2025-06-17：OTFSG Osaka Meetup #01

OTFSG
Open Table Format Study Group
#otfsg_tokyo

MicroAd
Redesigning the Future Life

マイクロアドのData LakehouseとIcebergテーブルの最適化について

株式会社マイクロアド
永富 安和 (𝕏 @yassan168)

profile:

name: 'Yasukazu Nagatomi（やっさん）'

location: Kyoto

role:

- シニアエンジニア
- 'インフラエンジニア (Hadoop /Container) '

private:

organizer: |

'Rancher JP', 'Cloud Native JP', 'OTFSG'

favorites: ['日本酒', 'Craft Beer', 'Nature Aquarium']

socialmedia:

- 'X: @yassan168'
- 'GitHub: yassan'

アジェンダ

1. マイクロアドのデータ基盤について
2. Icebergテーブルの概要 ちょっとだけ…
3. Icebergテーブルの最適化に関する話
4. まとめ

マイクロアドについて

マイクロアドが提供する事業

自社製品である**「データプロダクト」**と、
主に 他社製品を扱う**「コンサルティング」**の二つの事業

コンサルティング

他社製品を扱う販売代理店ビジネス

海外コンサルティングサービス

企業のデジタルマーケティングにおける総合的な
コンサルティングサービスを提供

メディア向けコンサルティングサービス

メディア企業の広告収益最大化を支援する
コンサルティングサービスを提供

データプロダクト

自社製品のプロダクト提供ビジネス

オンライン

業種に特化したマーケティングプロダクト

UNIVERSE
シラレル IASO IGNITION Pantry

オフライン

屋外広告や交通広告のデジタルサイネージ

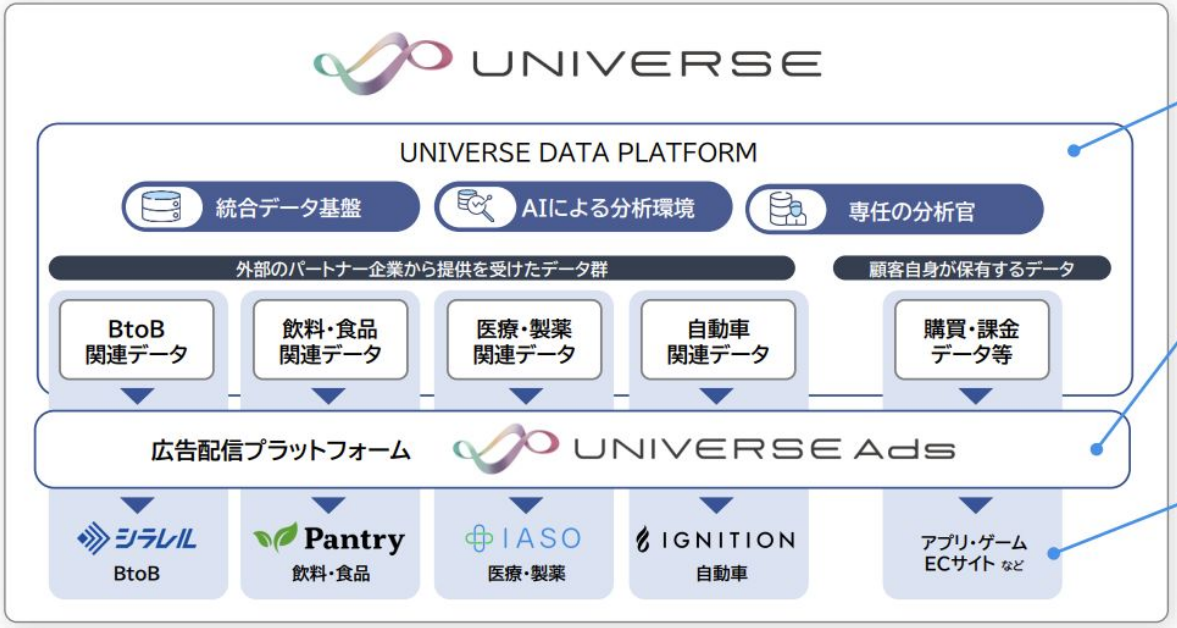
※2024年11月より運営する㈱MADSの株式譲渡により非連結化

MONOLITHS

※2025年9月期 決算説明資料より抜粋

データプロダクト「UNIVERSE」について

膨大なデータから業界業種毎の消費行動プロセスの違いを分析することで、
特定の製品カテゴリ毎に比較検討を開始したユーザ、最終的な購買検討段階に移行したユーザー、購買意欲が急激に高まった瞬間など、
個人の製品認知・購買プロセスの段階に応じたより効果的な広告配信を実現



UNIVERSE DATA PLATFORM

200を超える外部のデータ保有企業・メディア(※1)から、業界・業種に特化した消費行動データを収集・蓄積。そのデータを活用し、AIや専任の分析官によって業界業種毎の消費行動モデルの分析を行い、各種プロダクトの広告配信に活用。

UNIVERSE Ads

広告主向けの広告配信プラットフォーム。RTB(※2)という技術を用いて、ユーザー毎にリアルタイムに最適な広告を選択し、オークション形式で広告配信を行う。入札最適化機能によって、広告配信の費用対効果の最大化を実現。

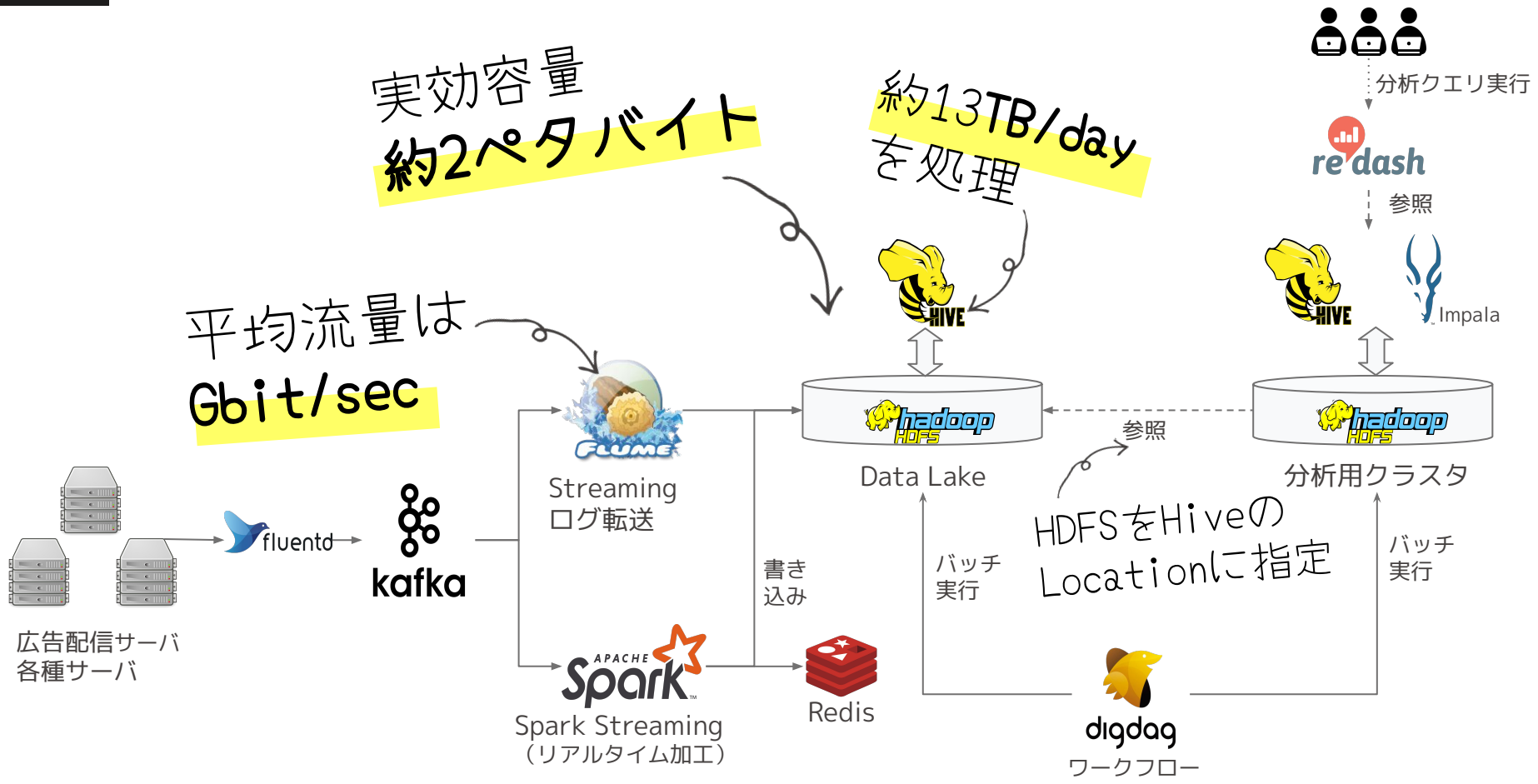
業界業種に特化したプロダクト展開

二つのプラットフォームを組み合わせることで、業界業種に特化した複数のプロダクトを開発。2024年9月時点で19業種に展開。

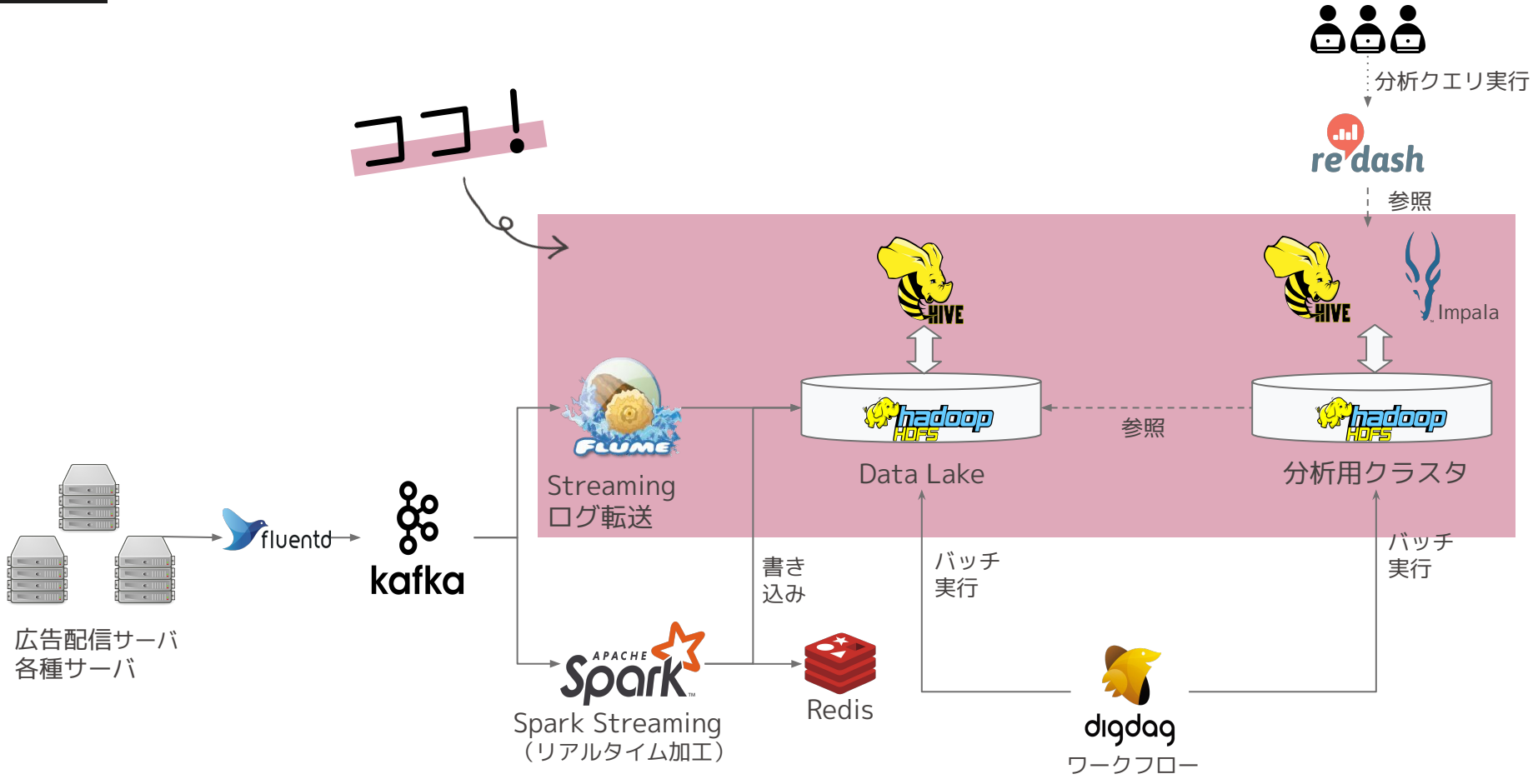
※1 2024年9月実績
※2 RTB： Real Time Biddingの略称

