Australian National University

SPIRIT: Scalable and Persistent In-Memory Indices for Real-Time Search

Adnan Hasnat
Adnan.Hasnat@anu.edu.au

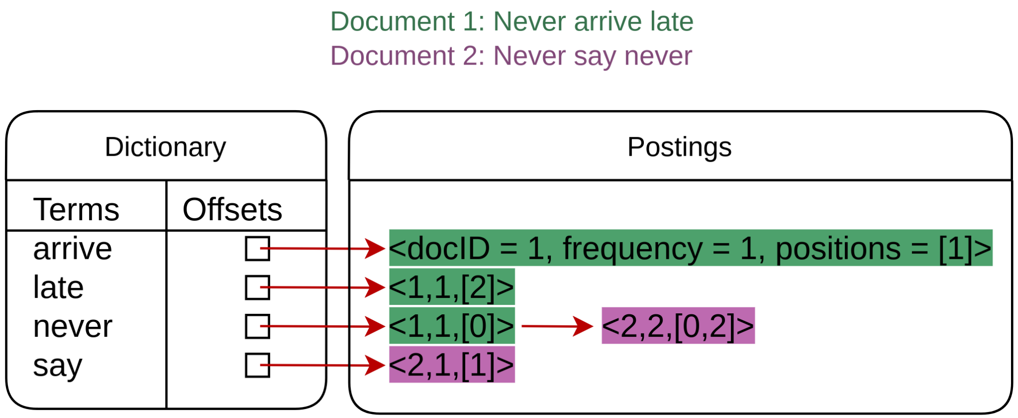
Shoaib Akram
shoaib.akram@anu.edu.au

Full-text search is ubiquitous

- Serves a large and impatient user base

- **Goals:**
  - High query throughput
  - Low average query latency (response time)
  - Low *tail* query latency

Inverted indices power search



- **Two important components**
  - **Dictionary:** for each word/term provides the offset into a postings file
  - **Postings:** IDs of documents in which the term appears and other meta-data
- In traditional search, indices are built offline and **read-optimized** for fast query execution

Real-time search

- "Real time" for search means *indexing happens in real time*
- **Social networking services** like Facebook and X must make new documents instantly searchable
  - Ingestion of new data must be fast
  - It must appear as part of search results upon ingestion
- Examples: ElasticSearch, Twitter EarlyBird – Both based on Lucene

High-level architecture

We split our entire tweet search index into three clusters: a **realtime** cluster indexing all public tweets posted in about the last 7 days, a **protected** cluster indexing all protected tweets for the same timeframe; and an **archive** cluster indexing all tweets ever posted, up to about two days ago.

Real-time search is challenging

- Real-time search poses challenges
  - Need to transform from write-optimized to read-optimized organization quickly
  - Concurrent writes (indexing) and reads (query evaluation)

- Want to serve as many queries as possible from memory to avoid incurring significant latency penalty of accessing storage

- **Problem: DRAM capacity cannot scale to high ingestion rates!**
  - Twitter users create 500 million tweets every day

Real-time search is challenging

- Traditional solutions (Apache Solr and ElasticSearch) retain DRAM latency advantage by keeping segments in DRAM page cache after copy to storage

- This has problems:
  - OS filesystem overhead from accessing the segment commit point is incurred even when the data being accessed is in memory
  - Reformatting data for efficient block device usage adds overhead

- **Alternative proposal**: extend memory capacity using Nonvolatile Memory
  - Direct memory access (DAX) feature avoids filesystem overhead
  - Byte addressability = no reformatting
  - Slower than DRAM by 2x, but much faster still than storage

Our contribution: SPIRIT

- An enterprise search engine with real-time query evaluation as a first principle
  - **Real-time:** Newly ingested document/post/tweet is instantly visible

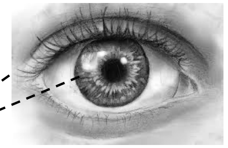
- Uses a **hybrid heap** for hosting inverted indices
  - Volatile (DRAM) heap for fresh ingestion
  - Non-volatile (Intel Optane Persistent Memory) heap for long-term preservation

- Optane NVM serves two roles
  - DRAM capacity expansion (dealing with limited memory)
  - Persistent memory (bypassing the expensive IO/filesystem stack)

Crash resilience & instant restart

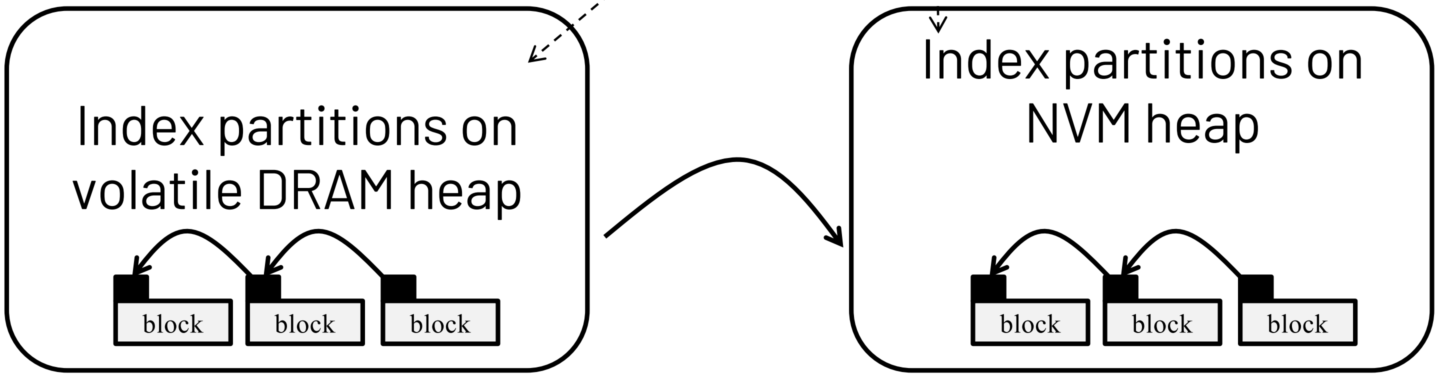
- Existing data-intensive frameworks maintain a large state in memory (OS page cache) with an ***fsync*** every few minutes
- Logs are used for recovery, but some background operations (e.g., merging) can still corrupt the index
- *fsync* is expensive and logs are (sometimes) on the critical path
- Restarting the service is expensive (OS page cache is empty on restart)
- Use NVM to enable better crash resilience (**hindsight:** consistency with NVM is also hard and incurs a performance hit, see paper for details)

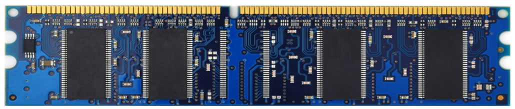
Design: High-Level overview



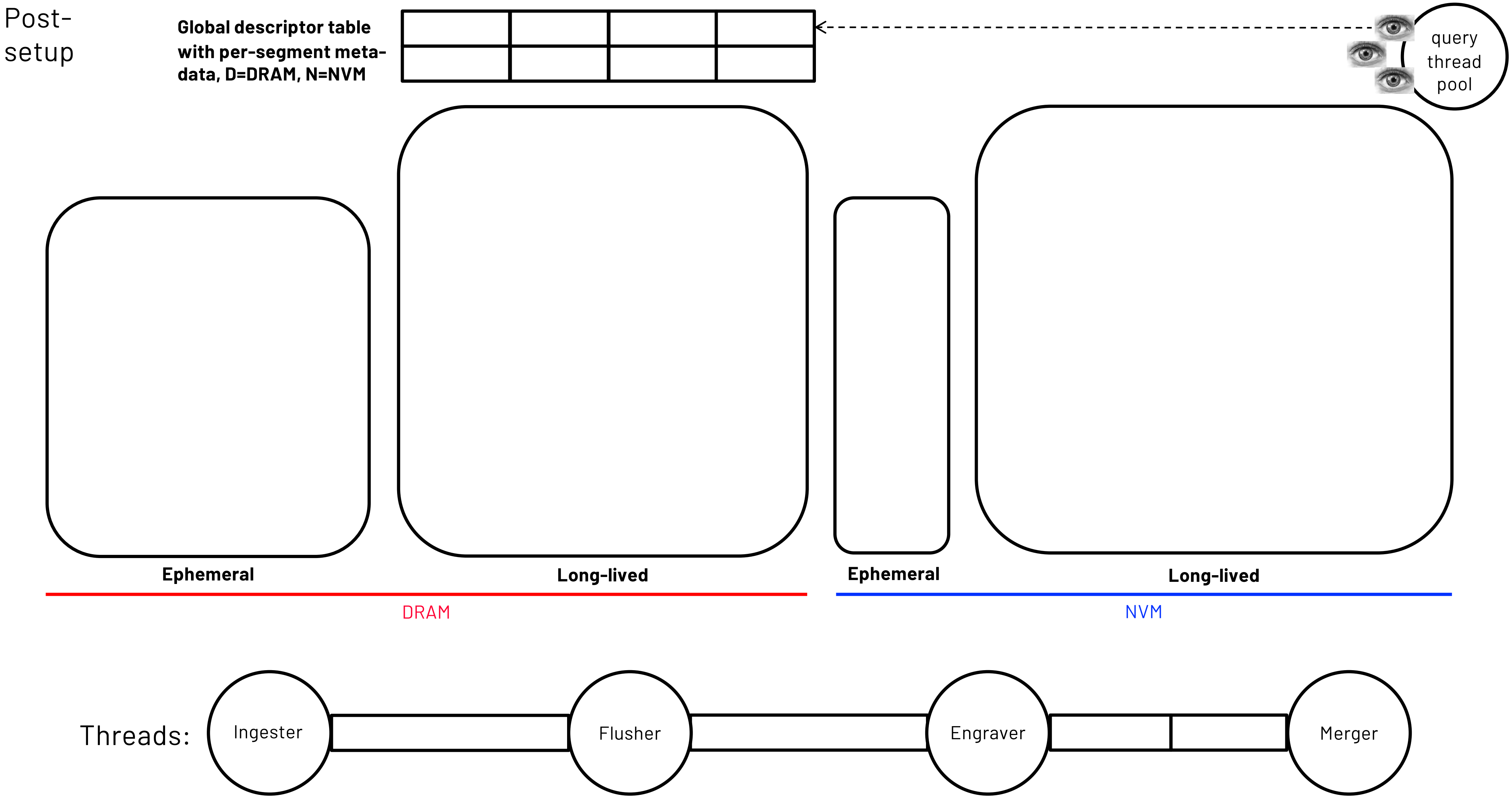
Global descriptor table in DRAM with partition meta-data **(no filesystem calls like Lucene)**

