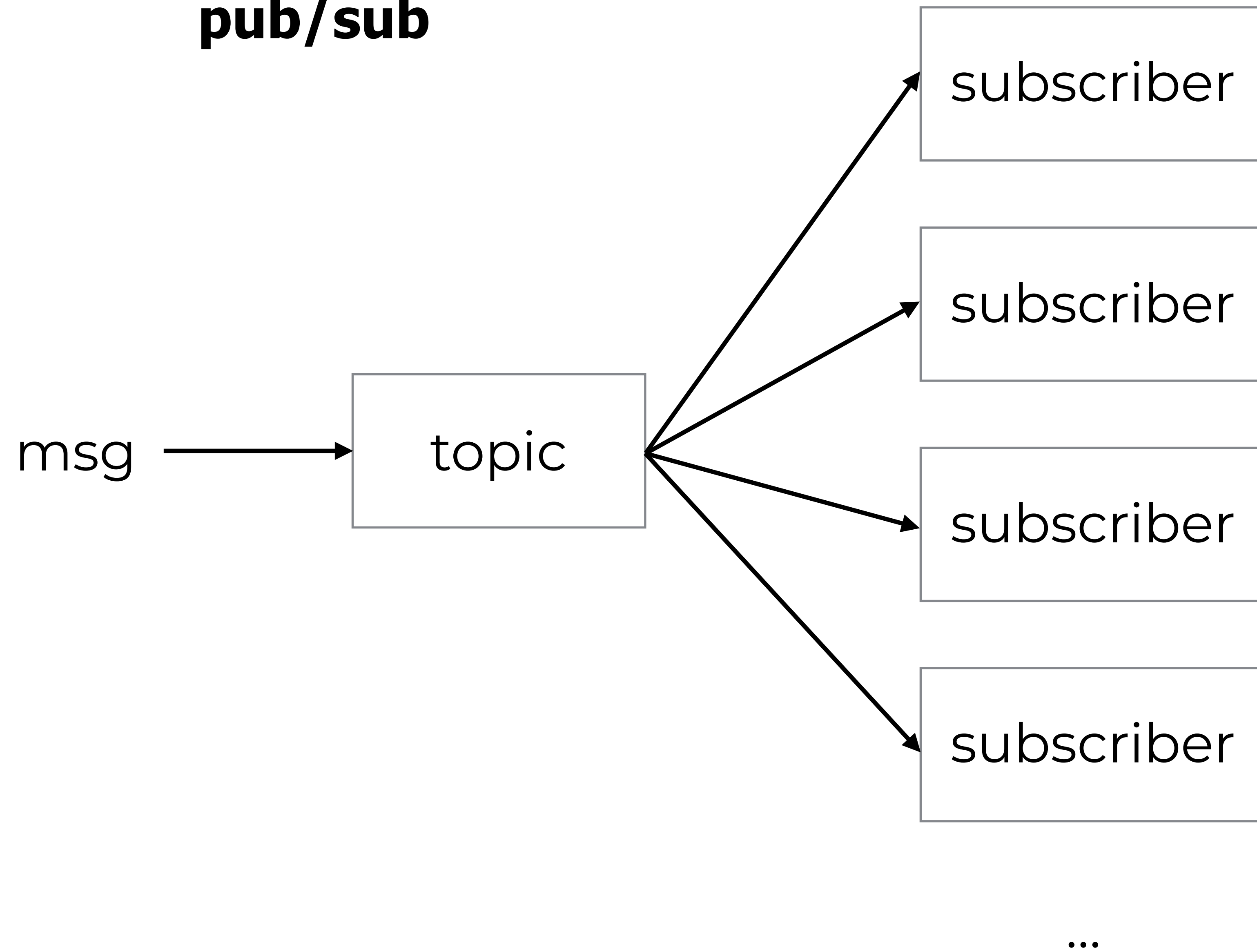


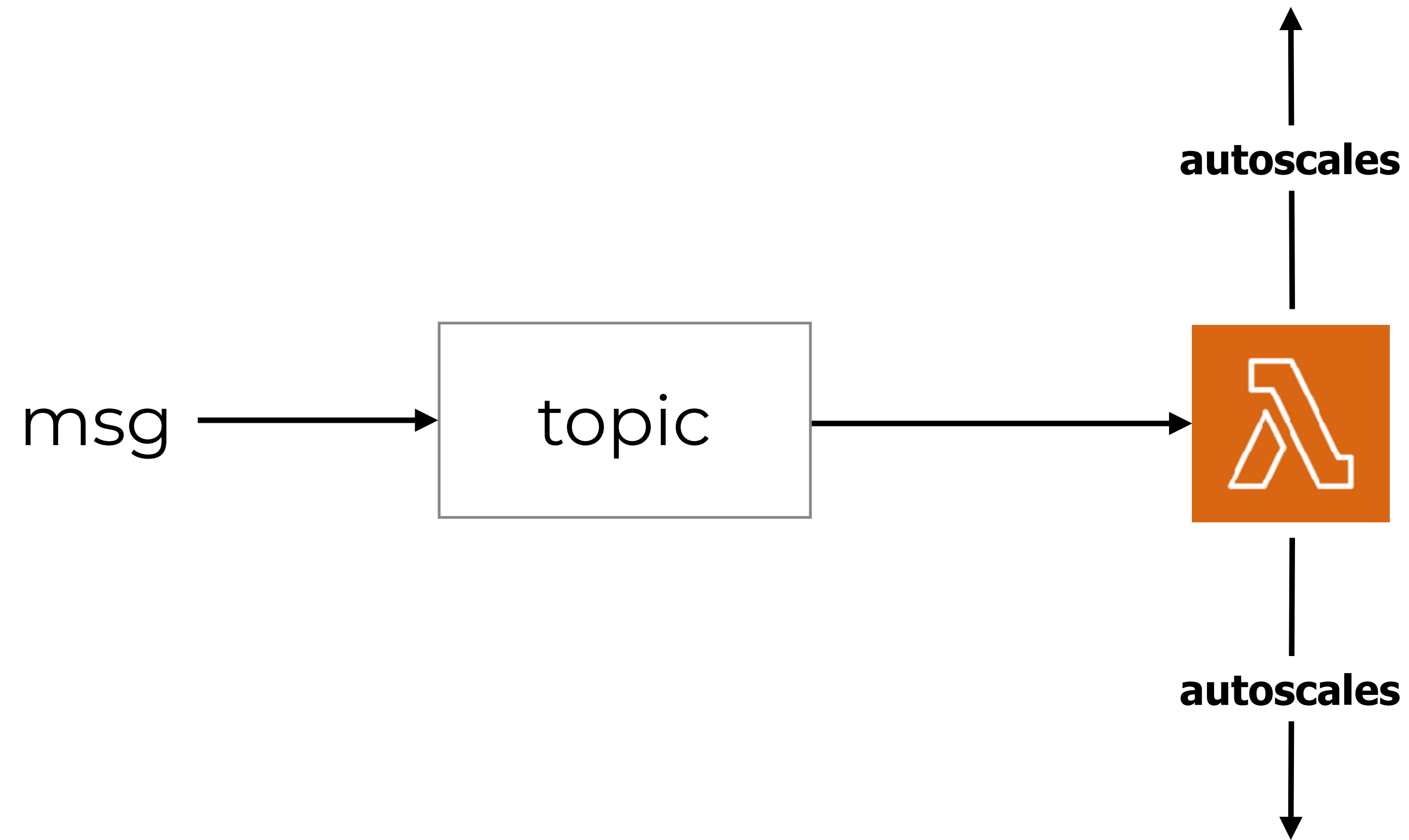
pub/sub

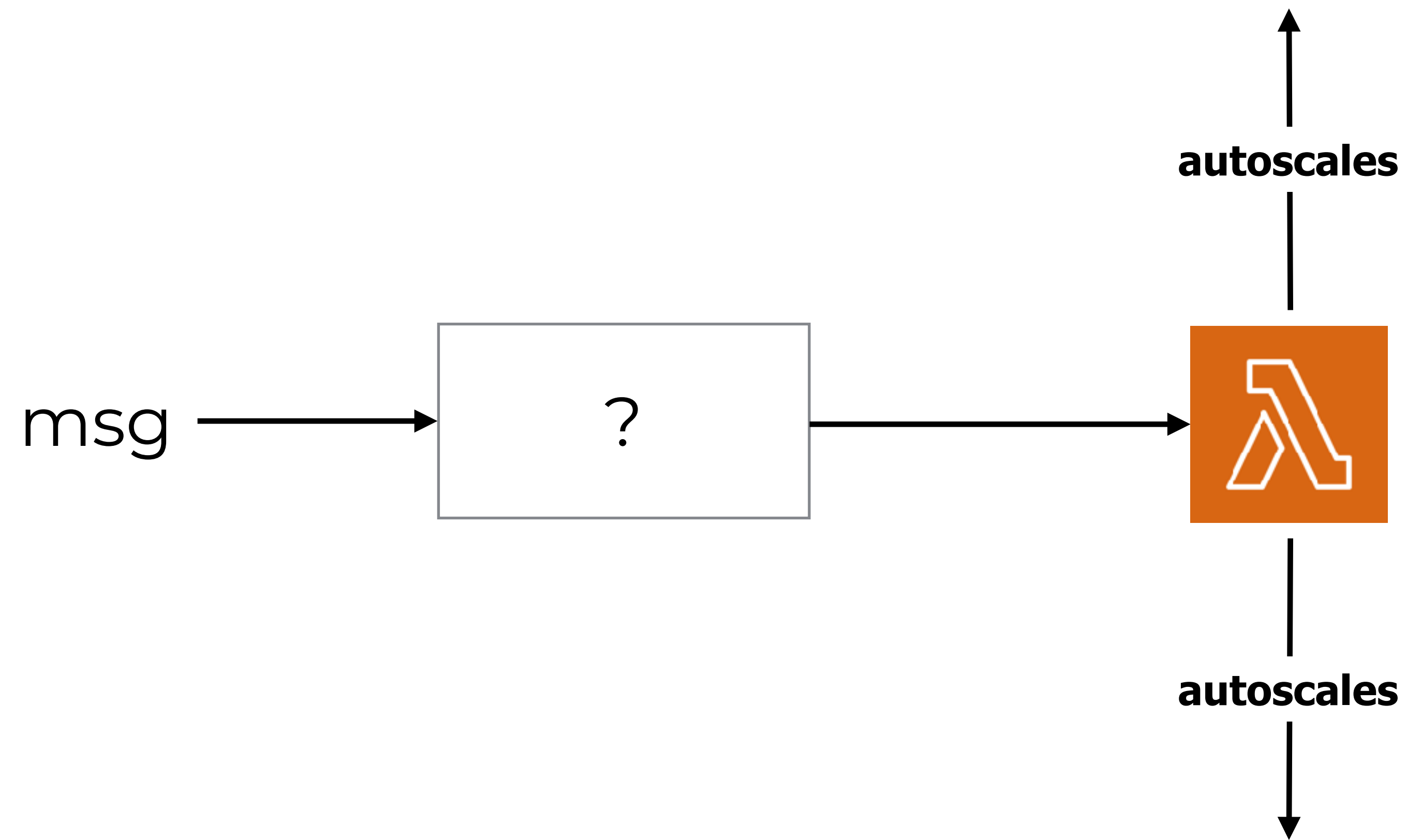


good for **decoupling** publishers and subscribers

always factor scale into the equation

services that pay by uptime are orders-of-magnitude
cheaper when running at scale







