ERASURE CODING AND CACHE TIERING

SAGE WEIL - SDC 2014.09.16

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

ARCHITECTURAL COMPONENTS

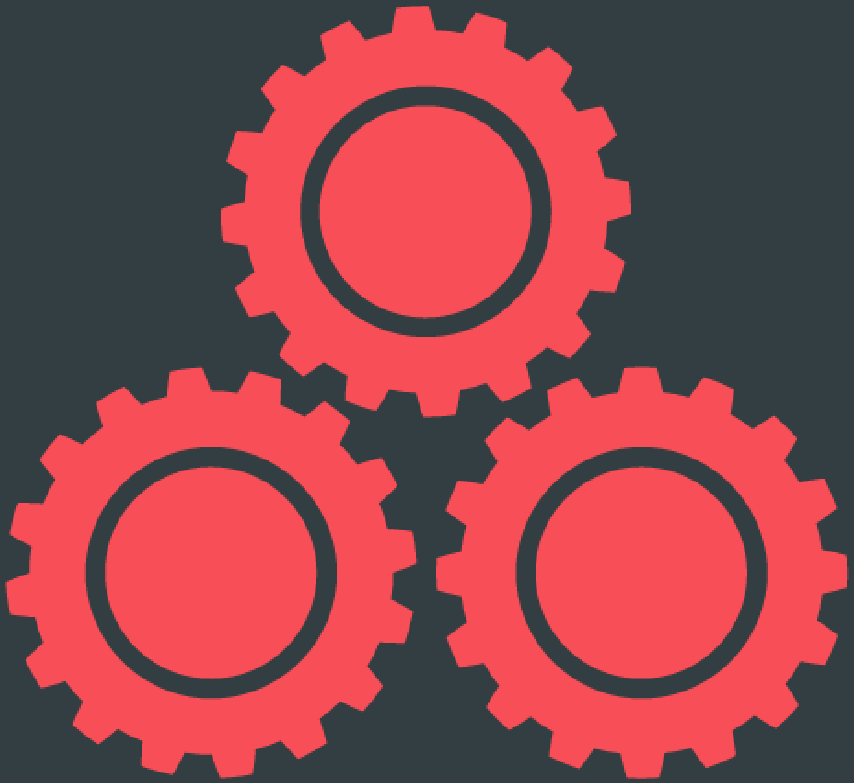
APP ↔ **RGW**
A web services gateway for object storage, compatible with S3 and Swift

HOST/VM ↔ **RBD**
A reliable, fully-distributed block device with cloud platform integration

CLIENT ↔ **CEPHFS**
A distributed file system with POSIX semantics and scale-out metadata management

LIBRADOS
A library allowing apps to directly access RADOS (C, C++, Java, Python, Ruby, PHP)

RADOS
A software-based, reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes and lightweight monitors



ROBUST SERVICES BUILT ON RADOS

ARCHITECTURAL COMPONENTS

APP | HOST/VM | CLIENT

RGW
A web services gateway for object storage, compatible with S3 and Swift

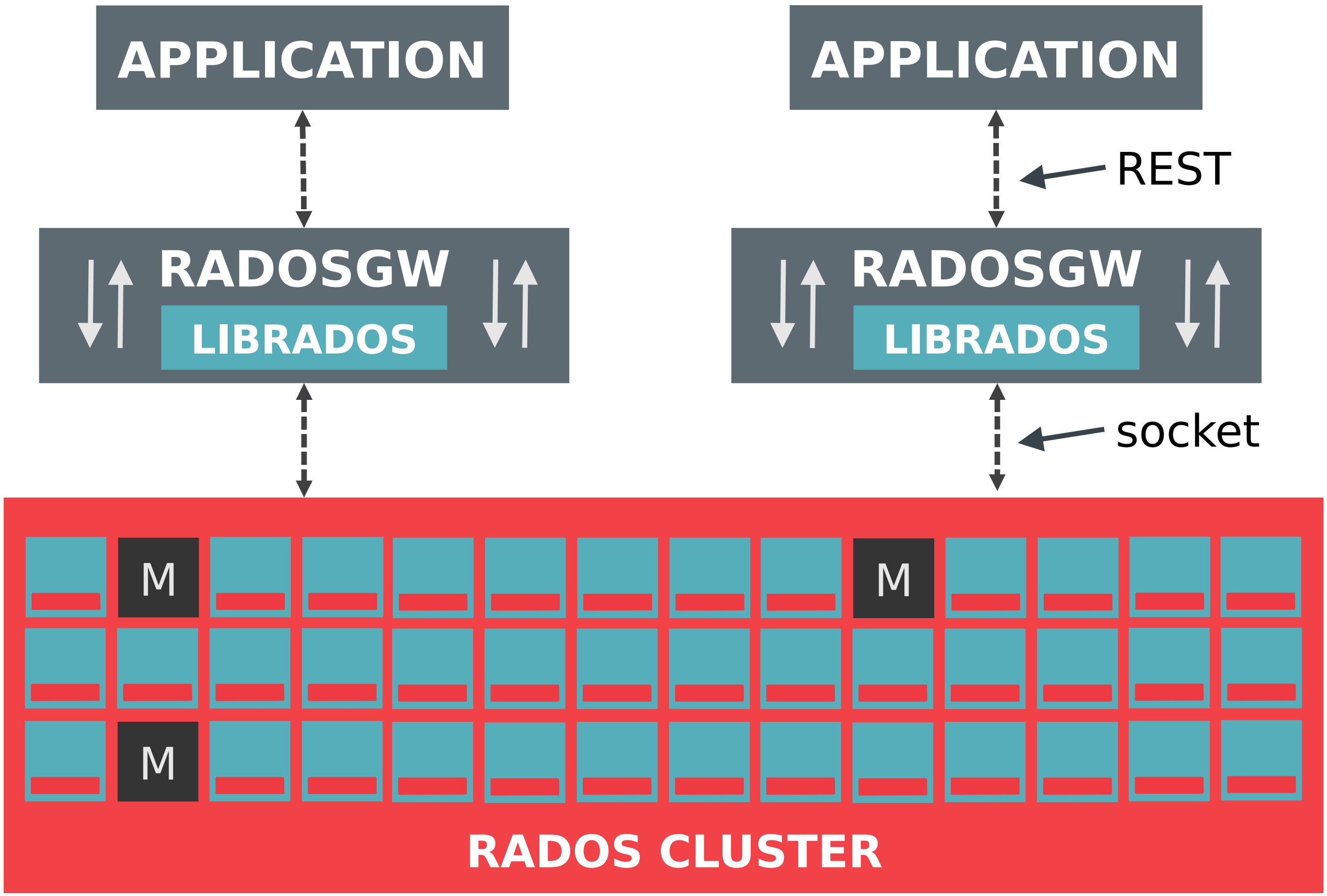
RBD
A reliable, fully-distributed block device with cloud platform integration

CEPHFS
A distributed file system with POSIX semantics and scale-out metadata management

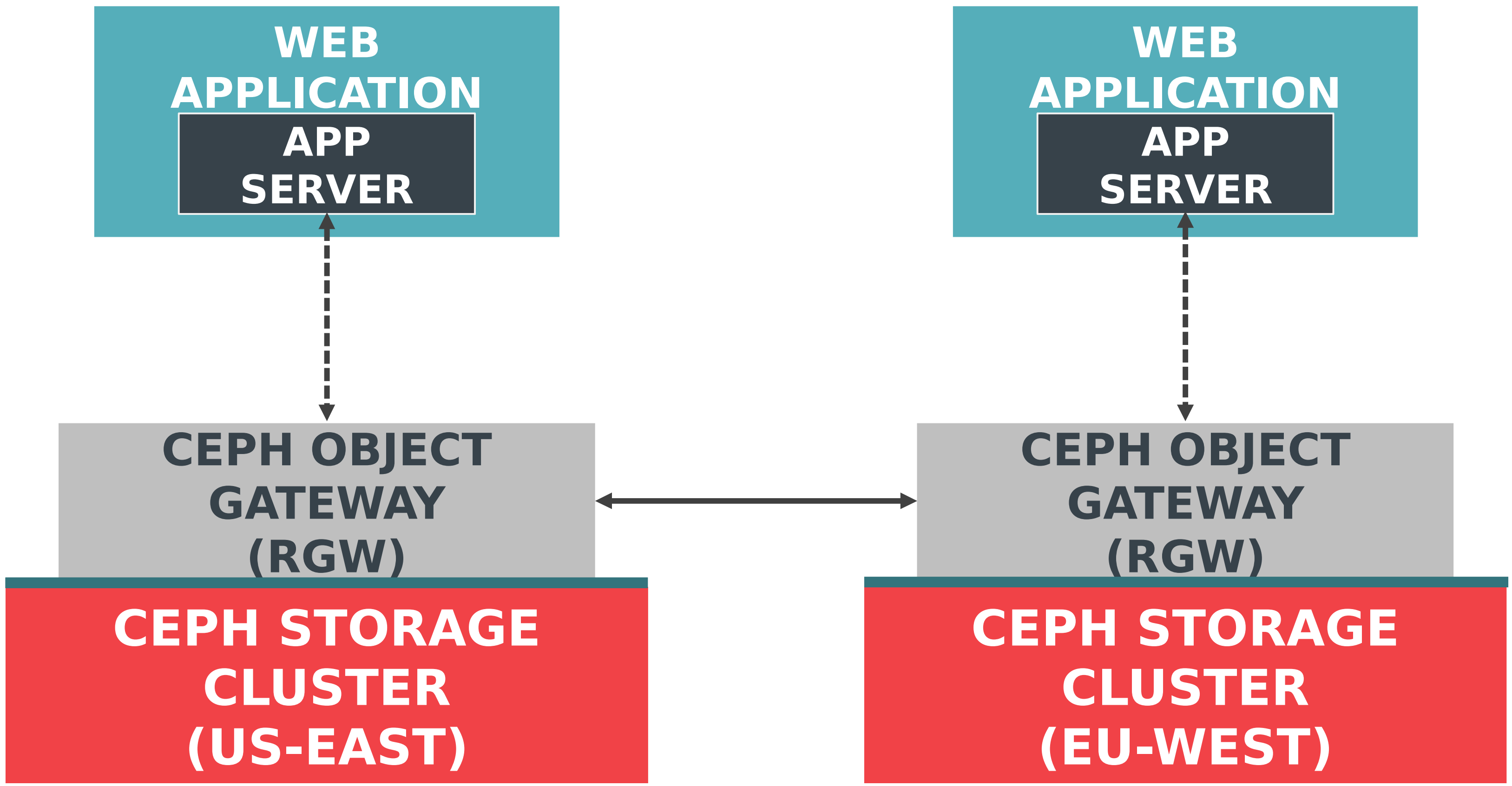
LIBRADOS
A library allowing apps to directly access RADOS (C, C++, Java, Python, Ruby, PHP)

RADOS
A software-based, reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes and lightweight monitors

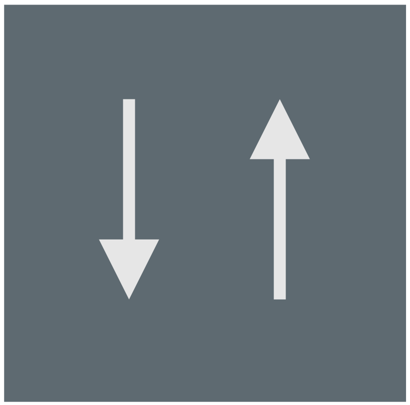
THE RADOS GATEWAY



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

APP

HOST/VM

CLIENT

RGW
A web services gateway for object storage, compatible with S3 and Swift

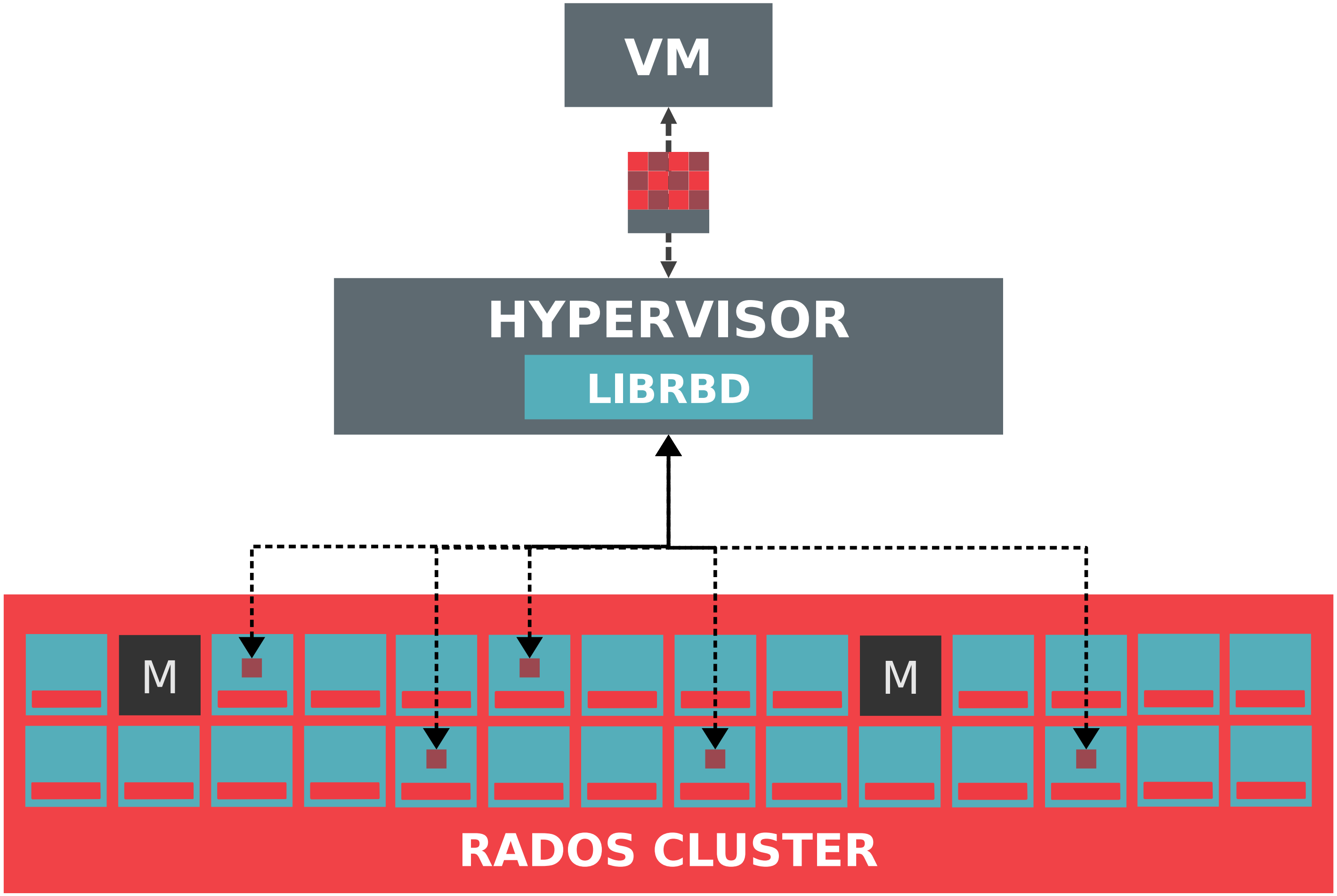
RBD
A reliable, fully-distributed block device with cloud platform integration

CEPHFS
A distributed file system with POSIX semantics and scale-out metadata management

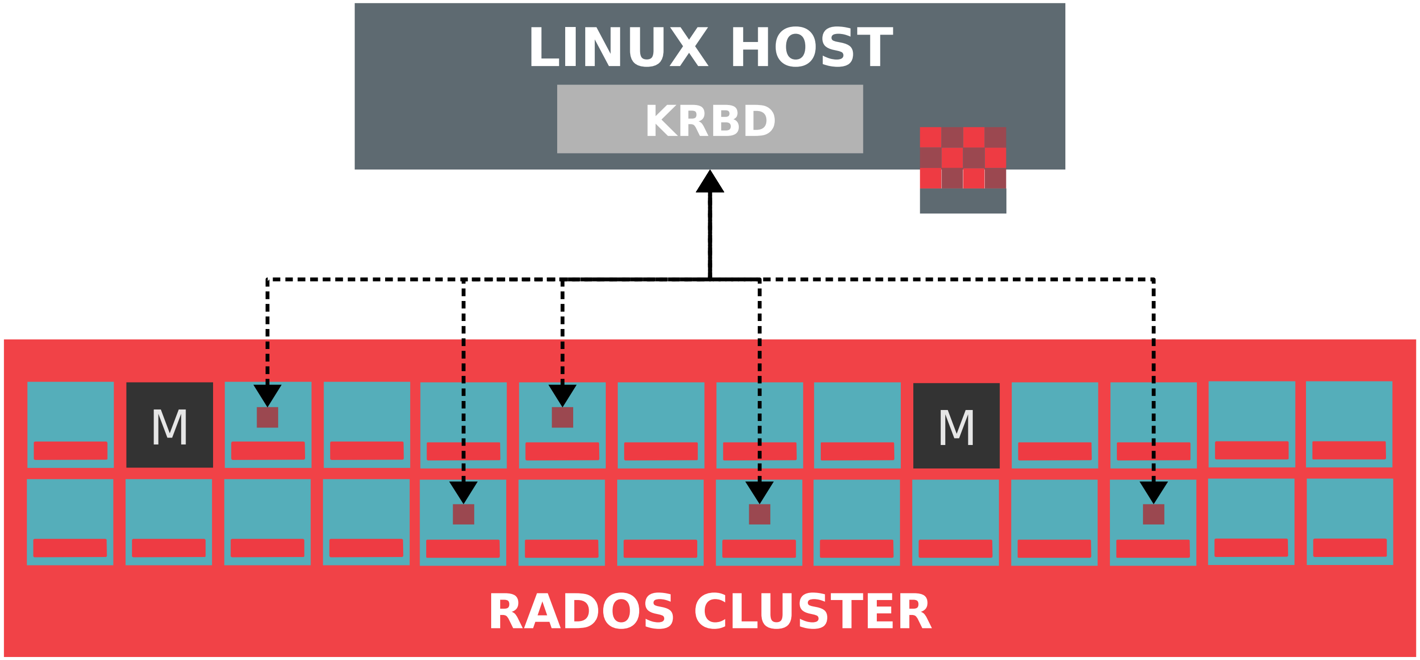
LIBRADOS
A library allowing apps to directly access RADOS (C, C++, Java, Python, Ruby, PHP)

RADOS
A software-based, reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes and lightweight monitors

STORING VIRTUAL DISKS

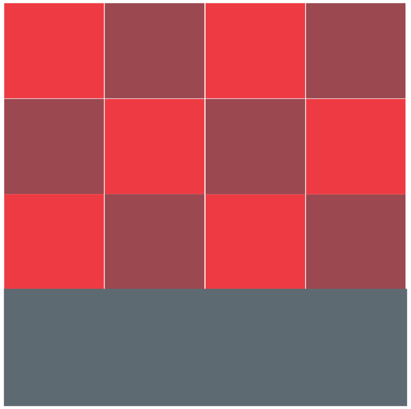


KERNEL MODULE



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

APP

HOST/VM

CLIENT

RGW
A web services gateway for object storage, compatible with S3 and Swift

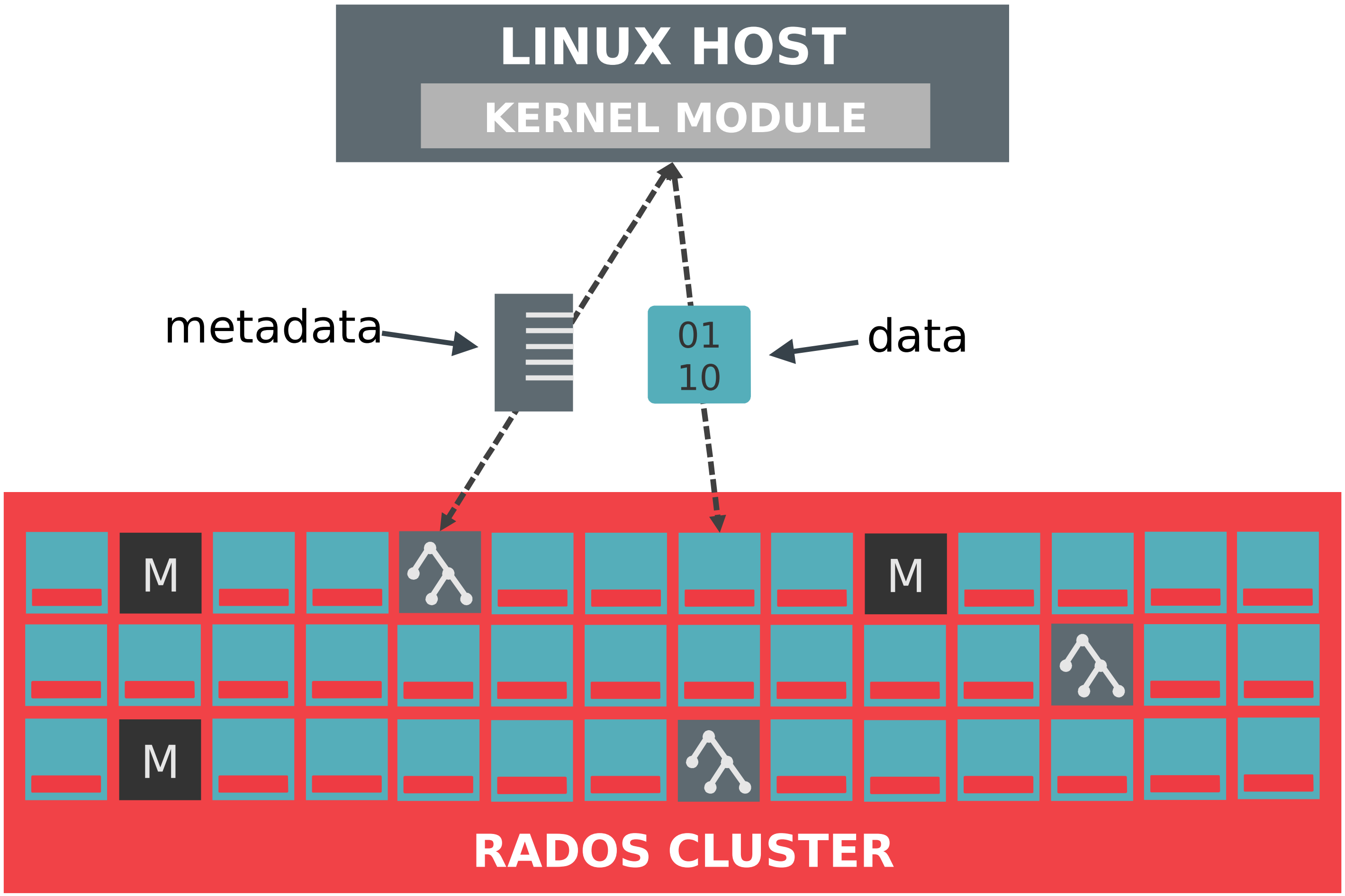
RBD
A reliable, fully-distributed block device with cloud platform integration

