JAPAN | JUNE 21, 2024

aws SUMMIT

DOL-13

Rust で Lambda はどれだけ高速化するのか "Rustacean Lambda"

Yoshitaka Kawaji

Solutions Architect
Amazon Web Services Japan K.K.

本セッションの概要

AWS Lambda に Rust がどのように寄与し、高速化につながるのかについての紹介を行います。

AWS における Rust を利用した高速化の取り組みの紹介。

自己紹介

川路 義隆

𝕏 @kawaji_scratch

アマゾン ウェブ サービス ジャパン合同会社
ソリューションアーキテクト

- 小売業のお客様の支援
- サーバーレス導入支援

Agenda

Lambda を高速化する動機
Rust を Lambda 関数で利用する方法
Lambda Extenstions
AWS による実験的取り組み
まとめ

Lambda を高速化する動機

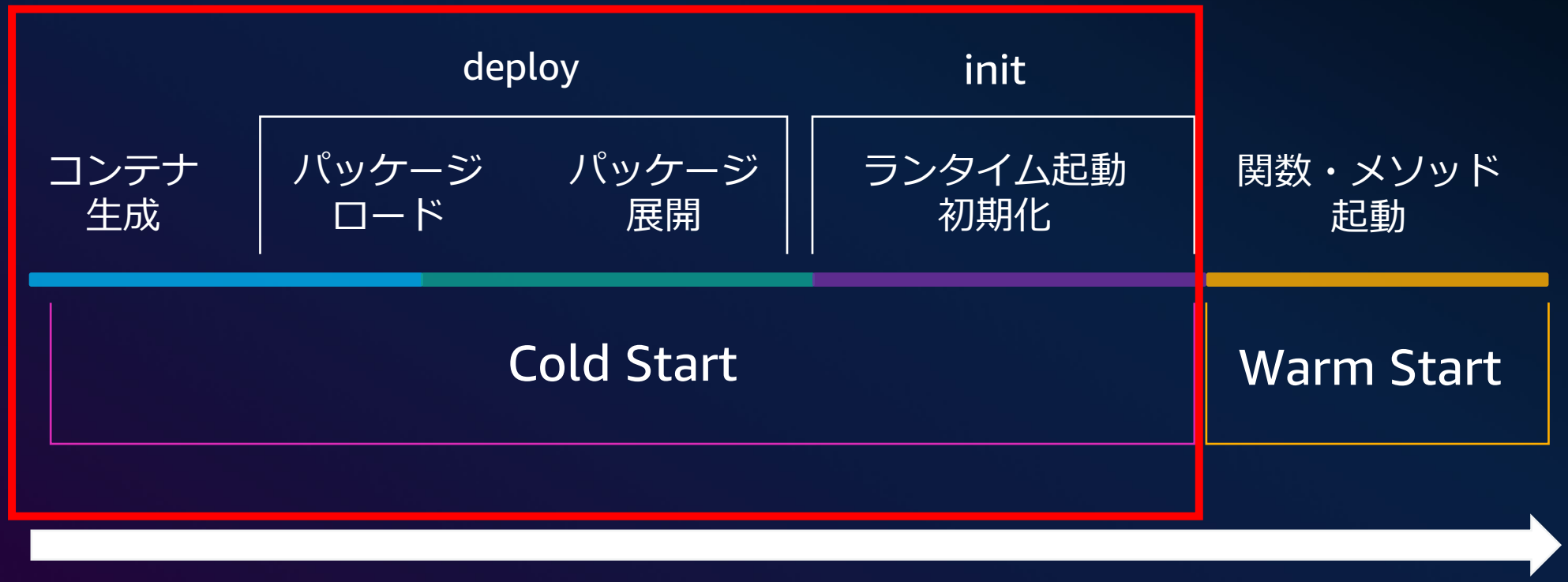
Lambda を高速化する動機

必要な処理を素早く起動し、
迅速に処理を終わらせたい。

可能な限り低コストで利用したい。

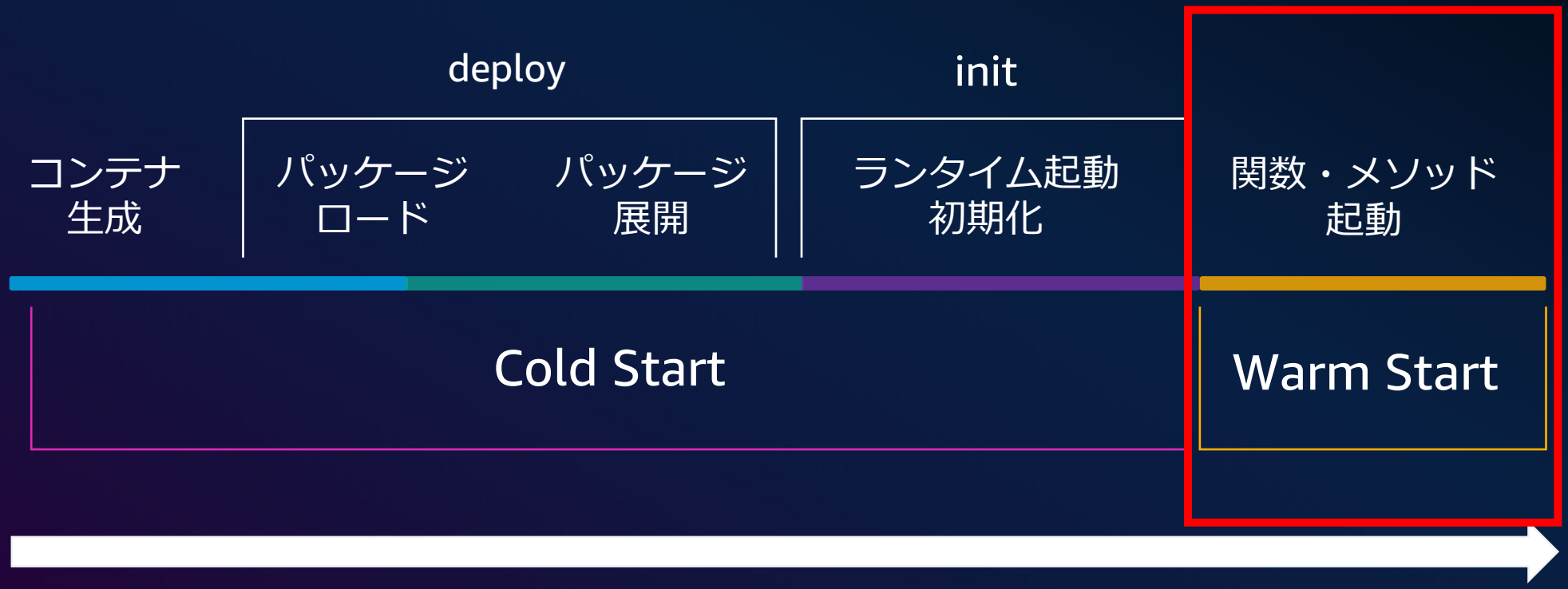
起動時間

Lambda 実行環境は Cold Start することがあり関数実行前に起動時間が発生する。



実行時間

Lambda 関数の実行時間は処理内容の他に言語ランタイムに依存します。



コスト

AWS Lambda の料金は実行時間(ミリ秒)で計算されます

メモリ (MB)	1ミリ秒あたりの料金	メモリ (MB)	1ミリ秒あたりの料金
128	0.0000000021USD	1,536	0.0000000250USD
512	0.0000000083USD	2,048	0.0000000333USD
1,024	0.0000000167USD	3,072	0.0000000500USD

東京リージョン x86料金 より抜粋　https://aws.amazon.com/jp/lambda/pricing/

Lambda を高速化するには

Cold Start の起動時間を短縮 + 実行時間の短縮を
検討することが必要です。

Rust とは

Rust とは

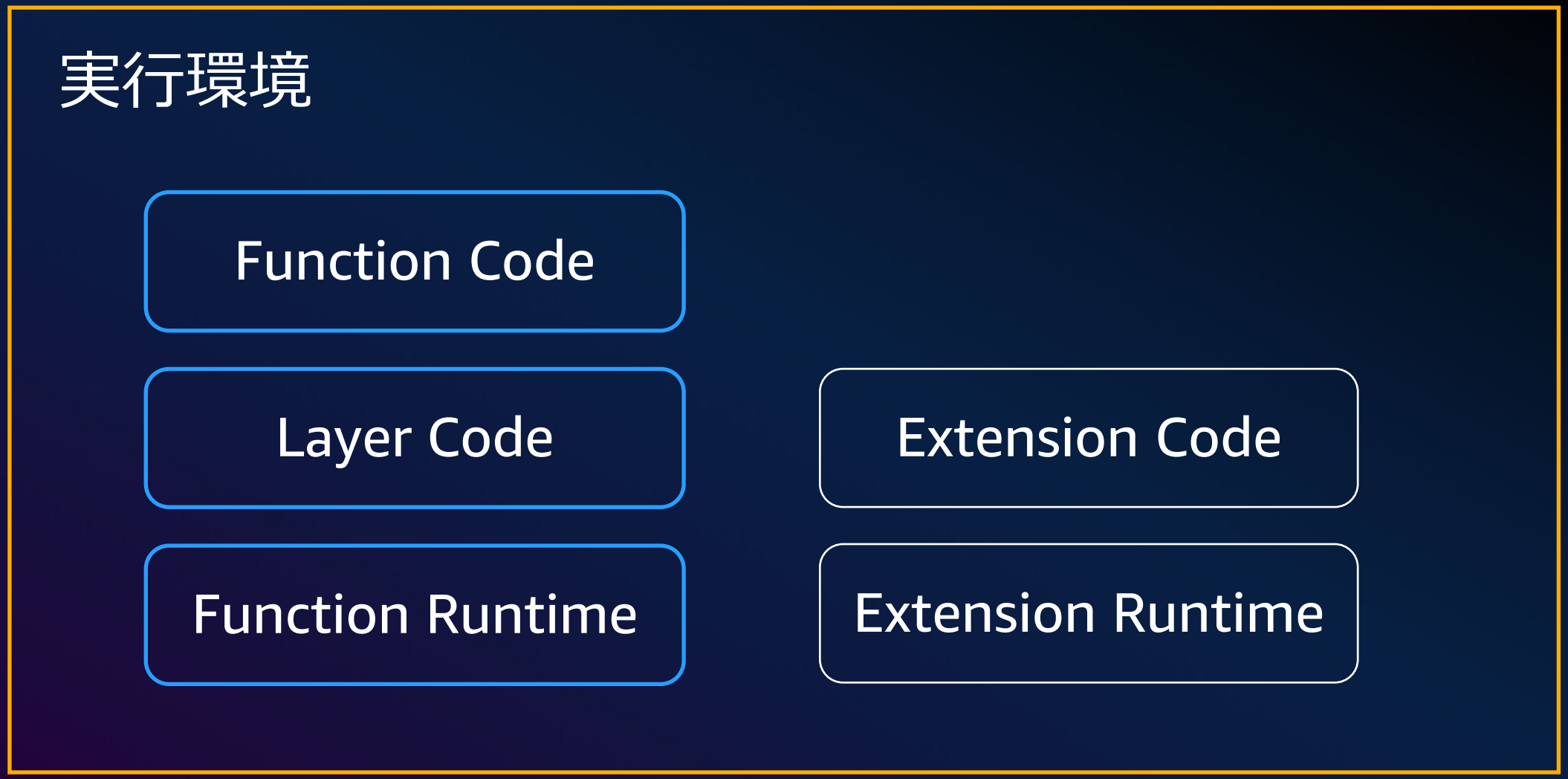
- パフォーマンス
  - 非常に高速
  - メモリ効率が高い
  - ランタイムやガベージコレクタがない

