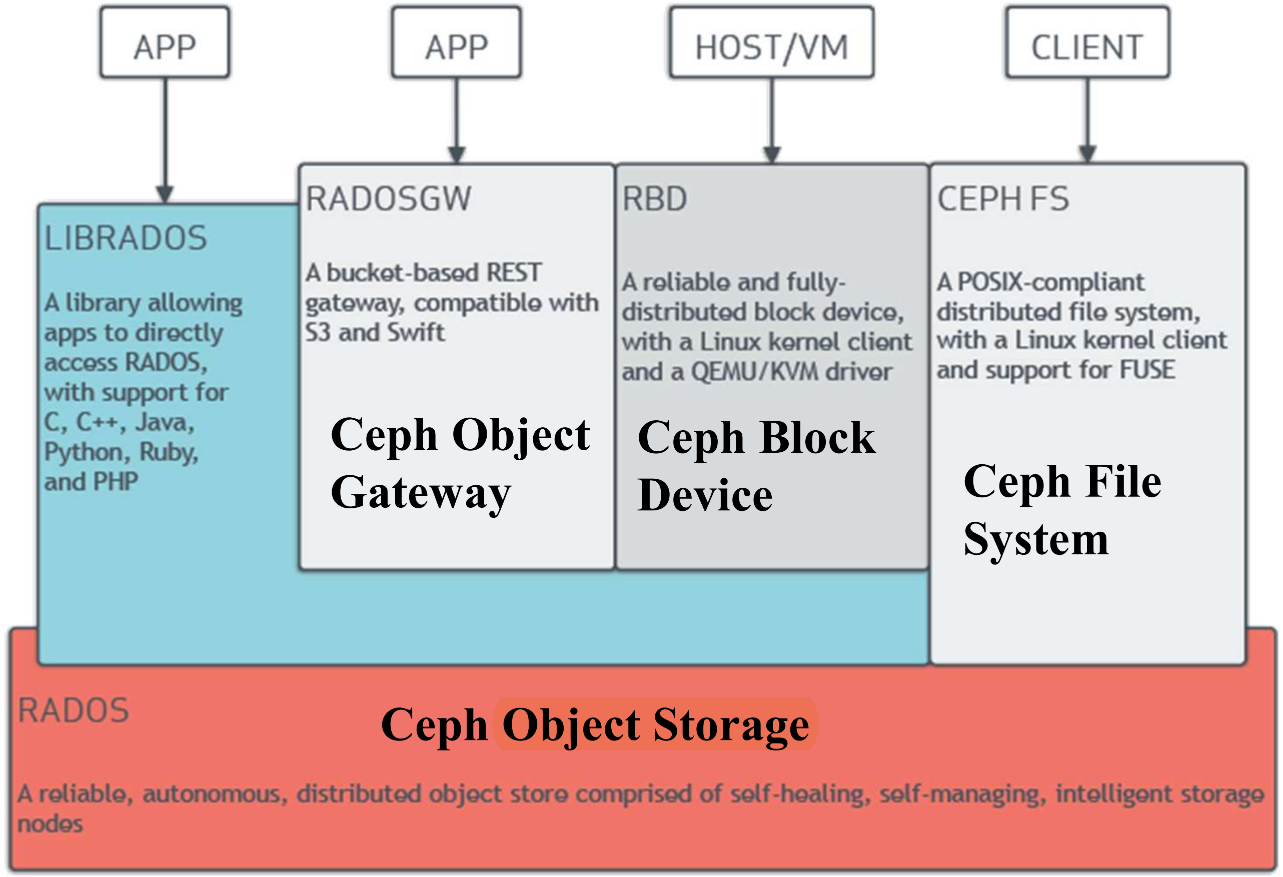
Ceph & RADOS

- **Reading**
  - Ceph: a scalable, high-performance distributed file system, S. A. Weil, et al., OSDI 2006.
- **References**
  - RADOS: a scalable, reliable storage service for petabyte-scale storage clusters, S. A. Weil, et al., PDSW 2007
  - CRUSH Controlled, Scalable, Decentralized Placement of Replicated Data, S. A. Weil, et al., SC2006
  - Ceph Architecture Document (https://docs.ceph.com/en/latest/architecture/)

Red Hat to Acquire Inktank, Provider of Ceph, for $175M, 2014.04.30

Ceph Architecture

- Ceph uniquely delivers object, block, and file storage in one unified system.



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
Ceph Object Gateway
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
Ceph Block Device
CEPH FS
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE
Ceph File System
RADOS
Ceph Object Storage
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

source: ceph (captured 2023.04)

