aws LATAM Compute Roadshow

Mejores practicas de optimización de costos y rendimiento de aplicaciones Serverless

Peterson Larentis
AWS
Arquicteto Sénior de Soluciones Especialista, Serverless - LATAM

aws

Objetivo de la sesión

Profundizar en el computo serverless, su ciclo de vida, modelos de precios y técnicas de optimización que permitirán reducir costos y mejorar el rendimiento.

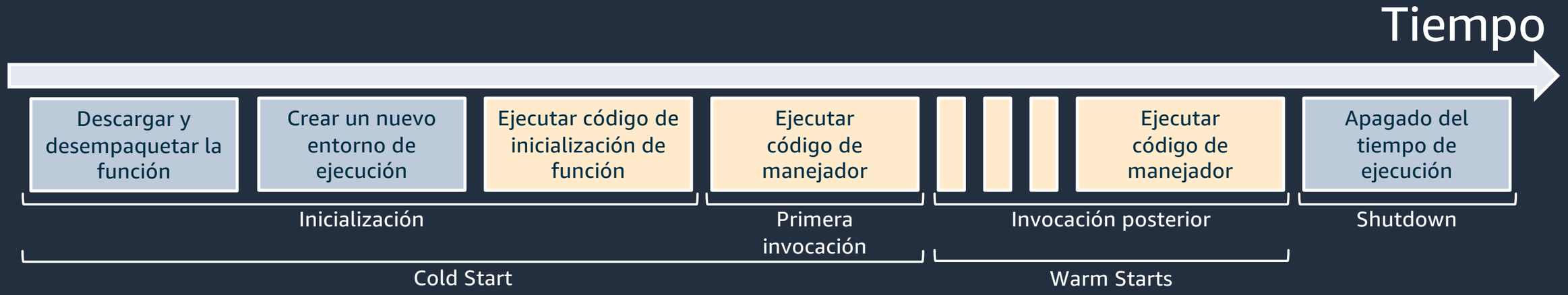
aws

aws LATAM Compute Roadshow

Optimización de las funciones de Lambda

aws

Ciclo de vida del entorno de ejecución de Lambda



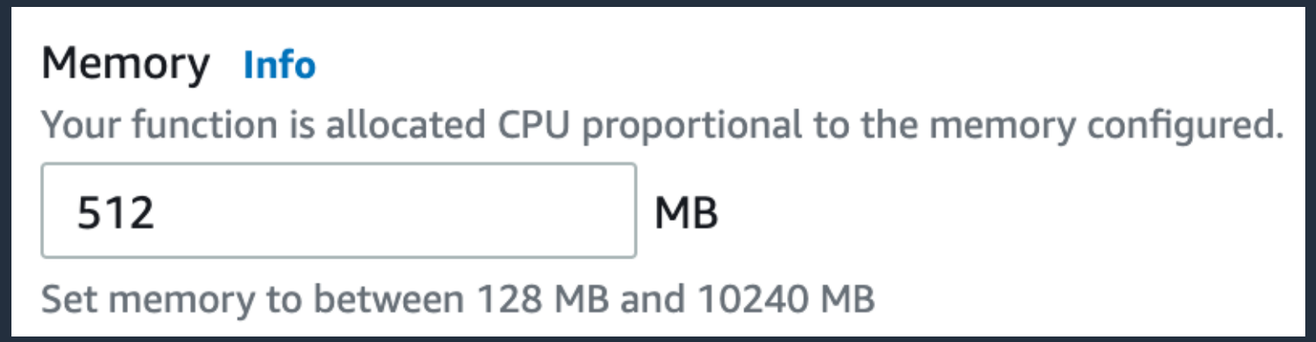
Inicio: Crear un entorno de ejecución nuevo o descongelar uno existente. Para nuevos entornos, ejecute el código de inicialización de la función.

Invocación: Ejecutar el código del manejador de la función, regresar una respuesta. Congelar el entorno de ejecución hasta la próxima invocación

Shutdown: se activa cuando se destruye el entorno de ejecución, por ejemplo, si la función no recibe invocaciones durante un período de tiempo

aws

Asignación de CPU y memoria de la función Lambda



La capacidad de CPU es asignado proporcionalmente a la cantidad de memoria asignada

En 1768 MB de memoria asignada, su función tendrá el equivalente de 1 vCPU completo y 10240 MB tendrá 6 vCPUs.

aws

Métricas de Facturación

AWS Lambda
Granularidad de Facturación
1ms

Architecture	Duration	Requests
x86 Price		
First 6 Billion GB-seconds / month	$0.0000166667 for every GB-second	$0.20 per 1M requests
Next 9 Billion GB-seconds / month	$0.000015 for every GB-second	$0.20 per 1M requests
Over 15 Billion GB-seconds / month	$0.0000133334 for every GB-second	$0.20 per 1M requests
Arm Price		
First 7.5 Billion GB-seconds / month	$0.0000133334 for every GB-second	$0.20 per 1M requests
Next 11.25 Billion GB-seconds / month	$0.0000120001 for every GB-second	$0.20 per 1M requests
Over 18.75 Billion GB-seconds / month	$0.0000106667 for every GB-second	$0.20 per 1M requests

$0.0000166667 por cada **GB-segundo**

$0.20 por 1M de requests

Duración = Memoria asignada (GB) x Tiempo de ejecución (segundos)
Costo = Duración x Número de invocaciones

(Se aplicarán cargos adicionales por número de solicitudes, transferencia de datos, almacenamiento efímero)

aws

Métricas de Facturación

¡Nuestro objetivo es equilibrar todo lo anterior!

aws

Ejemplo de Equilibrio de Costo/Rendimiento

“Calcular 1,000 veces todos los números primos <=1 M”

128 MB	11.722 seg	$0.024628
256 MB	6.678 seg	$0.028035
512 MB	3.194 seg	$0.026830
1024 MB	1.456 seg	$0.024638

aws

¡Examen sorpresa!

¿Cuál es más caro?

1000 ms con 2 GB de RAM

=

2000 ms con 1 GB de RAM

aws

¡Examen sorpresa!

¿Cuál es más caro?

128 MB de memoria
funcionando durante 10 segundos

128MB = $0.0000021/seg

10 seg @128MB = $0.000021

>

1 GB de memoria
funcionando por 1 segundo

1GB = $0.000017/seg

1 seg @1GB = $0.000017

(us-east-1)

aws

“Queremos una solución que soporte 10.000 peticiones por segundo.”

El Negocio

“Manteniendo los costes bajo control, con un buen rendimiento, e integración con nuestra base de datos relacional”

