ceph

architecting block and object geo-replication solutions with ceph

sage weil – inktank

sdc – 2013.09.6.11

overview

- a bit about ceph
- geo-distributed clustering and DR for radosgw
- disaster recovery for RBD
- cephfs requirements
- low-level disaster recovery for rados
- conclusions

distributed storage system

- large scale
  - 10s to 10,000s of machines
  - terabytes to exabytes
- fault tolerant
  - no single point of failure
  - commodity hardware
- self-managing, self-healing

ceph

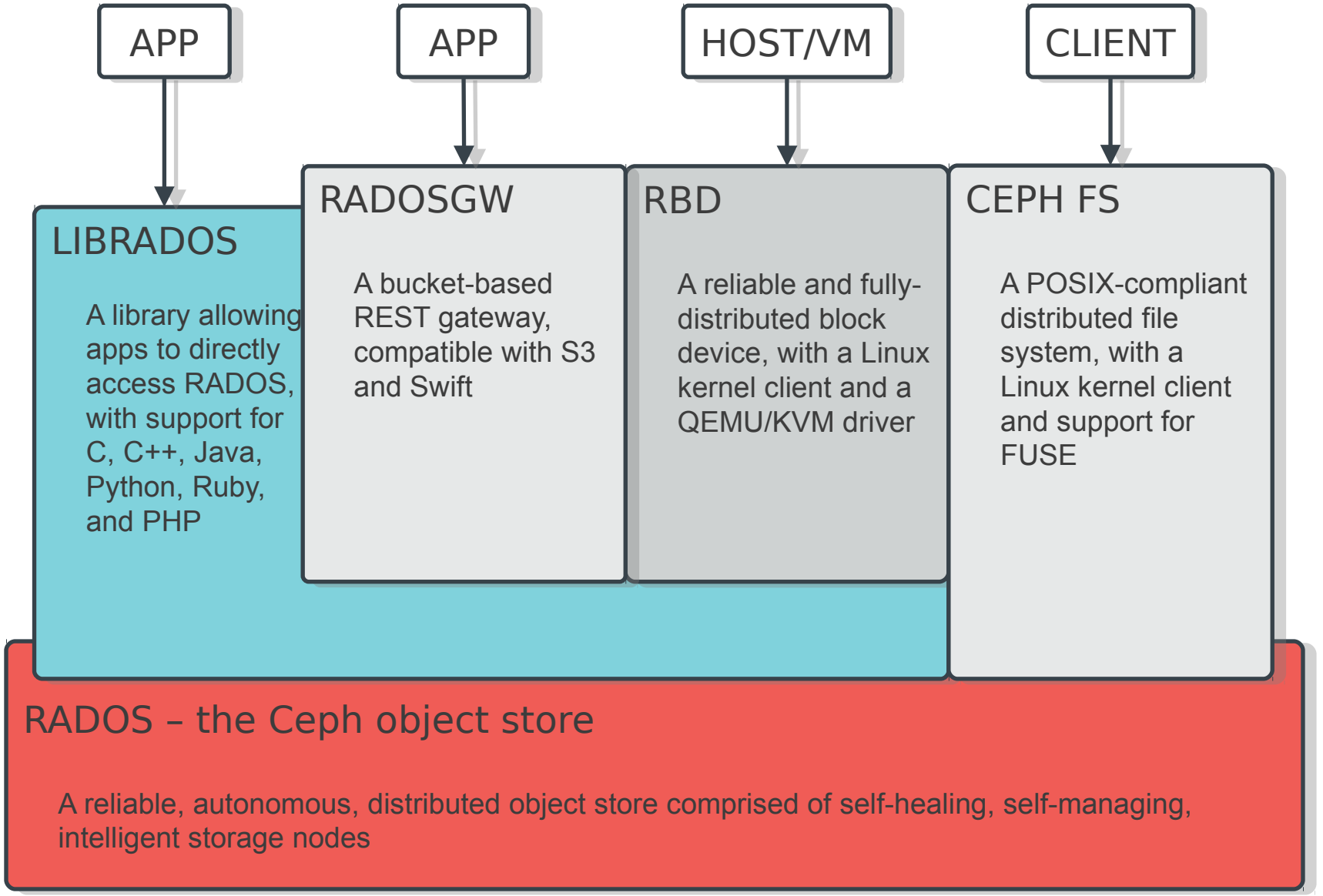
unified storage platform

- objects
  - rich native API
  - compatible RESTful API
- block
  - thin provisioning, snapshots, cloning
- file
  - strong consistency, snapshots

ceph

architecture features

- smart storage daemons
  - centralized coordination does not scale
  - richer storage semantics
  - emergent cluster behavior
- flexible object placement
  - “smart” hash-based placement (CRUSH)
  - avoid lookups in data path
  - awareness of hardware infrastructure, failure domains

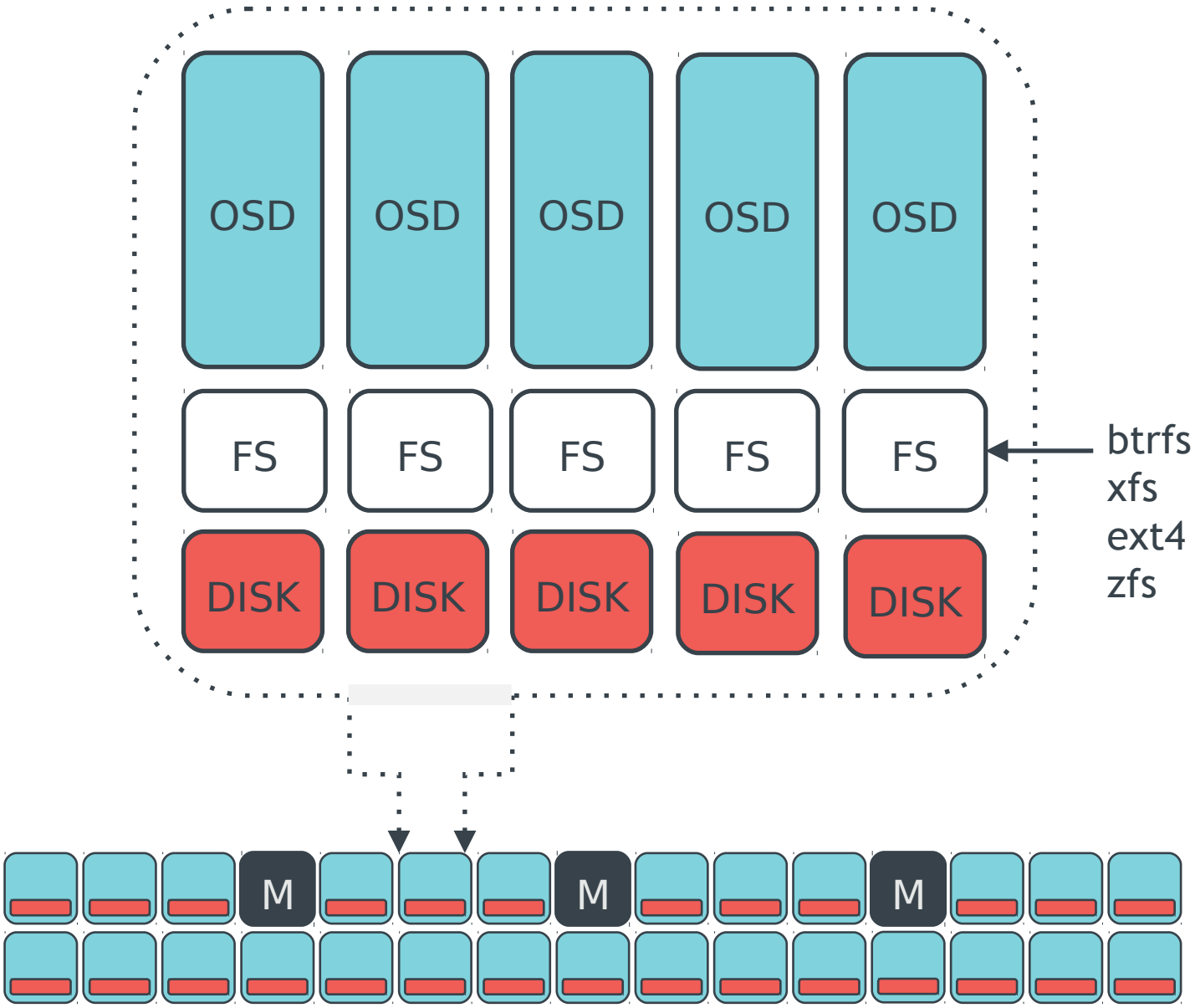

APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver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