Ceph Distributed File System

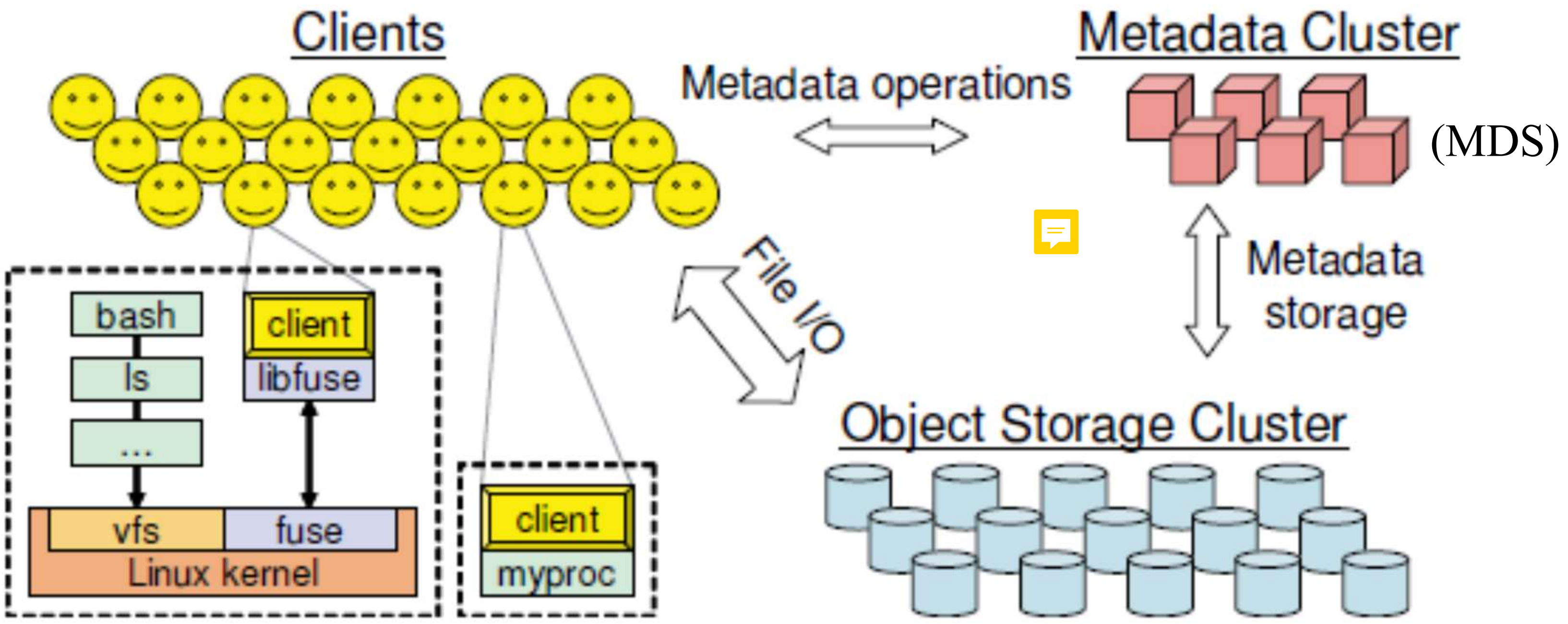
- Design Goal: a **scalable, reliable,** and **high-performance** distributed file system that can support
  - hundreds of peta-byte scale
  - thousands of concurrent reads/writes to the same file or creating files in the same directory (more general purpose as compared to GFS & HDFS).
  - avoid imbalance (e. g., recently deployed devices mostly idle or empty) or load asymmetries (e. g., new, hot data on new devices only),
  - near-POSIX

Key: Decouple data and metadata management by replacing file allocation tables with a pseudo-random data distribution function (CRUSH).

Ceph System Architecture

Clients interact with a metadata server (MDS) to perform metadata operations (open, rename), and communicate with OSDs to perform file I/O (reads and writes).

MDSs manage the namespace (file names and directories) while coordinating security, consistency and coherence.



The client code runs entirely in user space and can be accessed either by linking to it directly or as a mounted file system via FUSE (Filesystem in Userspace)

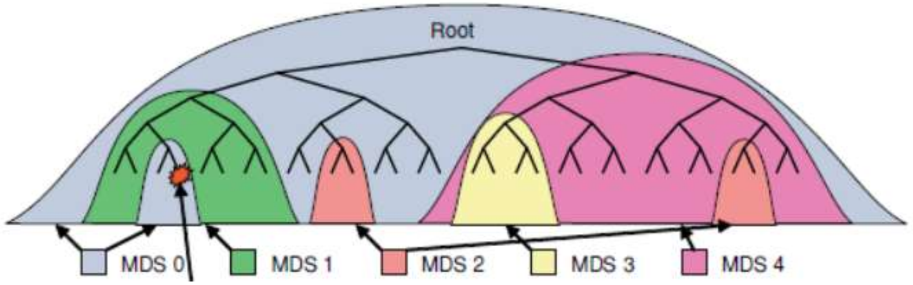
Object storage devices (OSDs), collectively stores all data and metadata

File I/O

- When a process opens a file:
  - the Ceph client sends a request to the MDS cluster.
  - An MDS traverses the file system hierarchy to translate the file name into the file inode (which has file metadata)
  - If file exists, the MDS returns the file inode, file size, capability (read, cache reads, write, and buffer writes), and stripe information (for mapping file into objects).
  - a simple striping strategy:
    - objects of a file are named: (file inode number, stripe number)

Reducing Metadata Server Load

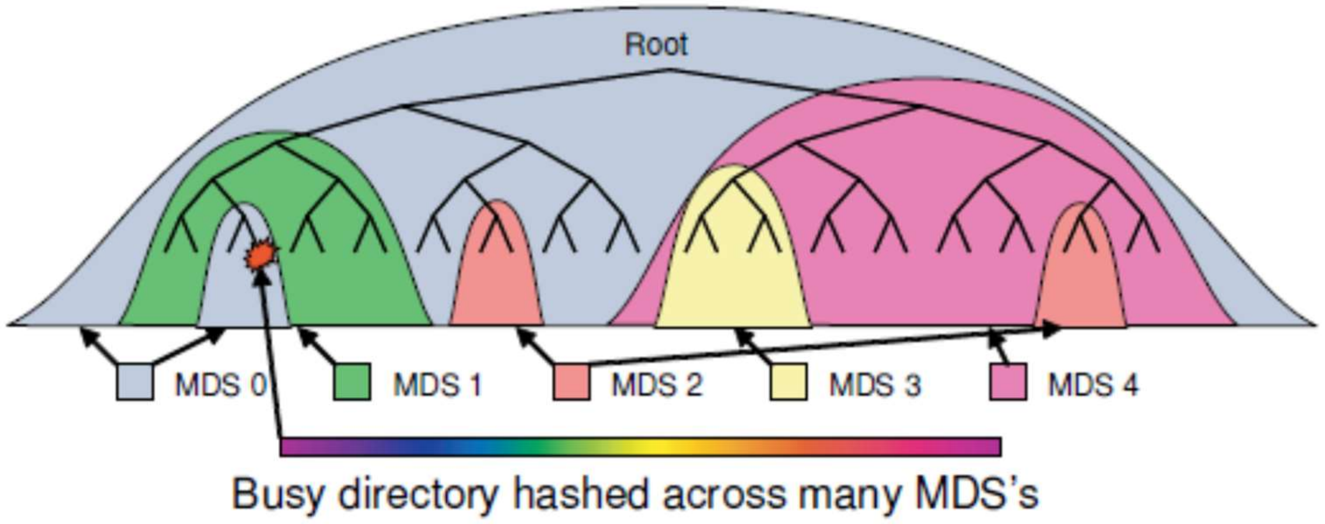
- Metadata operations often make up as much as half of file system workloads [Roselli, et al. 2000]
- File and directory metadata in Ceph are kept small to ease MDS load.
  - Consisting almost entirely of directory entries (file names) and inodes (80 bytes); no file allocation metadata
    - object names are constructed using the inode number
    - directory's content and metadata journals are written to the OSD cluster similarly as file data.
  - Use Dynamic Subtree Partitioning to further minimize metadata related disk I/O.



