
AWS
re:Invent

SVS403

AWS Lambda – Part 1: Optimizing your serverless applications

Chris Munns

Senior Manager/Principal Developer Advocate – Serverless

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why are we here today?



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Today's focus:



Serverless applications



AWS Lambda

Serverless applications

Function



Node.js
Python
Java
C#
Go
Ruby
Runtime API

Serverless applications

Event source



Function



Changes in data state



Requests to endpoints



Changes in Resource state



Node.js
Python
Java
C#
Go
Ruby
Runtime API

Serverless applications

Event source



Changes in data state



Requests to endpoints



Changes in Resource state



Function



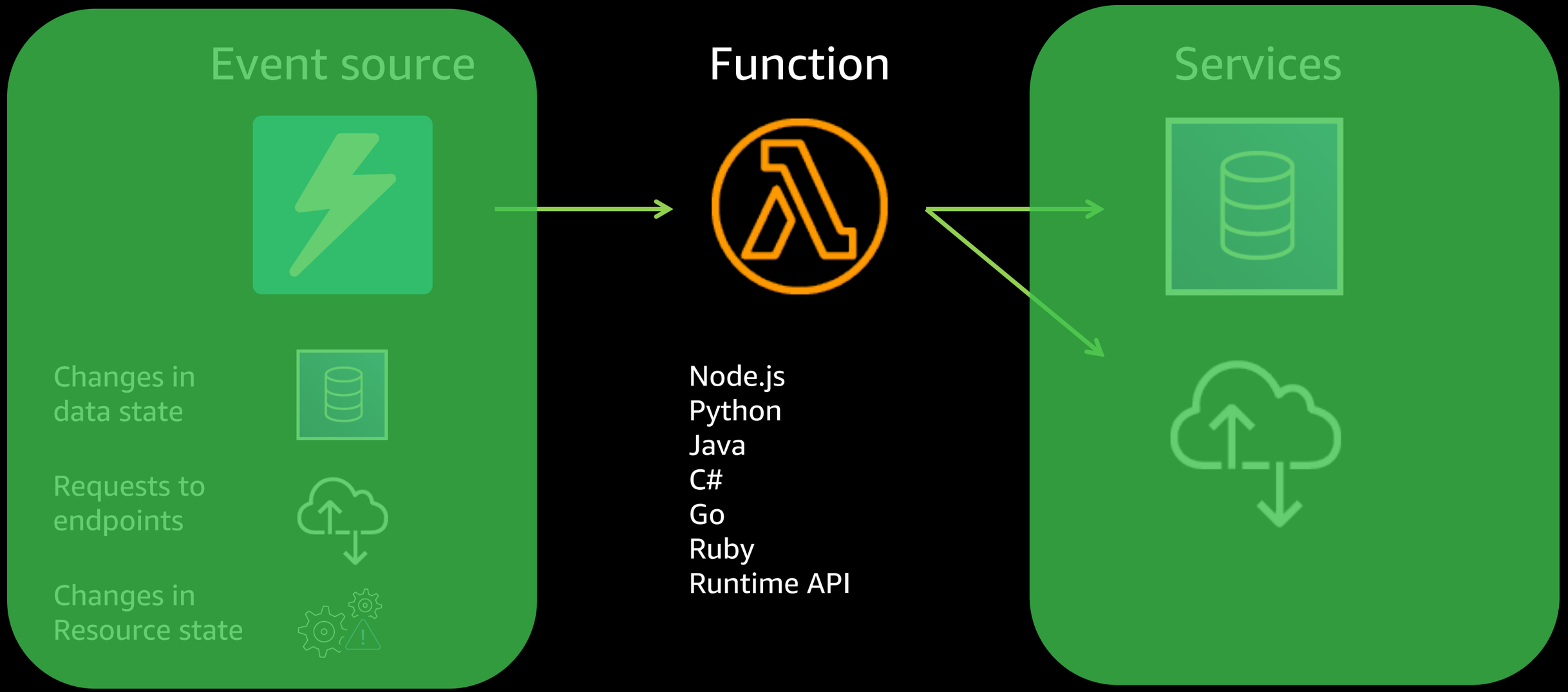
Node.js
Python
Java
C#
Go
Ruby
Runtime API

