SRA OSS, INC.

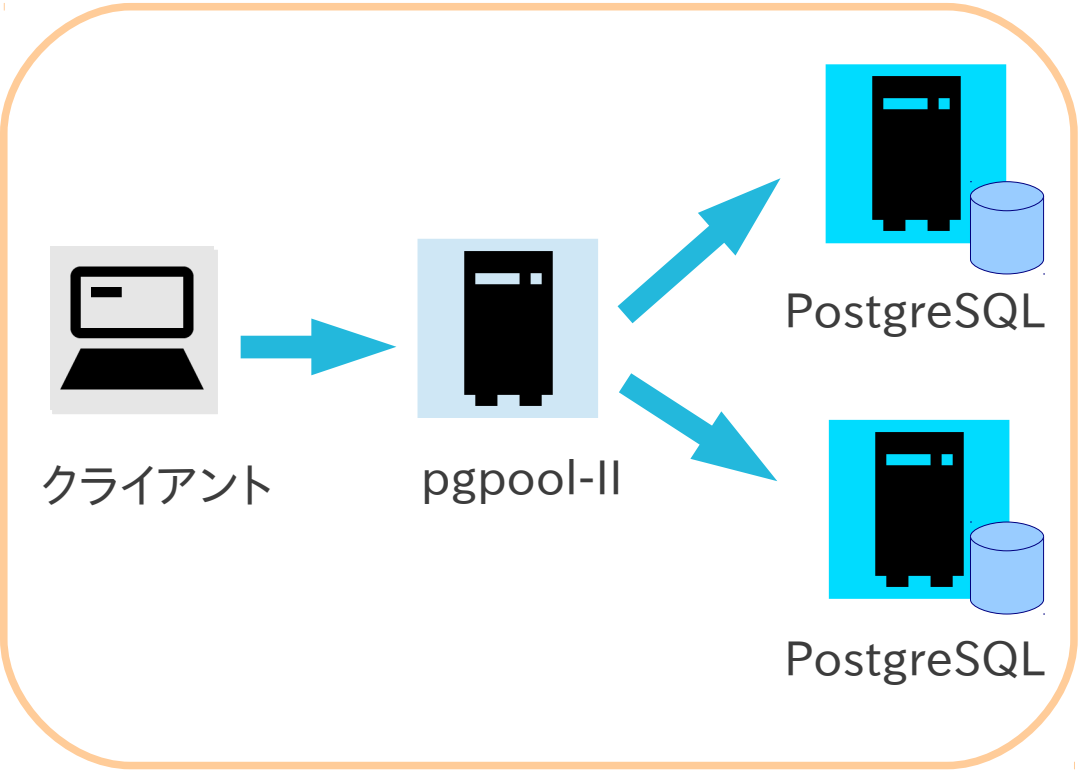
pgpool-II で PostgreSQL の性能と信頼性を高めよう

PgDay 2012 Japan
2012/11/30

SRA OSS, Inc. 日本支社
技術開発部　長田 悠吾

pgpool-II

- アプリケーションと PostgreSQL の間に介在し、クラスタリング機能を提供するミドルウェア
- オープンソースソフトウェア（BSDライセンス）
- 機能
  - コネクションプーリング
  - ロードバランス
  - オンメモリクエリキャッシュ
  - 自動フェールオーバー
  - レプリケーション
  - 組み込みHA（watchdog）



pgpool-II

- アプリケーションと PostgreSQL の間に介在し、クラスタリング機能を提供するミドルウェア
- オープンソースソフトウェア（BSDライセンス）
- 機能
  - コネクションプーリング
  - ロードバランス
  - オンメモリクエリキャッシュ

→ 性能を向上

  - 自動フェールオーバー
  - レプリケーション
  - 組み込みHA（watchdog）

→ 信頼性を向上

バージョン 3.2 の新機能！

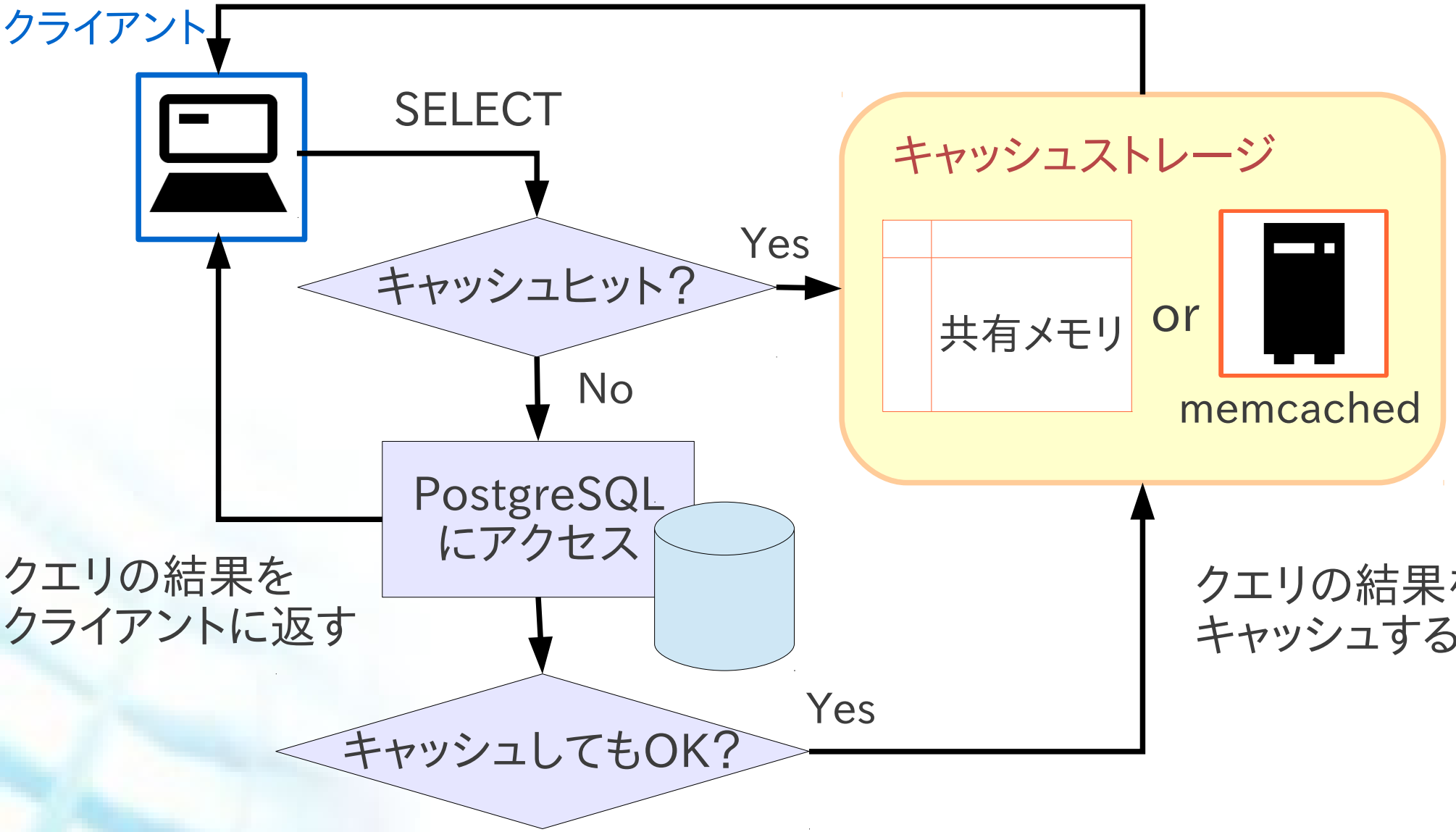
1. オンメモリクエリキャッシュ

オンメモリクエリキャッシュの基本機能

- SELECT 文の検索結果をキャッシュに保存
  - 次回以降、同じクエリが来たら PostgreSQL にアクセスせずにキャッシュされている結果を直接返す → 反応速度の向上!
- キャッシュストレージは以下から選択
  - 共有メモリ
    - アクセスが高速だが、サイズに制限あり
  - memcached
    - 容量の設定が比較的自由だが、ネットワーク通信のオーバーヘッドあり
- キャッシュの自動無効化
  - キャッシュ対象のテーブルが更新されたとき
  - キャッシュの有効期限が切れたとき

