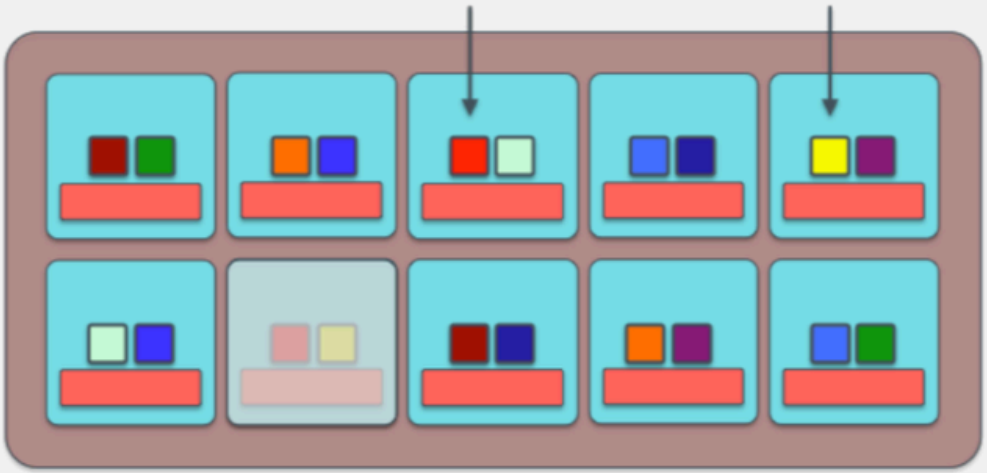
- **Failure is the norm rather than the exception**
- **The cluster map records cluster state at a point in time**
- **The cluster map contains the OSDs status(up/down, weight, IP)**
  - OSDs cooperatively migrate data
  - They do so to achieve recovery based on CRUSH rules
  - Any cluster map update potentially triggers data migration

CRUSH



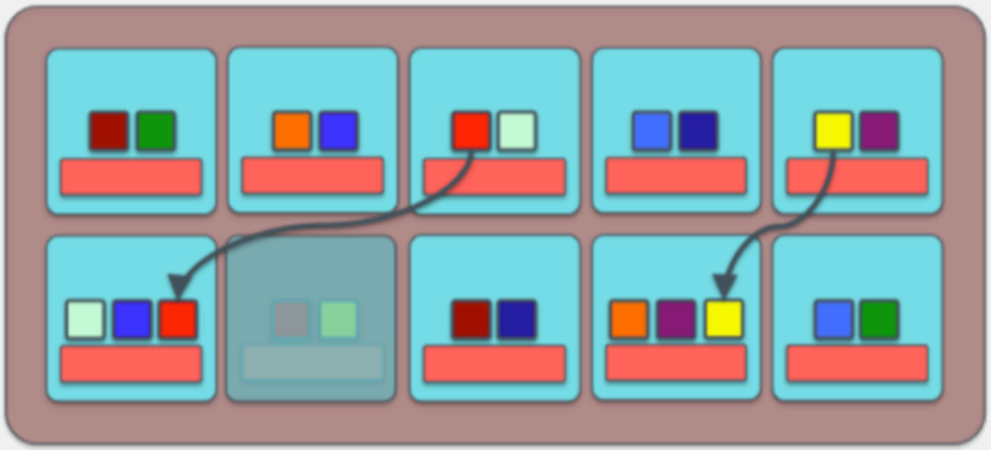
When it comes time to store an object in the cluster (or retrieve one), the client calculates where it belongs.