SNS



SQS



EventBridge



Kinesis

1 msg/s for a month, 1KB per msg



SNS

1 x 60s x 60m x 24hr x 30days
@ \$0.50 per mil

\$1.296



SQS

1 x 60s x 60m x 24hr x 30days
@ \$0.40 per mil

\$1.037



EventBridge

1 x 60s x 60m x 24hr x 30days
@ \$1.00 per mil

\$2.592



Kinesis

1 x 60s x 60m x 24hr x 30days
@ \$0.014 per mil
+
24hrs x 30days
@ \$0.015 per shard per hr

\$10.836

1,000 msg/s for a month, 1KB per msg



SNS

1000 x 60s x 60m x 24hr x 30days
@ \$0.50 per mil

\$1296.00



SQS

1000 x 60s x 60m x 24hr x 30days
@ \$0.40 per mil

\$1036.80



EventBridge

1000 x 60s x 60m x 24hr x 30days
@ \$1.00 per mil

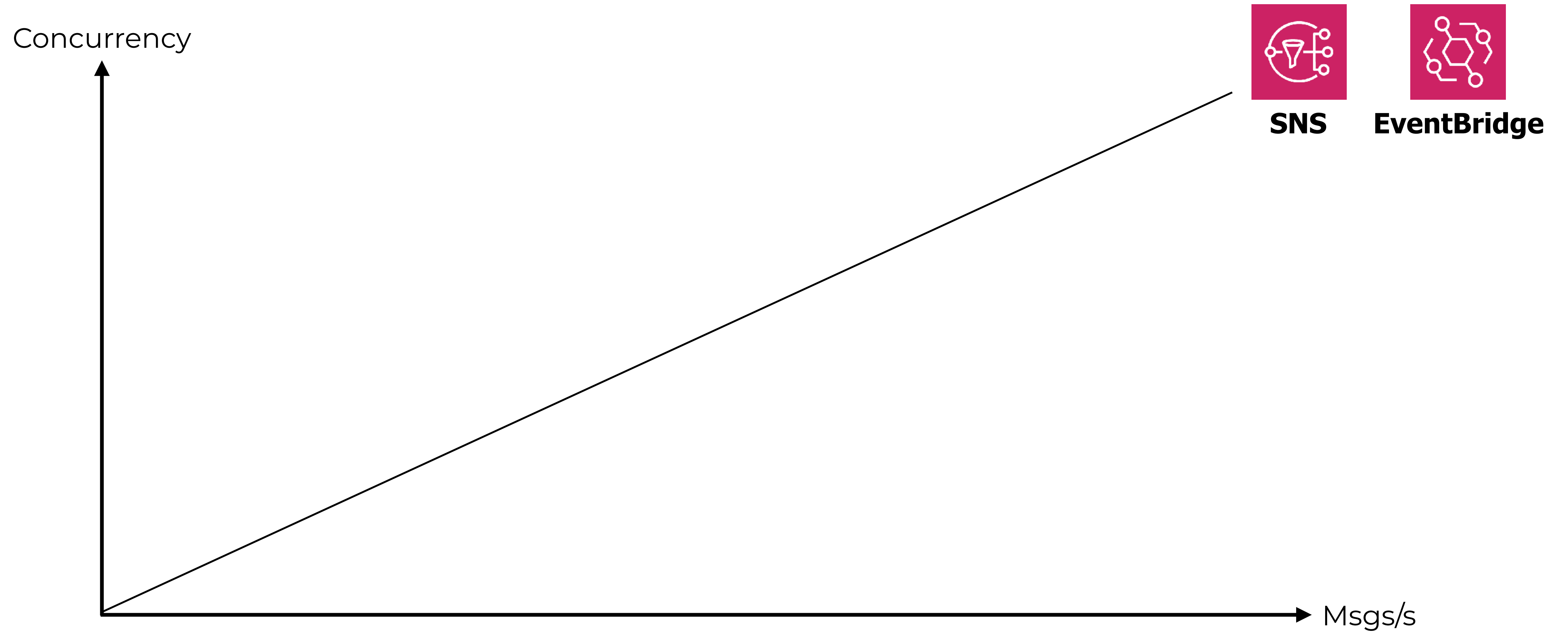
\$2592.00



Kinesis

1000 x 60s x 60m x 24hr x 30days
@ \$0.014 per mil
+
24hrs x 30days
@ \$0.015 per shard per hr

\$47.088



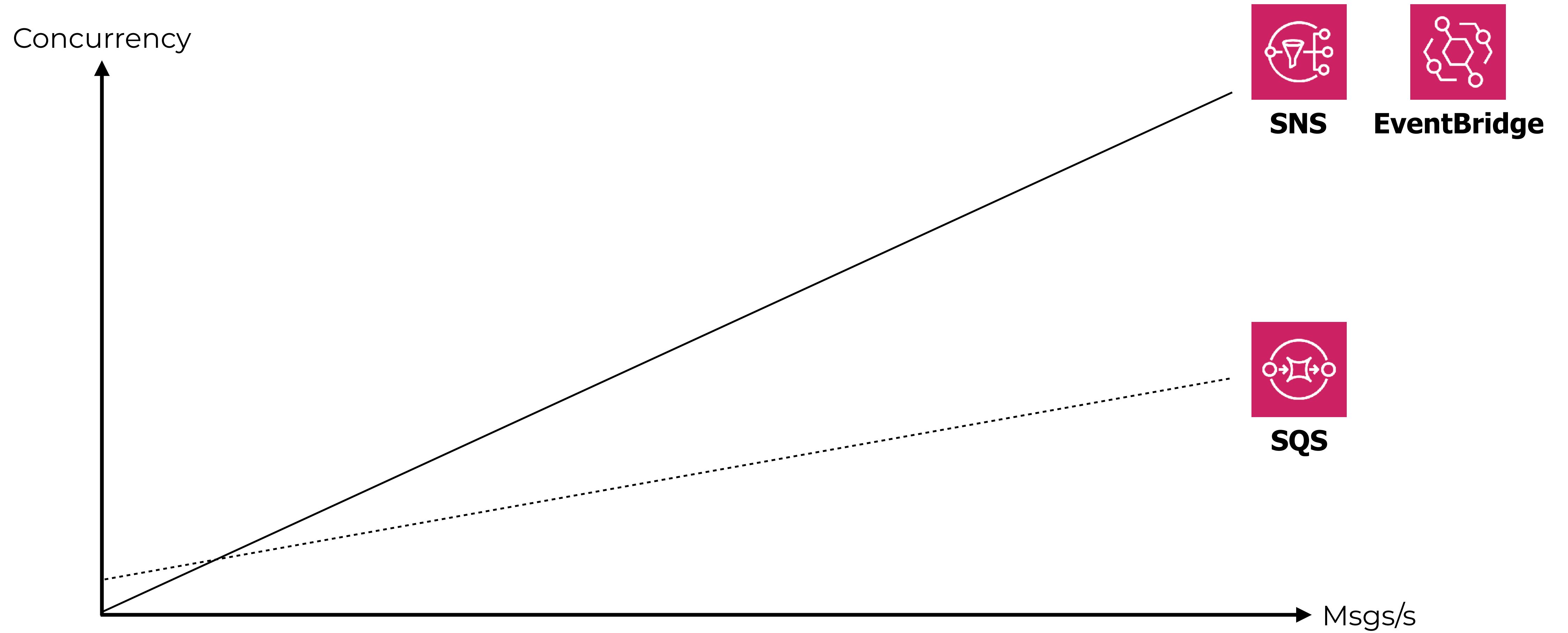
Scaling and Processing

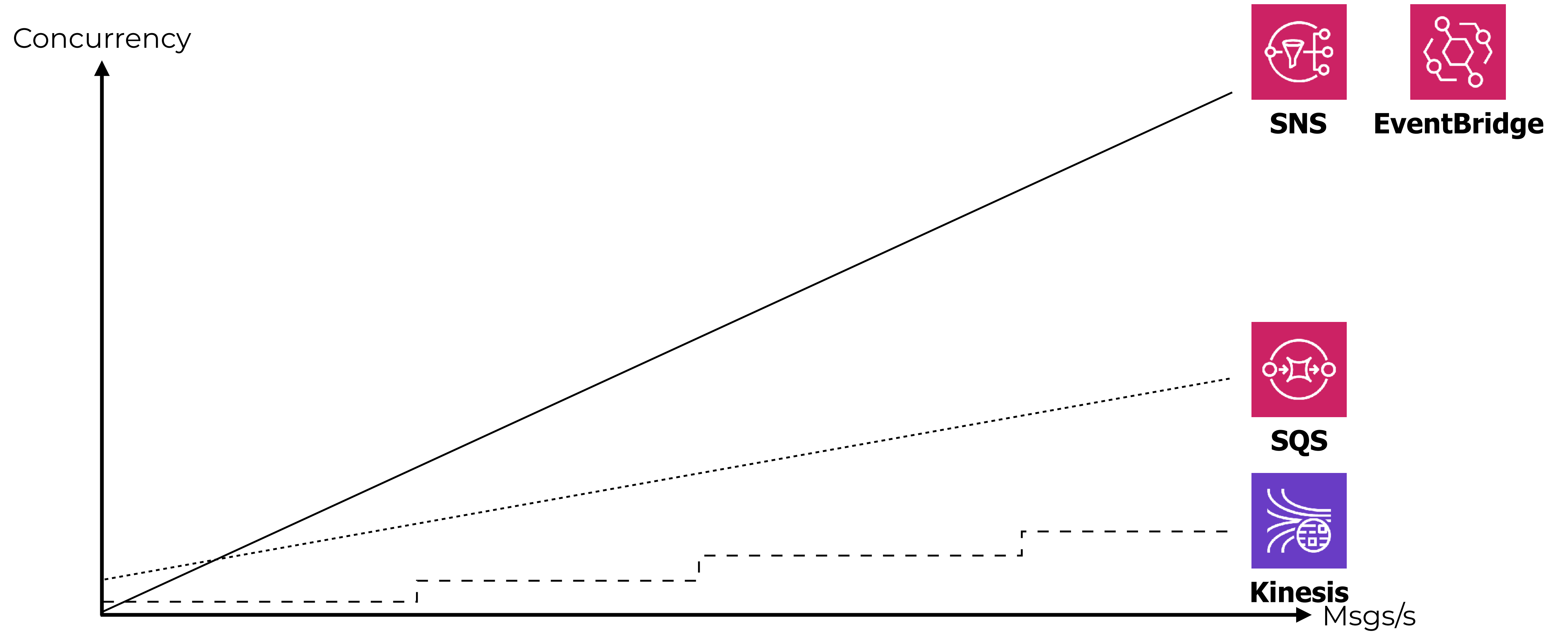
For standard queues, Lambda uses [long polling](#) to poll a queue until it becomes active. When messages are available, Lambda reads up to 5 batches and sends them to your function. If messages are still available, Lambda increases the number of processes that are reading batches by up to 60 more instances per minute. The maximum number of batches that can be processed simultaneously by an event source mapping is 1000.

