AWS re:Invent

NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

SVS305

Understanding AWS Lambda concurrency

Brian Zambrano (he/him)
Sr. Specialist Solutions Architect, Serverless
Amazon Web Services

Justin Pirtle (he/him)
Principal Solutions Architect
Amazon Web Services

"We want a serverless architecture that supports 10,000 requests per second."

- The business

“While keeping costs under control,

with good performance,

and integration with our relational database.”

AWS Lambda concurrency affects all of this

Agenda

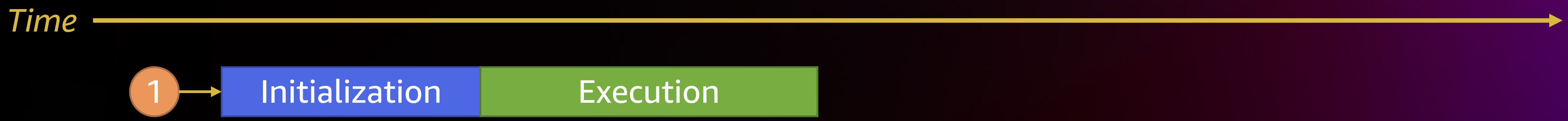
What we will cover

- AWS Lambda concurrency and concurrency controls
- Applying concurrency to a sample Serverless API architecture
- Best practices for scaling synchronous and asynchronous workloads

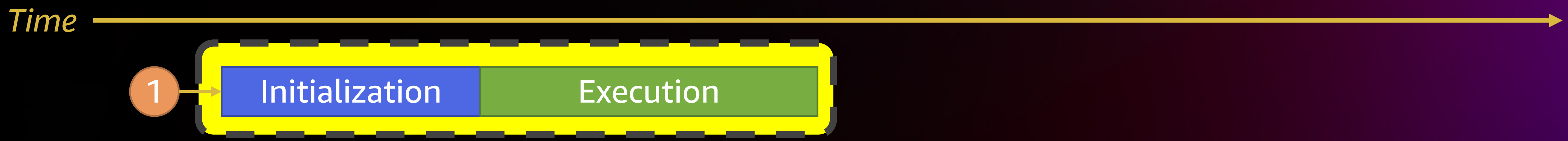
What we will not cover

- What is serverless?
- What are Amazon API Gateway, Amazon DynamoDB, AWS Lambda?

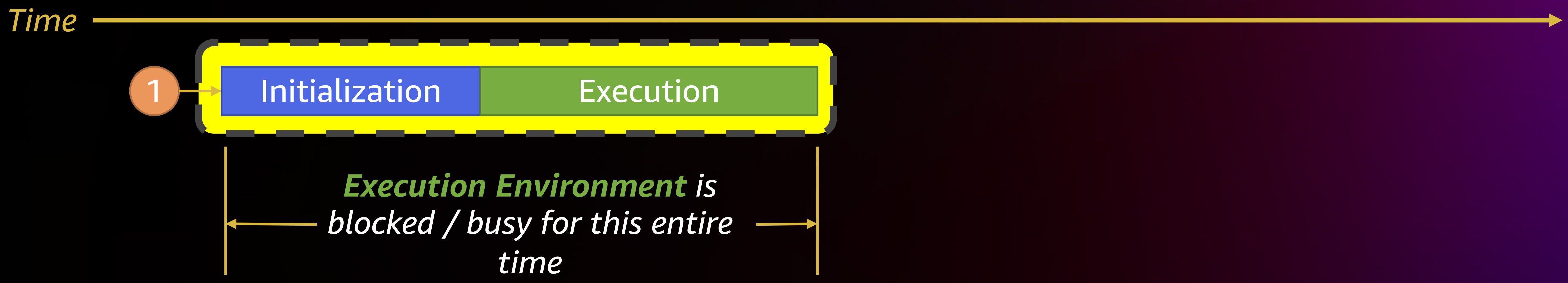
Lambda concurrency



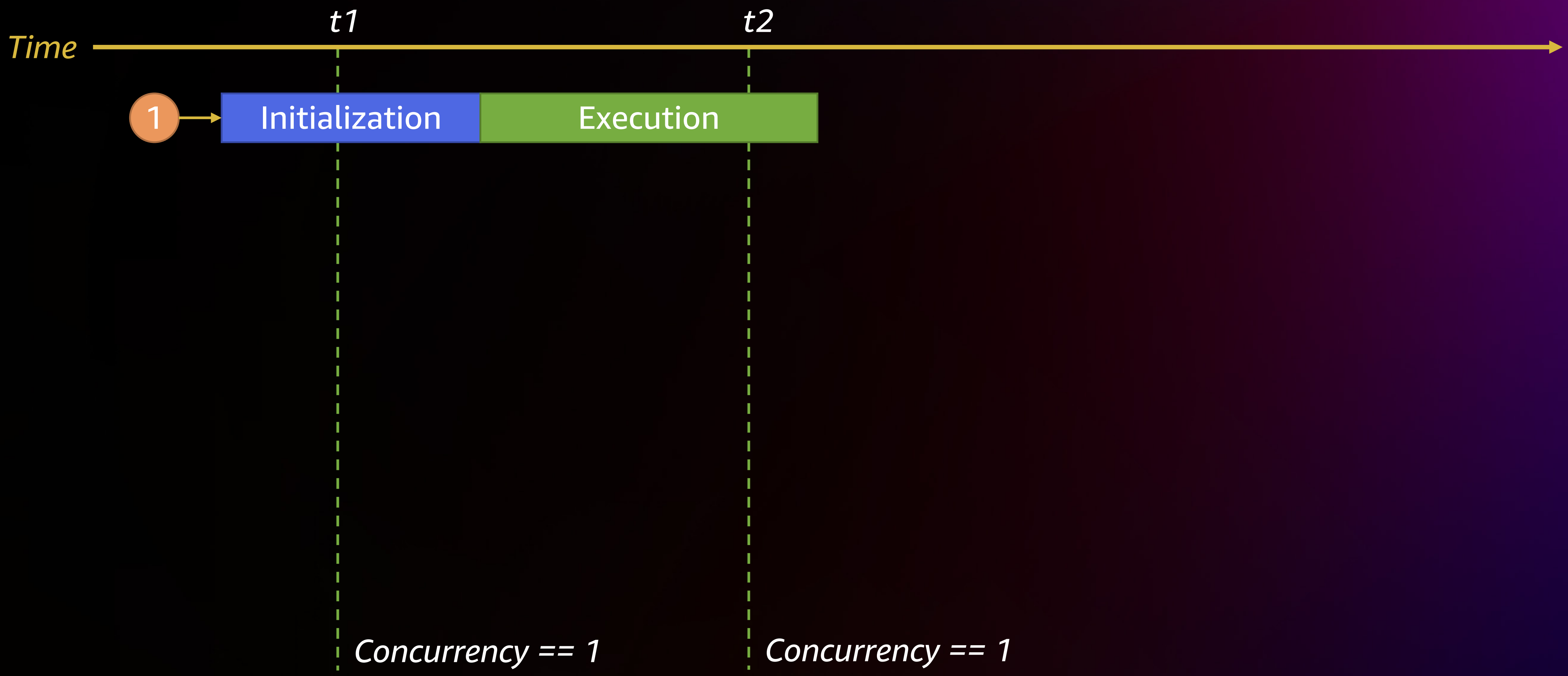
Lambda concurrency



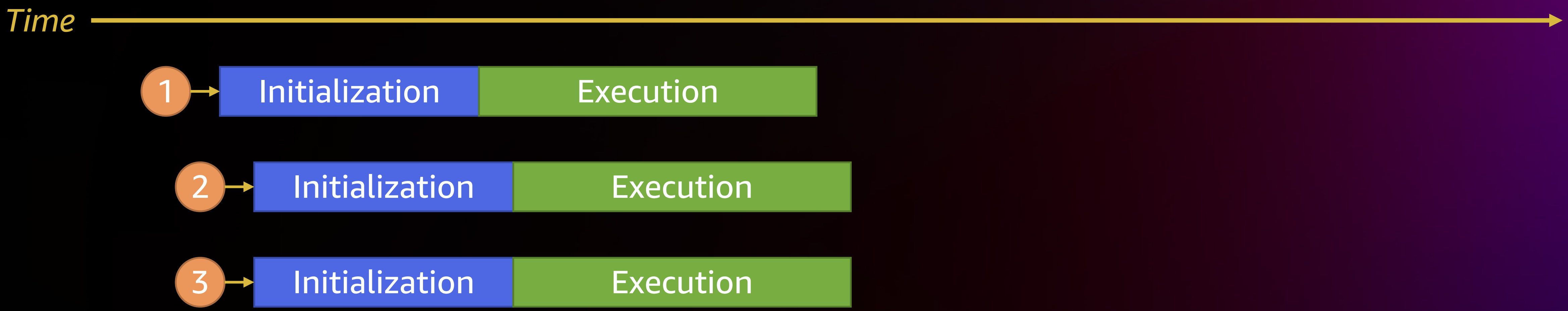
Lambda concurrency


Time
1
Initialization
Execution
Execution Environment is blocked / busy for this entire time

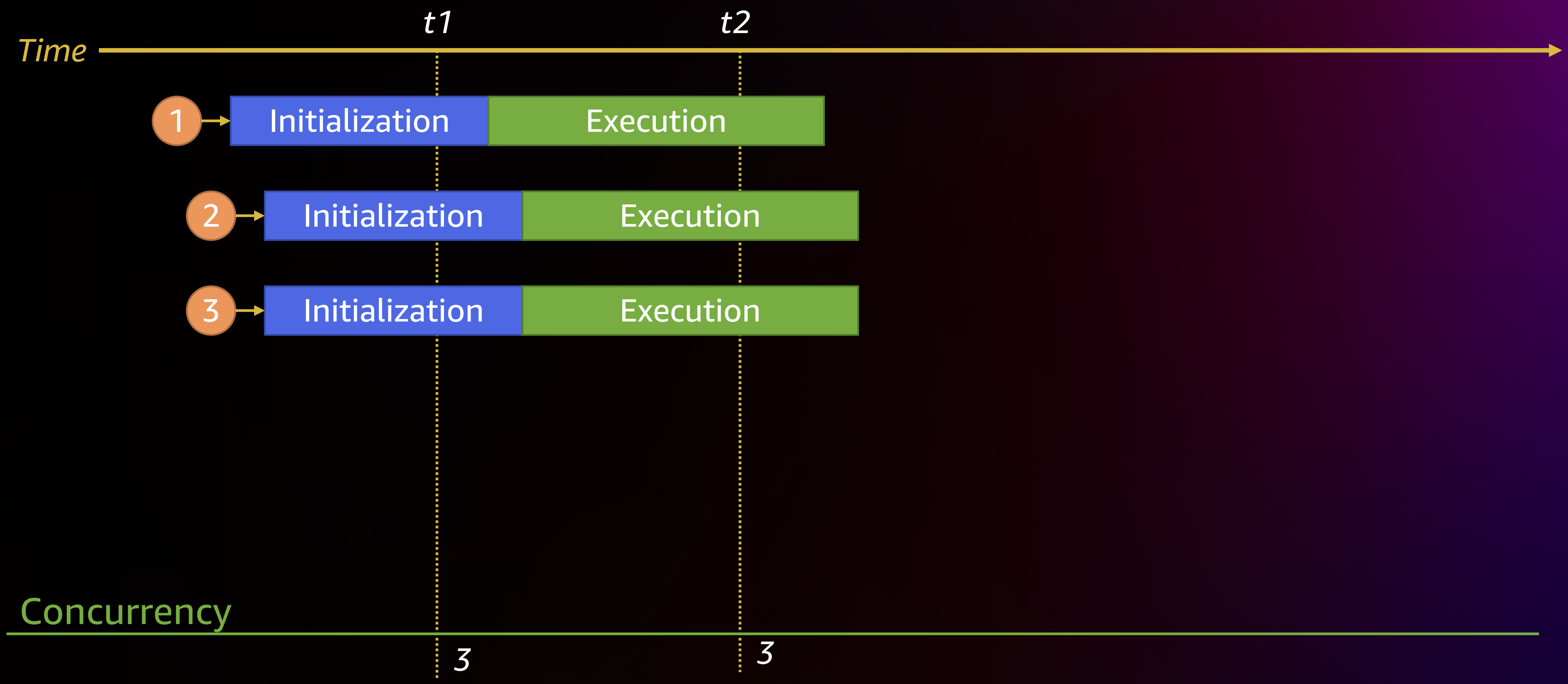
Lambda concurrency



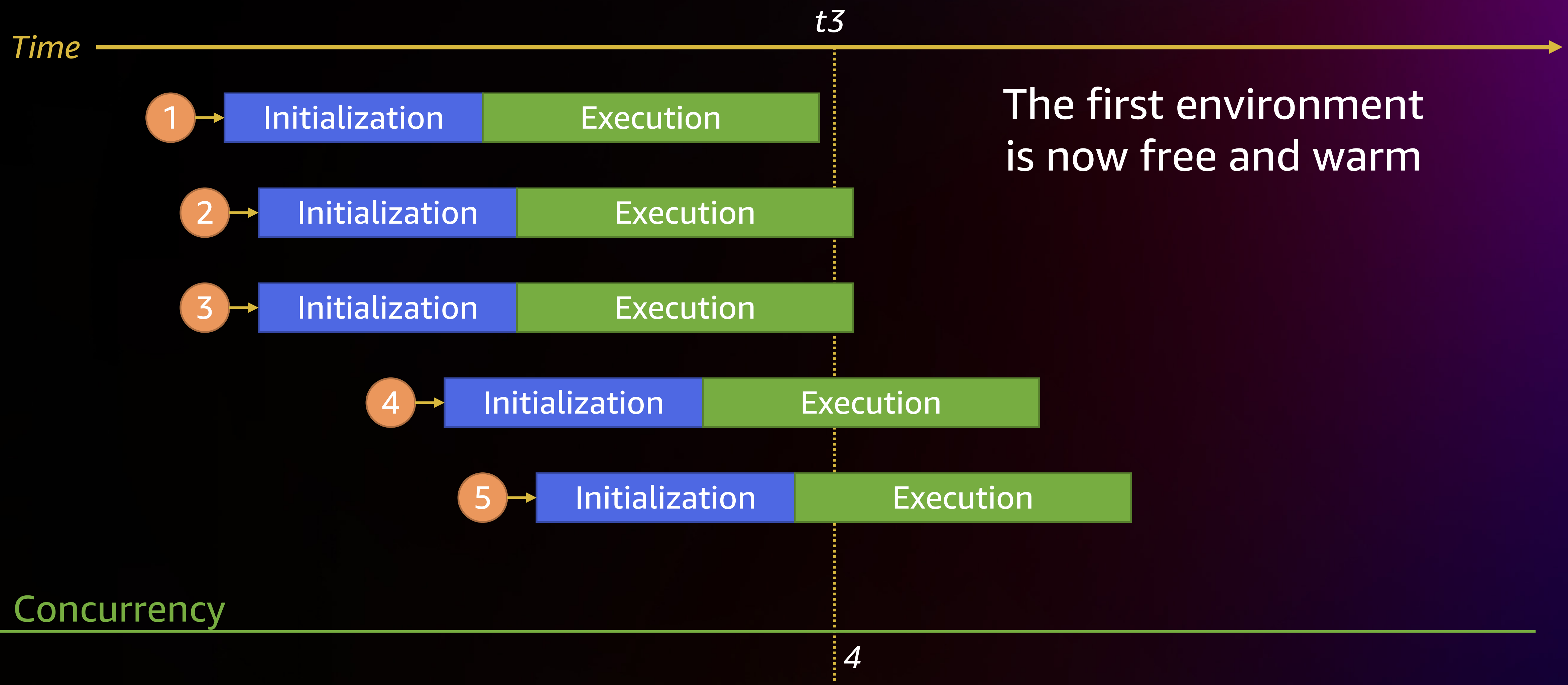
Lambda concurrency



Lambda concurrency

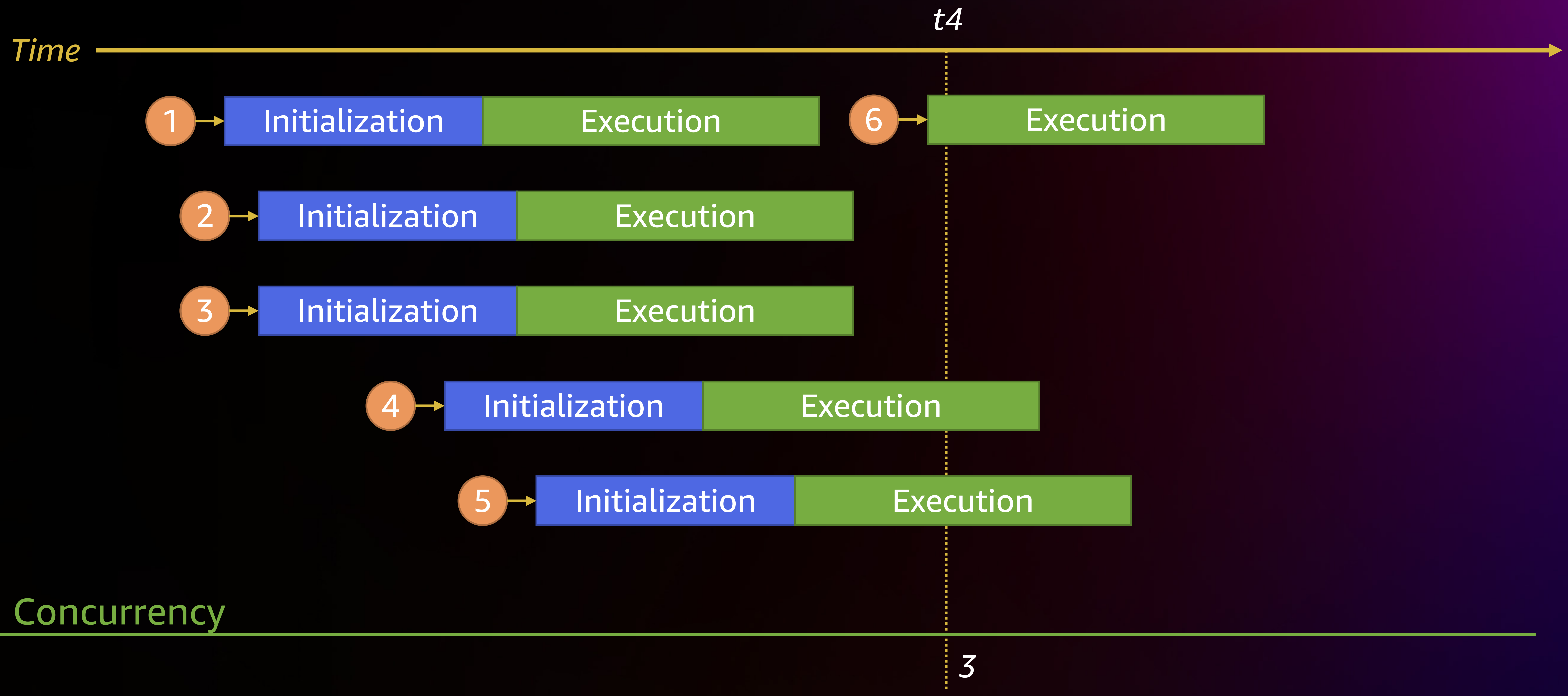


Lambda concurrency

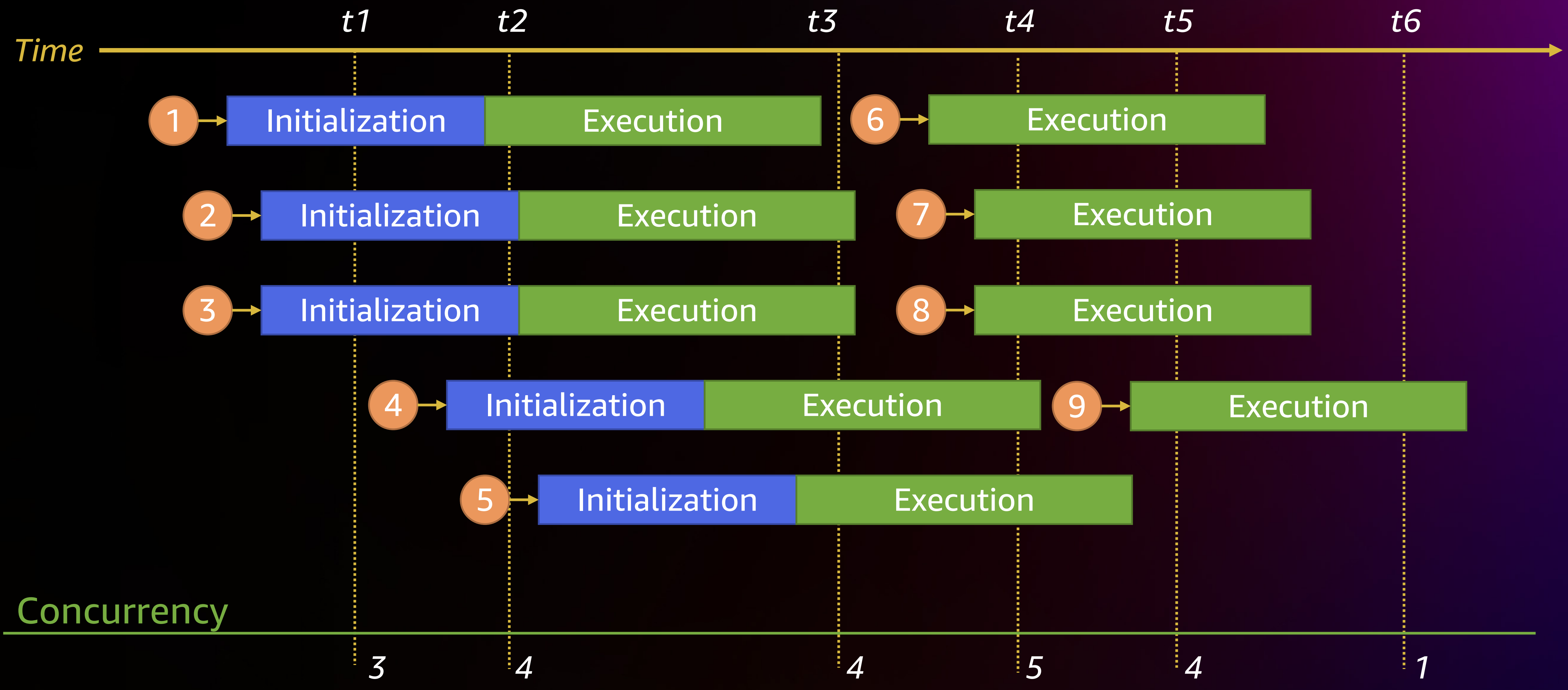


The first environment is now free and warm

Lambda concurrency



Lambda concurrency



Concurrency is a point-in-time measurement

Concurrency is a point-in-time measurement

and not the same as requests per second.

How to calculate concurrency requirements

Lambda concurrency

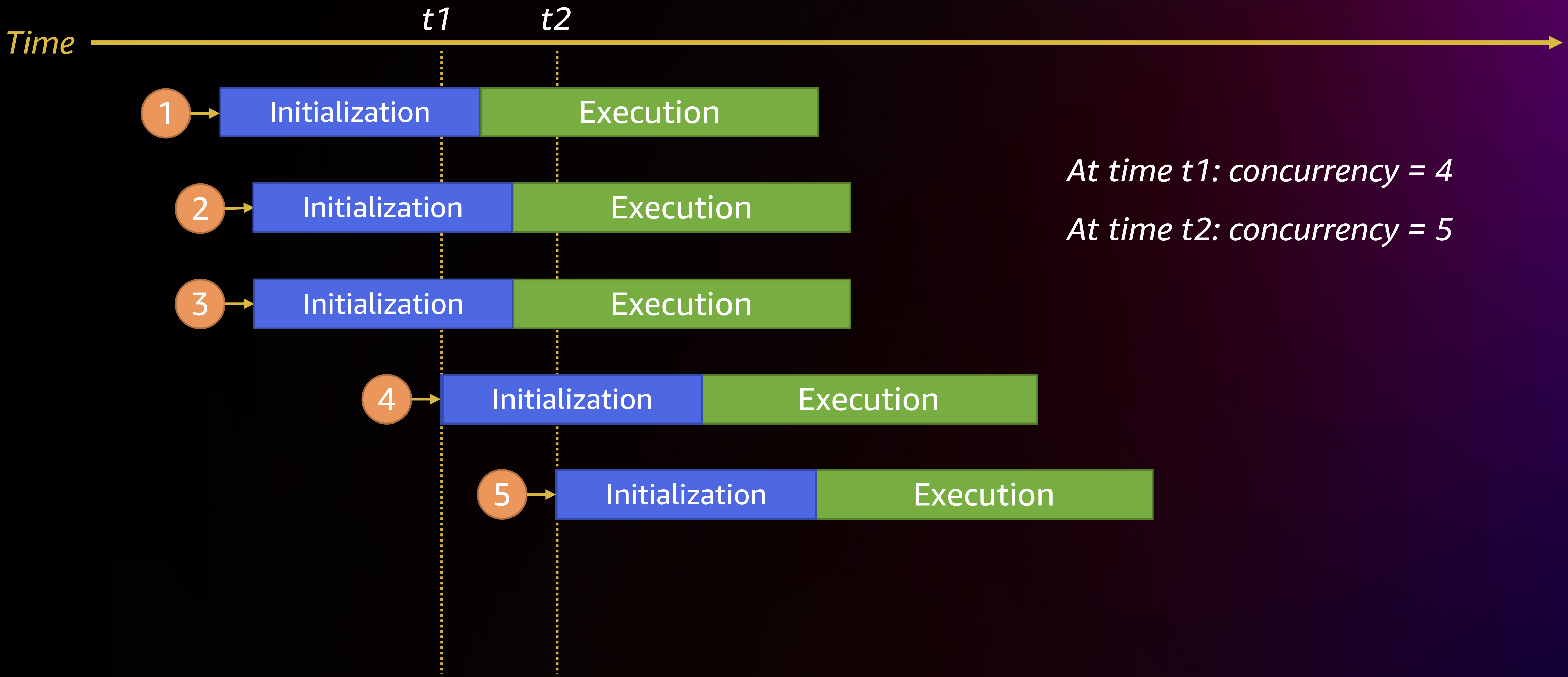
Lambda concurrency = requests per second (RPS) x duration in seconds

Examples

- 10,000 RPS x 1,000 ms = 10,000 concurrency
- 10,000 RPS x 500 ms = 5,000 concurrency
- 10,000 RPS x 100 ms = 1,000 concurrency

Long-running functions require more concurrency!

