SRA OSS

今日から始める Kafka Streams

2025/1/25

OSC 2025 Osaka

株式会社 SRA OSS

赤松 俊弘

Who am I

- 赤松 俊弘（Toshihiro Akamatsu）
  - 株式会社SRA OSS
    - OSS事業本部 技術部 基盤技術グループ
    - プリンシパルエンジニア
  - 担当
    - Zabbix を中心に約 10 年
    - 今年度から Kafka に従事
  - 趣味
    - 登山、日本酒

会社紹介

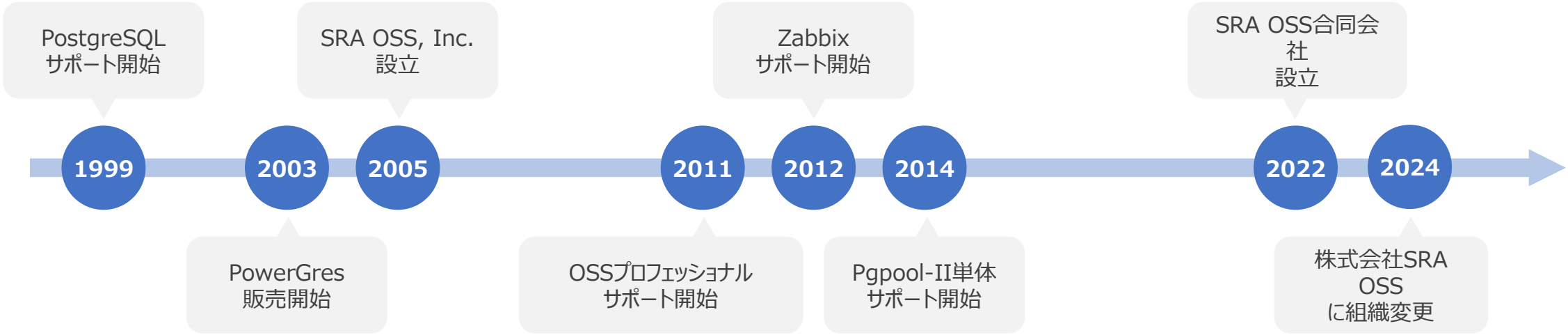
株式会社SRA OSS

所 在 地: 東京都豊島区南池袋2-32-8
設 立 日: 2022年6月17日
株　　主 : 株式会社SRA
株式会社NTTデータグループ
資 本 金: 7,000万円
社　　長: 稲葉 香理

事業内容

- オープンソースソフトウェア (OSS) 関連のサポート、製品開発・販売、構築・コンサル
- OSSの教育、開発、コミュニティ運営支援
- ソフトウェアの研究開発

顧　　問: 石井 達夫
技術顧問: 増永 良文 (お茶の水女子大学名誉教授)



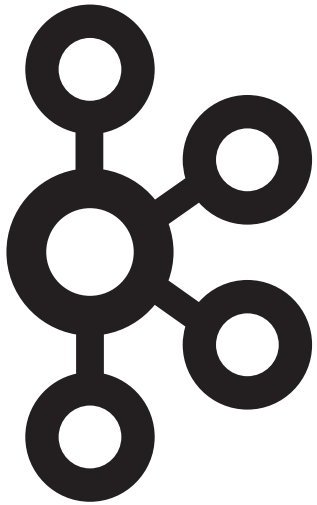
アジェンダ

- Apache Kafka の概要
- Kafka Streams とは
- Kafka Streams のコンセプト
- Kafka Streams の並列分散処理と耐障害性
- Kafka Streams 実装例
- まとめ

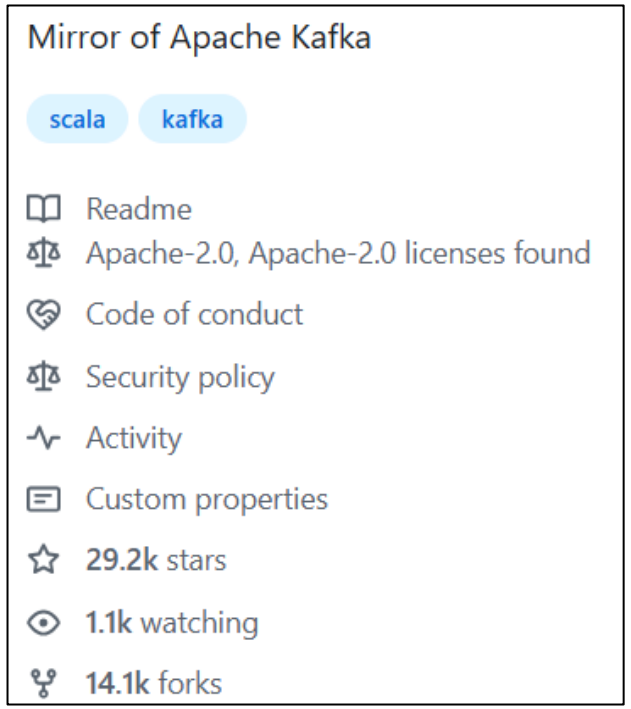
Apache Kafka の概要

Apache Kafkaとは

- Apache Kafka（以下Kafka）はストリーミング処理を実現するうえで欠かすことのできない、オープンソースの分散メッセージングシステム
- 元々は LinkedIn によって開発され、2011 年オープンソース化
- 2012 年から Apache Software Foundation (ASF) の TopLevel Project



