E-1

SQS + Lambda という非同期処理
黄金パターン再入門

白石 一乃

アマゾン ウェブ サービス ジャパン合同会社
技術統括本部 西日本ソリューション部
ソリューションアーキテクト

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

自己紹介

白石 一乃 (しらいし いちの)

ソリューションアーキテクト

- 西日本のお客様をメインで担当
- 国内SIer出身 Webアプリ開発、プロトタイピング、PM

好きなAWSのサービス： AWS Control Tower ...etc.

運用・開発が**楽になる**サービス

イベント情報など呟いています

Twitter：@piko_san_0000

好きな言葉：Good intentions don't work, mechanisms do！

AWSアカウントの管理を楽にする

AWS Control Tower については[こちら]！

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

本セッションは…

想定受講者

- 非同期処理に興味のある方
- AWS 上での非同期処理の実装方法に興味のある方
- Amazon SQS + AWS Lambda　の組み合わせによるメリットを知りたい方

ゴール

- Amazon SQS + AWS Lambda 黄金パターン
「いいね！」 となって、やってみたくなる！！

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

アジェンダ

- 非同期処理 とは
- Amazon SQS による「キュー」の実装
- SQS + Lambda アーキテクチャ
- まとめ

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

参考リンク

- SQS、Lambda など個別のサービスについての詳細

  参考： AWS サービス別資料（「SQS」などサービス名で検索）※Blackbelt資料など

  https://aws.amazon.com/jp/events/aws-event-resource/archive/

- 「キュー」以外のメッセージングサービスの選択肢

  参考：AWS DevAx::Connect On-demand シーズン1

  第 1 回 イベント駆動アーキテクチャ入門 ～基本となる考え方から実装パターンまで～ /
  第 2 回 「疎結合」を実現するメッセージングサービスの選択と利用

  https://aws.amazon.com/jp/devax-connect-on-demand/
  https://pages.awscloud.com/devax-connect-ondemand-202101-01-jp.html

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理 とは

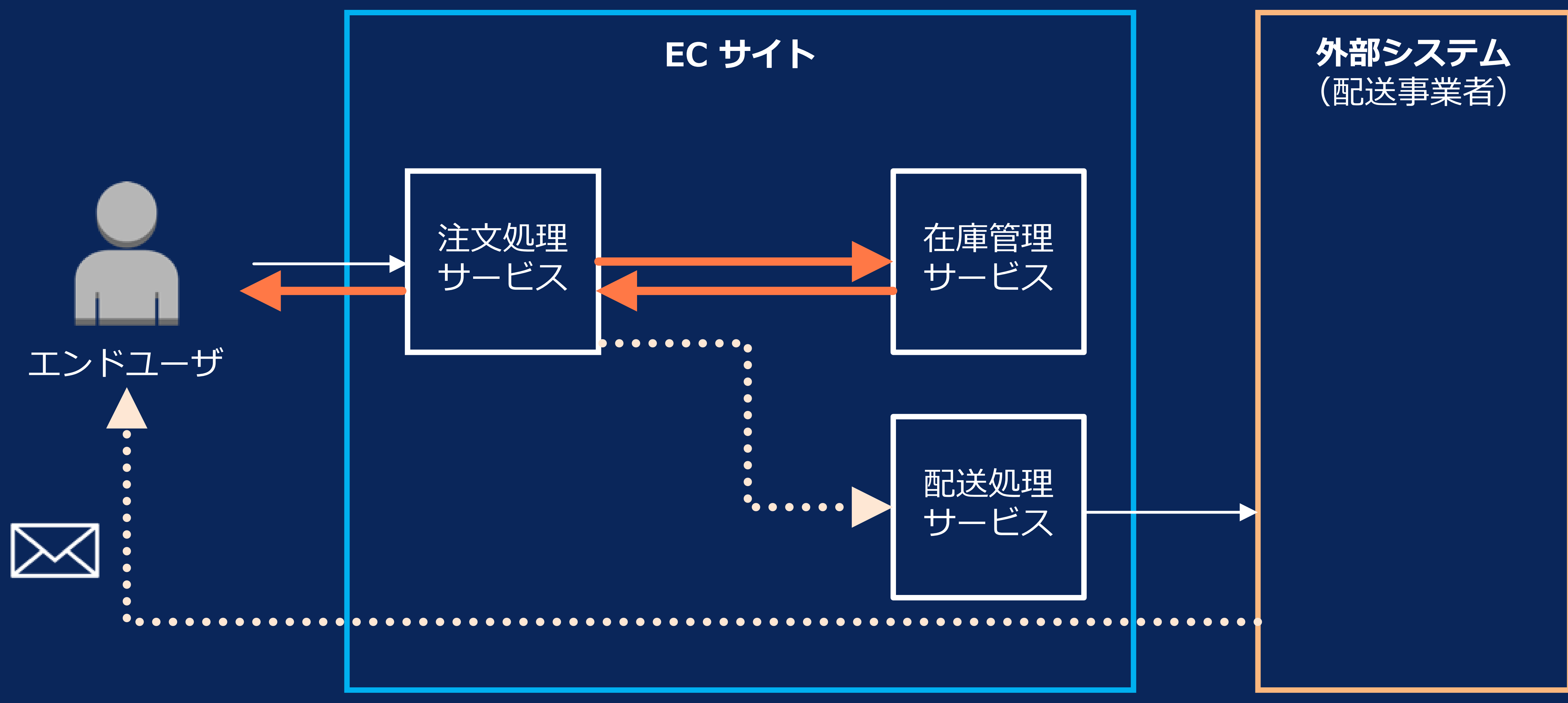
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理とは

「処理の結果を待つ必要が **“ない”** サービスの呼び出し」

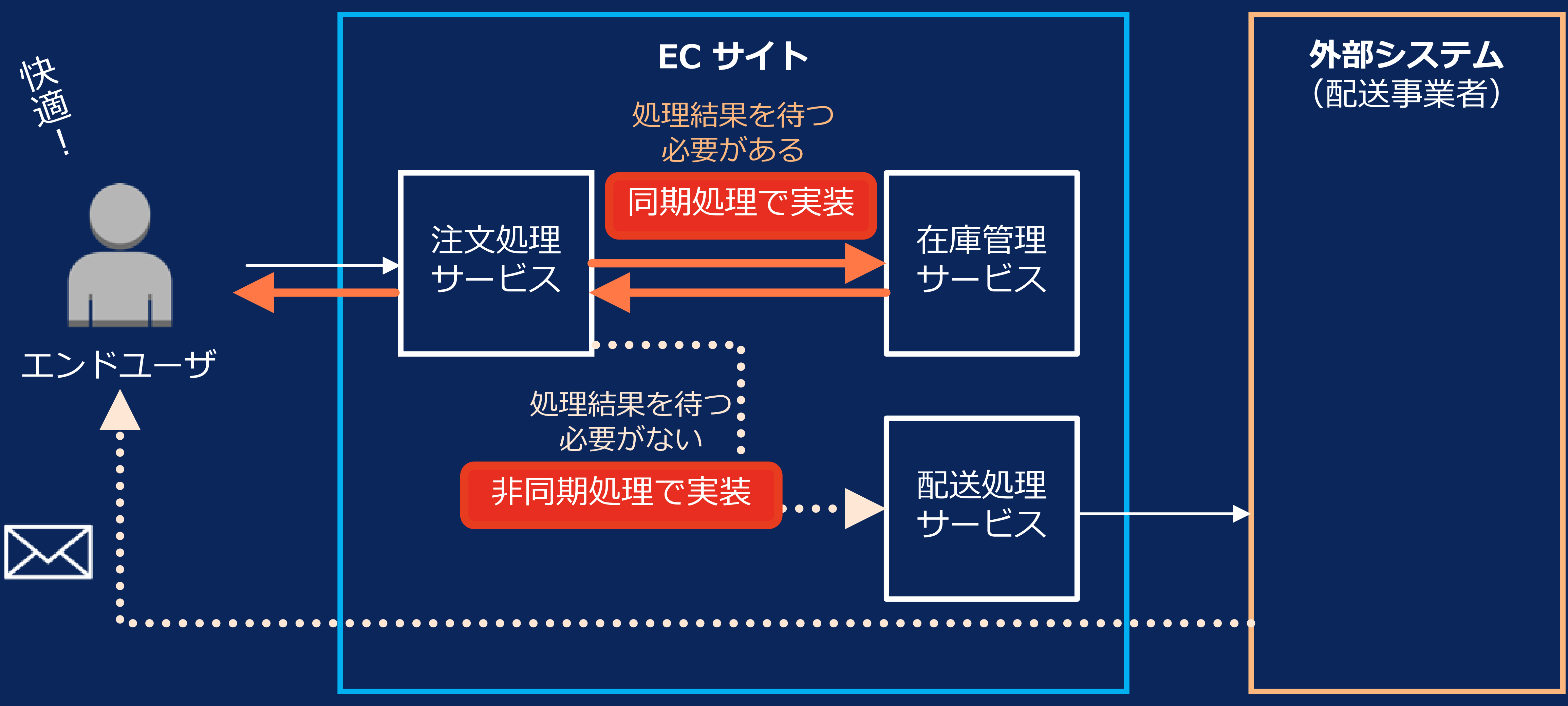
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

