Ceph – High performance without High Costs

The Ceph backend you've always wanted

Allen Samuels

Engineering Fellow, Western Digital

Forward-Looking Statements

During our meeting today we may make forward-looking statements.

Any statement that refers to expectations, projections or other characterizations of future events or circumstances is a forward-looking statement, including those relating to market position, market growth, product sales, industry trends, supply chain, future memory technology, production capacity, production costs, technology transitions and future products. This presentation contains information from third parties, which reflect their projections as of the
date of issuance.

Actual results may differ materially from those expressed in these forward-looking statements due to factors
detailed under the caption “Risk Factors” and elsewhere in the documents we file from time to time with the SEC, including our annual and quarterly reports.

We undertake no obligation to update these forward-looking statements, which speak only as of the date hereof.

Blah blah blah High Performance blah blah blah low cost

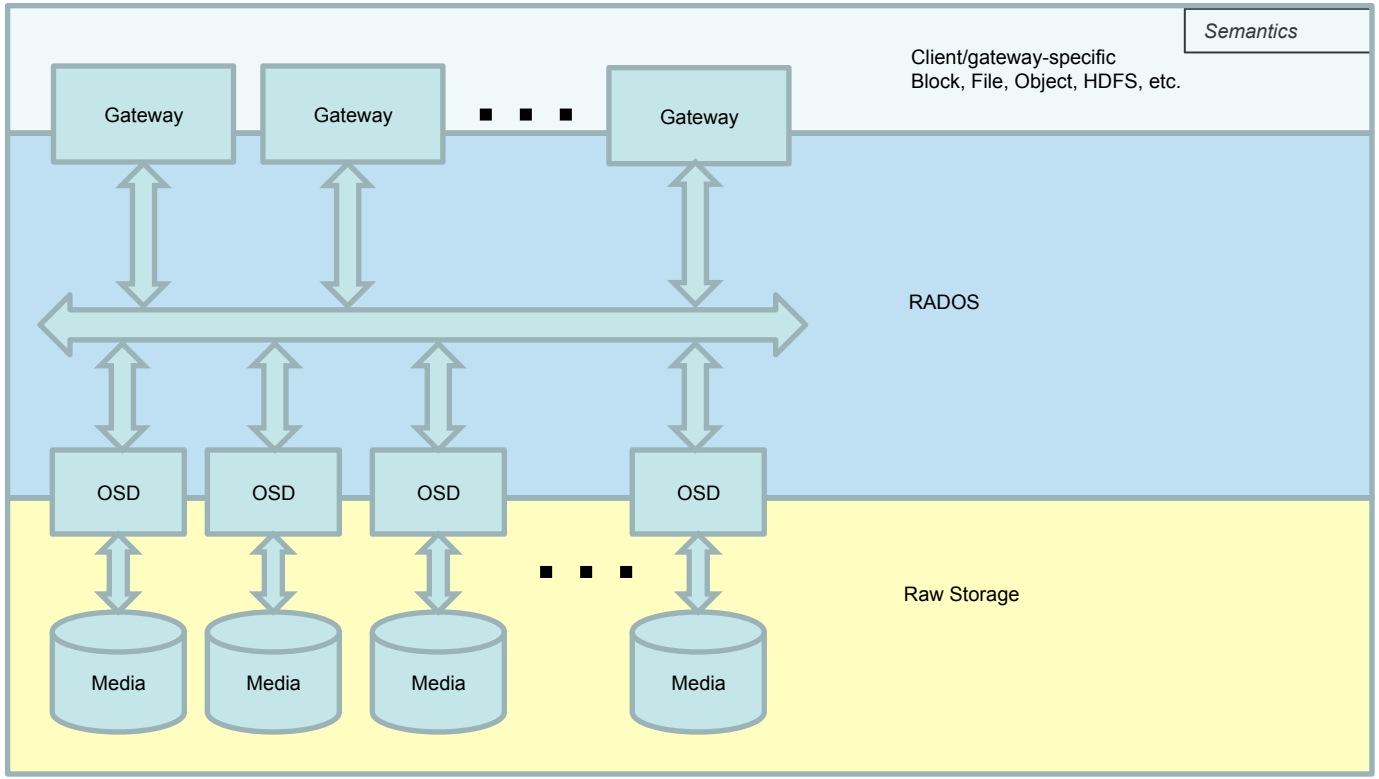
- For scale-out, it's really price/performance
  - Efficiently use resources
  - Be mindful of resource tradeoffs
- These are architectural issues
  - Focus on the algorithms

