PostgreSQL マルチマスタレプリケーション 徹底比較

2024-07-12　17:00-17:30

DB Tech Showcase [C20]

SRA OSS合同会社 高塚 遥

講演内容 / 講演者

- 講演内容
  - PostgreSQLにおけるマルチマスタレプリケーション利用について、
    現在の選択肢とそれらの実力、課題を解説します。

- 講演者
  - SRA OSS合同会社 高塚 遥
  - PostgreSQL のヘルプデスク、コンサルティング、設計構築、トレーナー等を
    15年以上にわたり担当
  - 特定非営利活動法人 日本PostgreSQLユーザ会 理事

会社紹介

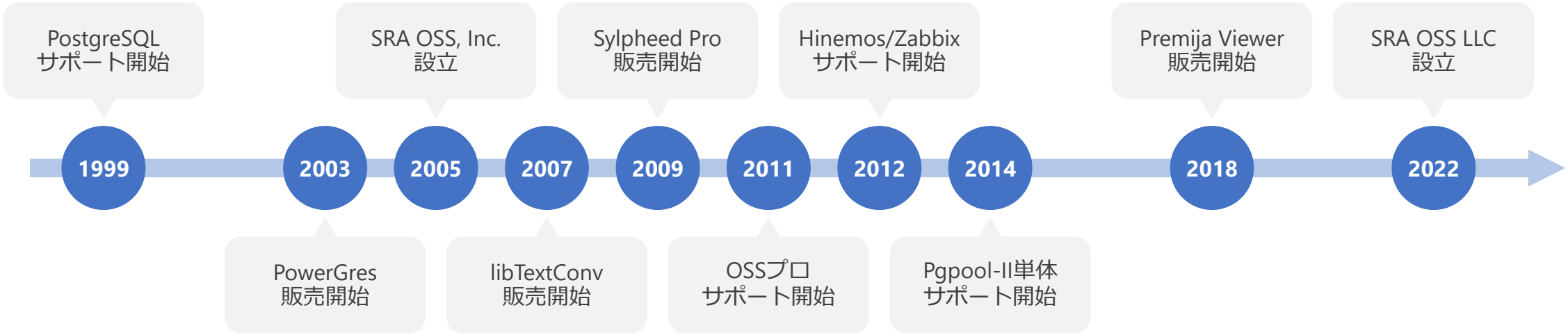
SRA OSS合同会社

所　在　地: 東京都豊島区南池袋2-32-8
設　立　日: 2022年6月17日
出　　資: 株式会社SRA
資　本　金: 7,000万円
社　　長: 稲葉 香理

事業内容

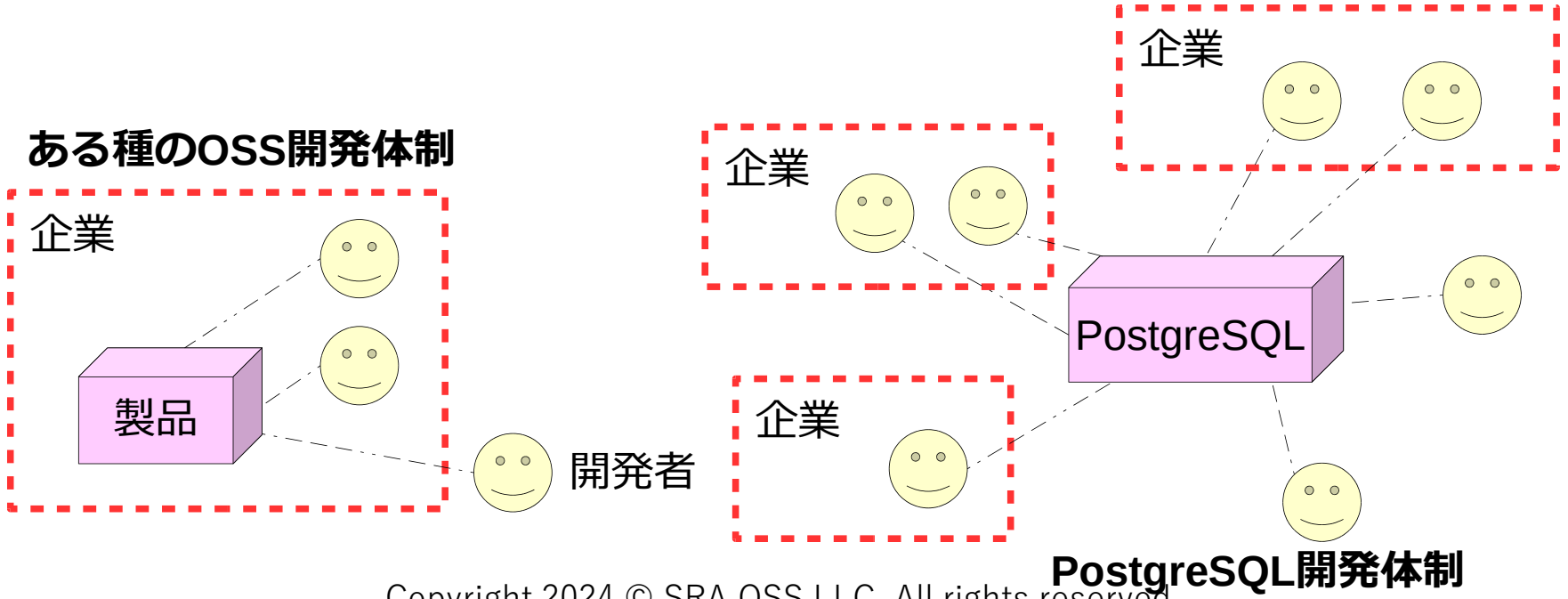
- オープンソースソフトウェア（OSS）関連のサポート、製品開発・販売、構築・コンサル
- OSSの教育、開発、コミュニティ運営支援
- ソフトウェアの研究開発