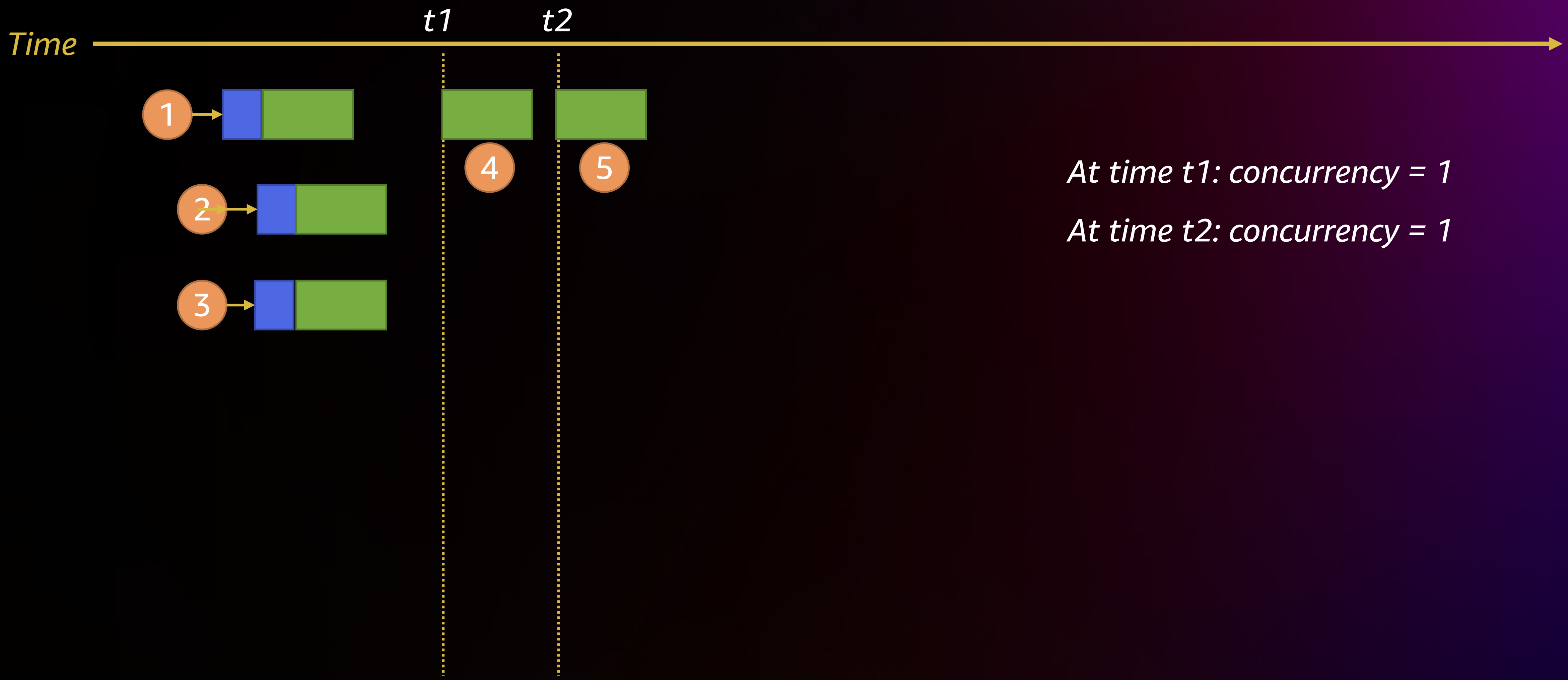
Decreasing concurrency



At time t1: concurrency = 4

At time t2: concurrency = 5

Decreasing concurrency

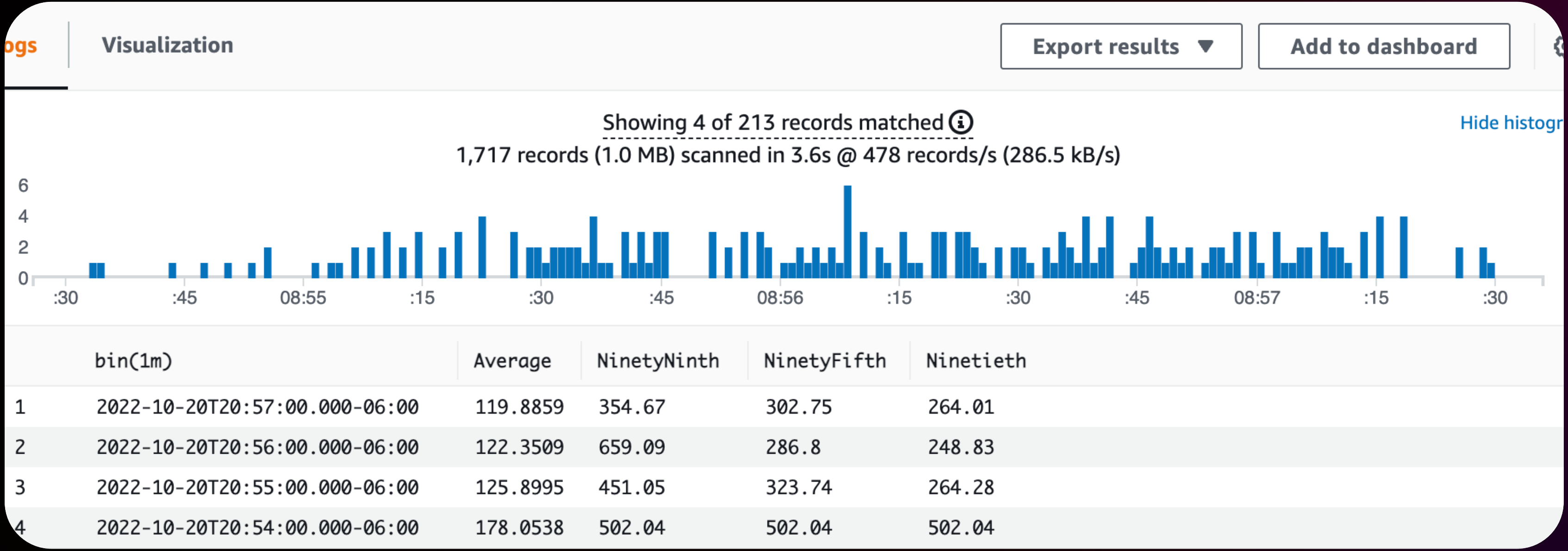


At time t1: concurrency = 1

At time t2: concurrency = 1

Measuring warm starts

```
filter @type = "REPORT" and @message not like /(?i)(Init Duration)/
| stats
  avg(@duration) as Average,
  pct(@duration, 99) as NinetyNinth,
  pct(@duration, 95) as NinetyFifth,
  pct(@duration, 90) as Ninetieth
by bin(1m)
```



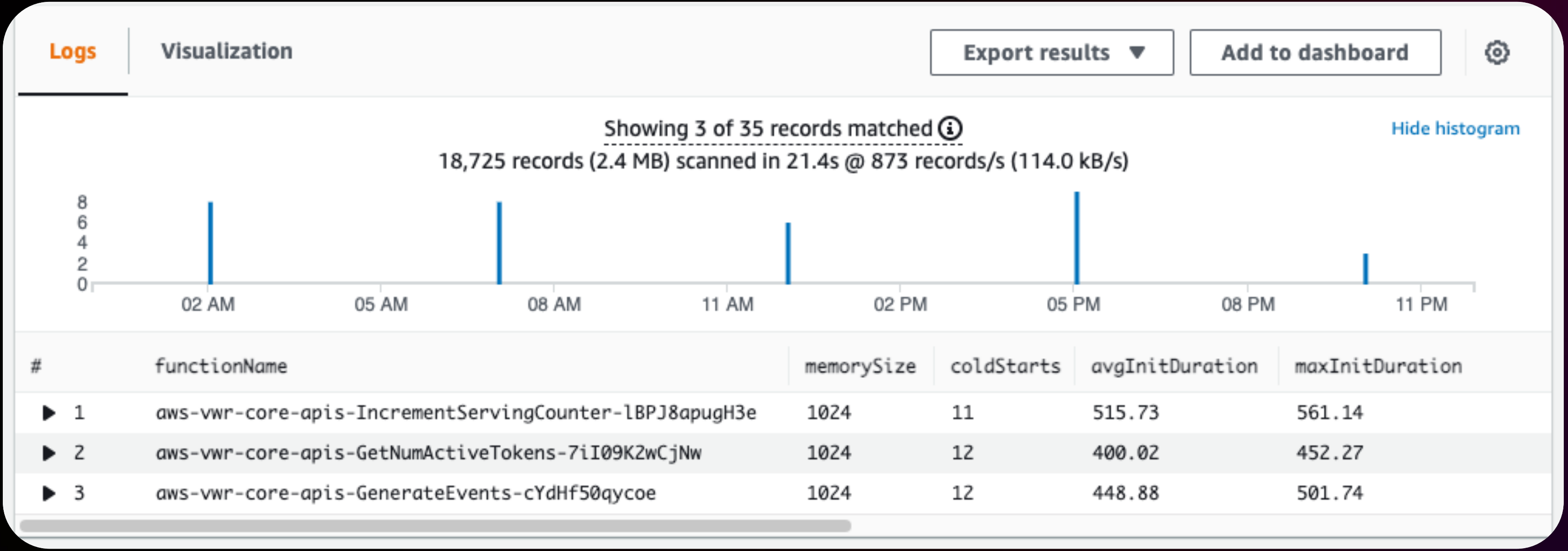
Showing 4 of 213 records matched
1,717 records (1.0 MB) scanned in 3.6s @ 478 records/s (286.5 kB/s)

	bin(1m)	Average	NinetyNinth	NinetyFifth	Ninetieth
1	2022-10-20T20:57:00.000-06:00	119.8859	354.67	302.75	264.01
2	2022-10-20T20:56:00.000-06:00	122.3509	659.09	286.8	248.83
3	2022-10-20T20:55:00.000-06:00	125.8995	451.05	323.74	264.28
4	2022-10-20T20:54:00.000-06:00	178.0538	502.04	502.04	502.04

Lambda optimizations: https://s12d.com/Lambda-Optimizations-2021

Measuring cold starts

```
filter @type="REPORT" and @message like /(?i)(Init Duration)/
| parse @message /^REPORT.*Init Duration: (?<initDuration>.*) ms.*/
| parse @log /^.*\/aws\/lambda\/(?<functionName>.*)/
| fields @memorySize / 1000000 as memorySize
| stats count() as coldStarts, avg(initDuration) as avgInitDuration, max(initDuration) as maxInitDuration
by functionName, memorySize
```



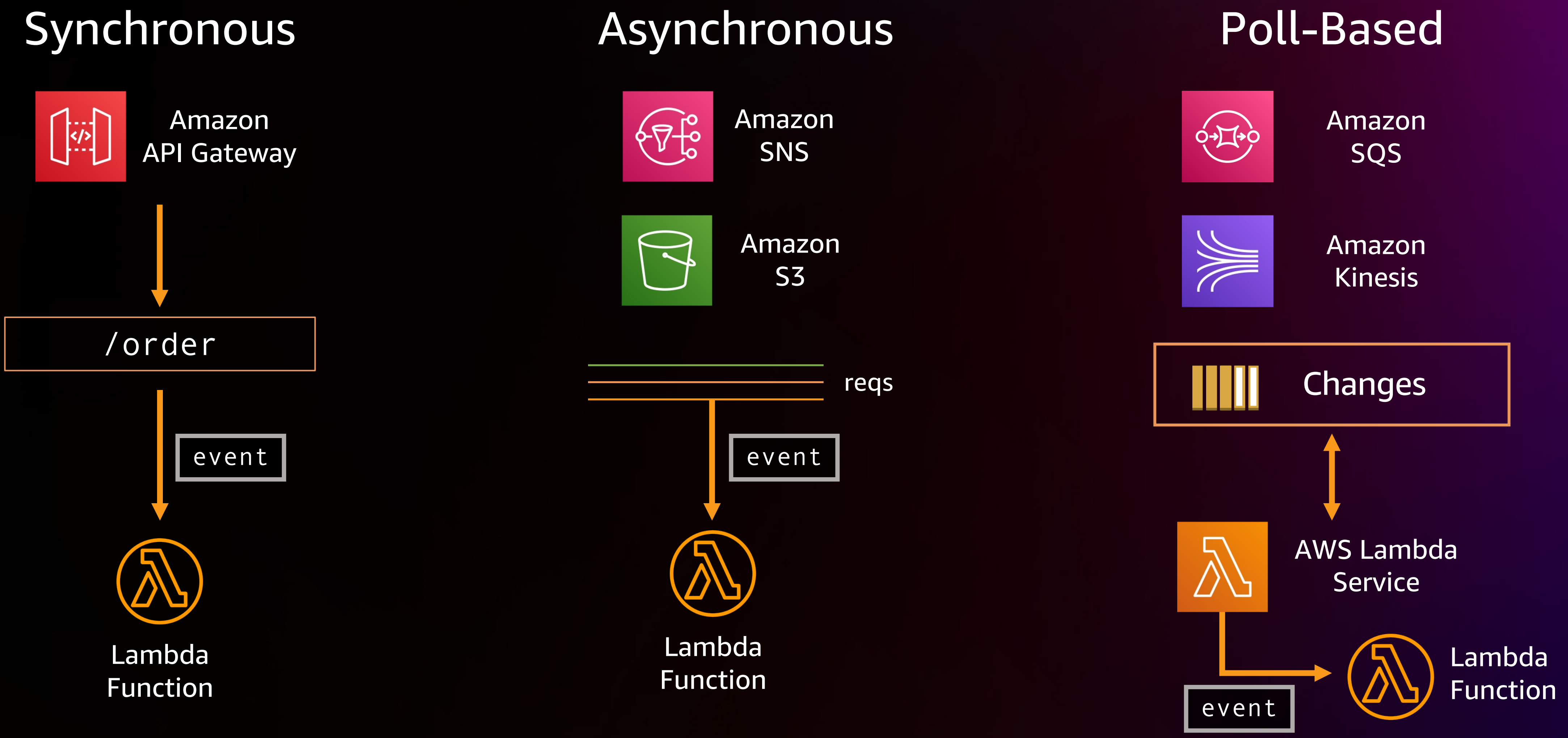
Showing 3 of 35 records matched
18,725 records (2.4 MB) scanned in 21.4s @ 873 records/s (114.0 kB/s)

#	functionName	memorySize	coldStarts	avgInitDuration	maxInitDuration
1	aws-vwr-core-apis-IncrementServingCounter-lBPJ8apugH3e	1024	11	515.73	561.14
2	aws-vwr-core-apis-GetNumActiveTokens-7iI09K2wCjNw	1024	12	400.02	452.27
3	aws-vwr-core-apis-GenerateEvents-cYdHf50qycoe	1024	12	448.88	501.74

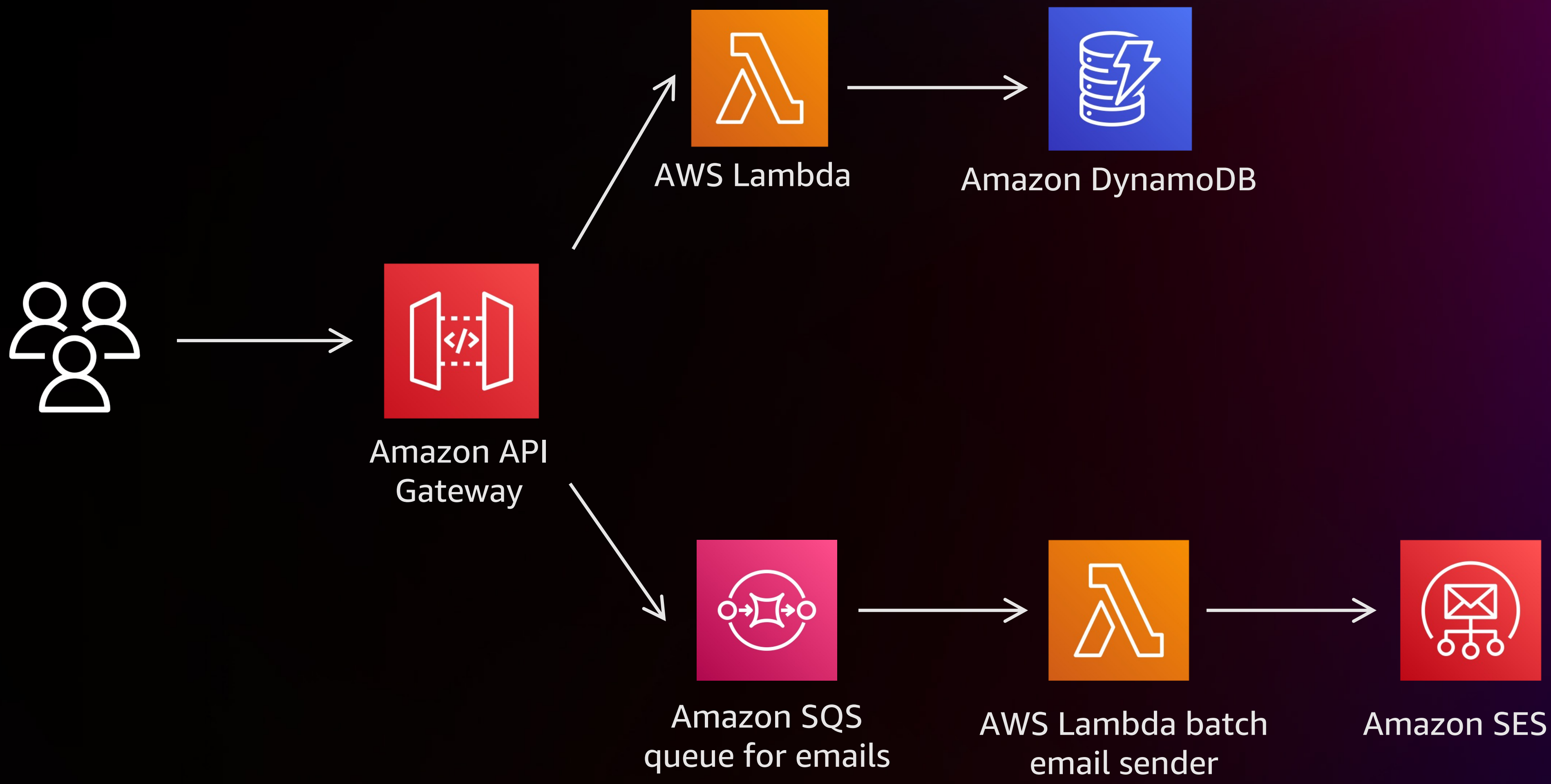
Lambda optimizations: https://s12d.com/Lambda-Optimizations-2021

Applied concurrency: Lambda invocation models

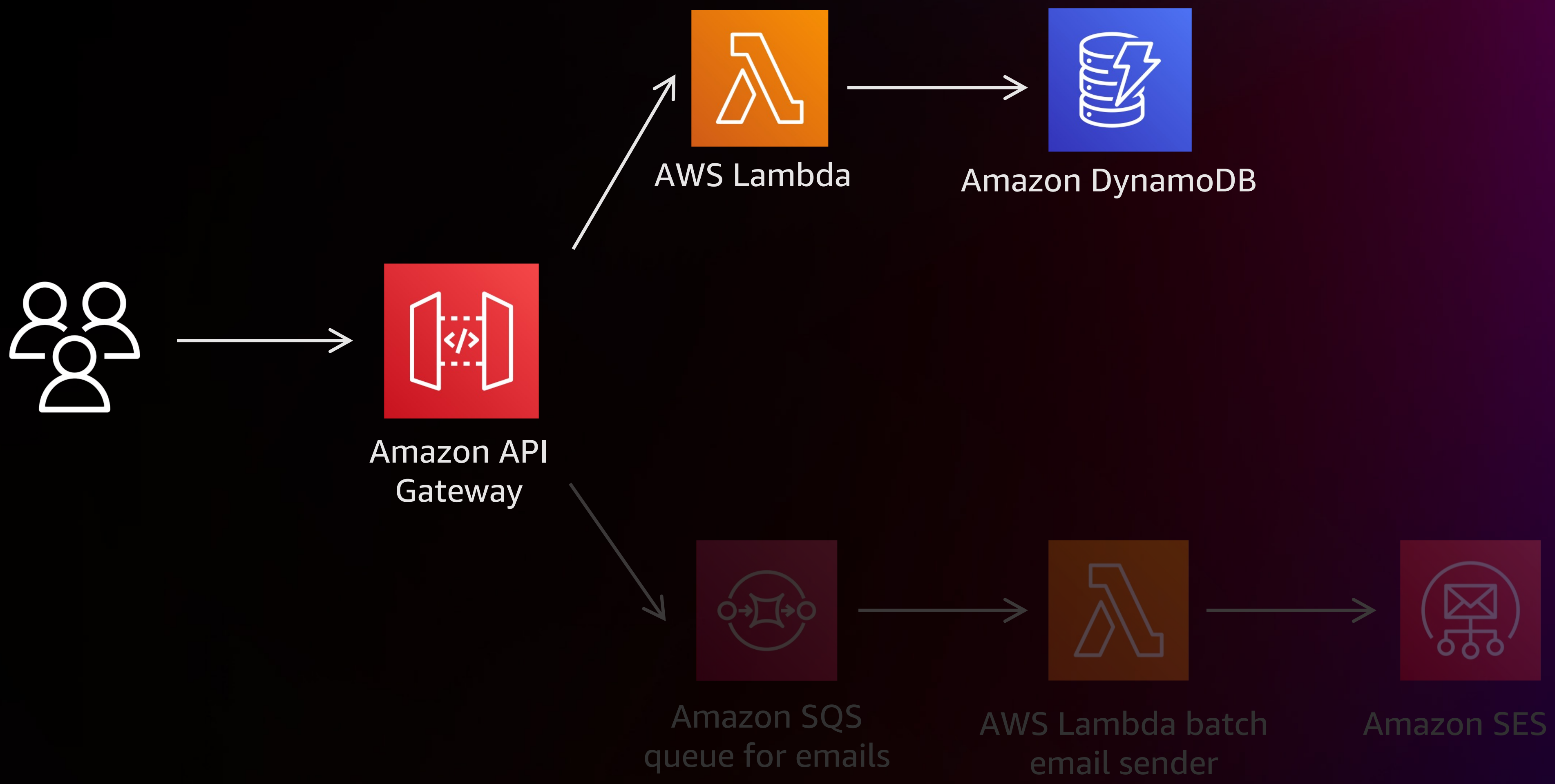
Lambda invocation methods



Today's application



Synchronous invocations



AWS Lambda function scaling quotas

Account concurrency quota

Maximum concurrency in a given Region
across all functions in an account

1,000 per Region by default*

This can be increased to customer needs.

