ERASURE CODING AND CACHE TIERING

SAGE WEIL – SCALE13X - 2015.02.22

AGENDA

- Ceph architectural overview
- RADOS background
- Cache tiering
- Erasure coding
- Project status, roadmap

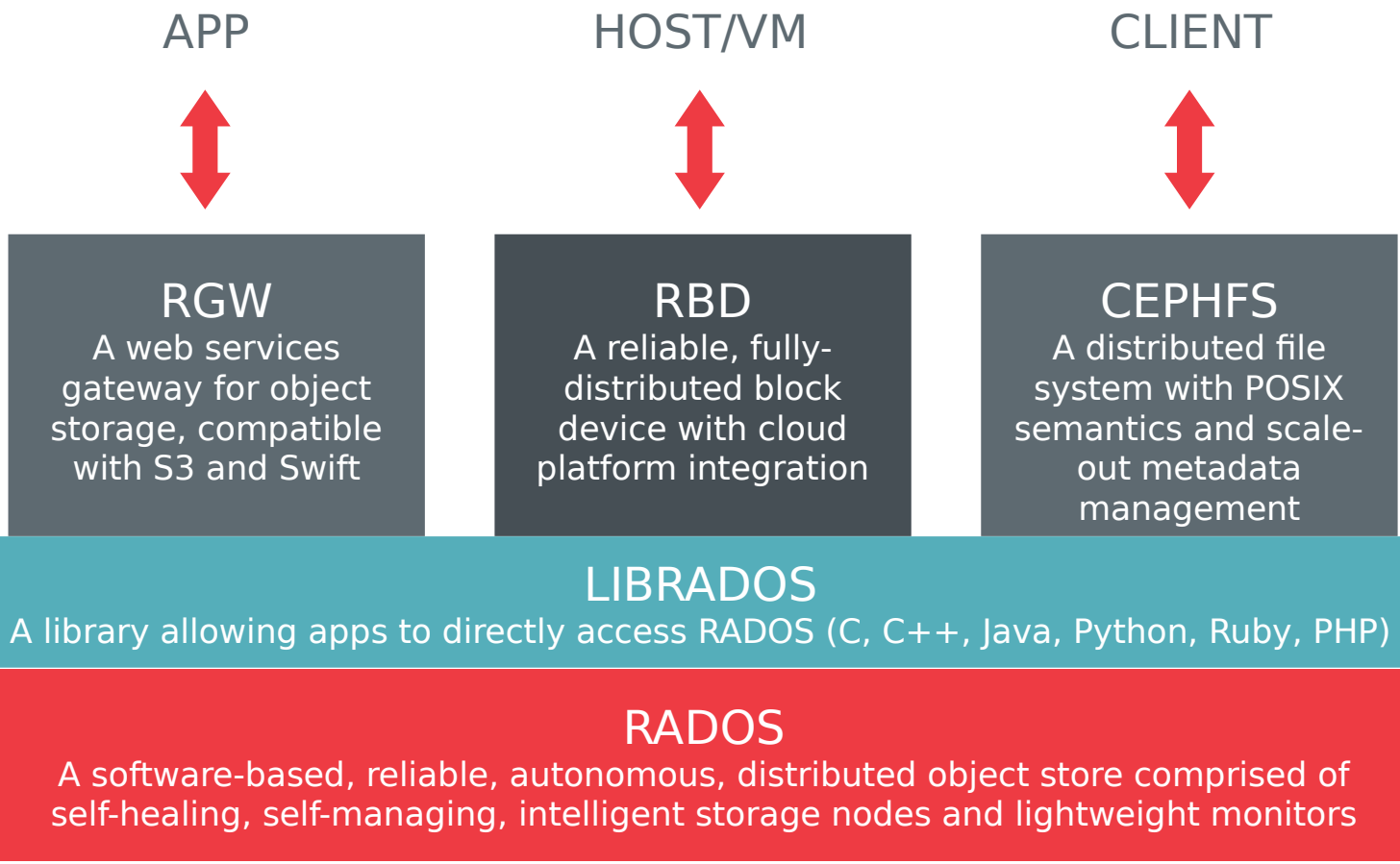
ARCHITECTURE

CEPH MOTIVATING PRINCIPLES

- All components must scale horizontally
- There can be no single point of failure
- The solution must be hardware agnostic
- Should use commodity hardware
- Self-manage whenever possible
- Open source (LGPL)

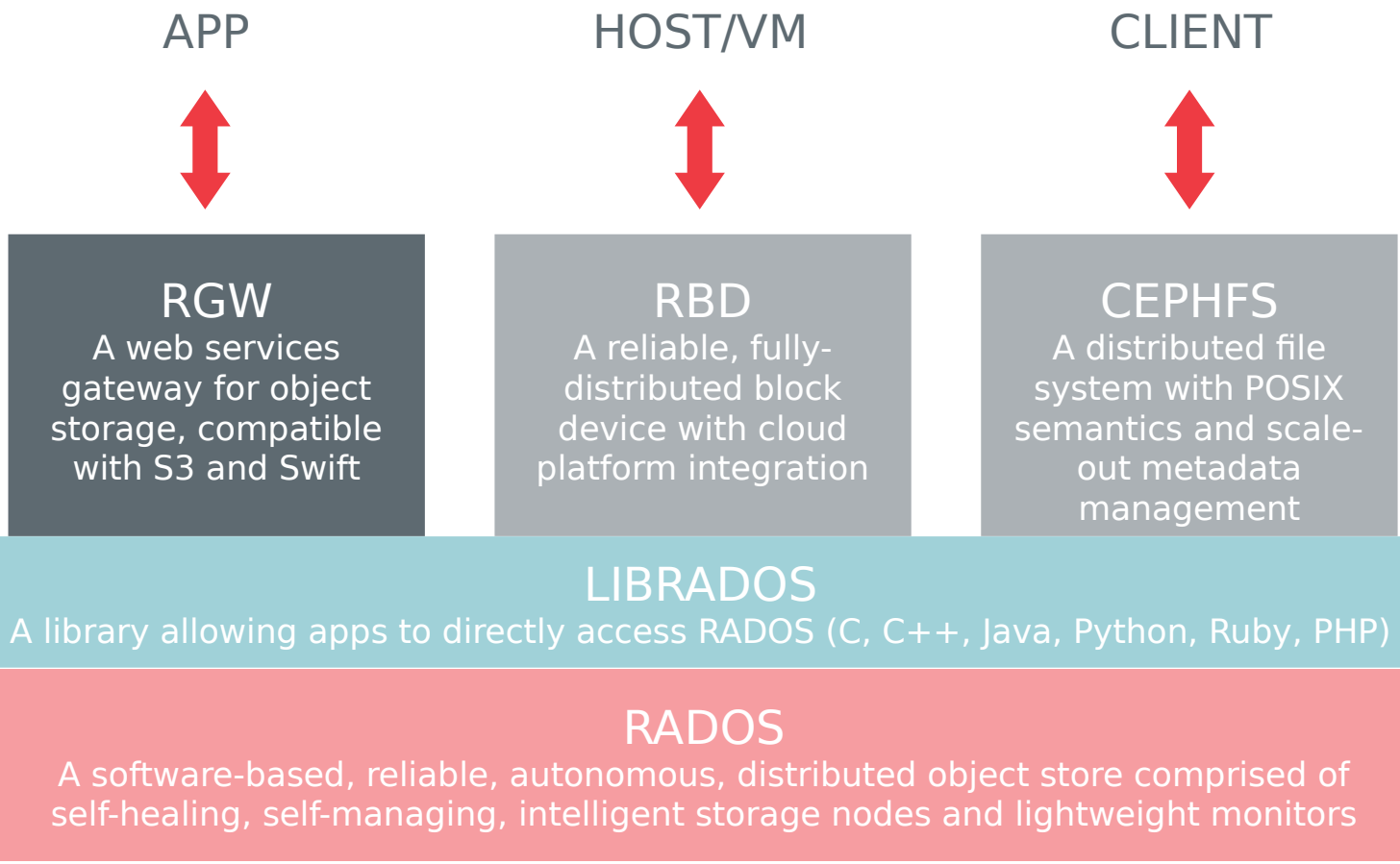
- Move beyond legacy approaches
  - Client/cluster instead of client/server
  - Ad hoc HA

CEPH COMPONENTS

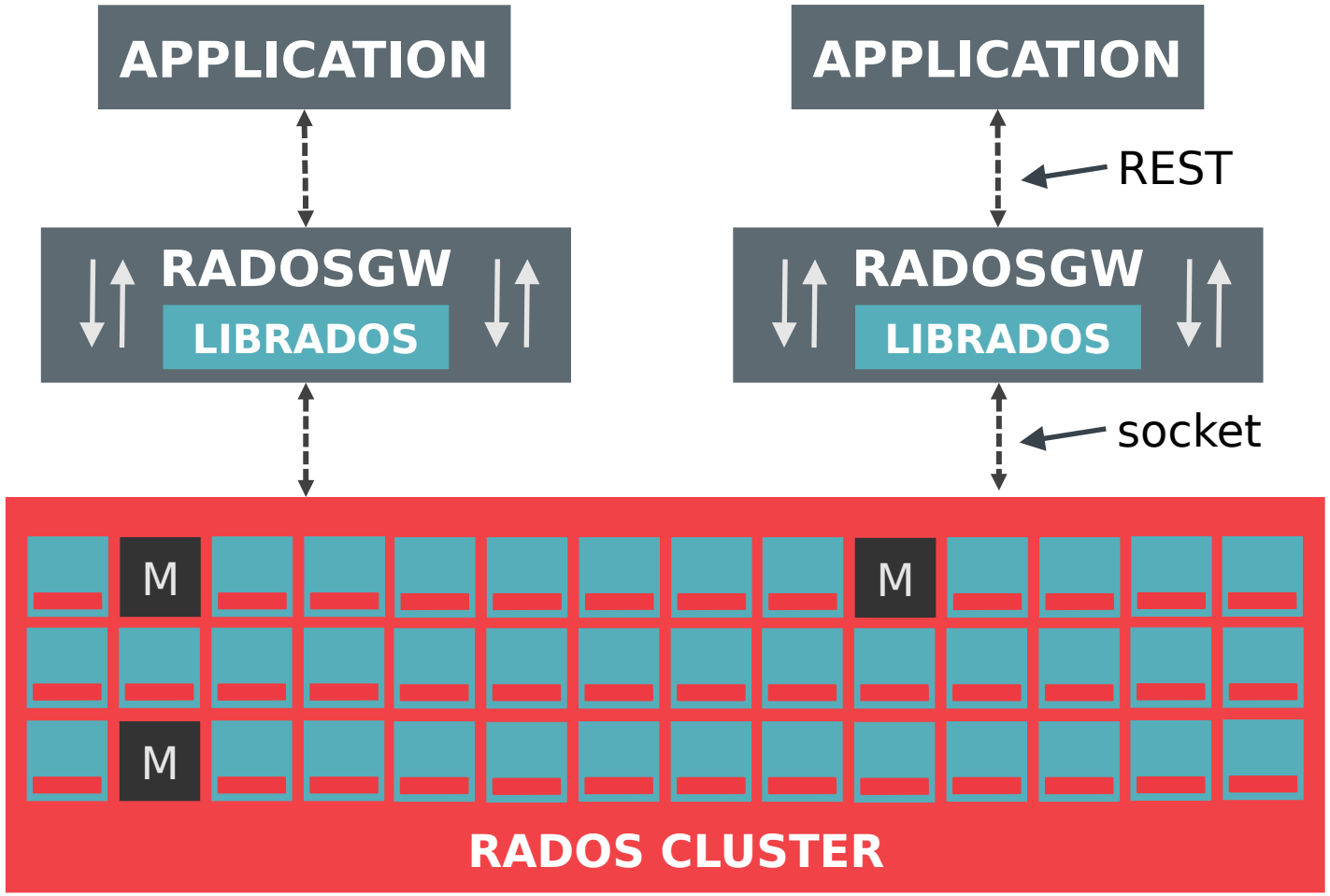


ROBUST SERVICES BUILT ON RADOS

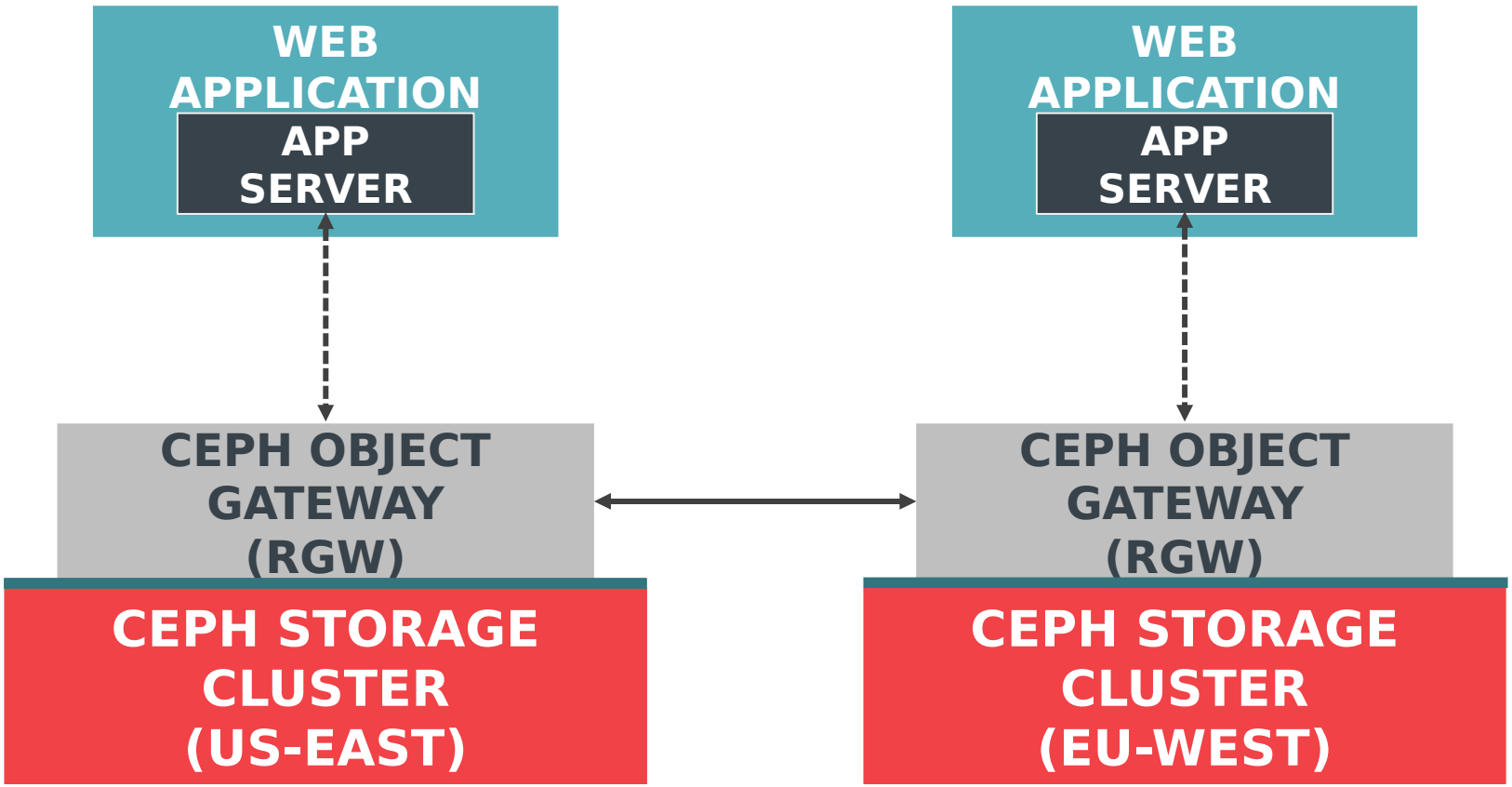
ARCHITECTURAL COMPONENTS



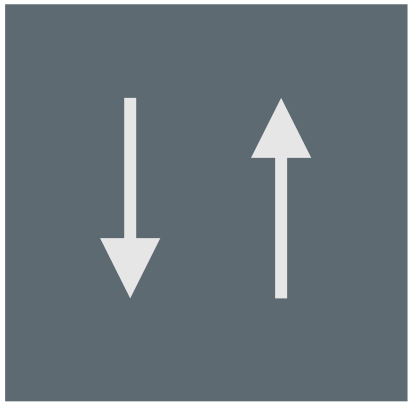
THE RADOS GATEWAY



MULTI-SITE OBJECT STORAGE



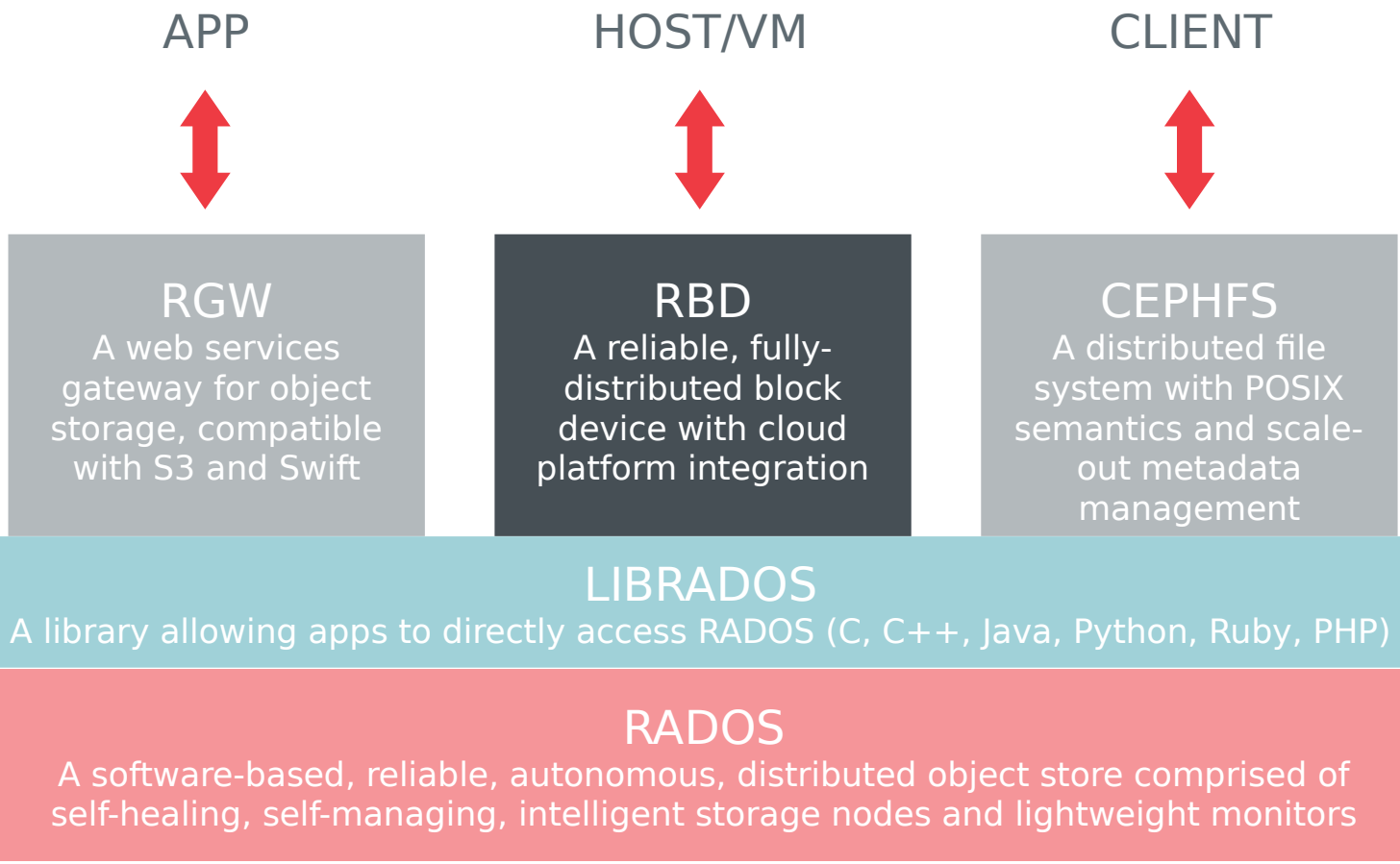
RADOSGW MAKES RADOS WEBBY



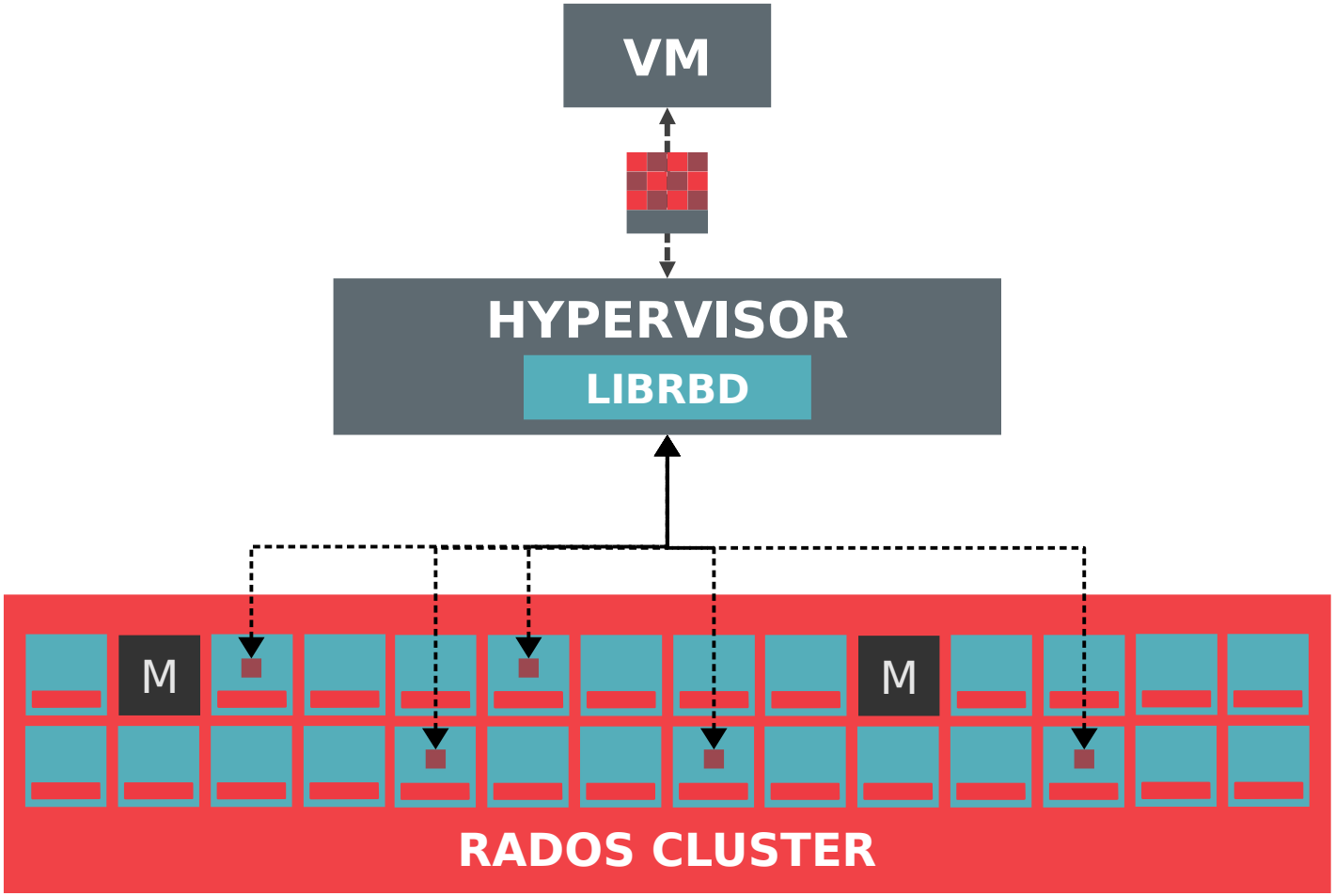
RADOSGW:

