YugabyteDBを選択すべき10の理由

2024年3月5日

SRA OSS LLC
彭 博 (ペン ボ)

自己紹介

ペン ボ
- 名前： 彭 博 (Bo Peng)
  pengbo@sraoss.co.jp
- 所属：SRA OSS LLC
  技術部 基盤技術グループ
- 職務：
  - OSS技術サポート、ミドルウェア構築
    - クラスタリングソフトウェア：Pacemaker/Corosync
    - 監視ソフトウェア：Zabbix
    - Kubernetes
    - YugabyteDB
    - など
  - OSS活動

SRA OSSはパートナーとしてYugabyteDB関連サービスを提供

- 2023年3月、Yugabyte社とパートナーシップを締結し、クラウドネイティブ分散データベースYugabyteDBの取り扱いを始めました。
- YugabyteDBのアセスメント・PoC、導入、サポートまでトータルに支援します。

	アセスメント	導入支援	サポート
新規	• ヒアリング・情報収集 • 機能・性能実現性の調査 • レポート作成・報告 • DB導入費用見積	• データベース構成の設計 • データベースの構築 • データベースの動作確認テスト	• 24/365サポート
移行	• ヒアリング・情報収集 • 移行可否および移行難易度の評価 • DB設計変更が必要となる項目を洗い出す • レポート作成・報告 • DB移行費用見積	• データベース構成の設計 • テーブル定義の移行 • データの移行 • バックアップなど運用手順の移行 • データベースの移行確認テスト	• 24/365サポート

YugabyteDBを選択すべき10の理由

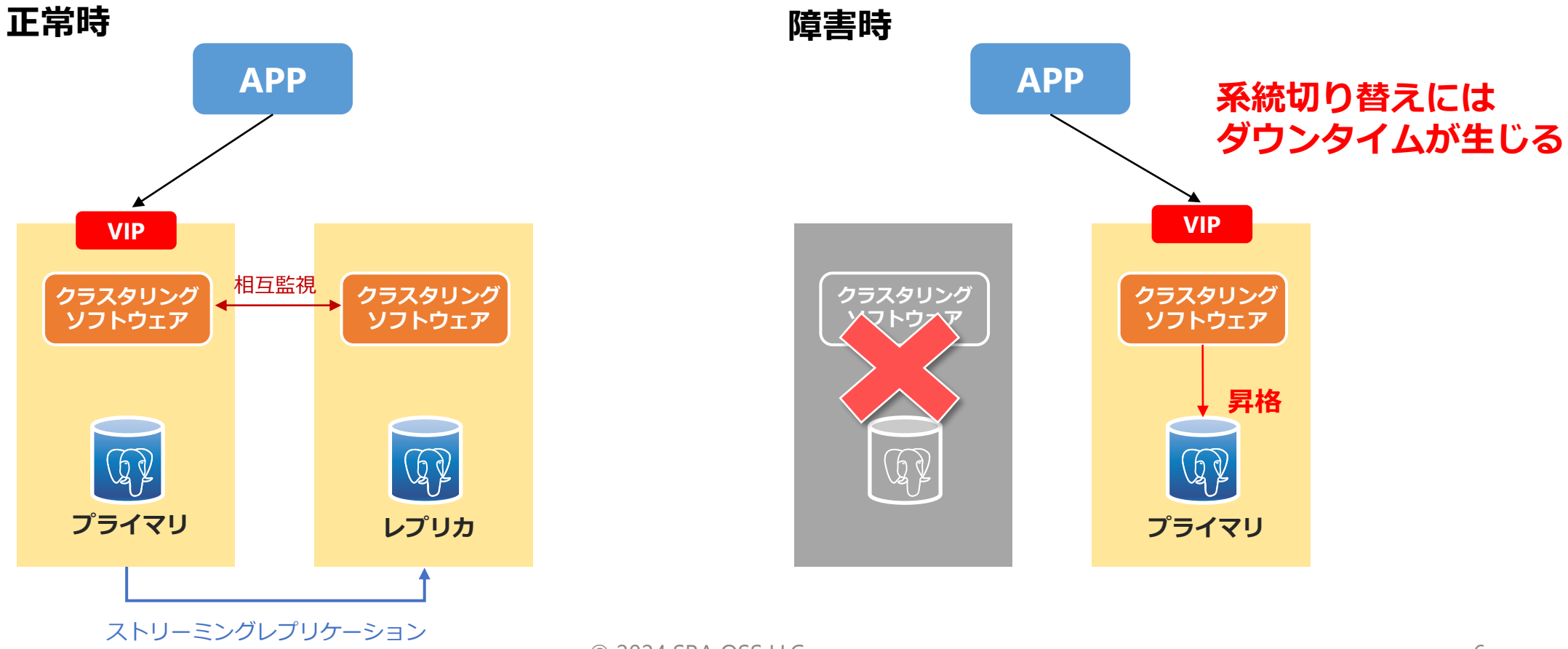
1. 高可用性
2. 耐障害性
3. スケールアウトによる大量データ処理の負荷分散
4. 自動シャーディング
5. 地理的パーティショニング
6. PostgreSQL互換、ACIDトランザクション対応
7. PostgreSQL MVCCの欠点をYugabyteDBで解決
8. PostgreSQLの肥大化問題の回避
9. 運用・管理コストの削減
10. ゼロ計画外・計画ダウンタイム