CEPHFS
A distributed file system with POSIX semantics and scale-out metadata management

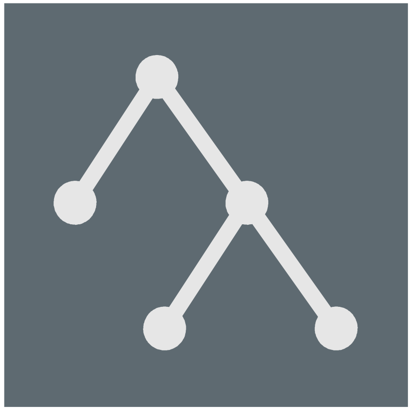
LIBRADOS
A library allowing apps to directly access RADOS (C, C++, Java, Python, Ruby, PHP)

RADOS
A software-based, reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes and lightweight monitors

SEPARATE METADATA SERVER

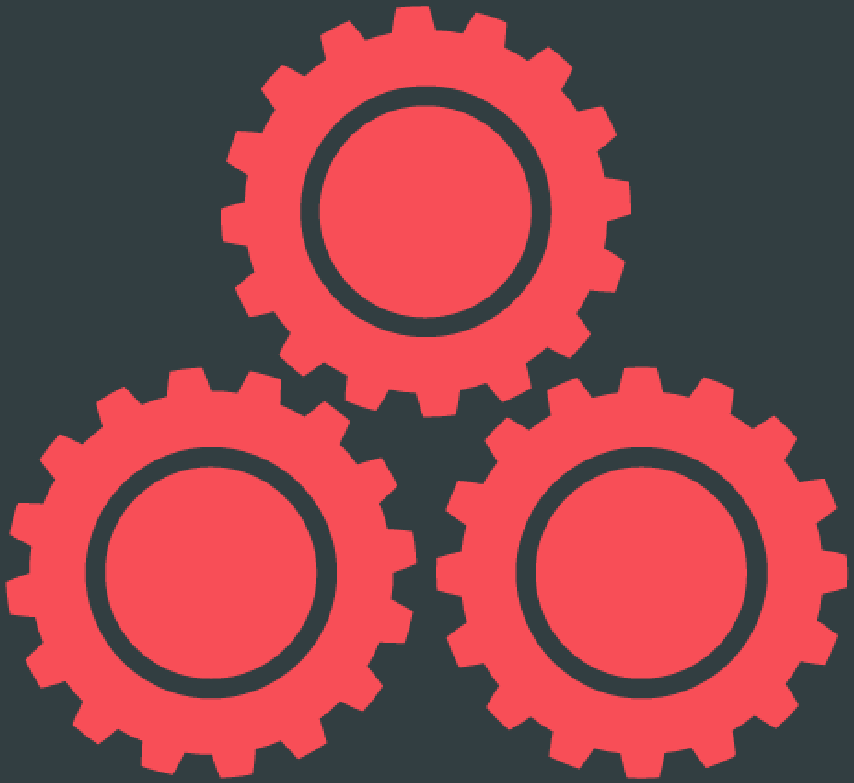


SCALABLE METADATA SERVERS



METADATA SERVER

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



RADOS

ARCHITECTURAL COMPONENTS

APP

HOST/VM

CLIENT

RGW
A web services gateway for object storage, compatible with S3 and Swift

RBD
A reliable, fully-distributed block device with cloud platform integration

CEPHFS
A distributed file system with POSIX semantics and scale-out metadata management

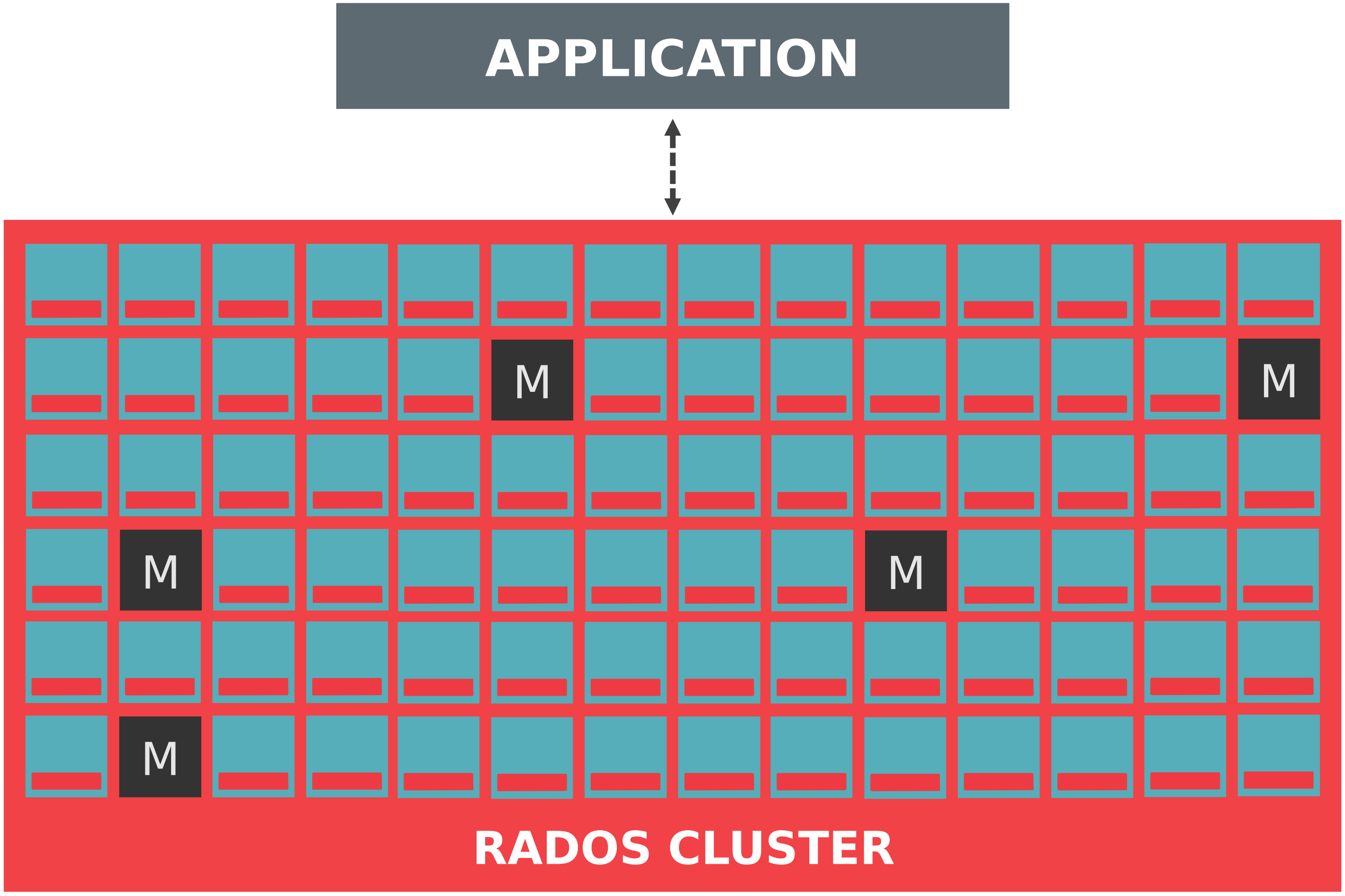
LIBRADOS
A library allowing apps to directly access RADOS (C, C++, Java, Python, Ruby, PHP)

RADOS
A software-based, reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes and lightweight monitors

RADOS

- Flat object namespace within each pool
- Rich object API (librados)
  - Bytes, attributes, key/value data
  - Partial overwrite of existing data
  - 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

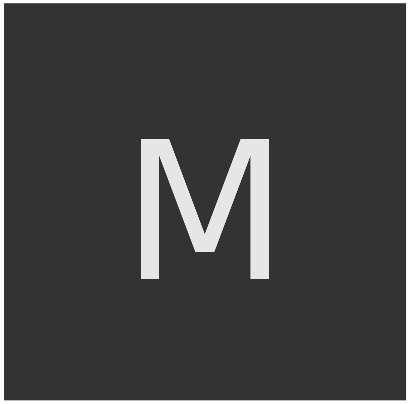
RADOS CLUSTER



RADOS COMPONENTS

OSDs:

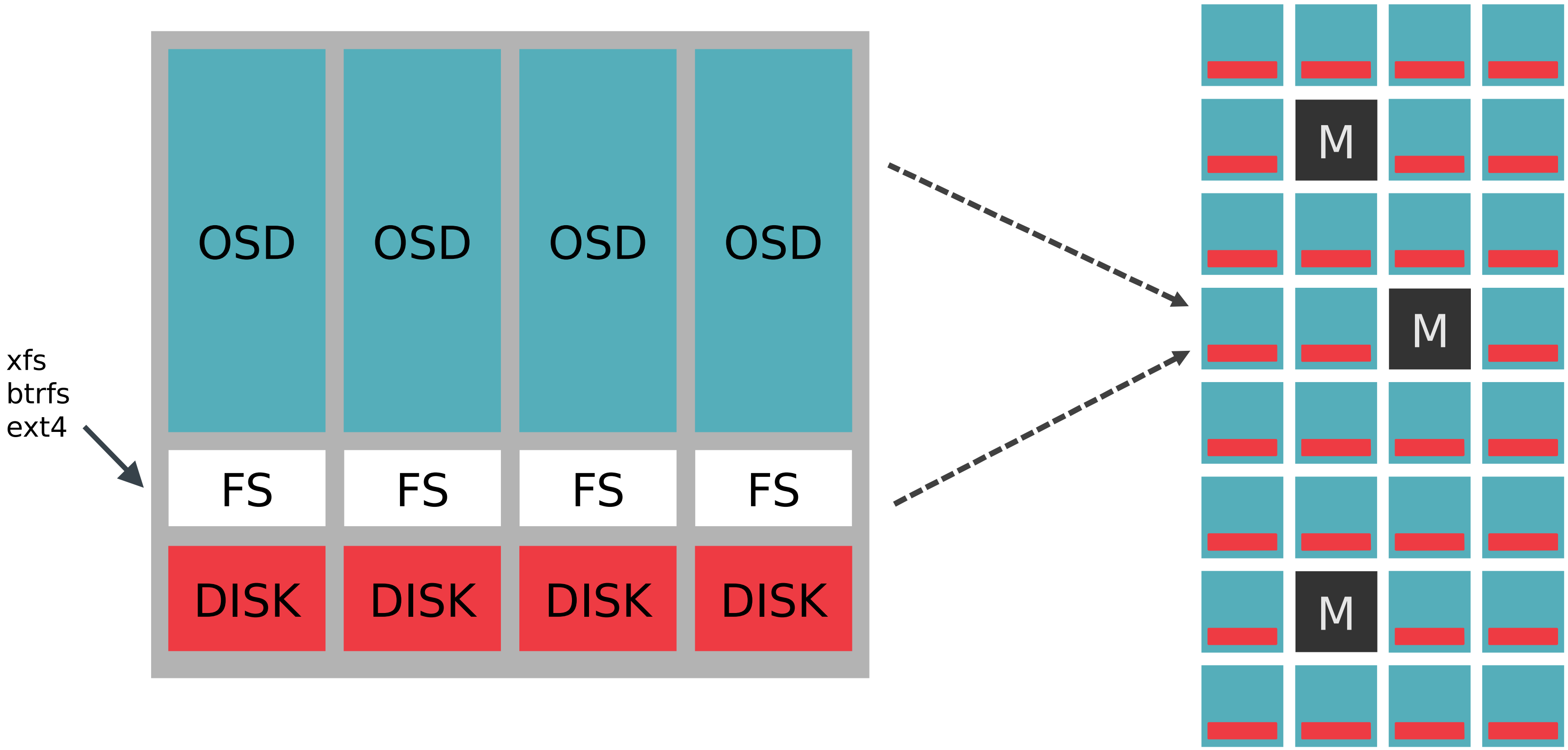
- 10s to 1000s in a cluster
- One per disk (or one per SSD, RAID group…)
- Serve stored objects to clients
- Intelligently peer for replication & recovery

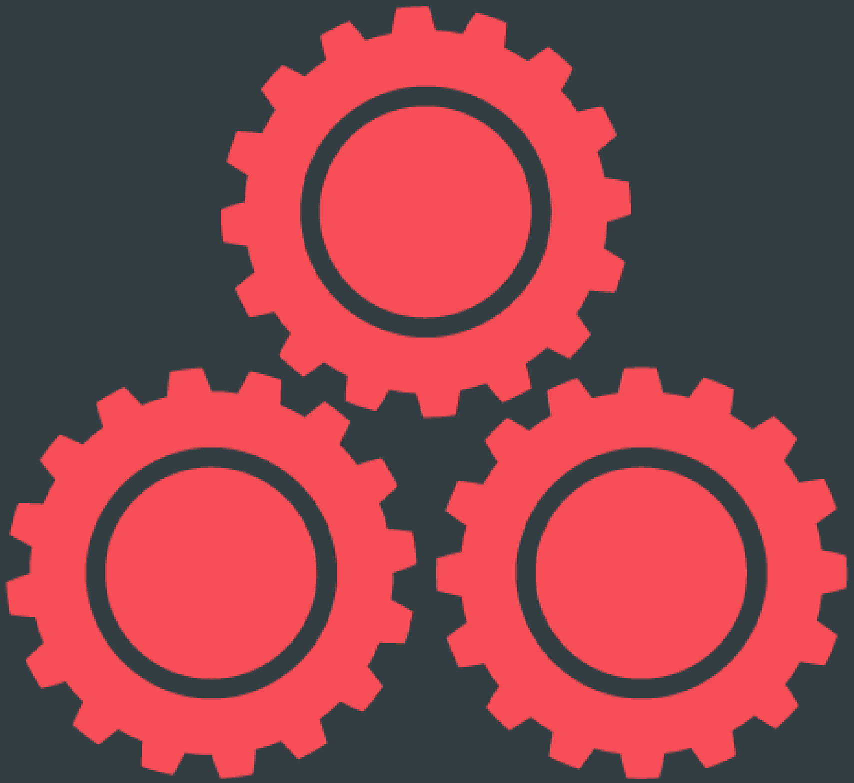


Monitors:

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

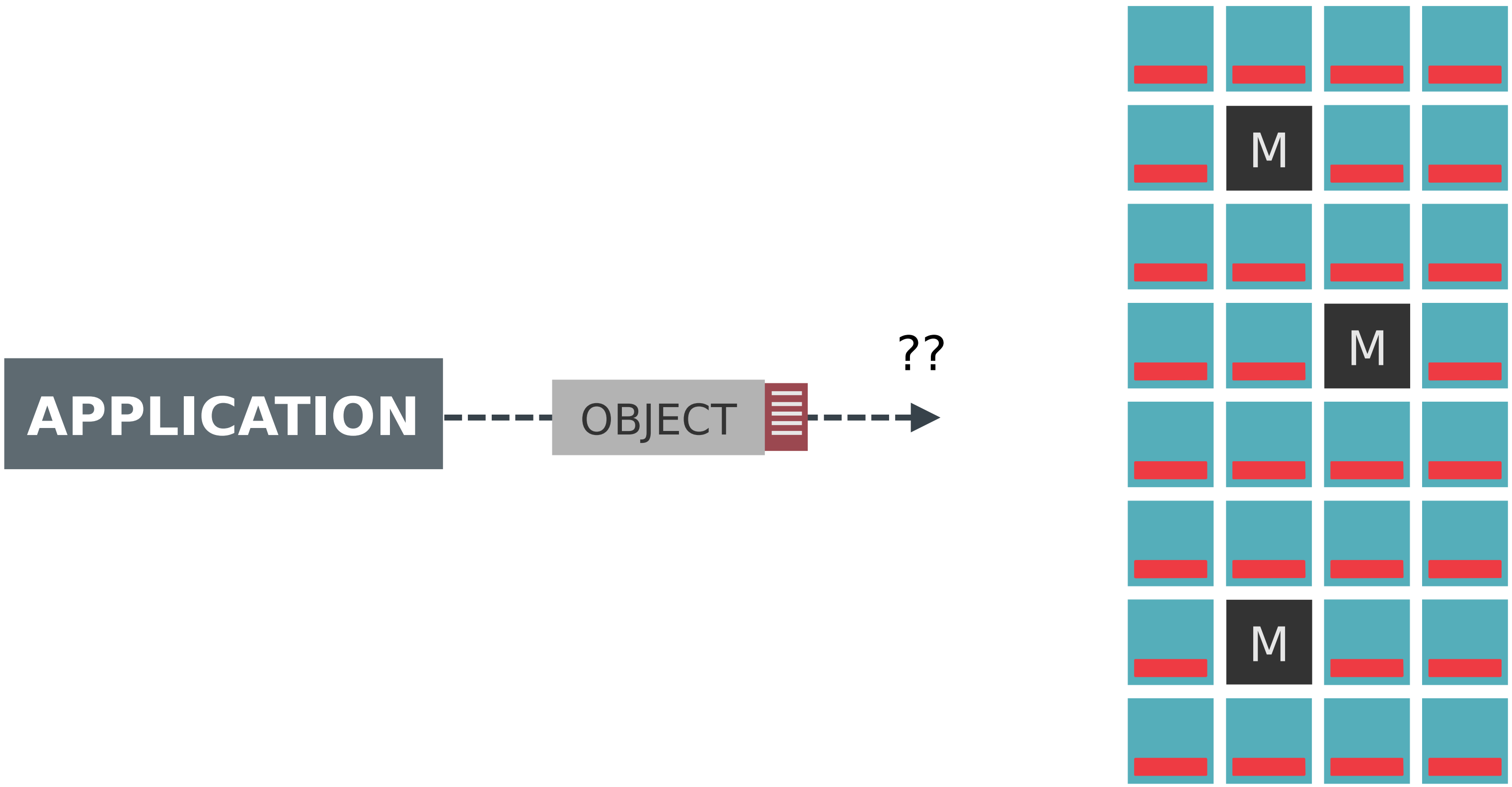
OBJECT STORAGE DAEMONS



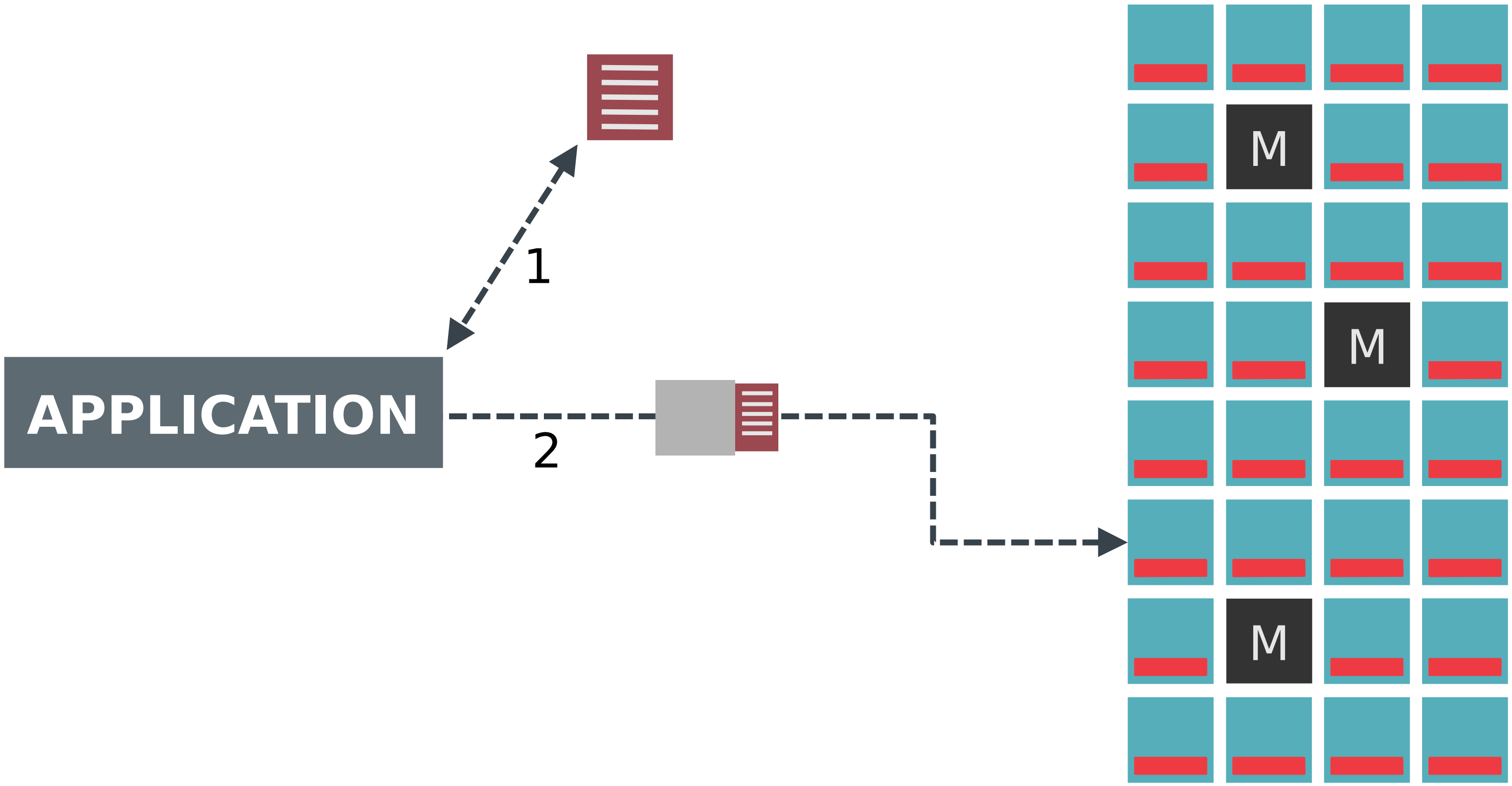


DATA PLACEMENT

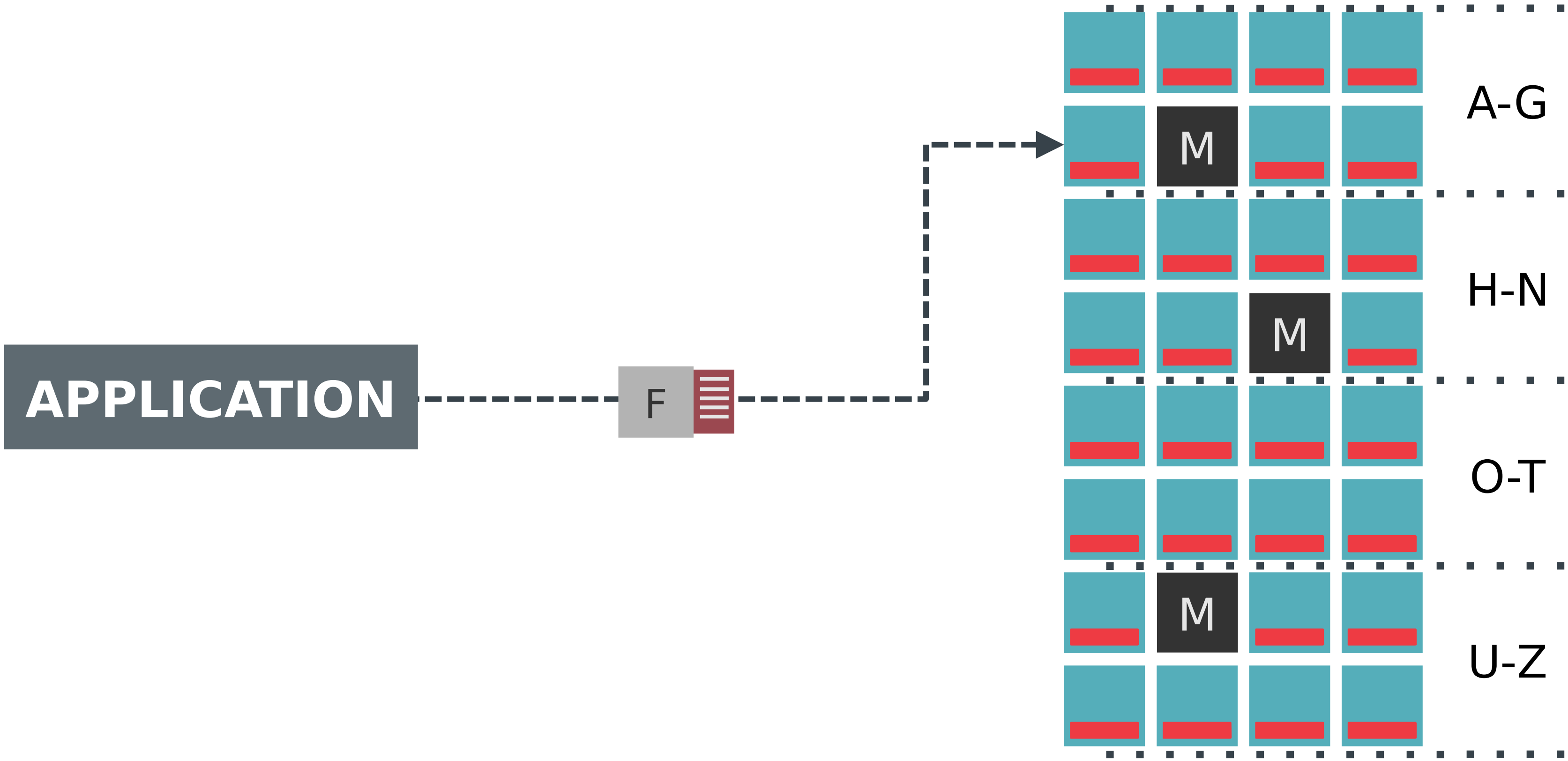
WHERE DO OBJECTS LIVE?



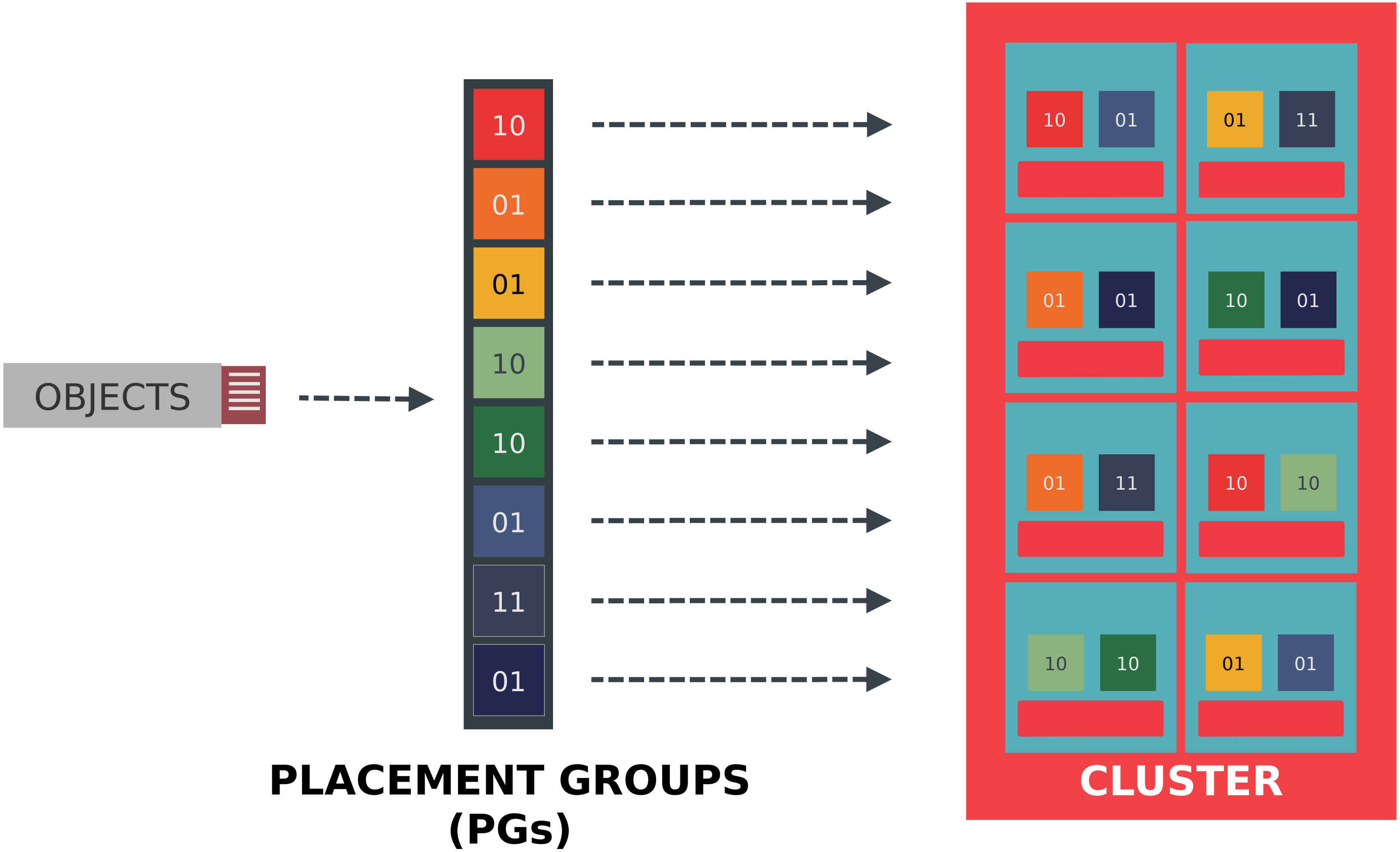
A METADATA SERVER?



