



How to Work with Open Table Formats

Presented by: William McKnight
President, McKnight Consulting Group
3 X Inc 5000

 /in/wmcknight
www.mcknightcg.com
(214) 514-1444



McKnight Consulting Group Enterprise Data & AI Architecture Services

Empirical benchmarking, lakehouse modernization, and deterministic infrastructure for agentic AI.



Data Architecture

Open Lakehouses: Decoupled Apache Iceberg and Delta Lake.

Data Mesh: Domain-owned products with strict API contracts.

Streaming Planes: Low-latency Kafka and Redpanda event pipelines.



Benchmarking & Health

Empirical Testing: Independent TPC-C, TPC-DS, and YCSB audits.

ML Observability: Automated circuit breakers preventing data corruption.

Financial TCO: Deterministic ROI and labor velocity modeling.



Semantic & Agentic AI

Vector & Graph: HNSW indexing and semantic RDF.

Automated MDM: Autonomous entity extraction and golden records.

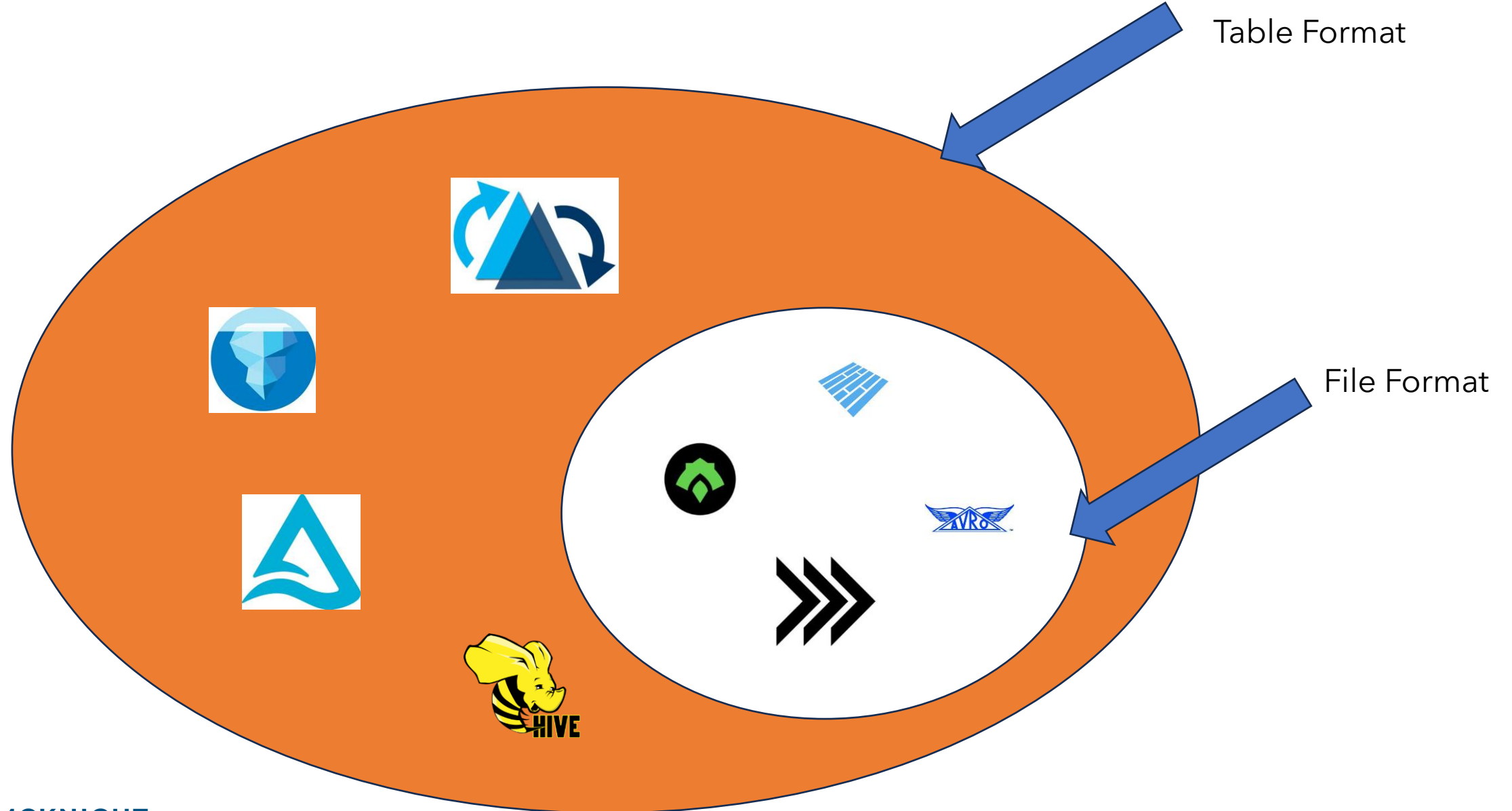
Agentic Context: Model Context Protocol grounding for LLMs.

wmcknight@mcknightcg.com

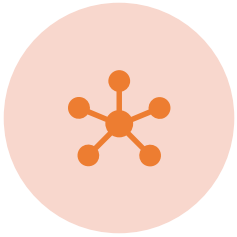
Data Formats

- **Table Format:** A method of organizing and structuring a dataset's files within a file system or object store, enabling them to be presented and queried as a unified, logical table.
- **File Format:** The physical format in which data is stored in files, such as:
 - Text formats (e.g., CSV, JSON)
 - Binary formats (e.g., Parquet, Avro, ORC, Arrow)