AWS Lambda function scaling quotas

Burst concurrency quota

Maximum increase in concurrency for an initial burst of traffic

3000 in Oregon, N. Virginia and Ireland

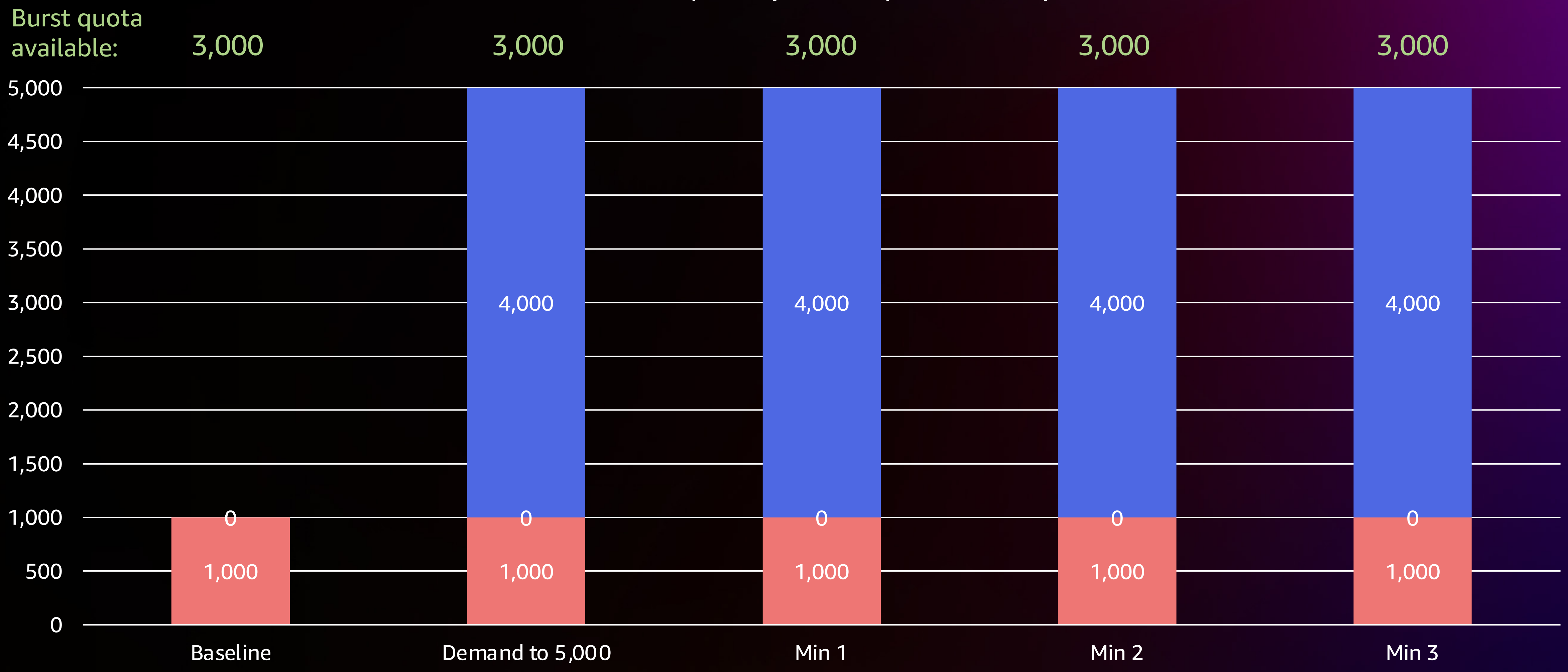
1000 in Tokyo, Frankfurt and Ohio

500 in all other regions

After the initial burst, your function concurrency can scale by an additional **500 instances each minute**

Lambda bursting and ramp up

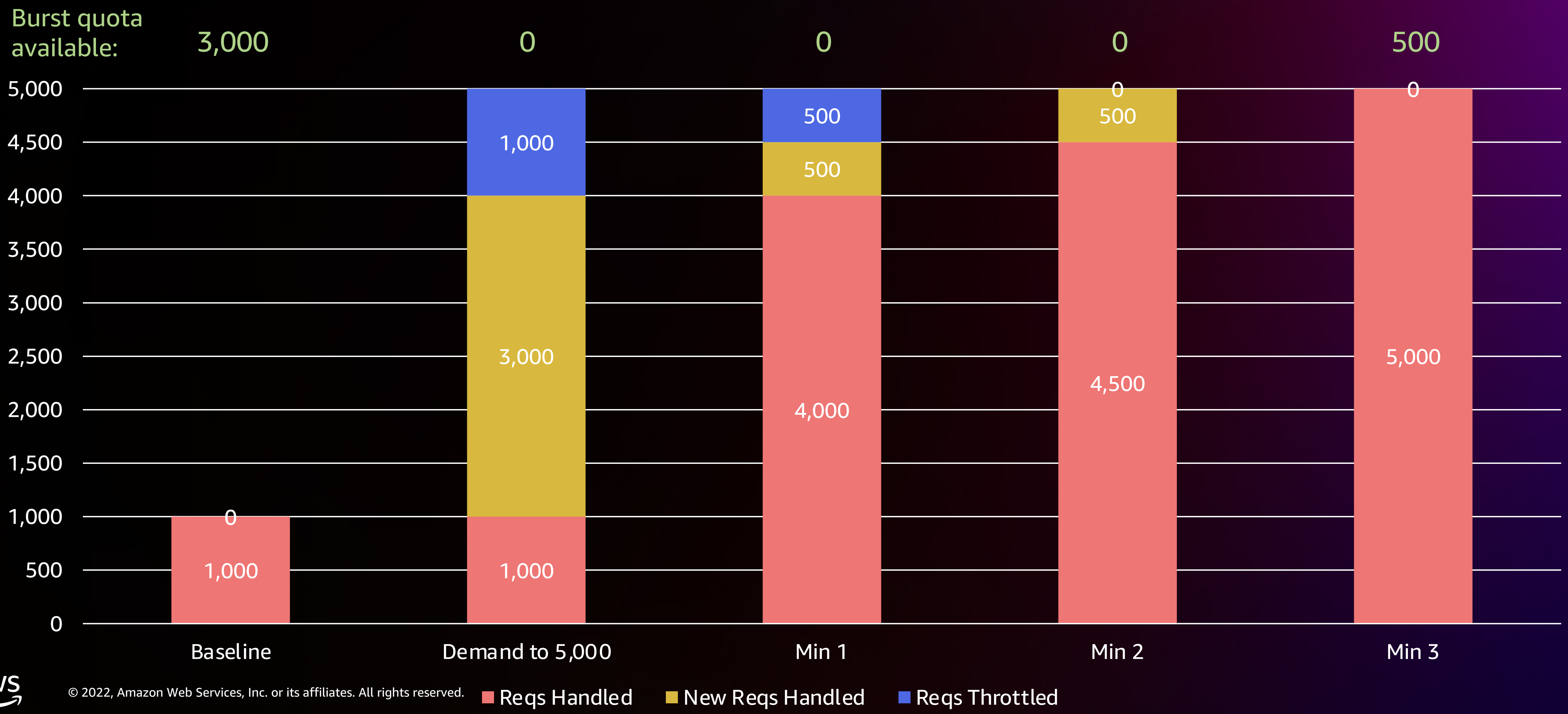
Concurrency limit = 1,000 (default, us-east-1)
Burst limit = 3,000 (default, us-east-1)



🟥 Reqs Handled 🟨 New Reqs Handled 🟦 Reqs Throttled

Lambda bursting and ramp up

Concurrency limit = 5,000 (increased, us-east-1)
Burst limit = 3,000 (default, us-east-1)



	Baseline	Demand to 5,000	Min 1	Min 2	Min 3
Burst quota available	3,000	0	0	0	500
🟥 Reqs Handled	1,000	1,000	4,000	4,500	5,000
🟨 New Reqs Handled	0	3,000	500	500	0
🟦 Reqs Throttled		1,000	500	0	

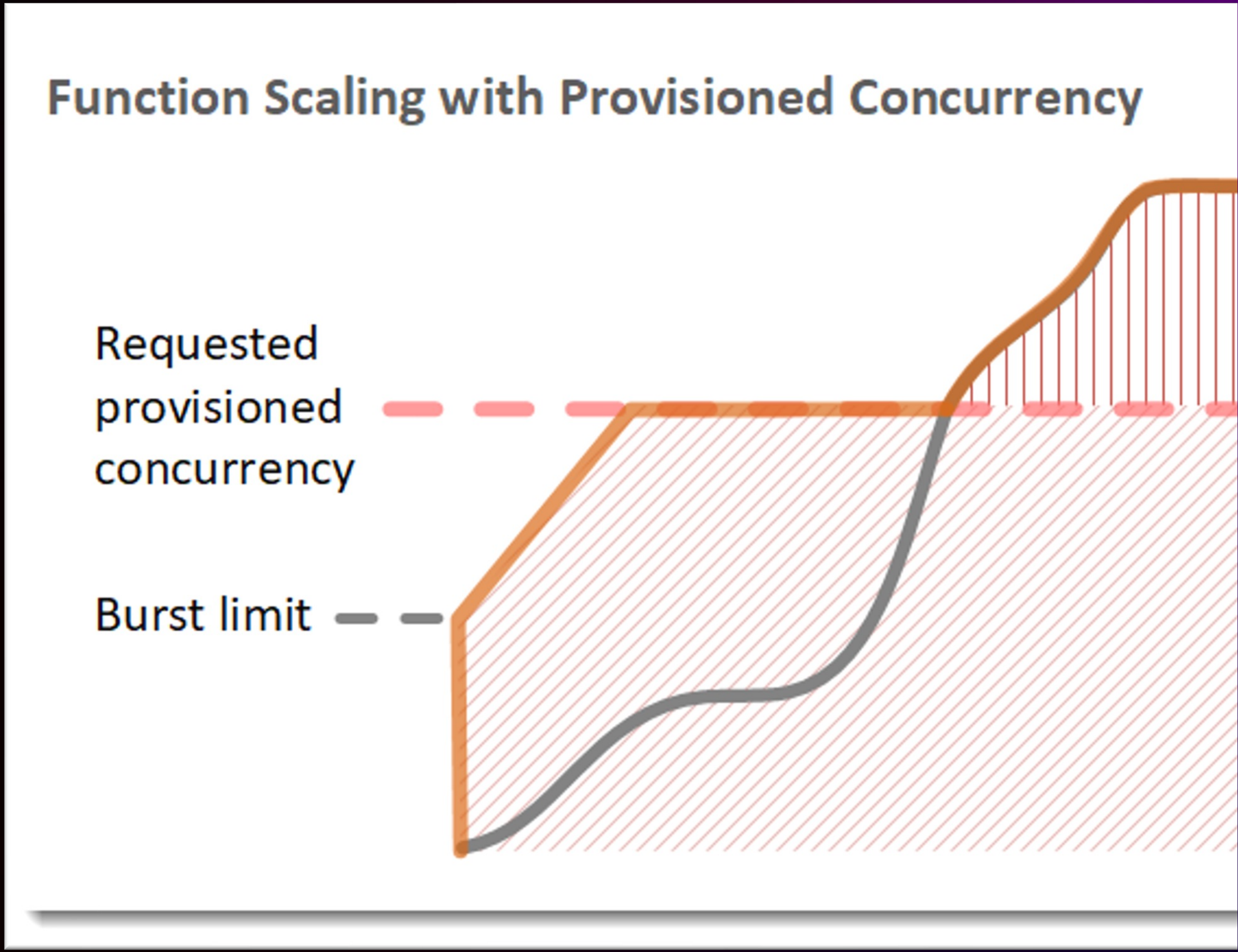
aws

What if you need more burst than the burst quota allows?

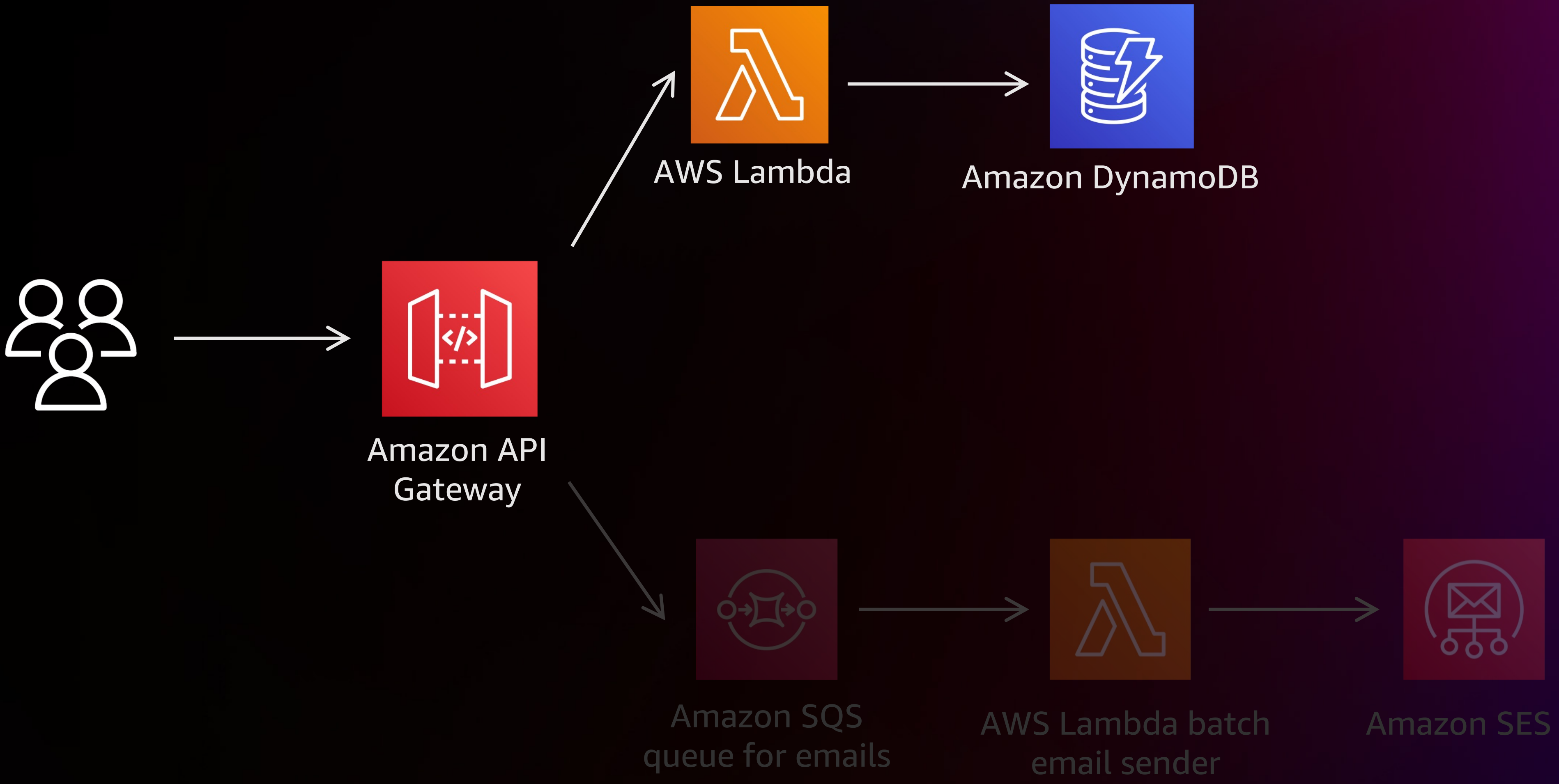
Lambda concurrency controls

Provisioned concurrency:

- Sets floor on minimum number of execution environments
- Pre-warm execution environments to reduce cold-start impact
- Burst to use standard concurrency if desired
- Can save costs in certain situations



Synchronous invocations

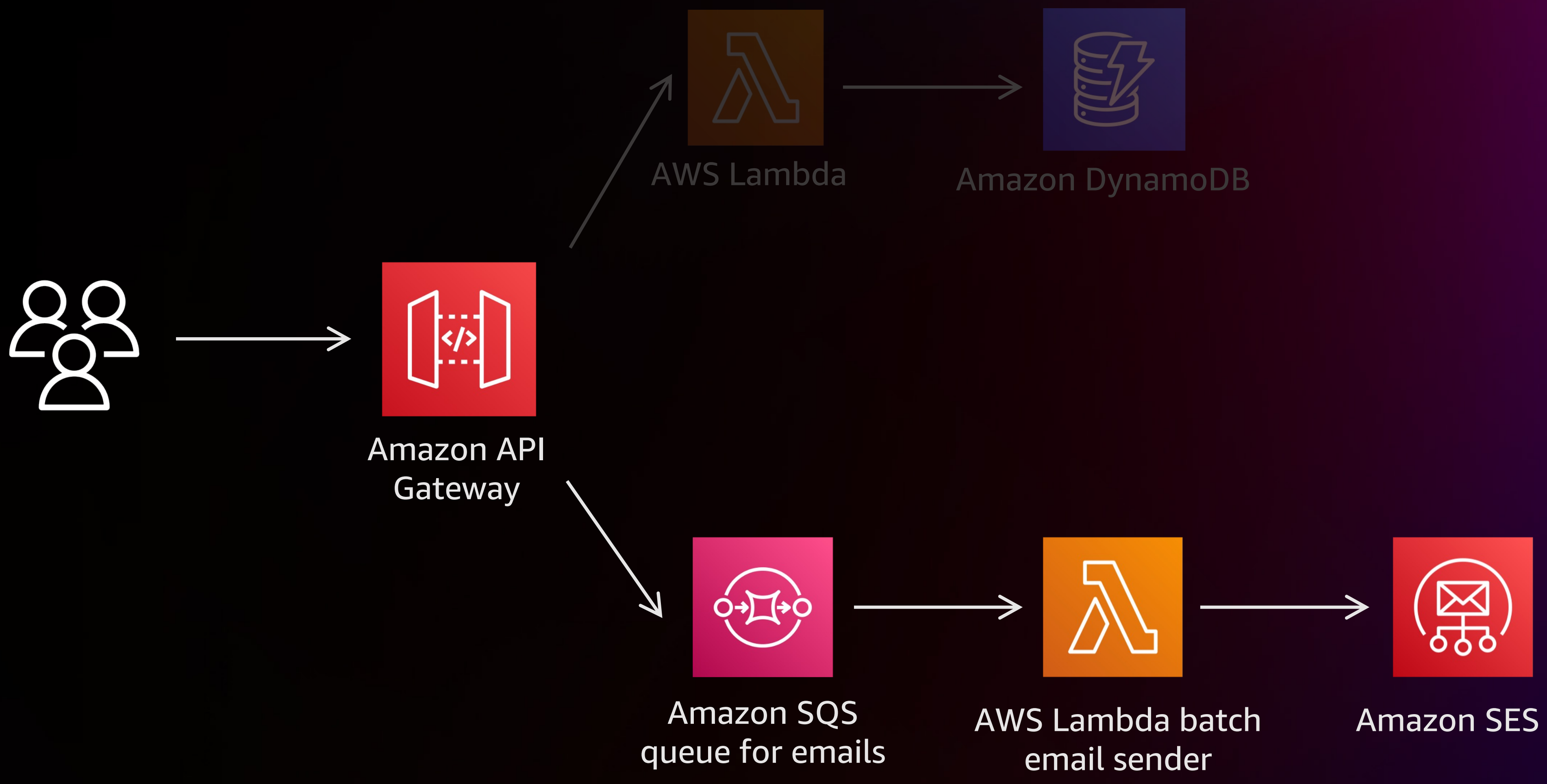


Synchronous invocations: best practices

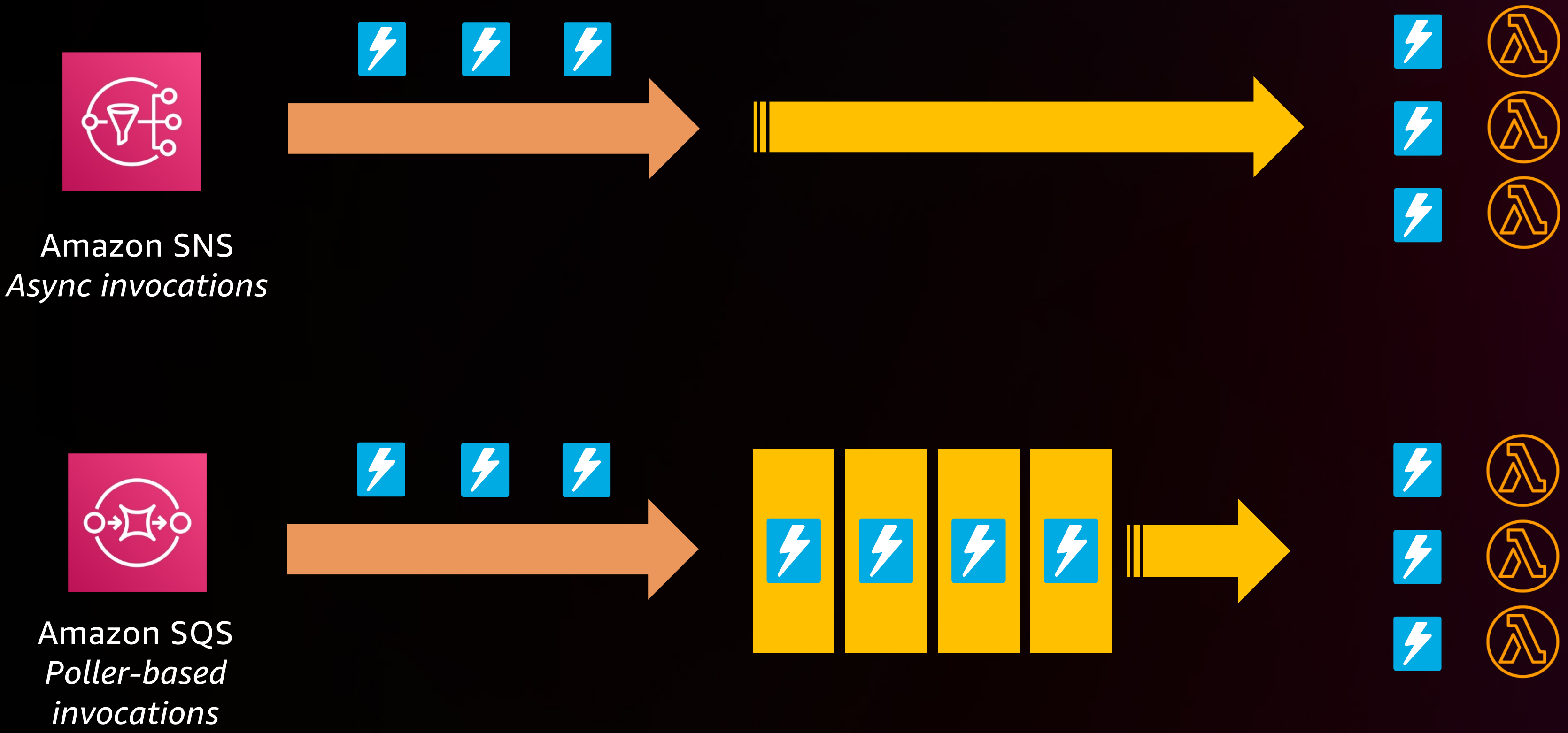
- **Load test**
  - Raise account-level concurrency quota in advance of bursts of demand
- **Set appropriate timeouts**
  - Set reasonable timeouts per Lambda function – do not set to 15 minutes
  - Configure API response timeout – default is 29 seconds
- **Retry with backoff**
  - Clients should retry, but with exponential backoff to not stampede the system
- **Implement idempotency**
  - Expect and account for retried operations
  - Application awareness and handling of scenarios is key

Dive deeper at https://s12d.com/reInvent2020-Error-Handling

Asynchronous invocations



Lambda function concurrency across models



No event store

"Fire and forget"

1 Lambda invocation per event

Message delivery and processing durability

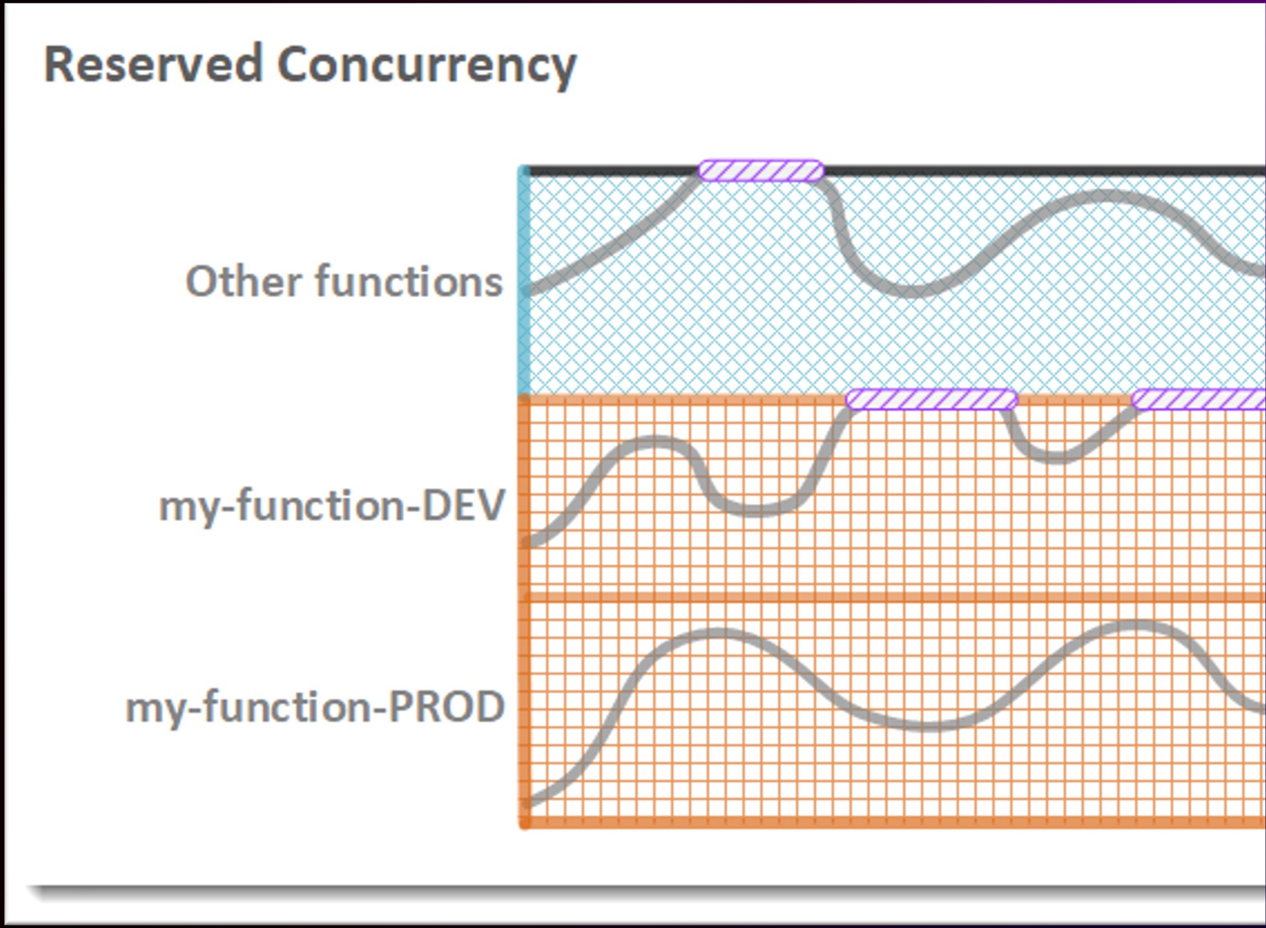
1 Lambda invocation per batch

What if scaling too fast may overwhelm downstream systems?

