ORACLE

Microservices 関連新機能

Oracle Database 23c新機能セミナー

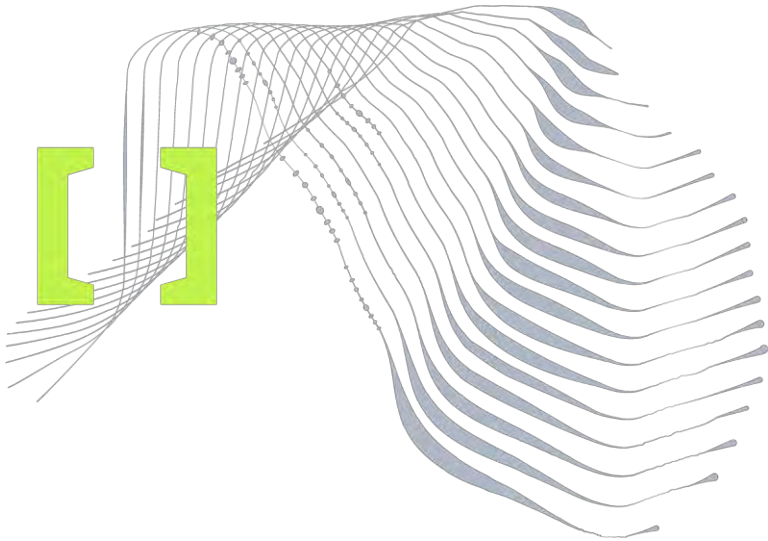
出口 龍之介

日本オラクル株式会社

2023/10/20

アジェンダ

1. Transactional Event Queues (TxEventQ) 関連新機能
2. Oracle Database Sagas (OSaga)
3. Oracle SQL Access to Kafka (OSaK)



マイクロサービス・アーキテクチャとは

サービス間の影響を極小化しシステムの変更容易性を高めるアーキテクチャ

保守とテストの容易性

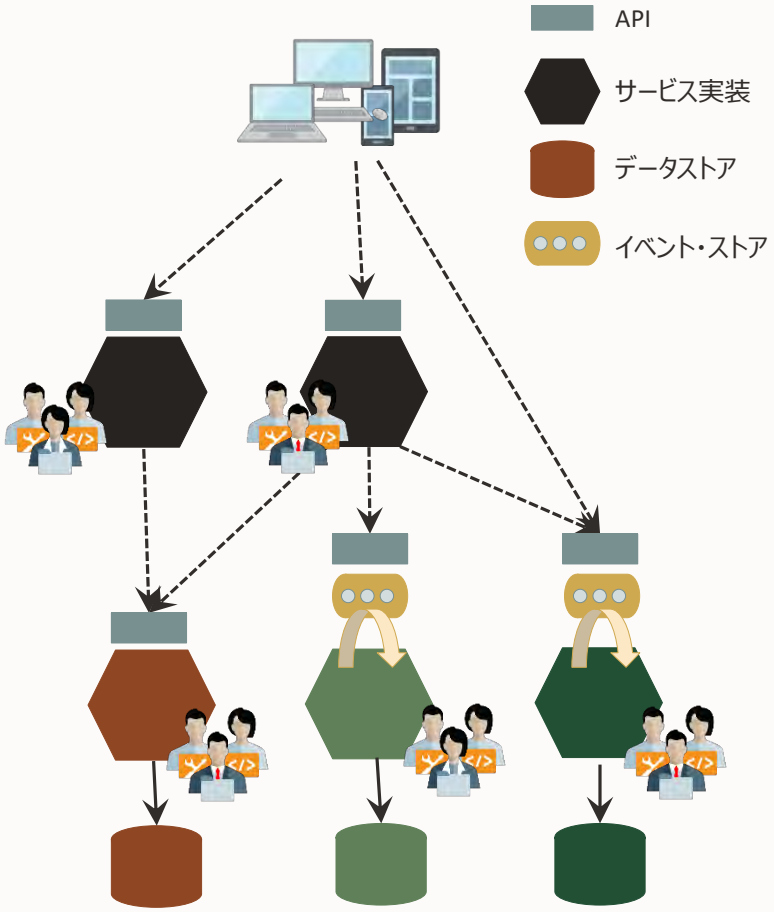
- 分割したサービス毎に組織を編成し開発・運用の自由度を高める
- 更新単位を最小限にすることでテスト規模を最小化

疎結合

- API化や非同期化によりサービス間の結合度を低減
- 変更による他の稼働中のサービスにへの影響を極小化

独立してデプロイ可能

- データソースやアプリケーション・モジュールをサービス毎で占有
- デプロイやスケールの変更の単位サービス毎で任意に最適化



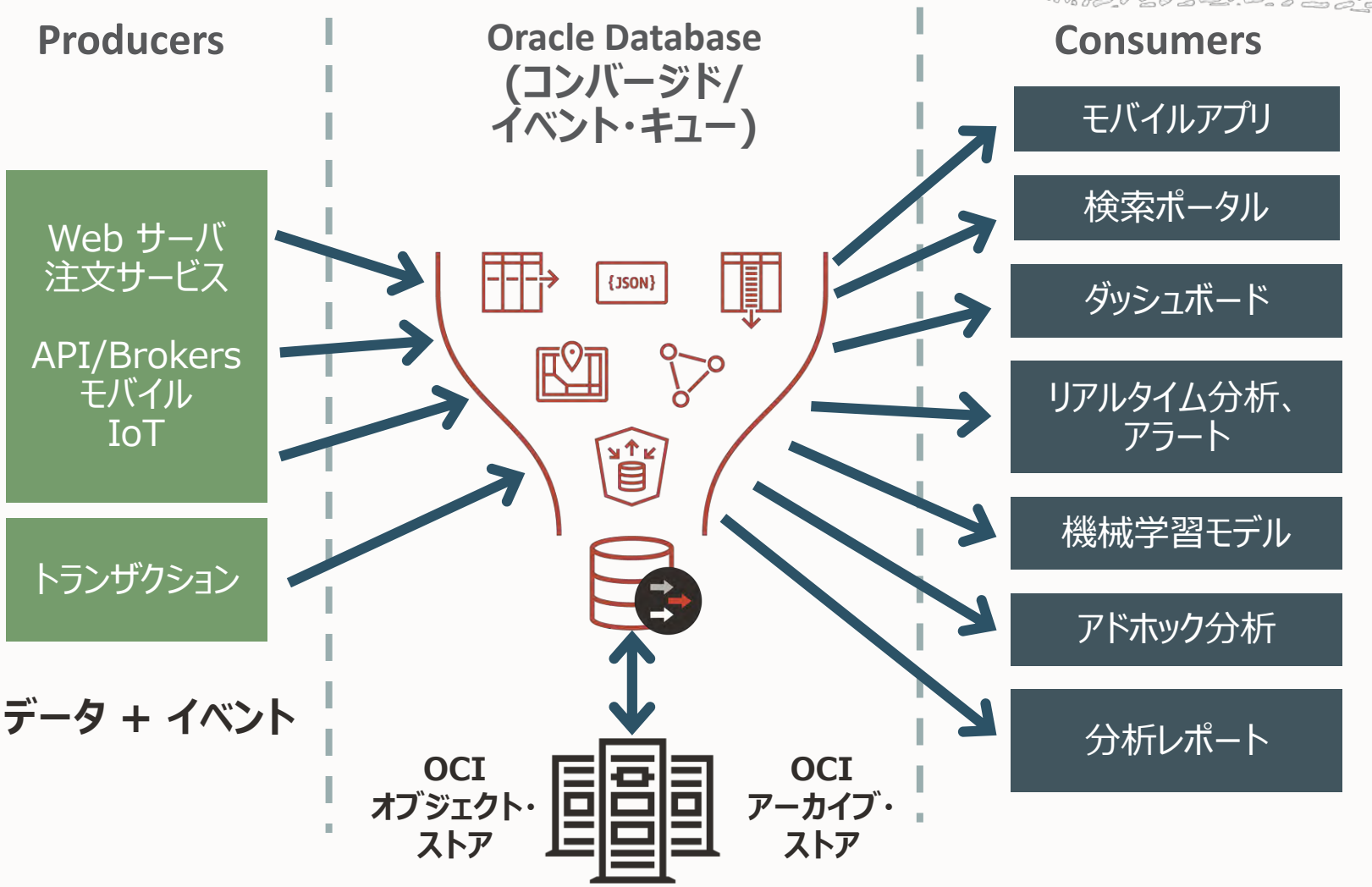
Transactional Event Queues (TxEventQ) 関連新機能

Oracle Advanced Queuing (AQ) / Oracle Transactional Event Queues (TxEventQ) とは

Oracle Databaseに統合された堅牢で多機能なメッセージ・キューイング・システム

- TxEventQへのエンキュー/デキュー、TxEventQと他のメッセージングシステム間のメッセージ伝播機能を提供
- 標準のデータベース機能（リカバリ、セキュリティなど）がサポートされる
- データ駆動およびイベント駆動のアーキテクチャから求められる要件に対処
- あらゆる機能が自動化されたマネージドなサービス
- データベースと統合してパフォーマンスを最適化

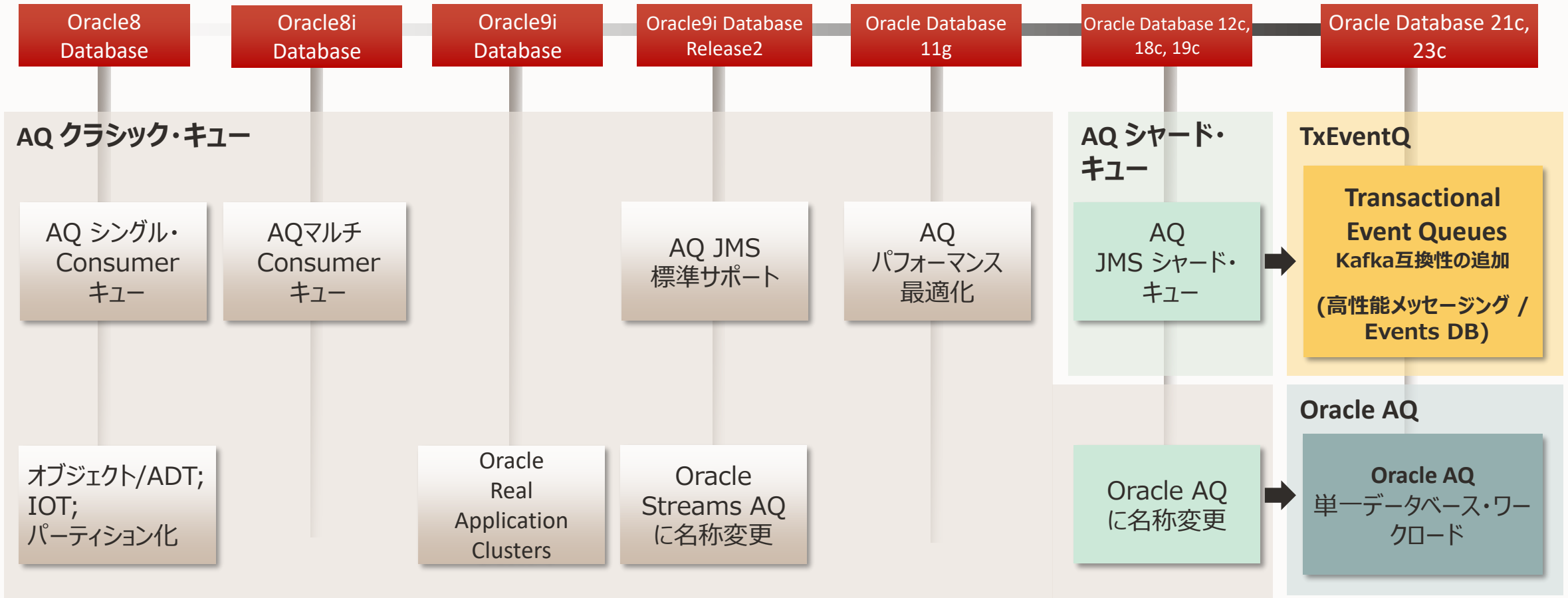
Oracle Advanced Queuing (AQ) / Oracle Transactional Event Queues (TxEventQ) とは



- データの統合
- マイクロサービスのサポート
- オープンなインタフェース
- 簡単なアプリ構築とAPI
- バックアップ、セキュリティなどの管理タスクの軽減

Oracle AQおよびトランザクション・イベント・キュー(TxEventQ) の歴史

Oracle TxEventQはOracle AQシャード・キューの21c以降の更新バージョン

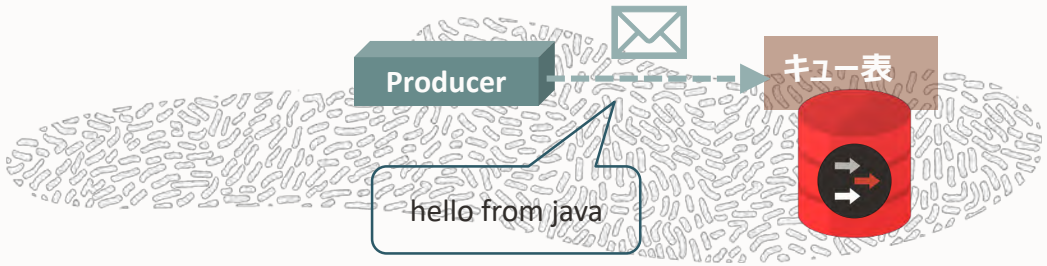


TxEventQ の基本的な流れ
サンプル・コード

キュー(トピック)の作成(PL/SQL)

```
begin
    -- TxEventQの作成
    dbms_aqadm.create_transactional_event_queue(
        queue_name         => 'my_teq',
        -- mutiple_consumersが true の場合, pub/subトピックが作成される
        multiple_consumers => true
    );
    -- TxEventQの開始
    dbms_aqadm.start_queue(
        queue_name         => 'my_teq'
    );
end;
/

-- サブスクライバの登録
declare
    subscriber sys.aq$_agent;
begin
    dbms_aqadm.add_subscriber(
        queue_name => 'my_teq',
        subscriber => sys.aq$_agent(
            'my_subscriber',   -- サブスクライバ名
            null               -- 通知のためのアドレスを設定
        )
    );
end;
/
```