ceph

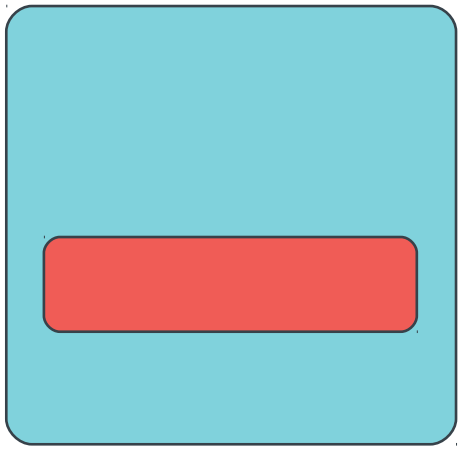
why start with objects?

- more useful than (disk) blocks
  - names in a simple flat namespace
  - variable size
  - relatively simple API with rich semantics
- more scalable than files
  - no hard-to-distribute hierarchy
  - update semantics do not span objects
  - workload is trivially parallel

ceph


OSD
OSD
OSD
OSD
OSD
FS
FS
FS
FS
FS
DISK
DISK
DISK
DISK
DISK
btrfs
xfs
ext4
zfs
M
M
M

ceph



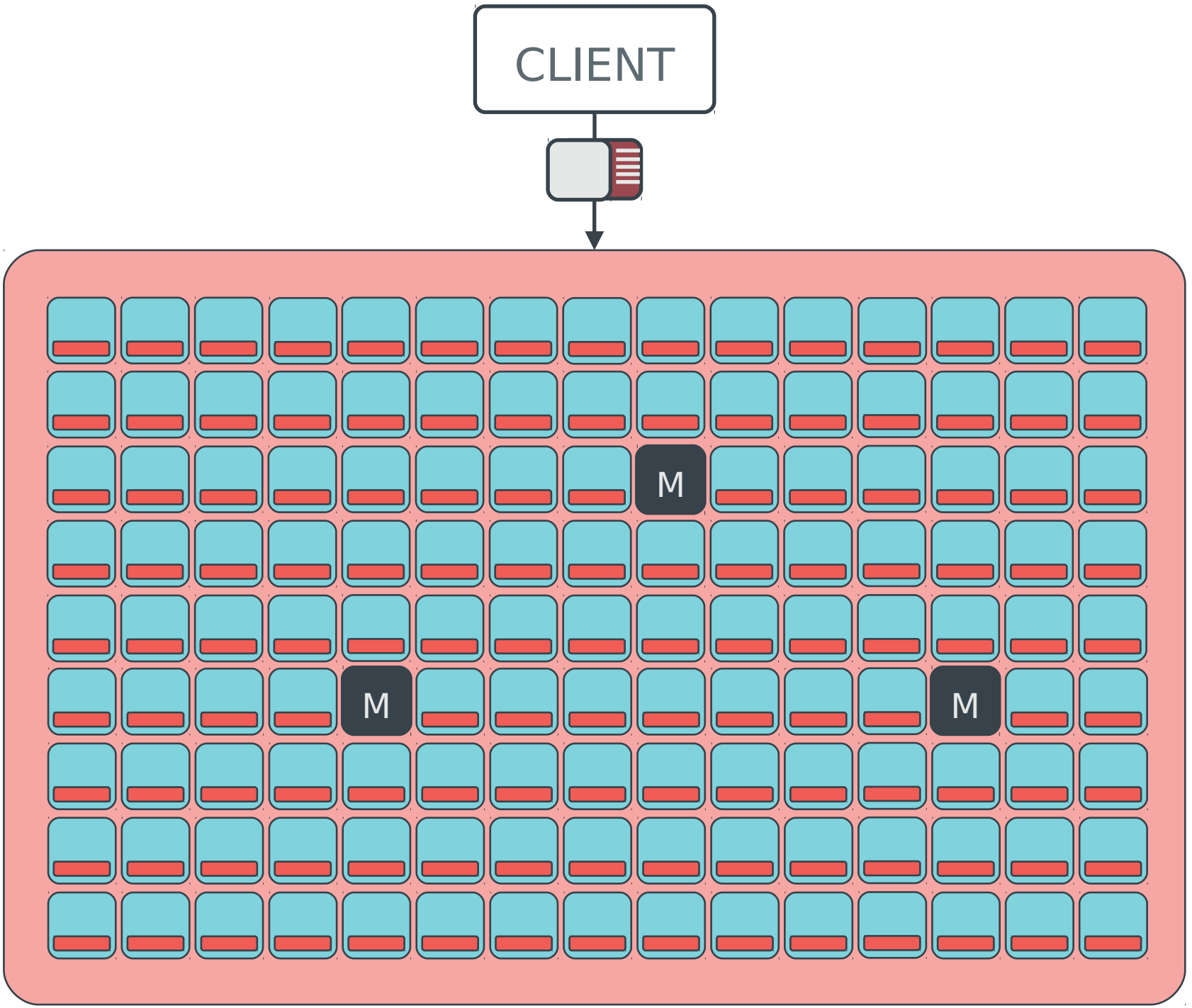
Object Storage Daemons (OSDs)

- 10s to 10000s in a cluster
- one per disk, SSD, or RAID group, or ...
  - hardware agnostic
- serve stored objects to clients
- intelligently peer to perform replication and recovery tasks

Monitors

- maintain cluster membership and state
- provide consensus for distributed decision-making
- small, odd number
- these do not served stored objects to clients

ceph

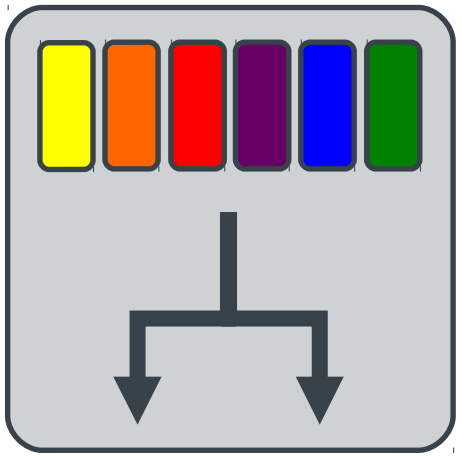


ceph

data distribution

- all objects are replicated N times
- objects are automatically placed, balanced, migrated in a dynamic cluster
- must consider physical infrastructure
  - ceph-osds on hosts in racks in rows in data centers

