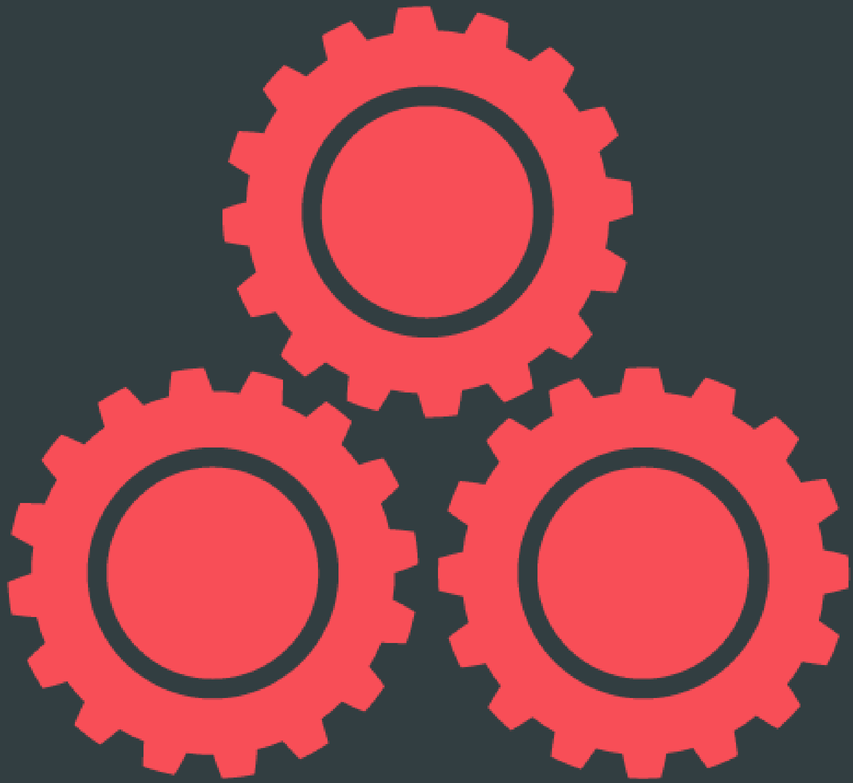
RECOVERY, ERASURE CODING, AND CACHE TIERING

SAMUEL JUST – 2015 CEPH DAY SHANGHAI



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
- Strong consistency (CP system)
- Infrastructure aware, dynamic topology
- Hash-based placement (CRUSH)
- Direct client to server data path

LIBRADOS Interface

- Rich object API
  - Bytes, attributes, key/value data
  - Partial overwrite of existing data
  - Single-object compound atomic operations
  - RADOS classes (stored procedures)

RADOS COMPONENTS

OSDs:

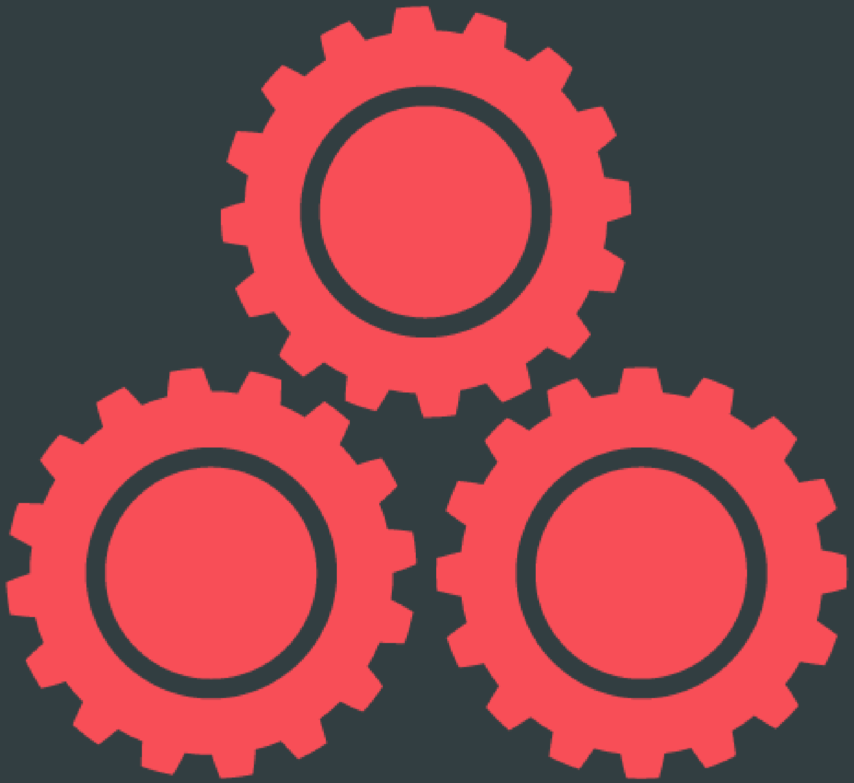
- 10s to 1000s in a cluster
- One per disk (or one per SSD, RAID group…)
- Serve stored objects to clients
- Intelligently peer for replication & recovery

RADOS COMPONENTS

M

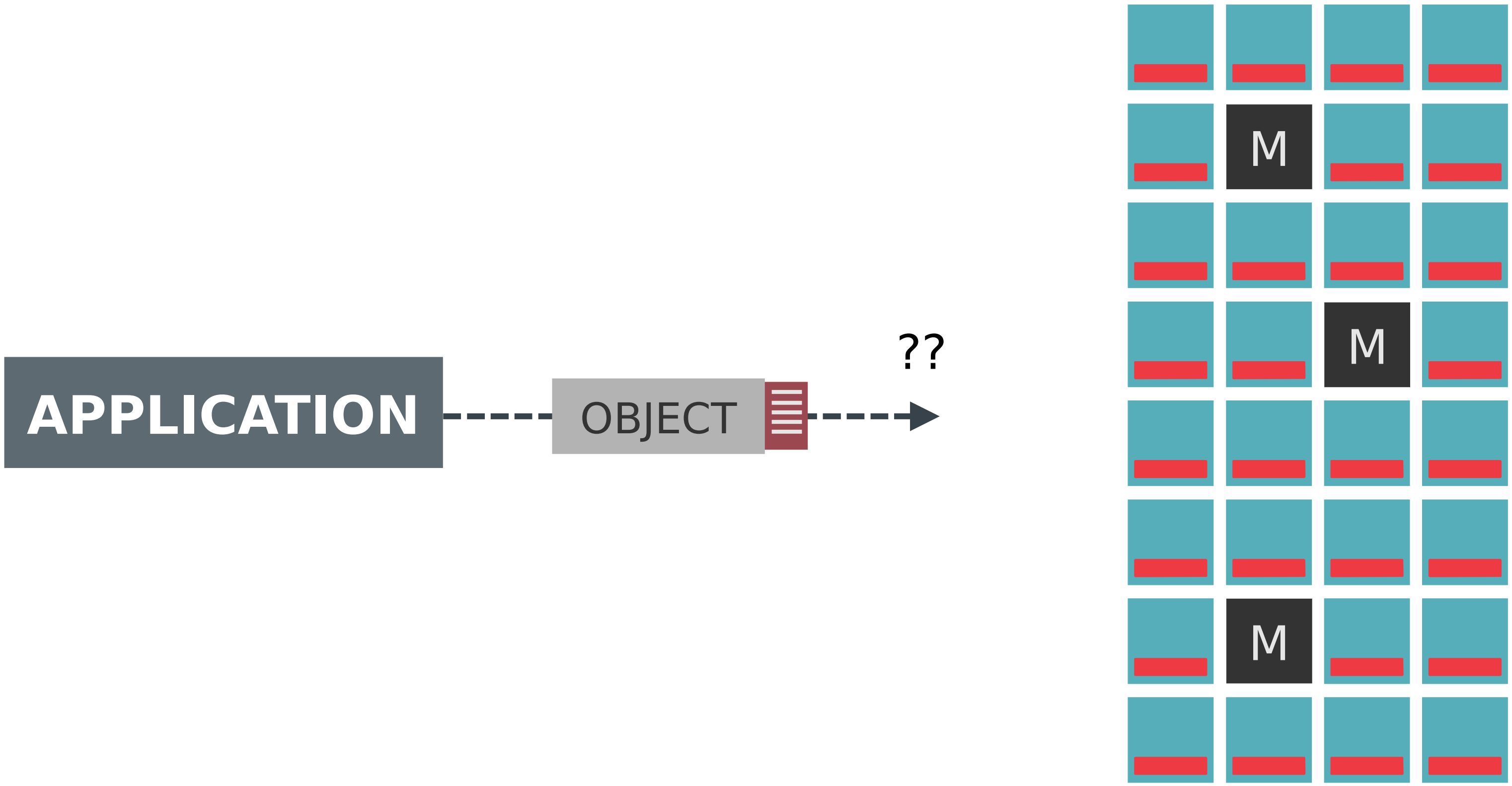
Monitors:

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

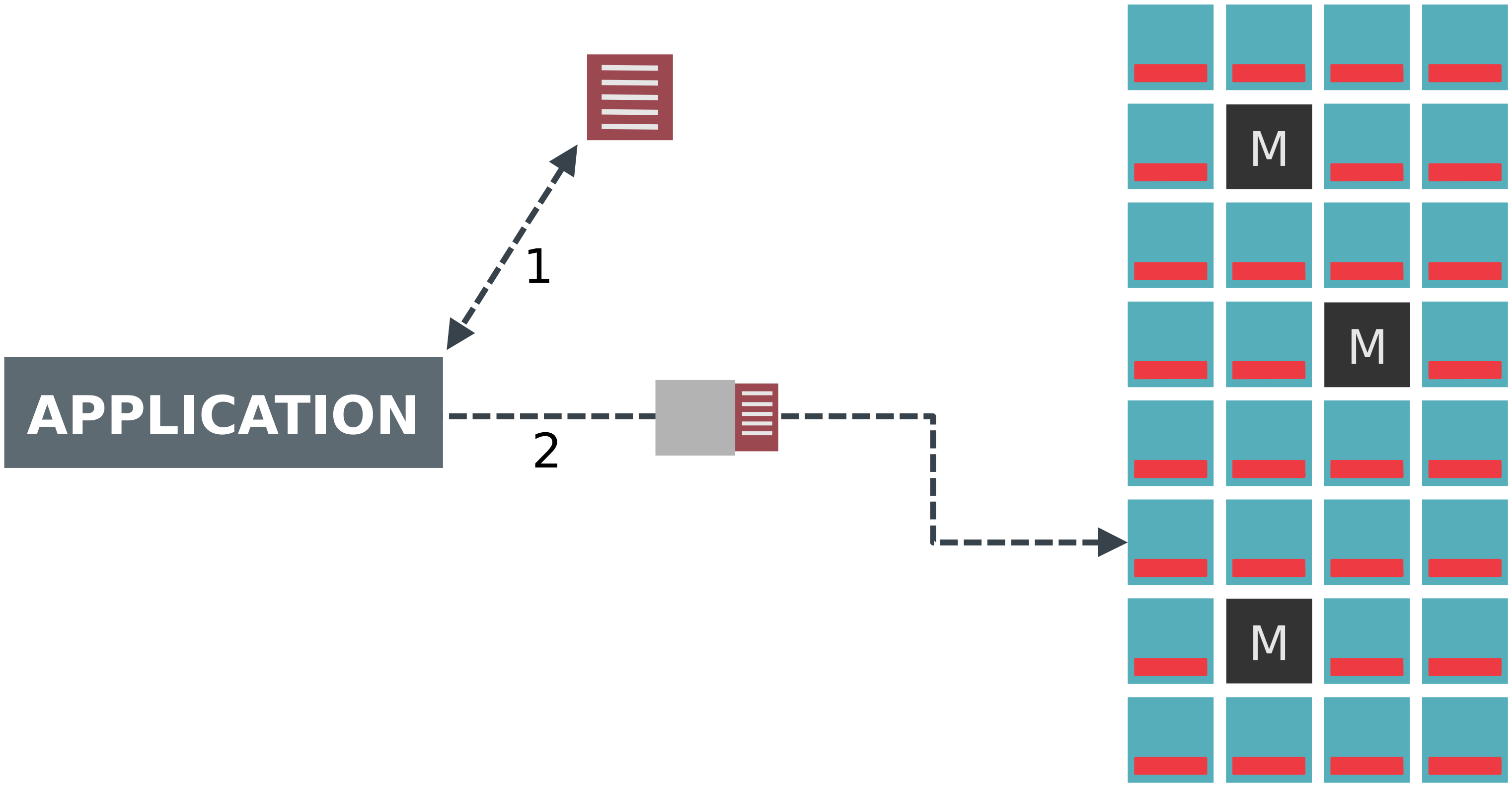


DATA PLACEMENT

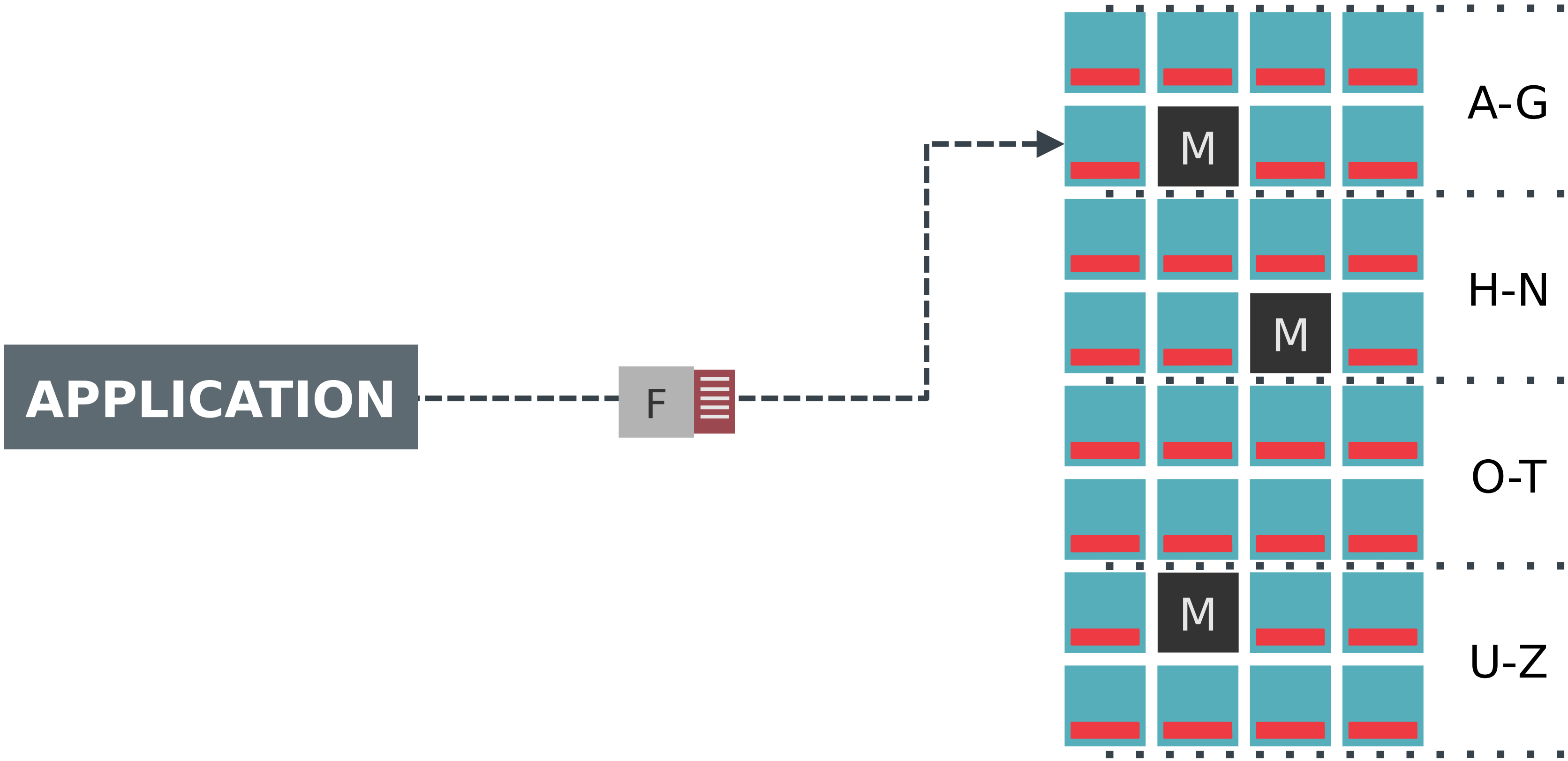
WHERE DO OBJECTS LIVE?