- REST-based object storage proxy
- Uses RADOS to store objects
  - Stripes large RESTful objects across many RADOS objects
- API supports buckets, accounts
- Usage accounting for billing
- Compatible with S3 and Swift applications

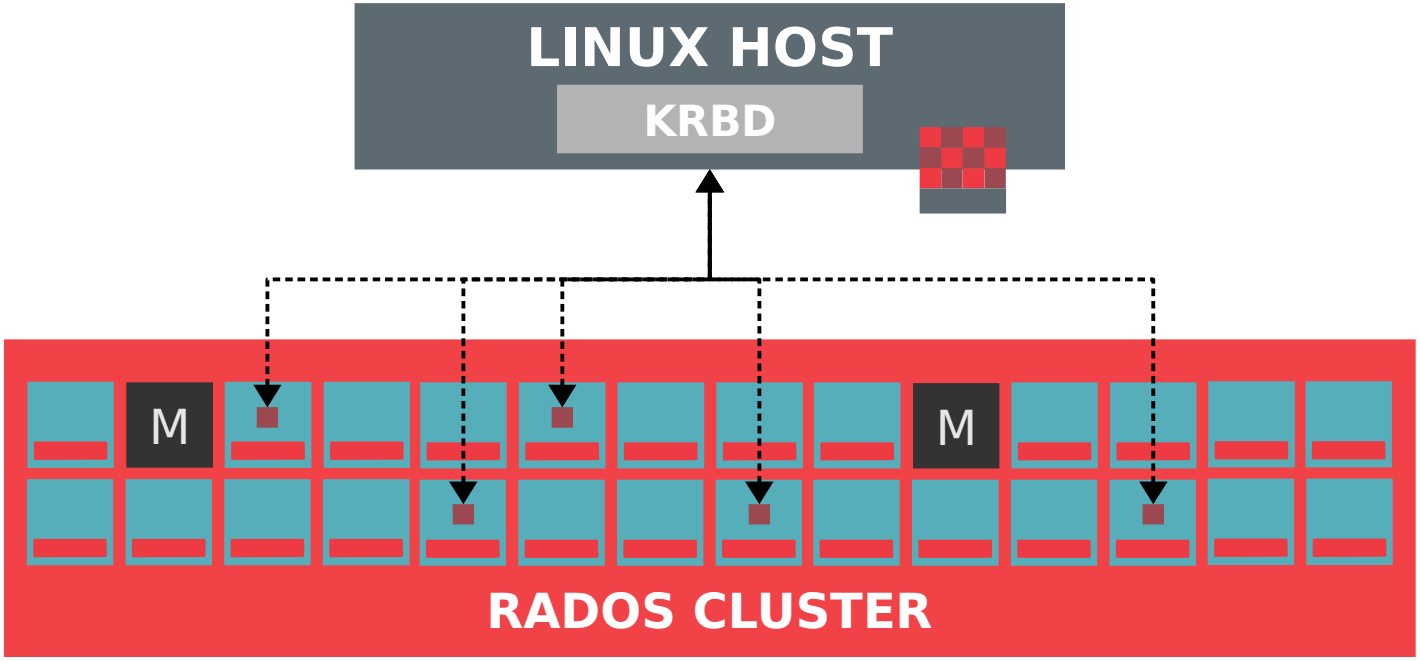
ARCHITECTURAL COMPONENTS



STORING VIRTUAL DISKS

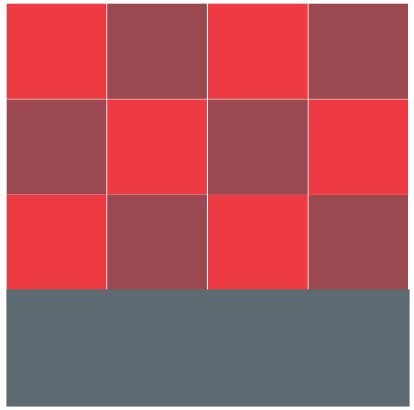


KERNEL MODULE

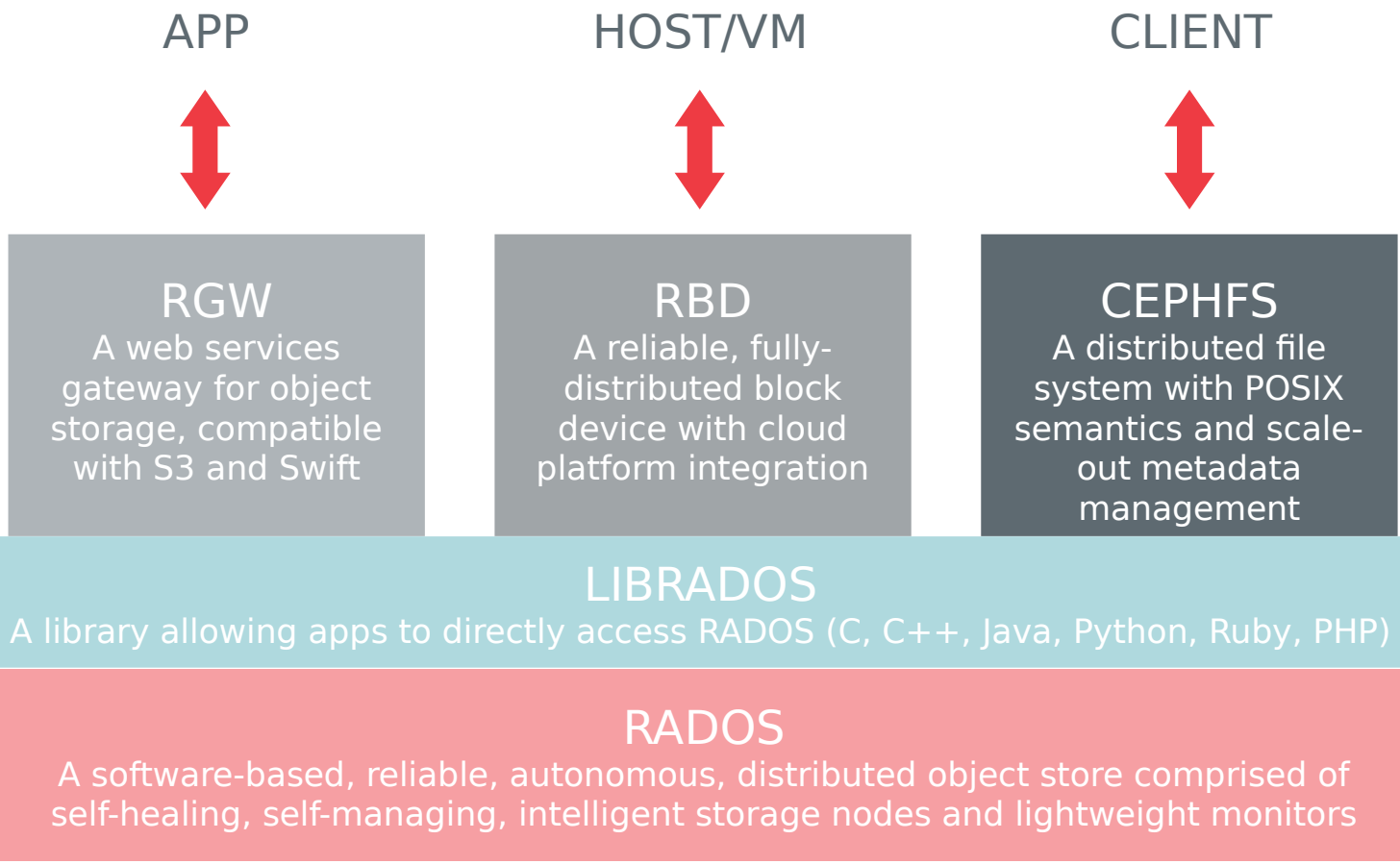


RBD FEATURES

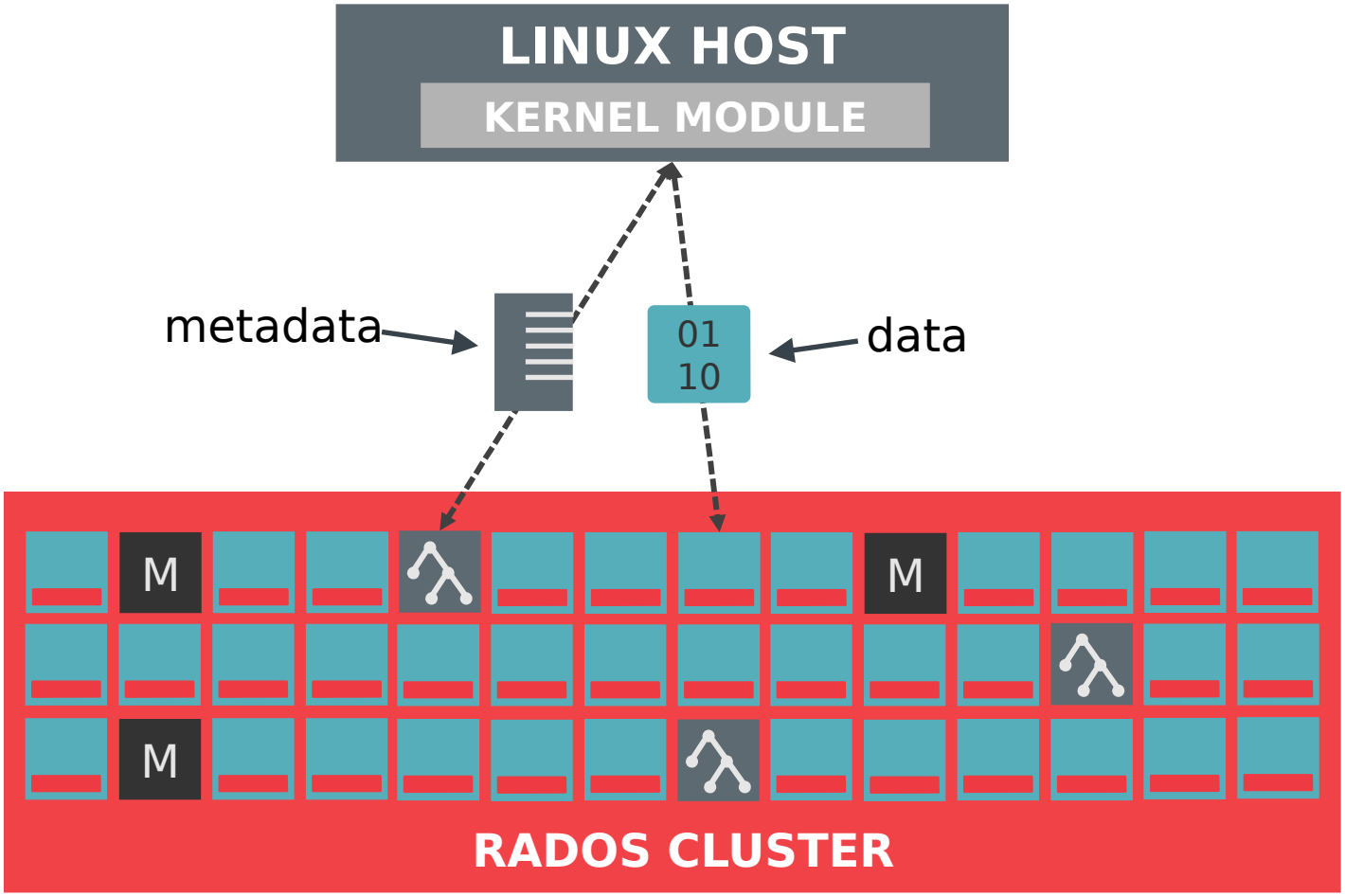
- Stripe images across entire cluster (pool)
- Read-only snapshots
- Copy-on-write clones
- Broad integration
  - Qemu
  - Linux kernel
  - iSCSI (STGT, LIO)
  - OpenStack, CloudStack, Nebula, Ganeti, Proxmox
- Incremental backup (relative to snapshots)



ARCHITECTURAL COMPONENTS



SEPARATE METADATA SERVER



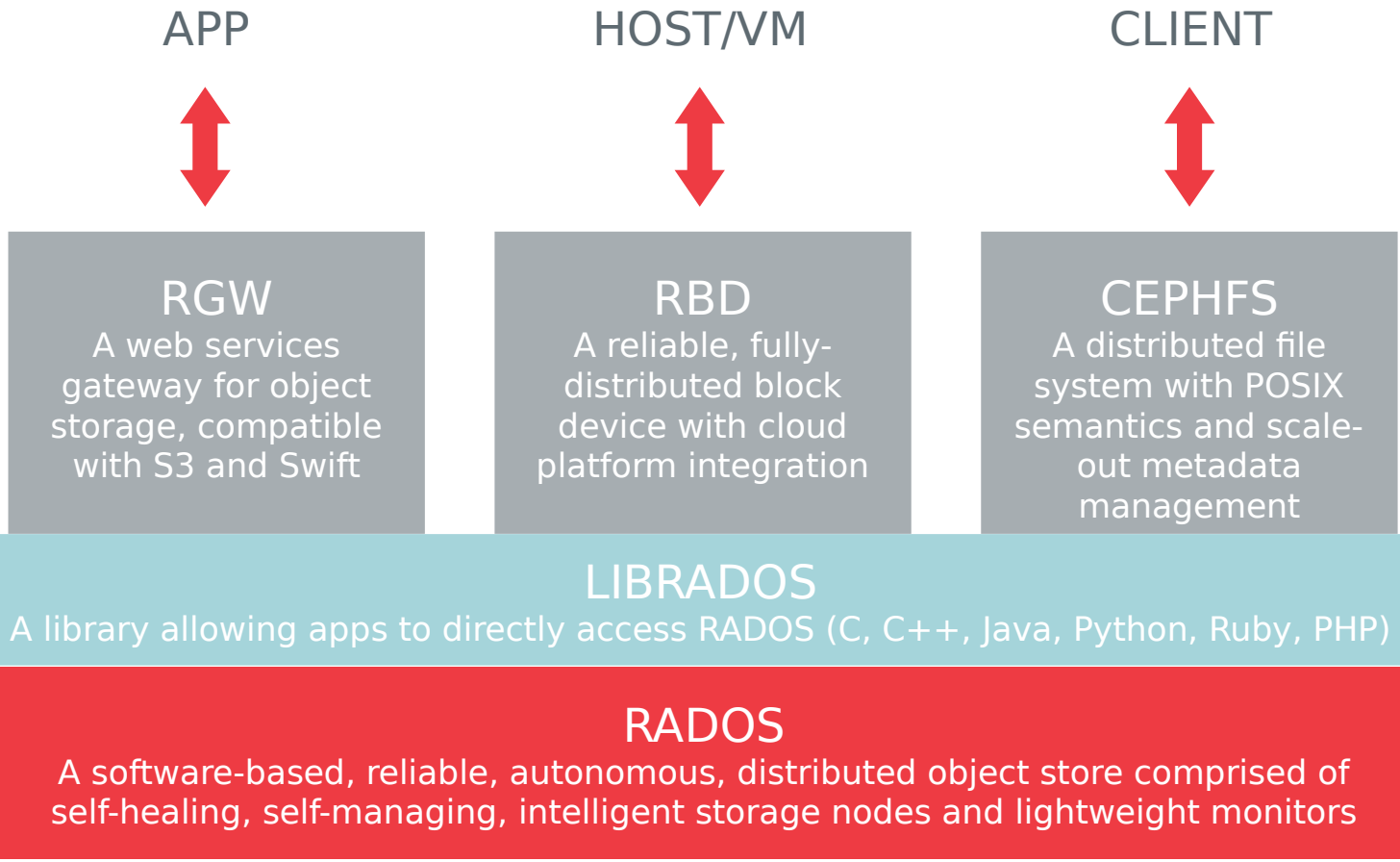
SCALABLE METADATA SERVERS

METADATA SERVER

- Manages metadata for a POSIX-compliant shared filesystem
  - Directory hierarchy
  - File metadata (owner, timestamps, mode, etc.)
  - Snapshots on any directory
- Clients stripe file data in RADOS
  - MDS not in data path
- MDS stores metadata in RADOS
- **Dynamic MDS cluster** scales to 10s or 100s
- Only required for shared filesystem

RADOS

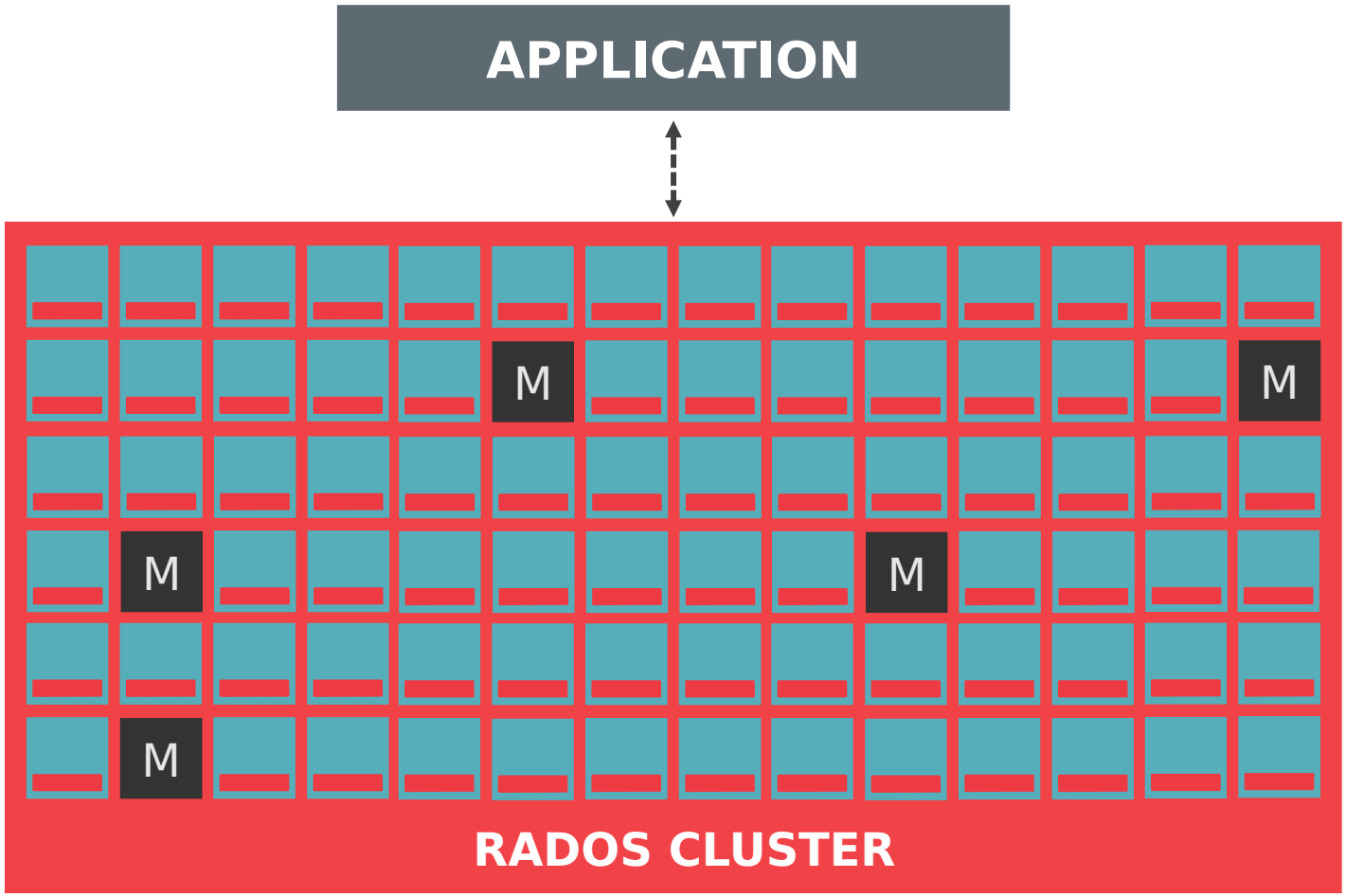
ARCHITECTURAL COMPONENTS



RADOS

- Flat object namespace within each pool
- Rich object API (librados)
  - Bytes, attributes, key/value data
  - Partial overwrite of existing data (**mutable objects**)
  - Single-object compound operations
  - RADOS classes (stored procedures)
- **Strong consistency** (CP system)
- Infrastructure aware, dynamic topology
- Hash-based placement (CRUSH)
- Direct client to server data path

