AWS

SUMMIT

Amazon Redshift テーブル設計詳細ガイド

分散スタイルとソートキーの決定方法

アマゾン ウェブ サービス ジャパン株式会社
柴田竜典

2017/6/1

amazon web services

自己紹介

柴田竜典［シバタツ］

- データベース関連の
  相談ごと何でも担当
    - AWSへの移行を機に
      RDBMSをAuroraに
      乗り換えたい
    - オンプレミスOracleを
      AWSにフォーク
      リフティングしたい
- 好きなAWSのサービス: S3

@rewse

すでにRedshiftをお使いの方の悩み

- クエリー性能をさらに向上させたい
- 同時実行を上手にさばきたい
- 料金を抑えたい

などなど

クエリー性能向上に大切なことは何か

AWS Documentation » Amazon Redshift » データベース開発者ガイド » Amazon Redshift のベストプラクティス » Amazon Redshift のテーブル設計のベストプラクティス

Amazon Redshift のテーブル設計のベストプラクティス

トピック

- テーブル設計のチューニングに関するチュートリアル
- 最良のソートキーの選択
- 最適な分散スタイルの選択
- COPY による圧縮エンコードの選択
- プライマリキーおよび外部キーの制約の定義
- 最小列サイズの使用
- 日付列での日時データ型の使用

データベースをプランニングする際、テーブル設計に関して、全体的なクエリパフォーマンスに多大な影響を与える重要な決定を行う必要があります。これらの設計上の選択により、ストレージ要件にも重大な影響があります。その結果、I/O 操作の数が減少し、クエリの処理に必要なメモリが最小限に抑えられるので、クエリパフォーマンスにも影響します。

最良のソートキーの選択

最適な分散スタイルの選択

分散スタイルとソートキーに関する悩み

- そもそも、それぞれの方式の
  メリット / デメリットが正確に理解できていない
- その場その場でなんとなく決めているので
  統一感がない
- DDL設計者によってポリシーが違う

などなど

分散スタイルとソートキーの決定方法

アジェンダ

- 分散スタイル
    - 分散スタイルはなぜ重要なのか
    - 分散キーの候補となる列の抽出
    - 分散スタイルの決定
    - 最適な分散キーの決定
- ソートキー
    - ソートキーについて
    - ソート形式の決定
    - 最良のソートキー列の決定

アジェンダ

- 分散スタイル
  - **分散スタイルはなぜ重要なのか**
  - 分散キーの候補となる列の抽出
  - 分散スタイルの決定
  - 最適な分散キーの決定
- ソートキー
  - ソートキーについて
  - ソート形式の決定
  - 最良のソートキー列の決定

分散スタイルとは何か

分散スタイルを考える身近な例: 紙での販売業務

- 1万枚の注文書（注文書1枚につき紙1枚）と
  5人の名前が書かれたお得意様がリスト（紙1枚）が
  ある
- お得意様からの注文書を10人で抽出したい

注文書とお得意様名簿を
どうやって10人に配る？

Redshiftには3つの分散スタイル

EVEN、KEY、ALL

- **EVEN分散の身近な例**

  1万枚の注文書を、上から1000枚ずつ10人に渡す

- **KEY分散の身近な例**

  1万枚の注文書を「注文者名がア行で始まる人」
  「カ行で始まる人」「サ行で始まる人」……と
  10グループに分ける

- **ALL分散の身近な例**

  お得意様名簿を10部コピーして
  1枚ずつ10人に渡す

最適な分散スタイルの例

結合の観点から考える

- EVEN分散された注文書とALL分散された名簿
    - ○ 10人がほぼ等しい時間で作業を終える

注文書の分散スタイルが最適でない例

- KEY分散された注文書とALL分散された名簿
  - △ ア行や力行担当に対してワ行担当の作業が少ない
- ALL分散された注文書とALL分散された名簿
  - × 10人が全く同じ仕事をする
  - × 注文書が10万枚にふくれあがり、紙代がかかる
    紙代 = Redshiftのストレージ

名簿の分散スタイルが最適でない例

- EVEN分散された注文書とEVEN分散された名簿

  × 結合できない = 分散しなおす必要がある（再分散）

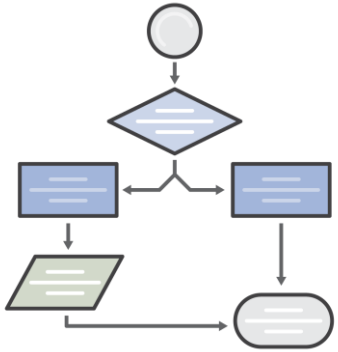
- EVEN分散された注文書とKEY分散された名簿

  × 結合できない = 分散しなおす必要がある（再分散）

現実はさらに難しい

- 様々なクエリーに対応する必要がある
- 分散キーは、1テーブルに1個しか選べない

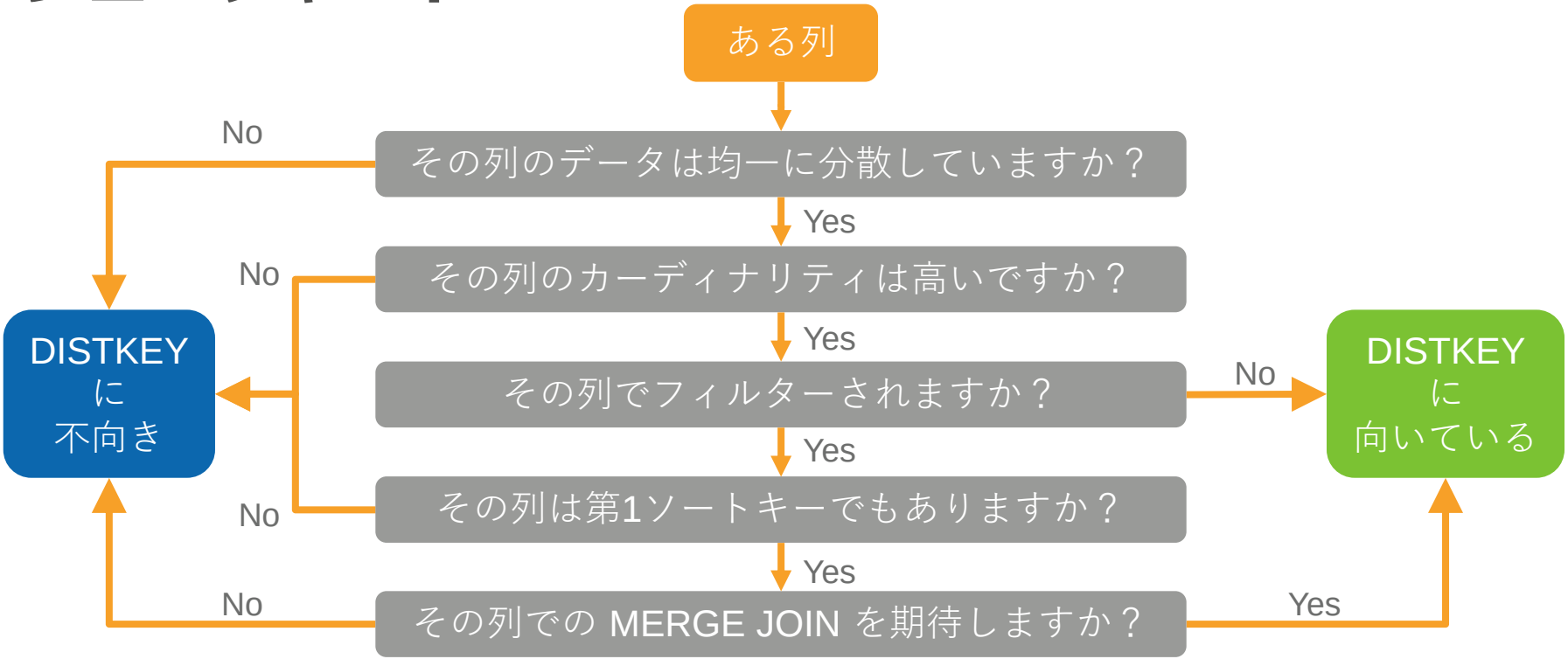
フローチャートで
機械的に判断しよう



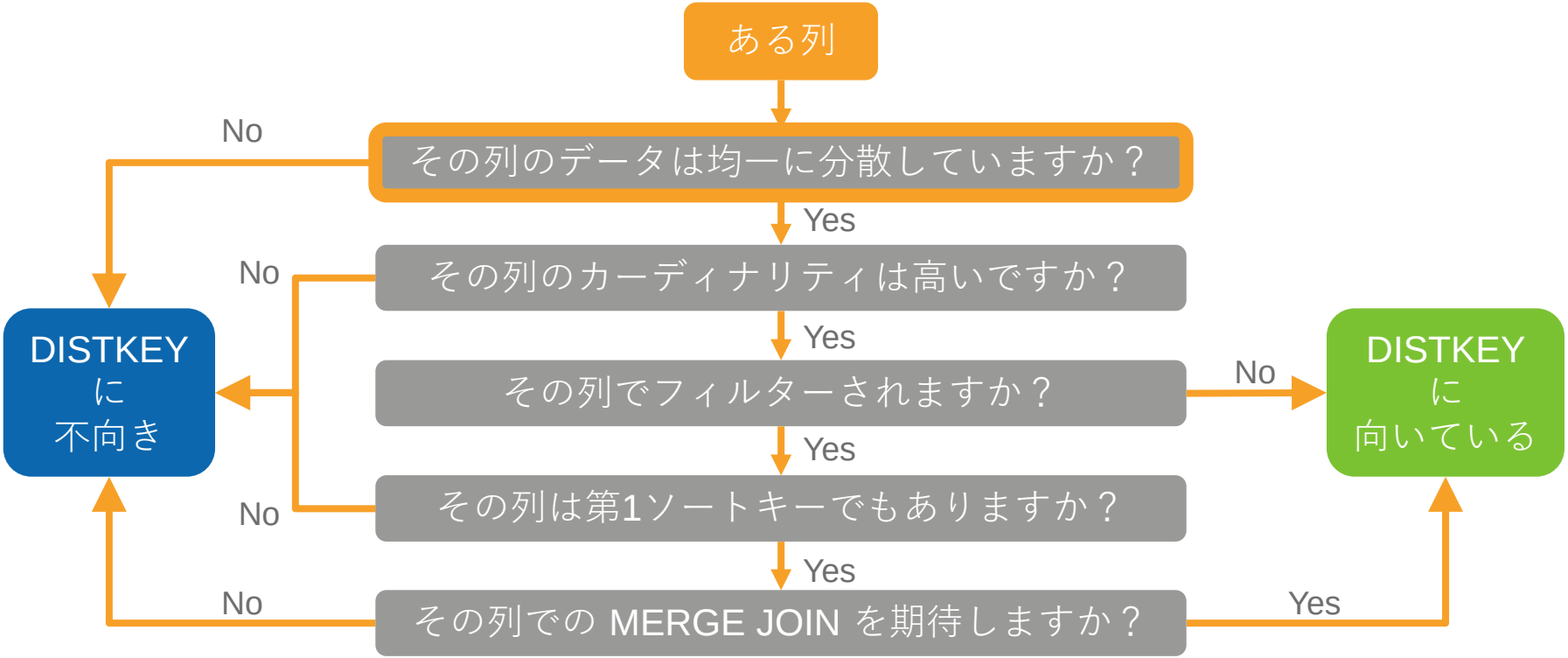
アジェンダ

- 分散スタイル
  - 分散スタイルはなぜ重要なのか
  - **分散キーの候補となる列の抽出**
  - 分散スタイルの決定
  - 最適な分散キーの決定
- ソートキー
  - ソートキーについて
  - ソート形式の決定
  - 最良のソートキー列の決定