Rust を利用したことで、実際の運用環境において CPU 使用量が 75% 減少し、メモリ使用量が 95% 減少しました

Alan Ning
Tenable.io

Rust を Lambda 関数 で利用する方法

Lambda 実行環境



Rust を Lambda 関数 で利用する方法

Rust 実行可能バイナリにネイティブコンパイルするため、
専用のランタイムは必要ありません。

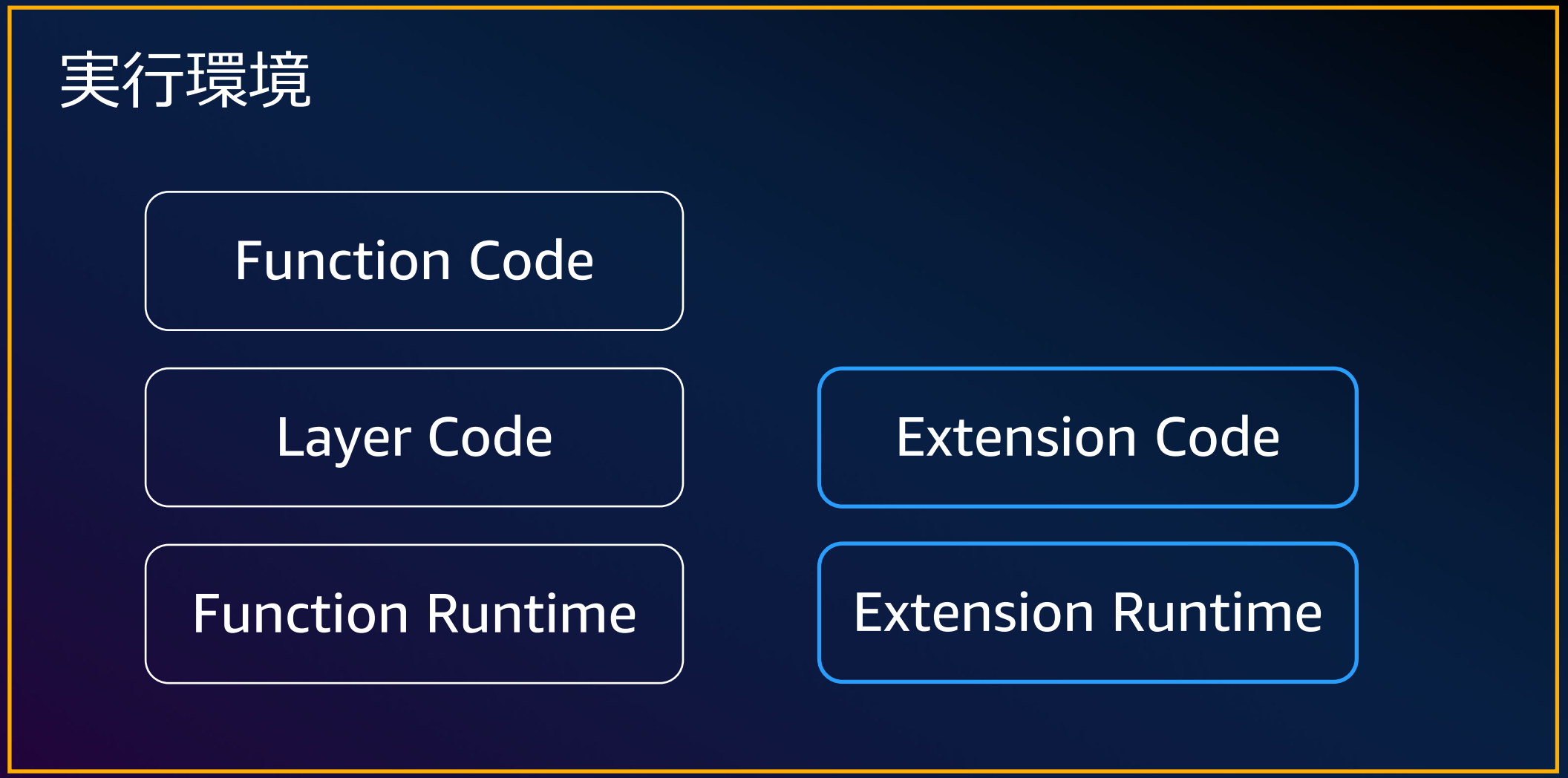
provided.al2023 または provided.al2 ランタイムを使用して
デプロイするか Container Image を作成する方法があります。

OS 専用ランタイム	オペレーティングシステム
provided.al2023	Amazon Linux 2023
provided.al2	Amazon Linux 2

Custom Runtime
(bootstrap + Handler)
OR
Container Image

Lambda Extensions

Lambda 実行環境



Lambda Extensions

Lambda Extensions は、お好みのモニタリング、オブザーバビリティ、セキュリティ、ガバナンス用ツールと Lambda との統合を簡単にする方法。

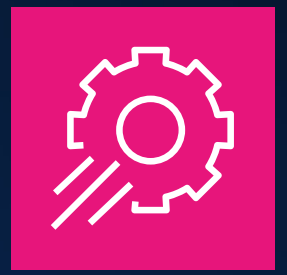
既存ツールを複雑なインストール、設定、運用のオーバヘッドをなくし、関数コードの削減を実現することが可能。

