Achieving Data Warehouse Performance on Apache Iceberg

Sida Shen
Product Manager

Agenda

Iceberg Promise, Speed Reality

Cost of slow queries on your Apache Iceberg Tables

Challenges With Query Performance

A quick tour: metadata overhead, data fetch, execution challenges.

Fix the Table First

Hidden partitions, right-sized files, V3 features—keep Iceberg healthy.

Then Fix the Engine

Query engine optimizations: distributed metadata scan, tiered cache, vectorized MPP compute.

Real Success Stories

Apache Iceberg Replacing Data Warehouse In Production

The Open Table Format For Analytic Datasets

Iceberg is a step-up from Hive for analytics workloads

- ACID control
- Schema & partition evolution, hidden partitions
- Time travel
- Better data freshness
- SQL

- Enable data warehouse workloads on open & standardized formats
- Unify all workloads on a single source of truth data
- Easy data governance, simple architecture, cost-effectiveness

Once data is in Iceberg, it never leaves, or **does it?**

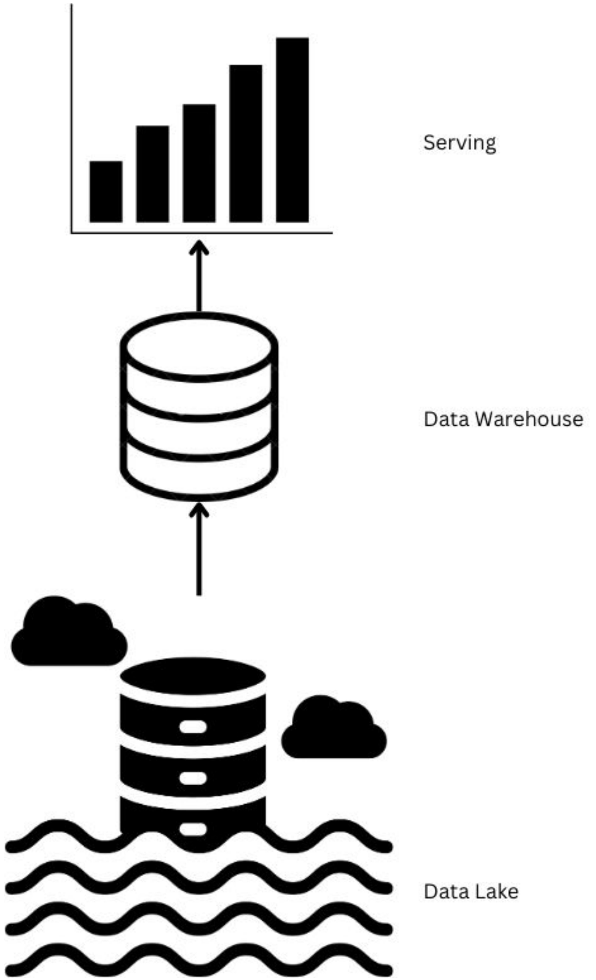
The Reality – Queries Are Not Fast Enough

Users are still copying their data out of Apache Iceberg for query acceleration

- Existing data lake(house) query engines are not built for data warehouse workloads
  - Not able to handle demanding workloads such as customer-facing analytics
- Poorly maintained Iceberg tables
- User overengineer or overspend on existing query engine to barely get passable performance, which is not sustainable nor future proof
- As workaround, users are forced to move workloads to a high performance data warehouse purely for query acceleration

