

# データインモーションの二つの哲学

Kafkaの「ログ」と標準メッセージングの「キュー」を理解する

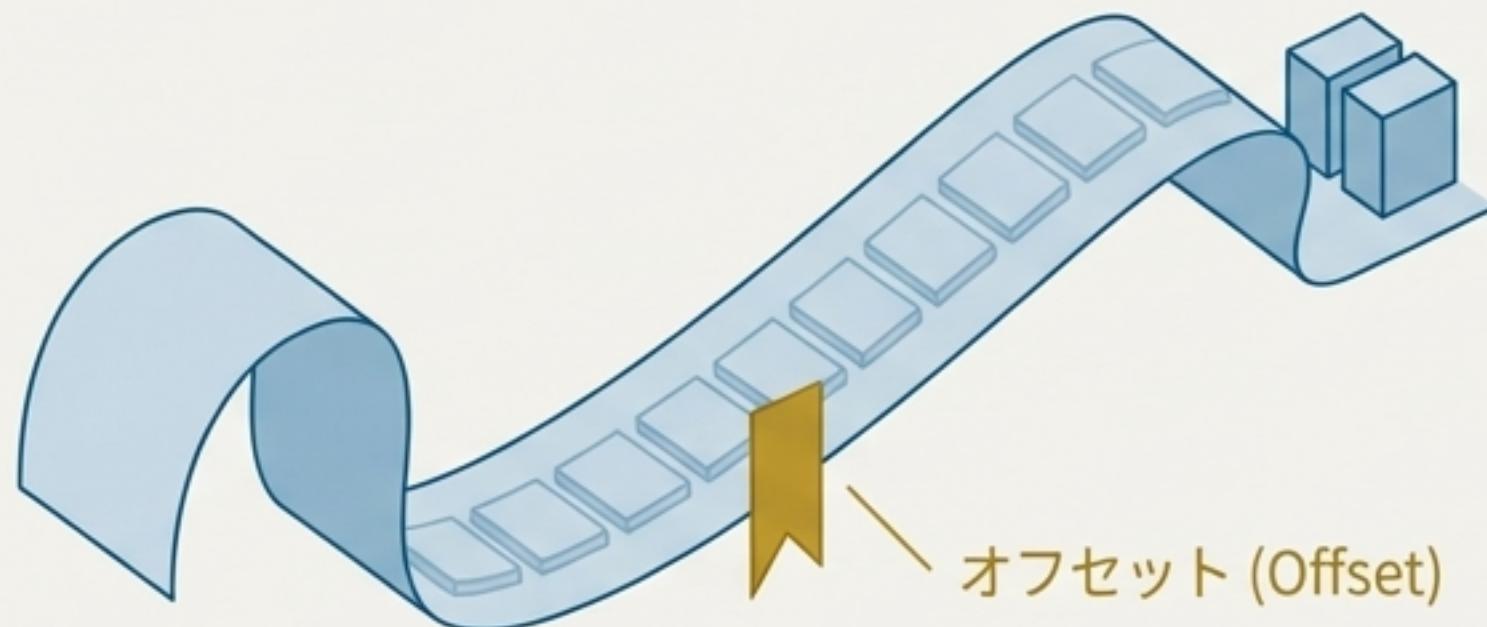
# なぜこの比較が重要なのか？

- ・ソフトウェアアーキテクチャにおいて、「Kafkaか、RabbitMQか」という問い合わせ頻繁に発生します。
- ・しかし、これは機能リストの比較で解決する問題ではありません。
- ・これらは、データの流れを扱う根本的な「哲学」が異なります。
- ・このセッションの目的は、その中核となる哲学を理解し、どちらがあなたのユースケースに適合するかを判断するための、強固なメンタルモデルを構築することです。



