

JAPAN | 2024

aws SUMMIT



# Amazon Aurora の技術と イノベーション Deep dive

**塚本 真理**

アマゾンウェブサービスジャパン合同会社  
デジタルソリューションサービス技術本部 ISV/SaaSソリューション部  
ソリューションアーキテクト

# アジェンダ

- Amazon Aurora の基本
  - アーキテクチャ
  - グローバルデータベース
- 管理性の向上に役立つ機能
  - Aurora Serverless
  - Managed Blue/Green Deployment
  - Amazon Redshift とのAmazon Aurora ゼロETL統合
- 新しいストレージタイプとOptimized Read
- 生成AIに活用できる機能

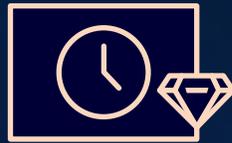
# Amazon Aurora

MySQL と PostgreSQL の完全な互換性を備え、世界規模で比類のない高いパフォーマンスと可用性を商用データベースの 10 分の 1 のコストで実現するように設計されています。



## パフォーマンスと 拡張性

- 一般的なMySQL の 5 倍、PostgreSQL の 3 倍のスループット
- 最大 15 個のリードレプリカまでスケールアウト
- ストレージとコンピューティングを分離することでコストの最適化を実現
- 高速なデータベースクローン作成
- 動的にスケールリングする分散型ストレージサブシステム



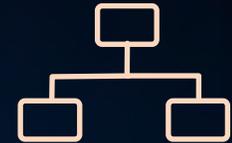
## 可用性と 耐久性

- 3 つの AZ にわたる 6 つのデータのコピー (1 つ分の料金支払いのみ)
- ポイントインタイムリカバリ (PITR) を備えた自動、継続的、増分バックアップ
- フォールトトレラント、自己修復、自動スケールのストレージ
- 災害復旧のためのグローバルデータベース



## 高い安全性

- ネットワークの分離
- 保存時/転送時の暗号化
- 複数の安全な認証メカニズムと監査制御をサポート



## フルマネージド

- ハードウェアのプロビジョニング、データベースのセットアップ、パッチ適用、バックアップなどの時間のかかる管理タスクの管理を自動化します。
- サーバーレス構成オプション

