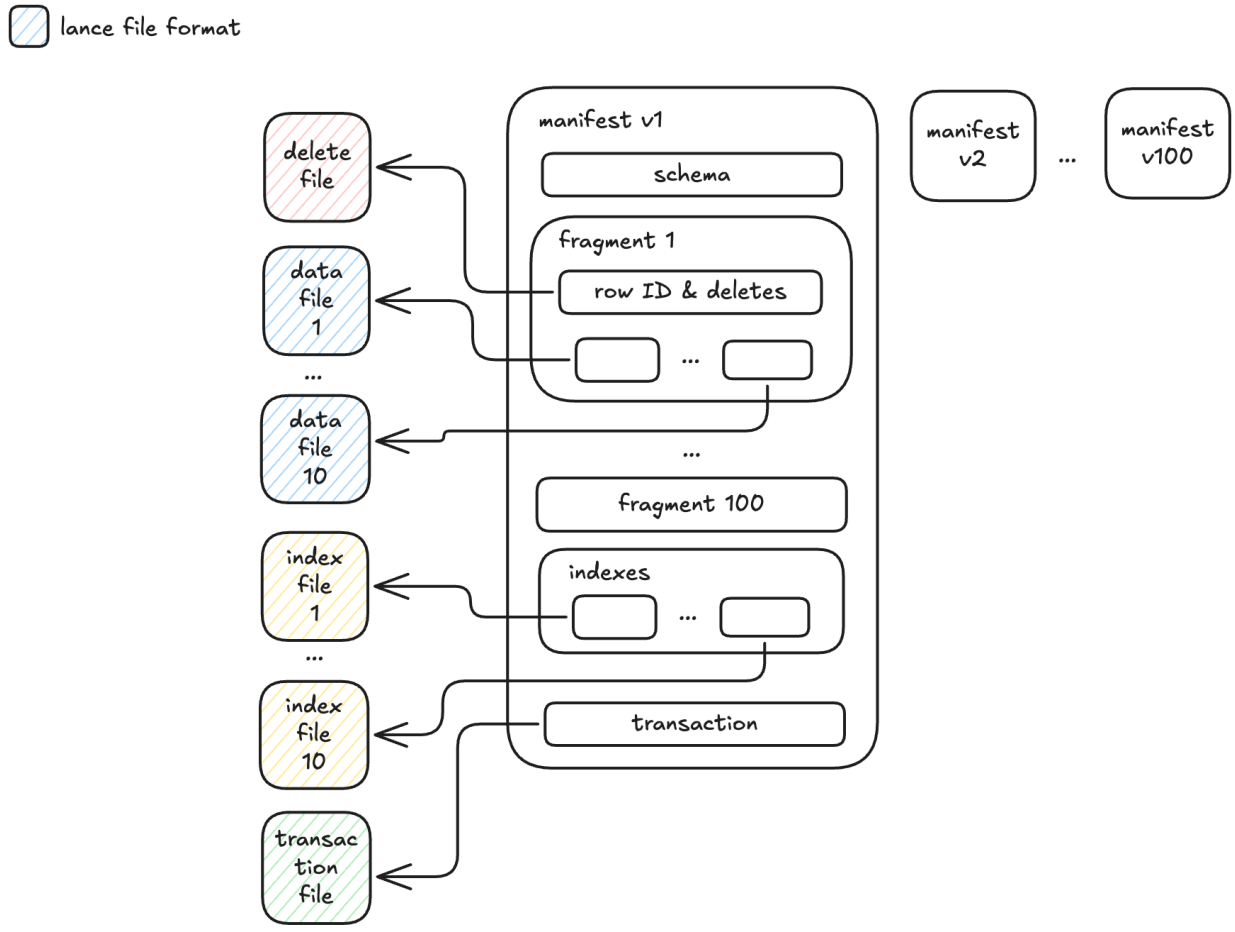
Apache Iceberg & Lance Integration

Jack Ye (jack@lancedb.com)
Weston Pace (weston@lance.com)

Apache Iceberg is introducing a pluggable DataFile API (ongoing development in Iceberg community) to allow new storage formats — beyond Parquet, ORC, and Avro — to seamlessly integrate at the file level. The Lance format, designed for efficient AI/ML and multimodal data workloads, supports columnar storage, fast random access, secondary indices, low metadata overhead and data evolution. This document describes how we could integrate Lance and Iceberg to get the best of both solutions.

Introduction to Lance

Lance is both a table and file format. Here is a basic overview of how the table and file format is related to each other:



The table format consists of a list of manifests with increasing version number. The latest manifest describes the latest state of the table and is retrieved by listing the table directory and finding the manifest with the latest version.

