Improvements in compactions

Ted Yu

Agenda

- **Date Tiered Compaction**
- **In memory compaction**
- **Q/A**

About myself

- **Been working on HBase for 6 years**
- **HBase committer / PMC**
- **Senior Staff Engineer at Hortonworks**

Date Tiered Compaction

- **Inspired by Cassandra's Date Tiered Compaction**
- **Write access pattern is mainly sequential writes by time of data arrival**
- **Read access pattern is mainly time-range scans**

Date Tiered Compaction Cont'd

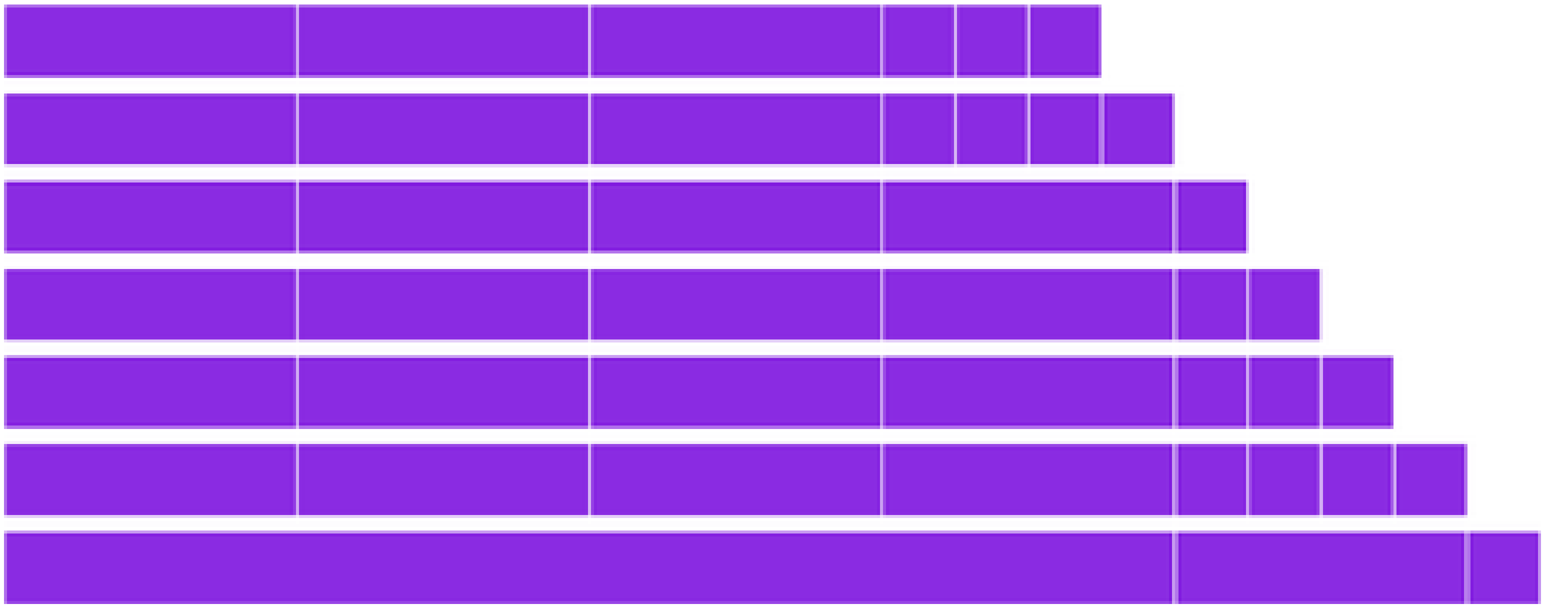


Figure 2. base window = 1 hour, windows per tier = 4

- **New time windows appear**
- **Old ones get merged into exponentially larger windows**
- **From https://labs.spotify.com/2014/12/18/date-tiered-compaction/**

Major config parameters

- **Base window: smallest time window for first tier**
- **Windows per tier: scale factor of window sizes from one tier to the next**
- **Max storefile age: how old it has to be before compaction stops – biggest tier**
- **Incoming windows threshold: number of files in incoming window before we compact to first tier**

- **http://hbase.apache.org/book.html#ops.date.tiered**

Benefits of Date Tiered Compaction

- **Better granularity beyond major compaction intervals for efficient timespan scans**
- **Reduced IO cost of compactions**
- **Efficient data rentention**
- **Better performance, lower latency**

Date Tiered Compaction Cont'd

- **HFiles are ordered by sequence Id**
- **Max timestamp is used to determine order of files and compaction window as secondary order**
- **Plugged-in per-window compaction policy to reduce wasteful compaction**
- **Suitable for time series data loaded periodically with minimum time range overlap … and more cases**

Date Tiered Compaction Cont'd

- **For the files carrying the following (seqId, timestamp) pairs:**
- **(1,0), (2, 13), (3,3), (4,10), (5,11), (6,1), (7,2), (8,12), (9,14), (10,15)**

- **After scan and update:**
- **(1,0), (2, 13), (3,13), (4,13), (5,13), (6,13), (7,13), (8,13), (9,14), (10,15)**

Date Tiered Compaction Cont'd

- **Undesirable scenario: file on the lower tier has long tails**
- **HBASE-15400, major and minor compactions with splitting by window boundaries will help**

- **All servers in the cluster will promote windows to higher tier at the same time**
- **using a compaction throttle is recommended**

