JAPAN | 2024

aws SUMMIT

AWS-39

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

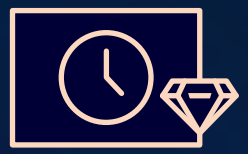
- Amazon Aurora の基本
  - アーキテクチャ
  - グローバルデータベース
- 管理性の向上に役立つ機能
  - Aurora Serverless
  - Managed Blue/Green Deployment
  - Amazon Redshift とのAmazon Aurora ゼロETL統合
- 新しいストレージタイプとOptimized Read
- 生成AIに活用できる機能

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

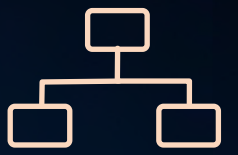


可用性と耐久性

- 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

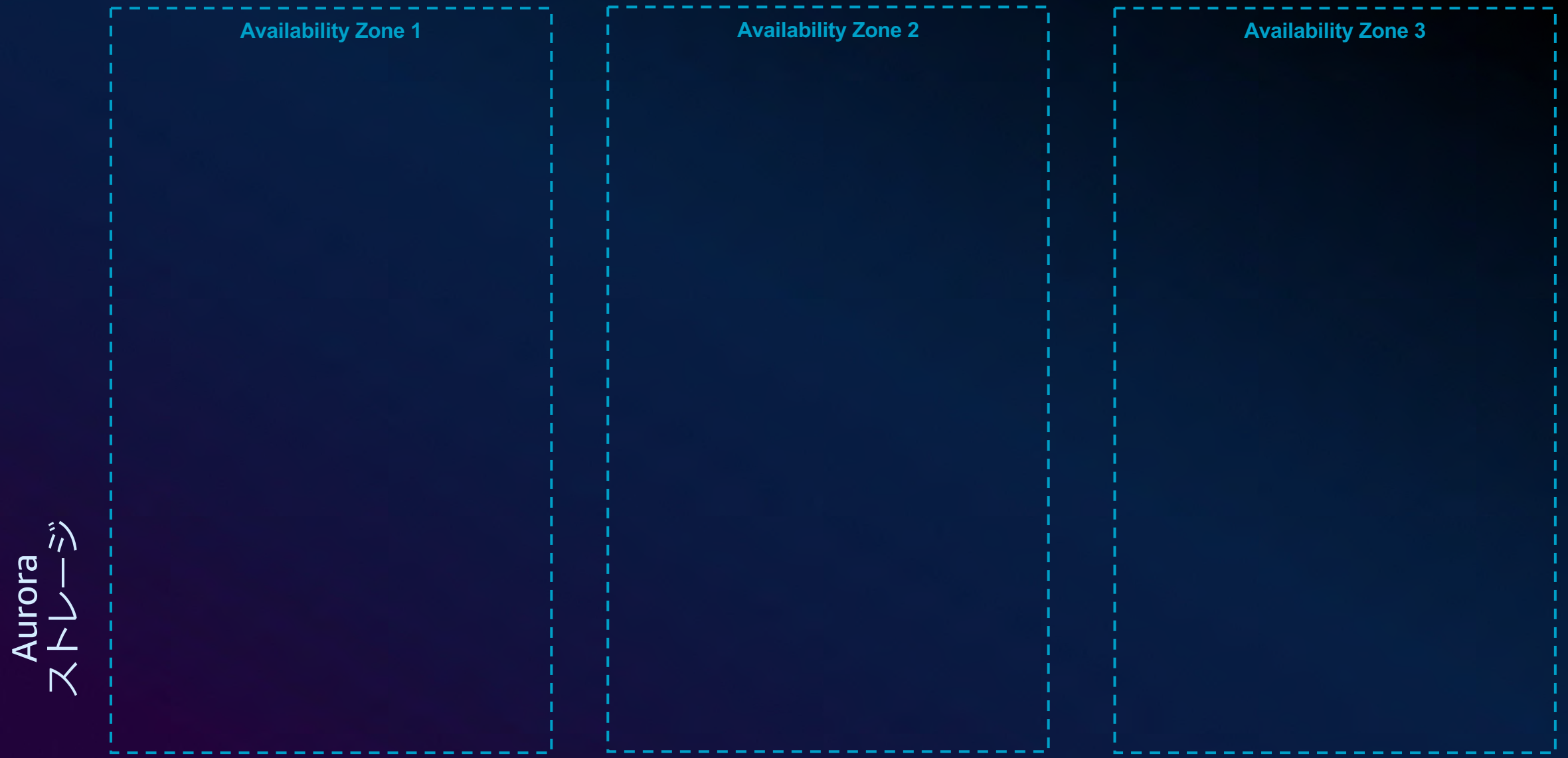


フルマネージド

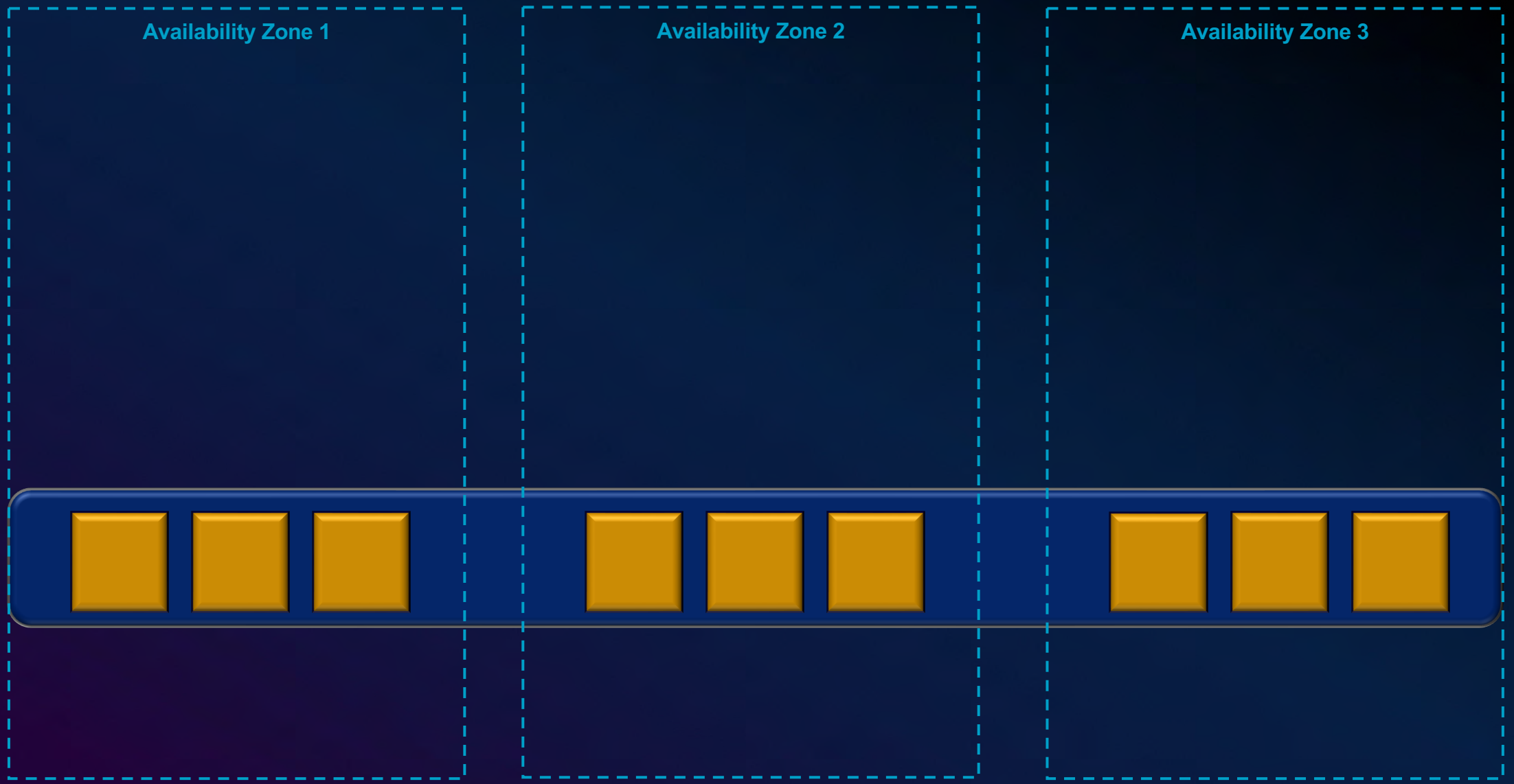
- ハードウェアのプロビジョニング、データベースのセットアップ、パッチ適用、バックアップなどの時間のかかる管理タスクの管理を自動化します。
- サーバーレス構成オプション