顧　　問: 石井 達夫
技術顧問: 増永 良文（お茶の水女子大学名誉教授）



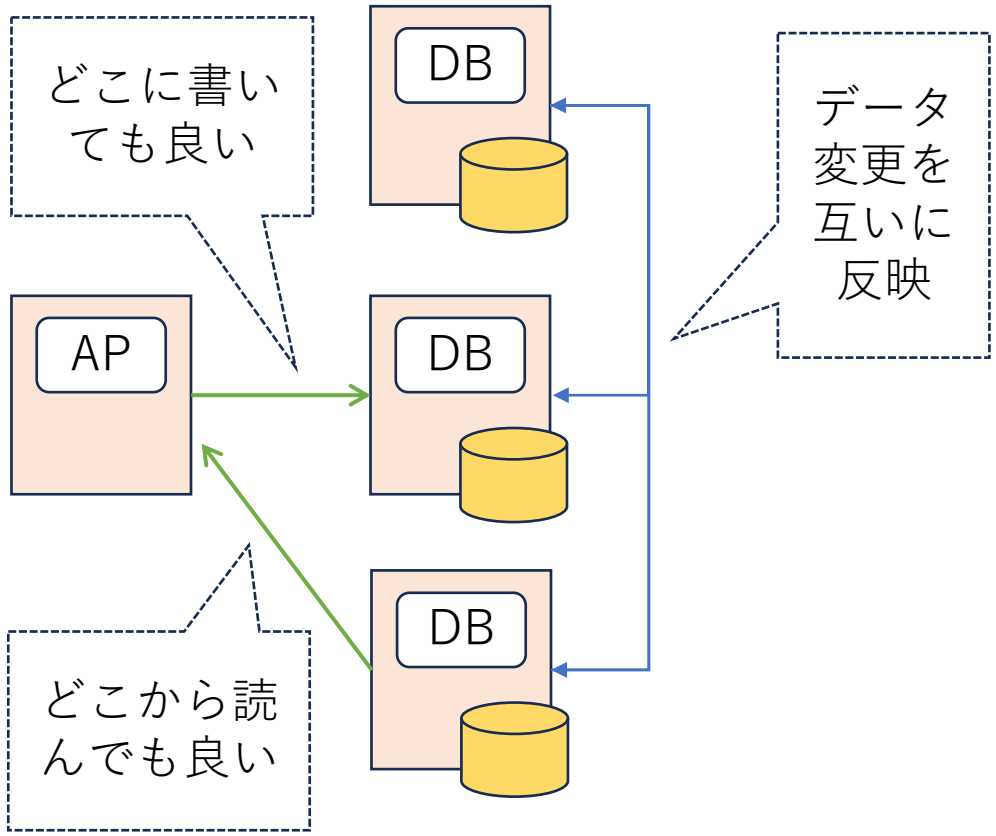
PostgreSQL とは

- 多機能、高性能、かつオープンソースの RDBMS
  - 伝統的な RDBMSで INGRES(‘70)、POSTGRES(‘80) から続く長い歴史
  - BSDタイプのライセンス、特定オーナー企業が無い
  - コア機能を小さく保つ傾向（HAクラスタやプランナヒントなどが外部拡張）
    - ***マルチマスタレプリケーションも永らく外部提供のみ***