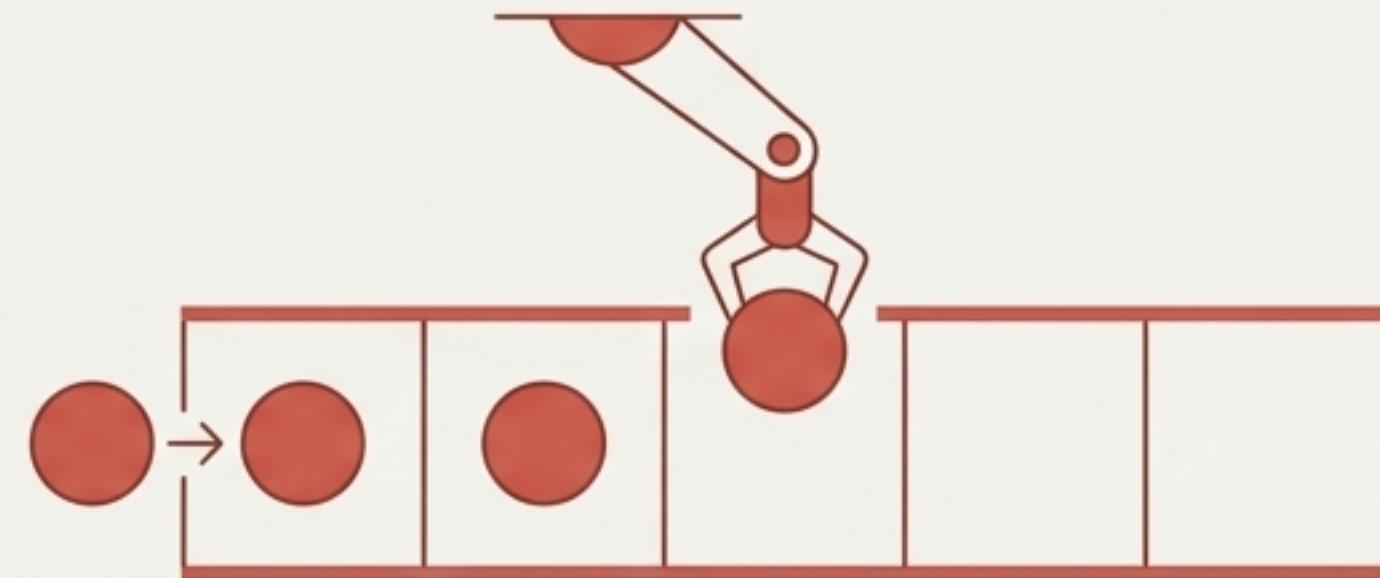
# すべての始まり：一つの根本的な違い

Kafka: 不変の「ログ」



永続的で、順序が保証された、追記専用のイベントの記録。データは消費されても削除されない。

標準MQ: 振発性の「キュー」

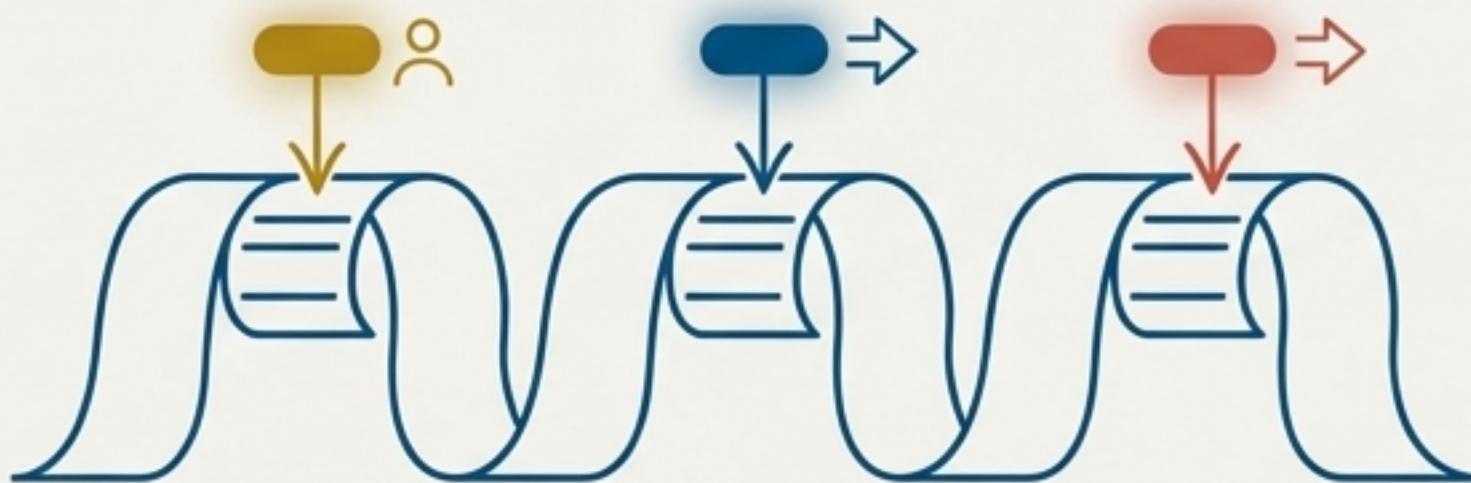


実行されるべきタスクのリスト。メッセージは処理が完了するとキューから削除される。

# 結果① データ保持の考え方

## Kafka

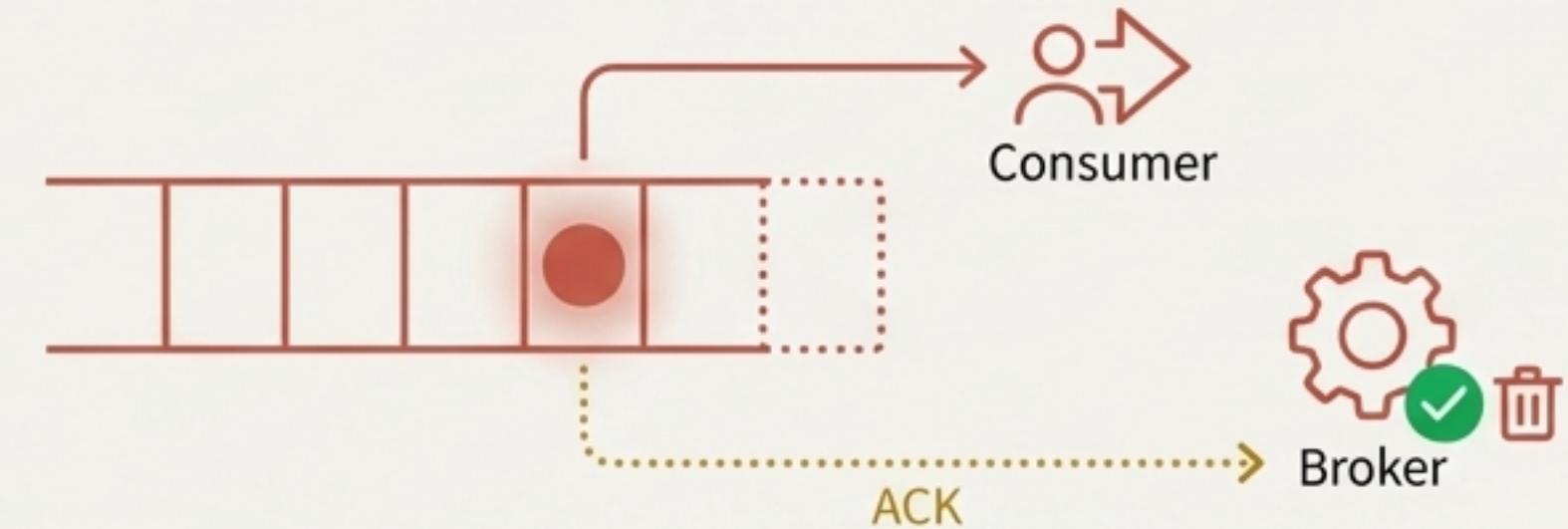