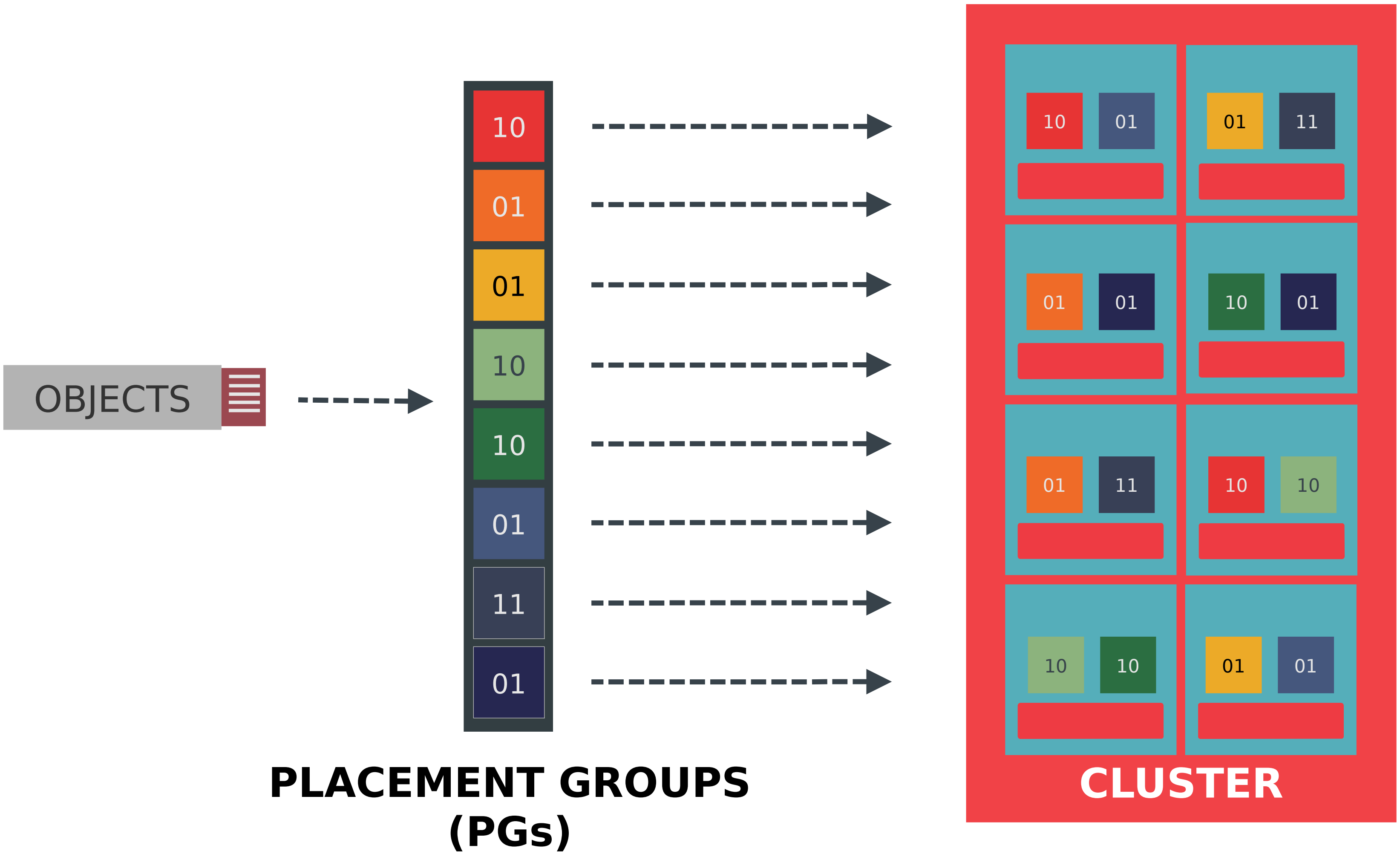
A METADATA SERVER?



