ZOZOTOWNのDWHをRedshiftからBigQueryにお引越しした話

株式会社ZOZOテクノロジーズ
マーケティングオートメーションチーム
塩崎健弘

ZOZO Technologies

自己紹介

ZOZOテクノロジーズ
マーケティングオートメーションチーム

塩崎 健弘

新卒でVASILYに入社後、M&Aを経てZOZOテクノロジーズへ。昔はRailsでweb　APIを作ったり、検索サーバーのパフォーマンスチューニングをやったり。最近のお仕事はデータ基盤の整理とか、メール・プッシュ配信基盤の整理とか。

ZOZO Technologies

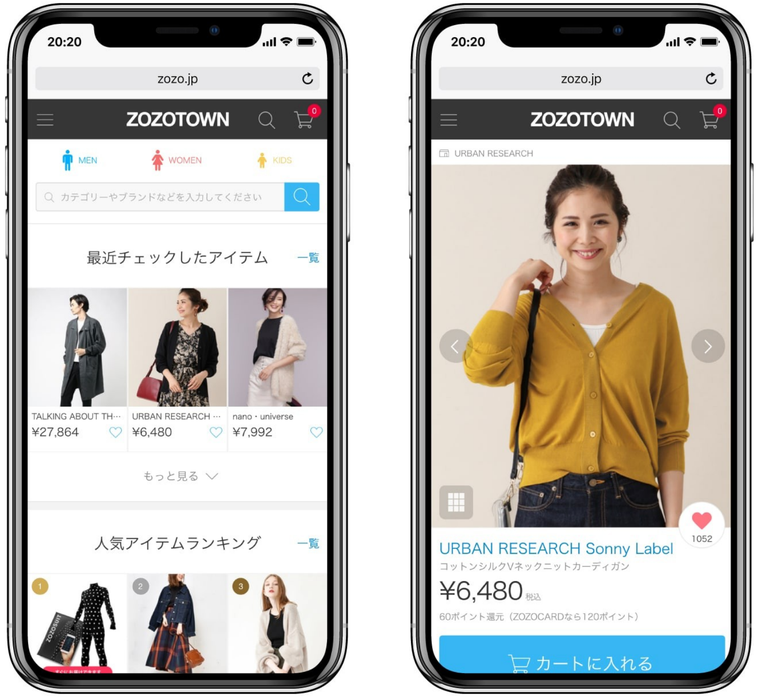
目次

・サービス紹介
・Redshift時代の紹介
・BigQuery移行の動機
・データレイク移行
・データマート移行
・インフラ紹介
・新たに生じた問題点とその解決策
・これからの展望
・まとめ