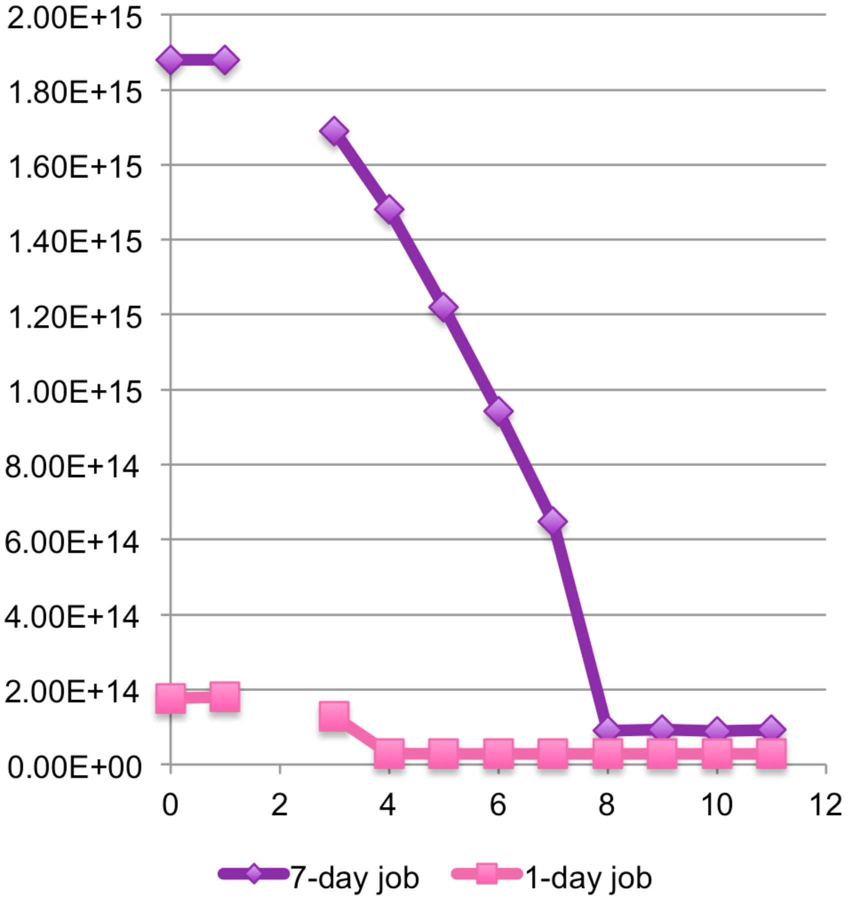
Date Tiered Compaction Cont'd

- **This compaction policy is unsuitable for following cases**

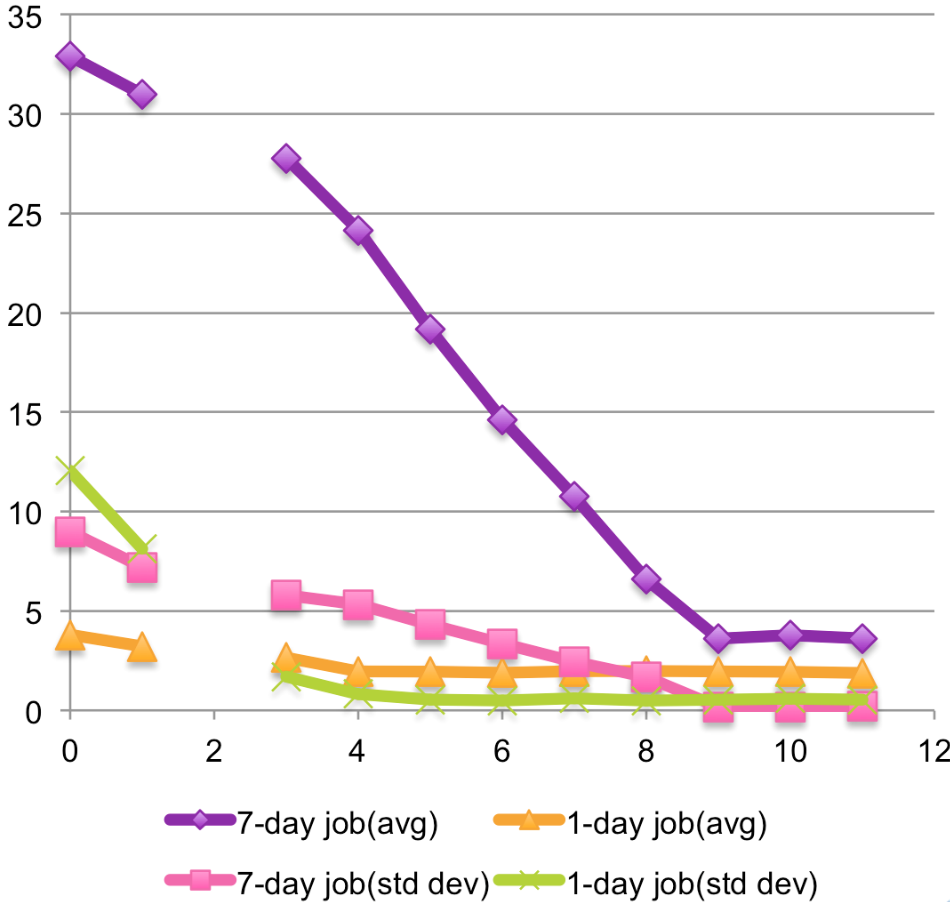
- **future timestamp is used in writes**
- **frequent deletes and updates**
- **random gets without a time range**
- **bulk load of heavily overlapping time-range data**

Perf Validation (days after turned on)

- HDFS Read Bytes

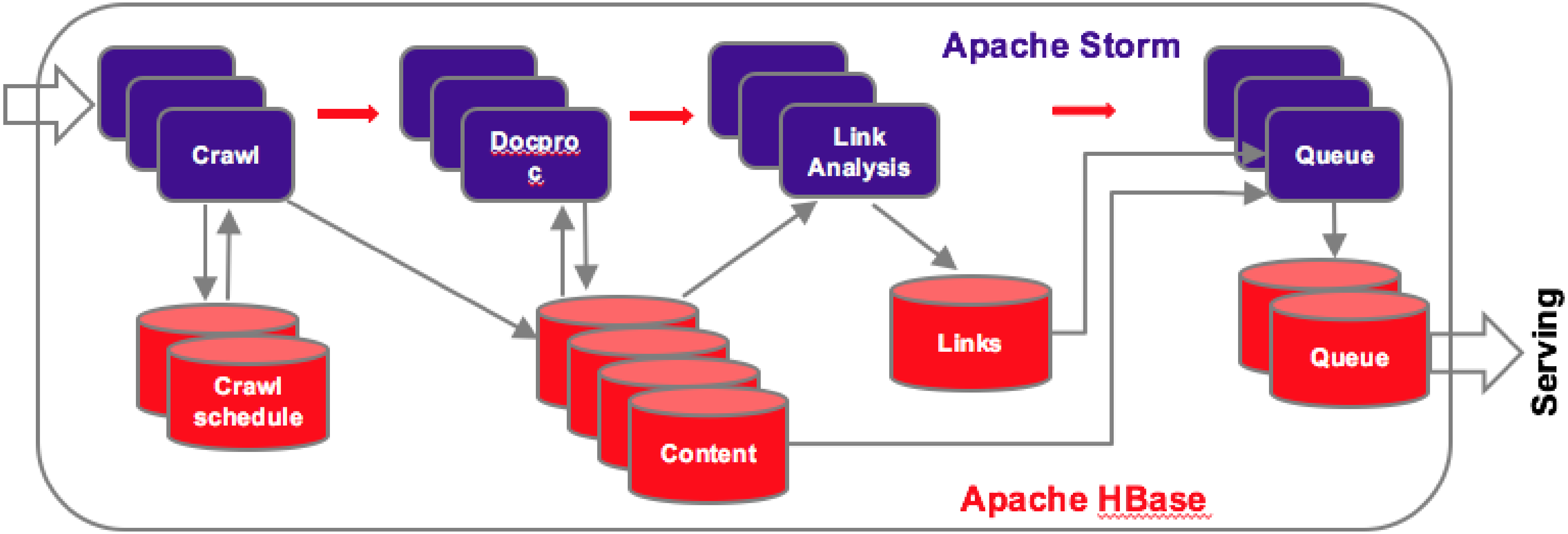


- Mapper Run Time Minutes



Dynamic Content Processing

- **Sieve – Yahoo's real-time content management platform**
- **Real-time content processing pipelines**
- **Storage and notifications on the same platform**



Workload Characteristics

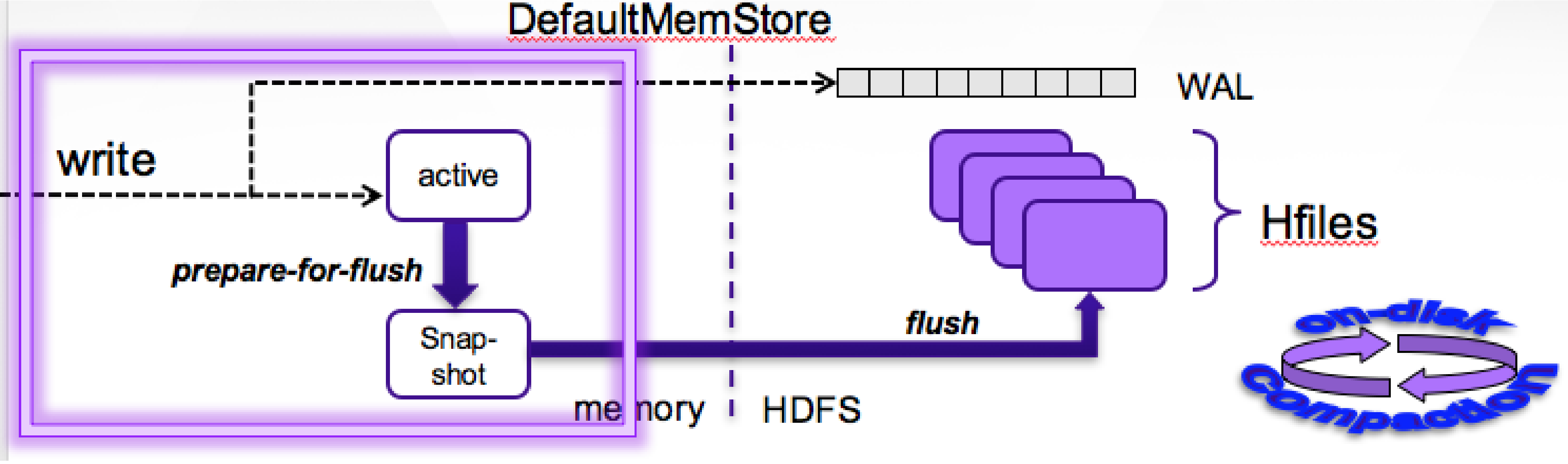
- **Small working set** but not necessarily a FIFO queue
- **Short life-cycle** delete message after processing it
- **High-churn workload** message state can be updated
- **Frequent scans** for consuming message

Two Basic Ideas

- **In-Memory Compaction**
  - Exploit redundancies in the workload to eliminate duplicates in memory
  - Gain is proportional to the duplicate ratio
- **In-Memory Index Reduction**
  - Reduce the index memory footprint, less overhead per cell
  - Gain is proportional to the cell size
- **Prolong in-memory lifetime, before flushing to disk**
  - Reduce write amplification effect (overall I/O)
  - Reduce retrieval latencies

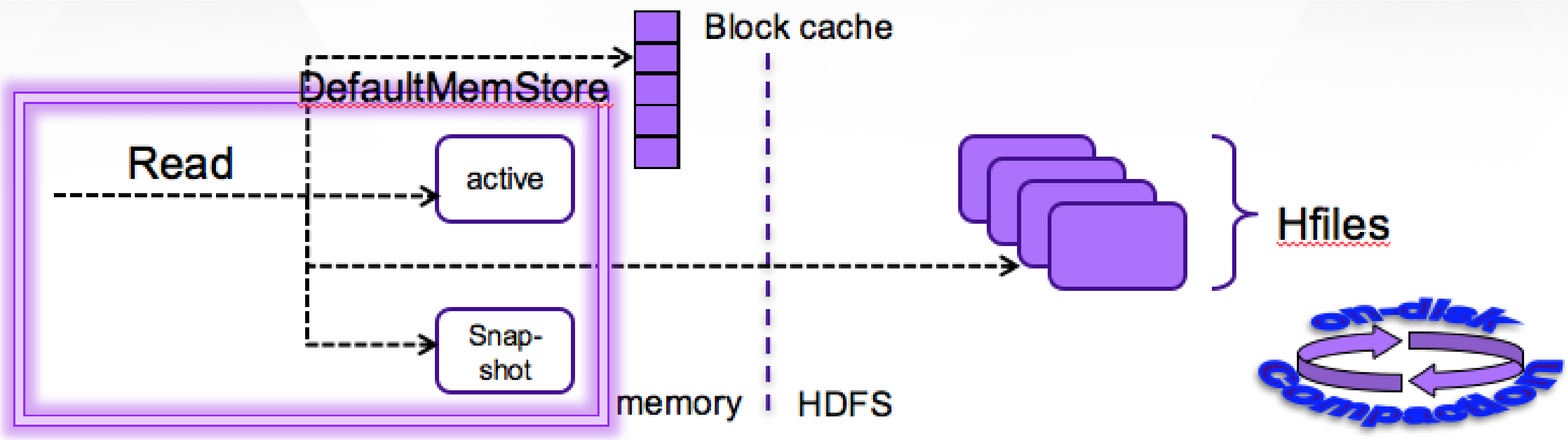
In-Memory Compaction Design

- **Random writes are absorbed in an active segment**
- **When active segment is full**
  - Becomes immutable segment (snapshot)
  - A new mutable (active) segment serves writes
  - Flushed to disk, truncate WAL
- **On-Disk compaction reads a few files, merge-sorts them, writes back new files**