Partition or segment	Address
1: Compressed state	XXXX
2: Fresh state	XXXX
3: Merged state	YYYY
4: Partially merged state	ZZZZ



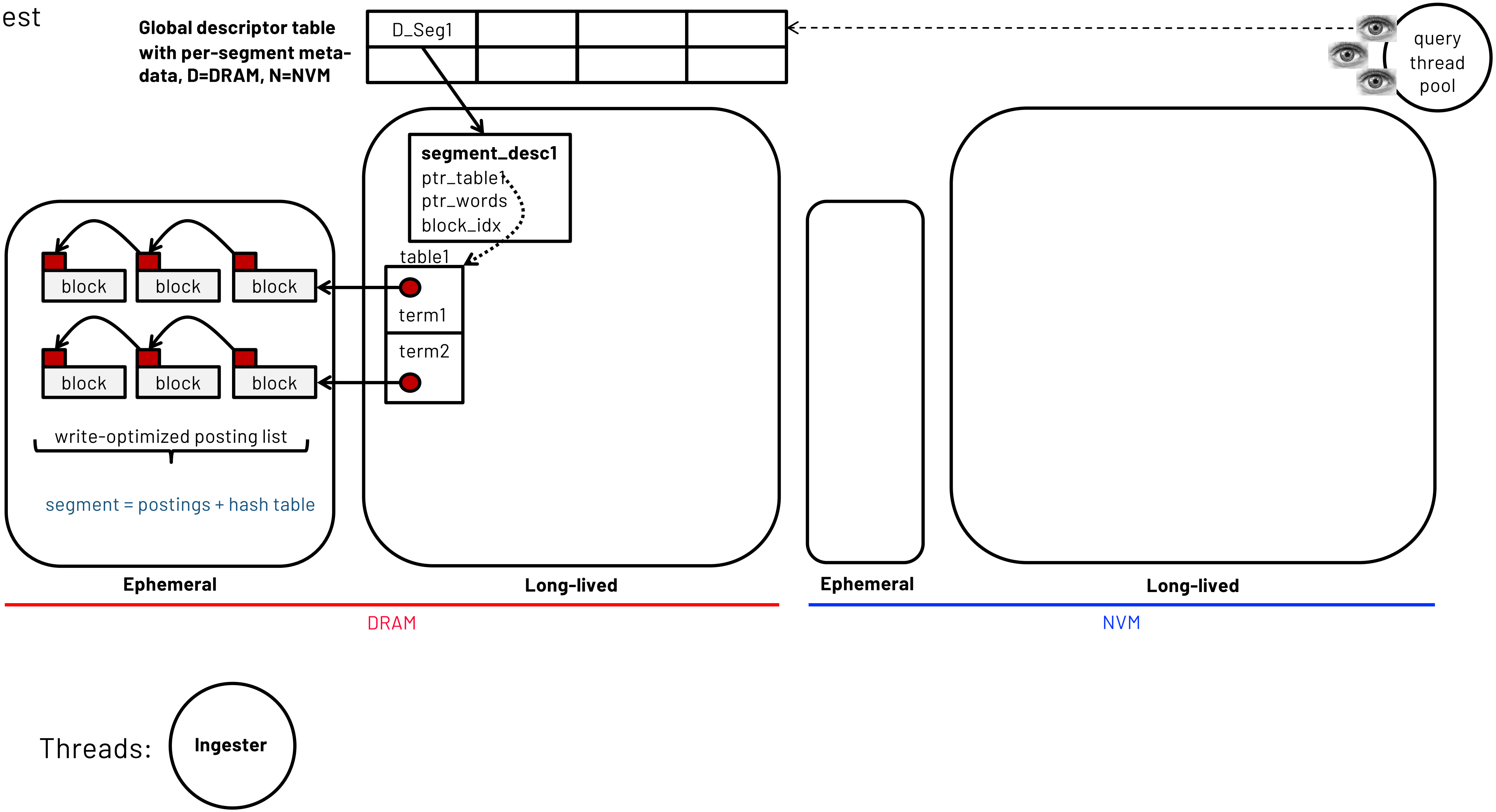


Design in detail

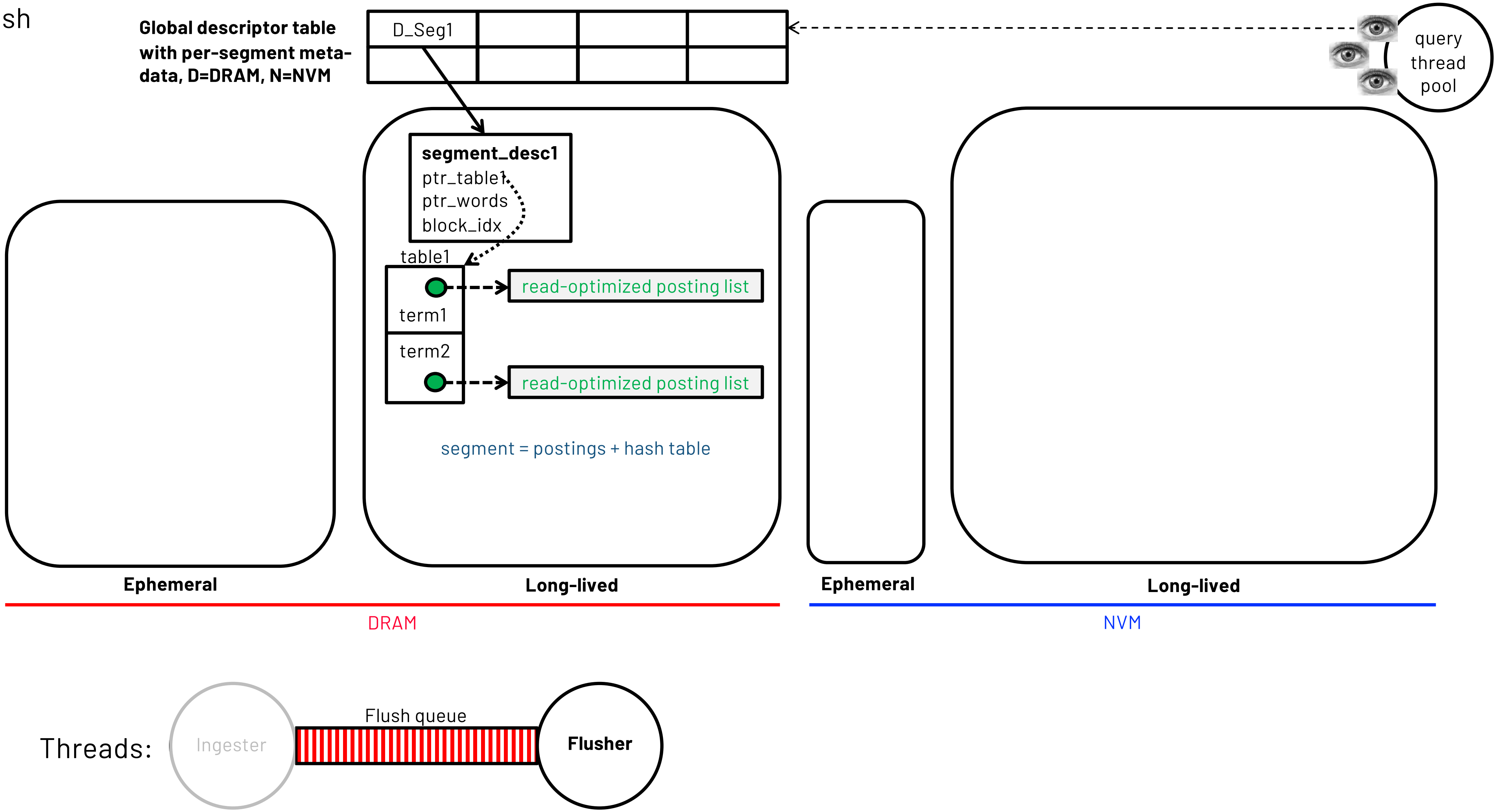


Post-setup
Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
query thread pool
Ephemeral
Long-lived
DRAM
Ephemeral
Long-lived
NVM
Threads:
Ingester
Flusher
Engraver
Merger