Open Table Format and Open File Format



Benefits of Open Data Formats



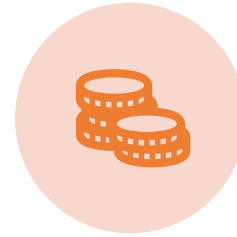
INTEROPERABILITY



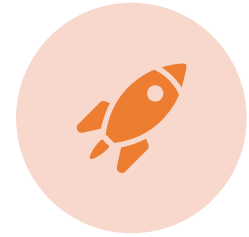
**FLEXIBILITY AND
SCALABILITY**



**COLLABORATION
AND COMMUNITY**



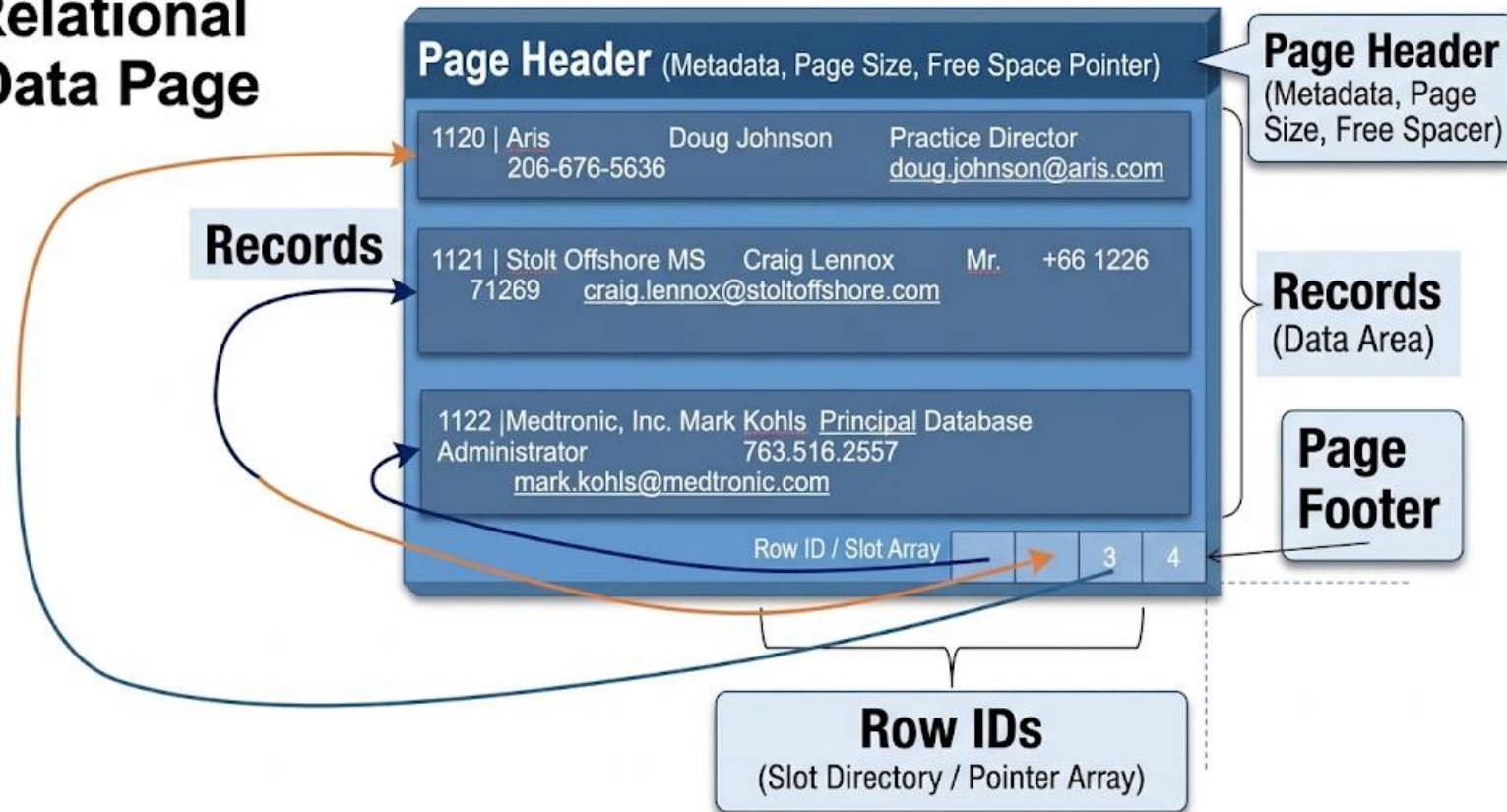
**COST-
EFFECTIVENESS**



FUTURE-PROOFING

The Original Data Format: Relational Database

Relational Data Page





Open Table Formats



Two Decades of Data Infrastructure: The Evolution to Apache Iceberg

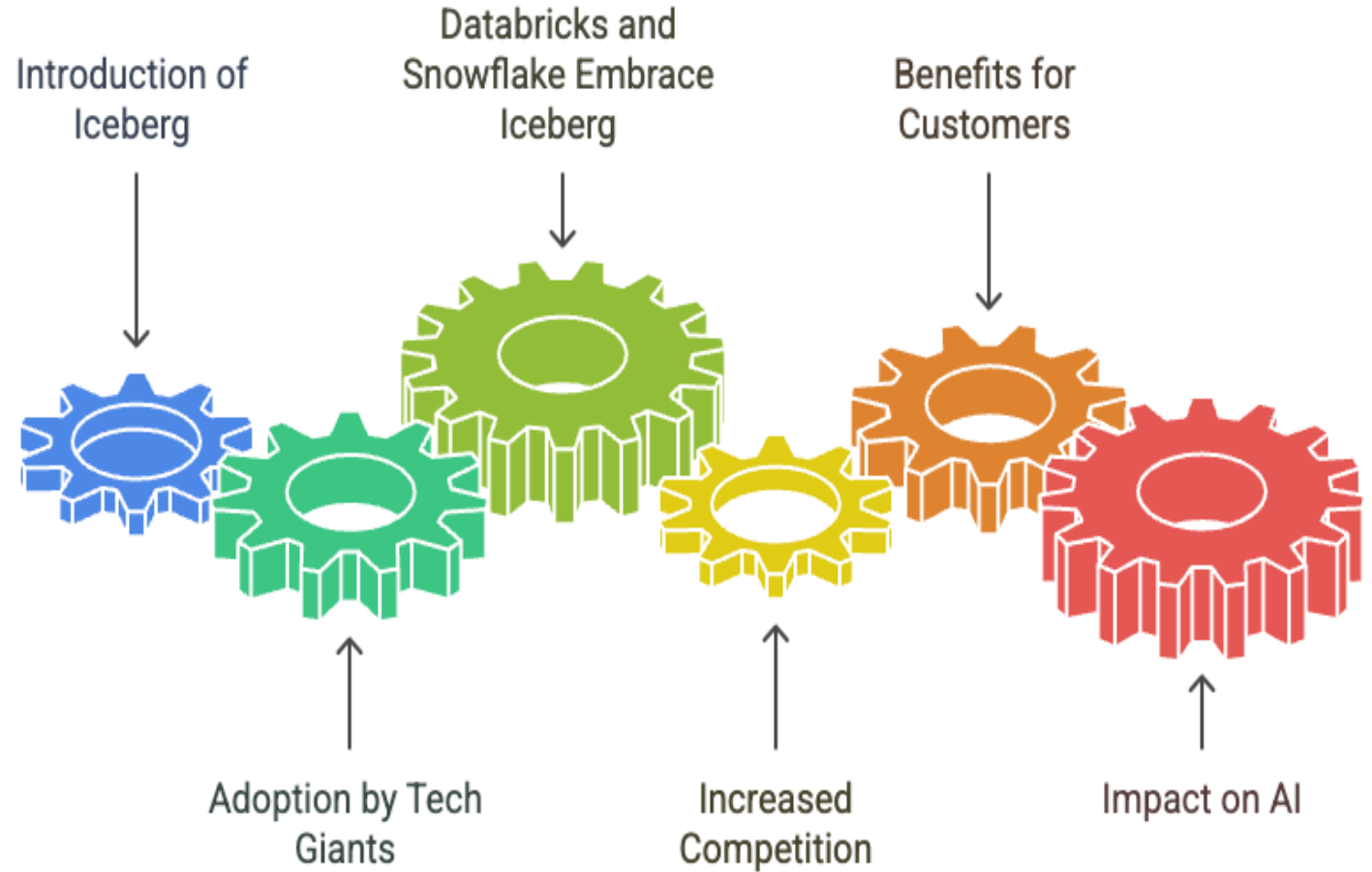
From Distributed Processing to Modern Decoupled Data Lakes