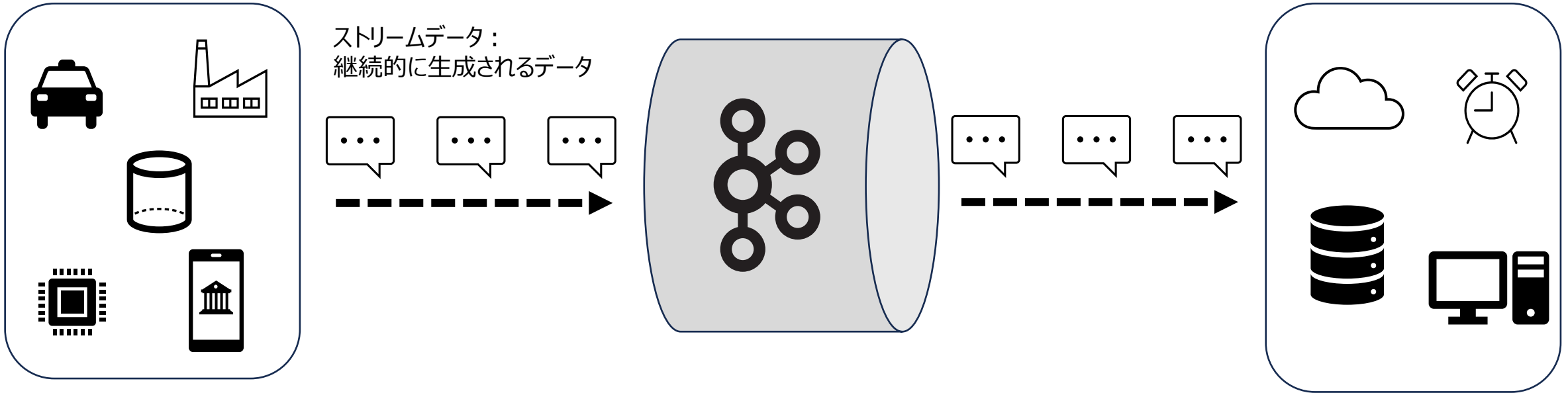
2025/01/15 時点



ストリーム処理について

Kafkaが活用される処理として、ストリーム処理という考え方がある
従来主流だったバッチ処理に比べ、到着したデータを逐次処理

ストリーム処理のデータの送信先として Kafka を利用することで、
データの取り回しが容易になる



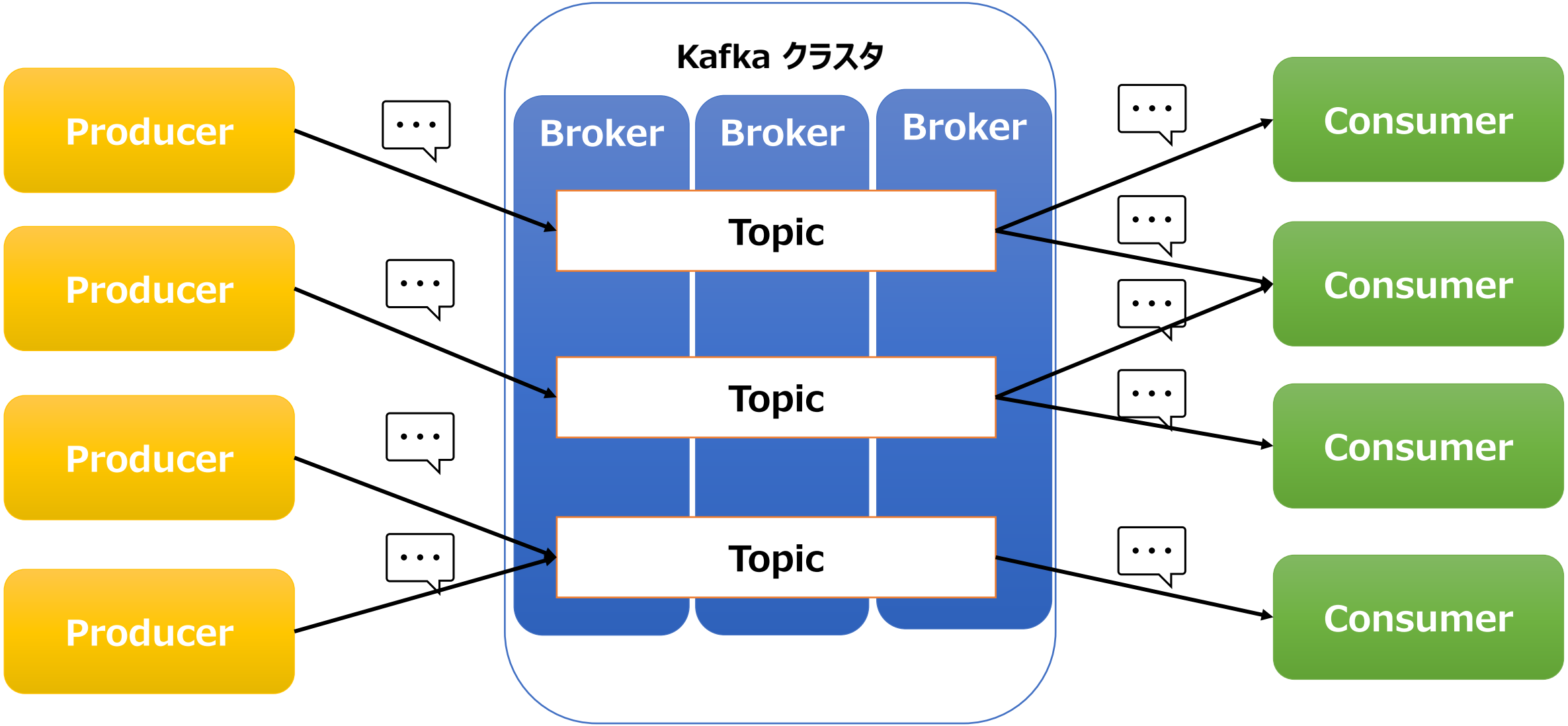
生成元
1つあたりのデータサイズは小さいことが多い
画像やセンサデータなど種類は様々

利用先
センサの異常検知
DBやその他システムへの蓄積など

Kafkaの特徴

- スケーラブル
  - サーバ台数はいつでも増やすことができ、処理に応じて拡張可能
- 低レイテンシ
  - 都度ディスクへの書き込みを行わないため高速に処理
- 高可用性
  - データは複数に複製されるため障害時にもデータを保全
  - 複数の送達保障レベルを実現するための機能
- 多様なシステム連携
  - 様々なクライアントライブラリをサポートし、多様なシステムと連携可能

Kafka コンポーネント概要図



Kafka Streams とは

Kafka Streams とは

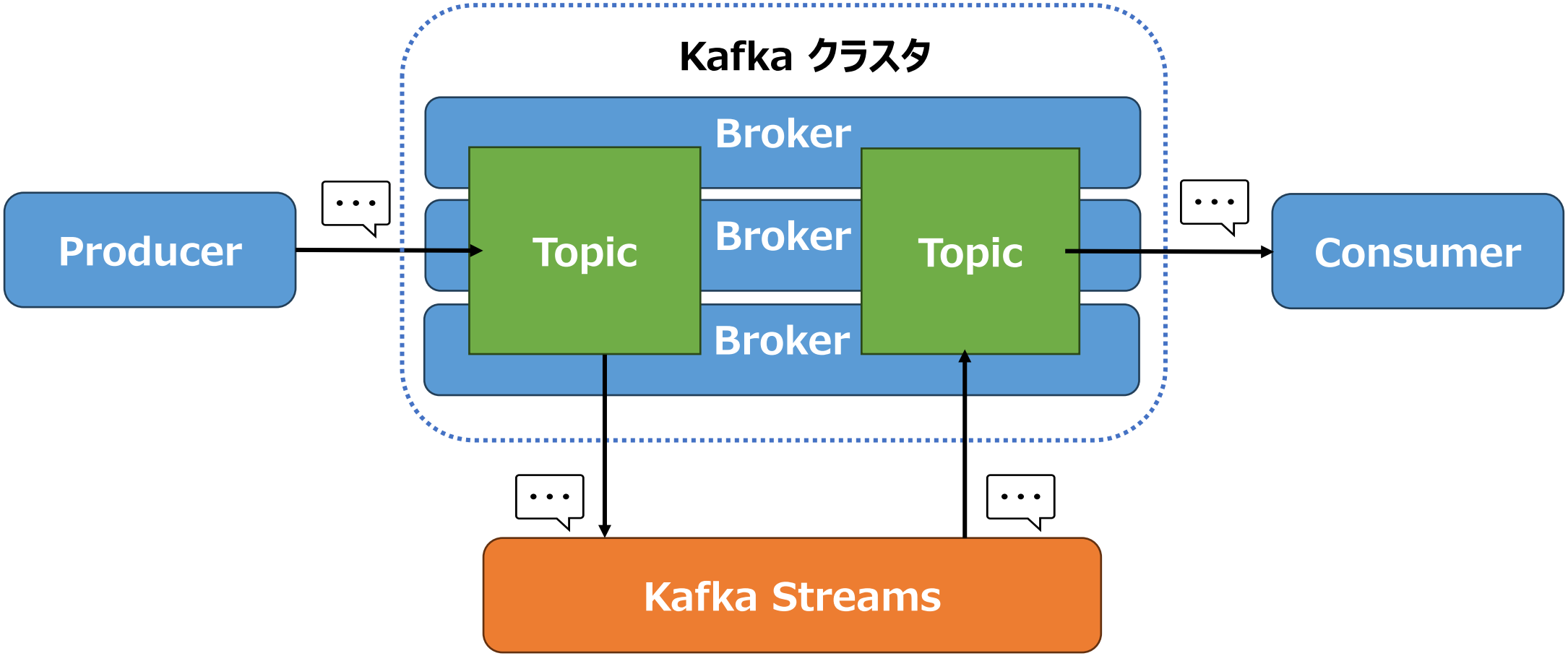
- Apache Kafka クラスタ上でストリーム処理アプリケーションを実装するためのクライアント・ライブラリ
- Java および Scala のライブラリとして提供
- 集めたストリームデータを利用先に配信する前に任意の加工を実施できる

Kafka Steams の特徴

- 扱いやすさ
  - 構築したアプリケーションは通常の Java アプリケーションとしてパッケージング、デプロイが可能
  - ベアメタル、仮想環境、コンテナ、クラウドといった様々な環境にデプロイ可能
  - 規模を問わず様々なユースケースに適用可能
- 高機能
  - Kafka クラスタの持つ拡張性、弾力性、分散性、耐障害性をアプリケーション側でも利用可能
  - Exactly-once (データの抜け漏れや重複なく必ず 1 回だけ) の配信セマンティクスをサポート
  - 複数レコード間の結合や集約、ウィンドウ処理が可能