The manifest describes the data of a table in fragments, where each fragment holds a list of data files that each contain data in a subset of the columns in a table.

There are other important files with pointers in the Lance manifest, such as:

- the row ID and delete file for each fragment used for merge-on-read deletion
- a list of indexes for different columns of the table for query and search planning
- and a transaction file describing the transaction that produced the the new manifest for conflict resolution

Existing Work

As a part of the 2025 Iceberg Summit, we have presented a prototype integration that uses the DataFile API to read and write Lance data files. More detailed work can be found in this branch: https://github.com/westonpace/iceberg/tree/feat/lance-reader-writer

However, there are 2 main limitations of the prototype solution:

- Features like random access, secondary indexes, data evolution are built in Lance table format level, so Iceberg users cannot enjoy those benefits just by using the Lance file format.
- Although Lance file format supports multimodal data types, the lack of corresponding data type support in Iceberg still makes those features not usable.

Proposed Solution

Instead of treating a Lance data file as a data file in Iceberg, we will treat a Lance table manifest with a specific fragment as a data file in Iceberg.

This makes the data file much closer to a file format like Parquet:

- The full schema of the fragment is provided by the manifest, similar to the Parquet schema in Parquet footer
- Indexes in manifest could accelerate access to the fragment similar to stats in Parquet footer
- Because data files in a fragment are not aligned, a fragment can be viewed similar to a Parquet file with one row group

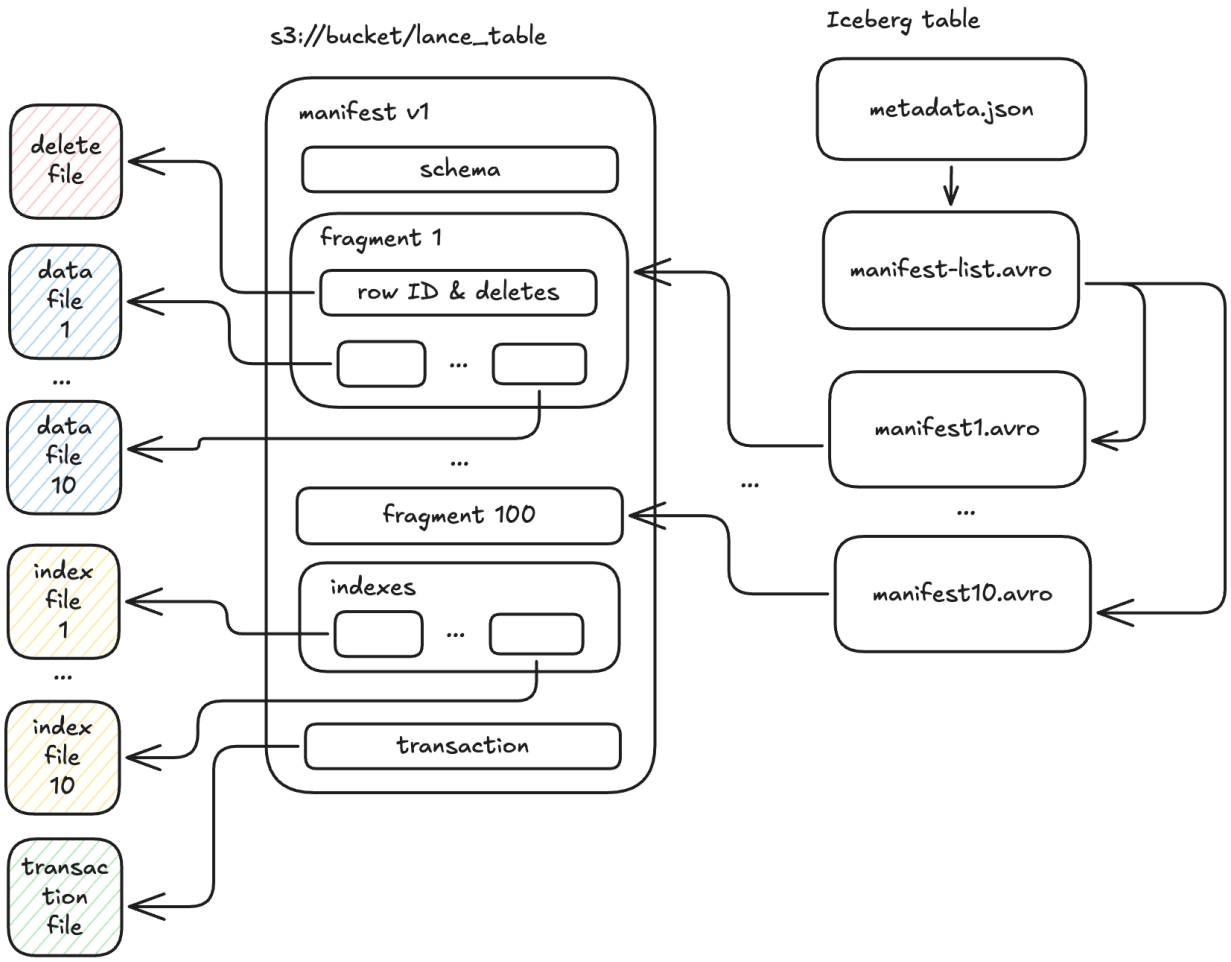
The location referenced in the Iceberg manifest needs to fully describe the information of the table and the specific fragment. We propose a file location should be in the form:

```
<table manifest location>#fragment=<fragment ID>
```