オンメモリクエリキャッシュ全体概要

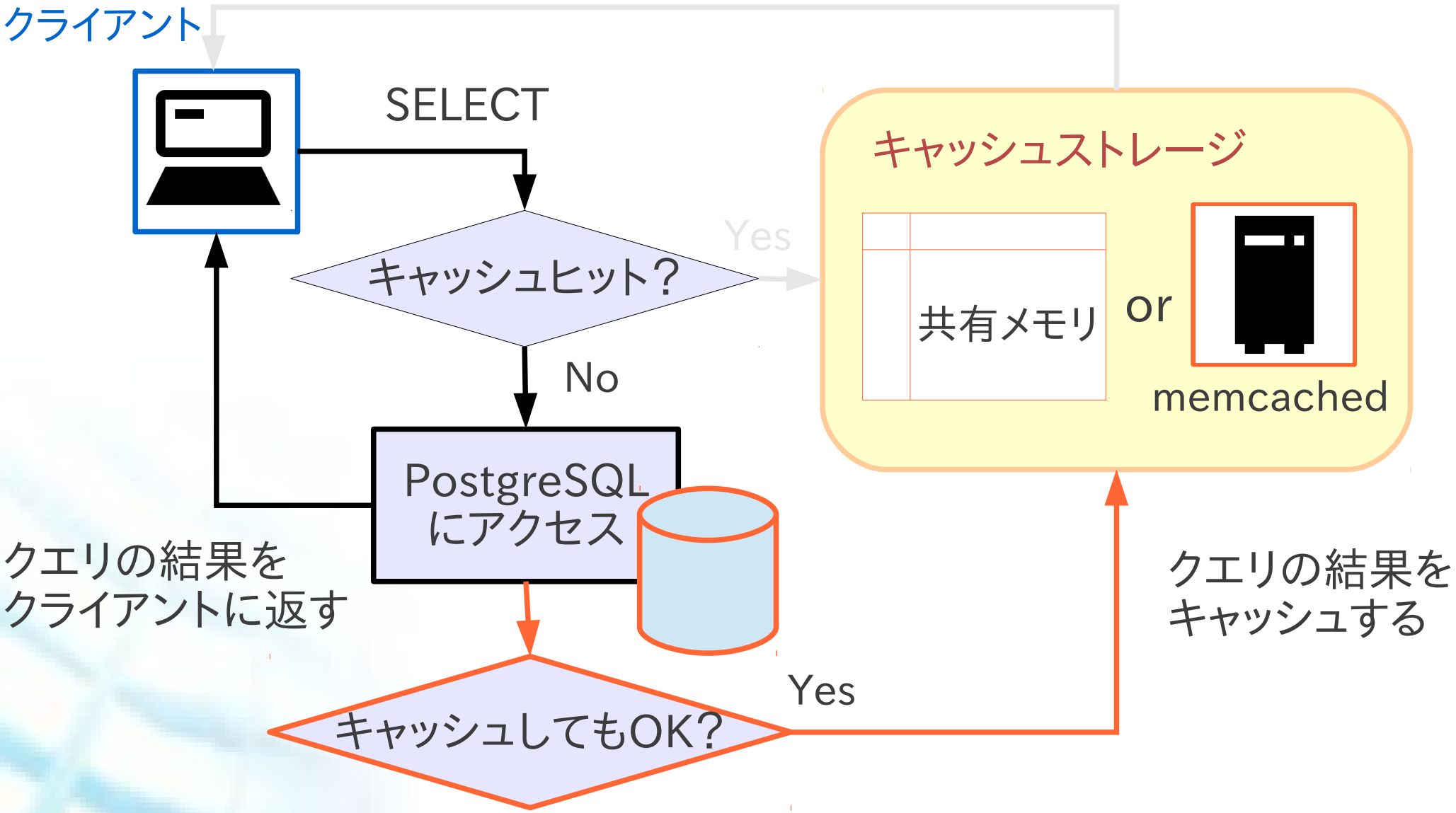
キャッシュされている結果をクライアントに返す → 速い！



SRA OSS, INC.

初回のクエリ

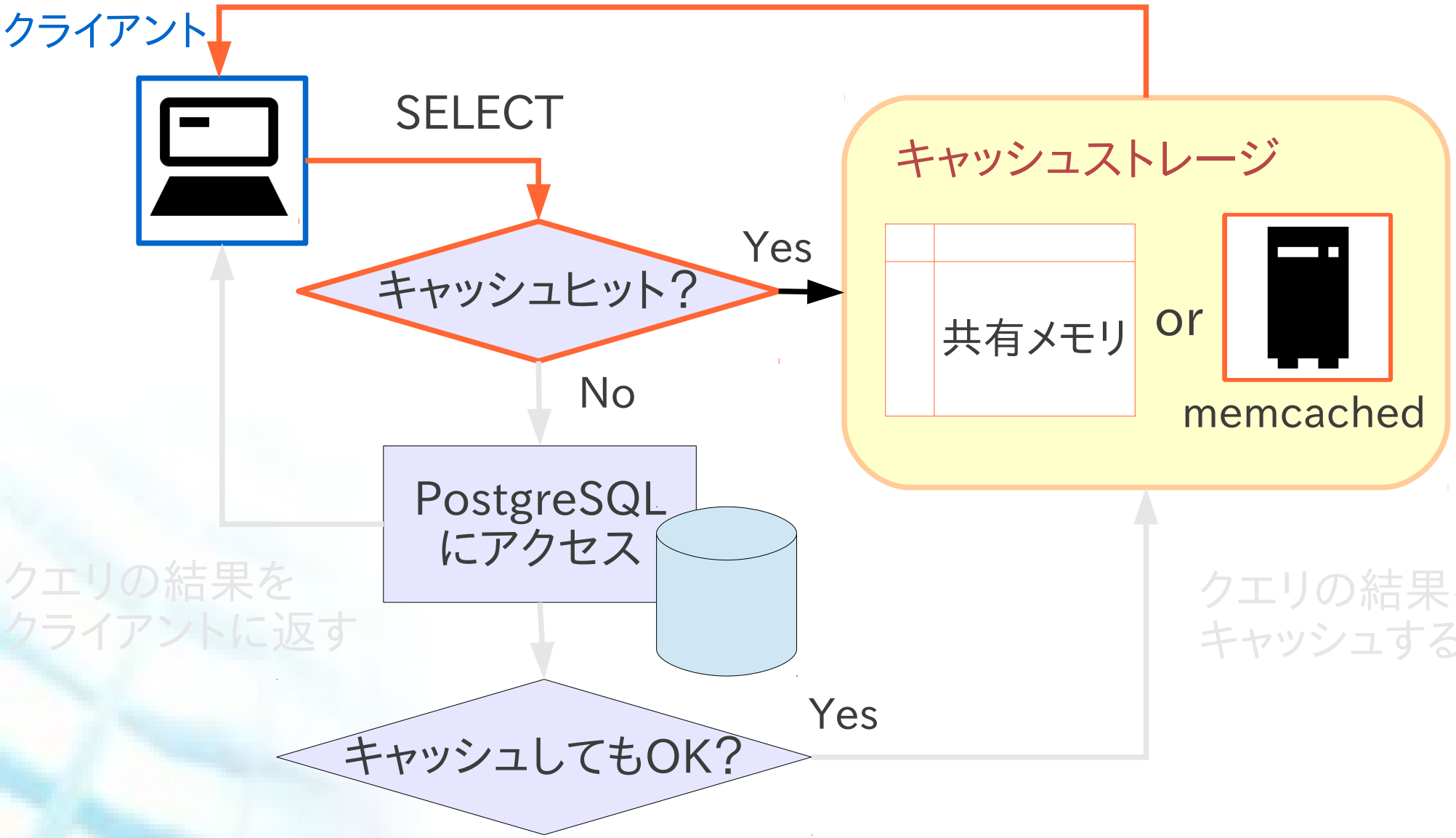
キャッシュされている結果をクライアントに返す → 速い！



SRA OSS, INC.

2回目以降のクエリ

キャッシュされている結果をクライアントに返す → 速い！



SRA OSS, INC.

キャッシュヒットの定義

- キャッシュヒット
  - 今回のクエリと「同じクエリ」の結果が既にキャッシュされているか？
- 以下のハッシュ値の一致を調べる（単純問い合わせの場合）

クエリハッシュ = md5 (クエリ文字列 + データベース名 + ユーザ名)

長大なSQL文の保存の必要性を避けるため
md5 ハッシュを用いる。

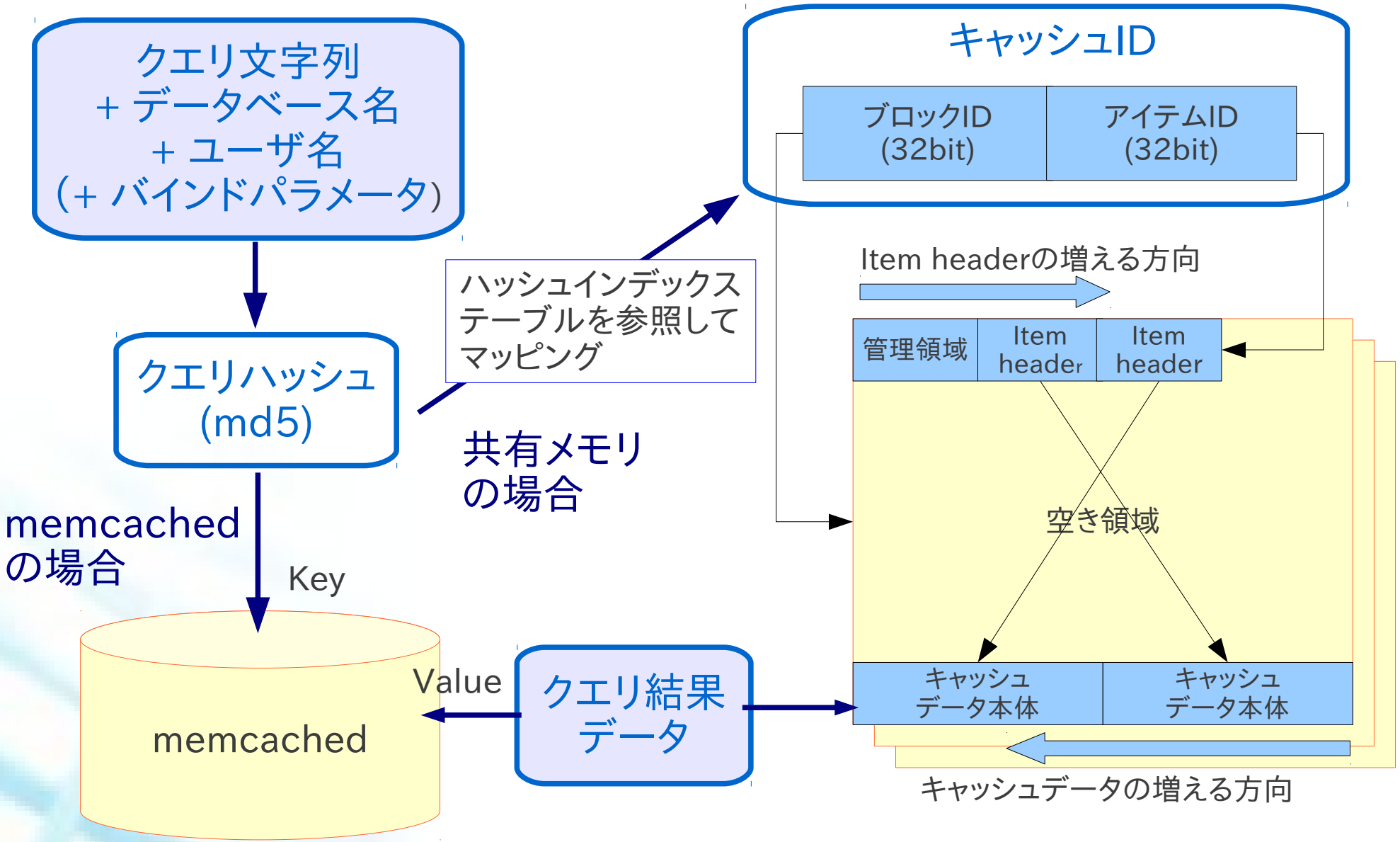
参照権限のないテーブルのデータをキャッシュ経由で他のユーザが参照できないようにするため。

