

Add destination

Destination configuration

Send invocation records to a destination when your function is invoked asynchronously, or if your function processes records from a stream.

Source

The type of invocation that maps to the destination.

- ☒ Asynchronous invocation
- ☐ Stream invocation

Condition

The condition for using the destination.

- ☒ On failure
- ☐ On success

Destination type

An SQS queue, SNS topic, Lambda function, or EventBridge event bus.

SNS topic ▼

Destination

▼



Cancel

Save

Add destination

Destination configuration

Send invocation records to a destination when your function is invoked asynchronously, or if your function processes records from a stream.

Source

The type of invocation that maps to the destination.

- ☒ Asynchronous invocation
☐ Stream invocation

Condition

The condition for using the destination.

- ☒ On failure
☐ On success

Destination type

An SQS queue, SNS topic, Lambda function, or EventBridge event bus.

SNS topic ▼

Destination

▼



SNS, SQS, Lambda,
EventBridge

Cancel

Save

```
{
  "version": "1.0",
  "timestamp": "2019-11-24T21:52:47.333Z",
  "requestContext": {
    "requestId": "8ea123e4-1db7-4aca-ad10-d9ca1234c1fd",
    "functionArn": "arn:aws:lambda:sa-east-1:123456678912:function:event-destinations:$LATEST",
    "condition": "RetriesExhausted",
    "approximateInvokeCount": 3
  },
  "requestPayload": {
    "Success": false
  },
  "responseContext": {
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Handled"
  },
  "responsePayload": {
    "errorMessage": "Failure from event, Success = false, I am failing!",
    "errorType": "Error",
    "stackTrace": [
      "exports.handler (/var/task/index.js:18:18)"
    ]
  }
}
```

```
{
  "version": "1.0",
  "timestamp": "2019-11-24T21:52:47.333Z",
  "requestContext": {
    "requestId": "8ea123e4-1db7-4aca-ad10-d9ca1234c1fd",
    "functionArn": "arn:aws:lambda:sa-east-1:123456678912:function:event-destinations:$LATEST",
    "condition": "RetriesExhausted",
    "approximateInvokeCount": 3
  },
  "requestPayload": {
    "Success": false
  },
  "responseContext": {
    "statusCode": 200,
    "executedVersion": "$LATEST",
    "functionError": "Handled"
  },
  "responsePayload": {
    "errorMessage": "Failure from event, Success = false, I am failing!",
    "errorType": "Error",
    "stackTrace": [
      "exports.handler (/var/task/index.js:18:18)"
    ]
  }
}
```