for example:

```
s3://bucket/lance_table/_versions/99999999#fragment=123456789012
```

Here is a graph describing the relationship between a Lance table and an Iceberg table with Lance format:

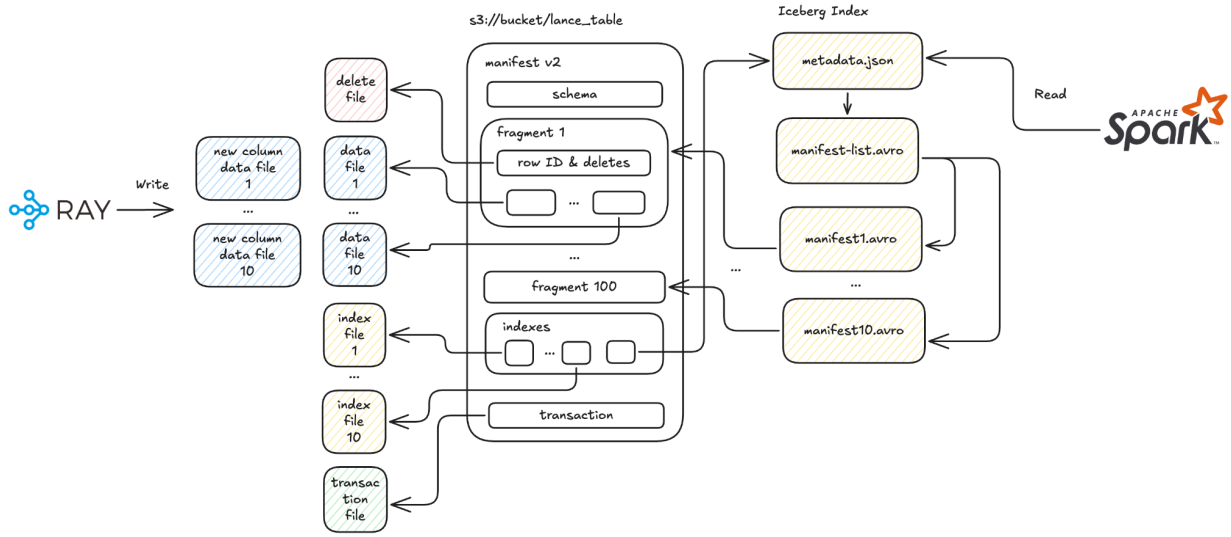


Because now each Iceberg data file has full access to the Lance table index, transaction, fragment index, etc. it can fully support key features in Lance like random access. The Lance reader leverages all that information to read data in a Lance table in the optimal way.

Reverse Integration - Iceberg Index for Lance Table

The Iceberg DataFile API feature is mainly targeted for using Iceberg with other formats. However, this solution unblocks another integration pattern, which we call a reverse integration.

Instead of directly creating Iceberg tables with Lance format, this approach allows users with a dual stack of both Lance and Iceberg-based workflows to create Lance tables, and read it using Iceberg-compatible engines.



For example, consider a ML feature engineering pipeline. A team might want to use Lance to allow easy data evolution to append hundreds of thousands of new feature columns to the table. At the same time, Lance will create an “Iceberg index”, which continuously reflects the latest state of the Lance table as an Iceberg table of Lance format, so that the team’s existing Spark-Iceberg connector can read the data. This will greatly improve the efficiency of the feature engineering pipeline because Lance does not require file rewrite for appending new column data. And on the other side the team can continue to use existing Iceberg tools to access the new data.