Lambda concurrency controls

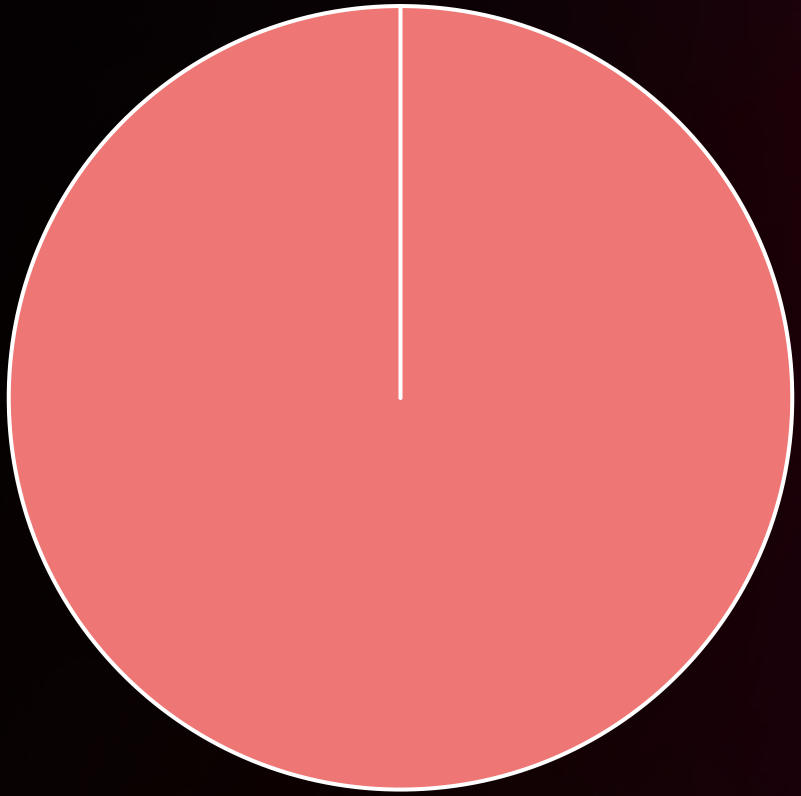
Reserved concurrency

- Sets ceiling on maximum number of execution environments – upper limit on maximum concurrency for a given function
- Also reserves that concurrency from the account's quota



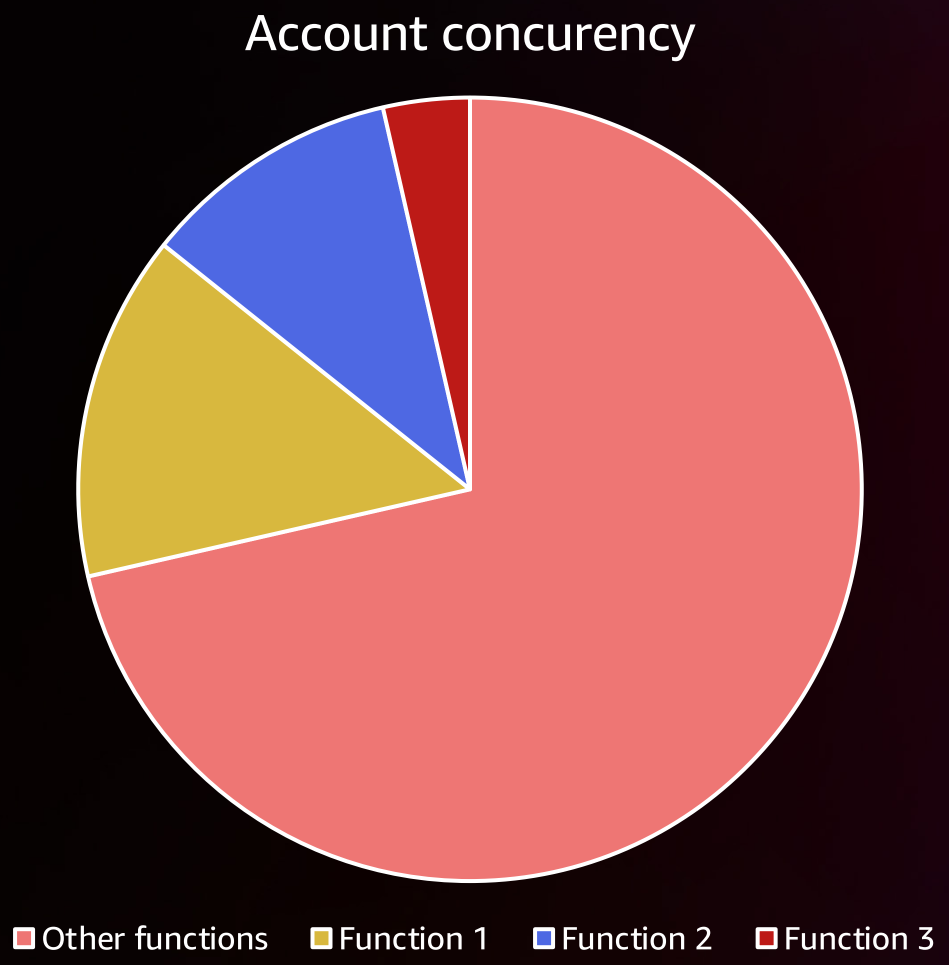
Reserved concurrency

Account concurency



🟥 All functions

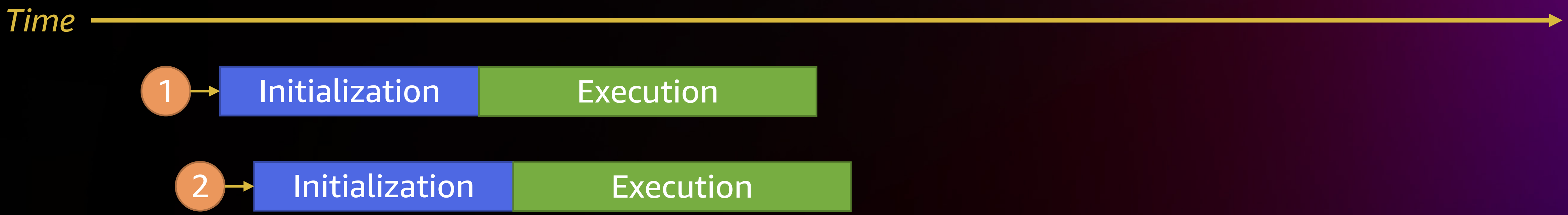
Reserved concurrency



Reserved concurrency is an upper limit for a Lambda function's concurrency.

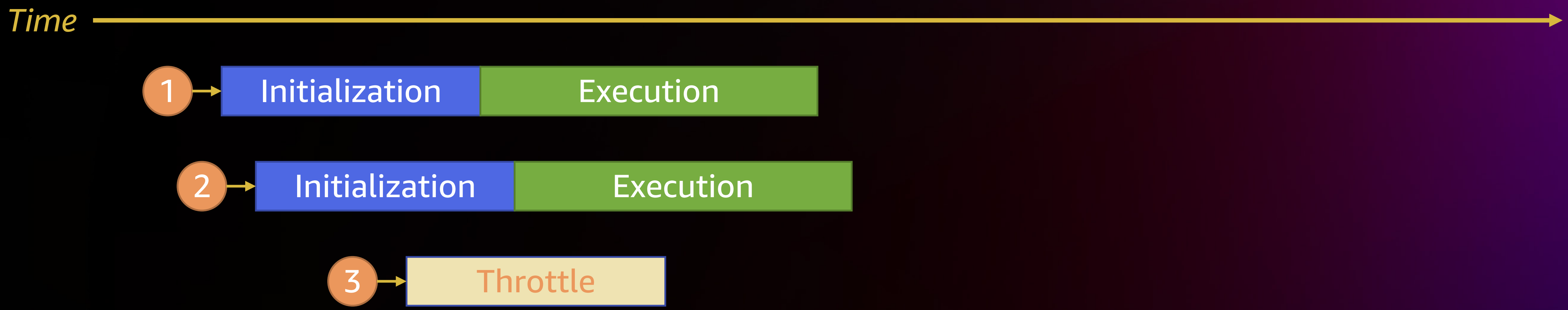
Reserved concurrency is always respected, regardless of integration.

How Lambda scales: Reserved Concurrency



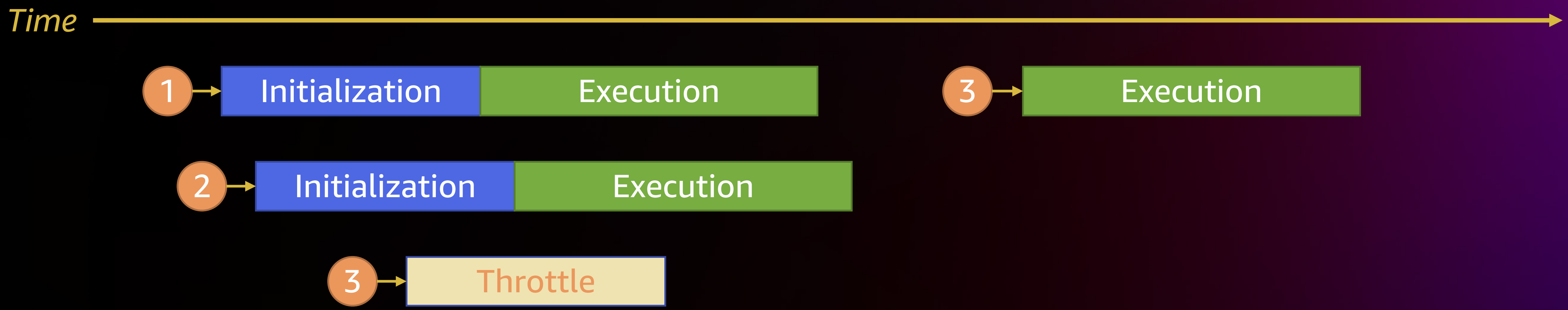
Reserved Concurrency = 2

How Lambda scales: Reserved Concurrency



Reserved Concurrency = 2

How Lambda scales: Reserved Concurrency



Reserved Concurrency = 2

Asynchronous invocations: best practices

- Invocation model
  - Leverage async invocations whenever possible
- Function configuration
  - Leverage batching for cost reduction and performance gains
  - Catch errors/failures and return altogether if using a batch size greater than 1
  - Configure Lambda on-failure destination to save and review failed messages
  - Monitor function error metrics to ensure they remain at or close to 0
- SQS queue configuration
  - Set the queue visibility timeout to at least 6x function timeout
  - Setup a DLQ with maxReceiveCount of at least 5

SQS Lambda Best Practices: https://s12d.com/SQS-Lambda-Best-Practices

Concurrency tips

- Understand fundamentals
- Plan ahead and load test
- Monitor, monitor, monitor
  - `ConcurrentExecutions`
  - `Errors`
  - `Throttles`
  - SQS queue depth, Kinesis iterator age, etc.

Open Discussion

Check out these other sessions…

SVS404: A closer look at AWS Lambda

Wednesday (Nov 30) @ 10:00 AM

SVS401: Best practices for advanced serverless developers

Tuesday (Nov 29) @ 12:30 PM

SVS314: AWS Lambda performance tuning: Best practices and guidance

Thursday (Dec 1) @ 4:15 PM

References

- Documentation
  - Invoke Lambda functions
    https://s12d.com/lambda-invocation
  - Using Lambda with Amazon SQS
    https://s12d.com/lambda-sqs
- Tools and guidance
  - AWS Lambda Powertools for Python
    https://s12d.com/powertools
  - Serverless Applications Lens - AWS Well-Architected Framework
    https://s12d.com/serverless-wa-lens

Continue your AWS Serverless learning

Learn at your own pace

Expand your Serverless skills with our Learning Plan on **AWS Skill Builder**

Increase your knowledge

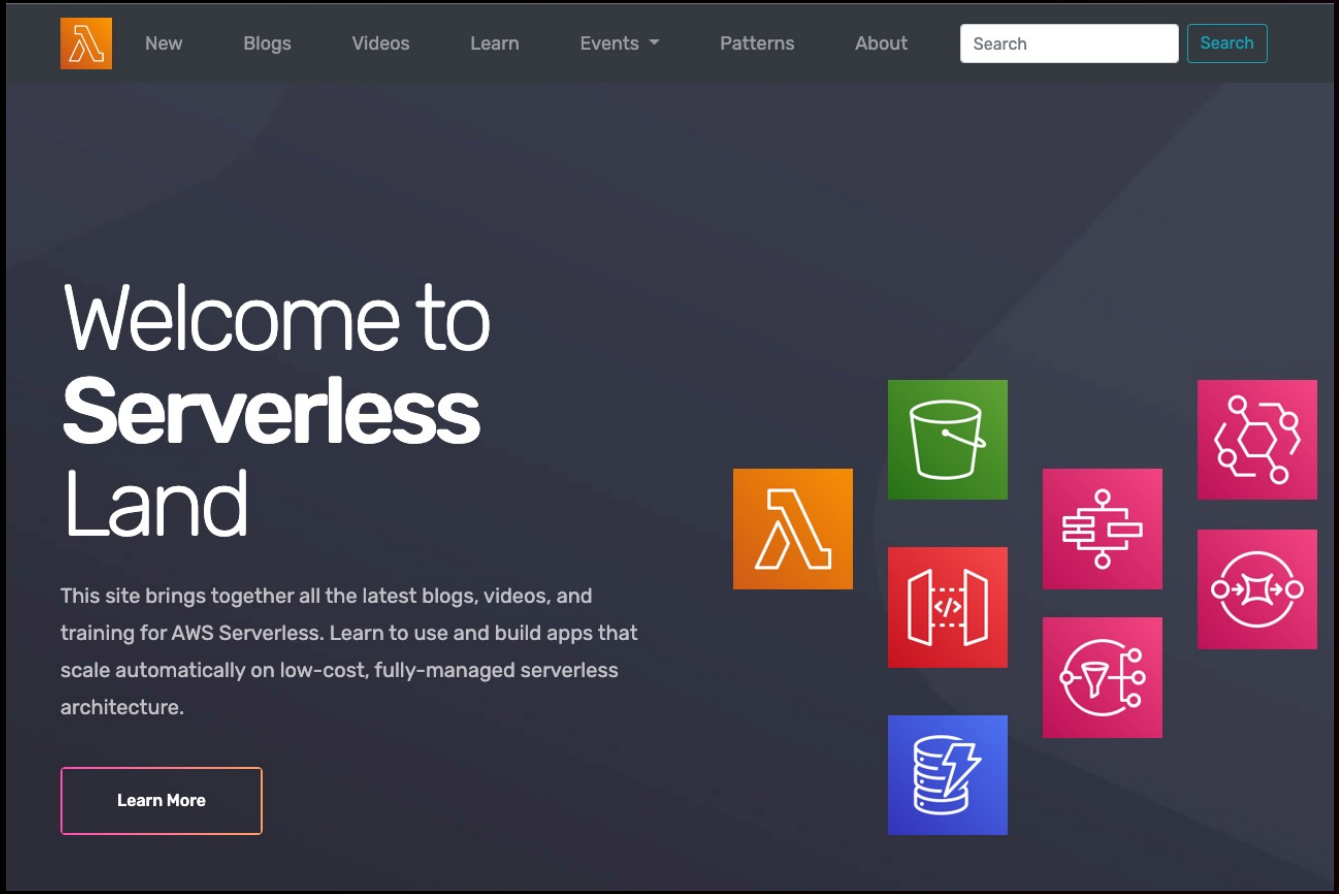
Use our **Ramp-Up Guides** to build your Serverless knowledge

Earn AWS Serverless badge

Demonstrate your Knowledge by achieving **digital badges**

https://s12d.com/serverless-learning

Serverless Land for more resources


New Blogs Videos Learn Events Patterns About Search Search

Welcome to **Serverless** Land

This site brings together all the latest blogs, videos, and training for AWS Serverless. Learn to use and build apps that scale automatically on low-cost, fully-managed serverless architecture.

Session slides and links: https://s12d.com/reInvent2022-Lambda-Concurrency

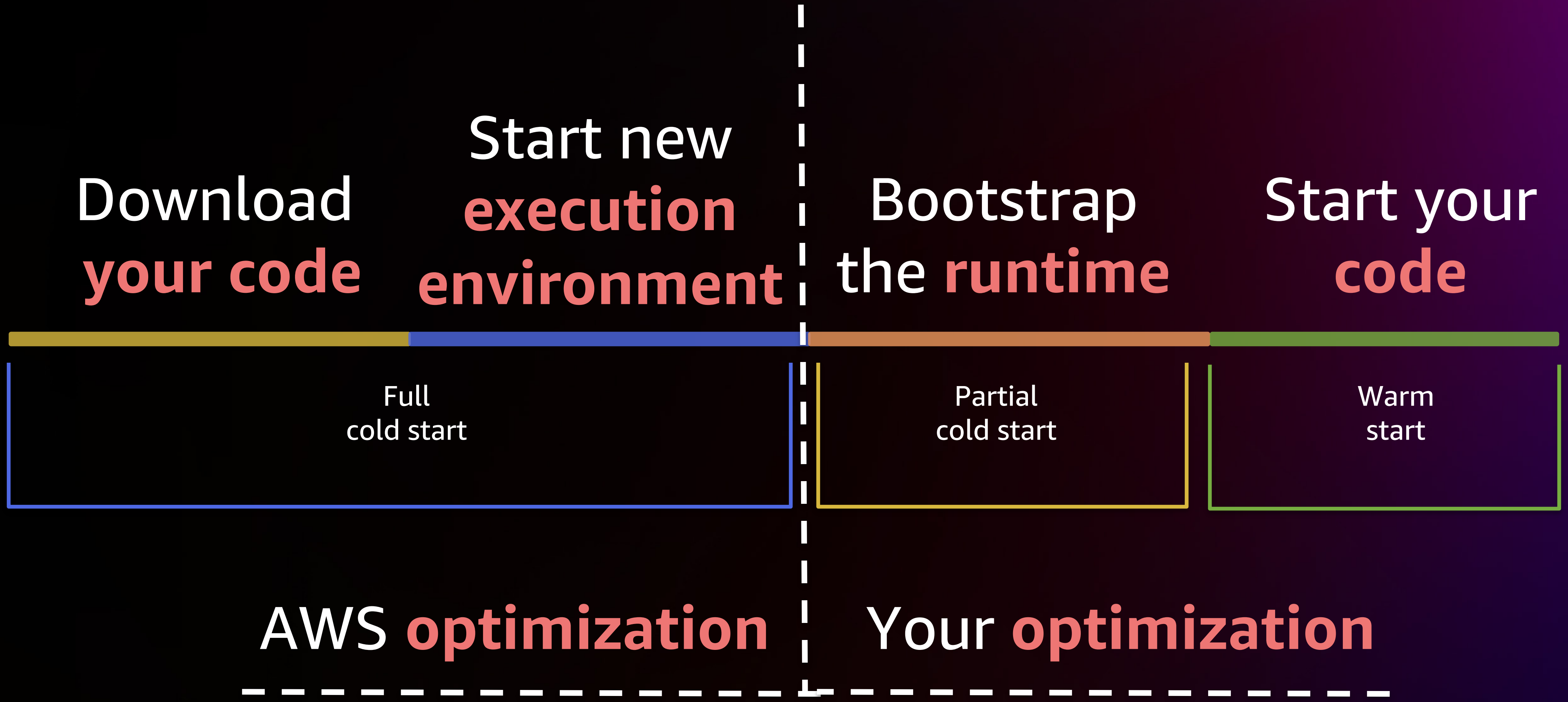
Please complete the session survey

Thank you!

Brian Zambrano (he/him)
@brianzambrano

Justin Pirtle (he/him)
@justinpirtle

The function lifecycle

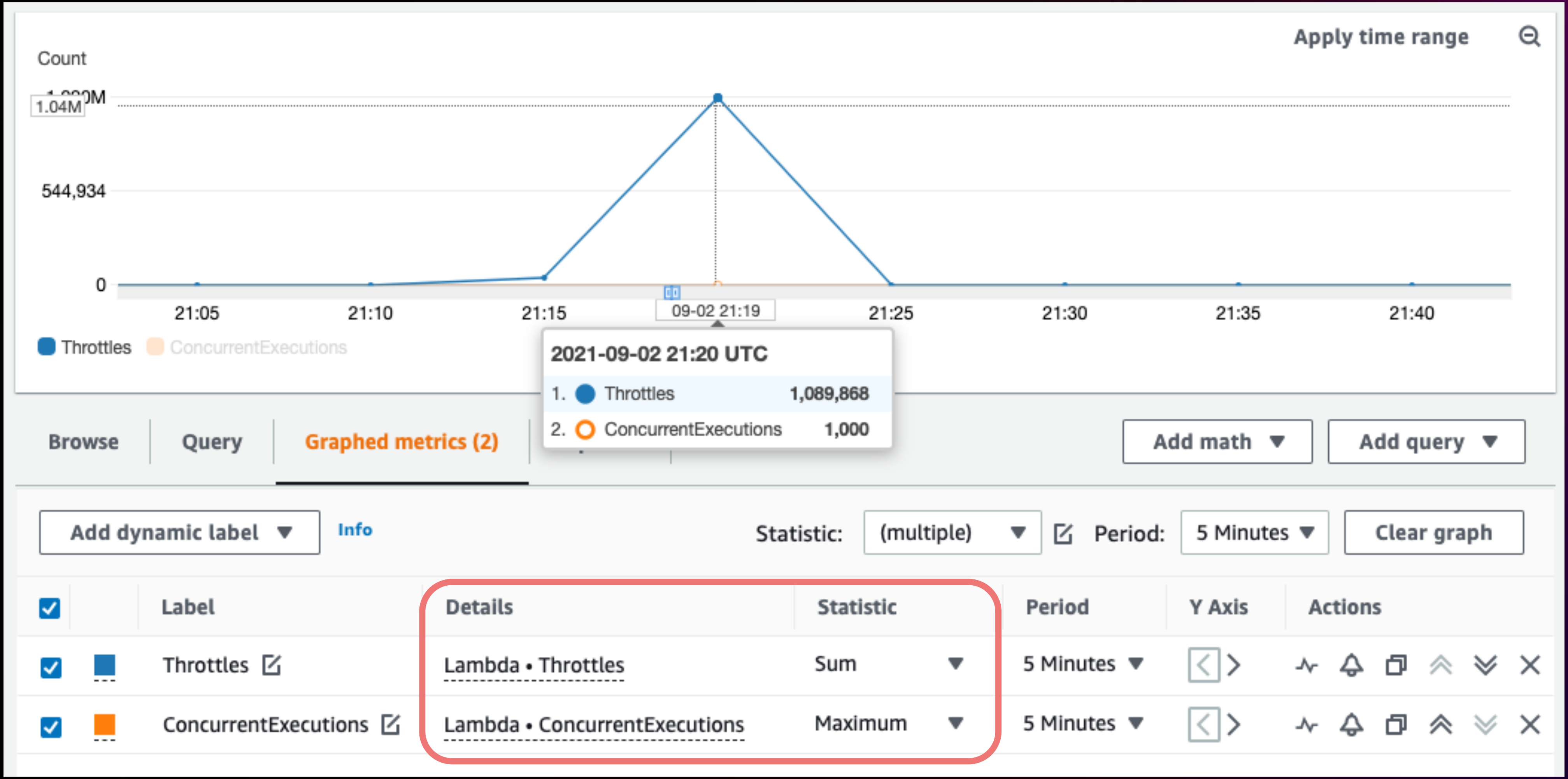


CloudWatch metrics: Lambda

What to monitor during testing and in production:

- *Errors* - Number of failed function invocations due to timeout, code exceptions, or service issues
- *Throttles* - Number of throttled function invocations caused by account/burst concurrency limit or intentional reserved concurrency ceiling
- *ConcurrentExecutions* – Count of function execution environments simultaneously processing events. View as Maximum for give time window
- *ProvisionedConcurrentExecutions* – Count of function execution environments simultaneously processing events on provisioned concurrently

Lambda concurrency quota reached



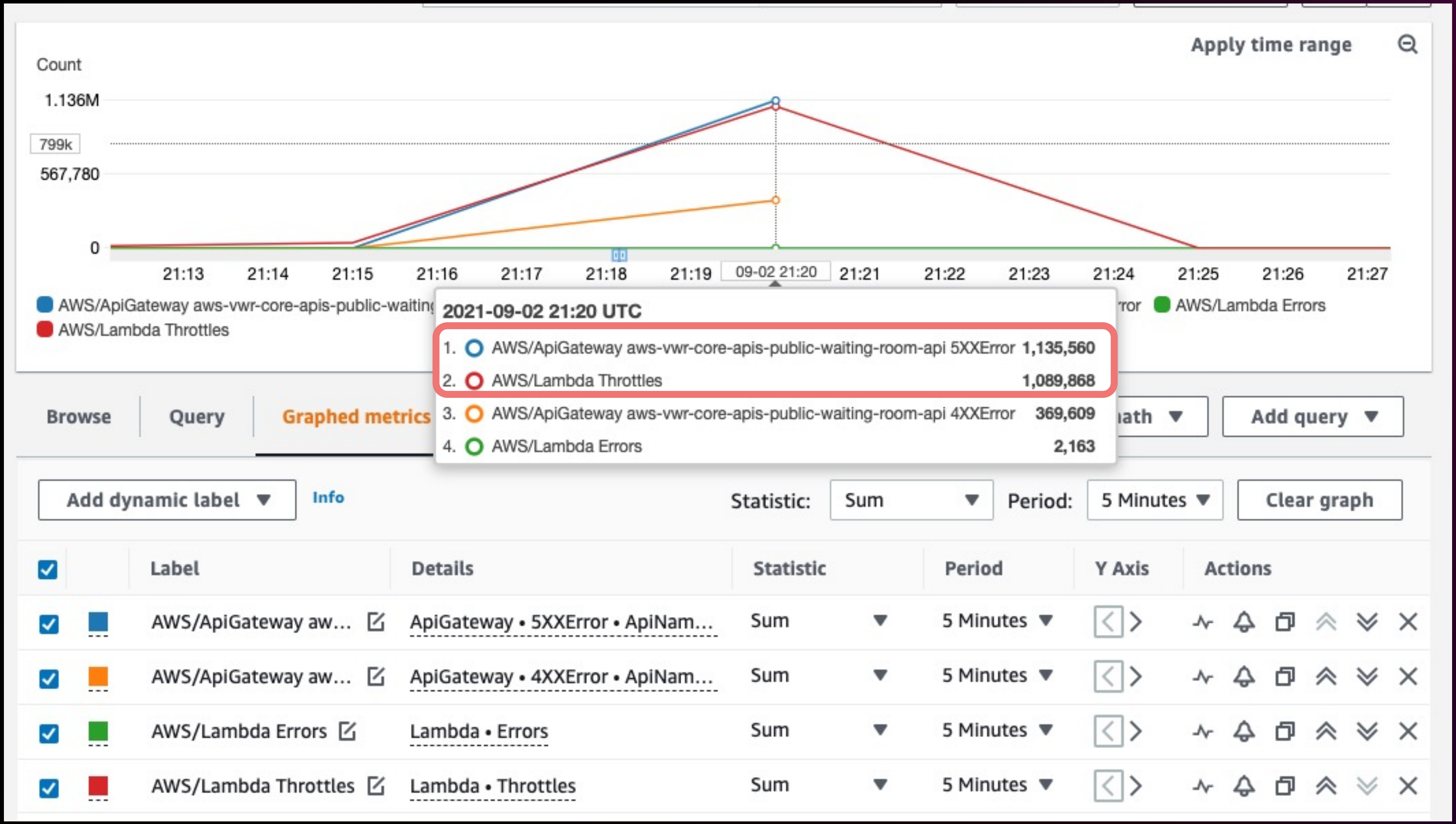
2021-09-02 21:20 UTC

1. ●	Throttles	1,089,868
2. ○	ConcurrentExecutions	1,000

Statistic: (multiple) Period: 5 Minutes

✓		Label	Details	Statistic	Period	Y Axis	Actions
✓	🟦	Throttles	Lambda • Throttles	Sum	5 Minutes		
✓	🟧	ConcurrentExecutions	Lambda • ConcurrentExecutions	Maximum	5 Minutes		