※ 技術の話多めです。エモ話はほぼないです。

ZOZO Technologies

ZOZOTOWN



○ 日本最大級のファッションショッピングサイト/アプリ

○1,100以上のショップ、6,900以上のブランドの取り扱い
（2018年9月末時点）

○ 常時65万点以上の商品アイテム数と
毎日平均3,100点以上の新着商品を掲載

○ 即日配送サービス / ギフトラッピングサービス / ツケ払い など

http://zozo.jp/

Download on the App Store

GET IT ON Google Play

ZOZO Technologies

WEAR

○ 日本最大級のファッションコーディネートアプリ

○1,200万ダウンロード突破、コーディネート投稿総数は800万件以上
（ともに2018年9月末時点）

○ 全世界（App Store / Google playが利用可能な全ての国）で
ダウンロードが可能

○ 10万人以上のフォロワーを持つユーザー（WEARISTA）も誕生

https://wear.jp/

Download on the App Store

GET IT ON Google Play

ZOZO Technologies

ZOZOSUIT

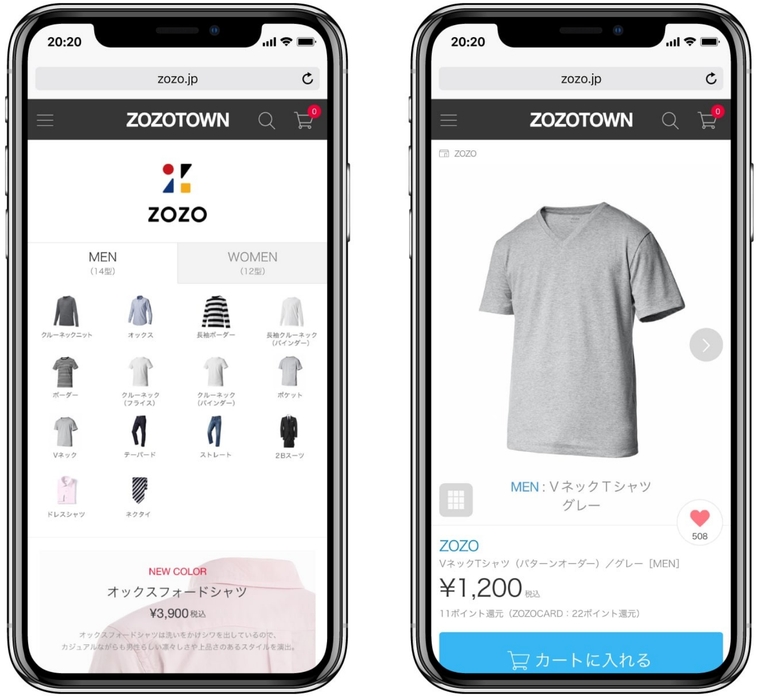
○ 当社が独自に開発した採寸用ボディースーツ

○ 全体に施されたドットマーカーをスマートフォンカメラで360度撮影することで、体型データを計測

○ 計測した体型データは、瞬時に3Dモデル化され、ZOZOTOWNアプリに保存。3Dモデルはあらゆる角度に動かすことができ、体型を360度チェックすることが可能

http://zozo.jp/zozosuit/

ZOZO Technologies



○「ZOZOSUIT」で計測した体型データをもとに、
　一人ひとりの体型に合った「あなたサイズ」のアイテム

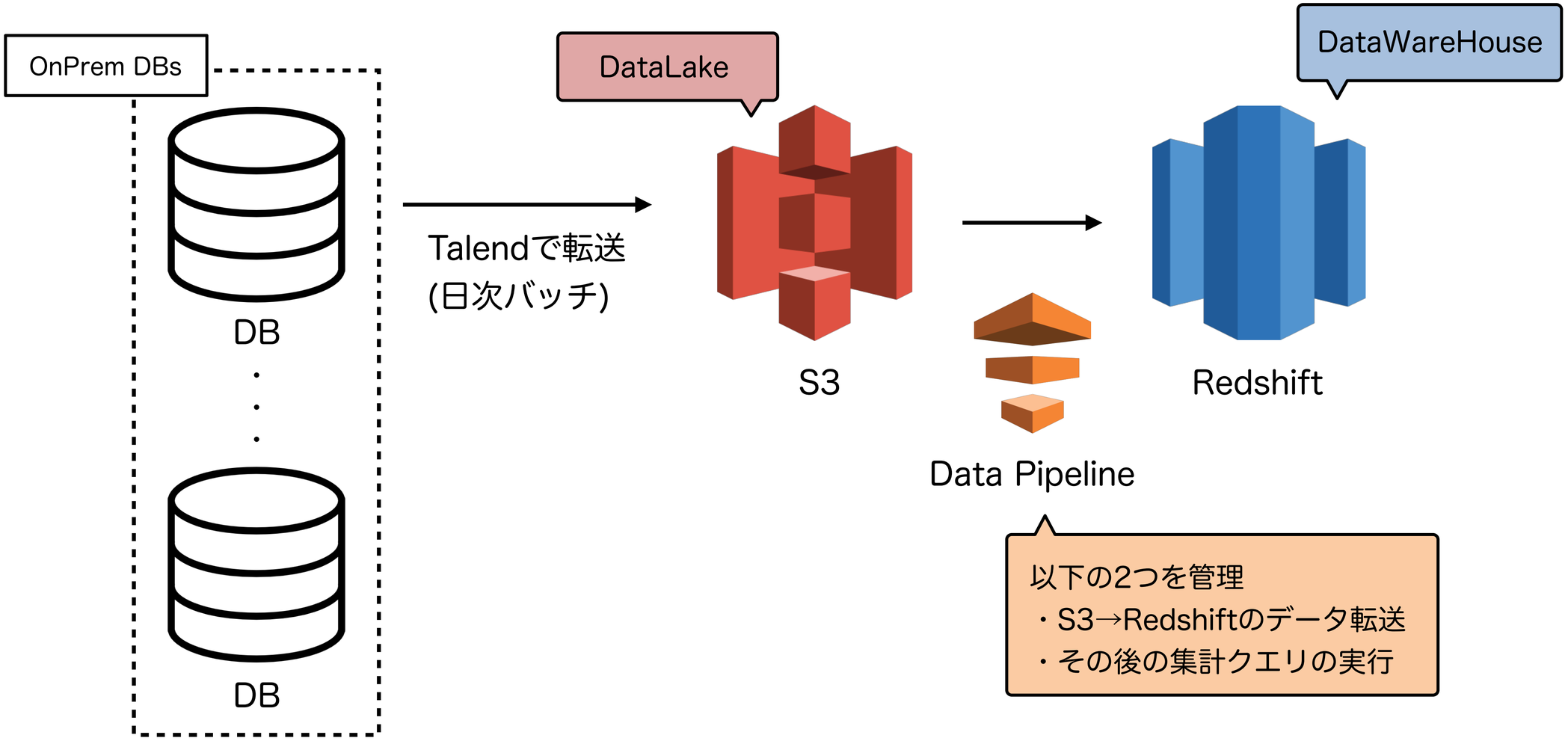
○「究極のフィット感」を実現したベーシックアイテムを提供

　グローバルサイト「ZOZO.com」で海外展開

○ アイテム：Ｔシャツ、デニムパンツ、シャツ、ビジネススーツ、ネクタイ、ボーダーＴシャツ、長袖クルーネックＴシャツ など

http://zozo.jp/pb/

Redshift時代の紹介(データフロー図)



ZOZO Technologies

S3に入れていたデータ

- ZOZOTOWNやWEARのマスタデータ
  - ユーザー情報
  - 商品情報
  - 注文情報
- GoogleAnalytics 360から取得したアクセスログ(Web・ネイティブアプリ)
- メールやプッシュの配信ログ

- 総テーブル数 >100
- 総データサイズ >1TB

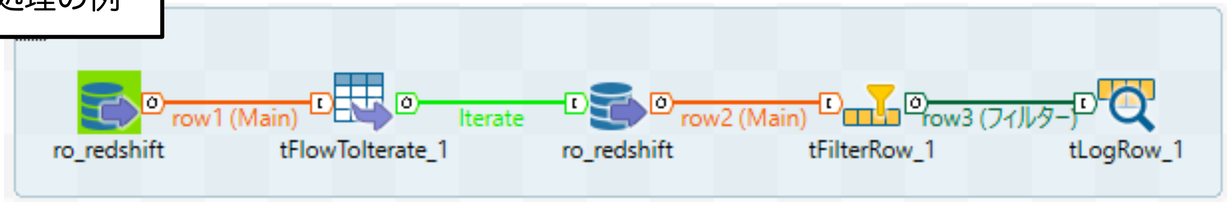
ZOZO Technologies

OnPrem DBs → S3への転送にはTalendを使用

・TalendOpenStudioという無償のETL(Extract, Transfer, Load)ツールを使用
・GUIでETL処理を記述する
・Talendで記述したETLバッチをWindowsタスクスケジューラーで起動

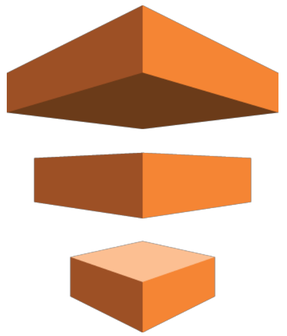
1つ1つのアイコンが実行する処理を表現
・ファイル入出力(CSV, Excel, etc.)
・DB入出力(SQL Server, PostgreSQL, Oracle, etc.)
・データ編集(置換、ソート、集計など)

ETL処理の例



S3 → RedshiftへのデータはData Pipelineを使用

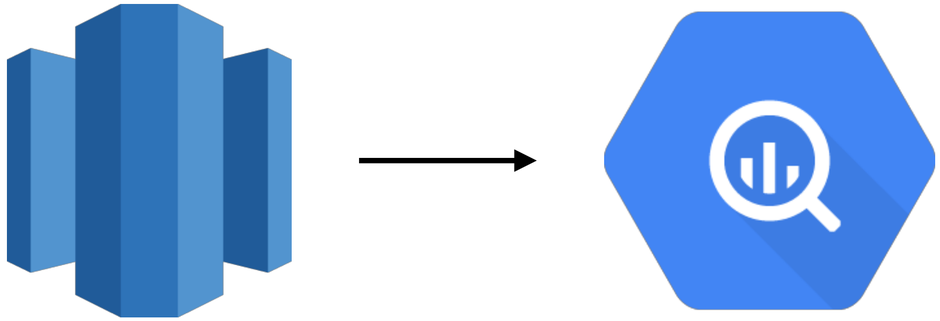
- データワークフローオーケストレーションサービス
- S3→RedshiftとRedshift上での集計クエリの実行を担当
  - RedshiftCopyActivityでS3→Redshiftの転送を実現
  - 転送完了後にSqlActivityでRedshift上でのデータ集計のSQL文を実行
- 当時の問題点
  - Talendによる転送が完了した後にData Pipelineが動く必要があるが、起動時刻は決め打ち
  - データマート集計のためのSqlActivity同士の依存感関係は人間が登録



ZOZO Technologies

BigQuery移行への動機

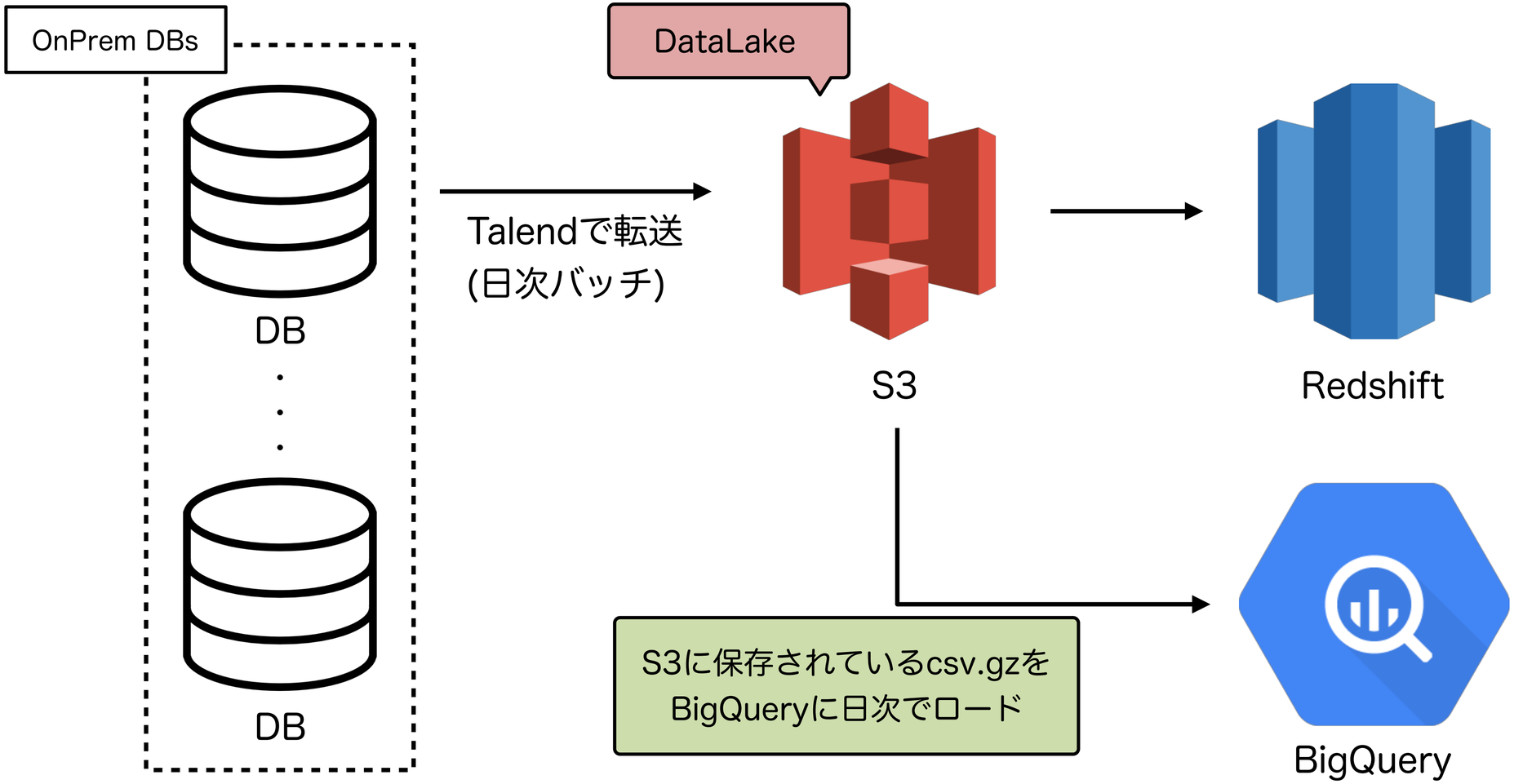
- Redshiftでは以下のものを管理する必要あり(ノウハウも不足)
  - 分散キー・ソートキー
  - ストレージの空き容量
  - ノード数・インスタンスタイプ
- BigQueryの方がよりマネージドなサービス
  - インデックスという概念なく、マシンパワーを強引に使ってフルスキャン
  - 瞬間的には100台以上ののノードで計算されることもあり
  - ストレージは実質無限で安い ($20 / TB month)
- ところで、Athenaは？
  - 2018年4月頃は当時は登場したばかり
  - クエリの実行速度はBigQueryの方が高速
  - ※データの保存形式を列指向にしたら高速になっていたかも（？）