YugabyteDBを選択すべき理由 1

高可用性

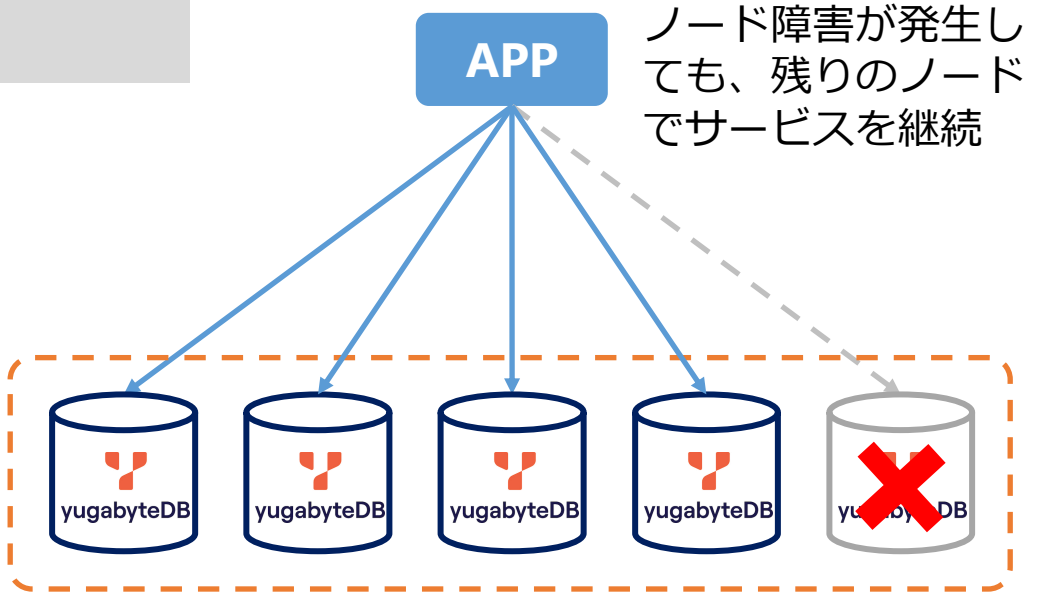
PostgreSQLで高可用性を実現するには

- PostgreSQLでは高可用性機能が提供されていない
- クラスタリングソフトウェアを用いて高可用性を実現するのが一般的



YugabyteDBの高可用性

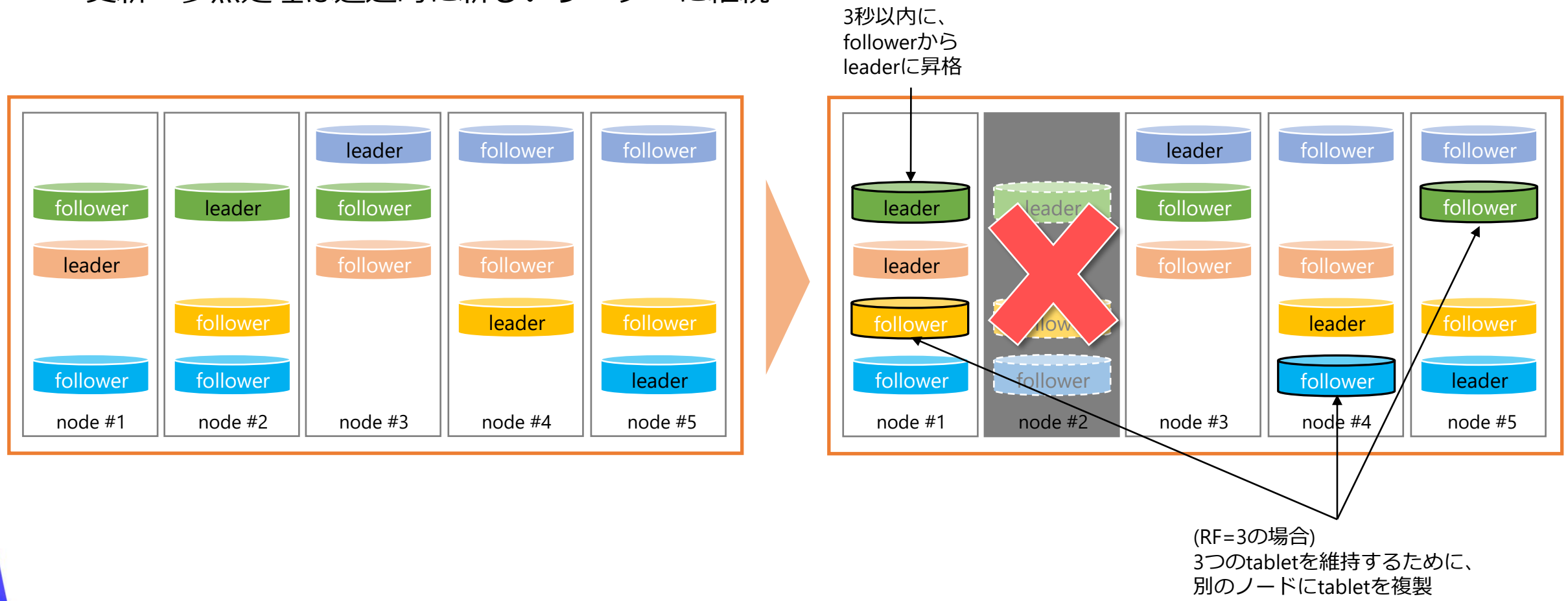
- 高可用性機能がビルトインされている
- ユーザにノード障害を意識させることなく運用続行可能
  - 障害時、**3秒以内**に新しいリーダーが選出される
  - 更新・参照処理は透過的にリトライされる
- 障害時データ損失なし



