dm-writeboost internal

Akira Hayakawa (@akiradeveloper)

The scope of this slides

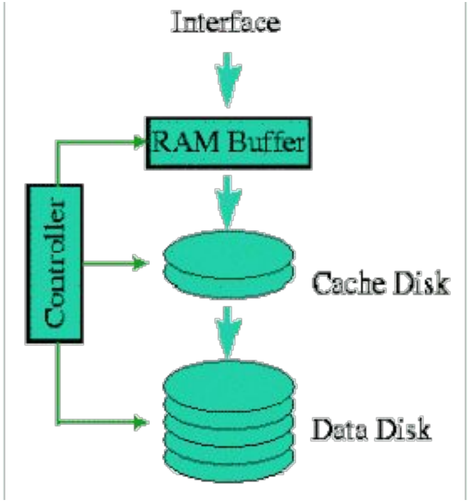
- Help those who wants to deeply understand dm-writeboost.
- Compliments doc/dm-writeboost-readme.txt
- Includes figures that helps you read the code.

Overview of dm-writeboost

- Block-level log-structured caching driver for Linux, influenced by Disk Caching Disk (DCD).
- Features
  - **Durable**: Each log contains data and metadata atomically. So never lose data on any failure.
  - **Long lifetime** of SSD cache device: We only need to write to SSD once per 127 writes.
  - **Fast**: Compared to dm-cache and bcache, random write is efficient.
  - **Portable**: Support kernel 3.10 to the latest.

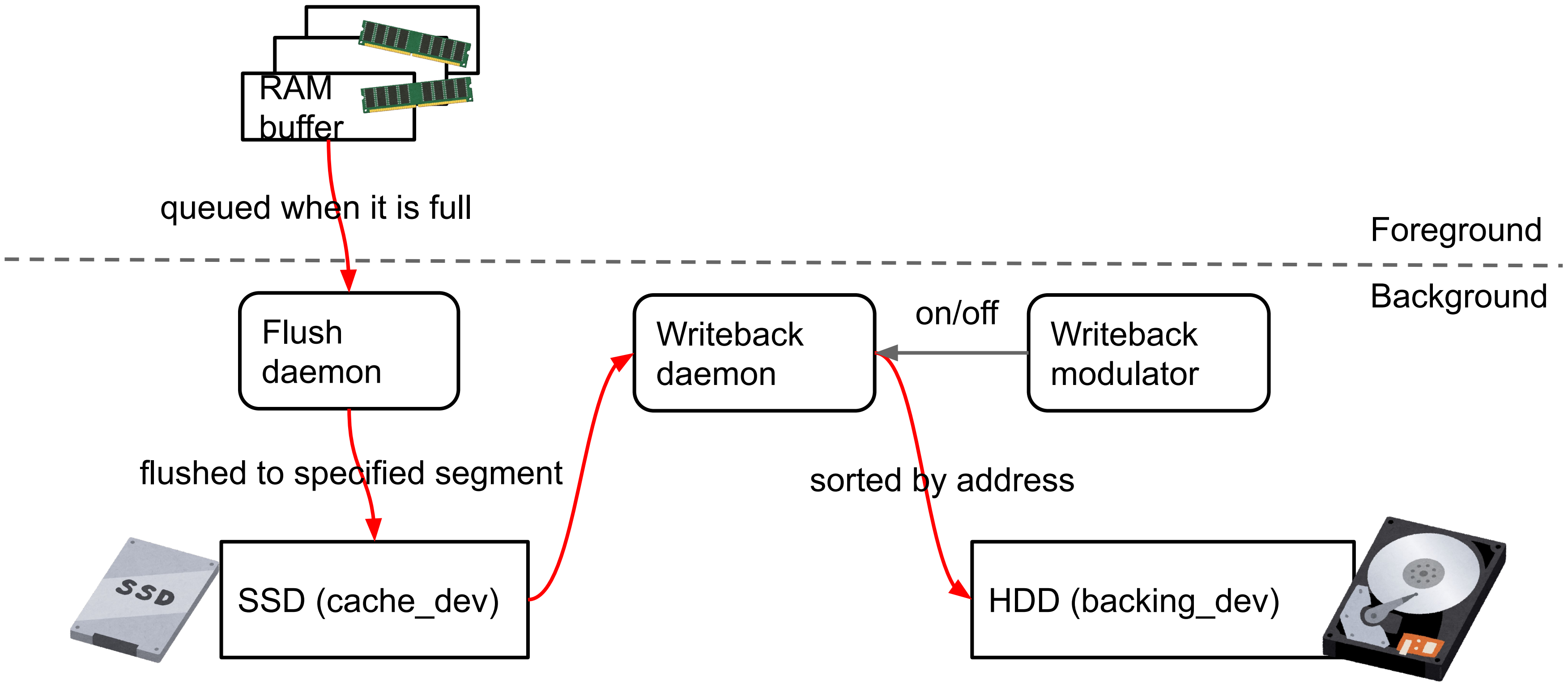
What is DCD?