CALCULATED PLACEMENT

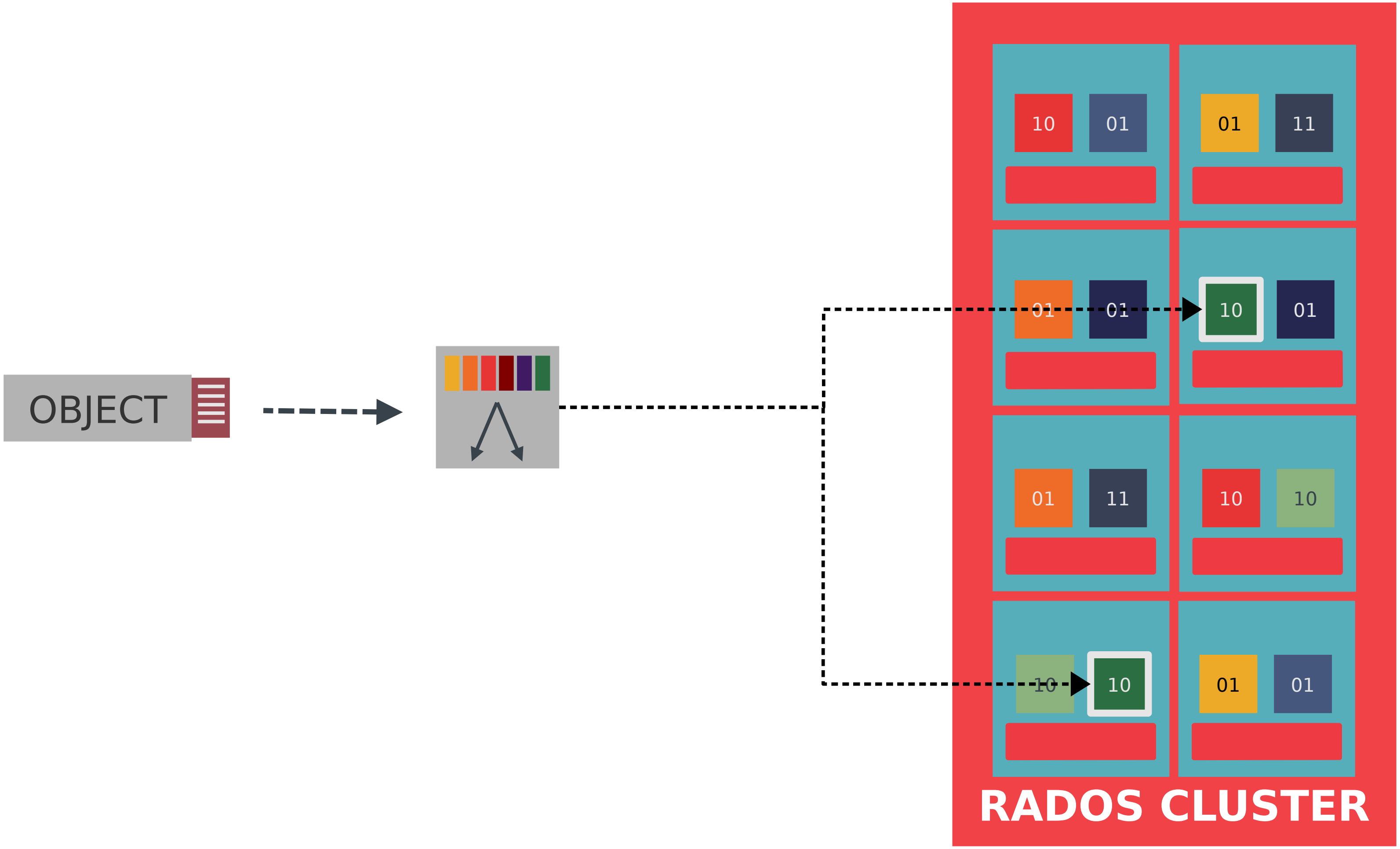


CRUSH

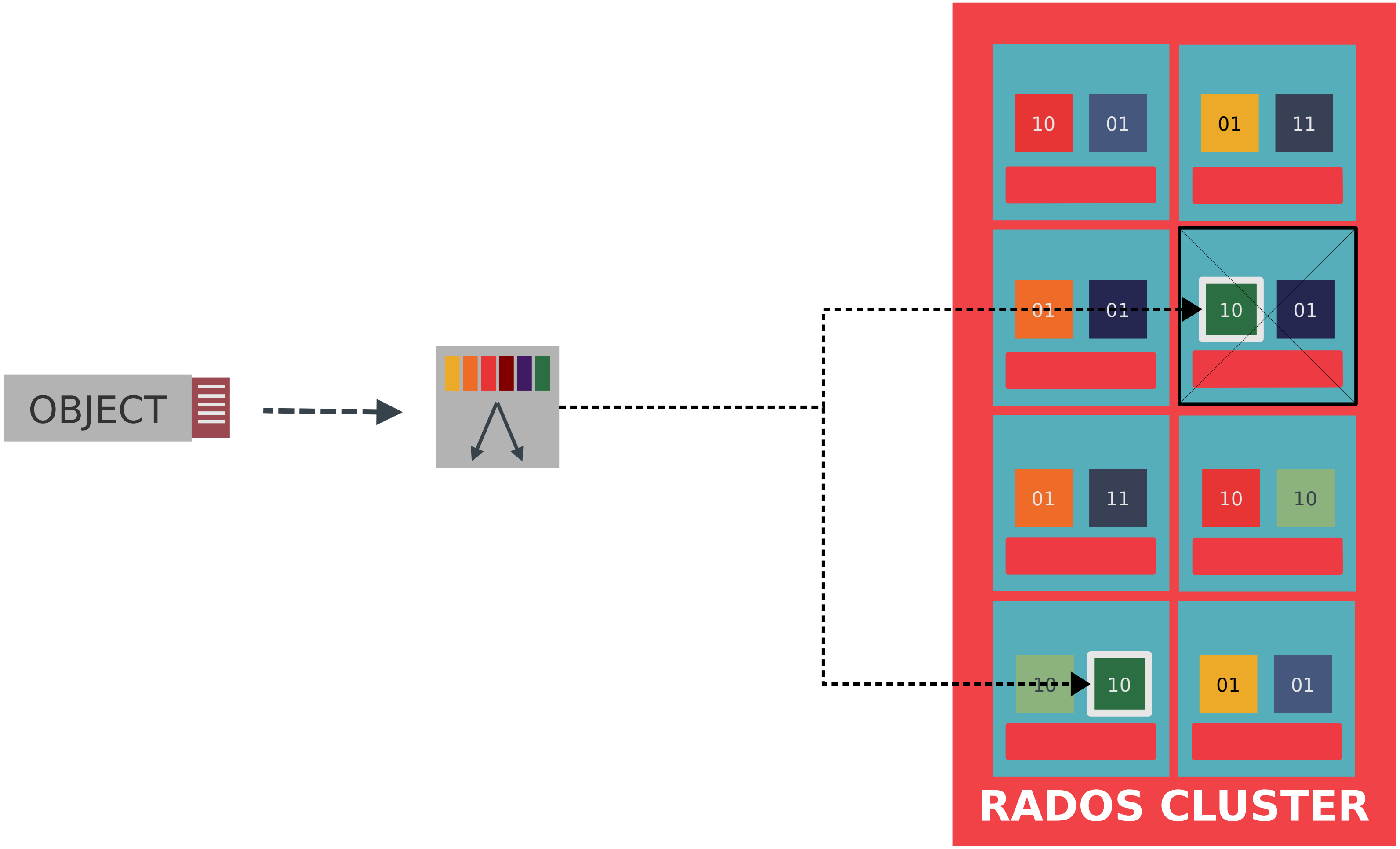


OBJECTS
10
01
01
10
10
01
11
01
PLACEMENT GROUPS
(PGs)
10
01
01
11
01
01
10
01
01
11
10
10
10
10
01
01
CLUSTER

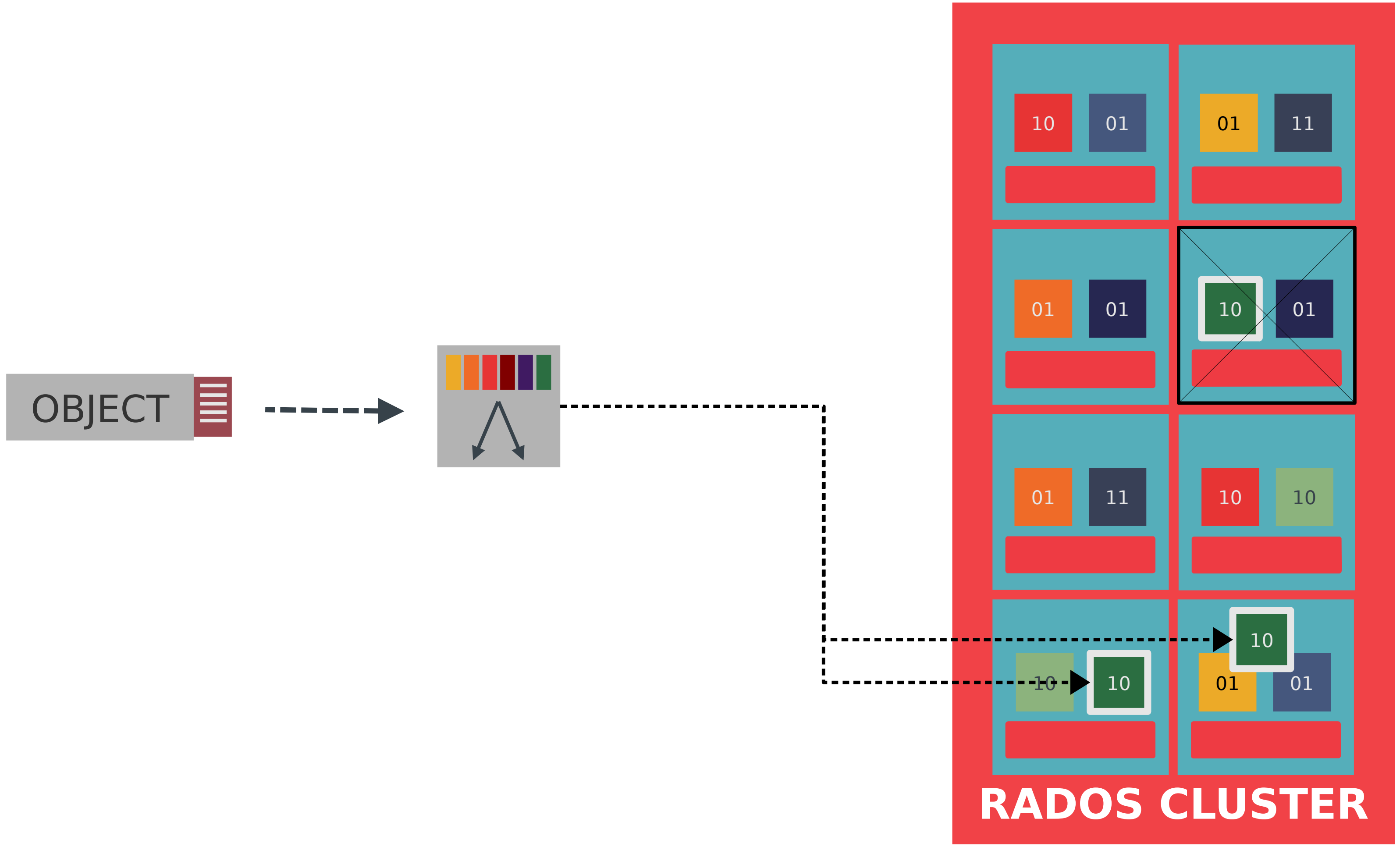
CRUSH IS A QUICK CALCULATION



CRUSH IS A QUICK CALCULATION

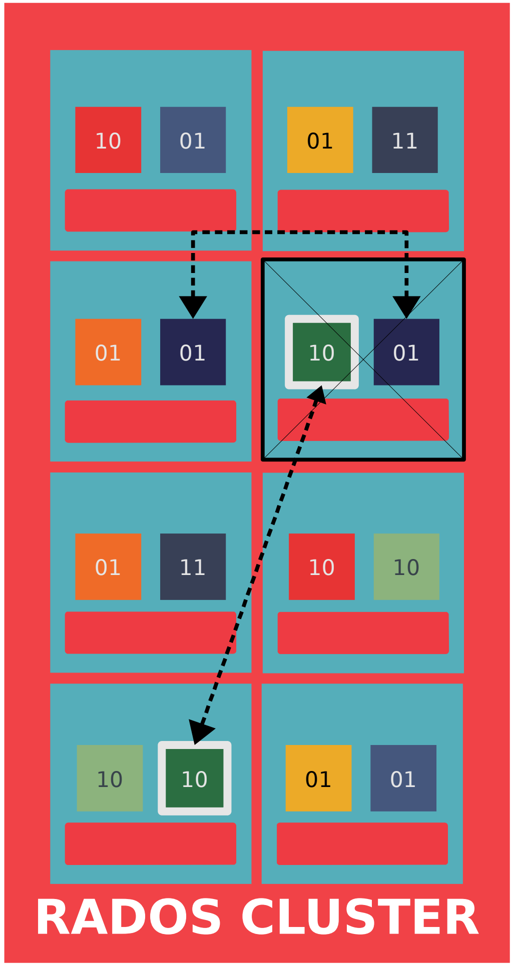


CRUSH AVOIDS FAILED DEVICES

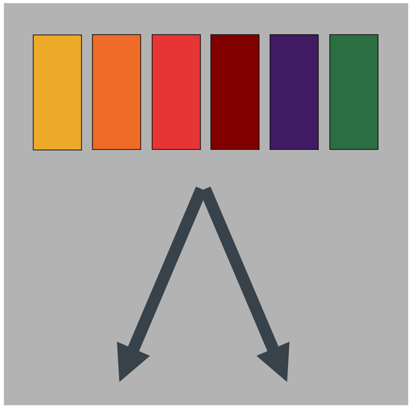


CRUSH: DECLUSTERED PLACEMENT

- 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



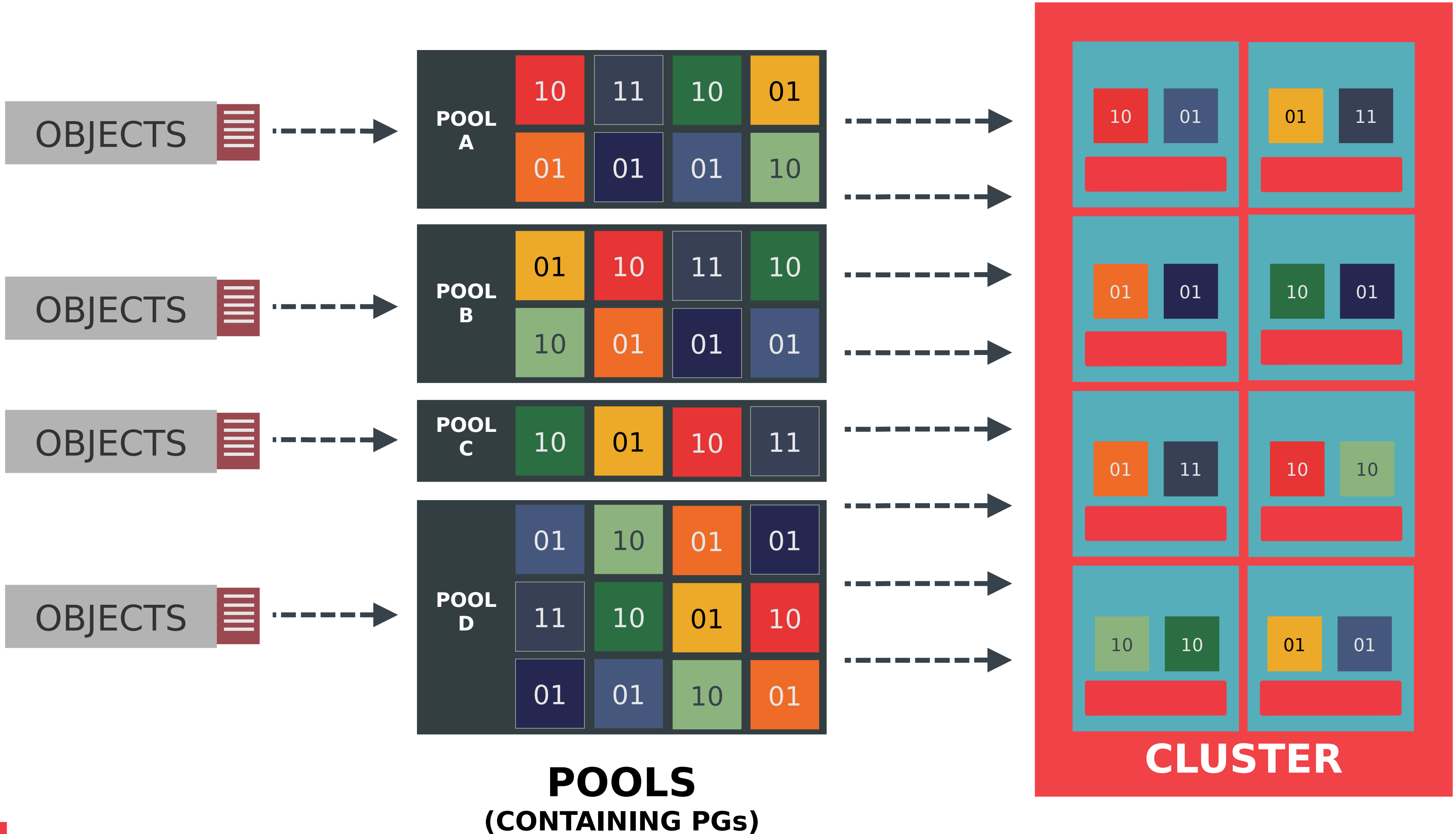
CRUSH: DYNAMIC DATA PLACEMENT



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



Peering and Recovery

Peering

- Each OSDMap is numbered with an epoch number
- The Monitors and OSDs store a history of OSDMaps
- Using this history, an OSD which becomes a new member of a PG can deduce every OSD which could have received a write which it needs to know about
- The process of discovering the authoritative state of the objects stored in the PG by contacting old PG members is called **Peering**

Peering

Epoch 20220:

11

Peering

Epoch 20220: 11 5 30

Peering

Epoch 20220:	11	5	30
Epoch 20113:	5	30	
Epoch 19884:	11	5	30

Recovery, Backfill, and PG Temp

- The state of a PG on an OSD is represented by a **PG Log** which contains the most recent operations witnessed by that OSD.
- The authoritative state of a PG after **Peering** is represented by constructing an authoritative **PG Log** from an up-to-date peer.
- If a peer's **PG Log** overlaps the authoritative **PG Log**, we can construct a set of out-of-date objects to recover
- Otherwise, we do not know which objects are out-of-date, and we must perform **Backfill**

Recovery, Backfill, and PG Temp

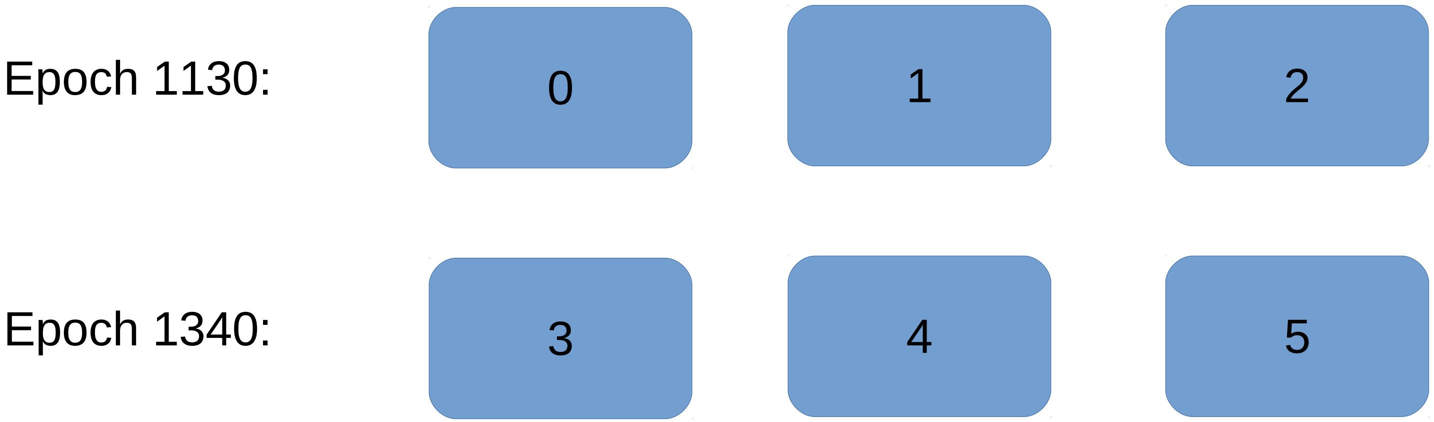
- If we do not know which objects are invalid, we certainly cannot serve reads from such a peer
- If after peering the primary determines that it or any other peer requires **Backfill**, it will request that the Monitor cluster publish a new map with an exception to the CRUSH mapping for this PG mapping it to the best set of up-to-date peers that it can find.
- Once that map is published, peering will happen again, and the up-to-date peers will independently conclude that they should serve reads and writes while concurrently backfilling the correct peers

Backfill Example

Epoch 1130:

0 1 2

Backfill Example



Backfill Example

Epoch 1130:	0	1	2
Epoch 1340:	3	4	5
Epoch 1345:	0	1	2

Backfill Example

Epoch 1130:	0	1	2
Epoch 1340:	3	4	5
Epoch 1345:	0	1	2
Epoch 1391:	3	4	5

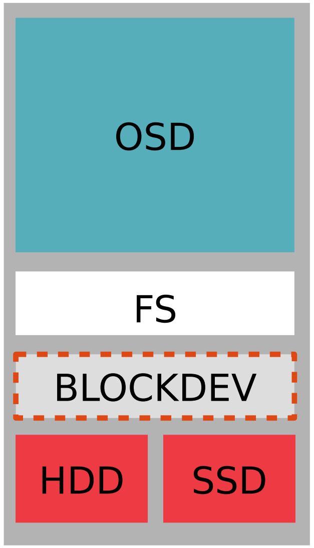
Backfill

- Backfill in object order (basically hash order) within the pg
- info.last_backfill
- Obj o <= info.last_backfill → object is up to date
- Obj o > info.last_backfill → object is not up to date

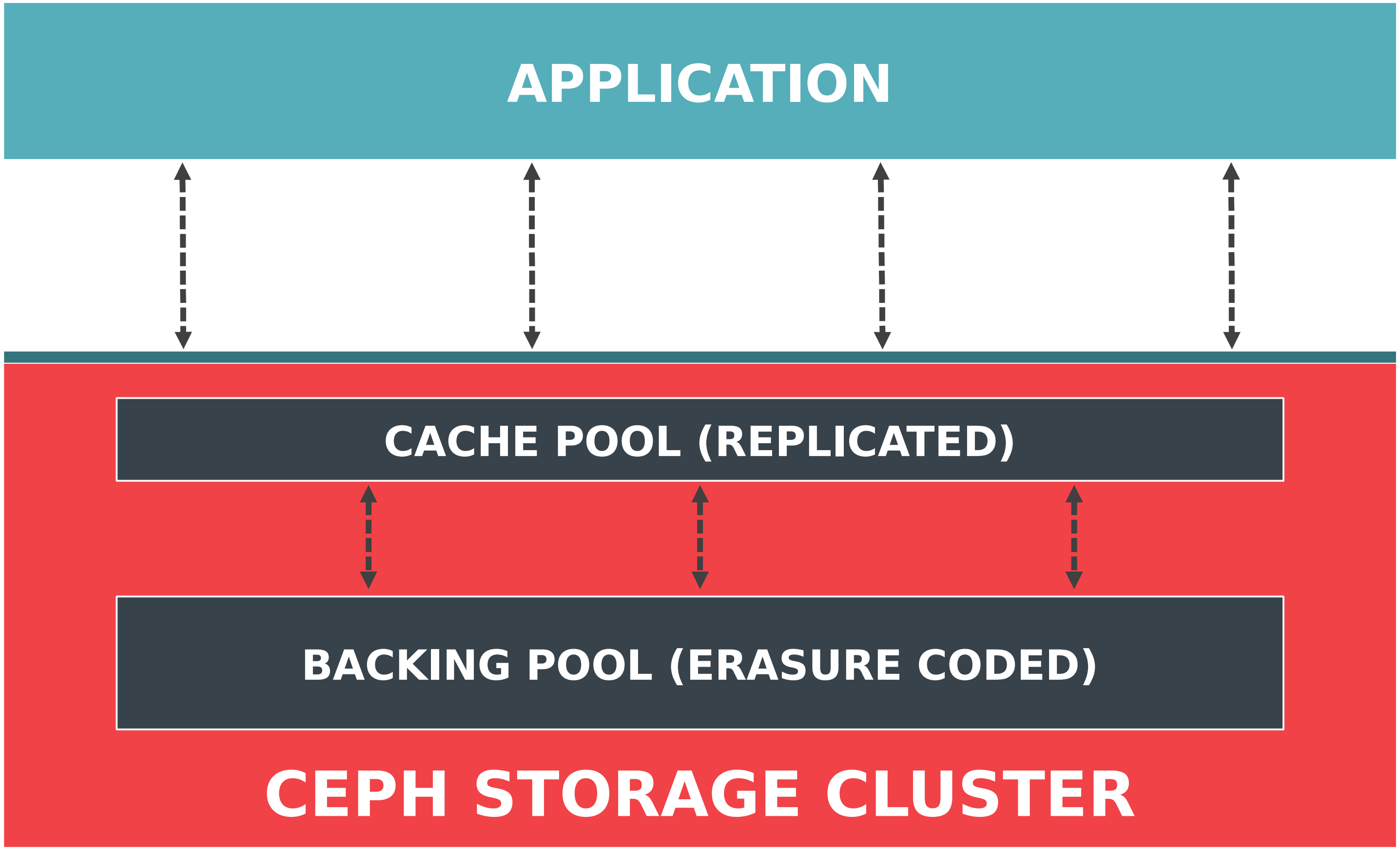
TIERED STORAGE

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



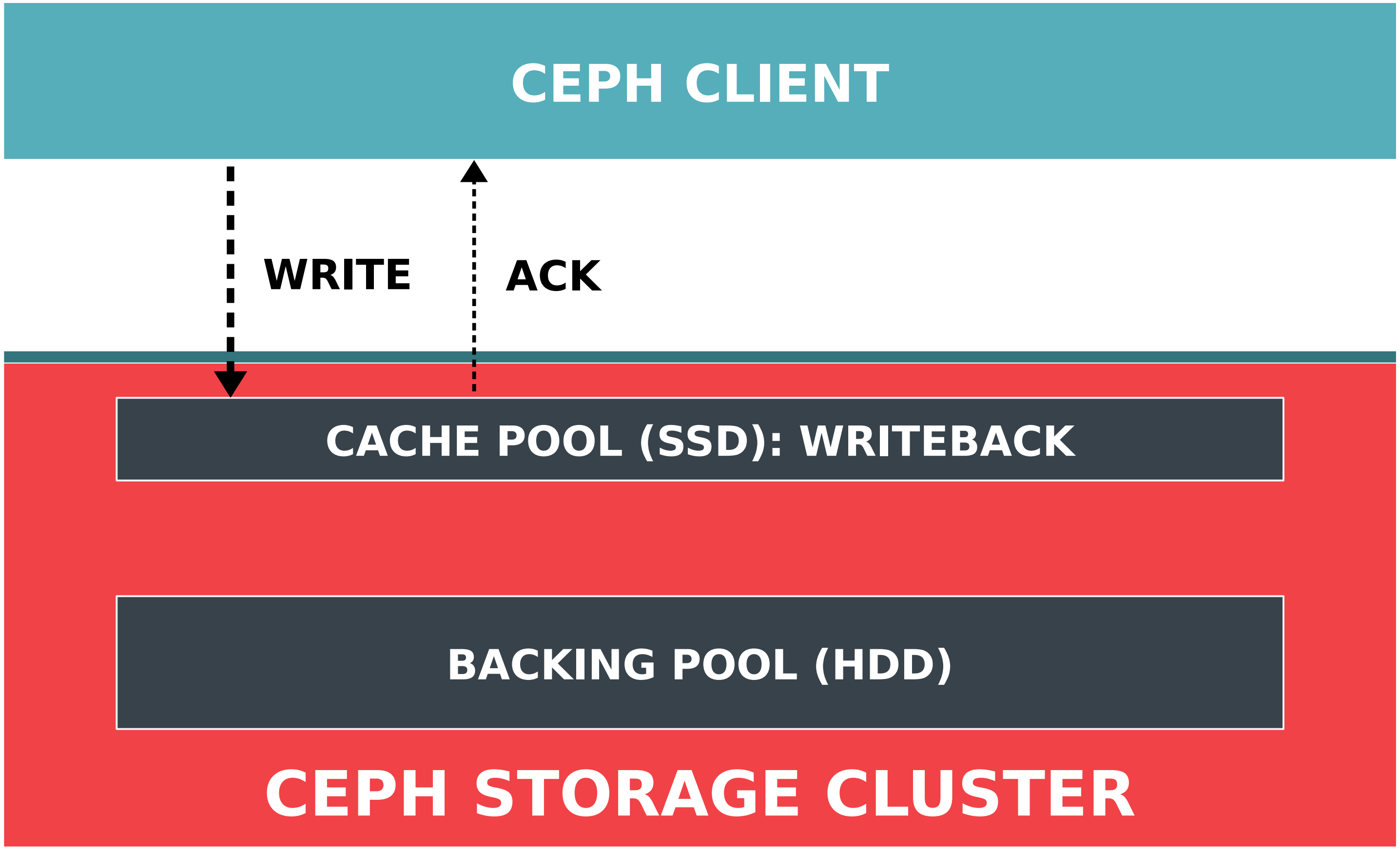
TIERED STORAGE



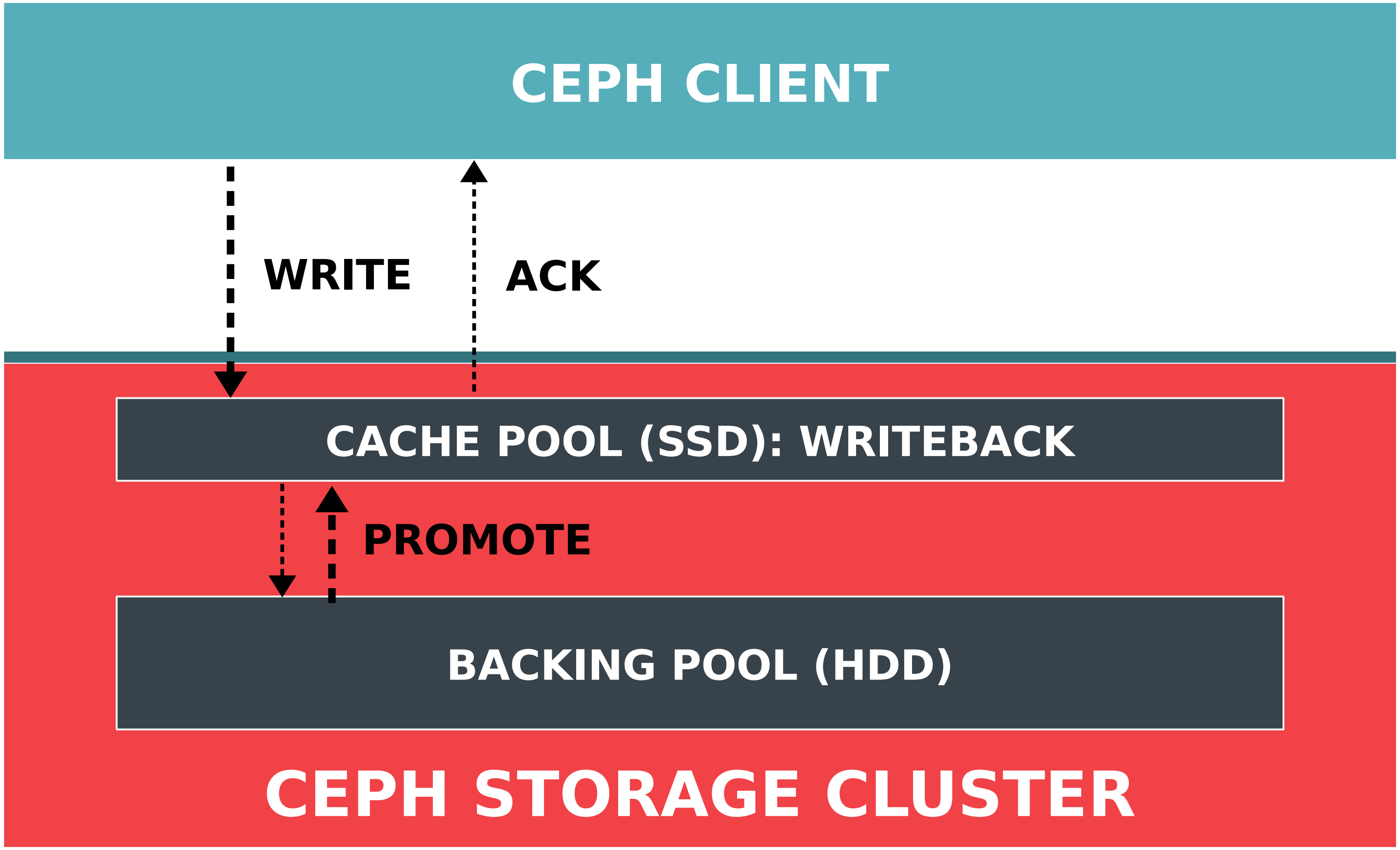
RADOS TIERING PRINCIPLES

- 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.