CALCULATED PLACEMENT

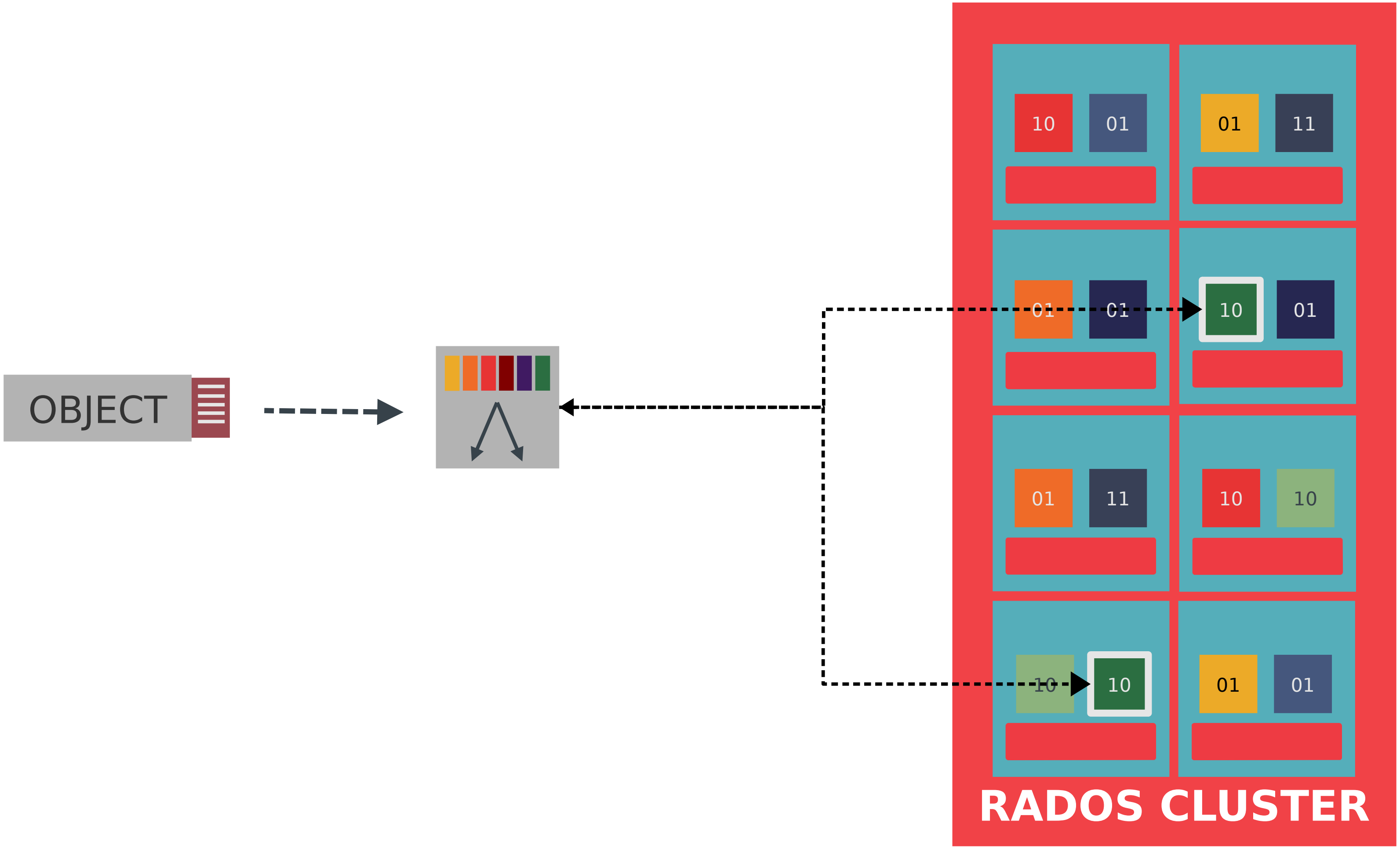


CRUSH

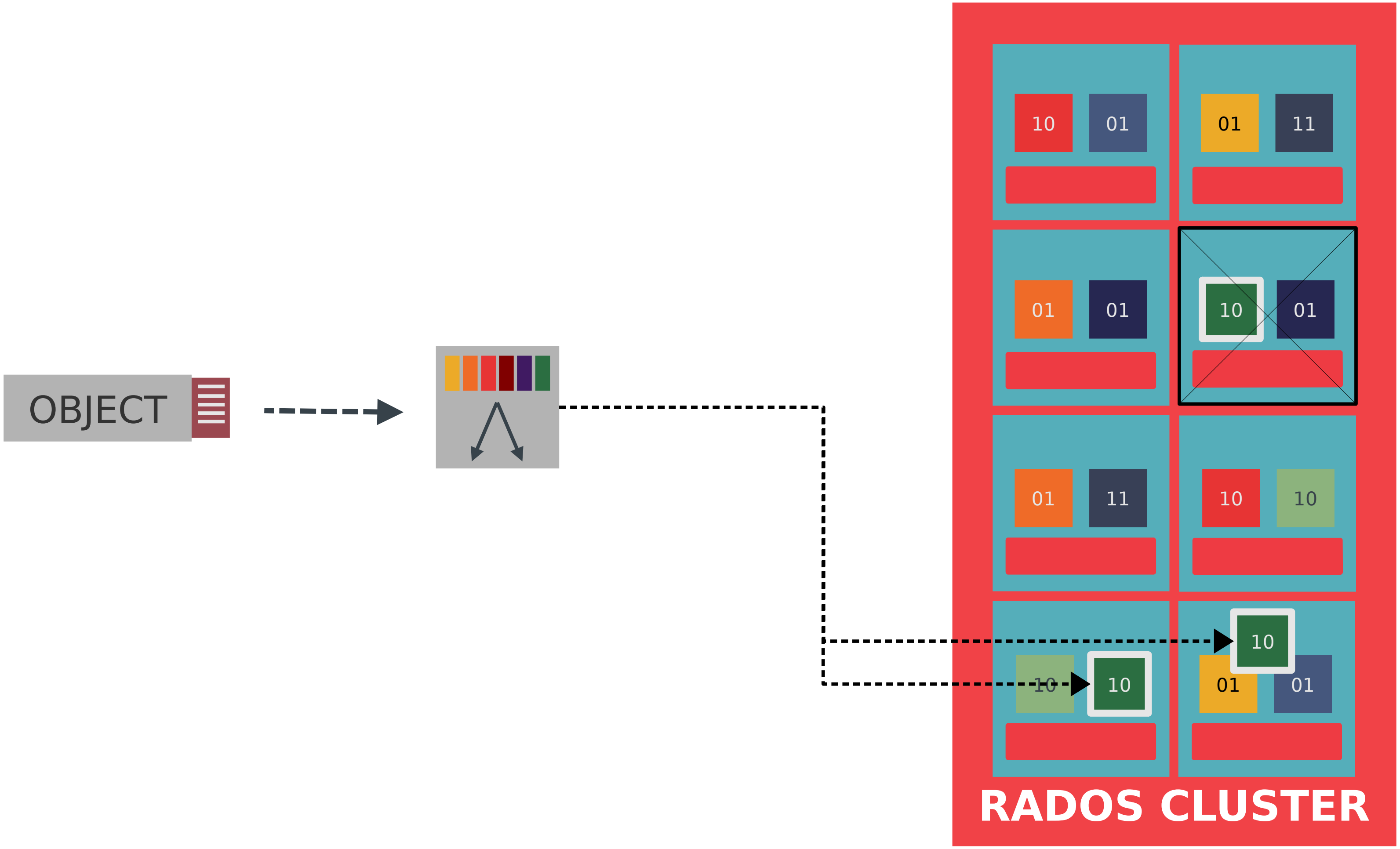


OBJECTS
10
01
01
10
10
01
11
01
PLACEMENT GROUPS (PGs)
10
01
01
11
01
01
10
01
01
11
10
10
10
10
01
01
CLUSTER

CRUSH IS A QUICK CALCULATION

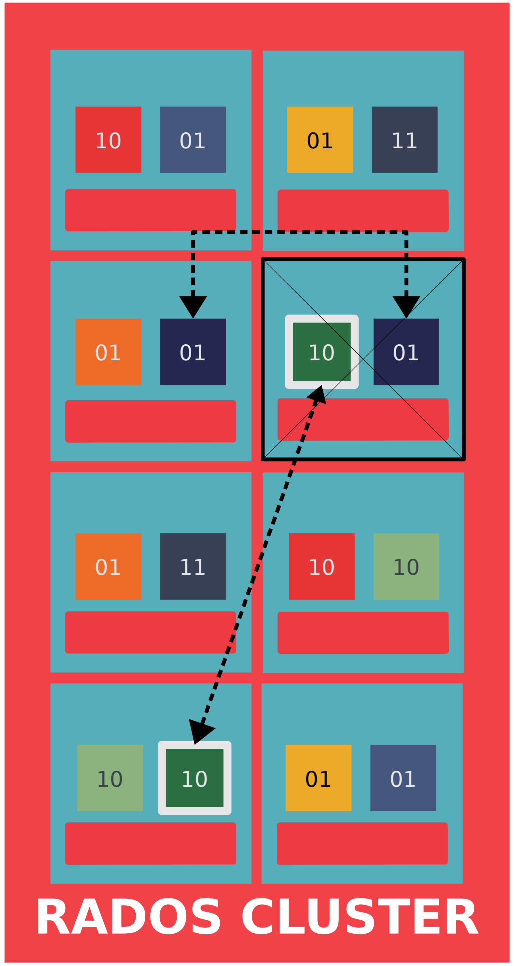


CRUSH AVOIDS FAILED DEVICES

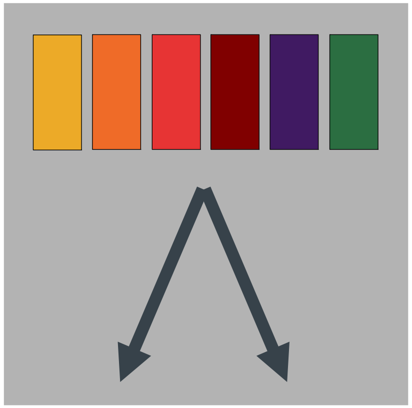


CRUSH: DECLUSTERED PLACEMENT

- 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



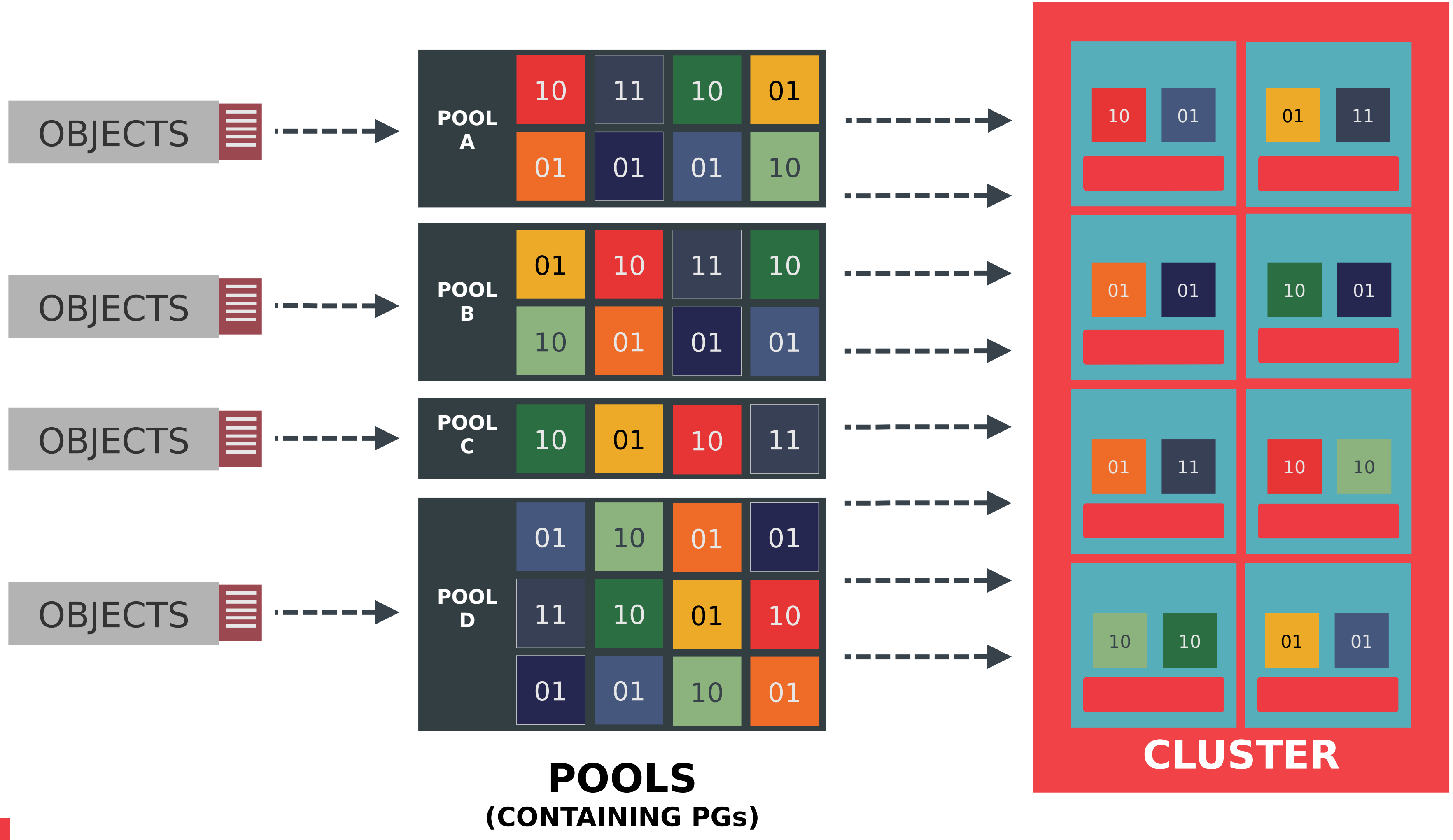
CRUSH: DYNAMIC DATA PLACEMENT



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
  - Weighting

DATA IS ORGANIZED INTO POOLS

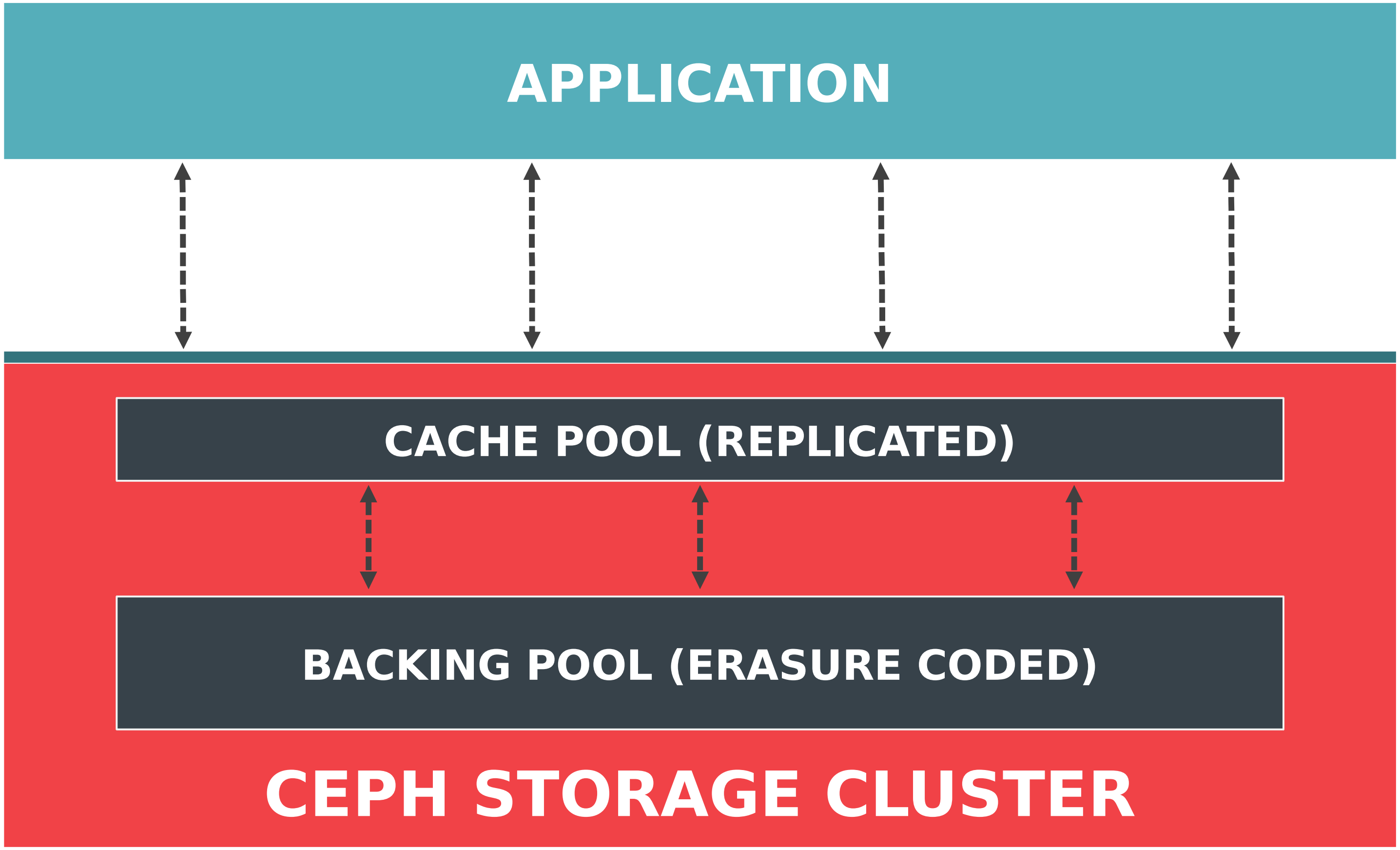


TIERED STORAGE

TWO WAYS TO CACHE

- Within each OSD
  - Combine SSD and HDD for 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

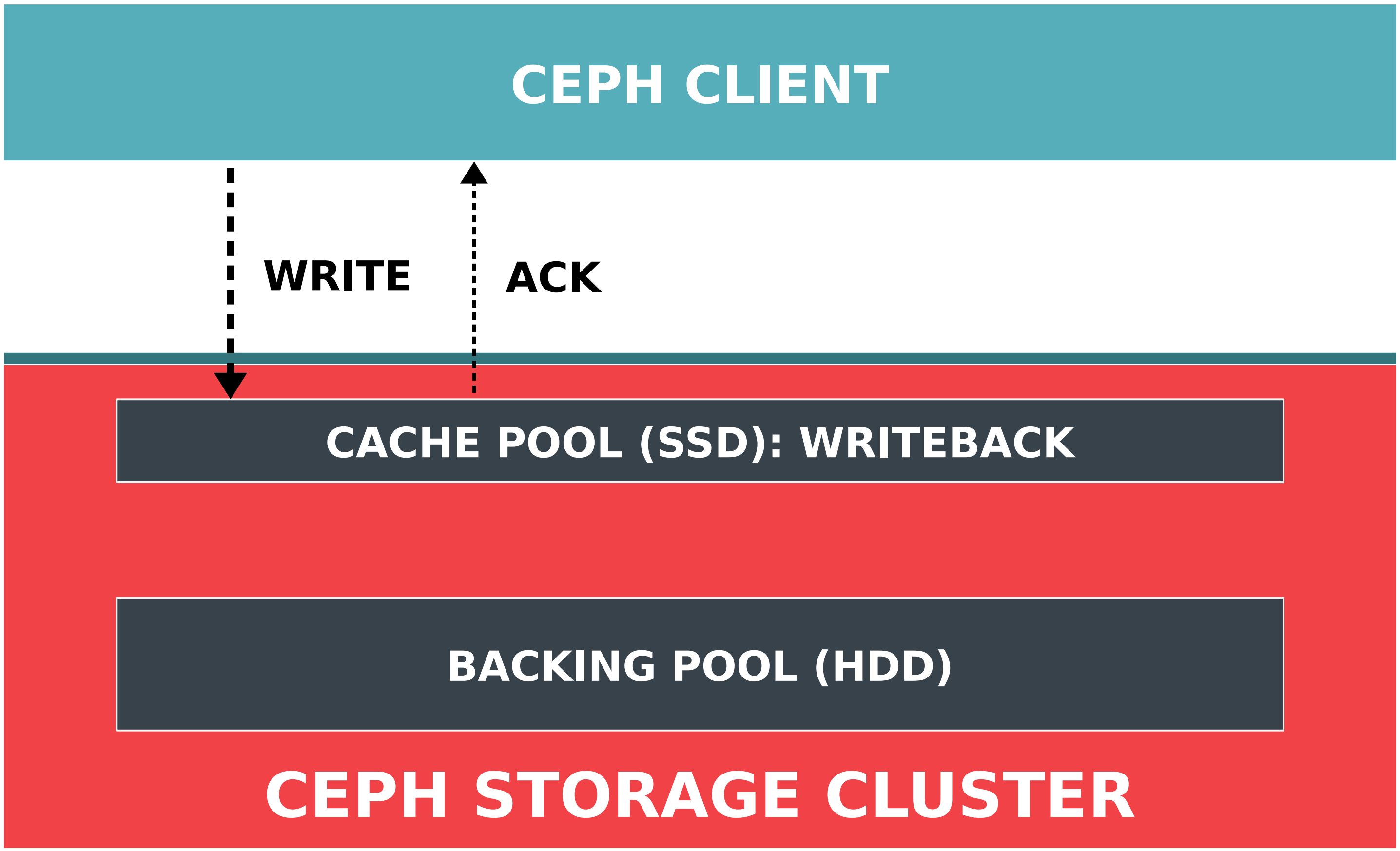
TIERED STORAGE



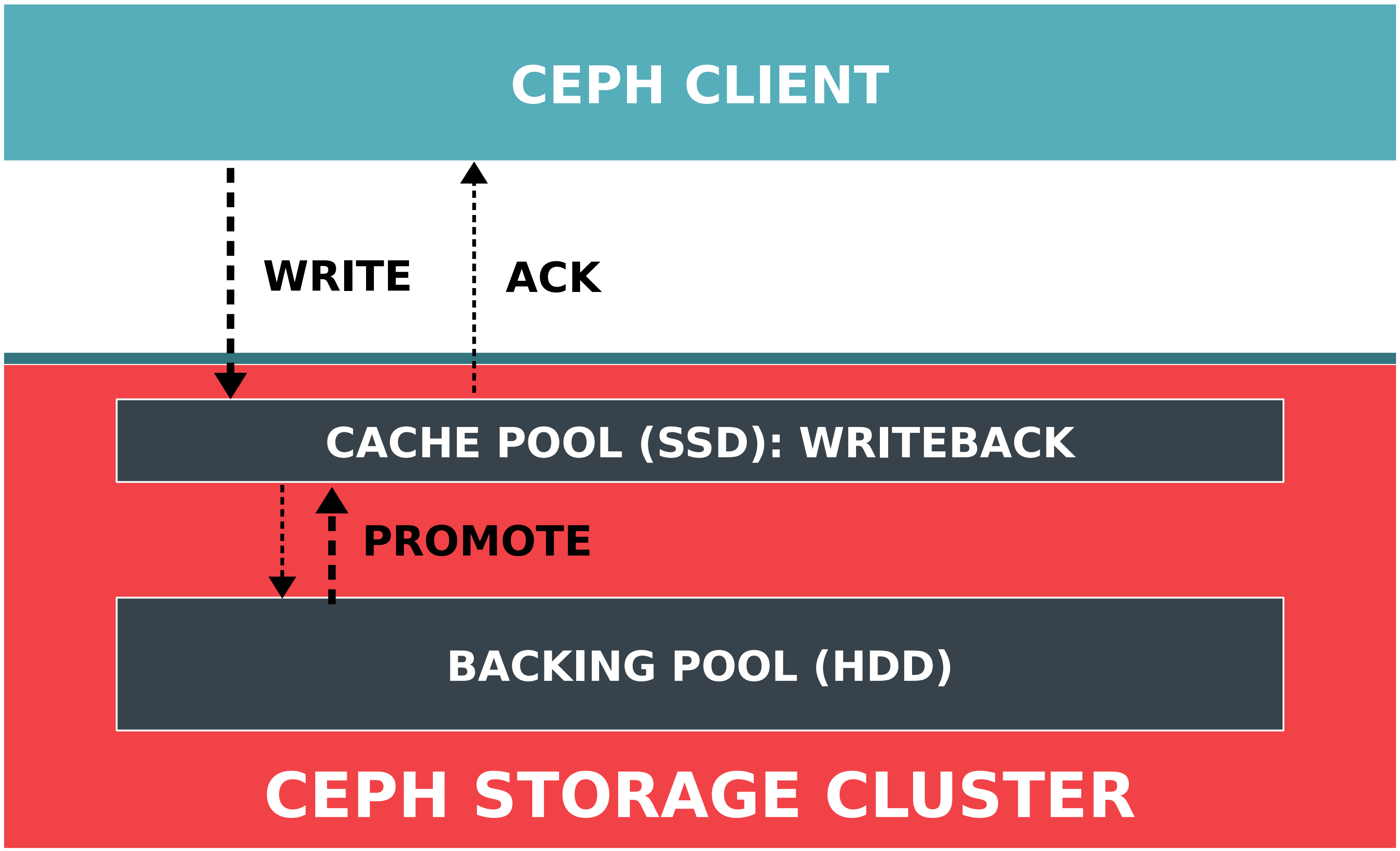
RADOS TIERING PRINCIPLES

- 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 to SSDs devices/hosts only
- librados clients adapt to tiering topology
  - Transparently direct requests accordingly
    - e.g., to cache
  - No changes to RBD, RGW, CephFS, etc.