CRUSH



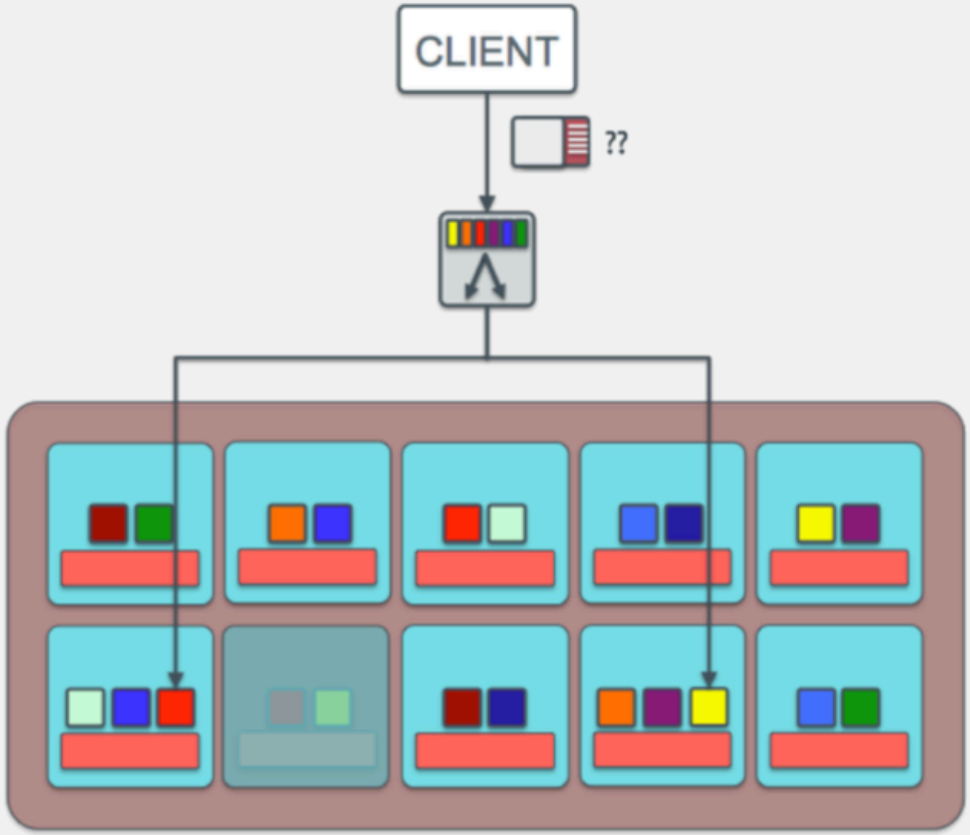
- What happens, though, when a node goes down?
  - The OSDs are always talking to each other (and the monitors)
  - They know when something is wrong
    - The 3^{rd} & 5^{th} nodes noticed that 2^{nd} node on the bottom row is gone