A block-level log-structured caching influenced by Splite LFS

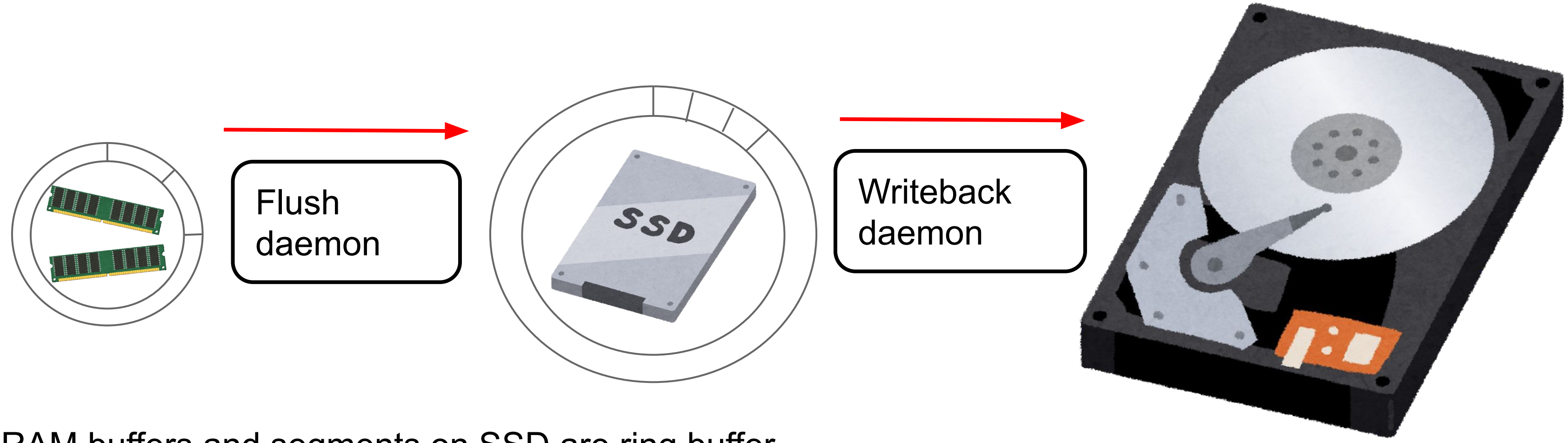


http://www.ele.uri.edu/research/hpcl/DCD/DCD.html

Architecture

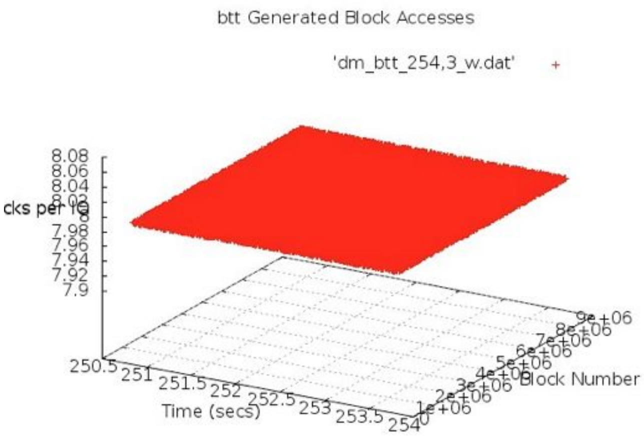


Producer-Consumer Model

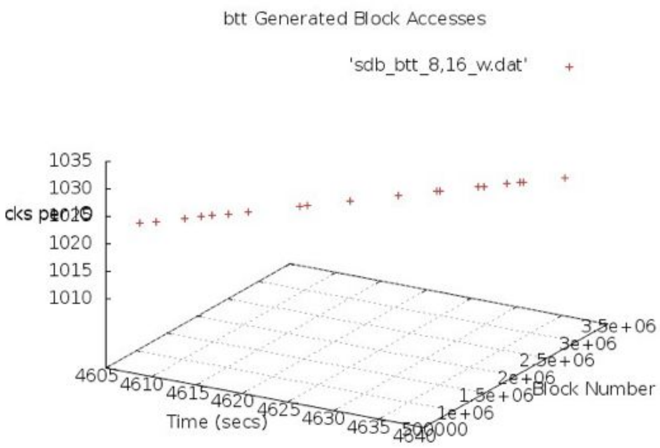