KEY分散に適切な列かどうかを判断するフローチャート



その列のデータは均一に分散していますか？



その列のデータは均一に分散していますか？

- ア行 vs. ワ行問題

  ワ行担当のスライスがいくら速く処理を終えても
  全員の処理が終わらないとクエリーは完了しない

- NULLの割合が大きい列

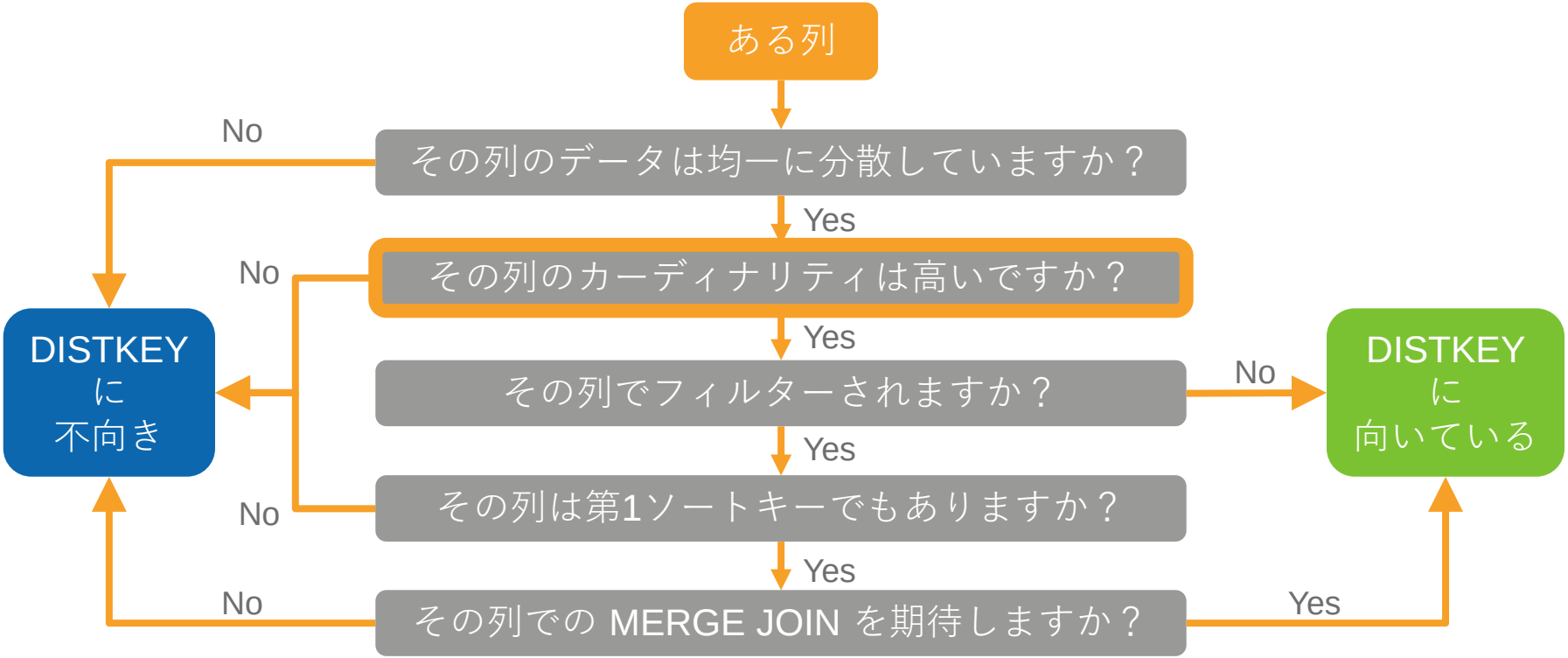
  NULL担当のスライスに仕事量が偏る

均一に分散するかの調査

```
SELECT
  col1,
  COUNT(*)
FROM lineitems
GROUP BY col1
ORDER BY 2 DESC;
```

```
          col1 |      count
---------------+-----------
        [NULL] | 124993010
     260642439 |         80
     240404513 |         80
      56095490 |         72
     348088964 |         72
     466727011 |         72
     438870661 |         72
...
```

その列のカーディナリティは高いですか？



その列のカーディナリティは高いですか？

- カーディナリティは**スライス数に対して**相対的に高い必要がある。スライス数の4から5倍以上が目安
- 576スライス（36台のds2.8xlarge) に対して300店舗の店舗ID

  300÷576=52%のスライスしかデータを持っていない
  →48%のスライスは動作しない
- 576スライスに対して577店舗の店舗ID

  ある1つのスライスだけ、
  ほかに比べて2倍のデータを持っている

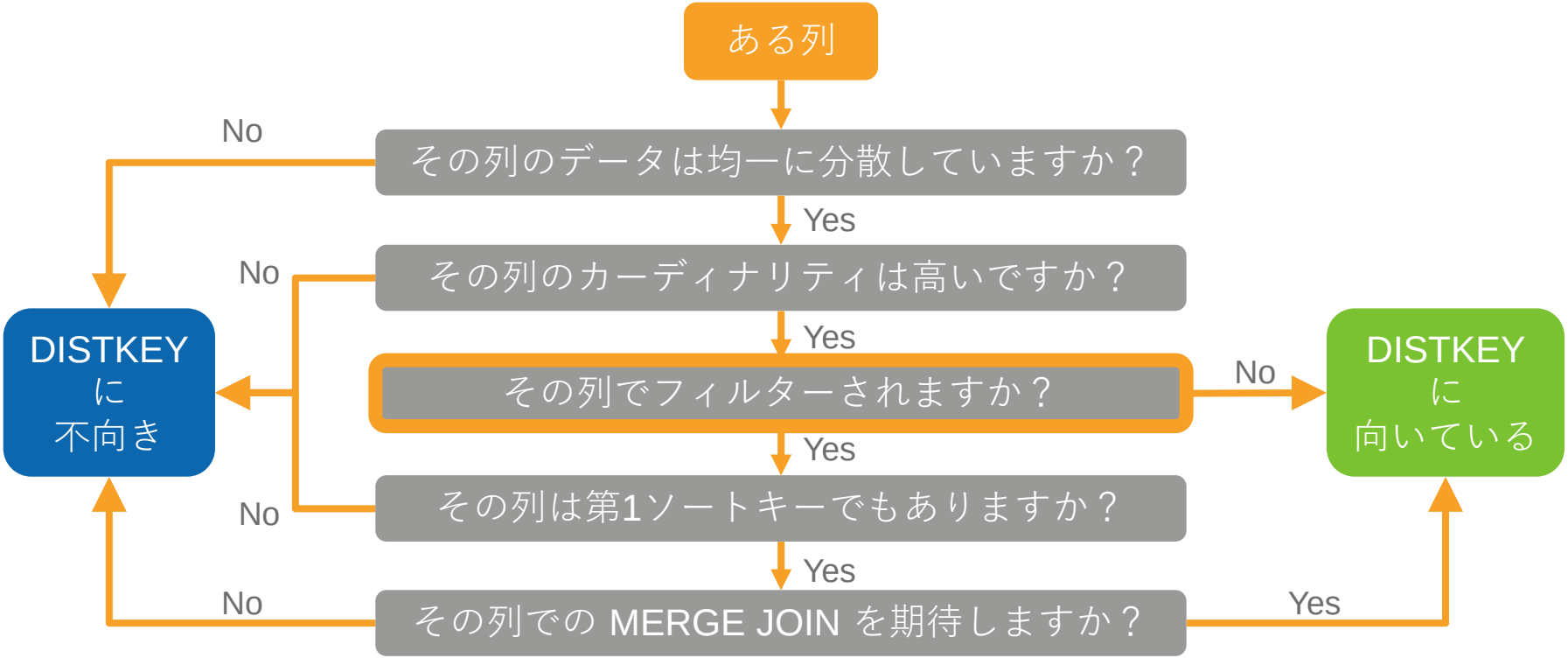
カーディナリティの調査

```
SELECT APPROXIMATE COUNT (DISTINCT sku)
FROM lineitems;
```

APPROXIMATE

- 誤差が2%程度ある代わりに、高速にクエリーを実行
- アクセスログからユニークユーザー数を数えるとき
  などにも便利

その列でフィルターされますか？



その列でフィルターされますか？

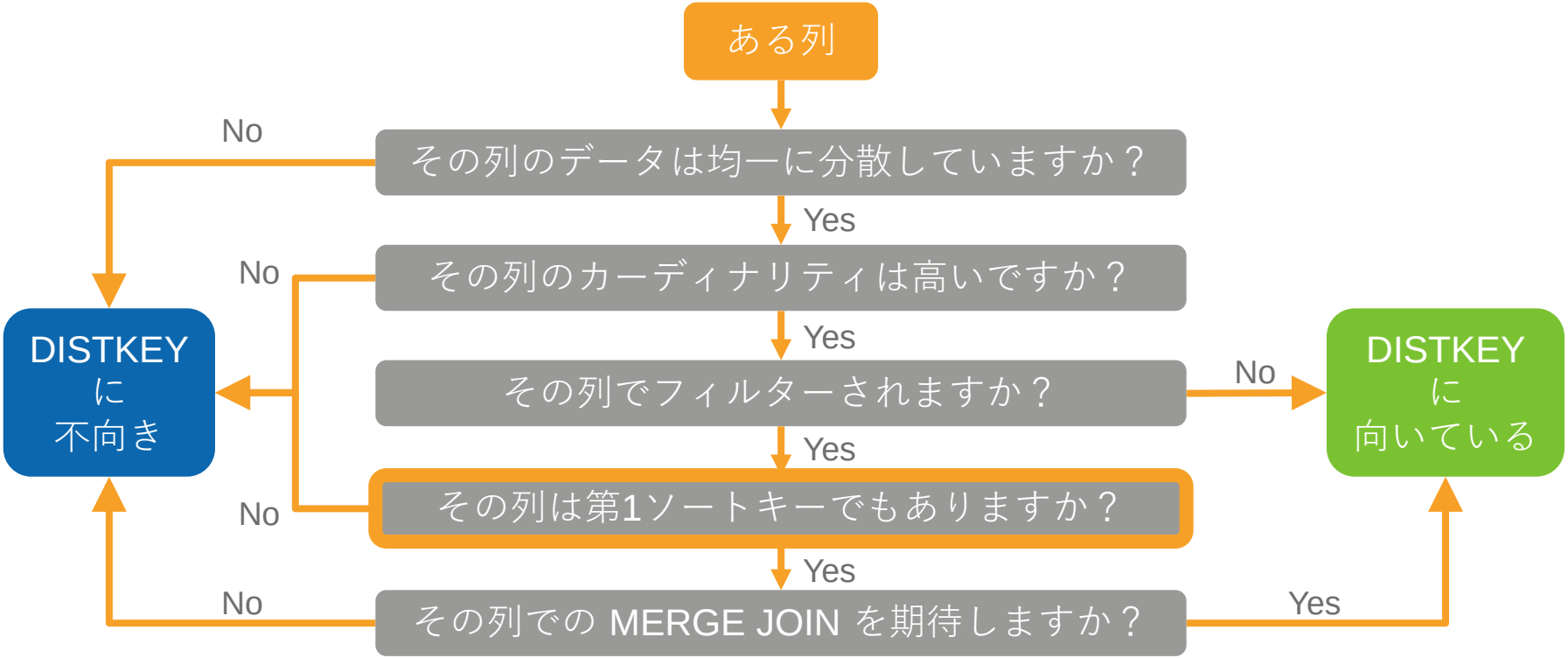
- 576スライスに対して3000店舗だと
  十分なカーディナリティがあるように思える

