
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

CON309

Build your application easily & efficiently with serverless containers

Nathan Peck (he/him)
Senior Developer Advocate
AWS

Omar Paul (he/him)
Senior Technical Product Manager
AWS

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

As a software builder, you need the right tools at the right time



Completely new software needs rapid development of new features

Established software needs maintenance, support, and reliability

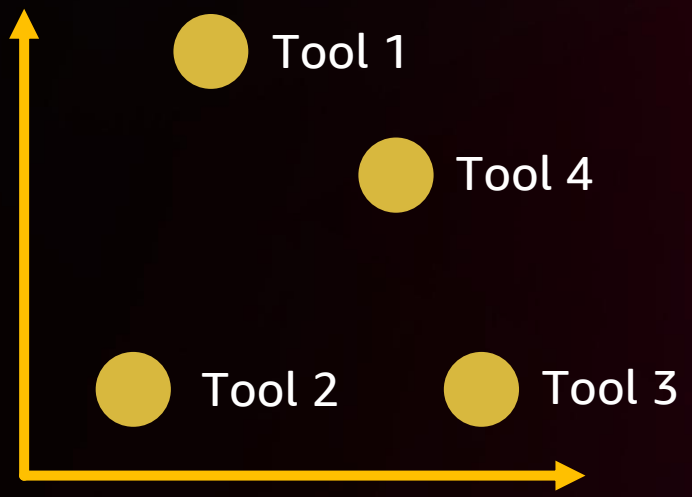
aws

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

Choosing the right tool at the right time requires looking at two main dimensions

Ease

Does this tool help me build better and faster? Or does it get in my way when I'm trying to build?



Efficiency

Is this tool's pricing, compute model, or optimization appropriate for what I'm trying to do with it?

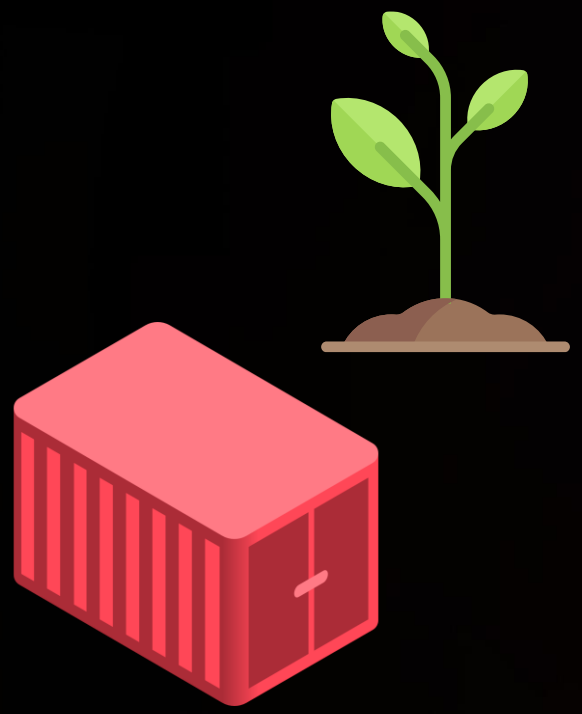
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Containers help with both ease and efficiency

From greenfield project to established application

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

For greenfield projects



Start new projects faster from prebuilt production-ready base container images

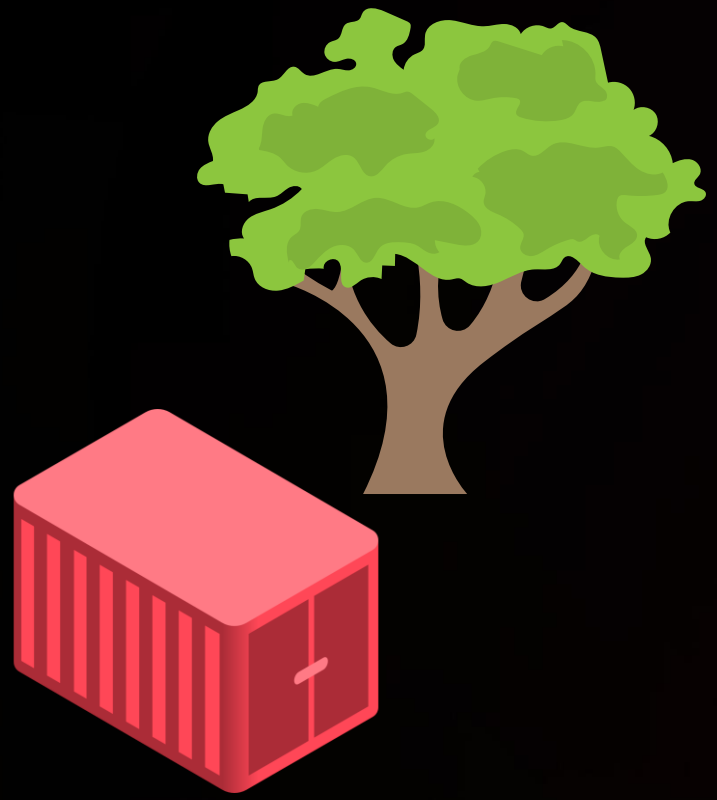
No need to build your own tooling for build and deploy

Easier local development experience, less time wasted on setting up developer environment tooling

aws

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

For established software



Standard application packaging format which works for all services and microservices; fewer “special snowflakes”

Pack applications onto infrastructure more efficiently without conflicts

Reliable and reproducible deployments that you can roll back with ease

aws

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

Modern compute efficiency has concurrency at its root

Computers do more things, more efficiently, with more concurrency

aws

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



Photo by Jud McCranie / CC BY-SA 4.0

Tandy 1000

No concurrency! If you wanted to run a different program, you had to turn the computer off, put a new disk in, and turn it back on

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

Cooperative multitasking

The foreground program is responsible for checking periodically to see if another program is asking for CPU; it can optionally yield to background programs





Photo by Jud McCranie / CC BY-SA 4.0



aws

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

Preemptive multitasking

The operating system can proactively suspend a program and schedules which program should get a CPU time slice next



Photo by Jud McCranie / CC BY-SA 4.0







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

Multicore processors

Multiple programs truly running at once on different cores of the same processor



Photo by Jud McCranie / CC BY-SA 4.0

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

Client/server architecture

Programs communicate over the internet to server clusters where work is done across many multicore computers

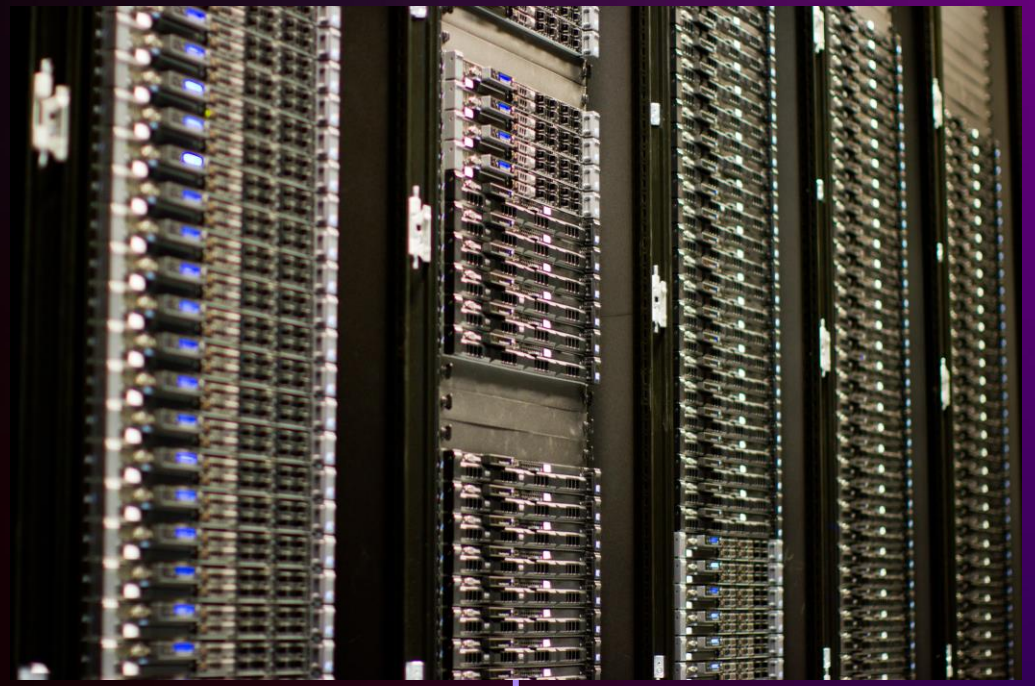


Photo by Victorgrigas / CC BY-SA 3.0



Photo by Jud McCranie / CC BY-SA 4.0

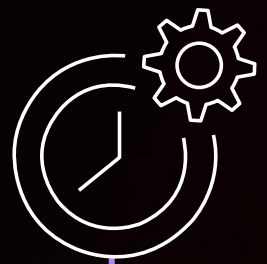
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Concurrency keeps rising exponentially as technology improves



Photo by Jud McCranie / CC BY-SA 4.0





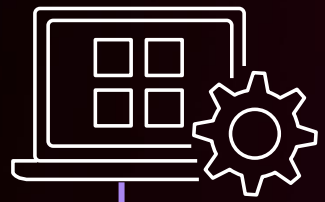




Photo by Victorgrigas / CC BY-SA 3.0

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

How do I build an application that can handle more concurrency, more efficiently?

From zero users to millions

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

Builders need compute optimized for low concurrency and compute optimized for high concurrency

Low-concurrency applications

Still searching for product market fit. Need easy development, low operational costs, low baseline costs.



