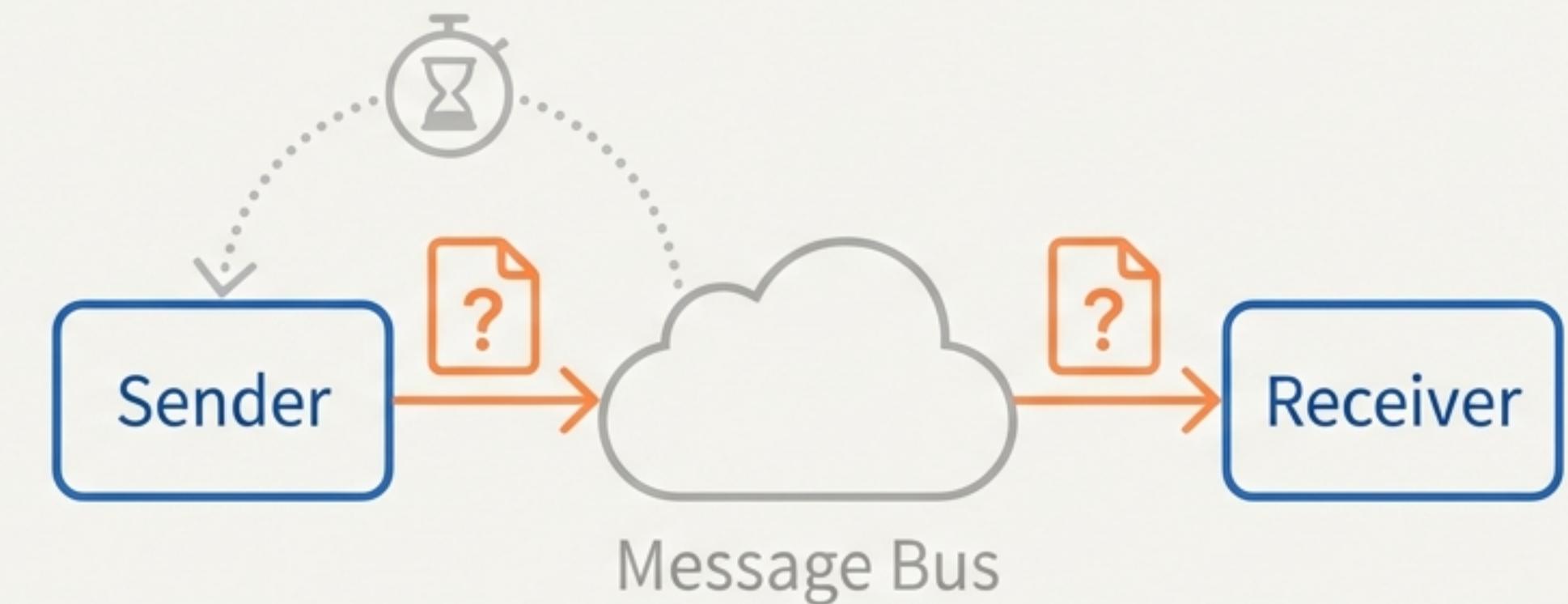


イベント駆動アーキテクチャ：リクエスト/ リプライ パターンを解き明かす

非同期通信で同期的応答を実現する2つの主要な手法

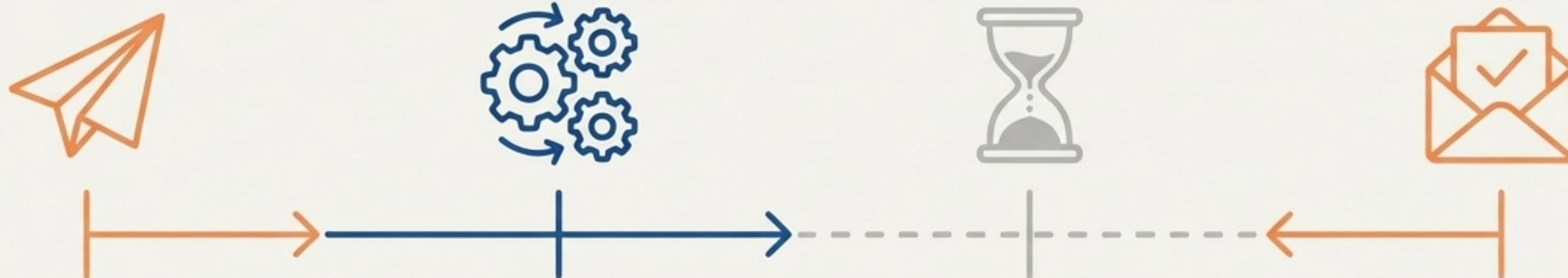
非同期通信の根本的な問い合わせ：どうすれば応答を受け取れるのか？

イベント駆動アーキテクチャは、サービスの疎結合化とスケーラビリティに優れています。しかし、リクエストを送信した後、その処理結果を直接受け取たい場面は頻繁に発生します。非同期プロトコルの中で、どのようにして確実に応答を取得するのでしょうか？