TxEventQ の基本的な流れ
サンプル・コード

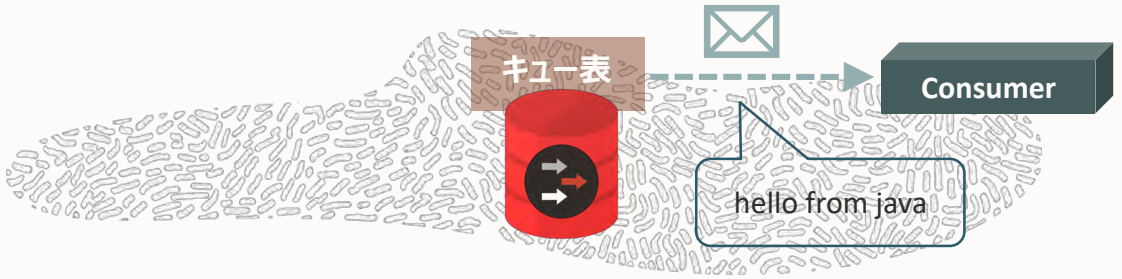


メッセージのパブリッシュ(Java)

```
public class PublishTEQ {

    private static String username = "pdbadmin";
    private static String url = "jdbc:oracle:thin:@//localhost:1521/pdb1";
    private static String topicName = "my_teq";

    public static void main(String[] args) throws AQException, SQLException, JMSException {

        // セッション作成は割愛
        // メッセージのパブリッシュ
        Topic topic = ((AQjmsSession) session).getTopic(username, topicName);
        AQjmsTopicPublisher publisher = (AQjmsTopicPublisher) session.createPublisher(topic);

        AQjmsTextMessage message = (AQjmsTextMessage) session.createTextMessage("hello from java");
        publisher.publish(message, new AQjmsAgent[] { new AQjmsAgent("my_subscriber”, null) });
        session.commit();

        // クリーンアップ
        publisher.close();
        session.close();
        conn.close();
    }
}
```

```
import java.sql.SQLException;

import javax.jms.JMSException;
import javax.jms.Session;
import javax.jms.Topic;
import javax.jms.TopicConnection;
import
javax.jms.TopicConnectionFactory;
import javax.jms.TopicSession;

import oracle.AQ.AQException;
import oracle.jms.AQjmsAgent;
import oracle.jms.AQjmsFactory;
import oracle.jms.AQjmsSession;
import oracle.jms.AQjmsTextMessage;
import oracle.jms.AQjmsTopicPublisher;
import oracle.ucp.jdbc.PoolDataSource;
import
oracle.ucp.jdbc.PoolDataSourceFactory;
```

TxEventQ の基本的な流れ
サンプル・コード



メッセージのコンシューム(Java)

```
public class ConsumeTEQ {

    private static String username = "pdbadmin";
    private static String url = "jdbc:oracle:thin:@//localhost:1521/pdb1";
    private static String topicName = "my_teq";

    public static void main(String[] args) throws AQException, SQLException, JMSException {

        // セッション作成は同様
        // メッセージが到着すると、printする
        while (true) {
            // 1秒でタイムアウト
            AQjmsTextMessage message = (AQjmsTextMessage) subscriber.receive(1000);
            if (message != null) {
                if (message.getText() != null) {
                    System.out.println(message.getText());
                } else {
                    System.out.println();
                }
            }
            session.commit();
        }
    }
}
```

AQシャード・キューからTxEventQへ

AQ シャード・キューから進化した新しいメッセージ・キューイング・システム

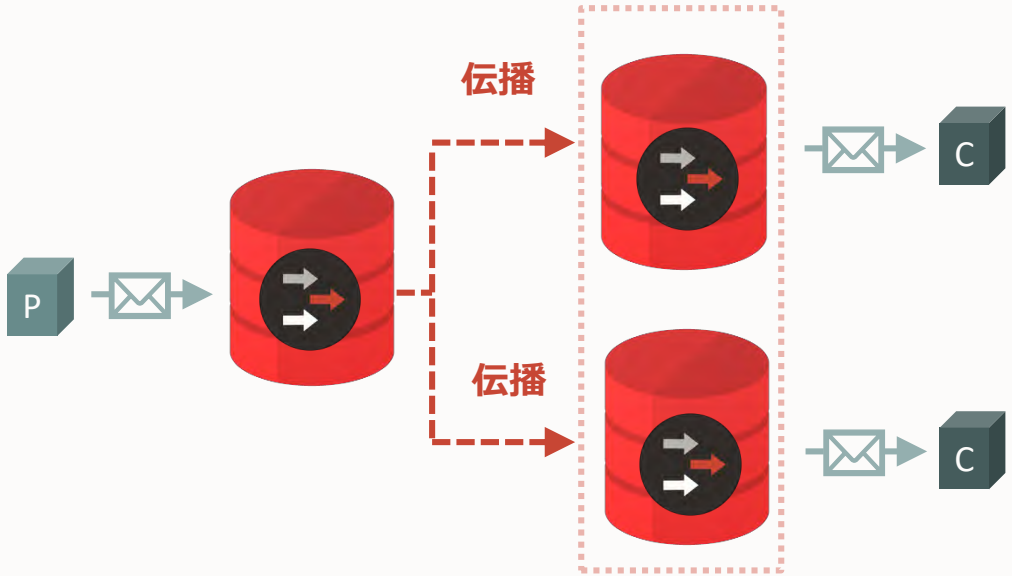
23c新機能

- 伝播機能
- イベントストリームの同時実行性が向上
- TxEventQ用のKafka Javaクライアント/Kafka実装の拡張
- 多くの言語のサポート
- リアルタイムなパフォーマンス監視
- AQからTxEventQへのオンライン移行ツール

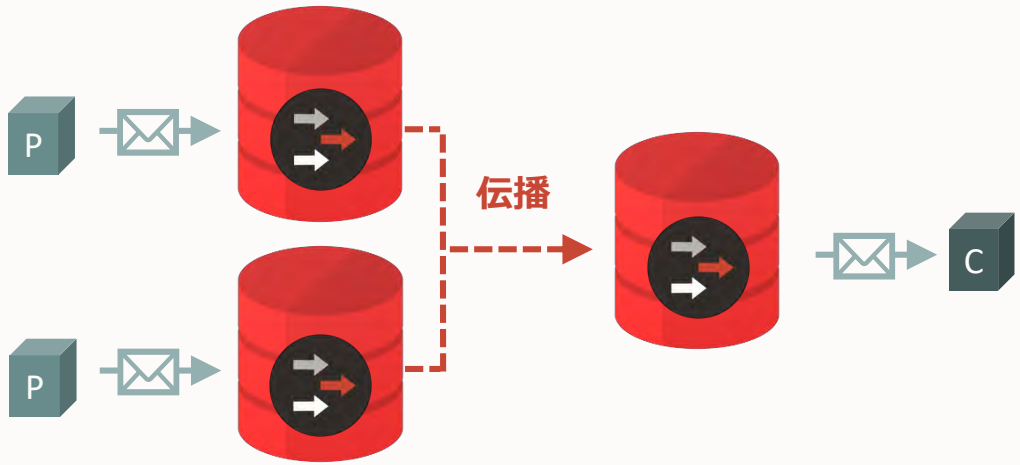
Oracle AQ / TxEventQ のメッセージ伝播
リモート・サブスクライバへの伝播

- メッセージは1つのキューから別のキューに伝播できる
- アプリケーションは同じデータベース・キューに接続されていなくても相互に通信可
  （宛先キューは同じデータベースでもリモート・データベースでも可能）➡ リモートでの処理のオフロード、バックアップ
- 伝播により、多くの受信者にメッセージを展開可能、異なるキューのメッセージを1つのキューに結合することもできる
- JMSセッション・レベルの順序付けセマンティクスを利用し、宛先のキューに送信