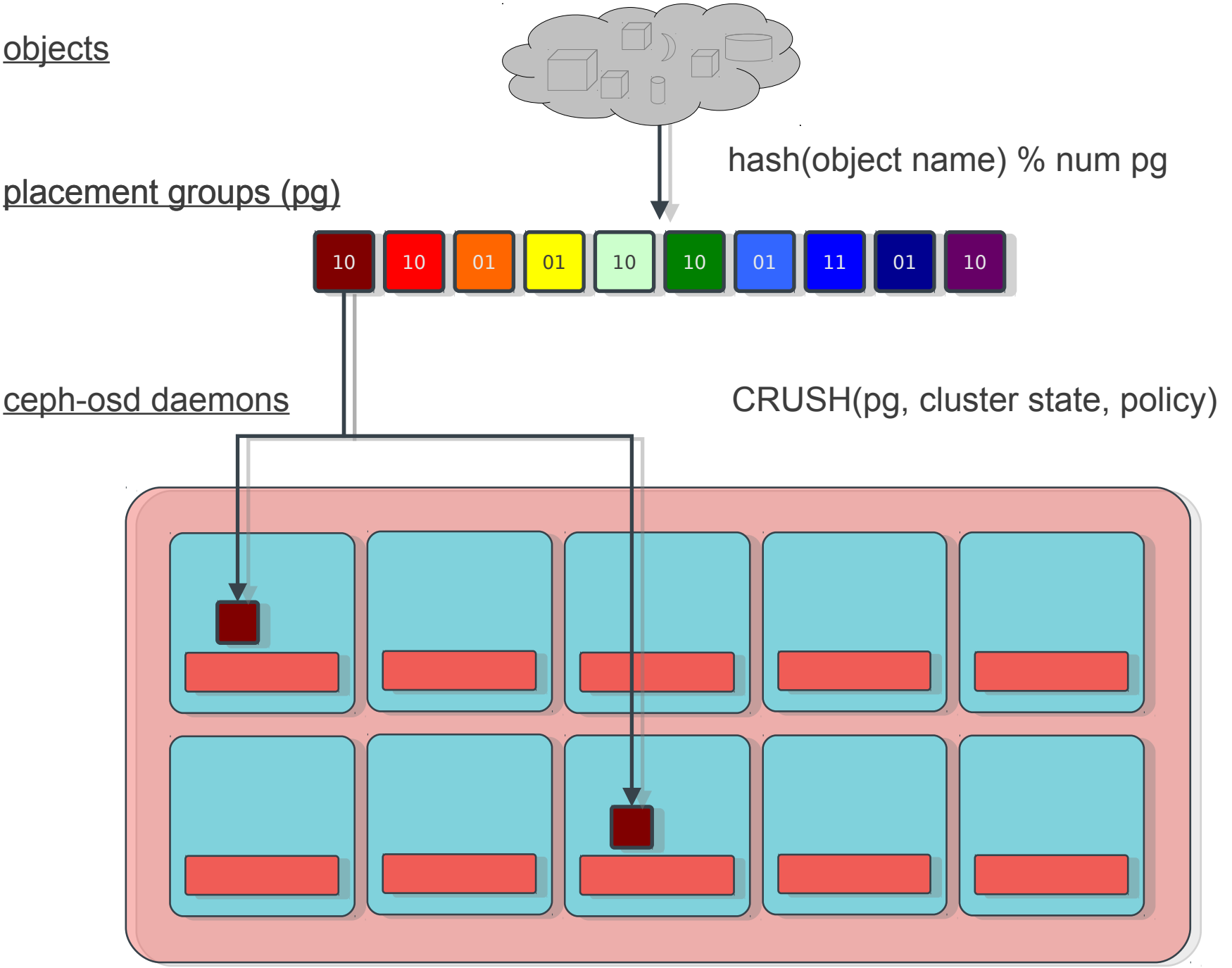
- three approaches
  - user tracked: pick a spot; remember where you put it
  - lookup: pick a spot; write down where you put it
  - functional: calculate where to put it, where to find it



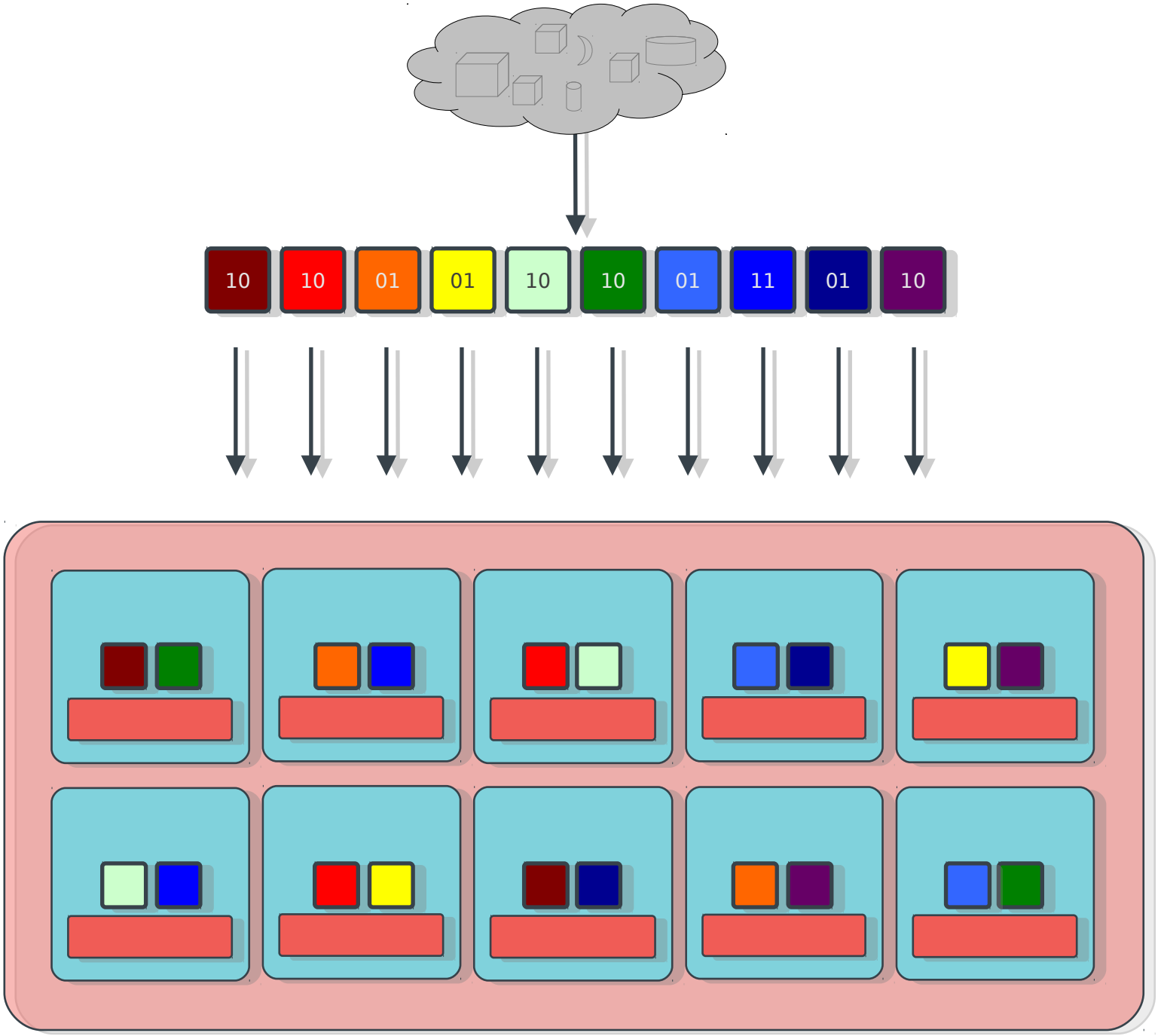
CRUSH

- pseudo-random placement algorithm
  - fast calculation, **no lookup**
  - repeatable, deterministic
- statistically uniform distribution
- stable mapping
  - limited data migration on change
- rule-based configuration
  - infrastructure topology aware
  - adjustable replication
  - allows weighting