    - They are also aware that they have replicas of the missing data

CRUSH



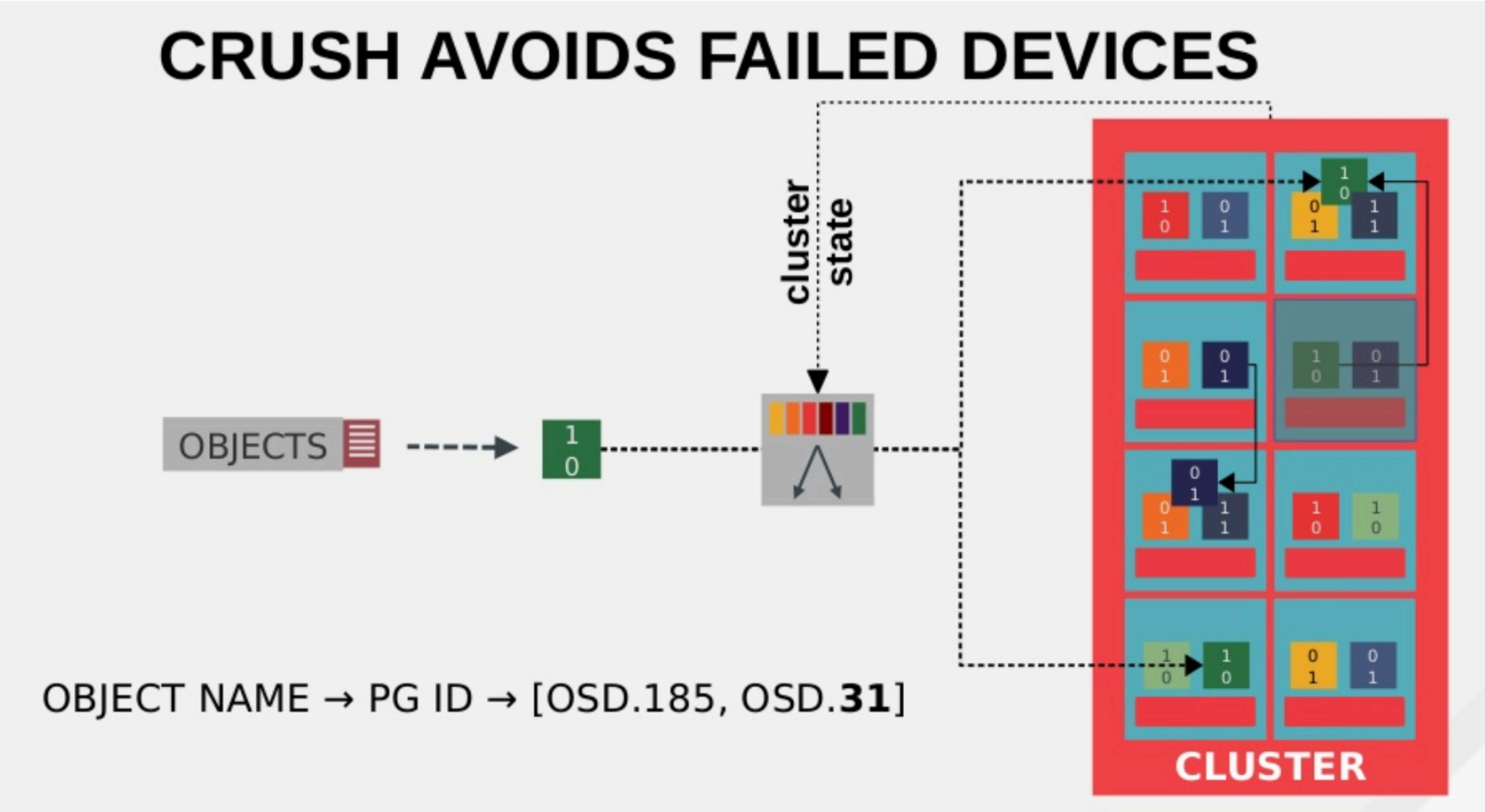
- The OSDs collectively
  - Use the CRUSH algorithm to determine how the cluster should look based on its new state
  - and move the data to where clients running CRUSH expect it to be

