

CEPH

Distributed Storage System

The Ceph Distributed Storage System

- Open-source
- Designed for
 - Commodity hardware
 - Resilience and data safety
 - Scalability
- Popular in Cloud



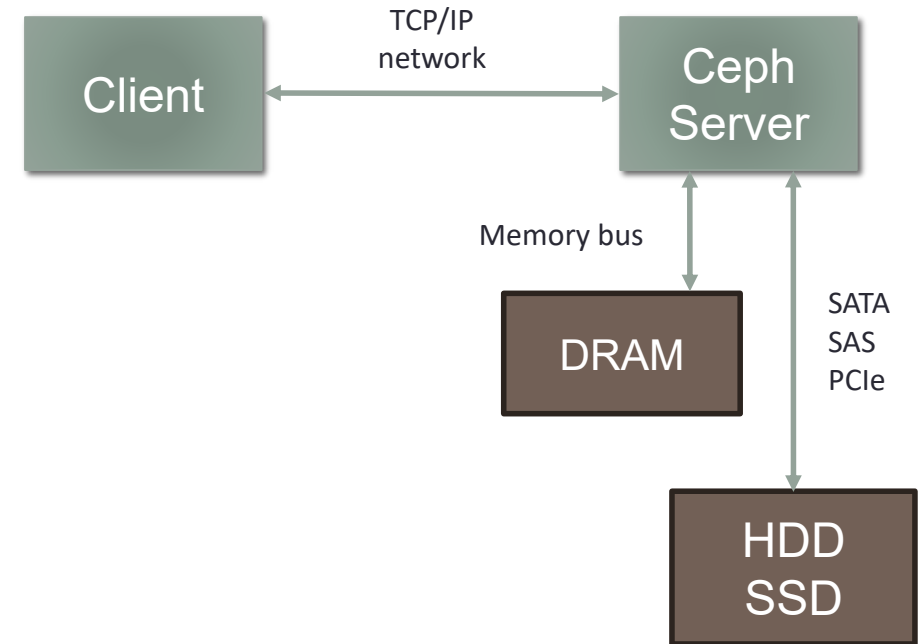
<https://ceph.io/en/discover/>

<https://docs.ceph.com/en/latest/>

<https://www.youtube.com/watch?v=PmLPbrf-x9g>

Hardware Support

- Ceph is software-defined
 - No specific hardware required
- Supports commodity and production hardware
 - HDDs
 - SSDs (SATA/SAS/NVMe)
 - TCP/IP networks
- No DPDK or RDMA support out-of-the-box yet



<https://docs.ceph.com/en/latest/start/hardware-recommendations/#data-storage>

<https://docs.ceph.com/en/latest/rados/configuration/network-config-ref/#general-settings>

Ceph Architecture

- Ceph Storage Cluster daemons (a.k.a. RADOS)
 - Object Storage Daemon (OSD)
 - Monitor Daemon
 - Manager Daemon
- Other daemons
 - Rados Block Device (RBD)
 - Rados GateWay (RGW) → S3
 - MetaData Server (MDS) → POSIX
- All daemons can be deployed and scaled independently



ceph-mon



ceph-mgr



ceph-osd

Object Storage Daemon (OSD) – ceph-osd

- Manages local storage in a node
 - Potentially multiple OSDs per node
- Exploits raw devices
- Stores object data
- Stores metadata index
 - Can exploit fast storage layer if present
- Clients perform I/O directly to OSDs
- At least 3 OSDs per system for data safety
- Require 2-4 GiB DRAM each
- Scales up to 10.000s of OSDs



ceph-osd

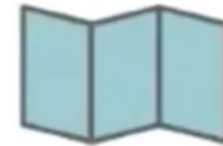


Monitor Daemon – ceph-mon

- Keeps up-to-date map of the cluster
 - OSDs up/down
 - Distribution in nodes, racks, ...
- Acts as cluster manager
- Is the authentication authority
- Map consensus with other monitors via Paxos
- 3 to 7 Monitors per system
- Require 32-128 GiB DRAM each



ceph-mon



Manager Daemon – ceph-mgr

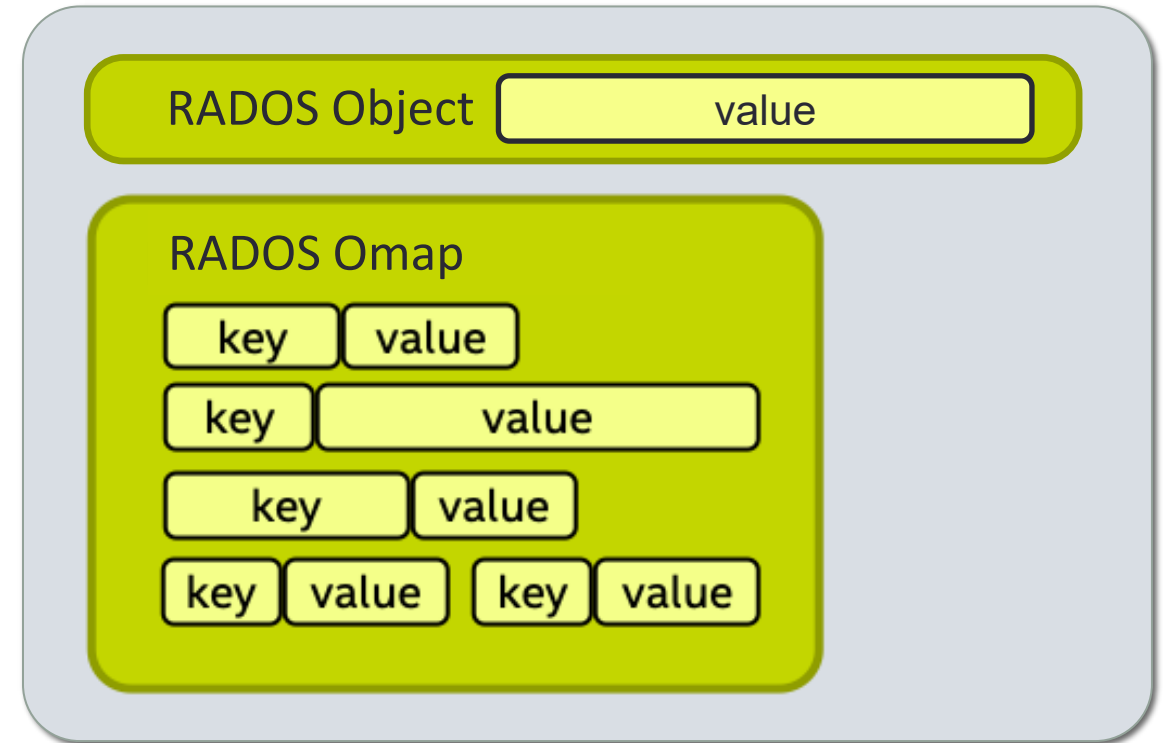
- Aggregates system metrics
- Exposes system metrics
- 2 Managers per system
- Require 32 GiB DRAM each



ceph-mgr

RADOS object storage API

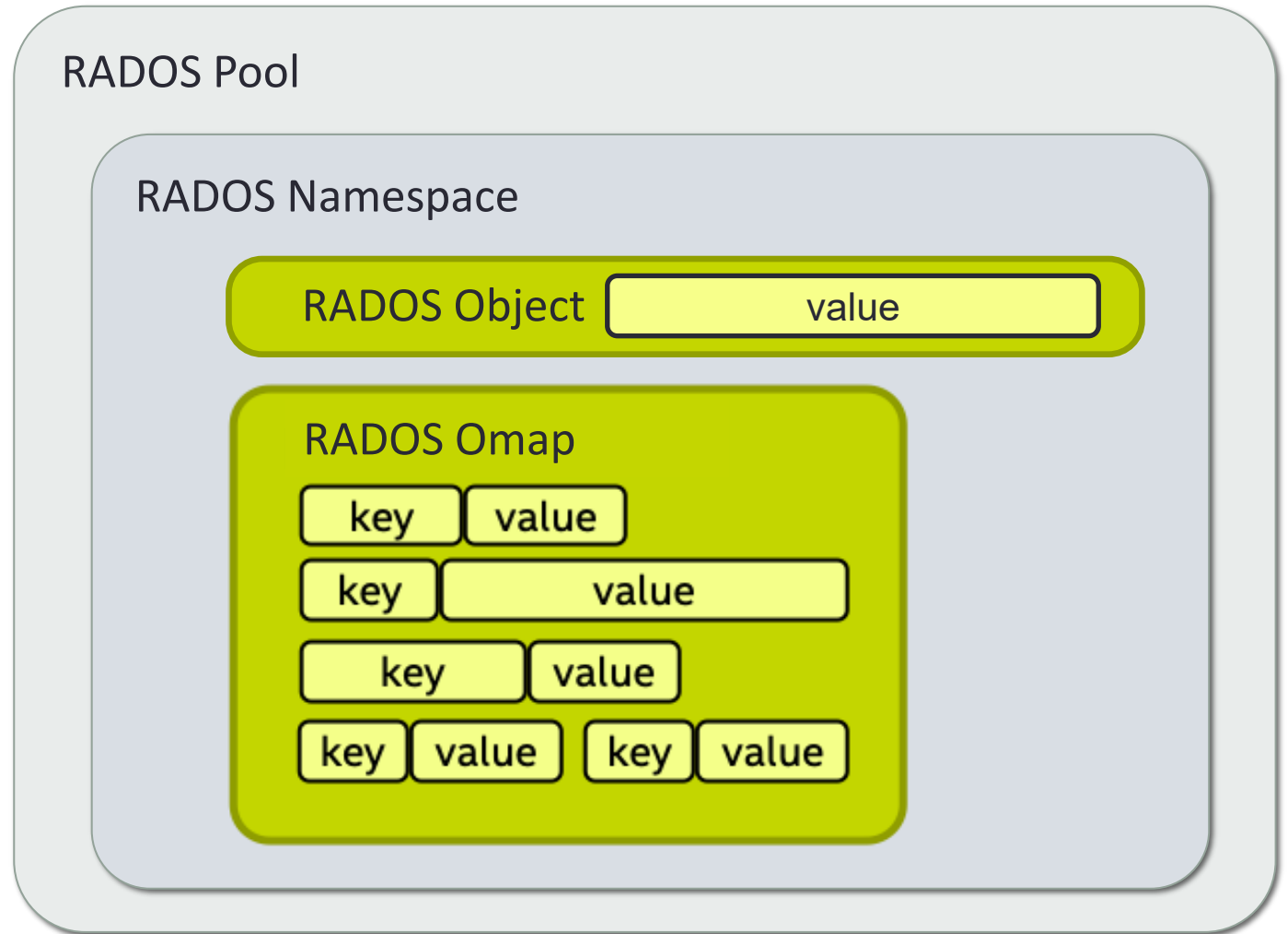
- Clients can interact with the API via librados
- Object
 - Identified by name
 - Can have attributes
 - Regular object
 - stores string of bytes
 - Omap object
 - provides key-value dictionary