障害時の挙動

Leaderタブレット障害時の挙動

- **3秒以内**に新しいリーダーを選出
- 更新・参照処理は透過的に新しいリーダーに継続



YugabyteDBを選択すべき理由 2

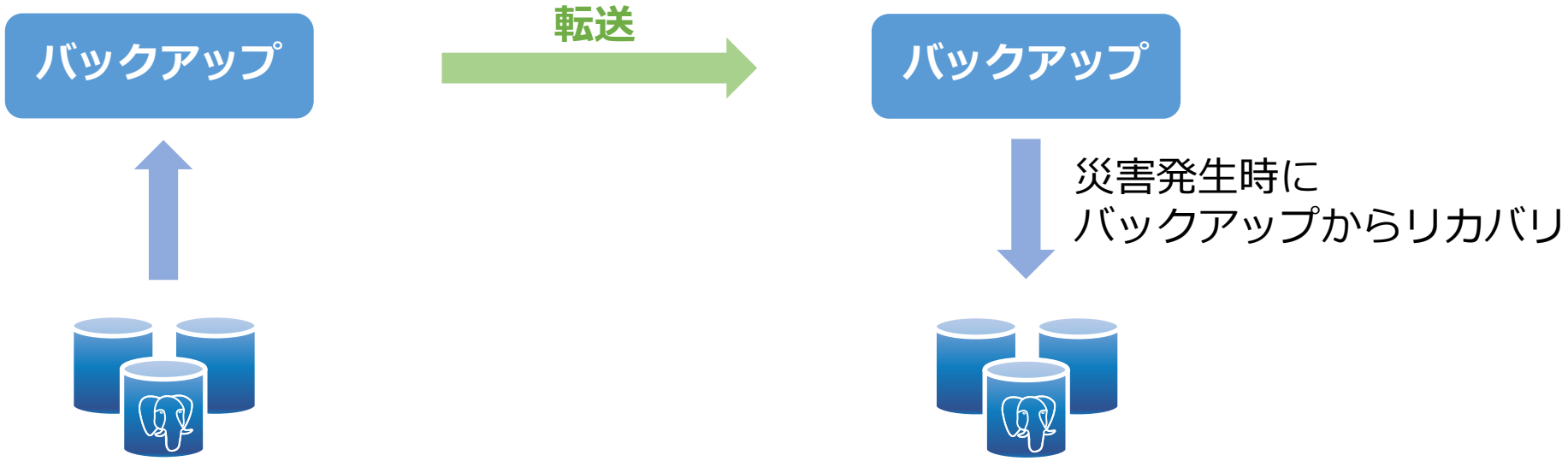
耐障害性

- あらゆる規模の障害や災害に対してダウンタイムなしで回復できる

PostgreSQLにおける災害対策

バックアップによる災害対策

- 定期的にバックアップを取得し、遠隔地へ転送

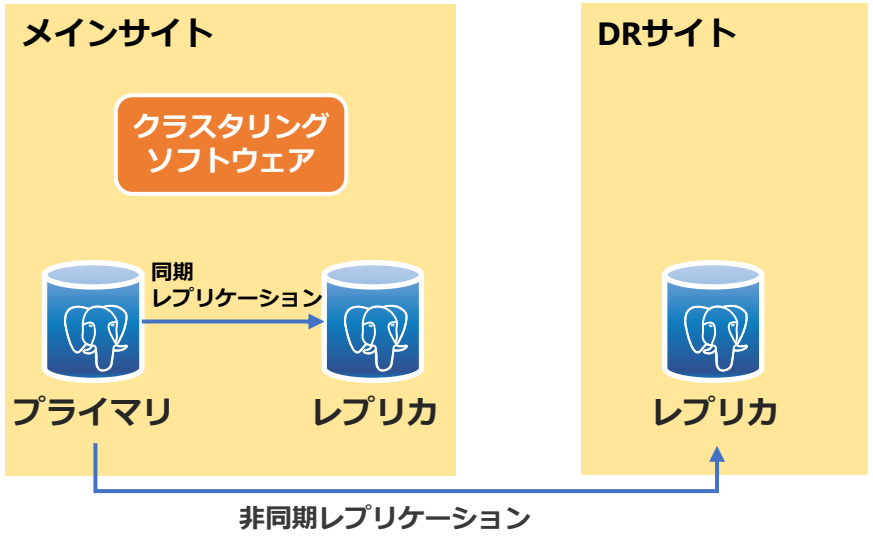


- **運用が容易**
- **バックアップを取得した時点までしか復旧できないため、データの損失は多い**

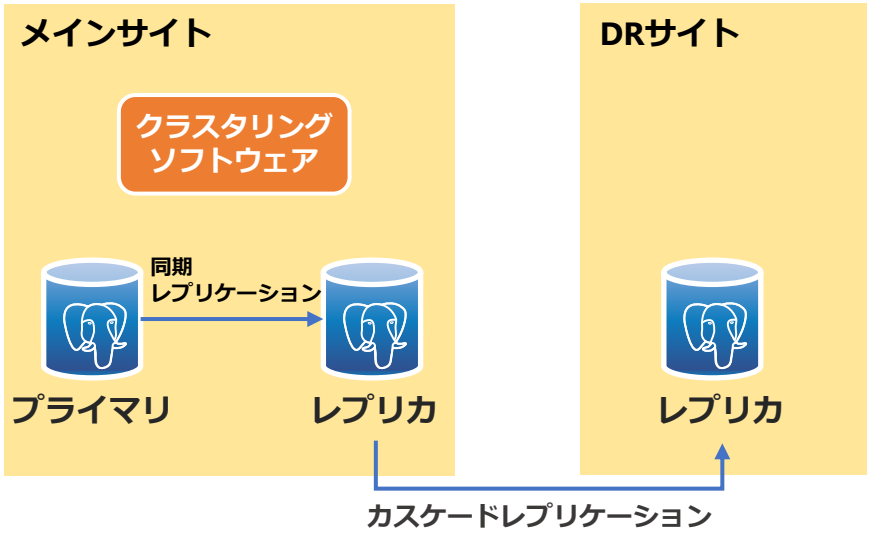
PostgreSQLにおける災害対策

レプリケーションによる災害対策

非同期レプリケーションによるサイト間データ同期



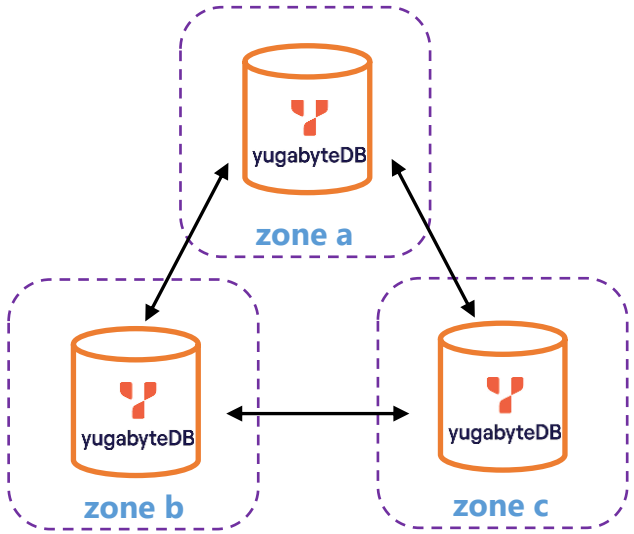
カスケードレプリケーションによるサイト間データ同期



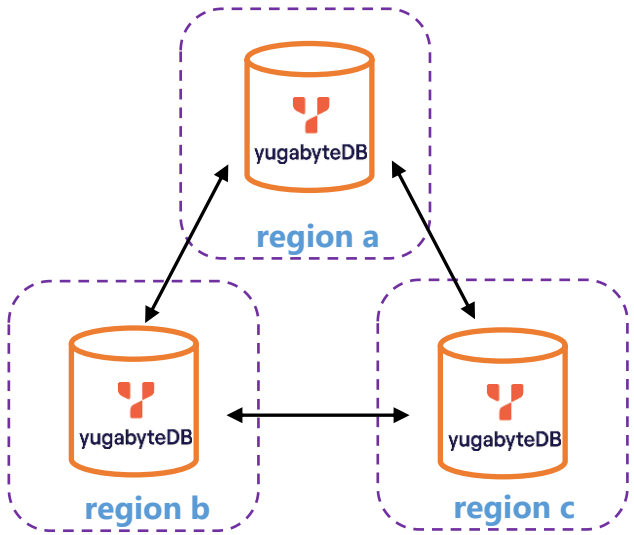
- **データ損失の可能性あり**
- **運用がやや複雑**
- **サイト障害時、手動でサイト切り替えが必要**

YugabyteDBの耐障害性

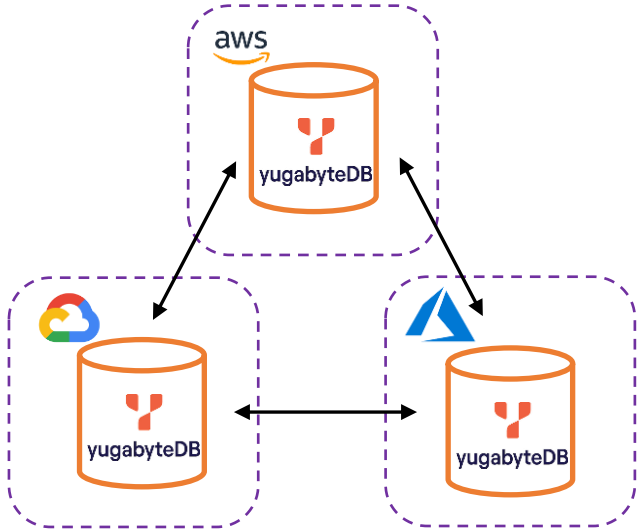
様々なデプロイメントパターン
AZレベルからクラウドサービスレベルまでの耐障害性