The Cost Of Slow Data Lake Queries

The Challenge that comes with moving your workload AND data

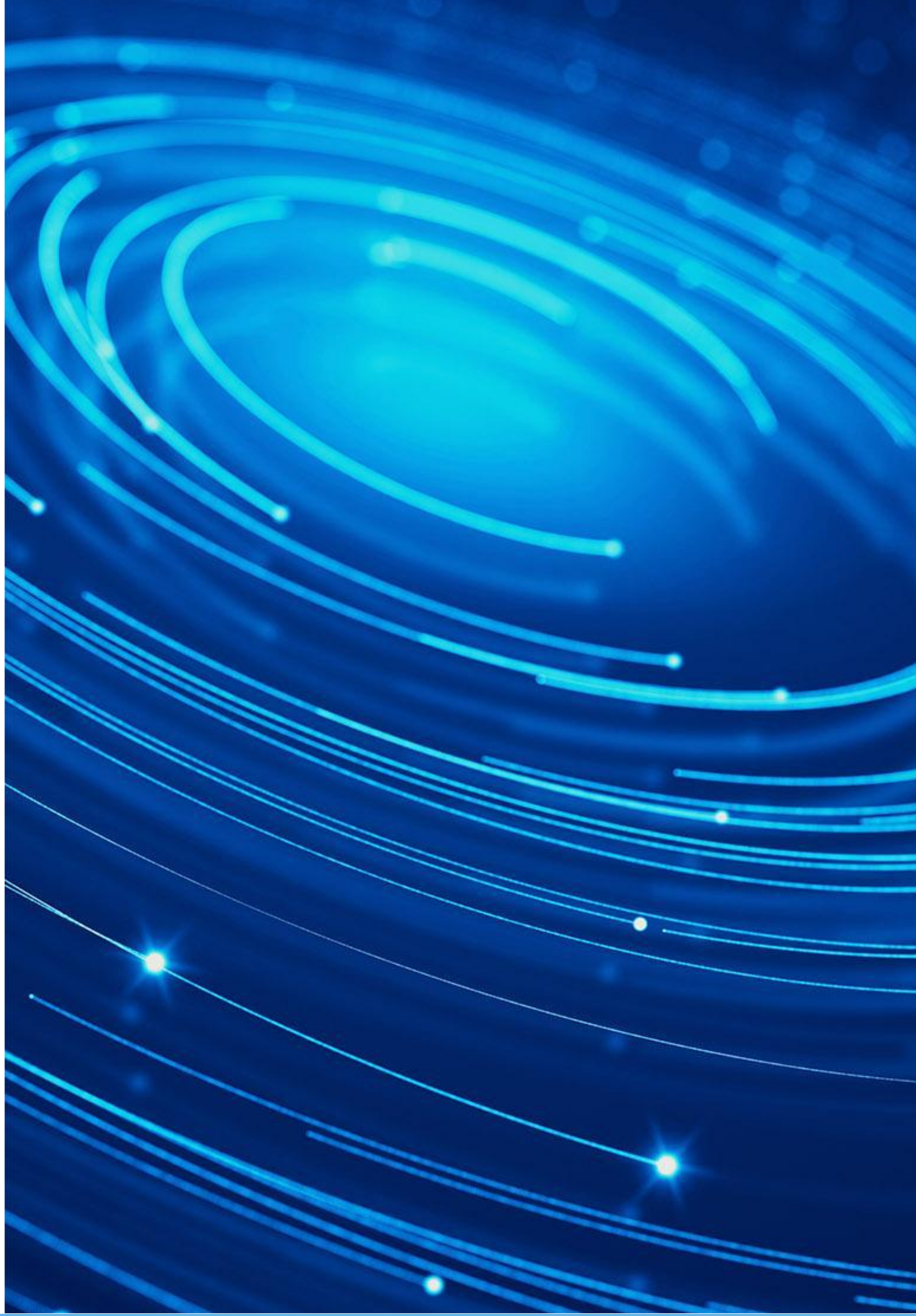


The Challenges

- Cost of maintaining an infra-level software
- The cost of the data ingestion pipeline
- Challenges from matching schema, data type, SQL, etc.
- Data governance challenges from duplicating the data

Challenges With Iceberg Queries

Engine-side Bottleneck -> Data + Execution



CelerData

Data Access

Slow data fetch

- Each Parquet/ORC object lives in S3 (~100 ms)
- Millions of < 1 MB data & delete files → random I/O storm
- Lack of caching

Delete files re-read & re-parsed

- Position / equality deletes shipped as standalone Parquet
- Every executor re-opens the same files, deserializes rows, builds bitmaps