マイクロアドのデータ基盤について

これまでのデータ基盤の概要



今回の刷新範囲


ココ！
分析クエリ実行
re:dash
参照
HIVE
HIVE
Impala
FLUME
hadoop HDFS
hadoop HDFS
Streaming
ログ転送
Data Lake
参照
分析用クラスタ
fluentd
kafka
広告配信サーバ
各種サーバ
書き
込み
バッチ
実行
バッチ
実行
APACHE Spark
Spark Streaming
（リアルタイム加工）
Redis
digdag
ワークフロー

これまでのデータ基盤の課題

1. **利用しているHadoopディストリビューションが継続利用出来ない**
   - 有償のCloudera CDPも検討したが費用面がクリア出来ず見送り
     （PaaSも色々検討したが費用や技術課題がクリア出来ず見送り。5年償却で見たクラウドは高い）
2. **レプリカ３が結構にコストに直結**
   - 実効容量２PBということは、物理Diskで６PBは必要な現実
   - Hadoop 3.x なら Erasure Coding もあるけど、
     ディスク消費を抑えやすいが CPU 使用率が増える問題
3. **実質的に Compute と Storage を切り離してスケールしづらい**
   - NodeManager と DataNode は 同居が前提（データ局所性）
     → ノード追加＝CPU＋ディスク丸ごと増量
   - 専用 Computeノードも構築可能だがデータ転送増＆帯域ボトルネックで割に合わない

これまでのデータ基盤の課題

4. **Impalaの統計情報の運用が非常に煩雑かつ有効に利用出来ない**
   - ワイドカラム・大量データの大規模テーブルの場合、ほぼ使えない
     (inc_stats_size_limit_bytesを大きく設定できない問題)
   - 統計情報が不十分なので効率の悪いクエリになりがち
5. **ETL/ELT処理は Hive on MapReduce なので確実に処理するが遅い**
   - Tez・LLAPを使うべきだがHiveが古くて利用できない
6. **Hive Metastore が大量パーティションで悲鳴を上げる**
   - 大量の DROP PARTITION / DROP TABLE が メタストア DB ロックを長時間保持
   - Metastore Thrift サーバーが フル GC → 接続タイムアウト を誘発
7. **テーブル構造が複雑でSQLベースでETL/ELT処理するのが辛い**
   - 複雑なクエリになりがちで、改修に難易度が高く手間がかかる

新しいデータ基盤に求める事

1. ComputeとStorageを分離できる事
2. ETL/ELT処理はSQLだけでなくコードベースで処理したい
3. 大規模なテーブルでも統計情報を更新・有効活用できる事
4. Hiveの様にオンラインで柔軟なスキーマ進化が可能である事

MDP(Microad Data Platform) Overview

分析クエリ

re:dash

trino GATEWAY

(Active/Standby)

trino

RKE2

R/W

REST Catalog Server

(Active/Standby)

管理

RANCHER

管理

Sparkアプリをapply

digdag

Spark Connect で接続して処理

APACHE Spark

RKE2

YUNIKORN

R/W

DELL ECS(S3互換ストレージ)

Sparkアプリの リソース制御

ざっくりな使い分け

Business User

Ad-hoc クエリ

Redash

Engineer

ETL/ELT処理

Digdag

Trino Gateway

Spark Connect Client

Spark Operator

Gateway Layer

Trino on K8s

Coordinator

Worker

Spark on K8s

Spark Connect Server

SparkApp

Compute Layer

ICEBERG

Iceberg REST Catalog

Catalog Layer

Dell ECS (S3-Compatible)

Storage Layer

得られた効果

得られた効果

1. ComputeとStorageを分離出来る事

- 😆 ComputeはSpark・Trino、Storageは S3互換ストレージと明確に分離出来た
- 😆 YARN・Zookeeperの依存がなくなり構成要素が減った

2. ETL/ELT処理はSQLだけでなくコードベースで処理したい

- 😆 SQLでは表現しづらい複雑なビジネスロジックをPySparkのDataFrame APIで柔軟に実装可能になった

得られた効果

3. 大規模なテーブルでも統計情報を更新・有効活用ができる事

😆 Icebergテーブルを使うことで、Impalaで問題になっている統計情報が作れない問題が発生しない

- パーティション情報といった基本的な統計情報は
  テーブル書き込み時に意識せずに作成が可能になった
- 実行エンジン側で柔軟に統計情報の更新も可能

4. Hiveの様にオンラインで柔軟なスキーマ進化が可能である事

😆 Iceberg特有のスキーマ進化（or パーティション進化）により、
以前より柔軟な運用が選択可能になった

Apache Icebergテーブルについて

Apache Icebergとは

Netflix、Apple、Tencent、Pinterest などの多くの大企業も本番利用している
巨大で複雑なデータセットにも対応可能なOpen Table Formatの1つ。

特徴

- 複数のエンジンから同じテーブルを操作できる
  - 例：Spark・Flink・Trino・Presto・Hive・Impala
- 安全なテーブル操作
  - ACIDなコミット（Atomic Commit + Optimistic Concurrency）
- スキーマ／パーティション進化
  - 既存データをコピーせず列やパーティションを変更
- Hidden Partitioning
  - パーティションを意識せずにクエリしたり、パーティションレイアウトを変更可能
- タイムトラベル／ロールバック
  - “過去の状態” でクエリ／ロールバックが可能

”Iceberg”＝仕様だけ　― 実装はエンジン & SDK が担当
言わば、Iceberg はデータ版 HTTPの様なもの。

- **HTTP**：プロトコル仕様だけを定め、Apache/Nginx/Chrome が実装
- **Iceberg**：テーブル仕様 だけを定め、Spark/Trino/PyIceberg … が実装

つまり、、、

「”Iceberg” という振る舞い」をテーブル仕様に基づき独自に実装

テーブル仕様は公開され・コミュニティで仕様を決定しているので、
様々な実行エンジン＆各言語SDKによりIcebergテーブルに対して操作が可能。

＝ベンダーロックインがない。

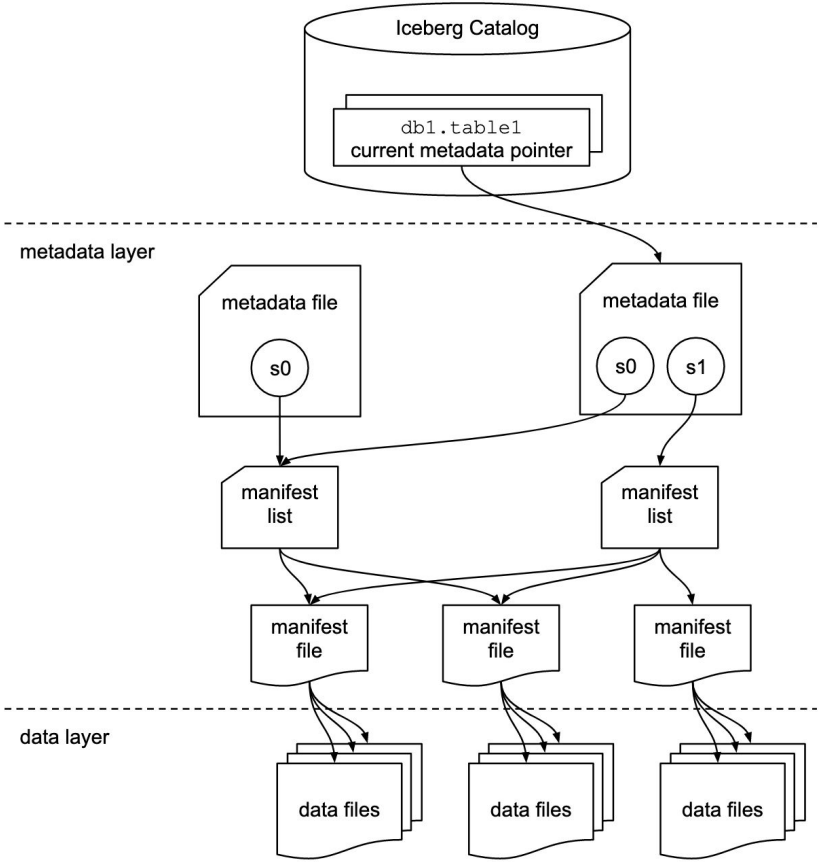
とは言え、、
StorageやComputeのベンダーロックインは回避できたけど、
今度は各社カタログでロックインしようとする動きが、、、

Icebergテーブルのアーキテクチャ

大きく３つに構成されます。

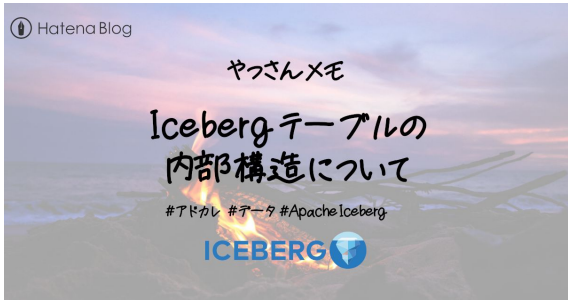
1. カタログ
2. メタデータ層
3. データ層

今回は、３→2→1 の順で、
下から順に説明していきます。



⚠️ ここから先は Icebergテーブル仕様 v2 を前提とします

尺が足りないので割愛。



以下で説明してます！

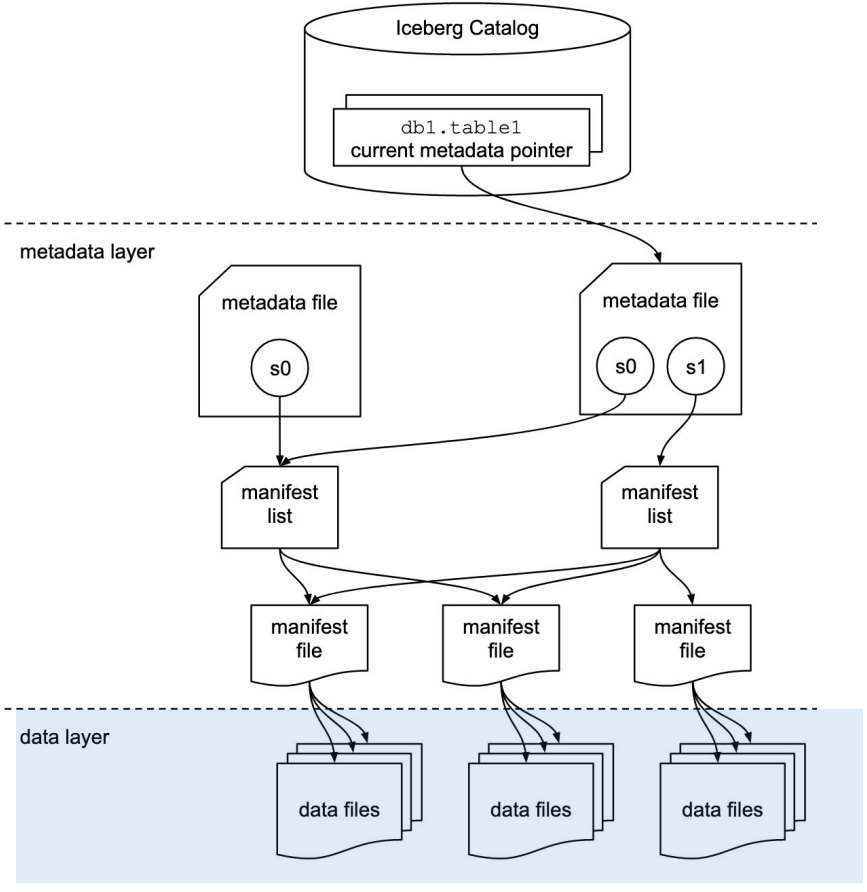
[Icebergテーブルの内部構造について - やっさんメモ]