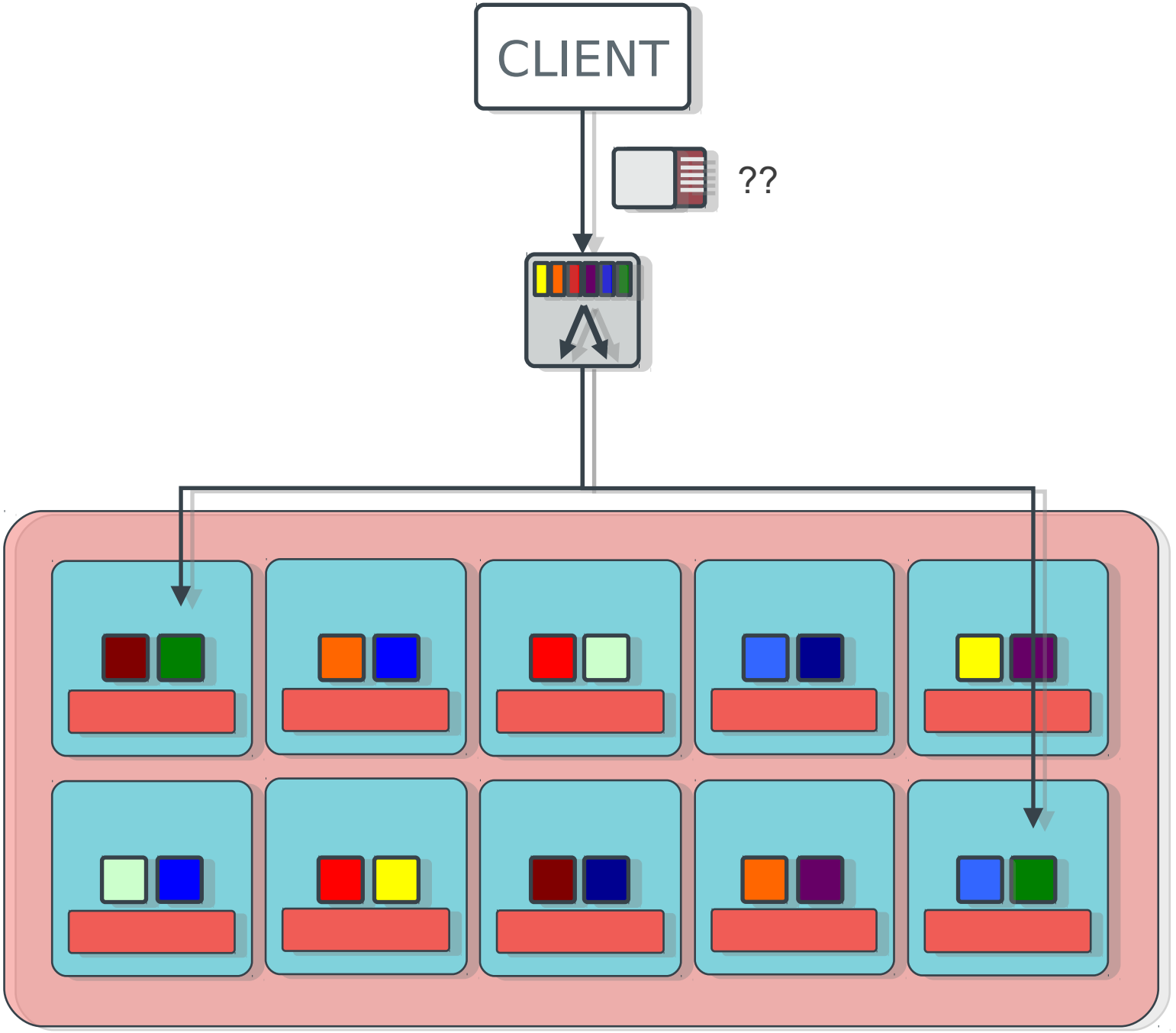
ceph


objects
hash(object name) % num pg
placement groups (pg)
10
10
01
01
10
10
01
11
01
10
ceph-osd daemons
CRUSH(pg, cluster state, policy)

ceph



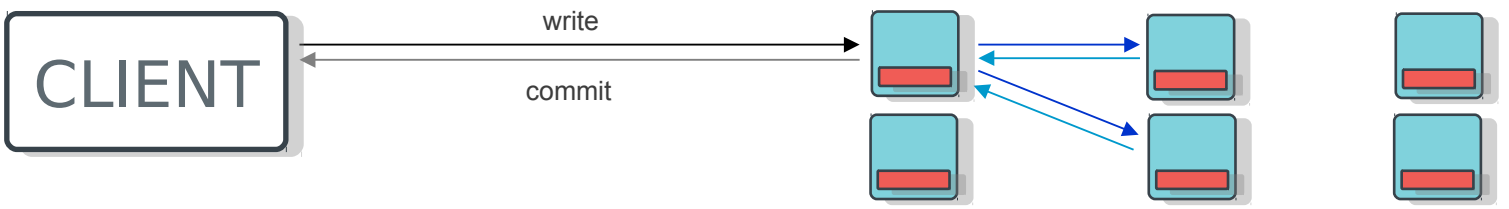
ceph



ceph

rados replication

- writes are replicated synchronously
- strong consistency model (i.e., read after write)

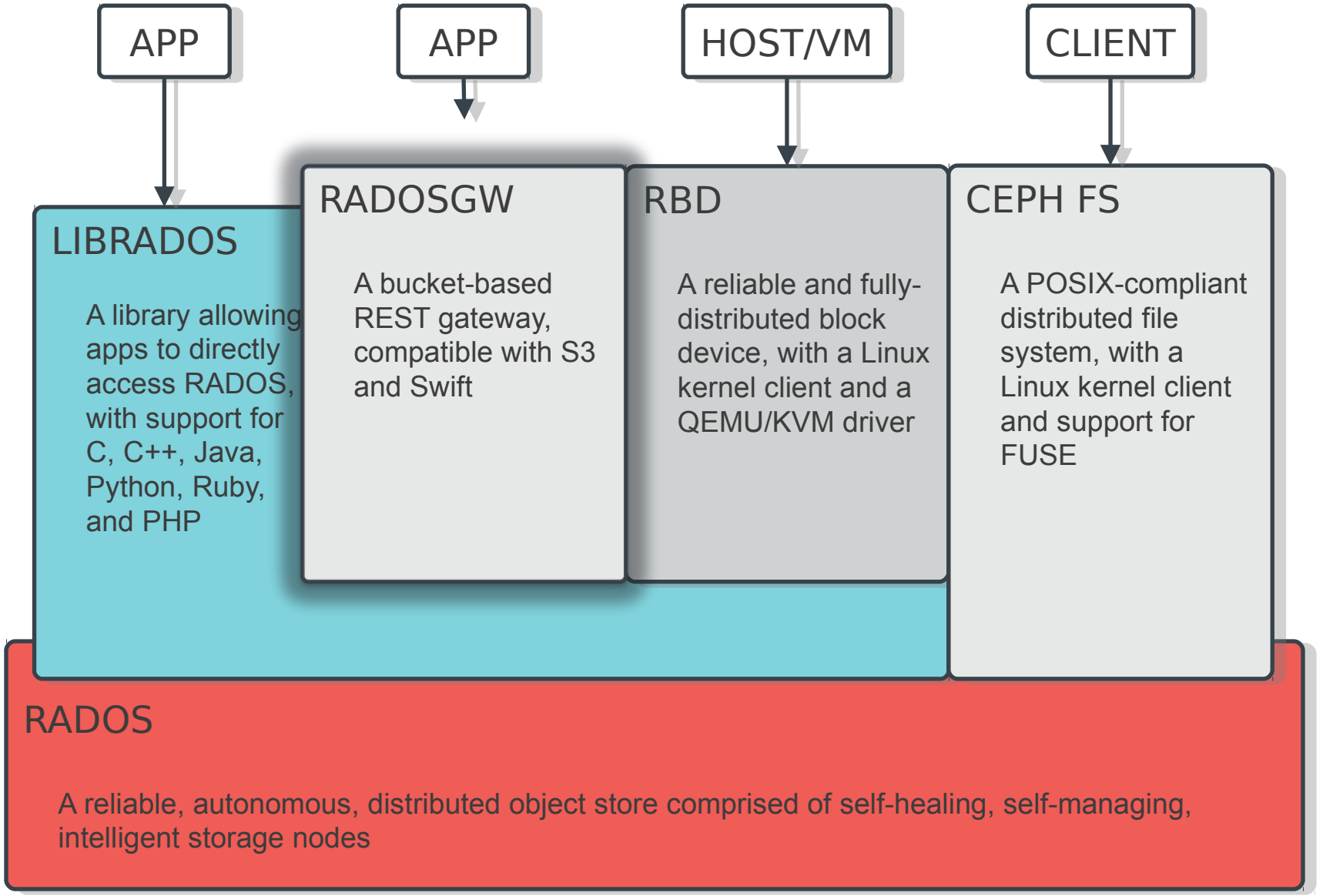


- great for single DC deployments
- great for nearby DCs with low latency
  - e.g., European telcos
- long distances → high write latency
  - untenable for many workloads, applications

ceph

disaster recovery

- temporary data center failure
  - power outages, hurricanes, cut fiber
- larger natural catastrophies
  - earthquakes
  - data is precious; stale data is better than no data (but it must be coherent)
- cluster-wide failures
- regulatory requirements
  - e.g., > N km away and < M seconds old