Extension は Lambda とリソース (CPU, Memory) を共有する

一例

DATADOG

Amazon CloudWatch
Lambda Insights

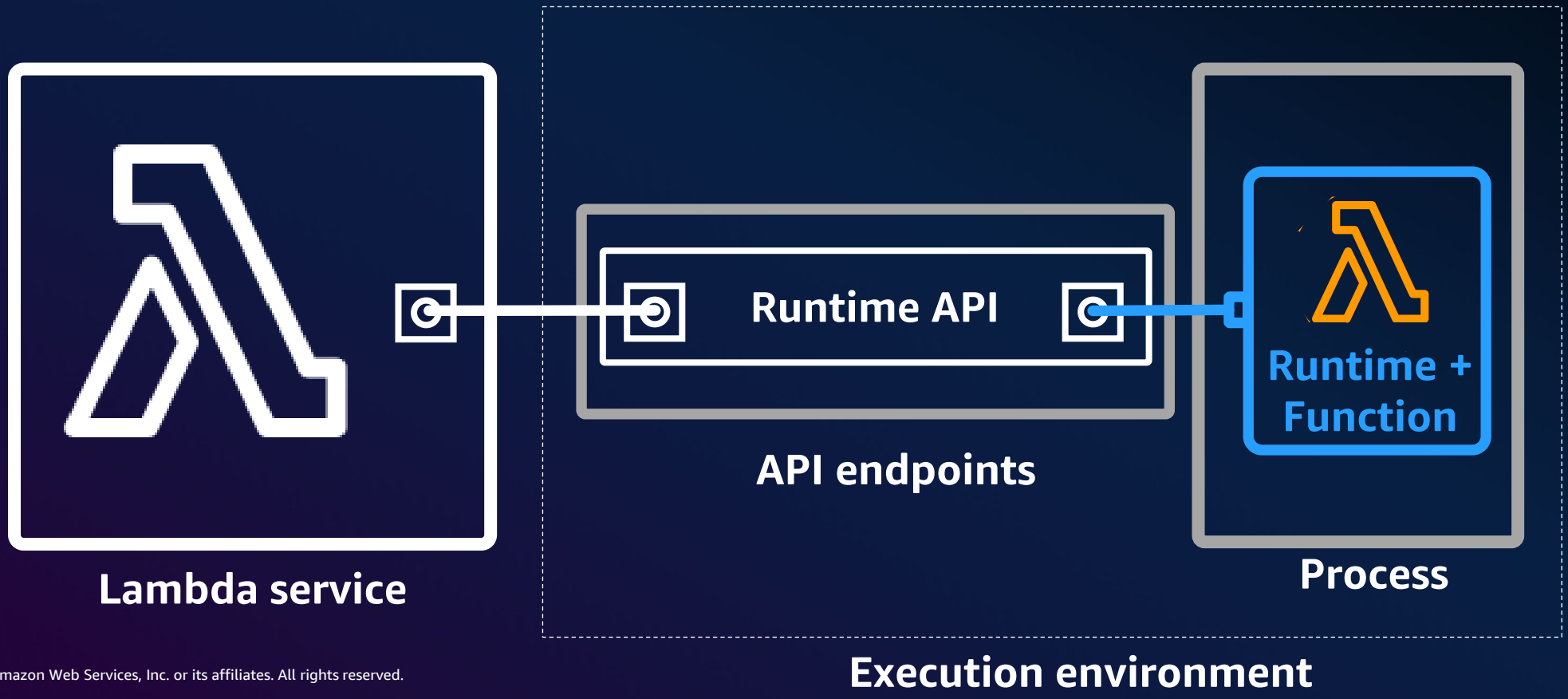


AWS AppConfig

Lambda 実行環境

BEFORE EXTENSIONS

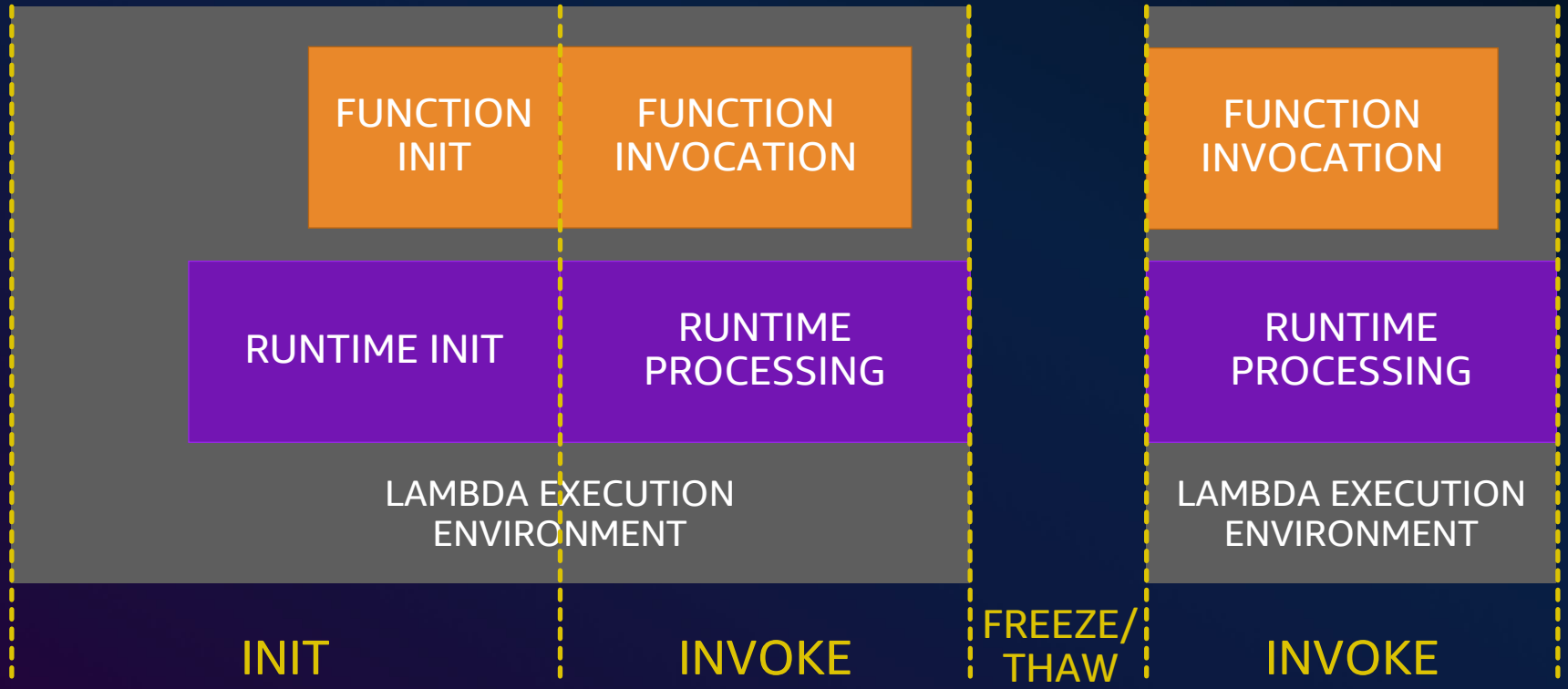
Lambda Runtime API はカスタムランタイムの持ち込みを可能にします



Lambda 実行環境

BEFORE EXTENSIONS

Lambda Runtime API はカスタムランタイムの持ち込みを可能にします



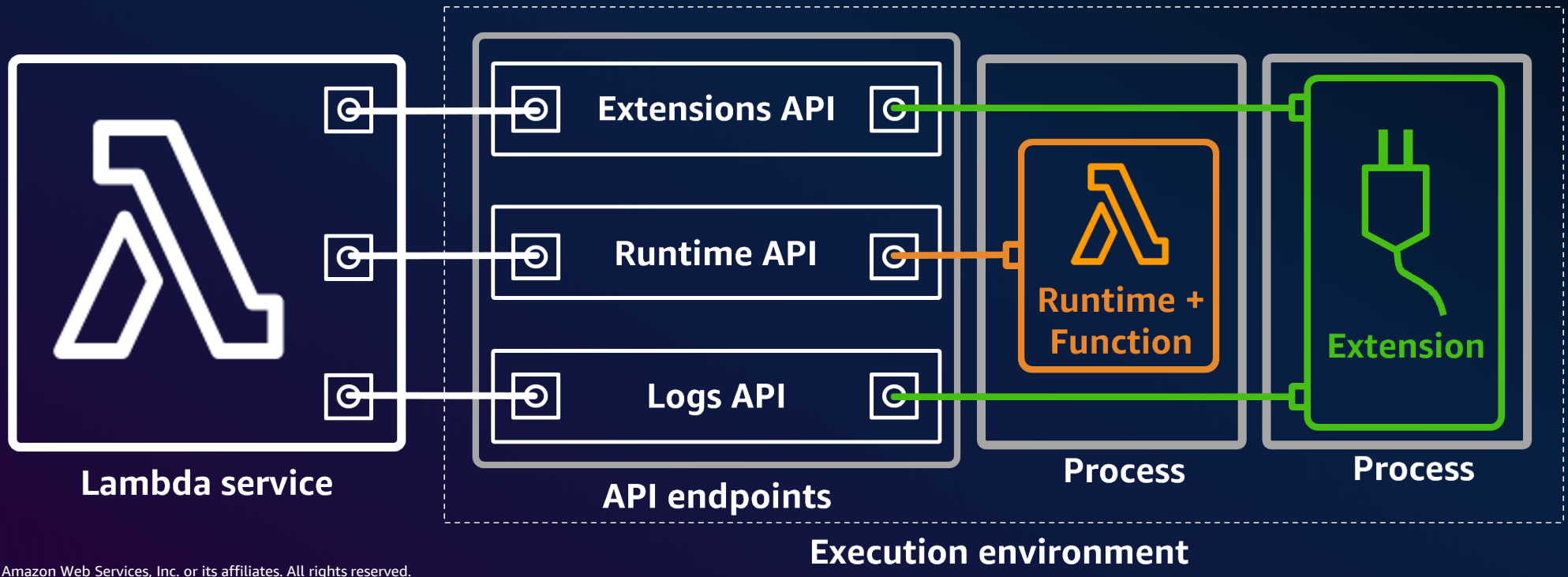
Lambda 実行環境

WITH EXTENSIONS

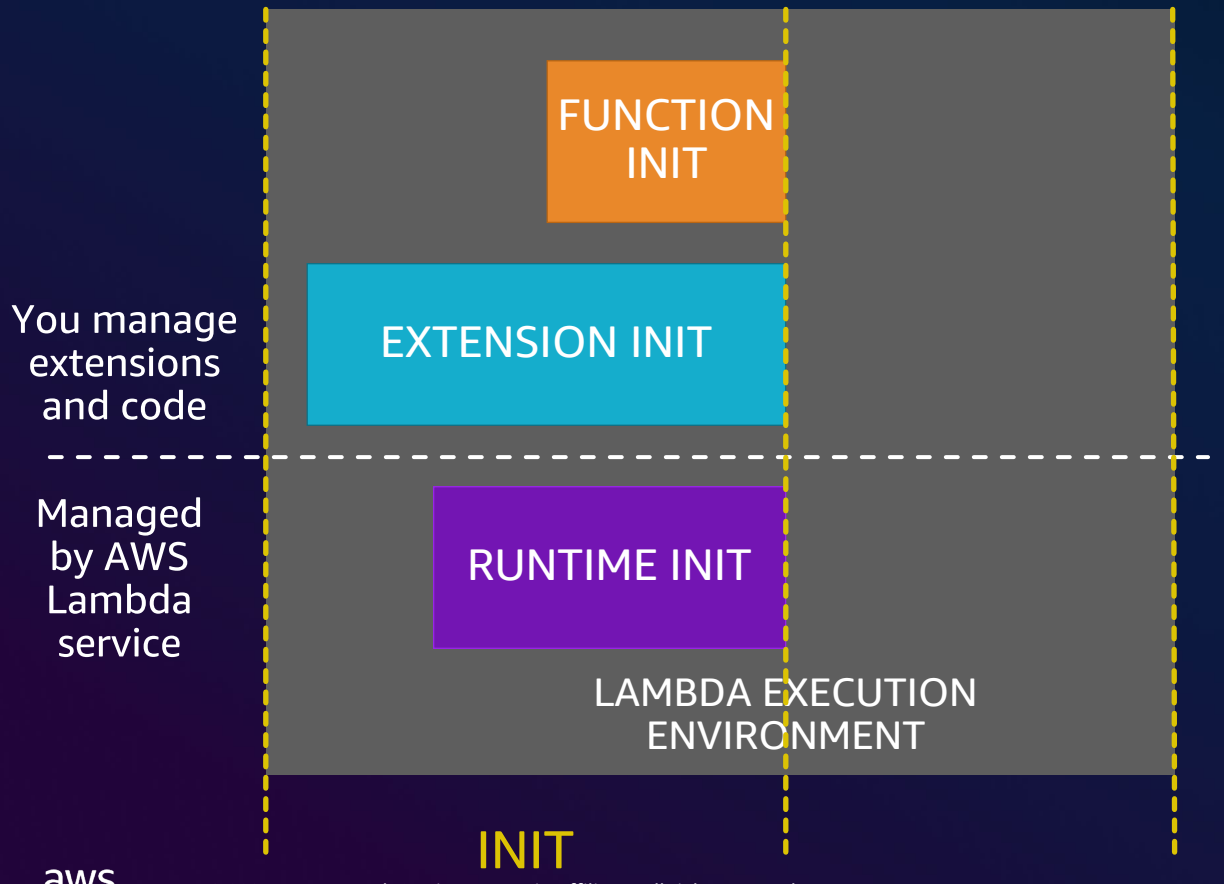
Extensions API は 既存の Lambda Runtime API をベースに構築されている