Icebergテーブル：データ層

以下の３つで構成

- データファイル
- 削除ファイル
- Puffinファイル

データ層は、分散ファイルシステム（HDFS）や、オブジェクトストレージに対応。



Icebergテーブル：データ層

データファイル

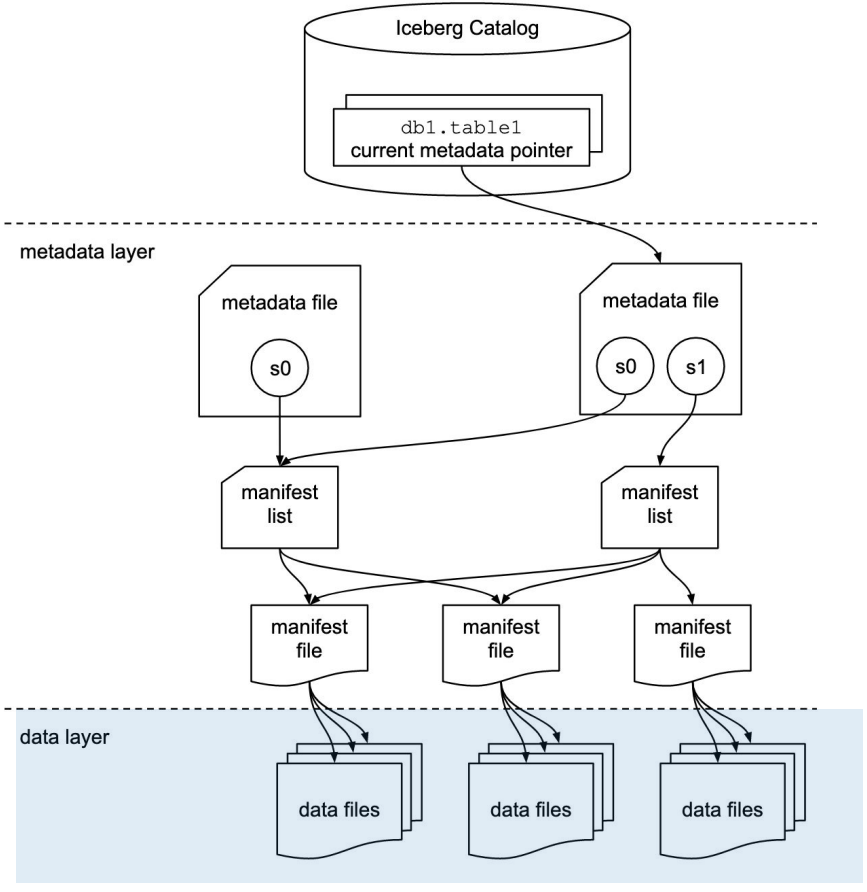
テーブルのデータそのもの。
Parquet・Avro・Orcに対応。
※同一テーブル中に複数のフォーマットでもアリ

削除ファイル

テーブルのデータセット内の どのレコードが削除されたかを追跡する際に利用。

以下の2種類の戦略がある。

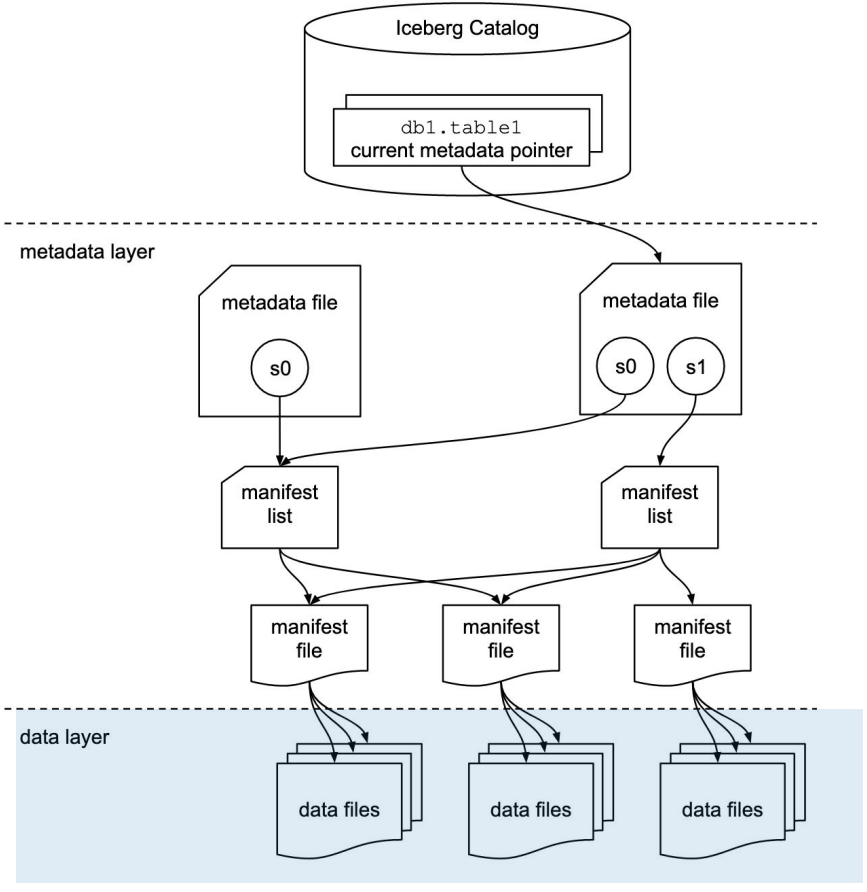
- Copy-On-Write(CoW)
- Merge-On-Read(MoR)



Icebergテーブル：データ層

Puffinファイル

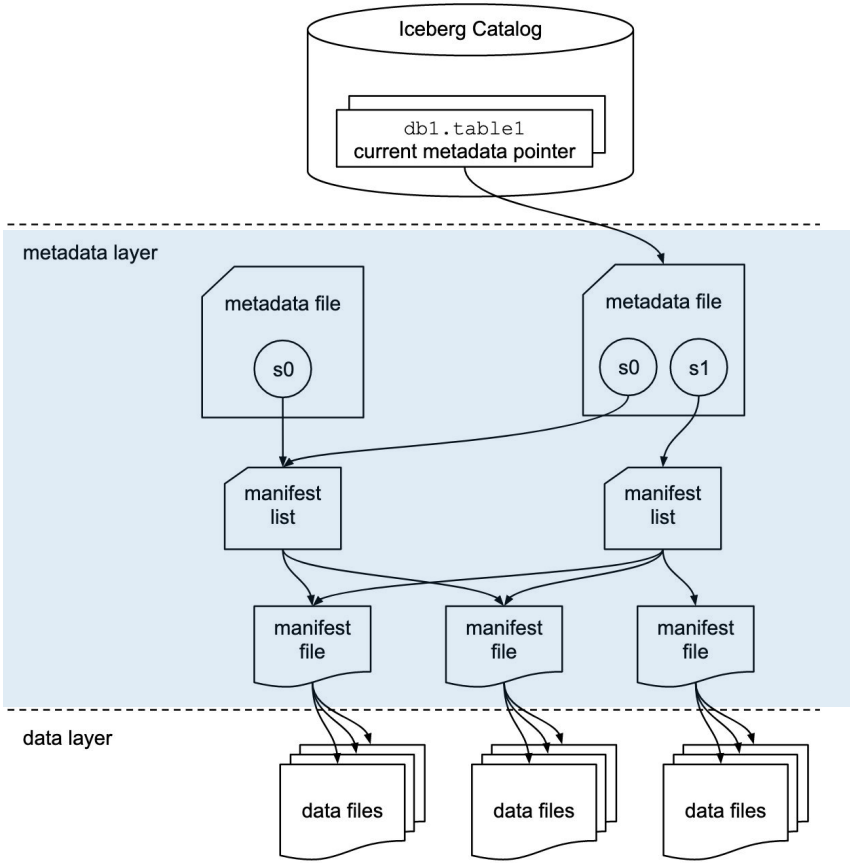
データファイルやメタデータファイルに格納される統計情報よりも更に
幅広いクエリのパフォーマンスを
向上させるデータテーブル内のデータに関する統計情報とインデックスを格納。



Icebergテーブル：メタデータ層

以下の３つで構成され、ツリー構造になっている。

- メタデータファイル
- マニフェストリスト
- マニフェストファイル



Icebergテーブル：メタデータ層

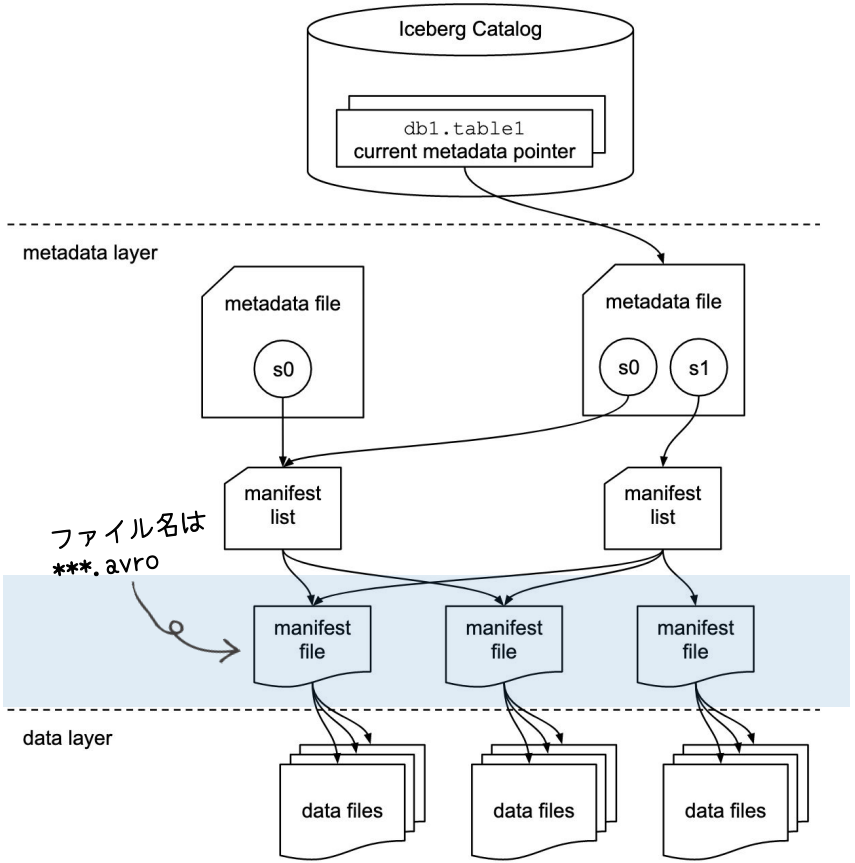
マニフェストファイル

データ層内のファイルの追跡と、各ファイルに関する追加の詳細や統計情報を保持。

１つのマニフェストファイルには、
１つの「データファイル」または
「削除ファイル」のみが含まれる。

各マニフェストファイルには、以下が含まれる

- パーティションのどこに紐づいているか
- レコード数
- カラムの上限と下限値
- etc..



