CEPH AND ROCKSDB

SAGE WEIL

HIVEDATA ROCKSDB MEETUP - 2016.02.03

OUTLINE

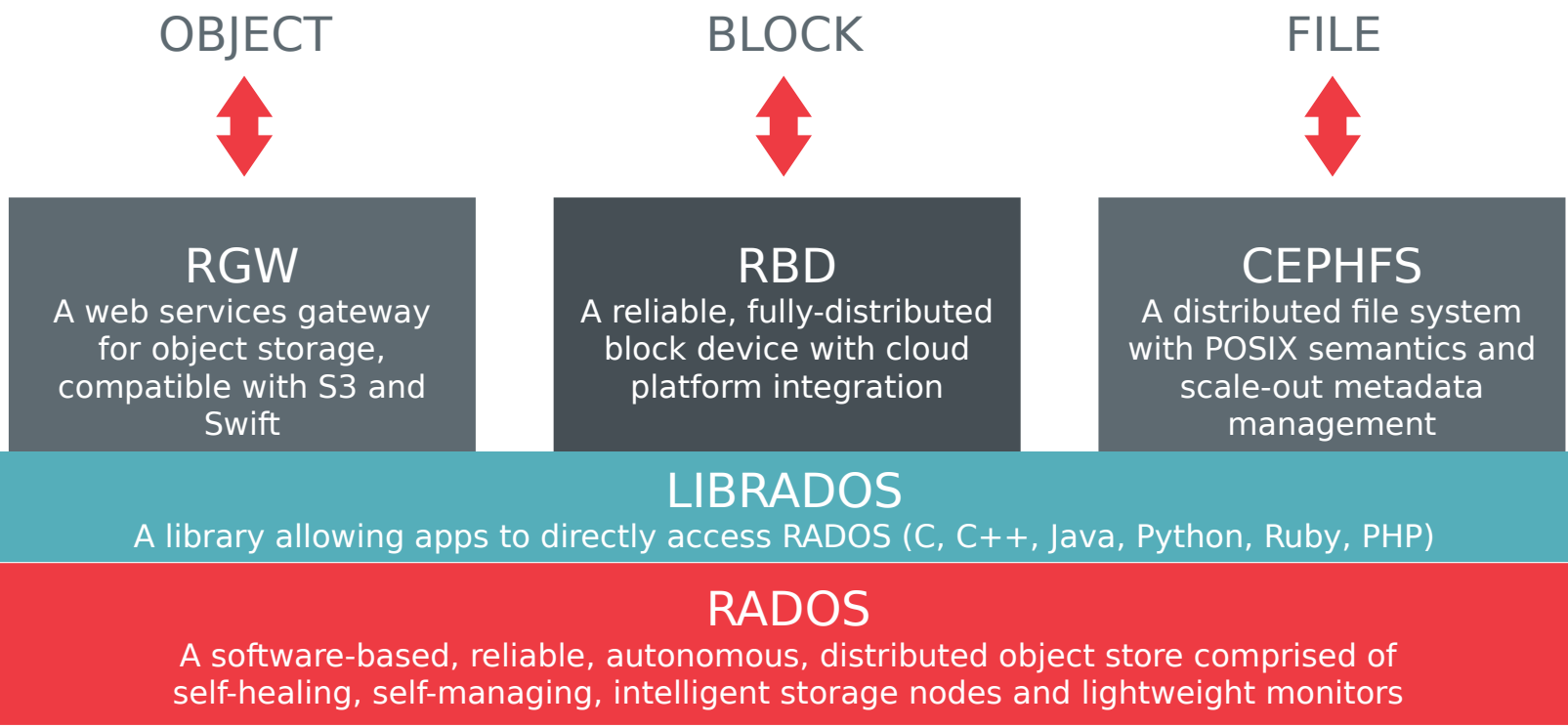
- Ceph background
- FileStore - why POSIX failed us
- BlueStore – a new Ceph OSD backend
- RocksDB changes
  - journal recycling
  - BlueRocksEnv
  - EnvMirror
  - delayed merge?
- Summary

