JAPAN | 2024

aws SUMMIT

AWS-40

Amazon Aurora Limitless Database 内部アーキテクチャ詳解 ～ スケーラビリティと高可用性の秘密 ～

星野 豊
アマゾン ウェブ サービス ジャパン合同会社
Principal Database Solutions Architect

Managed Database

Managed サービスを活用してイノベーションを

Database management

モニタリング

High availability

アップデート

クエリの作成

コンプライアンス

スケーリング

メンテナンス

クエリ最適化

セキュリティ

スキーマデザイン

パッチ適用

Backup & Recovery

Managed サービスを活用してイノベーションを

Database management

クエリの作成

スキーマデザイン　クエリ最適化

Your responsibility

High availability

コンプライアンス

セキュリティ

モニタリング

パッチ適用

アップデート

スケーリング

メンテナンス

Backup & Recovery

AWS responsibility

Amazon Aurora core values

Performance & scalability

Availability & durability

Security

Manageability at scale

Operational Analytics

generative AI

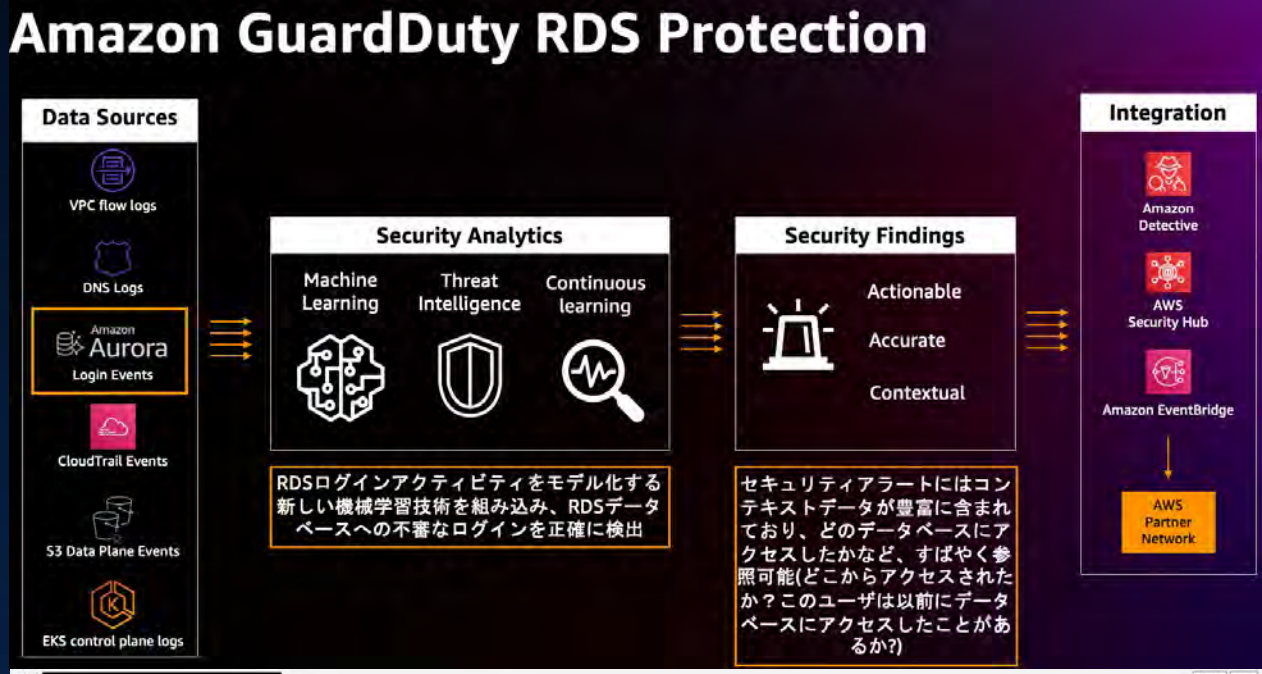
Ease of migration

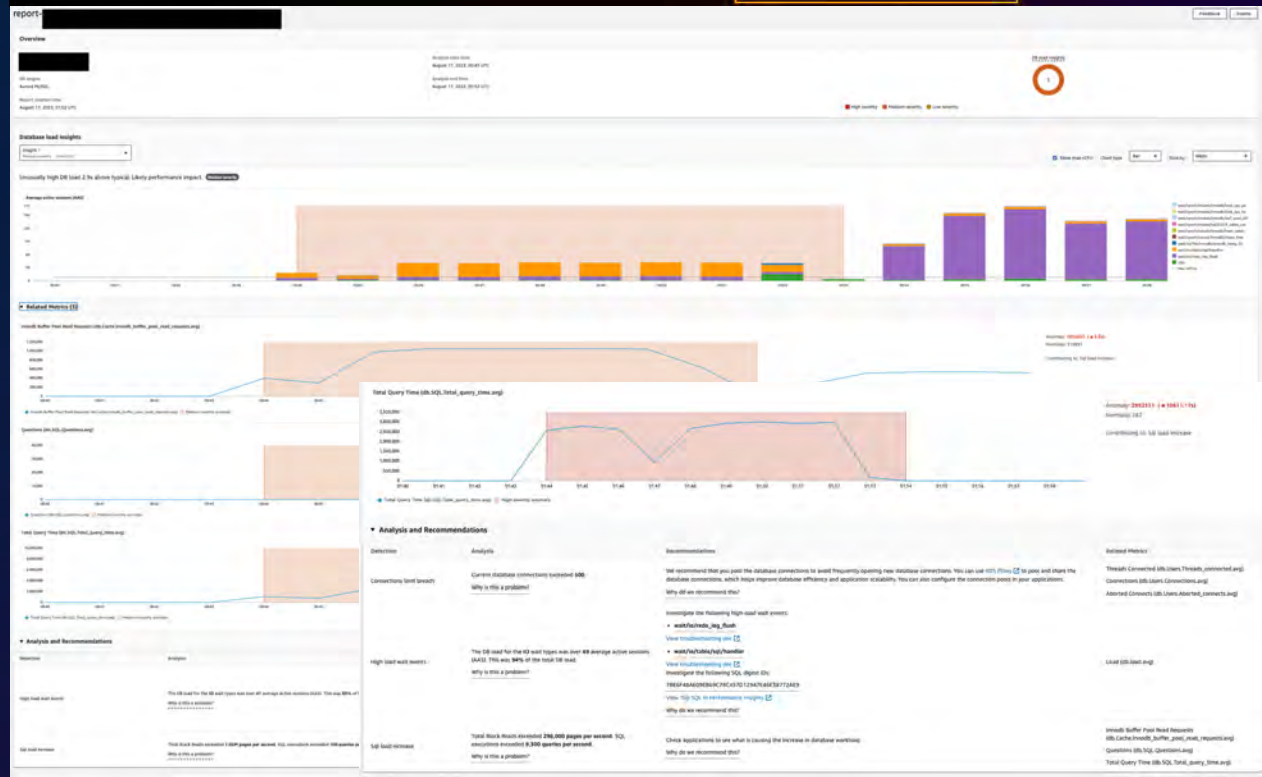
Recent innovation

新機能・機能改善を継続的にリリース

- **Amazon Bedrock integration**
- **Amazon GuardDuty RDS Protection**
- **Amazon DevOps Guru Proactive Insights**
- **On-demand Database performance analyzing**
- **I/O Optimized storage**
- **Advanced JDBC Wrapper Driver for Amazon Aurora**
- **Amazon Aurora MySQL database restart time optimizations**