アーキテクチャ

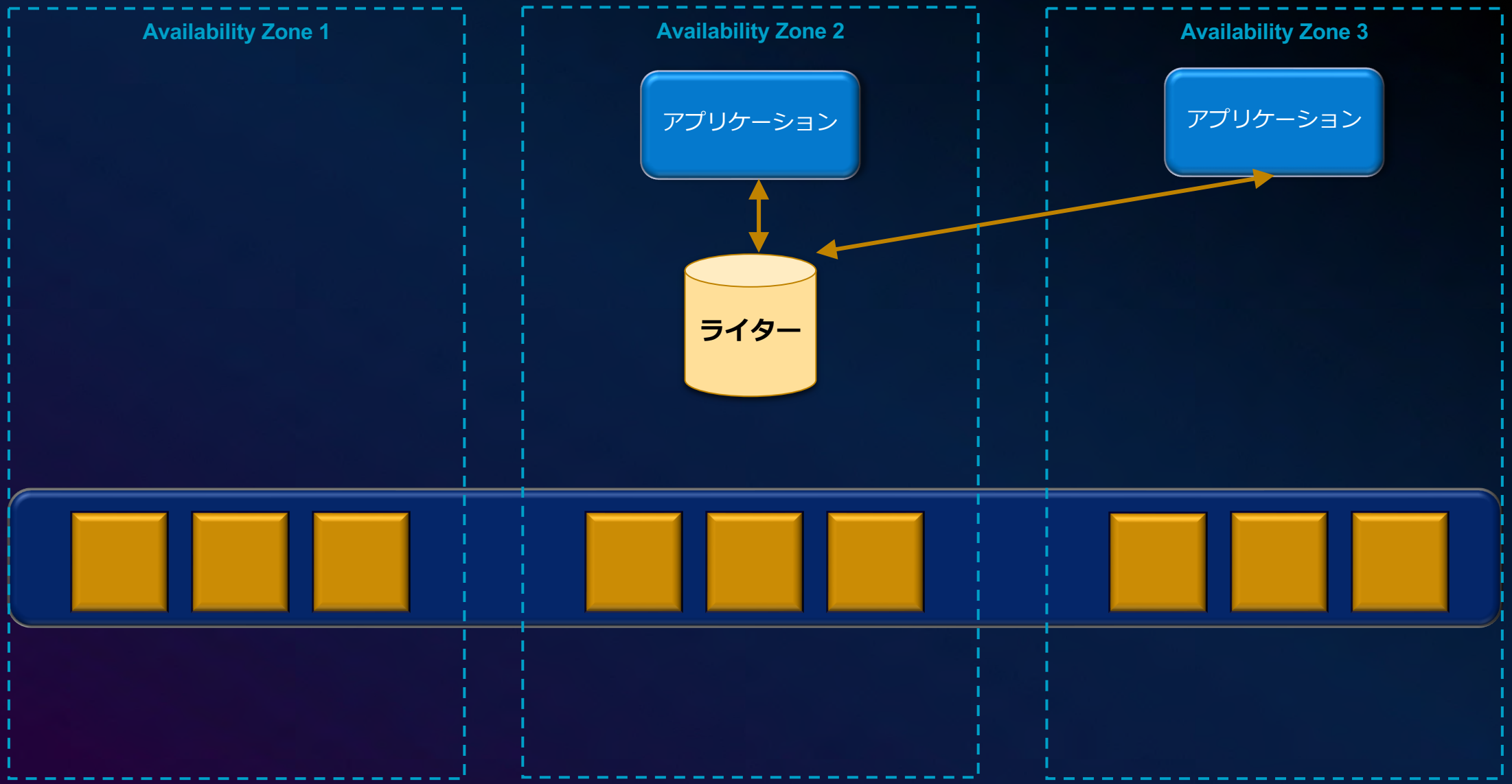
Aurora ストレージとレプリカ - 可用性と耐久性



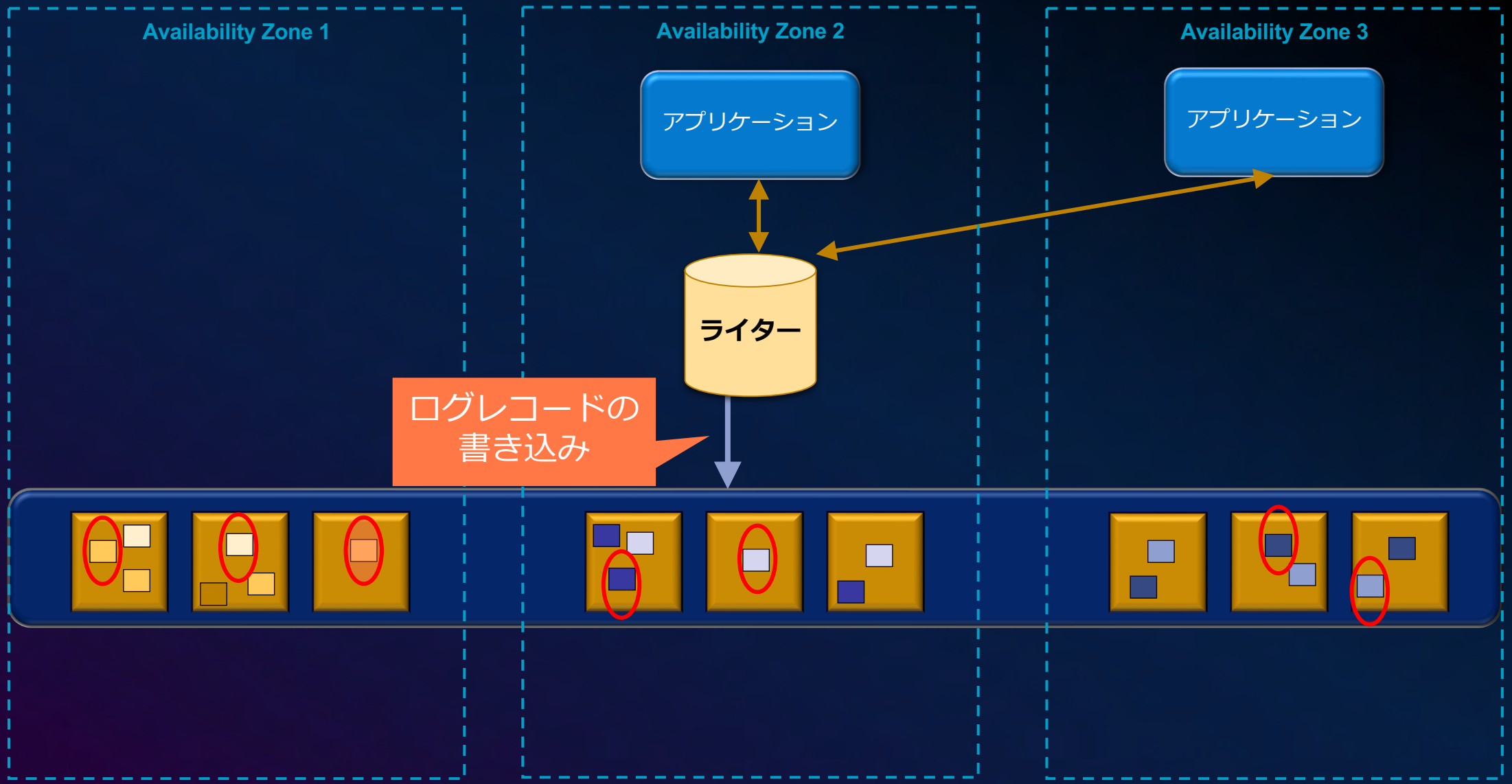
Aurora ストレージとレプリカ - 可用性と耐久性



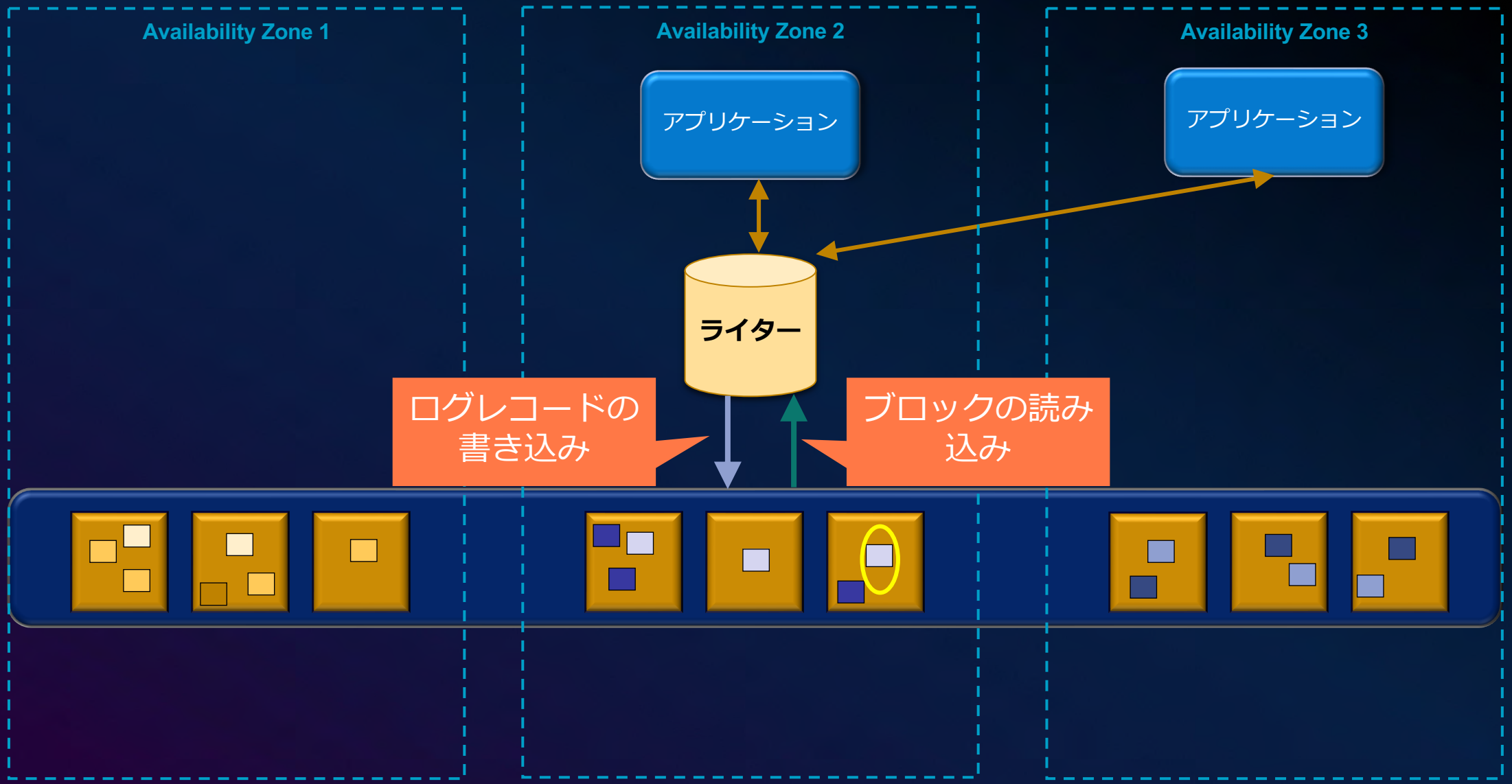
Aurora ストレージとレプリカ - 可用性と耐久性



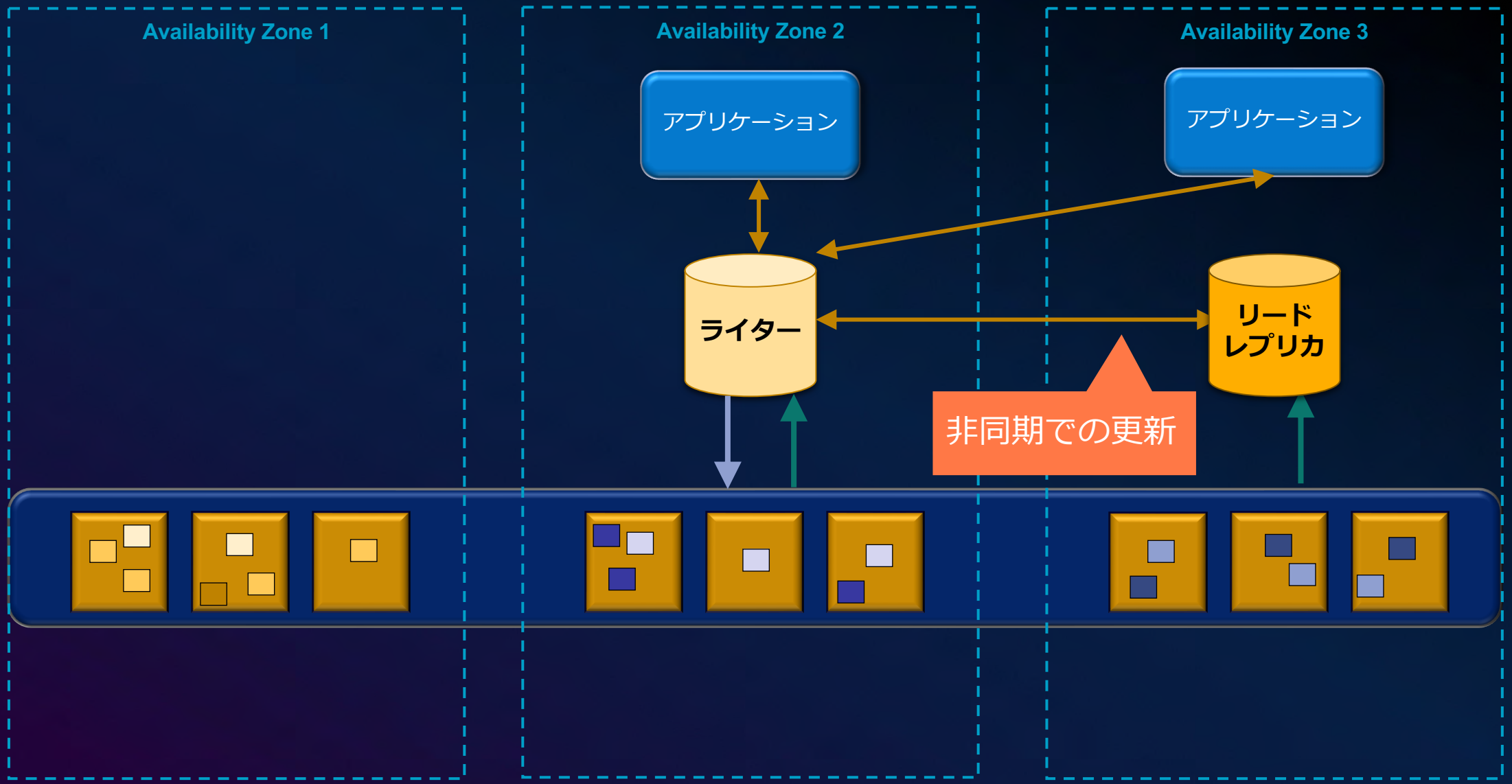
Aurora ストレージとレプリカ - 可用性と耐久性



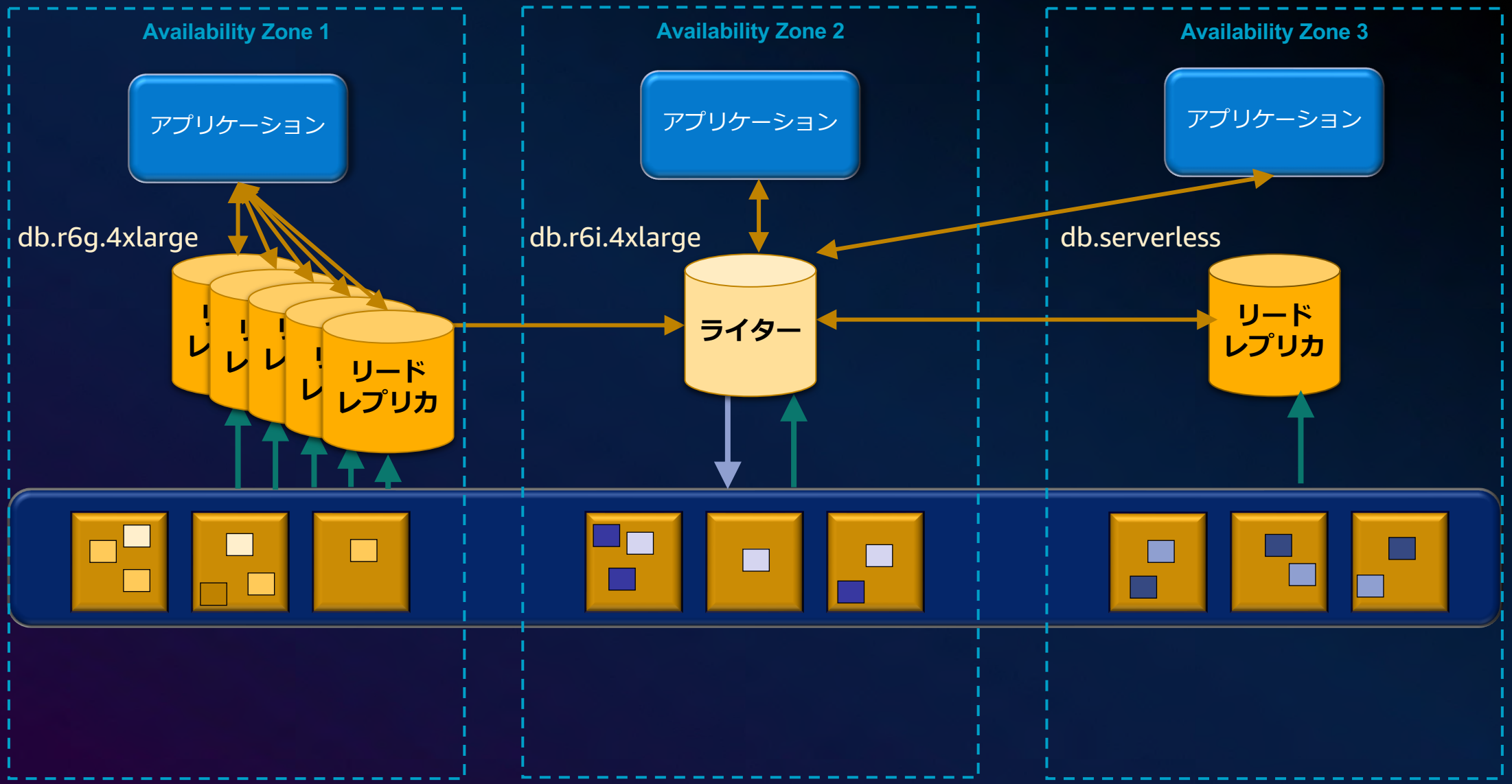
Aurora ストレージとレプリカ - 可用性と耐久性



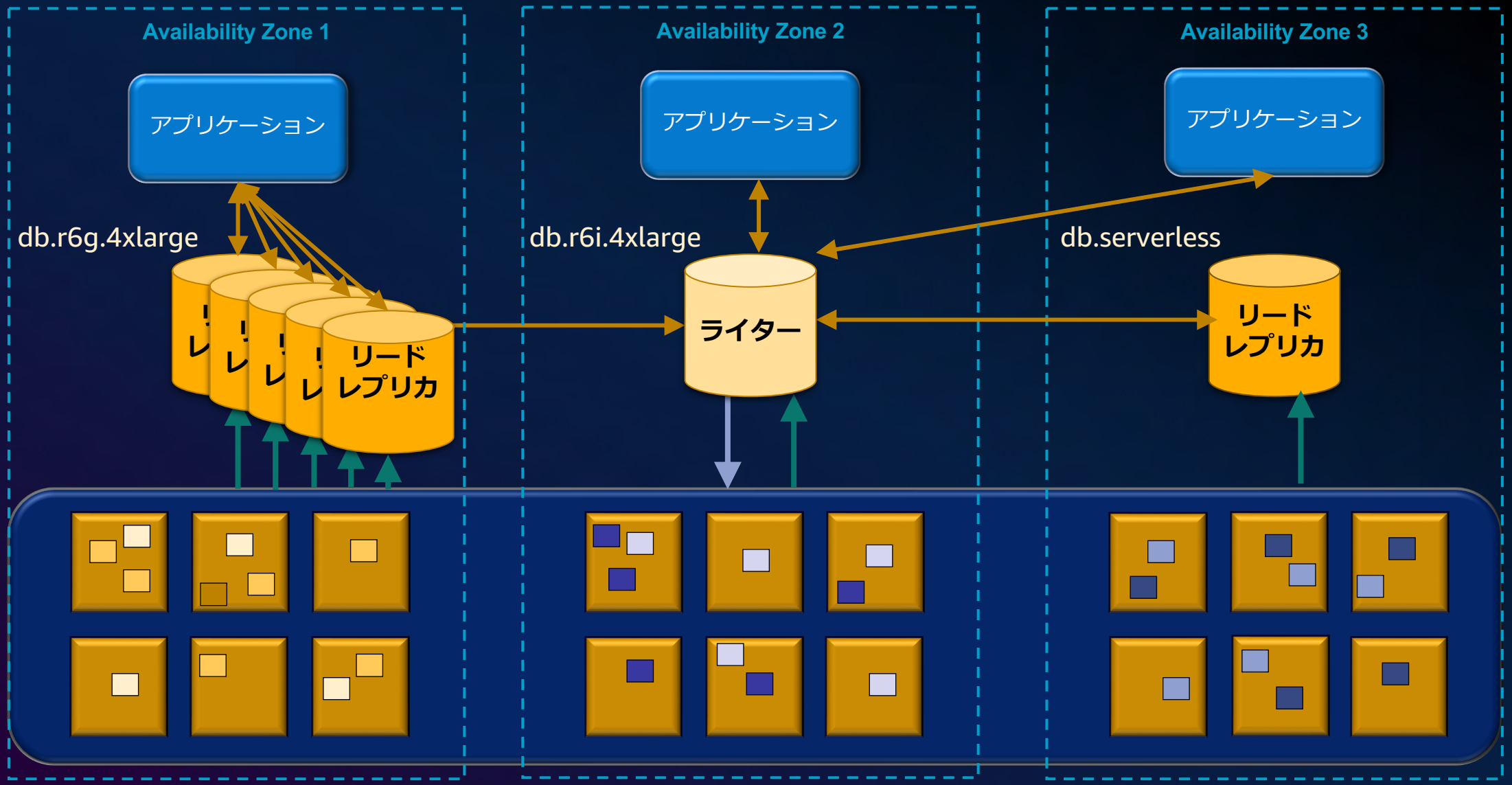
Aurora ストレージとレプリカ - 拡張性とパフォーマンス



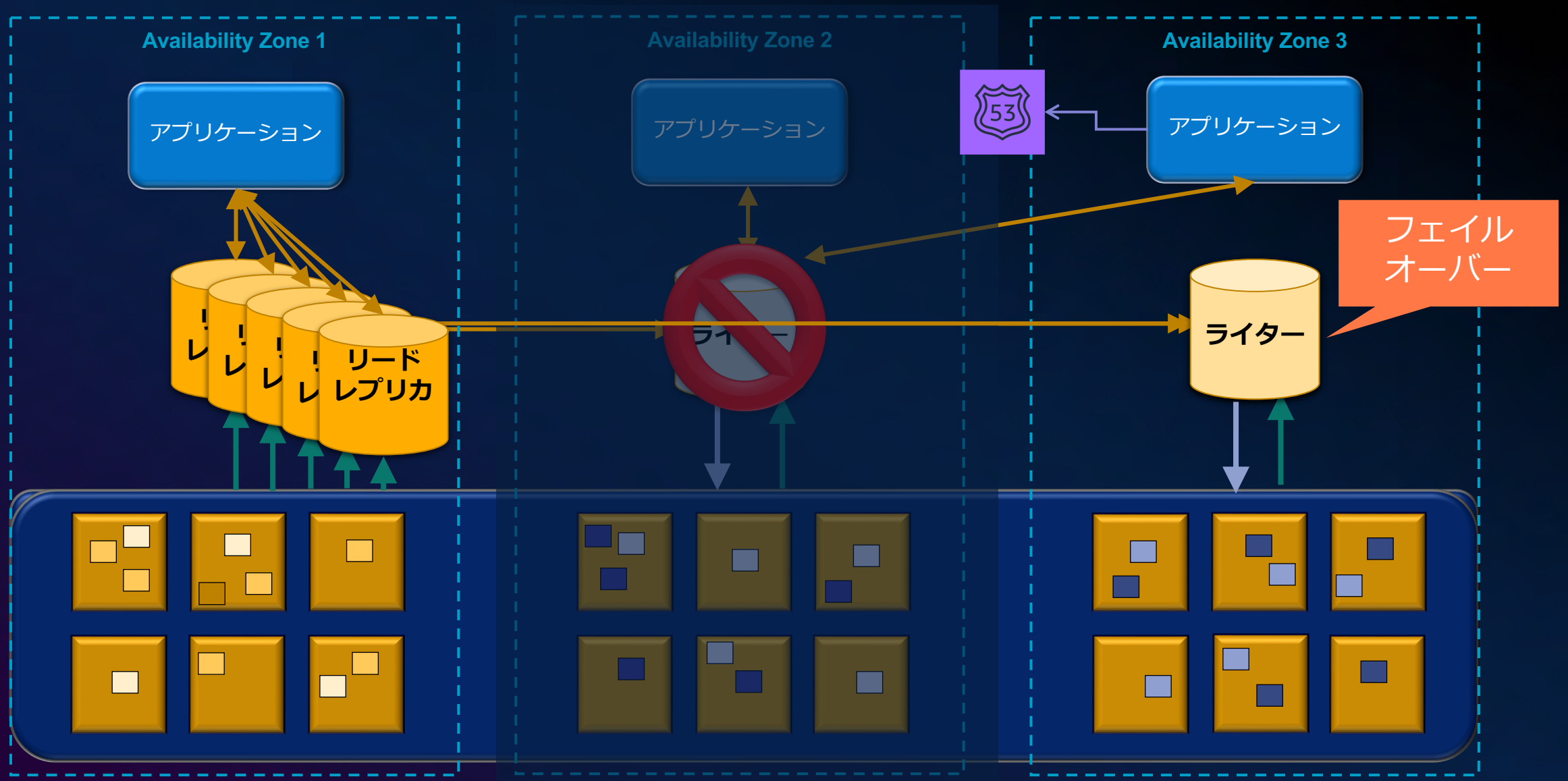
Aurora ストレージとレプリカ - 拡張性とパフォーマンス



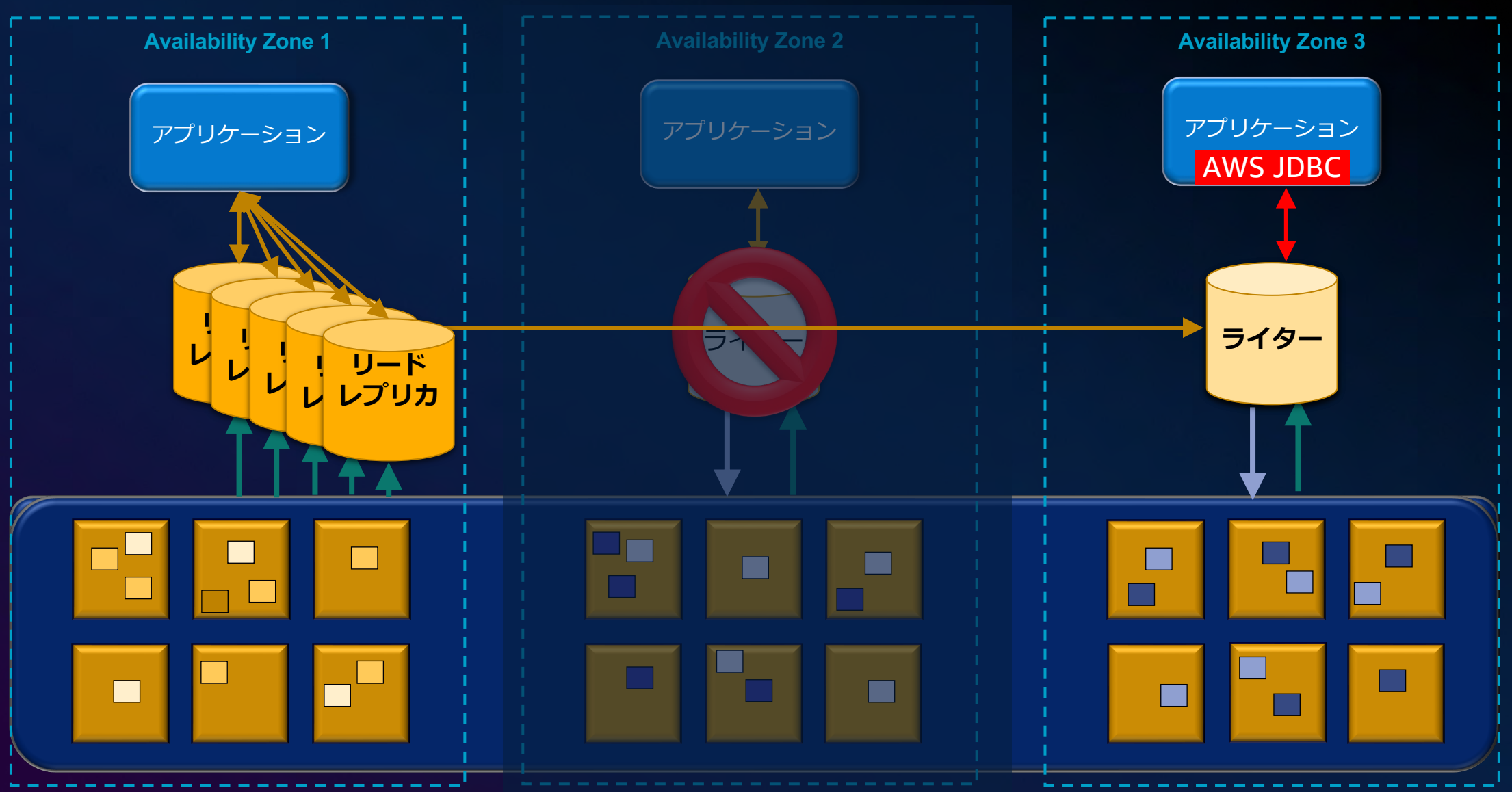
Aurora ストレージとレプリカ- 拡張性とパフォーマンス



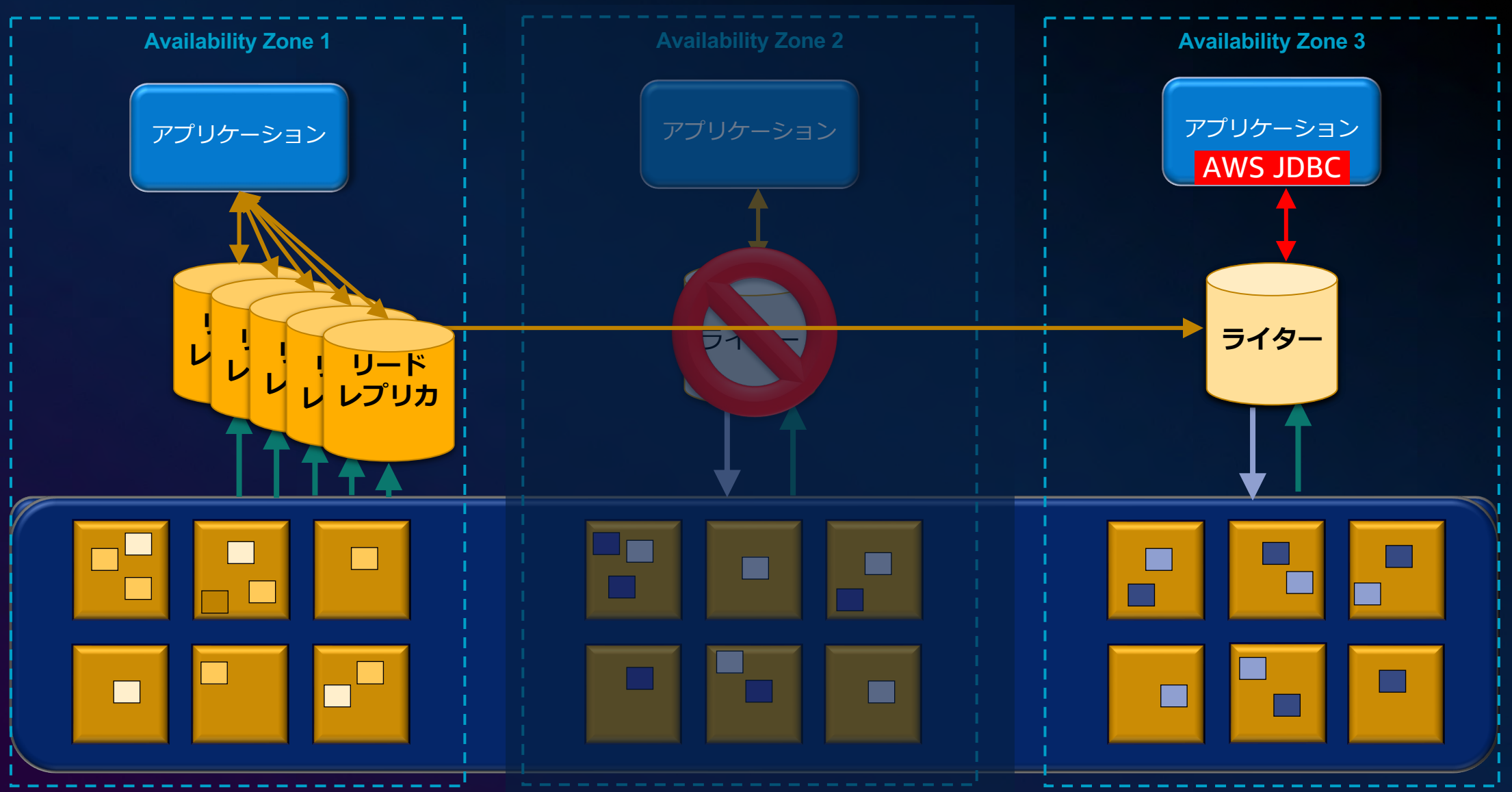
Aurora ストレージとレプリカ - フェイルオーバー



Aurora ストレージとレプリカ - フェイルオーバー

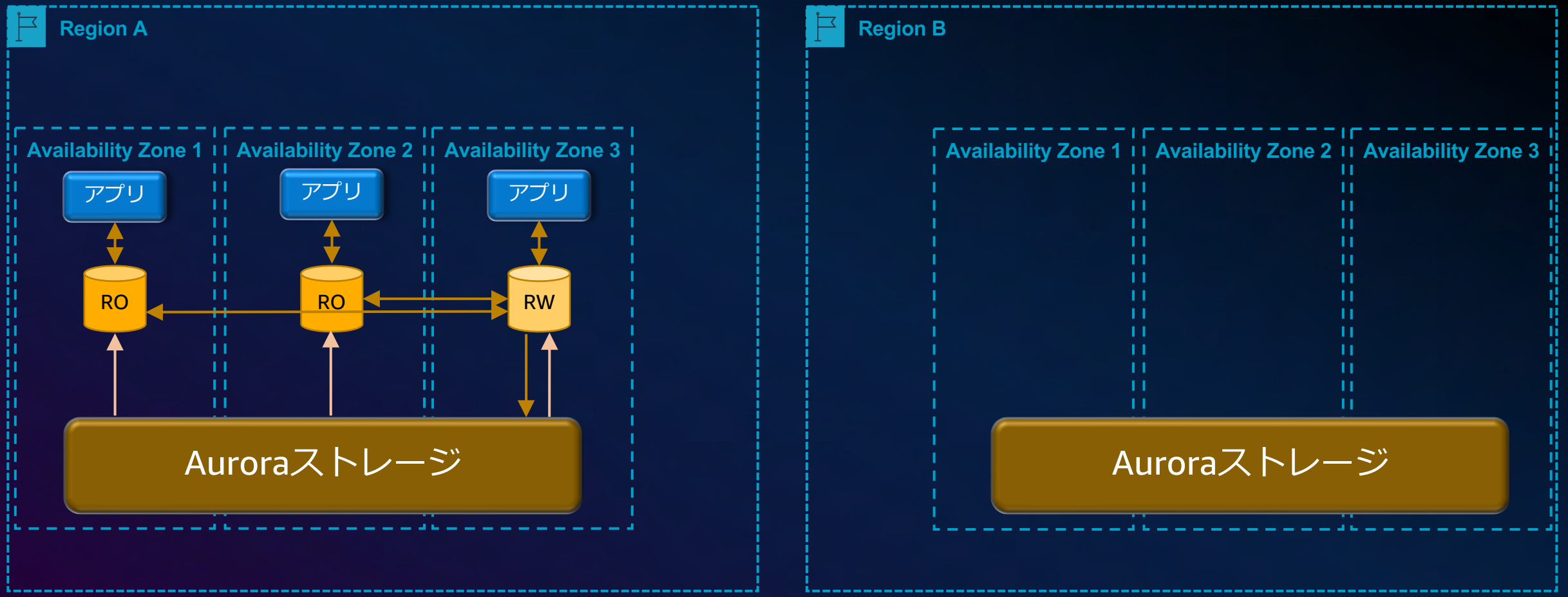


Aurora ストレージとレプリカ - フェイルオーバー



Amazon Aurora Global Database

RW：ライター
RO：リードレプリカ

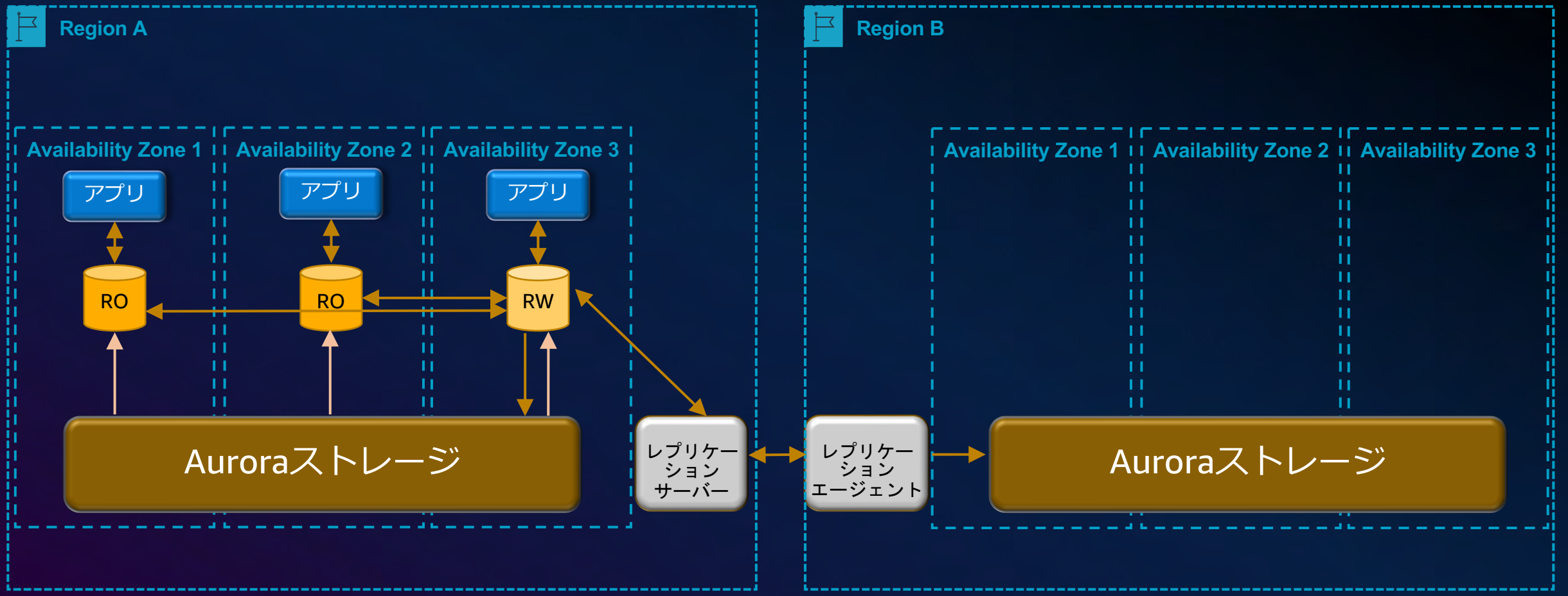


プライマリ DB クラスター

セカンダリ DB クラスター

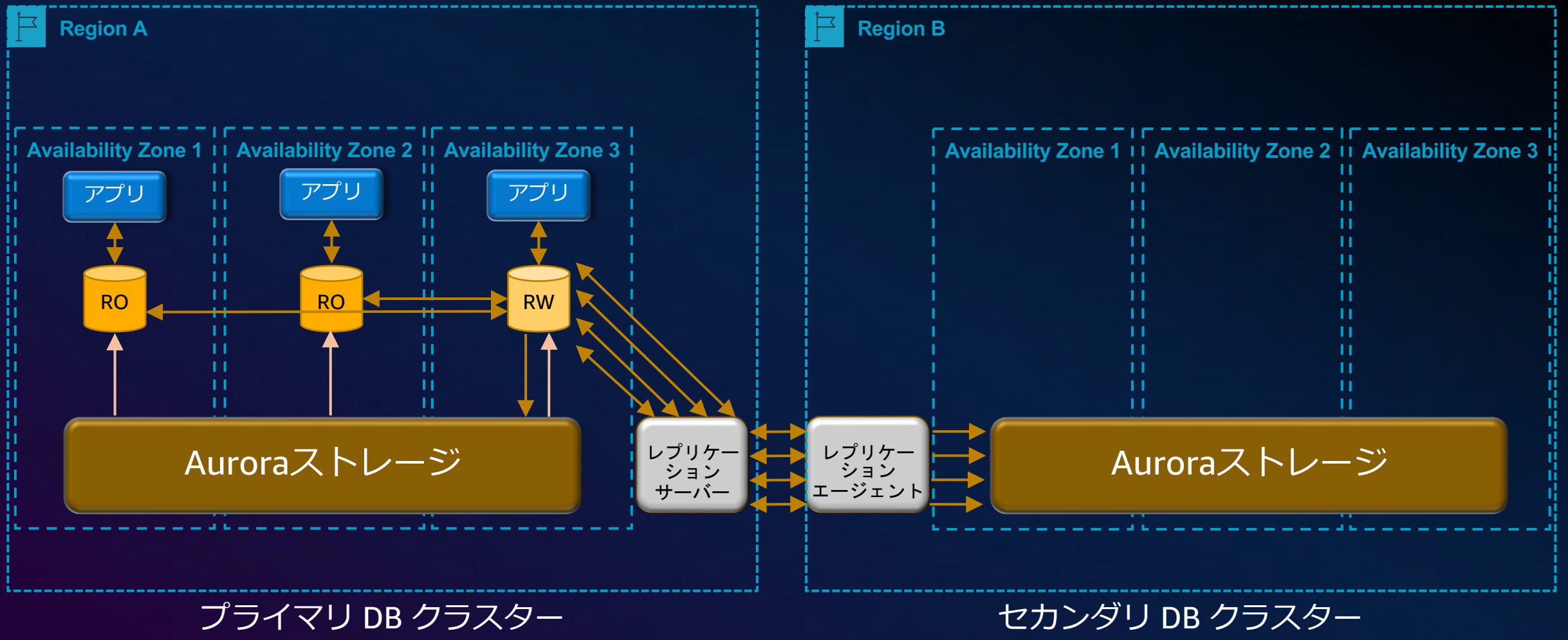
Amazon Aurora Global Database

RW：ライター
RO：リードレプリカ



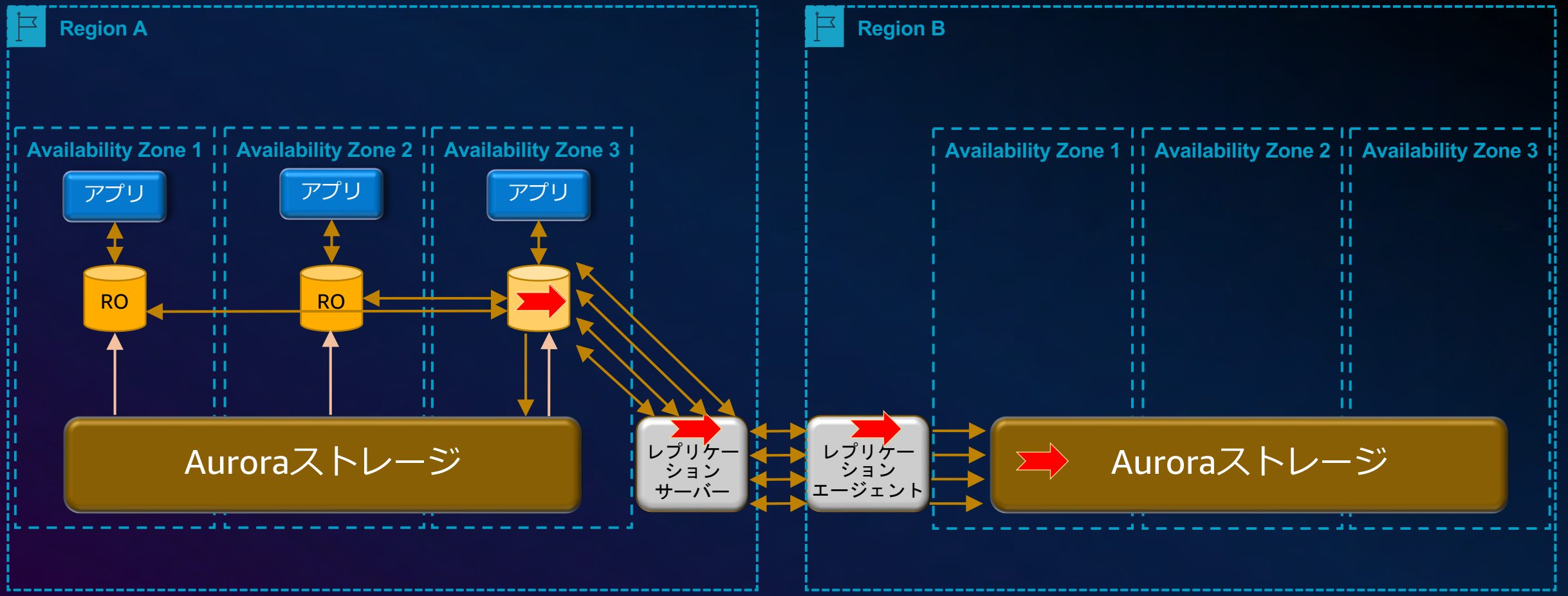
Amazon Aurora Global Database

RW：ライター
RO：リードレプリカ



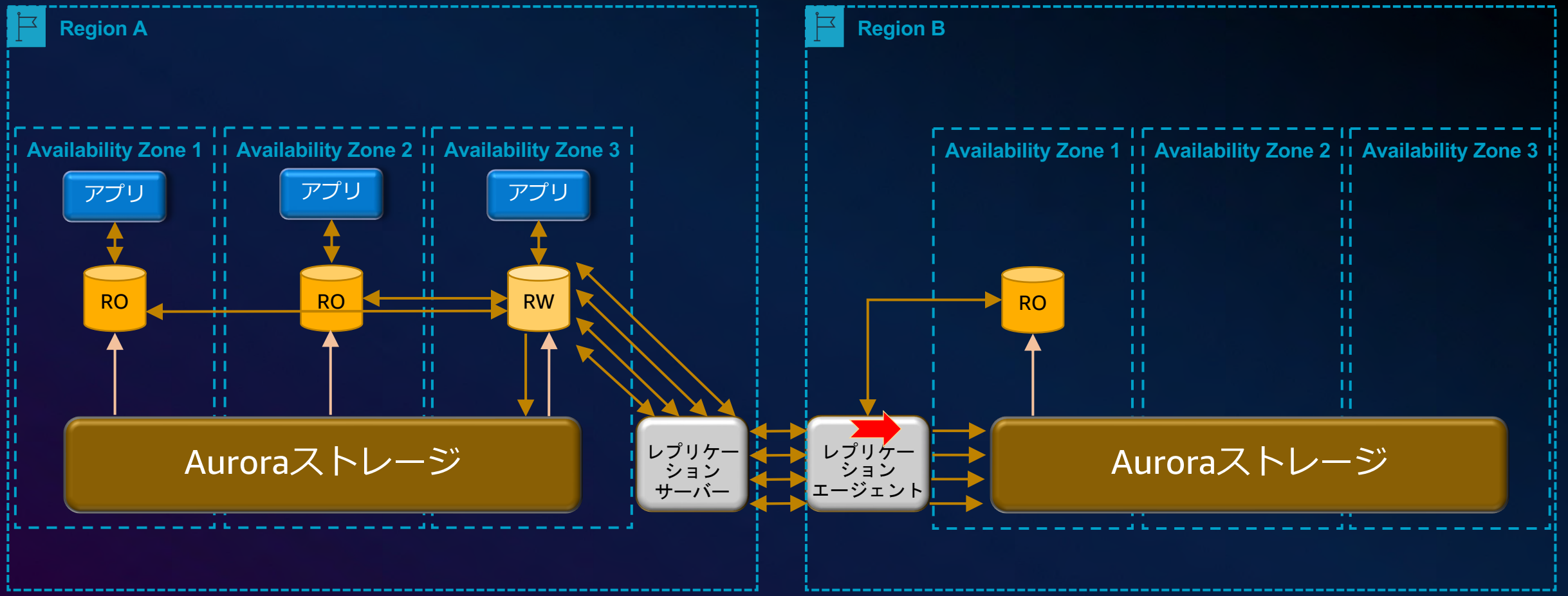
Amazon Aurora Global Database

RW：ライター
RO：リードレプリカ



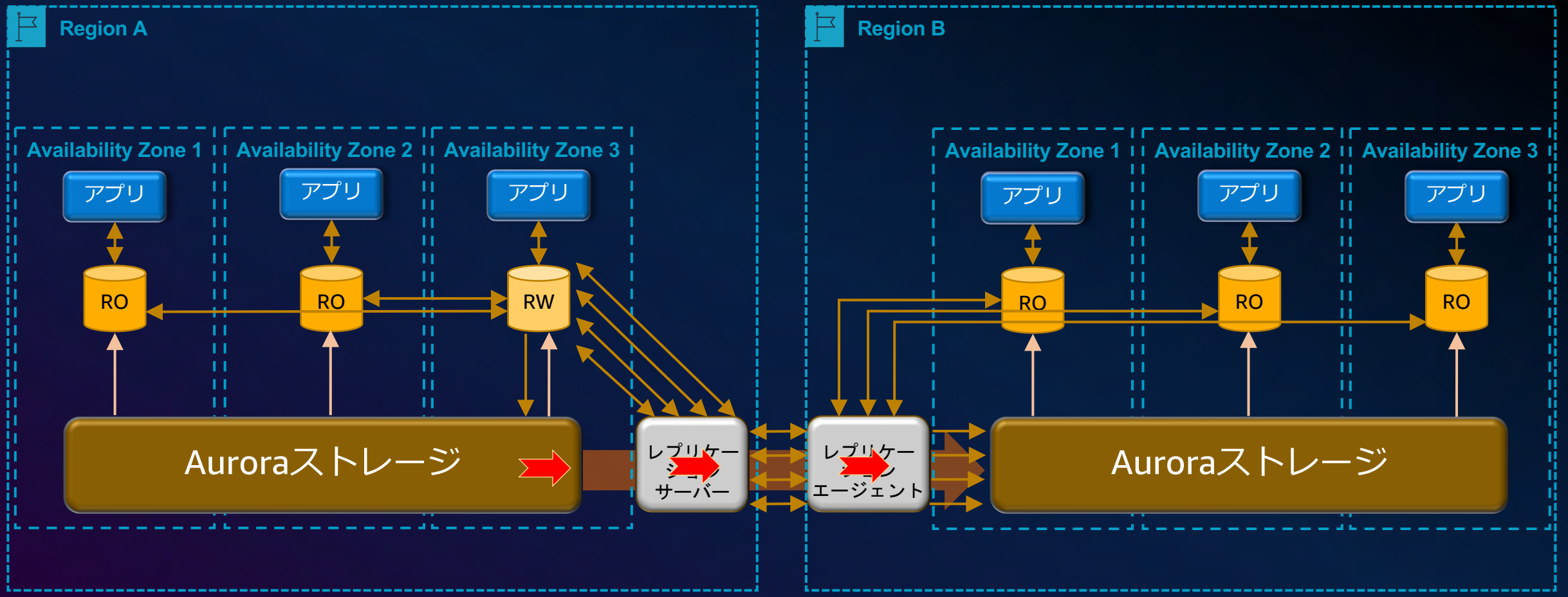
Amazon Aurora Global Database

RW：ライター
RO：リードレプリカ



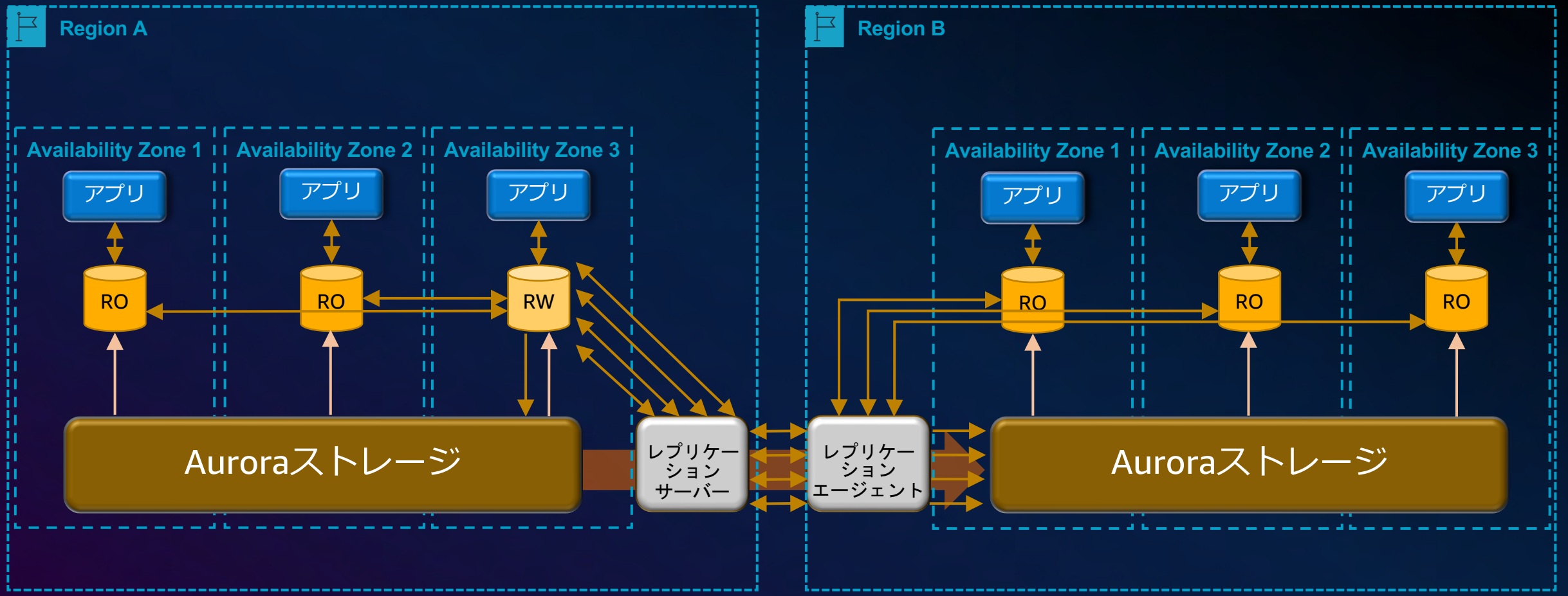
Amazon Aurora Global Database

RW：ライター
RO：リードレプリカ



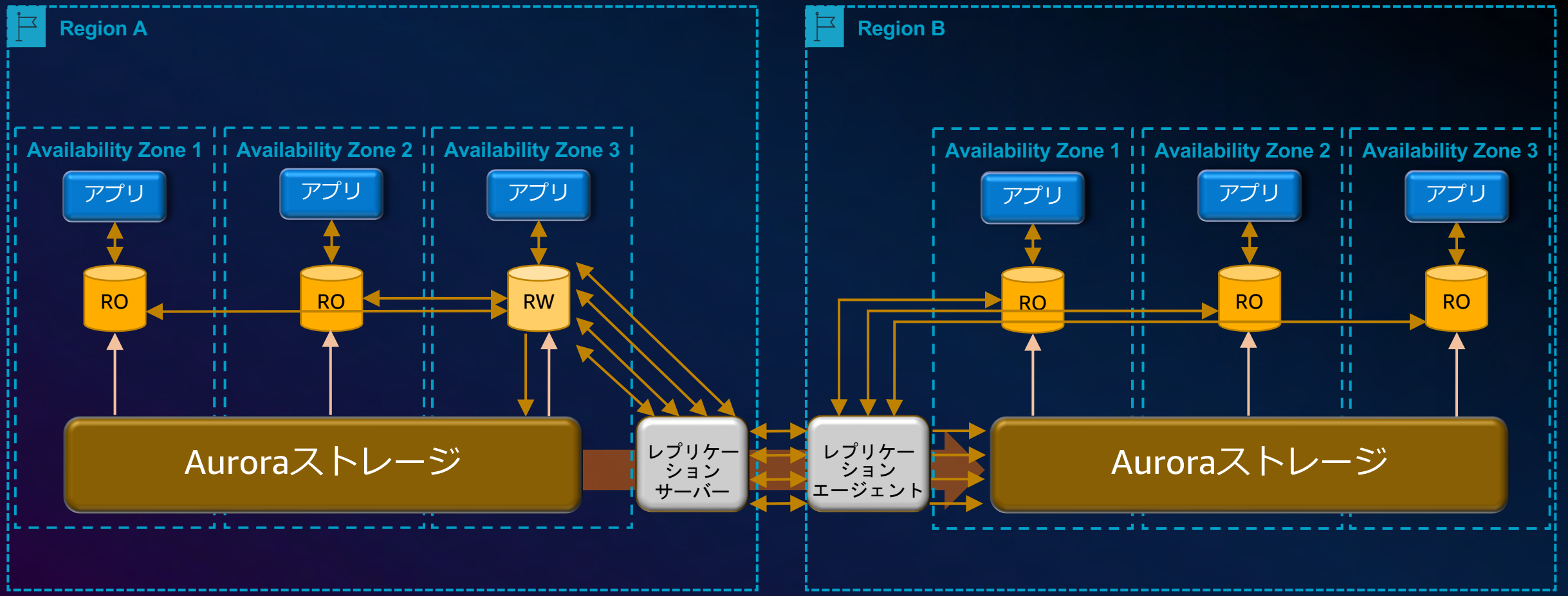
Aurora Global Database - Switchover/ Failover