- 拡張問い合わせ（プリペアド文）の場合

クエリハッシュ =

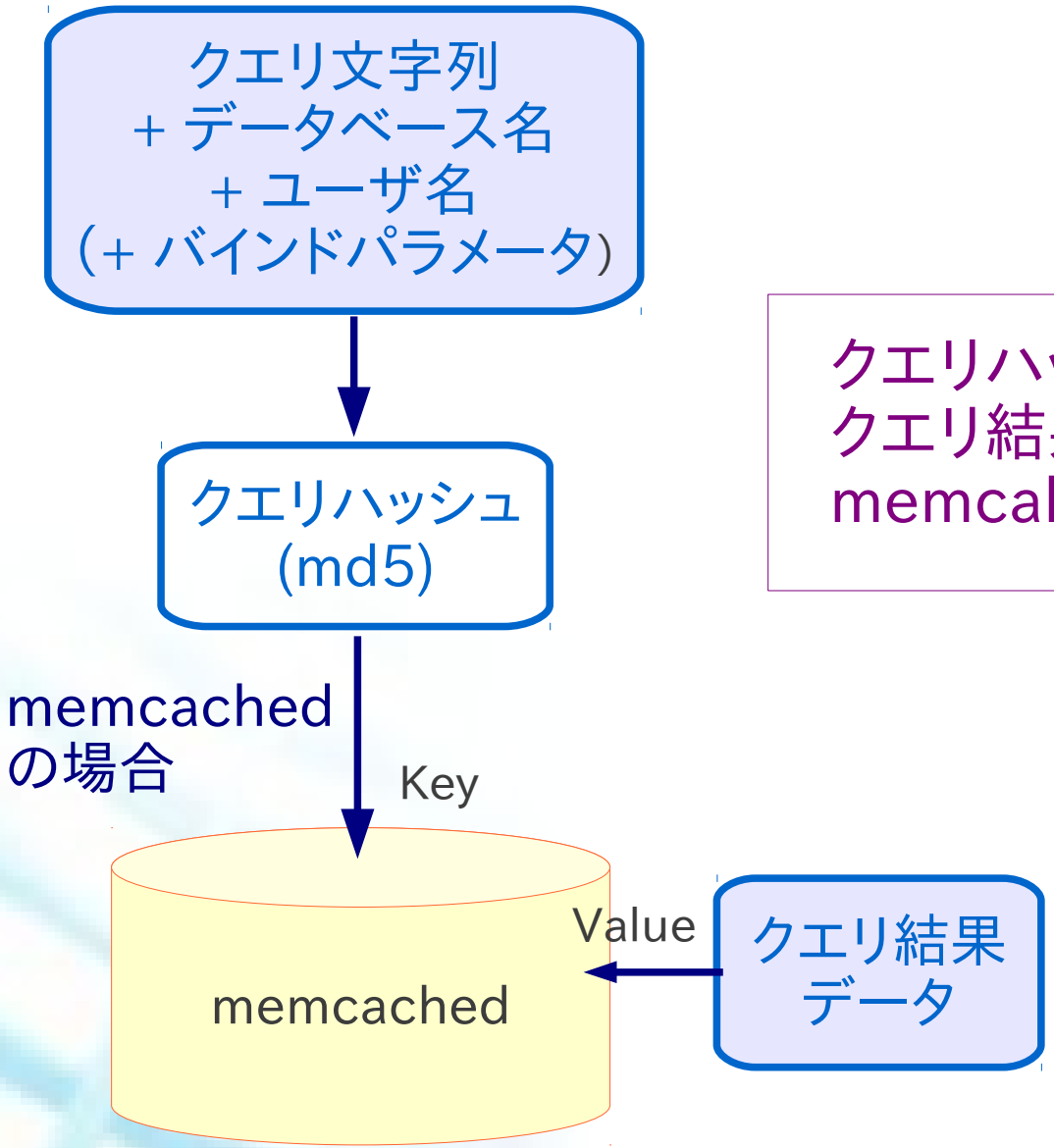
md5 (クエリ文字列 + バインドパラメータ + データベース名 + ユーザ名)

キャッシュストレージへの格納方法



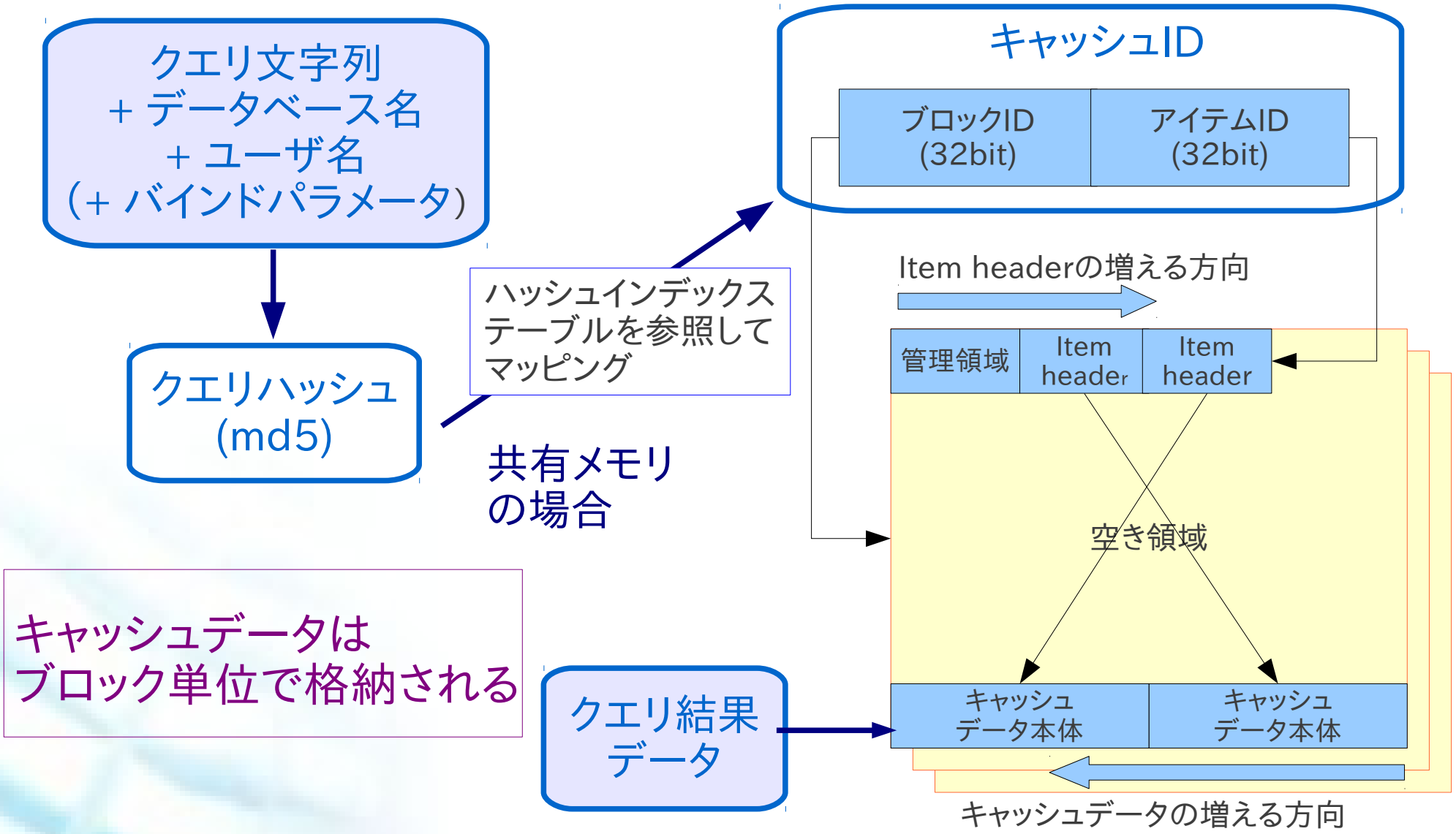
SRA OSS,INC.

キャッシュストレージへの格納方法 (memcached)



クエリハッシュ(md5)をキー、
クエリ結果データをバリューとして
memcahed に格納

キャッシュストレージへの格納方法（共有メモリ）



SRA OSS,INC.

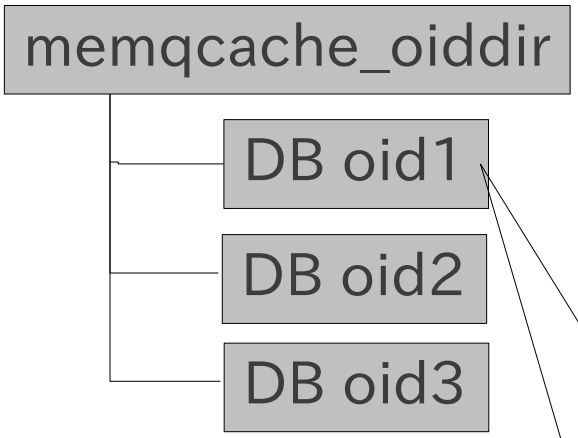
キャッシュの無効化

- キャッシュ対象のテーブルが一部でも更新されたら、そのテーブルを参照しているキャッシュをすべて自動的に削除する
  - 更新クエリ: INSERT/UPDATE/DELETE/TRUNCATE
  - この機能をオフにすることも可
- そのために、キャッシュを登録する際に参照しているテーブルのOIDを調べ、OIDマップ というファイルに登録しておく
  - 更新クエリが実行されたらOIDマップを調べて、削除すべきキャッシュのリストを取得する
- データベースやテーブルが削除(DROP)された場合も同様