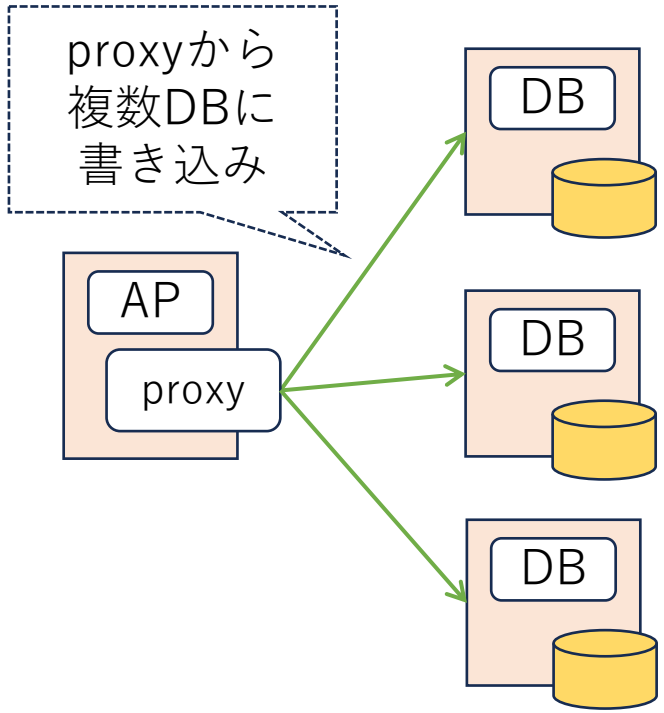
マルチマスタレプリケーションとは（1）

- 複数DBサーバがあって、どこに書いても良いし、どこから読んでも良い
- データ変更は他のDBサーバに自動的に反映される
- データ変更の他サーバへの反映を保証するかは、製品や設定による
  - 同期保証でコンフリクトはアボート
  - 同期保証でコンフリクトは適当に解決
  - 非同期でコンフリクトは適当に解決
  - 非同期でコンフリクトはエラー　　… etc.
- 大抵は Eventually Consistency
  - 一貫性のある読み取りは通常保証できない

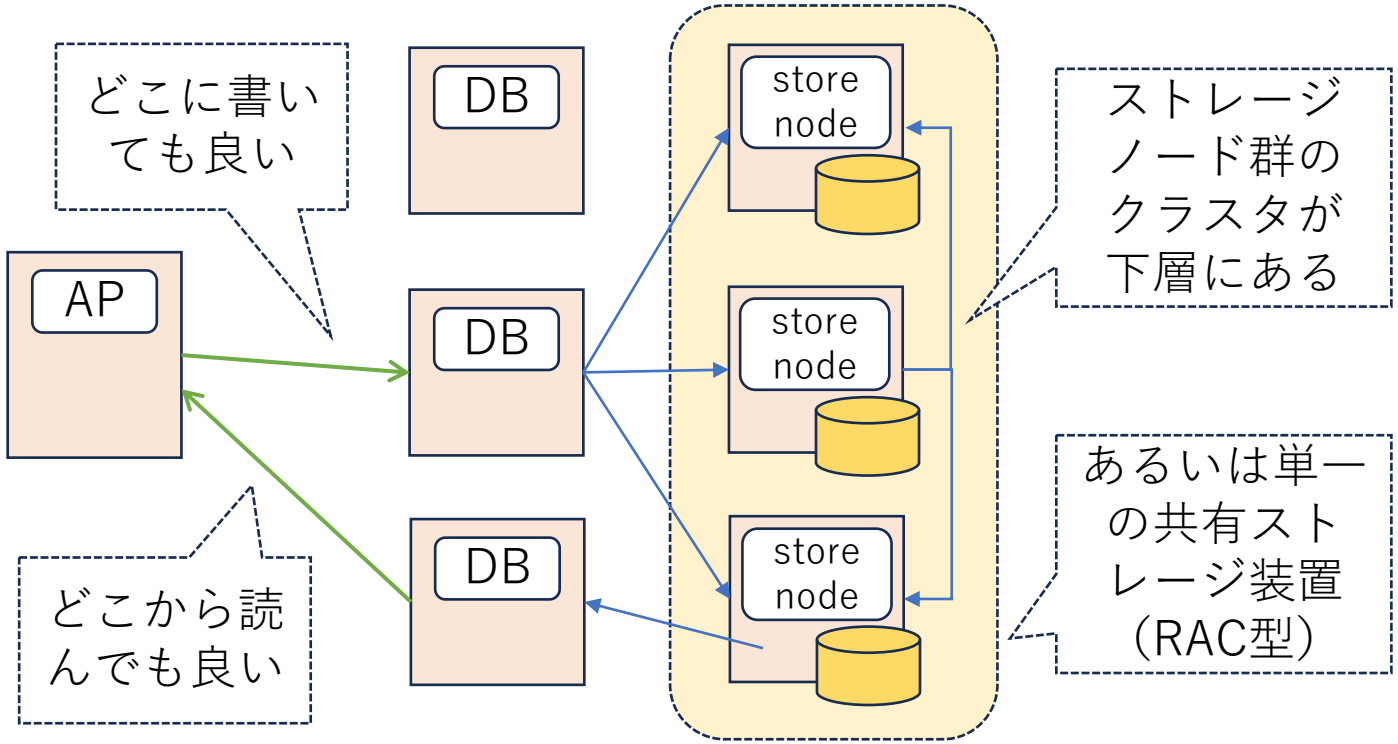


マルチマスタレプリケーションとは（2）

- 下記のような構成をマルチマスタと呼ぶこともありますが、本講演では対象外とします。



Pgpool-II のいくつかのモードや、
TPモニタ製品による2相コミット同期



YugabyteDB などの別アーキテクチャDBMS、
Oracle RAC（共有ストレージ装置の場合）

マルチマスタ構成のメリット・デメリット

メリット

- フェイルオーバが高速で単純
  - 単に壊れたノードを捨てれば良い
  - 「昇格」が不要
- 書き込みを分散できる
  - 特に、並行する書き込みで、それぞれ更新するデータ項目が異なっている場合に
  - 特に、同期しないデータもある場合