<https://docs.ceph.com/en/reef/rados/api/librados/>

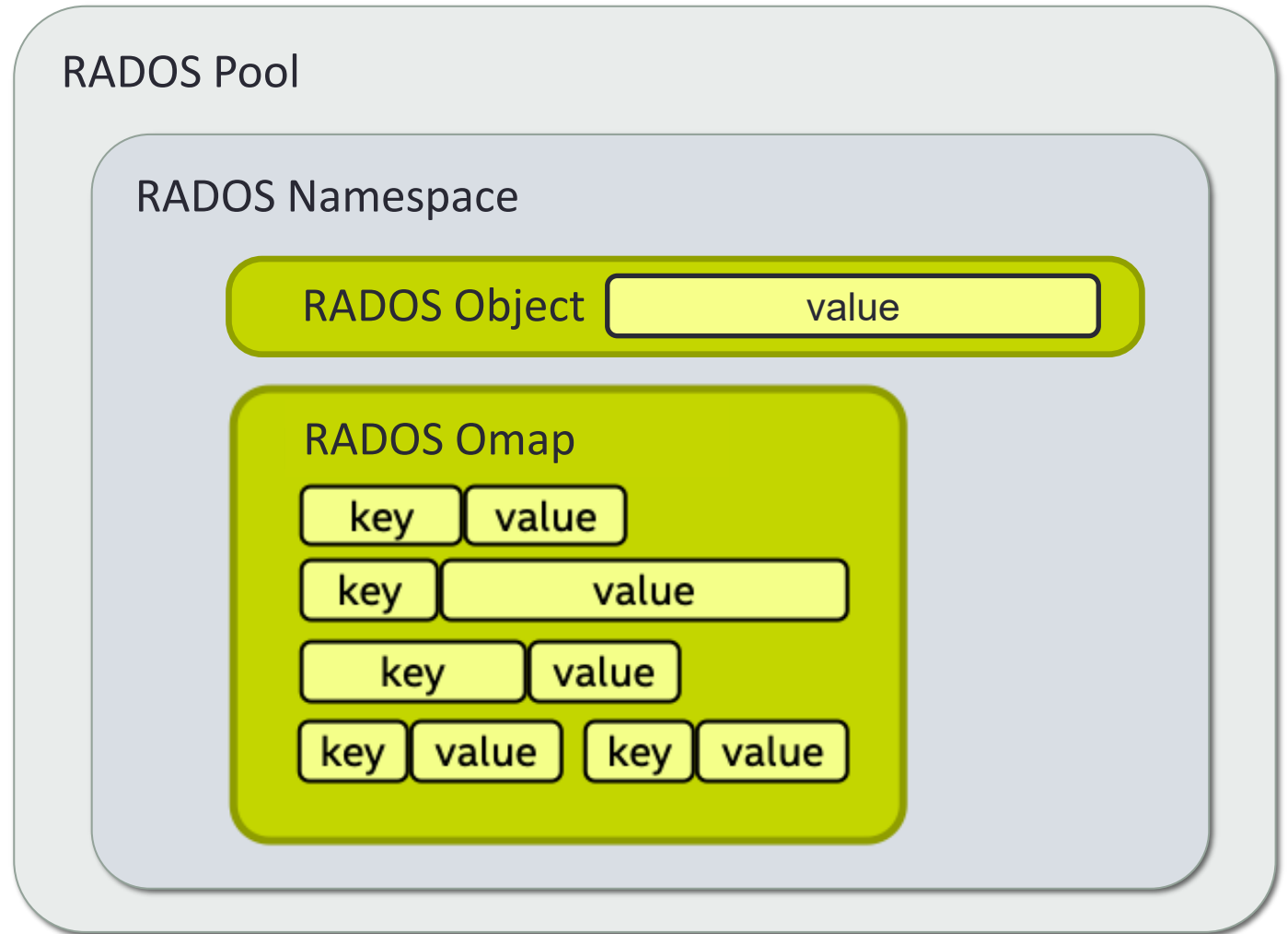
RADOS object storage API

- Pool
 - Partitions object namespace
 - Usually, one pool is created for each type of application
 - Can be bound to specific OSDs
 - Can be configured for
 - Replication
 - Erasure-Coding



RADOS object storage API

- Namespace
 - Partitions object namespace within a pool
 - Lightweight w.r.t. pool