1つのキューから異なる複数の別のキューに伝播



異なるキューのメッセージを1つのキューに伝播

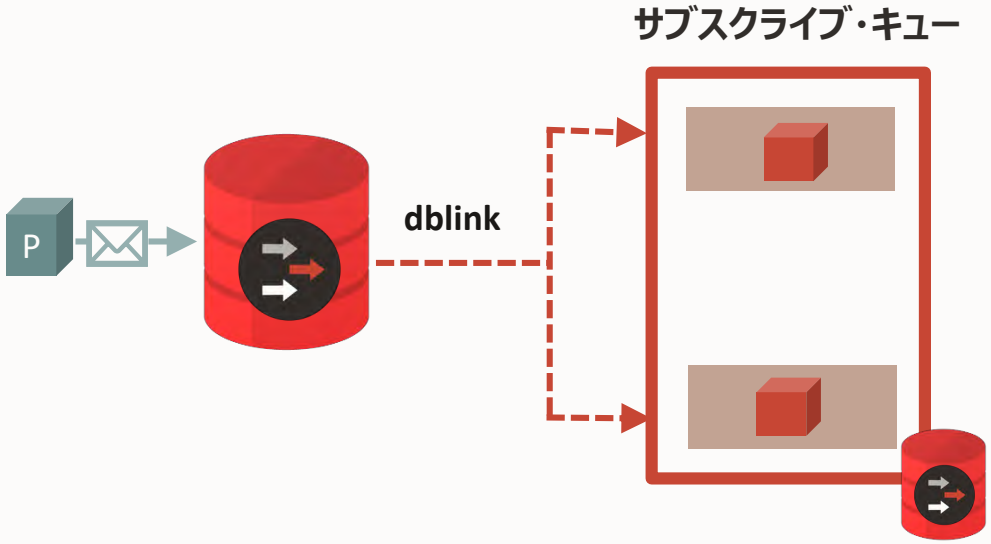


Oracle AQ / TxEventQ のメッセージ伝播
リモート・サブスクライバへの伝播

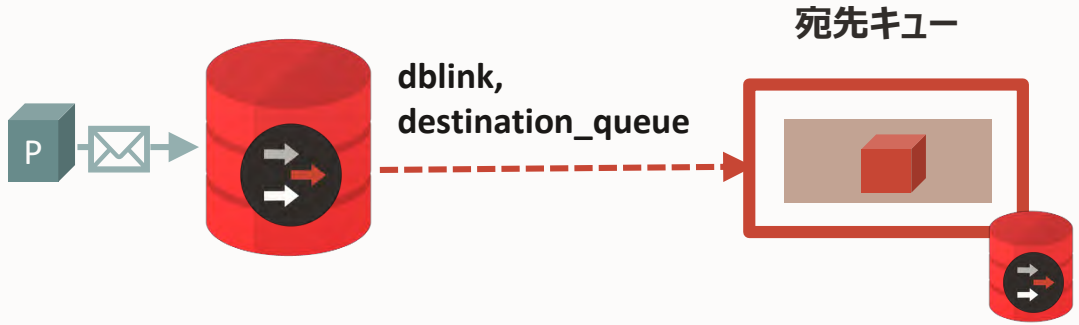
2種類の伝播

- キューからデータベース・リンクへの伝播
- キューからキューへの伝播

キューからデータベース・リンクへの伝播



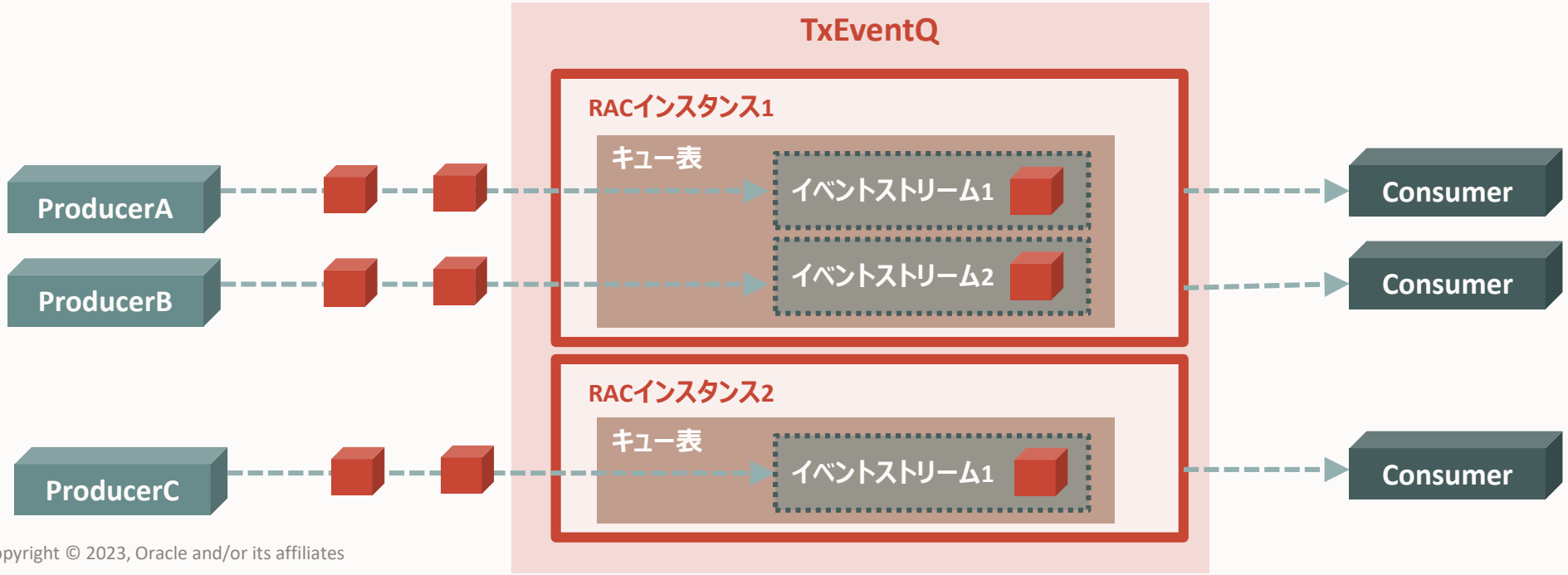
キューからキューへの伝播



イベントストリームの同時実行性が向上
イベントストリームとは

イベントストリーム（シャード）：キュー表のパーティションで構成

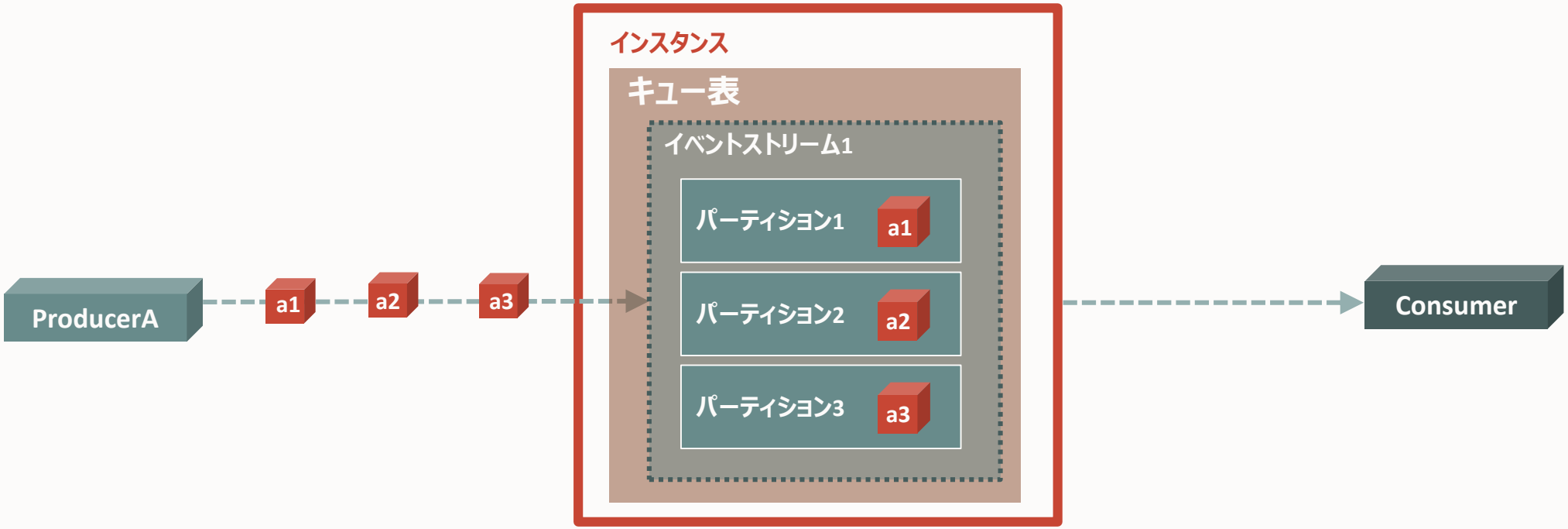
- 水平分割による高い同時実行性とスループットを実現する
- イベントストリームは自動でパーティション化
- インスタンス内にイベントストリームでエンキューセッションを分散
- インスタンス内にすべてのイベントストリームのデキュー



イベントストリームの同時実行性が向上

新機能：パーティション化されたイベントストリーム

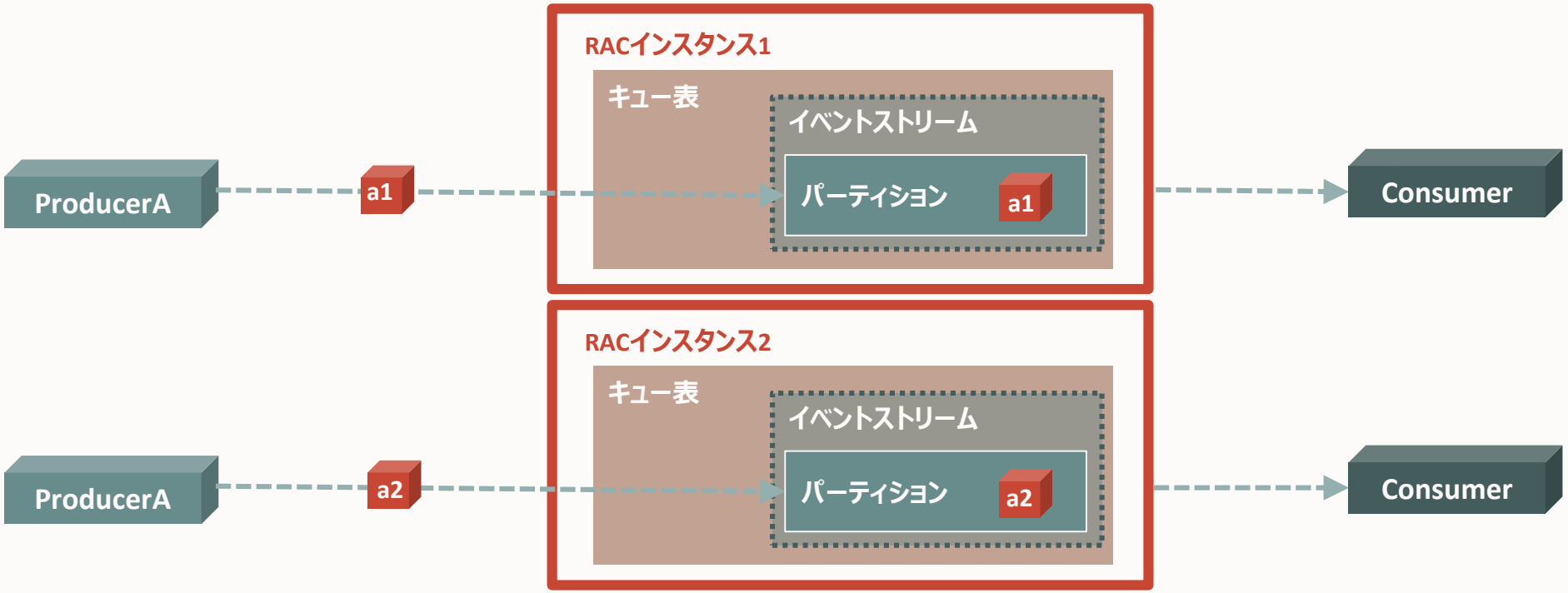
- TEQではイベントストリームがパーティション化されるように
- 新しいパーティションは、必要に応じて自動的に作成される
- パーティション内のすべてのメッセージがデキューされると、パーティションは切り捨てられ再利用される



イベントストリームの同時実行性が向上

新機能：RACを使用したときの同時実行性が向上

- パーティション化されたイベントストリームとRACインスタンスにアフィニティを持たせる
  - １つのイベントストリームの処理を１つのRACインスタンス内で完結させるようにする
  - 元々RACはサポートしていたが、新たにサポートされたパーティショニングと組み合わせて実現



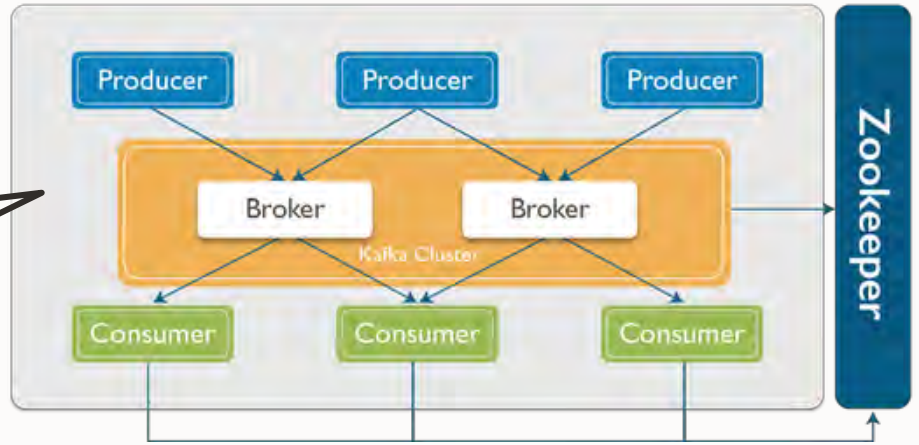
KafkaとOracle Databaseの互換性

Apache Kafkaとは？

- Apache Kafkaは、スケーラビリティに優れた分散メッセージキュー
- 広く使用され、人気のあるオープンソースのイベントストリーミングおよびメッセージングシステム
- 分散型のフォールトトレラントアーキテクチャで大量のデータを処理する機能を備える。

- Broker：Kafkaが稼働する1台のサーバー
- Kafka Cluster：Kafkaが実行されているサーバ(Broker)をグループ化したもの
- Zookeeper：Kafkaを管理するサーバ

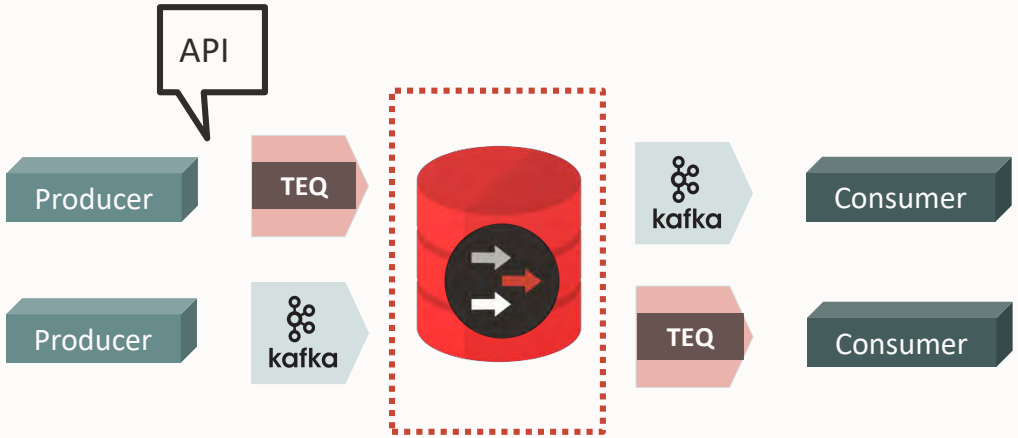
Kafka Brokers



KafkaとOracle Databaseの互換性

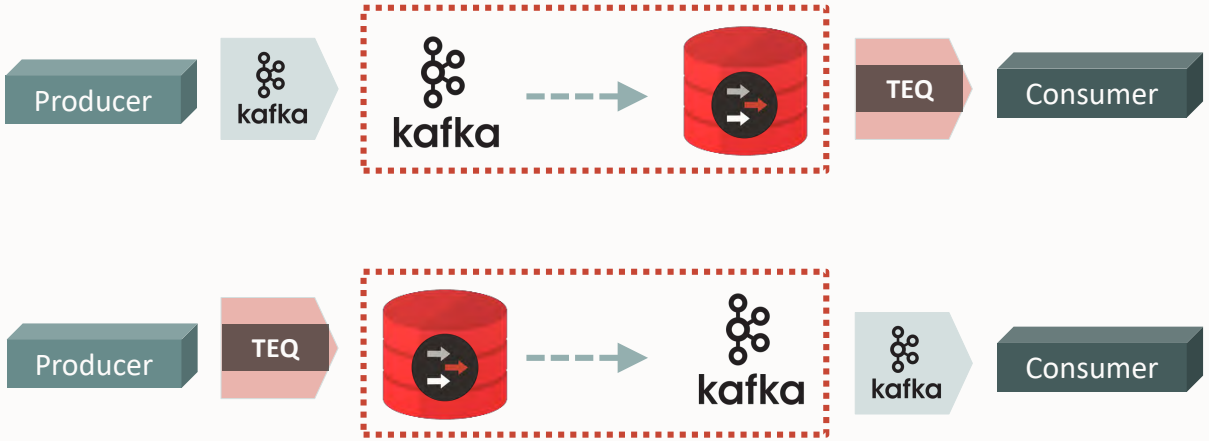
TxEventQがKafkaのインターフェースOKafkaを持つように

- Kafka APIアプリケーションとOracle APIアプリケーションがメッセージの互換性を持つ
  - Kafka Java APIはOracle Databaseサーバーに接続し、メッセージング・プラットフォームとしてTxEventQを使用可能
  - KafkaをTxEventQに置き換えての使用が可能



