Ceph: a large open source C++ codebase

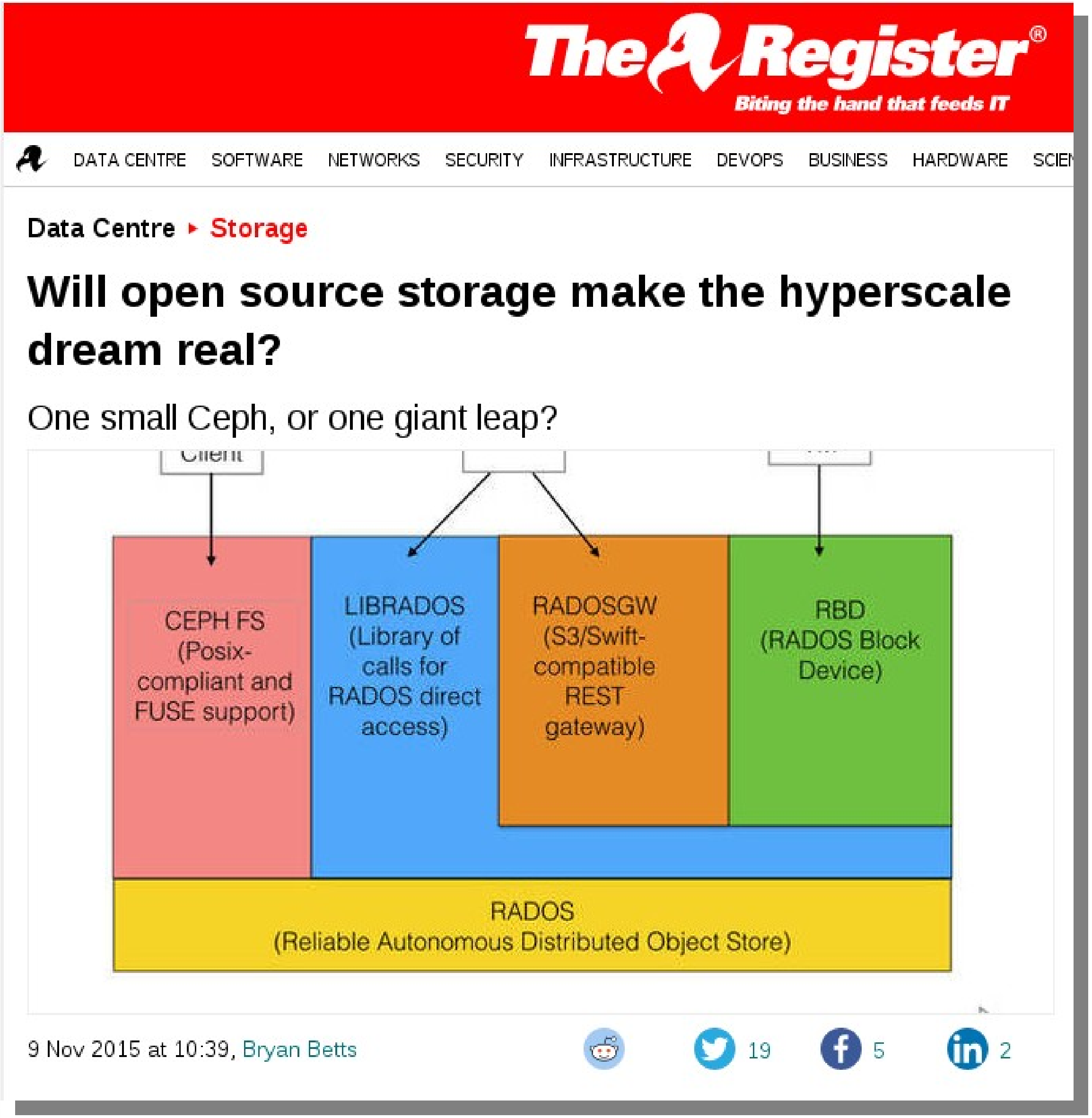
John Spray

john.spray@redhat.com
jcsp on #ceph-devel

Agenda

- Introducing Ceph & architecture
- Open source development in practice
- Technical aspects:
  - Concurrency
  - Serialization
  - Allocation
  - C++11 migration

Ceph



The Register®

Biting the hand that feeds IT

DATA CENTRE SOFTWARE NETWORKS SECURITY INFRASTRUCTURE DEVOPS BUSINESS HARDWARE SCIE

Data Centre ▸ Storage

Will open source storage make the hyperscale dream real?

One small Ceph, or one giant leap?