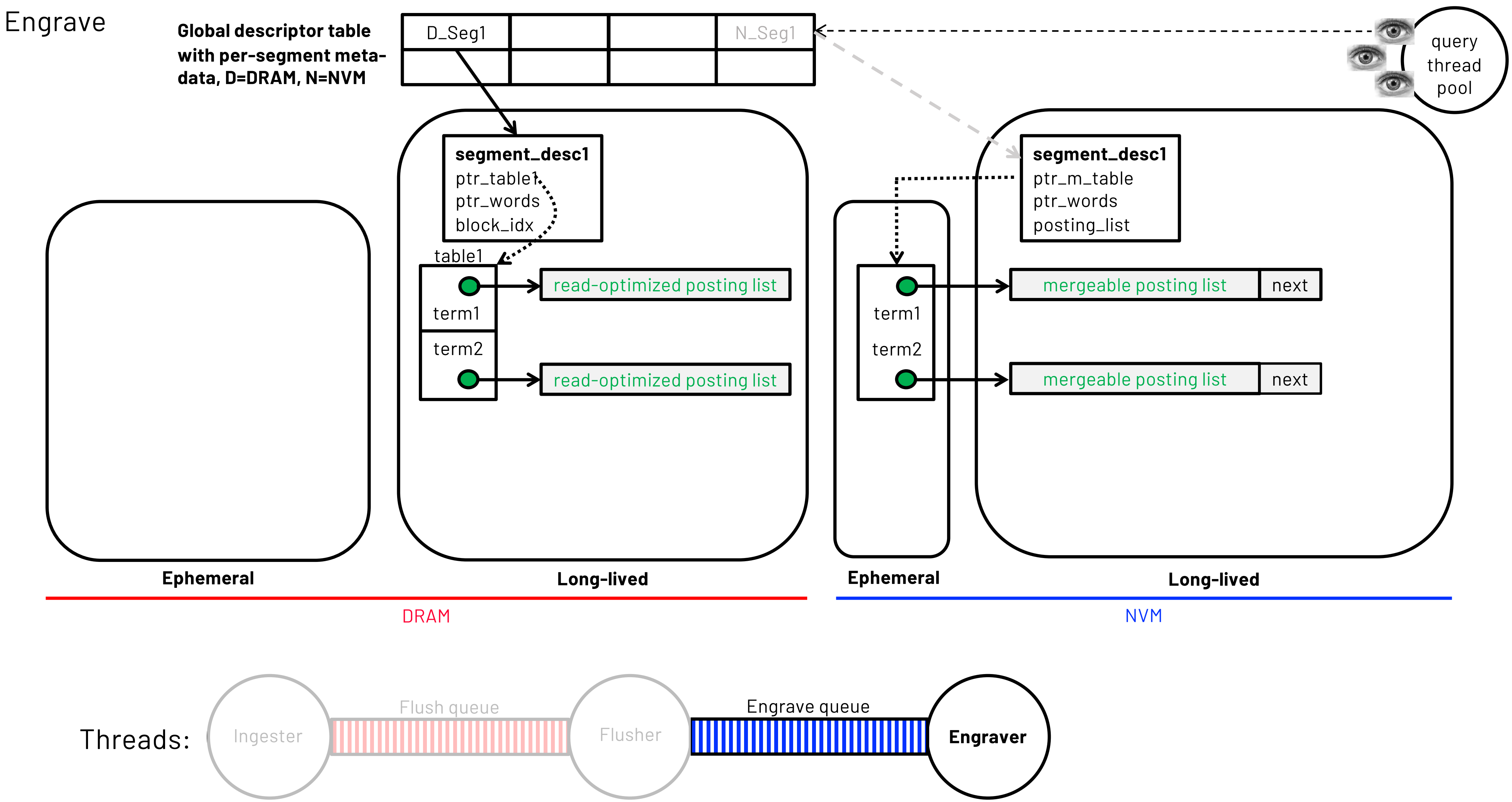
Ingest


Global descriptor table
with per-segment meta-
data, D=DRAM, N=NVM
D_Seg1
query
thread
pool
segment_desc1
ptr_table1
ptr_words
block_idx
table1
block
block
block
term1
term2
block
block
block
write-optimized posting list
segment = postings + hash table
Ephemeral
Long-lived
Ephemeral
Long-lived
DRAM
NVM
Threads:
Ingester

Flush

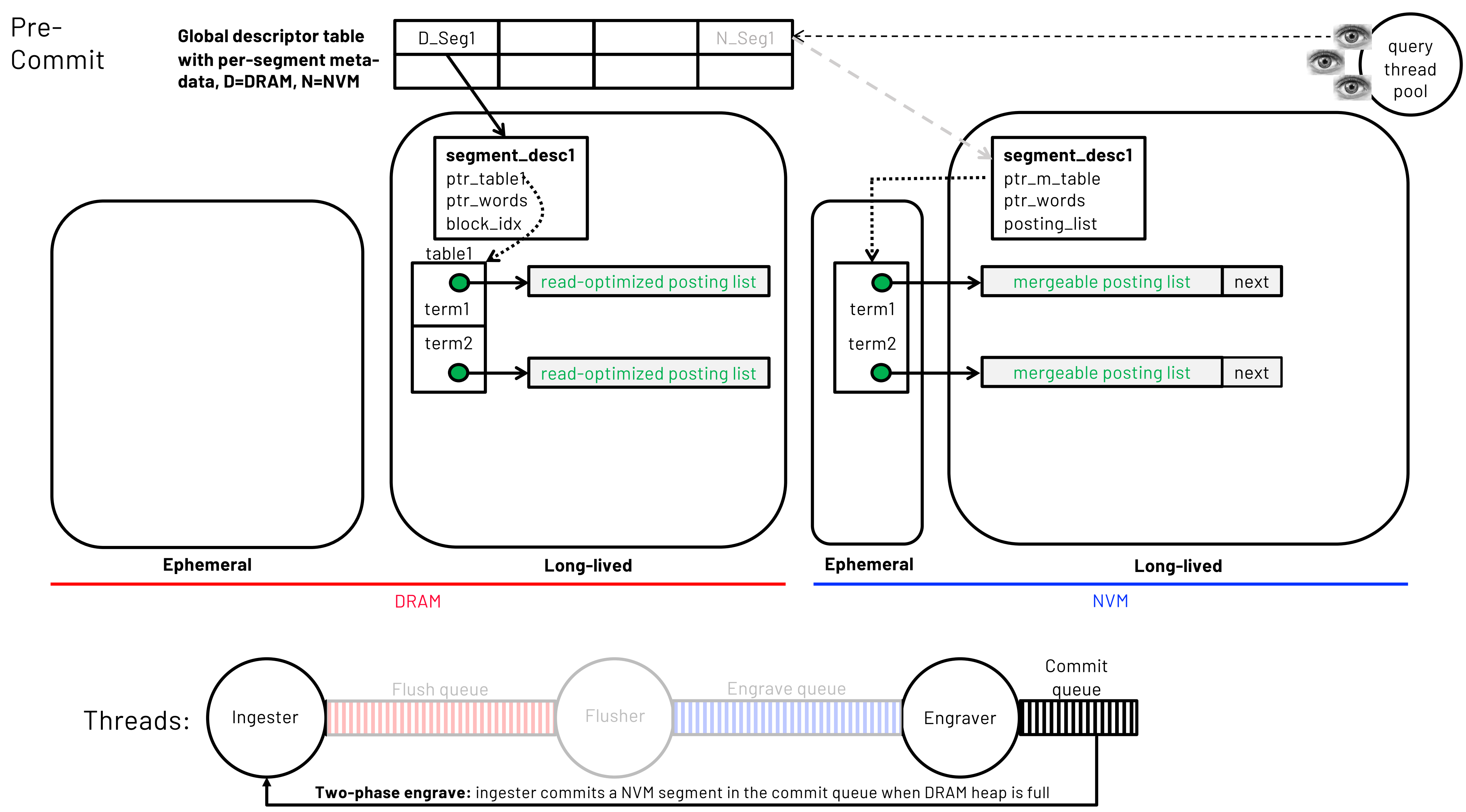


Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
D_Seg1
query thread pool
segment_desc1
ptr_table1
ptr_words
block_idx
table1
term1
term2
read-optimized posting list
read-optimized posting list
segment = postings + hash table
Ephemeral
Long-lived
Ephemeral
Long-lived
DRAM
NVM
Threads:
Ingester
Flush queue
Flusher

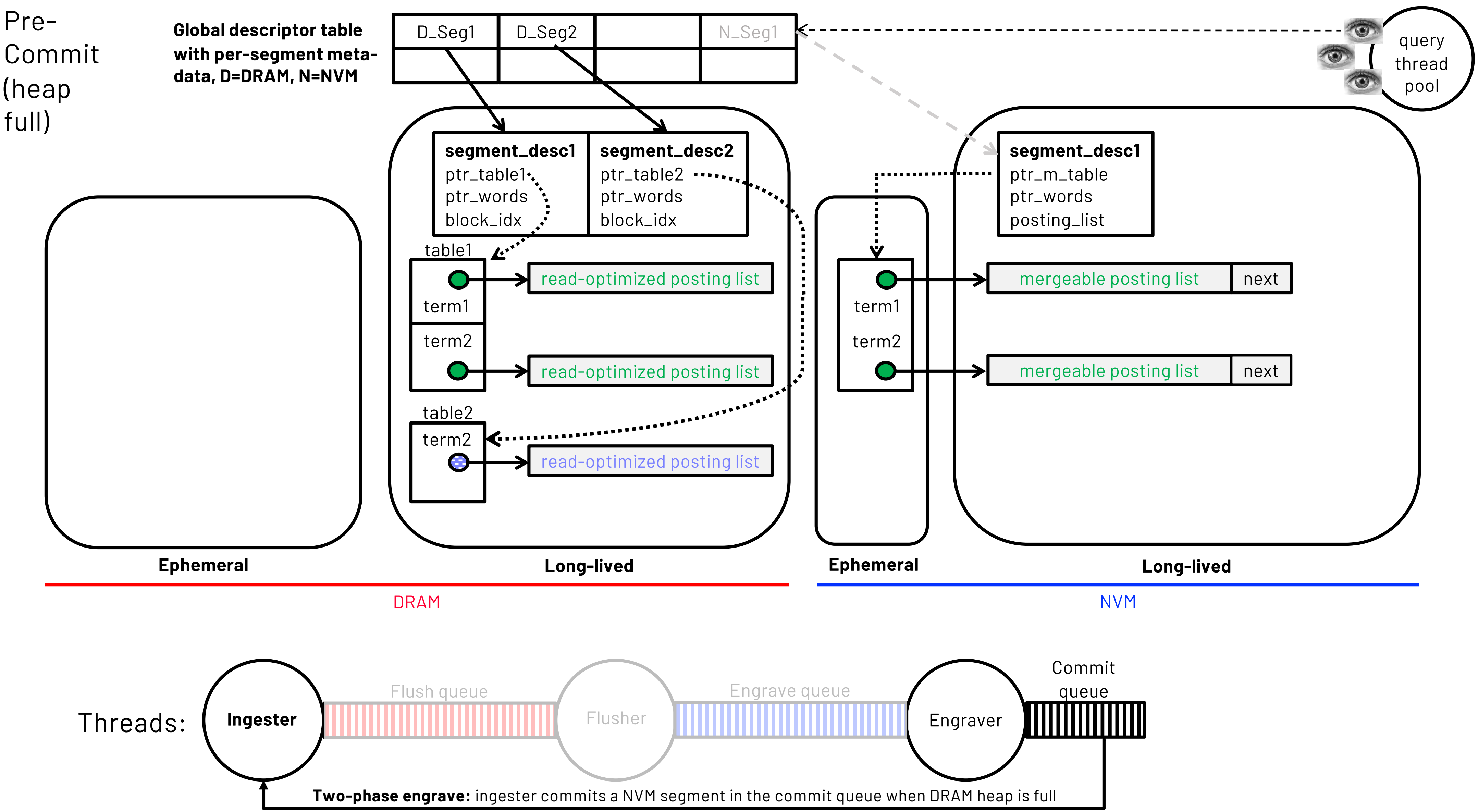

Engrave
Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
D_Seg1
N_Seg1
query thread pool
segment_desc1
ptr_table1
ptr_words
block_idx
table1
term1
term2
read-optimized posting list
read-optimized posting list
segment_desc1
ptr_m_table
ptr_words
posting_list
term1
term2
mergeable posting list
next
mergeable posting list
next
Ephemeral
Long-lived
DRAM
Ephemeral
Long-lived
NVM
Threads:
Ingester
Flush queue
Flusher
Engrave queue
Engraver

