ceph

managing a distributed storage system at scale

sage weil – inktank
LISA – 12.12.12

outline

- the scale-out world
- what is it, what it's for, how it works
- how you can use it
  - librados
  - radosgw
  - RBD, the ceph block device
  - distributed file system
- how do you deploy and manage it
- roadmap
- why we do this, who we are

why should you care about another storage system?

the old reality

- direct-attached storage
  - raw disks
  - RAID controller
- network-attached storage
  - a few large arrays
  - NFS, CIFS, maybe iSCSI
- SAN
  - expensive enterprise array
- scalable capacity and performance, to a point
- poor management scalability

new user demands

- “cloud”
  - many disk volumes, with automated provisioning
  - larger data sets
    - RESTful object storage (e.g., S3, Swift)
    - NoSQL distributed key/value stores (Cassandra, Riak)
- “big data”
  - distributed processing
  - “unstructured data”
- ...and the more traditional workloads

requirements

- diverse use-cases
  - object storage
  - block devices (for VMs) with snapshots, cloning
  - shared file system with POSIX, coherent caches
  - structured data... files, block devices, or objects?
- scale
  - terabytes, petabytes, exabytes
  - heterogeneous hardware
  - reliability and fault tolerance

cost

- near-linear function of size or performance
- incremental expansion
  - no fork-lift upgrades
- no vendor lock-in
  - choice of hardware
  - choice of software
- open

time

- ease of administration
- no manual data migration, load balancing
- painless scaling
  - expansion **and** contraction
  - seamless migration

- devops friendly

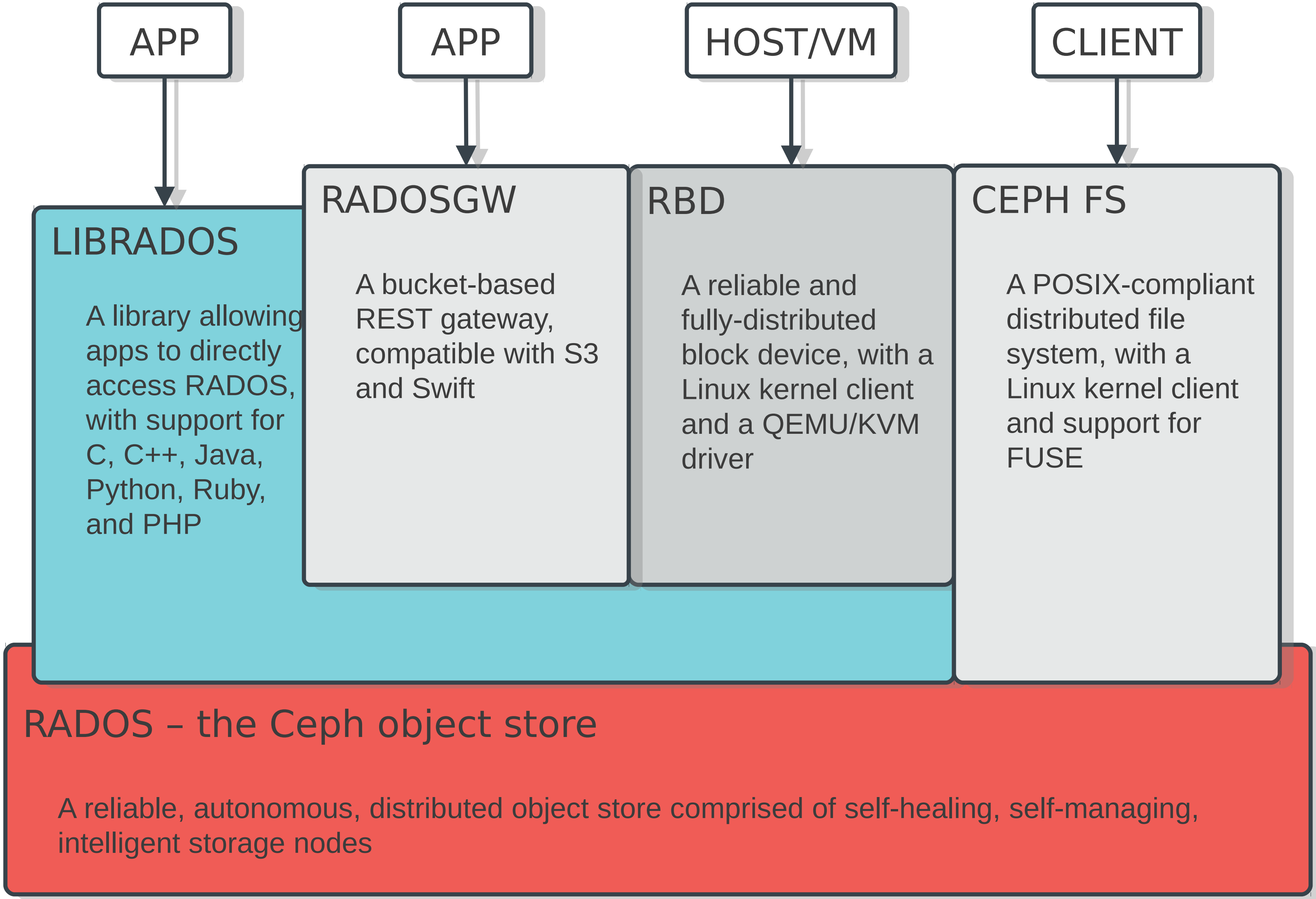
being a good devops citizen

- seamless integration with infrastructure tools
  - Chef, Juju, Puppet, Crowbar, or home-grown
  - collectd, statsd, graphite, nagios
  - OpenStack, CloudStack
- simple processes for
  - expanding or contracting the storage cluster
  - responding to failure scenarios: initial healing, or mop-up
- evolution of “Unix philosophy”
  - everything is just daemon
  - CLI, REST APIs, JSON
  - solve one problem, and solve it well

what is ceph?

unified storage system

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


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

distributed storage system

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

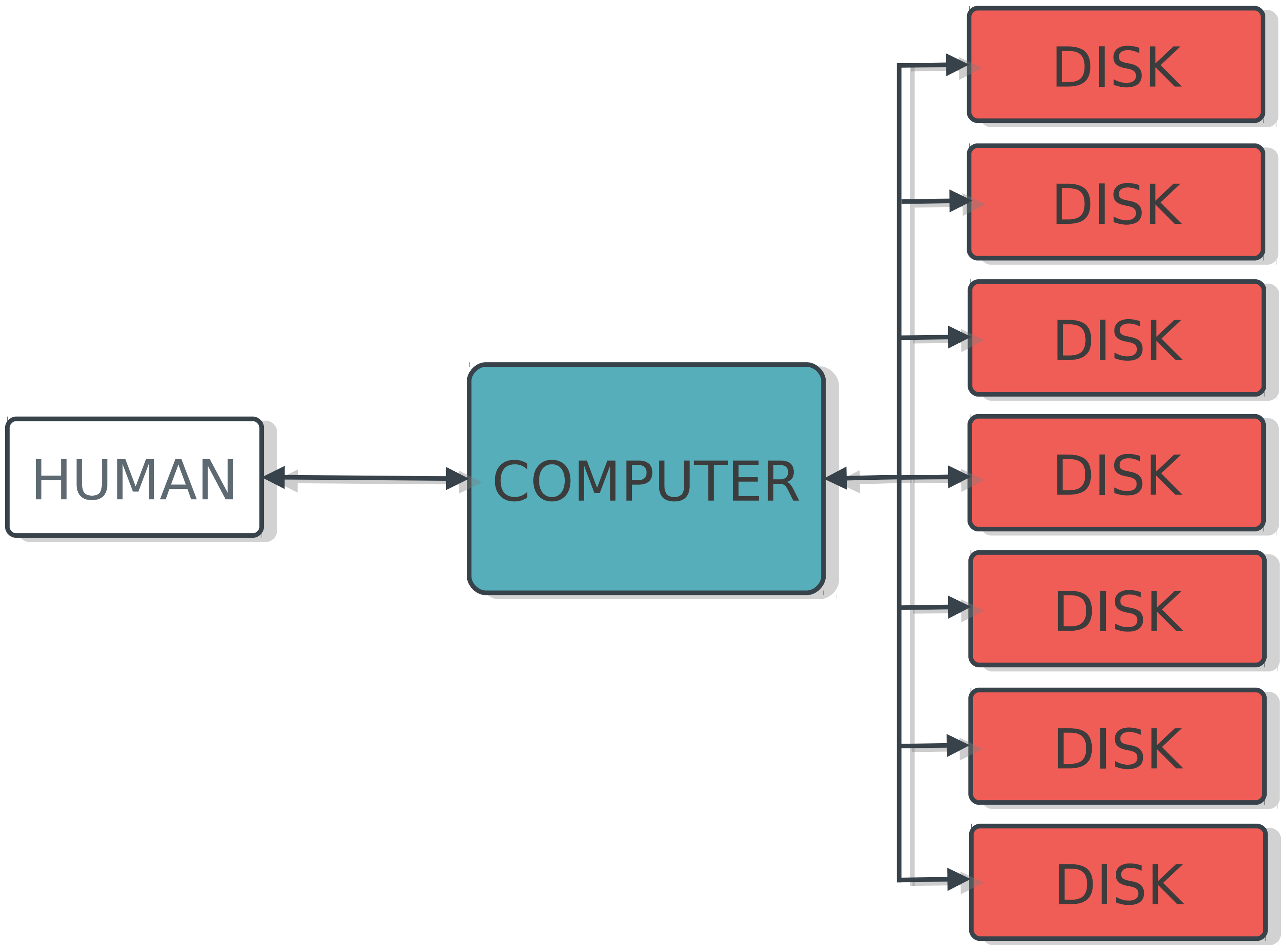
ceph object model

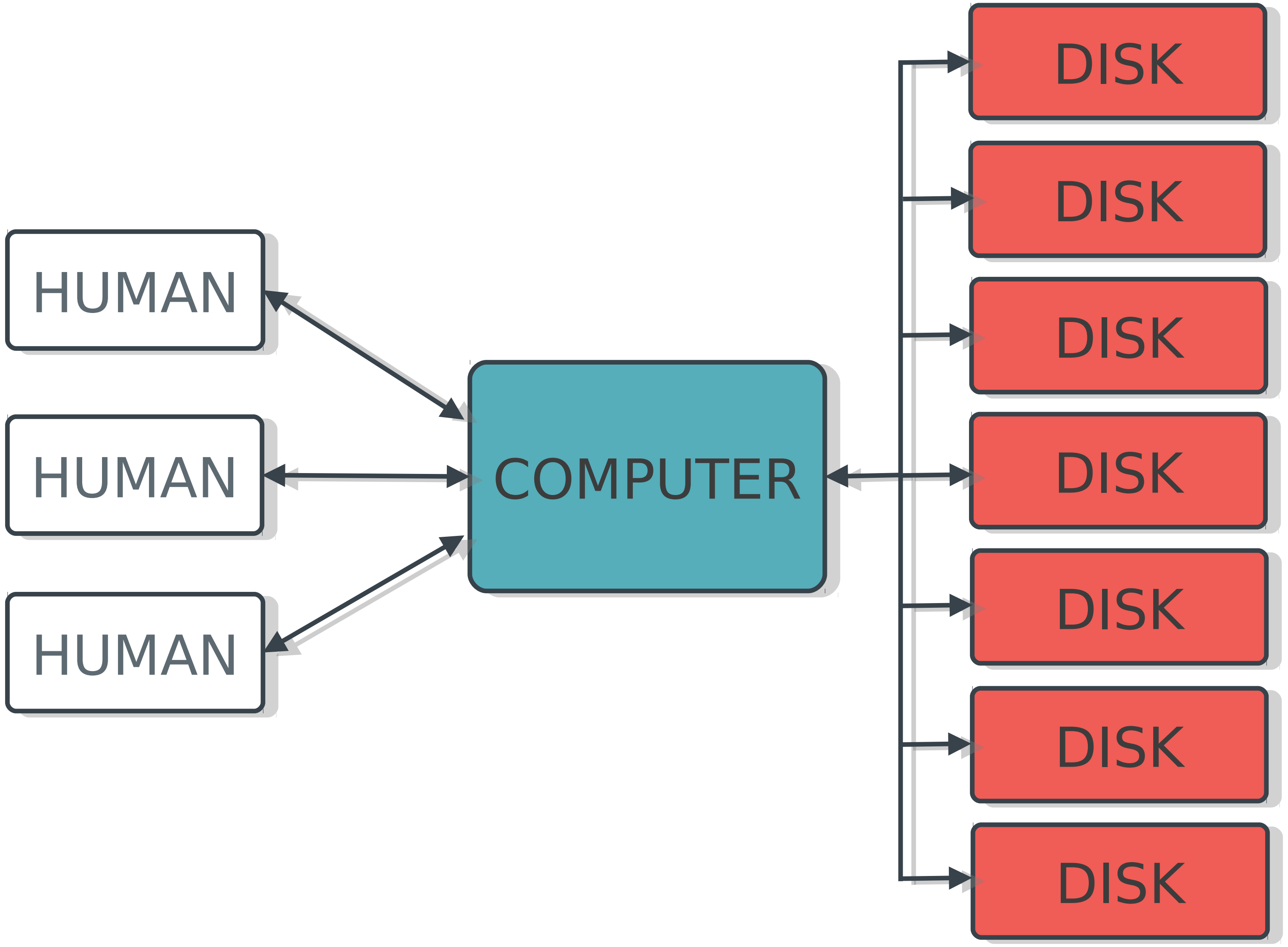
- pools
  - 1s to 100s
  - independent object namespaces or collections
  - replication level, placement policy
- objects
  - bazillions
  - blob of data (bytes to gigabytes)
  - attributes (e.g., “version=12”; bytes to kilobytes)
  - key/value bundle (bytes to gigabytes)

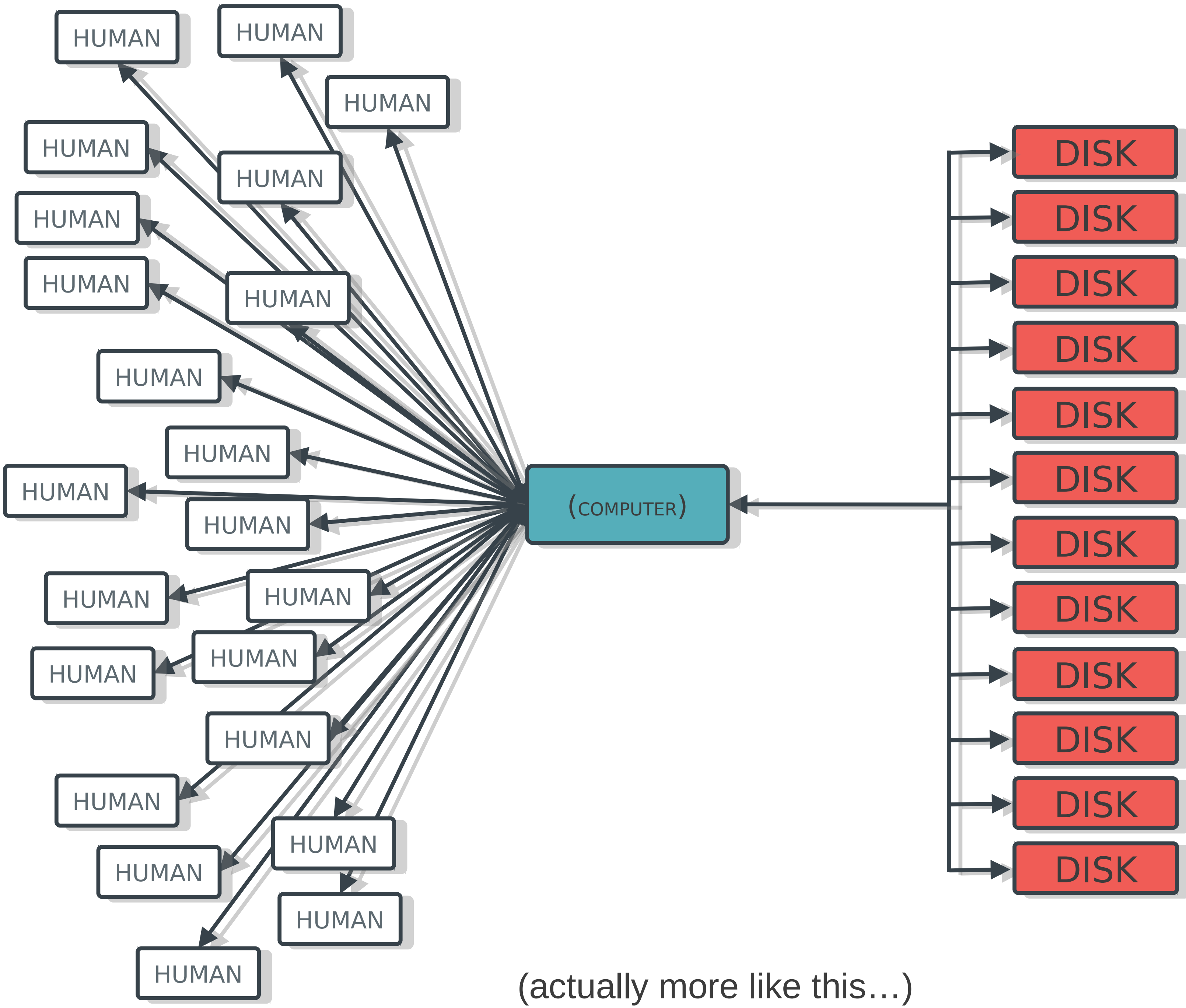
why start with objects?

