Ceph

A complete introduction.

Itinerary

- What is Ceph?
  - What’s this CRUSH thing?
- Components
- Installation
- Logical structure
- Extensions

Ceph is…

- An open-source, scalable, high-performance, distributed (parallel, fault-tolerant) filesystem.
- Core functionality: Object Store
  - Filesystem, block device and S3 like interfaces build on this.
  - Big idea: rados/CRUSH for block placement + location. (See Sage Weil's PhD thesis)

Naming

- Strongly octupus/squid-oriented naming convention (*ceph*alopod)

- Release Versions have names derived from species of cephalopod, alphabetically ordered

    - Argonaut, Bobtail, Cuttlefish, Dumpling, Emperor, **Firefly**, *Giant*

- Commercial support company called *Inktank.*

    - RedHat is now a major partner in this.

CRUSH

- Traditional storage systems store data locations in a table somewhere.
  - flat file, in memory, in MySQL db, etc…
- To write or read a file, you need to read this table.
  - Obvious bottleneck. Single point of failure?

CRUSH

- Rather than storing path as metadata, we could calculate it from a hash for each file.
    - (i.e. Rucio does this for ATLAS, at directory level)
- No lookups needed to get file if we know name…
- But doesn’t help load balancing etc…

CRUSH

- Simple hash-maps cannot cope with a change to storage geometry.

- CRUSH provides improved block placement, with a mechanism for migrating the mappings to a change in geometry.

- Notably, it claims to minimise the number of blocks which need relocated when that happens.

CRUSH Hierarchy

- CRUSH map is a tree, with configurable depth.
- “Buckets” map to particular depths in the tree. (e.g. Root -> Room -> Rack -> Node -> Filesystem)
- Ceph generates a default geometry
    - You can customise this as much as you want.
    - How about adding a “Site” bucket?

Example CRUSH map (1)

```
# begin crush map
tunable choose_local_tries 0
tunable choose_local_fallback_tries 0
tunable choose_total_tries 50
tunable chooseleaf_descend_once 1
```

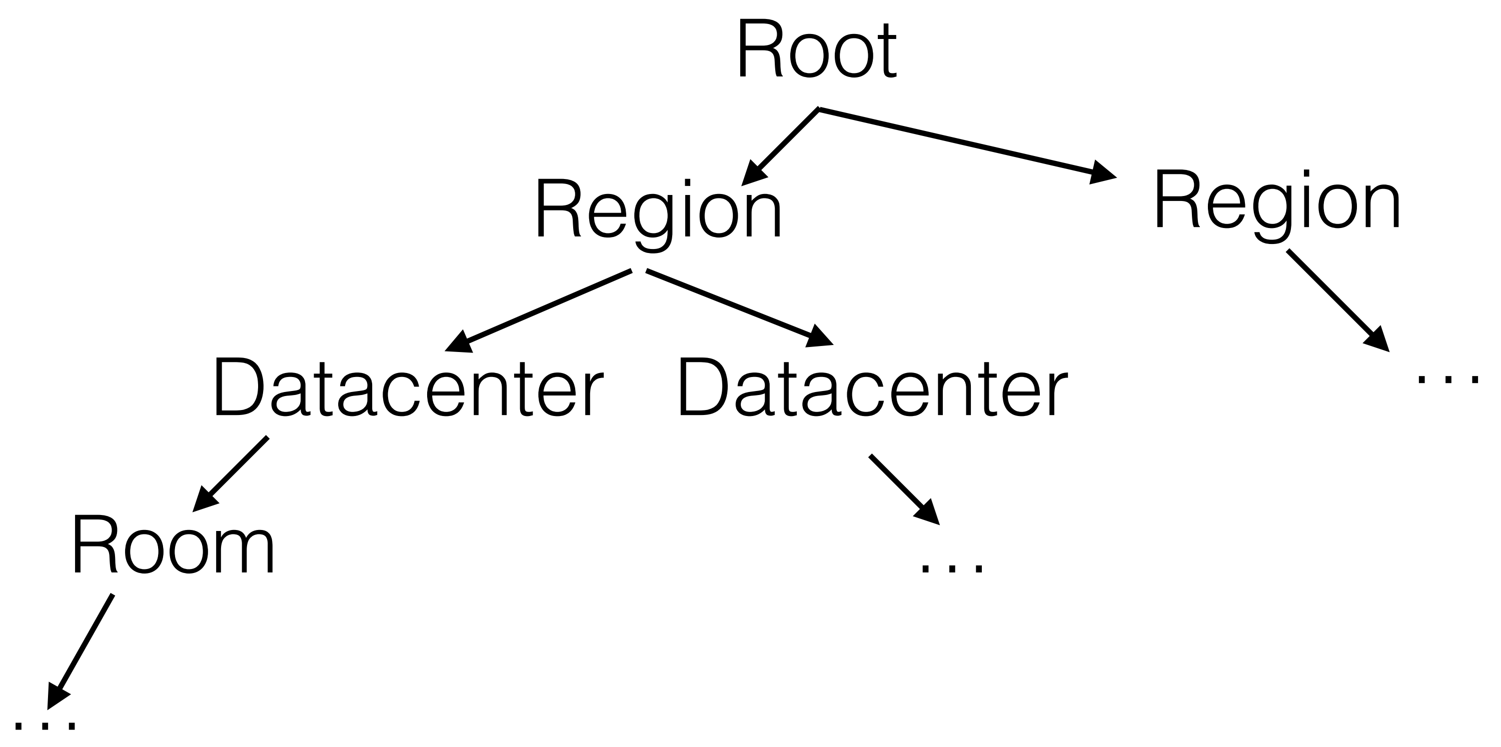
General settings

```
# devices
device 0 osd.0
device 1 osd.1
device 2 osd.2
```