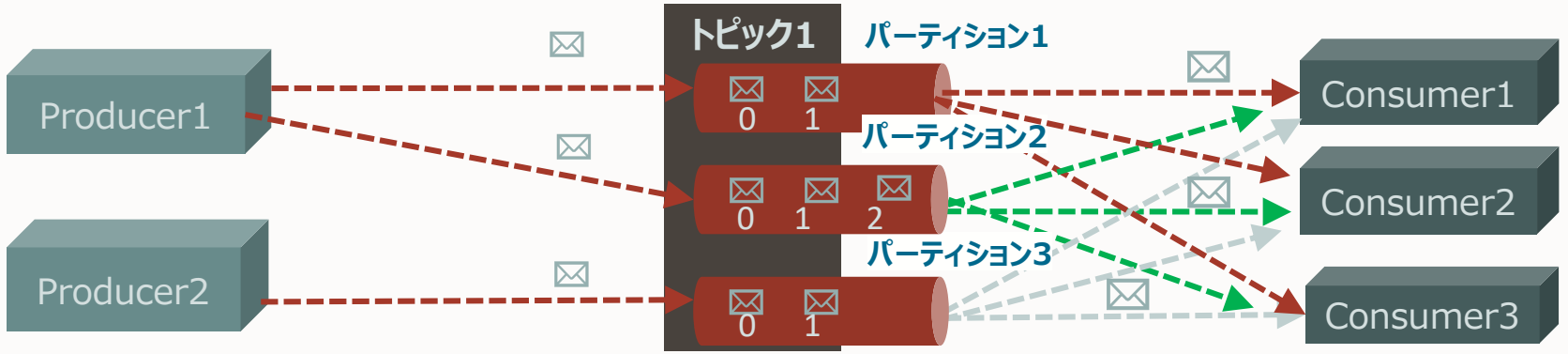
KafkaのAPIを使用してTxEventQへのエンキュー・デキューが可能に

- KafkaとTxEventQが相互にメッセージのやりとりできるように
  - Kafka Java APIを使用することでKafkaからTxEventQへ、TxEventQからKafkaへ相互にメッセージを送れるように



KafkaとTxEventQのメッセージのやり取りが透過的に

KafkaとOracle Databaseの互換性
アーキテクチャ



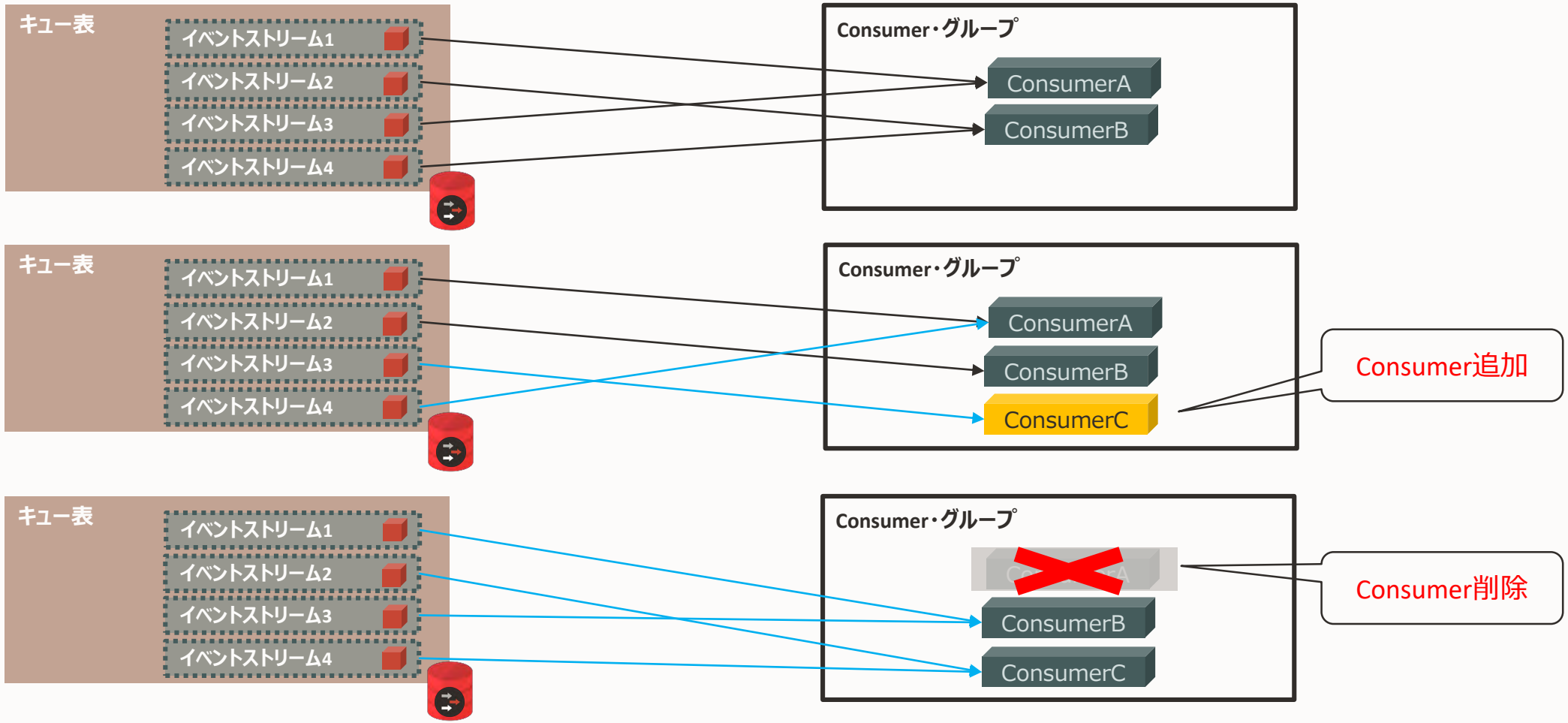
KafkaトピックおよびパーティションはTxEventQキュー表およびイベント・ストリームにマップされる

props.put("bootstrap.servers", "<kafka>:9092")

props.put("bootstrap.servers", "<db>:1521")

KafkaとOracle Databaseの互換性

Kafka Rebalancingのサポート



Kafka Javaクライアント – Producer

```
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import java.util.Properties;
public class SimpleProducerOKafka
{
  public static void main(String[] args) {
    try {
      Properties props = new Properties();
      props.put("bootstrap.servers", "kafka:9092");
      props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
      props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
      Producer<String, String> producer = new KafkaProducer<String, String>(props);
      Future<RecordMetadata> lastFuture = null; int msgCnt = 20000;
      for(int i=0;i<msgCnt;i++) {
        ProducerRecord<String, String> producerRecord = new ProducerRecord<String, String>("TxEventQ", i+"", "Test message # "
+i);
        lastFuture = producer.send(producerRecord);
      }
      System.out.println("Produced "+ msgCnt +" messages."); lastFuture.get(); producer.close();
    }
    catch(Exception e) {
      System.out.println("Exception in Main " + e );
      e.printStackTrace();
    }
  }
}
```

org.apache.kafka.clients.producer
パッケージよりKafkaProducerクラスをimport

Kafkaクラスタへの接続情報

TxEventQのKafka Javaクライアント– Producer

```
import org.oracle.okafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import java.util.Properties;
public class SimpleProducerOKafka
{
  public static void main(String[] args) {
    try {
      Properties props = new Properties();
      props.put("bootstrap.servers", "database:1521");
      props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
      props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
      Producer<String, String> producer = new KafkaProducer<String, String>(props);
      Future<RecordMetadata> lastFuture = null; int msgCnt = 20000;
      for(int i=0;i<msgCnt;i++) {
        ProducerRecord<String, String> producerRecord = new ProducerRecord<String, String>("TxEventQ", i+"", "Test message # " +i);
        lastFuture = producer.send(producerRecord);
      }
      System.out.println("Produced "+ msgCnt +" messages."); lastFuture.get(); producer.close();
    }
    catch(Exception e) {
      System.out.println("Exception in Main " + e );
      e.printStackTrace();
    }
  }
}
```

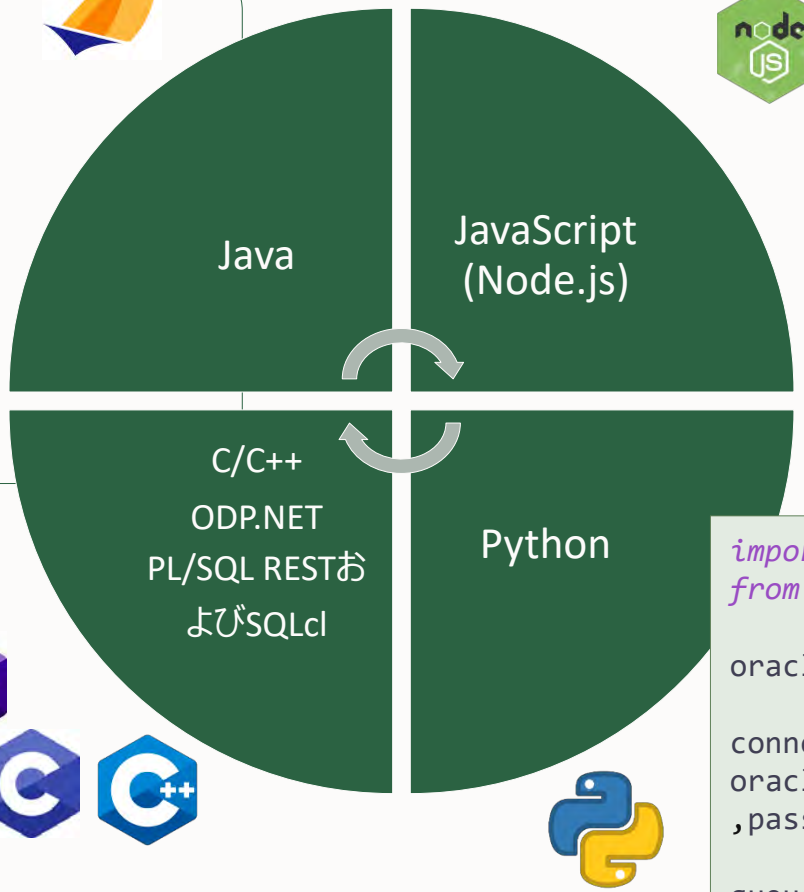
このインポートを**TxEventQ KafkaProducer**に変更

Oracle Databaseをブートストラップ・サーバー・リストに追加

コードの残りの部分の変更は必要なし！

多くの言語のサポート

- 複数のConsumerおよび遅延送信を含むJMS P2PおよびPub/Sub
- Spring JMSサポート用のSpring Bootスターター
- KafkaブローカをメッセージングブローカとしてTxEventQに置き換えるためのKafka Javaクライアントのサポート(Okafka)



Node.js

```
// get a connection to the database
oracledb.initOracleClient({});
connection = await
    oracledb.getConnection({
        user: 'pdbadmin',
        password: '********',
        connectString: 'db:1521/pdb1'
    })

// enqueue a message
const rawQueue = await
    connection.getQueue("my_TxEventQ");
await rawQueue.enqOne("Hi Mom!");
await connection.commit();
```

Python

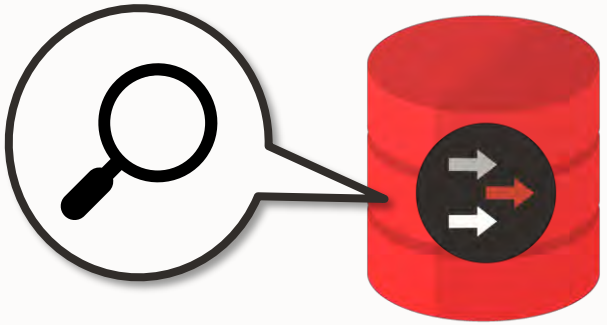
```
import oracledb
from os import environ as env

oracledb.init_oracle_client()

connection =
oracledb.connect(dsn='172.17.0.2:1521/pdb1',user='mark'
,password='******')

queue = connection.queue("myq", "JSON")

queue.enqOne(connection.msgproperties(payload={"develop
er day": "today"}))
connection.commit()
```

リアルタイムなパフォーマンス監視

TxEventQでモニタリング可能な要素

- TxEventQのパフォーマンス監視でモニタリングできる内容は次の通り：
  - ✓メッセージング・システムの正常性の確認
  - ✓エンキュー/デキューのスループット、キューの深さを含む、全体的な主要パフォーマンス指標の監視
  - ✓メッセージング・アクティビティに由来するCPUの負荷、メモリーの使用状況、およびデータベース待機クラスの監視
  - ✓各キューの正常性状態を確認して、パフォーマンスの低いキューを素早く特定

- TxEventQのパフォーマンス監視は、データベースの動的パフォーマンス・ビューから情報を取得
- キューの主要メトリックに関するレポートの出力は各種ユーザー・インタフェースと統合可能
- オープン・ソース・ツールPrometheusおよびGrafanaを使用してメトリックの監視が可能



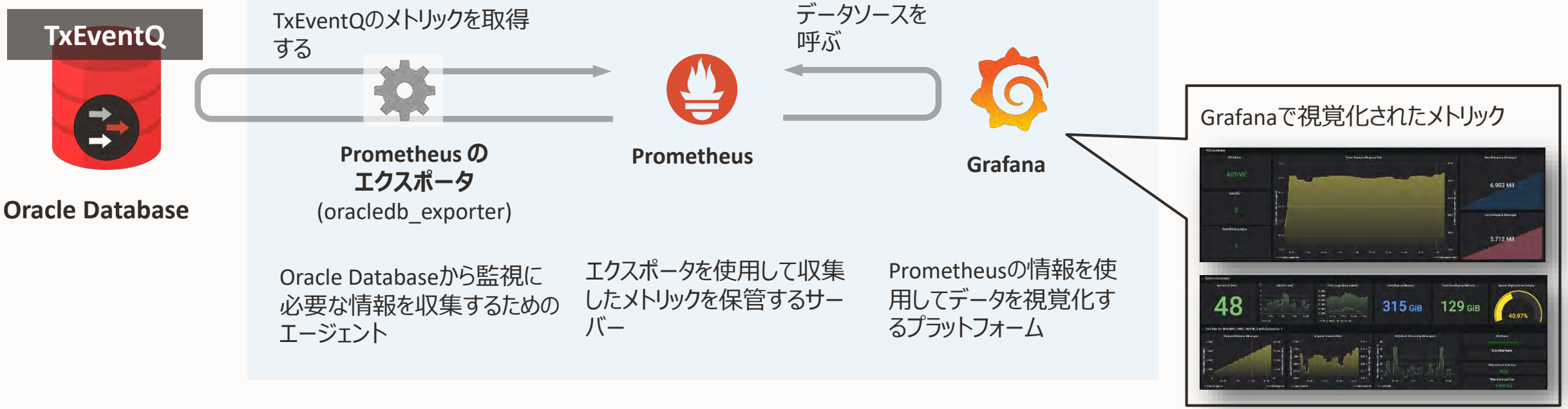
リアルタイムなパフォーマンス監視

TxEventQモニタ・システムの構成

- TxEventQのメトリックを視覚的に取得可能

次のようなメトリック情報を取得：
- ✓ ステータス
- ✓ キューの数
- ✓ サブスクライバの数
- ✓ エンキュー/デキューのスループット
- ✓ メッセージの数

dockerコンテナ上で動作



リアルタイムなパフォーマンス監視

Grafanaの使用によりサマリーを視覚的に確認することが可能に

TxEventQモニターではインスタンス・キュー・サブスクライバ・ディスク・グループを対象に次のサマリーを確認することが可能：

- すべてのTxEventQ全体のサマリー
- データベース・メトリックのサマリー
- システム・メトリックのサマリー
- TxEventQごとのサブスクライバのサマリー