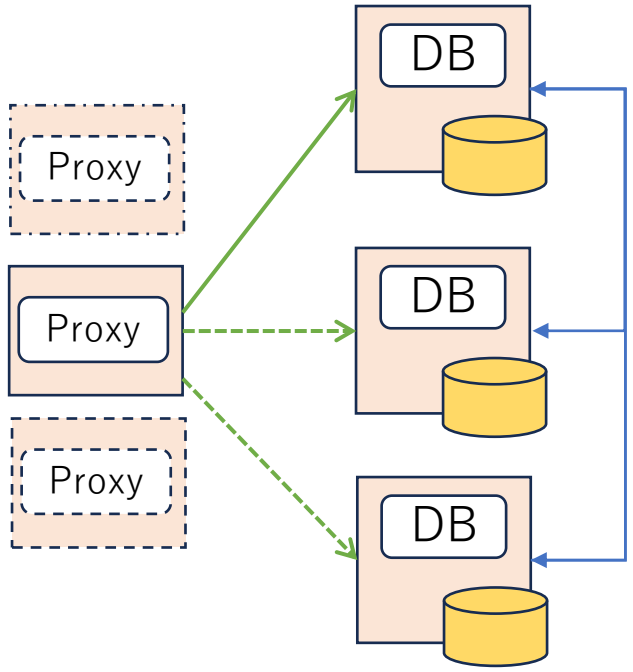
デメリット

- コンフリクトがある
  - 異なるノードで同データを操作
  - 異なるノードで同時に発番
  - 結果としてデータ不一致や、レプリケーション詰まりが起きる
  - これらへの「対策」が要る
- （しばしば）遅い
  - シングルマスタで可能な方式の方が低負荷高速な選択肢が多い

マルチマスタ構成のユースケース(1)

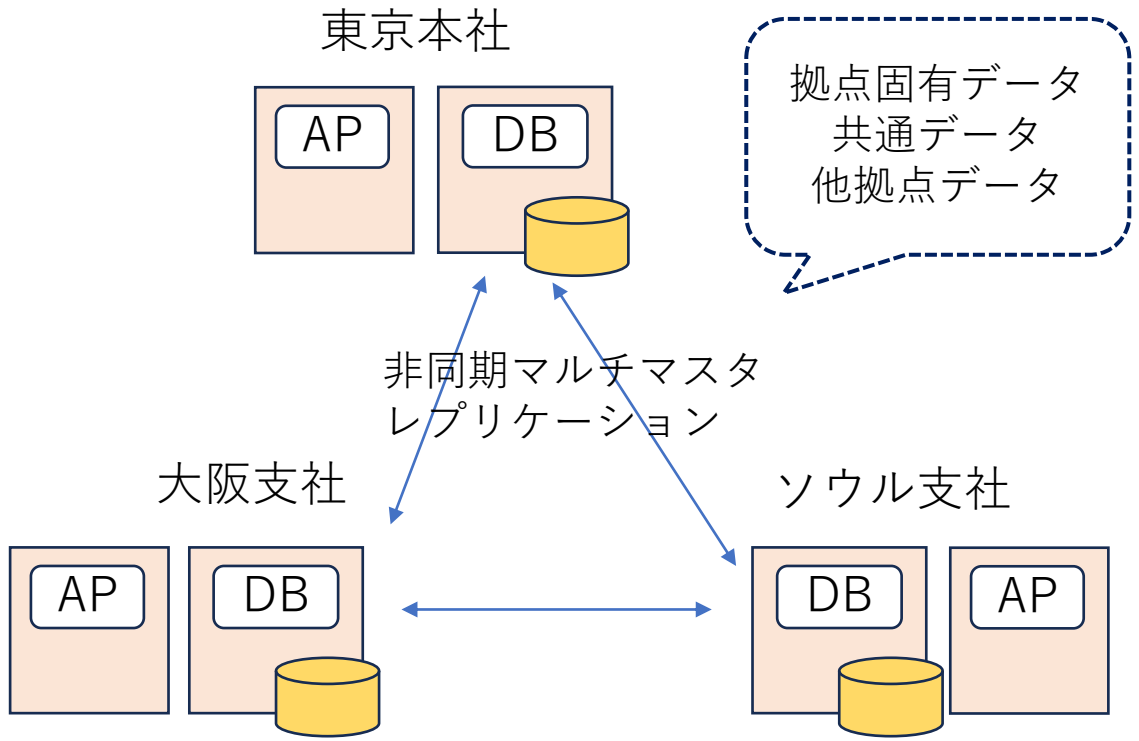
- 冗長化クラスタ
  - シングルマスタレプリケーションと基本的には同じ
  - 振り分けソフトウェア:
    - クライアントライブラリ(libpq や pgJDBC)で複数接続先指定
    - Proxyサーバを使う (Pgpool-II、pgcat、HA Proxy + patroni、PGD Proxy)
  - 書き込みを特定1台サーバにすることでコンフリクト防止
  - 障害発生時は「捨てるだけ」なのでダウンタイムを極小化