For FIFO queues, Lambda sends messages to your function in the order that it receives them. When you send a message to a FIFO queue, you specify a [message group ID](#). Amazon SQS ensures that messages in the same group are delivered to Lambda in order. Lambda sorts the messages into groups and sends only one batch at a time for a group. If the function returns an error, all retries are attempted on the affected messages before Lambda receives additional messages from the same group.

Your function can scale in concurrency to the number of active message groups. For more information, see [SQS FIFO as an event source](#) [↗](#) on the AWS Compute Blog.

<http://amzn.to/2EM3bI9>





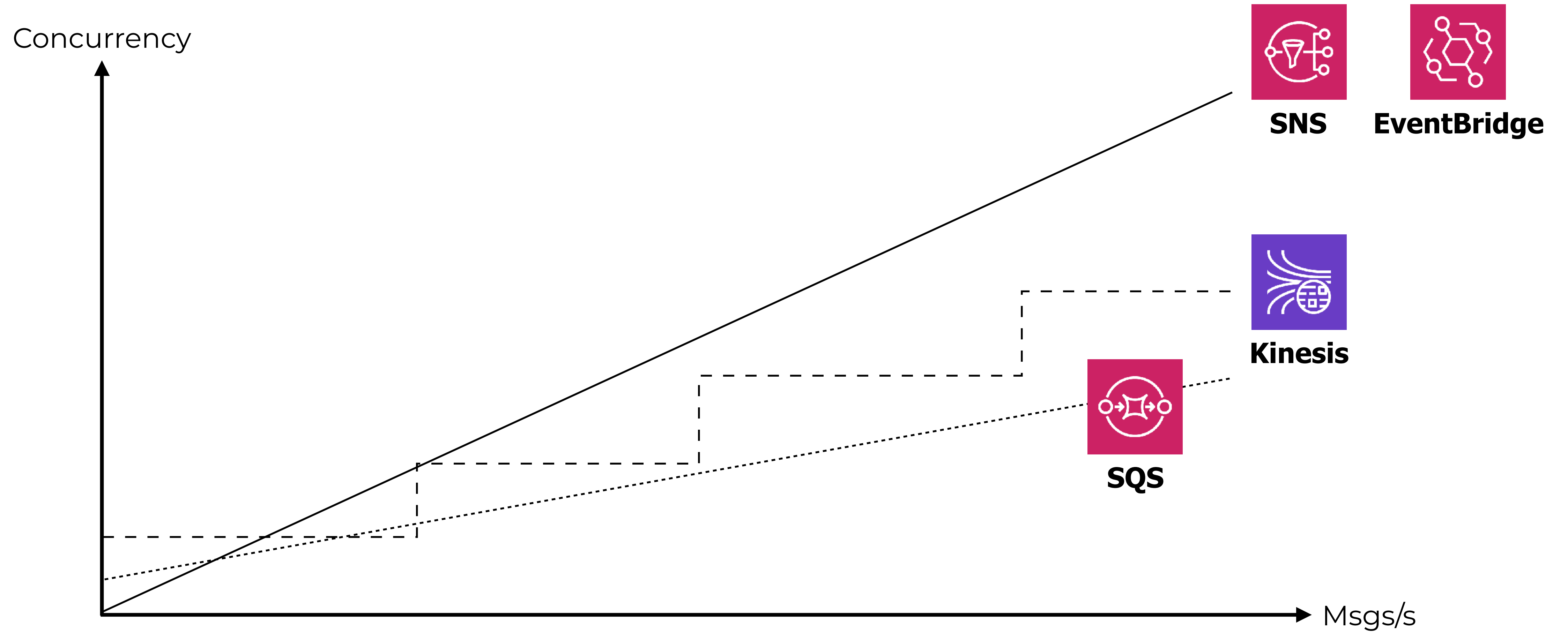
If your function returns an error, Lambda retries the batch until processing succeeds or the data expires. To avoid stalled shards, you can configure the event source mapping to retry with a smaller batch size, limit the number of retries, or discard records that are too old. To retain discarded events, you can configure the event source mapping to send details about failed batches to an SQS queue or SNS topic.

You can also increase concurrency by processing multiple batches from each shard in parallel. Lambda can process up to 10 batches in each shard simultaneously. If you increase the number of concurrent batches per shard, Lambda still ensures in-order processing at the partition-key level.

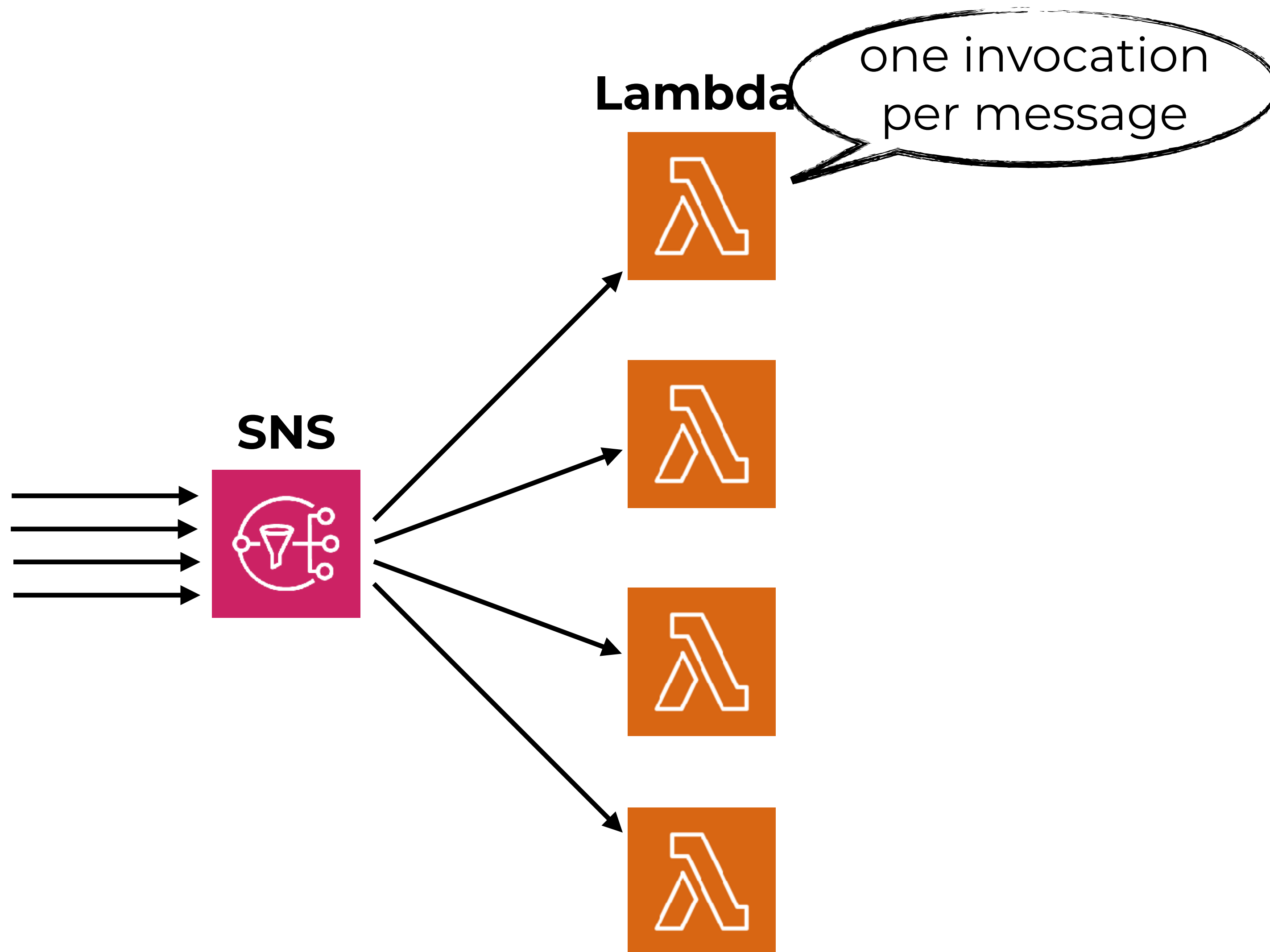
Sections

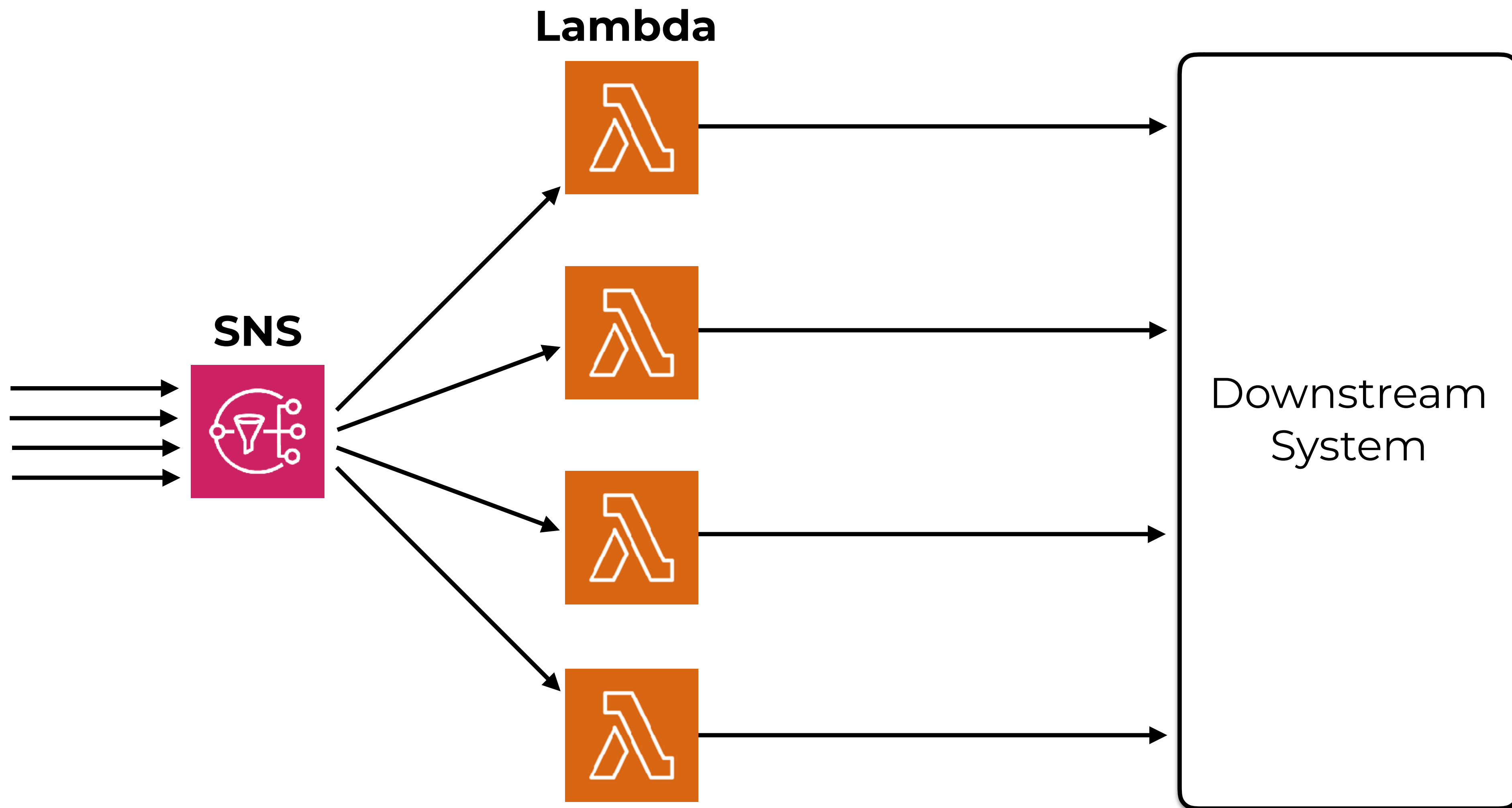
- [Configuring Your Data Stream and Function](https://amzn.to/2RudmGV)