Query Execution

CPU under-utilised, long-tail latency

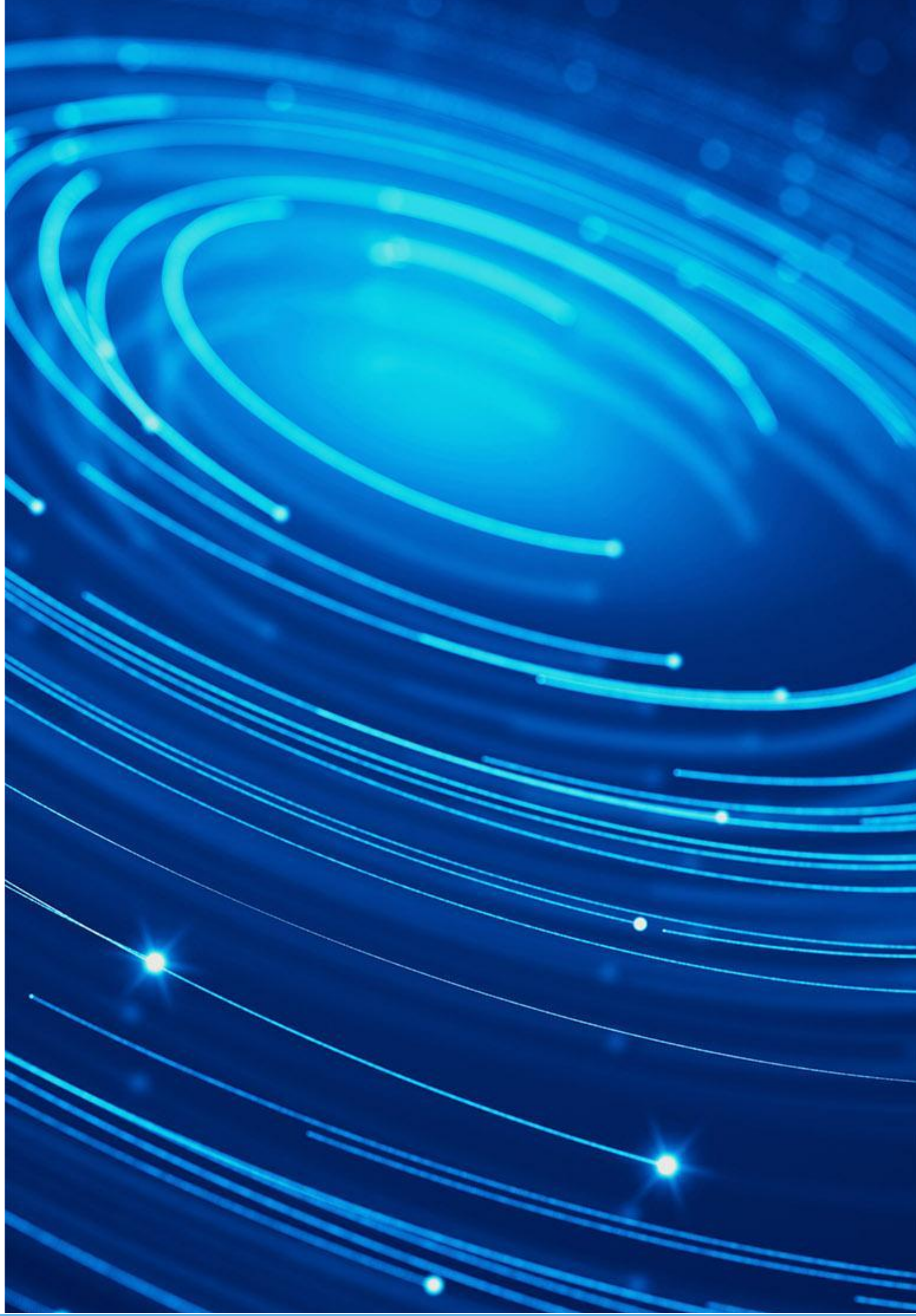
- Row-iterator scan path, little or no vectorisation / SIMD
- JVM shuffle and build hash tables → high GC overhead

Limited parallelism

- Single coordinator must gather full file list before workers start
- Pipelines stall when any stage blocks on I/O

Challenges With Iceberg Queries

Metadata/Planning



CelerData

Iceberg Metadata Overhead at Scale: A Real Case

Table Characteristics

- ~300 columns
