PostgreSQL 9.0 の新機能
Hot Standby / Streaming Replication

PostgreSQLしくみ分科会
藤井 雅雄 - 笠原 辰仁
2010.8.28

アジェンダ

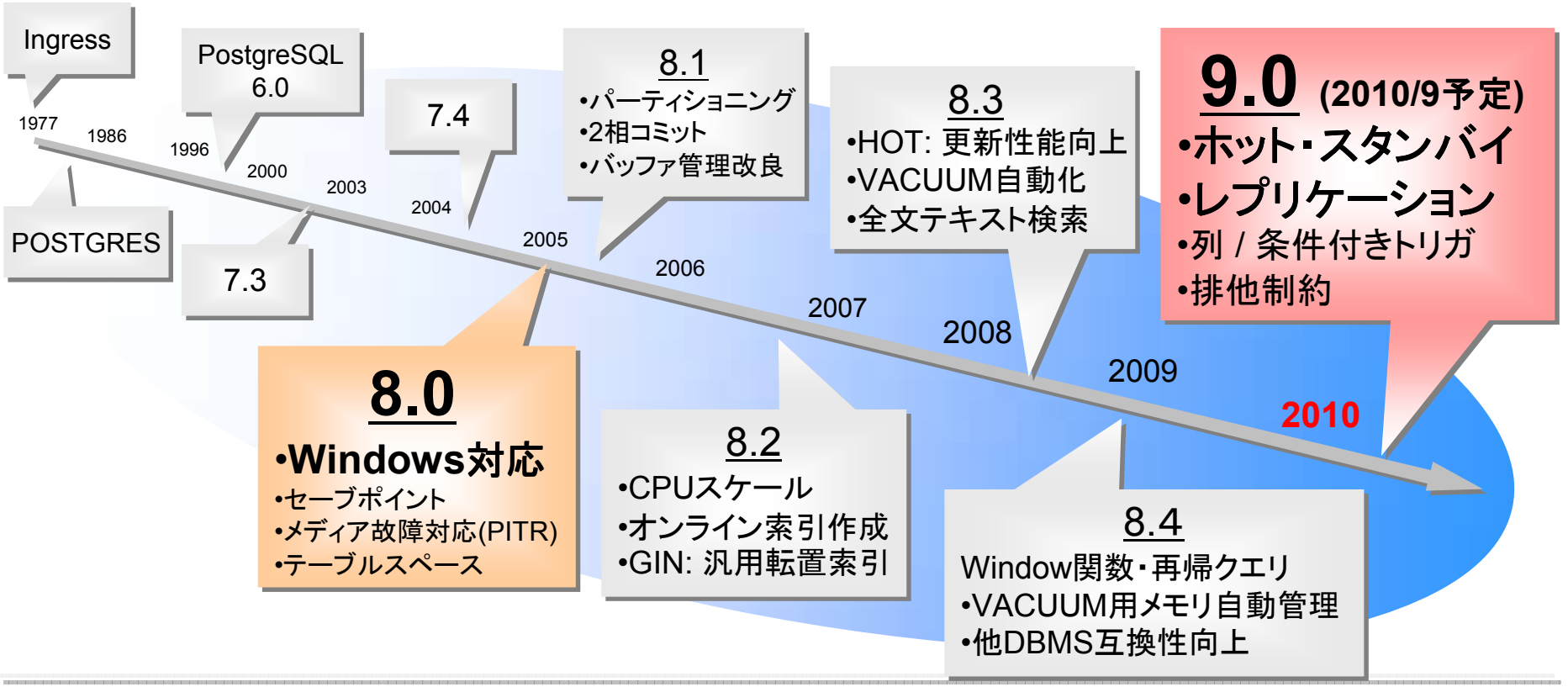
- はじめに
- レプリケーションの処理概要
- 9.0のレプリケーションでできること、できないこと
- パラメータ紹介
- TIPS
- → ハンズオンへ！