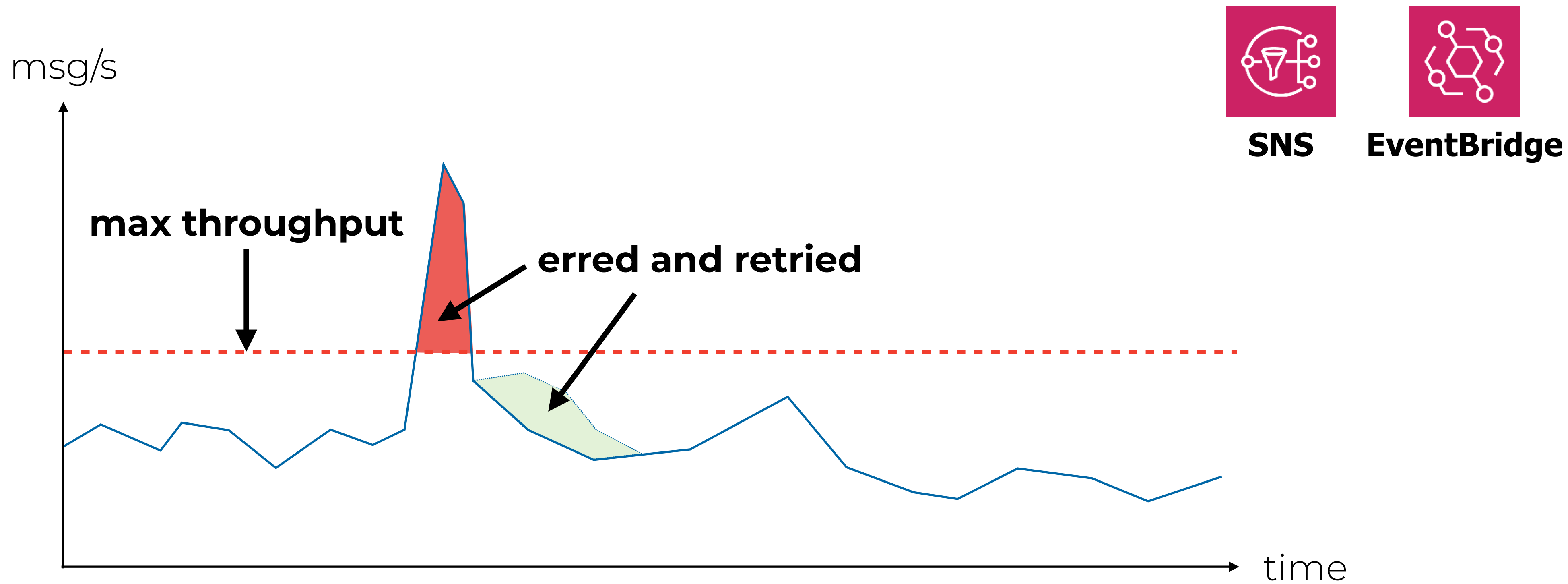
<https://amzn.to/2RudmGV>

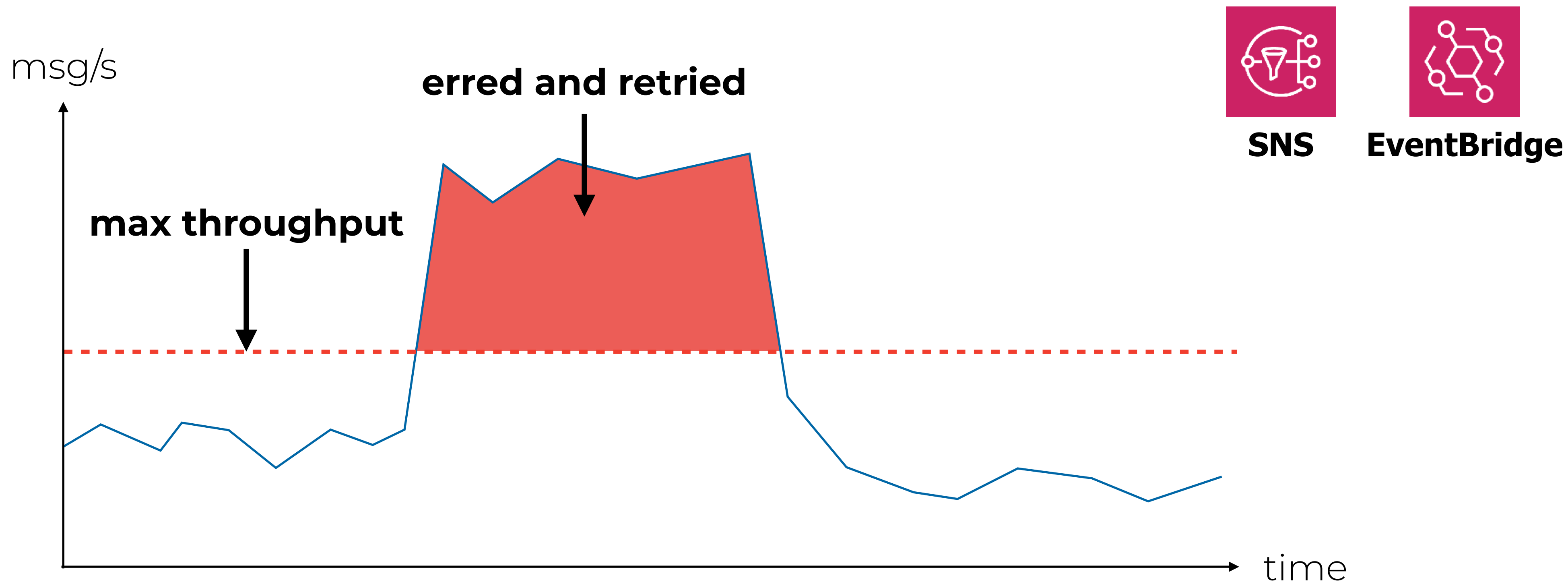


Controlling concurrency









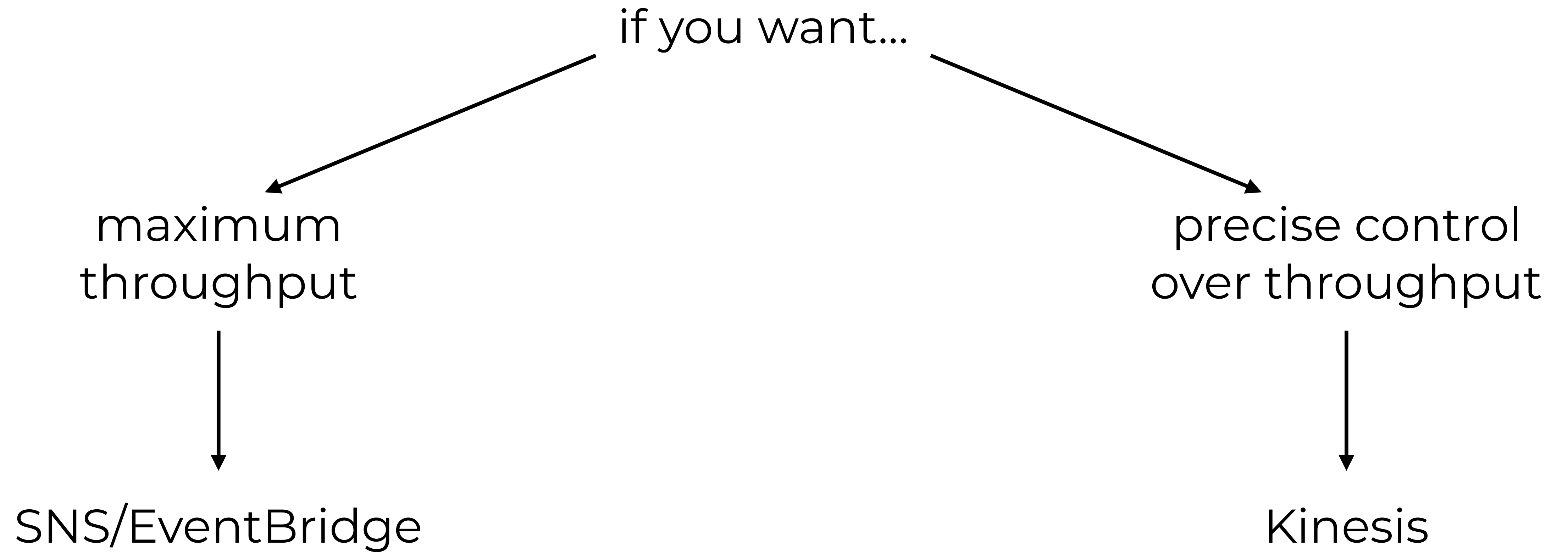
if you want...