Dynamic Subtree Partitioning

- Use primary-copy caching strategy to let an authoritative MDS responsible for managing cache coherence and serializing updates for any given piece of metadata.
- Uses dynamic subtree partitioning strategy to delegate the primary authority to the MDS cluster.
  - Each MDS measures the popularity of metadata within the directory hierarchy as follows:
    - use a counter with an exponential time decay.
    - any operation increments the counter on the affected inode and all of its ancestors up to the root directory, providing each MDS with a weighted tree describing the recent load distribution.
  - MDS load values are periodically compared, and appropriately-sized subtrees of the directory hierarchy are migrated to keep the workload evenly distributed.

Dynamic Subtree Partitioning (2)



- Ceph dynamically maps subtrees of the directory hierarchy to metadata servers based on the current workload.
- The tree-like distribution preserves locality in each MDS's workload.
- Individual directories are hashed across multiple nodes only when they become hot spots (at the expense of locality).

Why not use static subtree-based partitioning?

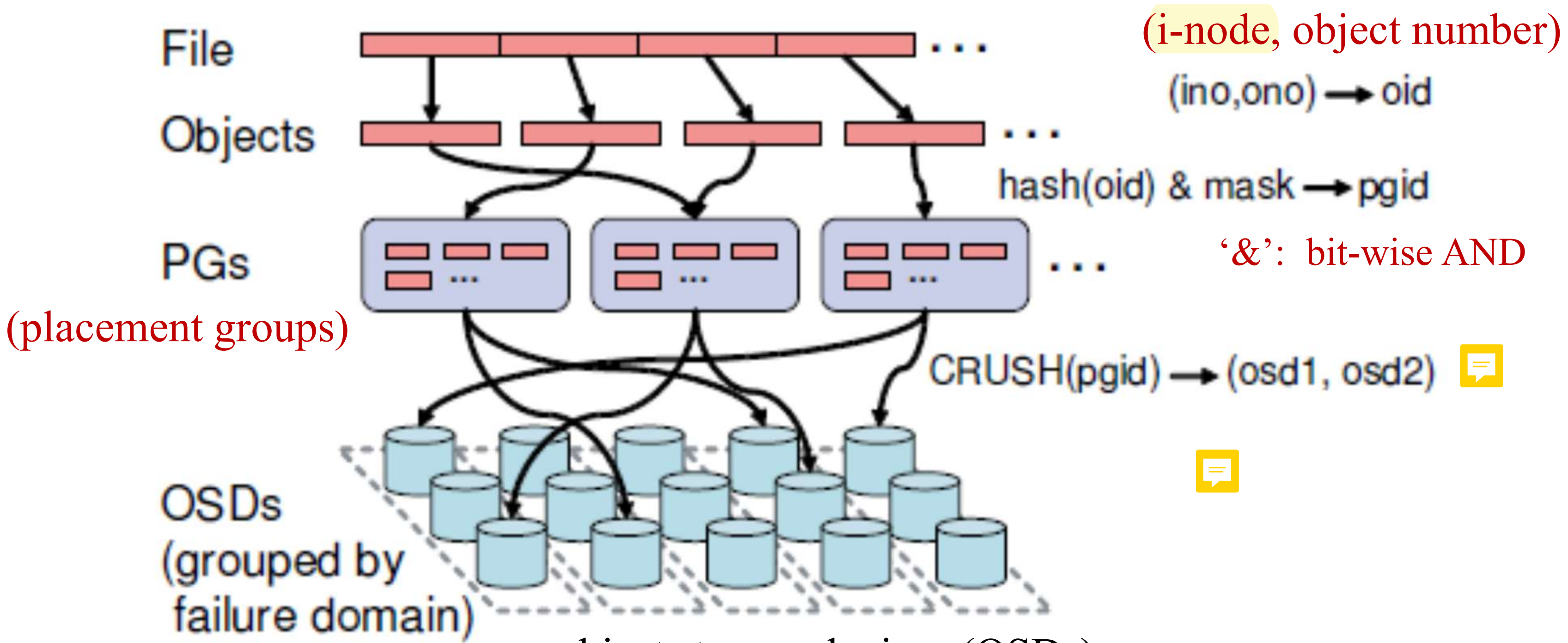
POSIX

- POSIX: reads reflect any data previously written, and writes are atomic
- Ceph ensures POSIX semantics by
  - when a file is opened by multiple readers and writers, the MDS will revoke any previously issued read caching and write buffering capabilities, forcing client I/O for that file to be synchronous.
  - each application read or write operation will block until it is acknowledged by the OSD, effectively placing the burden of update serialization and synchronization with the OSD storing each object.
  - When writes span object boundaries, clients acquire exclusive locks on the affected objects (granted by their respective OSDs)

near-POSIX

- synchronous I/O is a performance killer
- Ceph supports some I/O that allow performance-conscious applications to explicitly relax the usual coherency requirements for a shared-write file.
  - Applications manage their own consistency
    - e.g., by writing to different parts of the same file
  - buffer writes or cache reads are allowed (which otherwise would be performed synchronously)

