DISTRIBUTED STORAGE AND COMPUTE WITH LIBRADOS

SAGE WEIL – VAULT - 2015.03.11

AGENDA

- motivation
- what is Ceph?
- what is librados?
- what can it do?
- other RADOS goodies
- a few use cases



MOTIVATION

MY FIRST WEB APP

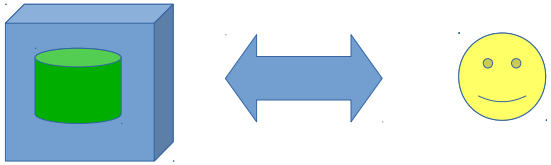
- a bunch of data files

  /srv/myapp/12312763.jpg

  /srv/myapp/87436413.jpg

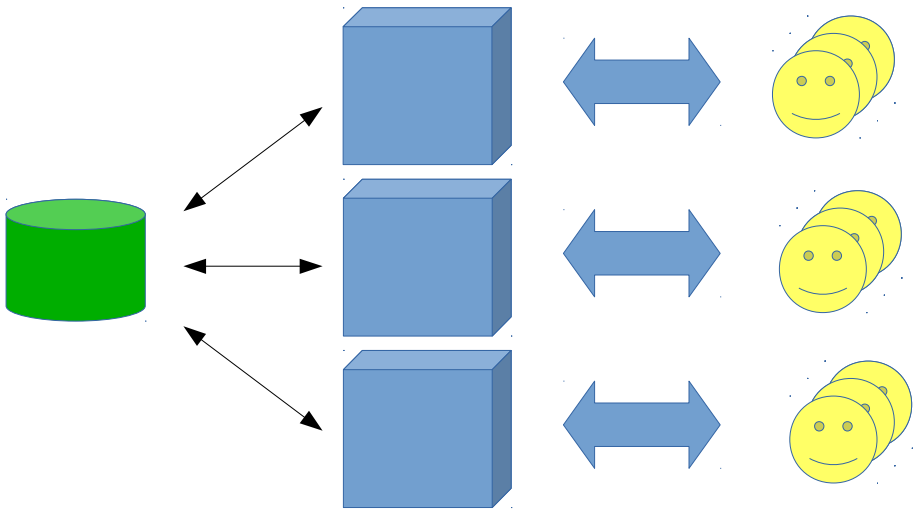
  /srv/myapp/47464721.jpg

  ...