RAM buffers and segments on SSD are ring buffer, which is a good data structure for producer-consumer model.

Visualize the I/O trace



Random writes to a caching device

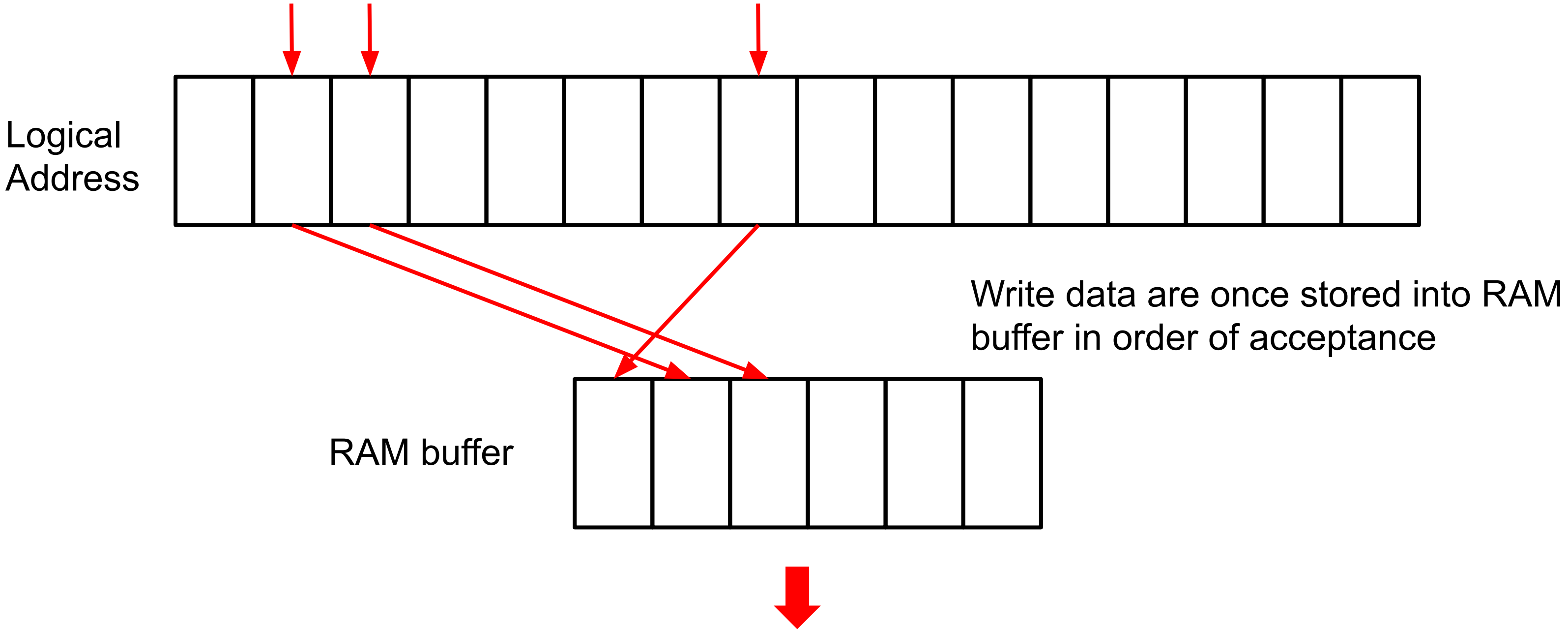


Writes to the cache device is sequential (little bit erronous but shows sequentiality)