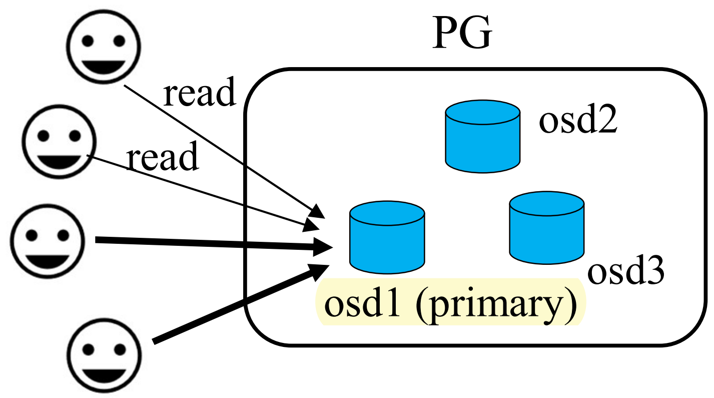
Data Distribution with CRUSH



object storage devices (OSDs).

Files are striped across many objects, grouped into placement groups (PGs), and distributed to OSDs via CRUSH, a specialized replica placement function, based on *placement rules*.

Replication



- ➢ Each object has a PG (with a version no.), and within a PG there is a primary OSD.
- ➢ Clients send all writes to the first non-failed OSD in an object's PG (the **primary**), which assigns a new version number for the object and PG and forwards the write to any additional replica OSDs. After each replica has applied the update and responded to the primary, the primary applies the update locally and the write is acknowledged to the client.
- ➢ Reads are also directed at the primary.

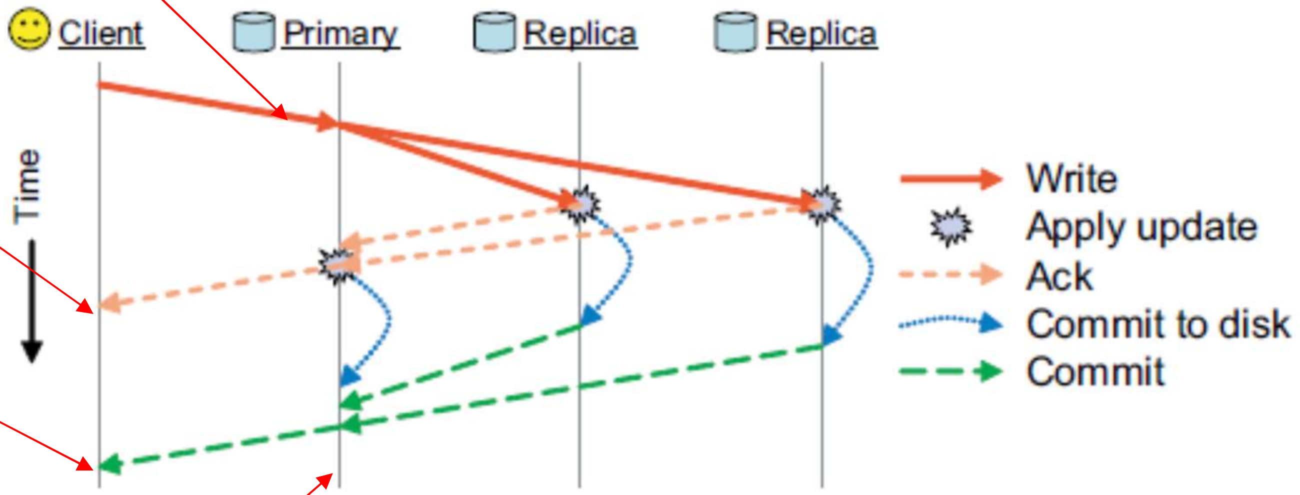
- ✓ Why not direct Reads to other replicas?
- ✓ Would the primary OSD be a hot spot?

Replication & Read/Write

Clients send all writes to the first non-failed OSD in an object's PG (the primary), which then forwards the write to any additional replica OSDs.