- 性能スケールアウト
  - 構成としては冗長化と同じ
  - 参照を負荷分散する - 遅延/不整合データ参照に対策が要る
  - 更新を負荷分散する - 一般には、それほど効果的ではない

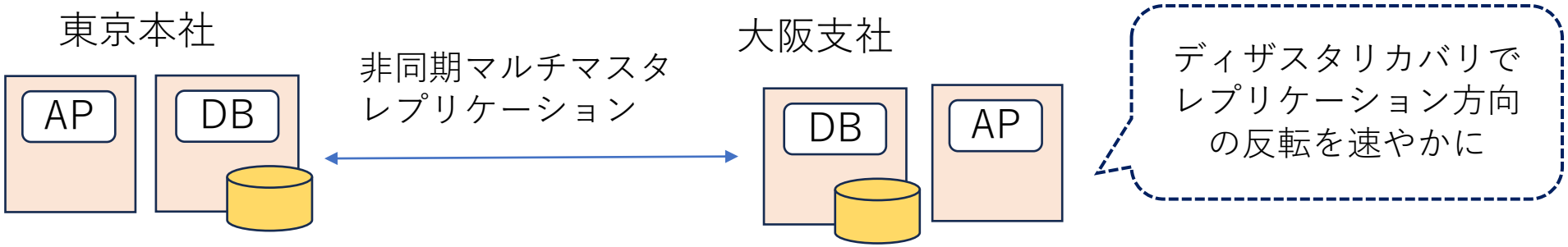


マルチマスタ構成のユースケース(2)

- 拠点間のデータ連携
  - 主に各拠点固有のデータを読み書き
  - 通常、共通データは参照のみ
    - 更新の頻度は低い／競合を避ける運用
  - 原則、他拠点データは参照のみ



- ディザスタリカバリ目的
  - 基本的にはレプリケーションは片方向
  - データ破損を伴わない切り替えに際して柔軟かつ速やかな逆向きレプリ再開



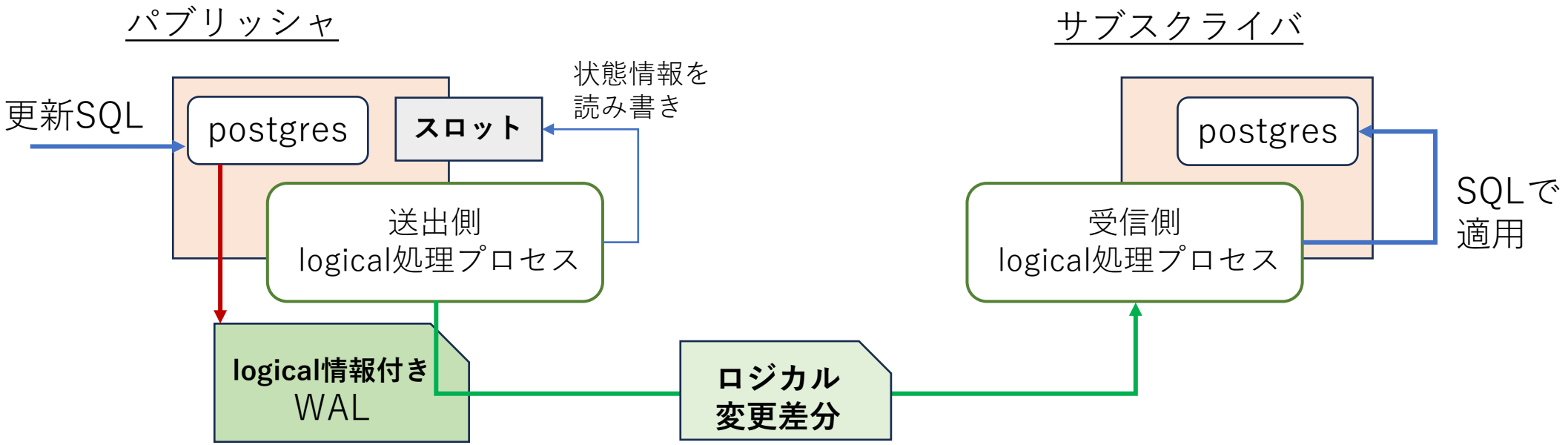
PostgreSQLむけマルチマスタレプリケーション

	指定単位・転送単位	方式	コンフリクト対策	ライセンス
PostgreSQL 16 以降のロジカルレプリケーション	テーブル・行	ロジカルデコーディング	なし	OSS
pgEdge	テーブル・行	ロジカルデコーディング	あり	OSS
EDB Postgres-Distributed（旧BDR）	テーブル・行	ロジカルデコーディング	あり	商用
Bucardo	テーブル・行	トリガ	あり	OSS
SymmetricDS (RDB汎用)	テーブル・行	トリガ	ほぼなし	OSS