CloudWatch metrics: Lambda errors/throttles



2021-09-02 21:20 UTC

1.	AWS/ApiGateway aws-vwr-core-apis-public-waiting-room-api 5XXError	1,135,560
2.	AWS/Lambda Throttles	1,089,868
3.	AWS/ApiGateway aws-vwr-core-apis-public-waiting-room-api 4XXError	369,609
4.	AWS/Lambda Errors	2,163

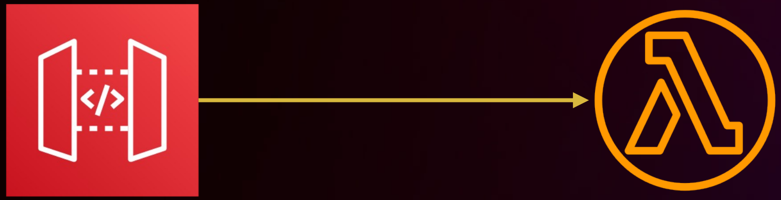
Statistic: Sum Period: 5 Minutes

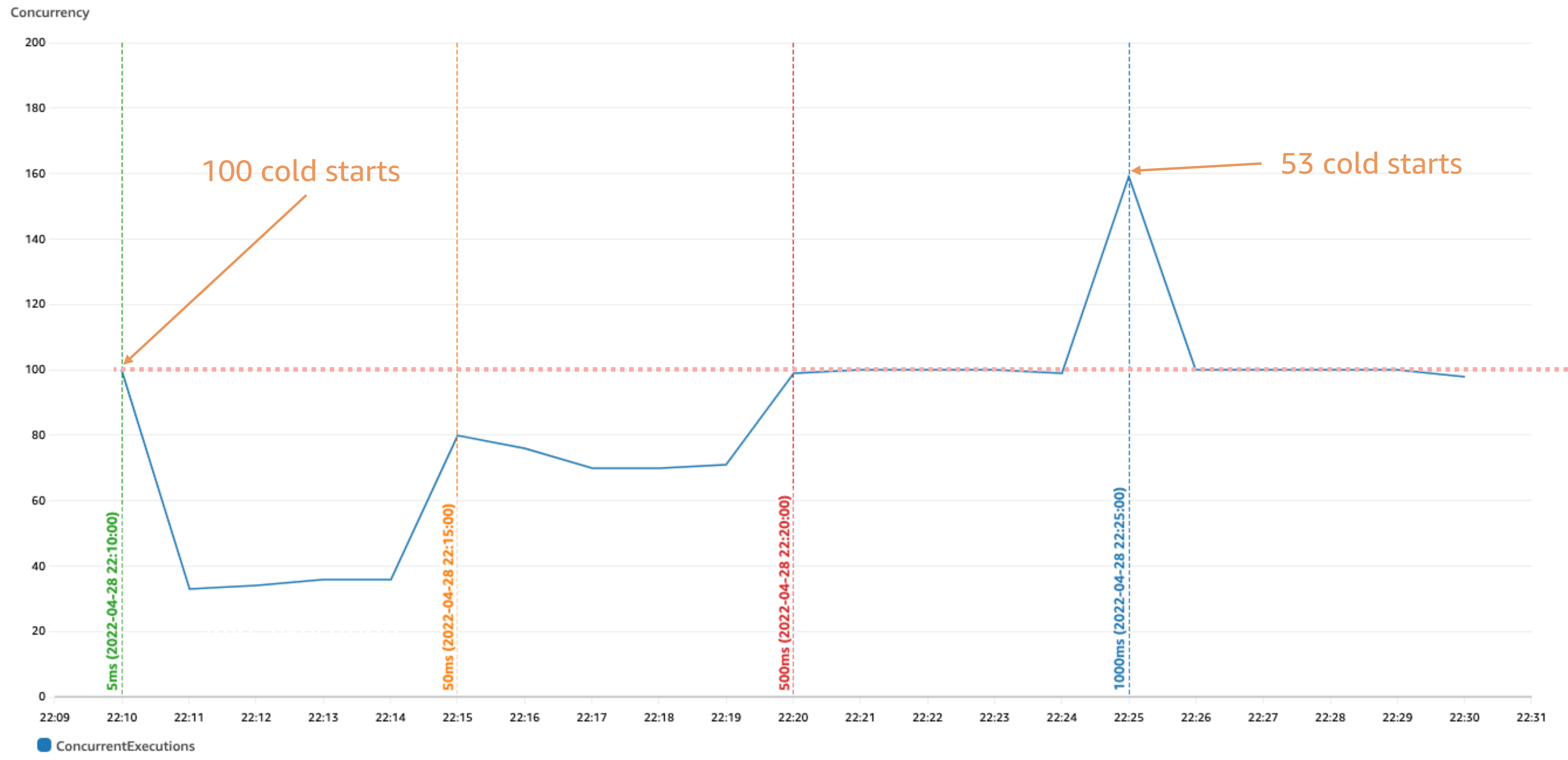
✓		Label	Details	Statistic	Period	Y Axis	Actions
✓	🟦	AWS/ApiGateway aw…	ApiGateway • 5XXError • ApiNam…	Sum	5 Minutes		
✓	🟧	AWS/ApiGateway aw…	ApiGateway • 4XXError • ApiNam…	Sum	5 Minutes		
✓	🟩	AWS/Lambda Errors	Lambda • Errors	Sum	5 Minutes		
✓	🟥	AWS/Lambda Throttles	Lambda • Throttles	Sum	5 Minutes		

Lambda 429 TooManyRequests throttles surfaced as:
- 500 status code response from REST API Gateway deployment
- 503 status code response from HTTP API Gateway deployment

Test: Effect of execution duration on concurrency

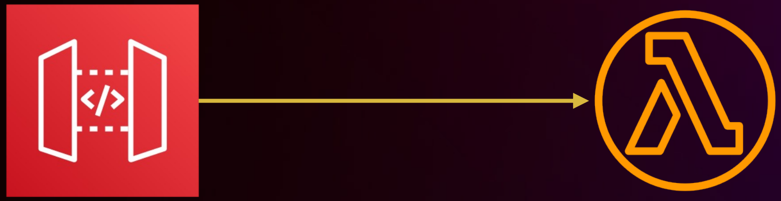
- Send 100 RPS to APIGW
- No delay between requests
- Change execution duration every 5 mins via query param
  - 5ms
  - 50ms
  - 500ms
  - 1000ms

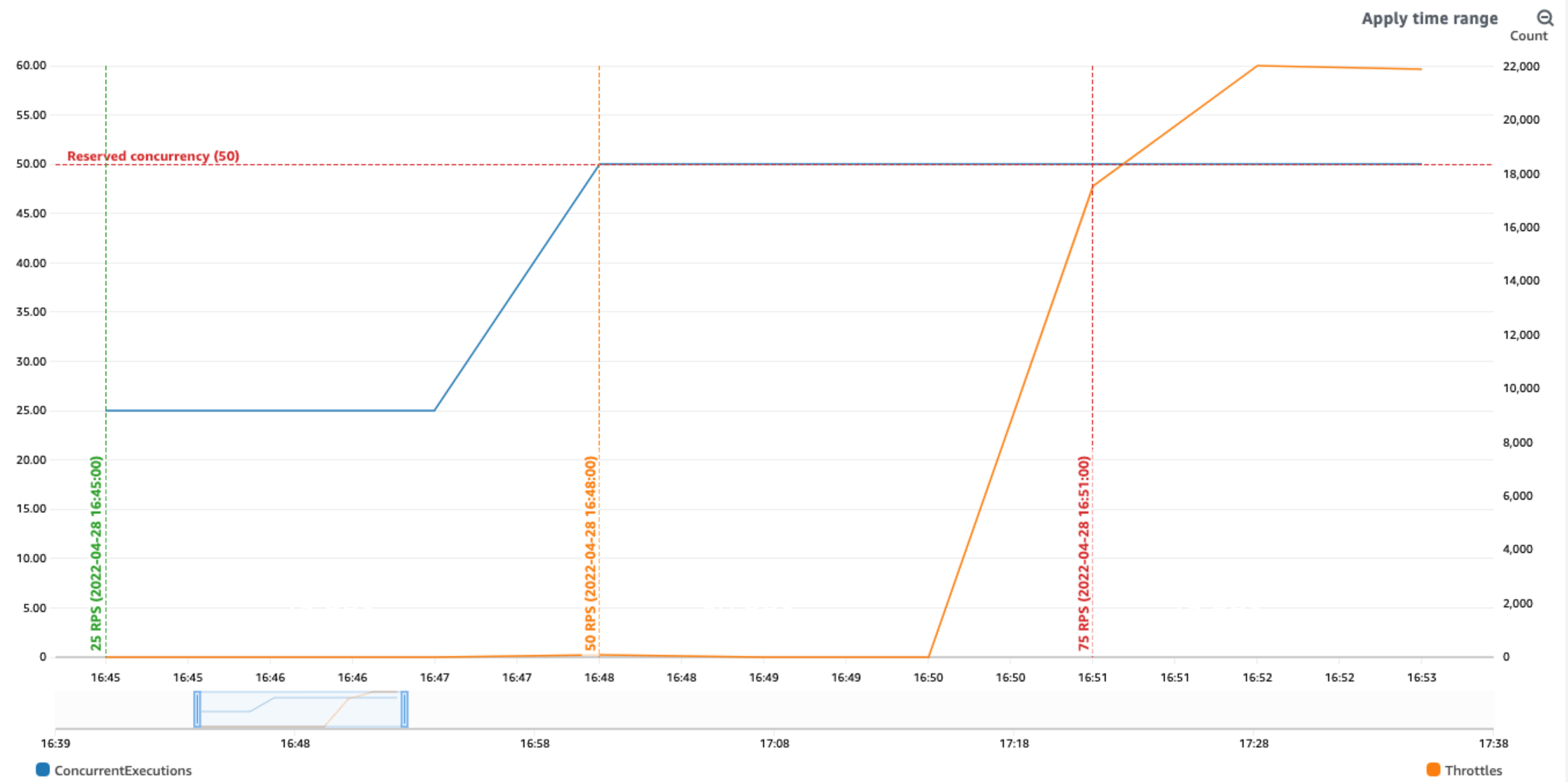



Concurrency
200
180
160
140
120
100
80
60
40
20
0
100 cold starts
53 cold starts
5ms (2022-04-28 22:10:00)
50ms (2022-04-28 22:15:00)
500ms (2022-04-28 22:20:00)
1000ms (2022-04-28 22:25:00)
22:09
22:10
22:11
22:12
22:13
22:14
22:15
22:16
22:17
22:18
22:19
22:20
22:21
22:22
22:23
22:24
22:25
22:26
22:27
22:28
22:29
22:30
22:31
ConcurrentExecutions

Test: Effect of RC with various RPS

- Set reserved concurrency to 50
- Average request duration 1000ms (1s)
- Send various TPS to APIGW
  - 25 RPS
  - 50 RPS
  - 75 RPS
- No delay between requests

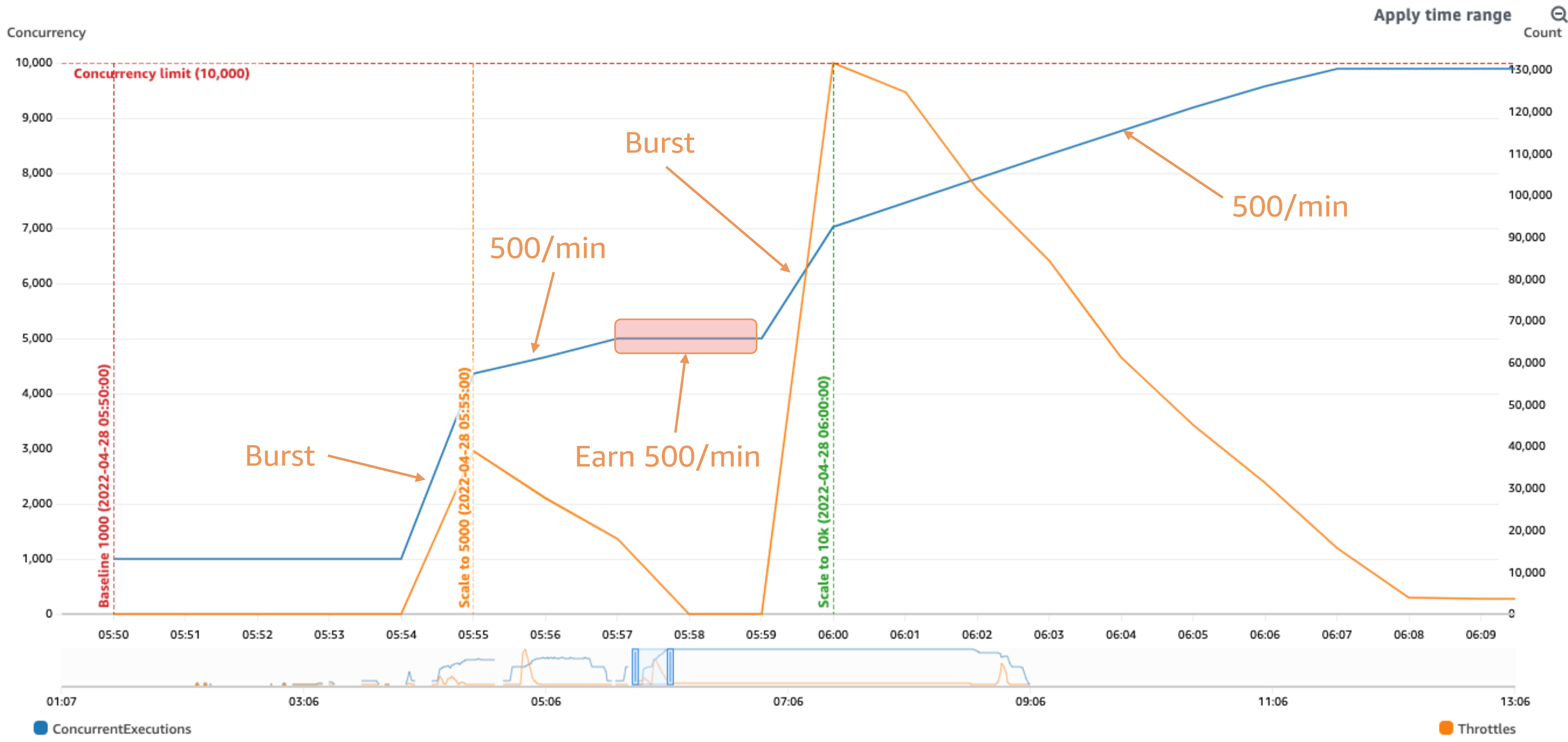




Apply time range
Count
Reserved concurrency (50)
25 RPS (2022-04-28 16:45:00)
50 RPS (2022-04-28 16:48:00)
75 RPS (2022-04-28 16:51:00)
60.00
55.00
50.00
45.00
40.00
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0
22,000
20,000
18,000
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
16:45
16:45
16:46
16:46
16:47
16:47
16:48
16:48
16:49
16:49
16:50
16:50
16:51
16:51
16:52
16:52
16:53
16:39
16:48
16:58
17:08
17:18
17:28
17:38
ConcurrentExecutions
Throttles

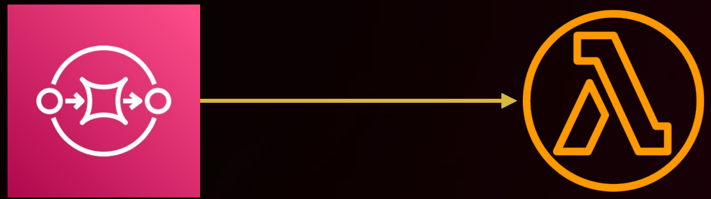
Test: Synchronous Lambda invocations

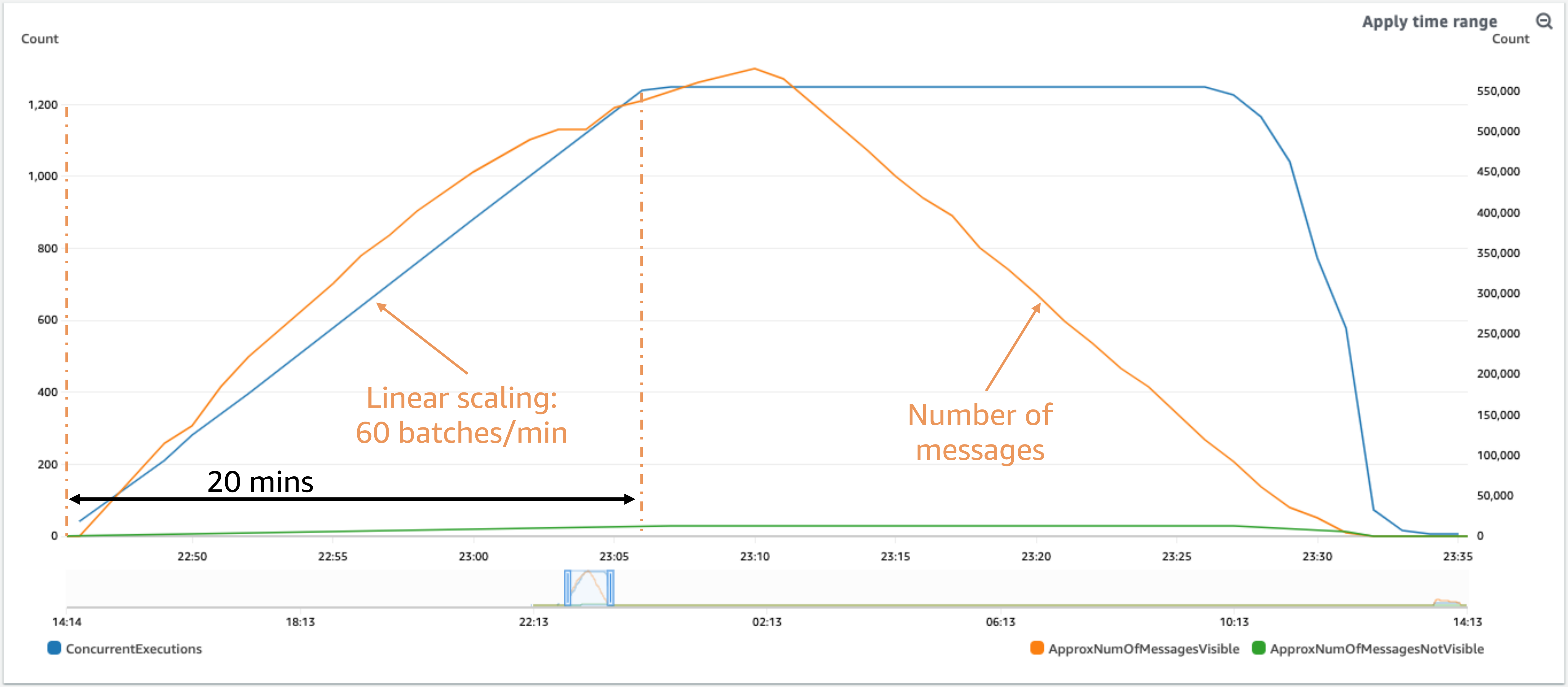
- Account/region quota: 10,000 (us-west-2)
- Direct Lambda sync invoke with Go
- Launch three batches with various concurrency every 5 mins
  - t1: 1000
  - t1 + 5m: 4000 (5000 total)
  - t1 + 10m: 5000 (10000 total)
- No delay between requests


Apply time range
Concurrency
Count
Concurrency limit (10,000)
Baseline 1000 (2022-04-28 05:50:00)
Scale to 5000 (2022-04-28 05:55:00)
Scale to 10k (2022-04-28 06:00:00)
Burst
500/min
Earn 500/min
Burst
500/min
10,000
9,000
8,000
7,000
6,000
5,000
4,000
3,000
2,000
1,000
0
130,000
120,000
110,000
100,000
90,000
80,000
70,000
60,000
50,000
40,000
30,000
20,000
10,000
0
05:50
05:51
05:52
05:53
05:54
05:55
05:56
05:57
05:58
05:59
06:00
06:01
06:02
06:03
06:04
06:05
06:06
06:07
06:08
06:09
01:07
03:06
05:06
07:06
09:06
11:06
13:06
ConcurrentExecutions
Throttles

Test: Lambda scaling with long SQS queue

- Account/region concurrency quota: 10,000 (us-west-2)
- Put an artificial delay in Lambda function
- Load a lot of messages onto standard queue

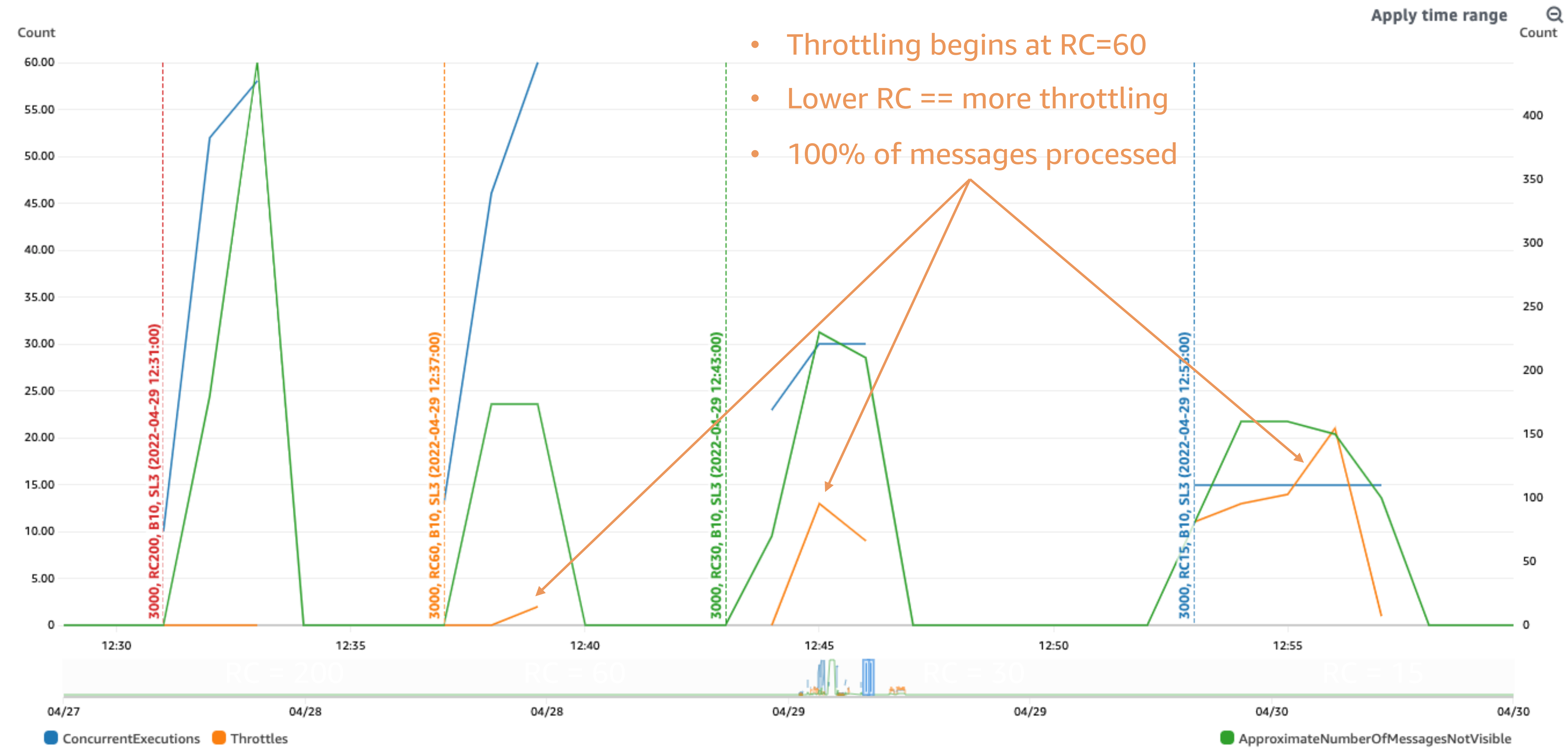



Apply time range
Count
Count
1,200
1,000
800
600
400
200
0
550,000
500,000
450,000
400,000
350,000
300,000
250,000
200,000
150,000
100,000
50,000
0
22:50
22:55
23:00
23:05
23:10
23:15
23:20
23:25
23:30
23:35
14:14
18:13
22:13
02:13
06:13
10:13
14:13
Linear scaling: 60 batches/min
Number of messages
20 mins
🟦 ConcurrentExecutions
🟧 ApproxNumOfMessagesVisible
🟩 ApproxNumOfMessagesNotVisible

Test: Lambda with reserved concurrency and SQS

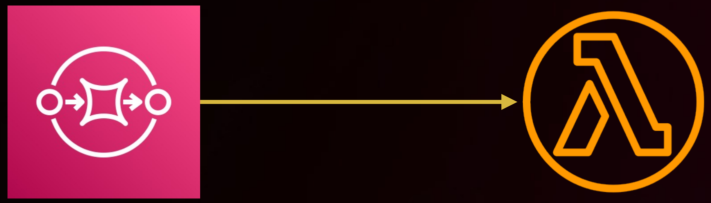
- Load 3,000 messages onto standard queue
- Use various reserved concurrency settings:
  - 200
  - 60
  - 30
  - 15