Services

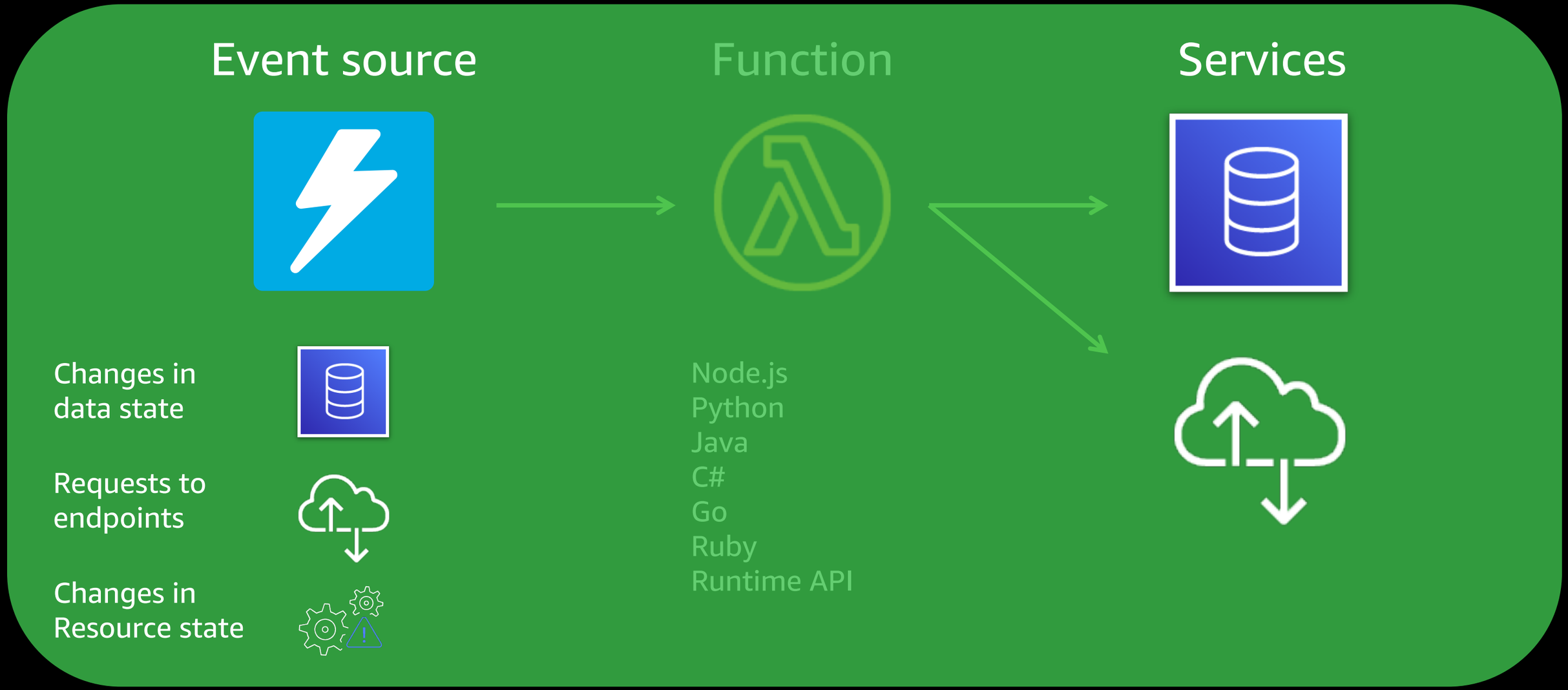




Serverless applications



Serverless applications



But we only have 30 minutes this year …



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

This talk is split with SVS405

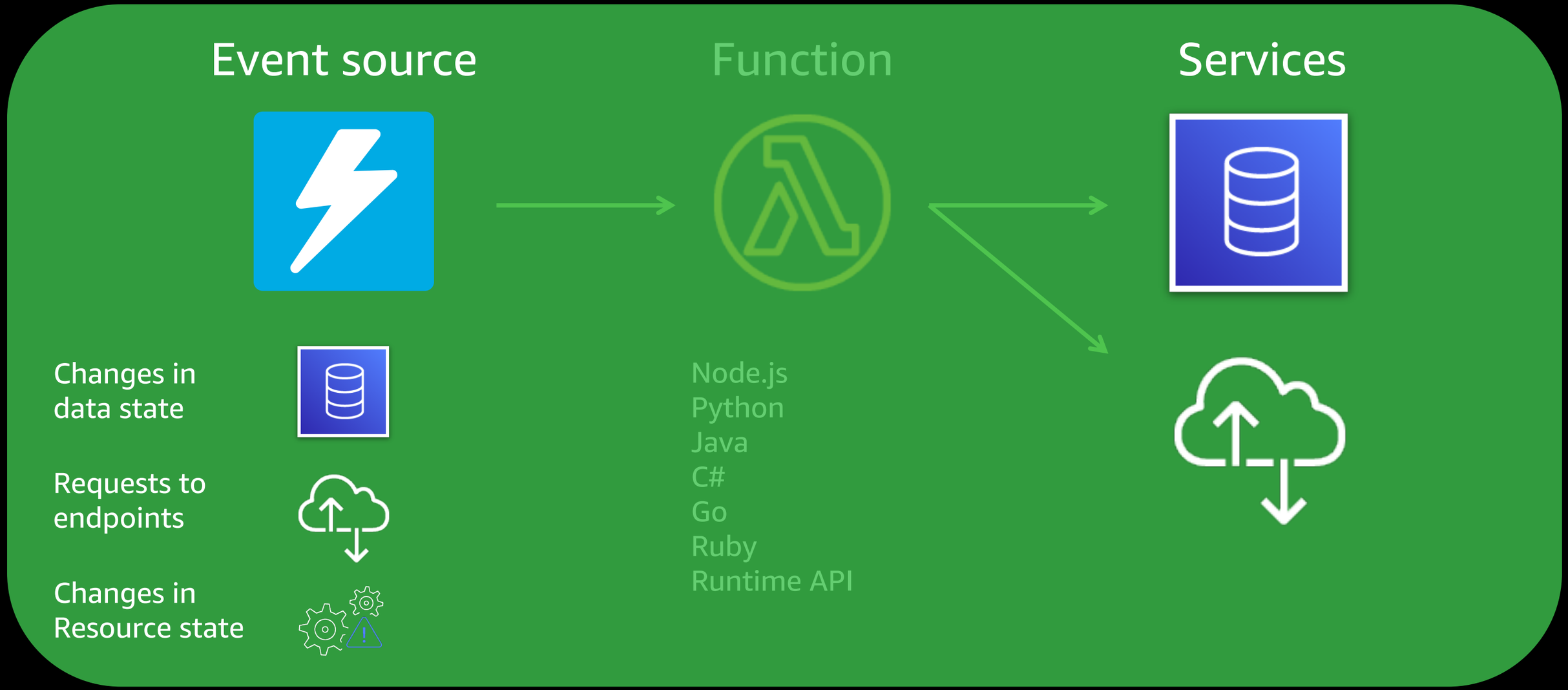
Check out **SVS405** – *AWS Lambda – Part 2: Optimizing your Lambda function performance* by Alex Casalboni

You'll learn about:

- Minimizing cold start impact with Lambda Provisioned Concurrency
- Tuning memory for function performance
- Using AWS Lambda Power Tuning



Serverless applications



Two focus areas for optimization

On the invoke side

- Using the right execution model
- Concurrency and you
- Handling retries and failures

On the integrated services side

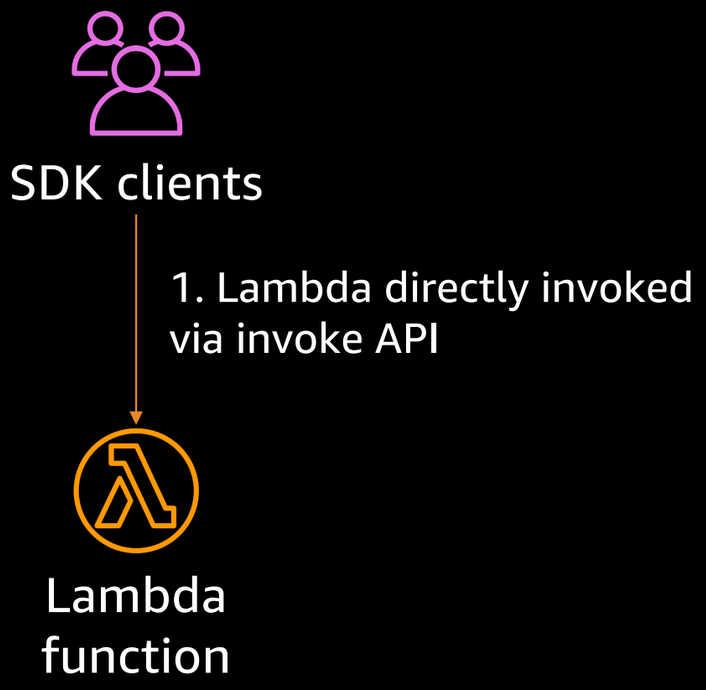
- Lambda storage options
- Database connectivity
- Managing secrets / config

Picking the right execution model



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lambda API



API provided by the Lambda service

Used by all other services that invoke Lambda across all models

Supports sync and async

Can pass any event payload structure you want

Client included in every SDK

Lambda execution model

Synchronous (push)



Amazon API Gateway

/order



Lambda function

Asynchronous (event)



Amazon SNS



Amazon S3



reqs



Lambda function

Stream (Poll-based)