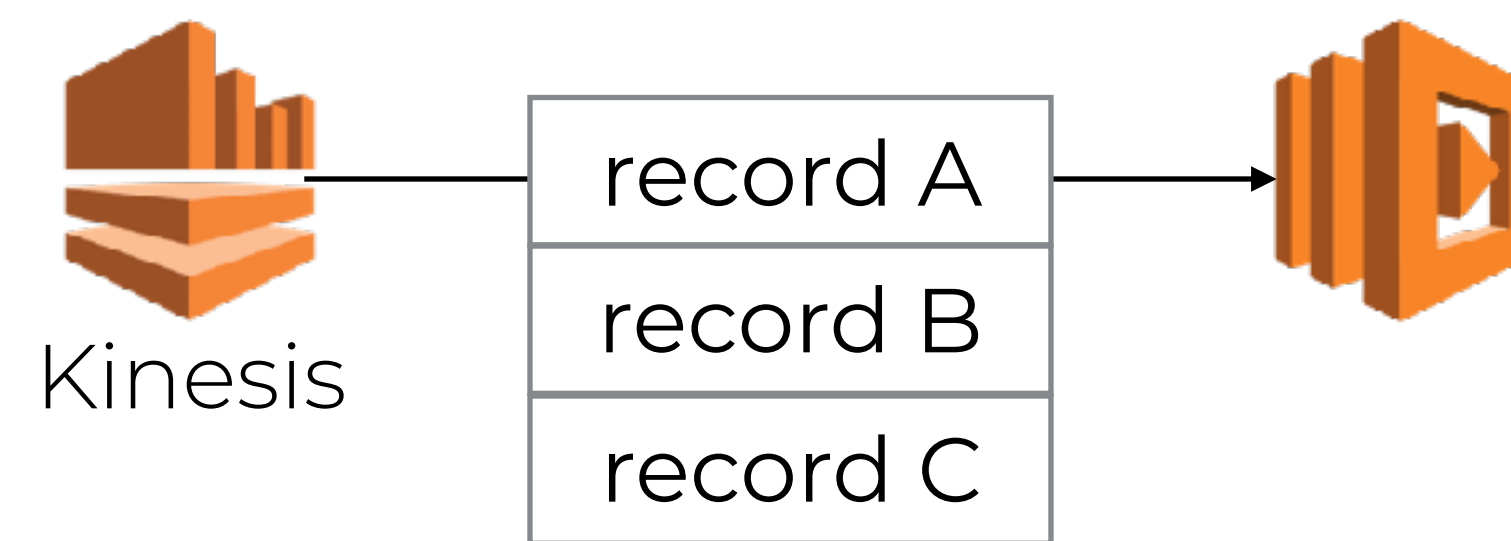


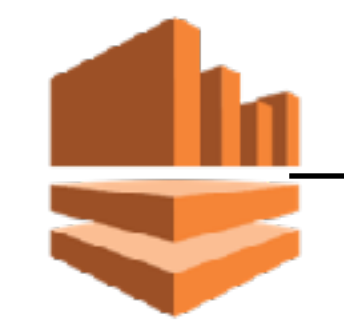
events should be processed in realtime
(i.e. within a few seconds)

failed events should be retried, but the retries
should not violate the realtime constraint

unprocessed events should be retrievable
(i.e. available for human intervention, or root cause analysis)

- **Stream-based event sources** – For stream-based event sources (Amazon Kinesis Data Streams and DynamoDB streams), AWS Lambda polls your stream and invokes your Lambda function. Therefore, if a Lambda function fails, AWS Lambda attempts to process the erring batch of records until the time the data expires, which can be up to seven days for Amazon Kinesis Data Streams. The exception is treated as blocking, and AWS Lambda will not read any new records from the stream until the failed batch of records either expires or processed successfully. This ensures that AWS Lambda processes the stream events in order.



