Beyond Hive: The Open Table Format Revolution

Erasmus Data Collaboratory | House of AI

THE LEGACY BOTTLENECK

(The "Data Swamp")

Rigidity and Unsafe Writes

Legacy Hive tables lack ACID compilance, meaning falled writes can lead to silent data corruption.

The "Rewrite" Nightmare

Simple partition changes (e.g., Month to Day) require expensive, full table rewrites in legacy systems.

Performance Latency

Slow directory-based scans create massive overhead as data volumes grow into the patabyte range.

THE MODERN SOLUTION

(The "Lakehouse Vision")

Scale Meets Governance

Lakehouses combine low-cost cloud storage with the ACID transactions and governance of a database.

The Metadata "Secret Sauce"

Formats like Iceberg and Delta Lake use a metadate layer to enable features like Time Travel.

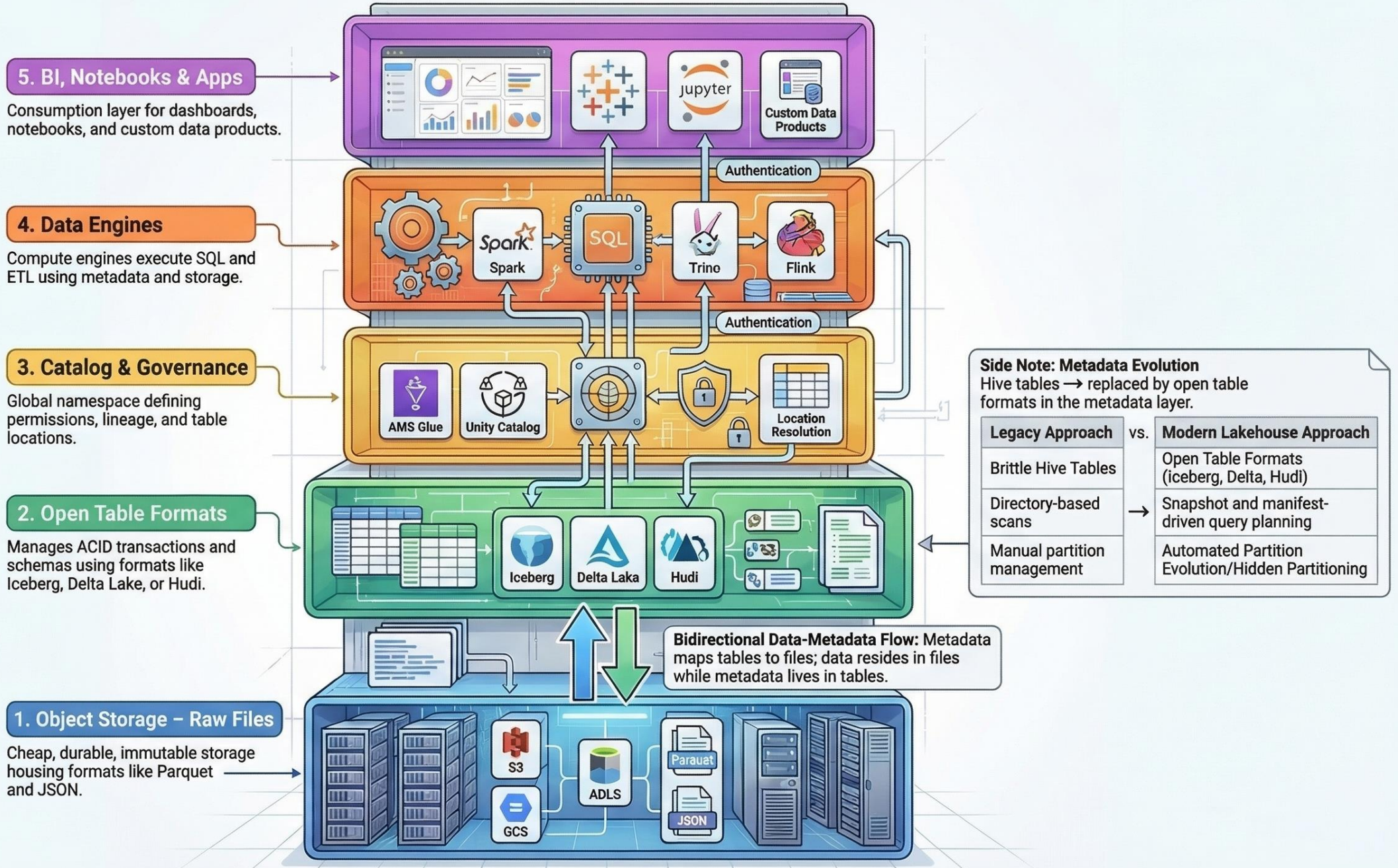
Multi-Engine Freedom

Open formats allow the same data to be accessed by Spark, Trino, Flink, and DuckDB simultaneously.

Jos van Dongen
Director Erasmus Data Collaboratory | House of AI

Metadata-Enabled, High-Performance Data Architecture with Open Table Formats. | Reliable. Scalable. Open.

Modern Data Lakehouse Architecture (Open Table Formats)



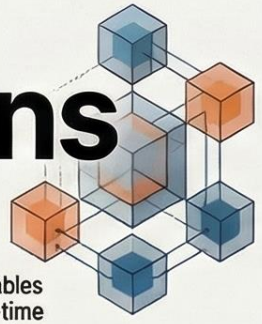
Lakehouse at Scale: Industry Giants & Open Format Metrics

Enterprises are shifting away from legacy Hive-based data lakes toward open table formats like Apache Iceberg and Delta Lake. This transition enables warehouse-like reliability (ACID transactions, schema evolution) at a petabyte scale while drastically reducing costs and latency.

Apache Iceberg Implementations

Apple: Billions of events per day

Billions



Manages petabyte-scale tables across all divisions for real-time streaming and ETL workloads.

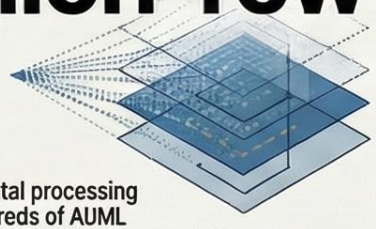
Pinterest: 24-hour latency cut to 15 minutes

24h to 15m

Consolidated event logging into Iceberg, saving $1.5MM in annual infrestructure costs.

Netflix: Trillion-row data scale

Trillion-row



Uses incremental processing to power hundreds of AUML applications and global content insights.

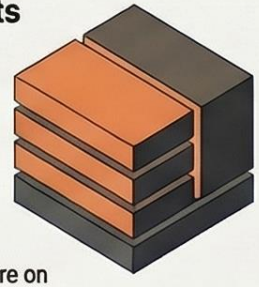
Comparing high-volume event processing across Iceberg adopters

Company	Daily Metric	Key Outcome
Adobe	32 Billion Events	Eliminated data corruption via schema enforcement
Skyscanner	30 Billion Events	Simplified analytical infrastructure for AI governance

Delta Lake & Data Intelligence

Block: 12x reduction in computing costs

12x



Standardized infrastructure on a Data Intelligence Platform to accelerate GenAI onboarding.

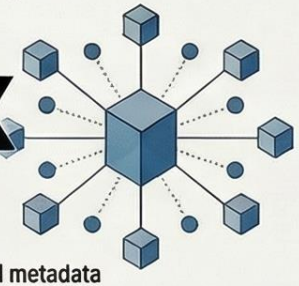
Itaú: 70% faster data availability

70%

Processes 650,000+ daily events with transactional data reaching the mesh in under 10 minutes.

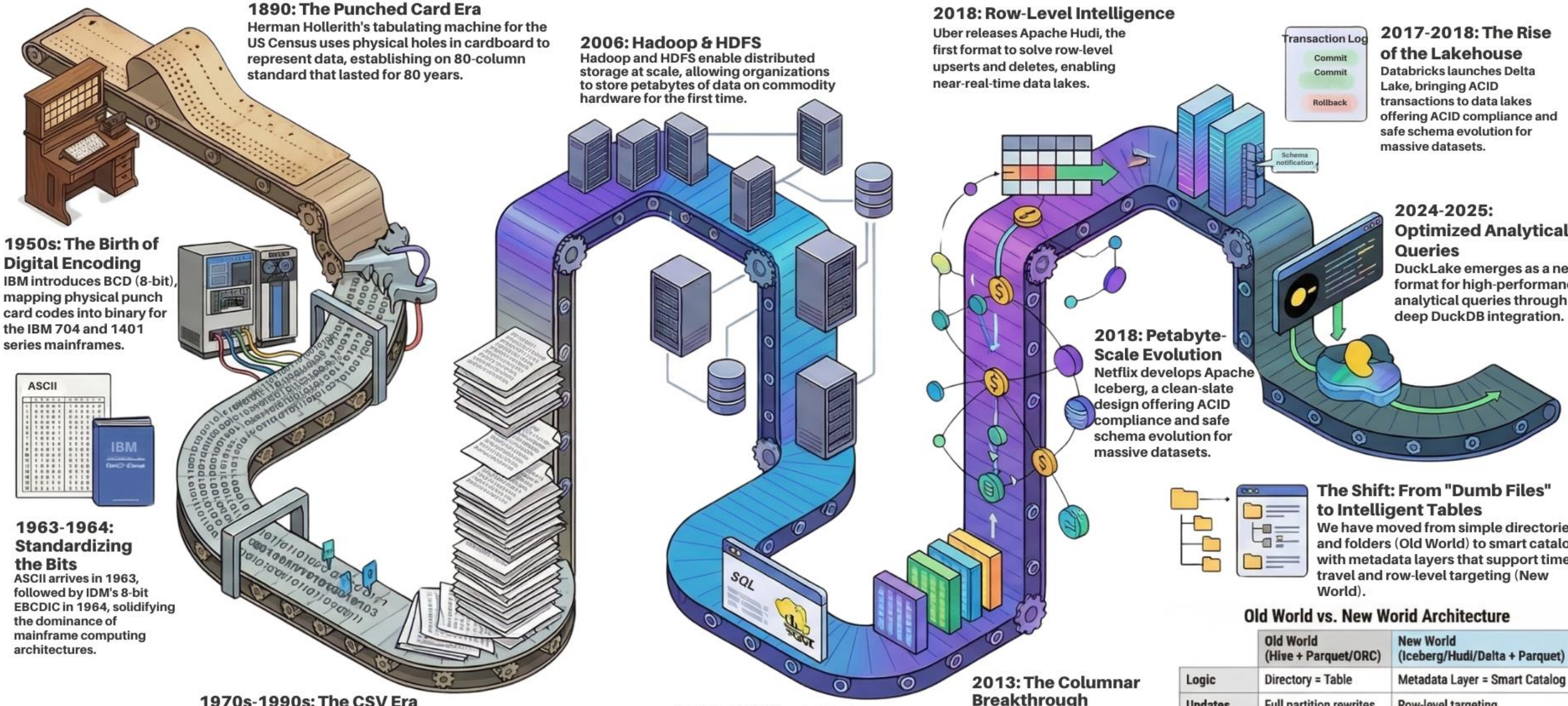
Unilever: 10x increase in development speed

10x



Utilizes a centralized metadata framework to serve over 5,000 internal users.

The Great Data Conveyor: 134 Years from Punched Cards to Intelligent Tables



1890: The Punched Card Era

Herman Hollerith's tabulating machine for the US Census uses physical holes in cardboard to represent data, establishing on 80-column standard that lasted for 80 years.

1950s: The Birth of Digital Encoding

IBM introduces BCD (8-bit), mapping physical punch card codes into binary for the IBM 704 and 1401 series mainframes.

1963-1964: Standardizing the Bits

ASCII arrives in 1963, followed by IDM's 8-bit EBCDIC in 1964, solidifying the dominance of mainframe computing architectures.

1970s-1990s: The CSV Era

Simple, human-readable text files become the norm, however, they lack schemas, compression, and support only sequential reading.

2006: Hadoop & HDFS

Hadoop and HDFS enable distributed storage at scale, allowing organizations to store petabytes of data on commodity hardware for the first time.

2008: SQL Meets Hadoop

Apache Hive introduces SQL-like querying to Hadoop, though it still relles on "dumb" file storage like CSV and sequence files.

2013: The Columnar Breakthrough

The introduction of Parquet and ORC formats deliver 10x to 100x compression, optimized specifically for Hive and analytical workloads.

2018: Row-Level Intelligence

Uber releases Apache Hudi, the first format to solve row-level upserts and deletes, enabling near-real-time data lakes.

2017-2018: The Rise of the Lakehouse

Databricks launches Delta Lake, bringing ACID transactions to data lakes offering ACID compliance and safe schema evolution for massive datasets.

2018: Petabyte-Scale Evolution

Netflix develops Apache Iceberg, a clean-slate design offering ACID compliance and safe schema evolution for massive datasets.

2024-2025: Optimized Analytical Queries

DuckLake emerges as a new format for high-performance analytical queries through deep DuckDB integration.

The Shift: From "Dumb Files" to Intelligent Tables

We have moved from simple directories and folders (Old World) to smart catalogs with metadata layers that support time travel and row-level targeting (New World).

Old World vs. New World Architecture

	Old World (Hive + Parquet/ORC)	New World (Iceberg/Hudi/Delta + Parquet)
Logic	Directory = Table	Metadata Layer = Smart Catalog
Updates	Full partition rewrites	Row-level targeting
Consistency	"Hope and pray"	ACID via transaction logs
Changes	Hard schema changes	Safe schema evolution
History	No history/snapshots	Time travel (Any snapshot)

The ‘Why’ and ‘How’ of Open Table Formats: From Raw Files to Reliable Lakehouses

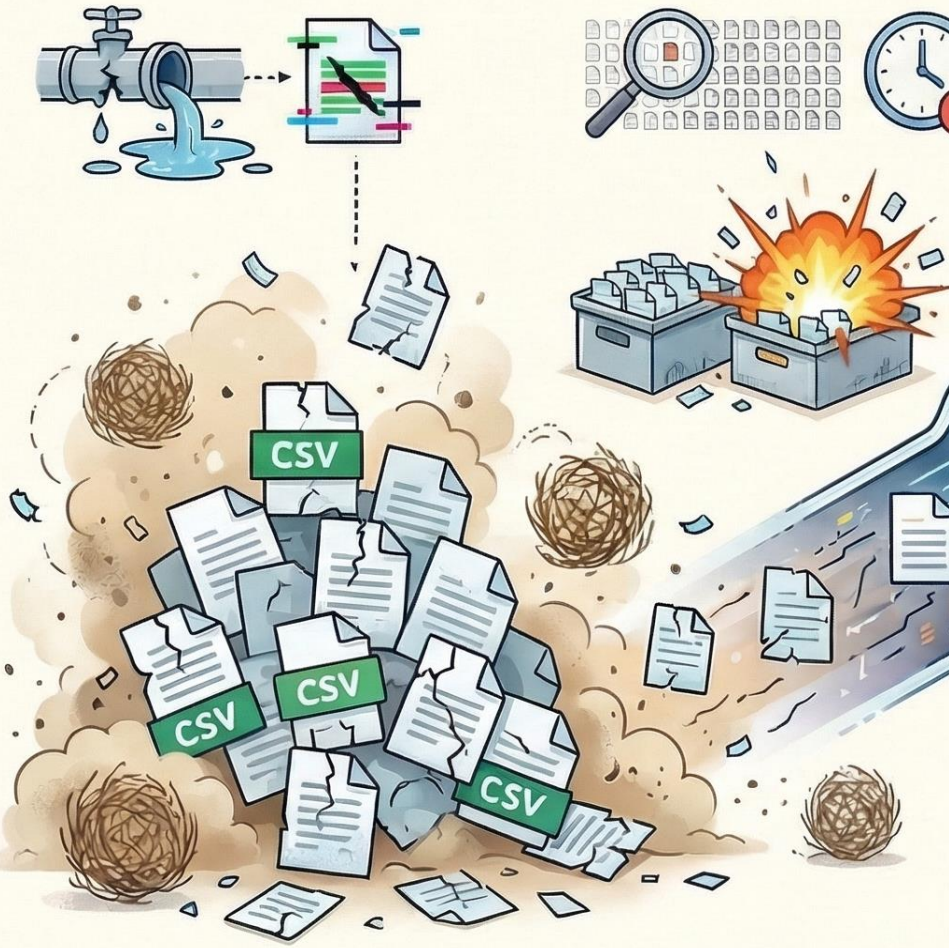
THE ‘WHY’—FIXING THE BRITTLE DATA LAKE

The “Wild West” of Raw Files

Raw files lack transactions, causing corrupted data when multiple jobs write simultaneously.