# アーキテクチャ



# Aurora ストレージとレプリカ - 可用性と耐久性

Availability Zone 1

Availability Zone 2

Availability Zone 3

Aurora  
ストレージ



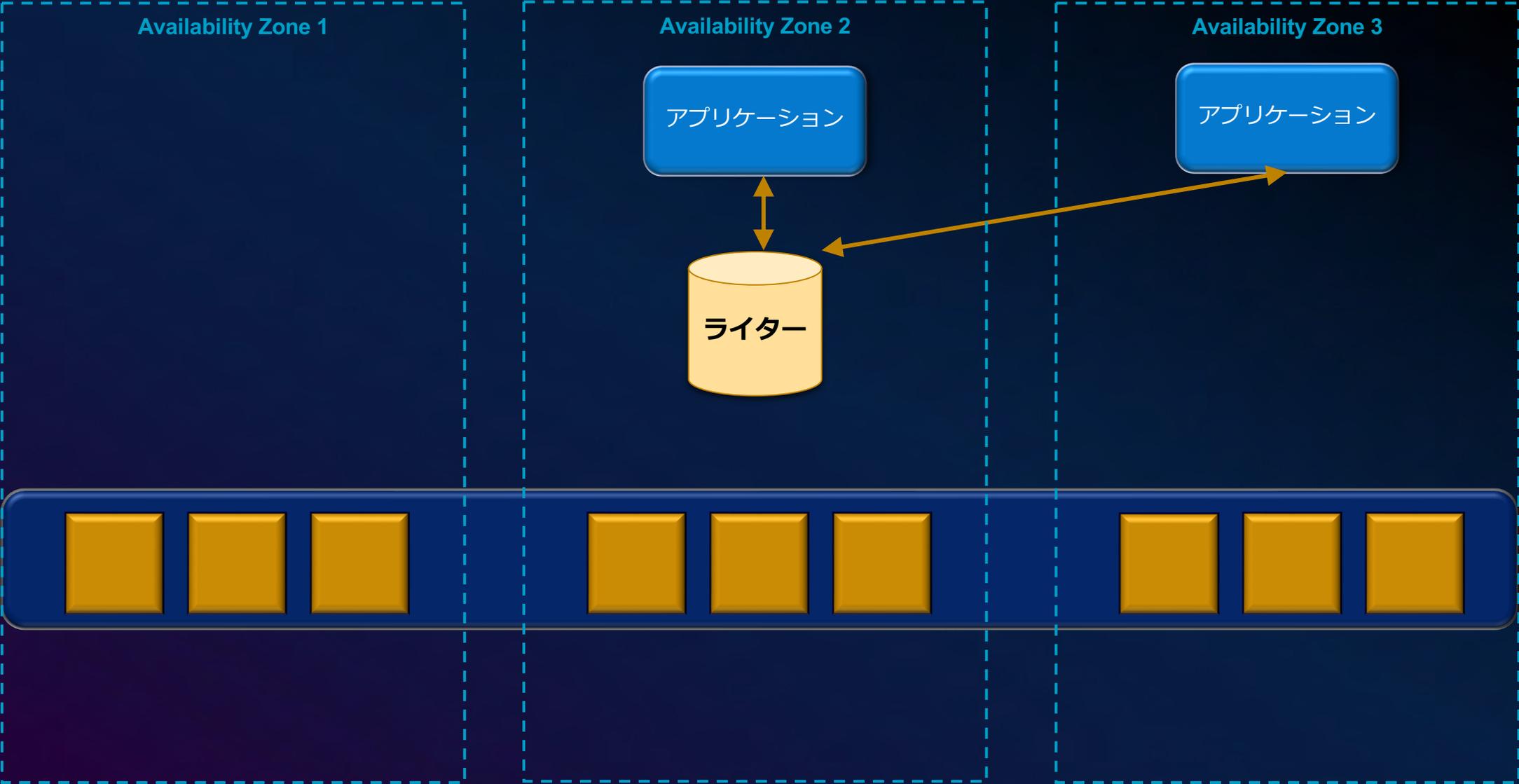
# Aurora ストレージとレプリカ - 可用性と耐久性



Aurora  
ストレージ



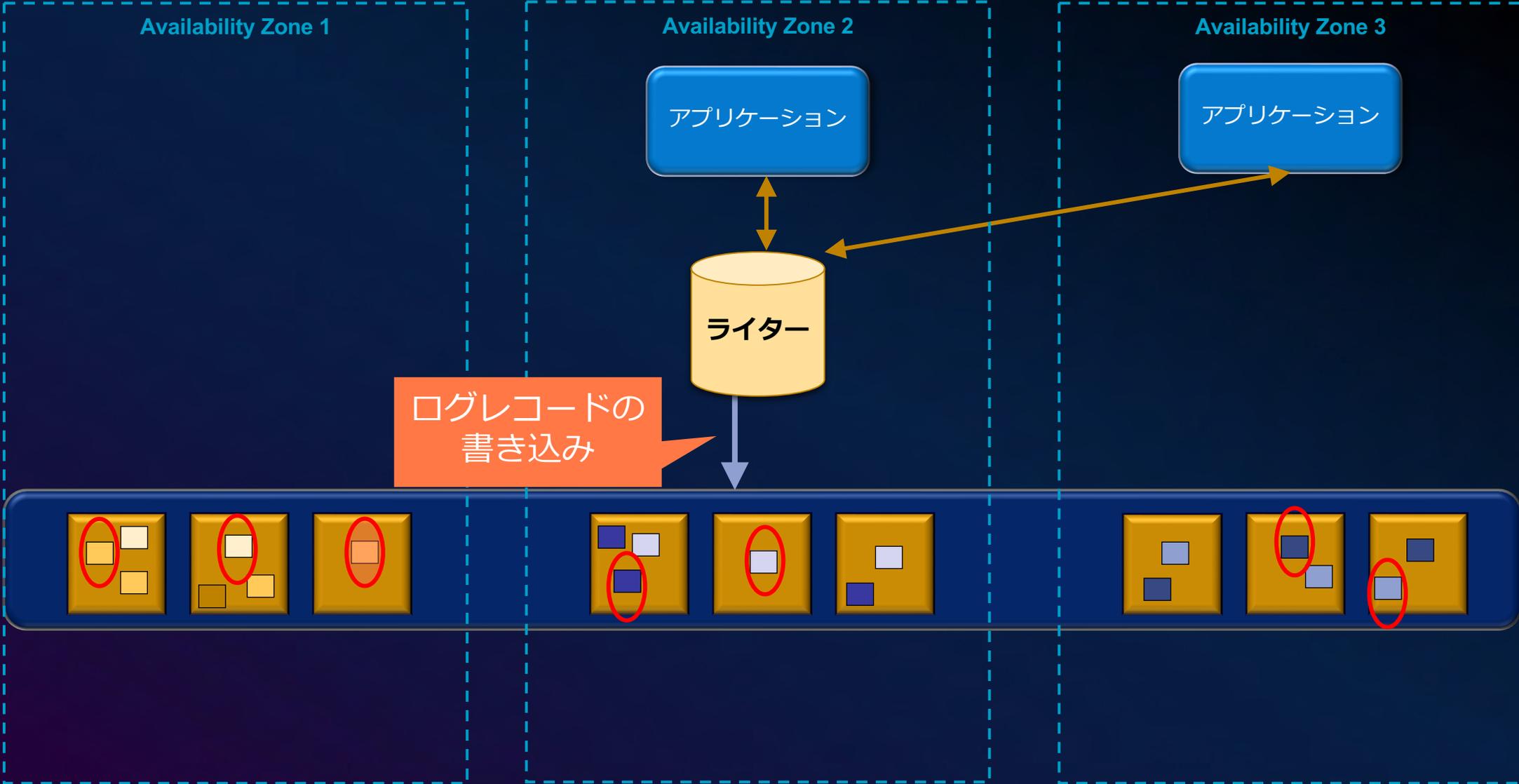
# Aurora ストレージとレプリカ - 可用性と耐久性



Aurora  
ストレージ



# Aurora ストレージとレプリカ - 可用性と耐久性

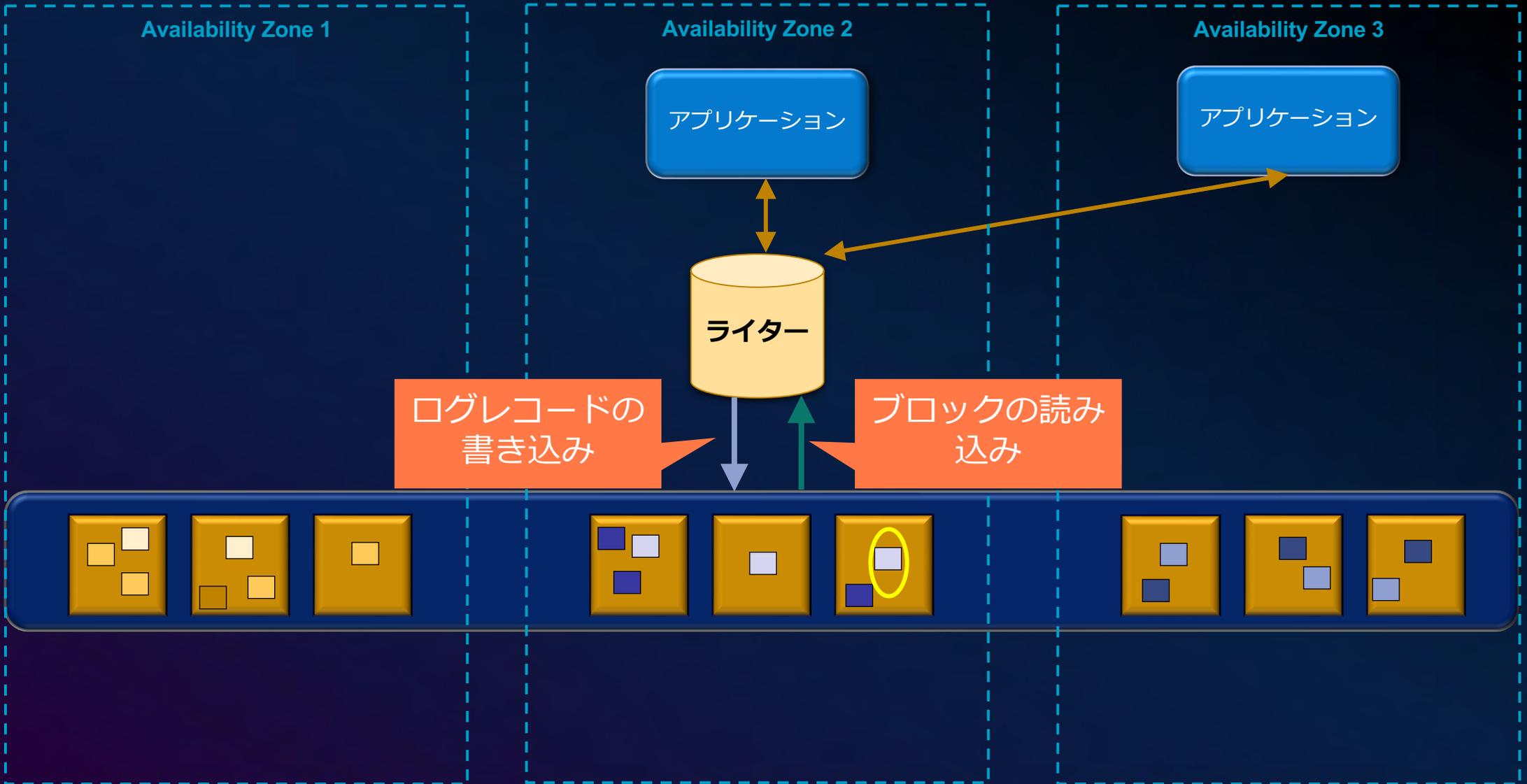


Aurora  
ストレージ

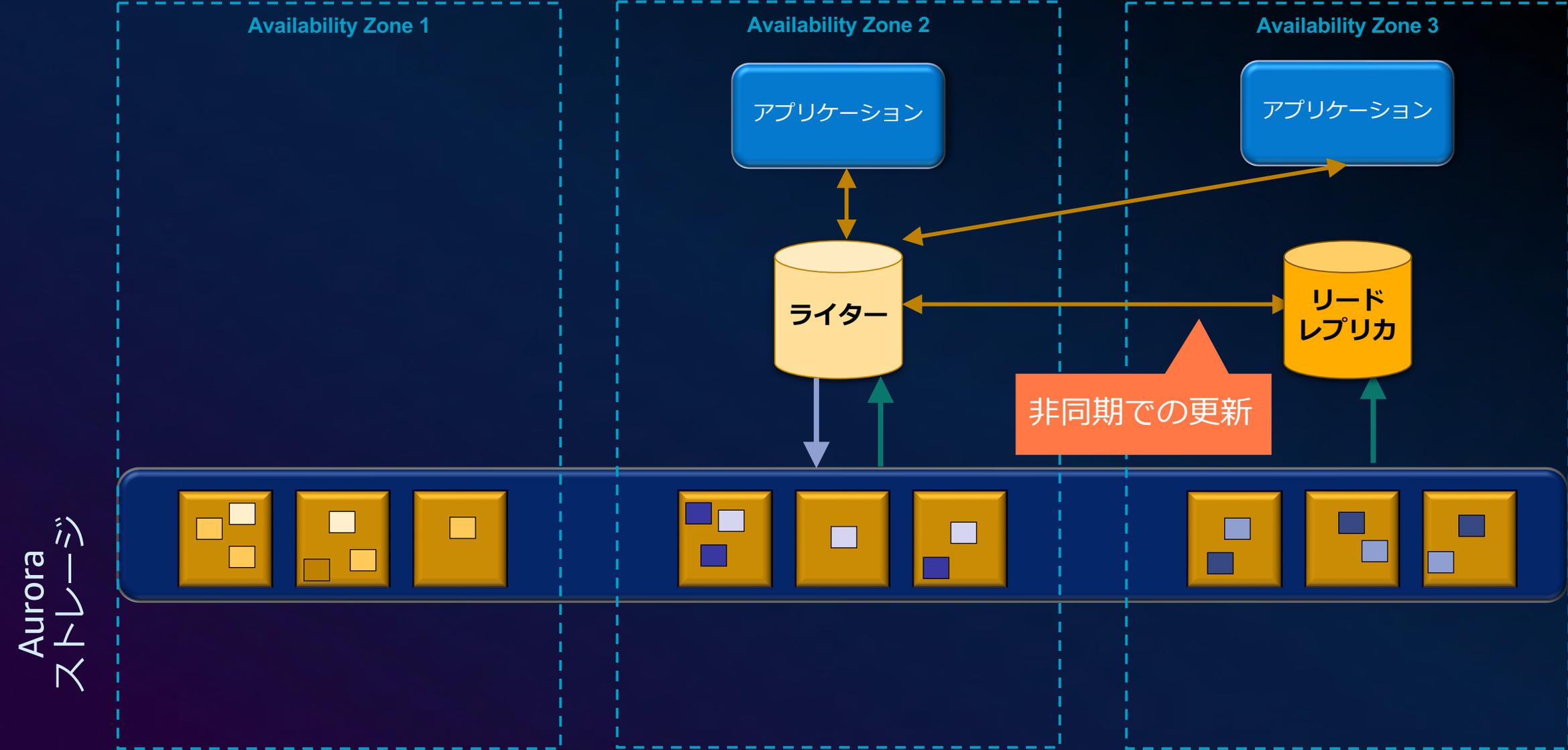


# Aurora ストレージとレプリカ - 可用性と耐久性

Aurora  
ストレージ



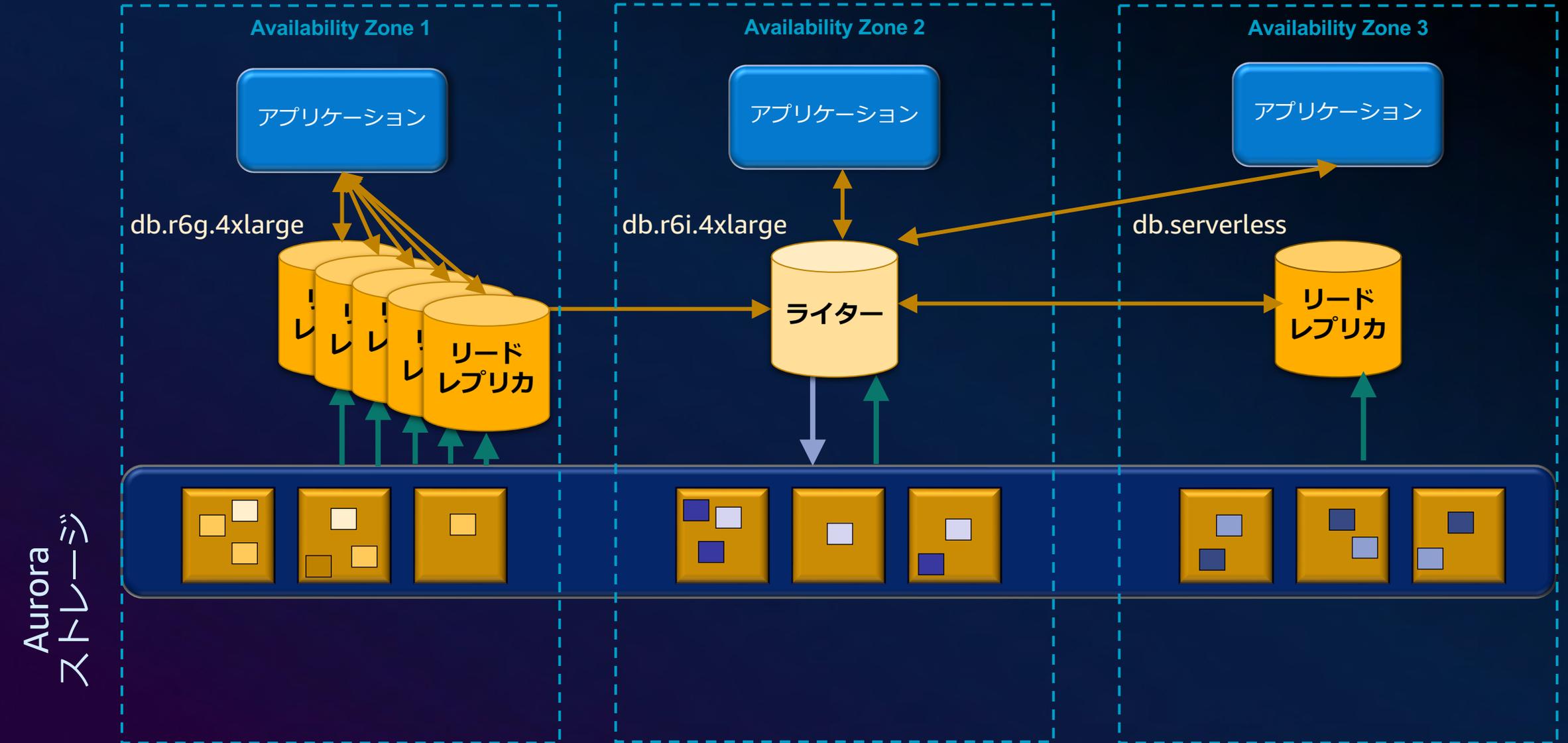
# Aurora ストレージとレプリカ - 拡張性とパフォーマンス



Aurora  
ストレージ



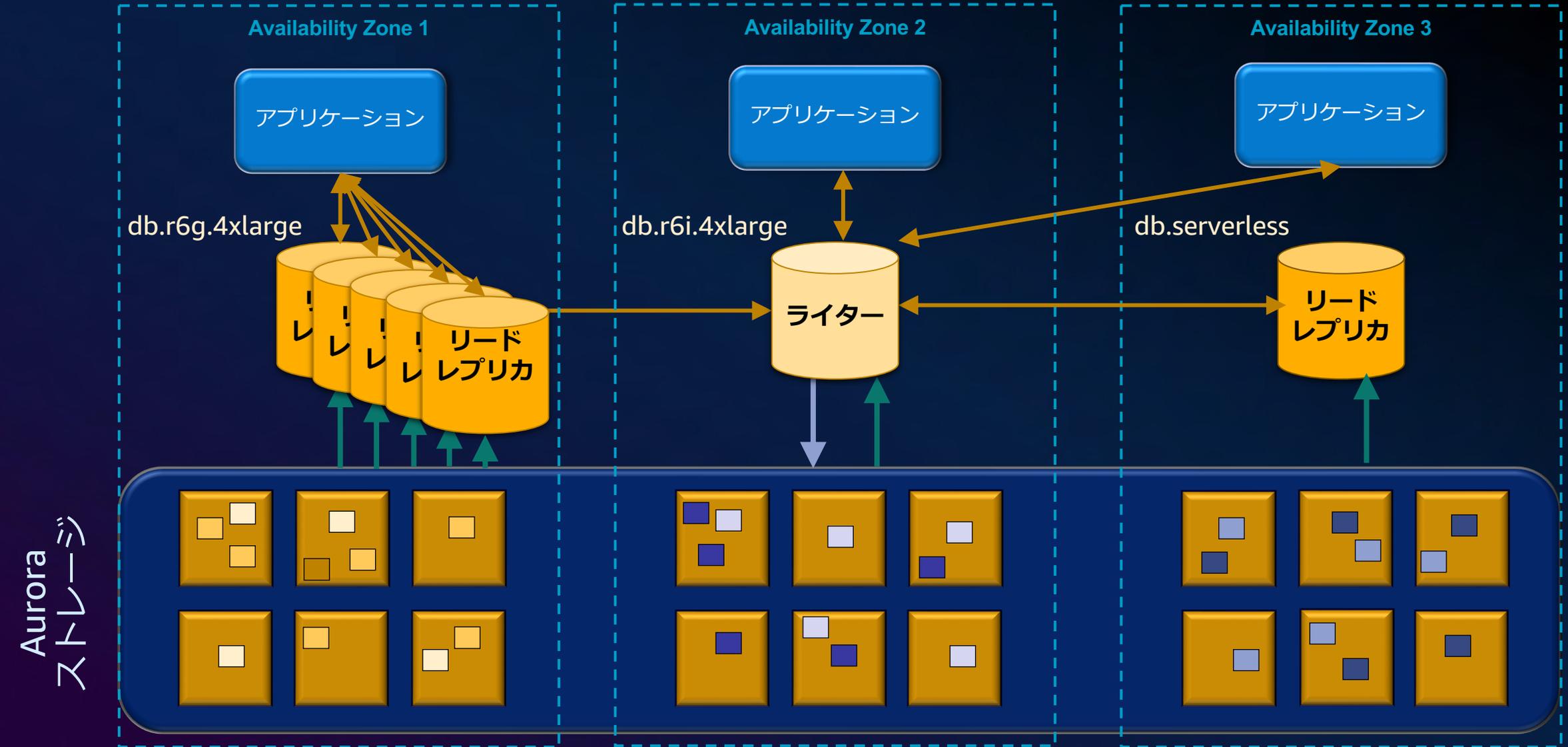
# Aurora ストレージとレプリカ - 拡張性とパフォーマンス



Aurora  
ストレージ



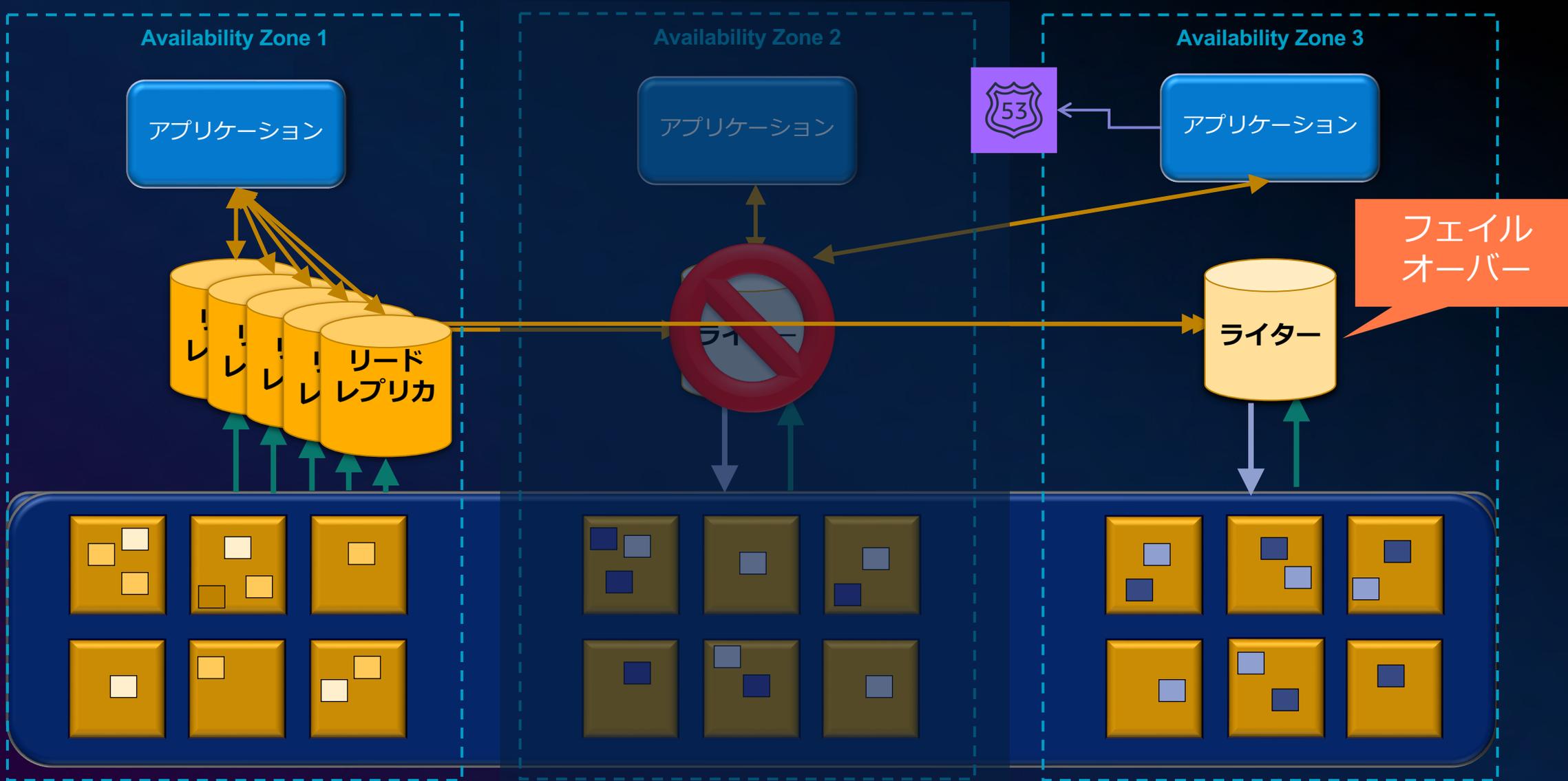
# Aurora ストレージとレプリカ- 拡張性とパフォーマンス



Aurora  
ストレージ



# Aurora ストレージとレプリカ - フェイルオーバー



Aurora  
ストレージ



# Aurora ストレージとレプリカ - フェイルオーバー



Aurora  
ストレージ



# Aurora ストレージとレプリカ - フェイルオーバー

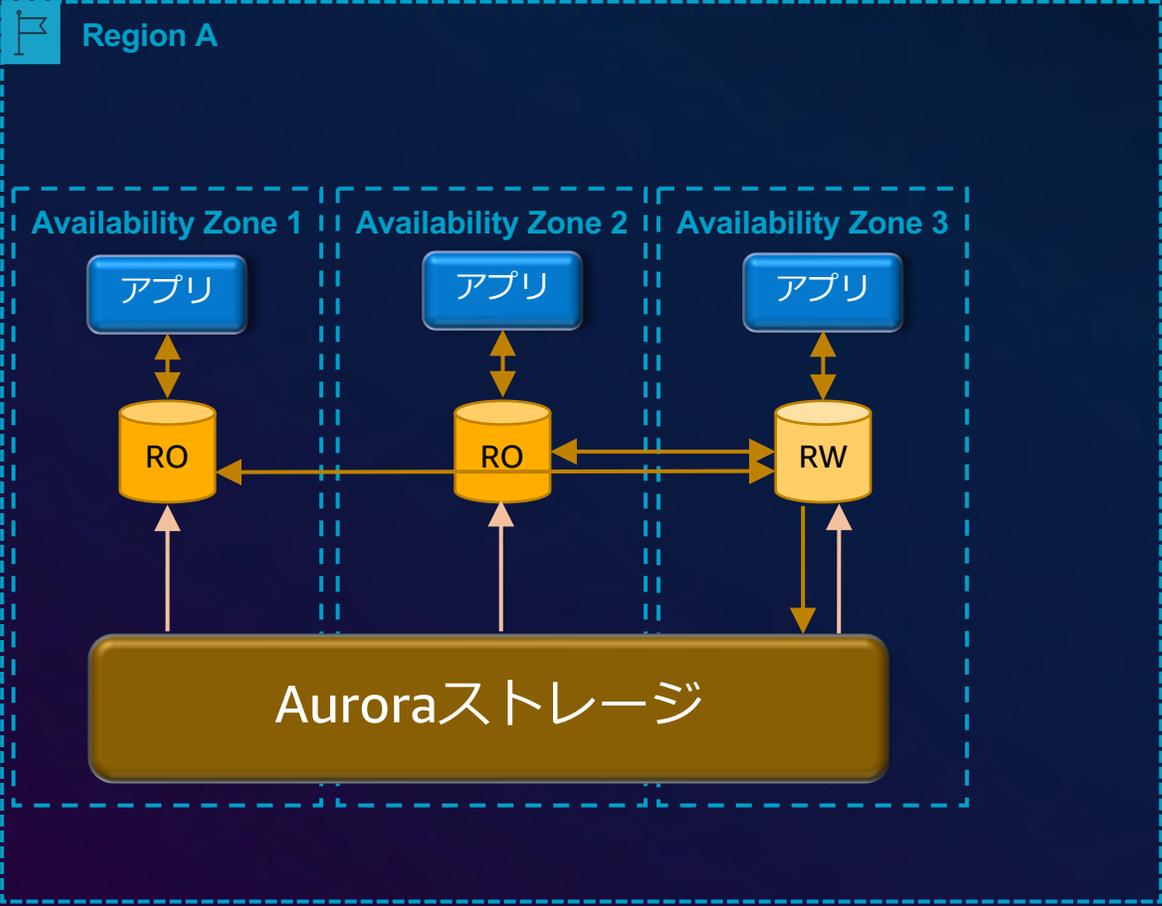


Aurora  
ストレージ

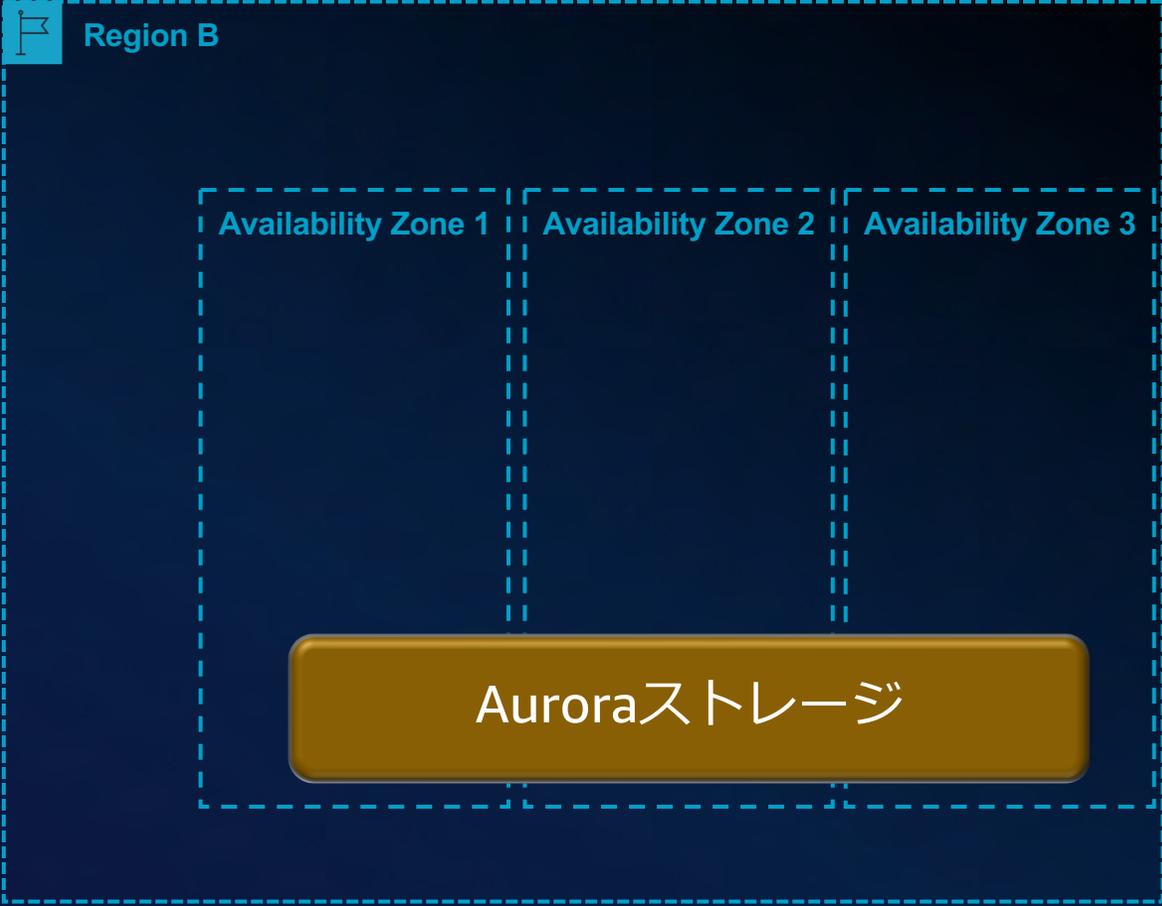


# Amazon Aurora Global Database

RW : ライター  
RO : リードレプリカ



プライマリ DB クラスター

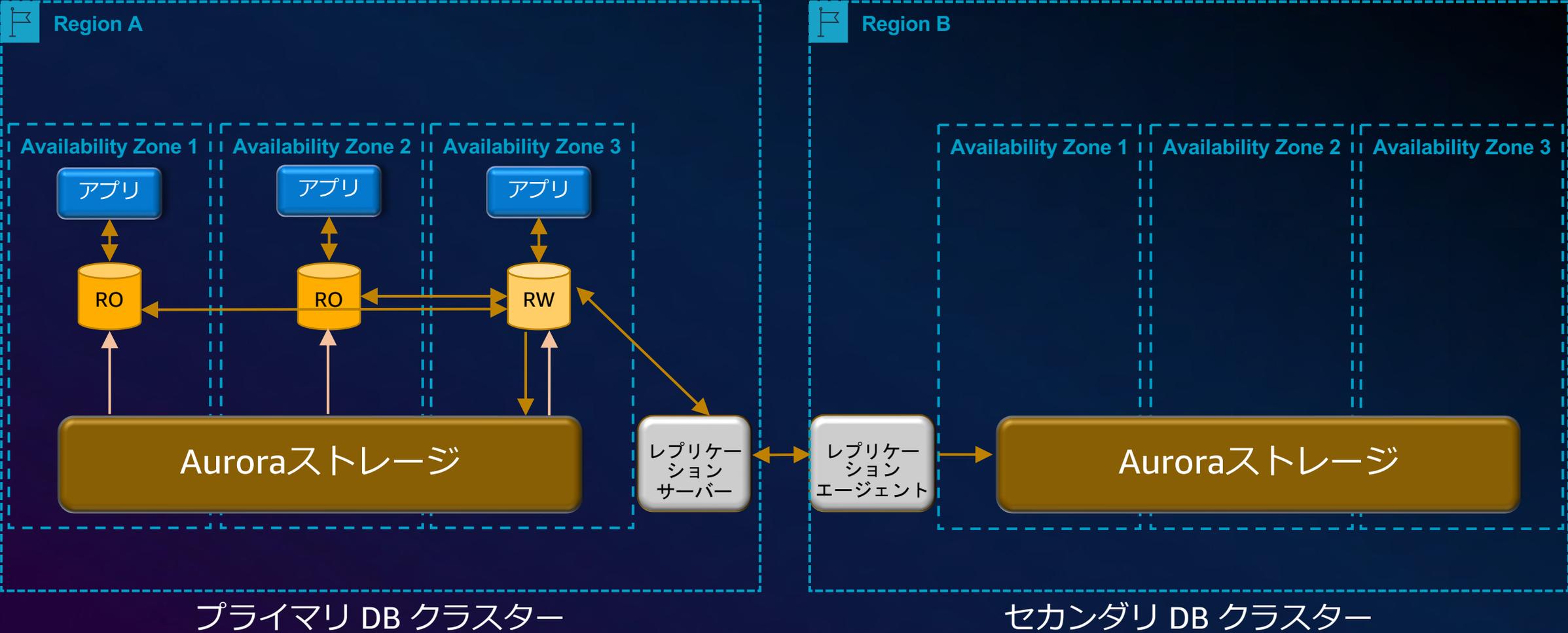


セカンダリ DB クラスター



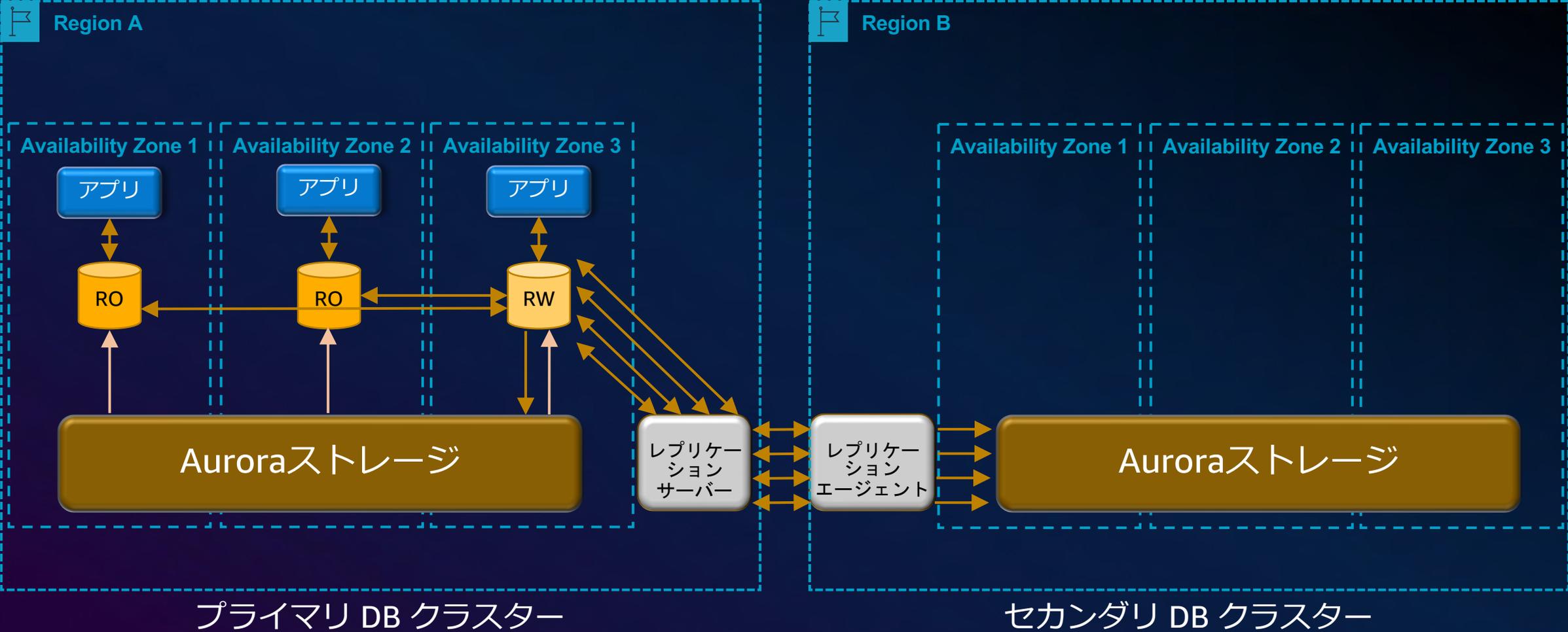
# Amazon Aurora Global Database

RW : ライター  
RO : リードレプリカ



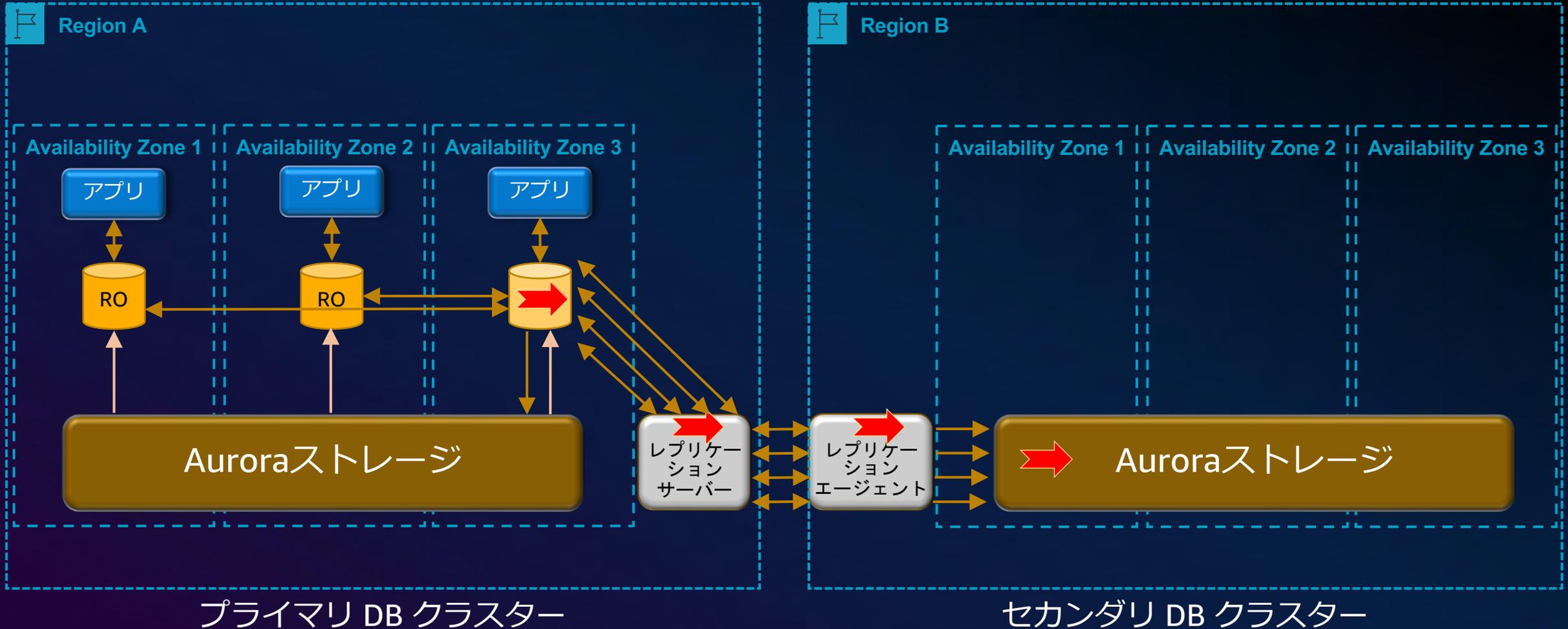
# Amazon Aurora Global Database

RW : ライター  
RO : リードレプリカ



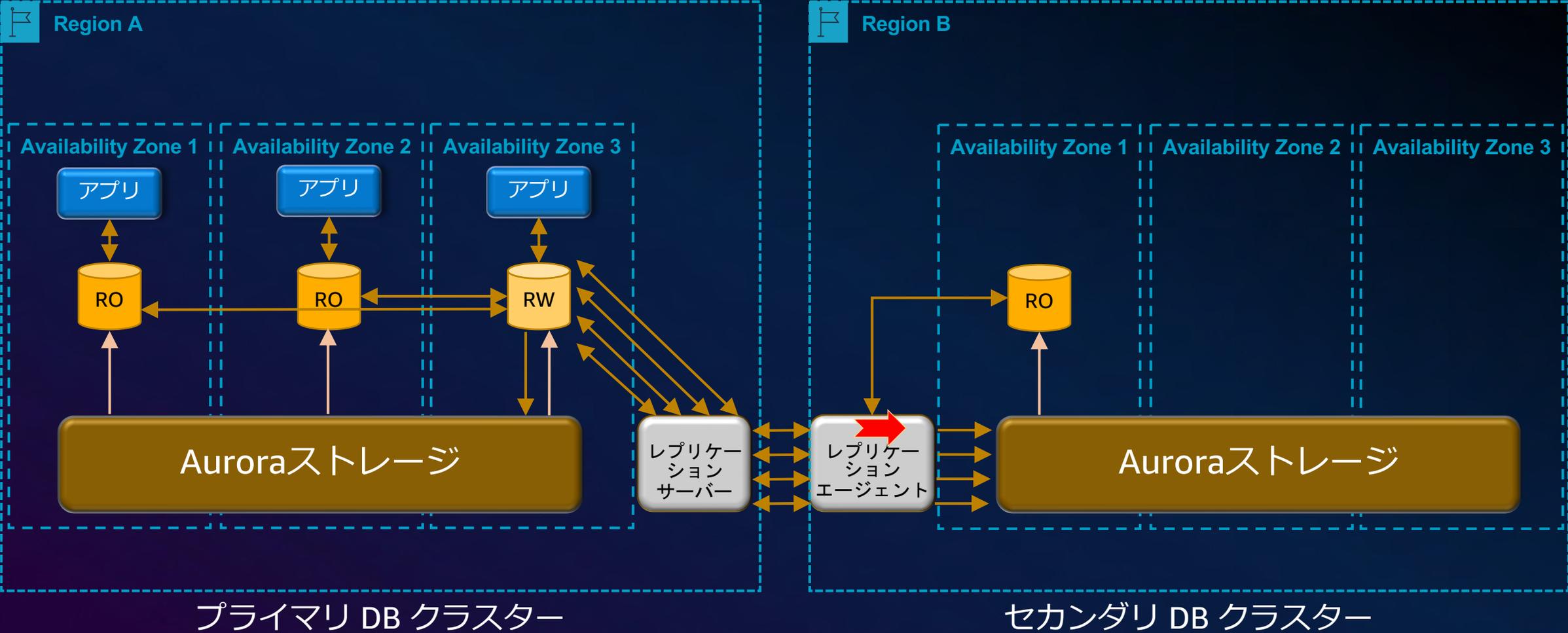
# Amazon Aurora Global Database

RW : ライター  
RO : リードレプリカ



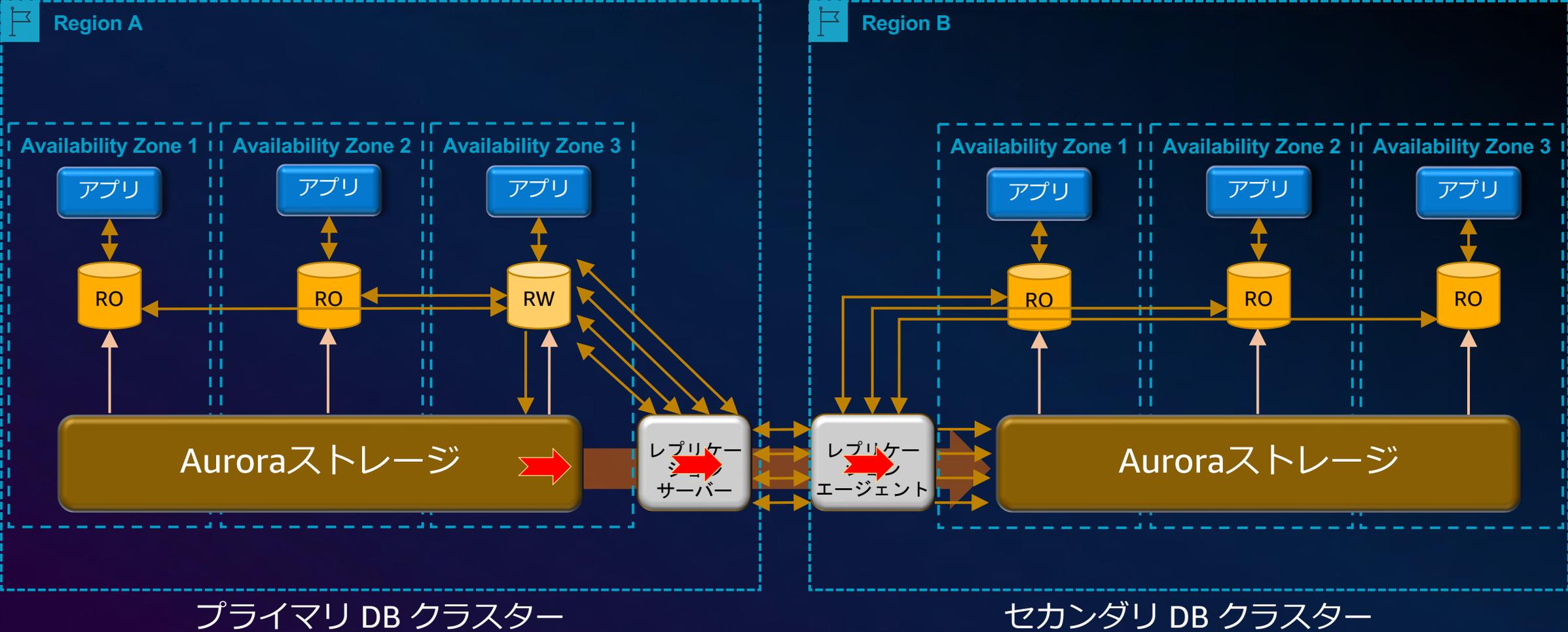
# Amazon Aurora Global Database

RW : ライター  
RO : リードレプリカ



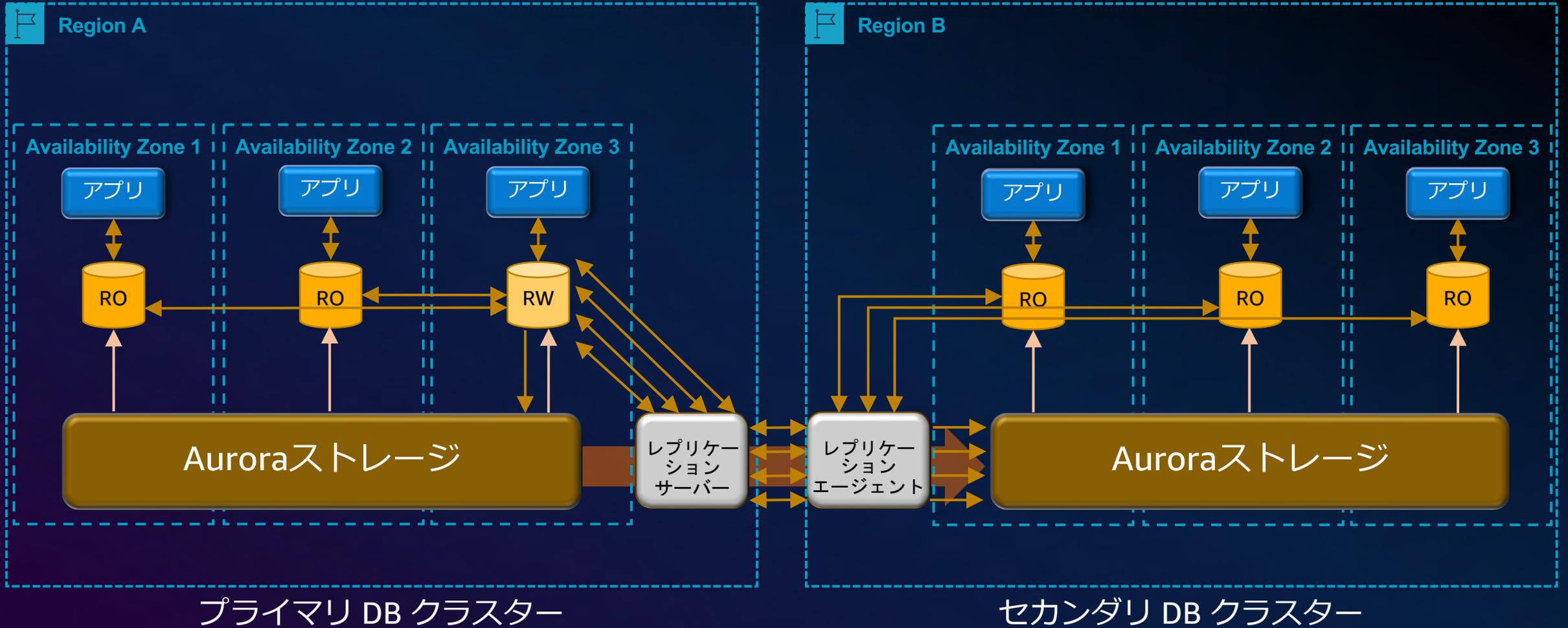
# Amazon Aurora Global Database

RW : ライター  
RO : リードレプリカ



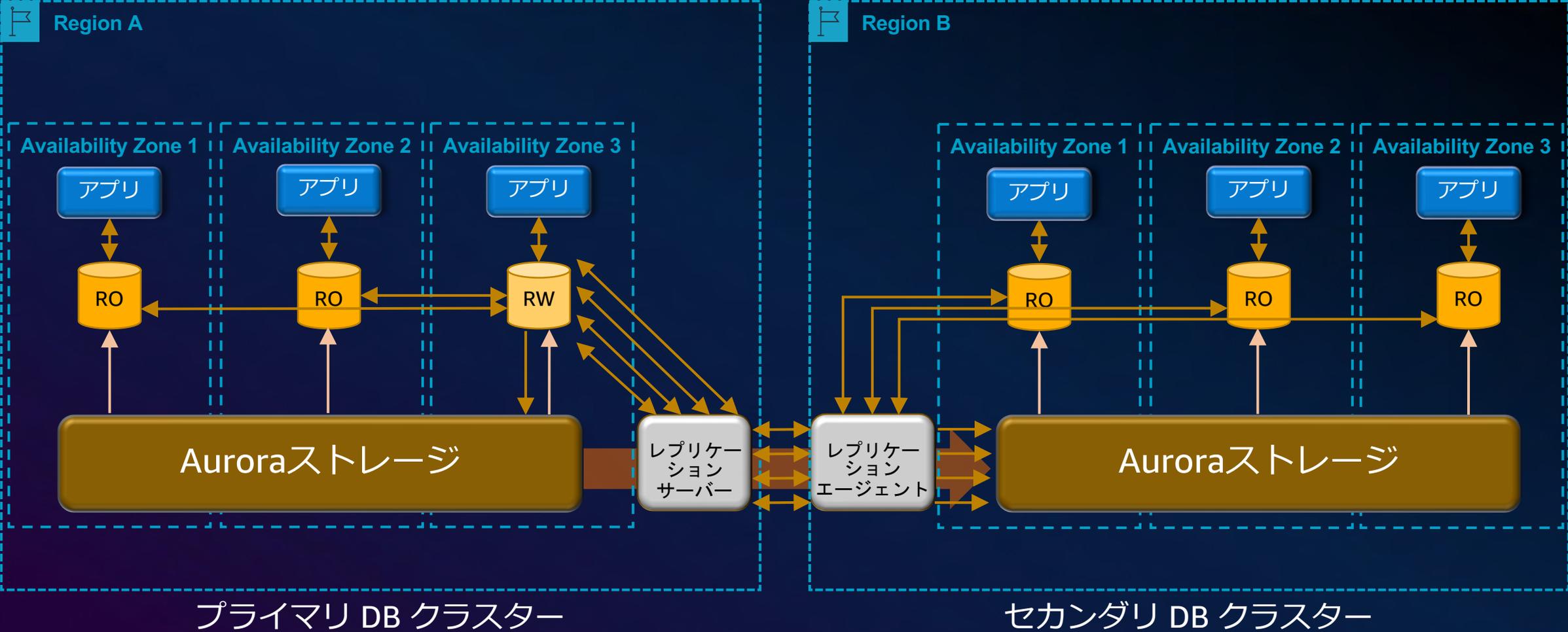
# Aurora Global Database - Switchover/ Failover