Amazon DynamoDB



Amazon Kinesis



changes

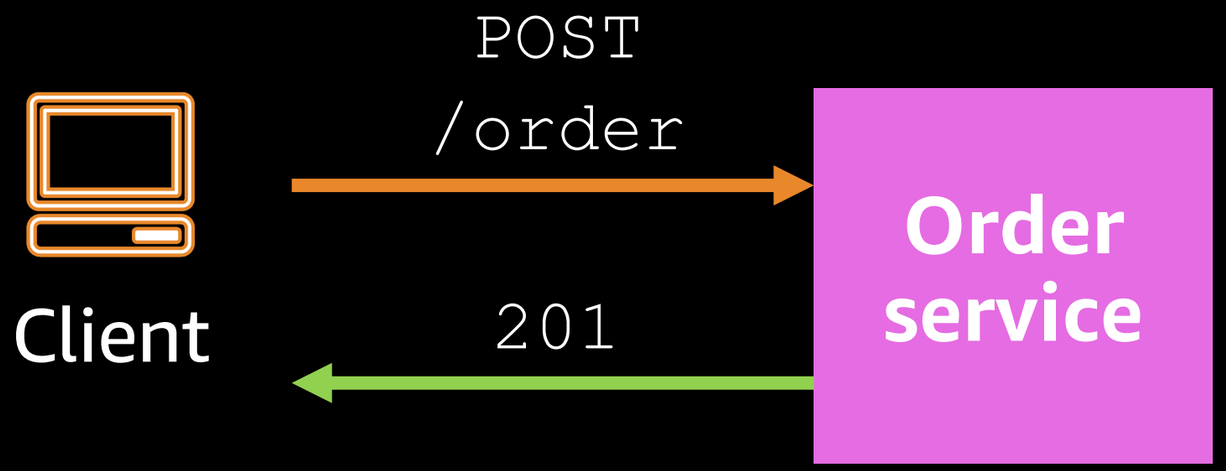


AWS Lambda service

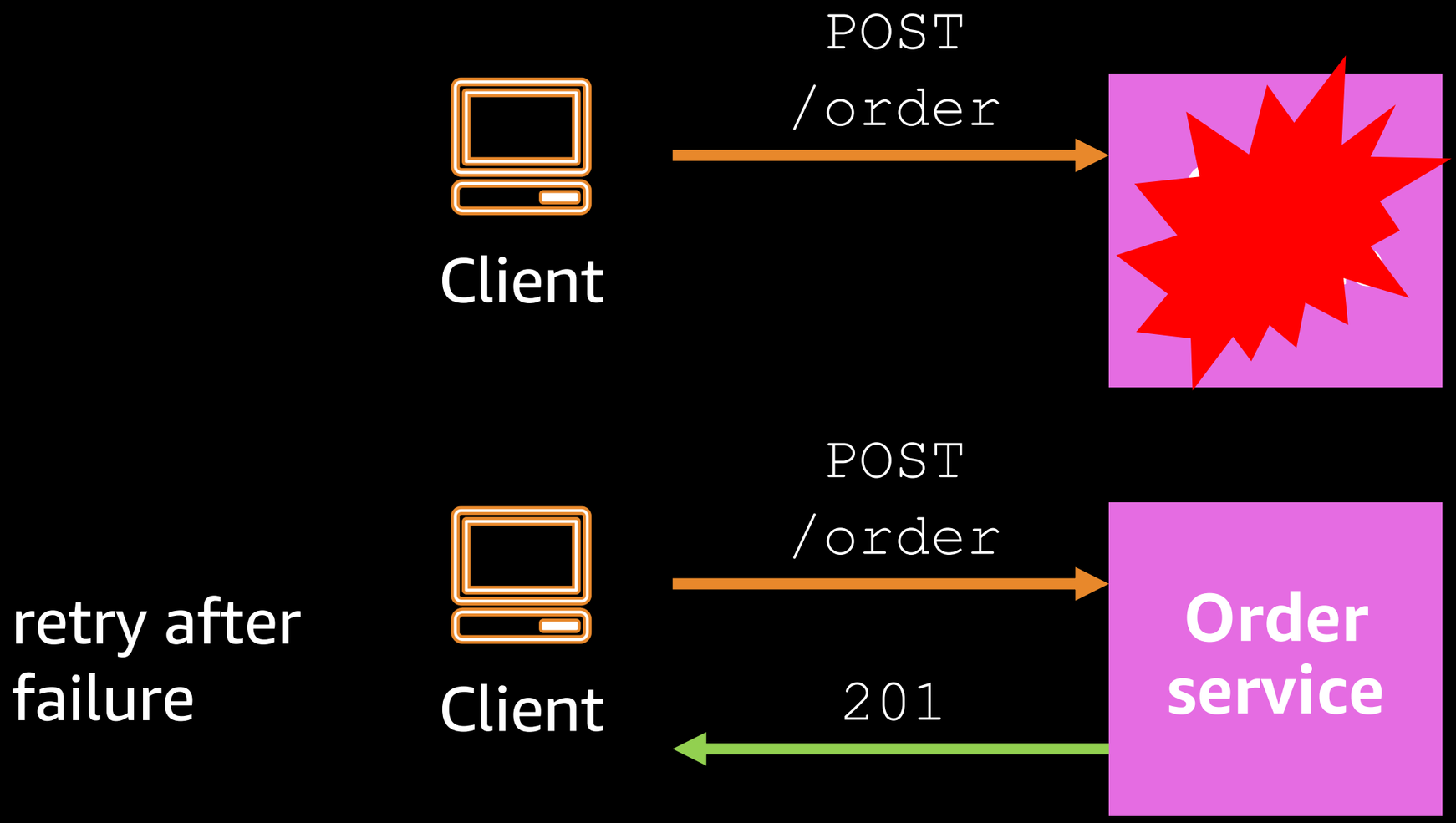


function

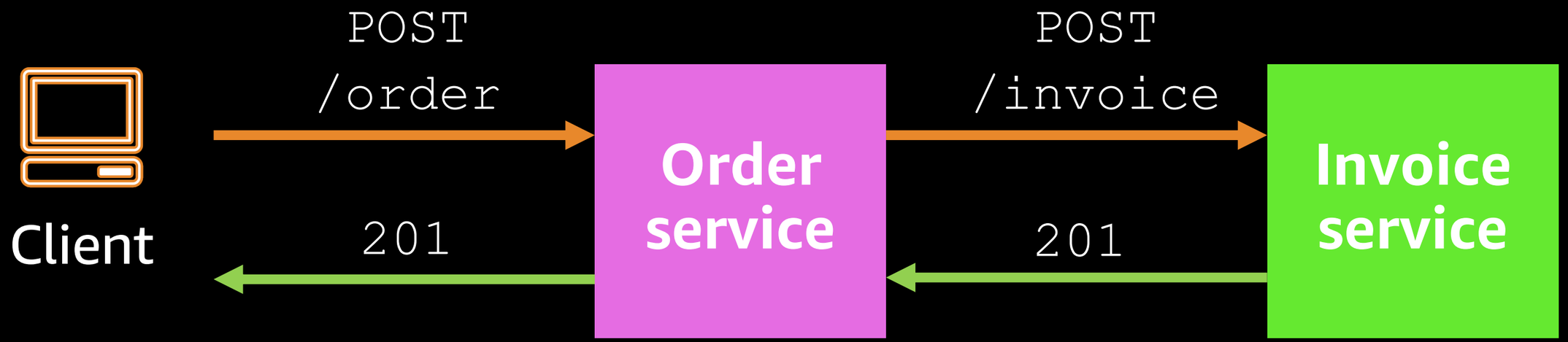
Synchronous APIs



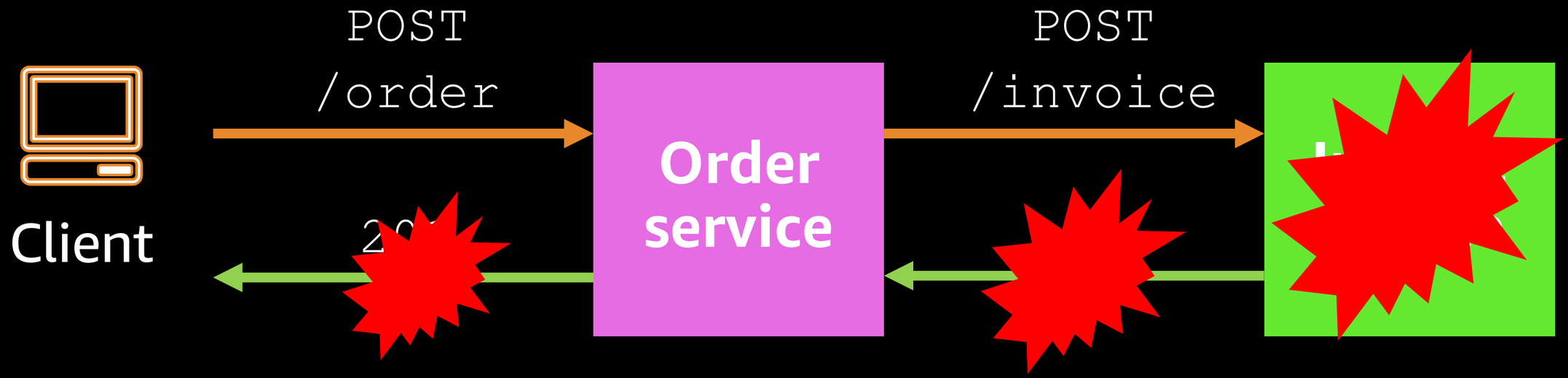
Synchronous APIs



Synchronous APIs



Synchronous APIs

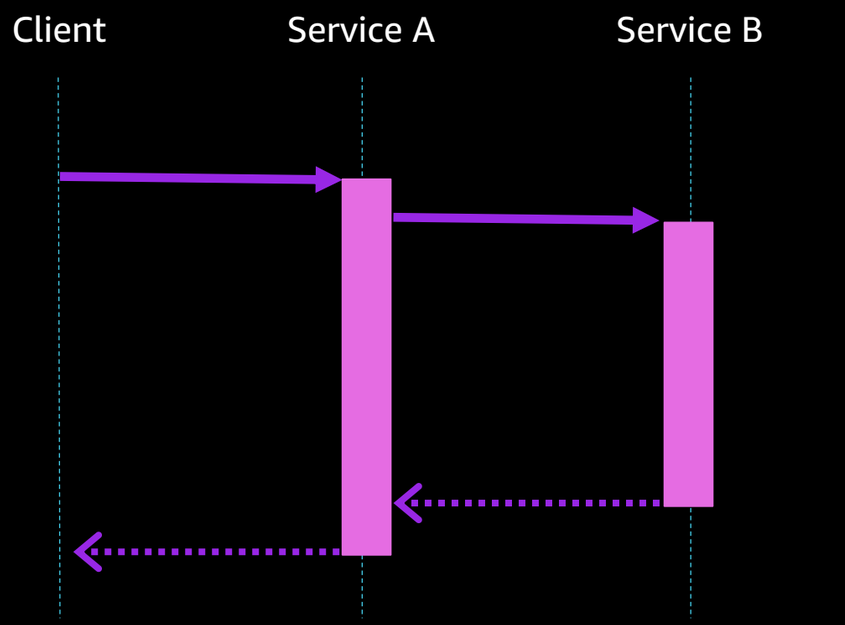


Synchronous APIs

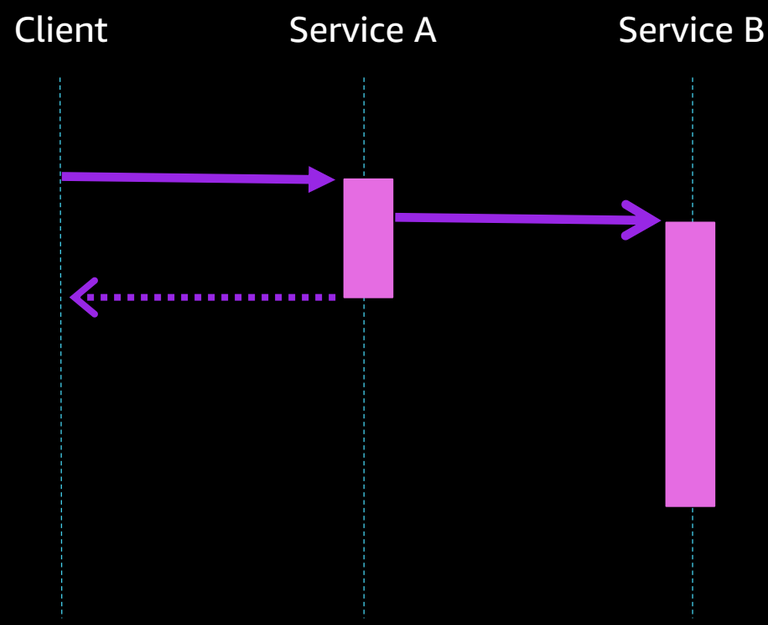


This effectively creates a “tight coupling” where failures become harder to recover from