High Maintenance & “Small-File Explosions”

Changing schemas or deleting single rows requires rewriting entire partitions, causing performance bottlenecks.



THE BRAINS OF A DATABASE

Table formats turn piles of files into reliable tables using metadata and protocols.

THE ‘HOW’—CRITICAL LAKEHOUSE CAPABILITIES

ACID Transactions & Schema Evolution

Ensures concurrent write reliability and allows renaming fields without breaking downstream queries.

Time Travel & Efficient Querying

Enables auditing historical data versions and uses metadata to prune unnecessary file scans.

Multi-Engine Interoperability

Allows the same table to be accessed by Spark, Flink, Trino, and DuckDB.

Table Format Comparison: At a Glance

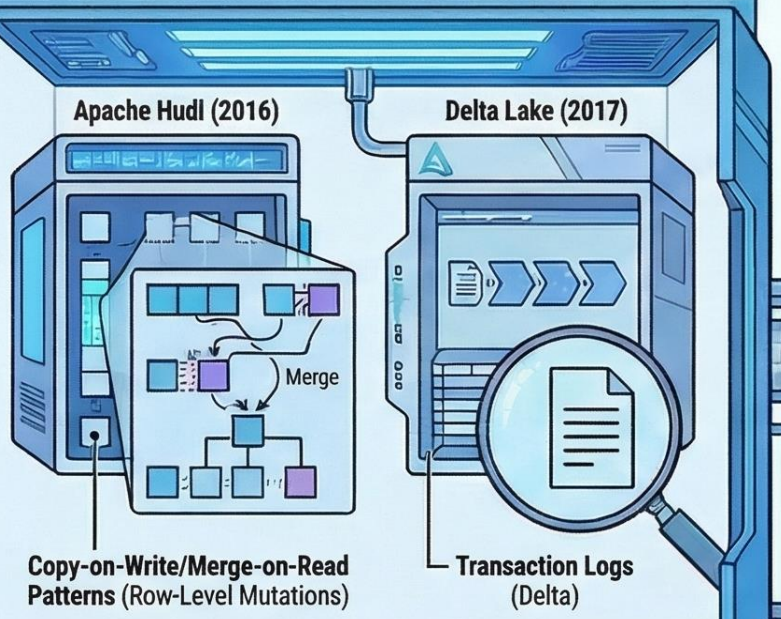
Format	Primary Optimization	Key Strength
Iceberg	Large-scale Analytics	Engine-agnostic and wide industry adoption.
Delta Lake	Batch & Streaming	Deep integration with Spark and Databricks.
Hudi	Upserts & CDC	Built for near-real-time ingestion and incremental processing.
Paimon	Streaming-first	LSM-tree design for high-velocity IeT and COC updates.
DuckLake	Metadata Simplicity	Stores metadata in a SQL database for fast commits.

The Three Waves of Open Table Format Innovation: From Hive to Modern Lakehouses

Wave 1 — The Incremental Pioneers (2016–2018)

Fixing Hive with Transactionality

Addressed Hive's inability to handle updates/deletes and rewrites for small changes.

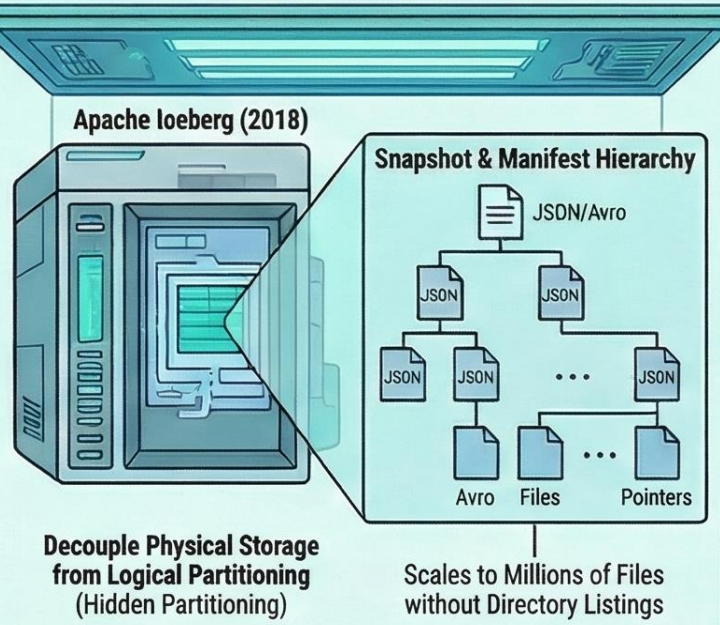


Philosophy of "Fixing Hive":
Add transactionality to existing batch-centric infrastructures.

Wave 2 — The Universal Standard (2018–2022)

Engine Neutrality & Metadata Scalability

Solved Wave 1 engine lock-in and metadata bottlenecks at petabyte scale.

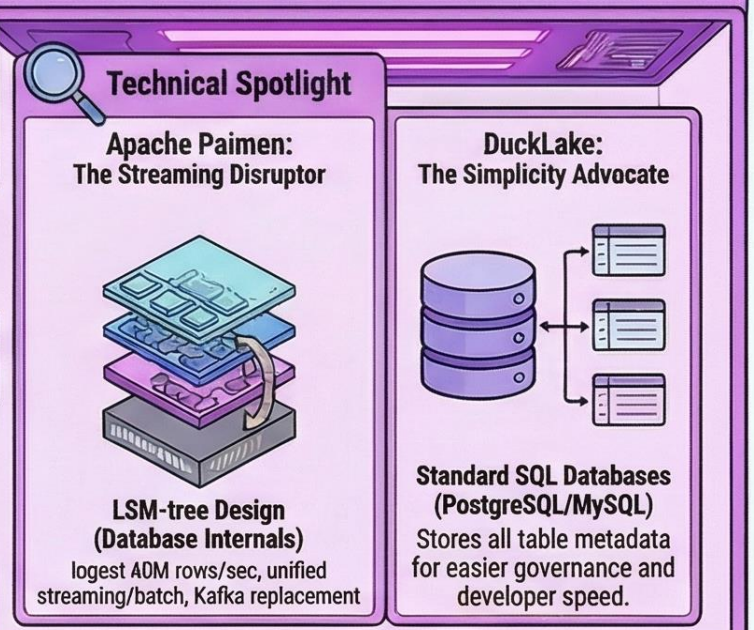


Philosophy of "Outlast Any Engine":
Vendor-neutral standard for AWS, Google, Snowflake, Databricks.

Wave 3 — Streaming Disruptors & Simplicity (2022–Present)

Rethinking Core Assumptions

Addresses Wave 2's latency struggles and high metadata complexity for smaller teams.



Philosophy of Radical Efficiency:
Prioritizes sub-minute freshness and ease of use over batch-centricity.

Positioning & Decision Guide

Wave	Format	Best For	Key Advantage
1	Hudi	CDC, AWS Stacks, Audit Trails	Proven incremental pipelines
1	Delta Lake	Databricks/Spark Ecosystems	Deep Spark integration
2	Iceberg	Interoperability, Large Batch	Engine-agnostic, safe bet
3	Paimon	Sub-minute freshness, Flink	LSM-tree for high-velocity streeming
3	DuckLake	Greenfield, Dev Velocity	Radical simplicity via SQL metadata

2016–2018 (Wave 1) 2018–2022 (Wave 2) 2022–Present (Wave 3)

Navigating the Lakehouse: A Guide to Open Table Formats

THE FORMAT COMPARISON MATRIX

Metadata Architecture Differences

Formats range from file-based manifests to sequential logs and relational databases.

Row-Level Strategy: COW vs. MOR

Formats use Copy-on-Write (COW) or Merge-on-Read (MOR) to balance write speed against query performance.

	Apache Iceberg	Delta Lake	Apache Hudi	Apache Paimon	DuckLake
	Snapshot/Manifests	Transaction Log	Commit Timeline	LSM-Tree Design	SQL Database
	COW + Delete Files	COW + Deletion Vectors	COW or MOR	Native MOR	COW (Fast Commits)
	Engine-Agnostic (Trino, Snowflake)	Spark & Databricks	AWS & CDC Ingestion	Flink & Streaming	DuckDB & MotherDuck

STRATEGIC SELECTION GUIDE

Choose Iceberg for Universal Neutrality

Best for organizations requiring the widest vendor support and cross-engine interoperability.

Choose Deita Lake for Databricks Centricity

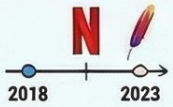
Ideal for teams heavily invested in Spark and Databricks' optimized performance features.

Choose Paimon or Hudi for Real-Time Needs

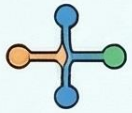
Best for high-velocity streaming, CDC ingestion, and sub-minute data freshness.

ANATOMY OF THE UNIVERSAL STANDARD: INSIDE APACHE ICEBERG

CORE IDENTITY & ORIGIN



HEADLINE: Born at Netflix (2018)
SUPPORTING DETAIL: Created to fix Hive's sealability issues and dunatnd to the Apache Software Foundation in 2019.

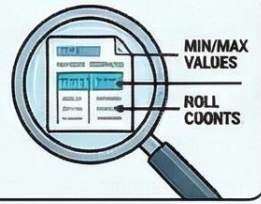


HEADLINE: Engine-Agnostic Philosophy
SUPPORTING DETAIL: Designed with an architecture intended to outlast any single compute platform or query engine.

HEADLINE: Patakyto-Scale Maturity
SUPPORTING DETAIL: Production-proven across hundreds of major organizations including Apple, Adobe, and Linkedin.

Adobe

HEADLINE:
MANIFESTS & DATA FILES
SUPPORTING DETAIL:
Manifests track individual data files (Parquet/DRC) and provide column level stats for efficient pruning.



TRADE-OFFS & OPERATIONAL REALITIES

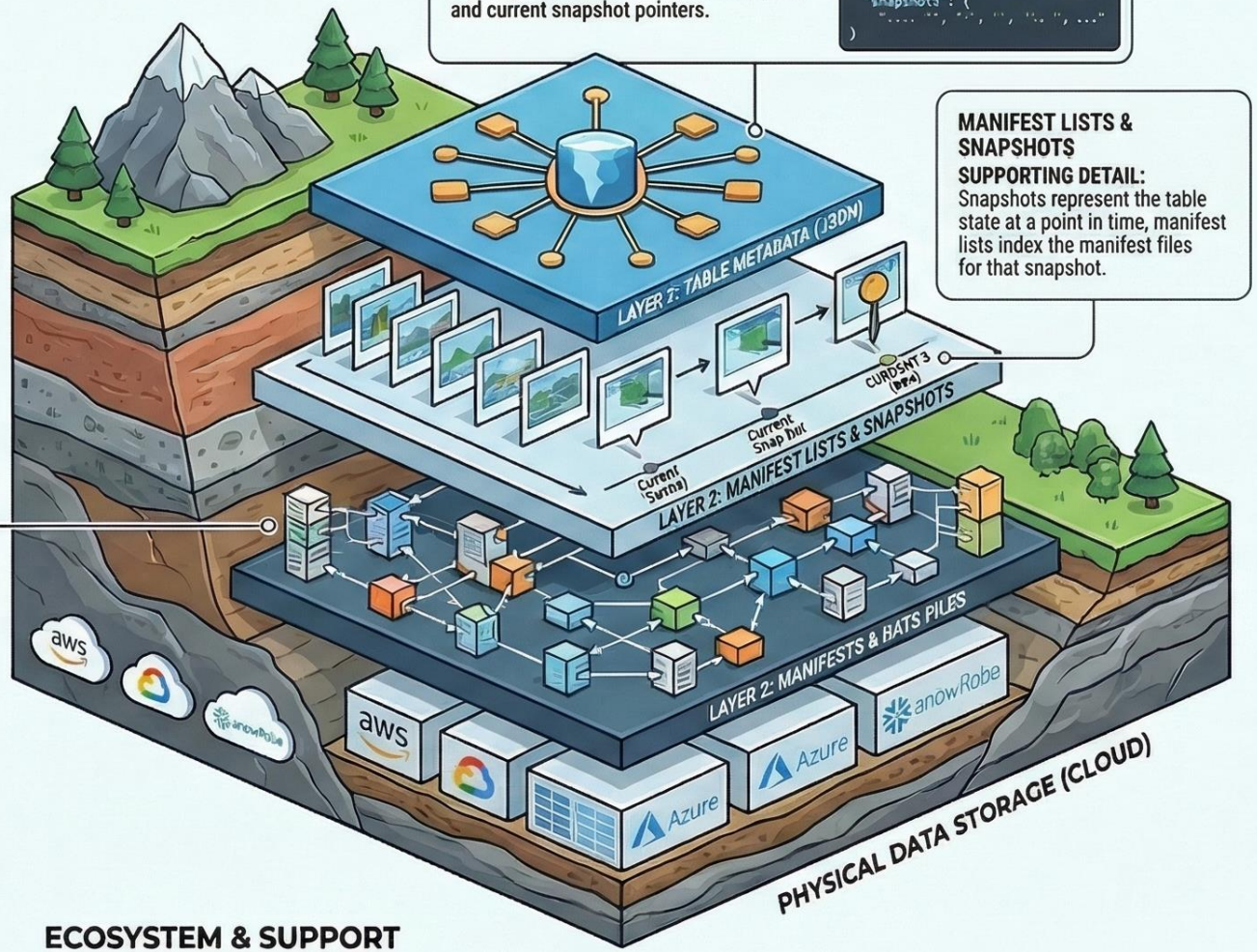
HEADLINE: Batch-Optimised Design
SUPPORTING DETAIL: Not ideal for sub-minute streeming latency requirements where formats like Paimon may exeel.

HEADLINE: Operational Maintenance
SUPPORTING DETAIL: Requires active orchestration for snapshot expiration and "orphan file" cleanup to manage storage costs.

HEADLINE: Metadats Complexity
SUPPORTING DETAIL: The three layer hierarchy offers high performance but reguires toning and deeper understanding for troubleshooting.

HEADLINE: TABLE METADATA (JSON)
SUPPORTING DETAIL: The central hub tracking schema versions, partition specs, and current snapshot pointers.

MANIFEST LISTS & SNAPSHOTS
SUPPORTING DETAIL:
Snapshots represent the table state at a point in time, manifest lists index the manifest files for that snapshot.