Kafka Streams でのデータ処理の流れ

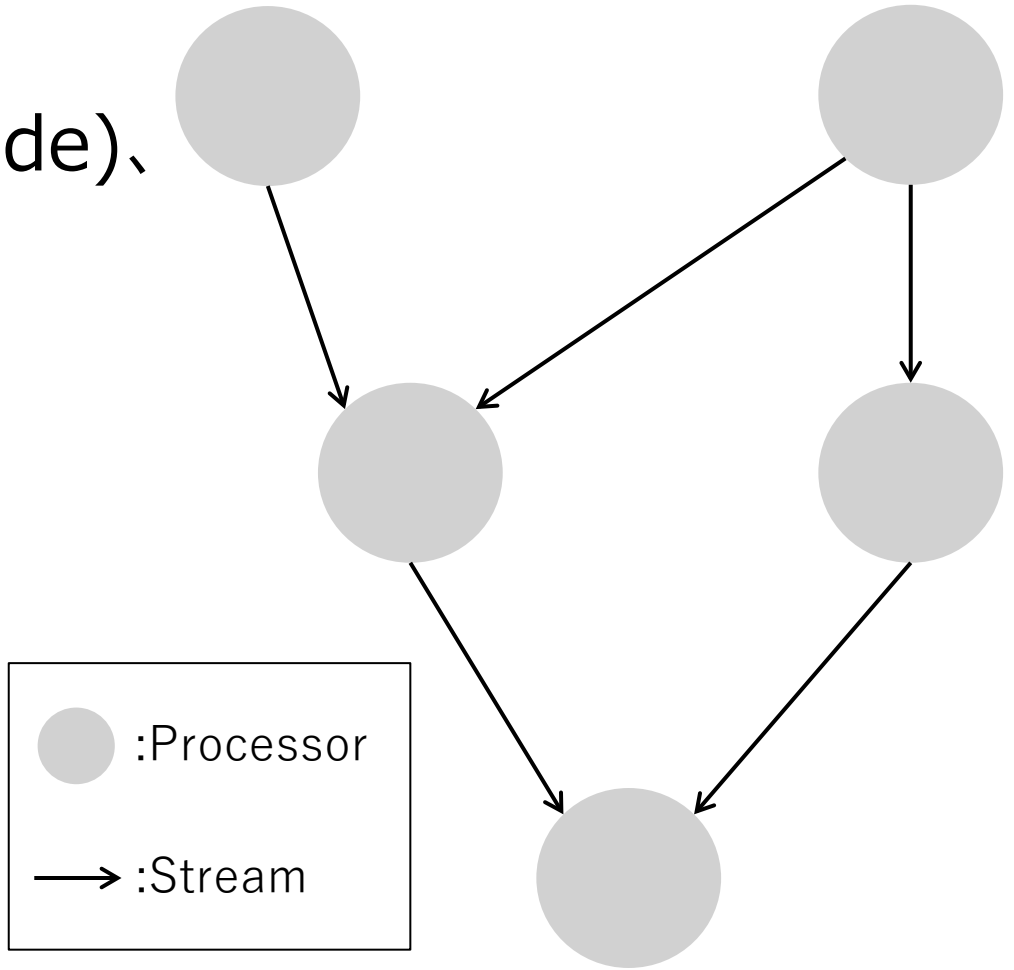
Kafka の Topic にあるデータを入力とし、Kafka Streams で
変換処理を行った後、別の Topic に結果を書き戻す



Kafka Streams のコンセプト

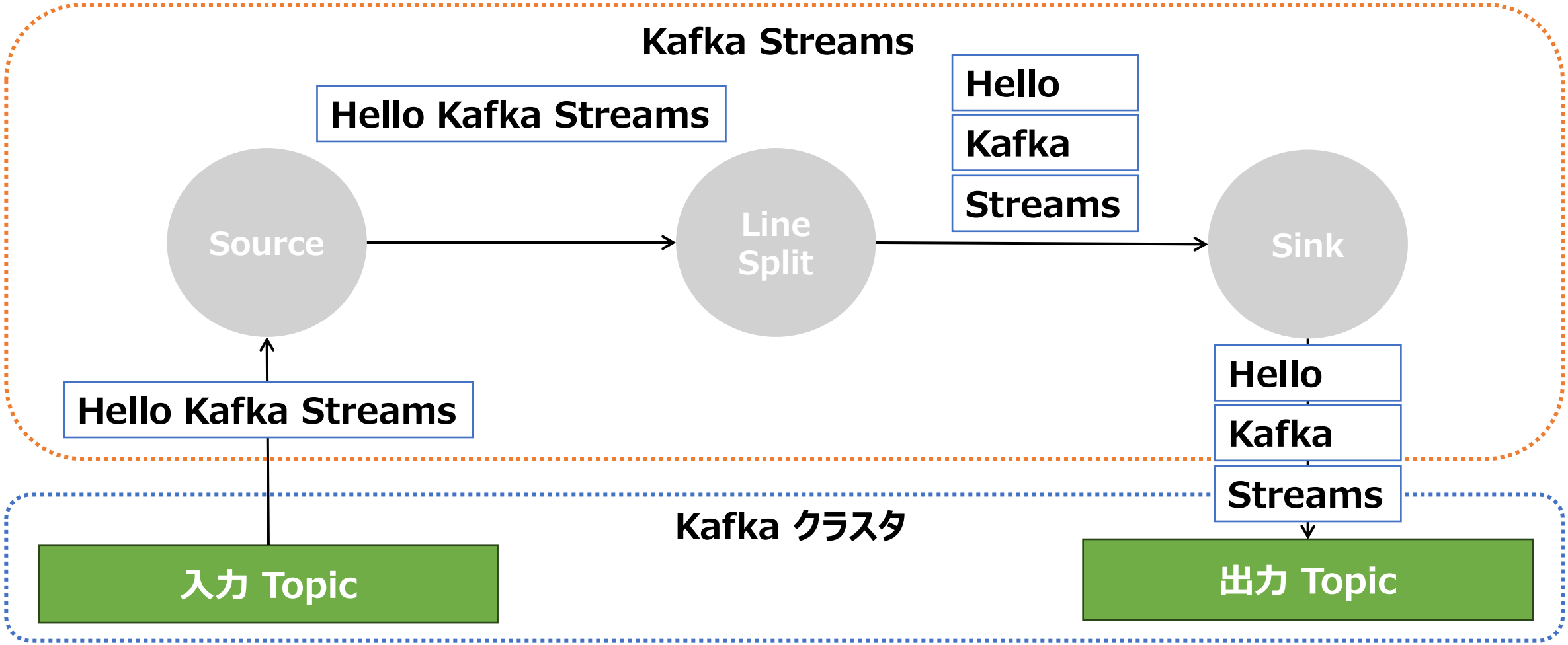
Topology

- ストリーム処理アプリケーションで実行される
  データ処理の流れを表したもの
- データ変換の処理単位を processor (node)、
  データの流れを stream (edge) とした
  グラフで表現
- 特別な processor
  - Source processor
    - トポロジの入力
    - Topic からのデータ読込を担当
  - Sink processor
    - トポロジの出力
    - Topic へのデータ書込を担当



Topology 定義の例

文をスペースで区切って単語に分割する処理



Topology の定義方法

ライブラリで提供される Processor API か Kafka Streams DSL を利用

Processor API

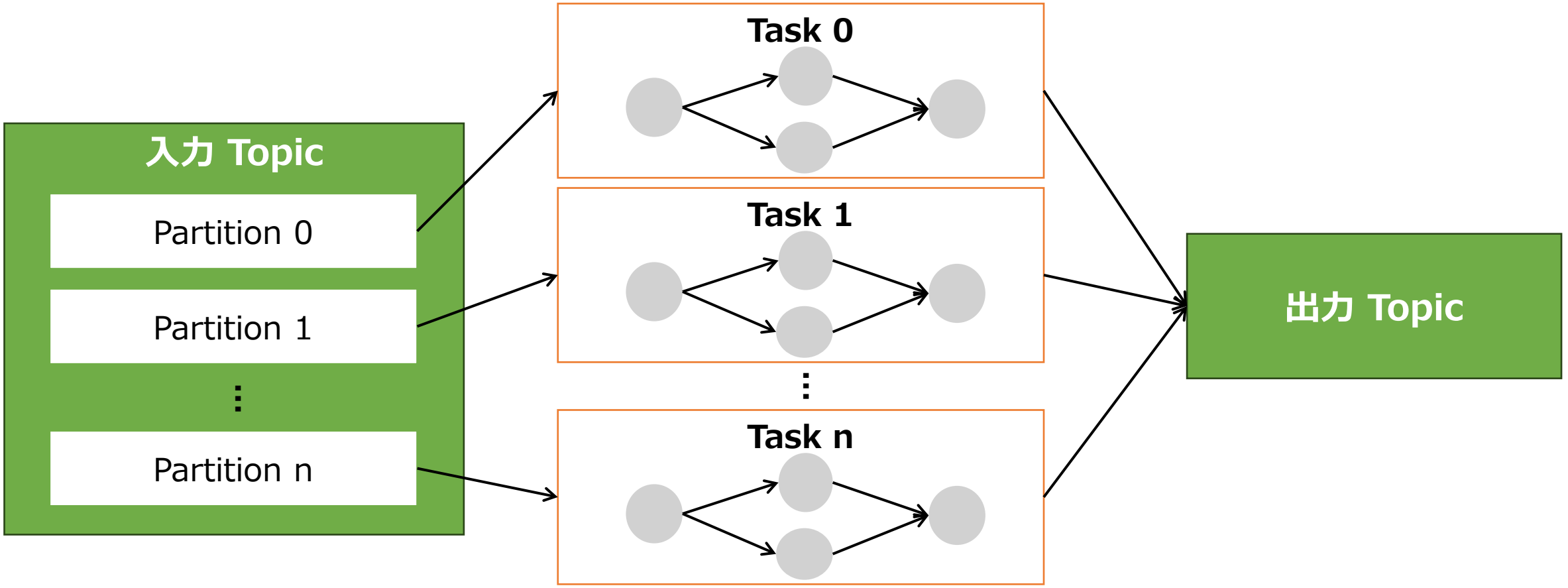
- 命令型の低レベル API
- ユーザ独自の processor の定義や state store との直接のやりとりが可能
- 柔軟性が高いがより多くのコーディングが必要