Device -> OSD mappings
(osds are always the leaves of the tree)

```
# types
type 0 osd
type 1 host
type 2 chassis
type 3 rack
type 4 row
type 5 pdu
type 6 pod
type 7 room
type 8 datacenter
type 9 region
type 10 root
```

"Bucket hierarchy"



Root

Region

Region

Datacenter

Datacenter

…

Room

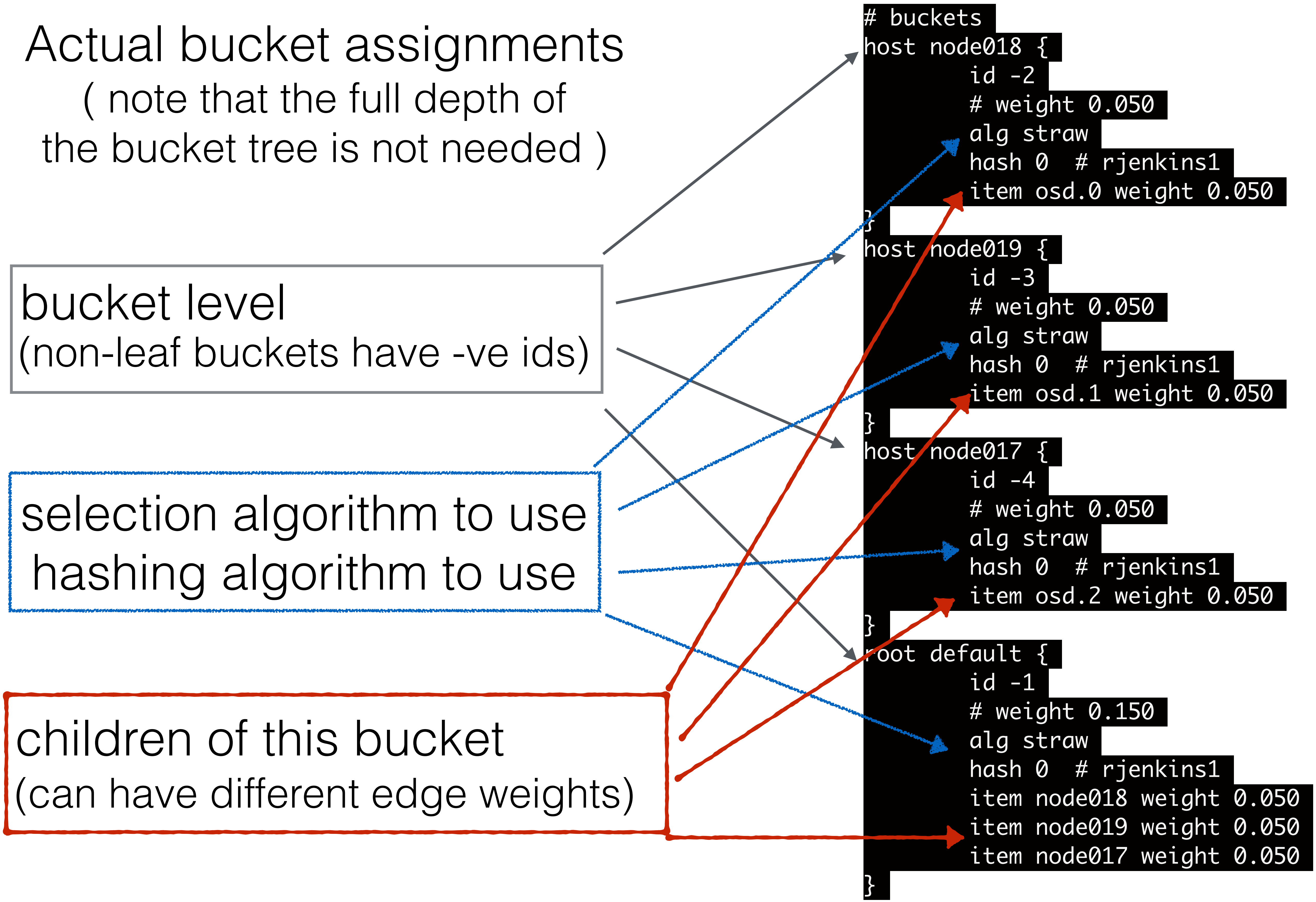
…

…

Example CRUSH map (2)

Actual bucket assignments

(note that the full depth of the bucket tree is not needed)



bucket level
(non-leaf buckets have -ve ids)

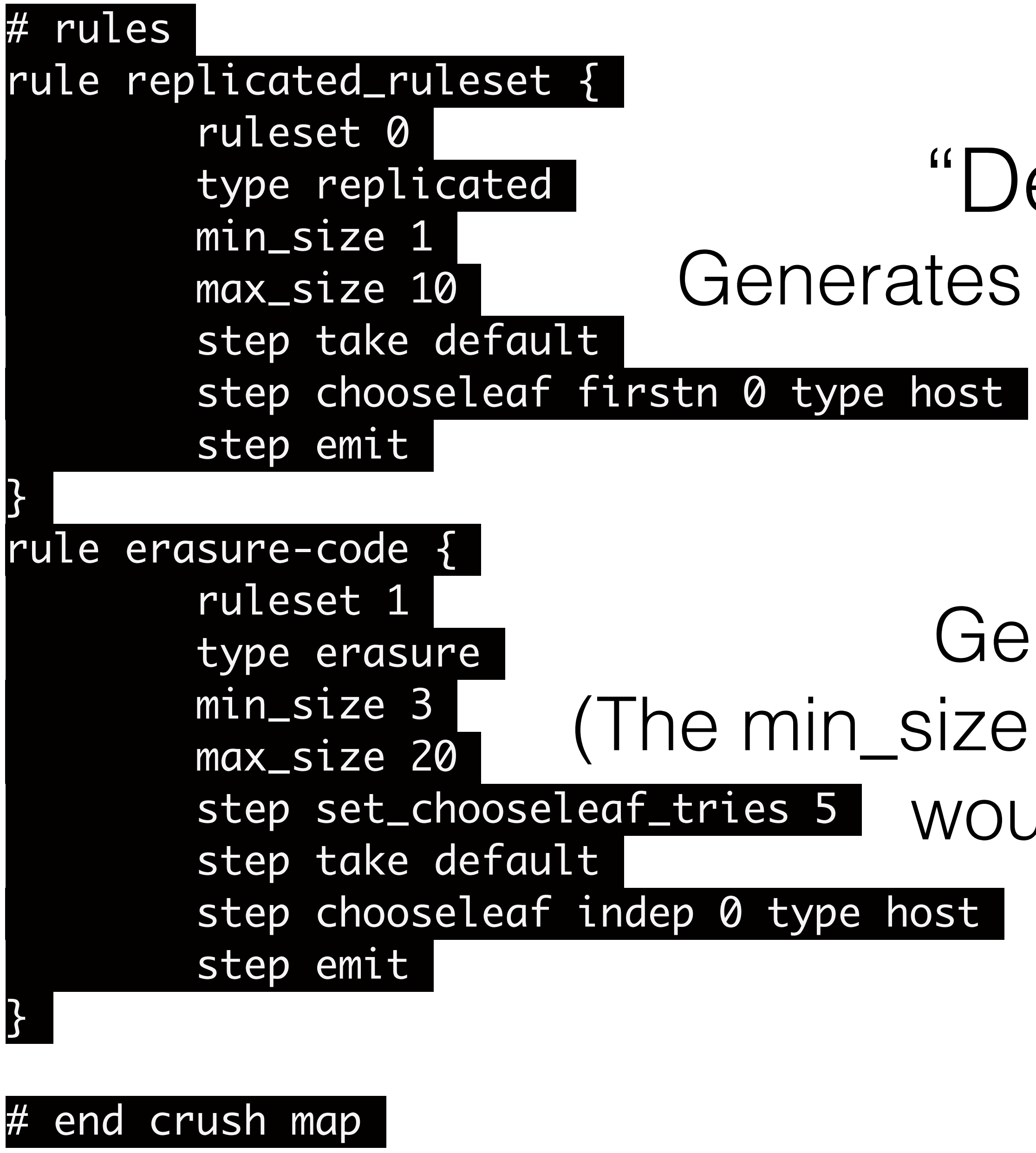
selection algorithm to use
hashing algorithm to use