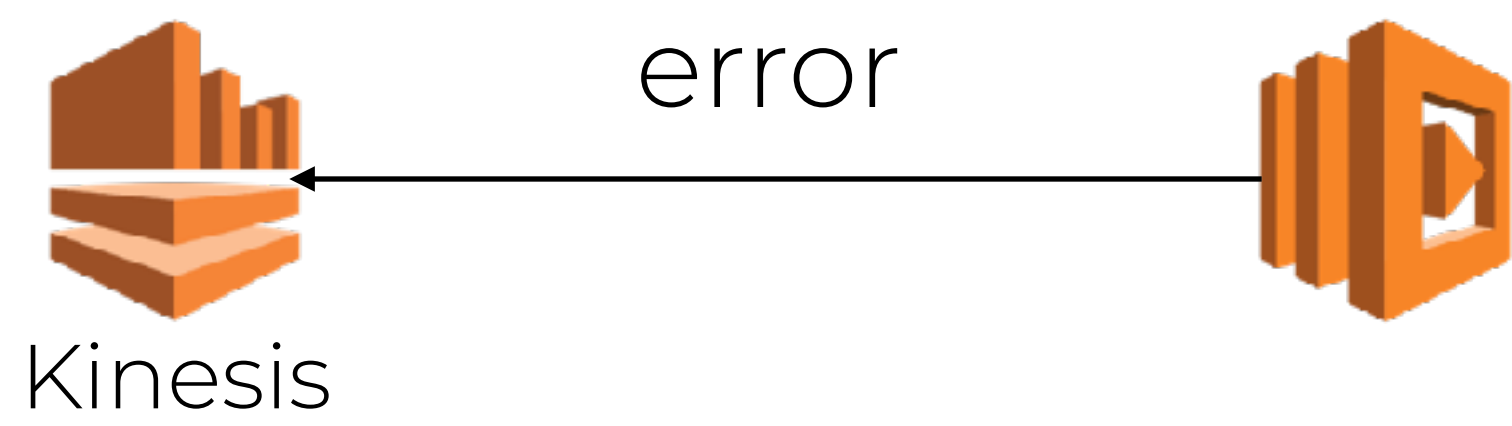


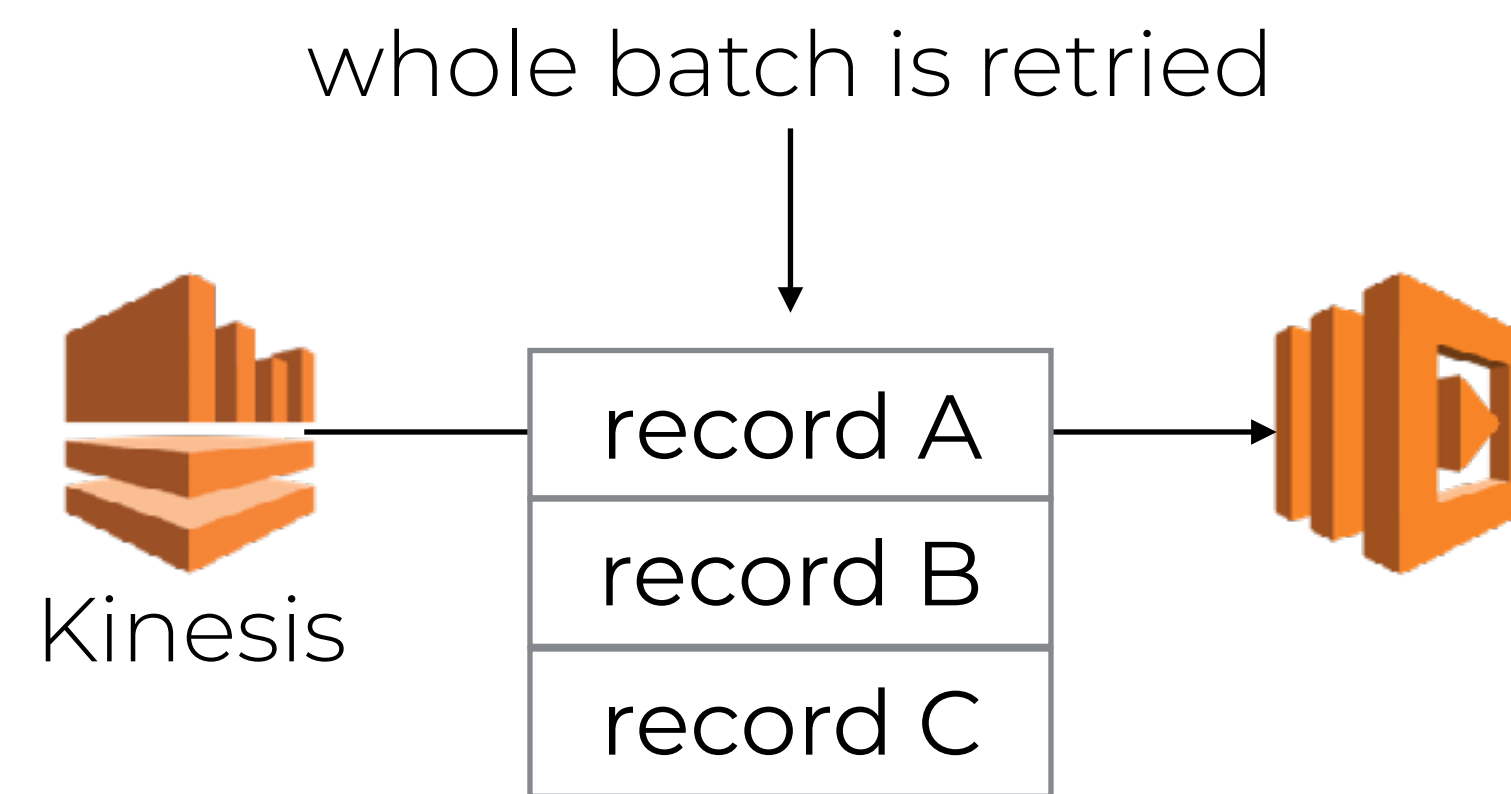
Kinesis



record A
record B
record C

← erred





you need to consider **Partial Failures** and **Idempotency** when processing Kinesis and DynamoDB streams with Lambda

failed events should be retried, but the retries
should not violate the realtime constraint

failed events should be retried, but the retries
should not violate the realtime constraint

you need to consider **Partial Failures** and **Idempotency** when processing Kinesis and DynamoDB streams with Lambda

On-failure destination

Lambda discards records that are expired or fail all retry attempts. You can send discarded records from a stream to an Amazon SQS queue or an Amazon SNS topic.

Queue or topic ARN

Retry attempts

The maximum number of times to retry when the function returns an error.

10000

Maximum age of record

The maximum age of a record that Lambda sends to a function for processing. The age can be up to 604,800 seconds (7 days).

604800

Split batch on error

If the function returns an error, split the batch into two and retry.

☐

Concurrent batches per shard

Process multiple batches from the same shard concurrently.

1

On-failure destination

Lambda discards records that are expired or fail all retry attempts. You can send discarded records from a stream to an Amazon SQS queue or an Amazon SNS topic.

Queue or topic ARN

Retry attempts

The maximum number of times to retry when the function returns an error.

10000

Maximum age of record

The maximum age of a record that Lambda sends to a function for processing. The age can be up to 604,800 seconds (7 days).

604800

Split batch on error

If the function returns an error, split the batch into two and retry.

☐

Concurrent batches per shard

Process multiple batches from the same shard concurrently.

1

On-failure destination

Lambda discards records that are expired or fail all retry attempts. You can send discarded records from a stream to an Amazon SQS queue or an Amazon SNS topic.

Queue or topic ARN

Retry attempts

The maximum number of times to retry when the function returns an error.

10000

Maximum age of record

The maximum age of a record that Lambda sends to a function for processing. The age can be up to 604,800 seconds (7 days).

604800

Split batch on error

If the function returns an error, split the batch into two and retry.

☐

Concurrent batches per shard

Process multiple batches from the same shard concurrently.