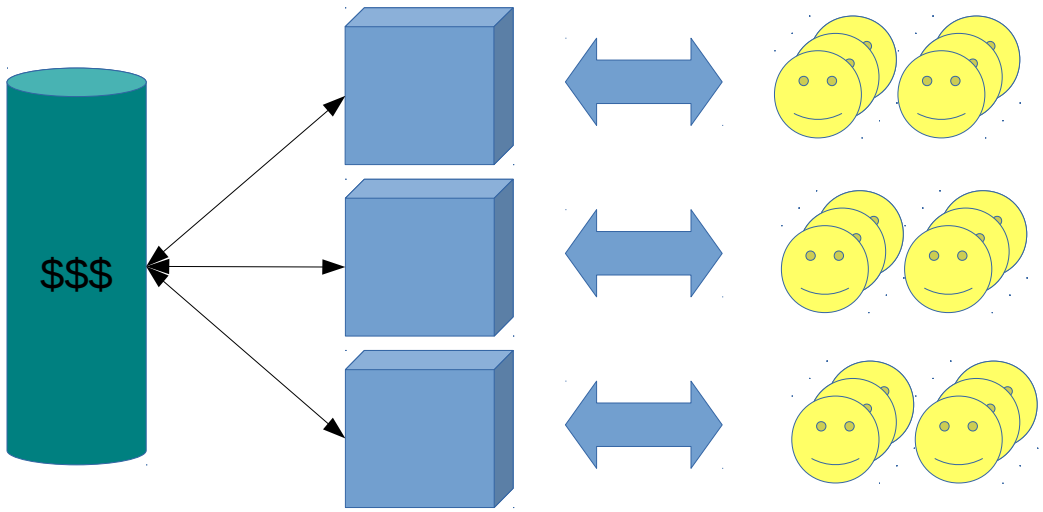
ACTUAL USERS

- scale up
  - buy a bigger, more expensive file server



SOMEBODY TWEETED

- multiple web frontends
  - NFS mount /srv/myapp



NAS COSTS ARE NON-LINEAR

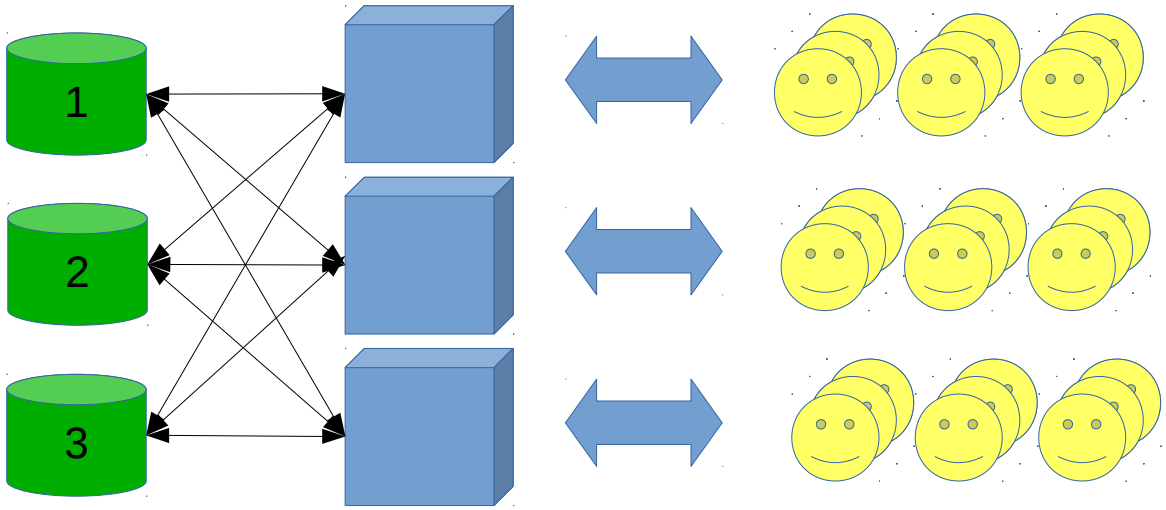
- scale out: hash files across servers

  /srv/myapp/1/1237436.jpg

  /srv/myapp/2/2736228.jpg

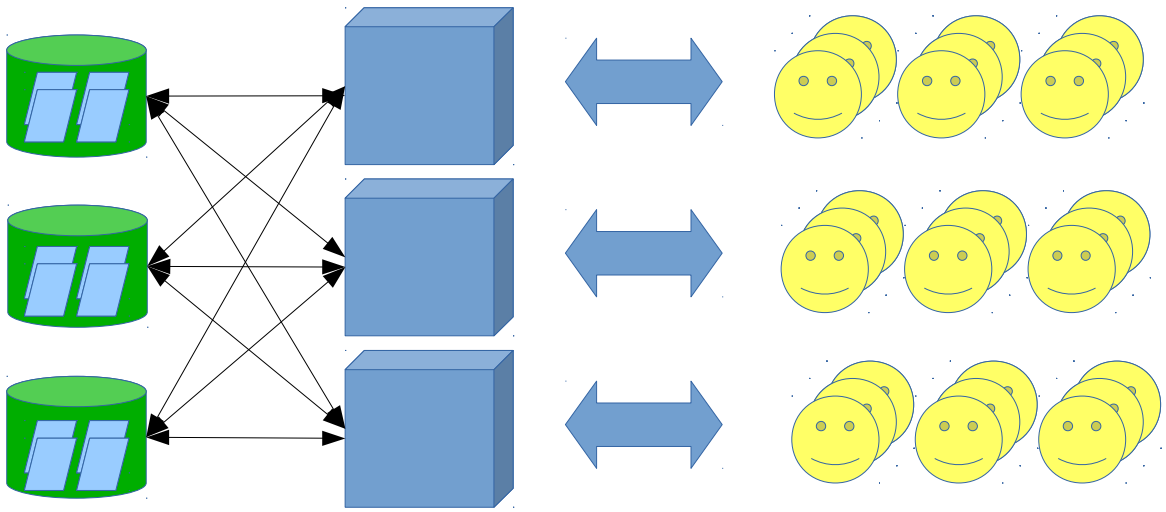
  /srv/myapp/3/3472722.jpg

  ...



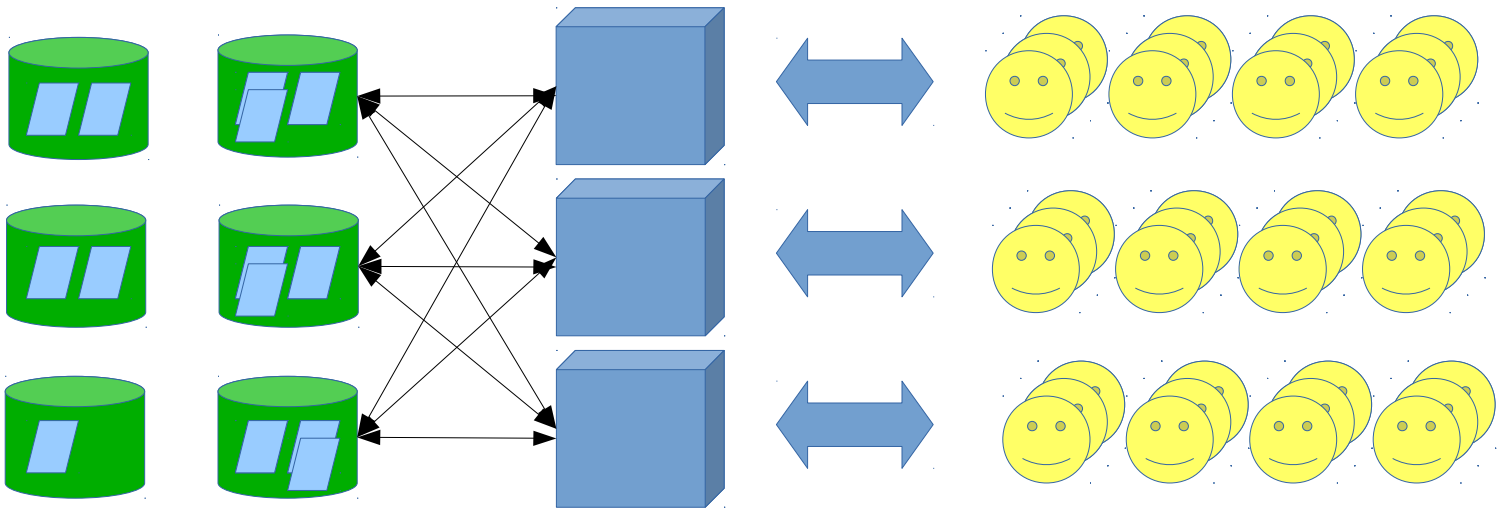
SERVERS FILL UP

- ...and directories get too big
- hash to shards that are smaller than servers



LOAD IS NOT BALANCED

- migrate smaller shards
  - probably some rsync hackery
  - maybe some trickery to maintain consistent view of data