Icebergテーブル：メタデータ層

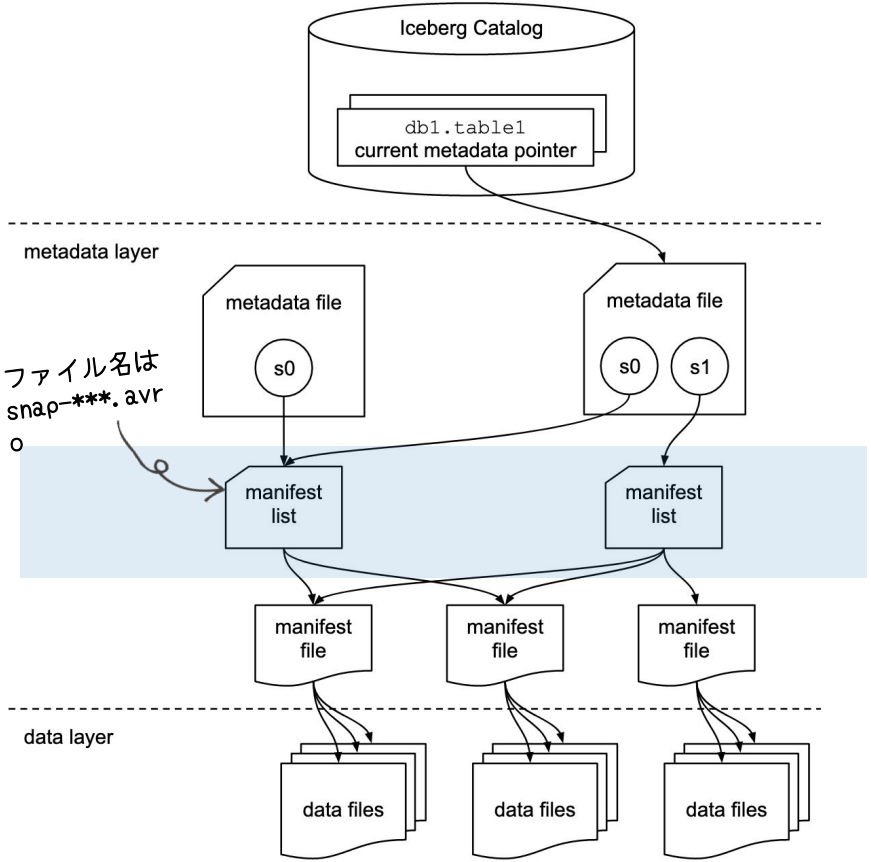
マニフェストリスト

マニフェストファイルのリスト。

マニフェストリストには、テーブルのある時点での状態である「スナップショット」を構成する各マニフェストファイルに関する情報がある。

その情報には、以下が含まれます。

- マニフェストファイルのパス
- どのスナップショットの一部として追加されているのか
- 属するパーティションに関する情報
- 追跡するデータファイルのパーティション列の下限と上限値



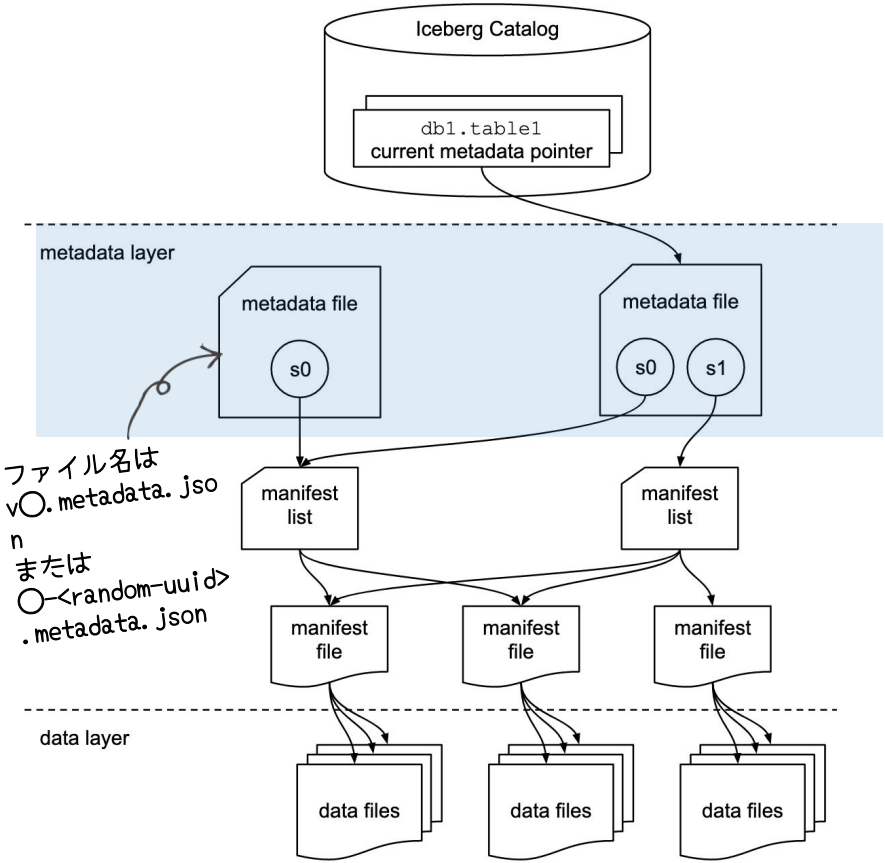
Icebergテーブル：メタデータ層

メタデータファイル

テーブルに関するメタデータを格納し、
以下の情報が含まれています。

- テーブルのスキーマ
- パーティション情報
- スナップショット
- 現状のスナップショットはどれか

Icebergテーブルが変更されるたびに
新しいメタデータファイルが作成されます。
また、次に説明するカタログにより
アトミックに最新版のメタデータファイル
として登録される。



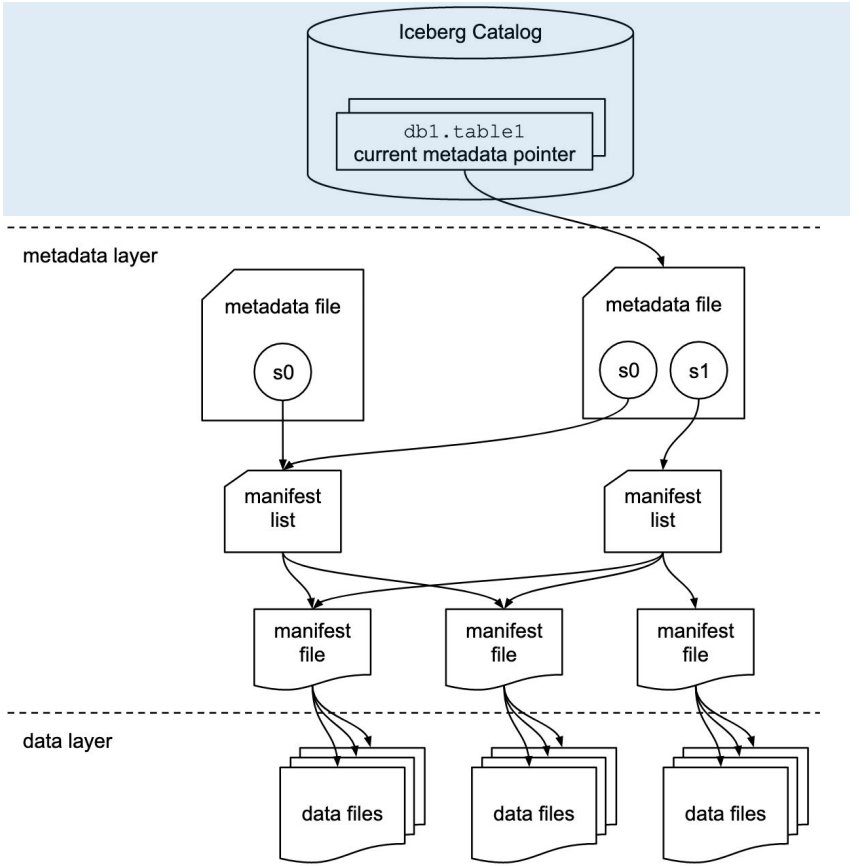
Icebergテーブル：カタログ

カタログ

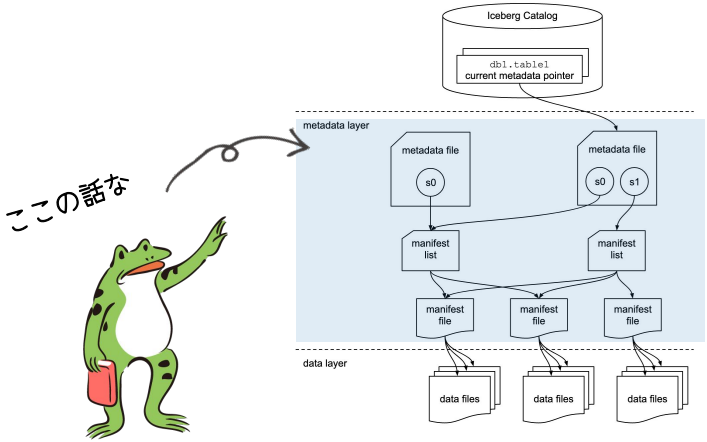
最新のメタデータがどこにあるかを保持する。その為、カタログの主な要件は、現在のメタデータのポインタを更新するためのアトミックな操作をサポートすることにある。