High-concurrency applications

Wildly successful, viral adoption. Need large amounts of compute at cheap sustained-usage rates.

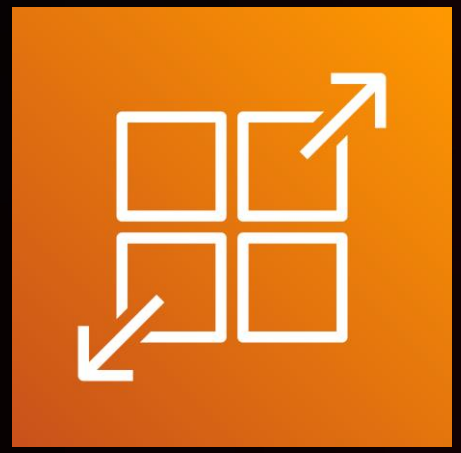
aws

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

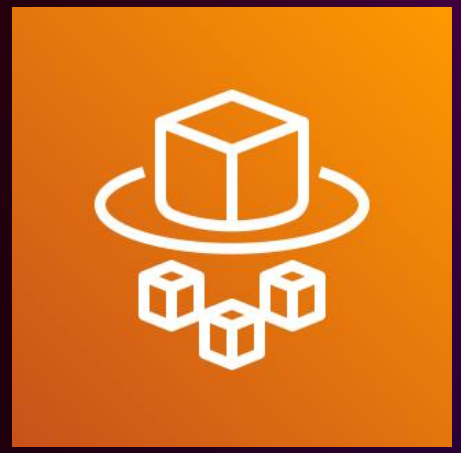
Serverless container technologies help you start out small and scale out when needed



AWS Lambda



AWS App Runner



AWS Fargate

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



AWS Lambda

Your containerized event handling function in the cloud

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

AWS Lambda runs your function in the cloud

Handler function

```
exports.handler = (event, context, callback) => {
  callback(null, "Hello world!");
}
```

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

Different ways to run the code: Events



AWS Lambda

Invoke code by calling the AWS Lambda API

Handler function

```
exports.handler = (event, context, callback) => {
  callback(null, "Hello world!");
}
```



Amazon EventBridge

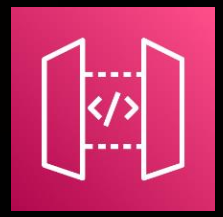
Invoke code when a specific event happens – upload to an S3 bucket, for example

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

Different ways to run the code: Web requests



Application Load Balancer



Amazon API Gateway



Lambda function URL

Handler function

```
exports.handler = (event, context, callback) => {
  callback(null, "Hello world!");
}
```

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

Firecracker microVM

Handler function

```
exports.handler = (event, context, callback) => {
  callback(null, "Hello world!");
}
```

Firecracker microVM

Handler function

```
exports.handler = (event, context, callback) => {
  callback(null, "Hello world!");
}
```

Lambda spins up multiple isolated copies of your containerized code in microVMs that are strongly isolated from each other

aws

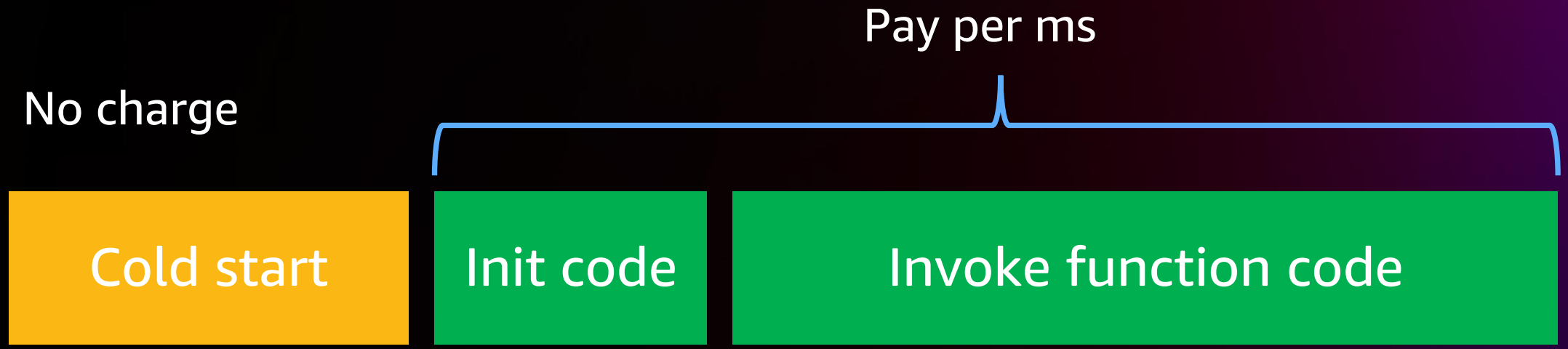
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lifecycle of a Lambda function instance

Cold start	Init code	Invoke function code
Downloading your container, extracting it, and loading the code for launch in its own isolated runtime sandbox	Any setup code outside your handler function	Running your handler code, waiting for it to respond back with a result

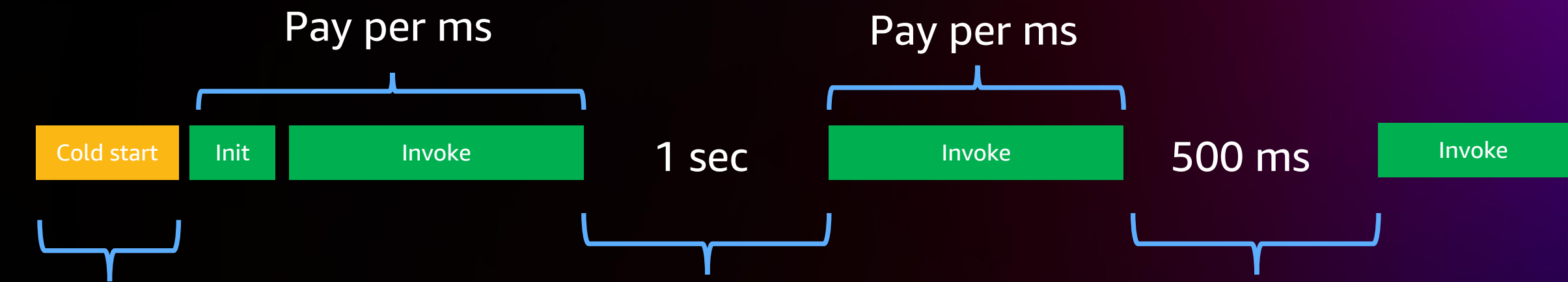
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

You pay per millisecond for time spent in init code and invoking the handler function



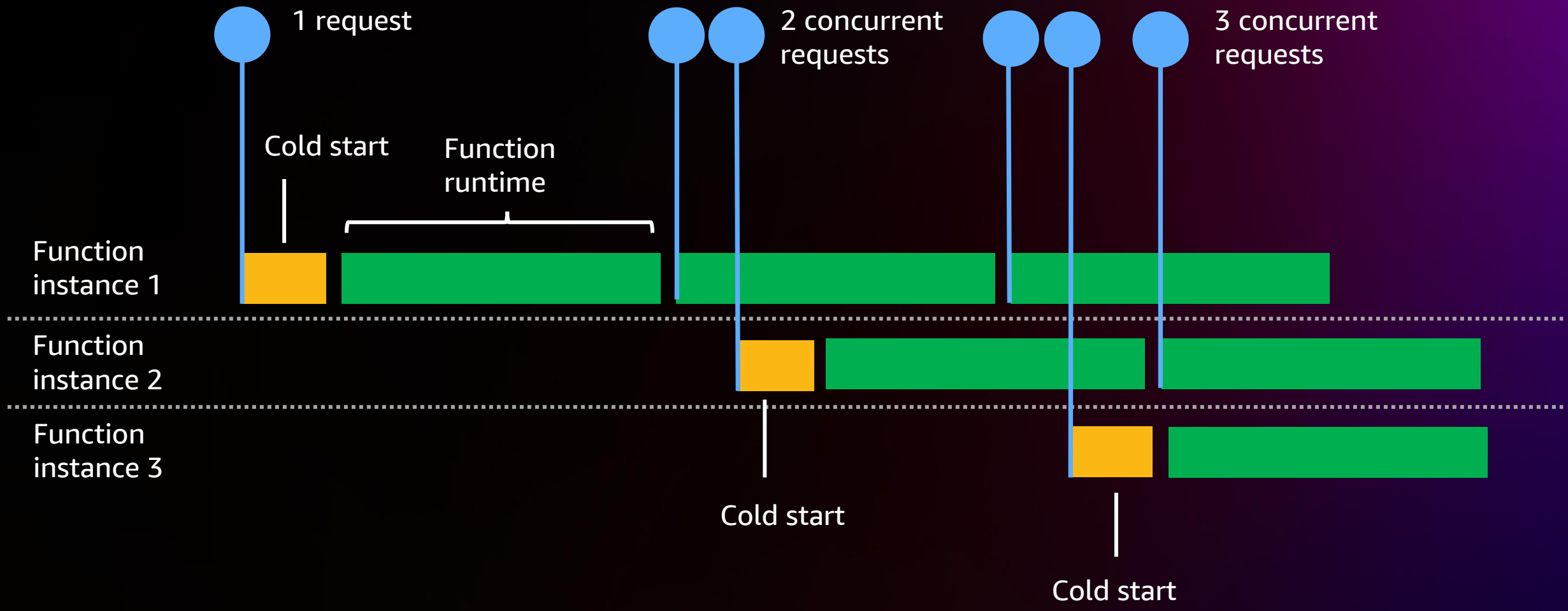
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS Lambda is extremely cost effective when there are gaps between work to do