Era / Year	Milestone	Key Innovation	Constraint / Catalyst
2005	Apache Hadoop	HDFS distributed storage & MapReduce parallel compute	High complexity (required Java MapReduce)
2008	Apache Hive	Hive Metastore bringing SQL queries to HDFS	Coarse metadata pointers; slow for real-time analytics
2010s	Cloud & Real-Time	Shift to S3 cloud storage & interactive engines (Presto)	Hive metastore bound to HDFS; couldn't scale
2017	Apache Iceberg	Independent metadata layer separating storage from compute	Solves Hive bottlenecks with file-level indexing
Mid-2020s	The AI Era	Open-source standard for high-scale data lakes	Critical foundation for training & tuning AI models

CORE ICEBERG CAPABILITIES

ACID Transactions • Schema & Partition Evolution • Time-Travel Data Versioning • Universal Engine Support (Spark, Presto)

Evolution of Data Storage and Compute Engines



The Data Lake



Files of structured and unstructured data.

Data Warehouse versus Data Lake

- Ingestion & Data Movement Costs (Major Advantage: S3 Data Lake)
- Structured Analytics Performance (Near Parity)
- Semi-Structured & JSON Querying (Major Advantage: S3 Data Lake)
- Operational Architectural Fit
 - Relational data warehouses: high-concurrency reporting, strict ACID compliance, and predictable low-latency BI
 - S3 data lakes excel at high-volume ingestion, unstructured data, ML/data science, and low-cost historical retention

Table Format



Table A

Table B

A framework that presents multiple files in the data lake as a unified table.

Directory Method vs File List Method

/table1
 /field1
 /field2
 /field3
 /field4
 /field5

Table 1

File1.parquet-field=1
File2.parquet-field=2
File3.parquet-field=3
File4.parquet-field=4
File5.parquet-field=5

Table 2

File1.parquet-field=1
File2.parquet-field=2
File3.parquet-field=3
File4.parquet-field=4
File5.parquet-field=5

Iceberg

- Improved query performance
- Efficient metadata management
- Good for large scale data analytics in modern cloud environments
- ACID transactions
- Schema Evolution
- Partition Evolution
- Compaction Management
- Upserts and Deletes



Snowflake Performance across Iceberg Catalog Architectures

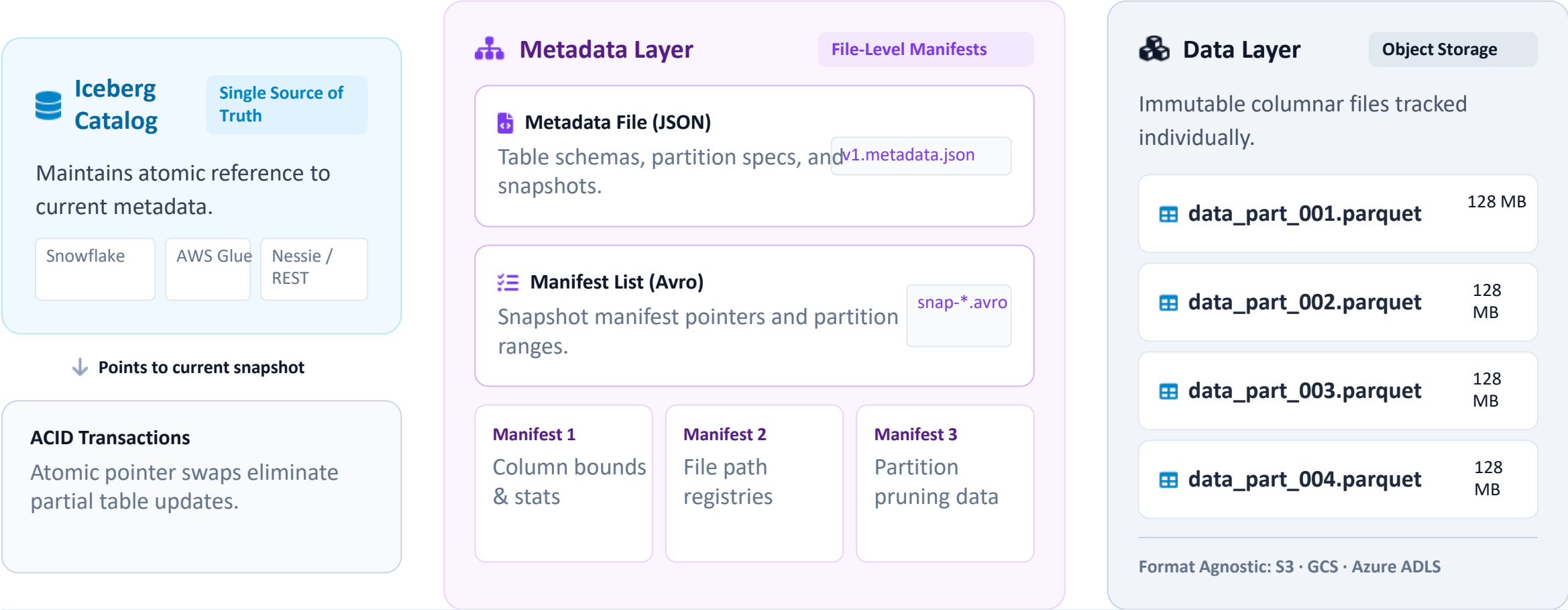
In a 10TB TPC-DS benchmark using single-user warm power runs, Snowflake (Gen2 2X-Large Warehouse) was tested across three Iceberg catalog setups:

- **Internal / Vendor-Managed Catalog (Snowflake-managed Iceberg): 595 seconds.**
- **Neutral External Catalog (AWS Glue-managed Iceberg): 759 seconds.**
- **Competing External Catalog (Databricks-managed Iceberg): 808 seconds.**



Apache Iceberg Architecture

Catalog Layer Metadata Layer Data Layer



NO DIRECTORY LISTINGS

Metadata pruning skips non-relevant data files directly.