CRUSH



- Placement is calculated rather than centrally controlled
- Node failures are transparent to clients

OSD Fail


CRUSH AVOIDS FAILED DEVICES
cluster state
OBJECTS
1
0
OBJECT NAME → PG ID → [OSD.185, OSD.31]
CLUSTER

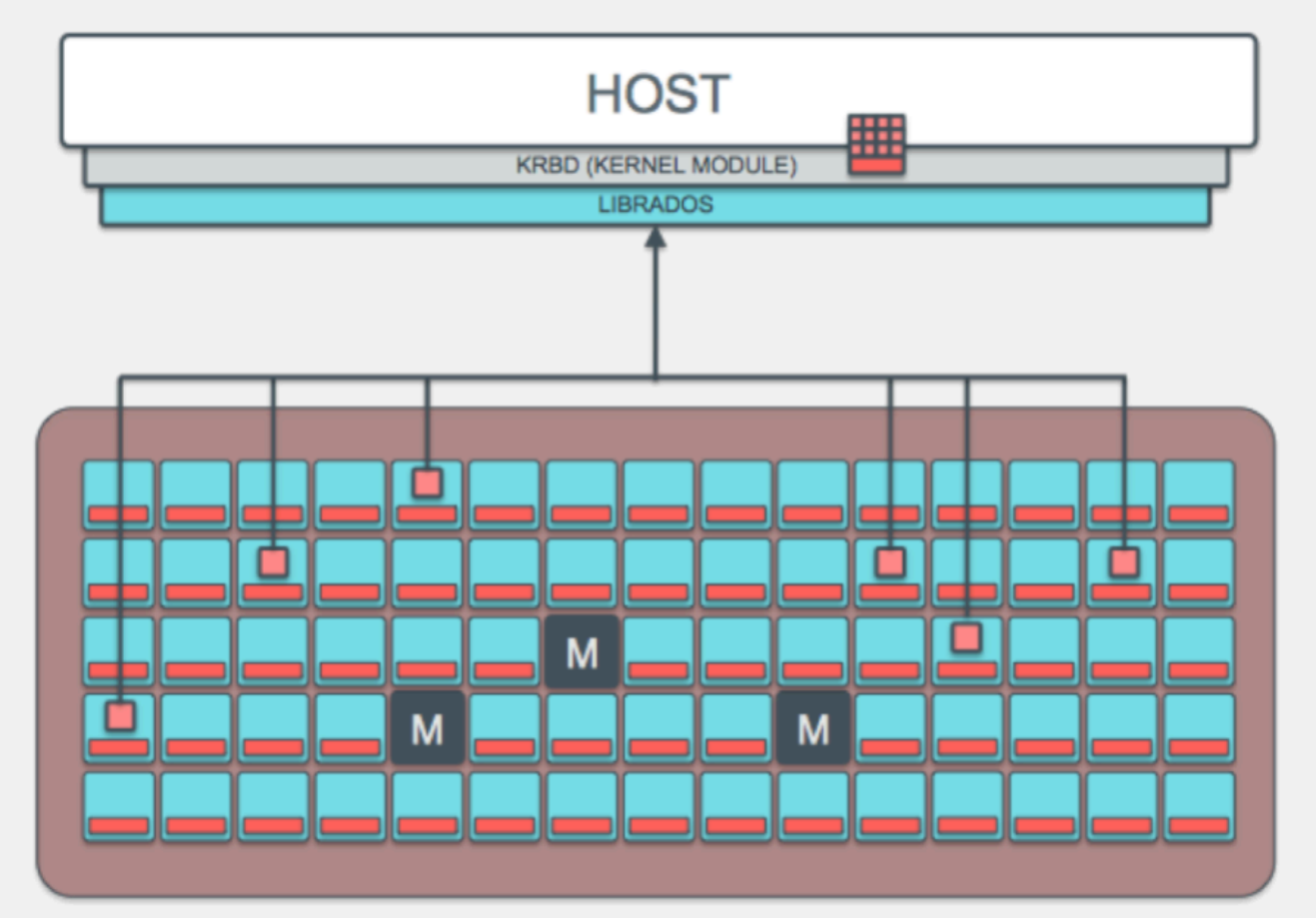
Python S3 Example

```python
# creating a connection
conn = boto.connect_s3(aws_access_key_id = access_key,
    aws_secret_access_key = secret_key,
    host = 'objects.dreamhost.com',
    calling_format = boto.s3.connection.OrdinaryCallingFormat())

# listing owned buckets
for bucket in conn.get_all_buckets():
    print "{name}\t{created}".format(name = bucket.name, created = bucket.creation_date)
# mahbuckat1    2011-04-21T18:05:39.000Z
# mahbuckat2    2011-04-21T18:05:48.000Z

# listing a bucket's content
for key in bucket.list():
    print "{name}\t{size}\t{modified}".format(name = key.name, size = key.size, modified = key.
      last_modified)
# myphoto1.jpg 251262  2011-08-08T21:35:48.000Z
# myphoto2.jpg 262518  2011-08-08T21:38:01.000Z

# creating an object
key = bucket.new_key('hello.txt')
key.set_contents_from_string('Hello World!')

# download an object to a file
key = bucket.get_key('perl_poetry.pdf')
key.get_contents_to_filename('/home/larry/documents/perl_poetry.pdf')
```

Create and Mount RBD

```
1  # create a block device pool
2  $ ceph osd pool create rbd 128
3  $ rbd pool init
4
5  # create a block device image(1024MB)
6  $ rbd create --size 1024 foo
7  $ rbd list
8  foo
9
10 # map a block device
11 $ sudo rbd device map rbd/foo --id admin
12 /dev/rbd0
13 $ rbd device list
14 id pool image snap device
15 0  rbd  foo   -    /dev/rbd0
16
17 # format the block device and create a file system
18 $ sudo mkfs.ext4 -m0 /dev/rbd0
19 $ sudo mkdir /mnt/rbd
20 $ sudo mount /dev/rbd/rbd/foo /mnt/rbd
21
22 # 查看挂载情况
23 $ df -h
24 ...
25 /dev/rbd0            976M  2.6M  958M    1% /mnt/rbd
```