- しかし、クエリーに必ず WHERE 店舗ID = n
  という条件フィルターがついていたら？
  → 常に1つのスライスしか使われない

どの列でフィルターされているかの調査

```
SELECT
  ti."table", ti.diststyle, RTRIM(a.attname) column_name,
  COUNT(DISTINCT s.query ||'-'|| s.segment ||'-'|| s.step) as num_scans,
  COUNT(DISTINCT CASE
    WHEN TRANSLATE(TRANSLATE(info,')',' '),'(',' ') LIKE ('%'|| a.attname ||'%')
      THEN s.query ||'-'|| s.segment ||'-'|| s.step END) AS column_filters
FROM stl_explain p
JOIN stl_plan_info i ON (i.userid = p.userid AND i.query = p.query AND
  i.nodeid=p.nodeid )
JOIN stl_scan s ON (s.userid = i.userid AND s.query = i.query AND
  s.segment = i.segment AND s.step = i.step)
JOIN svv_table_info ti ON ti.table_id = s.tbl
JOIN pg_attribute a ON (a.attrelid = s.tbl AND a.attnum > 0)
WHERE s.tbl IN ([table_id])
GROUP BY 1, 2, 3, a.attnum
ORDER BY attnum;
```

その列は第1ソートキーですか？



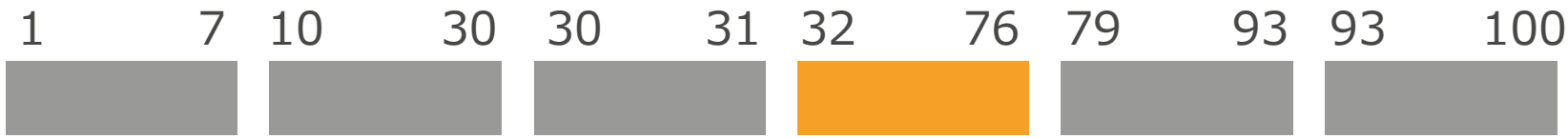
その列は第1ソートキーですか？

- フィルターされる列はソートキーにも使用されている
  ことが少なくない
- ソートキーにも使用されている場合
  （複合ソートキーのときは第1キーの場合）
  ゾーンマップが効く可能性がある
- ゾーンマップが効いた場合、スキャン速度の大幅向上が
  処理スライス数の少なさを補う可能性がある

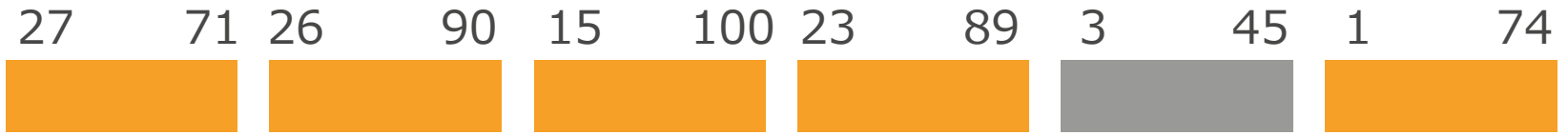
ゾーンマップとは

1MB単位のデータブロック内の最小値と最大値を
メタデータとして格納

ソート済みのcol1から50を見つける

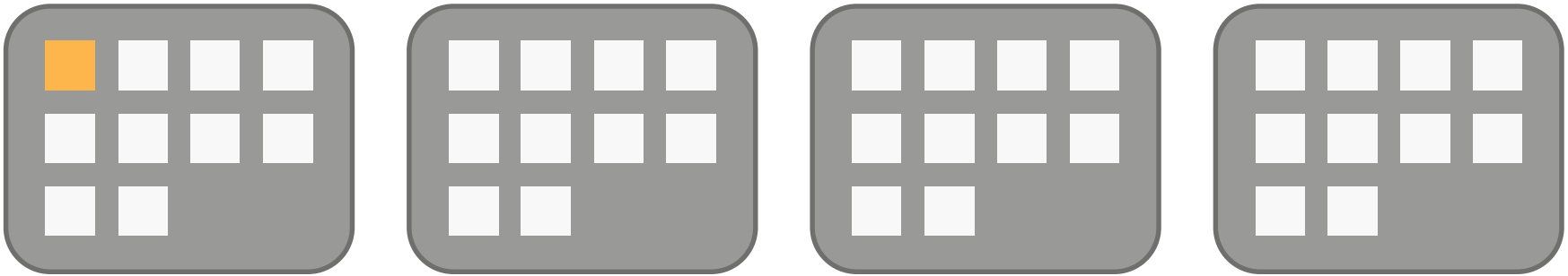


未ソートのcol1から50を見つける



ゾーンマップがスライス数を補う例

4つのスライスに対して40店舗をKEY分散させた状態で
WHERE 店舗ID ＝ 1 する

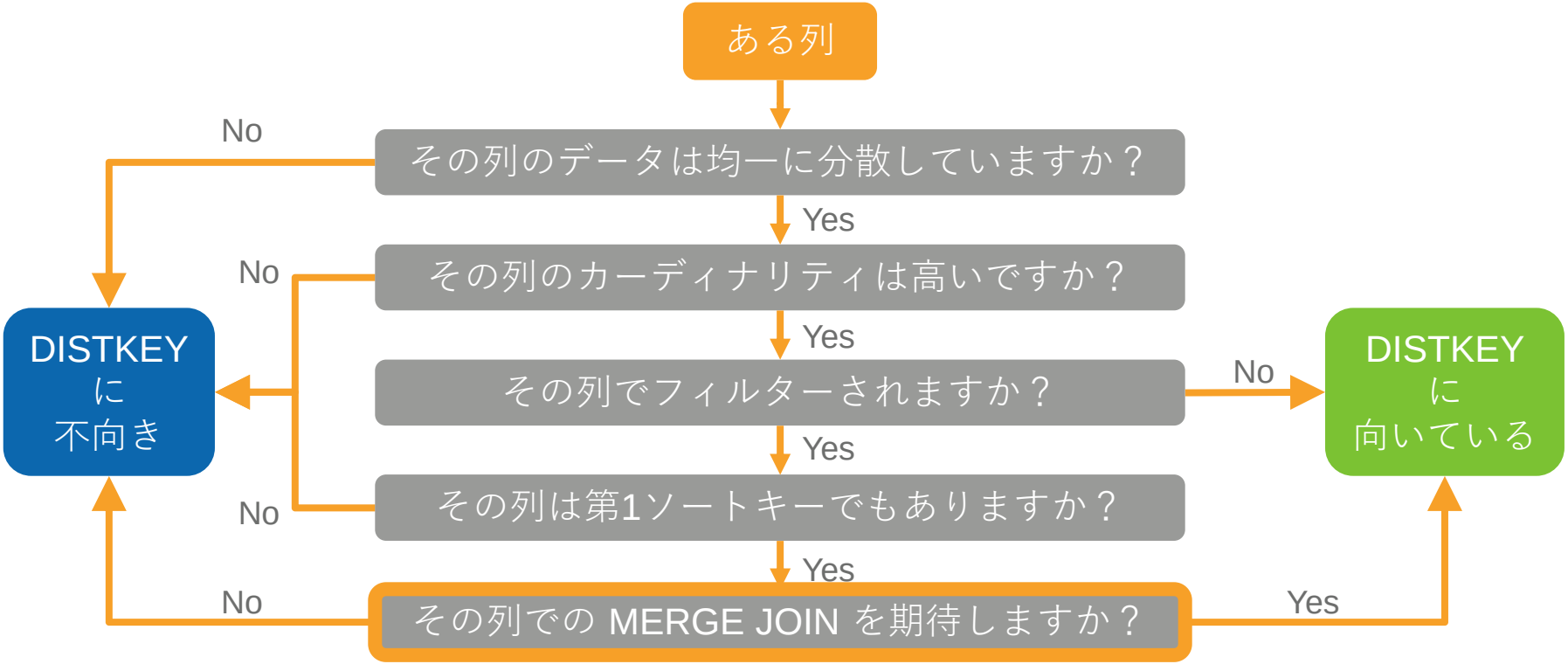


CPUリソース（スライス数）は4分の1だが、
ゾーンマップによってIO量は10分の1に

第1ソートキーの調査

```
SELECT attname
FROM pg_attribute
WHERE attrelid = [table_id]
  AND attsortkeyord = 1;
```

その列での MERGE JOIN を期待しますか？



その列での MERGE JOIN を期待しますか？

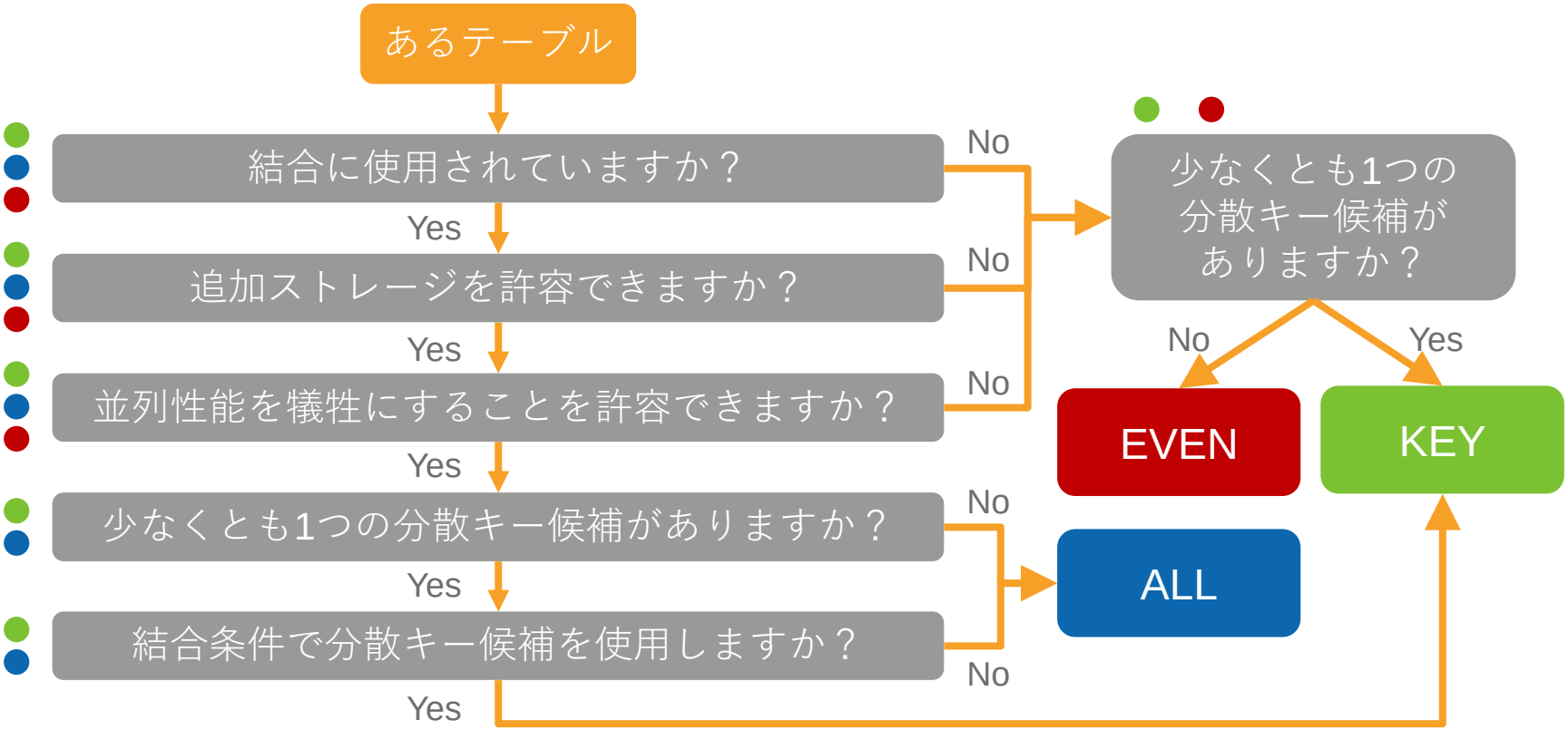
以下の条件をすべて満たすとき、
最も高速な結合操作である MERGE JOIN が行われる