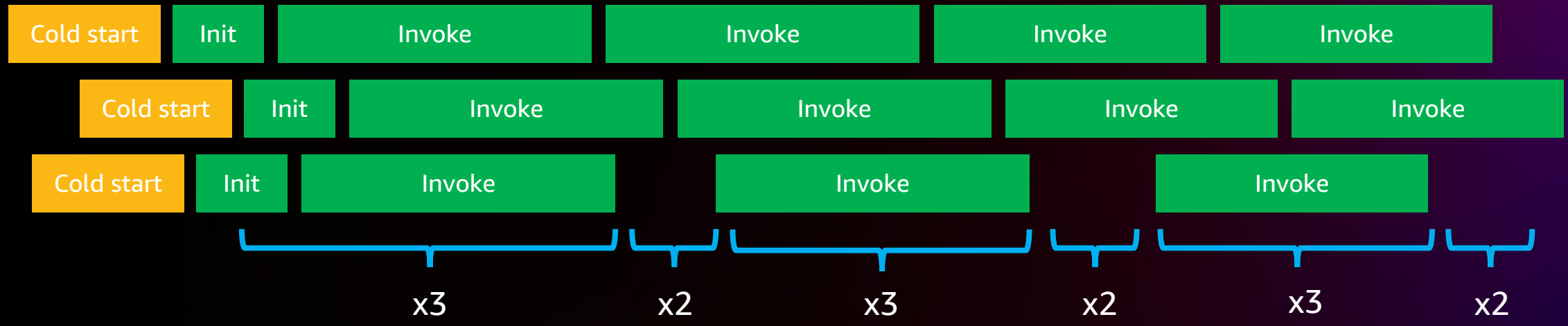
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

A function instance only handles one invoke at a time



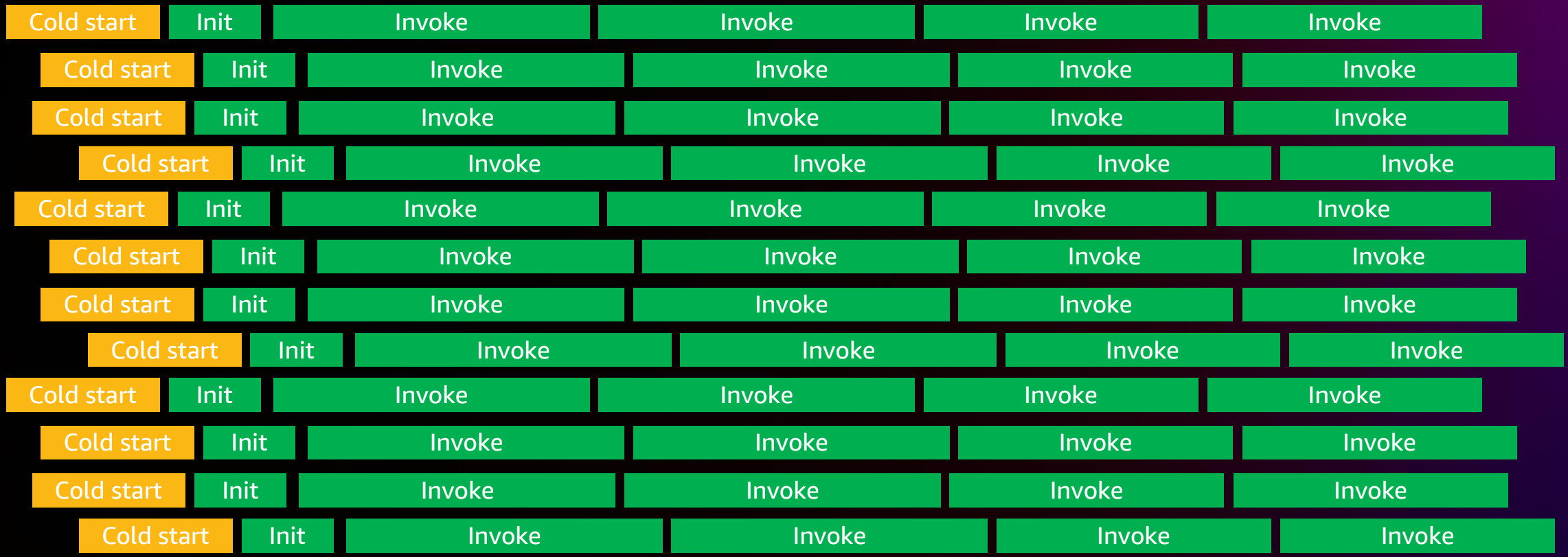
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

When there are multiple concurrent Lambda function instances running, each one charges concurrently



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

What about when there are many concurrent Lambda functions?



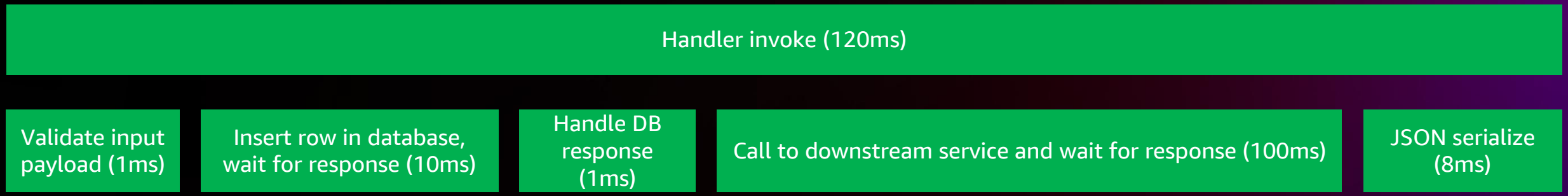
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Concurrency deep dive

What is actually happening during a request?

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

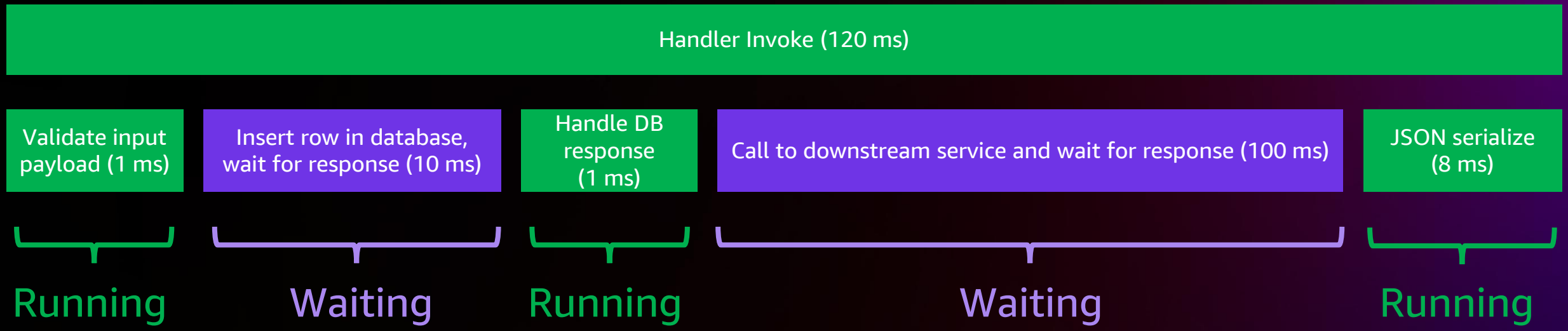
What is my application actually doing?



Many modern workloads are full of Input/Output (I/O) operations

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

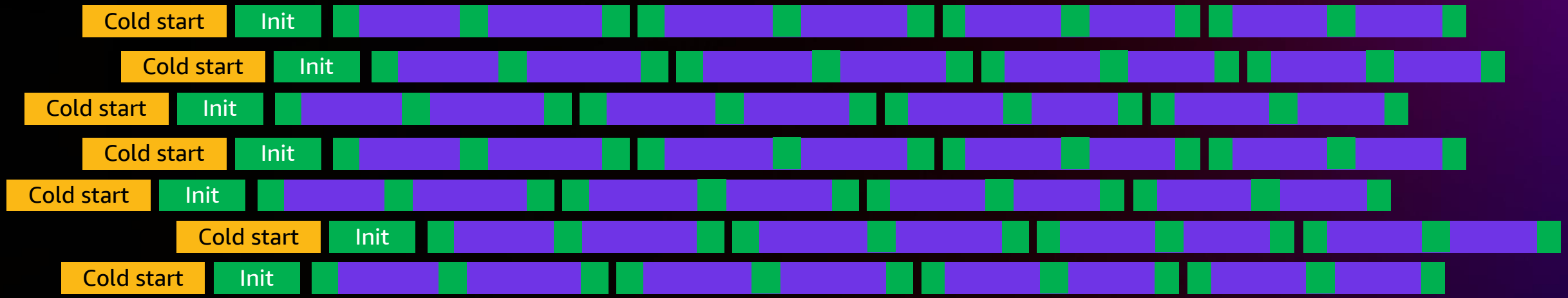
What is my application actually doing?



I/O heavy workloads often spend more time waiting than running CPU instructions

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

The more concurrent activity I have, the more milliseconds of waiting there are compared to milliseconds of running CPU instructions



What alternatives are there?