Kafka Streams DSL

- Processor API を基にした宣言型、関数型の Domain Specific Language (DSL)
- map、filter、join、count、reduce などの関数を使って記述
- 複雑な変換処理を簡単に記述できるので初心者におすすめ

Kafka Streams における Task

- 入力 Topic の Partition ごとに Topology を Task として複製
- 入力 Partition と Task は 1：1 で固定



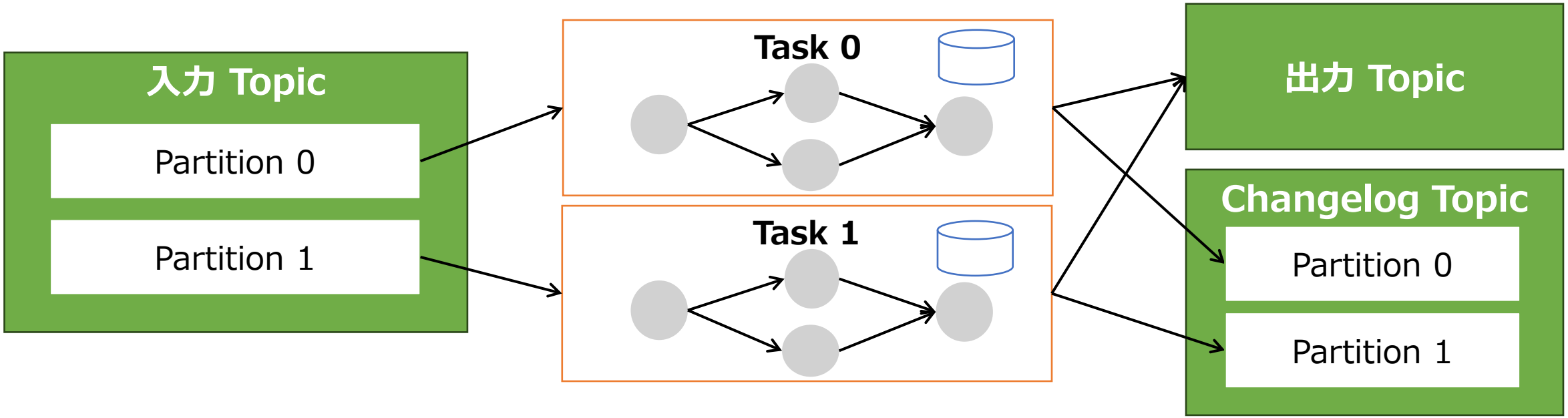
ステートレスとステートフル

ストリーム処理はステートレスとステートフルに分けられる

- ステートレス
  - 単一のレコードで処理が完結
  - 単純にレコード A をレコード B に変換するような処理
- ステートフル
  - 複数のレコードや処理結果を保存し、組み合わせて結果を出力
  - 複数レコードの集計や Window 処理などが該当
  - Kafka Streams では状態保存のために Task ごとに State Store を持つ

State Store

- ステートフル処理においてデータを保存しておくデータストア
- データはテーブルとして保持
- Kafka Streams では Task ごとに State Store を保持
- State Store の変更ログは Kafka 本体の Changelog Topic に格納



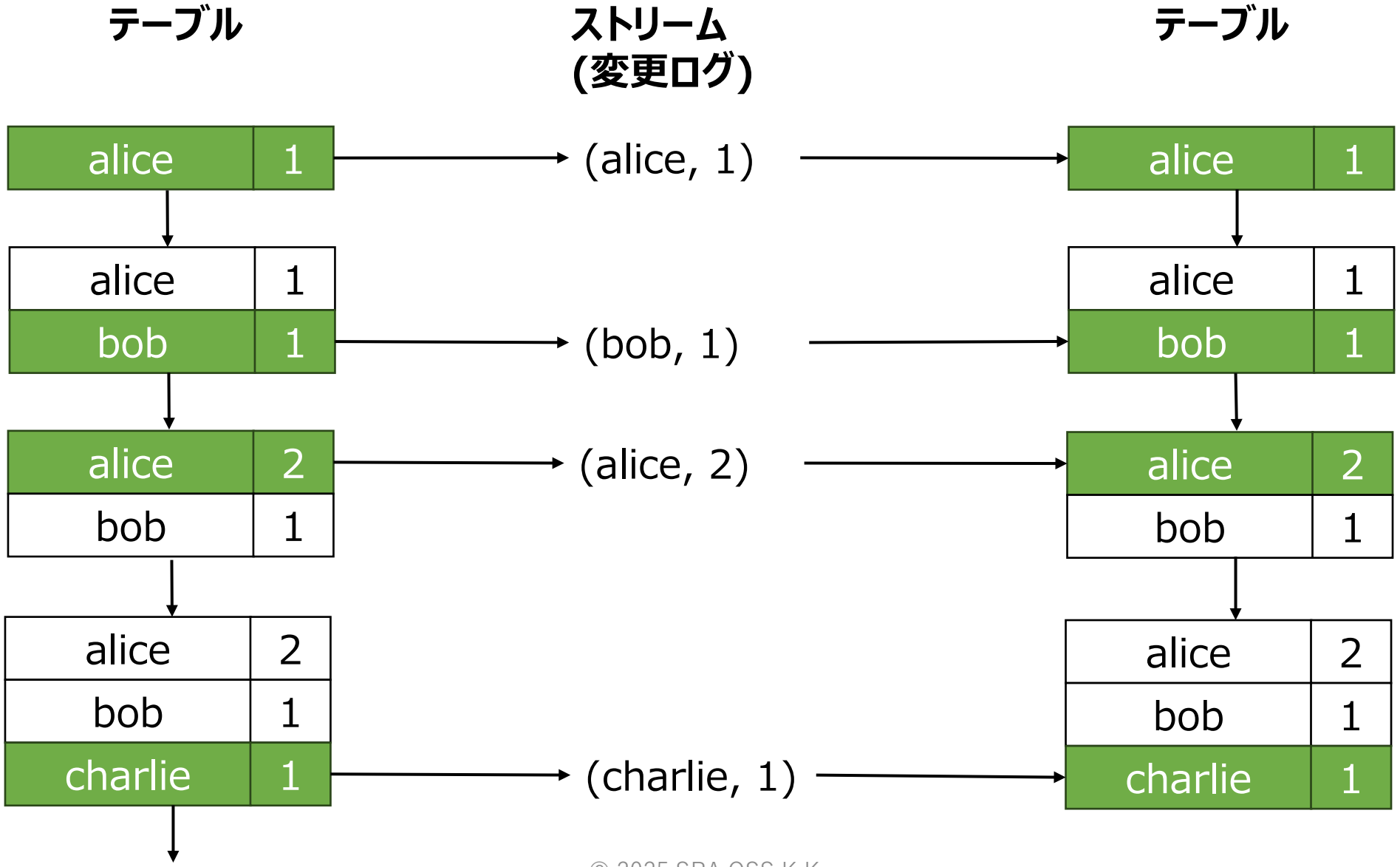
ストリームとテーブルの二重性

本質的にストリームはテーブルとして、テーブルはストリームとして見なせる

- テーブルとしてのストリーム
  - ストリームはテーブルの変更ログと見なせる
  - ストリーム内のデータレコードを最初から最後まで追うことでテーブルを再構築可能
- ストリームとしてのテーブル
  - テーブルは、ストリーム内の各キーに対応するある時点での最新の値のスナップショットと見なせる
  - 異なる時点のテーブル内容を比較することでストリームに変換可能