Optimizing Ceph for the future

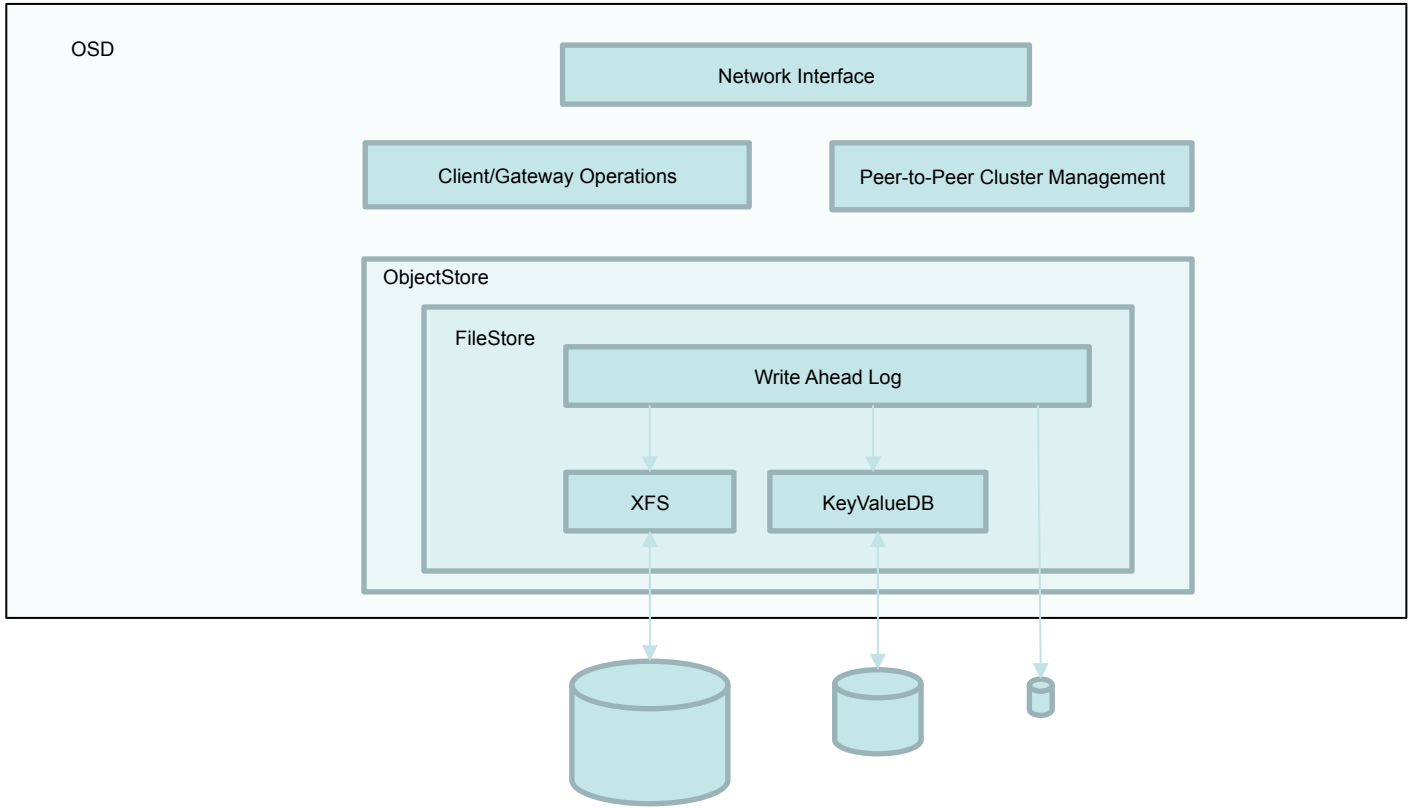
- With the vision of an all flash system, SanDisk engaged with the Ceph community in 2013
- Limit to no wire or storage format changes
- Result: Jewel release is up to 15x vs. Dumpling
  - Read IOPS are decent, Write IOPS still suffering
- Further improvements require breaking storage format compatibility

Flash Memory SUMMIT

Conceptual Ceph System Model



Inside the OSD



What's wrong with FileStore?