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

Modern application frameworks make it possible to handle multiple concurrent requests in a single application process



Validate input payload (1ms)

Handle DB response (1ms)

JSON serialization (8ms)

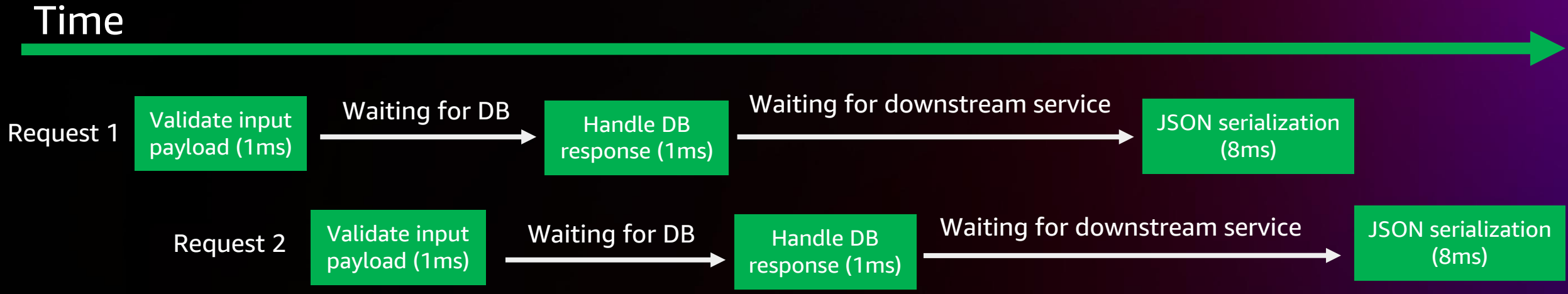
Each second has 1000ms for a CPU core to do work

Grab more work off the event loop whenever we are waiting on I/O

aws

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

Instead of doing nothing while waiting, grab another event and do some work while we wait



The event loop schedules work into all 1000ms of time per second

A single application process can keep the CPU core busy by doing other work (like answering another HTTP request) while it waits on I/O

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

Node.js async/await

The await keyword yields back to the event loop to let the process start handling another request while it waits for the I/O to complete

```
exports.handler = async function (event, context, callback) {
  // Validate the event

  let dbResponse = await db.insert(); // Wait for IO from DB

  // Handle DB response

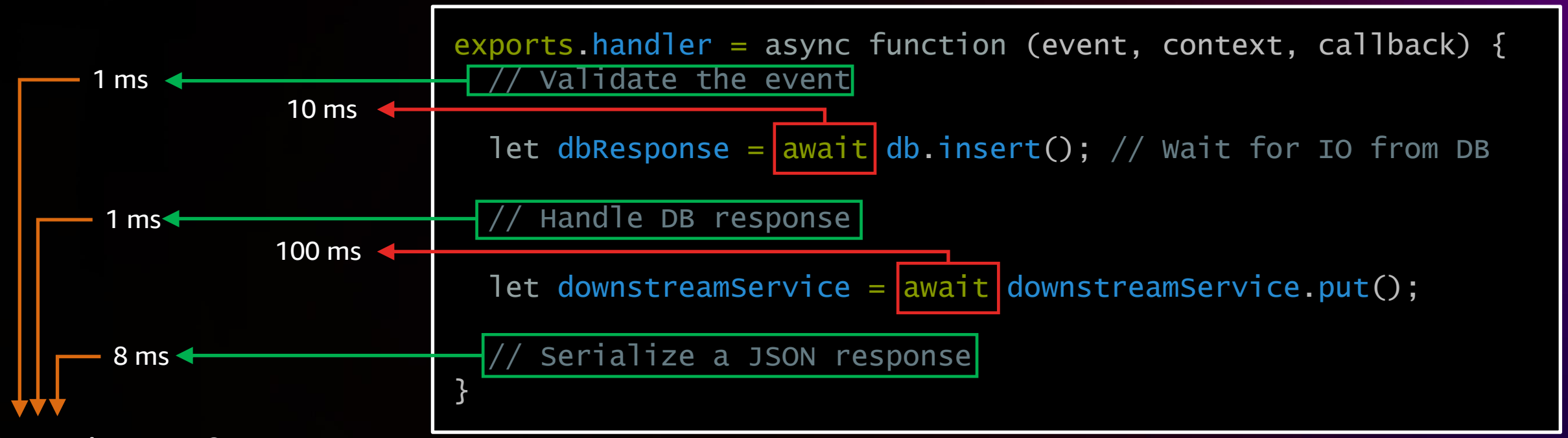
  let downstreamResponse = await downstreamService.put();

  // Serialize a JSON response
}
```

aws

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

The amount of possible concurrency for a function depends on the CPU time consumed by its instructions



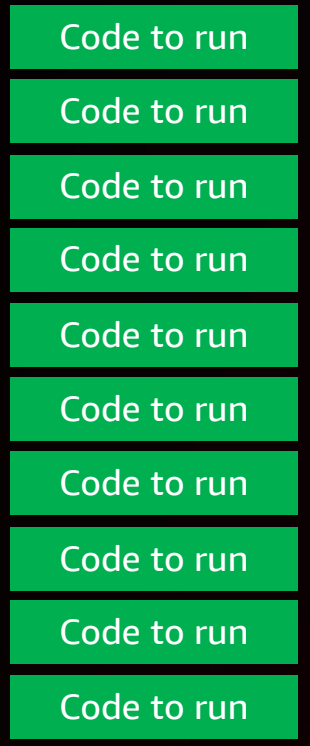
CPU time: 10 ms

1,000 ms / 10 ms = 100 function invokes/sec per single threaded app process

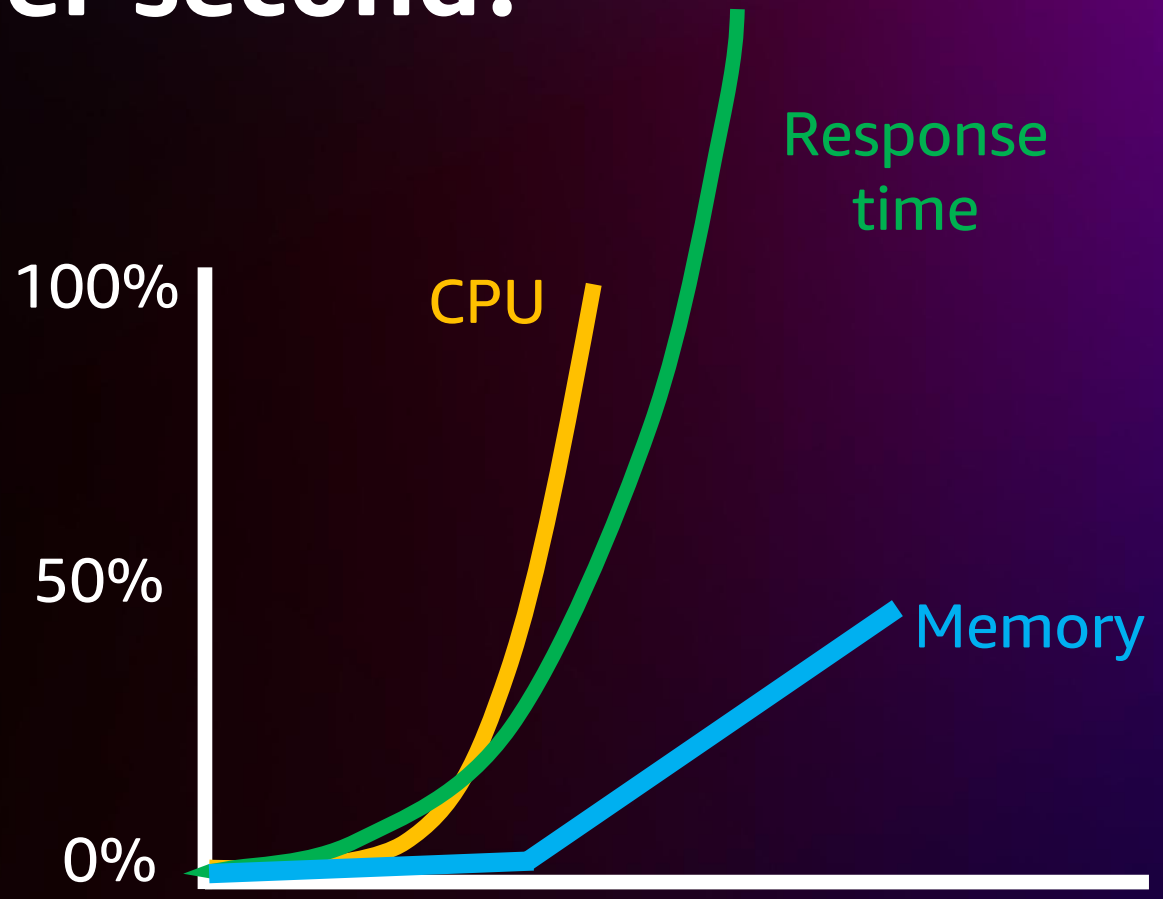
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Downside of concurrency: What happens if you have too many requests per second?