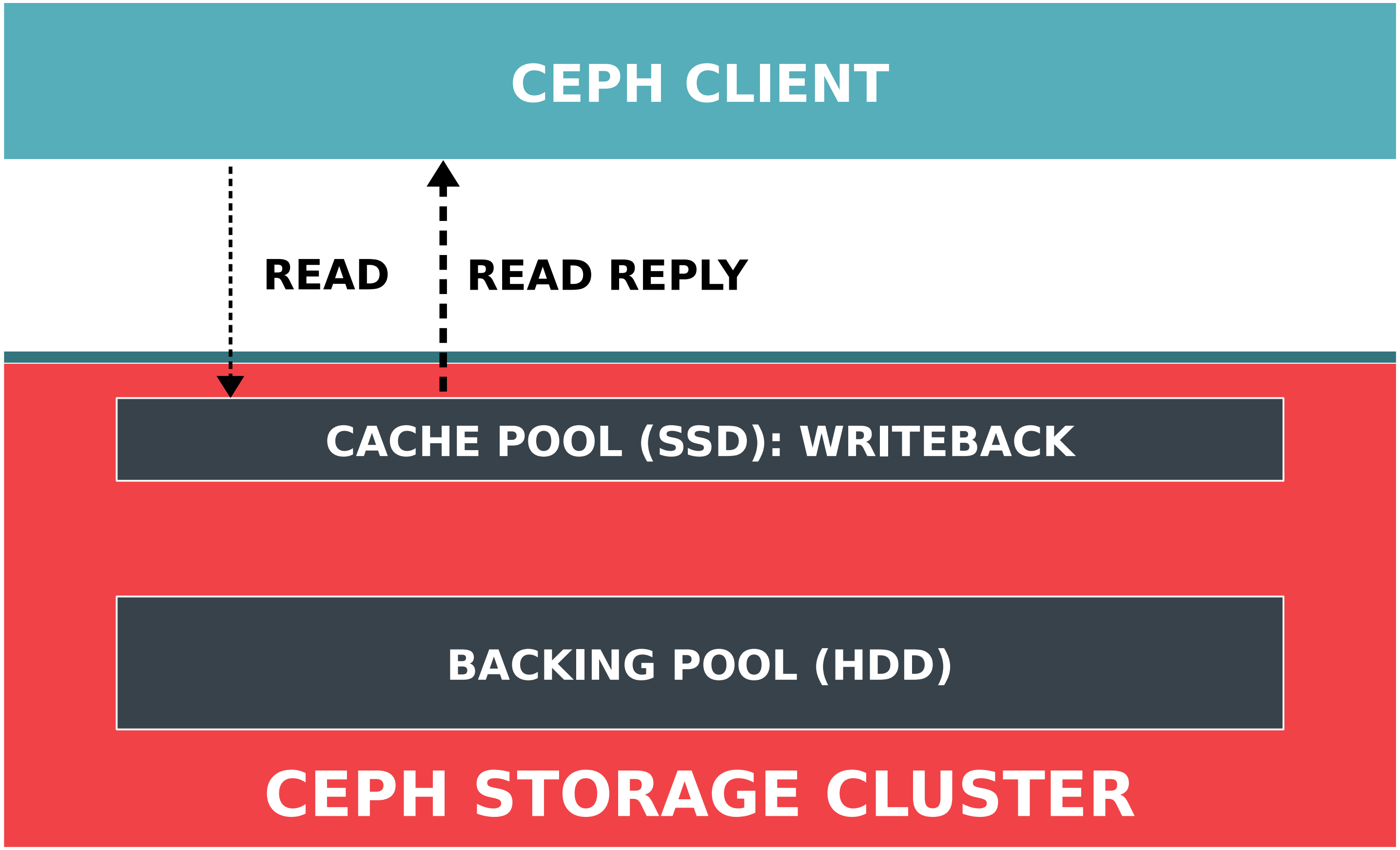
WRITE INTO CACHE POOL



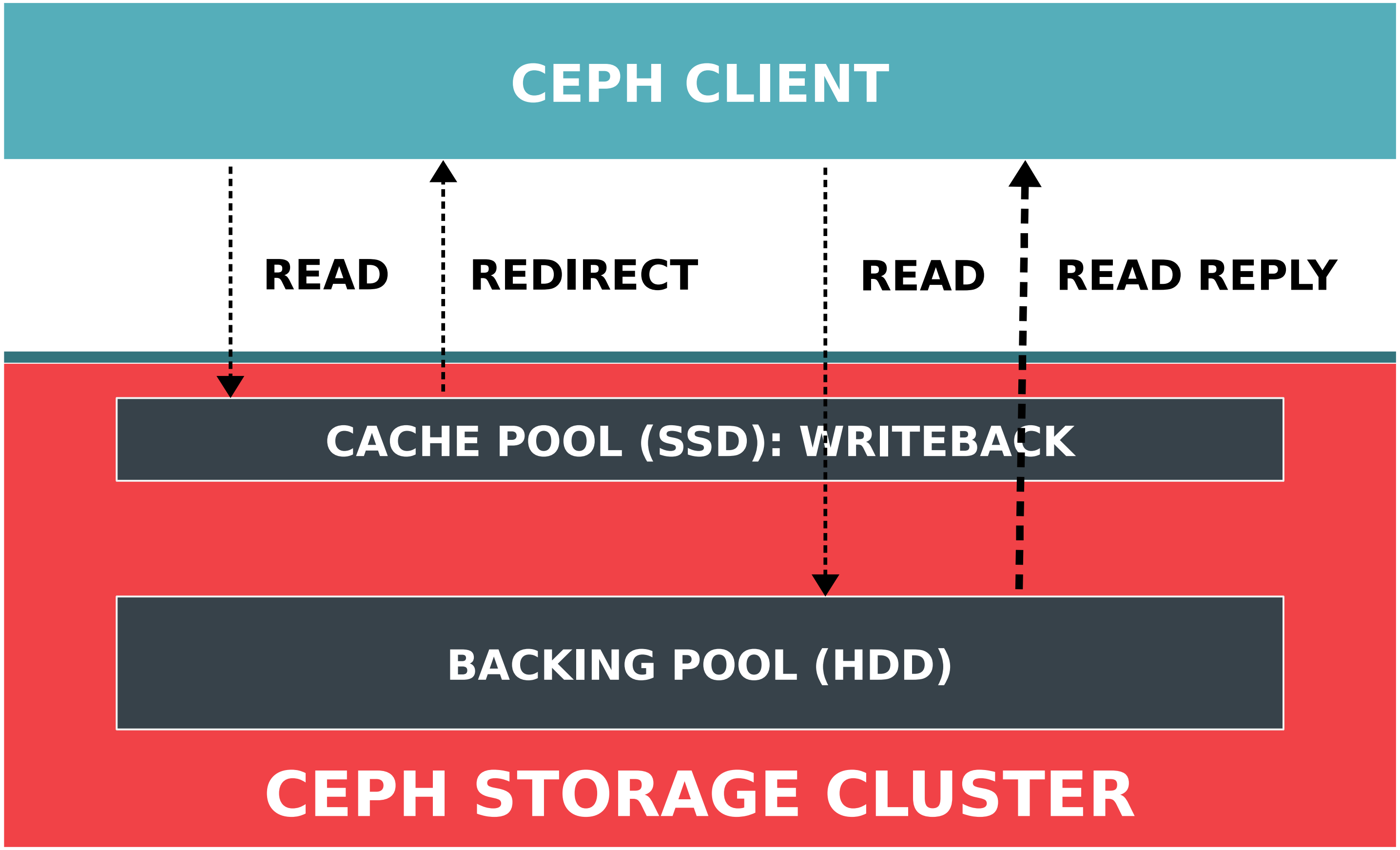
WRITE INTO CACHE POOL



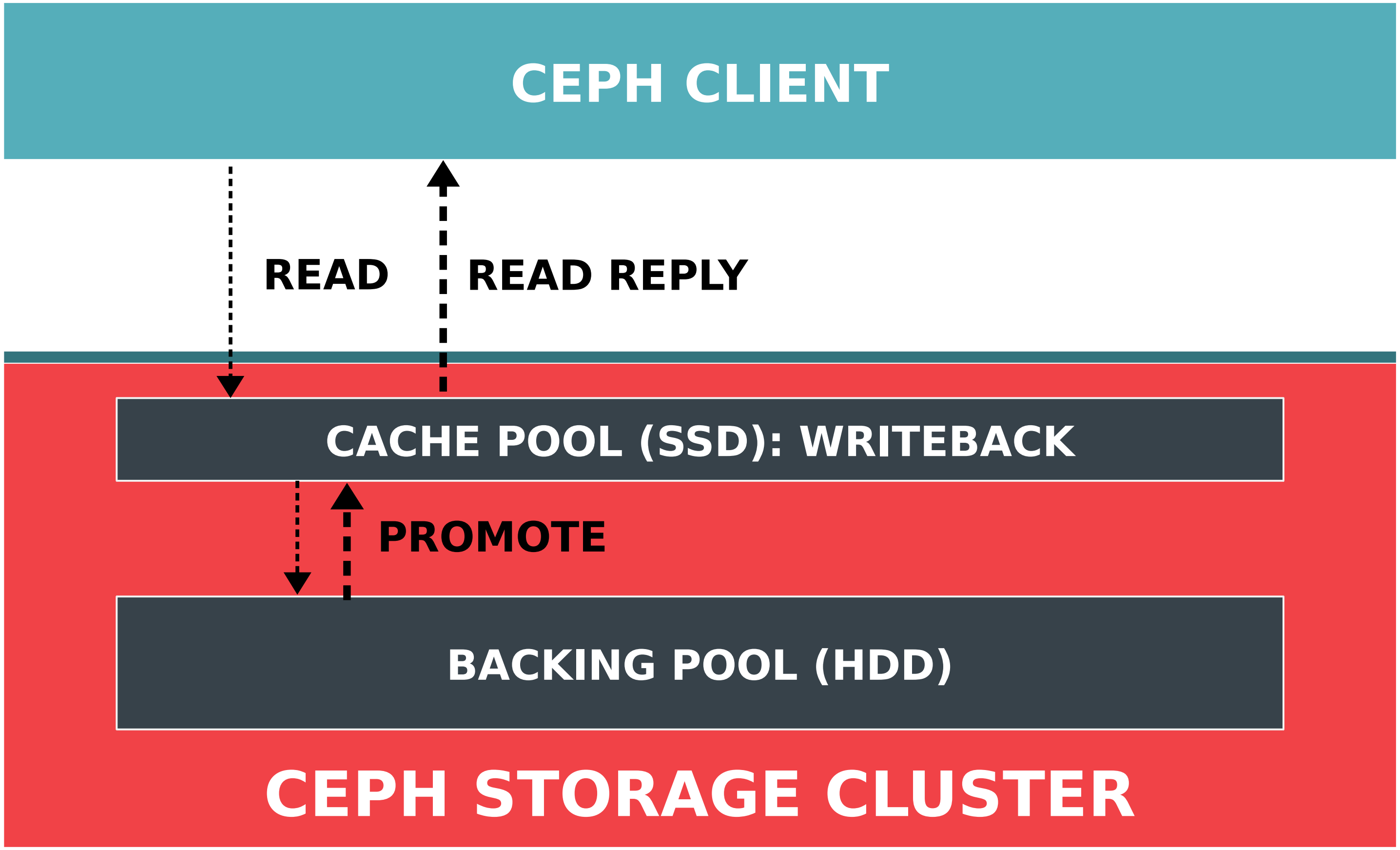
READ (CACHE HIT)



READ (CACHE MISS)



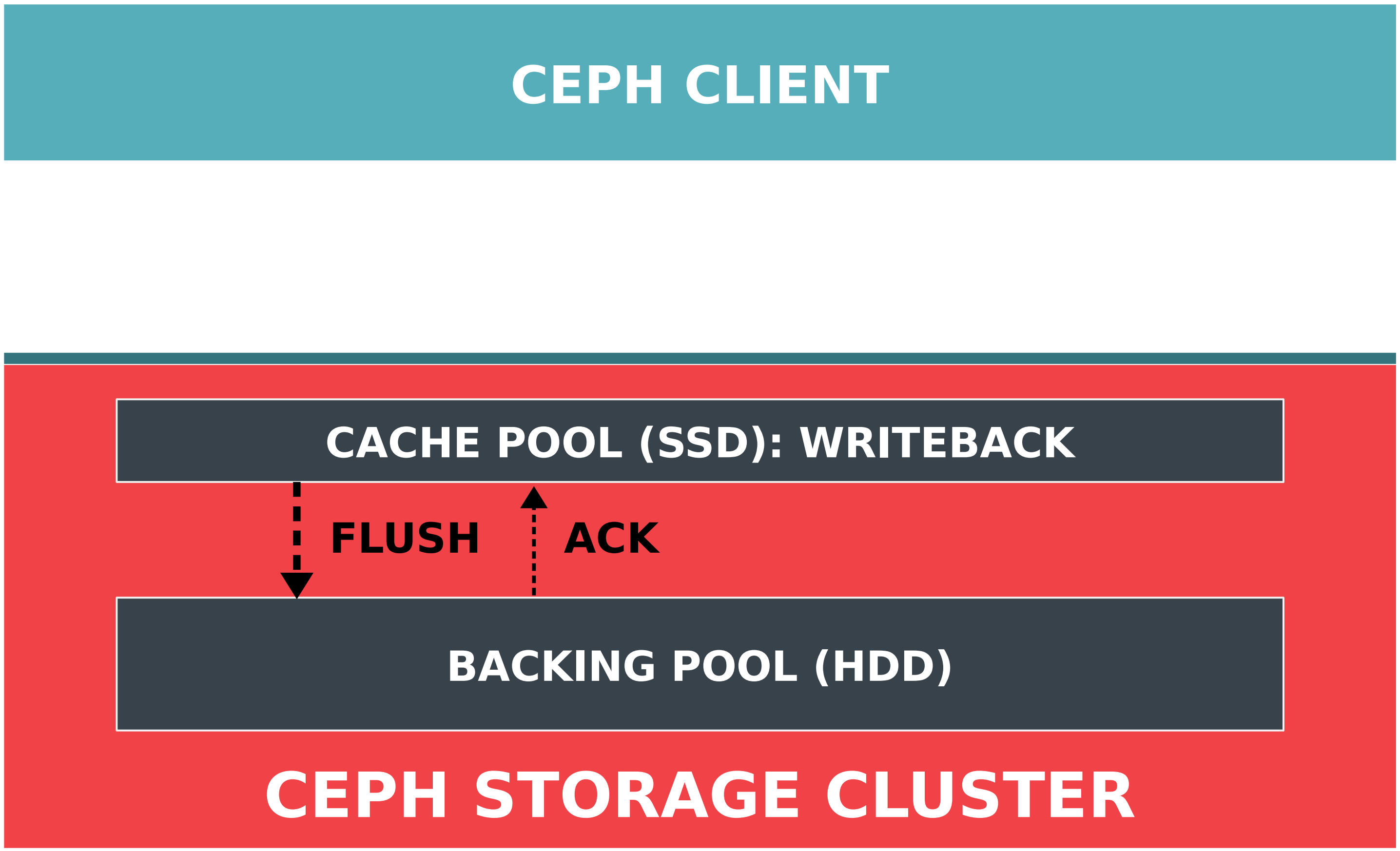
READ (CACHE MISS)



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%)
  - Maintain most recent N filters on disk
- Estimate temperature
  - Has object been accessed in any of the last N periods?
  - ...in how many of them?
  - Informs flush/evict decision
- Estimate “recency”
  - How many periods since the object hasn't been accessed?
  - Informs read miss behavior: promote vs redirect

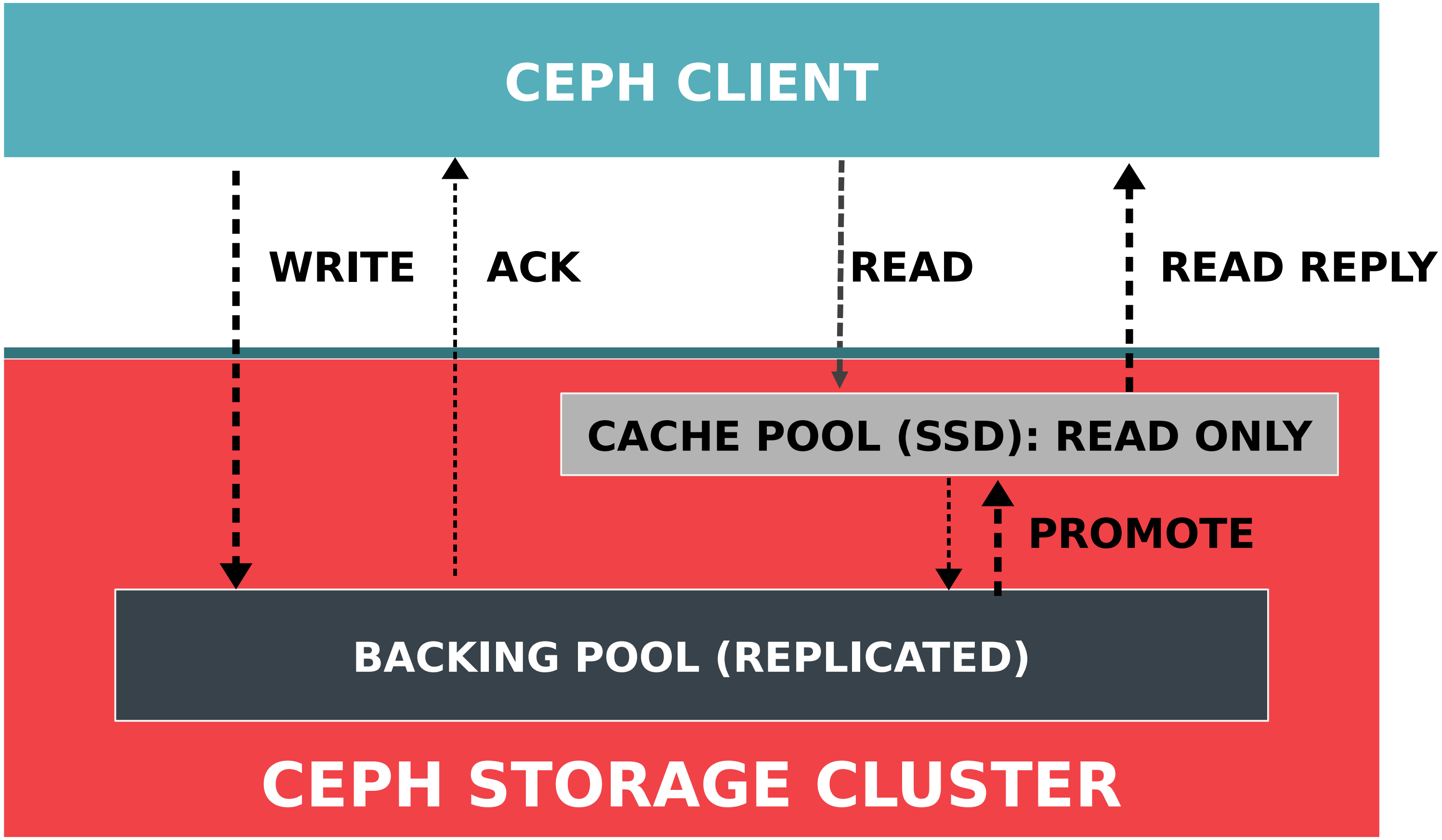
AGENT: FLUSH COLD DATA



TIERING AGENT

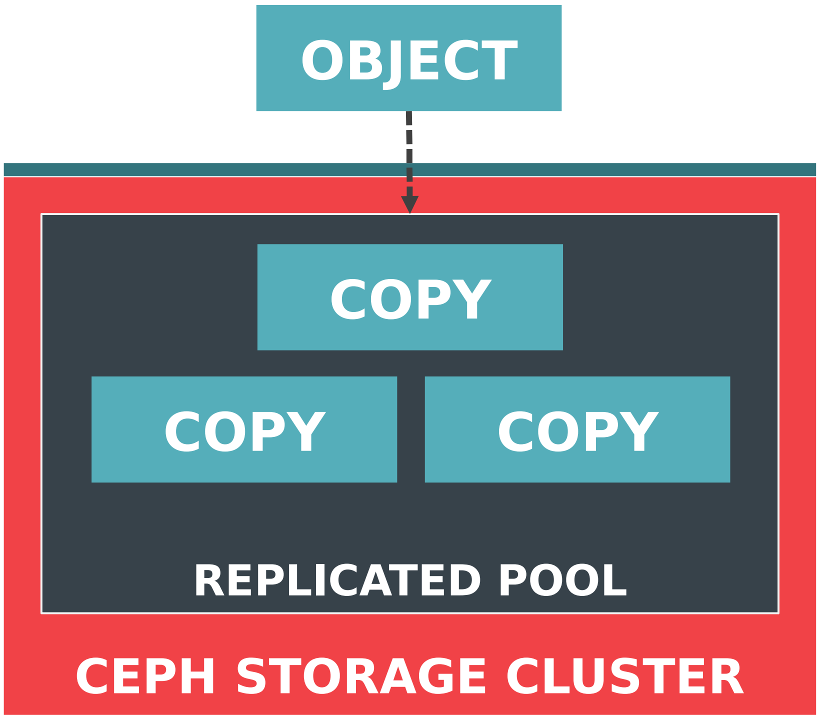
- Each PG has an internal tiering **agent**
  - Manages PG based on administrator defined policy
- Flush **dirty** objects
  - When pool reaches target **dirty ratio**
  - Tries to select cold objects
  - Marks objects **clean** when they have been written back to the base pool
- Evict clean objects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

READ ONLY CACHE TIER



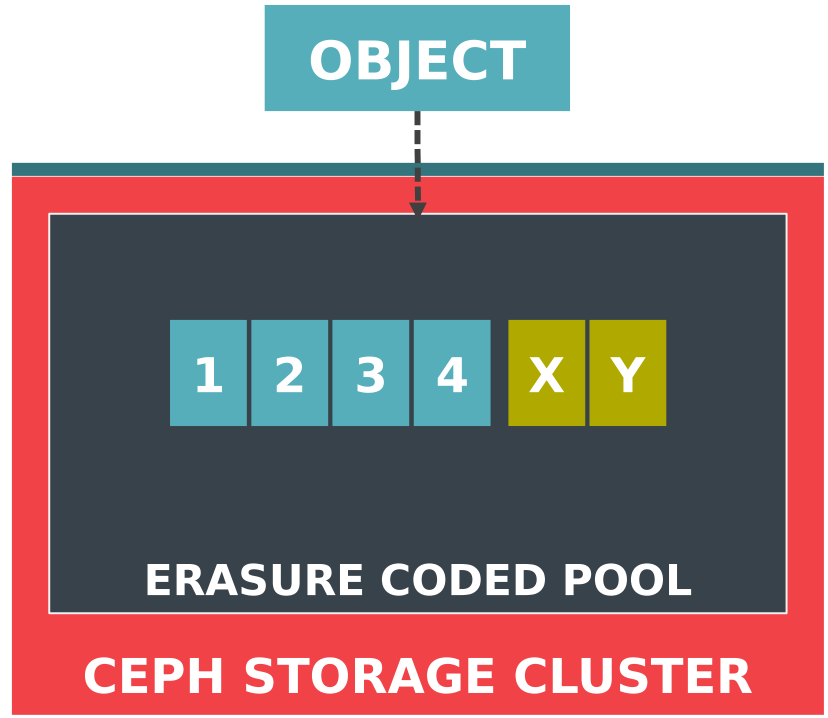
ERASURE CODING

ERASURE CODING



Full copies of stored objects

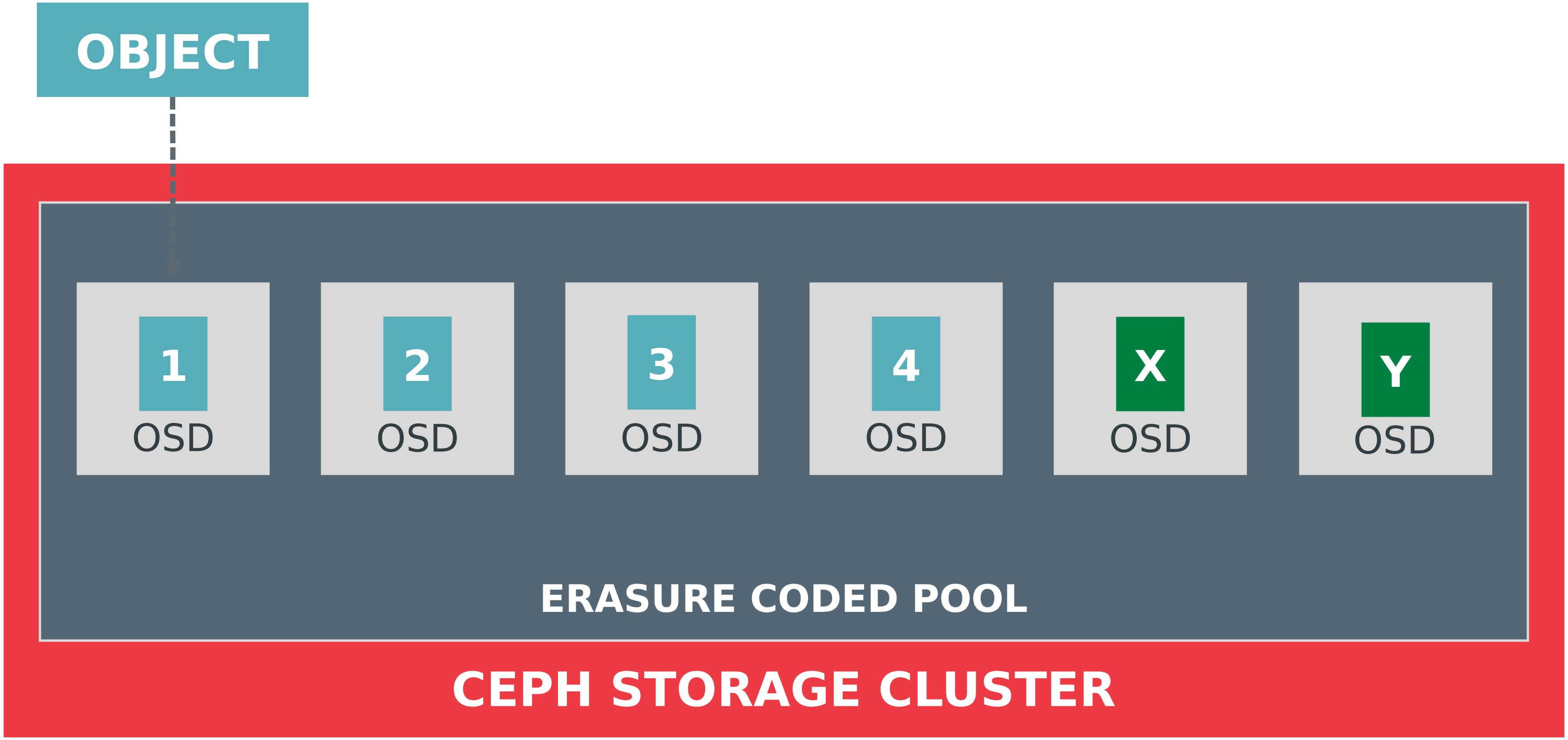
- Very high durability
- 3x (200% overhead)
- Quicker recovery