RBD: Native

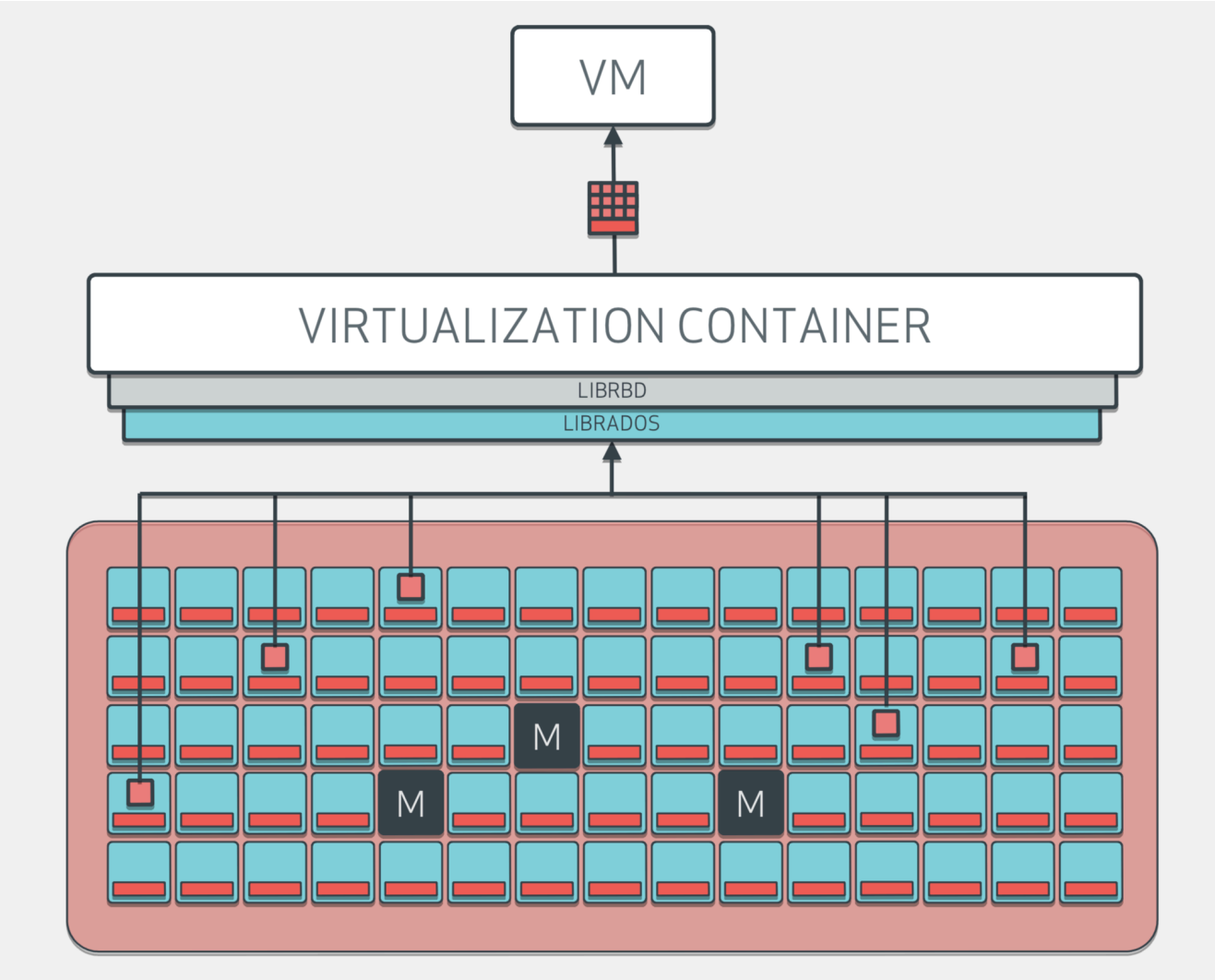


Machines can mount an RBD image using native linux kernel drivers

RBD: Virtual Machines

- **The *librbd* library**
  - Maps data blocks into objects for storage in the Ceph Object Store
  - Inherit *librados* capabilities such as snapshots and clones
- **Virtualization containers**
  - KVM or QEMU can use VM images that are stored in RADOS
  - Virtualization containers can also use RBD block storage in OpenStack and CloudStack platforms
- **Ceph based VM Images**
  - Are striped across the entire cluster
  - Allow simultaneous read access from different cluster nodes