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

ceph

radosgw – RESTful object storage

- weaker consistency is okay
  - image archives
  - web content
- AP (instead of CP) model
  - eventually consistency often tenable
  - merge updates from fail-over slave back to master on recovery
- simpler to implement *above* rados layer
  - async mirroring between rados clusters/pools
- RESTful APIs means → simple agents

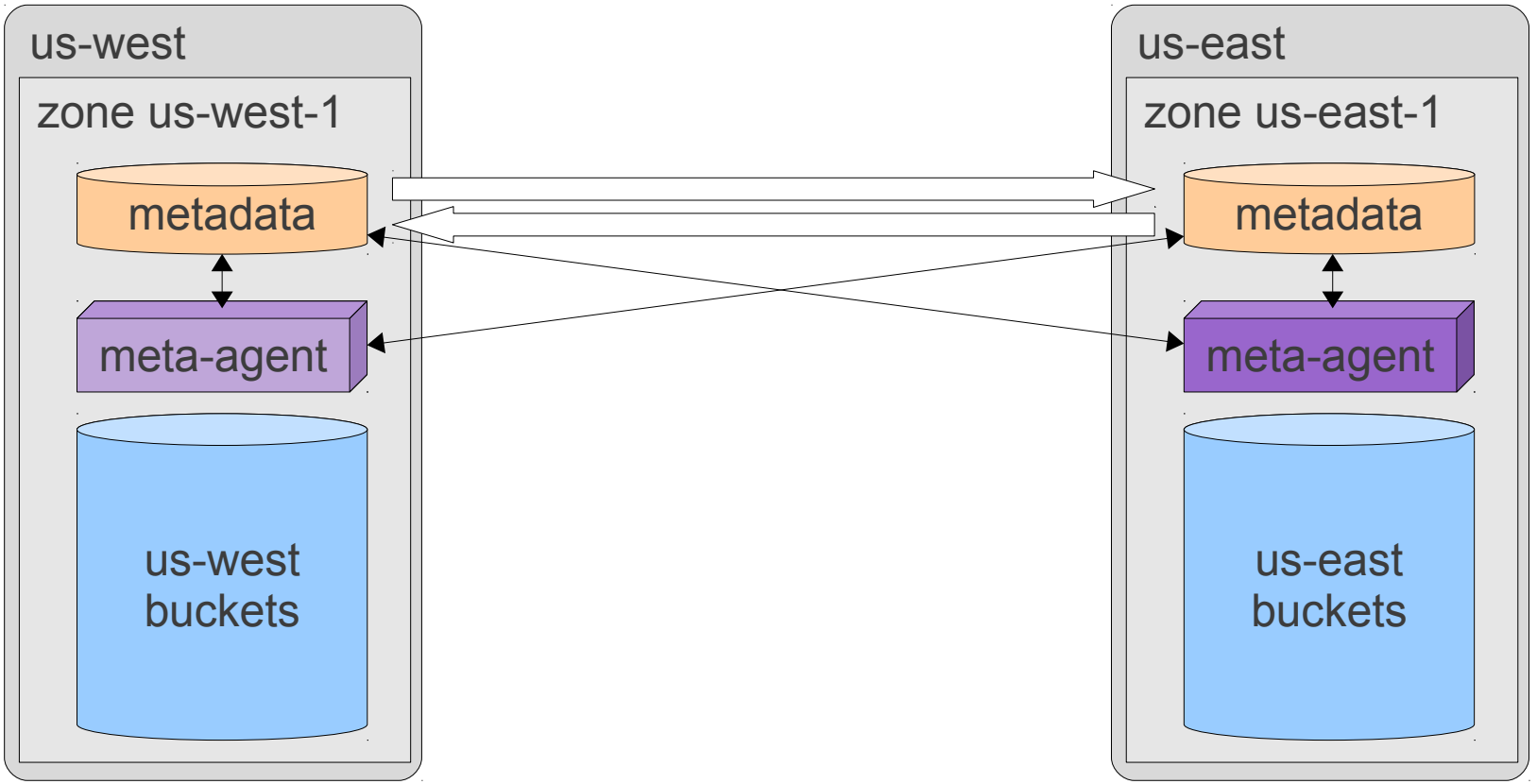
regions and zones

- zone
  - user + bucket metadata rados pool(s)
  - bucket content pool(s)
  - set of radosgw daemons serving that content
    - how any rgw deployment looked up until now
- region
  - a logical geography
    - east coast, west coast, iceland, singapore, …
    - data center?

region map

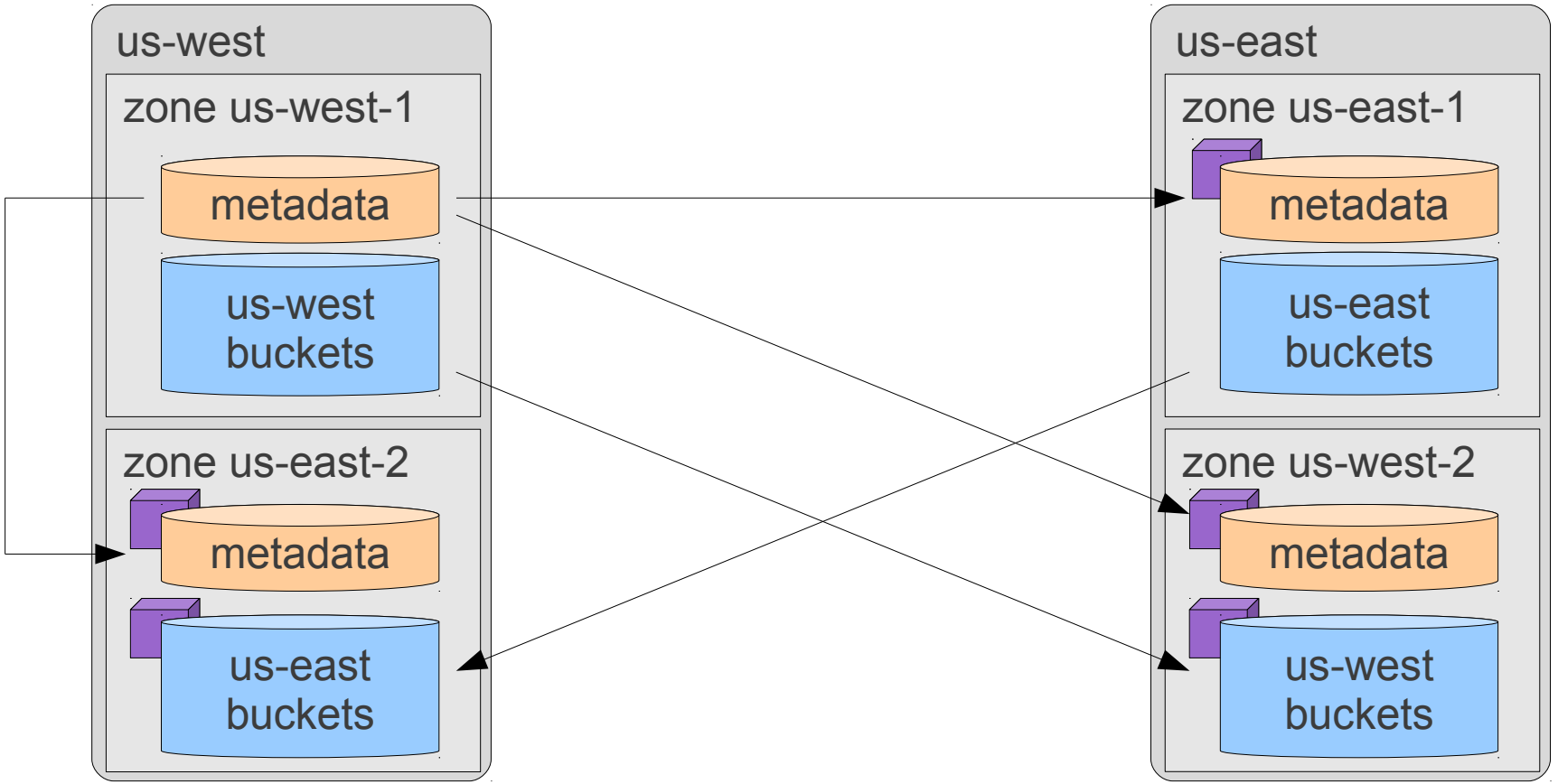
- admin-controlled metadata about a region
  - which zone is service content for this region?
  - which zones are replicating this region?
  - how should reads and writes be directed?
    - inform dynamic dns behavior, http redirects
- master + slave(s) for the user and bucket metadata
  - one region (zone) is master for a federated cluster
  - other zones sync metadata from the master