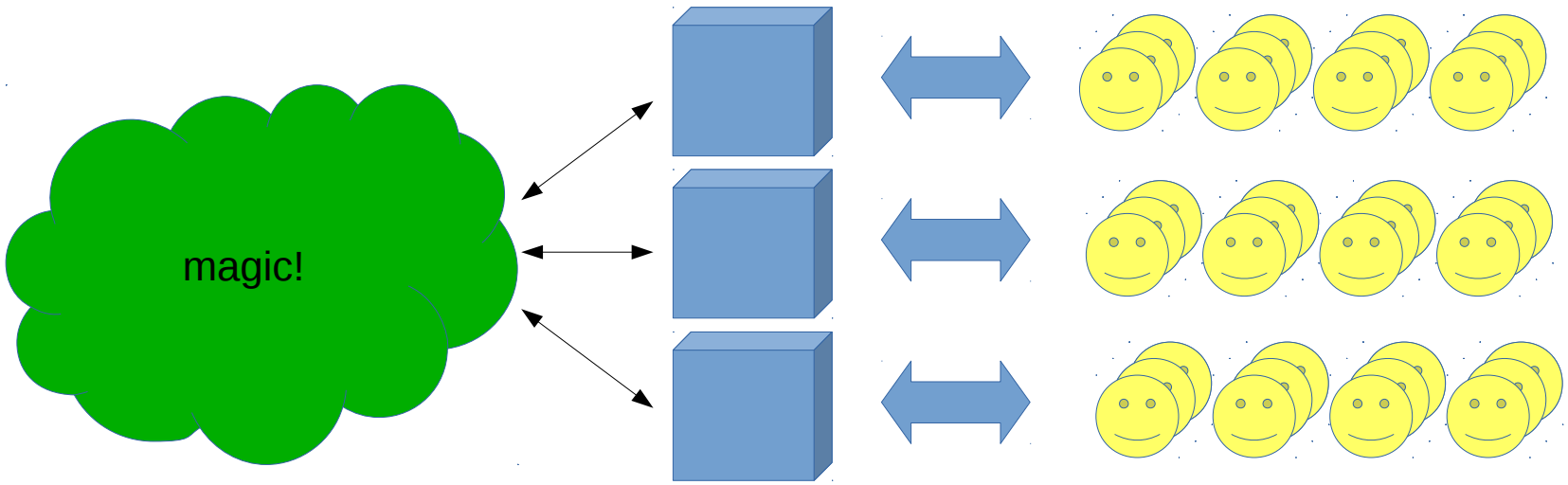


IT'S 2014 ALREADY

- don't reinvent the wheel
  - ad hoc sharding
  - load balancing
- reliability? replication?

DISTRIBUTED OBJECT STORES

- we want transparent
  - scaling, sharding, rebalancing
  - replication, migration, healing
- simple, flat(ish) namespace