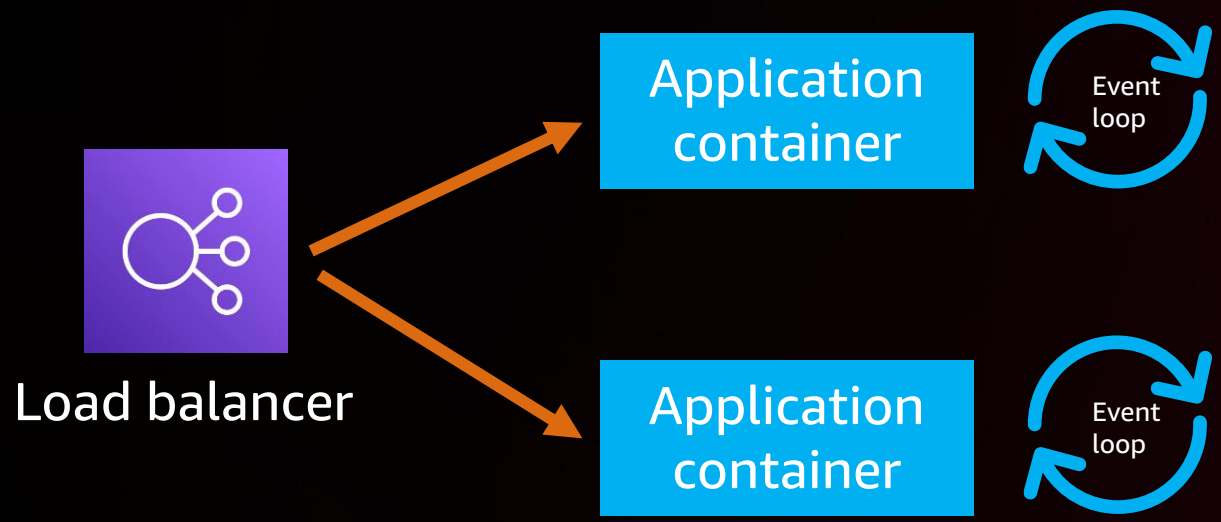
Work starts queueing up



As CPU saturates,
the response time skyrockets

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

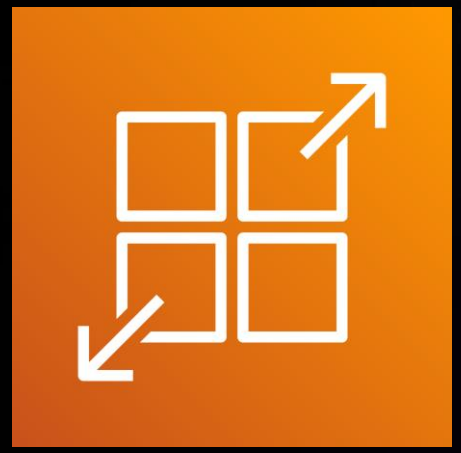
Solution: Load balance concurrency across multiple copies of the application process, each with its own event loop



Processes could be both running on the same physical machine and sharing separate cores of a multicore machine.

Or they could be on their own machines.

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



AWS App Runner

Each copy of your application
handles many concurrent requests

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

AWS App Runner manages horizontal scaling for you

AWS = Fully managed by AWS
You = Your responsibility to manage



Client code (You)



Ingress and load balancing (AWS)

Scaling (AWS)

MicroVM (AWS)

Application container

Event loop

Application code (You)

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

You



Your client side code sends requests to the endpoint for an AWS App Runner service

AWS

Envoy Proxy load balancer

The endpoint goes to a fully AWS managed Envoy Proxy load balancer

Overflow queue

Concurrent requests limit

Each instance of your application container has its own concurrency limit and overflow queue

You

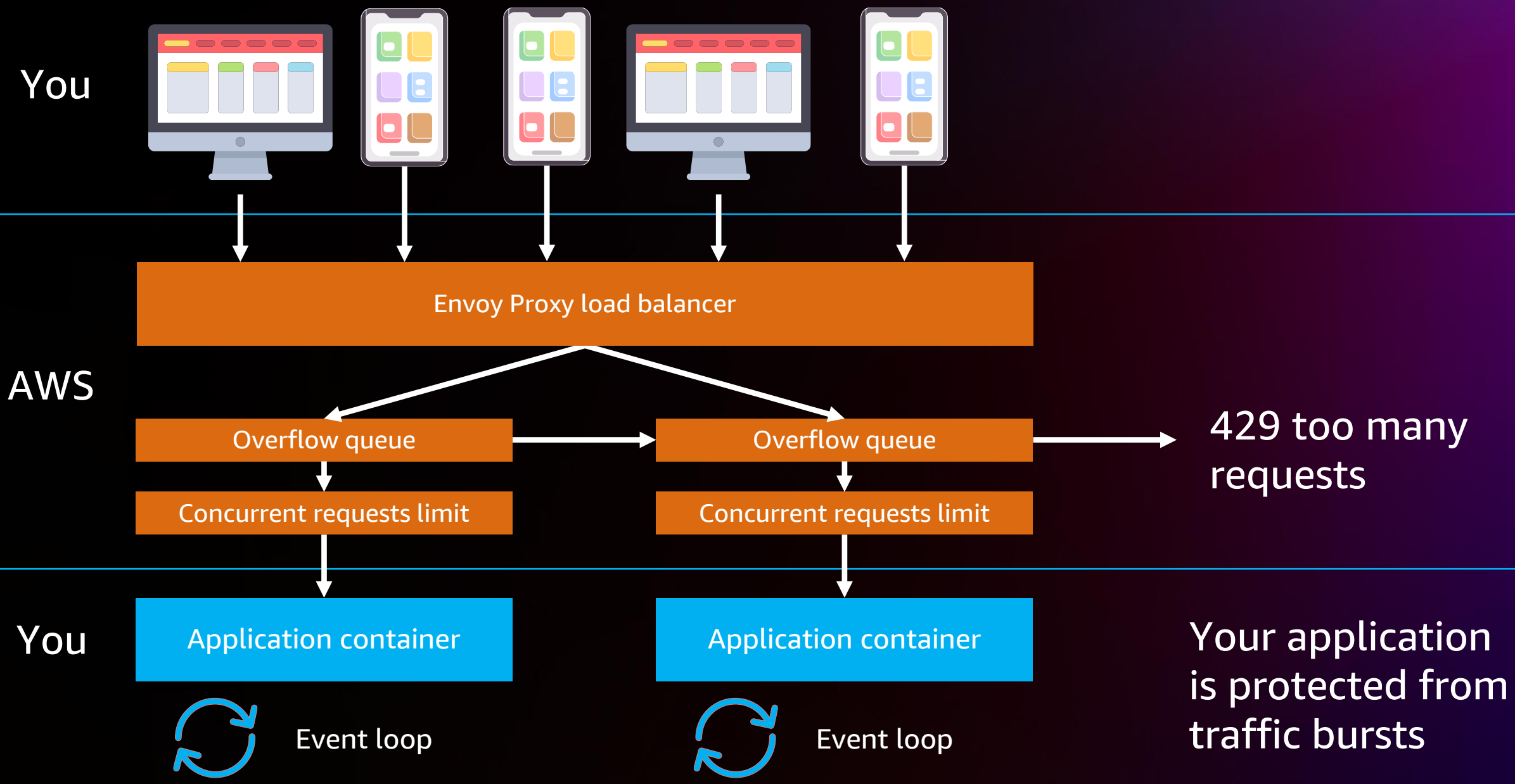
Application container

Event loop

Your application receives concurrent requests (up to the limit you defined) and processes them concurrently

aws

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



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

In this example: Max container instances = 3



App Runner scales out preemptively but make sure your app starts up fast or traffic might exceed available concurrency capacity

Extra traffic that can’t be served by three container instances starts to get 429 status code

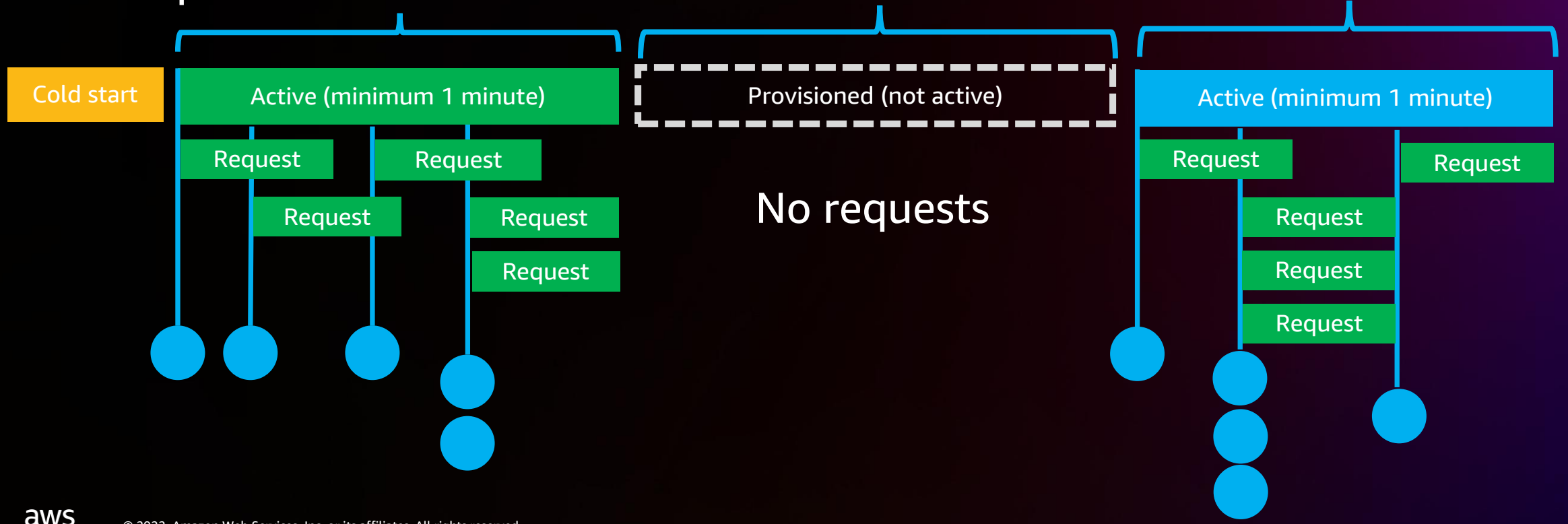
aws
© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