federated multi-region example



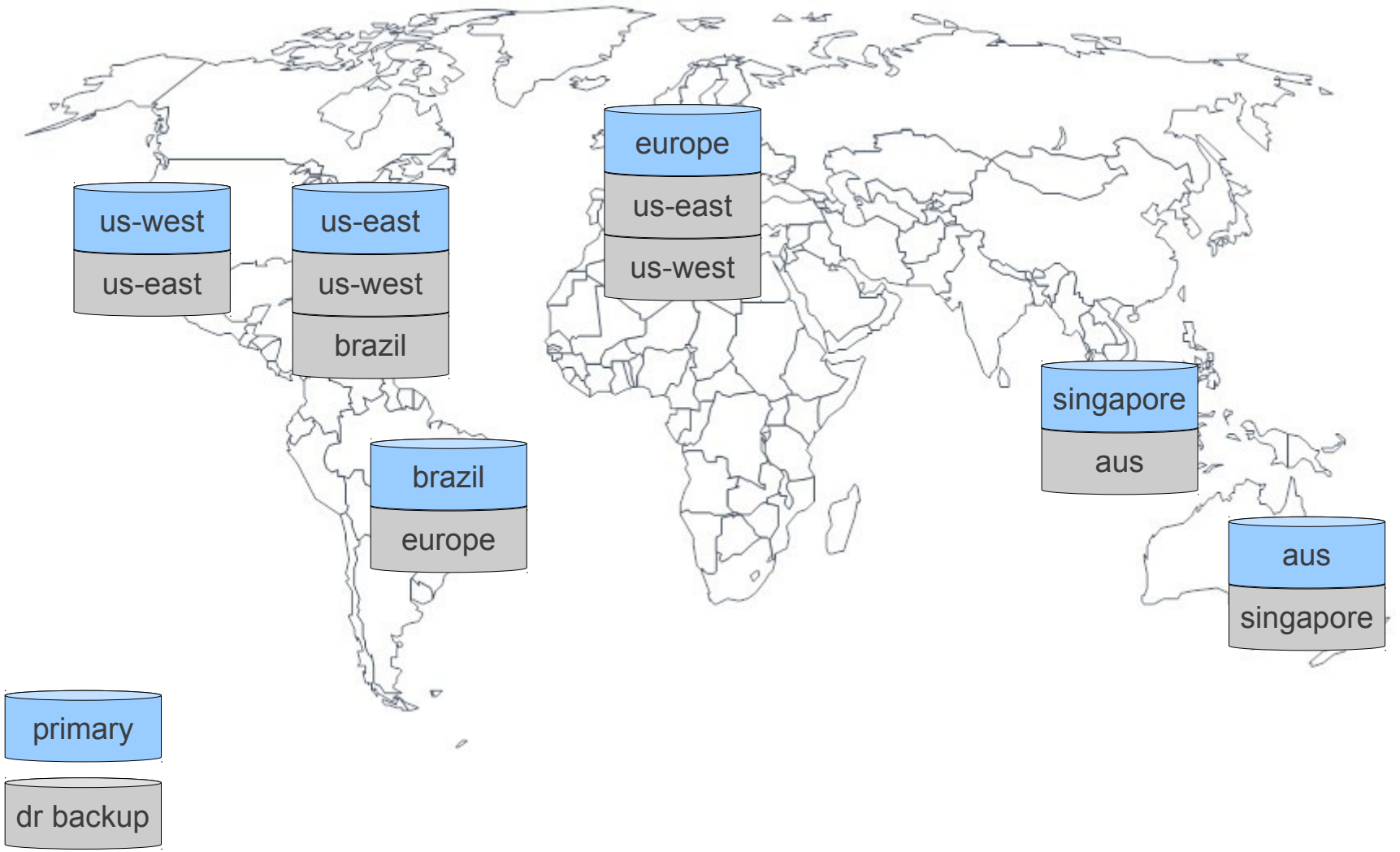
ceph

multi-region + DR example



ceph

a more complex example



ceph

versions, logs, and agents

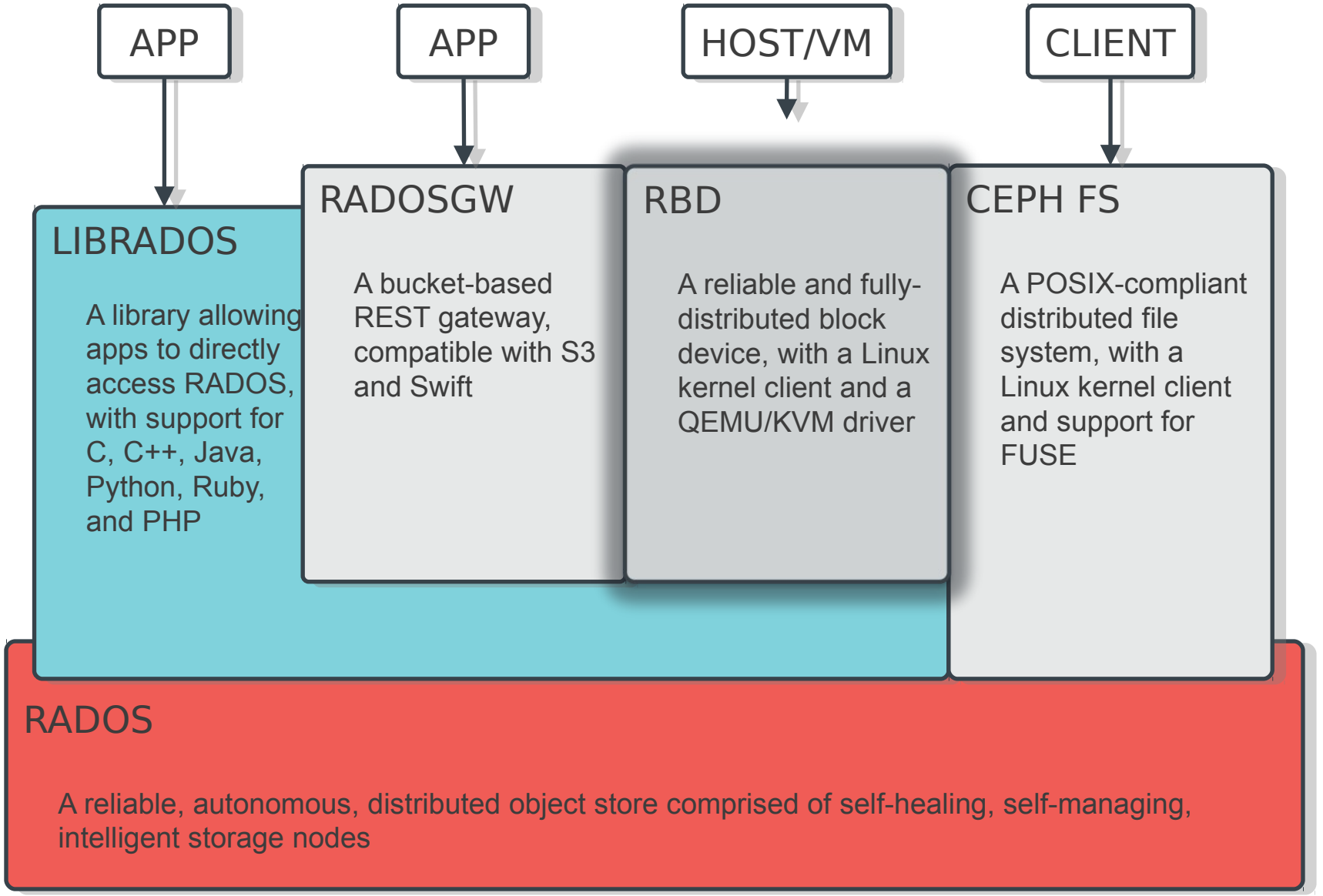
- version values for all metadata objects
- log updates to sharded log objects
- simple replication agent
  - lock shard for given slave
  - sync log to slave zone
  - sync logged updates; update markers in replica log