Client
CEPH FS (Posix-compliant and FUSE support)
LIBRADOS (Library of calls for RADOS direct access)
RADOSGW (S3/Swift-compatible REST gateway)
RBD (RADOS Block Device)
RADOS (Reliable Autonomous Distributed Object Store)

9 Nov 2015 at 10:39, Bryan Betts

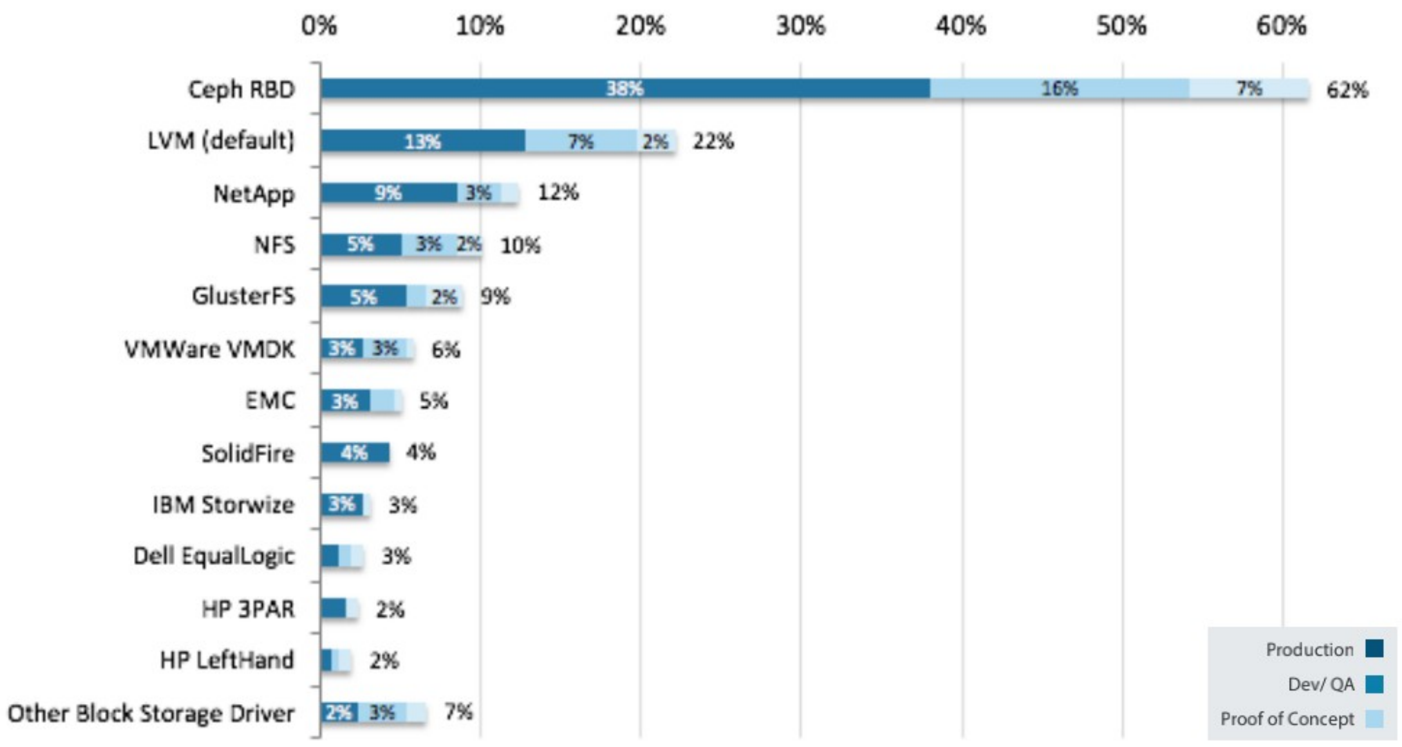
19 5 2

Source: http://www.theregister.co.uk/2015/11/09/open_source_hyperscale_storage/

Ceph

- Very high scale distributed storage system
- Underlying small object store (RADOS), with object/block/file interfaces layered on top
- Open source development, commercial support available from multiple vendors