Kafka Streams では弾力性の担保やステートフル処理の実現などに利用

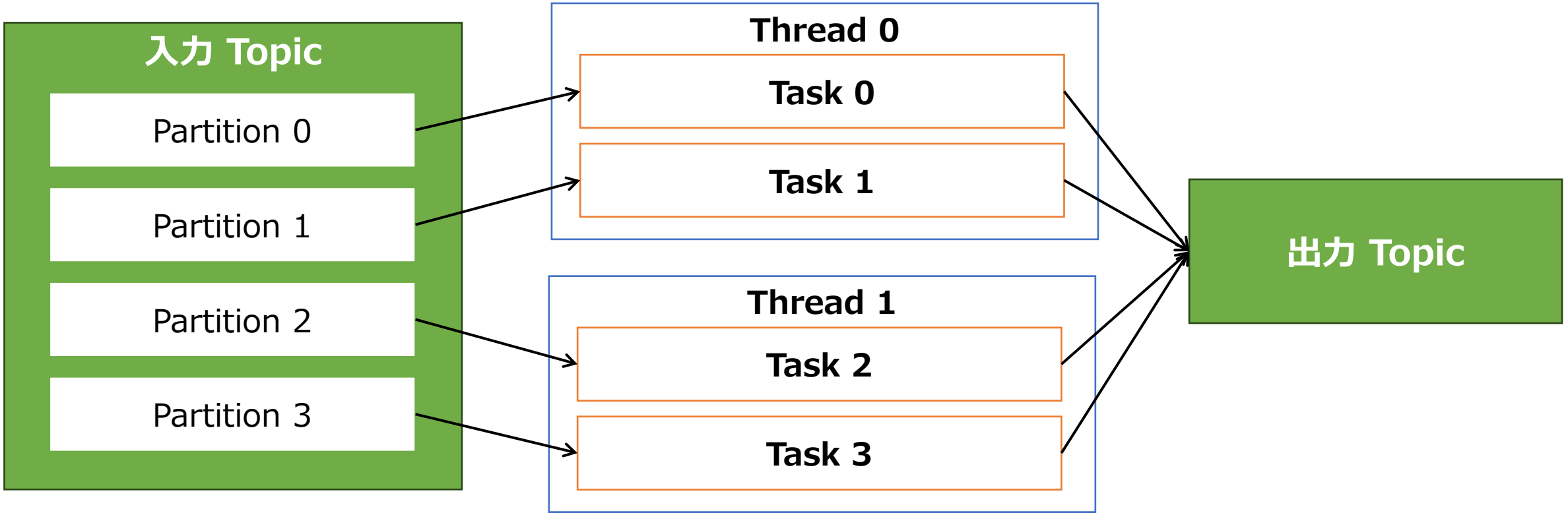
ストリームとテーブルの変換例



Kafka Streams の 並列分散処理と耐障害性

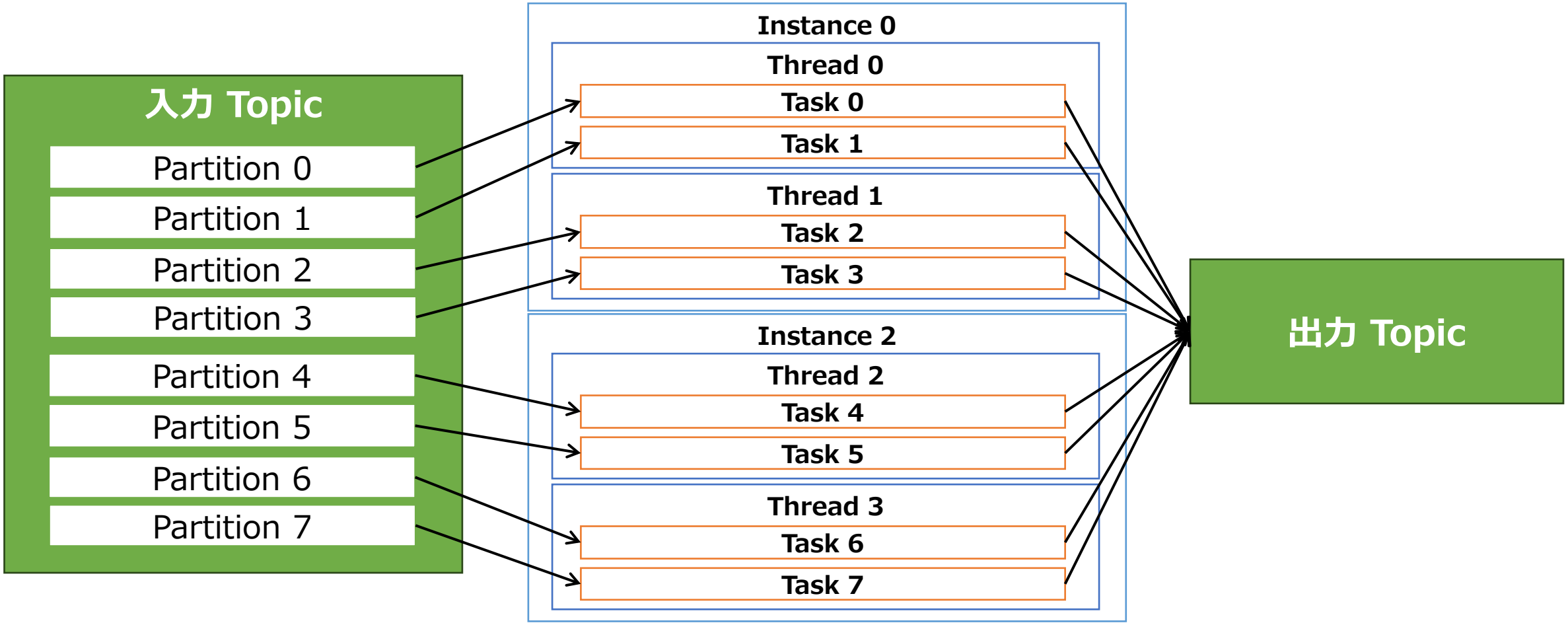
スレッドによる並列分散処理

- アプリケーション内で複数の Java スレッドによる並列処理も可能
- 各スレッドは複数の Task を受け持つ



複数インスタンスによる並列分散処理

- アプリケーションを複数起動することで並列分散処理も可能