HBase Reads

- **Random reads from active segment or snapshot or Hfiles (Block Cache)**
- **When data piles-up on disk**
  - Hit ratio drops and retrieval latency up
- **Compaction re-writes small files into fewer bigger files**

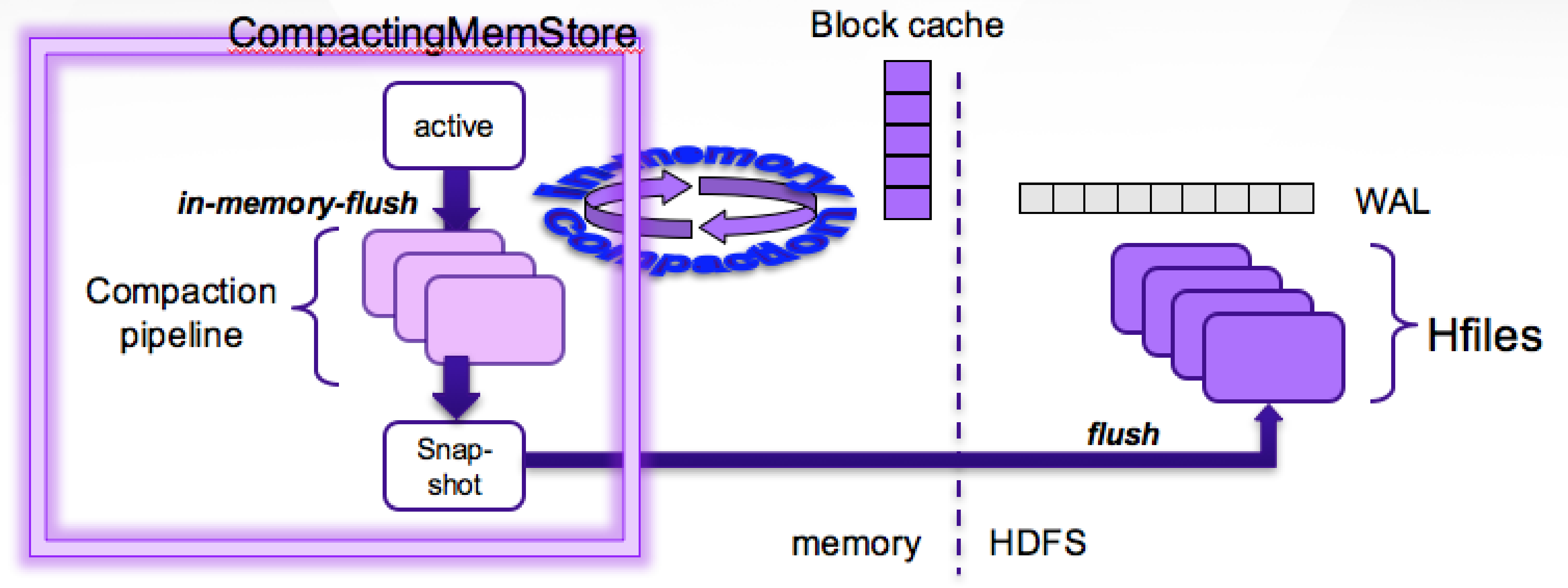


In-Memory Compaction

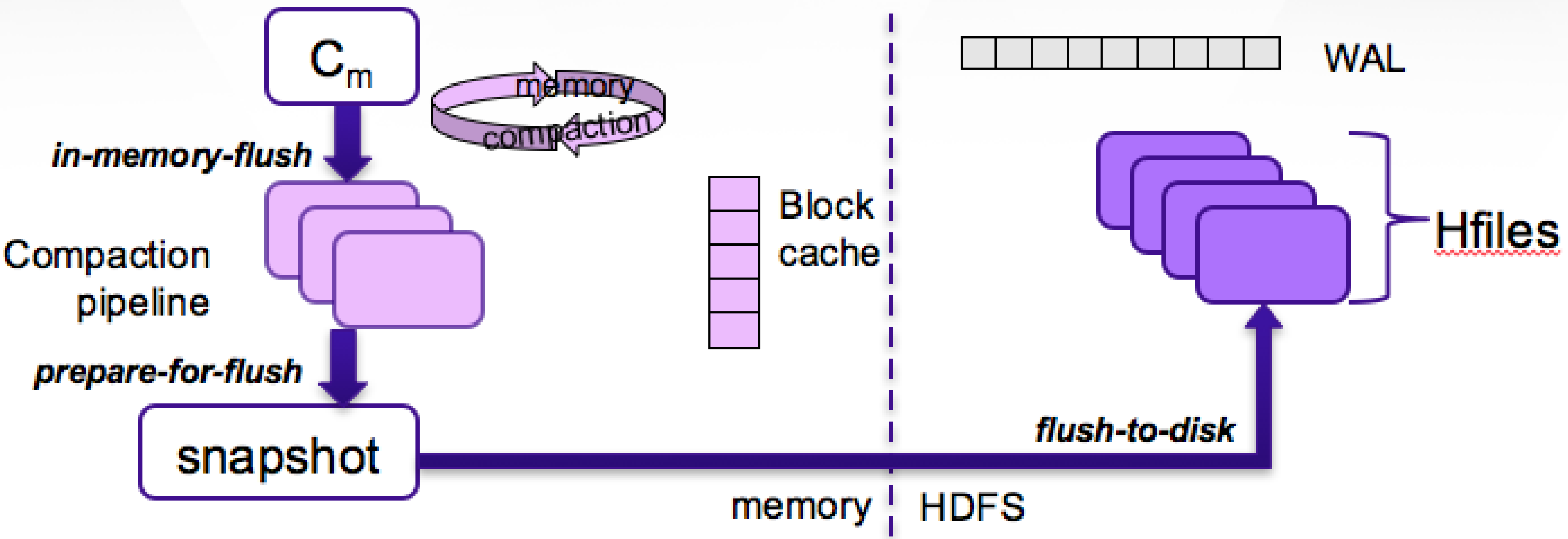
Active segment flushed to pipeline

Pipeline segments compacted in memory

Flush to disk only when needed

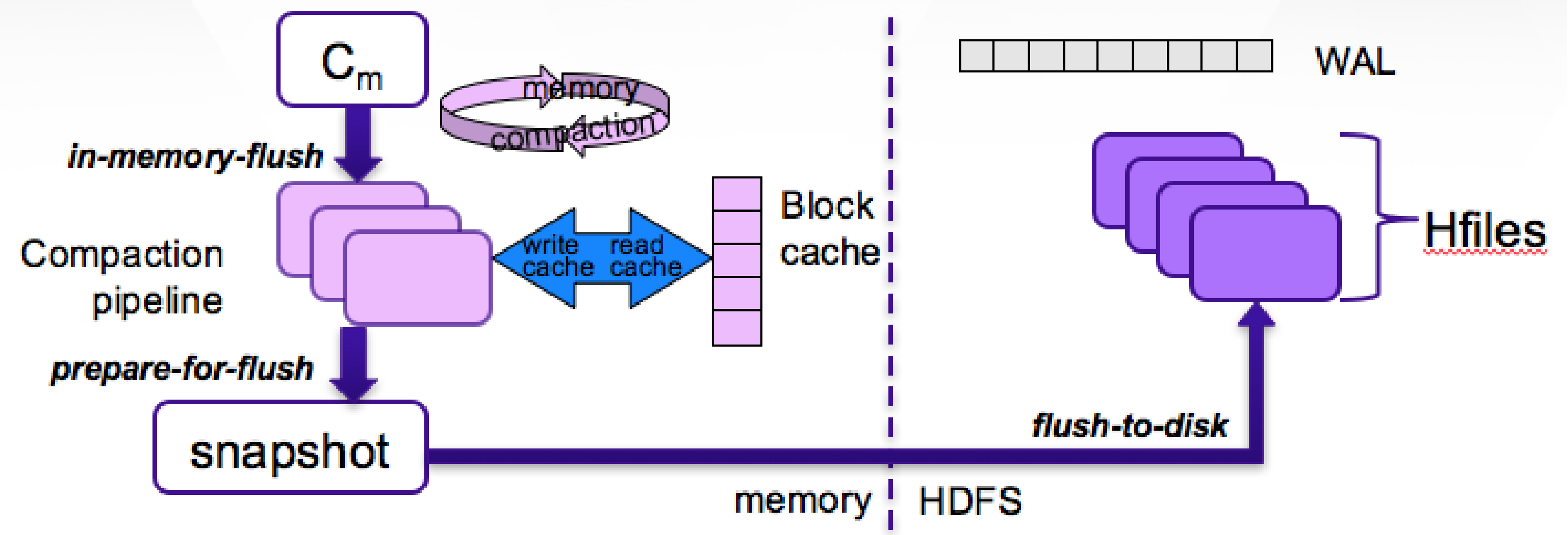