- 500K new data files/day
- 30 manifest files/day (~8MB each)

Querying Just 1 Month of Data

- ~1 minute spent parsing metadata
- High CPU & GC pressure during planning phase

Scalability Hits a Wall

- Even with ~100 execution nodes,
- Performance is bottlenecked by metadata parsing
- Execution doesn't start until planning completes

Iceberg Metadata Overhead at Scale: A Real Case

Query Timeline Breakdown

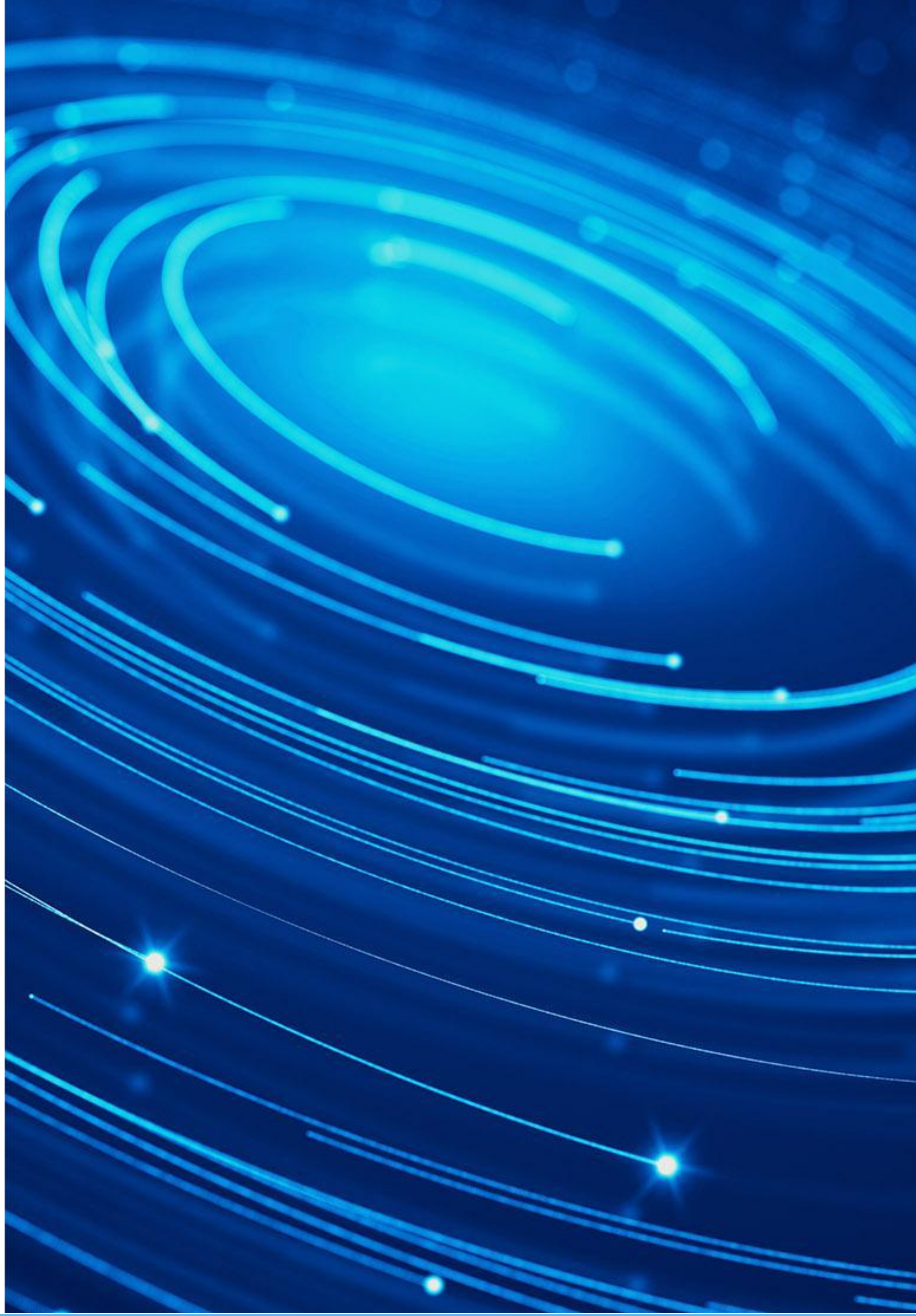
- Over 80% of total query time is spent just parsing metadata