ECOSYSTEM & SUPPORT

Spark trino

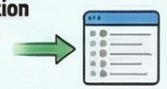
HEADLINE: Broadest Engine Support
SUPPORTING DETAIL: Native, first-class integration with Spark, Flink, Trino, Presto, Athona, Snowflake, and BigQuery.

HEADLINE: Zero Vendor Lock-in
SUPPORTING DETAIL: Adopted as a native format by AWS, Google Cloud, and Snowflake, ensuring long-term data portability.

HEADLINE: Governance & Tooling
SUPPORTING DETAIL: Supported by robust cataiog services like Apache Polaris, Nessle, and AWS Glue.

EVOLUTIONARY STRENGTHS

HEADLINE: Comprehensive Schema Evolution
SUPPORTING DETAIL: Add, drop, rename, or reorder cotionns using unique IDs– never breaking downstream queries.



HEADLINE: Hidden Partitioning
SUPPORTING DETAIL: Decouples physical layout from ingical auetev, users dan't need to know the partition column to query efficiently.

HEADLINE: True Partition Evolution
SUPPORTING DETAIL: Change partitioning schemes (e.g., Daily to Houris) without rewriting existing historical data.

HEADLINE: Point-in-Time "Time Travel"
SUPPORTING DETAIL: Snapshot-based queries allow for historical analysis, auditing, and easy data rollbecks.

DECISION MATRIX: WHEN (AND WHEN NOT) TO CHOOSE

IDEAL: Multi-Engine Bata Platforms (→)
SUPPORTING DETAIL: Best when Spark, Trino, and Flink must all interact with the same tables simultaneously.

IDEAL: Long-Term Modernization (→)
SUPPORTING DETAIL: Perfect for organizations requiring decades of backward compatibility and future-proofing.

AVDID: Streaming-First Needs (×)
SUPPORTING DETAIL: Avoid if your architecture requires sub-minute latency for real-time CDC.

AVDID: Simple/Small Use Cases (×)
SUPPORTING DETAIL: Overheed may exceed benefits for simple projects; consider lighter options like DuckLake.

BY THE NUMBERS

Motric	Value
Supported Engines	10+ Native Integrations
Cloud Support	AWS, GCP, Axure, Snowflake
Community Size	Largest contributor base in format category
Production Scale	Petabytes per organization

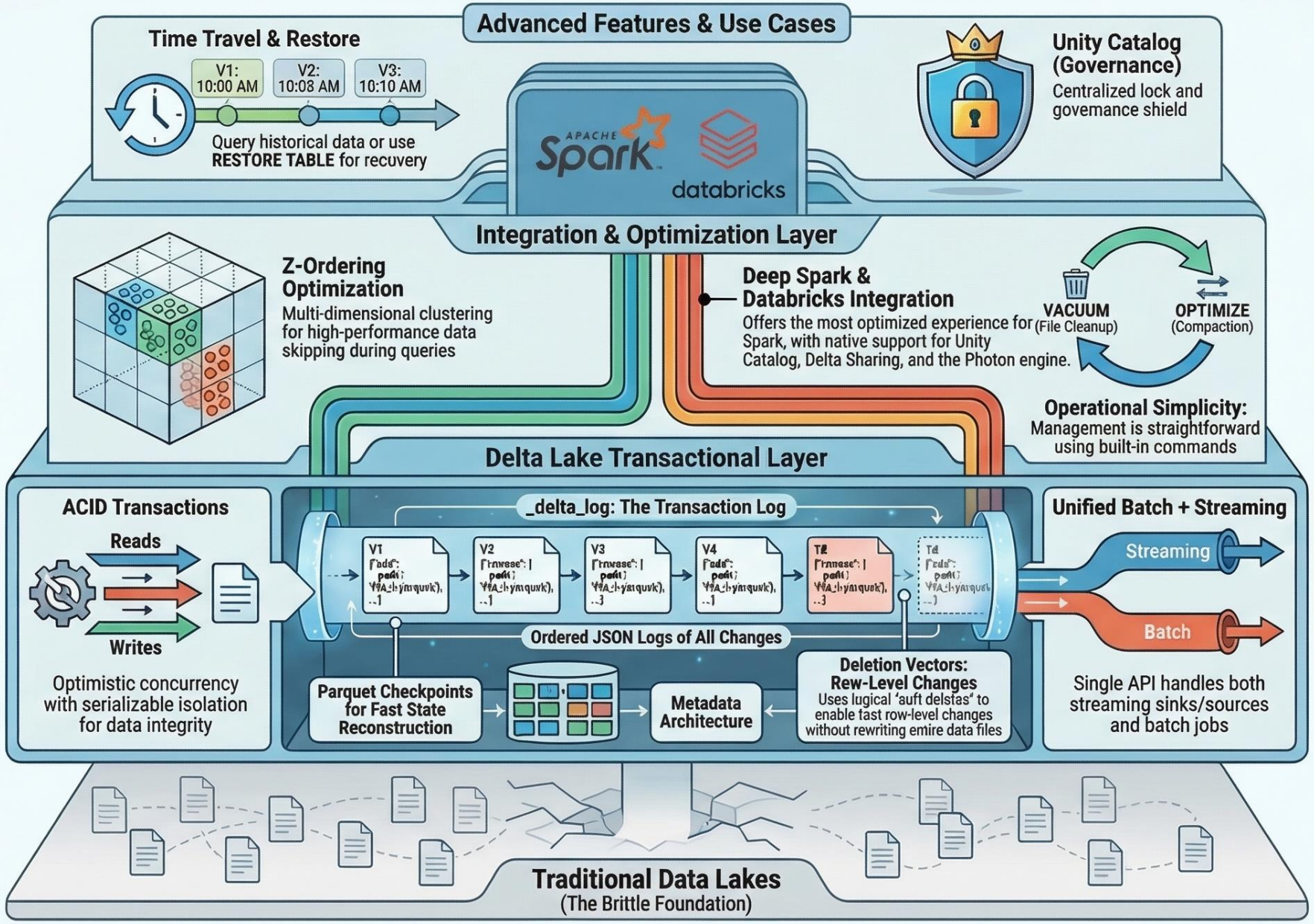
Delta Lake: The Spark-Native Transactional Layer

ACID Transactions Meet Data Lakes in the Databricks Ecosystem

Key Strengths & Ideal Use Cases

- **Choose Delta Lake For:** Databricks-native platforms, Spark-centric architectures, and an architecture running performant architecture.
- **Operational Simplicity:** Management is straightforward to compllate connorturiers sources and pretent gaming queries.
- **Choose Delta Lake For:** Databricks-native platforms, Spark-centric architectures, and an architecture running pertiform entise.

- **Thousands of Production Users:** Powered by Databricks for major organizations including Comcast, Shell, Regeneron, and ABN AMRD.
- **Minute-Level Latency:** Optimized for near real-time processing via Spark Structured Streaming.



Brittle Data Lakes
Lack Reliability & Integrity

Trade-offs & Limitations & When to Avoid

- **Databricks & Engine Contricity:** Best performance tied to Databricks; limited adoption outside Spark.
- **Evolution Constraints:** Schema and partition evolution are less comprehensive than Iceberg (e.g., no column reordering).
- **GDPR Compliance Workflow:** Hard deletes require multi-step process (REORG & VACUUM).
- **Avoid Delta Lake If:** You require a cloud-agnostic strategy, multi-engine support (Trino/Flink), or sub-minute streaming without Spark.

Apache Hudi: The Incremental Pioneer

VISUAL CUES & PERFORMANCE SPECS

3–10 MINUTE LATENCY — Typical data freshness achieved in production environments (Minute-level, not sub-minute).

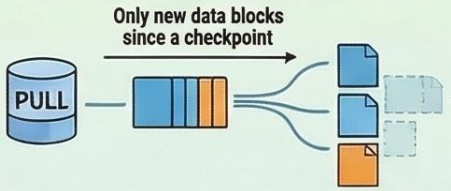
OPTIMIZED FOR FLINK 1.1+
Recent optimizations hove significantly improved throughput for Flink-based streaming ingestion.

KEY STRENGTHS & USE CASES

OPTIMIZED FOR CDC PIPELINES

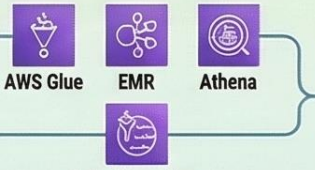
Purpose-built for capturing changes from transactional databases.

EFFICIENT INCREMENTAL QUERIES



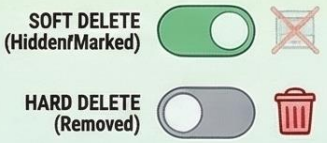
Enables "incremental pulls," allowing users to read only the specific records that have changed since the last checkpoint.

DEEP AWS INTEGRATION



Native, high-performance integration with AWS Gibs, SMR, Athene, and Lake Formation.

FLEXIBLE DELETE SEMANTICS



Explicit support for both Soft and Hard deletes, facilitating complex audit trads and compliance workflows.

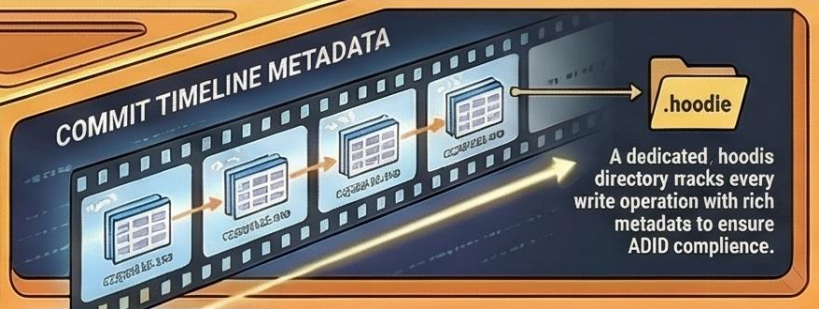
IDEAL USE CASES VS. WHEN TO AVOID

CHOOSE HUDI FOR: Neat Real-Time CDC
Suited for high-velocity ingestion and AWS-native service architectures.

AVOID HUDI FOR: Sub-Minute Latency
For streaming·first, sub-minute requirements, Psimon's LEM architecture is generally superior.

ARCHITECTURAL HIGHLIGHTS

COMMIT TIMELINE METADATA



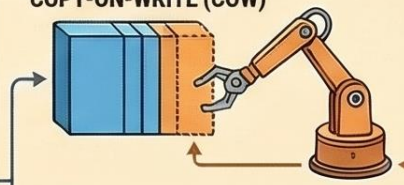
A dedicated .hoodis directory racks every write operation with rich metadats to ensure ADID complience.

NATIVE UPSERTS & INDEXING

Uses Bloom filters and HBase-style record-level indeving to locate and update specific records rapility.

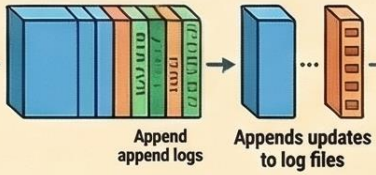
COW VS. MOR STORAGE TYPES

COPY-ON-WRITE (COW)



Primary Goal:	Read Performance
Write Latency:	Higher (Rowrites files)
Data File Format:	Parquet only
Read Latency:	Lower

MERGE-ON-READ (MOR)



Primary Goal:	Write Freshness
Write Latency:	Looas (Appends to logs)
Data File Format:	Pamuel A Acre (Log)
Read Latency:	Higher (Marge required)

TRADE-OFFS & LIMITATIONS

- **Spark-Centric Ecosystem** (best performance with Spark)
- **The GDPR Trap** (PII in (requires active management)
- **Limited Schema Evolution** (no column renaming)
- **ACID Snapshot Isolation:** Supports multi-version concurrency control, ensuring consistent reads even during high-frequency writes.

CORE IDENTITY & ORIGINS

CREATED BY UBER (2016)

Hudi was the first open table format designed upecifically to address hilve's limitations regarding data updates and deletes.

THE "INCREMENTAL" PHILOSOPHY

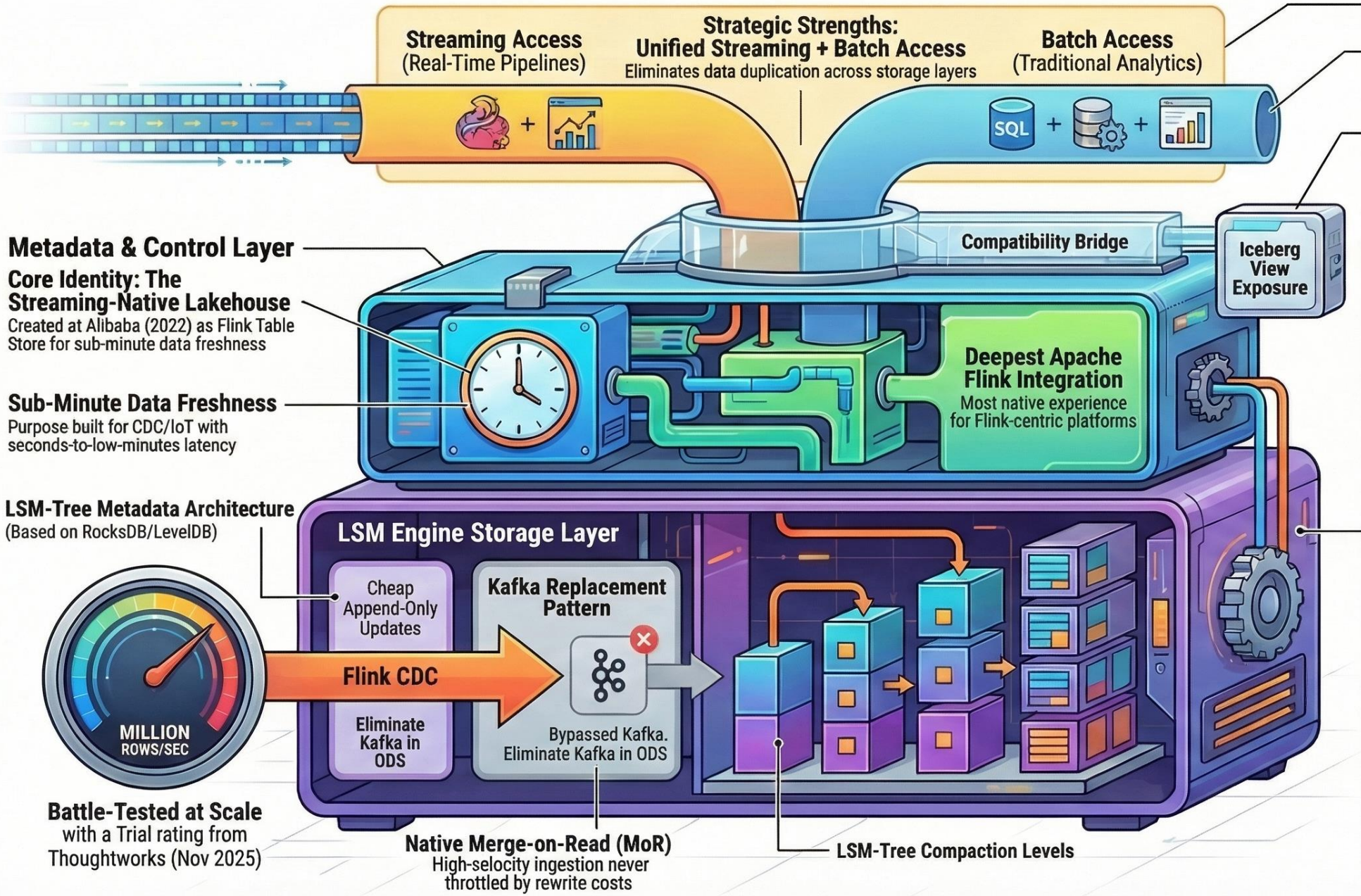
Built to bring database-style upsert semantics to data lakes, allowing for efficient, record revel data management.