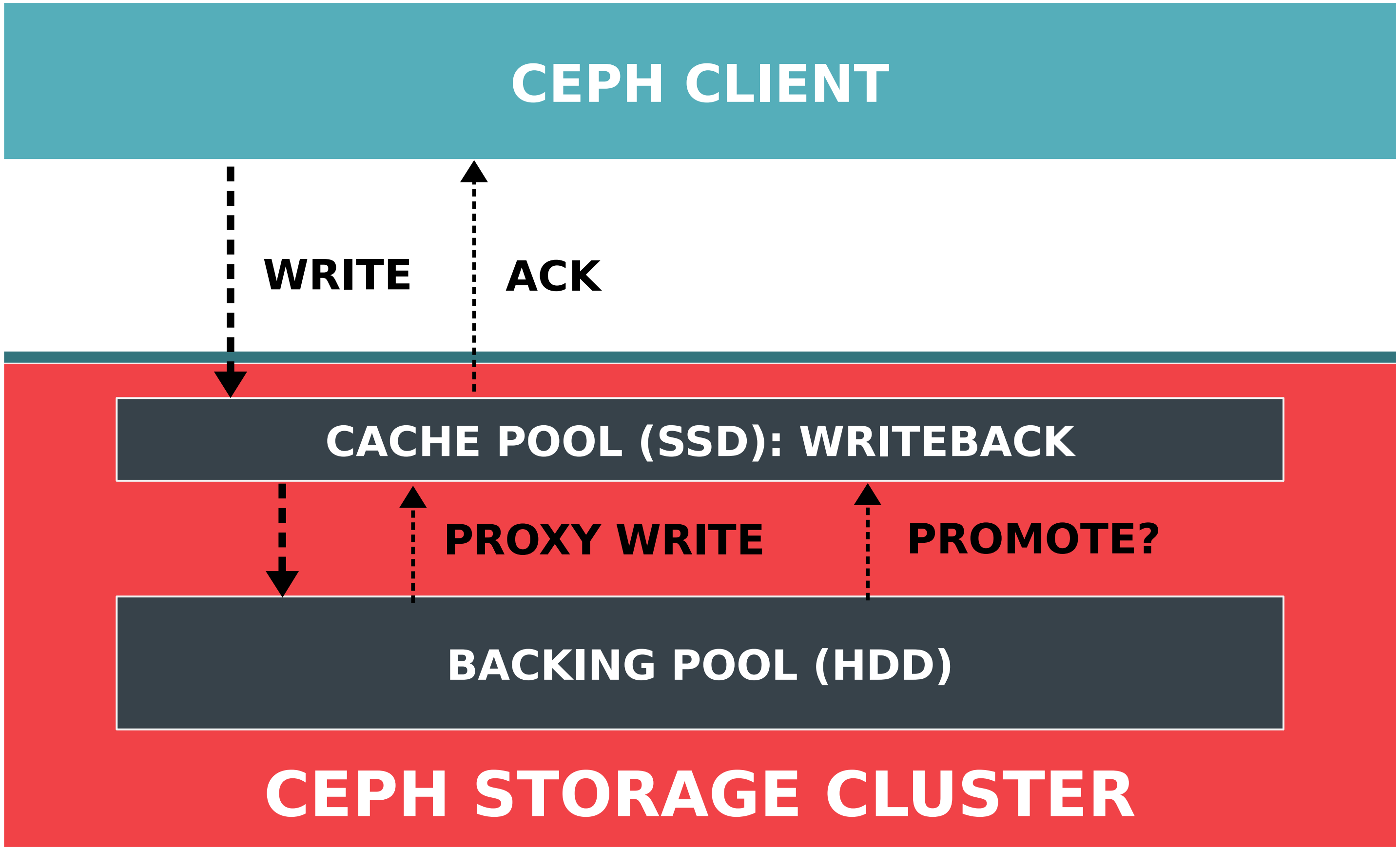
WRITE INTO CACHE POOL



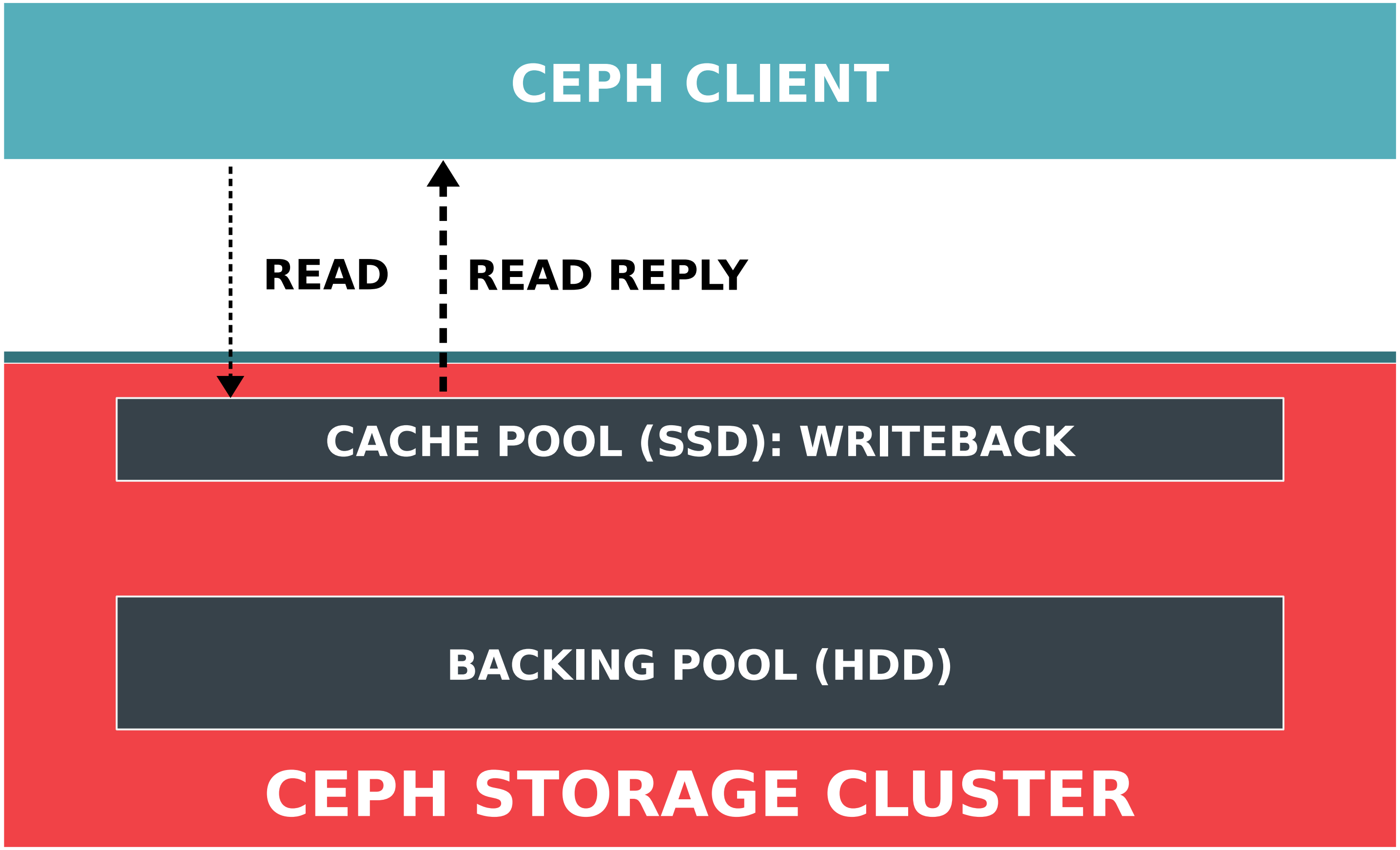
WRITE (MISS)


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

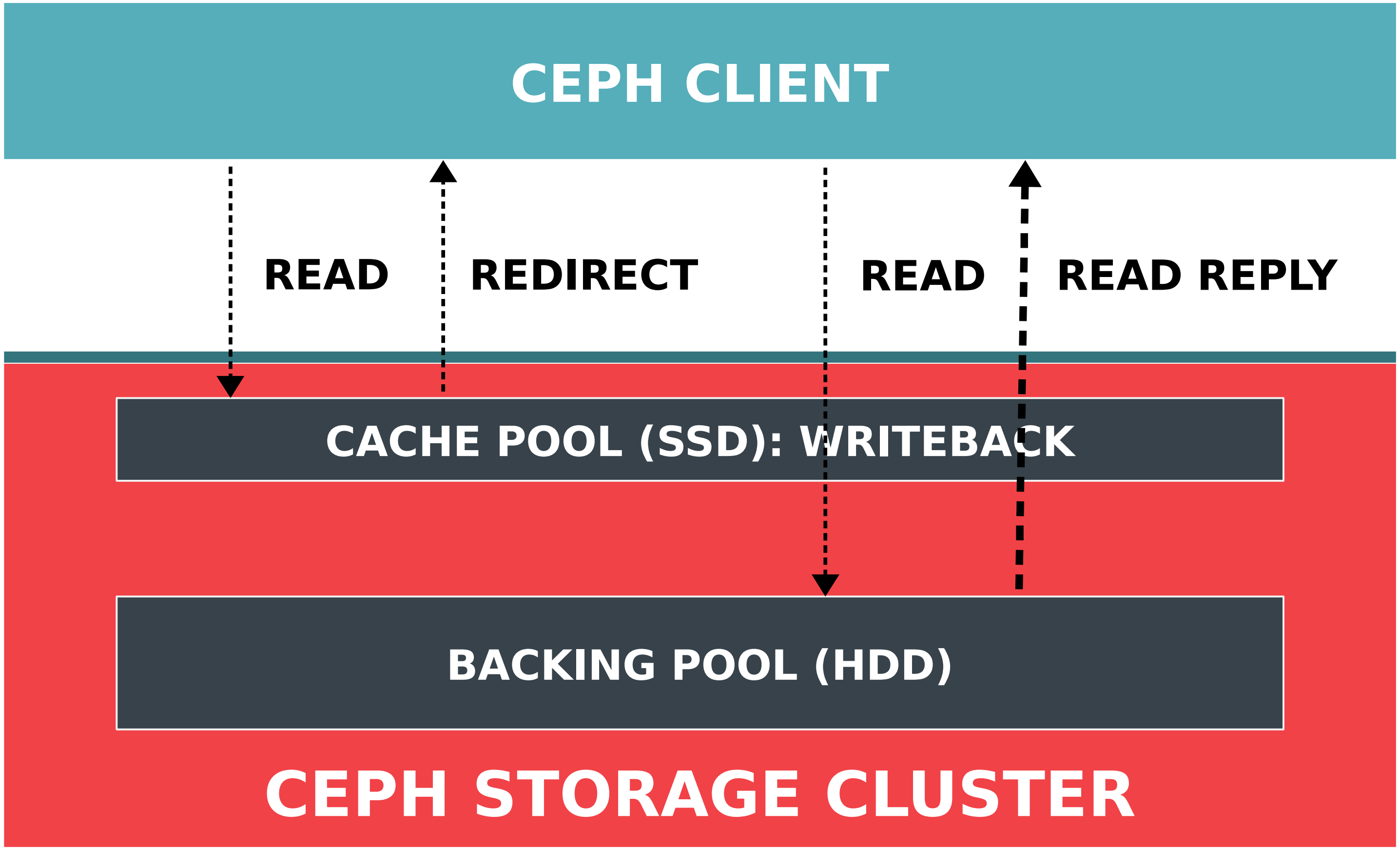
WRITE (MISS) PROXY!



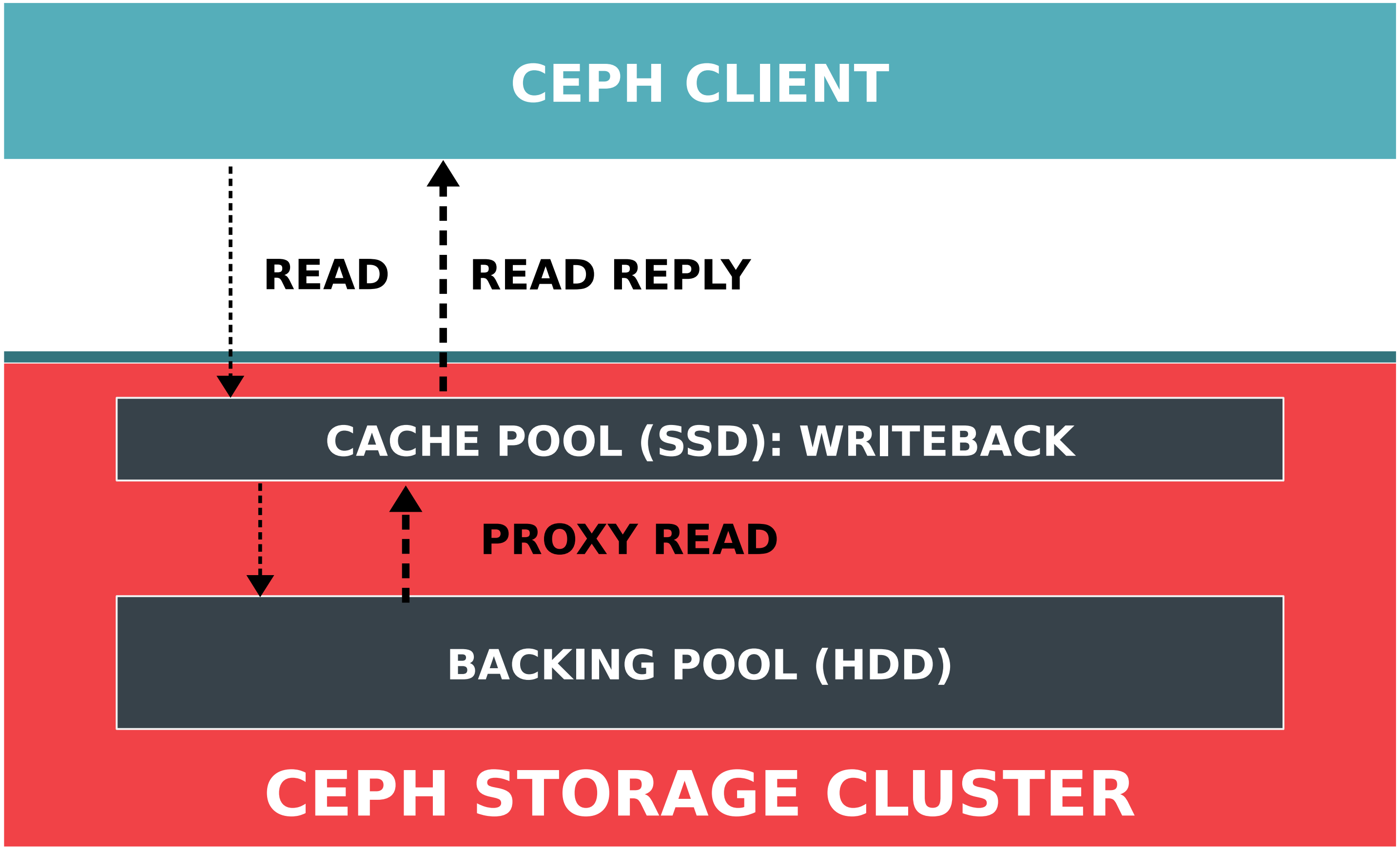
READ (CACHE HIT)