The visualization is helped by Etsukata

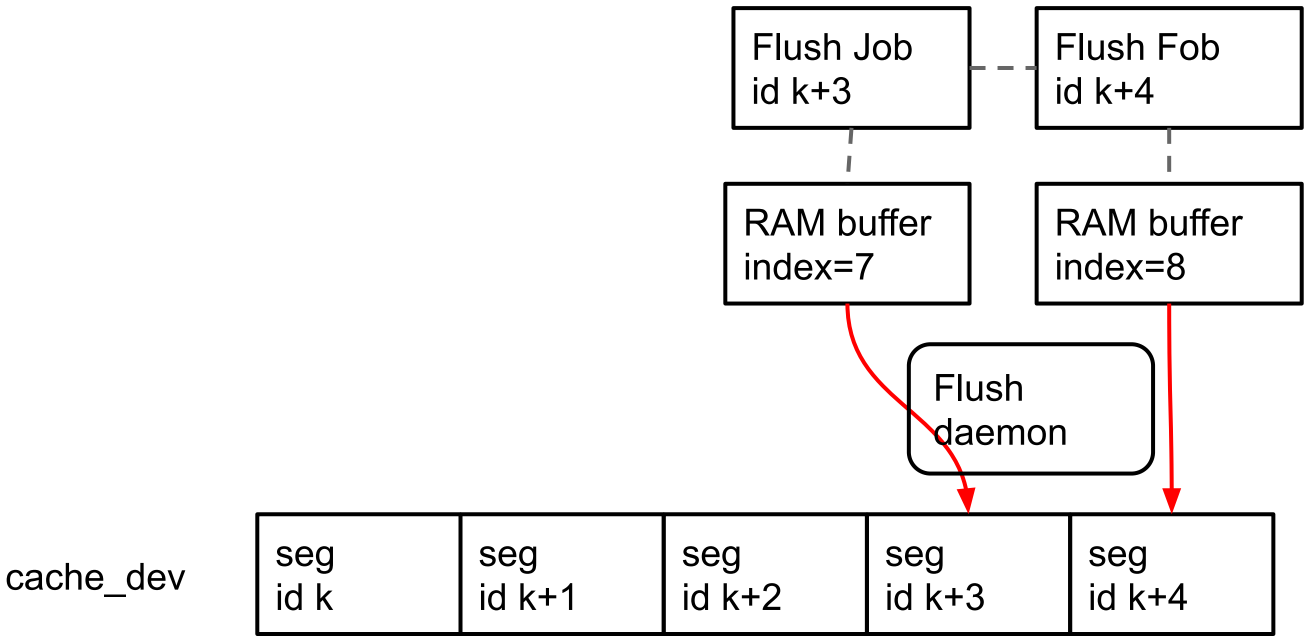
Flushing the Logs
From RAM buffer to SSD