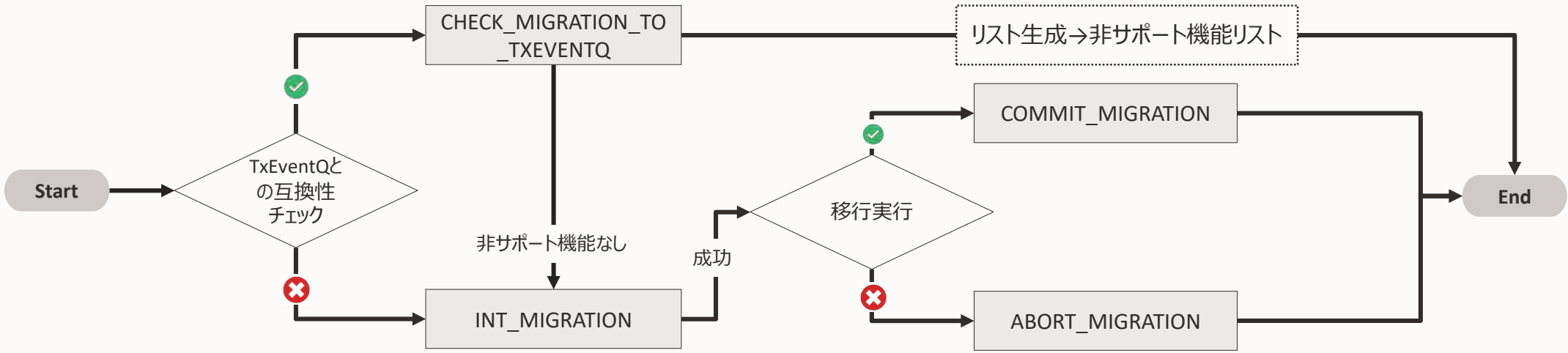
▶ データベース・サマリー・ダッシュボード：
全体的なDBのパフォーマンスと統計情報

▲ システム・サマリー・ダッシュボード：
システム・レベルのメトリックとキュー・レベルのメトリックを表示
（CPU使用率と使用メモリーなど）

▲ TxEventQサマリー・ダッシュボード：
全体的に集計されたTxEventQの統計情報(ステータス、キューの数、サブスクライバの数、エンキュー率/デキュー率、メッセージの数など)

AQ から TxEventQ へのオンライン移行ツール

- 移行ステップを自動化するPL/SQLパッケージDBMS_AQMIGTOOLの提供
- 機能
  - AQの定義とデータをチェックし、移行の許可/不許可、適応可能かどうかのレポートと推奨事項の表示
  - 要件に応じた移行モードが選択可能：AUTOMATIC/INTERACTIVE/OFFLINE/ONLY_DEFINITION
  - 移行のコミット/フォールバックが選択可能
  - キューの移行履歴の確認



AQシャード・キューからTxEventQへ

AQ シャード・キューから進化した新しいメッセージ・キューイング・システム

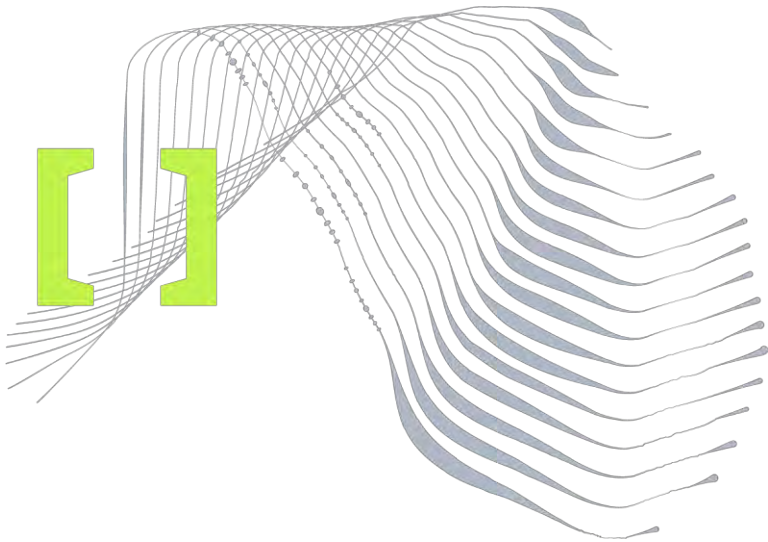
23c新機能

- 伝播機能
- イベントストリームの同時実行性が向上
- TxEventQ用のKafka Javaクライアント/Kafka実装の拡張
- 多くの言語のサポート
- リアルタイムなパフォーマンス監視
- AQからTxEventQへのオンライン移行ツール

Oracle Database Sagas (OSaga)

Agenda

1. Sagaパターンとは
2. Oracle DatabaseのSaga実装



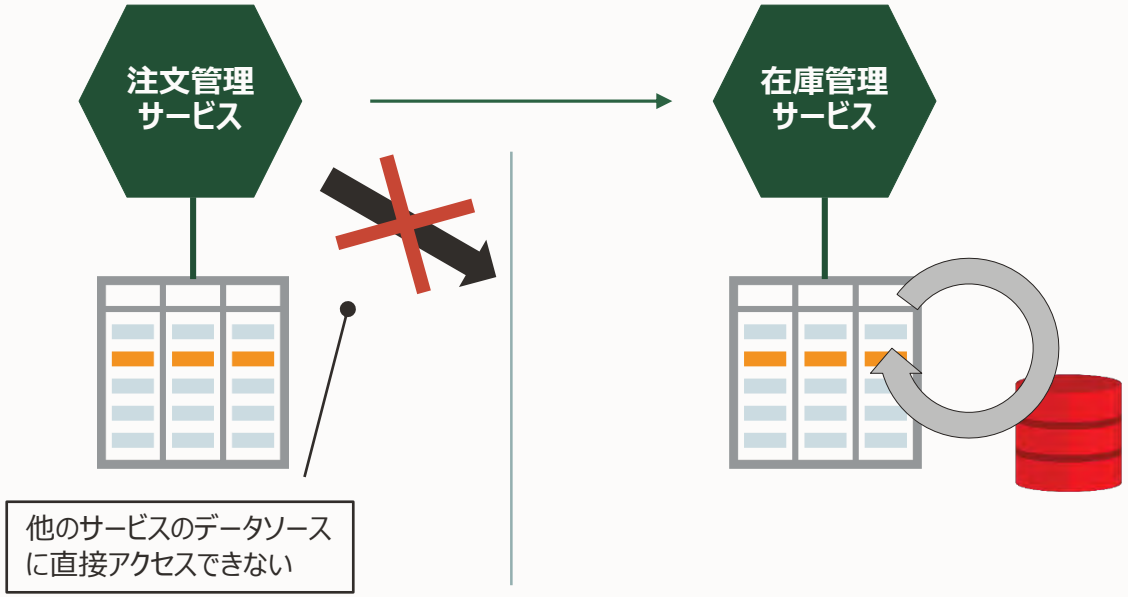
Sagaパターンとは

サービス間をまたがるトランザクションの結果整合性の担保

マイクロサービスにおける一貫性の考え方
結果整合性による一貫性

相手のサービスのデータソースに直接アクセスできない

- トランザクションによる一貫性が利用できない
- 自サービスと相手のサービスそれぞれの「結果整合性」により全体の一貫性を保つ



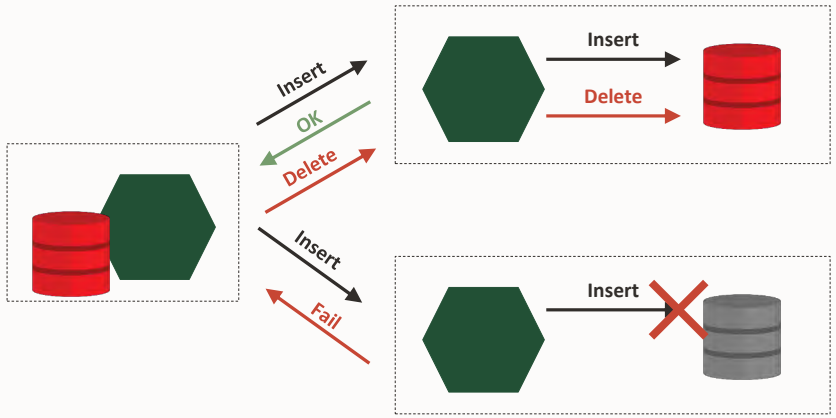
マイクロサービスで検討される代表的な一貫性の仕組み

サービス間をまたがるトランザクションの仕組み

Sagaパターン

補償トランザクションによる(事後)結果整合性

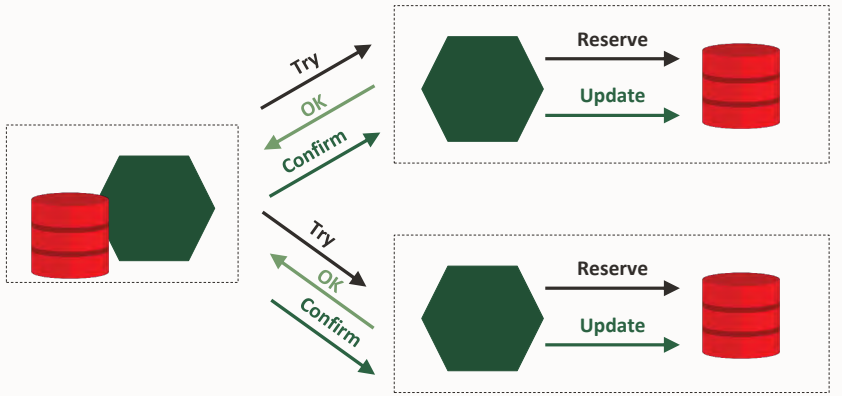
- 処理の成功を前提とした楽観的な呼出し手法
- 一部の処理が失敗した場合に、既に完了した処理を取り消す「補償トランザクション」により整合性を取る



TCC (Try/Confirm/Cancel) パターン

予約ベースの(事前)結果整合性

- 処理完了の可否を事前に確認した上で処理を実施
- Tryフェーズにより不整合の生じる処理を行わないことで整合性を担保



Sagaパターンとは

サービス間をまたがるトランザクションの仕組み

実装方法

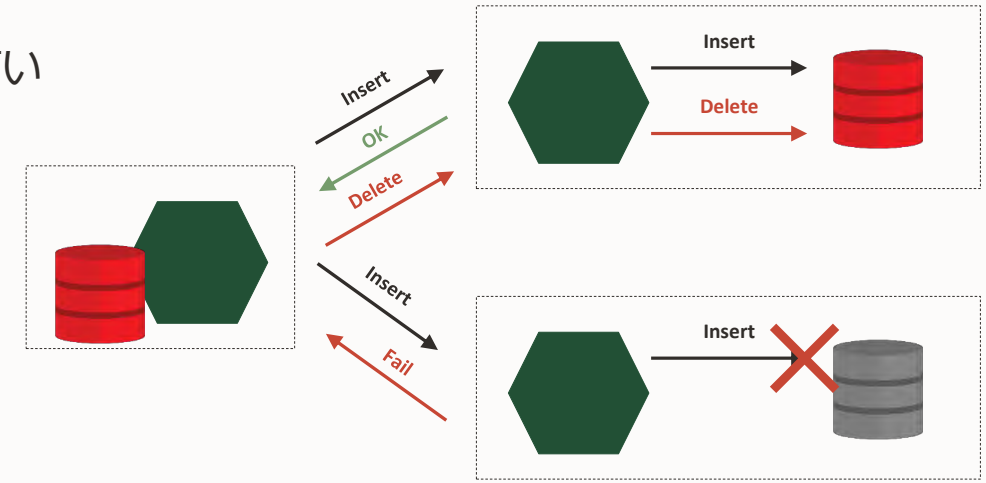
- 1つのビジネストランザクション(注文処理など)を”Saga”という一連のトランザクションで実装
- 各サービスのデータアクセスは、それぞれのローカル・トランザクションで実施

向いているトランザクション制御の特性

- トランザクションの存続期間が長く、1 つのマイクロサービスが長時間実行されても、他のマイクロサービスがブロックされることは避けたい
- ワークフロー内の操作が失敗した場合にロールバックできる必要がある
- 他のトランザクションと分離されやすく、補償処理に人手が関与しやすい