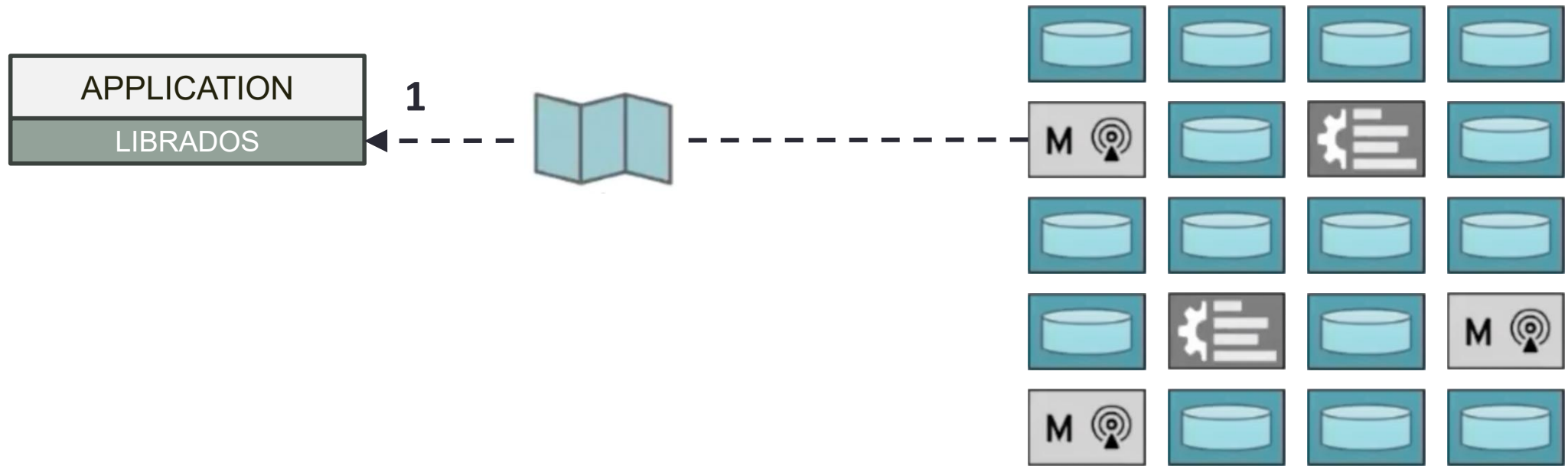


Algorithmic placement



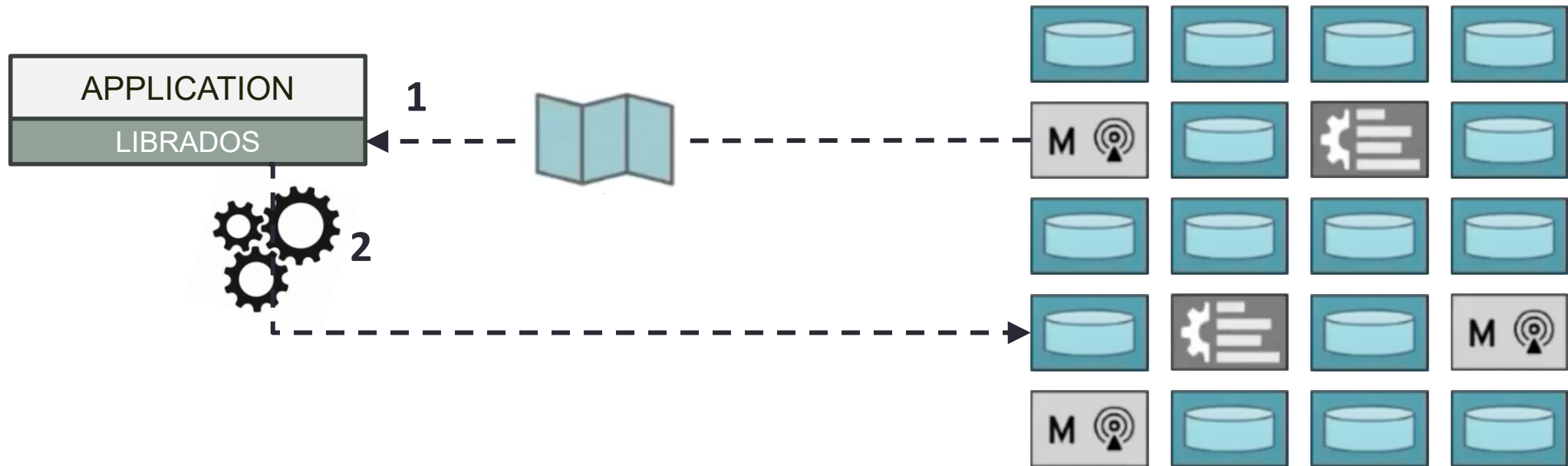
Algorithmic placement

- 1: retrieve up-to-date cluster map



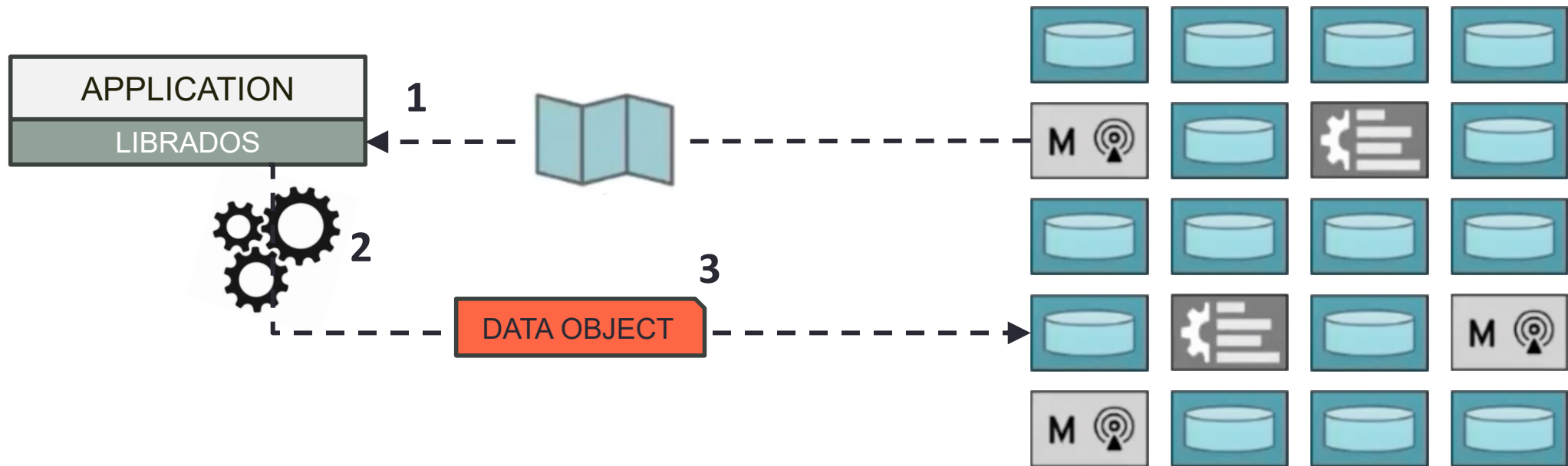
Algorithmic placement

- 1: retrieve up-to-date cluster map
- 2: calculate placement based on object name and map



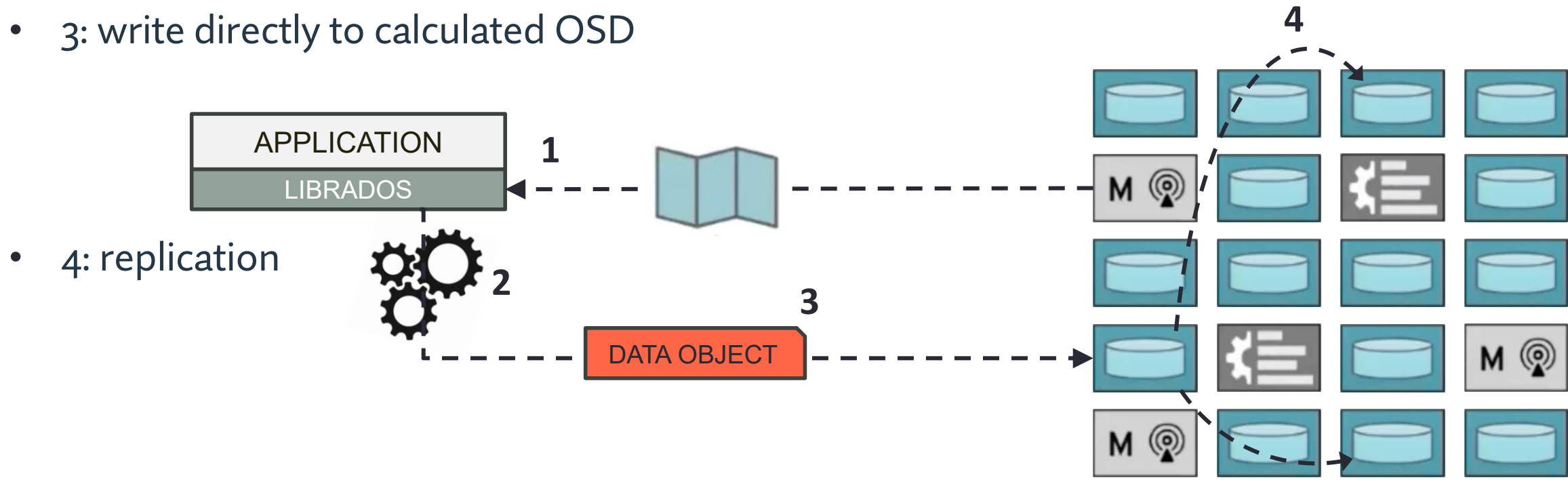
Algorithmic placement

- 1: retrieve up-to-date cluster map
- 2: calculate placement based on object name and map
- 3: write directly to calculated OSD



Algorithmic placement

- 1: retrieve up-to-date cluster map
- 2: calculate placement based on object name and map
- 3: write directly to calculated OSD



- 4: replication

Algorithmic placement

- 1: retrieve up-to-date cluster map
- 2: calculate placement based on object name and map
- 3: write directly to calculated OSD