マルチAZ
AZレベルの耐障害性



マルチリージョン
リージョンレベルの耐障害性

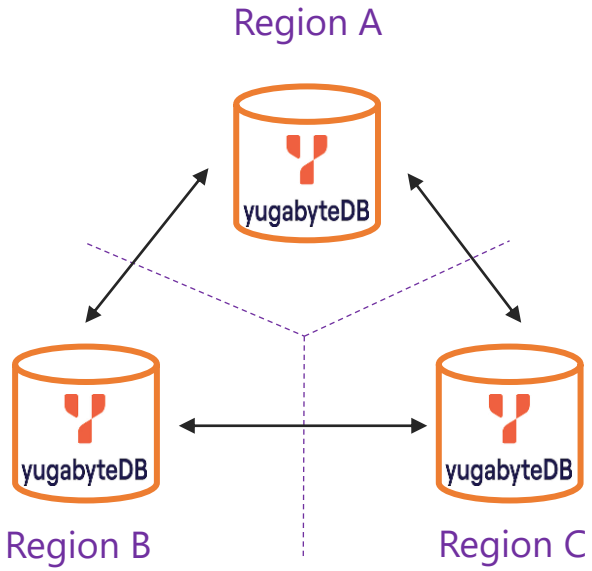


マルチクラウド
クラウドレベルの耐障害性

YugabyteDBにおける災害対策

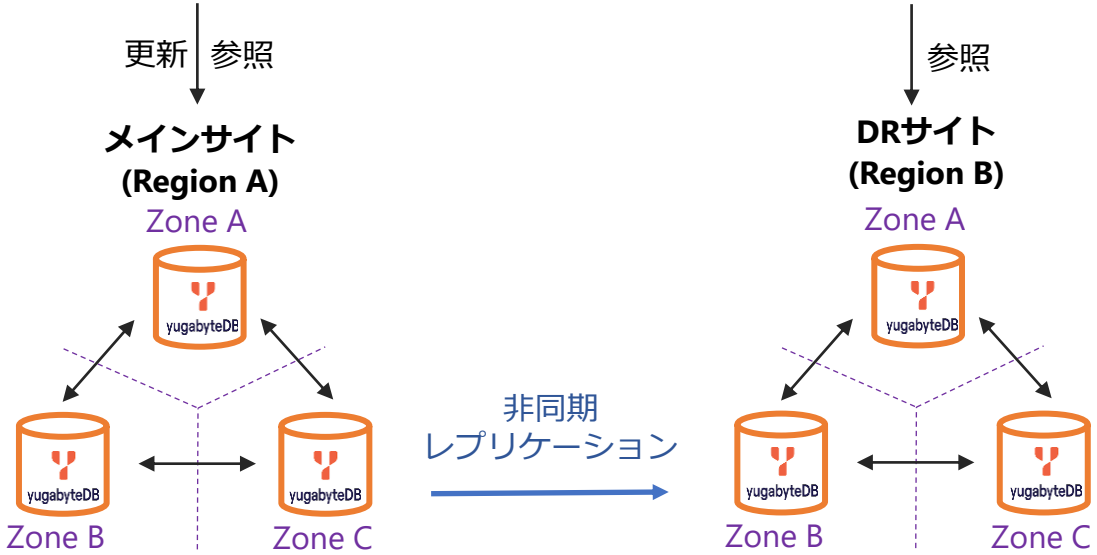
マルチリージョン構成

複数のリージョンに跨ってクラスタを構築



xCluster構成

クラスタ間非同期レプリケーション



YugabyteDBを選択すべき理由 3

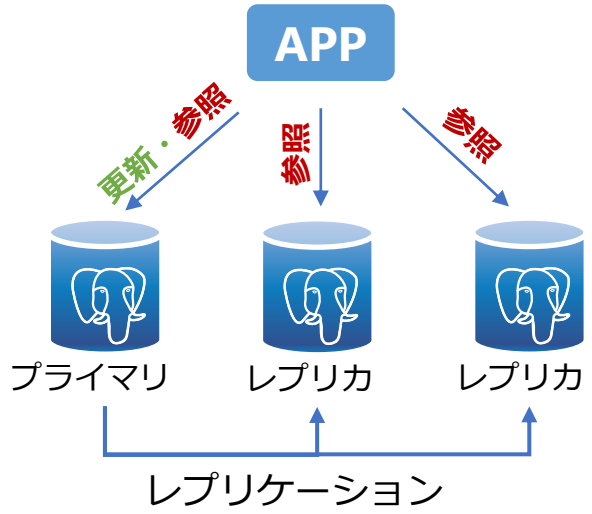
スケールアウトによる大量データ処理の負荷分散

PostgreSQLのスケールアウト

PostgreSQLの機能のみでスケールアウトするには

ストリーミングレプリケーション

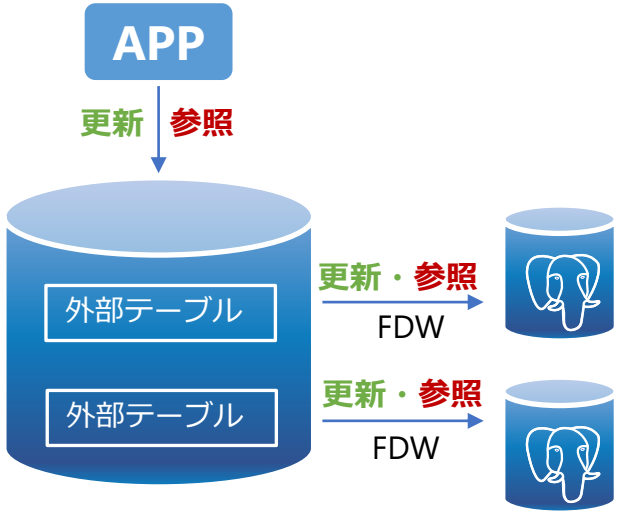
- 1台のプライマリと複数台のレプリカからなるストリーミングレプリケーション構成
- レプリカを増やすことで、参照処理をスケールアウトできる



- **更新処理はスケールアウトできない**
- **大規模データになると、処理性能が落ちてしまう**
- **大規模データの書き込み処理性能が求められるアプリケーションには適していない**

postgres_fdwによるシャーディング

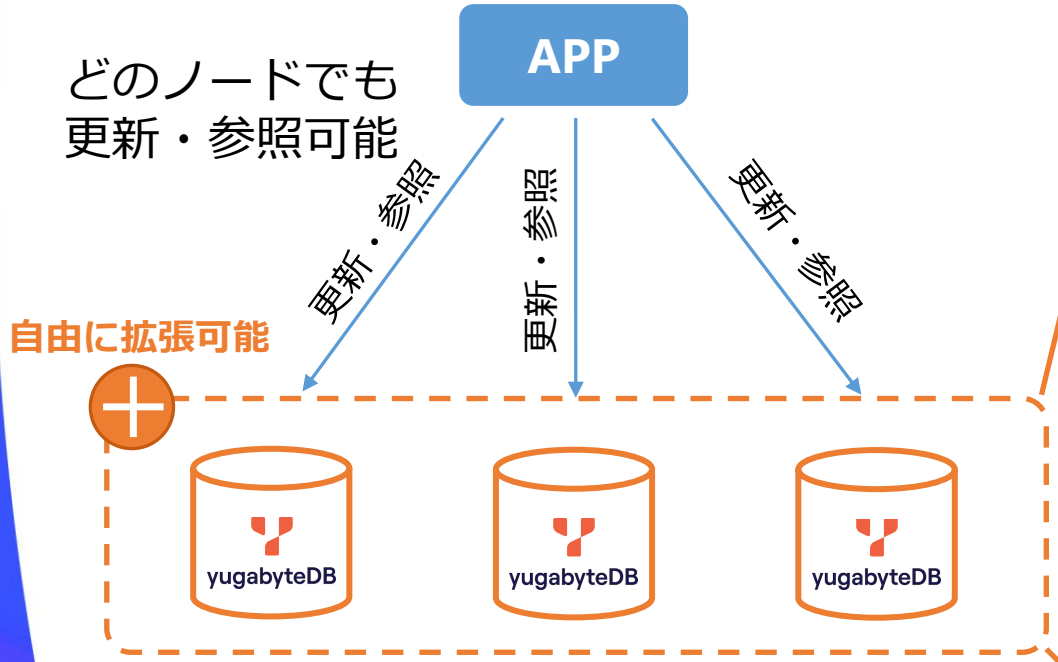
- データを複数台のPostgreSQLに分割配置
- postgres_fdwを利用してSQLの実行負荷を分散させることで、スケールアウトを実現
- 参照・更新ともにスケールアウト可能