Writes to NVM (Engraving)

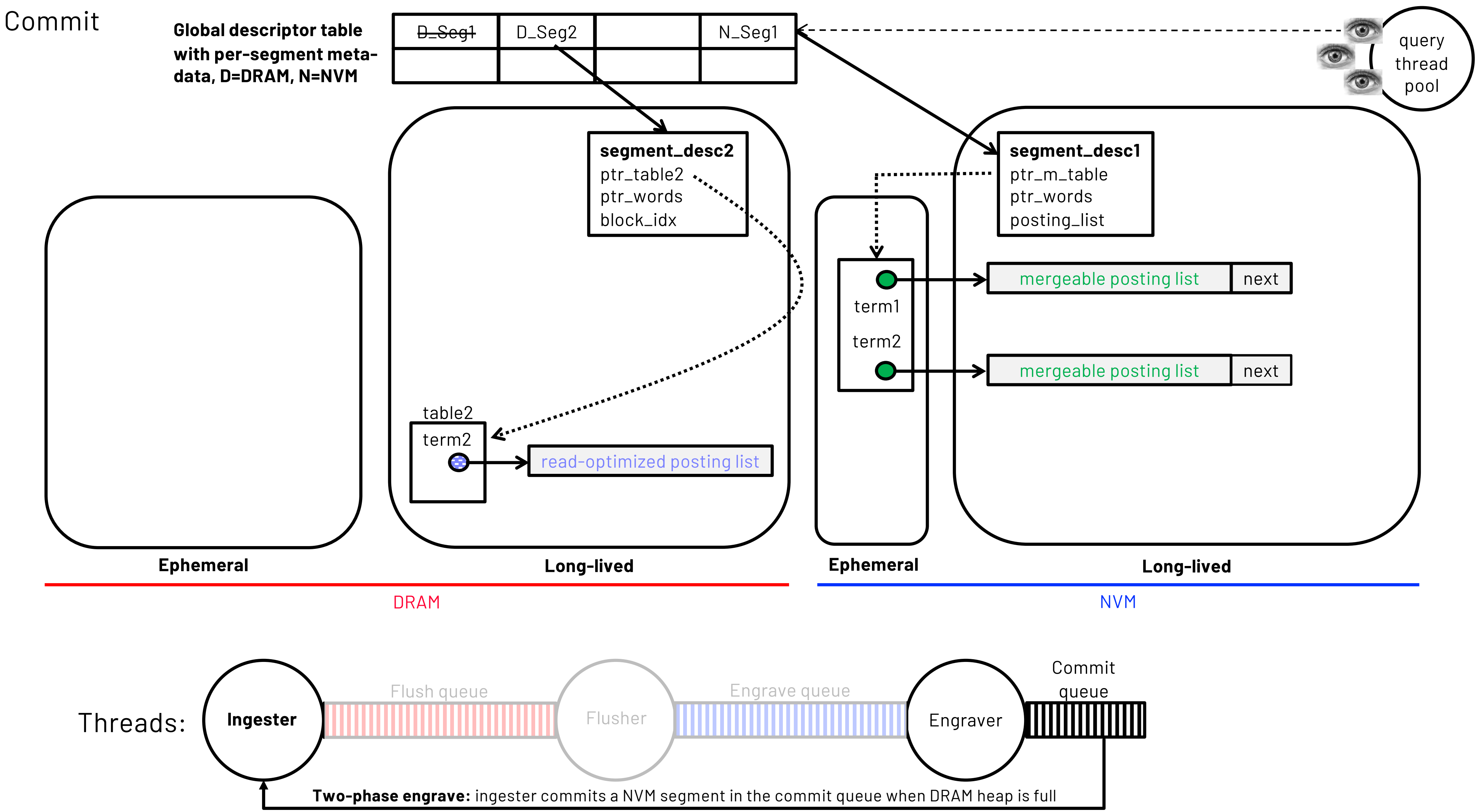
- SPIRIT writes index partitions (segments) to NVM after they are immutable
- Writes to NVM are direct without OS buffering
- Writes are synchronous (calling thread does not return from memory copy until the copy is complete)
- It eases crash consistency
- Queries do not see a massive pause due to device overload during bulk write (such as *fsync*)



Pre-Commit
Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
D_Seg1
N_Seg1
query thread pool
segment_desc1
ptr_table1
ptr_words
block_idx
table1
term1
term2
read-optimized posting list
read-optimized posting list
segment_desc1
ptr_m_table
ptr_words
posting_list
term1
term2
mergeable posting list
next
mergeable posting list
next
Ephemeral
Long-lived
DRAM
Ephemeral
Long-lived
NVM
Threads:
Ingester
Flush queue
Flusher
Engrave queue
Engraver
Commit queue
Two-phase engrave: ingester commits a NVM segment in the commit queue when DRAM heap is full



Pre-Commit (heap full)
Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
D_Seg1
D_Seg2
N_Seg1
query thread pool
segment_desc1
ptr_table1
ptr_words
block_idx
segment_desc2
ptr_table2
ptr_words
block_idx
table1
term1
term2
read-optimized posting list
read-optimized posting list
table2
term2
read-optimized posting list
segment_desc1
ptr_m_table
ptr_words
posting_list
term1
term2
mergeable posting list
next
mergeable posting list
next
Ephemeral
Long-lived
DRAM
Ephemeral
Long-lived
NVM
Threads:
Ingester
Flush queue
Flusher
Engrave queue
Engraver
Commit queue
Two-phase engrave: ingester commits a NVM segment in the commit queue when DRAM heap is full



Commit
Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
D_Seg1
D_Seg2
N_Seg1
query thread pool
segment_desc2
ptr_table2
ptr_words
block_idx
segment_desc1
ptr_m_table
ptr_words
posting_list
term1
term2
mergeable posting list
next
mergeable posting list
next
table2
term2
read-optimized posting list
Ephemeral
Long-lived
Ephemeral
Long-lived
DRAM
NVM
Threads:
Ingester
Flush queue
Flusher
Engrave queue
Engraver
Commit queue
Two-phase engrave: ingester commits a NVM segment in the commit queue when DRAM heap is full

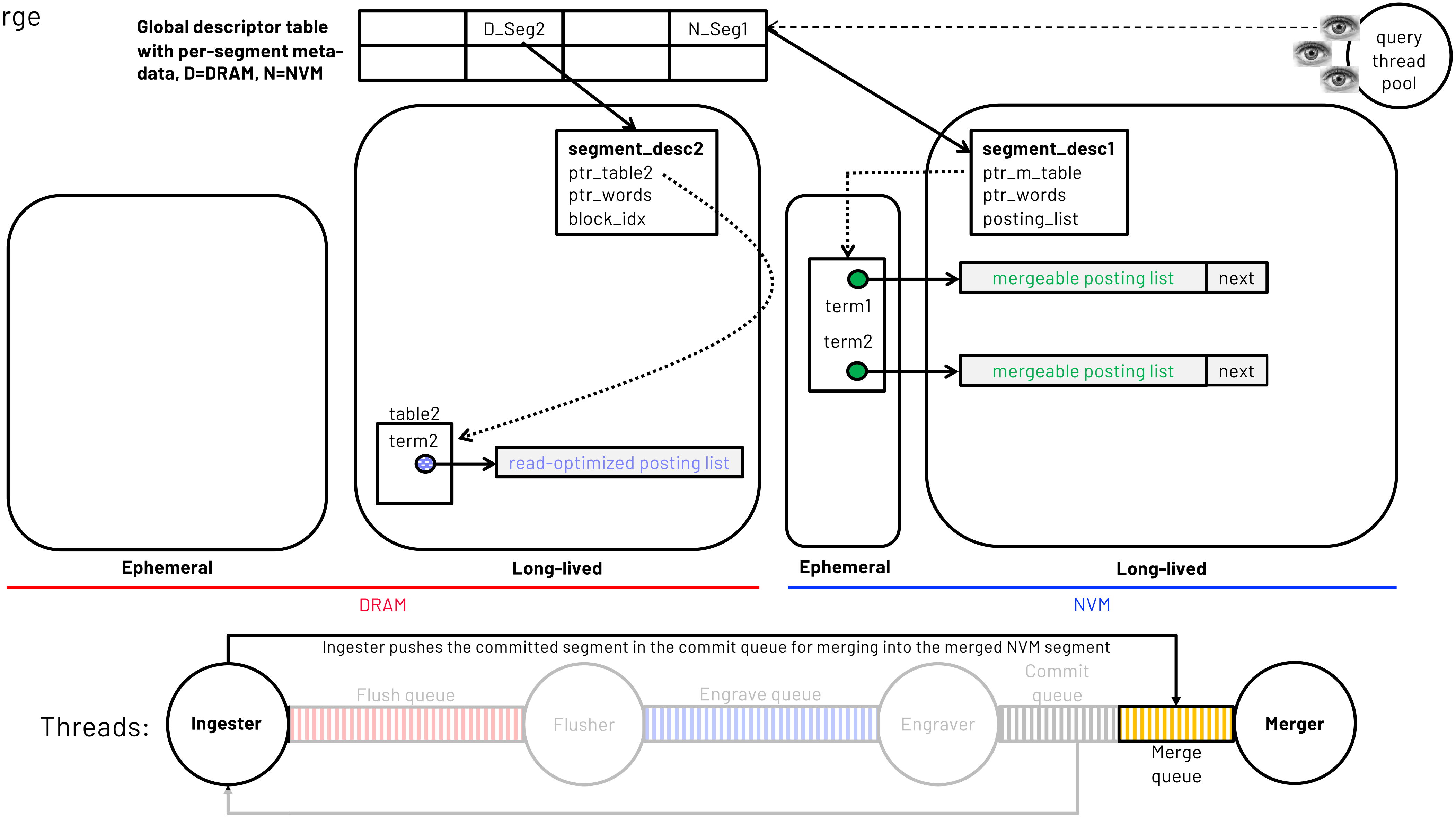
Eager NVM Write – Lazy Pointer Update

- Query evaluators do not access the recently written index partition into NVM
  - NVM is slower than DRAM