RBD: Virtual Machines



Access RBD using librbd

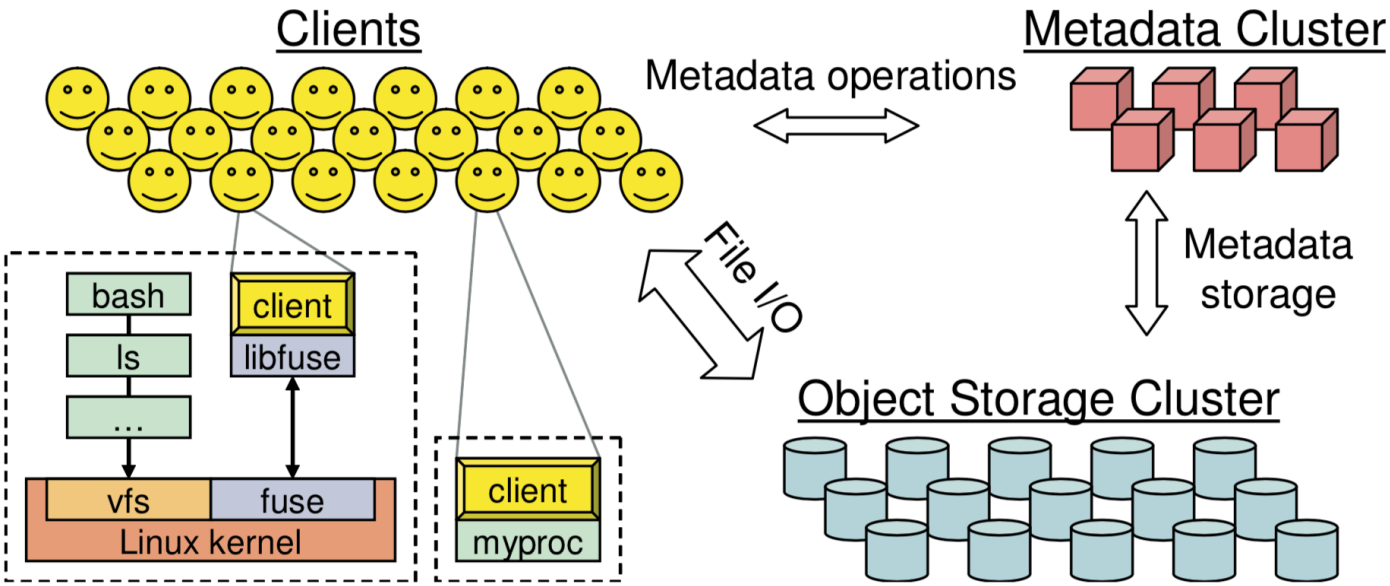
```
with rados.Rados(conffile='my_ceph.conf') as cluster:
    with cluster.open_ioctx('mypool') as ioctx:
        rbd_inst = rbd.RBD()
        size = 4 * 1024**3  # 4 GiB
        rbd_inst.create(ioctx, 'myimage', size)
        with rbd.Image(ioctx, 'myimage') as image:
            data = 'foo' * 200
            image.write(data, 0)
```

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: ceph-rbd-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteOnce
  rbd:
    monitors:
    - '172.31.56.105:6789'
    - '172.31.57.232:6789'
    - '172.31.58.195:6789'
    pool: rbd
    image: foo
    user: admin
    secretRef:
      name: ceph-secret
    fsType: ext4
    readOnly: false
```

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: ceph-rbd-pv-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ceph-rbd-pv-pod1
spec:
  containers:
  - name: rbd-rw
    image: kubernetes/pause
    volumeMounts:
    - name: ceph-rbd-vol1
      mountPath: /mnt/rbd
      readOnly: false
  volumes:
  - name: ceph-rbd-vol1
    persistentVolumeClaim:
      claimName: ceph-rbd-pv-claim
```

System Architecture



System Architecture

- **Clients**
  - Export a near-POSIX file system interface
- **Cluster of OSDs**
  - Store all data and metadata
  - Communicate directly with clients
- **Metadata Server Cluster**
  - Manages the namespace (files + directories)
  - Security, consistency and coherence

Key Ideas

- **Seperate data and metadata management tasks**
- **Dynamic partitioning of metadata data tasks inside metadata cluster**
  - Avoid hot spots
- **Let OSDs handle file migration and replication tasks**

Create and Mount CephFS

```
## creating pools
$ ceph osd pool create cephfs_data 128
$ ceph osd pool create cephfs_metadata 128
## creating a file system
$ ceph fs new cephfs cephfs_metadata cephfs_data
$ ceph fs ls
name: cephfs, metadata pool: cephfs_metadata, data pools: [cephfs_data ]
```

```
## there are 2 ways to mount a file system
## ceph-fuse is most of the time slower than the cephfs kernel module

## mount cephfs with the kernel driver
$ sudo mkdir /mnt/mycephfs
$ sudo mount -t ceph 192.168.0.1:6789:/ /mnt/mycephfs

## mount cephfs using fuse
$ sudo mkdir /home/username/cephfs
$ sudo ceph-fuse -m 192.168.0.1:6789 /home/username/cephfs
```