RW : ライター  
RO : リードレプリカ



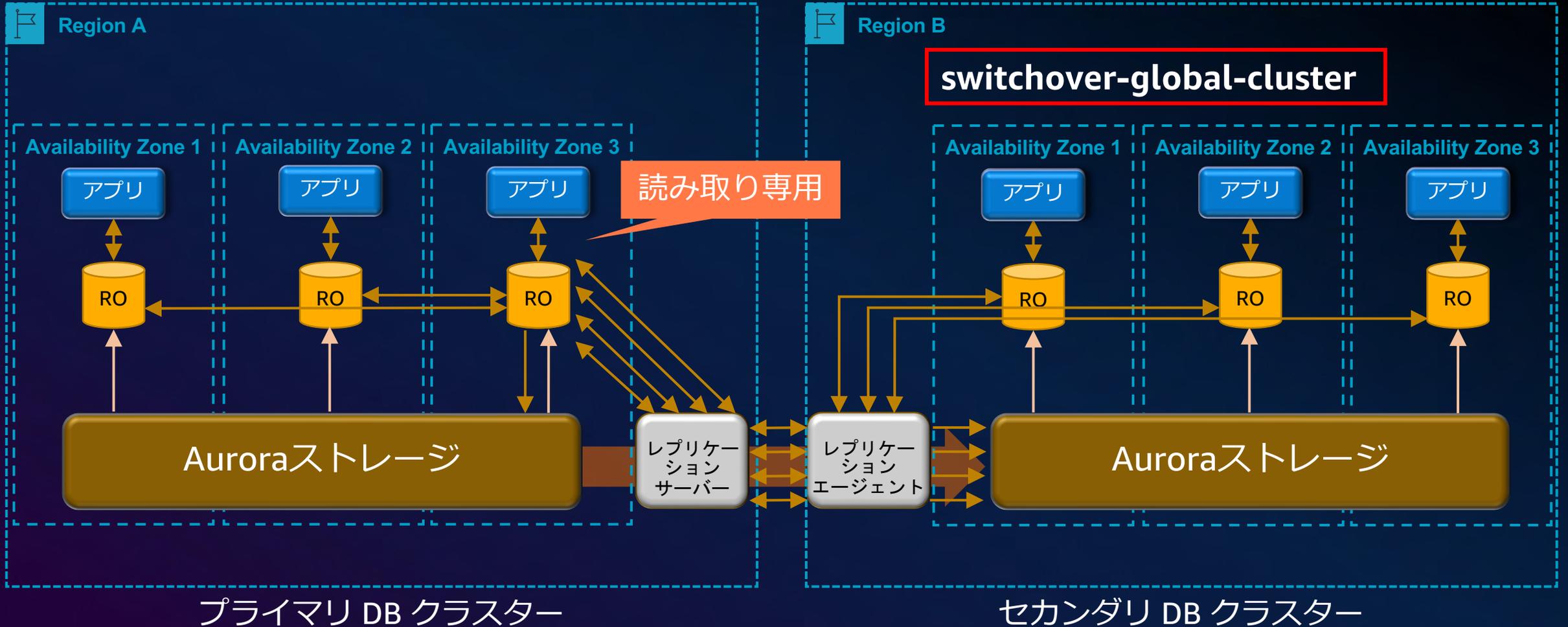
# Aurora Global Database - Switchover

RW : ライター  
RO : リードレプリカ



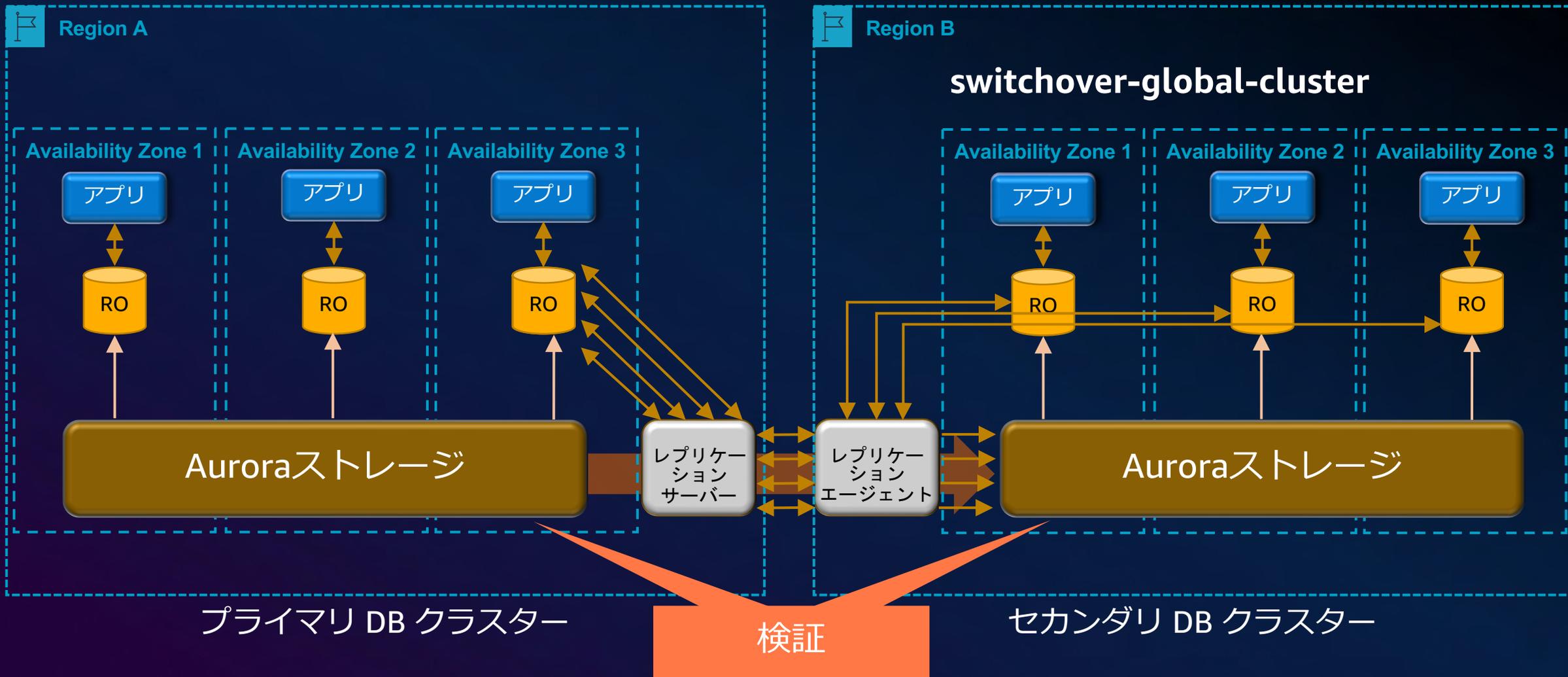
# Aurora Global Database - Switchover

RW : ライター  
RO : リードレプリカ



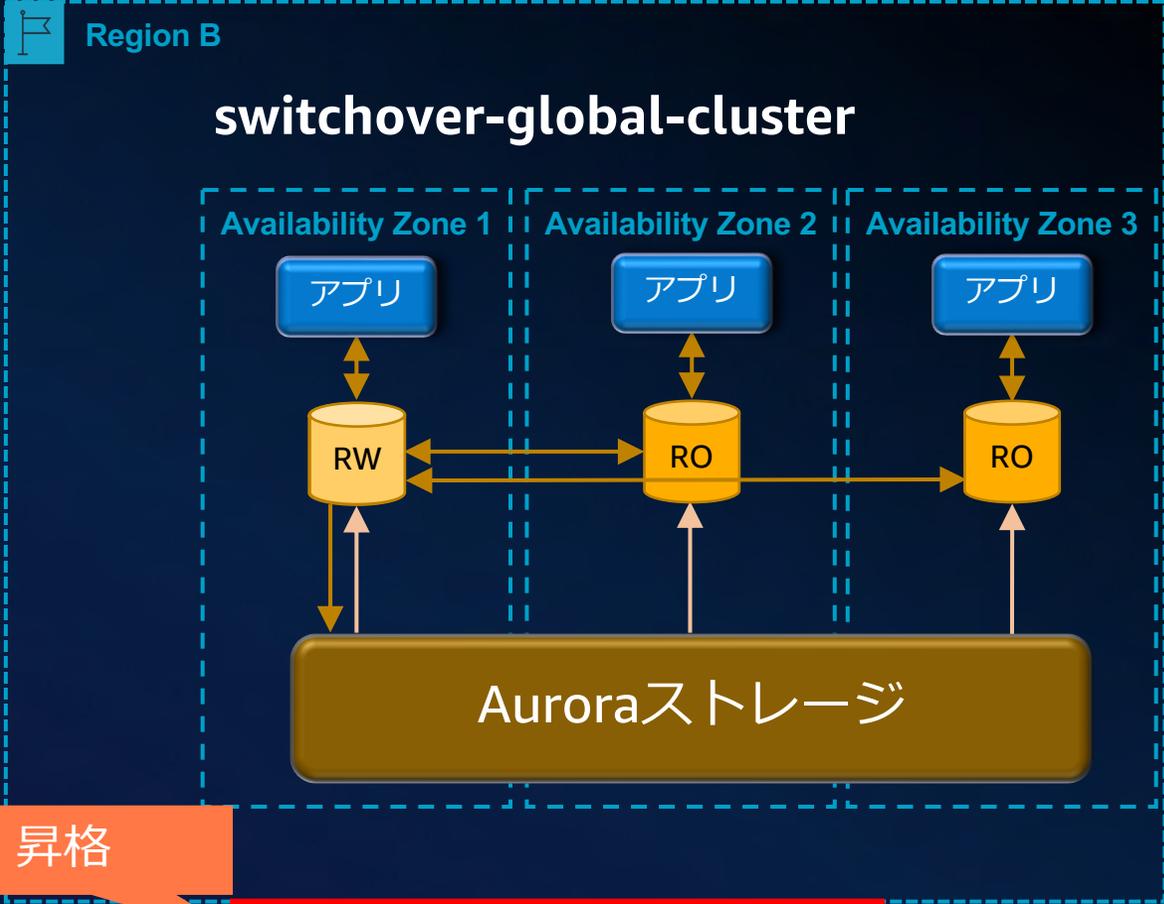
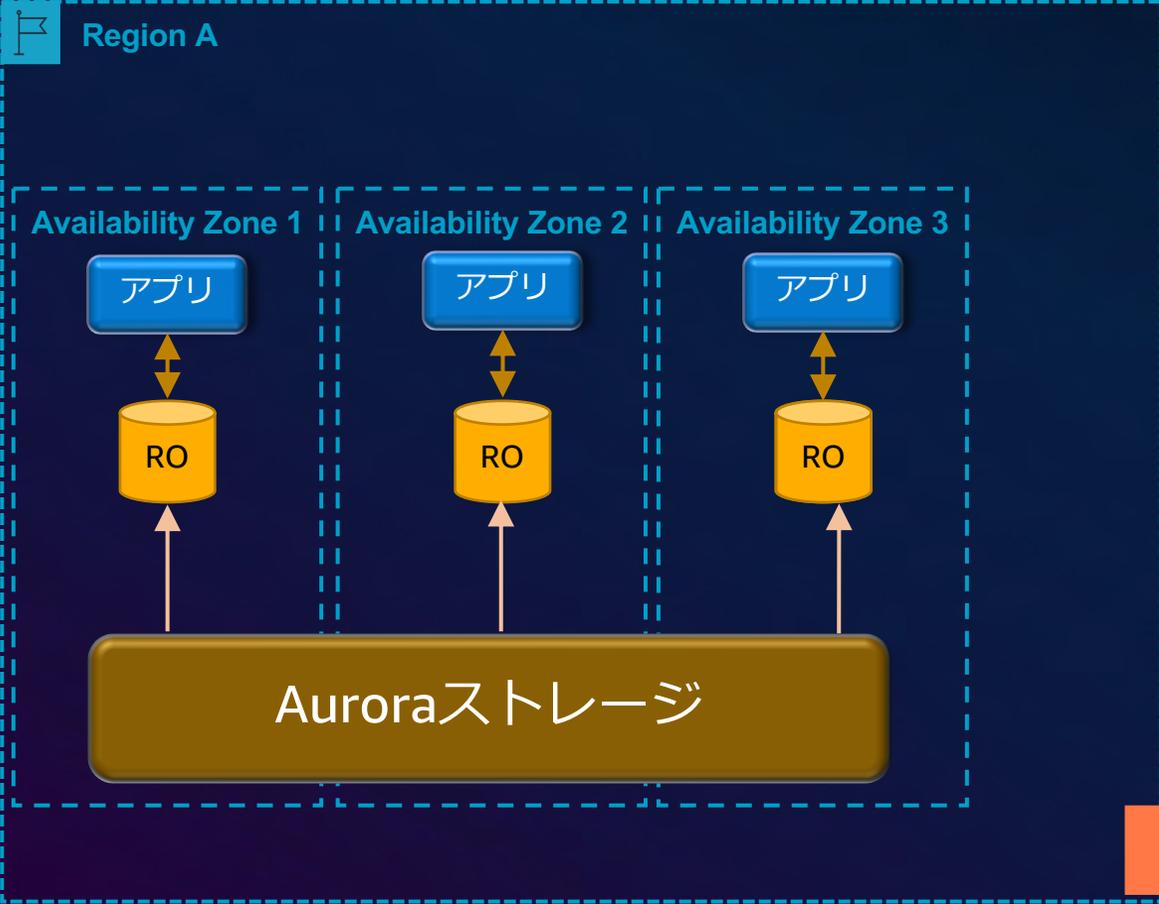
# Aurora Global Database - Switchover