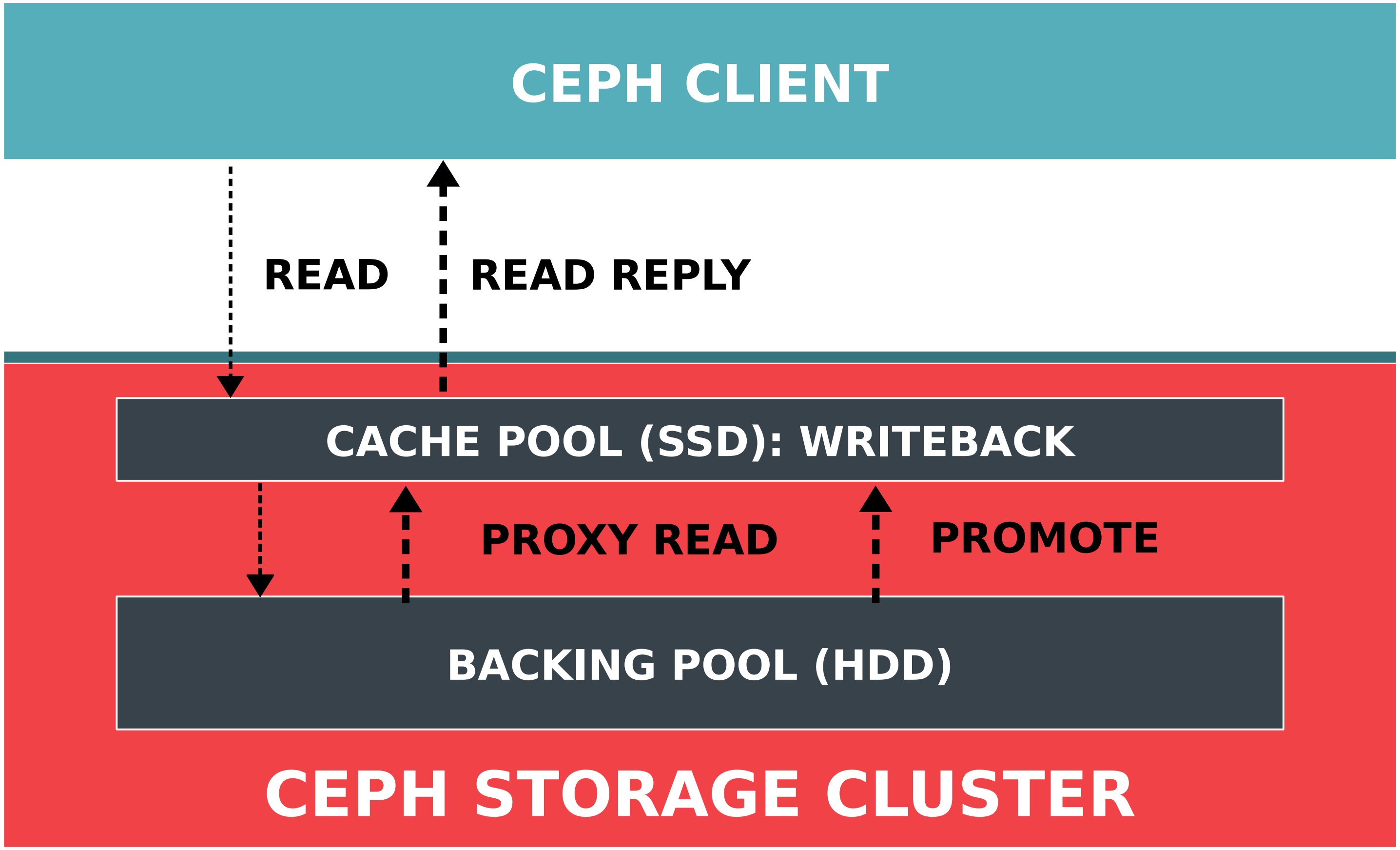
READ (CACHE MISS)



READ (CACHE MISS) PROXY!



READ (CACHE MISS)



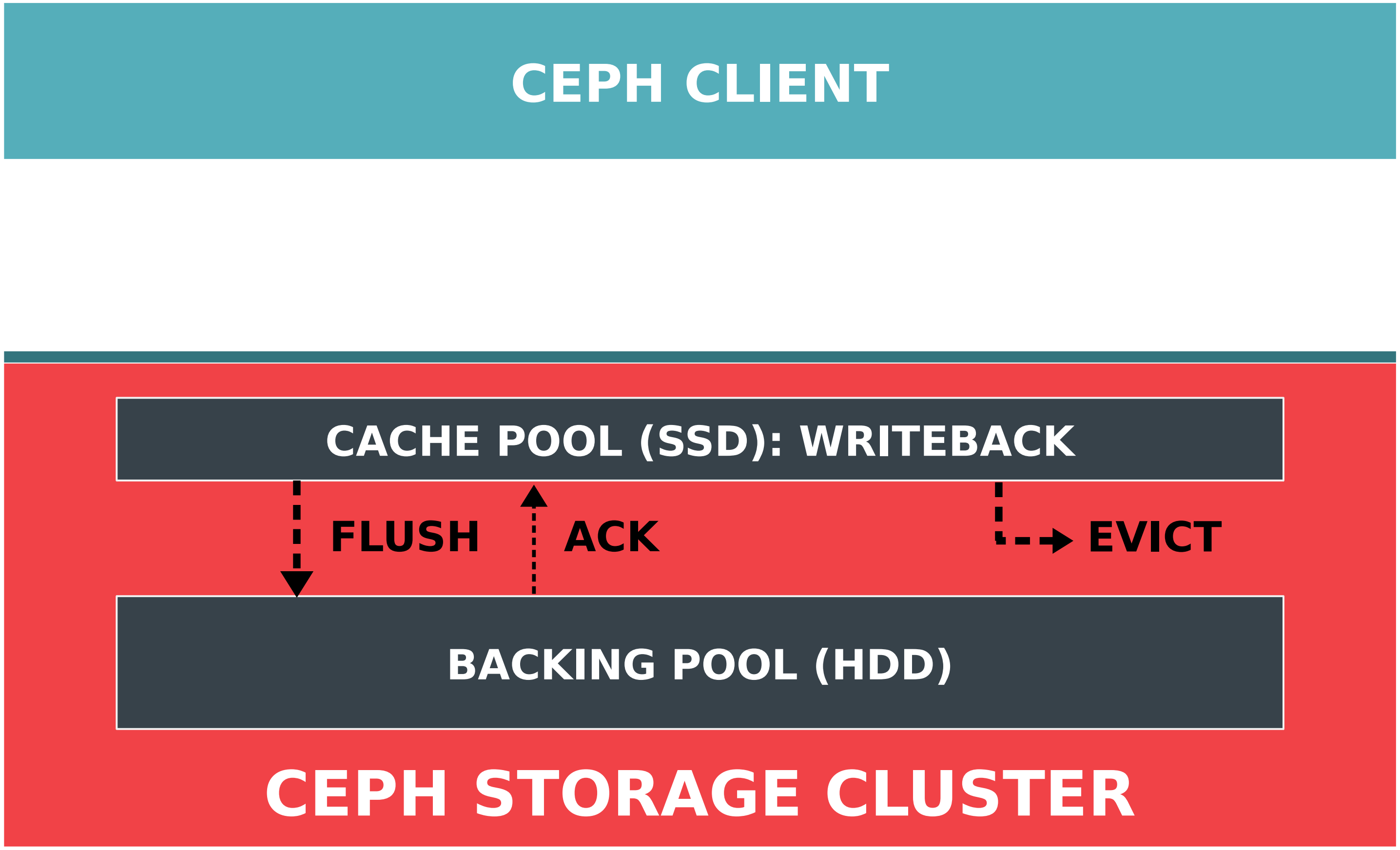
CACHE TIERING ARCHITECTURE

- Cache and backing pools are otherwise independent rados pools with independent placement rules
- OSDs in the cache pool are able to handle promotion and eviction of their objects independently allowing for scalability.
- RADOS clients understand the tiering configuration and are able to send requests to the right place mostly without redirects.
- Librados users can perform operations on a cache pool transparently trusting the library to correctly handle routing requests between the cache pool and base pool as needed

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

FLUSH AND/OR EVICT COLD DATA



TIERING AGENT

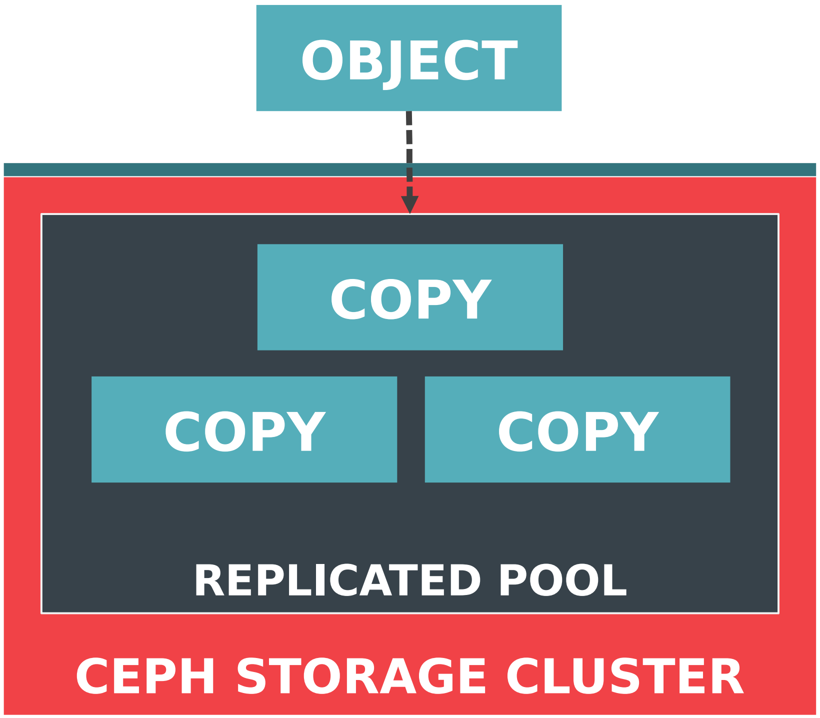
- Each PG has an internal tiering **agent**
  - Manages PG based on administrator defined policy
- Flush **dirty** objects
  - When pool reaches target **dirty ratio**
  - Tries to select cold objects
  - Marks objects **clean** when they have been written back to the base pool
- Evict clean objects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

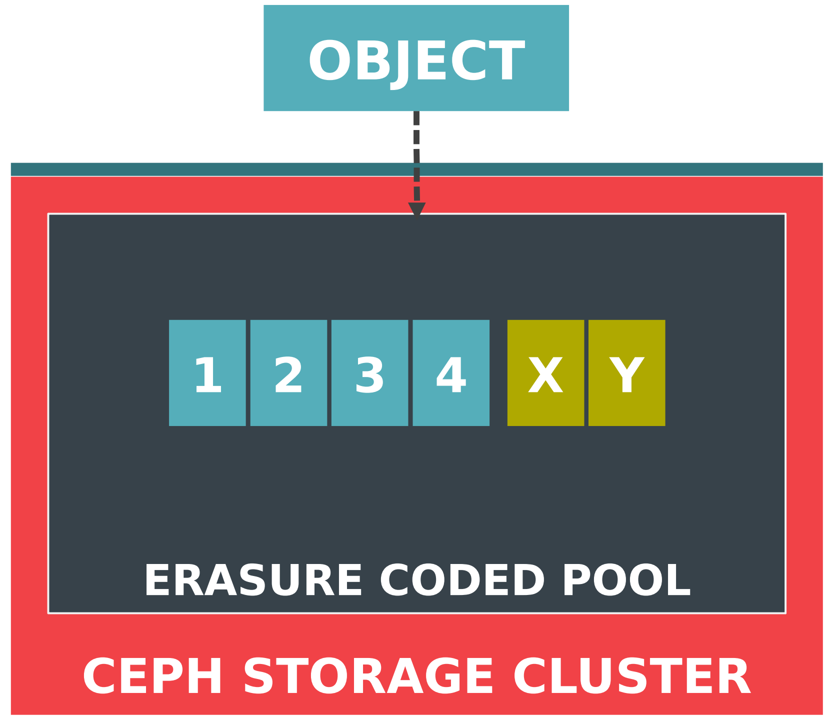
ERASURE CODING

ERASURE CODING



Full copies of stored objects

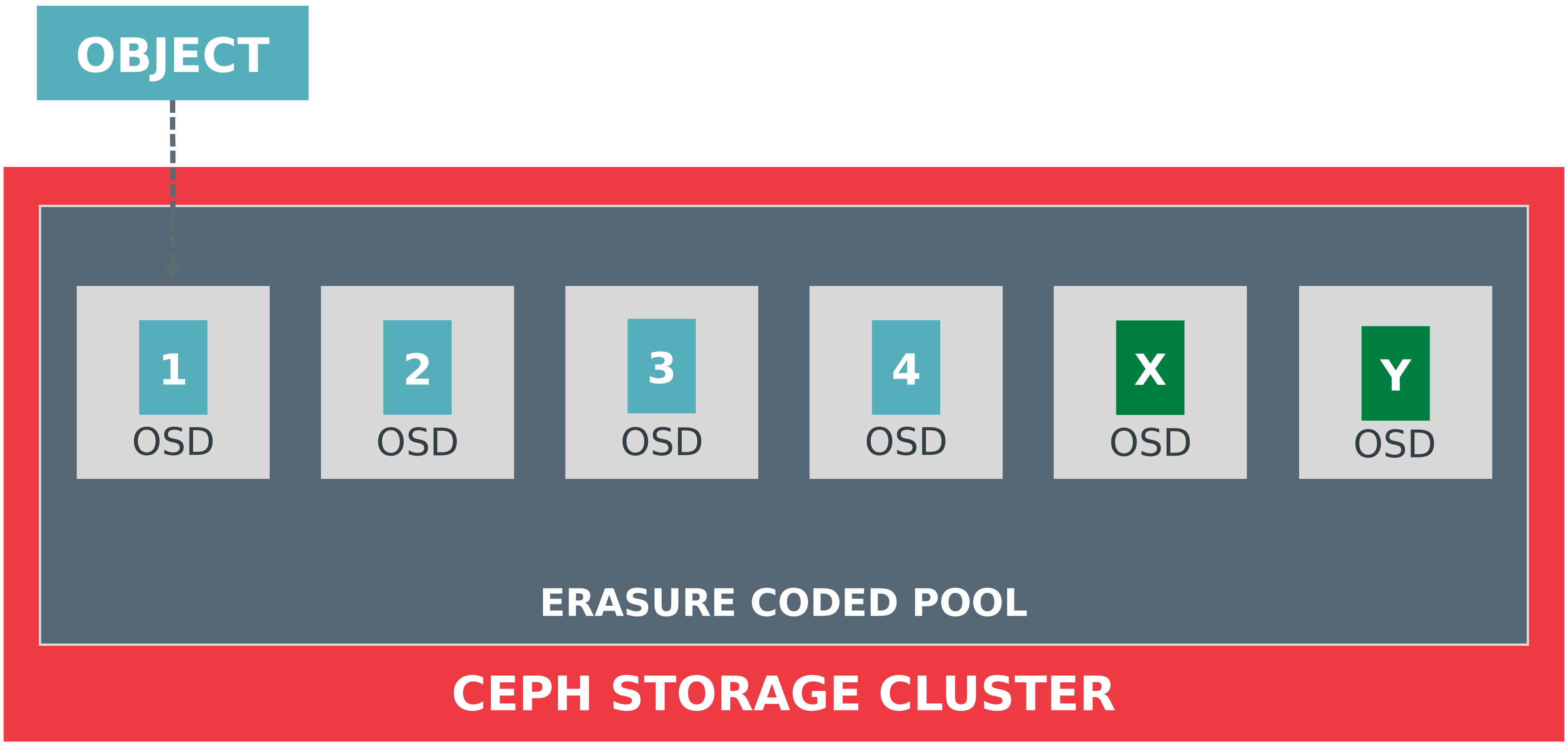
- Very high durability
- 3x (200% overhead)
- Quicker recovery