Planning > Execution

- Query execution is fast
- But planning is blocked by manifest
- Even small number of manifest files can cause huge overhead

How To Accelerate Iceberg Queries

Tips, tricks, and best practices



CelerData

Keep Your Iceberg Tables Healthy

Smart partition layout

- Use hidden transforms (truncate(date), bucket(user_id))
- Watch for skew & NULL “hot” partitions

Right-sized data files

- Aim for 128 – 512 MB (object-store sweet spot)
- Run minor compaction to merge small delete files

Keep Your Iceberg Tables Healthy

Lean metadata

- Periodically rewrite manifests (merge tiny ones)
- Expire snapshots: keep N days or N versions only

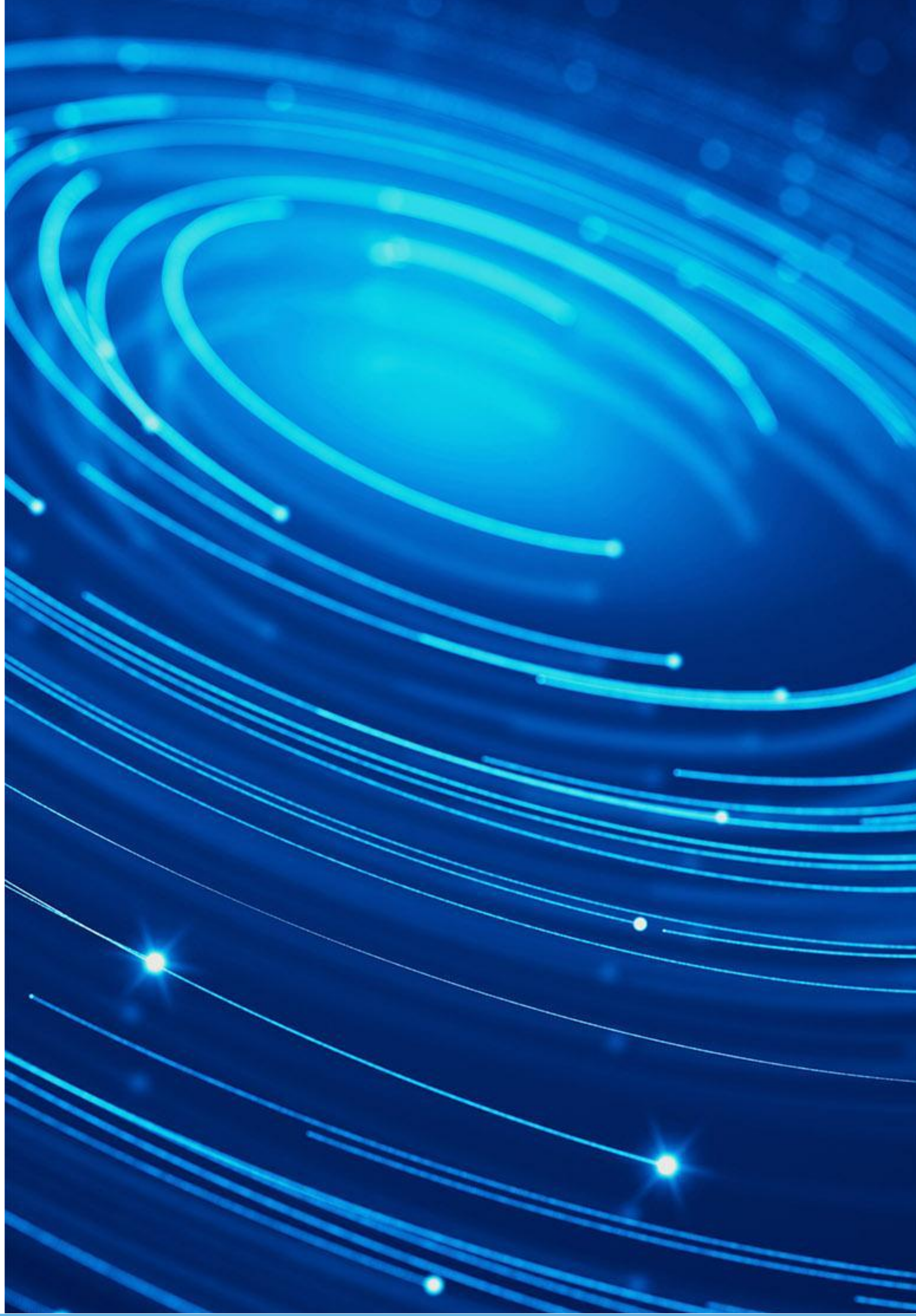
CHOOSE THE RIGHT ENGINE!!