- Metadata split into two disjoint environments
  - Ugly logging required to meet transactional semantics
- Posix directories are poor indexes for objects
- Missing virtual copy and merge semantics
  - Virtual copies become actual copies
- BTRFS hope didn't pan out
  - Snapshot/rollback overhead too expensive for frequent use
  - Transaction semantics aren't crash proof

What's wrong with FileStore?

- Bad Write amplification
  - Write ahead logging for everything
  - levelDB (LSM)
  - Journal on Journal
- Bad jitter due to unpredictable file system flushing
  - Binge/purge cycle is very difficult to ameliorate
- Bad CPU utilization
  - syncfs is VERY expensive

BlueStore a rethink of ObjectStore

- Original implementation written by Sage late in ‘15
- Tech preview available in Jewel Release
- Preserves wire compatibility
- Storage Format incompatible
- Target Write performance ≥ 2x FileStore
- Target Read performance ≥ FileStore

BlueStore a rethink of ObjectStore

- Efficiently Support current and future HW types
  - SCM, Flash, PMR and SMR hard drives, standalone or hybrid combinations
- Improve performance
  - Eliminate double write when unneeded
  - Better CPU utilization through simplified structure and tailored algorithms
- Much better code stability

BlueStore a rethink of ObjectStore

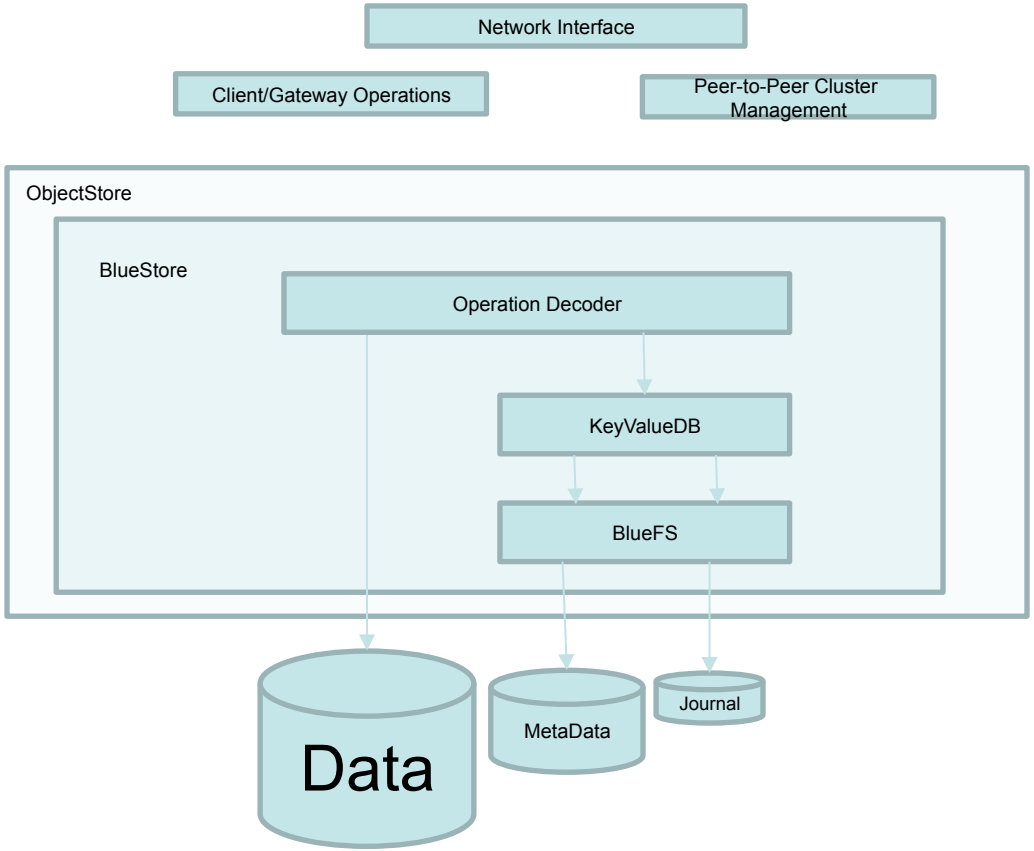
- Wire compatible but not data format compatible
  - Mixed FileStore/Bluestore nodes in a cluster transparently supported
  - FileStore continues for legacy systems
  - In place upgrade/conversion supported via rebuild