関数を実行環境のライフサイクルイベントに登録できます

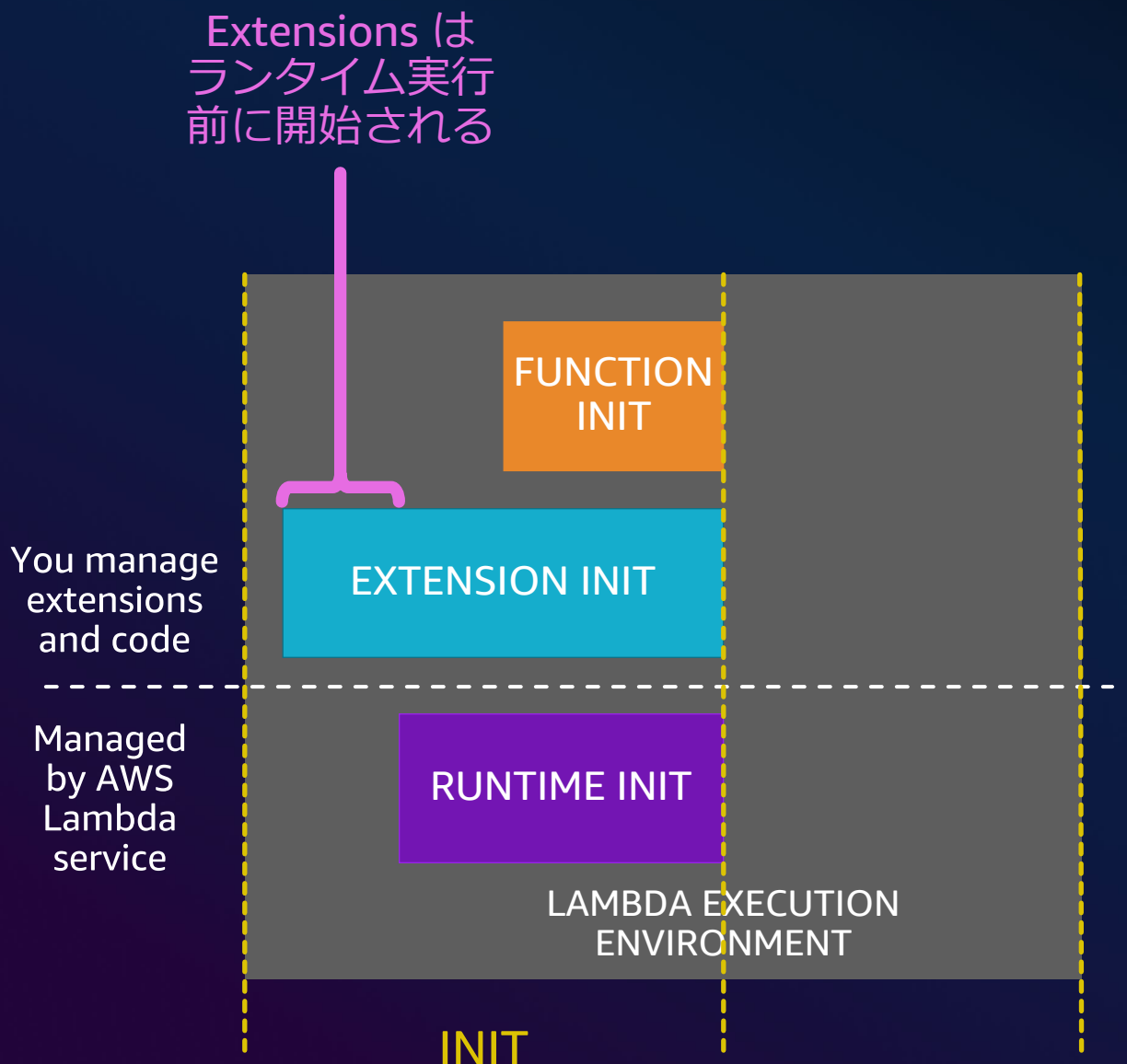
Extensions は Logs API と連携し Lambda からの Log を直接受け取ることが可能です