RADOS CLUSTER



RADOS COMPONENTS

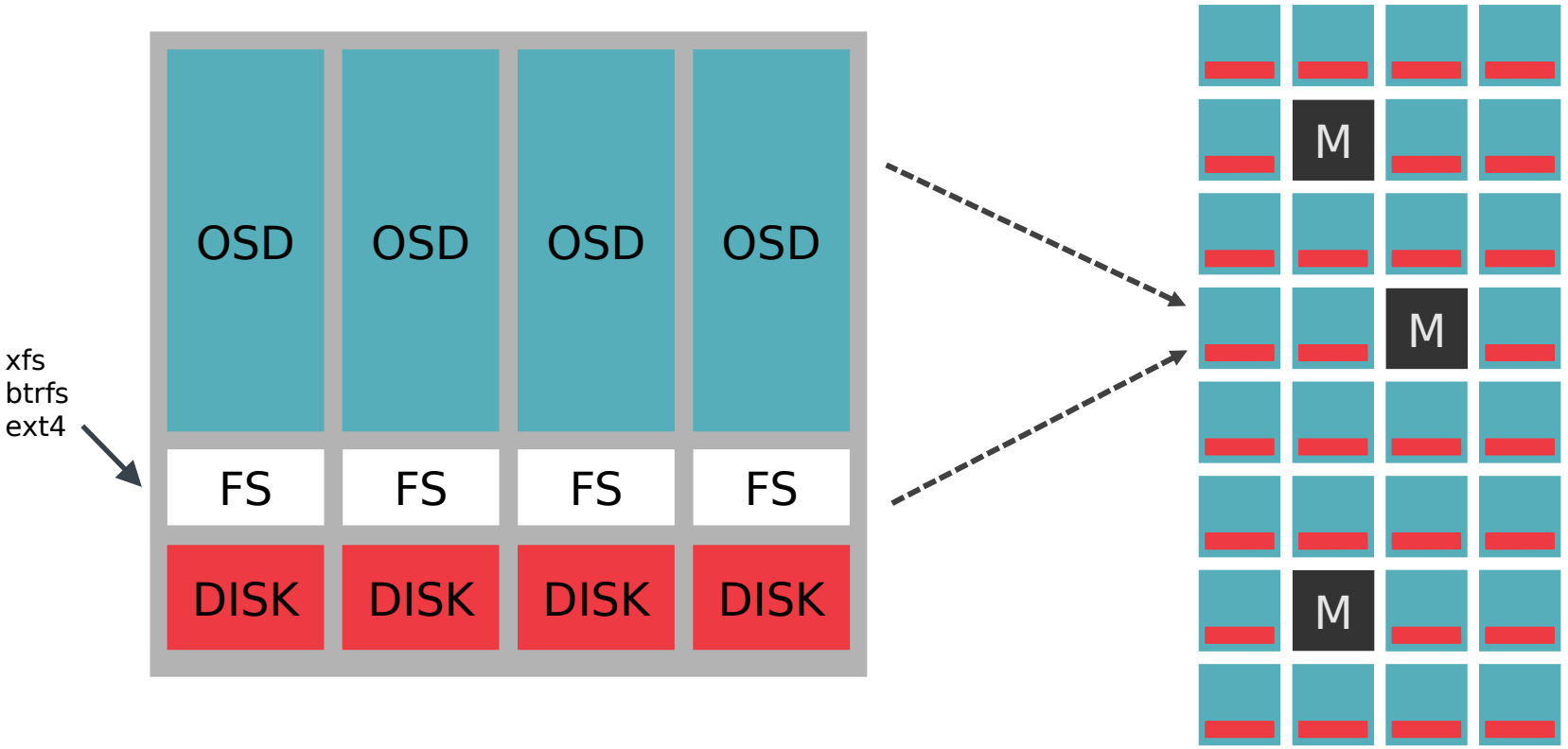
OSDs:

- 10s to 1000s in a cluster
- One per disk (or one per SSD, RAID group…)
- Serve stored objects to clients
- Intelligently peer for replication & recovery

Monitors:

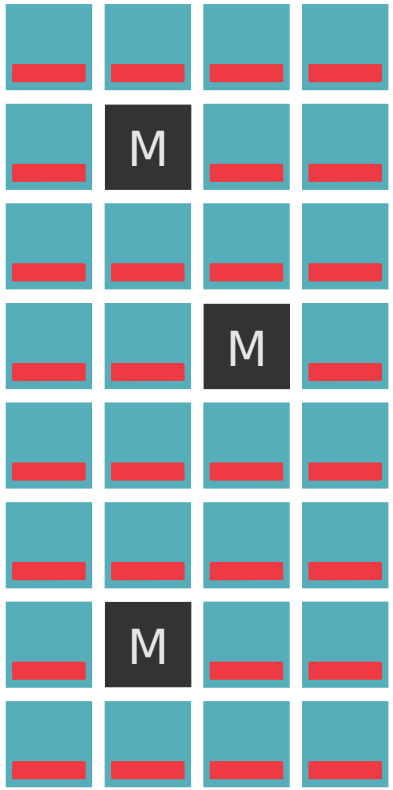
- Maintain cluster membership and state
- Provide consensus for distributed decision-making
- Small, odd number (e.g., 5)
- Not part of data path

OBJECT STORAGE DAEMONS

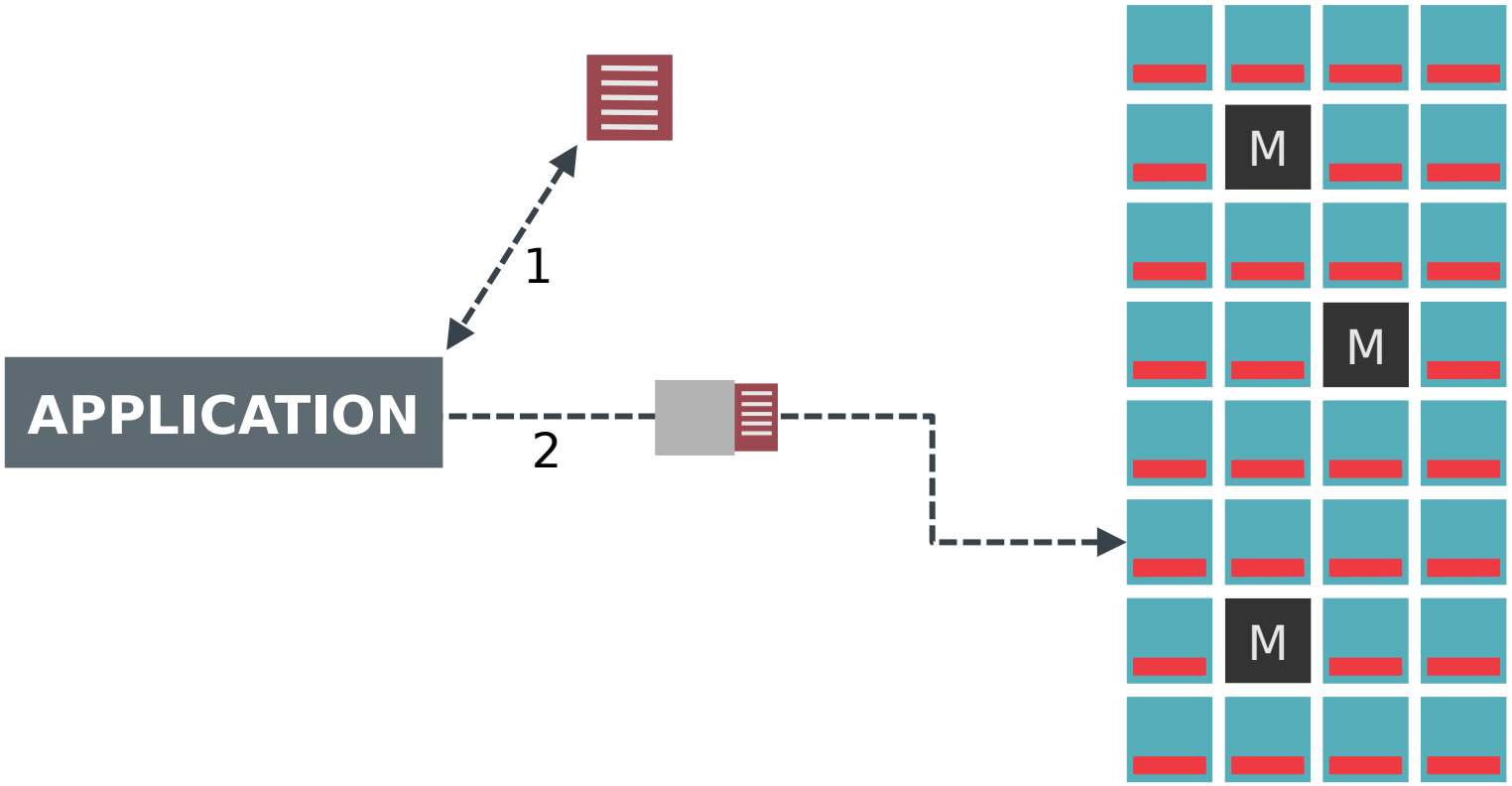


DATA PLACEMENT

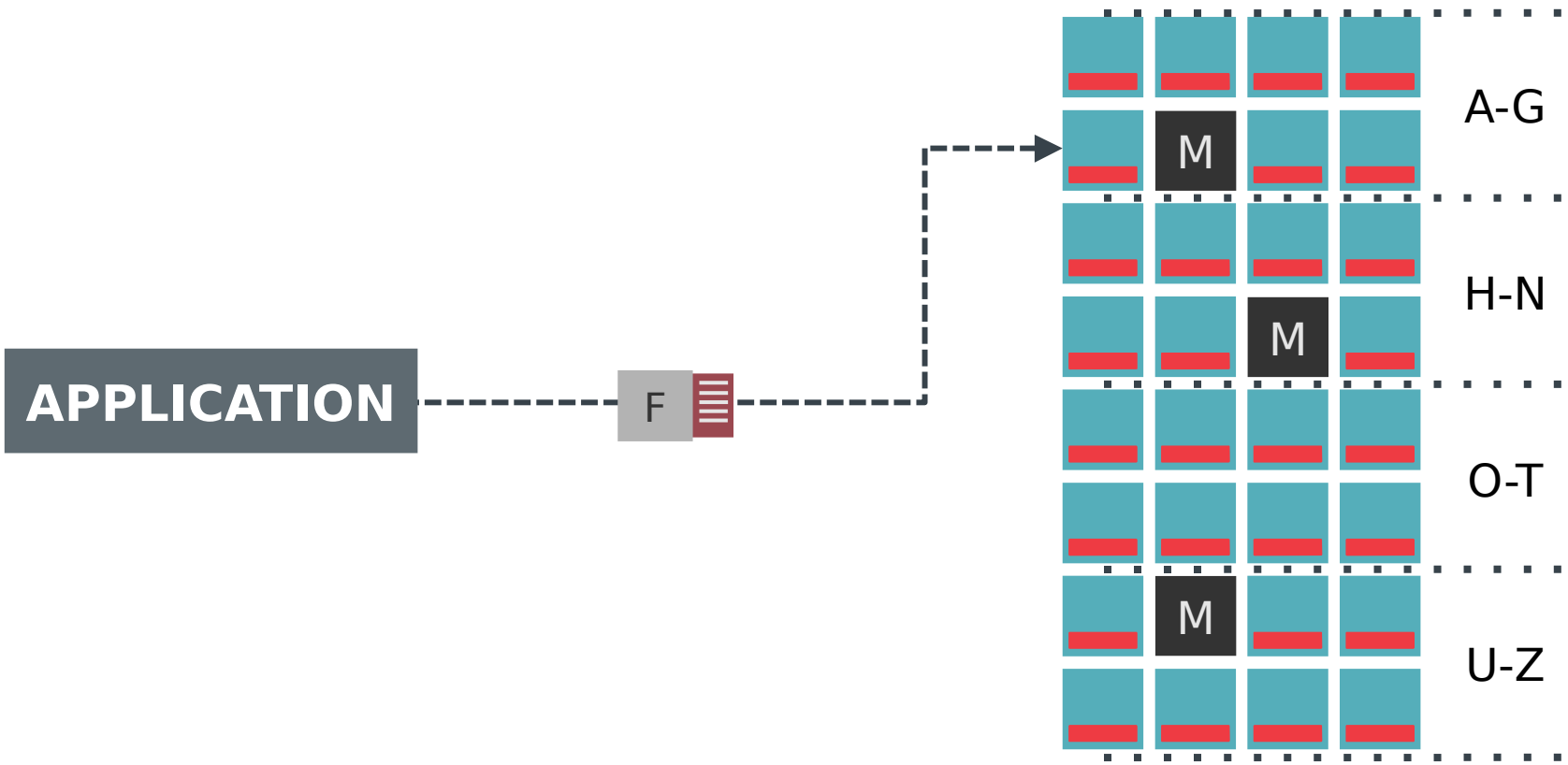
WHERE DO OBJECTS LIVE?



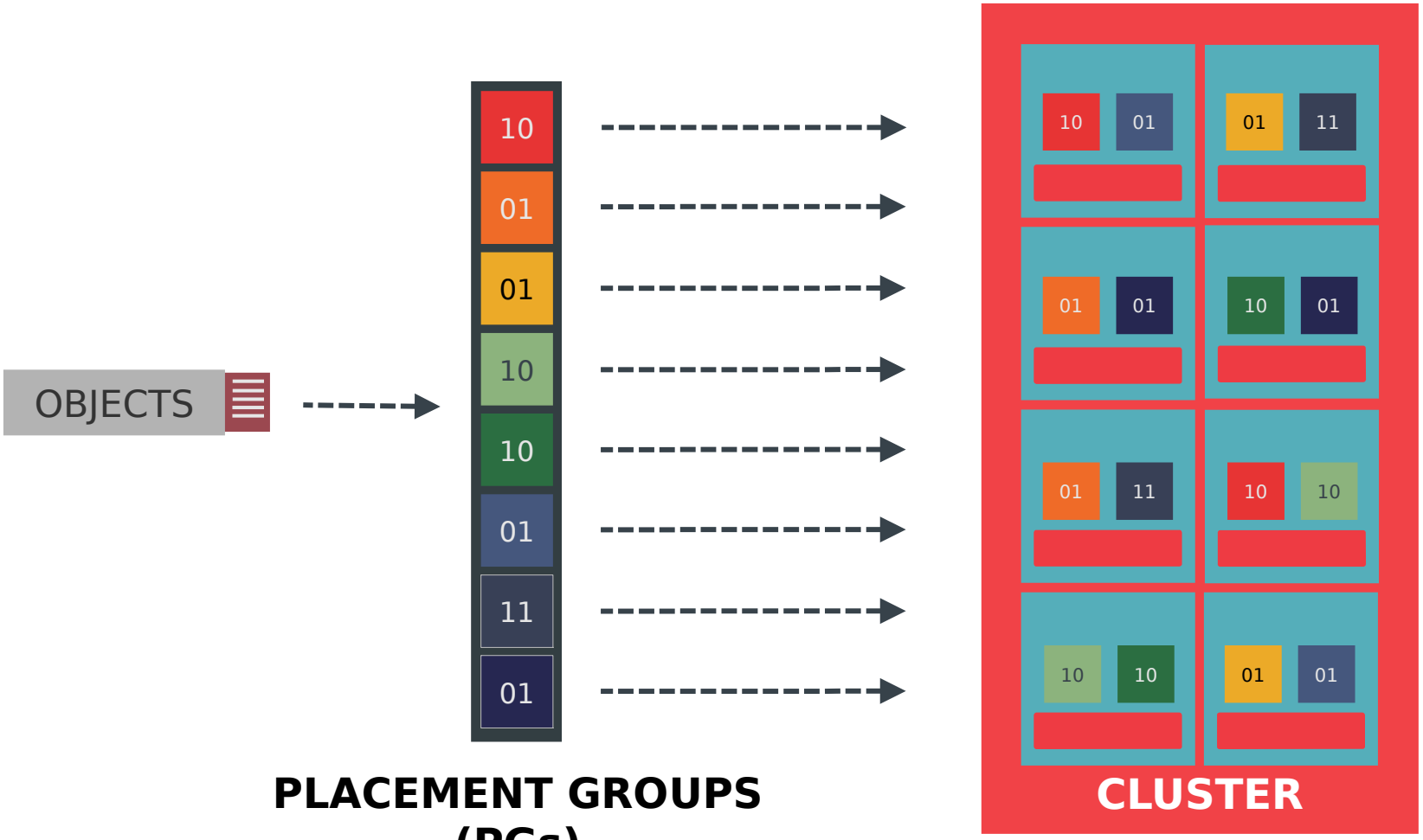
A METADATA SERVER?