BlueStore a rethink of ObjectStore

- Optional Checksum on all read operations
- Enhanced Functionality
  - Inline Compression
    - Pluggable, Snappy and Zlib initially
  - Virtual clone
    - Efficient implementation of snapshots
  - Virtual move
    - Enables RBD/CephFS to directly use erasure coded pools

Flash Memory SUMMIT

BlueStore



BlueStore Architecture

- One, Two or Three raw block devices
  - Data, Metadata/WAL and KV Journaling
  - When combined no fixed partitioning is needed
- Use a single transactional KV store for all metadata
  - Semantics are well matched to ObjectStore transactions
- Use raw block device for data storage
  - Support Flash, PMR and SMR HDD

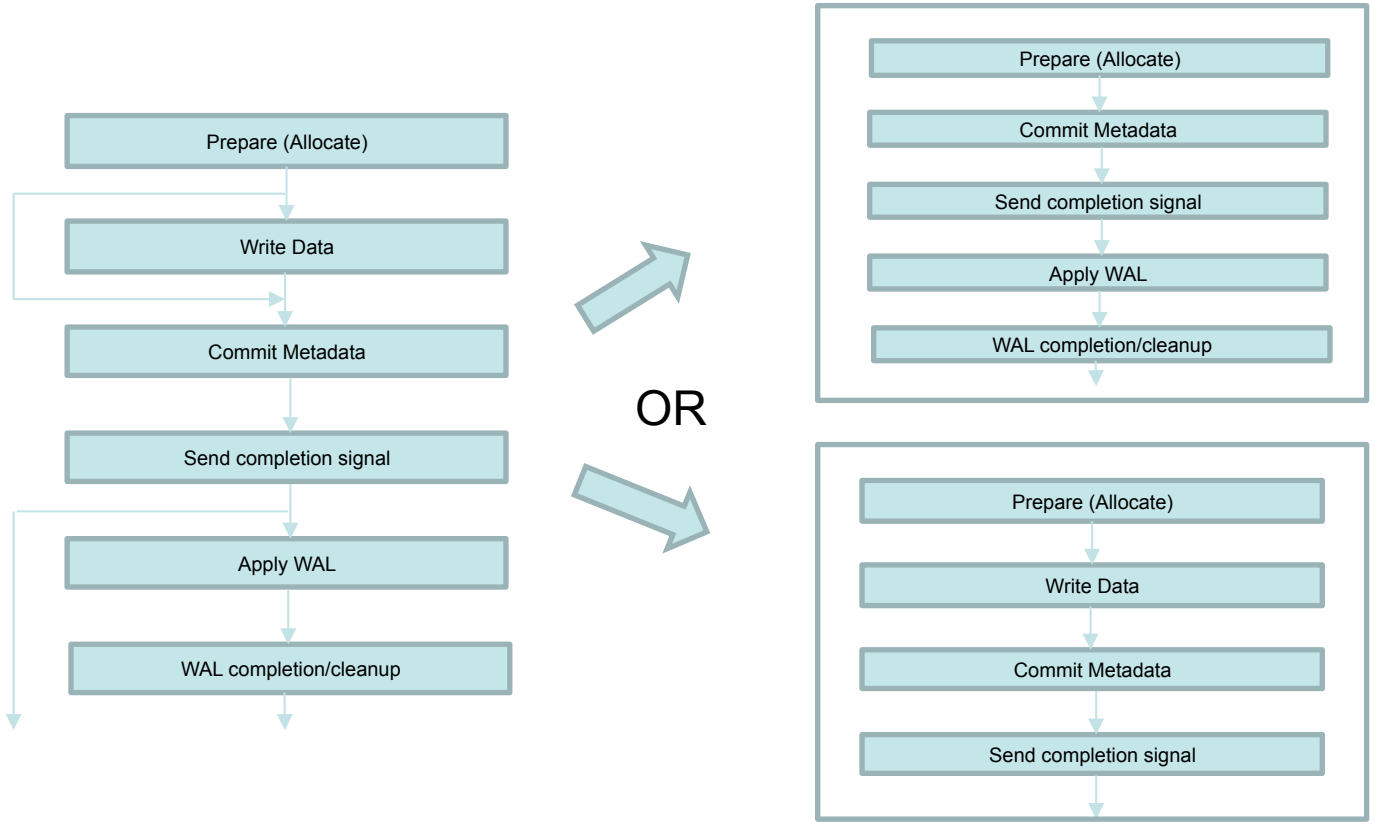
Two Write Path Options

- Direct Write
  - (1) Write data to unused space (Copy-on-write style)
    - Trivially crash-proof
  - (2) Modify metadata through single KV transaction
    - Transactions semantics of KV store shine here!
  - (3) Send completion signal

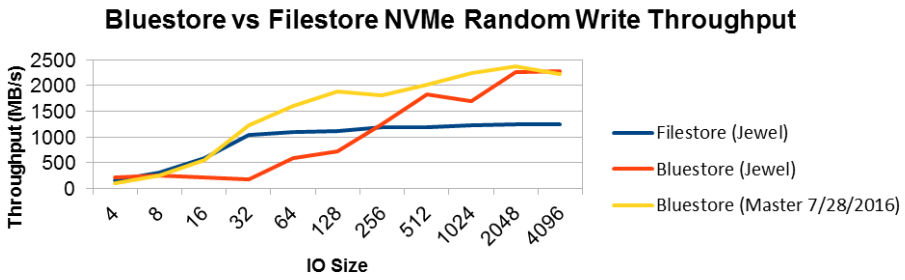
Two Write Path Options

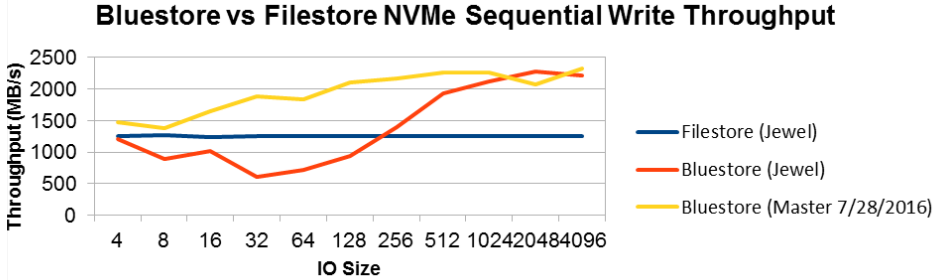
- Write-ahead Log (WAL)
  - (1) Commit data and metadata into single KV transaction
  - (2) Send completion signal
  - (3) Move data from KV into destination (Idempotent and crash restartable)
  - (4) Update KV to remove data and WAL operation