PRODUCTION-PROVEN MATURITY

amazon Uber Robinhood Disney+

Extensively used in high-velocity CDC workloads by major organizations.

Apache Paimon: The Streaming Disruptor



Practical Trade-offs & Limitations

- **Operational Complexity vs. Iceberg:** Requires deep Flink expertise, compaction for deletions
- **Higher Cloud API Costs:** S3 API calls ~3,000/hr, significanity higher than iceberg
- **Young Ecosystem:** Limited schema evolution, fewer third-party governance tools

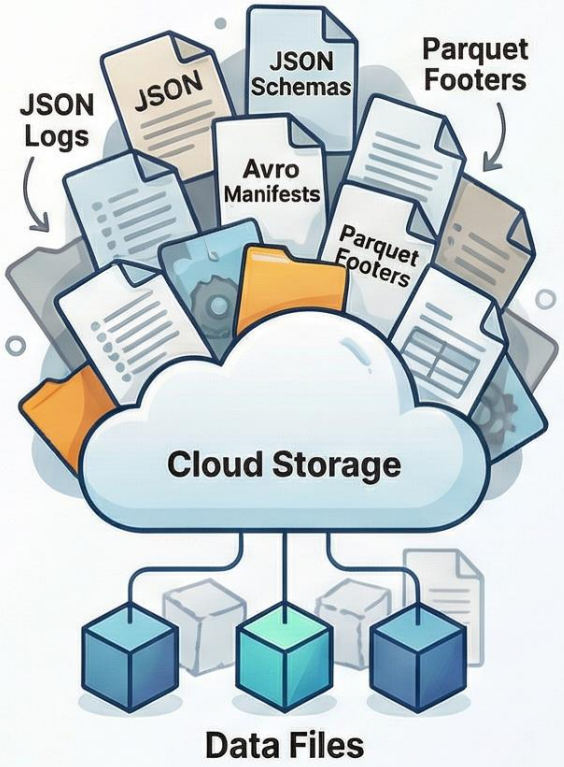
Usage Matrix

Choose Paimon If...	Avoid Paimon If...
✓ Requirement is sub-minute freshness	✗ Workloads are pri-marrily batch-oriented
✓ Running high-velocity CDC/IoT streams	✗ Team lacks deep Apache Flink expertise
✓ Eliminating Kafka from ODS is a goal	✗ Broad support for SnowflakorBigQuery is required
✓ Using Flink as the primary compute engine	✗ Frequent complex schema reordering is needed

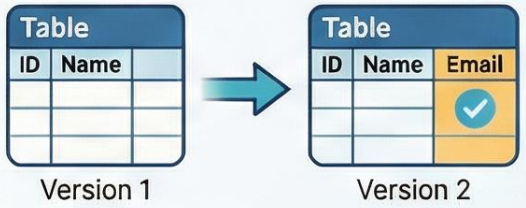
DuckLake: Radical Simplicity via SQL-Native Metadata

THE DISRUPTOR: The "File-Heavy" Way (Messy Piles)

Fragmented Metadata, Slow Query Planning, Complex Rewrites



SCHEMA EVOLUTION: The "No-Rewrite" Advantage



NO REWRITE: Zero impact on physical data in the lake

SQL-Based Schema Versioning: Logical changes only

THE DUCKLAKE WAY (Radical Simplicity)

INSTANT METADATA-ONLY UPDATE

COMPUTE ENGINE (DuckDB)

Transaction Protocol: Multi-Table ACID transactions

SQL CATALOG DATABASE (Unified Metadate)

Stores all metadata in PostgreSQL or DuckDB. Zero Data Movement, Instant Updates

STORAGE LAYER: IMMUTABLE PARQUET

Raw data on S3/GCS/R2. Portability, High-Performance Reads

TIME TRAVEL: Navigating History via SQL

SNAPSHOT CATALOGING: Every write creates a new, immutable snapshot entry in SQL

Log Replay vs. SQL Querying

Instant "Point-in-Time" Access: Run "AS OF" queries to audit or recover instantly

FASTER QUERY PLANNING

Skip directory scanning; retrieve file locations directly from SQL

MULTI-TABLE ATOMICITY

Commit changes across multiple tables simultaneously via database transaction

UNIFIED SINGLE-FILE SYSTEM

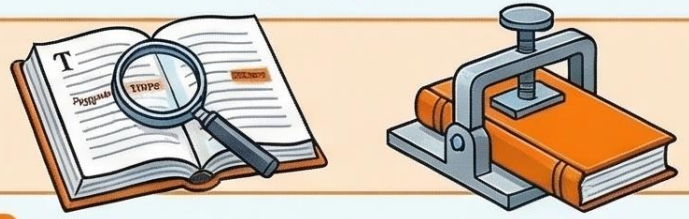
Replace fragmented files with a queryable single-file or database-backed system

Copy-on-Write vs Merge-on-Read: How Table Formats Handle Updates

Exploring Data Lake Table Format Strategies for Immutable Storage

Copy-on-Write (CoW) Analogy

Think of it as reprinting an entire book every single time you need to fix a single type in one chapter.



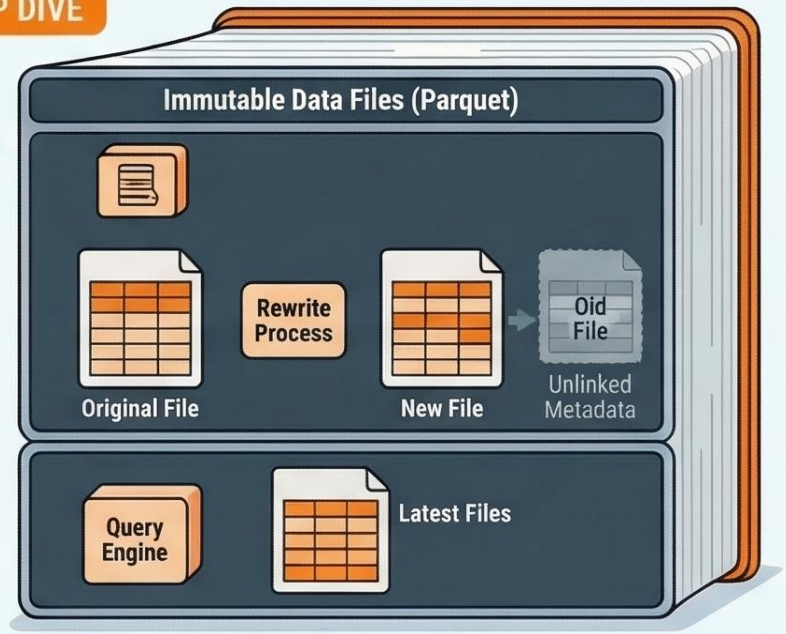
Merge-on-Read (MoR) Analogy

Think of it as keeping a stack of sticky notes with corrections inside the original book, only reprinting the book occasionally.

COPY-ON-WRITE (CoW) DEEP DIVE

The Write Path: Full Rewrite

When a row is **UPDATED** or **DELETED**, the system identifies the file containing that row and rewrites the entire file with the changes applied, the old file is then unlinked in the metadata.



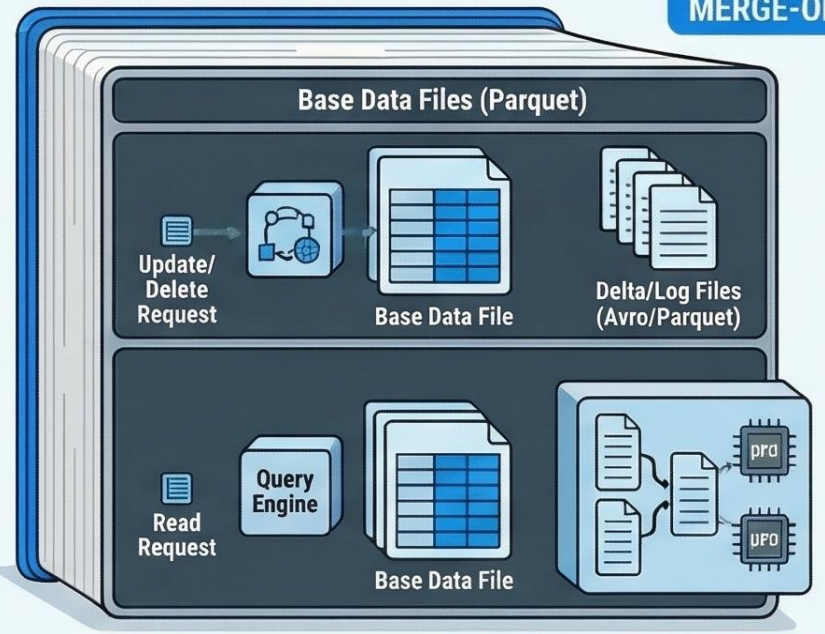
The Read Path: Direct Scan

Reads are simple and extremely fest because the engine only needs to seen the latest set of fully-formed Parquet files.

Trade-off: High Write Cost

Writing is expensive due to high "write amplification"—oven a single row change requires rewitting a large (e.g., 128MB) data file.

MERGE-ON-READ (MoR) DEEP DIVE



The Write Path: Delta Appends

Updates and deletes are written as small, separate "delta" or "log" files (oftan Avro or specialized Parquet), leaving the original base data files untouched.

The Read Path: On-the-Fly Merging

The engine must read the base file **AND** the deita files, merging them in memory to provide the correct result, which adds CPU overhead.

Compaction (The Cleanup)

Periodic background processes eventually marge the small delta files back into new base files to keep read performance from degrading.

COMPARISON AND IMPLEMENTATION

Aspect	Copy-on-Write (CoW)	Merge-on-Read (MoR)
Read Path	Simple, Direct, Fast	Complex (Merge required)
Write Cost	High (Heavy Amplification)	Low (Append-friendly)
Best For	Read-heavy batch analytics	Streaming, CDC, loT Ingestion
Ops Effort	Snapehot expiration	Compaction scheduling
Format Mapping	**Ieeberg** uses CoW by default (MoR in v2/v3); **Dalta Lake** uses CoW with "Deletion Vectors"; Nodí allows users to choose per table; **Paimon** is natively Molt using an LSM-tree design.	

DECISION GUIDE

When to Choose Copy-on-Write: Select CoW when reads dominate your workload, updates are rate/infrequent, or you are running heavy batch analytics.

The Key Takeaway: "Both strategies are valid—the right choice depends on your read/write mix, latency requirements, and operational complexity tolerance."

When to Choose Merge-on-Read: Select MoR for frequent updates, low-latency streaming ingestion (CBC), or when write latency is your primarv bottleneck.

Snapshot in Time: How Apache Iceberg Enables Data Time Travel

The Table Metadata File (JSON)
The central "source of truth" that tracks schema versions, partition specs, and—most importantly—the list of all valid Snapshots.

Current State

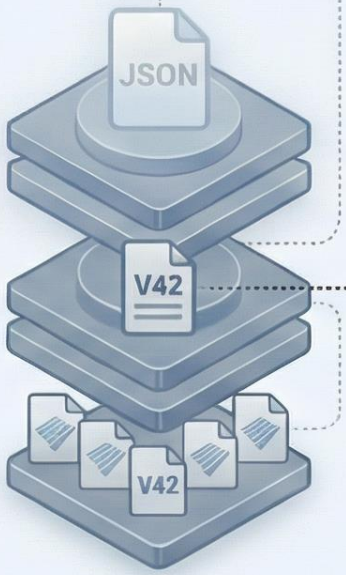
Historical State

Section 1: The Brains—The Metadata Hierarchy

Manifest Lists & Manifests (Avro)
These act as an index, the Manifest List points to individual Manifests, which in turn point to the actual Parquet data files and provide column-level stats.

Total Immutability
Every data file (Parquet) and metadata file (Avro) is immutable; once written, they are never modified, only replaced by newer pointers in the JSON file.

Instant & Atomic for new readers



Reader
(Querying the Past)

Section 2: How a New Version is "Shuffled" (The Write Process)

Writer
(Creating V43)

1. Write New Data Files

2. Create a New Manifest List

3. Atomic Pointer Swap

Section 3: Time Travel—Querying the Past

Jumping Back in Time
Compute engine from the current engine:

```
SELECT * FROM table VERSION AS OF 42
```

The compute engine ignors the current V43 pointer, and directly following the select V43 pointer.

Snapshot Isolation in Action
While a 'Writer' is creating Snapshot V43, a 'Reader" can still query Snapshot V42 simultaneously without seeing partial or corrupted data.

Zero Replay Required
Iceberg provides instant access to any historical state by simply reading that snapshot's manifest list.

Section 4: Why Time Travel Matters

Debugging & Recovery
If a pipeline accidentally overwrites emails with NULLs, engineers can query the last "good" snapshot to restore data.

Regulatory Compliance & Auditing
Organizations can prove exactly what data was used for a specific report or model on a specific date.

Safe Machine Learning
Data scientists can "pin" a model to a specific snapshot to ensure experiments are reproducible even as the underlying table continues to update.

GDPR & The Hard Truth of Hard Deletes: From 'Row Deleted' to 'Bits Gone from Disk'

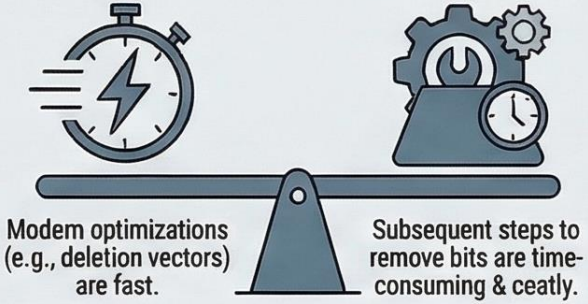
GDPR: The Great Format Equalizer

Every format must answer:
How to go from "deleted" row to physical bits gone?

The Performance vs. Physicality Trade-off

Initial Logical Delete (Fast)

Physical Hard Delete (Expensive & Slow)



Modem optimizations (e.g., deletion vectors) are fast.

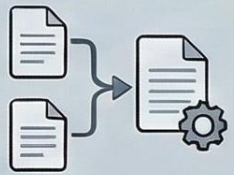
Subsequent steps to remove bits are time-consuming & ceatly.

The 4-Stage Compliance Lifecycle

Stage 1: Logical Delete (Fast)

Metadata-only operation. Enables time travel/snapshots, hides data from active queries.

Stage 2: Compaction & Rewrite (Heavy Compute)



Merges delete metadata & physically rewrites files into new versions without deleted records.

Stage 3: Retention Expiration (Metadata Cleanup)

Drops old snapshots & historical versions pointing to sensitive data files.

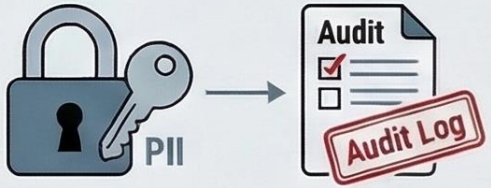
Stage 4: Physical Garbage Collection

Actual removal of abandoned, "orphan" files from object storage. Bits truly gone.

