



WHITE PAPER · DATA STORAGE ENGINES

# State-of-the-Art Data Storage Engines

A review of architectural paradigms and future frontiers

---

White Paper 1 · Data Storage Engines series · Version 1.0 · 26 October 2026

Bassirah Technology — Innovation & R&D

# State-of-the-Art Data Storage Engines: A Review of Architectural Paradigms and Future Frontiers

White Paper 1 · Data Storage Engines series · Bassirah Technology — Innovation & R&D · 26 October 2026

## Document control

| Field   | Value                                                                                                                                             |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| Series  | Data Storage Engines, paper 1 of 6                                                                                                                |
| Author  | Bassirah Technology — Innovation & R&D                                                                                                            |
| Version | 1.0 · 26 October 2026                                                                                                                             |
| Scope   | A technology survey. It describes published research and publicly documented systems; it does not report Bassirah benchmarks or customer results. |

## Contents

|                                                                |   |
|----------------------------------------------------------------|---|
| 1. Abstract.....                                               | 3 |
| 2. Introduction and background .....                           | 3 |
| 2.1 The hardware evolution .....                               | 3 |
| 2.2 The classic dilemma: the RUM conjecture.....               | 3 |
| 2.3 The baseline engines .....                                 | 3 |
| 3. A structural taxonomy of modern data engines .....          | 3 |
| 3.1 Class A: write-optimized engines (LSM-tree variants) ..... | 3 |
| 3.2 Class B: analytical and read-optimized engines .....       | 4 |
| 3.3 Class C: hardware-native and in-memory architectures ..... | 4 |
| 3.4 Class D: AI-oriented and vector storage engines.....       | 4 |
| 3.5 Class E: open table formats and the lakehouse .....        | 4 |
| 4. Emerging paradigms and future frontiers.....                | 4 |
| 4.1 Learned index structures .....                             | 4 |
| 4.2 CXL and memory pooling .....                               | 4 |
| 4.3 Serverless and compute-storage decoupling.....             | 4 |
| 5. Comparative analysis.....                                   | 5 |
| 6. Open challenges and conclusion .....                        | 5 |
| 7. What this means for platform decisions .....                | 5 |
| References .....                                               | 5 |

# 1. Abstract

The architecture of data storage engines is changing. Engines that were designed for single-threaded, disk-bound operation are giving way to designs built for many cores, flash storage and hardware acceleration. This paper surveys current storage engines. It contrasts traditional Log-Structured Merge-tree (LSM-tree) and B-tree designs with newer architectures, and extends the usual taxonomy to cover AI-oriented vector storage, memory disaggregation through Compute Express Link (CXL), and the open table formats that underpin the data lakehouse. By grouping these approaches into five classes, it explains how current systems work around classical hardware bottlenecks to meet the demands of very large, AI-driven and widely distributed data platforms.

## 2. Introduction and background

### 2.1 The hardware evolution

Traditional engines were optimized for sequential I/O, to hide the rotational latency of hard disk drives (HDDs). High-IOPS NVMe solid-state drives (SSDs) and parallel multi-core processors largely inverted this bottleneck, moving the constraint to CPU efficiency and memory bandwidth. The hardware frontier has now moved again: Compute Express Link (CXL) and memory disaggregation let systems pool memory across machines, which changes how engines manage buffers and caches.

### 2.2 The classic dilemma: the RUM conjecture

Storage design is still shaped by the RUM conjecture: a three-way trade-off between Read overhead, Update cost and Memory (space) overhead. The conjecture holds that an access method that bounds two of these overheads must give up ground on the third. This trade-off explains much of the divergence between operational, analytical and vector-oriented engines.

### 2.3 The baseline engines

The modern era of embedded key-value storage was shaped by LevelDB, an embedded LSM-tree library optimized for writes. Its limits under concurrent workloads led to RocksDB, a fork that was built to make better use of multi-core processors and fast SSDs. RocksDB became a common foundation for operational data storage, and many of its concurrency and compaction techniques are now standard practice.

## 3. A structural taxonomy of modern data engines

### 3.1 Class A: write-optimized engines (LSM-tree variants)

Write-optimized engines maximize ingest throughput by buffering writes in memory and flushing them asynchronously to immutable on-disk files (SSTables). Modern LSM engines such as RocksDB and Pebble achieve high concurrency through fine-grained synchronization and pipelined write-ahead logging. The newest members of this class are cloud-native: engines such as SlateDB write directly to object storage (for example Amazon S3), decoupling compute from storage and trading lower latency for elastic capacity and durability.

### **3.2 Class B: analytical and read-optimized engines**

Analytical engines are optimized for read-heavy queries over large datasets, and avoid tuple-at-a-time processing. Engines such as DuckDB and ClickHouse use vectorized execution: they process batches of columnar data sized for CPU caches and SIMD instructions, which lets filtering and aggregation run in parallel at the hardware level.

### **3.3 Class C: hardware-native and in-memory architectures**

Class C engines are designed to exploit low-latency NVMe media. They often use kernel-bypass and asynchronous I/O interfaces (for example SPDK and io\_uring) to reduce operating-system overhead. Systems such as Umbra treat fast NVMe SSDs as the primary persistent layer while approaching in-memory performance, using variable-size pages and pointer swizzling in their buffer management.