bucket data

- include log in existing bucket index objects
  - already part of the PUT update sequence
- shared bucket update map
  - scalable, sharded record of which buckets have recent updates
- agents
  - lock shard for given slave
  - process buckets
    - marker on bucket index, bucket update map shard to indicate progress
- new copy operation
  - copy objects between clusters (zones)
  - agents not in data path
  - preserve object metadata (timestamps, ACLs, etc.)

radosgw DR status

- multi-site federation – done
  - metadata agent done
  - dumpling (released August)
- async replication – in progress
  - data agent in progress
  - emperor (November)
- looser consistency appropriate for most radosgw workloads
- multi-site federation is a key feature independent of the DR capability
  - mirrors S3 product features


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

ceph

rbd – rados block device

- file systems require consistent behavior from underlying disks
  - e.g., some parts of platter old + some parts new

    → depend on fsck to recover data...
  - we need a point-in-time view of the disk
- already have efficient snapshot capability
  - read-only point-in-time views
  - managed per-image
  - efficient; stored in-place

incremental snap backup/diff

- identify changed extents
  - between two snaps
  - between snap and head
- simple streamable file format
  - notes image uuid, start/end snap names, changed image extents
  - can be archived or applied to a copy of an image
- somewhat efficient
  - iterates over image objects, queries snap overlap
  - no data read, but still O(n) scan of metadata

```
rbd export-diff mypool/myimage@endsnap --from-snap startsnap - | \
    ssh -C othercluster rbd import-diff - mypoolbackup/myimage
```

ceph

rbd DR status

- works for pure 'disaster recovery'
  - it's a backup
- strongly consistent
  - no fsck needed for journaling fs's
  - sufficient for many user
- integrated into OpenStack backup service
- could benefit from simple agent
  - for all images in a pool/cluster,
    - take periodic snaps
    - sync to remote cluster

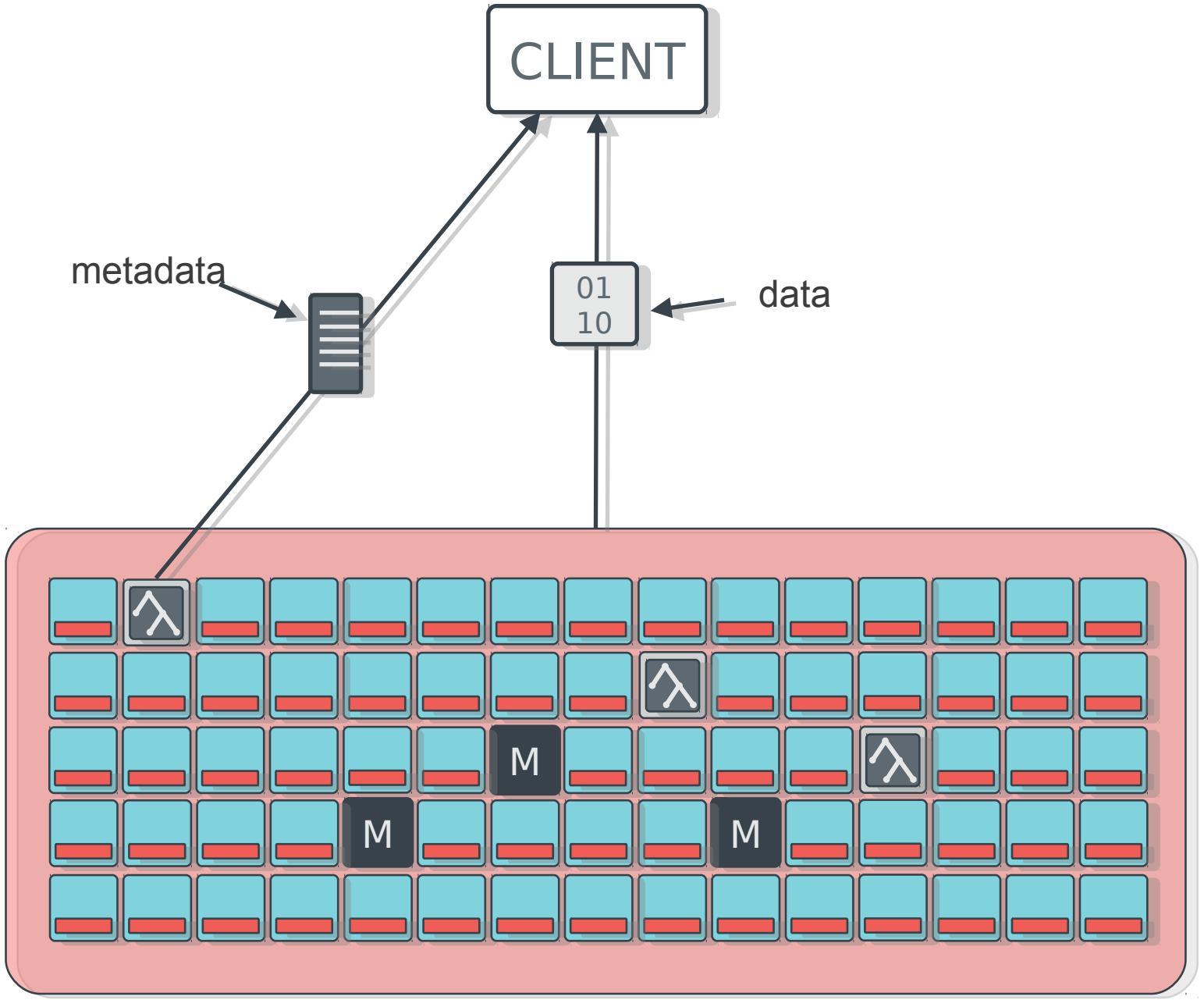
ceph

cephfs

- posix, cache-coherent distributed fs
- complicated update semantics
  - files, directories
  - rename, link, unlink
  - fsync ordering

    → hard to get right even for local file systems
- decoupled data and metadata paths
- distributed mds

ceph



ceph

cephfs DR via rados DR

- rely on a rados-level DR solution?
  - mds cluster, clients built entirely on rados pools
  - if rados provides point-in-time mirror of fs pools...
  - ...fail-over to mirror looks like a cluster restart
- rados DR also captures rbd and librados users