One copy plus parity

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

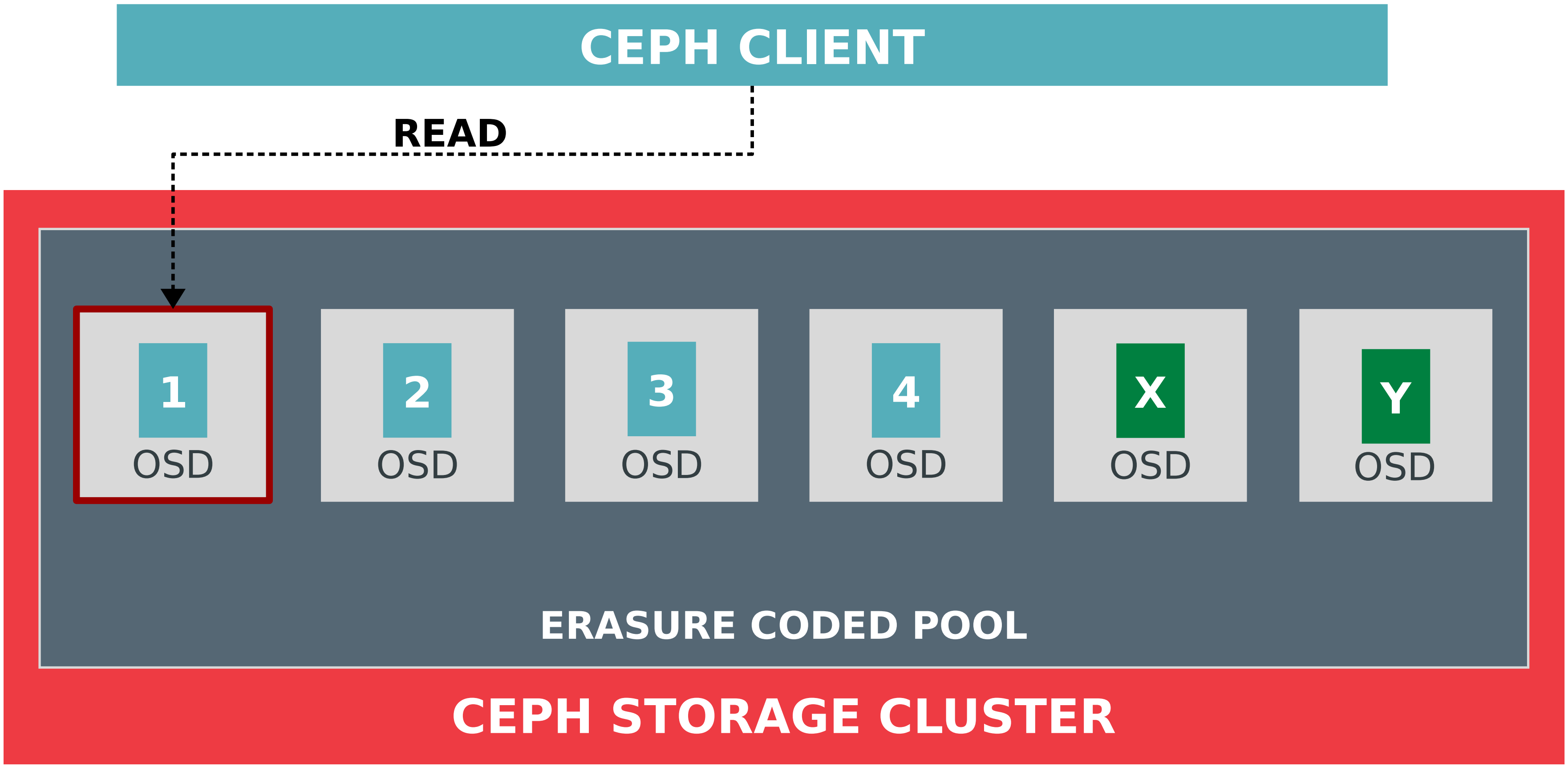
ERASURE CODING SHARDS



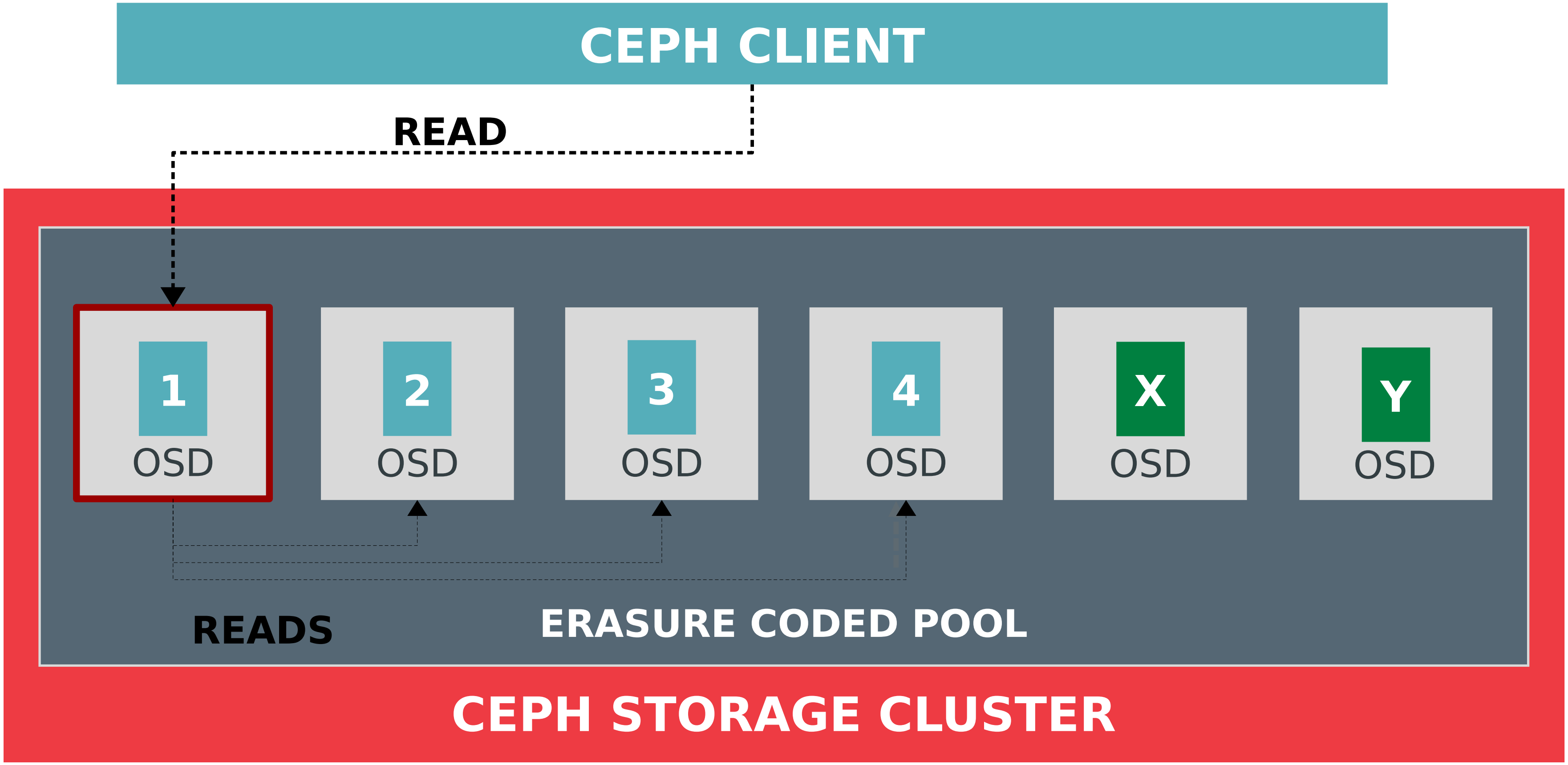
ERASURE CODING SHARDS

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

EC READ

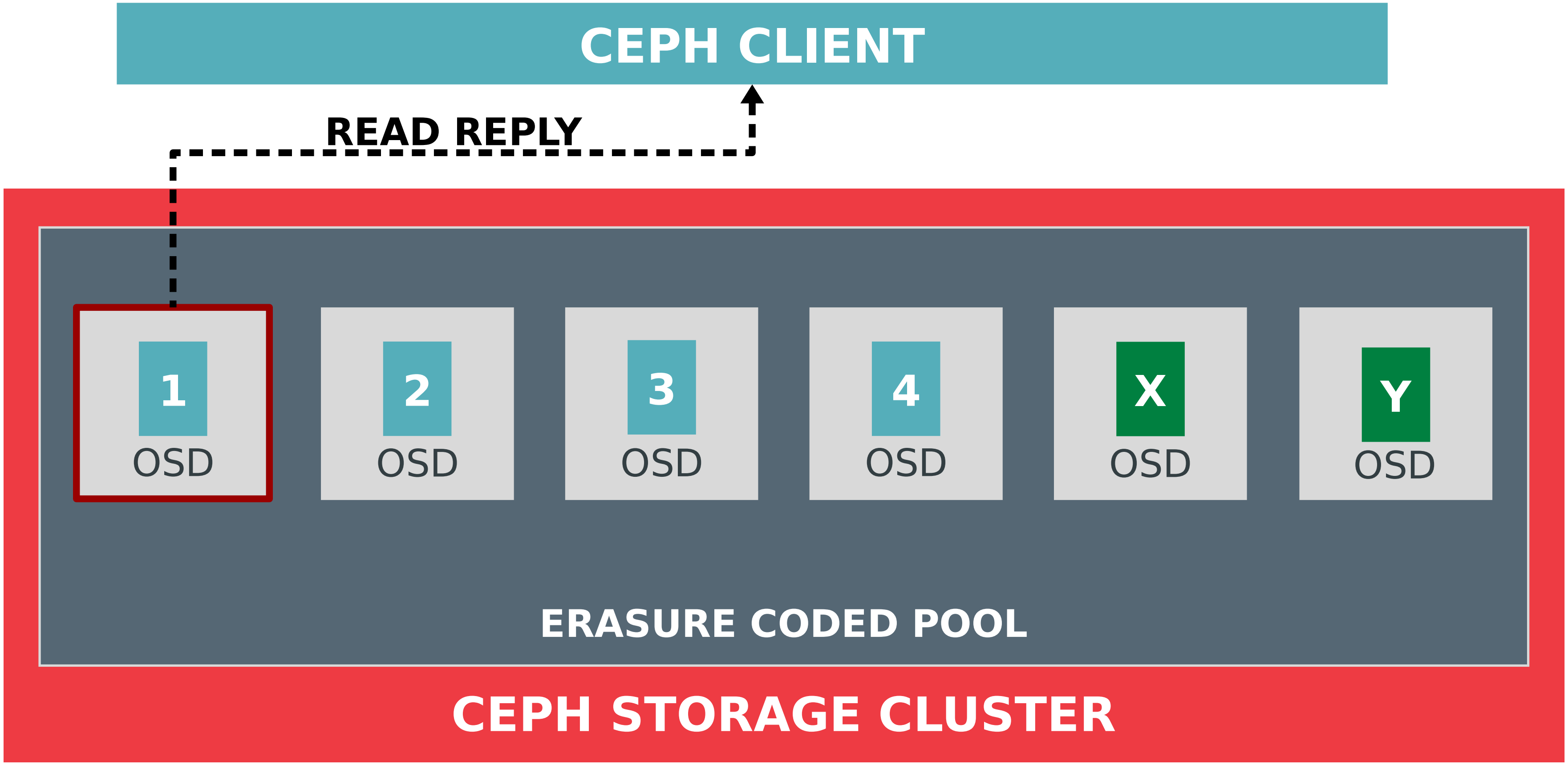

CEPH CLIENT
READ
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
ERASURE CODED POOL
CEPH STORAGE CLUSTER

EC READ

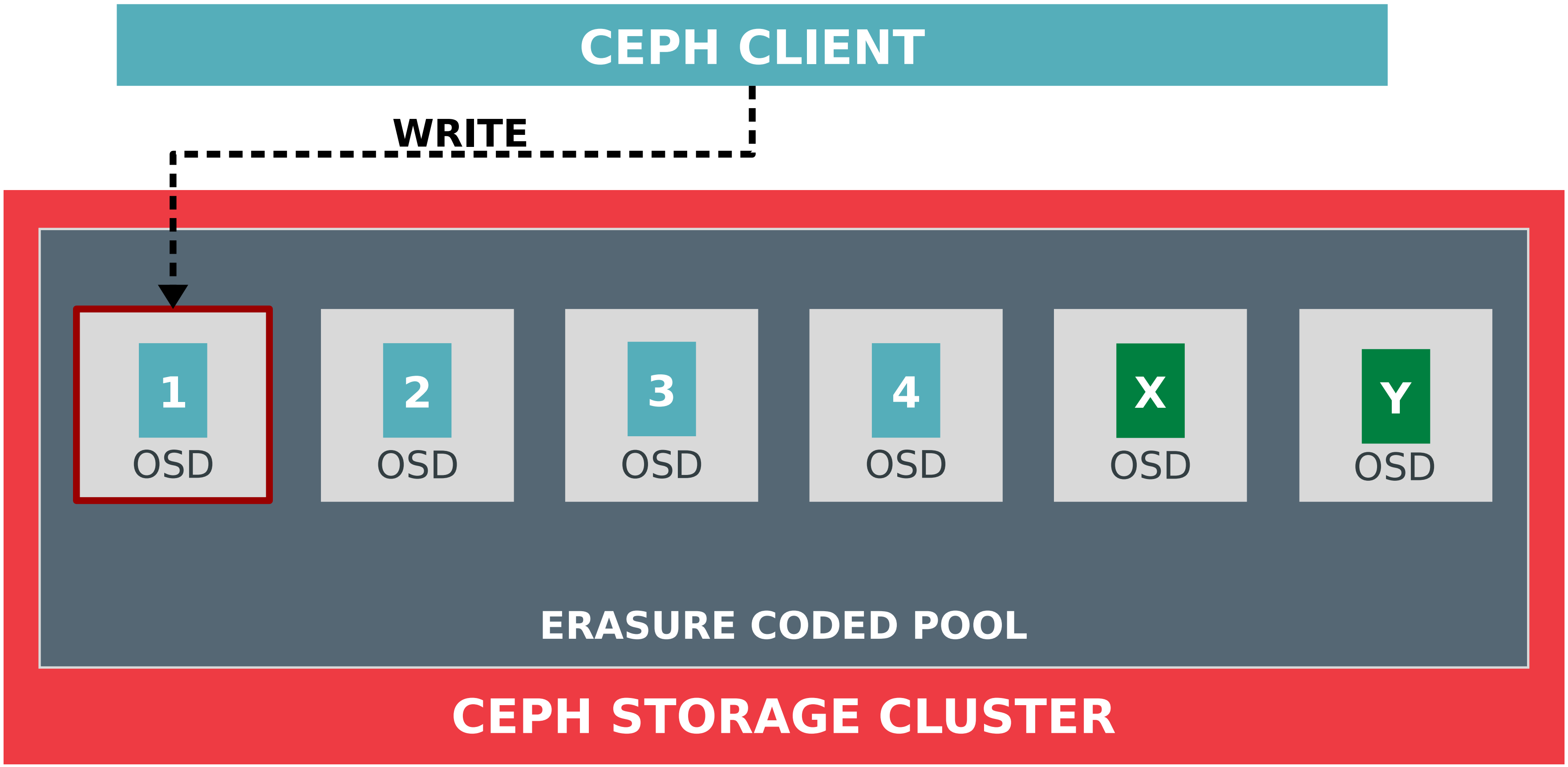


CEPH CLIENT
READ
1
OSD
2
OSD
3
OSD
4
OSD
X
OSD
Y
OSD
READS
ERASURE CODED POOL
CEPH STORAGE CLUSTER

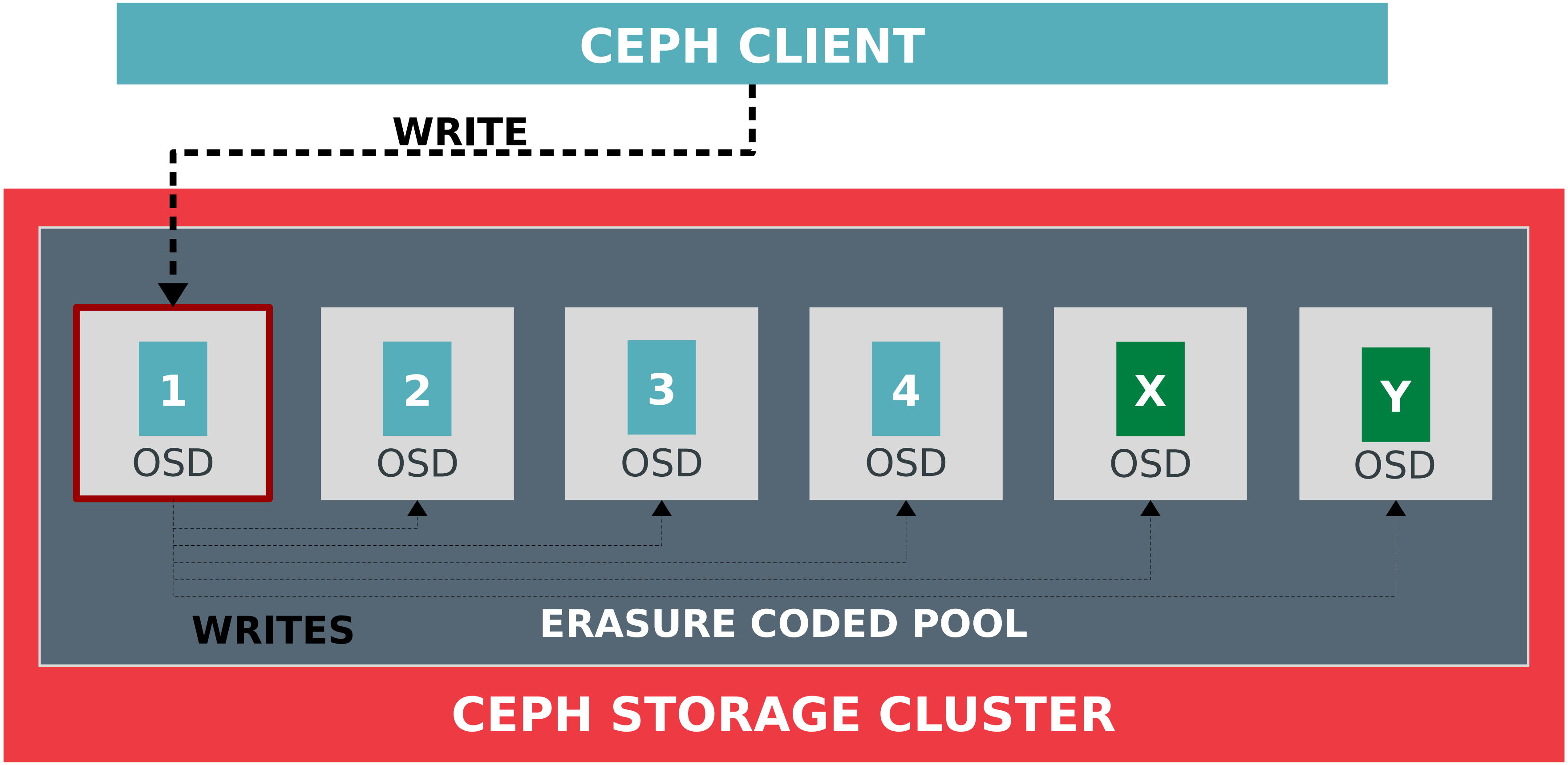
EC READ


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

