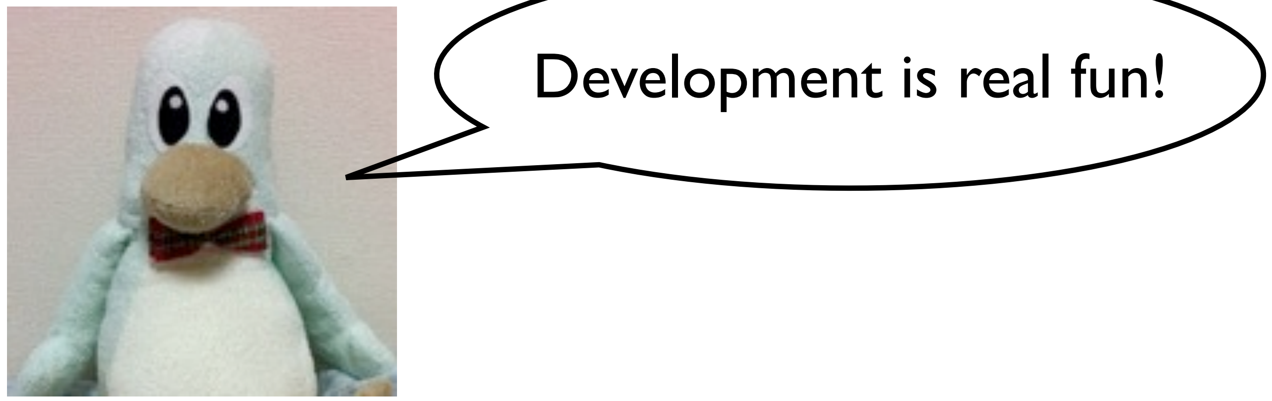
What is Writeboost

Akira Hayakawa (@akiradeveloper)



The role of this slides

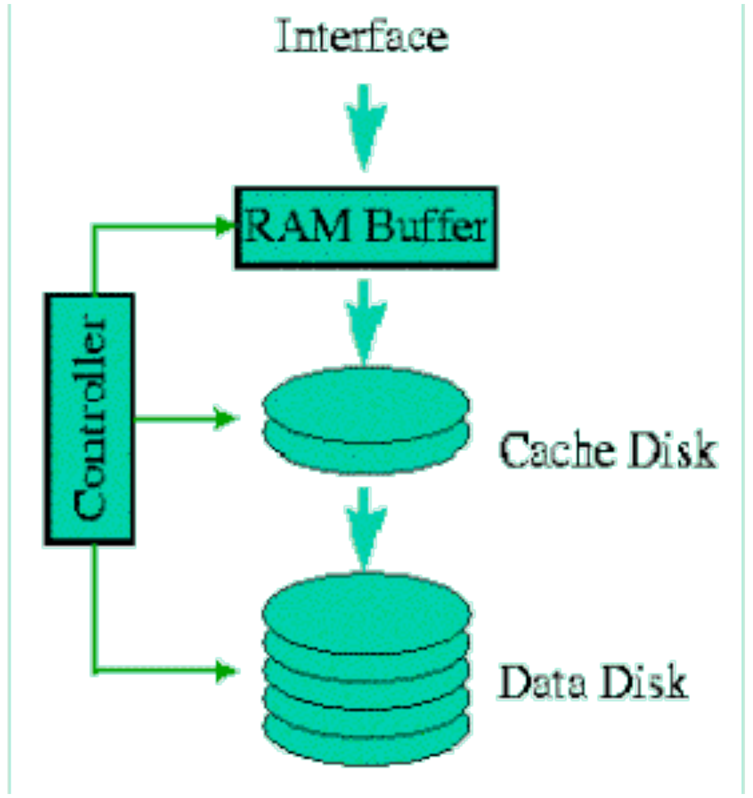
- Supplimentary document of the Documentation/writeboost.txt
- Showing figures is important but explanations aren't. Helps you read the code for more detail.

Features of Writeboost

- Block-level log-structured caching module for Linux, influenced by Disk Caching Disk (DCD).
- Pure write caching (unlike other caching softwares).
    - Optimization for writes can also improve reads.
    - Prominent writeback optimization.