「疑似同期的」メッセージング：非同期の柔軟性と応答性を両立する

Sender's Activity Timeline



1. リクエスト送信

送信者はリクエストをキューに送信します。

2. 別処理の実行

送信後、送信者は待機せず、他の処理を自由に実行できます。

3. ブロッキング待機

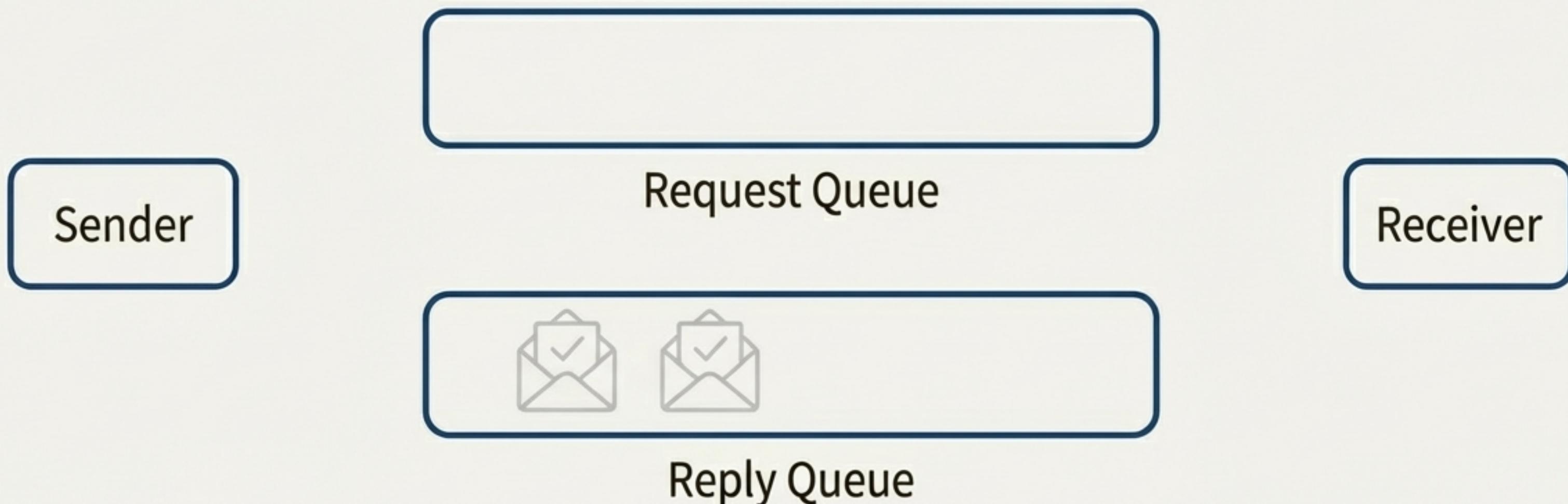
応答が必要なタイミングで、応答キューを監視し、メッセージを待ちます。

4. 応答受信

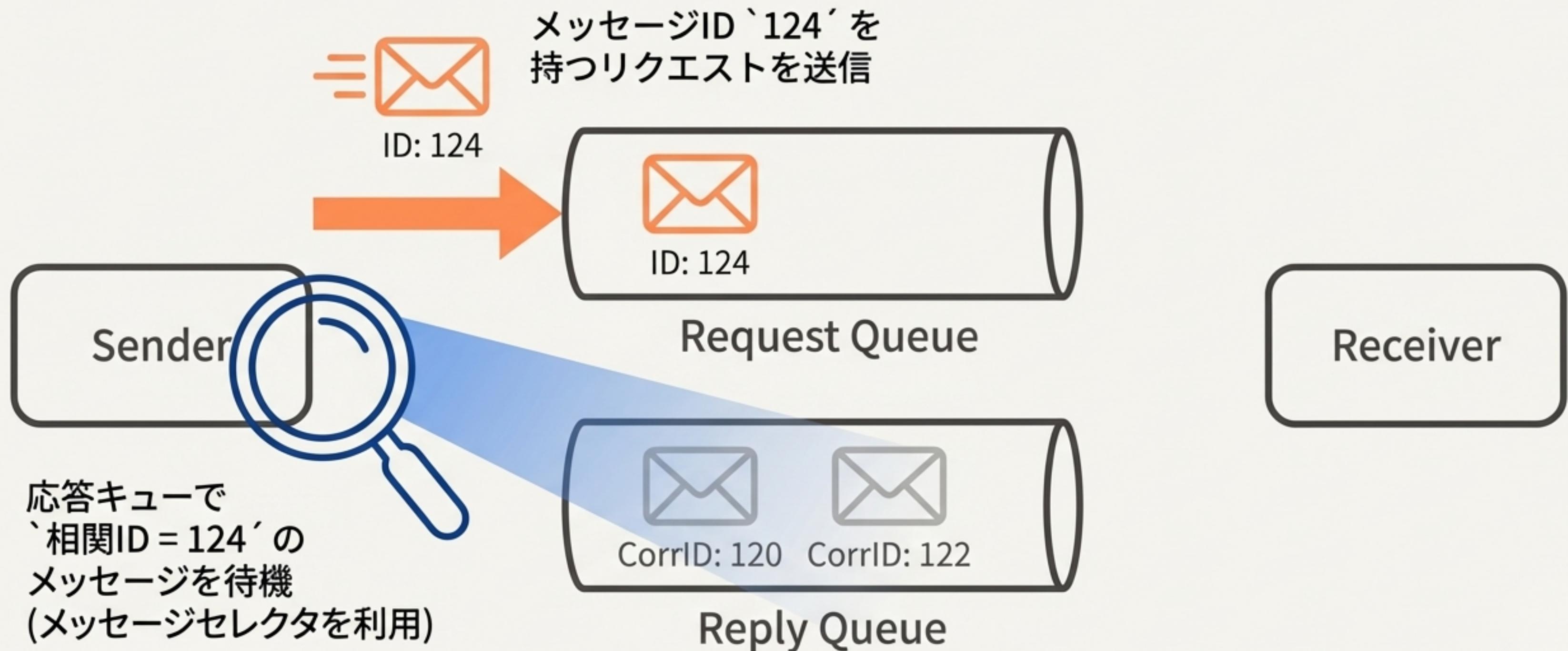
受信者からの応答がキューに届けられ、送信者はそれを受け取ります。

実装方法①：相関ID（Correlation ID）を利用した確実な応答マッチング

共有された応答キュー上で、多数のメッセージの中から「自分の」応答を特定するための手法です。各リクエストにユニークなIDを付与し、応答メッセージにそのIDを含めることで、両者を関連付けます。