RW : ライター  
RO : リードレプリカ



# Aurora Global Database - Switchover

RW : ライター  
RO : リードレプリカ



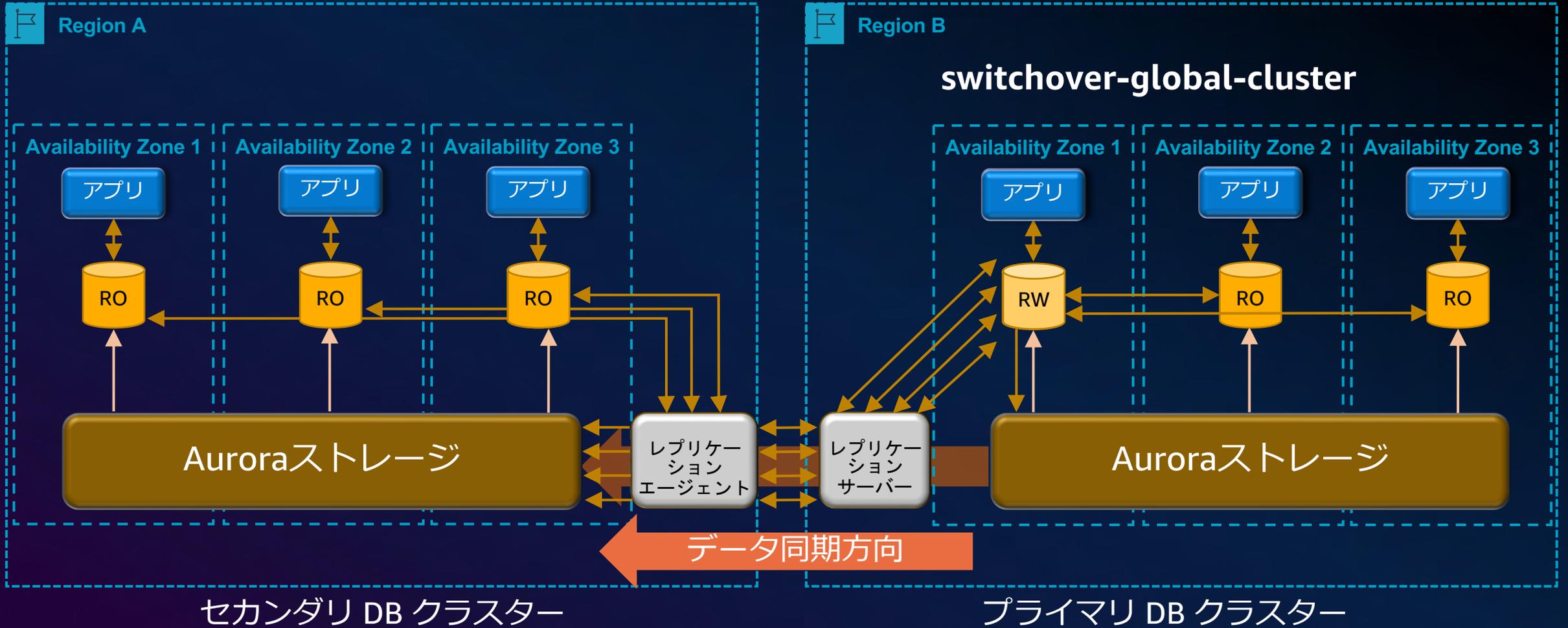
昇格

プライマリ DB クラスター



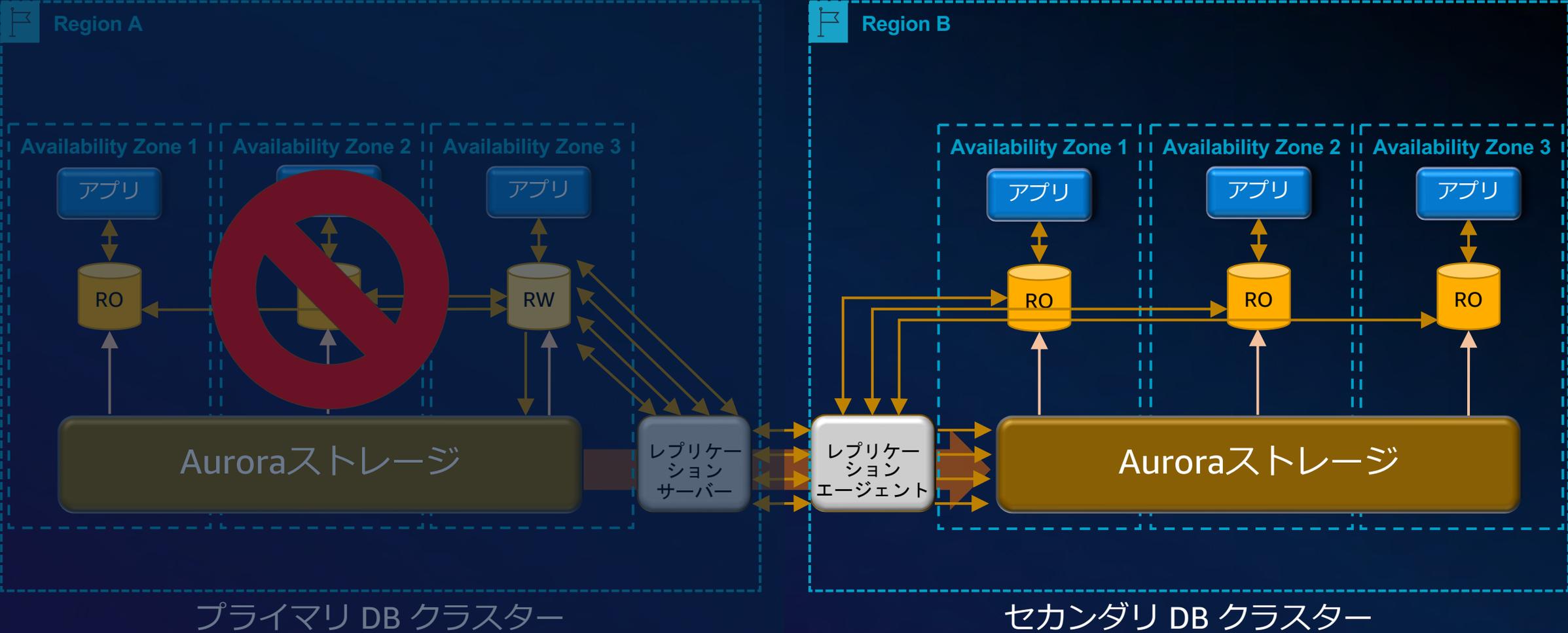
# Aurora Global Database - Switchover

RW : ライター  
RO : リードレプリカ



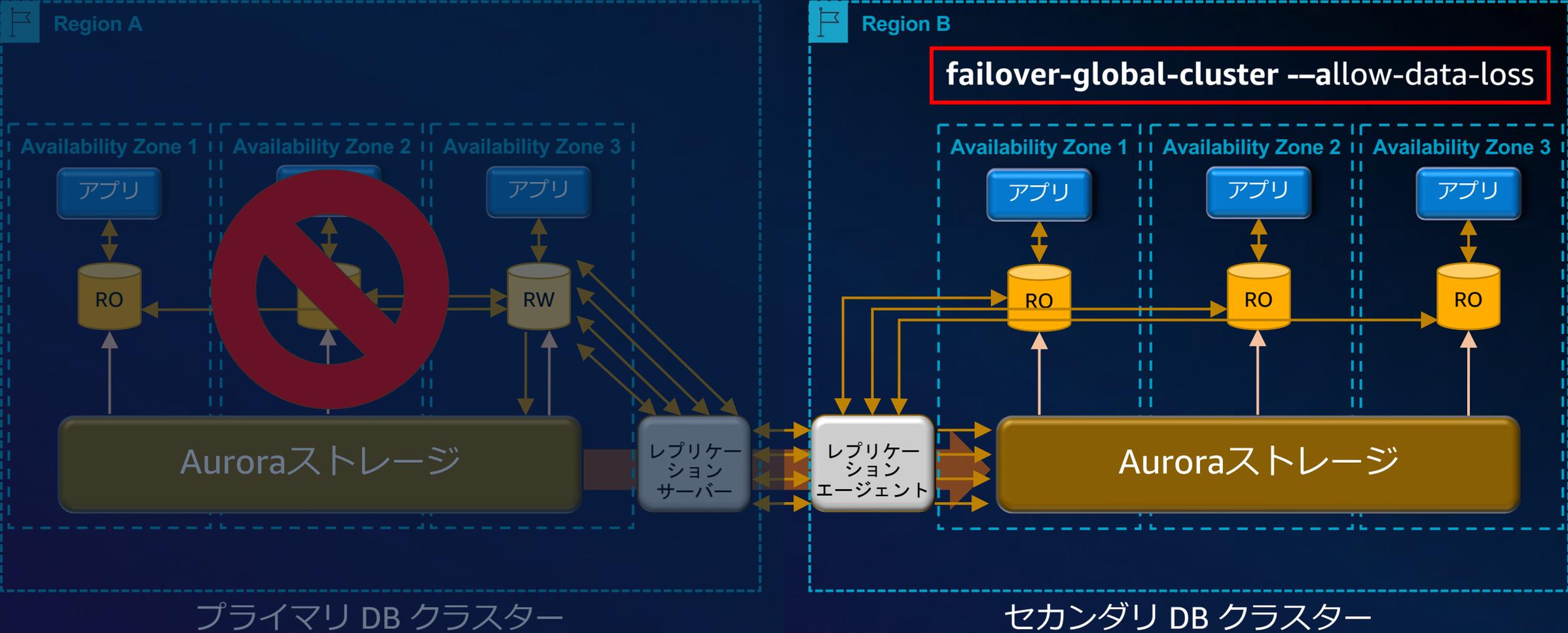
# Aurora Global Database - Failover

RW : ライター  
RO : リードレプリカ



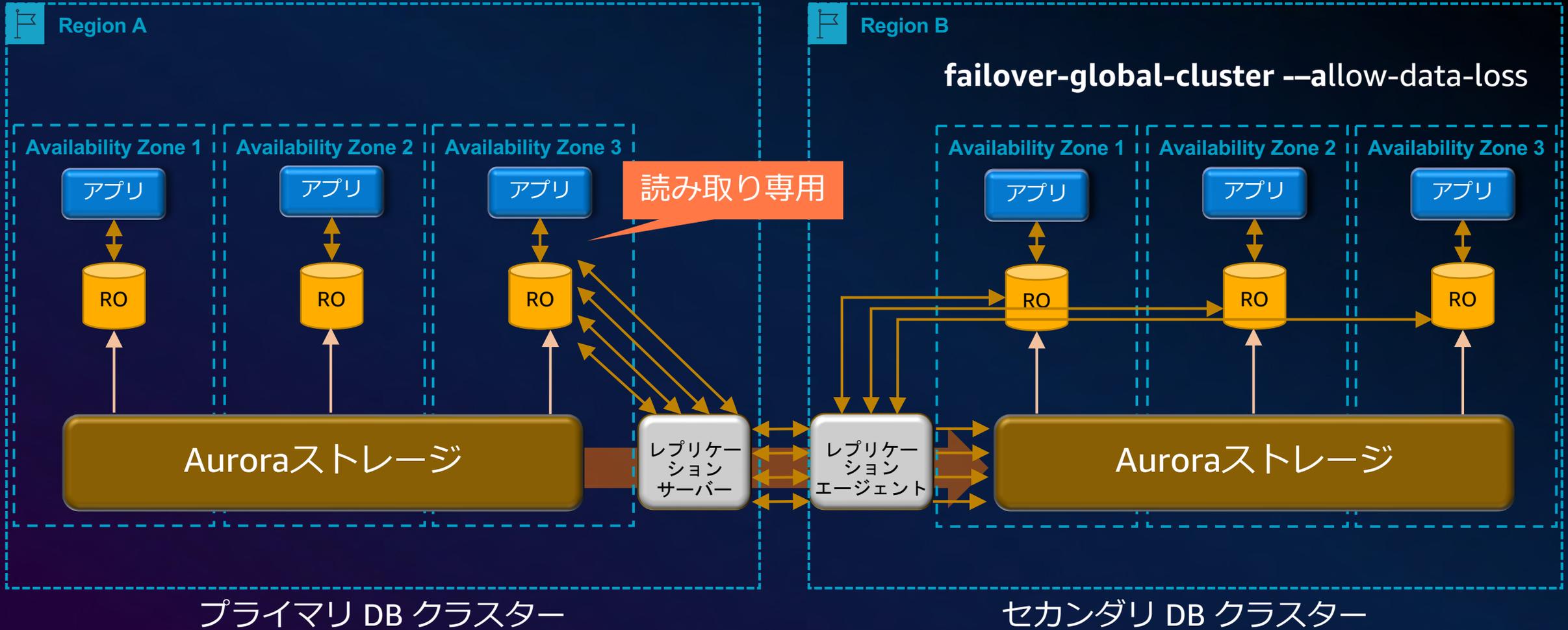
# Aurora Global Database - Failover

RW : ライター  
RO : リードレプリカ



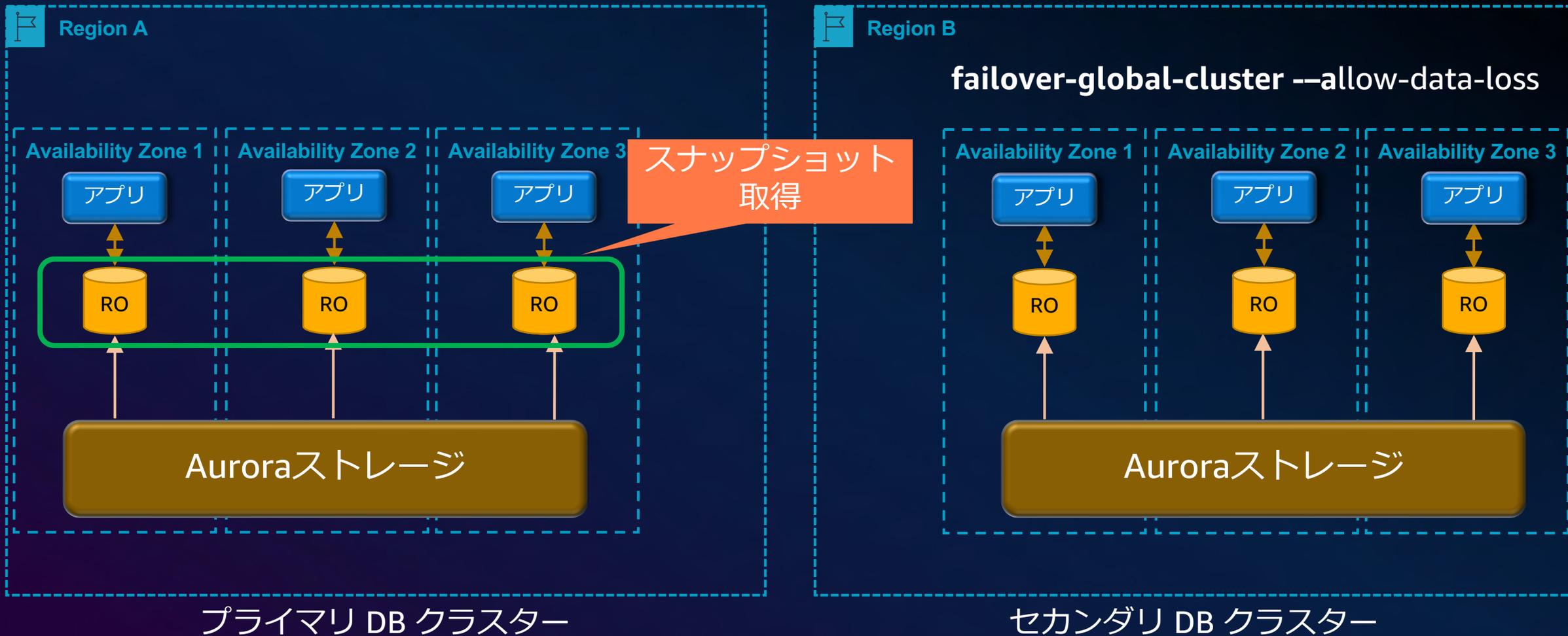
# Aurora Global Database - Failover

RW : ライター  
RO : リードレプリカ



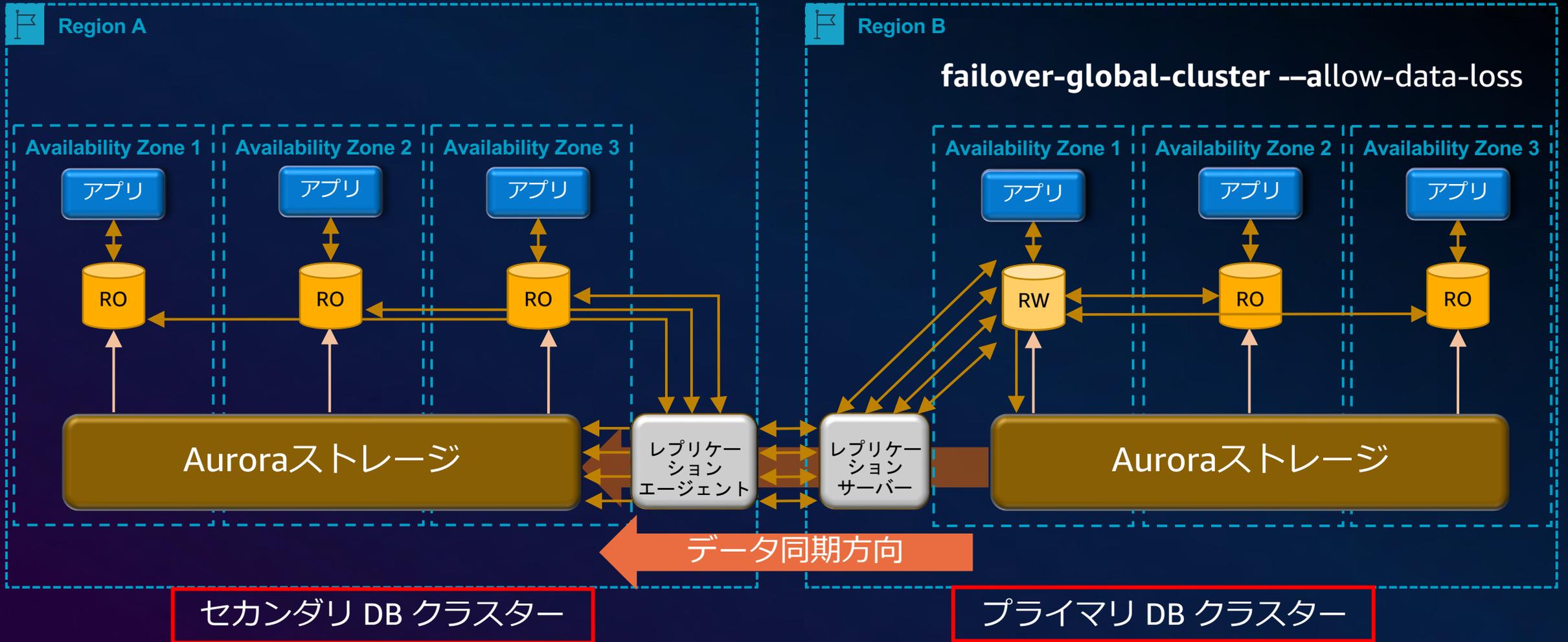
# Aurora Global Database - Failover

RW : ライター  
RO : リードレプリカ



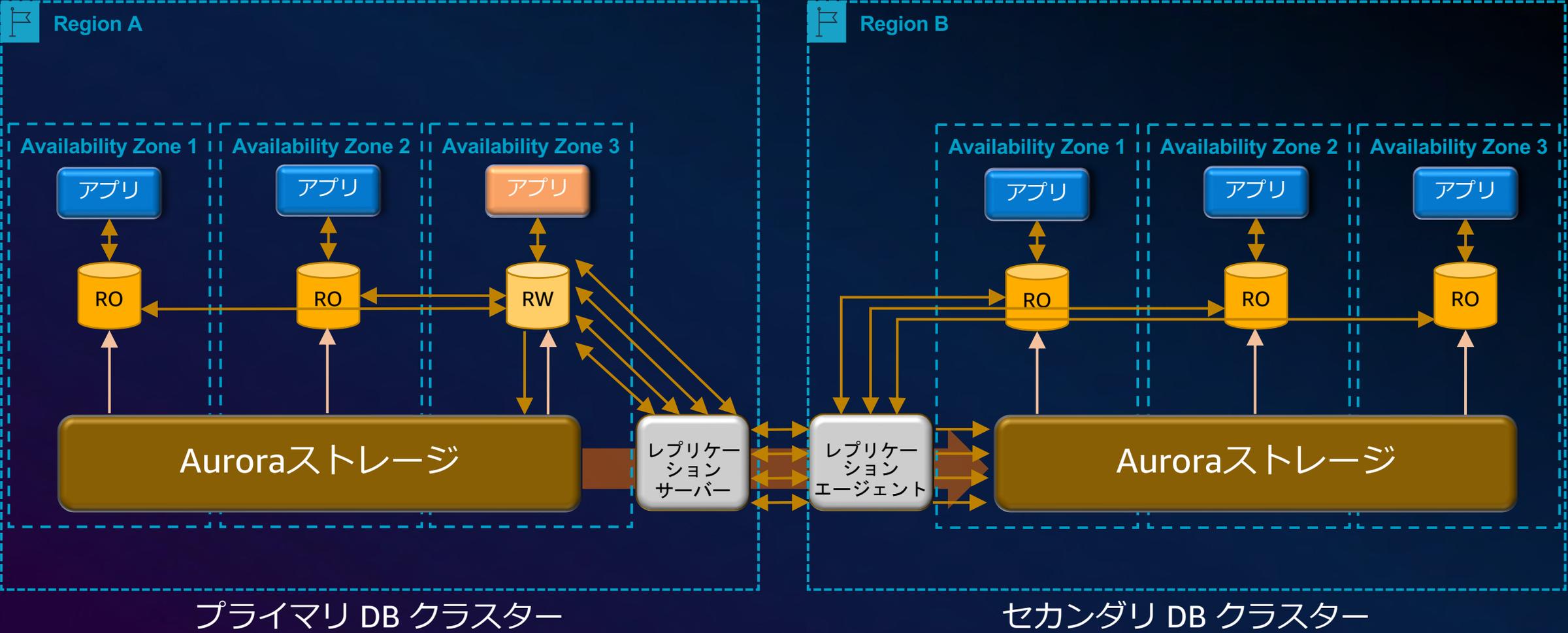
# Aurora Global Database - Failover

RW : ライター  
RO : リードレプリカ



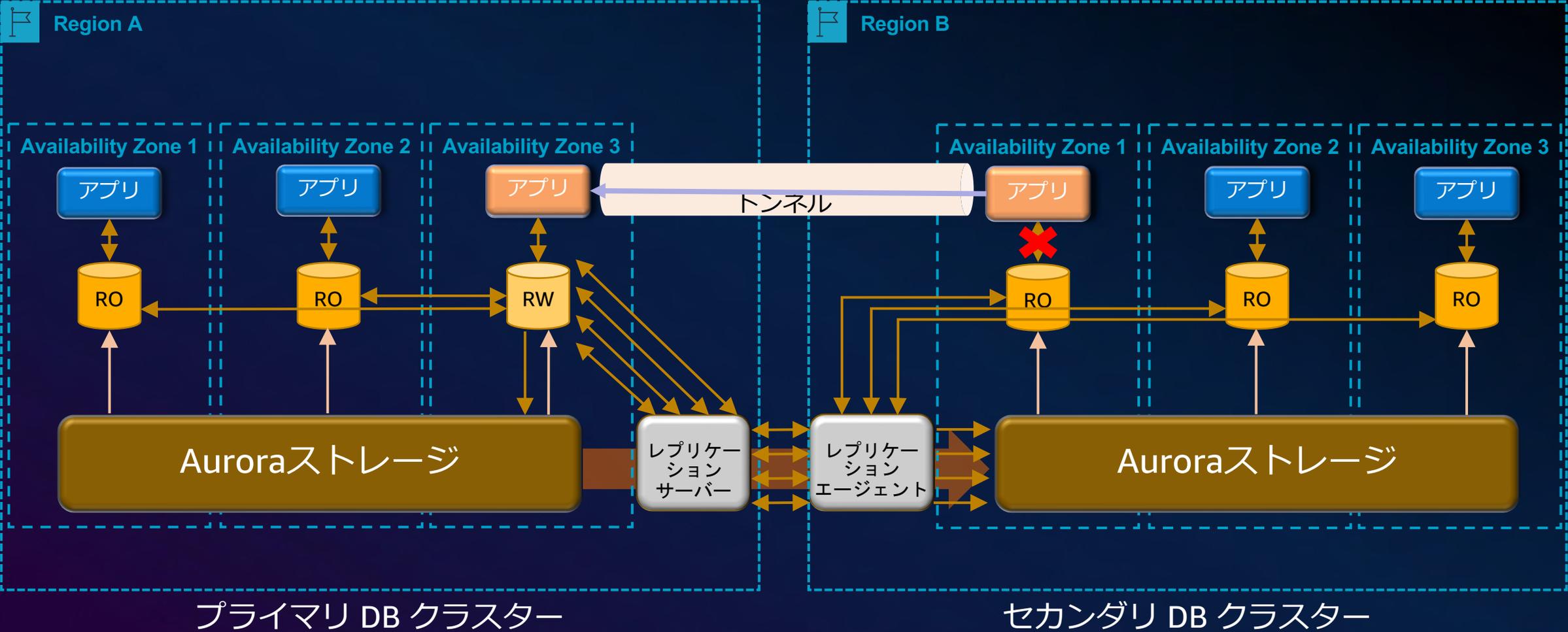
# Aurora Global Database - Write Forwarding

RW : ライター  
RO : リードレプリカ



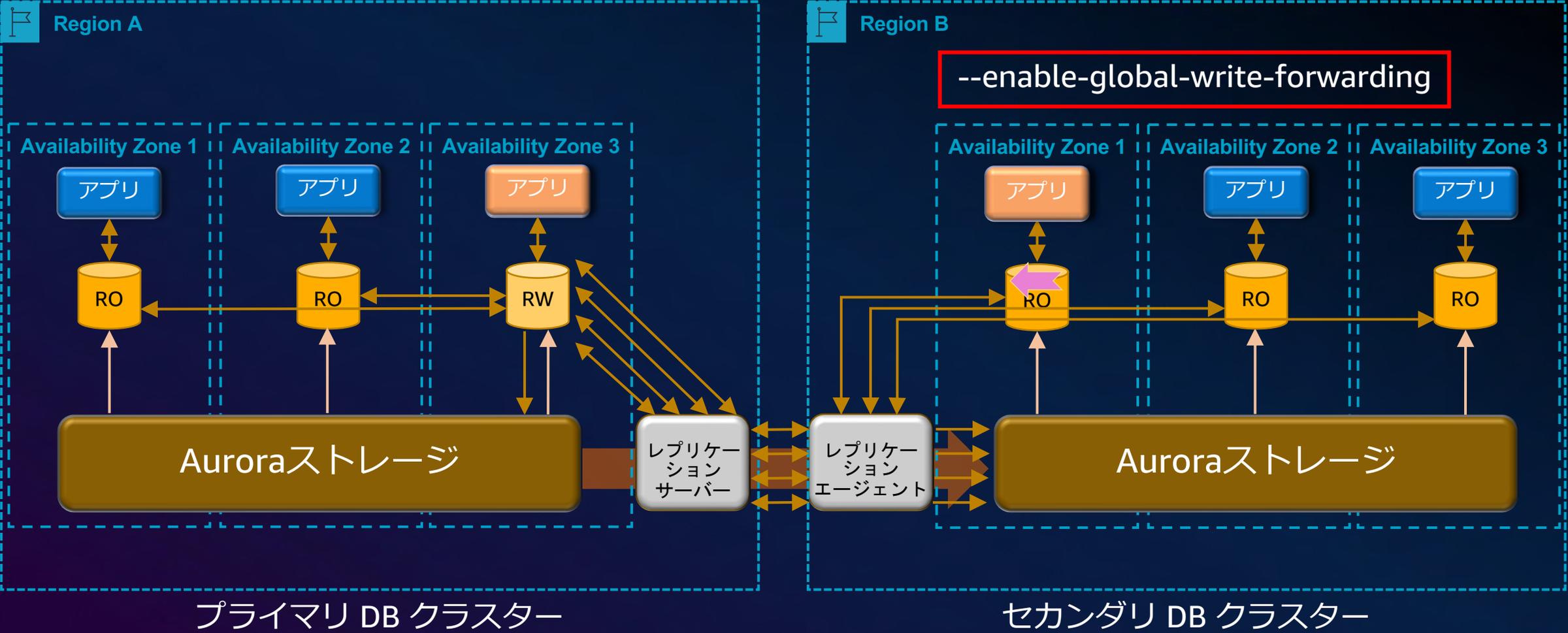
# Aurora Global Database - Write Forwarding

RW : ライター  
RO : リードレプリカ



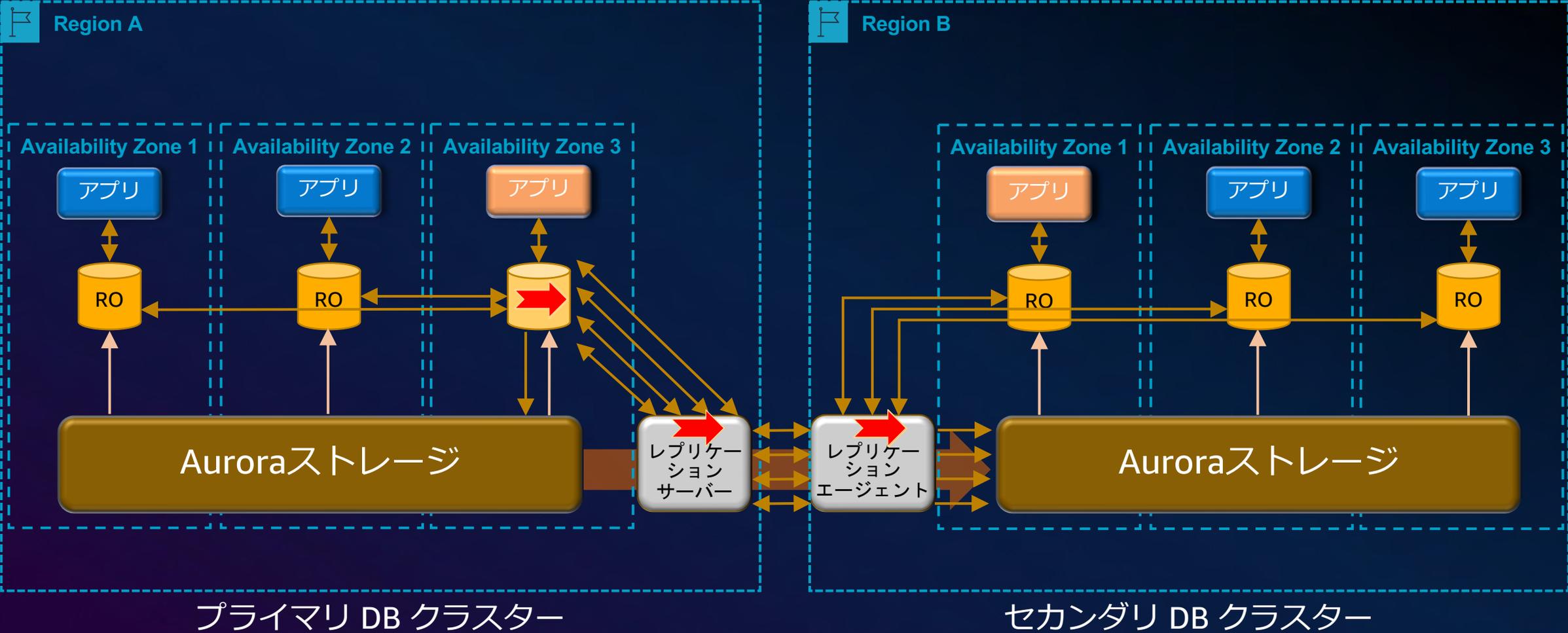
# Aurora Global Database - Write Forwarding

RW : ライター  
RO : リードレプリカ



# Aurora Global Database - Write Forwarding

RW : ライター  
RO : リードレプリカ

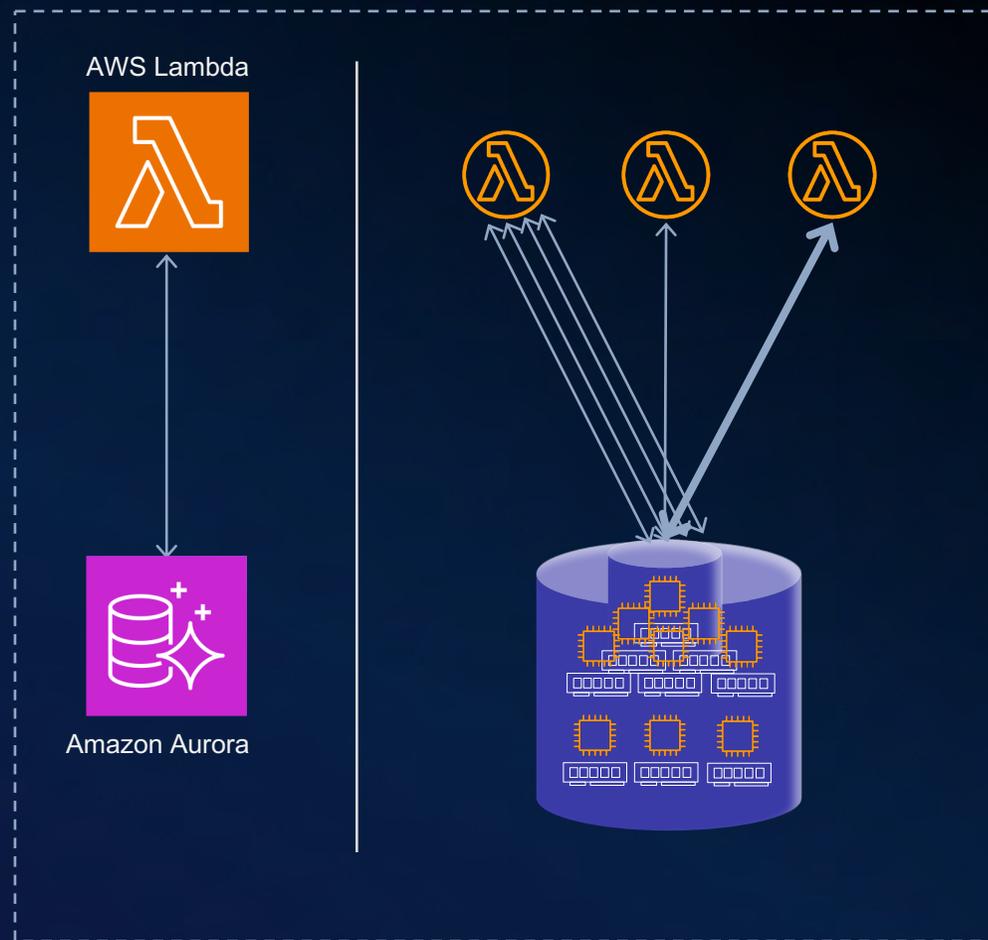


# 管理性 – Aurora Serverless

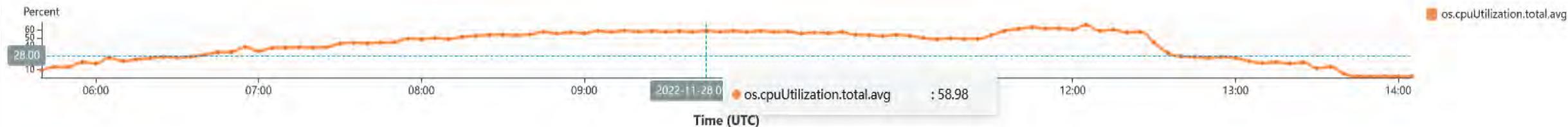


# Amazon Aurora Serverless v2

- CPU とメモリのリソースを追加することで **1 秒以内に自動的にスケールし**、秒単位で課金されます。
- 数十万のトランザクションを実行しても スケーリングによる **影響なし**
- シンプルな 1 秒ごとの **従量課金制**



# Aurora – ワークロード例



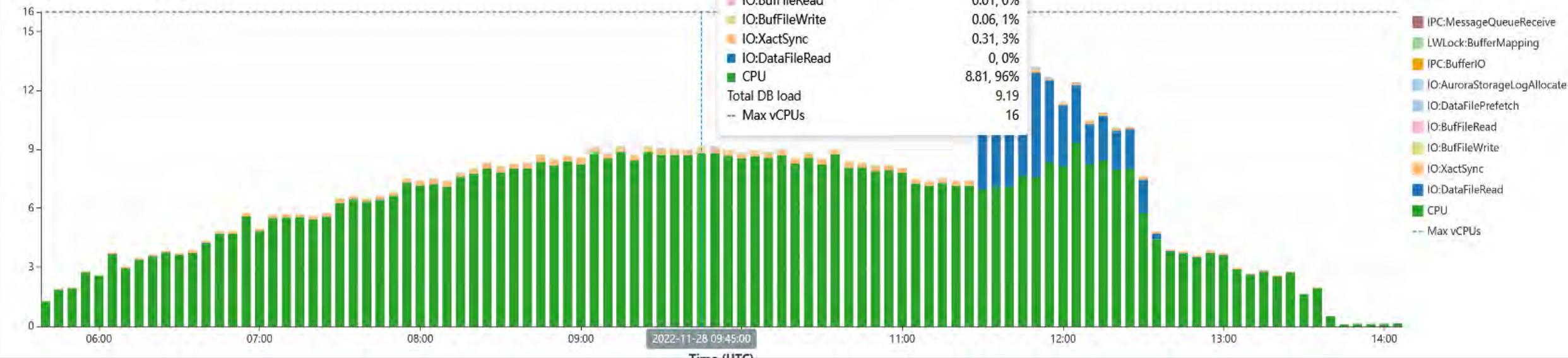
## Database load

Current activity measured in average active sessions (AAS) [Info](#)

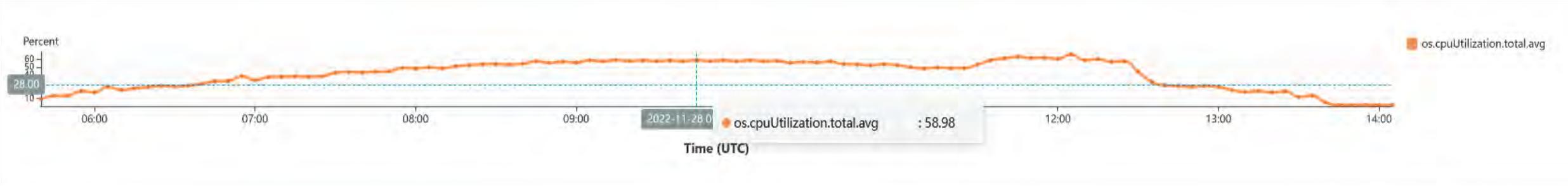
Show max vCPU

**db.r6g.4xlarge**

Average active sessions (AAS)



# Aurora – ワークロード例



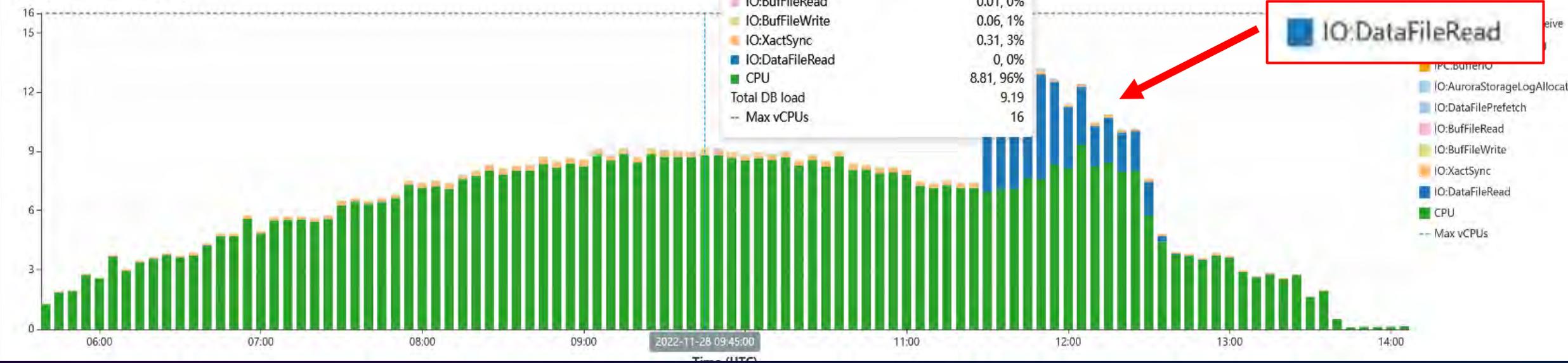
## Database load

Current activity measured in average active sessions (AAS) [Info](#)

Show max vCPU

**db.r6g.4xlarge**

Average active sessions (AAS)



# Aurora – ワークロード例



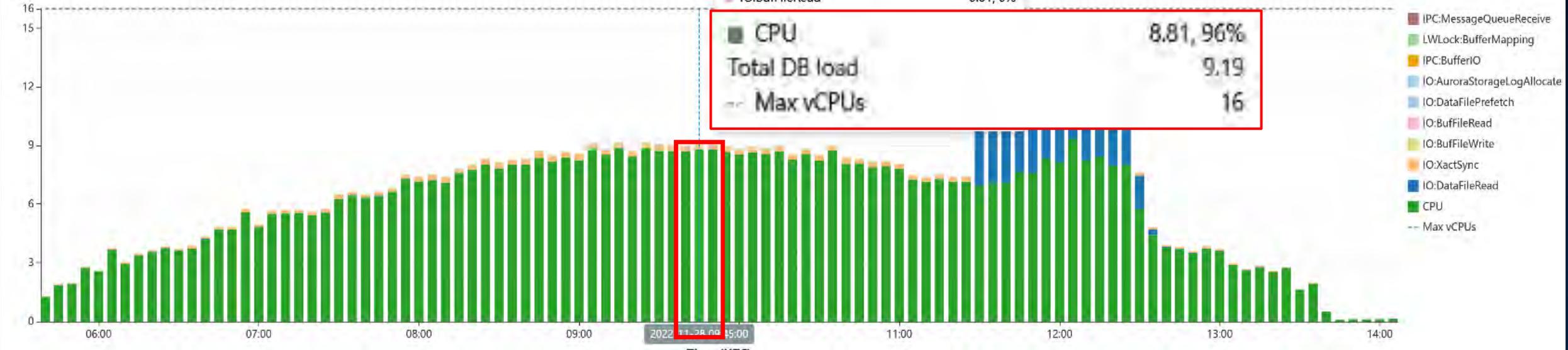
## Database load

Current activity measured in average active sessions (AAS) [Info](#)

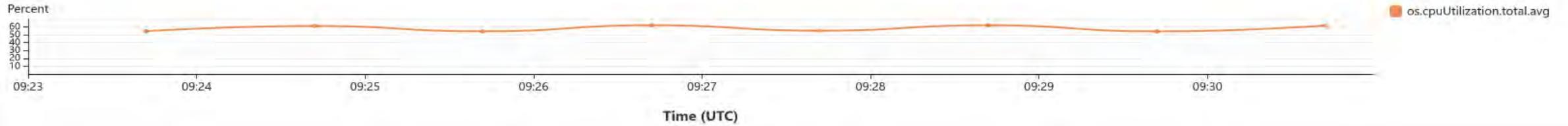
Show max vCPU

**db.r6g.4xlarge**

Average active sessions (AAS)



# Aurora – ワークロード例 (DB load過多)



## Database load

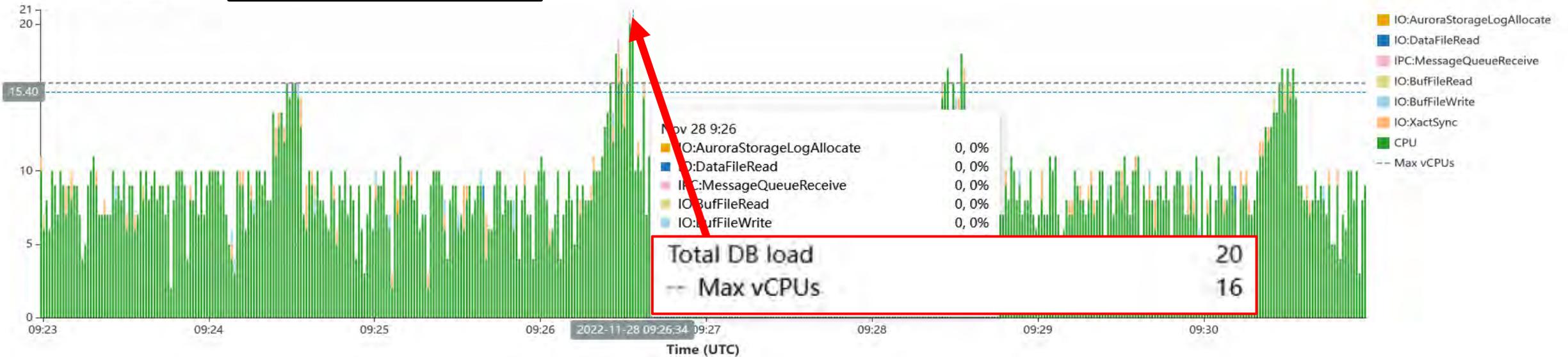
Current activity measured in average active sessions (AAS) [Info](#)