RW：ライター
RO：リードレプリカ



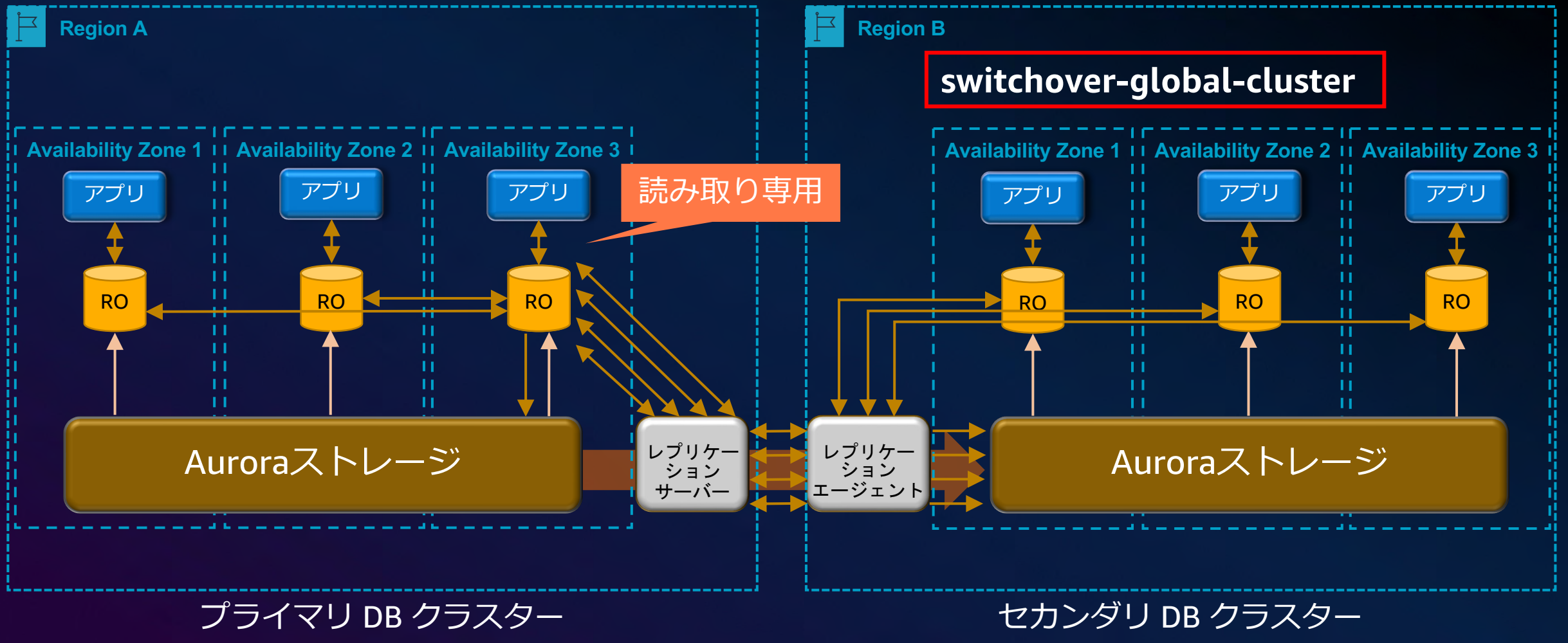
Aurora Global Database - Switchover

RW：ライター
RO：リードレプリカ



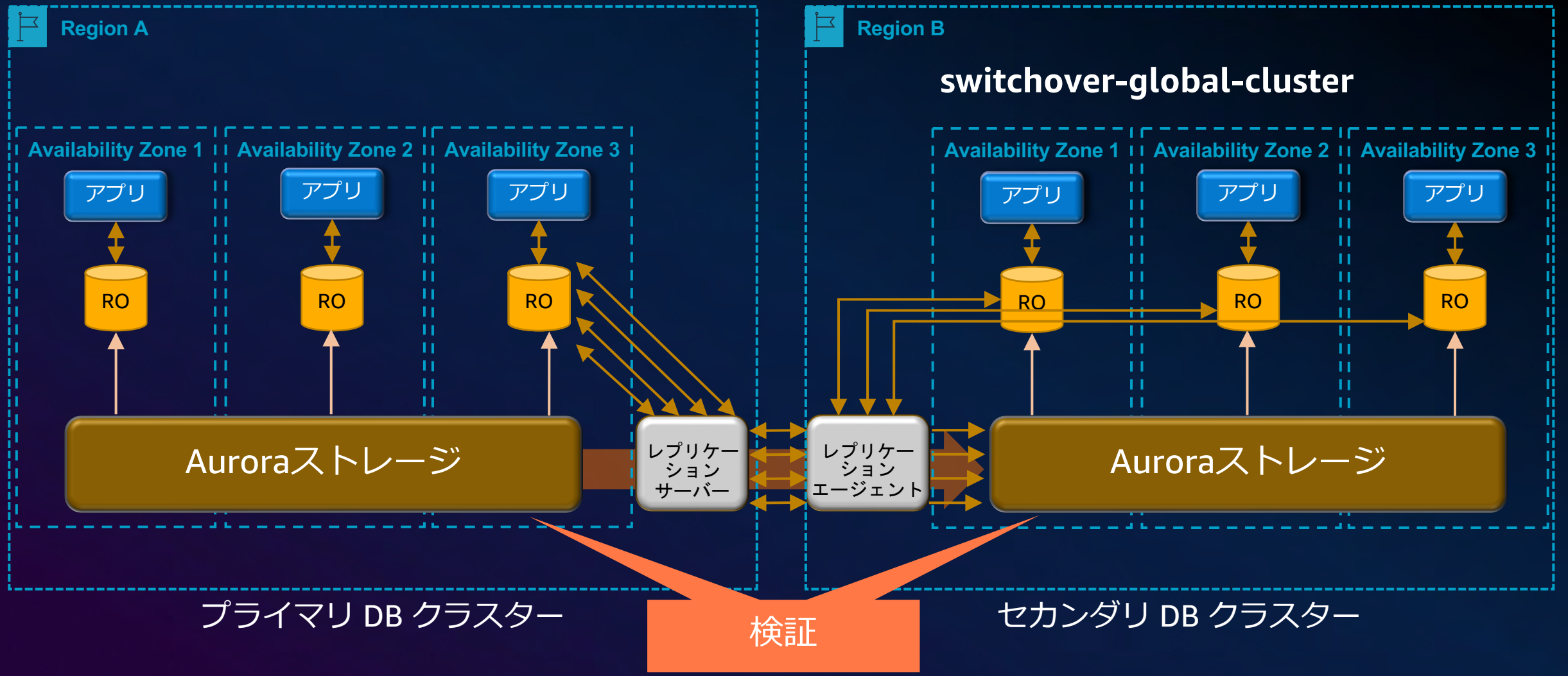
Aurora Global Database - Switchover

RW：ライター
RO：リードレプリカ



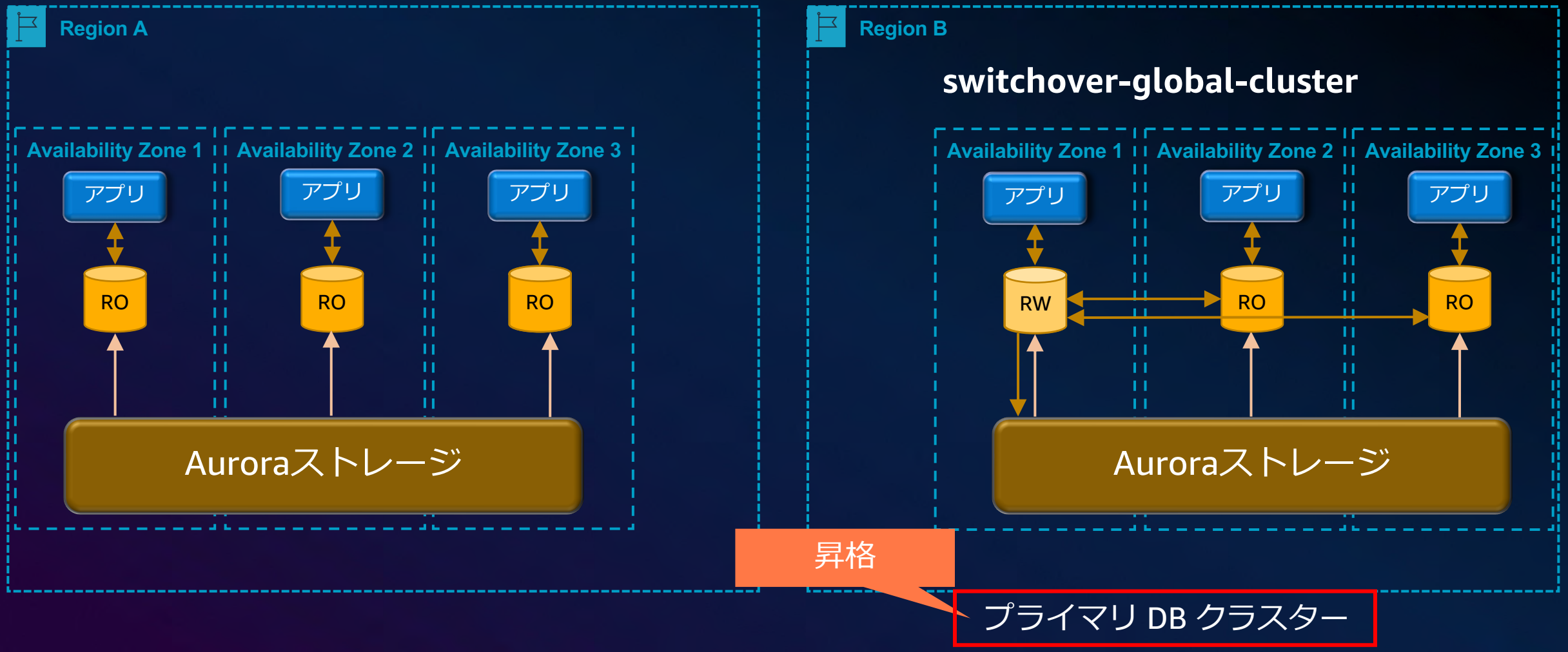
Aurora Global Database - Switchover

RW：ライター
RO：リードレプリカ



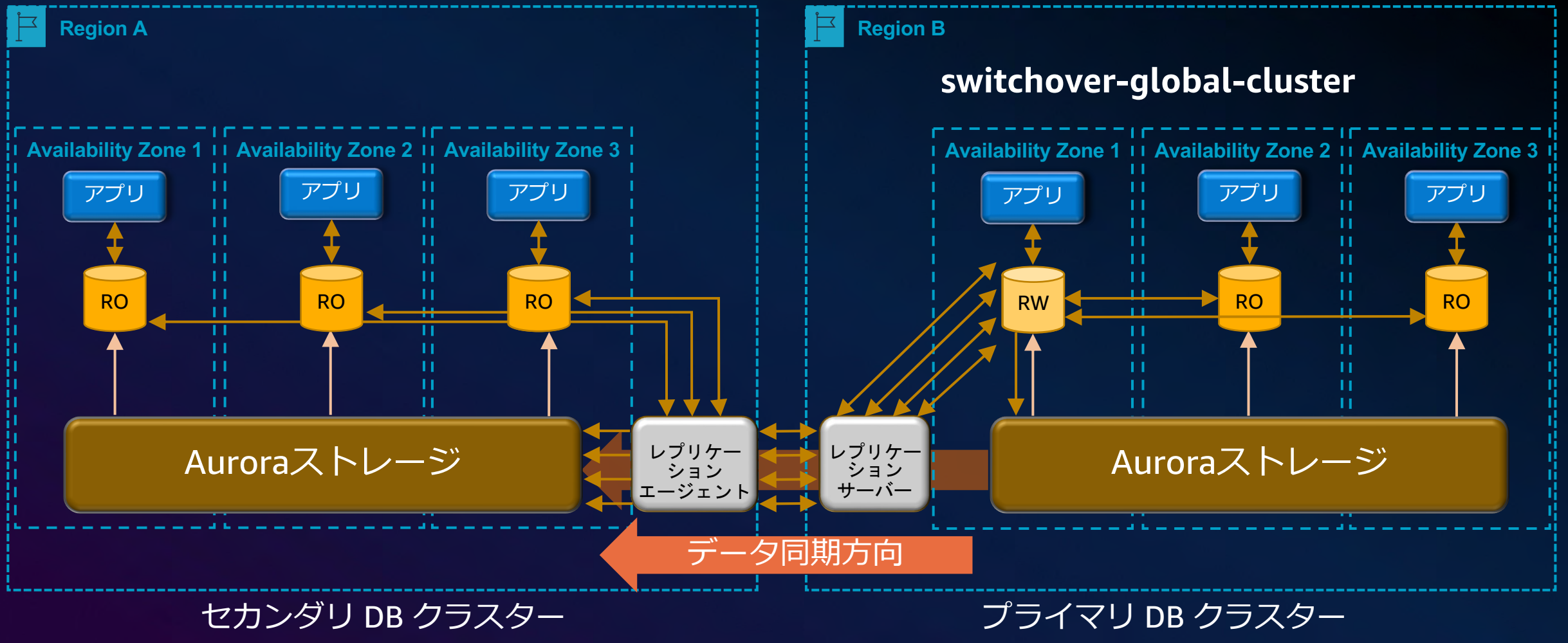
Aurora Global Database - Switchover

RW：ライター
RO：リードレプリカ



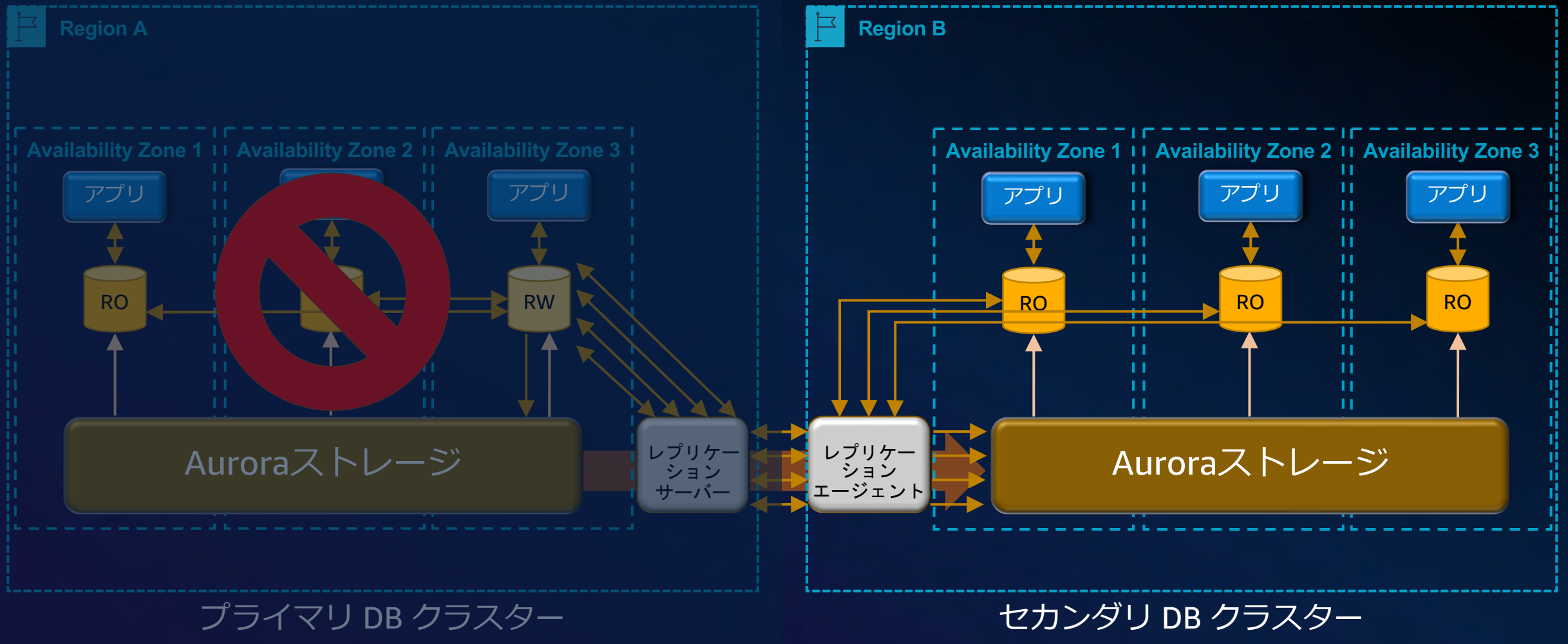
Aurora Global Database - Switchover

RW：ライター
RO：リードレプリカ



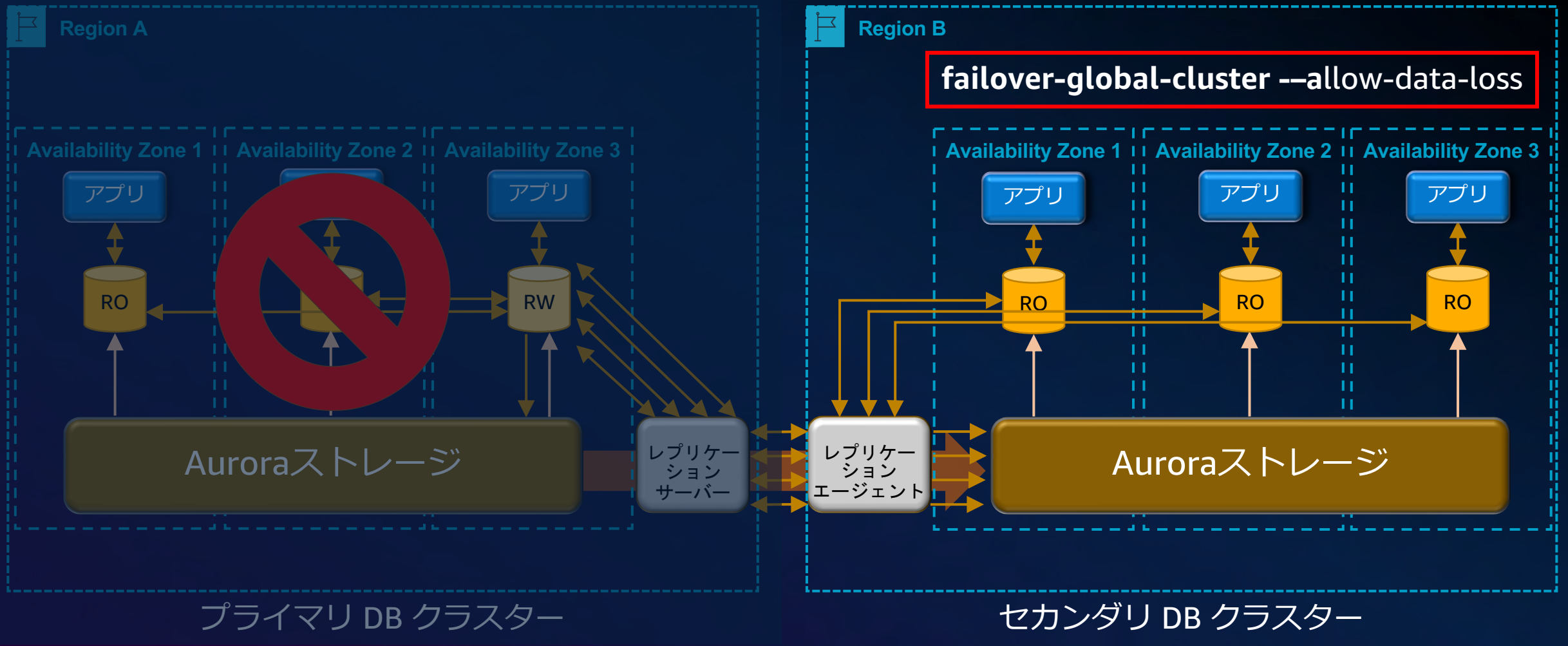
Aurora Global Database - Failover

RW：ライター
RO：リードレプリカ



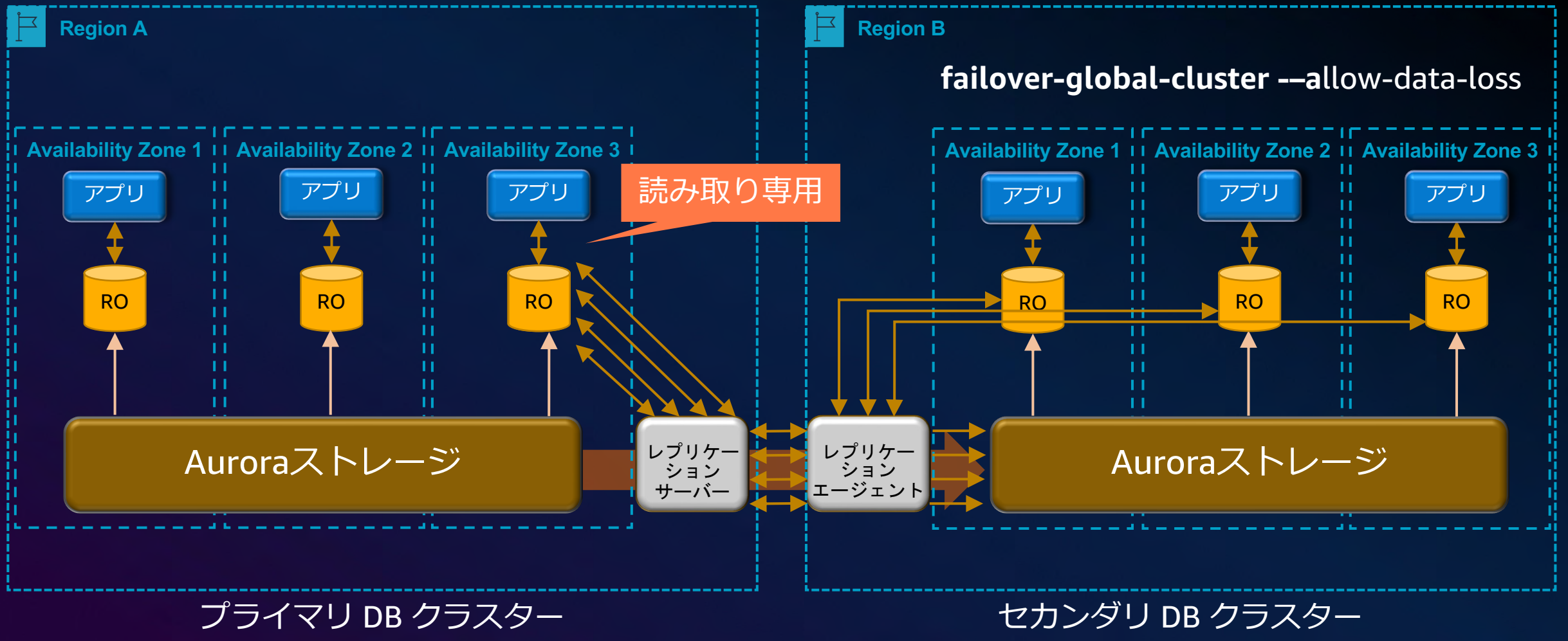
Aurora Global Database - Failover

RW：ライター
RO：リードレプリカ



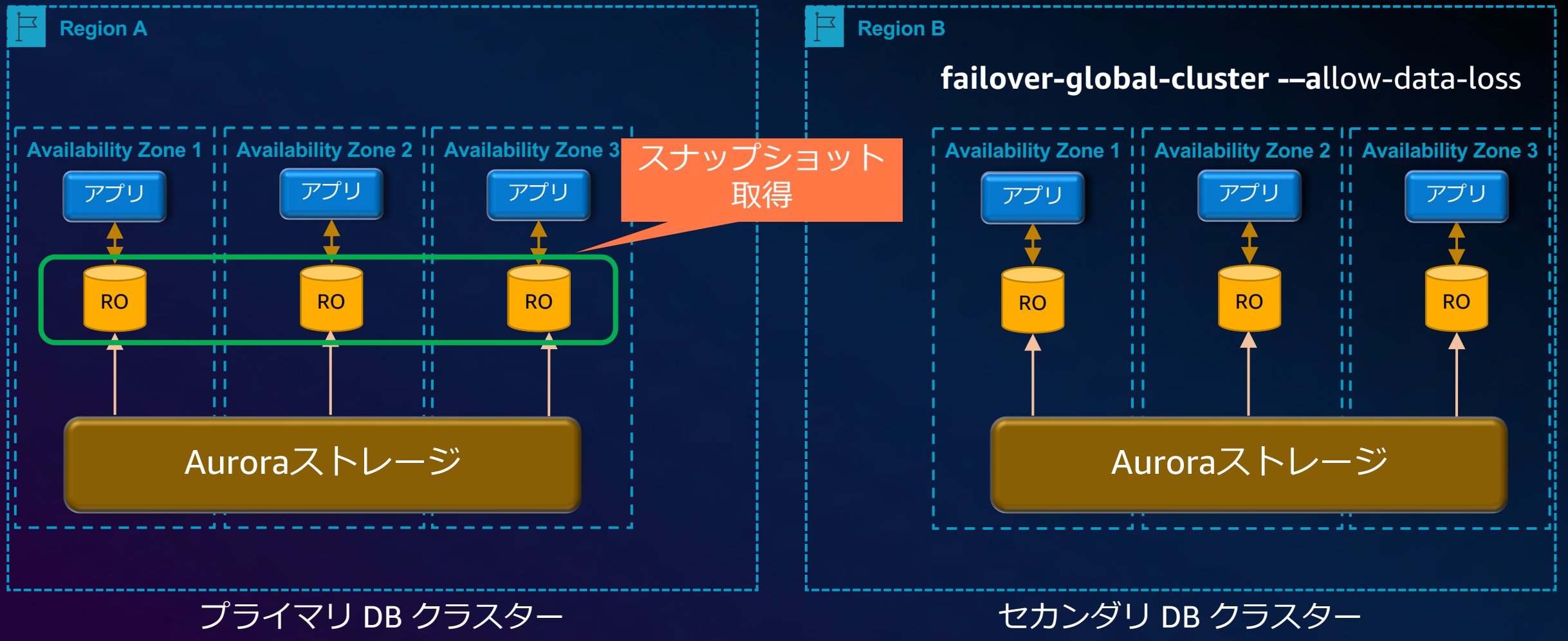
Aurora Global Database - Failover

RW：ライター
RO：リードレプリカ



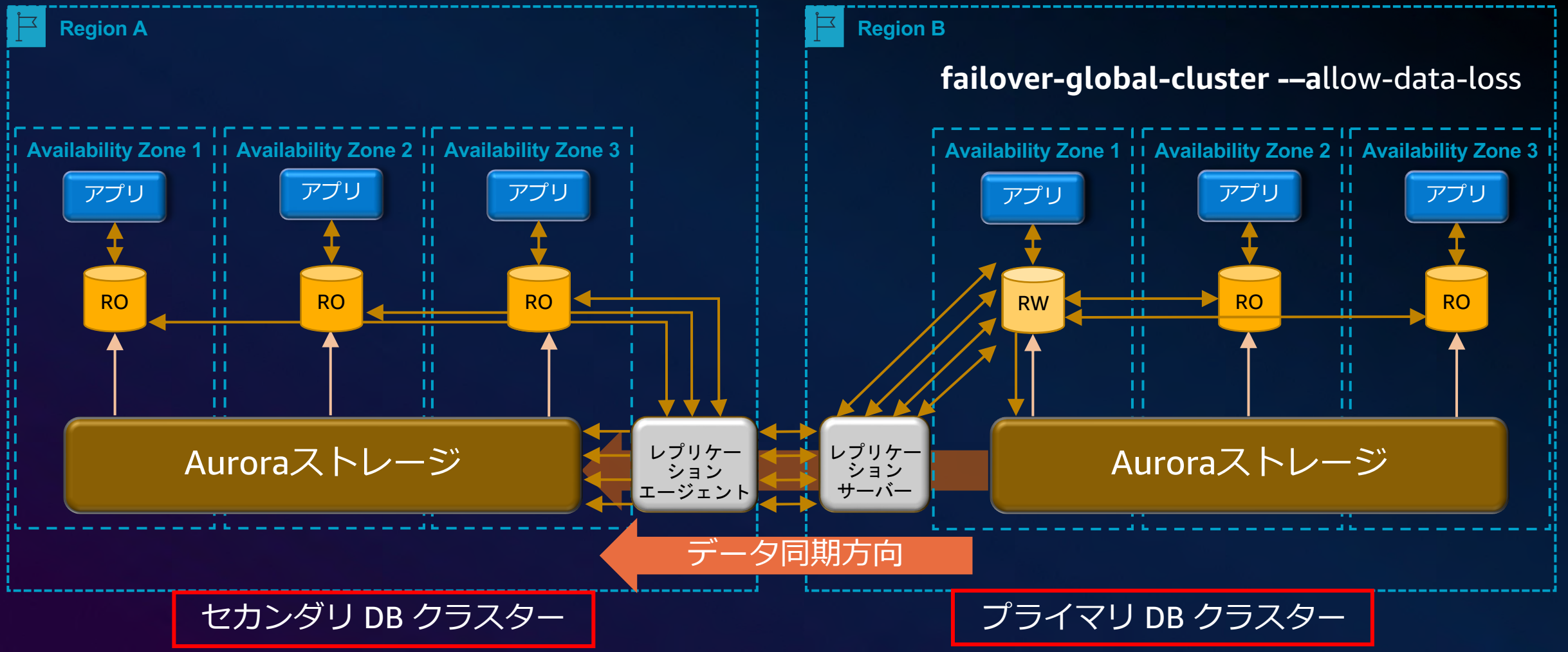
Aurora Global Database - Failover

RW：ライター
RO：リードレプリカ



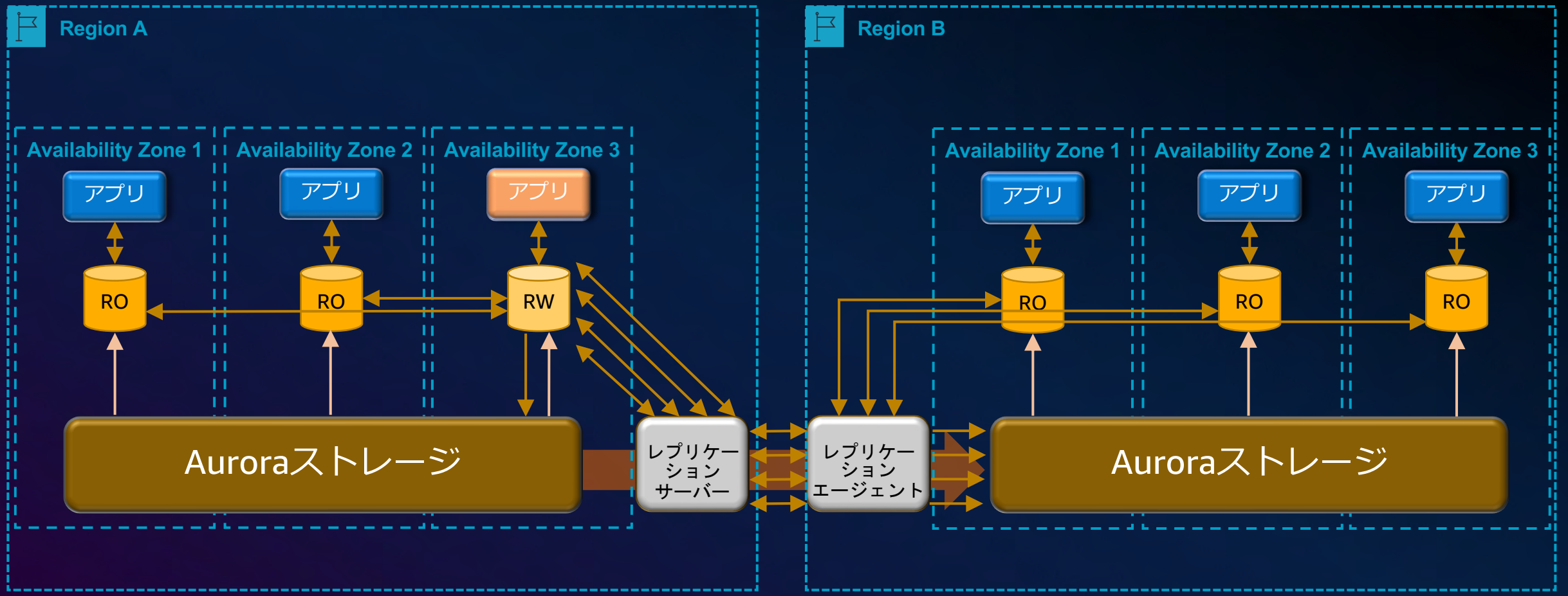
Aurora Global Database - Failover

RW：ライター
RO：リードレプリカ



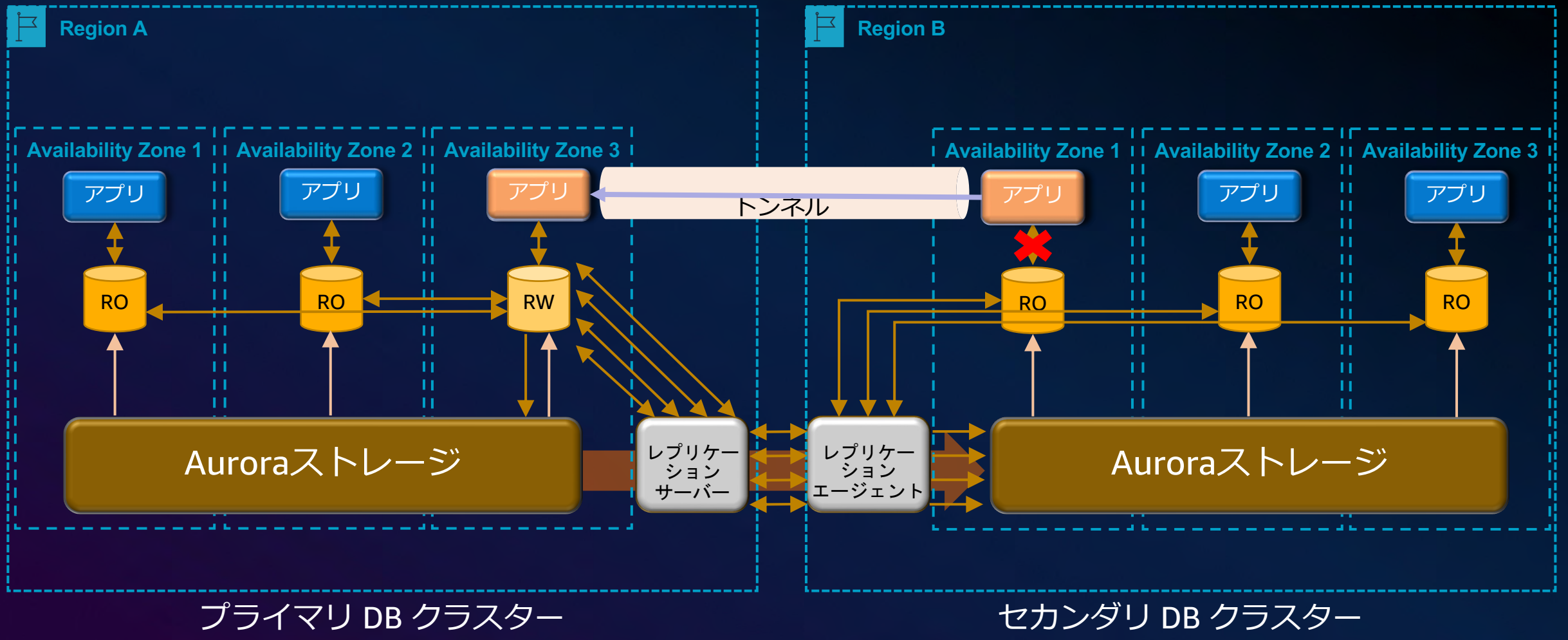
Aurora Global Database - Write Forwarding