in-memory buffer caches

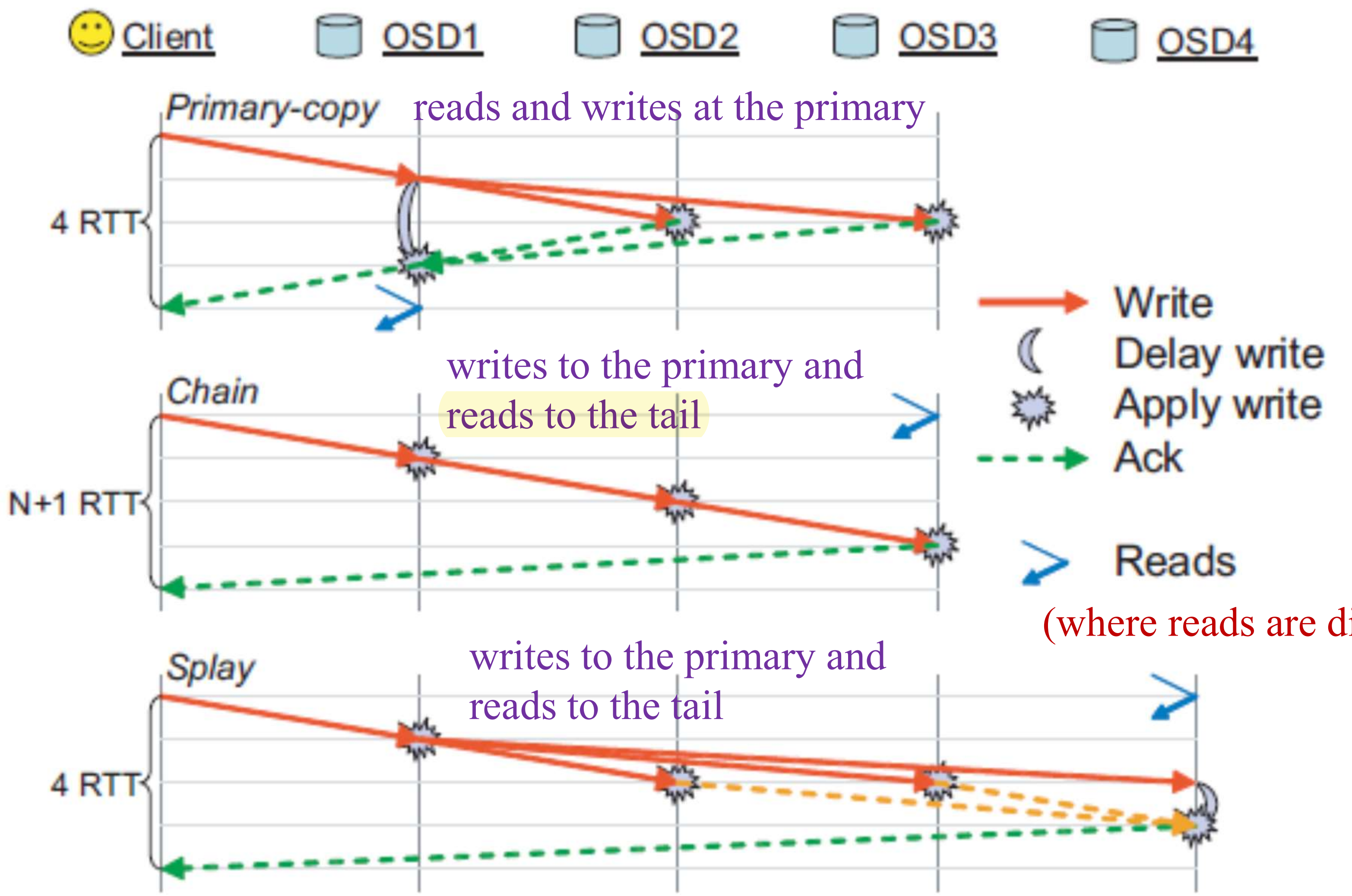
data is safely committed to disk



After each replica has applied the update and responded to the primary, the primary applies the update locally and the write is acknowledged to the client

reads are directed to the primary

More Replication Strategies



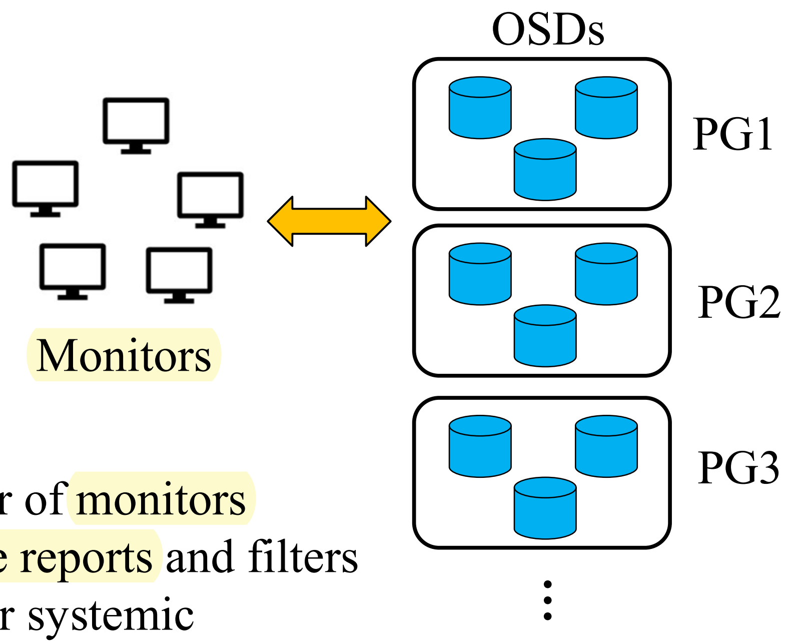
reads and writes at the primary

writes to the primary and reads to the tail

writes to the primary and reads to the tail

(where reads are directed to)

Failure Detection & OSD Monitors

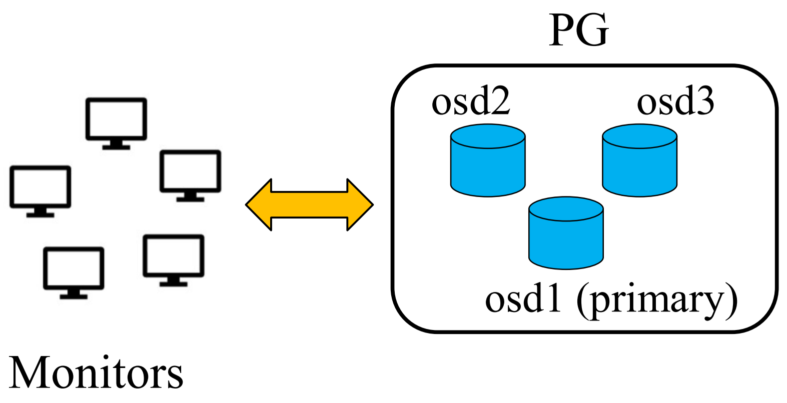


OSDs in a PG communicate regularly to replicate objects, so they can detect if their peers are down and report the failures.

A small cluster of monitors collects failure reports and filters out transient or systemic problems (e.g., a network partition) centrally.

Monitors use elections, active peer monitoring, short-term leases, and two-phase commits to collectively provide consistent and available access to the **cluster map.**

Recovery and Cluster Updates



Each object has a PG (and a PG may be responsible for more than one object).

Within a PG there is a primary OSD.

OSDs maintain a version number for each object and a log of recent changes.