El Equipo de Arquitectura Corporativa

aws

La concurrencia de AWS Lambda afecta a todo esto

aws

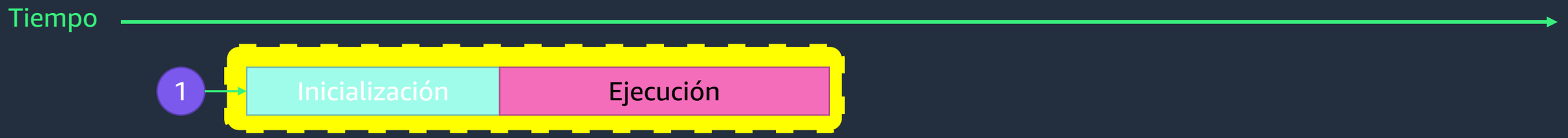
La concurrencia es una medida puntual

y no es lo mismo que solicitudes por segundo

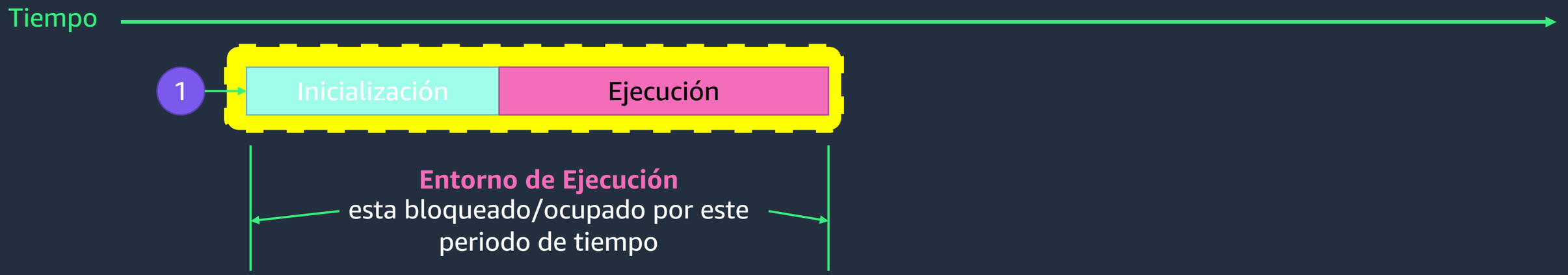
aws

Concurrencia en Lambda

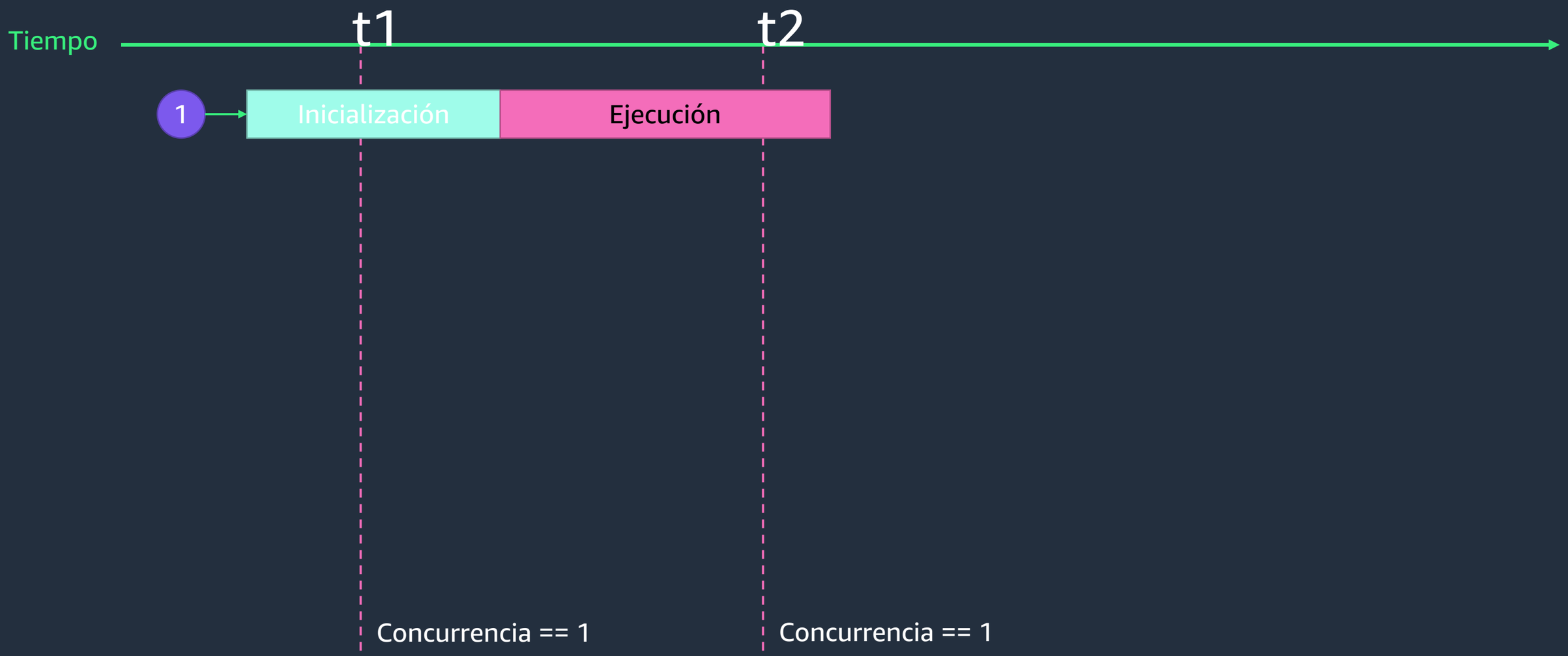
Concurrencia en Lambda



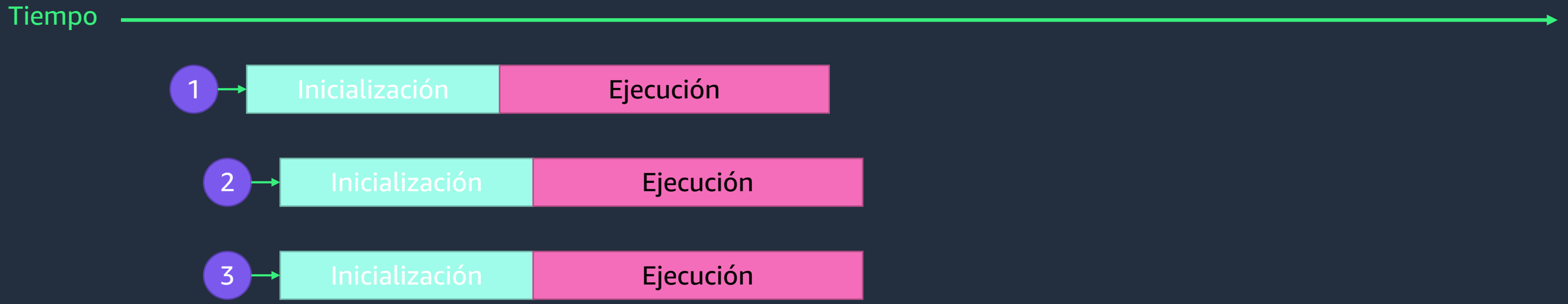
Concurrencia en Lambda



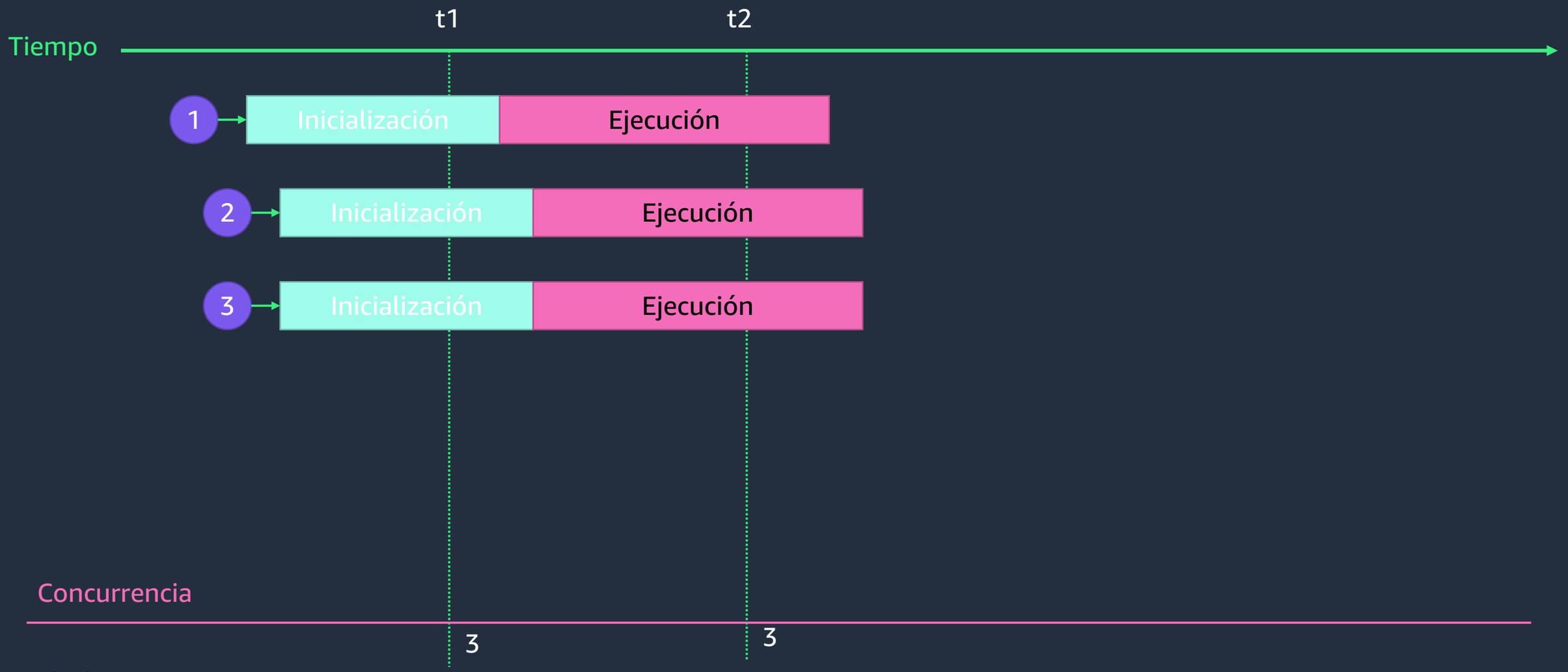
Concurrencia en Lambda



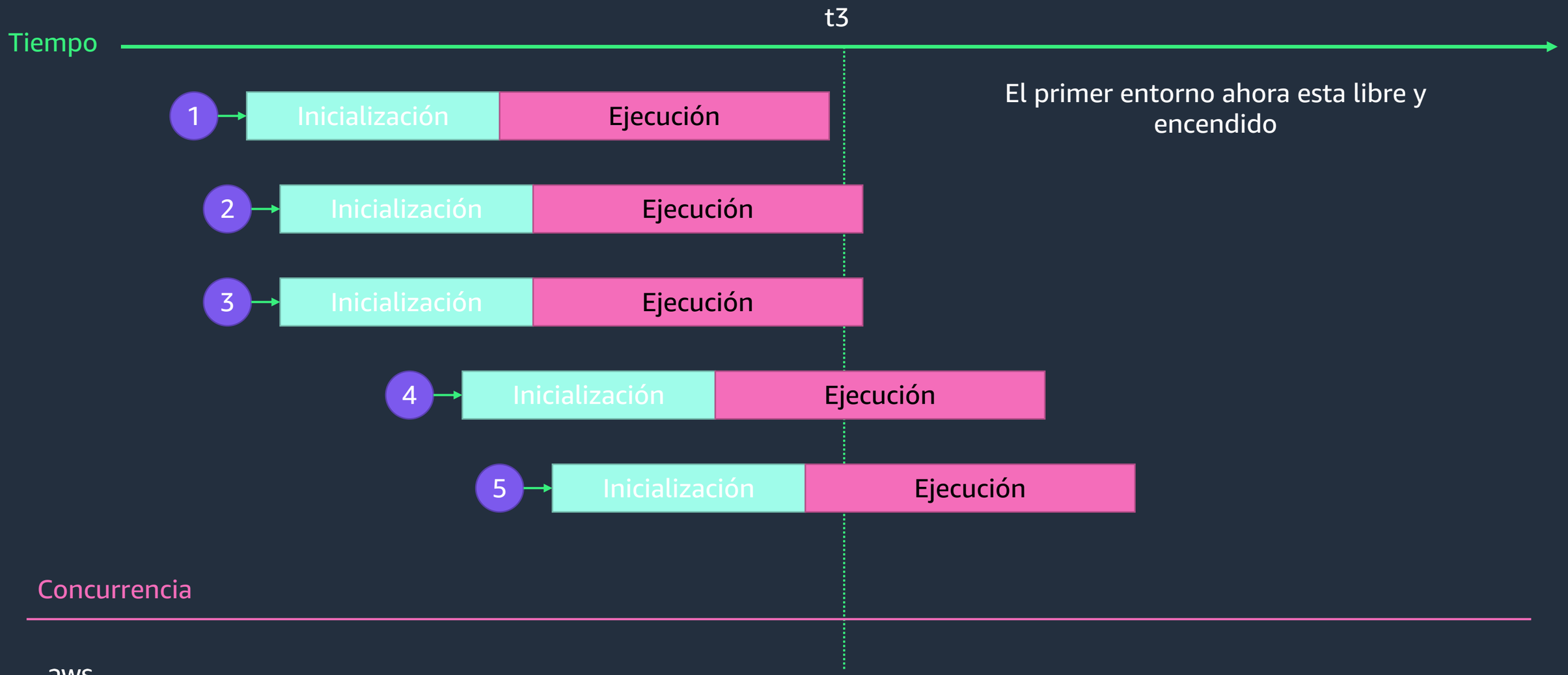