- 2つのテーブルで同じソートキーが指定され、
  同じ列で分散されている
- どちらのテーブルも80%以上ソートされている
- **2つのテーブルがJOIN条件でDISTKEY列と
  SORTKEY列の両方を使用して結合されている**

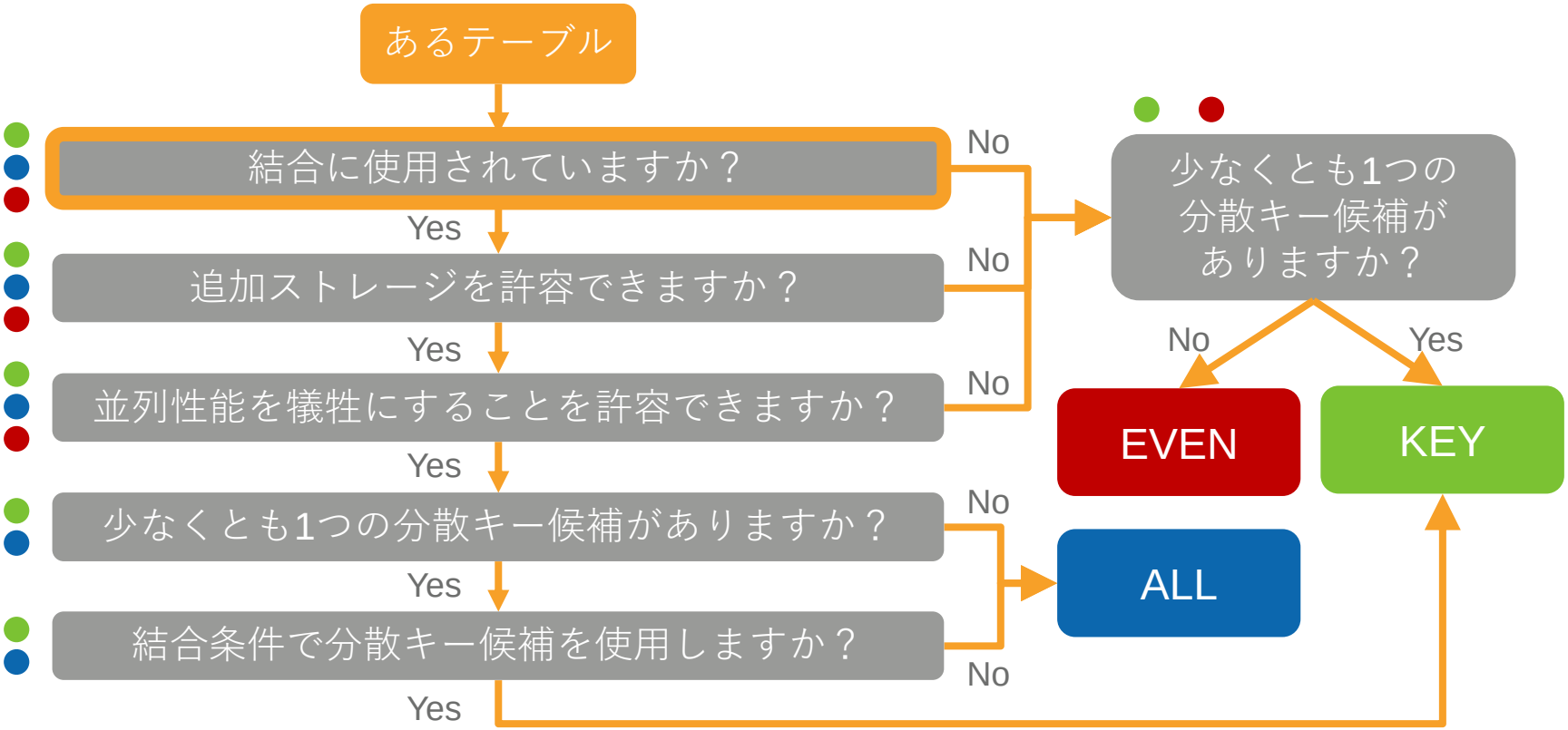
アジェンダ

- 分散スタイル
  - 分散スタイルはなぜ重要なのか
  - 分散キーの候補となる列の抽出
  - **分散スタイルの決定**
  - 最適な分散キーの決定
- ソートキー
  - ソートキーについて
  - ソート形式の決定
  - 最良のソートキー列の決定

分散スタイルを決定するフローチャート



そのテーブルは結合に使用されていますか？



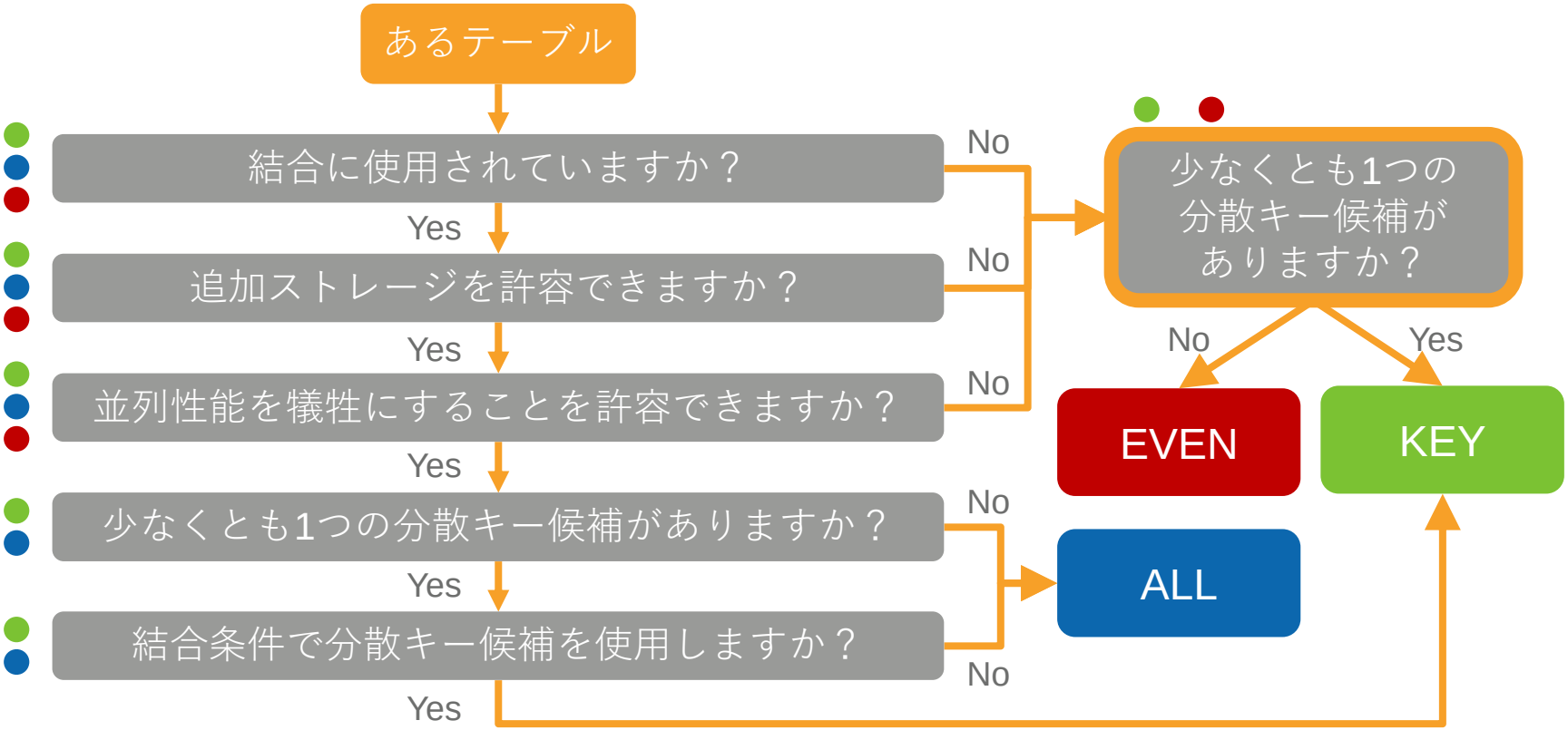
そのテーブルは結合に使用されていますか？

- 結合に使用されていない場合、
  ALL分散の重複コストに対してメリットが何もない
  → ALL分散が選択肢からなくなる
- 結合はJOIN句だけでなく、IN、NOT IN、MINUS、
  EXCEPT、INTERSECT、EXISTSなどでも
  使われるので注意

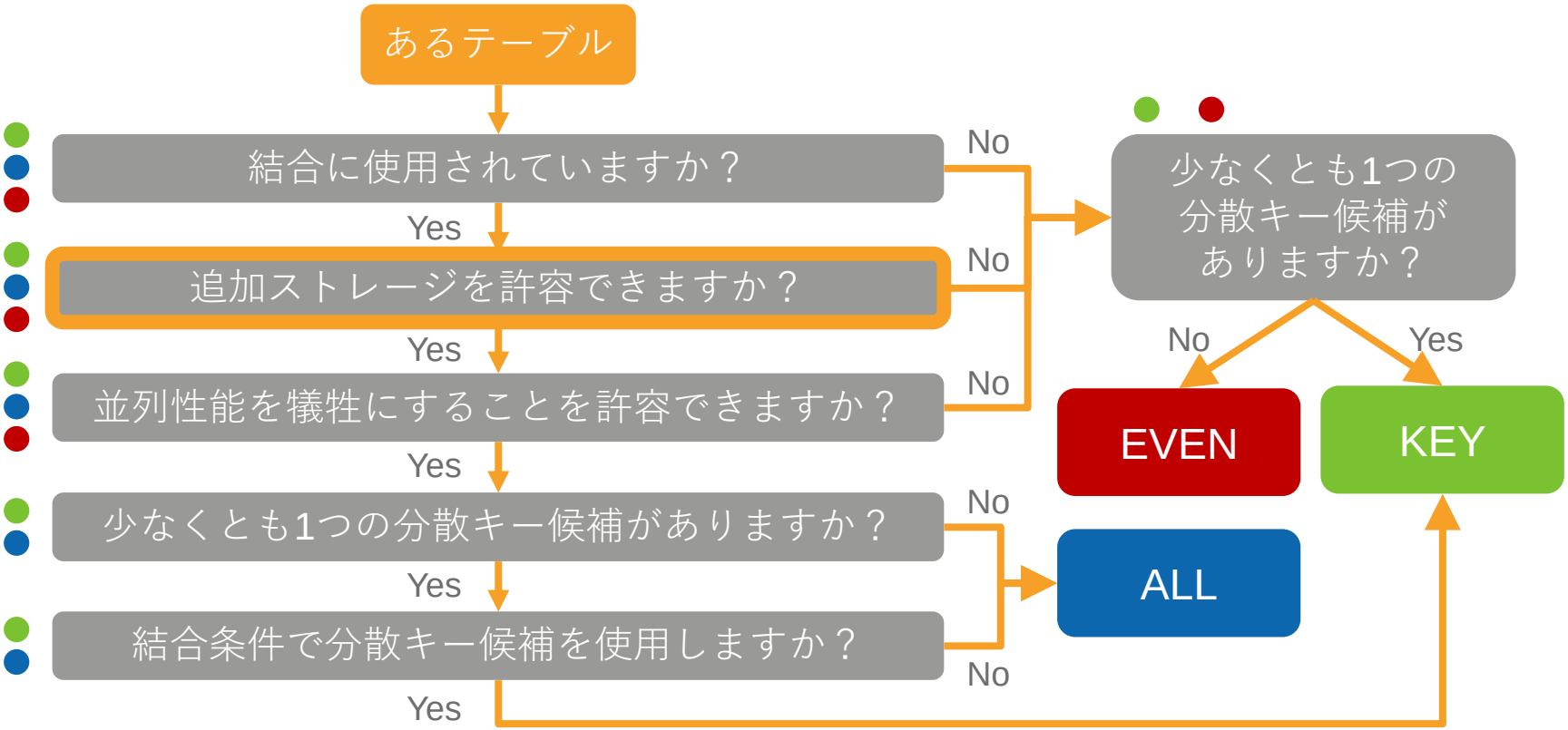
結合に使われたかどうかの調査

```
SELECT COUNT(*) FROM (
  SELECT DISTINCT query
  FROM stl_scan
  WHERE tbl = [table_id]
    AND type = 2
    AND userid > 1
  INTERSECT (
    SELECT DISTINCT query FROM stl_hashjoin
    UNION
    SELECT DISTINCT query FROM stl_nestloop
    UNION
    SELECT DISTINCT query FROM stl_mergejoin
));
```