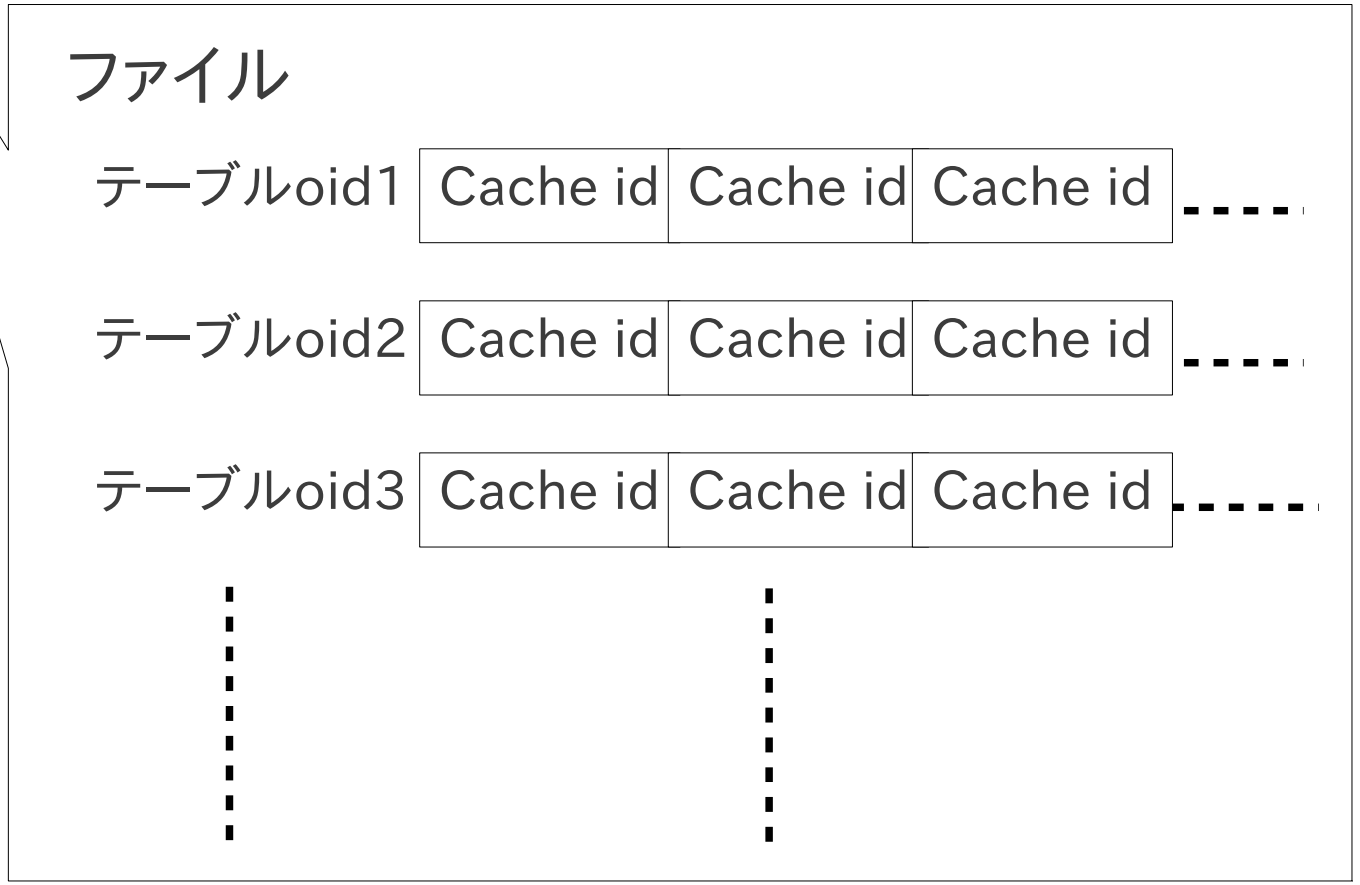
- 有効期限が過ぎたキャッシュも削除される
  - デフォルトでは無効

OIDマップ

ディレクトリ



データベース/テーブルのOIDと、
それを参照しているクエリキャッシュのID
を関連付けたリスト



キャッシュ削除を高速化

SRA OSS, INC.

キャッシュされないSELECT (1) ～特殊なクエリなど～

- Immutable でない（実行するたびに結果が異なる）関数の呼び出しを含むSELECT
  - current_timestamp, nextval(), random() など
- SELECT FOR SHARE/UPDATE
- SELECT INTO

- /* NO QUERY CACHE */ で始まるクエリ

- SELECTの結果データが大きいもの
  - memqcache_maxcache オプションで最大サイズを指定
- 失敗したSELECT
- ロールバックされたトランザクション内のSELECT

キャッシュされないSELECT (2) ～特殊なテーブルなど～

- 一時テーブルを含んでいるSELECT
- システムカタログを含んでいるSELECT

- VIEW を含む SELECT
- UNLOGGED テーブルを含んでいるSELECT

- 例外となるテーブルの設定
  - ブラックリスト（black_memqcache_table_list）に指定された名前のテーブルはキャッシュされない
  - ホワイトリスト（white_memqcache_table_list）に指定されたVIEW, UNLOGGED テーブルはキャッシュされる

オンメモリクエリキャッシュの注意事項/制限事項

- テーブルの一部でも更新されると、テーブル全体のキャッシュが削除される
  - 更新が多いシステムでは効果がでない
  - キャッシュヒット率 70% 以上だとかなり有効
- 暗黙的な更新が認識できない
  - トリガによって暗黙的に更新された場合
  - ON DELETE CASCADEなどで他のテーブルの行が暗黙的に更新された場合
  - VIEWが参照しているテーブルが更新された場合
- 複数の pgpool-II を動かす環境で共有メモリを使っている場合、他のpgpool-IIで行われた更新が認識できない
  - このような環境ではキャッシュストレージに memcached を使う

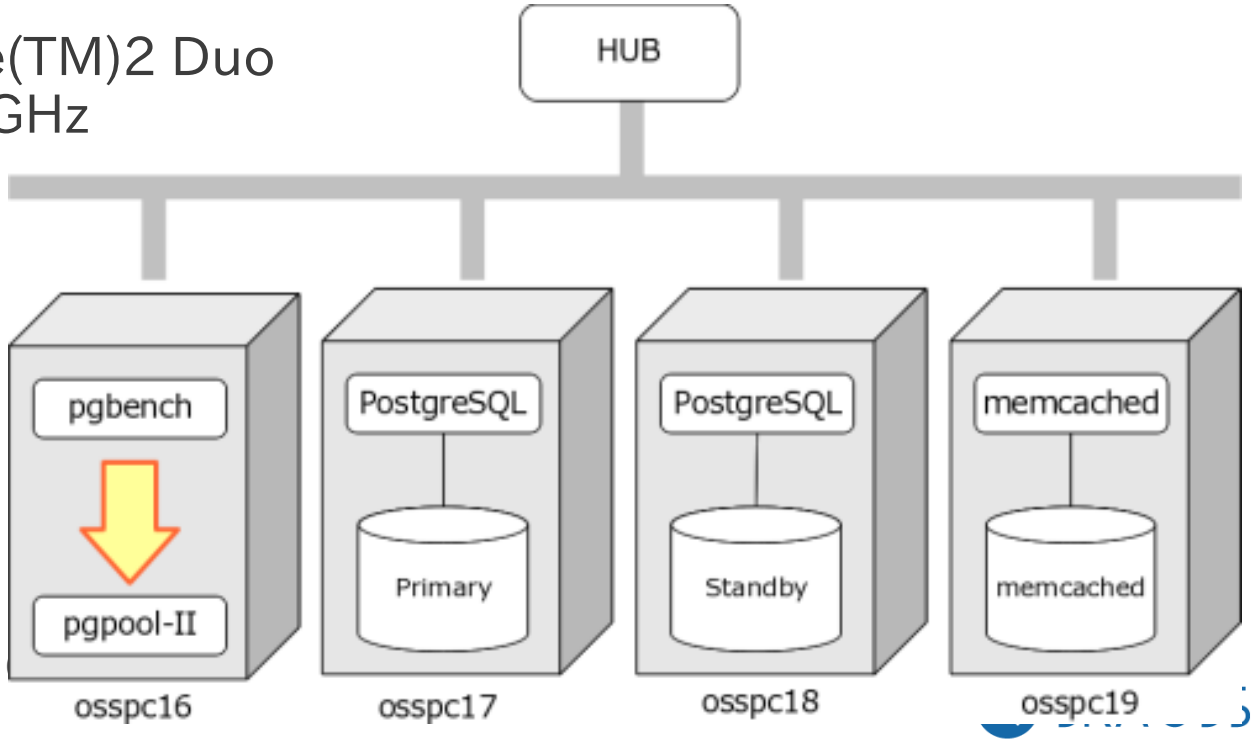
ベンチマーク!

- pgbench
  - Scale factor = 1（10万件）
  - SELECT のみのトランザクション
    - pgbench -S -T 10 -c 10
    - クライアント数 10, 各回 10 秒間
- プライマリキーインデックスを削除し、あえて検索に時間がかかる状態にして測定
  - 各 SELECT の所要時間は約 6.7 ms
- キャッシュストレージのサイズは 64MB