Modern deletes (Spec V3)

- Switch position + equality files → Puffin Deletion Vectors
- One compressed bitmap per data-file, many per Puffin file

Entering StarRocks

Query engine built for data warehouse workloads

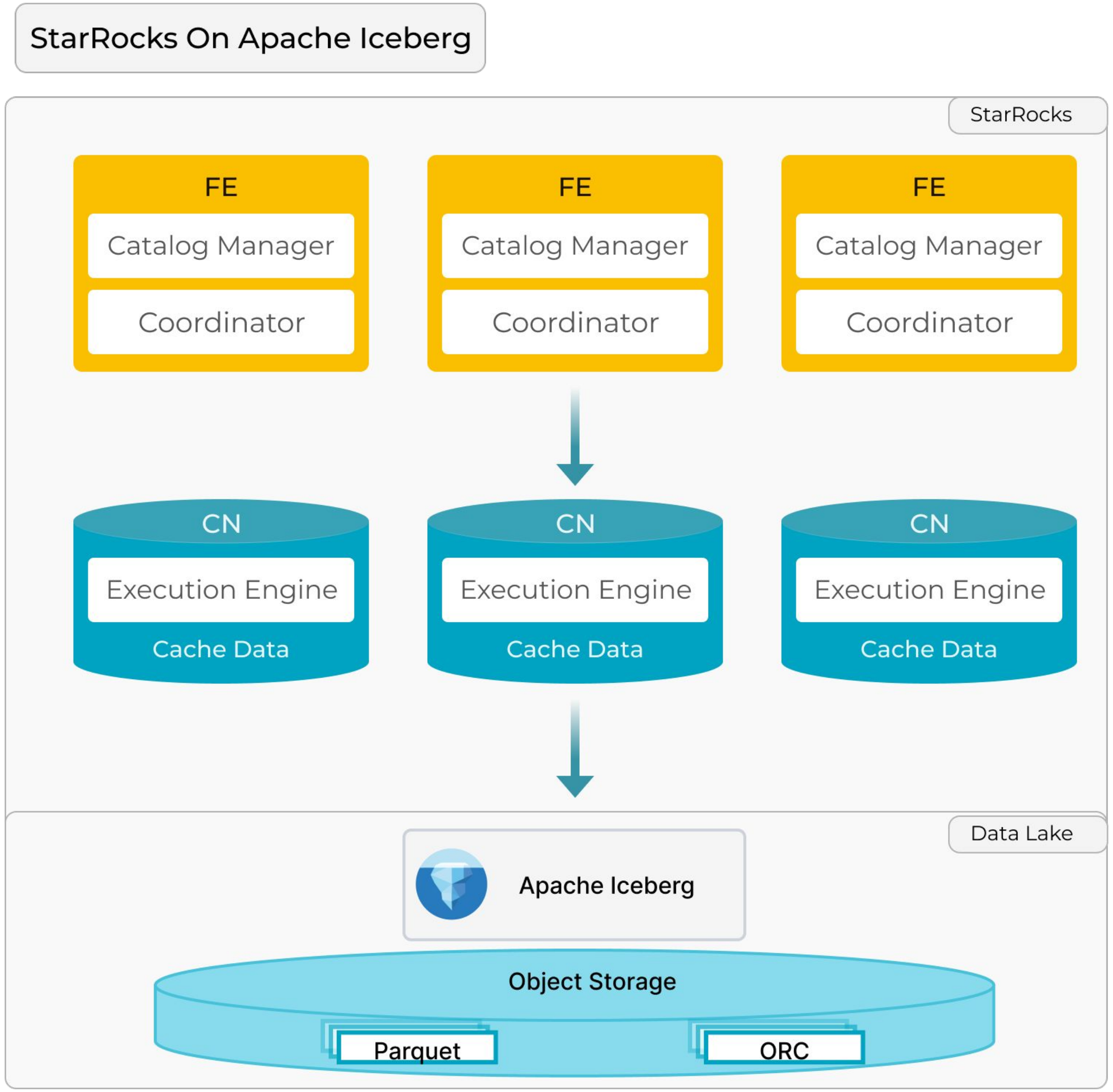


StarRocks + Apache Iceberg

Lakehouse for high-concurrency low-latency queries

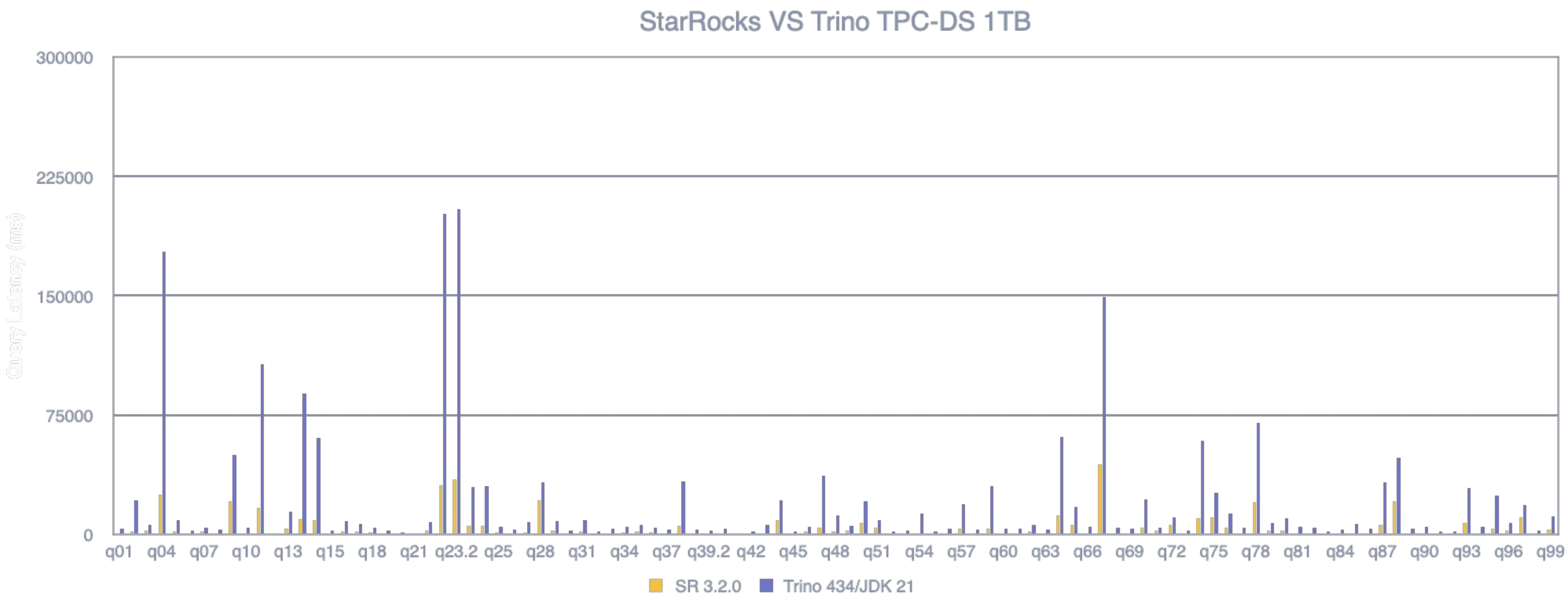
StarRocks

- Linux foundation open-source project
- Purposely built for data warehouse workloads
- Natively integrated with Apache Iceberg and open & standard file format
- Simple architecture with no external dependencies
- No denormalization: JOINs on the fly



Comparing StarRocks To Trino On Apache Iceberg

How fast is a purposely built query engine for data warehouse



StarRocks (3.2) is 4.62 times faster than Trino (434 on JDK 21) on TPC-DS 1TB

Distributed Manifest Parsing + Tiered Metadata Cache

Distributed Manifest Scan

- FE (planner) sends scan fragments that read AVRO manifest files in parallel on every CN nodes.
- Output is a global “file table” that is union-all-ed back to FE.

Metadata Cache

- In-memory + on-disk cache for metadata.json, manifest-list, manifest files.
- LRU admission via Caffeine

SIMD / Vectorized Scan

SIMD Vectorized Parquet Reader

- Column projection + predicate push-down select only needed pages.
- Decompress & decode via AVX2 / AVX-512.
- Batches 4096 rows per vector.