1

On-failure destination

Lambda discards records that are expired or fail all retry attempts. You can send discarded records from a stream to an Amazon SQS queue or an Amazon SNS topic.

Queue or topic ARN

Retry attempts

The maximum number of times to retry when the function returns an error.

10000

Maximum age of record

The maximum age of a record that Lambda sends to a function for processing. The age can be up to 604,800 seconds (7 days).

604800

Split batch on error

If the function returns an error, split the batch into two and retry.

☐

Concurrent batches per shard

Process multiple batches from the same shard concurrently.

1

6, 3, 1, 1, 1, 1, ...

On-failure destination

Lambda discards records that are expired or fail all retry attempts. You can send discarded records from a stream to an Amazon SQS queue or an Amazon SNS topic.

Queue or topic ARN

only count the "same" batch

Retry attempts

The maximum number of times to retry when the function returns an error.

10000

Maximum age of record

The maximum age of a record that Lambda sends to a function for processing. The age can be up to 604,800 seconds (7 days).

604800

6, 3, 1, 1, 1, 1, ...

Split batch on error

If the function returns an error, split the batch into two and retry.

☐

Concurrent batches per shard

Process multiple batches from the same shard concurrently.

1


```
{
  "Body": "{
    \"requestContext\":{
      \"requestId\": \"0cdf90e8-6290-4834-a46c-cffdd6277c4f\",
      \"functionArn\": \"arn:aws:lambda:us-east-1:374852340823:function:aws-sdk-example\",
      \"condition\": \"RetryAttemptsExhausted\",
      \"approximateInvokeCount\": 11
    },
    \"responseContext\":{
      \"statusCode\": 200,
      \"executedVersion\":
        \"$LATEST\",
      \"functionError\":
        \"Unhandled\"
    },
    \"version\": \"1.0\",
    \"timestamp\": \"2020-02-08T19:53:20.998Z\",
    \"KinesisBatchInfo\":{
      \"shardId\": \"shardId-000000000000\",
      \"startSequenceNumber\": \"49604045060801379268037077355262289118696897411062693890\",
      \"endSequenceNumber\": \"49604045060801379268037077355262289118696897411062693890\",
      \"approximateArrivalOfFirstRecord\": \"2020-02-08T19:47:05.582Z\",
      \"approximateArrivalOfLastRecord\": \"2020-02-08T19:47:05.582Z\",
      \"batchSize\": 1,
      \"streamArn\": \"arn:aws:kinesis:us-east-1:374852340823:stream/yc-test\"
    }
  }"
}
```



```
{
  "Body": "{
    \"requestContext\":{
      \"requestId\": \"0cdf90e8-6290-4834-a46c-cffdd6277c4f\",
      \"functionArn\": \"arn:aws:lambda:us-east-1:374852340823:function:aws-sdk-example\",
      \"condition\": \"RetryAttemptsExhausted\",
      \"approximateInvokeCount\": 11
    },
    \"responseContext\":{
      \"statusCode\": 200,
      \"executedVersion\":
        \"$LATEST\",
      \"functionError\":
        \"Unhandled\"
    },
    \"version\": \"1.0\",
    \"timestamp\": \"2020-02-08T19:53:20.998Z\",
    \"KinesisBatchInfo\":{
      \"shardId\": \"shardId-000000000000\",
      \"startSequenceNumber\": \"49604045060801379268037077355262289118696897411062693890\",
      \"endSequenceNumber\": \"49604045060801379268037077355262289118696897411062693890\",
      \"approximateArrivalOfFirstRecord\": \"2020-02-08T19:47:05.582Z\",
      \"approximateArrivalOfLastRecord\": \"2020-02-08T19:47:05.582Z\",
      \"batchSize\": 1,
      \"streamArn\": \"arn:aws:kinesis:us-east-1:374852340823:stream/yc-test\"
    }
  }"
}
```

On-failure destination

Lambda discards records that are expired or fail all retry attempts. You can send discarded records from a stream to an Amazon SQS queue or an Amazon SNS topic.

Queue or topic ARN

Retry attempts

The maximum number of times to retry when the function returns an error.

10000

Maximum age of record

The maximum age of a record that Lambda sends to a function for processing. The age can be up to 604,800 seconds (7 days).

604800

Split batch on error

If the function returns an error, split the batch into two and retry.