CEPH

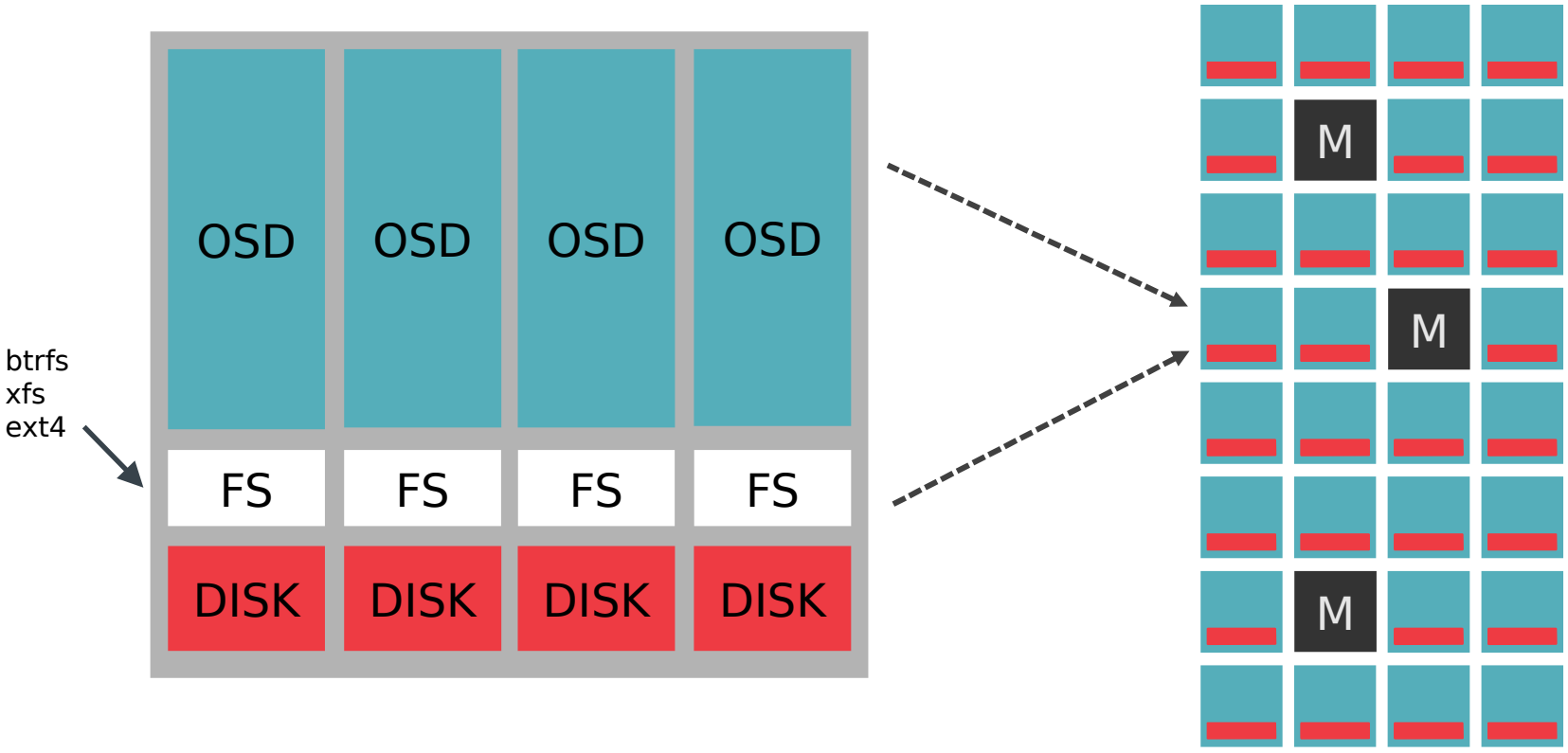
- Object, block, and file storage in a single cluster
- All components scale horizontally
- No single point of failure
- Hardware agnostic, commodity hardware
- Self-manage whenever possible
- Open source (LGPL)

- Move beyond legacy approaches
  - client/cluster instead of client/server
  - avoid ad hoc approaches HA