- Task の割り当ては Kafka が自動で実施



Kafka Streams のリバランス

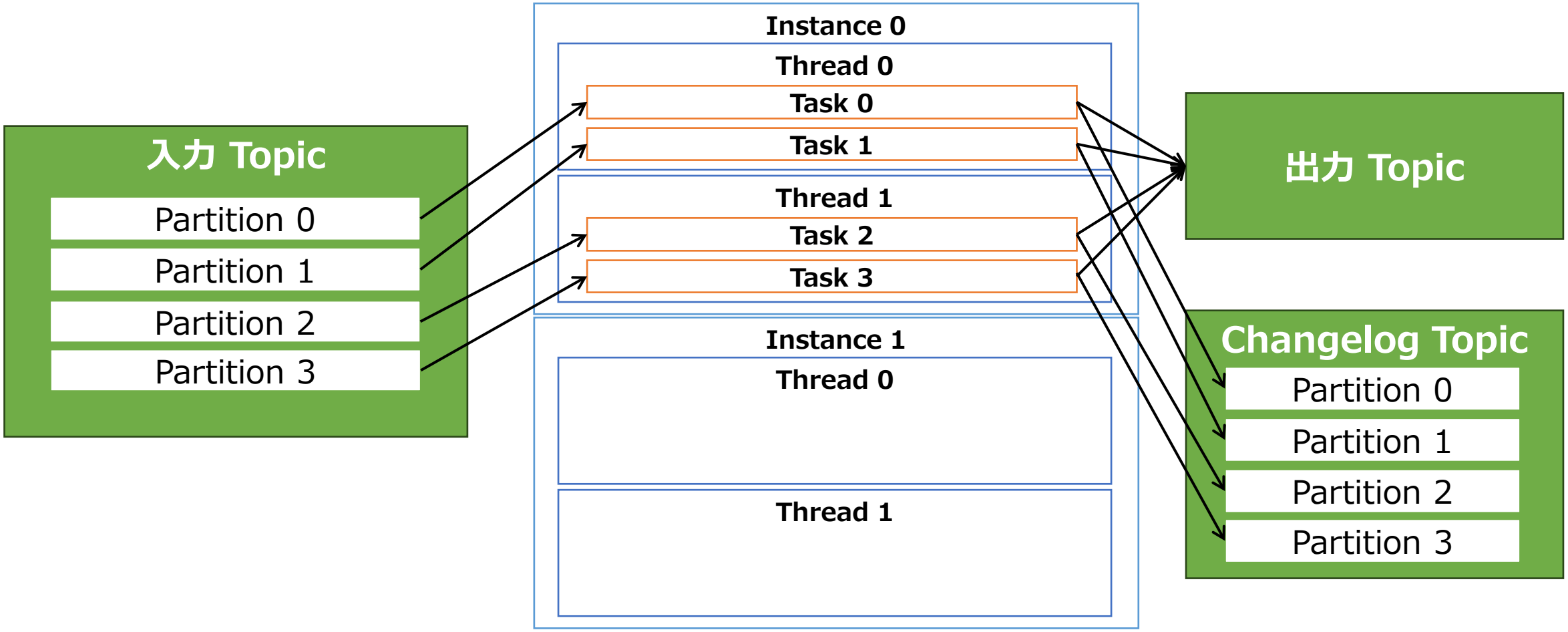
- インスタンス上の Task を別のインスタンス上に配置しなおすこと

- 以下のような利用できるインスタンス数が増減する状況で発生
  - スケールイン
  - スケールアウト
  - サーバ障害

- リバランス時には Task の再配置と State Store の再構築を実施

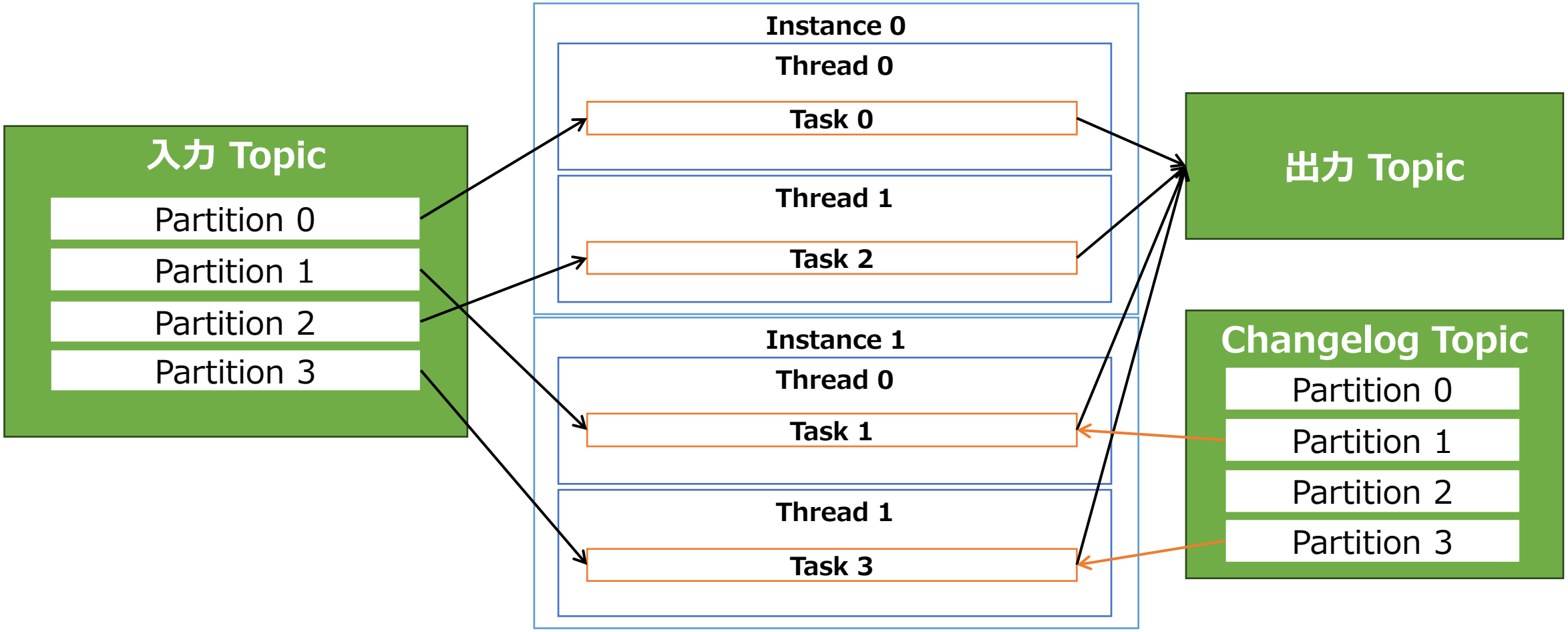
スケールアウト