Scan Operator

- Equality deletes loaded once
- One left anti-join executed distributedly across CNs

Vectorized MPP Pipeline + Adaptive Execution

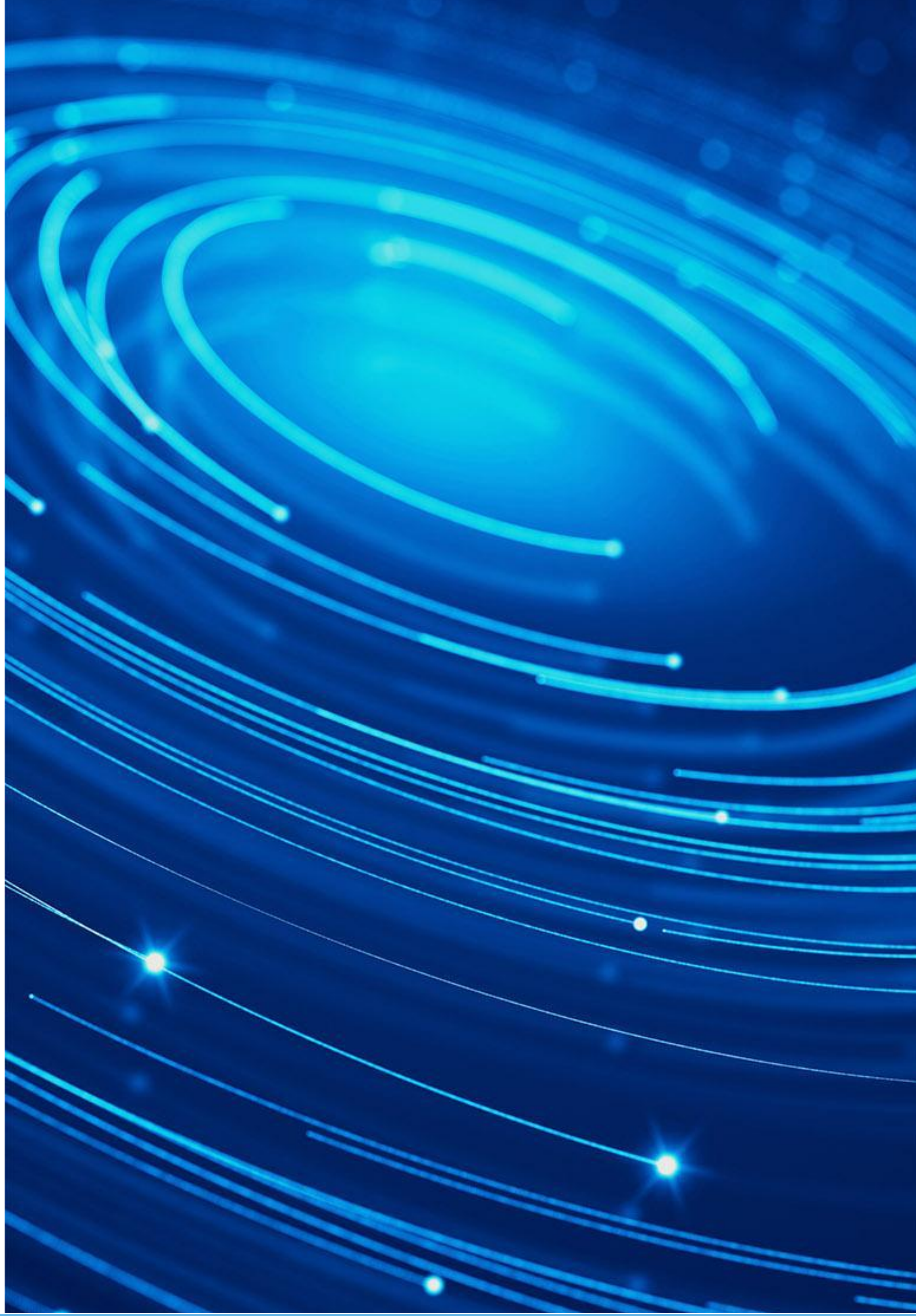
Vectorized Pipeline Execution Engine

- All operators (scan → join → agg → sort) run in C++, process fixed-size column-wise batches, push-based scheduler overlaps stages.
- CPU cores ≥95 % busy, GC ≈ 0 ms, SIMD speeds up operators 3-5x
- Asynchronous plan/execution: Incremental file dispatch

Cost-Based Optimizer & Runtime Filters

- Bloom/IN filters created mid-query and broadcast; join type switches to hash-join or broadcast based on row stats.
- Prune rows before heavy joins.