Chart type Bar Slice by Waits

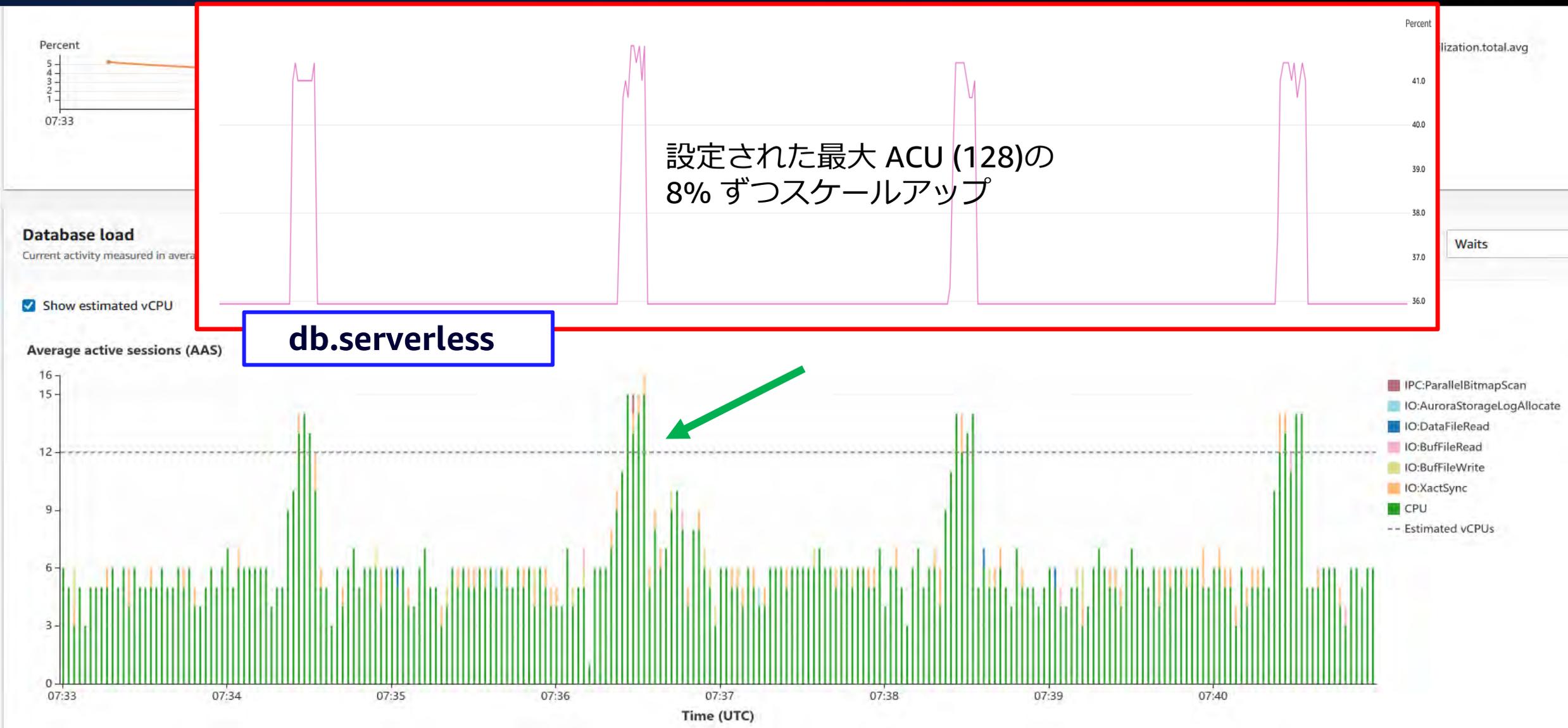
Show max vCPU

**db.r6g.4xlarge**

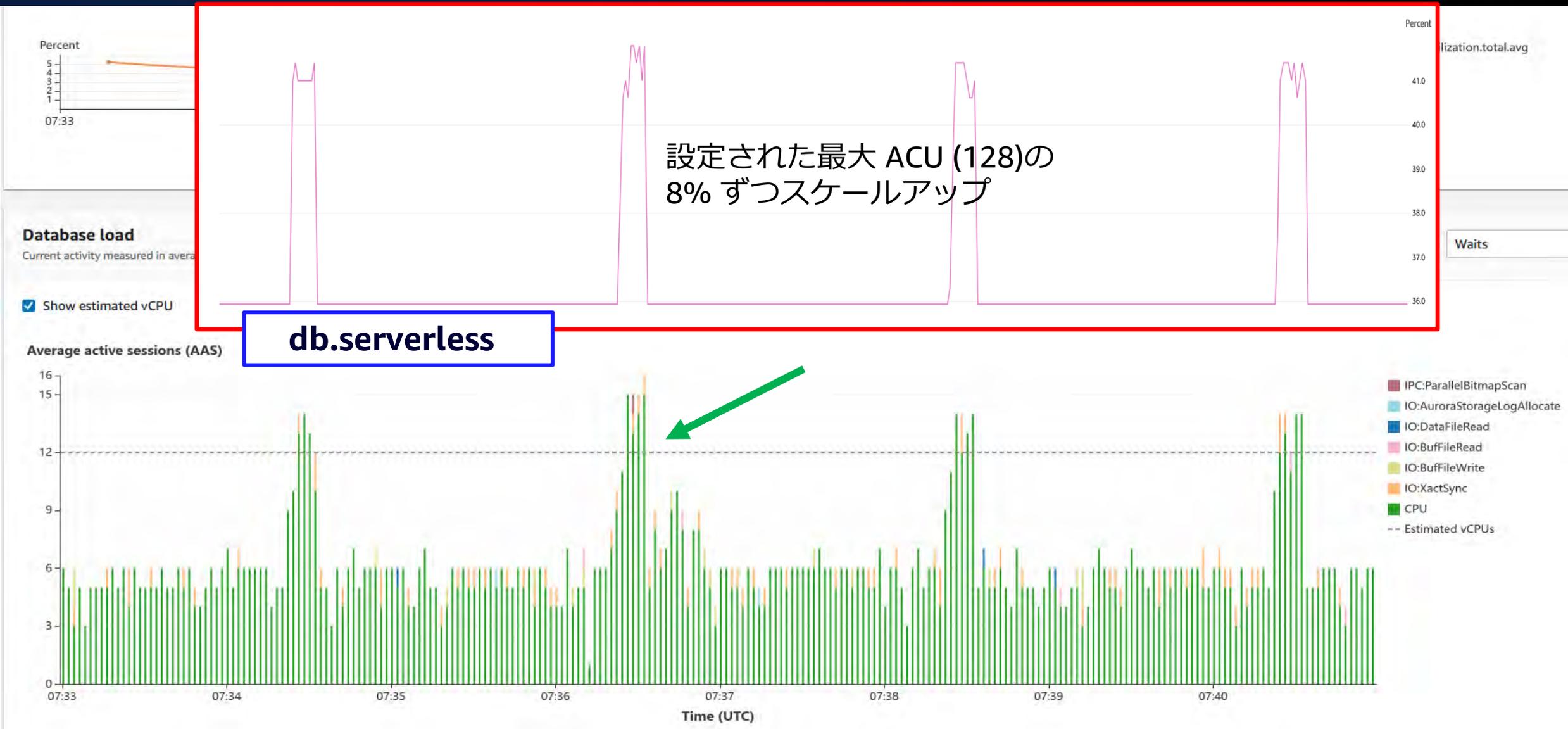
Average active sessions (AAS)



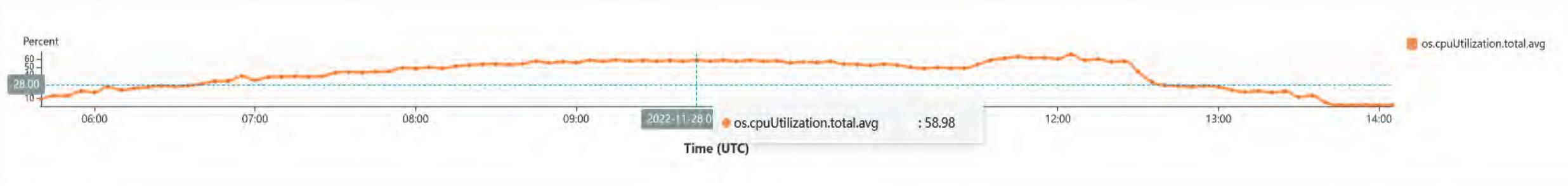
# Aurora Serverless – CPU スケール



# Aurora Serverless – CPU スケール



# Aurora - ワークロード例



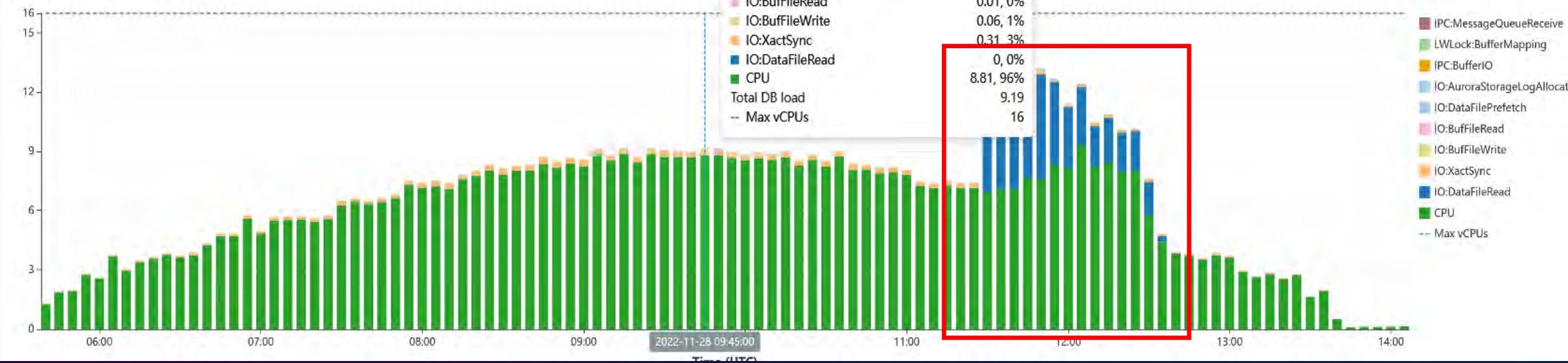
## Database load

Current activity measured in average active sessions (AAS) [Info](#)

Show max vCPU

**db.r6g.4xlarge**

Average active sessions (AAS)



# Aurora – ワークロード例 (IO待機)



## Database load

Current activity measured in average active sessions (AAS) [Info](#)

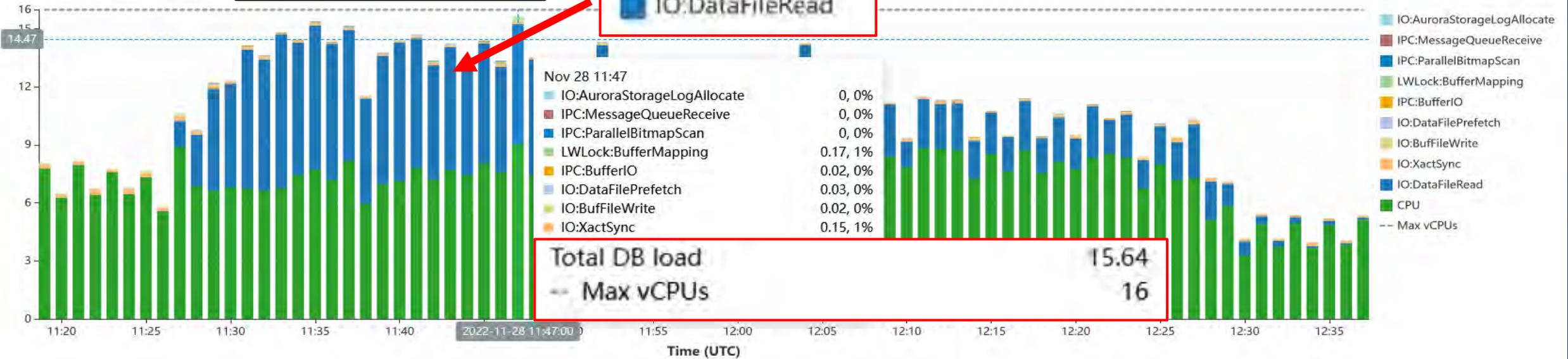
Chart type Bar Slice by Waits

Show max vCPU

Average active sessions (AAS)

db.r6g.4xlarge

IO:DataFileRead





# Aurora – ワークロード例 (IO待機)



## Database load

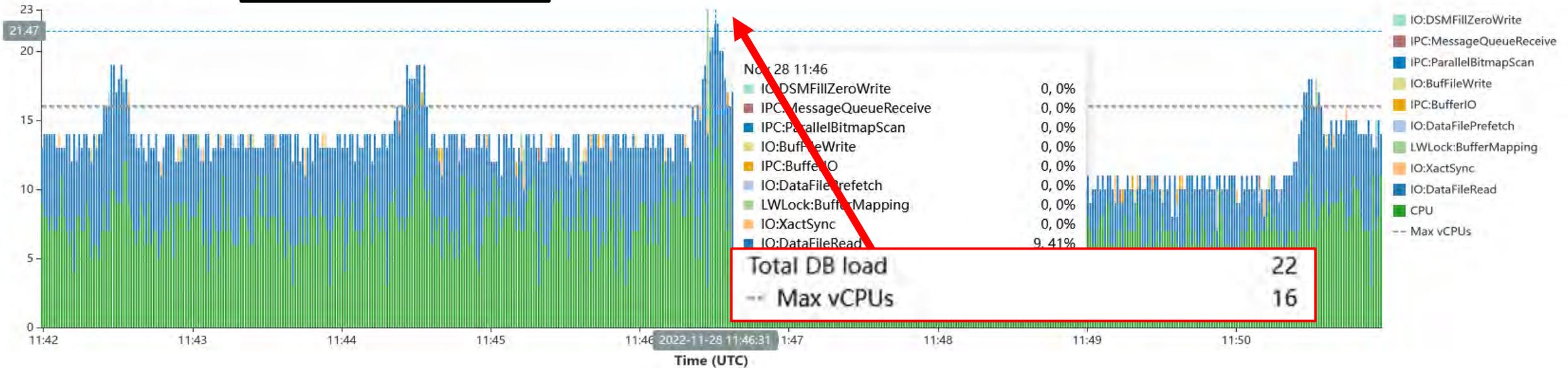
Current activity measured in average active sessions (AAS) [Info](#)

Chart type Bar Slice by Waits

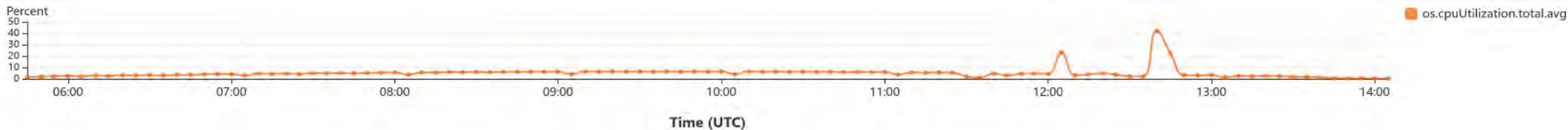
Show max vCPU

Average active sessions (AAS)

**db.r6g.4xlarge**



# Aurora Serverless – メモリと CPU のスケール



## Database load

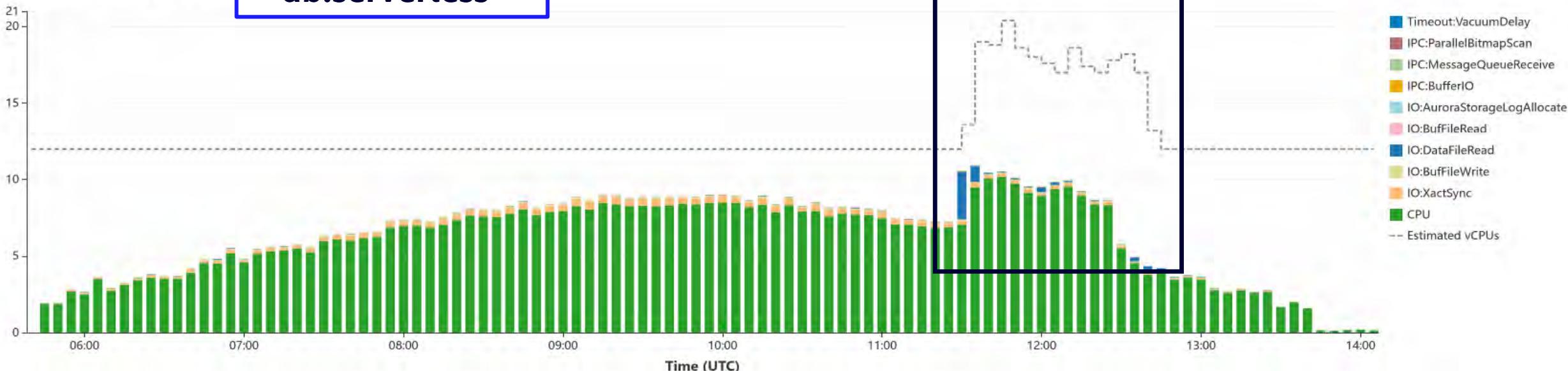
Current activity measured in average active sessions (AAS) [Info](#)

Chart type **Bar** Slice by **Waits**

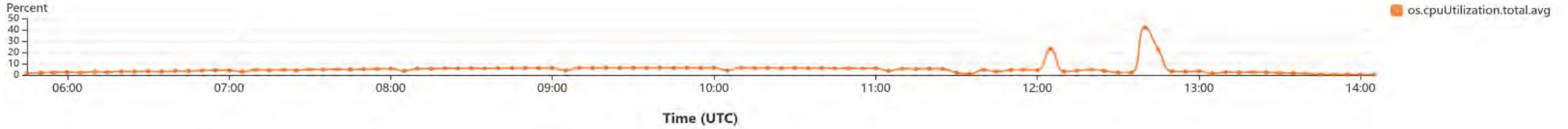
Show estimated vCPU

Average active sessions (AAS)

**db.serverless**



# Aurora Serverless – メモリと CPU のスケール



## Database load

Current activity measured in average active sessions (AAS) [Info](#)

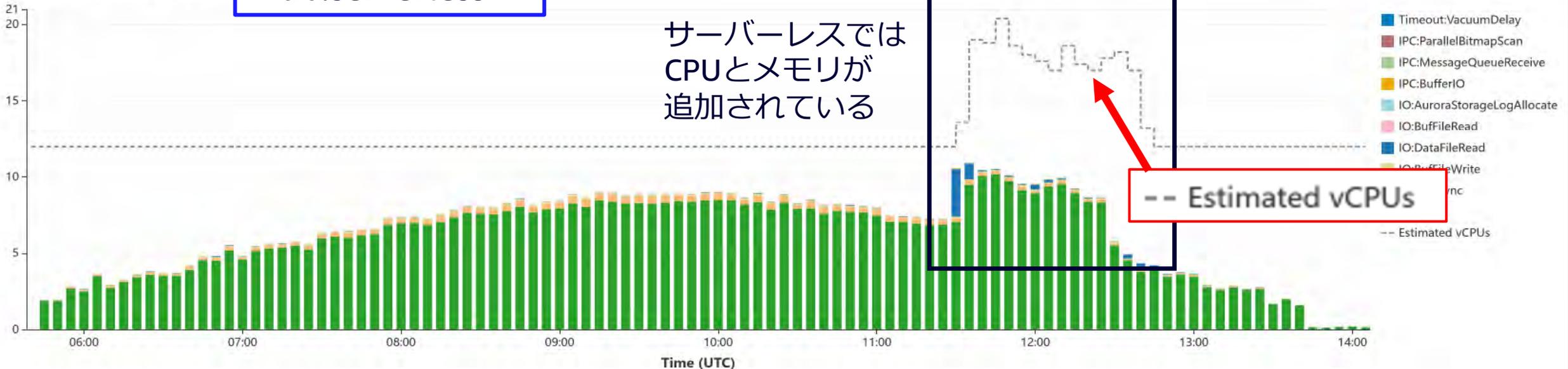
Chart type Bar Slice by Waits

Show estimated vCPU

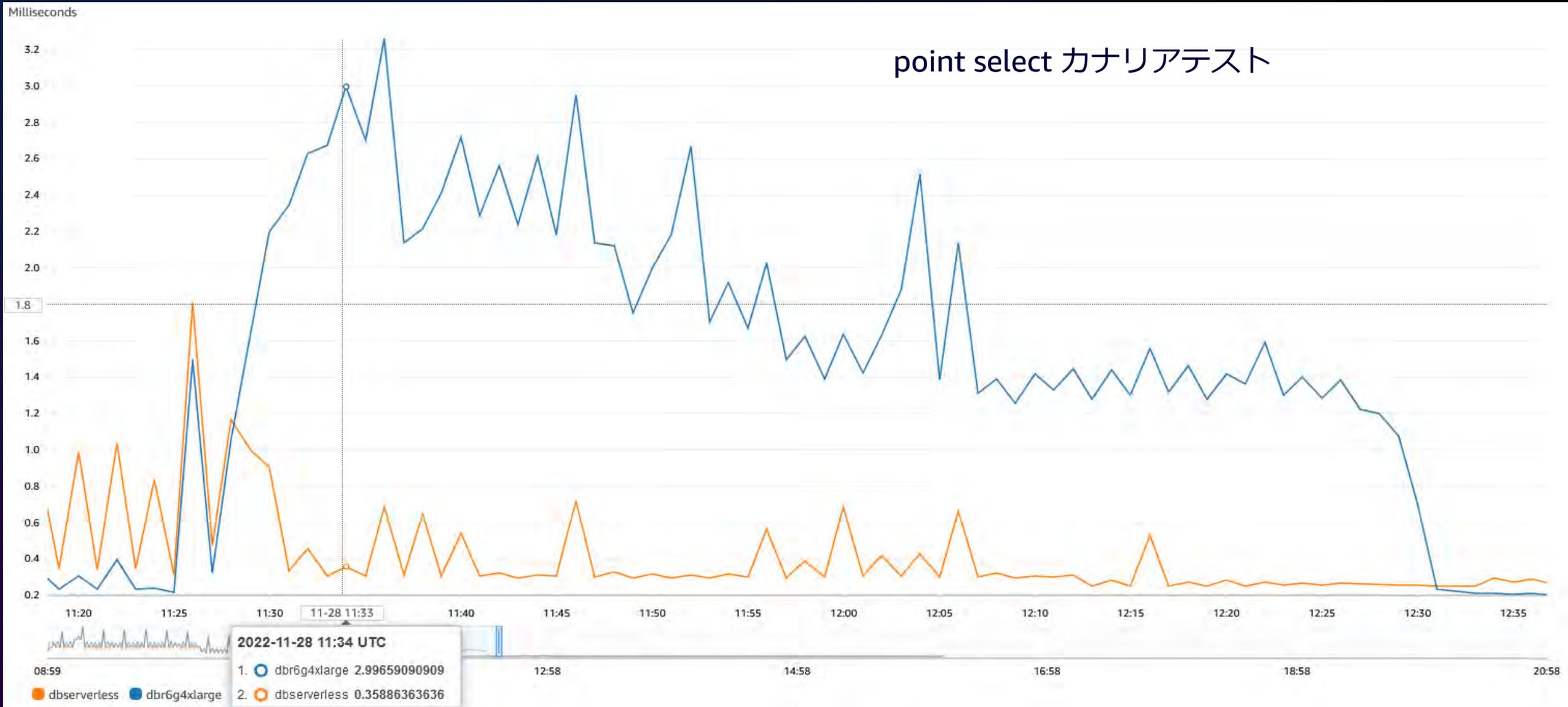
Average active sessions (AAS)

db.serverless

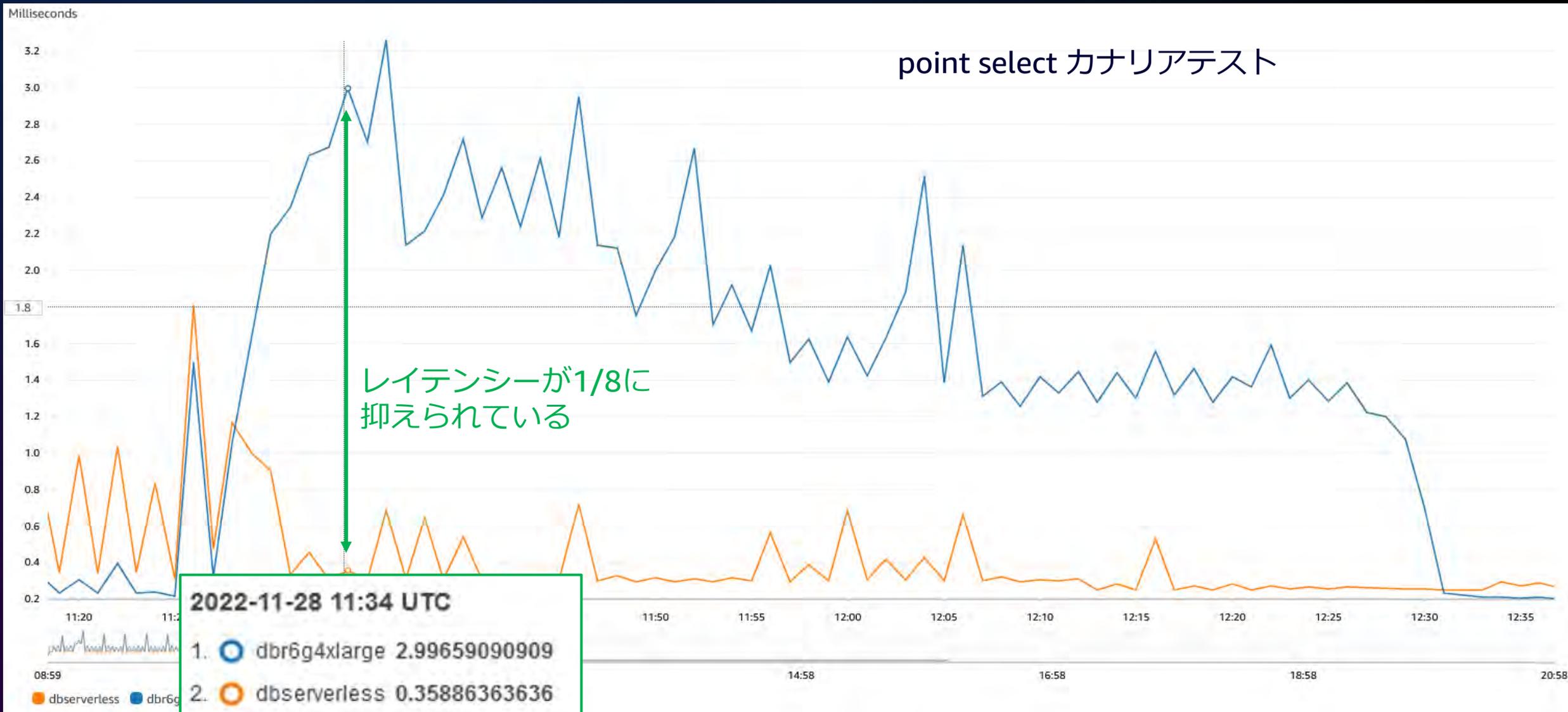
サーバーレスでは  
CPUとメモリが  
追加されている



# Aurora Serverless – カナリアテスト比較



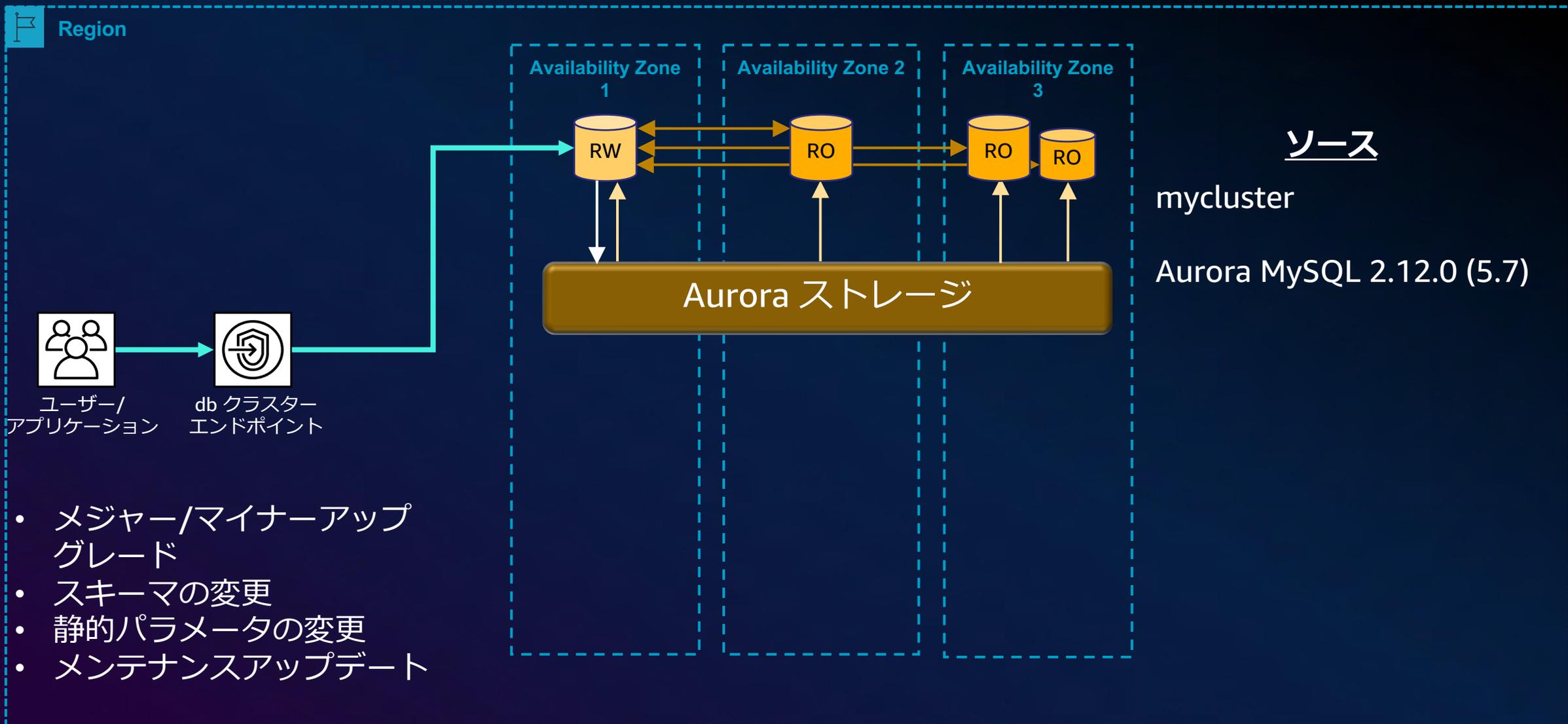
# Aurora Serverless – カナリアテスト比較



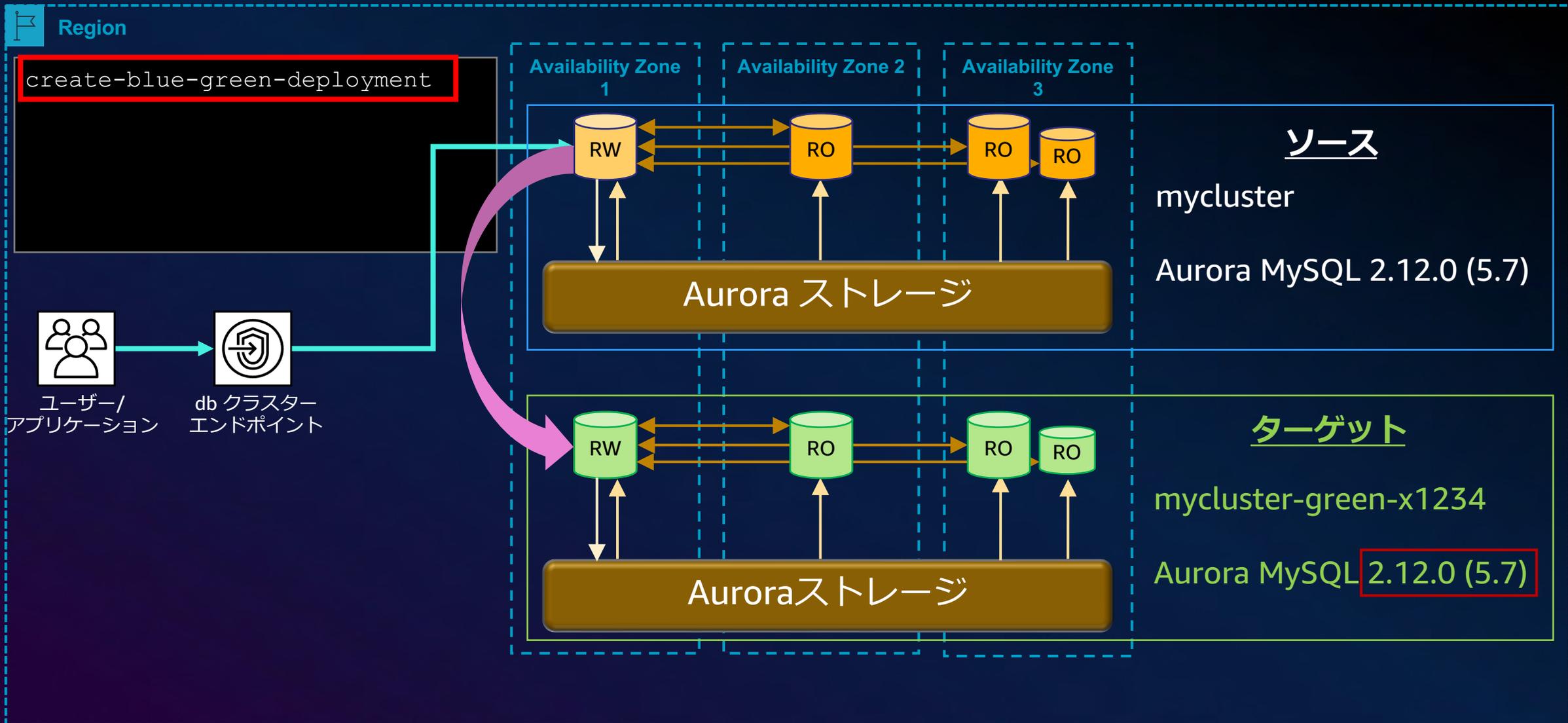
# 管理性 – Managed Blue/Green deployment



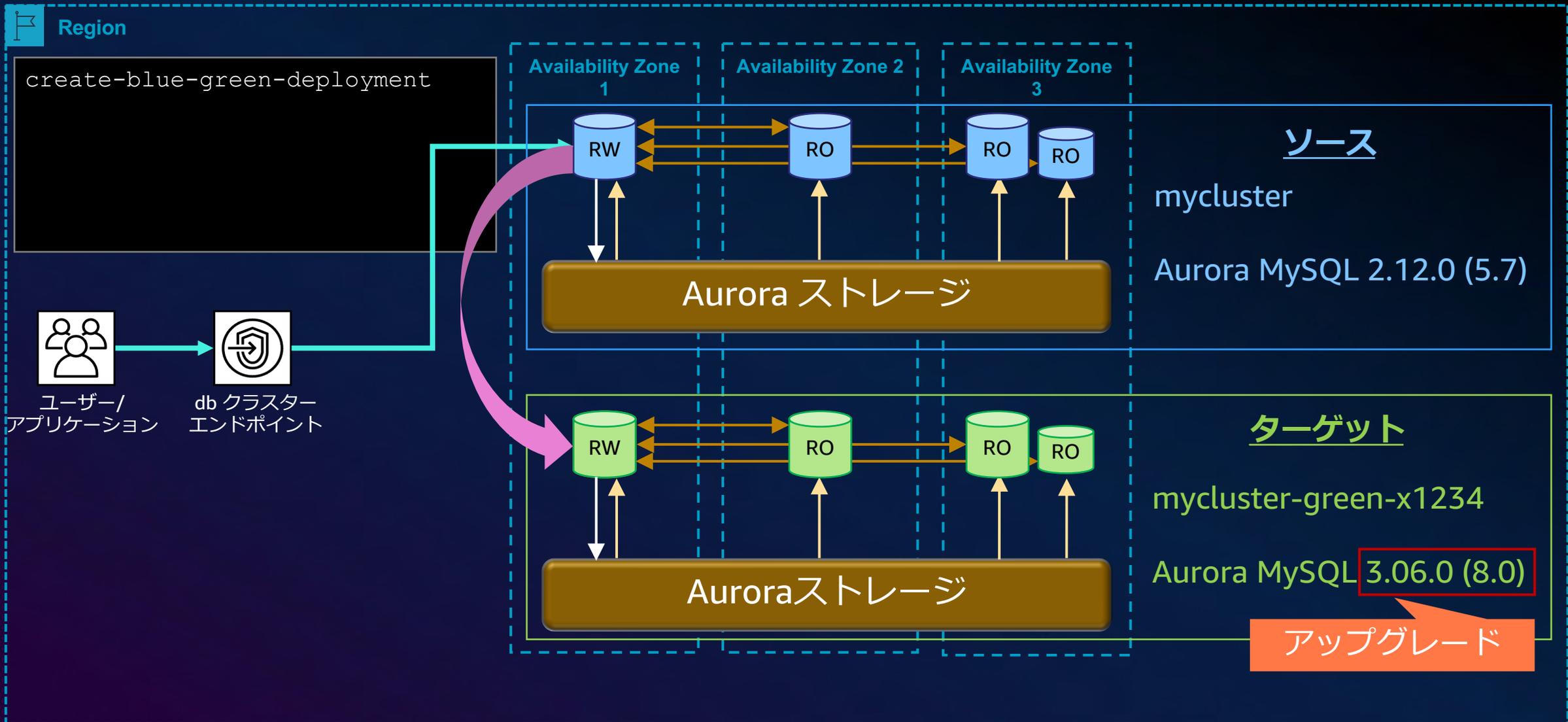
# Amazon Aurora Managed Blue/Green deployment