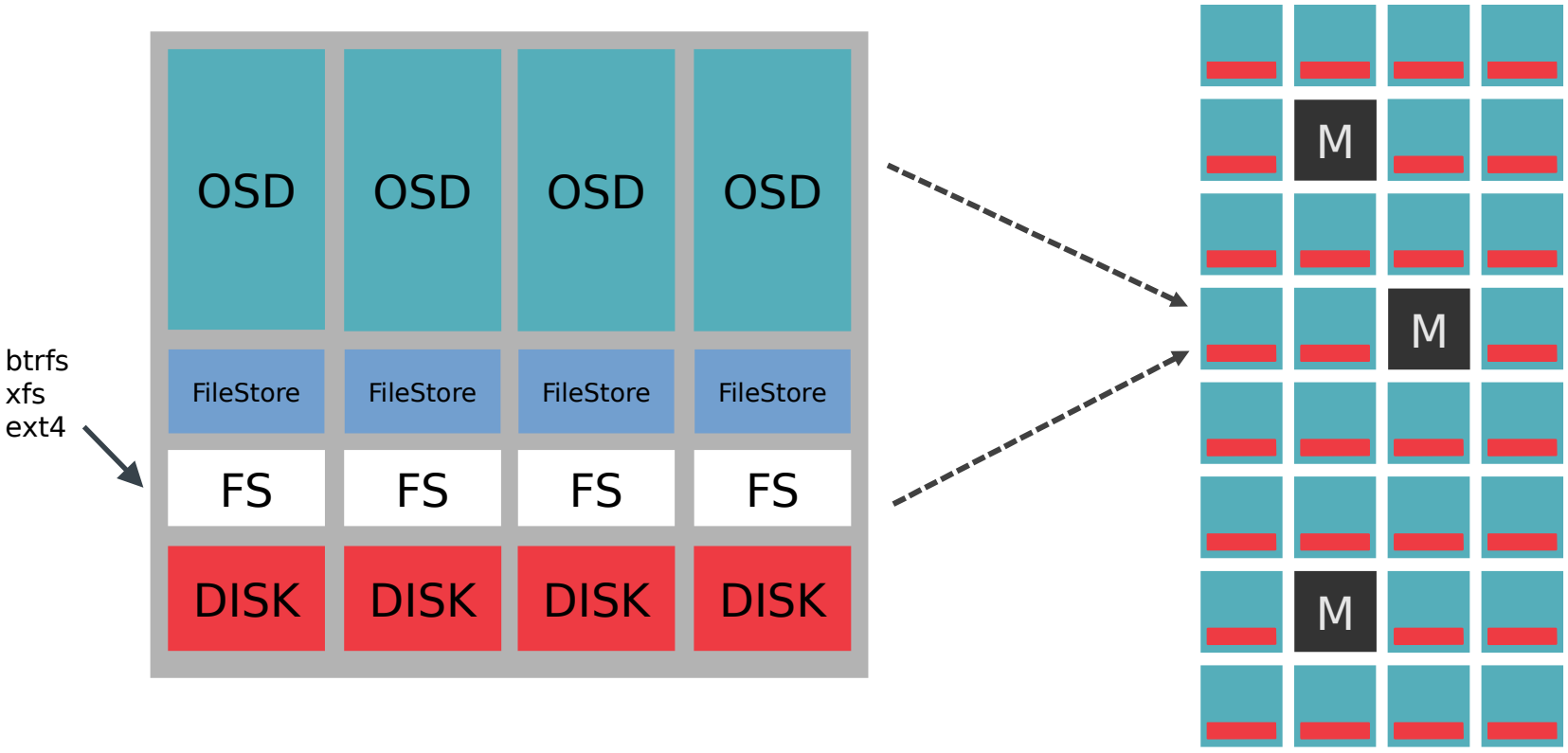
CEPH COMPONENTS



OBJECT STORAGE DAEMONS (OSDS)



OBJECT STORAGE DAEMONS (OSDS)



POSIX FAILS: TRANSACTIONS

- OSD carefully manages consistency of its data
- All writes are transactions (we need A+D; OSD provides C+I)
- Most are simple
  - write some bytes to object (file)
  - update object attribute (file xattr)
  - append to update log (leveldb insert)

  ...but others are arbitrarily large/complex

- Btrfs transaction hooks failed for various reasons
- But write-ahead journals work okay
  - write entire serialized transactions to well-optimized FileJournal
  - then apply it to the file system
  - half our disk throughput

POSIX FAILS: ENUMERATION

- Ceph objects are distributed by a 32-bit hash
- Enumeration is in hash order
  - scrubbing
  - “backfill” (data rebalancing, recovery)
  - enumeration via librados client API
- POSIX readdir is not well-ordered
- Need O(1) “split” for a given shard/range

- Build directory tree by hash-value prefix
  - split any directory when size > ~100 files
  - merge when size < ~20 files
  - read entire directory, sort in-memory