同期処理と非同期処理を使い分ける



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

同期処理と非同期処理を使い分ける



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理にすることのメリット

非同期処理をアプリケーション設計に取り入れる

サービス間を「疎結合」につなぐことによって以下のメリットが期待できる

1. **応答性の改善**
2. **耐障害性・可用性の向上**
3. **スループットの向上**
4. **コスト削減**

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理のメリット① 応答性の改善

同期処理

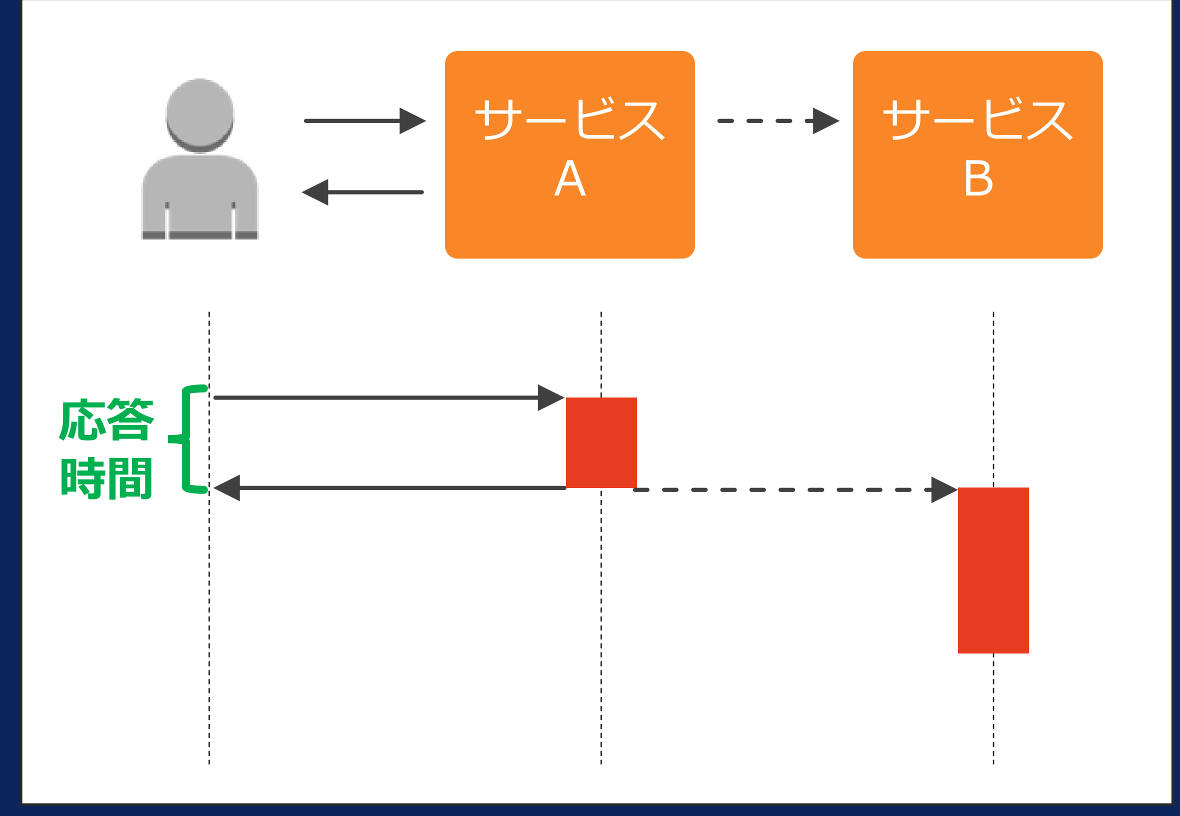
サービスA

サービスB

応答時間

待機

非同期処理

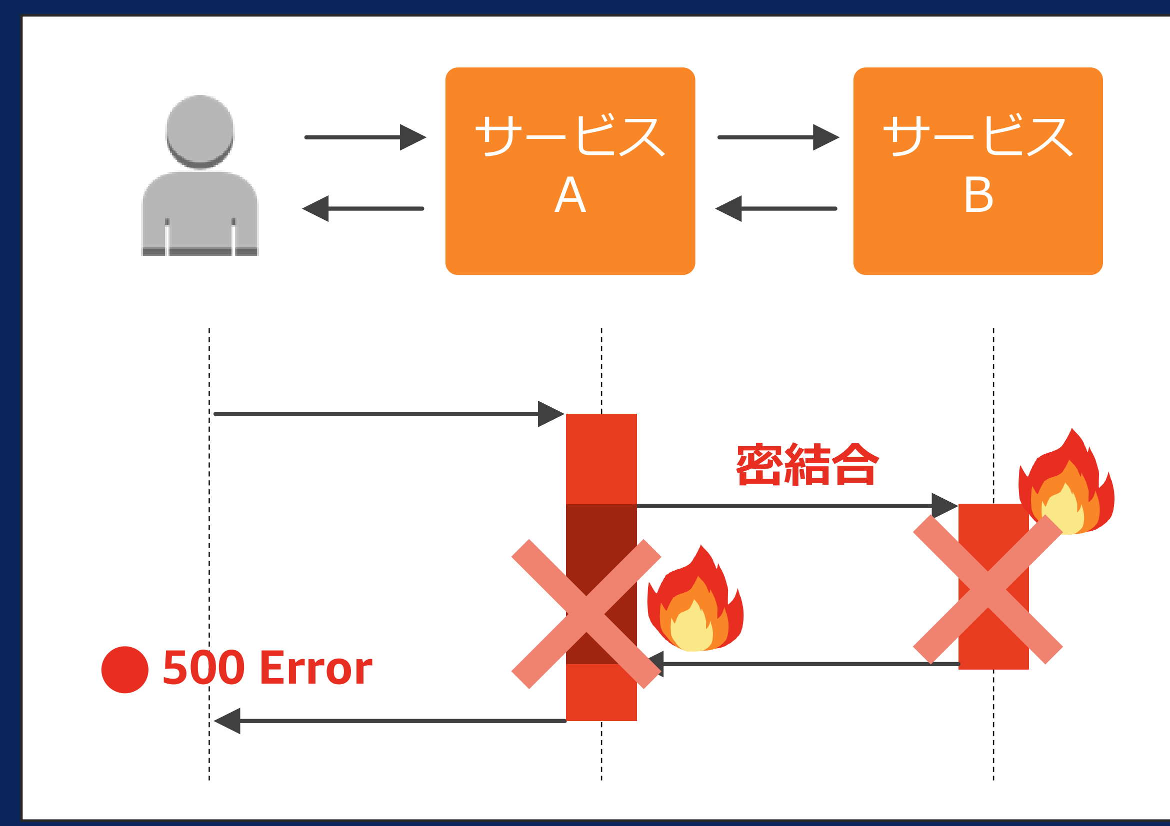


ユーザへの応答時間を短くする

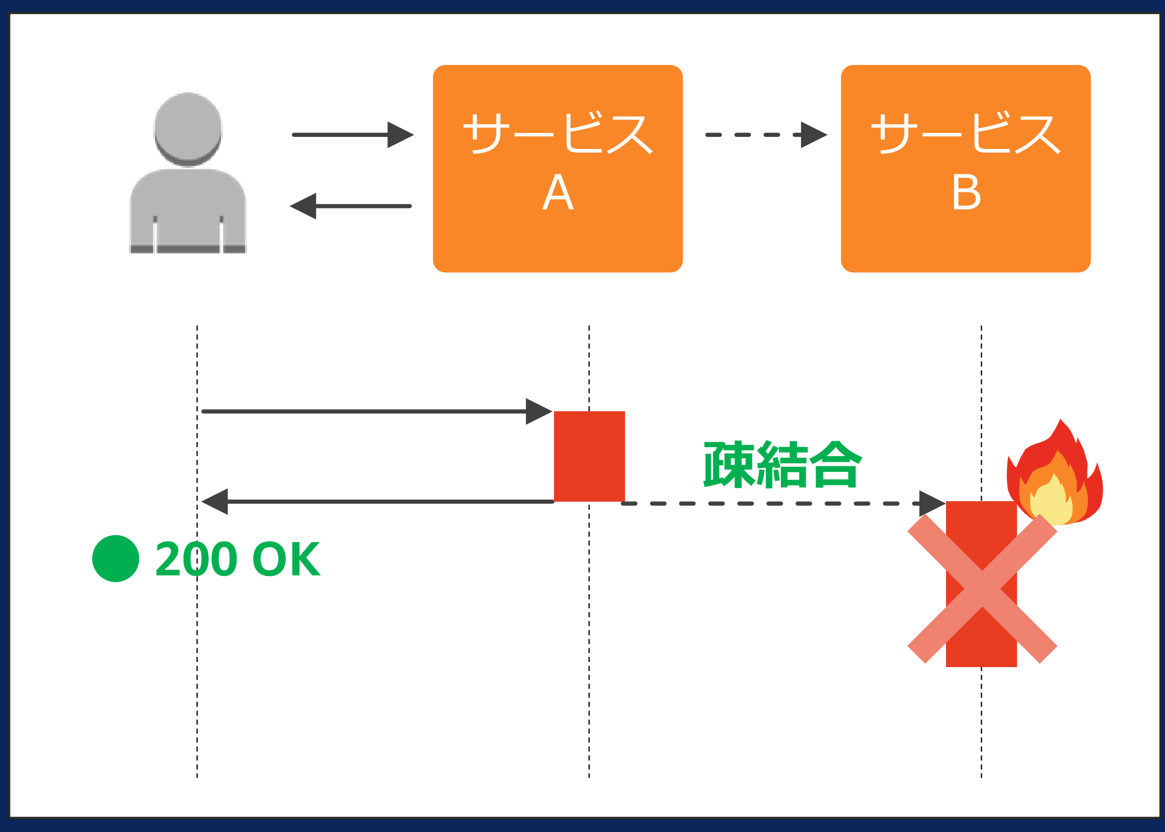
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理のメリット② 耐障害性・可用性の向上

同期処理



非同期処理



障害時の影響範囲を限定、可用性を向上

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

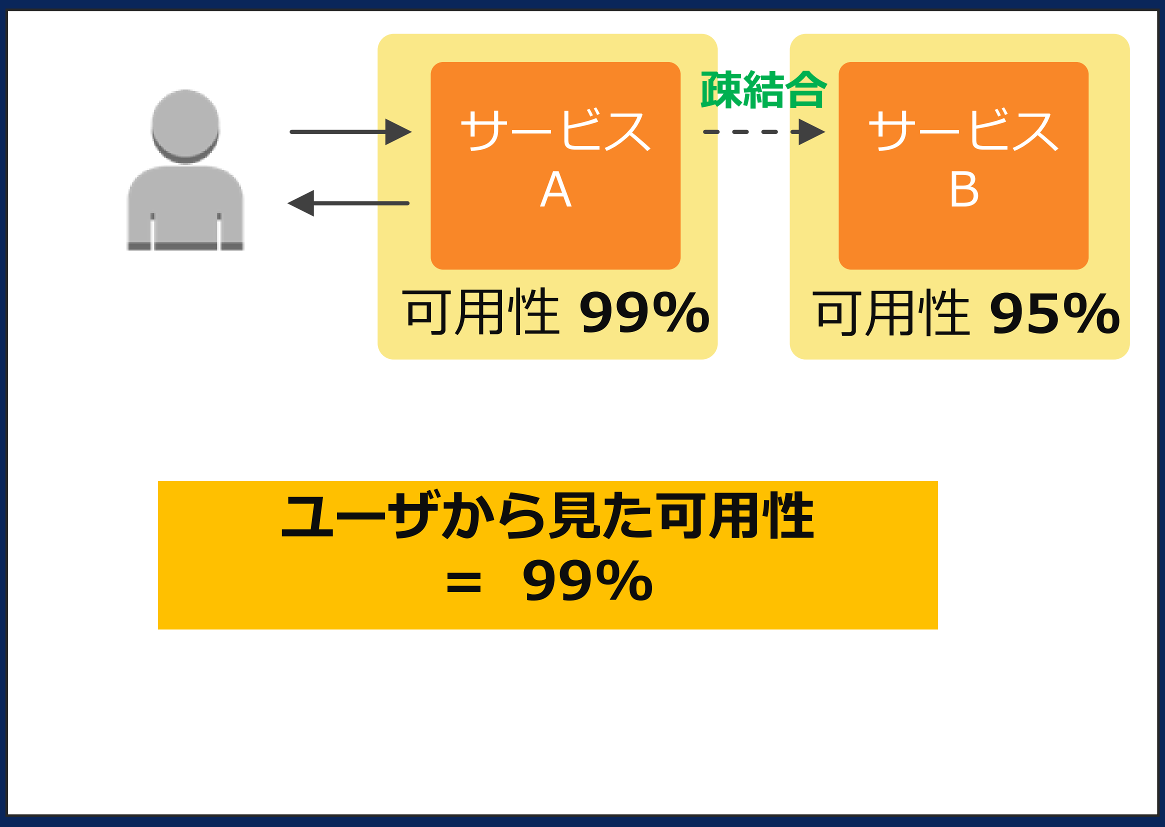
非同期処理のメリット② 耐障害性・可用性の向上

同期処理

サービス A
密結合
サービス B
可用性 **99%**　　可用性 **95%**

ユーザから見た可用性
= 0.99 × 0.95
= 0.9405
= 94%

非同期処理



障害時の影響範囲を限定、可用性を向上

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

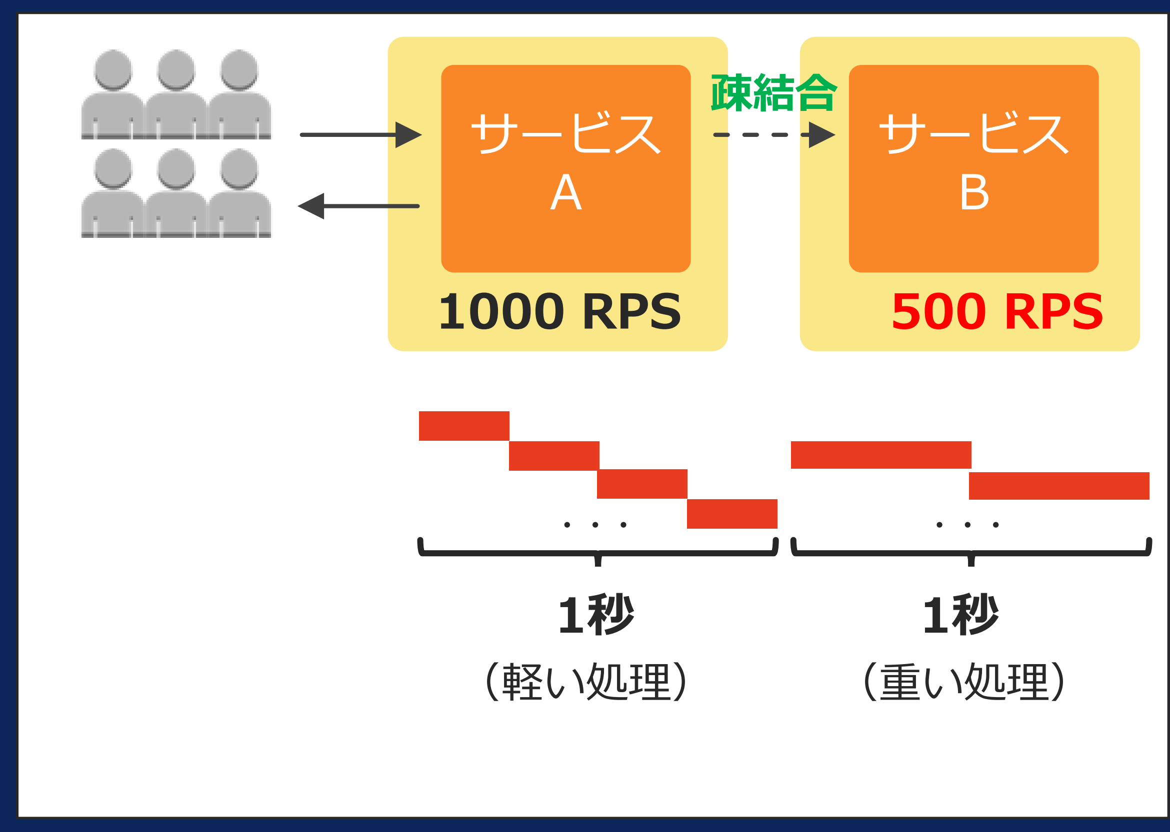
非同期処理のメリット③ スループットの向上

同期処理

サービス A
密結合
サービス B
1000 RPS
500 RPS
・・・
1秒
(軽い処理)
・・・
1秒
(重い処理)