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

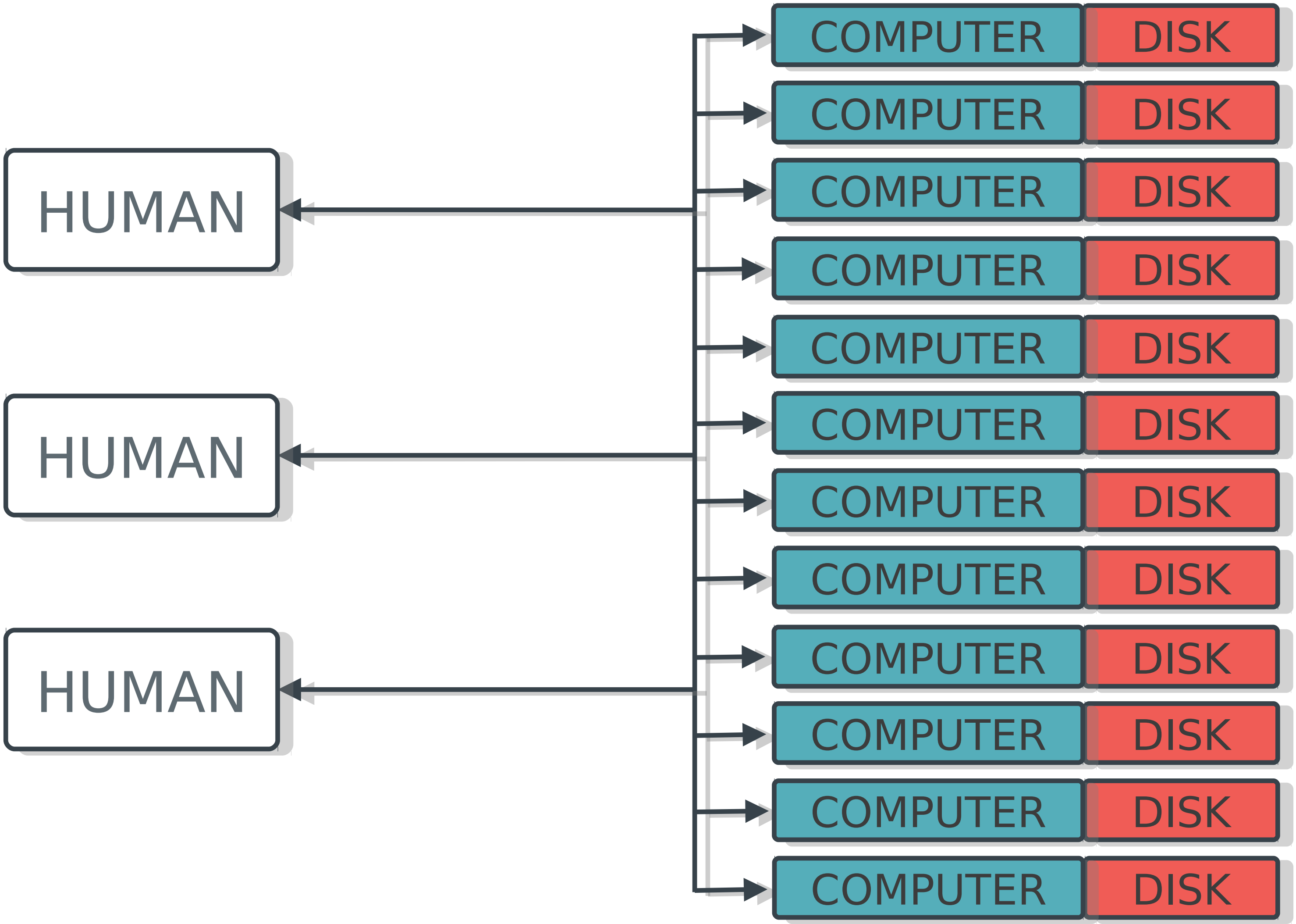
How do you design
a storage system that scales?


HUMAN
COMPUTER
DISK
DISK
DISK
DISK
DISK
DISK
DISK

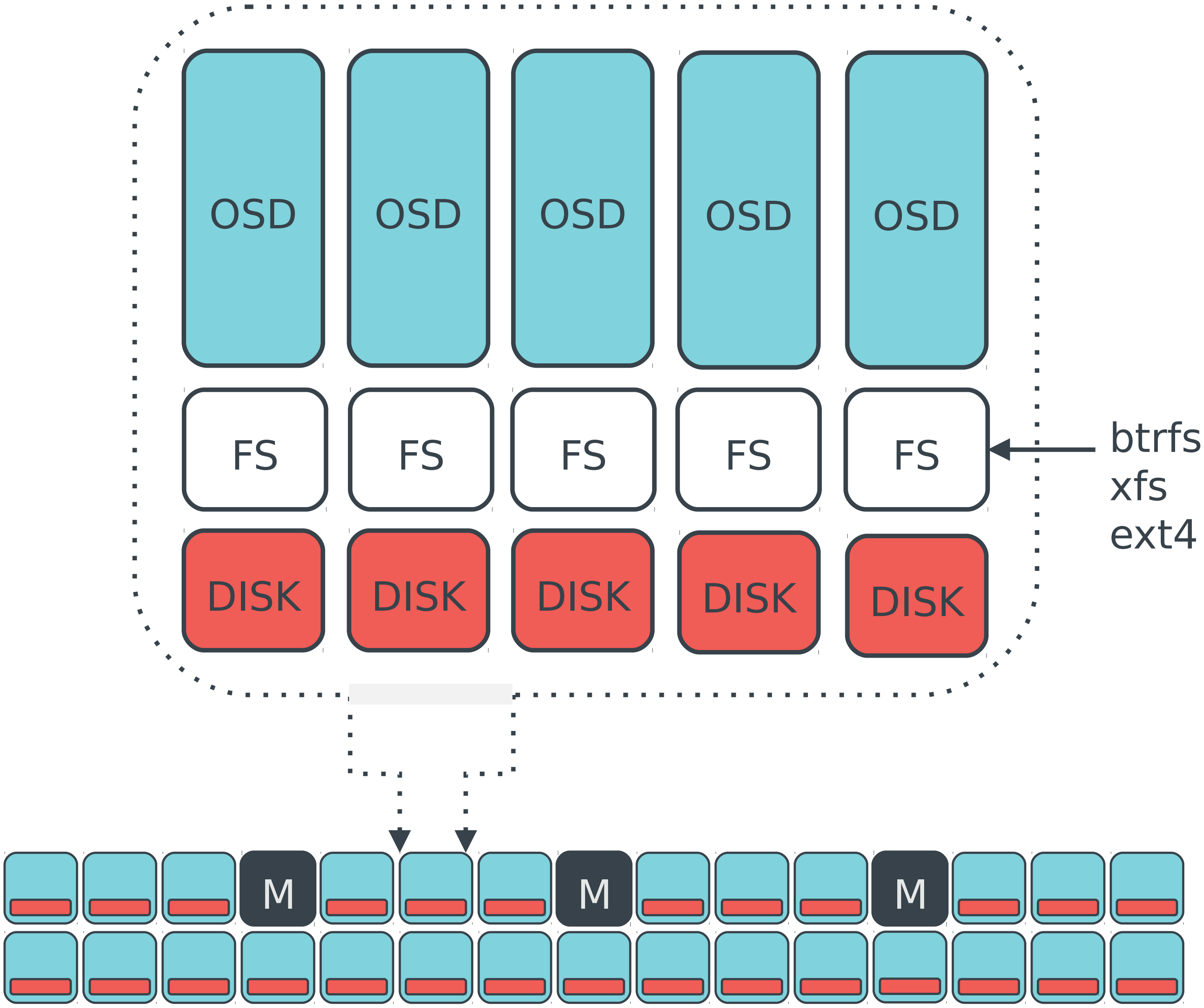

HUMAN
HUMAN
HUMAN
COMPUTER
DISK
DISK
DISK
DISK
DISK
DISK
DISK



(actually more like this…)

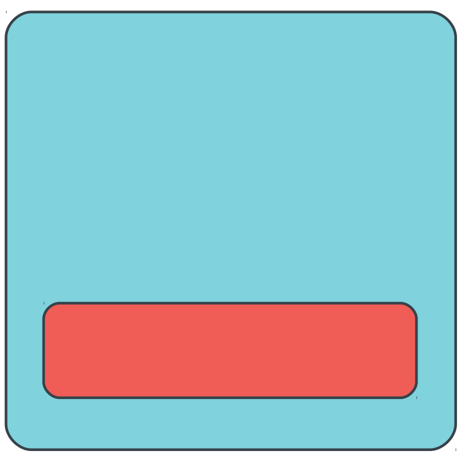

HUMAN
HUMAN
HUMAN
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK

ceph


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

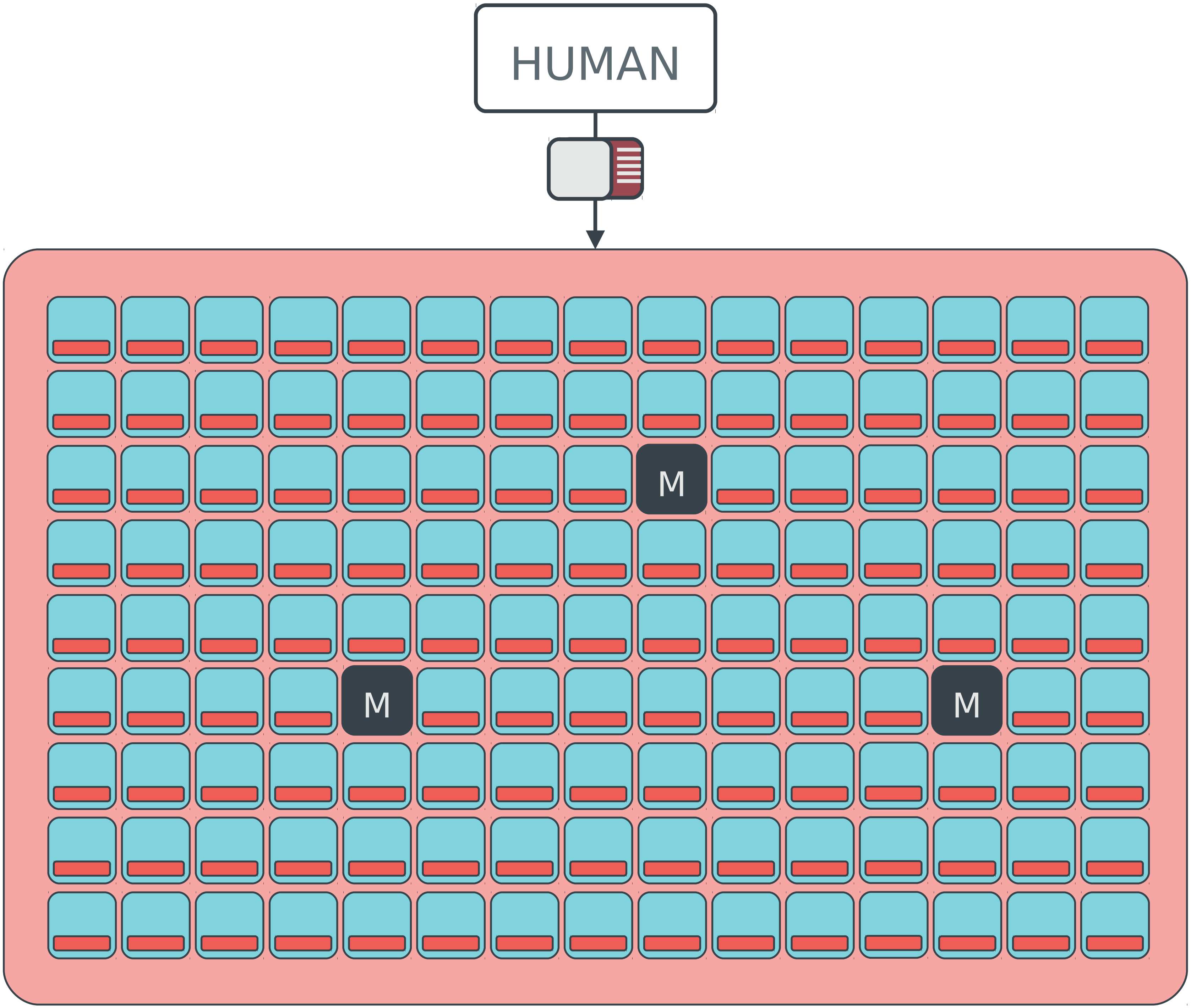
Monitors

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



Object Storage Daemons (OSDs)

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

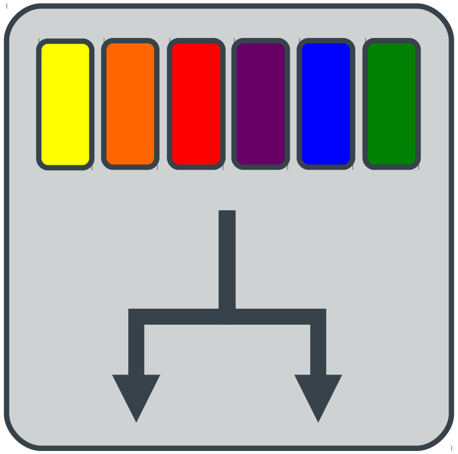


ceph

data distribution

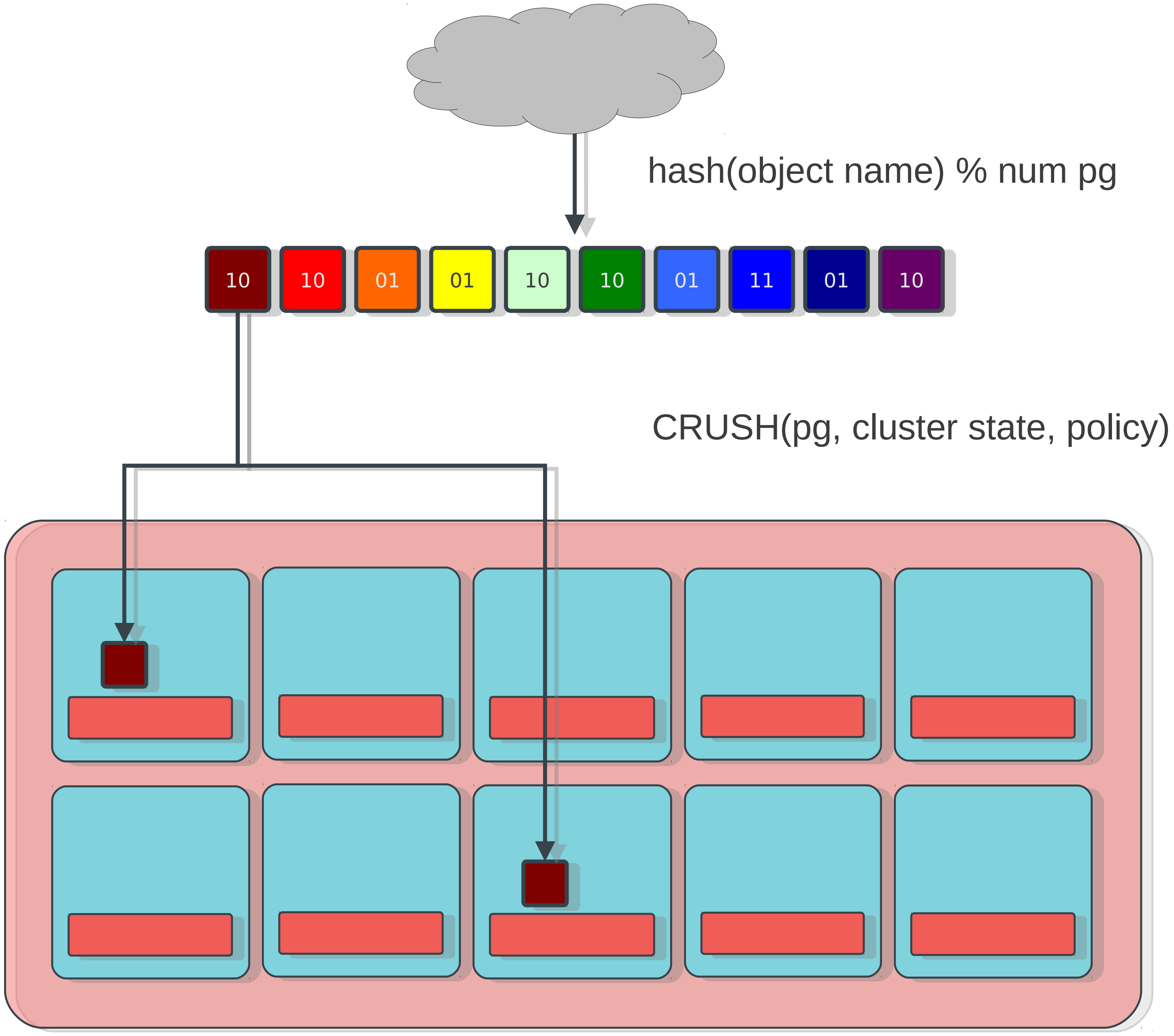
- all objects are replicated N times
- objects are automatically placed, balanced, migrated in a dynamic cluster
- must consider physical infrastructure
  - ceph-osds on hosts in racks in rows in data centers