Redshift　　BigQuery

ZOZO Technologies

DataLake移行



ZOZO Technologies

S3→BigQueryのために使用した要素技術

- Embulk
  - TreasureData製のOSS
  - ETLツール
  - 大量のデータをバッチ転送することに特化
  - データの入力・加工・出力をする部分はプラグインとして提供
  - 設定ファイルはYAMLで書かれている
- Digdag
  - こちらもTreasureData製のOSS
  - ワークフロー管理ツール
  - 大量のEmbulkジョブを効率的に管理
    - 並列実行・リトライ・タスクの実行状況の可視化・実行ログの表示
  - 設定ファイルはYAML(相当のフォーマット)で書かれている

ZOZO Technologies

EmbulkのYAML紹介

CSVファイルをstdoutに出力

```
in:
  type: file
  path_prefix: /path/to/csv_files
  decoders:
  - {type: gzip}
  parser:
    charset: UTF-8
    newline: LF
    type: csv
    delimiter: ','
    quote: '"'
    escape: '"'
    null_string: 'NULL'
    trim_if_not_quoted: false
    skip_header_lines: 1
    allow_extra_columns: false
    allow_optional_columns: false
    columns:
    - {name: id, type: long}
    - {name: account, type: long}
    - {name: time, type: timestamp, format: '%Y-%m-%d %H:%M:%S'}
    - {name: purchase, type: timestamp, format: '%Y%m%d'}
    - {name: comment, type: string}
out: {type: stdout}
```

転送設定はすべてYAMLで書かれている
以下の部分をプラグインで書くことが出来る

- input (ファイルシステムやS3などから読み出し)
- decoder (圧縮ファイルの解凍など)
- parser (デシリアライズ)
- filter (ハッシュ化などのデータの変換処理)
- output (ファイルシステムやS3への書き出し)
- encoder (ファイルの圧縮など)
- formatter (シリアライズ)

embulk

ZOZO Technologies

EmbulkのYAML紹介(with Liquid)

CSVファイルをstdoutに出力

```
in:
  type: file
  path_prefix: { env.FILENAME } # ①
  decoders:
  - {type: gzip}
  parser:
    {% include 'common/csv_format' %} # ②
    columns:
    - {name: id, type: long}
    - {name: account, type: long}
    - {name: time, type: timestamp, format: '%Y-%m-%d %H:%M:%S'}
    - {name: purchase, type: timestamp, format: '%Y%m%d'}
    - {name: comment, type: string}
out: {type: stdout}
```

EmbulkはLiquidテンプレートエンジンを搭載
以下の工夫によってYAMLファイルをDRYにできる

① 環境変数をYAMLファイルに展開
② 共通する処理を他のファイルに書き出して読み込み

ifとかforとかの制御構文を使うことも出来る

embulk

ZOZO Technologies

Digdagのワークフロー紹介

YAML(※厳密には違う)でワークフローを記述
手続き型言語に似た雰囲気
シーケンシャルなループを並列実行に切り替えるのが簡単

逐次実行

```
+my_task_1:
  sh>: "echo this task runs first."
+my_task_2:
  sh>: "echo this task runs next."
```

ループ

```
+repeat:
  for_each>:
    fruit: [apple, orange]
  _do:
    echo>: ${fruit}
```

条件分岐

```
 +run_if_param_is_true:
   if>: ${param}
   _do:
     echo>: ${param} == true
```

並列実行

```
+repeat:
  for_each>:
    fruit: [apple, orange]
  _parallel: true
  _do:
    echo>: ${fruit}
```

ZOZO Technologies

Digdagのワークフロー紹介(with Ruby)

sample.dig

```
timezone: UTC

+set_tables:
  require: tasks.rb
  rb>: set_tables

+transfer_tables:
  for_each>:
    table: ${tables}
  _parallel: true
  _do:
    sh>: embulk run embulk/${table}.yml
```

tasks.rb

```
def set_tables
  tables = Dir.glob('embulk/*.yml')
  Digdag.env.store(tables: tables)
end
```

embulk以下に格納されているすべての転送定義を読み出して
それらを引数にしてembulkを並列に実行

digdag

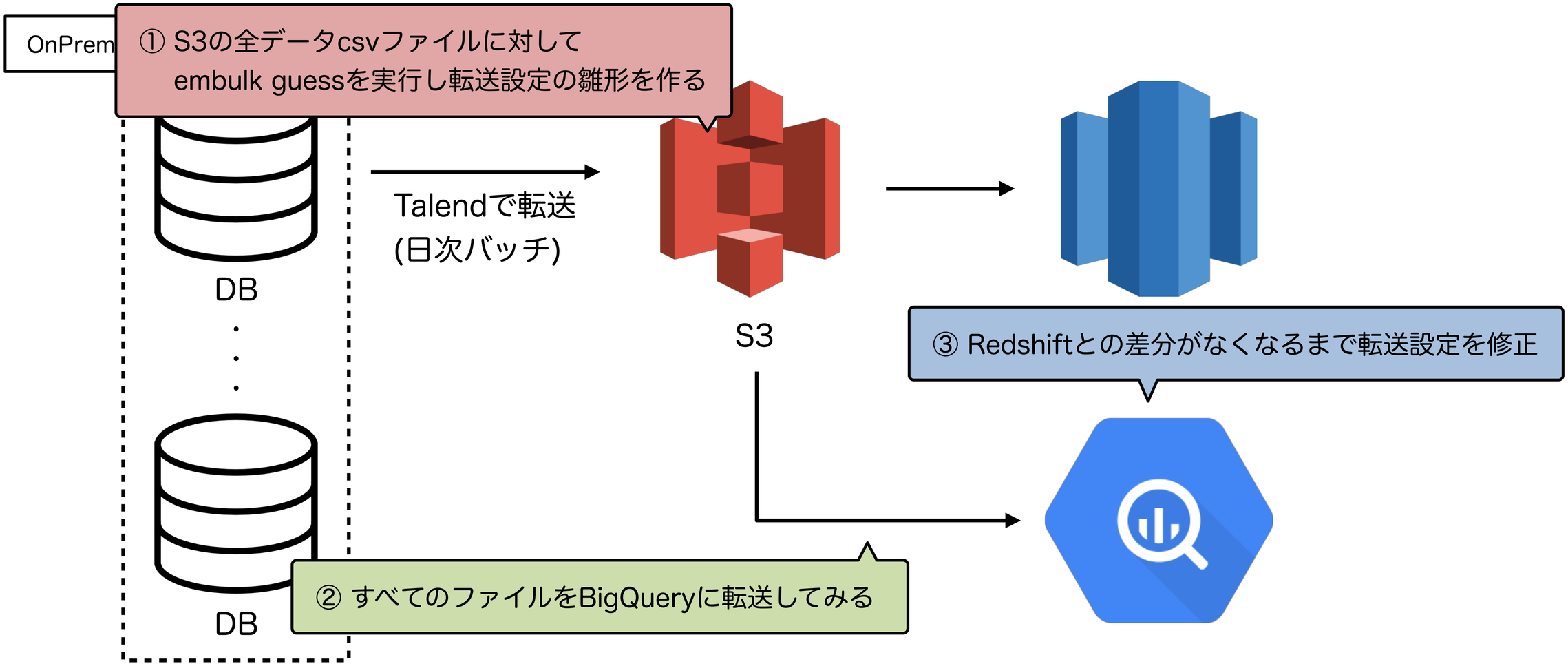
ZOZO Technologies

なぜTalendを使い続けなかったのか

- Embulk + Digdagを使うことで「普通」のソフトウェア開発でのノウハウを活かしたかった
  - 転送設定のワークフローのYAMLをgithubでバージョン管理
  - 本番環境へ出す前にチーム内でレビュー
  - CircleCIなどのCI SaaSで自動テスト
  - 本番反映作業の自動化
- Digdagを使うことでタスクの並列度の調整をしやすくしたかった
  - Embulkは可能な限りマルチスレッドで動作してCPUを絞り尽くす
  - CPUの強いインスタンスを増やして複数台で分散実行させることで全テーブルの転送時間を短縮化