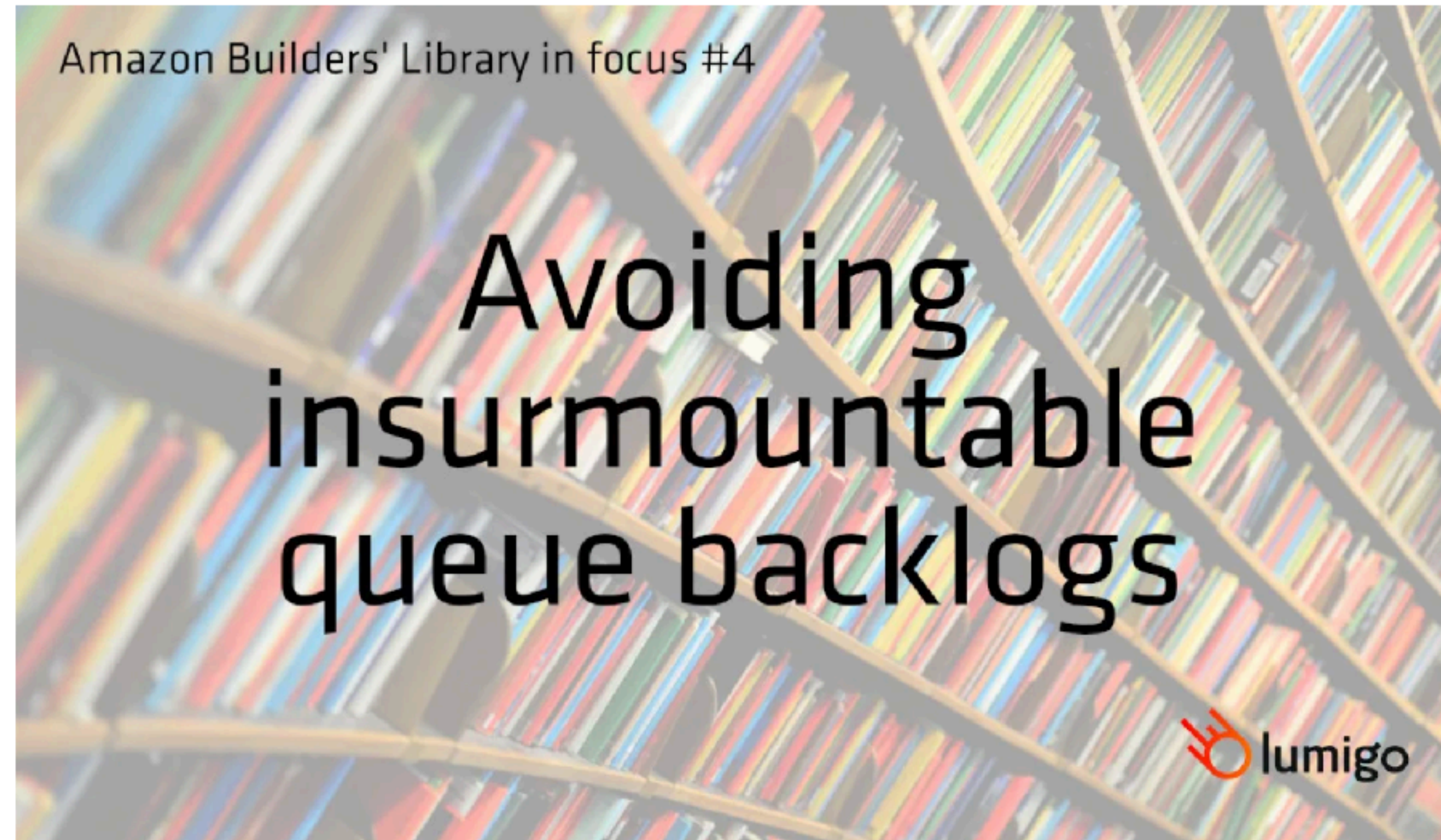
☐

Concurrent batches per shard

Process multiple batches from the same shard concurrently.

1

AMAZON BUILDERS' LIBRARY IN FOCUS #4: AVOIDING INSURMOUNTABLE QUEUE BACKLOGS



In the latest article in our series focusing on the **Amazon Builders' Library**, Yan Cui highlights the key insights from **Avoiding insurmountable queue backlogs** by AWS Principal Engineer David Yanacek.

<https://lumigo.io/blog/amazon-builders-library-in-focus-4-avoiding-insurmountable-queue-backlogs>