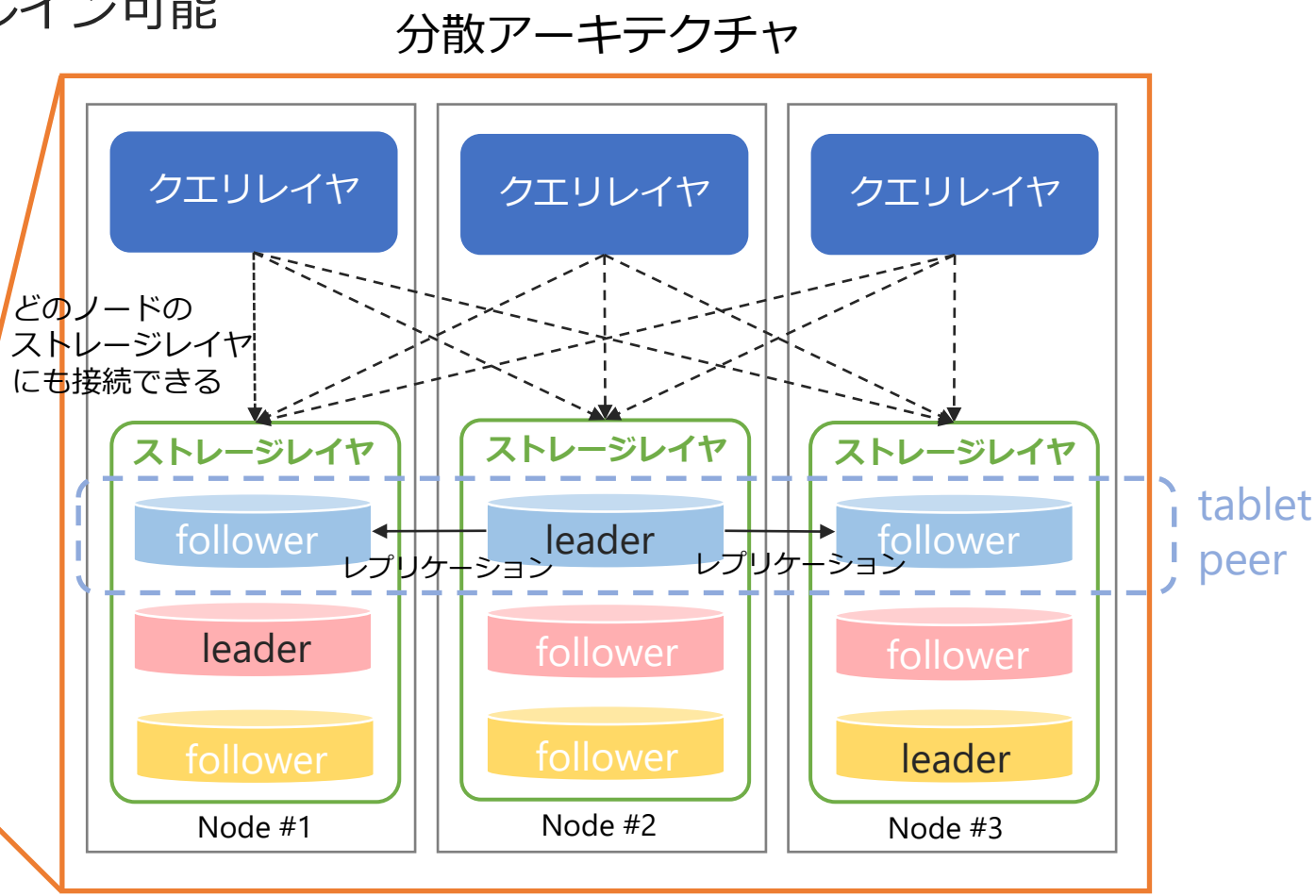


分散トランザクションが未対応

YugabyteDBのスケールアウト

- 分散アーキテクチャを採用している
  - クエリレイヤ：クエリを受け付けて処理する
  - ストレージレイヤ：データを処理・保管する
- 必要に応じてスケールアウト/スケールイン可能
- 参照・更新ともにスケールアウト可能
- 分散トランザクション対応

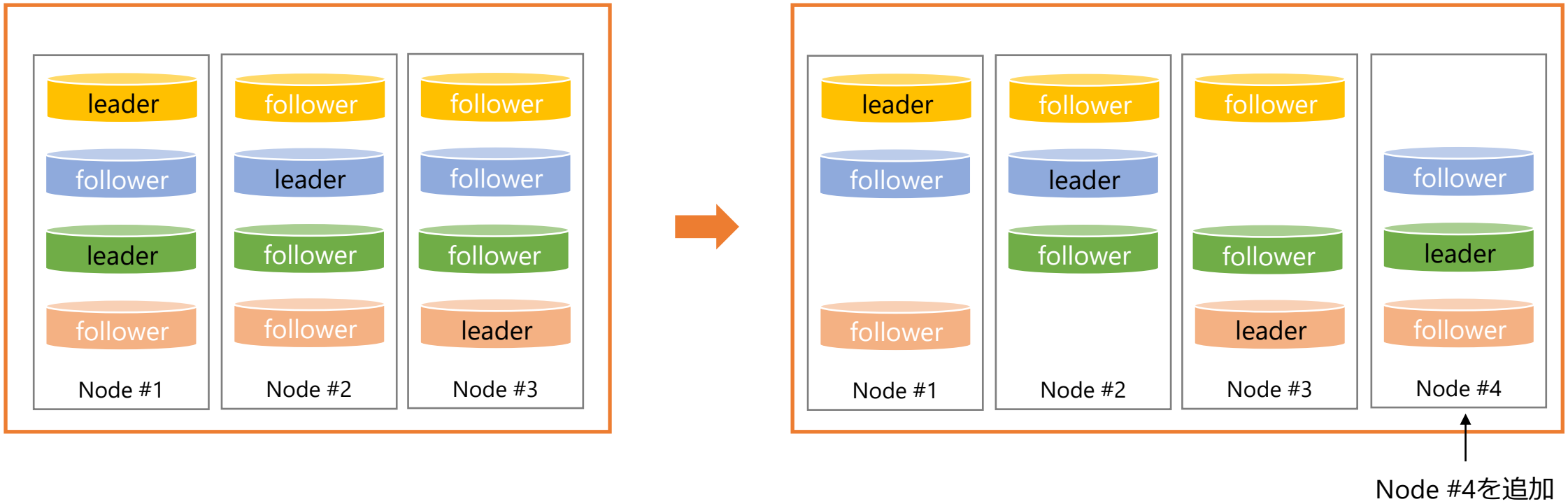




YugabyteDBのスケールアウト

ノードの追加

ノードを追加すると、Tabletのリバランスは自動で行われる



YugabyteDBを選択すべき理由 4

自動シャーディング

・RDBにおけるシャーディングの構築・管理の複雑さを軽減

YugabyteDBにおけるシャーディング

- YugabyteDBにおけるシャーディング（Tabletの分割）
  - テーブルのプライマリキーで**自動的に**行われる
  - テーブルを作成すると、デフォルトではノードごとに1つのTabletが作成される
    (Tabletの数を指定してテーブルを作成することも可能)
- Tabletのシャーディング方法は**Hash**または**Range**
- ホットスポットを防ぐためにHashシャーディングを使う

Hashシャーディングの場合

```
CREATE TABLE user_table (user_id int, name VARCHAR NOT NULL, PRIMARY KEY (user_id));
```

デフォルトではHASH
シャーディングとなる

Rangeシャーディングの場合

```
CREATE TABLE user_table (user_id int, name VARCHAR NOT NULL, PRIMARY KEY (user_id ASC));
```

Rangeシャーディングの場
合は、ASC/DESCを指定

YugabyteDBを選択すべき理由 5

地理的パーティショニング

地理的パーティショニング

地理的パーティショニングとは

- パーティショニングとtablespace機能を利用
- 特定のパーティションテーブルを特定の地域に紐づけて、行単位でデータの地理的な配置場所を制御

ユースケース

- グローバルに広がった複数の拠点にまたがってYugabyteDBをデプロイし、利用ユーザの近いところにデータを配置することで、よりレイテンシーを抑える
- 個人情報などの機密情報を特定の地域のデータセンターに保管することで、データの地理的な配置場所を制御できる