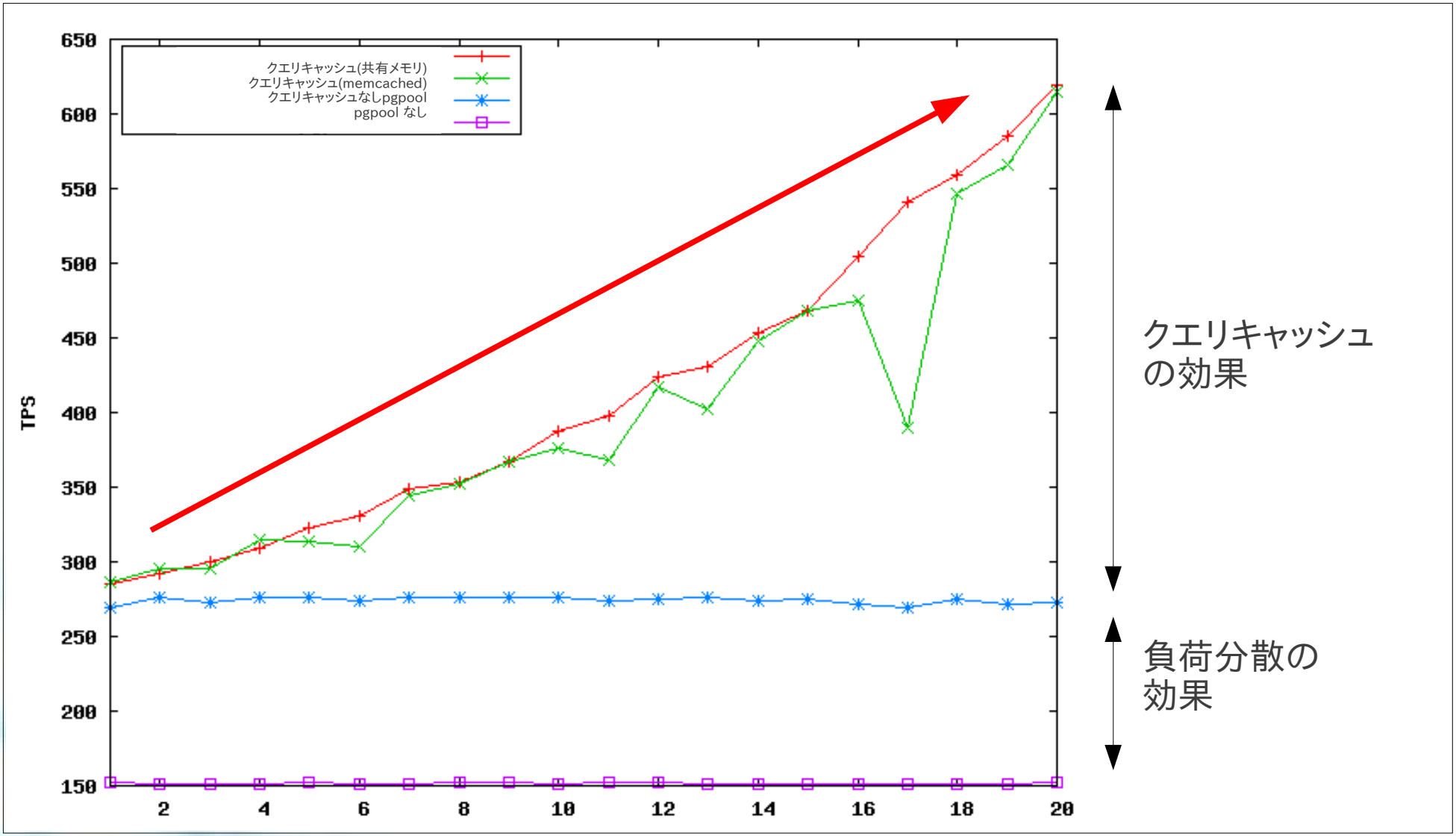
  （共有メモリと memcached で同じ）

ベンチマーク環境

- Pgpool-II　　3.2 alpha
- PostgreSQL　9.1.2
- memcached　1.4.13
- マシンスペック（実サーバ / 4 台とも共通）
  - 品番　　DELL OPTIPLEX 755
  - OS　CentOS release 5.5 (final)
  - CPU　　Intel(R) Core(TM)2 Duo CPU E6550 @ 2.33GHz
  - メモリ　2 GB

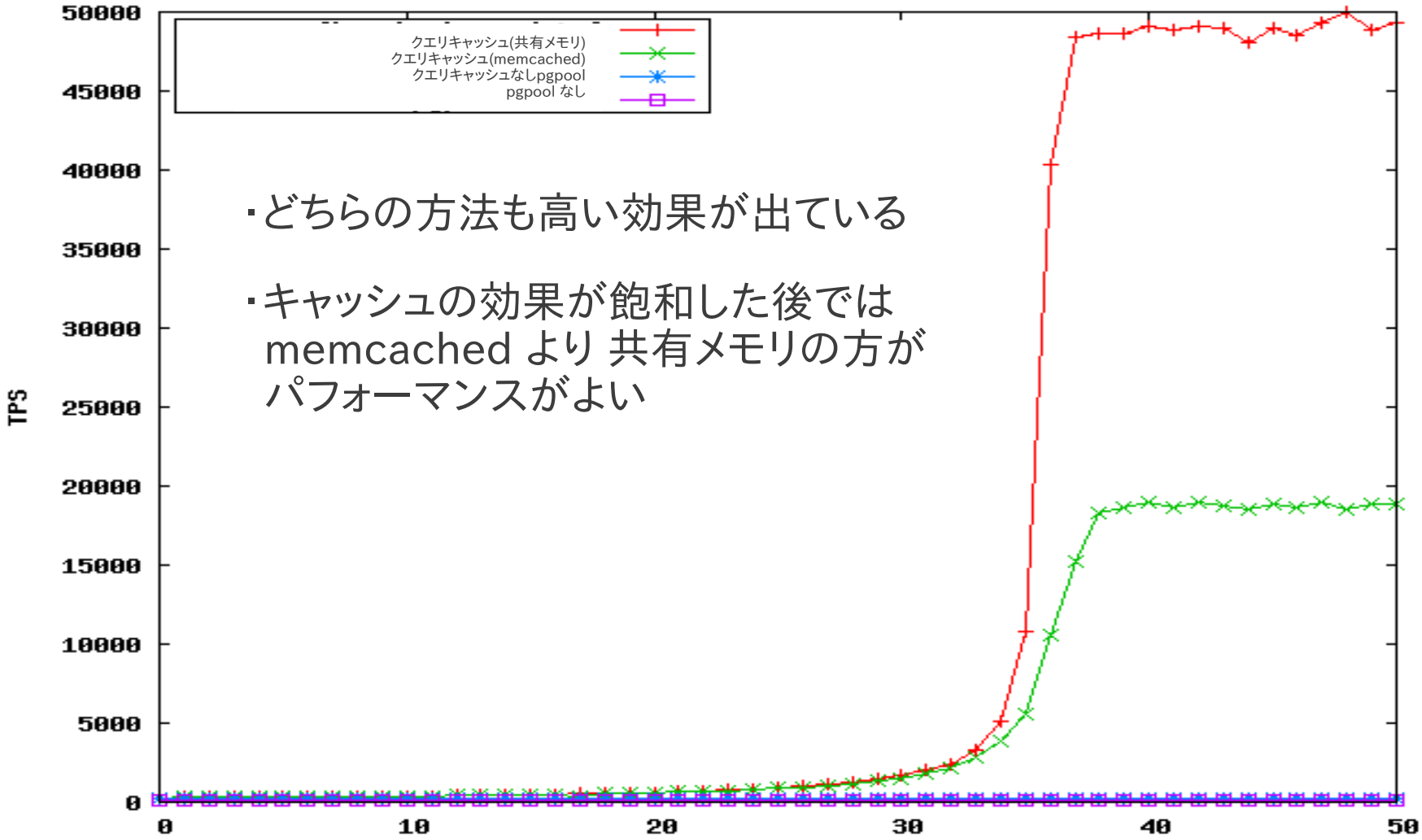


クエリキャッシュの効果～最初の20回（200秒分）



SRA OSS,INC.

クエリキャッシュの効果～キャッシュ効果の飽和



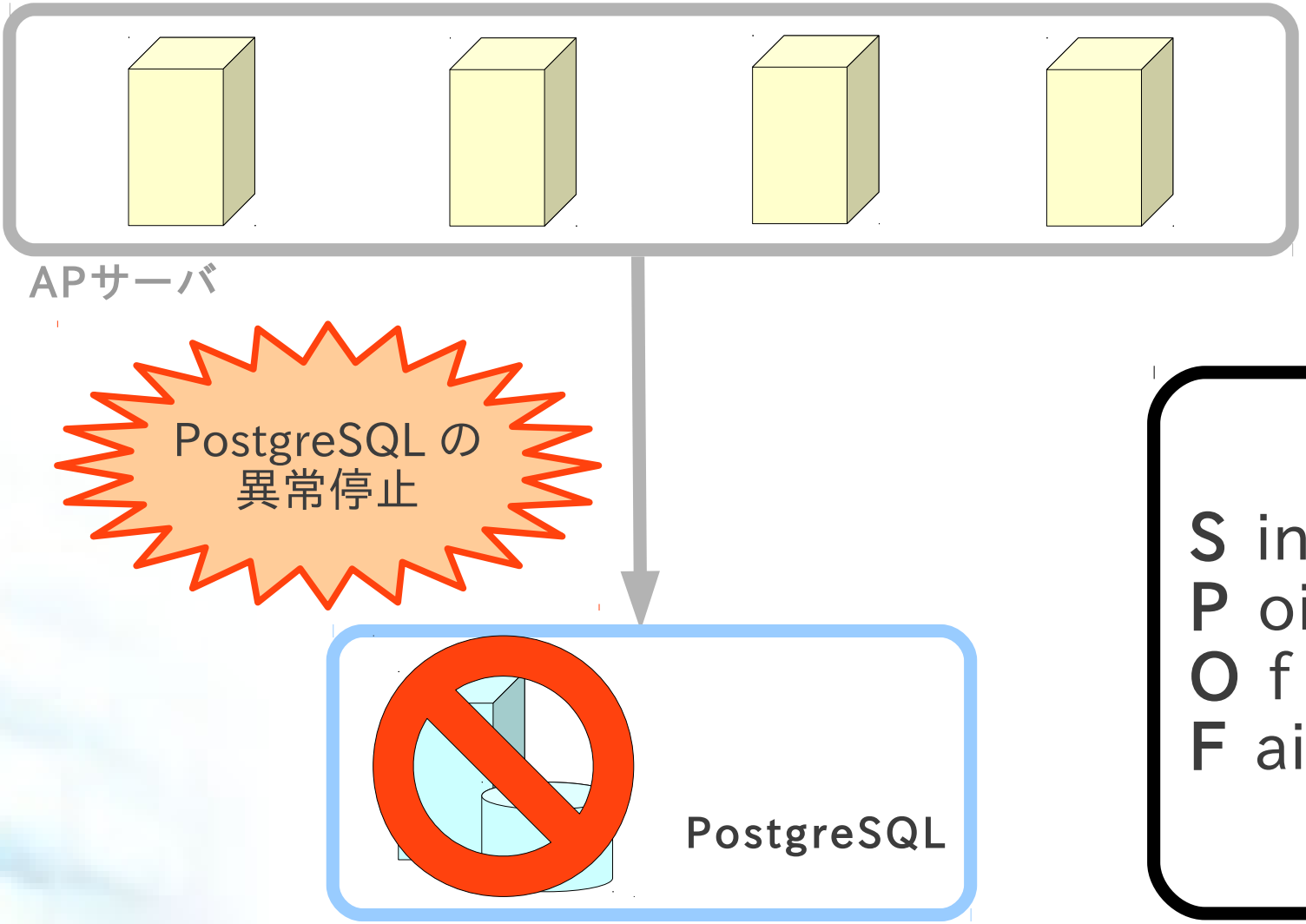
- どちらの方法も高い効果が出ている
- キャッシュの効果が飽和した後では memcached より 共有メモリの方がパフォーマンスがよい

SRA OSS,INC.