性能向上はもちろん、運用性・セキュリティ、可用性の向上を大きな柱として開発を続けています

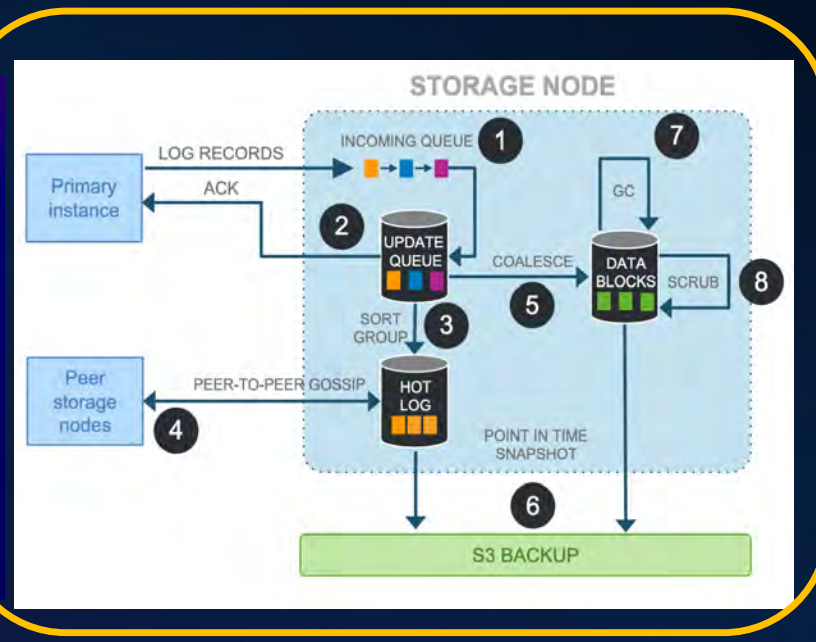




New Technology for Databases in the Cloud

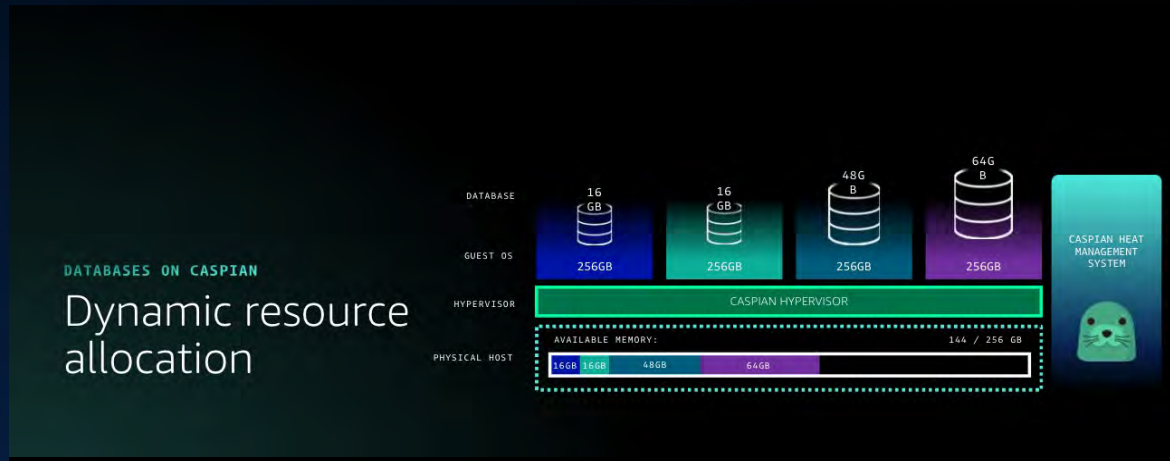
AWS database innovations

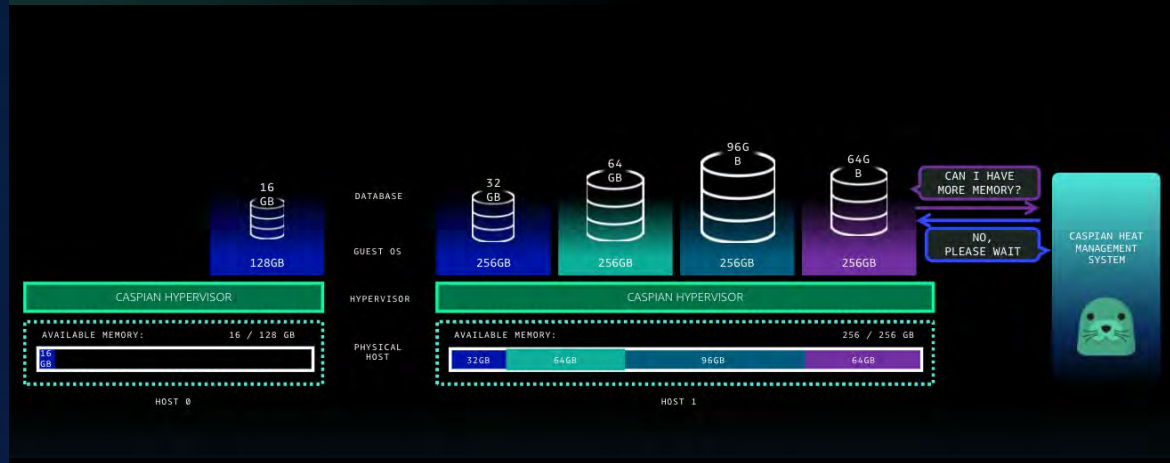
- Amazon Aurora 専用のストレージ **Grover** を開発
  - The log is the Database
  - Grover はログをストアするだけでなく処理も行うことで、多くの機能や性能向上を実現
  - Amazon Aurora では、 Disk I/O を 80% 削減

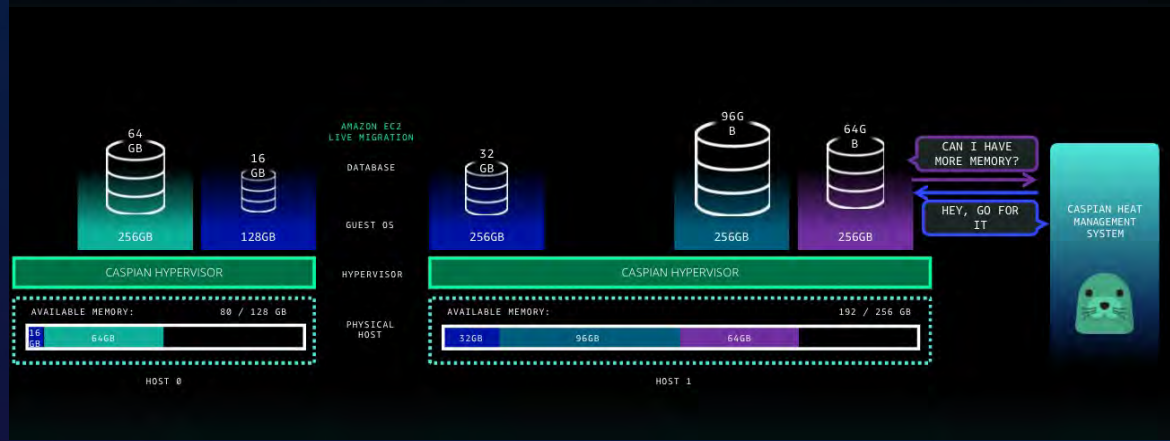


AWS Database innovations

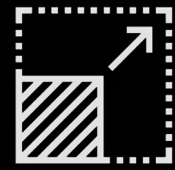
- Serverless Database のために **Caspian** を開発
  - ハイパーバイザ、ヒートマネジメントを行うコンポーネント
  - オーバサブスクリプションでデータベースインスタンスを配置し、リクエストされたリソースを **ms** 単位で割り当てる
  - スケール後のリソースが確保出来ない場合は Live migration を実施



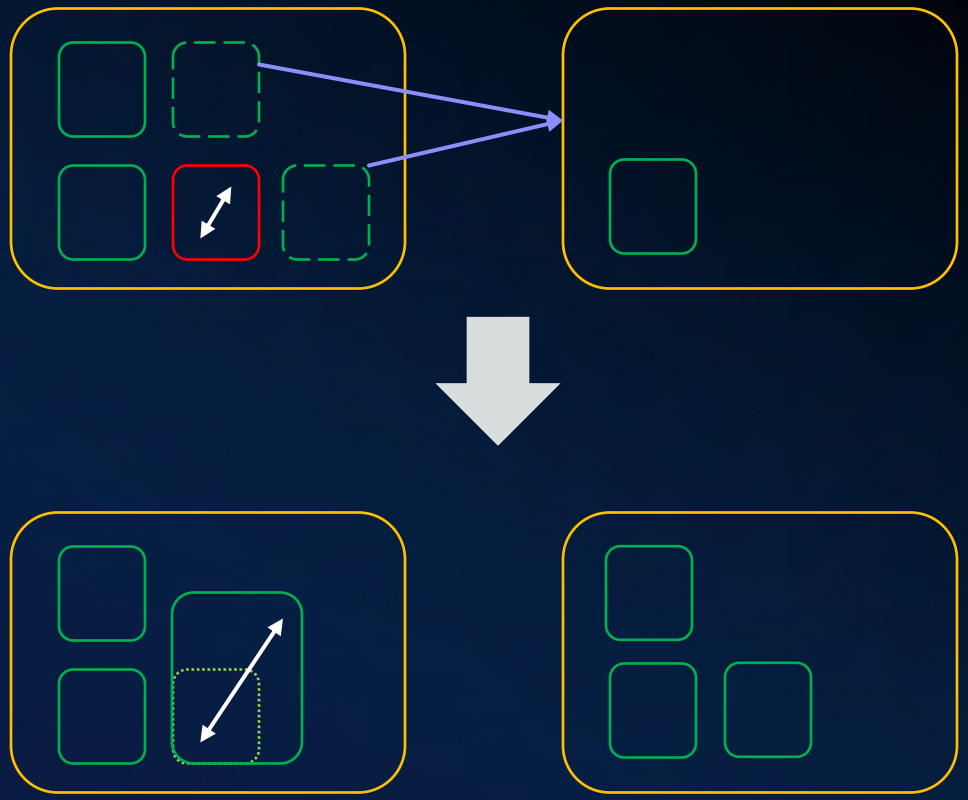




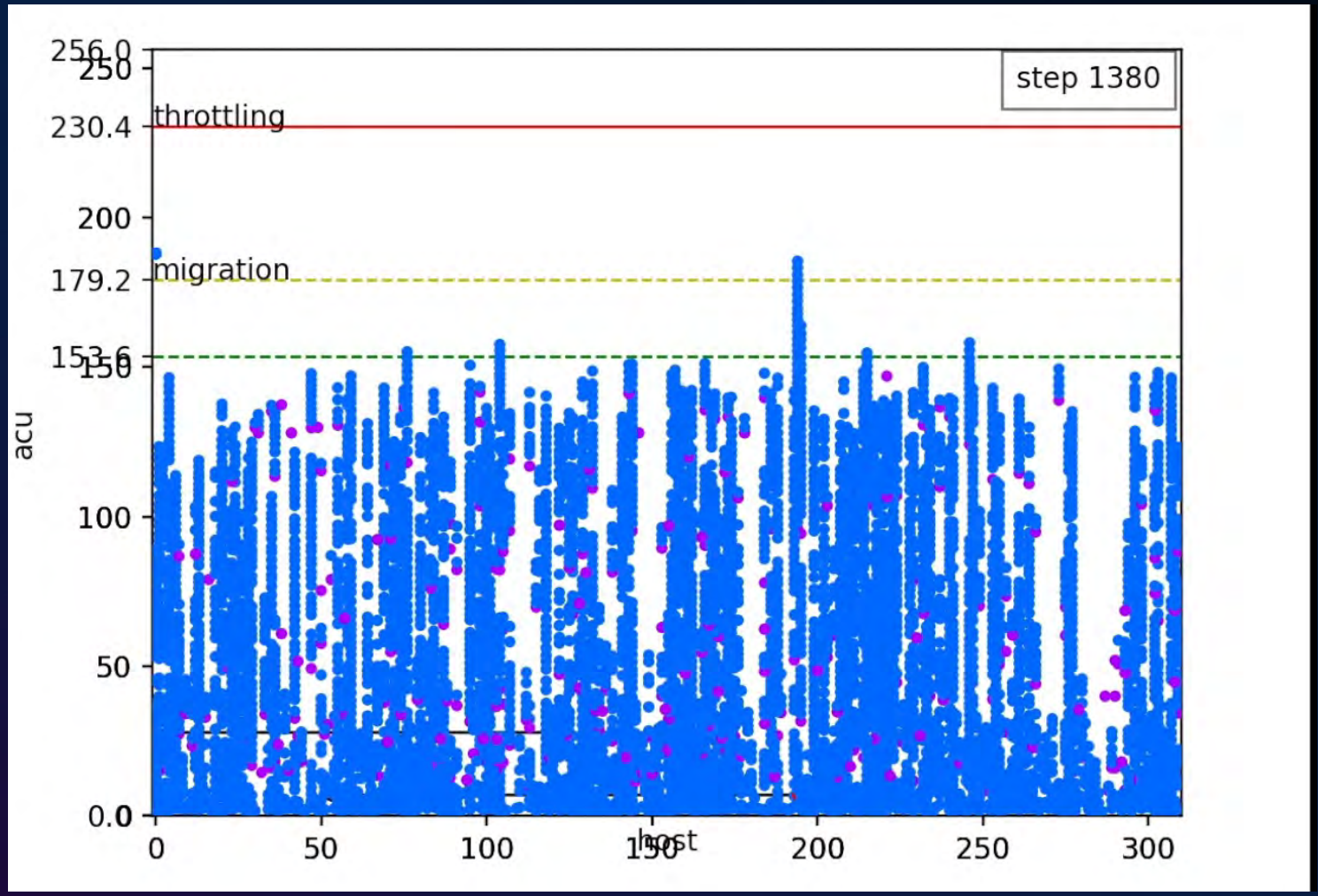
データベースを拡張するのに十分な容量はあるのか？



- リージョン単位で capacity planning
- heat management のためにコンピュートフリートを継続的に監視
- 状態(バッファプール、コネクションなど)を保持したまま、インスタンスをバックグラウンドで移動



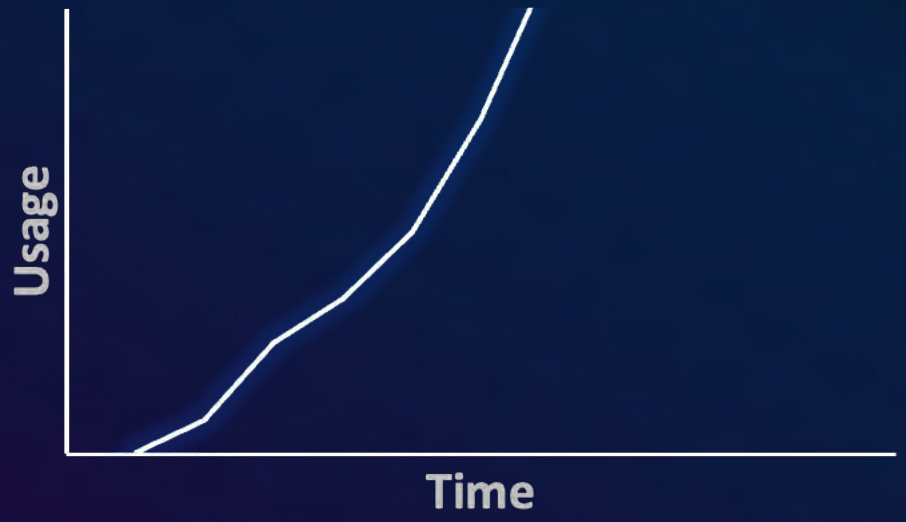
How is heat managed at scale?