本講演ではこれらを特に取り上げます

他に商用製品でRDB汎用のものがいくつかある

ロジカルデコーディングによるレプリケーション



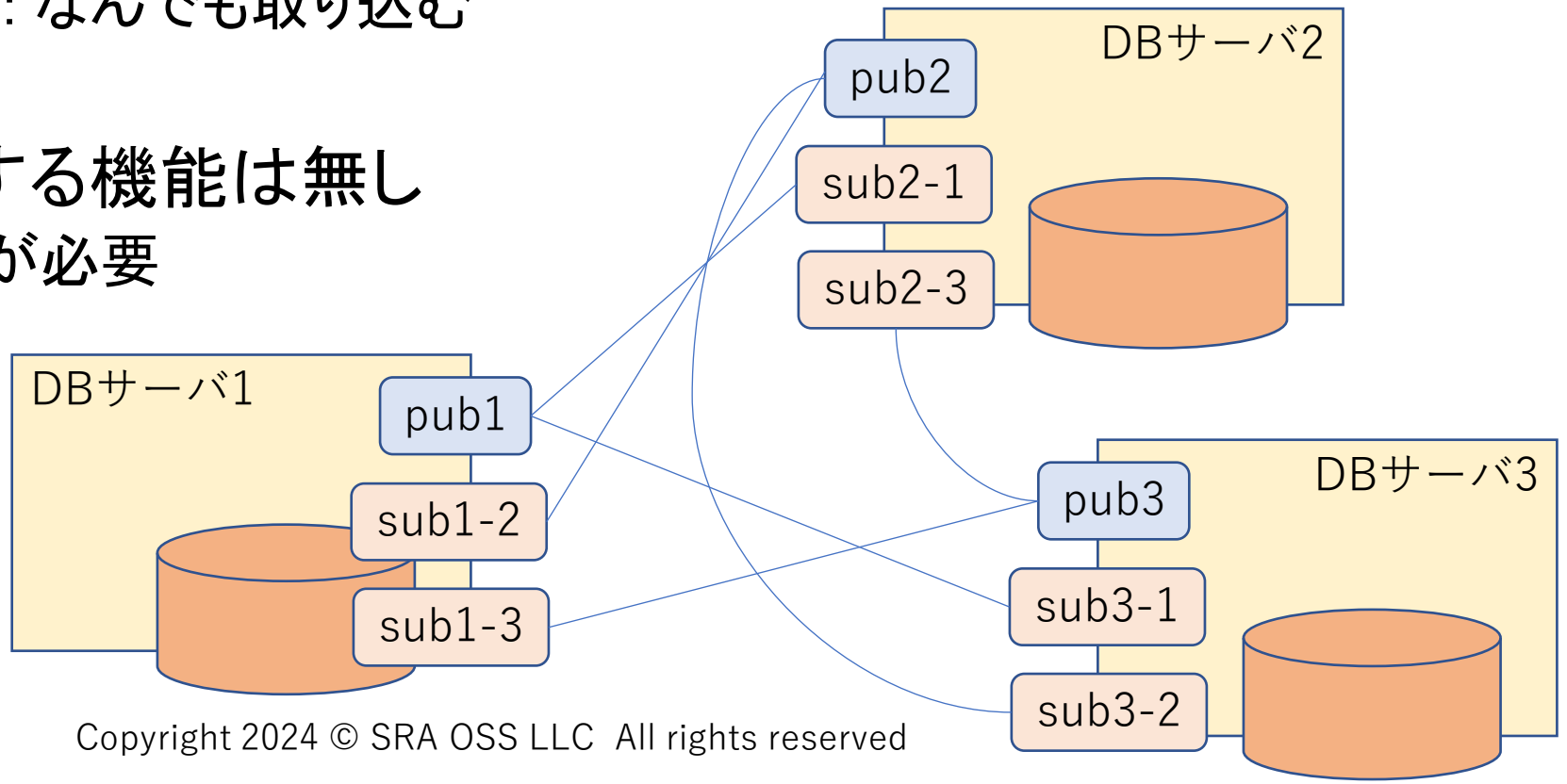
- 基本的なアーキテクチャは各ソフトウェアとも大差ない
  - 差異は：送受信形態、どこで溜めるか、並列送信するか、並列適用するか

主要機能の比較

	シーケンス対応	ラージオブジェクト同期	DDL同期	コンフリクト自動解決	列・行フィルタ	振り分けProxy
PostgreSQL 16のロジカルレプリケーション	×	×	×	×	○	× 別途用意 (Pgpool-II、HA-proxyなど)
pgEdge (spock v3.3)	○ (snowflake sequence／レプリも可)	○ (lolerによる)	○	○	○	△ 既存ソフトに対応 (pgcat、HA-proxy など)
EDB Postgres-Distributed (PGD) (v5)	○ (PGD global sequence／snowflakeも可)	×	○	○	○	○ 統合された専用ツール PGD Proxy

PostgreSQL16 のマルチマスタ(1)

- サブスクリプション origin = none 指定
  - オリジンがある更新データは取り込まない – 更新のピンポンを防止
    ⇒ マルチマスタ構成が可能になった
  - デフォルトは any : なんでも取り込む
- 更新の衝突に関する機能は無し
  - 利用者側で対策が必要