少なくとも1つの分散キー候補がありますか？



追加ストレージを許容できますか？



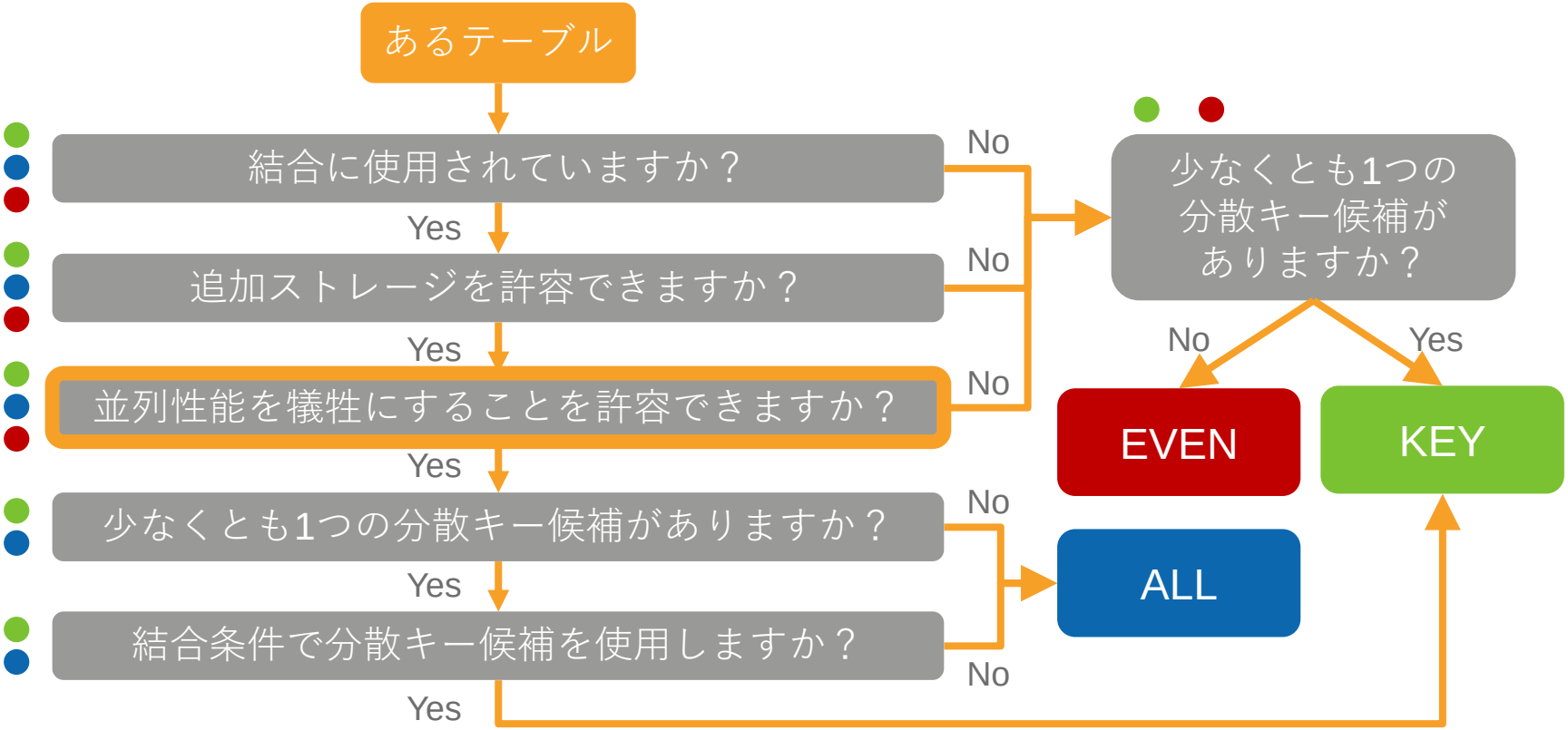
追加ストレージを許容できますか？

- ALL分散の場合、そのテーブルはすべてのノードにコピーされる
  →容量が×ノード数になる
- クラスターの空き容量が不足した場合、
  ノード追加をしても、ALL分散しているテーブル分は
  空き容量が増えない点に注意

ALL分散した場合の容量見積もり

```
SELECT "table", size, pct_used,
  CASE diststyle
    WHEN 'ALL' THEN size::TEXT
    ELSE '< ' || size * (
      SELECT COUNT(DISTINCT node) FROM stv_slices)
  END est_distall_size,
  CASE diststyle
    WHEN 'ALL' THEN pct_used::TEXT
    ELSE '< ' || pct_used * (
      SELECT COUNT(DISTINCT node) FROM stv_slices)
  END est_distall_pct_used
FROM svv_table_info
WHERE table_id = [table_id];
```

並列性能を犠牲にすることを許容できますか？



並列性能を犠牲にすることを許容できますか？

- ALL分散は同じデータがすべてのノードに
  コピーされているため、並列性能を犠牲にしてしまう
- ある1行を更新する場合
  - KEYまたはEVEN: 特定の1ノードのみで更新
  - ALL: 全ノードで更新
- 結合に使用される場合
  - メリット: ネットワークIOが減る
  - デメリット: 計算量とディスクIOが増える

ALL分散が不向きな一般的なガイドライン

- 読み取り操作
  - 大きなファクトテーブルに対するスキャン
  - 結合しない単一テーブルスキャン
  - 複雑な集計のあるテーブルへのスキャン
    （たとえばウインドウ集約関数）
- 書き込み操作
  - DML文で頻繁に変更されるテーブル
  - 膨大なデータをロードするテーブル
  - VACUUM操作で頻繁にメンテナンスする
    必要のあるテーブル

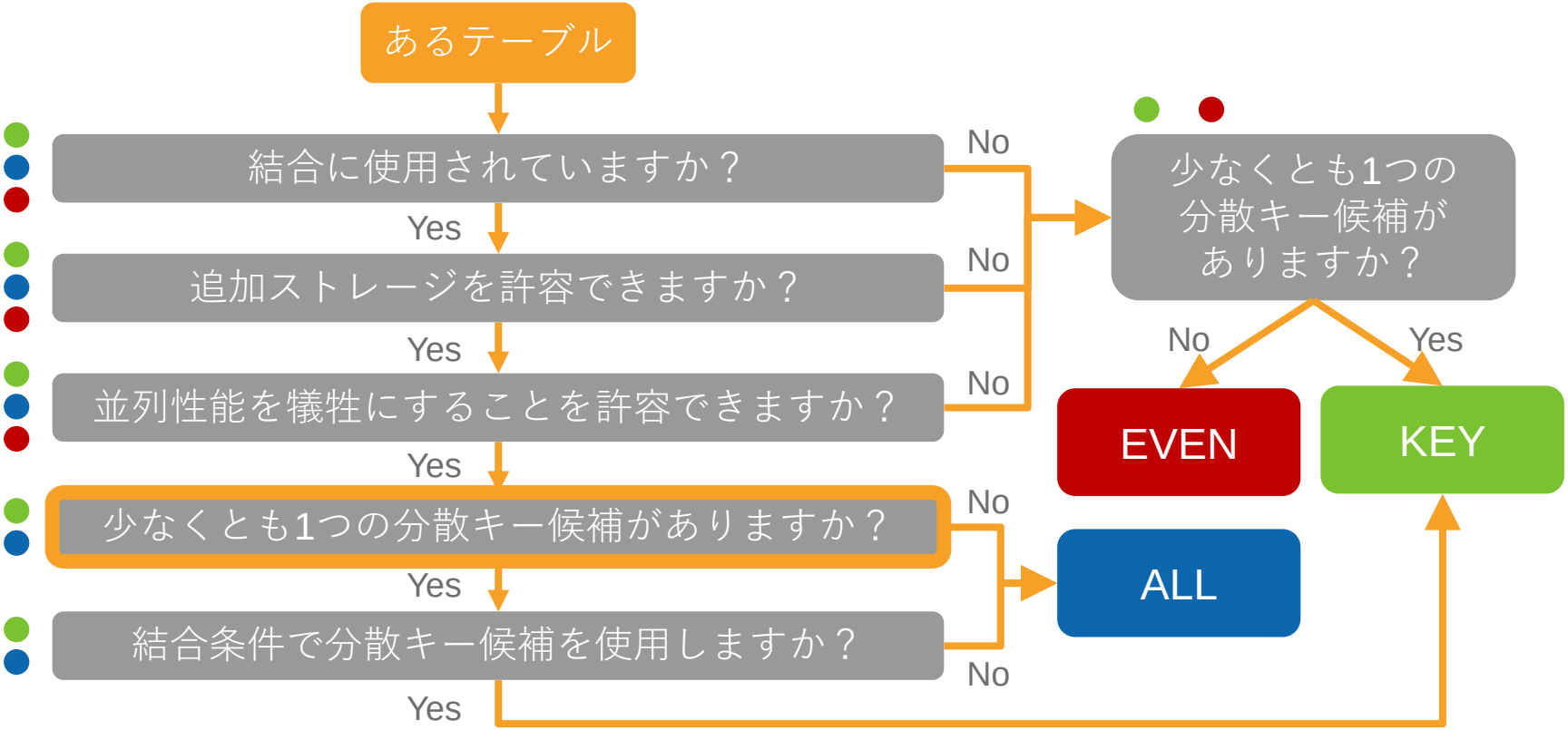
書き込まれているかの調査

```
SELECT '[table_id]' AS “table_id”,
  (SELECT count(*) FROM (
    SELECT DISTINCT query FROM stl_insert
      WHERE tbl = [table_id]
    INTERSECT
    SELECT DISTINCT query FROM stl_delete
      WHERE tbl = [table_id])
  ) AS num_updates,
  (SELECT count(*) FROM (
    SELECT DISTINCT query FROM stl_delete
      WHERE tbl = [table_id]
    MINUS
    SELECT DISTINCT query FROM stl_insert
      WHERE tbl = [table_id])
  ) AS num_deletes,
```