SNAPSHOT ISOLATION

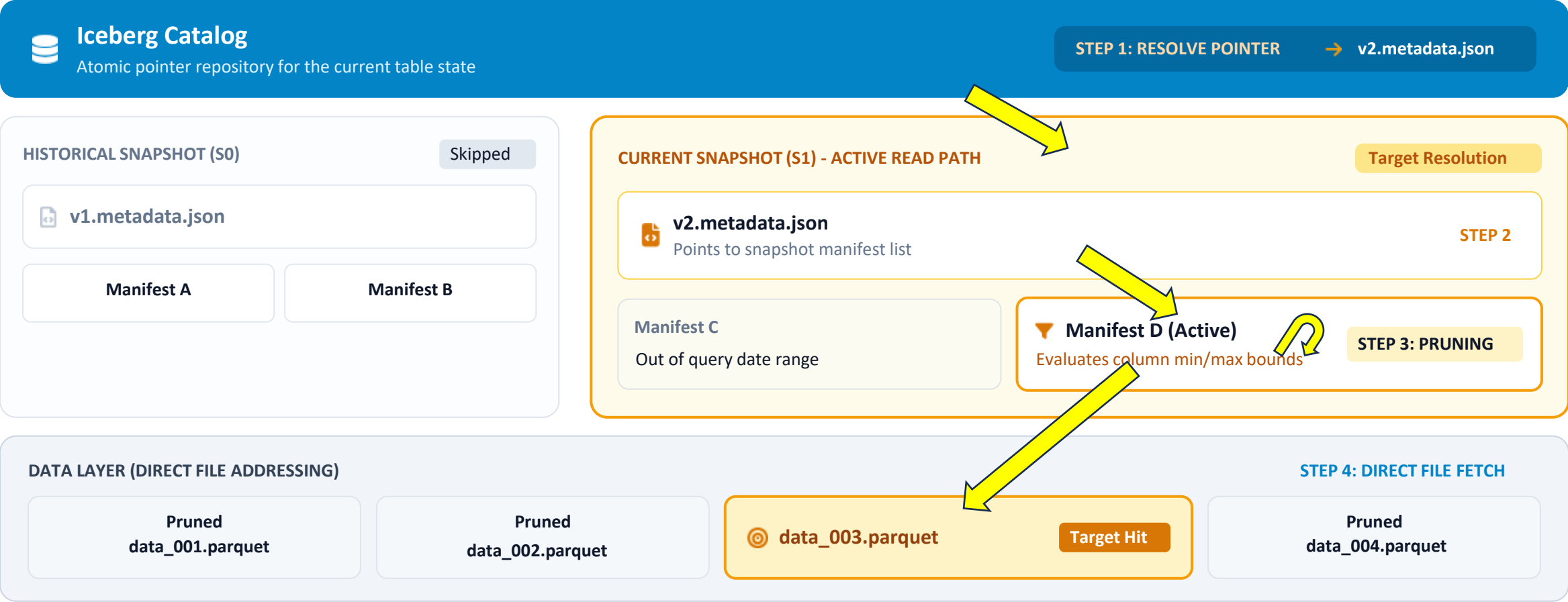
Readers always query an immutable, consistent state.

TIME TRAVEL

Previous snapshots remain queryable for audits and rollback.

The Iceberg Read Path

Active Traversal Path Pruned / Skipped Files



1. POINTER RESOLUTION

Catalog identifies current snapshot in constant time $O(1)$.

2. STATISTICAL PRUNING

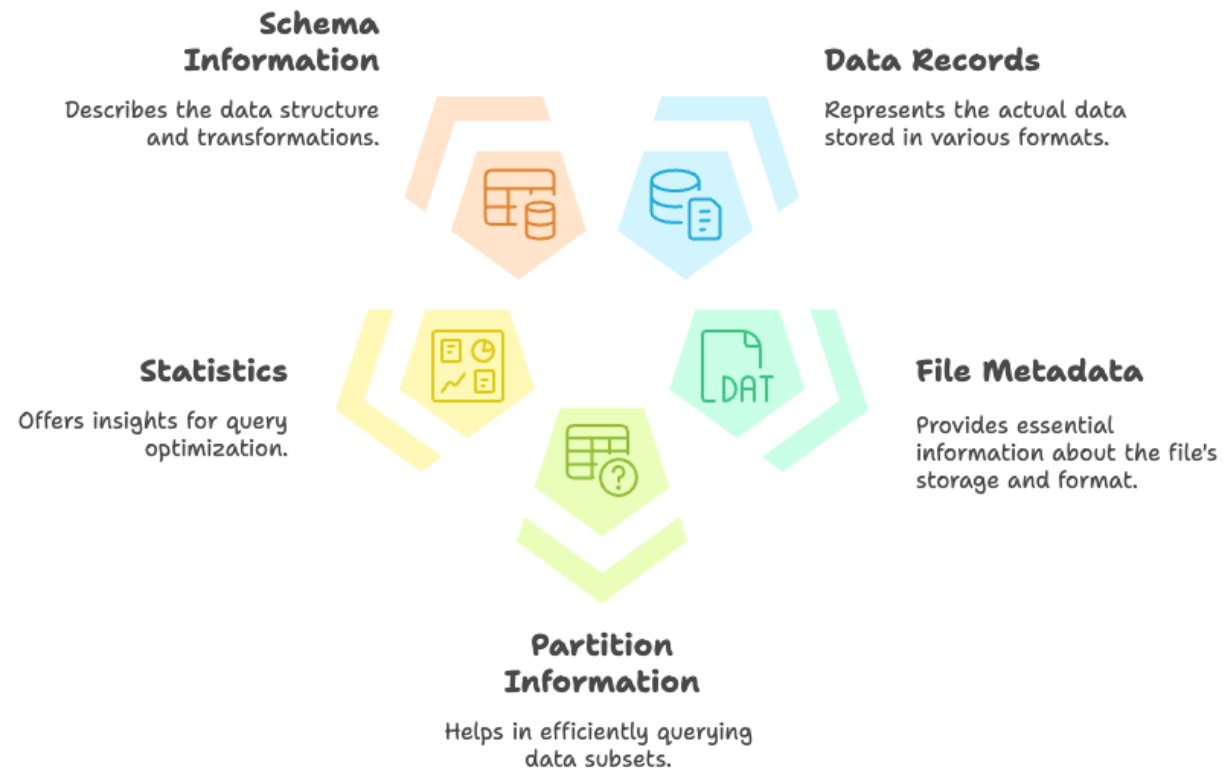
Manifest min/max statistics discard non-matching files in memory.