- 2 つめのインスタンスを追加



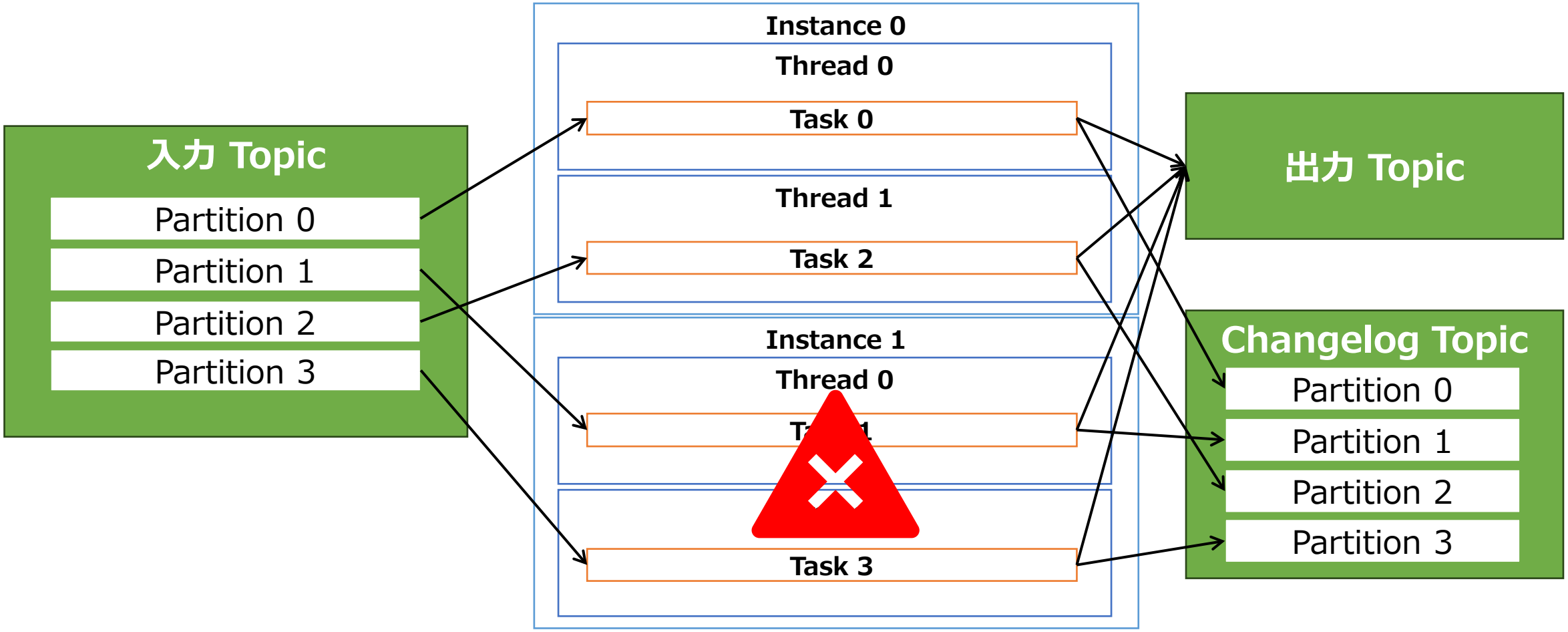
スケールアウト

- Task を均等に再配置
- Changelog Topic から State Store を再構築



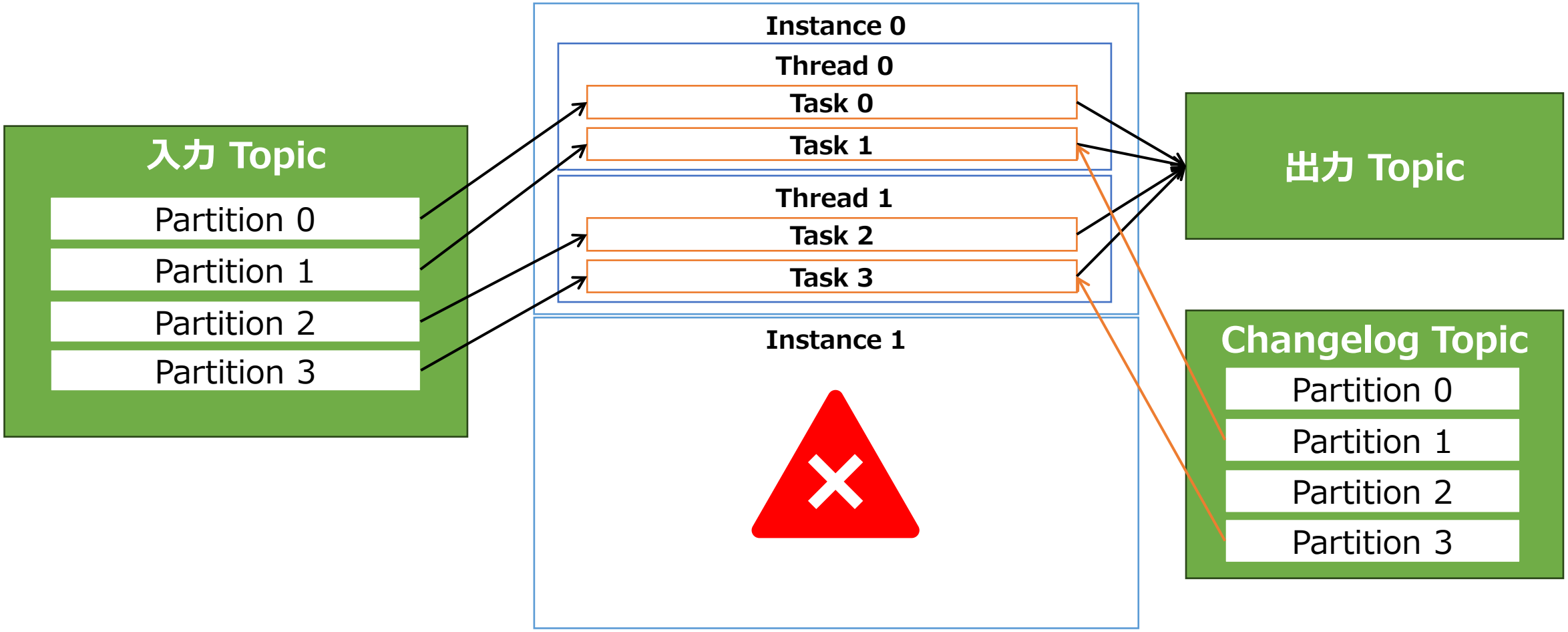
サーバ障害

- サーバで障害発生



サーバ障害

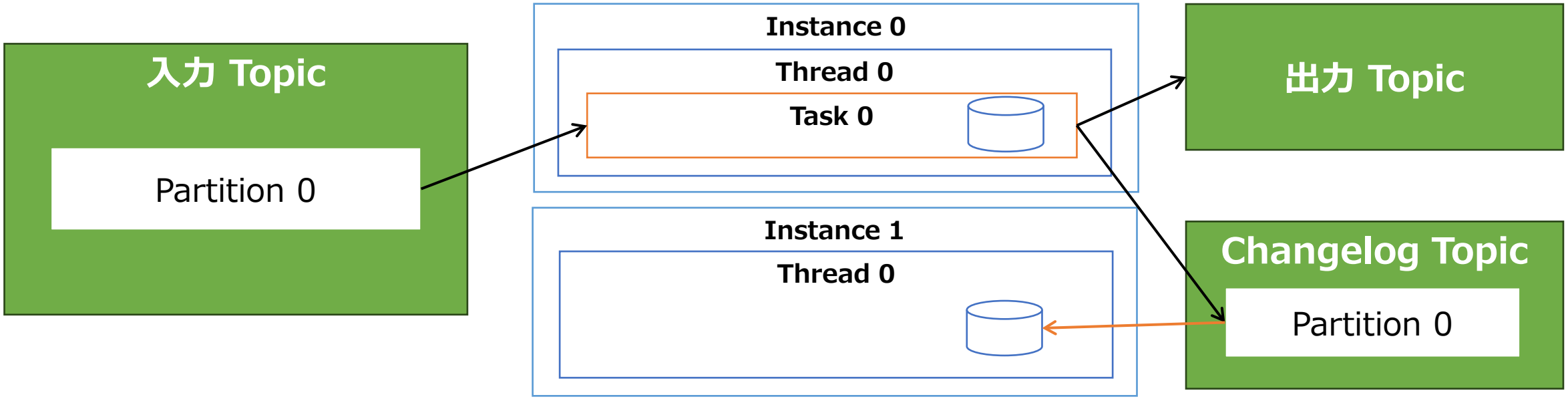
- 残ったインスタンスに Task を移動
- Changelog Topic から State Store を再構築