CEPH

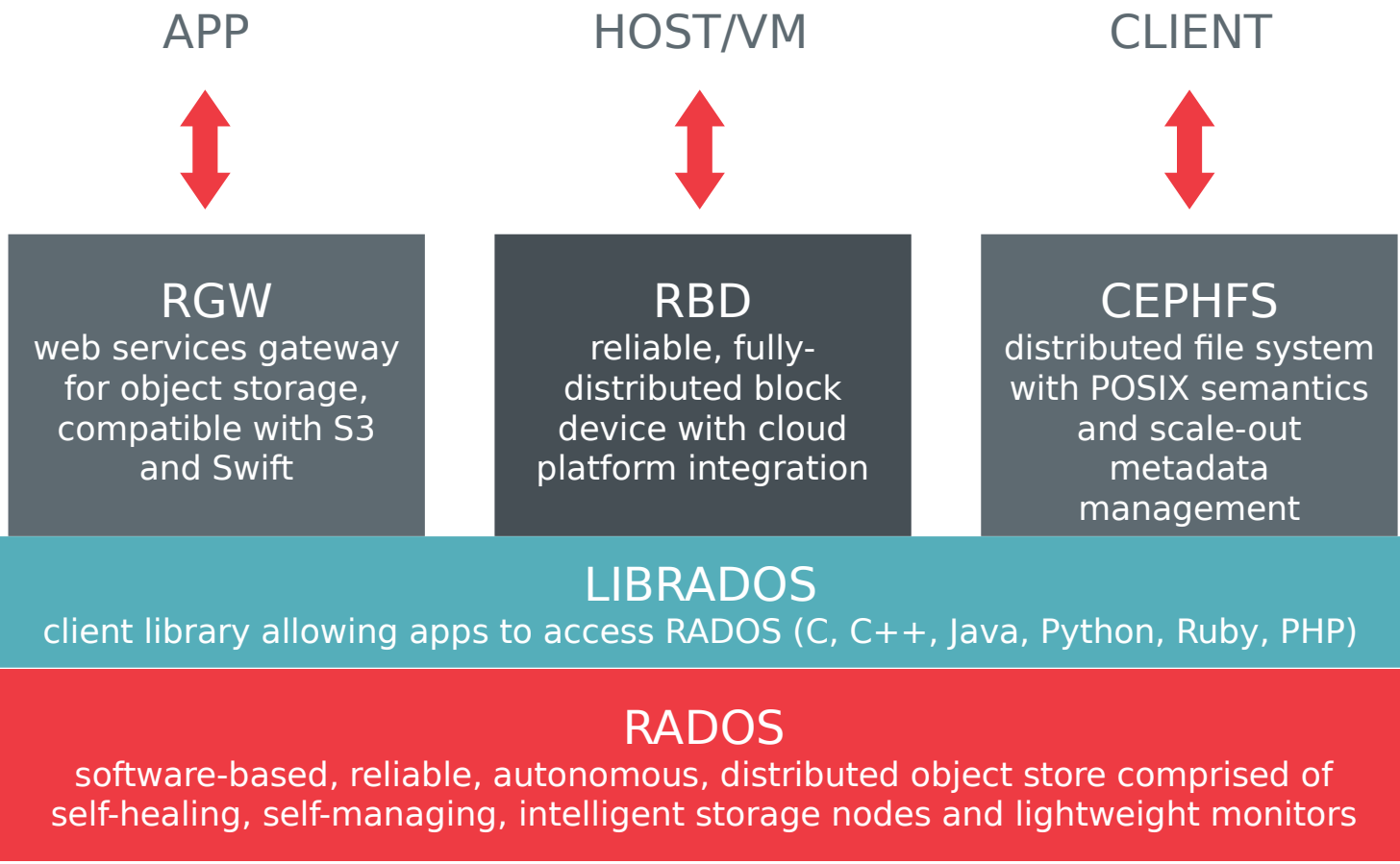
CEPH MOTIVATING PRINCIPLES

- everything must scale horizontally
- no single point of failure
- commodity hardware
- self-manage whenever possible
- move beyond legacy approaches
  - client/cluster instead of client/server
  - avoid ad hoc high-availability
- open source (LGPL)

ARCHITECTURAL FEATURES

- smart storage daemons
  - centralized coordination of dumb devices does not scale
  - peer to peer, emergent behavior
- flexible object placement
  - “smart” hash-based placement (CRUSH)
  - awareness of hardware infrastructure, failure domains
- no metadata server or proxy for finding objects
- strong consistency (CP instead of AP)

CEPH COMPONENTS



CEPH COMPONENTS

ENLIGHTENED APP

LIBRADOS

client library allowing apps to access RADOS (C, C++, Java, Python, Ruby, PHP)

RADOS

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

LIBRADOS

LIBRADOS

- native library for accessing RADOS
  - librados.so shared library
  - C, C++, Python, Erlang, Haskell, PHP, Java (JNA)
- direct data path to storage nodes
  - speaks native Ceph protocol with cluster
- exposes
  - mutable objects
  - rich per-object API and data model
- hides
  - data distribution, migration, replication, failures

OBJECTS

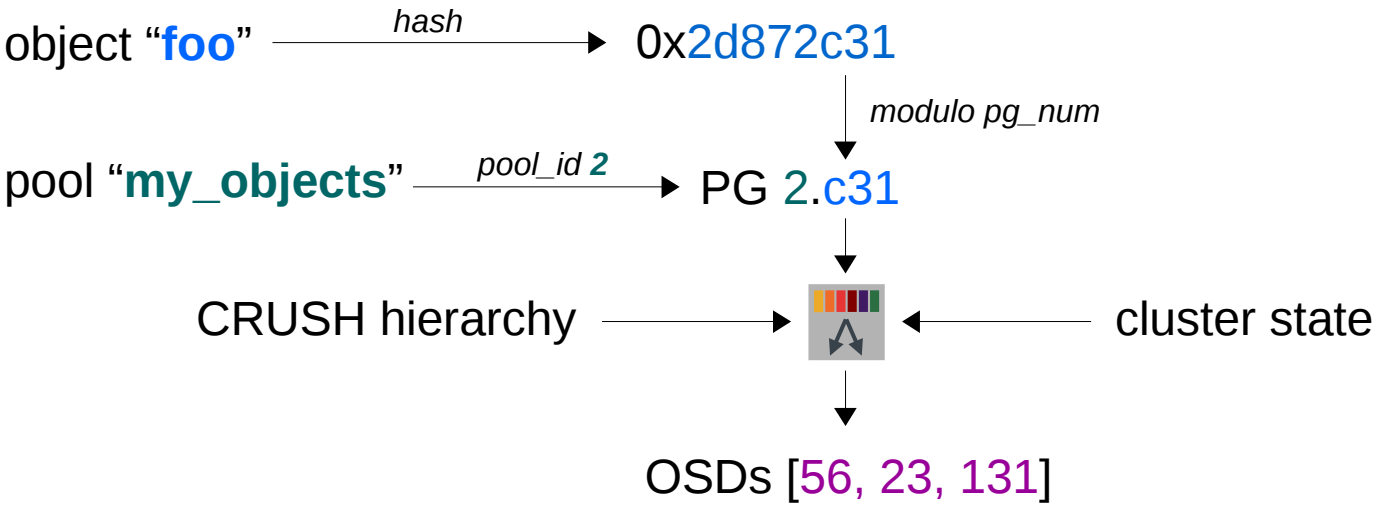
- name
  - alphanumeric
  - no rename
- data
  - opaque byte array
  - bytes to 100s of MB
  - byte-granularity access (just like a file)
- attributes
  - small
  - e.g., “version=12”
- key/value data
  - random access insert, remove, list
  - keys (bytes to 100s of bytes)
  - values (bytes to megabytes)
  - key-granularity access