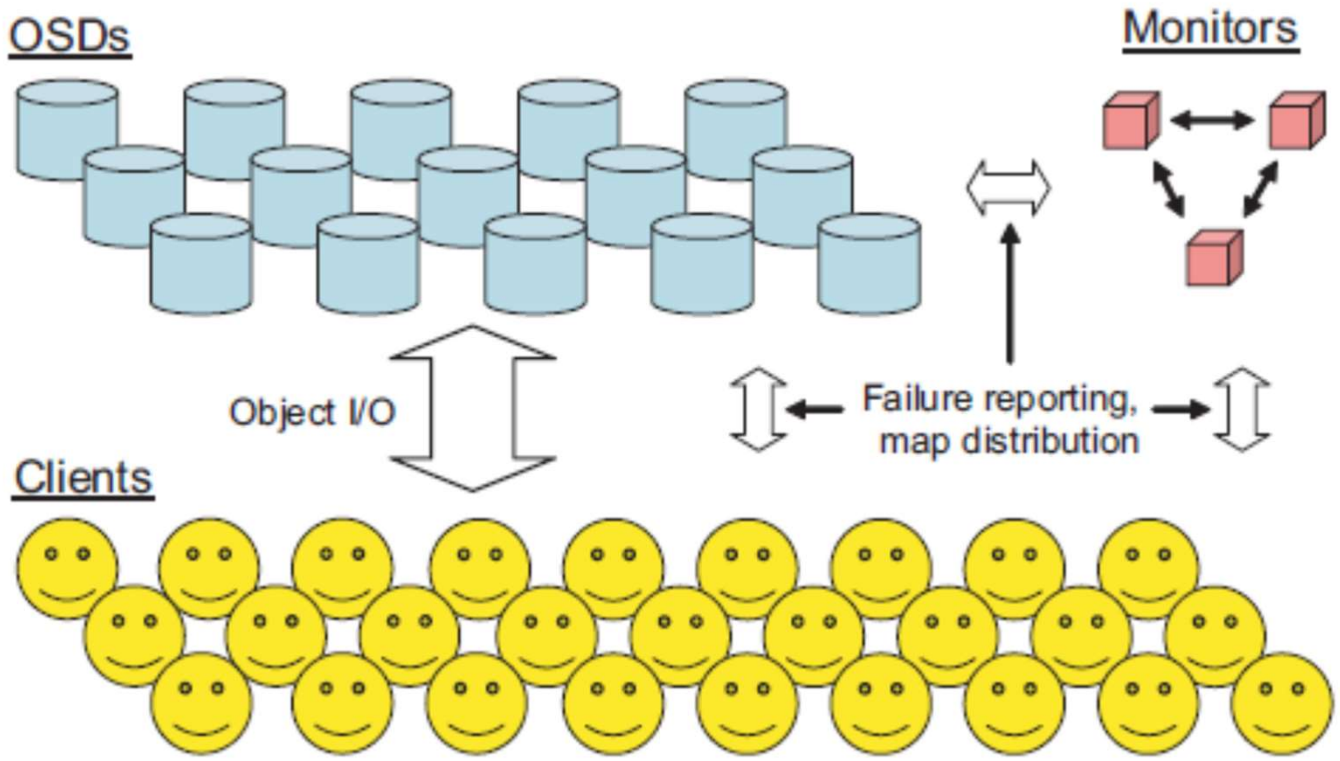
- When an active OSD receives an updated cluster map (due to some OSD add, down or up), it iterates over all locally stored PGs and determine (by CRUSH mapping) which ones it is responsible for (as a primary. or as a replica).
- A primary OSD collects current (and former) replicas' PG versions, and then sends each replica an incremental log update (or a complete content summary, if needed) to synchronize their knowledge of the PG contents.
- Only after the primary determines the correct PG state and shares it with replicas is I/O to objects in the PG permitted. OSDs are then independently responsible for retrieving (in the background) missing or outdated objects from their peers.

As such, each PG affected by a failed OSD will recover in parallel to other PGs.

RADOS
(Reliable Autonomic Distributed Object Store)

- A reliable object storage service that can scales to many thousands of devices by leveraging the intelligence present in individual storage nodes.
- Preserves consistent data access and strong safety semantics while allowing nodes to act semi-autonomously to self-manage replication, failure detection, and failure recovery through the use of a small cluster map.

A RADOS System



A RADOS system consists of a large collection of OSDs (object storage devices) and a small group of monitors responsible for managing OSD cluster membership.

Source: RADOS: a scalable, reliable storage service for petabyte-scale storage clusters, S. A. Weil, et al., PDSW 2007.

Placement Group

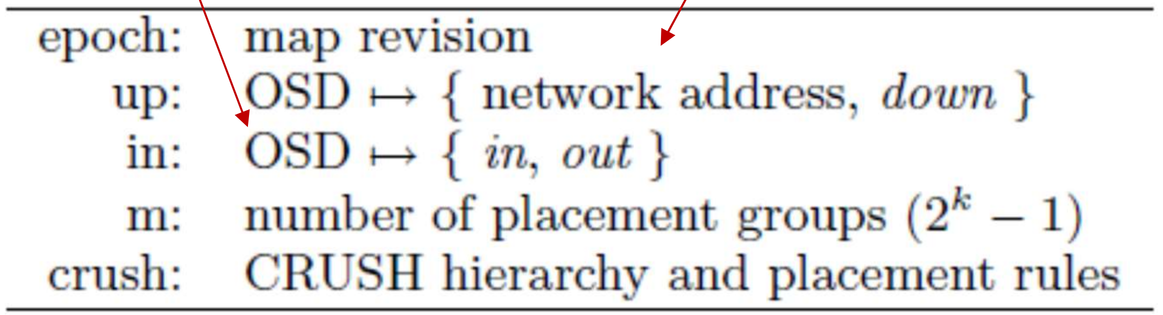
- Each object stored by the system is first mapped into a *placement group* (PG) to determine a logical collection of objects that are replicated by the same set of devices.
  - pgid = (r, hash(o) & m)
    - where r is the desired level of replication
    - o is the object name
    - & is a bit-wise AND
    - m is a bit mask to control the total number of PGs in the system
- Placement groups are assigned to OSDs based on the cluster map, which maps each PG to an ordered list of *r* OSDs upon which to store object replicas.
  - Implemented by CRUSH (see later slides)

RADOS Cluster Map

RADOS avoids initiating widespread data re-replication due to systemic problems by marking OSDs down but not out.

whether an OSD is included or excluded in this epoch

whether an OSD is up (with a network addr to connect to) or is down



epoch:	map revision
up:	OSD $\mapsto$ { network address, *down* }
in:	OSD $\mapsto$ { *in*, *out* }
m:	number of placement groups ($2^k - 1$)
crush:	CRUSH hierarchy and placement rules

The cluster map specifies cluster membership, device state, and the mapping of data objects to devices.