Performance & scalability

Amazon Aurora Limitless Database (Limited Preview)

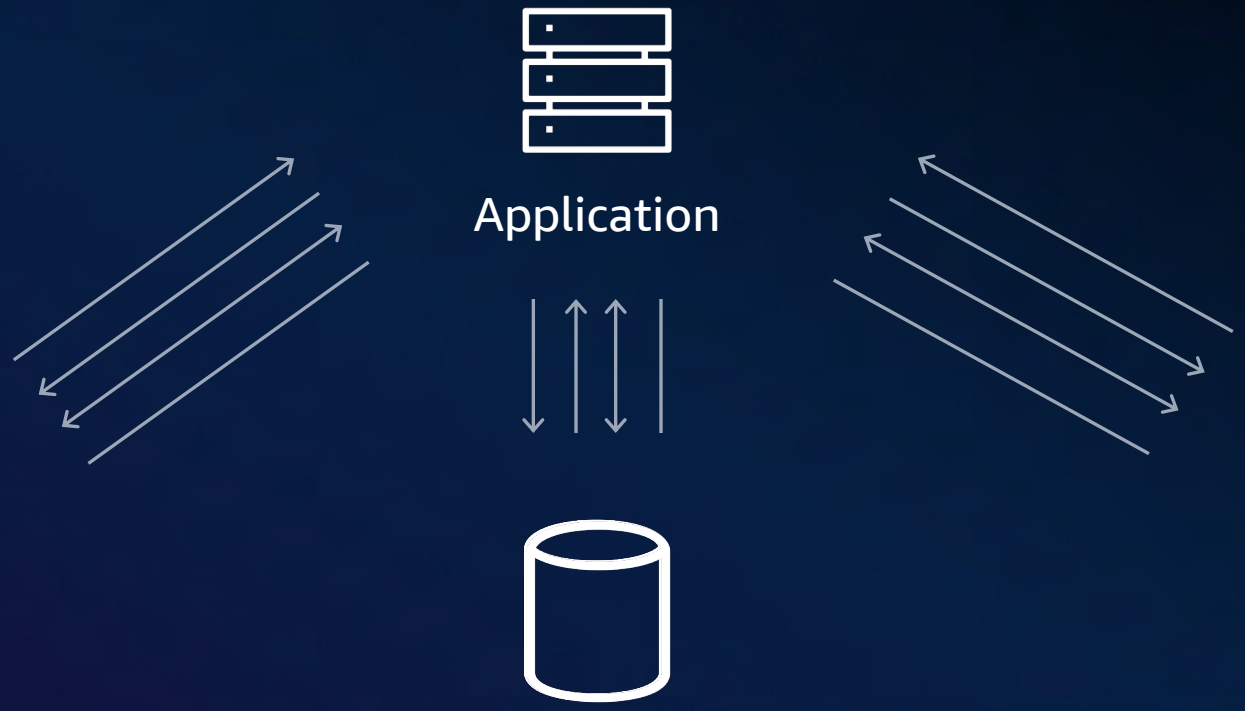
Scale databases



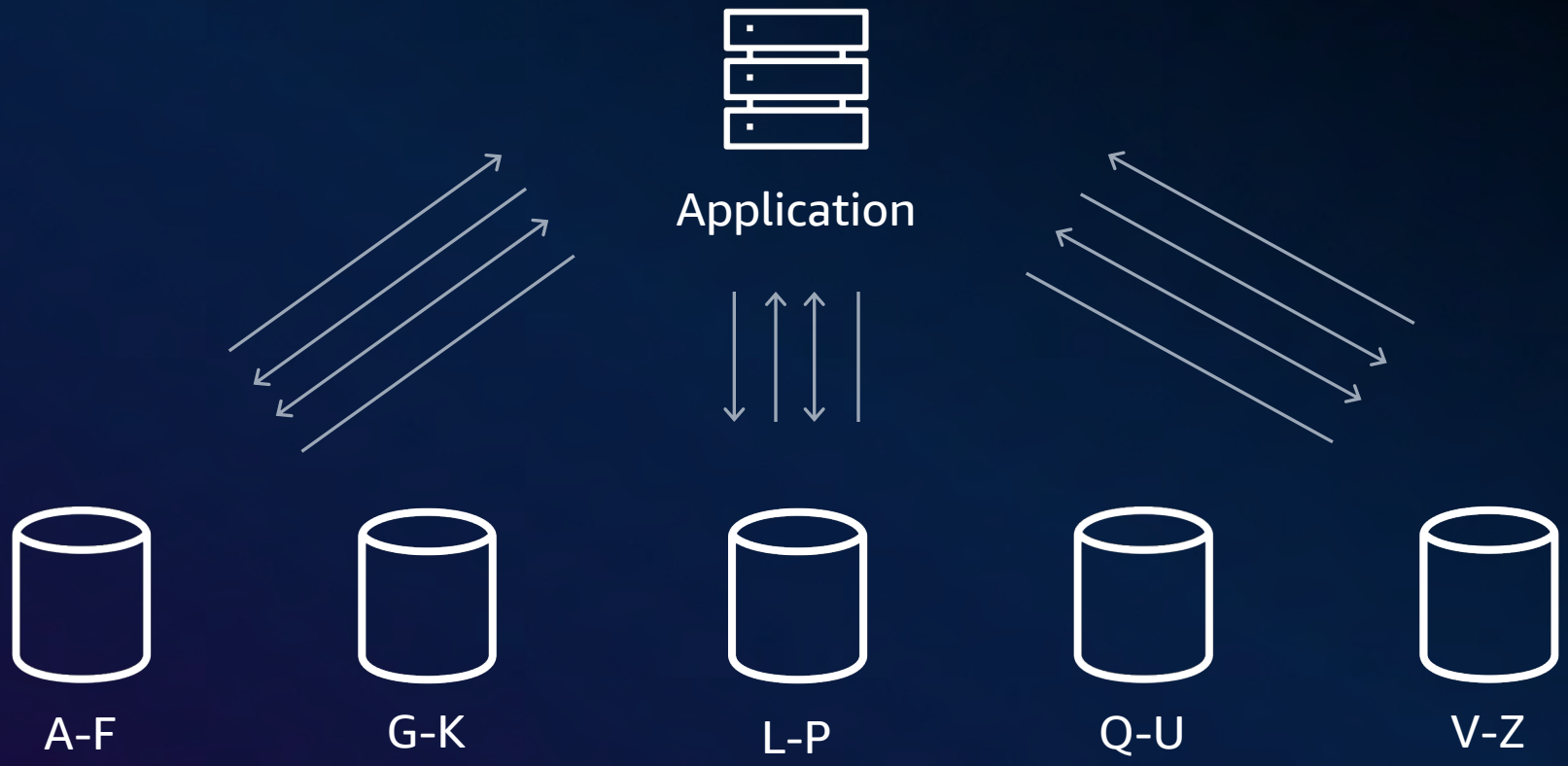
Data size

Queries

Sharding

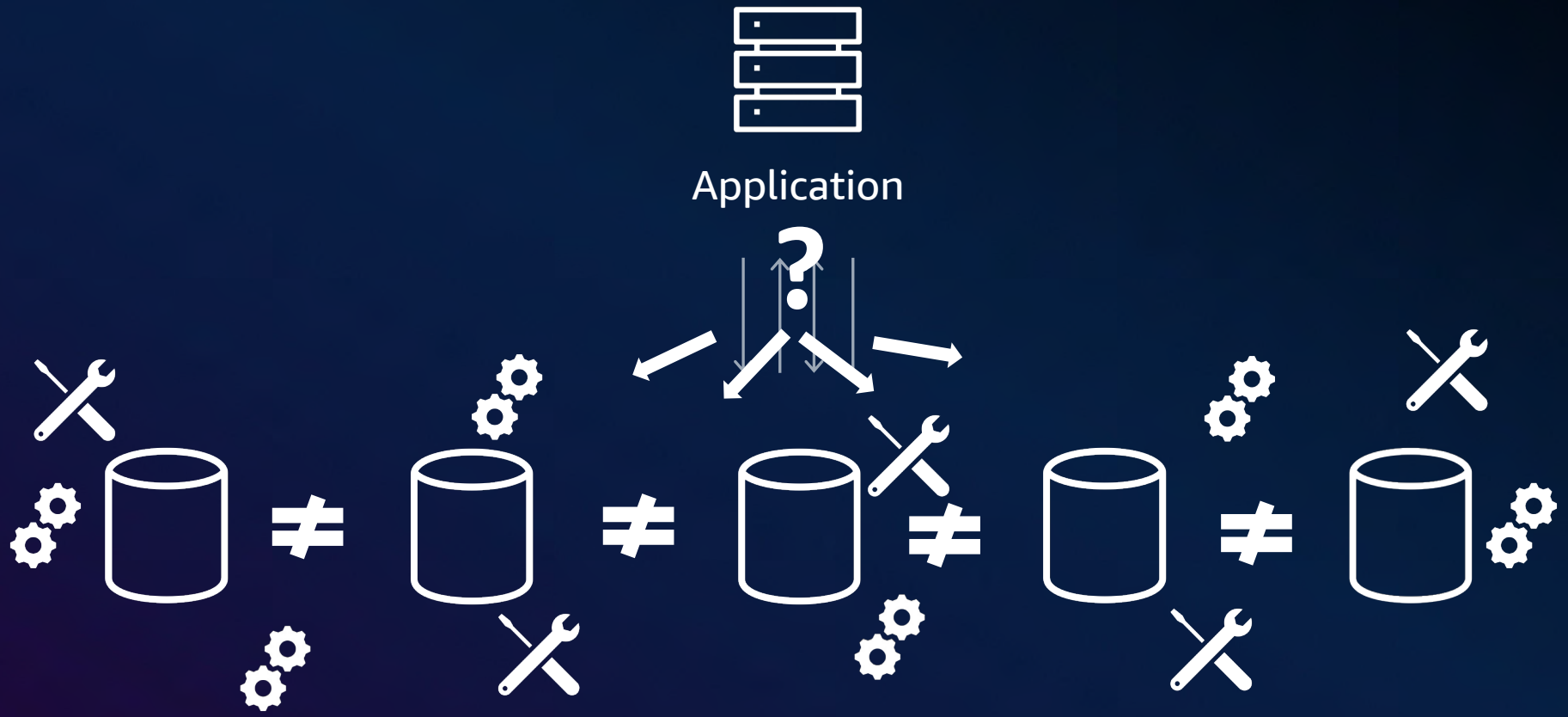


Sharding brings *scale*



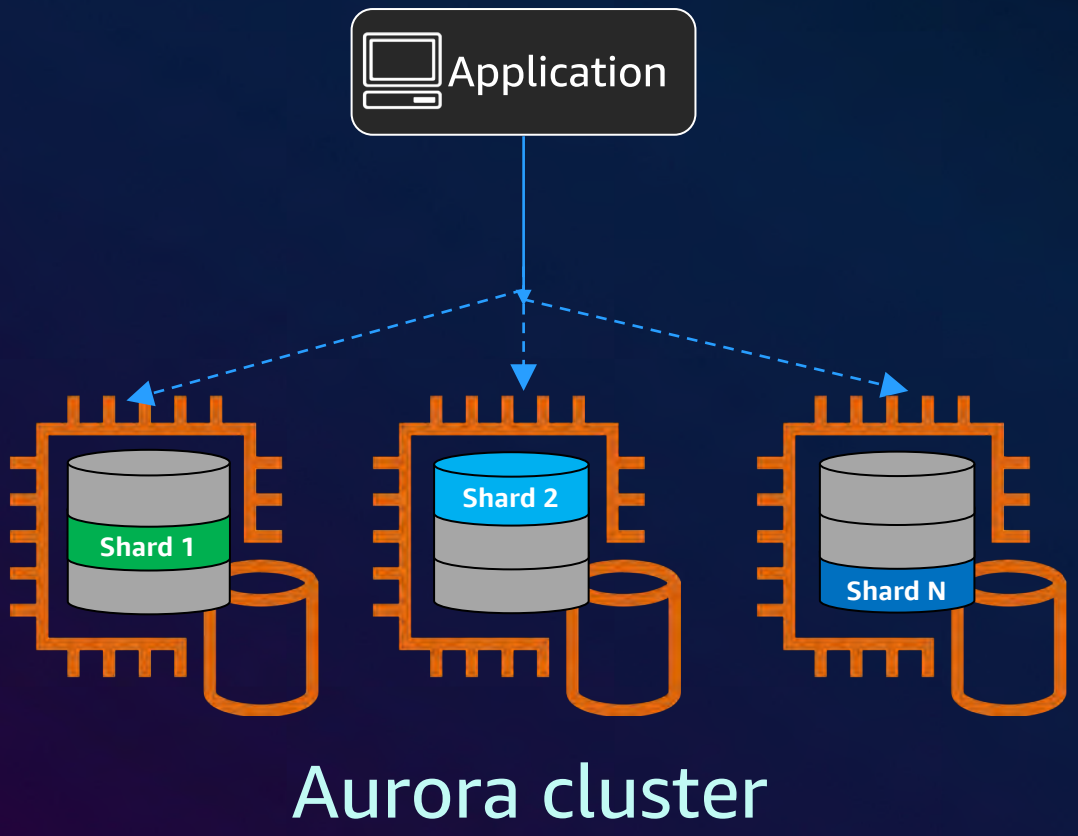
Sharding brings ...

Sharding brings … *problems*



Querying? **Consistency?** **Maintenance?**

Amazon Aurora Limitless Database (Limited Preview)



- Amazon Aurora クラスタを **1 秒あたり数百万件を超える書き込みトランザクション**にスケーリングし、**ペタバイト単位のストレージ**を管理可能
  - カスタムアプリケーションロジックを作成したり、複数のデータベースを管理する必要がない
  - 1つのエンドポイントに接続するだけで、トランザクションの一貫性を維持しながら、データやクエリを複数のシャードに自動で分散し、ワークロードを簡単にスケール
- アプリケーションのワークロードをもとに、リソースを垂直・水平方向に自動的にスケール
- **Distributed query / transaction**
- **Sharded / Reference tables**
- **PostgreSQL 互換**

Amazon Aurora Limitless Database

Scaling *Managed*

Serverless

Distributed

Single interface

Transactionally
Consistent

Millions of
transactions

Petabytes of
Data

Fundamentally Aurora PostgreSQL

PostgreSQL wire compatible

PostgreSQL parser and semantics

Broad surface area coverage

Selected extensions

Query execution basics

PostgreSQL foreign tables foundation

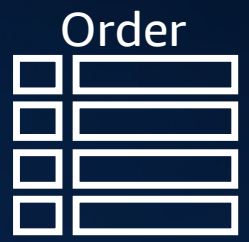
Enhancements in core engine

A custom foreign data wrapper