POOLS

- name
- many objects
  - bazillions
  - independent namespace
- replication and placement policy
  - 3 replicas separated across racks
  - 8+2 erasure coded, separated across hosts
- sharding, (cache) tiering parameters

DATA PLACEMENT

- there is no metadata server, only OSDMap
  - pools, their ids, and sharding parameters
  - OSDs (storage daemons), their IPs, and up/down state
  - CRUSH hierarchy and placement rules
  - 10s to 100s of KB



EXPLICIT DATA PLACEMENT

- you don't choose data location
- except relative to other objects
  - normally we hash the object name
  - you can also explicitly specify a different string
  - and remember it on read, too

object “**foo**” —*hash*→ 0x2d872c31

object “**bar**”
key “**foo**” —*hash*→ 0x2d872c31

HELLO, WORLD

```
#include <rados/librados.hpp>

librados::Rados cluster;
cluster.init("admin");
cluster.connect();

librados::IoCtx p;
rados.ioctx_create("my_pool", p);

bufferlist data;
data.append("Hello, world!");
p.write_full("my object", data);

bufferlist attr;
attr.append("1");
p.setxattr("my object", "version", attr);

bufferlist data2;
p.read("my object", data2, data.length(), 0);
assert(data.contents_equal(data2));

cluster.shutdown();
```

connect to the cluster

p is like a file descriptor

atomically write/replace object

COMPOUND OBJECT OPERATIONS

- group operations on object into single request
  - atomic: all operations commit or do not commit
  - idempotent: request applied exactly once

```
bufferlist data;
data.append("Hello, world!");
bufferlist attr;
attr.append("1");

ObjectWriteOperation op;
op.create(true);  // exclusive create
op.write_full(data);
op.setxattr("version", attr);

p.operate("my object", &op);
```

CONDITIONAL OPERATIONS

- mix read and write ops
- overall operation aborts if any step fails
- 'guard' read operations verify condition is true
  - verify xattr has specific value
  - assert object is a specific version
- allows atomic compare-and-swap

```
bufferlist newdata, attr1, attr2;
newdata.append("Hello, Perth!");
attr1.append("1");
attr2.append("2");

ObjectWriteOperation op;
op.cmpxattr("version", LIBRADOS_CMPXATTR_OP_EQ, attr1);
op.write_full(newdata);   // implicit truncate before write
op.setxattr("version", attr2);

p.operate("my object", &op);
```

KEY/VALUE DATA

- each object can contain key/value data
  - independent of byte data or attributes
  - random access insertion, deletion, range query/list
- good for structured data
  - avoid read/modify/write cycles
  - RGW bucket index
    - enumerate objects and there size to support listing
  - CephFS directories
    - efficient file creation, deletion, inode updates

SNAPSHOTS

- object granularity
  - RBD has per-image snapshots
  - CephFS can snapshot any subdirectory
- librados user must cooperate
  - provide “snap context” at write time
  - allows for point-in-time consistency without flushing caches
- triggers copy-on-write inside RADOS
  - consume space only when snapshotted data is overwritten

RADOS CLASSES

- write new RADOS “methods”
  - code runs directly inside storage server I/O path
  - simple plugin API; admin deploys a .so
- read-side methods
  - process data, return result
- write-side methods
  - process, write; read, modify, write
  - generate an update transaction that is applied atomically

A SIMPLE CLASS METHOD

```
int compute_md5(cls_method_context_t hctx, bufferlist *in, bufferlist *out)
{
  size_t size;
  int ret = cls_cxx_stat(hctx, &size, NULL);
  if (ret < 0)
    return ret;

  bufferlist data;
  ret = cls_cxx_read(hctx, 0, size, data);
  if (ret < 0)
    return ret;

  byte digest[16];
  MD5().CalculateDigest(digest, (byte*)data.c_str(), data.length());
  out->append(digest, sizeof(digest));
  return 0;
}
```

INVOKING A METHOD

```
ObjectReadOperation op;

uint64_t size;
time_t mtime;
op.stat(&size, &mtime, NULL);

bufferlist in, out;
op.exec("my_class", "compute_md5", in, &out);

int r = p.operate("my object", &op);

assert(out.length() == 16 || r < 0);
cout << "size " << size << ", mtime " << mtime << endl;
out.hexdump(cout);
```