Format-Specific Workflow Comparison

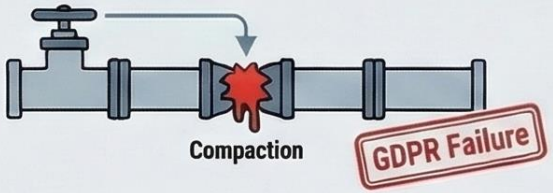
Format	Deletion Mechanism	Workflow Summary	Risk Level
Iceberg v3	Deletion Vectors	Vectors , Compaction , Snapshot Expiration , Orphan Cleanup	Moderate
Delta Lake	Deletion Vectors	Vectors , REORG , VACUUM operations , Physical Removal	Moderate
Apache Hudi	Soft vs. Hard Semantics	Choose Soft (Audit) or Hard (Immediate) , Compaction , Cleaning	HIGH
Apache Paimon	LSM-tree + Vectors	Append to LSM , Mandatory Compaction , Dual Expiration , GC	HIGH
DuckLake	Delete Files (v2 Pattern)	Delete Files , Compaction , Retention Expiration	HIGH

Apache Hudi: The PII Key Trap



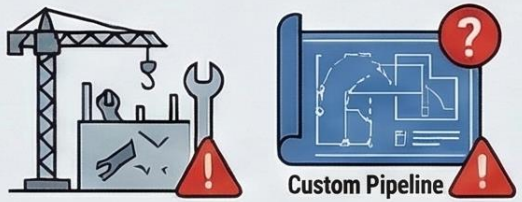
If Primary Keys contain PII, Hudl's "soft delete" audit trail becomes a legal liability under GDPR.

Apache Paimon: The Compaction Dependency



Paimon's Dual Retention Configs: Compliance requires independent management of both snapshot and changelog retention settings to ensure data is fully purged.

DuckLake: Governance Immature



Lacks mature automated governance tooling, requiring custom, high-risk compliance pipelines.

The Hidden Costs of Compliance & Key Takeaway

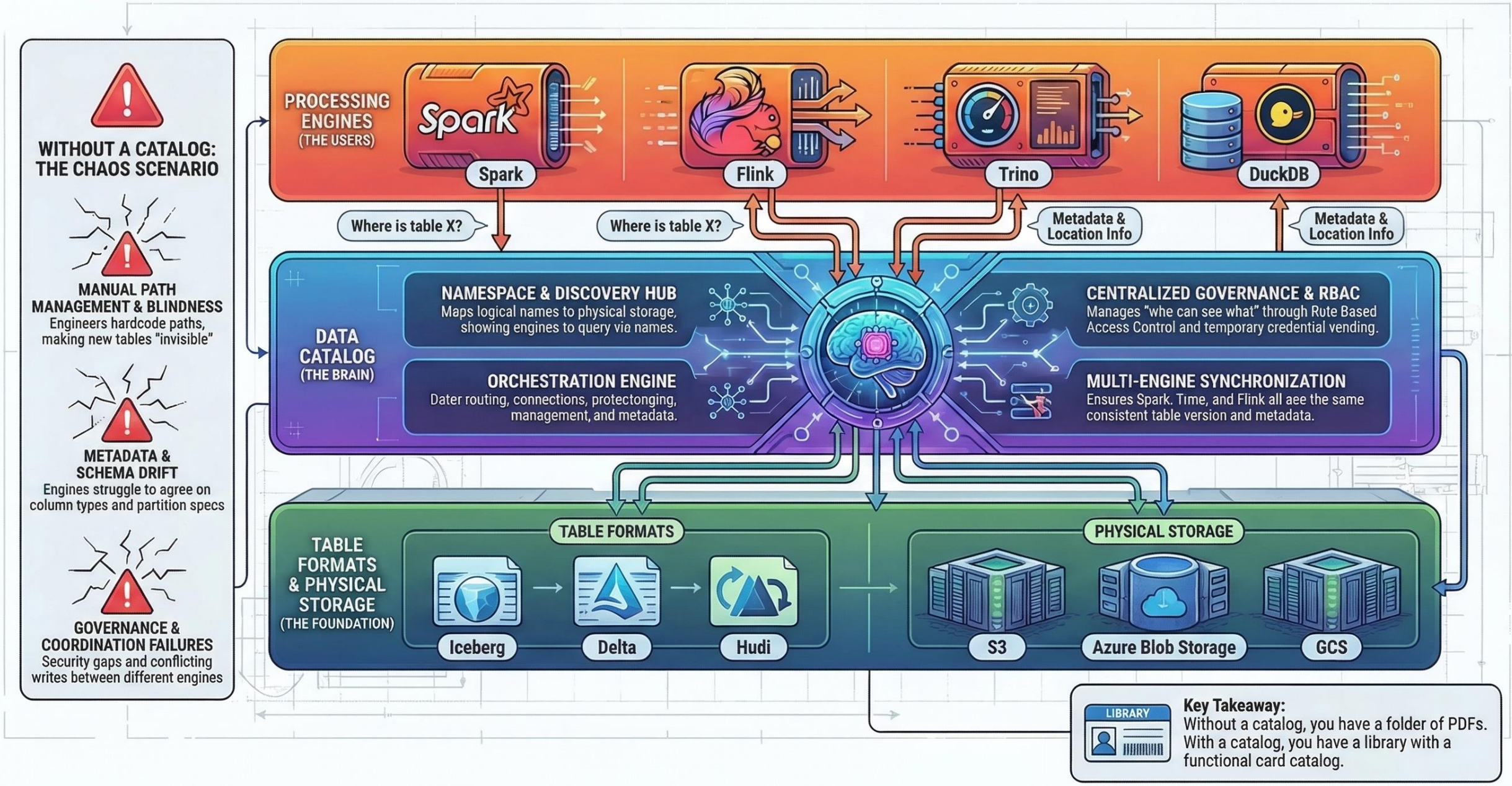
SLA Enforcement Gap & Metadata Housekeeping

No "free" compliance; requires active monitoring & pipelines. Orchestrated cleanup of manifest files, transaction logs, catalogs needed.

Operationalize the Full Pipeline

> *"Choose your format based on your team's ability to operationalize the full hard-delete pipeline, not just the initial soft delete performance."*

The Lakehouse Brain: Why Data Catalogs are Essential



The Three Waves of Data Catalog Evolution: From Metadata to Versioning

"Wave 1 gave us metadata. Wave 2 gave us governance. Wave 3 is giving us versioning and neutrality."
Select your catalog based on your specific governance needs and your tolerance for vendor lock-in.

"Just as table formats evolved in three waves, so did catalogs—from centralized databases to git-like workflows for data."

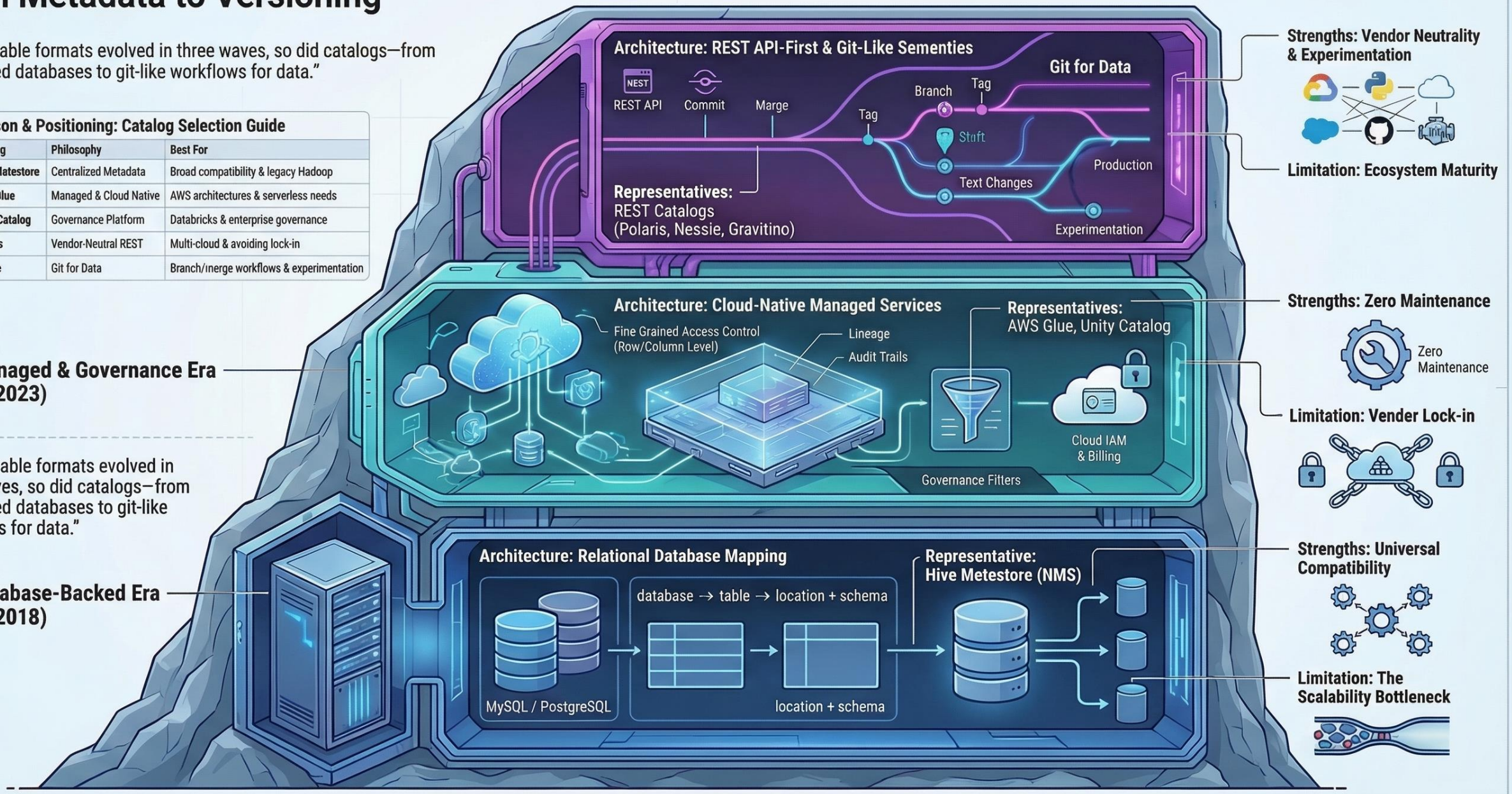
Comparison & Positioning: Catalog Selection Guide

Wave	Catalog	Philosophy	Best For
1	Hive Matestore	Centralized Metadata	Broad compatibility & legacy Hadoop
2	AWS Dlue	Managed & Cloud Native	AWS architectures & serverless needs
	Unity Catalog	Governance Platform	Databricks & enterprise governance
3	Polarls	Vendor-Neutral REST	Multi-cloud & avoiding lock-in
	Nessie	Git for Data	Branch/inerge workflows & experimentation

Wave 2: The Managed & Governance Era (2018–2023)

"Just as table formats evolved in three waves, so did catalogs—from centralized databases to git-like workflows for data."

Wave 1: The Database-Backed Era (2010–2018)



The Data Lakehouse Catalog Compatibility Matrix & Decision Framework (2025-2026)

OPTIMAL PAIRINGS & KEY TAKEAWAYS
(Best Match Configurations & Final Strategy)

KEY FINDING

KEY TAKEAWAY: CATALOG CHOICE = STRATEGY CHOICE
(Iceberg offers maximum catalog flexibility, while Dalta and Hudi perform best within their preferred native catalog ecosystems.)

BEST MATCH: ICEBERG + POLARIS/NESSIE
(Aligne with "Wave 3" neutrality and provides advanced versioning or multi-engine interoperability.)

BEST MATCH: DELTA LAKE + UNITY CATALOG
(Leverages Databricks native performance and specialized governance features like UniForm.)

THE EVOLUTION PATH — MIGRATION WAVES
(Visualizing Transition)

THE LOGIC LAYER — DECISION FRAMEWORK
(Guiding the Architect)

ORGANIZATIONAL & ARCHITECTURAL GOALS?

UNIVERSAL LEGACY SUPPORT	SERVERLESS AWS ARCHITECTURES	DATABRICKS ECOSYSTEMS (Elite Governance)	VENDOR NEUTRALITY (API-First)	GIT-LIKE DATA WORKFLOWS (Data-as-Code)	RADICAL SIMPLICITY (Developer-Centric)
CHOOSE HIVE METAXTORE (Best for migrating from Hadoop Infrastructure whore Sparll, Thut, and Trino need a well understood, mattoo option, howeves it faces bottleneces at patabyte cacle.)	**CHOOSE AWS GLUE** (Ideal for AWS centric stacks nealting MM integration and managed service simplicity, though it may lead to cloud luck-in and API rate lmiits.)	**CHOOSE UNITY CATALOG** Provides akte entergrise governance (ficeape, adoil, Dalta Sitaring) for Dalta Lake and Iceberg, provided Lob are emnfortable with the Databricks platform lock-in.)	**CHOOSE REST CATALOG (POLABIS)** (The choice for modem, API-first, narlo cloud manitectures prioritizing Iceberg, though it is a ynenger ecesystem with evewing governance features.)	**CHOOSE PROJECT NESAIE** (Enables "Data as Code" via breaching, tagging, and multi-table transactions; requires a tegm comfortable with Devsips/DROps cultures.)	**CHOOSE SQL DB (DUCKLAKE)** (Leveragas PostgraSQ), RBAC for smail to medium seaic developer-centric projects that prioritize velocity over multi-format compatibility.)

SECTION 1: THE FOUNDATION — COMPATIBILITY MATRIX
(Technical Gross-Reference)

Catalog	Iceberg Support	Delta Lake Support	Hudi Support	Paimon Support	DuckLake	Maturity
Hive Metastore	✓ Full	✓ Limited*	✓ Full	✓ Full	✗	Mature
AWS Glue	✓ Full	✓ Limited*	✓ Full	✓ Via HMS	✗	Mature
Unity Catalog	✓ Full	✓ Native	✓ Full	✓ Via Iceberg	✗	Mature
REST (Polaris)	✓ Native	⚠ Partial	⚠ Growing	⚠ Via Iceberg	✗	Emerging
Nessie	✓ Native	⚠ Partial	⚠ Limited	✗	✗	Emerging
SQL DB (DuckLake)	✗	✗	✗	✗	✓ Native	Emerging

*Delta/Glue Limitation. (Supporting Betai). Delta Lake support in Hive/Glue Hive/Dloe often requires specific storage handlers, which can hind cross-engine support compared to native integrations.)

PHYSICAL STORAGE
(Data Foundation)

KEY FINDING: Iceberg is the Interoperability Leader.
Apache Iceberg has the broadest catalog compatibility across the board, making it the safest choice for multi-vendor environments.

DuckLake: What if a SQL Database IS the Catalog?

Era 1: The Hive Metastore
Early metadata stored in relational databases struggled with file-listing bottlenecks at petabyte scales.