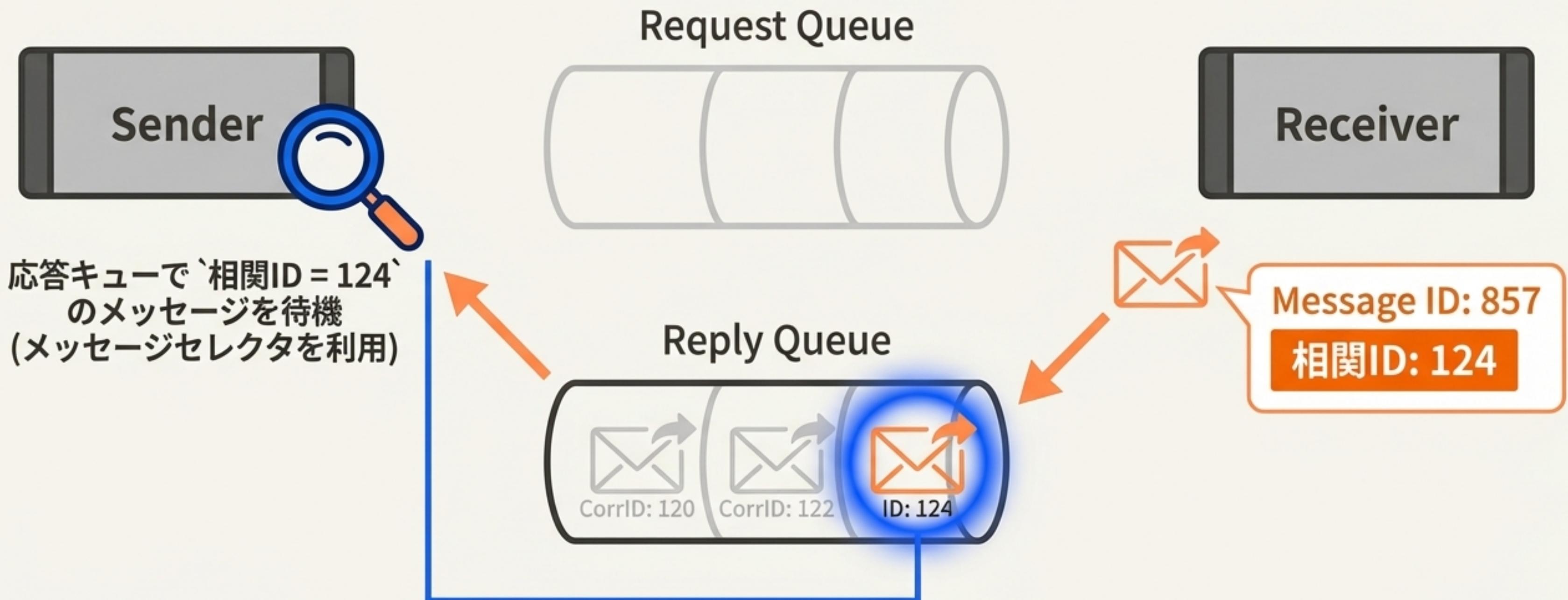
ステップ 1/3: リクエストの送信と応答の待機



ステップ 2/3: リクエストの受信と処理

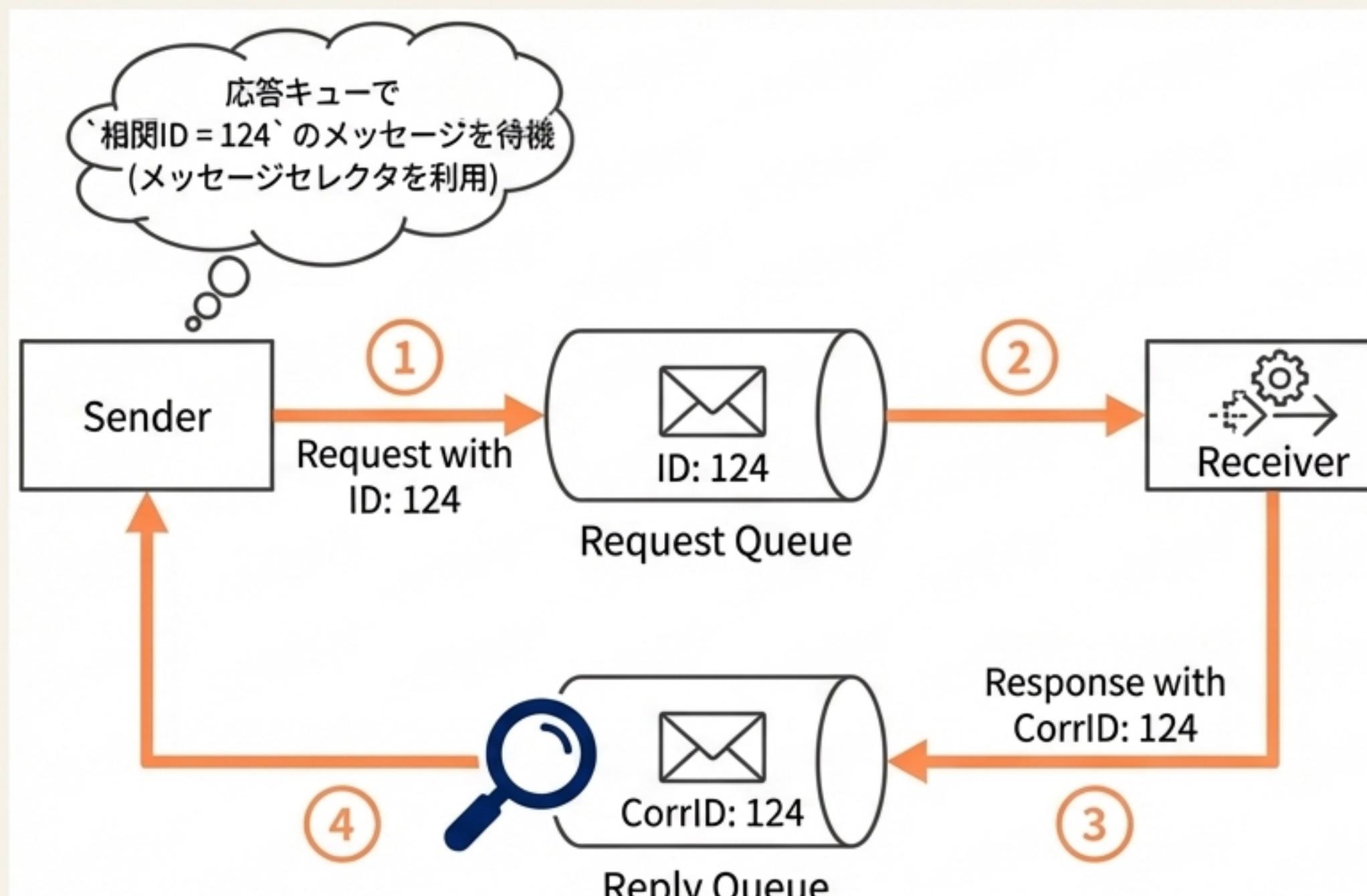


ステップ 3/3: 相関IDを設定した応答の返信