ZOZO Technologies

データレイク移行手順



ZOZO Technologies

RedshiftとBigQueryでのデータ差異の原因

・多くの場合はEmbulkのYAMLでの型指定ミス
- Embulk guessではcsvファイルのスキーマを100%の精度で推測できない(型情報がないCSVの限界)
- 先頭行のみをみてスキーマの推測を行う
- ファイルの後方で予想外のデータがあることもあることもある
- Embulkの実行ログでスキップされた行を確認して、Redshiftのスキーマと突き合わせ
- ※今に思えば最初にRedshiftのスキーマとの突き合わせをしておけば良かった・・・

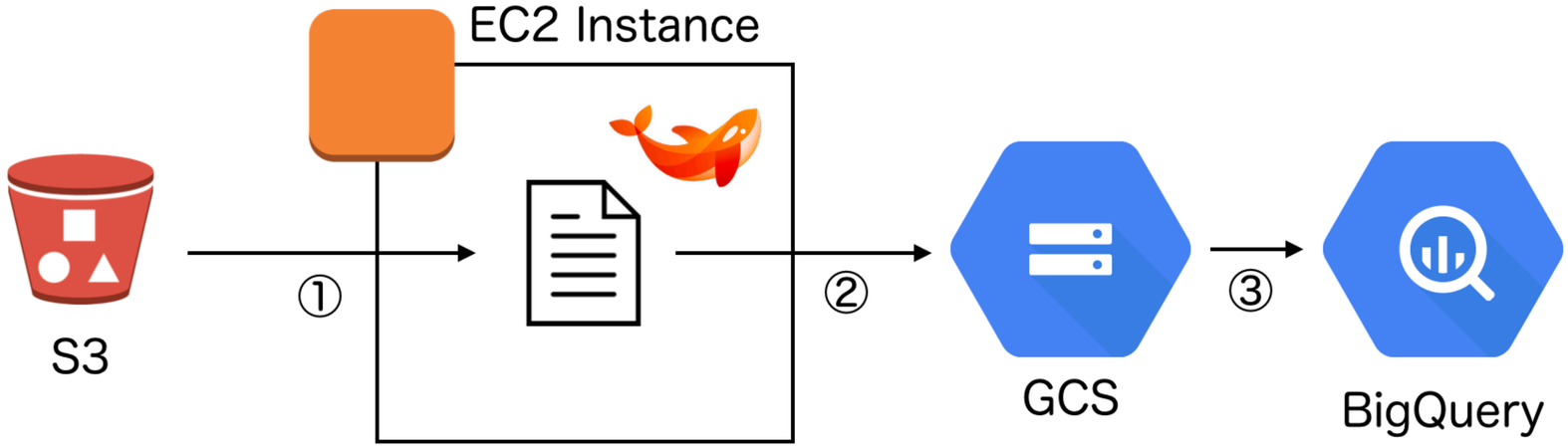
・Embulkのcsvパーサーは文字列中の'\0'を正しくパース出来ない
- 何回リトライしても消えないエラーを確かめるためにバイナリエディタで確認
- Embulkのソースコードを確認して原因に気付く
- GitHubにIssueは上がっているけど直されていない・・・

ZOZO Technologies

S3→BigQueryの転送の高速化のために行ったこと

・embulk-output-bigqueryプラグインのデータ転送の様子は下図のとおり
・GCSに配置するファイルを無圧縮にすると転送速度が数倍高速になった (③の部分)
・GCSからBigQueryへのデータロードが並列になったことが要因

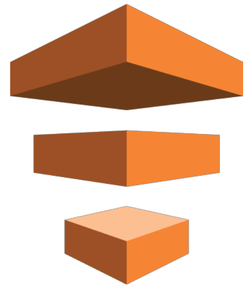
詳細な話はこちら
https://qiita.com/shiozaki/items/20535b94e2656c03620f



ZOZO Technologies

差分更新の仕組み(Redshift+Datapipeline)

- 一部の巨大なテーブルは毎日全量更新をしていない
- 最近に変更のあった行のみがS3に配置される
- RedshiftCopyActivityのinsertModeオプションで差分更新を実現
  - APPEND(単純にテーブルに追記)
  - OVERWRITE_EXISTINGを使用(主キーを指定し、主キーの重複があったら上書き)



ZOZO Technologies

差分更新の仕組み(BigQuery)

- 同様の差分更新の仕組みをBigQueryで作る必要があり
- APPEND方式の差分更新
  - SDKからクエリを投げる時のwriteオプションでappendを指定
- OVERWRITE方式の差分更新
  - WINDOW関数で旧テーブルのデータか新テーブルのデータかを判別
  - 重複する場合は新テーブルのデータのみを残す
  - 参考: 数百GBのデータをMySQLからBigQueryに同期する
  - https://tech.mercari.com/entry/2018/06/28/100000

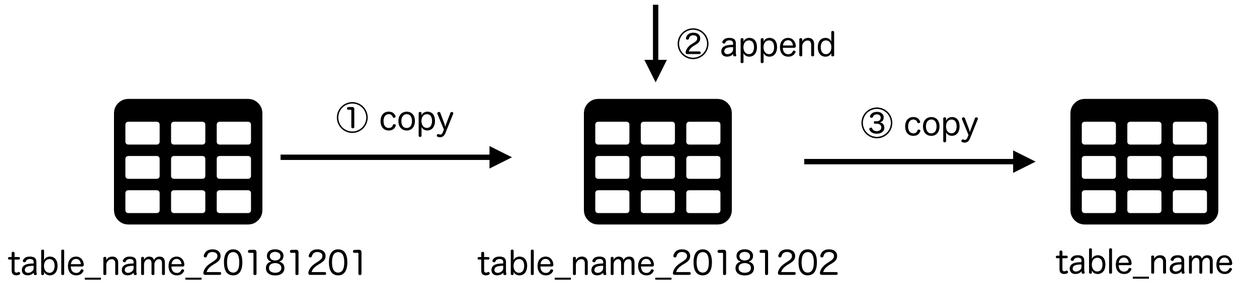
- これだと更新処理が冪等にならないので、もう一捻りした

ZOZO Technologies

冪等な差分更新の仕組み(BigQuery)