In-Memory Flush and Compaction



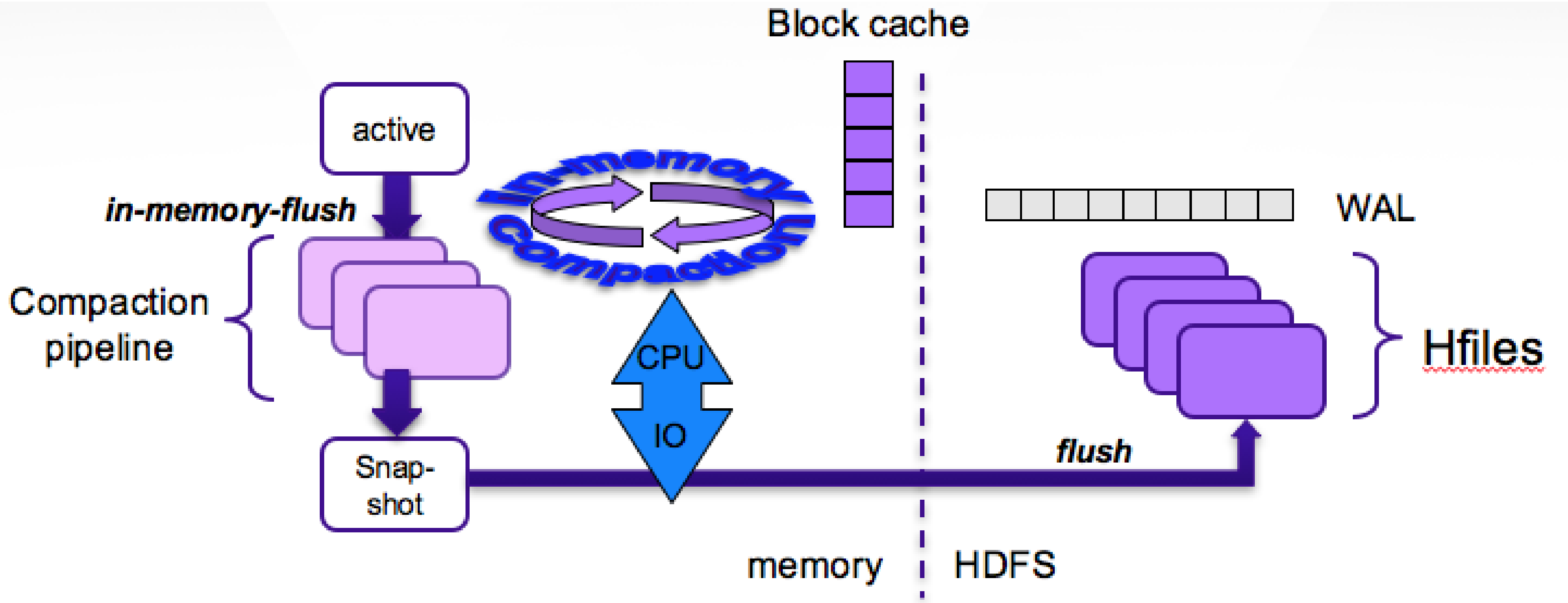
In-Memory Compaction: tradeoffs

- **Trade read cache (BlockCache) for write cache (compaction pipeline)**



In-Memory Compaction

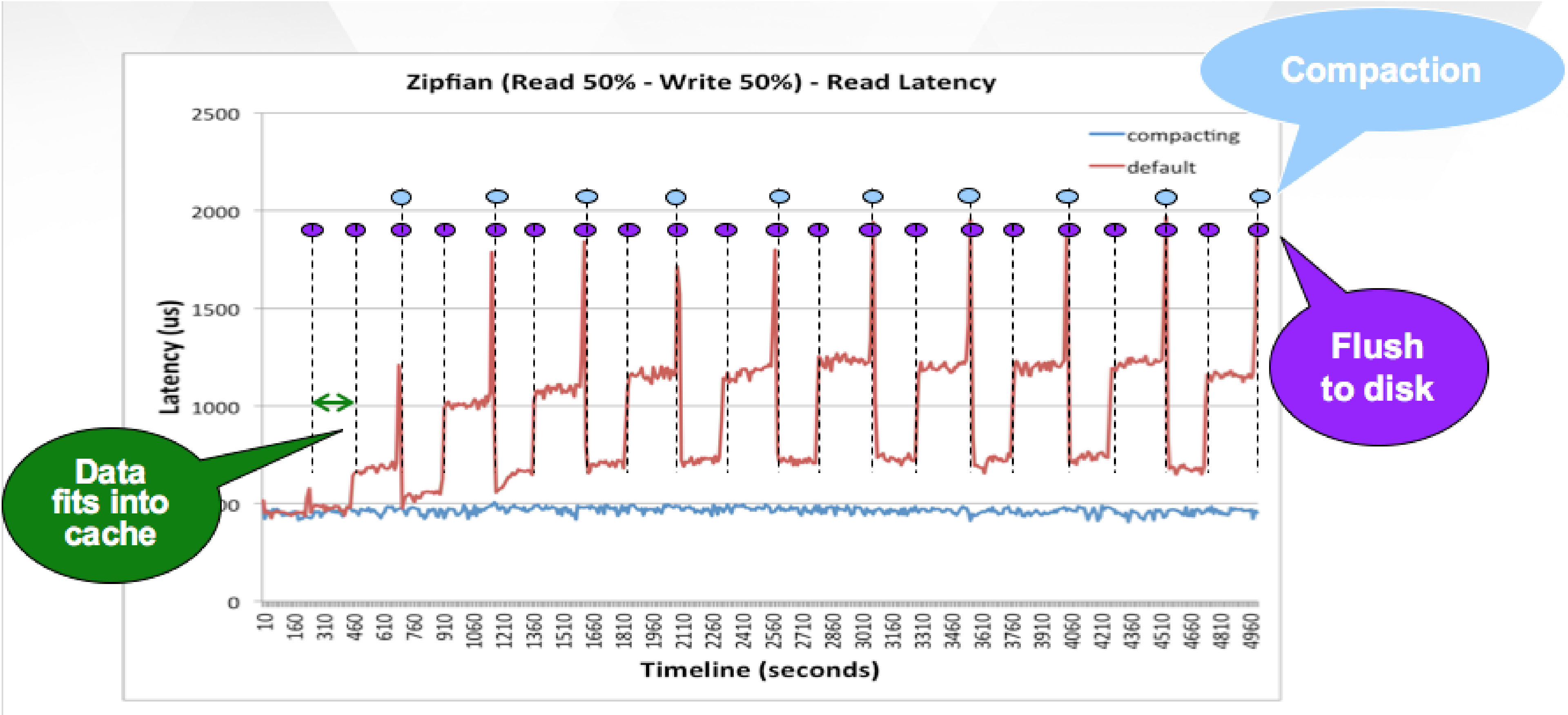
- **Trade CPU cycles for less I/O**



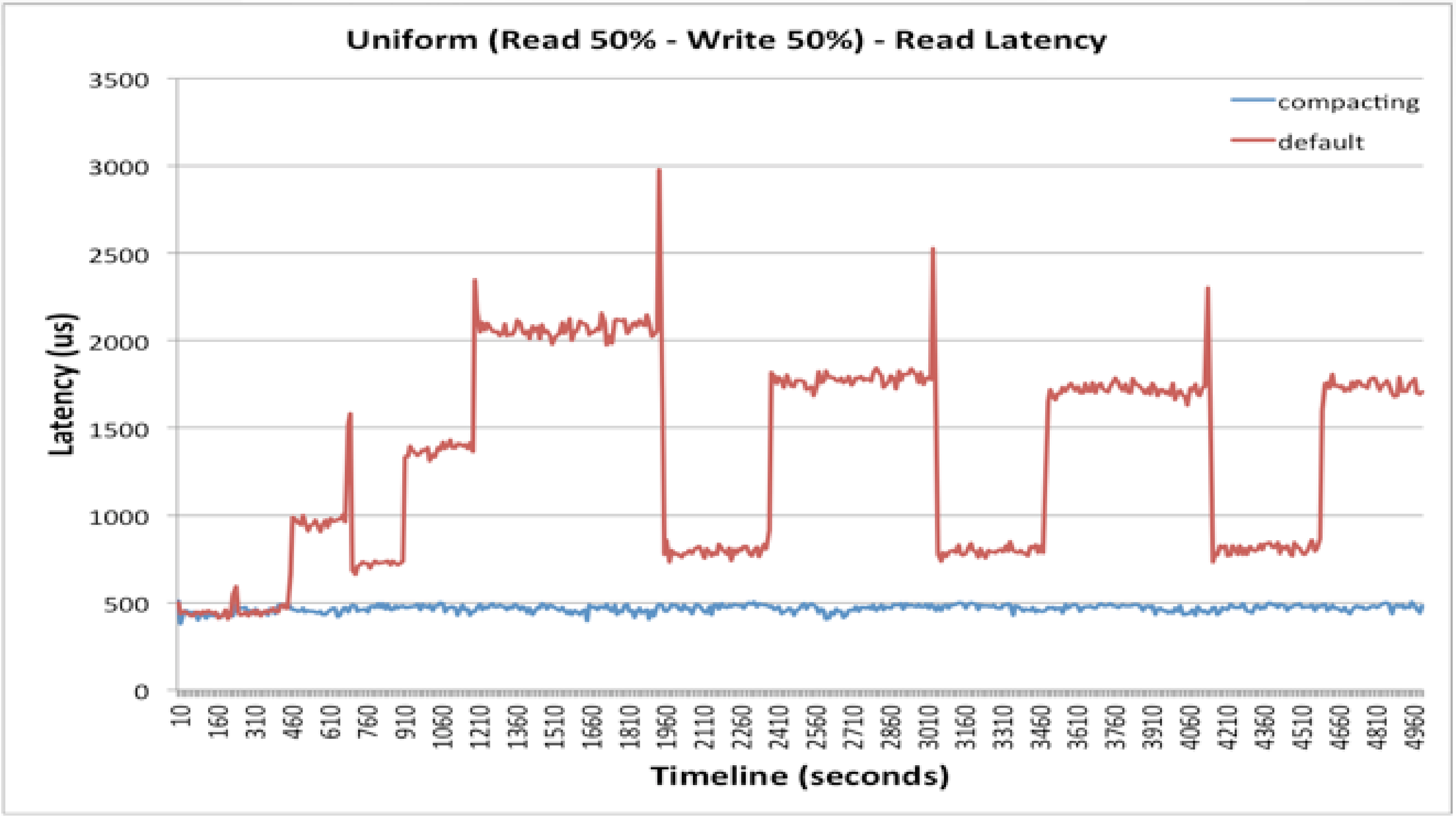
In-Memory Working Set

- **YCSB: compares Compacting vs. Default MemStore**
- **Small cluster: 3 HDFS nodes on a single rack, 1 RS**