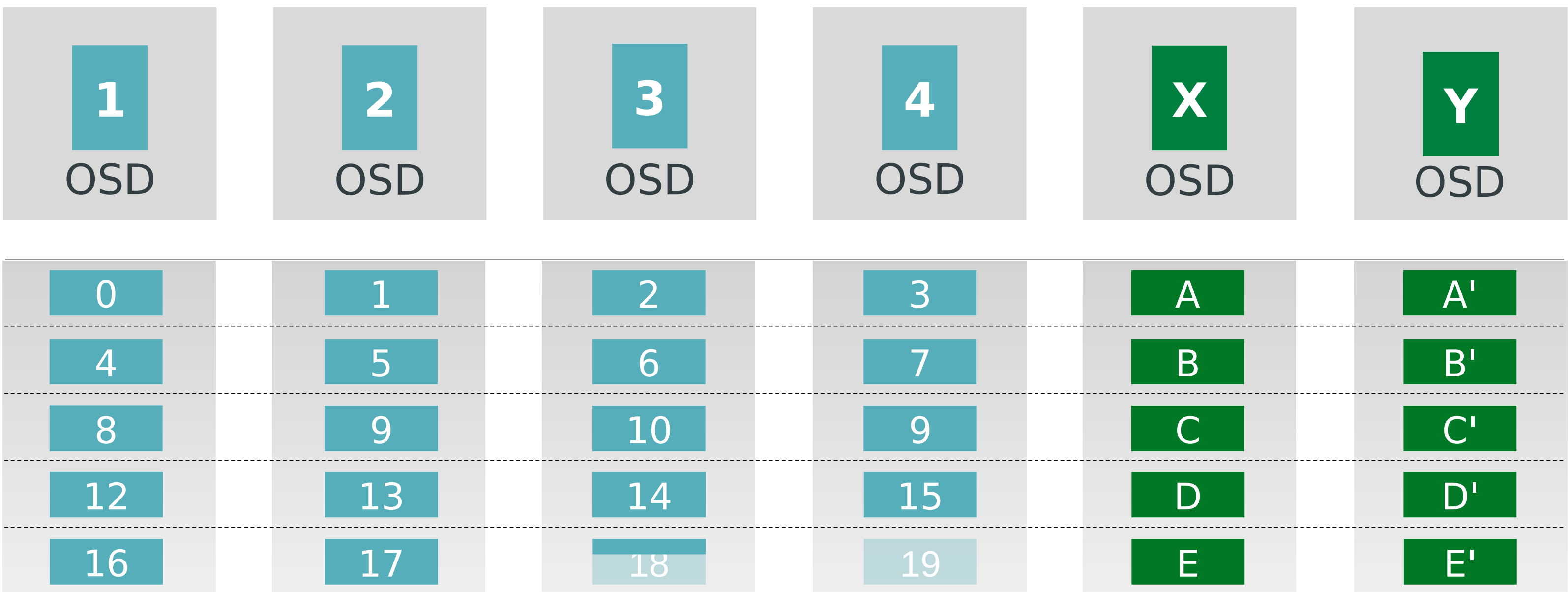
One copy plus parity

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

ERASURE CODING SHARDS

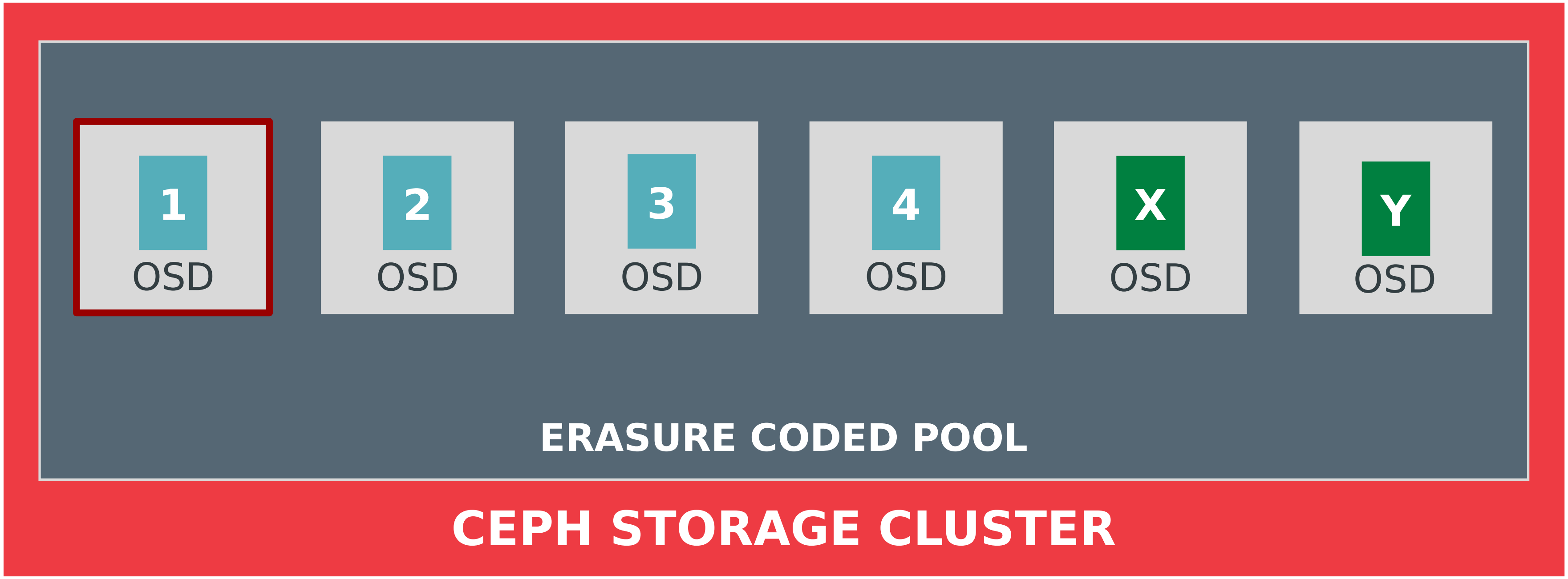


ERASURE CODING SHARDS

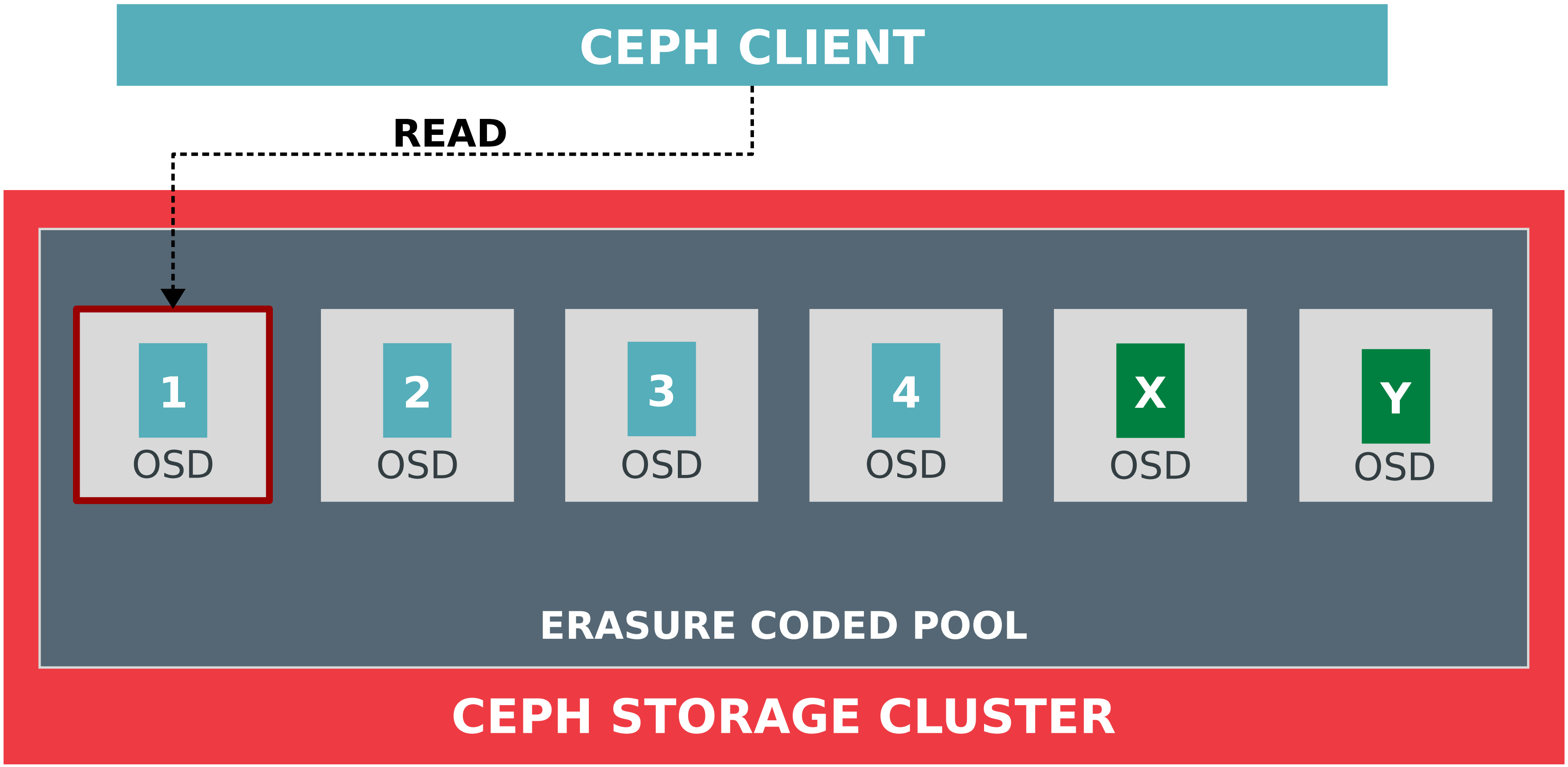


- Variable stripe size
- Zero-fill shards (logically) in partial tail stripe

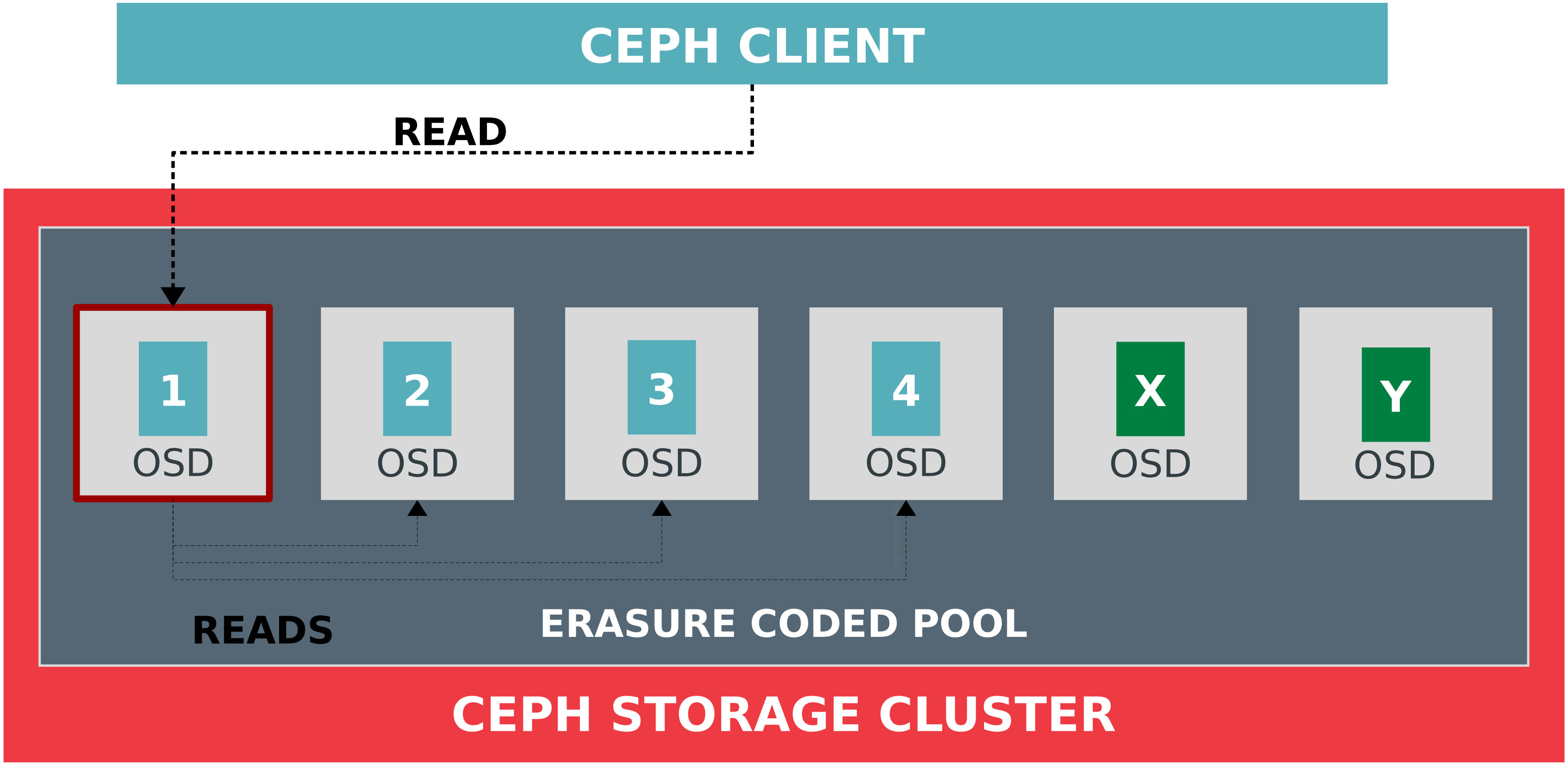
PRIMARY


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

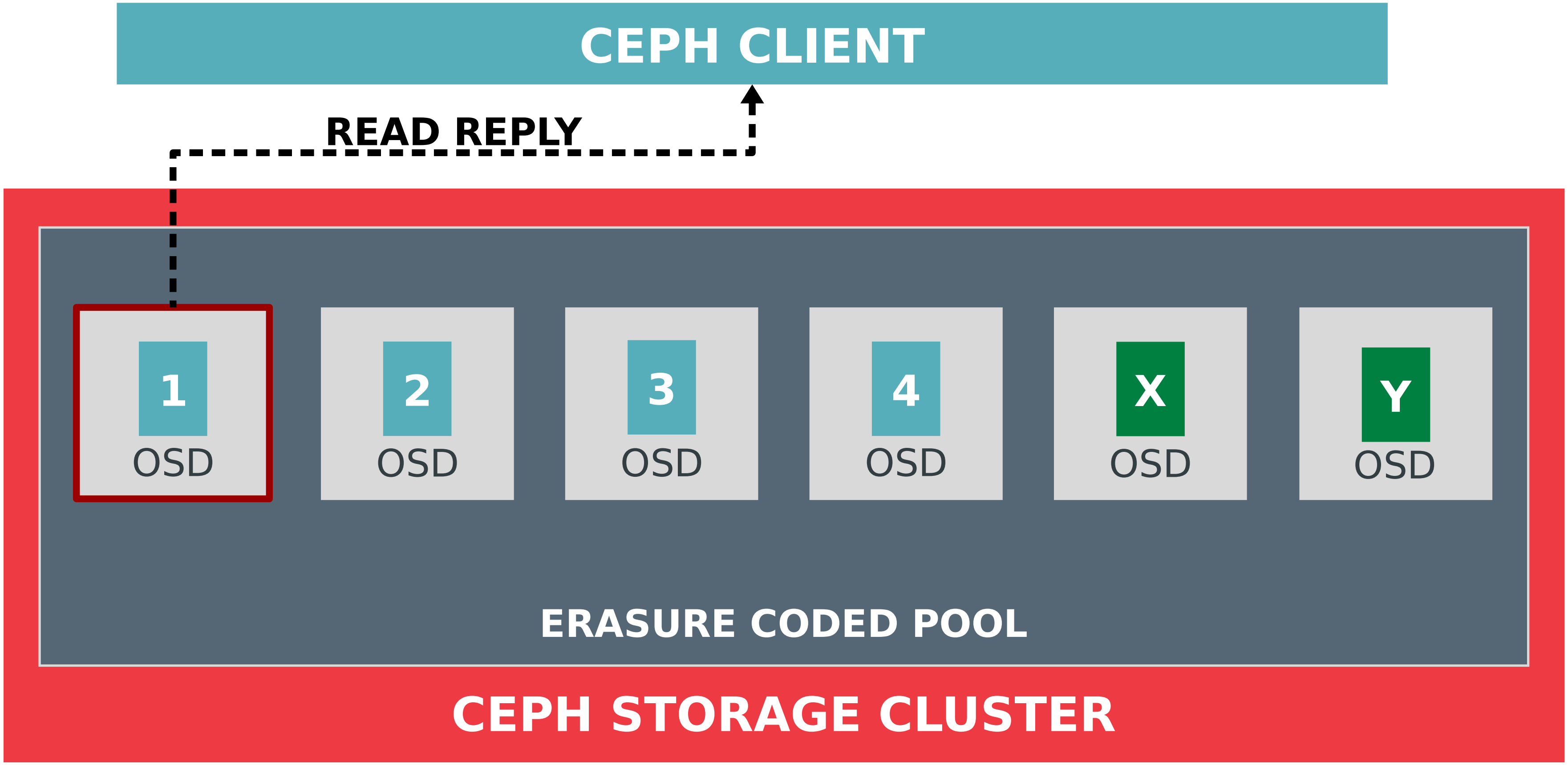
EC READ


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

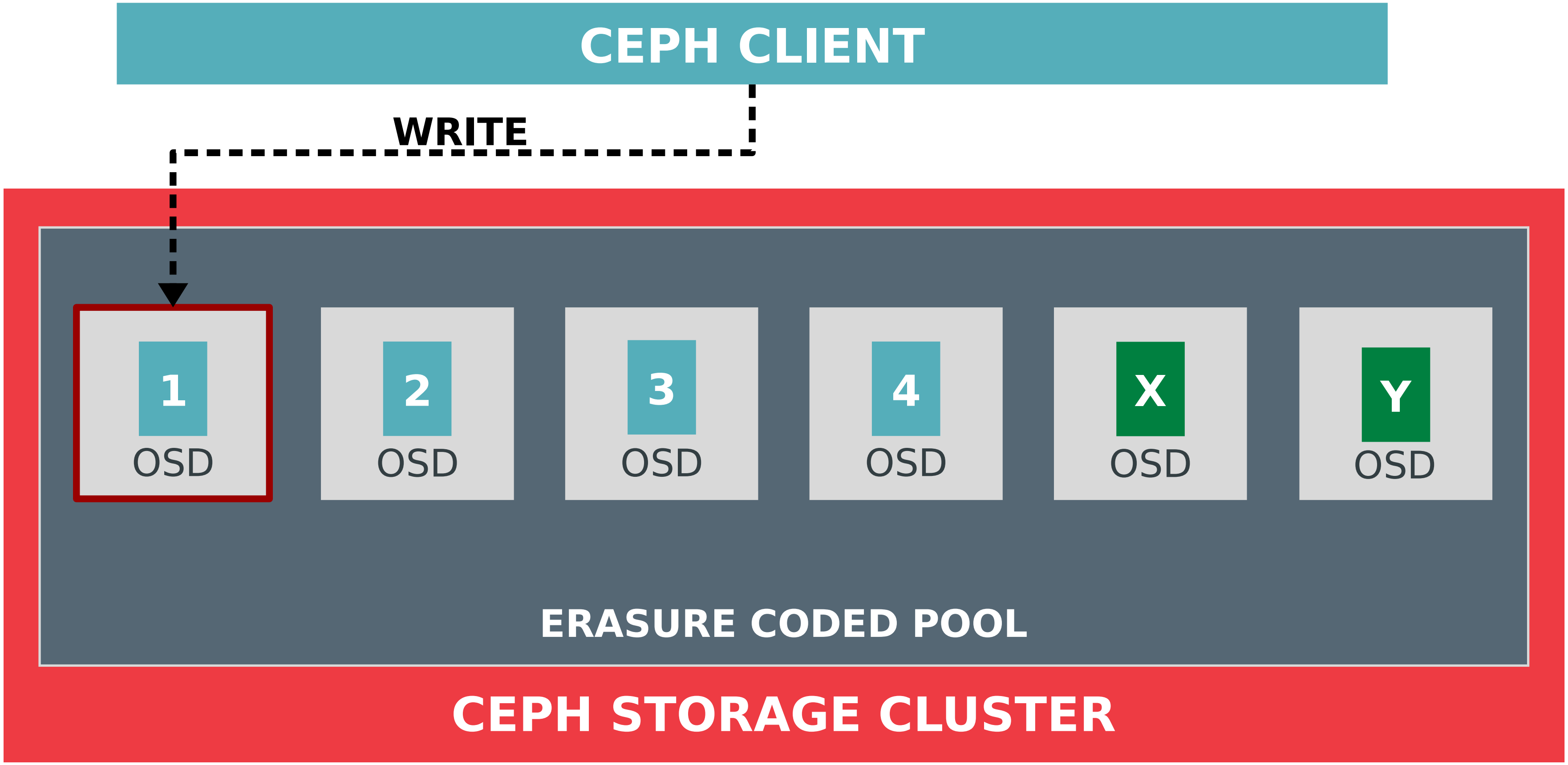
EC READ


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

EC READ

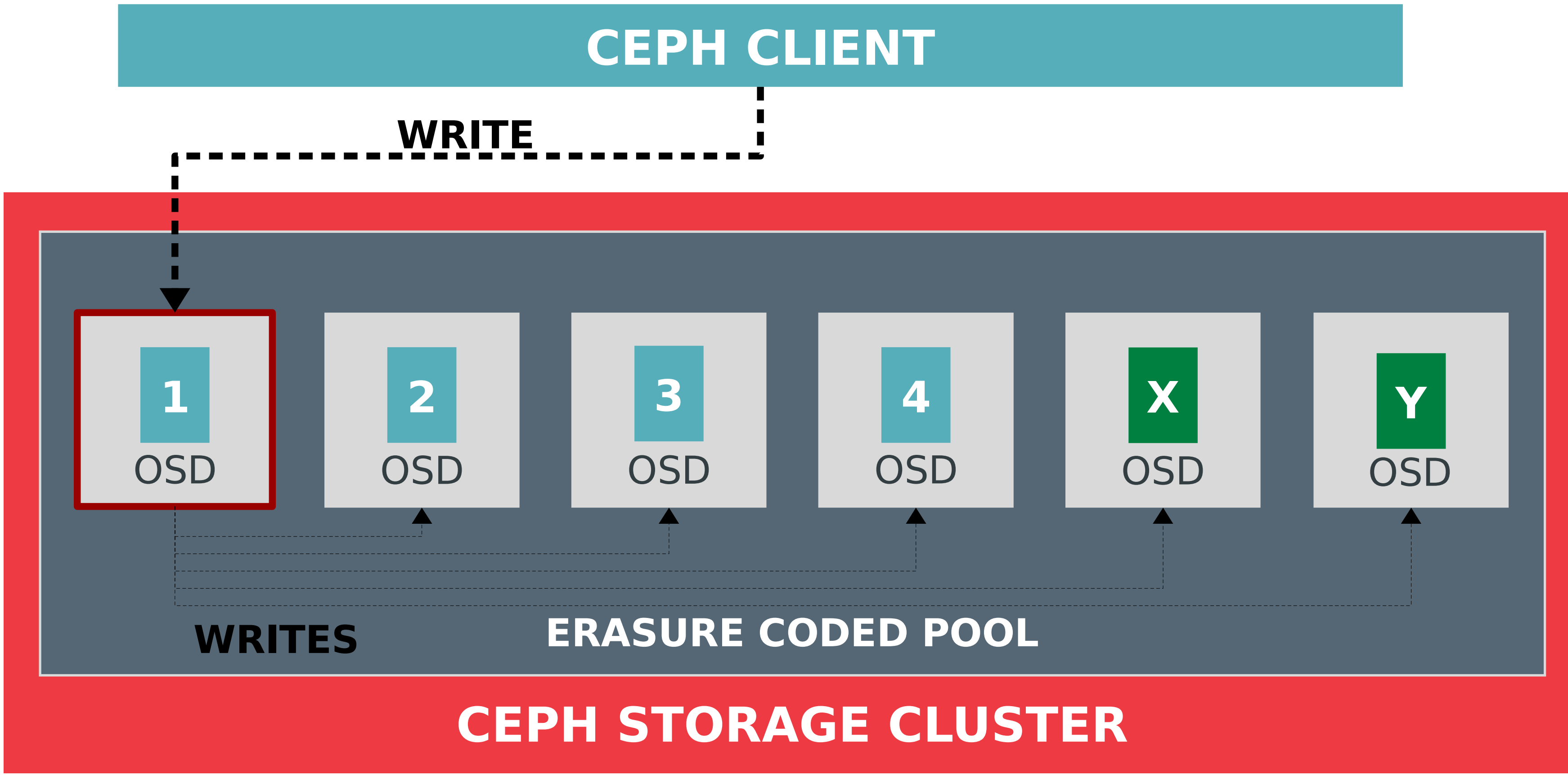

CEPH CLIENT
READ REPLY
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
ERASURE CODED POOL
CEPH STORAGE CLUSTER

EC WRITE

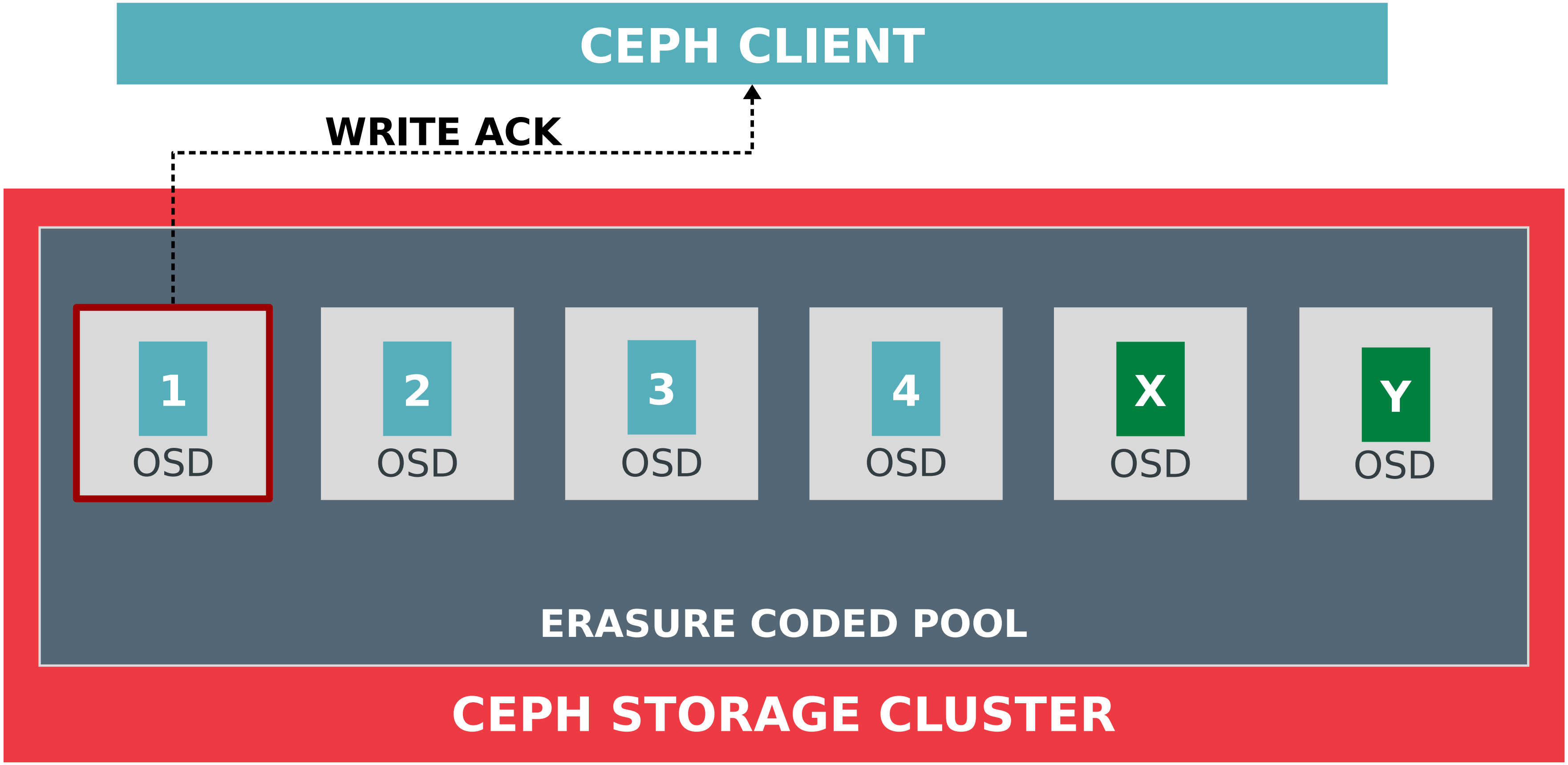


CEPH CLIENT
WRITE
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
ERASURE CODED POOL
CEPH STORAGE CLUSTER

EC WRITE