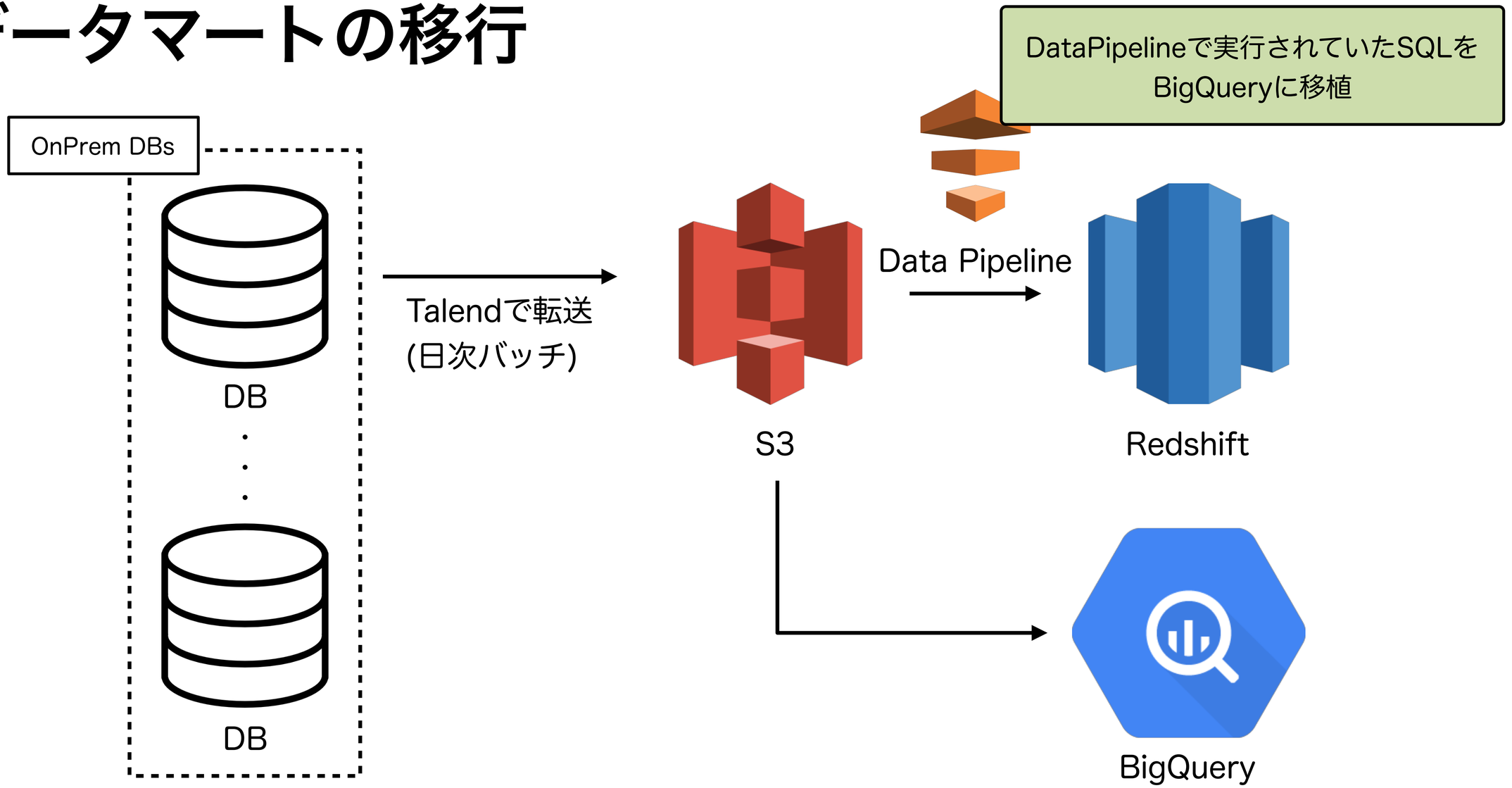
- 日毎のバックアップテーブルを作ることで差分更新を冪等に
- ①〜③のどの処理で落ちても安全に最初からやり直すことができる
- BigQueryにはトランザクションがないので、クエリを投げる側で調整

```
table_yesterday = table('table_name_20181201')
table_yesterday.copy('table_name_20181202')                                          # ①
table_today = table('table_name_20181202')
query("集計クエリ", destination_table: 'table_name_20181202', write: 'append')  # ②
table_today.copy('table_name')                                                       # ③
```



ZOZO Technologies

データマートの移行



ZOZO Technologies

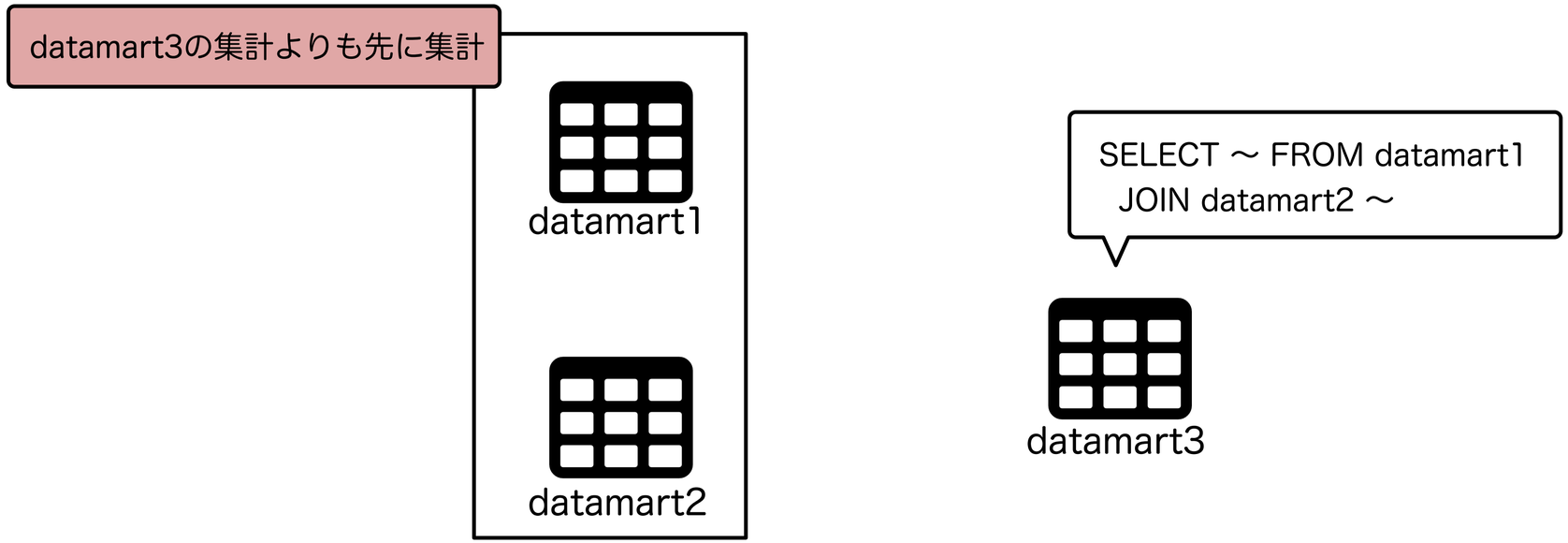
データマートの移行

- RedshiftのSQLで書かれたデータマート集計用のクエリをBigQueryのSQLに変換
- PostgreSQL方言のSQLをBigQuery方言のSQLに書き換え
- 必要なもの: 根気
  - NULLを含んだときのJOINの挙動が違ったり
  - 関数の引数の順番が違ったり
  - 一部の関数はBigQueryになかったり
  - etc...

ZOZO Technologies

データマートの集計の依存関係解決

・あるデータマートが別のデータマートの集計結果を使っている場合、依存関係が発生
・順番は守りつつ、可能な限り並列度は上げたい
・makeコマンドの -j オプションみたいな感じ



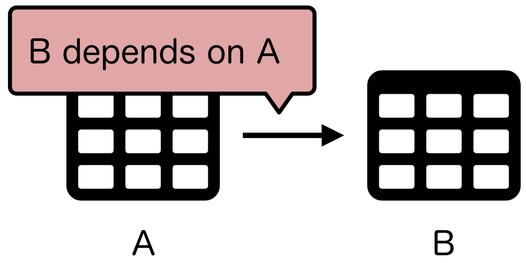
データマートの集計の依存関係解決

1. データマートのSQL文を解析して依存関係をグラフ構造(DAG)にする

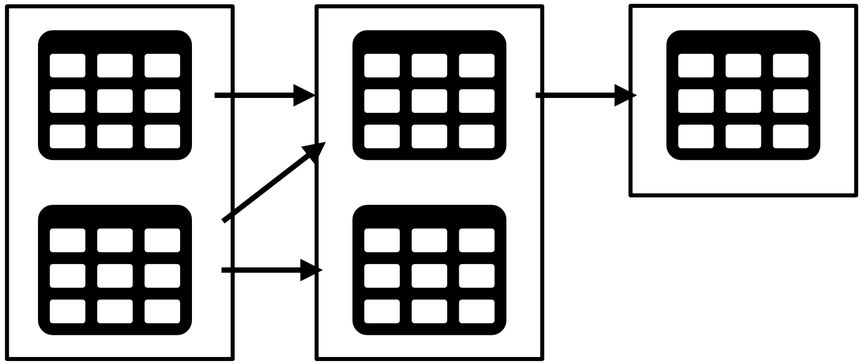
```
sql.scan(/(?:FROM|JOIN)\s+`(.+?)`/i)
```