カタログには色々な実装があります。

- Hadoopカタログ(File System)
- Hiveカタログ
- Nessieカタログ
- AWS Glueカタログ
- RESTカタログ



メタデータ層の動きについて



テーブル作成直後（おさらい）

メタデータファイル

＝「テーブル全体の現行設定とスナップショット一覧を管理」

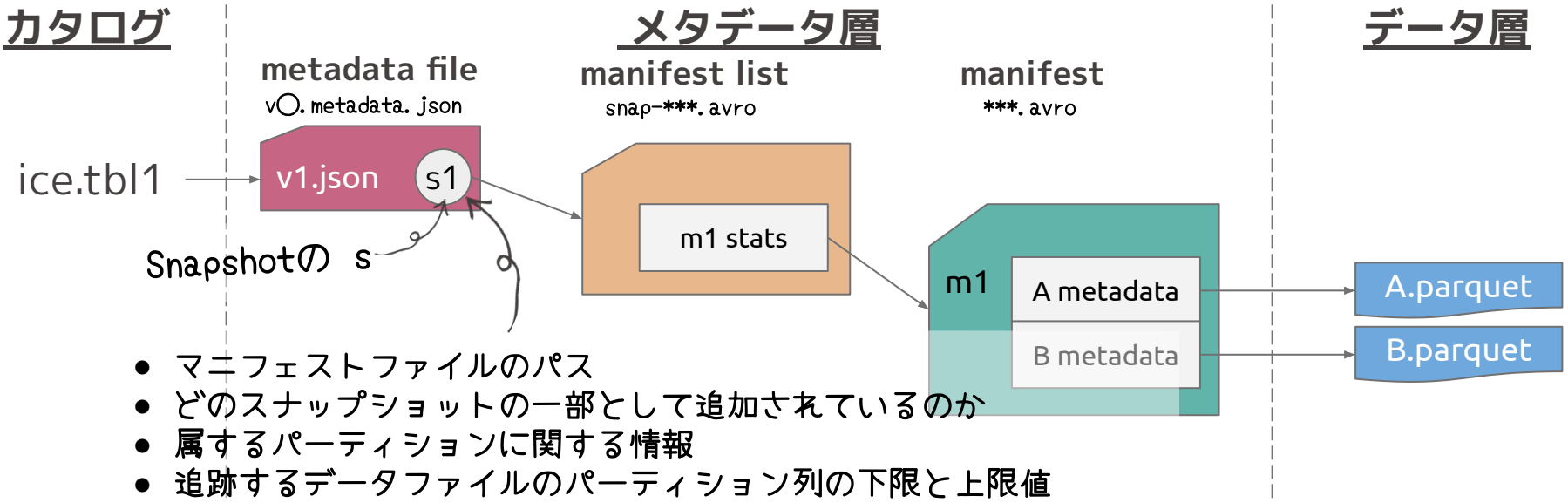
💡カタログで管理してるのは、こいつの最新のパス

スナップショット

＝「ある時点でのテーブルの全体像（＝ データファイルの集合）」

💡タイムトラベル／ロールバック単位。

メタデータはツリー構造（おさらい）



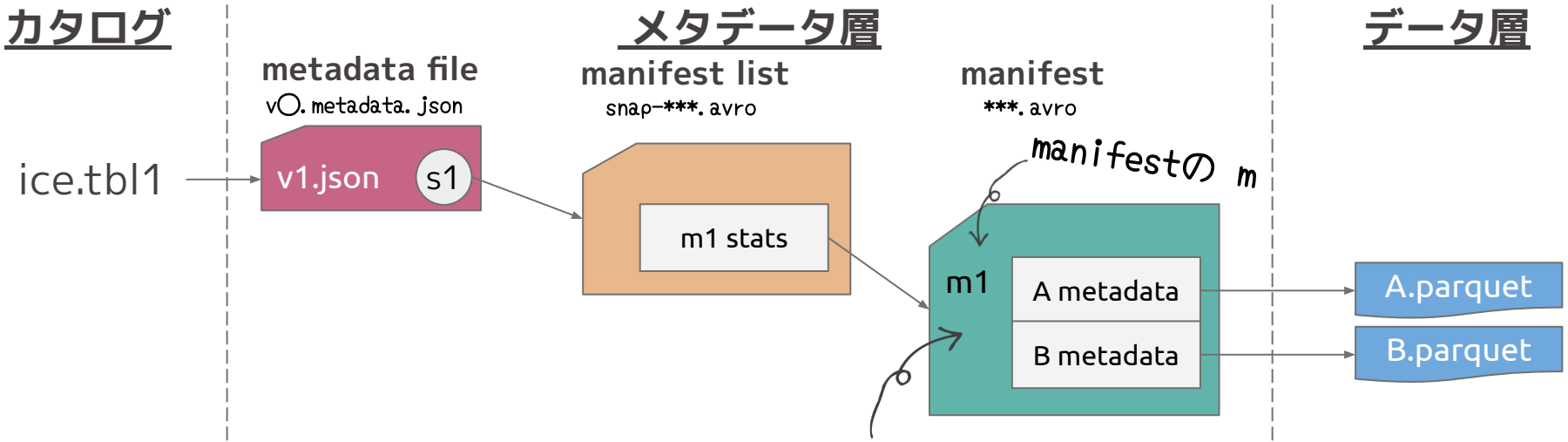
- マニフェストファイルのパス
- どのスナップショットの一部として追加されているのか
- 属するパーティションに関する情報
- 追跡するデータファイルのパーティション列の下限と上限値

マニフェストリスト

＝「スナップショットが参照するマニフェストファイルのリスト」

💡このスナップショットに何が入ってるの？を高速に把握

メタデータはツリー構造（おさらい）



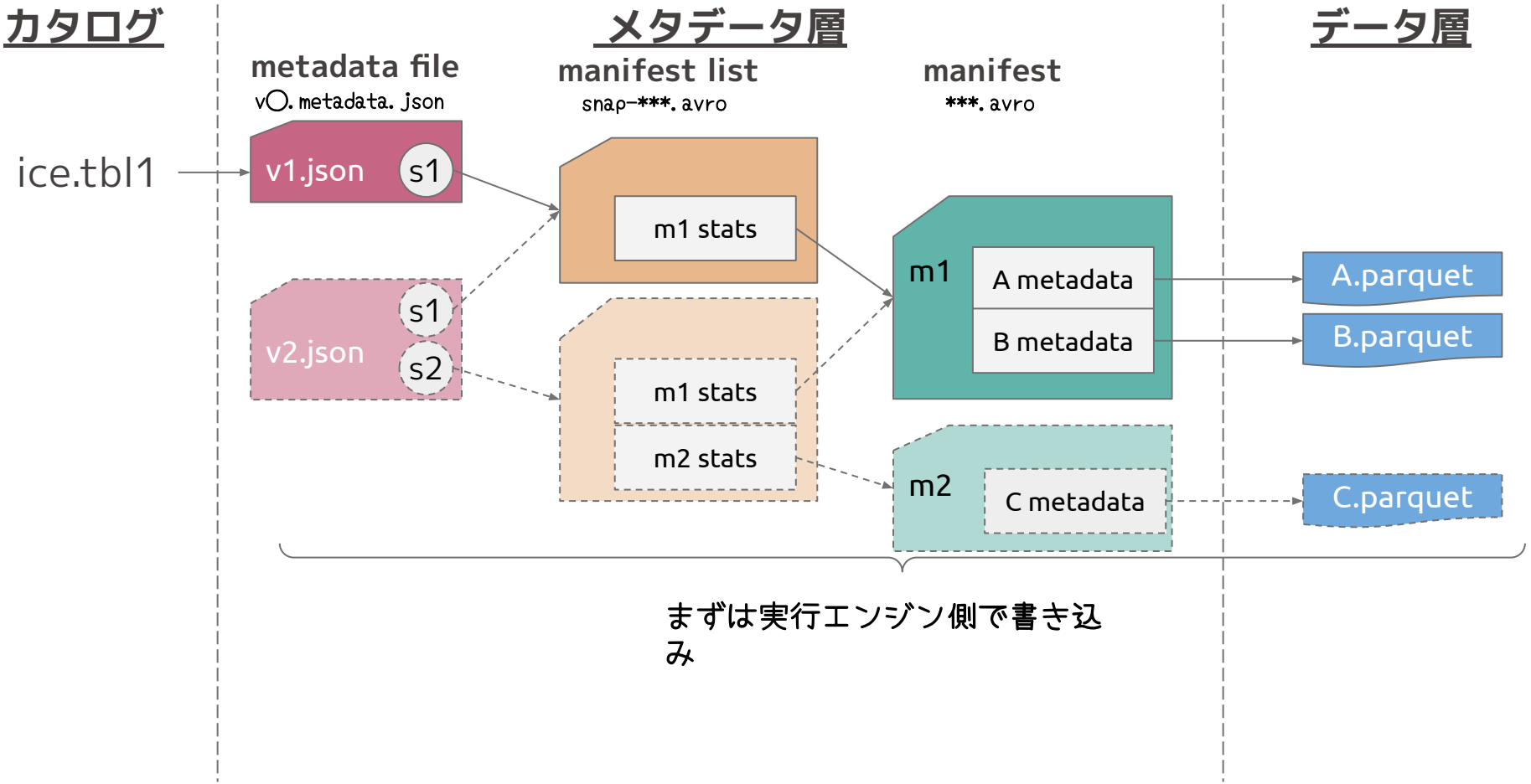
- パーティションのどこに紐づいているか
- レコード数
- カラムの上限と下限値

マニフェストファイル

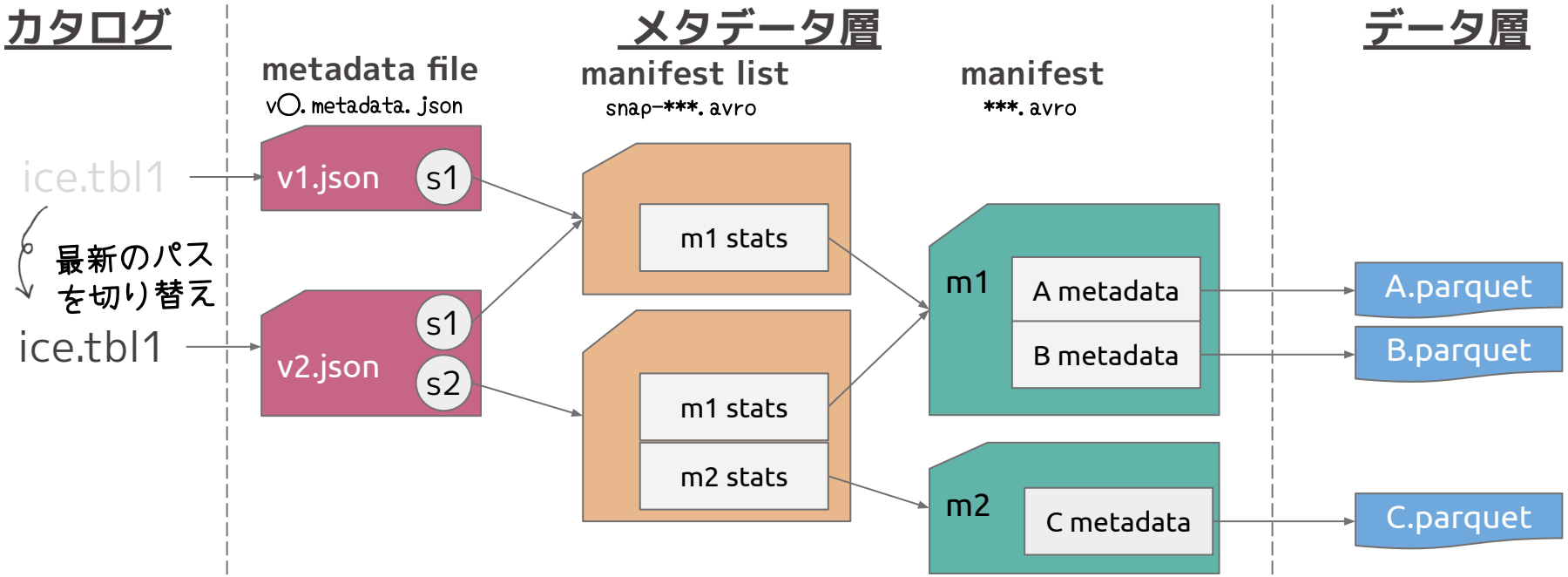
＝「データファイルのメタ情報（パーティション値・列統計など）」

💡**スキャン計画時に不要ファイルを即座に除外**

データを挿入（書き込み中）



データを挿入（書き込み完了）



📑同時書き込み制御の話はべりんぐさんのブログがとても参考になります！

Apache Icebergの同時実行制御における競合解決の仕組みと注意点
https://bering.hatenadiary.com/entry/2025/01/18/234339

Icebergテーブルのメンテナンス

なぜIcebergテーブルにメンテナンスが必要？

よくある “詰まり” シナリオ

- **高頻度アップサート／ストリーム書込み**
  → マニフェストが爆増して **プランニング遅延**
- **小さなバッチ Insert** が続く
  → “Smallファイル問題” で **I/O 効率 ↓↓↓**
- **分析速度を上げるための並べ替え**（ソート/クラスタリング）
  → 既存ファイルを**再編成**しない限り効果が出ない

放置すると…

1. **Query Planning が遅い**
   - 1 クエリで数千 manifest を開く事態も
2. **スキャン効率 & 物理コストが悪化**
   - S3 API コール増加 / キャッシュ不可ファイル増殖