- 4: replication
- 5: return

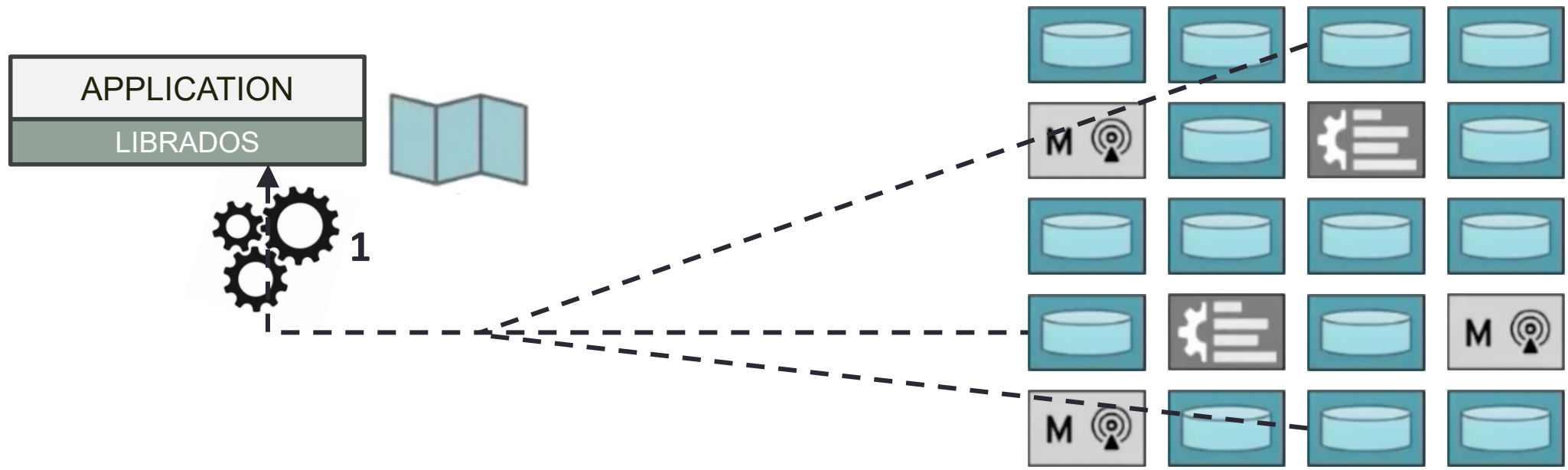


Algorithmic placement – read



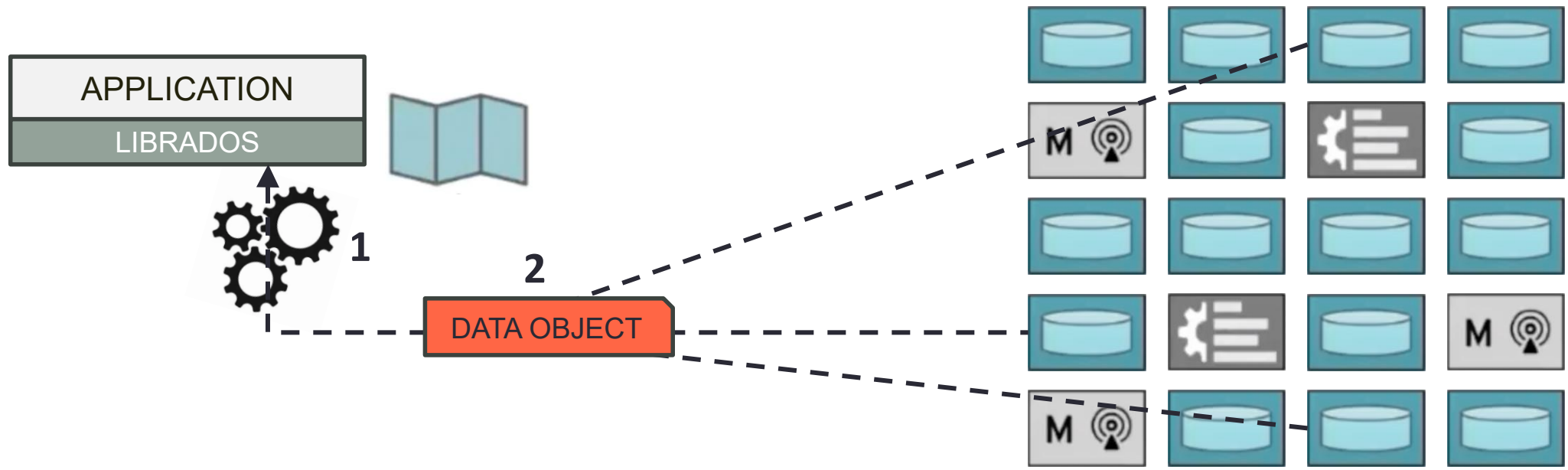
Algorithmic placement – read

- 1: calculate placement based on object name and map



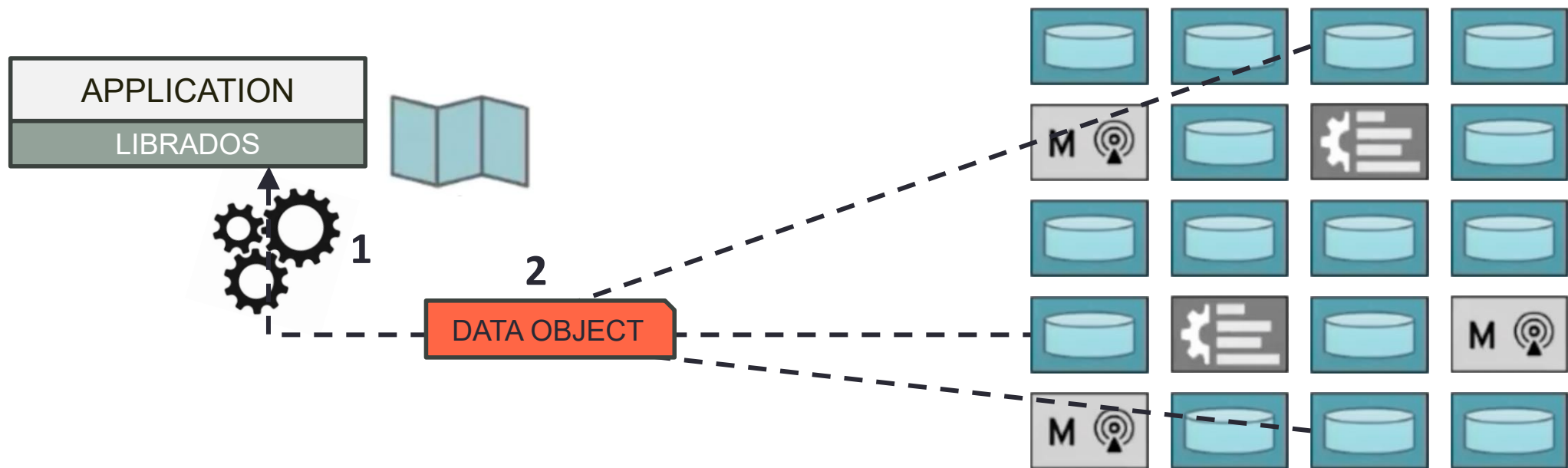
Algorithmic placement – read

- 1: calculate placement based on object name and map
- 2: read object directly from OSDs



Algorithmic placement

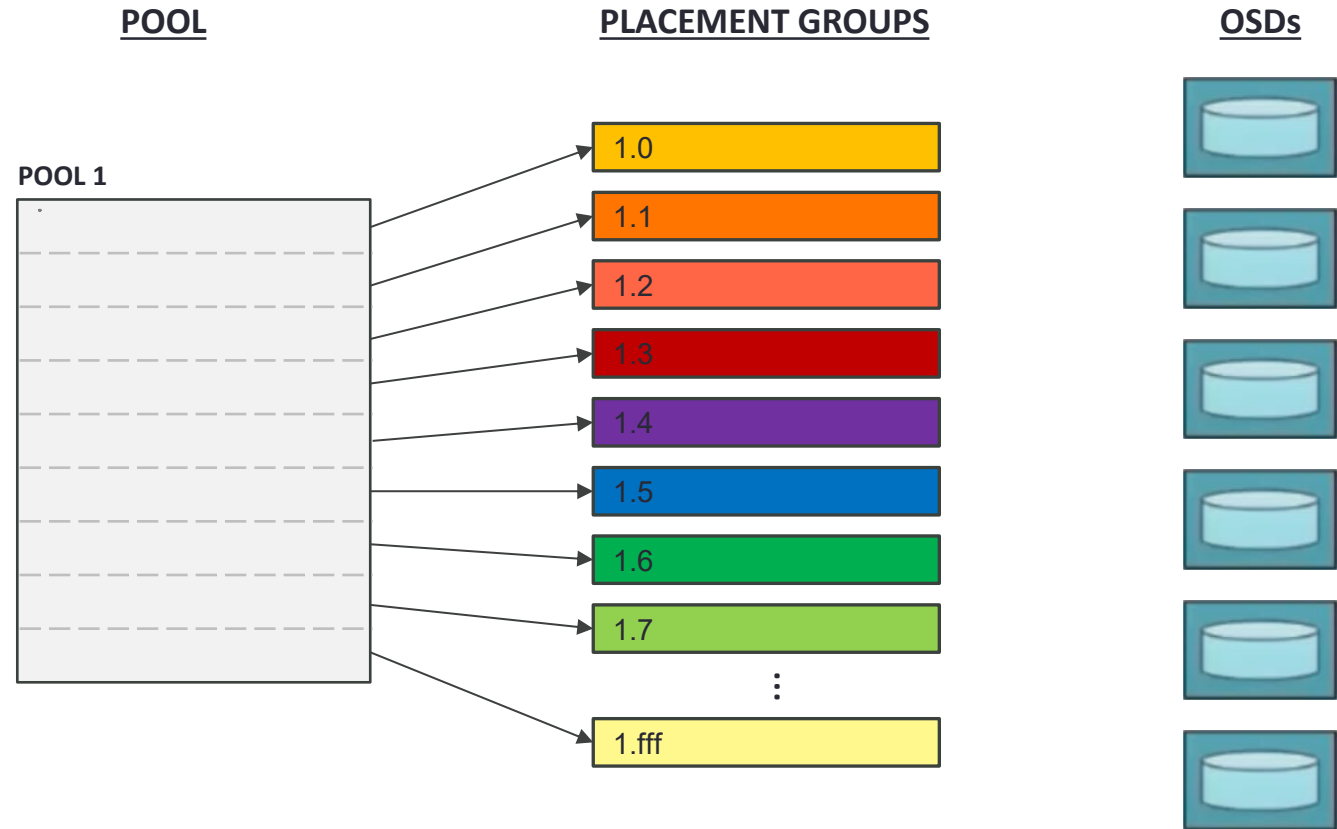
- Algorithmic placement enables scalability
- RADOS' placement algorithm is deterministic and repeatable – CRUSH
- Failure resilient