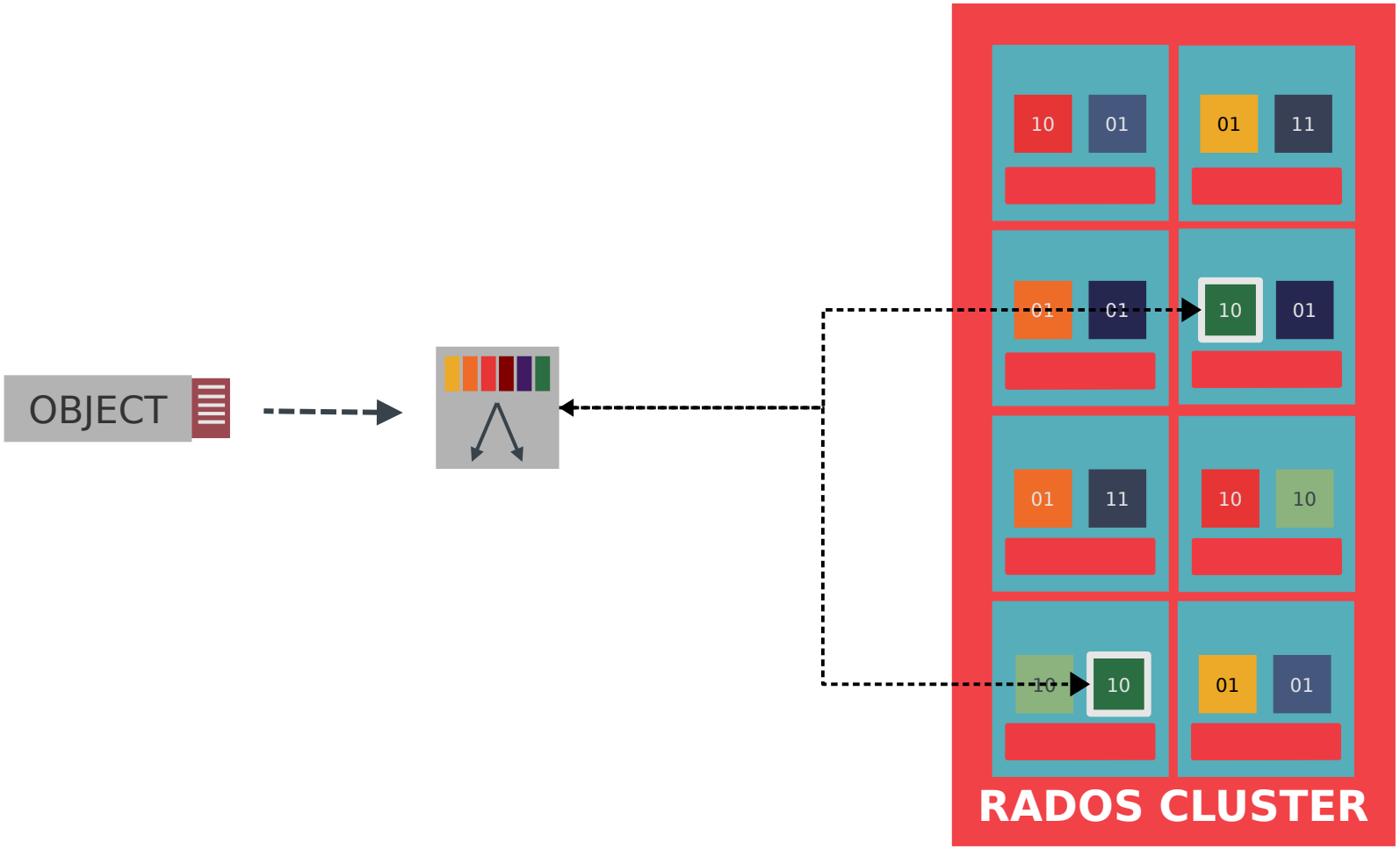
CALCULATED PLACEMENT



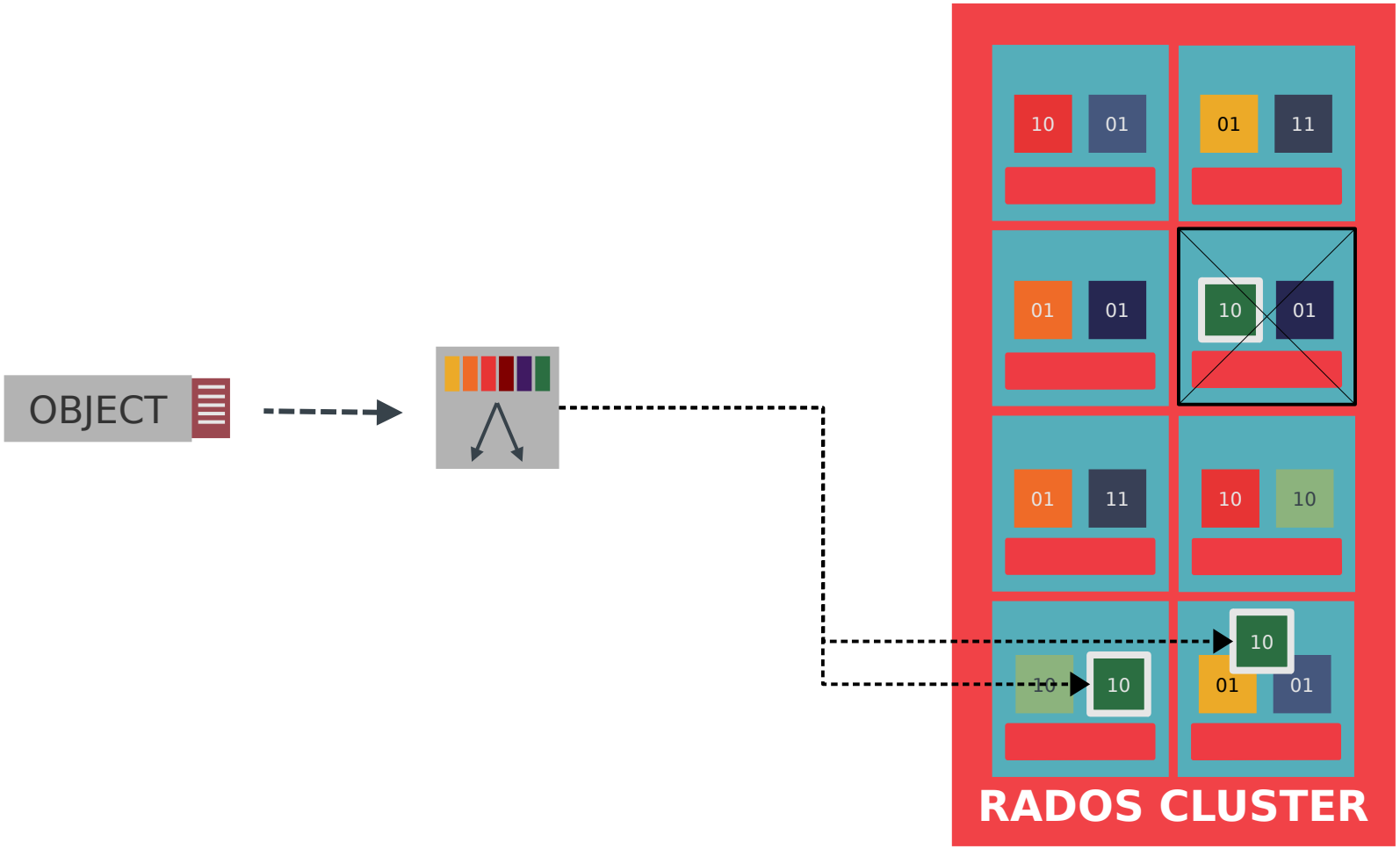
CRUSH


OBJECTS
10
01
01
10
10
01
11
01
PLACEMENT GROUPS
(PGs)
10 01
01 11
01 01
10 01
01 11
10 10
10 10
01 01
CLUSTER

CRUSH IS A QUICK CALCULATION

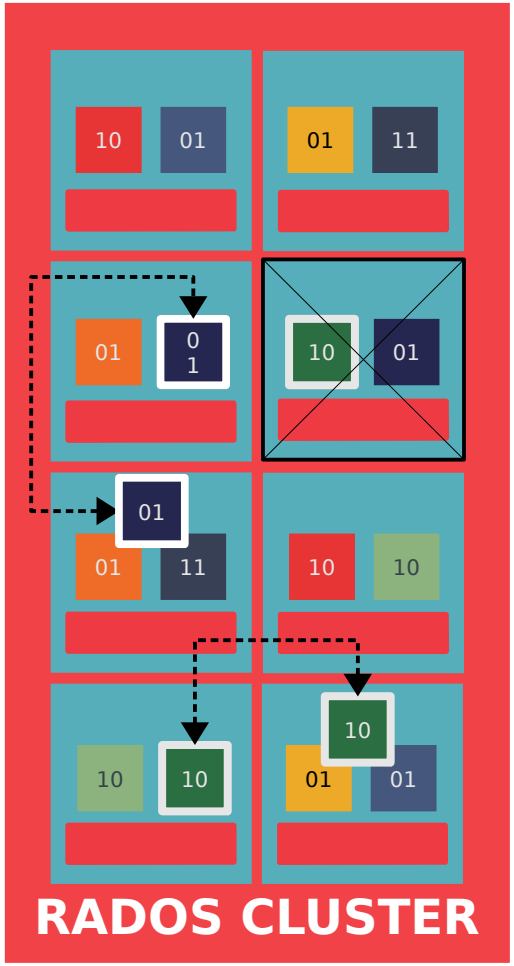


CRUSH AVOIDS FAILED DEVICES

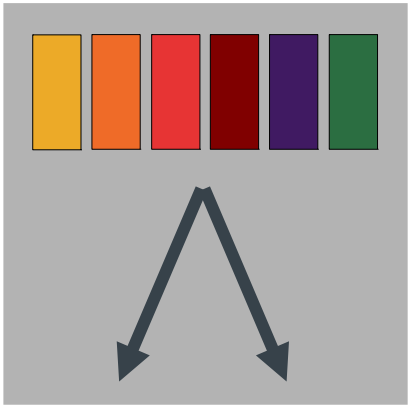


CRUSH: DECLUSTERED PLACEMENT

- Each PG independently maps to a pseudorandom set of OSDs
- PGs that map to the same OSD generally have replicas that do not
- When an OSD fails, each PG it stored will generally be re-replicated by a different OSD
  - Highly **parallel recovery**
  - Avoid single-disk recovery bottleneck



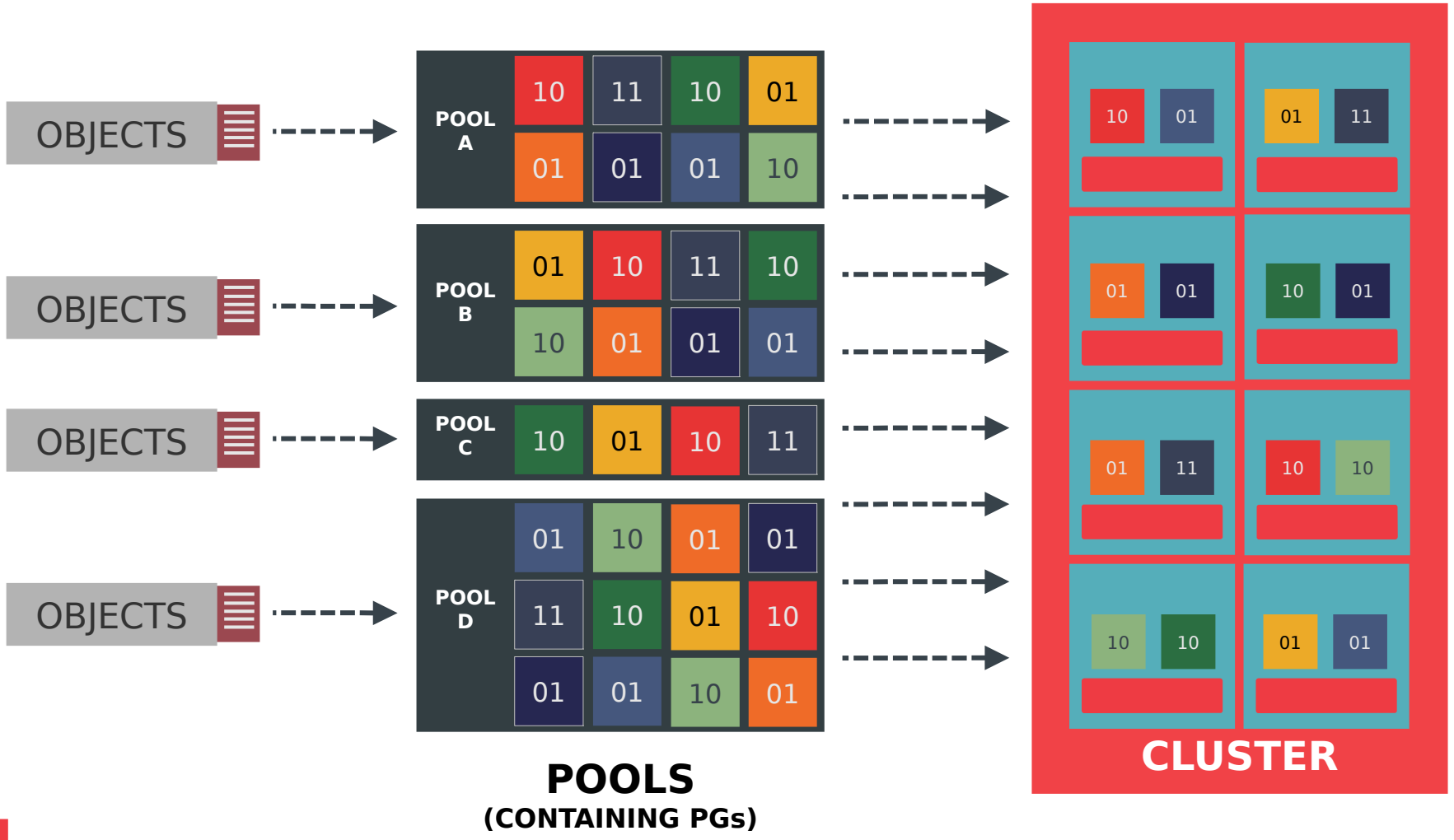
CRUSH: DYNAMIC DATA PLACEMENT



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
  - Weighted devices (different sizes)

DATA IS ORGANIZED INTO POOLS

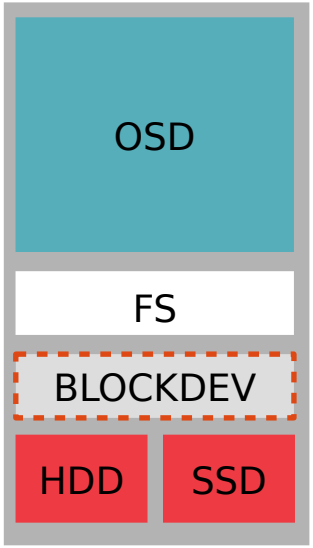




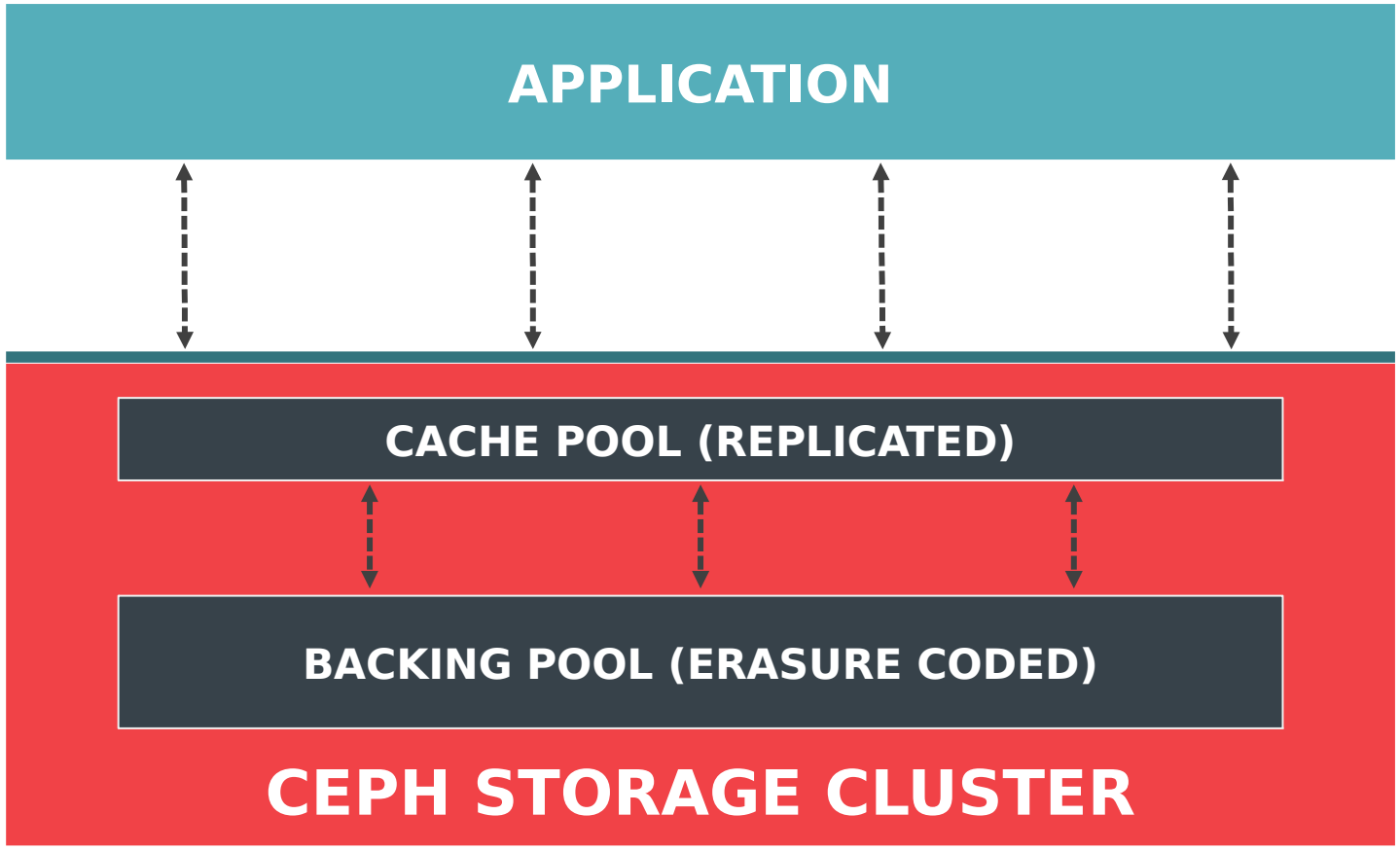
TIERED STORAGE

TWO WAYS TO CACHE

- Within each OSD
  - Combine SSD and HDD under each OSD
  - Make **localized** promote/demote decisions
  - Leverage existing tools
    - dm-cache, bcache, FlashCache
    - Variety of caching controllers
  - We can help with hints
- Cache on separate devices/nodes
  - Different hardware for different tiers
    - Slow nodes for cold data
    - High performance nodes for hot data
  - Add, remove, **scale each tier independently**
    - Unlikely to choose right ratios at procurement time



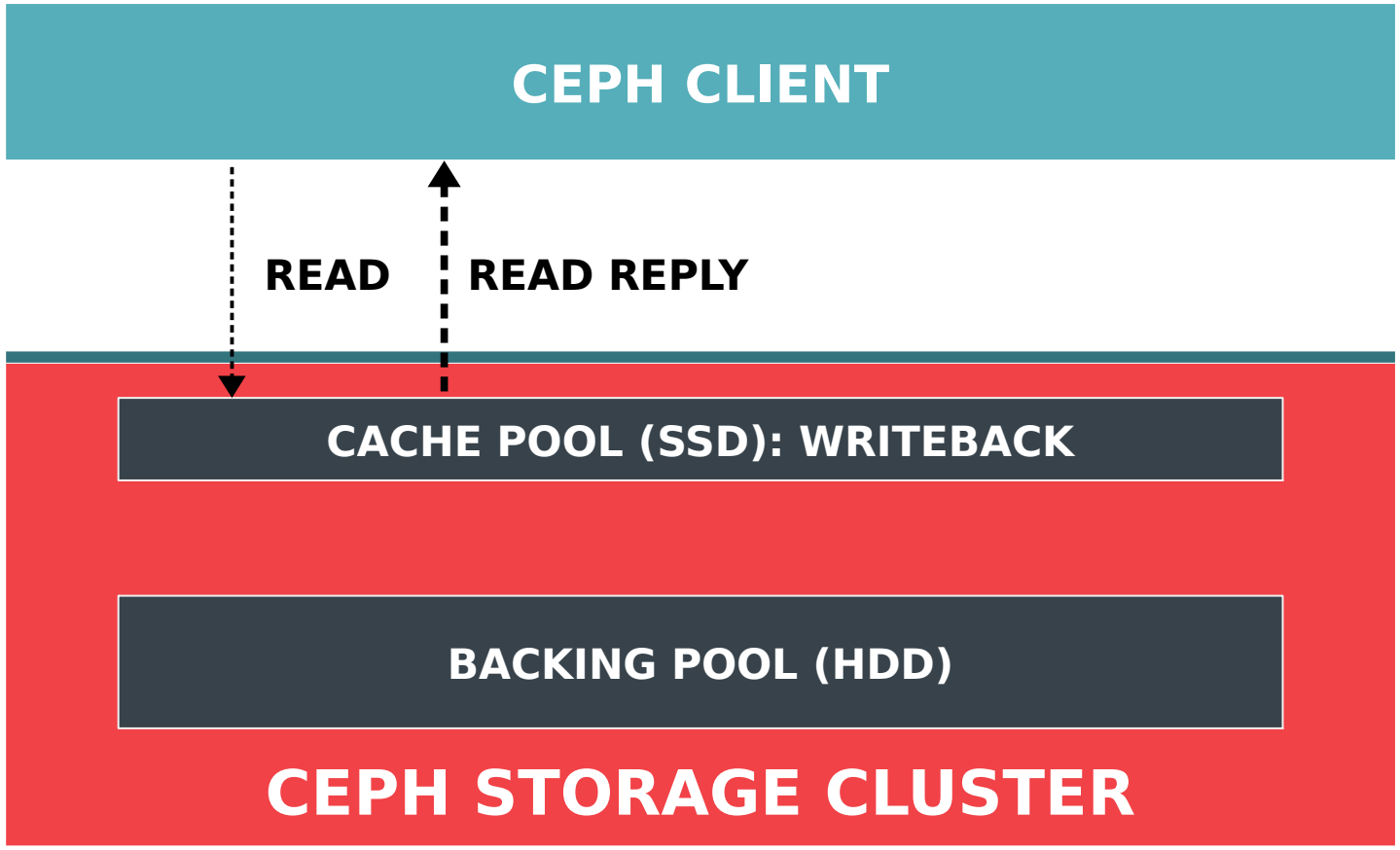
TIERED STORAGE



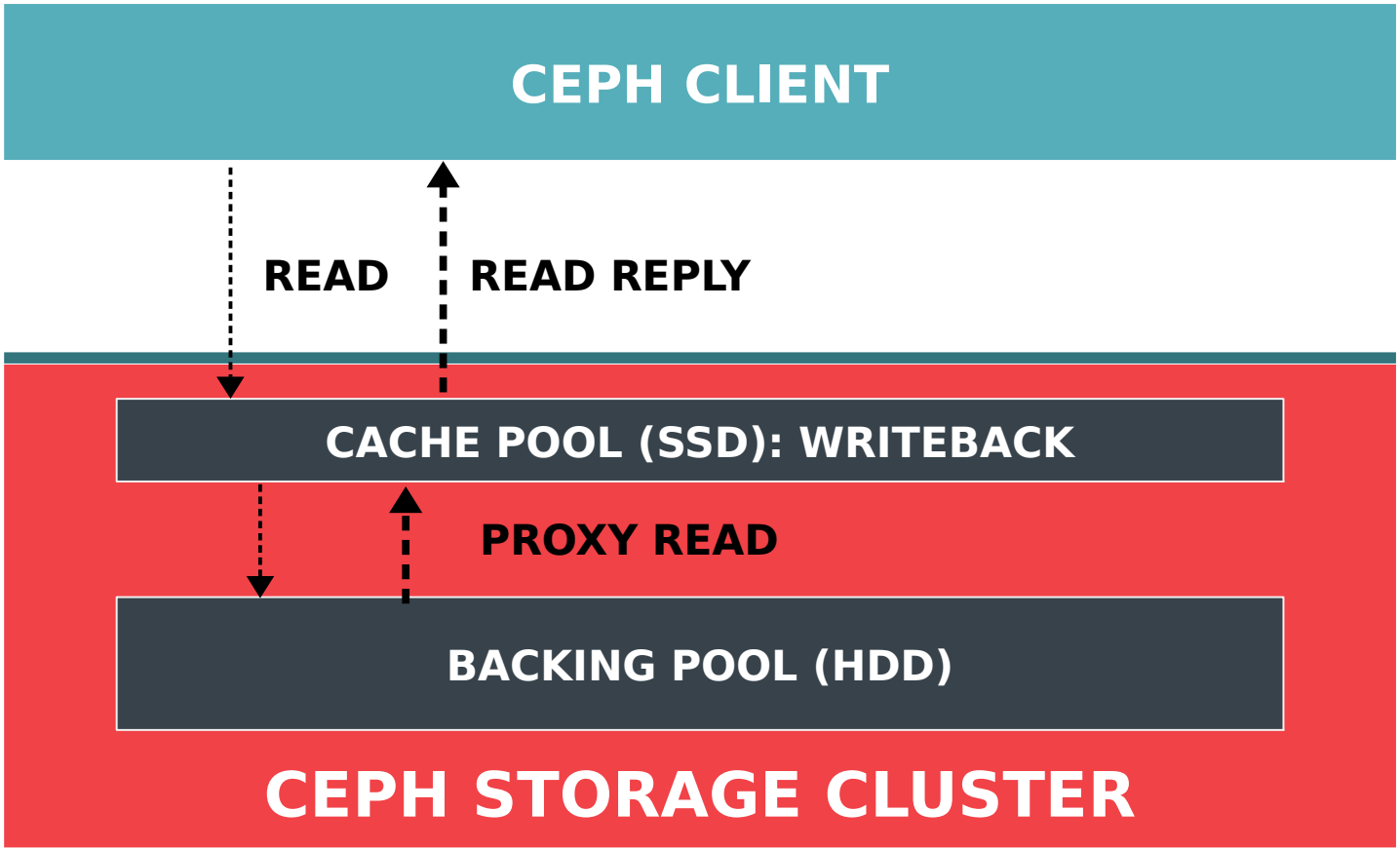
RADOS TIERING PRINCIPLES

- Each tier is a RADOS pool
  - May be replicated or erasure coded
- Tiers are **durable**
  - e.g., replicate across SSDs in multiple hosts
- Each tier has its own CRUSH policy
  - e.g., map cache pool to SSDs devices/hosts only
- librados adapts to tiering topology
  - Transparently direct requests accordingly
    - e.g., to cache
  - **No changes** to RBD, RGW, CephFS, etc.