原則：ログは「記録」である



- データは消費されても消えない。
- 保持期間は時間（例：7日間）やサイズ（例：1TB）で設定される。
- これにより、過去のイベントへのアクセスと「再生」が可能になる。

## Standard MQ

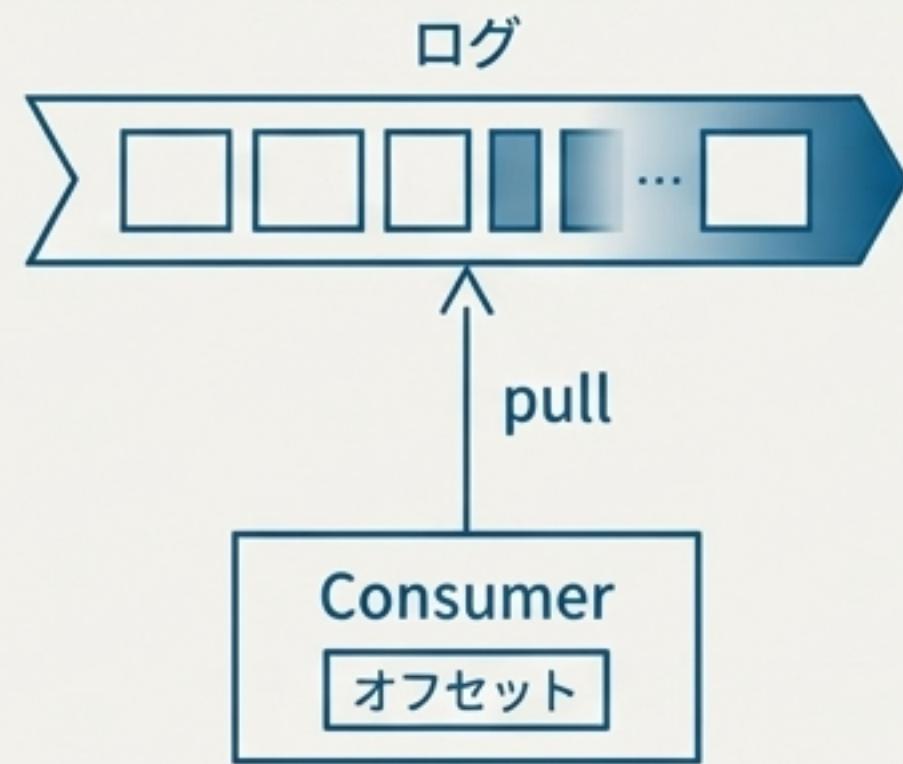
原則：キューは「To-Doリスト」である



- メッセージは、コンシューマによる受信確認（ACK）をもって削除されるのが基本。
- ブローカーが各メッセージの配信状態を積極的に管理する。
- 目的は、タスクが確実に一度処理されることを保証すること。