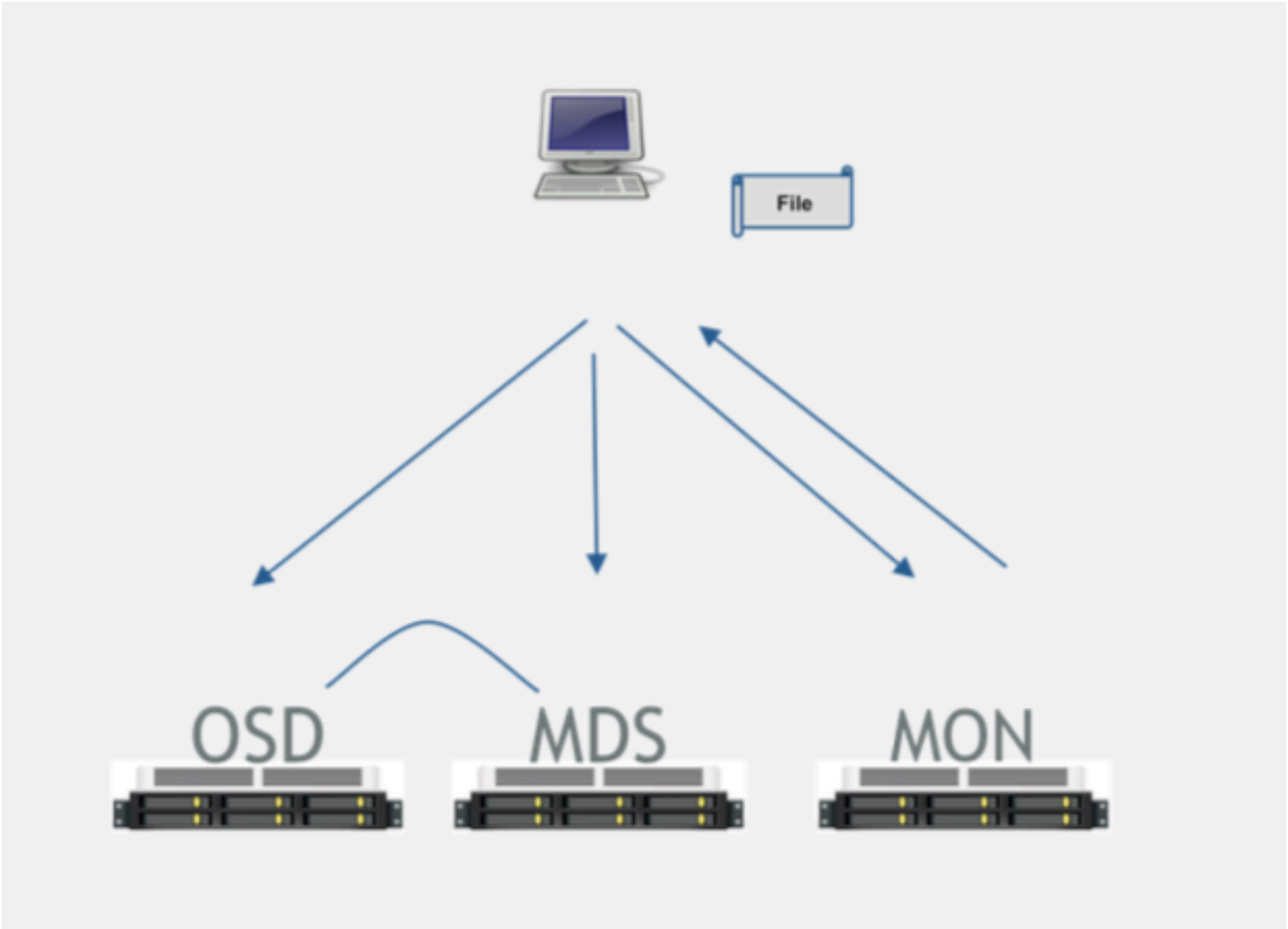
MDS High Availabilty

- MDSs can be running in two modes
  - active
  - standby
- A standby MDS can become active
  - If the previously active daemon goes away
- Multiple active MDSs for load balancing
  - Are a possibility
  - This configuration is currently not supported/recommended