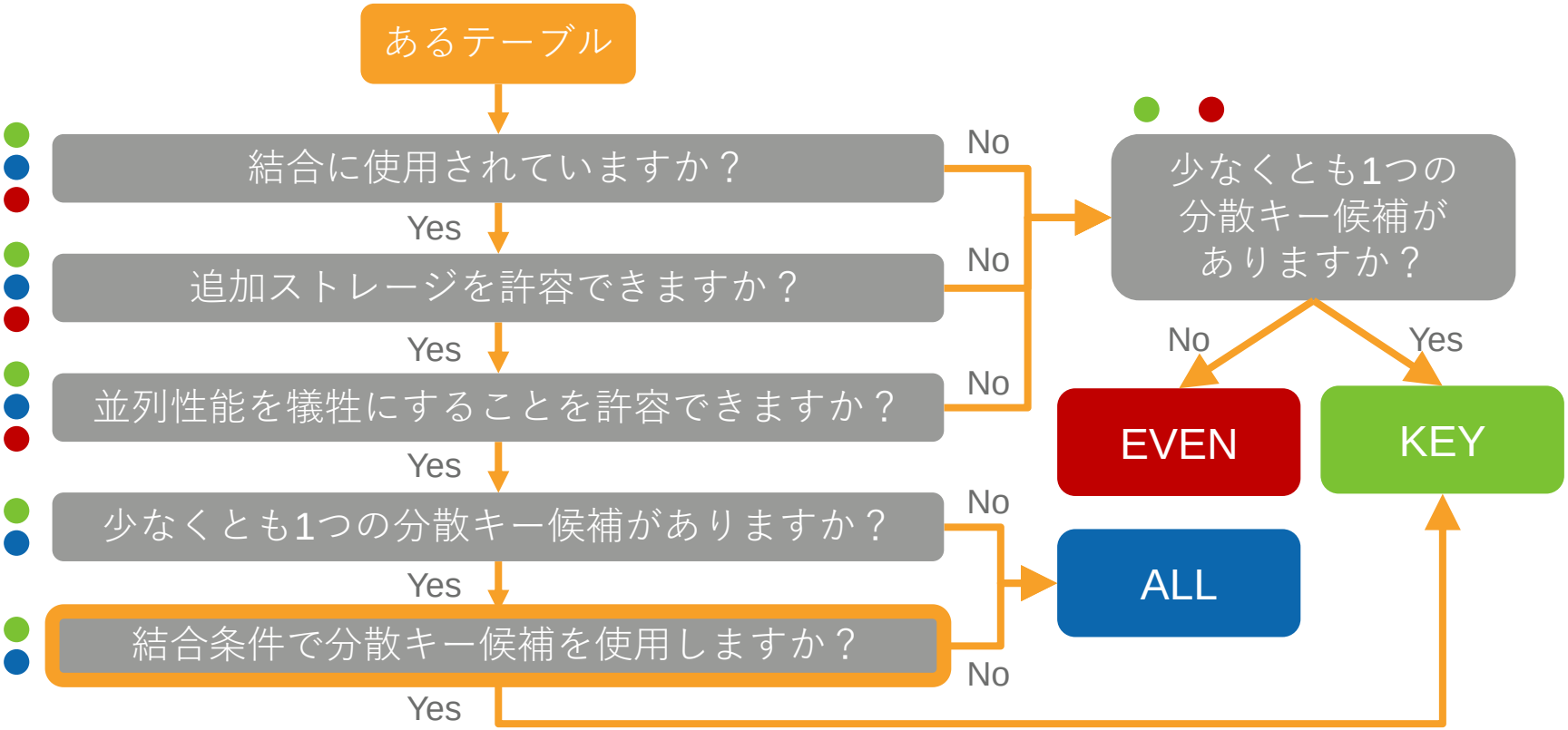
（右へ続く）

```
(SELECT COUNT(*) FROM (
  SELECT DISTINCT query FROM stl_insert
    WHERE tbl = [table_id]
  MINUS
  SELECT distinct query FROM stl_s3client
  MINUS
  SELECT DISTINCT query FROM stl_delete
    WHERE tbl = [table_id])
) AS num_inserts,
(SELECT COUNT(*) FROM (
  SELECT DISTINCT query FROM stl_insert
    WHERE tbl = [table_id]
  INTERSECT
  SELECT distinct query FROM stl_s3client)
) as num_copies,
(SELECT COUNT(*) FROM (
   SELECT DISTINCT xid FROM stl_vacuum
   WHERE table_id = [table_id]
   AND status NOT LIKE 'Skipped%')
) AS num_vacuum;
```

少なくとも1つの分散キー候補がありますか？



結合条件で分散キー候補を使用しますか？



アジェンダ

- 分散スタイル
  - 分散スタイルはなぜ重要なのか
  - 分散キーの候補となる列の抽出
  - 分散スタイルの決定
  - **最適な分散キーの決定**
- ソートキー
  - ソートキーについて
  - ソート形式の決定
  - 最良のソートキー列の決定

どのクエリーを優先すべきか

- クエリーごとで結合条件に使用する列が異なる場合、どのクエリーを優先すべきかから分散キーを選ぶ
  - バッチとレポーティングはどちらを優先？
  - 定型レポートのSLAと
    インタラクティブなクエリーのユーザー体験
  - 重要度は低いが1日に何千回も結合される列と
    重要度は高いが1日に10回しか結合されない列
  - 1日に1000回実行される5秒のクエリーを2秒に
    改善する分散キーと、1日に2回しか実行されない
    60分のクエリーを20分に改善する分散キー

アジェンダ

- 分散スタイル
  - 分散スタイルはなぜ重要なのか
  - 分散キーの候補となる列の抽出
  - 分散スタイルの決定
  - 最適な分散キーの決定
- ソートキー
  - **ソートキーについて**
  - ソート形式の決定
  - 最良のソートキー列の決定

ソートするメリット

- ゾーンマップによりディスクIOを削減する
- クエリー実行時のソート処理をなくす
- MERGE JOIN によって
  結合のパフォーマンスを向上する

ソート形式の種類

- COMPOUND

  定義した順が重要で、
  第1ソートキー（先頭に定義した列）が使用されない場合は、
  第2ソートキー以降も使われない

- INTERLEAVED

  定義した列すべてが同じ重要度。
  第1ソートキーを使う場合はCOMPOUNDより少し遅いが、
  使わない場合は圧倒的に速い。
  ただし、**メンテナンスコストが非常に高い**