## 結果② 消費モデル：誰が主導権を握るか

### コンシューマが主導する「プル」モデル



### プローカーが主導する「pussh」モデル



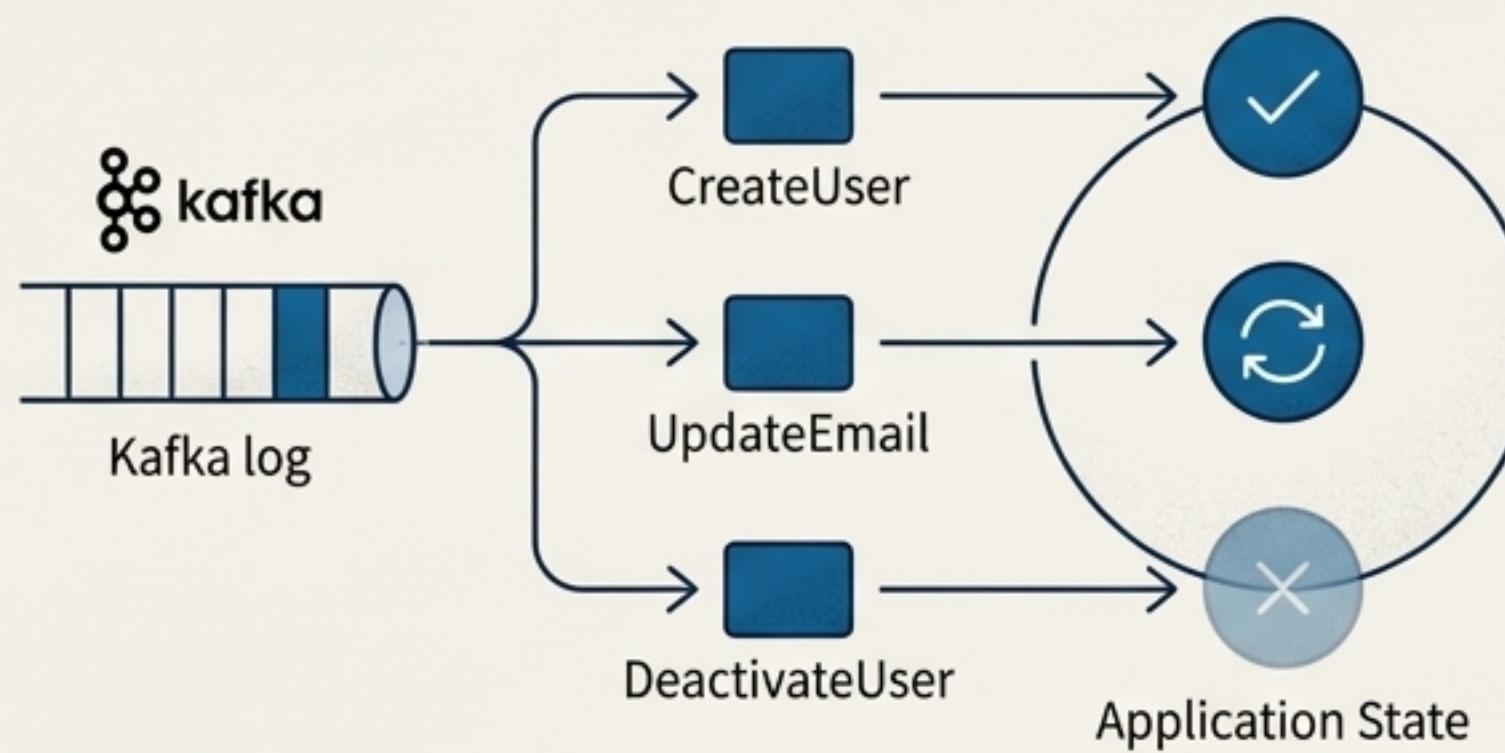
- ・コンシューマは、自分がどこまで読み進めたか（オフセット）を自己管理する。
- ・プローカーはコンシューマの状態を追跡しないため、コンシューマの追加・削除が容易。
- ・コンシューマは自身のペースでデータを「プル」する。

- ・プローカーがメッセージをコンシューマに「pussh」し、配信状況を監視する。
- ・プローカーのロジックはより複雑になるが、メッセージ単位でのきめ細やかな制御が可能。
- ・ラウンドロビンなどの高度なディスパッチ戦略を実装できる。