Ceph and OpenStack



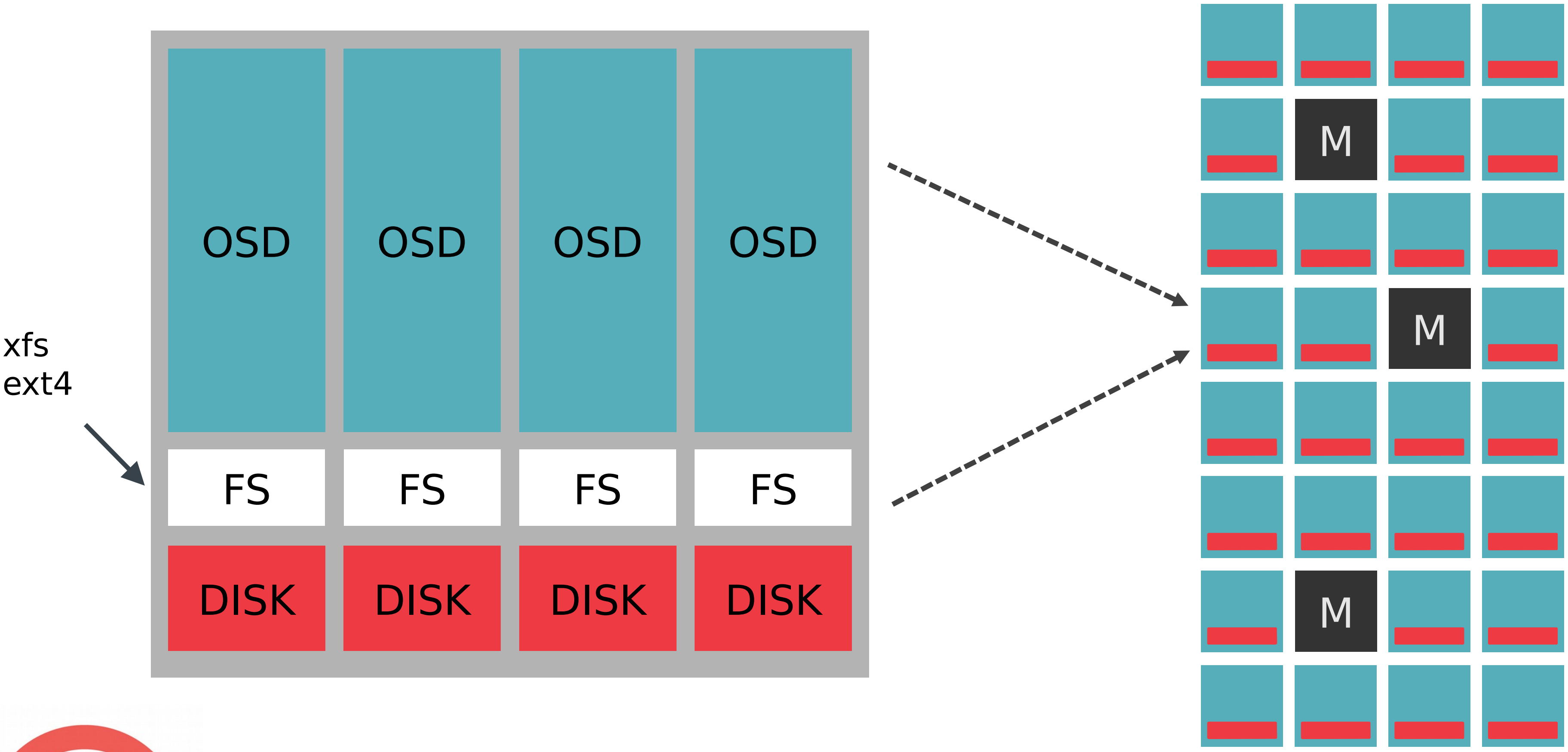
Which OpenStack Block Storage (Cinder) drivers are in use?