EXAMPLE: RBD

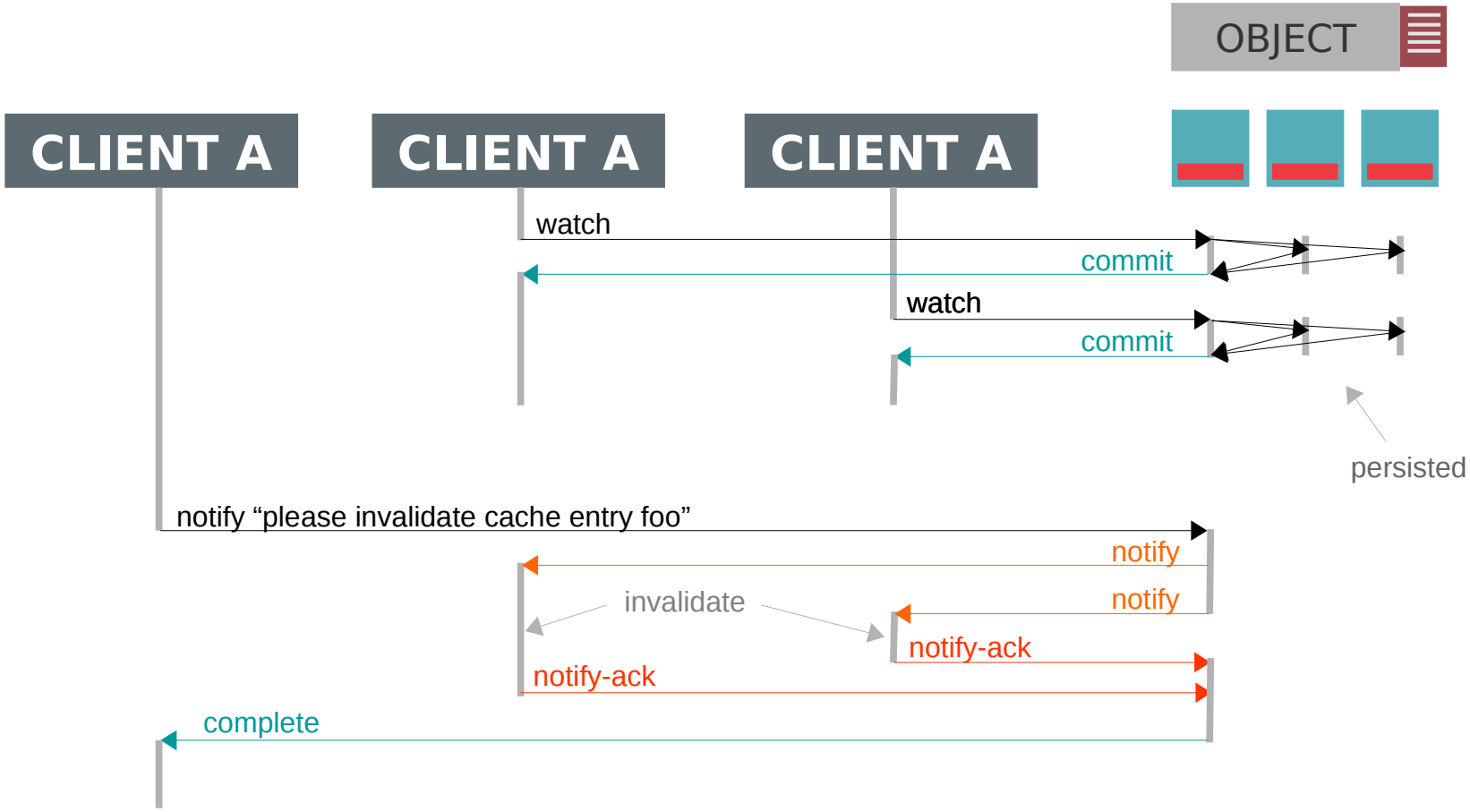
- RBD (RADOS block device)
- image data striped across 4MB data objects
- image header object
  - image size, snapshot info, lock state
- image operations may be initiated by any client
  - image attached to KVM virtual machine
  - 'rbd' CLI may trigger snapshot or resize
- need to communicate between librados client!

WATCH/NOTIFY

- establish stateful 'watch' on an object
  - client interest persistently registered with object
  - client keeps connection to OSD open
- send 'notify' messages to all watchers
  - notify message (and payload) sent to all watchers
  - notification (and reply payloads) on completion
- strictly time-bounded liveness check on watch
  - no notifier falsely believes we got a message

- example: distributed cache w/ cache invalidations

WATCH/NOTIFY



A FEW USE CASES

SIMPLE APPLICATIONS

- cls_lock – cooperative locking
- cls_refcount – simple object refcounting
- images
  - rotate, resize, filter images
- log or time series data
  - filter data, return only matching records
- structured metadata (e.g., for RBD and RGW)
  - stable interface for metadata objects
  - safe and atomic update operations

DYNAMIC OBJECTS IN LUA

- Noah Wakins (UCSC)
  - http://ceph.com/rados/dynamic-object-interfaces-with-lua/
- write rados class methods in LUA
  - code sent to OSD from the client
  - provides LUA view of RADOS class runtime
- LUA client wrapper for librados
  - makes it easy to send code to exec on OSD

```
local script = [[
function say_hello(input, output)
  output:append("Hello, ")
  if #input == 0 then
    output:append("world")
  else
    output:append(input:str())
  end
  output:append("!")
end
cls.register(say_hello)
]]
```

```
local ret, outdata = clslua.exec(ioctx, "oid", script, "say_hello", "")
print(outdata)

local ret, outdata = clslua.exec(ioctx, "oid", script, "say_hello", "John")
print(outdata)
```

```
Hello, world!
Hello, John!
```

VAULTAIRE

- Andrew Cowie (Anchor Systems)
- a data vault for metrics
  - https://github.com/anchor/vaultaire
  - http://linux.conf.au/schedule/30074/view_talk
  - http://mirror.linux.org.au/pub/linux.conf.au/2015/OGGB3/Thursday/
- preserve all data points (no MRTG)
- append-only RADOS objects
- dedup repeat writes on read
- stateless daemons for inject, analytics, etc.