- Eventually, SPIRIT updates the segment descriptor to point to NVM copy based on many factor (e.g., running out of DRAM)

- These policies are possible as directing query evaluators to DRAM or NVM segment happens via an in-memory pointer table
  - Lucene has a file (commit point) that stores locations of index partitions
  - So, one set of system calls to access commit point and one set of system calls to access the actual index (hence near-real-time)

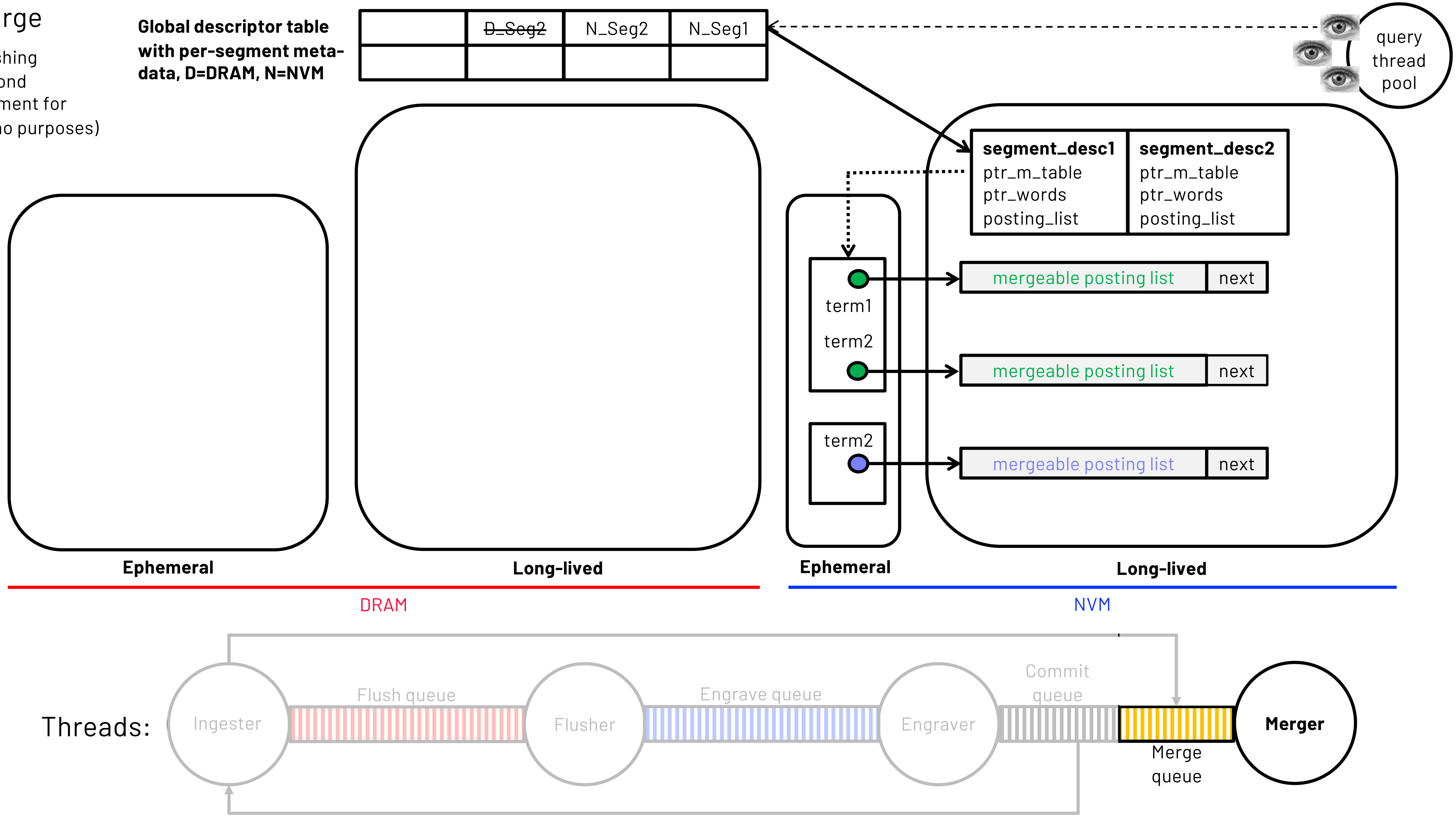
Merge



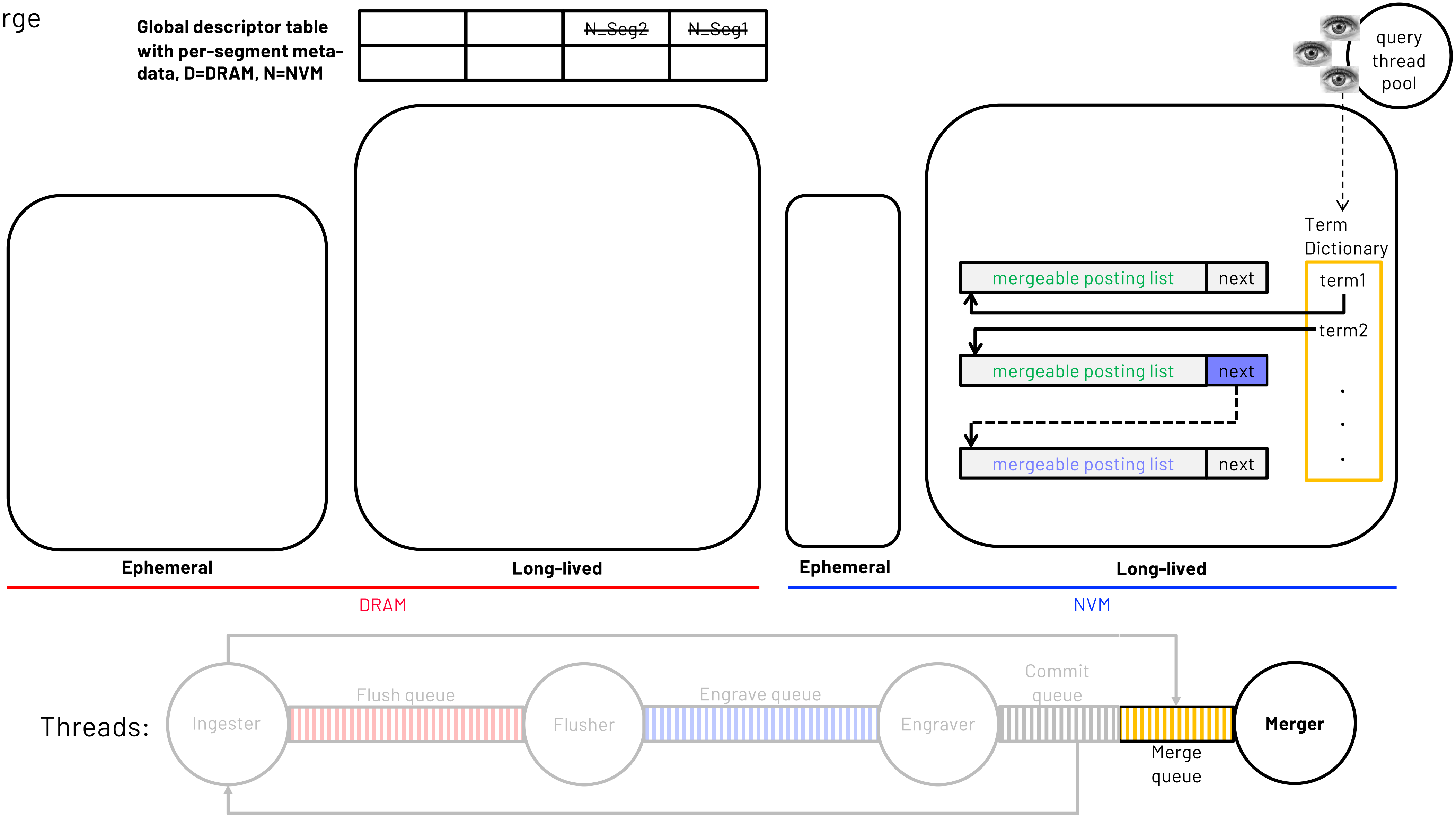
Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
D_Seg2
N_Seg1
query thread pool
segment_desc2
ptr_table2
ptr_words
block_idx
segment_desc1
ptr_m_table
ptr_words
posting_list
term1
term2
mergeable posting list
next
mergeable posting list
next
table2
term2
read-optimized posting list
Ephemeral
Long-lived
Ephemeral
Long-lived
DRAM
NVM
Ingester pushes the committed segment in the commit queue for merging into the merged NVM segment
Threads:
Ingester
Flush queue
Flusher
Engrave queue
Engraver
Commit queue
Merge queue
Merger

Merge

(flushing second segment for demo purposes)