※RPS = (Request Per Second)

非同期処理

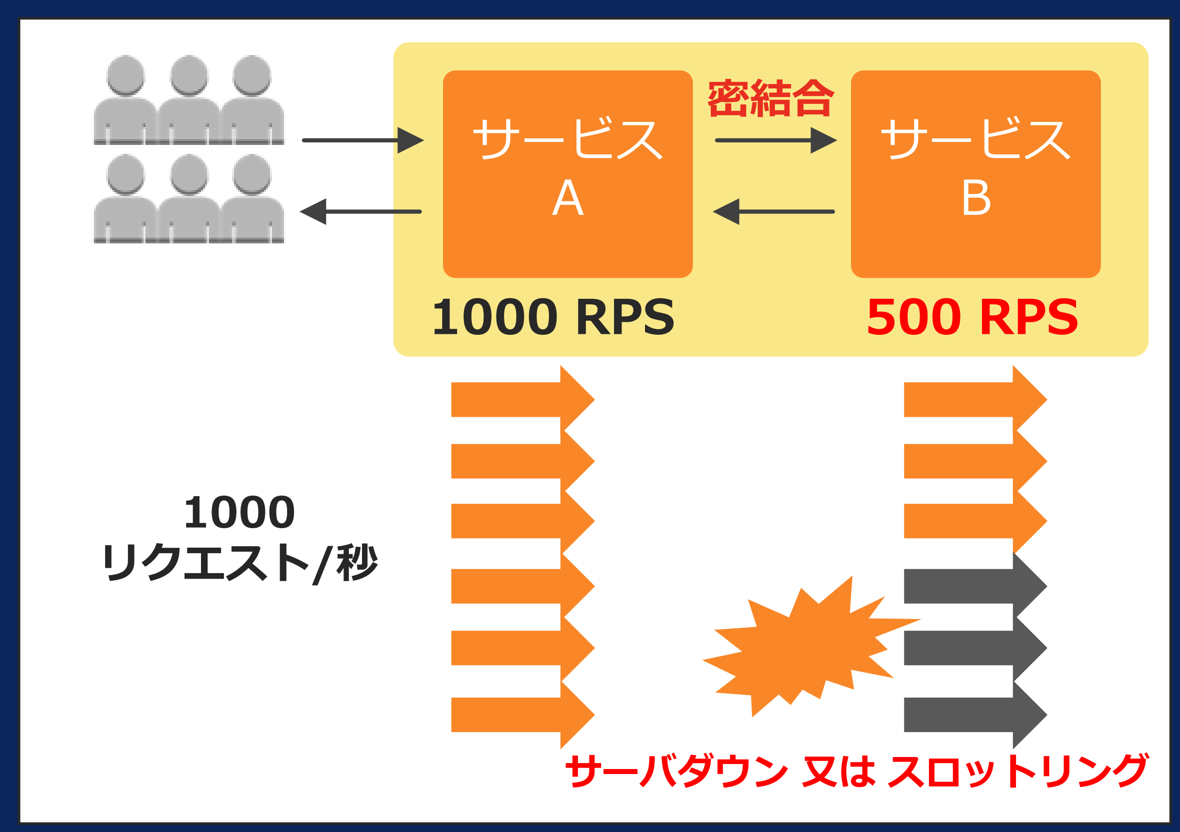


→　例えば サービスB がより長い実行時間を必要とする処理を行なう場合、
(同じプロセス・スレッド数の場合) サービスB の方が単位時間あたりの処理可能なリクエスト量は少なくなる

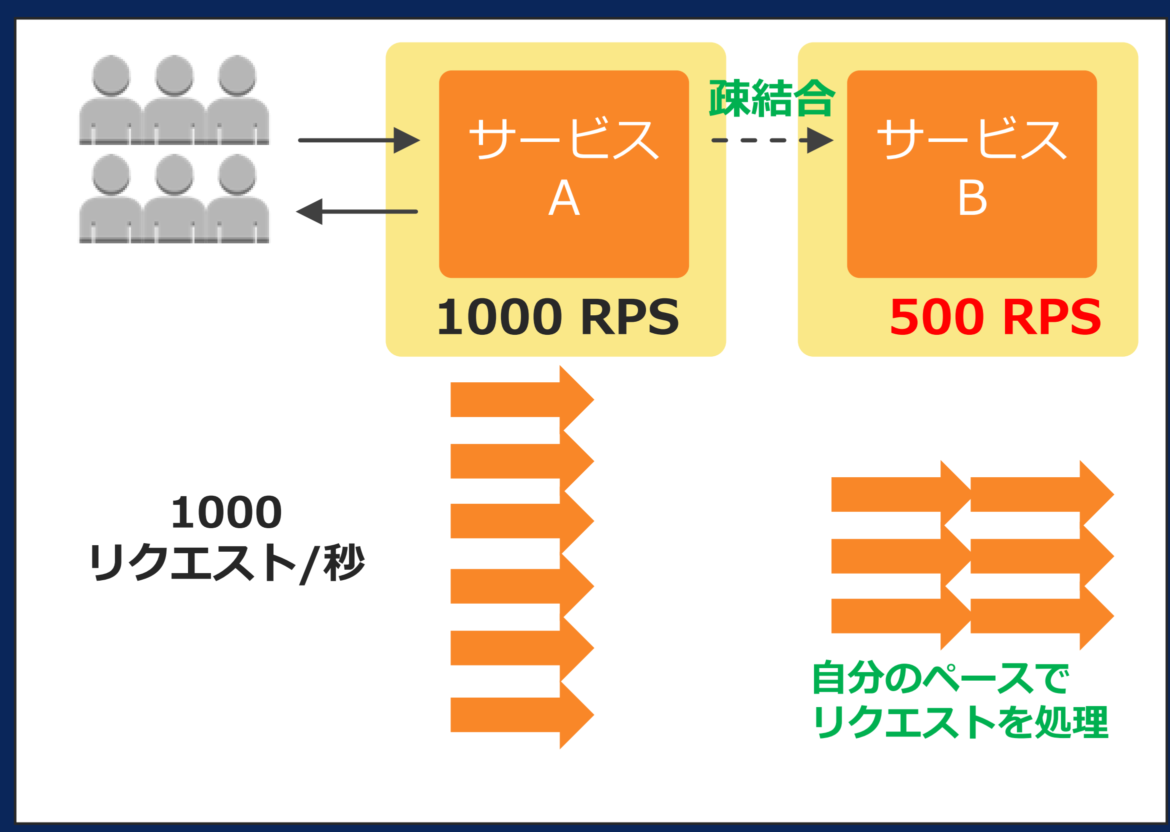
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理のメリット③ スループットの向上

同期処理



非同期処理

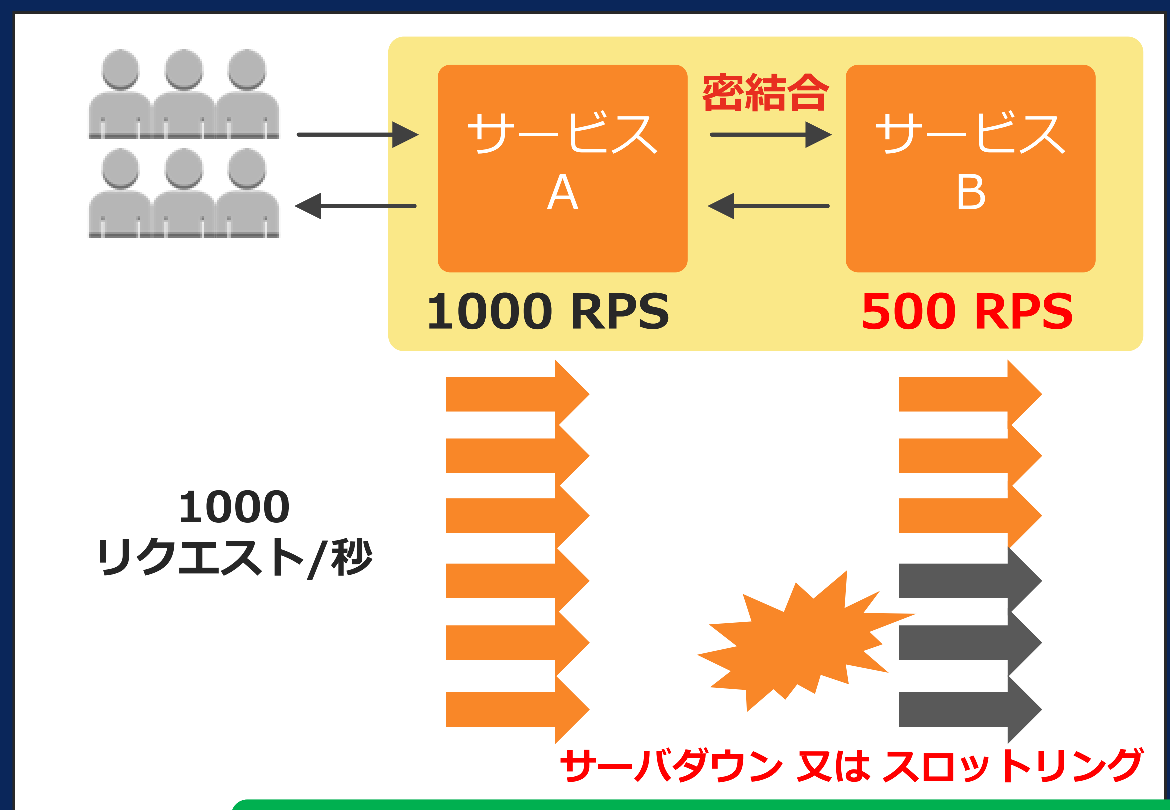


例えば サービスA が秒間 1000 リクエストを捌けたとしても、密結合で繋がる サービスB が
ボトルネックとなり システム全体の処理可能リクエスト数は、RPS の低い方のサービスに引っ張られる

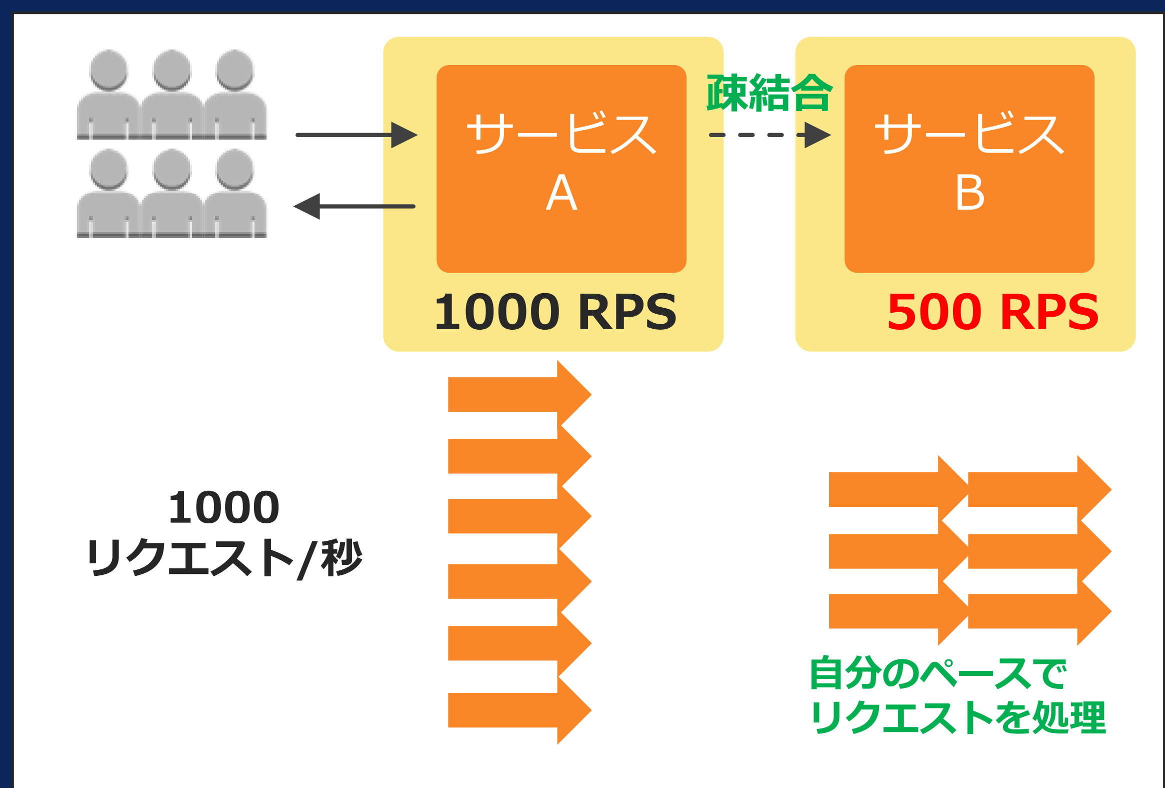
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理のメリット③ スループットの向上