ユースケース

- ユーザー単位で行われる受付業務
- カート→チェックアウト→配送のような長期的なトランザクション

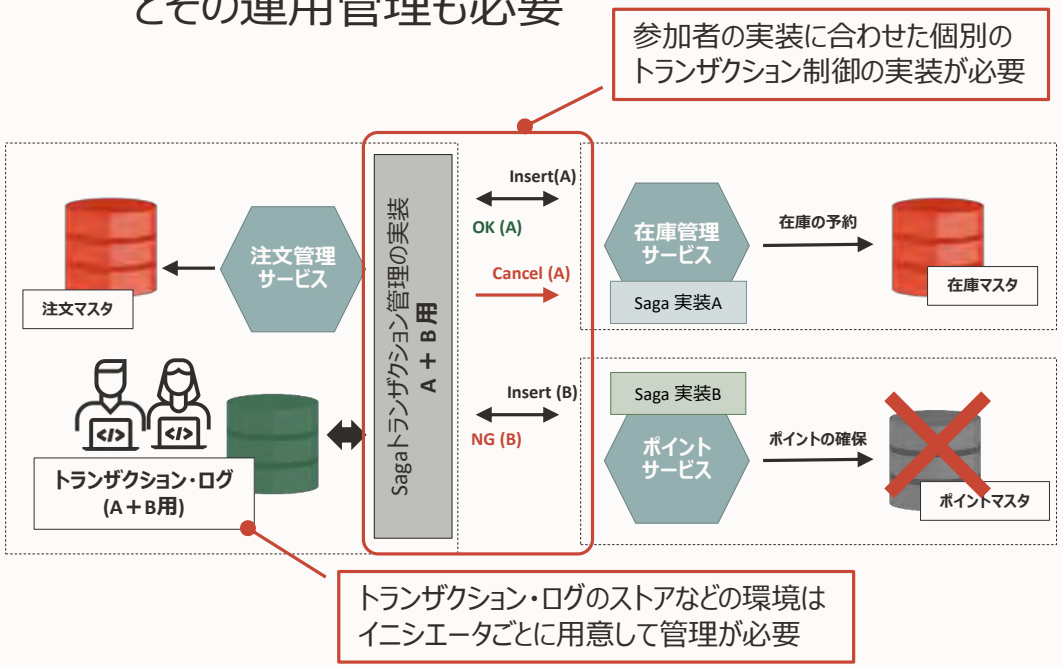


トランザクション・コーディネータの必要性

コーディネータが参加者のやり取りを仲介し、アプリケーション実装コストを軽減し、管理性を向上させる

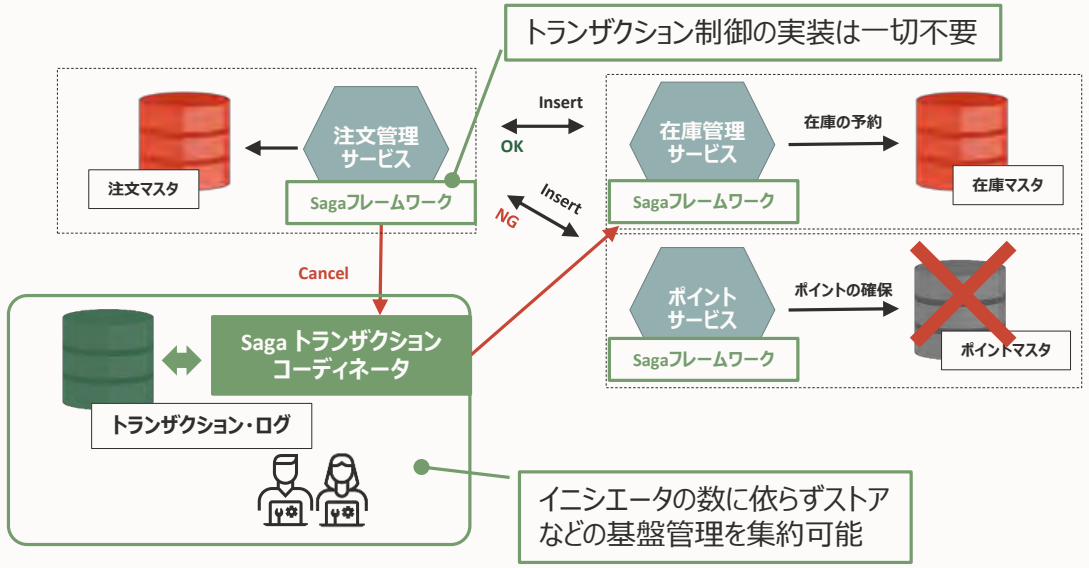
【トランザクション管理の独自実装による課題】

- イニシエータ側でトランザクション制御の実装が必要
- イニシエータ側でトランザクション・ログの構成が必要
  - トランザクション制御の実装ごとに、ログ用のストアとその運用管理も必要



【コーディネータによるトランザクション管理の集約の効果】

- トランザクション管理を共通フレームワーク化
  - イニシエータや参加者による**トランザクション制御実装が不要**
- トランザクション管理をイニシエータから分離して運用
  - ログ用**ストアなどの基盤管理を集約**して効率化

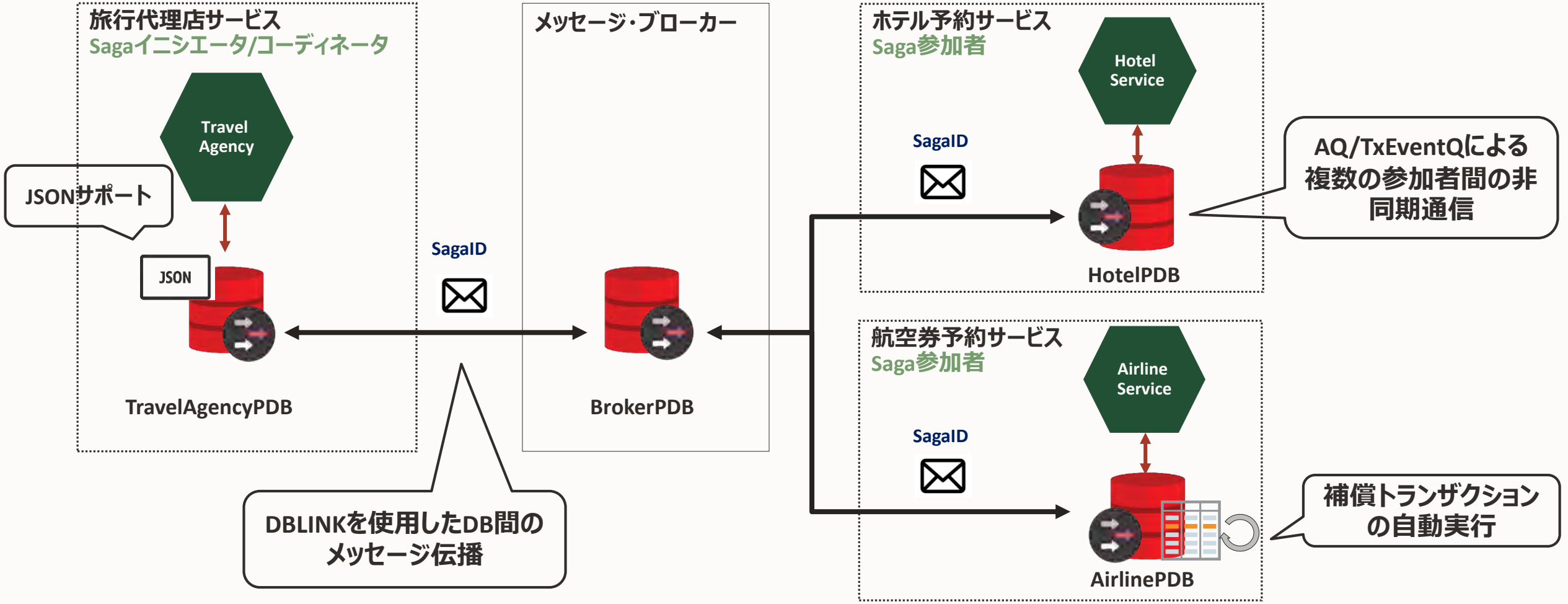


Oracle DatabaseのSaga実装

データベース機能を利用したOSaga

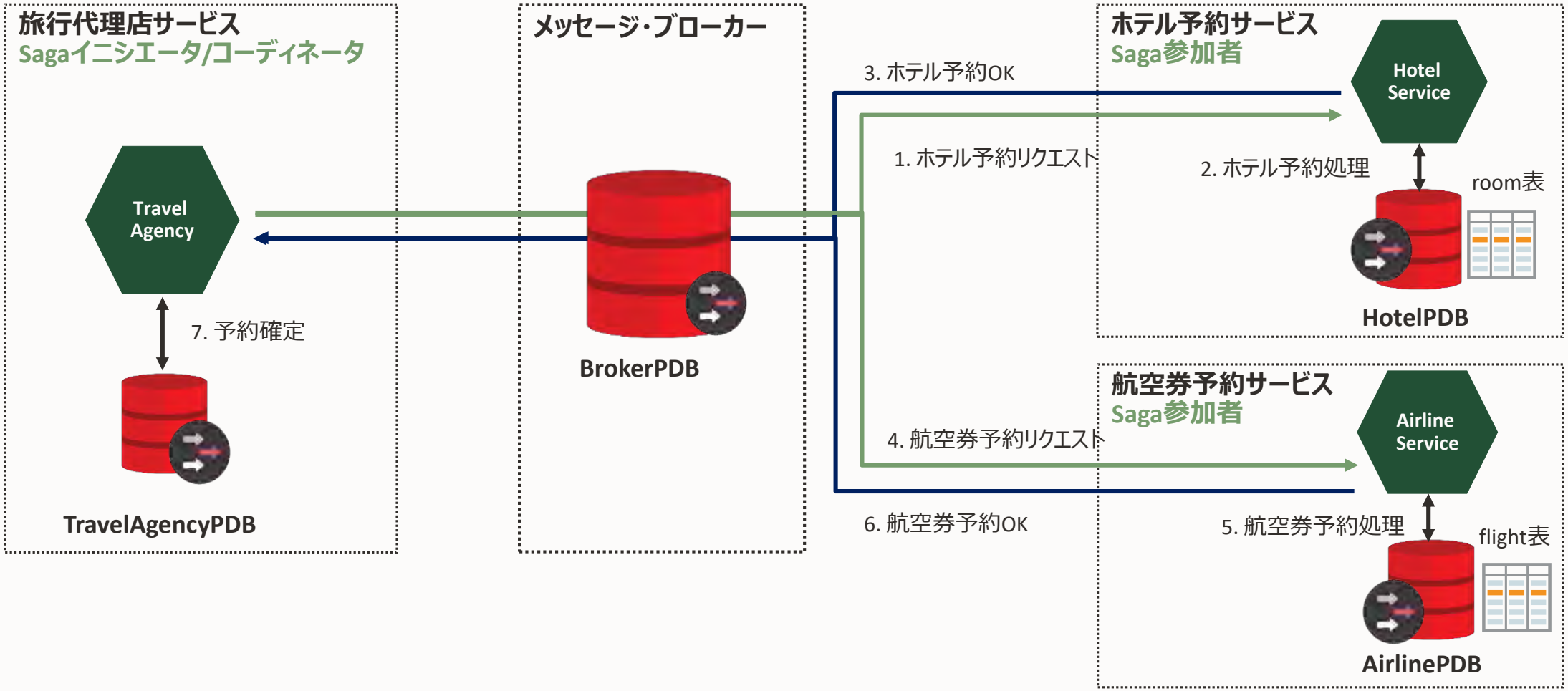
Oracle Databaseを使用したSagaの実装

組み込みコーディネータによるシンプルで堅牢性の高いアーキテクチャ



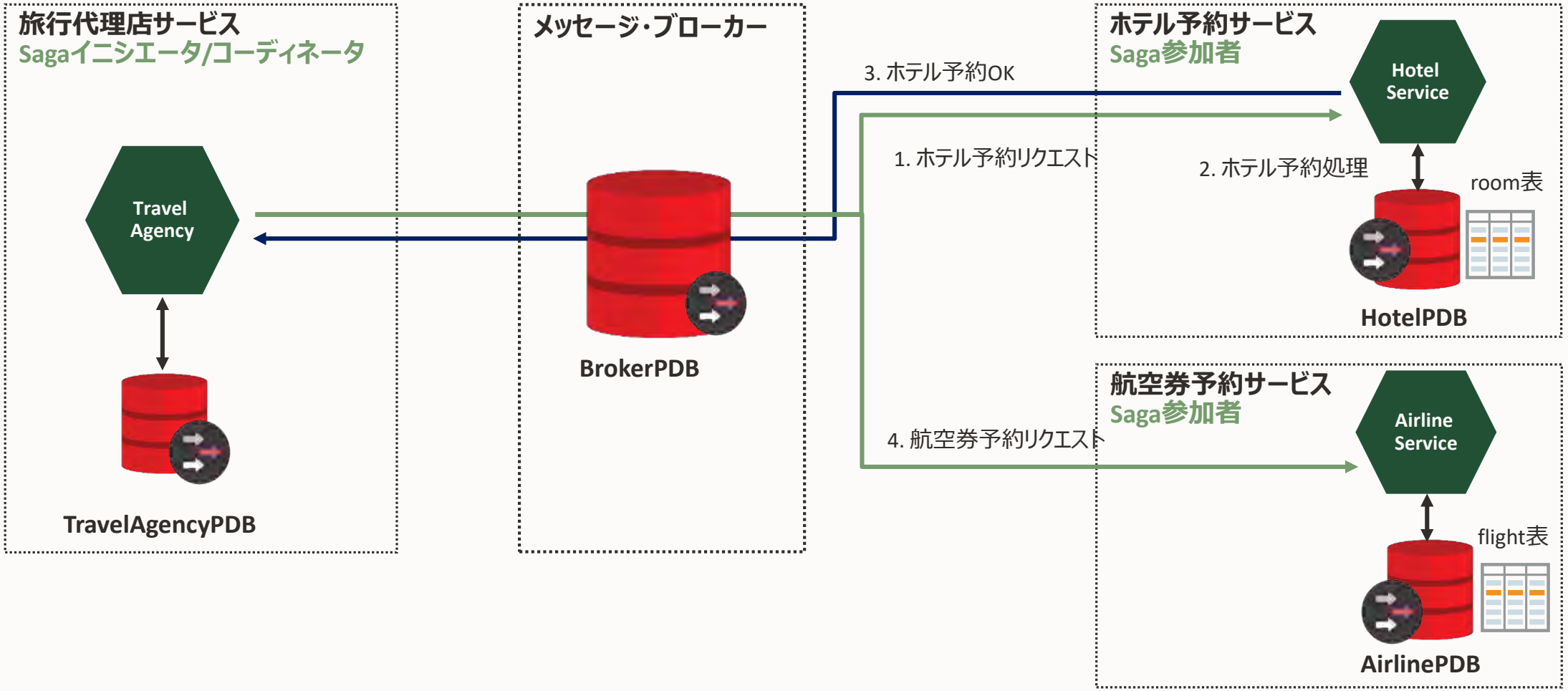
Oracle Databaseを使用したSagaの実装(正常パターン)