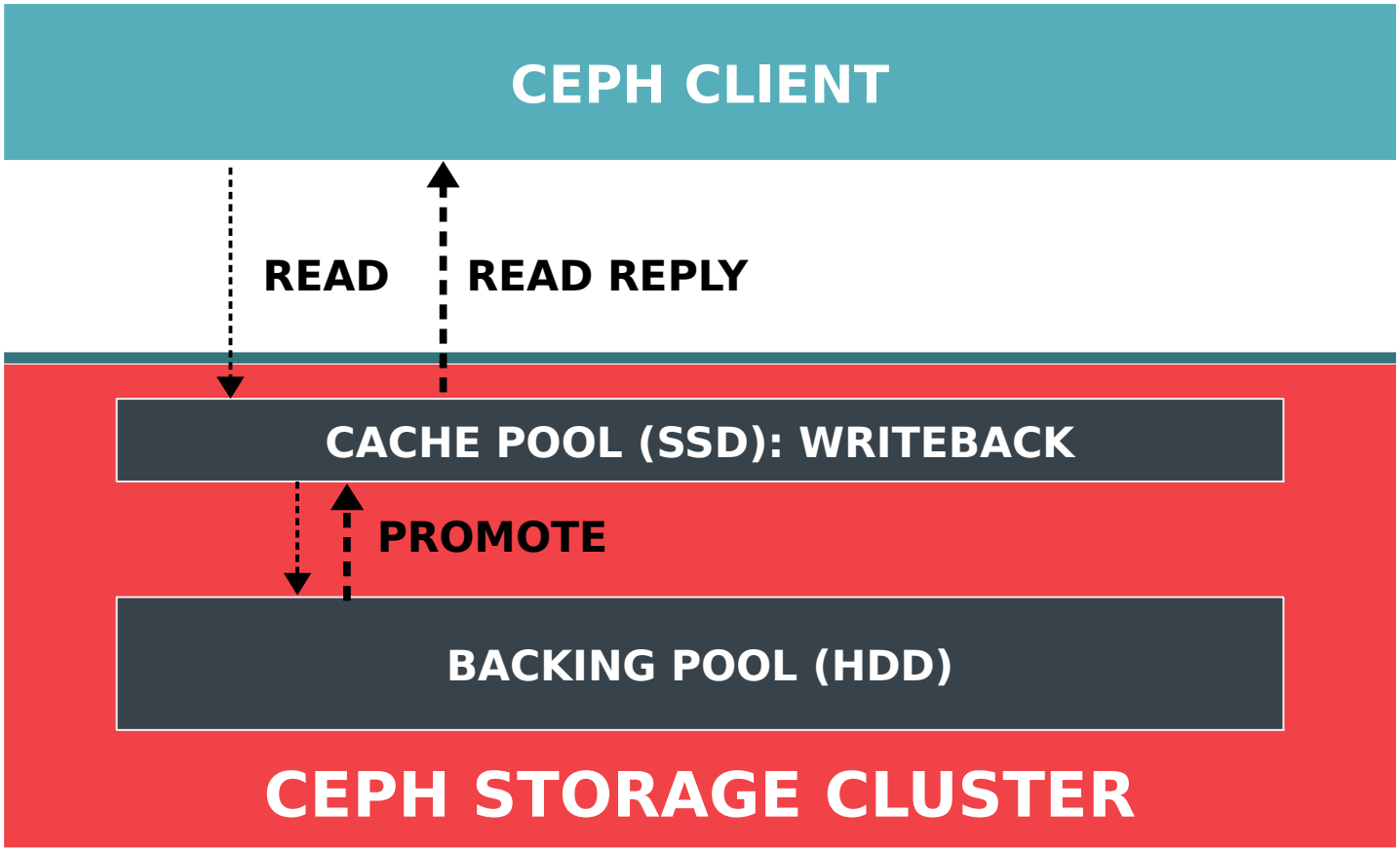
READ (CACHE HIT)



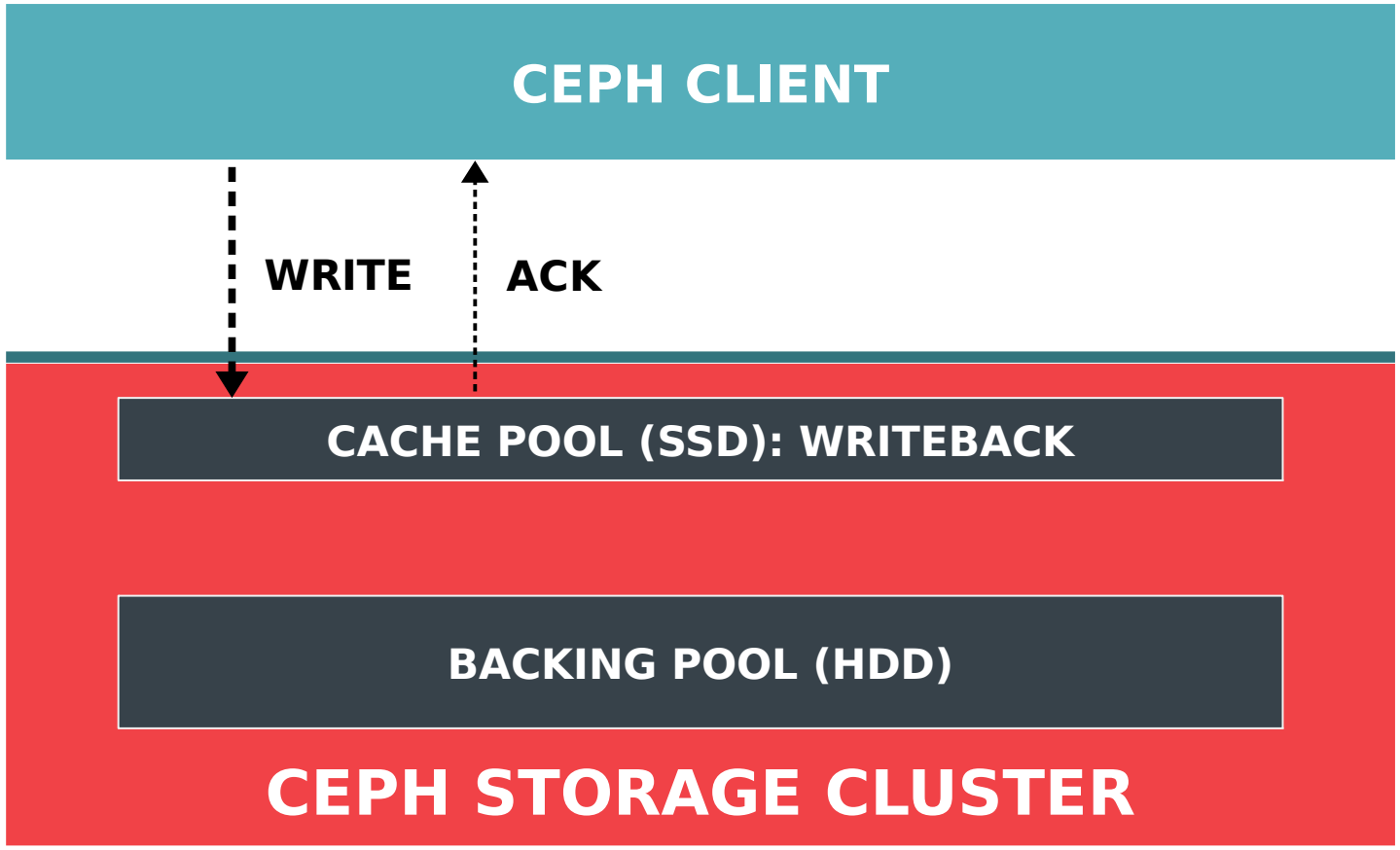
READ (CACHE MISS)



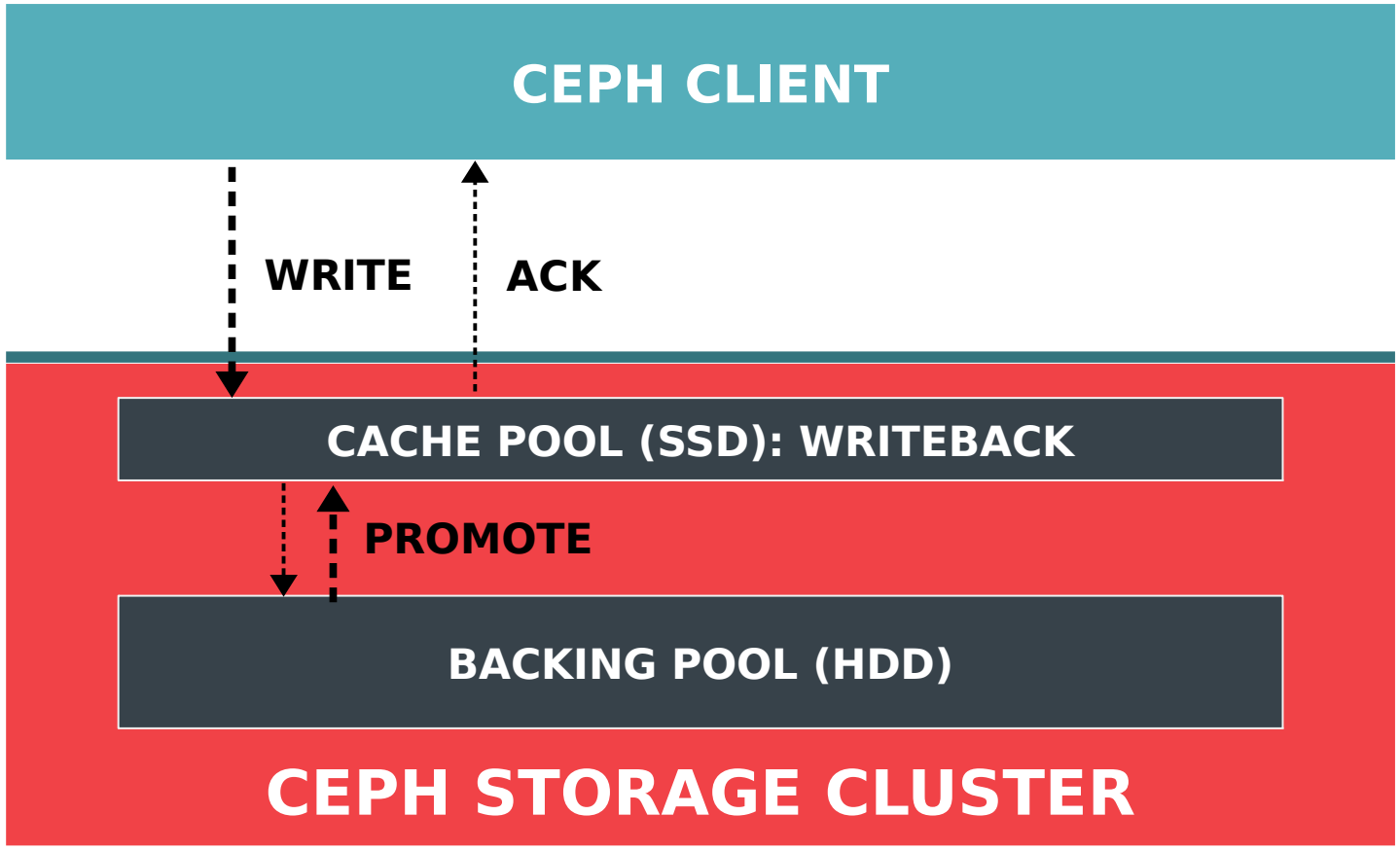
READ (CACHE MISS)



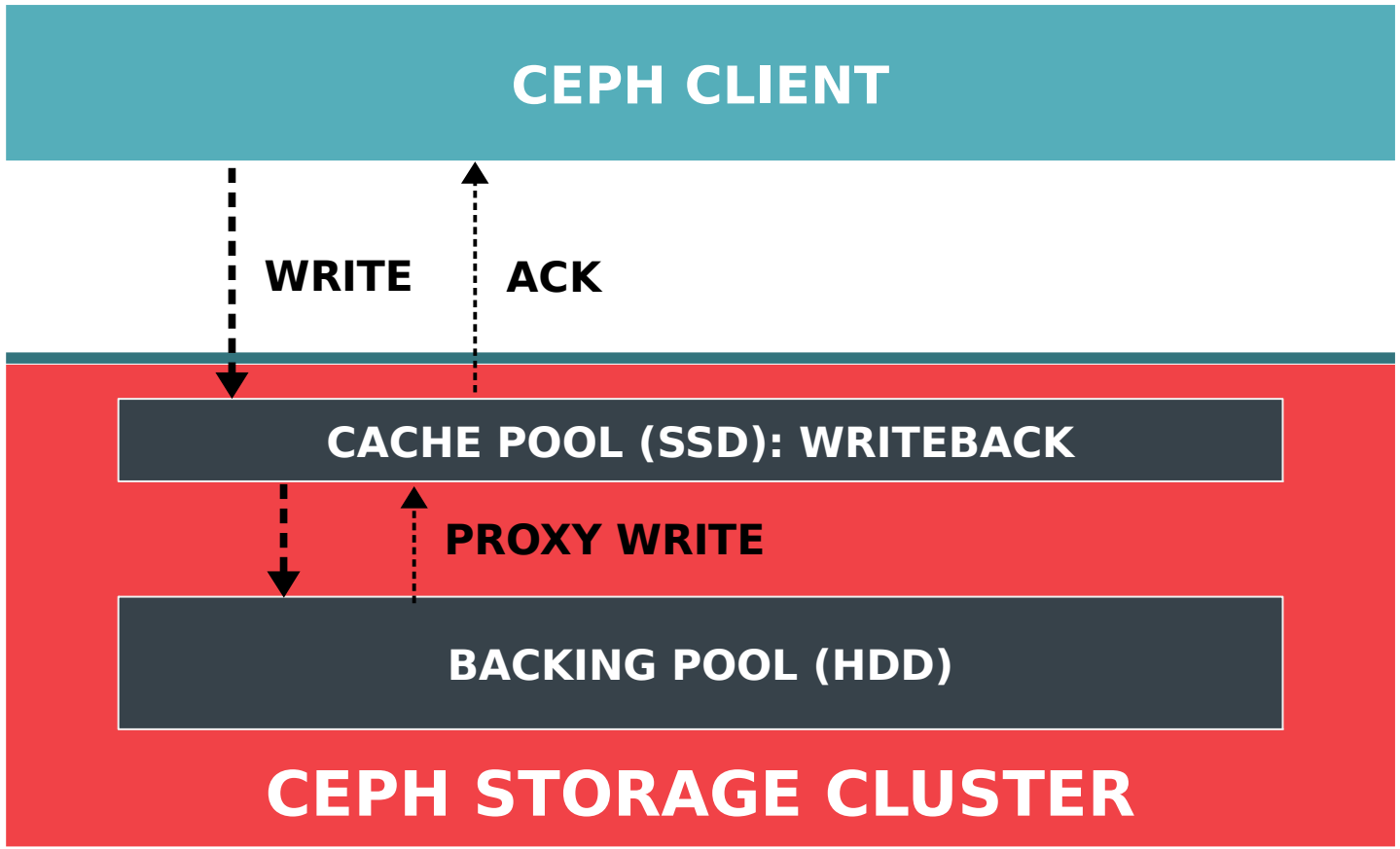
WRITE (HIT)


CEPH CLIENT
WRITE
ACK
CACHE POOL (SSD): WRITEBACK
BACKING POOL (HDD)
CEPH STORAGE CLUSTER

WRITE (MISS)


CEPH CLIENT
WRITE
ACK
CACHE POOL (SSD): WRITEBACK
PROMOTE
BACKING POOL (HDD)
CEPH STORAGE CLUSTER

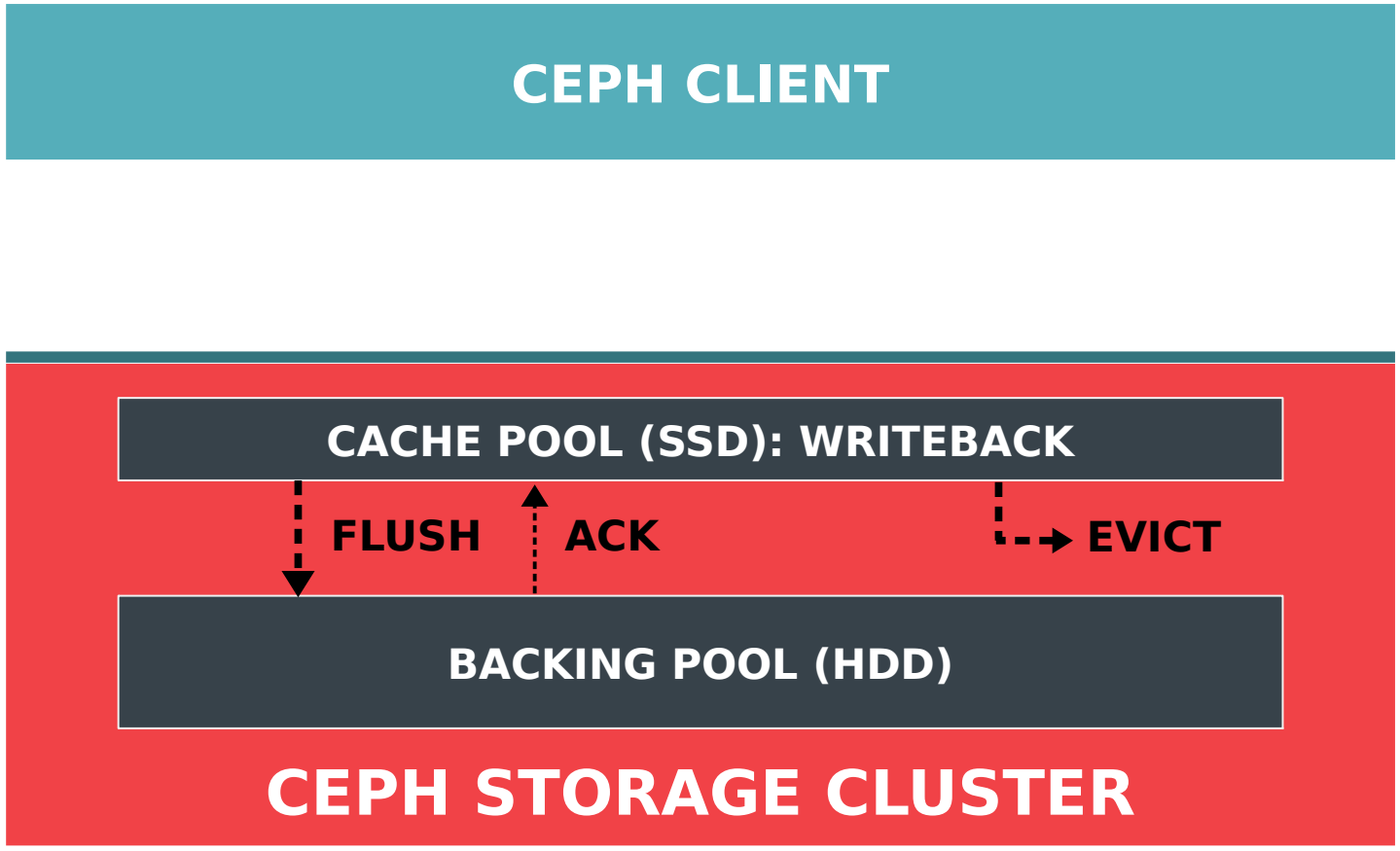
WRITE (MISS) (COMING SOON)



ESTIMATING TEMPERATURE

- Each PG constructs in-memory bloom filters
  - Insert records on both read and write
  - Each filter covers **configurable period** (e.g., 1 hour)
  - Tunable false positive probability (e.g., 5%)
  - Store most recent N periods on disk (e.g., last 24 hours)
- Estimate **temperature**
  - Has object been accessed in any of the last N periods?
  - ...in how many of them?
  - Informs the flush/evict decision
- Estimate "**recency**"
  - How many periods since the object hasn't been accessed?
  - Informs read miss behavior: proxy vs promote

FLUSH AND/OR EVICT COLD DATA



TIERING AGENT

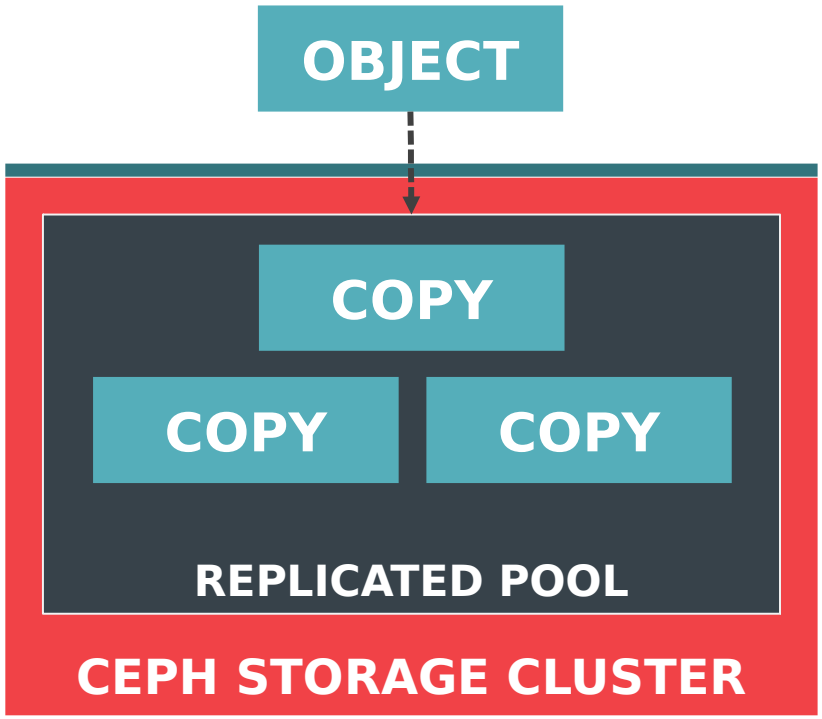
- Each PG has an internal tiering **agent**
  - Manages PG based on administrator defined policy
- Flush **dirty** objects
  - When pool reaches target dirty ratio
  - Tries to select cold objects
  - Marks objects **clean** when they have been written back to the base pool
- Evict (delete) clean objects
  - Greater “effort” as cache pool approaches target size

CACHE TIER USAGE

- Cache tier should be **faster** than the base tier
- Cache tier should be **replicated** (not erasure coded)
- Promote and flush are expensive
  - Best results when object temperature are **skewed**
    - Most I/O goes to small number of hot objects
  - Cache should be big enough to capture most of the acting set
- Challenging to benchmark
  - Need a realistic workload (e.g., not 'dd') to determine how it will perform in practice
  - Takes a long time to “warm up” the cache