RW：ライター
RO：リードレプリカ



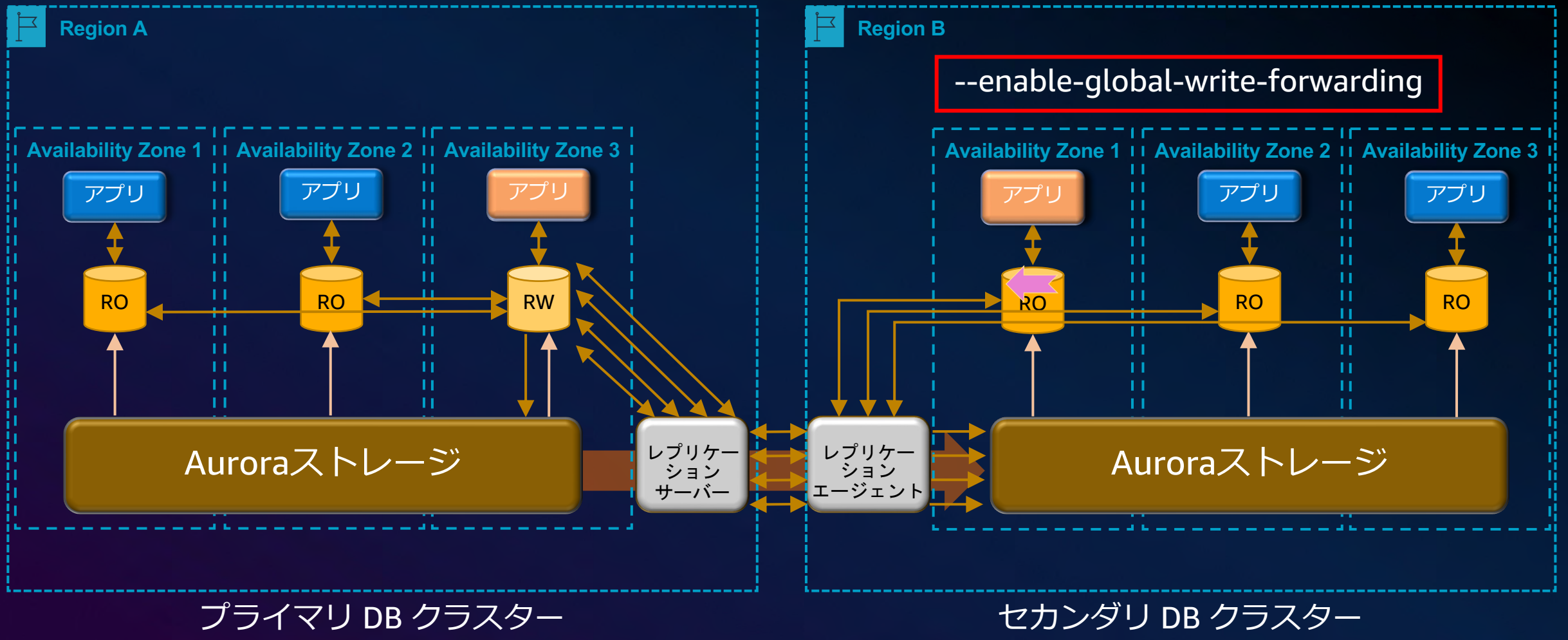
Aurora Global Database - Write Forwarding

RW：ライター
RO：リードレプリカ



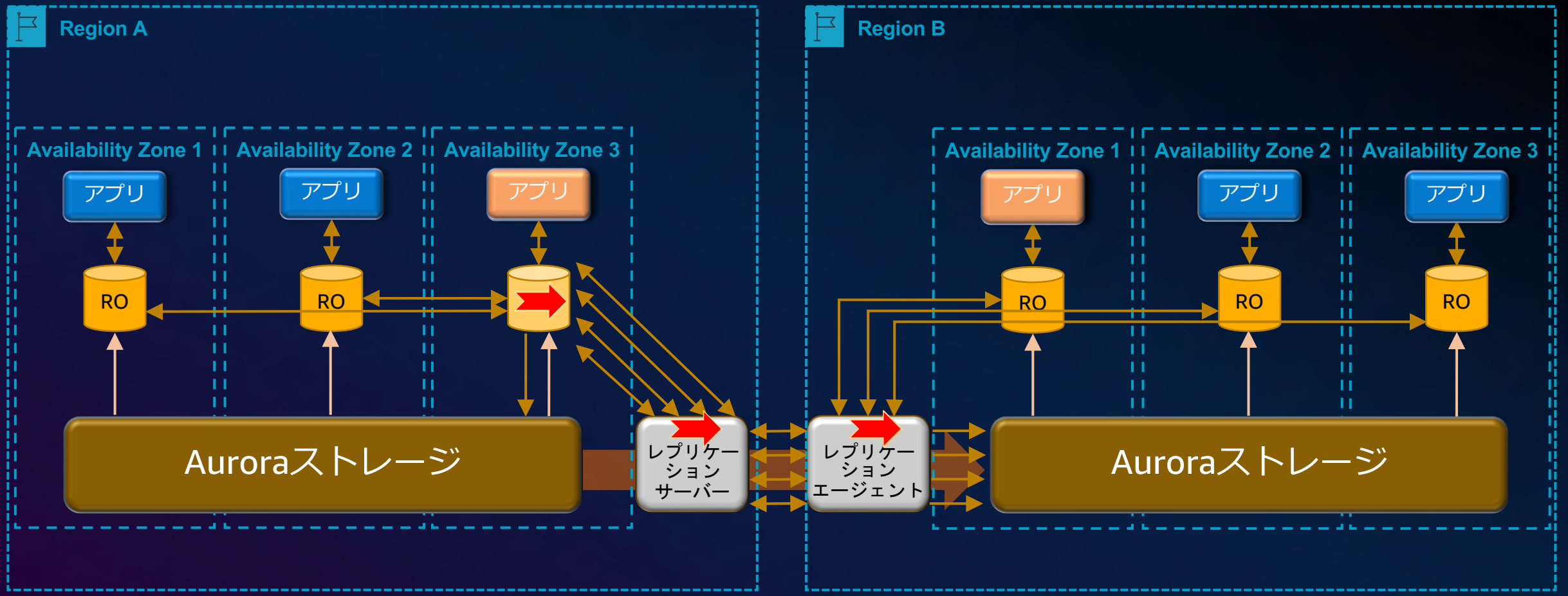
Aurora Global Database - Write Forwarding

RW：ライター
RO：リードレプリカ



Aurora Global Database - Write Forwarding

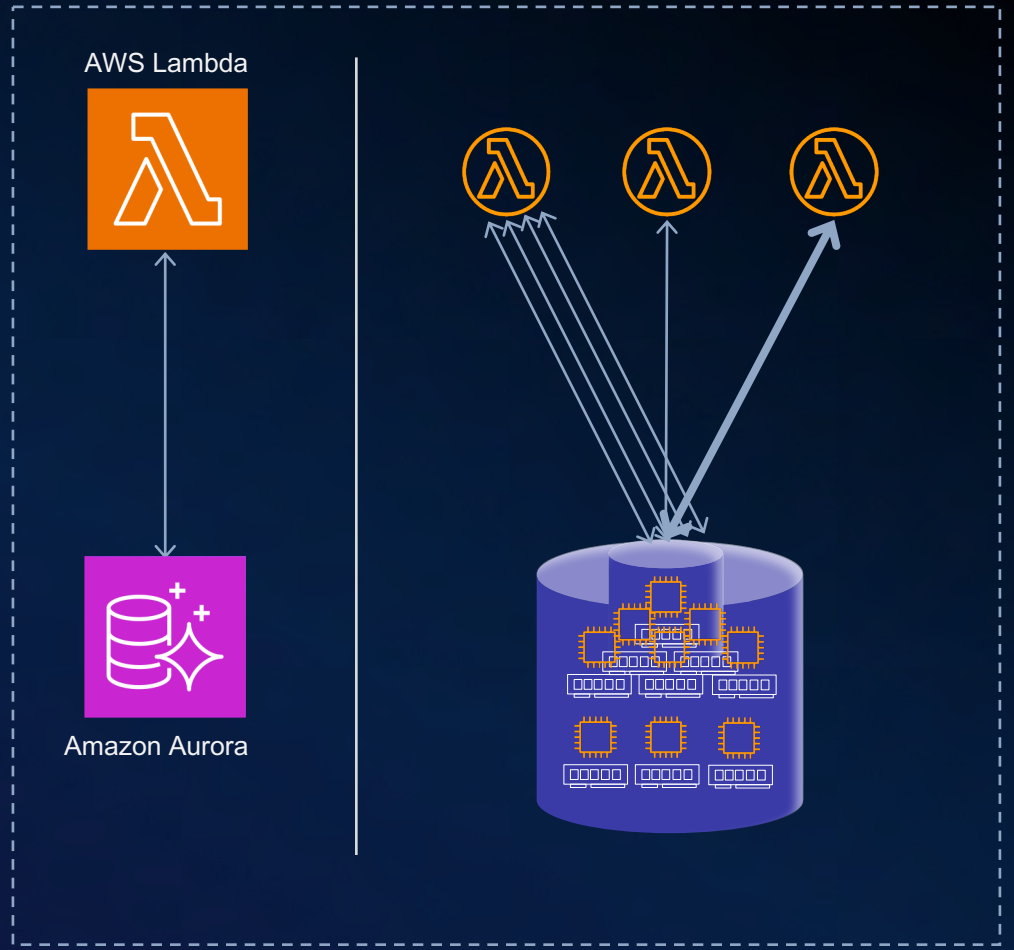
RW：ライター
RO：リードレプリカ



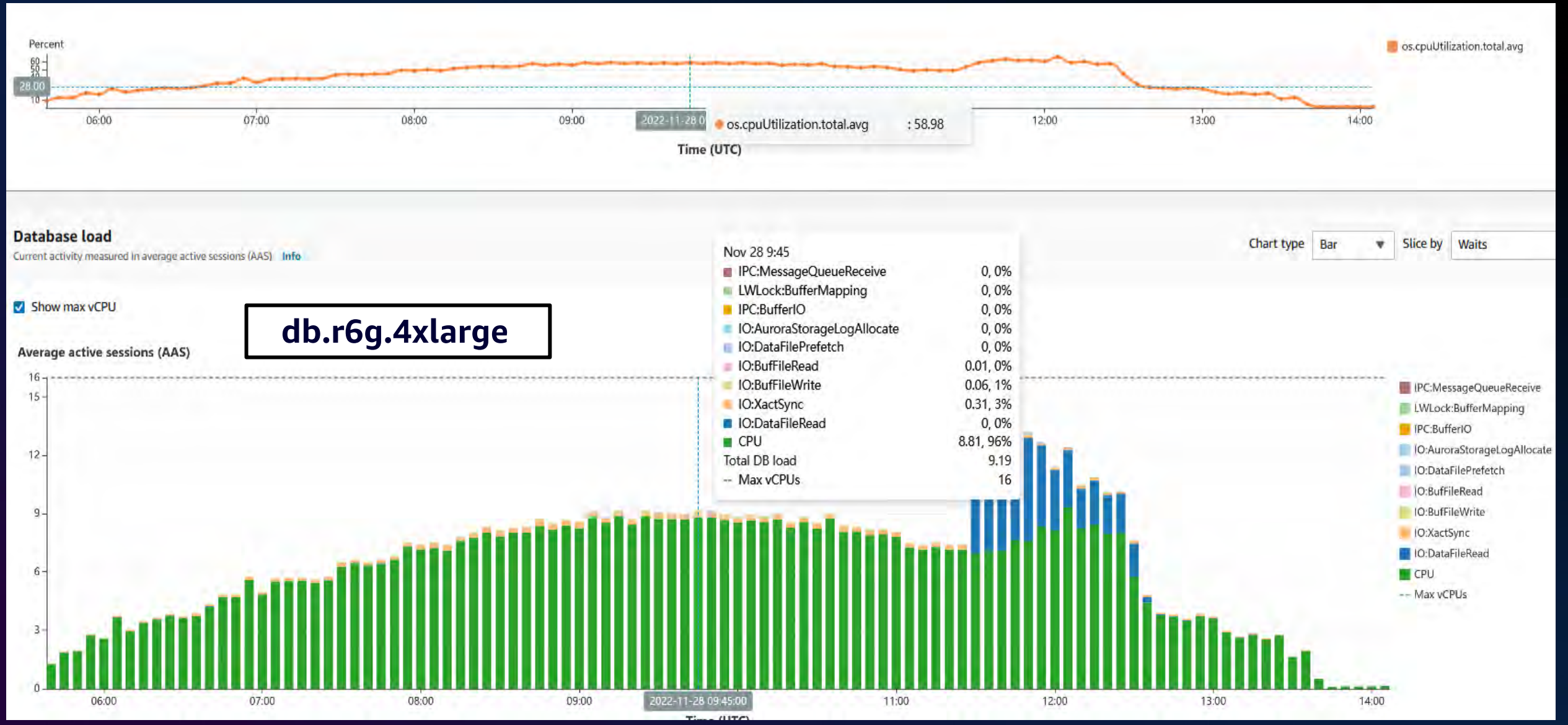
管理性 – Aurora Serverless

Amazon Aurora Serverless v2

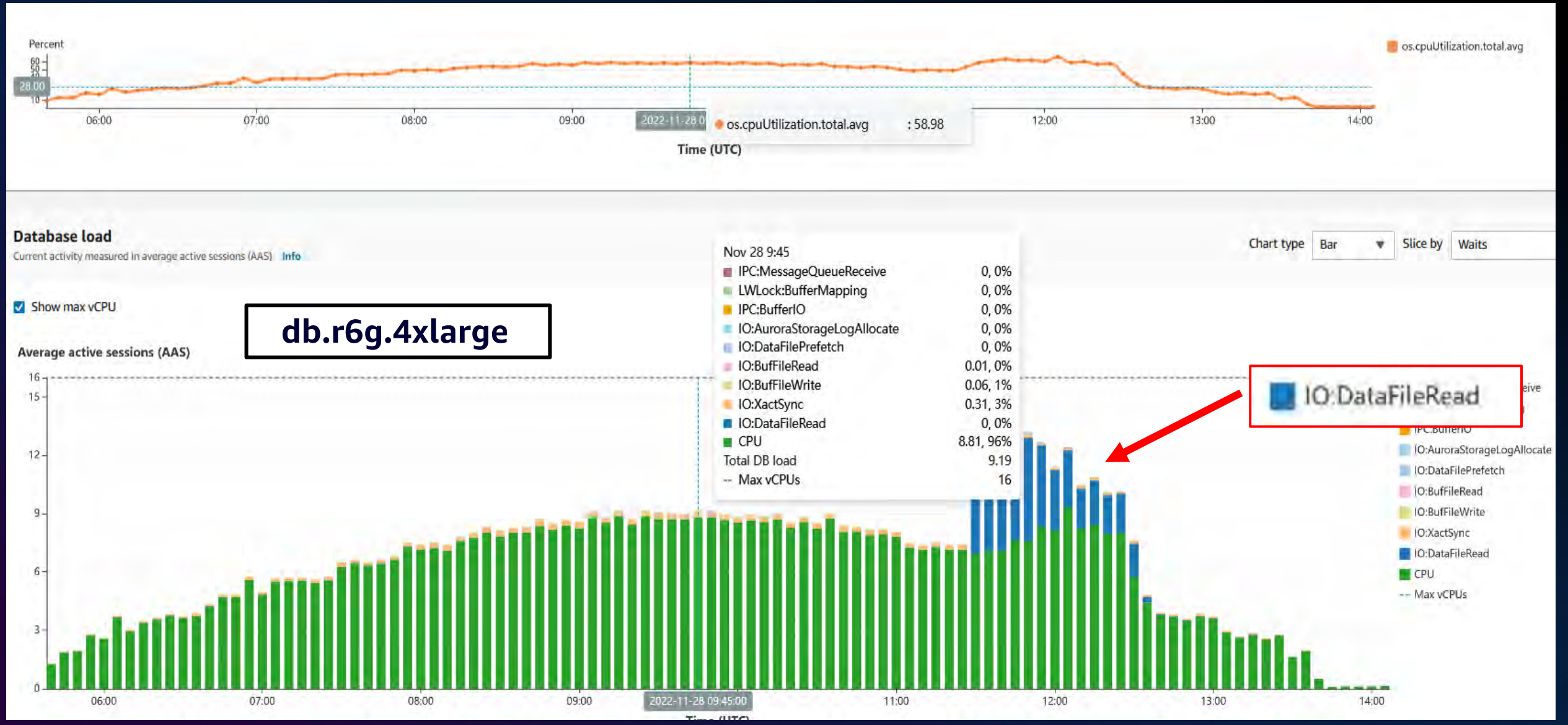
- CPU とメモリのリソースを追加することで１秒以内に自動的にスケールし、秒単位で課金されます。
- 数十万のトランザクションを実行してもスケーリングによる影響なし
- シンプルな 1 秒ごとの従量課金制



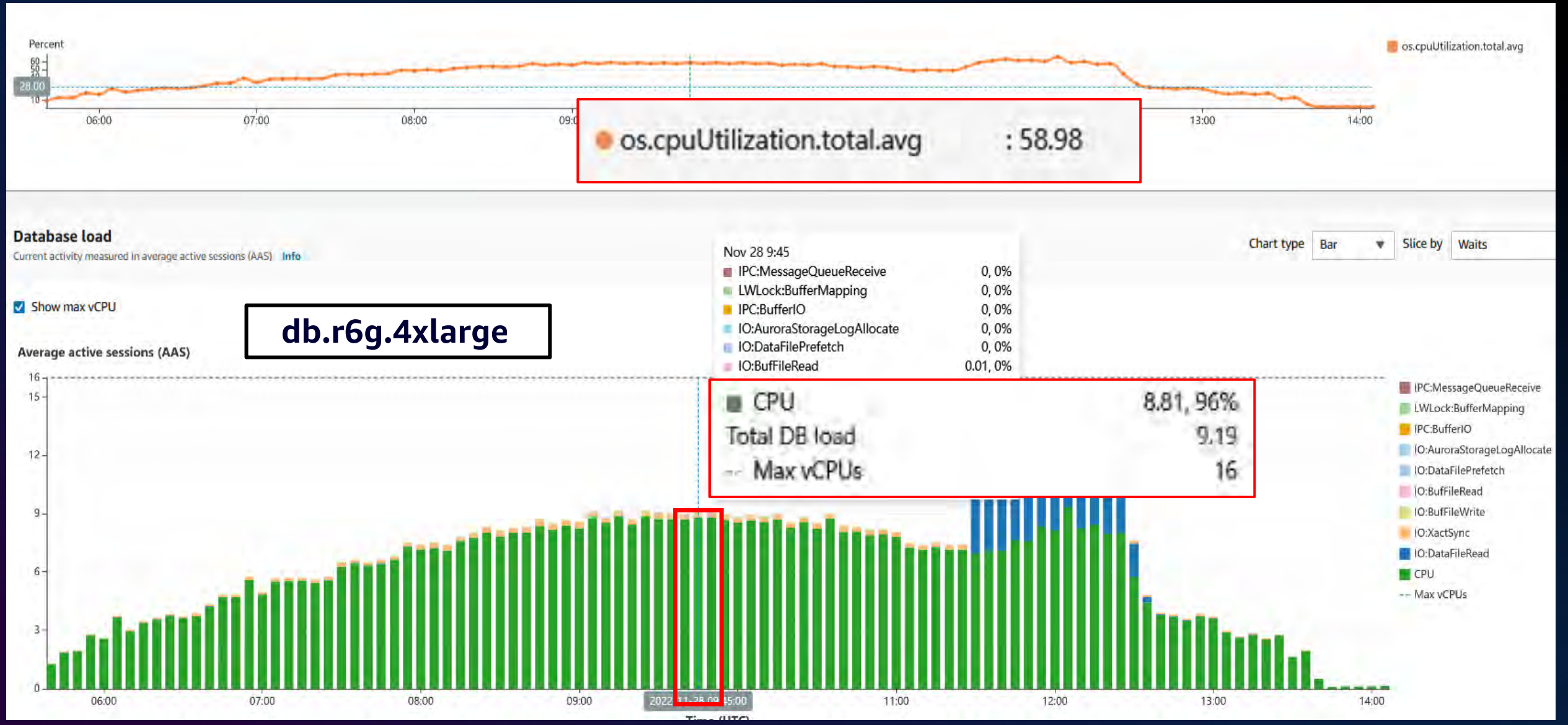
Aurora – ワークロード例



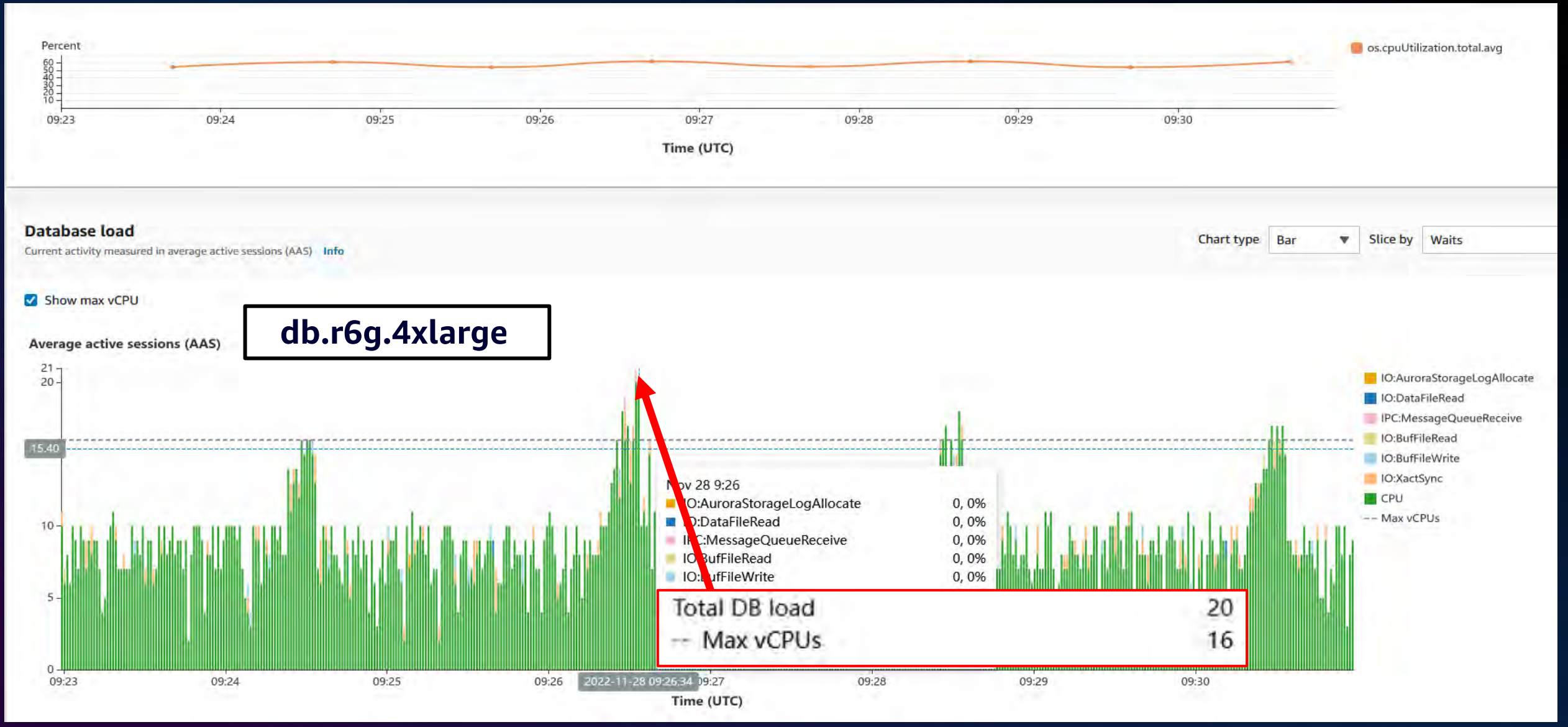
Aurora – ワークロード例



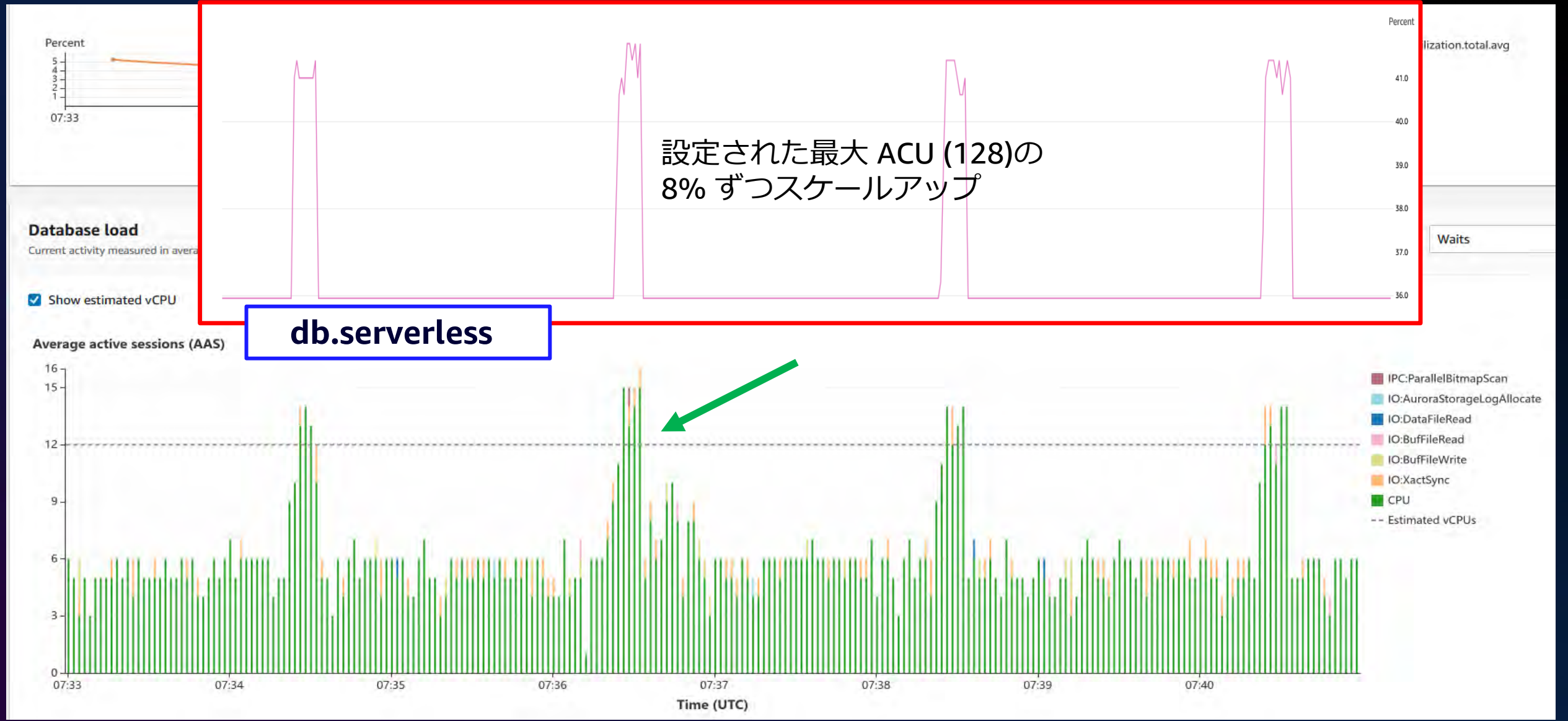
Aurora – ワークロード例



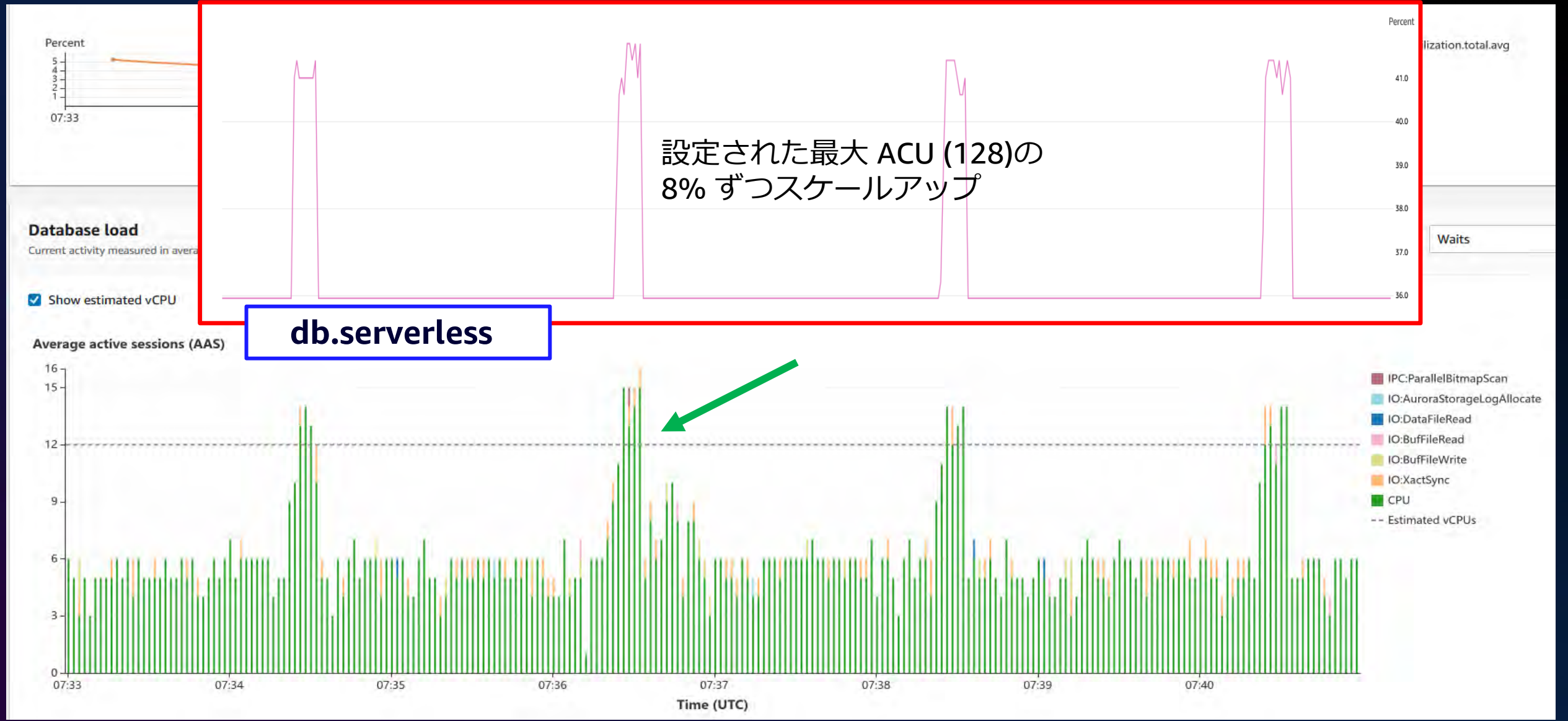
Aurora – ワークロード例 (DB load過多)