同期処理



非同期処理



処理の重さに応じてサービスを分離し
スループットを向上できる

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理のメリット④ コスト削減

同期処理

非機能要件
処理可能な1秒あたりのリクエスト数：1000
ユーザから見た可用性 = 99%

コスト

サービス A

密結合

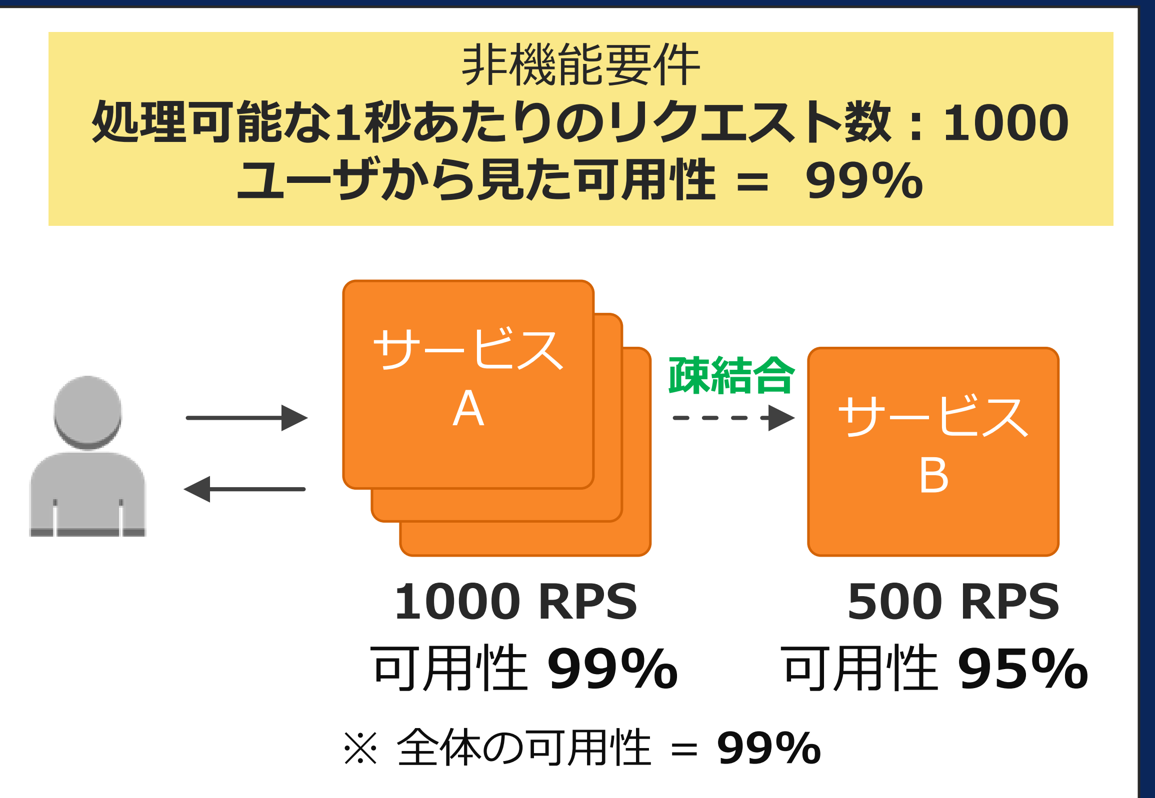
サービス B

1000 RPS
可用性 **99%**

1000 RPS
可用性 **99.9%**

※ 全体の可用性 = **99 × 99.9 = 98.9%**

非同期処理



コンポーネントごとに最適なリソース割当が可能

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理にすることのメリット

非同期処理をアプリケーション設計に取り入れる

サービス間を「疎結合」につなぐことによって以下のメリットが期待できる

① **応答性の改善**
② **耐障害性・可用性の向上**
③ **スループットの向上**
④ **コスト削減**

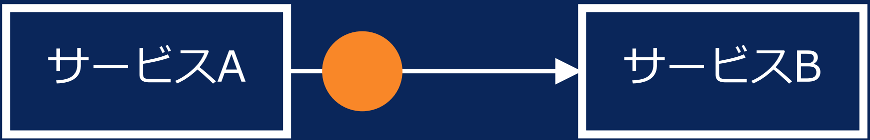
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon SQS による「キュー」の実装

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

サービス間を繋ぐ

同期処理

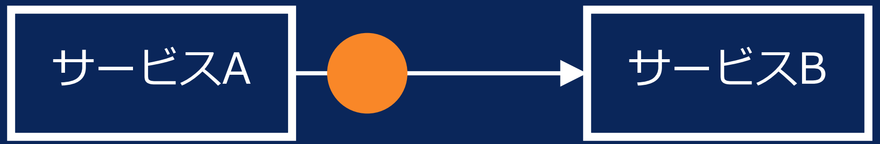


非同期処理

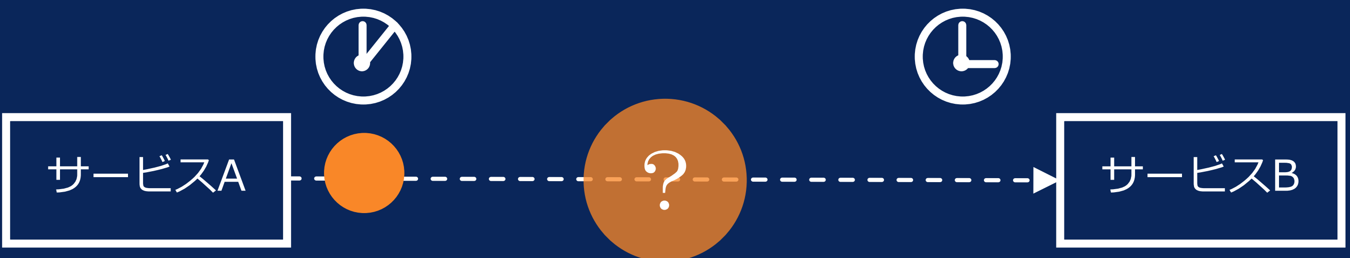
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

サービス間を繋ぐ

同期処理



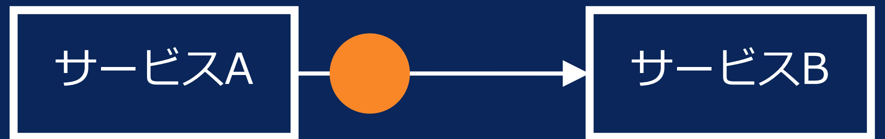
非同期処理



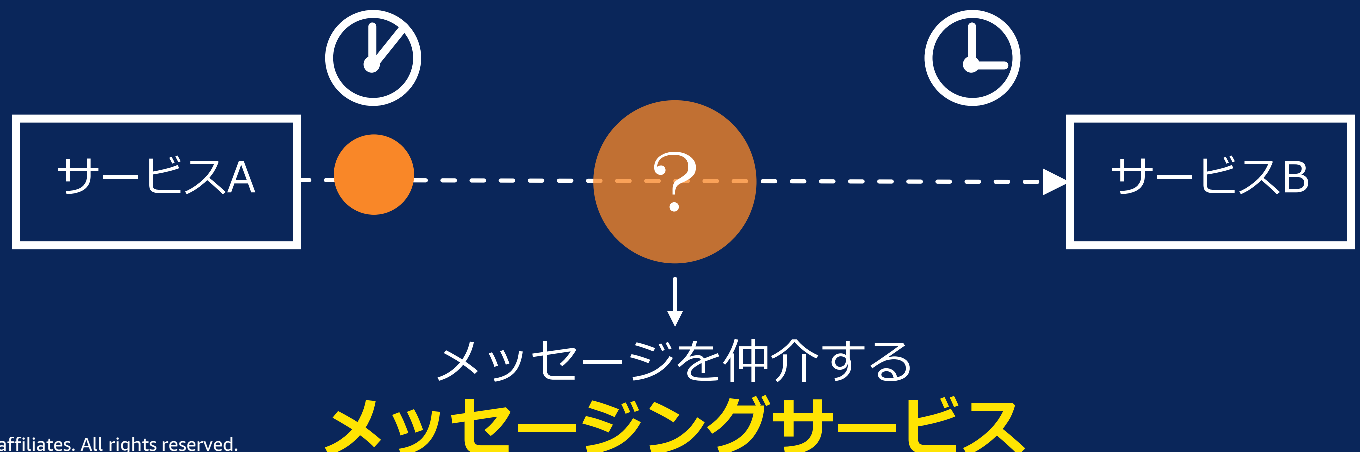
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

サービス間を繋ぐ

同期処理

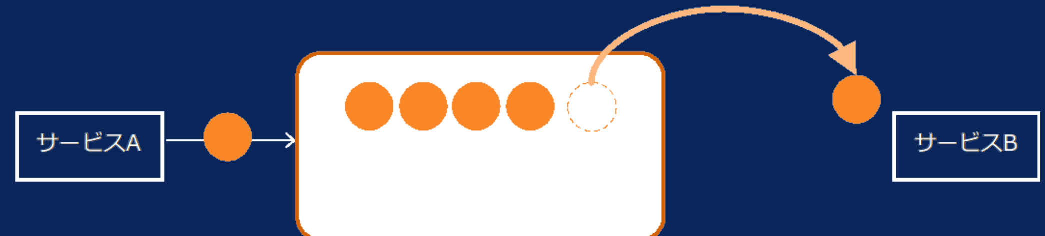


非同期処理



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理を実装するために利用可能



キュー(Queue)

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

（参考）メッセージングサービスのモデル大別

モデル1: キュー

→ ポイントツーポイント、ワーカーキュー

モデル2: ストリーム

→ データストリーム、イベントストリーム

モデル3: トピック

→ ルーティング

モデル4: バス

→ ハブ、イベントバス、エンタープライズサービスバス (ESB)

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キュー(Queue)

キュー

自分でメッセージを
取りに行く

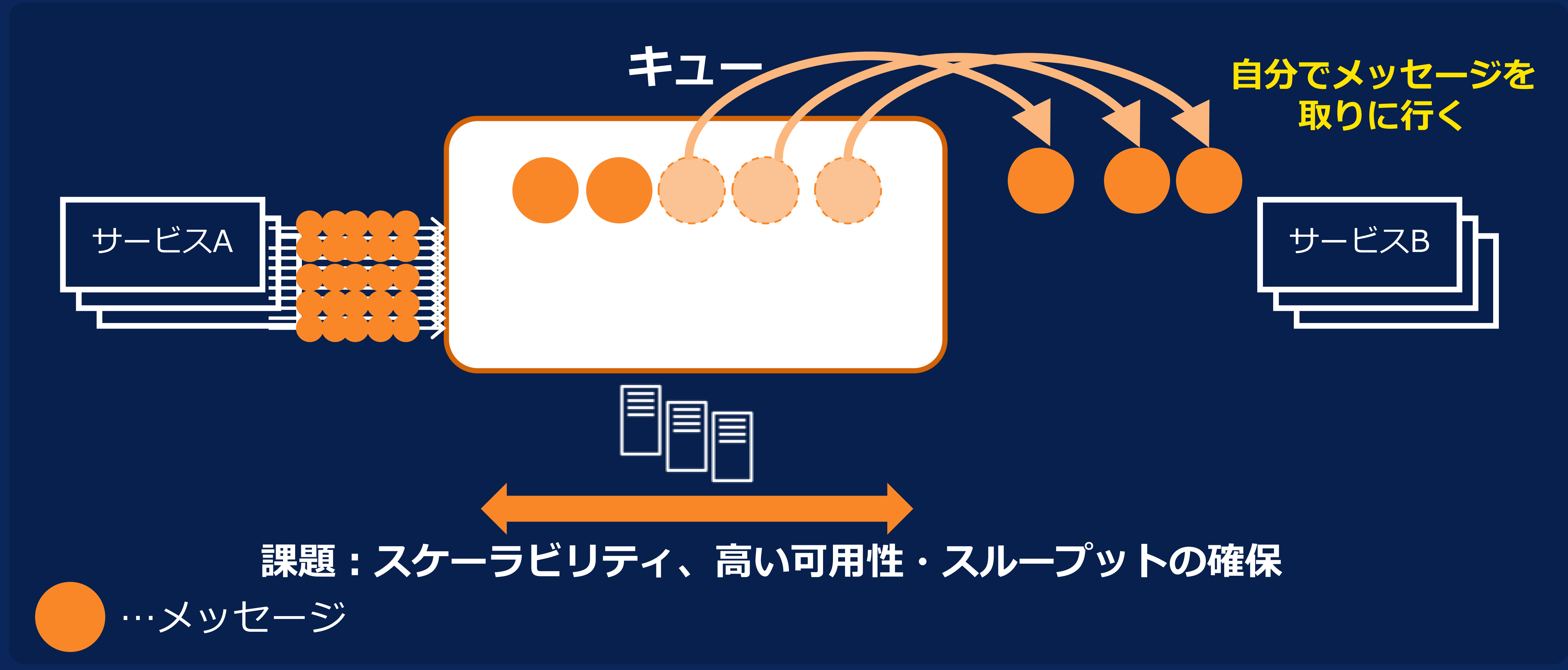
サービスA

サービスB

…メッセージ

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キュー(Queue)