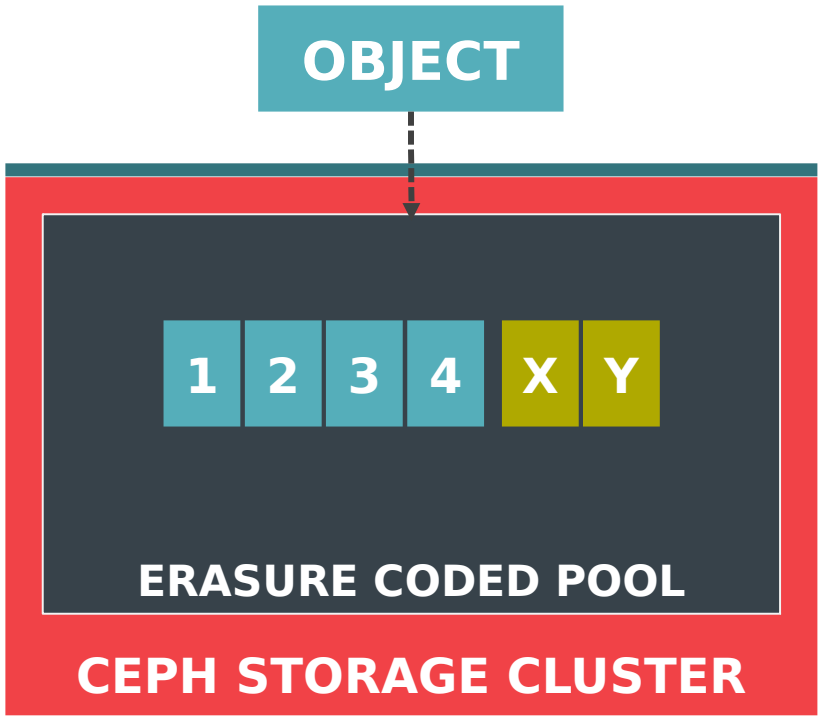
ERASURE CODING

ERASURE CODING



Full copies of stored objects

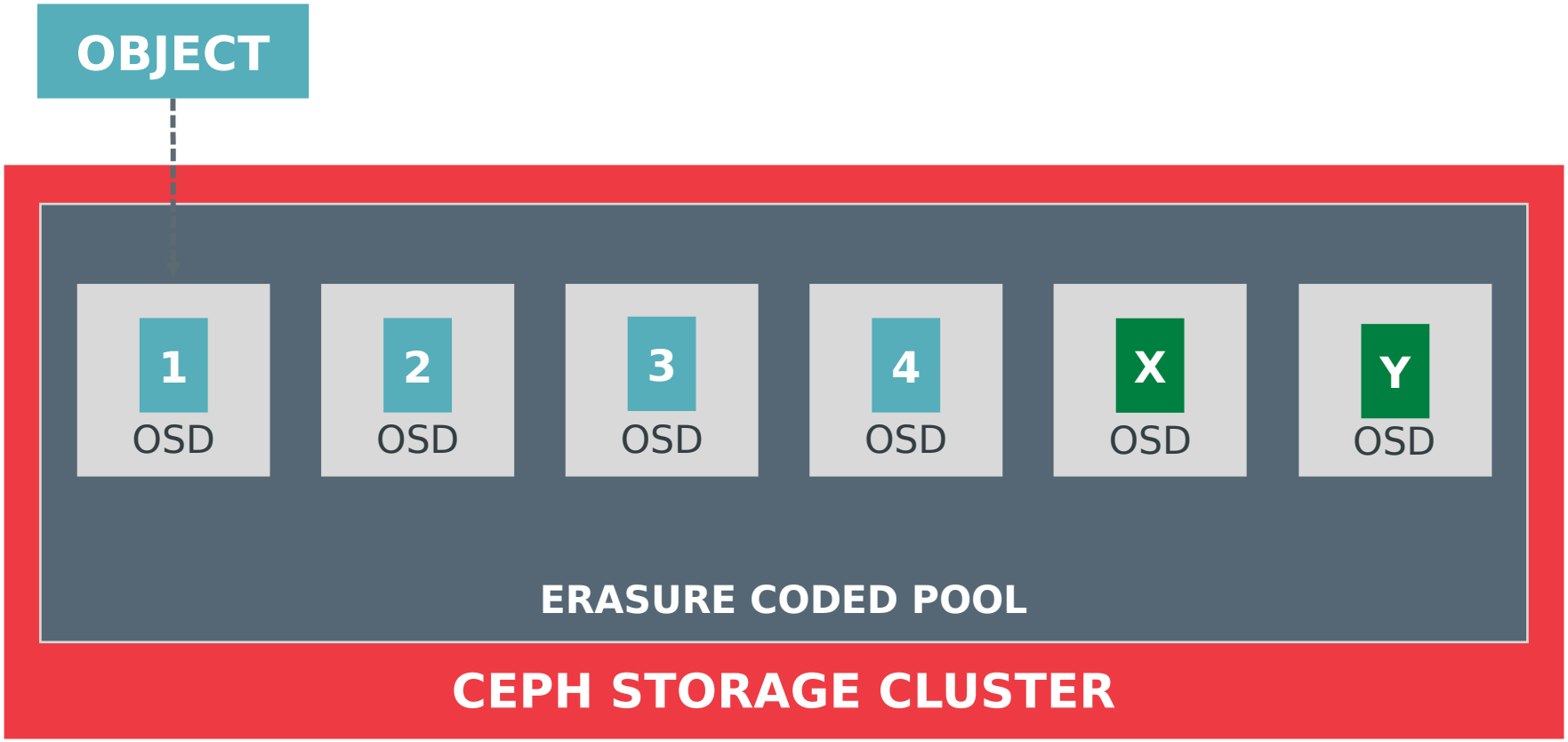
- Very high durability
- 3x (200% overhead)
- Quicker recovery



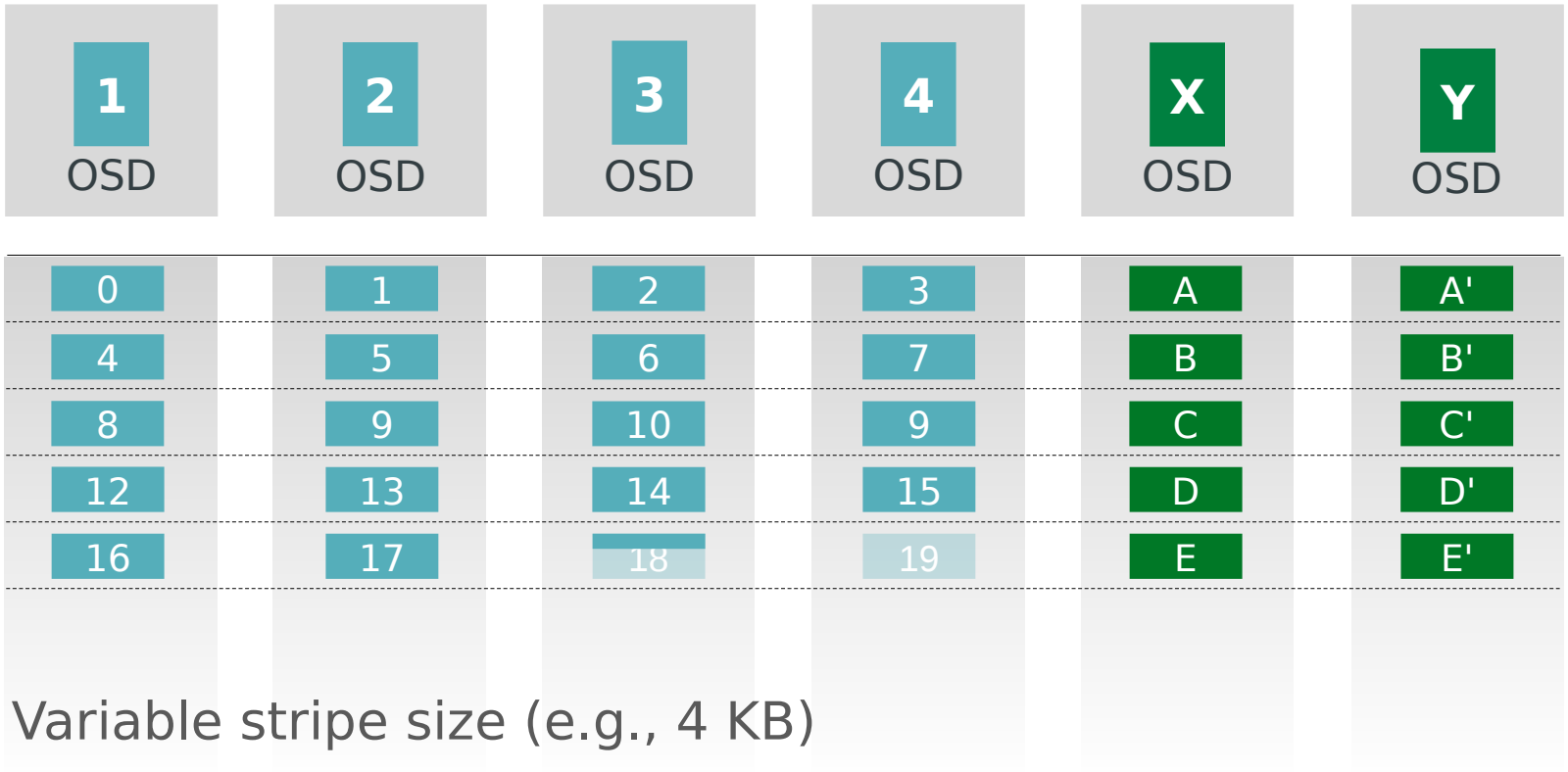
One copy plus parity

- Cost-effective durability
- 1.5x (50% overhead)
- Expensive recovery

ERASURE CODING SHARDS

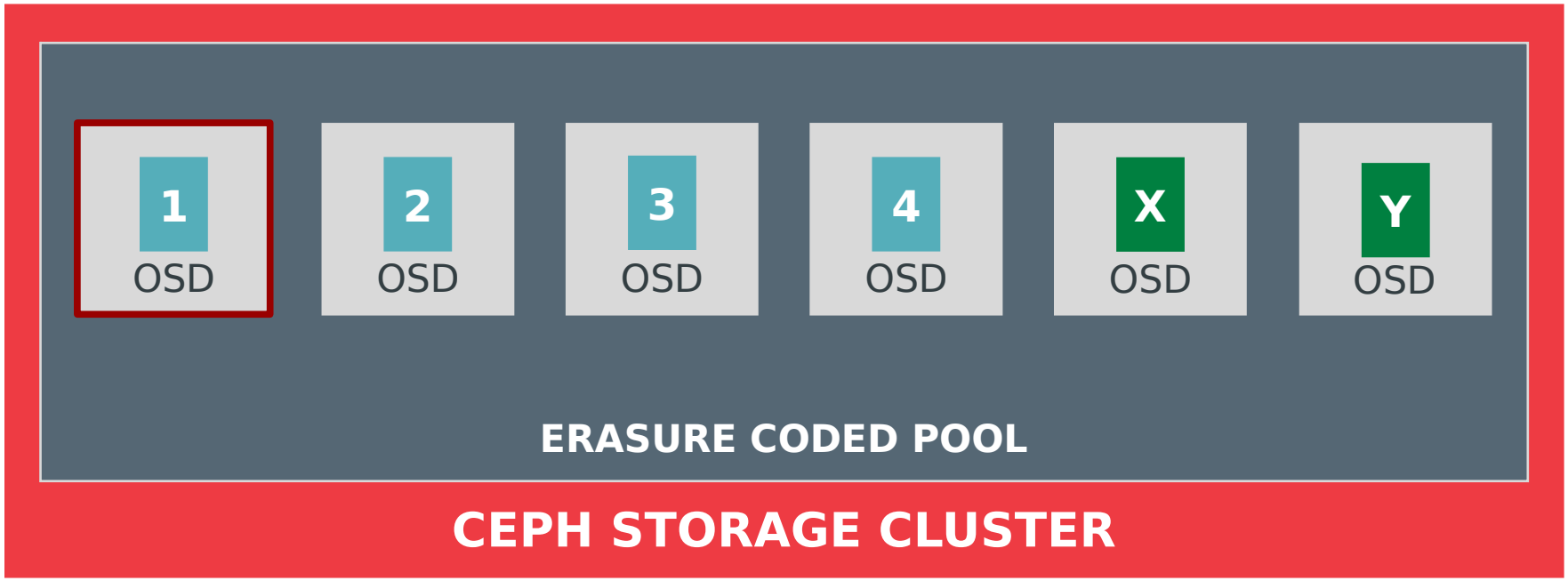


ERASURE CODING SHARDS

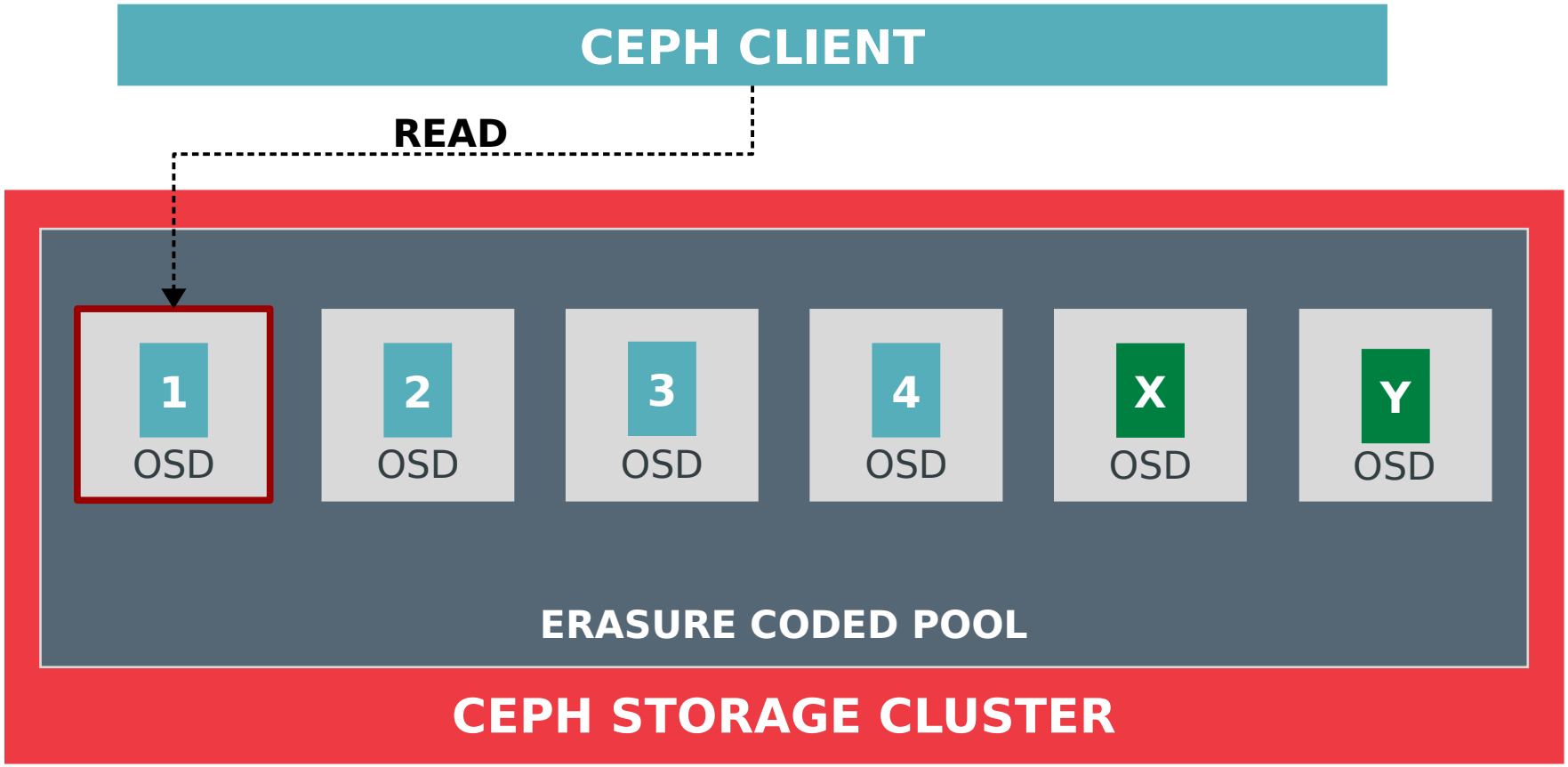


- Variable stripe size (e.g., 4 KB)
- Zero-fill shards (logically) in partial tail stripe