children of this bucket
(can have different edge weights)

```
# buckets
host node018 {
        id -2
        # weight 0.050
        alg straw
        hash 0  # rjenkins1
        item osd.0 weight 0.050
}
host node019 {
        id -3
        # weight 0.050
        alg straw
        hash 0  # rjenkins1
        item osd.1 weight 0.050
}
host node017 {
        id -4
        # weight 0.050
        alg straw
        hash 0  # rjenkins1
        item osd.2 weight 0.050
}
root default {
        id -1
        # weight 0.150
        alg straw
        hash 0  # rjenkins1
        item node018 weight 0.050
        item node019 weight 0.050
        item node017 weight 0.050
}
```

Example CRUSH map (3)

Rules for different pool types



```
# rules
rule replicated_ruleset {
        ruleset 0
        type replicated
        min_size 1
        max_size 10
        step take default
        step chooseleaf firstn 0 type host
        step emit
}
rule erasure-code {
        ruleset 1
        type erasure
        min_size 3
        max_size 20
        step set_chooseleaf_tries 5
        step take default
        step chooseleaf indep 0 type host
        step emit
}

# end crush map
```

“Default” replication rule.

Generates replicas distributed across OSDs.

Erasure-coding rule.

Generates additional EC chunks.

(The min_size is 3 because even a single chunk object would need additional EC chunks.)