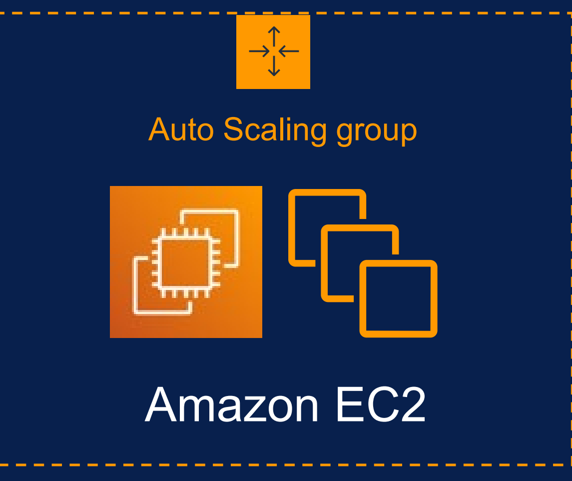


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

（課題）キューを自前で作ろうと思うと・・・

オンプレ上の仮想サーバ

大量のリクエストに備えるための
スケーラビリティの確保が課題



EC2とオートスケーリング

新規インスタンスの起動時間が必要
スパイクなどの急激なリクエストの増加への対応が課題

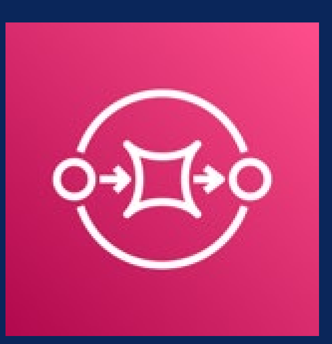
※ FargateやLambdaも同様、スケーリング時の起動時間をゼロにはできない

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

（課題）キューを自前で作ろうと思うと・・・

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWSのマネージドサービスを利用することで、可用性・スケーラビリティを実現



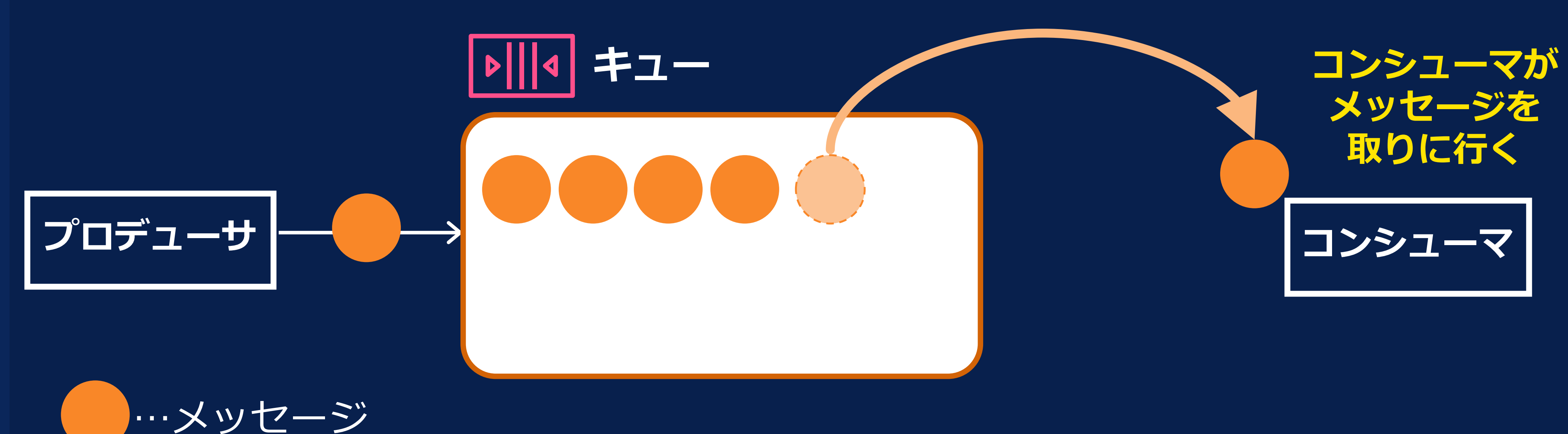
Amazon SQS

- **フルマネージド型**（サーバ管理不要）の**メッセージキューサービス**
- **ほぼ無制限のTPS**（Transactions per second）
- **利用した分だけの従量課金**（API実行回数＋データ転送量）
- 分散キューモデルによる高い可用性を提供

料金例：（標準キュー、東京リージョン）
100万リクエストで0.40USD

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Amazon SQSの構成



メッセージ…最大 256KB のテキスト（XML, JSON, フォーマットなし）

メッセージは最大14日間保持可能

キューに保存できるメッセージ数 無制限 ∞

（1 秒あたりのトランザクション数） ：ほぼ 無制限

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

キュー

メッセージの送信

ポーリング

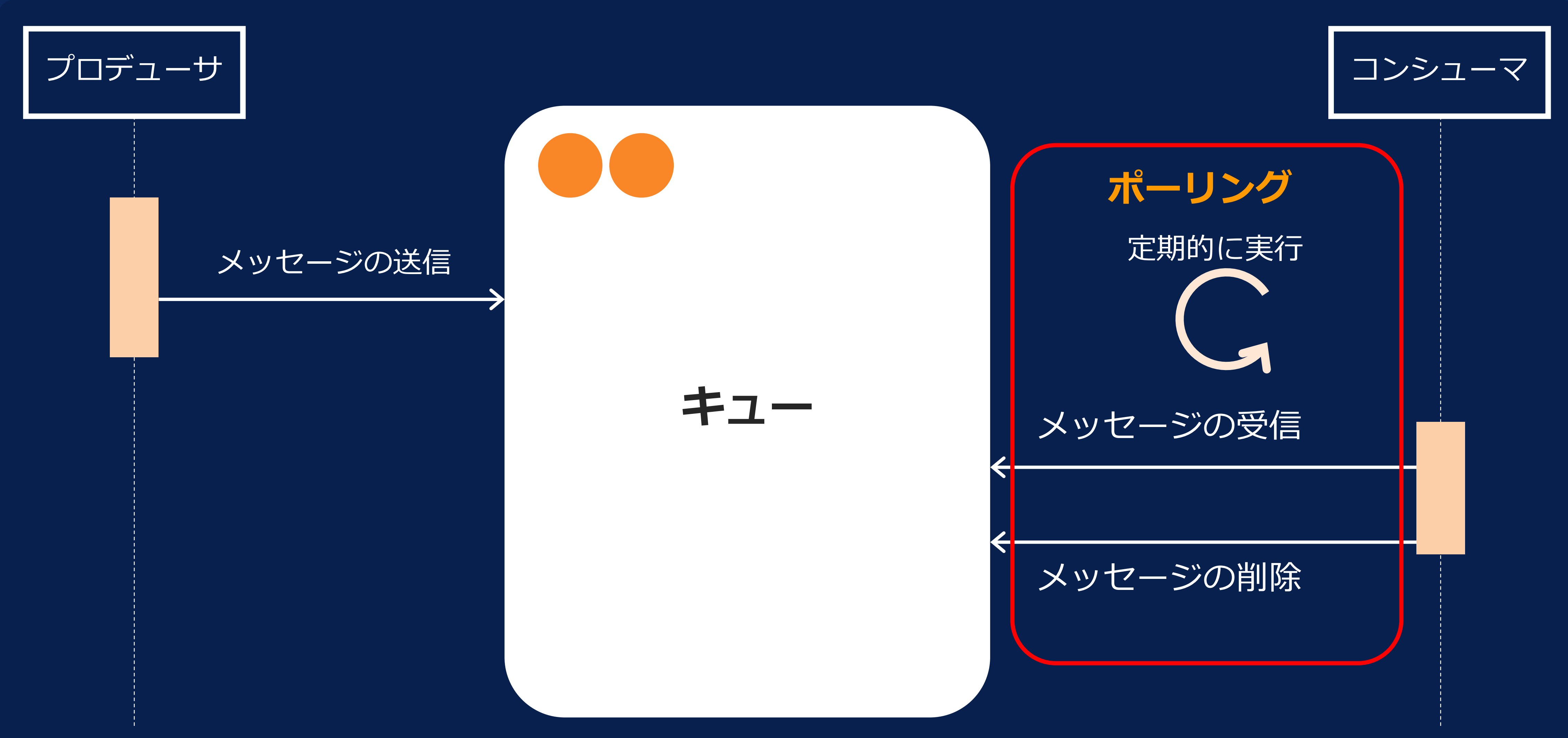
定期的に実行

メッセージの受信

メッセージの削除

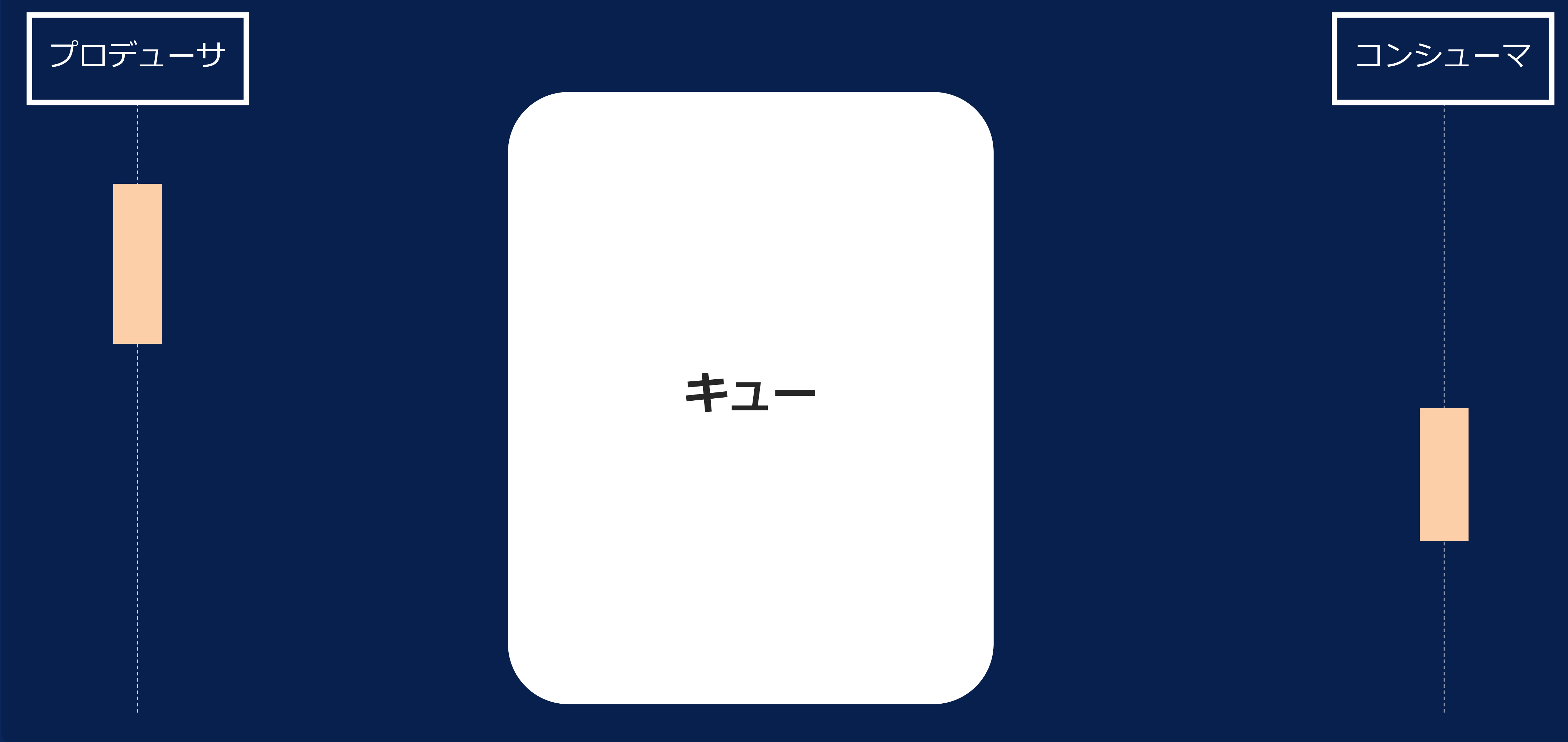
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み