ZLOG – CORFU ON RADOS

- Noah Watkins (UCSC)
  - http://noahdesu.github.io/2014/10/26/corfu-on-ceph.html
- high performance distributed shared log
  - use RADOS for storing log shards instead of CORFU's special-purpose storage backend for flash
  - let RADOS handle replication and durability
- cls_zlog
  - maintain log structure in object
  - enforce epoch invariants

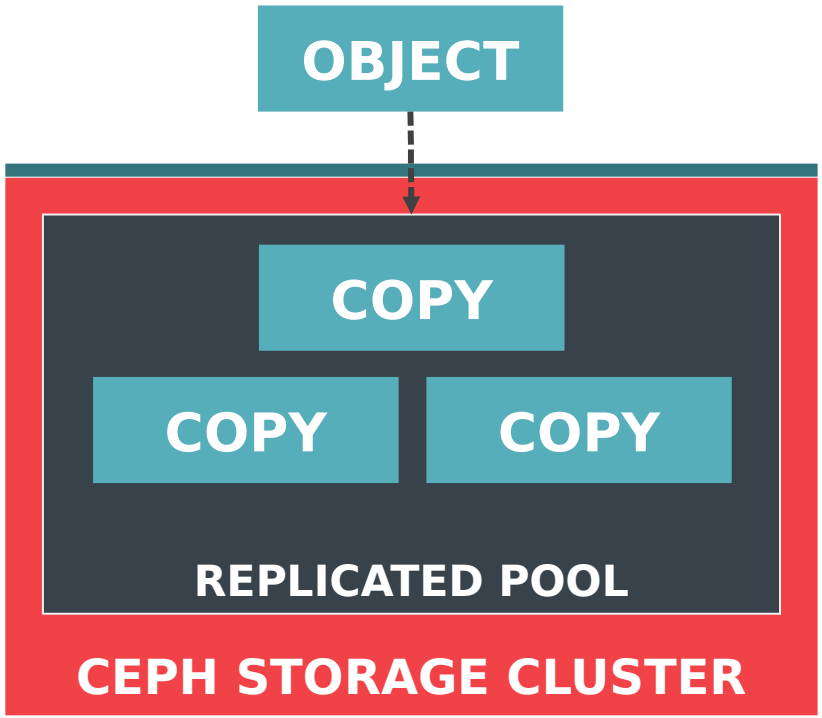
OTHERS

- radosfs
  - simple POSIX-like metadata-server-less file system
  - https://github.com/cern-eos/radosfs
- glados
  - gluster translator on RADOS
- several dropbox-like file sharing services
- iRODS
  - simple backend for an archival storage system
- Synnefo
  - open source cloud stack used by GRNET
  - Pithos block device layer implements virtual disks on top of librados (similar to RBD)