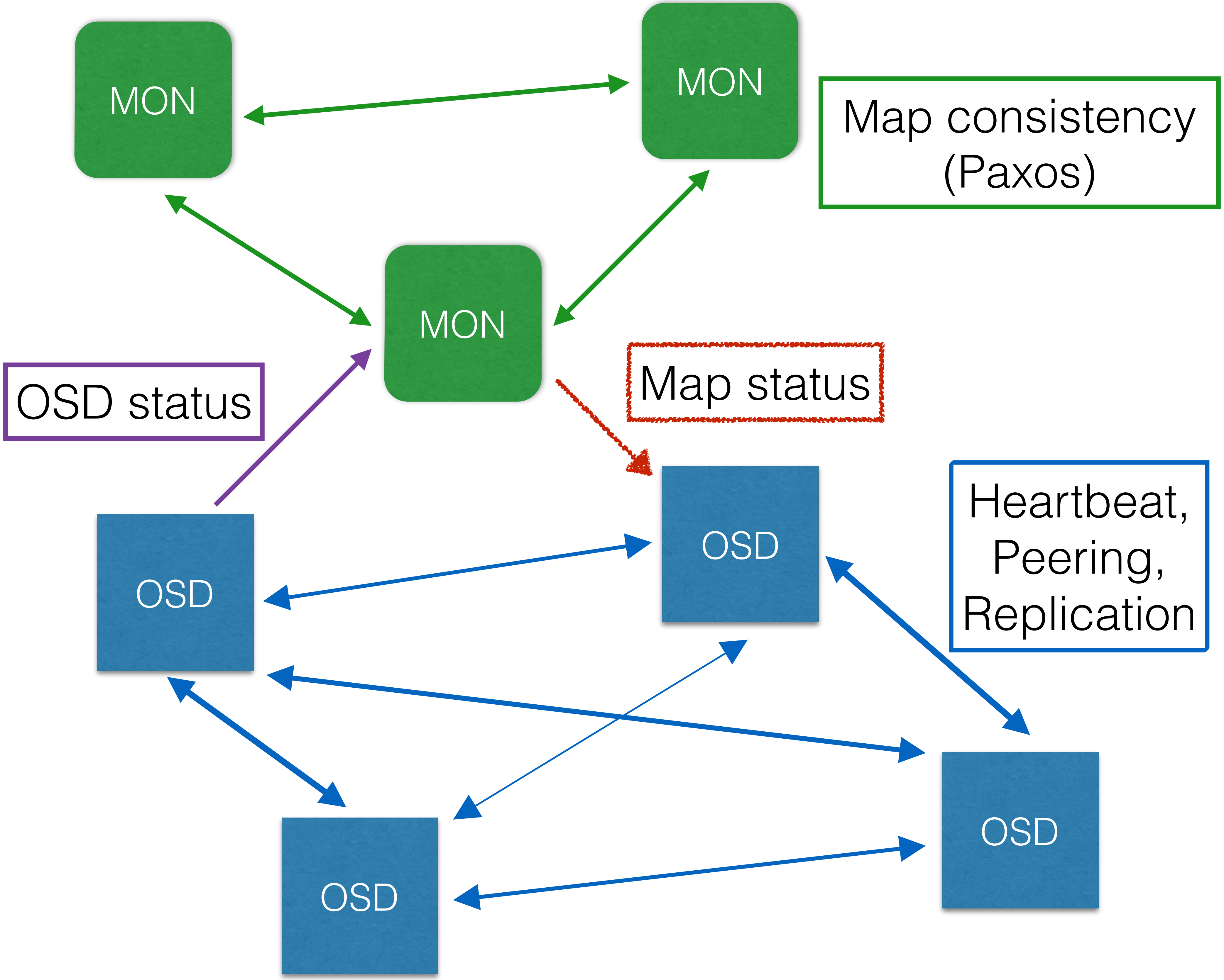
Components

- MON
  - Monitor - knows the Cluster Map (=CRUSH Map + some other details)
  - Can have more than one (they vote on consistency via Paxos for high availability and reliability).
  - Talk to everything to distribute the Storage Geometry, and arrange updates to it.