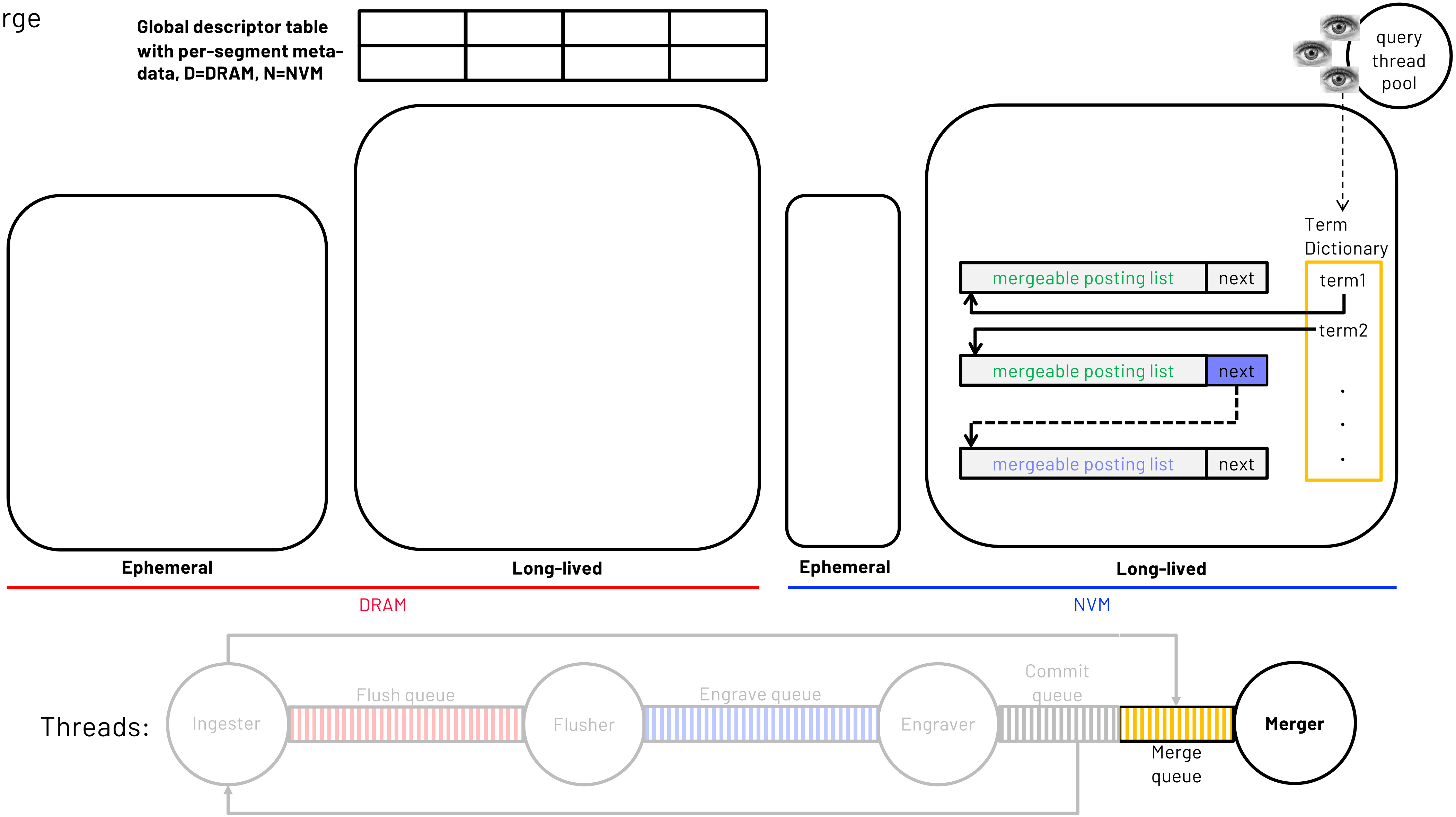


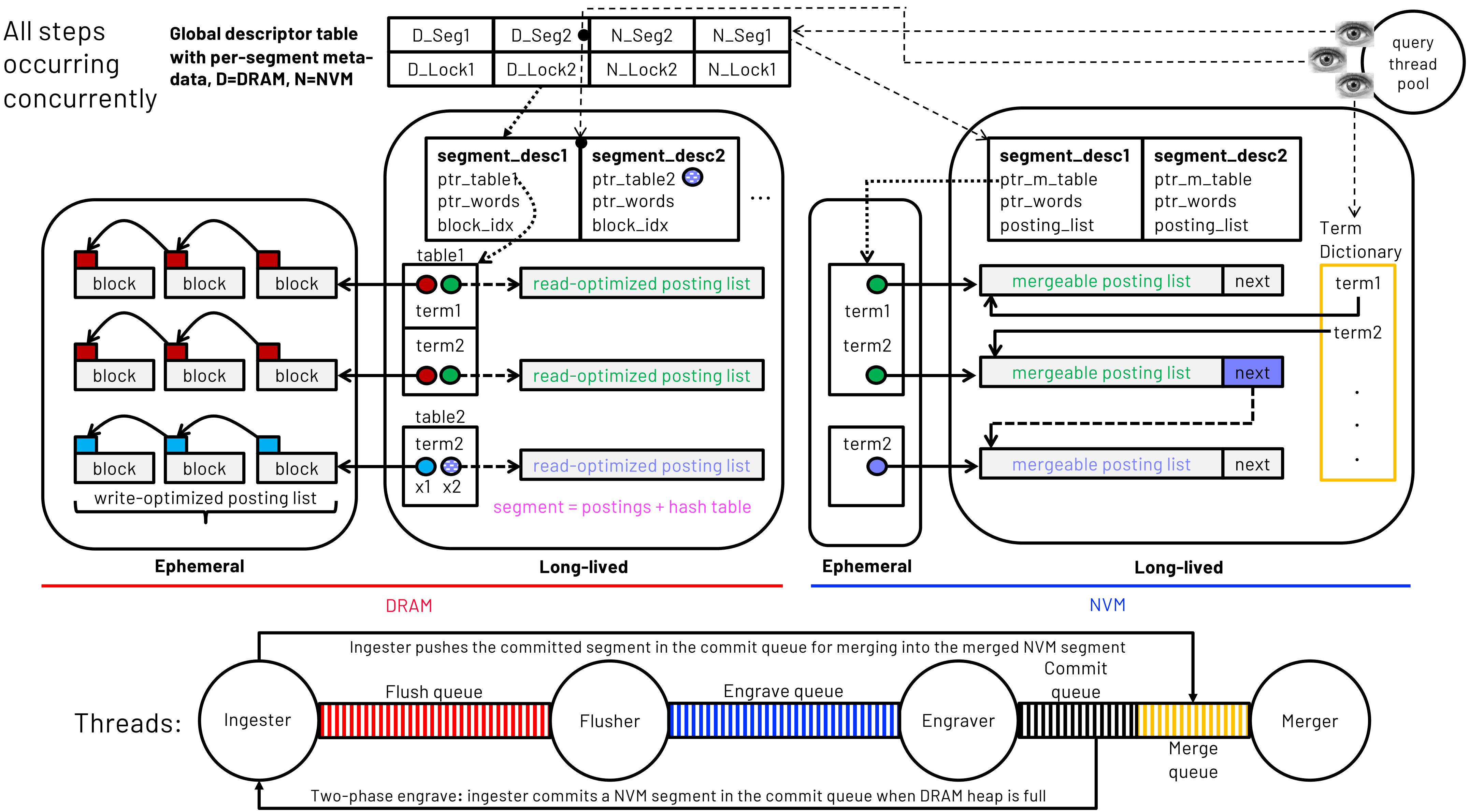
Merge



Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
N_Seg2
N_Seg1
query thread pool
Term Dictionary
term1
term2
mergeable posting list
next
mergeable posting list
next
mergeable posting list
next
Ephemeral
Long-lived
Ephemeral
Long-lived
DRAM
NVM
Threads:
Ingester
Flush queue
Flusher
Engrave queue
Engraver
Commit queue
Merge queue
Merger

Merge


Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
query thread pool
Term Dictionary
term1
term2
mergeable posting list
next
mergeable posting list
next
mergeable posting list
next
Ephemeral
Long-lived
Ephemeral
Long-lived
DRAM
NVM
Threads:
Ingester
Flush queue
Flusher
Engrave queue
Engraver
Commit queue
Merge queue
Merger



All steps occurring concurrently
Global descriptor table with per-segment meta-data, D=DRAM, N=NVM
D_Seg1
D_Seg2
N_Seg2
N_Seg1
D_Lock1
D_Lock2
N_Lock2
N_Lock1
query thread pool
segment_desc1
ptr_table1
ptr_words
block_idx
segment_desc2
ptr_table2
ptr_words
block_idx
...
segment_desc1
ptr_m_table
ptr_words
posting_list
segment_desc2
ptr_m_table
ptr_words
posting_list
Term Dictionary
term1
term2
block
block
block
block
block
block
block
block
block
write-optimized posting list
table1
term1
term2
table2
term2
x1 x2
read-optimized posting list
read-optimized posting list
read-optimized posting list
segment = postings + hash table
term1
term2
term2
mergeable posting list
next
mergeable posting list
next
mergeable posting list
next
Ephemeral
Long-lived
Ephemeral
Long-lived
DRAM
NVM
Threads:
Ingester pushes the committed segment in the commit queue for merging into the merged NVM segment
Ingester
Flush queue
Flusher
Engrave queue
Engraver
Commit queue
Merge queue
Merger
Two-phase engrave: ingester commits a NVM segment in the commit queue when DRAM heap is full

Design principles of SPIRIT (1)

- **Make indexed data instantly visible**
  - No expensive transformation because everything is memory encourages instant visibility
- **Operate nonstop from a user-space hybrid memory heap**
  - Expensive kernel entry points that prohibit real-time response are eliminated
  - No block storage IO. No filesystem calls. No external memory allocators
- **Perform macro-management**
  - Minimum locking
  - Many operations are performed in bulk (like freeing heap memory)

Design principles of SPIRIT (2)

- **Persist proactively but control visibility**
  - Move index segments to NVM instantly (direct, byte-addressable writes), but delay visibility until DRAM is under pressure

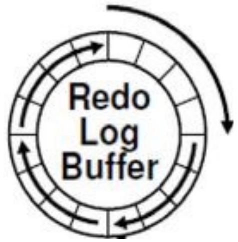
- **Maximize memory economy**
  - In-place merging (enabled by NVM)
  - Metadata sharing

- **Allow multiple operational modes**
  - Volatile and graceful shutdown
  - Crash-consistent indexing (beware the performance hit!)

Crash consistency

- **Experience:** NVM consistency is harder than disk (many byte-granular updates)

- Requires a combination of atomic operations, cache line flushes,
- fences, and undo/redo logs



- Can recover all writes to NVM including partial NVM writes and partially merged partitions

- DRAM partitions are unrecoverable but log enough information to rebuild the index

- Stronger consistency guarantees but slows down in-place merging significantly (future work)

Query evaluation

- No filesystem operations to access the index
- Up to date DRAM and NVM partitions visible via pointer indirection
- Query evaluator requires minor changes to traverse DRAM and NVM segments
- Minimum locking overhead due to concurrent indexing
- Query caching for frequently encountered queries

Methodology

- Implementation in C++, using Intel PMDK API to access NVM

- Benchmarking datasets:
  - Indexing dataset: Wikipedia English corpus. 1M/5M/10M docs, clipped to 1 KB each
  - Query dataset: Generated from top 50K terms ranked on occurrence in the corpus as provided by luceneutil, classified by frequency (low/medium/high). Includes single term (L, M, H) and double term (LL, MM, HH).

- SPIRIT generally run with **concurrent indexing/querying**: queries run constantly while last 20% of docs are ingested.