Thinking asynchronously

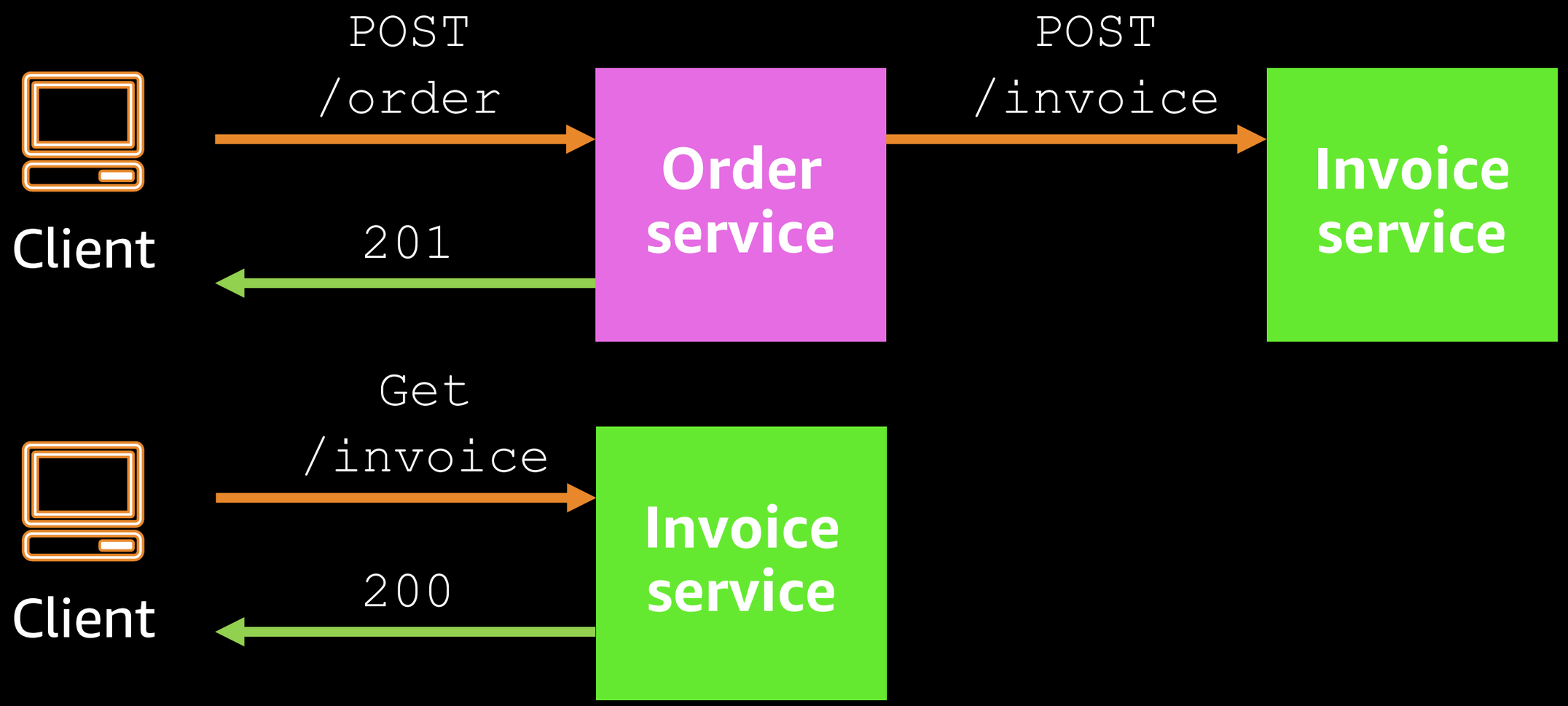


Synchronous commands



Asynchronous events

Asynchronous APIs



If you don’t need a response, use async

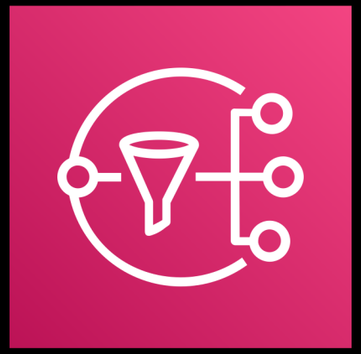
Use the Lambda APIs to start an asynchronous execution

- Built-in queue (Amazon SQS behind the scenes)
- Automatic retries
- Dead letter queue for failed events

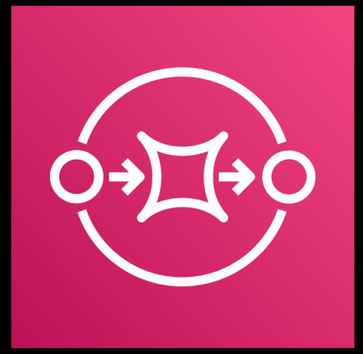
```
client =
boto3.client("lambda")

client.invoke_async(
    FunctionName="test"
    InvokeArgs=json_payload
)
```

Async friends: Topics, streams, queues, and buses



Amazon SNS



Amazon SQS

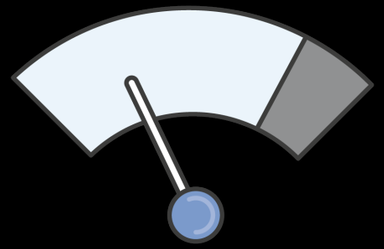


Amazon EventBridge

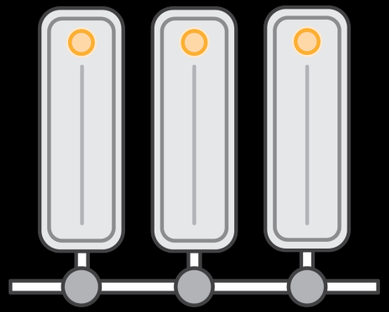


Amazon Kinesis Data Streams

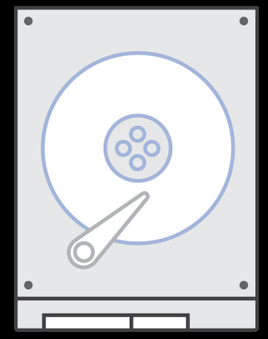
Ways to compare



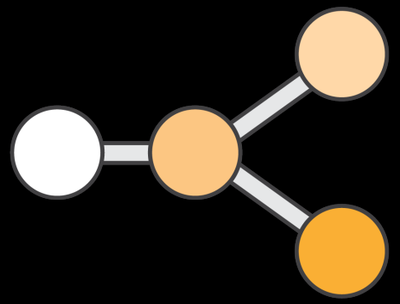
Scale/concurrency controls



Durability



Persistence



Consumption models



Retries



Pricing

Concurrency and you



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Concurrency and you

A single Lambda execution environment can only process a single event at a time

- Regardless of event source or invoke type
- Batches pulled from Amazon Kinesis Data Streams, Amazon SQS, or Amazon DynamoDB Streams count as a single event

Concurrent requests require new execution environments to be created

- Limited in concurrency by burst rate per account per Region

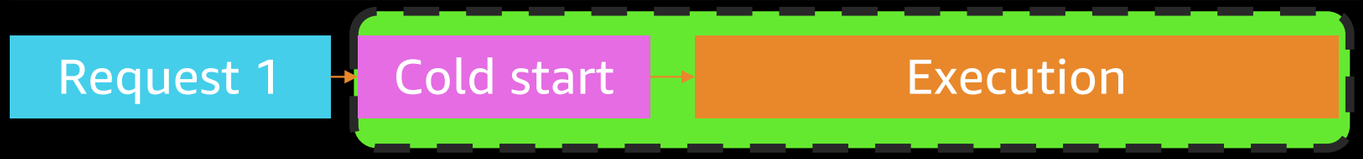
Burst Concurrency Limits

- **3000** – US West (Oregon), US East (N. Virginia), Europe (Ireland).
- **1000** – Asia Pacific (Tokyo), Europe (Frankfurt).
- **500** – Other Regions.