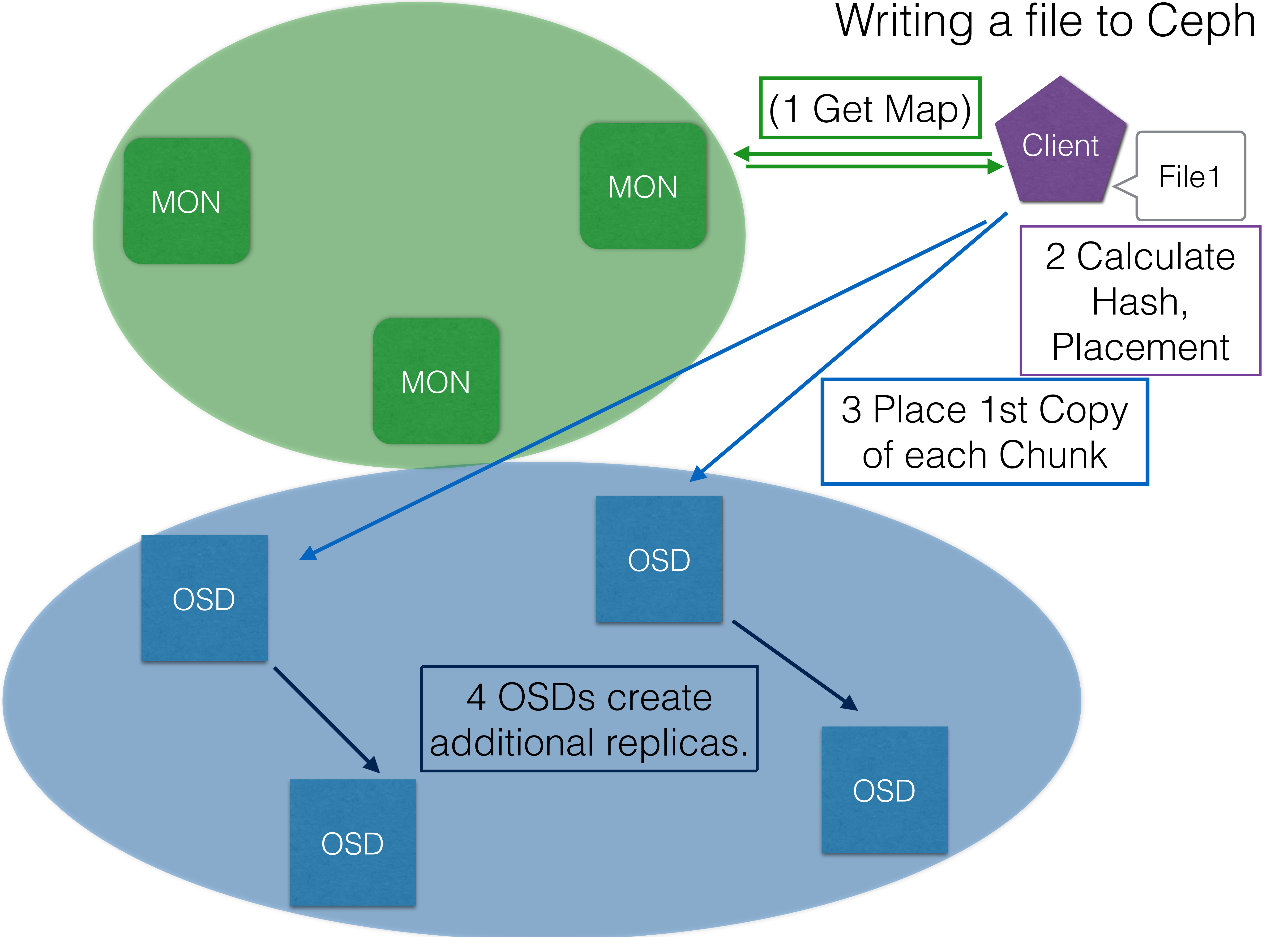
Components

- OSD
  - Object Storage Device - stores blocks of data (and metadata).
  - Need at least three for resilience in default config.
  - Talk to each other to agree on replica status, check health.
  - Talk to MONs to update Storage Geometry

Autonomic functions in Ceph



Writing a file to Ceph



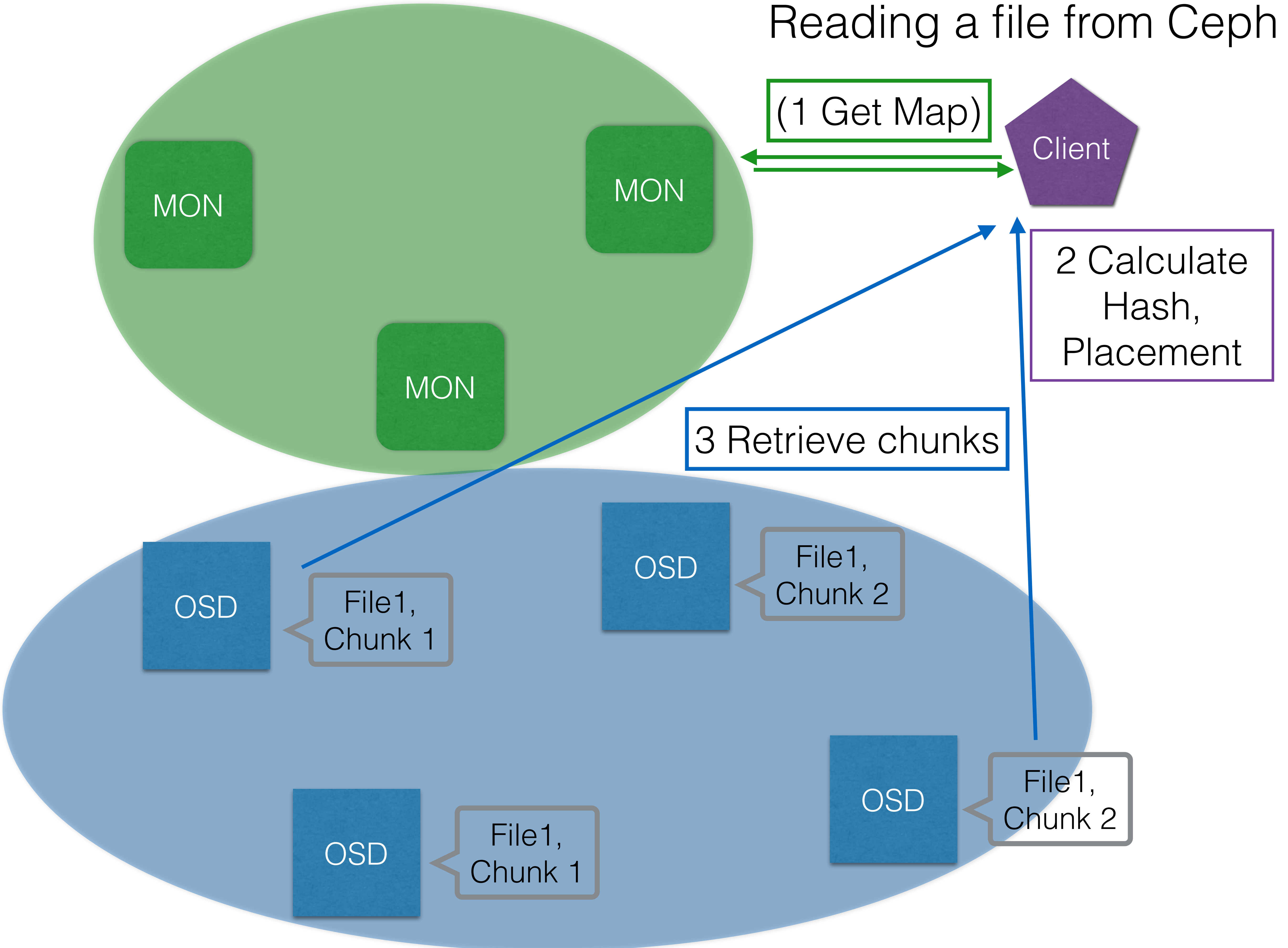
(1 Get Map)