Methodology

- **SPIRIT parameters varied in comparisons:**
  - DRAM heap size relative to total index size
    - Loose (L): 100%
    - Moderate (M): 55%
    - Tight (T): 15%
  - Persistence modes:
    - Volatile mode (V)
    - Graceful shutdown mode (G)
    - Crash consistent mode (C)

Methodology

- Lucene configs for comparison with SPIRIT:
  - **NRT**: Near-Real Time, refreshes reader to ingest new docs at interval. Index on DRAM, unlimited DRAM provisions. Remaining configs have a static index.
  - **DAX**: Off-heap index on NVM (with DAX), with DRAM as heap.
  - **NODAX/SSD**: Off-heap index on NVM (without DAX) and SSD respectively, with DRAM as heap and page cache. *This requires filesystem access to segments.*
  - **DPF**: On-heap index using Lucene's Direct Postings Format (DPF), with heap backed by NVM and DRAM provisions created as page cache
- Use best practices to try mitigate the effects of Lucene's managed runtime
- Total DRAM provisions matched in comparisons to SPIRIT

Evaluation system details

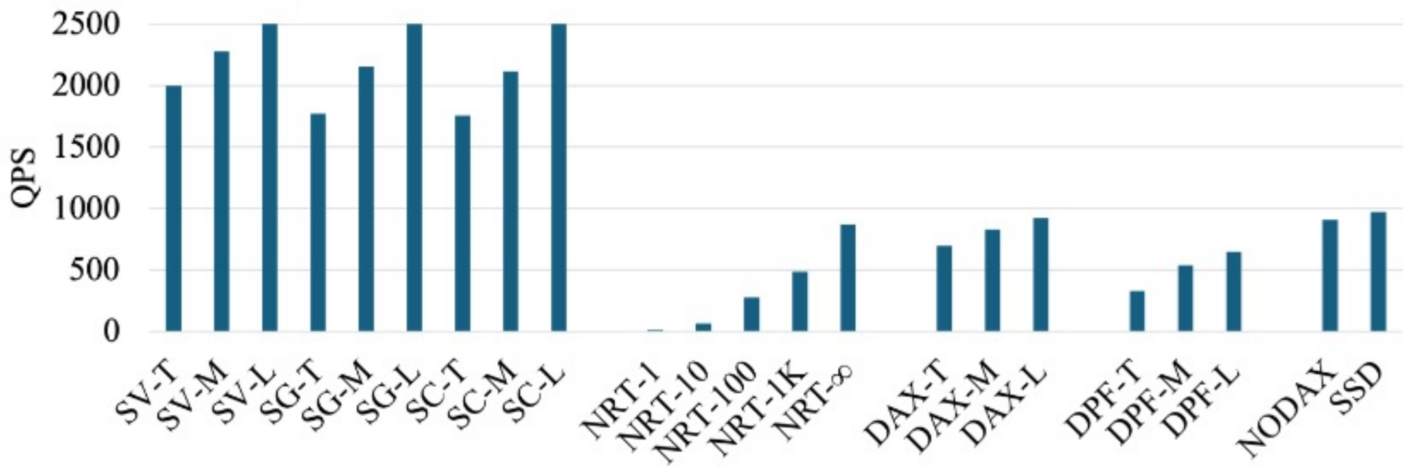
- Some experiments use small subset of total DRAM/NVM capacity; this is for tractability purposes, and key findings were validated with larger datasets.

System	
Operating System	Ubuntu 18.04.1 Linux OS (5.4.0 kernel)
Hardware	Dell PowerEdge R740 Server
Processor	
Processors	Intel Xeon Gold 6252N
Number of cores	48 physical cores (96 logical)
Core frequency	2.3 GHz
Issue width	4-wide
ROB size	128 entries
Branch predictor	hybrid local/global predictor
Max. outstanding	48 loads, 32 stores, 10 L1-D misses

DRAM	
Capacity	400 GB
Bus frequency	800 MHz (DDR 1.6 GHz)
Bus width	64 bits
Channels	6
Ranks	1 rank/channel
Banks	8 banks/rank
NVM	
Capacity	1.5 TB
Hardware	Intel Optane Persistent Memory
SSD	
Capacity	1 TB
Hardware	3.5-Inch, Seagate, SATA (6 Gbps)

Throughput comparisons

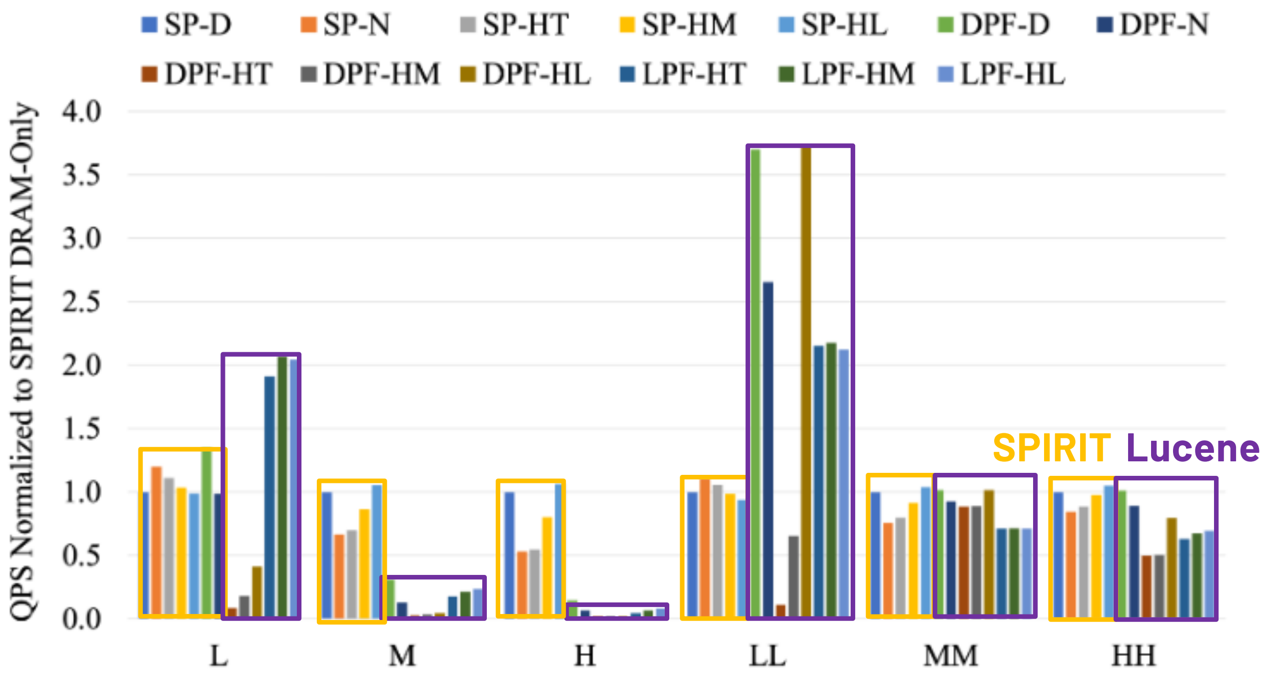
- Harmonic mean of QPS across workloads (higher is better)
- Crash consistency modes have negligible impact on QPS
- **SPIRIT achieves higher average throughput over all Lucene modes**



Throughput comparisons

- More detailed breakdown shows SPIRIT only underperforms for L/LL queries

Note:
LPF here refers to Lucene DAX config



Latency comparisons

- NRT Lucene performs very poor for both average and tail latency
  - **Penalty of filesystem operations incurred by reading new segments is significant**

	L				M				H				LL				MM				HH			
	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg
SP-C-T	67	91	105	65	96	172	247	105	258	1854	2197	425	114	177	210	111	289	511	719	301	1609	8003	11832	2410
SP-C-M	45	62	71	45	65	116	167	71	179	1074	1471	288	78	121	143	77	217	422	604	232	1436	6421	10476	2125
SP-C-L	28	36	41	28	41	75	108	45	126	1007	1109	211	48	71	87	47	158	331	509	172	1277	5549	9580	1868
NRT-1	42815	306709	542697	83717	45131	297782	477702	82816	63795	377984	577864	111173	47487	318163	545872	89286	51384	322060	556774	92507	61659	384500	564877	112931
NRT-10	358	53637	242252	10226	671	56869	281596	11354	2703	115079	362296	21704	811	61355	292364	12055	1420	74535	315342	14356	3976	120219	411704	21891
NRT-100	109	340	12849	1305	331	1028	29172	2028	1772	13164	80319	6862	230	720	25458	1854	597	1249	25846	2190	2822	9592	56248	7504
NRT-1K	67	154	223	229	208	663	1396	602	1663	12489	39404	5336	127	287	573	351	329	779	1346	769	2441	7708	16988	5114
NRT-∞	12	47	88	19	127	546	1044	219	1526	12086	35025	3456	27	56	108	33	185	515	830	225	2163	7011	14286	2930

Latencies in microseconds

Latency comparisons

- Latency breakdown again shows SPIRIT underperforms only for L/LL queries

	L				M				H				LL				MM				HH			
	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg
SP-C-T	67	91	105	65	96	172	247	105	258	1854	2197	425	114	177	210	111	289	511	719	301	1609	8003	11832	2410
SP-C-M	45	62	71	45	65	116	167	71	179	1074	1471	288	78	121	143	77	217	422	604	232	1436	6421	10476	2125
SP-C-L	28	36	41	28	41	75	108	45	126	1007	1109	211	48	71	87	47	158	331	509	172	1277	5549	9580	1868
DAX-T	11	51	102	19	157	713	1500	290	1884	24902	45781	5024	24	54	98	27	214	608	947	241	2252	8102	24595	3029
DAX-M	11	53	105	19	156	694	1446	248	1776	14349	44533	3783	25	56	100	27	223	625	976	245	2245	7661	14616	2918
DAX-L	11	53	110	17	155	701	1232	238	1462	12373	25403	3062	26	57	102	27	233	642	1005	266	2344	7660	13894	2882
NODAX	8	43	87	16	134	614	1395	205	1507	10882	30731	3137	19	47	102	24	232	661	1069	264	2479	7878	12702	2964
SSD	9	45	92	16	132	619	1424	204	1301	8667	26119	2663	19	46	93	24	229	659	1114	259	2532	7803	12257	2996

Latencies in microseconds

Latency comparisons

- Crash consistency incurs a latency penalty, especially when using tighter heaps
  - Attributable to additional NVM bandwidth contention from logging