2. watchdog（組み込みHA）

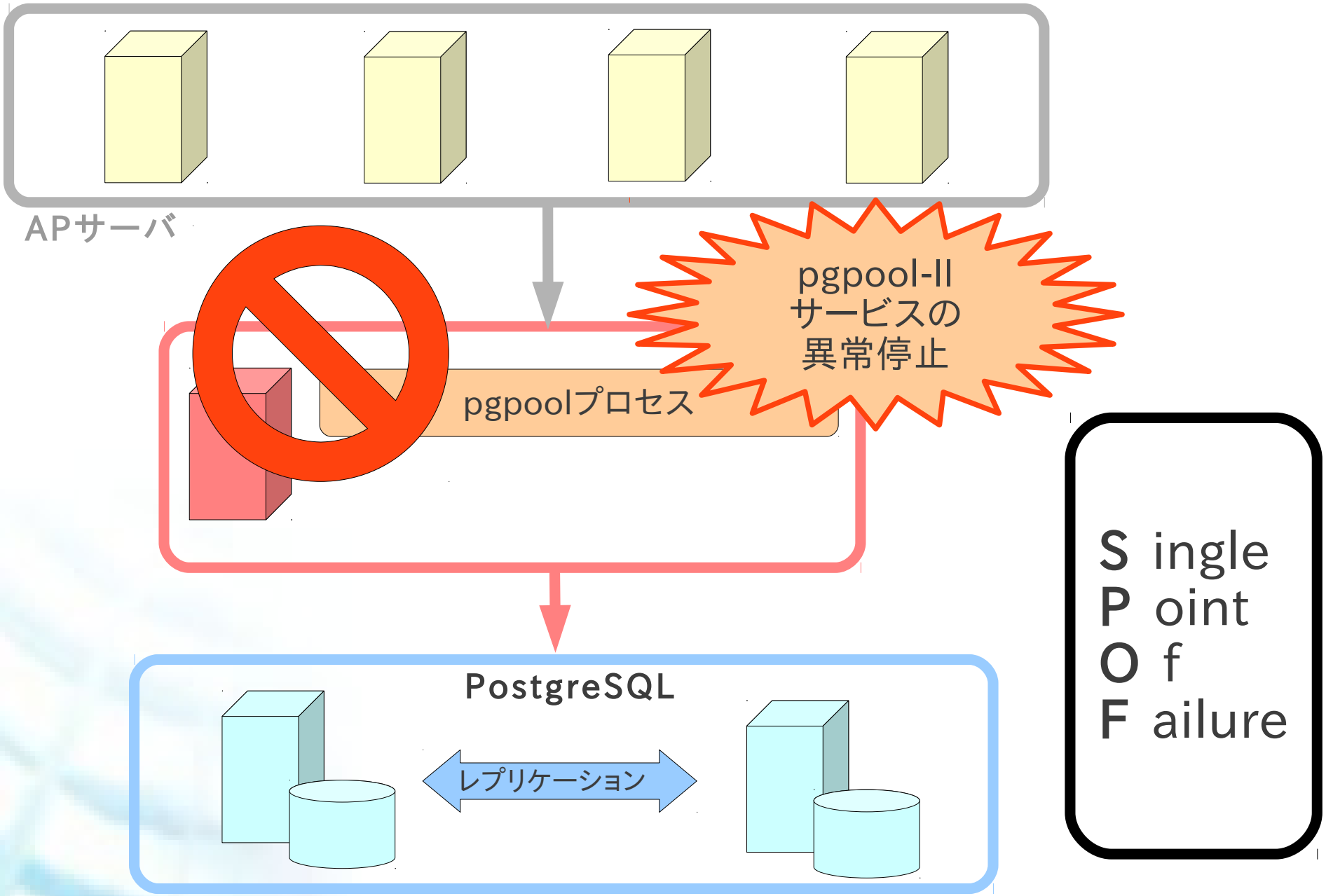
SRA OSS,INC.

単一障害点 ～ 1台のPostgreSQL



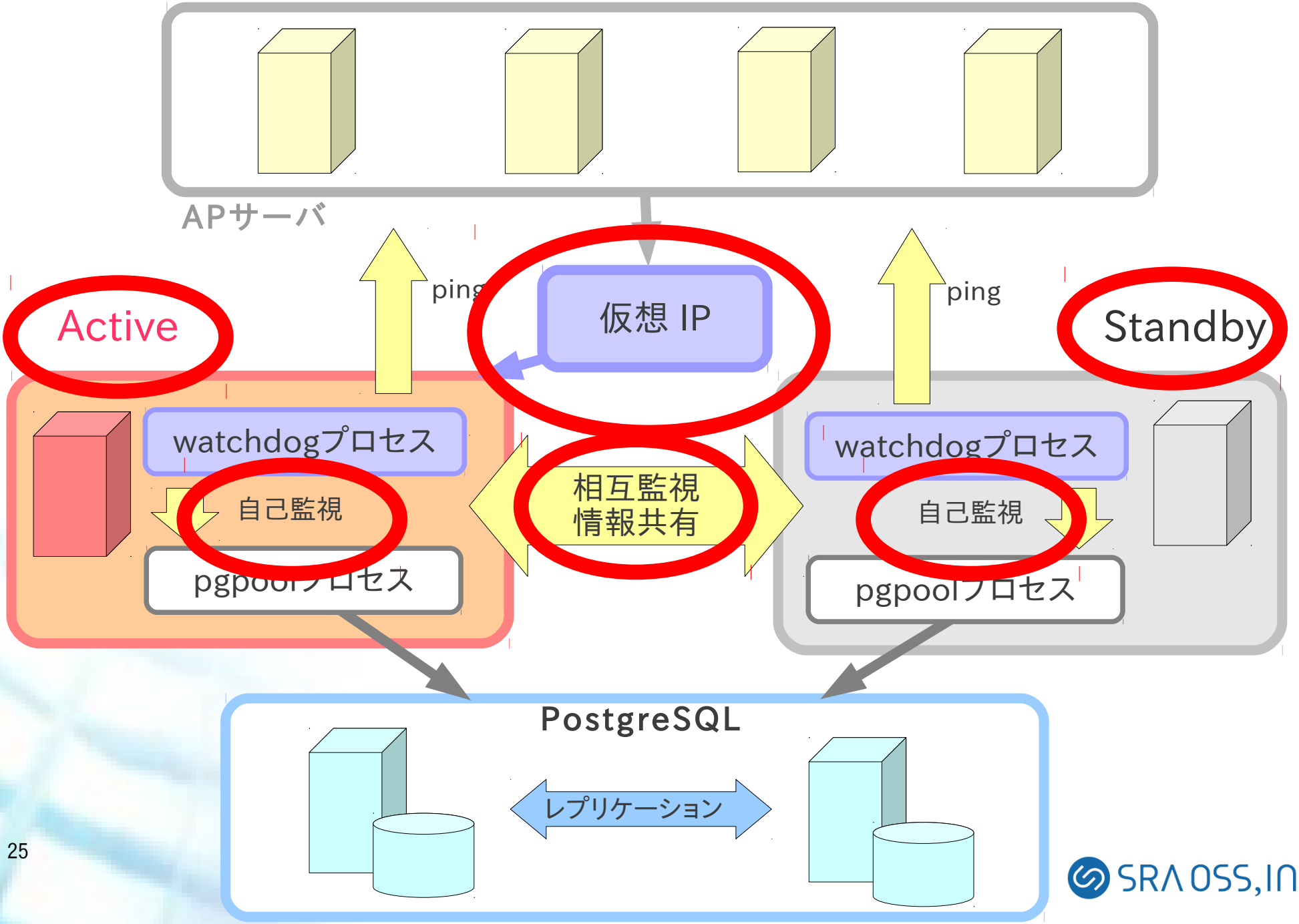
SRA OSS, INC.

単一障害点 ～ 1台の pgpool-II



SRA OSS, INC.

watchdogを使ったActive / Standby構成



SRA OSS, INC.

watchdog の死活監視の詳細

wd_lifecheck_interval 秒おきに、以下を実行する
（デフォルトでは10 秒）

1. 全上位サーバとの接続をチェック（ping）

自分はサービスを提供可能か？

2. 自分と同居している pgpool-II に wd_lifecheck_query のクエリを実行してみる

3. 自分以外の watchdog と同居している pgpool-II に wd_lifecheck_query のクエリを実行してみる

監視対象 pgpool-IIはサービスを提供可能か？

- 実行するクエリ
  - デフォルトではSELECT 1
  - wd_life_point 回リトライ（デフォルト3回）

Active/Standby の切り替え

- watchdog のステータス

Active:	仮想IPを保持している pgpool-II（全体で1つのみ）
Standby:	仮想IPを保持していない pgpool-II
Down:	サービスを提供できない pgpool-II