Concurrencia en Lambda



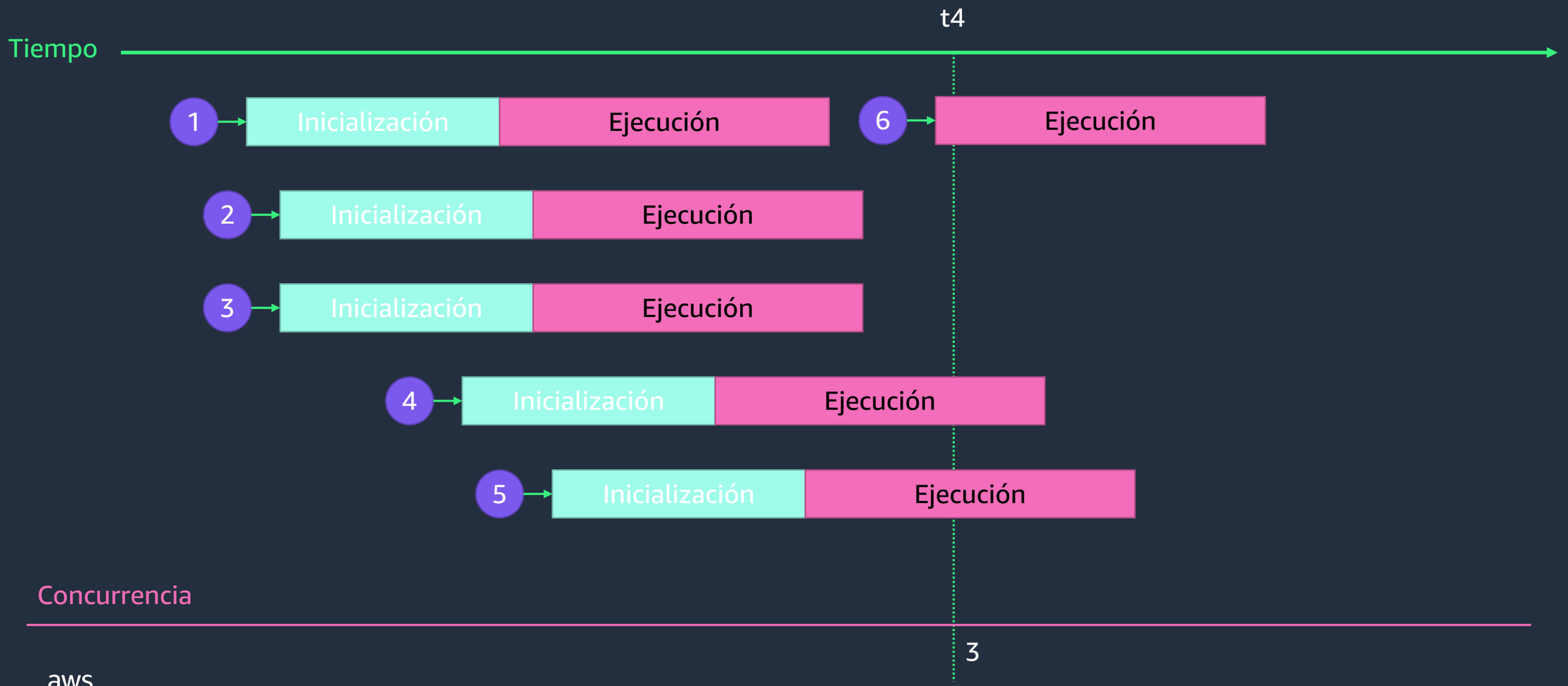
Concurrencia en Lambda



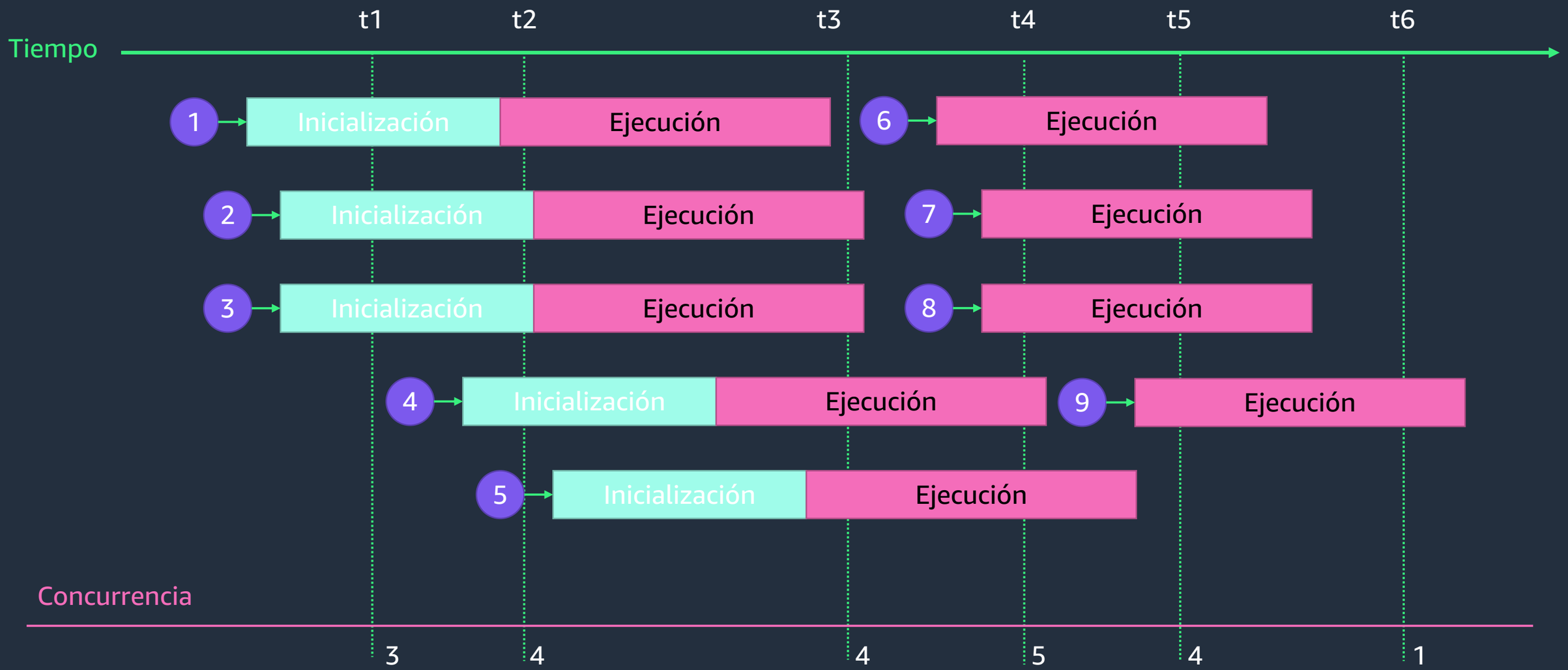
Concurrencia en Lambda



Concurrencia en Lambda



Concurrencia en Lambda



La concurrencia es una medida puntual

y no es lo mismo que solicitudes por segundo

aws

aws LATAM Compute Roadshow

¿Cómo optimizar y aprovechar al máximo la concurrencia?

aws

Concurrencia en Lambda

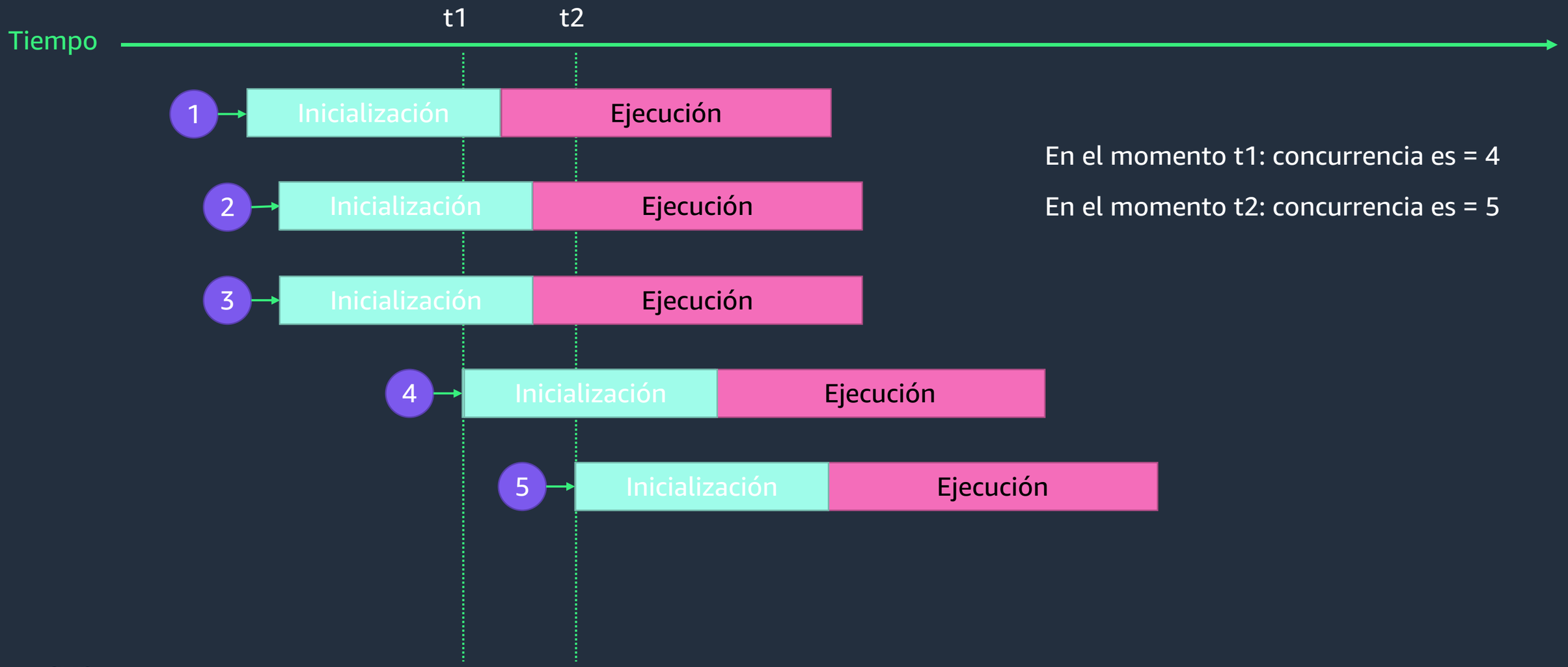
Concurrencia en Lambda = peticiones por segundo (RPS) x duración en segundos

Ejemplos:

- 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

Las funciones de larga ejecución requieren más concurrencia

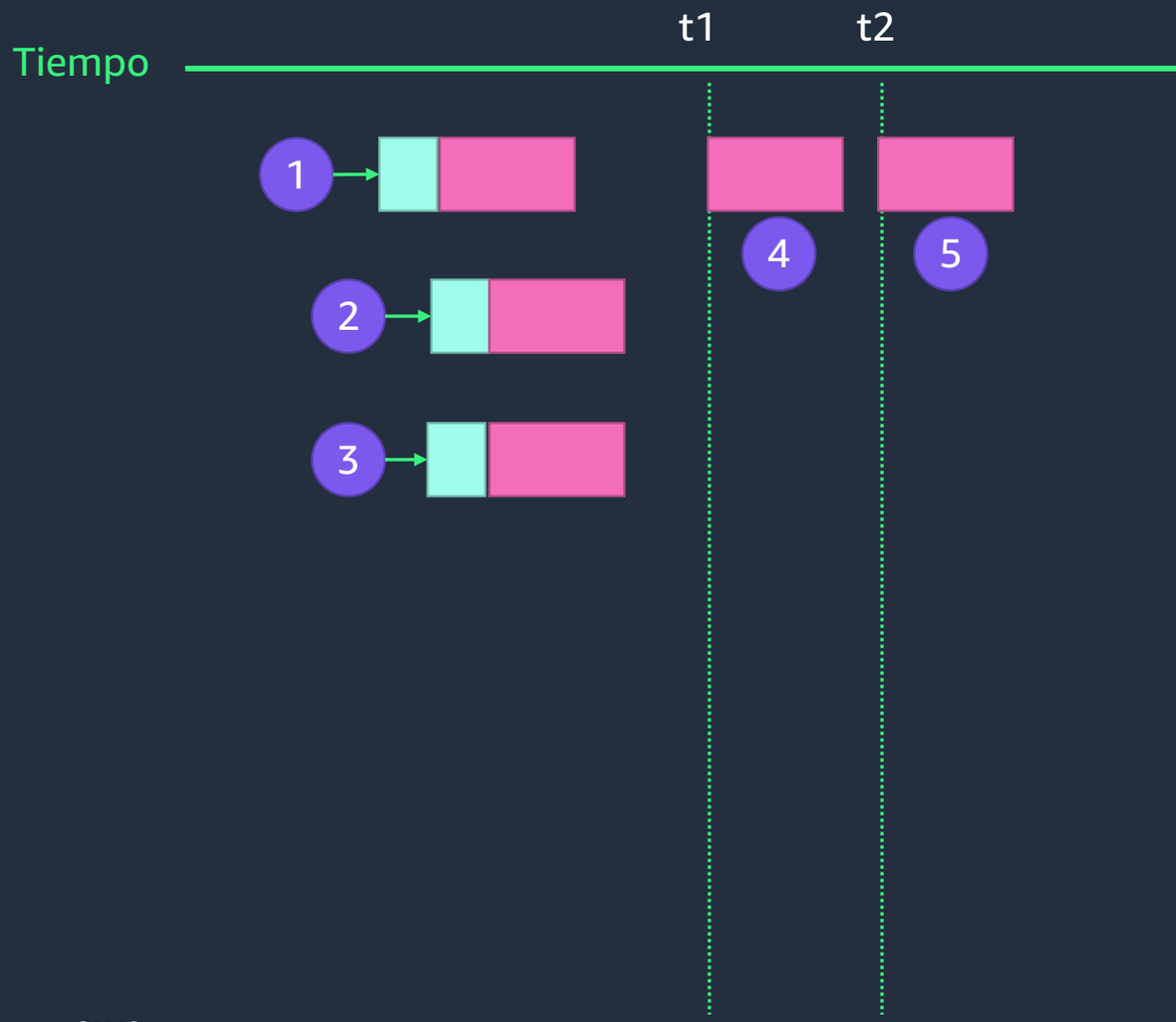
Disminución de la concurrencia



En el momento t1: concurrencia es = 4

En el momento t2: concurrencia es = 5

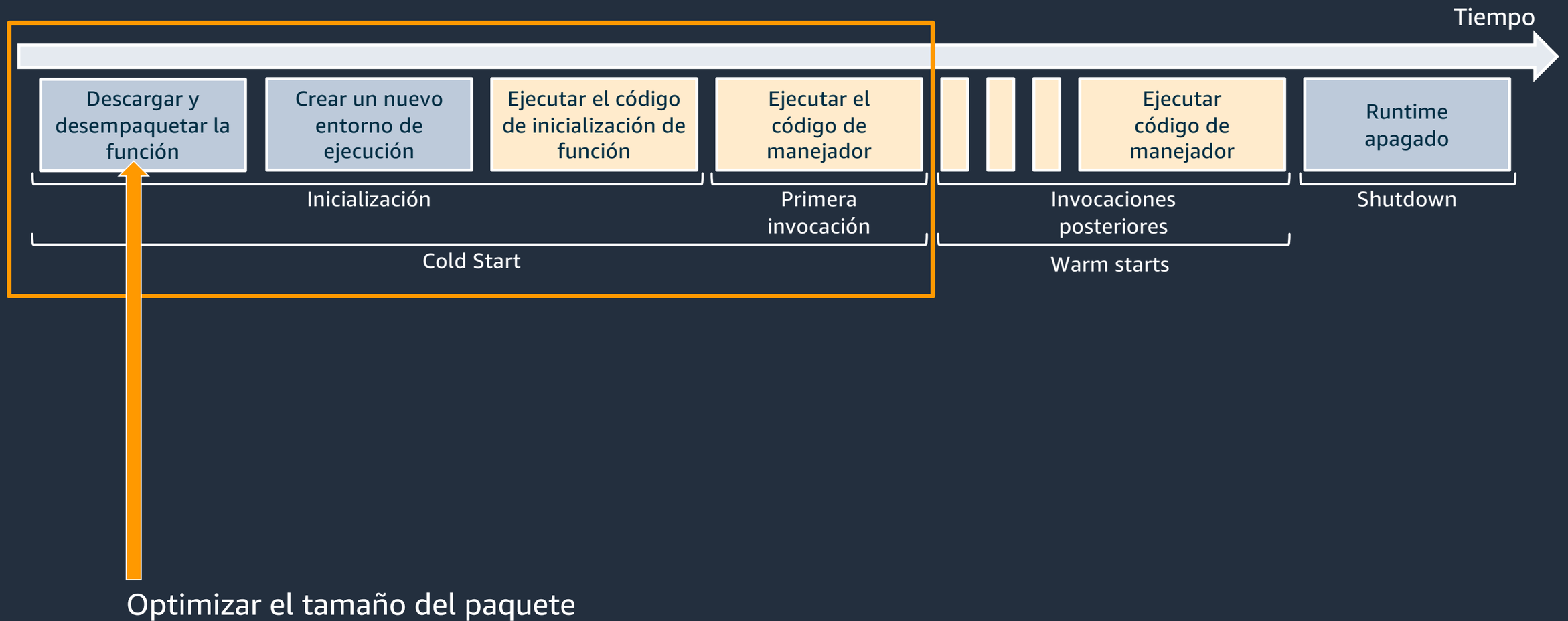
Disminución de la concurrencia



En el momento t1: concurrencia es = 1

En el momento t2: concurrencia es = 1

Cold Starts



Optimizar el tamaño del paquete

Beneficio

Reducción en los arranques en frío de la función Lambda, optimización del rendimiento

Herramientas

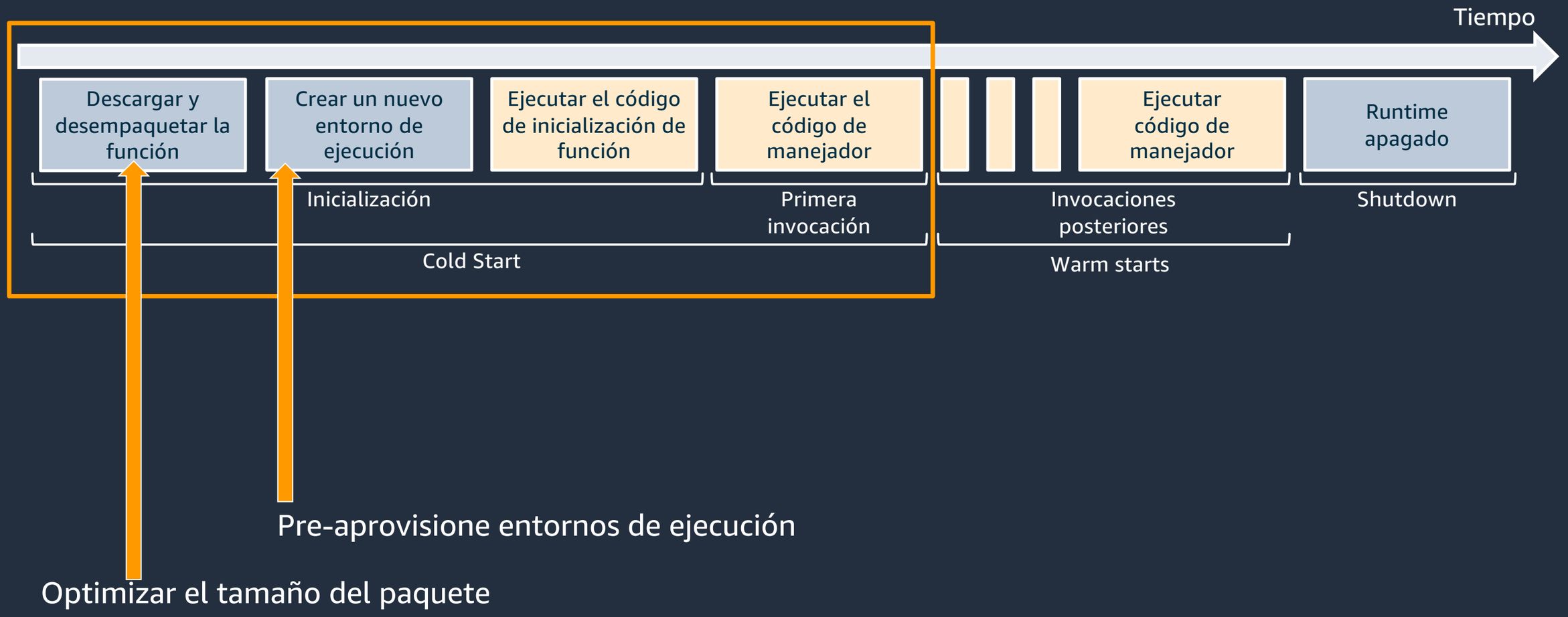
Maven, Gradle, WebPack, esbuild y otras más

Acción

- Evite las funciones monolíticas
- Reduzca el tamaño del paquete de implementación (por ejemplo, minificar el código de producción)
- Agrupe el conjunto mínimo requerido de dependencias
- Utilice las capacidades de sus herramientas de agrupamiento como análisis de dependencia de Maven o esbuild tree shaking para detectar y eliminar dependencias no utilizadas

aws

Cold Starts



Concurrencia aprovisionada

Beneficio

Reducción de arranques en frío de Lambda. Evite el estrangulamiento de ráfaga. Ahorre costos en cargas de trabajo con tráfico consistente.

Herramientas

Escalado automático de aplicaciones

Acción

- Utilice la concurrencia aprovisionada cuando espere un pico de tráfico
- Habilite la concurrencia aprovisionada para cargas de trabajo con tráfico relativamente consistente
- Utilice AWS Autoscaling para ajustar las cargas de trabajo que tienen tráfico variable

aws

Concurrencia aprovisionada

	Precios on-demand (us-east-1)
Solicitudes	$0.20 por 1M solicitudes
Duración de la Invocación	$0.0000166667 por GB-segundo

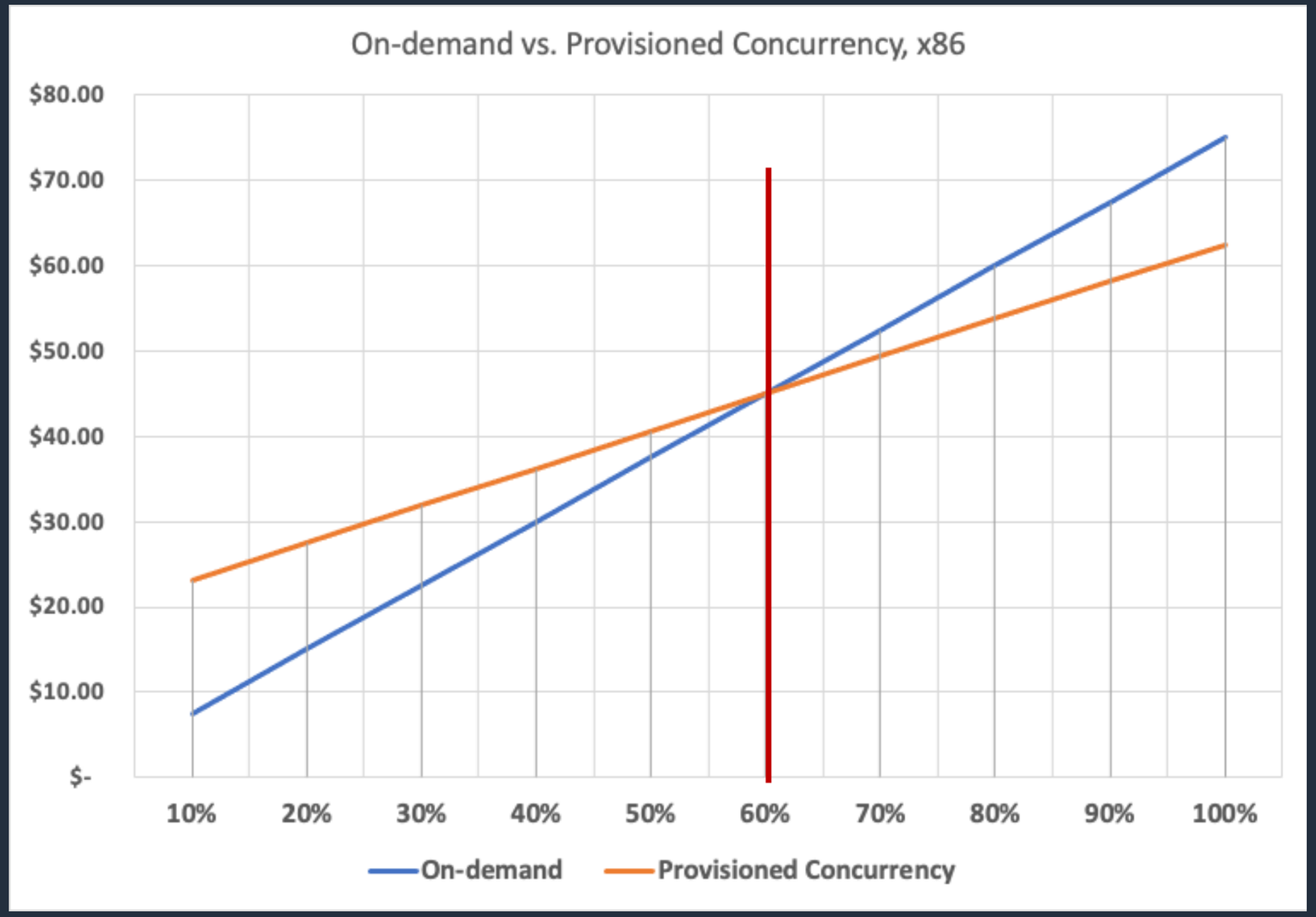
	Precios de Concurrencia aprovisionada (us-east-1)
Solicitudes	$0.20 por 1M solicitudes
Concurrencia Aprovisionada	$0.0000041667 por GB-segundo
+ Duración de la Invocación	$0.0000097222 por GB-segundo