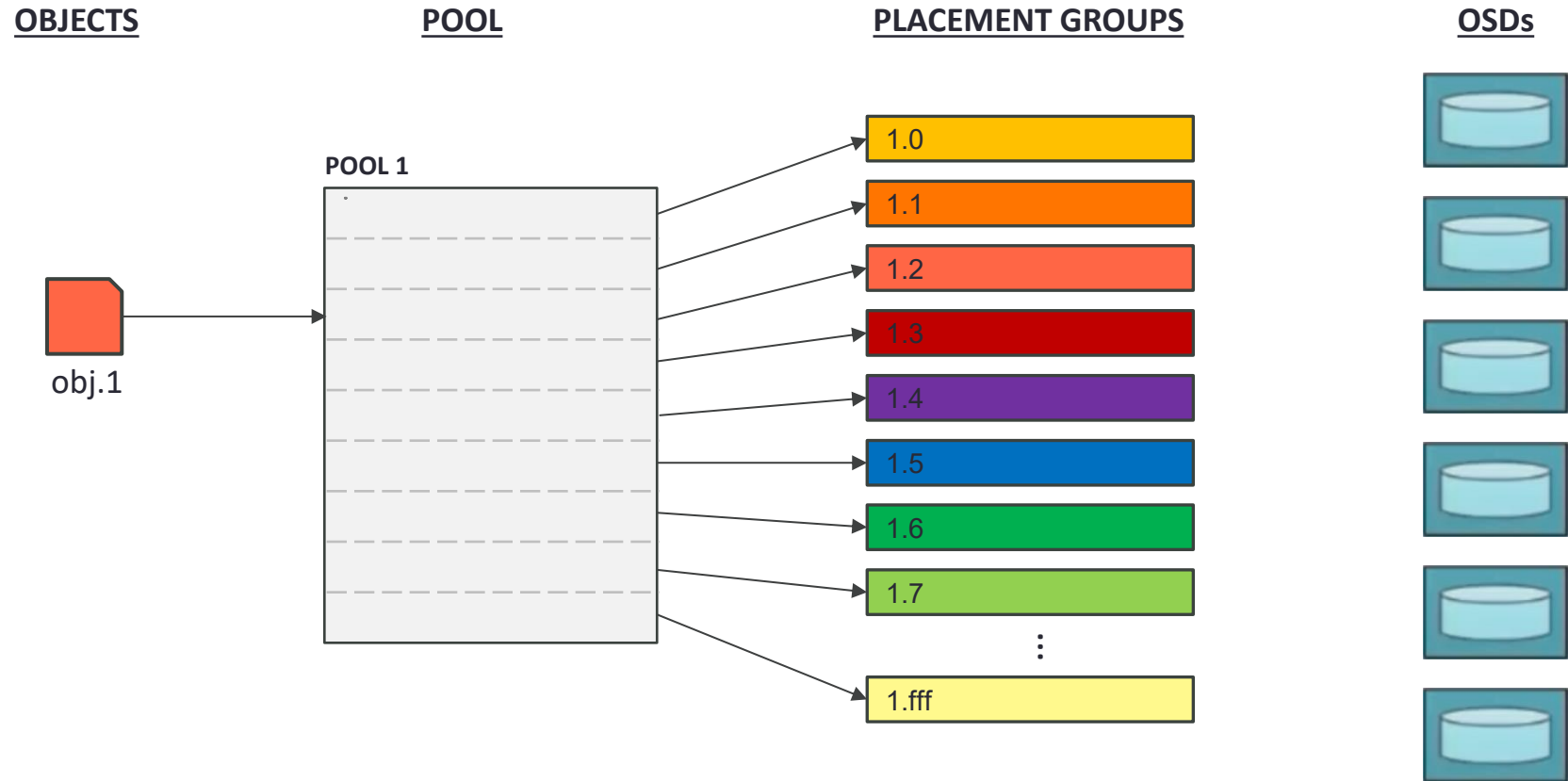
Placement Groups



Placement Groups

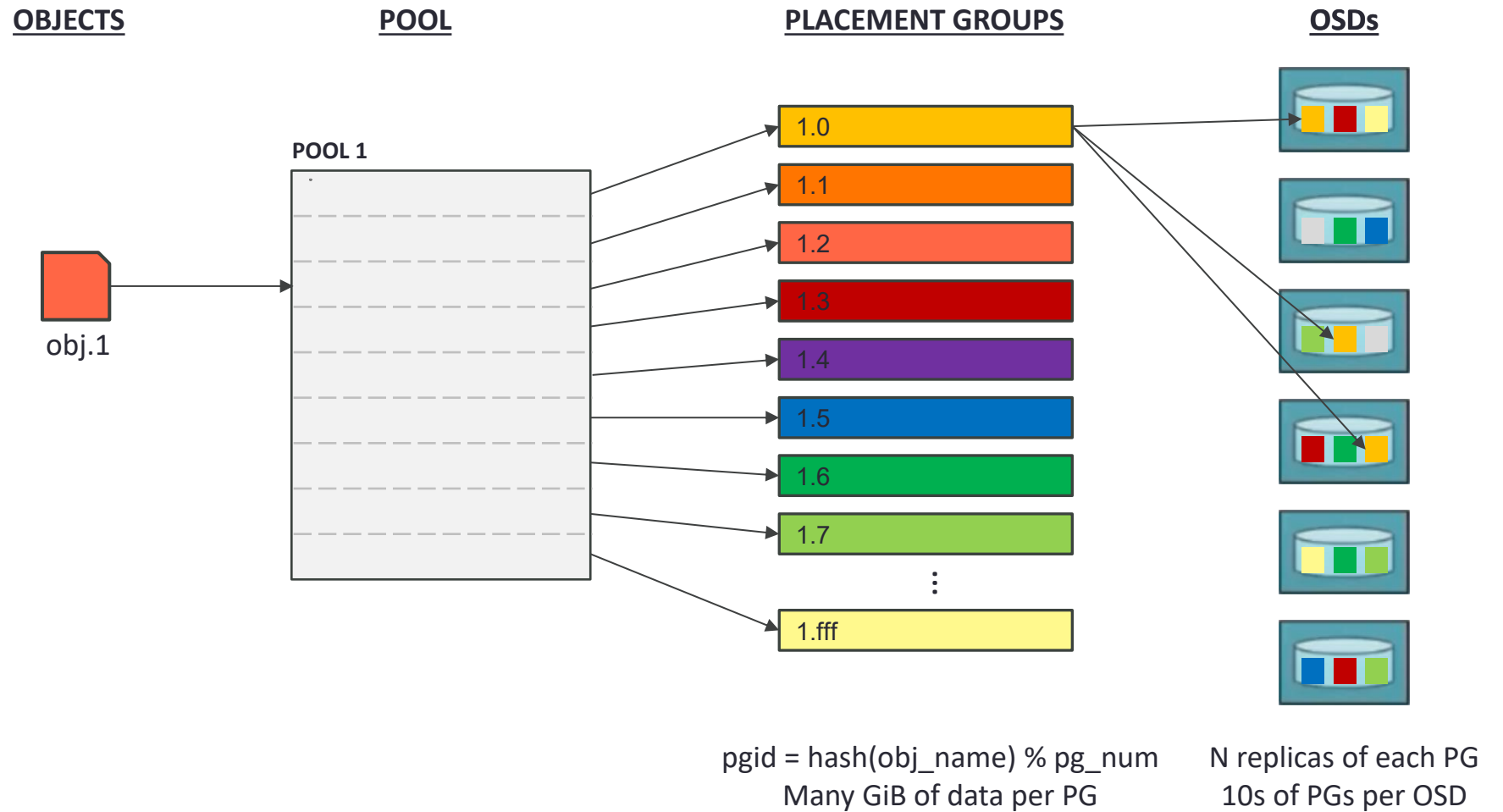


Placement Groups



$\text{pgid} = \text{hash}(\text{obj_name}) \% \text{pg_num}$
Many GiB of data per PG

Placement Groups

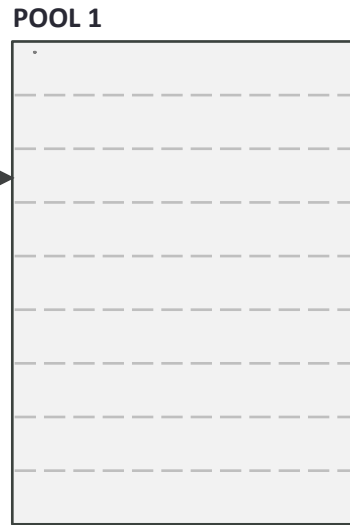


Placement Groups

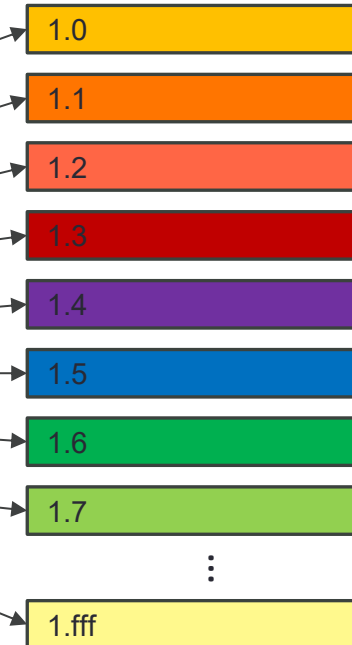
OBJECTS



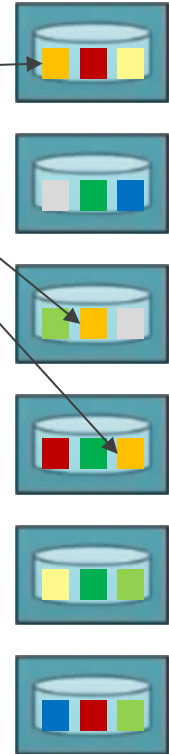
POOL



PLACEMENT GROUPS



OSDs



- PGs are auto-adjusted by default
- As a rule of thumb, a RADOS system should have