地理的パーティショニング

Step 1: tablespace作成

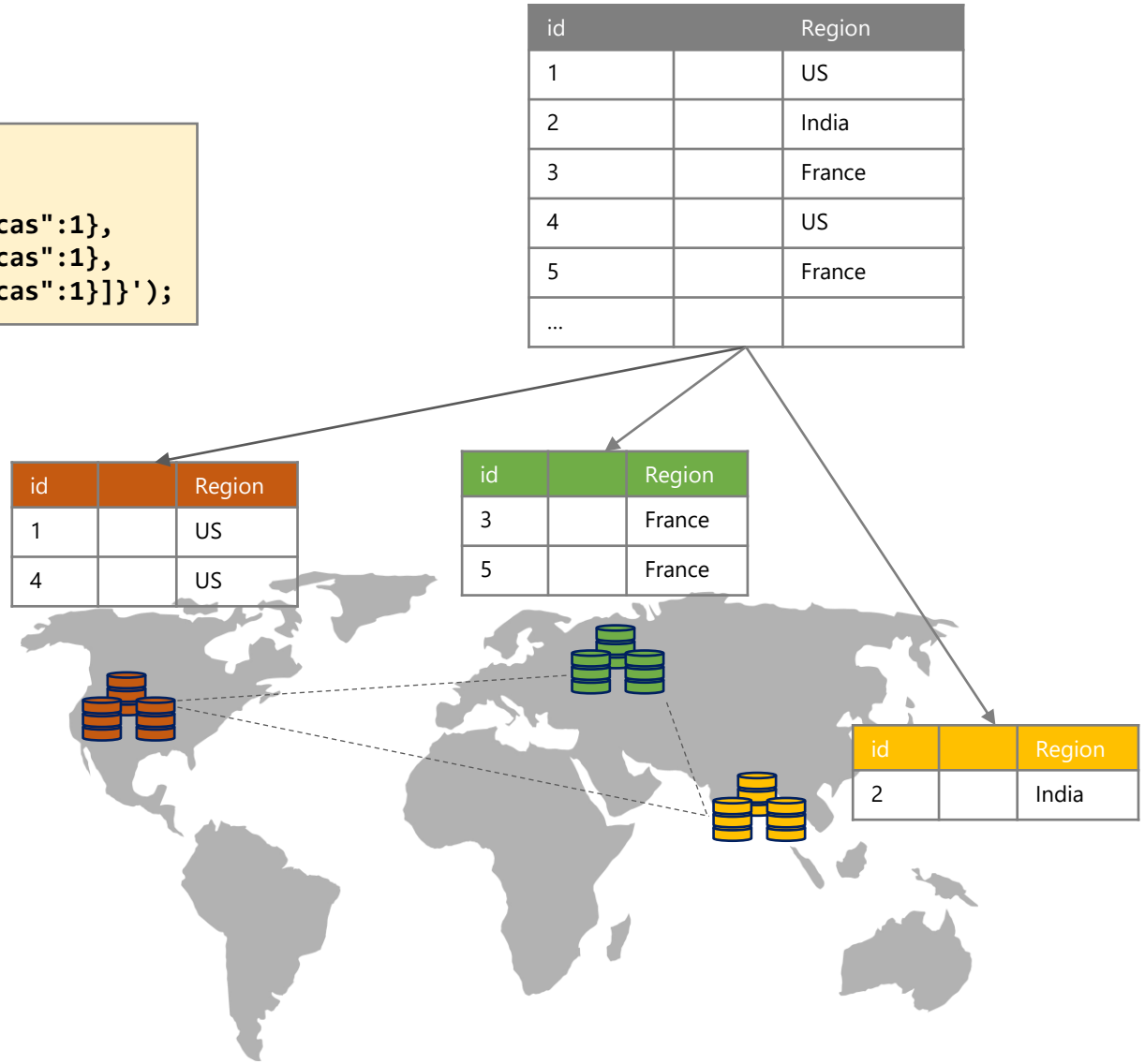
```
CREATE TABLESPACE us_west_2_tablespace WITH (
replica_placement='{"num_replicas": 3, "placement_blocks": [
{"cloud":"aws","region":"us-west-2","zone":"us-west-2a","min_num_replicas":1},
{"cloud":"aws","region":"us-west-2","zone":"us-west-2b","min_num_replicas":1},
{"cloud":"aws","region":"us-west-2","zone":"us-west-2c","min_num_replicas":1}]}');
```

Step 2: 親テーブル作成

```
CREATE TABLE users (
    id   INTEGER NOT NULL,
    ...
    region VARCHAR,
) PARTITION BY LIST (region);
```

Step 3: パーティションテーブル作成

```
CREATE TABLE users_us PARTITION OF users (
    id, ..., region,
    PRIMARY KEY (id HASH))
    FOR VALUES IN ('US') TABLESPACE us_west_2_tablespace;
```

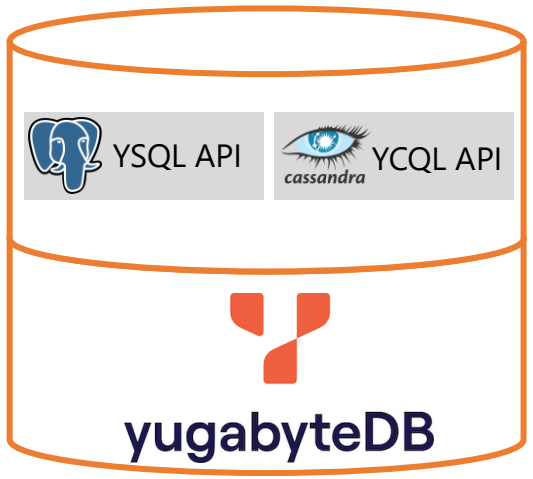


YugabyteDBを選択すべき理由 6

PostgreSQL互換、ACIDトランザクション対応

PostgreSQL互換、ACIDトランザクション対応

- PostgreSQLと高い互換性を持っている
- SQLを使って複雑な集計や検索が可能
- ACIDトランザクションをサポートしており、強力なデーター貫性を担保できる