一般的に、90%のケースはCOMPOUDで十分

ソート形式の具体例

COMPOUND

c1	c2
1	A
1	B
1	C
1	D
2	A
2	B
2	C
2	D

c1	c2
3	A
3	B
3	C
3	D
4	A
4	B
4	C
4	D

c1 = 1 → 1ブロック
c2 = C → 4ブロック

INTERLEAVED

c1	c2
1	A
1	B
2	A
2	B
1	C
1	D
2	C
2	D

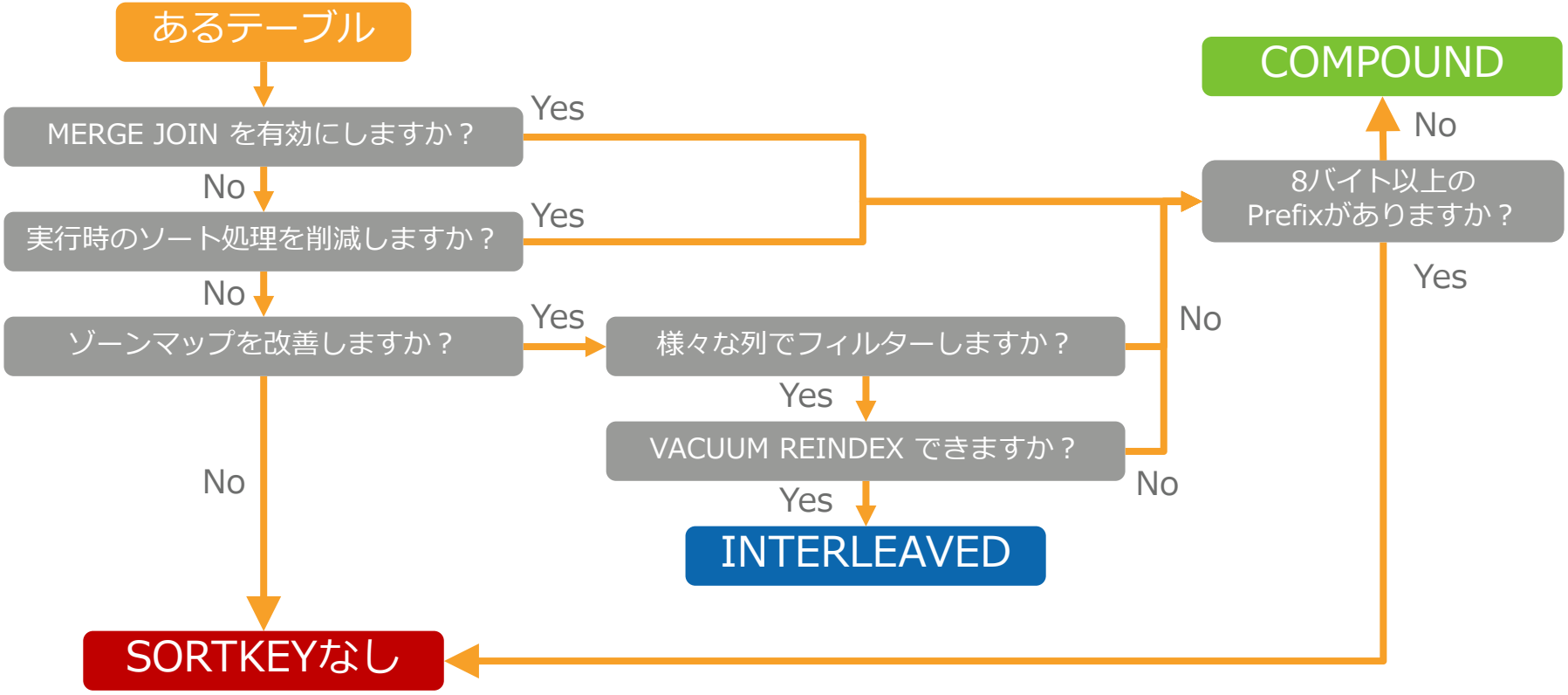
c1	c2
3	A
3	B
4	A
4	B
3	C
3	D
4	C
4	D

c1 = 1 → 2ブロック
c2 = C → 2ブロック

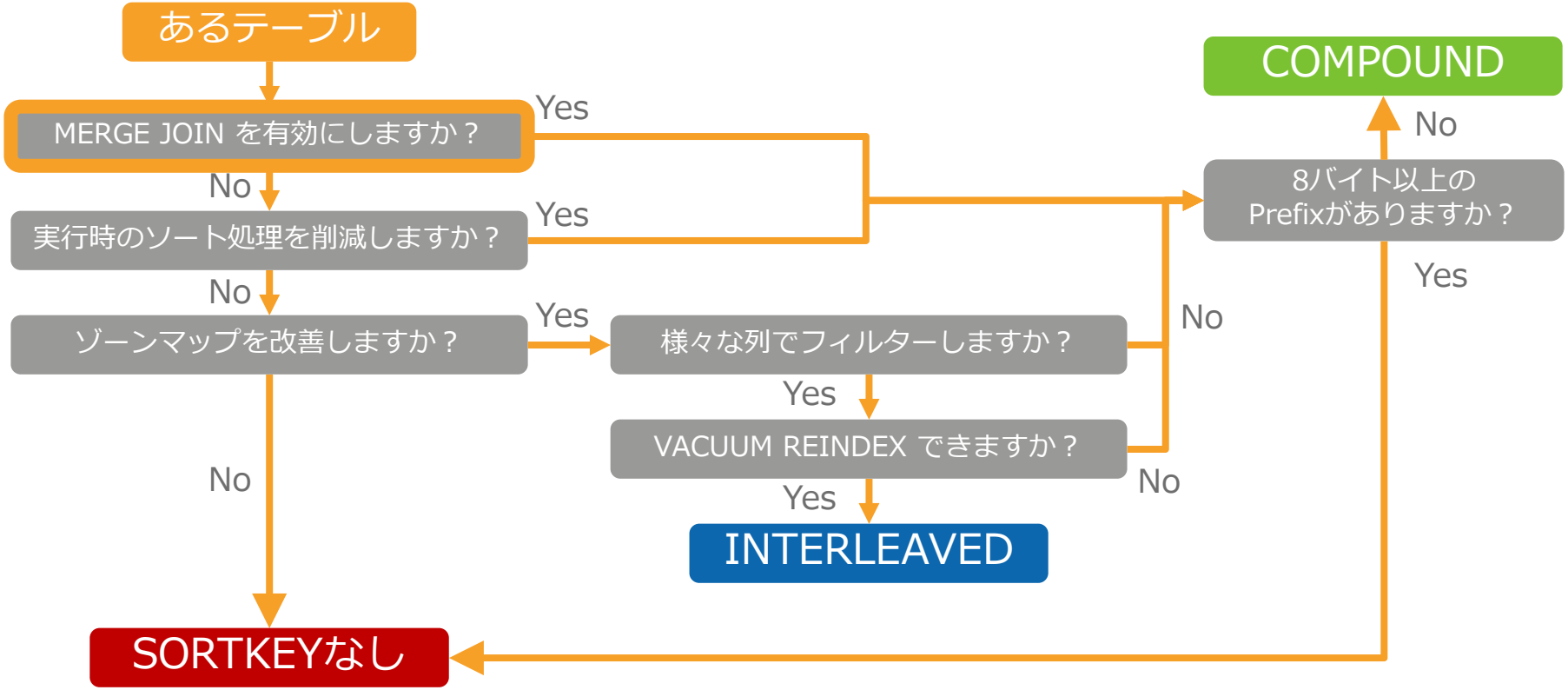
アジェンダ

- 分散スタイル
  - 分散スタイルはなぜ重要なのか
  - 分散キーの候補となる列の抽出
  - 分散スタイルの決定
  - 最適な分散キーの決定
- ソートキー
  - ソートキーについて
  - **ソート形式の決定**
  - 最良のソートキー列の決定

ソートキーを決定するフローチャート その1



ソートは MERGE JOIN を有効にしますか？

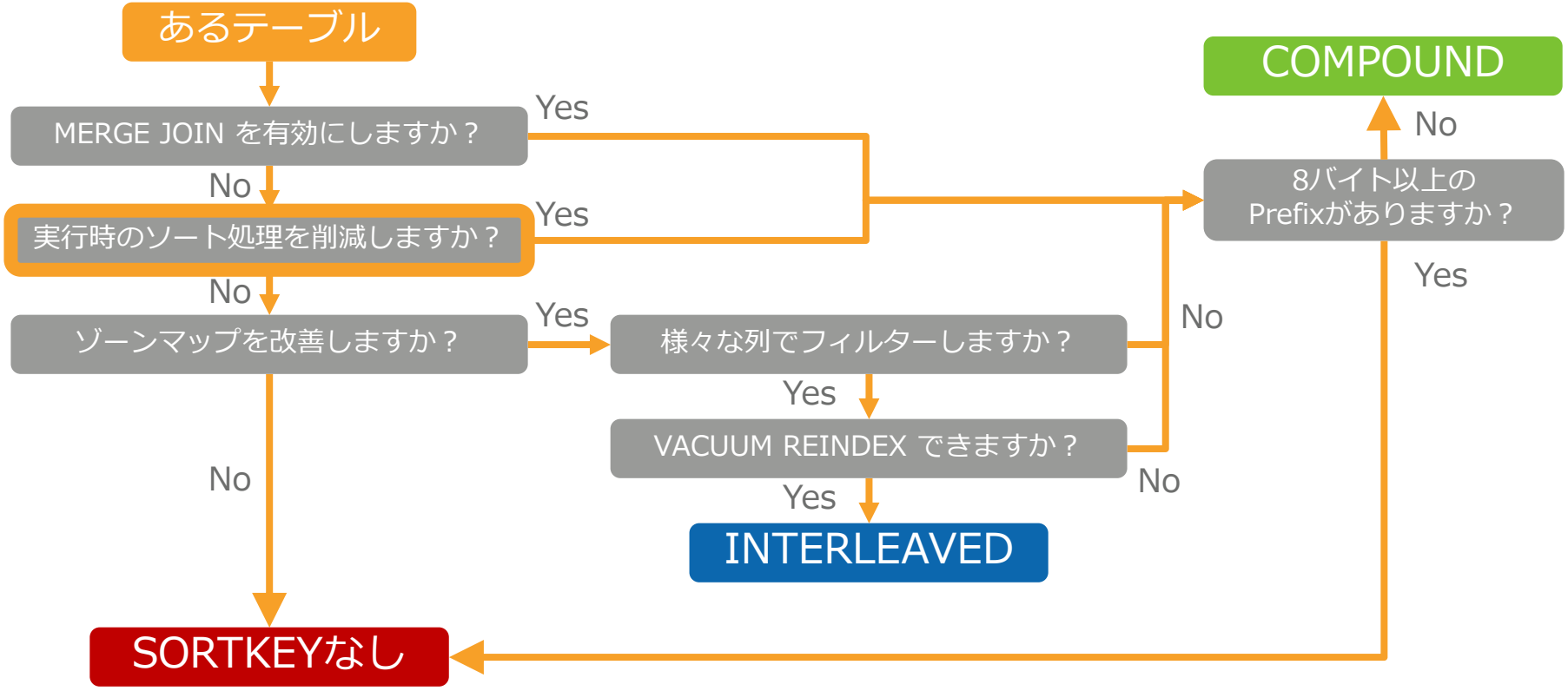


ソートは MERGE JOIN を有効にしますか？

以下の条件をすべて満たすとき、
最も高速な結合操作である MERGE JOIN が行われる

- **2つのテーブルで同じソートキーが指定され、同じ列で分散されている**
- どちらのテーブルも80%以上ソートされている
- 2つのテーブルがJOIN条件でDISTKEY列とSORTKEY列の両方を使用して結合されている