# 「ログ」がもたらす力：再生可能性と新たなアーキテクチャ

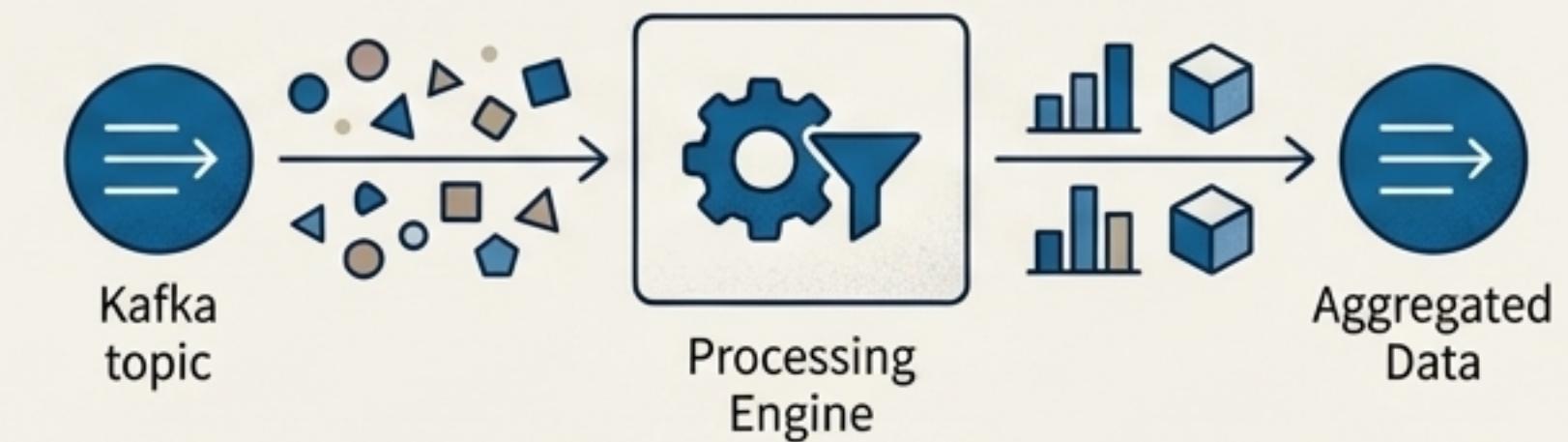
Kafkaの不变ログの原則は、単なるメッセージング以上の可能性を拓きます。

## イベントソーシング (Event Sourcing)



アプリケーションの状態をイベントのシーケンスとして保存する。システムの現在の状態は、すべてのイベントを最初から再生することでいつでも再構築できる。

## ストリームプロセッシング (Stream Processing)

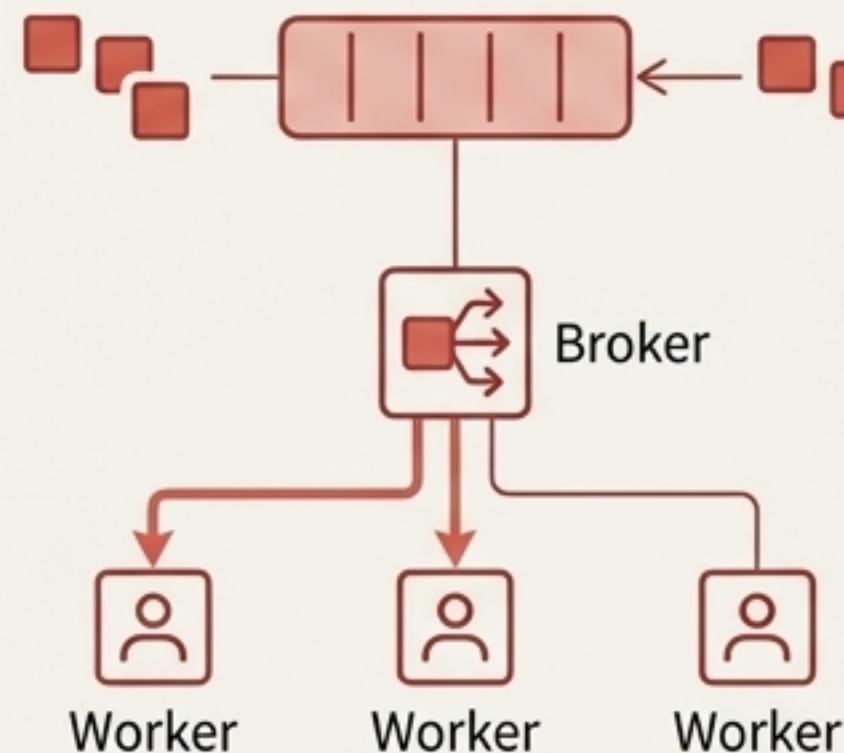


リアルタイムでイベントストリームを変換、集計、分析する。複数の独立したアプリケーションが同じデータストリームを異なる目的で利用できる。

# 「キュー」がもたらす信頼性：確実なタスク処理とワークフロー

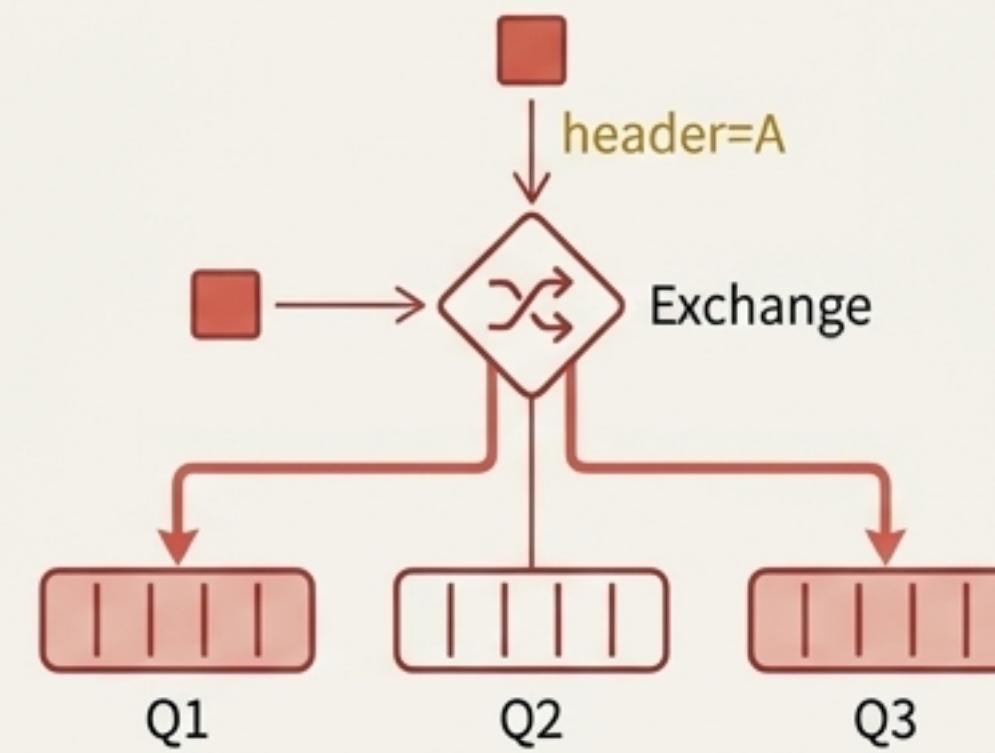
標準MQのキュー モデルは、分散システムにおけるタスク処理の信頼性を保証するために最適化されています。