なぜIcebergテーブルにメンテナンスが必要？

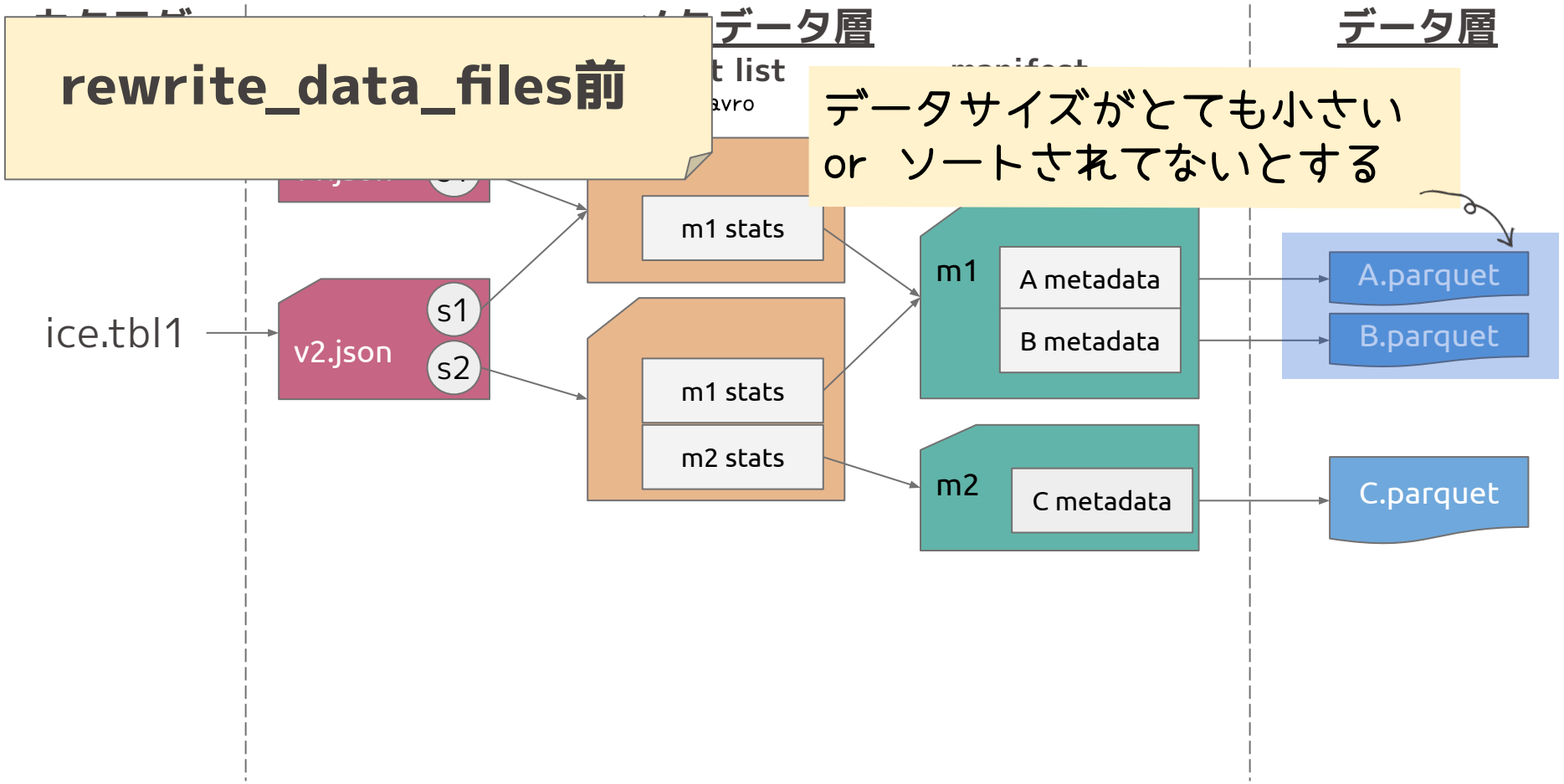
なので、メタデータ層・データ層において
”定期的なお掃除”が必要！

2. **スキャン効率 & 物理コストが悪化**
    - S3 API コール増加 / キャッシュ不可ファイル増殖

📑以降はSparkのプロシージャを例に解説していきます

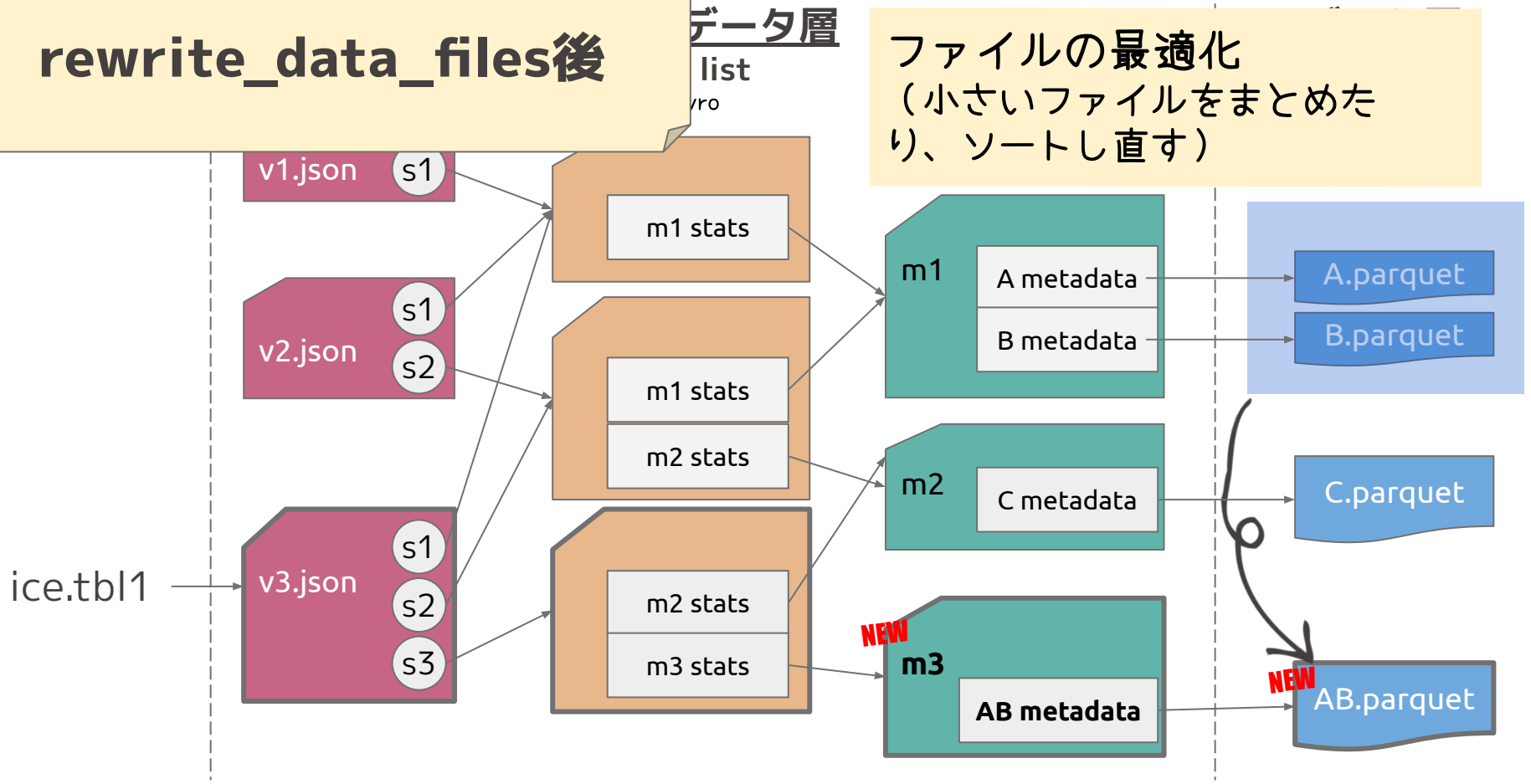
データ層の整理

データファイルのCompaction（rewrite_data_files）



データファイルのCompaction（rewrite_data_files）

rewrite_data_files後



データファイルのCompaction

行レベル削除ファイル（Position Delete）のコンパクション

rewrite_position_delete_files

- **merge-on-read （MoR）** モードで増える小粒 Position Delete をまとめ、
  - 目標サイズ（既定 64 MB）で再書き出し
  - 「宙ぶらりん（dangling）」な delete 行も同時に除去
- **Position Deleteは Icebergテーブル仕様 v3 では非推奨**
  （新規追加は禁止／既存は読み取り可）
  - 後継は Deletion Vectors (DV)