Shard management

Use Limitless Database

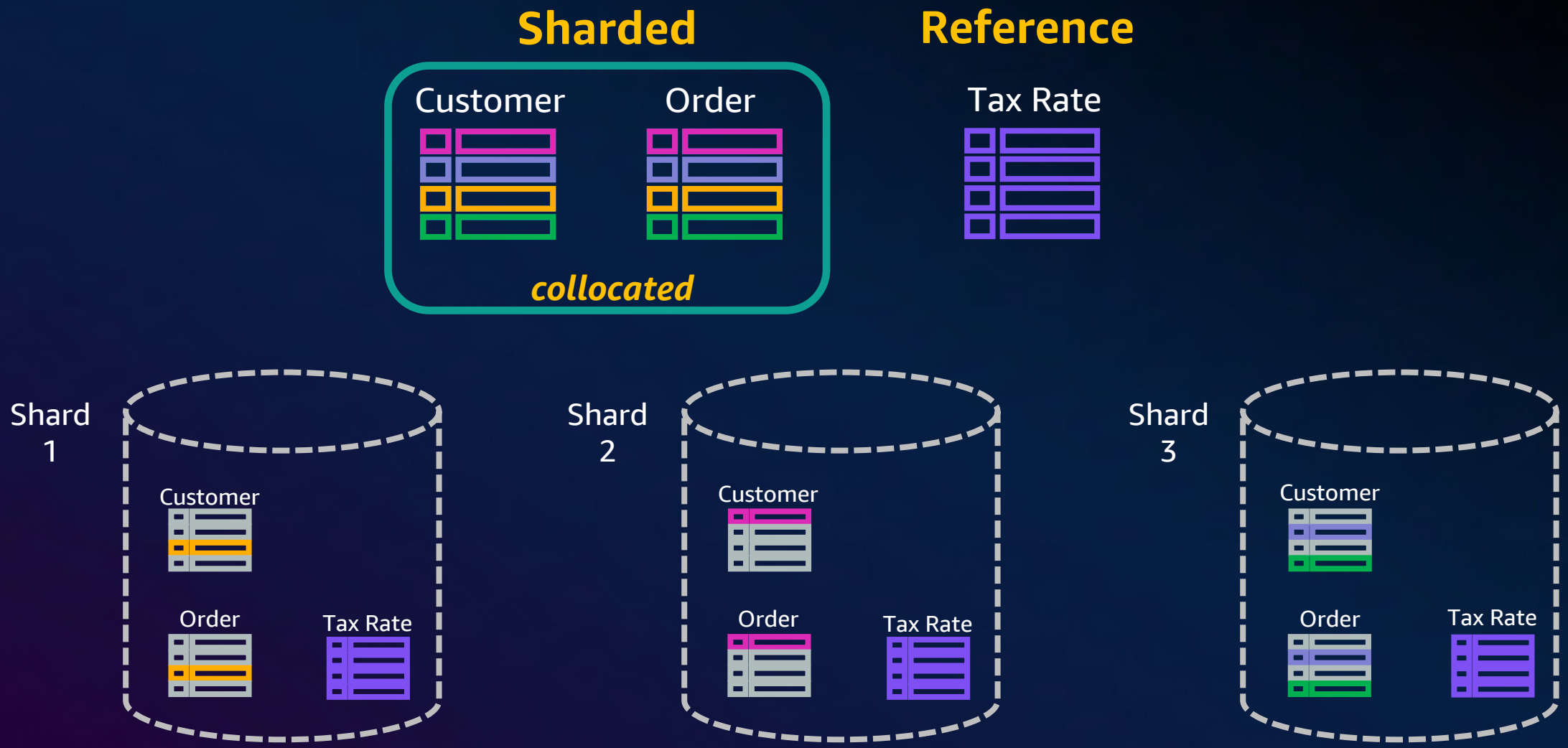
cust_id
name
email



order_id
cust_id
amount
tax_rate_id

tax_rate_id
city
state
country
tax_rate

Use Limitless Database



Create sharded customer table

```
SET rds_aurora.limitless_create_table_mode='sharded';

SET rds_aurora.limitless_create_table_shard_key='{"cust_id"}';

CREATE TABLE customer (
      cust_id INT PRIMARY KEY NOT NULL,
      name    TEXT,
      email   VARCHAR(100)
      );
```

Create co-located order table

```
SET rds_aurora.limitless_create_table_mode='sharded';
SET rds_aurora.limitless_create_table_shard_key='{"cust_id"}';


SET rds_aurora.limitless_create_table_collocate_with='customer';

CREATE TABLE order (
      order_id     INT NOT NULL,
      cust_id      INT NOT NULL,
      amount       DOUBLE NOT NULL,
      tax_rate_id DOUBLE,
      PRIMARY KEY (order_id, cust_id)
      );
```

Create reference table tax_rate

```
SET rds_aurora.limitless_create_table_mode='reference';


CREATE TABLE tax_rate (
      tax_rate_id INT PRIMARY KEY NOT NULL,
      city          TEXT NOT NULL,
      state         TEXT,
      country       TEXT NOT NULL,
      tax_rate      DOUBLE NOT NULL
      );

SET rds_aurora.limitless_create_table_mode='standard';
```