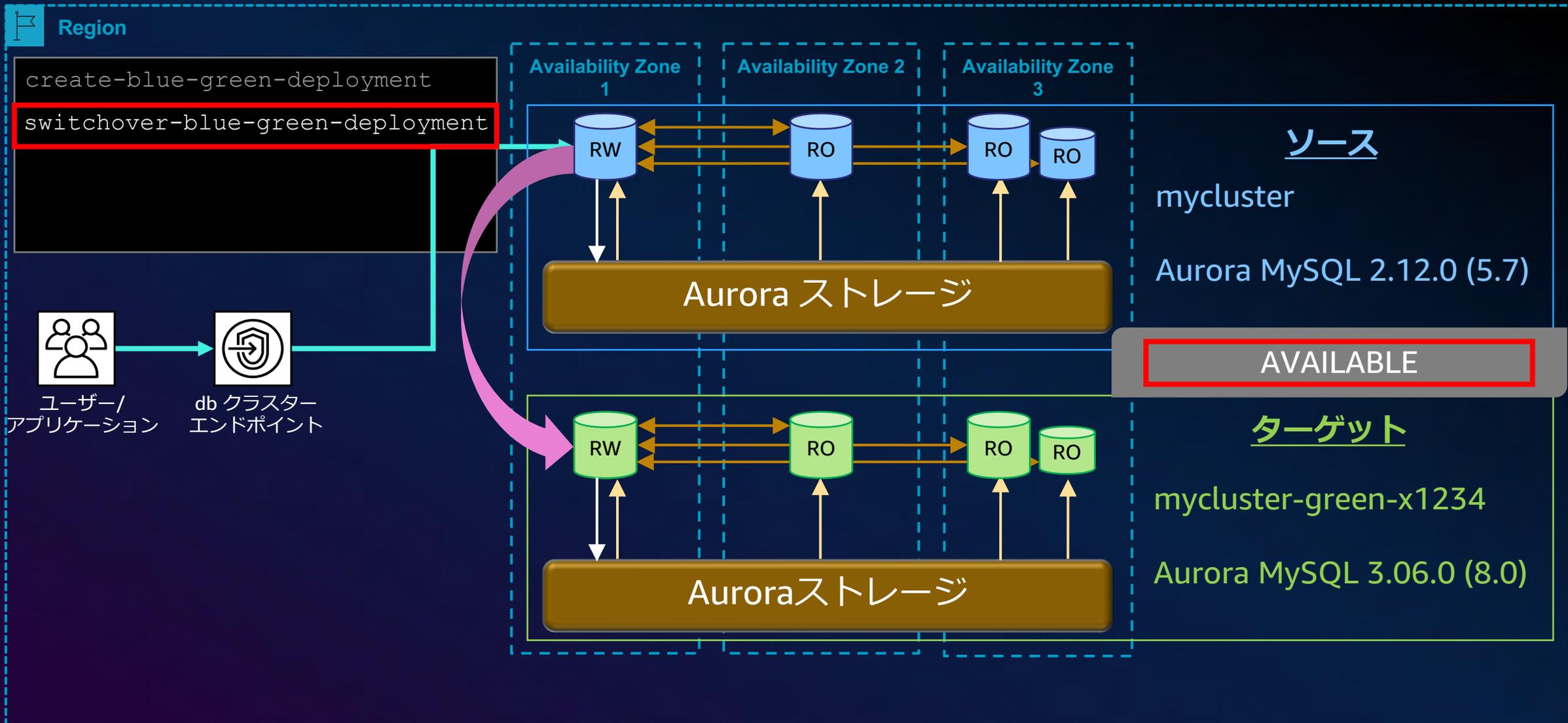
# Amazon Aurora Managed Blue/Green deployment



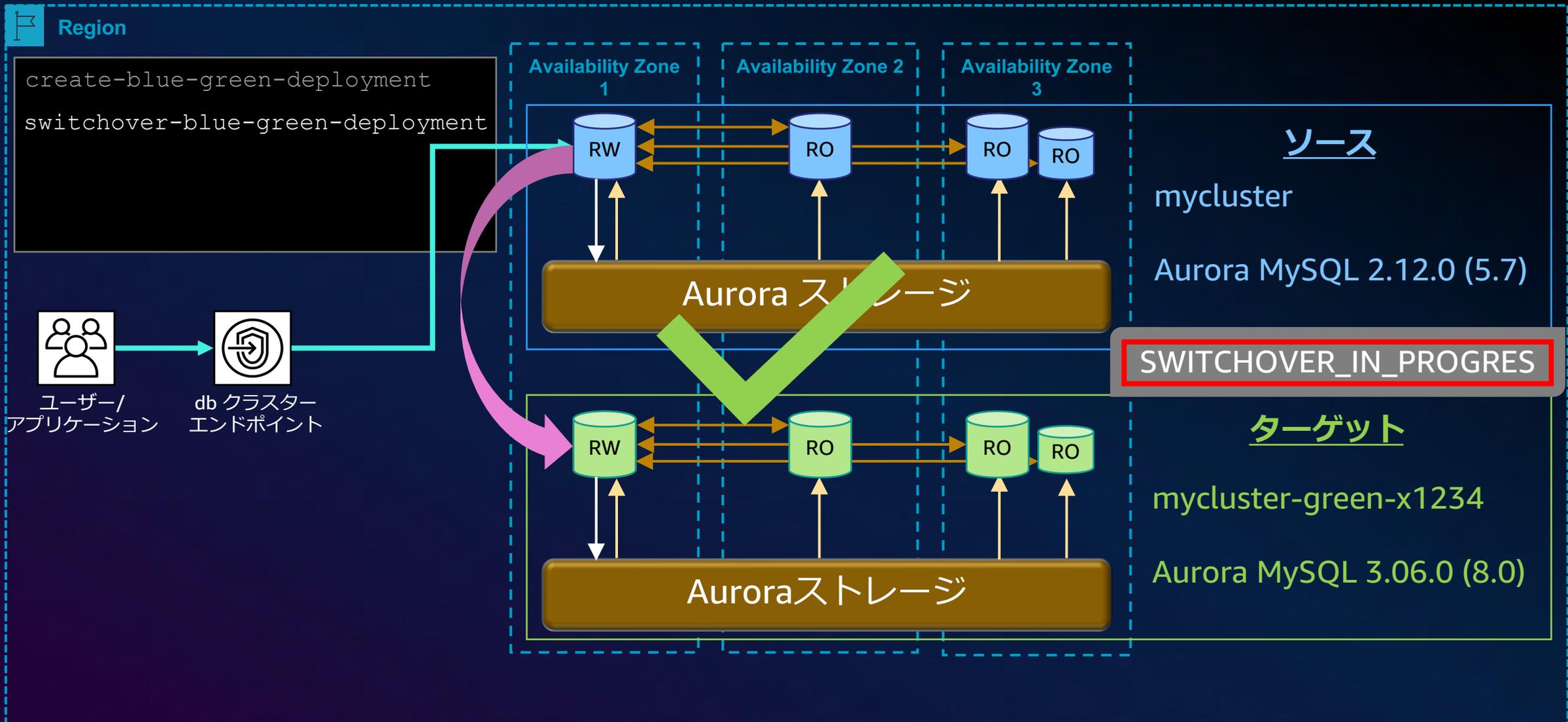
# Amazon Aurora Managed Blue/Green deployment



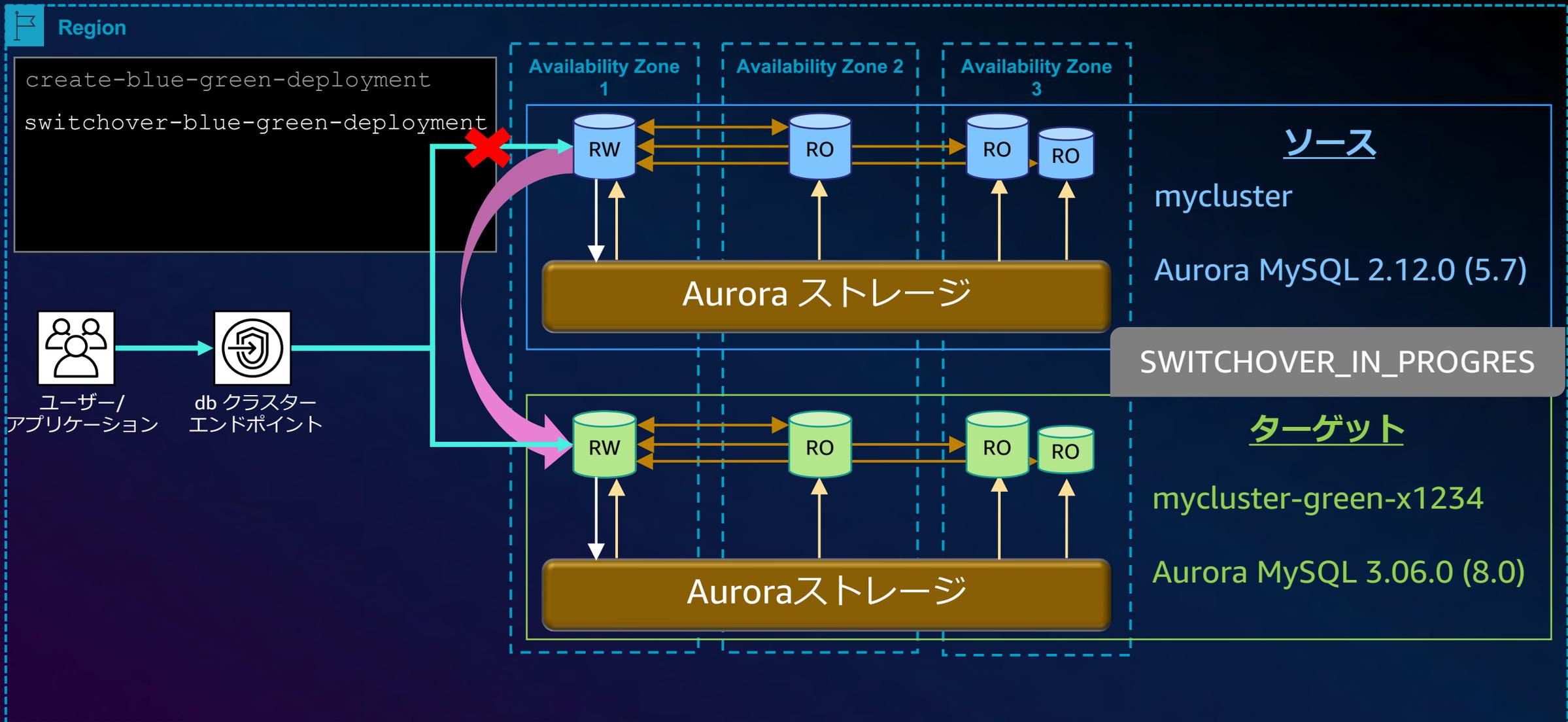
# Amazon Aurora Managed Blue/Green deployment



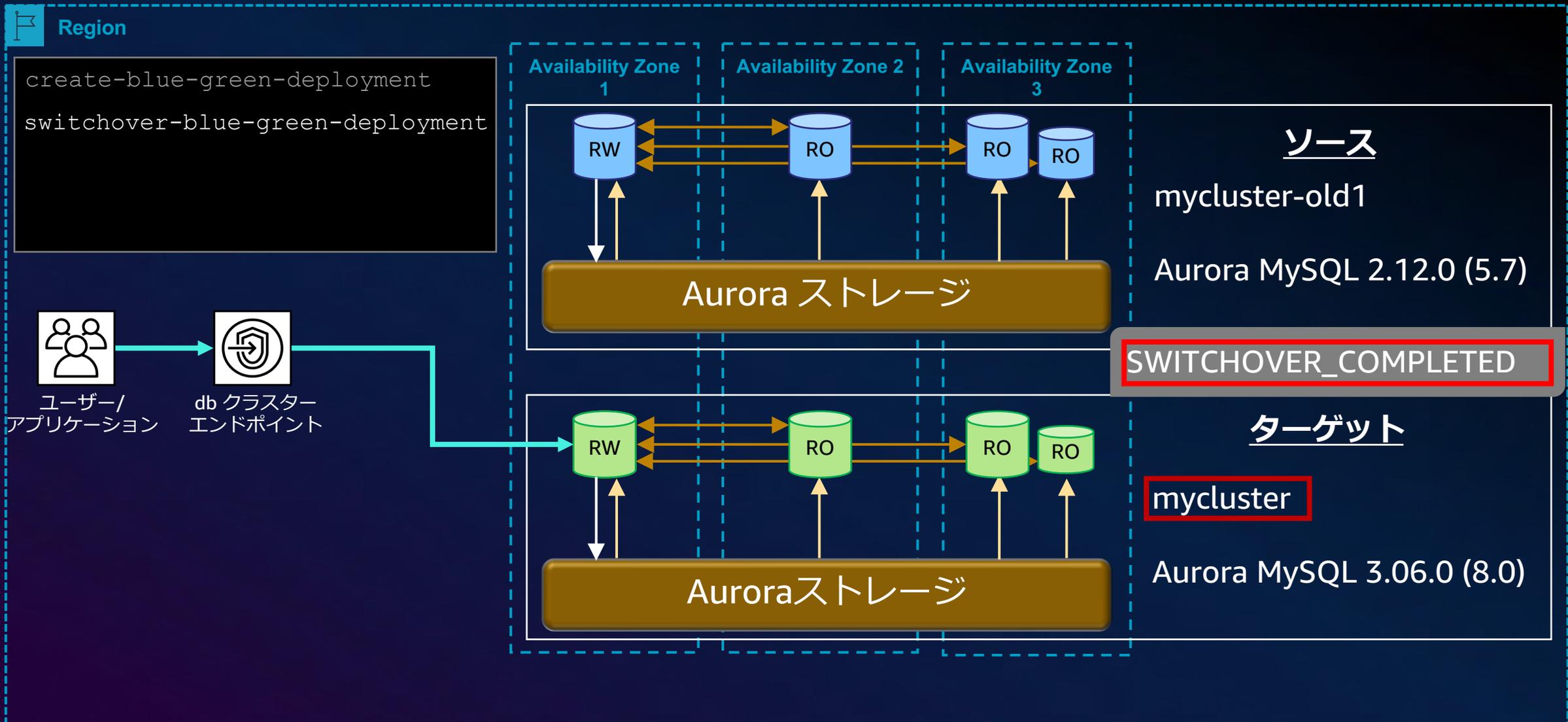
# Amazon Aurora Managed Blue/Green deployment



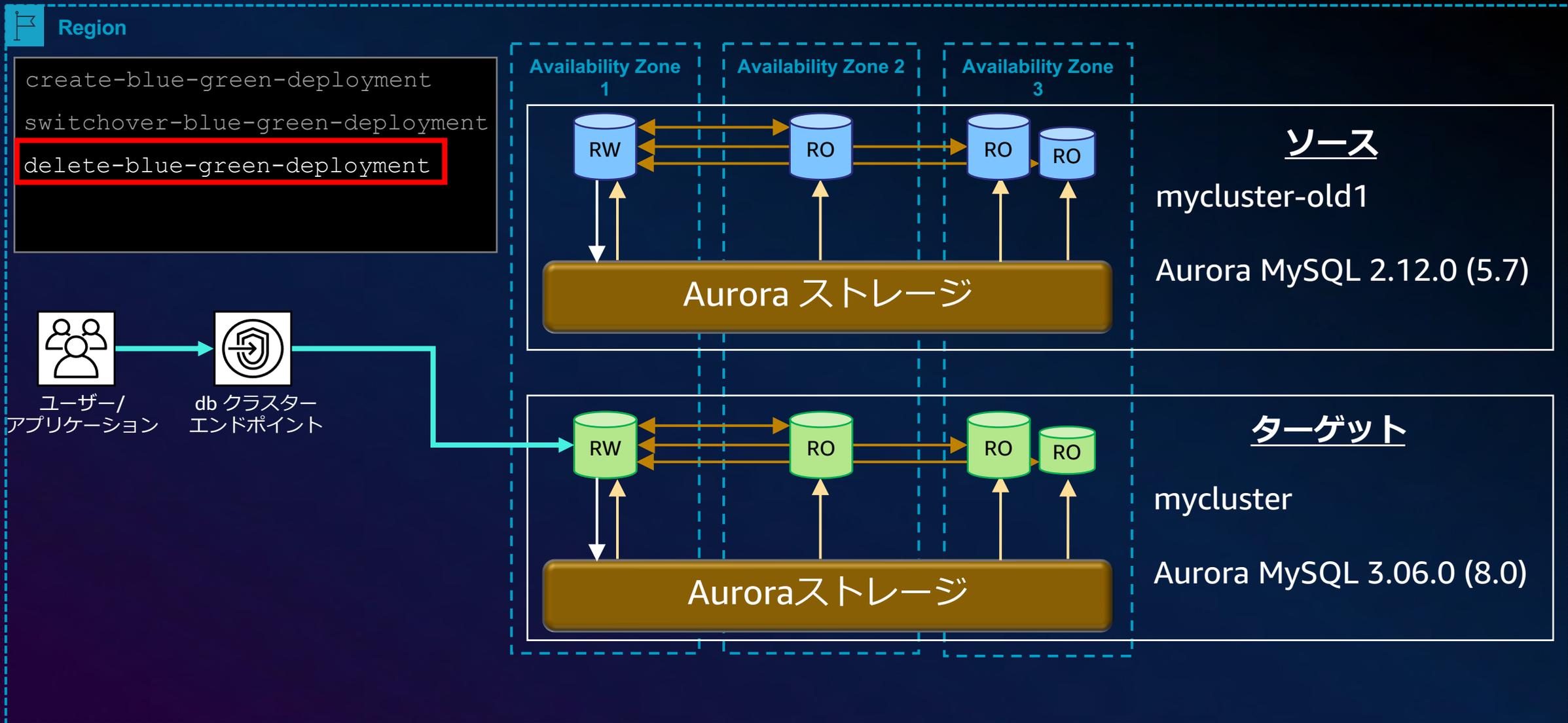
# Amazon Aurora Managed Blue/Green deployment



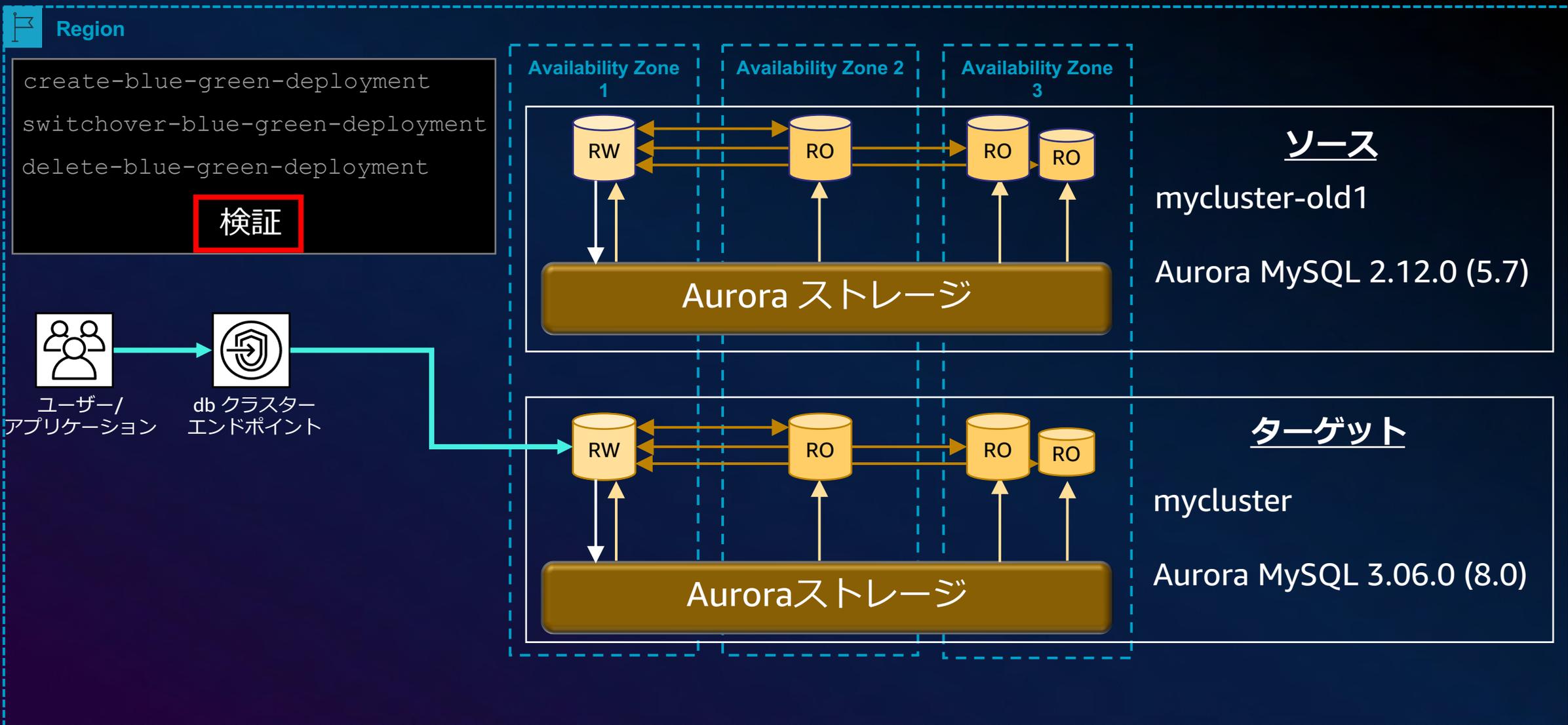
# Amazon Aurora Managed Blue/Green deployment



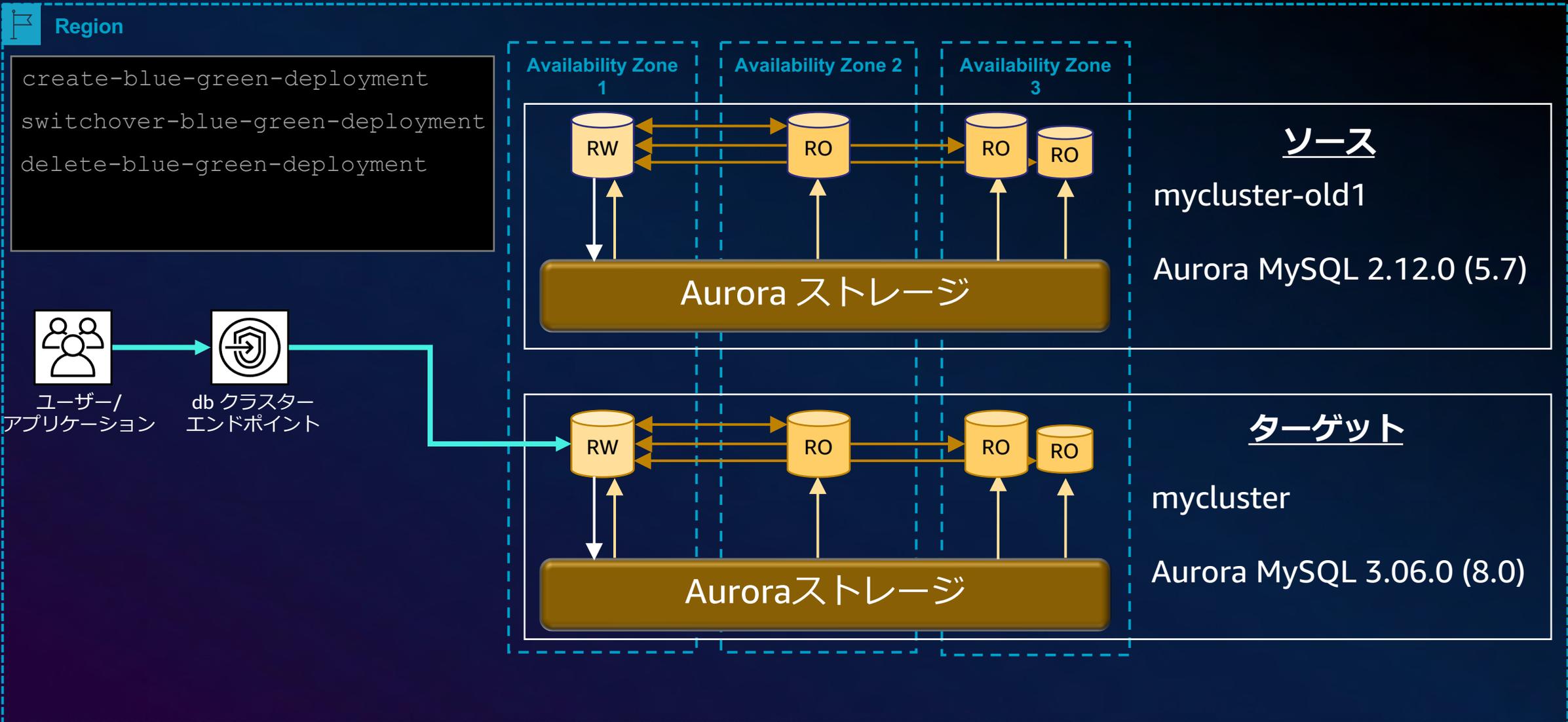
# Amazon Aurora Managed Blue/Green deployment