組み込みコーディネータによるシンプルで堅牢性の高いアーキテクチャ



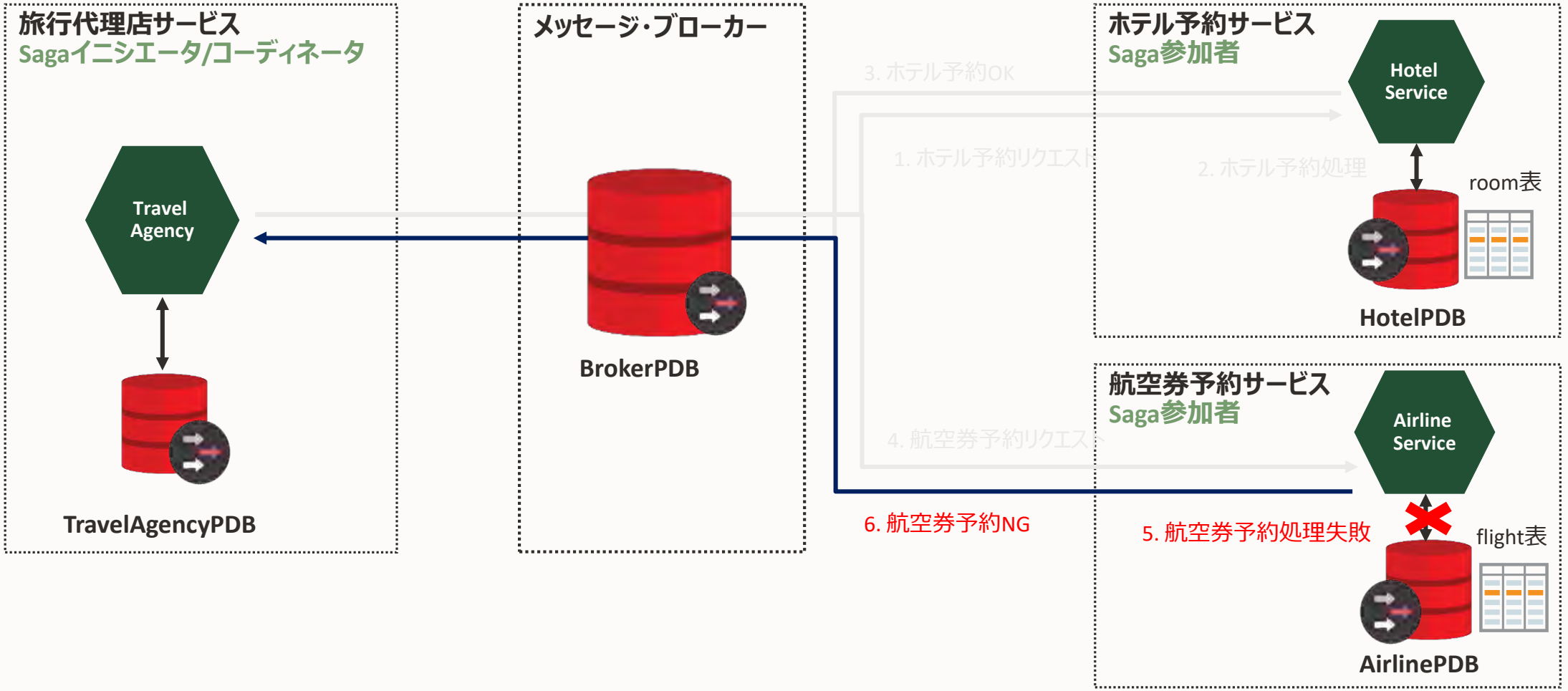
Oracle Databaseを使用したSagaの実装(異常パターン)

組み込みコーディネータによるシンプルで堅牢性の高いアーキテクチャ



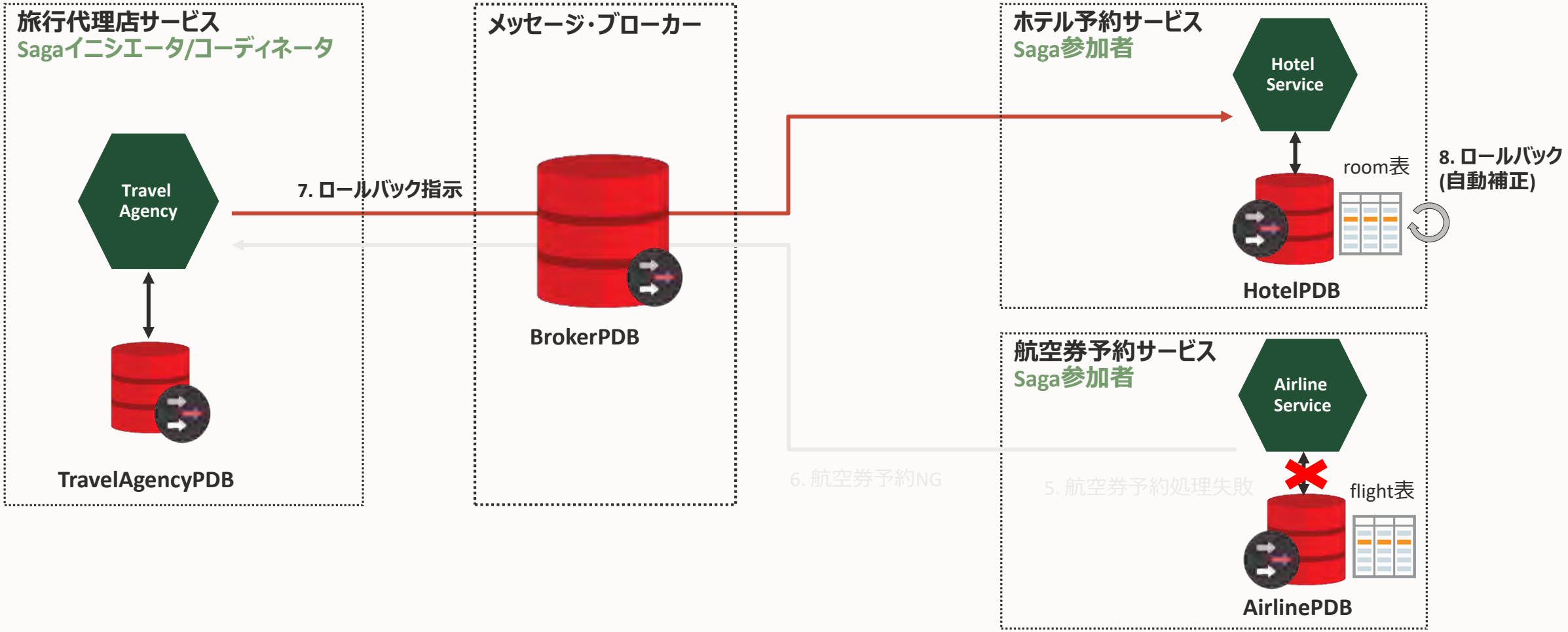
Oracle Databaseを使用したSagaの実装(異常パターン)

組み込みコーディネータによるシンプルで堅牢性の高いアーキテクチャ



Oracle Databaseを使用したSagaの実装(異常パターン)

組み込みコーディネータによるシンプルで堅牢性の高いアーキテクチャ



Oracle Databaseを使用したSagaの実装
サンプル・コード

OSaga サポートなし

```
// スタートアップ・ロジック: タイムアウトの設定やリカバリ処理、Sagaレコード
// のログを維持する仕組みが必要
  readRecoveryLogs(connection);
  // リカバリ処理
  startRecoveryProcessing(connection);
  startExpirationProcessing(connection);

// 個別のSagaロジック実装
  String sagaId = beginSaga(connection);
  persistSagaRecord(connection, sagaId);
  persistFlightParticipantRecord(connection, sagaId);
  reserveFlight(connection, sagaId);
  persistHotelParticipantRecord(connection, sagaId);
  reserveHotel(connection, sagaId);

// Sagaイベントの送信と状態変化を1つのローカル・トランザクションで実装
  if(isCommit){
    connection.setAutoCommit(false);
    sendCommitEventToAllParticipants(connection, sagaId);
    setStateToCommittedOnParticipants(connection, sagaId);
    purgeSagaAndParticipantEntries(connection, sagaId);
    connection.commit();
  } else {
    …
```

OSaga サポートあり

```
// スタートアップ・ロジックは不要

// 個別のSagaロジック実装
beginOSaga(); //Osaga.beginSaga();
reserveFlight(); //
reserveHotel(); //

if(isCommit) commitSaga(); //Osaga.commitSaga();
else rollbackSaga(); //Osaga.rollbackSaga();
```

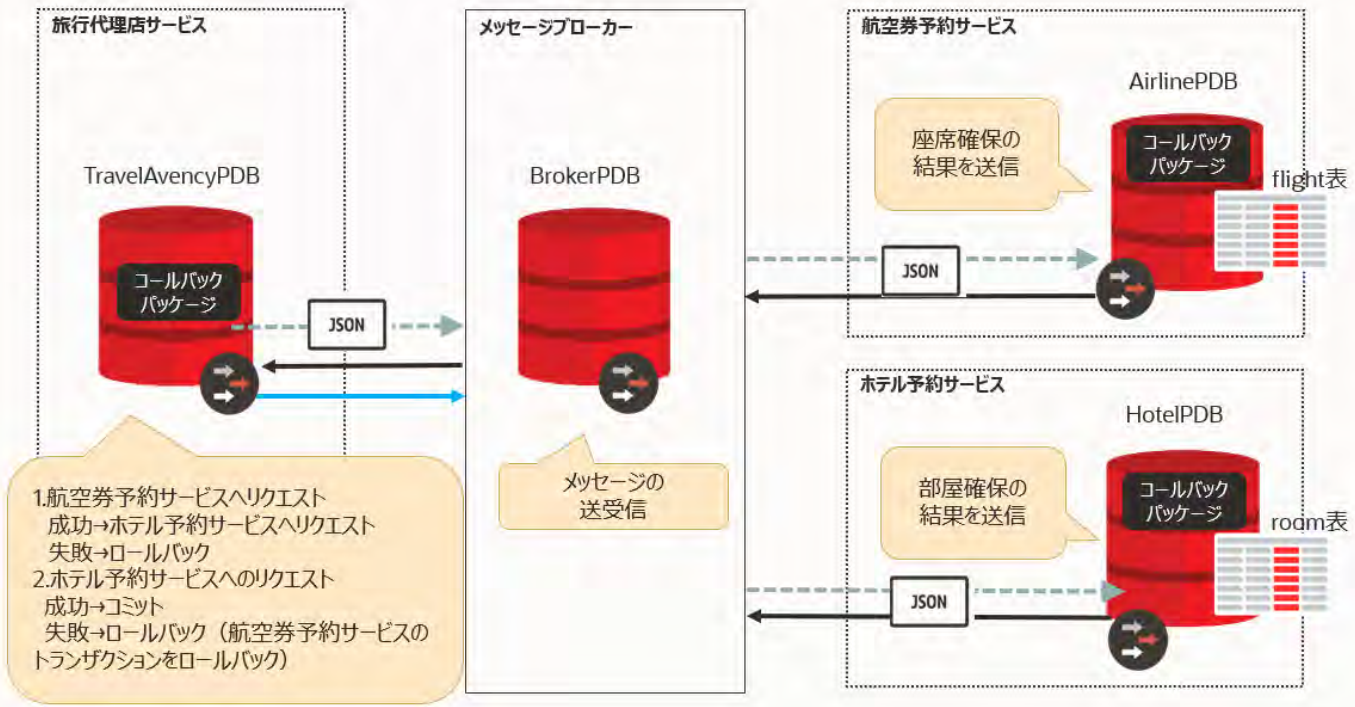
Oracle Databaseを使用したSagaの実装
データベース・トランザクションの優れた結果整合性の提供