ハンズオンの前に、レプリケーションについての簡単な解説をします！

はじめに：PostgreSQL 9.0のリリース

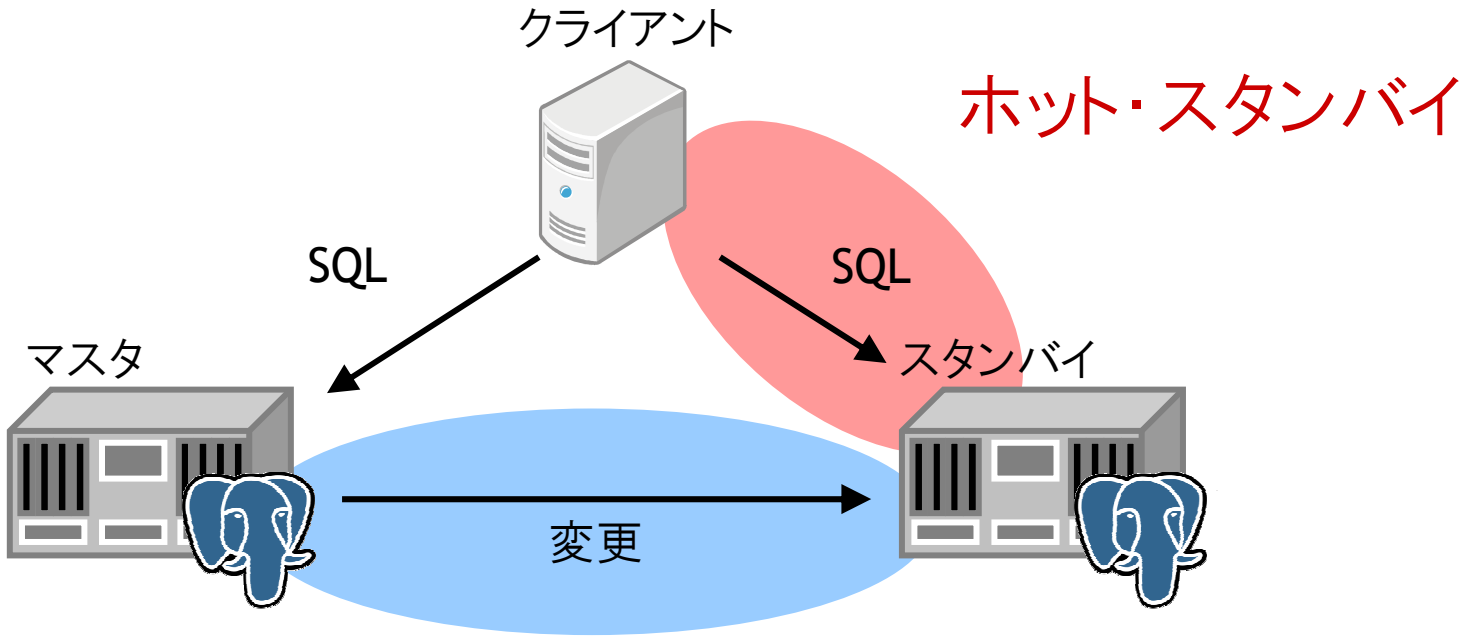
- 9.0は5年ぶりの「記念リリース」
  - 上一桁の更新は「適用領域の大幅な拡大」を表す
  - メジャーバージョンは上2桁 / バグ修整は3桁目
- ホット・スタンバイとレプリケーションをぜひ使って下さい！

9.0.0
メジャー マイナー



はじめに:9.0レプリケーションの構成

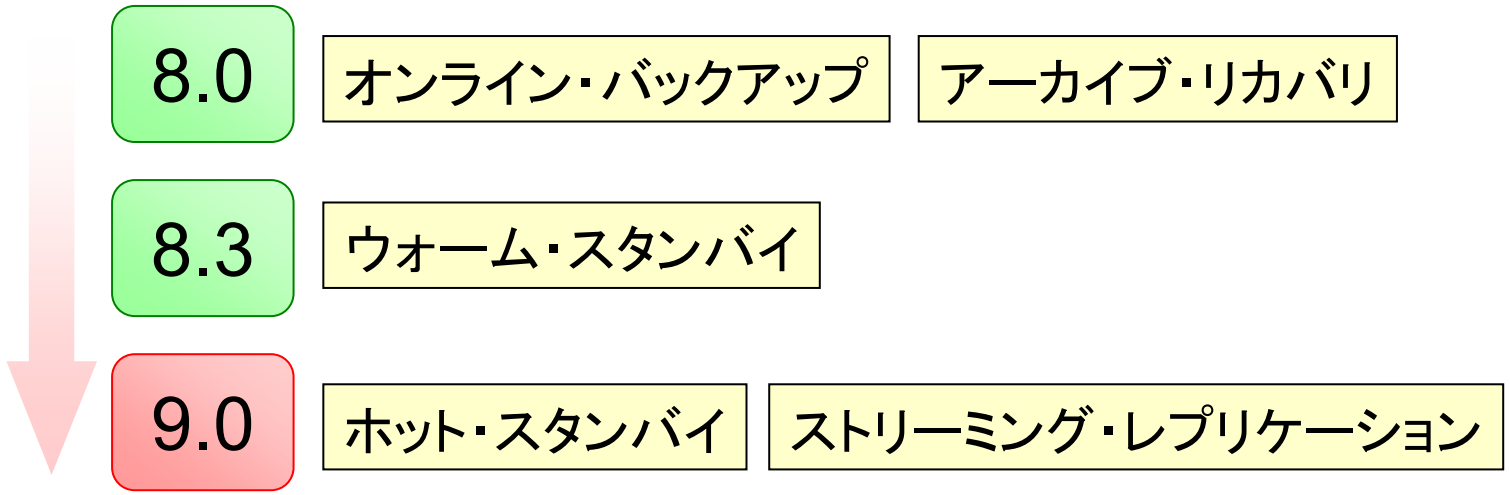
- PostgreSQL 9.0のレプリケーション機能
  - ストリーミング・レプリケーション (SR)
    - WALレコードをスタンバイへ流す
  - ホット・スタンバイ (HS)
    - スタンバイで参照を許可する