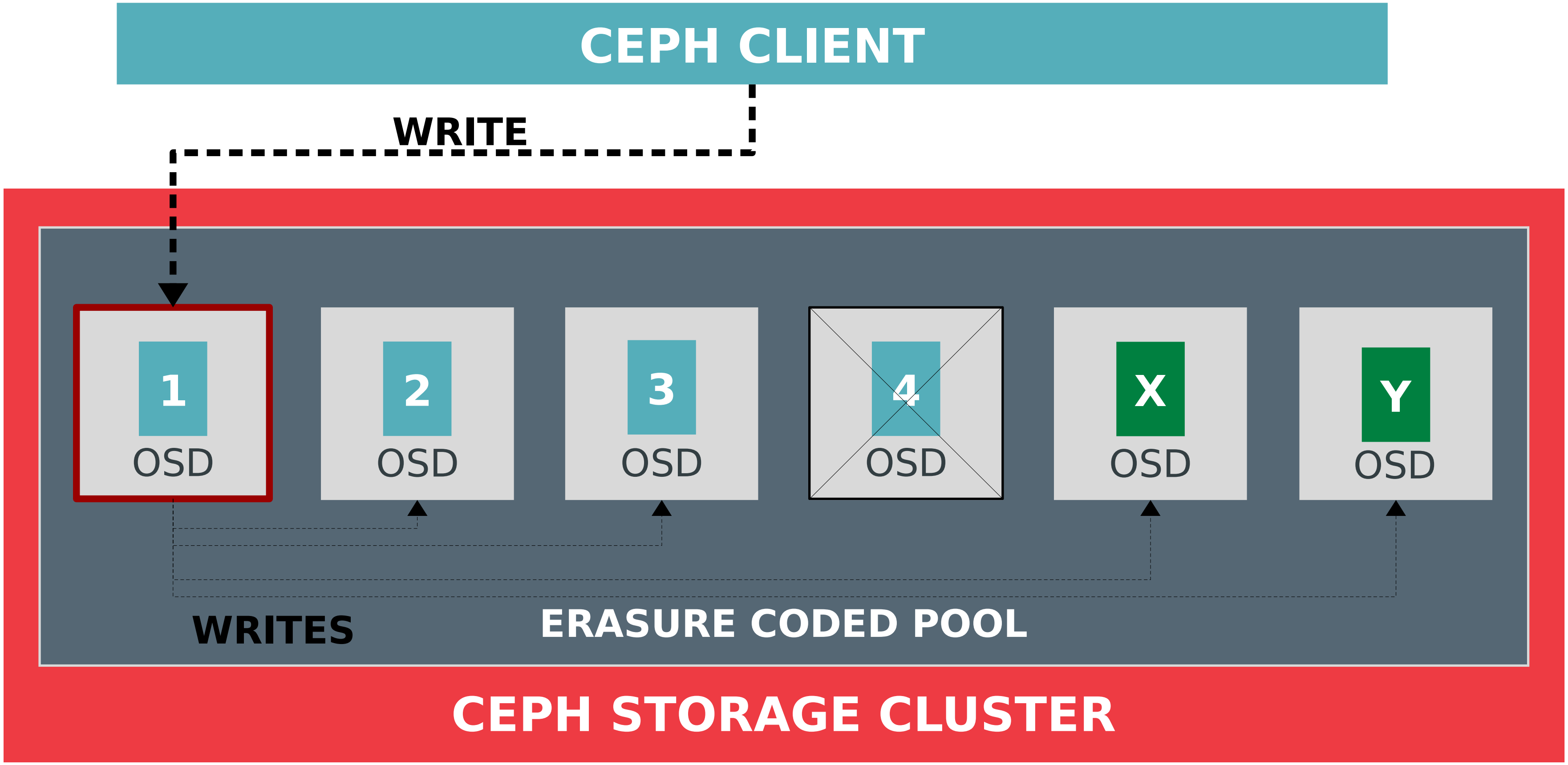

CEPH CLIENT
WRITE
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
WRITES
ERASURE CODED POOL
CEPH STORAGE CLUSTER

EC WRITE

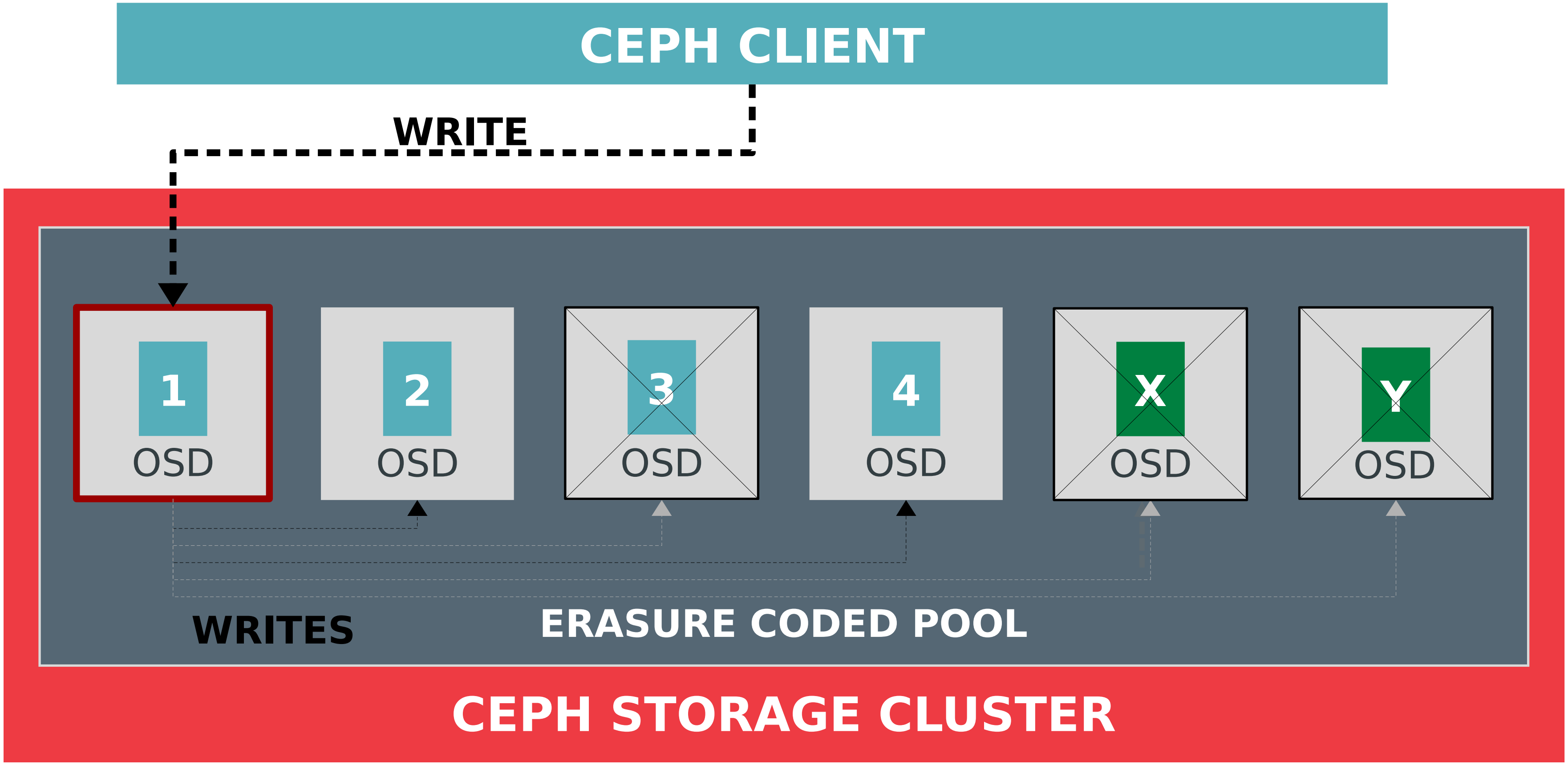


CEPH CLIENT
WRITE ACK
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
ERASURE CODED POOL
CEPH STORAGE CLUSTER

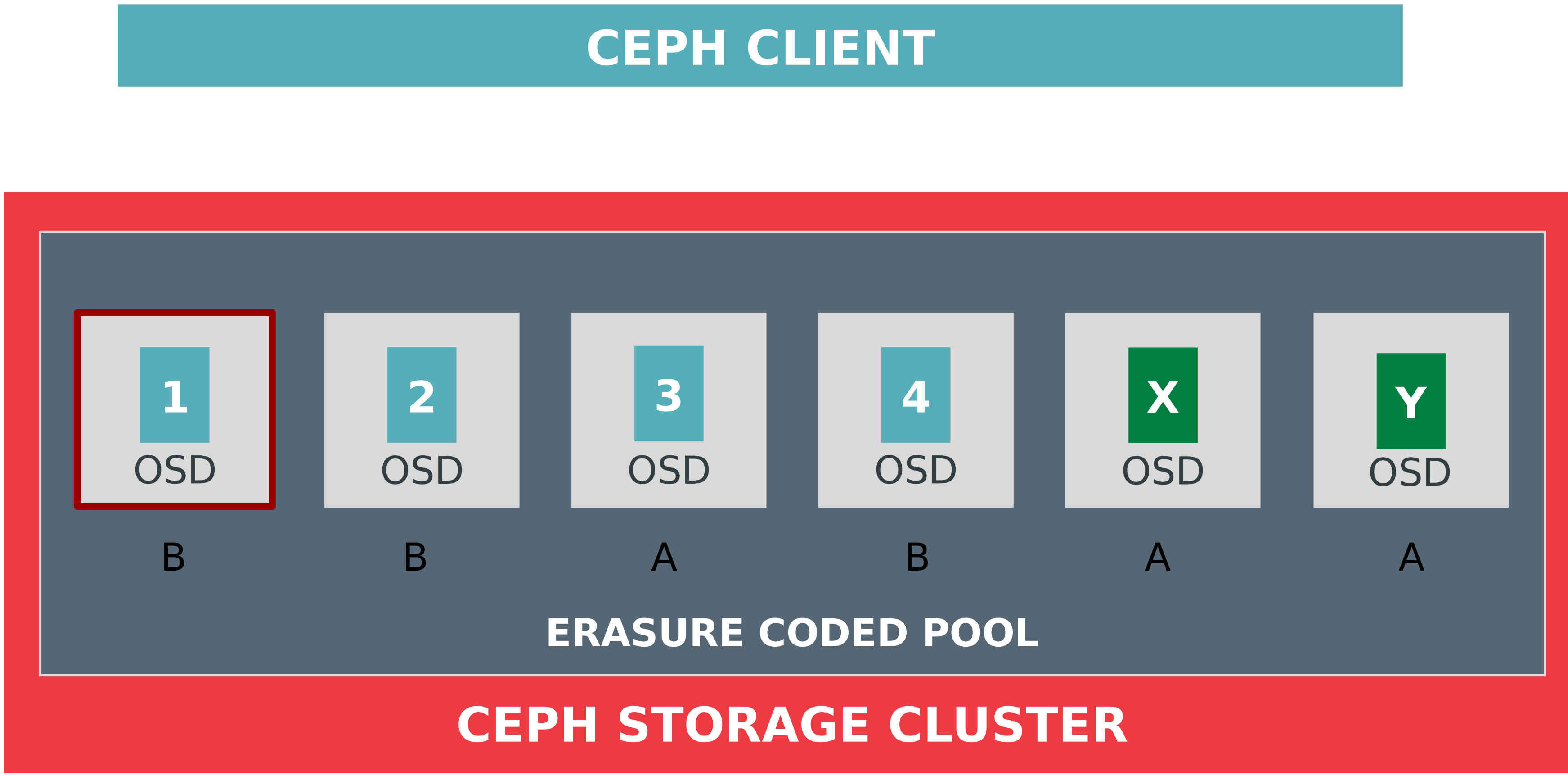
EC WRITE: DEGRADED



EC WRITE: PARTIAL FAILURE



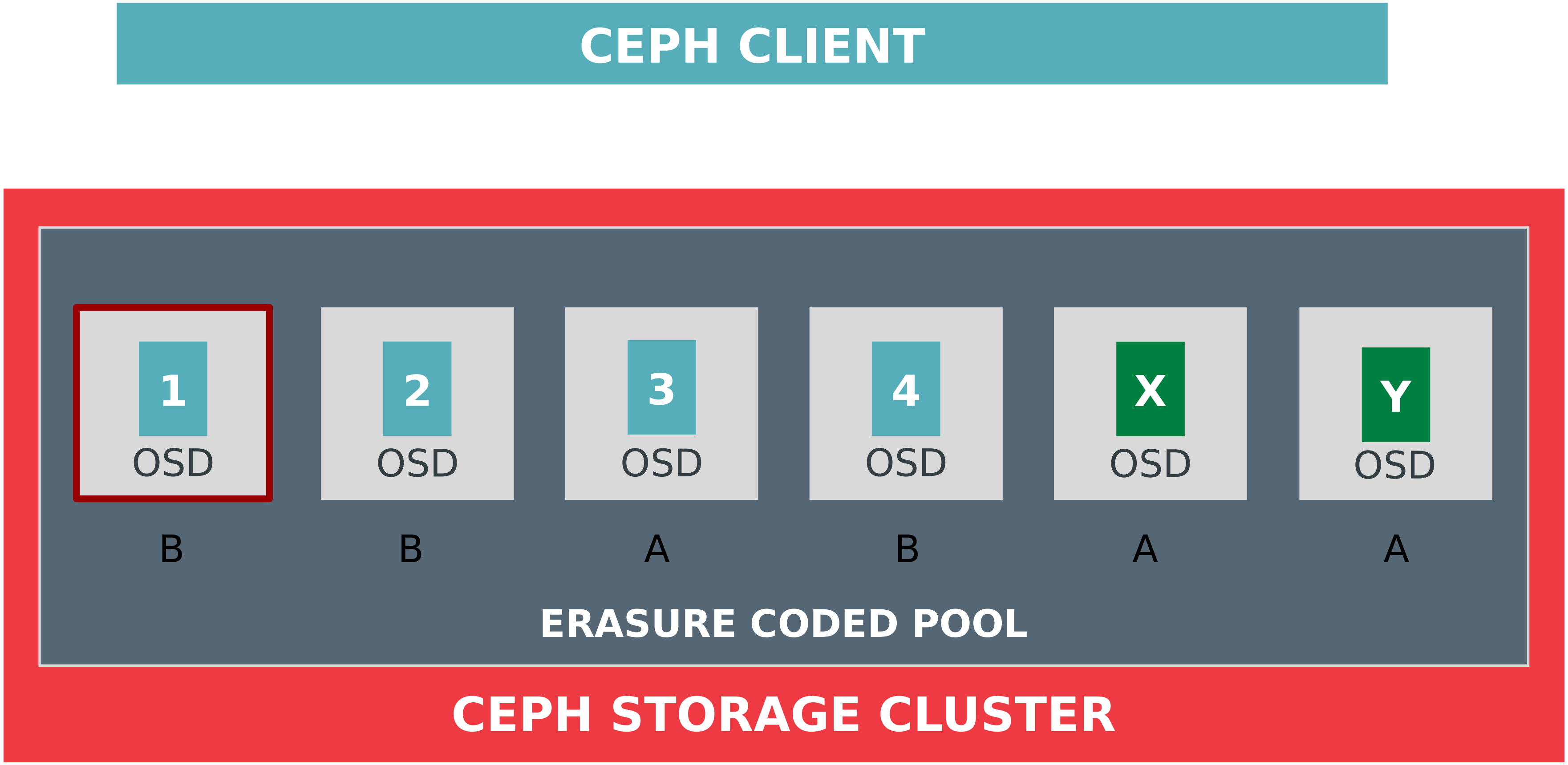
EC WRITE: PARTIAL FAILURE



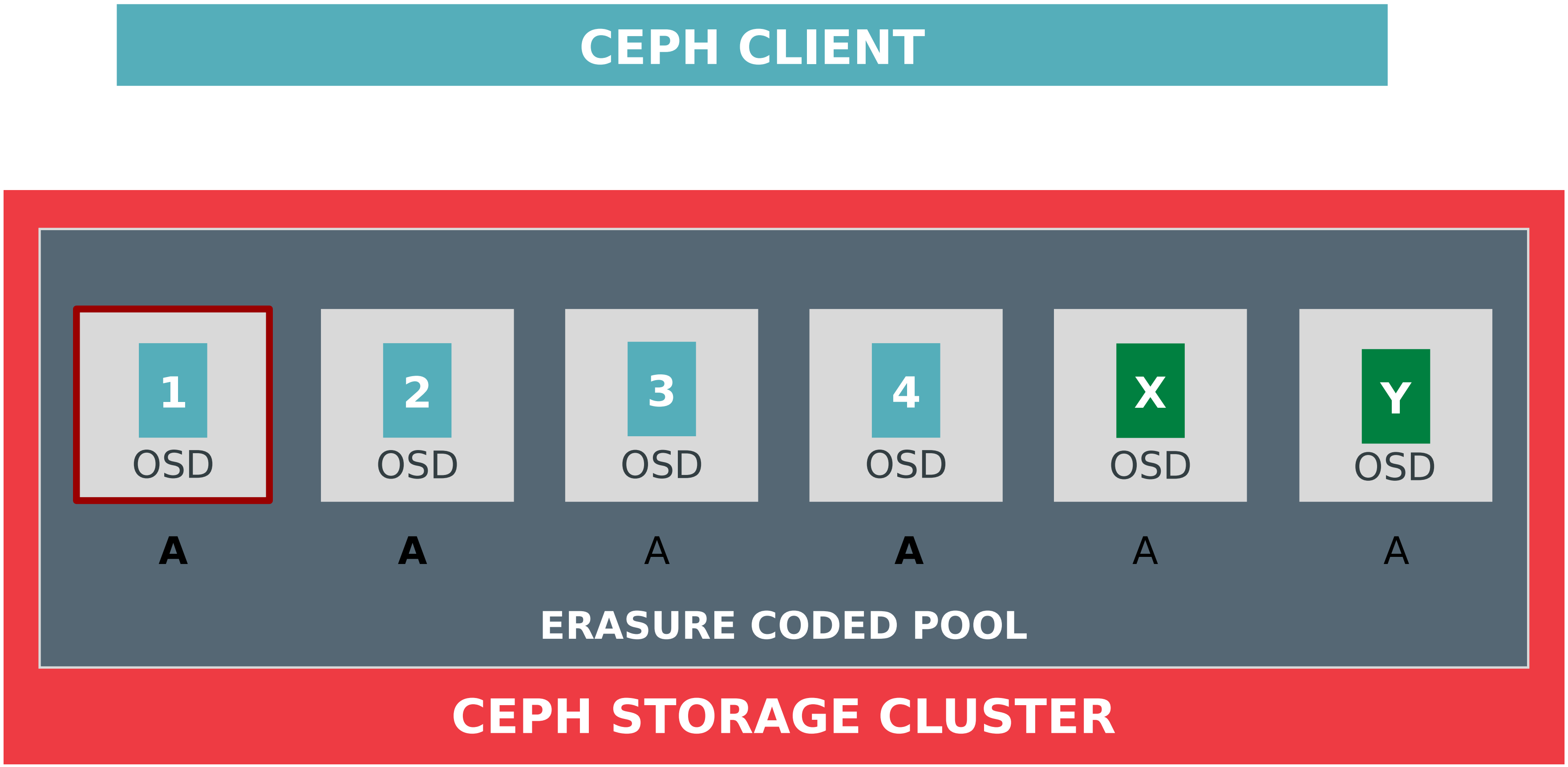
EC RESTRICTIONS

- 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, these operations are also inefficient for erasure codes
  - Generally require read/modify/write of affected stripe(s)
- Some applications 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

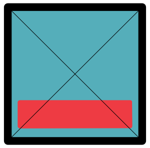
WHICH ERASURE CODE?

- The EC algorithm and implementation are pluggable
  - jerasure (free, open, and very fast)
  - ISA-L (Intel library; optimized for modern Intel procs)