2. DAGを解析していくつかの並列実行可能なグループに分割

①



②



ZOZO Technologies

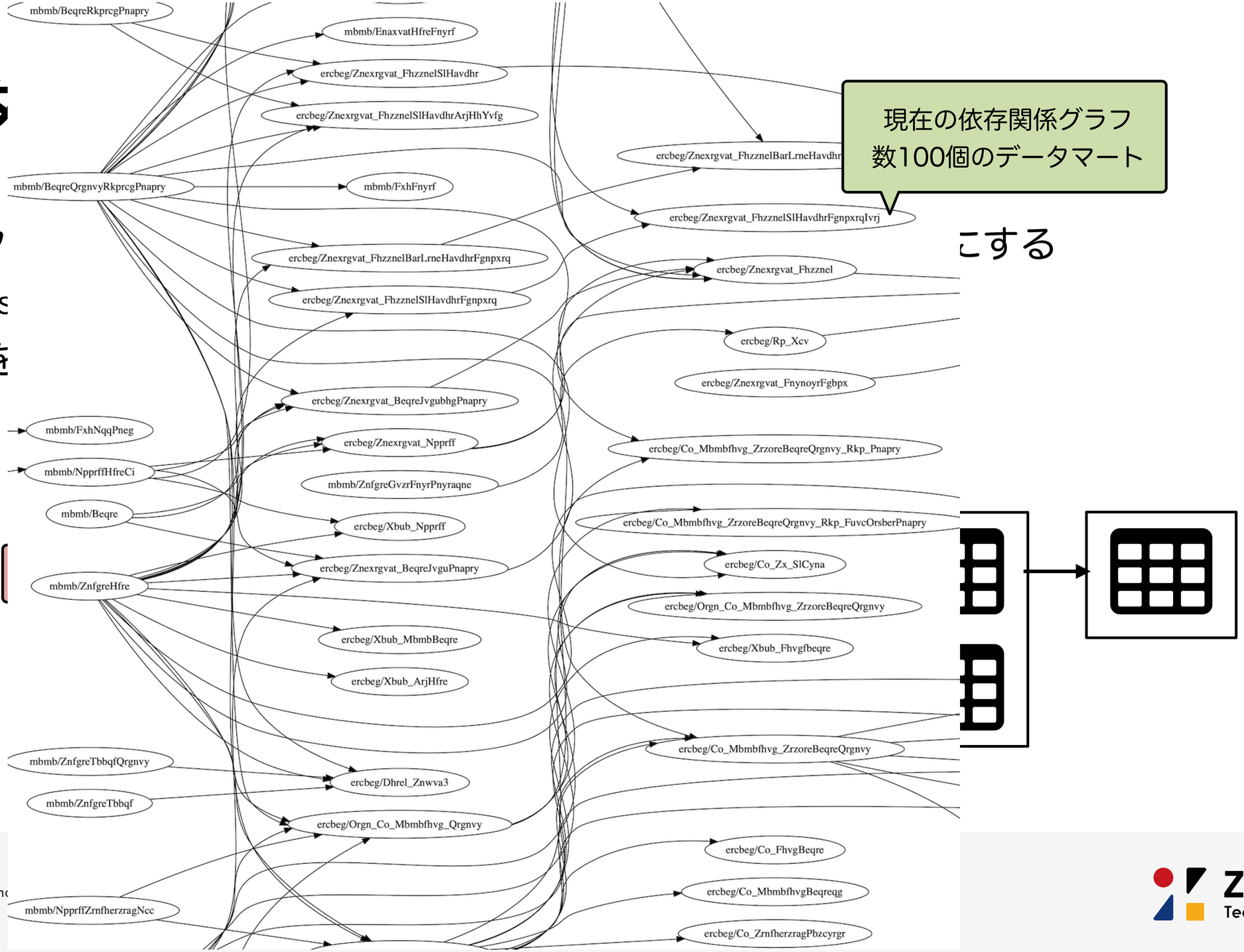
データ

1. データ

sql.s

2. DAGを

①

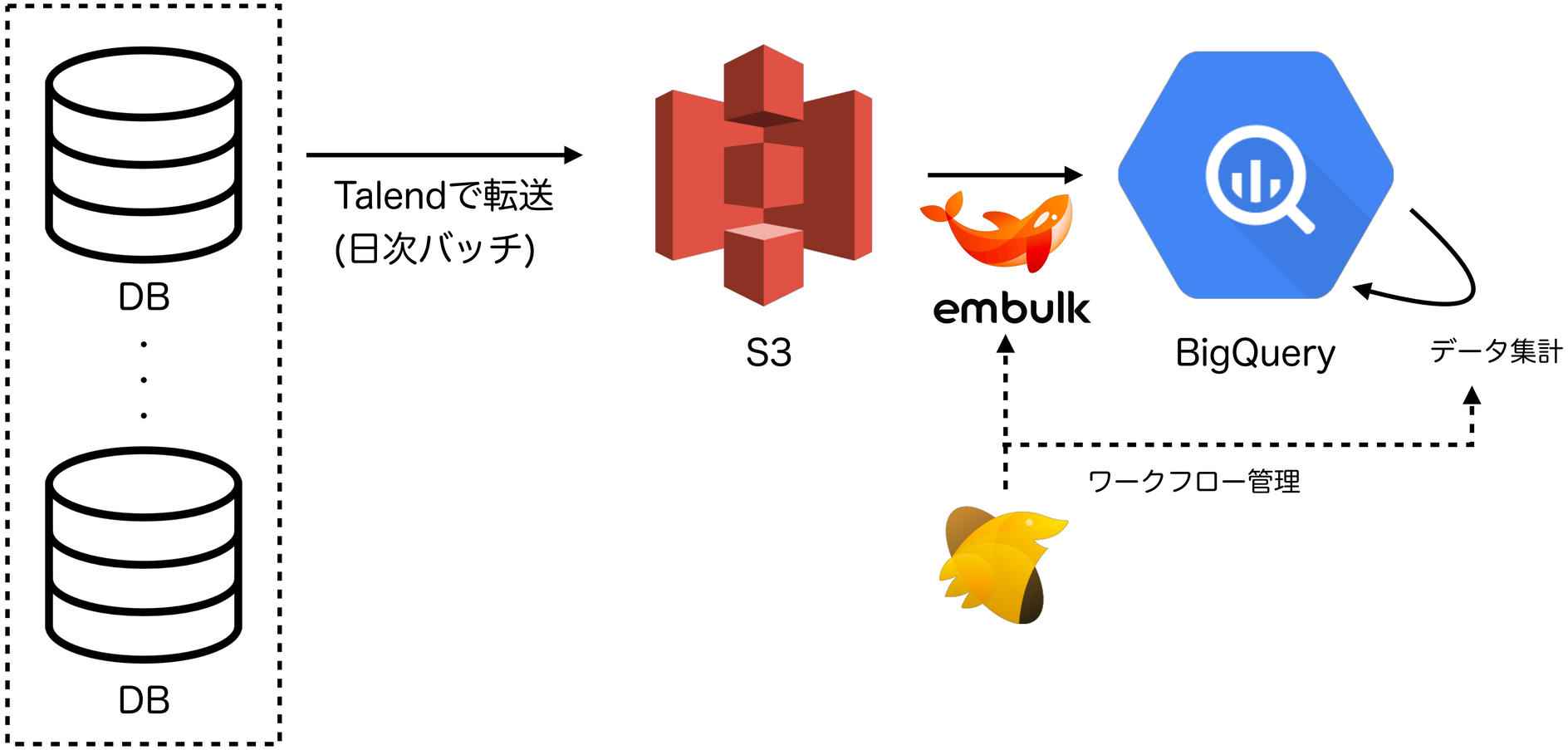


現在の依存関係グラフ
数100個のデータマート

こする

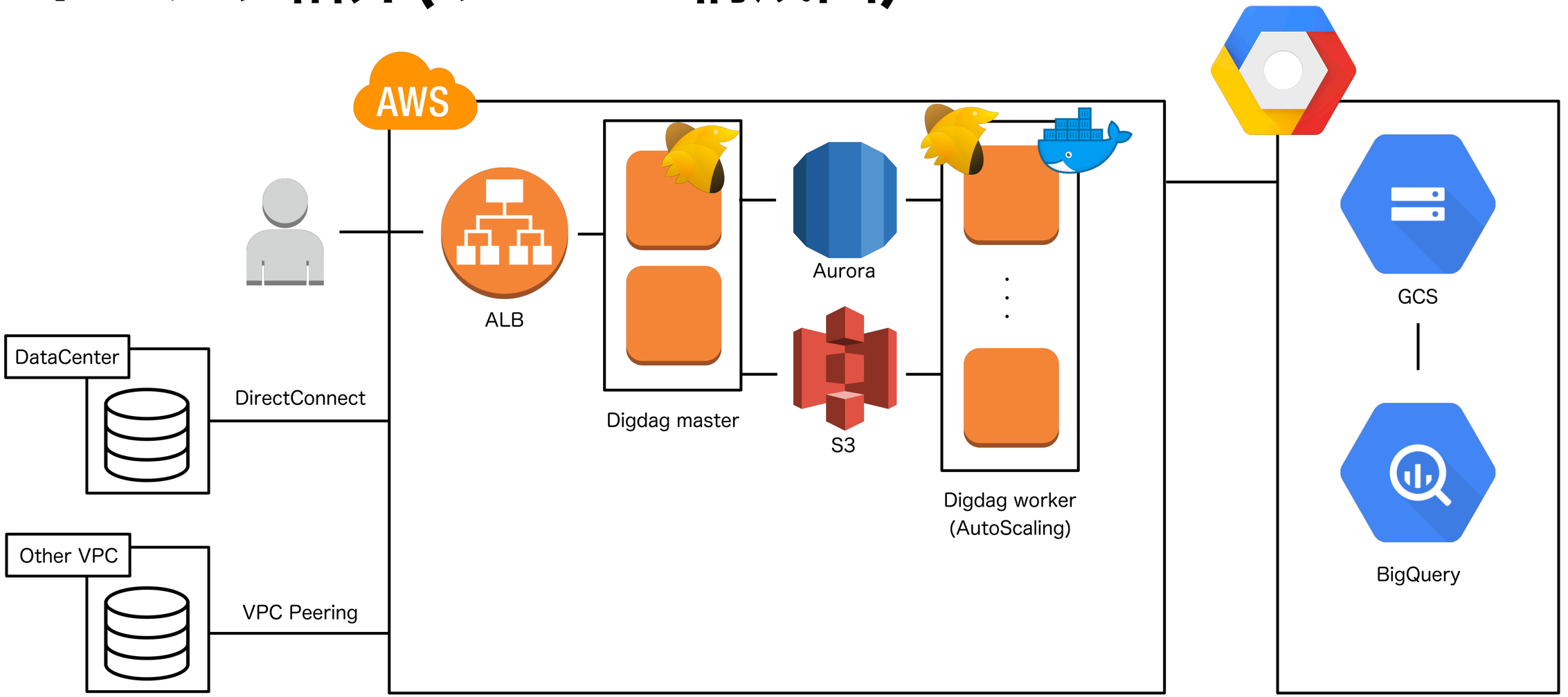
ZOZO Technologies

インフラ紹介(データフロー図)