2 Calculate Hash, Placement

3 Place 1st Copy of each Chunk

4 OSDs create additional replicas.

Reading a file from Ceph



(1 Get Map)

Client

2 Calculate Hash, Placement

3 Retrieve chunks

MON

MON

MON

OSD

File1, Chunk 1

OSD

File1, Chunk 2

OSD

File1, Chunk 1

OSD

File1, Chunk 2

Installing

- On RHEL (…SL…Centos…)
  - Add ceph repo to all nodes in storage cluster.
  - Install “admin node” (manages other nodes’ services).
    - `sudo yum update && sudo yum install ceph-deploy`
  - Set up passwordless ssh between admin and other nodes.

Installing (2)

- Create initial list of MONs:
  - `ceph-deploy new node1 node2 (etc)`
- Install/activate node types:
  - `ceph-deploy mon create node1`

    MON

  - `ceph-deploy osd prepare nodex:path/to/fs`
  - `ceph-deploy osd activate nodex`

    OSD

Logical structure

- Partition global storage into “Pools”
  - Can be just a logical division
  - Can also enforce different permissions, replication strategies, etc
- Ceph creates a default pool for you when you install.

Placement Groups

- Pools contain Placement Groups (PGs).
  - Like individual stripe sets for data.
  - A given object is assigned to a PG for distribution.
  - Automatically generated for you!

PG ID = Pool.PG

```
[ceph@node017 my-cluster]$ ceph pg map 0.1
osdmap e68 pg 0.1 (0.1) -> up [1,2,0] acting [1,2,0]
```

vector of OSD ids to stripe over
(first OSD in vector is master)

Examples

Ceph “Status”

```
[ceph@node017 my-cluster]$ ceph -s
    cluster 1738aad3-1413-42b8-9ef8-d3955da0af83
     health HEALTH_OK
     monmap e3: 3 mons at
{node017=10.141.101.17:6789/0,node018=10.141.101.18:6789/0,node019=10.141.101.19:6789/0},
election epoch 22, quorum 0,1,2 node017,node018,node019
     osdmap e68: 3 osds: 3 up, 3 in
      pgmap v64654: 488 pgs, 9 pools, 2048 MB data, 45 objects
            24899 MB used, 115 GB / 147 GB avail
                 488 active+clean
```

MON Status
(note PAXOS election info)