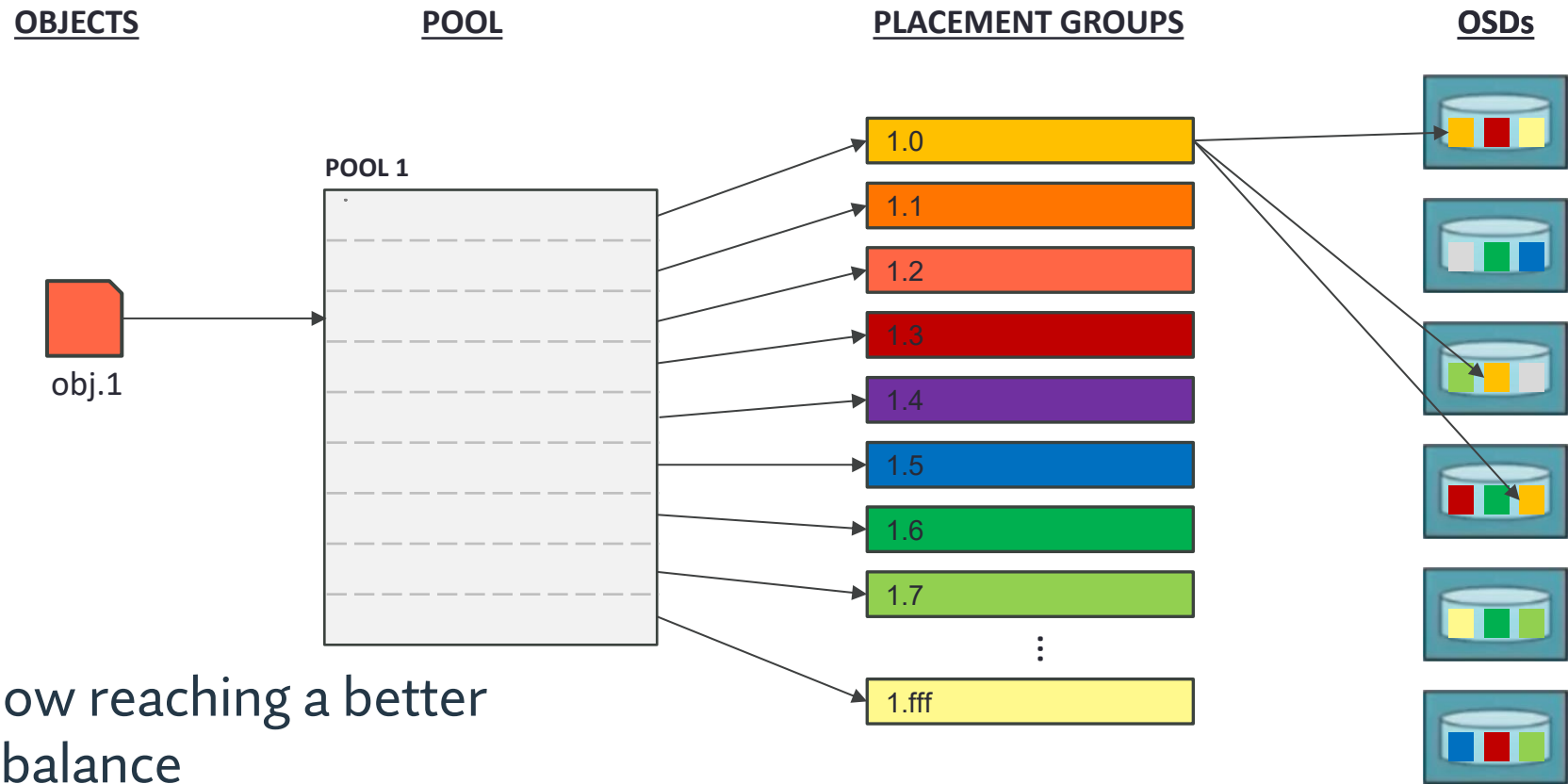
$$\#PG = 100 * \#OSD / \#replicas$$

<https://docs.ceph.com/en/latest/rados/operations/pgcalc/>

$pgid = \text{hash}(\text{obj_name}) \% pg_num$
Many GiB of data per PG

N replicas of each PG
10s of PGs per OSD

Placement Groups



- Placement Groups allow reaching a better performance / safety balance

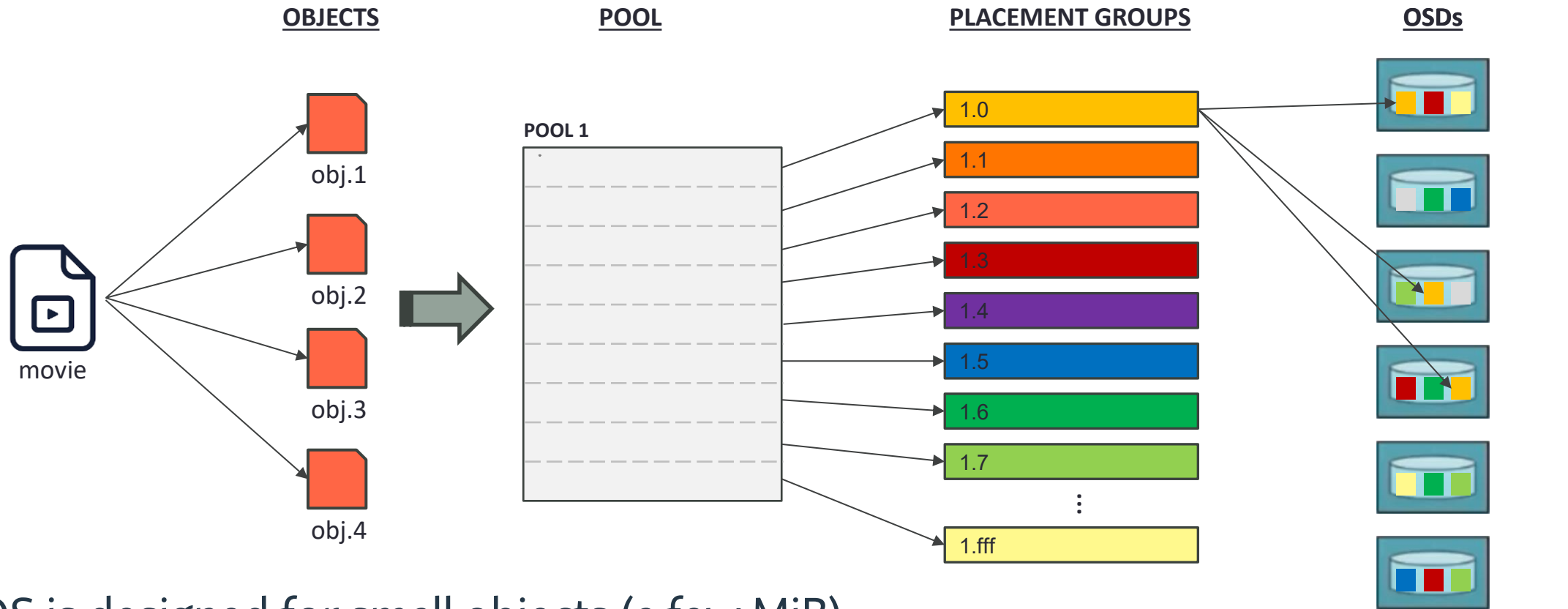
<https://www.youtube.com/watch?v=PmLPbrf-x9g>

<https://ceph.io/assets/pdfs/weil-rados-pdsw07.pdf> - section 2

$\text{pgid} = \text{hash}(\text{obj_name}) \% \text{pg_num}$
Many GiB of data per PG

N replicas of each PG
10s of PGs per OSD

Placement Groups



- RADOS is designed for small objects (a few MiB)

$\text{pgid} = \text{hash}(\text{obj_name}) \% \text{pg_num}$
Many GiB of data per PG

N replicas of each PG
10s of PGs per OSD

RADOS consistency and persistency

- Strong consistency guarantees
 - Write-read, read-write, write-write
 - No client-side caching
- Algorithm similar to MVCC
 - Placement calculation
 - Write to primary OSD
 - Replicate to peer OSDs in PG
 - Index object location and return
 - Reader always checks latest index entry on primary OSD