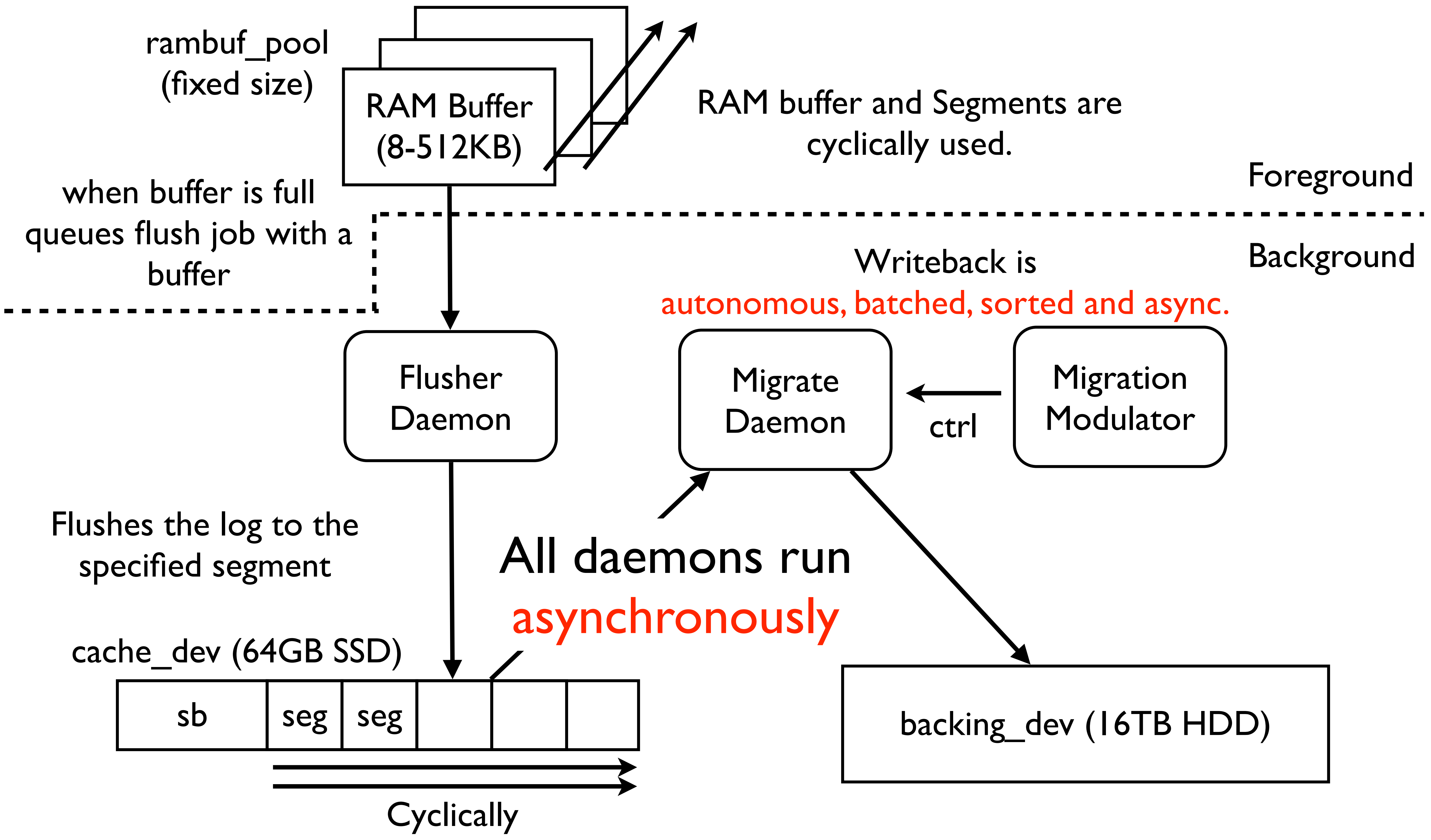
DCD?
Long long time ago, in the (ry

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

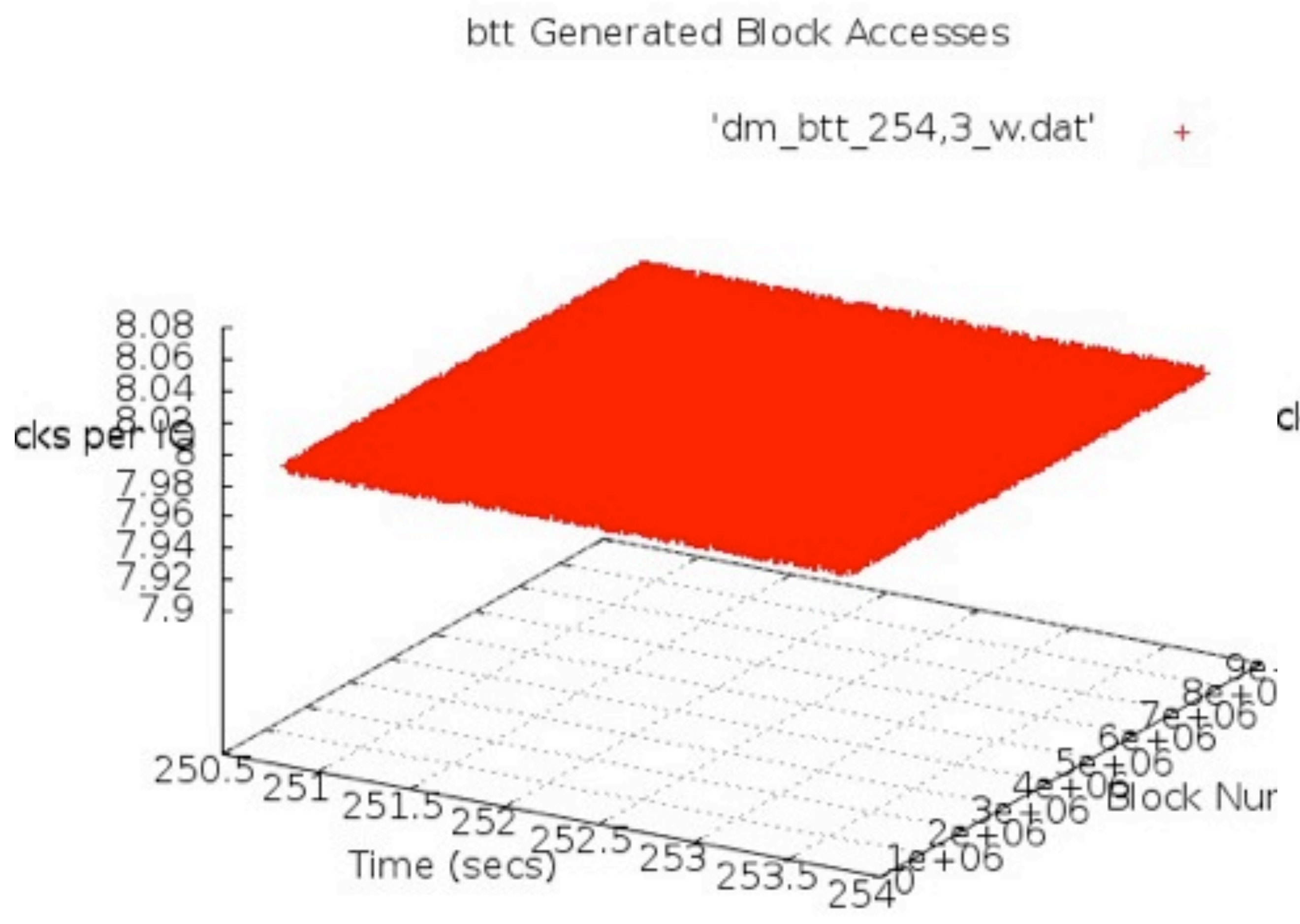


http://www.ele.uri.edu/research/hpcl/DCD/DCD.html

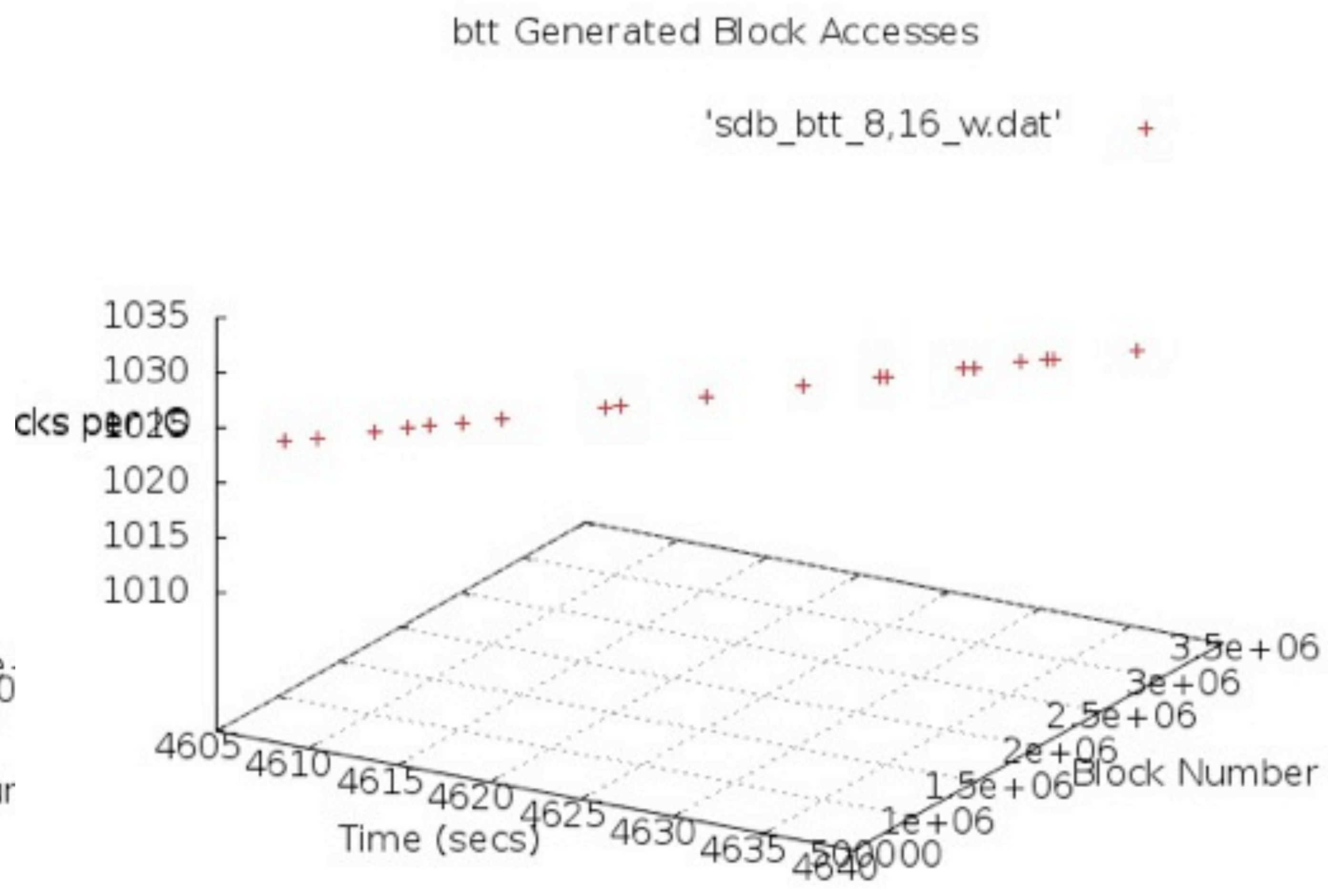
Basic Mechanism Overview


rambuf_pool
(fixed size)
RAM Buffer
(8-512KB)
RAM buffer and Segments are cyclically used.
Foreground
Background
when buffer is full queues flush job with a buffer
Writeback is
autonomous, batched, sorted and async.
Flusher Daemon
Migrate Daemon
ctrl
Migration Modulator
Flushes the log to the specified segment
All daemons run asynchronously
cache_dev (64GB SSD)
sb
seg
seg
backing_dev (16TB HDD)
Cyclically