Total: $0.0000138889

~ 16% más barato que on-demand si se utiliza completamente

aws

Concurrencia aprovisionada



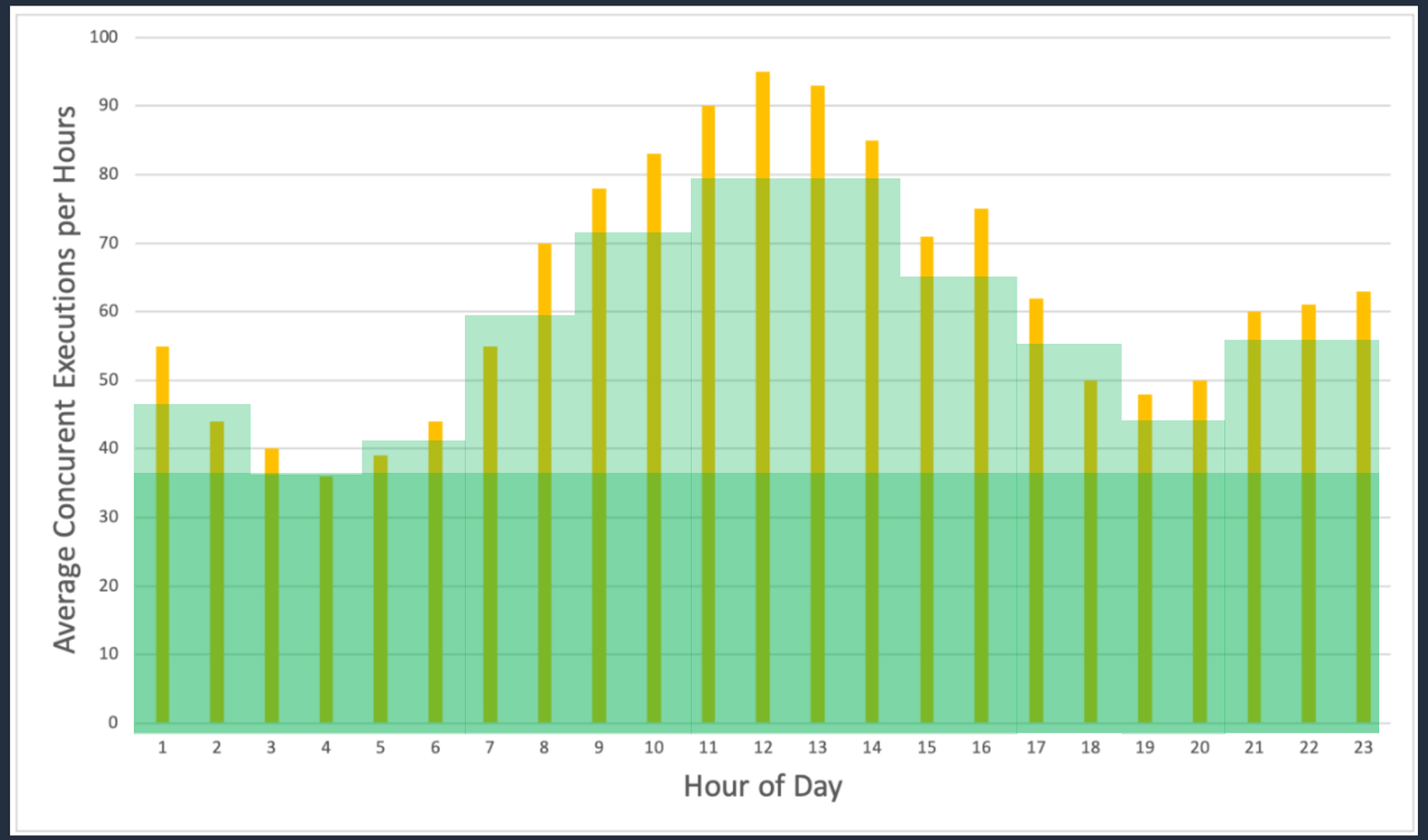
[us-east-1] - Los precios difieren por región, diferentes regiones tendrán diferentes umbrales

Concurrencia aprovisionada

1. Analizar patrones de ejecuciones concurrentes

2. Comience con Concurrencia aprovisionada estática

3. Evolucione a Concurrencia aprovisionada dinámica con application auto-scaling



SnapStart

Beneficio

Ofrece un rendimiento de inicio hasta 10 veces más rápido para las funciones Java, sin costo adicional y normalmente sin cambios en su código.

Herramientas

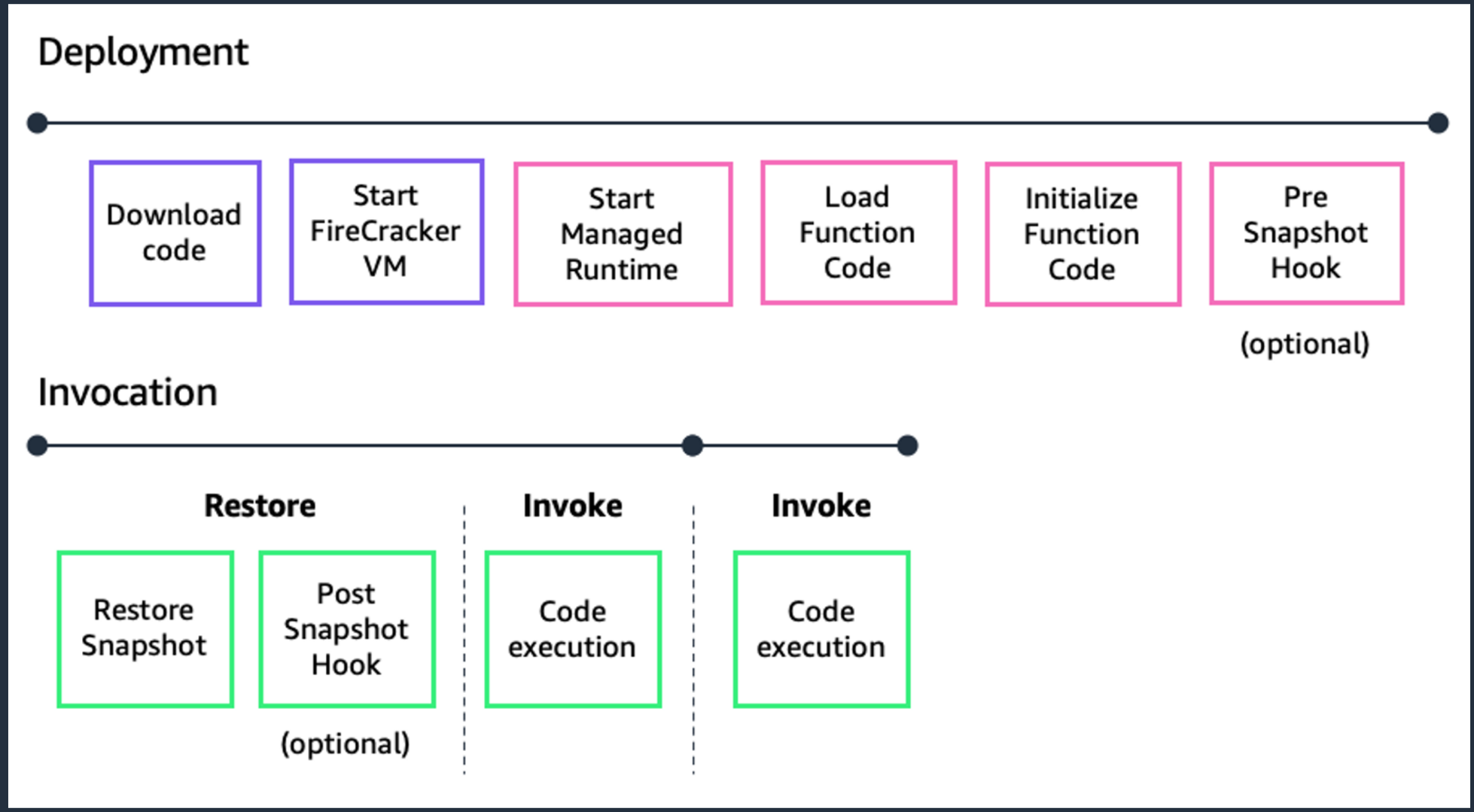
No se requieren herramientas adicionales

Acción

- Use SnapStart al crear funciones Java en AWS Lambda
- Uselo para aplicaciones sensibles a la latencia

aws

SnapStart Lifecycle



aws

Runtime Hooks Código de ejemplo

```java
public class HelloHandler implements RequestHandler<String, String>, Resource {

    public HelloHandler() {
        Core.getGlobalContext().register(this);
    }

    public String handleRequest(String name, Context context) throws IOException {
        System.out.println("Handler execution");
        return "Hello " + name;
    }

    @Override
    public void beforeCheckpoint(org.crac.Context<? extends Resource> context) throws Exception {
        System.out.println("Before Checkpoint");
    }

    @Override
    public void afterRestore(org.crac.Context<? extends Resource> context) throws Exception {
        System.out.println("After Restore");
    }
}
```

CRaC interface

Hook code

aws

SnapStart

"SnapStart requiere poco o ningún cambio de código y ofrece mejoras significativas de cold starts en numerosos casos de uso de Lambda... Aplicar el poder de la tecnología serverless nos permite enfocarnos más en el valor del cliente y menos en la gestión de la infraestructura"