Figure 5.8 n=258

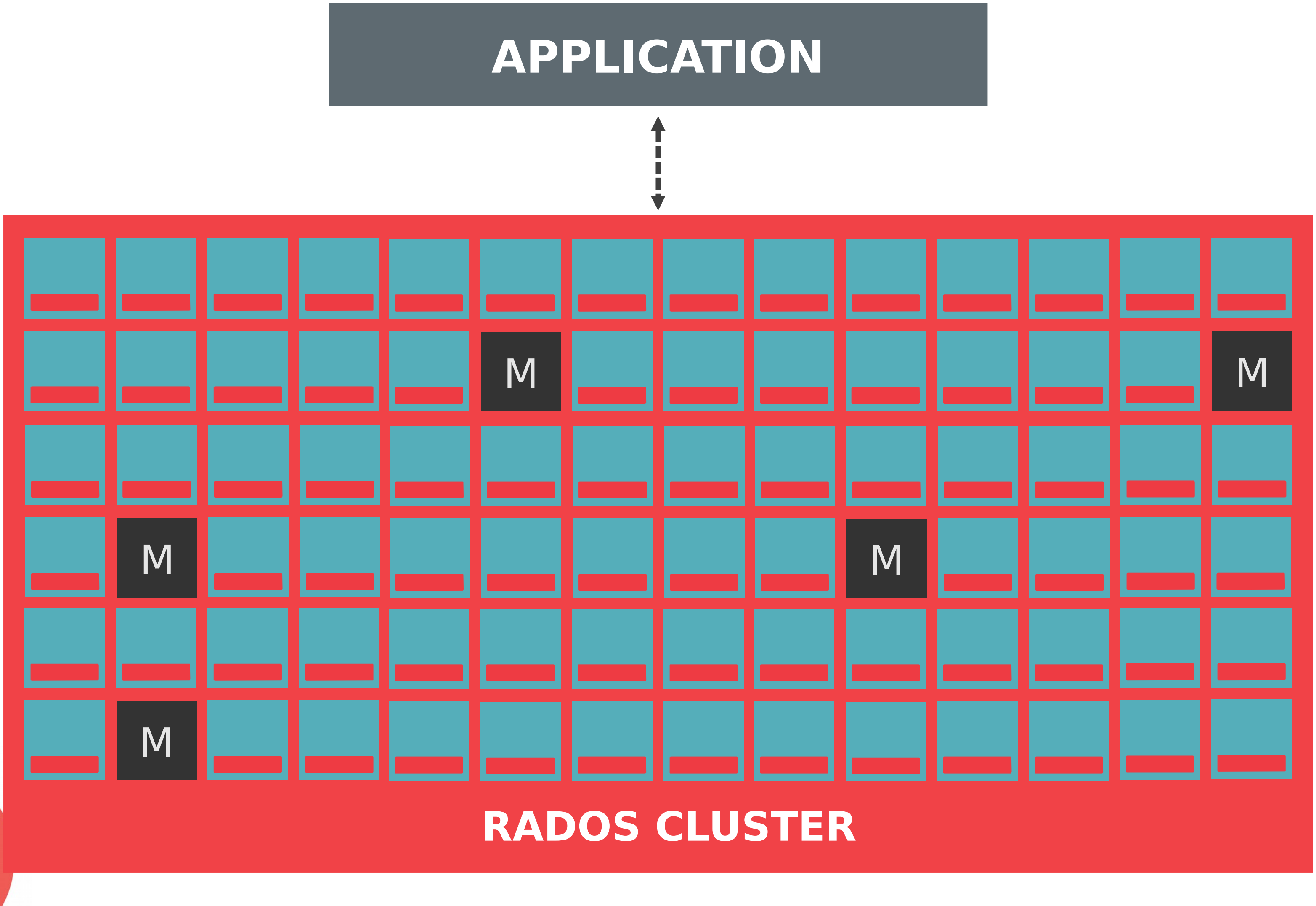
Source: http://www.openstack.org/assets/survey/Public-User-Survey-Report.pdf
OpenStack User Survey (Liberty cycle)