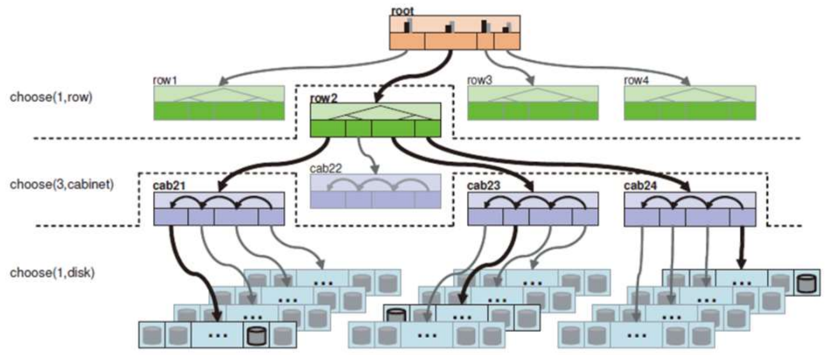
Source: RADOS: a scalable, reliable storage service for petabyte-scale storage clusters, S. A. Weil, et al., PDSW 2007.

CRUSH
(Controlled Replication Under Scalable Hashing)

- A scalable pseudorandom data distribution function designed for distributed object-based storage systems that efficiently maps data objects to storage devices without relying on a central directory.
  - Facilitate the addition and removal of storage while minimizing unnecessary data movement.
- Motivation: Static placement of file blocks will result in an imbalanced load distribution
  - new disks either sit empty or contain only new data (as data is rarely moved once it is written)
  - Why not just use hash to randomly distribute objects to storage devices?

Replica Placement

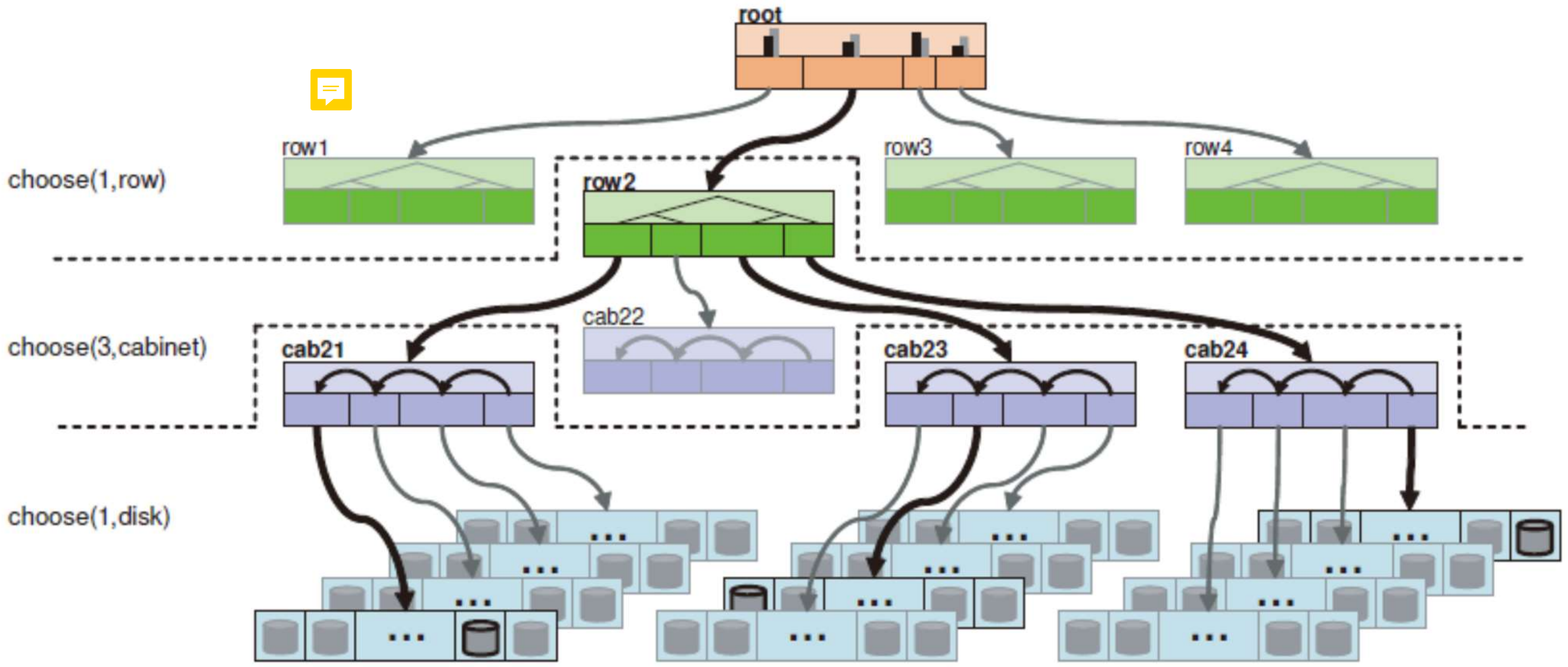
- Given an input *x* (e.g., object name), CRUSH outputs an ordered list R of *n* distinct storage targets.
  - Use a hierarchical **cluster map**, composed of *devices* and *buckets,* to represent available storage resources
    - e.g., rows of server cabinets, cabinets filled with disk shelves, and shelves filled with storage devices.
  - Use **placement rules** to specify how many replica targets are chosen from the cluster and the restrictions
    - e.g., three mirrored replicas are to be placed on devices in different physical cabinets



Action	Resulting $\vec{i}$
take(root)	root
select(1,row)	row2
select(3,cabinet)	cab21 cab23 cab24
select(1,disk)	disk2107 disk2313 disk2437
emit	

take(a) selects an item (typically a bucket) within the storage hierarchy and assigns it to a vector I, which serves as an input to subsequent operations.