RADOS consistency and persistency

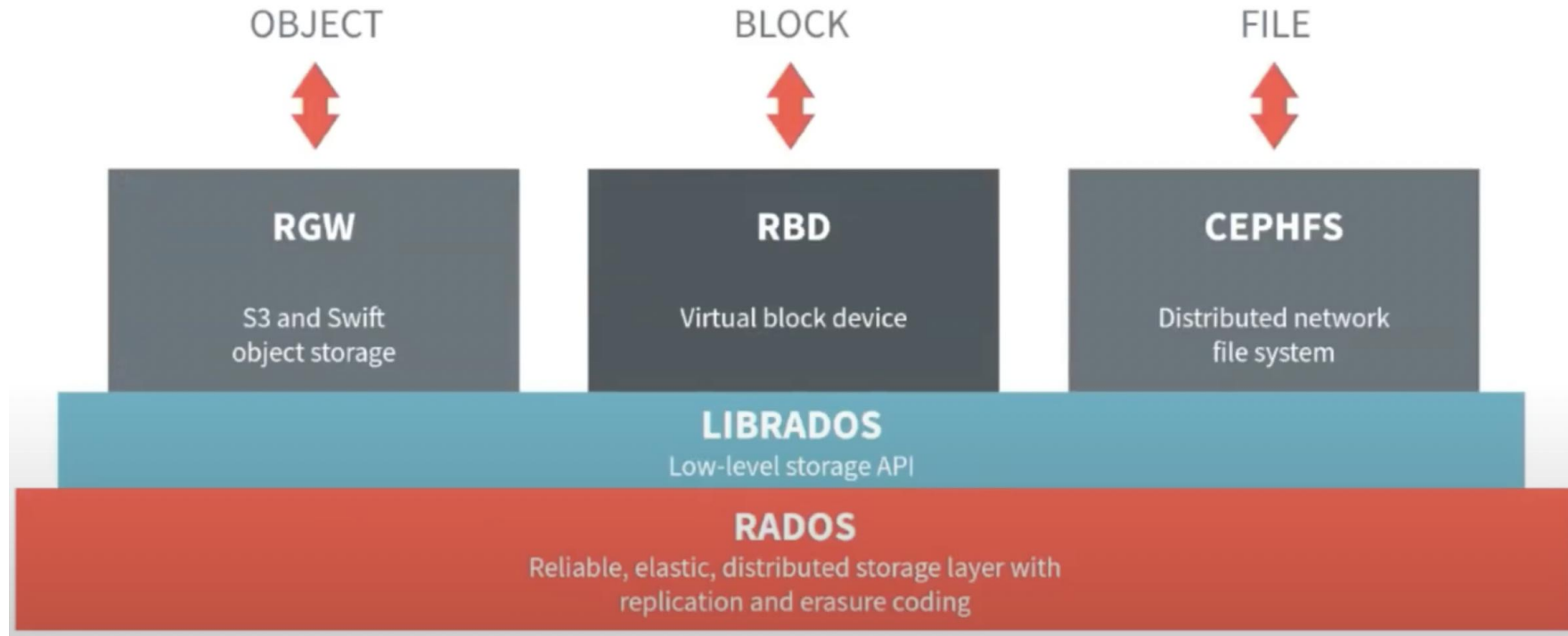
- Immediate persistency
 - Two-step persisting procedure
- Inefficiencies
 - Frequent RPCs to primary OSD
 - Copy on write for partial writes
 - Full transfer to client for partial reads
 - Two-step persistency increases latency

<https://ceph.io/assets/pdfs/weil-rados-pdsw07.pdf> - section 3

<https://docs.ceph.com/en/latest/architecture/#interrupted-full-writes>

https://docs.ceph.com/en/latest/dev/osd_internals/erasure_coding/enhancements/#partial-overwrites

Ceph interfaces on RADOS



- RBD and RGW have earned Ceph popularity in Cloud environments
- RGW and CephFS can eliminate the small object constrain in RADOS

<https://www.youtube.com/watch?v=PmLPbrf-x9g>

RADOS performance

- Designed to scale
- Designed for safety
 - Overhead for RPCs to primary OSDs
 - Overhead for two-phase persistence
 - Partial write/read inefficiencies
- Performance analysis papers

https://sdm.lbl.gov/pdc/pubs/201811_PDSW2018-ObjEval.pdf

<https://msstconference.org/MSST-history/2017/Papers/CephObjectStore.pdf>

<https://www.croit.io/blog/ceph-performance-benchmark-and-optimization>

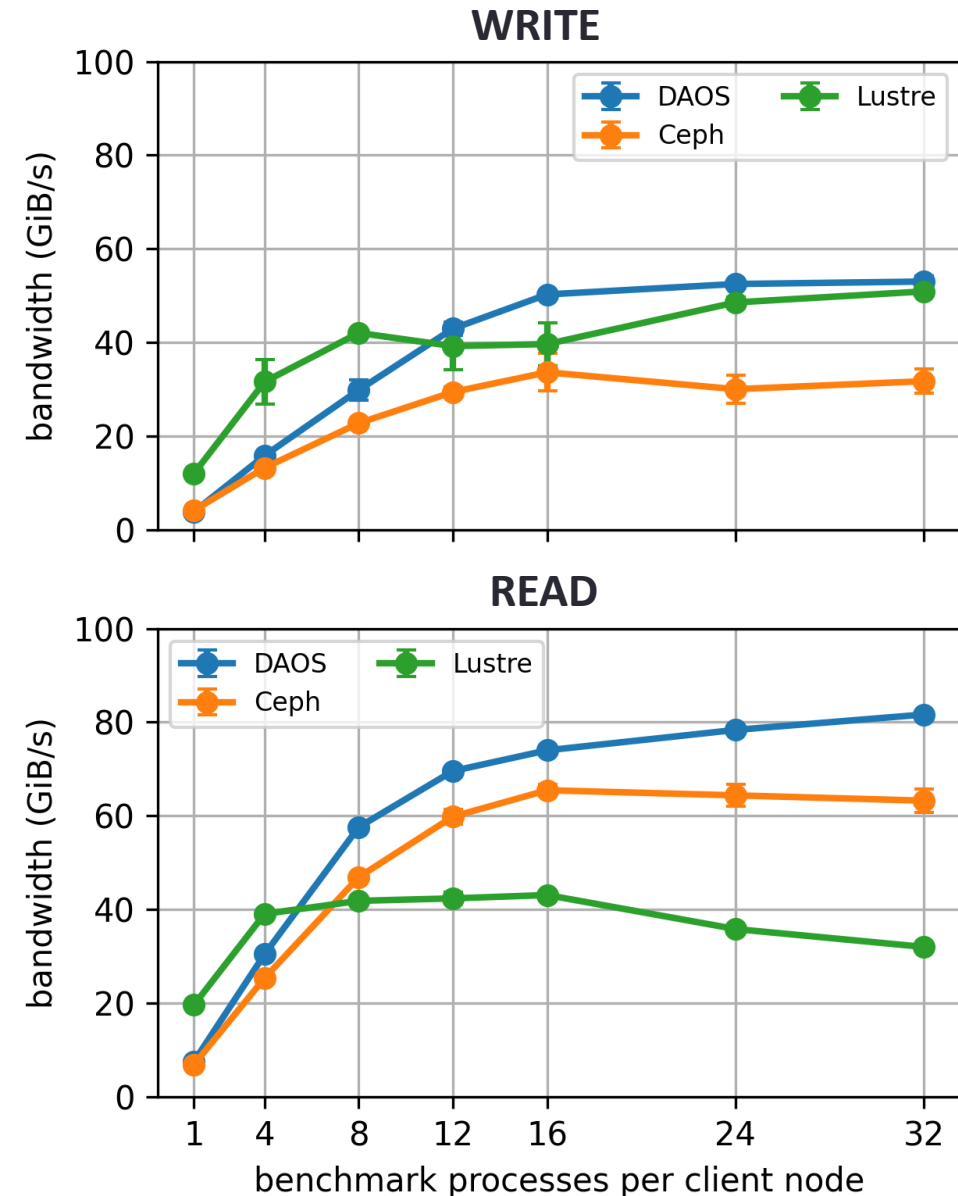
<https://ceph.io/en/news/blog/2024/ceph-a-journey-to-1tibps/>

<https://arxiv.org/pdf/2409.18682>

RADOS performance on NVMe

- RADOS/DAOS/Lustre deployments on 16x 6TiB nodes
- I/O benchmark (fdb-hammer) runs on 32 client nodes
- 10000 I/O operations of 1 MiB per process
- No replication or erasure-coding