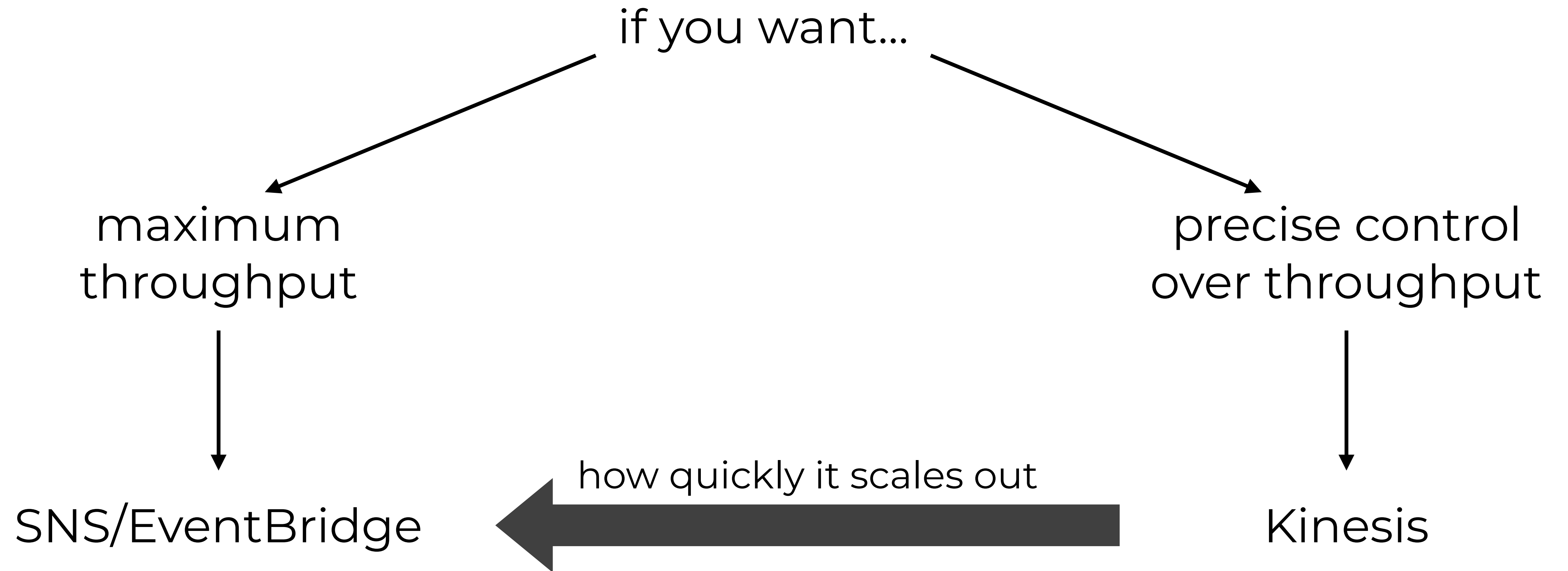
maximum
throughput

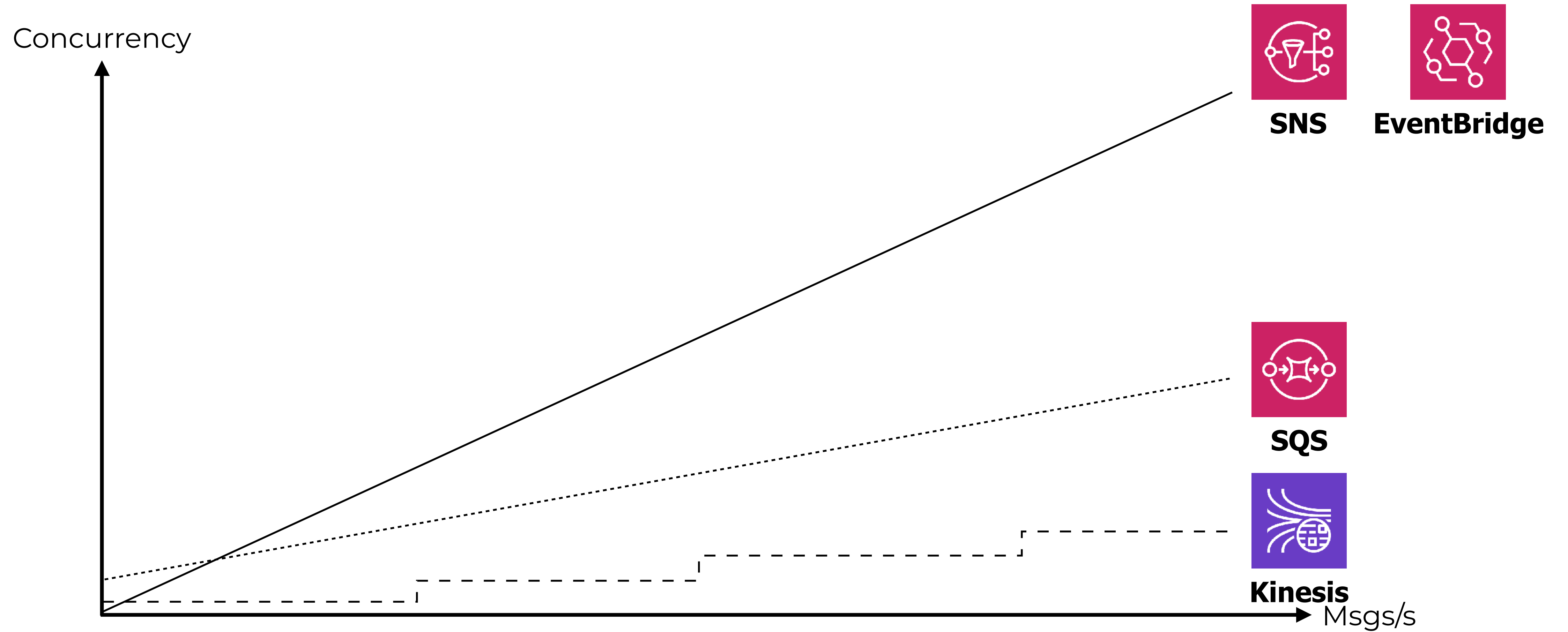
precise control
over throughput

SNS/EventBridge

Kinesis

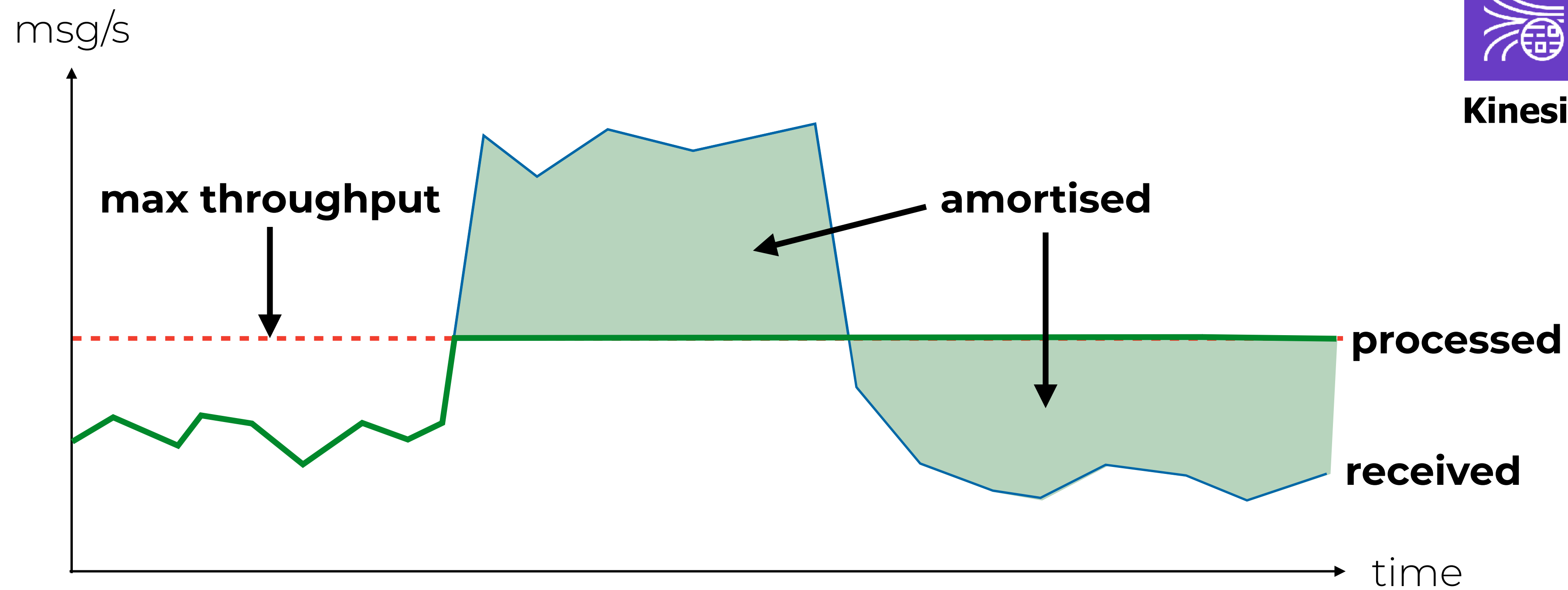








Kinesis



Event Source Options

- **Kinesis stream** – The Kinesis stream to read records from.
- **Consumer** (optional) – Use a stream consumer to read from the stream over a dedicated connection.
- **Batch size** – The number of records to send to the function in each batch, up to 10,000. Lambda passes all of the records in the batch to the function in a single call, as long as the total size of the events doesn't exceed the [payload limit](#) for synchronous invocation (6 MB).
- **Batch window** – Specify the maximum amount of time to gather records before invoking the function, in seconds.
- **Starting position** – Process only new records, all existing records, or records created after a certain date.
 - **Latest** – Process new records that are added to the stream.
 - **Trim horizon** – Process all records in the stream.
 - **At timestamp** – Process records starting from a specific time.

After processing any existing records, the function is caught up and continues to process new records.

- **On-failure destination** – An SQS queue or SNS topic for records that can't be processed. When Lambda discards a batch of records because it's too old or has exhausted all retries, it sends details about the batch to the queue or topic.
- **Retry attempts** – The maximum number of times that Lambda retries when the function returns an error. This doesn't apply to service errors or throttles where the batch didn't reach the function.
- **Maximum age of record** – The maximum age of a record that Lambda sends to your function.
- **Split batch on error** – When the function returns an error, split the batch into two before retrying.
- **Concurrent batches per shard** – Process multiple batches from the same shard concurrently.
- **Enabled** – Disable the event source to stop processing records. Lambda keeps track of the last record processed and resumes processing from that point when it's reenabled.

<https://amzn.to/2RudmGV>

A self-healing Kinesis function that adapts its throughput based on performance

[AWS, Design Patterns, Kinesis, Lambda, Programming, Serverless / May 20, 2019](#)

Check out my new course [Learn you some Lambda best practice for great good!](#) and learn the best practices for performance, cost, security, resilience, observability and scalability.



I spent time with a client this week to solve an interesting problem – to adjust the number of concurrent requests to a downstream service dynamically based on response time and error rate. This is a common challenge when integrating with third parties, so we decided to share our approach so others might benefit from it.

The problem

My client is in the financial services sector and they integrate with over 50 service providers. They have API access to the providers' data and they need to fetch them from every provider on a daily basis.

However, many of these providers place restrictions and rate limits on their APIs, for example:

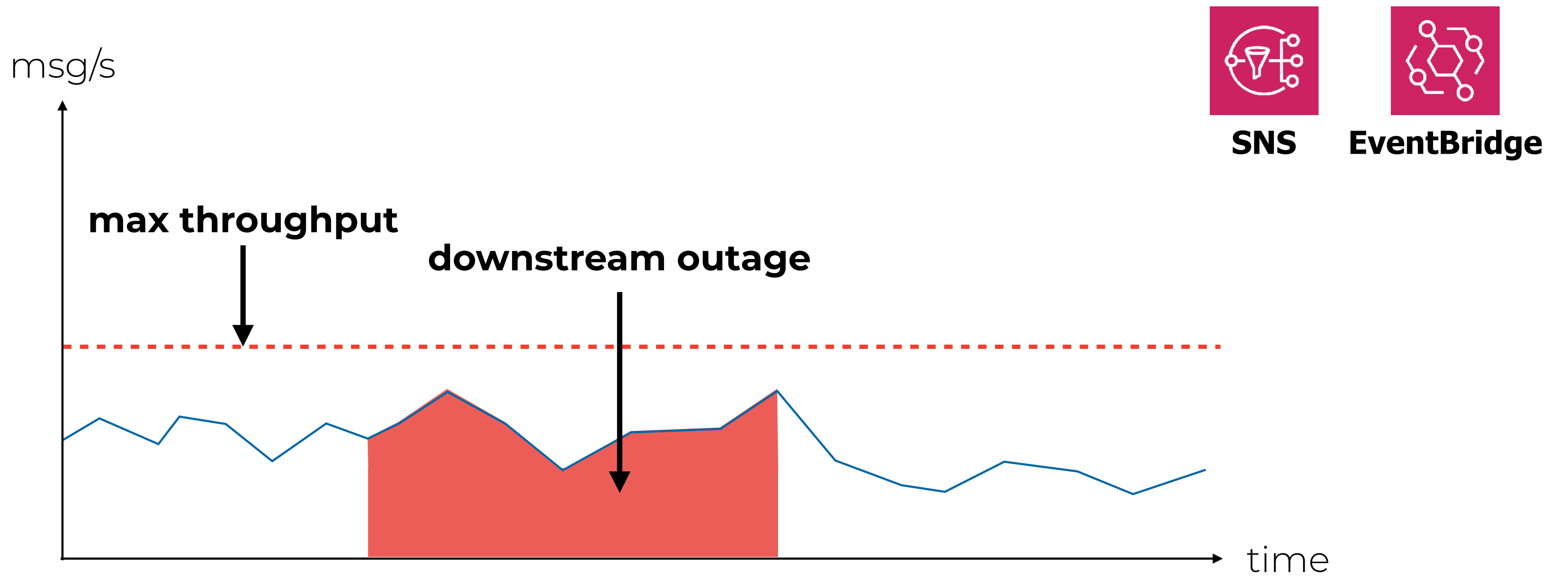
- You can only access the API during off-peak hours.
- You cannot access the API during the weekly maintenance window.
- You can only make X concurrent requests to the API.