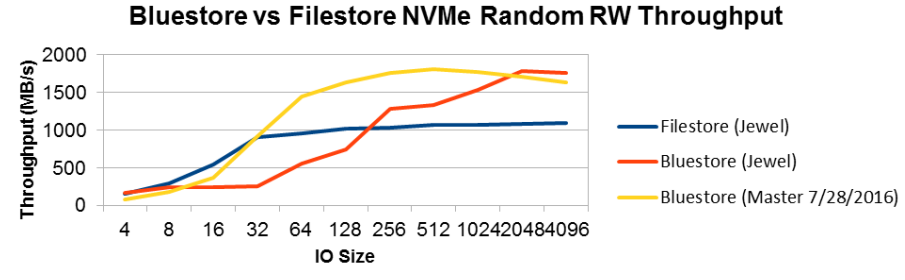
Basic Write pipeline

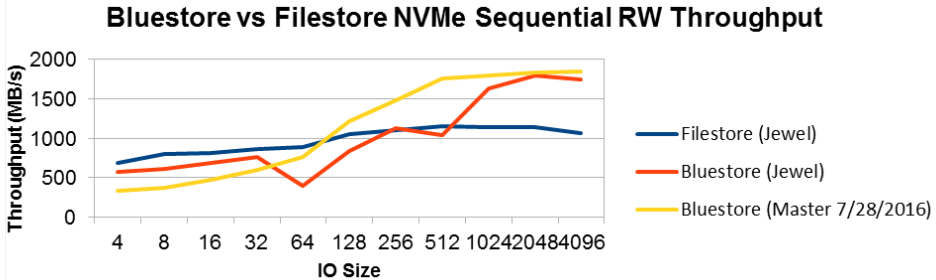


BlueStore vs FileStore







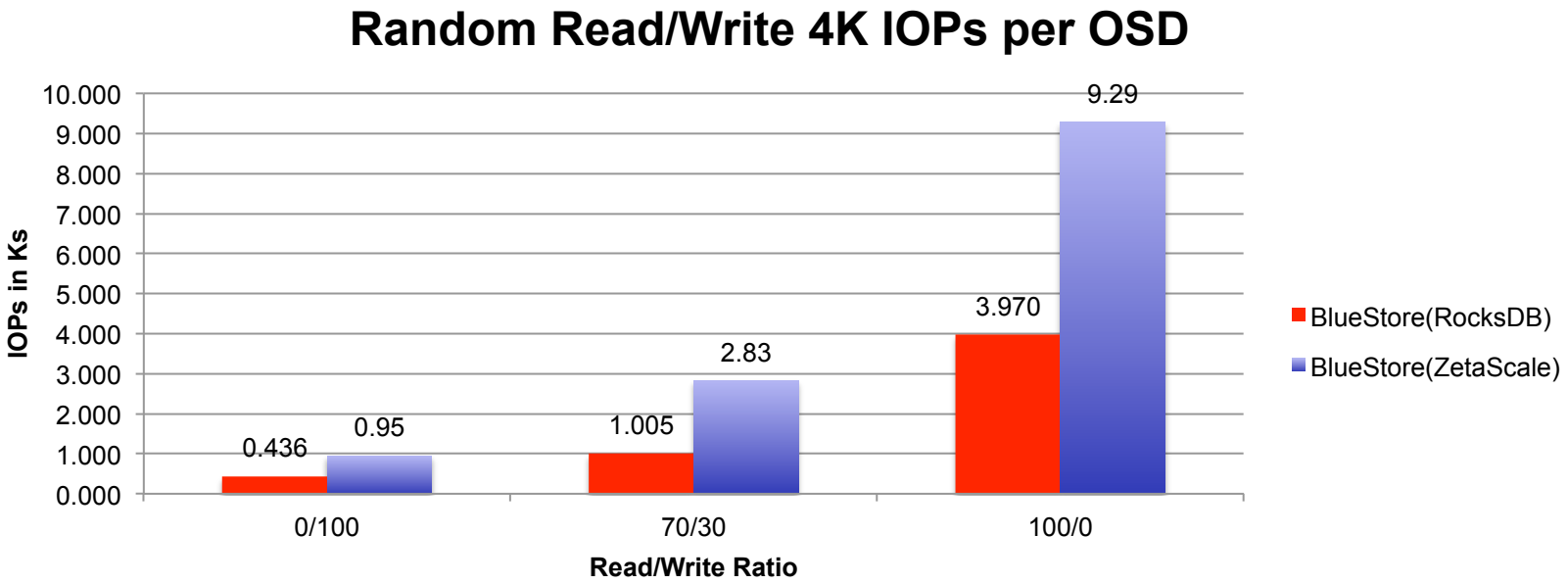


1 800GB P3700 card (4 OSDs per), 64GB ram, 2 x Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz, 1 x Intel 40GbE link
client fio processes and mon were on the same nodes as the OSDs.

KV Store Options

- RocksDB is a Facebook extension of levelDB
  - Log Structured Merge (LSM) based
  - Ideal when metadata is on HDD
  - Merge is effectively host-based GC when run on flash
- ZetaScale™ from SanDisk® now open sourced
  - B-tree based
  - Ideal when metadata is on Flash
  - Uses device-based GC for max performance

BlueStore ZetaScale v RocksDB Performance



Test Setup:
1 OSD, 8TB SAS SSD, 10GB ram, Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz , fio, 32 thds, 64 iodepth, 6TB dataset, 30 min

BlueStore Status

- Tech Preview in Jewel Release
  - Unreliable, not format compatible with master
- Significant CPU/Memory optimizations in pipeline
  - Better small block performance (CPU limited on flash)
  - SMR support in development (release date tbd)
- Target for release in Kraken, later this year
- Target as default in Luminous (next year)