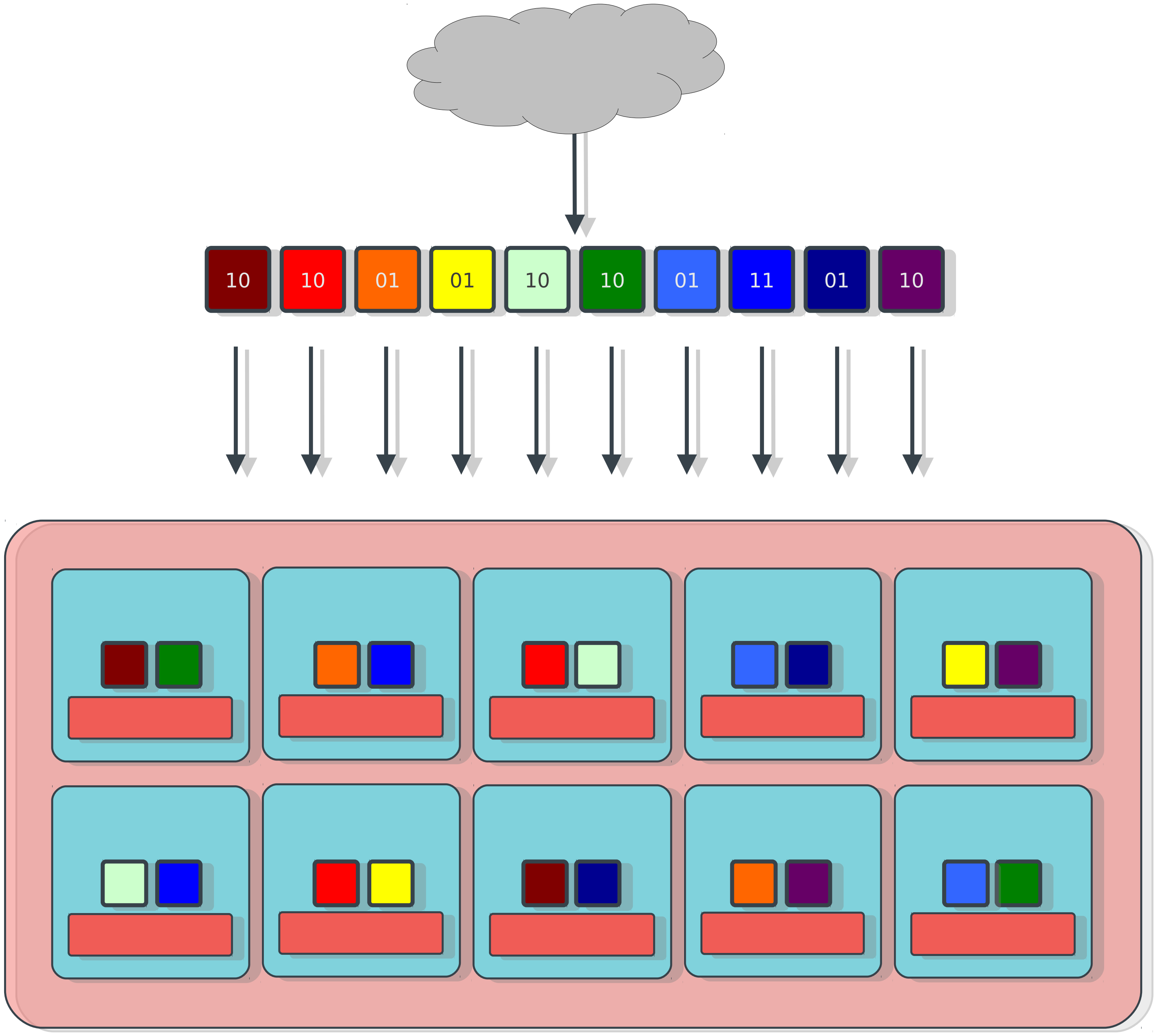
- three approaches
  - pick a spot; remember where you put it
  - pick a spot; write down where you put it
  - calculate where to put it, where to find it



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

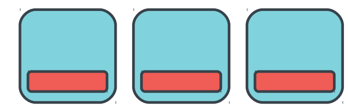

hash(object name) % num pg
10
10
01
01
10
10
01
11
01
10
CRUSH(pg, cluster state, policy)


10
10
01
01
10
10
01
11
01
10

RADOS

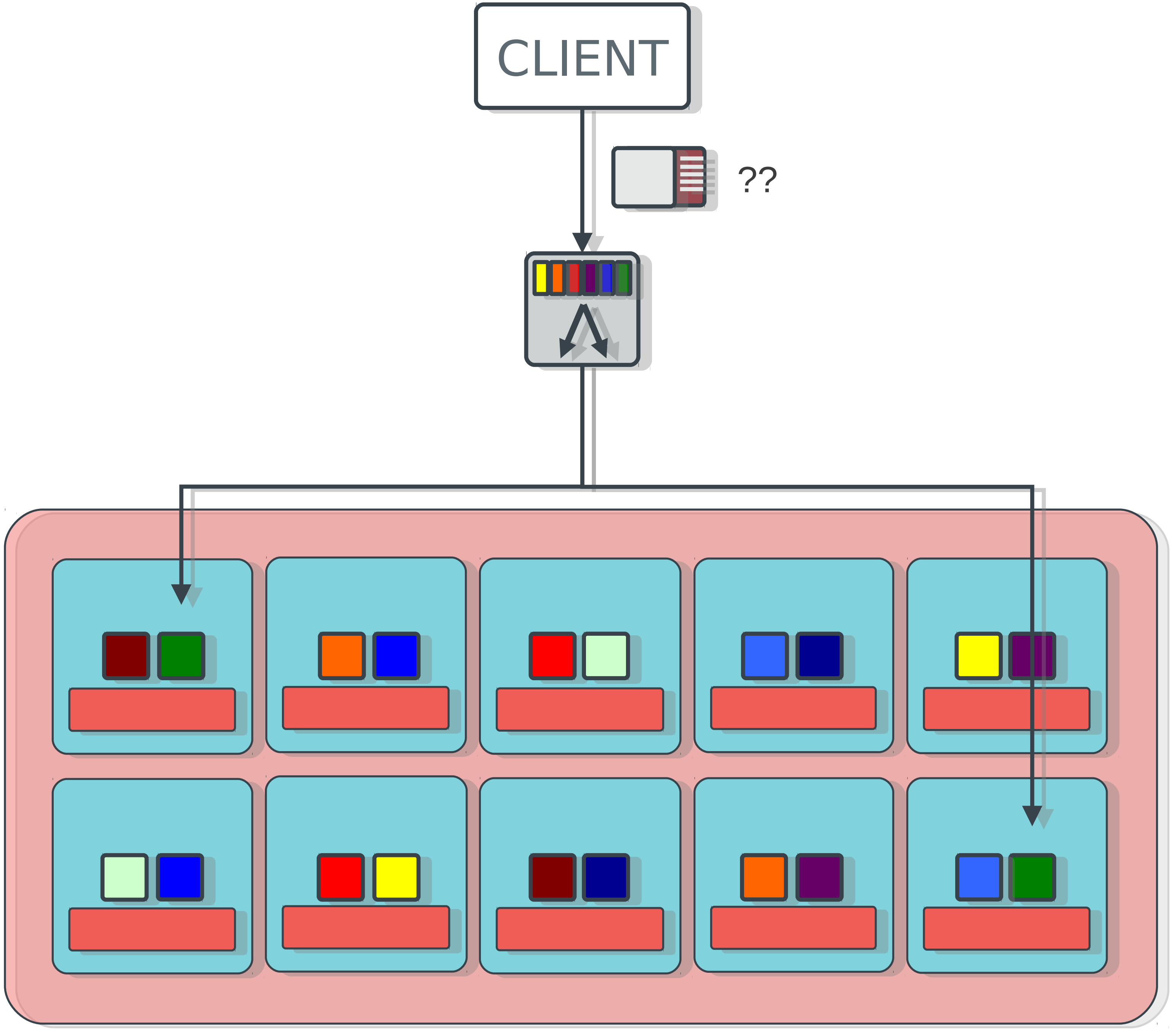
- monitors publish **osd map** that describes cluster state

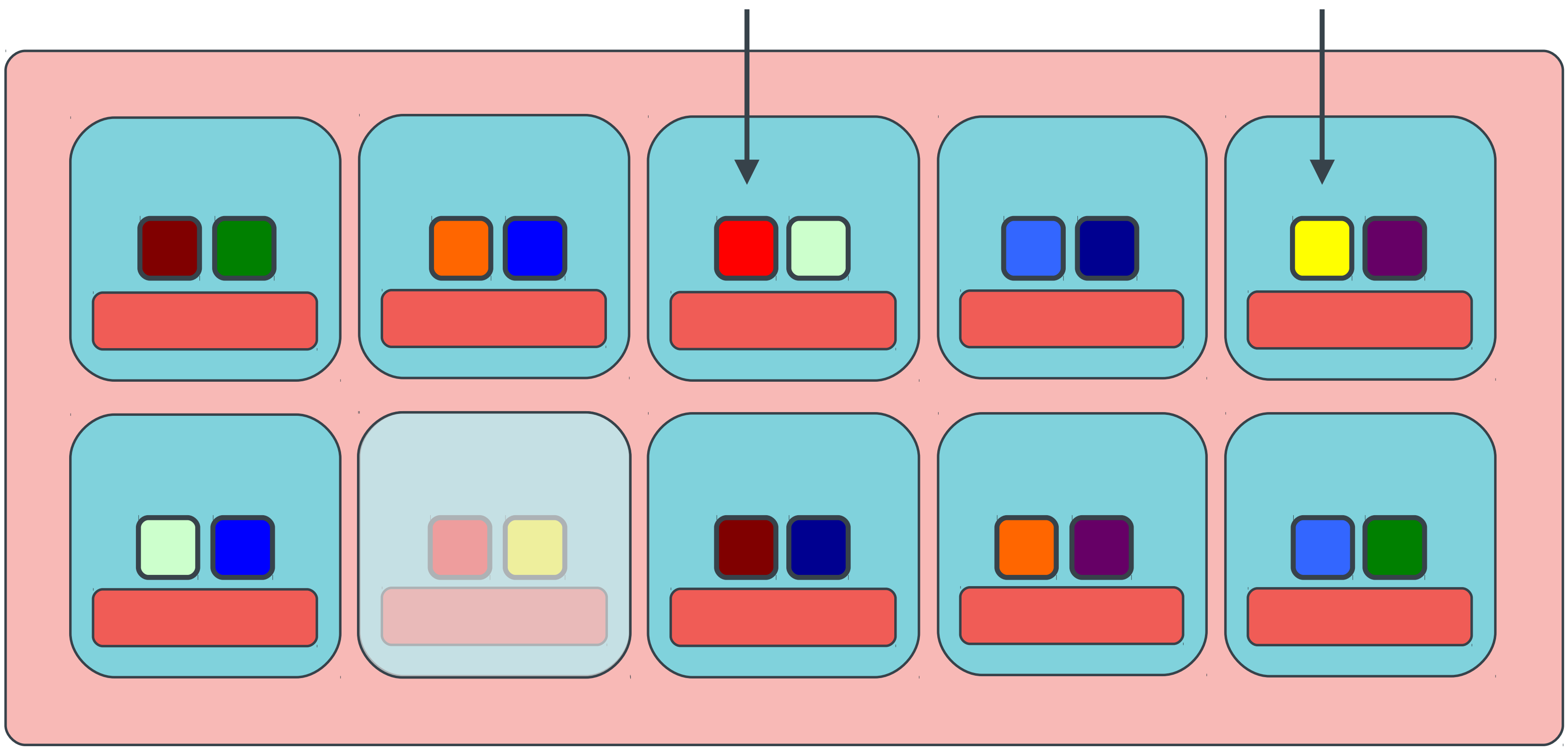
  - ceph-osd node status (up/down, weight, IP)
  - CRUSH function specifying desired data distribution
- object storage daemons (OSDs)

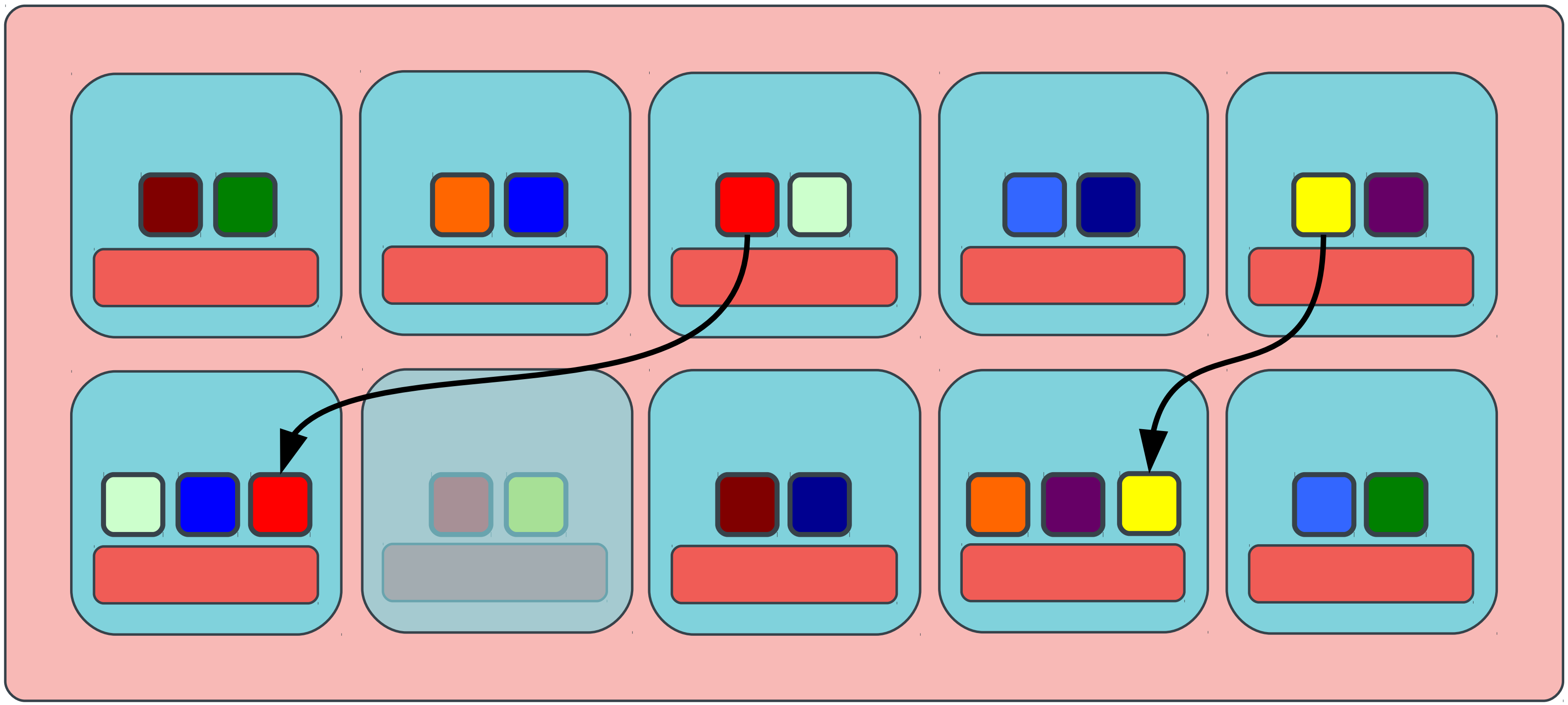


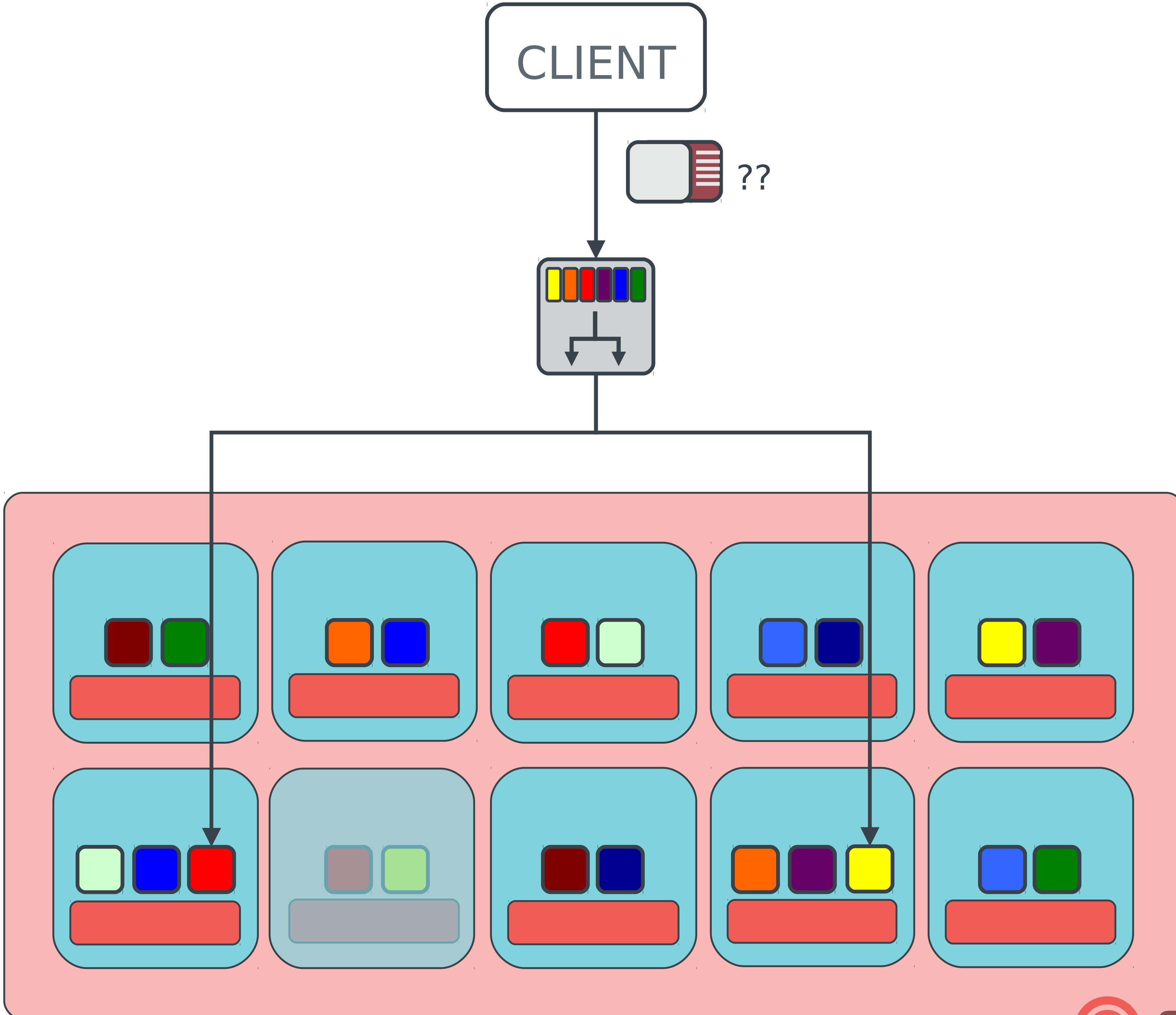
  - safely replicate and store object
  - migrate data as the cluster changes over time
  - coordinate based on shared view of reality
- decentralized, distributed approach allows
  - massive scales (10,000s of servers or more)
  - the illusion of a single copy with consistent behavior