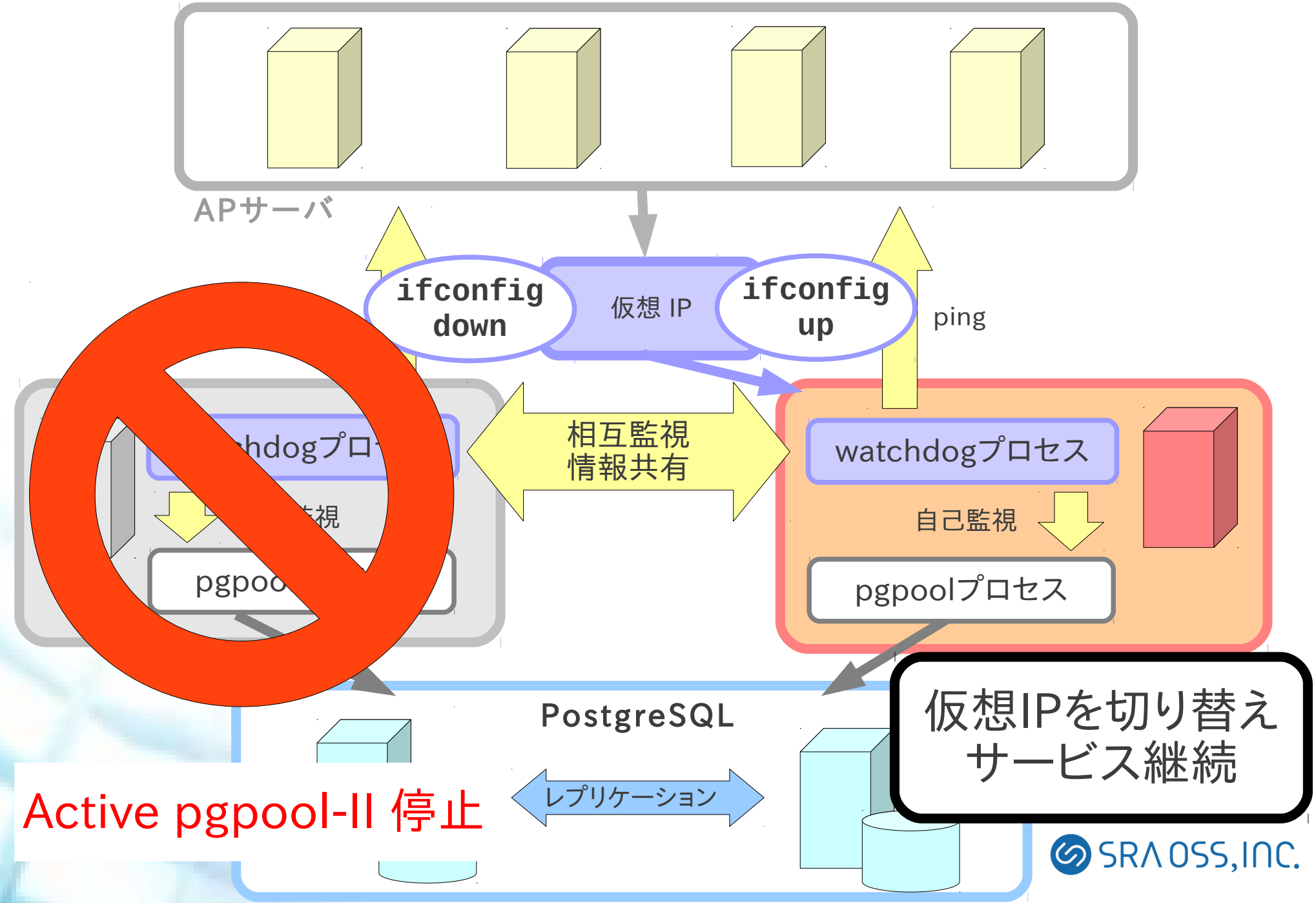
- 自分自身がdownした場合
  - その事を他の pgpool-II に通知する

正常終了
or
自己監視の結果

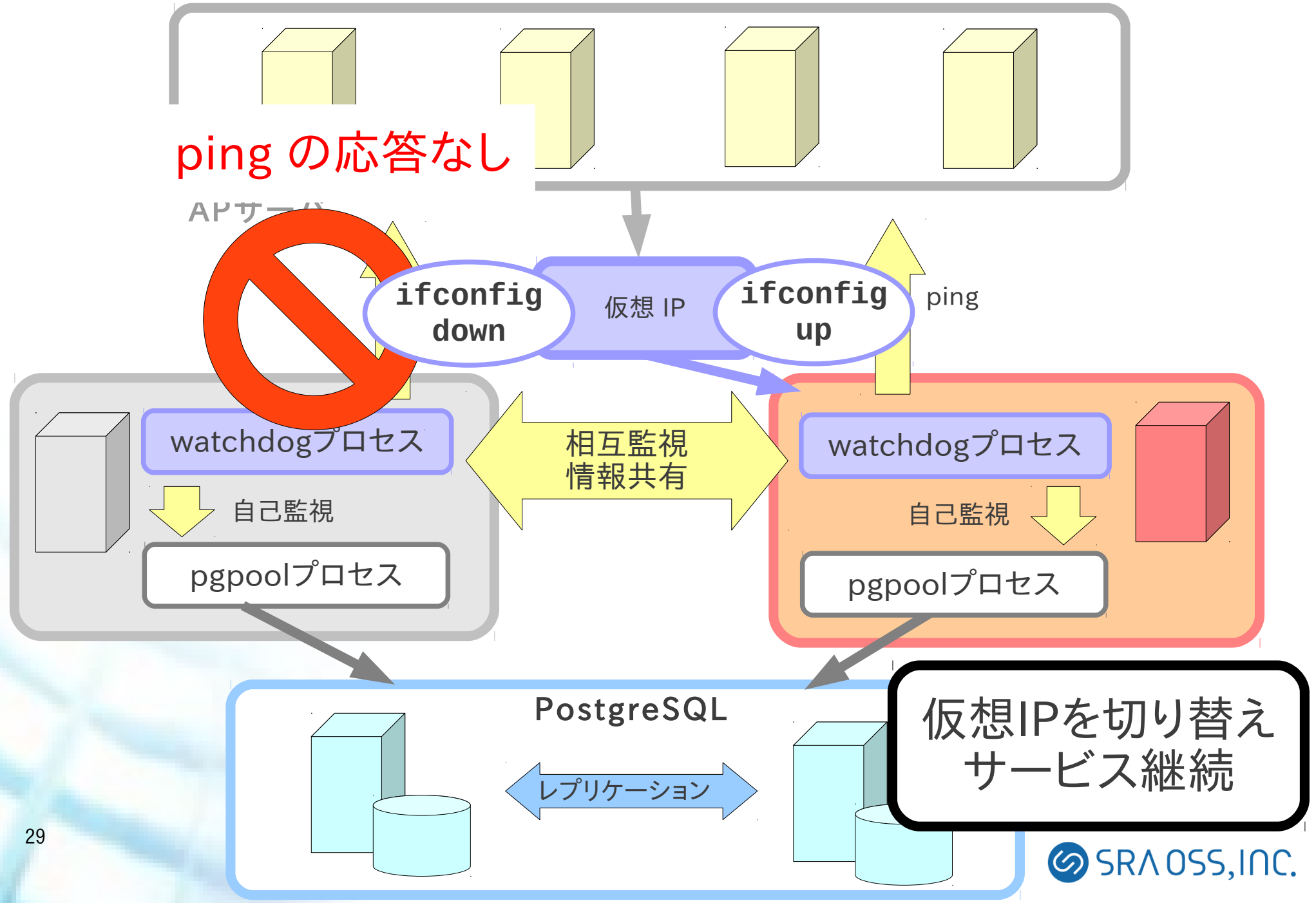
- 他の pgpool-II がdownした場合
  - active 不在でかつ自分が最古参ならば、active に昇格

down通知の受信
or
相互監視の結果

Active の pgpool-II の停止



上流サーバへのping失敗



DBノード情報の同期

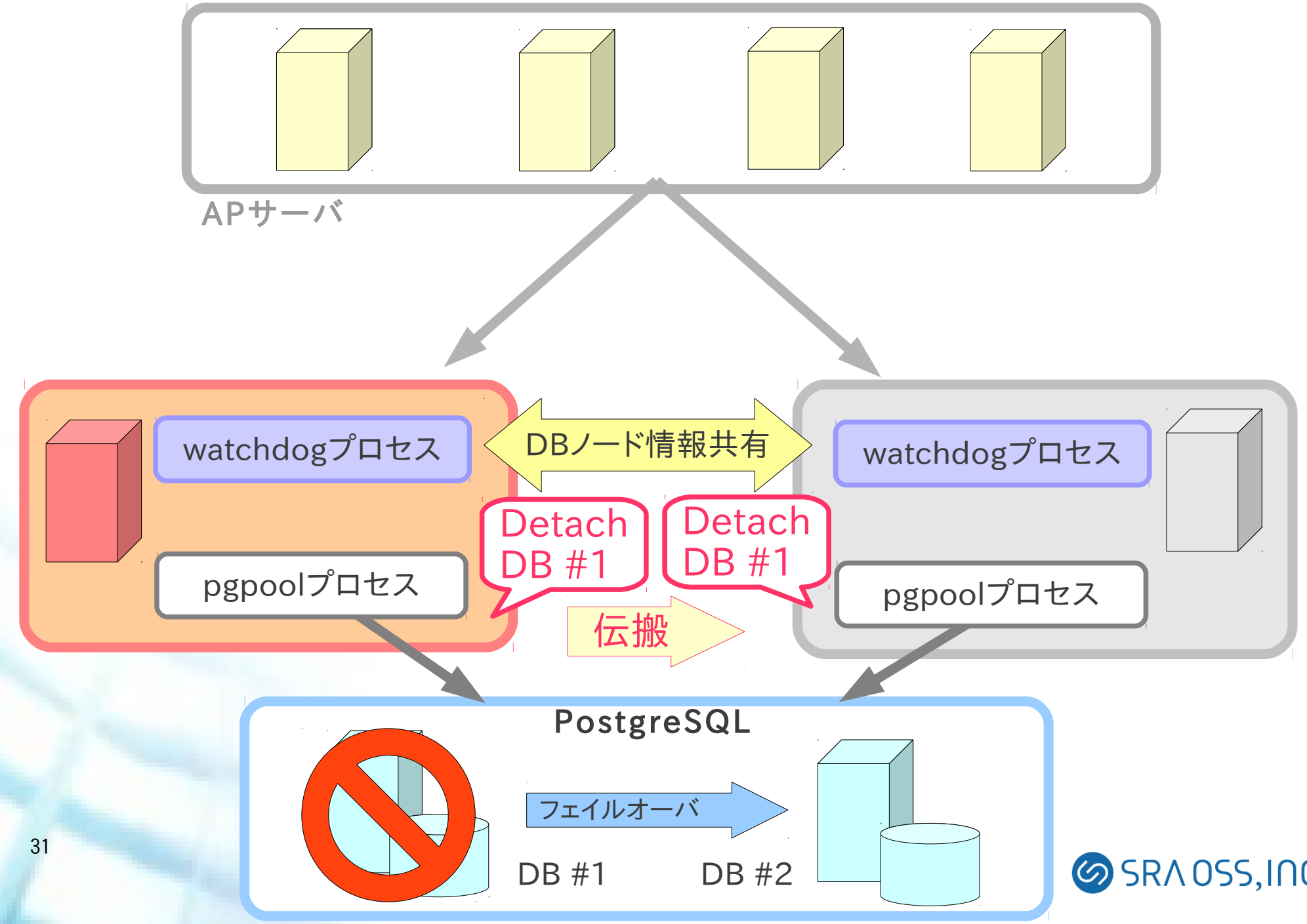
- failover command 実行時
  - 全ての pgpool-II で同じ failover_command が実行される
    - DB ノードの追加　(attach)
    - DB ノード切り離し　(detach)
    - プライマリへの昇格　(promote)