- **High-churn workload, small working set**
  - 128,000 records, 1KB value field
  - 10 threads running 5 millions operations, various key distributions
  - 50% reads 50% updates, target 1000ops
  - 1% (long) scans 99% updates, target 500ops
- **Measure average latency over time**
  - Latencies accumulated over intervals of 10 seconds

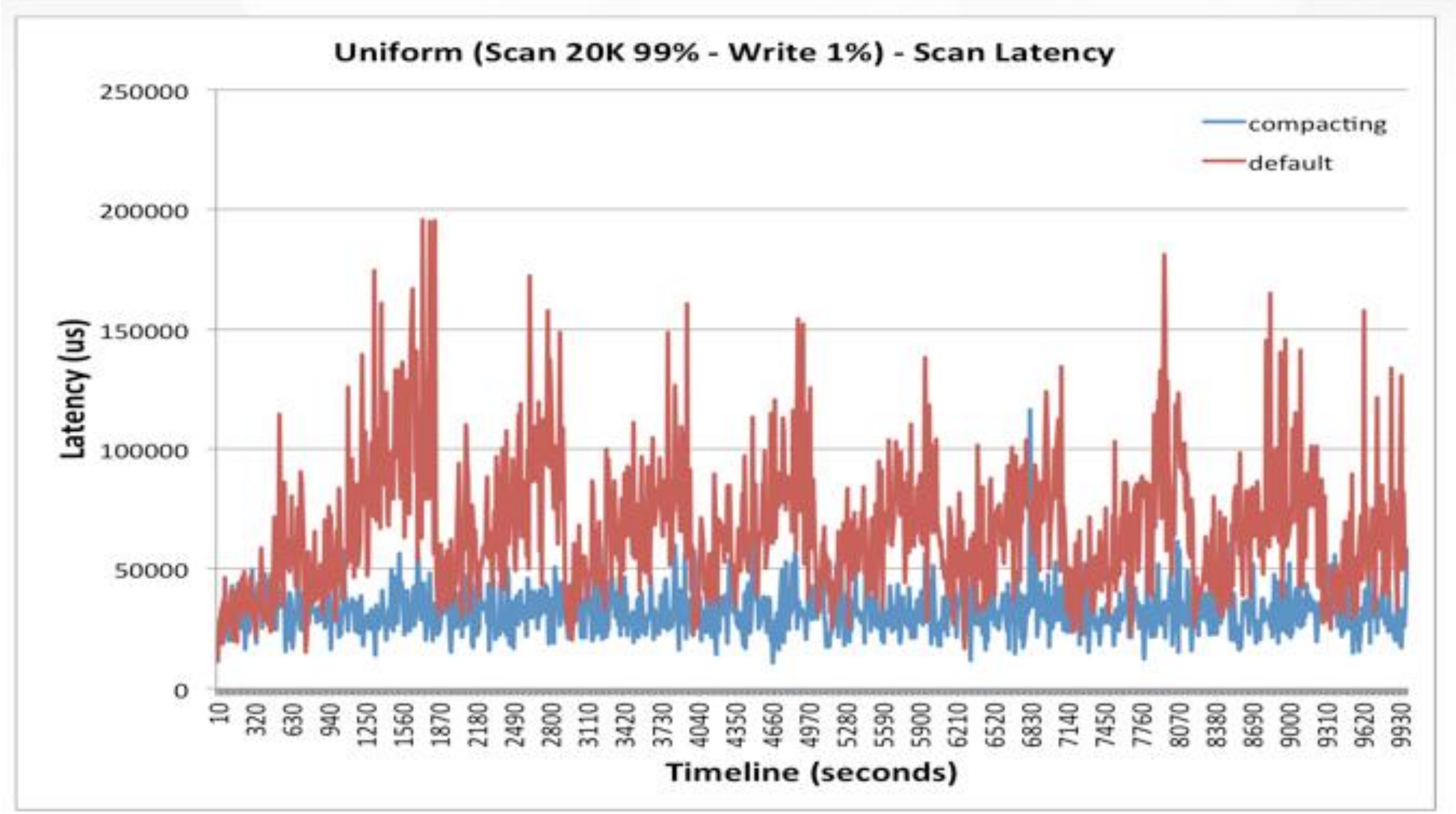
Read Latency (Zipfian Distribution)



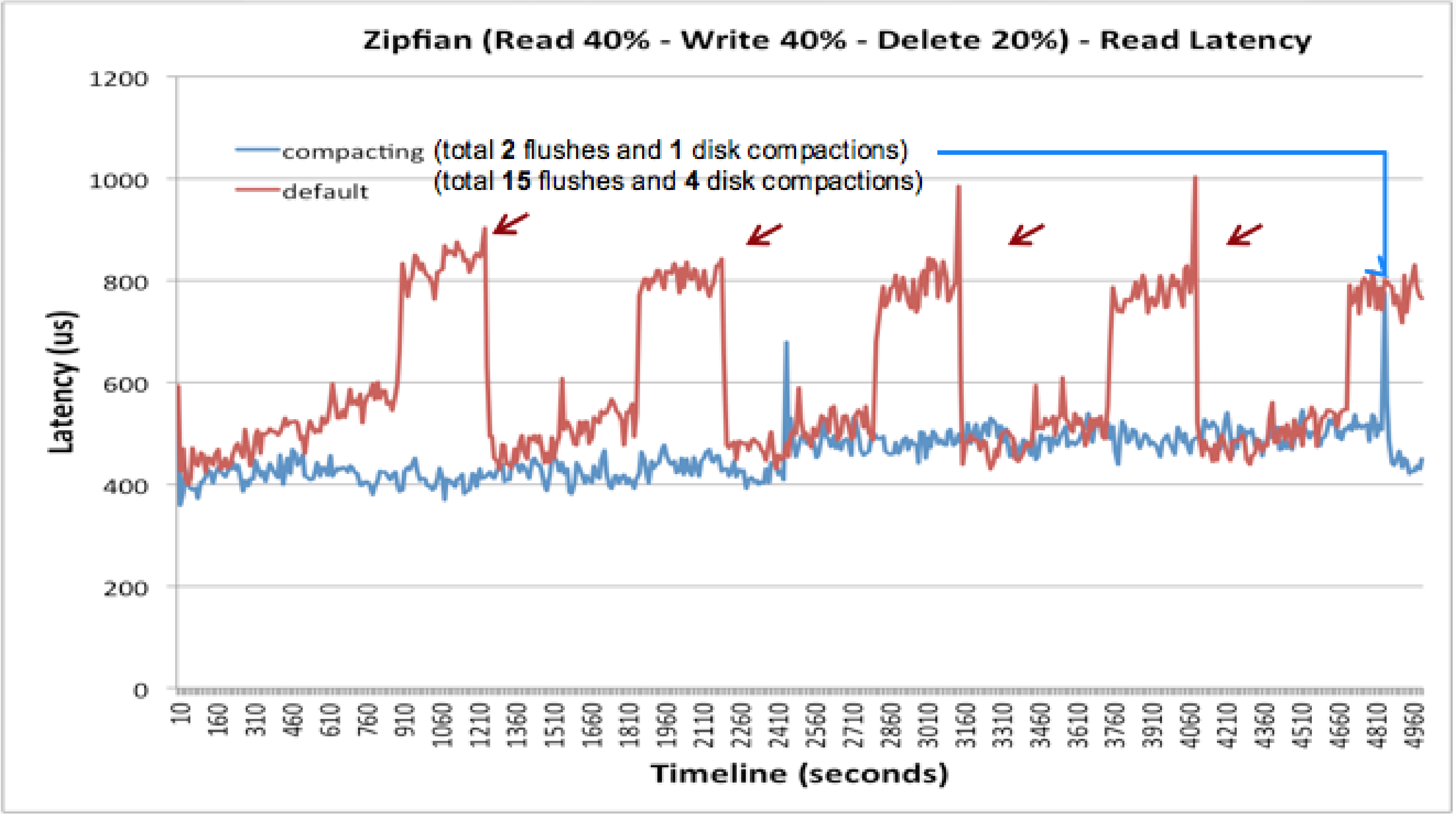
Read Latency (Uniform Distribution)