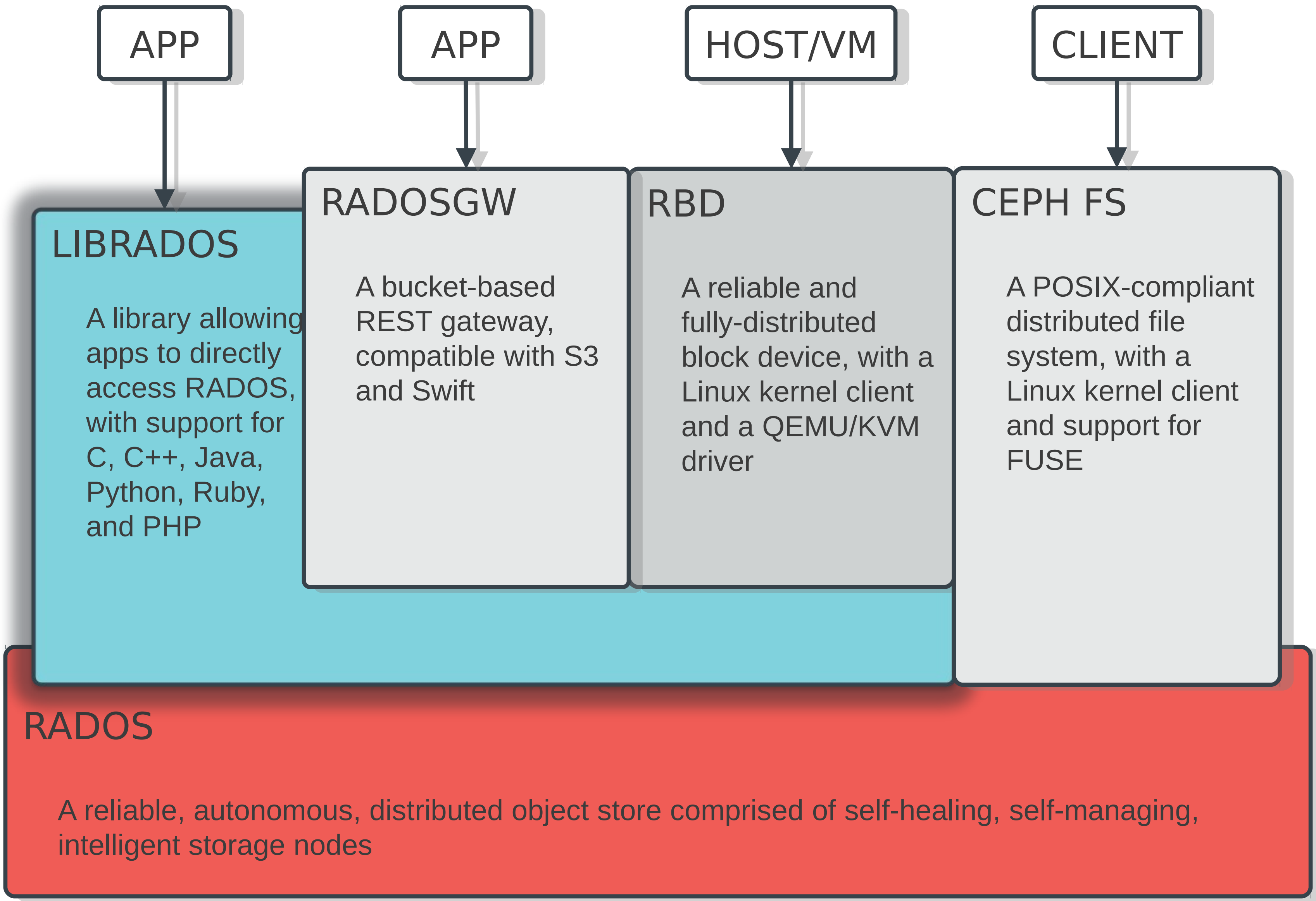
ceph


CLIENT
??

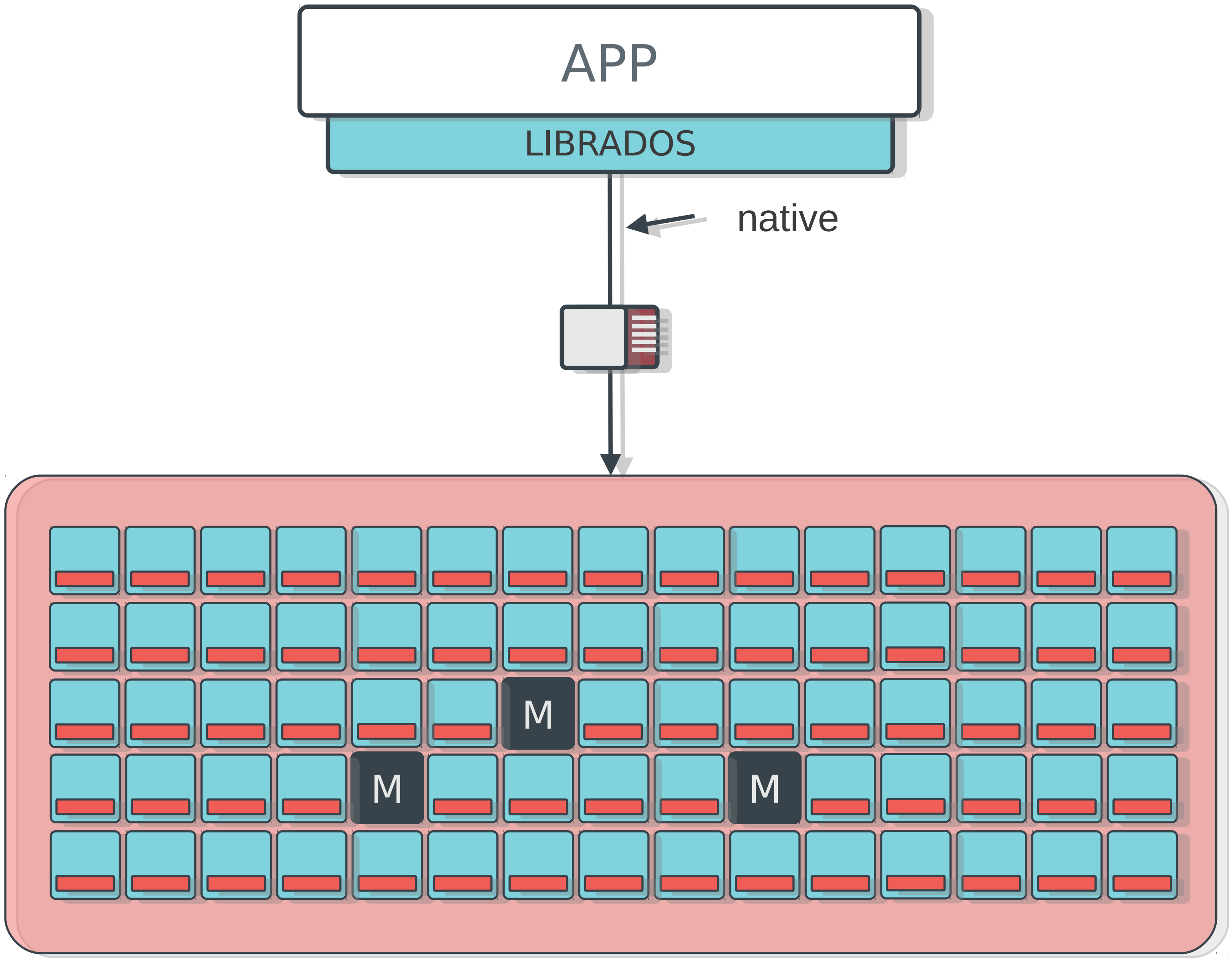


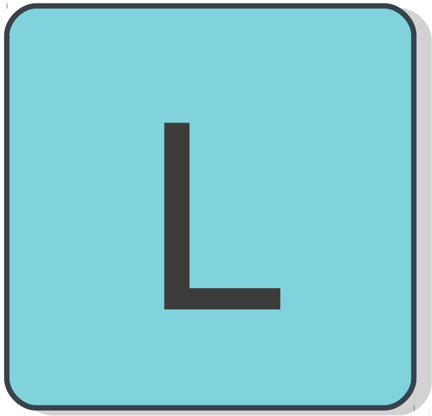



CLIENT
??


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


APP
LIBRADOS
native
M
M
M

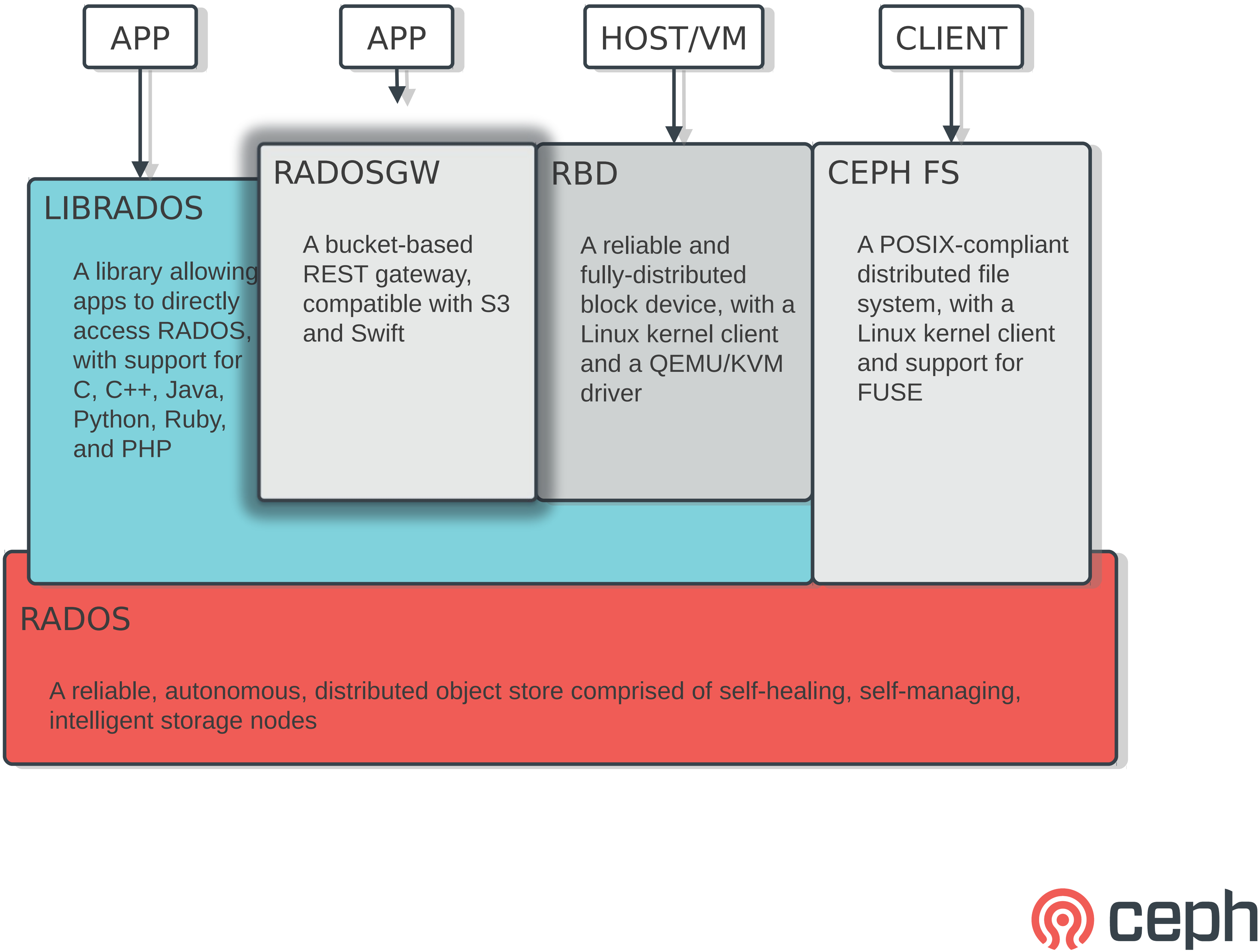


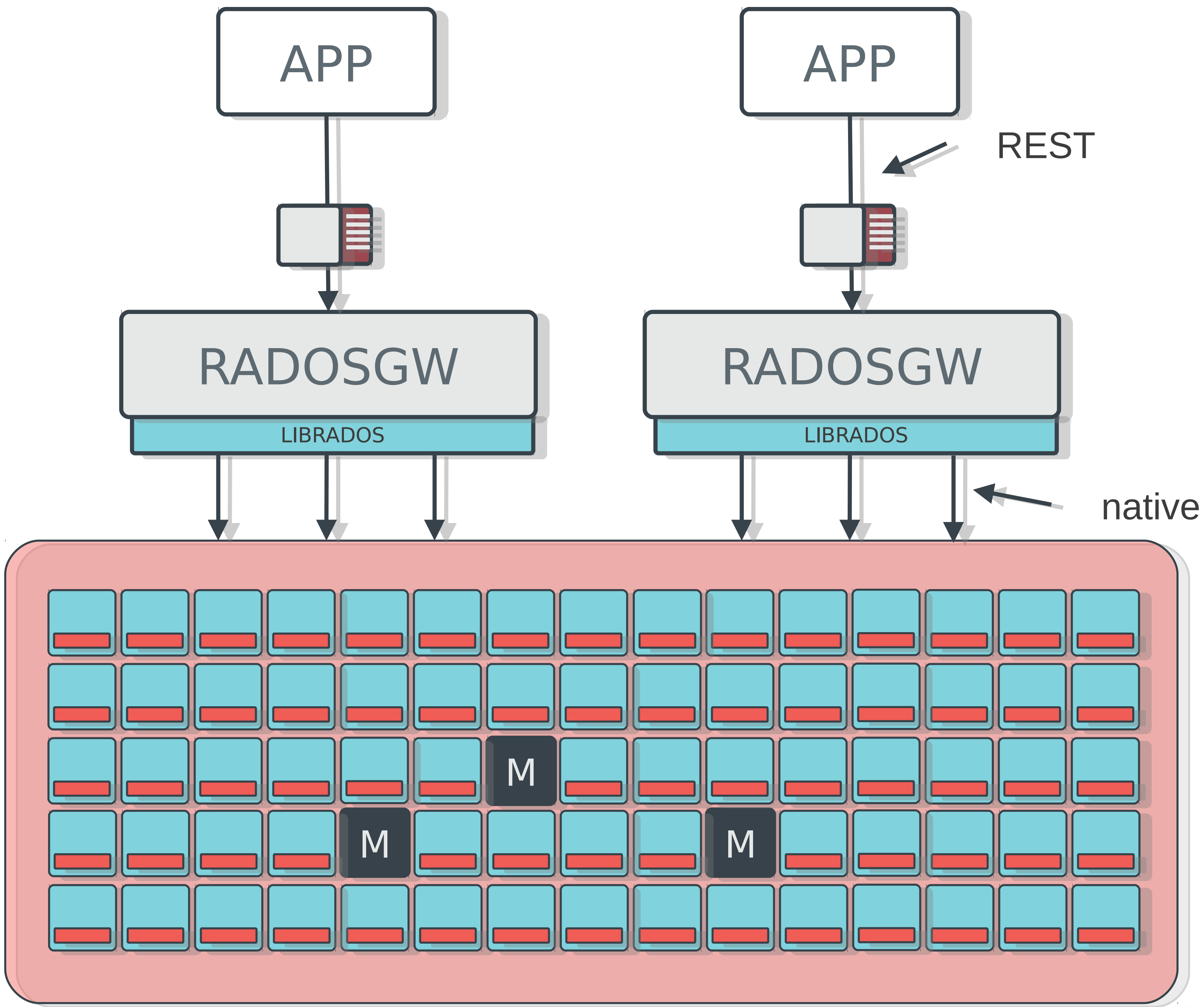
librados

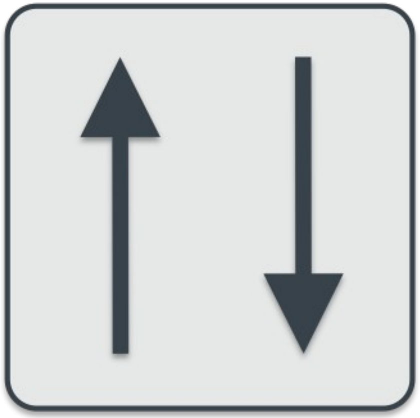
- direct access to RADOS from applications
- C, C++, Python, PHP, Java, Erlang
- direct access to storage nodes
- no HTTP overhead

rich librados API

- atomic single-object transactions
  - update data, attr together
  - atomic compare-and-swap
- efficient key/value storage inside an object
- object-granularity snapshot infrastructure
- embed code in ceph-osd daemon via plugin API
- inter-client communication via object