  - LRC (local recovery code – layers over existing plugins)
- Parameterized
  - Pick k or m, stripe size
- OSD handles data path, placement, rollback, etc.
- Plugin handles
  - Encode and decode
  - 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

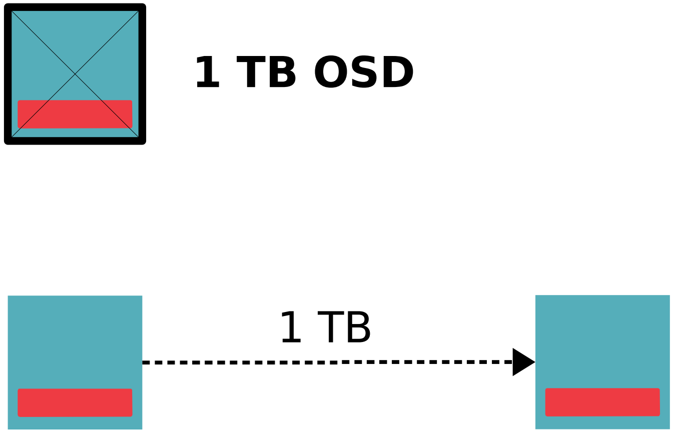
1 TB OSD

COST OF RECOVERY

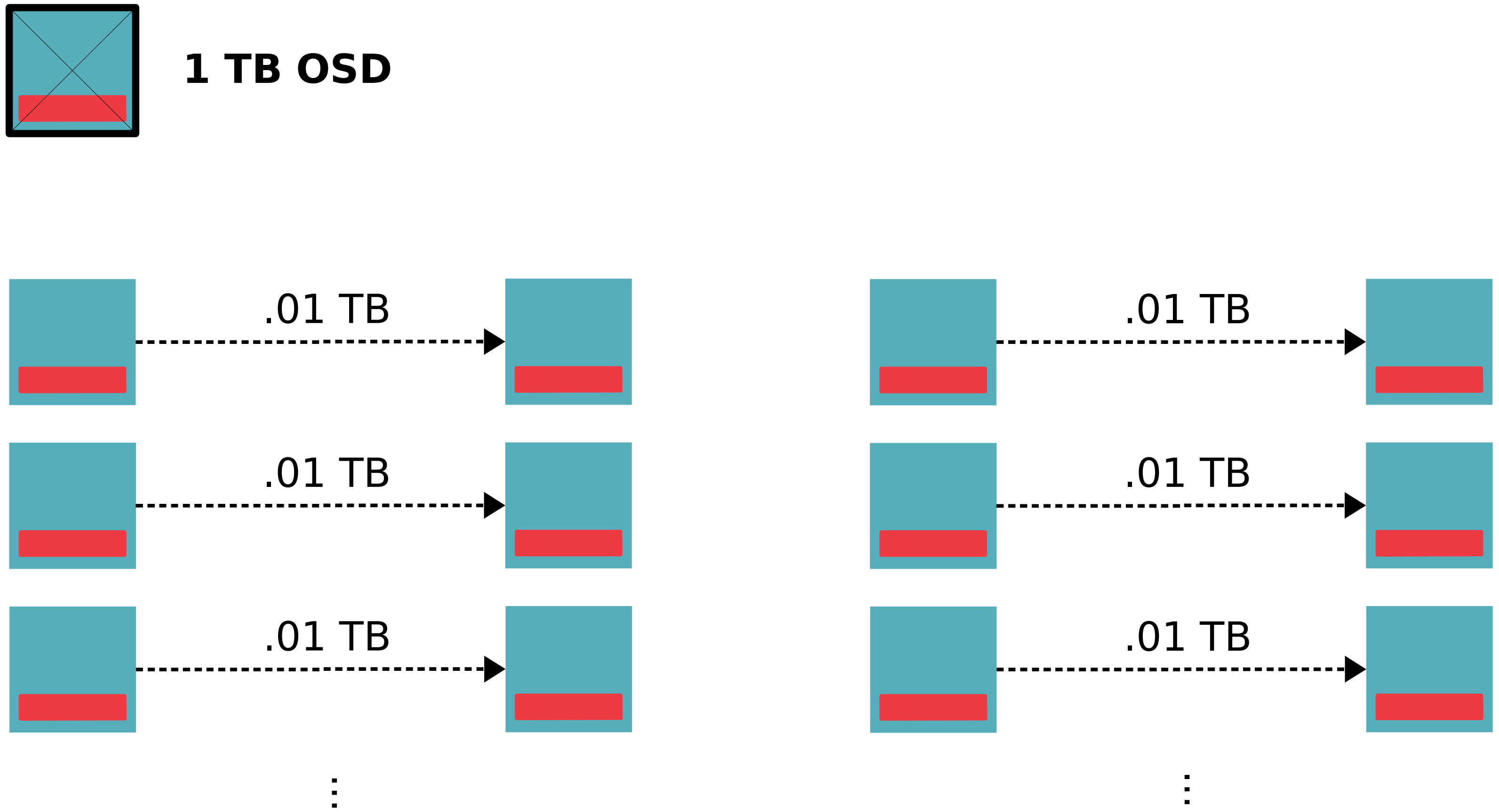


1 TB OSD

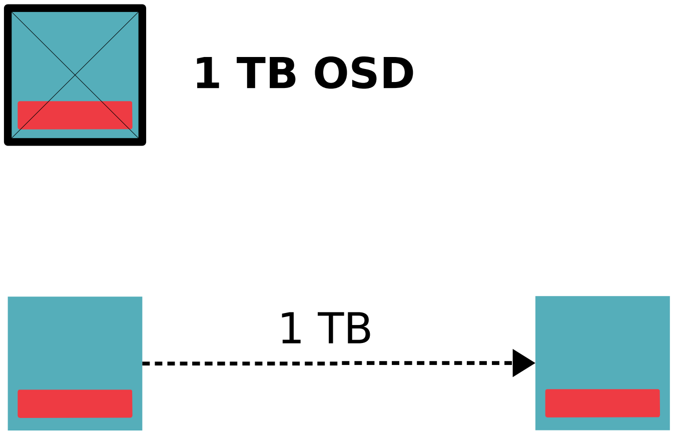
COST OF RECOVERY (REPLICATION)



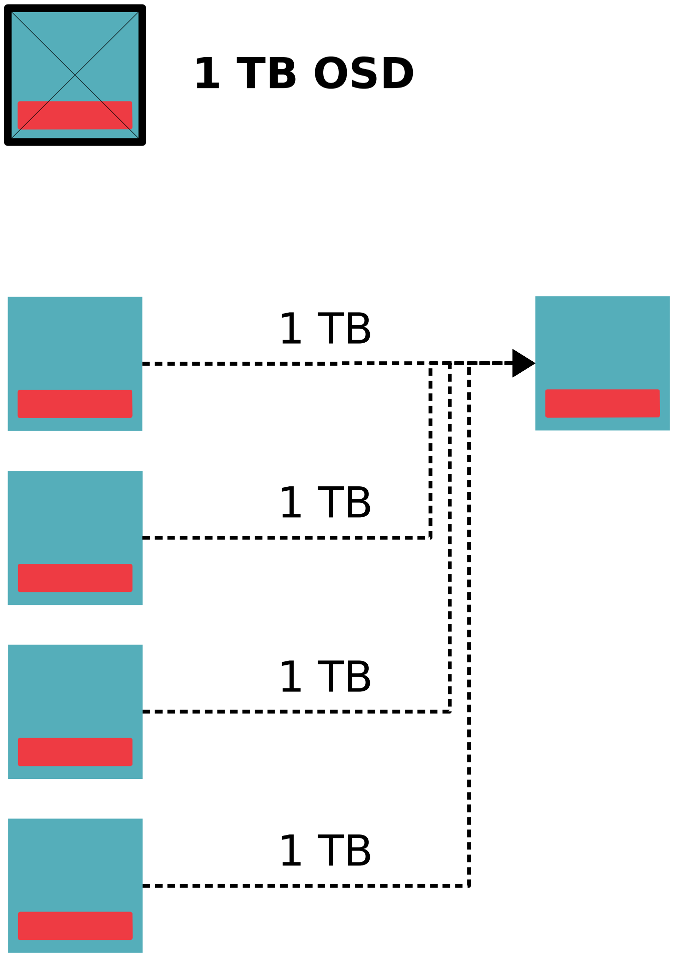
COST OF RECOVERY (REPLICATION)



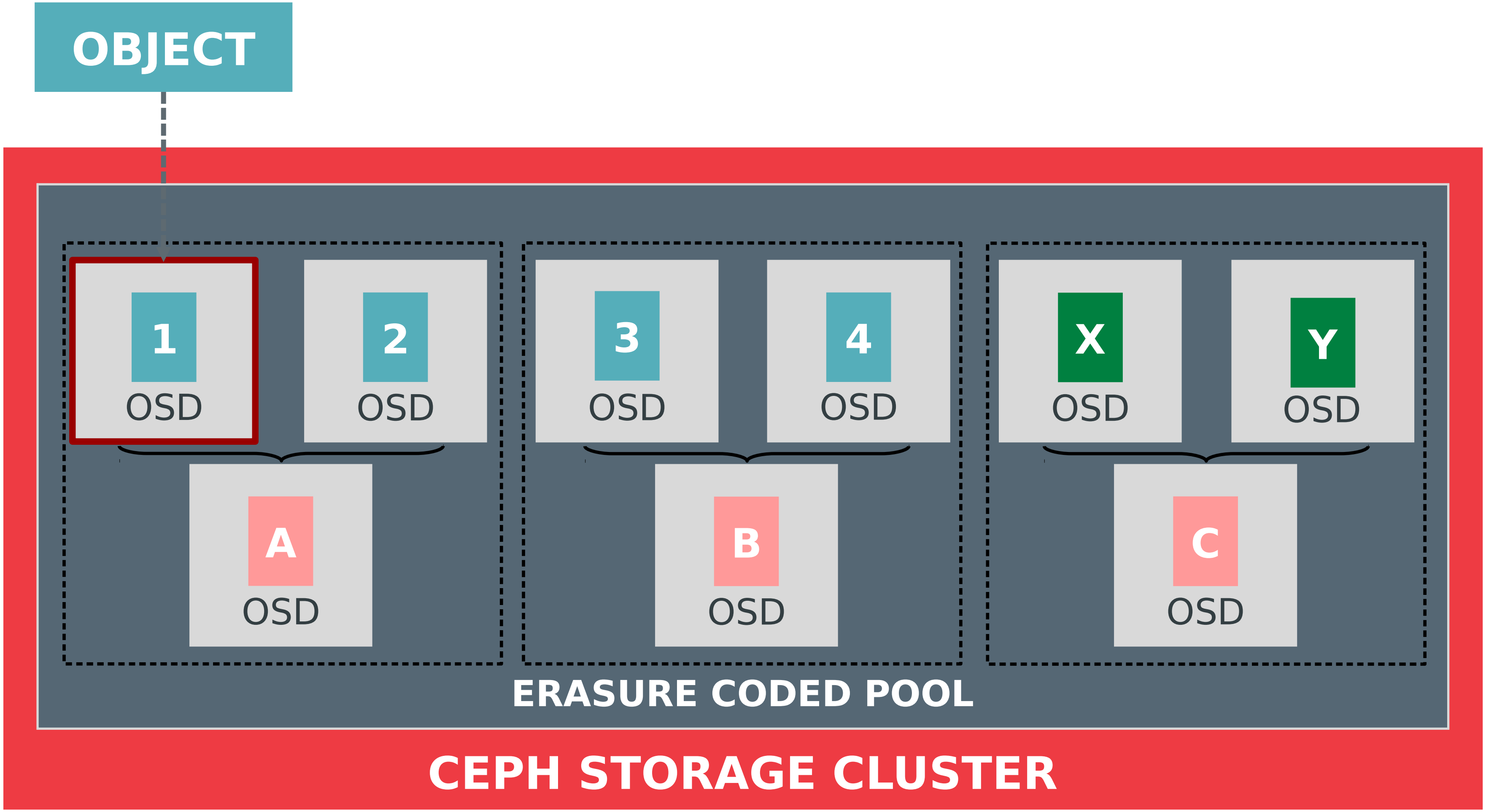
COST OF RECOVERY (REPLICATION)



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
  - Jim Plank (University of Tennessee)
  - Kevin Greenan (Box)

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 cold tier below base pool
  - Dynamic spin-down

OTHER ONGOING WORK

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

THANK YOU!

Sage Weil

CEPH PRINCIPAL ARCHITECT

sage@redhat.com

@liewegas