日時キー付きテーブルの「不要データ」の削除

勘違いしやすいやつの１つ。

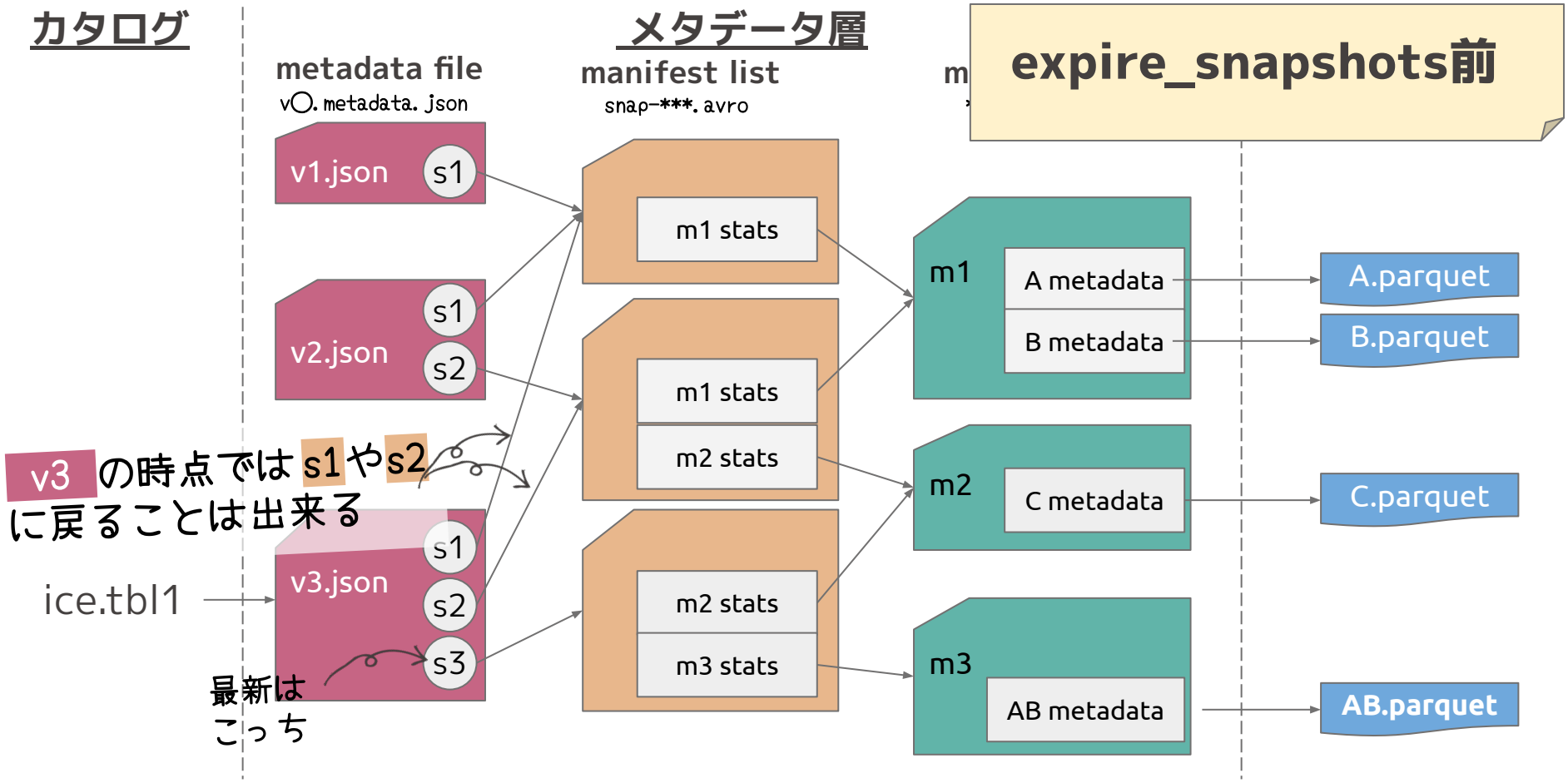
スナップショットを期限切れにしたところでレコードは削除されない。

Iceberg は 自動 TTL を持たない。
古データを捨てるには 2 段階で手動実行が必要

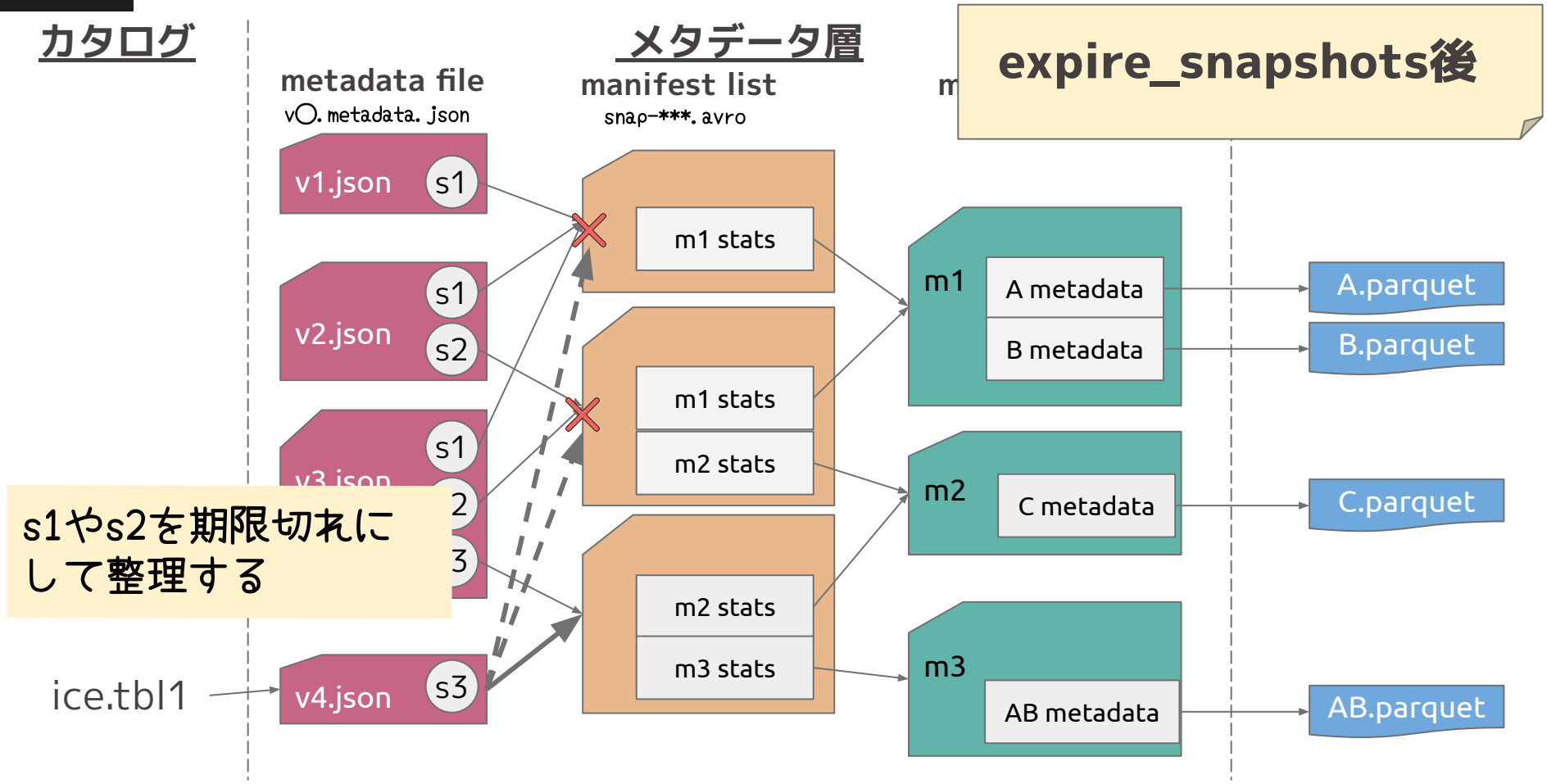
1. **DELETE文で論理削除**
   - -- 90日前のレコードは削除
     DELETE FROM c.ns.tbl **WHERE** event_date < date_sub(current_date, 90);
2. **Deleteされて紐づかなくなったスナップショットをexpire_snapshotsすることでファイルを物理削除**