OSD and PG Status

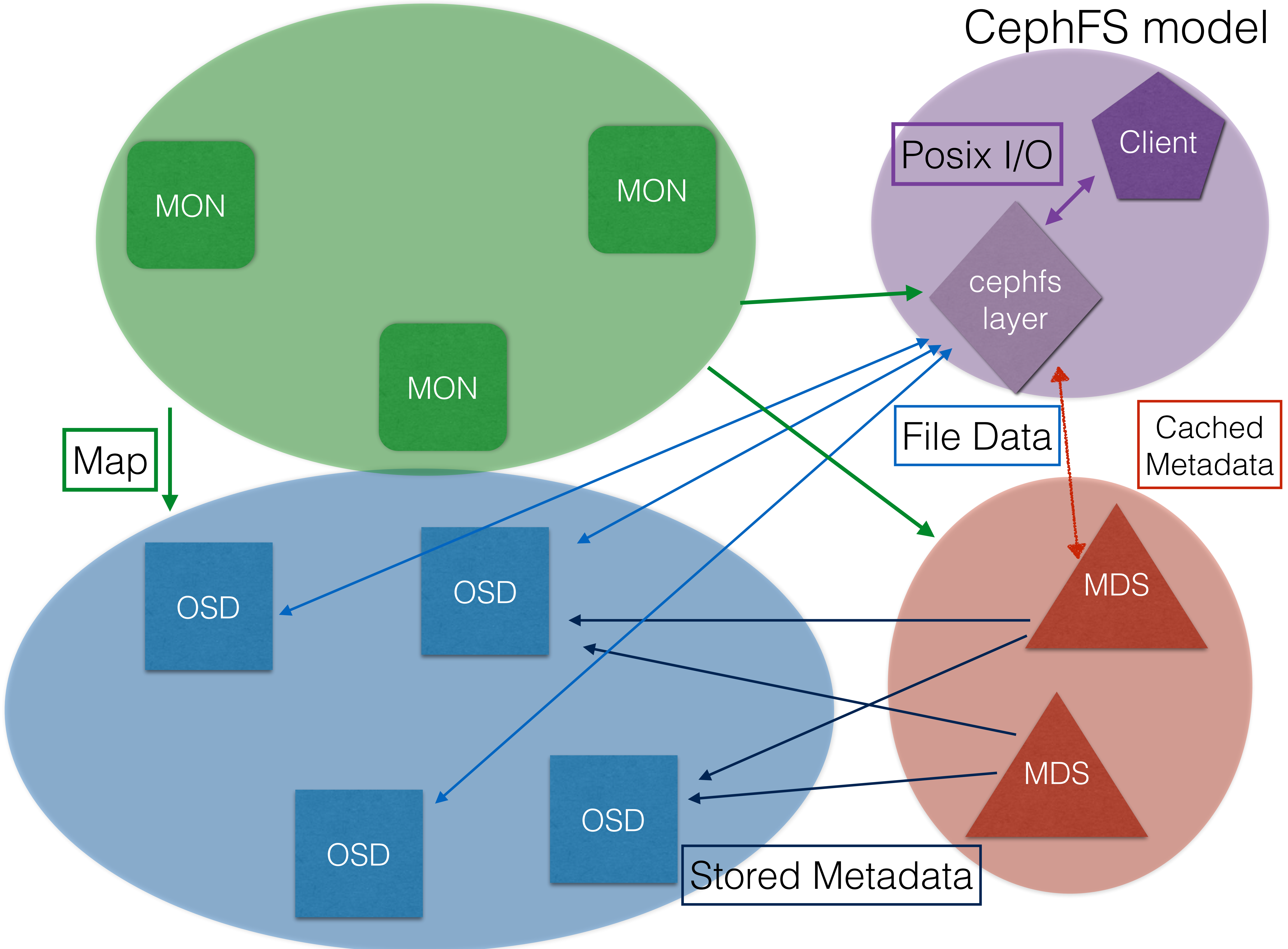
```
[ceph@node017 my-cluster]$ ceph osd lspools
0 data,1 metadata,2 rbd,3 ecpool,4 .rgw.root,5 .rgw.control,6 .rgw,7 .rgw.gc,8 .users.uid,
```

List all pools in this Ceph Cluster
data is the default pool
metadata is also default (used by CephFS extension)
rbd created by Ceph Block Device extension
ecpool is a test erasure-encoded pool
remainder support Ceph Object Gateway (S3, Swift)

Extensions

- POSIX(ish) Filesystem CephFS
  - Need another component - MDS (MetaData Server).
  - MDS handles the metadata heavy aspects of being a POSIX filesystem.
  - Can have more than one (they do failover and load balancing).

CephFS model



Extensions

- Object Gateway (S3, Swift)
  - Need another component - radosgw
  - Provides HTTP(S) interface
  - Maps Ceph Objects to S3/Swift style objects.
  - Supports federated cloud storage.

Extensions

- Block Device
  - Need another component - librbd
  - Presents storage as a Block Device (stored as 4MB chunks on underlying Ceph Object Store)
  - Interacts poorly with erasure-coded pool backends (on writes).

Extensions

- Anything you want!
    - librados has a well documented, public API
    - All extensions are built on it.
    - (I’m currently working on a GFAL2 plugin for it, for example.)

Further Reading

- Sage Weil’s PhD Thesis: http://ceph.com/papers/weil-thesis.pdf (2007)
- Ceph support docs: http://ceph.com/docs/master/