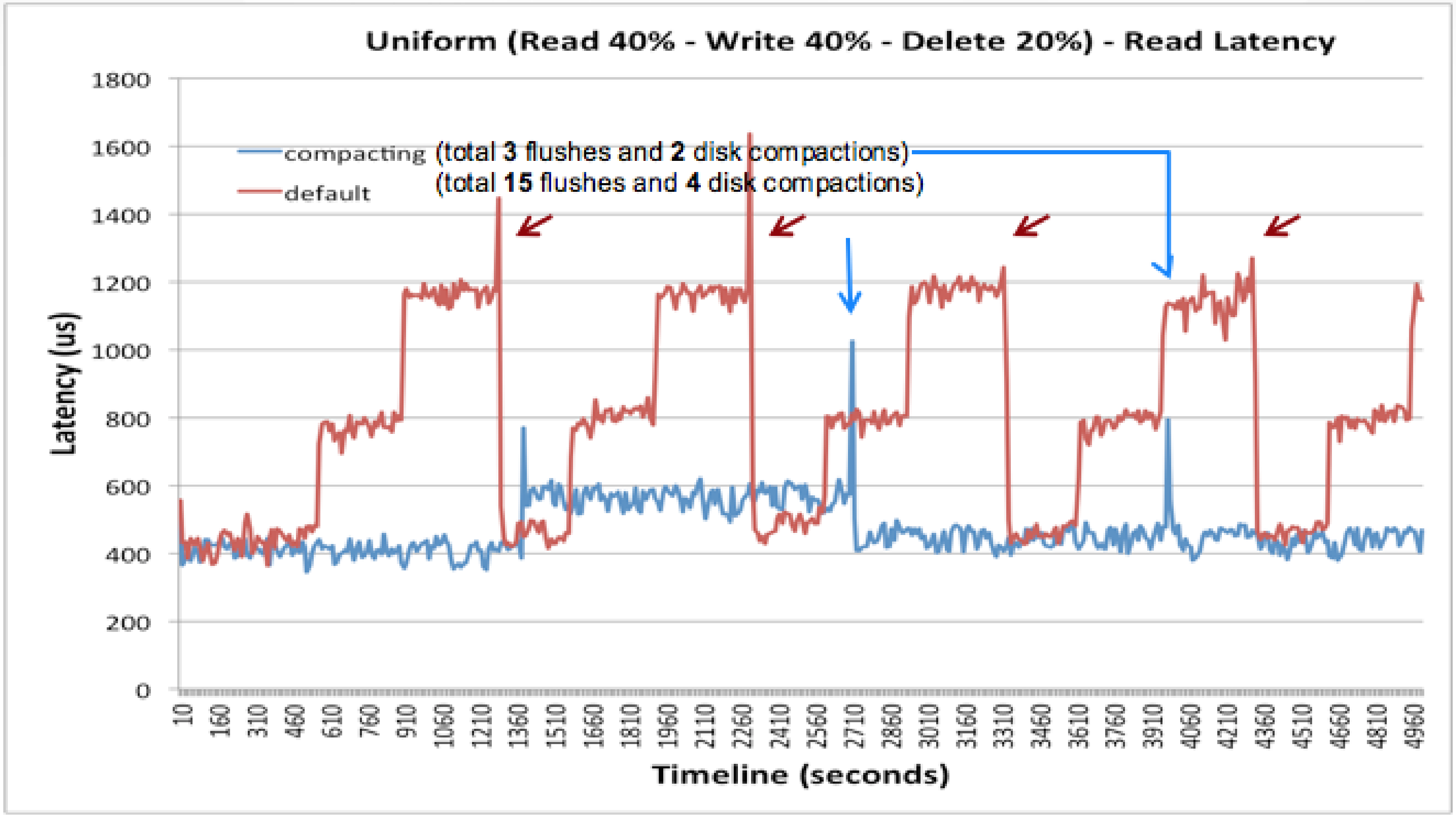
Scan Latency (Uniform Distribution)



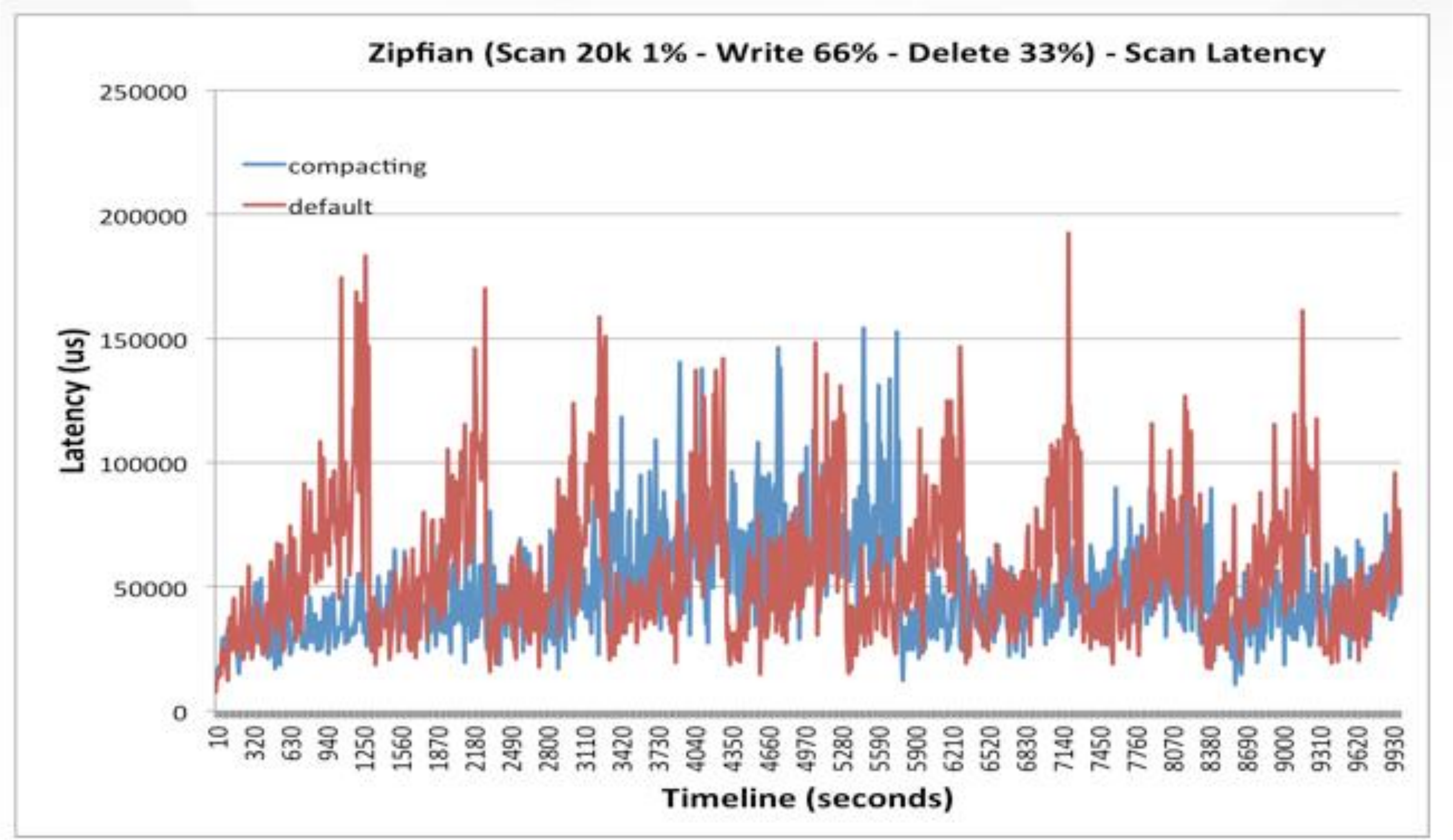
Read Latency(Zipfian Distribution) Handling Tombstones