RADOS architecture

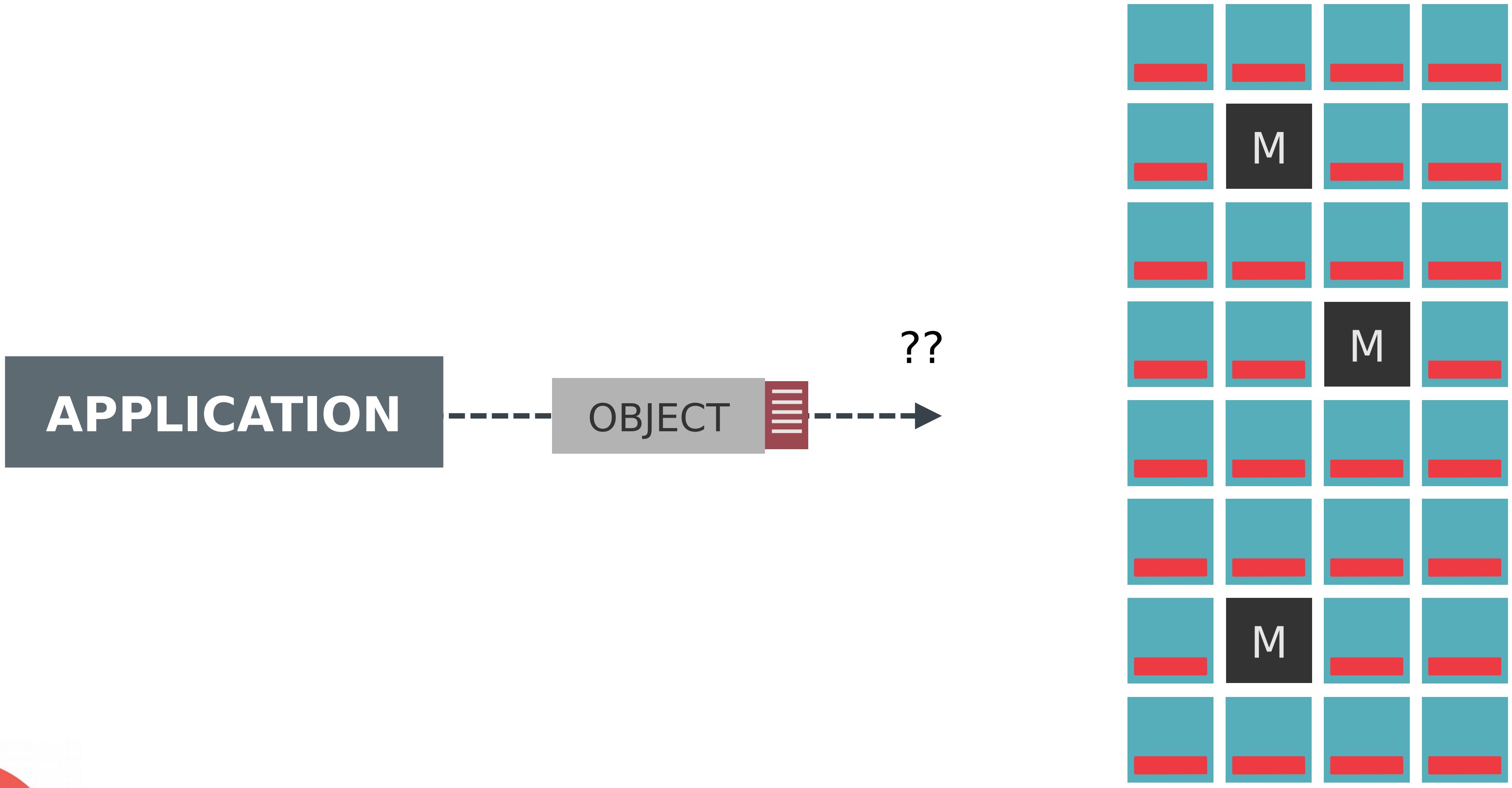
Object Storage Daemons