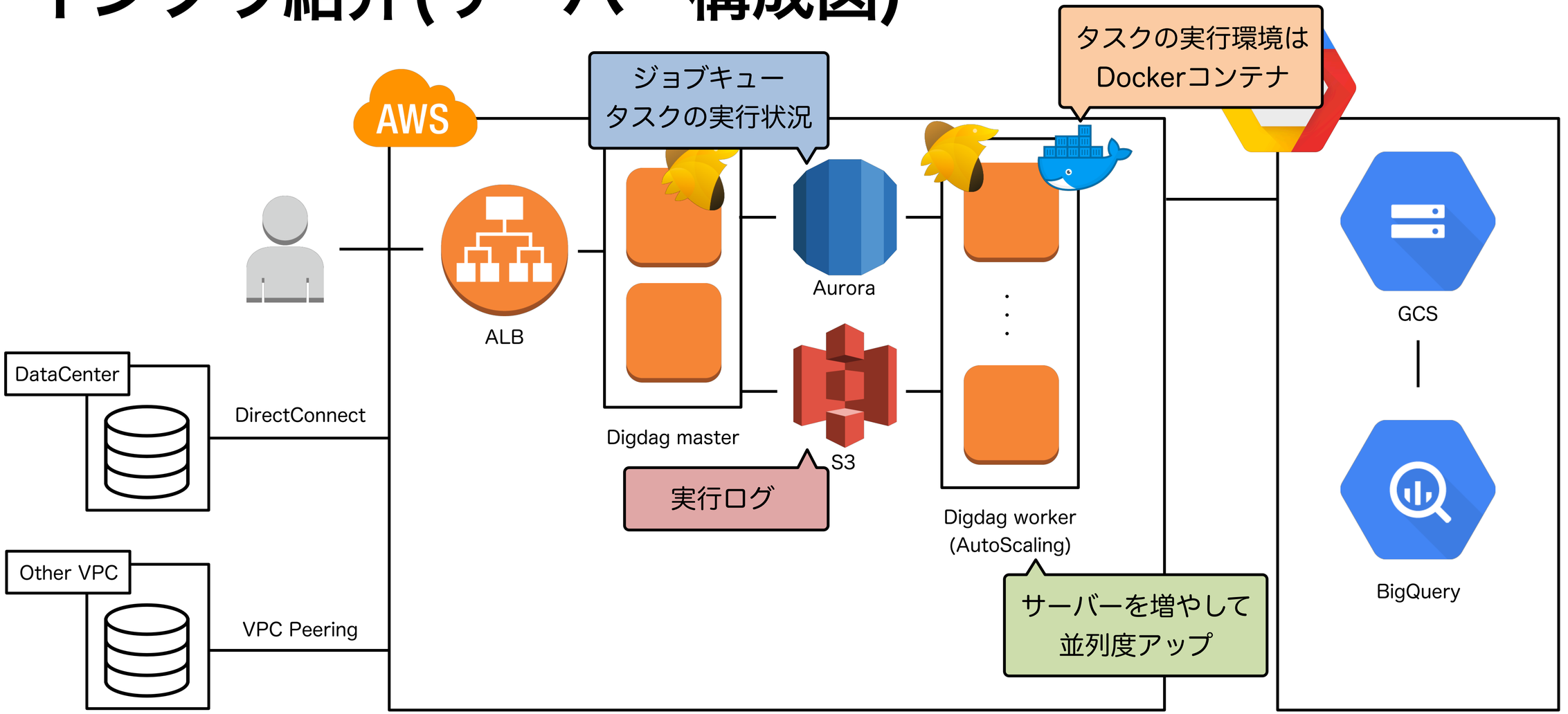
ZOZO Technologies

インフラ紹介(サーバー構成図)



ZOZO Technologies

インフラ紹介(サーバー構成図)



ZOZO Technologies

新たに発生した問題とその対処

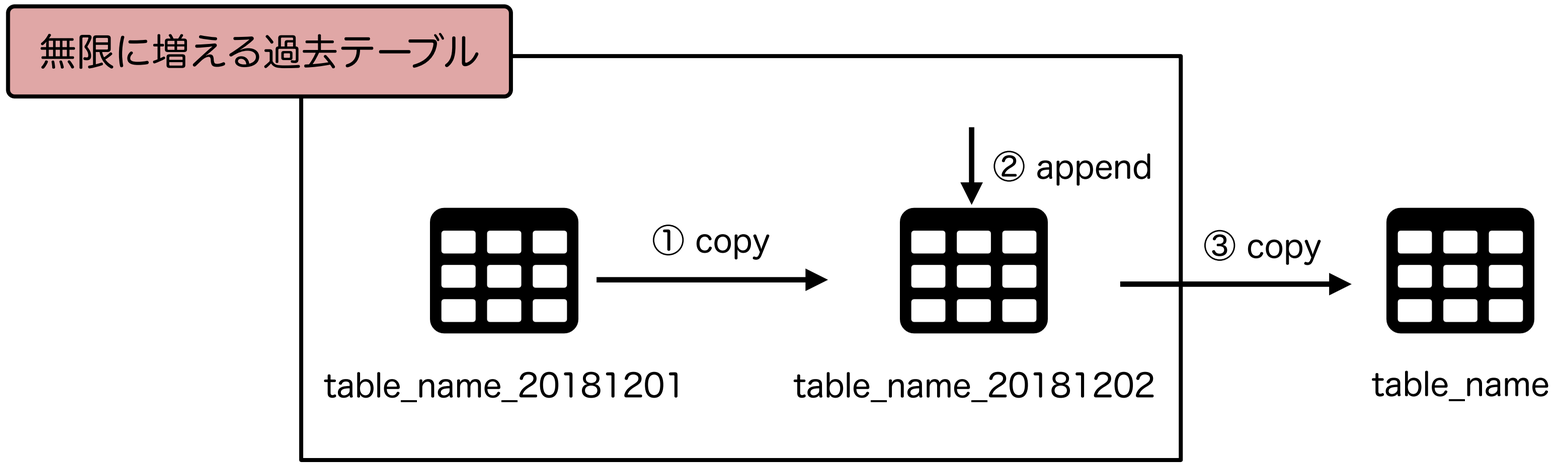
・データ保存料金跳ね上がり
　→ 定期的に古いデータを削除
・クエリ料金跳ね上がり
　→ Partitioned Table
・本番に出すまで落ちるか不明なクエリ
　→　masterマージ前にCircleCIでの自動テスト