Concurrency and you

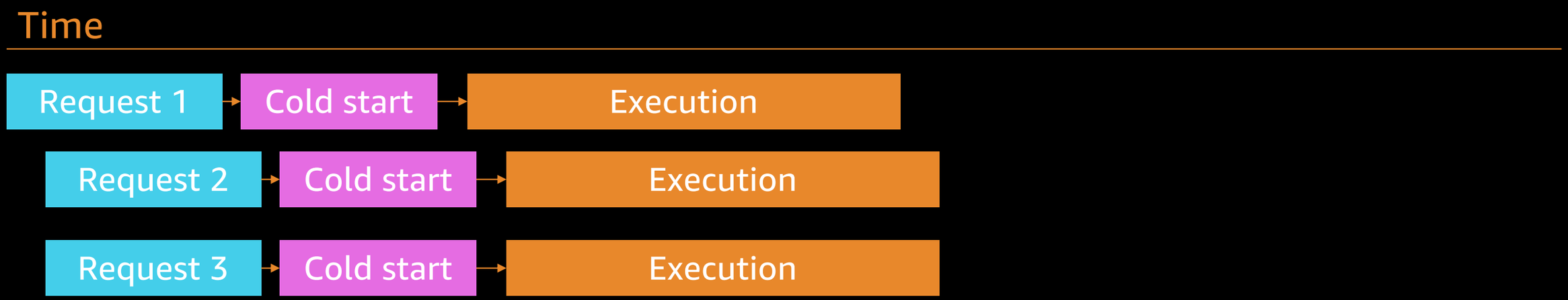


Concurrency and you

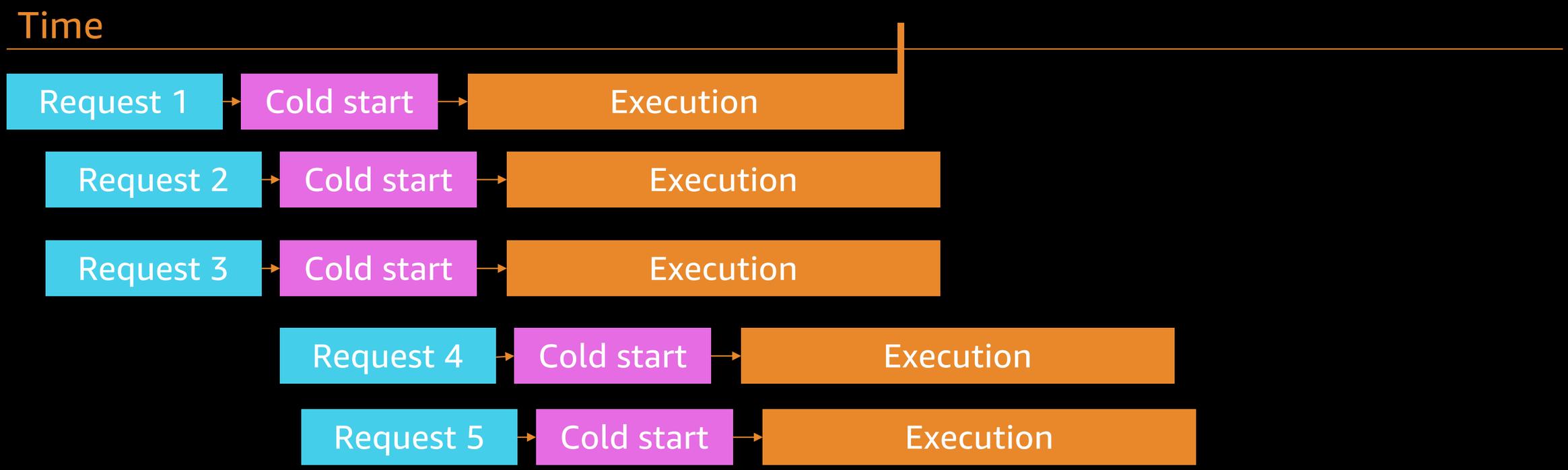


This execution environment is blocked for this entire time

Concurrency and you

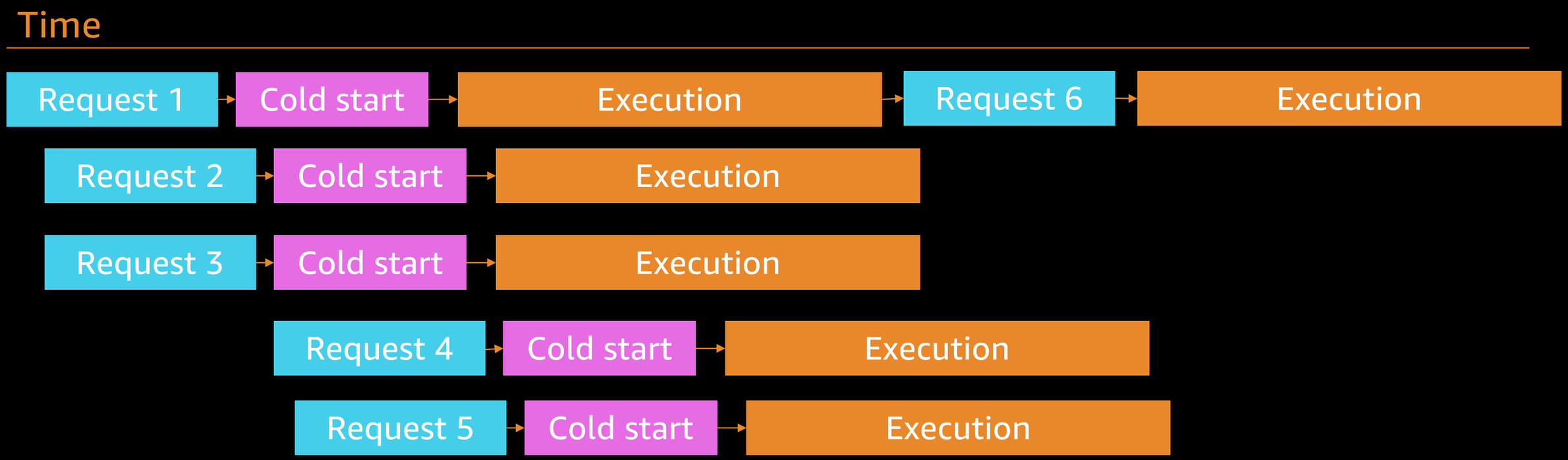


Concurrency and you

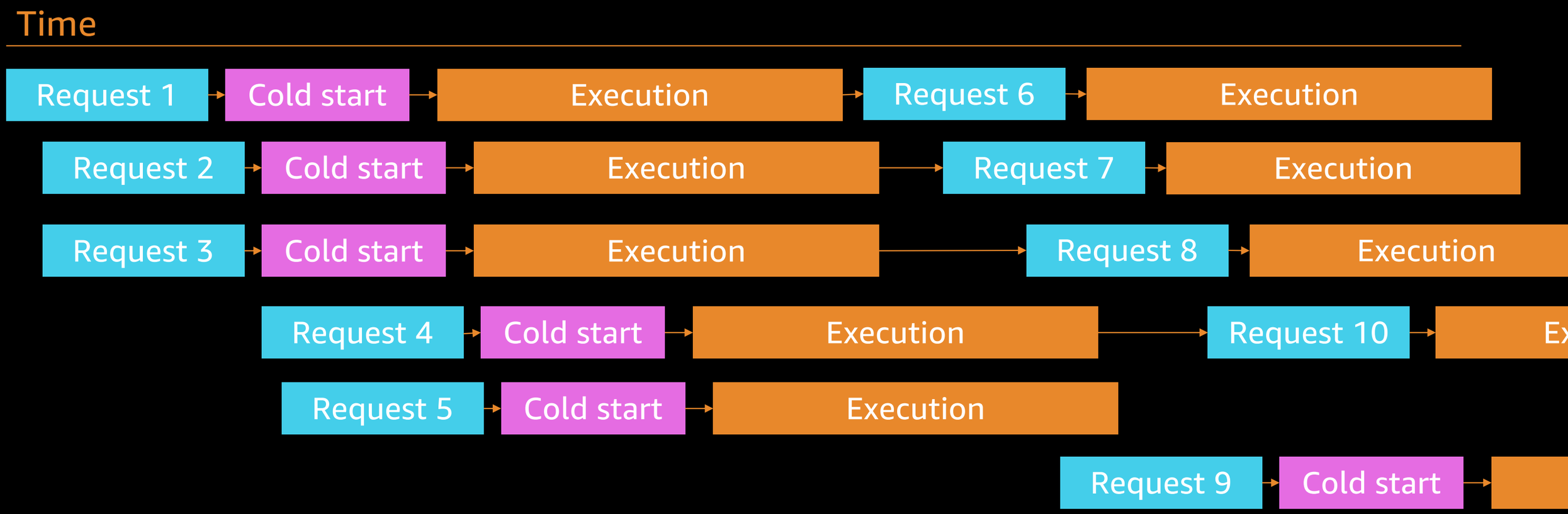


The first environment is now free and warm

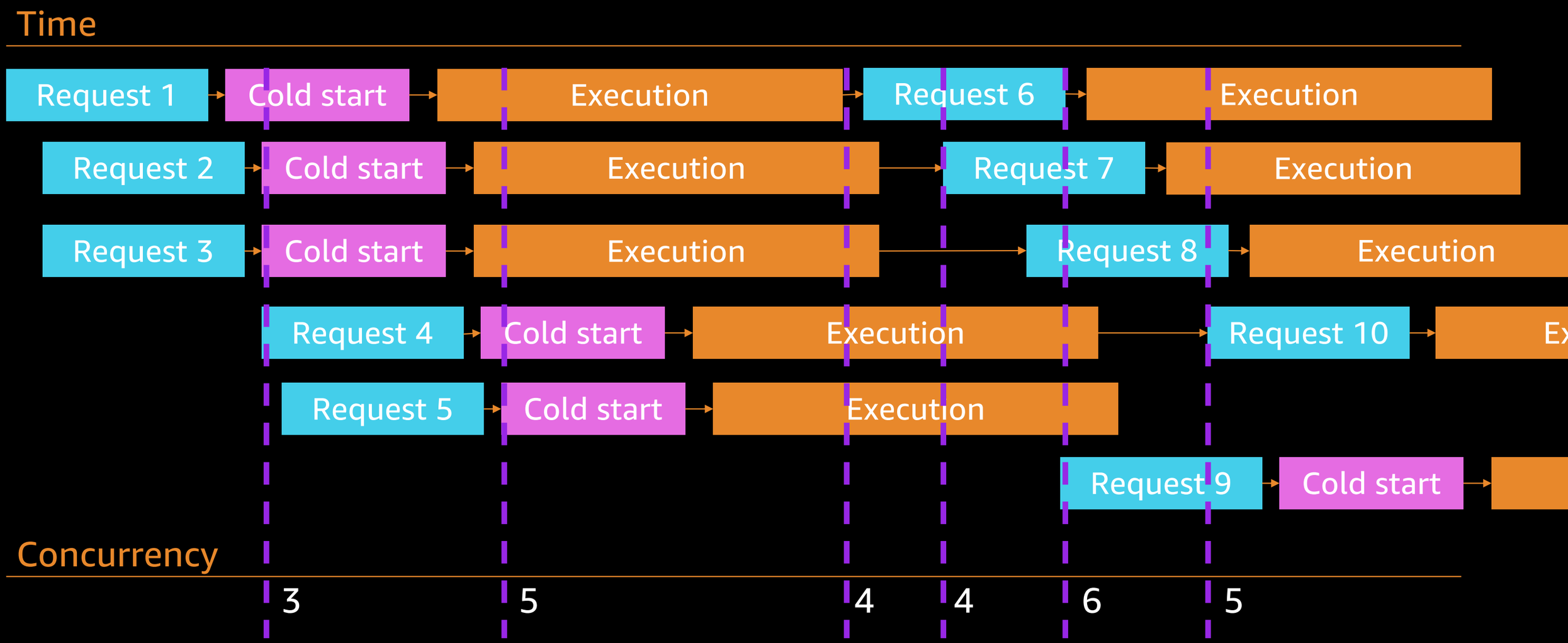
Concurrency and you



Concurrency and you

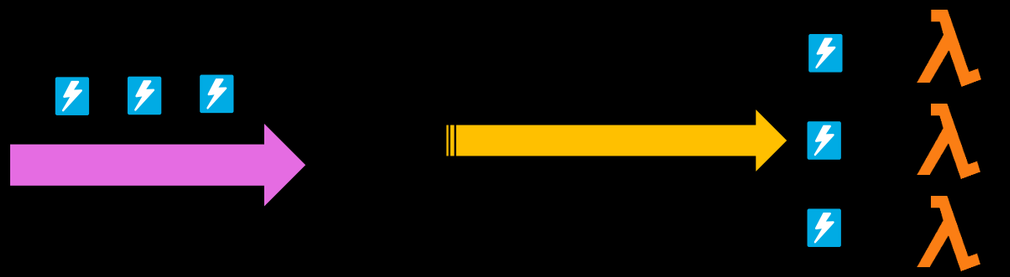


Concurrency and you

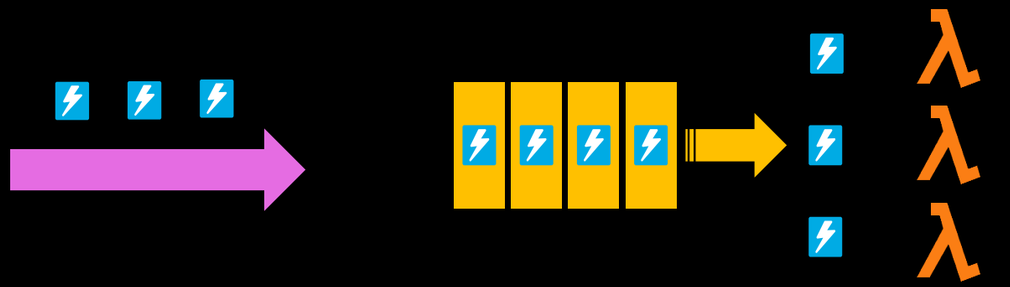


Concurrency across execution models

SNS / API
No event store



Queue based



Stream based



Handling retries and failures



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Handling retries and failures

Sync invocation

- Up to the client/invoking service

Async invocation

- Retries by default (configurable)

Streaming / poll invocation

- Depends on service
  - Amazon SQS – batches retried, can be split and retry failed parts (configurable)