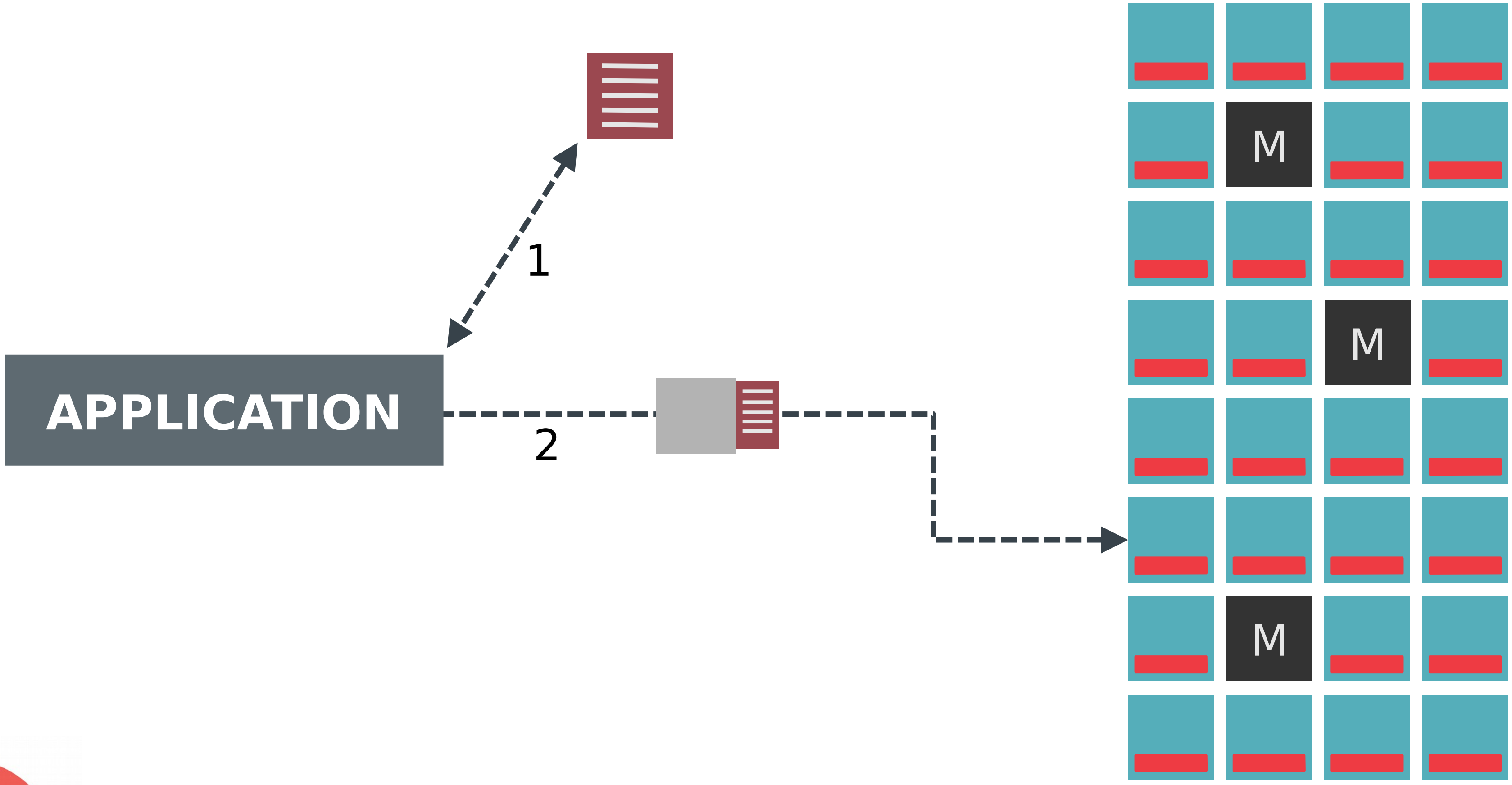
Rados Cluster



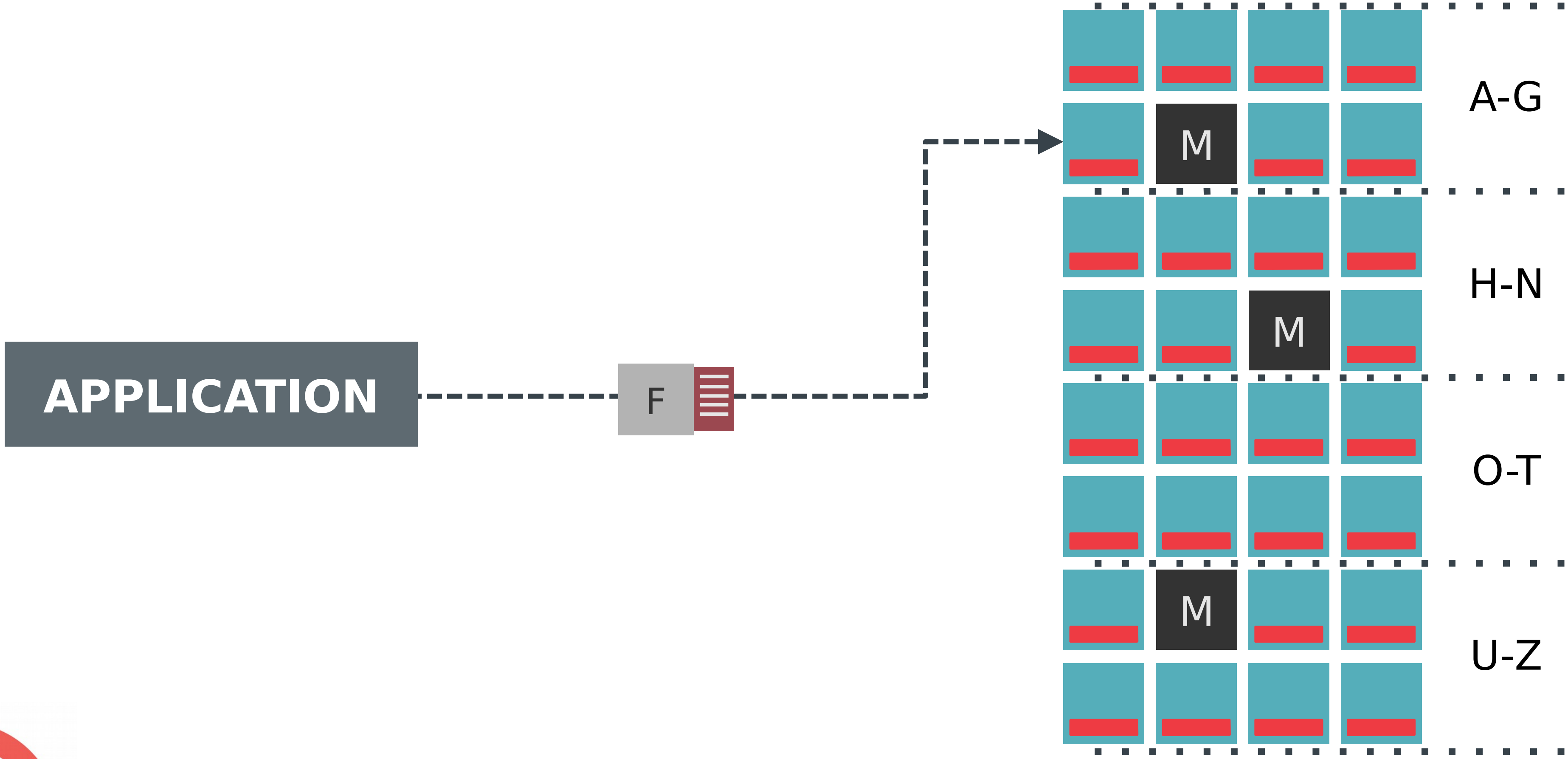
Where do objects live?