セレクタが一致するメッセージを発見し、受信成功

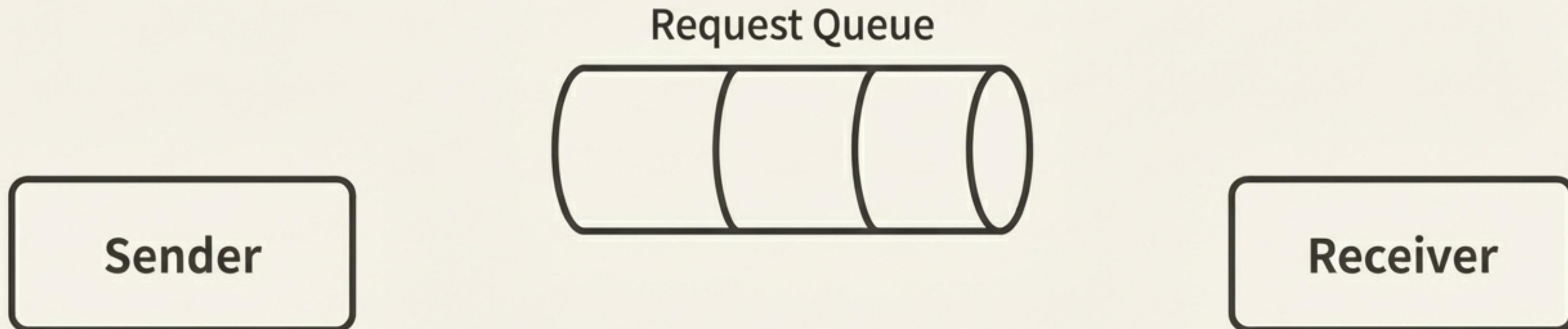
相関IDパターンのフローまとめ



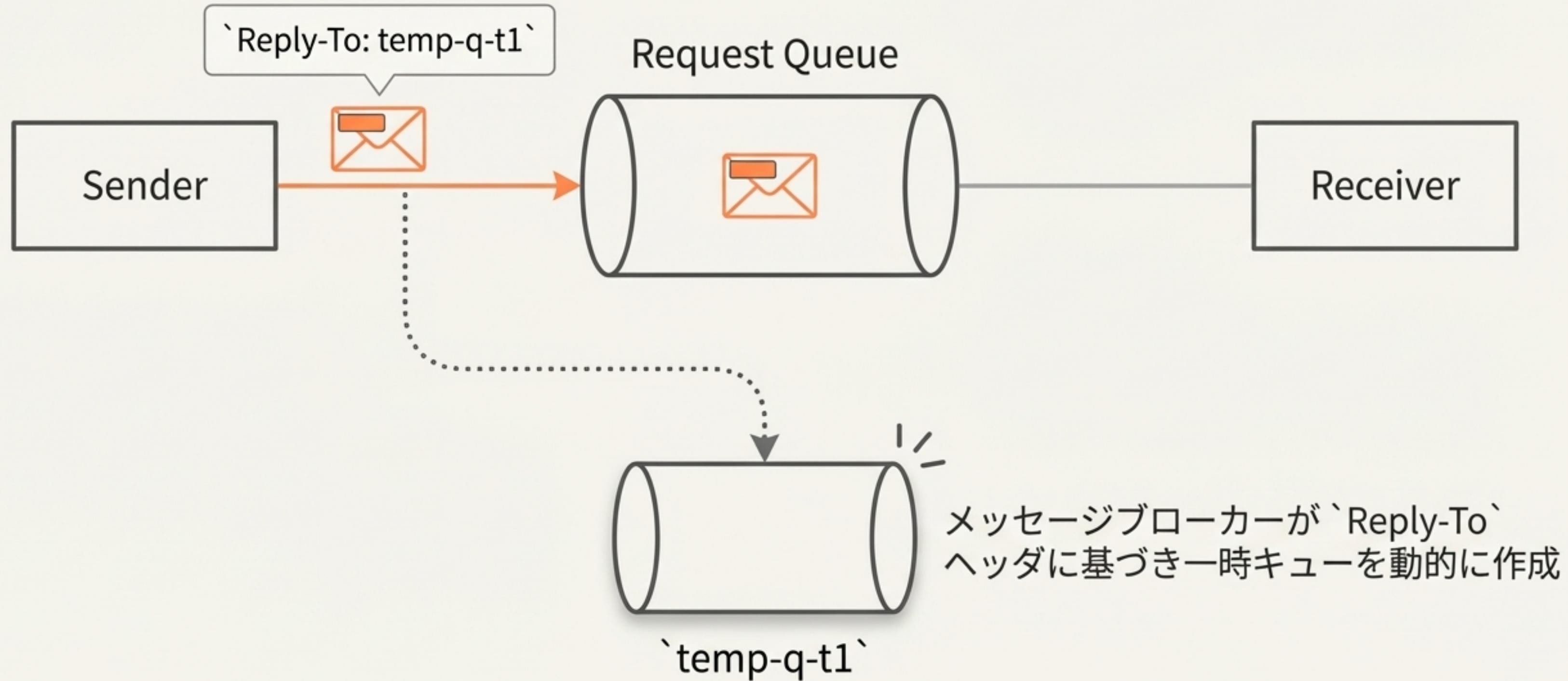
- **メッセージID (Message ID)** : 送信するリクエストに付与される一意の識別子。
- **相関ID (Correlation ID)** : 応答メッセージに設定される値。元のリクエストのメッセージIDと一致させる。
- **メッセージセレクタ (Message Selector)**: 送信者が応答キューを監視する際に使用するフィルター。「`Correlation ID = '自分のID'`」という条件で待機する。