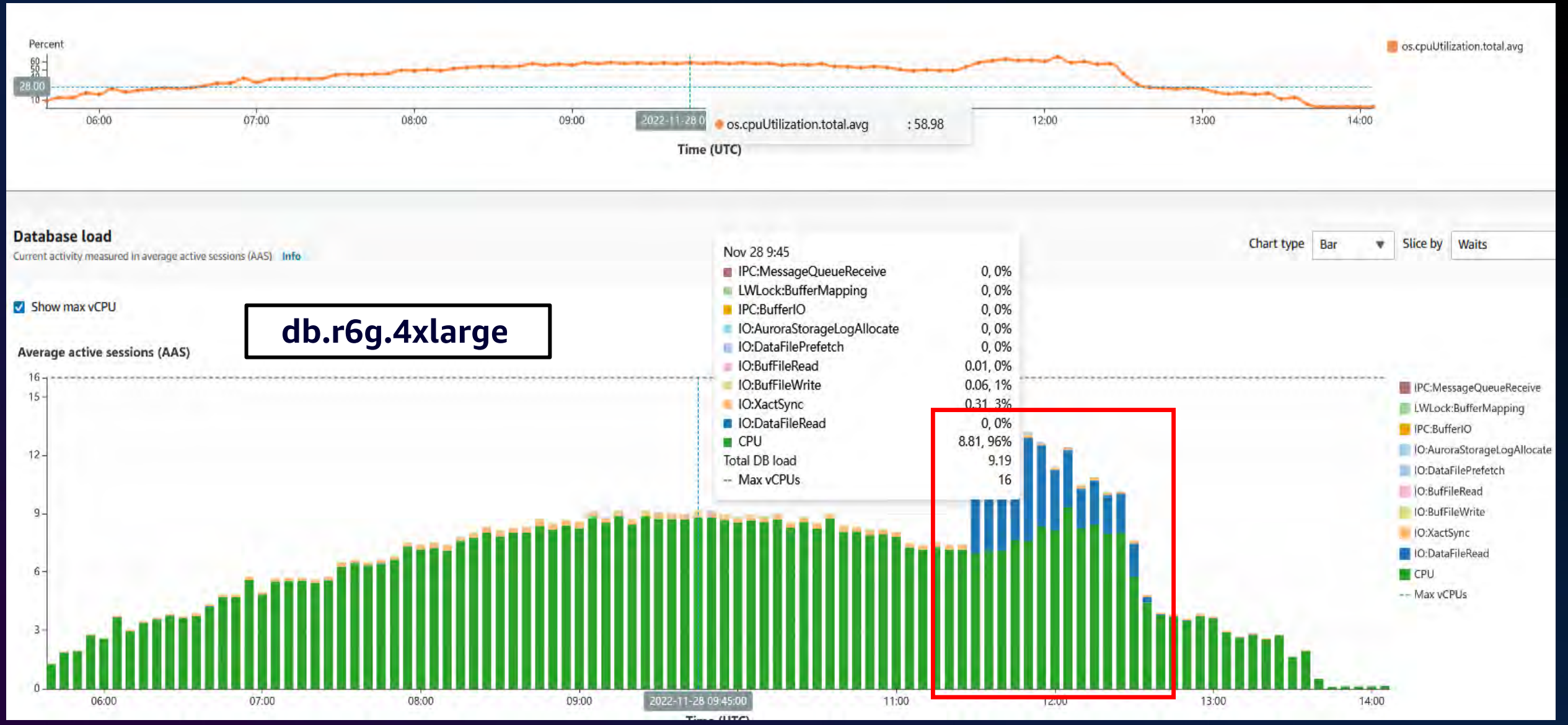
Aurora Serverless – CPU スケール



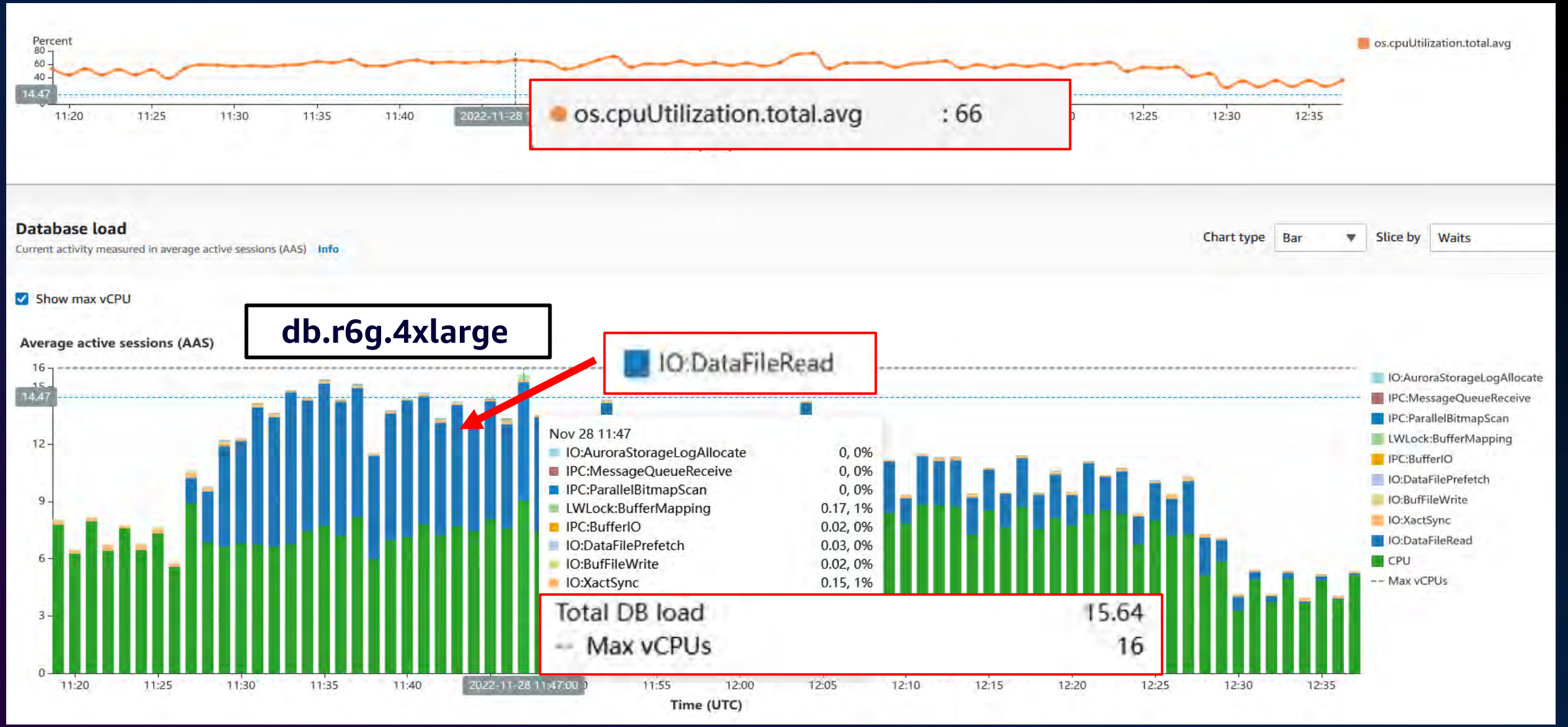
Aurora Serverless – CPU スケール



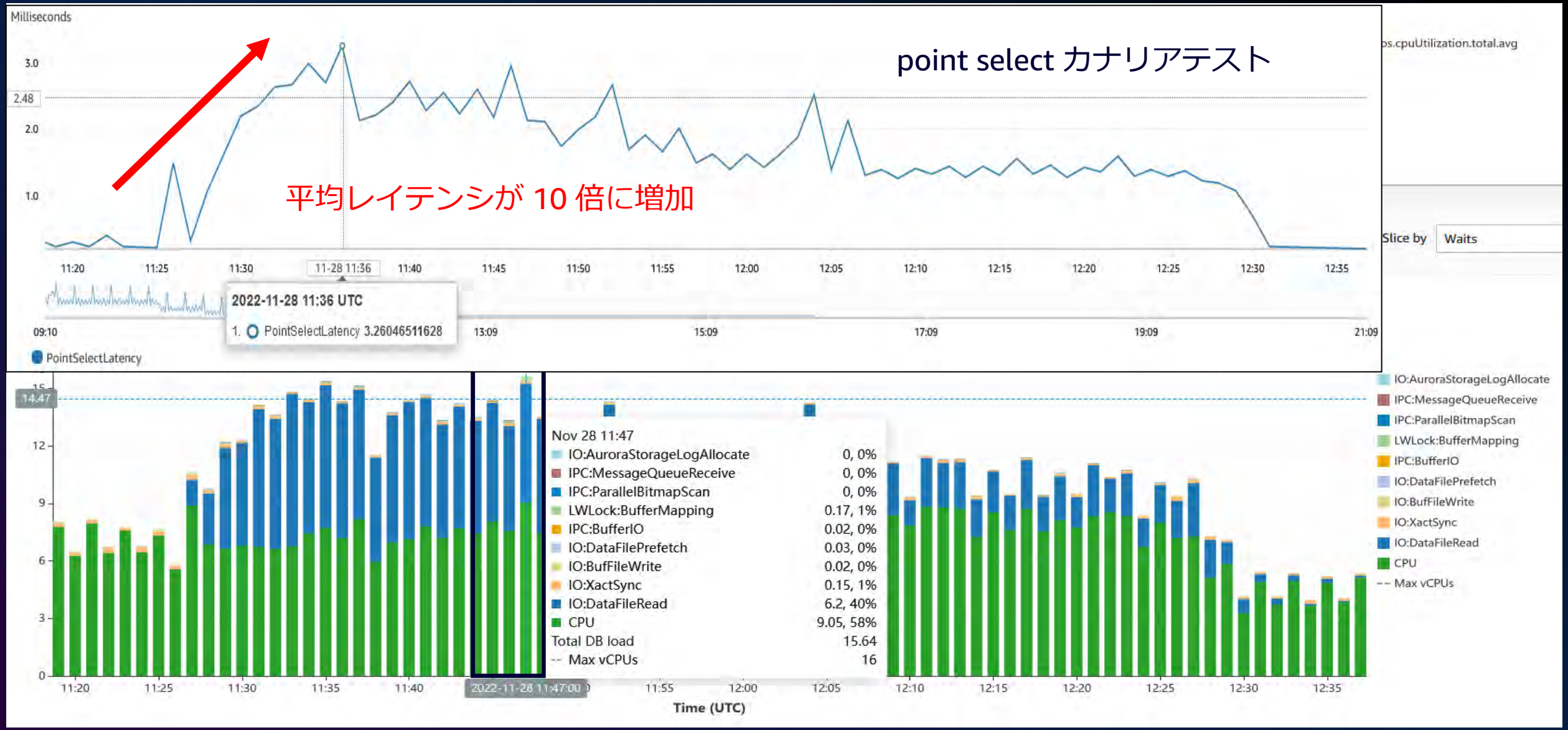
Aurora –ワークロード例



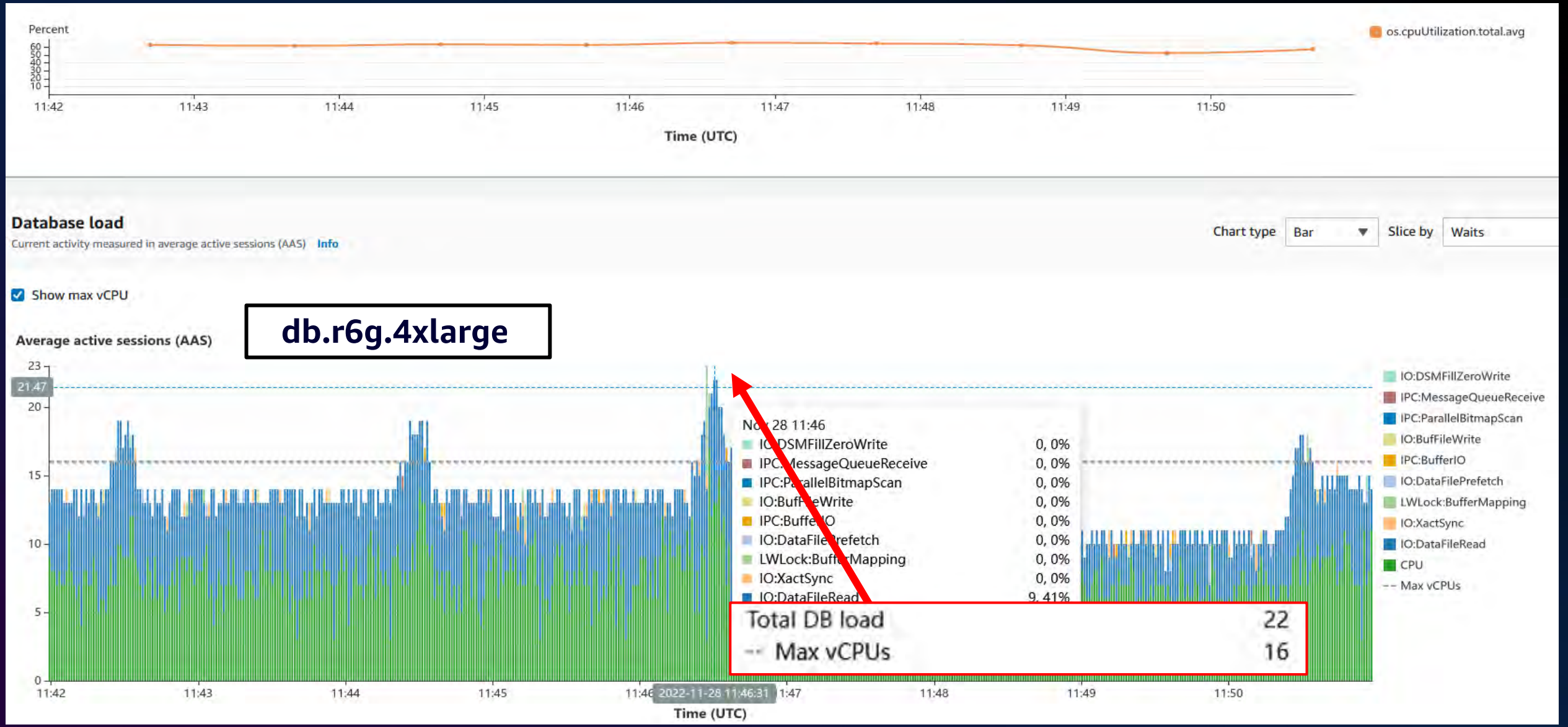
Aurora – ワークロード例 (IO待機)



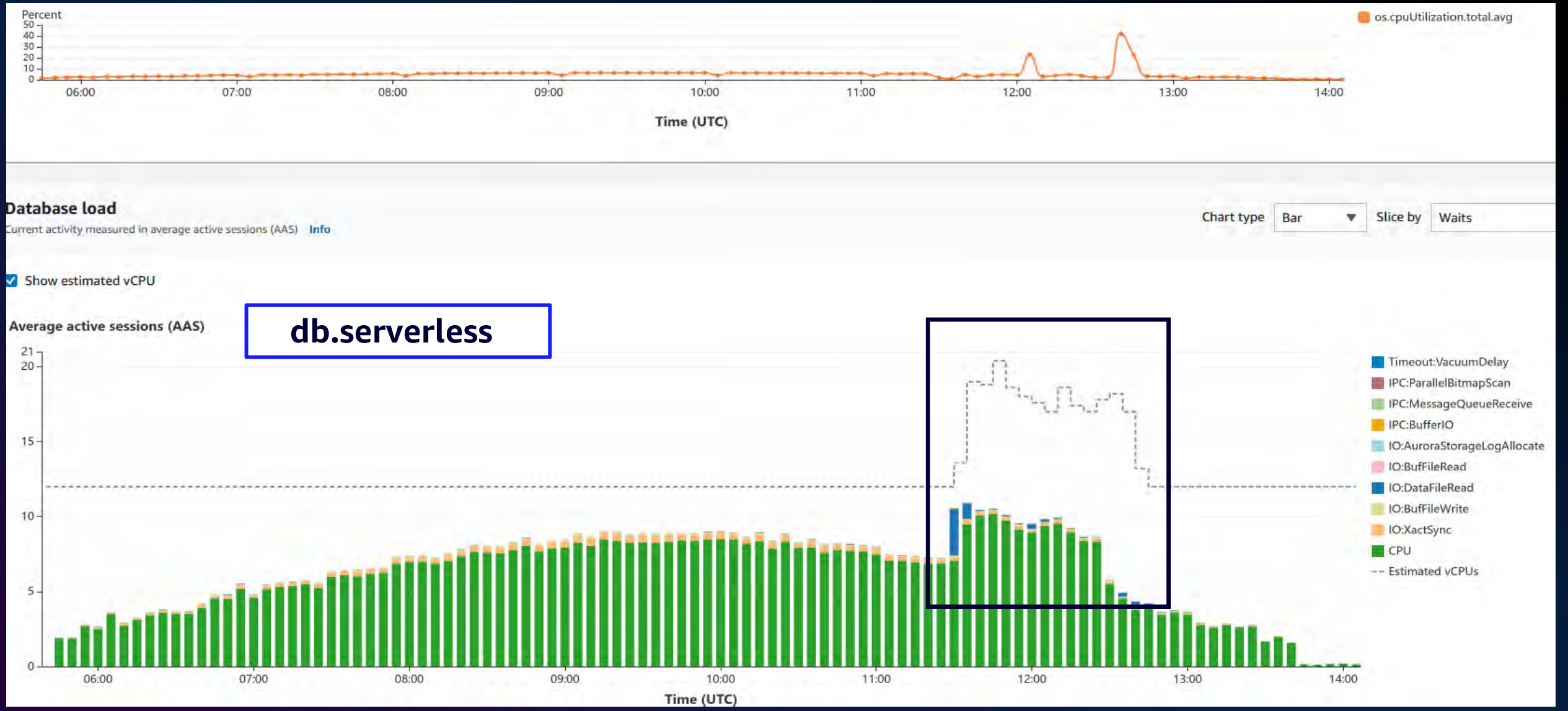
Aurora – ワークロード例 (IO待機)



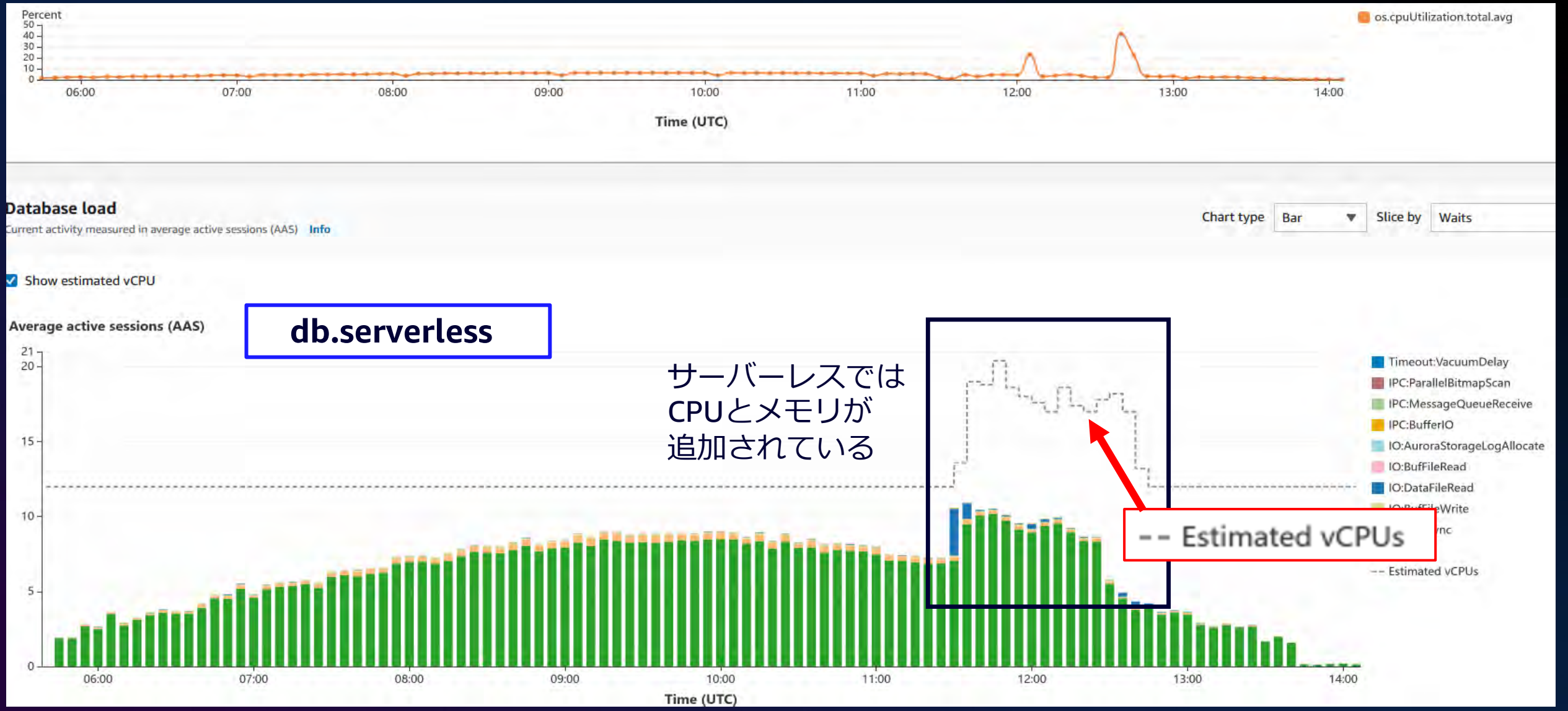
Aurora – ワークロード例 (IO待機)