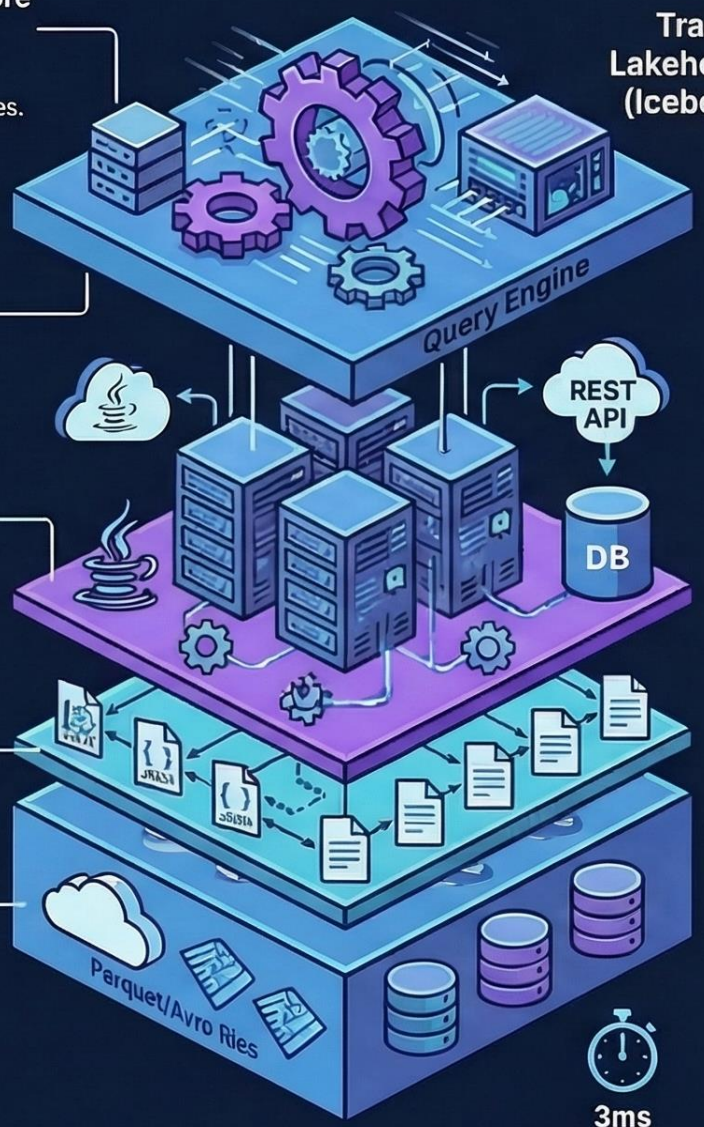
Era 2: File-Based Table Formats
Metadata moved to object storage (JSON/Avro) for infinite scale, but introduced "file-system dancing" and high latency.

Catalog Server (Java/REST/DB)
Hive Matastore Era: MySQL/Postgres for paths, file-listing oottienecks

Table Format (JSON Manifests)
JSON/Avro

Object Storage
Parquet/Avro files

Traditional Lakehouse Stack (Iceberg/Delta)

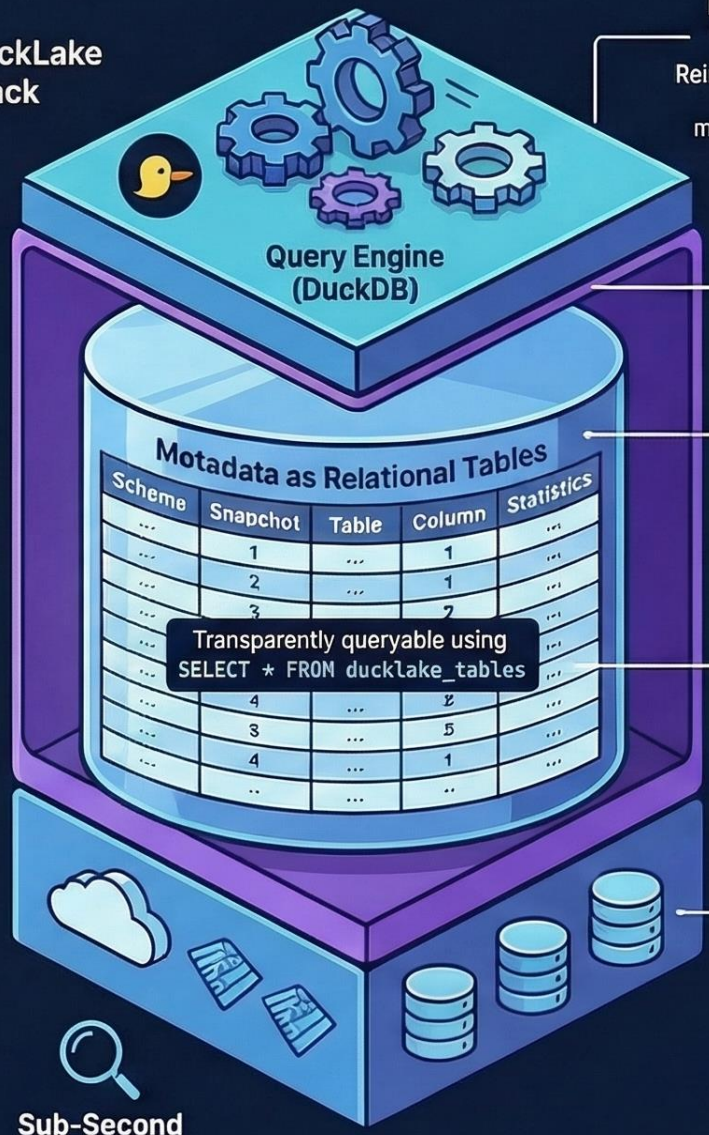


Collapses two layers into one.

The DuckLake Question

If Iceberg + REST catalogs are the "Linux of data lakes," DuckLake asks:
What if PostgreSQL IS the catalog?

The DuckLake Stack



Era 3: The DuckLake "Radical Simplicity"
Reintroducing the SQL database as the catalog, handling metadata as relational tables without the "file zoo."

SQL Database (Catalog + Metadata Merged)

Core Relational Components
Manages snapshots (state), schemas, tables (Inception-style), columns, and file-level statistics

Data Inlining
Small appends are buffered directly in the SQL database, then flushed to Parquet later to avoid small files problem

Object Storage (Parquet)

PERFORMANCE BENCHMARKS

3ms Metadata Inserts
Adding a new snapshot to a petabyte-scale dataset takes just 3 milliseconds in DuckLake.

4x Faster Query Planning
DuckLake outpaced Iceberg by 4x due to fewer network round trips and no Java/REST overhead.

Sub-Second Pruning
Selecting a specific day or month of data from a petabyte-scale lake remains sub-second via SQL indexing.

KEY TRADE-OFFS: SIMPLICITY VS. SCALE

DuckLake offers radical simplicity and multi-table ACID transactions, but introduces a database dependency and is currently DuckDB-centric.

DuckLake: From Browser to Production in 5 Steps

Step 0: Browser Preview (DuckDB Wasm)

Start with SQL directly in the browser window with no installation required, allowing for immediate data exploration.

Step 1: In-Process Analytics (Python + DuckDB)

Move to local development to query CSV and Parquet files with zero configuration using DuckDB as a universal data wrangling tool.

Step 2: The Lakehouse Layer (DuckLake)

DuckLake

Attach the DuckLake extension to add ACID transactions, snapshots, and time travel capabilities to your local files.

Step 3: Visual ETL (Apache Hop)

APACHE HOP

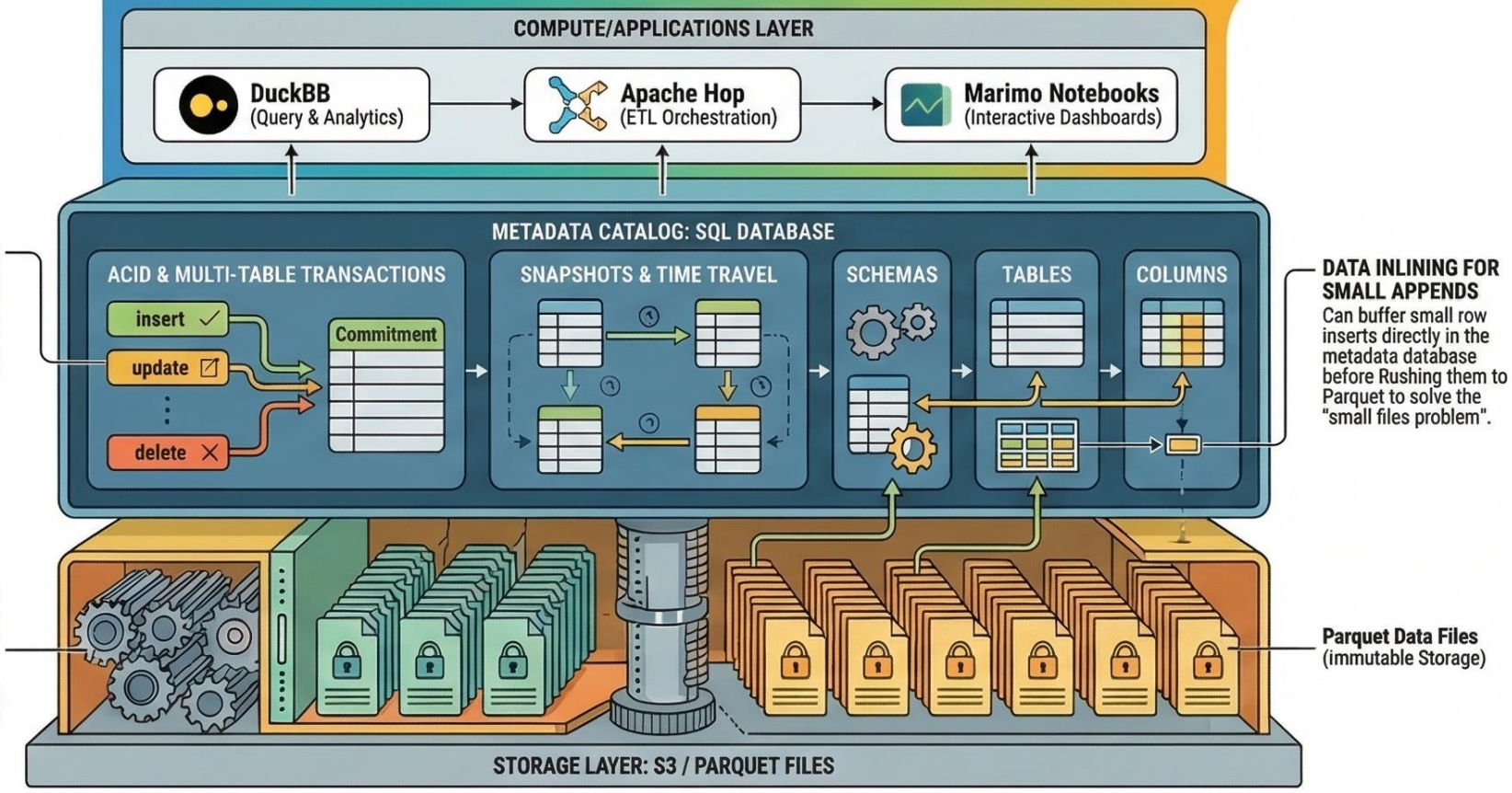
Implement drag-and-drop orchestration to build and manage data pipelines that feed directly into the DuckLake format.

Step 4: Reactive Analysis (Marimo Notebooks)

Marimo

Create interactive dashboards and reactive graphs to visualize and analyze lakehouse data in real time.

Complete Production Stack (Step 5)



Supports holistic transactions across 15+ different modifications (insert, update, delete, schema changes) in a single commit via the metadata store.

DATA INLINING FOR SMALL APPENDS
Can buffer small row inserts directly in the metadata database before flushing them to Parquet to solve the "small files problem".

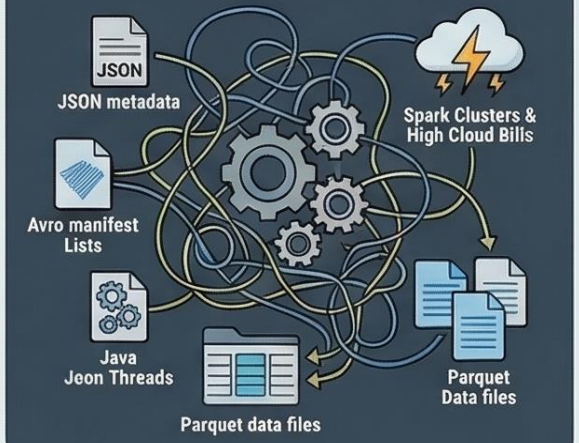
Built-in Encryption: Automatically generates separate encryption keys for every Parquet file, allowing secret tables to be stored safely on untrusted storage.

Parquet Data Files (immutable Storage)

Architectural Contrast: Complexity vs. Simplicity

The Traditional "File Zoo" Stack

Relies on a 528 complexity of Java threads, REST catalogs, JSON metadata, Avro manifest lists, and Parquet data files, often requiring Spark clusters and high cloud bills.

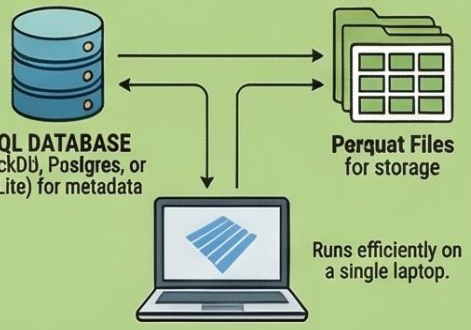


KEY FINDING: 4x Performance Advantage

Benchmarks show DuckLake is up to four times faster for updates than leeberg because it eliminates multiple non pandlalicable network round trips to metadata files.

The DuckLake Lean Architecture

Replaces the "xco" with a simple SQL database for metadata and Parquet for storage, running efficiently on a single laptop.



From Browser to Lakehouse: The DuckLake Architecture

Step 0: The 30-Second Browser Preview

Step 1: The DuckLake "Database + Files" Architecture

Zero-Installation SQL

Open **shell.duckdb.org** in any browser to run instant SQL without any setup.