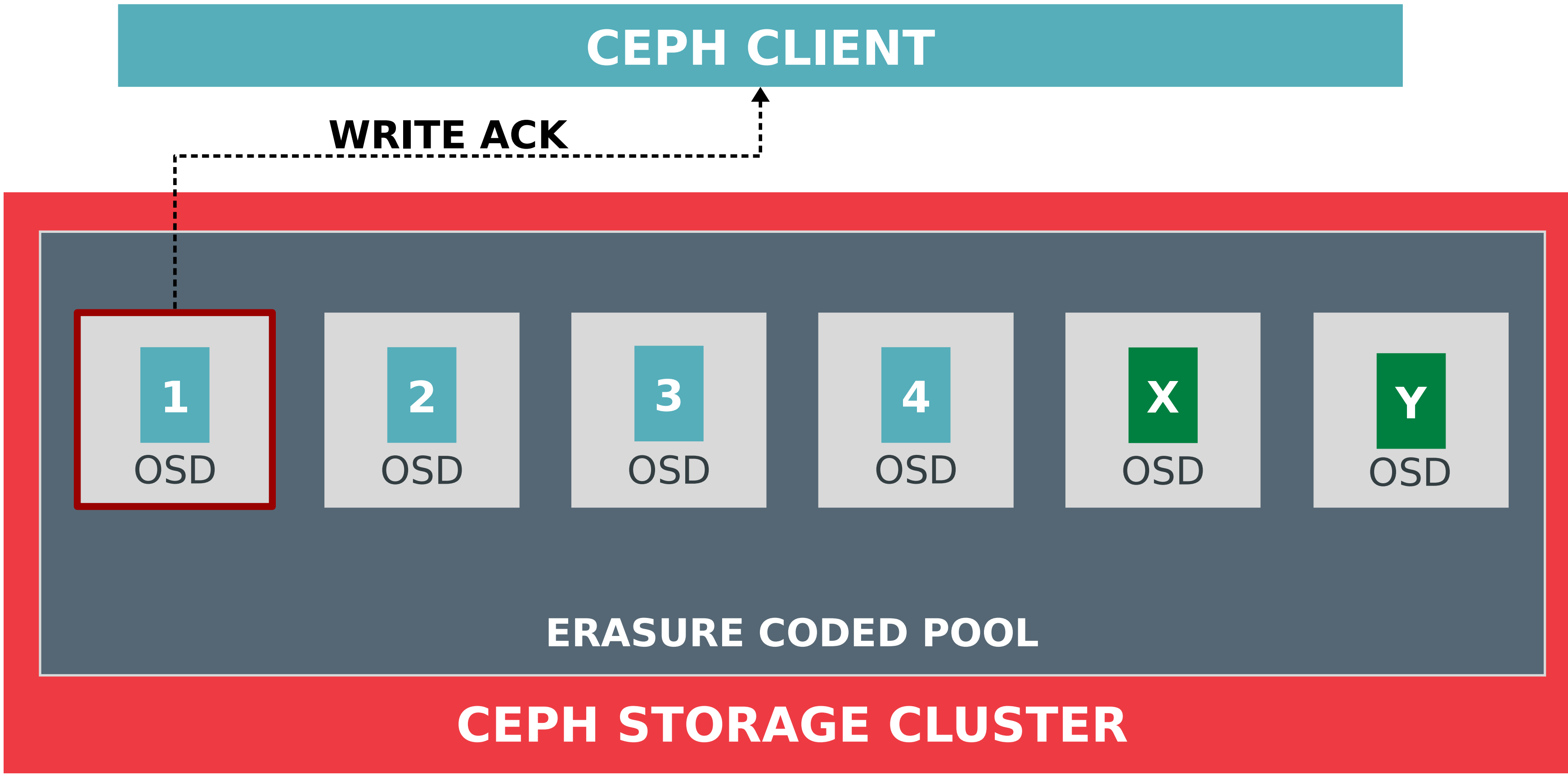
EC WRITE


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

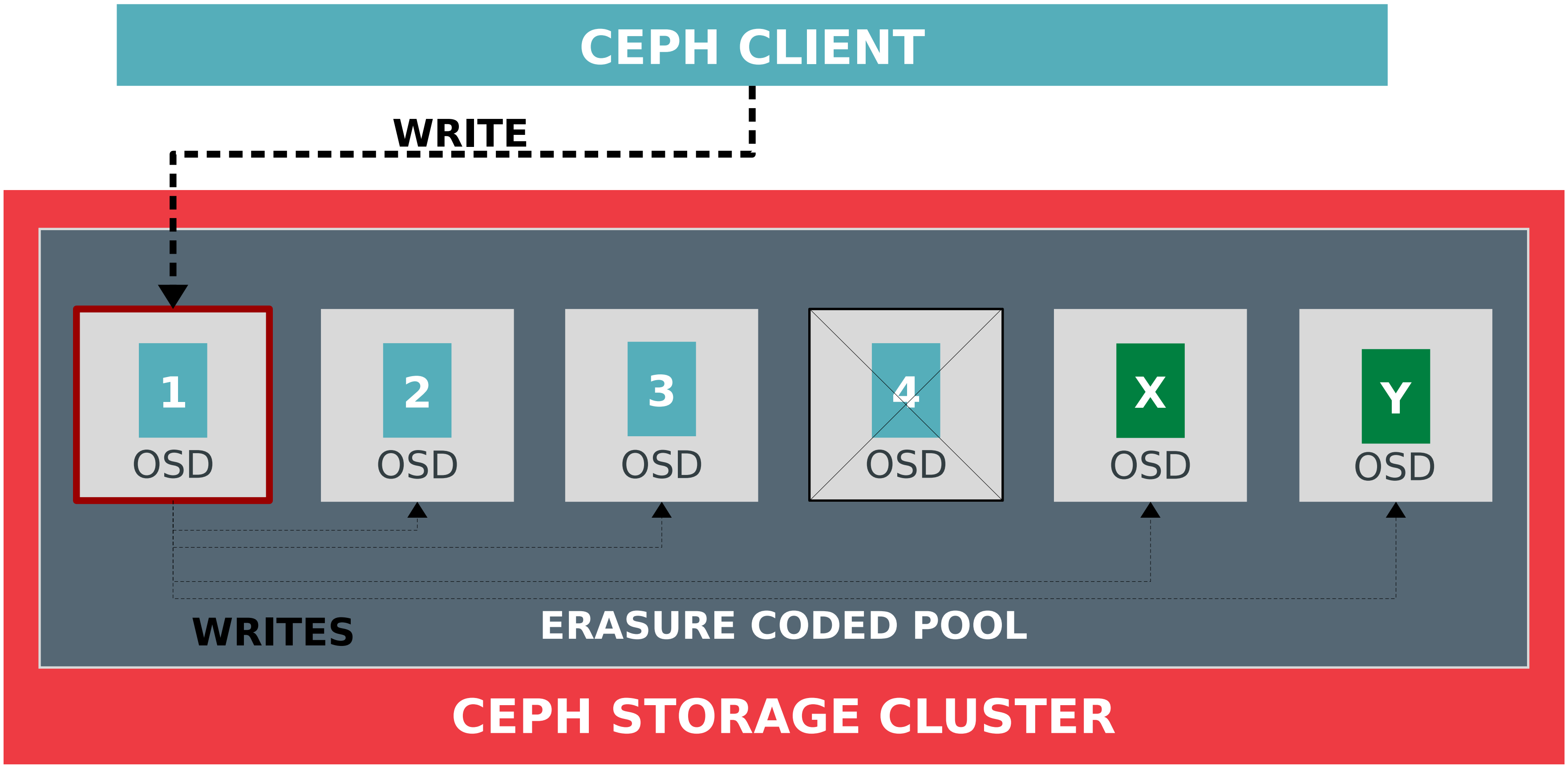
EC WRITE


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

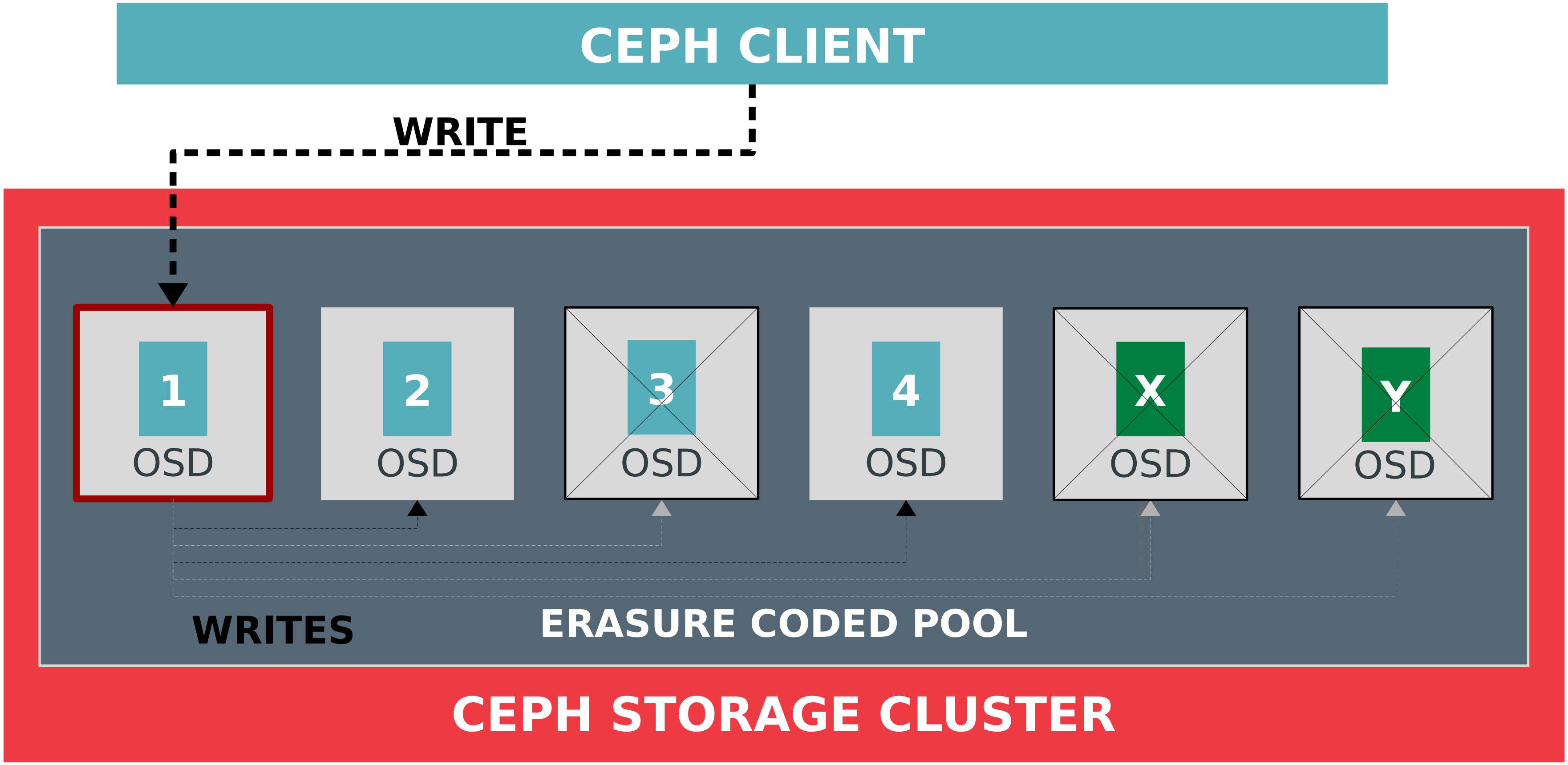
EC WRITE


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

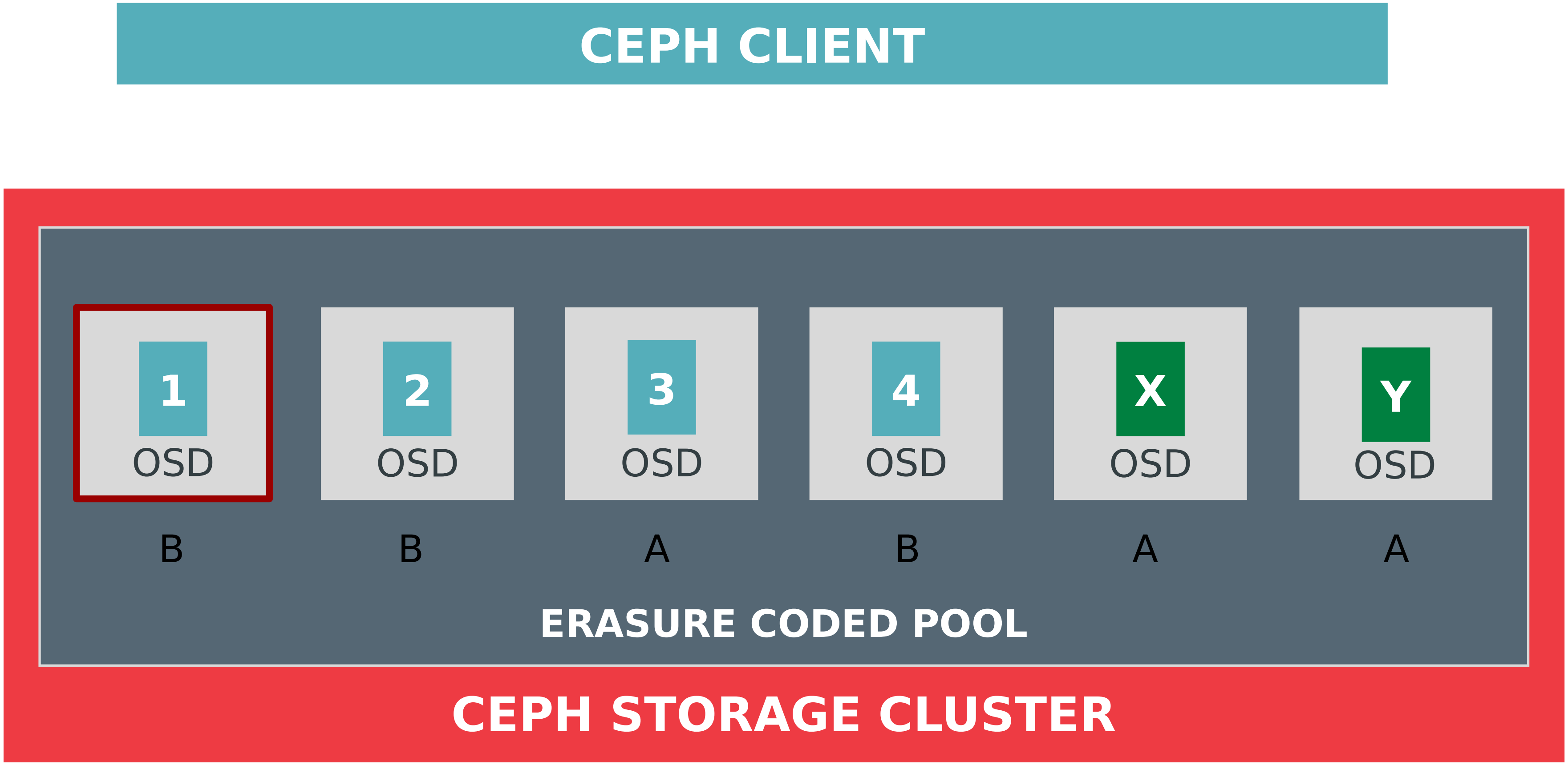
EC WRITE: DEGRADED



EC WRITE: PARTIAL FAILURE



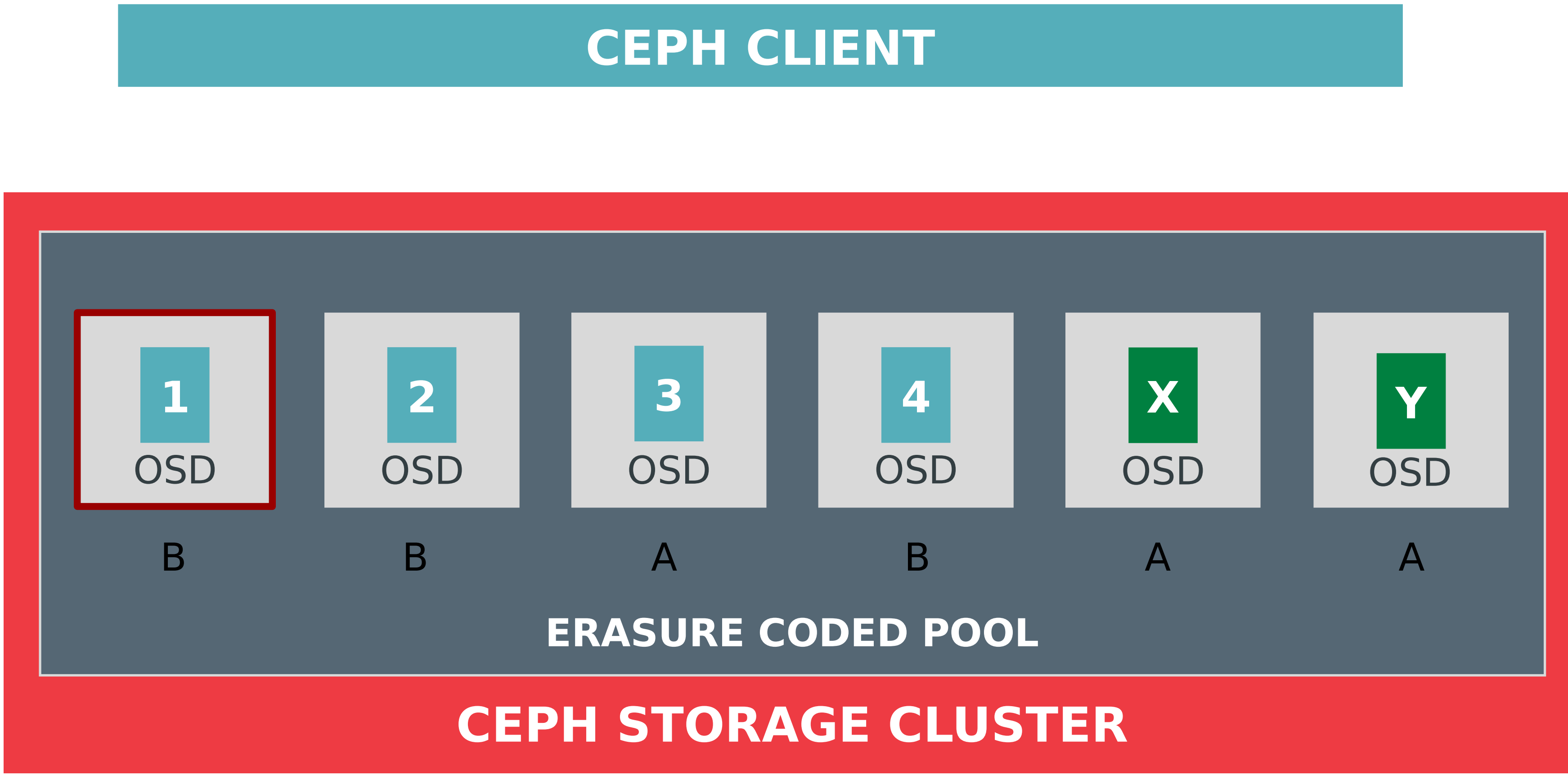
EC WRITE: PARTIAL FAILURE



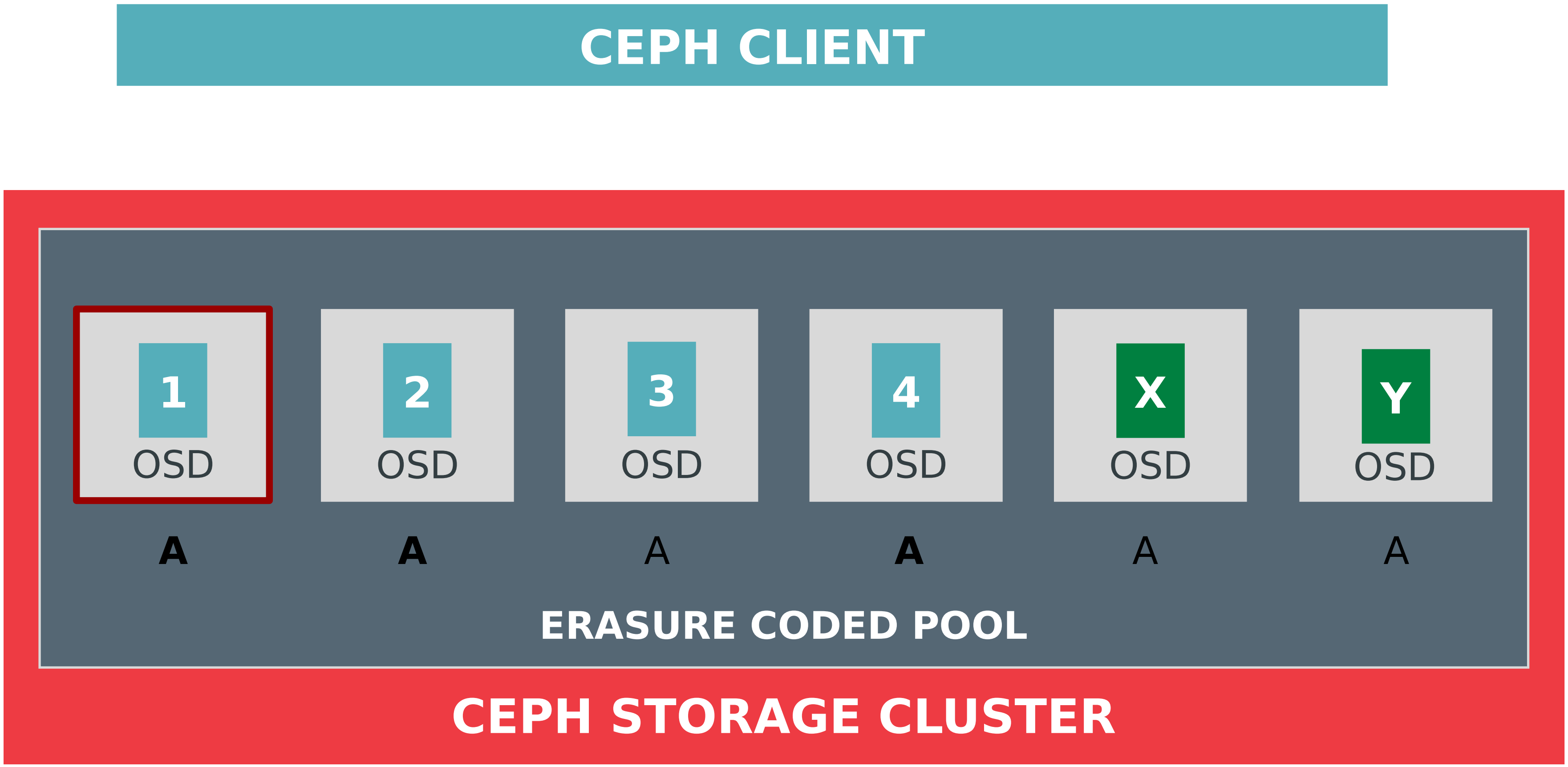
EC RESTRICTIONS

- 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

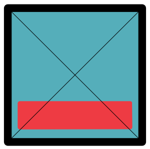
WHICH ERASURE CODE?