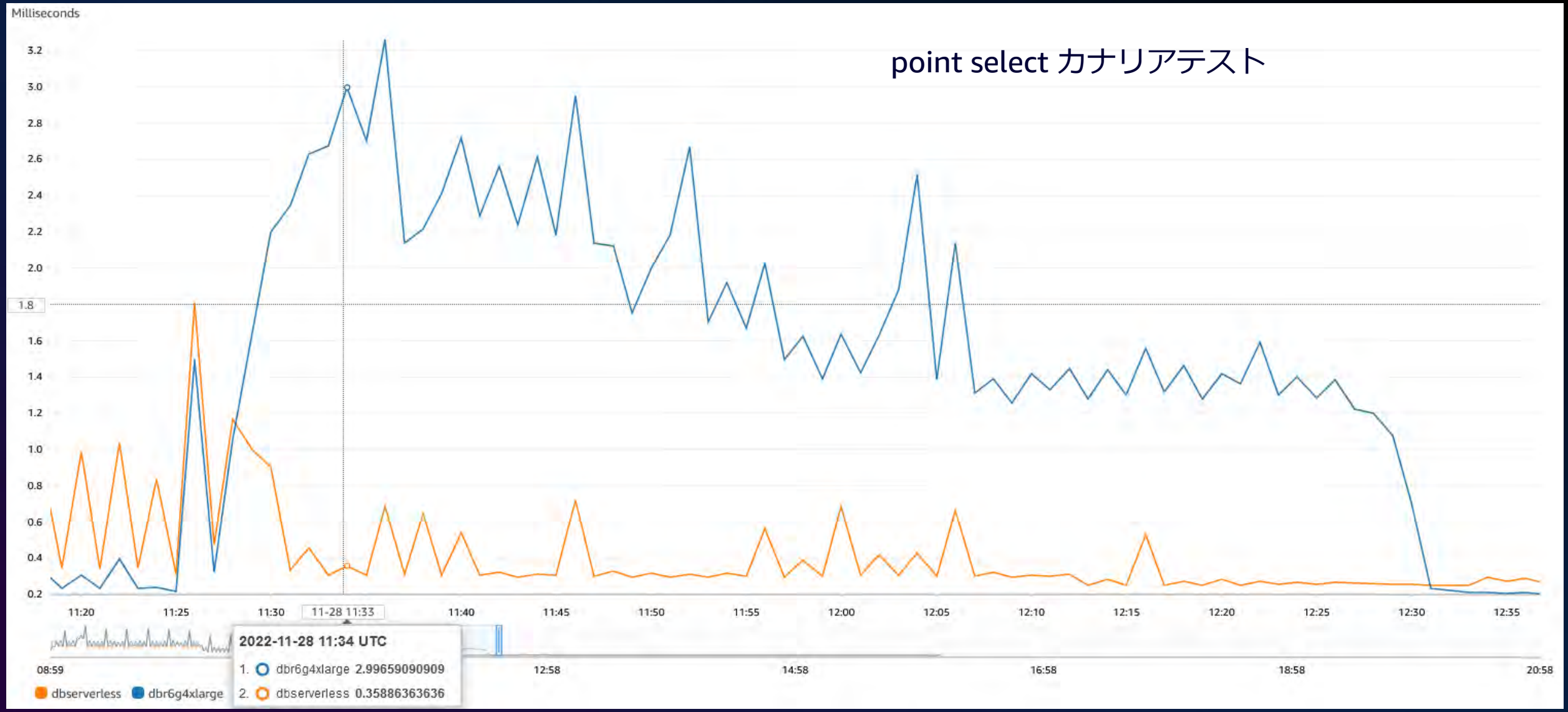
Aurora Serverless – メモリと CPU のスケール



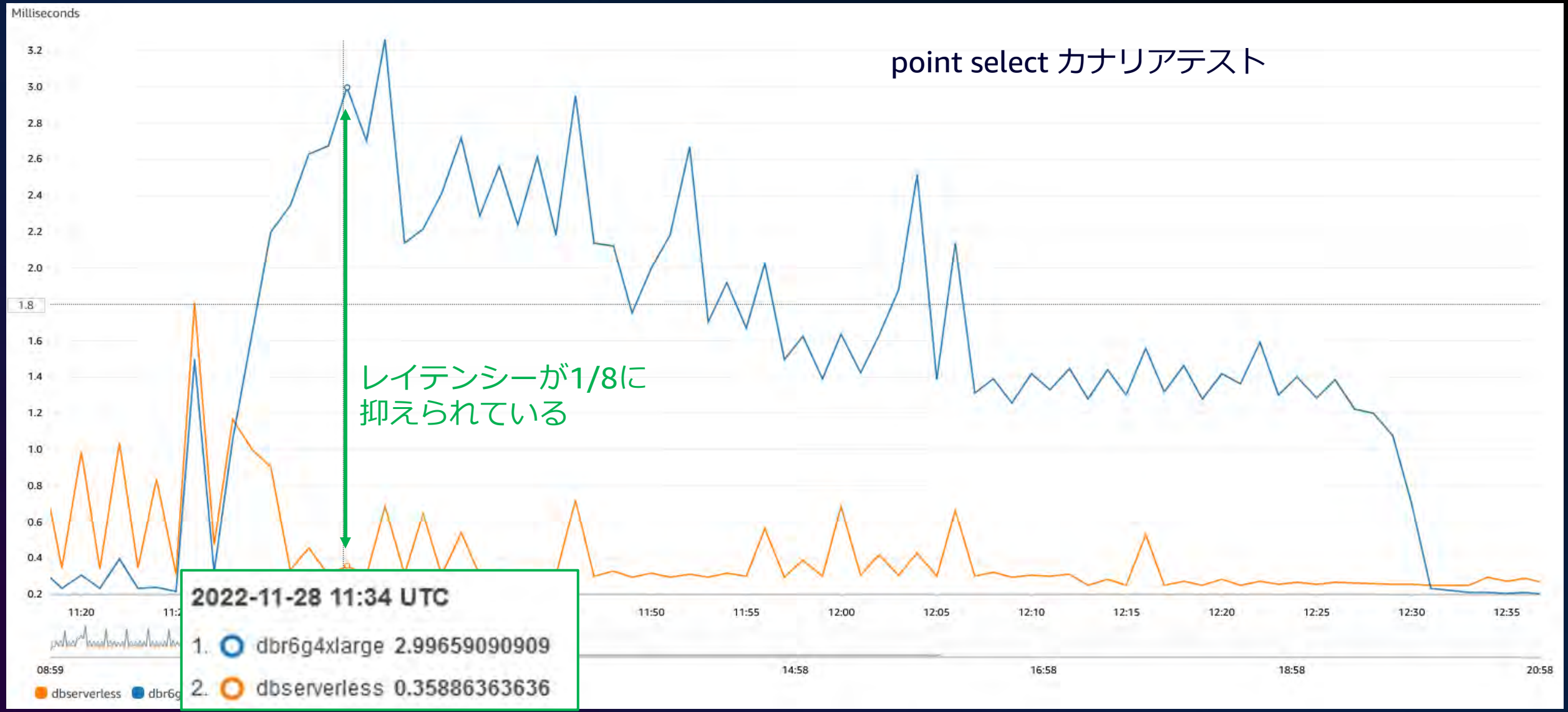
Aurora Serverless – メモリと CPU のスケール



Aurora Serverless – カナリアテスト比較

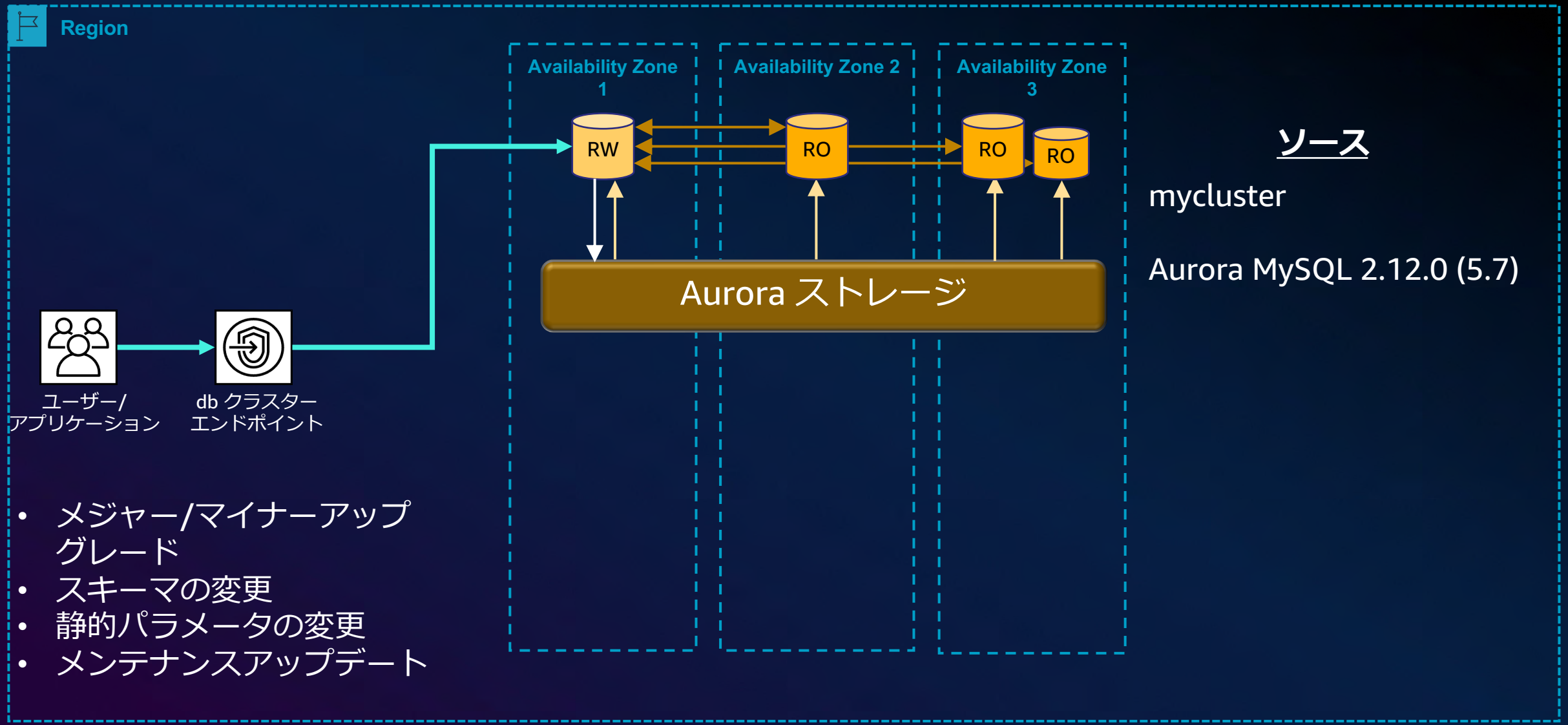


Aurora Serverless – カナリアテスト比較



管理性 – Managed Blue/Green deployment

Amazon Aurora Managed Blue/Green deployment



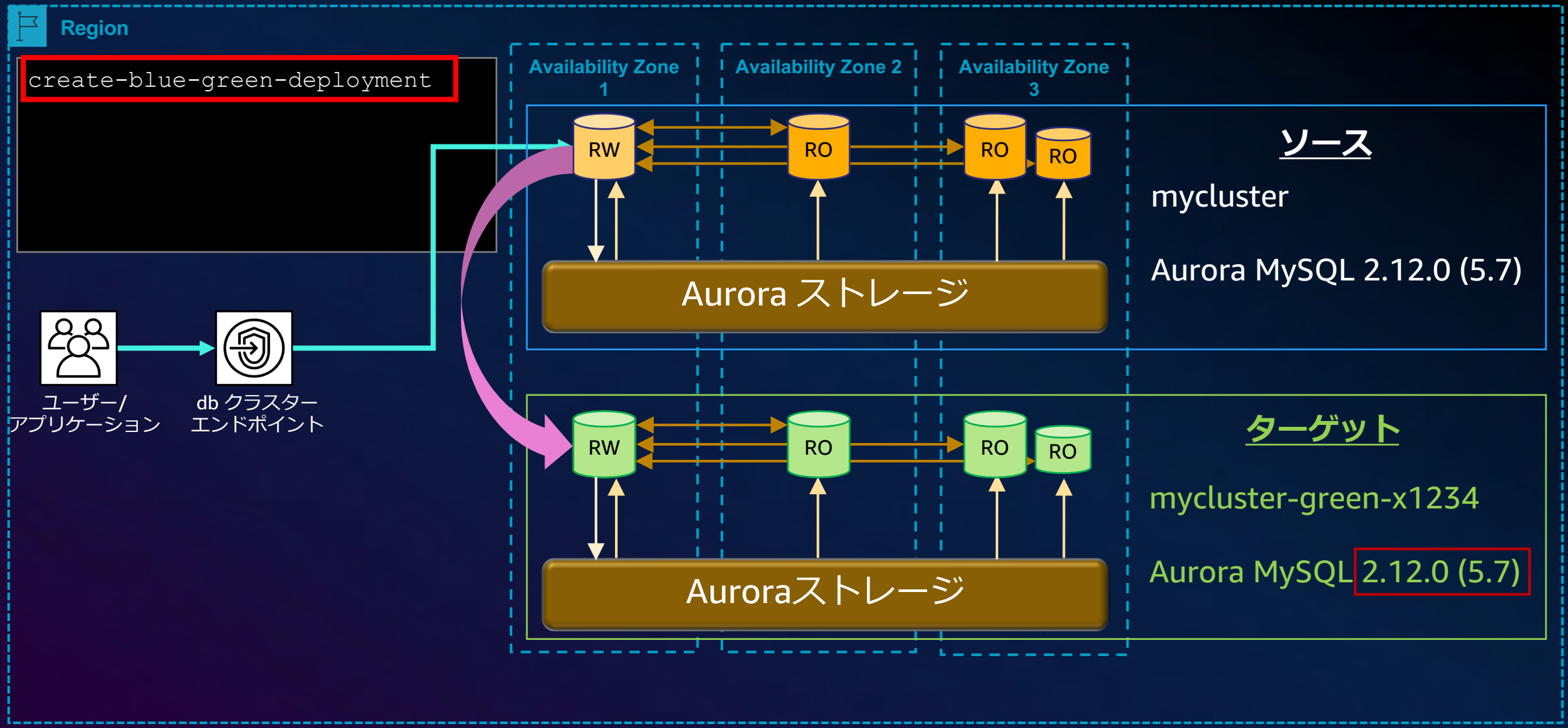
ソース

mycluster

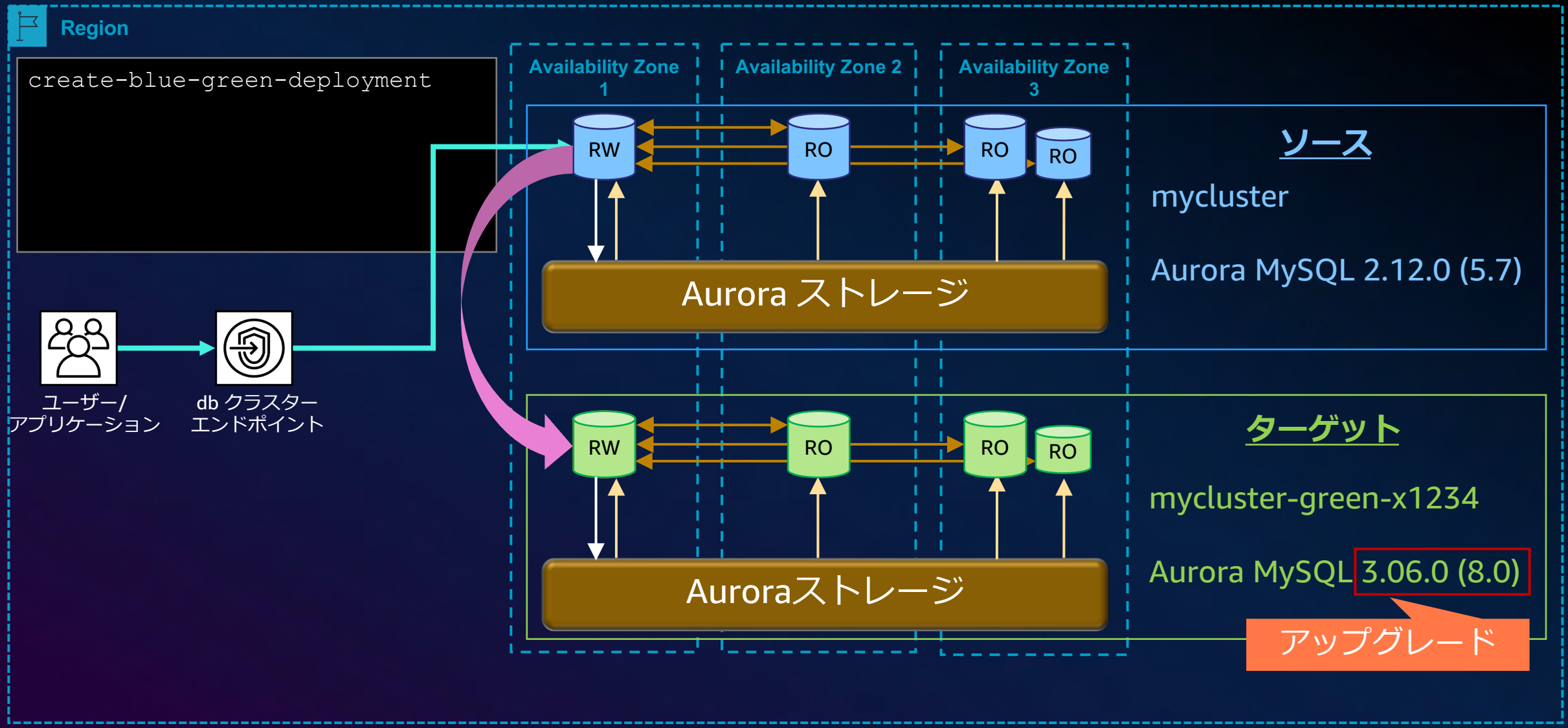
Aurora MySQL 2.12.0 (5.7)

- メジャー/マイナーアップグレード
- スキーマの変更
- 静的パラメータの変更
- メンテナンスアップデート

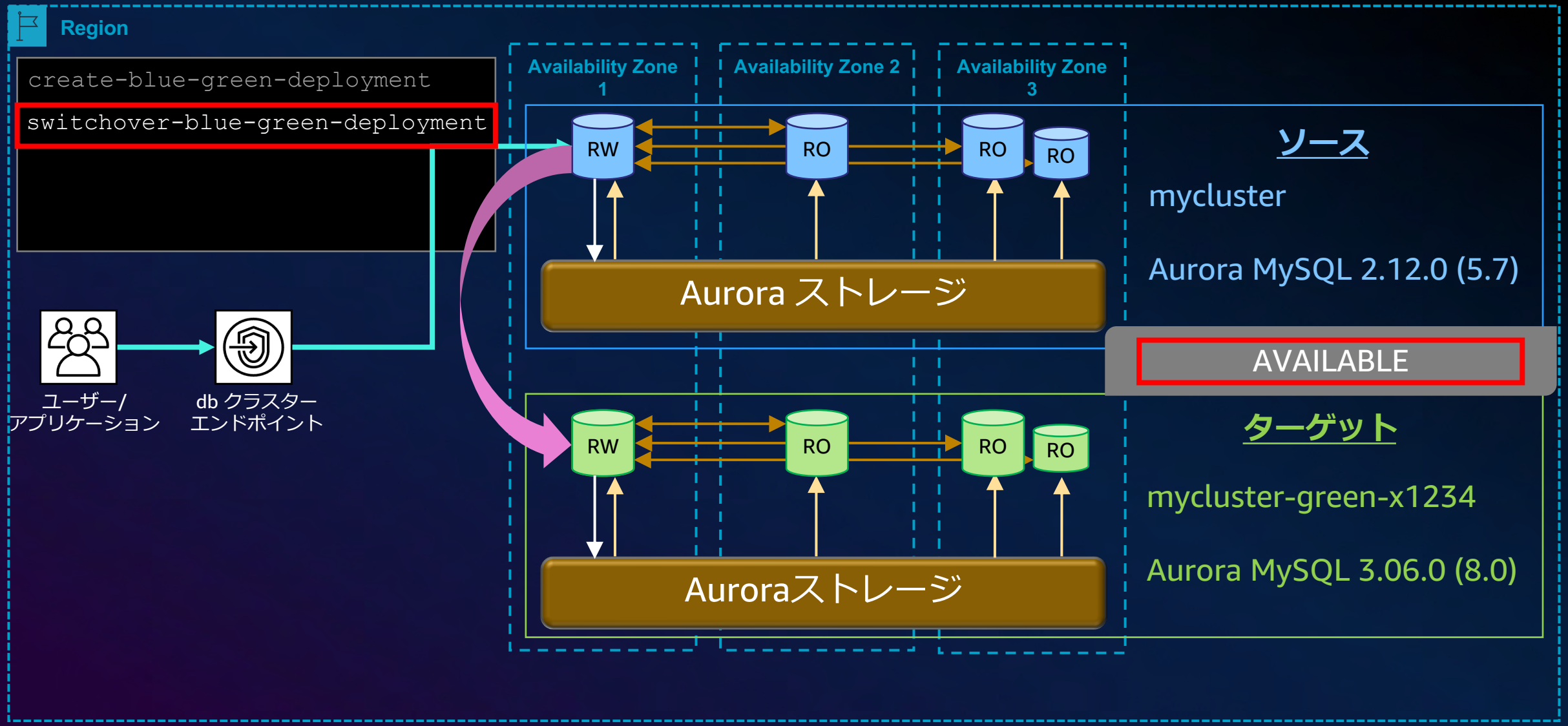
Amazon Aurora Managed Blue/Green deployment



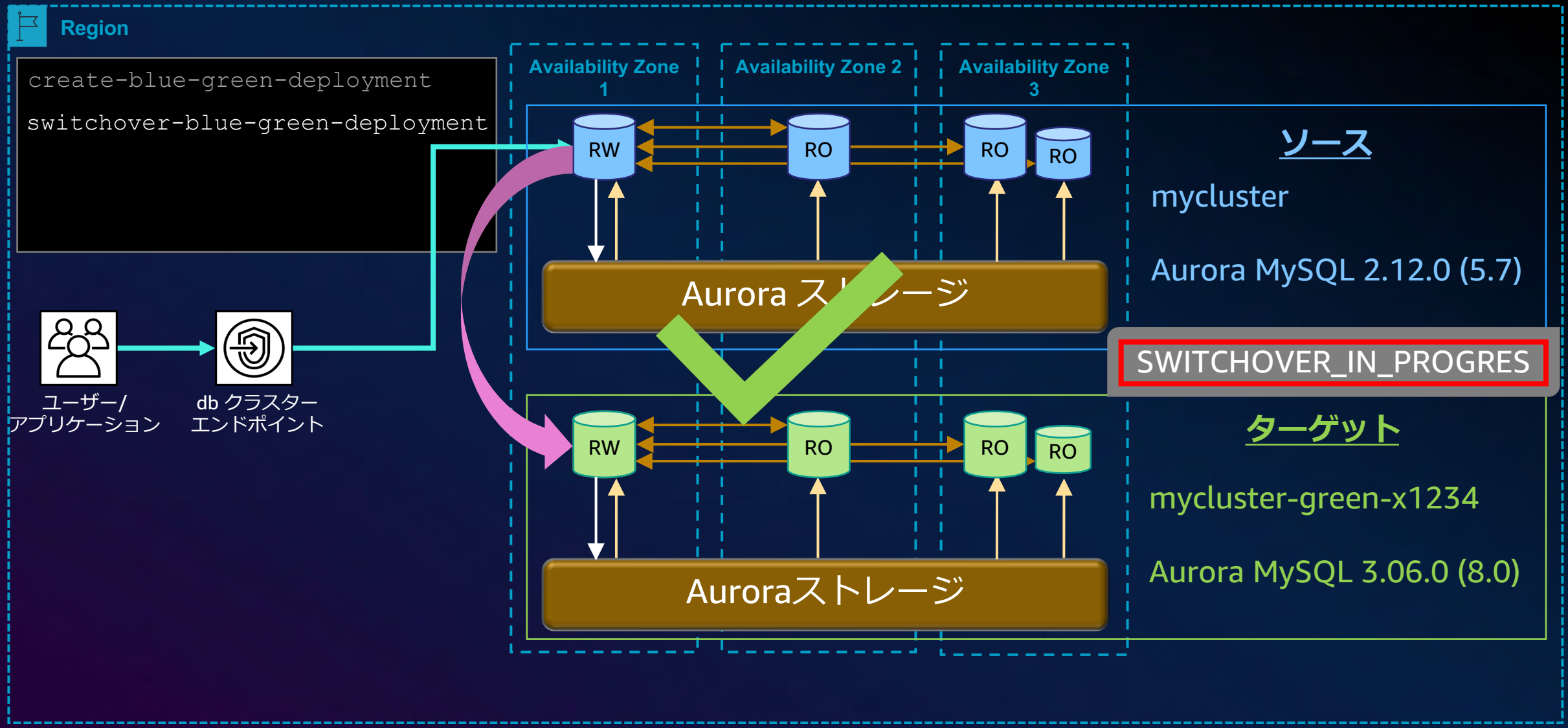
Amazon Aurora Managed Blue/Green deployment



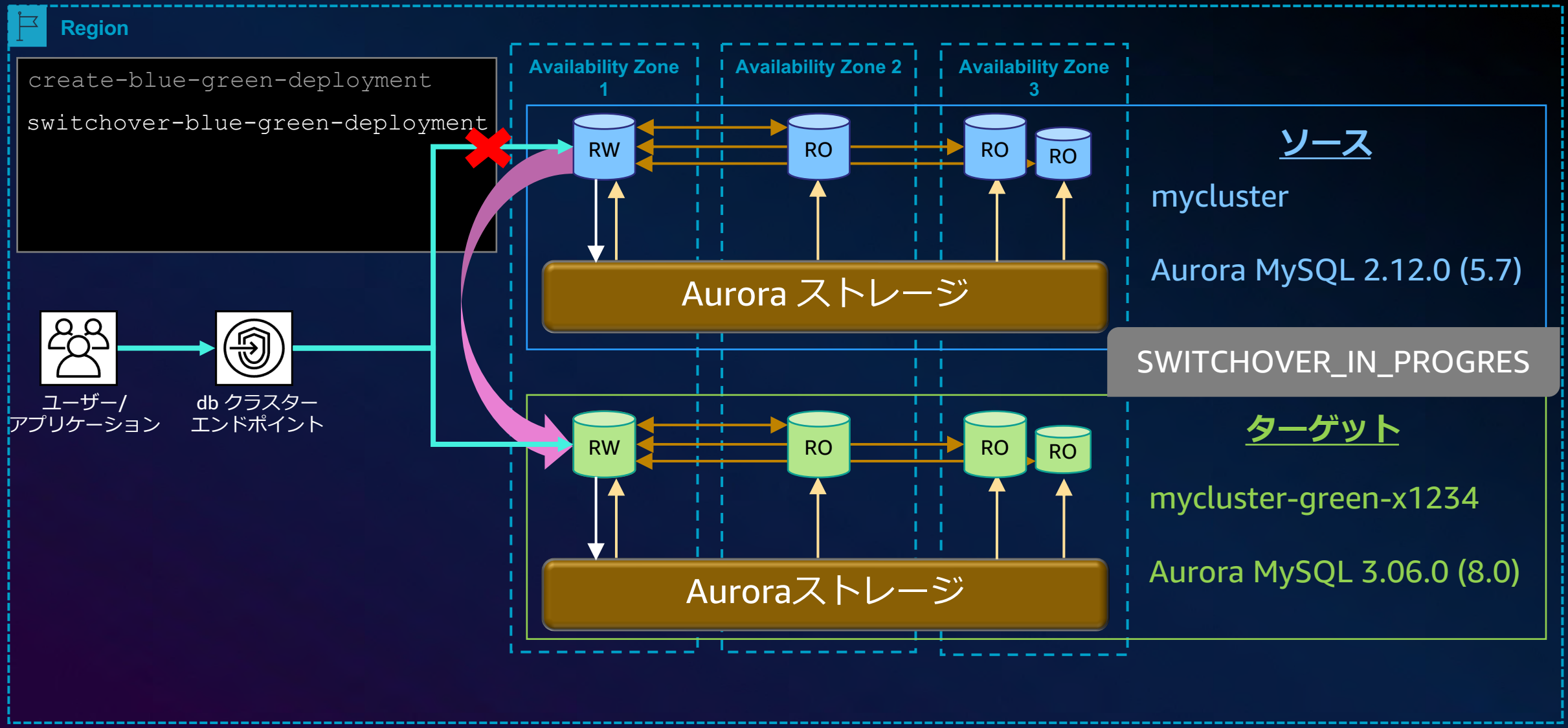
Amazon Aurora Managed Blue/Green deployment



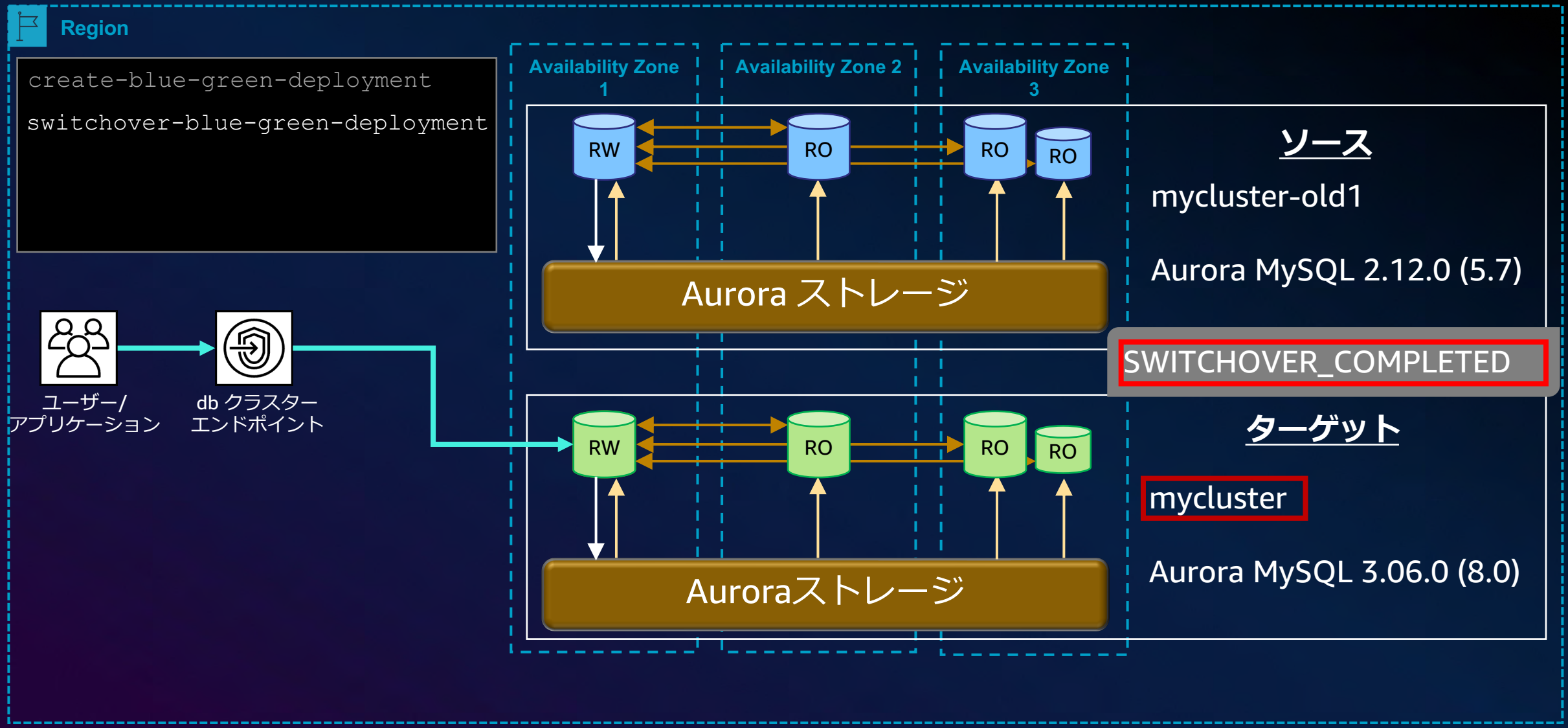
Amazon Aurora Managed Blue/Green deployment



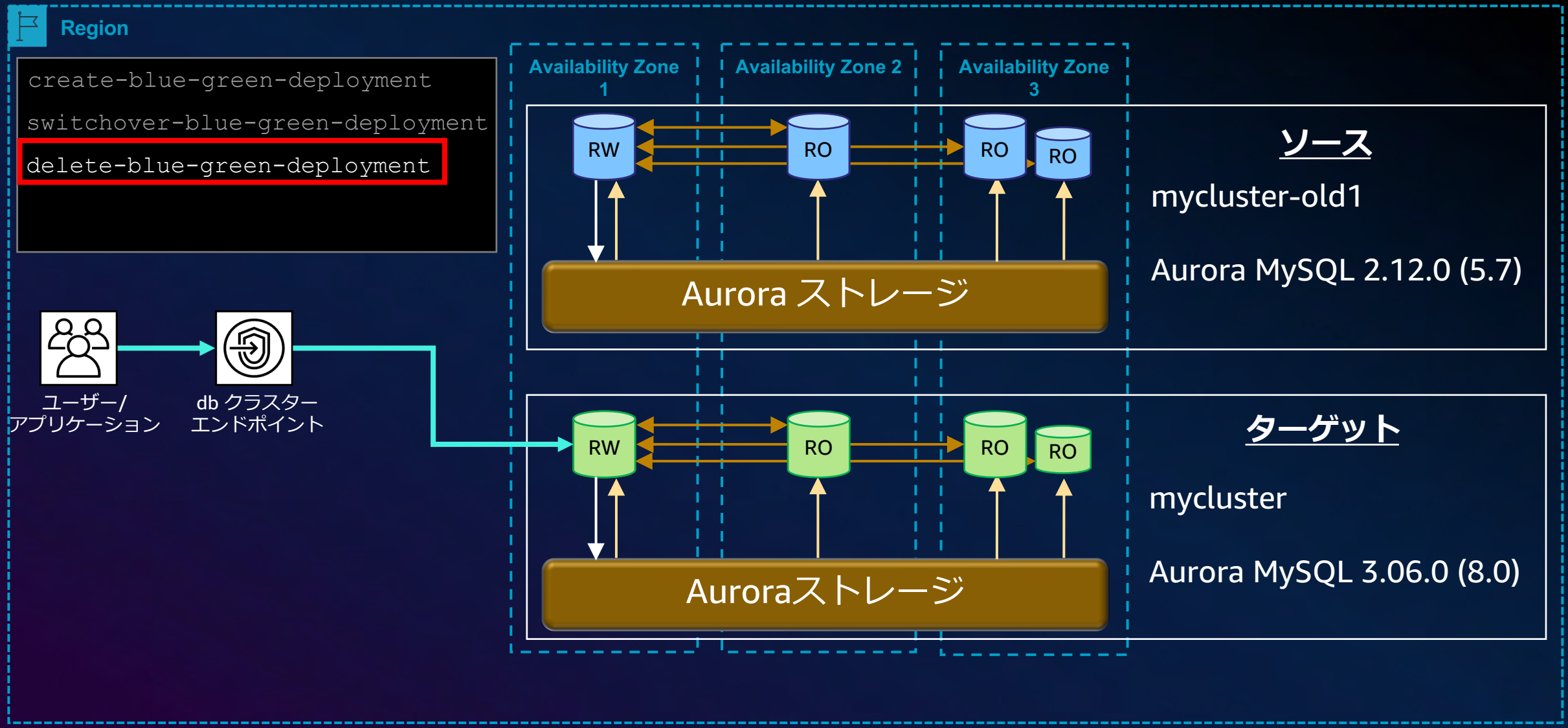
Amazon Aurora Managed Blue/Green deployment



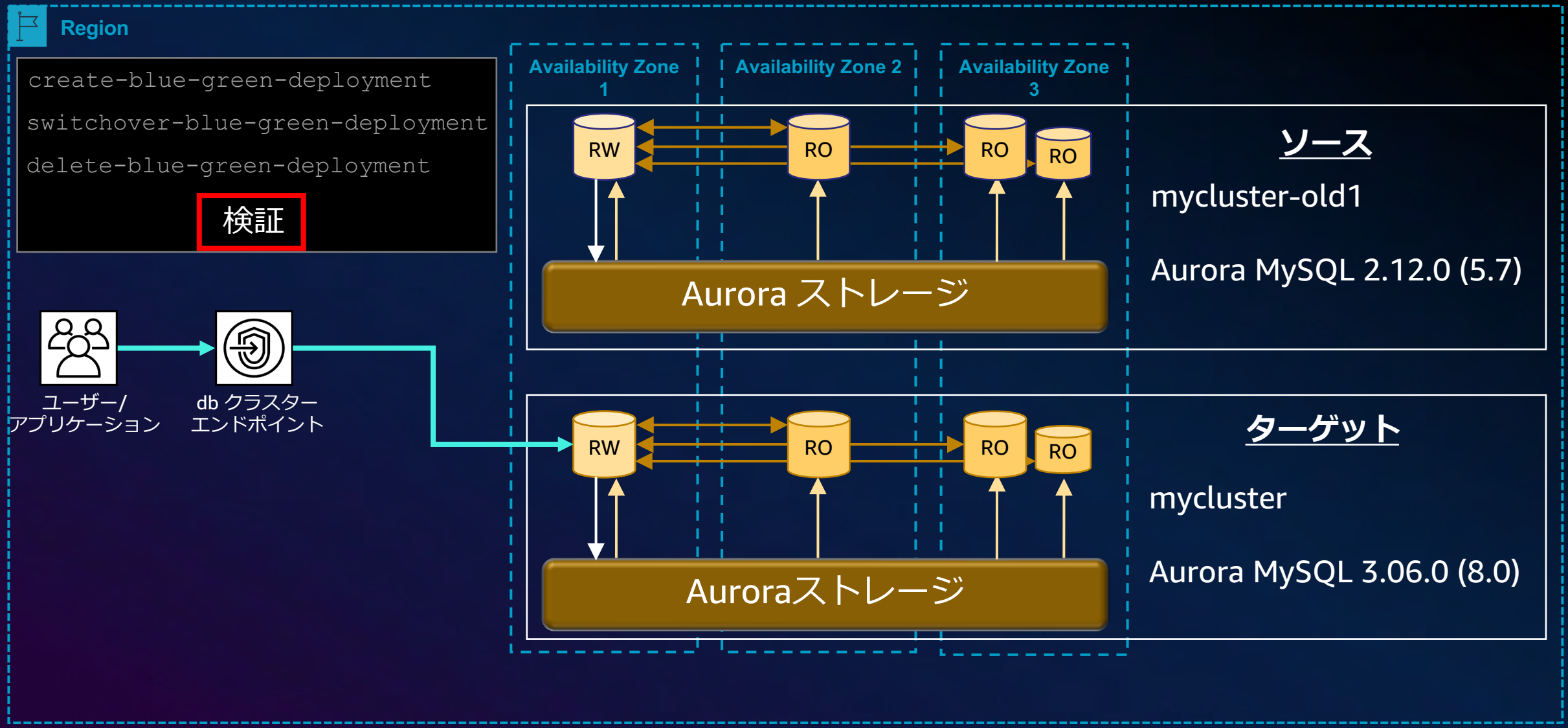
Amazon Aurora Managed Blue/Green deployment



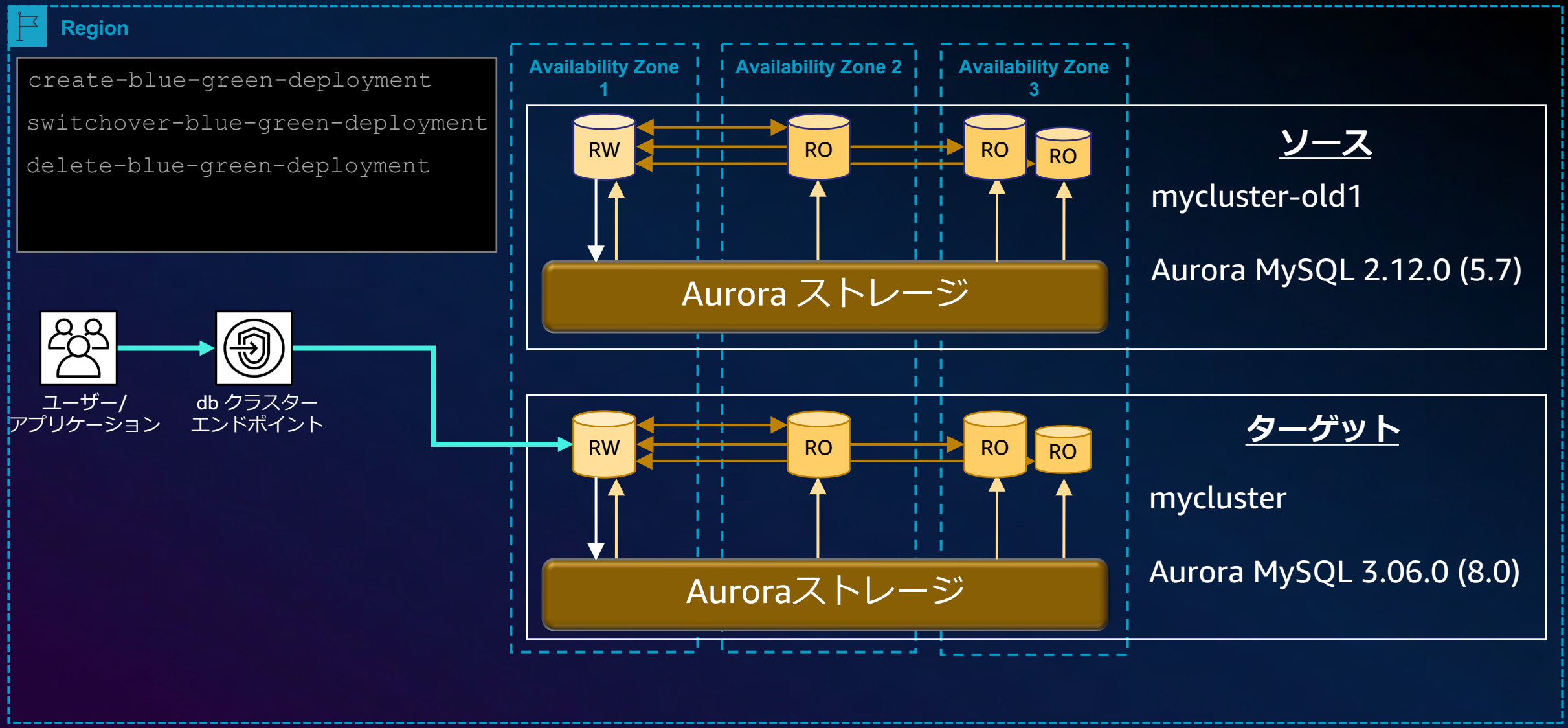
Amazon Aurora Managed Blue/Green deployment