Let’s visualize blktrace + bno_plot.py



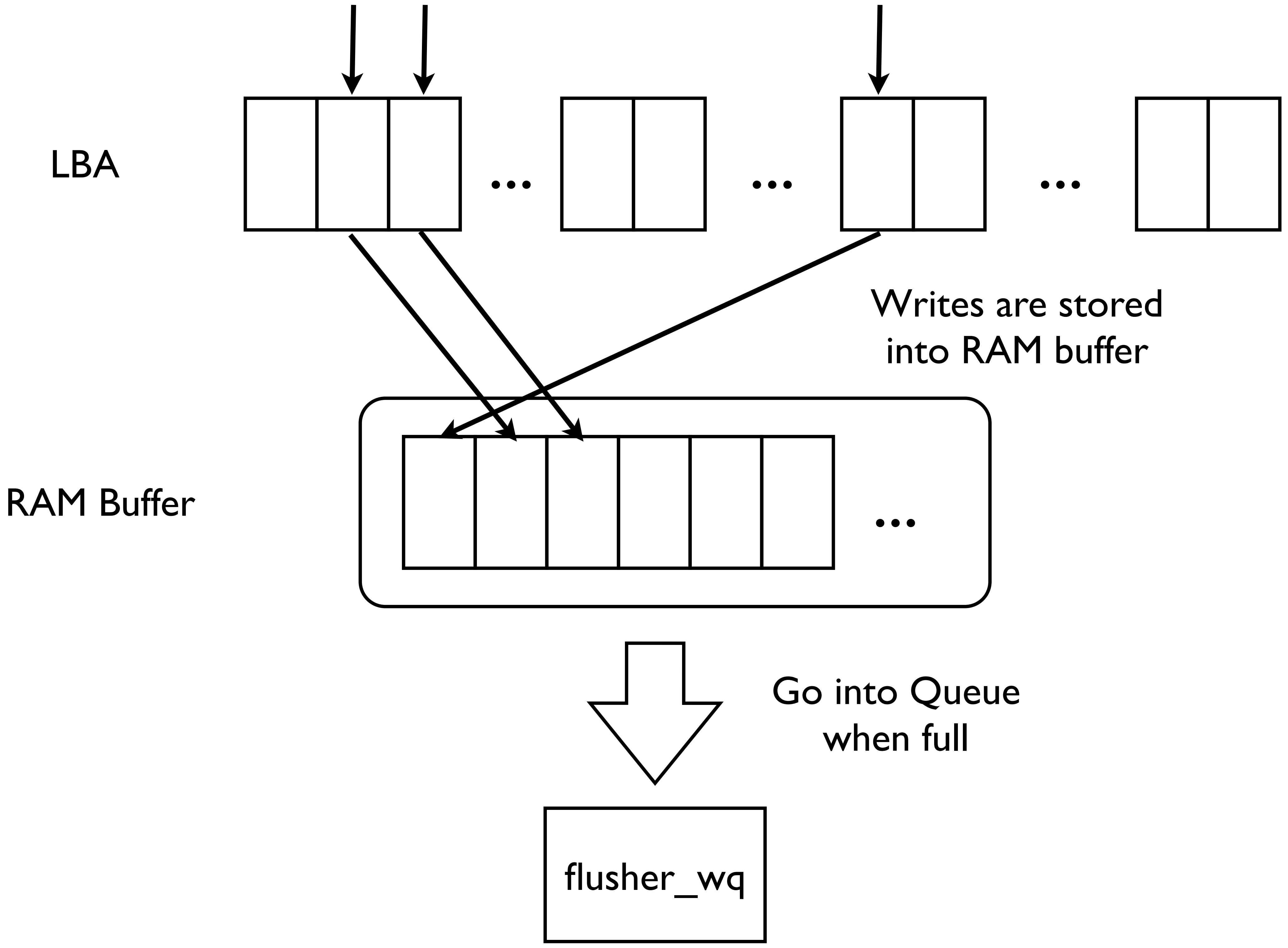
Random Writes to the Writeboost virtual device



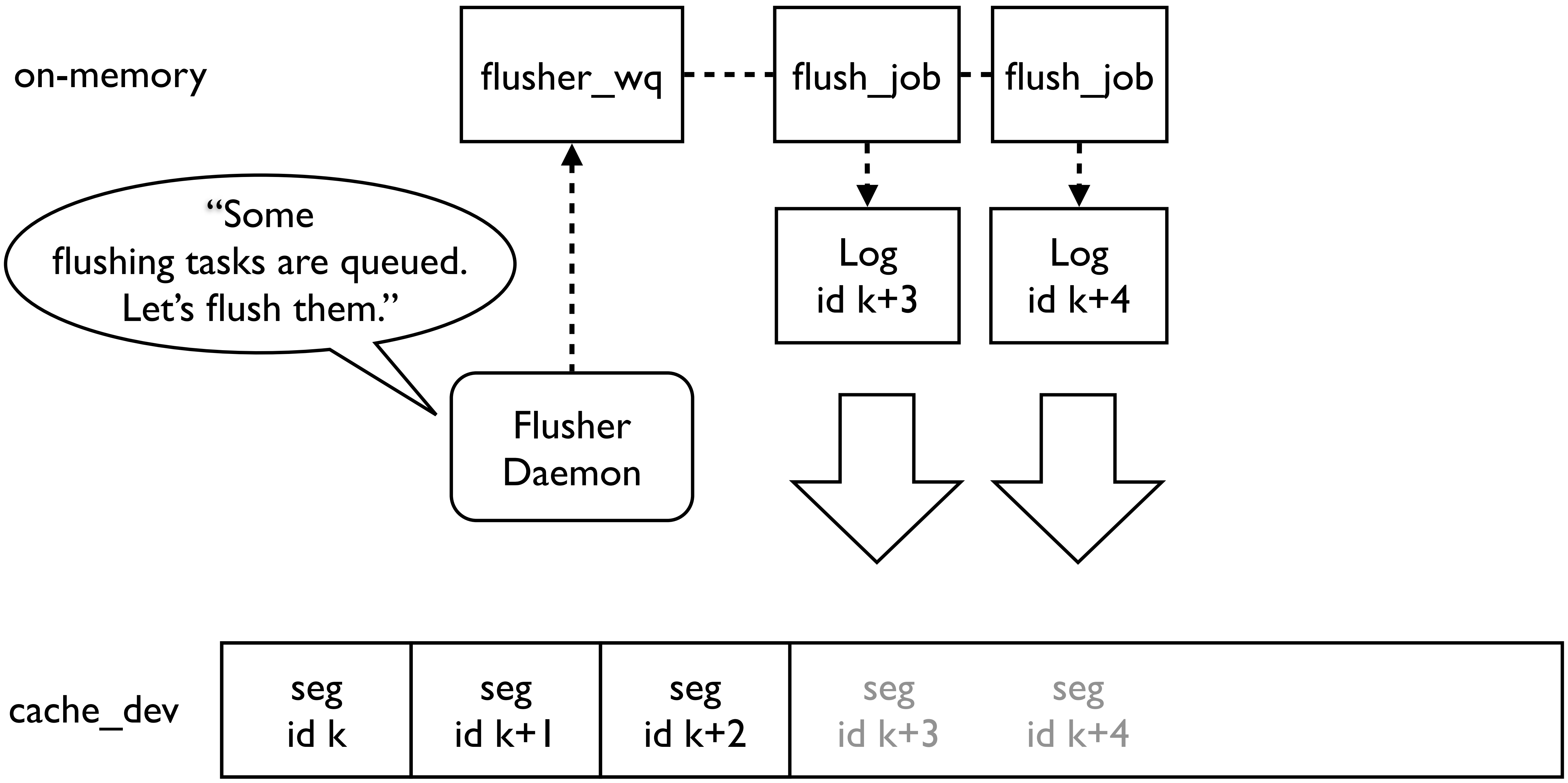
Sequential Writes to the cache device (little bit erroneous but shows the sequentiality)