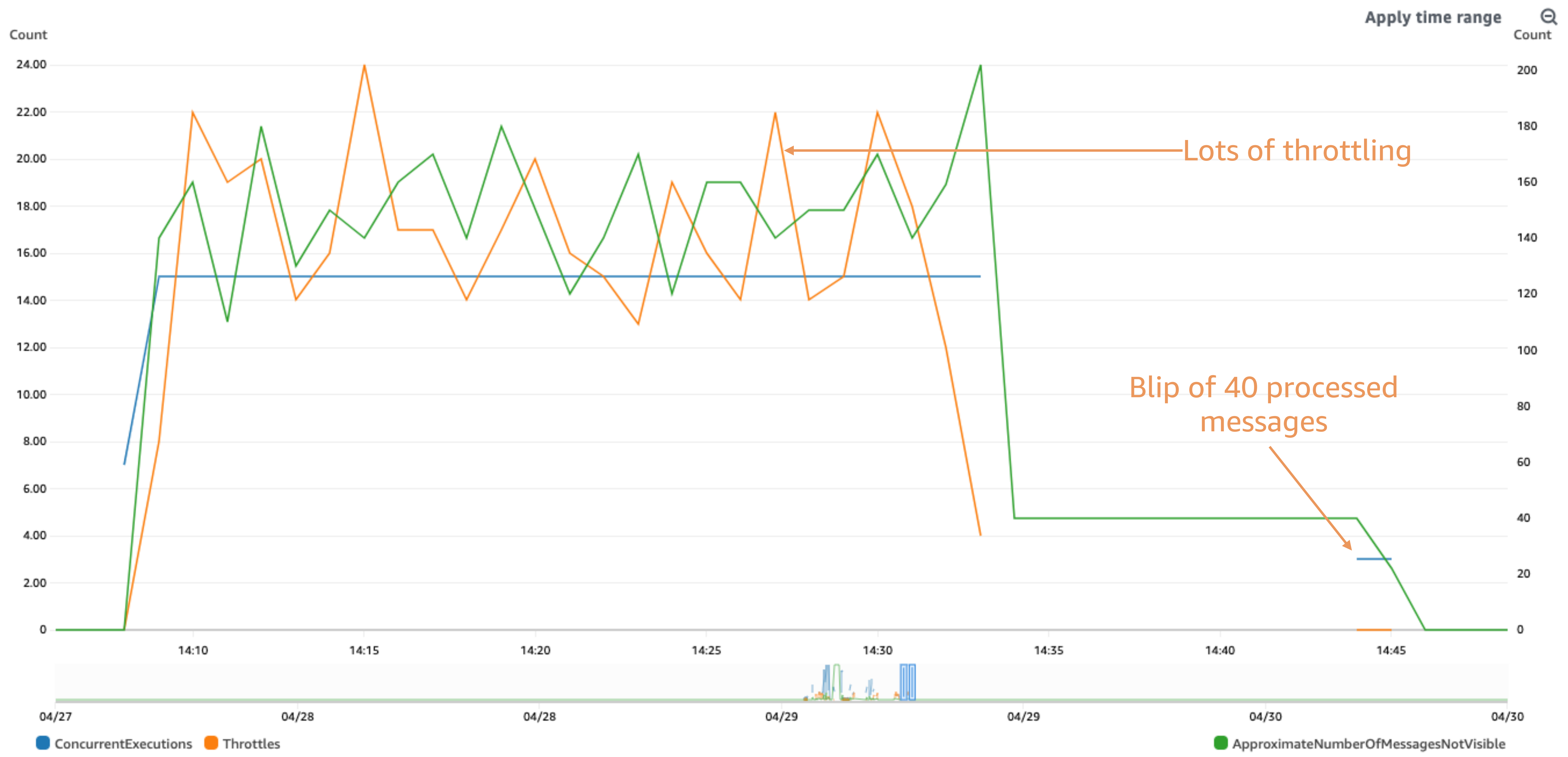
- Throttling begins at RC=60
- Lower RC == more throttling
- 100% of messages processed

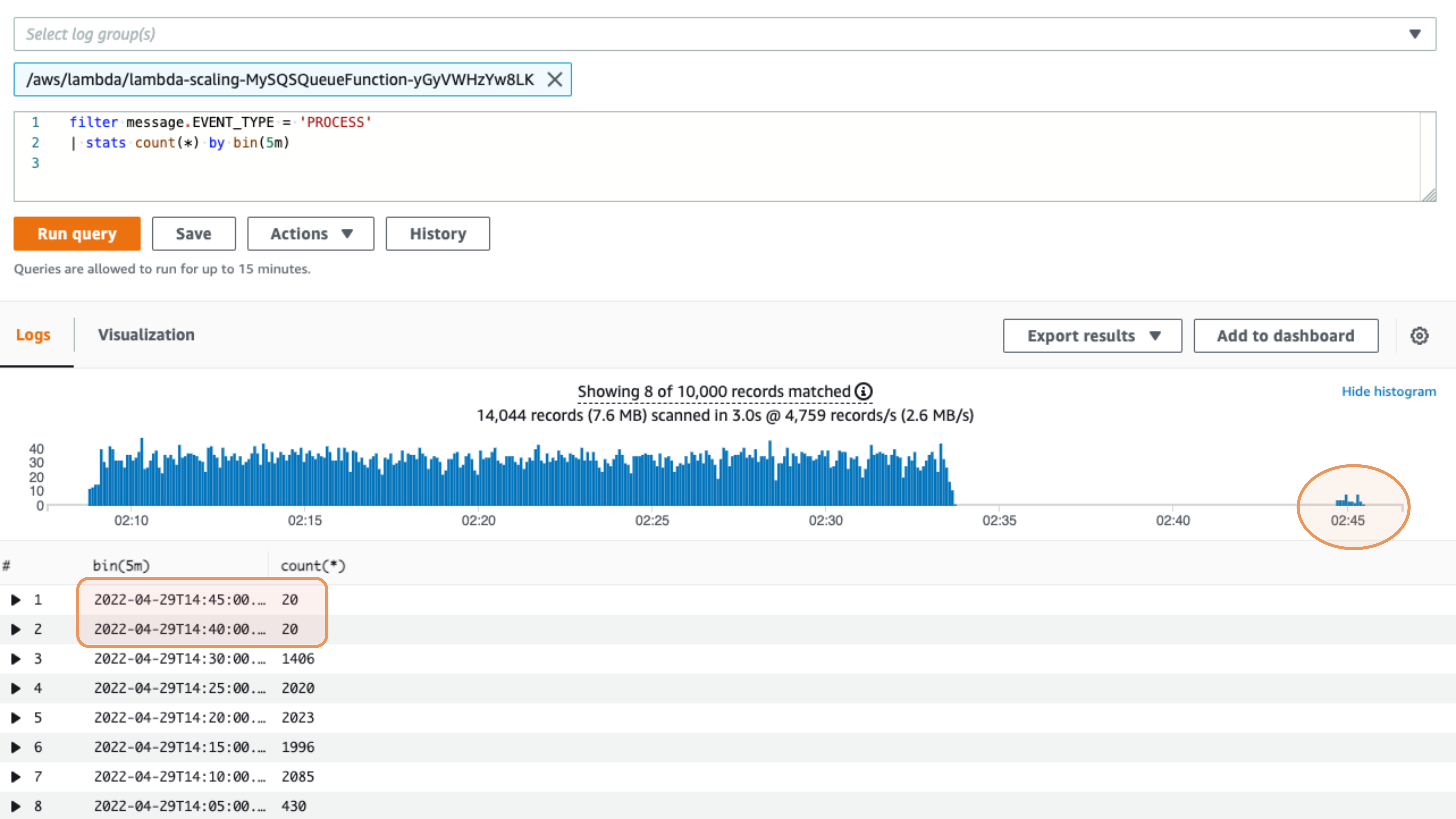


Test: Lambda with reserved concurrency and SQS

- Load 10,000 messages onto standard queue
- Set reserved concurrency to 15
- Insert longer artificial delay




Apply time range
Count
Count
24.00
22.00
20.00
18.00
16.00
14.00
12.00
10.00
8.00
6.00
4.00
2.00
0
200
180
160
140
120
100
80
60
40
20
0
14:10
14:15
14:20
14:25
14:30
14:35
14:40
14:45
04/27
04/28
04/28
04/29
04/29
04/30
04/30
Lots of throttling
Blip of 40 processed messages
ConcurrentExecutions
Throttles
ApproximateNumberOfMessagesNotVisible



```
filter message.EVENT_TYPE = 'PROCESS'
| stats count(*) by bin(5m)
```

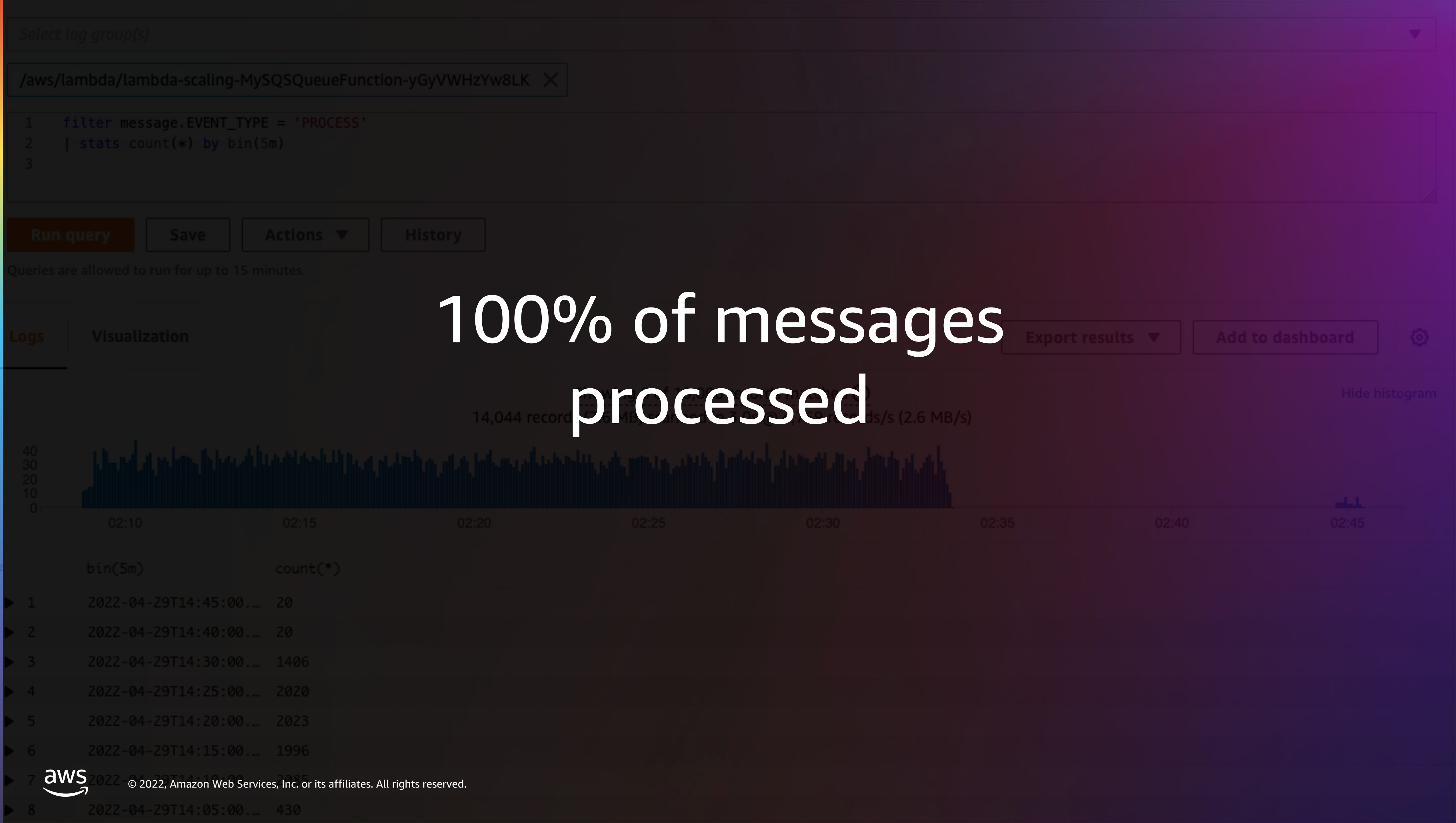
Queries are allowed to run for up to 15 minutes.

Showing 8 of 10,000 records matched

14,044 records (7.6 MB) scanned in 3.0s @ 4,759 records/s (2.6 MB/s)

#	bin(5m)	count(*)
1	2022-04-29T14:45:00.…	20
2	2022-04-29T14:40:00.…	20
3	2022-04-29T14:30:00.…	1406
4	2022-04-29T14:25:00.…	2020
5	2022-04-29T14:20:00.…	2023
6	2022-04-29T14:15:00.…	1996
7	2022-04-29T14:10:00.…	2085
8	2022-04-29T14:05:00.…	430

100% of messages processed



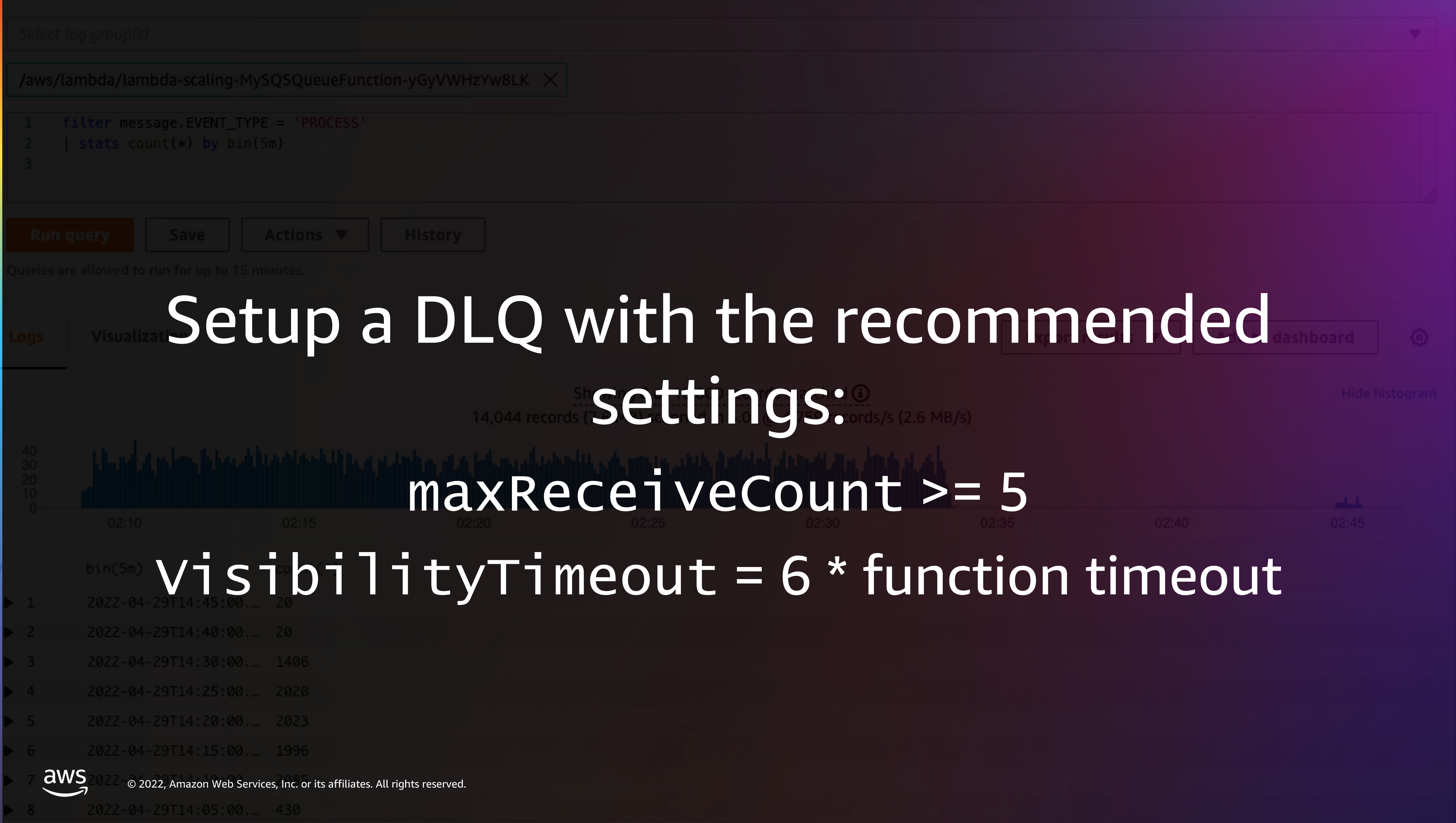
/aws/lambda/lambda-scaling-MySQSQueueFunction-yGyVWHzYw8LK

```
filter message.EVENT_TYPE = 'PROCESS'
| stats count(*) by bin(5m)
```

Queries are allowed to run for up to 15 minutes.

14,044 record ... ds/s (2.6 MB/s)

```
    bin(5m)                    count(*)
1   2022-04-29T14:45:00...     20
2   2022-04-29T14:40:00...     20
3   2022-04-29T14:30:00...     1406
4   2022-04-29T14:25:00...     2020
5   2022-04-29T14:20:00...     2023
6   2022-04-29T14:15:00...     1996
7   2022-04-29T14:10:00...     2085
8   2022-04-29T14:05:00...     430
```



/aws/lambda/lambda-scaling-MySQSQueueFunction-yGyVWHzYw8LK

```
filter message.EVENT_TYPE = 'PROCESS'
| stats count(*) by bin(5m)
```

Queries are allowed to run for up to 15 minutes.

14,044 records

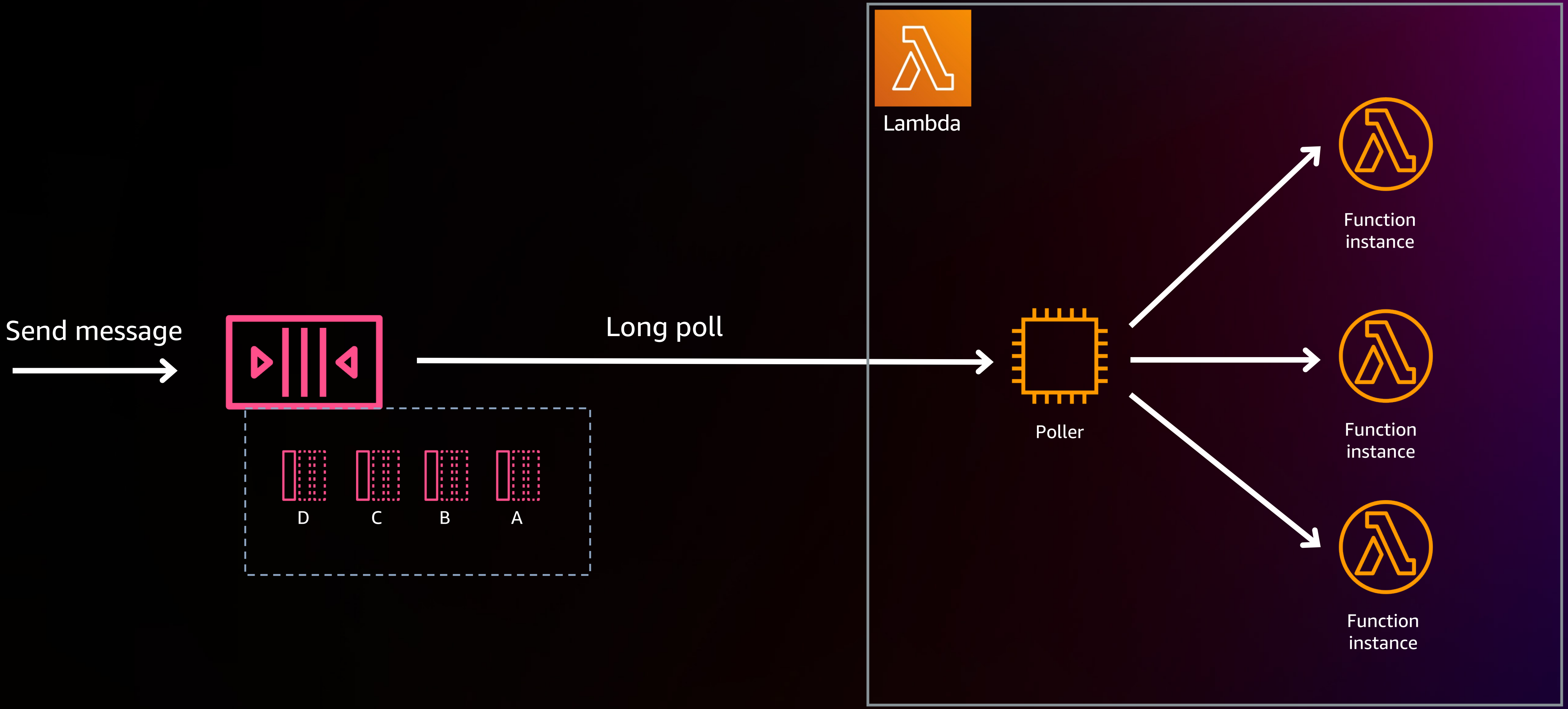
```
bin(5m)
1  2022-04-29T14:45:00...  20
2  2022-04-29T14:40:00...  20
3  2022-04-29T14:30:00...  1406
4  2022-04-29T14:25:00...  2020
5  2022-04-29T14:20:00...  2023
6  2022-04-29T14:15:00...  1996
8  2022-04-29T14:05:00...  430
```

Setup a DLQ with the recommended settings:

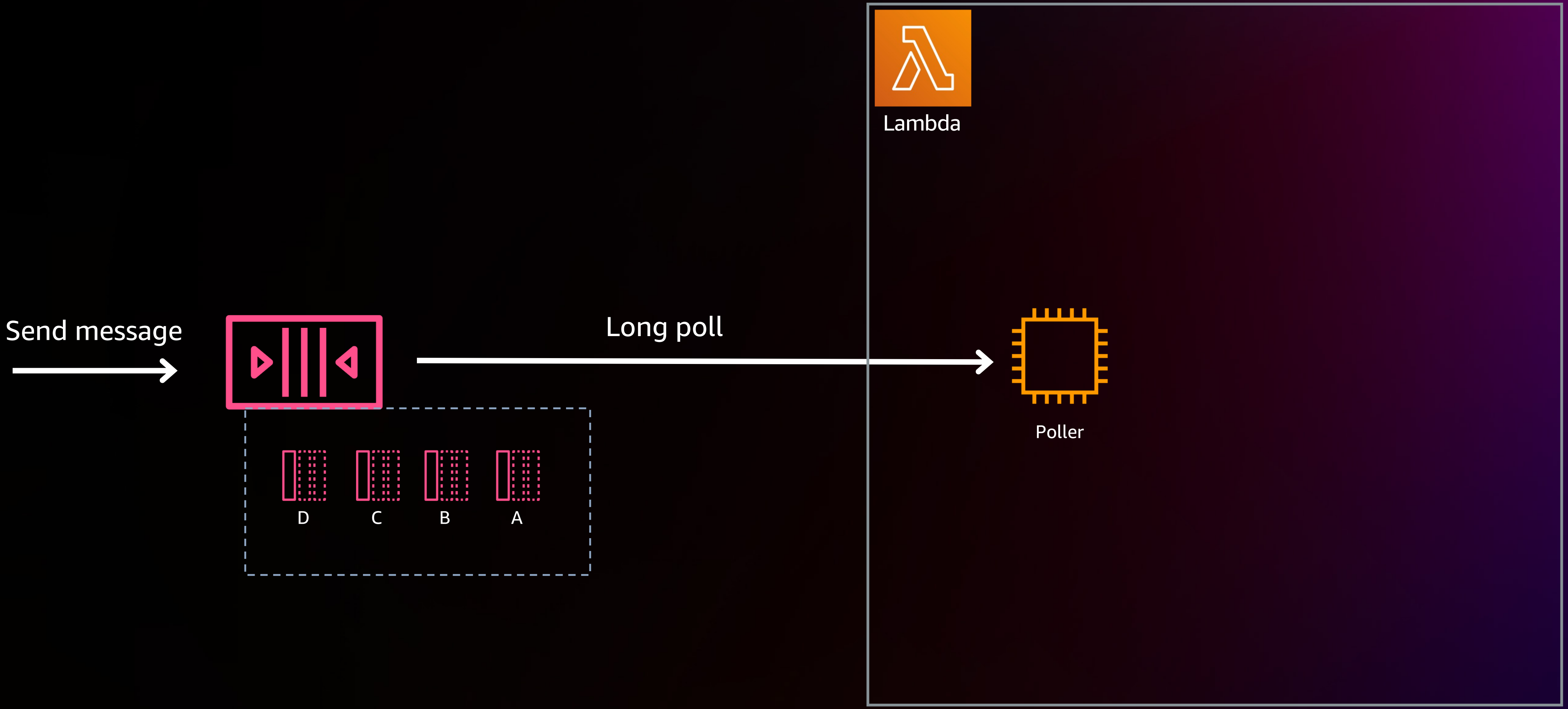
`maxReceiveCount >= 5`

`VisibilityTimeout` = 6 * function timeout

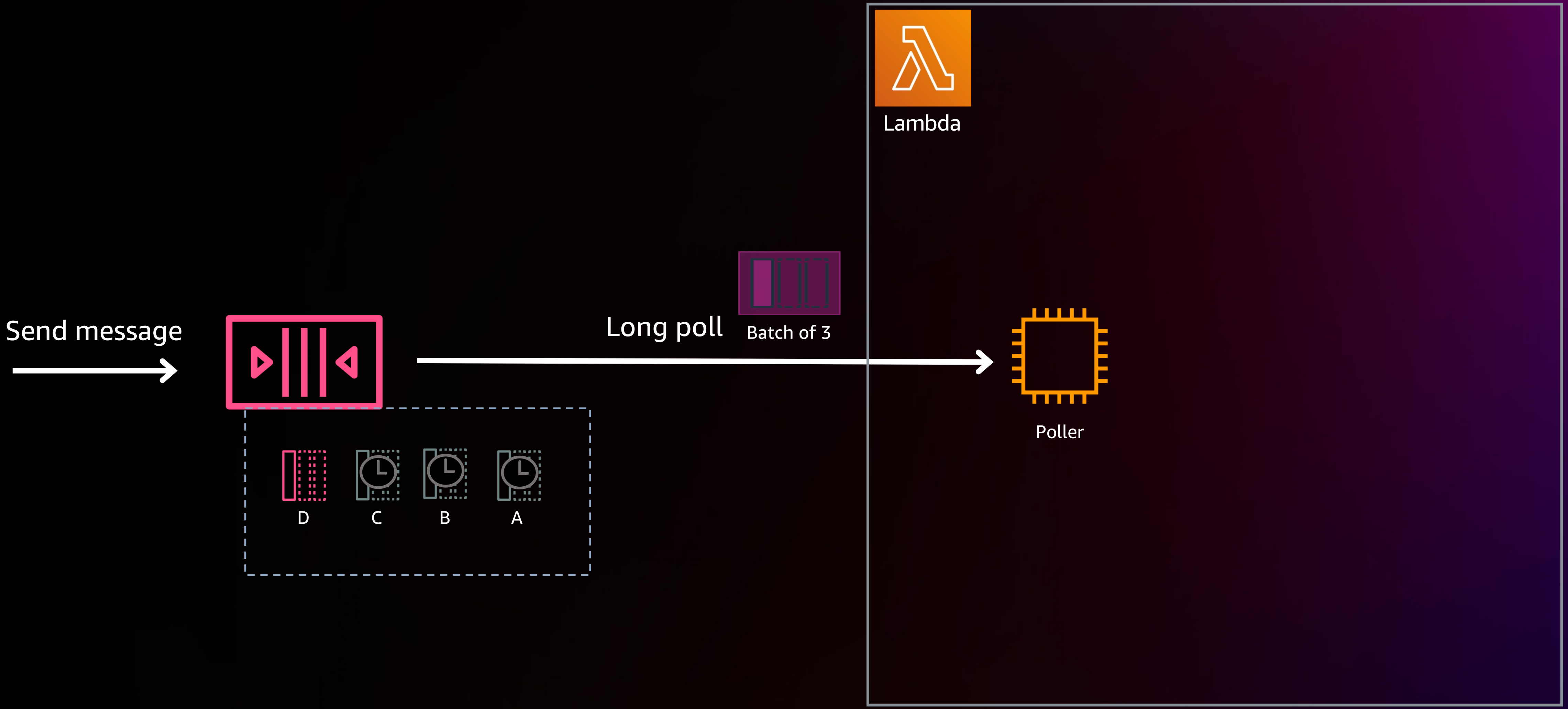
Amazon SQS to Lambda: Scaling Invocations