3. ZERO DIRECTORY SCANS

Engine requests exact byte offsets directly from object storage.

Data Files

- Physical files with data
- In an open file format like Parquet, ORC, Avro or Arrow
- Optimized for reads and writes



Create Table with Iceberg

```
CREATE TABLE ALBUM (  
  ALBUM_NAME VARCHAR (75),  
  ARTIST VARCHAR (75),  
  GENRE VARCHAR (25),  
  YEAR INTEGER)  
WITH  
(TYPE='ICEBERG');
```



Insert into Iceberg Table

INSERT INTO

ALBUM (ALBUM_NAME, ARTIST, GENRE, YEAR)

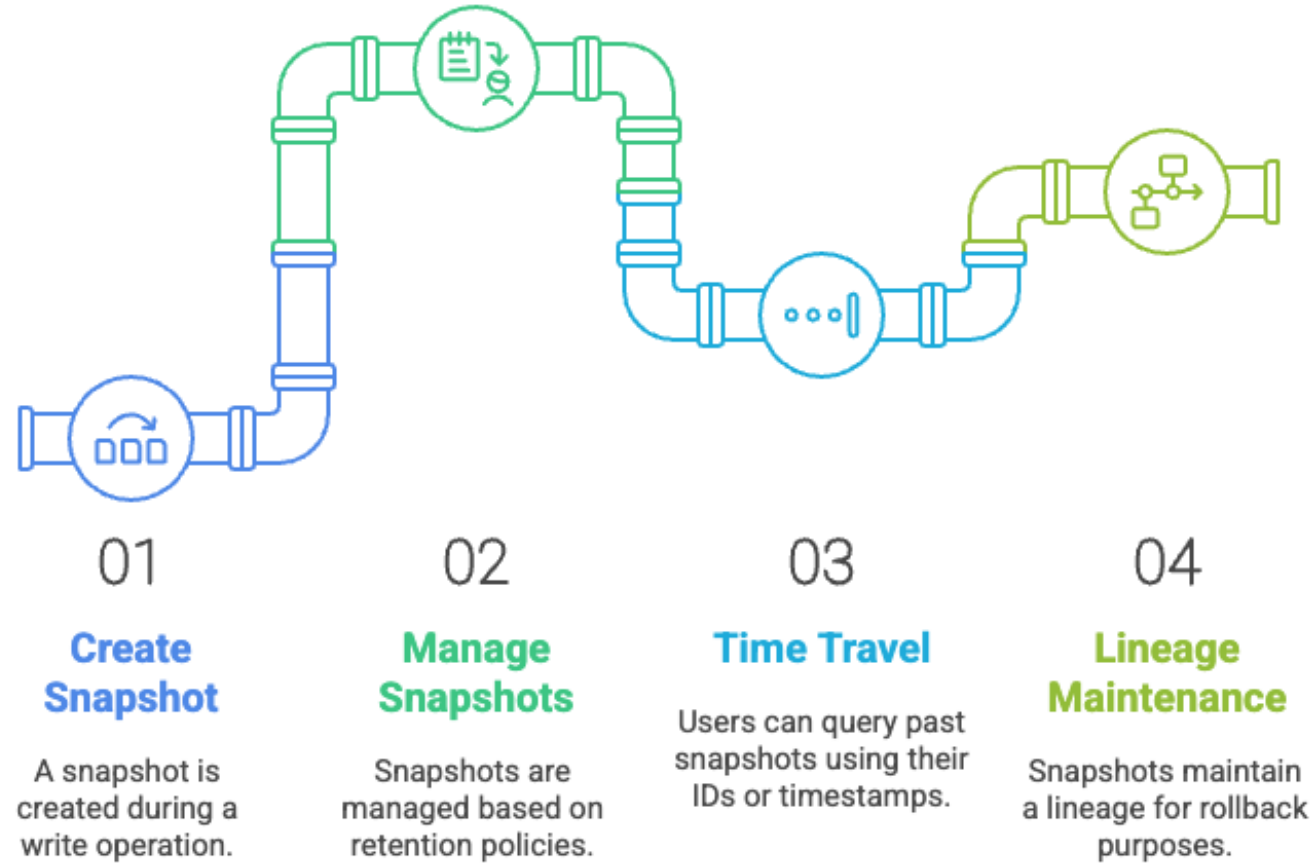
VALUES

('IV','LED ZEPPELIN','ROCK', 1971),

('DON'T LOOK BACK','BOSTON','ROCK',1978);



Snapshots



Time Travel

- State of table is a snapshot
- Previous snapshots are also “tables”
- Can rollback to prior version
- Can use multiple versions in a single select

```
SELECT * FROM table_name AS OF TIMESTAMP '2025-01-01 12:00:00'
```

```
SELECT * FROM table_name AS OF VERSION 123
```

```
SELECT * FROM table_name AS OF SNAPSHOT 'snapshot-id'
```

Schema Evolution

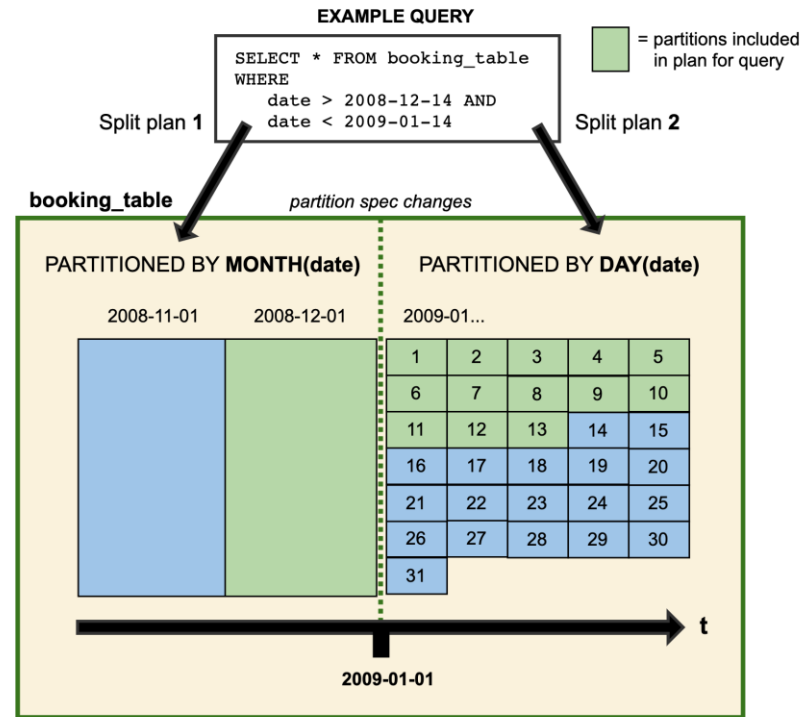
Date	Store #	State	Class	Sales	Category
2/13/25	32	NY	A	6	Gen
2/13/25	35	CT	A	9	Spec
2/13/25	36	CT	C	11	Gen
2/13/25	39	SD	D	8	Gen
2/13/25	42	KY	A	5	Spec
2/13/25	43	VT	C	14	Spec
2/13/25	47	GA	A	31	Gen
2/13/25	51	MD	A	4	Sub
2/13/25	55	DC	D	16	Gen
2/13/25	59	NY	B	7	Gen
2/13/25	62	NJ	C	9	Spec