ソートは実行時のソート処理を削減しますか？

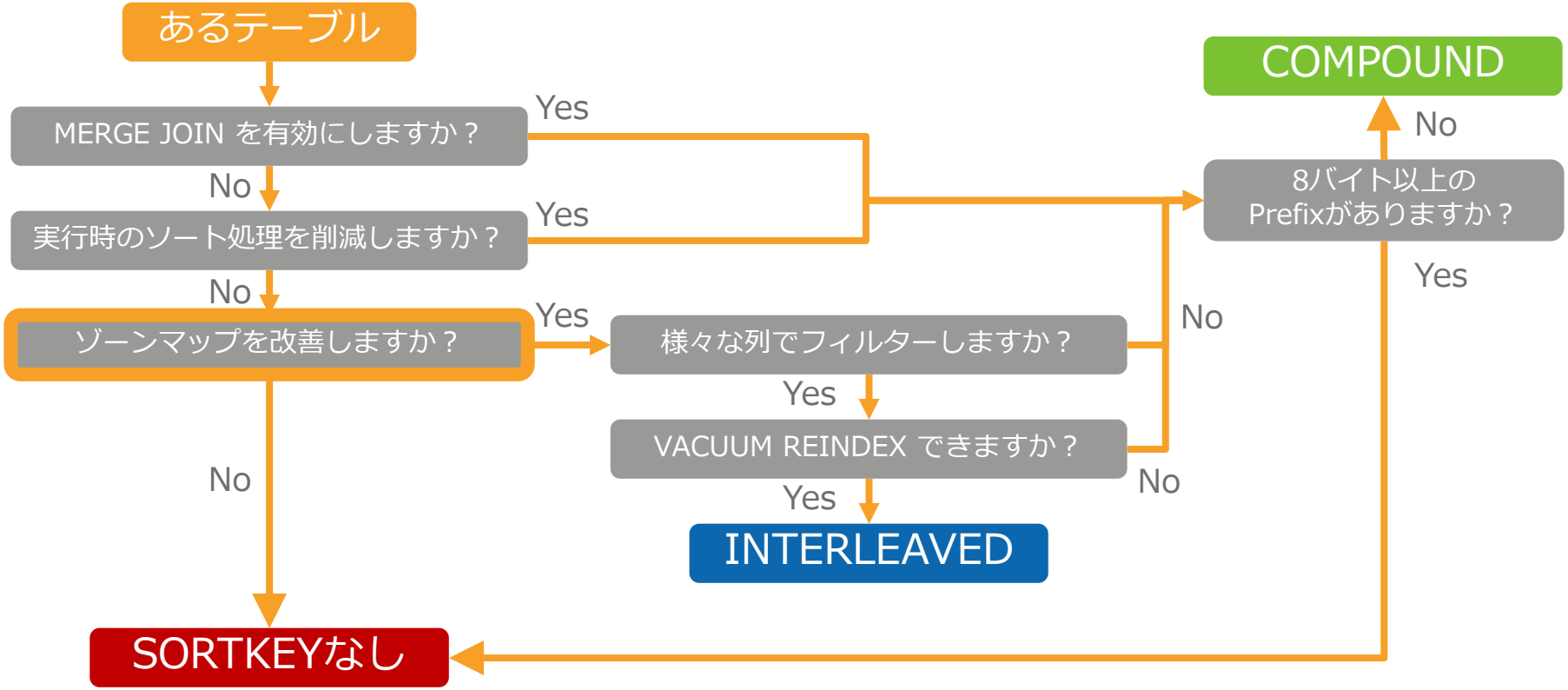


ソートは実行時のソート処理を削減しますか？

- ORDER BY、GROUP BY
  WINDOW関数内の PARTITION BY、ORDER BY は
  ソート処理が行われる

- 上記で指定される列を事前ソートしておけば
  クエリー時のソート処理を減らせる

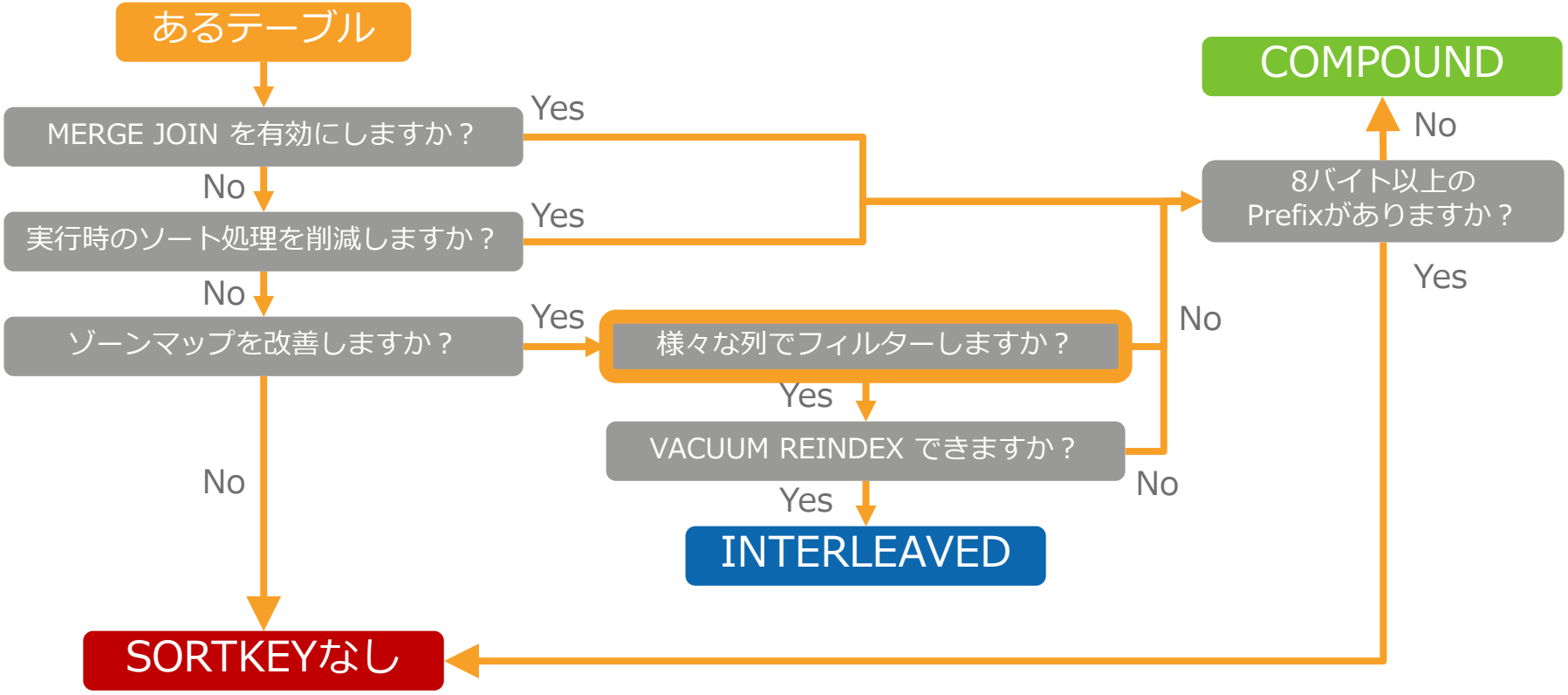
ソートはゾーンマップを改善できますか？



ゾーンマップを改善できないケース

- 各スライスに1MBのブロックが1つしかない場合
- 列に1つの値のみが含まれている場合

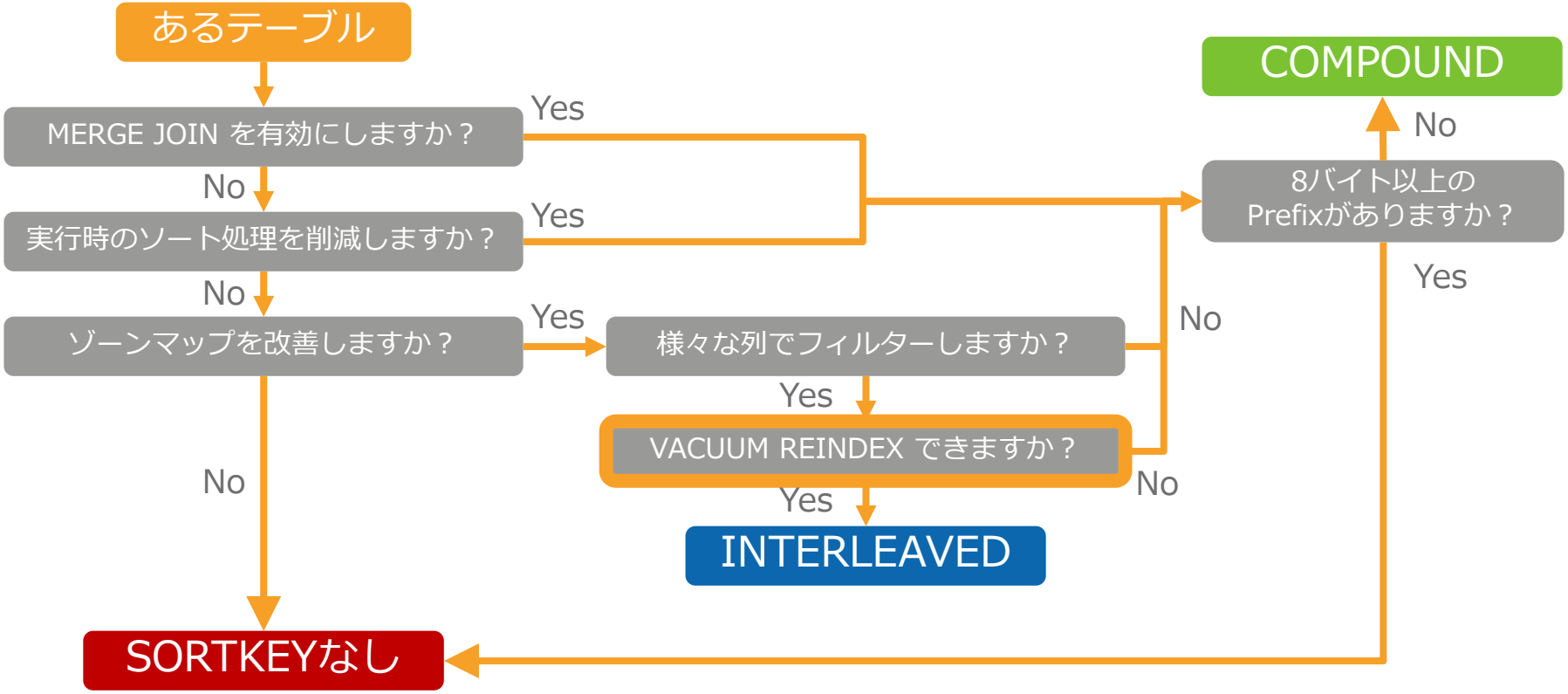
クエリーは様々な列でフィルターしますか？



クエリーは様々な列でフィルターしますか？

- 各テーブルにはソートキーは1つしか指定できない
- クエリーごとでファイルターに使用する列が異なる場合、
  どのクエリーを優先すべきかからソートキーを
  選ぶべきだが、甲乙つけがたいときもある

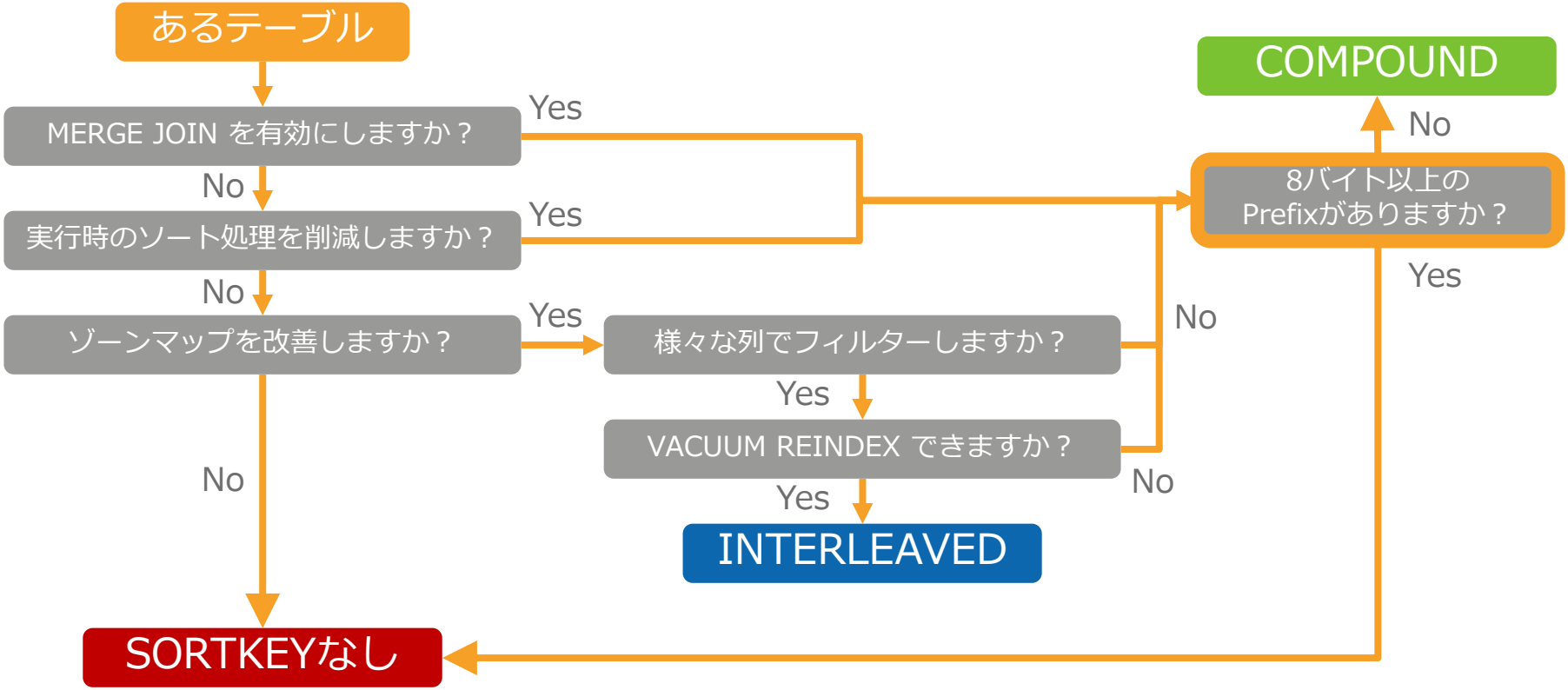
必要に応じて VACUUM REINDEX できますか？



必要に応じて VACUUM REINDEX できますか？

- COMPOUD SORTKEY はソート済みのデータをロードする場合はVACUUMが不要
    - タイムスタンプ列が COMPOUND SORTKEY でタイムスタンプ順にロードされるときなど
- INTERLEAVED SORTKEY はロード後に VACUUM REINDEX しないと効果が弱まる
- VACUUM REINDEX はIOコストが非常に高い

データに8バイト以上のPrefixがありますか？



データに8バイト以上のPrefixがありますか？

- COMPOUND SORTKEY は
  データの先頭8バイトまでしかソート順に考慮しない
    - http://www や https:// で始まることが多い
      URL列は COMPOUND SORTKEY に使用できない
- INTERLEAVED SORTKEY は
  データ全体をソート順に考慮する
- ただし、一般的にURL列などの長い文字列は
  結合条件や前方一致フィルター条件にならない

アジェンダ

- 分散スタイル
  - 分散スタイルはなぜ重要なのか
  - 分散キーの候補となる列の抽出
  - 分散スタイルの決定
  - 最適な分散キーの決定
- ソートキー
  - ソートキーについて
  - ソート形式の決定
  - **最良のソートキー列の決定**

最良のソートキー列の決定

- MERGE JOIN のためのソートキー
  → DISTKEYと同じ列

- ソート処理を削減するためのソートキー
  → ORDER BY などで使われる列

- ゾーンマップを改善するためのソートキー
  → フィルターされる列

まとめ

- 分散スタイルの決定方法
    1. KEY分散に向く列が存在するか検討する
    2. ALL分散に向くか検討する
    - 分散スタイルは結合に注目
- ソートキーの決定方法
    1. 何のためにソートするのか検討する
    2. ゾーンマップを改善するための場合は
       INTERLEAVEに向くか検討する
    - ソートキーは結合、ORDER BY、フィルターに注目

AWS

SUMMIT

Thank You!

アマゾン ウェブ サービス ジャパン株式会社
柴田竜典 | シバタツ

amazon web services

本セッションのFeedbackをお願いします

受付でお配りしたアンケートに本セッションの満足度やご感想などをご記入ください
アンケートをご提出いただきました方には、もれなく素敵なAWSオリジナルグッズを
プレゼントさせていただきます

アンケートは受付、パミール3FのEXPO展示会場内にて回収させて頂きます

AWSソリューションDay 2017: Database Day

-すでに始まっている！「クラウドへのデータベース移行」と「データレイクを軸としたビッグデータ活用」-

- Database Day とは？

  ユーザー企業 / パートナー / AWSによる導入事例や活用動向また技術情報を
  ご紹介するIT部門（エンジニア / 管理者など）向けのカンファレンス
- 開催日時 / 会場
  - 2017年7月5日(水) 10:00-17:30 （9:30開場予定）
  - 大崎ブライトコアホール（JR大崎駅より徒歩5分）
- セッション
  - 基調講演 + 2トラック構成ブレイクアウトセッション
    - トラック1: データベース移行
    - トラック2: データレイク
- お申込み

  https://aws.amazon.com/jp/about-aws/events/2017/solutiondays20170705/