My client has implemented a *scheduler* service to manage the timing constraints. The *scheduler* knows **when** to kick off the ingestion process for each provider. But there's also a secondary process in which their users (financial advisers) can schedule ad-hoc ingestions themselves.

And there are other additional contexts that we had to keep in mind:

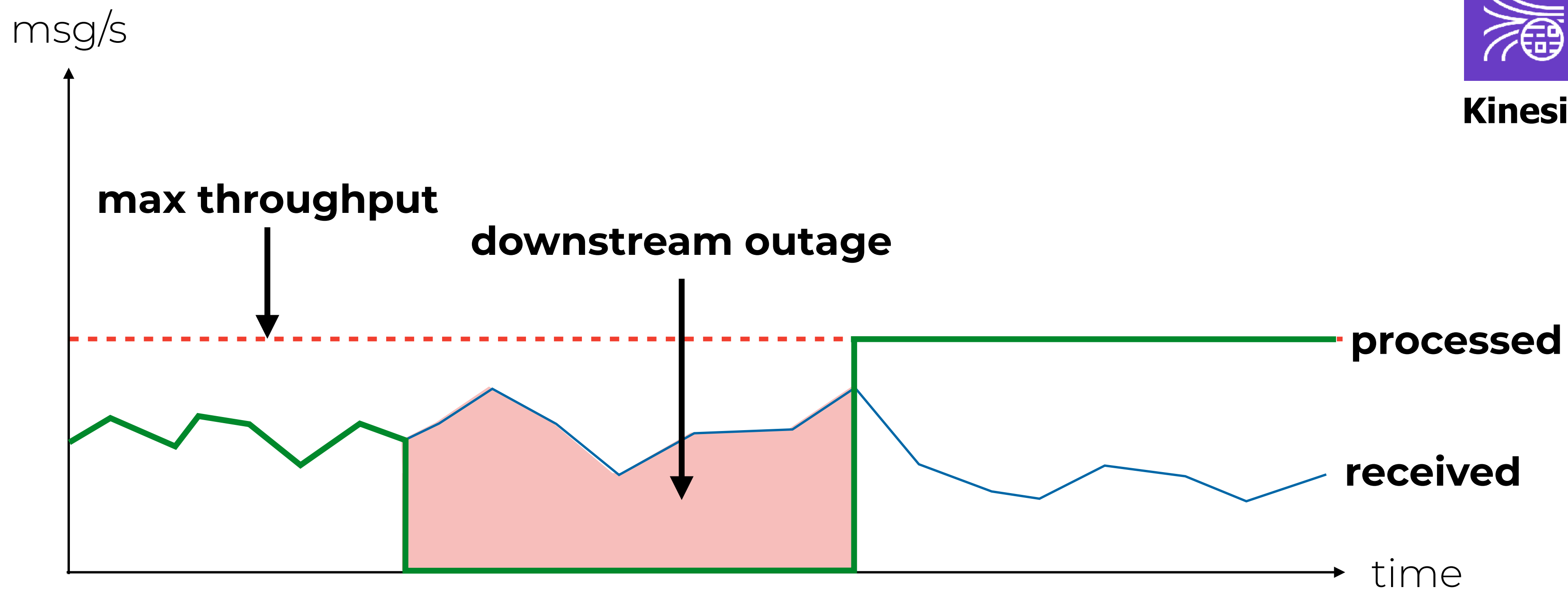
- The APIs follow an established industry standard. Unfortunately, they can only return one record at a time.

<https://theburningmonk.com/2019/05/a-self-healing-kinesis-function-that-adapts-its-throughput-based-on-performance>





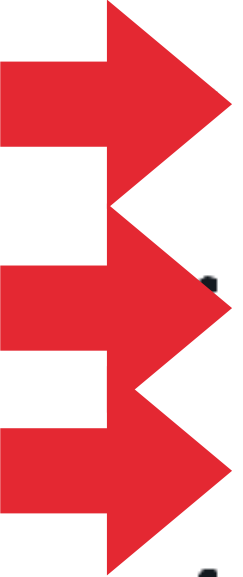
Kinesis



Event Source Options

- **Kinesis stream** – The Kinesis stream to read records from.
- **Consumer** (optional) – Use a stream consumer to read from the stream over a dedicated connection.
- **Batch size** – The number of records to send to the function in each batch, up to 10,000. Lambda passes all of the records in the batch to the function in a single call, as long as the total size of the events doesn't exceed the [payload limit](#) for synchronous invocation (6 MB).
- **Batch window** – Specify the maximum amount of time to gather records before invoking the function, in seconds.
- **Starting position** – Process only new records, all existing records, or records created after a certain date.
 - **Latest** – Process new records that are added to the stream.
 - **Trim horizon** – Process all records in the stream.
 - **At timestamp** – Process records starting from a specific time.

After processing any existing records, the function is caught up and continues to process new records.



On-failure destination – An SQS queue or SNS topic for records that can't be processed. When Lambda discards a batch of records because it's too old or has exhausted all retries, it sends details about the batch to the queue or topic.

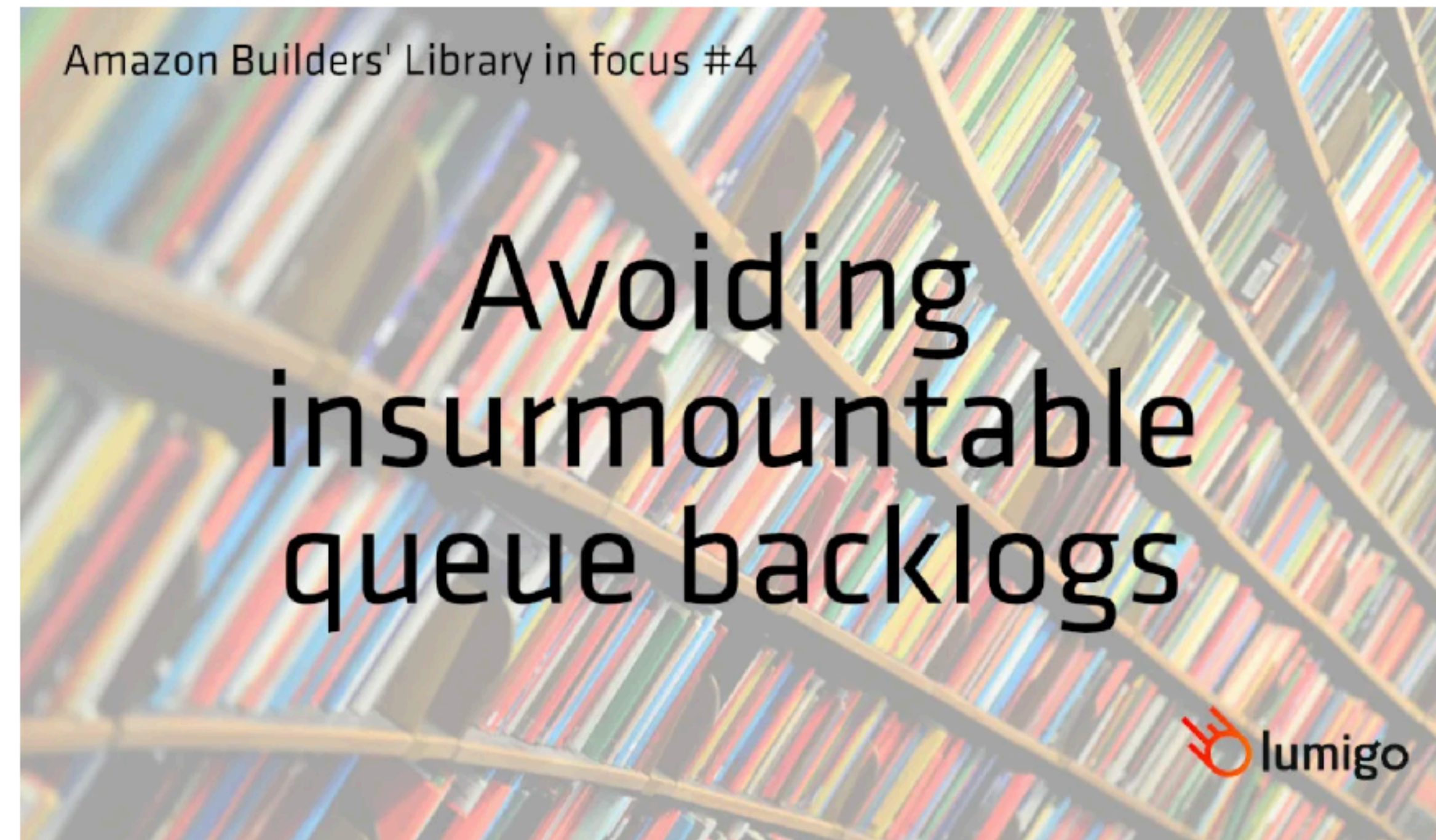
• **Retry attempts** – The maximum number of times that Lambda retries when the function returns an error. This doesn't apply to service errors or throttles where the batch didn't reach the function.

• **Maximum age of record** – The maximum age of a record that Lambda sends to your function.

- **Split batch on error** – When the function returns an error, split the batch into two before retrying.
- **Concurrent batches per shard** – Process multiple batches from the same shard concurrently.
- **Enabled** – Disable the event source to stop processing records. Lambda keeps track of the last record processed and resumes processing from that point when it's reenabled.