実装方法②：一時キュー (Temporary Queue) を利用したシンプルな応答経路

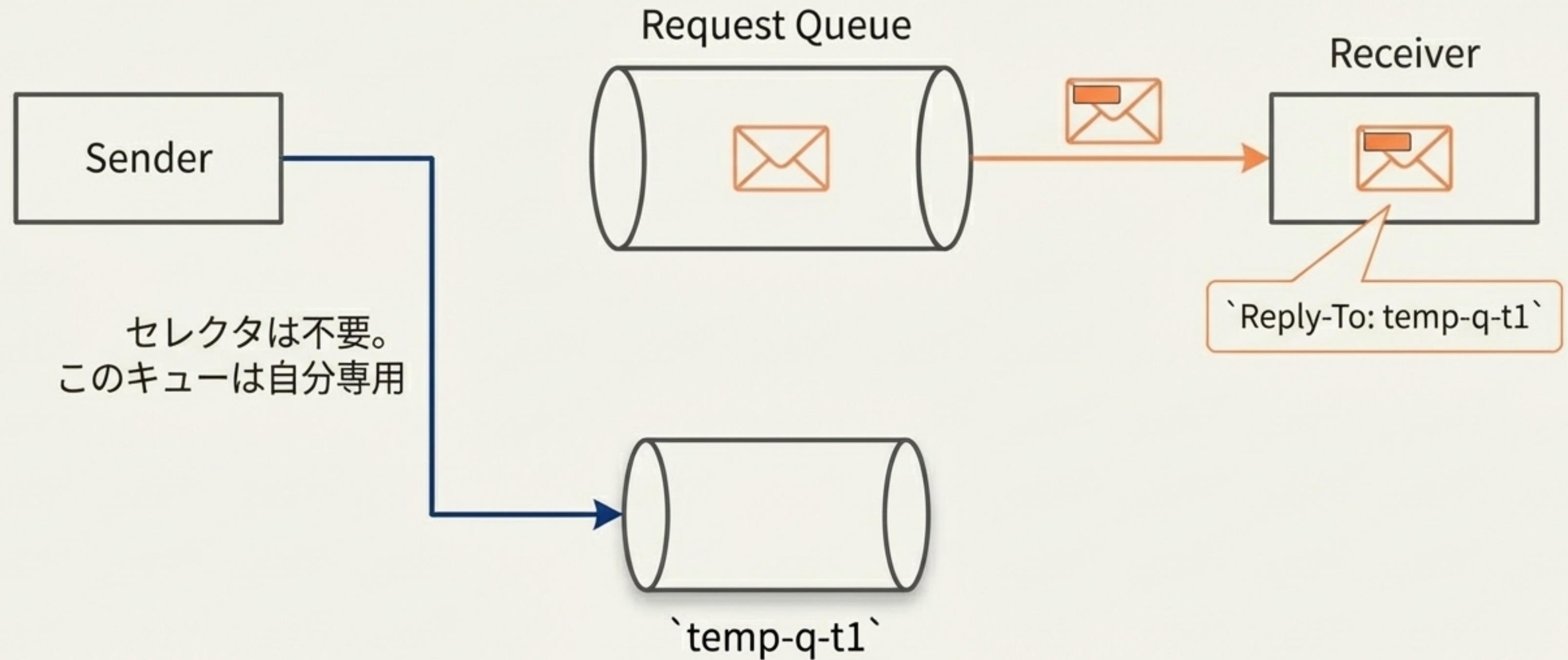
応答ごとに専用のプライベートなキューを動的に作成するアプローチ。共有キューやメッセージセレクタが不要になり、構成が大幅に簡素化されます。