Direct Data Lake Access

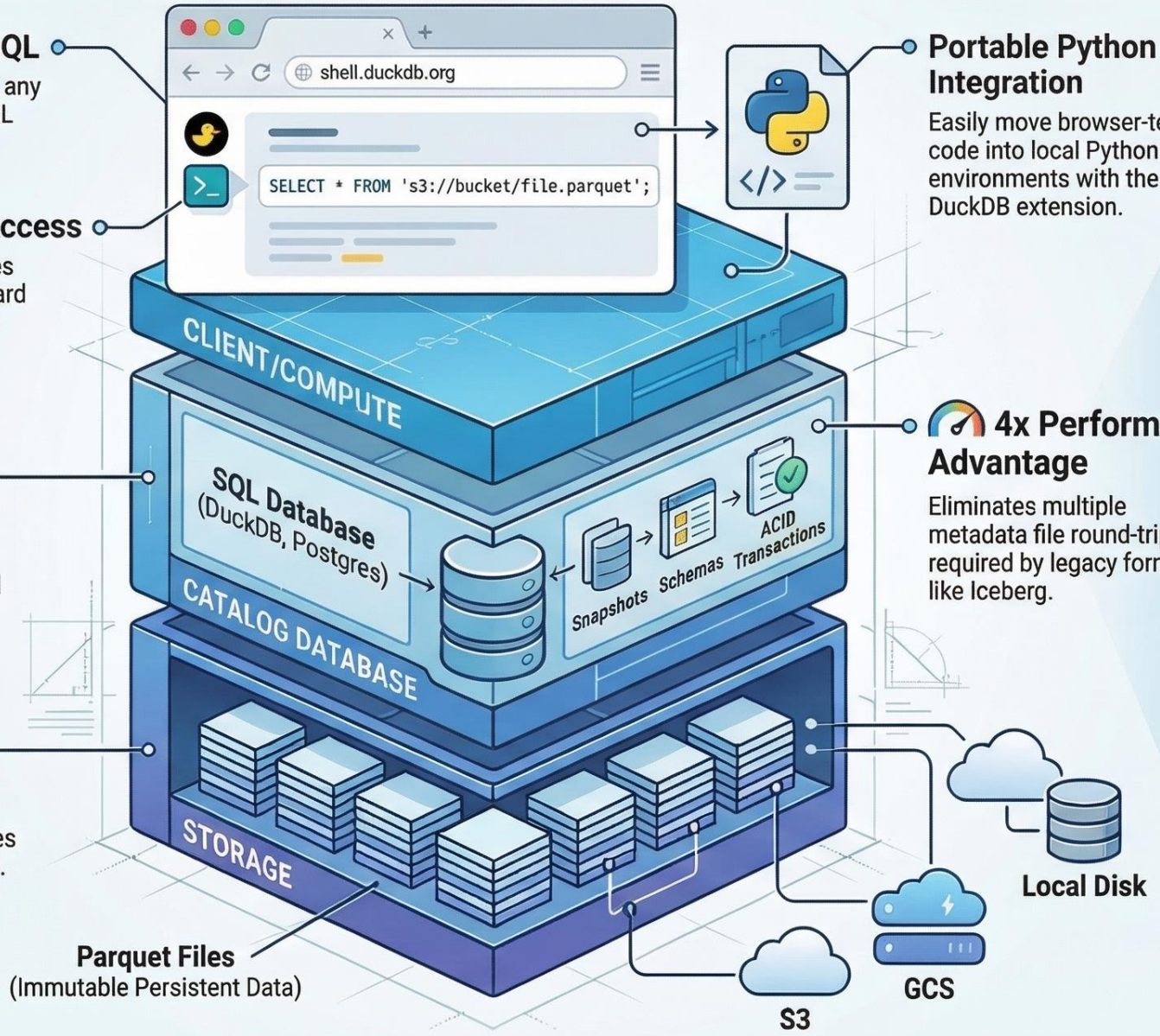
Query remote Parquet files immediately using standard SELECT statements from the web.

The Metadata Catalog Layer

Uses a standard SQL database to manage snapshots, schemas, and ACID transactions.

The Scalable Storage Layer

Persistent data is stored as immutable Parquet files on S3, GCS, or local disks.



Portable Python Integration

Easily move browser-tested code into local Python environments with the DuckDB extension.

4x Performance Advantage

Eliminates multiple metadata file round-trips required by legacy formats like Iceberg.

ARCHITECTURAL COMPLEXITY COMPARISON

LEGACY TABLE FORMATS (e.g., Iceberg)	DUCKLAKE
	
Metadata Storage: Nested JSON/Avro File "Zoo"	**Metadata Storage:** Single SQL Database
	
Tech Stack: Java, Spark, REST Catalog	**Tech Stack:** DuckDB + SQL
	
Update Speed: High Latency (Multiple Writes)	**Update Speed:** 3-5ms (Sub-second planning)

DuckLake

Step 2: Add DuckLake – The Lakehouse Layer

Terminal & Code

```python
import duckdb
con = duckdb.connect()

# Install and load DuckLake extension
con.sql("INSTALL ducklake")
con.sql("LOAD ducklake")

# Attach a DuckLake catalog
con.sql("""
    ATTACH 'ducklak:metadata.ducklake' AS my_lake
    (DATA_PATH 'data_files')
""")

# Switch to the lakehouse
con.sql("USE my_lake")

# Creating and Populating Tables
# Create a lakehouse-managed table
con.sql("CREATE TABLE sales (id INT, amount DECIMAL, product VARCHAR)")

# Insert data with ACID guarantees
con.sql("INSERT INTO sales VALUES (1, 100.50, 'Widget')")
con.sql("INSERT INTO sales VALUES (2, 250.75, 'Gadget')")

# Advanced Operations (Time Travel)
# Query the lakehouse table
con.sql("SELECT * FROM sales").show()

# Time travel: query previous snapshot
con.sql("SELECT * FROM sales AS OF SYSTEM TIME '-5 minutes'").show()
```

Install and Initialize
With just three lines of SQL, DuckLake is installed, loaded, and atteched as a catalog to manage your data files.

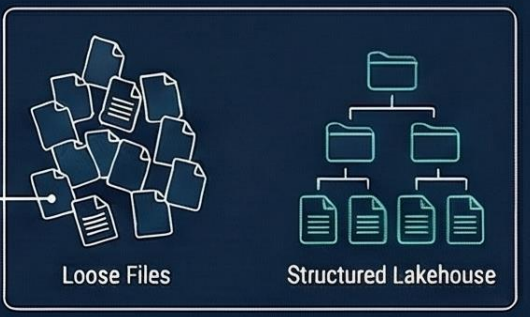
Instant Time Travel
Because DuckLake tracks every snapshot in its metadata catalog, users can query the database as of a specific system time or version.

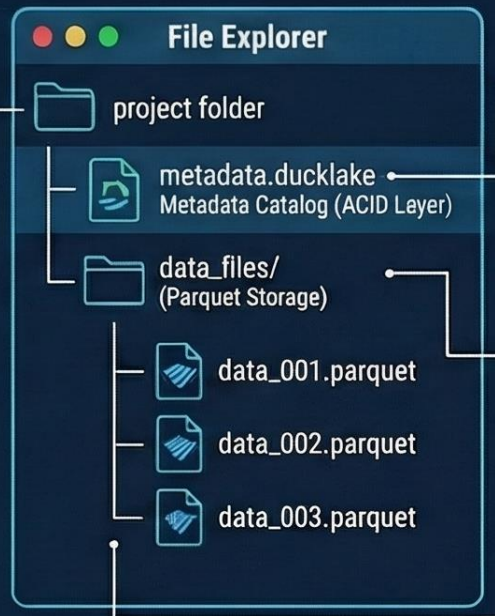
THE COMMAND CENTER
(INTERFACE)

THE LAKEHOUSE ENGINE
(METADATA/CATALOG)

THE DEEP STORAGE
(FOUNDATION)

Before/After Comparison

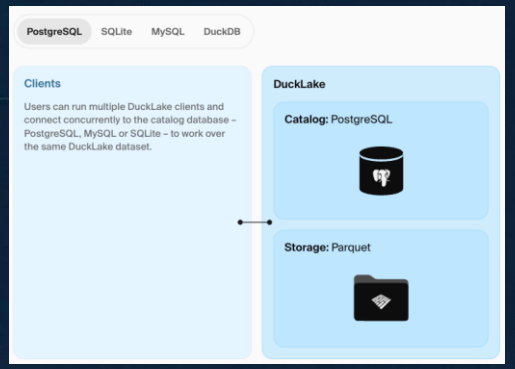




Catalog-Driven Management:
The metadata layer stores snapshots, schemas, and statistics, acting as the "brain" that coordinates how compute nodes interact with storage.

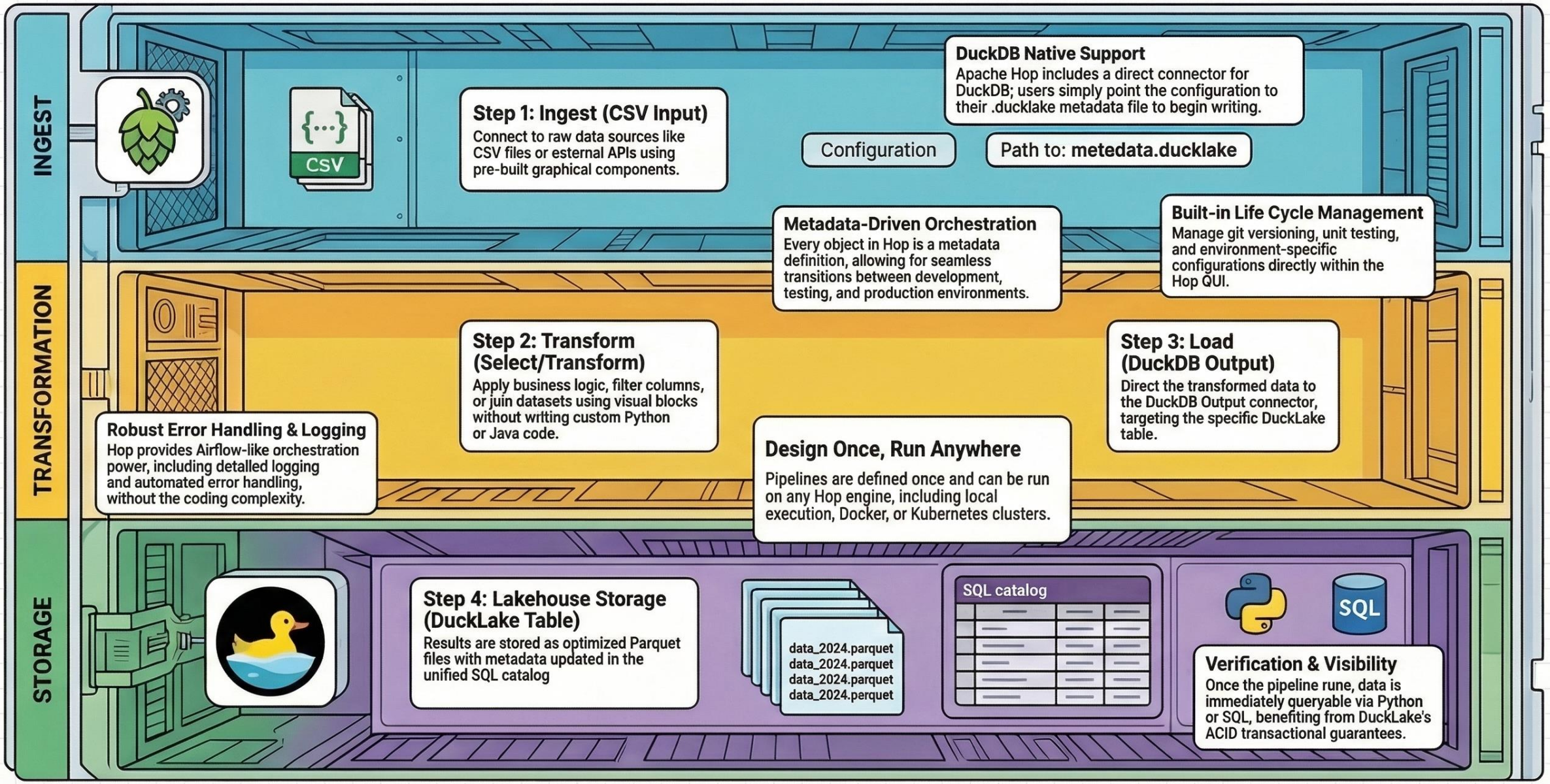
Parquet-Based Persistence:
Raw data is stored as immutable Parquet files in the "data_files" directory, ensuring high-performance compression and portability.

ACID-Guaranteed Transactions:
DuckLake uses a SQL metadata database to ensure that every insert or update is atomic and consistent, preventing data corruption during concurrent access.



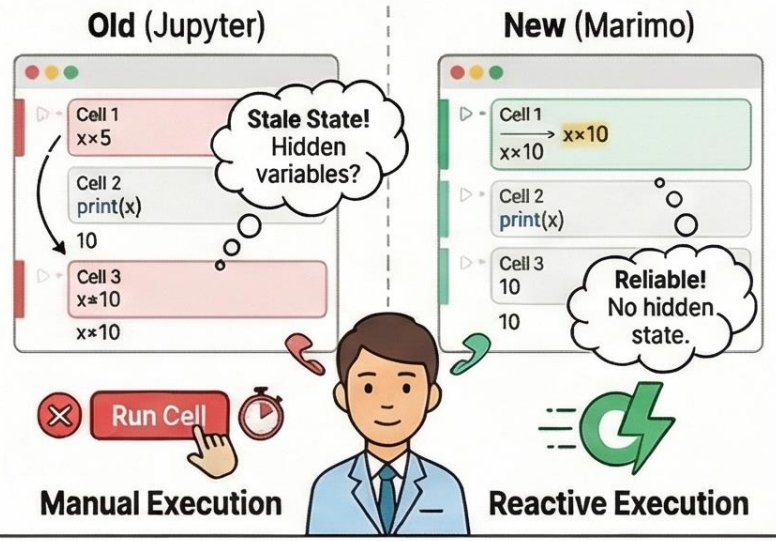
Orchestrate Pipelines: Drag-Drop ETL to Your Lakehouse

Apache Hop & DuckLake: Build Repeatable Pipelines Without Python ETL Code



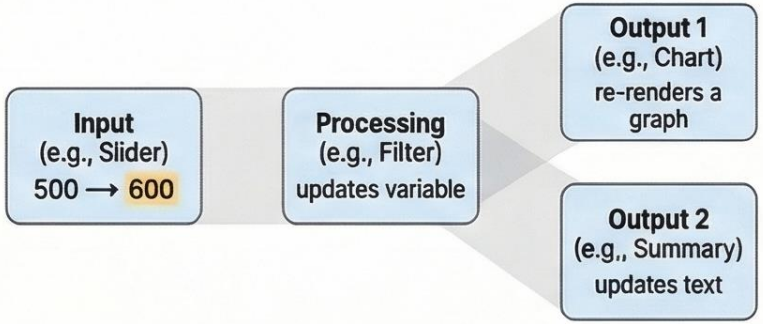
Step 4: Marimo Notebooks – Reactive Analysis

1. The Problem with Traditional Notebooks



2. The Marimo Reactive Solution

Dataflow Graph Enforcement: Change an input, all dependent cells re-run automatically based on dependency.



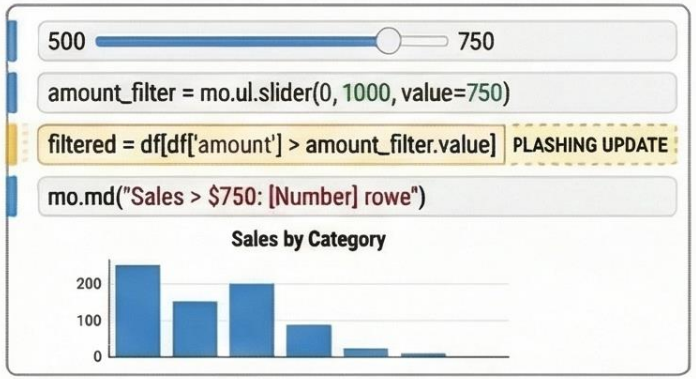
No More Hidden State: Deleting a cell removes its variables, keeping the environment clean and reproducible.

3. Interactive UI Components

```
import marimo as mo
import duckdb

con = duckdb.connect('metadata.ducklake')
df = con.sql("SELECT * FROM sales").df()

amount_filter = mo.ul.slider(0, 1000, value=500)
filtered = df[df['amount'] > amount_filter.value]

mo.md(f"Sales > $(amount_filter.value):
(hm(filtered)) rows")
```

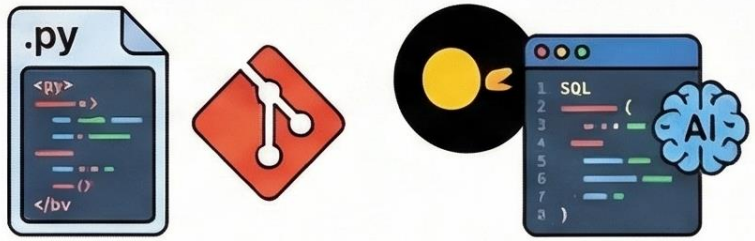
Notebooks as Functional Apps: Use `mo.ui` (silders, dropdowns, buttons) for reactive inputs that trigger downstream logic.