## ワークディストリビューションと負荷分散 (Work Distribution and Load Balancing)



複数のコンシューマ（ワーカー）でタスクを効率的に分担する。ワーカーの増減に合わせて自動的に負荷が分散される。

## 複雑なルーティングとフィルタリング (Complex Routing and Filtering)

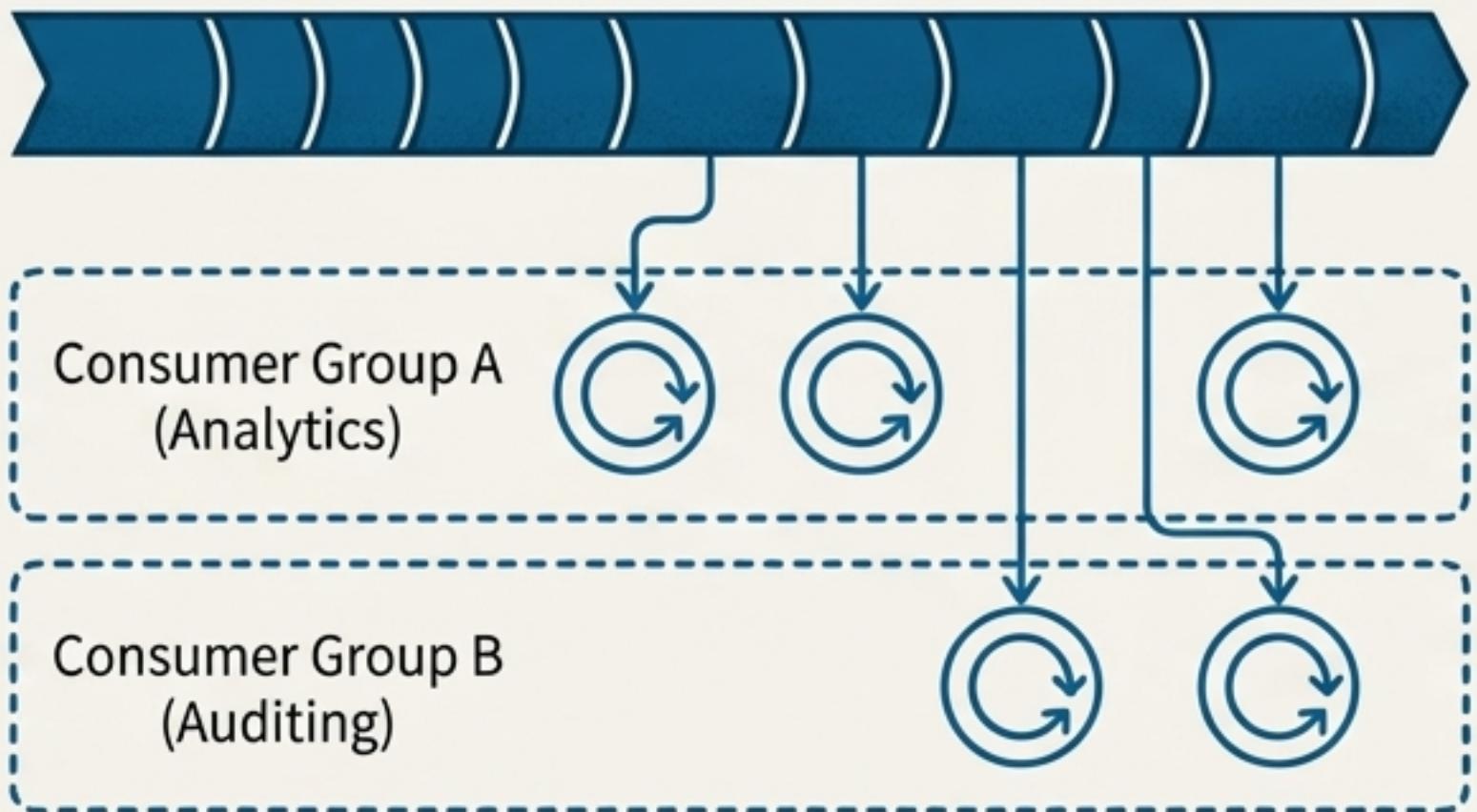


メッセージの内容や属性に基づいて、特定のコンシューマにメッセージをインテリジェントに振り分ける。デッドレターキュー (DLQ) などの高度なエラー処理パターンを実装しやすい。