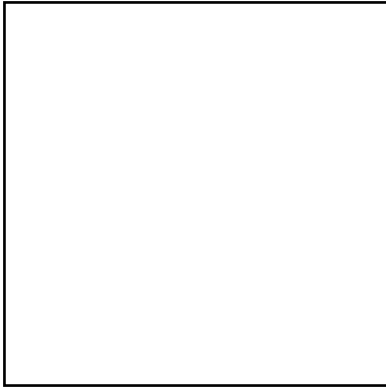
SQS



Lambda



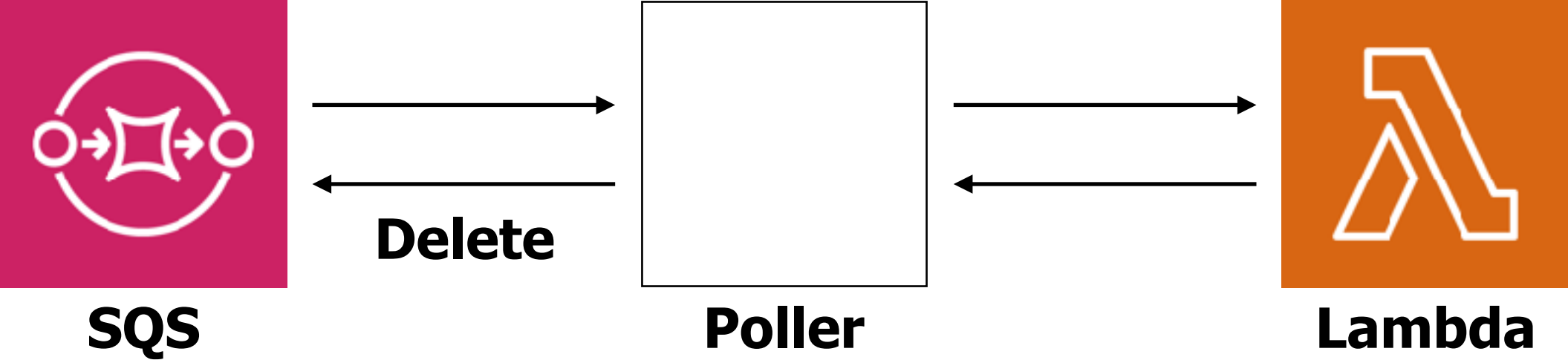
SQS

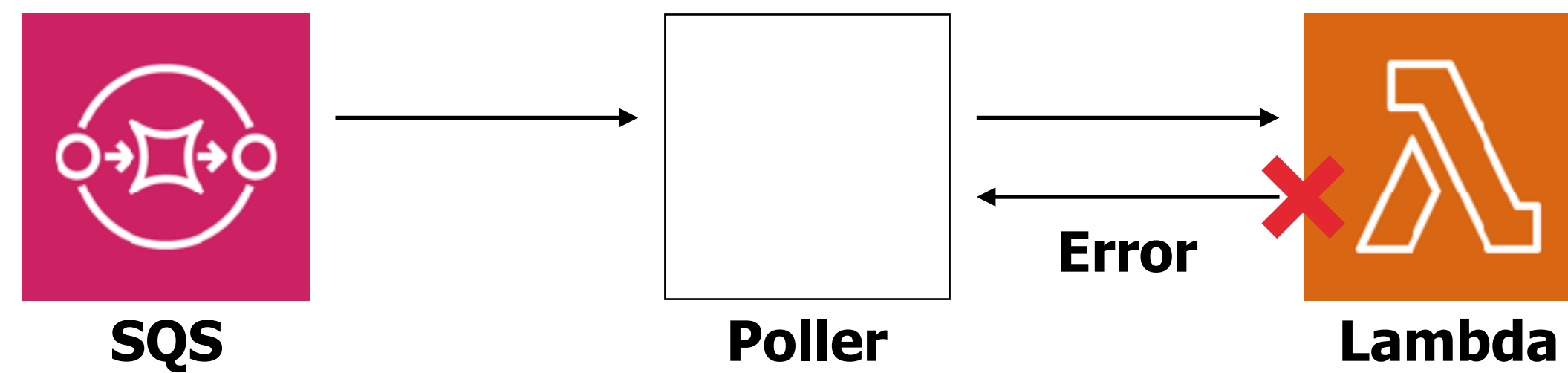


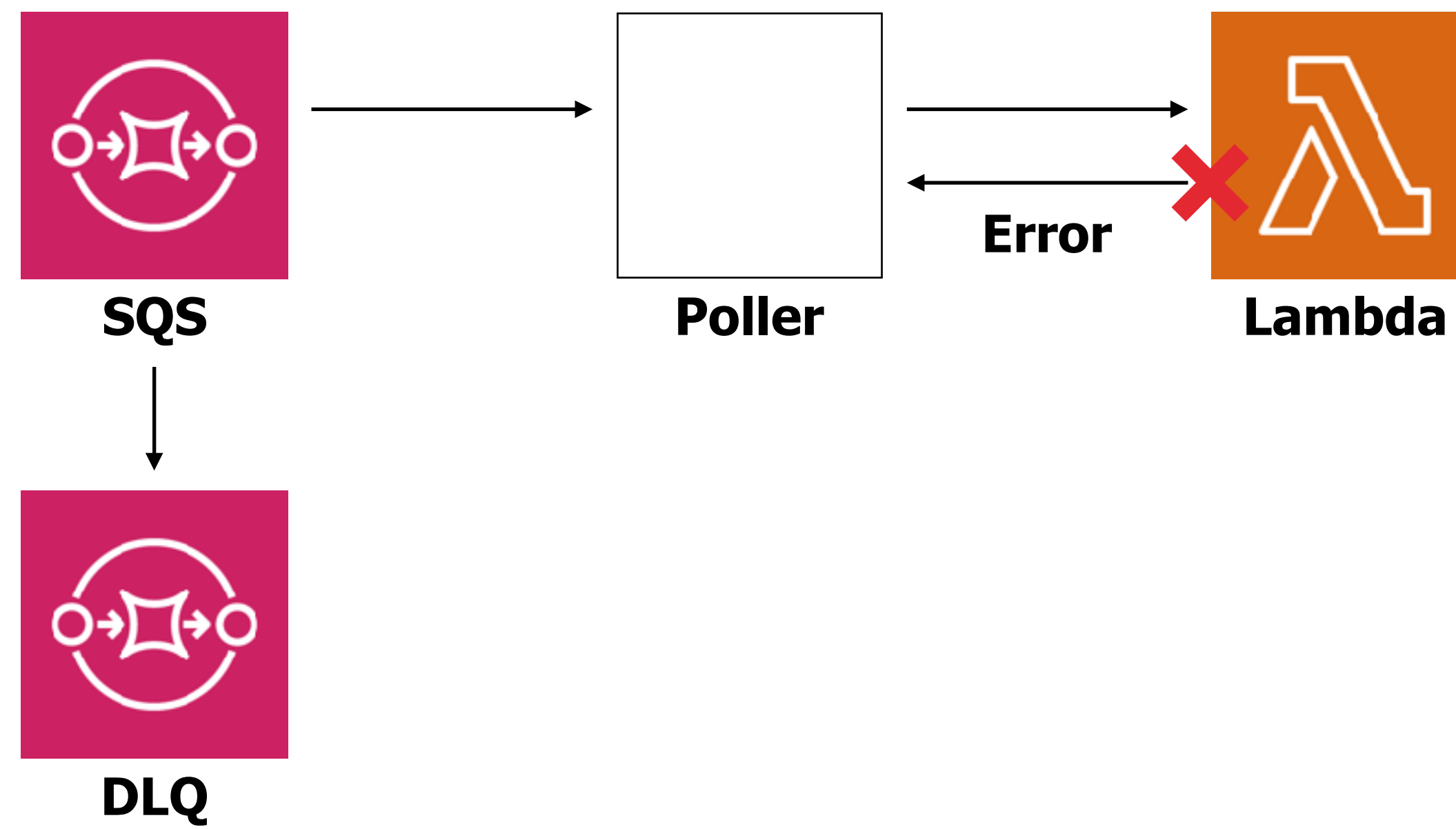
Poller

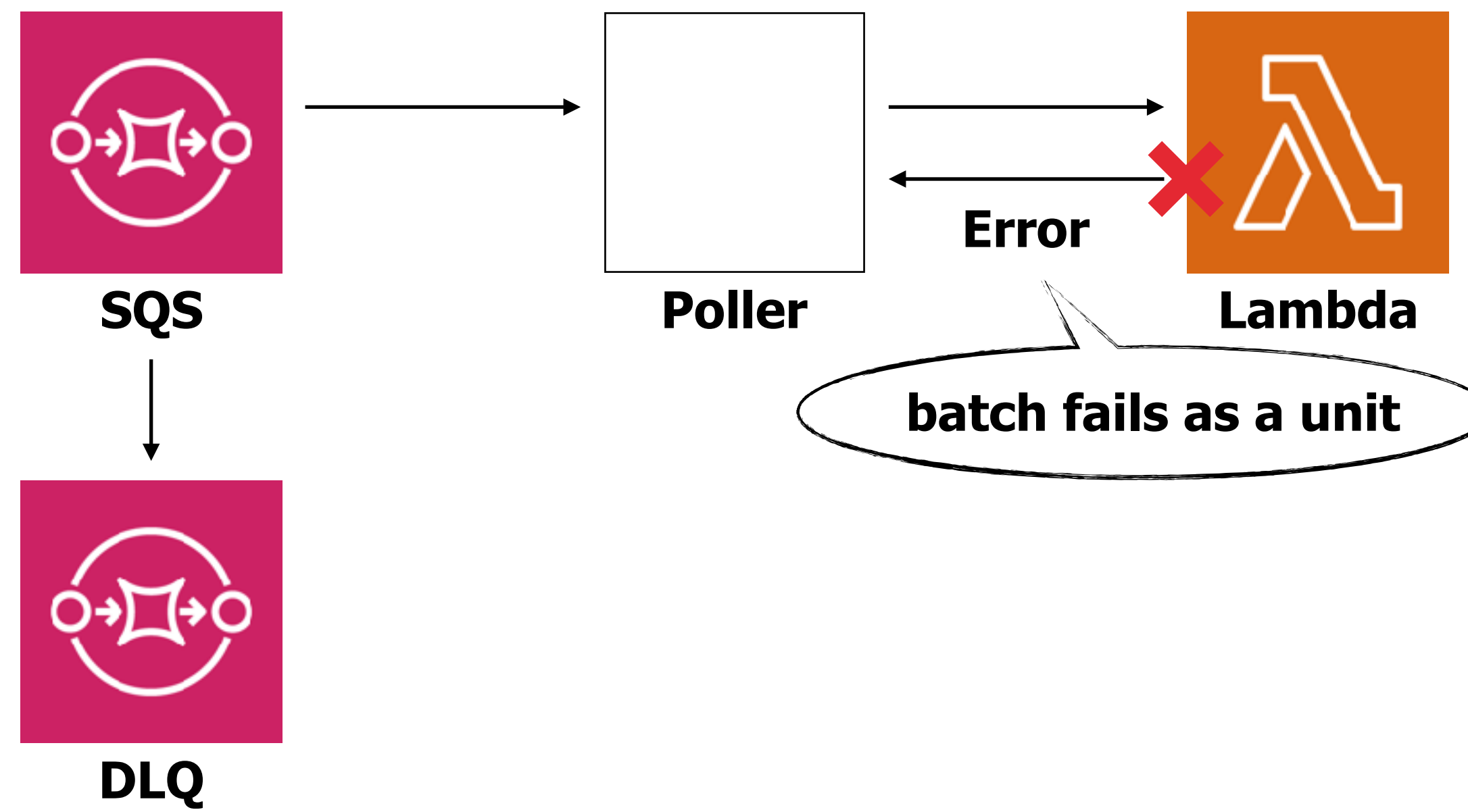


Lambda









SQS AND LAMBDA: THE MISSING GUIDE ON FAILURE MODES



Since **Lambda added SQS** as an event source, there has been a misconception that SQS is now a “push-based” service. This seems true from the perspective of your function because you no longer have to poll SQS yourself. However, SQS itself hasn’t changed – it is still very much a “poll-based” service. The difference is that the Lambda service is managing the pollers (and paying for them!) on your behalf.

<https://lumigo.io/blog/sqs-and-lambda-the-missing-guide-on-failure-modes>

Use a batchSize of 1. This eliminates the problem of partial failures altogether but limits the throughput of how quickly you are able to process messages. In low traffic scenarios, it can be a simple and yet effective solution.

Ensure idempotency. This allows messages to be safely processed more than once. But in processing a message, if you have to update multiple systems then it's difficult to achieve idempotency. You might find yourself reaching out for solutions such as **saga patterns**, which adds much complexity to the system.

My personal preference is to:

- Catch errors when processing each message.
- At the end of the batch, check if there are any partial failures.