- Graceful shutdown mode has no difference on the other hand

	L				M				H				LL				MM				HH			
	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg	P50	P95	P99	Avg
SP-V-T	26	41	52	27	59	137	211	68	228	1834	2126	398	42	85	116	43	215	426	599	226	1467	7421	11227	2245
SP-V-M	26	38	46	27	48	101	152	55	166	1061	1446	277	42	77	99	43	183	378	559	195	1369	6361	10502	2034
SP-V-L	26	36	42	27	41	75	109	46	125	1000	1094	213	42	72	85	43	159	334	507	174	1281	5601	9697	1878
SP-C-T	67	91	105	65	96	172	247	105	258	1854	2197	425	114	177	210	111	289	511	719	301	1609	8003	11832	2410
SP-C-M	45	62	71	45	65	116	167	71	179	1074	1471	288	78	121	143	77	217	422	604	232	1436	6421	10476	2125
SP-C-L	28	36	41	28	41	75	108	45	126	1007	1109	211	48	71	87	47	158	331	509	172	1277	5549	9580	1868

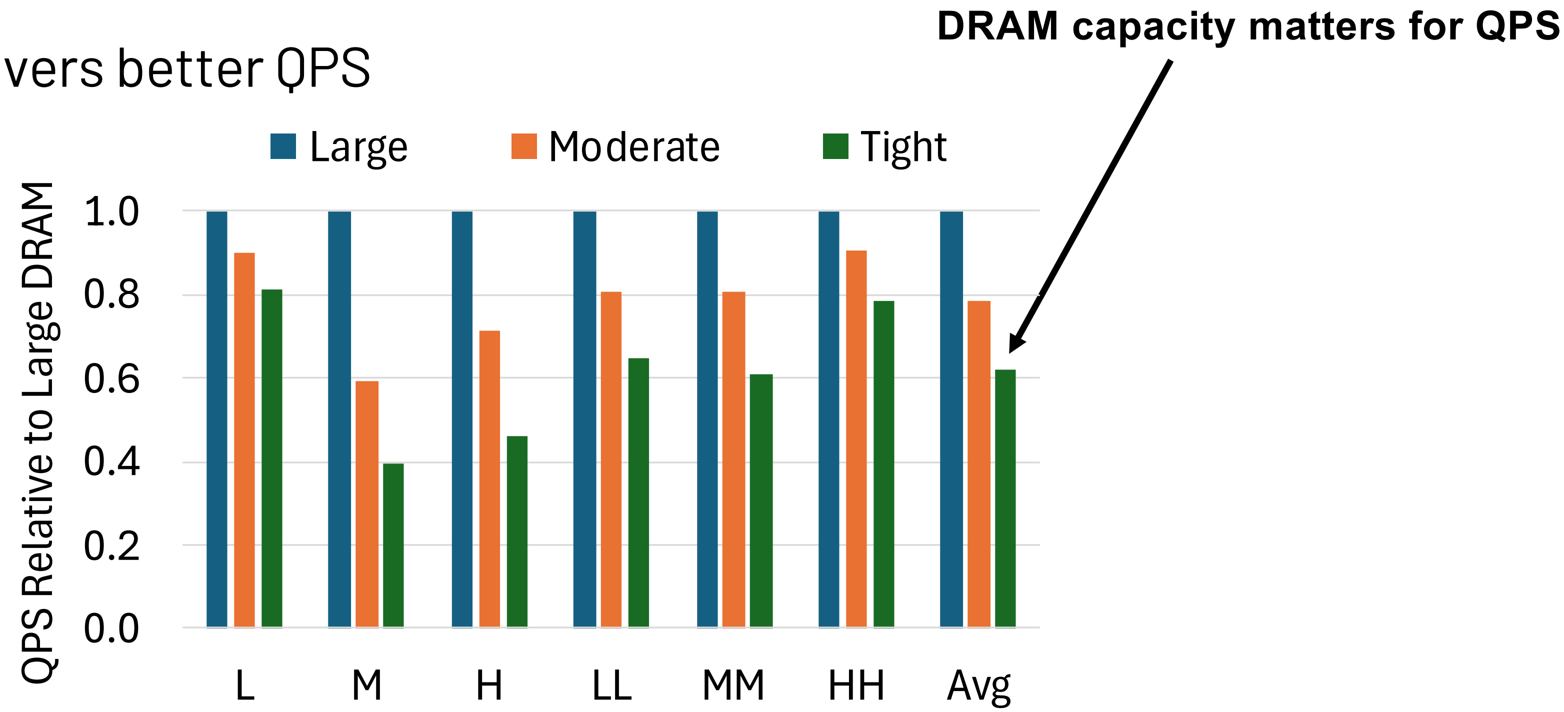
Latencies in microseconds

Indexing performance

- In volatile mode, SPIRIT's indexing is **2.5x** faster than Lucene's
  - **6.74x** faster merging
  - **3.78x** faster committing to persistent medium
- Graceful shutdown SPIRIT is negligibly slower, and remains faster than Lucene
- **However**, full crash consistency slows down SPIRIT segment merging significantly
  - Difficult to control granularity of logging during merging, as undoing/redoing a partial merge on a segment-granularity is intractable

Performance scales with DRAM capacity

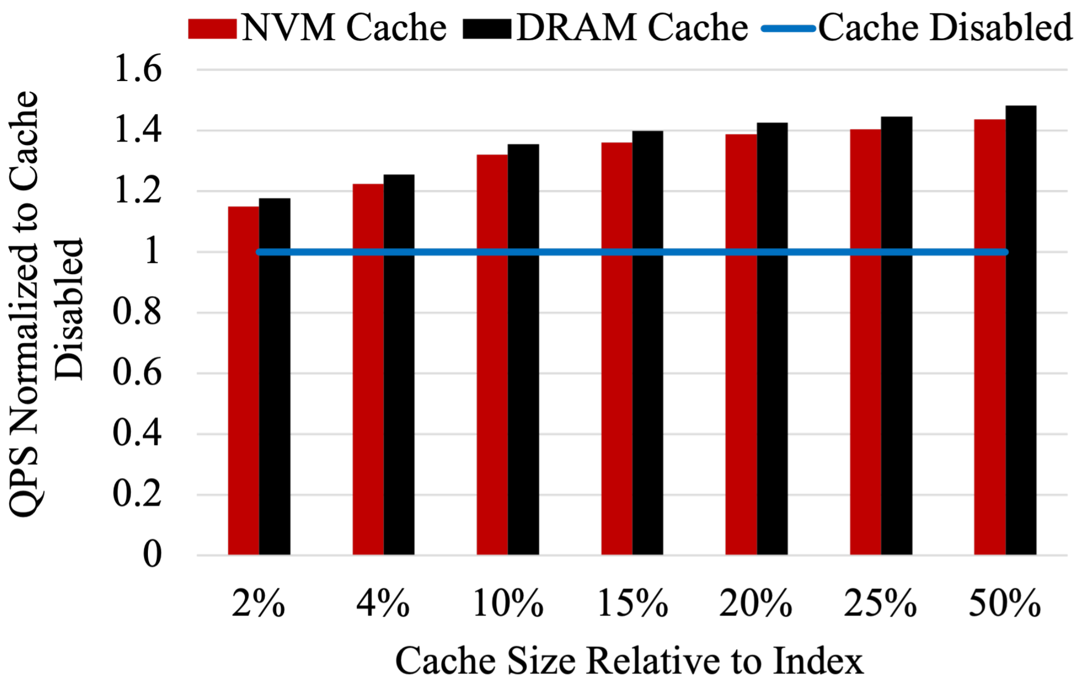
- Large DRAM delivers better QPS



- NVM bandwidth is limited lowering QPS when DRAM is scarce

Query caching helps even using NVM as cache medium

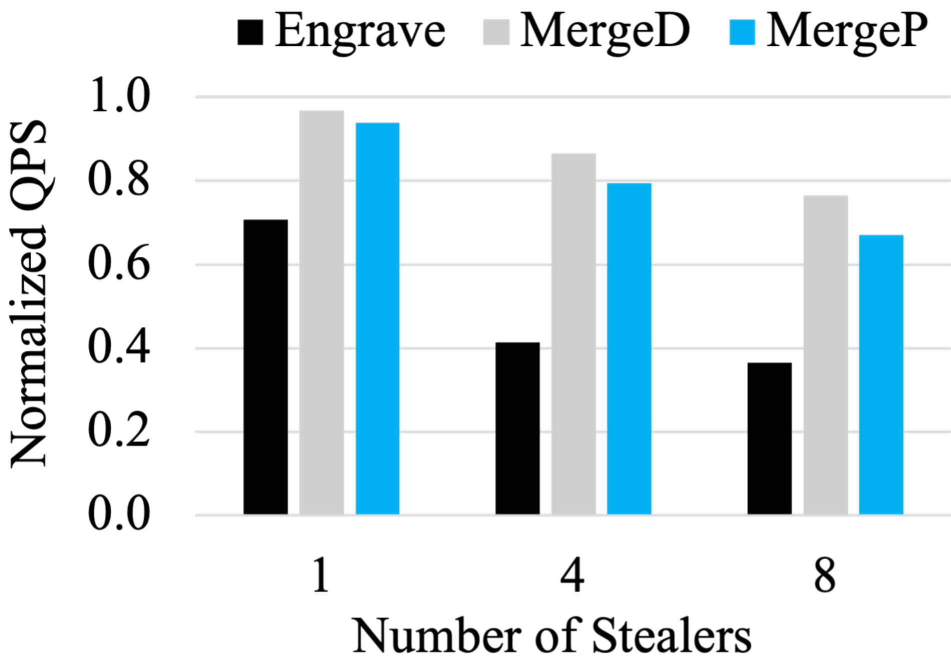
- Query cache stores results of an earlier query obviating (re)computation



- **Interesting result:** One can place the query cache in NVM and still gain QPS

Eager engrave policy is justified in limiting NVM contention

- Experiment: Execute stealing instances that only do engraving or merging
- D and P are two types of merge operations



- QPS is lower when engraving (NVM writes) in progress showing NVM bw is limited

Key Takeaways

- Growing datasets demand more memory for real-time search engines

- **Filesystem operations are expensive in state-of-the-art enterprise engines inhibiting real-time operation**
  - DRAM-NVM server delivers better QPS and tail response than highly optimized SSD-DRAM ones

- **SPIRIT offers instant visibility of ingested documents**
  - A memory-centric design and operation facilitates instant visibility

- Crash consistency guarantees are stronger than SSD-based engines but slows down ingestion-side operations