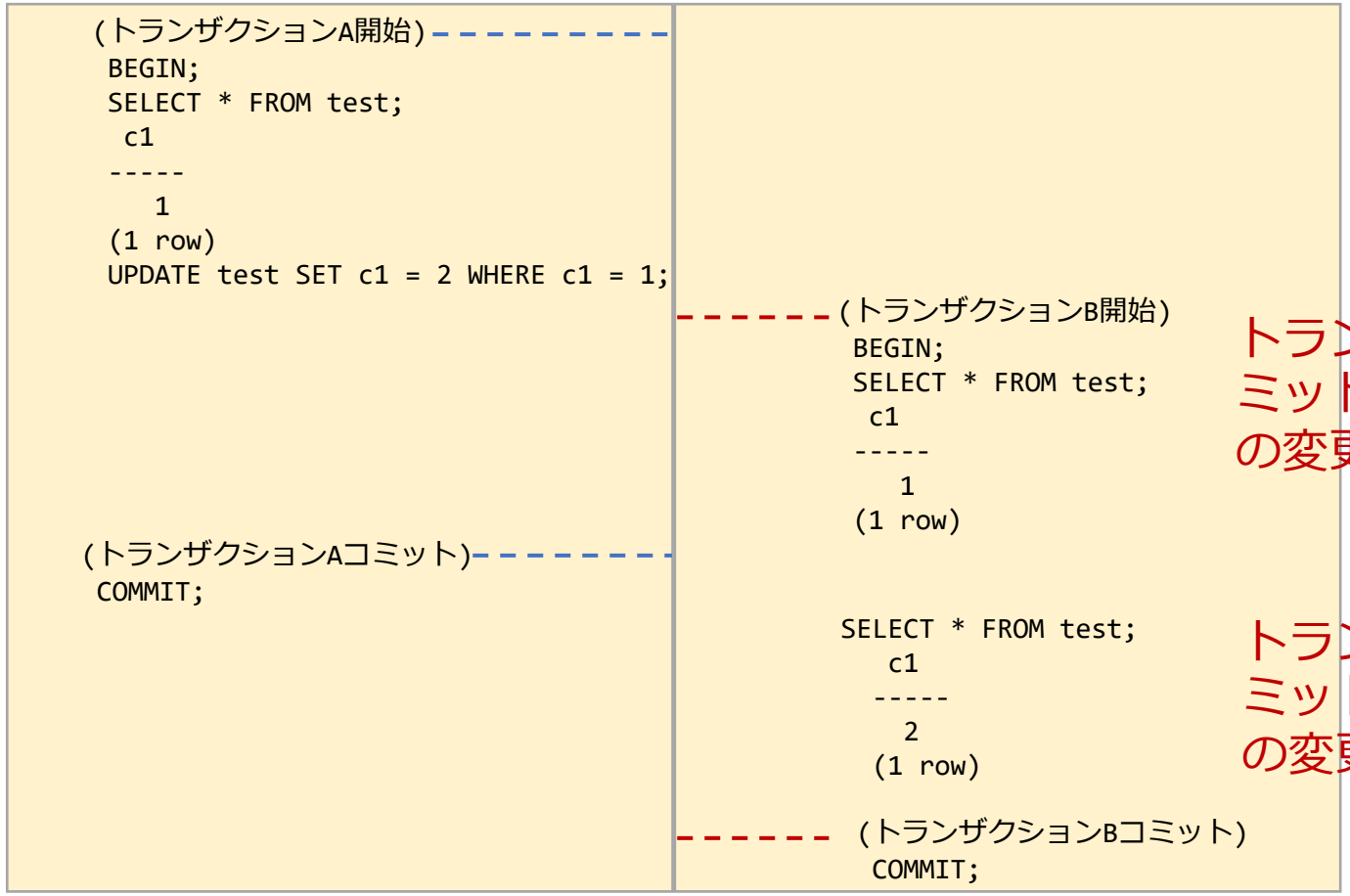


YugabyteDBを選択すべき理由 7

PostgreSQL MVCCの欠点をYugabyteDBで解決

PostgreSQL MVCCの仕組み

- PostgreSQLは**多版型同時実行制御(MVCC)**をサポートしている
  - MVCCとは、複数のトランザクションを同時に実行する際にデータの一貫性を保証する仕組み



トランザクションID (XID) とは

PostgreSQLでは、同時実行制御にトランザクションIDを利用している

- すべてのトランザクションに**XID**と呼ばれるトランザクションIDが割り当てられる
- XIDは単調増加する32bitの非負整数値
- 各タプルのヘッダに**xmin**、**xmax**と呼ばれるXIDが格納されている
  - `xmin`：タプルを挿入するトランザクションID
  - `xmax`：タプルを削除するトランザクションID

PostgreSQLのMVCCの課題

XID周回問題

- 利用できるXIDは約42億（2^32-1）のみ
- XIDが最大値に達し、周回すると、存在するはずのデータが見えなくなってしまう
- 防止するために、VACUUMが必要

YugabyteDBにおけるMVCC

YugabyteDBにおけるMVCC

- XIDを利用していない
- ハイブリッドロジカルクロック（HLC）を利用している
  - HLCは物理クロックと論理クロックを組み合わせたもの
  - 使い切ることはない

(physical component, logical component)

NTPを使用して同期された物理クロック

因果関係を示すLamportクロック

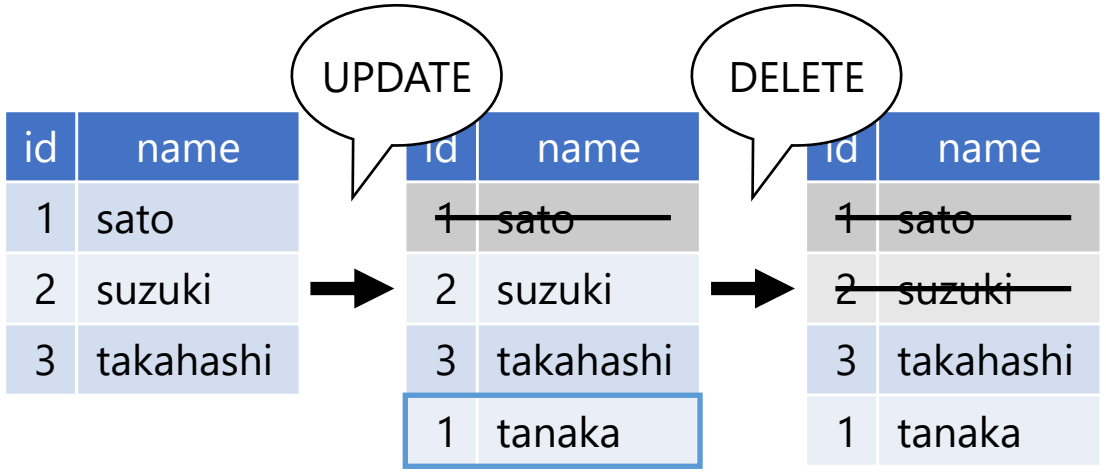
YugabyteDBを選択すべき理由 8

PostgreSQLの肥大化問題の回避

- **VACUUM不要**
- **自動コンパクション**

PostgreSQLにおける更新／削除の仕組み

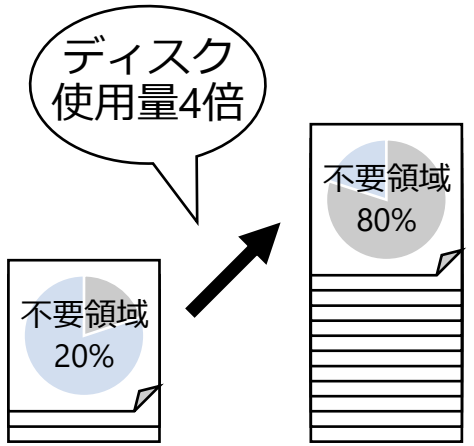
- PostgreSQLは追記型アーキテクチャを採用している
- UPDATE、DELETEで発生した不要領域が発生する



- UPDATEでは、更新前のデータに削除フラグをつけて、更新後のデータを新たに登録
- DELETEでは、削除対象のデータに削除フラグをつける

削除フラグのついた領域＝不要領域

- 不要領域が増えると、どうなるのか？
  - ファイルサイズが大きくなり、ディスク使用量が増える
  - メモリにキャッシュされにくくなり、ディスクアクセ スが増え、性能が下がる