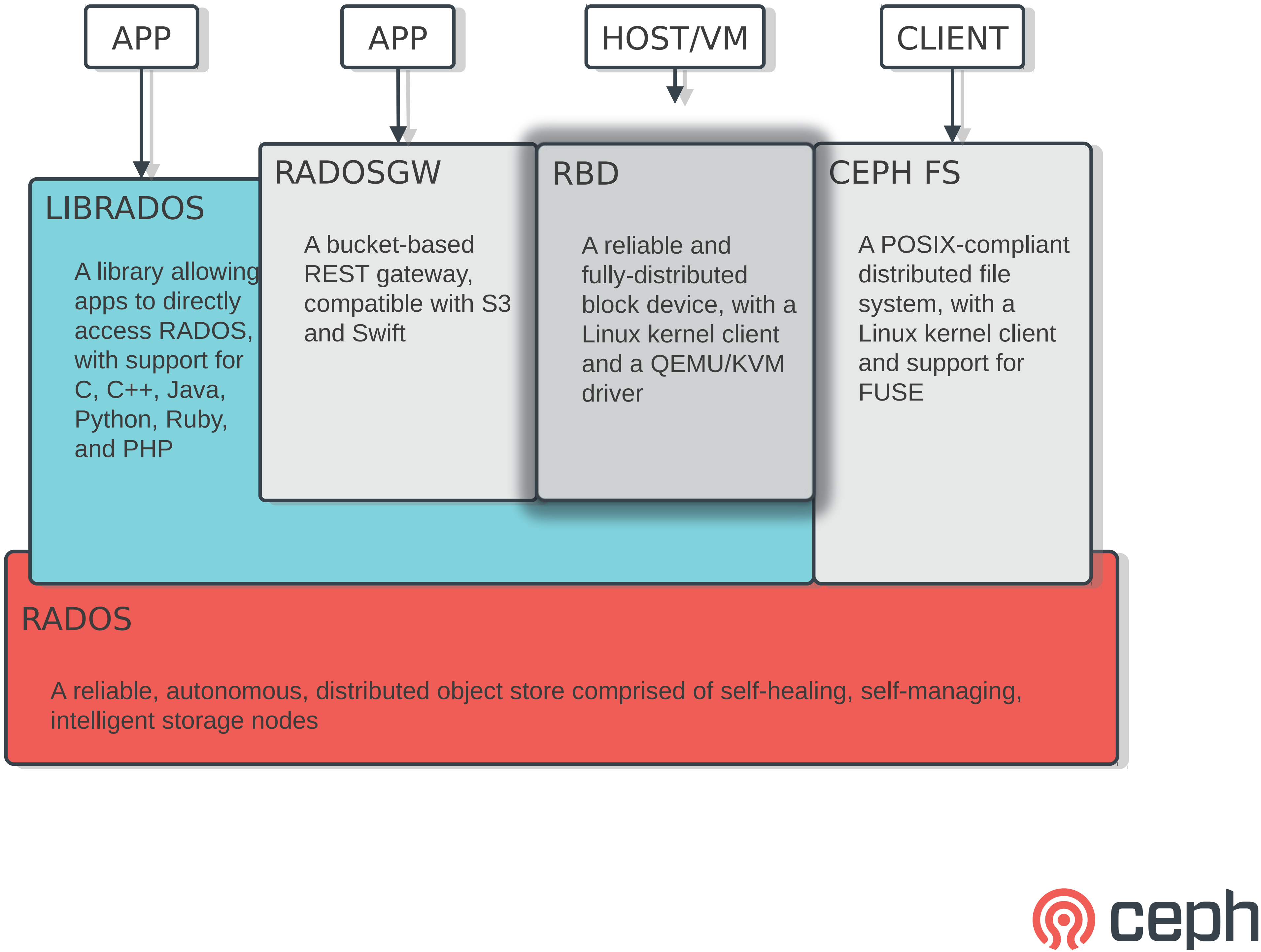

APP
APP
HOST/VM
CLIENT
LIBRADOS
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP
RADOSGW
A bucket-based REST gateway, compatible with S3 and Swift
RBD
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver
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

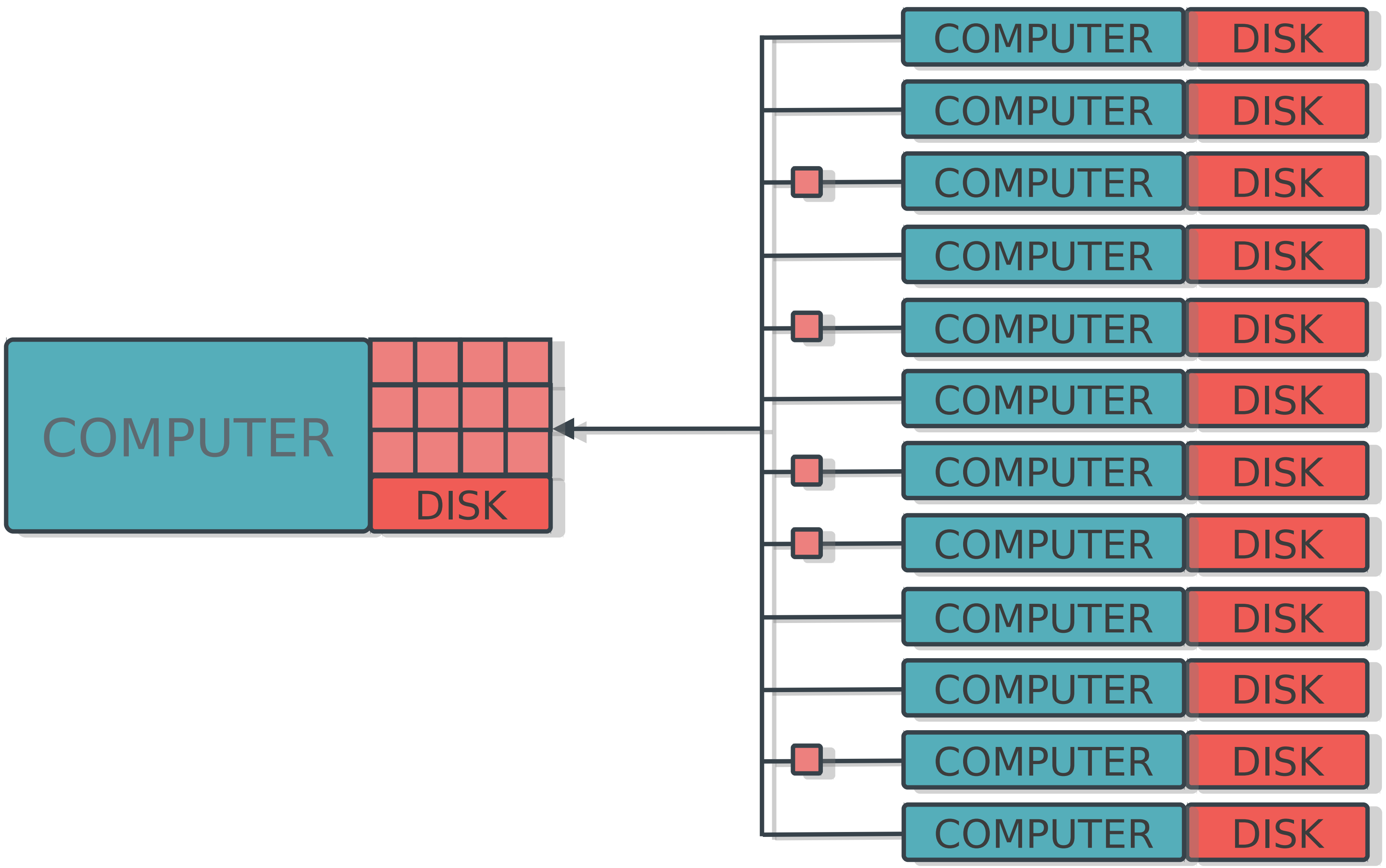

APP
APP
REST
RADOSGW
RADOSGW
LIBRADOS
LIBRADOS
native
M
M
M

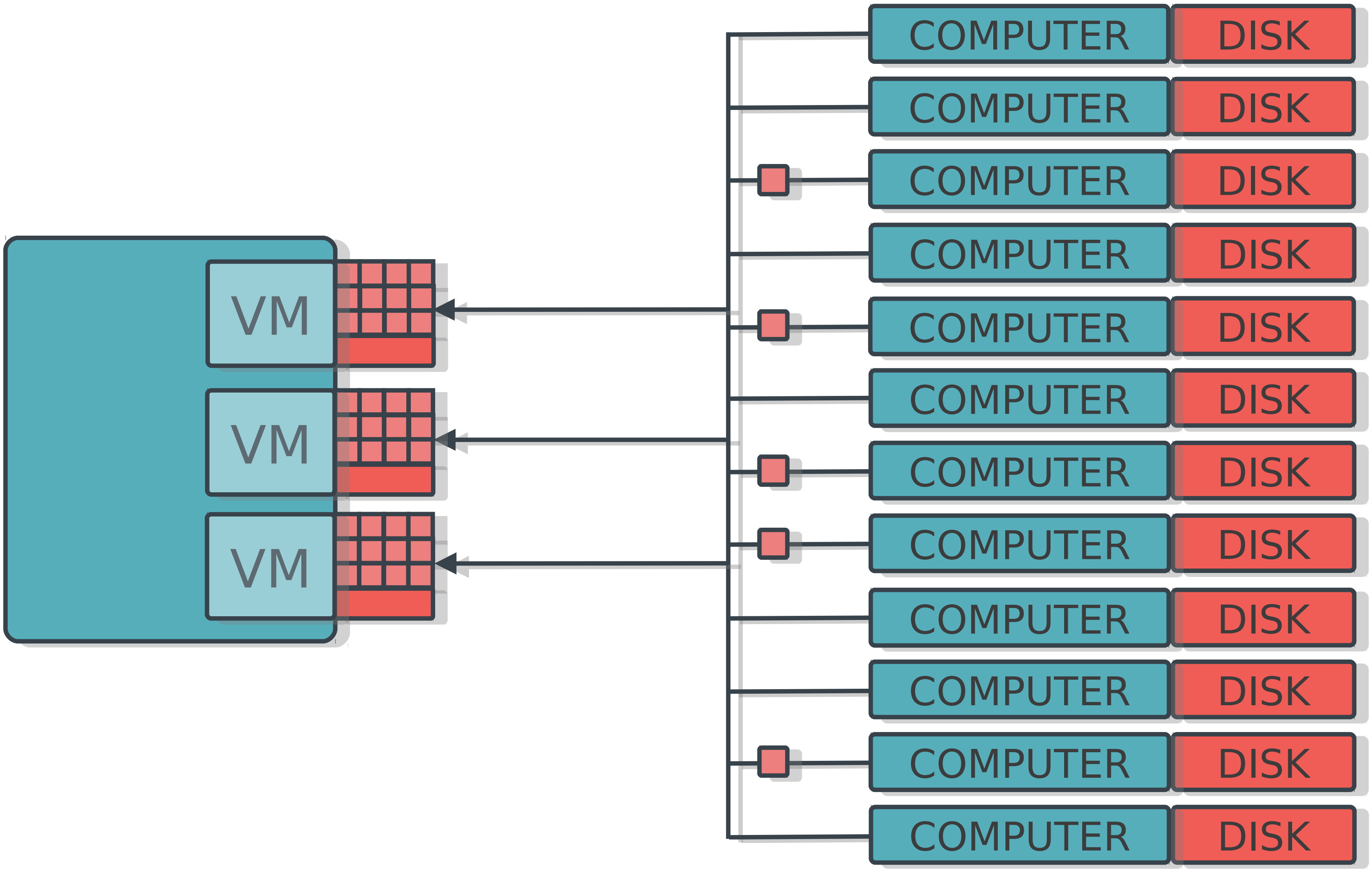


RADOS Gateway

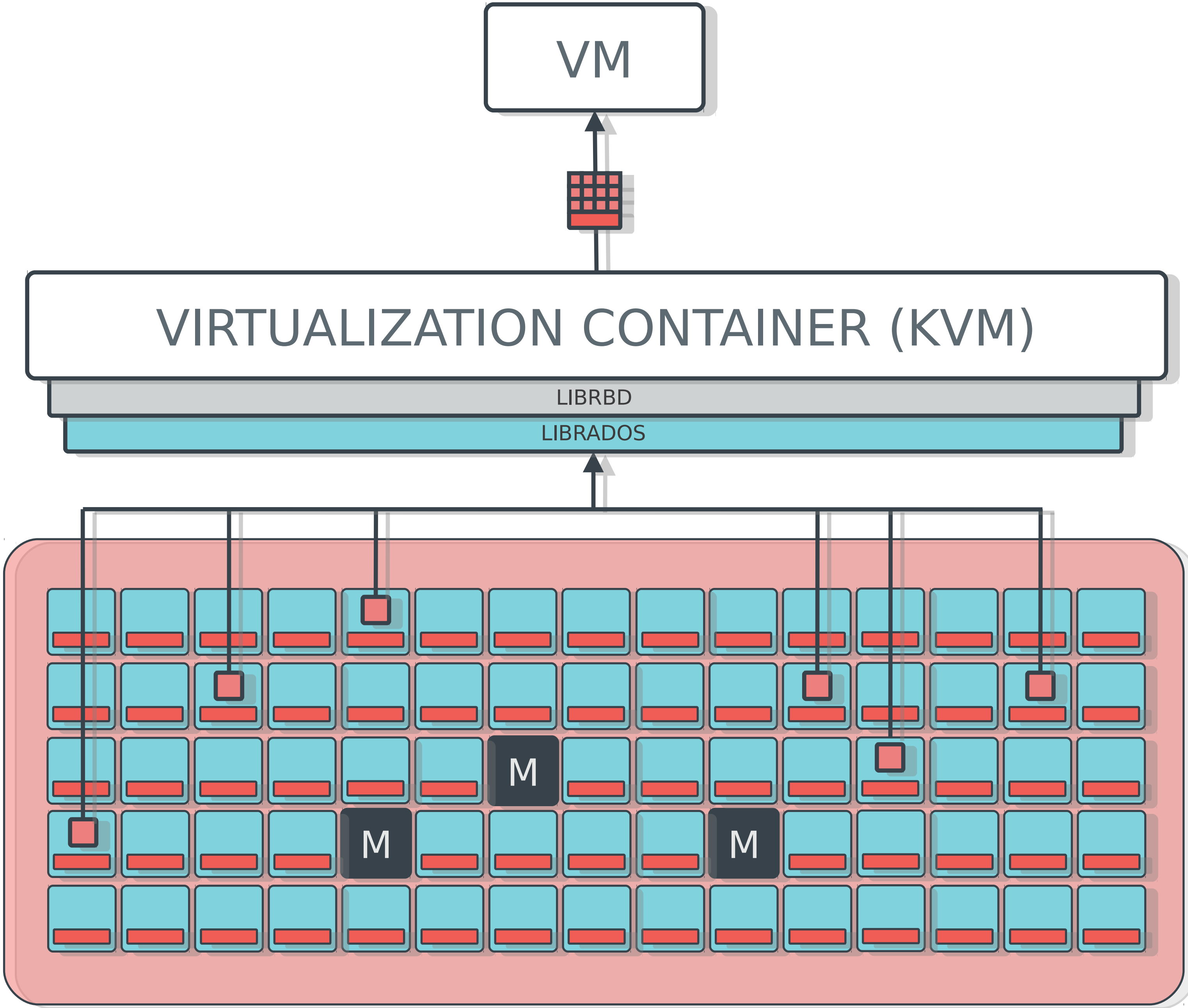
- REST-based object storage proxy
- uses RADOS to store objects
- API supports buckets, accounting
- usage accounting for billing purposes
- compatible with S3, Swift APIs