Foreground Processing
Storing writes in RAM buffer

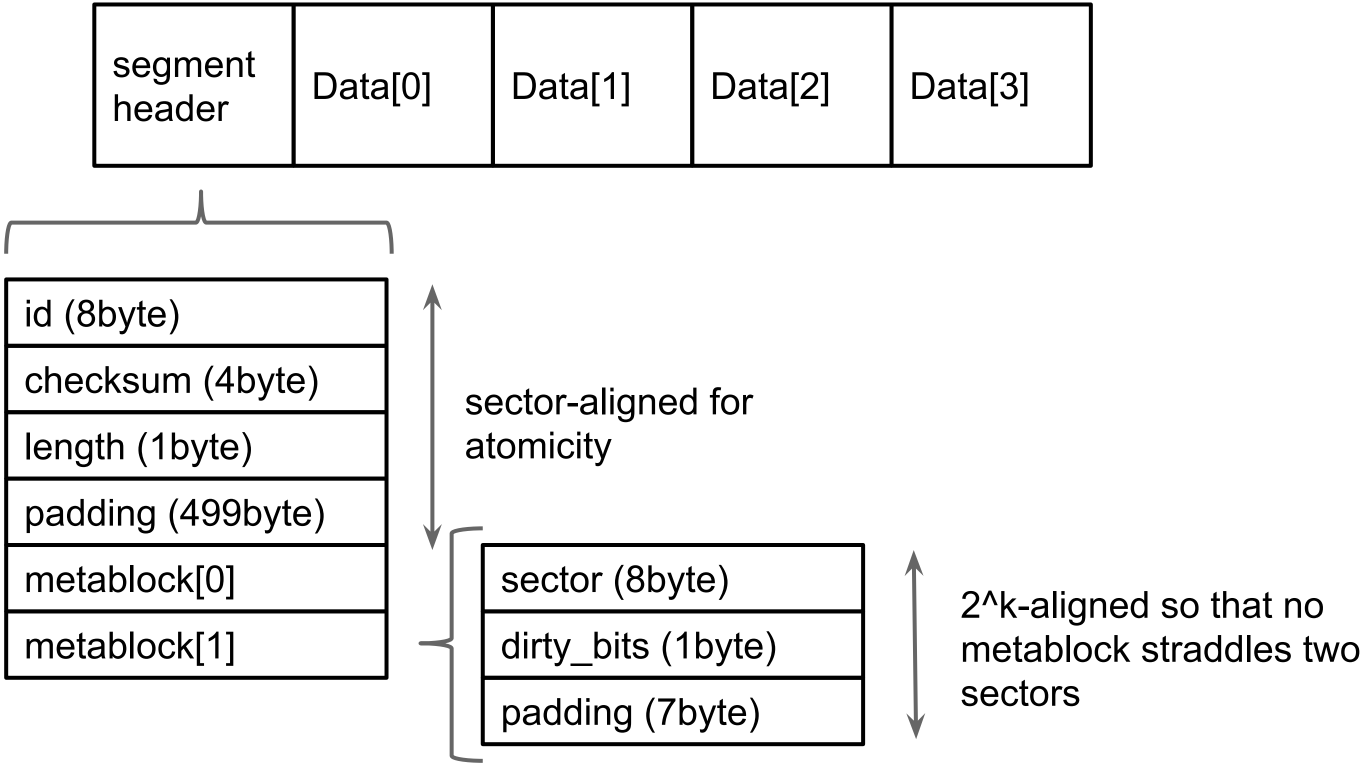


RAM buffer is queued as a flush job when it is full

Background Processing

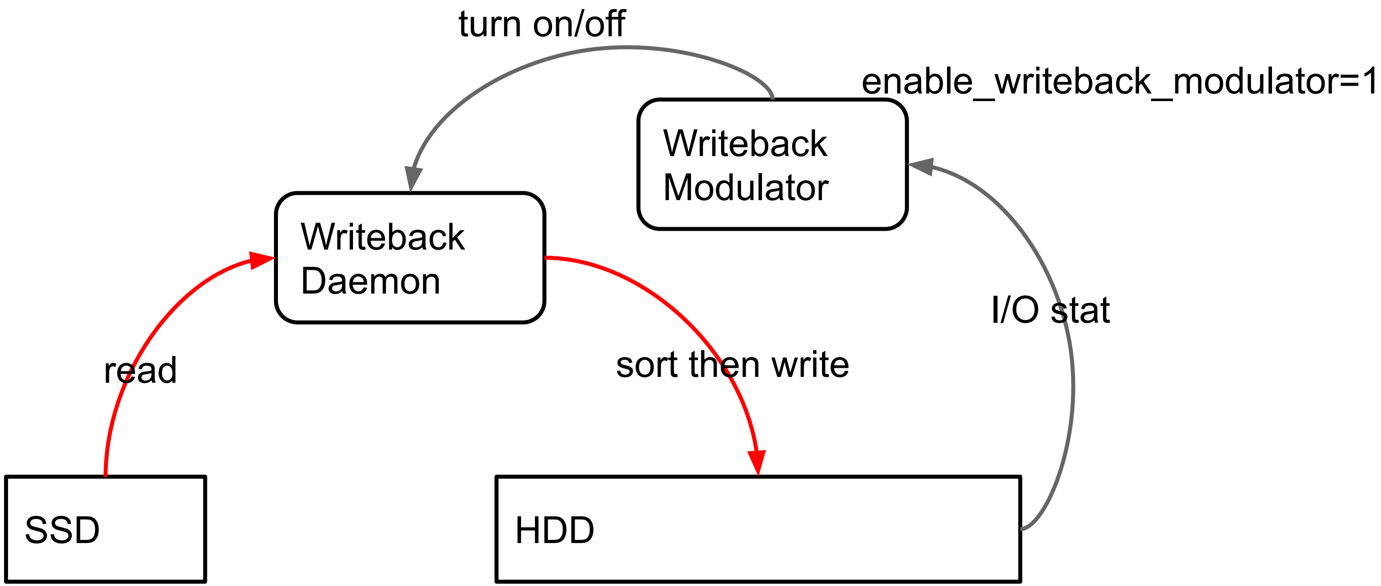


Log format
with alignment care for crash durability



Writeback
From SSD to HDD

Autonomous writeback switching



Batching and Sorting