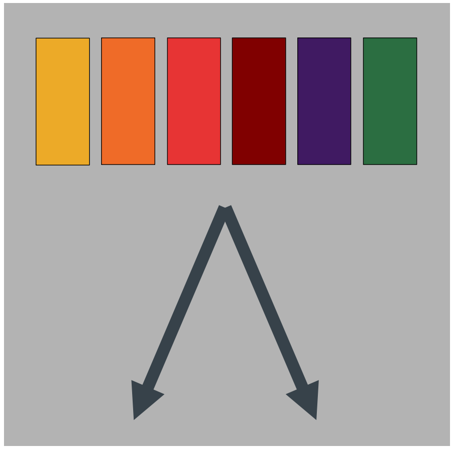
A Metadata Server?



Calculated placement



CRUSH: Dynamic data placement

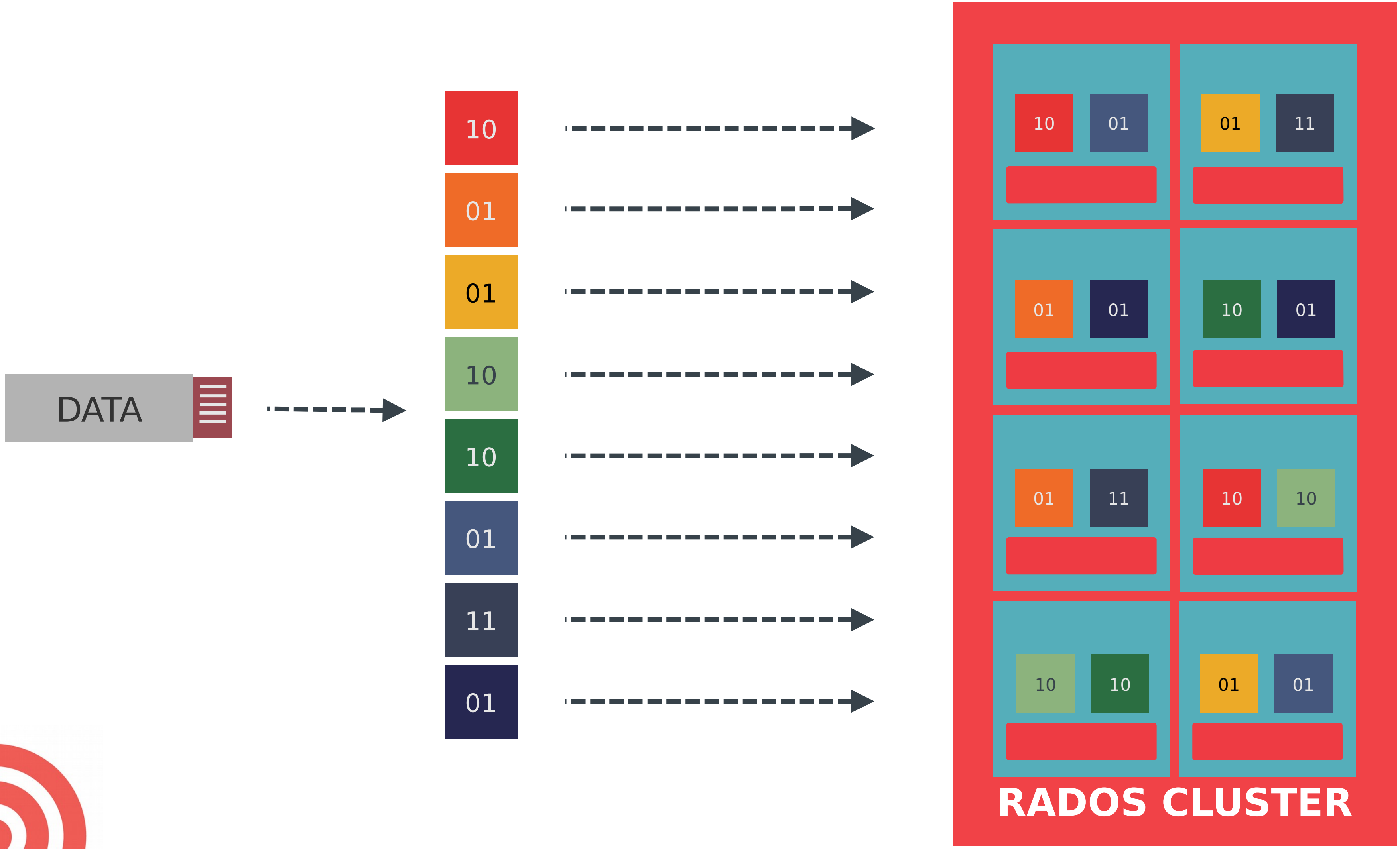


Pseudo-random placement algorithm

- 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

CRUSH: Replication



Recovering from failures

- OSDs notice when their peers stop responding, report this to monitors
- After some time, monitors mark the OSD “**out**”
- New peers selected by CRUSH, data is re-replicated across whole cluster
- Faster than RAID rebuild because we share the load
- Does not require administrator intervention

The Project

Codebase

- https://github.com/ceph/ceph
- 200k-ish C++ LOC (low estimate)
- LGPL license
- Contributions from variety of parties (software companies, hardware companies, users)
- Planning/design done via periodic online design summits (open to public)

Open source in practice

- Keep building on newest Fedora/Ubuntu: don't wait to update for dependency changes
- Use submodules where distro packages don't keep up (civetweb, rocksdb)
- Be disciplined on landing patches: keep a (fairly) stable master branch, and backport selectively to actual stable branches
- Upstream first. Communicate in the open.

Contribution workflow

- Github Pull Requests
- Commits must be small and clear
  - Requires discipline
  - Enables backporting
  - Enables answering “why?” from git history
- Gate commits on fast unit tests
- Slower tests run nightly and on hand-curated PR-testing branches

The Code

Request & Contexts

- General request flow: examine message, take some action, construct context.
- Callback objects, enqueued while waiting for e.g. I/O operations.
- Sometimes context is just “try handling this request again”, e.g. when acquiring distributed locks.
- Single threaded servers can get you a long way (scale-out more important than scale-up)

Concurrency

- Some things are (relatively) easily parallelised:
  - Issuing requests to OSDs (Objecter)