# Amazon Aurora Managed Blue/Green deployment



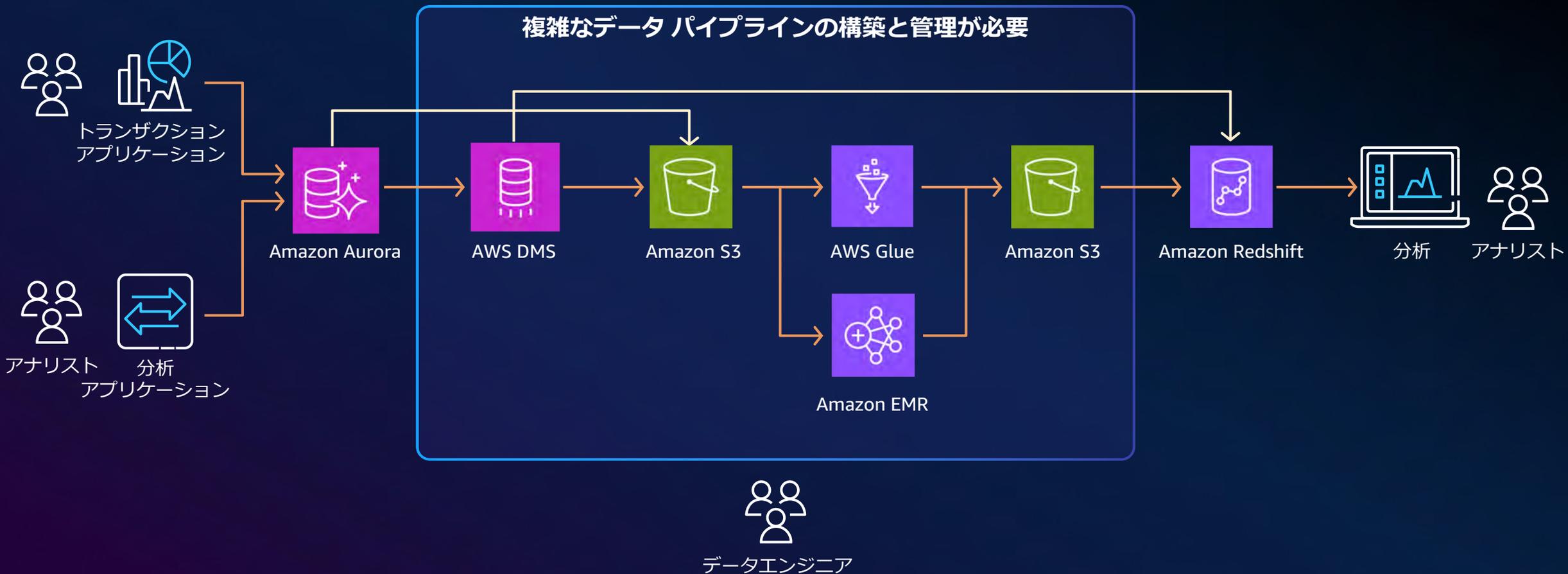
# Amazon Aurora Managed Blue/Green deployment



# 管理性 – Amazon Redshift との Amazon Aurora ゼロ ETL 統合



# 大規模なトランザクション分析



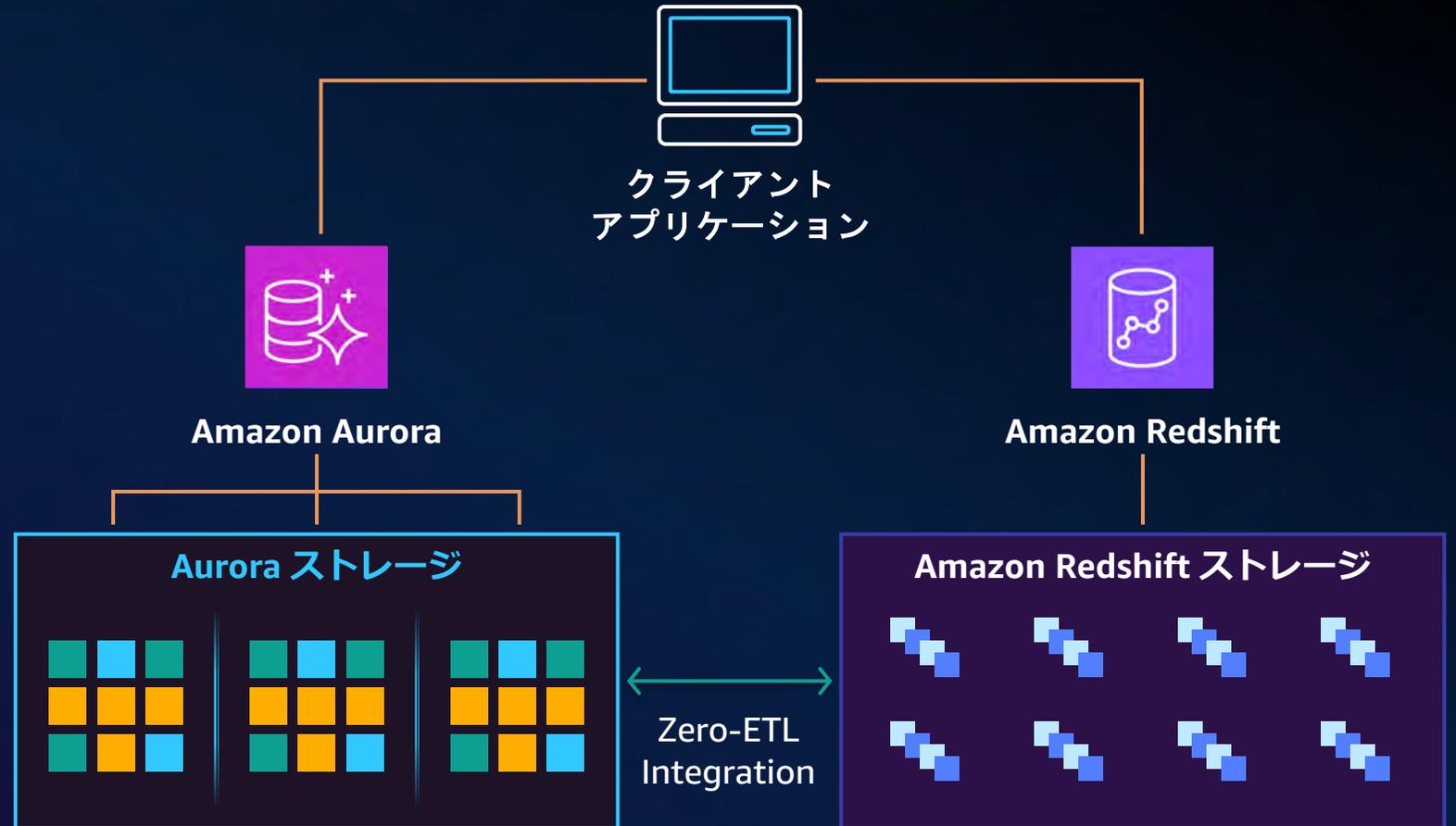
# Amazon Redshift との Amazon Aurora ゼロ ETL 統合の導入

ペタバイト規模のトランザクション データに対してほぼリアルタイムの分析を可能にする簡単かつ安全な方法



# シンプルさとパフォーマンスを重視した設計

- Amazon Redshift 統合ターゲットを簡単に作成
- ストレージ層での自動データシーディングと継続的レプリケーション
- 取り込みと並行して分析を実行する
- Amazon Redshift システムテーブルと Amazon CloudWatch から進行状況、遅延、パフォーマンスを監視します



# 新しい Aurora ストレージタイプ



# Aurora I/O-Optimized

## Auroraクラスターストレージ

**Standard**

**I/O-Optimized**

- **コンピューティング** (オンデマンド / RI)
- **ストレージ** (標準 - 従量課金制)
- **I/O** (従量課金制)

# Aurora I/O-Optimized

## Aurora クラスター ストレージ

### Standard

- コンピューティング (オンデマンド / RI)
- ストレージ (標準 - 従量課金制)
- I/O (従量課金制)

### I/O-Optimized

- コンピューティング (オンデマンド / RI) + 30%
- ストレージ (標準 - 従量課金制) + 125%
- I/O - コストなし

# Amazon Aurora I/O-Optimized

すべてのワークロードに対する予測可能な価格

IO 負荷の高いワークロードのコストパフォーマンスの向上

新しいクラスター構成により、顧客はコンピューティングとストレージの料金のみを支払うことができ、読み取り/書き込み IO の料金は発生しません。

## Cluster configuration - new [Info](#)

Choose the Aurora cluster configuration that best fits your application's price predictability and price-performance needs. The storage type of your Aurora database is automatically set based on the configuration you choose.

### Configuration options

Database instance, storage, and IO charges will vary based on the configuration. Review [Amazon Aurora pricing](#) for more information.

#### Aurora Standard

- Cost-effective pricing for many applications with moderate IO usage (IO spend is less than an est. 25% of database spend).
- Pay-per-request IO charges apply. Database instance and storage price does not include IO usage.

#### Aurora IO-optimized

- Predictable pricing for all applications. Improved price-performance for IO-intensive applications (IO spend exceeds an est. 25% of database spend).
- No additional charges for read/write IOs. Database instances and storage price includes IO usage.

すべてのワークロードに対する予測可能な価格

IO 支出が Aurora データベースの合計支出の 25% を超えた場合に、最大 40% のコスト削減により価格パフォーマンスが向上します。

Aurora Serverless v2、オンデマンド、およびリザーブドインスタンス全体の Aurora PostgreSQL および Aurora MySQL で利用可能

リザーブドインスタンスを使用すると、お客様はさらに IO を節約できます

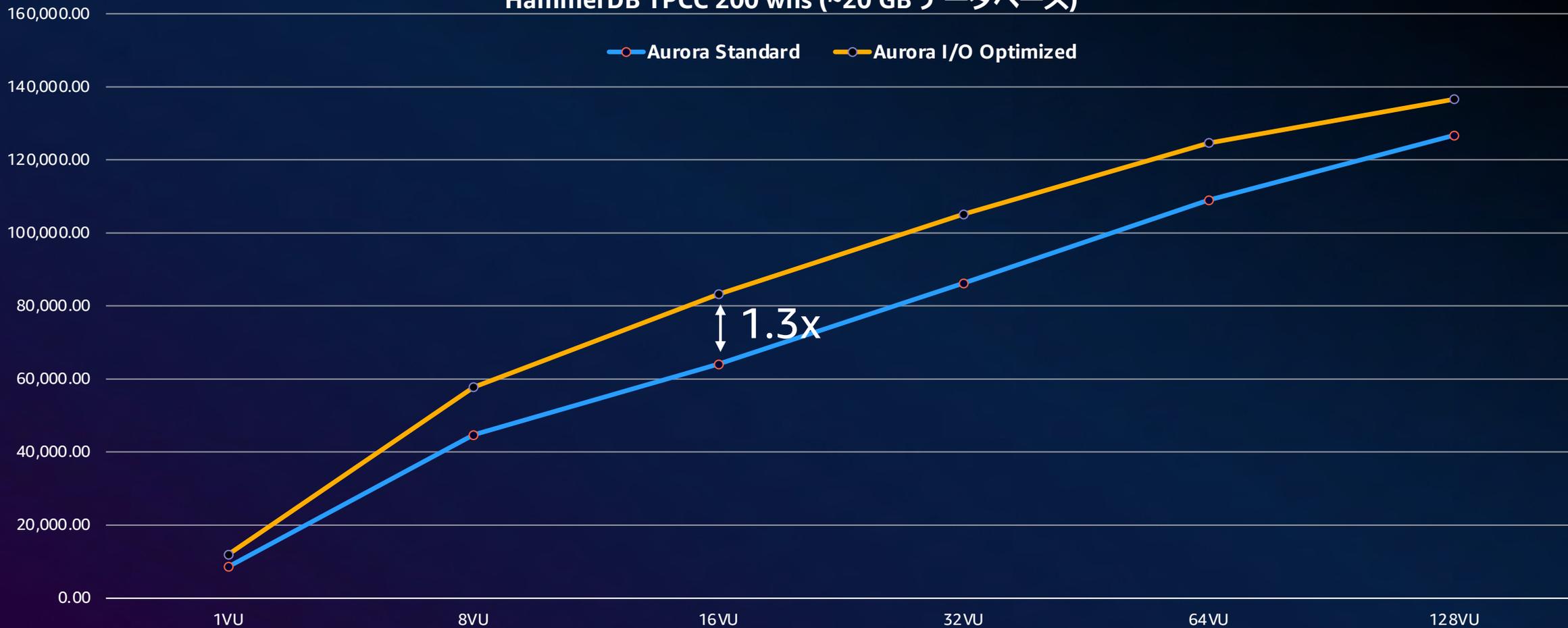
# Aurora I/O-Optimized

- Aurora I/O-Optimized は **クラスターストレージ構成** です
- Aurora クラスターは月に 1 回 **ストレージオプション (標準から I/O 最適化)** を変更し、いつでも元に戻すことができます
- **Aurora PostgreSQL 13.x 以降**、および **Aurora MySQL 3.0.3.1 以降** で利用可能
- 以下の互換性があります
  - Intel ベースの Aurora データベース インスタンス タイプ (t3、r5、r6i)
  - Graviton ベースのデータベース インスタンス タイプ (t4g、r6g、r7g、x2g)
  - **Aurora サーバーレス v2**
- Aurora Global Database クラスターはクラスターレベルで異なる **Aurora ストレージ構成** を持つことができます。つまり、プライマリクラスターとセカンダリクラスターは異なる構成で構成できます。

# パフォーマンス

r6g.2xlarge での Aurora PostgreSQL 標準と I/O 最適化ストレージタイプの比較

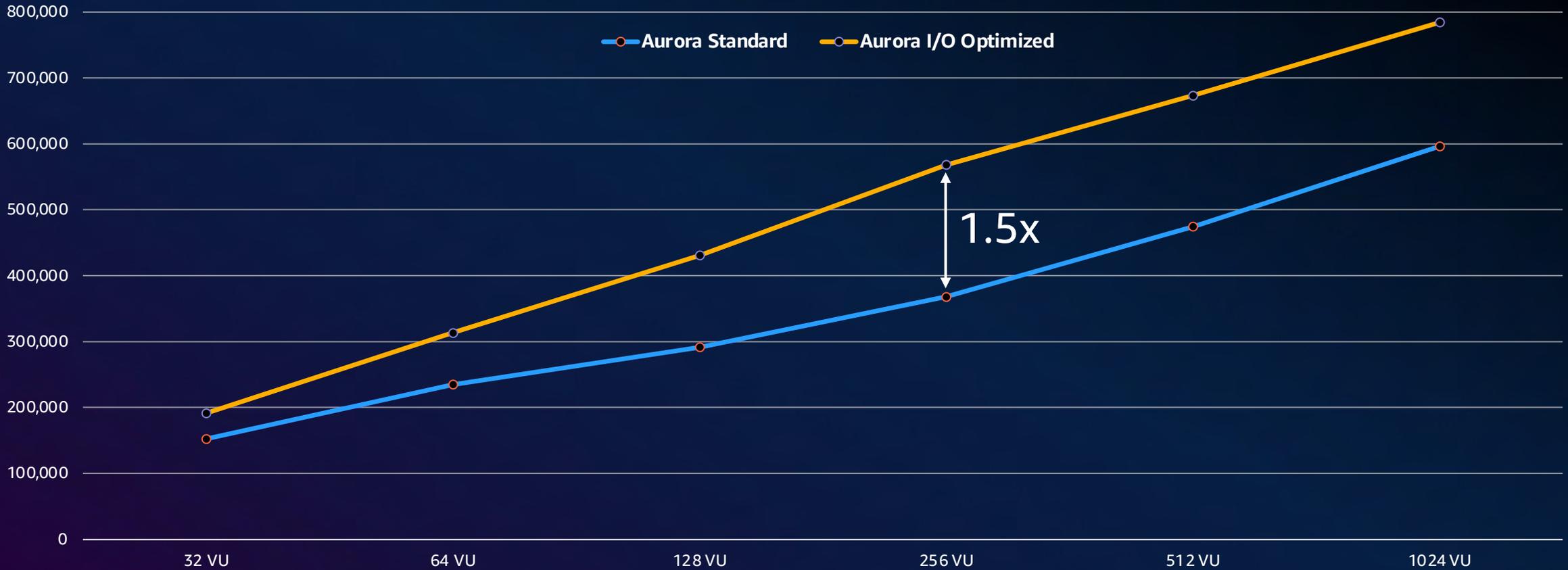
HammerDB TPCC 200 whs (~20 GB データベース)



# パフォーマンス

r6i.16xl での Aurora PostgreSQL 標準と I/O 最適化ストレージタイプの比較

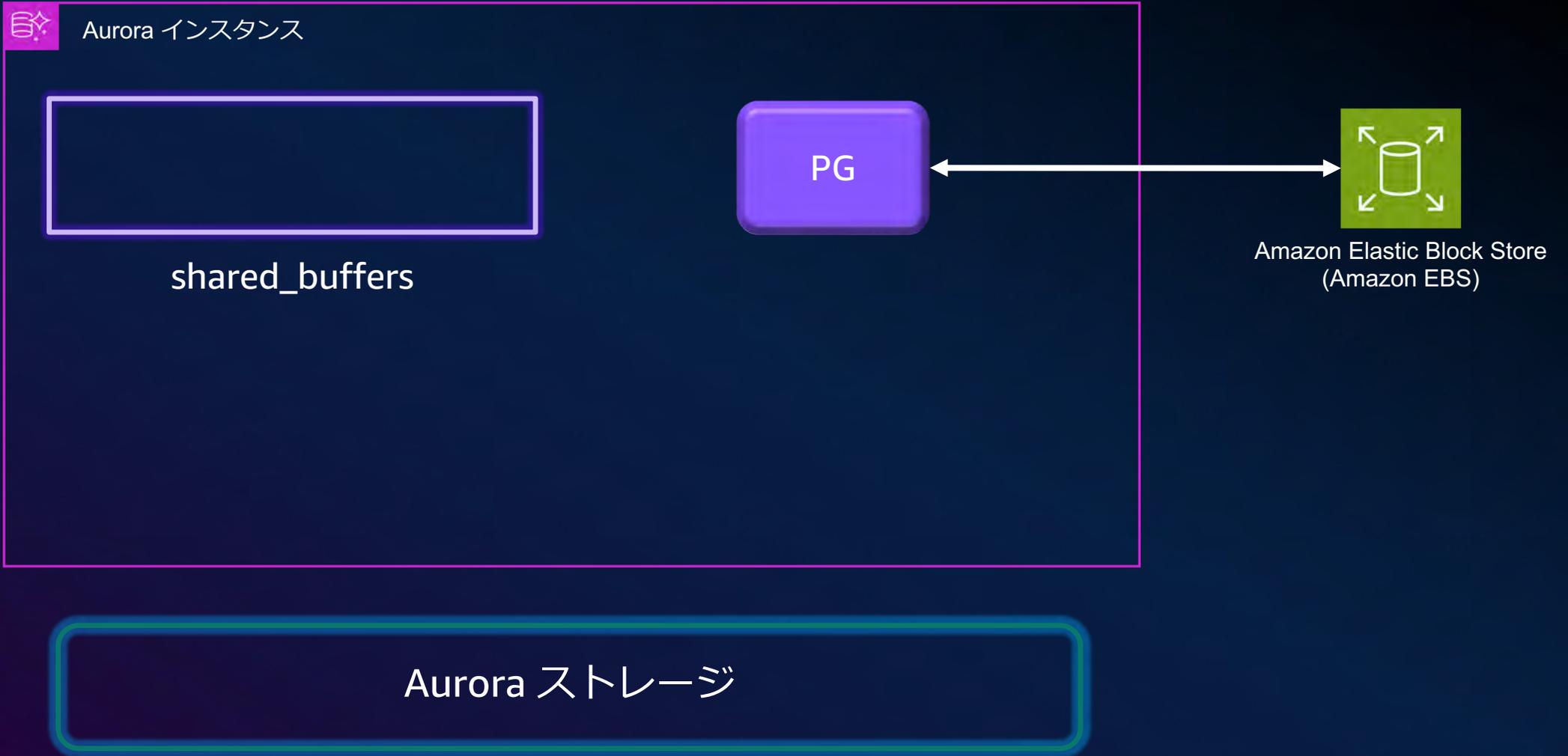
HammerDB TPCC 1000 whs (~100 GB データベース)