Flushing Logs

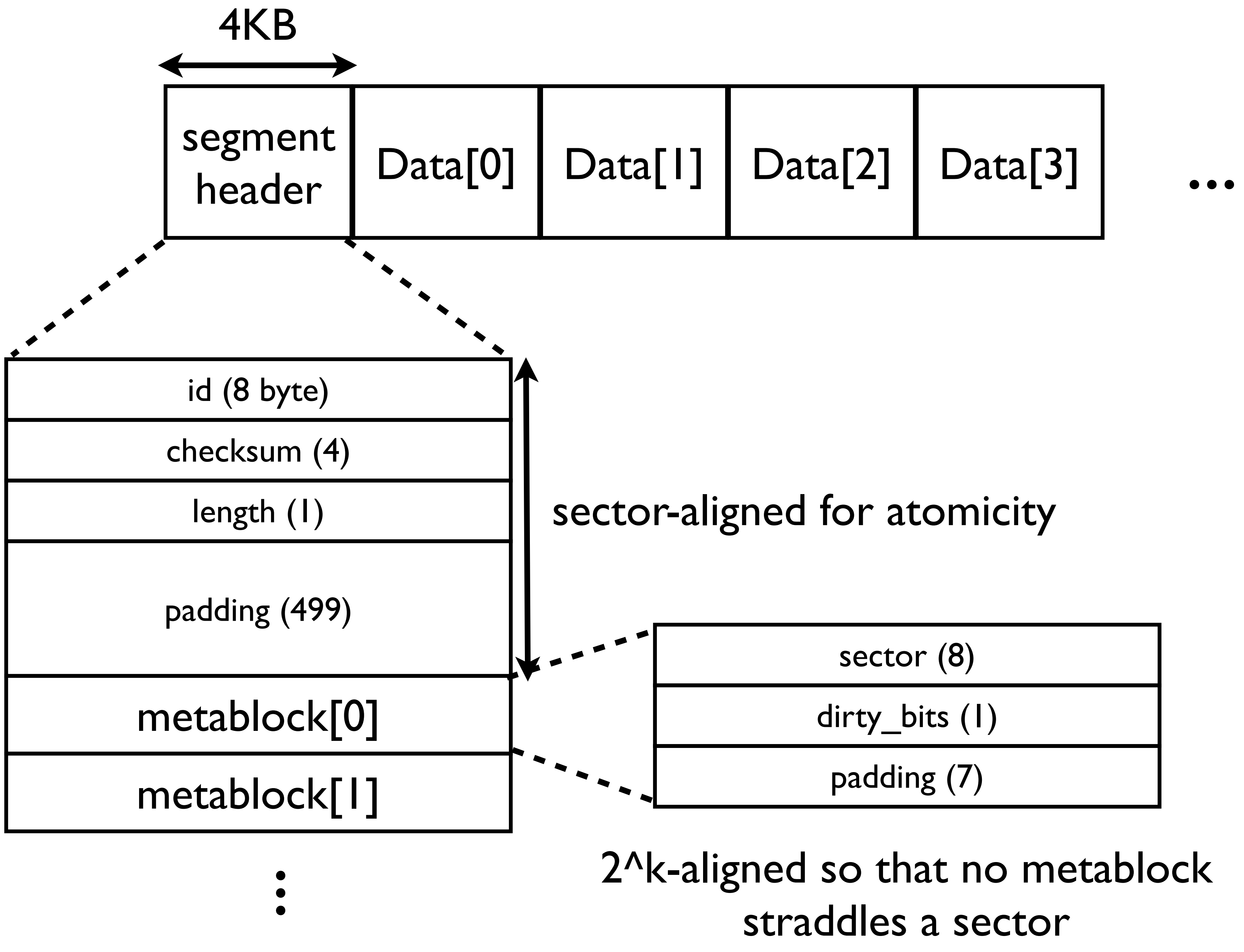
Foreground processing: Storing writes in RAM Buffer



Background Processing: Flusher Daemon (Using workqueue)