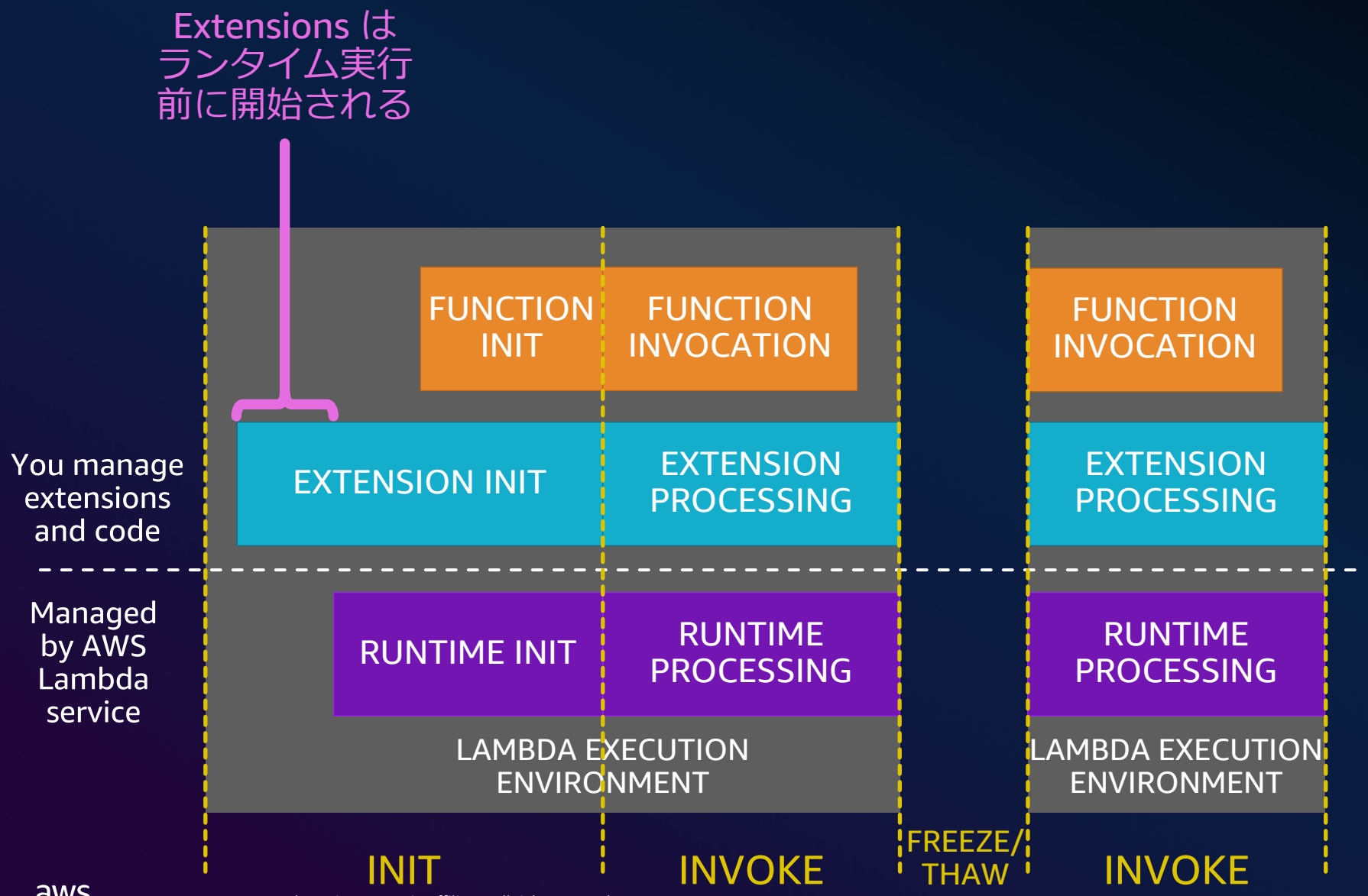
Lambda 関数と Extension のライフサイクル



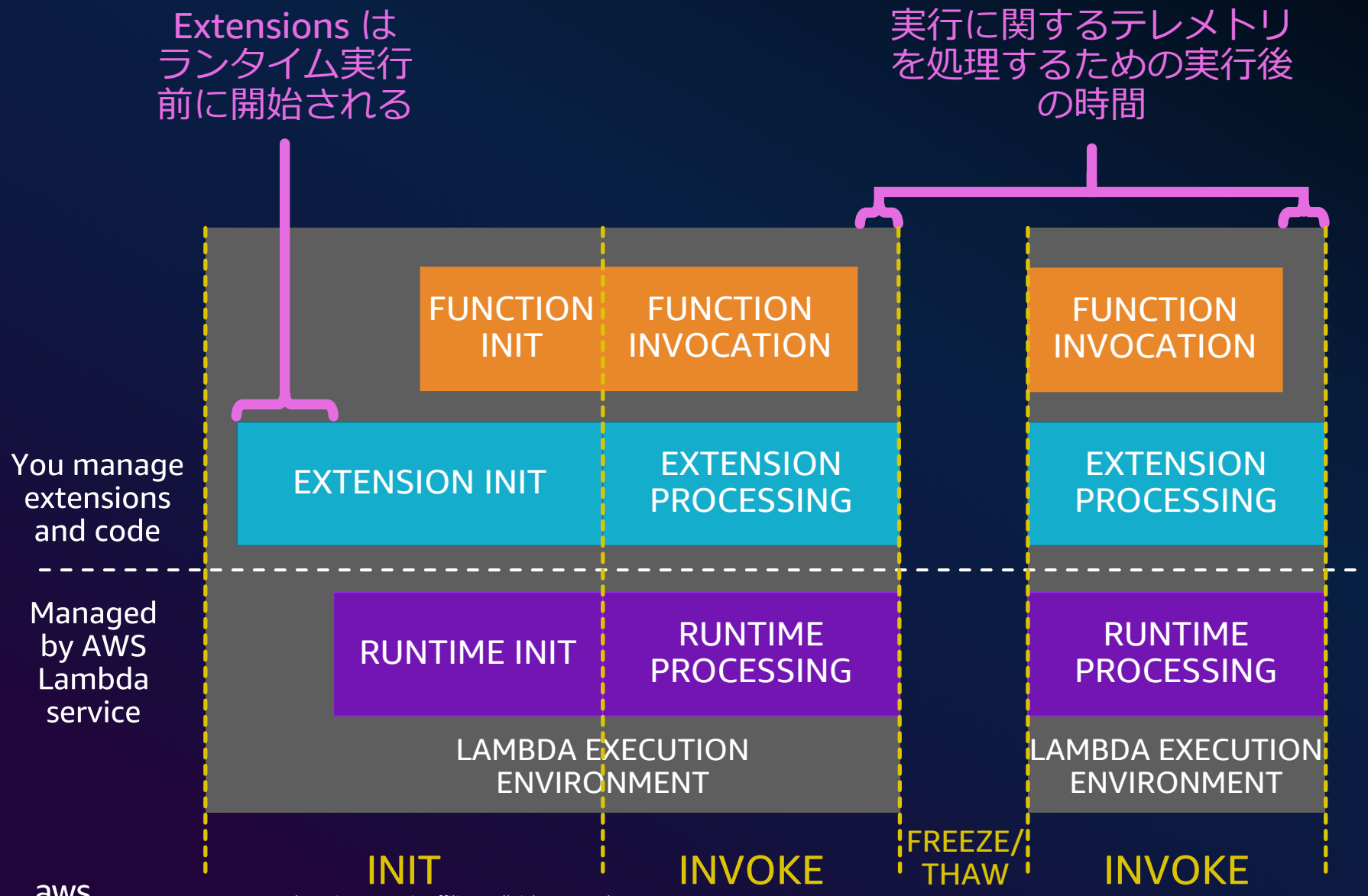
Lambda 関数と Extension のライフサイクル



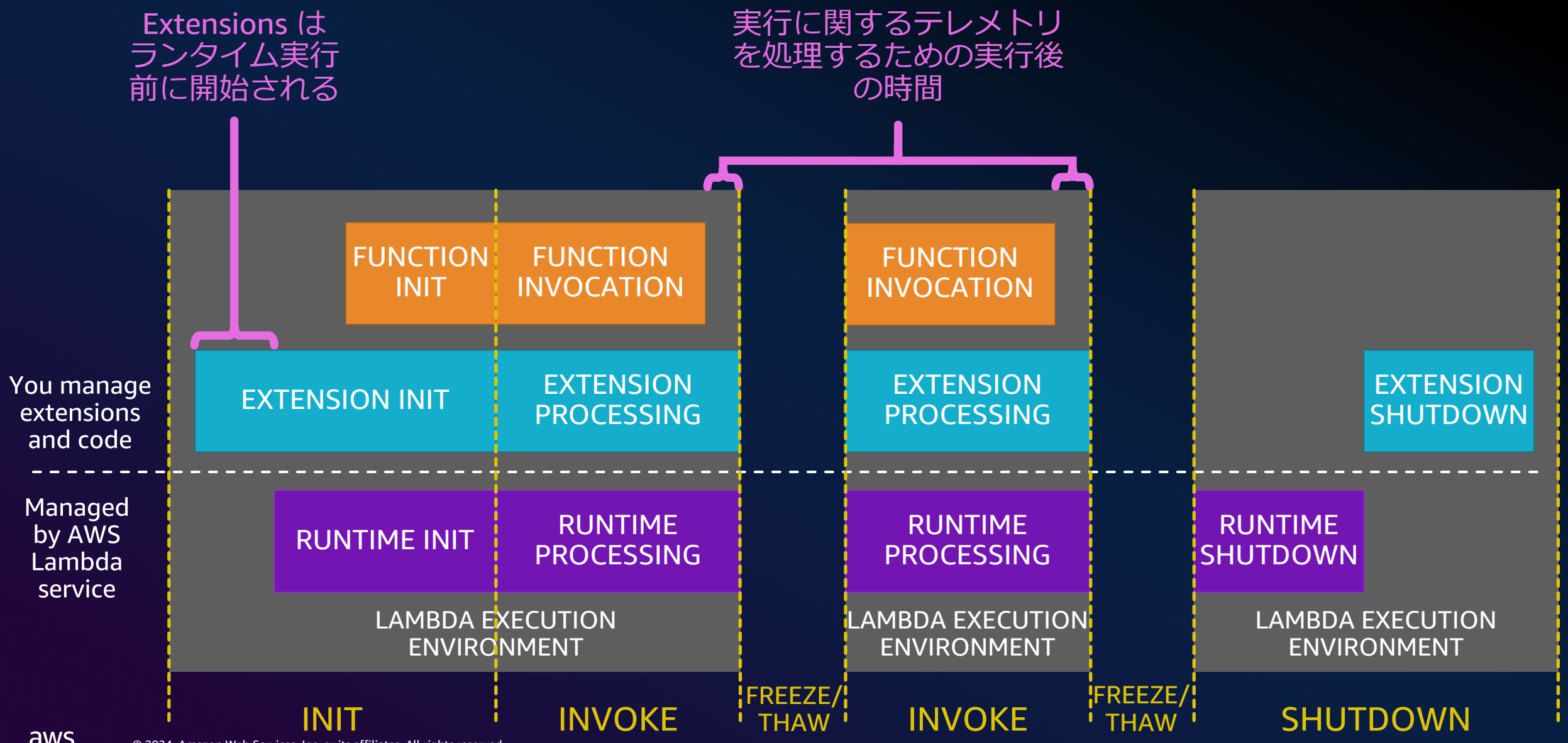
Lambda 関数と Extension のライフサイクル



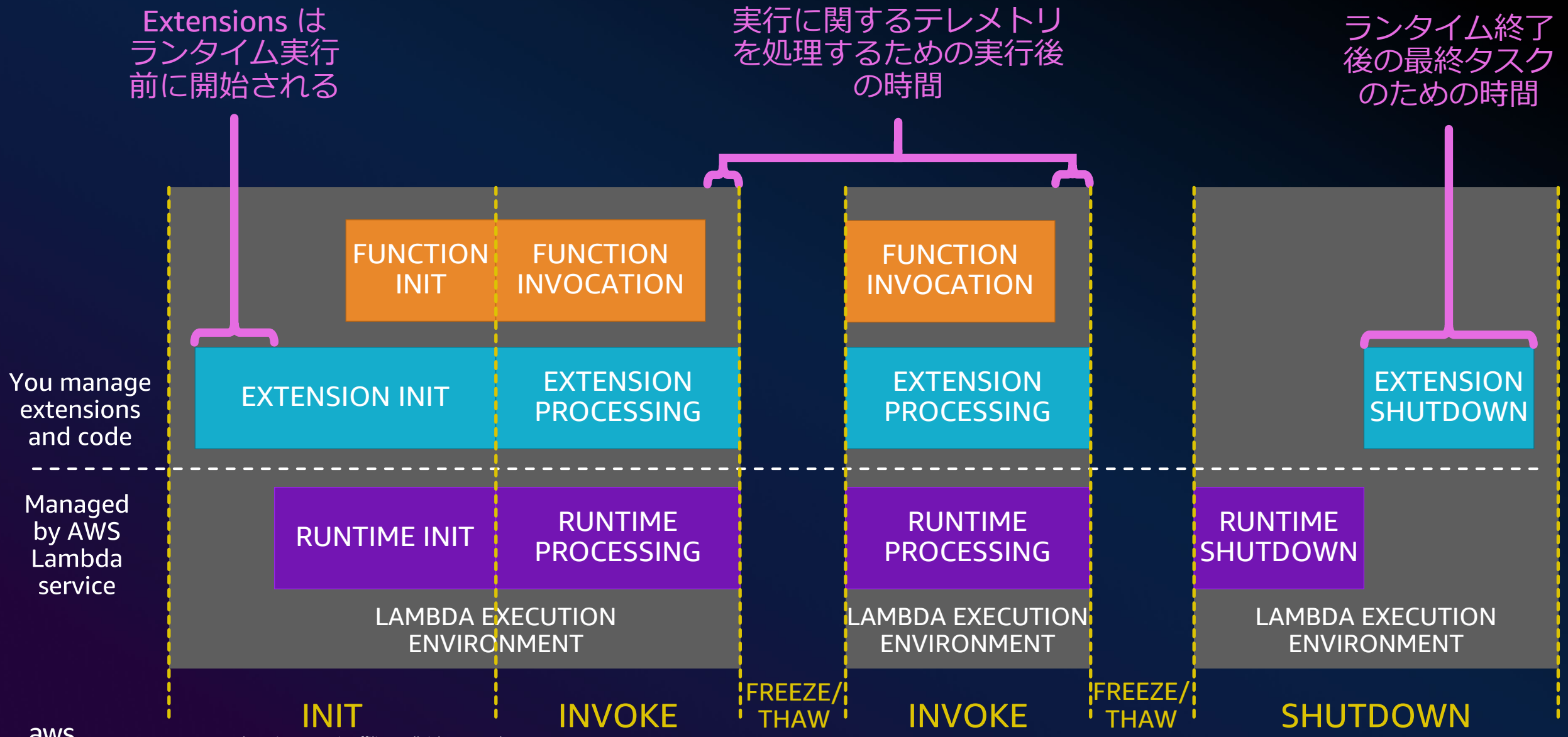
Lambda 関数と Extension のライフサイクル



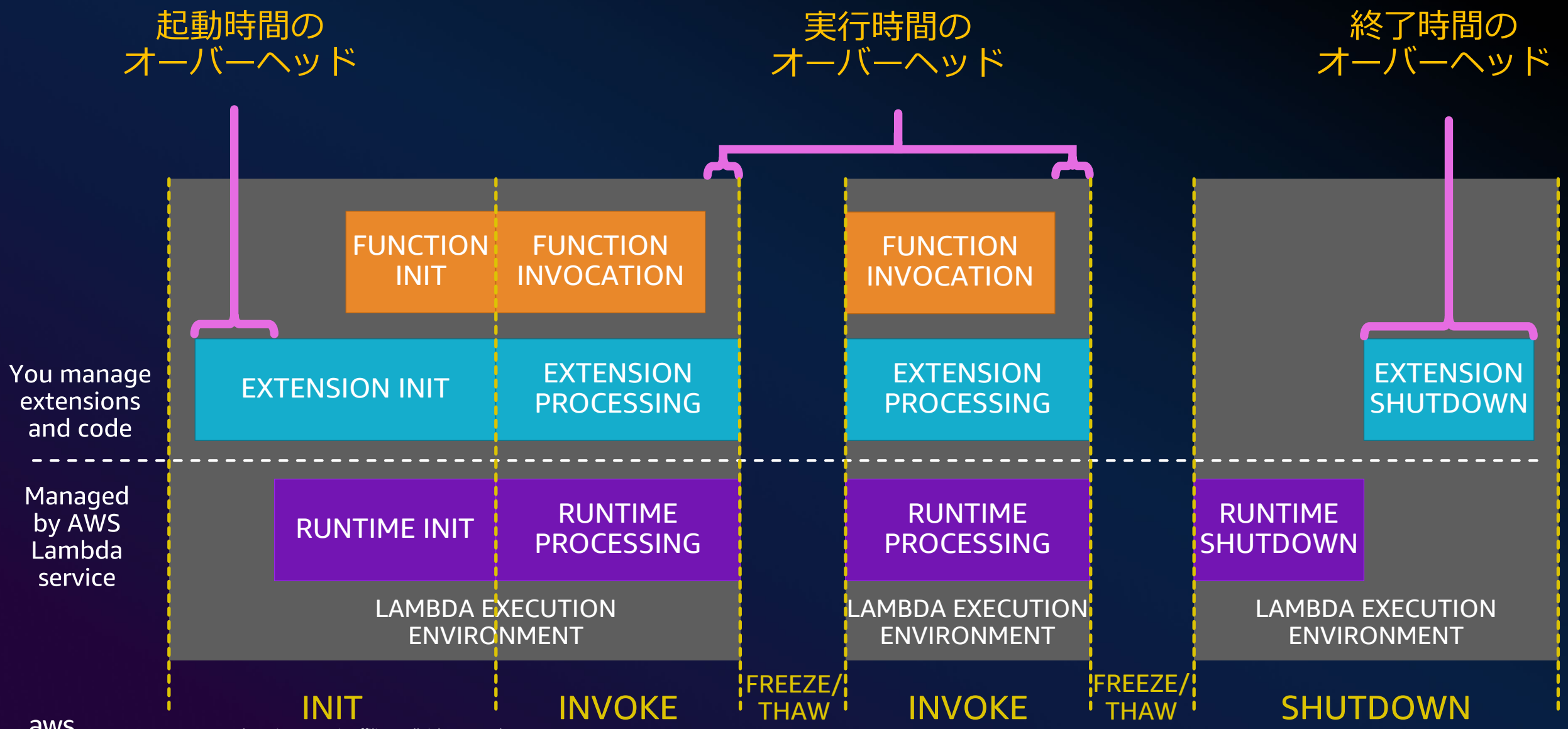