  - Reading and deserialising network IO (Messenger)
- Prioritisation is important
  - Extensive use of priority queues in OSD
  - e.g. data scrubbing vs. backfilling vs. client IO
- Re-entrancy is a problem:
  - Enqueue completions on separate “Finisher” thread
- Some things are (much) harder to parallelise:
  - Filesystem metadata: classic example is two opposing mvs between two directories

Allocation

- Allocator performance matters!
- JEMalloc, TCMalloc
- Allocator performance sensitive to threading
- Historically CPU performance relatively unimportant compared with disk latency, but all that changes with NVRAM and fast SSDs.

Serialization

- Simple homebrew serialization scheme defined for basic types, STL containers, and derived types as needed
- Versioned, reasonably fast, integrates with same “bufferlist” structure used throughout code, easy interop with kernel C code
- Unfortunately makes it hard to handle serialized structures from non-C++ code

Other housekeeping

- Homebrew code for:
  - Logging
  - Configuration
  - Performance counters
  - Admin commands
- Not as bad as it sounds: small, easy to learn interfaces, no 3rd party deps. Little ongoing maintenance.

CMake migration

- Autotools is painful
  - Arcane syntax(es)
  - Slow invokation
  - Gratuitous rebuilds
- CMake migration relatively quick for main executables, long tail of little things for packaging etc.
- Use CMake by default for your new projects

C++11 migration

- Woohoo!
- Helpful things in new code: auto, for loops, lambdas
- std::function vs. Context (reduce allocations)
- Larger patches for standardized date types, standardized threading
- Main pain point was compiler/ABI support on LTS distros (RHEL6, Ubuntu 12.04)

Wrap up

- This was a very quick look at Ceph
- Want to learn more?
  - https://github.com/ceph/ceph
  - https://ceph.com/resources/mailing-list-irc/
  - http://docs.ceph.com/docs/master/
- Questions...