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

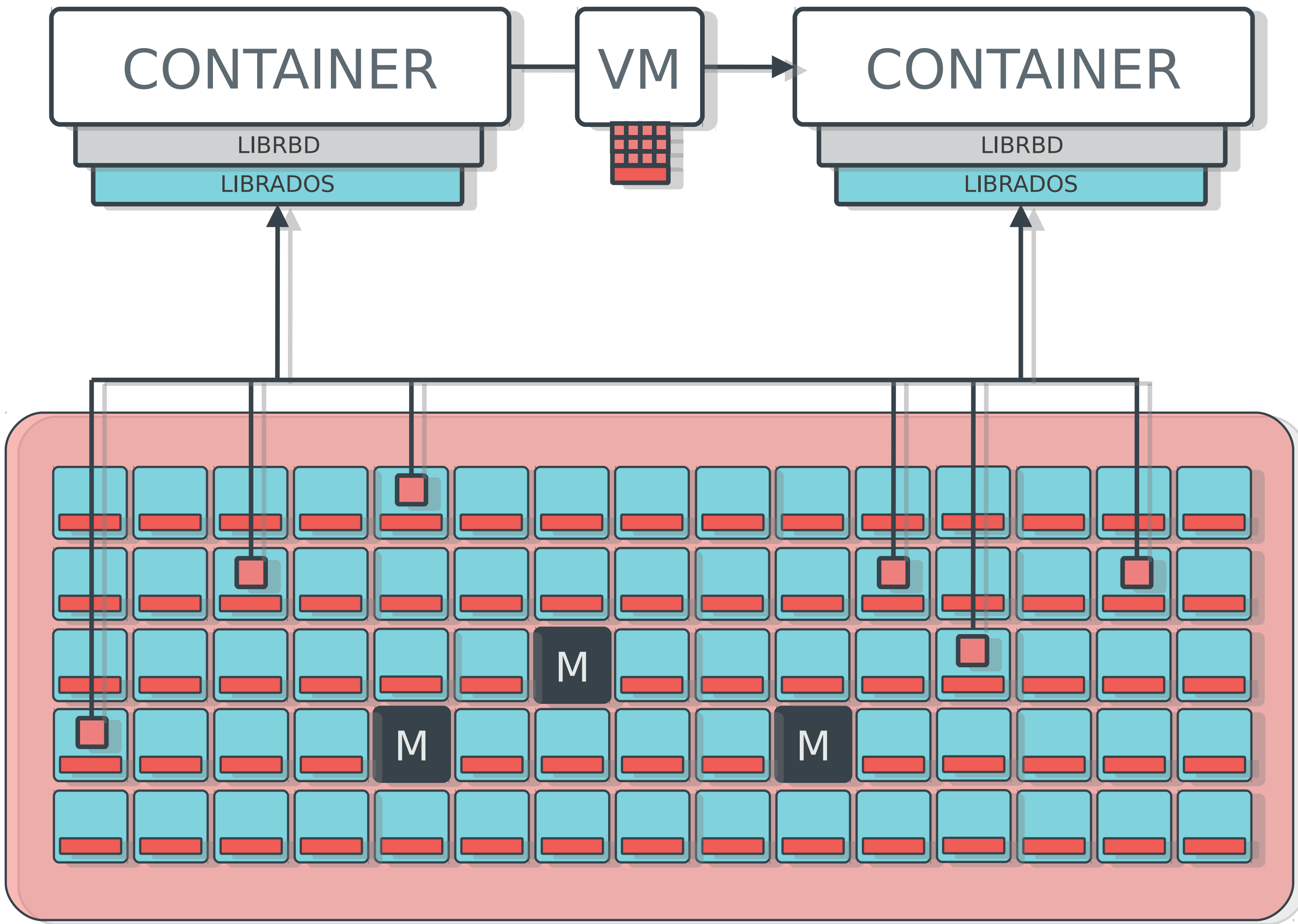

COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK


VM
VM
VM
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK
COMPUTER
DISK

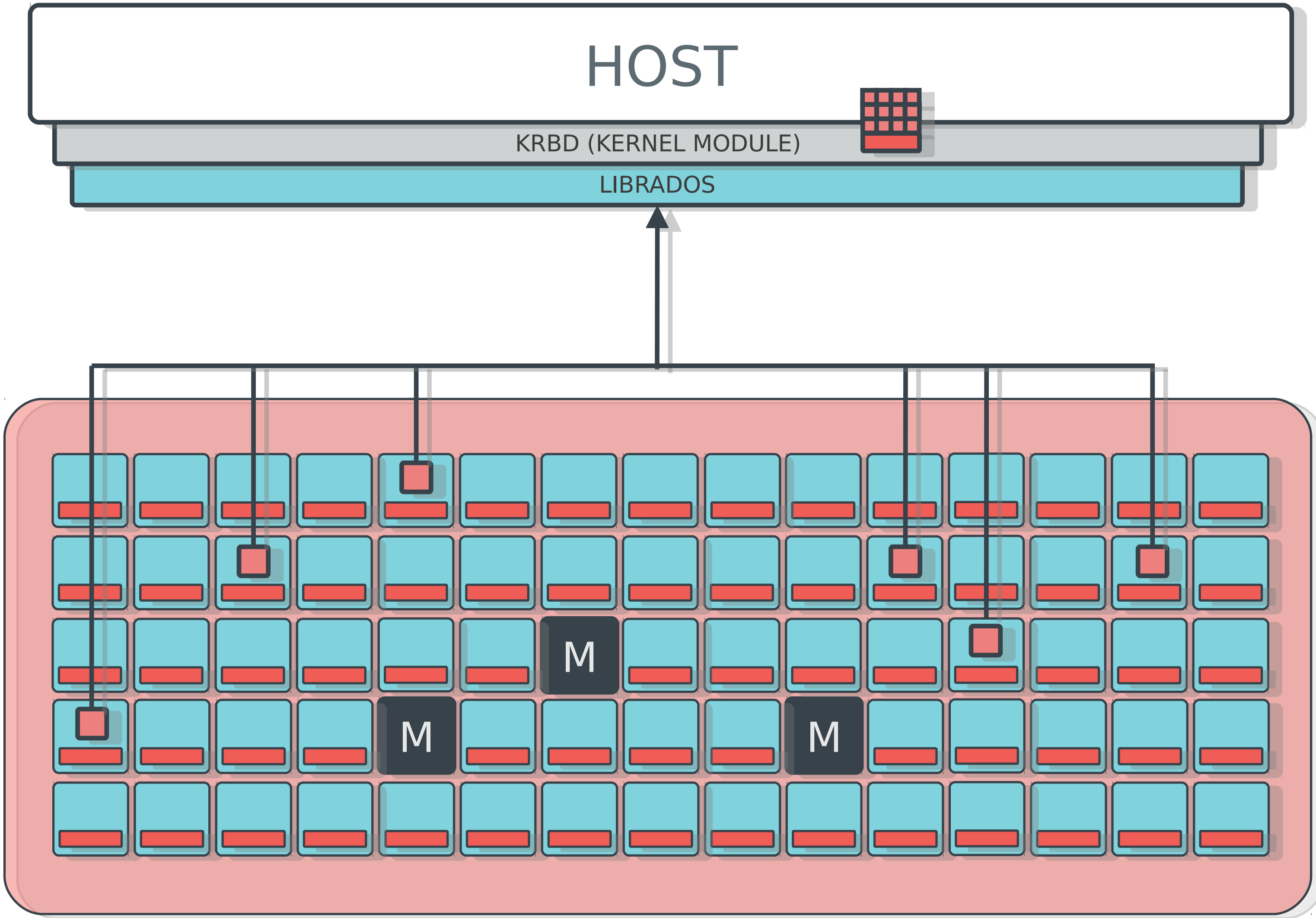
ceph

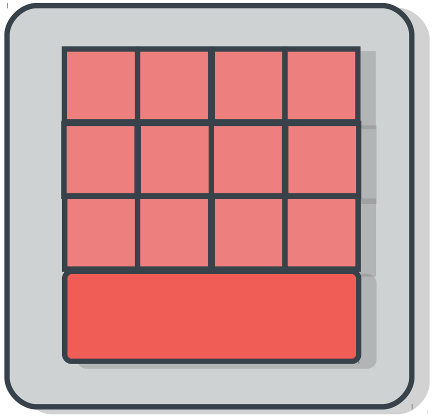


ceph


CONTAINER
VM
CONTAINER
LIBRBD
LIBRADOS
LIBRBD
LIBRADOS
M
M
M

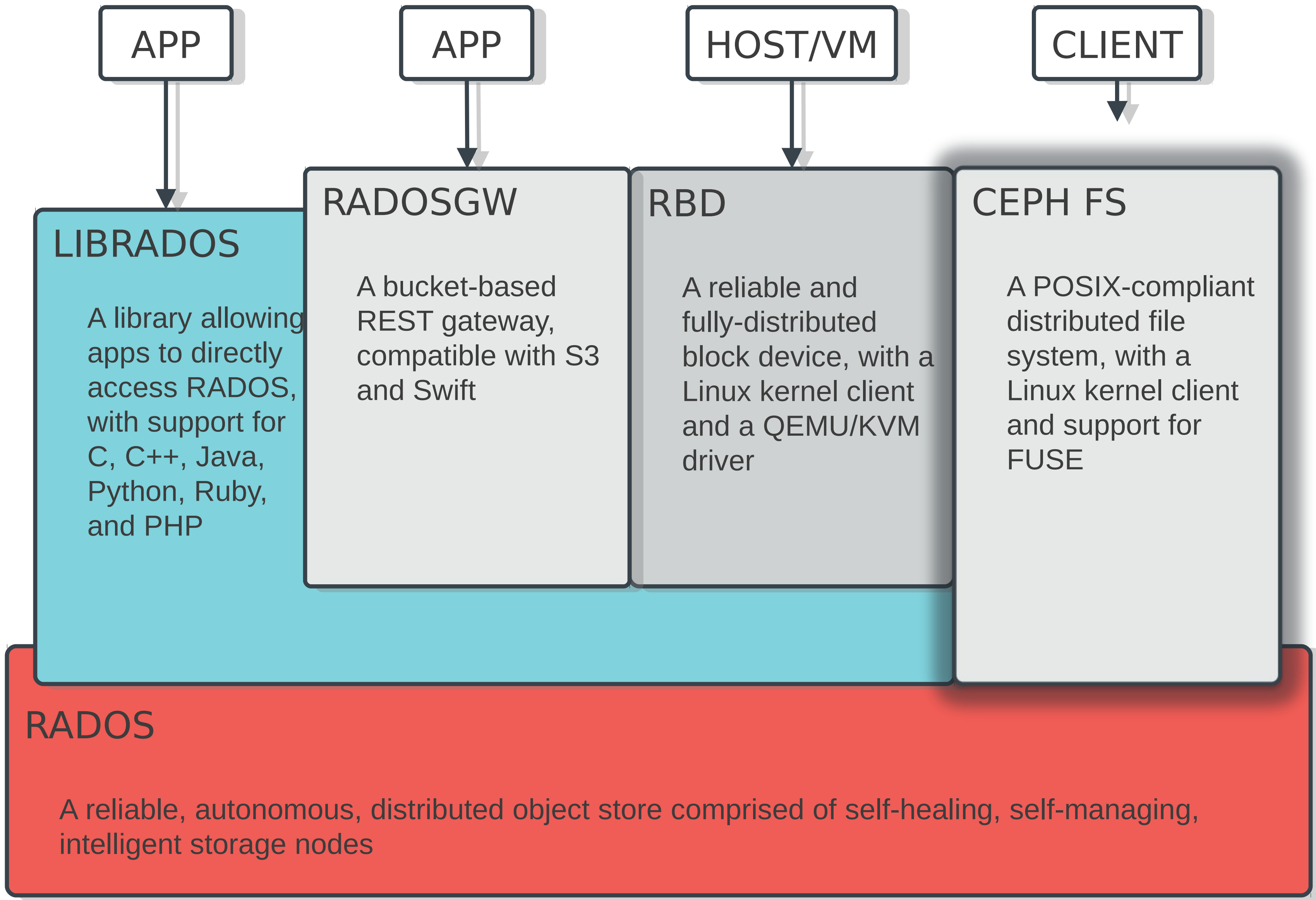
ceph

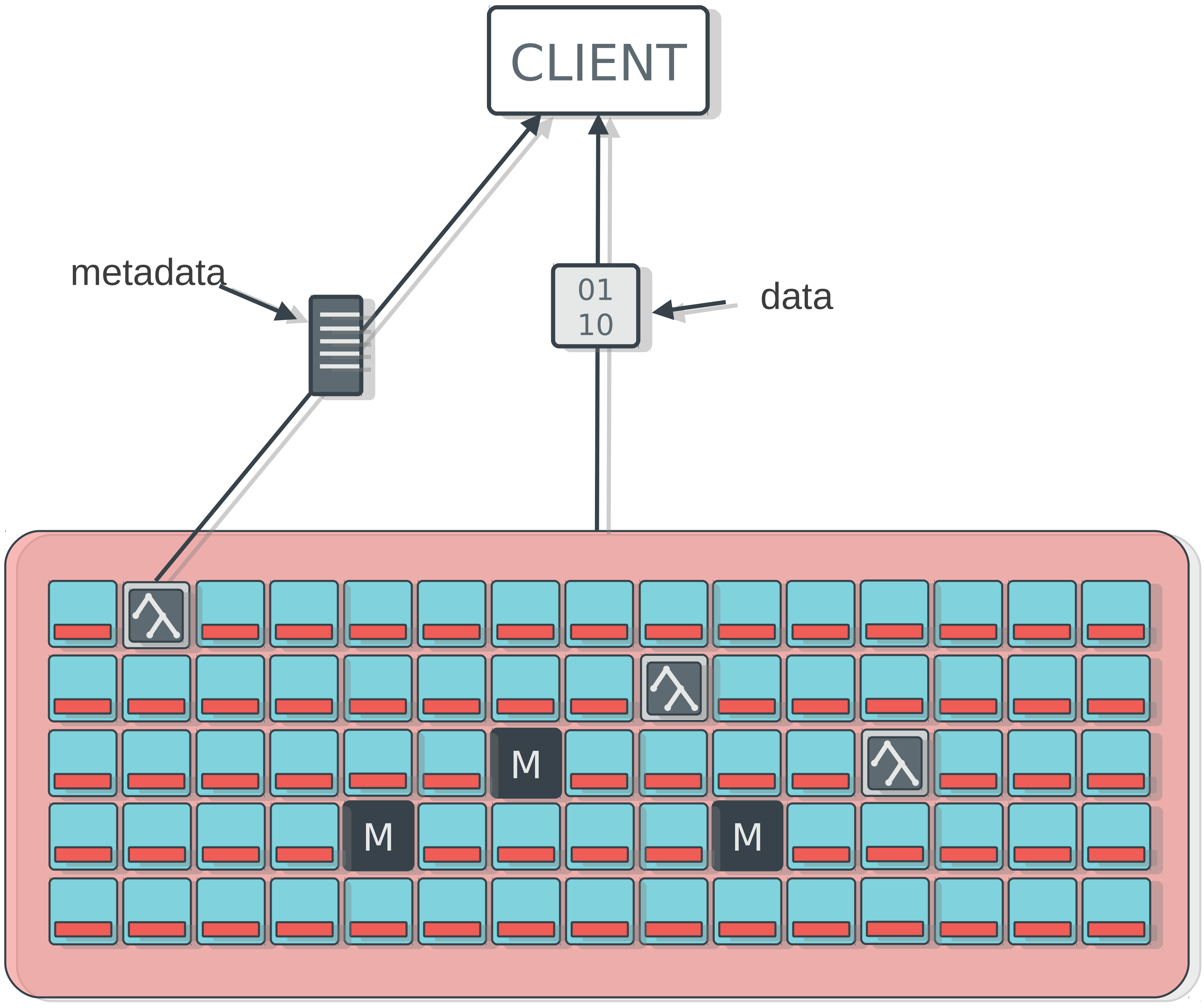

HOST
KRBD (KERNEL MODULE)
LIBRADOS
M
M
M

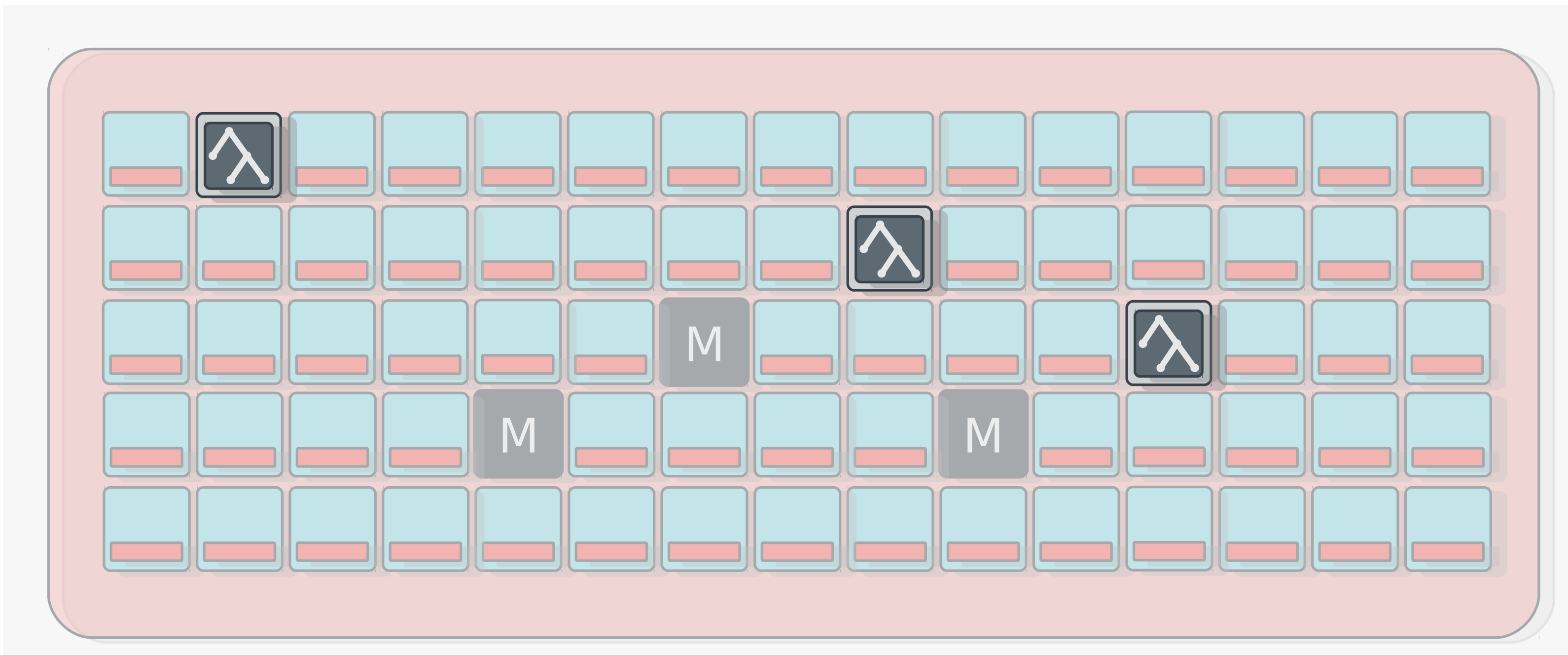


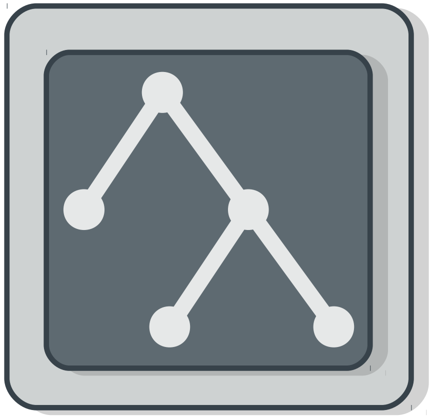
RADOS Block Device

- storage of disk images in RADOS
- decouple VM from host
- images striped across entire cluster (pool)
- snapshots
- copy-on-write clones
- support in
  - Qemu/KVM
  - mainline Linux kernel (2.6.39+)
  - OpenStack, CloudStack