Sharded tables

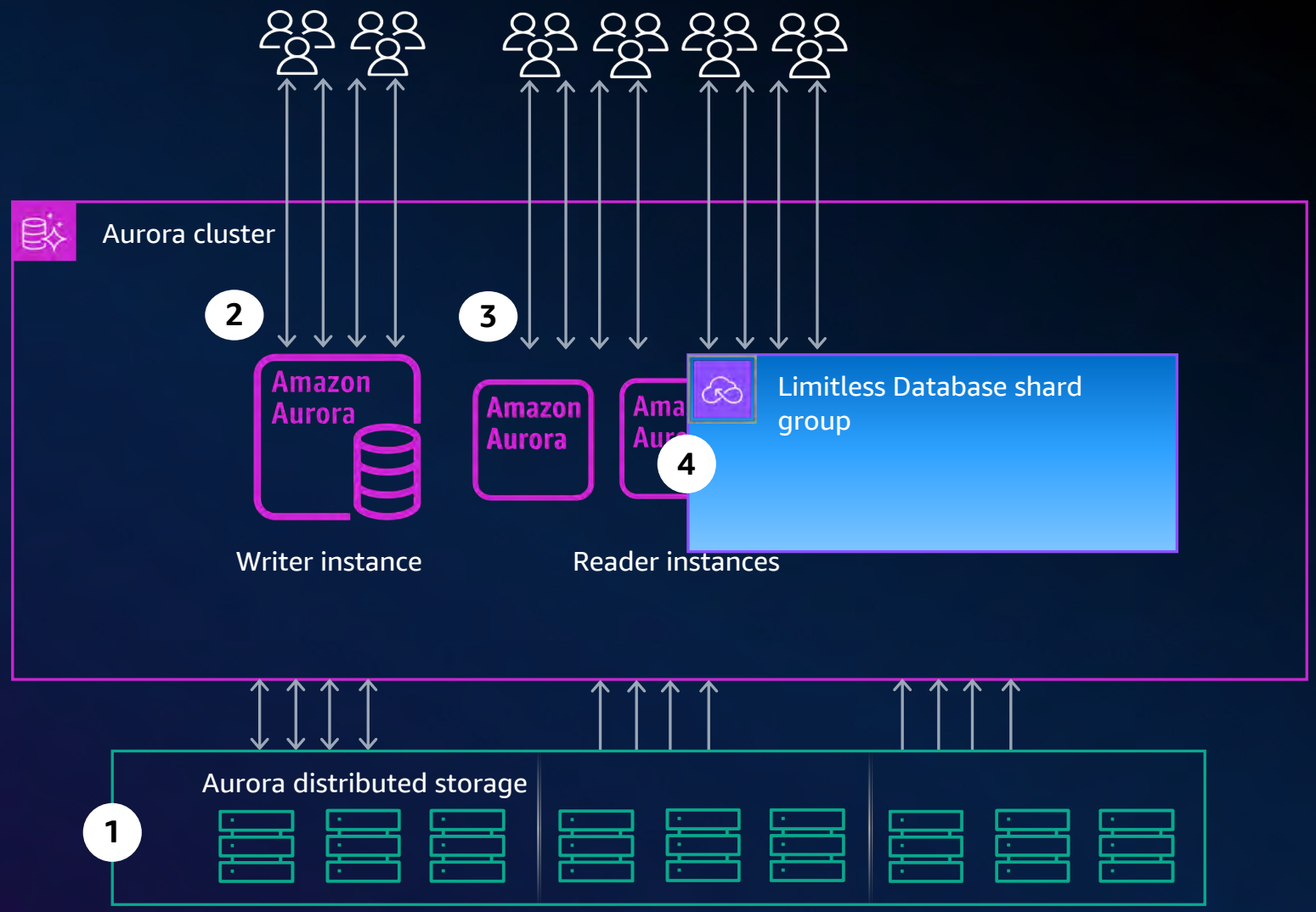
```
set rds_aurora.limitless_create_table_mode='sharded';
set rds_aurora.limitless_create_table_shard_key='{bid}';

create table pgbench_branches(
  bid int not null,
  bbalance int,
  filler char(88));
```

```
postgres_limitless=> \d+ pgbench_branches
                                  Partitioned table "public.pgbench_branches"
  Column  |     Type      | Collation | Nullable | Default | Storage  | Compression | Stats target | Description
----------+---------------+-----------+----------+---------+----------+-------------+--------------+-------------
 bid      | integer       |           | not null |         | plain    |             |              |
 bbalance | integer       |           |          |         | plain    |             |              |
 filler   | character(88) |           |          |         | extended |             |              |
Partition key: HASH (bid)
Partitions: pgbench_branches_fs00001 FOR VALUES FROM (MINVALUE) TO ('-4611686018427387904'),
            pgbench_branches_fs00002 FOR VALUES FROM ('-4611686018427387904') TO ('0'),
            pgbench_branches_fs00003 FOR VALUES FROM ('0') TO ('4611686018427387904'),
            pgbench_branches_fs00004 FOR VALUES FROM ('4611686018427387904') TO (MAXVALUE)
```

Internal Architecture

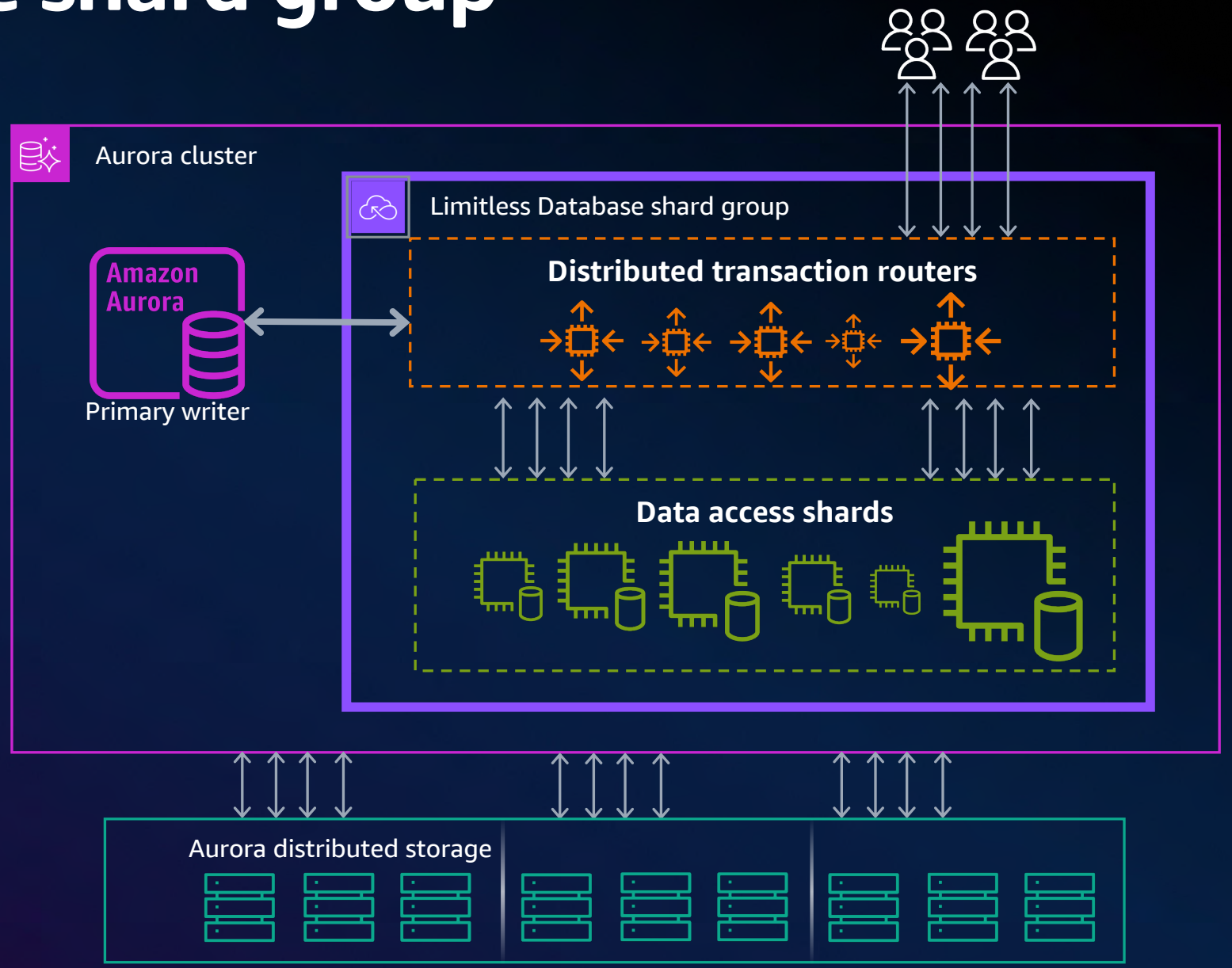
Standard Aurora architecture

1. 分散ストレージの Aurora ボリューム (Grover)
2. Aurora Writer instance
3. 可用性と読み取りスケーリングのための Reader instance
4. Limitless Database は**シャードグループ**の概念を導入



Limitless Database shard group

- Aurora cluster 内に作成
- Limitless Database に必要な機能が全て所属
- **単一エンドポイント**を提供
- 指定されたキャパシティ内で自動でダウンタイムなく水平・垂直スケーリング



Limitless Database shard group

DB identifier	Status	Role	Engine	Region & AZ	Size
apg-limitless-database	Available	Regional cluster	Aurora PostgreSQL	ap-northeast-1	1 instance
apg-limitless-database-instance-1	Available	Writer instance	Aurora PostgreSQL	ap-northeast-1d	Serverless v2 (8 - 64 ACUs)
limitless-shard	Available	DB shard group	Aurora PostgreSQL	ap-northeast-1	Limitless Database (Max: 384 ACUs)

Limitless Database configuration

Maximum capacity per DB shard group
384 ACUs (768 GiB) per AZ

Shard group deployment
Single DB instances

Endpoints (1)