  - Kinesis Data Streams – retried until expiration, can also be split (configurable)

Long story short, you can control how this works now



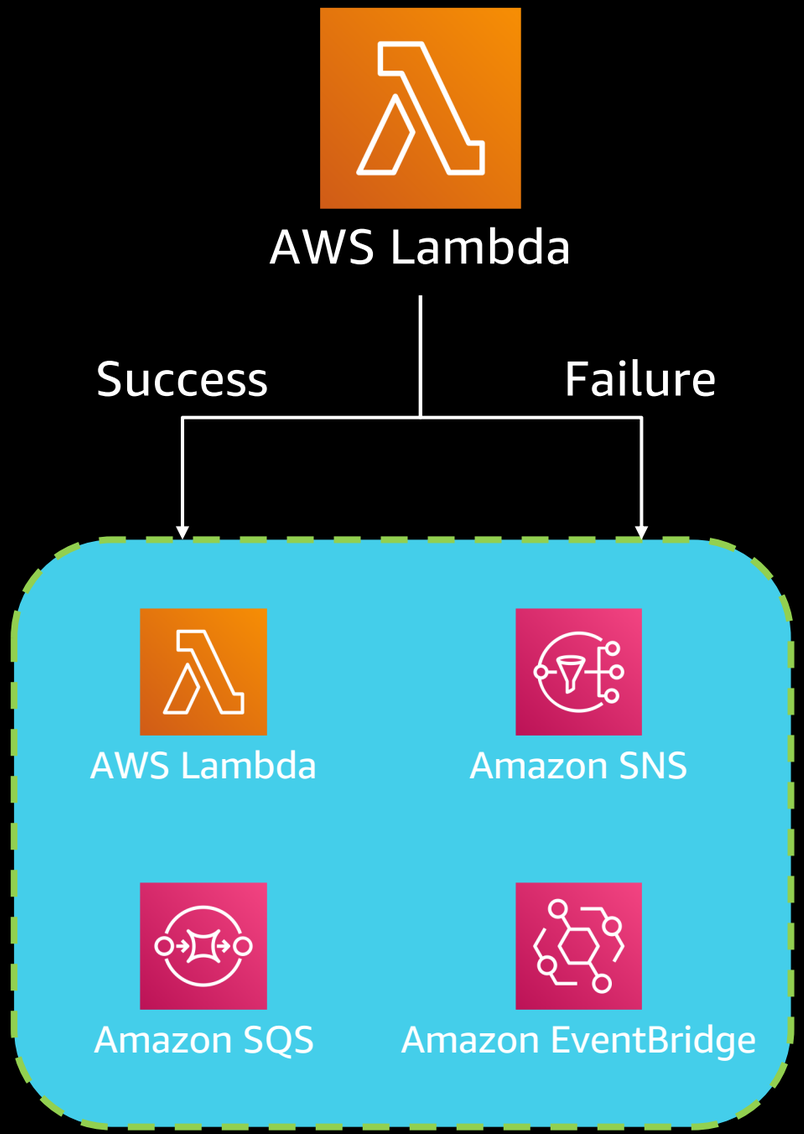
Lambda Destinations

For asynchronous invocations, capture success or failure

- Record contains details about the request and response in JSON format
- Contains more information than data sent to a DLQ
- Can send both outcomes to same destination

  or

- Can send success to one destination, failure to another



Lambda dead letter queues



"By default, a failed Lambda function invoked asynchronously is retried twice, and then the event is discarded."
– https://docs.aws.amazon.com/lambda/latest/dg/dlq.html

- Turn this on! (for async use cases)
- Monitor it via an SQS Queue length metric / alarm
- If you use SNS, send the messages to something durable and / or a trusted endpoint for processing
  - Can send to Lambda functions in other Regions
- If and when things go "**boom**" DLQ can save your invocation event information