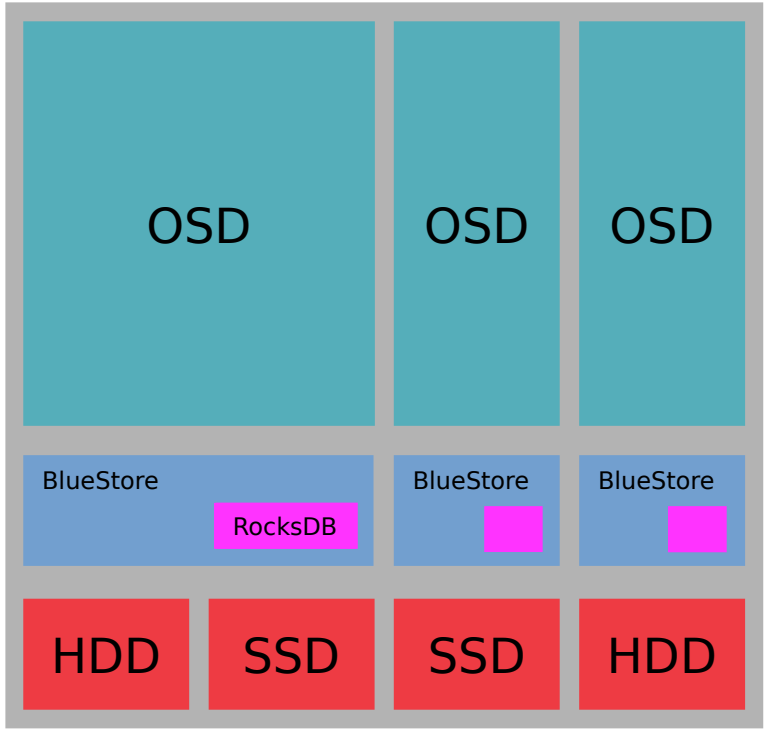
```
...
A/A03224D3_qwer
A/A247233E_zxcv
...
B/8/B823032D_foo
B/8/B8474342_bar
B/9/B924273B_baz
B/A/BA4328D2_asdf
...
```

WE WANT TO AVOID POSIX FILE INTERFACE

- POSIX has the wrong metadata model for us
  - rocksdb perfect for managing our namespace
- NewStore = rocksdb + object files

- Layering over POSIX duplicates consistency overhead
  - XFS/ext4 journal writes for fs consistency
  - rocksdb wal writes for our metadata

- BlueStore = N**ewStore** over **bl**ock

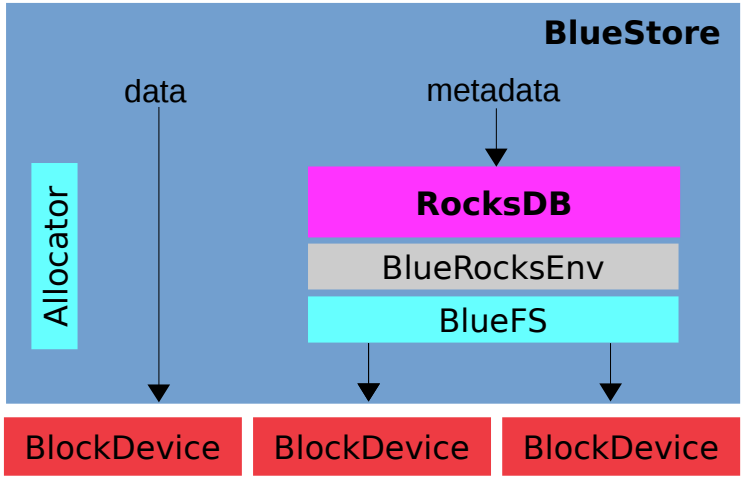


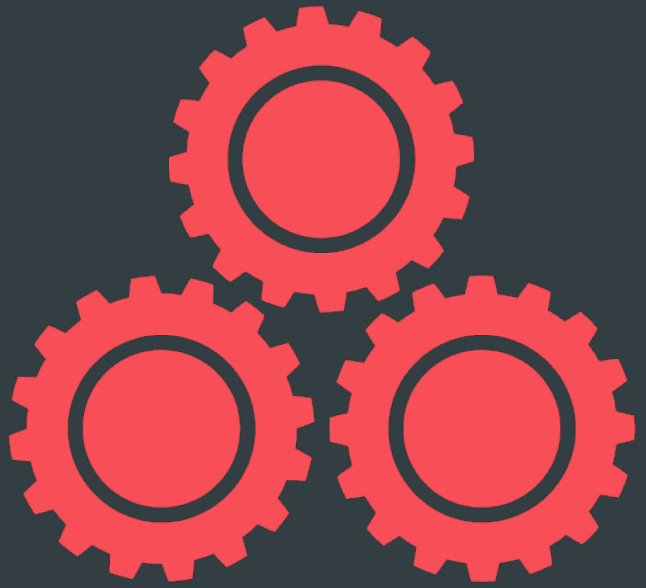
WHY ROCKSDB?