PostgreSQL16 のマルチマスタ(2)

- 更新の衝突は実際どの水準？ OLTP を分散実行したら？
  - データがすぐに一貫性を欠き壊れる？ → 発生する
    - 他ノードには行ロックは効かないし、直列化維持のためのアボートも生じない
      - 行ロック付き口座振替トランザクションの同時実行で各行データ不一致、合計金額も不一致
  - ノード間でデータが異なってしまう？ → これもある
    - 同データ同時並行の更新でノード間の最終状態が一致しない
      - ノード毎に異なった「後勝ち」が生じうる
  - すぐにエラーで止まってしまう？ → これもある
    - 主キー重複や外部キー制約違反の自動解決ができない
    - 典型的には同キー行の同時挿入で止まる

pgEdge (1)

- オープンソースソフトウェア – pglogical から派生
  - 商用クラウドサービス提供あり
- 中核の拡張spock と関連拡張とで構成
- 他OSS拡張と共存可 – citus、timescale、orafce – 組み合わせ利用のためのパッチも提供
- いくぶん開発途上感がある – 例えば pii
- カタログがシンプル – 概念名称はSlony-I と似ている （EDB PGD では 3倍以上ある）

```
db1(5432)=# SELECT * FROM spock. (TAB補完)
spock.channel_summary_stats  spock.node                    spock.replication_set_table
spock.channel_table_stats    spock.node_interface          spock.resolutions
spock.conflict_tracker       spock.pii                     spock.resolutions_id_seq
spock.depend                 spock.pii_id_seq              spock.sequence_state
spock.lag_tracker            spock.queue                   spock.subscription
spock.local_node             spock.replication_set         spock.tables
spock.local_sync_status      spock.replication_set_seq
```

pgEdge (2)

- DDL同期
  - 新テーブルの自動レプリケーションセット追加はトリガを作って実現
- Snowflakeシーケンス
  - ノード番号 ＋ タイムスタンプ ＋ ローカル連番 で発番して競合を回避
  - 連番にはならない
  - 基本的にノードで一つのシーケンスを使う
- ラージオブジェクト同期 loler
  - ラージオブジェクトのカタログテーブルを置き換えして、同期する
  - OID は snowflakeシーケンスと同じ要領で競合を回避して発番

pgEdge (3)

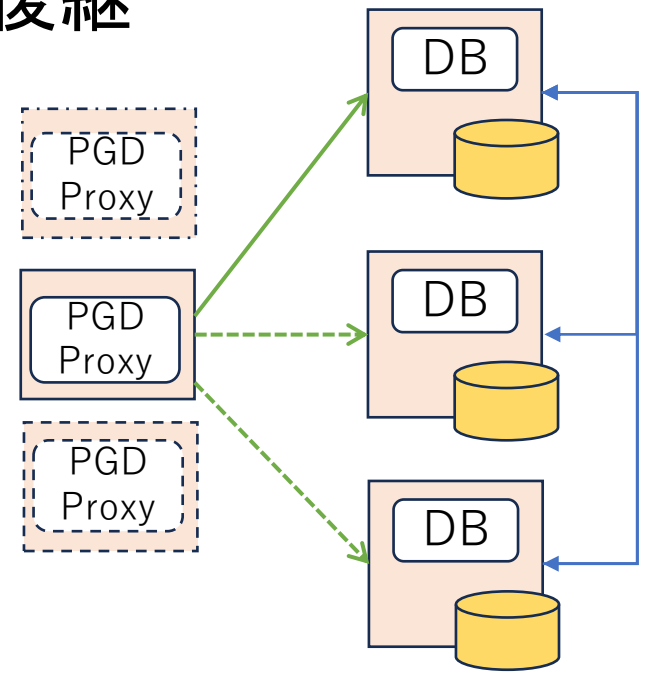
- コンフリクト自動解決 – spock.conflict_resolution設定:
  - error, keep_local, apply_remote, first_update_wins, last_update_wins,

- Conflict-Free Delta-apply
  - 列の現在に差分を適用する場合の競合を高速回避
    - UPDATE accounts SET balance = balance + 10000 WHERE aid = 1234 などに
    - ALTER TABLE accounts ALTER abalance SET (log_old_value = true) と設定
    - 2ノード、3ノードで「行ロック付き口座振替トランザクションの同時実行」をすると・・・
      → 各ノードの行データは完全一致、合計金額も初期状態と一致
      → オーバーヘッドほぼ無し

- Anti Chaos Engine
  - 既に発生してしまった不整合を見つけて、データ修復するツール
    - データ比較
    - 正とするノードを指定してデータ修復

EDB Postgres Distributed (1)

- クローズドソース製品
- EDB社の PostgreSQLビルド、Postgres Extended/Advanced Server で利用可能
- Postgres-BDR というオープンソースソフトウェアの後継
- 専用の PGD Proxy が統合されている
  - フェイルオーバ／ノード復旧 などの高レベル操作
- 商用クラウドサービスあり
  - EDB BigAnimal「distributed high-availability cluster」
    - 優れたGUI … ノード間同期遅延状況が参照できる
    - 個々ノードへの接続手段なし