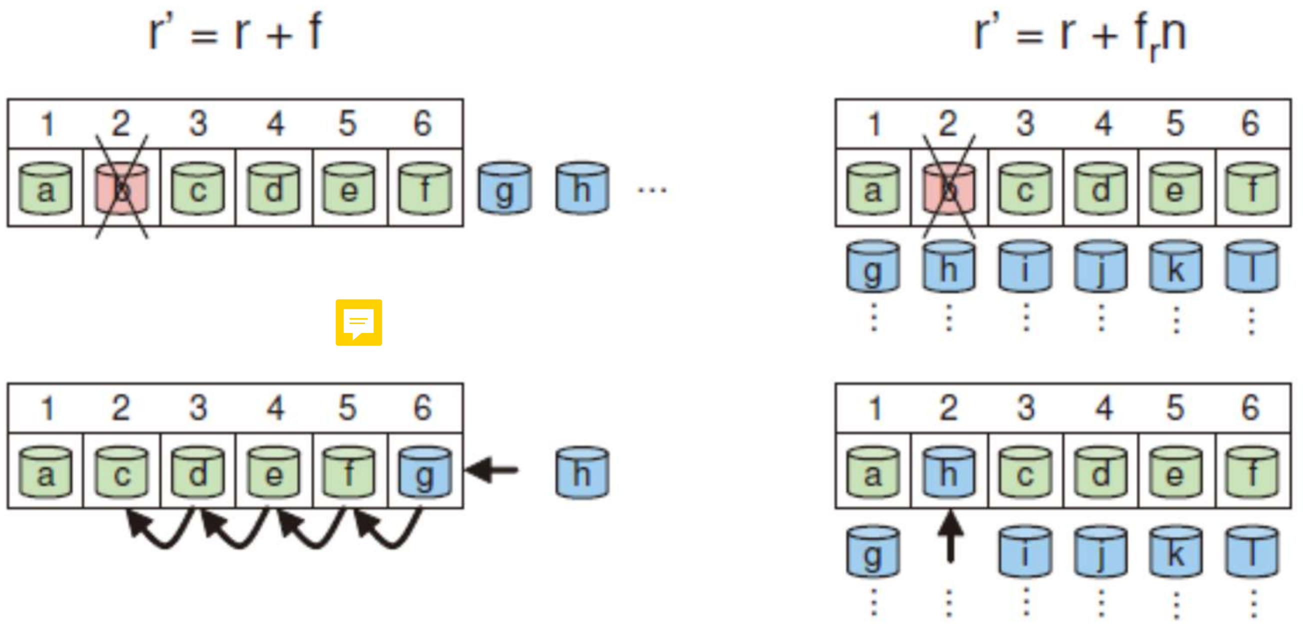
select(n,t) iterates over each element i ∈ I from previous operation, and chooses n distinct items of type t in the subtree rooted at that point (for subsequent operations)



Handling Collisions, Failure, and Overload

- select(n,t) may reject and reselect items using a modified input r′ for three
- different reasons: if an item has already been selected in
- the current set (a collision—the select(n,t) result must be
- distinct), if a device is failed, or if a device is overloaded.

Device Failures and Additions



- Reselection behavior of select(6,disk) when device r = 2 (b) is rejected, where the boxes contain the CRUSH output R of n = 6 devices numbered by rank.
- The left shows the "first n" approach in which device ranks of existing devices (c,d,e, f) may shift. (suitable for **primary copy replication** scheme)
- On the right, each rank has a probabilistically independent sequence of potential targets; here fr = 1, and r′ =r+ frn=8 (device h) (suitable **erasure coding** schemes)

Bucket Types and Choosing Functions

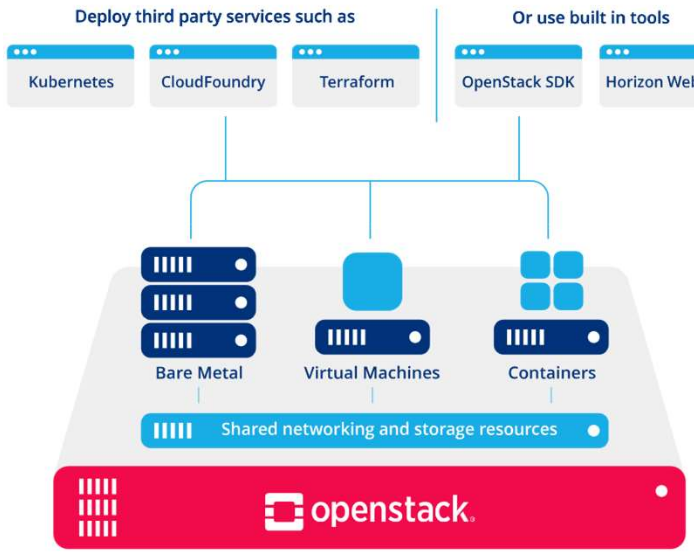
- CRUSH defines four types of buckets to represent internal nodes in the cluster hierarch:
  - Uniform Buckets
  - List Buckets
  - Tree Buckets
  - Straw Buckets
- They affect efficiency and scalability of the mapping algorithm, and the amount of data migration when the cluster changes due to the addition or removal of devices.

See the CRUSH paper if you are interested in the detail.

OpenStack

openstack

- OpenStack is a set of software components that provide common services for cloud infrastructure.
- OpenStack began in 2010 as a joint project of Rackspace Hosting and NASA. As of 2012, it was managed by the non-profit OpenStack Foundation.
- OpenStack cloud at CERN has over 300,000 cores to meet the needs of the Large Hadron Collider (source, 2022.09.30)
- Ceph's Storage Clusters (RADOS) can provide back-end storage for OpenStack.



source: openstack