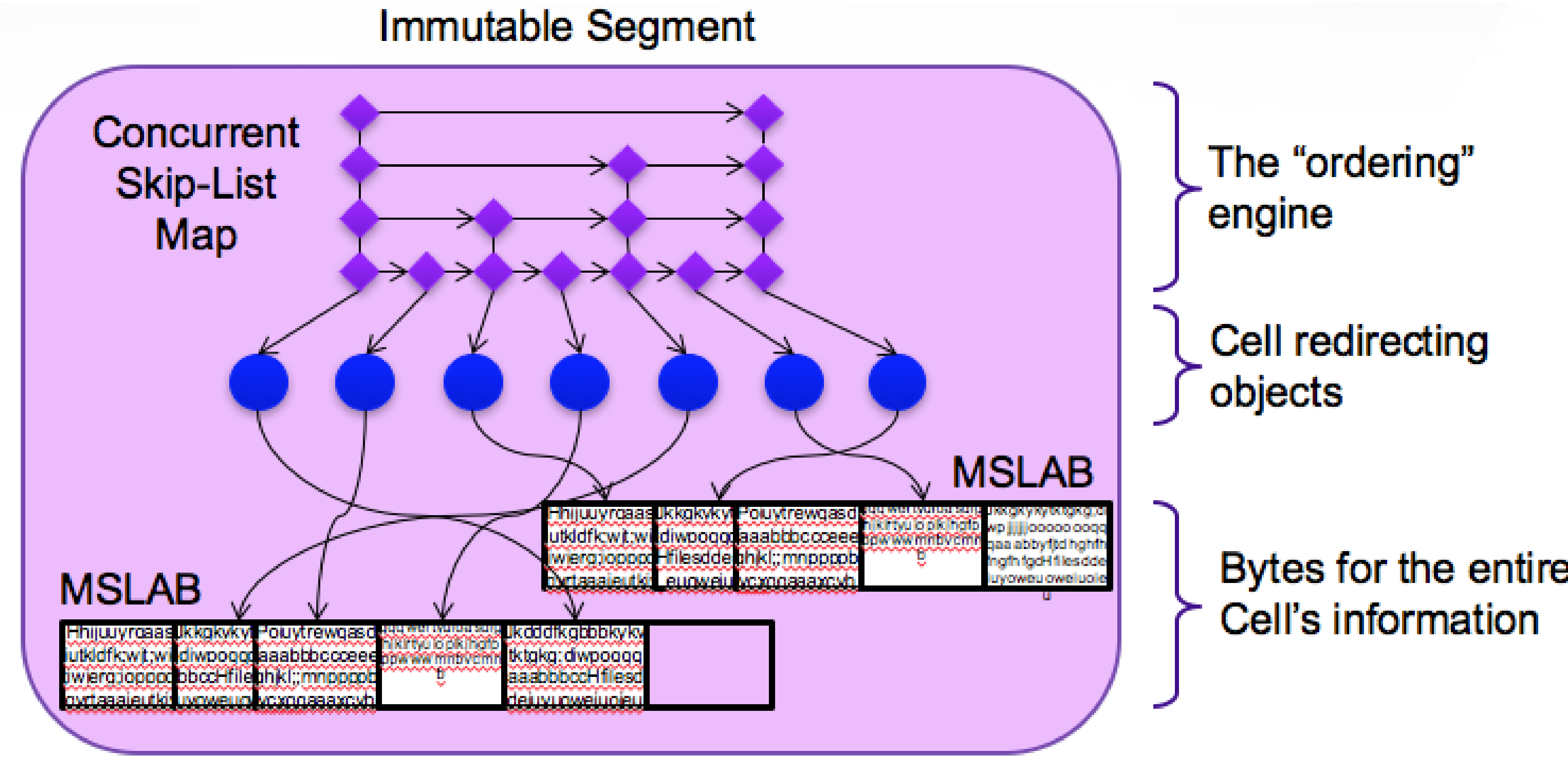
Read Latency (Uniform Distribution)



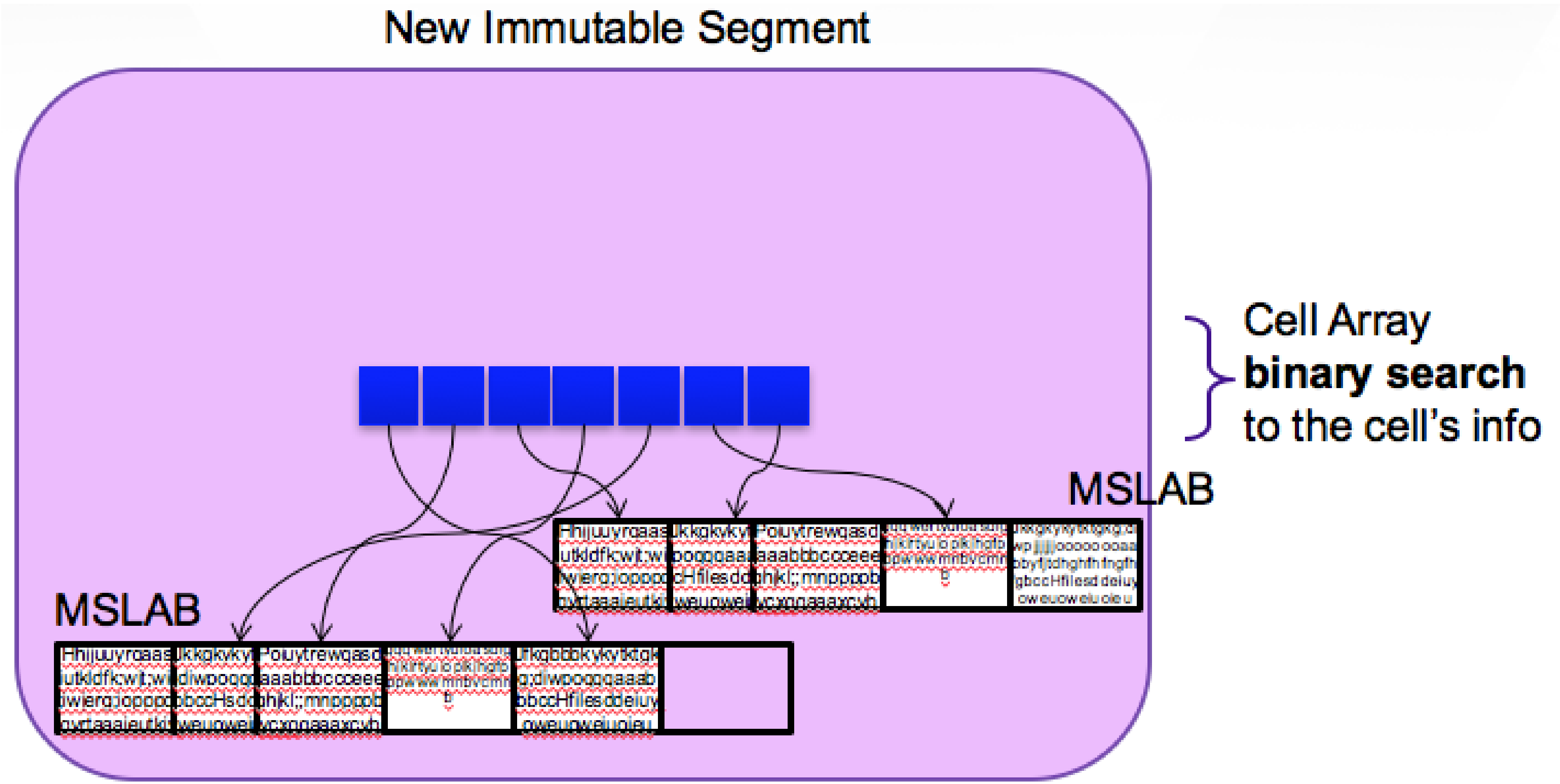
Scan Latency (Zipfian Distribution)