= Request arriving

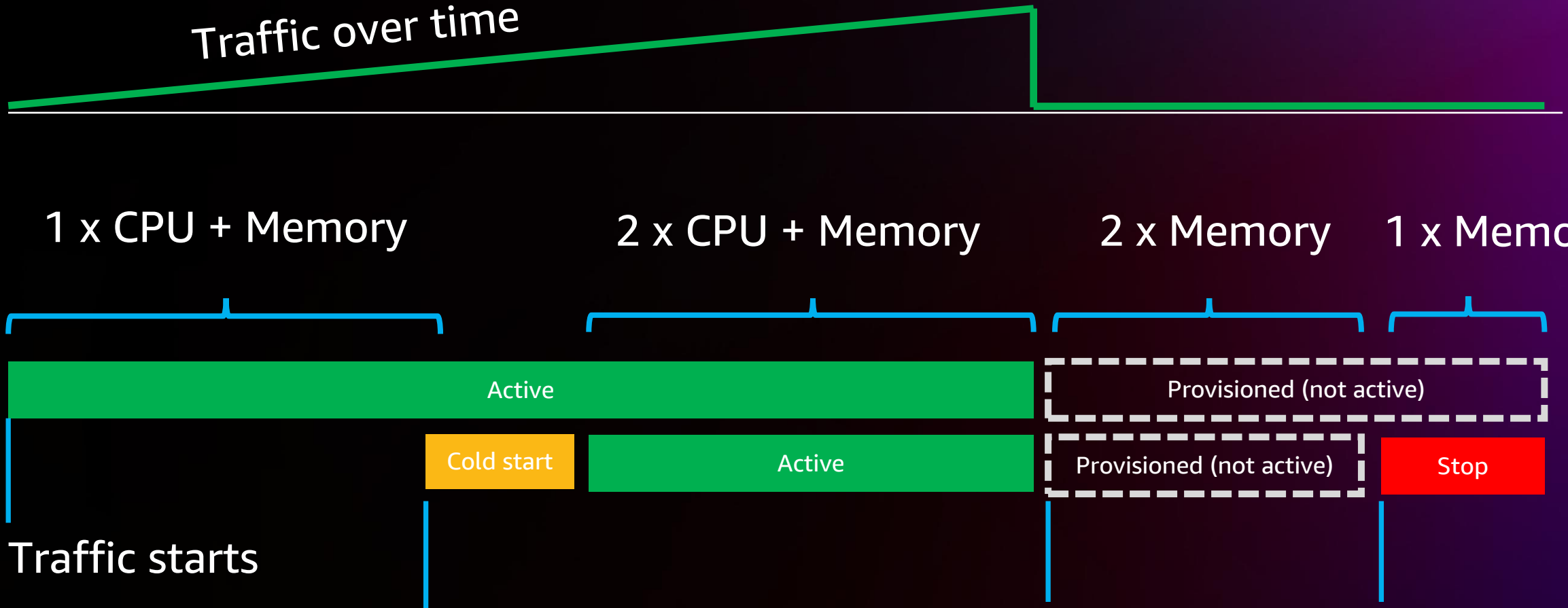
Pay for CPU and memory when the application gets requests

Pay only for memory while there are no requests

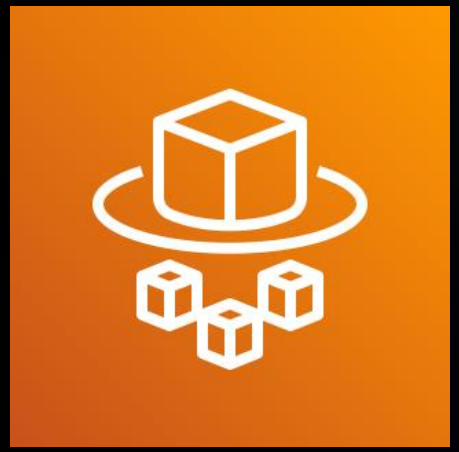
Back to paying for CPU + memory



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



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



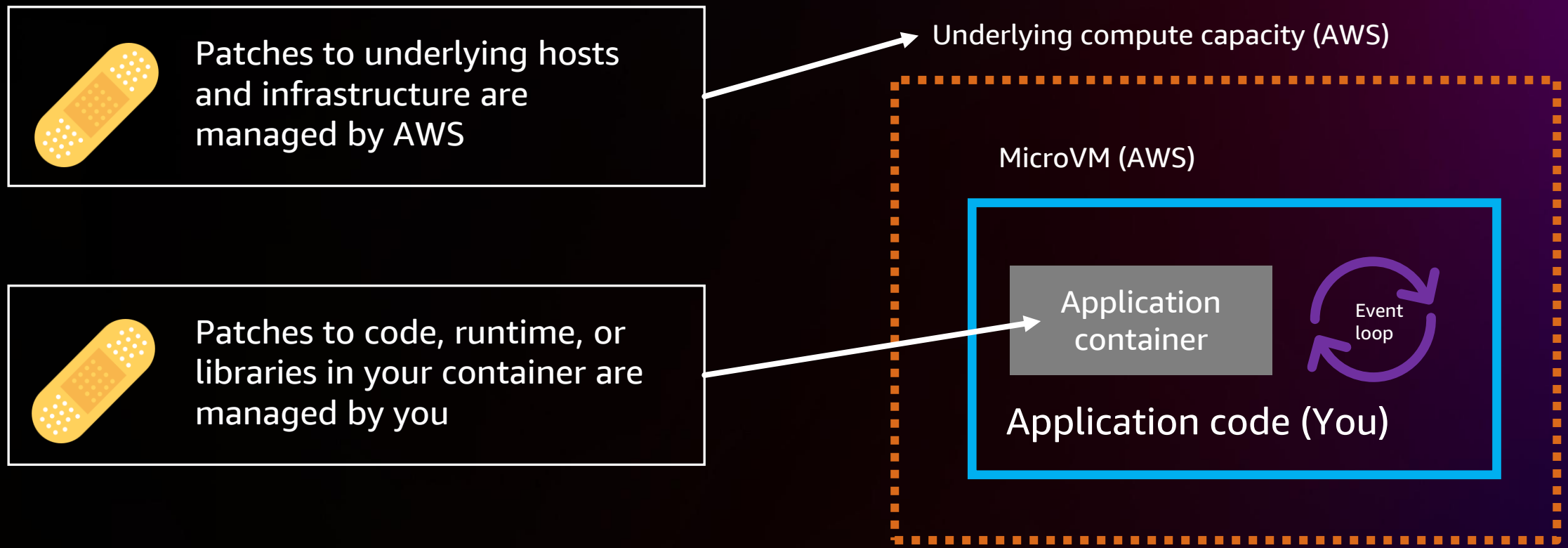
AWS Fargate

Serverless containers, but you manage scaling and concurrency

aws

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

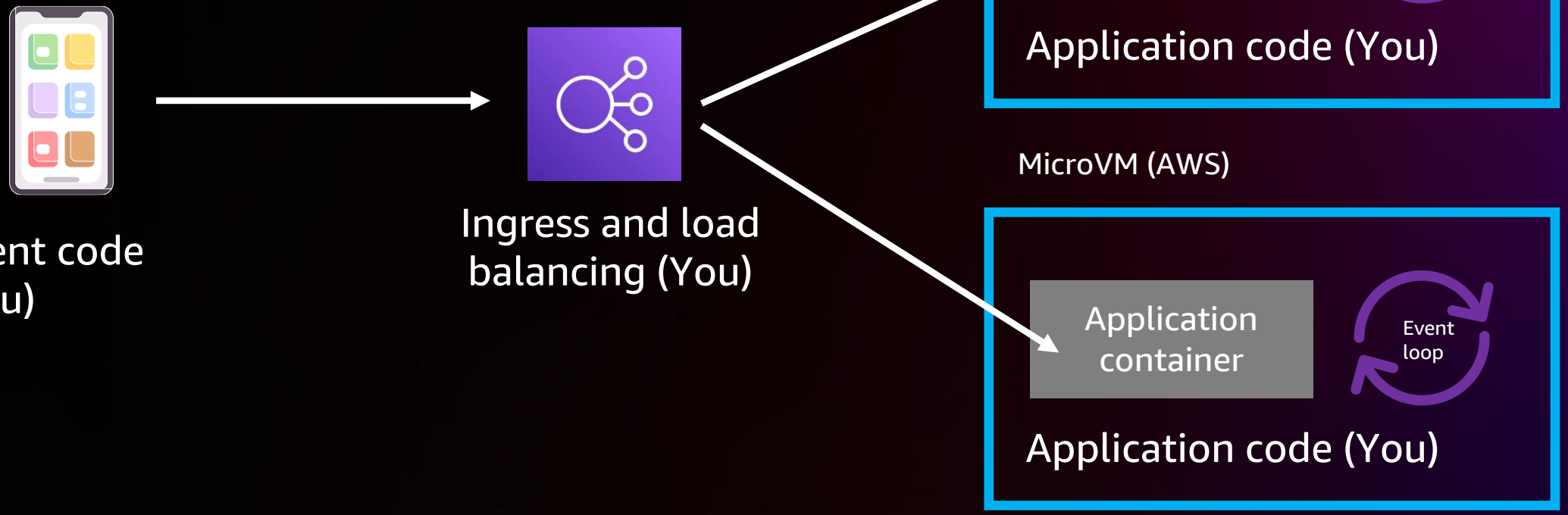
AWS Fargate provides microVMs on demand, patches, and maintains them