ZOZO Technologies

データ保存料金跳ね上がり

・差分更新の時に過去のテーブルが残り続ける
  ・1つのテーブルのサイズが数TB
  ・BigQueryはストレージが無制限とはいえお金はかかる

・対処: 日付の古すぎるテーブルの自動削除



ZOZO Technologies

クエリ料金跳ね上がり

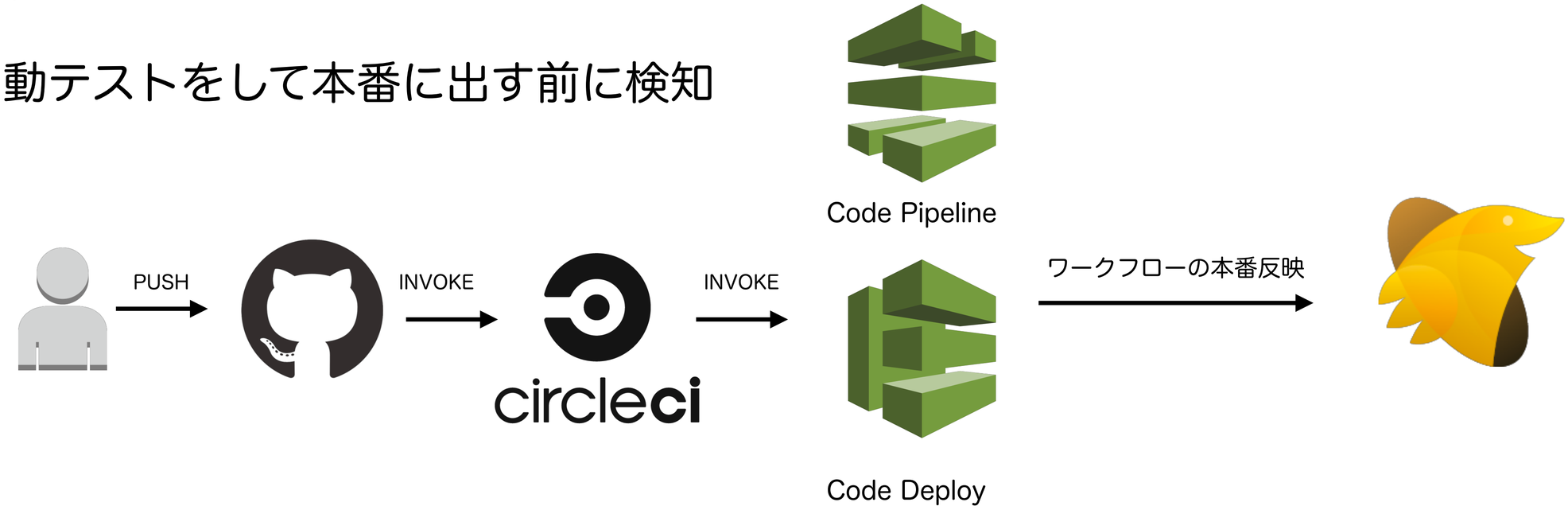
・あるデータマートのスキャン量が数TBになっていた
  ・SELECT * などの邪悪なことはしていない
  ・単に非常に縦長なログテーブルだった(数年分)
  ・必要なデータは直近のものだけ

・対処: TimePartitionedテーブル
・TIMESTAMP型の列を指定してパーティション分割
・WHERE句で絞り込むことでスキャン量を抑えられる

ZOZO Technologies

本番に出すまで落ちるか不明なクエリ

・データマート作成用のクエリのミスが頻発
  ・文字コードがSJIS
  ・テーブル名を参照するときのプロジェクト名忘れ
  ・循環参照するデータマート
  ・etc...
・CircleCIで自動テストをして本番に出す前に検知



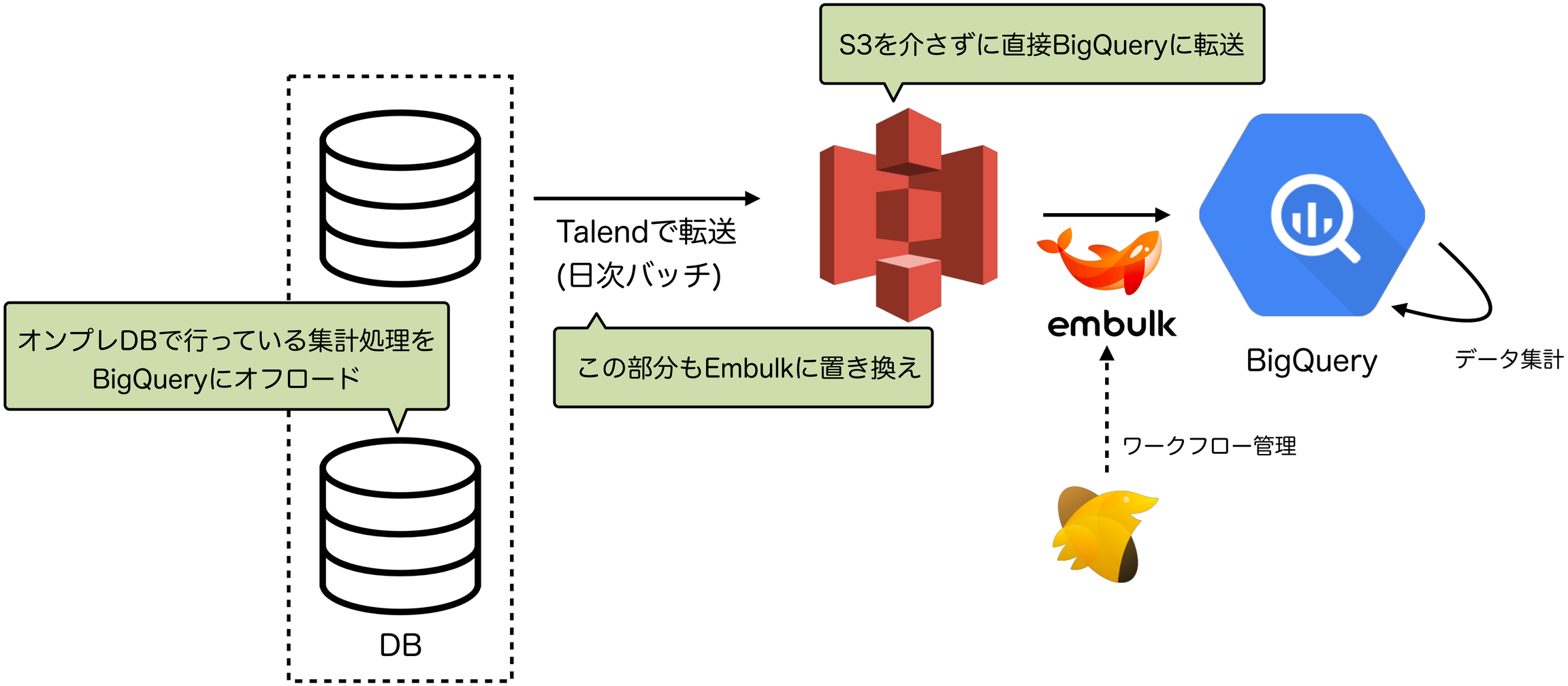
ZOZO Technologies

BigQueryの利用をお考えの方へのアドバイス

- BigQueryはパワフルすぎるので、課金額にご用心
  - 数TBのデータを一瞬でスキャンする
  - Quotaの設定
  - 課金額を毎日チェック
- 日本語ドキュメントを信用しない
  - 英語版にしか存在しない機能
  - 英語版ではBetaの表記が取れている機能
- 英語ドキュメントすら信用しない
  - 最終的に一番信頼できるのはSDKのソースコードとREST APIのドキュメント
  - SDKの使い方を知りたい場合は、ドキュメントではなくUnit Testを読みましょう
  - ※Ruby版のSDKだけの問題かもしれない

ZOZO Technologies

これからの展望



ZOZO Technologies

まとめ

・ZOZOTOWNのDWHをRedshiftからBigQueryに引っ越しした
・BigQueryにしたことによって運用負荷の低減に繋がった
  ・Redshift時代と比べて人間が気にする必要のある「モノ」が減った
  ・完全マネージドサービスなので力技を使いやすい

・まだまだやりたいことが多すぎて人が足りない
・ブースも出しているので気軽に遊びに来てください〜
・https://tech.zozo.com/recruit/

ZOZO Technologies