Date	Store #	Class	Sales	Category	Cashier
2/13/25	32	A	6	Gen	26
2/13/25	35	A	9	Spec	14
2/13/25	36	C	11	Gen	57
2/13/25	39	D	8	Gen	32
2/13/25	42	A	5	Spec	36
2/13/25	43	C	14	Spec	85
2/13/25	47	A	31	Gen	2
2/13/25	51	A	4	Sub	68
2/13/25	55	D	16	Gen	223
2/13/25	59	B	7	Gen	69
2/13/25	62	C	9	Spec	75

- Retain current data in old schema with new data in new schema
- No need to rewrite schema
- Applies to Add/Drop/Rename/Update/Reorder

Partition Evolution



Example from iceberg.apache.org

- Partitioning enables you to divide data into smaller, more manageable chunks based on specific columns
- Changing the partition scheme over time: add, remove, update partitions

Day 1+ Lakehouse Operational Disciplines



Small-File Bin-Packing

Automated compaction.

Combines small append-stream files into optimal 128MB–512MB columnar row groups.



Snapshot Expiration

Controlling metadata debt.

Prunes historical manifest branches to prevent query planning latency degradation.



Orphan Cleanup

Storage leak prevention.

Deletes uncommitted storage objects left behind by failed distributed driver nodes.

Other Open Table Formats

1. Delta Lake

- ACID transactions: Ensures data consistency and reliability.
- Schema evolution: Supports changes to table schema over time.
- Time travel: Enables querying historical data versions.

2. Apache Hudi (Hadoop Upsert Delete and Incremental)

- Upsert and delete support: Enables updating and deleting individual records.
- Incremental data processing: Supports processing incremental data changes.

Iceberg vs Delta Lake

- **Metadata Architecture:** Iceberg uses a directory-agnostic manifest tree. Delta Lake relies on an ordered, directory-bound transaction log (`_delta_log/`).
- **Engine Neutrality:** Iceberg was built for engine parity across Snowflake, Trino, and Spark. Delta Lake centers on Spark and Databricks.
- **Partitioning:** Iceberg features metadata-only evolution. Delta Lake relies on physical paths, shifting toward Liquid Clustering.
- **Catalog Standards:** Iceberg anchors on a neutral REST Catalog protocol. Delta Lake centers on Unity Catalog, which bridges Iceberg via REST.
- **Row-Level ACID:** Iceberg natively handles Copy-on-Write and Merge-on-Read. Delta Lake traditionally favored Copy-on-Write, adding deletion vectors in newer protocols.

Universal Formats Without Data Duplication



Delta UniForm

Universal Format generation.

Generates Iceberg & Hudi metadata automatically on write, sharing raw Parquet blocks.



Apache XTable

Omni-directional abstraction.

Cross-syncs Delta, Iceberg, and Hudi metadata bidirectionally without moving bytes.

The Open Table Format Evolution

From competing file formats to open catalog governance and native cloud storage.



SHIFT 01

Format Truce

Tabular Acquired: Core engine creators united.

Metadata Lenses: UniForm eliminates Parquet rewrites.

Feature Parity: Iceberg v3 adds deletion vectors.



SHIFT 02

The Catalog War

REST Standard: Industry standardizes on Iceberg REST.

Unity Catalog: Databricks open-sources multi-format governance.

Polaris Launch: Snowflake counters with neutral catalog.



SHIFT 03

Storage Native

Amazon S3 Tables: Iceberg embedded directly in S3.

Auto Maintenance: Continuous automated compaction at rest.

Multi-Cloud Native: BigQuery and Fabric integrate Iceberg.

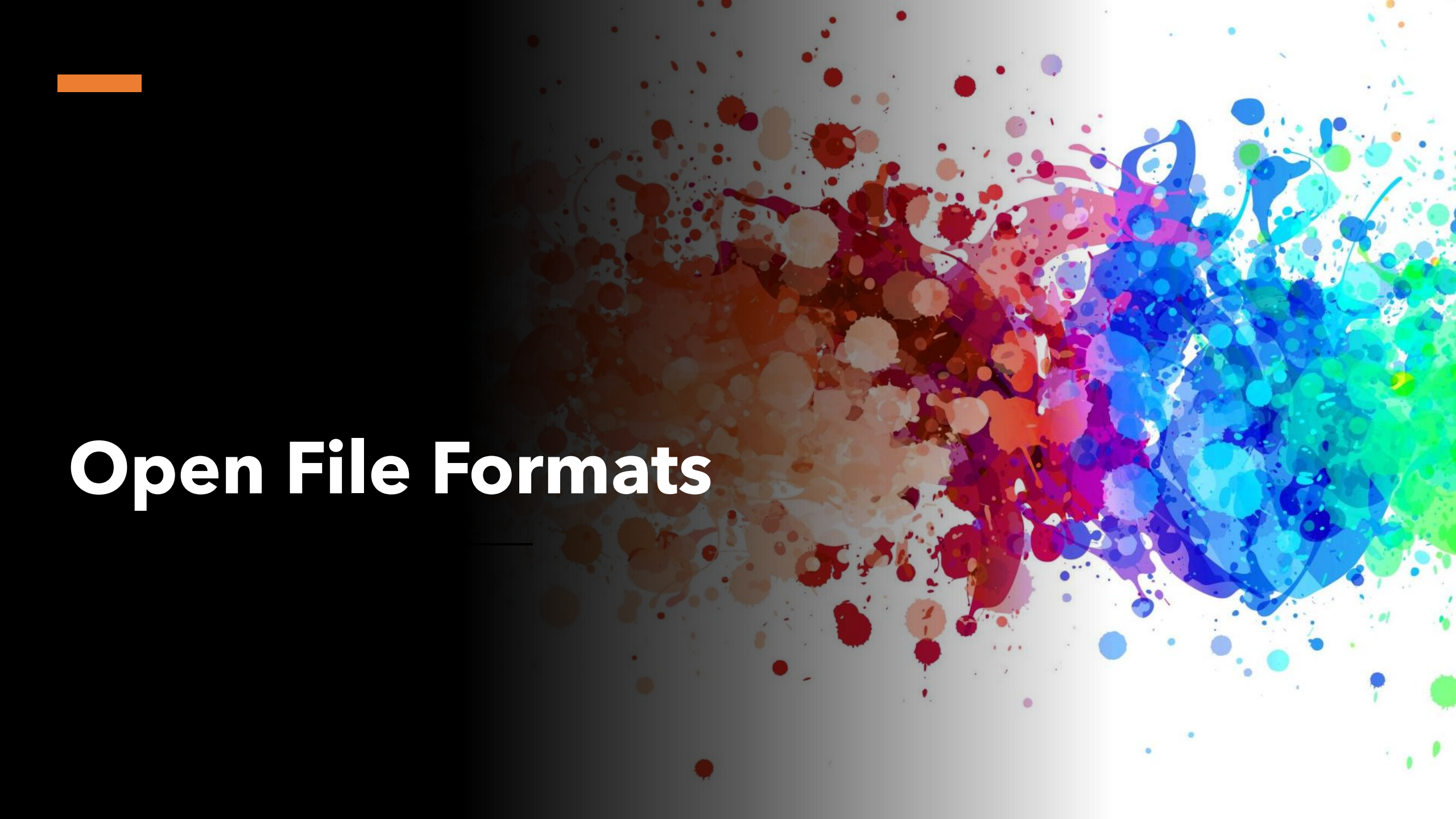
Strategic Consensus: Storage is standard Parquet; engines are swappable; governance lives in the REST catalog.