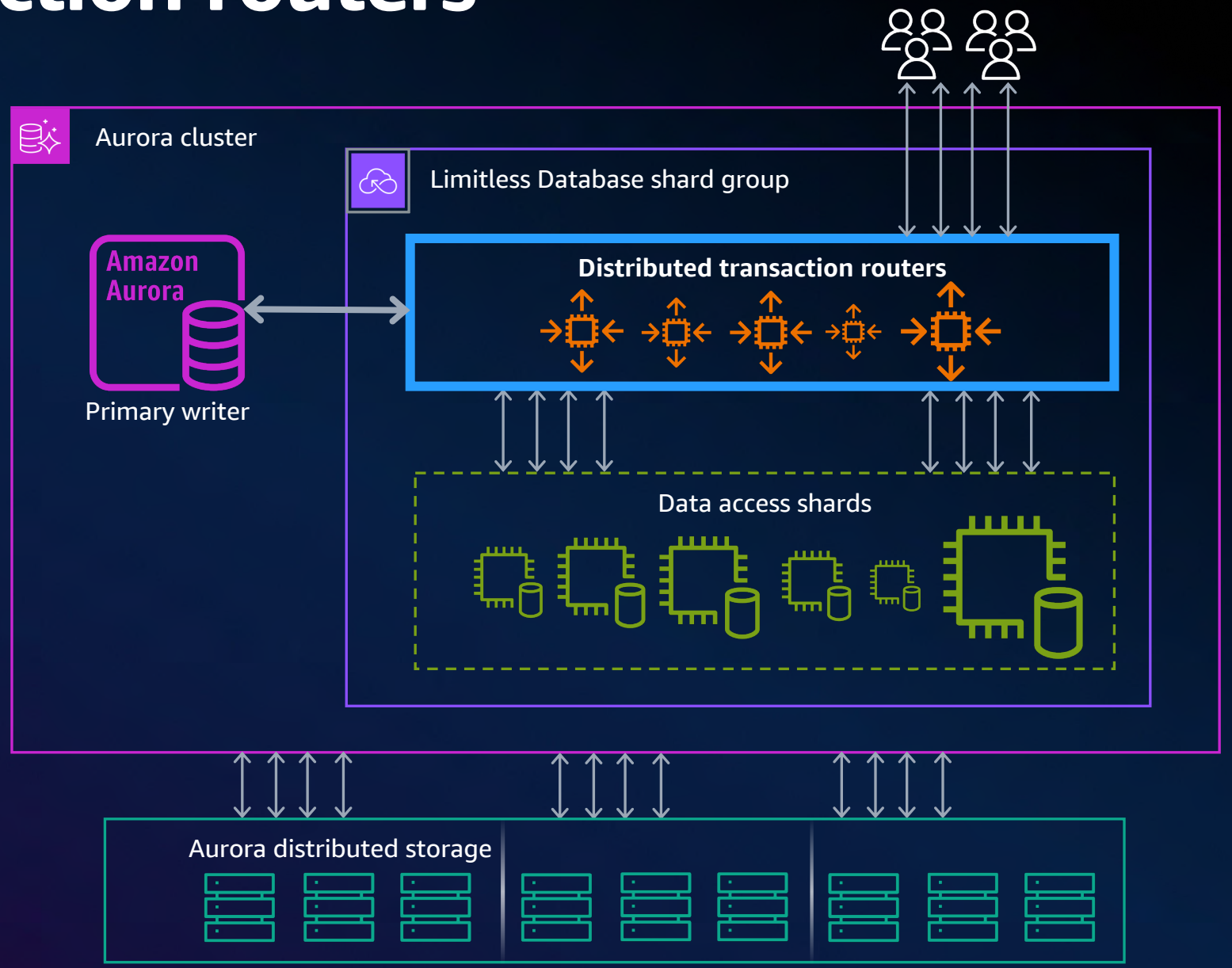
Find resources

1

Endpoint name	Status	Type	Port
apg-limitless-database.limitless-[redacted]ap-northeast-1.rds.amazonaws.com	Available	DB shard group	5432

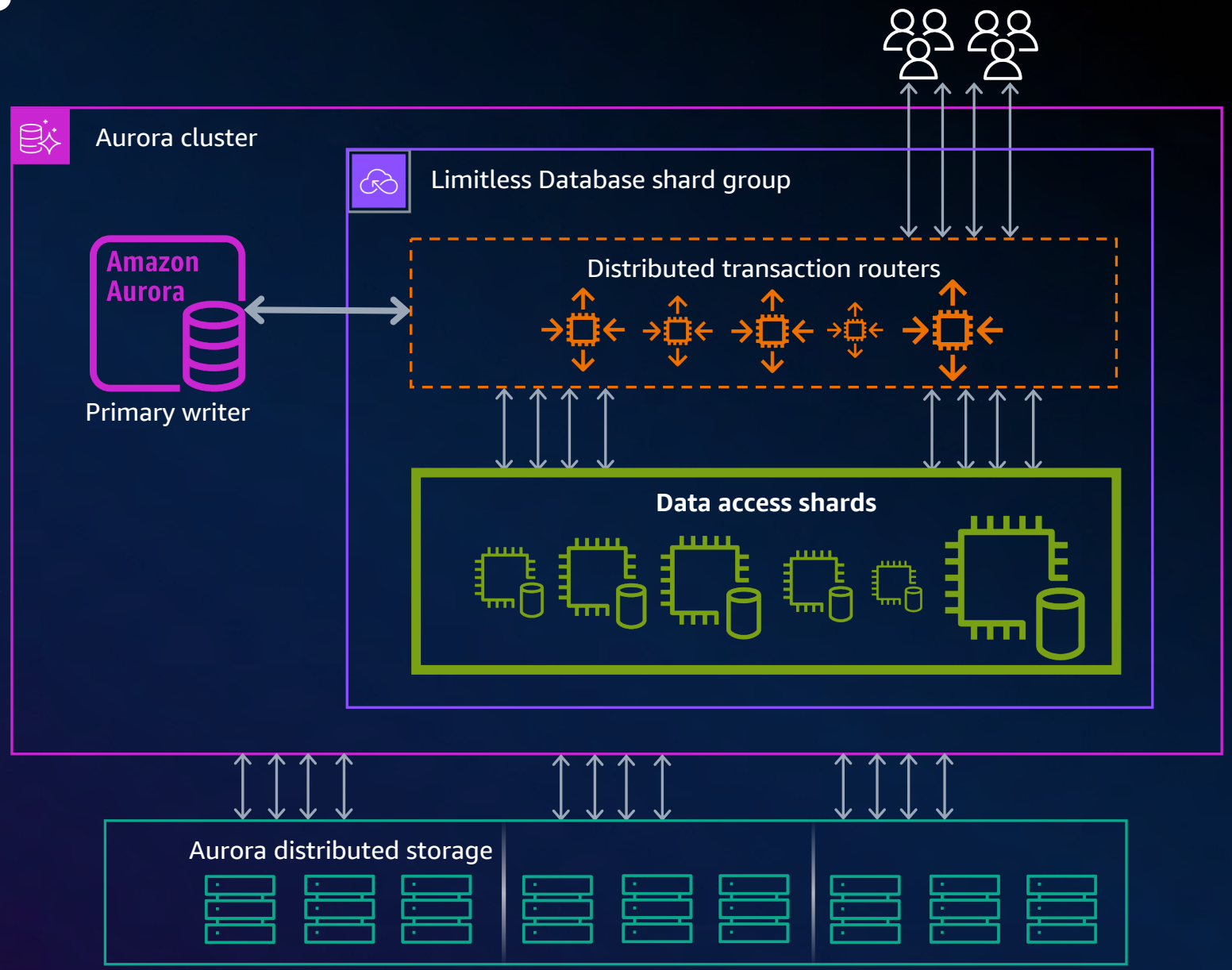
Distributed transaction routers

- Database shard group エンドポイント
- スキーマと shard key に関するメタデータを管理
- 時間ベースのトランザクションを使用し、トランザクション分離と一貫性を保つ
- クエリを解析し、シャード全体に分散させる最適な方法を決定
- マルチシャードトランザクションでは、分散コミットを実行
- マルチシャードクエリでは、各シャードの結果を集約



Data access shards

- Sharded table の key space の一部を所有し、reference table の完全なコピーを保持
- ルータからトランザクションコンテキストとプラン情報を受け取り、最終計画を作成し、クエリを実行
- シングルシャードトランザクションは、シャードがコミット全体を実行



Query flow

Router

1. クライアントからクエリを受信
2. シャードに送信する内容と、実行する JOIN をもとに実行プランを作成
3. トランザクションコンテキストを使用し部分クエリと時刻情報をシャードに送信

7. 必要に応じて最終的な結合、フィルタ、および集約を行う

Shard

4. ルータから部分クエリなどを受信
5. ローカル結合とスキャンを計画
6. クエリを実行し結果をルータに送信

シングルシャードクエリ

クエリが1つのシャードで処理可能な場合に最高のパフォーマンスを実現

```
postgres_limitless=> EXPLAIN (VERBOSE, COSTS OFF) SELECT * FROM customers WHERE customer_id = 100;
------------------------------------------------------------------------------------------------------------------------------------------------------
 Foreign Scan
   Output: customer_id, dummy_id, customer_name, balance
   Remote Plans from Shard postgres_s4:
         Seq Scan on public.customers_ts00004 customers_fs00002
           Output: customers_fs00002.customer_id, customers_fs00002.dummy_id, customers_fs00002.customer_name, customer
s_fs00002.balance
           Filter: (customers_fs00002.customer_id = 100)
   Remote SQL:  SELECT customer_id,
     dummy_id,
     customer_name,
     balance
    FROM public.customers customers_fs00002
   WHERE (customer_id = 100)
(12 rows)
```

Hash-range partitioning

- 64 ビット空間の範囲に各シャードのデータを割り当て
  - **データ: テーブルフラグメント**
  - **メタデータ: テーブルフラグメントリファレンス**
- テーブルフラグメントをシャーディング
- transaction router と data access shard は**テーブルフラグメントリファレンスのみ保持**し、**データの実態は保持しない**