4. Real-Time Data Analysis



Reactive Data Filtering: Variables update instantly when UI inputs change.
Instant Visual Feedback: Charts and summaries re-render without clicking 'Run'.

5. Built for Developers



Pure Python & Git-Friendly

Marimo notebooks are standard .py files for easy version control, reviews, and testing (e.g., PyTest).

SQL & AI Integration

Built-in support for DuckDB, SQL cells, and AI-native features like autocomplete and inline edits.

6. Why Reactive Analysis Matters

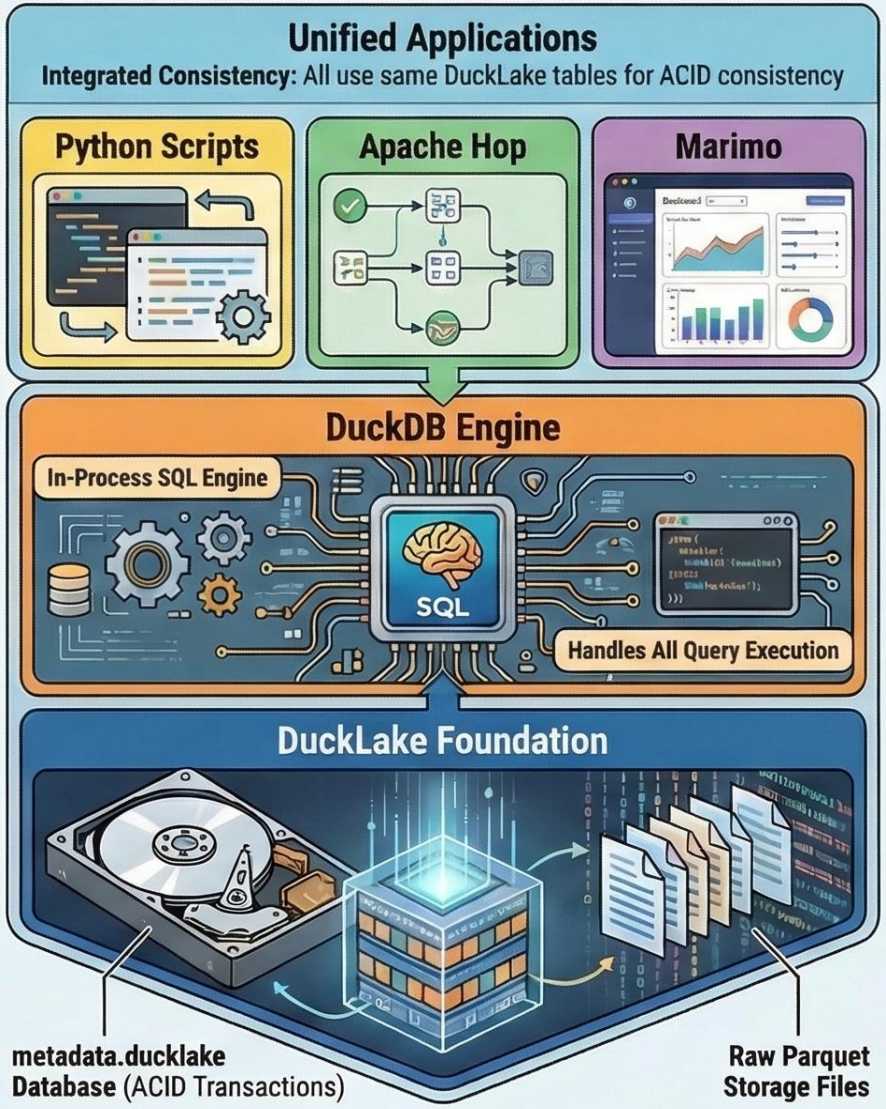
> **Fixing the Broken Default:** Moves analysis away from fragile, stateful messes toward reliable, shareable apps.

Deploy Anywhere: Export as WebAssembly HTML or serve as a web app using Marimo CLI, working the same locally and deployed.

Production-Ready Lakehouse on a Laptop: The $0/Month Modern Data Stack

Visual Demo: The Integrated Workflow

Top Left: Terminal (Python/DuckDB)
Top Right: Apache Hop GUI
Bottom Left: File Explorer
Visualizing the physical storage layer
Bottom Right: Marimo Notebook
Visualizing the consumption layer



The Revelation: Cost vs. Complexity

Component	Traditional Enterprise Stack	Local "Laptop" Stack
Messaging/Ingestion	Kafka Cluster (3+ Nodes)	Apache Hop (Java App)
Compute Engine	Spark Cluster (Driver + Executors)	DuckDB (Single Process)
Orchestration	Airflow (Scheduler + Workers + DB)	Apache Hop
Workspace/Storage	Databricks ($$$$) + Cloud Storage	DuckLake (Local Files)
Dashboarding	Tableau/PowerBI (Licenses)	Marimo (Open Source Python)
Total Cost	**$5,000–$30,000/Month + DevOps**	**$0/Month + Runs on Laptop**

The Scaling Path

Step 1: Cloud Storage
Point DuckLake to S3/GCS bucket (minimal connection change)

Step 2: Server Orchestration
Move Apache Hop pipelines to scheduled server or container

Step 3: Web Deployment
Deploy Marimo dashboard as standalons web application

Step 4: Serverless Growth
Integrate MotherDuck for serverless capabilities as data grows

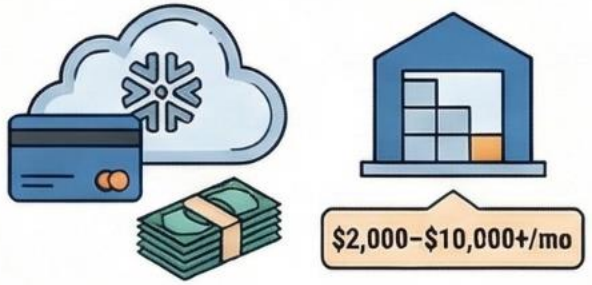
"Don't Start with Databricks and Work Backwards."

Start Simple, Scale Incrementally. The future of data engineering is eliminating complexity. Save expensive infrastructure for proven needs.

From Bill Shock to Blazing Fast: Definite's Migration to the Duck Stack

PART 1: STEP-BY-STEP: THE PATH TO THE DUCK STACK

Step 1: Identify the "Bill Shock"

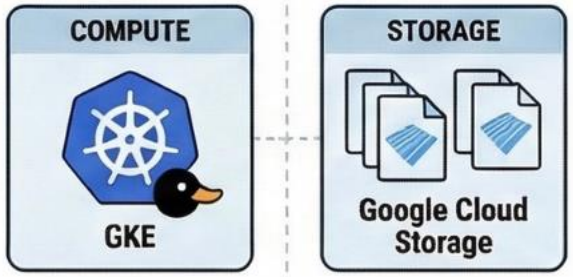


Traditional cloud warehouses can cost mid-sized companies thousands monthly due to complex credit models and storage markups.

Step 2: Choose the Pragmatic Engine

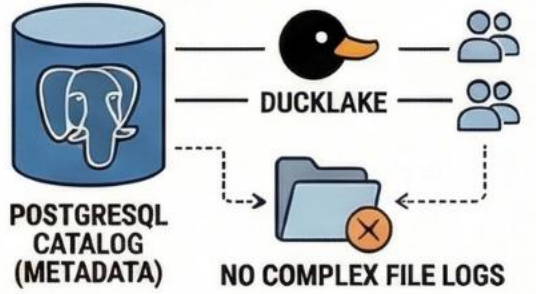
Definite bypassed Postgres and ClickHouse in favor of DuckDB for its embeddable simplicity and Postgres-compatible SQL.

Step 3: Decouple Compute and Storage



DuckDB runs on Kubernetes (GKE) while data is stored as Parquet files on Google Cloud Storage for elastic, low-cost scaling.

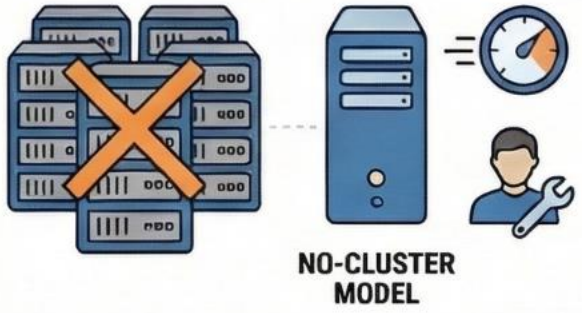
Step 4: Implement DuckLake for ACID



To handle concurrent writers and metadata, DuckLake stores table metadata in a standard PostgreSQL database.

PART 2: THE PERFORMANCE & VELOCITY PAYOFF

Step 5: Eliminate Cluster Overhead



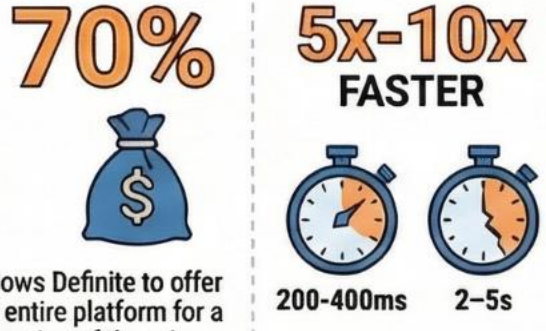
Moving to a "no-cluster" model removes cold-start latencies, rebalancing issues, and need for dedicated infrastructure teams.

The Economic Revolution

MONTHLY COST COMPARISON		
	SNOWFLAKE + LOOKER + FIVETRAN	DEFINITE (OUCKDD/DUCKLAKE)
Compute:	$2k - $5k	Included
Storage (1TB):	$40 - $80	$20 - $40
BI & Connectors:	$1.5k - $5k	Included
Total Monthly:	$3,500 - $10,000+	**$250 - $500**

Total monthly costs for a mid-market stack plummeted from nearly $10,000 to under $500.

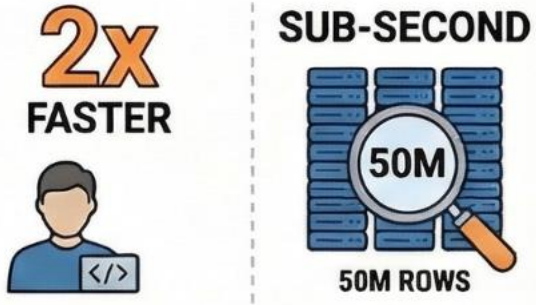
70% Cost Reduction & 5x-10x Faster Queries



Allows Definite to offer its entire platform for a fraction of the price.

Typical queries dropped from 2–5 seconds to 200–400ms by eliminating network hops.

2x Engineering Velocity & Sub-Second Ad-hoc Exploration



Developers run the full stack locally, shortening the development lifecycle.

Filtering and grouping 50M rows happens in under 1 second for instant feedback

Beyond Hive: The Anatomy of the Modern Lakehouse

THE THREE WAVES OF EVOLUTION

Wave 1: 2016-2018

Wave 1: The Incremental Pioneers

Led by Hudi & Delta Lake. Introduced ACID transactions and CDC for data lakes, proving cloud storage could be mutable.

Hudi

Delta Lake

Wave 2: 2018-2022

Wave 2: The Universal Standard

Apache Iceberg prioritized engine neutrality and comprehensive scheme evolution, making interoperability the new priority.

Wave 3: 2022-Present

Wave 3: Streaming & Simplicity

Newcomers like Apache Paimon (streaming first) and DuckLake (SQL-native) rethink tundamentals to prioritize sub-minute freshness and developer velocity.

Paimon

DuckLake

FORMAT FACE-OFF & DECISION MATRIX

Apache Iceberg — For Neutrality
Best for multi-engine platforms and long-term vendor neutrality.

Delta Lake — For Spark Ecosystems
Optimized for Databricks and Spark-native environments.

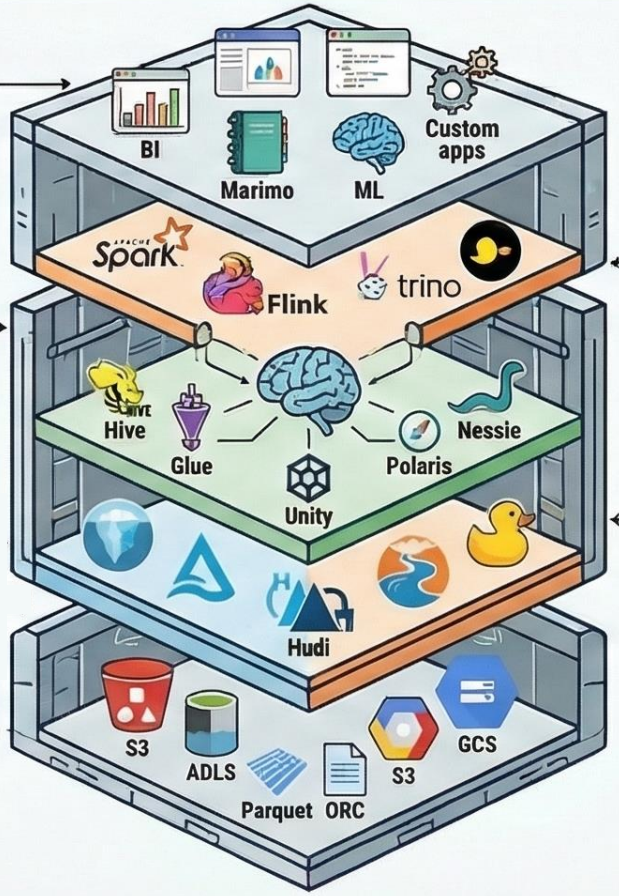
Apache Hudi — For CDC Specialization
The preferred choice for Change Data Capture-heavy architectures on AWS

Apache Paimon — For Streaming Freshness
Designed for Flink-heavy workloads requiring sub-minute data freshness

DuckLake — For Radical Simplicity
Chosen for high developer velocity where a SQL database serves as the catalog

LAYER 5: User Interface & Tools
The top-level access points including BI tools, Marimo notebooks, ML applications, and custom apps.

LAYER 3: The Catalog
The 'brain' of the stack (Hive, Glue, Unity, Polaris, Nessie); it manages governance and location discovery.



LAYER 4: Compute Engines
The processing power (Spark, Flink, Trino, DuckDB) that interacts with the table formats to query data.

LAYER 2: Open Table Format
The metadata layer (Iceberg, Delta, Hudi, Paimon, DuckLake) that defines how files are organized and managed.

LAYER 1: Persistent Object Storage
The foundation consisting of S3, ADLS, or GCS, storing data in open files like Parquet or ORC.

THE HIDDEN COSTS REALITY CHECK

The 4-Step GDPR Deletion Workflow:

• Logical Delete → Compaction → Expiration → Garbage Collection (Automation mandatory).

Write Strategy Trade-offs:
Copy-on-Write (CoW) provides fast reads but expensive writes, Merge-on-Read (MoR) allows fast writes but complex, slower reads.

Operational Overheads:
Costs scale with metadata complexity (S3 API bills) and team expertise (e.g., Flink for Paimon).

THE PRACTICAL PATH FORWARD

The 'Start Simple' Progression:
Browser (DuckDB Wasm) → Python + DuckDB → Add DuckLake → Apache Hop pipelines → Marimo notebooks

> **Start simple. Prove value. Scale when needed.**
> (You don't need expensive proprietary platforms to build a functional lakehouse).

Questions?