メタデータ層の整理

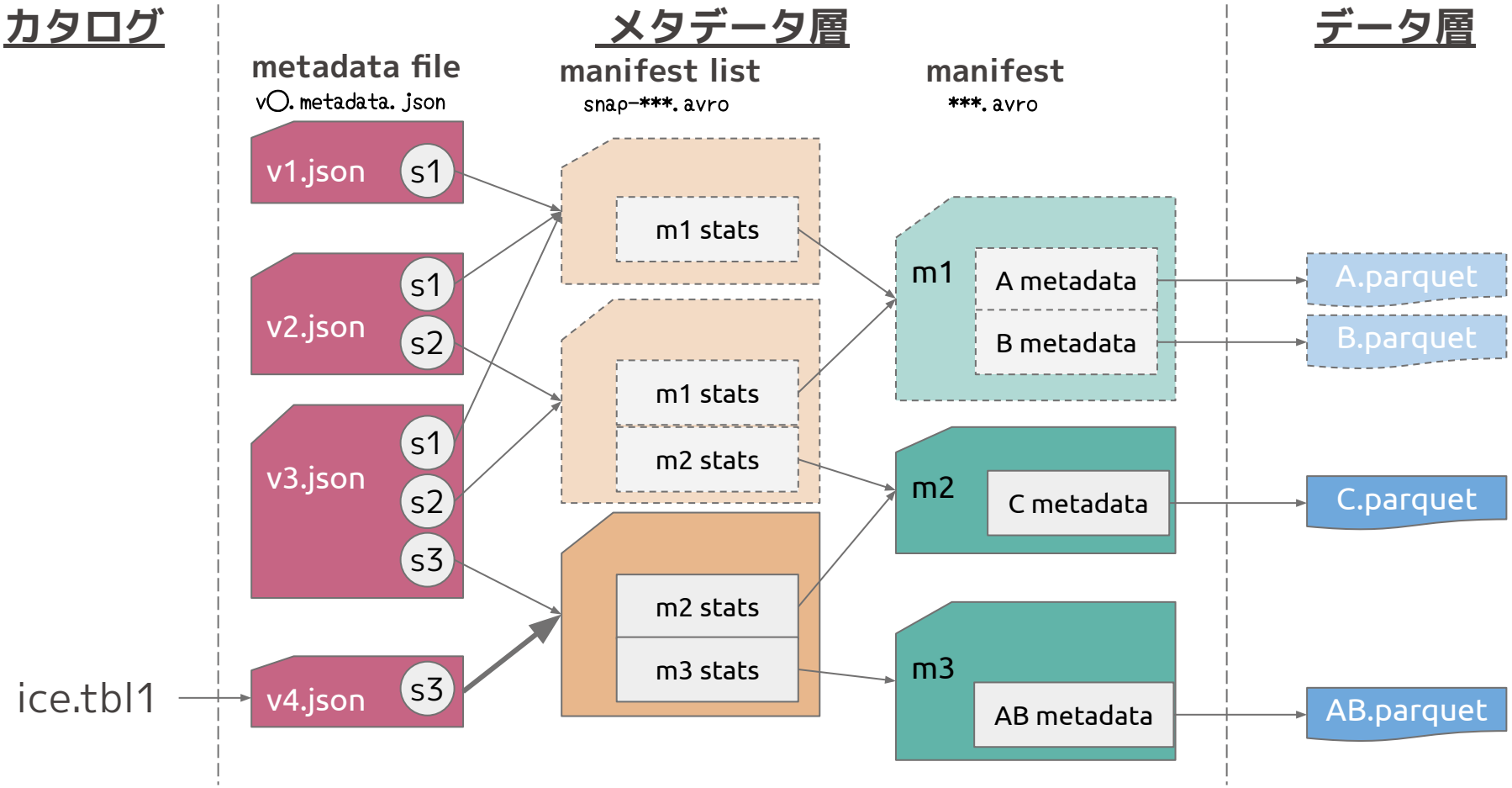
メタデータのCompaction（expire_snapshots）



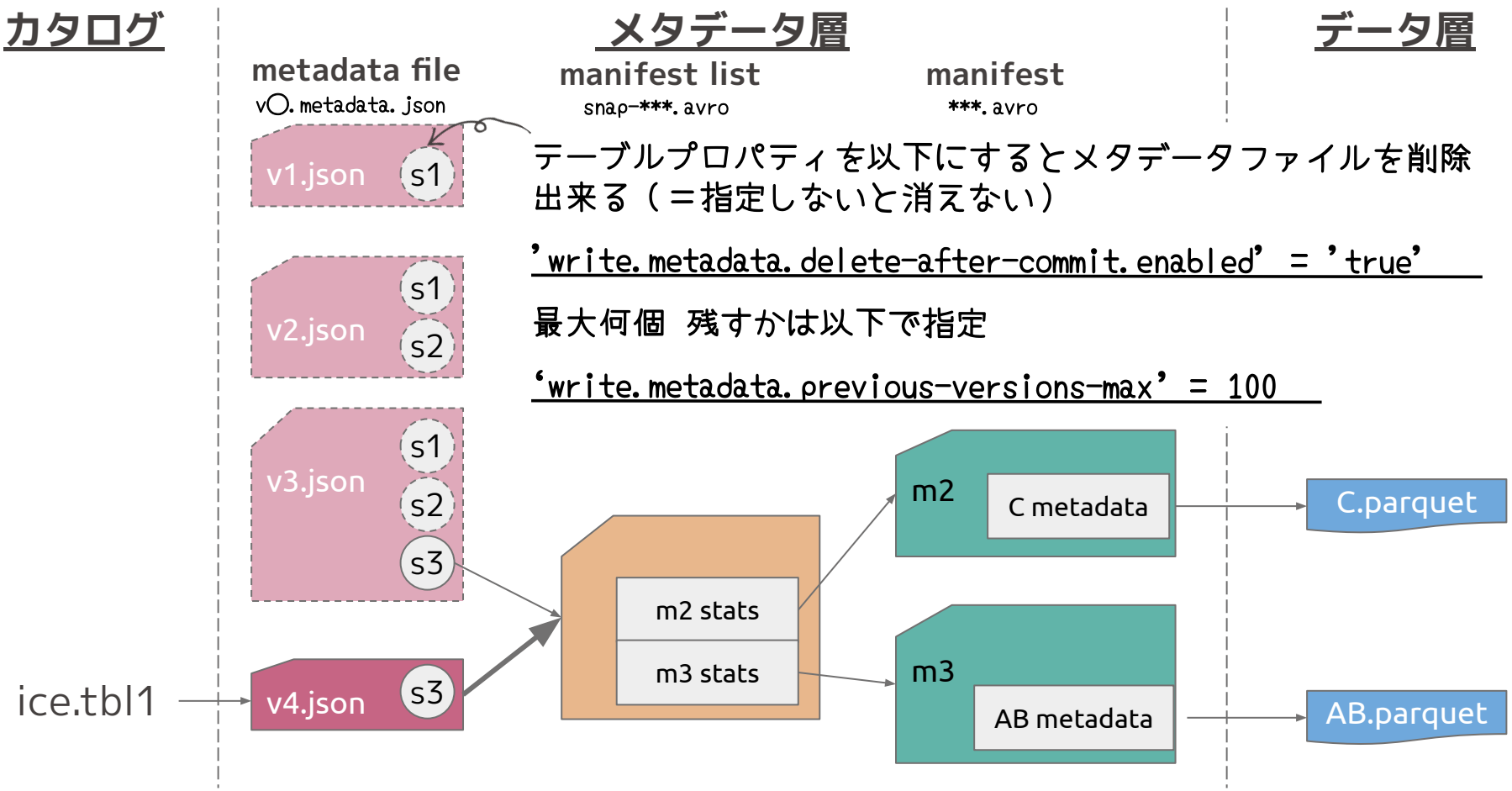
メタデータのCompaction（expire_snapshots）



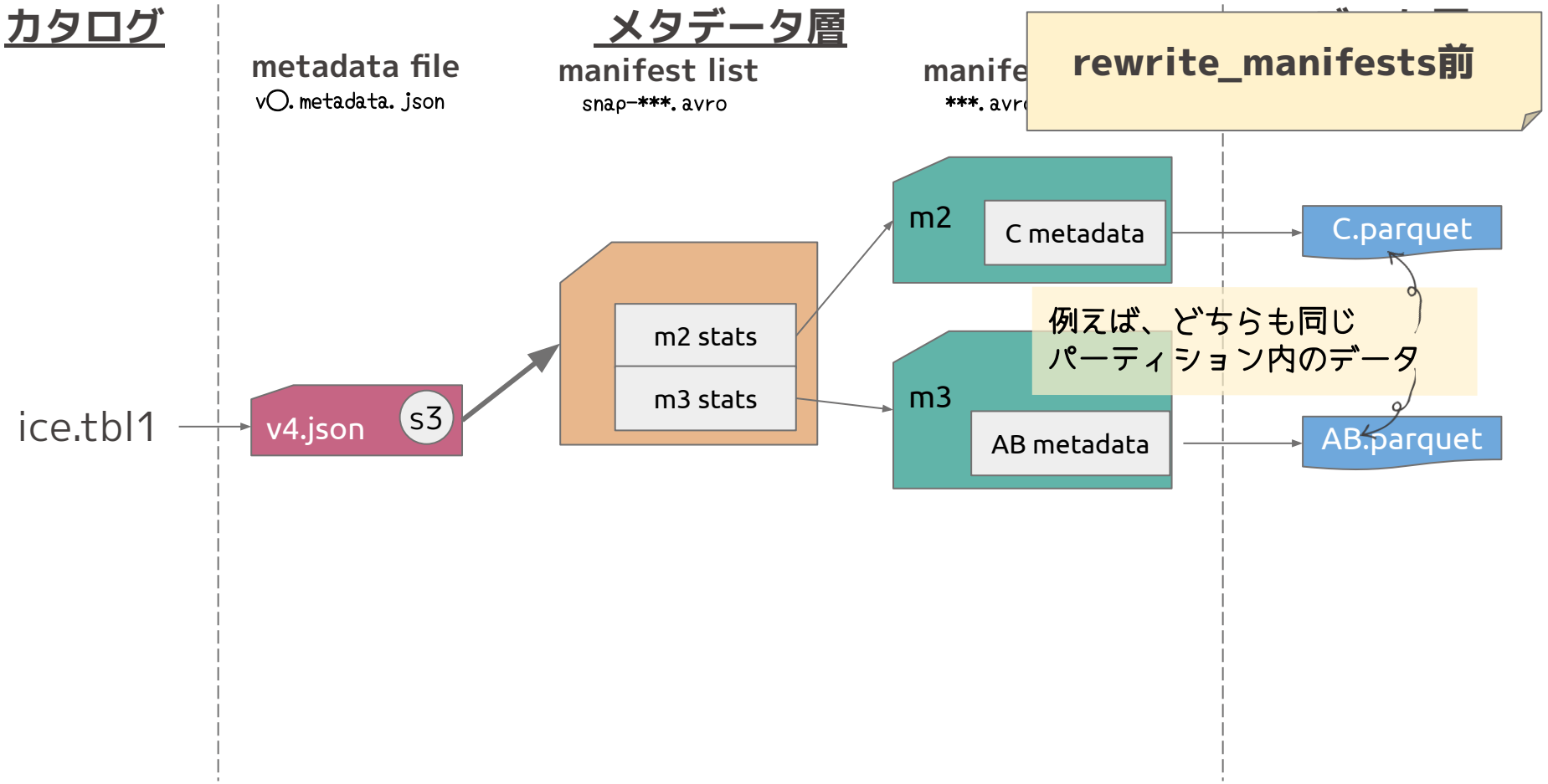
メタデータのCompaction（expire_snapshots）



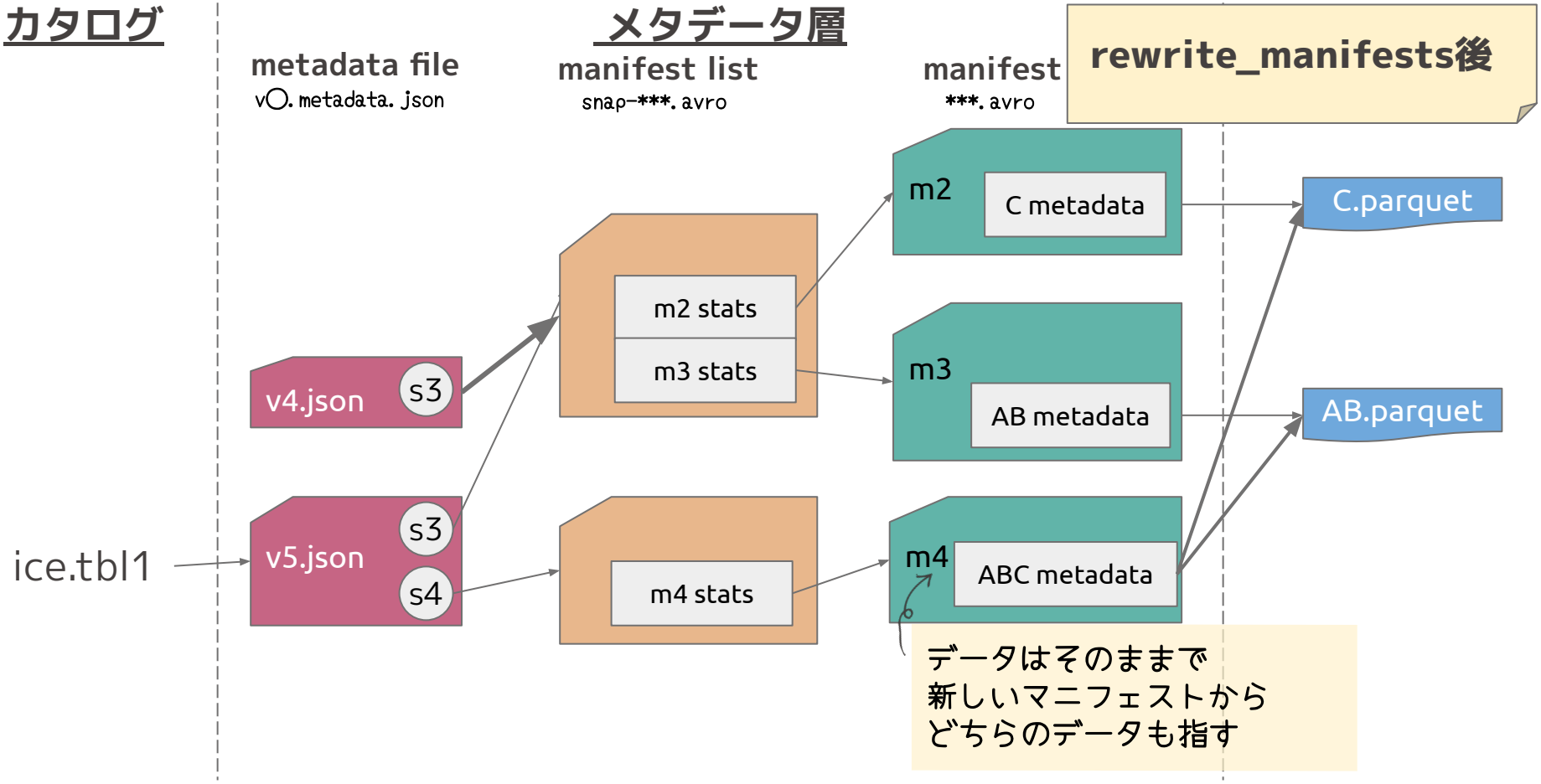
メタデータファイルはいつ消えるの？



メタデータのCompaction（rewrite_manifests）



メタデータのCompaction（rewrite_manifests）



他にも、、

孤立ファイルのクリーンアップ (remove_orphan_files)

- メタデータに参照されない orphan file （孤児ファイル）を**物理**削除
- 既定で 3 日より古いファイルのみ 対象
- older_than でデフォルト「3日」から調整は可能

注意点

- **別でデータ書き込み中のファイルが remove_orphan_files の対象期間中だった場合、誤って削除する可能性がある**ので、ワークロードに合わせた older_than の設定が重要
  - 実行時には戻り値の orphan_file_location をログ出力しておいて、何を削除したのか追跡可能にしておくと安心

問題です！！！

どの順で実行するとコスパが良いでしょうか？

1. delete from c.ns.tbl where ** (要らなくなったレコードの削除)
2. remove_orphan_files
3. rewrite_data_files
4. rewrite_position_delete_files
5. rewrite_manifests
6. expire_snapshots

正解は、、、（多分だけどな）

1. 古いレコードの削除
   delete from c.ns.tbl where **
2. データの最適化（適正サイズ化・ソート）
   rewrite_data_files / rewrite_position_delete_files
3. マニフェストの最適化
   rewrite_manifests
4. スナップショットをGC
   expire_snapshots
5. メタデータのどこにも紐づいていないファイルの削除
   remove_orphan_files

期限切れのスナップショットに紐づく
要らないデータやマニフェストも削除

注意点

- データまたはメタデータを整理に伴い、コミットが発生するので、
  対象が他メンテナンス処理含む**全てのデータ更新と重複させない対策が必要**
  （被ったらプロシージャが失敗する）

  例えば、以下のプロシージャのパラメータ **older_than を使って対象期間を指定**出来る

  - remove_orphan_files（孤児ファイルの削除）
  - expire_snapshots （スナップショットのGC）
- データのソートやクラスタリングは慎重に選ばないと、デカいテーブル程、やり直しにはかなりの計算コストが必要になるので注意して検討が必要

悩みのタネ — みなさんどうしてますか？

カラムの統計ってどうしてますか？

min / max / null_count はテーブル書き込み時に作成済みだけど、、

compute_table_stats 使ってNDV (distinct_count)作成してますか？

- かなりの計算コストを支払うので、そこまでやるメリットがどれくらいあるのか？
  - 追記/削除が多いテーブルはNDV が陳腐化するから再計算も必要になる
- 高カーディナリティになりがちのSTRING型カラムは含めてますか？

Icebergテーブルの最適化のタイミングの決定する基準は？

- 一律 1日1回とか？
- remove_orphan_files と expire_snapshots の older_than ってどうしてます？

まとめ

まとめ

- マイクロアドのData Lakehouse事例紹介
- Icebergテーブルについて
- Icebergテーブルの最適化について

参考

- Apache Iceberg とは何か - Bering Note – formerly 流沙河鎮
  https://bering.hatenadiary.com/entry/2023/09/24/175953
- Icebergテーブルの内部構造について - やっさんメモ
  https://yassan.hatenablog.jp/entry/advent-calendar-2023-1201
- Apache Iceberg's Best Secret: A Guide to Metadata Tables - YouTube
  https://www.youtube.com/watch?v=4K0HBWUIEKI
- Apache Icebergの同時実行制御における競合解決の仕組みと注意点 - Bering Note – formerly 流沙河鎮
  https://bering.hatenadiary.com/entry/2025/01/18/234339
- Lakehouse テーブル形式とカタログの活用 | データエンジニアリング オープンフォーラム 2025 - YouTube
  https://youtu.be/wLsMrdfBuwU?si=4Irh8Ds2HV6EmN7s
- ダ鳥獣戯画
  https://chojugiga.com/