API Reference
https://docs.aws.amazon.com/ja_jp/AWSSimpleQueueService/latest/APIReference/API_Operations.html

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

SendMessage

メッセージの送信

キュー

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

SendMessageBatch
複数メッセージの送信

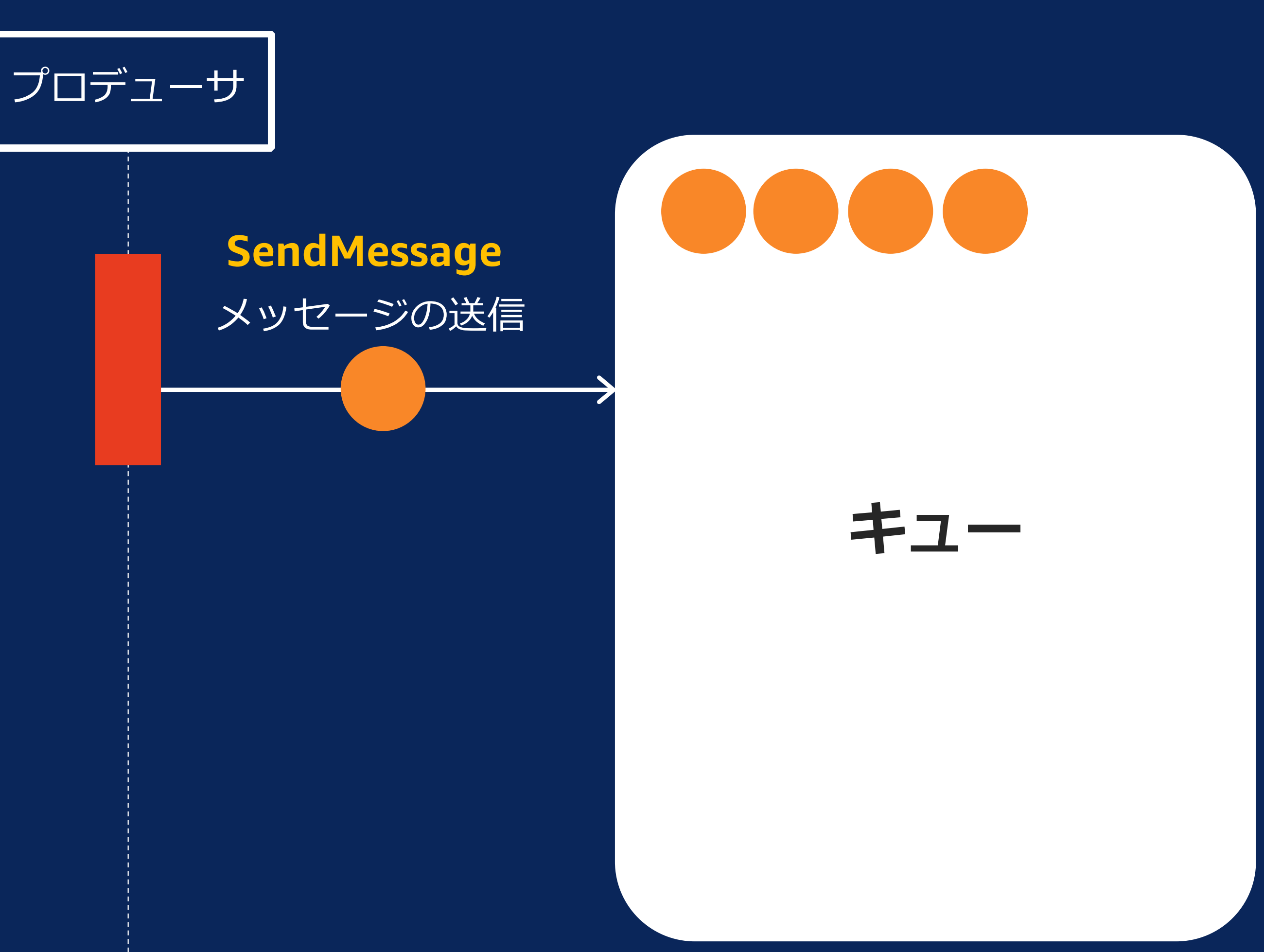
（最大10個）

キュー

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

メッセージ送信側の実装例

```
# SQS client の作成
sqs = boto3.client('sqs’)

# キューURL（宛先となるキュー）
queue_url = ‘SQS_QUEUE_URL’

# キューにメッセージを送信
response = sqs.send_message(
    QueueUrl=queue_url,
    MessageBody='Hello World ! '
)
```



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

メッセージの送信

キュー

メッセージを送信する側の
責務はここまで

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

キュー

ここからキューの中の
メッセージを処理する**責務**は
受信側に移る

メッセージの受信

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

キュー

メッセージの受信

aws

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

プロデューサ

コンシューマ

キュー

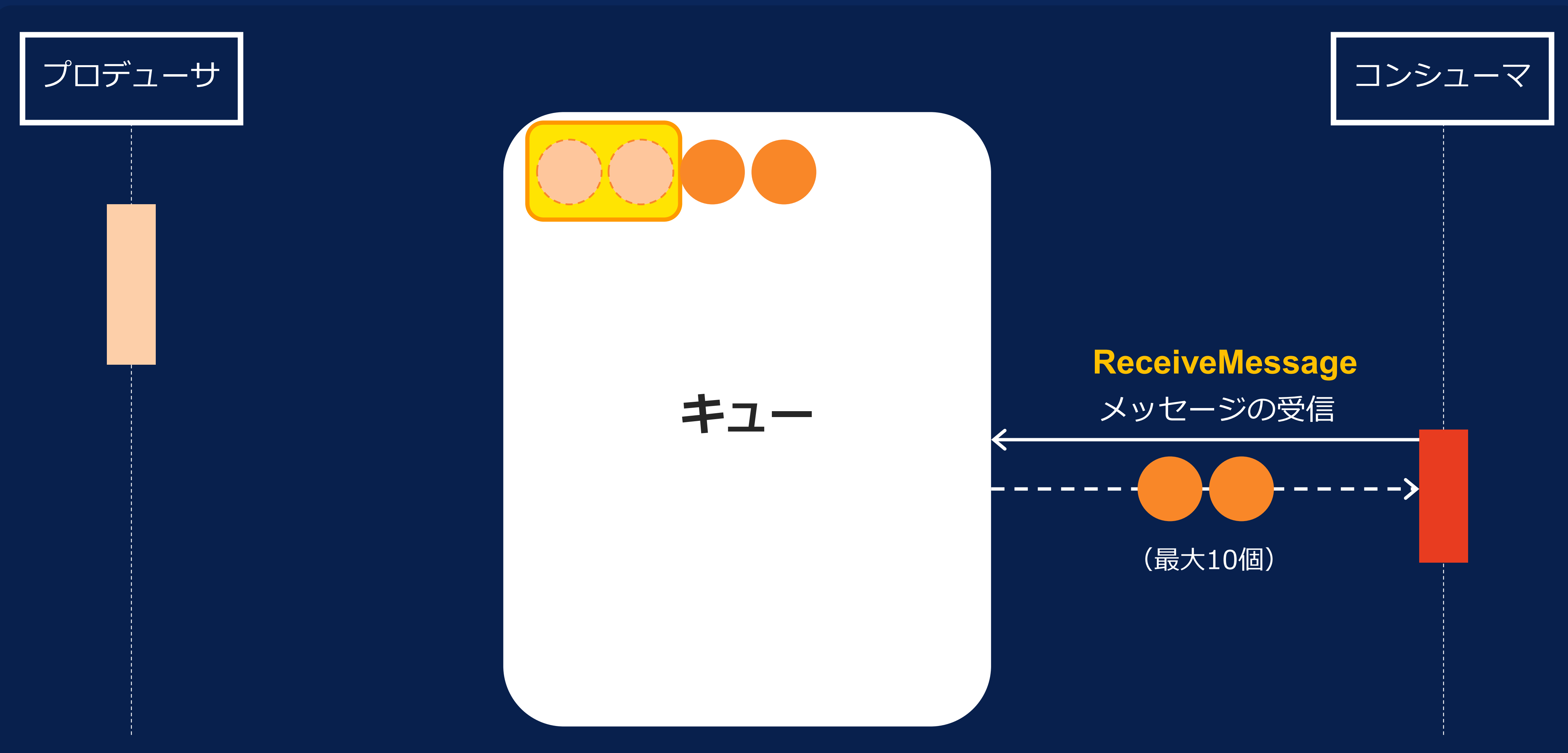
ReceiveMessage

メッセージの受信

（最大10個）

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

可視制タイムアウトの
時間内に後続処理の
完了を目指す
&
冪等性を担保した処理
にする（理想）

可視制タイムアウト で
設定された期間は
受信できなくなる

キュー

ReceiveMess

メッセージの受信

（最大10個）

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

キュー

メッセージの削除

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

キュー

メッセージの削除
DeleteMessage
× n 回

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

キュー

複数メッセージの削除
DeleteMessageBatch
（最大10個）

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

キュー

ポーリング

定期的に実行

メッセージの受信

メッセージの削除

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

メッセージ受信側の実装例

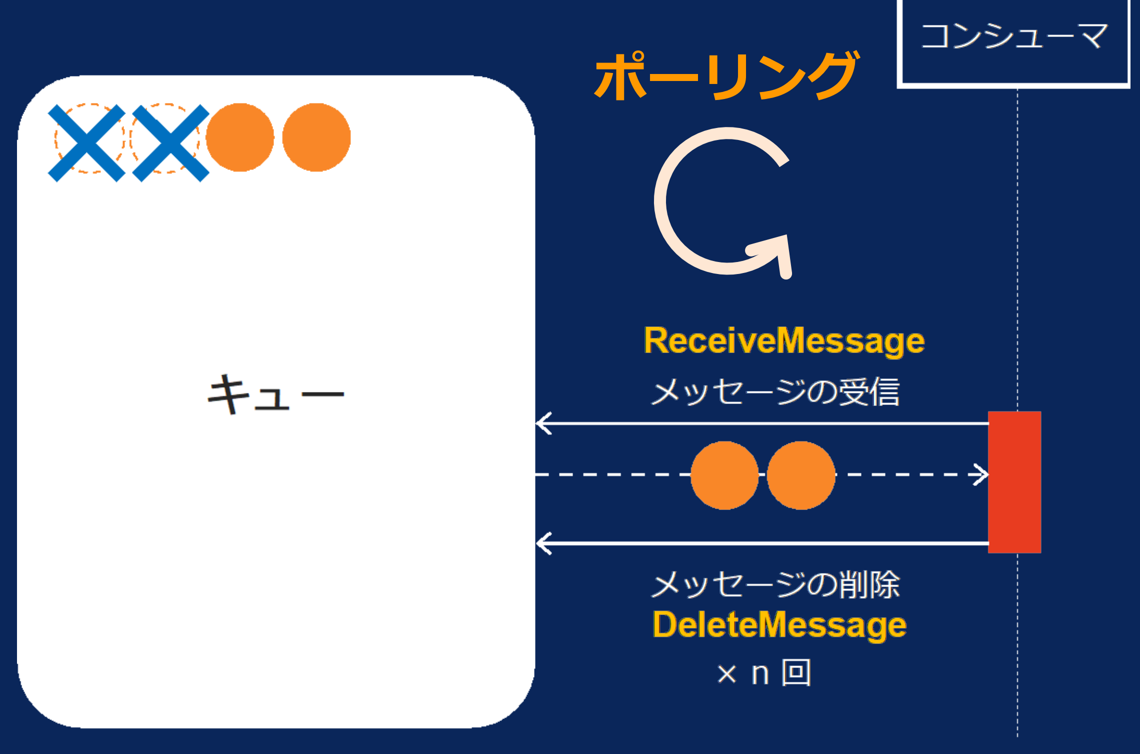
```python
# SQS client の作成
sqs = boto3.client('sqs')
# キューURL（宛先となるキュー）
queue_url = 'SQS_QUEUE_URL'
# キューからメッセージを受信
response = sqs.receive_message(
    QueueUrl=queue_url,
    MaxNumberOfMessages=10
)
# メッセージの取得
for message in response.get("Messages", []):
    message_body = message["Body"]

    # メッセージを使った処理
    print(message_body)

    # メッセージ削除用の受信ハンドルの取得
    receipt_handle = message['ReceiptHandle']

    # キューから受信したメッセージを削除
    sqs.delete_message(
        QueueUrl=queue_url,
        ReceiptHandle=receipt_handle
    )
```