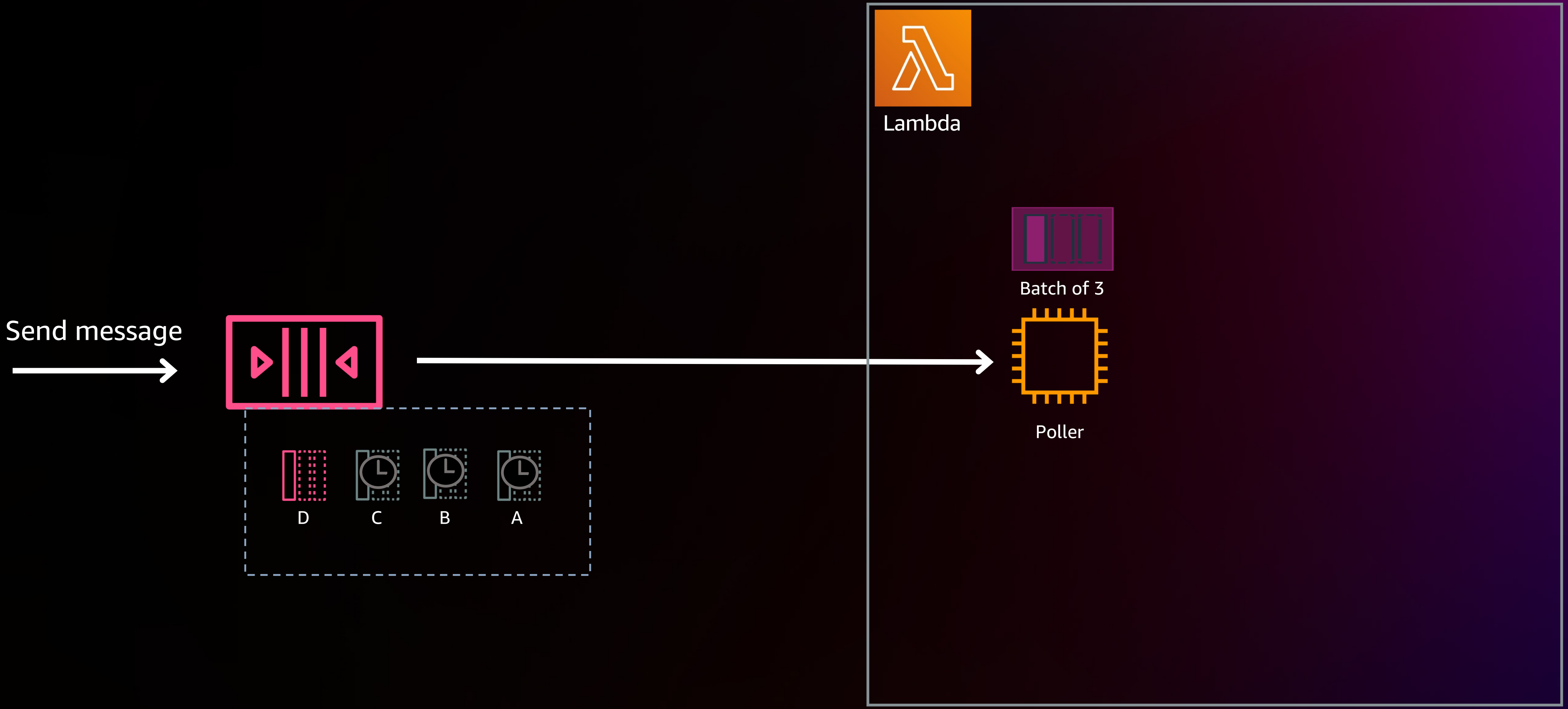
Amazon SQS to Lambda: Scaling Invocations



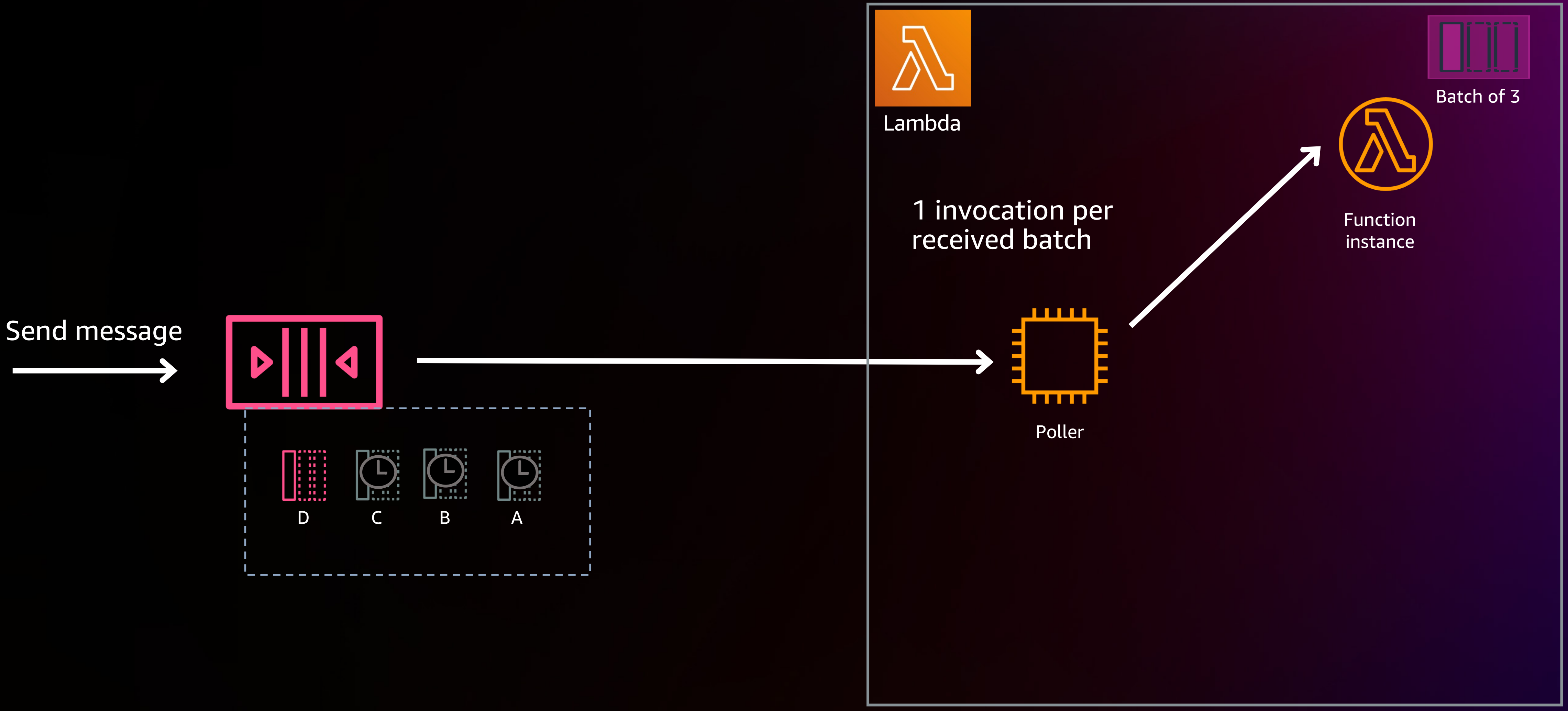
Amazon SQS to Lambda: Scaling Invocations



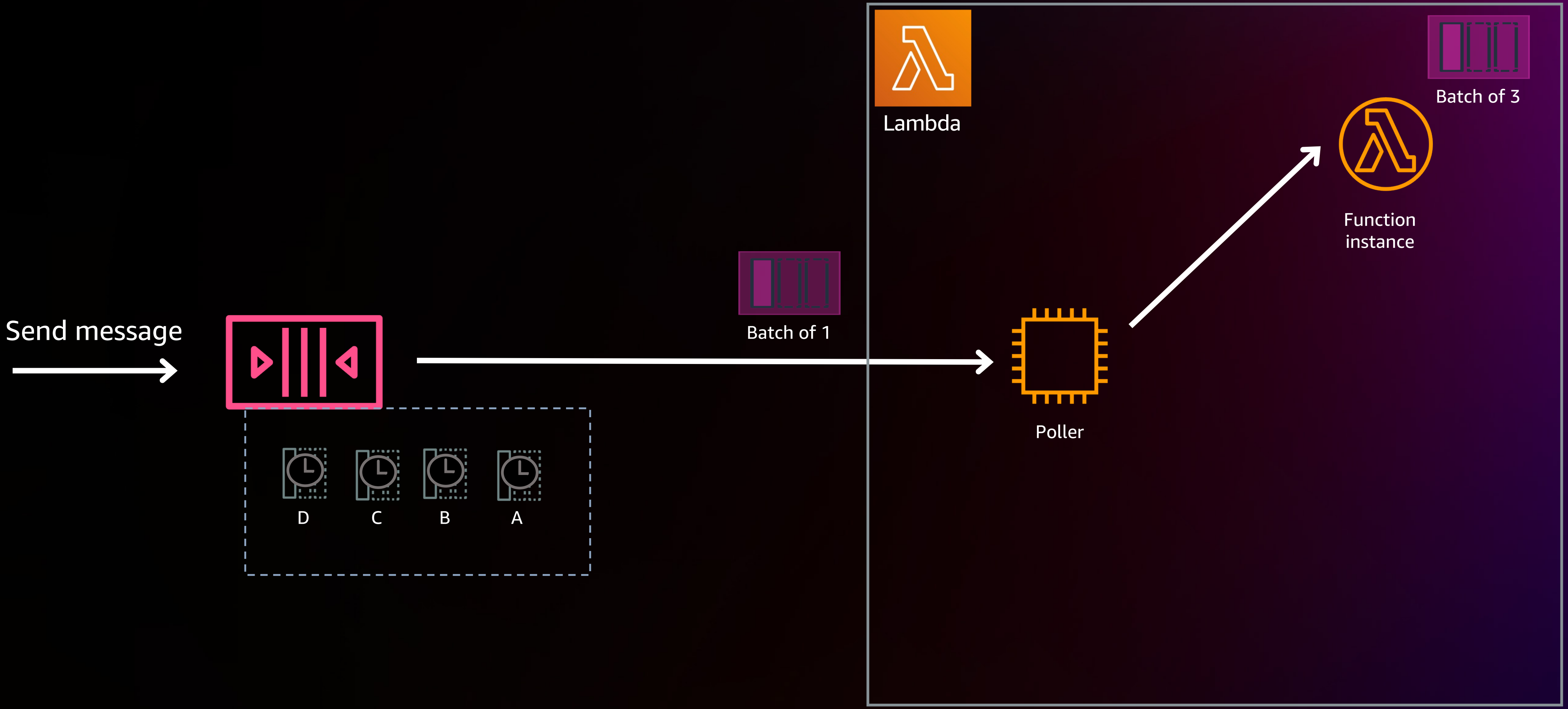
Amazon SQS to Lambda: Scaling Invocations



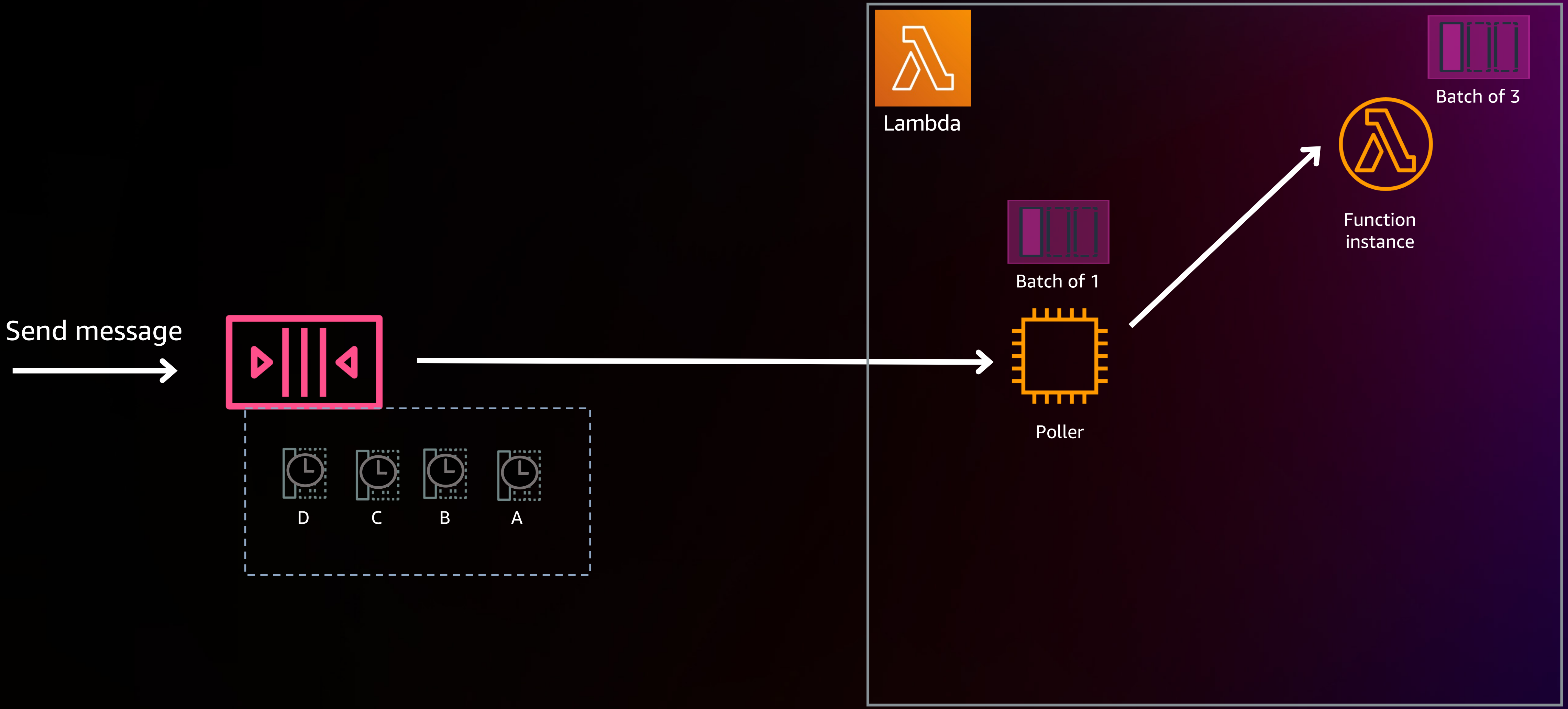
Amazon SQS to Lambda: Scaling Invocations



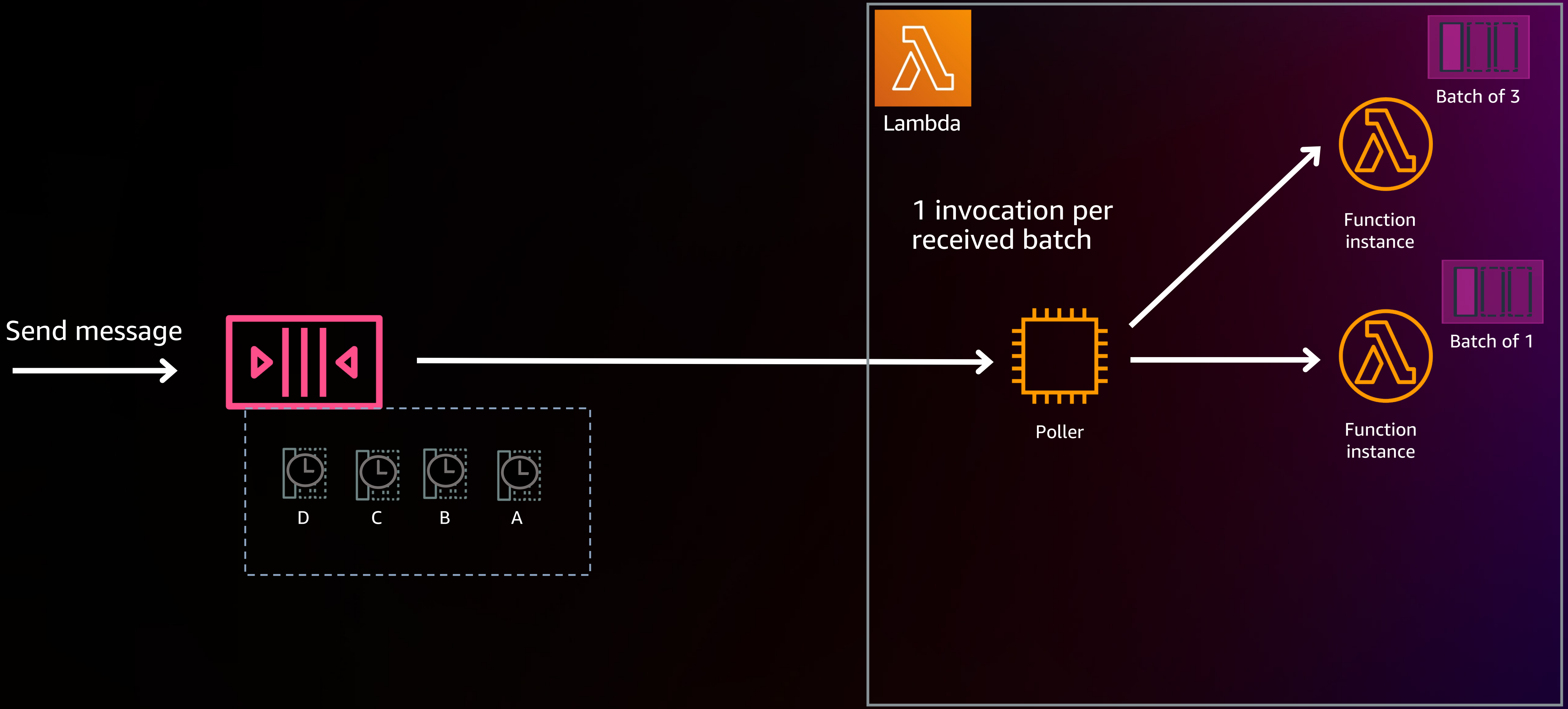
Amazon SQS to Lambda: Scaling Invocations



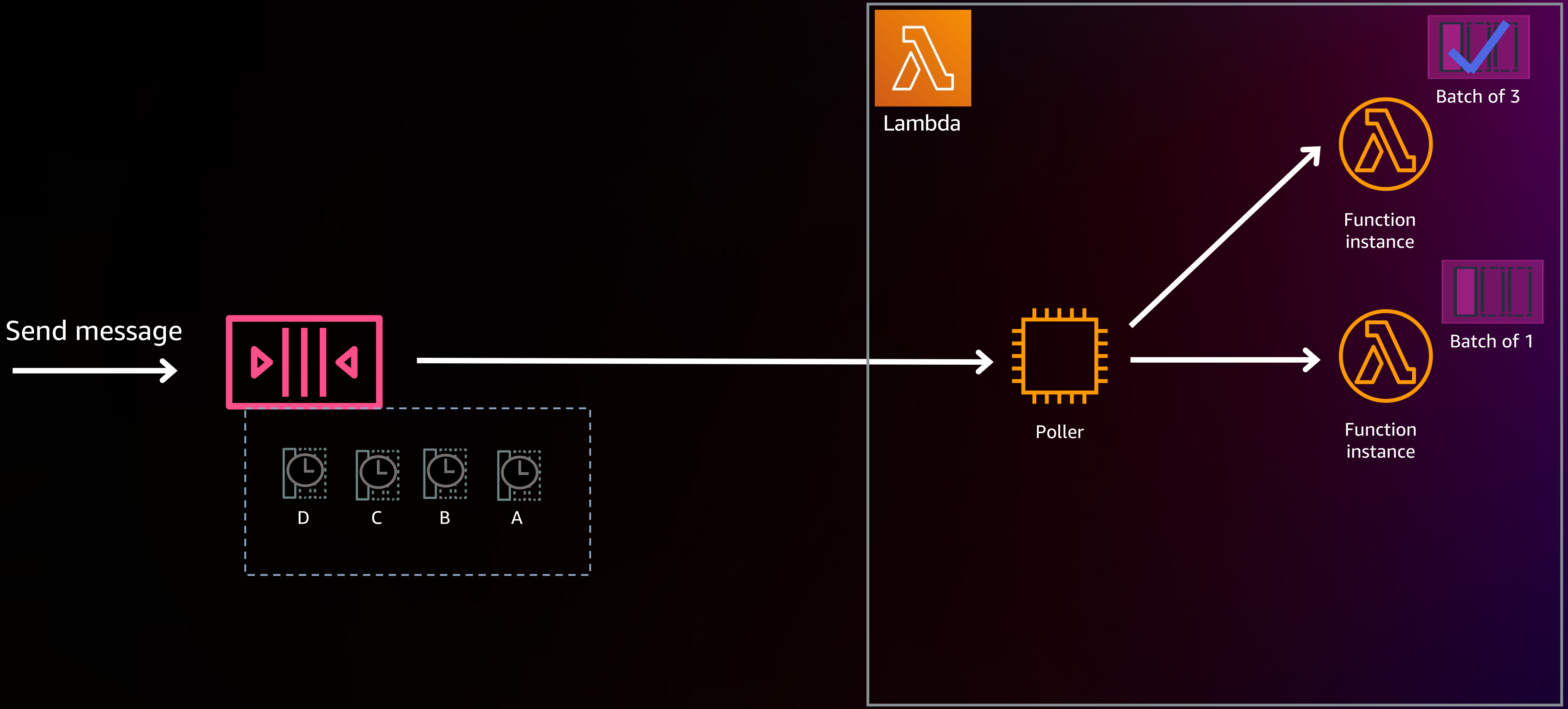
Amazon SQS to Lambda: Scaling Invocations



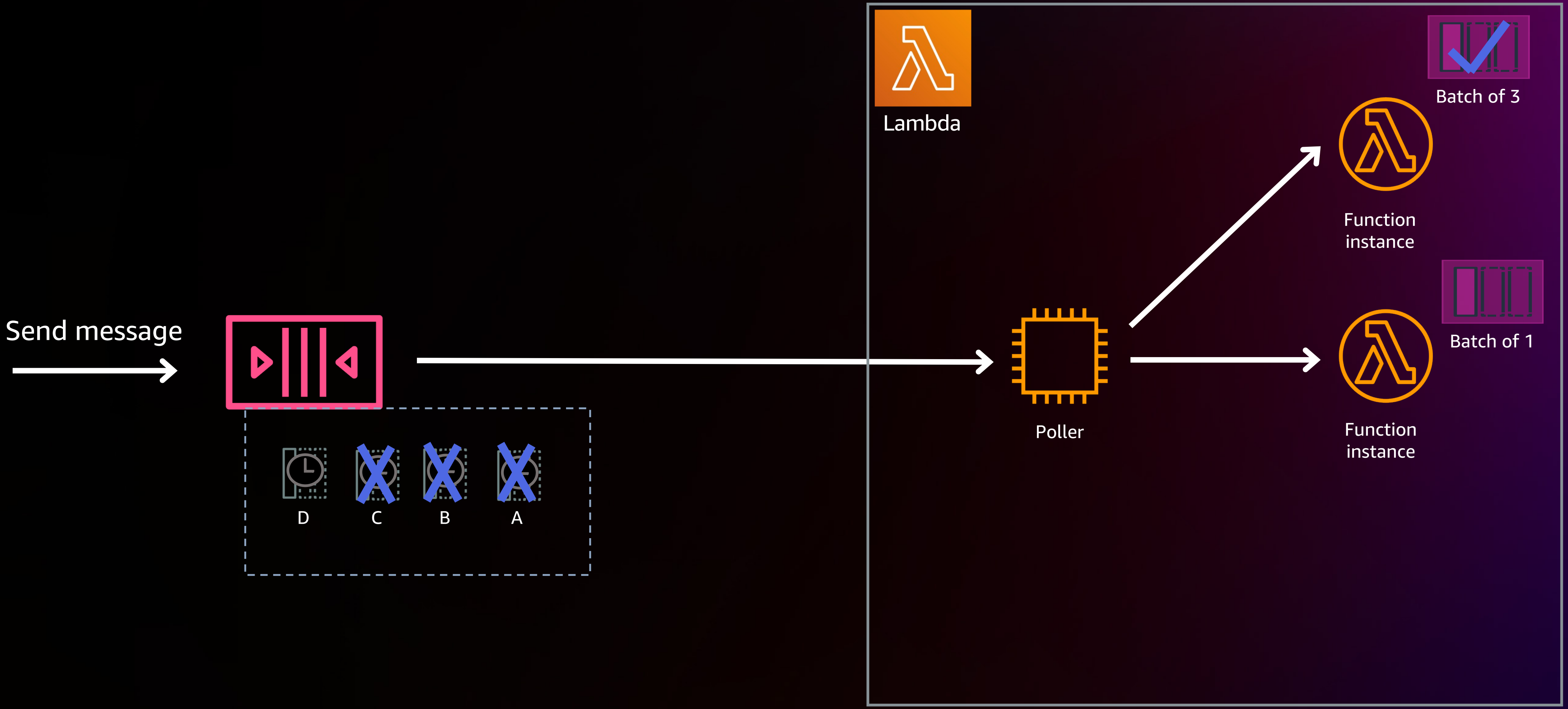
Amazon SQS to Lambda: Scaling Invocations