 - If no then do nothing. The poller would delete the messages from the queue for me.
 - If there are partial failures, then:
 1. call the **DeleteMessage** API on the successful messages to manually delete them from the queue.
 2. throw an aggregate error with information about the message IDs that failed and their corresponding errors. This leaves the erroneous messages on the queue, so they can be processed again after the visibility timeout.

<https://lumigo.io/blog/sqs-and-lambda-the-missing-guide-on-failure-modes>

Use a **batchSize of 1**. This eliminates the problem of partial failures altogether but limits the throughput of how quickly you are able to process messages. In low traffic scenarios, it can be a simple and yet effective solution.

Ensure idempotency. This allows messages to be safely processed more than once. But in processing a message, if you have to update multiple systems then it's difficult to achieve idempotency. You might find yourself reaching out for solutions such as **saga patterns**, which adds much complexity to the system.

My personal preference is to:

- Catch errors when processing each message.
- At the end of the batch, check if there are any partial failures.
 - If no then do nothing. The poller would delete the messages from the queue for me.
 - If there are partial failures, then:
 1. call the **DeleteMessage** API on the successful messages to manually delete them from the queue.
 2. throw an aggregate error with information about the message IDs that failed and their corresponding errors. This leaves the erroneous messages on the queue, so they can be processed again after the visibility timeout.

<https://lumigo.io/blog/sqs-and-lambda-the-missing-guide-on-failure-modes>

Use a batchSize of 1. This eliminates the problem of partial failures altogether but limits the throughput of how quickly you are able to process messages. In low traffic scenarios, it can be a simple and yet effective solution.

Ensure idempotency. This allows messages to be safely processed more than once. But in processing a message, if you have to update multiple systems then it's difficult to achieve idempotency. You might find yourself reaching out for solutions such as **saga patterns**, which adds much complexity to the system.

My personal preference is to:

- Catch errors when processing each message.
- At the end of the batch, check if there are any partial failures.
 - If no then do nothing. The poller would delete the messages from the queue for me.
 - If there are partial failures, then:
 1. call the **DeleteMessage** API on the successful messages to manually delete them from the queue.
 2. throw an aggregate error with information about the message IDs that failed and their corresponding errors. This leaves the erroneous messages on the queue, so they can be processed again after the visibility timeout.

<https://lumigo.io/blog/sqs-and-lambda-the-missing-guide-on-failure-modes>