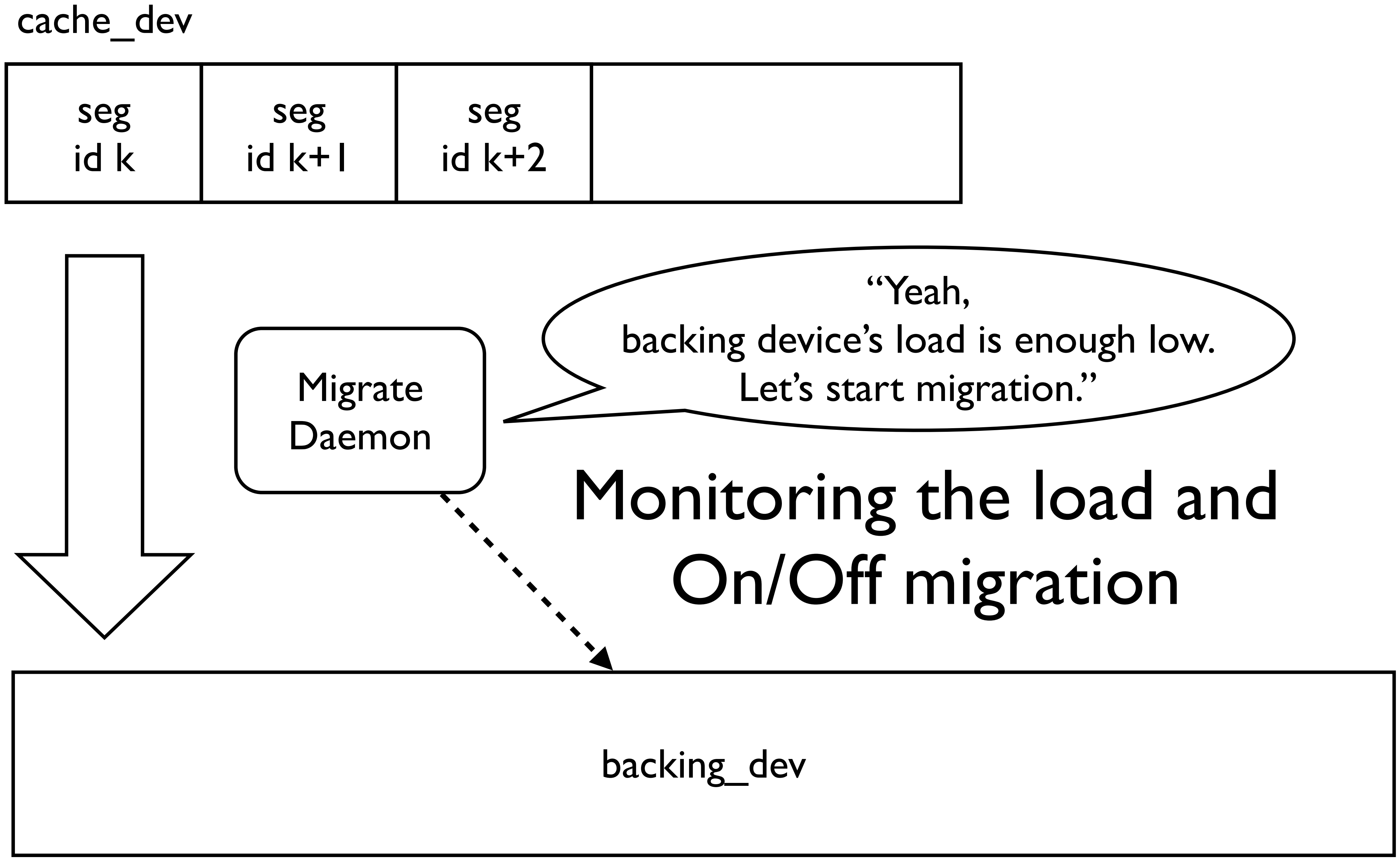


Log Format (Alignment care to crash durability)