Lambda 関数と Extension のライフサイクル



Lambda 関数と Extension のライフサイクル

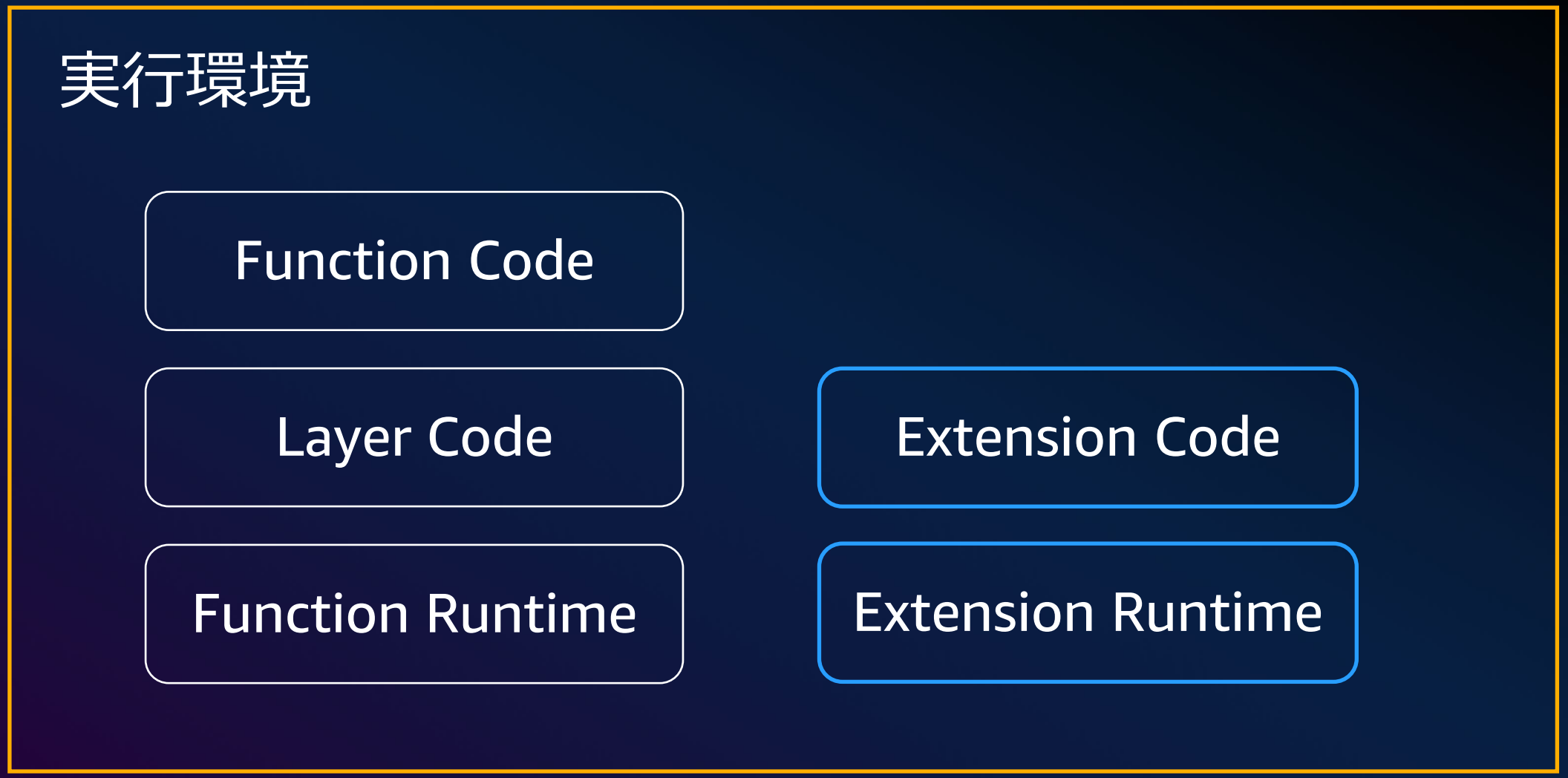


Lambda 関数と Extension のライフサイクル



Lambda Web Adapter

Lambda 実行環境



Lambda Web Adapter (Python + Flask の場合)

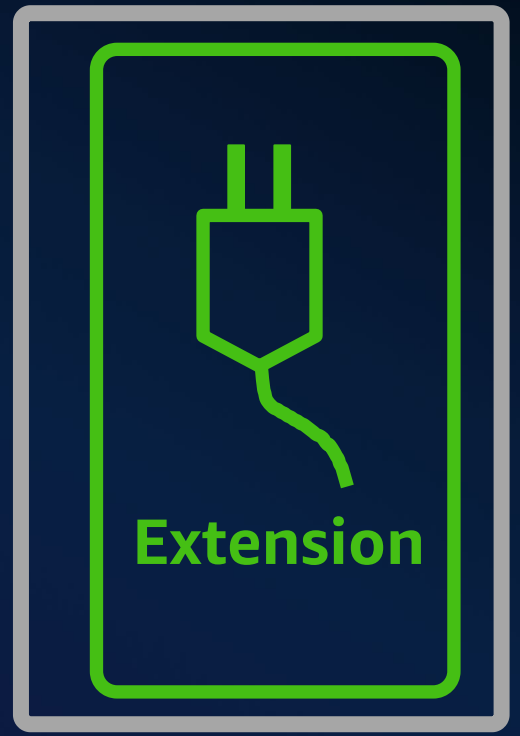
- Lambda Event を HTTP Request に変換する Adapter を組み込む