さらに自前で必要

定期的に実行するための仕組み（常駐プロセス）

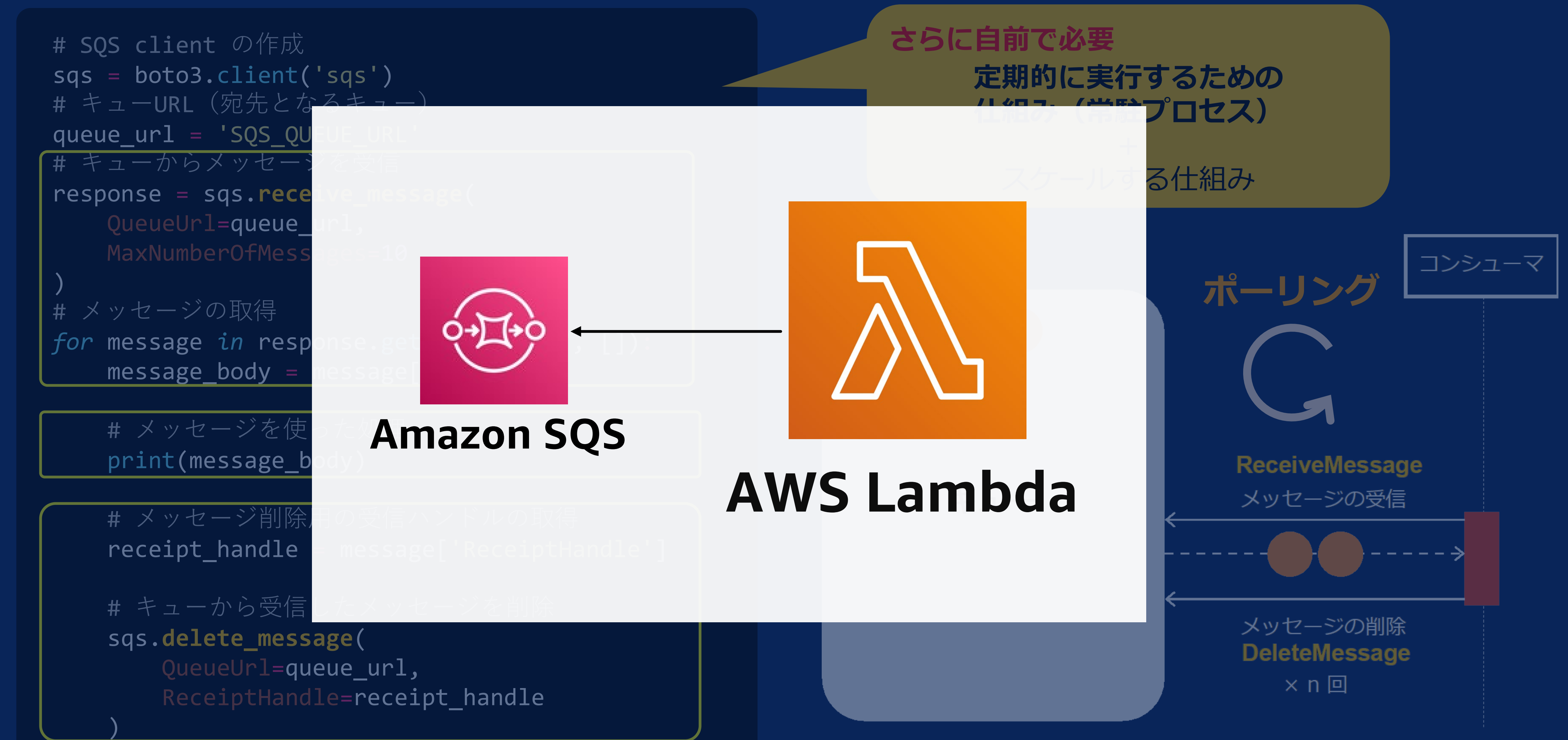
＋

スケールする仕組み



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

メッセージ受信側の実装例

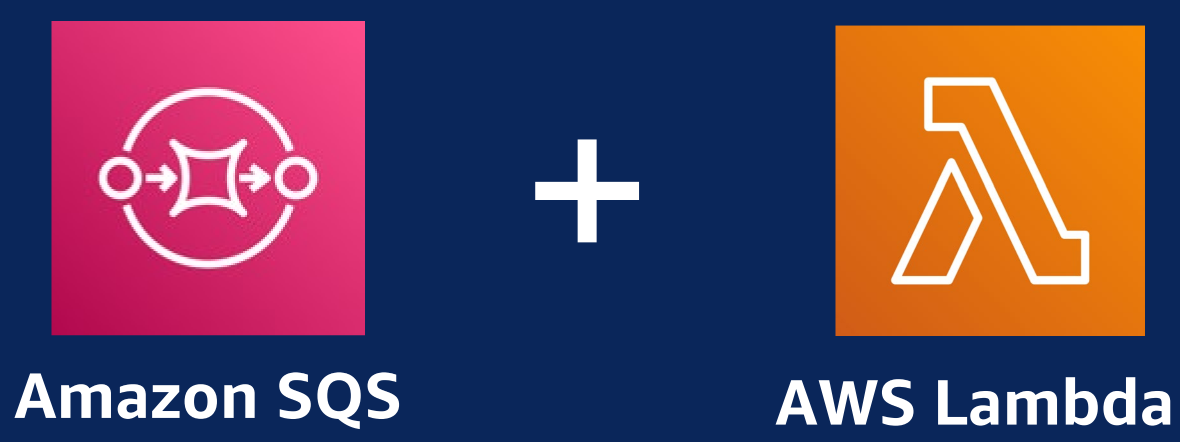


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

SQS + Lambda アーキテクチャ

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

SQS + Lambda の組み合わせ



SQS と Lambda を組み合わせることで

受信処理の実装がシンプルになる

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

SQS のコンシューマとしての Lambda

<u>メッセージの受信処理のために必要な処理</u>

・ポーリング（定期的な受信処理の呼び出し）

・メッセージ量にあわせた実行環境のスケーリング

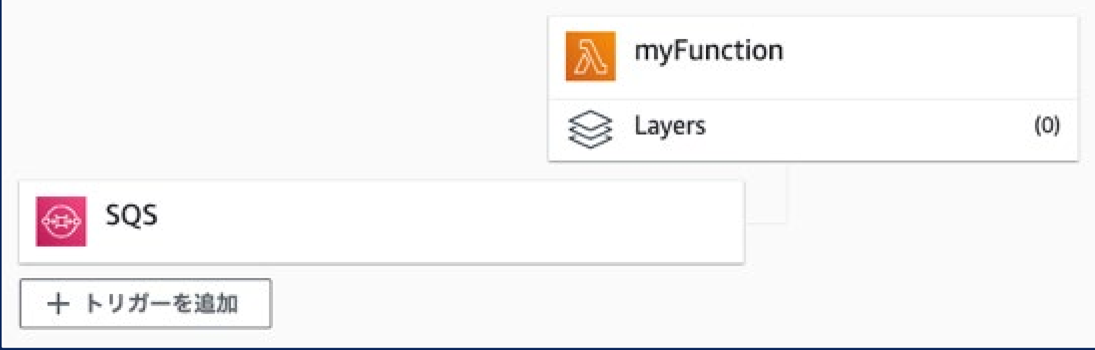
・メッセージの削除処理などの実装

Lambda にオフロード

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

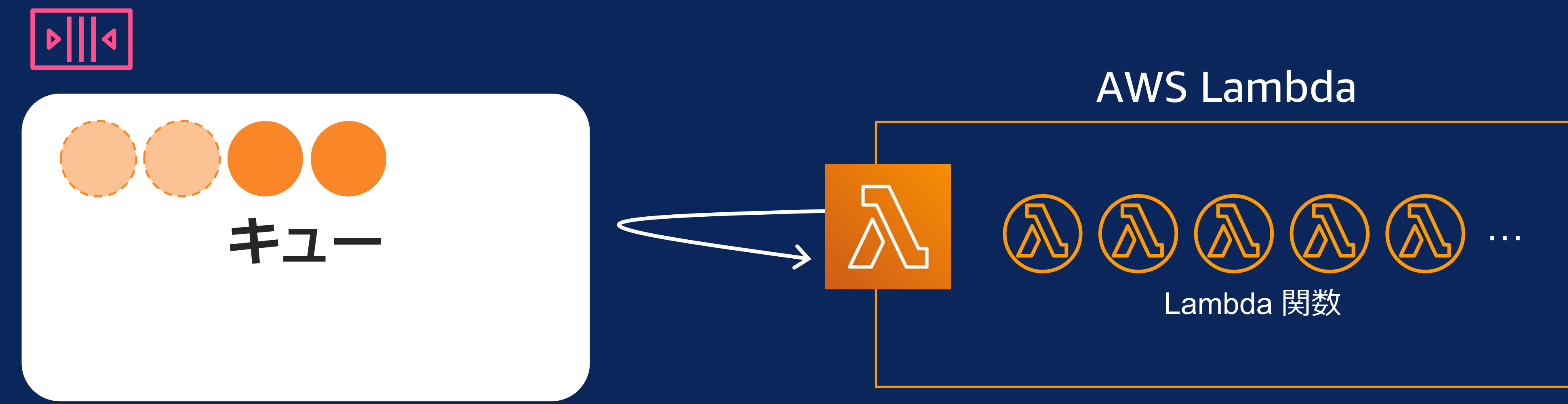
Lambda による実装方法

LambdaのイベントソースとしてSQSのキューを指定



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda による SQS メッセージの受信



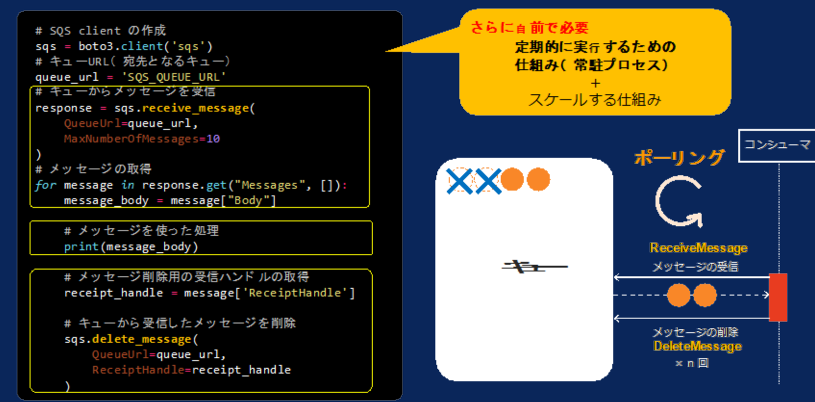
- Lambda が自動でキューをポーリング
- キューにメッセージがある場合、Lambda関数を実行する
- Lambda 関数が正常に実行された場合はメッセージを自動で削除
- １度の実行で 1～10,000 レコードのメッセージをバッチで取得
- キューに残るメッセージの数に応じて自動で実行環境をスケール

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

メッセージ受信の実装例（ Lambda 関数の場合）

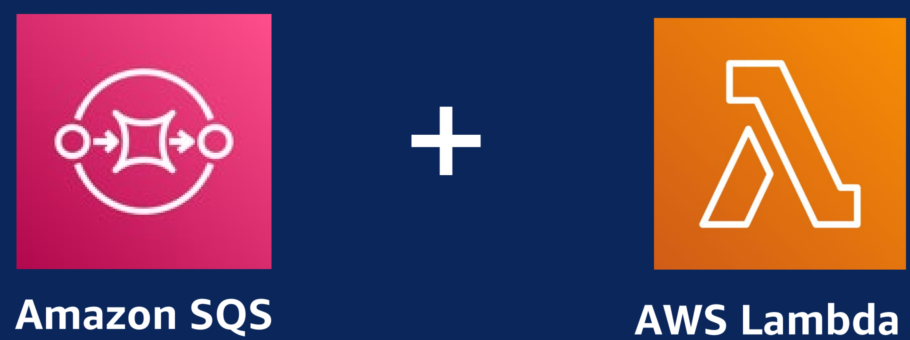
```python
def lambda_handler(event, context):

    for record in event['Records']:
        # メッセージを使った処理
        message_body = record["body"]
        print(str(message_body))
```

メッセージ受信側の実装例



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

SQS + Lambda 黄金パターン によるメリット



SQS と Lambda を組み合わせることによって
以下の受信処理の実装をLambdaにオフロードでき
実装者はメッセージの処理に集中することができるようになる