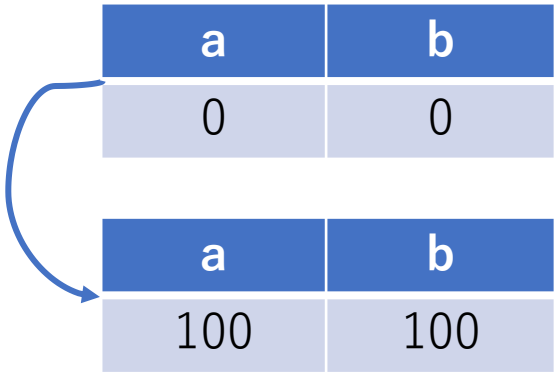


EDB Postgres Distributed (2)

- DDL同期
  - テーブル自動追加も可能
- グローバルシーケンス
  - アプリケーション透過的に使える同番発番しないシーケンスが利用可能
    - snowflakeid
    - galloc (globally allocated range) : ノード合意して発番可能値の範囲を確保
- PGD AutoPartition
  - PGDと統合された自動パーティショニング
    - パーティション自動追加、自動削除、自動レプリケーション設定

EDB Postgres Distributed (3)

- きめ細やかなコンフリクト自動解決
  - コンフリクトタイプ(13種)ごとに解決方法(11種) を指定
    - insert_exists → update_if_newer、delete_missing → skip など
  - 列レベルの解決
    - 同時に「UPDATE t SET a = 100」「UPDATE t SET b = 100」



  - CRDT（Conflict-free replicated data types）
    - 差分 UPDATE に強い専用の数値データ型が用意されている
      - CREATE TABLE accounts (id int PRIMARY KEY, balance bdr.crdt_delta_counter)
      - UPDATE accounts SET balance = balance + 10000 WHERE aid = 1234
      - テクノロジーとしては pgEdge の Conflict Free Delta Apply と類似
  - Eager Conflict Resolution
    - コミット時の他ノードで起きうるコンフリクトを調べて、アボートさせる
      - Group Commit コミットスコープのルールとして指定

EDB Postgres Distributed (4)

- Timestamp-based snapshot
  - タイムスタンプに基づいた「指定時点」のデータを取り出す
    - bdr.timestamp_snapshot_keep = 10s　…過去版行を保持する時間
    - SET snapshot_timestamp TO '2024-07-12 17:00';
    - SELECT …
- コミットスコープ
  - 変更のコミットを各ノードにどこまでどう反映させるかの設定
  - 4つの種別と、ルール、Confirmation Level を指定
    - Group Commit　データ損失を無くす／整合的な書き込み／内部で 2PC 使用も
    - CAMO (Commit At Most Once)　アプリ側のやり直しを不要に
    - Lag Control　遅延時間や遅延データ長で制御
    - PGD Synchronous Commit　同期レプリケーション／伝搬してからローカル応答

Oracle Database ／ 汎用RDBむけ商用製品

- Oracle Database でマルチマスタ論理レプリケーションは撤退傾向
  - アドバンスト・レプリケーションは 12c までで終了
    - 更新可能マテビューでの双方向レプリケーション
    - マルチマスタレプリケーション
  - Oracle Streams レプリケーションも 18c までで終了
  - ロジカル・マルチマスタレプリケーションは GoldenGate に集約？
- 汎用RDBむけ論理レプリケーション商用製品
  - GoldenGate
  - SharePlex
  - Synti Replication

    → コンフリクト対策もそれぞれある。PostgreSQL専用ソフトウェアと違って、
    専用データ型、特別な ALTER 文による設定、といった手段は提供されない。

まとめ

- PostgreSQL16 ネイティブ機能のみでマルチマスタは困難
- 今のところの実用用途は「比較的安全な使いかた」に限られる
  - 優れたコンフリクト/不整合対策の機能が用意されているにもかかわらず、製品としては「更新は 1つのノードに向ける構成」を推している傾向
    - コンフリクト/不整合 を「アプリケーションから見て」「正しく解決」は難題
    - 同時更新の可能性がある箇所があってもいいけど、限定的にして使うべし
- マルチマスタ論理レプリケーション固有のメリットはある
  - 障害切り替えを極小ダウンタイム
  - 同期方向切り替えをゼロダウンタイム
  - 「論理レプリ」であることによる柔軟性
    - 一部テーブルのみ同期、極力全データ再コピーを回避する運用、など

SRA OSS では

- PostgreSQL等の OSSのサポート(ヘルプデスク)を提供しています
  - 「PostgreSQL/PowerGres サポート＆保守サービス」
    - 年単位 – 対象サーバ単位 – 問い合わせインシデント無制限
  - その他さまざまなメニューがあります
- PostgreSQL等のコンサルティングを提供しています
  - 初期設計支援
  - 性能チューニング
  - 商用DBMSからのマイグレーション支援
- EDB社 Postgres製品の取り扱いをしております
  - EDB Postgres Distributed 導入のご相談もぜひ