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

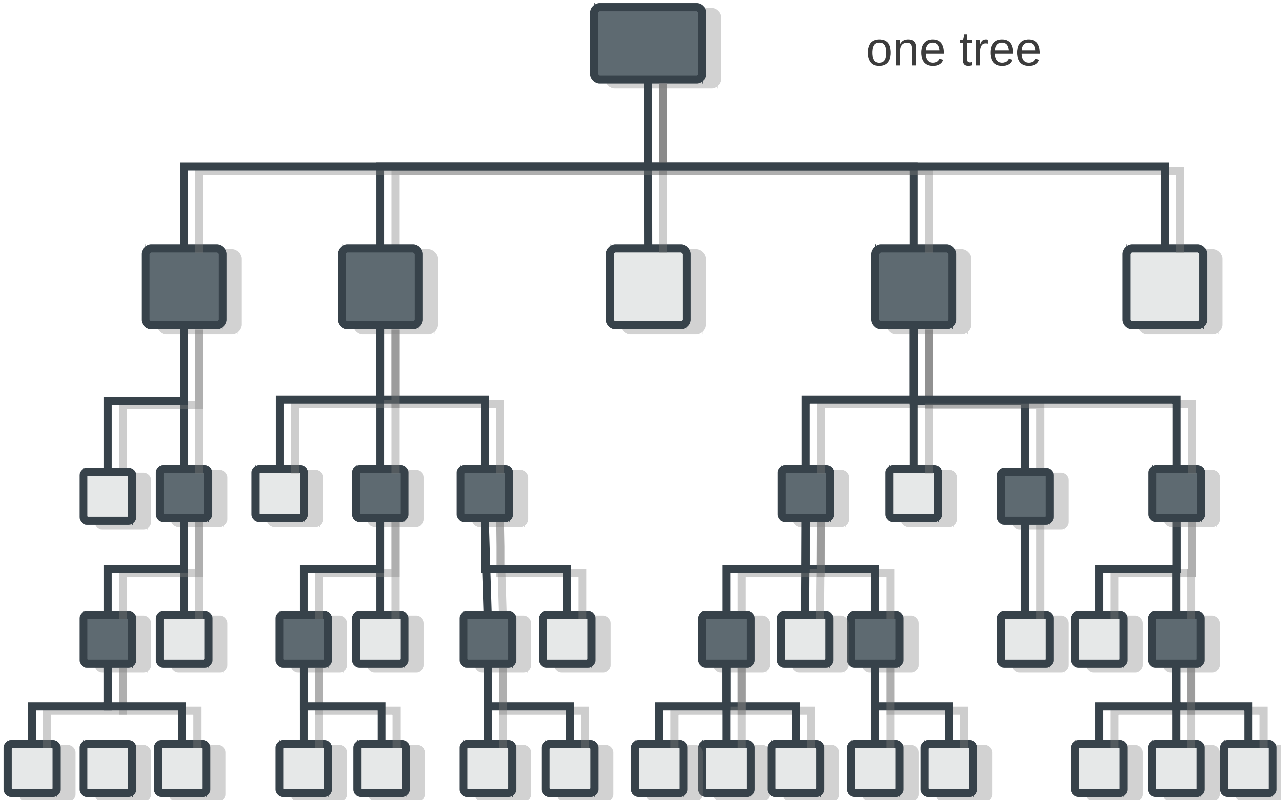

CLIENT
metadata
01
10
data
M
M
M


M
M
M



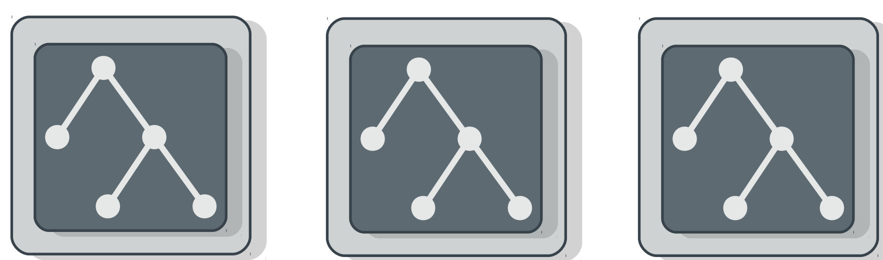
Metadata Server (MDS)

- manages metadata for POSIX shared file system
  - directory hierarchy
  - file metadata (size, owner, timestamps)
- stores metadata in RADOS
- does not serve file data to clients
- only required for the shared file system

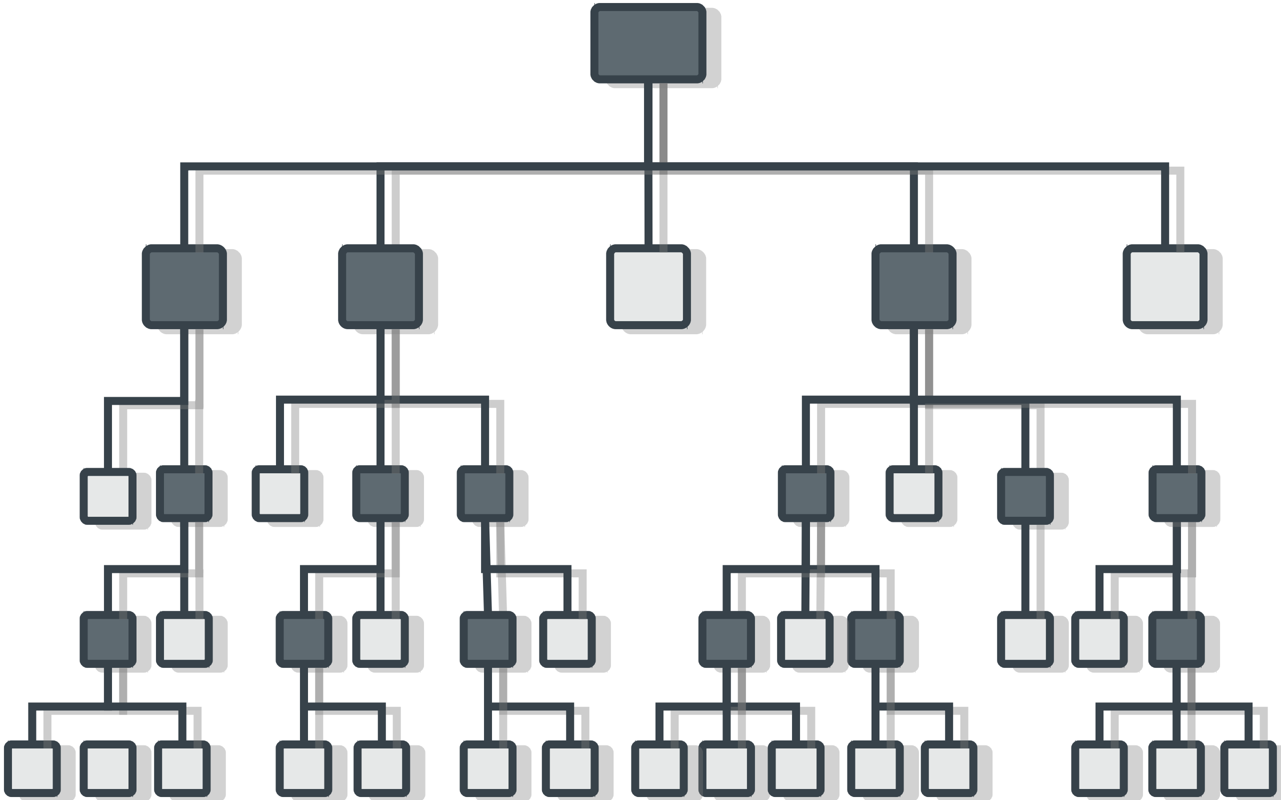


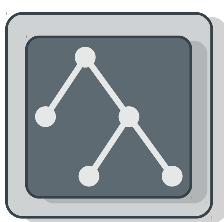
one tree

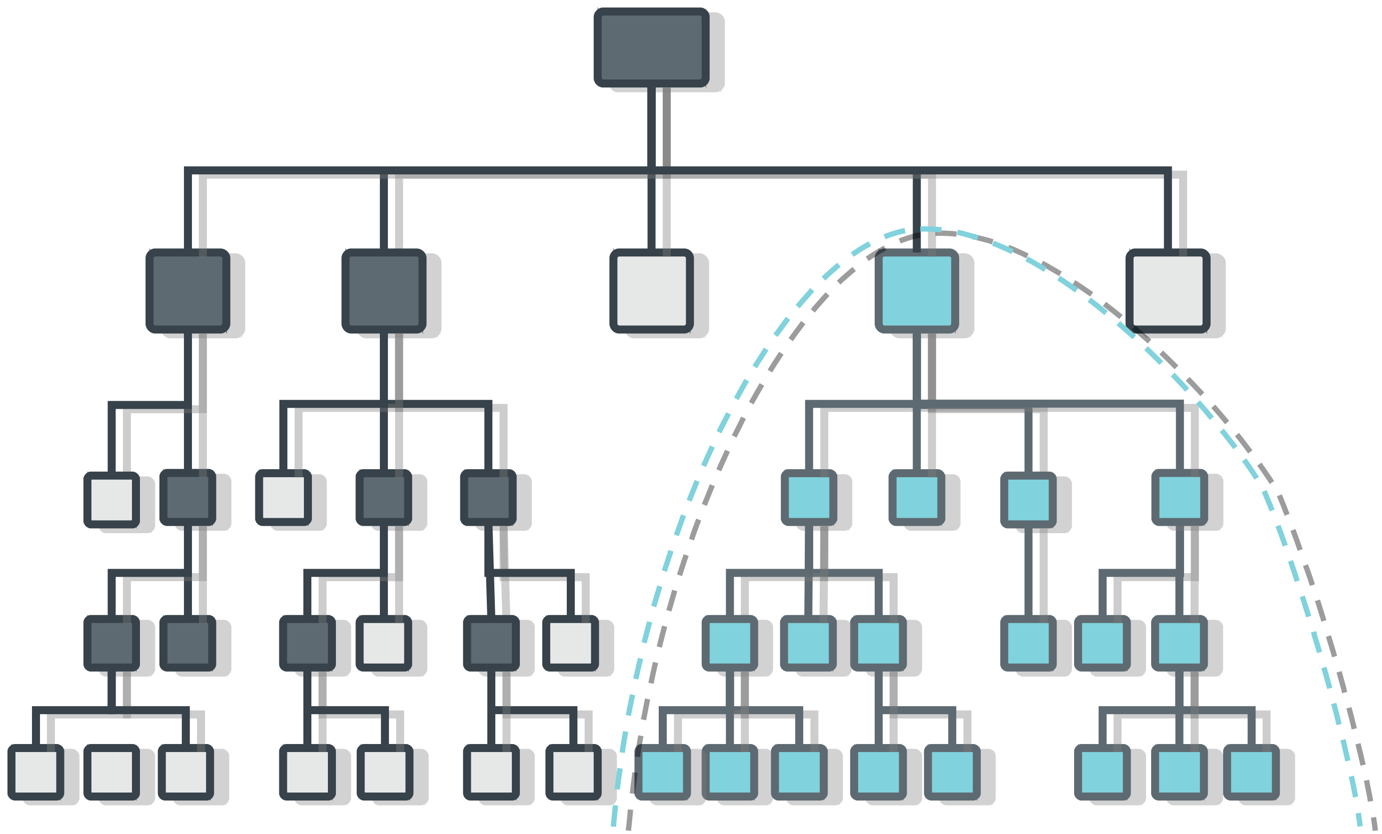
three metadata servers

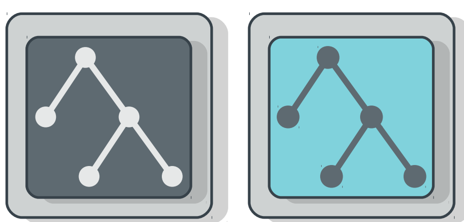


??