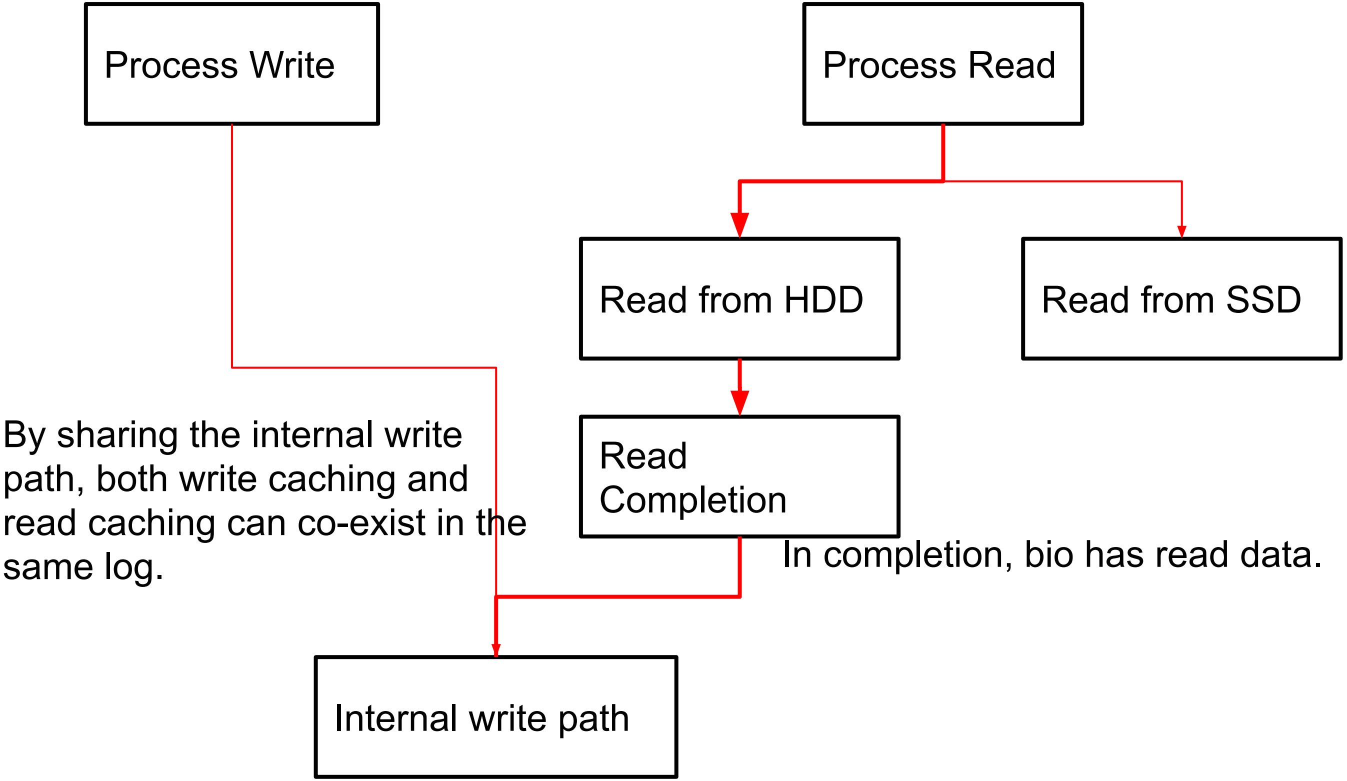
- **Batching**: Writeback daemon fetches multiple segments (tuned by nr_max_batched_writeback)
- **Sorting**: And then sorts all the cache blocks in the segments by the destination address, using rbtree. This can make use of sequential write performance of backing rotational disk.
    - We should not trust I/O scheduler