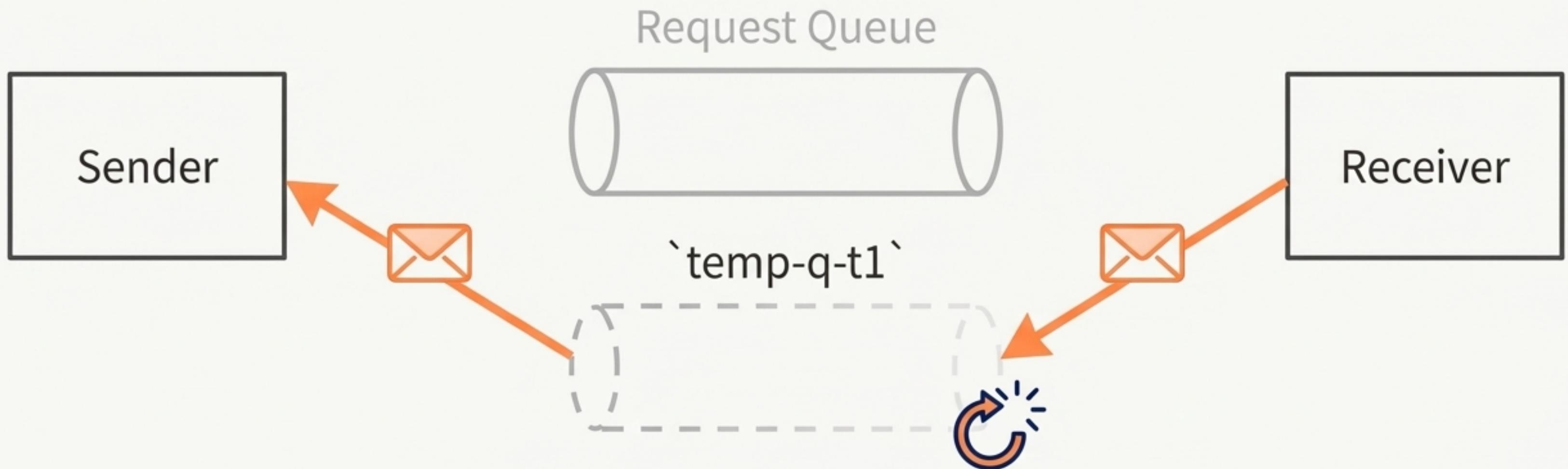
ステップ 1/3: 一時キューを指定してリクエストを送信



ステップ 2/3: 専用キューでの待機とリクエスト処理

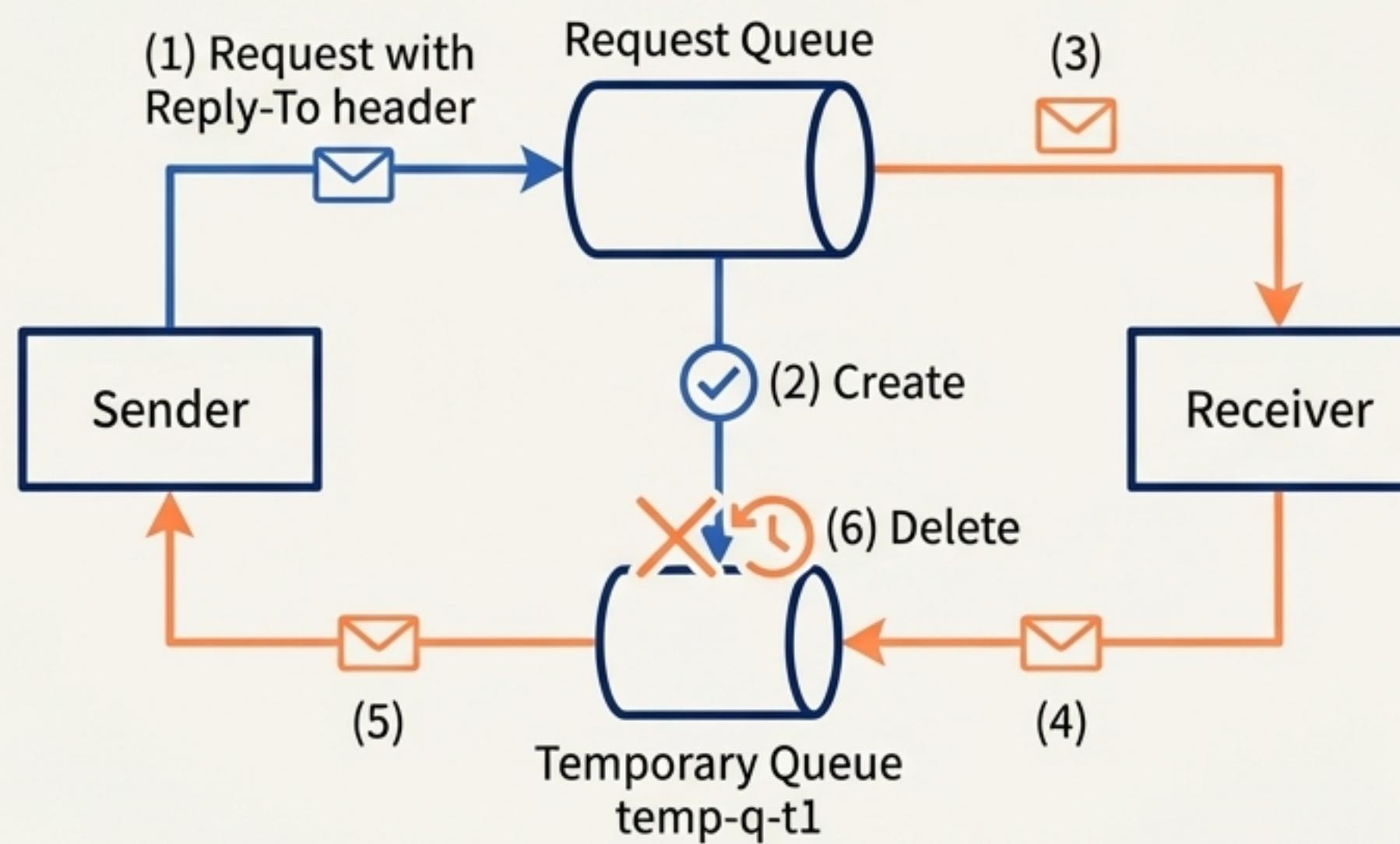


ステップ 3/3: 直接応答とキューの自動削除



応答受信後、メッセージブローカーが
一時キューをクリーンアップ

一時キューパターンのフローまとめ



- **Reply-To ヘッダ (Reply-To Header):** リクエストメッセージに含めるヘッダ。応答の送信先となる一時キューの名前を指定する。
- **一時キュー (Temporary Queue):** リクエストごとにブローカーによって動的に生成・削除される、短命なプライベートキュー。
- **メッセージブローカーの役割 (Role of the Message Broker):** キューの動的な作成と削除を担う。

2つのアプローチの比較：相関ID vs. 一時キュー

		
特徴 (Feature)	相関ID (Correlation ID)	一時キュー (Temporary Queue)
応答キュー (Reply Queue)	永続的・共有	一時的・プライベート
応答の特定方法 (Response ID)	メッセージセレクタ	キュー 자체が識別子
主要な要素 (Key Element)	`CorrelationID` フィールド	`Reply-To` ヘッダ
複雑さ (Complexity)	やや高い (Slightly - requires filtering logic)	低い (Lower - relies on broker functionality)

実装を見る：実際のコード例

このパターンが実際にどのように機能するか、以下のリポジトリで確認できます。



JMS 1.1 & 2.0 with ActiveMQ

`request-reply` のサンプルコードで相関IDの実装を確認できます。

github.com/wmr513/messaging 



RabbitMQ

`TradeGenerator`（送信者）と `TradeValidator`（受信者）のクラスで、
ブロッキング待機を伴うリクエスト/リプライの実装を確認できます。

github.com/wmr513/streaming 