Python + awsgi

- Lambda Web Adapter を利用

Python

Rust



Lambda Web Adapter (Python + Flask の場合)

Python で実装

- Lambda Event を HTTP Request に変換する Adapter を組み込む

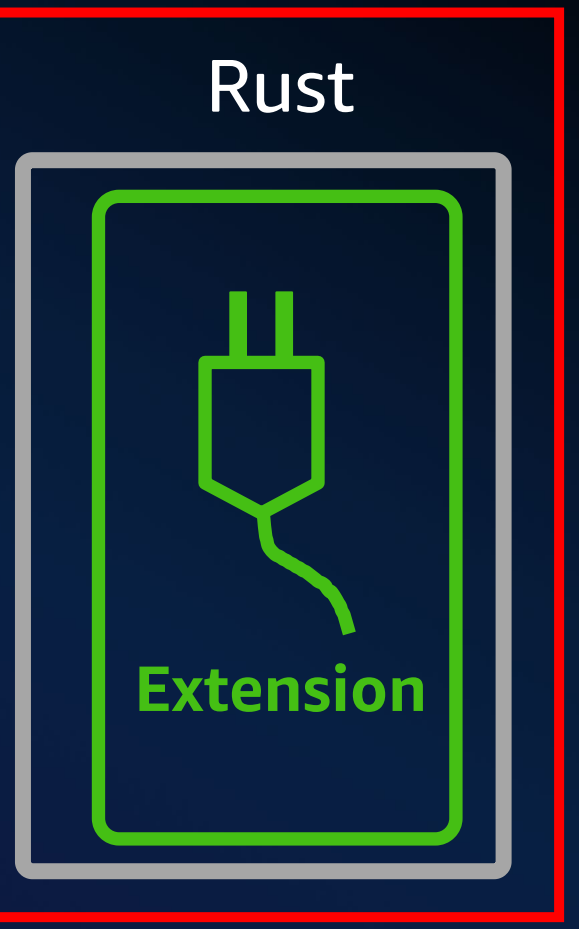
Python + awsgi

Extension を Rust で実装

- Lambda Web Adapter を利用

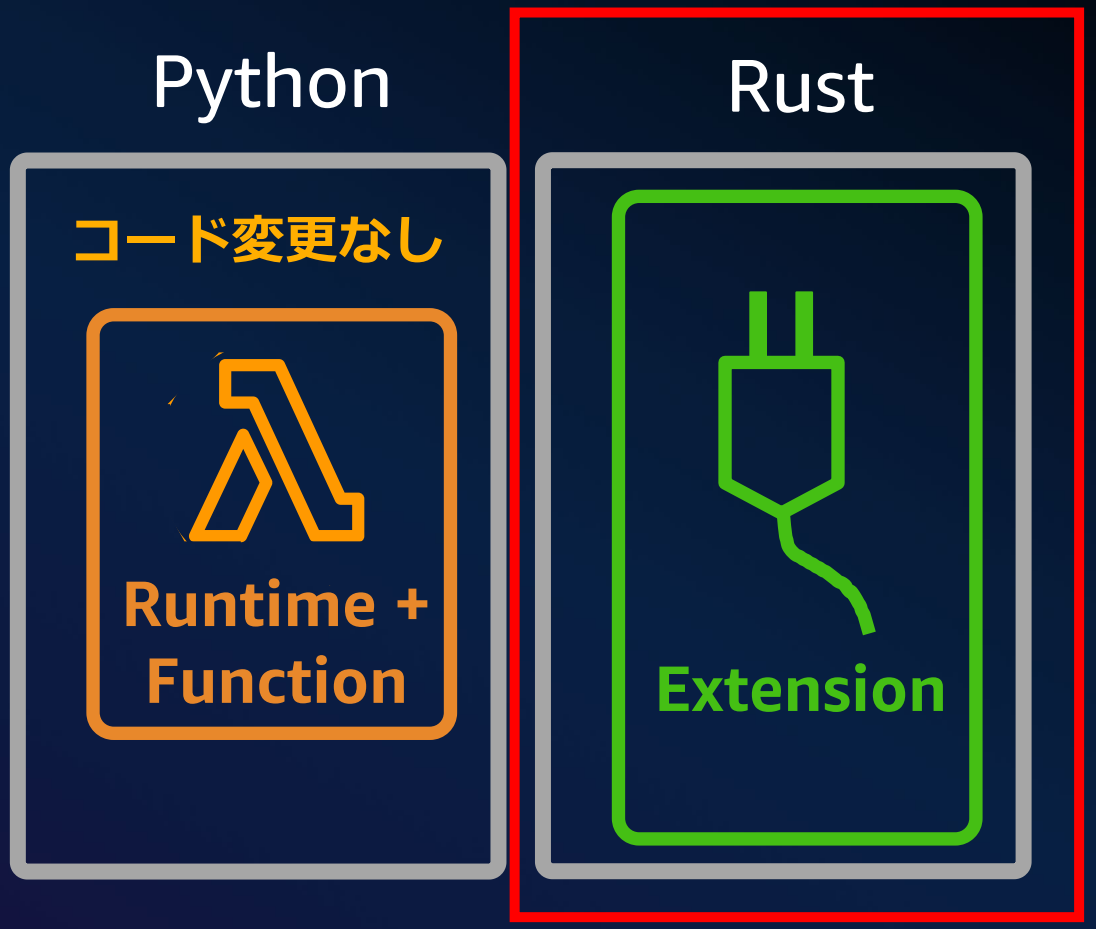
Python

Rust



Lambda Web Adapter (Python + Flask の場合)

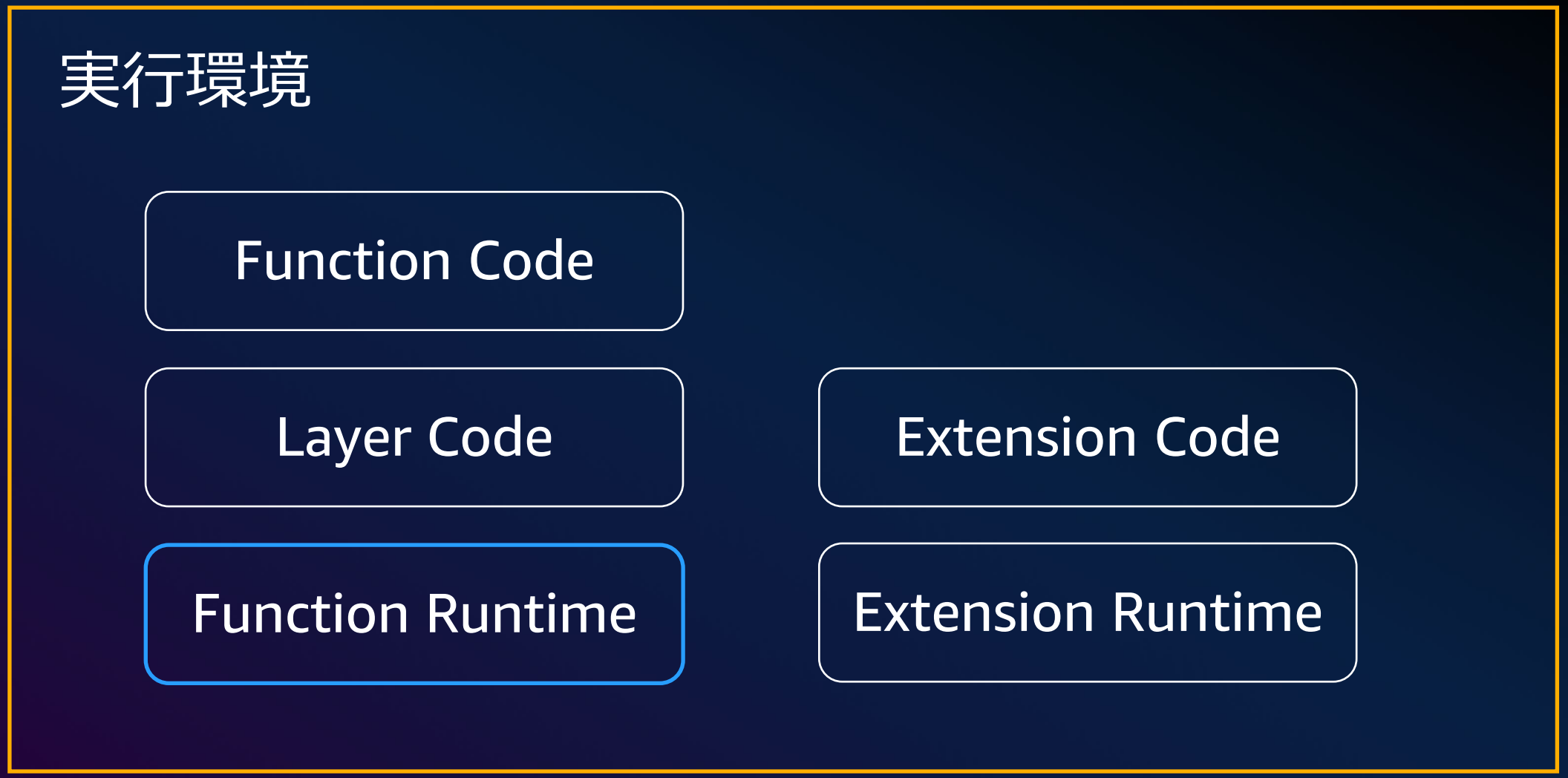
- 既存コードの再利用性を Extension により実現
- Extension のオーバヘッドを Rust 実装により軽減