- Ideal key/value interface
  - transactions
  - ordered enumeration
  - fast commits to log/journal
- Common interface
  - can always swap in another KV DB if we want
- Abstract storage backend (rocksdb::Env)
- C++ interface
- Strong and active open source community

BLUESTORE DESIGN

- rocksdb
  - object metadata (onode) in rocksdb
  - write-ahead log (small writes/overwrites)
  - ceph key/value “omap” data
  - allocator metadata (free extent list)
- block device
  - object data
- pluggable allocator
- rocksdb shares block device(s)
  - BlueRocksEnv is rocksdb::Env
  - BlueFS is super-simple C++ “file system”
- 2x faster on HDD, more on SSD





ROCKSDB

ROCKSDB: JOURNAL RECYCLING

- Problem: 1 small (4 KB) Ceph write → 3-4 disk IOs!
  - BlueStore: write **4 KB of user data**
  - rocksdb: append record to WAL
    - **write update block** at end of log file
    - fsync: XFS/ext4/BlueFS journals **inode size/alloc update** to its journal
- fallocate(2) doesn't help
  - data blocks are not pre-zeroed; fsync still has to update alloc metadata

- rocksdb LogReader only understands two modes
  - read until end of file (need accurate file size)
  - read all valid records, then ignore zeros at end (need zeroed tail)

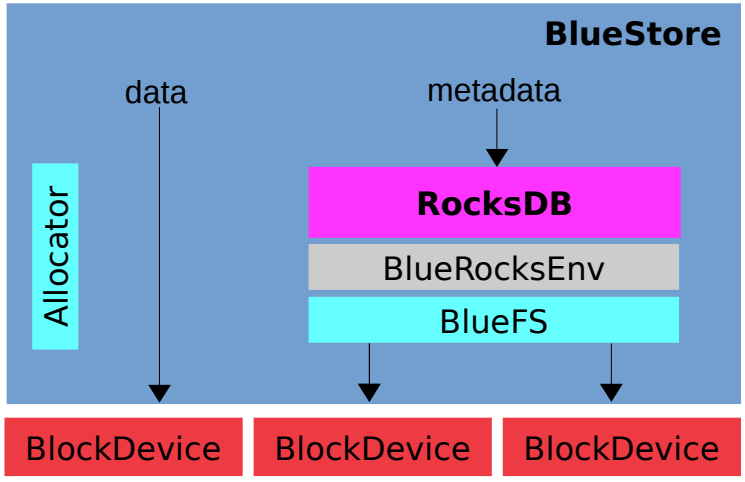
ROCKSDB: JOURNAL RECYCLING (2)

- Put old log files on recycle list (instead of deleting them)
- LogWriter
  - overwrite old log data with new log data
  - include log number in each record
- LogReader
  - stop replaying when we get garbage (bad CRC)
  - or when we get a valid CRC but record is from a previous log incarnation
- Now we get one log append → one IO!

- Upstream, but missing a bug fix (PR #881)

ROCKSDB: BLUEROCKSENV + BLUEFS

- class BlueRocksEnv : public rocksdb::EnvWrapper
  - passes file IO operations to BlueFS
- BlueFS is a super-simple “file system”
  - all metadata loaded in RAM on start/mount
  - no need to store block free list; calculate it on startup
  - coarse allocation unit (1 MB blocks)
  - all metadata updates written to a journal
  - journal rewritten/compacted when it gets large



- Map “directories” (db/, db.wal/, db.bulk/) to different block devices
  - WAL on NVRAM, NVMe, SSD
  - level0 and hot SSTs on SSD
  - cold SSTs on HDD
- BlueStore periodically balances free space between itself and BlueFS

ROCKSDB: ENVMIRROR

- include/rocksdb/utilities/env_mirror.h
- class EnvMirror : public rocksdb::EnvWrapper {

  EnvMirror(Env* a, Env* b)
- mirrors all writes to both **a** and **b**
- sends all reads to both **a** and **b**
  - verifies the results are identical

- Invaluable when debugging BlueRocksEnv
  - validate BlueRocksEnv vs rocksdb's default PosixEnv

ROCKSDB: DELAYED LOG MERGE

- We write lots of short-lived records to log
  - insert wal_1 = 4 KB
  - insert wal_2 = 8 KB
  - ...
  - insert wal_10 = 4 KB
  - delete wal_1
  - insert wal_11 = 4 KB
- Goal
  - prevent short-lived records from ever getting amplified
  - keep, say, 2N logs
  - merge oldest N to new level0 SST, but **also** remove keys updated/deleted in newest N logs

SUMMARY

- Ceph is great
- POSIX was poor choice for storing objects
- Our new BlueStore backend is awesome
- RocksDB rocks and was easy to embed
- Log recycling speeds up commits (now upstream)
- Delayed merge will help too (coming soon)

THANK YOU!

Sage Weil
CEPH PRINCIPAL ARCHITECT


sage@redhat.com


@liewegas