Amazon Aurora Managed Blue/Green deployment

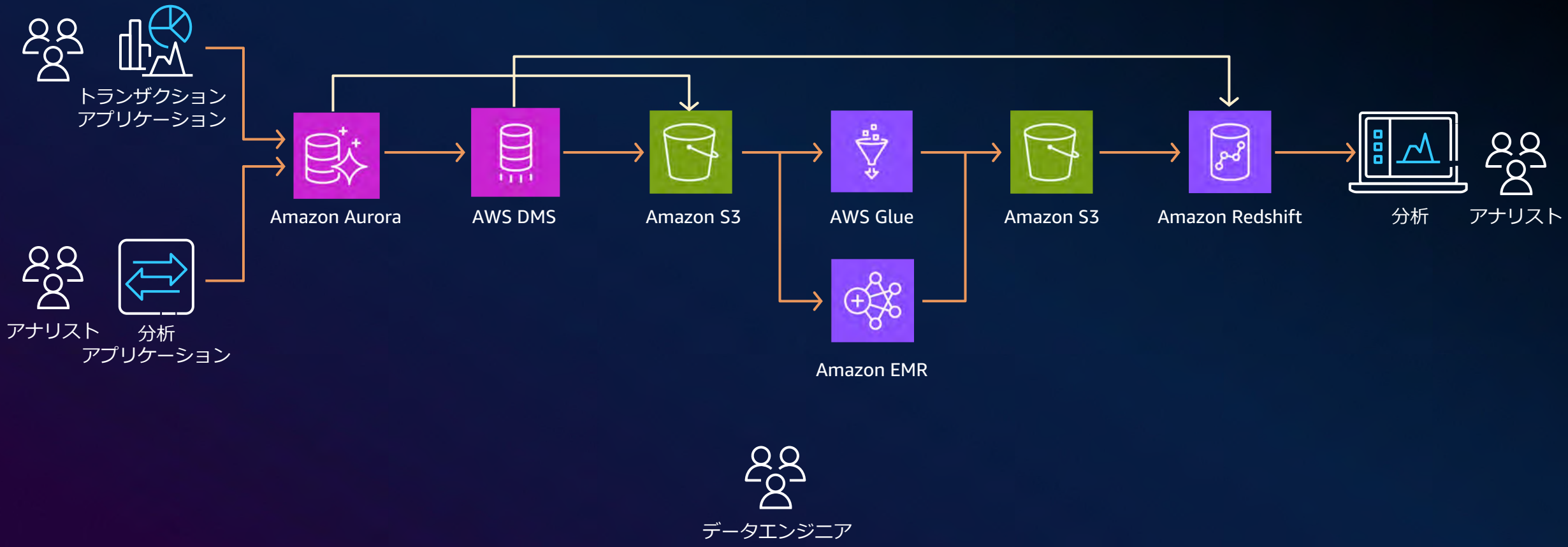


Amazon Aurora Managed Blue/Green deployment

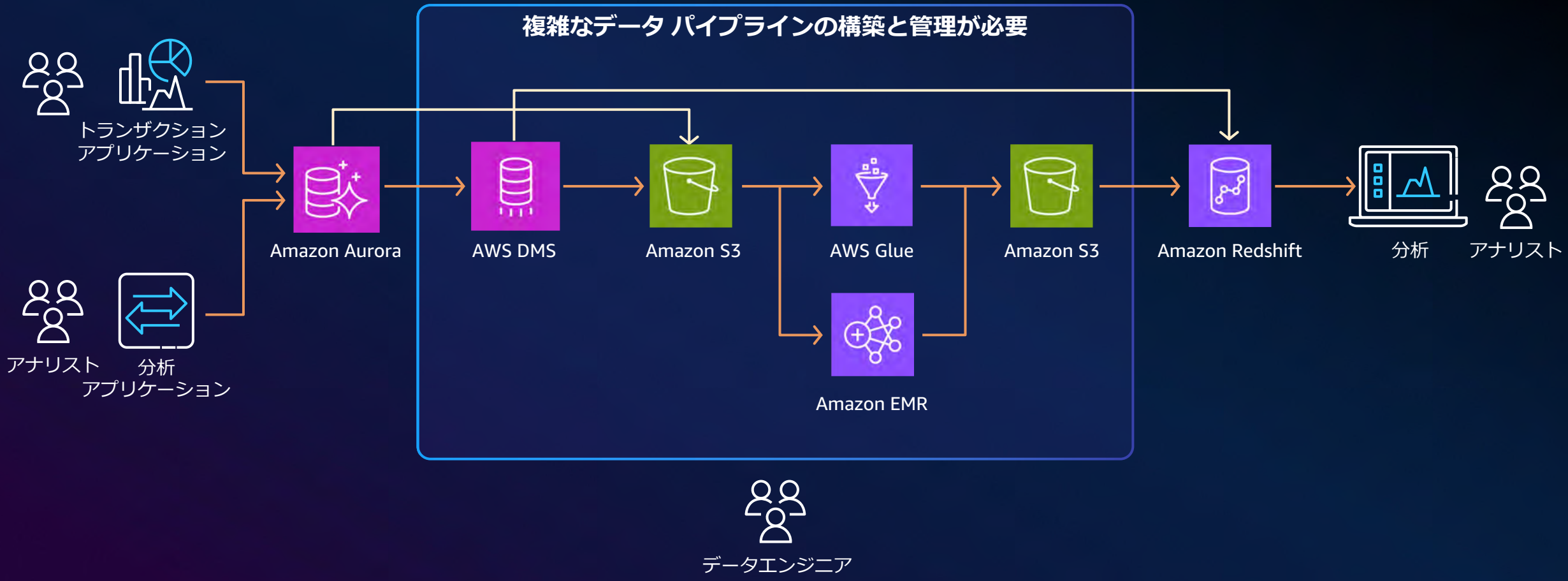


管理性 – Amazon Redshift との Amazon Aurora ゼロ ETL 統合

大規模なトランザクション分析



大規模なトランザクション分析

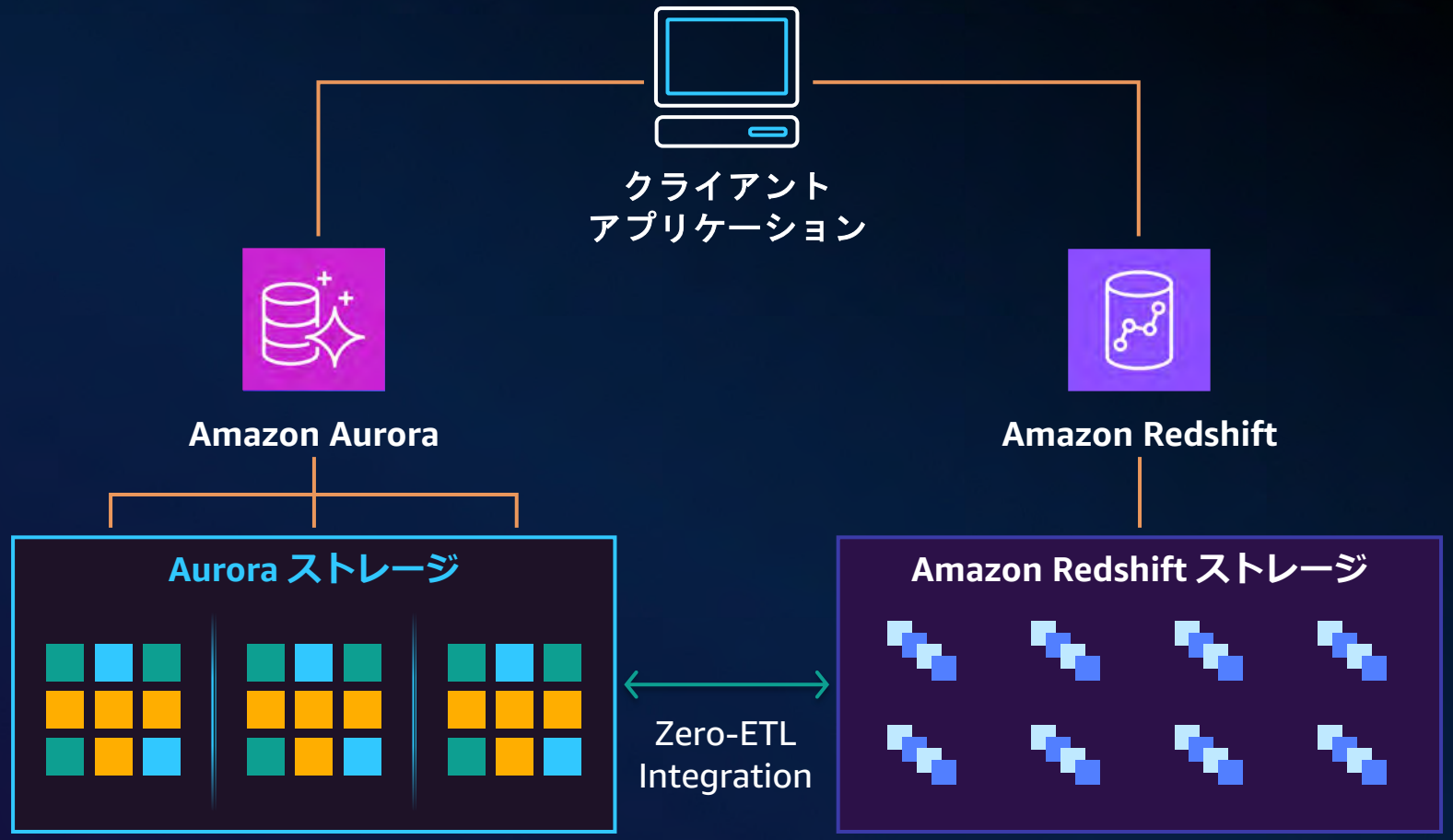


Amazon Redshift との Amazon Aurora ゼロ ETL 統合の導入

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

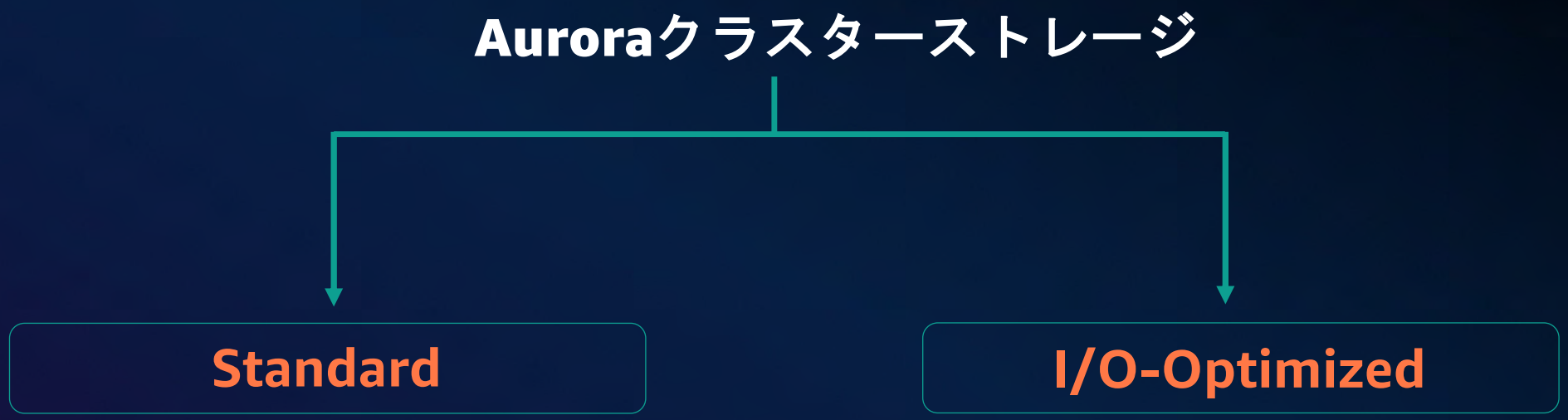
シンプルさとパフォーマンスを重視した設計

- 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



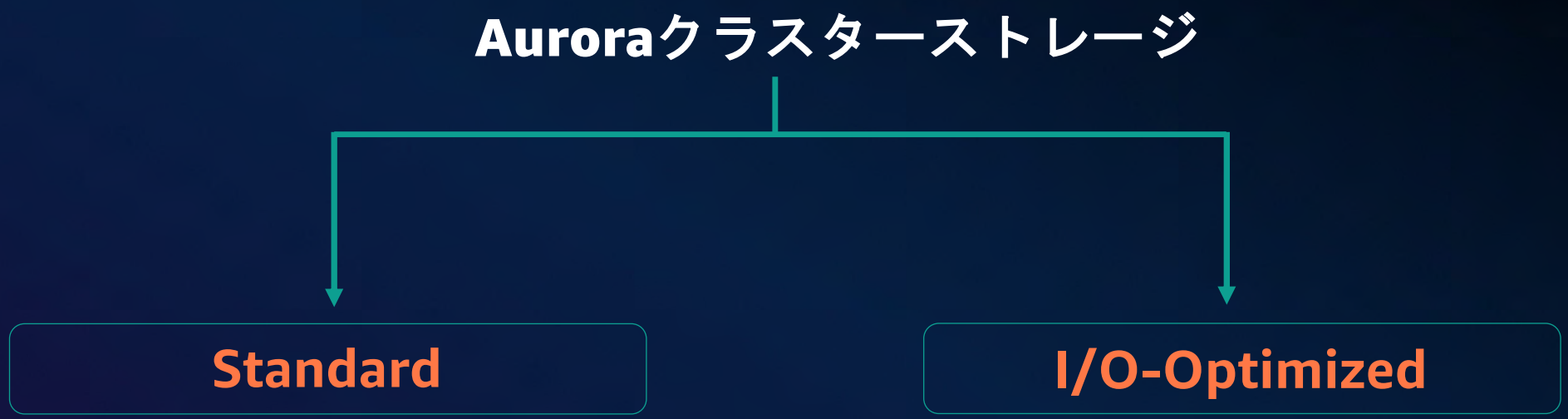
新しい Aurora ストレージ タイプ

Aurora I/O-Optimized



- 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



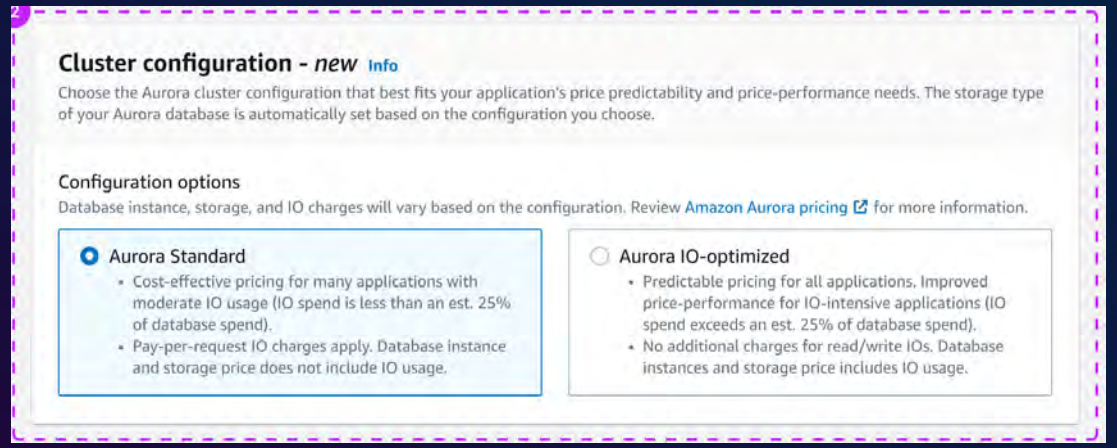
- コンピューティング (オンデマンド / RI)
- ストレージ (標準 – 従量課金制)
- I/O (従量課金制)

- 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。

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。

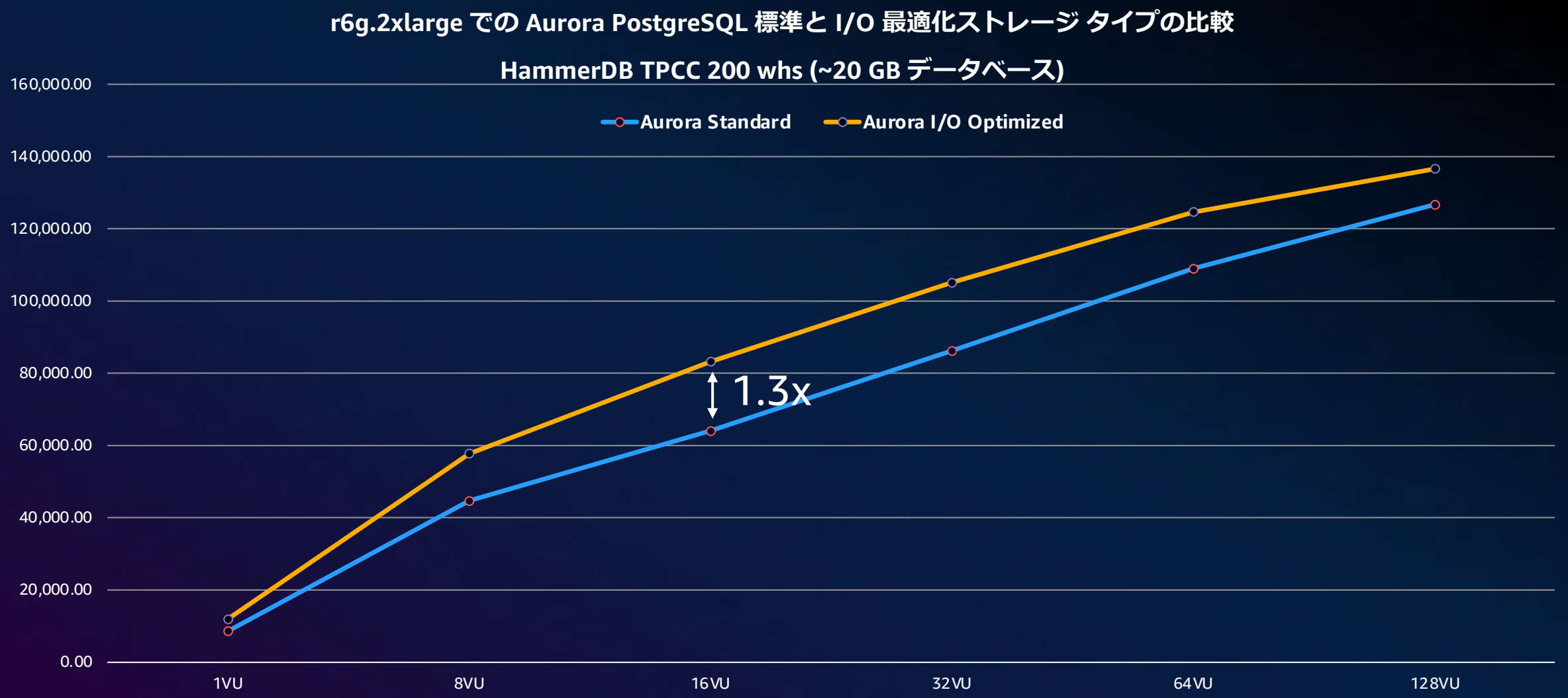
Aurora Serverless v2、オンデマンド、およびリザーブドインスタンス全体の Aurora PostgreSQL および Aurora MySQL で利用可能