MDS Functionality

- The client learns about MDSs and OSDs from MON
  - via MON Map, OSD Map and MDS Map
- Clients talk to MDS for access to Metadata
  - Permission bits, ACLs, file ownership, etc.
- Clients talk to OSD for access to data
- MDSs themselves store all their data in OSDs
  - In a separate pool called metadata

MDS Functionality



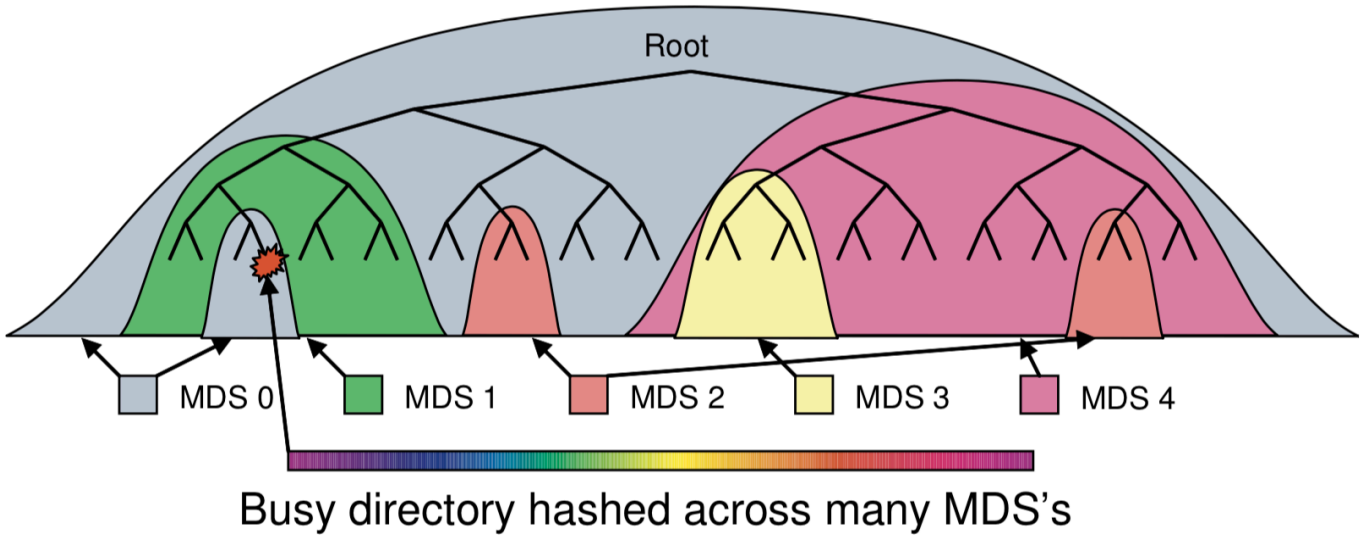
Client Access Example

- Client sends open request to MDS
- MDS returns capability, file inode, file size and stripe information
  - capability specifies authorized operations on file
- Client read/write directly from/to OSDs
- MDS manages the capability
- Client sends close request, relinquishes capability, provides details to MDS

Metadata management

- Metadata operations often make up as much as half of file system workloads...
- **Dynamic Subtree Partitioning**
  - Lets Ceph dynamically share metadata workload among tens or hundreds of metadata servers(MDSs)
  - Sharing is dynamic and based on current access patterns
- Results in near-linear performance scaling in the number of MDSs

Mapping Subdirectories to MDSs



```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: cephfs-pv
spec:
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteMany
  cephfs:
    monitors:
    - '172.31.56.105:6789'
    - '172.31.57.232:6789'
    - '172.31.58.195:6789'
    user: admin
    secretRef:
      name: ceph-secret
    readOnly: false
```

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: cephfs-pv-claim
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 10Gi
```

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: cephfs-pvc-pod
  name: cephfs-pv-pod1
spec:
  containers:
  - name: cephfs-pv-busybox1
    image: busybox
    command: ["sleep", "60000"]
    volumeMounts:
    - mountPath: "/mnt/cephfs"
      name: cephfs-vol1
      readOnly: false
  volumes:
  - name: cephfs-vol1
    persistentVolumeClaim:
      claimName: cephfs-pv-claim
```

Conclusion

- Scalability, Reliablity, Performance
- Seperation of data and metadata
- Object based storage
- Pseudo-Random data distribution function(CRUSH)