LLRT(Low Latency Runtime)

AWS による実験的取り組み

Lambda 実行環境



AWS による実験的取り組み

LLRT（Low Latency Runtime）は、 Rust で構築された軽量 JavaScript ランタイムです。 (JIT を取り除いている為 Rust だけの軽量化ではない)

Cold starts: 100.00% (64/64)

(index)	p0	p25	p50	p75	p90	p95	p99	p100
λ	'1457.93'	'1492.26'	'1511.93'	'1538.59'	'1563.80'	'1599.94'	'1660.68'	'1678.28'
HTTP	'1575.16'	'1687.36'	'1719.00'	'1740.46'	'1756.37'	'1844.19'	'2463.06'	'3072.96'

Warm starts: 100.00% (20000/20000)

(index)	p0	p25	p50	p75	p90	p95	p99	p100
λ	'6.10'	'21.37'	'33.20'	'51.27'	'74.44'	'106.69'	'193.63'	'599.87'
HTTP	'27.12'	'51.17'	'62.08'	'79.97'	'105.44'	'137.67'	'225.06'	'627.71'

Node.js 20

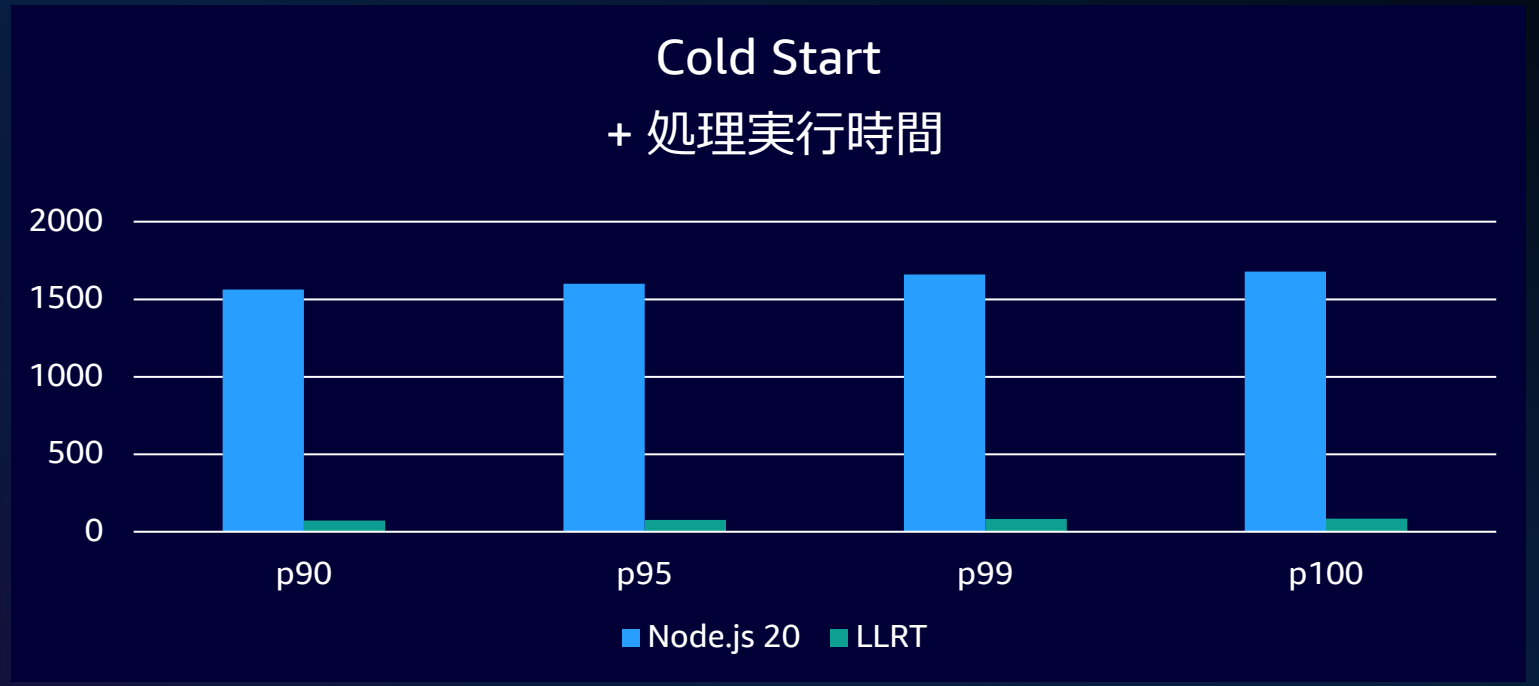
Cold starts: 100.00% (64/64)

(index)	p0	p25	p50	p75	p90	p95	p99	p100
λ	'48.54'	'58.08'	'64.16'	'69.98'	'72.62'	'76.33'	'83.17'	'85.61'
HTTP	'155.37'	'201.67'	'235.61'	'247.57'	'253.87'	'261.37'	'285.32'	'304.57'

Warm starts: 100.00% (20000/20000)

(index)	p0	p25	p50	p75	p90	p95	p99	p100
λ	'5.03'	'9.94'	'14.94'	'20.81'	'31.50'	'33.40'	'35.79'	'236.86'
HTTP	'27.14'	'39.04'	'43.72'	'54.67'	'60.48'	'62.47'	'68.61'	'272.19'

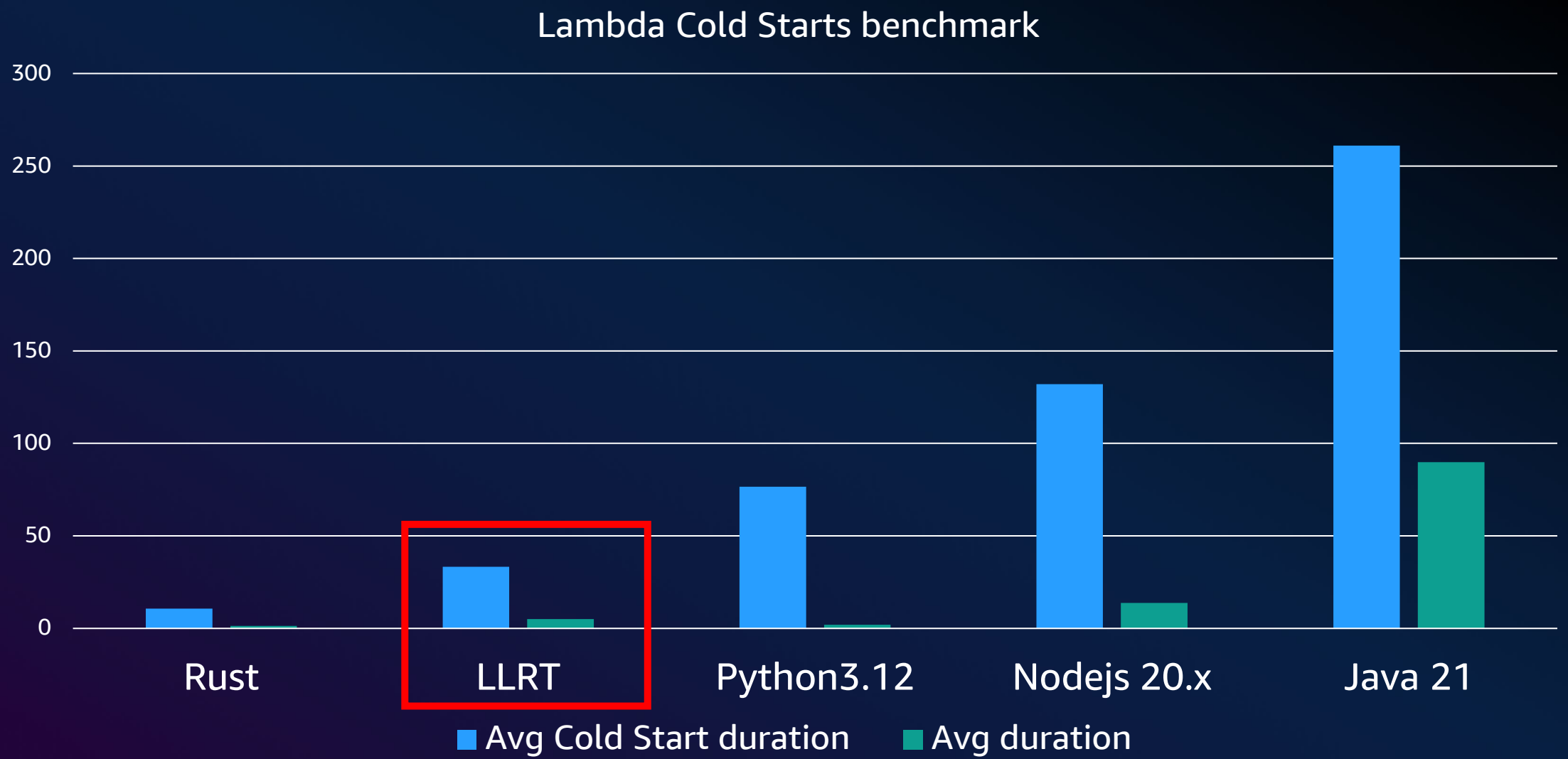
LLRT



最大 10 倍以上の高速起動 & 最大 2 倍のコスト削減

こちらで公開されているコードをカスタマイズ利用 https://maxday.github.io/lambda-perf/

計測結果



作業者の環境で計測した結果であり公式の値ではありません

まとめ

- Lambda を高速化するメリットは 実行時間を短くするコスト削減以外に Cold Start の起動時間を短縮することでユーザー体験を良くすることができる

- Extension の利用によりハンドラーとは別言語 Rust を利用が可能になり高速化することが可能

- 言語ランタイムを Rust で実装する LLRT の高速化アプローチの紹介

Thank you!