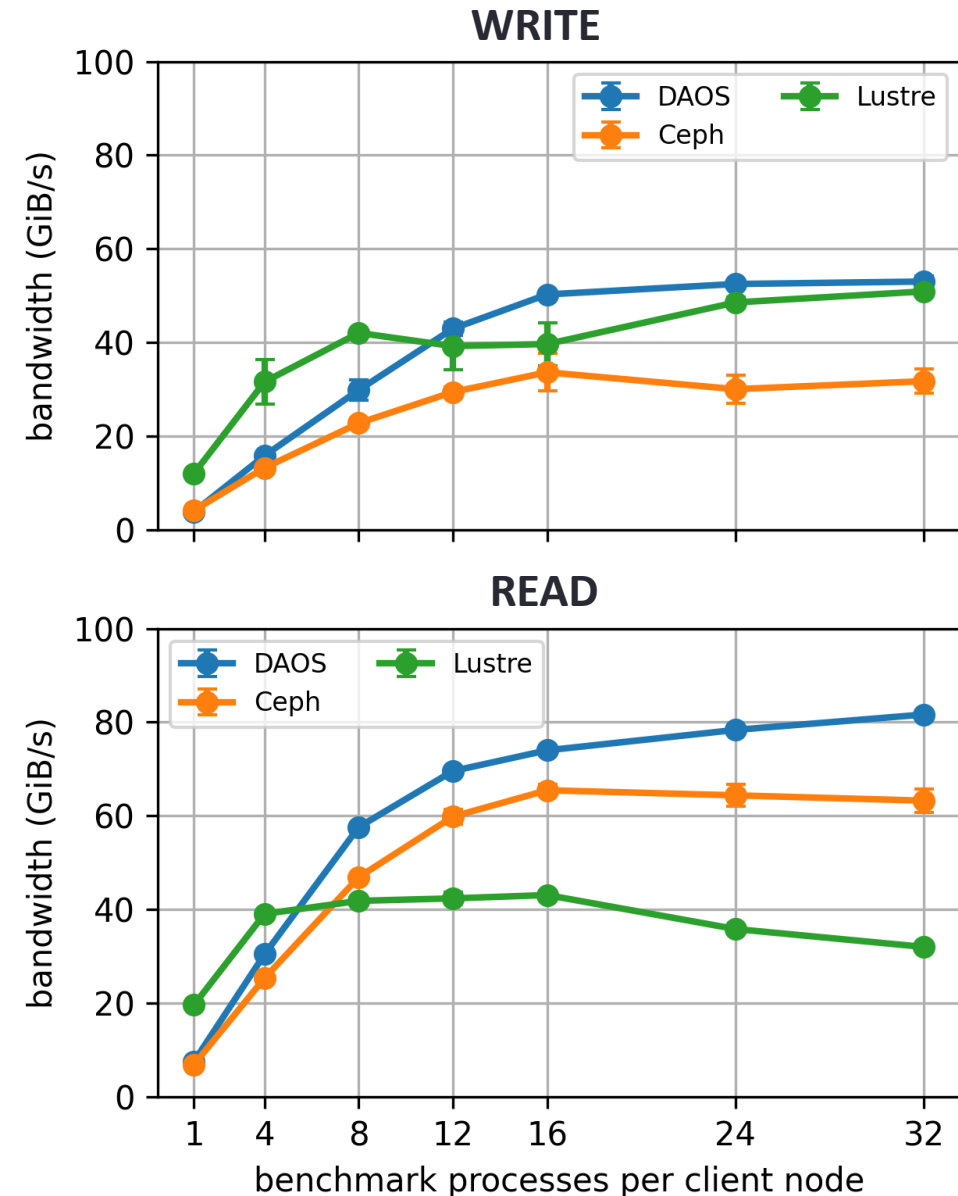
<https://arxiv.org/pdf/2409.18682>



RADOS performance on NVMe

- DAOS reaches close to hardware bandwidths for both write and read
- RADOS performance is lower but decent
- POSIX/Lustre struggles with metadata operations when managing small (1MiB) objects

<https://arxiv.org/pdf/2409.18682>



RADOS vs DAOS vs Lustre

Feature	RADOS	DAOS	Lustre
Algorithmic placement	○	○	
Client-side caching	●	●	○
Kernel involved	●		○
Centralised metadata			○
Strictly consistent	○	○	○
Immediately persistent	○	○	●
Provides POSIX files/directories	●	●	○
Provides objects	○	○	
Provides key-values	○	○	
Software-level data safety	○	○	

Feature	RADOS	DAOS	Lustre
High-performance networks		○	○
Byte-addressable		○	
Zero-copying		○	
Can exploit fast storage tier	○	○	○
Supports HDDs	○		○
Can scale to O(10k) nodes	○	○	○
Performs for GiB objects		○	○
Performs for MiB object	○	○	
Performs for KiB objects		○	

○ Yes ● Can do □ No

librados

- Provides functionality to manipulate all entities in RADOS
 - Daemons, Pools, Objects / Omaps
- Available in many programming languages, including
 - C, C++, Python, Java, PHP, ...
 - The C API is the richest and best documented
 - Ceph distributes the rados command-line tool wrapping librados



<https://docs.ceph.com/en/reef/rados/api/librados/>

(RADOS authentication)

- Users can be created in a RADOS cluster
- Users can be granted permissions
 - To perform management actions
 - To access or manipulate certain pools
- Usually, one user per type of application (e.g. admin, RGW, CephFS, ...)
- A Keyring file is generated per user
 - Contains authentication token
 - Is deployed on client nodes usually under `/etc/ceph/ceph.<username>.keyring`

librados – cluster handle

```
int rados_create2(rados_t *pcluster, const char *const clustername,  
                  const char *const name, uint64_t flags)
```

- Initialises a `rados_t` struct given the cluster name and RADOS username to user for subsequent librados calls

librados – configuration

- The minimum configuration required by librados includes
 - address and port of one ceph-mon (ideally of all ceph-mon)
 - path to keyring file
- Can be specified in different ways
 - Manually item by item via `rados_conf_set()`
 - Via environment variables plus `rados_conf_parse_env()`
 - Via a configuration file plus `rados_conf_read_file()`

librados – configuration

```
int rados_conf_read_file(rados_t cluster, const char *path)
```

- If path is NULL, the following paths are checked:
 - \$CEPH_CONF environment variable
 - /etc/ceph/ceph.conf
 - ~/.ceph/config
 - ceph.conf in the current working directory
- If a keyring path is not given in the configuration (“keyring” item), a keyring file with the name `ceph.<username>.keyring` is looked for in the same directories

librados – cluster connect

int rados_connect(rados_t cluster)

- Opens a connection with a RADOS cluster
- If succeeds (`rc == 0`), it must be released with `rados_shutdown()`

int rados_shutdown(rados_t cluster)

- Closes an open connection with a RADOS cluster

(RADOS pool create – admin only)

```
ceph osd pool create ${pool_name} ${pg_count} ${pgp_count} replicated
```

- Creates a replicated pool
- Defaults to 3 replicas

```
ceph osd pool set ${pool_name} size ${replica_count}
```

- Sets the replica count for a replicated pool

(RADOS pool create – admin only)

```
ceph osd erasure-code-profile set myprofile k=${k} m=${m} \  
crush-failure-domain=host
```

- Creates an erasure-code profile

```
ceph osd pool create ${pool_name} ${pgc} ${pgpc} erasure [myprofile]
```

- Creates an erasure-coded pool with a given EC profile
- Defaults to 2+2
- Does not support omap objects

librados – pool connect

```
int rados_ioctx_create(rados_t cluster, const char *pool_name,  
                      rados_ioctx_t *ioctx)
```

- Initialises an IO context struct for an existing pool
- The rados_ioctx_t allows performing IO operations on a pool
- If succeeds (`rc == 0`), it must be released with `rados_ioctx_destroy()`

```
int rados_ioctx_destroy(rados_ioctx_t io)
```

- Signals librados that an IO context will no longer (and must not) be used
- The iocxt may not be destroyed immediately if it holds active async operations

librados – namespaces

```
void librados_ioctx_set_namespace(rados_ioctx_t io, const char *nspace)
```

- Sets the namespace to use for a given IO context
- nspace can be set to LIBRADOS_ALL_NAMESPACES to list all objects in a pool with `rados_nobjects_list_open()`

librados – I/O APIs

- Synchronous I/O
 - Blocks until operation is complete
 - Provides methods to perform I/O to regular objects
- Transactional API
 - Allows specifying a set of operations to be performed atomically
 - Blocks until transaction is complete
 - Provides methods to perform I/O to regular objects as well as Omaps
- Async API
 - Does not block unless `wait_for_complete()` or `flush()` are called
 - Provides the same I/O features as the transactional API

librados – synchronous I/O

```
int rados_write(rados_ioctx_t io, const char *oid, const char *buf,  
               size_t len, uint64_t off)
```

- Write len bytes from buf into the oid object, starting at offset off
- Creates object if n.e.
- Returns 0 on success or a negative value on failure

```
int rados_write_full(rados_ioctx_t io, const char *oid,  
                    const char *buf, size_t len)
```

- If the object exists, it is atomically truncated and then written

librados – synchronous I/O

```
int rados_append(rados_ioctx_t io, const char *oid, const char *buf,  
                size_t len)
```

- Append len bytes from buf into the oid object
- Returns 0 on success or a negative value on failure

```
int rados_read(rados_ioctx_t io, const char *oid, char *buf,  
              size_t len, uint64_t off)
```

- Read len bytes starting at off from object into buf
- Returns number of read bytes on success or a negative value on failure

librados – synchronous I/O

```
int rados_remove(rados_ioctx_t io, const char *oid)
```

- Delete an object.
- Returns 0 on success or a negative value on failure.

```
int rados_trunc(rados_ioctx_t io, const char *oid, uint64_t size)
```

- Resize an object to size.
- If shrinking, the excess data is deleted. If enlarging, filled with zeros.

librados – transactional I/O

- A single “rados operation” can perform multiple operations on one object atomically
- The whole operation will succeed or fail, and no partial results will be visible
- Operations may be either reads, which can return data, or writes, which cannot
- The effects of writes are applied and visible all at once
 - E.g. an operation that sets an xattr and then checks its value will not see the updated value

create_write_op → write_op_XXX → write_op_YYY → write_op_operate → release_write_op
create_read_op → read_op_XXX → read_op_YYY → read_op_operate → release_read_op

librados – transactional I/O

```
int rados_write_op_operate(rados_write_op_t write_op,  
                           rados_ioctx_t io, const char*oid, time_t *mtime, int flags)
```

```
int rados_read_op_operate(rados_read_op_t read_op,  
                           rados_ioctx_t io, const char*oid, int flags)
```

- Execute a “rados operation” on an object

librados – transactional I/O - Omap

```
void rados_write_op_omap_set2(rados_write_op_t write_op,  
    char const *const *keys, char const *const *vals,  
    const size_t*key_lens, const size_t *val_lens, size_t num)
```

- Insert an array of key-value pairs into an Omap

```
void rados_read_op_omap_get_vals_by_keys2(rados_read_op_t read_op,  
    char const *const *keys, size_t num_keys,  
    constsize_t *key_lens, rados_omap_iter_t *iter, int *prval)
```

- Query the values associated to a given array of keys
- Returns an iterator with the values
- Any failures are signaled via prval

librados – transactional I/O - Omap

```
unsigned int rados_omap_iter_size(rados_omap_iter_t iter)
```

- Returns number of elements in Omap iterator

```
int rados_omap_get_next2(rados_omap_iter_t iter, char **key,  
                        char **val, size_t *key_len, size_t *val_len)
```

- Extracts the next key-value pair from an Omap iterator
- If the end of the list has been reached, key=val=NULL, and keylen=vallen=0
- key and val should be copied as rados_omap_get_end() releases them

```
void rados_omap_get_end(rados_omap_iter_t iter)
```