→　全 pgpool-II で DBノードの状態を同じに保つ

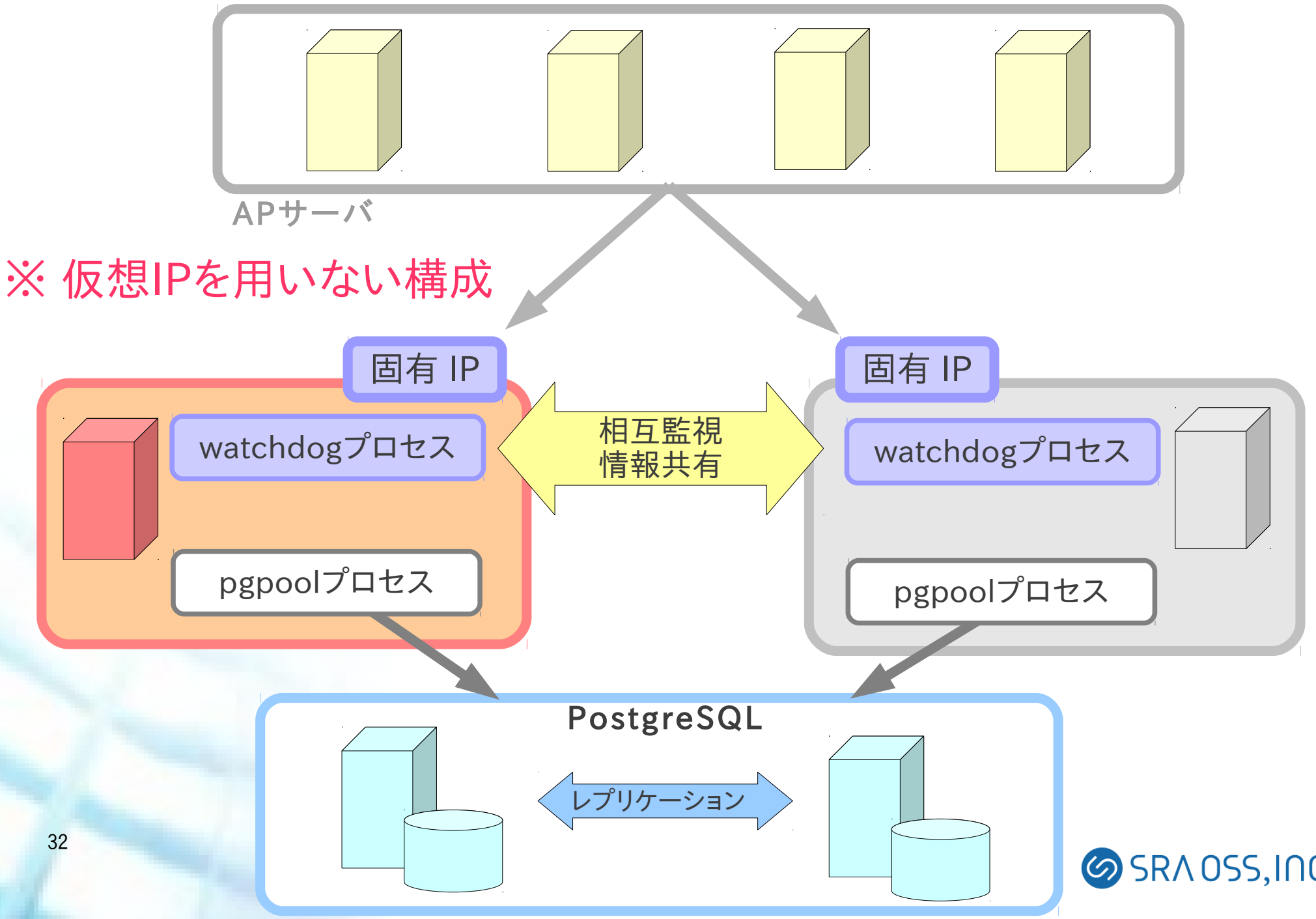
- オンラインリカバリのタイミング　（2nd ステージ)
  - 開始時と終了時に、他の pgpool-II へ通知する
    - リカバリ開始前に、全ての接続が終了するまで待機
    - リカバリ中は、新規接続の受付を中止

→　DBノードが完全にレプリケートされるのを待つ

フェイルオーバの伝搬



watchdog を使った multi-master 構成



SRA OSS, INC.

watchdog の注意事項/制限事項

- 仮想IPの制御に root 権限が必要
  → OS の ifconfig, arping コマンドを実行するため
  - 方法 1: root 権限で pgpool-II を起動
  - 方法 2: ifconfig, arping コマンドに setuid を設定
    - pgpoolAdmin を使用するときは「方法2」をとる必要がある

- pgpool-II から全DBノードが切り離された場合は、pgpool-II を再起動する必要がある
  - pcp_attach_node コマンドや pgpoolAdmin を使ってノードを復帰させると、active 状態の pgpool-II が複数できてしまう

pgpoolAdmin: pgpool-II の GUI 管理ツール（Webアプリ）
pcp_attach_node: pgpool-II にDBノードを追加するコマンド

watchdog デモ

SRA OSS,INC.

まとめ

pgpool-II 3.2 では PostgreSQLの性能と信頼性を向上する
2 つの新機能が追加されました!

1.オンメモリクエリキャッシュ

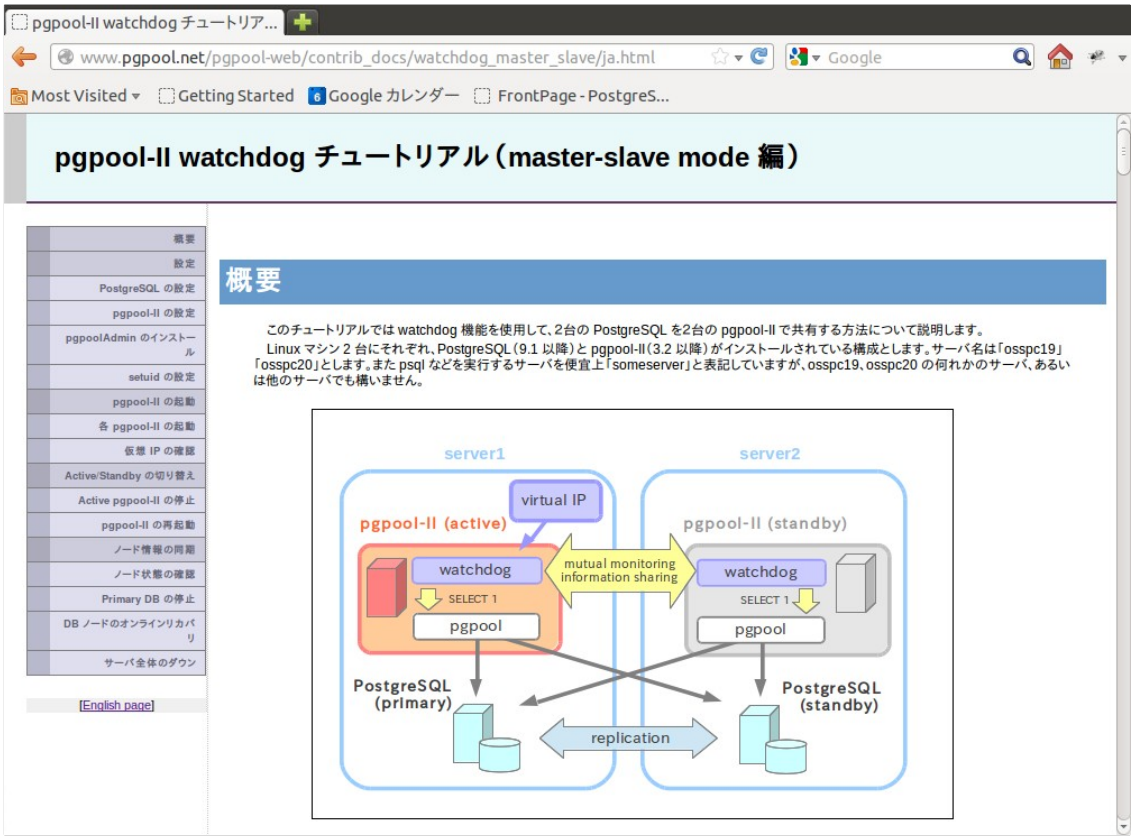
- SELECTの結果をキャッシュして高速化
- 検索の多いシステムでは、かなりの効果が期待できる

2.Watchdog

- pgpool-II の HA 構成
- マルチマスタ構成、アクティブ/スタンバイ構成で、PostgreSQL のより安全な運用ができる

information

- pgpool-II オフィシャルサイト
  - http://www.pgpool.net/
  - ダウンロード
  - マニュアル
  - チュートリアル
  - メーリングリスト
  - バグトラック
- Twitter
  - @pgpool2



- Let's Postgres:「pgpool-II 3.2 の新機能」
  - http://lets.postgresql.jp/documents/technical/pgpool-II_3.2/

SRA OSS,INC.

ご清聴ありがとうございました。

SRA OSS,INC.