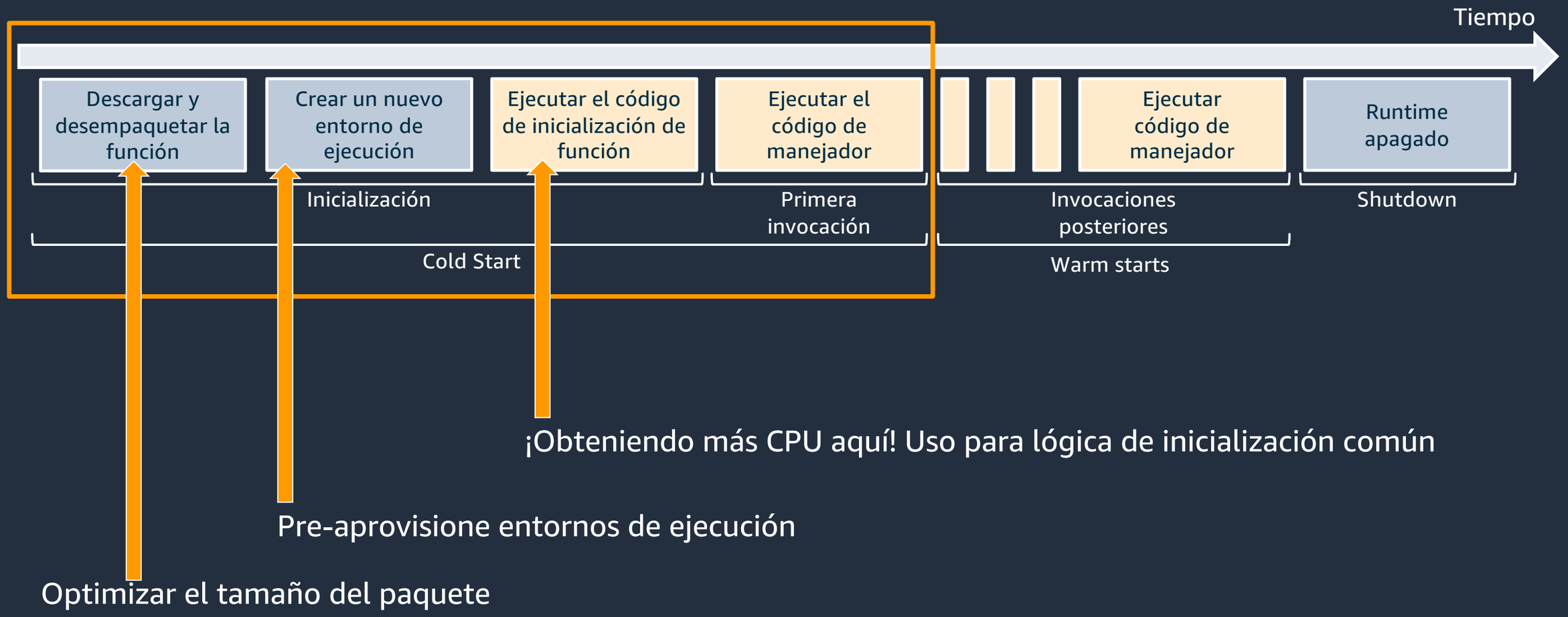
- Marty Andolino
Vicepresidente de Ingeniería, Arquitecto Jefe Divisional

"SnapStart ha mejorado 95% el desempeño del arranque en frío para algunos de nuestros servicios clave. Más del 50% de nuestras funciones de AWS Lambda en producción utilizan el runtime de Java, sirviendo a 6.5M invocaciones al día"

- Ivonne Roberts
Arquitecto de Software

aws

Cold Starts



Aproveche la capacidad adicional de la CPU durante la función Init

Beneficio

Tiempo de ejecución eficiente. Arranques en frío más rápidos.

Herramientas

AWS Lambda

Acción

- Crear, inicializar y configurar objetos que se espera que sean reutilizados en múltiples invocaciones durante la inicialización de funciones en lugar de hacerlo en el manejador (por ejemplo, constructor estático).

aws

Aproveche la capacidad adicional de la CPU en la inicialización de la Función

```
import { SSMClient, GetParameterCommand } from '@aws-sdk/client-ssm';

export async function handler (event, context) {
    const parameterValue = await getParamValue();
    return 'Parameter value: ' + parameterValue;
}

async function getParamValue(){
    const ssmClient = new SSMClient();
    const command = new GetParameterCommand({
        Name: 'foo'
    });
    const resp = await ssmClient.send(command);
    return resp.Parameter.Value;
}
```

Tamaño de Memoria: 128 MB

Duración del Inicio: 657ms
Duración de Invocación: 888ms
Tiempo total de invocación en frío: 1545ms
Duración Facturada: 888ms

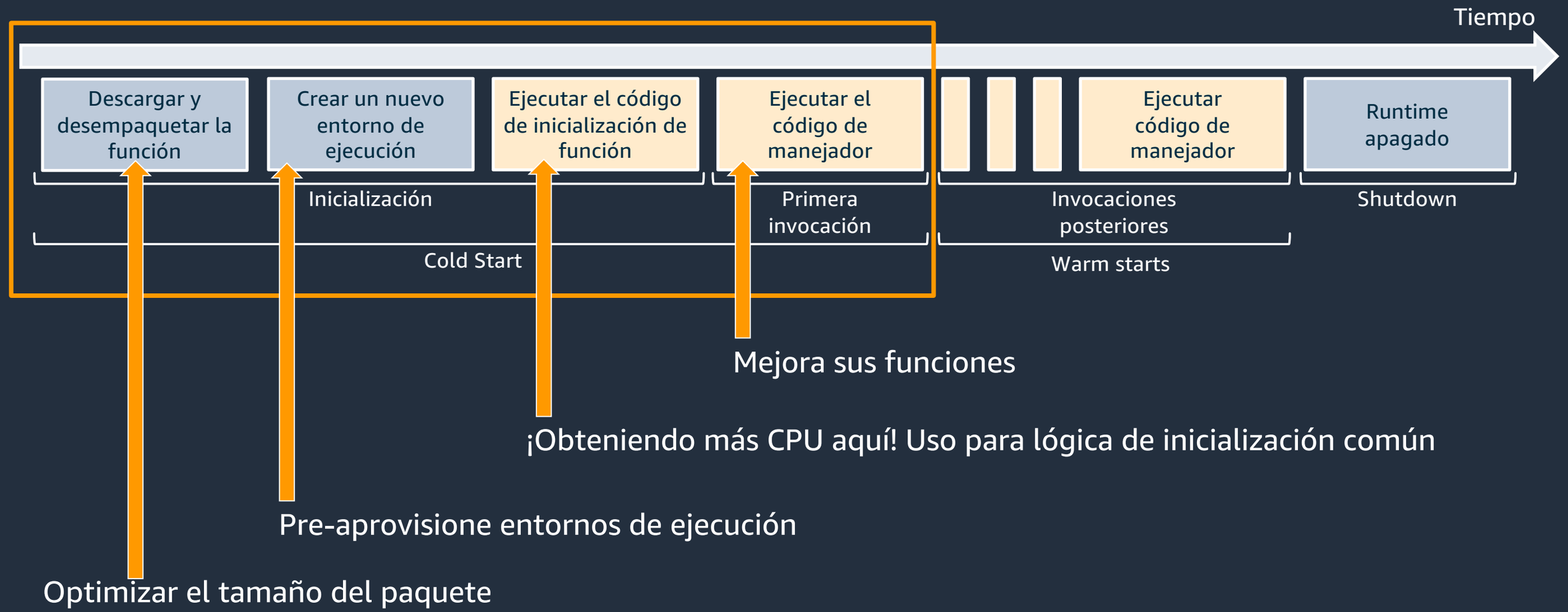
```
import { SSMClient, GetParameterCommand } from '@aws-sdk/client-ssm';

const ssmClient = new SSMClient();
const parameterValue = await getParamValue();

export async function handler (event, context) {
    return 'Parameter value: ' + parameterValue;
}

async function getParamValue(){
    const command = new GetParameterCommand({
        Name: 'foo'
    });
    const resp = await ssmClient.send(command);
    return resp.Parameter.Value;
}
```

Tamaño de Memoria: 128 MB

Duración del Inicio: 765ms
Duración de Invocación: 3ms
Tiempo total de invocación en frío: 768ms
Duración Facturada: 3ms

aws

Cold Starts



Ajustar el tamaño de sus funciones (Right-size)

Beneficio

Ejecute la carga de trabajo al menor costo sin comprometer el rendimiento

Herramientas

AWS Lambda Power Tuning, AWS Compute Optimizer, CloudWatch Lambda Insights

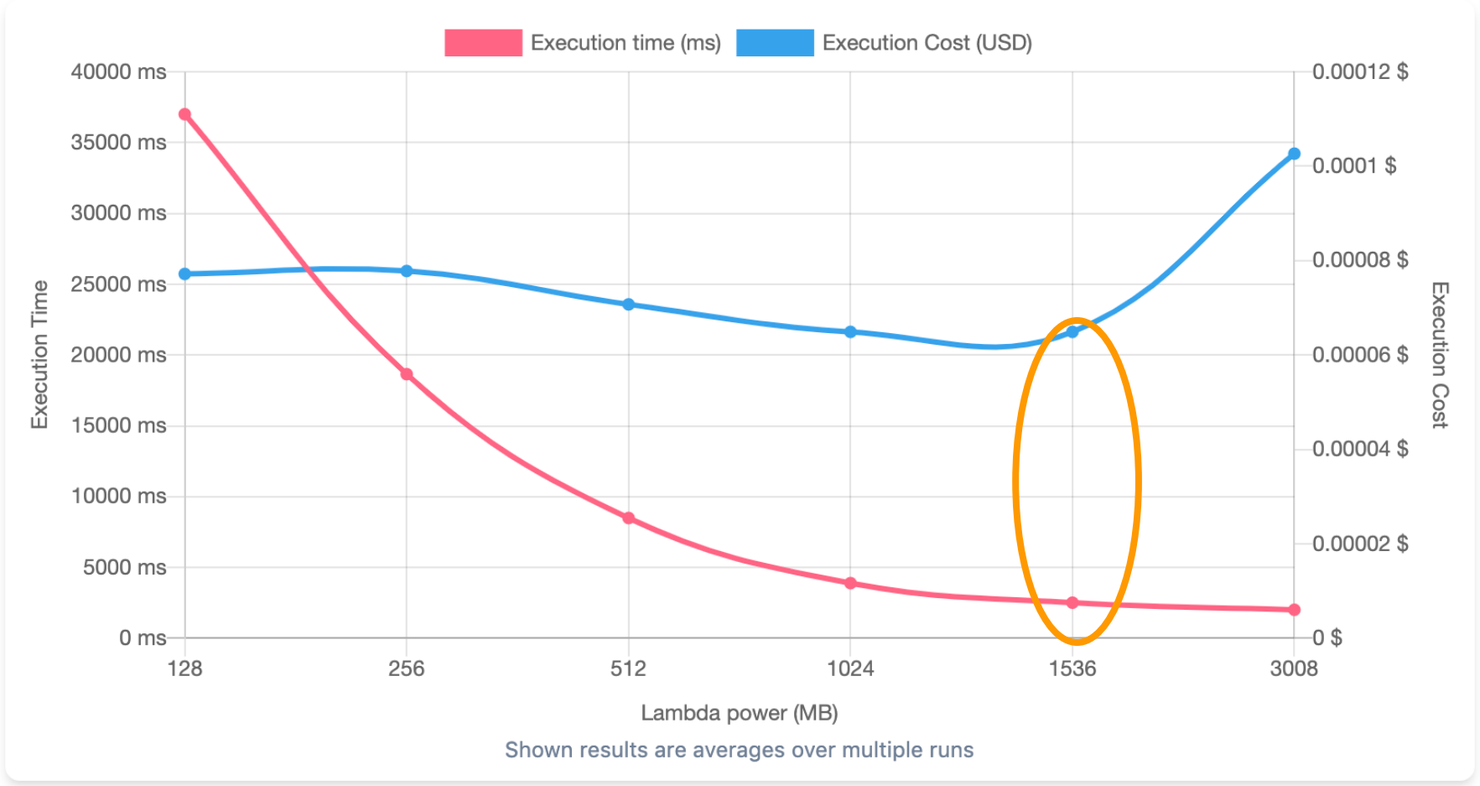
Acción

- Utilice Lambda Power Tuning para identificar la configuración óptima de la memoria sin aumentar los costos
- Considere las sugerencias de Compute Optimizer
- Reevalue después de cambios importantes