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

You manage load balancing and ingress

AWS = Fully managed by AWS
You = Your responsibility to manage

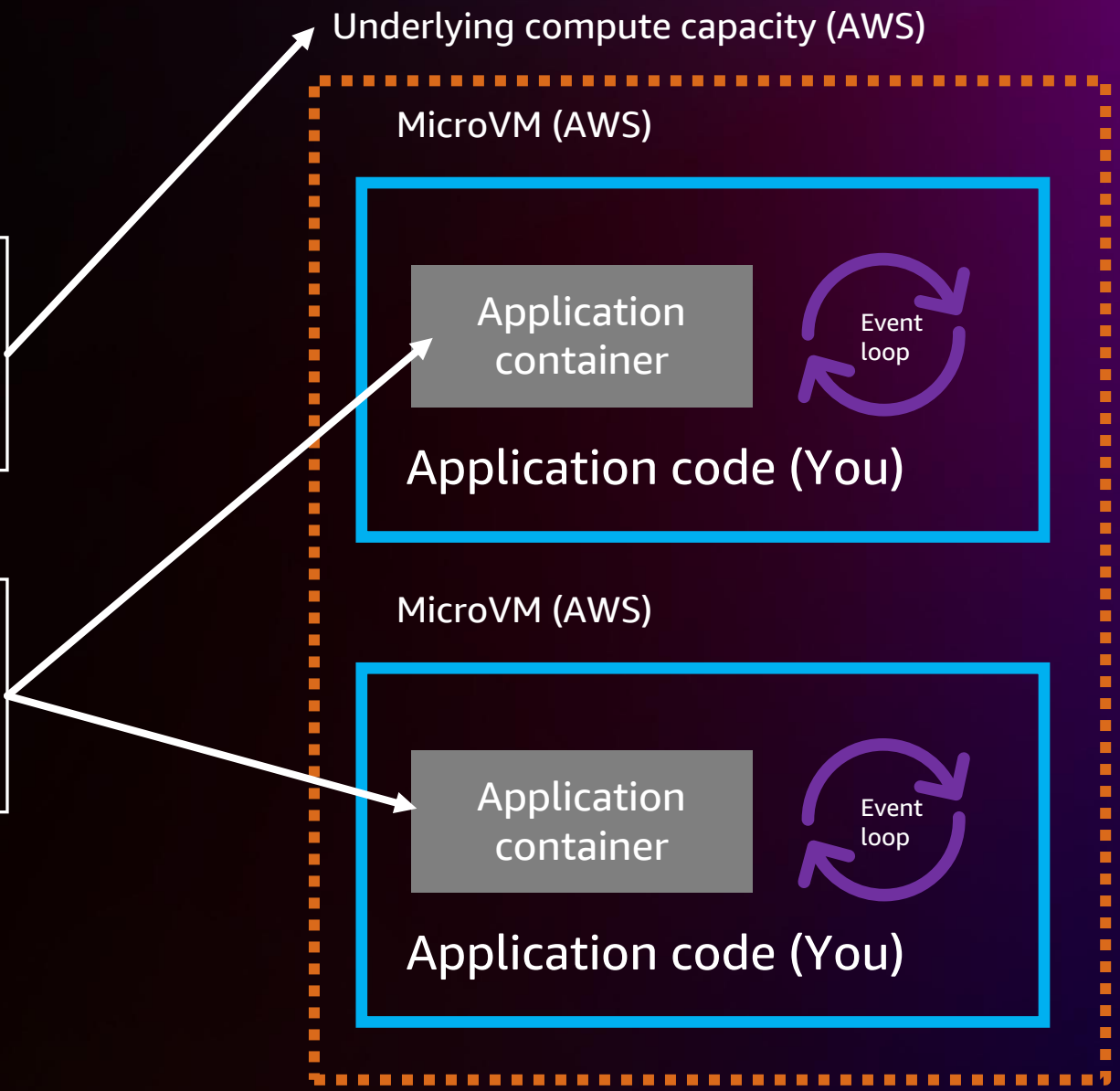


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

You configure scaling

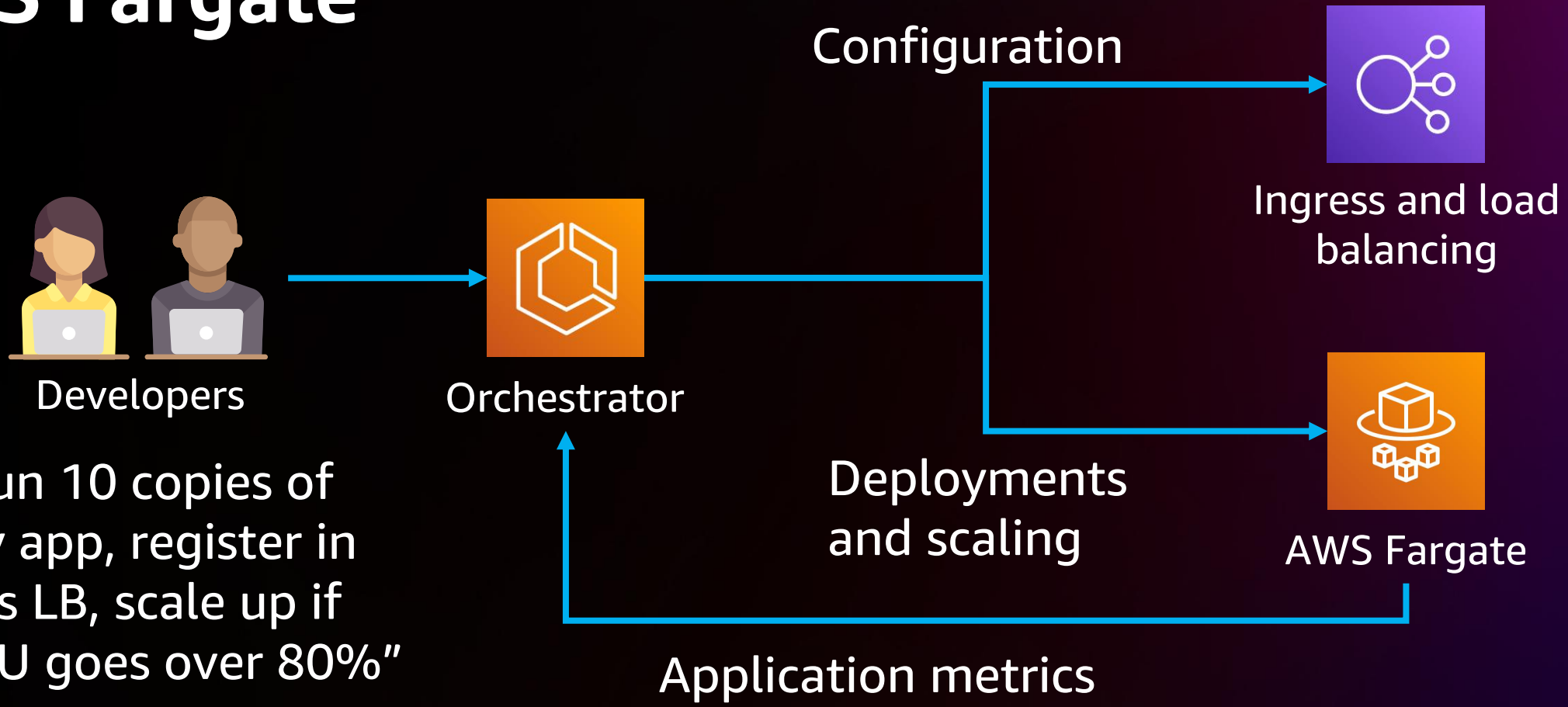
Underlying compute capacity is provided by AWS to make sure you can launch containers on demand

You have to decide how many containers you need and what triggers scaling for those containers



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

You can use an orchestrator to configure and manage your own customized deployment on AWS Fargate



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

AWS Fargate gives you more choices

API



Amazon Elastic Container Service

Fully AWS-managed serverless API. Only pay for the Fargate tasks.