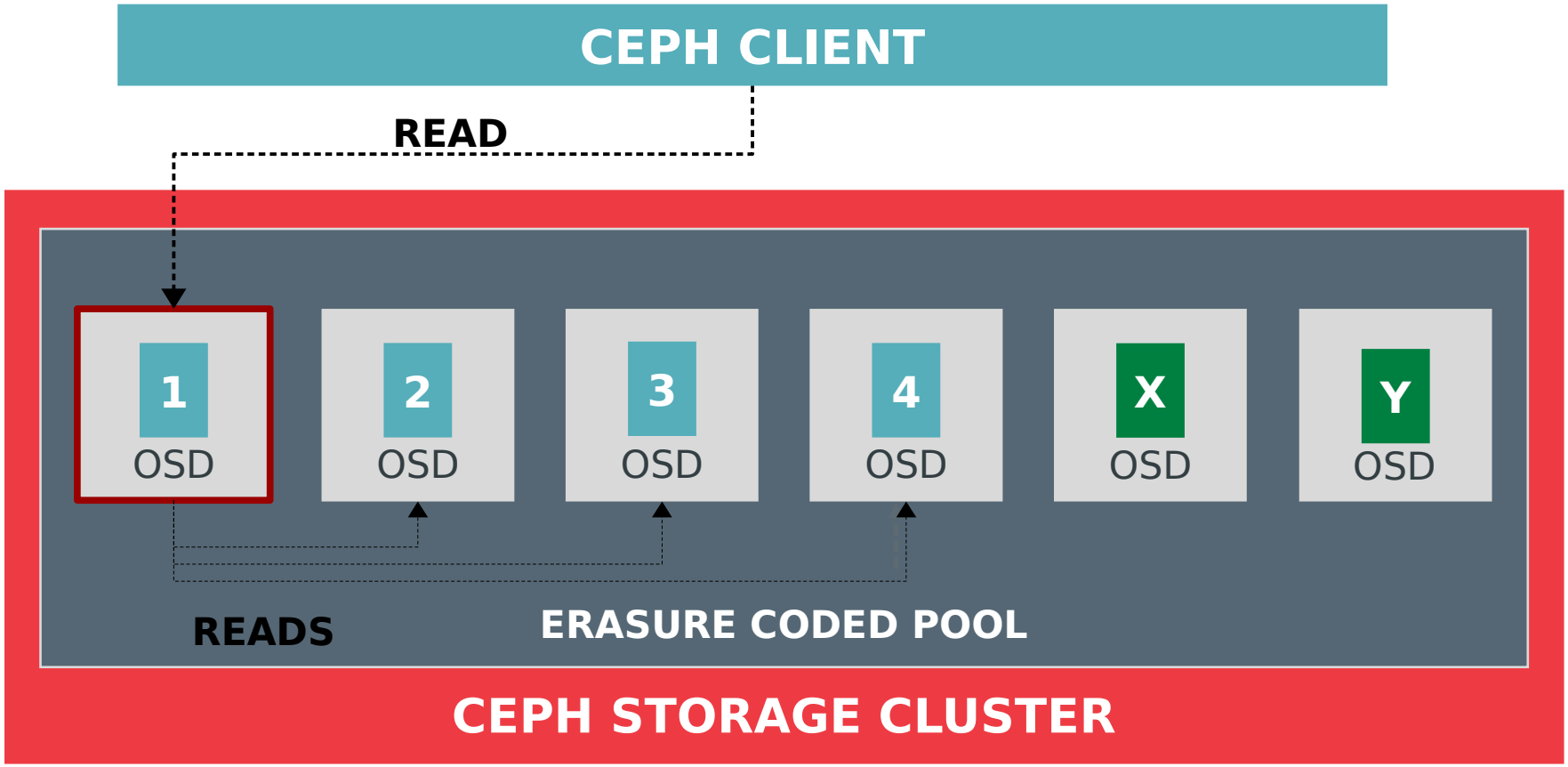
PRIMARY COORDINATES



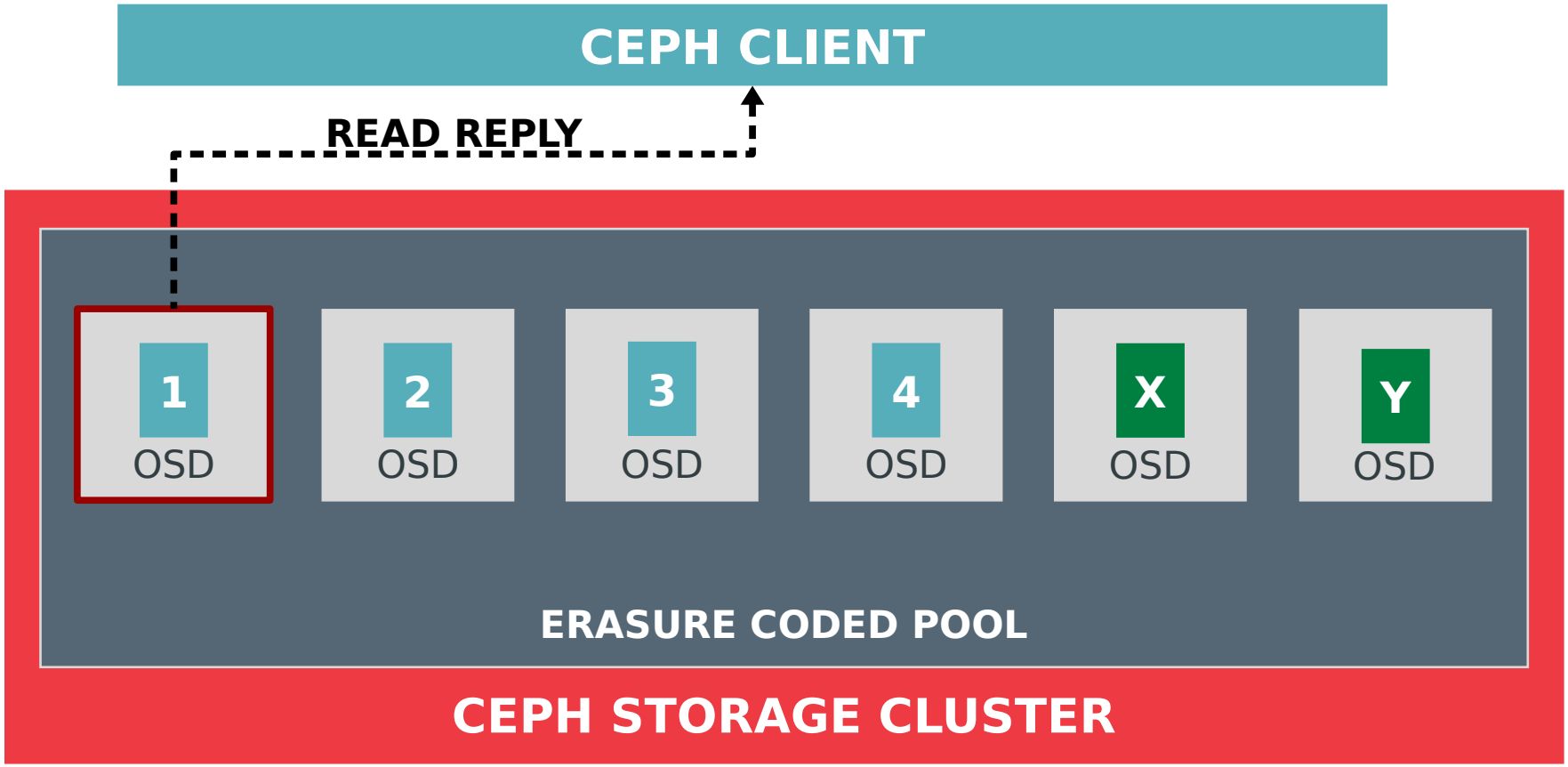
EC READ


CEPH CLIENT
READ
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
ERASURE CODED POOL
CEPH STORAGE CLUSTER

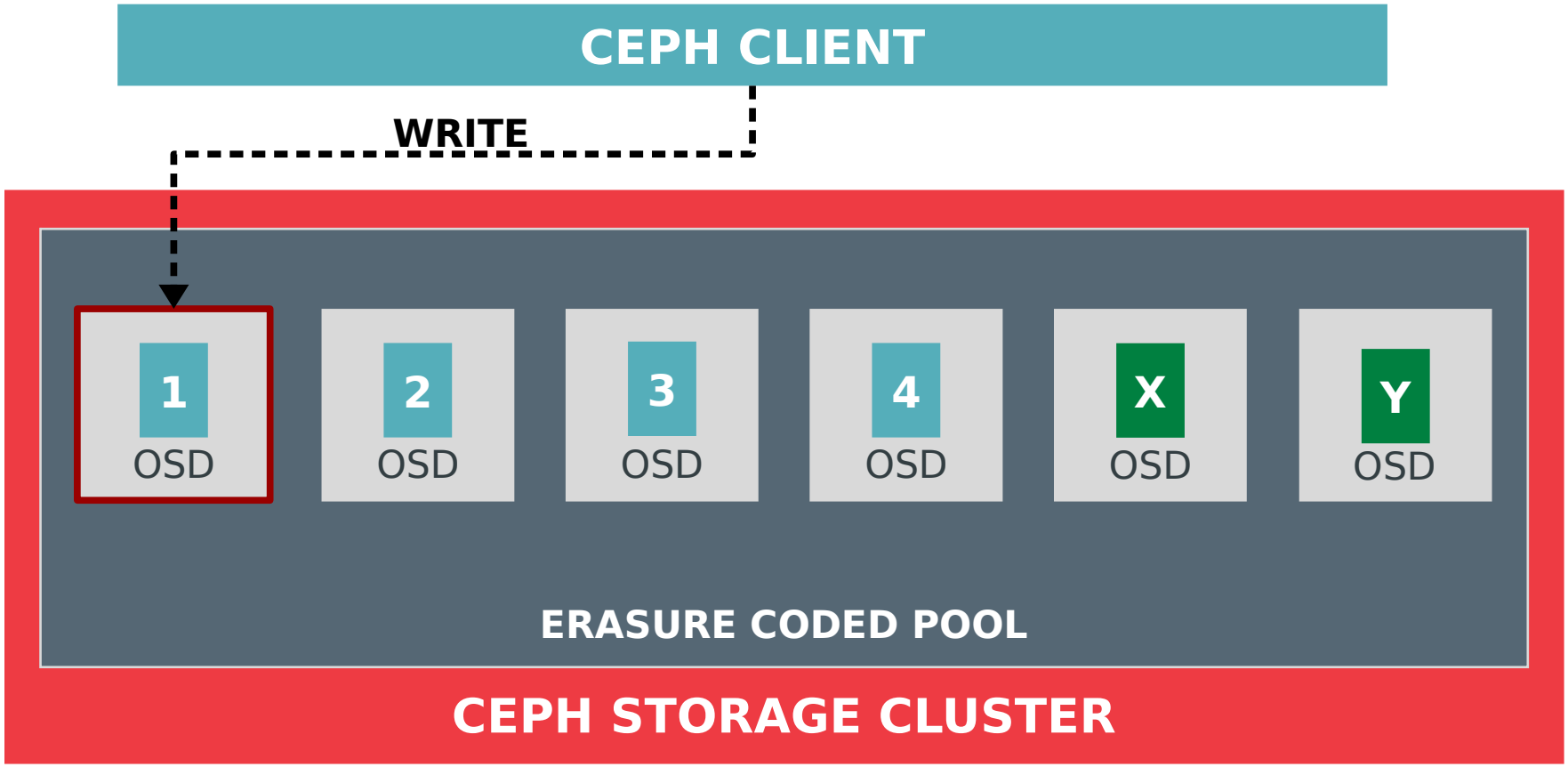
EC READ


CEPH CLIENT
READ
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
READS
ERASURE CODED POOL
CEPH STORAGE CLUSTER

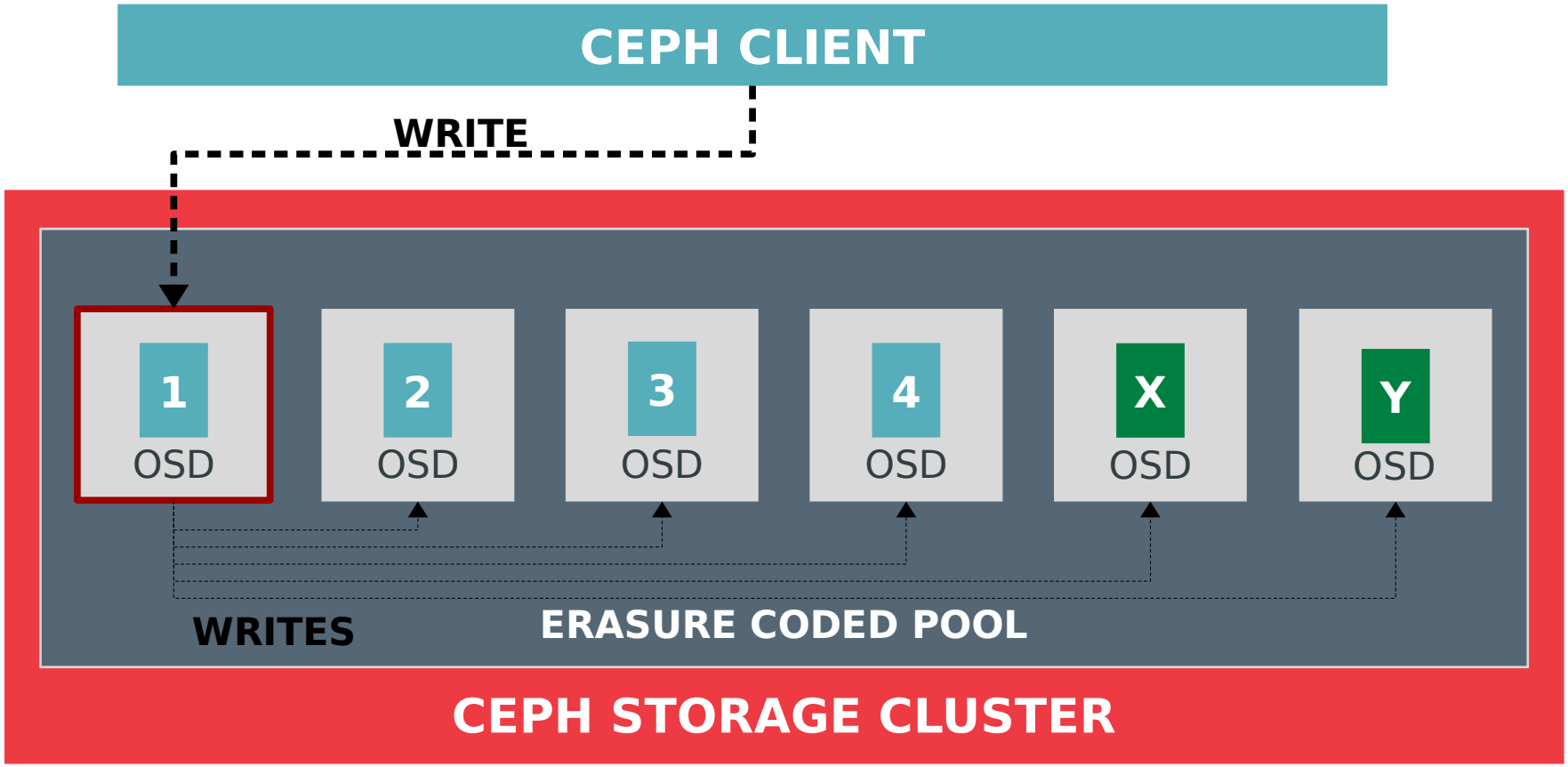
EC READ


CEPH CLIENT
READ REPLY
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
ERASURE CODED POOL
CEPH STORAGE CLUSTER

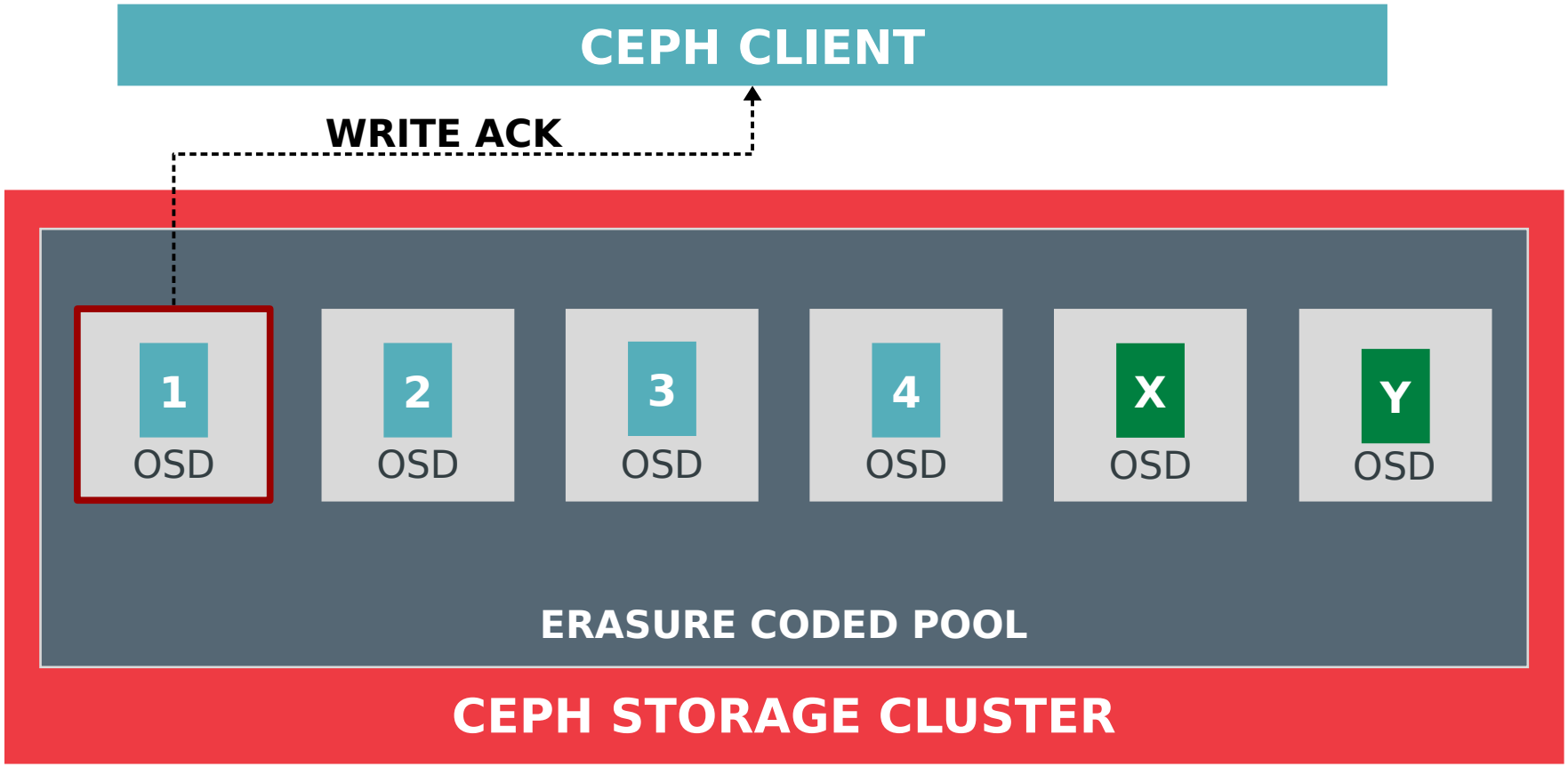
EC WRITE


CEPH CLIENT
WRITE
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
ERASURE CODED POOL
CEPH STORAGE CLUSTER

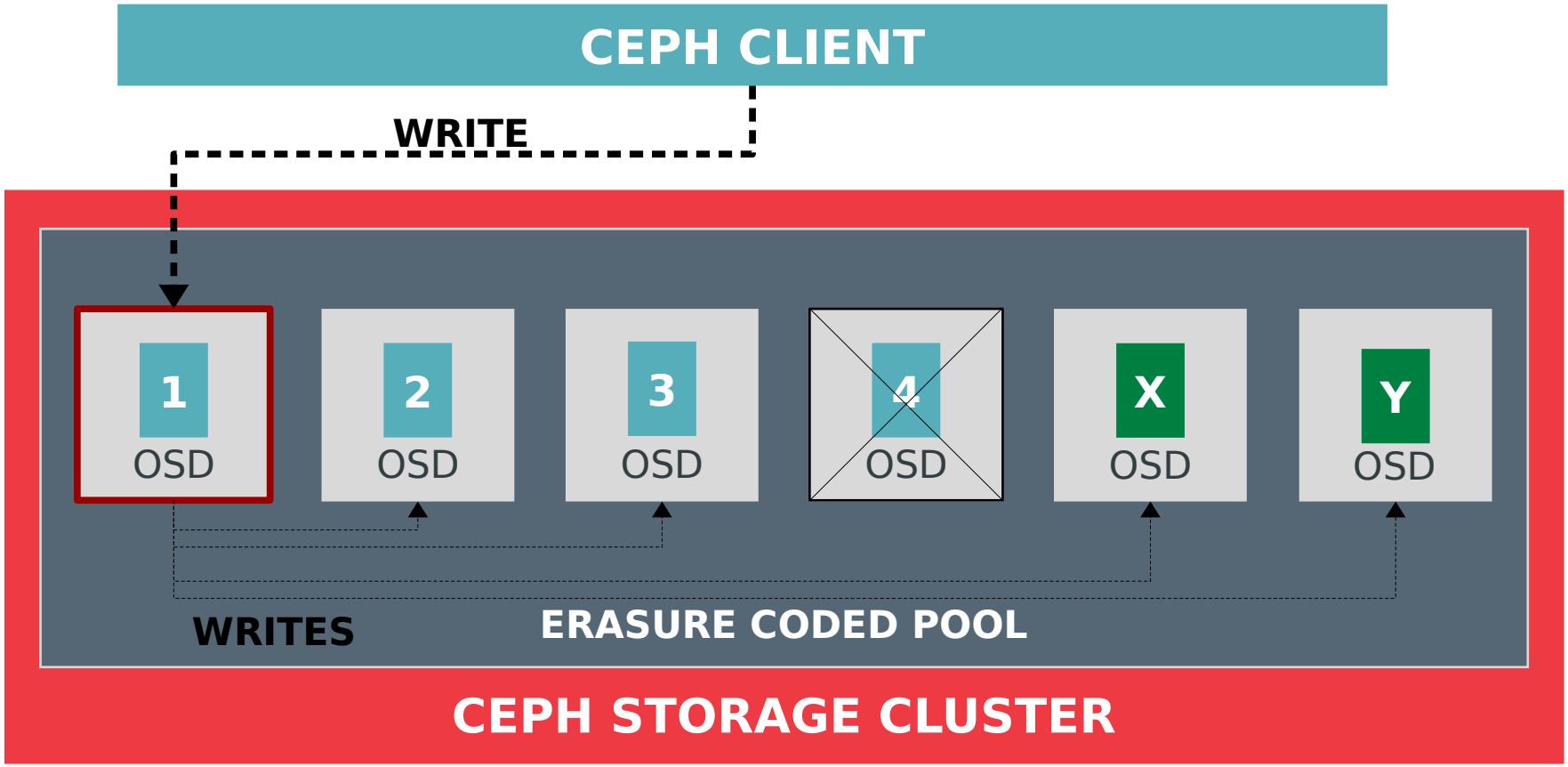
EC WRITE


CEPH CLIENT
WRITE
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
WRITES
ERASURE CODED POOL
CEPH STORAGE CLUSTER

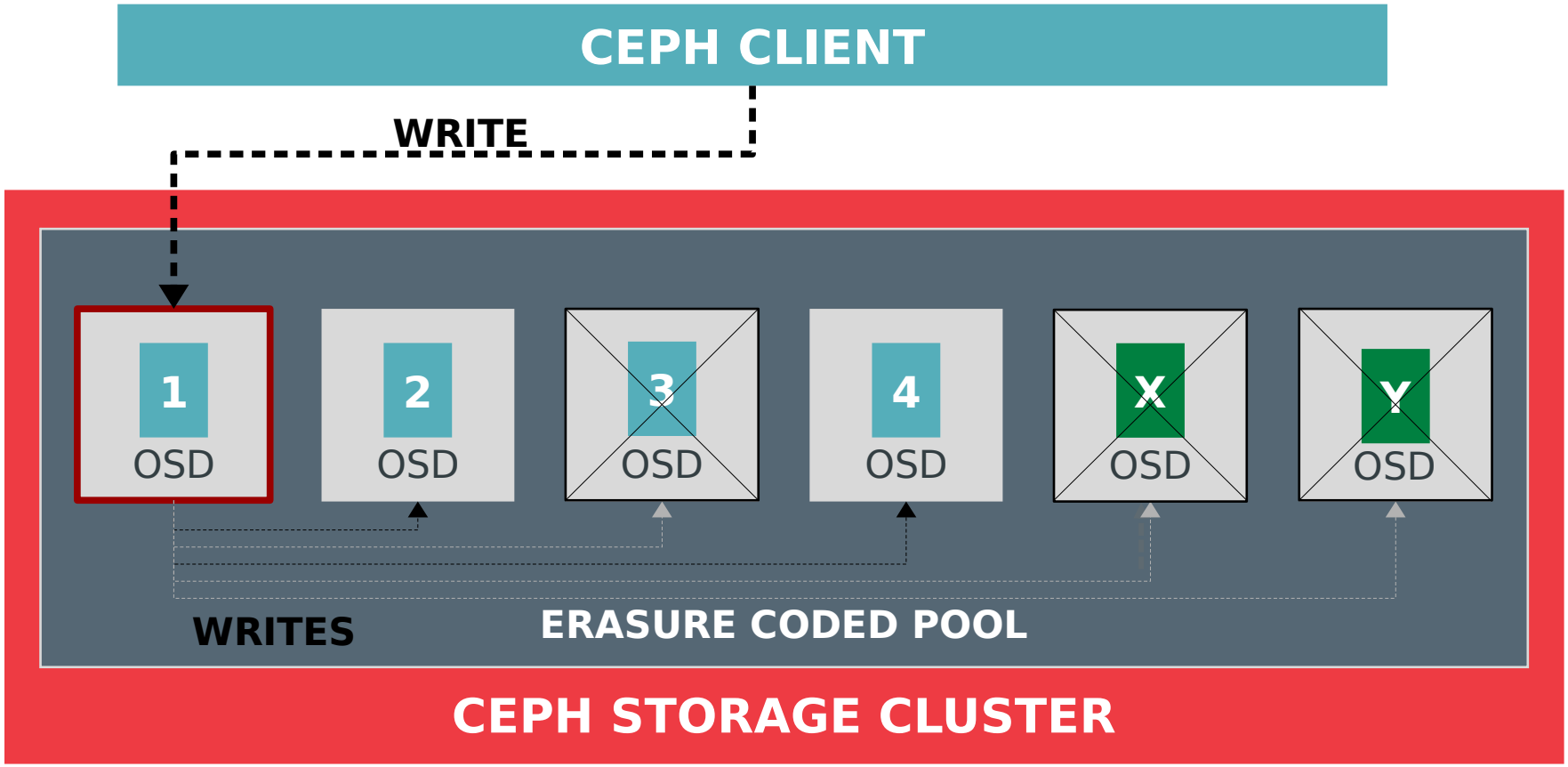
EC WRITE


CEPH CLIENT
WRITE ACK
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
ERASURE CODED POOL
CEPH STORAGE CLUSTER

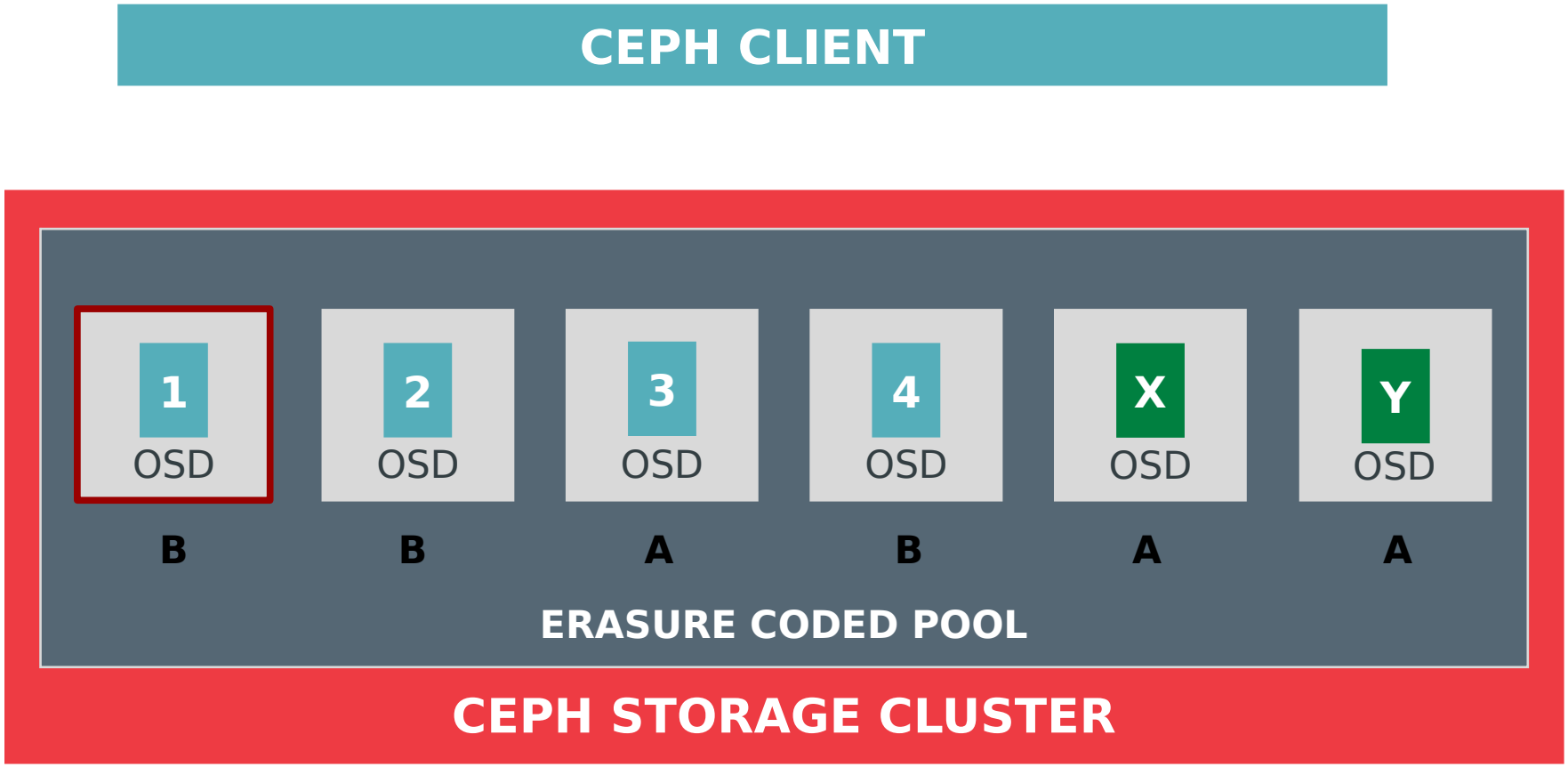
EC WRITE: DEGRADED



EC WRITE: PARTIAL FAILURE



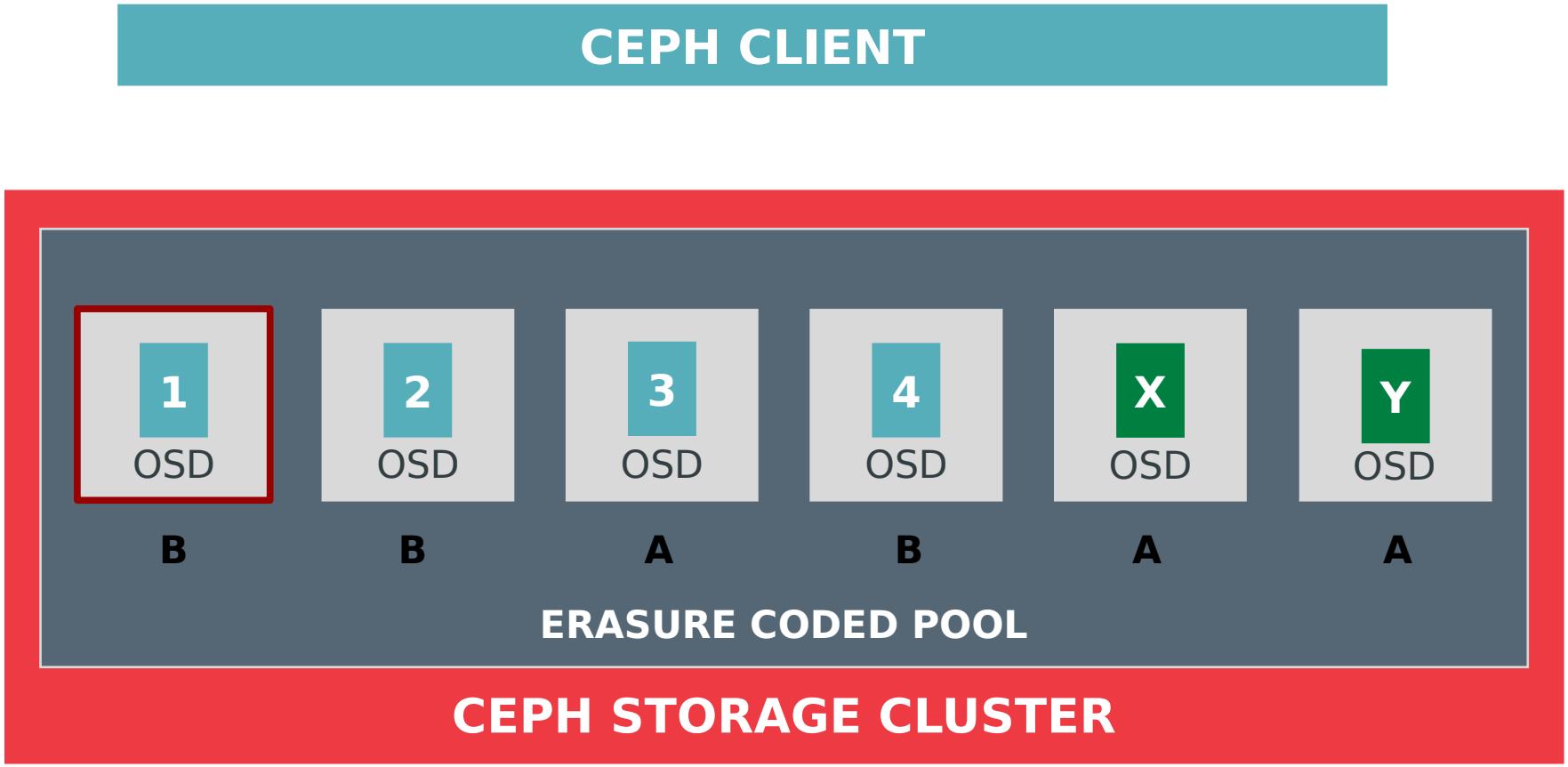
EC WRITE: PARTIAL FAILURE



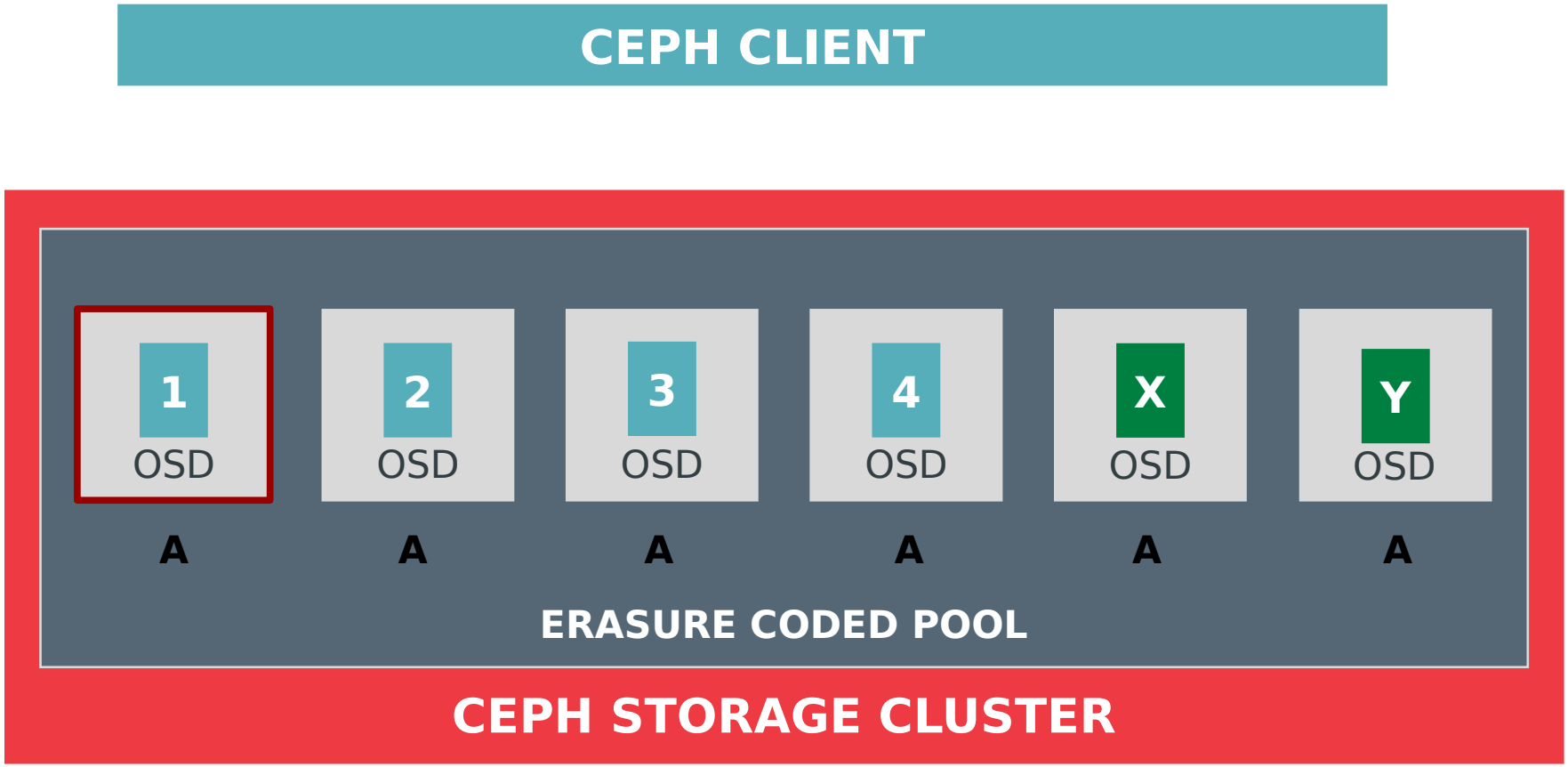
EC RESTRICTIONS

- Overwrite in place will not work in general
- Log and 2PC would increase complexity, latency
- We chose to restrict allowed operations
  - create
  - append (on stripe boundary)
  - remove (keep previous generation of object for some time)
- These operations can all easily be rolled back locally
  - create → delete
  - append → truncate
  - remove → roll back to previous generation
- Object attrs preserved in existing PG logs (they are small)
- Key/value data is not allowed on EC pools

EC WRITE: PARTIAL FAILURE



EC WRITE: PARTIAL FAILURE



EC RESTRICTIONS

- This is a small subset of allowed librados operations
  - Notably cannot (over)write any extent
- Coincidentally, unsupported operations are also inefficient for erasure codes
  - Generally require read/modify/write of affected stripe(s)
- Some can **consume EC directly**
  - RGW (no object data update in place)
- Others can **combine EC with a cache** tier (RBD, CephFS)
  - Replication for warm/hot data
  - Erasure coding for cold data
  - Tiering agent skips objects with key/value data

WHICH ERASURE CODE?

- The EC algorithm and implementation are **pluggable**
  - jerasure/gf-complete (free, open, and very fast)