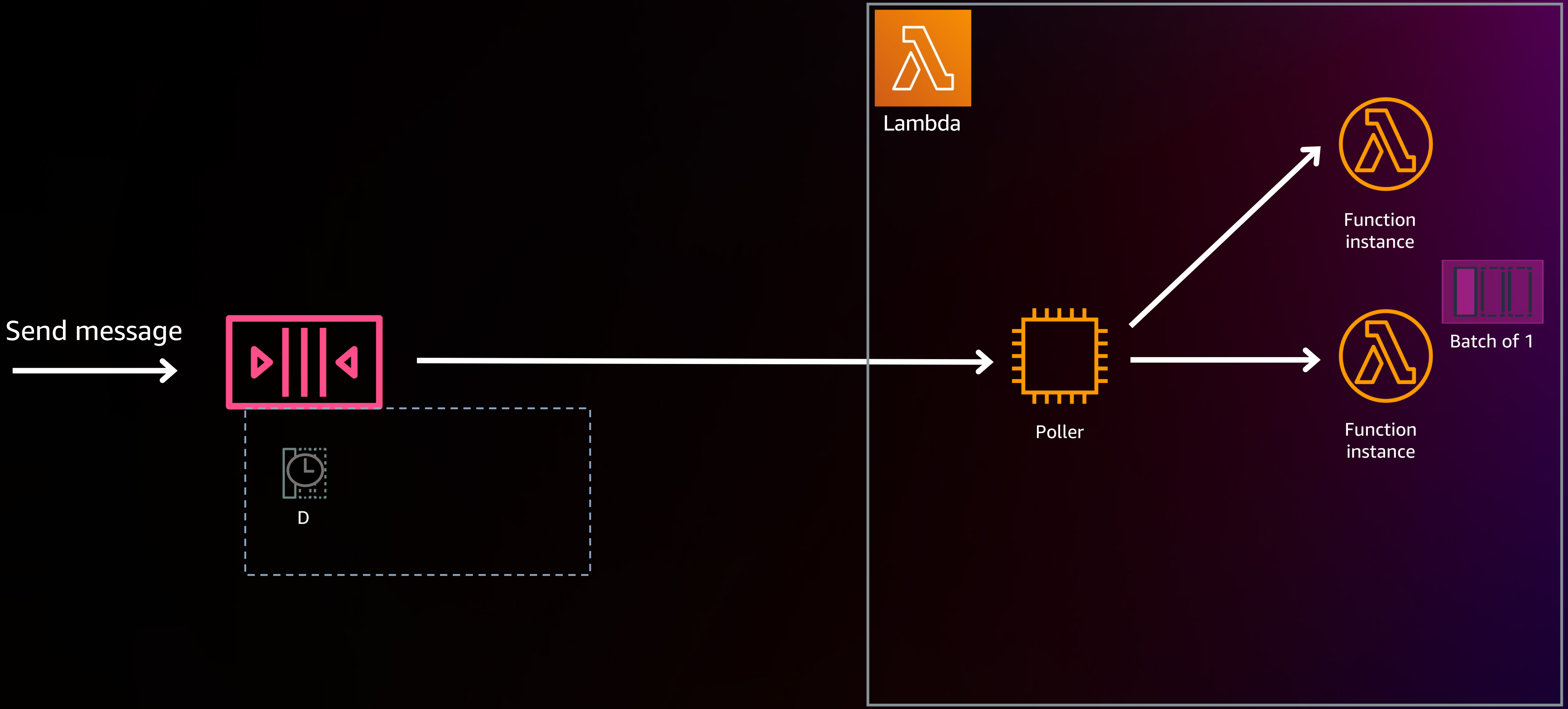
Amazon SQS to Lambda: Scaling Invocations



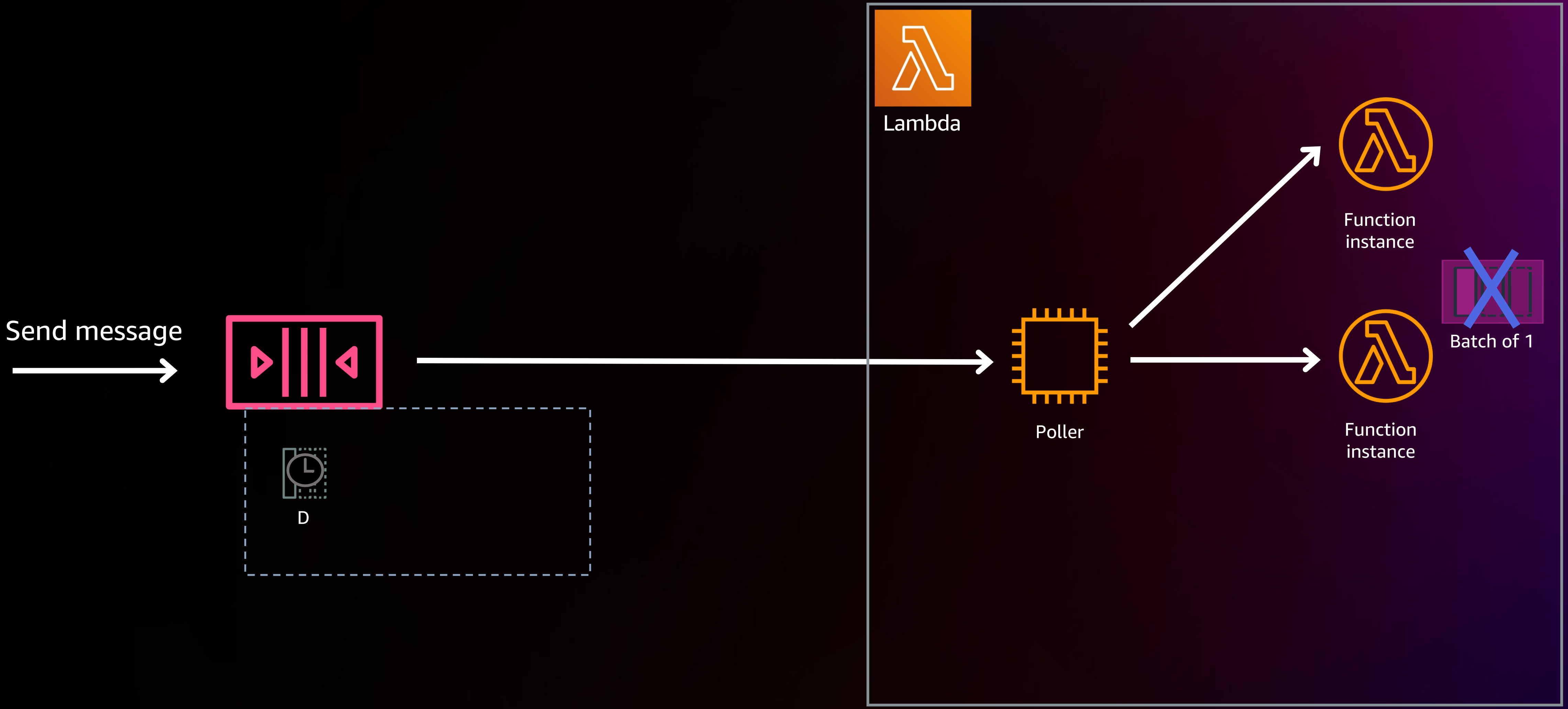
Amazon SQS to Lambda: Scaling Invocations



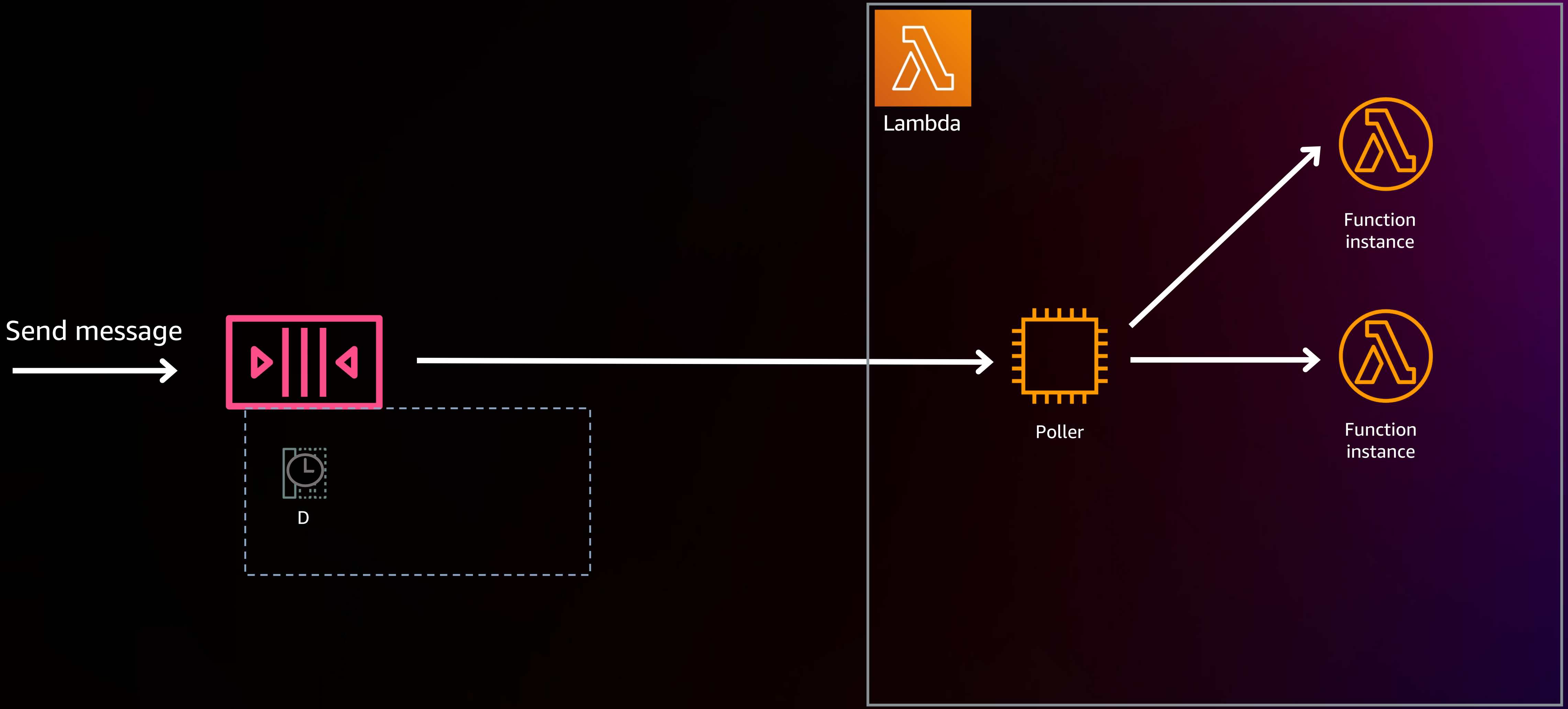
Amazon SQS to Lambda: Scaling Invocations



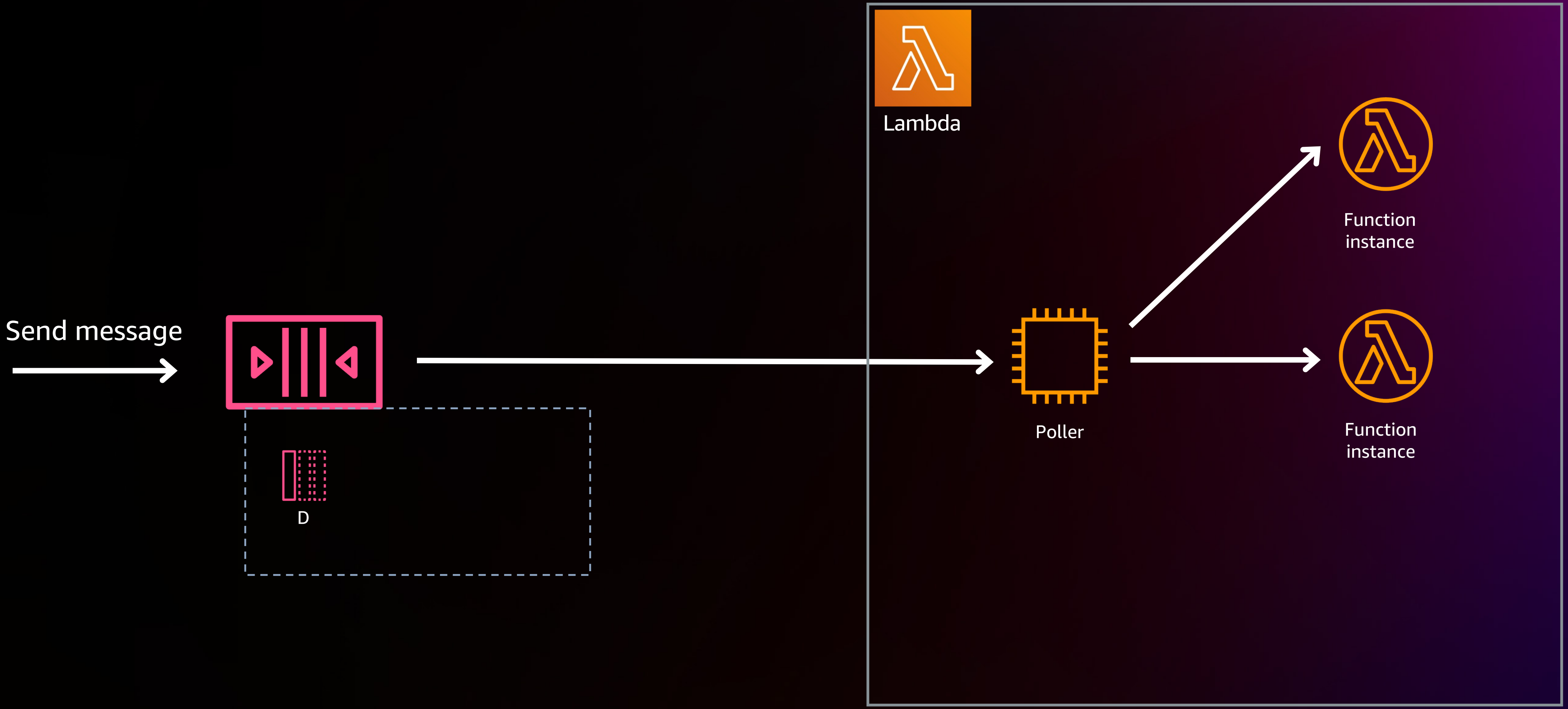
Amazon SQS to Lambda: Scaling Invocations



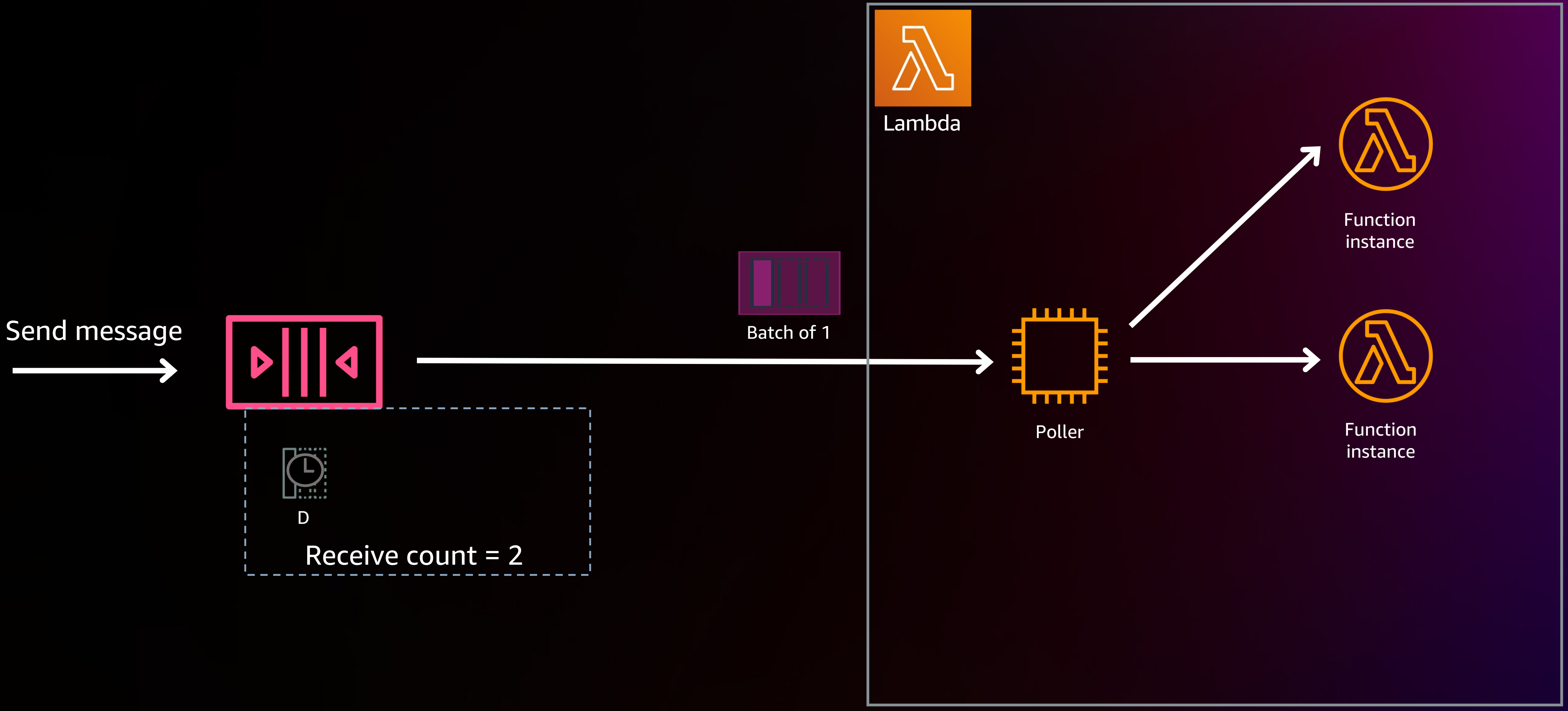
Amazon SQS to Lambda: Scaling Invocations



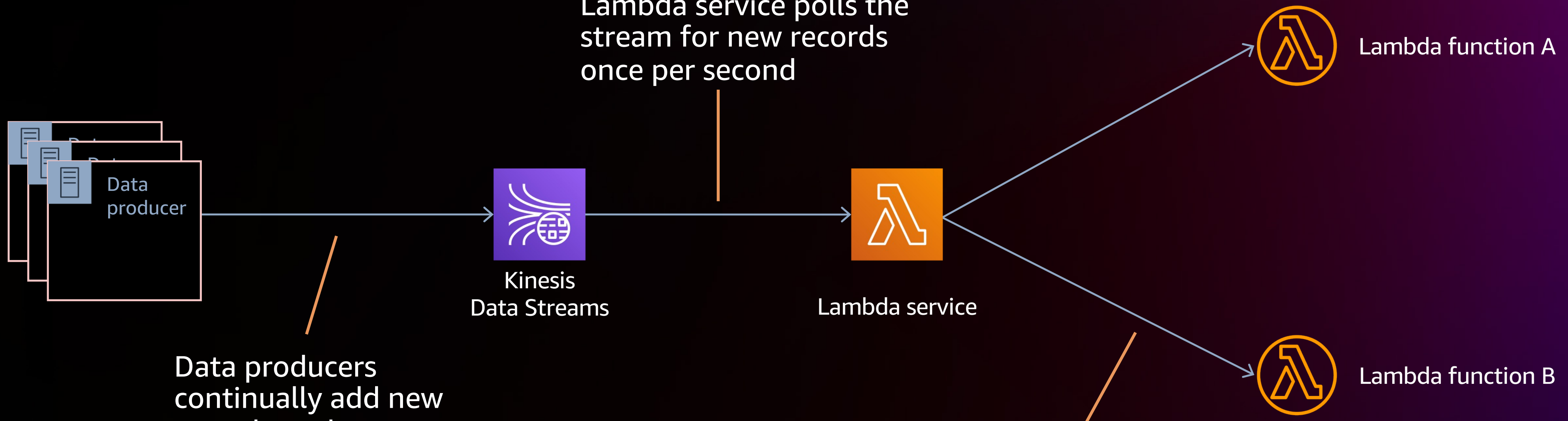
Amazon SQS to Lambda: Scaling Invocations



Amazon SQS to Lambda: Scaling Invocations



Kinesis Data Streams



Up to five Lambda functions can be subscribed to a single stream*

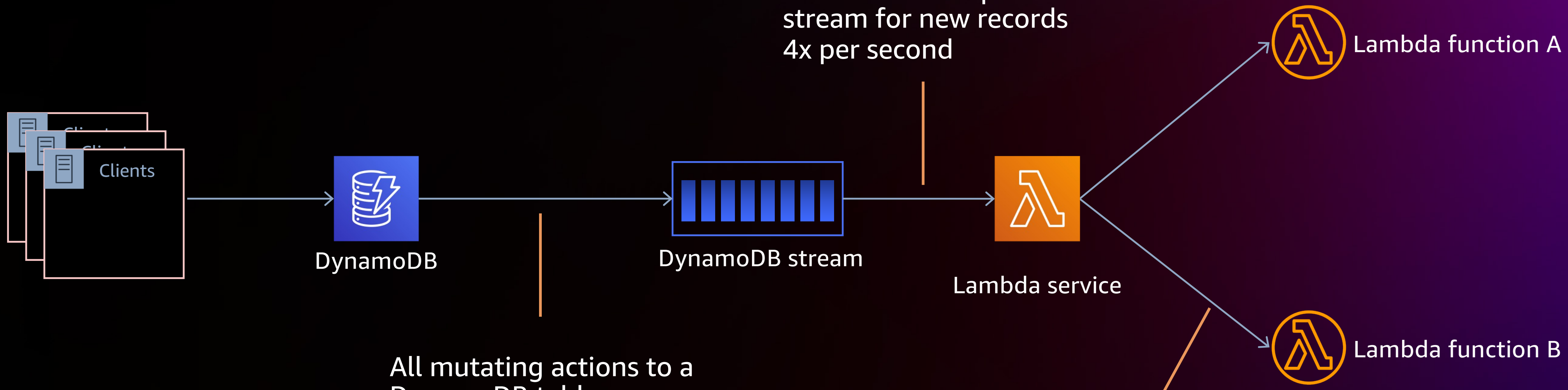
Lambda service polls the stream for new records once per second

Data producers continually add new records to the stream

When new records are available, the subscribed Lambda functions are invoked with new records

*Kinesis Data Analytics and Kinesis Data Firehose also count toward this subscriber limit

DynamoDB Streams



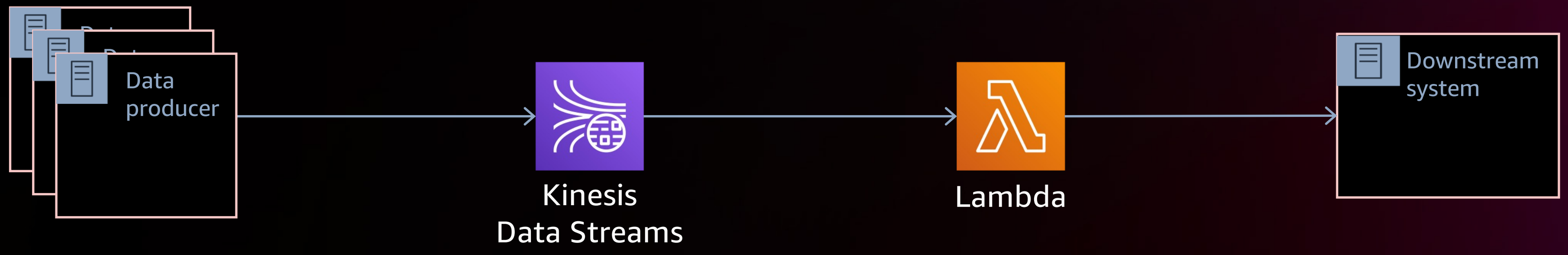
Up to two Lambda functions can be subscribed to a single DynamoDB stream

Lambda service polls the stream for new records 4x per second

All mutating actions to a DynamoDB table are captured as an item on the stream

When new records are available, the subscribed Lambda functions are invoked with new records

Kinesis data stream monitoring



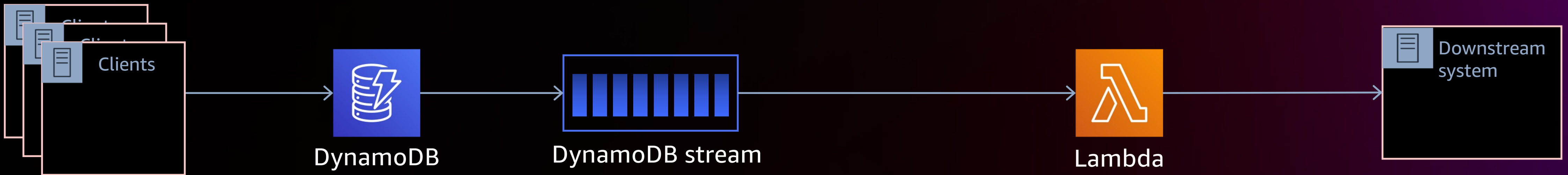
Kinesis Metrics/Alarms

- GetRecords.IteratorAgeMilliseconds
- IncomingRecords/IncomingBytes
- ReadProvisionedThroughputExceeded
- WriteProvisionedThroughputExceeded

Lambda Metrics/Alarms

- Errors
- IteratorAge
- Throttles

DynamoDB stream monitoring



DynamoDB Streams metrics/alarms

- ReturnedRecordsCount/ReturnedBytes
- UserErrors

Lambda metrics/alarms

- Errors
- IteratorAge
- Throttles
- Duration