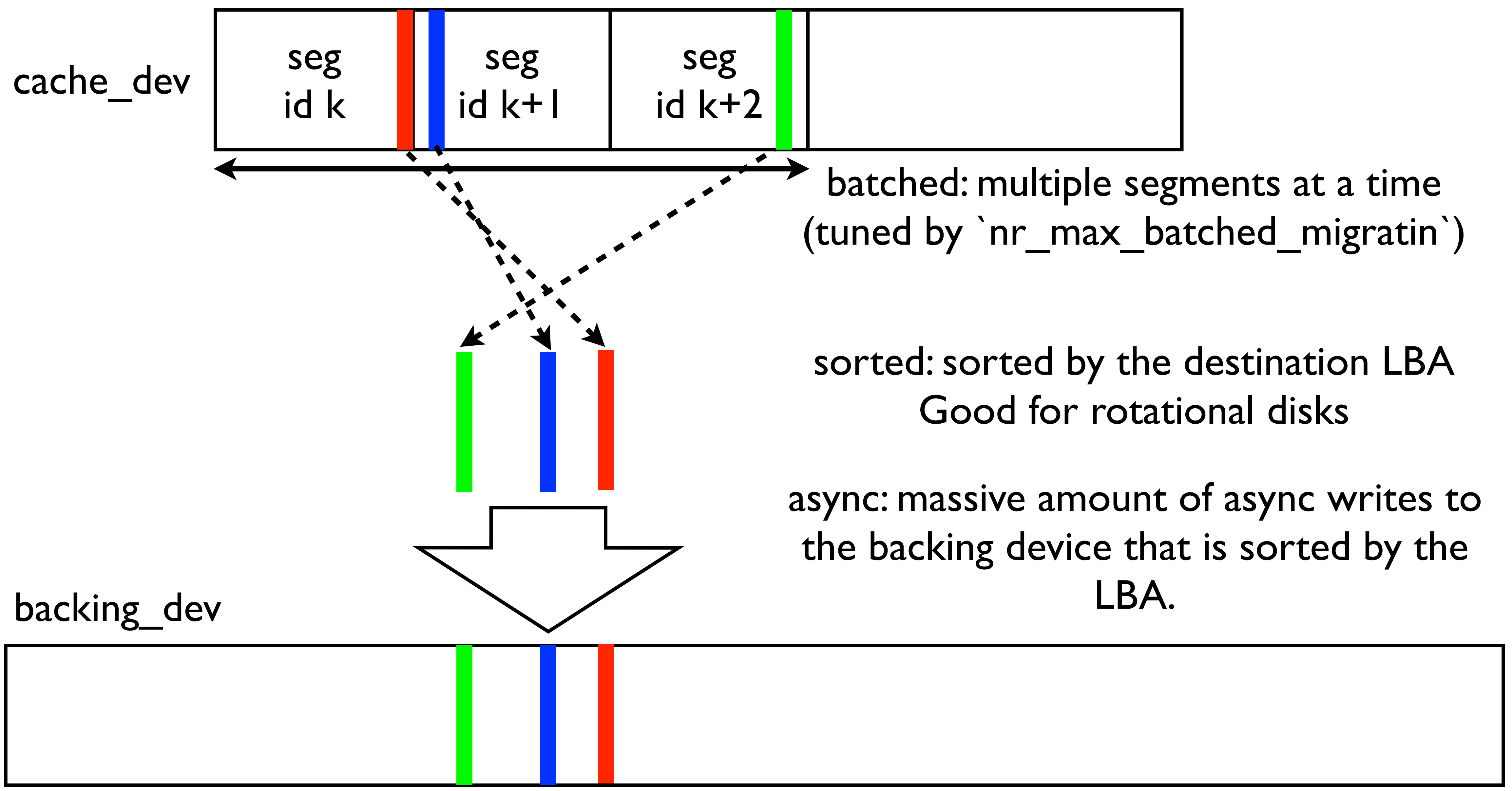
Writeback
(or Migration)

Autonomous Writeback Switching


cache_dev
seg id k
seg id k+1
seg id k+2
Migrate Daemon
"Yeah, backing device's load is enough low. Let's start migration."
backing_dev

Monitoring the load and On/Off migration

Writeback is batched, sorted and async



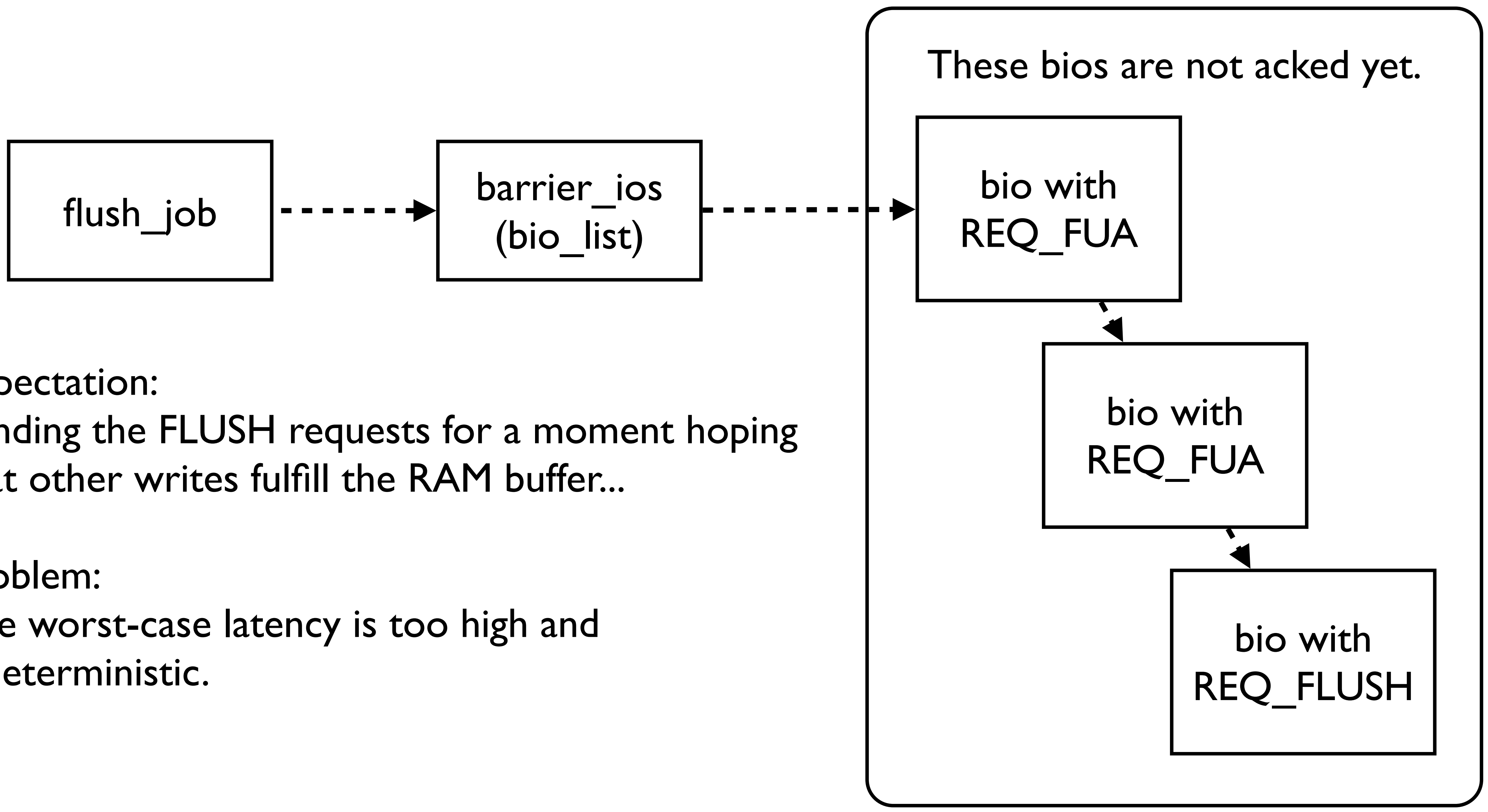
batched: multiple segments at a time (tuned by `nr_max_batched_migratin`)