- Saga実装はデータベースに統合
  - Sagaのコード化、デプロイ、メンテナンスが容易
  - Saga中断時のロールバック処理を行う組み込み済の自動補正ロジック(ロック・フリー列値の予約機能)
  - Oracle Databaseのスケーラビリティ、高可用性、一貫性、堅牢性の活用
  - Oracle Multitenantによる一元的なDB管理
  - ディクショナリ表やビューですべてのSaga参加者の一元的な状態管理が可能
- アプリ開発者のハードル低減
  - 障害時のリカバリ・ロジックの記述が不要
  - 組み込み済のAQおよびTxEventQにより、メッセージおよびイベントの生成が可能 – Kafkaコーディネータ不要
  - より高レベルのAPIおよびSpring Bootとの統合
  - Javaアノテーションの使用 – 既存コードの流用が可能
- コンバージド・データベースの活用
  - アプリケーション・ペイロードのJSONサポート

参考：Oracle Databaseを使用したSagaの実装サンプル(PL/SQL)

[23c Oracle Saga Frameworkを試してみた その1：設定編]

[23c Oracle Saga Frameworkを試してみた その2：実行編]

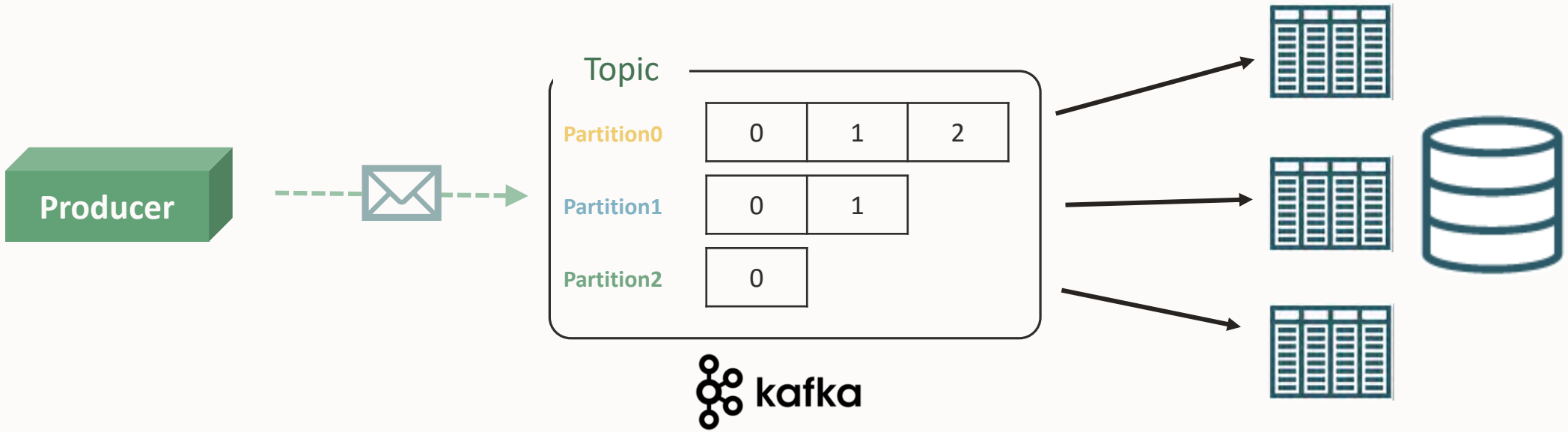


Oracle SQL Access to Kafka (OSaK)

Oracle SQL access to Kafka (OSaK) とは

Oracle SQL APIを使用してKafkaトピックに動的にクエリを実行できる

- Oracle Database 23cからのネイティブ機能
- SQL構文でKafkaストリーミングデータを処理
- Oracle DatabaseにKafkaアクセスが統合されているため、外部クライアントが不要
- DBMS_KAFKA および DBMS_KAFKA_ADMパッケージで構成される



Oracle SQL access to Kafka (OSaK) のメリット

Oracle SQL APIを使用してKafkaトピックに動的にクエリを実行

KafkaデータとOracle Database内のデータを組み合わせたデータ分析

- データベースに保存せずに一時的な利用が可能

データ処理はOracle Databaseのトランザクションとして実行・制御

- トランザクションはデータベースのACID(原子性・一貫性・独立性・永続性） 要件に準拠
- トランザクションでオフセットを管理することで、システム障害時のKafkaレコードの消失やアプリケーションによる再処理を防ぐ（Kafkaデータの分離と耐久性の向上）

アプリ開発者でなくともストリーミング・データの活用が可能

- データ処理はSQL, PL/SQLで記述

Oracle SQL access to Kafka (OSaK) の機能

Oracle SQL APIを使用してKafkaトピックに動的にクエリを実行

3つのデータ・アクセス・モード

- ロード：Kafkaトピックのデータをデータベースのテーブルにロードし、様々なアプリからアクセス可能にする。主にDWH用途。
- ストリーミング：Kafkaレコードを順番に1度だけSQLやPL/SQLを使って処理する。
- シーカブル：指定した開始と終了のタイムスタンプ間のKafkaレコードにアクセスする。過去の時点のデータをまとめて取得する用途。

OSaKがサポートしているデータ形式

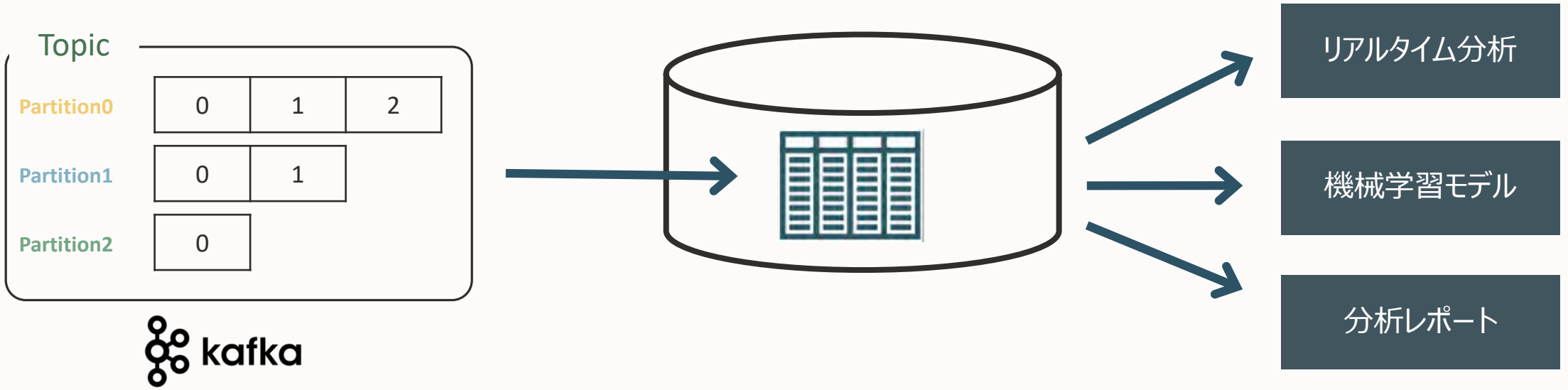
- 区切りテキストデータ(csvなど)
- JSON
- Avro

Oracle SQL access to Kafka (OSaK) の機能

Oracle SQL APIを使用してKafkaトピックに動的にクエリを実行

3つのデータ・アクセス・モード

- ロード：Kafkaトピックのデータをデータベースのテーブルにロードし、様々なアプリからアクセス可能にする。主にDWH用途。

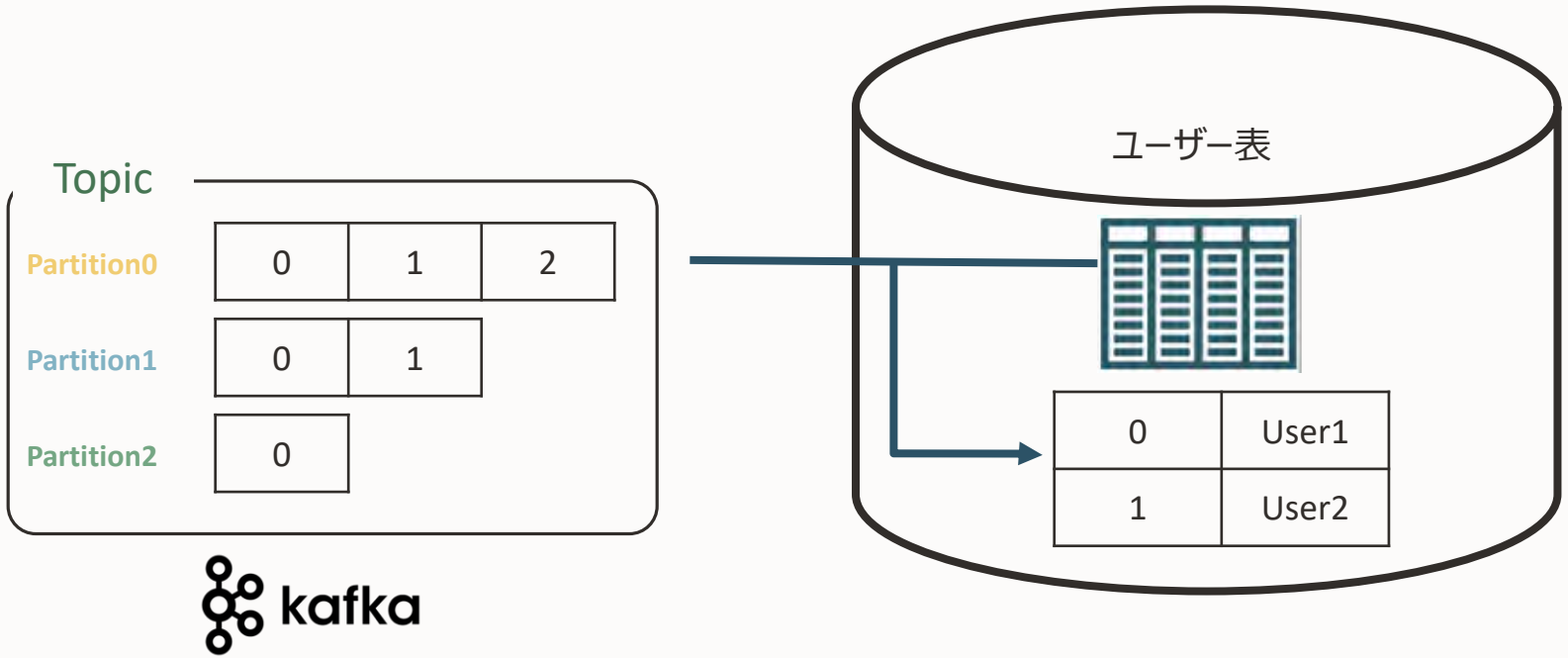


Oracle SQL access to Kafka (OSaK) の機能

Oracle SQL APIを使用してKafkaトピックに動的にクエリを実行

3つのデータ・アクセス・モード

- ストリーミング：Kafkaレコードを順番に1度だけSQLやPL/SQLを使って処理する。

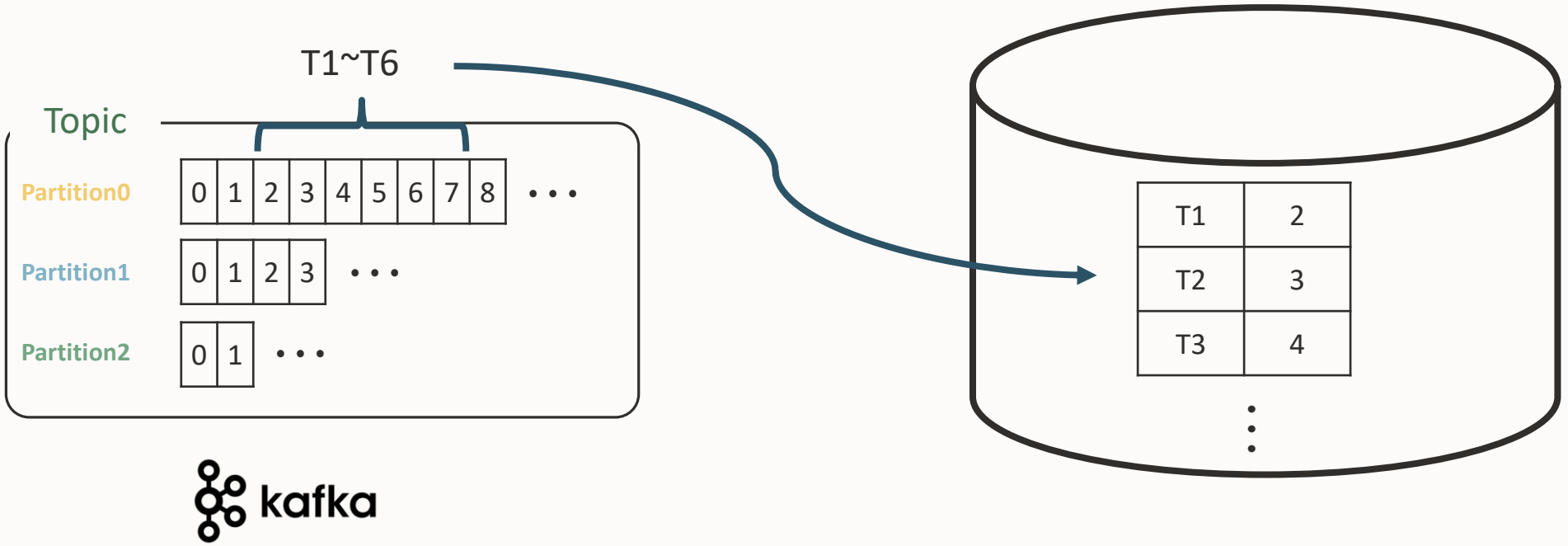


Oracle SQL access to Kafka (OSaK) の機能

Oracle SQL APIを使用してKafkaトピックに動的にクエリを実行

3つのデータ・アクセス・モード

- シーカブル：指定した開始と終了のタイムスタンプ間のKafkaレコードにアクセスする。過去の時点のデータをまとめて取得する用途。



ありがとうございました