# Optimized Reads

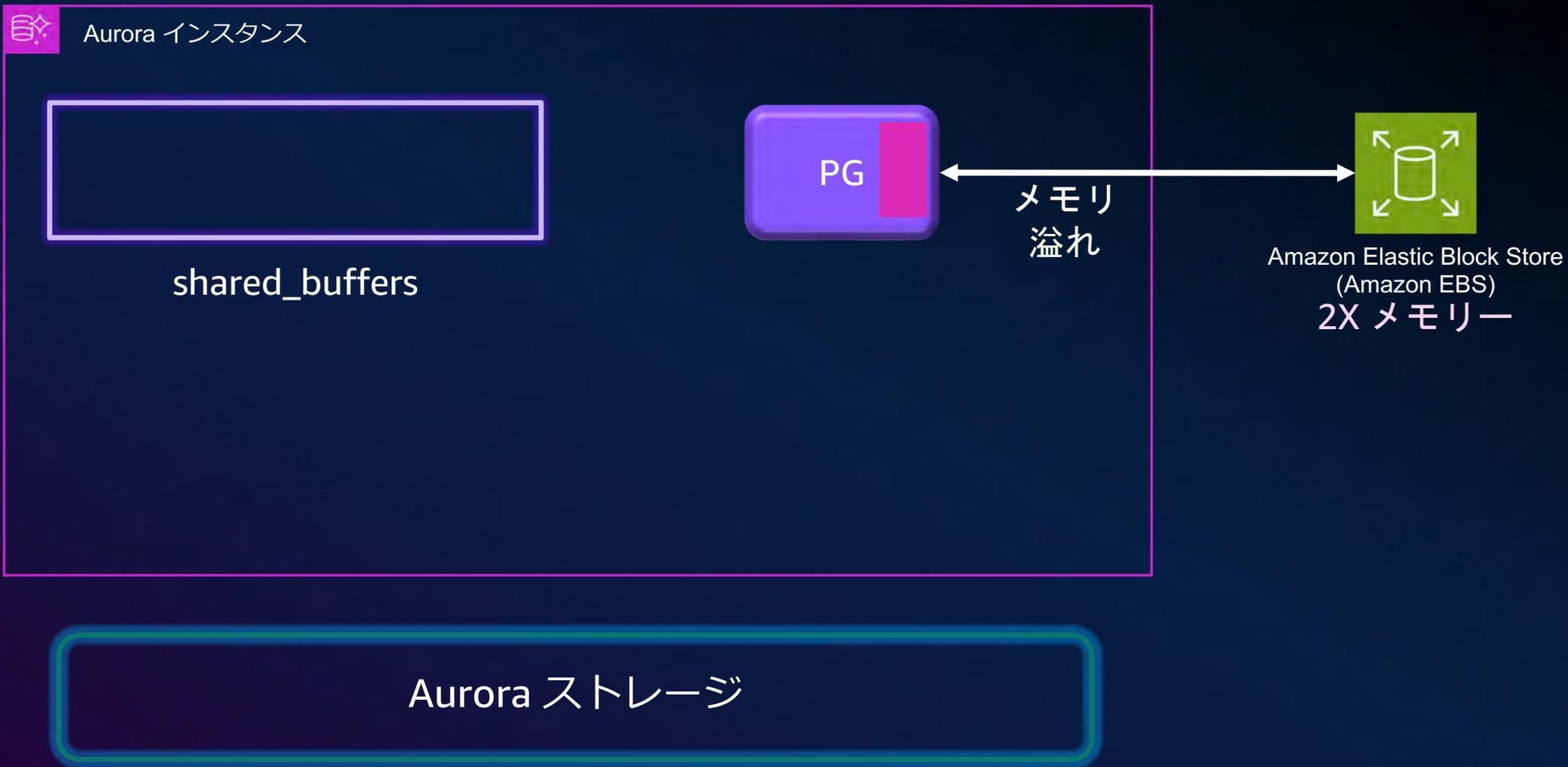
# Optimized Reads - 一時オブジェクト

PostgreSQL



# Optimized Reads - 一時オブジェクト

PostgreSQL



# Optimized Reads - 一時オブジェクト

PostgreSQL

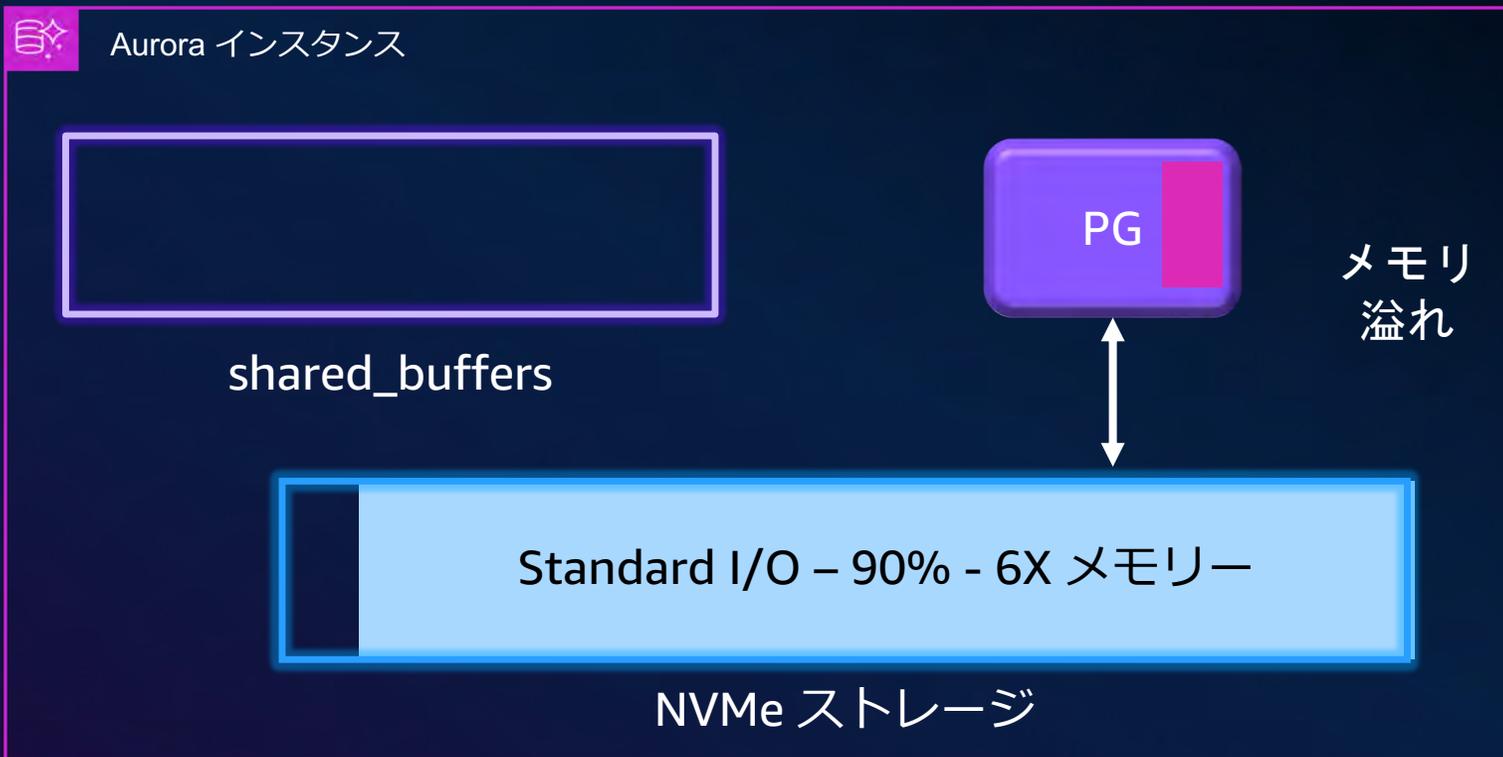


Amazon Elastic Block Store  
(Amazon EBS)  
2X メモリー

- 対象**
- db.r6gd
  - db.r6id
- Aurora  
PostgreSQL  
14.9+, 15.3+

# Optimized Reads - 一時オブジェクト

PostgreSQL



Amazon Elastic Block Store  
(Amazon EBS)  
2X メモリー

## 対象

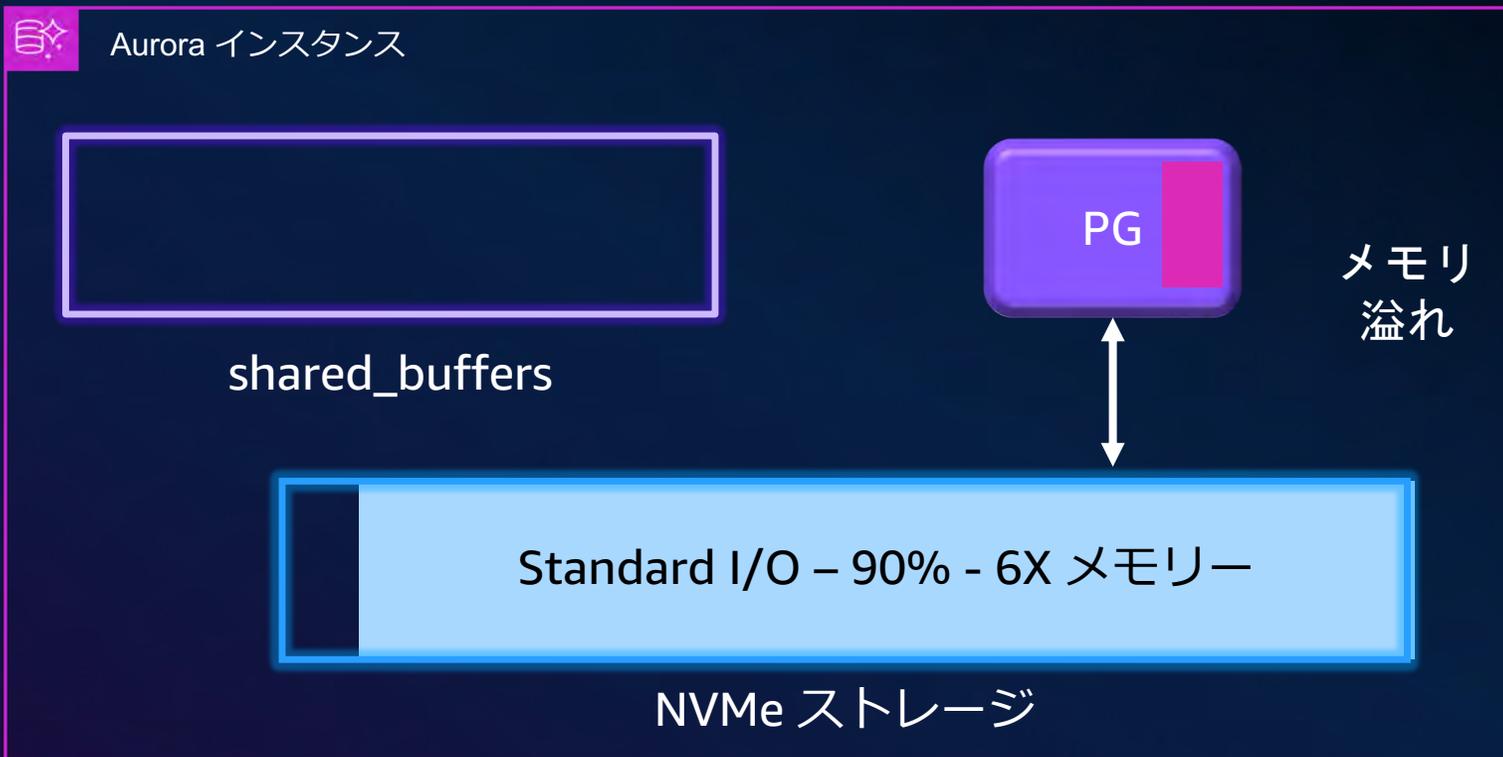
db.r6gd  
db.r6id

Aurora  
PostgreSQL  
14.9+, 15.3+

Aurora ストレージ

# Optimized Reads - 一時オブジェクト

PostgreSQL



Amazon Elastic Block Store  
(Amazon EBS)  
2X メモリー

Aurora ストレージ

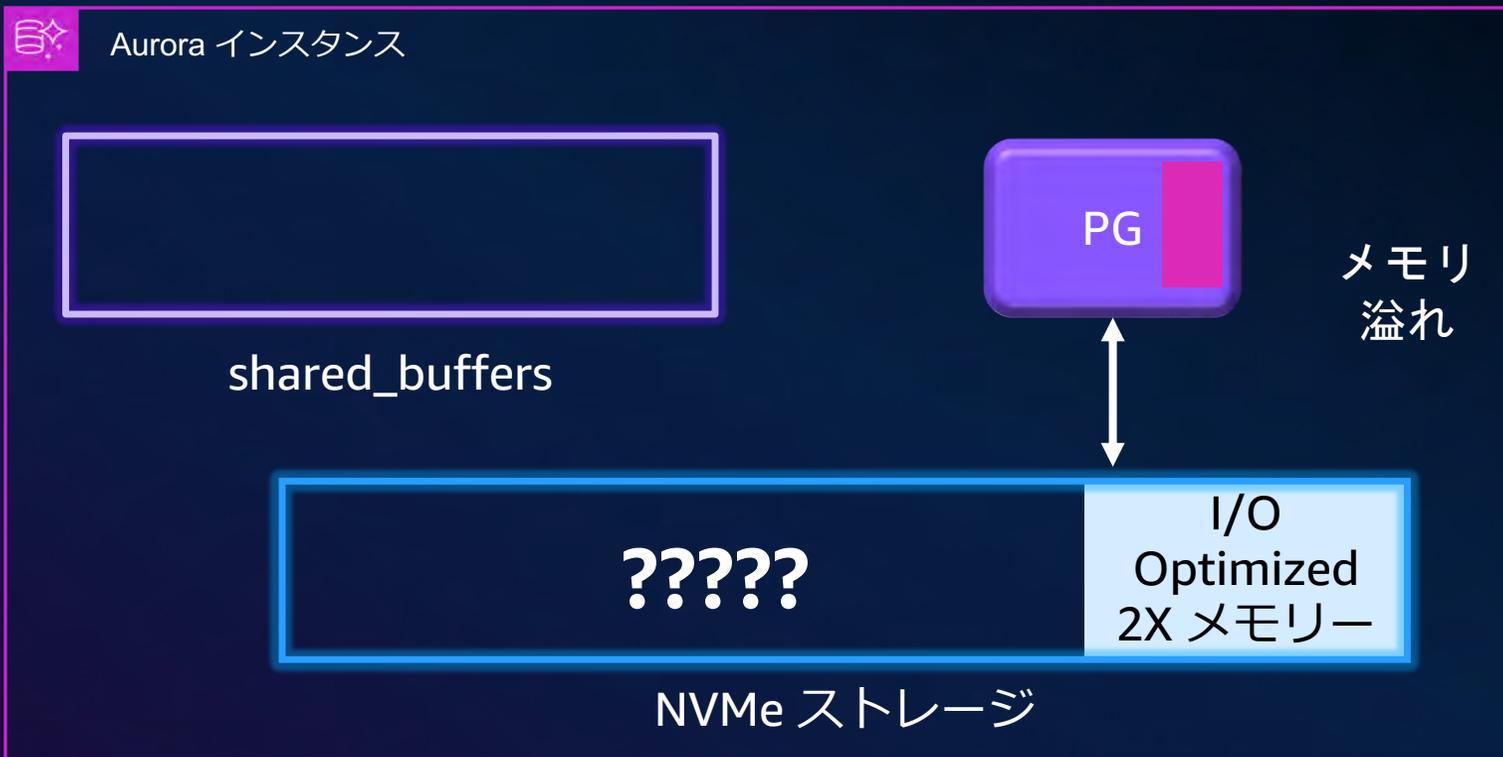
**対象**

db.r6gd  
db.r6id

Aurora  
PostgreSQL  
14.9+, 15.3+

# Optimized Reads - 階層型キャッシュ

PostgreSQL



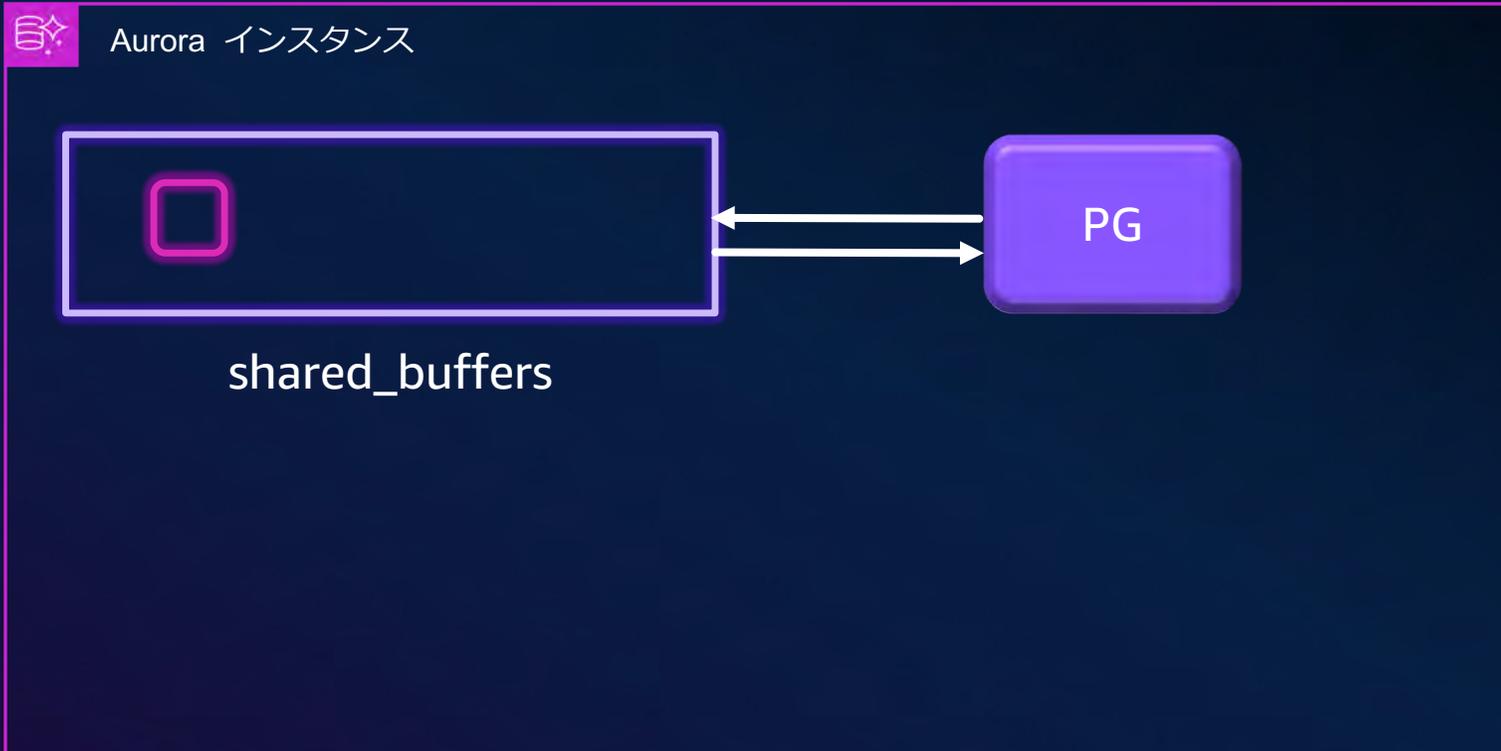
Amazon Elastic Block Store  
(Amazon EBS)  
2X メモリー

Aurora ストレージ

- 対象**
- db.r6gd
  - db.r6id
- Aurora  
PostgreSQL  
14.9+, 15.3+

# Optimized Reads - 階層型キャッシュ

PostgreSQL



I/O-Optimized  
利用時

対象

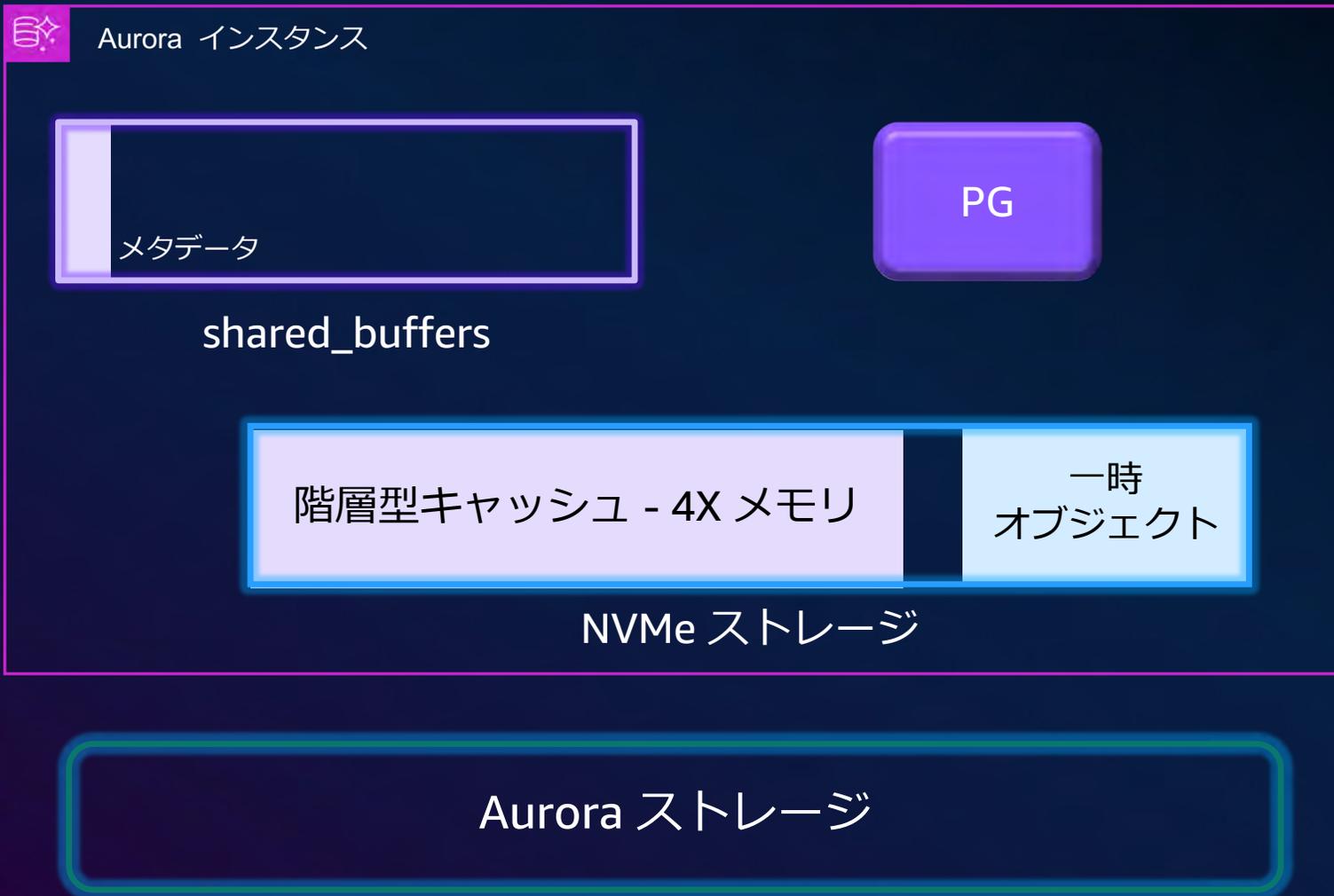
db.r6gd  
db.r6id

Aurora  
PostgreSQL  
14.9+, 15.3+



# Optimized Reads - 階層型キャッシュ

PostgreSQL



I/O-Optimized  
利用時

対象

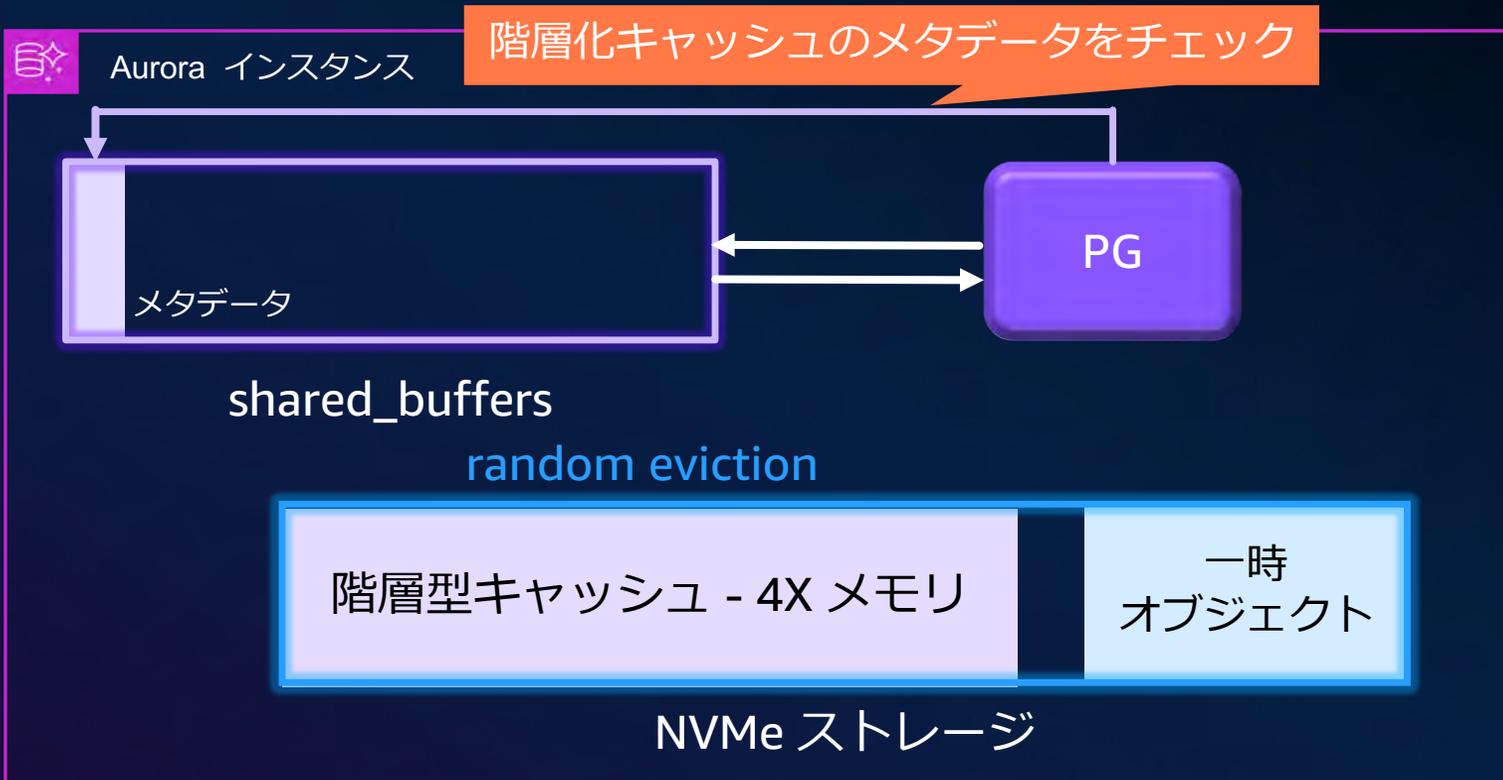
db.r6gd  
db.r6id

Aurora  
PostgreSQL  
14.9+, 15.3+



# Optimized Reads - 階層型キャッシュ

PostgreSQL



I/O-Optimized  
利用時

対象

db.r6gd  
db.r6id

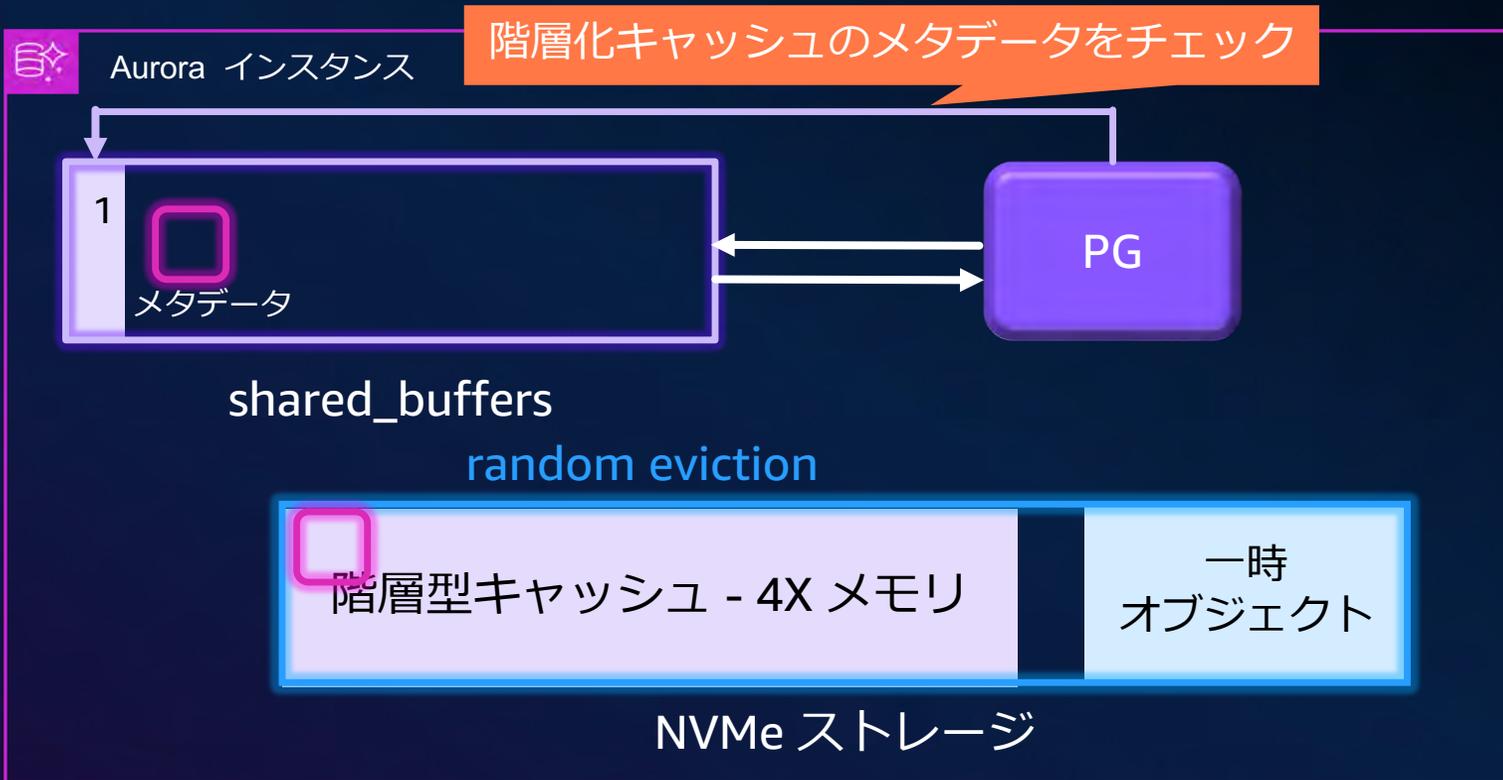
Aurora  
PostgreSQL  
14.9+, 15.3+

 Aurora ストレージ



# Optimized Reads - 階層型キャッシュ

PostgreSQL



I/O-Optimized  
利用時

**対象**

db.r6gd  
db.r6id

Aurora  
PostgreSQL  
14.9+, 15.3+

Aurora ストレージ



# pgvector 拡張機能

# pgvector とは

オープンソースの拡張機能で、

距離検索を提供しストレージ、インデックス作成、検索、メタデータをサポート

ベクトルデータ型

完全最近傍 (KNN)  
近似最近傍(ANN)

距離演算子(<->, <=>, <#>)

IVFFLAT / HNSW インデックスをサポート

埋め込みと同じ場所に配置

## Generative AI

Aurora PostgreSQL 15.3+ ,14.8+, 13.11+, 12.15+でサポート

<https://github.com/pgvector/pgvector>

# pgvector の使用例

```
CREATE TABLE chats (  
  id int GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  
  content text  
  embedding vector(1536));  
-- load data – e.g. 500K rows
```

テーブル作成

```
CREATE INDEX ON chats USING ivfflat (embeddings vector_cosine_ops)  
WITH (lists = 500);
```

インデックス作成

```
SELECT id, content  
FROM chats  
ORDER BY embeddings <=> $1 LIMIT 5
```

検索

<https://aws.amazon.com/blogs/database/building-ai-powered-search-in-postgresql-using-amazon-sagemaker-and-pgvector/>

# pgvector の使用例

PostgreSQL

```
CREATE TABLE chats (  
  id int GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  
  content text  
  embedding vector(1536));  
-- load data – e.g. 500K rows
```

テーブル作成

```
CREATE INDEX ON chats USING ivfflat (embeddings vector_cosine_ops)  
WITH (lists = 500);
```

インデックス作成

```
SELECT id, content  
FROM chats  
ORDER BY embeddings <=> $1 LIMIT 5
```

検索

<https://aws.amazon.com/blogs/database/building-ai-powered-search-in-postgresql-using-amazon-sagemaker-and-pgvector/>



# pgvector の使用例

```
CREATE TABLE chats (  
  id int GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,  
  content text  
  embedding vector(1536));  
-- load data – e.g. 500K rows
```

テーブル作成

```
CREATE INDEX ON chats USING ivfflat (embeddings vector_cosine_ops)  
WITH (lists = 500);
```

インデックス作成

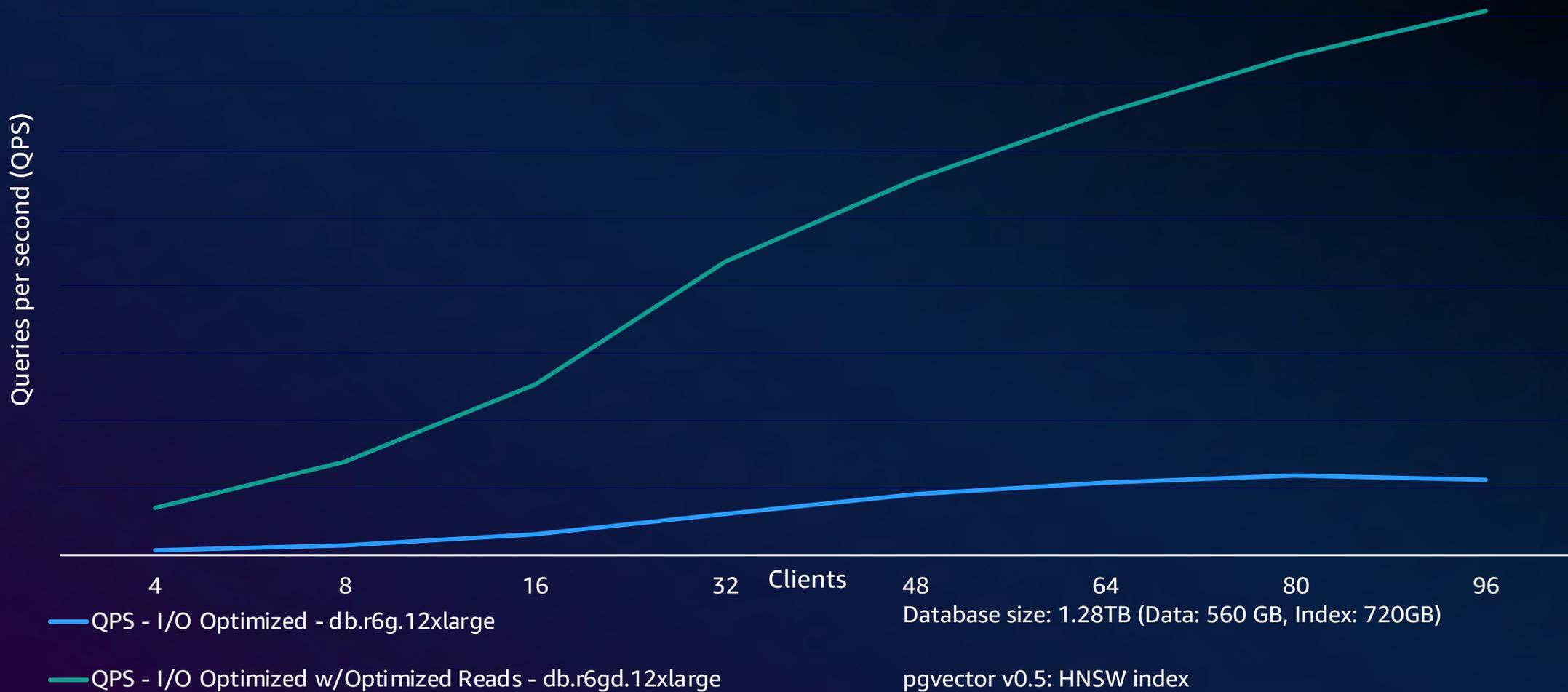
```
SELECT id, content  
FROM chats  
ORDER BY embeddings <=> $1 LIMIT 5
```

検索

<https://aws.amazon.com/blogs/database/building-ai-powered-search-in-postgresql-using-amazon-sagemaker-and-pgvector/>

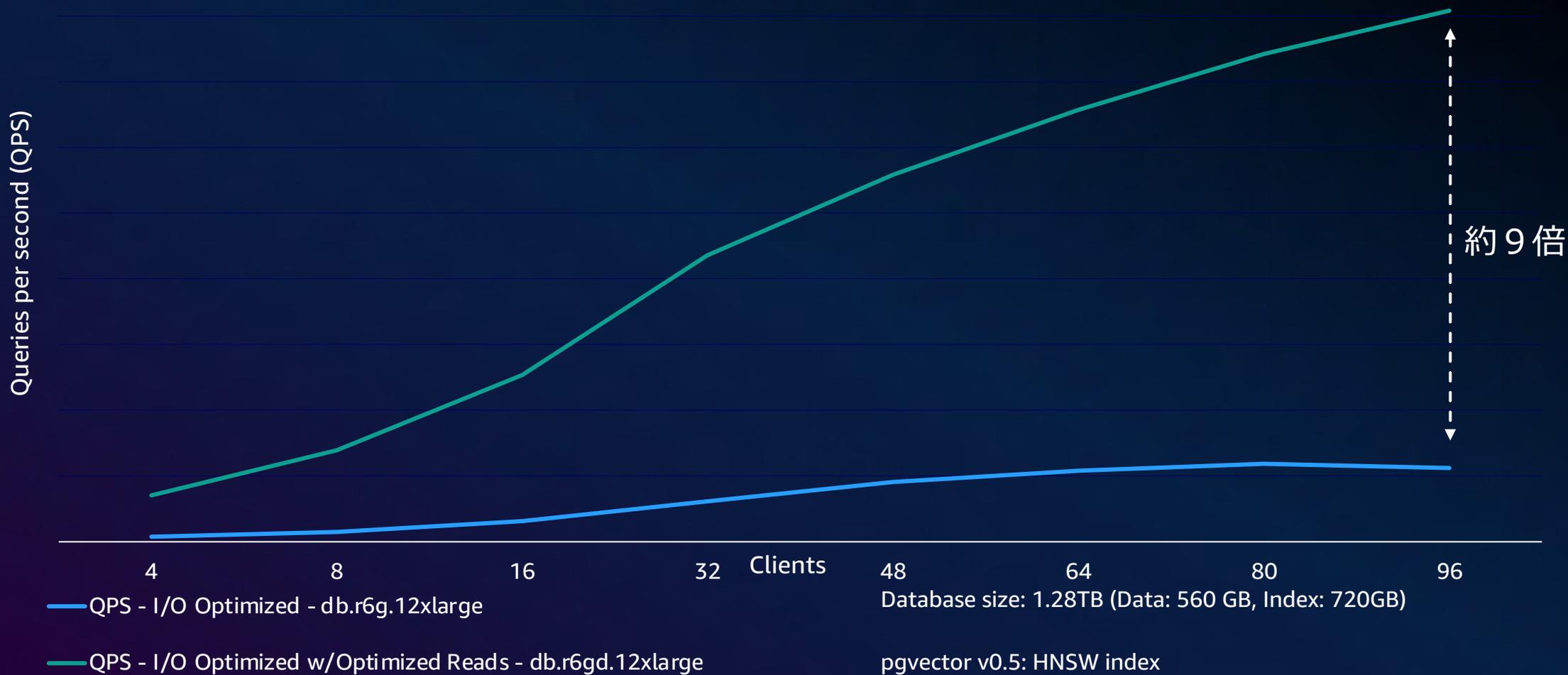
# 階層型キャッシュによるベクトルのパフォーマンス

10億のベクトルによるBigANNベンチマークと0.9578のリコール



# 階層型キャッシュによるベクトルのパフォーマンス

10億のベクトルによるBigANNベンチマークと0.9578のリコール



# Thank you!

**Mari Tsukamoto**

tsukamar@amazon.co.jp