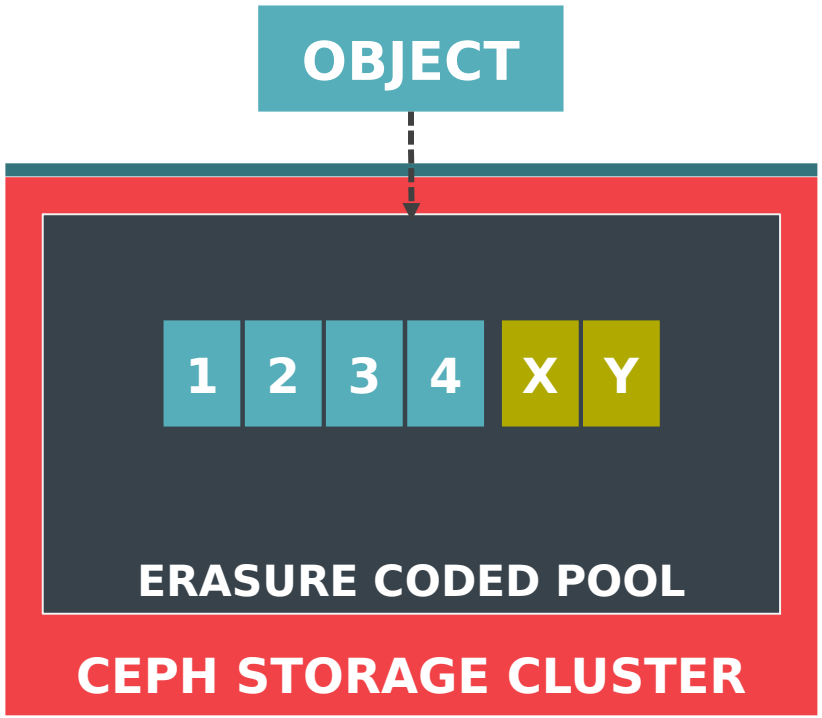
OTHER RADOS GOODIES

ERASURE CODING



Full copies of stored objects

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



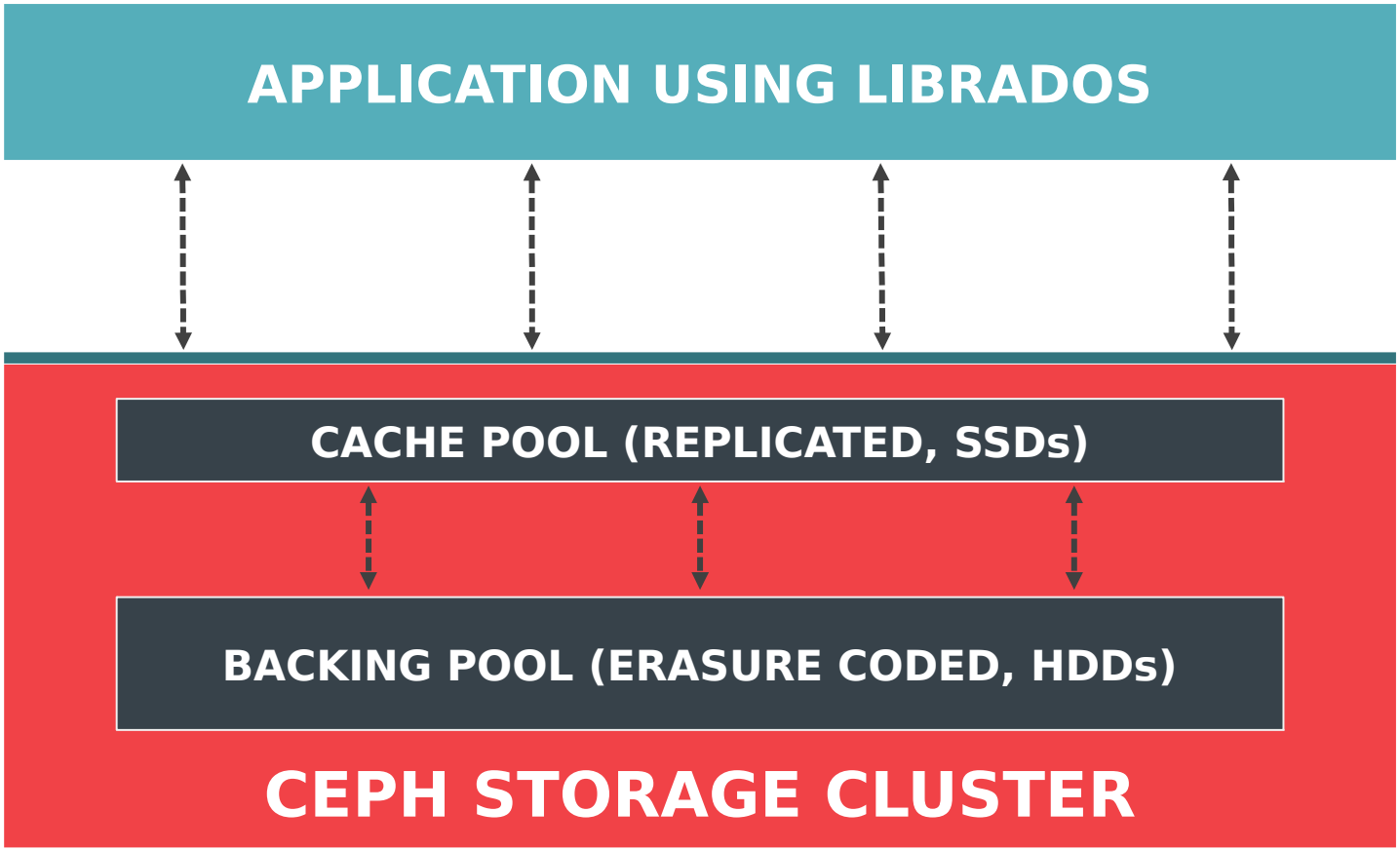
One copy plus parity

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

ERASURE CODING

- subset of operations supported for EC
  - attributes and byte data
  - append-only on stripe boundaries
  - snapshots
  - compound operations
- but not
  - key/value data
  - rados classes (yet)
  - object overwrites
  - non-stripe aligned appends

TIERED STORAGE



WHAT (LIB)RADOS DOESN'T DO

- stripe large objects for you
  - see libradosstriper
- rename objects
  - (although we do have a “copy” operation)
- multi-object transactions
  - roll your own two-phase commit or intent log
- secondary object index
  - can find objects by name only
  - can't query RADOS to find objects with some attribute
- list objects by prefix
  - can only enumerate in hash(object name) order
  - with confusing results from cache tiers

PERSPECTIVE

- Swift
  - AP, last writer wins
  - large objects
  - simpler data model (whole object GET/PUT)
- GlusterFS
  - CP (usually)
  - file-based data model
- Riak
  - AP, flexible conflict resolution
  - simple key/value data model (small object)
  - secondary indexes
- Cassandra
  - AP
  - table-based data model
  - secondary indexes

CONCLUSIONS

- file systems are a poor match for scale-out apps
  - usually require ad hoc sharding
  - directory hierarchies, rename unnecessary
  - opaque byte streams require ad hoc locking
- librados
  - transparent scaling, replication or erasure coding
  - richer object data model (bytes, attrs, key/value)
  - rich API (compound operations, snapshots, watch/notify)
  - extensible via rados class plugins

THANK YOU!

Sage Weil
CEPH PRINCIPAL ARCHITECT


sage@redhat.com


@liewegas