# コンシューマの結合度とスケーラビリティ

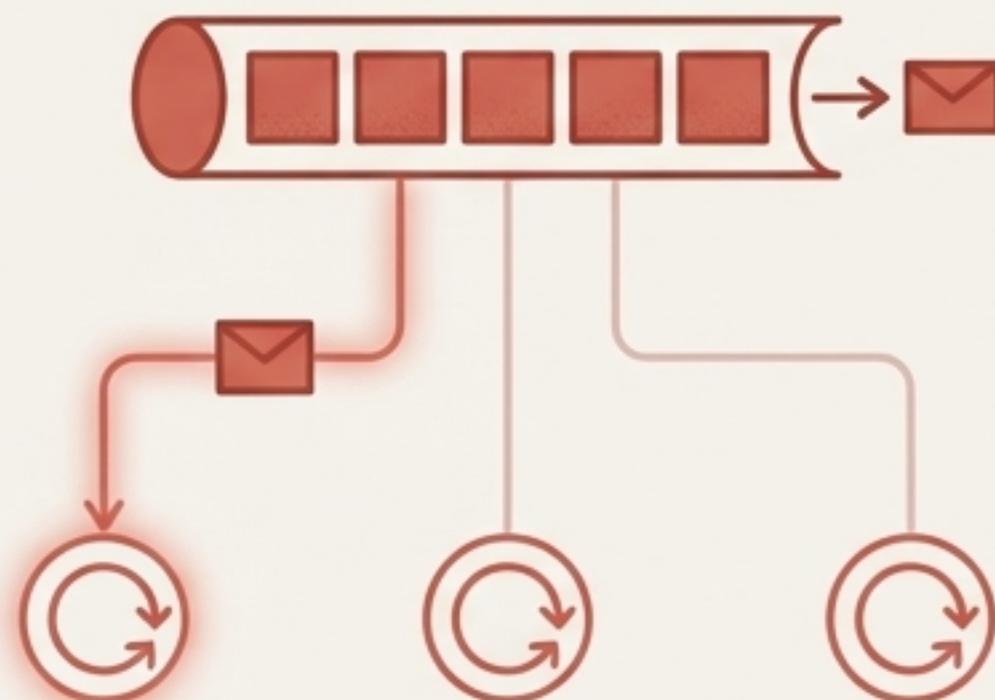
## Kafka

### 独立したコンシューマグループ



## Standard MQ

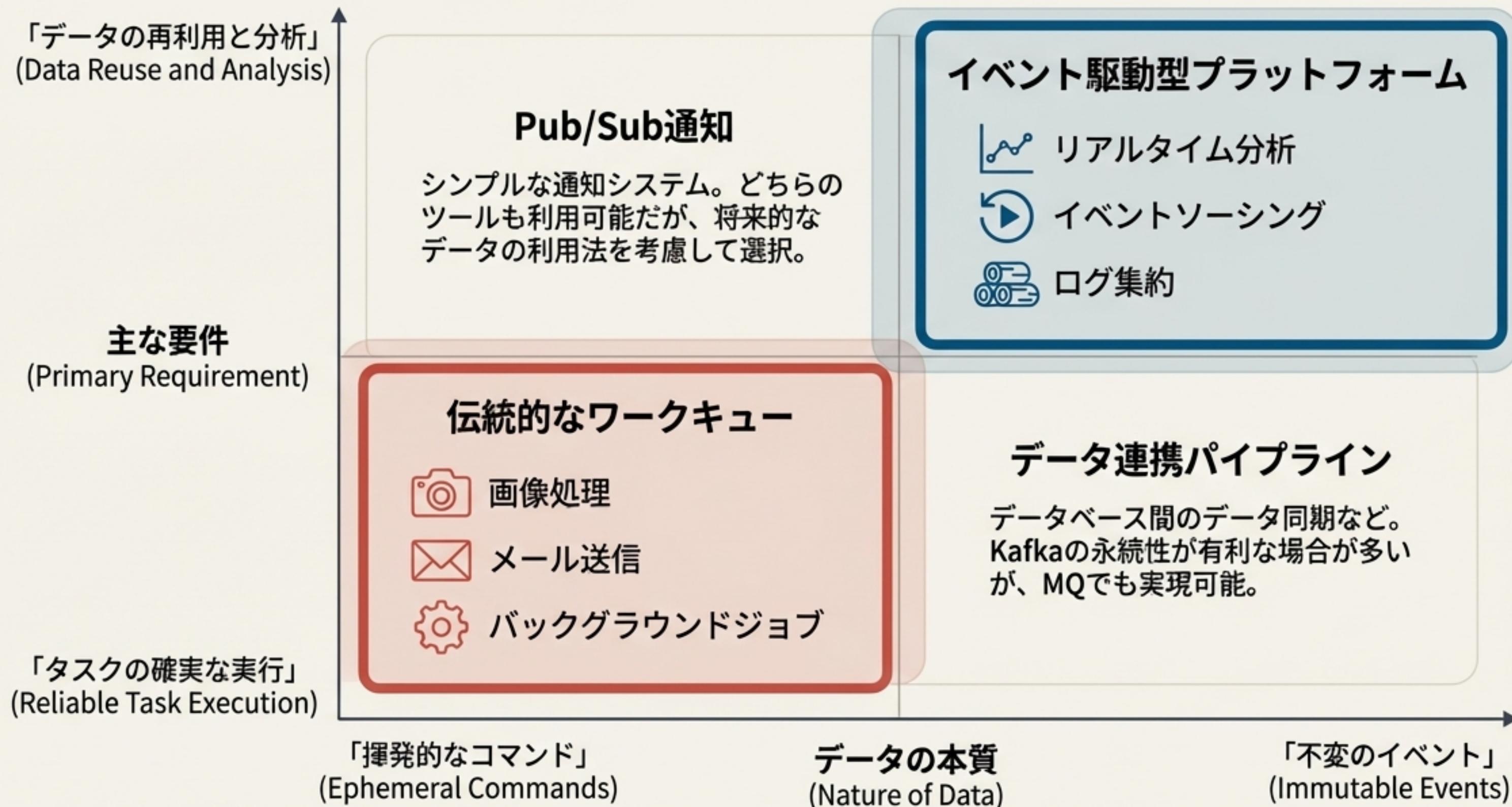
### 競合するコンシューマ



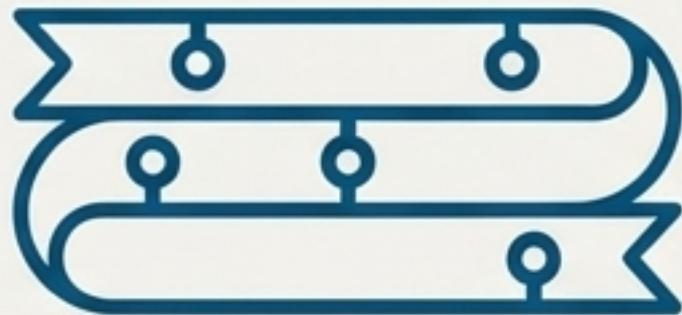
- コンシューマグループは互いに完全に独立している。
- 同じデータを複数の目的で、互いに影響を与えることなく利用できる。
- 新しいユースケースが生まれても、既存のシステムに影響を与えずに新しいコンシューマグループを追加するだけ。

- 一つのキューに接続されたコンシューマは、メッセージを取り合う「競合」関係にある。
- これはタスクの並列処理には理想的。
- 同じメッセージを複数のコンシューマに届けたい場合は、Pub/Subモデル（例：Fanout Exchange）を別途設定する必要がある。

# 選択のためのフレームワーク：あなたの問題領域はどちらか？



# 結論：哲学を理解し、適切なツールを選ぶ



Kafkaは「イベントの記録システム」である

- データがビジネスの中心的な資産である場合
- 複数のチームが同じデータを異なる方法で利用する場合
- 過去のデータへのアクセスが将来価値を生む場合



標準MQは「タスクの分散システム」である

- 特定の作業を確実に非同期で実行したい場合
- メッセージ単位での複雑なルーティングやライフサイクル管理が必要な場合
- ブローカーによるきめ細やかな制御と保証を求める場合

優れたアーキテクトは、  
単にツールを比較するのではない。

その根底にある思想を理解し、  
問題に最も適したパラダイムを選択する。