  - ISA-L (Intel library; optimized for modern Intel procs)
  - LRC (local recovery code – layers over existing plugins)
  - SHEC (trades extra storage for recovery efficiency – new from Fujitsu)
- Parameterized
  - Pick “k” and “m”, stripe size
- **OSD handles data path**, placement, rollback, etc.
- Erasure plugin handles
  - Encode and decode **math**
  - Given these available shards, which ones should I fetch to satisfy a read?
  - Given these available shards and these missing shards, which ones should I fetch to recover?

COST OF RECOVERY

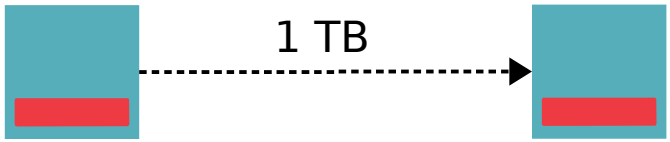
1 TB OSD

COST OF RECOVERY

1 TB OSD

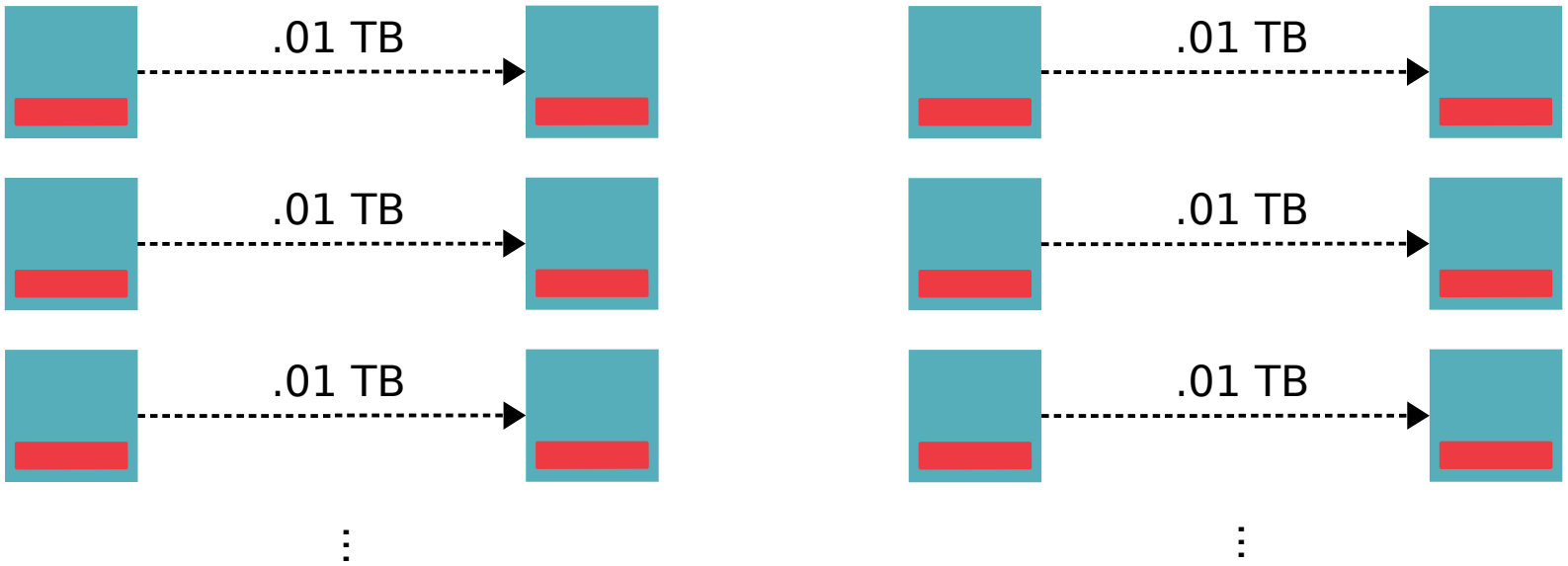
COST OF RECOVERY (REPLICATION)

1 TB OSD



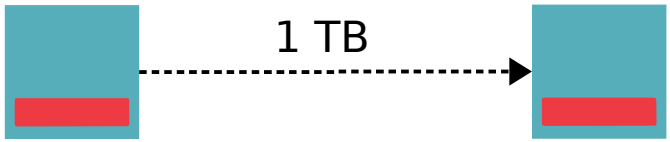
COST OF RECOVERY (REPLICATION)

1 TB OSD

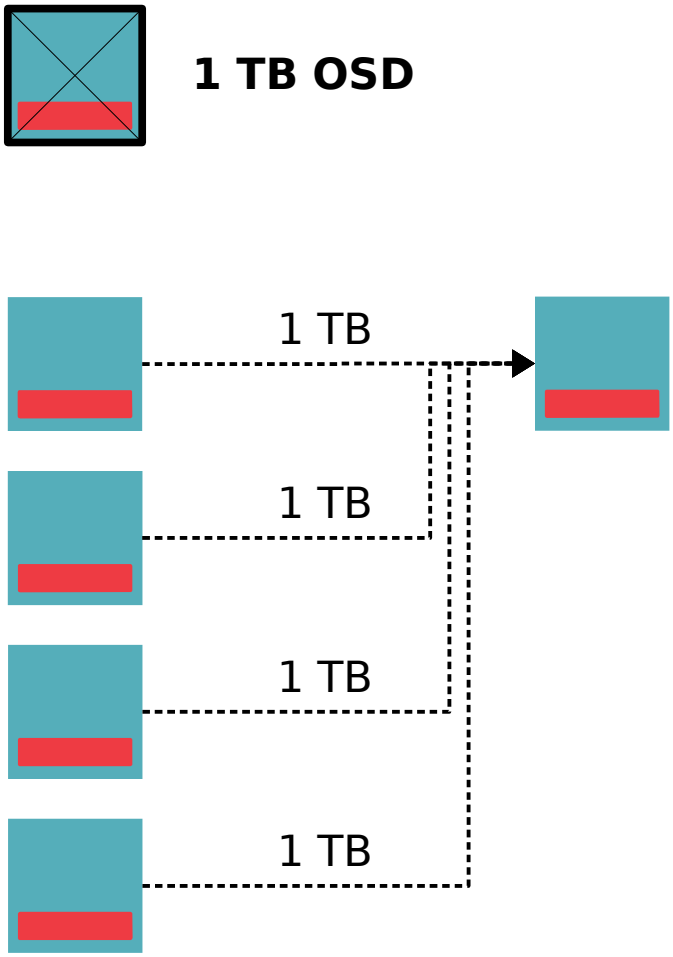


COST OF RECOVERY (REPLICATION)

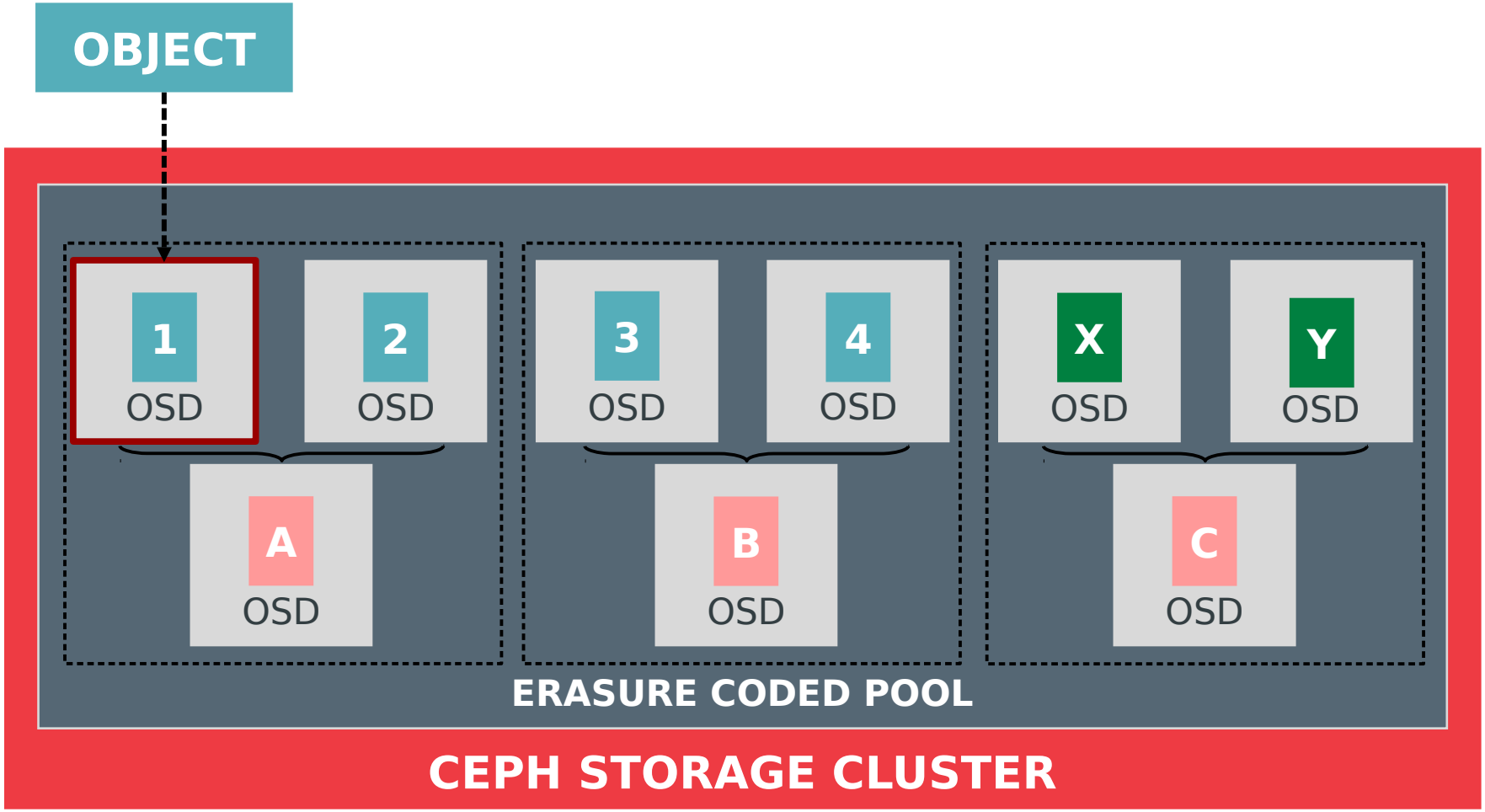
1 TB OSD



COST OF RECOVERY (EC)



LOCAL RECOVERY CODE (LRC)



BIG THANKS TO

- Ceph
  - Loic Dachary (CloudWatt, FSF France, Red Hat)
  - Andreas Peters (CERN)
  - Sam Just (Inktank / Red Hat)
  - David Zafman (Inktank / Red Hat)
- jerasure / gf-complete
  - Jim Plank (University of Tennessee)
  - Kevin Greenan (Box.com)
- Intel (ISL plugin)
- Fujitsu (SHEC plugin)

ROADMAP

WHAT'S NEXT

- Erasure coding
  - Allow (optimistic) client reads directly from shards
  - ARM optimizations for jerasure
- Cache pools
  - Better agent decisions (when to flush or evict)
  - Supporting different performance profiles
    - e.g., slow / “cheap” flash can read just as fast
  - Complex topologies
    - Multiple readonly cache tiers in multiple sites
- Tiering
  - Support “redirects” to (very) cold tier below base pool
  - Enable dynamic spin-down, dedup, and other features

OTHER ONGOING WORK

- **Performance** optimization (SanDisk, Intel, Mellanox)
- Alternative OSD backends
  - New backend: hybrid key/value and file system
  - leveldb, rocksdb, LMDB
- Messenger (network layer) improvements
  - RDMA support (libxio – Mellanox)
  - Event-driven TCP implementation (UnitedStack)
- CephFS
  - Online consistency checking and repair tools
  - Performance, robustness
- Multi-datacenter RBD, RADOS replication

FOR MORE INFORMATION

- http://ceph.com
- http://github.com/ceph
- http://tracker.ceph.com
- Mailing lists
  - ceph-users@ceph.com
  - ceph-devel@vger.kernel.org
- irc.oftc.net
  - #ceph
  - #ceph-devel
- Twitter
  - @ceph

THANK YOU!

Sage Weil
CEPH PRINCIPAL ARCHITECT


sage@redhat.com


@liewegas