Who Supports What

- Virtually every major analytical engine, cloud data warehouse, and lakehouse platform support Iceberg
- Delta Lake is the default for Databricks, Apache Spark, and Microsoft Fabric ecosystems with universal accessibility either directly or via UniForm
- Hudi
 - AWS: Long-standing native integration across Amazon EMR, Athena, and AWS Glue.
 - Interoperability (Apache XTable): Backed by Onehouse, Microsoft, and Google, enabling Hudi tables to be read seamlessly by Iceberg and Delta Lake query engines without data copying.
 - Compute Engines: Deep native read/write pipelines in Apache Spark and Apache Flink; read support across Trino and PrestoDB.
- Hive
 - Cloudera: Led the development efforts for Apache Hive; now transitioning to Iceberg
 - Many large companies still running Hive in production



Open File Formats



You don't have to use an Open File Format with an Open Table Format

You can use:

- **Open file formats without an open table format:** Use open file formats like Parquet or ORC without an open table format. This approach is simpler but lacks features like schema evolution and time travel.
- **Proprietary table formats with open/closed file layers:** Using vendor-managed analytical formats (e.g., Snowflake native internal tables, Google BigQuery native storage). While highly optimized for their proprietary compute engines, they introduce vendor lock-in and restrict open multi-engine access.
- **Open table formats over open file formats:** The current industry standard (Apache Iceberg, Delta Lake)—using open metadata specifications over standard Parquet files to provide full ACID governance without compute engine lock-in.

Why Parquet?

- Columnar
- Compression
- Language agnostic
- Open Source
- Complex Data Types



Column Approach Origins 2005

C-Store: A Column-oriented DBMS

Mike Stonebraker*, Daniel J. Abadi*, Adam Batkin*, Xuedong Chen†, Mitch Cherniack*,
Miguel Ferreira*, Edmond Lau*, Amerson Lin*, Sam Madden*, Elizabeth O'Neil†,
Pat O'Neil†, Alex Rasin†, Nga Tran†, Stan Zdonik‡

*MIT CSAIL
Cambridge, MA

†Brandeis University
Waltham, MA

‡UMass Boston
Boston, MA

§Brown University
Providence, RI

Abstract

This paper presents the design of a read-optimized relational DBMS that contrasts sharply with most current systems, which are write-optimized. Among the many differences in its design are: storage of data by column rather than by row, careful coding and packing of objects into storage including main memory during query processing, storing an overlapping collection of column-oriented projections, rather than the current fare of tables and indexes, a non-traditional implementation of transactions which includes high availability and snapshot isolation for read-only transactions, and the extensive use of bitmap indexes to complement B-tree structures.

We present preliminary performance data on a subset of TPC-H and show that the system we are building, C-Store, is substantially faster than popular commercial products. Hence, the architecture looks very encouraging.

1. Introduction

Most major DBMS vendors implement record-oriented storage systems, where the attributes of a record (or tuple) are placed contiguously in storage. With this *row store* architecture, a single disk write suffices to push all of the fields of a single record out to disk. Hence, high performance writes are achieved, and we call a DBMS with a row store architecture a *write-optimized* system. These are especially effective on OLTP-style applications.

In contrast, systems oriented toward ad-hoc querying of large amounts of data should be *read-optimized*. Data warehouses represent one class of read-optimized system,

in which periodically a bulk load of new data is performed, followed by a relatively long period of ad-hoc queries. Other read-mostly applications include customer relationship management (CRM) systems, electronic library card catalogs, and other ad-hoc inquiry systems. In such environments, a *column store* architecture, in which the values for each single column (or attribute) are stored contiguously, should be more efficient. This efficiency has been demonstrated in the warehouse marketplace by products like Sybase IQ [FREN95, SYBA04], Addamark [ADDA04], and KDB [KDB04]. In this paper, we discuss the design of a column store called C-Store that includes a number of novel features relative to existing systems.

With a column store architecture, a DBMS need only read the values of columns required for processing a given query, and can avoid bringing into memory irrelevant attributes. In warehouse environments where typical queries involve aggregates performed over large numbers of data items, a column store has a sizeable performance advantage. However, there are several other major distinctions that can be drawn between an architecture that is read-optimized and one that is write-optimized.

Current relational DBMSs were designed to pad attributes to byte or word boundaries and to store values in their native data format. It was thought that it was too expensive to shift data values onto byte or word boundaries in main memory for processing. However, CPUs are getting faster at a much greater rate than disk bandwidth is increasing. Hence, it makes sense to trade CPU cycles, which are abundant, for disk bandwidth, which is not. This tradeoff appears especially profitable in a read-mostly environment.