State Store のスタンバイレプリカ

- 一から再構築するのは時間がかかる
- あらかじめスタンバイレプリカを別インスタンスに準備
- Task をレプリカがあるインスタンスに割り当て

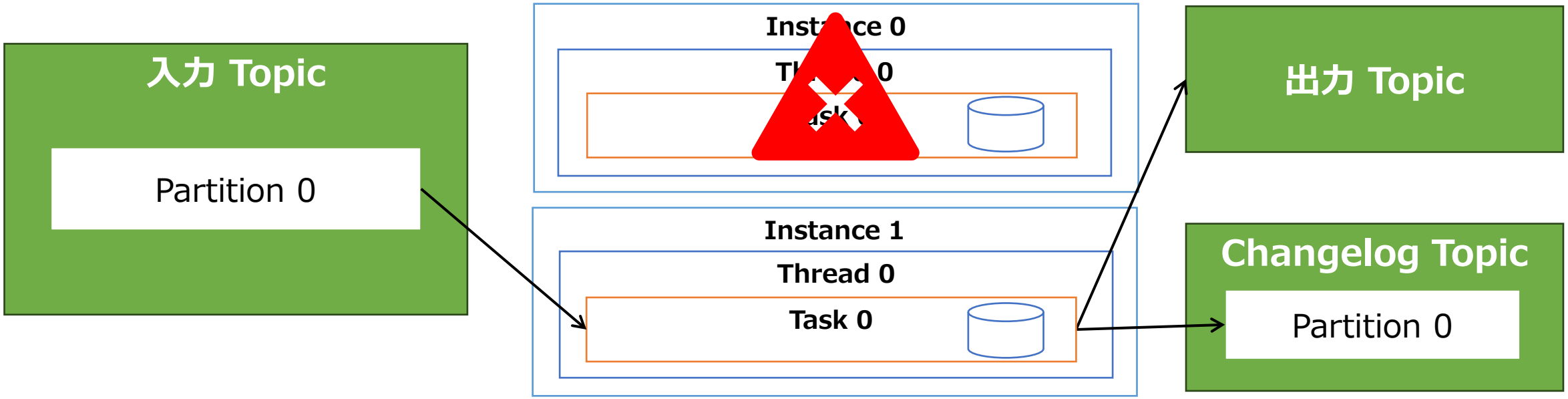
State Store の リストアコスト最小化



State Store のスタンバイレプリカ

- 一から再構築するのは時間がかかる
- あらかじめスタンバイレプリカを別インスタンスに準備
- Task をレプリカがあるインスタンスに割り当て

State Store の
リストアコスト最小化



Kafka Streams 実装例

Line Split

- 文をスペースで区切って単語に分割する処理

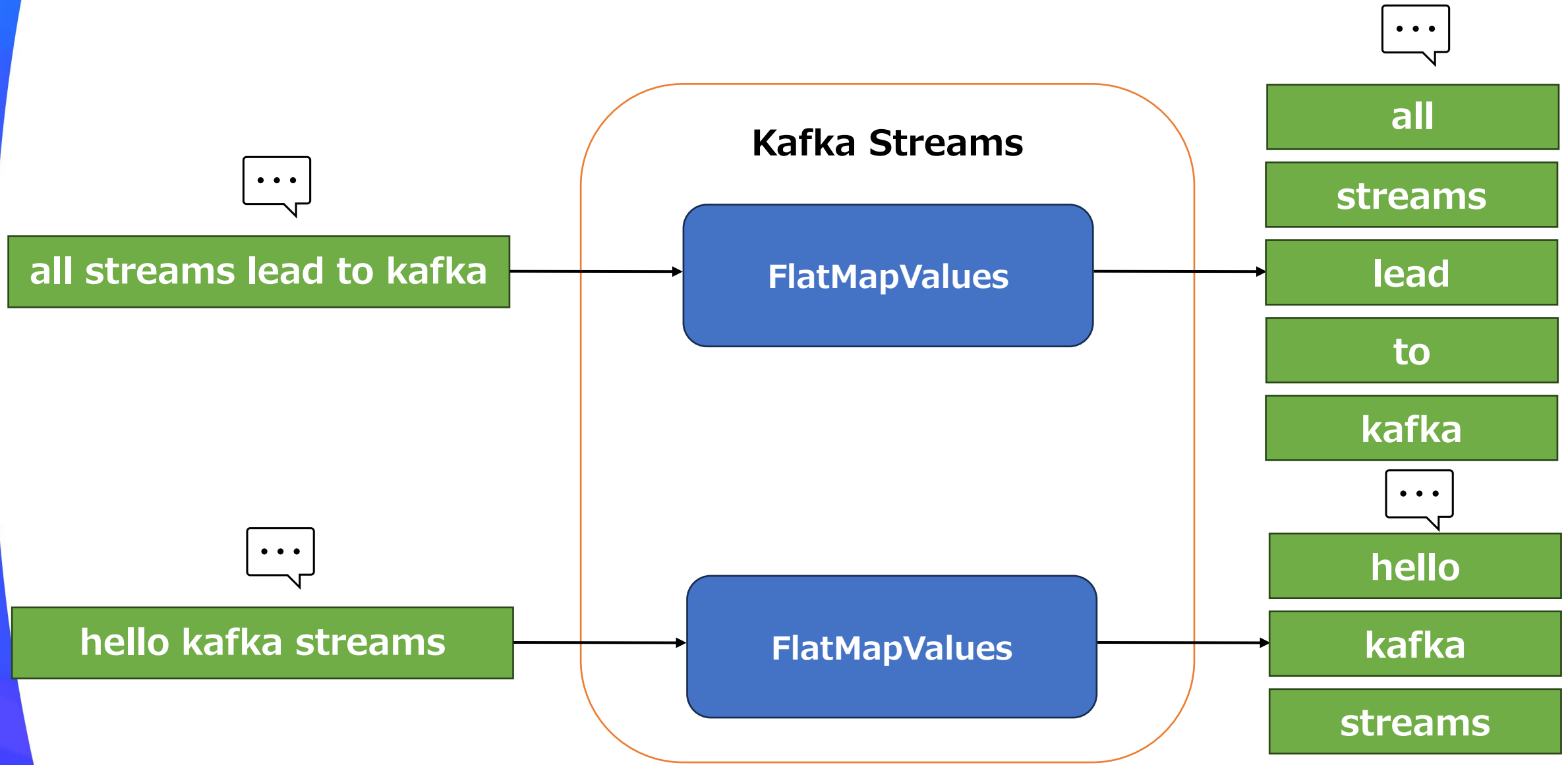
```
public class LineSplit {
  public static void main(String[] args) throws Exception {
    // アプリケーションの設定 Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-linesplit");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.NUM_STREAM_THREADS, 2);
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());

    // トポロジを定義するためのビルダ
    final StreamsBuilder builder = new StreamsBuilder();

    // 処理内容を DSL で記述
    KStream<String, String> source = builder.stream("streams-plaintext-input"); // 入力用 Topic を指定
    source.flatMapValues(value -> Arrays.asList(value.split("¥¥W+"))) // 入力をスペースで区切って分割
      .to("streams-linesplit-output"); // 出力用 Topic に書き込む

    // ビルダからトポロジに変換し、実行
    final Topology topology = builder.build();
    final KafkaStreams streams = new KafkaStreams(topology, props);
    streams.start();
  }
}
```

Line Split



Word Count

文中の単語の出現数をカウントする処理

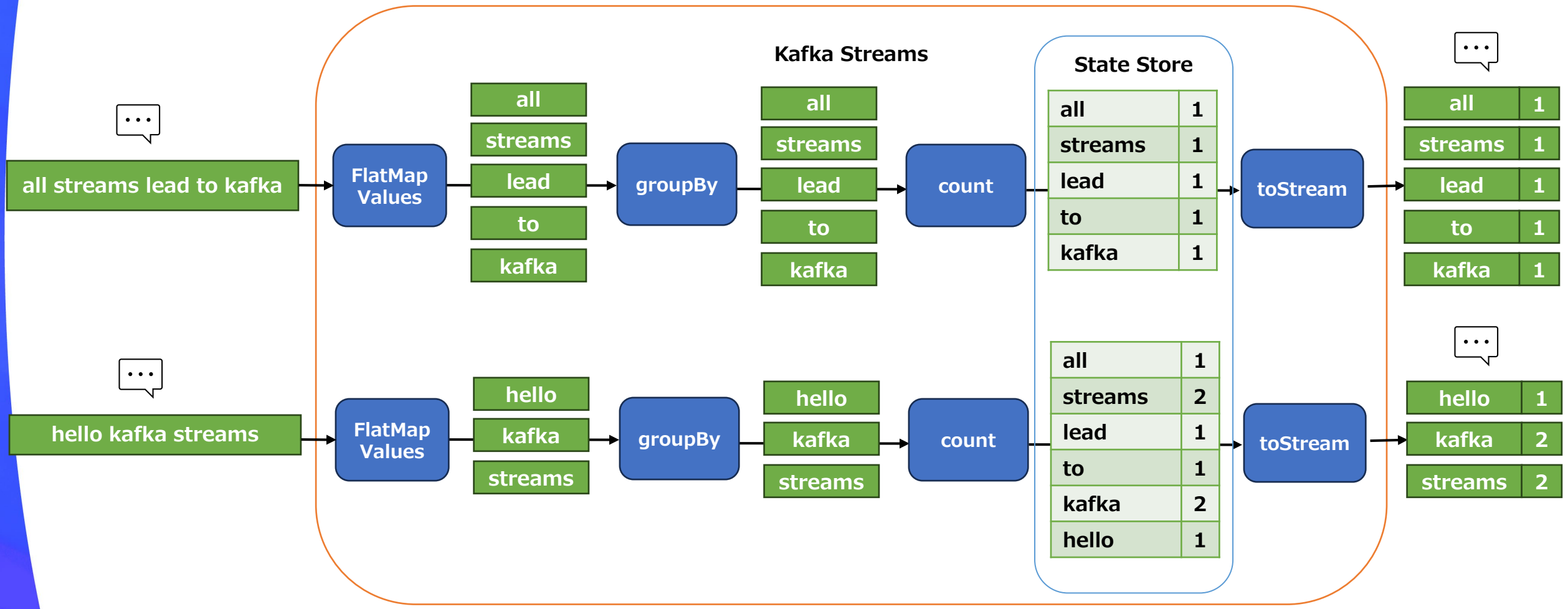
```
public class WordCount {
  public static void main(String[] args) throws Exception {
    // アプリケーションの設定 Properties props = new Properties();
    props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-wordcount");
    props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
    props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.String().getClass());
    props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.String().getClass());

    // トポロジを定義するためのビルダ
    final StreamsBuilder builder = new StreamsBuilder();

    // 処理内容を DSL で記述
    KStream<String, String> source = builder.stream("streams-plaintext-input"); // 入力用 Topic を指定
    source.flatMapValues(value -> Arrays.asList(value.split("¥¥W+"))) // 入力をスペースで区切って分割
      .groupBy((key, value) -> value) // 単語ごとにグループ化
      .count(Materialized.<String, Long, KeyValueStore<Bytes, byte[]>>as(“counts-store”)) // カウントしてテーブルとして State Storeに保存
      .toStream() // テーブルからストリームに変換
      .to("streams-wordcount-output", Produced.with(Serdes.String(), Serdes.Long())); // 出力用 Topic に書き込む

    // ビルダからトポロジに変換し、実行
    final Topology topology = builder.build();
    final KafkaStreams streams = new KafkaStreams(topology, props);
    streams.start();
  }
}
```

Word Count



まとめ

まとめ

- Kafka Streams とは
  - Kafka クラスタ上でストリーム処理アプリケーションを構築するための Java/Scala ライブラリ

- 特徴
  - Java が実行可能な様々な環境にデプロイ可能
  - 高度なストリーム処理に対応
  - 別途クラスタ基盤の用意なしで弾力性や耐障害性を担保

非常に簡単にストリーム処理アプリケーションを構築可能

SRA OSS

www.sraoss.co.jp sales@sraoss.co.jp 03-5979-2701