- provide rados-level async replication of pool(s)
  - consider awareness at fs layer to ease coordination, orchestration

rados async replication

- async replication / disaster recovery
  - 1 master, 0 or more slaves
  - single pool or a set of pools (e.g., for cephfs)
  - time-delayed
- on failure, slaves can become new master
  - minus some recent changes
- slaves trail master by bounded amount of time
  - as close as possible... ~minute
  - deliberately delayed by some amount of time (e.g., 1 hour)

slave consistency

- rados provides (and applications expect) strong consistency
  - example
    - write a; write b; write a'; write b'
  - ok
    - a
    - a + b
    - a' + b
    - a' + b'
  - not ok
    - b (but no a)
    - a' (but no b)
    - a + b'

  → relative order matters

  → we need point-in-time consistency

conventional strategy: snapshots

- master
  - periodic snapshots (e.g., every few minutes)
- slave
  - initialize by mirroring an initial snapshot
  - while true
    - mirror delta to next snapshot
    - record snapshot locally
  - on failure, roll back to last complete snapshot
- advantages
  - conceptually simple; leverages existing snapshot infrastructure when present
- limitations
  - slave rolls forward in chunky, step-wise fashion; on average always at least period/2 behind
  - requires efficient, frequent snapshots
  - requires efficient snapshot delta

conventional strategy: update log

- master
  - record ordered log of all updates (and data)
  - trim when all slaves have consumed events
- slave
  - stream and apply update log
- advantages
  - low (or tunable) slave latency/delay
- limitations
  - log may include redundant information (e.g., rewrites)
  - logging overhead
  - requires full ordering of events – how do you scale out?

rados challenges

- scale-out
  - 1 → 1 update stream conceptually simple
  - 1000 → 1000 update streams are harder
- complex update semantics
  - mutable objects
  - non-trivial mutations
    - byte extent overwrites; key/value, attr updates; clone
- must order operations in parallel streams
  - sharding decouples updates for scaling...

ordering

- bad news
  - no such thing as absolute ordering of events in a distributed system
- good news
  - a partial, relative ordering does – Lamport '78
  - causal ordering wrt messages passed within the distributed system

- could get absolute snap ordering by quiescing io
  - pause all io to create an ordering 'barrier'

    → but it's to expensive in a distributed system

    → that's why ceph snapshots also don't work this way

lamport clocks

- a → b : a “happens-before” b
- C(a) : timestamp of event a

- lamport clock: simple counter for each process
  - increment clock before any event (clock sample)
  - include clock value when sending any message
  - on message receipt,

    clock = MAX(message stamp, local clock) + 1

ceph

what does the causal ordering mean?

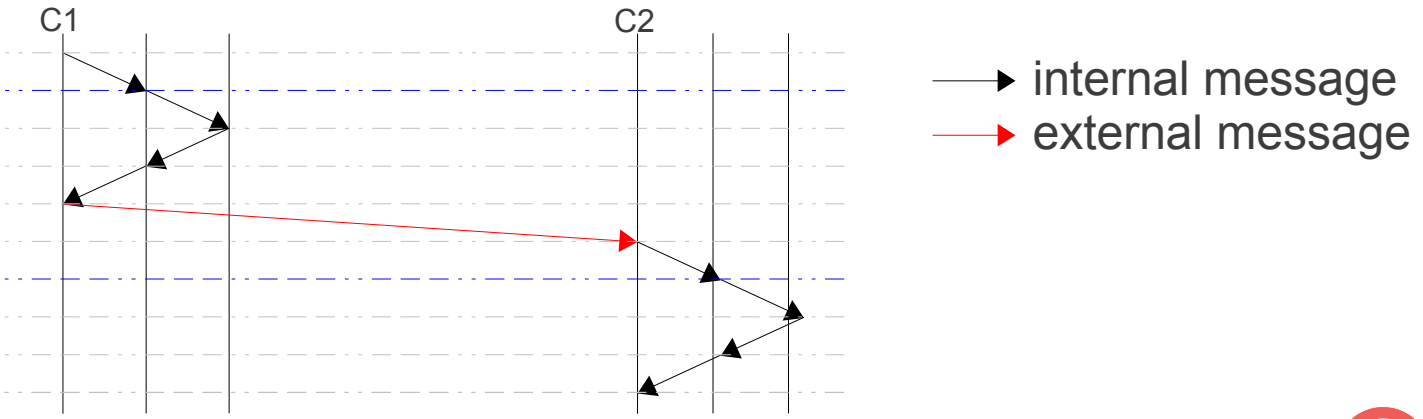
- a → b implies C(a) < C(b)
  - for events in the same process or processes that communicate, any causation is reflected by relative timestamps
- C(a) < C(b) implies
  - a happened-before b or we aren't sure
  - b did not happen-before a
- ambiguity is ok
  - vector clocks tell us more, but nothing useful to us here
- ordering does not reflect *external* communication
  - X does A; tells Y via external channel; Y does A
  - we may not realize that A → B

do external messages matter?

- for an fs in rbd, cephfs client
  - no: single client initiates all IO → ordered
- for clients coordinating via ceph
  - e.g., librados watch/notify; fs locks; stored data
  - no: communication via ceph enforces partial order
- for systems built on multiple ceph services
  - if they communicate externally, maybe
  - individual services individually consistent
    - but possibly not with each other

how (im)probable is that?

- huge number of messages exchanged internally by ceph
  - heartbeats
  - replication (4 hops and 6-8 message per write)
- hard for 1 external hop + 4 internal hops to win a race



ceph

we can do better: hybrid clock

- use lamport clock rules for causal ordering
- use real clock to advance at realistic rate
  - instead of +1, add delta since last clock sample
- in other words: use high-precision timestamp
  - adjust our clock forward when we see a msg from the future
  - converge on fastest clock in cluster
- strong approximation of “real” time even when
  - messages delayed or backed up
  - two nodes many hops away in cluster mesh

rados dr: a checkpoint-based strategy?

- already have object-granularity snap infrastructure
  - but it is fully exposed via rados API
  - application-defined snapshots, independent timelines
  - messy to layer internal snap/checkpoint timeline too
- rollback on remote cluster
  - expensive without low-level support from (btr)fs
  - difficult at pool/PG granularity
- step-wise approach introduces latency
  - difficult to push slave delay to low levels (i.e. seconds)

rados DR: a streaming strategy

- master
  - write-aside update log for each PG when mirroring is enabled
  - trim when all slave(s) have pulled it down
  - requires extra IO
- slaves
  - each PG pulls update log in parallel
  - cluster maintains lower-bound on progress
  - PGs apply / roll forward to that bound

can we do better?

- lots of extra log IO (on master and slave)
- can we re-use existing journals? probably not
  - bounded circular buffer
  - no per-pg granularity: includes all local OSD IO
  - too low-level
    - beneath local replication, migration, recovery
    - this copy of the PG, not logical updates
- btrfs clone can mitigate copies
  - master
    - WIP will share blocks on osd journal and object writes
    - can extend to async log
  - slave
    - can clone from async log on apply

conclusions

- range of requirements for async replication/DR
  - depends on API, app requirements
- RESTful object storage
  - AP-like semantics more suitable
  - API-specific implementation complements multi-site
  - available Real Soon Now
- rbd
  - strong consistency
  - inc backup/diff solution offers coarse solution today
- cephfs – would be complex
- rados
  - non-trivial, but we have a plan, and can't wait to do it

ceph

a bit (more) about ceph

- hardware agnostic
- 100% free and open source
  - LGPL; distributed developer community
- future depends on robust free/oss storage tech
- open solutions better
  - for world, businesses, users devs
  - everyone but entrenched vendors
- ceph is a strong platform for building next-gen solutions

ceph

a question...

- why do developers choose to build closed software?
  - don't care about free/non-free or open/closed
  - prefer closed software, ownership
  - it was the only good option at the time
  - more money: higher salaries or equity
  - smaller, tighter, focused dev teams
  - more exciting technologies
  - ?

thanks

sage weil

sage@inktank.com

@liewegas

http://github.com/ceph

http://ceph.com/

inktank

ceph