In many ways, Lambda Destinations supersedes this

Controlling which events get to Lambda

Read only what you need; for example:

- Path-based routing in API Gateway vs. function logic
- Message filters in Amazon SNS
- Fine-grained rules in Amazon EventBridge
- Query filters in Amazon RDS Aurora
- Use Amazon S3 Select
- Properly indexed databases



Use functions to TRANSFORM, not TRANSPORT

- Use purposeful built services for communication fan-out, message handling, data replication, writing to data stores / databases

Less code > more code



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Integrated services



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Lambda environment variables

- Key-value pairs that you can dynamically pass to your function
- Available via standard environment variable APIs such as process.env for Node.js or os.environ for Python
- Can optionally be encrypted via AWS Key Management Service (AWS KMS)
  - Allows you to specify in AWS Identity and Access Management (IAM) what roles have access to the keys to decrypt the information
- Useful for creating environments per stage (dev, testing, production)



AWS Systems Manager – Parameter Store

Centralized store to manage your configuration data

- Supports hierarchies
- Plaintext or encrypted with AWS KMS
- Can send notifications of changes to Amazon SNS / AWS Lambda
- Can be secured with IAM
- Calls recorded in AWS CloudTrail
- Can be tagged
- Integrated with AWS Secrets Manager
- Available via API / SDK

Useful for centralized environment variables, secrets control, feature flags

```
from __future__ import print_function
import json
import boto3
ssm = boto3.client('ssm', 'us-east-1')

def get_parameters():
    response = ssm.get_parameters(
        Names=['LambdaSecureString'],WithD
ecryption=True
    )
    for parameter in
response['Parameters']:
        return parameter['Value']

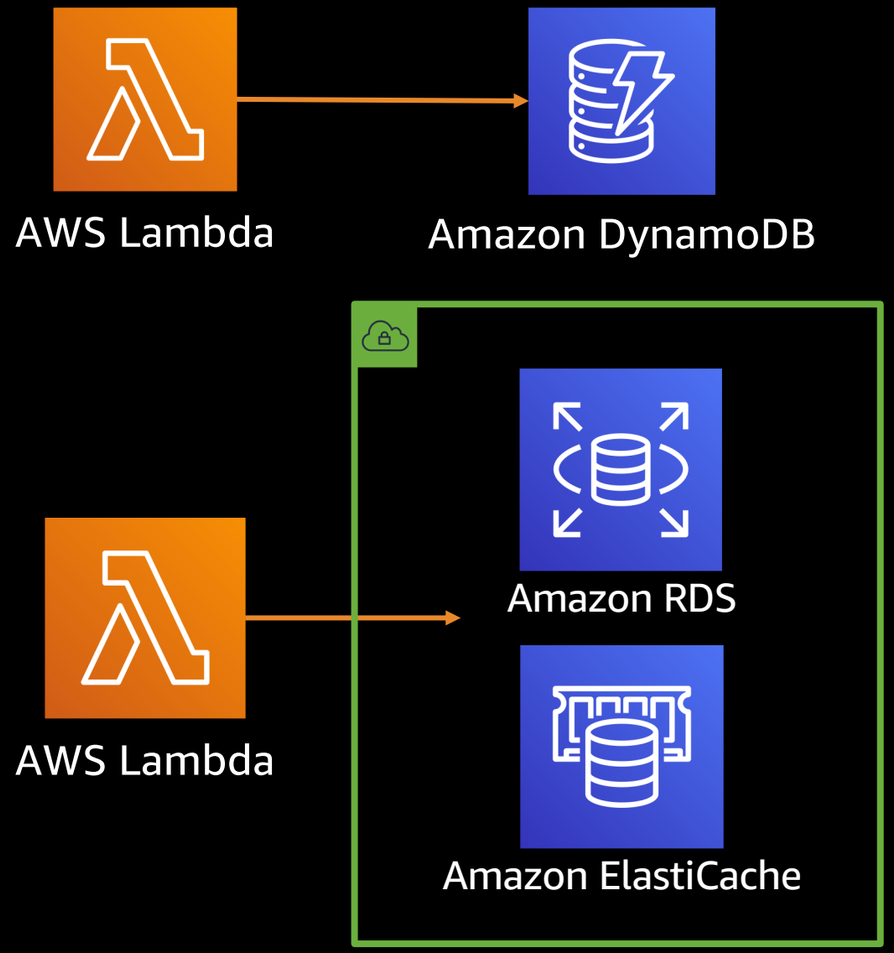
def lambda_handler(event, context):
    value = get_parameters()
    print("value1 = " + value)
    return value  # Echo back the first
key value
```

Database connectivity

General guidance

- If you can use an HTTP-based database such as DynamoDB, resource management is less of an issue
- If you require a more traditional database or cache service, then connection management and overhead become more of a focus

Good news – you can make either work well with Lambda now



Amazon RDS Proxy

- Fully managed, highly available database proxy feature for Amazon RDS
- Pools and shares DB connections to make applications more scalable, resilient to database failures, and secure
- For MySQL and PostgreSQL