Read caching

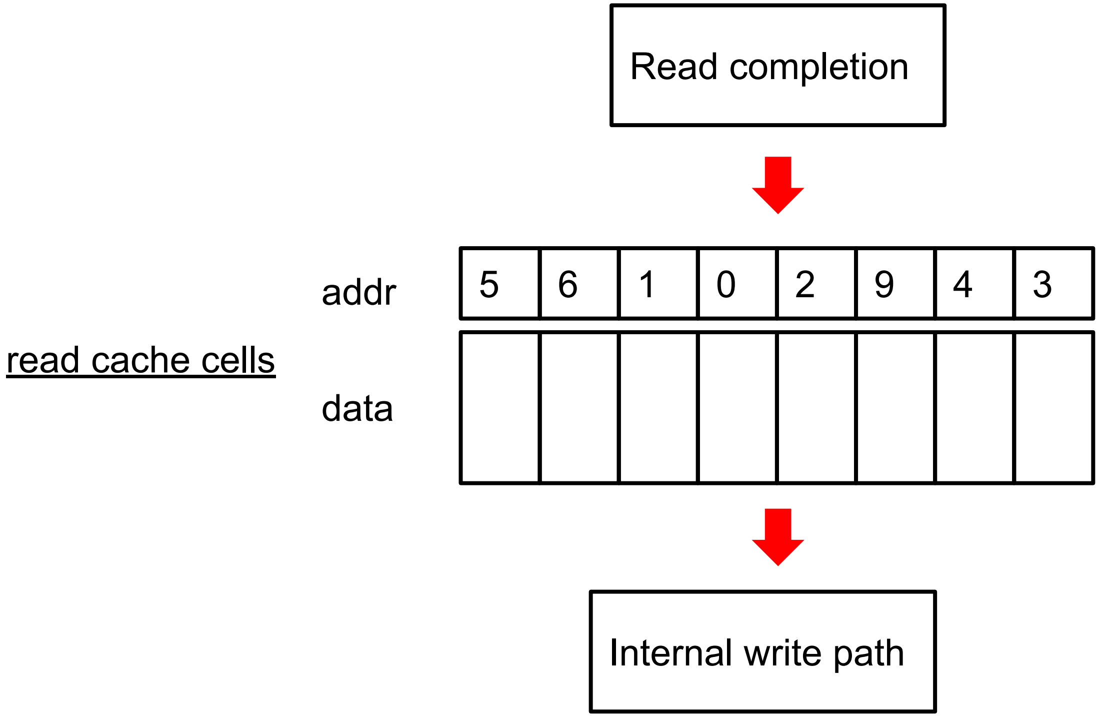
Requirements

- Read caching works in log-structured manner as well as write caching. They should co-exists in the same cache device. => share the internal write path
- Don't cache read data larger than user-defined threshold.
  - Problem: Since we split the in-coming bio to 4KB chunks, we lost the information of how large the original bio is. => read cache cells

The basic concept:
Write the read data after completion



Thresholding



Read data are once buffered in read cache cells so we can detect the sequentiality. In this figure, we can detect sequences of length 7 (from 0 to 6) and a separate length 1 (only 9), respectively.

Sequentiality is detected to not cache data sequence which is too sequential.