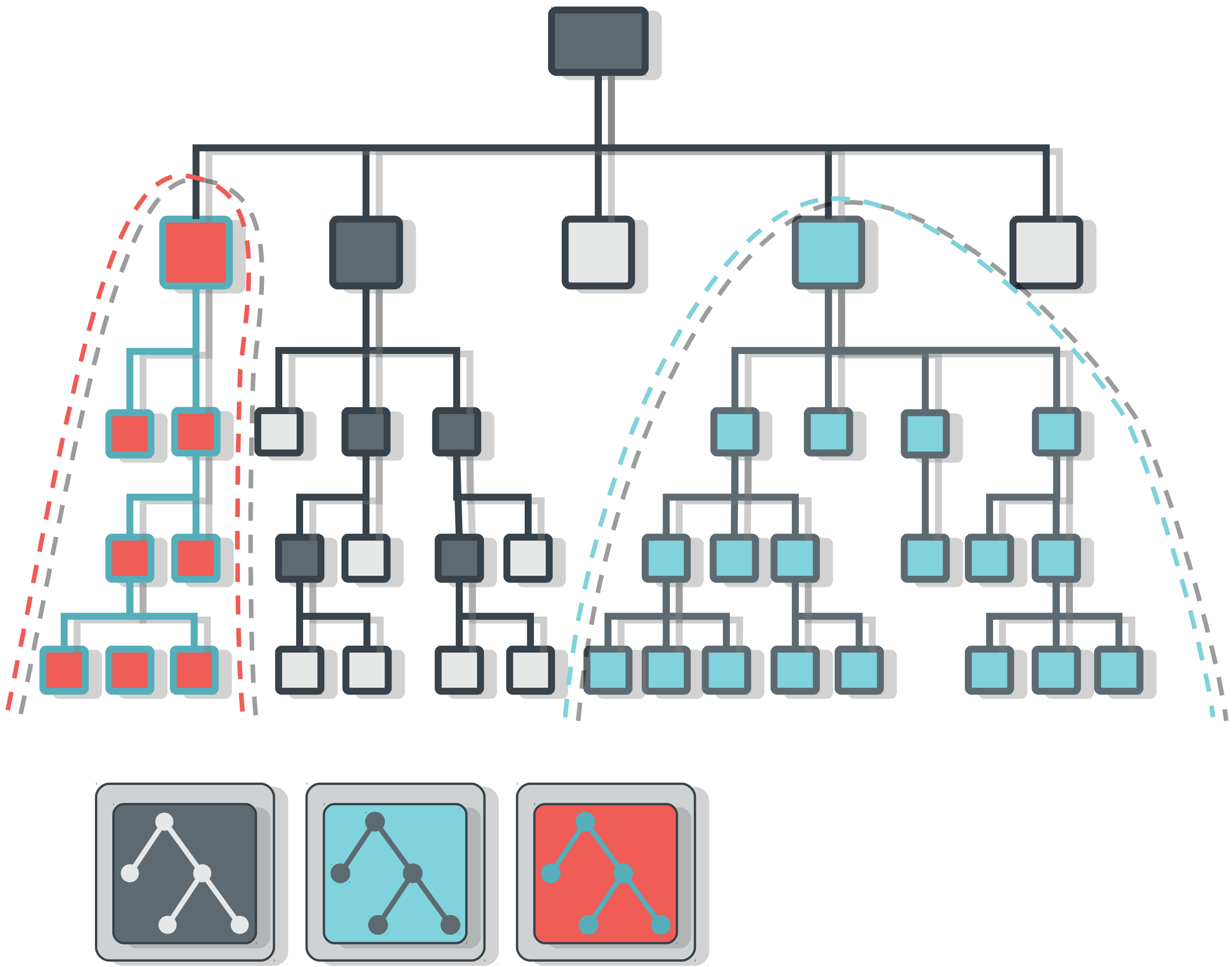




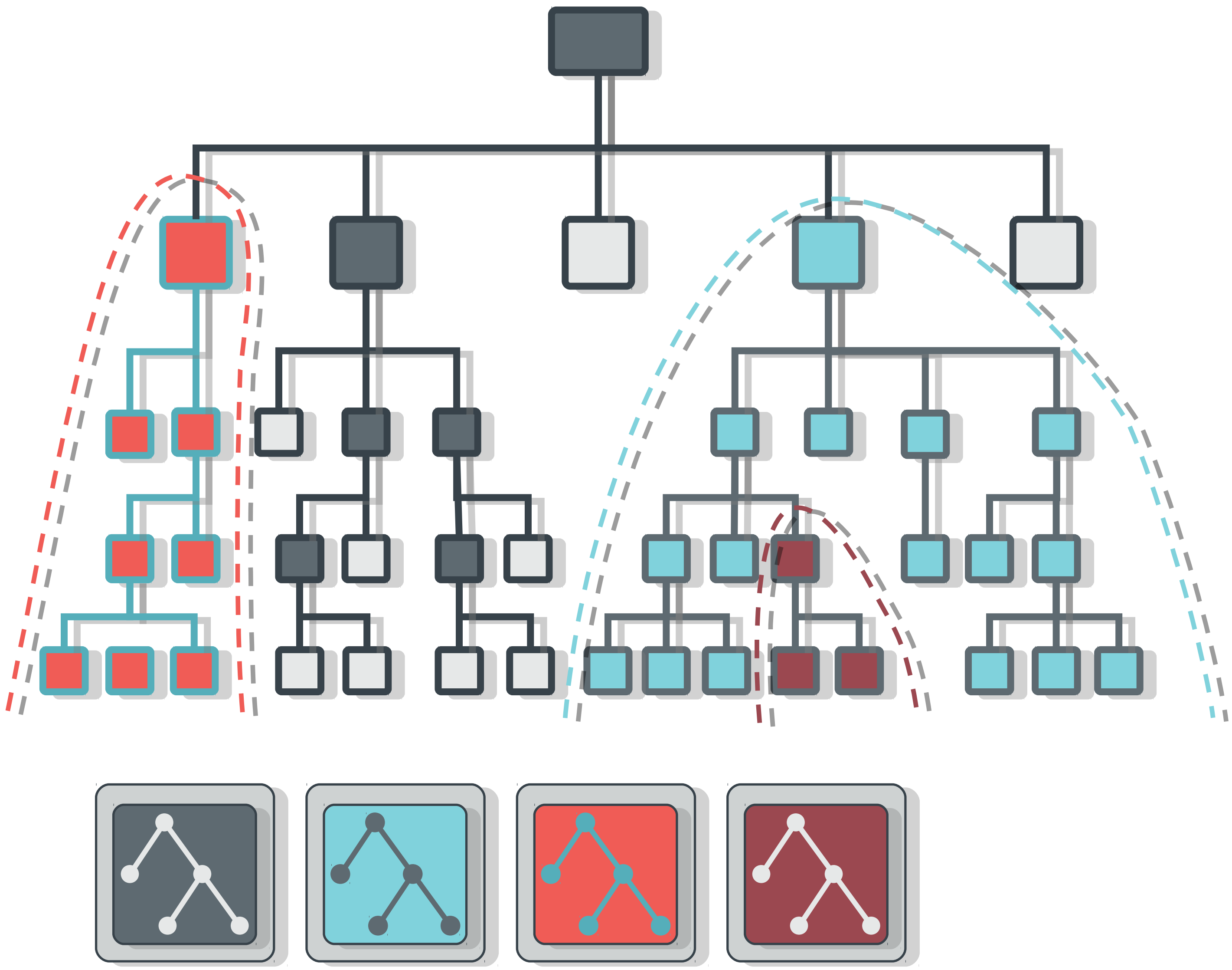


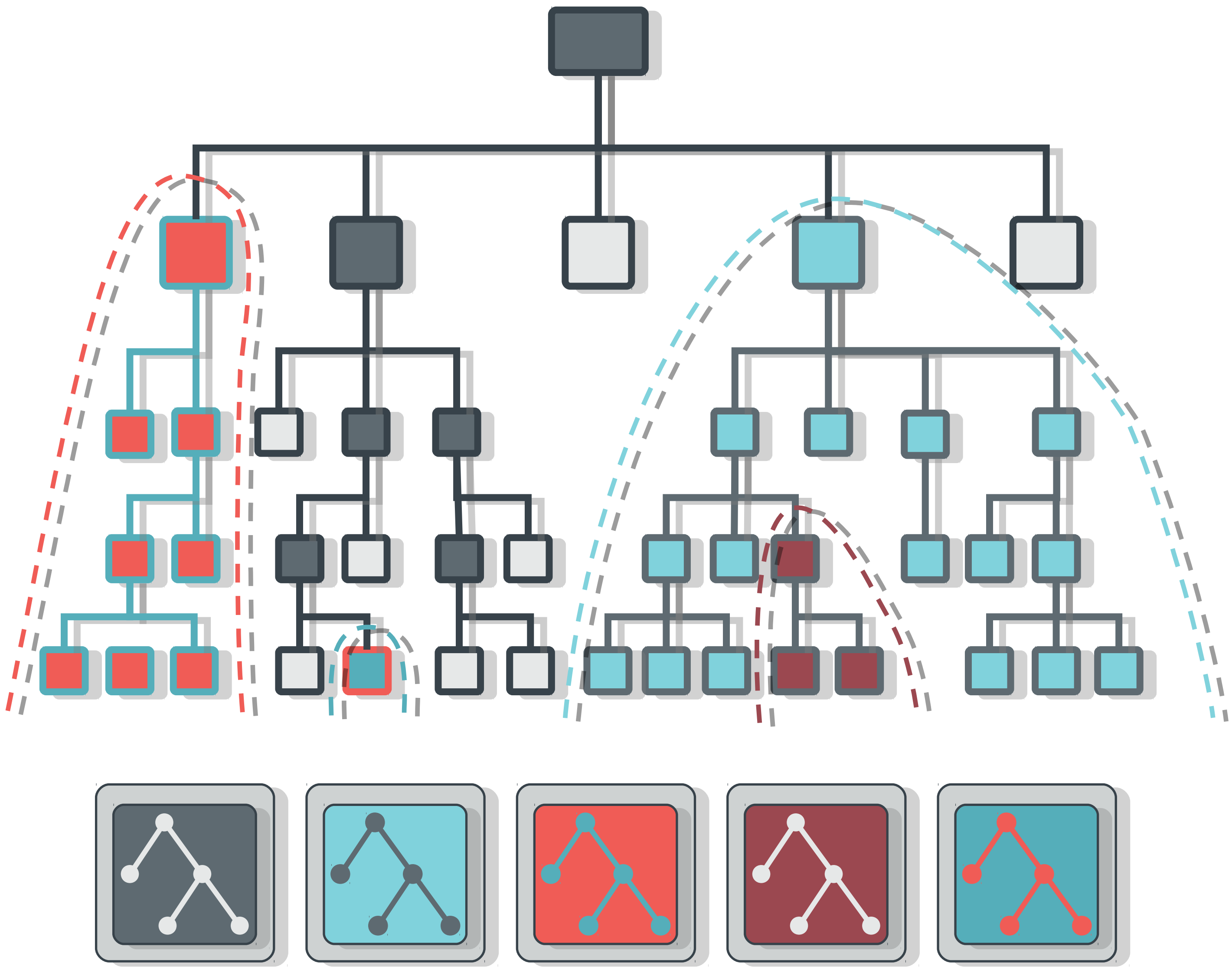


ceph



ceph





DYNAMIC SUBTREE PARTITIONING

ceph

recursive accounting

- ceph-mds tracks recursive directory stats
  - file sizes
  - file and directory counts
  - modification time
- virtual xattrs present full stats
- efficient

```
$ ls -alSh | head
total 0
drwxr-xr-x 1 root            root      9.7T 2011-02-04 15:51 .
drwxr-xr-x 1 root            root      9.7T 2010-12-16 15:06 ..
drwxr-xr-x 1 pomceph         pg4194980 9.6T 2011-02-24 08:25 pomceph
drwxr-xr-x 1 mcg_test1       pg2419992  23G 2011-02-02 08:57 mcg_test1
drwx--x--- 1 luko            adm        19G 2011-01-21 12:17 luko
drwx--x--- 1 eest            adm        14G 2011-02-04 16:29 eest
drwxr-xr-x 1 mcg_test2       pg2419992 3.0G 2011-02-02 09:34 mcg_test2
drwx--x--- 1 fuzyceph        adm       1.5G 2011-01-18 10:46 fuzyceph
drwxr-xr-x 1 dallasceph      pg275     596M 2011-01-14 10:06 dallasceph
```

snapshots

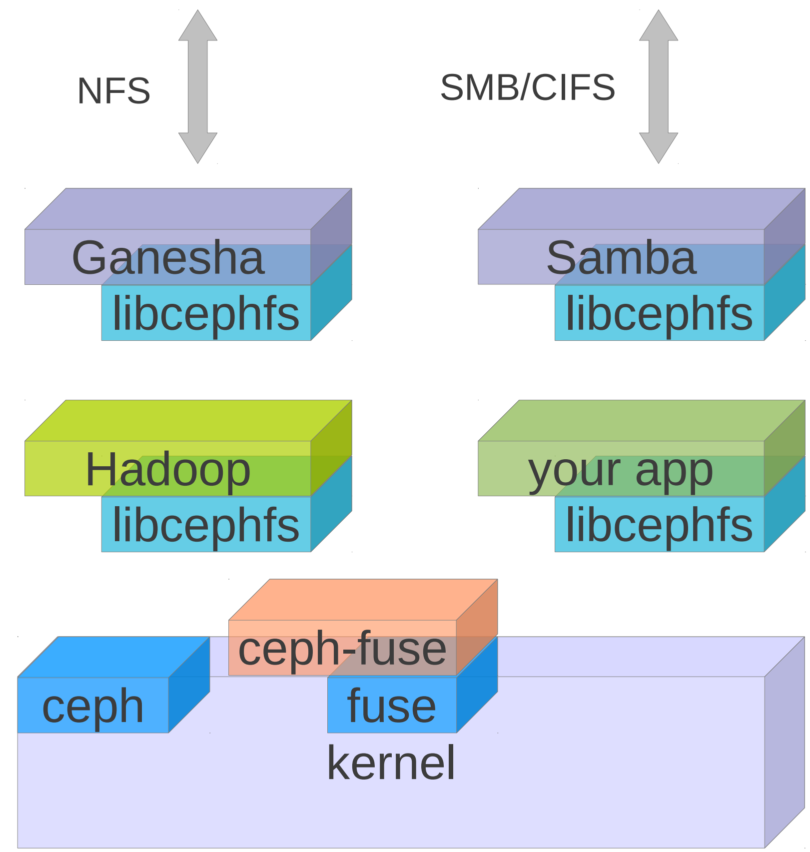
- volume or subvolume snapshots unusable at petabyte scale
  - snapshot arbitrary subdirectories
- simple interface
  - hidden '.snap' directory
  - no special tools

```
$ mkdir foo/.snap/one    # create snapshot
$ ls foo/.snap
one
$ ls foo/bar/.snap
_one_1099511627776       # parent's snap name is mangled
$ rm foo/myfile
$ ls -F foo
bar/
$ ls -F foo/.snap/one
myfile  bar/
$ rmdir foo/.snap/one    # remove snapshot
```

ceph

multiple protocols, implementations

- Linux kernel client
  - mount -t ceph 1.2.3.4:/ /mnt
  - export (NFS), Samba (CIFS)
- ceph-fuse
- libcephfs.so
  - your app
  - Samba (CIFS)
  - Ganesha (NFS)
  - Hadoop (map/reduce)



How do you deploy and manage all this stuff?

installation

- work closely with distros
  - packages in Debian, Fedora, Ubuntu, OpenSUSE
- build releases for all distros, recent releases
  - e.g., point to our deb src and apt-get install
- allow mixed-version clusters
  - upgrade each component individually, each host or rack one-by-one
  - protocol feature bits, safe data type encoding
- facilitate testing, bleeding edge
  - automatic build system generates packages for all branches in git
  - each to test new code, push hot-fixes

ceph

configuration

- minimize local configuration
  - logging, local data paths, tuning options
  - cluster state is managed centrally by monitors
- specify option via
  - config file (global or local)
  - command line
  - adjusted for running daemon
- flexible configuration management options
  - global synced/copied config file
  - generated by management tools
    - Chef, Juju, Crowbar
- be flexible

provisioning

- embrace dynamic nature of the cluster
  - disks, hosts, rack may come online at any time
  - anything may fail at any time
- simple scriptable sequences

  'ceph osd create <uuid>' → allocate osd id

  'ceph-osd --mkfs -i <id>' → initialize local data dir

  'ceph osd crush …' → add to crush map
- identify minimal amount of central coordination
  - monitor cluster membership/quorum
- provide hooks for external tools to do the rest

old school HA

- deploy a pair of servers
- heartbeat
- move a floating IP between then
- clients contact same “server”, which floats

- cumbersome to configure, deploy
- the “client/server” model doesn't scale out

make client, protocol cluster-aware

- clients learn topology of the distributed cluster
  - all OSDs, current ips/ports, data distribution map
- clients talk to the server they want
  - servers scale out, client traffic distributes too
- servers dynamically register with the cluster
  - when a daemon starts, it updates its address in the map
  - other daemons and clients learn map updates via gossip
- servers manage heartbeat, replication
  - no fragile configuration
  - no fail-over pairs

ceph disk management

- label disks
  - GPT partition type (fixed uuid)
- udev
  - generate event when disk is added, on boot
- 'ceph-disk-activate /dev/sdb'
  - mount the disk in the appropriate location (/var/lib/ceph/osd/NNN)
  - possibly adjust cluster metadata about disk location (host, rack)
- upstart, sysvinit, …
  - start the daemon
  - daemon "joins" the cluster, brings itself online
- no manual per-node configuration

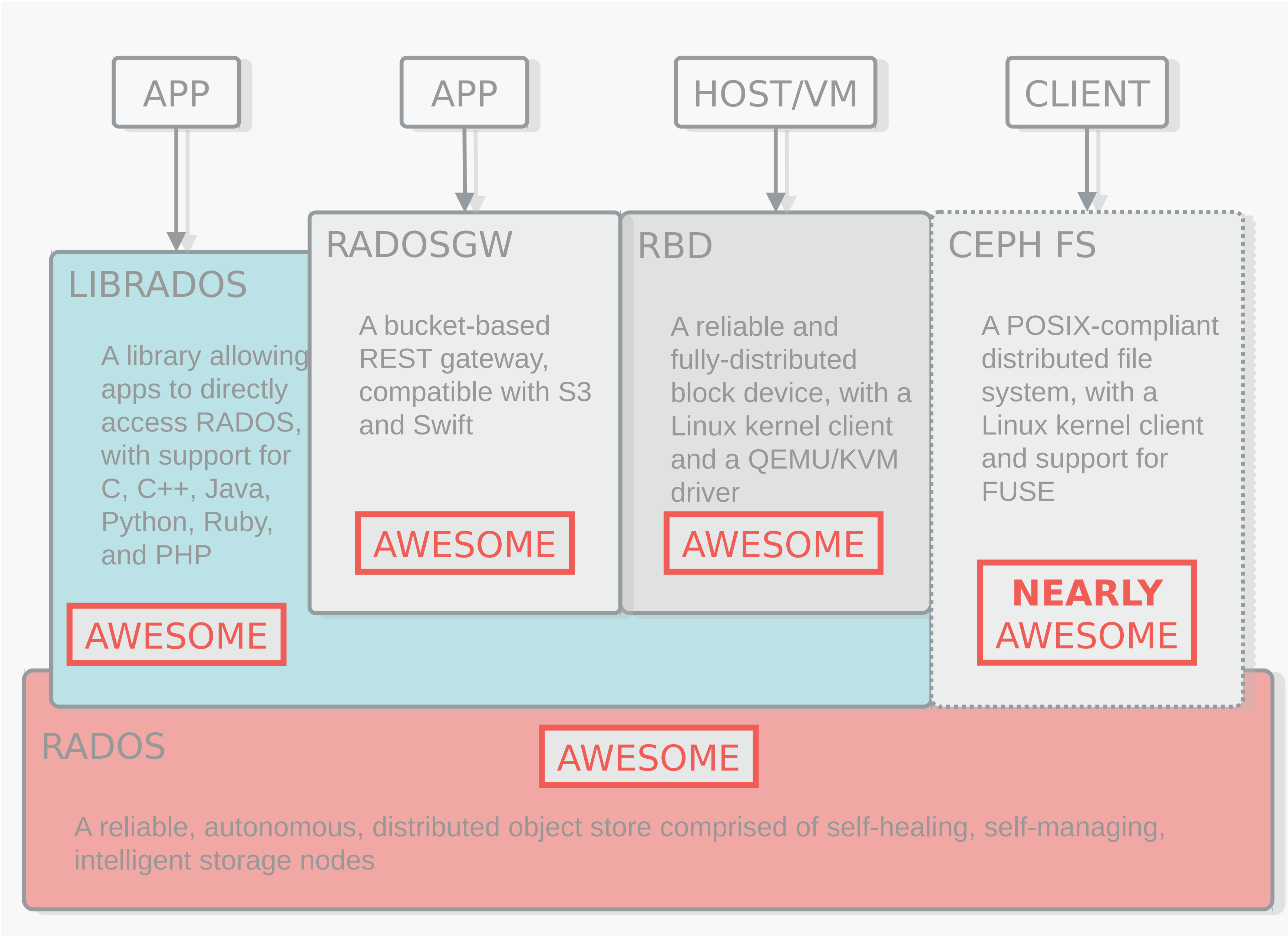
distributed system health

- ceph-mon collects, aggregates basic system state
  - daemon up/down, current IPs
  - disk utilization
- simple hooks query system health
  - 'ceph health [detail]'
  - trivial plugins for nagios, etc.
- daemons instrumentation
  - query state of running daemon
  - use external tools to aggregate
    - collectd, graphite
    - statsd

many paths

- do it yourself
  - use ceph interfaces directly
- Chef
  - ceph cookbooks, DreamHost cookbooks
- Crowbar
- Juju
- pre-packaged solutions
  - Piston AirFrame, Dell OpenStack-Powered Cloud Solution, Suse Cloud, etc.

Project status and roadmap


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

ceph

current status

- argonaut stable release v0.48
  - rados, RBD, radosgw
- bobtail stable release v0.56
  - RBD cloning
  - improved performance, scaling, failure behavior
  - radosgw API, performance improvements
  - release in 1-2 weeks

ceph

cuttlefish roadmap

- file system
  - pivot in engineering focus
  - CIFS (Samba), NFS (Ganesha), Hadoop
- RBD
  - Xen integration, iSCSI
- radosgw
  - multi-side federation, disaster recovery
- RADOS
  - geo-replication, disaster recovery
  - ongoing performance improvements

ceph

why we do this

- limited options for scalable open source storage
- proprietary solutions
  - expensive
  - don't scale (well or out)
  - marry hardware and software
- users hungry for alternatives
  - scalability
  - cost
  - features
  - open, interoperable

licensing

<yawn>

- promote adoption
- enable community development
- prevent ceph from becoming proprietary
- allow organic commercialization

LGPLv2

- “copyleft”
  - free distribution
  - allow derivative works
  - changes you distribute/sell must be shared
- ok to link to proprietary code
  - allow proprietary products to incude and build on ceph
  - does not allow proprietary derivatives of ceph

ceph

fragmented copyright

- we do not require copyright assignment from contributors
  - no single person or entity owns all of ceph
  - no single entity can make ceph proprietary
- strong community
  - many players make ceph a safe technology bet
  - project can outlive any single business

why it's important

- ceph is an ingredient
  - we need to play nice in a larger ecosystem
  - community will be key to success
- truly open source solutions are disruptive
  - frictionless integration with projects, platforms, tools
  - freedom to innovate on protocols
  - leverage community testing, development resources
  - open collaboration is efficient way to build technology

who we are

- Ceph created at UC Santa Cruz (2004-2007)
- developed by DreamHost (2008-2011)
- supported by Inktank (2012)
  - Los Angeles, Sunnyvale, San Francisco, remote
- growing user and developer community
  - Linux distros, users, cloud stacks, SIs, OEMs

http://ceph.com/

inktank ceph

thanks

sage weil

sage@inktank.com

@liewegas

http://github.com/ceph

http://ceph.com/

why we like btrfs

- pervasive checksumming
- snapshots, copy-on-write
- efficient metadata (xattrs)
- inline data for small files
- transparent compression
- integrated volume management
  - software RAID, mirroring, error recovery
  - SSD-aware
- online fsck
- active development community