sorted: sorted by the destination LBA
Good for rotational disks

async: massive amount of async writes to the backing device that is sorted by the LBA.

Persistency

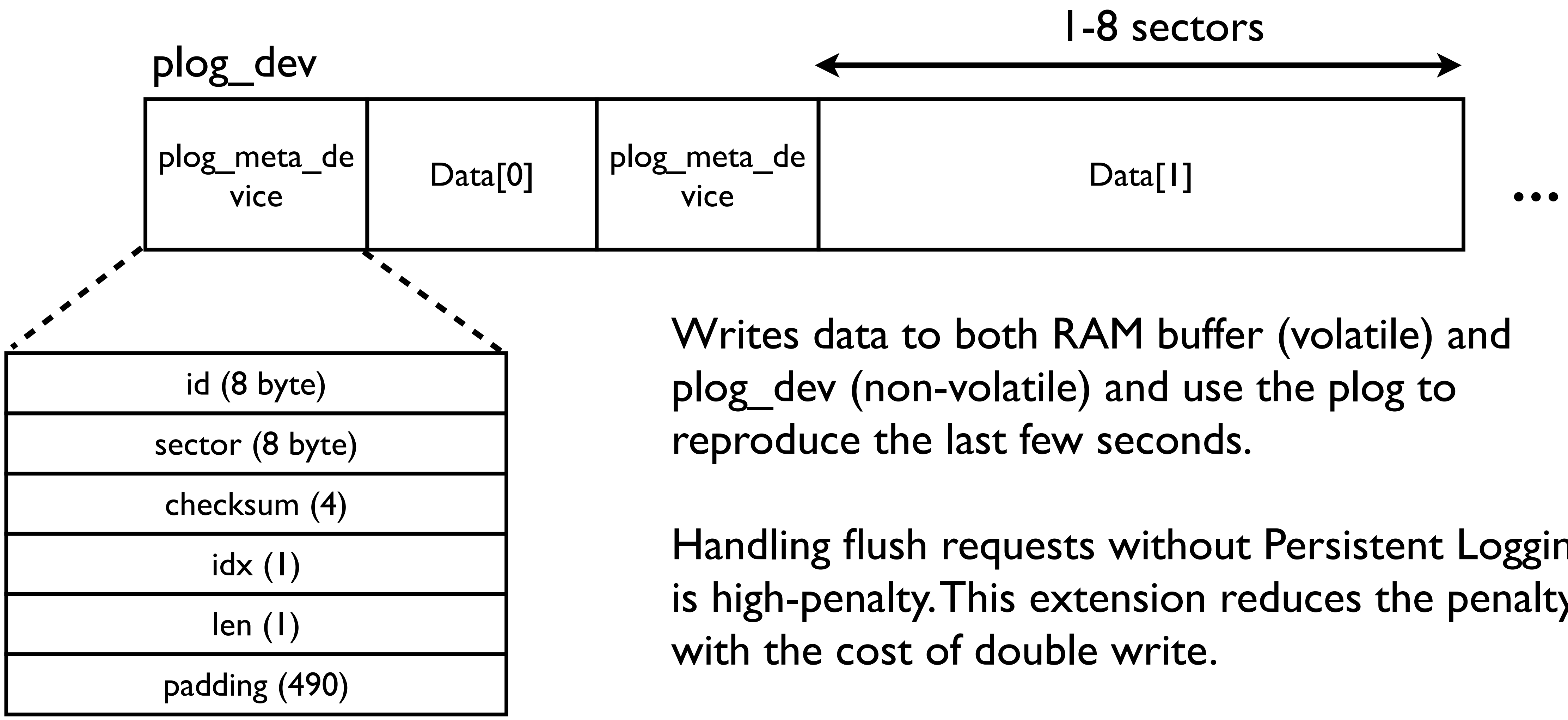
Deferred ACK for write barriers (Type 0 only)



Expectation:
Pending the FLUSH requests for a moment hoping that other writes fulfill the RAM buffer...

Problem:
The worst-case latency is too high and indeterministic.

"Persistent Logging" (type 1 or later, like full-data journaling)



Writes data to both RAM buffer (volatile) and plog_dev (non-volatile) and use the plog to reproduce the last few seconds.

Handling flush requests without Persistent Logging is high-penalty. This extension reduces the penalty with the cost of double write.

```
(pseudo code)
recover_cache():
    flush_plogs() # flushing all plogs to cache device
    replay_log_on_cache() # replay on cache device
```

Penalty Comparison: Handling FLUSH requests (w/ or wo plog)

- w/
  - flushes cache device only. (simple, yeah!)
- wo (deferred ACK for barrier writes)
  - makes a log that may not be fulfilled (maybe, after some delay).
  - queues it.
  - flush daemon flushes the log (with REQ_FLUSH).
  - and then acks the pending FLUSH requests.

Any question or discussion is welcome either on dm-devel or via personal/office e-mails.