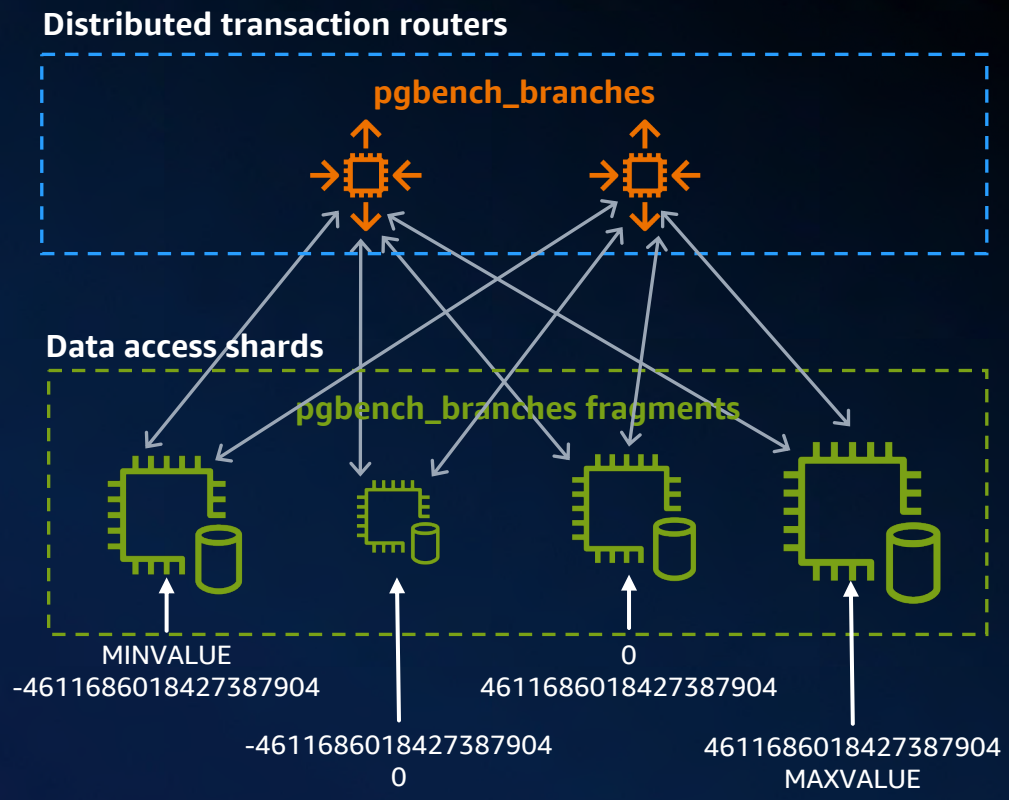
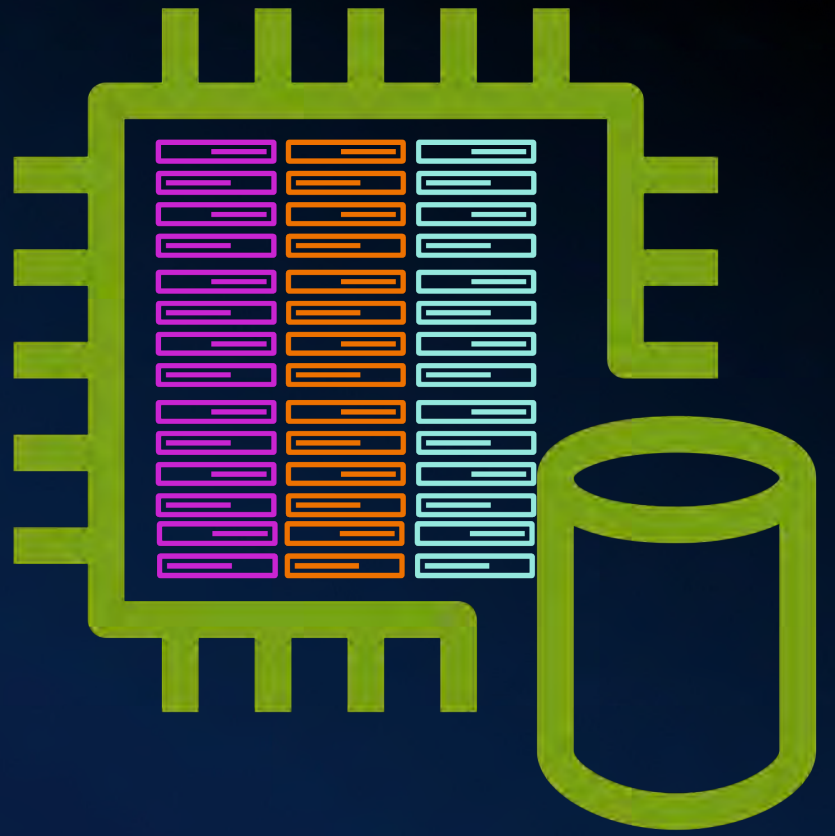


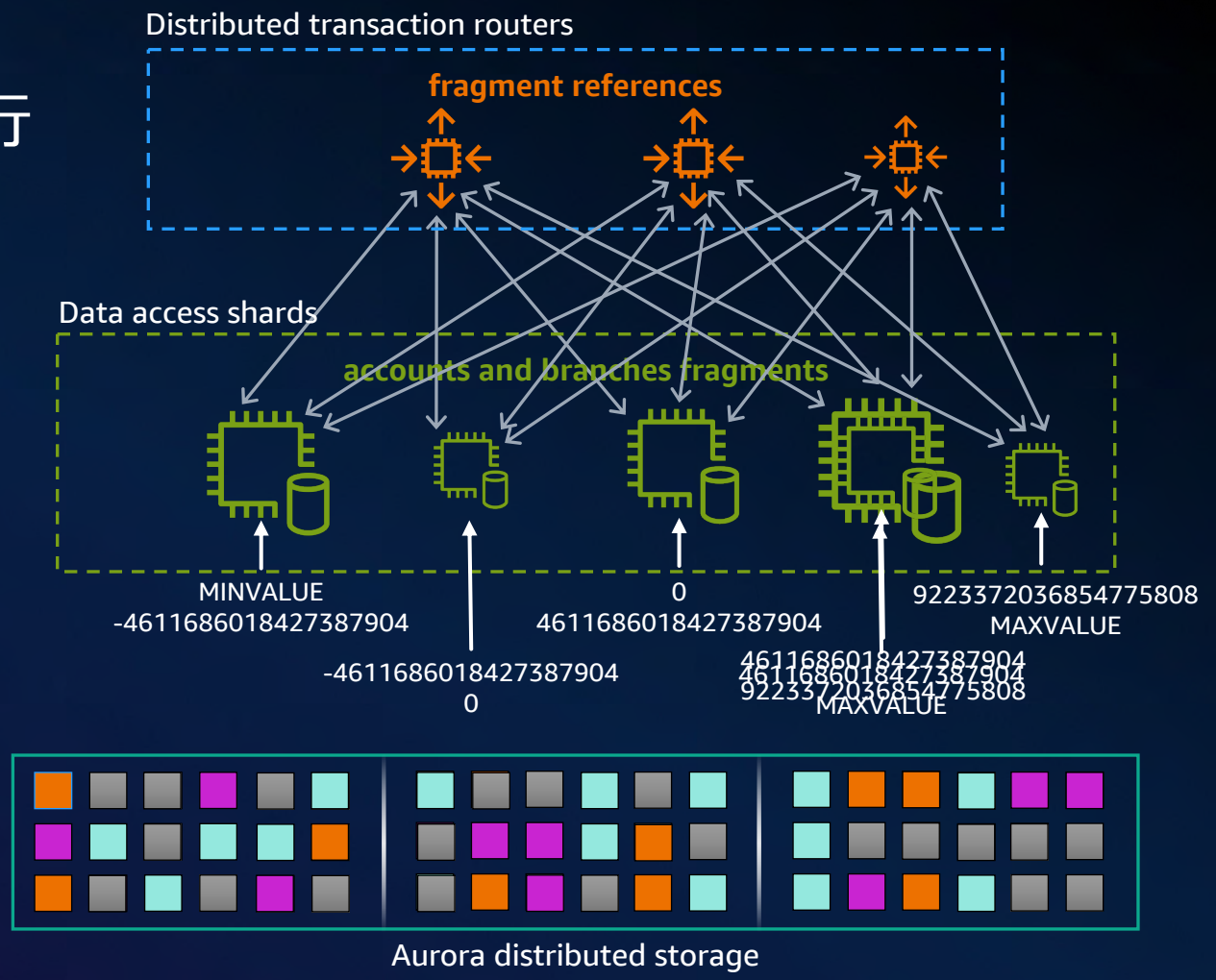
Table slicing

- テーブルフラグメントはサブレンジのスライスに分割
- ユーザから直接参照できない
- シャード内の並列性向上
- 水平スケールアウトで再配置



Horizontal scale out

- 負荷やストレージの使用量に応じて実行
- Collocated table slices は同時に移動
- Aurora storage level cloning と replication を活用
- ルータの追加・削除も実行



Creating a shard group

```
aws rds create-db-shard-group
    --db-cluster-identifier proddb
    --db-shard-group-identifier proddb-sg
    --max-acu 1536
    --compute-redundancy 2
```

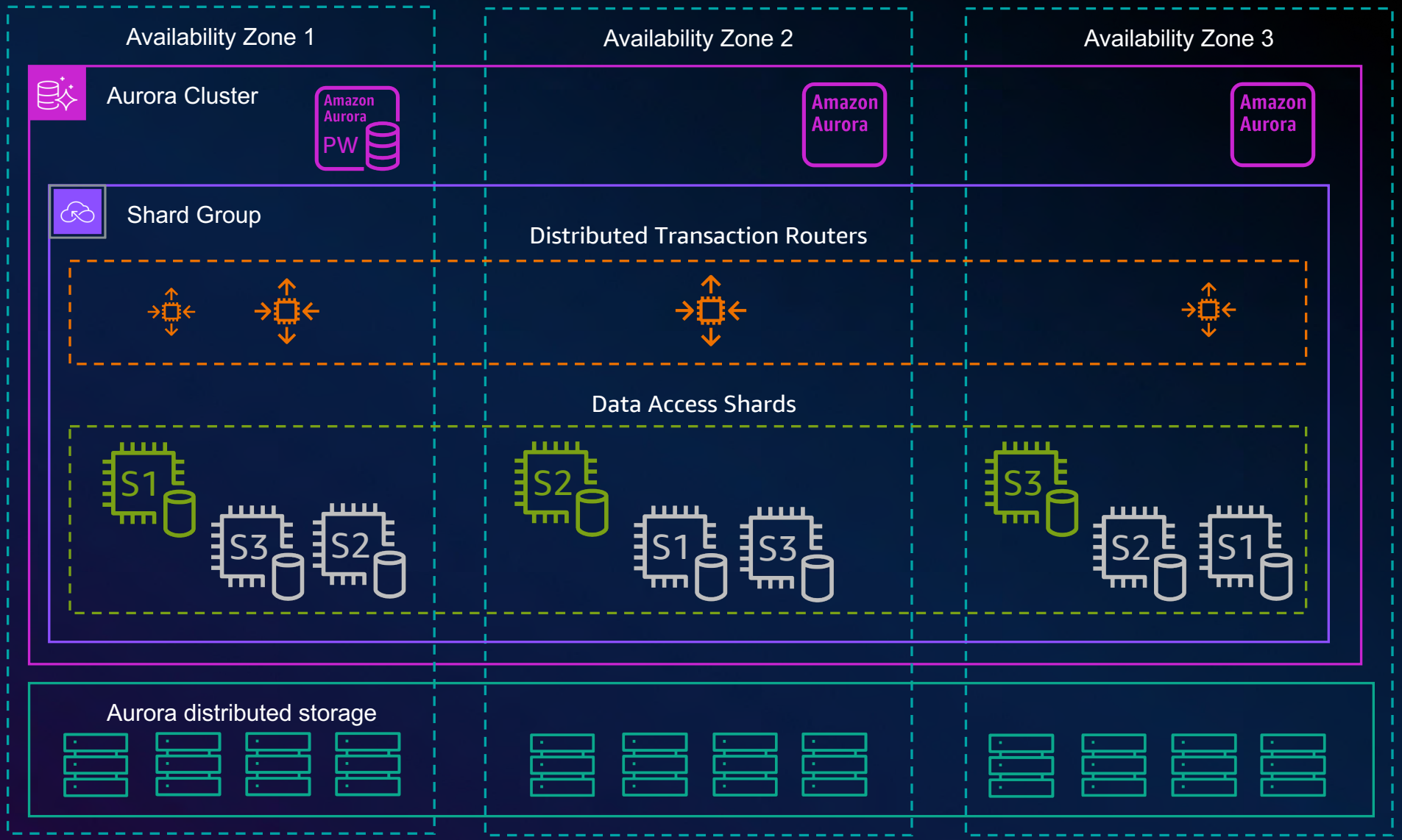
Compute redundancy

Compute redundancy 0

Compute redundancy 1

Compute redundancy 2

Primary writer のHAも設定可能



Challenges in a distributed database

Challenges in a distributed database

Coordination limits scalability

Transaction scope unknown until commit

Query fragments execute at different times

Maintain order

Consistent restores

Transaction support (Isolation level)