1GBのデータを格納するのに、不要領域の割合が20%だと1.25GB必要なのに対して、80%だと５GB必要になる

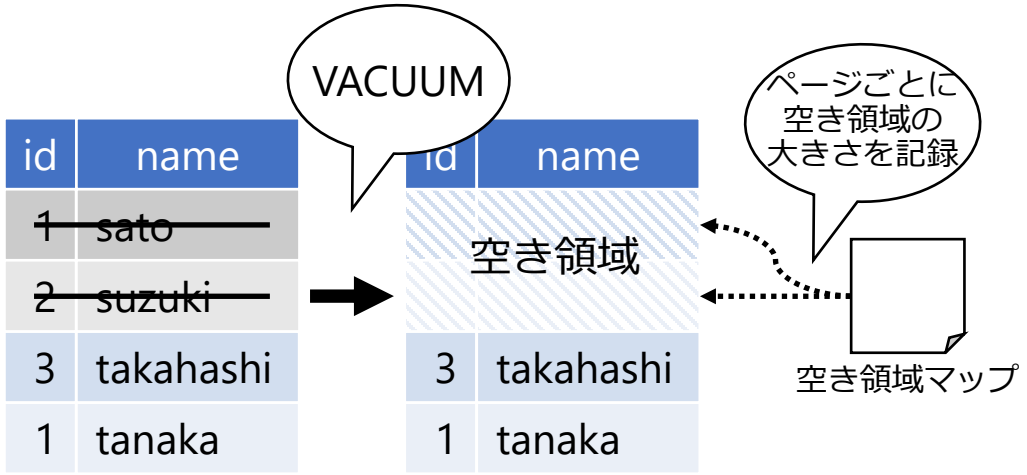
不要領域が増えないように
定期的なVACUUMの実行が必要

VACUUMとは

VACUUMはVACUUMとVACUUM FULLの2種類

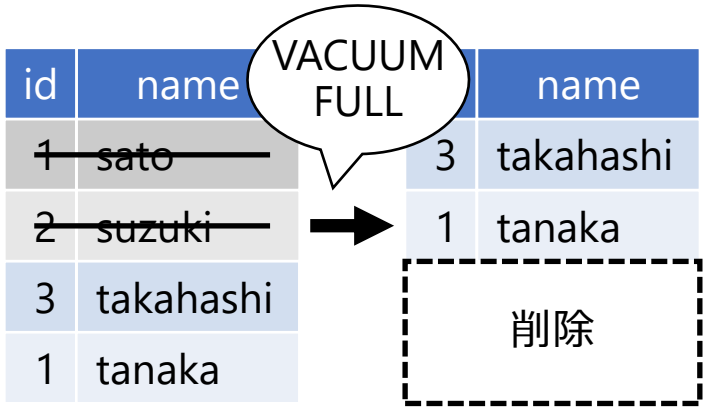
VACUUM

- 不要領域を空き領域マップ(FSM)に登録して、再利用可能にする処理
- ファイルサイズは基本的に小さくならない
- INSERT、UPDATE、DELETEと並行して実行できる



VACUUM FULL

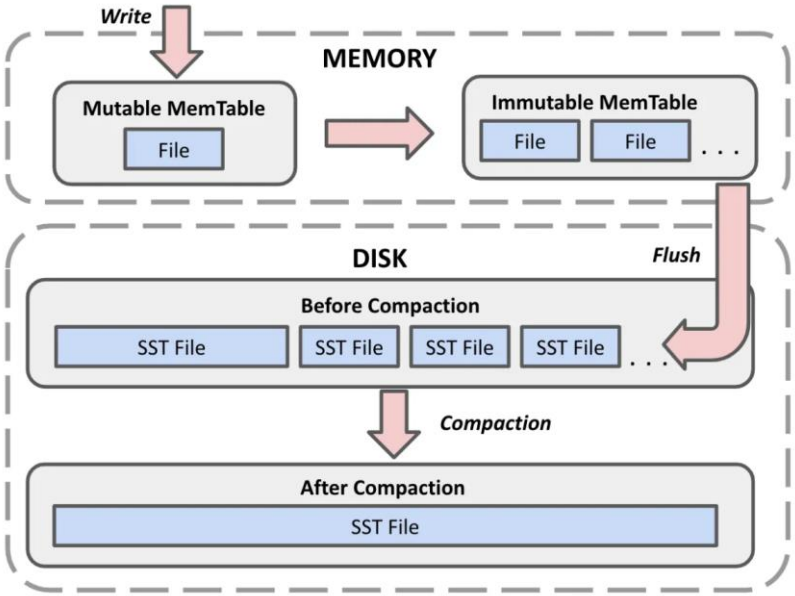
- 不要領域を削除する処理
- ファイルサイズが小さくなる
- SELECTとも並行して実行できない
- 対象テーブルと同程度の空き容量が必要
- インデックスも再構築される



自動VACUUMが適切に実行されるように、ワークロードの特徴に合わせて自動VACUUMの設定を調整することが重要

YugabyteDB自動コンパクション

- コンパクションとは
  - DocDBに書き込まれた複数のSSTファイルを1つのファイルにまとめる
  - 不要となったデータをディスクから削除し、ストレージの使用容量を削減



- データはまずメモリに書き込まれる
- メモリ上のデータが定期的にディスクにフラッシュされ、新しいSSTファイルに書き込まれる
- コンパクションにより複数のSSTファイルが１つのファイルに集約

- 読み取りパフォーマンスの向上
- ディスク容量の最適化

YugabyteDBを選択すべき理由 9

運用・管理コストの削減

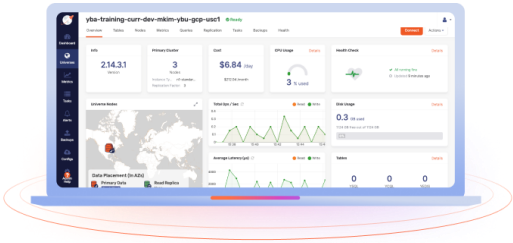
運用・管理コストの削減

ほとんどの運用管理タスクは管理コンソールで操作できる

アップグレード
無停止バージョンアップ

スケーリング
無停止ノード追加/削除
インスタンスタイプ変更

Day2運用が容易に



トラブルシューティング
実行中クエリの可視化
スロークエリの検出

バックアップ・リストア
自動バックアップ
オンデマンドバックアップ
リストア
PITR

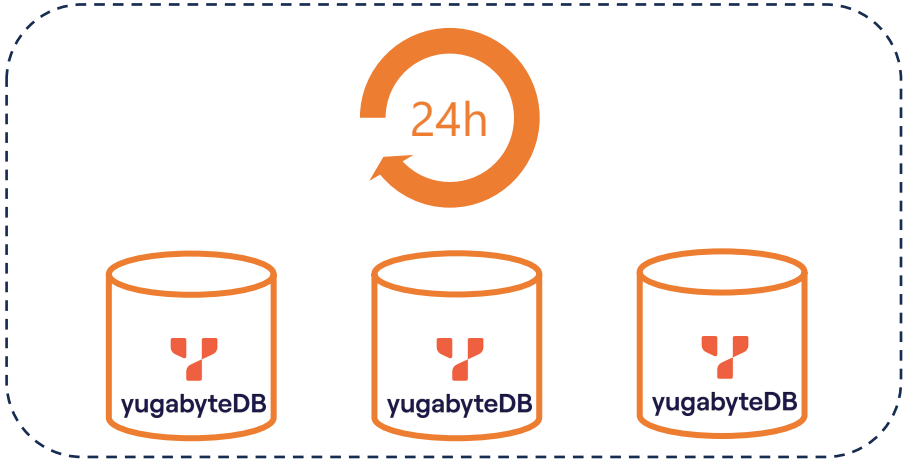
モニタリング
メトリクス収集・可視化
障害やリソース負荷超過時にアラート
通知メール送信
リソース使用の効率化
インデックスやスキーマの改善

YugabyteDBを選択すべき理由 10

ゼロ計画外・計画ダウンタイム

ゼロ計画外・計画ダウンタイム

- ユーザにノード障害を意識させることなく運用続行可能
- ダウンタイムなしで
  - スケールアウト/スケールイン
  - バージョンアップなどのメンテナンス



まとめ

- YugabyteDBは「一貫性」、「スケーラビリティ」、「高可用性」を兼ね備えたクラウドネイティブなデータベース
- YugabyteDBを使うことで、従来のRDBの以下の課題を解決できる
  - スケールアウトが困難
  - ダウンタイム
  - 障害時のデータ損失
  - XID問題、データの肥大化などのMVCCの課題
  - 運用・管理の複雑さ
  - など

ご清聴ありがとうございました。

製品・サービスに関するお問い合わせ: sales@sraoss.co.jp 03-5979-2701