- ポーリング（定期的な受信処理の呼び出し）
- メッセージ量にあわせた実行環境のスケーリング
- メッセージの削除処理などの実装

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

（補足） Lambda で設定可能な項目

LambdaでSQSをイベントソースとしてマッピングする際
設定可能な項目は以下の通り

① バッチサイズ

② バッチウィンドウ

③ バッチ項目の失敗の報告

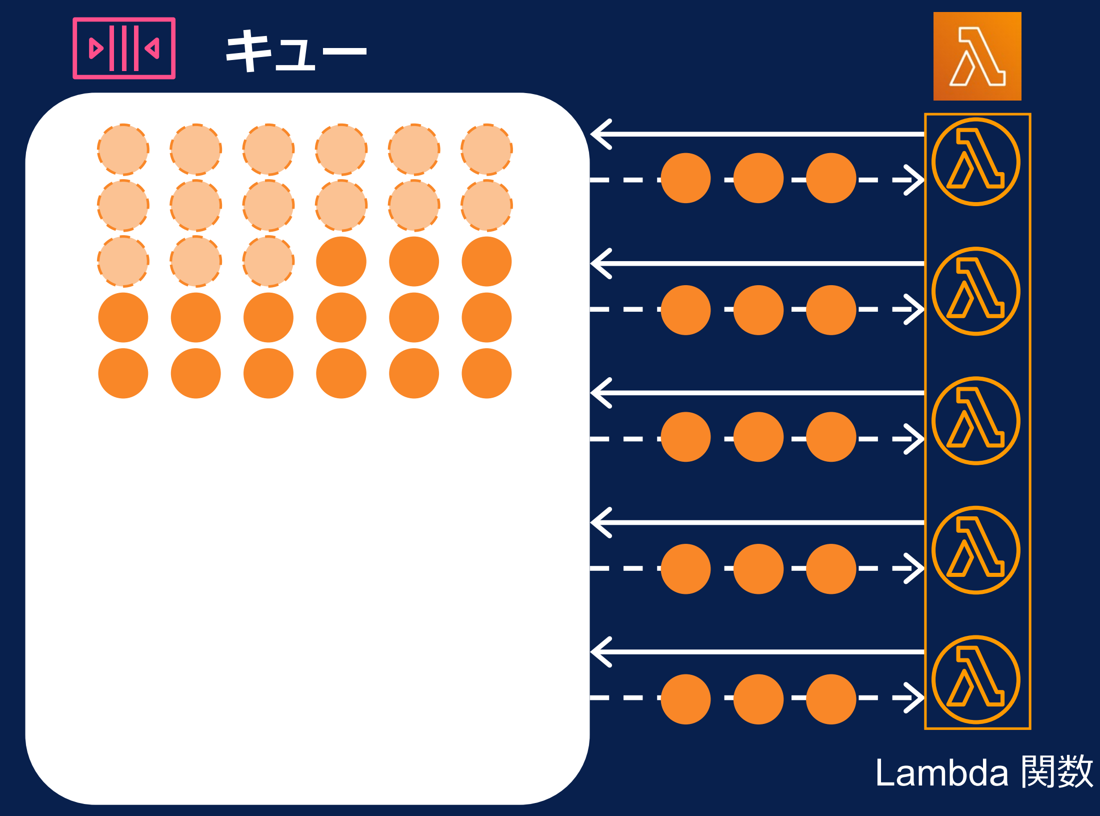
④ フィルタリング条件

NEW （2021/11～）

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

（補足）Lambda関数のスケーリング：標準キューの場合

Lambda は、メッセージを同時に5つのバッチ処理で取得し、5 つのLambda 関数を実行するところからスタート



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

https://aws.amazon.com/blogs/compute/understanding-how-aws-lambda-scales-when-subscribed-to-amazon-sqs-queues/

（補足）Lambda関数のスケーリング：標準キューの場合

キューにさらに多くのメッセージがある場合、
Lambda は 1 分あたり最大 60 個の関数 (最大 1,000 個の関数) を追加してメッセージを処理

キュー

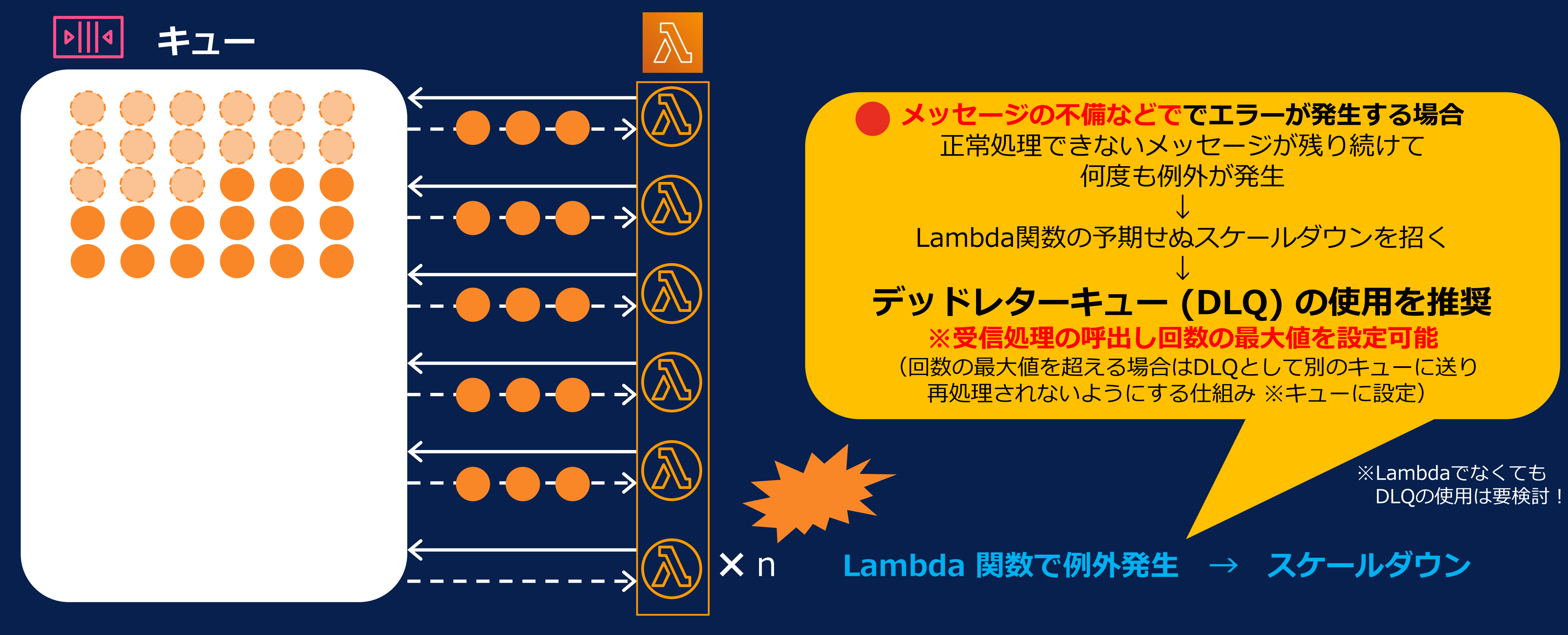
× n　　1 分あたり最大 60 個の関数 を追加

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

https://aws.amazon.com/blogs/compute/understanding-how-aws-lambda-scales-when-subscribed-to-amazon-sqs-queues/

（補足）Lambda関数のスケーリング：標準キューの場合

ただし Lambda 関数がエラーをスローする場合、キューにメッセージが残っている場合でも
誤った呼び出しを最小限に抑えるため、 Lambda は関数をスケールダウンさせる



© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

https://aws.amazon.com/jp/premiumsupport/knowledge-center/lambda-sqs-scaling/

まとめ

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理とは

「処理の結果を待つ必要が **“ない”** サービスの呼び出し」

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

同期処理と非同期処理を使い分ける

快適！

エンドユーザ

EC サイト

処理結果を待つ
必要がある

同期処理で実装

注文処理
サービス

在庫管理
サービス

処理結果を待つ
必要がない

非同期処理で実装

配送処理
サービス

外部システム
（配送事業者）

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

非同期処理にすることのメリット

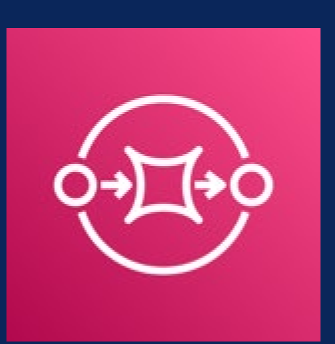
非同期処理をアプリケーション設計に取り入れる

**サービス間を「疎結合」につなぐことによって
以下のメリットが期待できる**

① **応答性の改善**
② **耐障害性・可用性の向上**
③ **スループットの向上**
④ **コスト削減**

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWSのマネージドサービスを利用することで、可用性・スケーラビリティを実現



Amazon SQS

- **フルマネージド型**（サーバ管理不要）の**メッセージキューサービス**
- **ほぼ無制限の TPS**（Transactions per second）
- **利用した分だけの従量課金**（API 実行回数＋データ転送量）
- 分散キューモデルによる高い可用性を提供

料金例：（標準キュー、東京リージョン）
100万リクエストで0.40USD

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

キューの仕組み

プロデューサ

コンシューマ

メッセージの送信

キュー

ポーリング

定期的に実行

メッセージの受信

メッセージの削除

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

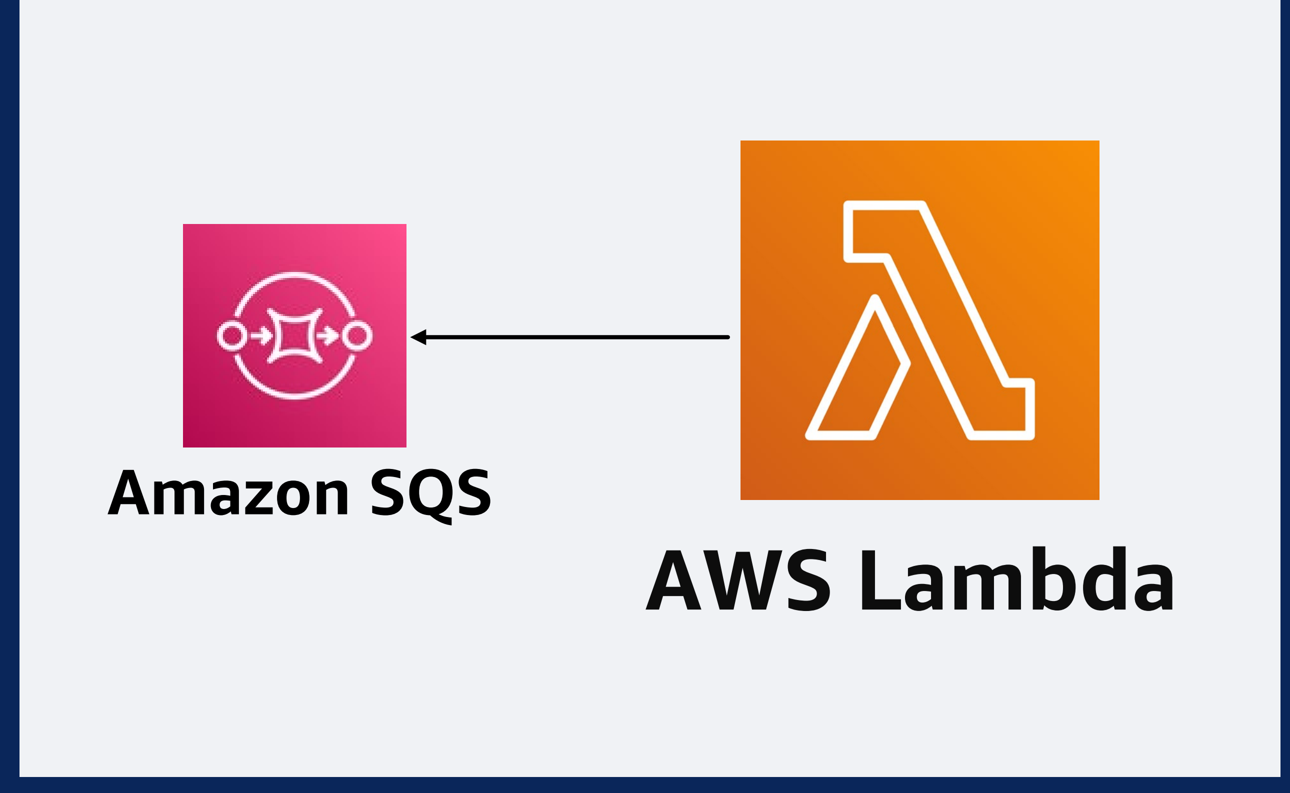
SQS + Lambda 黄金パターン によるメリット

SQS と Lambda を組み合わせることによって
以下の受信処理の実装をLambdaにオフロードでき
実装者はメッセージの処理に集中することができるようになる

・ポーリング（定期的な受信処理の呼び出し）
・メッセージ量にあわせた実行環境のスケーリング
・メッセージの削除処理などの実装

aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

本日のゴール



Amazon SQS + AWS Lambda 黄金パターン
「いいね！」 となって、やってみたくなる！！

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Ichino Shraishi

@piko_san_0000

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.