### **3.4 Class D: AI-oriented and vector storage engines**

The growth of generative AI and large language models (LLMs) has produced a class of engines and services built for high-dimensional vector embeddings, such as Milvus, Qdrant and Pinecone. Instead of scalar indexing, they rely on Approximate Nearest Neighbor (ANN) search, using structures such as Hierarchical Navigable Small World (HNSW) graphs and DiskANN, and they prioritize fast similarity search over dense, unstructured data.

### **3.5 Class E: open table formats and the lakehouse**

Combining storage-engine functions with metadata layers directly on object storage has produced the data lakehouse. Open table formats such as Apache Iceberg, Delta Lake and Apache Hudi act as distributed storage layers: they provide ACID transactions, time travel and schema evolution on top of immutable Parquet files in object stores, bridging data lakes and data warehouses.

## **4. Emerging paradigms and future frontiers**

### **4.1 Learned index structures**

Learned indexes propose replacing heuristic structures such as B-trees with models that learn the data distribution. By modeling the cumulative distribution function (CDF) to predict where a key is stored, these structures can reduce lookup cost and index size for suitable data distributions, turning index design into a model-fitting problem.

### **4.2 CXL and memory pooling**

Physical memory disaggregation through CXL is reshaping buffer-pool management. Because CXL lets nodes share memory with lower latency than network transfers, storage engines can treat pooled memory as an extension of local cache, which can reduce read latency for large analytical and AI workloads.

### **4.3 Serverless and compute-storage decoupling**

Database deployment is moving toward serverless models. Architectures such as Neon (PostgreSQL with separated compute and storage) and Amazon Aurora Serverless scale compute independently of the storage layer, so resources can follow demand and cost can follow use.

## 5. Comparative analysis

Table 1 summarizes the main structural differences and the primary bottleneck of each paradigm.

| Engine / architecture                      | Primary workload                   | Storage layout                                  | Bottleneck focus                      |
|--------------------------------------------|------------------------------------|-------------------------------------------------|---------------------------------------|
| <b>Class A: RocksDB (LSM-tree)</b>         | High-throughput OLTP and key-value | Row-oriented, tiered SSTables on SSD            | Write amplification, compaction CPU   |
| <b>Class B: DuckDB (vectorized)</b>        | Interactive OLAP                   | Columnar, vectorized chunks                     | Memory bandwidth, CPU cache           |
| <b>Class C: Umbra (hardware-native)</b>    | Near in-memory performance on NVMe | Pointer-swizzled, variable-size pages           | Kernel overhead, NVMe queue depth     |
| <b>Class D: Milvus / Qdrant (vector)</b>   | AI and LLM similarity search       | High-dimensional embeddings, HNSW               | ANN index memory size, search latency |
| <b>Class E: Apache Iceberg (lakehouse)</b> | Very large-scale analytics         | Parquet on object storage with a metadata layer | Network latency, metadata scaling     |

Table 1: Structural comparison of modern data engine paradigms.

## 6. Open challenges and conclusion

Several challenges remain. Write amplification still shortens the life of flash media under LSM-tree engines. The "complexity paradox", where engines expose hundreds of tunable parameters, is a strong case for automated, model-driven tuning.

For vector search, latency at very large scale remains a hurdle for real-time AI applications, and interoperability between table formats remains a challenge for lakehouse environments. Overall, storage engines are converging toward unified hybrid transactional/analytical processing (HTAP) layers. With CXL memory pooling and AI-oriented structures, future engines are likely to act as elastic, hardware-accelerated layers between distributed serverless compute and increasingly varied storage media.

## 7. What this means for platform decisions

The class of engine a workload needs follows from its dominant trade-off, not from a product list. Bassirah's platform work applies this taxonomy: ClickHouse (partner-led or ClickHouse OSS) for Class B analytical serving, Apache Iceberg tables on S3-compatible object storage for Class E lakehouse designs, and PostgreSQL or YugabyteDB for operational SQL, each chosen per workload and validated on a representative workload before it is adopted. The next papers in this series examine each class in turn.

## References

1. Dong, S., Kryczka, A., Jin, Y. and Stumm, M. (2021). RocksDB: Evolution of Development Priorities in a Key-value Store Serving Large-scale Applications. ACM Transactions on Storage, 17(4), Article 26.
2. Raasveldt, M. and Mühleisen, H. (2019). DuckDB: an Embeddable Analytical Database. SIGMOD.

3. Neumann, T. and Freitag, M. (2020). Umbra: A Disk-Based System with In-Memory Performance. CIDR.
4. Kraska, T., Beutel, A., Chi, E. H., Dean, J. and Polyzotis, N. (2018). The Case for Learned Index Structures. SIGMOD.
5. Wang, J., et al. (2021). Milvus: A Purpose-Built Vector Data Management System. SIGMOD.
6. Armbrust, M., Ghodsi, A., Xin, R. and Zaharia, M. (2021). Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. CIDR.
7. Athanassoulis, M., Kester, M. S., Maas, L. M., Stoica, R., Idreos, S., Ailamaki, A. and Callaghan, M. (2016). Designing Access Methods: The RUM Conjecture. EDBT.

Product and project names are trademarks of their respective owners. Their mention describes publicly documented technology and does not imply a business relationship, except where Bassirah names a Strategic Technology Partner.