Efficient index representation



No Dynamic Ordering for Immutable Segment

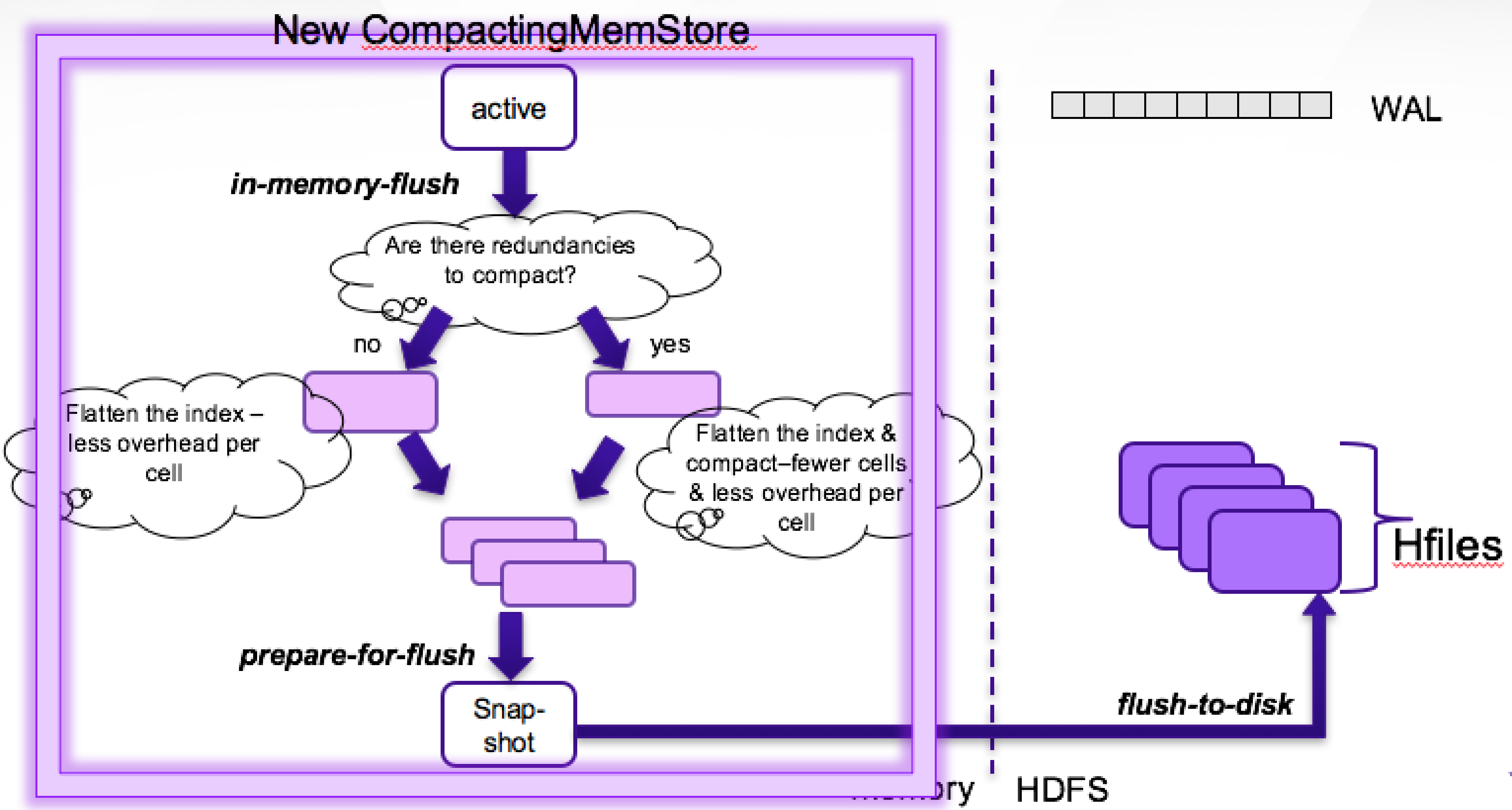


Exploit the Immutability of a Segment after Compaction

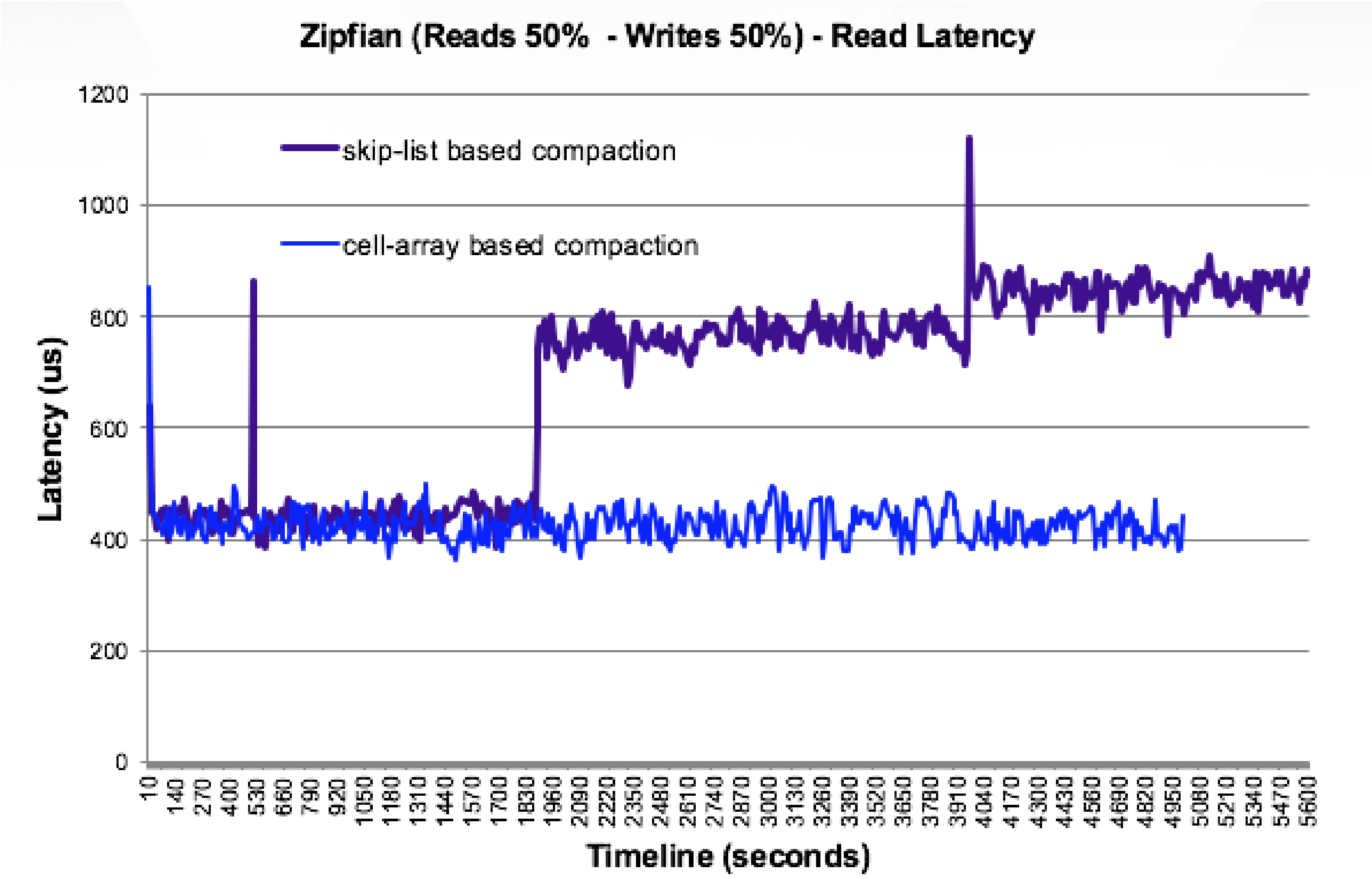
- New Design: Flat layout for immutable segments index
  - Less overhead per cell
  - Manage (allocate, store, release) data buffers off-heap

- **Pros**
  - Better utilization of memory and CPU
  - Locality in access to index
  - Reduce memory fragmentation
  - Significantly reduce GC work

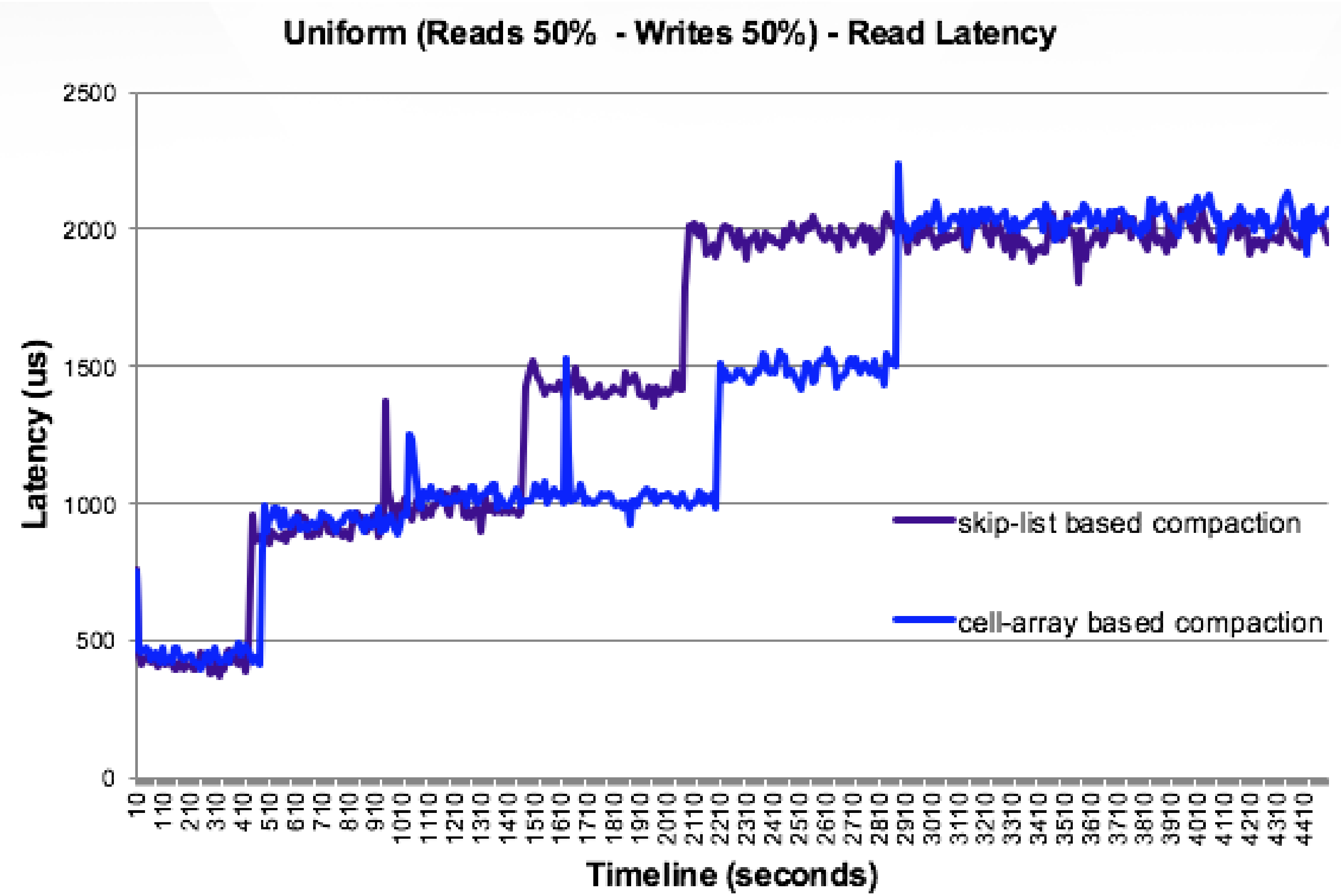
New Design: Putting it all together



Read Latency -- 100Byte Cell



Read Latency -- 1K Cell



Q/A

Thank you.