aws

Lambda Power Tuning

AWS Lambda Power Tuning Results



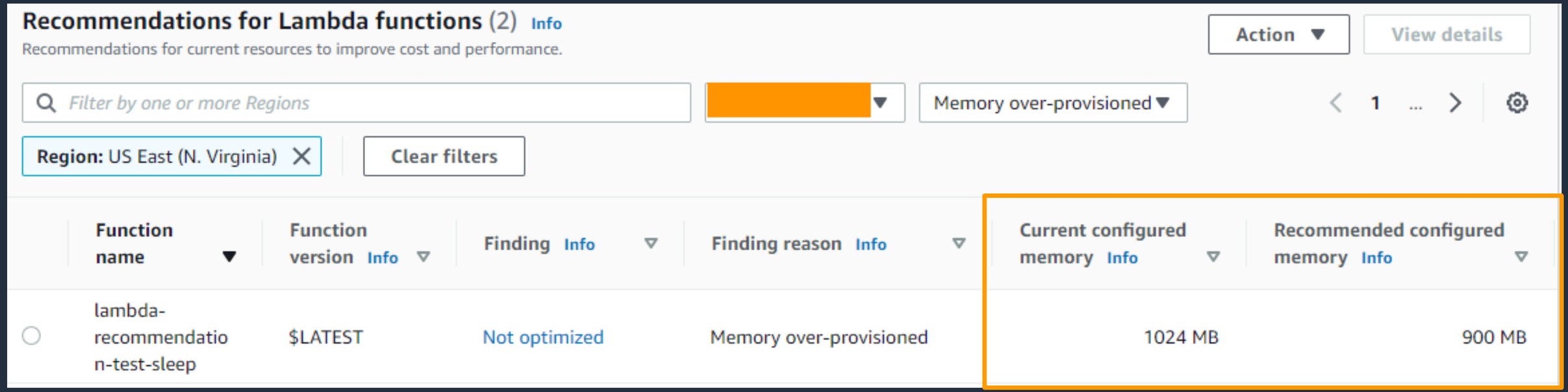
Best Cost	Best Time
1536MB	3008MB

Worst Cost	Worst Time
3008MB	128MB

https://docs.aws.amazon.com/lambda/latest/operatorguide/profile-functions.html

aws

AWS Compute Optimizer



https://aws.amazon.com/blogs/aws/new-for-aws-compute-optimizer-resource-efficiency-metrics-to-estimate-savings-opportunities-and-performance-risks/

Optimiza tu código

Beneficio

Menor tiempo de ejecución.

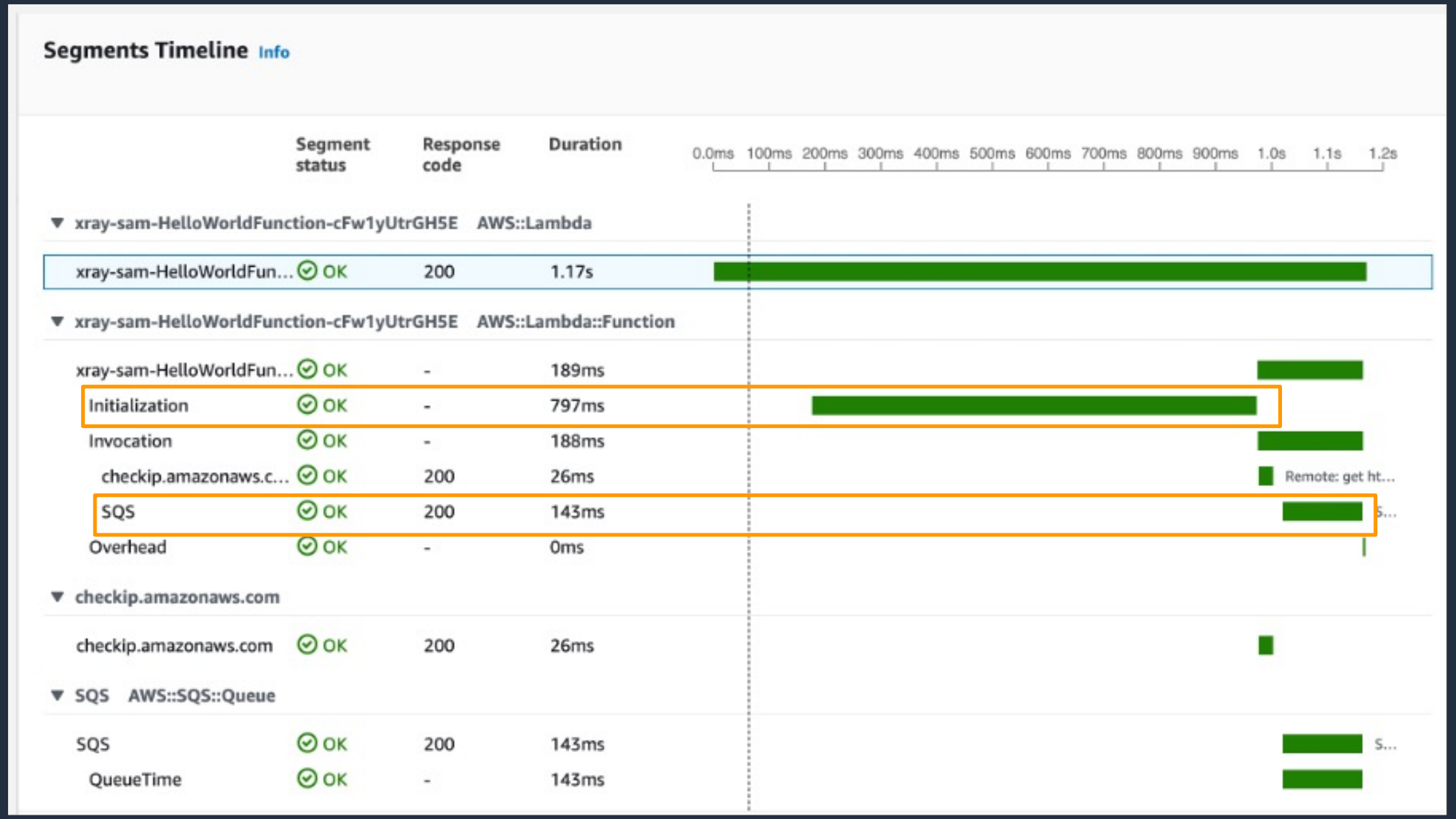
Herramientas

AWS X-Ray, Amazon Code Guru

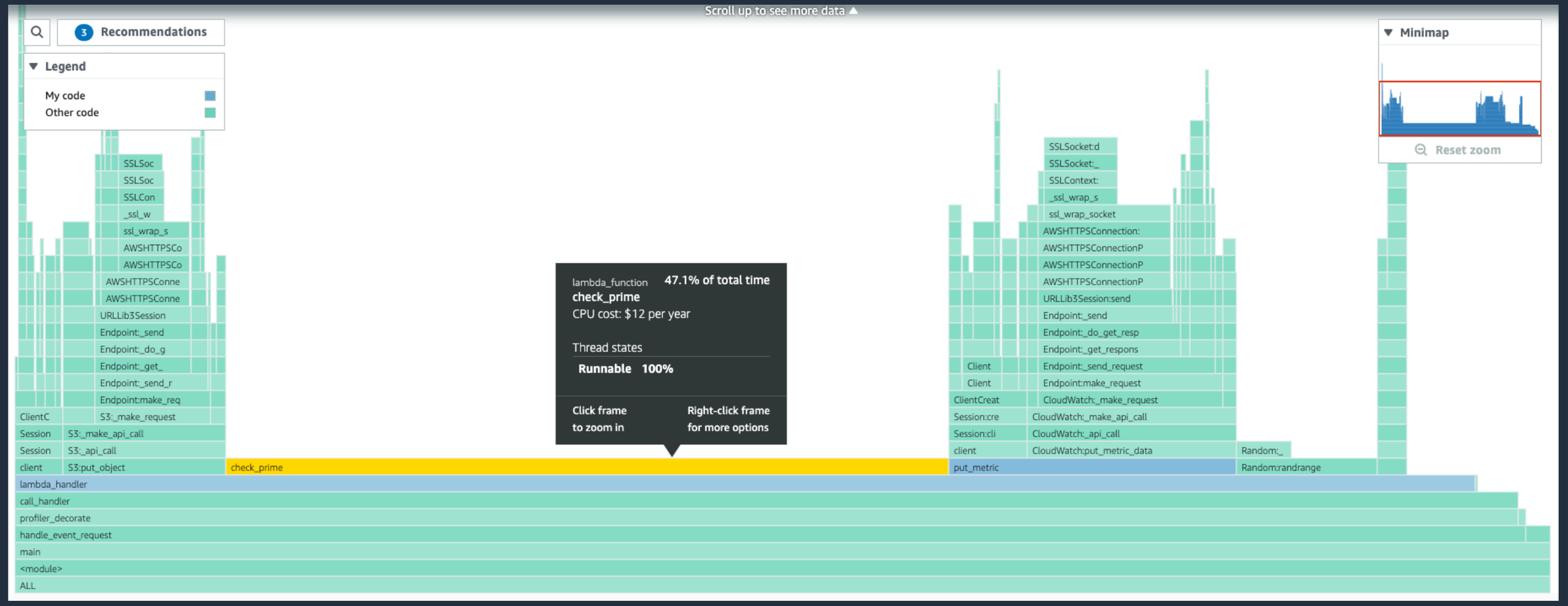
Acción

- Configurar AWS X-Ray: los subsegmentos/anotaciones pueden identificar bottlenecks
- Habilitar CodeGuru- actua sobre recomendaciones para reducir código caro
- Eliminar código y bibliotecas innecesarias
- Apunte primero los cambios que tengan mayor impacto

AWS X-Ray



CodeGuru Profiler



Utilice el modelo de precios óptimos

Beneficio

Precio óptimo para el uso de línea base mensual. Ahorre hasta 17% de los costos de duración vs on-demand

Herramientas

Compute Savings Plan

Acción

- Comprender la línea base mínima mensual esperada
- Comprar Compute Savings Plan para la línea base (al menos para cubrir la concurrencia aprovisionada)

aws

Graviton2

Beneficio

Rendimiento de precio hasta un 34 % mejor

Herramientas

AWS Power Tuning

Acción

- Identificar las cargas de trabajo que podrían beneficiarse de la ejecución en Graviton2
- Probar antes de poner en produccion
- Utilice Lambda Power Tuning para comparar las versiones x86 y arm64