READ COMMITTED

クエリ開始前にコミットされた最新のデータを参照可能

トランザクション内のクエリごとに異なるデータが表示される可能性がある

REPEATABLE READ

トランザクション開始前にコミットされた最新のデータを参照可能

トランザクション内のすべてのクエリには同じデータが表示される

Design goal: PostgreSQL のトランザクション セマンティクスを維持

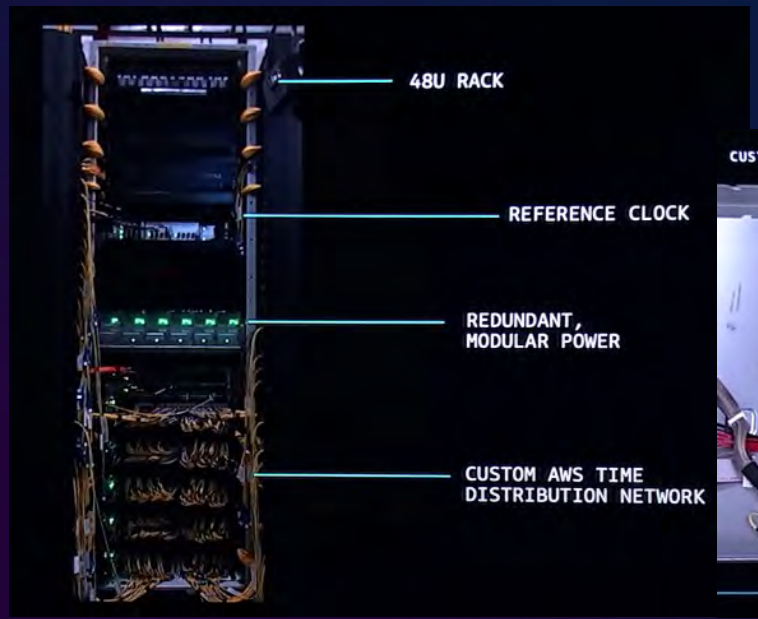
Transaction support (Isolation level)

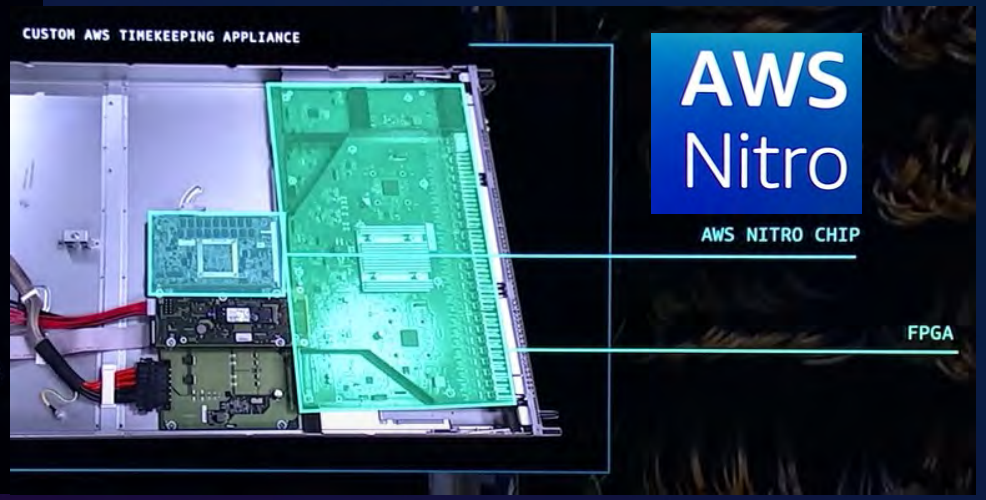
… どのように分散データベースで実現するか

Solved with bounded clocks

EC2 TimeSync service (µs)

- Current time
- Earliest possible time
- Latest possible time

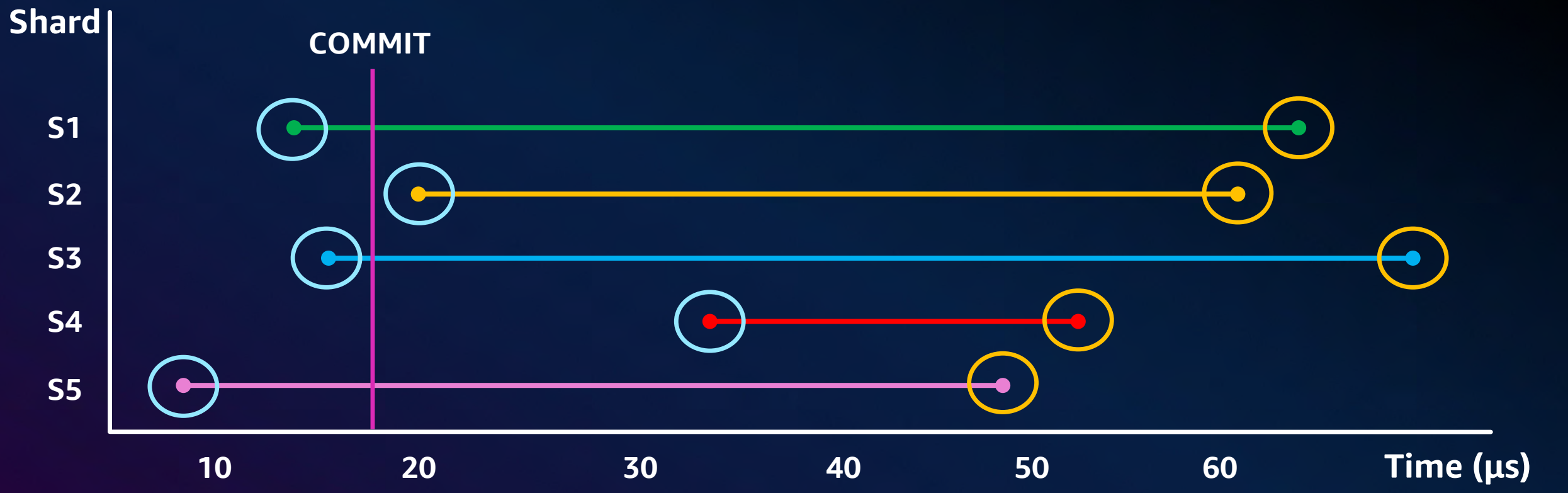




Integrated into PostgreSQL

- スナップショットとコミットのタイムスタンプを利用した Tuple の可視性
- Global read-after-write
- One-phase & two-phase commit

Bounded clocks



Earliest possible time

Latest possible time

READ COMMITTED

Transaction T1

```
SELECT SUM(abalance) FROM pgbench_accounts;
10000000
```

1) router gets time t100
2) executes sum() on each shard using snapshot@t100
3) aggregates the result

Transaction T3

```
SELECT SUM(abalance) FROM pgbench_accounts;
10000000
```

1) router gets time t116
2) executes sum() on each shard using snapshot@t116
3) aggregates the result

Transaction T2

```
BEGIN;
UPDATE pgbench_ac
abalance - 500 WHERE bid = 619 and aid =
61890340;
UPDATE pgbench_accounts SET abalance =
abalance +
80044011;
COMMIT;
```

1) router gets time t103
2) execute on shard w/bid 619 using snapshot@t103

1) router gets time t107
2) execute on shard w/bid 801 using snapshot@t107

1) router determines 2PC, asks shards to prepare
2) shard w/619 prepares@t118
   shard w/801 prepares@t112
3) router assigns commit@t120
4) acks commit when
   a) writes durable on disk
   b) earliest possible time > t120
5) router tells shards to commit@t120

REPEATABLE READ

Transaction T1

```
BEGIN TRANSACTION ISOLATION LEVEL
REPEATABLE READ;

SELECT abalance FROM pgbench_accounts WHERE
bid = 619 and aid = 61890340;
704
```

1) router gets time t100
2) execute on shard w/bid 619 using snapshot@t100

1) execute on shard w/bid 801 using snapshot@t100

```
SELECT abalance FROM pgbench_accounts WHERE
bid = 801 and aid = 80044011;
1
```

Transaction T2

```
BEGIN;

SELECT abalance FROM pgbench_accounts
WHERE bid = 801 and aid = 80044011 FOR
UPDATE;
1

UPDATE pgb
1001 WHERE
COMMIT;
```

1) router gets time t103
2) execute on shard w/bid 801 using snapshot@t103

1) router uses 1PC on shard
2) shard assigns commit@t110
3) acks commit when
   a) writes durable on disk
   b) earliest possible time > t110

Transaction T3

```
SELECT abalance FROM pgbench_accounts
WHERE bid = 801 and aid = 80044011;
1001
```

1) router gets time t125
2) execute on shard w/bid 801 using snapshot@t125

Summary

Amazon Aurora Limitless Database (Limited Preview)

- Amazon Aurora Limitless Database を利用することで、shard key をもとに自動的にシャーディングが行われる
  - Write scalability の向上
  - ストレージリミットの緩和

- **1つのエンドポイント**に対しクエリを実行することで、自動的に適切なシャードに対しデータの更新・取得が行える
  - **アプリケーションにシャーディングロジックを追加する必要がなくなる**

Amazon Aurora Limitless Database (Limited Preview)

- 各コンポーネントは負荷に応じて**自動的にダウンタイム無くスケールアップ・ダウン**が行われる
  - Serverless テクノロジを採用し、最大キャパシティを超えた場合は、**シャードを担当するコンピュートノードを分割**
  - シャード毎に負荷が異なる環境においてもコスト最適化を容易に行える
  - **ストレージサイズの上限を大幅に向上**

Amazon Aurora Limitless Database (Limited Preview)

- データアクセスパターンに応じてテーブルの配置戦略を選択可能
  - **Sharded tables**: Shard key に応じて自動で data access shard にシャーディングされるテーブル
  - **Reference tables**: 全ての data access shard に配置されるテーブル
    - データサイズが小さく更新頻度が低いテーブルが適している

- **Distributed query / Distributed transactions**
  - シングルシャードクエリとして実行し、クロスシャードクエリのレイテンシを軽減するために
    - 頻繁に JOIN されるテーブルは collocation
    - Reference tables を活用

Amazon Aurora Limitless Database (Limited Preview)

- Aurora Limitless Database の使い所
  - Write scaling / Storage size 上限を引き上げることにプラスして
    - MySQL / PostgreSQL インターフェースを利用したい
    - BEGIN-COMMIT / ROLLBACK を使った複数ステートメントのような複雑なトランザクション管理をデータベースレイヤに任せたい
    - JOIN など複数のテーブルをまたいだデータを処理したい
    - 各エンジンの組込関数 / Extension を利用したい

Thank you!