Amazon Elastic Kubernetes Service

AWS-managed open-source deployment. Pay for your personal deployment of the control plane, plus AWS Fargate tasks.

aws

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

AWS Fargate gives you more choices

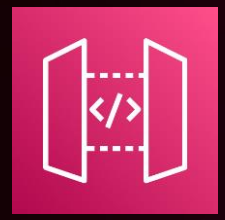
Ingress and load balancing



Application Load Balancer
Level 7, HTTP(S)



Network Load Balancer
Level 4

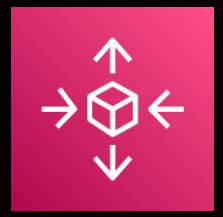


Amazon API Gateway

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

AWS Fargate gives you more choices

Scaling



AWS Auto Scaling

Step scaling
Target tracking
Scheduled scaling



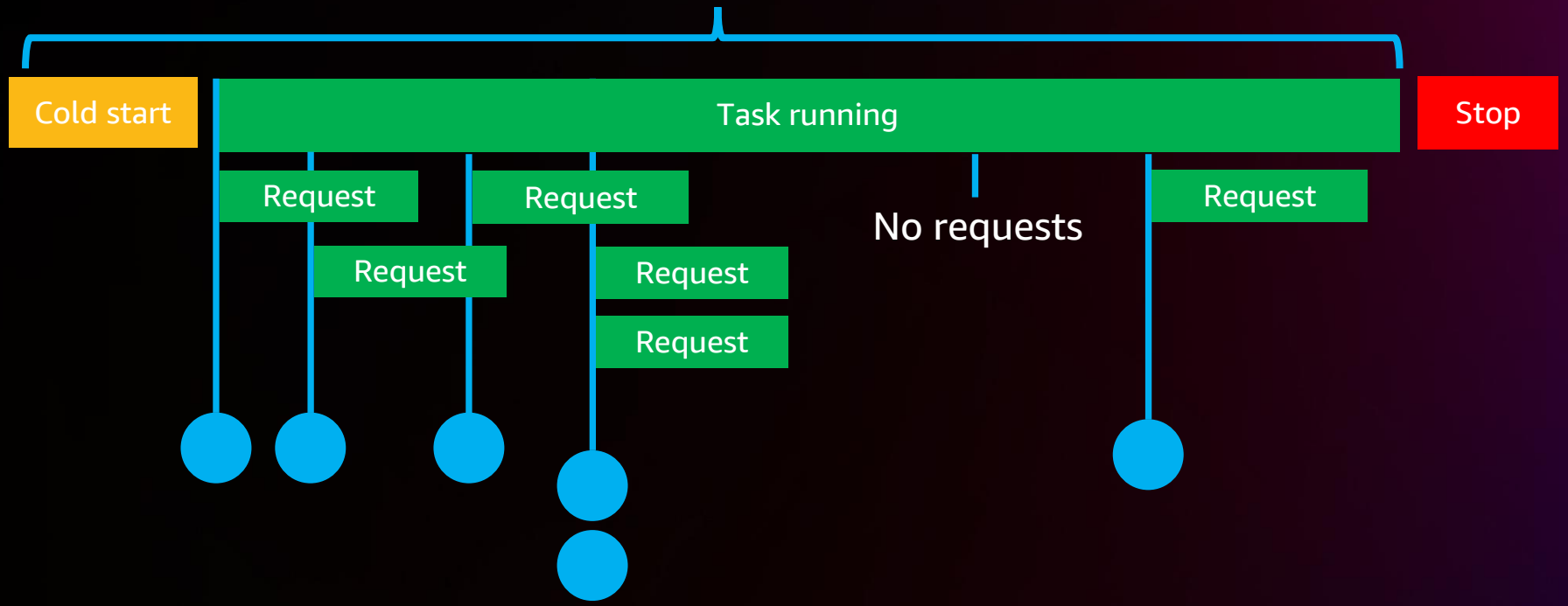
Amazon CloudWatch

Scale based on concurrency, CPU utilization, available messages in an SQS queue, or any other custom metric

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

AWS Fargate charges based on time, not activity

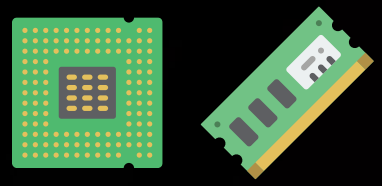
Pay a constant rate per second for the task CPU and memory



Lowest per-request cost when concurrent traffic is high and code is efficient

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

AWS Fargate price-saving strategies



Fine- tune CPU and memory size to match application needs; Fargate offers tiny task sizes for small applications



Savings plans

AWS Fargate is part of Compute Savings Plans



Spot tasks



AWS Graviton2

AWS Fargate has spot tasks for a discounted price as well as AWS Graviton2-powered tasks with Arm architecture

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

So which serverless option is right for my application?

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

	 AWS Lambda	 AWS App Runner	 AWS Fargate
Pricing	Per ms, per invoke, based on memory size of function Each concurrent invoke is charged separately	Per container instance per second, based on CPU and memory Container instances only charge for memory when not serving traffic	Per container instance per second, based on CPU and memory Constant price whether concurrent traffic is zero or hundreds of requests per second

aws

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

	 AWS Lambda	 AWS App Runner	 AWS Fargate
Pricing resolution	Each request is priced per millisecond, rounded up to the nearest millisecond Scale to zero between active requests	When a container instance activates, there is a minimum active time of 1 minute Active duration rounded up to the nearest second	Pricing is calculated per second with a 1-minute minimum Task duration rounded up to the nearest second

aws

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

	 AWS Lambda	 AWS App Runner	 AWS Fargate
Scaling	Each function instance only serves a single invoke at a time Lambda increases the number of function instances as needed	Each container instance serves many concurrent requests up to your limit App Runner manages the number of container instances automatically	High concurrency, but no built-in scaling You define your own custom scaling policy using an orchestrator, your metrics, and AWS Auto Scaling

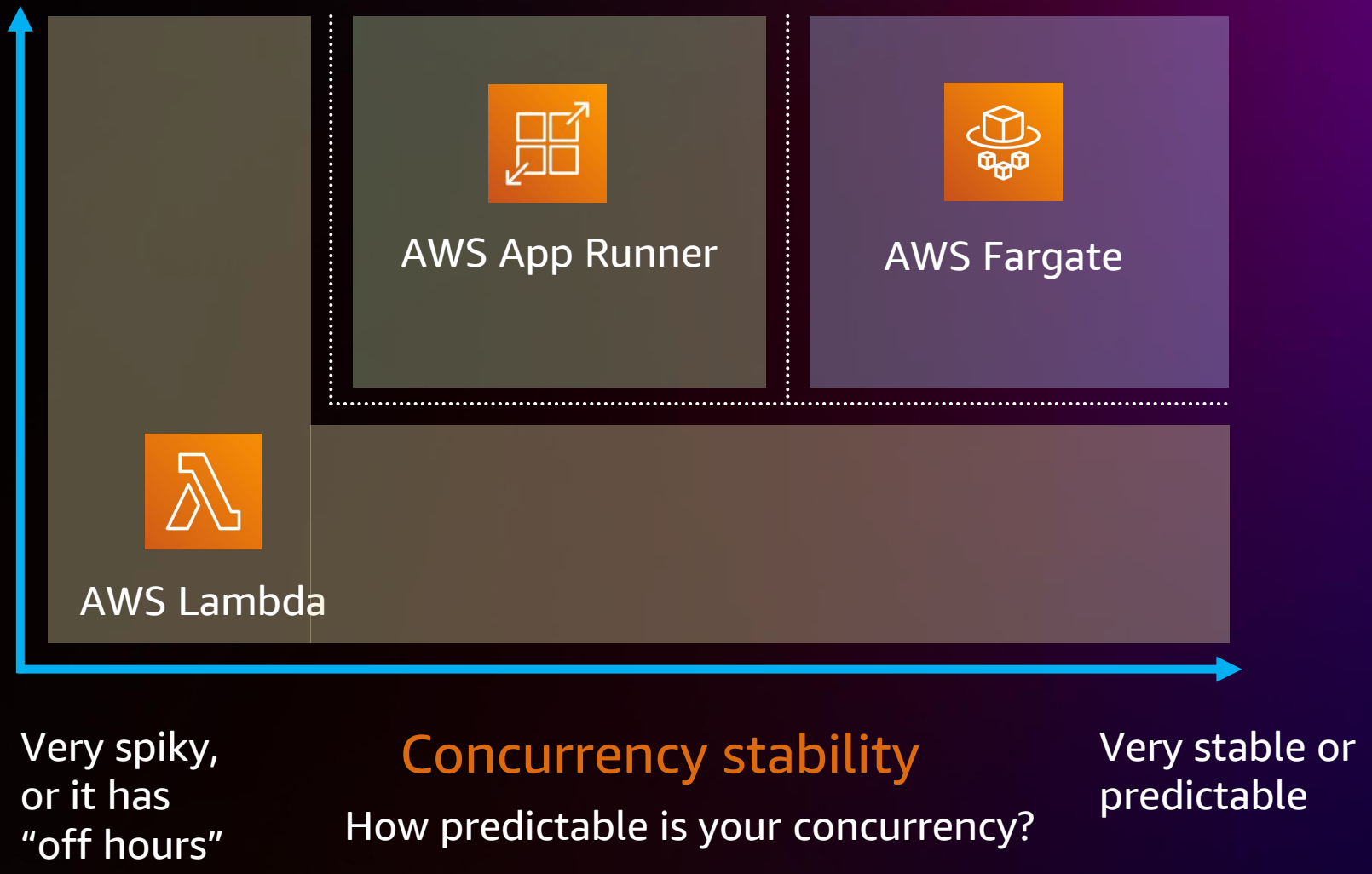
aws

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

	 AWS Lambda	 AWS App Runner	 AWS Fargate
Web traffic and ingress	 Your choice Manage your own API Gateway or Application Load Balancer or use built-in Lambda function URL	 Built in Just call the endpoint for your App Runner service, no need to manage anything	 Not included Manage your own API Gateway or Application Load Balancer

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

Serverless compute sweet spots for your container

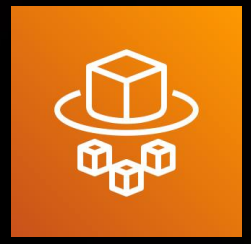


© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

	AWS Lambda	AWS App Runner	AWS Fargate
Per-process isolation	✓	✓	✓
Per-request isolation	✓	✗	✗

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

A range of serverless compute for different needs



AWS Fargate



AWS App Runner



AWS Lambda



Configurable
Control
More complex

Opinionated
Fully managed
Simple

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

Thank you!

Nathan Peck
@nathankpeck

Omar Paul



Please complete the session survey in the **mobile app**

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