aws

Graviton2

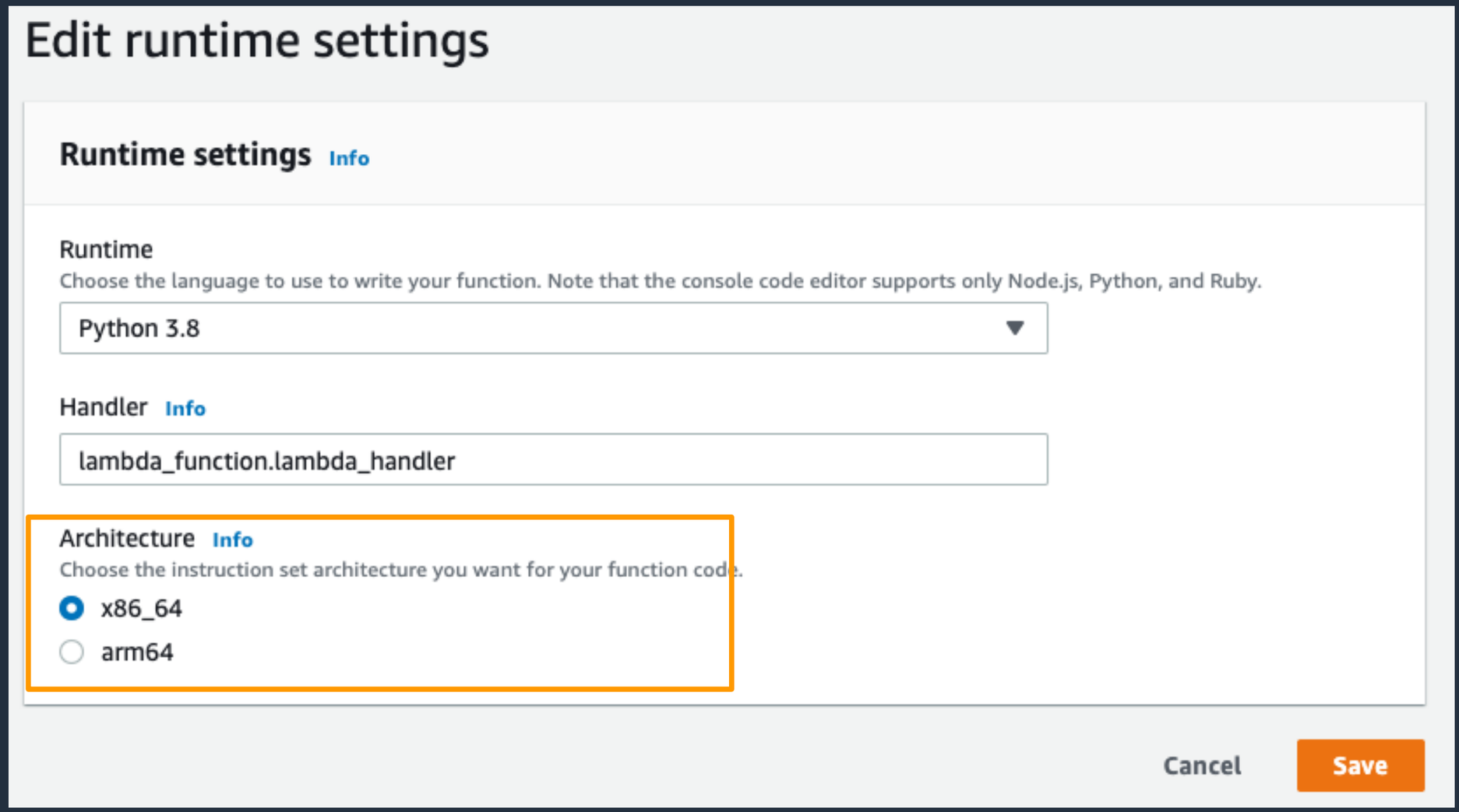
Cargas de trabajo

- Multithreading o realizar muchas operaciones de E/S
- Inferencia de aprendizaje automático basada en la CPUs
- Codificación de vídeo
- Gaming
- Procesamiento de logs

Costo

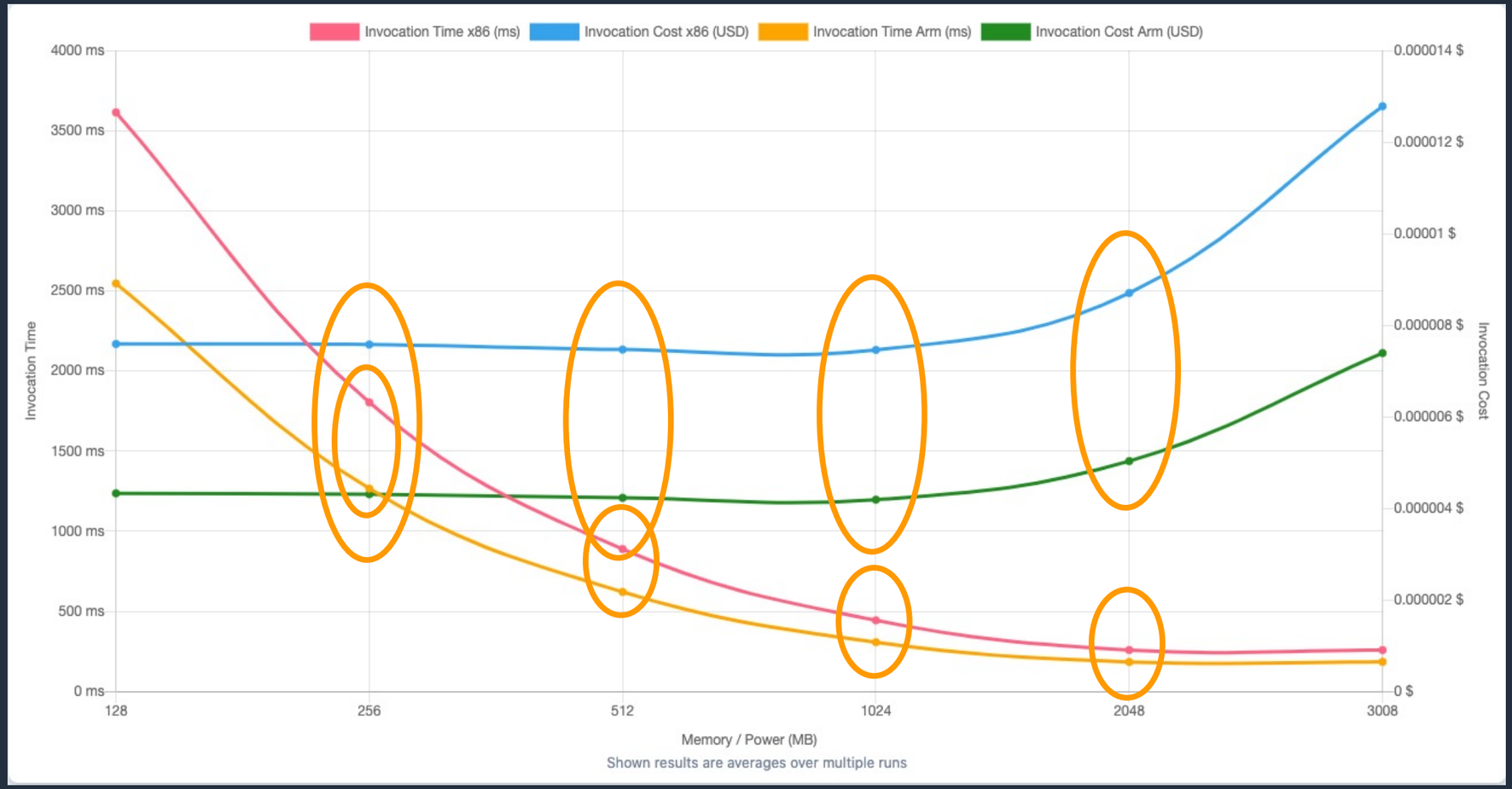
- 20% más económico, incluida la concurrencia aprovisionada
- Apoyado en Compute Savings Plans

Graviton2



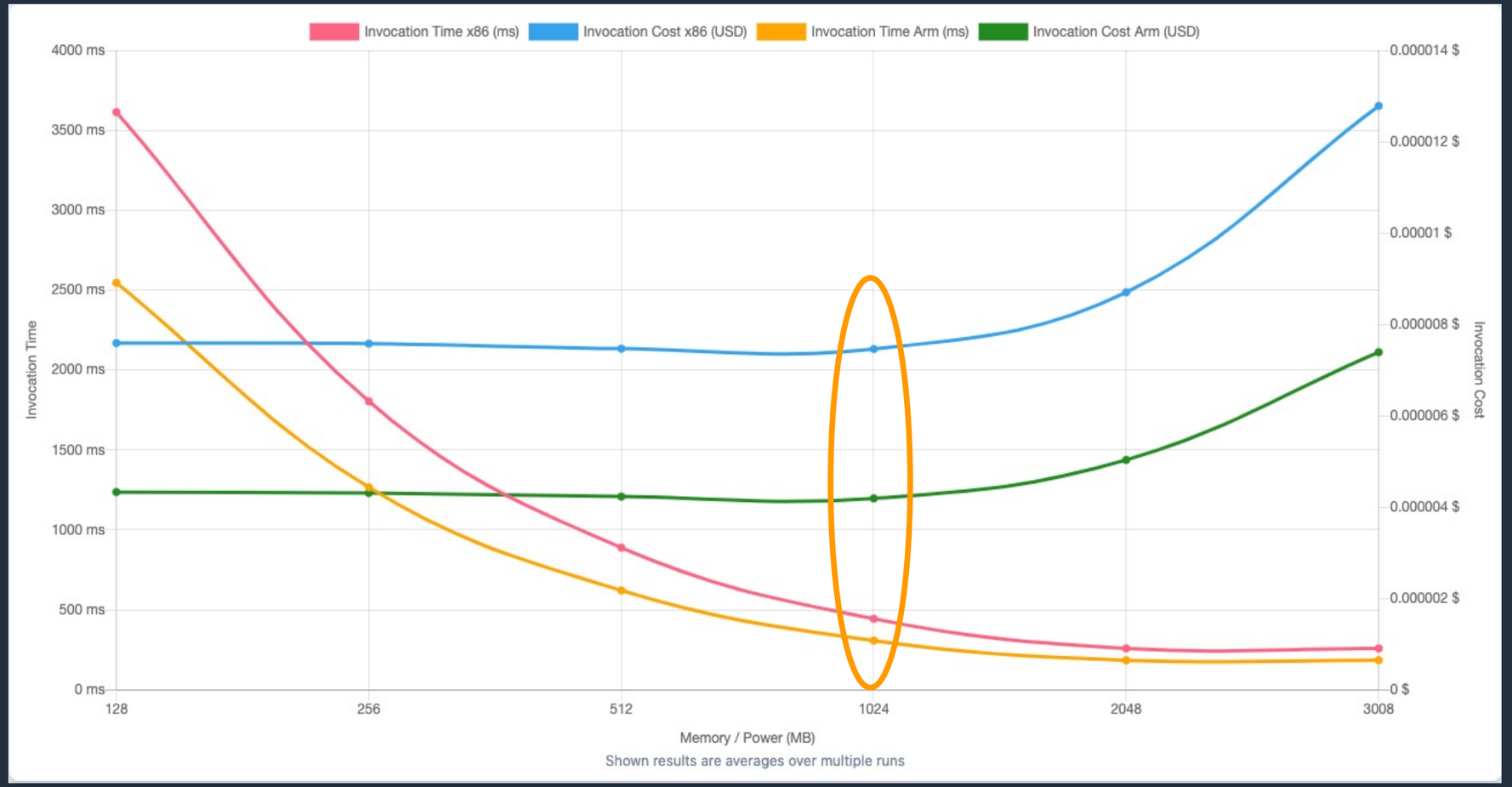
aws

Graviton2



aws

Graviton2



aws

Recursos adicionales

Serverless Cost Optimization Resources

Welcome to the **Serverless Cost Optimization** repository! Here, you'll find a curated list of resources to help you effectively manage and optimize costs for your AWS Lambda-based serverless applications.

Articles and Blog Posts

- When is the Lambda Init Phase Free, and when is it Billed?
- Optimizing your AWS Lambda costs – Part 1
- Optimizing your AWS Lambda costs – Part 2
- AWS Lambda Pricing
- Optimizing AWS Lambda cost and performance using AWS Compute Optimizer
- Building well-architected serverless applications: Optimizing application costs
- Well-Architected Framework Serverless Lens
- The best ways to save money on Lambda
- Understanding techniques to reduce AWS Lambda costs in serverless applications

Documentation and Guides

- Node.js Keep Alive
- Serverless Optimization Workshop (Performance and Cost)

Videos

- Serverless Office Hours | AWS Lambda Cost Optimization

Feel free to explore these resources to gain insights into best practices and strategies for optimizing the costs of your serverless applications running on AWS Lambda. Remember that optimizing costs while maintaining performance is a crucial aspect of building efficient serverless architectures.

LATAM Compute Roadshow

Gracias!

Encuesta 🙏

P⭐

LATAM Compute Roadshow

Gracias!

Encuesta 🙏

C⭐⭐