- 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

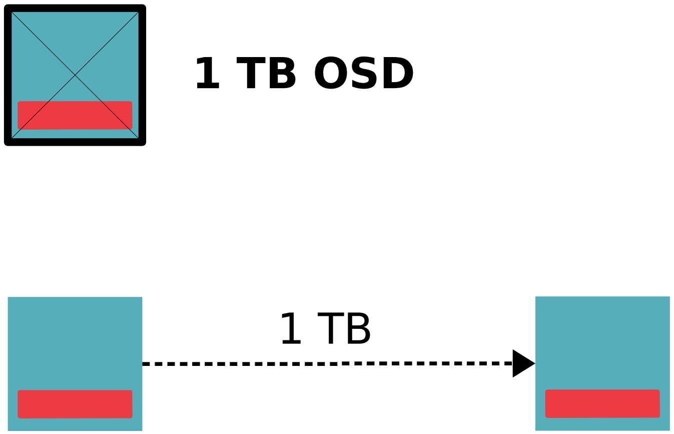
1 TB OSD

COST OF RECOVERY

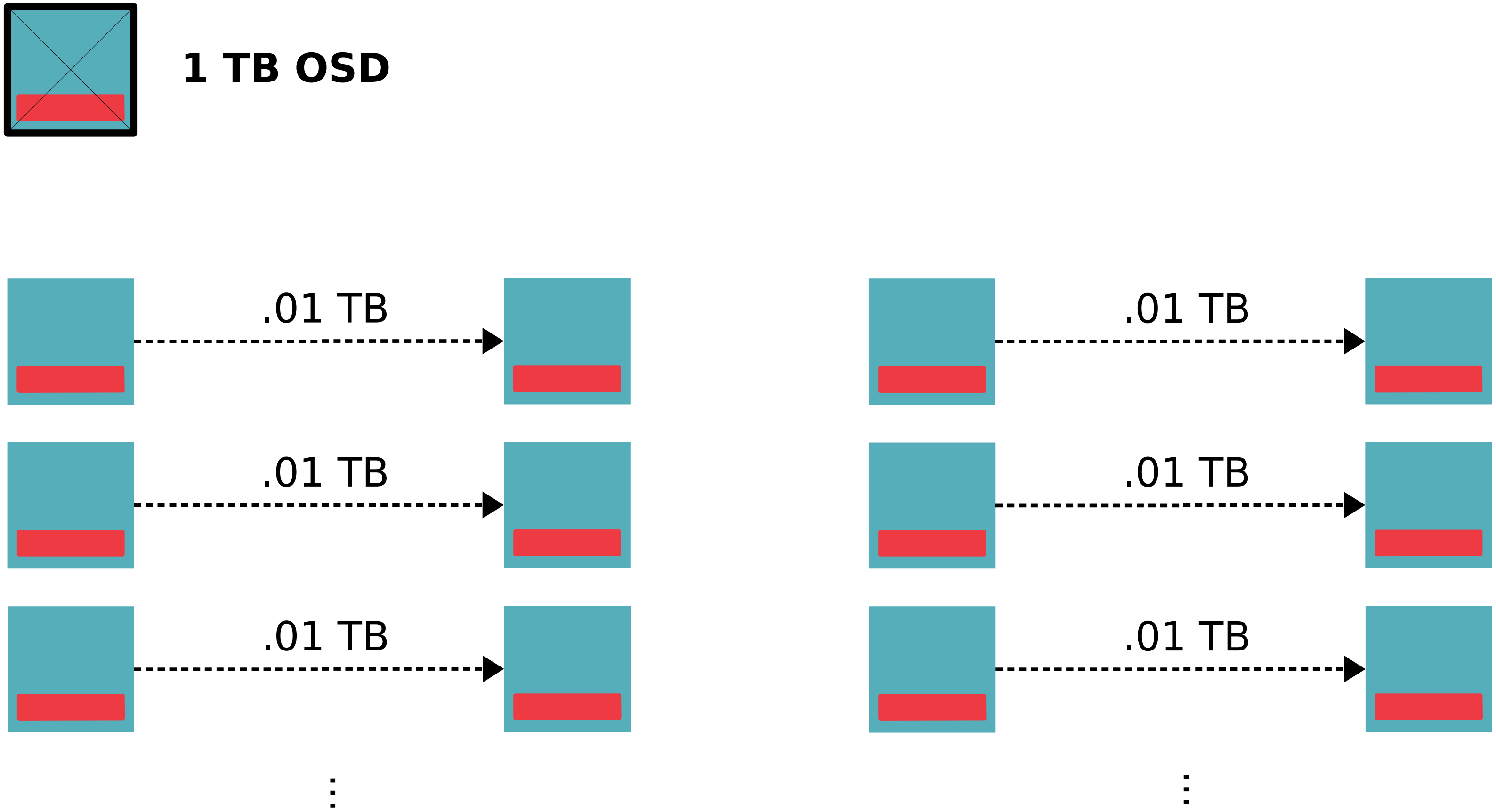


1 TB OSD

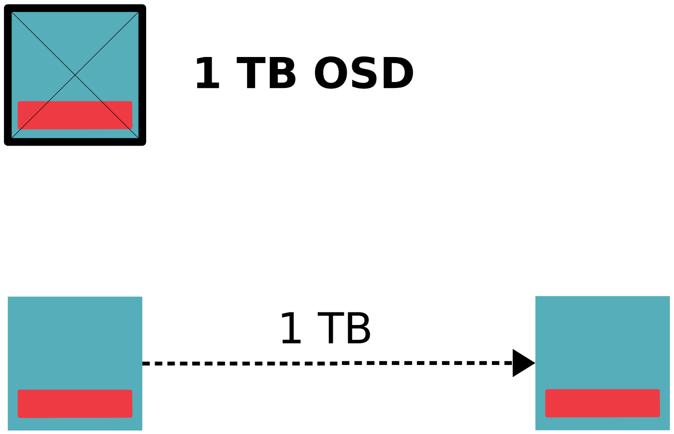
COST OF RECOVERY (REPLICATION)



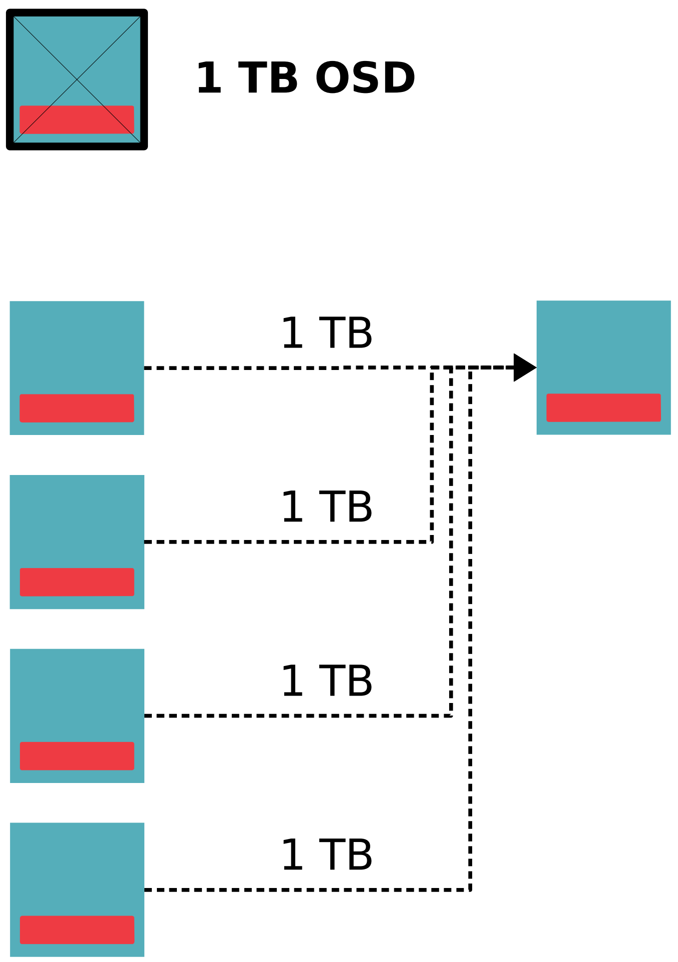
COST OF RECOVERY (REPLICATION)



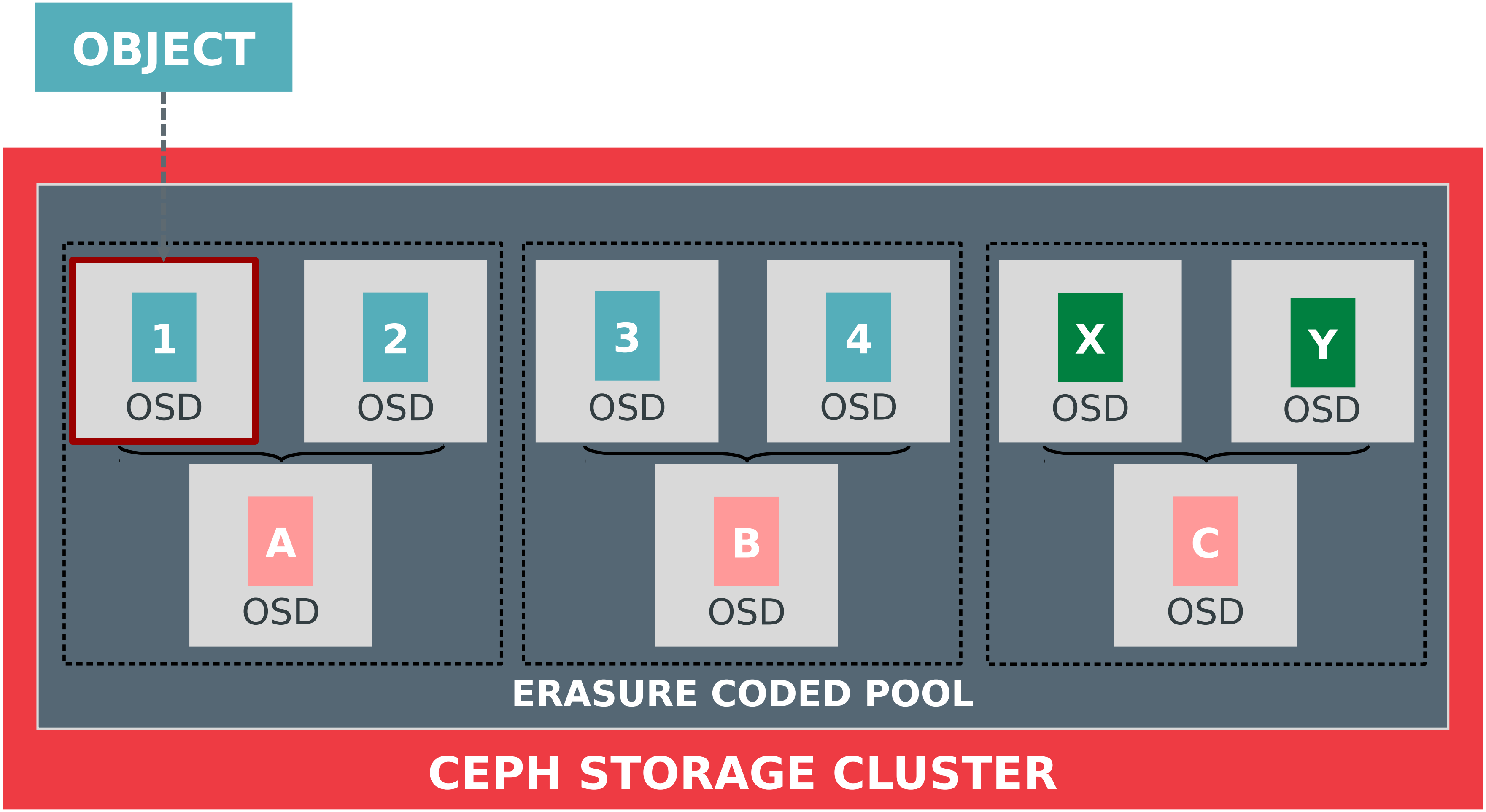
COST OF RECOVERY (REPLICATION)



COST OF RECOVERY (EC)



LOCAL RECOVERY CODE (LRC)



BIG THANKS TO

- Ceph
  - Loic Dachary (CloudWatt, FSF France, Red Hat)
  - Andreas Peters (CERN)
  - 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)

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!

Samuel Just

sjust@redhat.com