Iceberg + StarRocks In Production



CelerData

Wechat - Before

Social media platform with 1.3 billion MAU

Architecture

- Hadoop-based data lake
- Various data warehouses for query acceleration

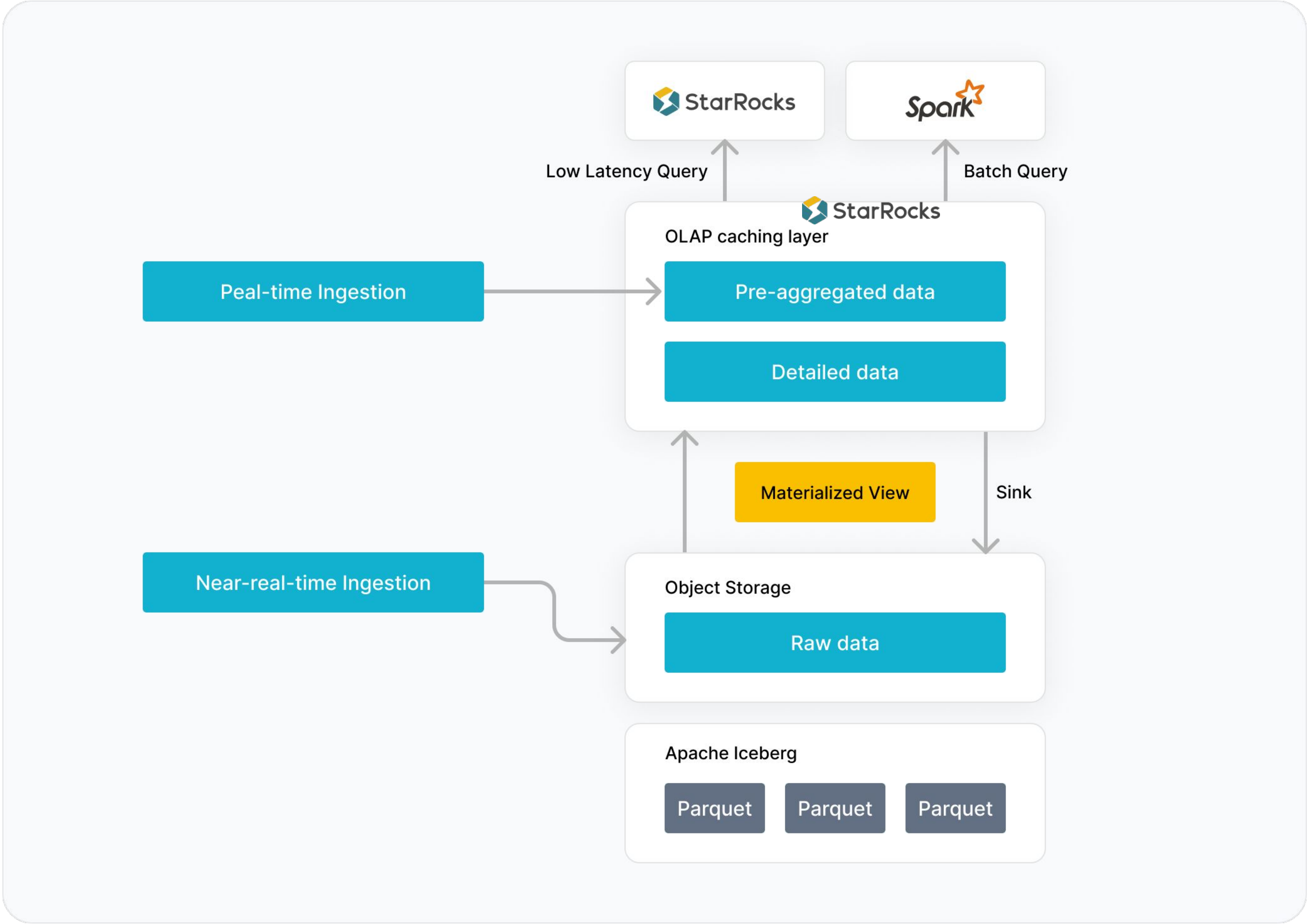
Requirements

- Handle trillions of records per day
- P90 latency < 5s

Challenges

- Juggling multiple systems from separated real-time and batch analytics pipelines
- Maintaining data ingestion pipelines for data warehouses
- Governance challenges from managing multiple copies of the same data
- Challenges in standardizing data analysis processes
- Data Freshness is not good

Wechat - After



New architecture

- Real-time data into StarRocks and periodically sink to Apache Iceberg
- Batch data ingest into Apache Iceberg directly

Result

- Handling Trillions of daily records
- Data freshness from minutes – hrs to seconds– minutes
- 65% storage cost reduction
- Shortened development cycle for offline tasks by 2 hours

More StarRocks + Apache Iceberg Success Stories

From BigQuery to Lakehouse: How We Built a Petabyte-Scale Data Analytics Platform (Iceberg Summit 2025)

- TRM Labs Customer-facing analytics, 100+ TB growing 25-45% annually, complex JOIN and high-cardinality AGG heavy with 3 second P95 SLA
- Considered Trino, **StarRocks**, and DuckDB; 50% improvement in P95, 54% reduction in query timeout errors

RedNote: Leveraging Iceberg for AI/BI Workloads at RedNote (Iceberg Summit 2025)

- 100 PB history, 3 PB daily increased
- Better than data warehouse performance directly on Apache Iceberg

Uniting Petabytes of Siloed Data with Apache Iceberg at Tencent Games (Iceberg Summit 2024)

- PB scale Iceberg Lakehouse with 10 second data freshness

StarRocks + Apache Iceberg

- Join Slack: https://starrocks.io/slack

Thank you. | CelerData