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

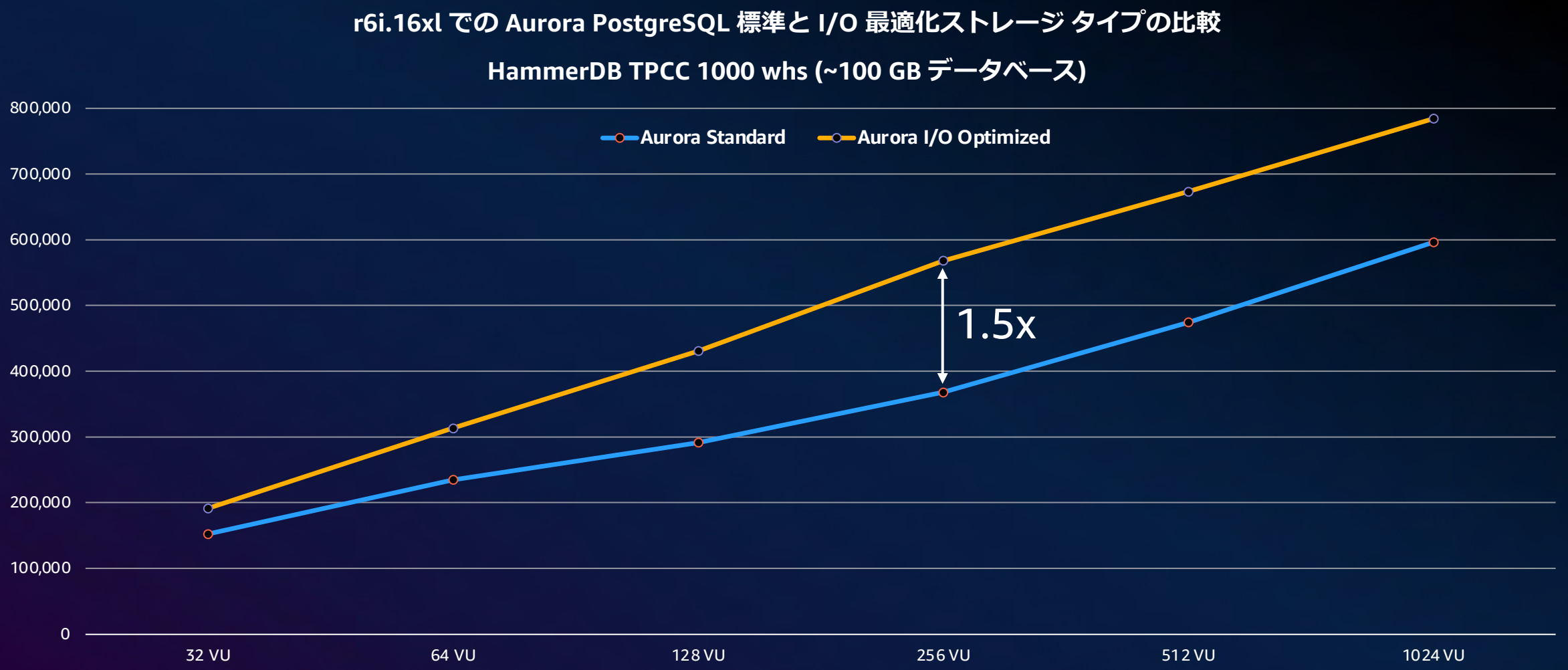
Aurora I/O-Optimized

- 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



パフォーマンス

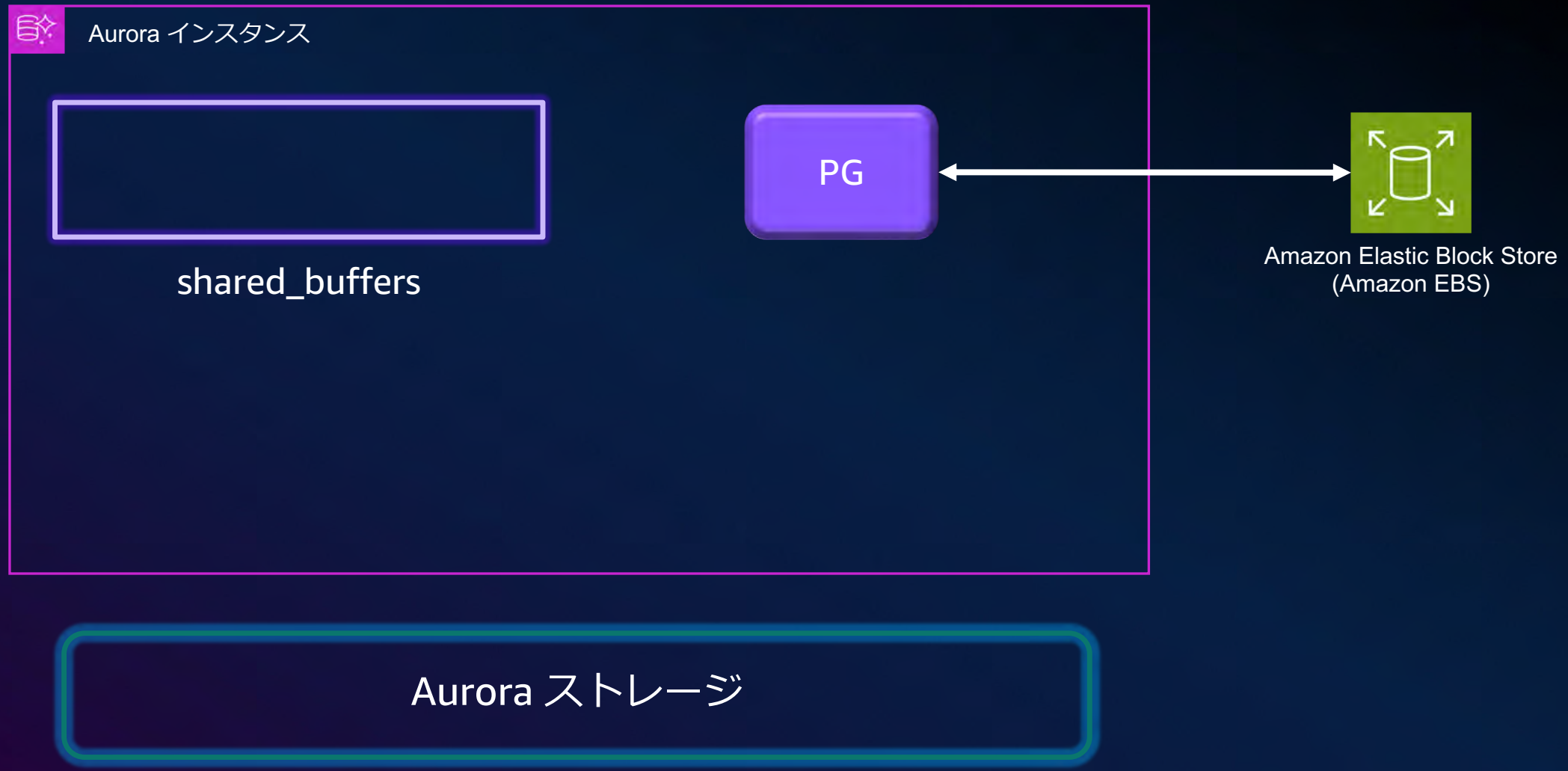


Optimized Reads

PostgreSQL

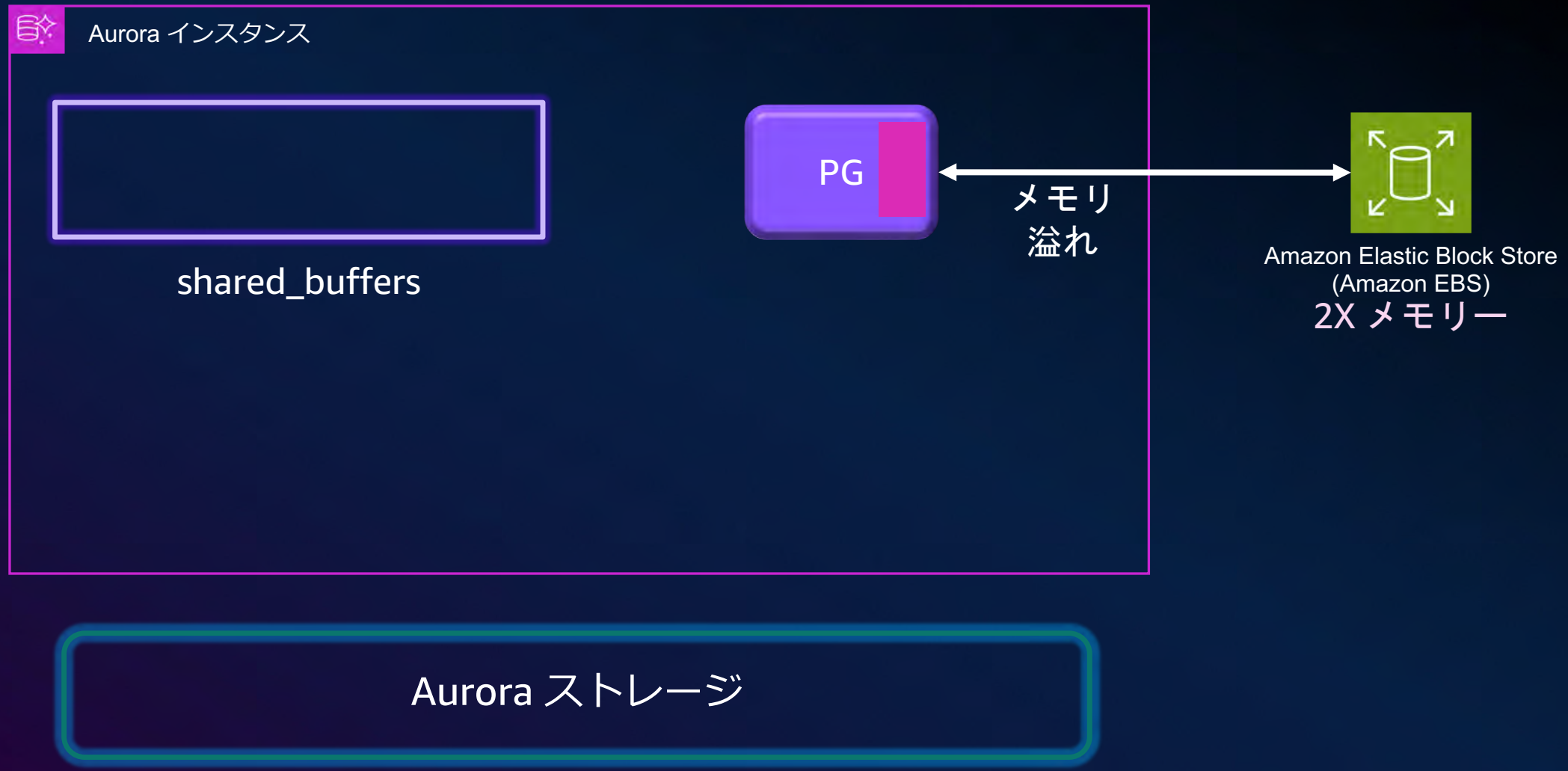
Optimized Reads - 一時オブジェクト

PostgreSQL



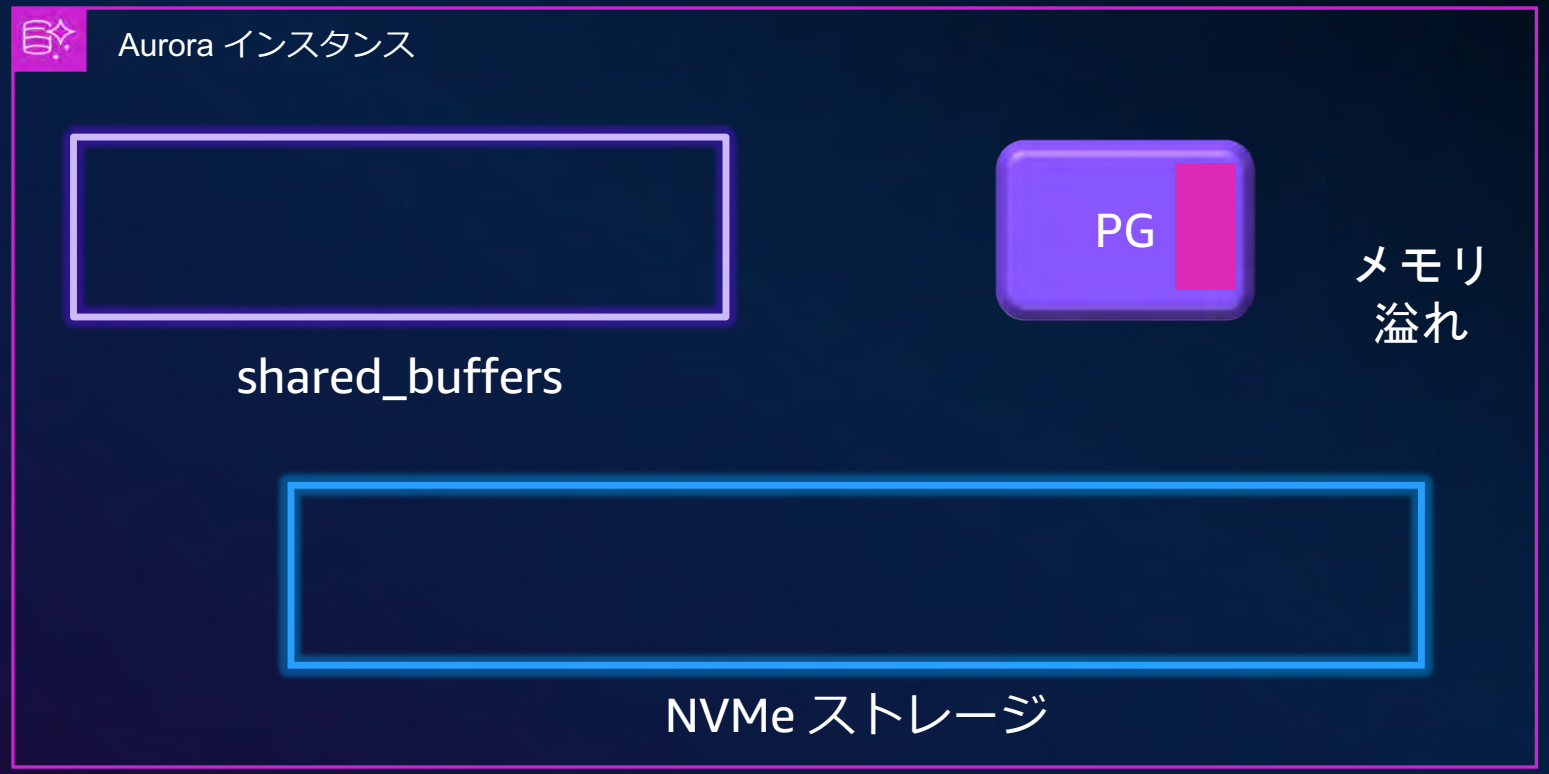
Optimized Reads - 一時オブジェクト

PostgreSQL



Optimized Reads - 一時オブジェクト

PostgreSQL



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

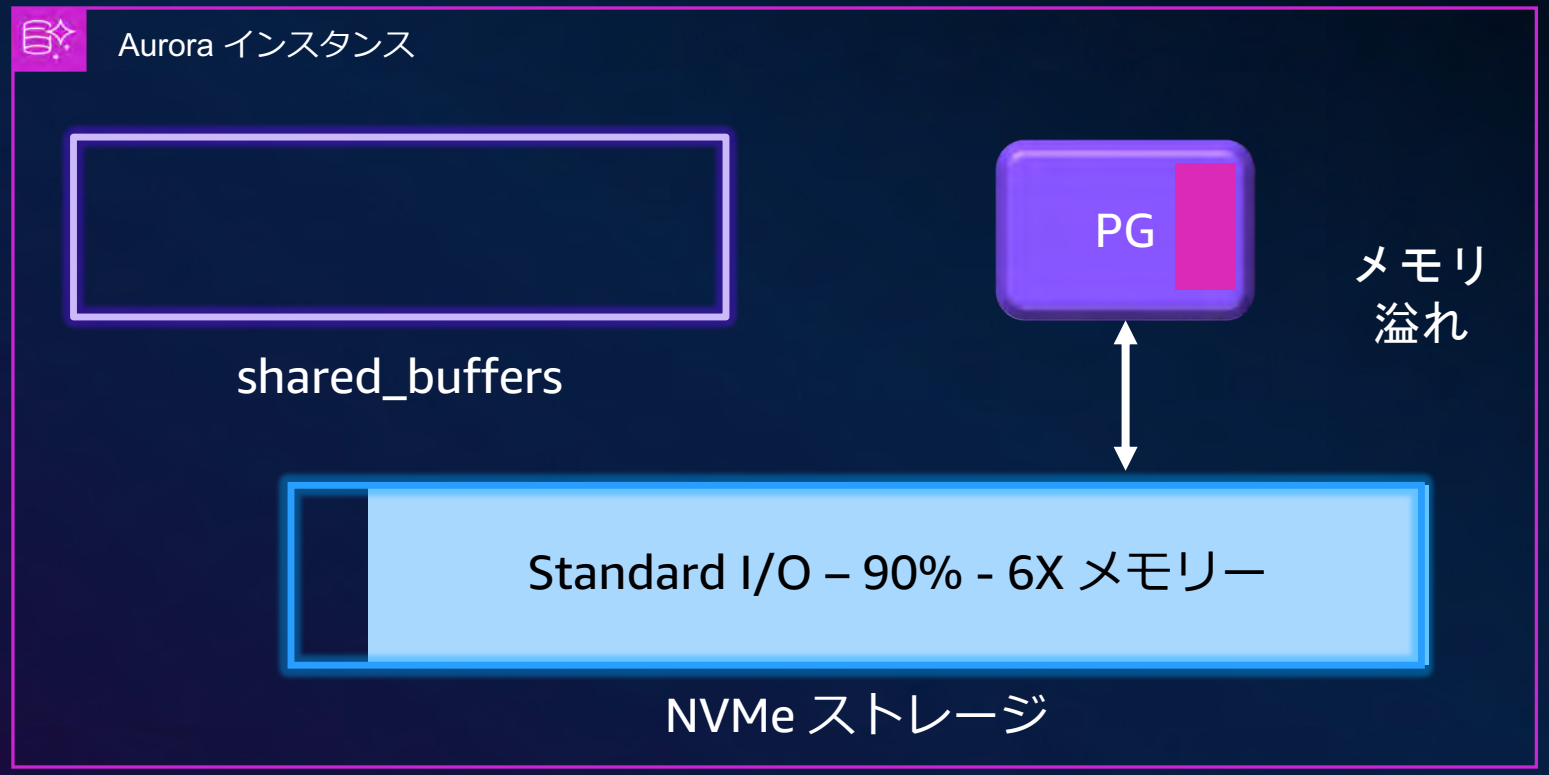
対象

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

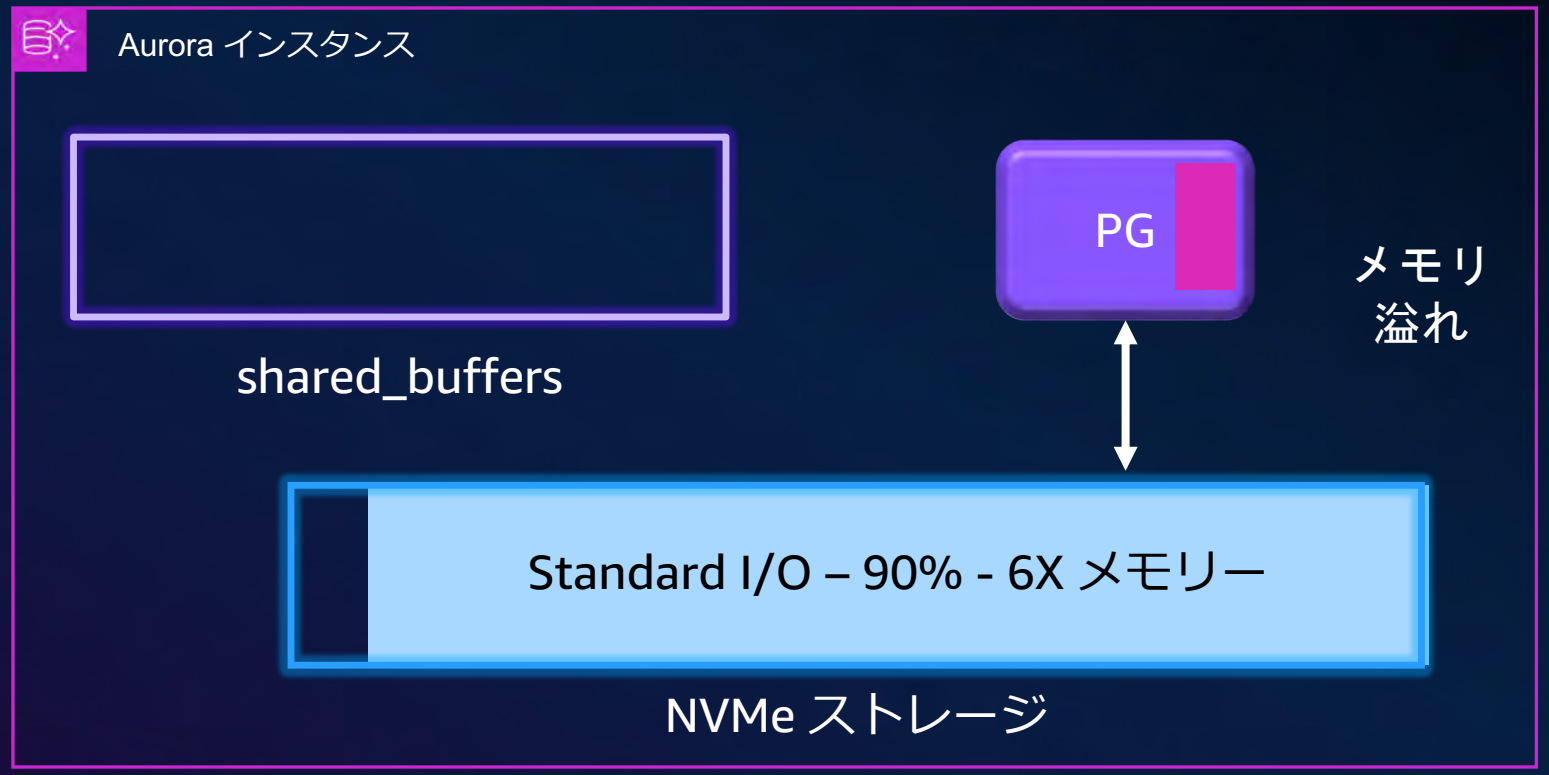
対象

db.r6gd
db.r6id

Aurora
PostgreSQL
14.9+, 15.3+

Optimized Reads - 一時オブジェクト

PostgreSQL



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

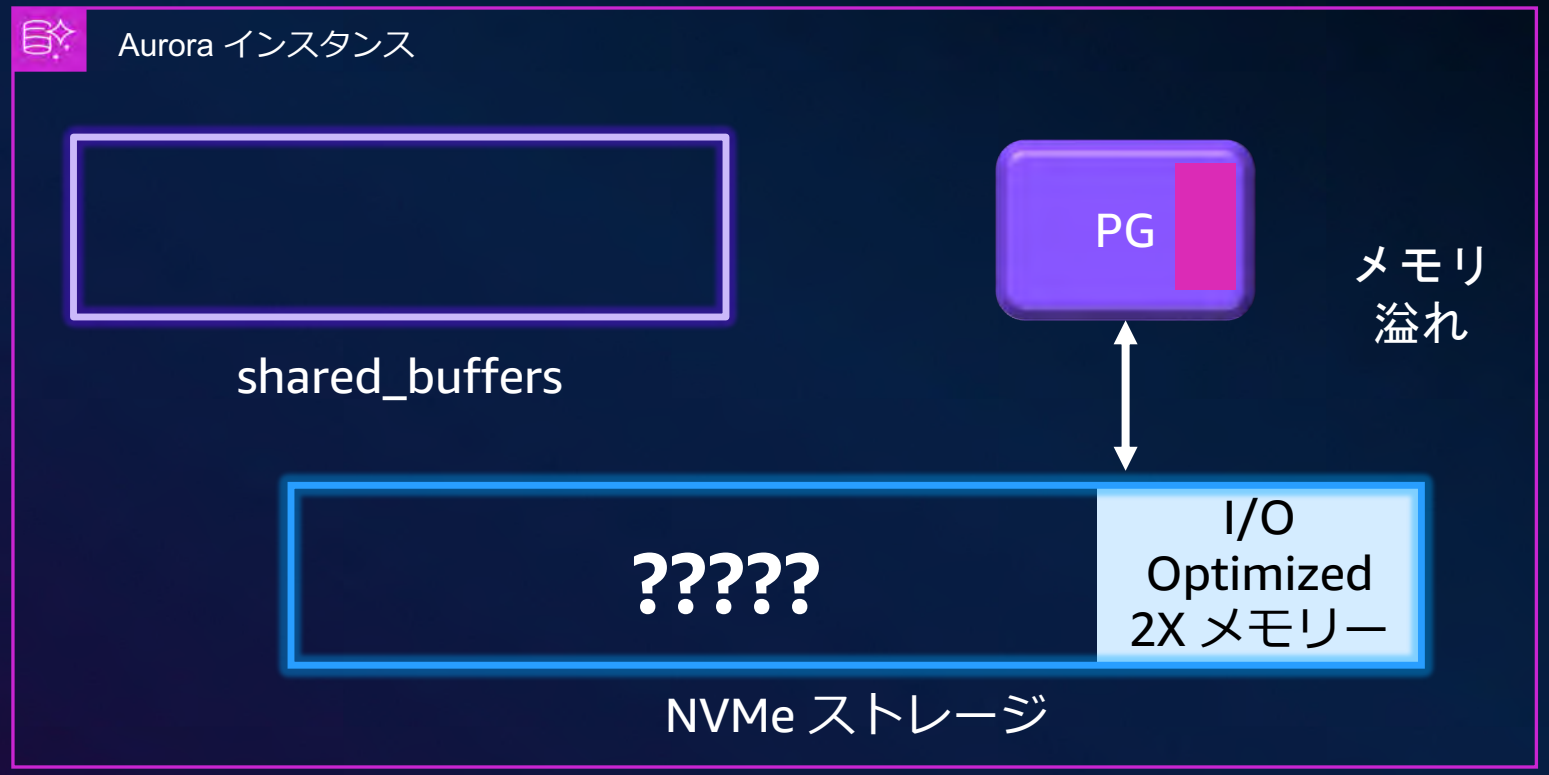
対象

db.r6gd
db.r6id

Aurora
PostgreSQL
14.9+, 15.3+

Optimized Reads - 階層型キャッシュ

PostgreSQL



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

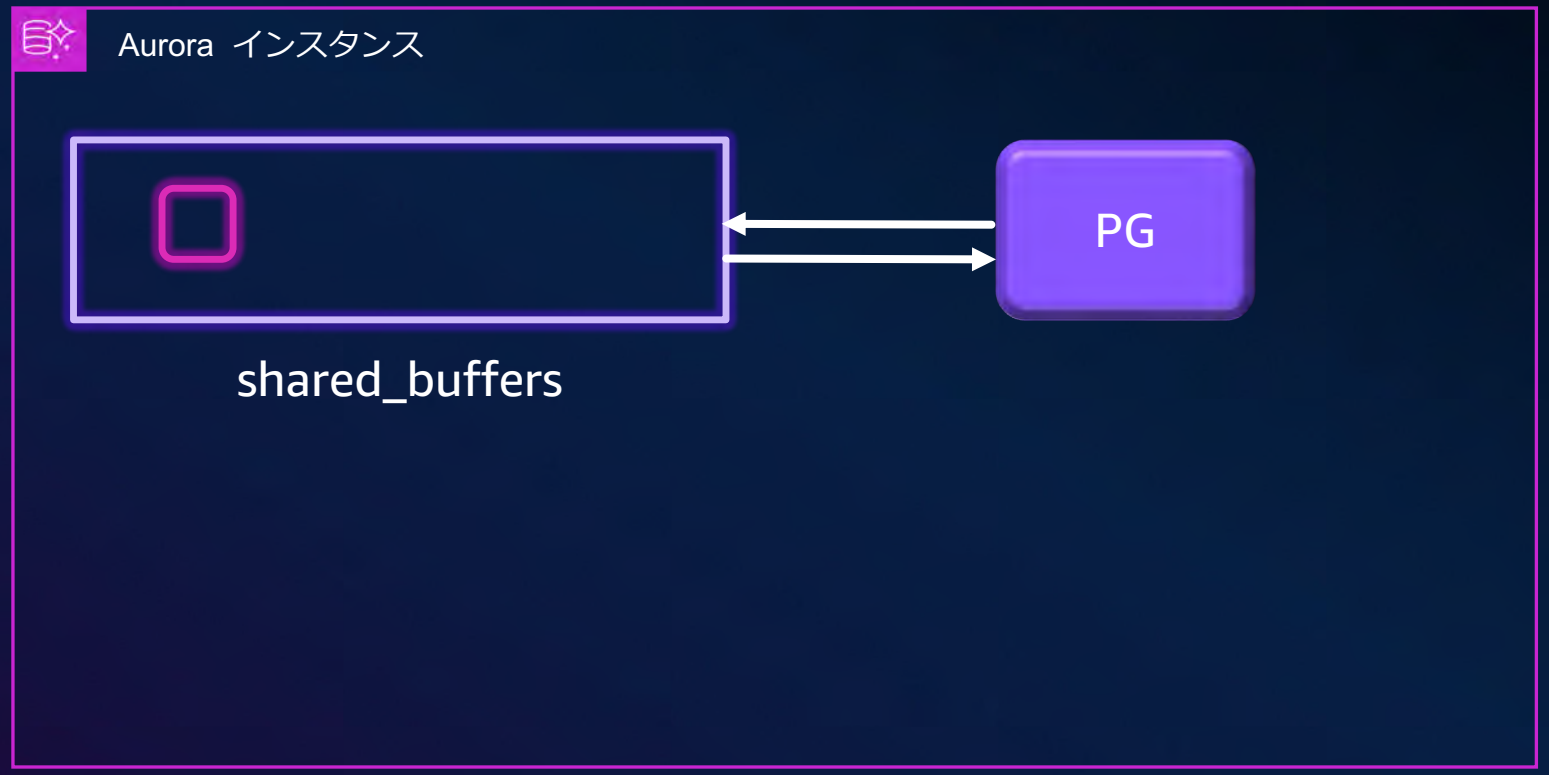
対象

db.r6gd
db.r6id

Aurora
PostgreSQL
14.9+, 15.3+

Optimized Reads - 階層型キャッシュ

PostgreSQL



I/O-Optimized 利用時

対象

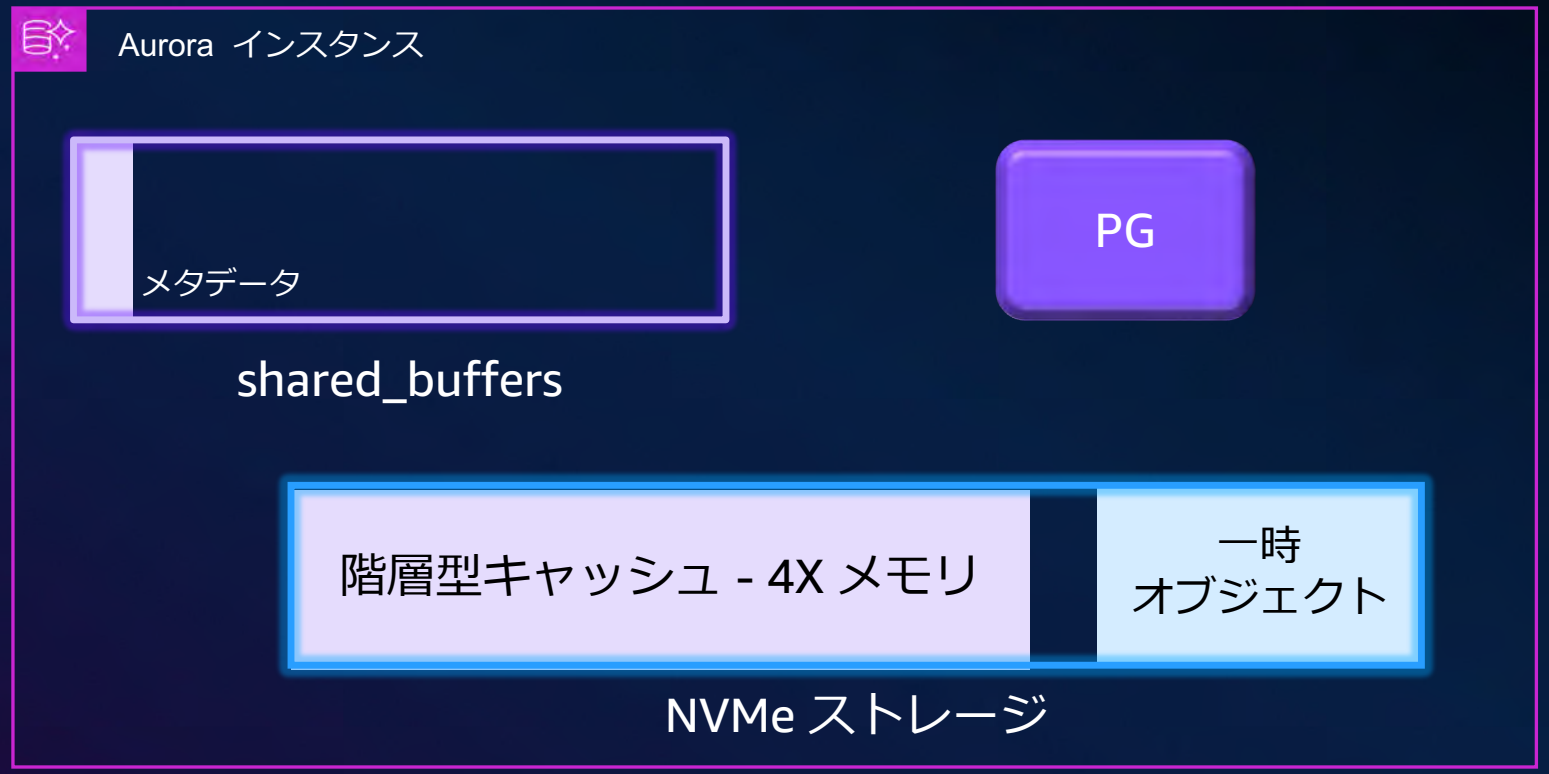
db.r6gd
db.r6id

Aurora PostgreSQL 14.9+, 15.3+

https://aws.amazon.com/blogs/database/new-amazon-aurora-optimized-reads-for-aurora-postgresql-with-up-to-8x-query-latency-improvement-for-i-o-intensive-applications/

Optimized Reads - 階層型キャッシュ

PostgreSQL



I/O-Optimized 利用時

対象

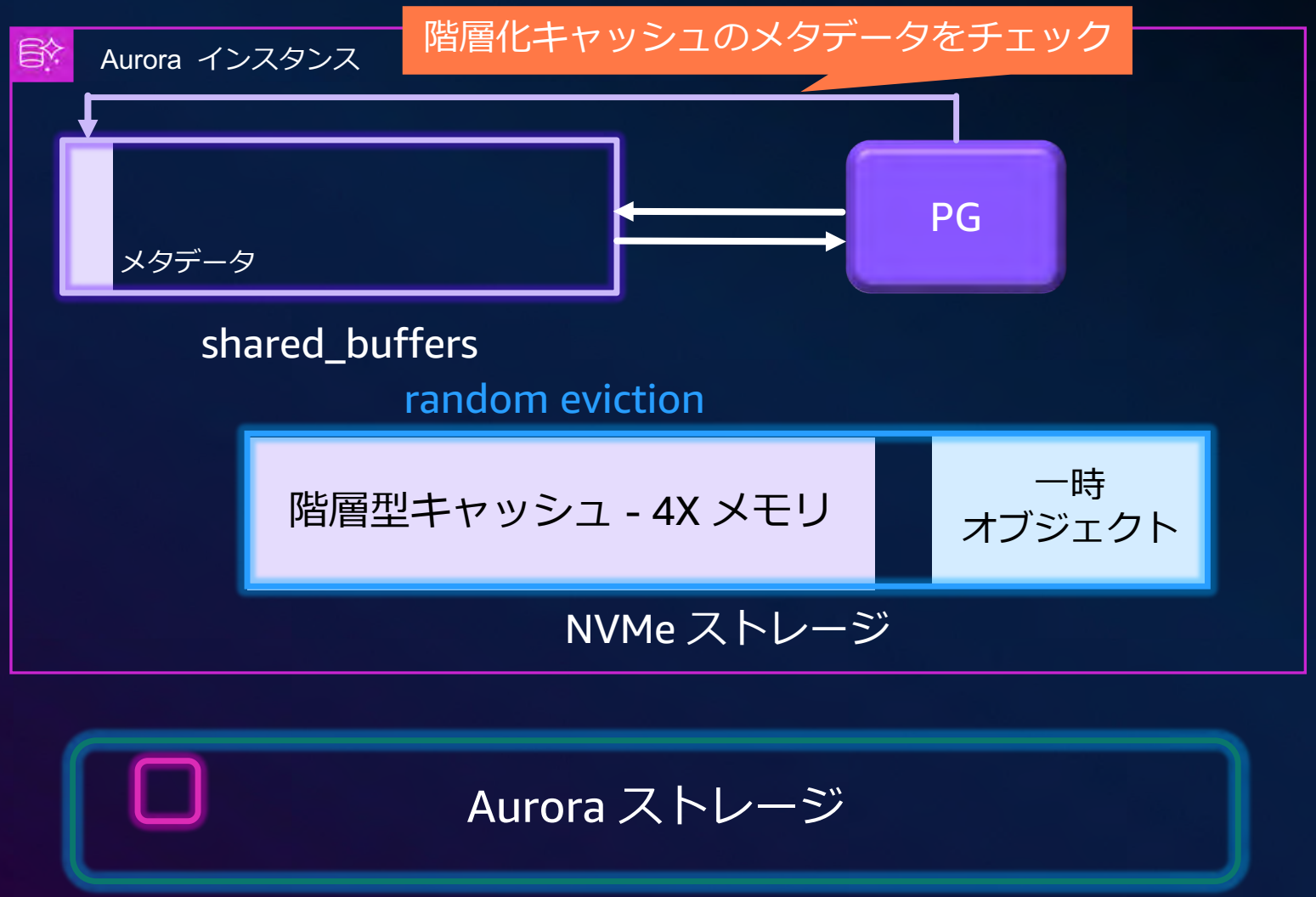
db.r6gd
db.r6id

Aurora
PostgreSQL
14.9+, 15.3+

https://aws.amazon.com/blogs/database/new-amazon-aurora-optimized-reads-for-aurora-postgresql-with-up-to-8x-query-latency-improvement-for-i-o-intensive-applications/

Optimized Reads - 階層型キャッシュ

PostgreSQL



I/O-Optimized 利用時

対象

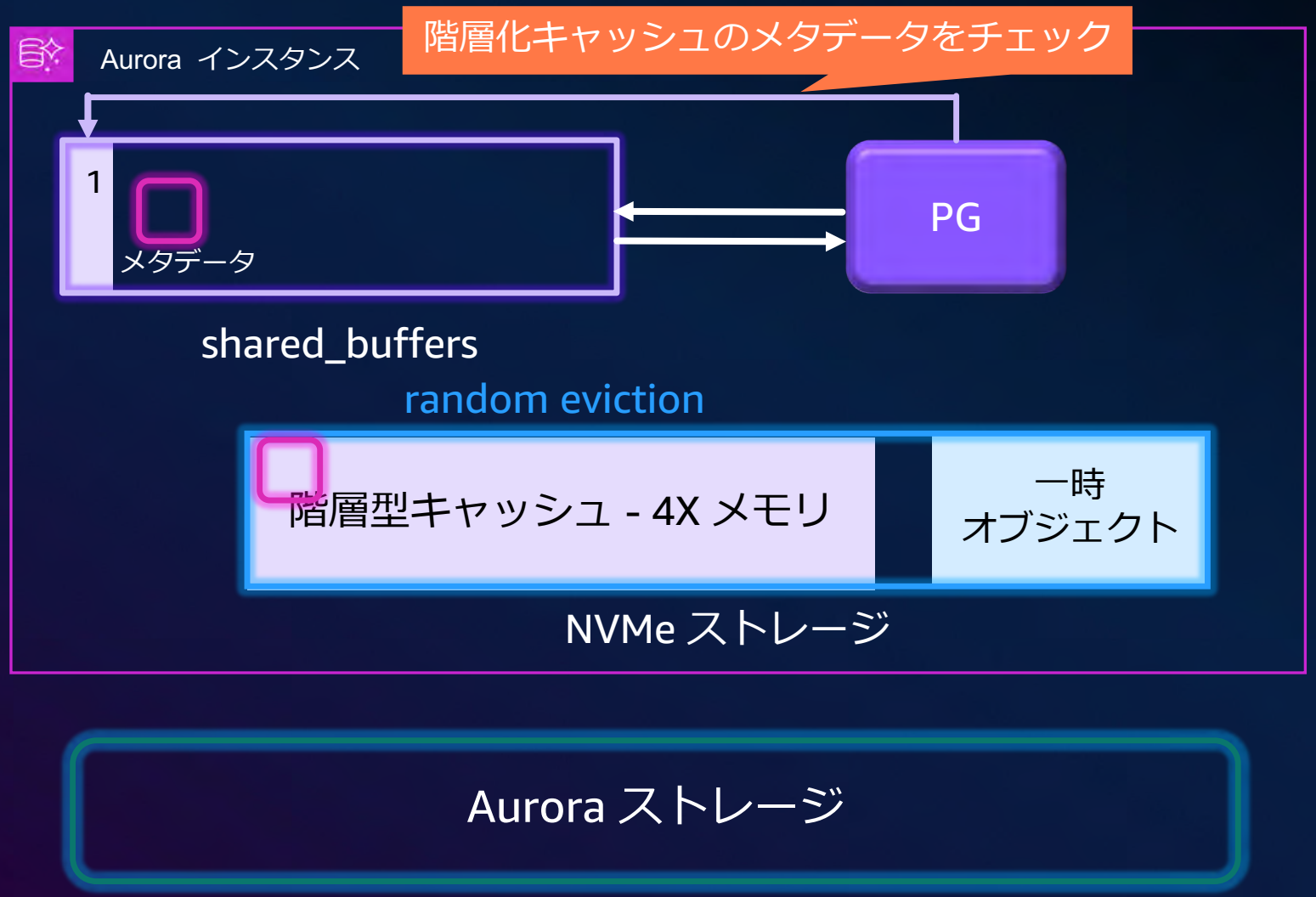
db.r6gd
db.r6id

Aurora
PostgreSQL
14.9+, 15.3+

https://aws.amazon.com/blogs/database/new-amazon-aurora-optimized-reads-for-aurora-postgresql-with-up-to-8x-query-latency-improvement-for-i-o-intensive-applications/

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+

https://aws.amazon.com/blogs/database/new-amazon-aurora-optimized-reads-for-aurora-postgresql-with-up-to-8x-query-latency-improvement-for-i-o-intensive-applications/

pgvector 拡張機能

PostgreSQL

pgvector とは

PostgreSQL

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

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

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

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);
```

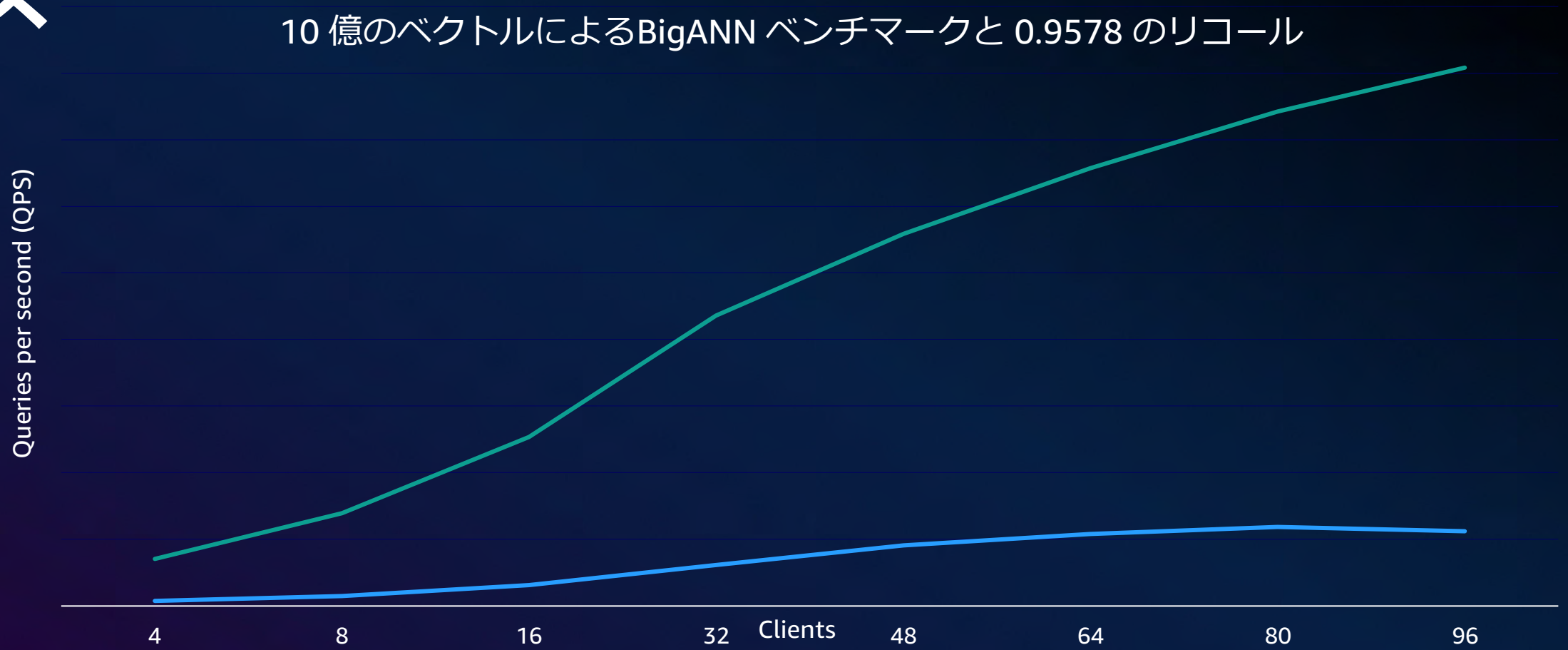
インデックス作成

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

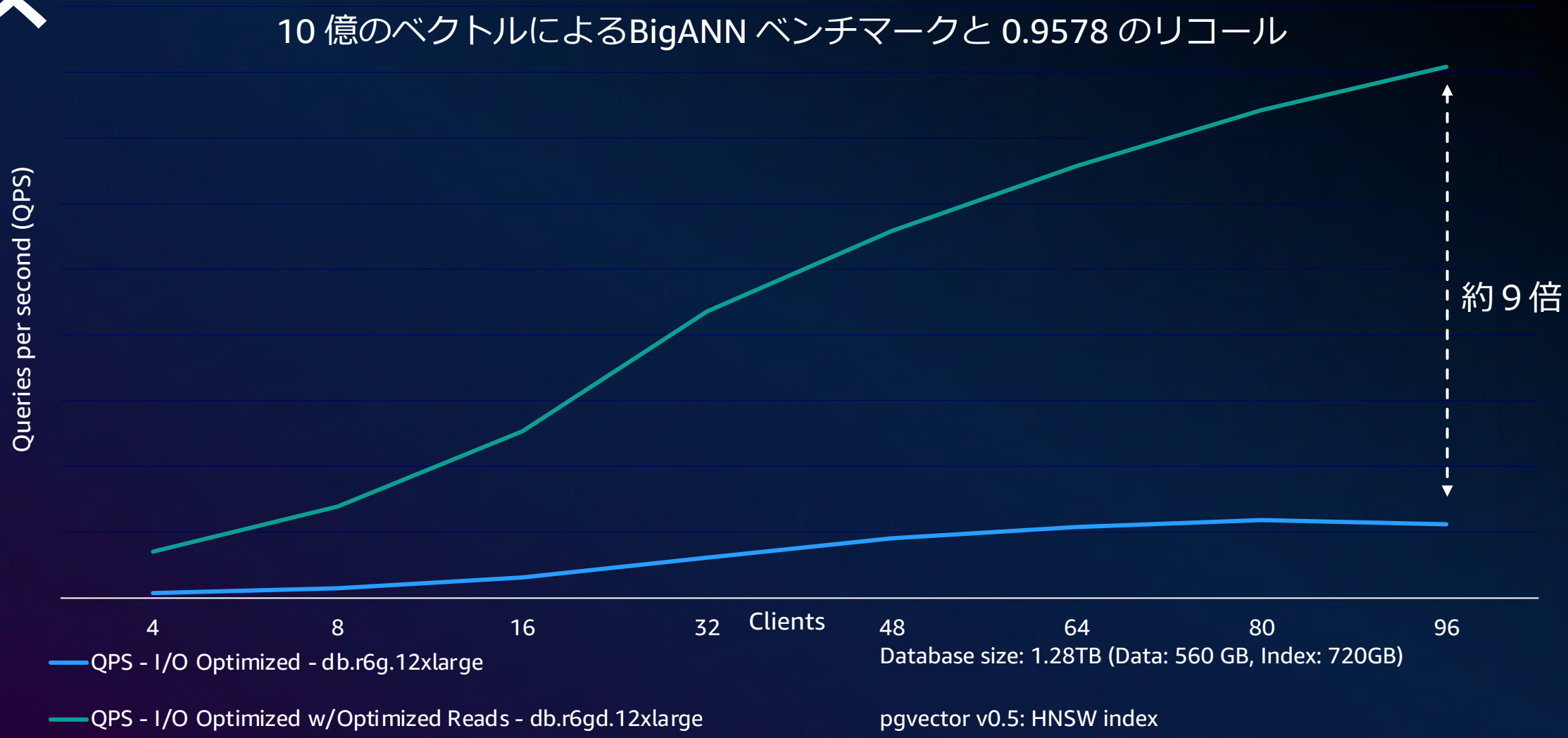
検索

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

階層型キャッシュによるベクトルのパフォーマンス



階層型キャッシュによるベクトルのパフォーマンス



Thank you!

Mari Tsukamoto

tsukamar@amazon.co.jp