<https://amzn.to/2RudmGV>

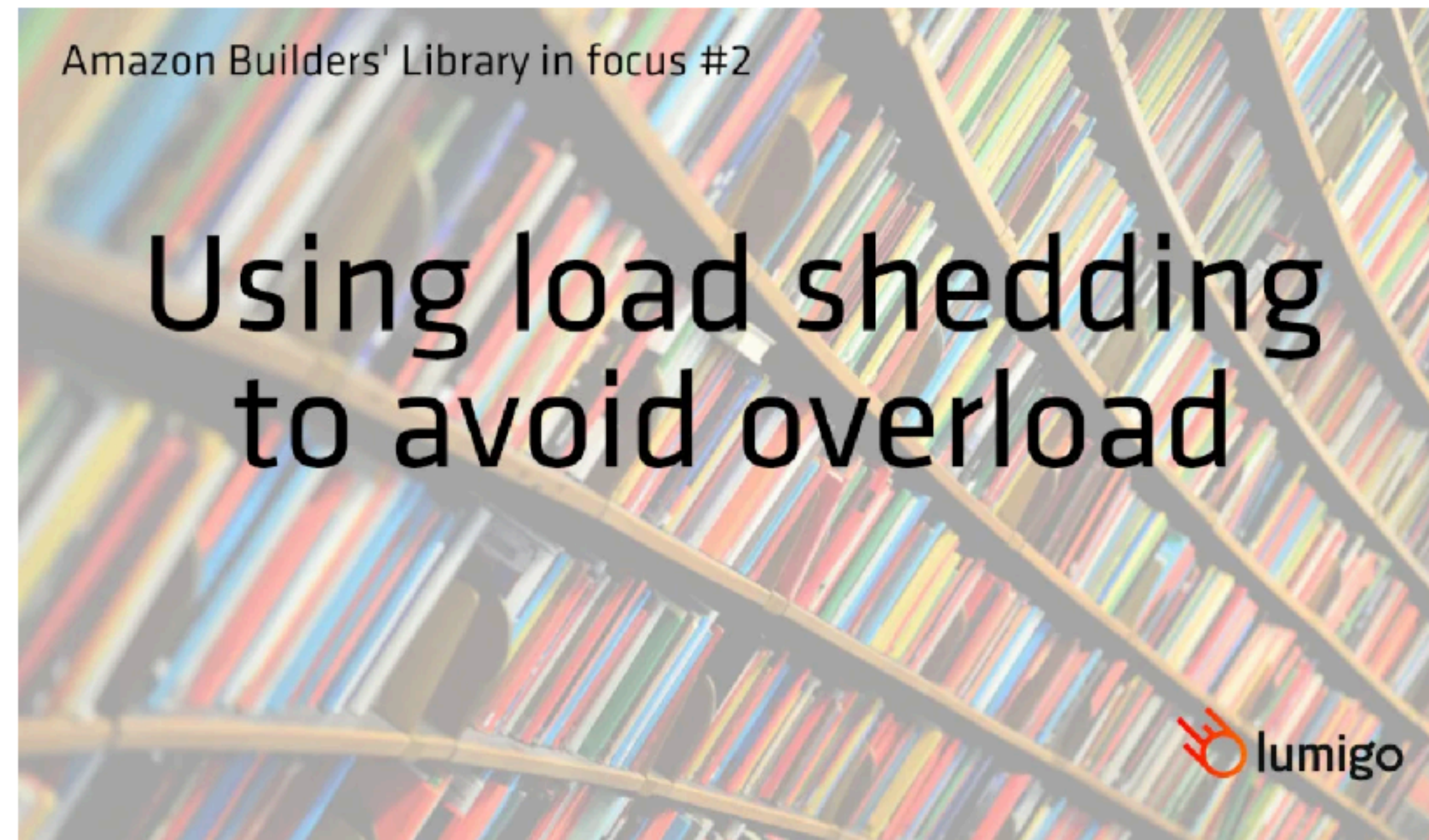
AMAZON BUILDERS' LIBRARY IN FOCUS #4: AVOIDING INSURMOUNTABLE QUEUE BACKLOGS



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>

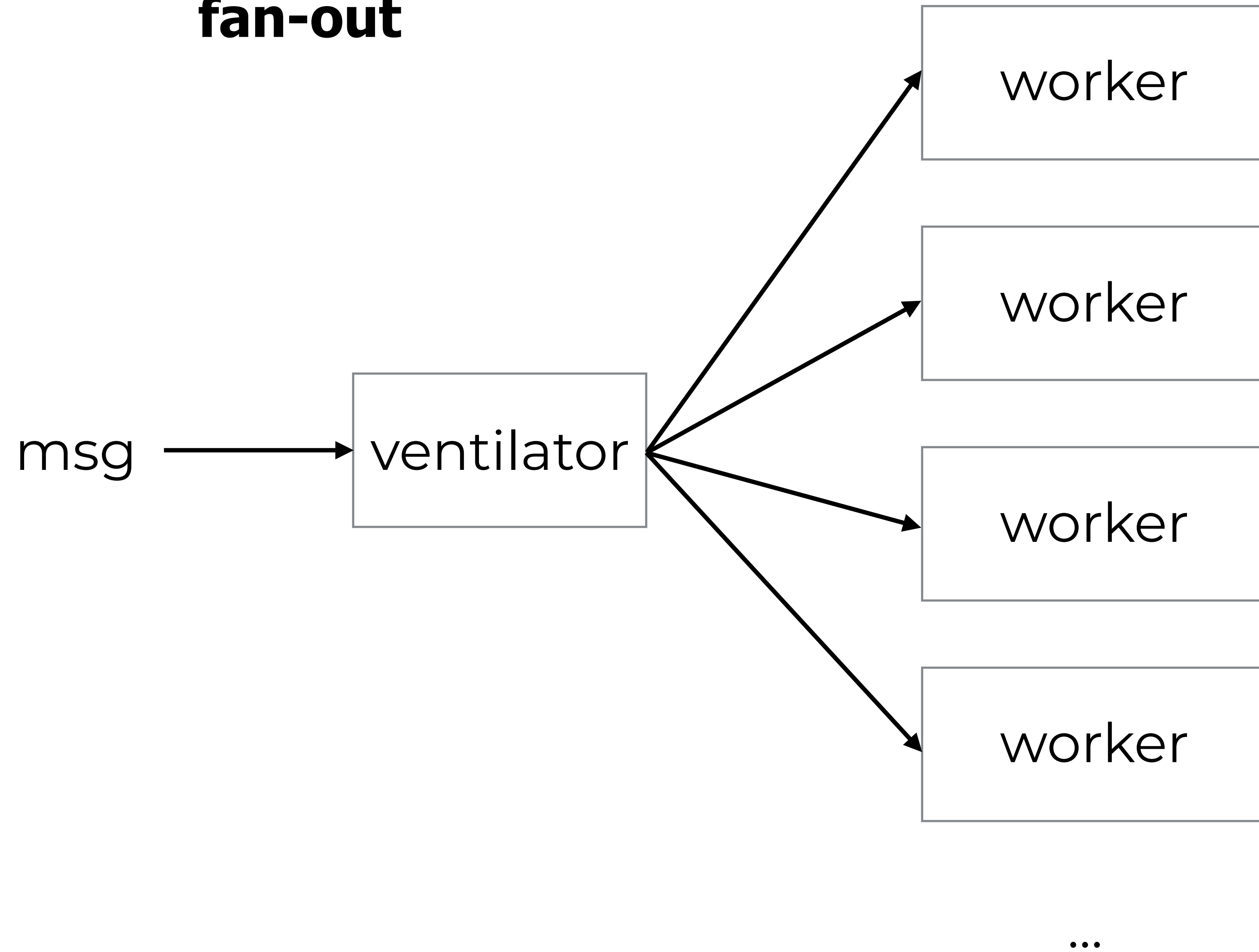
AMAZON BUILDERS' LIBRARY IN FOCUS #2: USING LOAD SHEDDING TO AVOID OVERLOAD



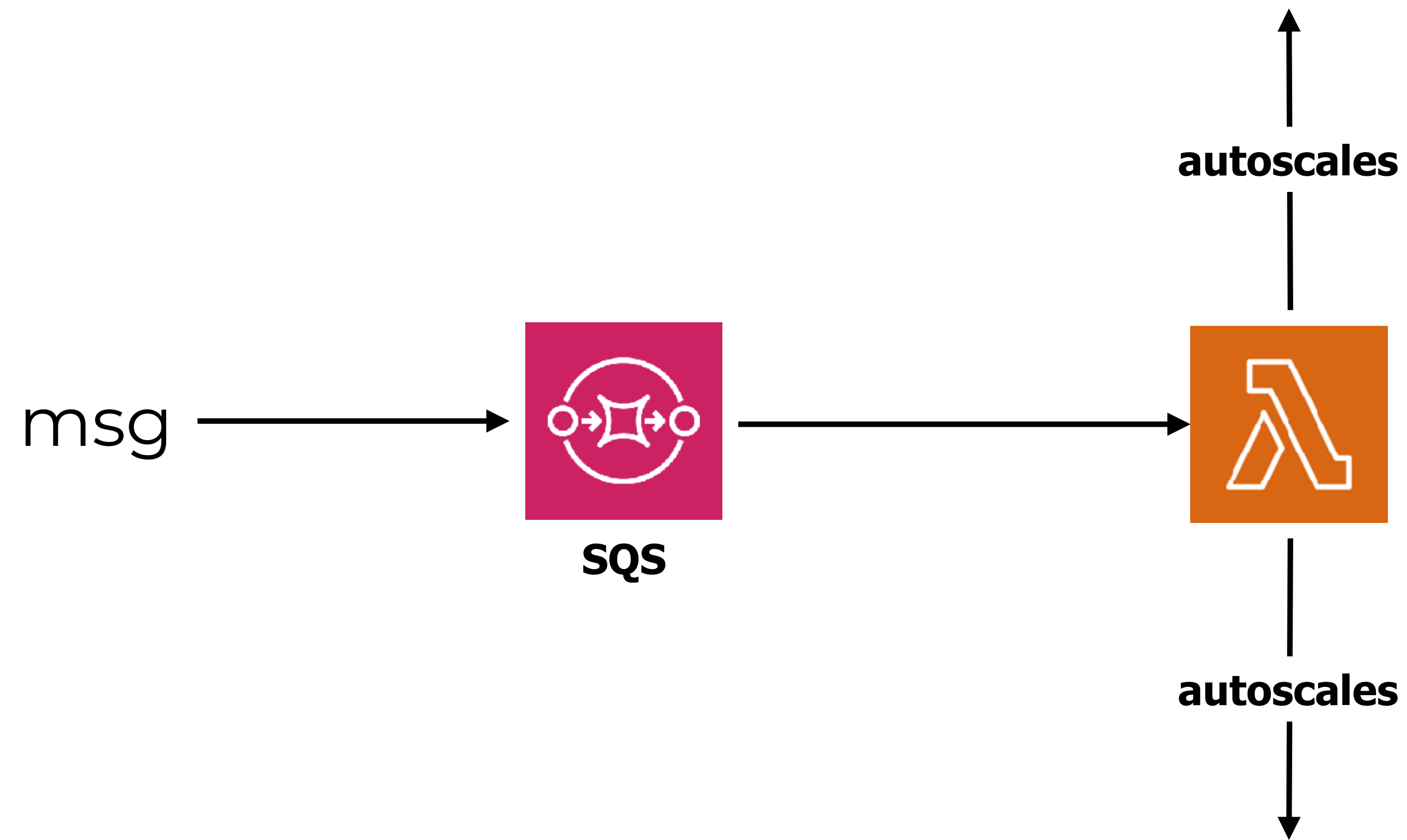
In the second of our new series of posts, Yan Cui highlights the key insights from the Amazon Builders' Library article, [Using load shedding to avoid overload](https://lumigo.io/blog/amazon-builders-library-in-focus-2-using-load-shedding-to-avoid-overload/), by AWS Principal Engineer (AWS Lambda) **David Yanacek**.