Pool and share DB connections for improved app scaling



Increase app availability and reduce DB failover times

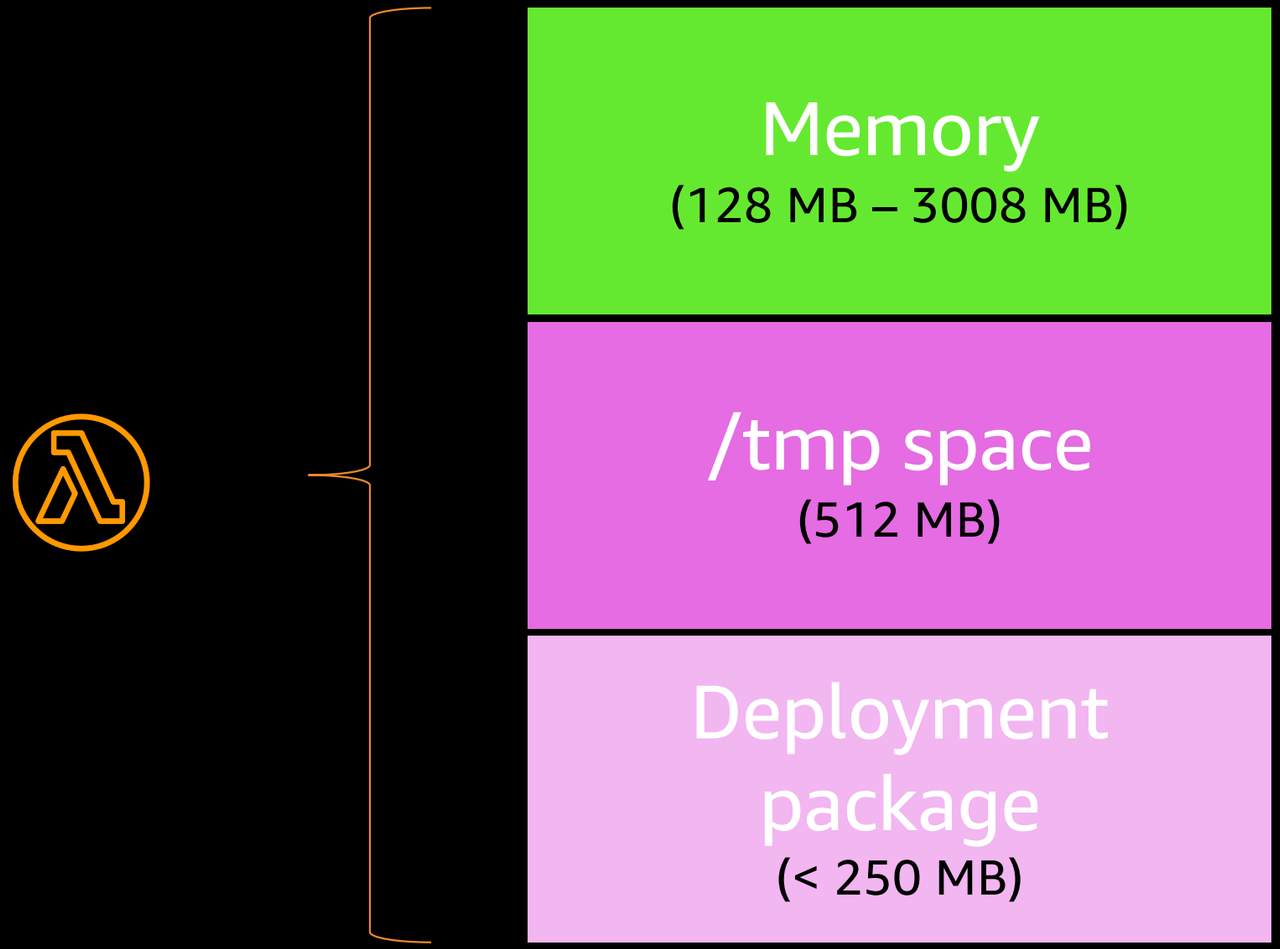


Manage app data security with DB access controls

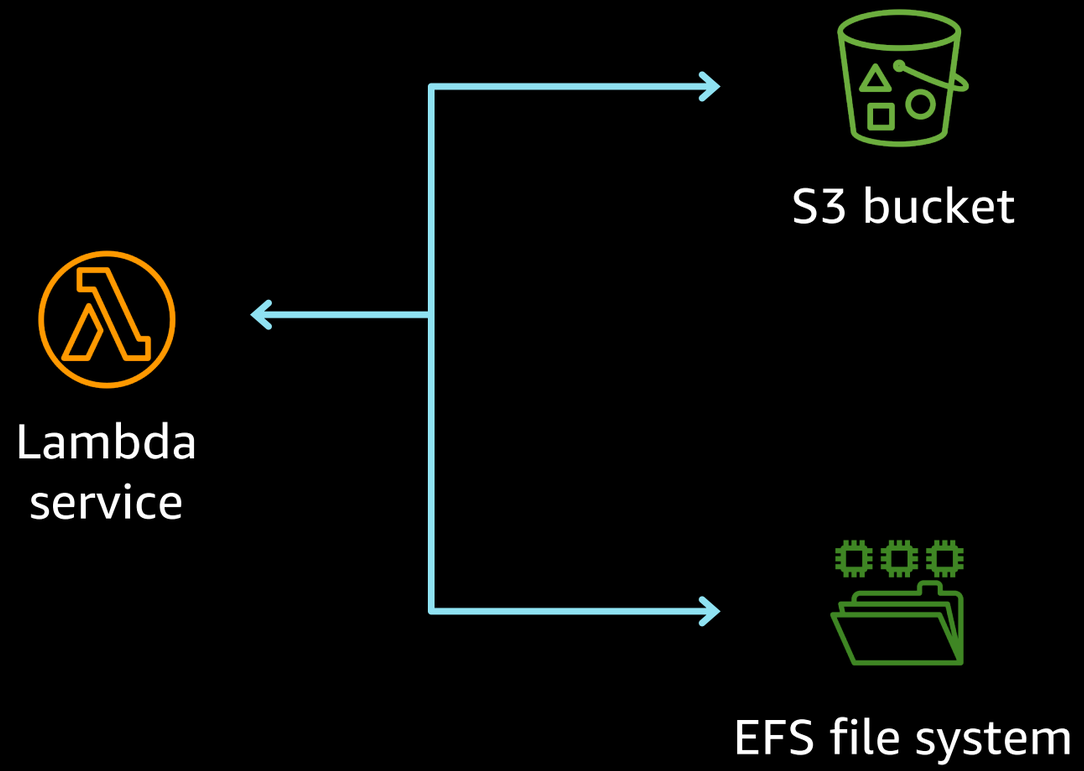


Fully managed DB proxy, compatible with your database

Storage in Lambda

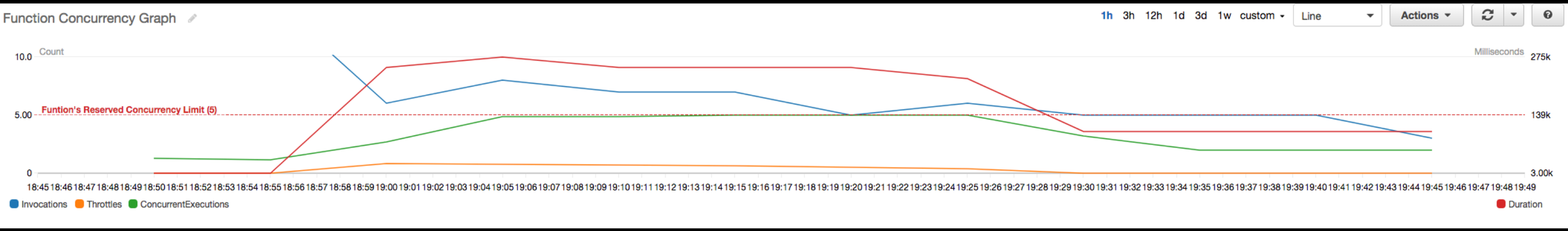


Lambda external storage options

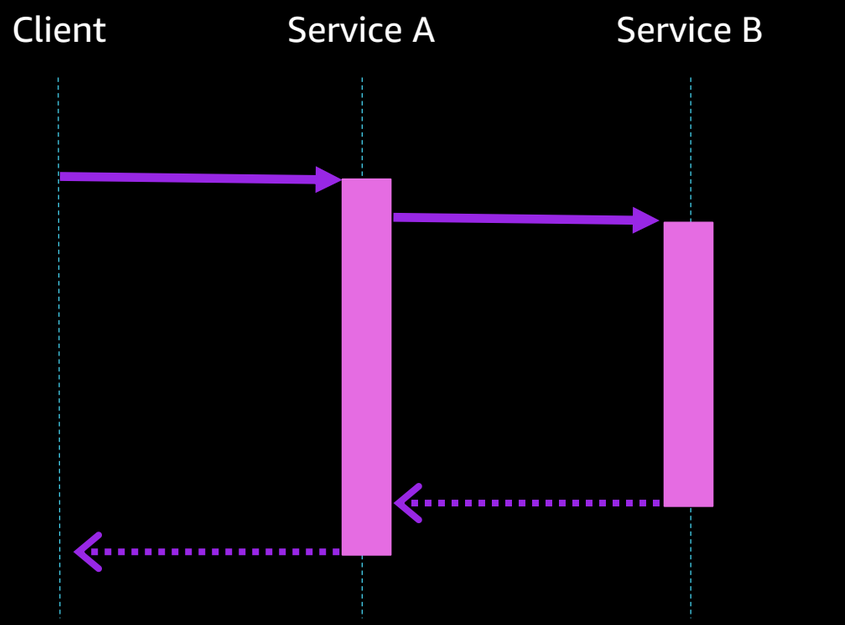


Lambda per function concurrency controls

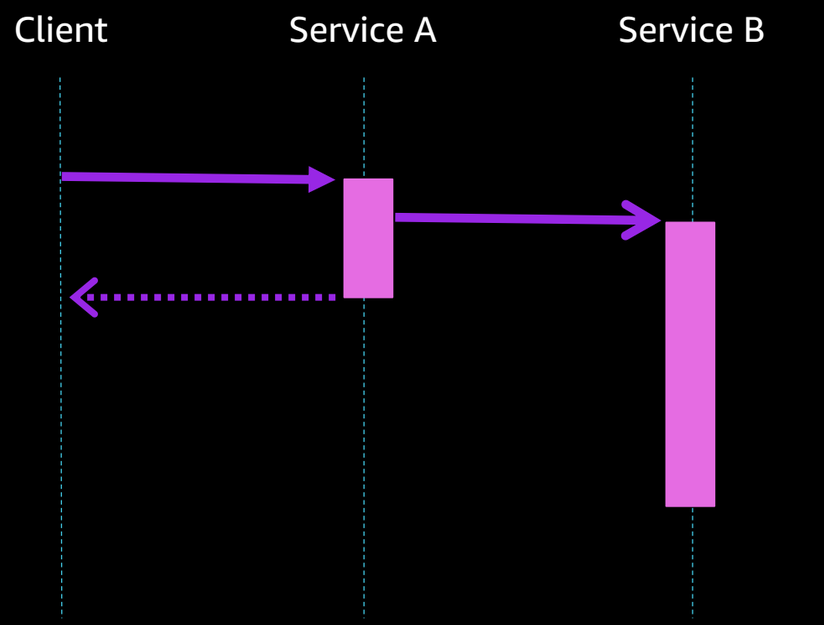
- Concurrency a shared pool by default
- Separate using per function concurrency settings
    - Acts as reservation
- Also acts as max concurrency per function
    - Especially critical for downstream resources like databases
- “Off switch” – set per function concurrency to zero



Thinking asynchronously



Synchronous commands



Asynchronous events

When integrating with other services from AWS Lambda

- Reduce need for synchronous communication
- Cache what you can in the function but have a mechanism for flushing that cache / updating it regularly
- Pay attention to how concurrency might impact consumption of resources
- Small, more fine-grained functions require less resources and have reduced “weight” on other services
- Pay attention to the limits of other services, as they can often be smaller than what Lambda can do in terms of both requests per second (RPS) but also latency they incur

FIN / ACK: Two focus areas for optimization

On the invoke side

- Using the right execution model
- Concurrency and you
- Handling retries and failures

On the integrated services side

- Lambda storage options
- Database connectivity
- Managing secrets / config



So much more here:

https://s12d.com/svs403

Thank you!

Chris Munns

munns@amazon.com
@chrismunns

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.