There are two ways a column store can use CPU cycles to save disk bandwidth. First, it can code data elements into a more compact form. For example, if one is storing an attribute that is a customer's state of residence, then US states can be coded into six bits, whereas the two-character abbreviation requires 16 bits and a variable length character string for the name of the state requires many more. Second, one should *densepack* values in storage. For example, in a column store it is straightforward to pack N values, each K bits long, into N

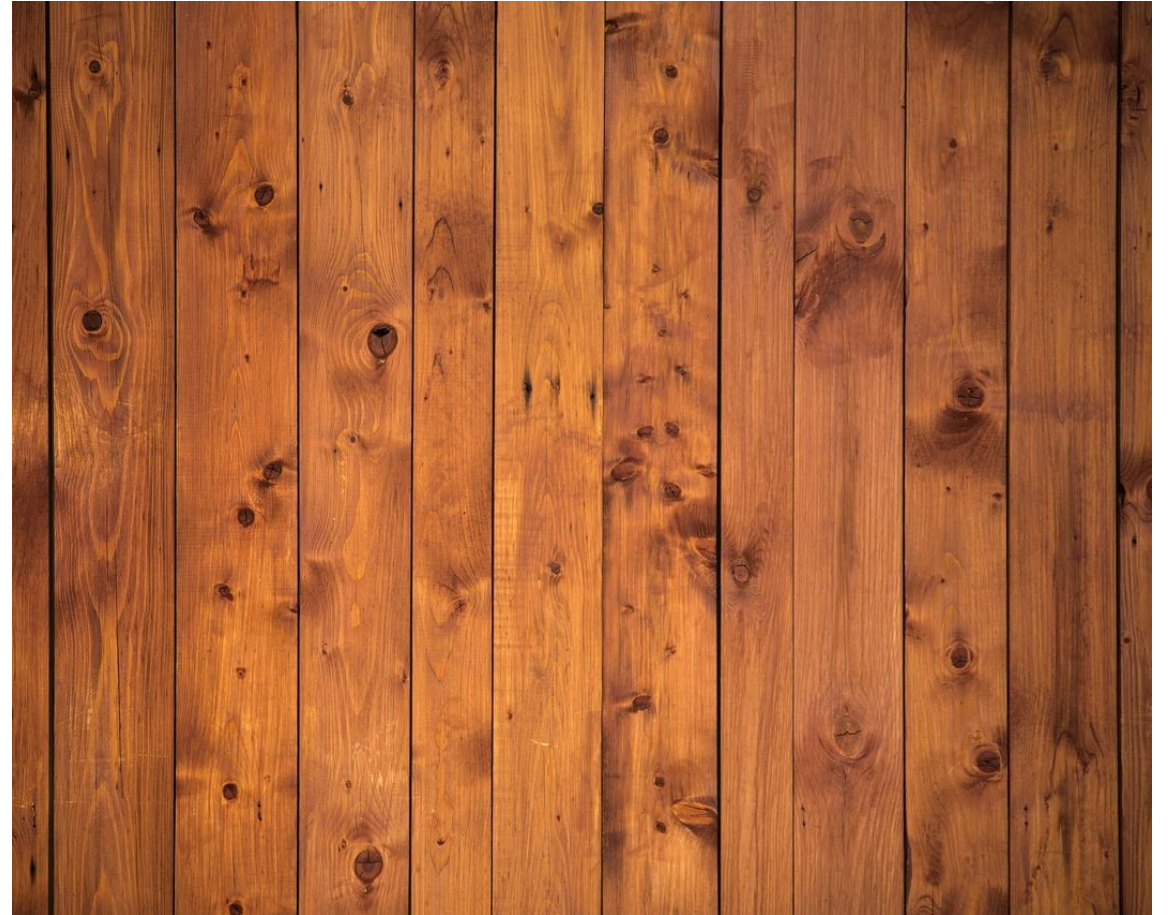
* K bits. The coding and compressibility advantages of a

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment

Proceedings of the 31st VLDB Conference,
Trondheim, Norway, 2005

Benchmark Conclusions

- Columnar is a little slower to load, but much faster on queries
- 2.3x faster on simple single column scans
- 3.7x on simple aggregations
- 7.2x on an analytics query with a 3-table join



Other Open File Formats



STREAMING

Apache Avro

Row-Oriented: Compact binary format optimized for rapid writes.

Schema Evolution: Resilient schema-with-data JSON registry contracts.

Primary Role: Industry standard for Kafka and event streams.



AT-REST

ORC

Storage Pruning: Dictionary encoding and columnar min/max statistics.

High compression, primarily legacy Hadoop/Hive.



IN-MEMORY

Apache Arrow

In-Memory Only: Standardized RAM format; not an on-disk file.

Zero-Copy IPC: Shares memory across engines without serialization.

Modern Engines: Backbone for DuckDB, Polars, and Arrow Flight.

Architectural Rule: Avro serializes streaming data in transit; Parquet stores analytical data at rest; Arrow executes in memory.

Who Supports What

- Apache Parquet: Supported universally by virtually every modern analytical query engine, cloud data warehouse (Snowflake, BigQuery, Redshift), and lakehouse platform (Databricks, Fabric) as the default open columnar standard.
- Apache ORC: Primarily supported by Hadoop-ecosystem engines like Apache Hive, Spark, Trino, and Presto, as well as an optional underlying storage option in Apache Iceberg.
- Apache Avro: Supported across the entire enterprise streaming and ingestion ecosystem, led by Apache Kafka, Confluent Schema Registry, event buses, and data integration platforms.
- Apache Arrow: Supported as the open standard for vectorized in-memory compute and inter-process exchange across modern engines and libraries like DuckDB, Polars, DataFusion, Spark, and language runtimes (Python, C++, Rust).

Summary



- The benefits of Open Data Formats include Interoperability, Flexibility and Scalability, Collaboration and Community, Cost-Effectiveness and Future-Proofing
- The limitations of Hive's approach led to the development of more modern open table formats like Apache Iceberg, Delta Lake, and Apache Hudi
- An Iceberg table consists of multiple components: data files, manifest files, manifest lists and metadata files; Each of these components plays a crucial role in the architecture
- Iceberg includes Schema Evolution, Partition Evolution and Time Travel
- Virtually every major analytical engine, cloud data warehouse, and lakehouse platform support Iceberg
- Tools like Databricks UniForm and Apache XTable translate metadata across formats, allowing a single copy of Parquet data to be read as either Delta or Iceberg
- Vendor competition has shifted up the stack from file formats to governance and catalog layers, coalescing around the Iceberg REST Catalog API standard and open engines like Apache Polaris and Unity Catalog
- Parquet is an Open File Format that is columnar, has great compression, is language agnostic, open source and supports complex data types



How to Work with Open Table Formats

Presented by: William McKnight
President, McKnight Consulting Group
3 X Inc 5000

 /in/wmcknight
www.mcknightcg.com
(214) 514-1444