<https://lumigo.io/blog/amazon-builders-library-in-focus-2-using-load-shedding-to-avoid-overload/>

fan-out

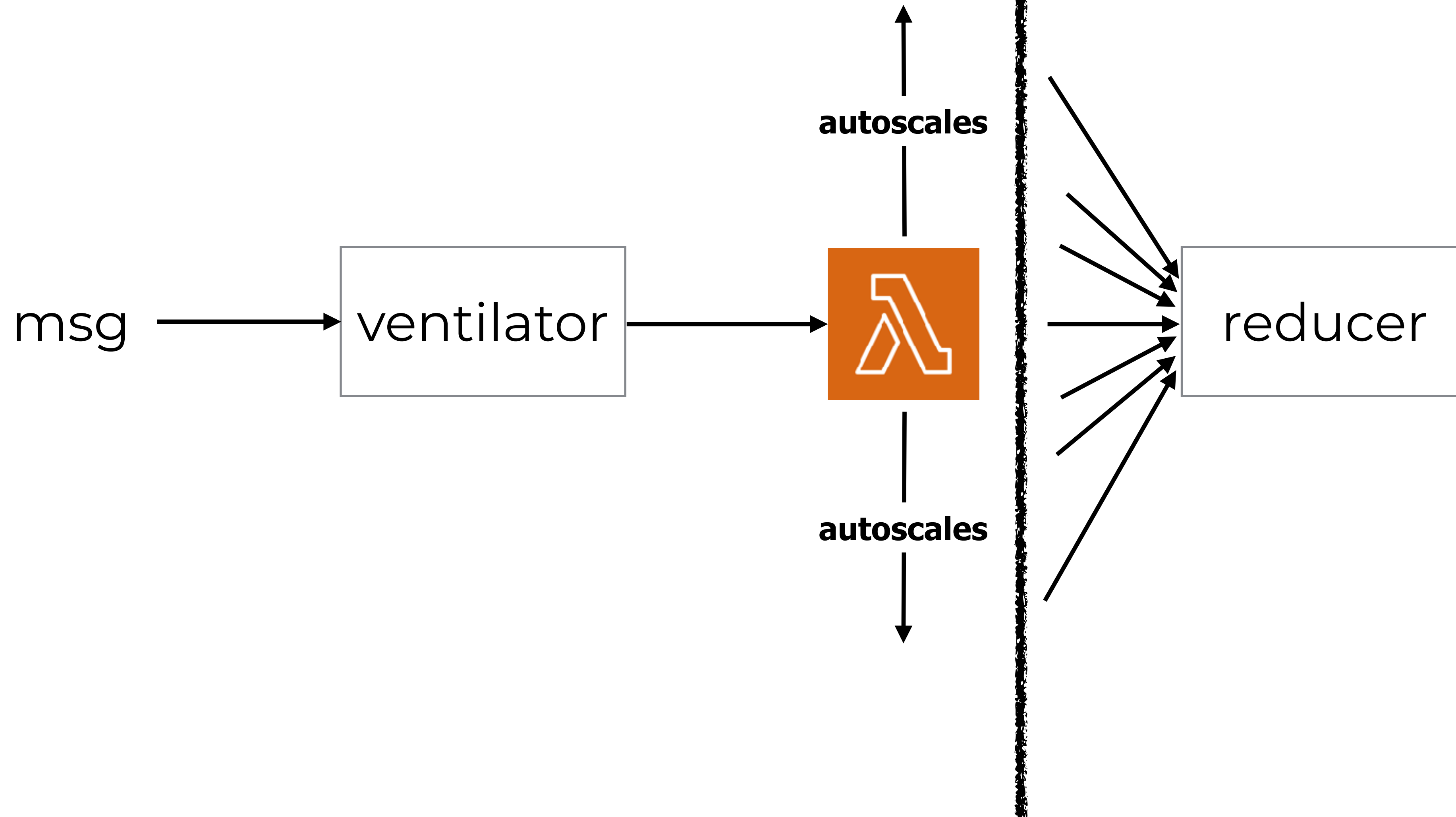


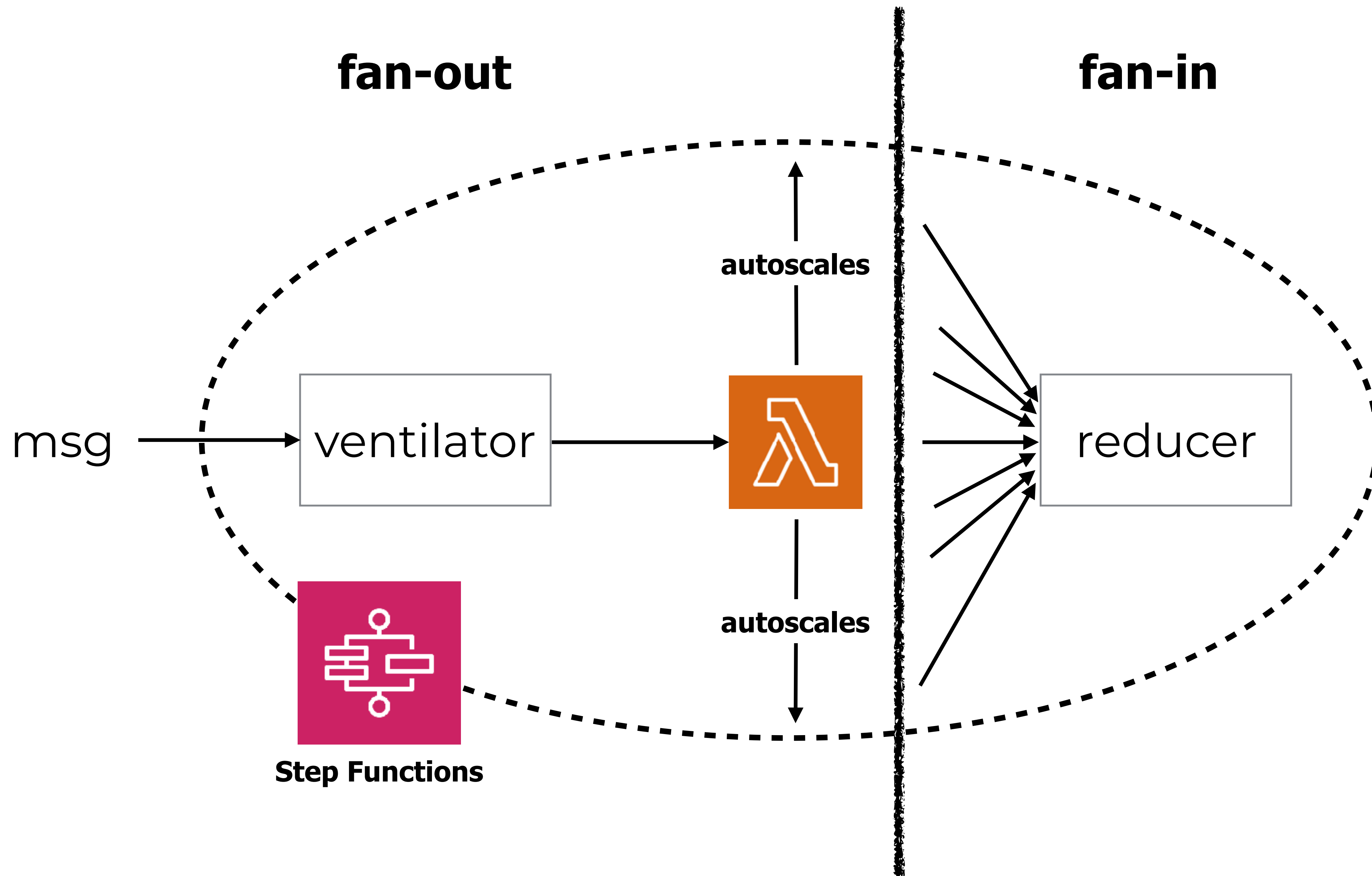
good for **divide-and-conquer** to improve
throughput and processing time

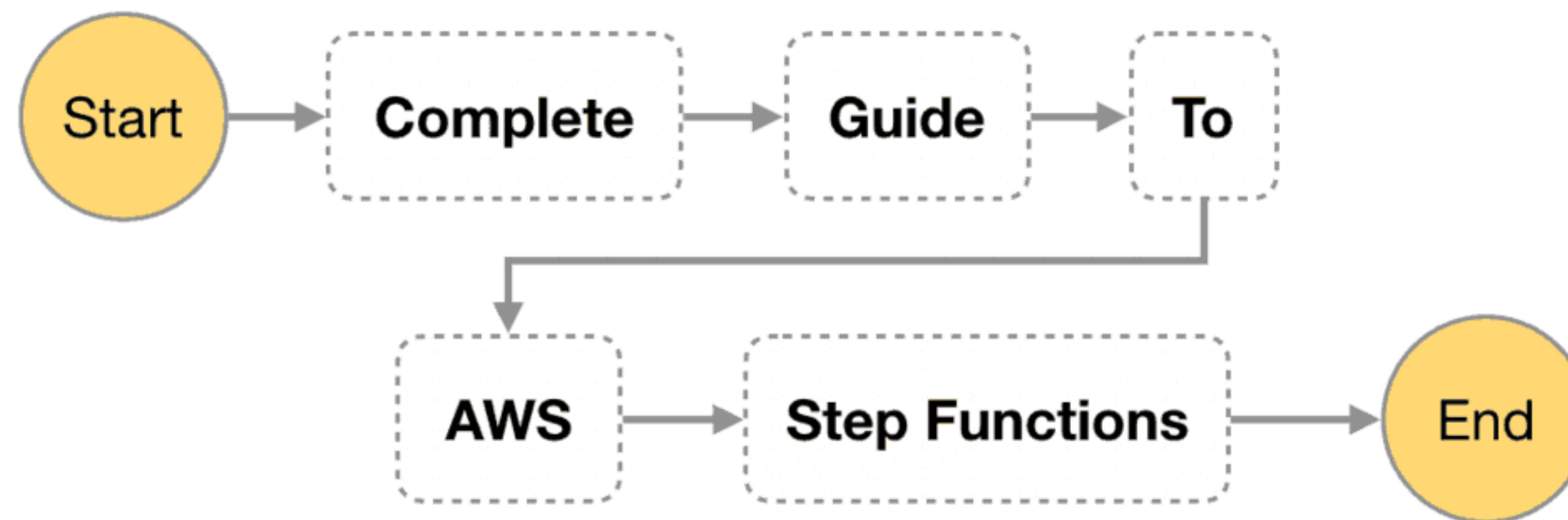


fan-out

fan-in







<https://bit.ly/complete-guide-to-aws-step-functions>

	SNS	SNS FIFO	SQS	SQS FIFO	EventBridge	Kinesis	DynamoDB Streams
Subscribers	one-to-many	one-to-many	one-to-one	one-to-one	one-to-many	one-to-many	one-to-many
Ordering	none	by message group ID	none	by message group ID	none	by shard	by shard (internal)
Replay Events	none	none	none	none	archive and replay	up to a year	up to 24 hours
Batch Processing	none	none	batch window (up to 5 mins)	batch window (up to 5 mins)	none	up to 10,000 records	up to 1,000 records
Retry	up to 2 retries + DLQ	up to 2 retries + DLQ	redrive policy	redrive policy	up to 2 retries + DLQ	up to 10,000 retries + failure destination	up to 10,000 retries + failure destination
Concurrency	fan-out	fan-out	auto-scaled	auto-scaled	fan-out	1-10 per shard	1-10 per shard