ストリーミング・レプリケーション

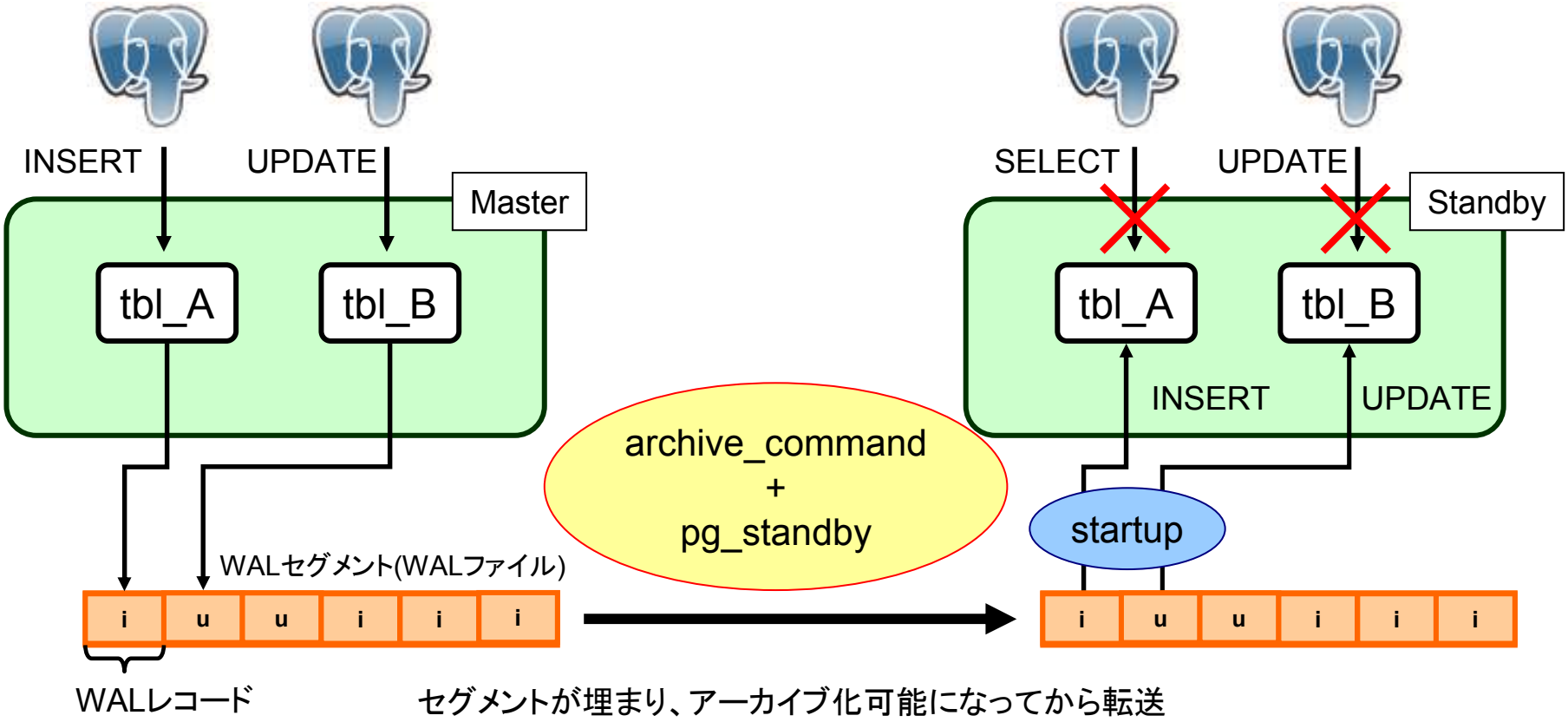
レプリケーション機能とは？

- これまでのウォーム・スタンバイの発展系



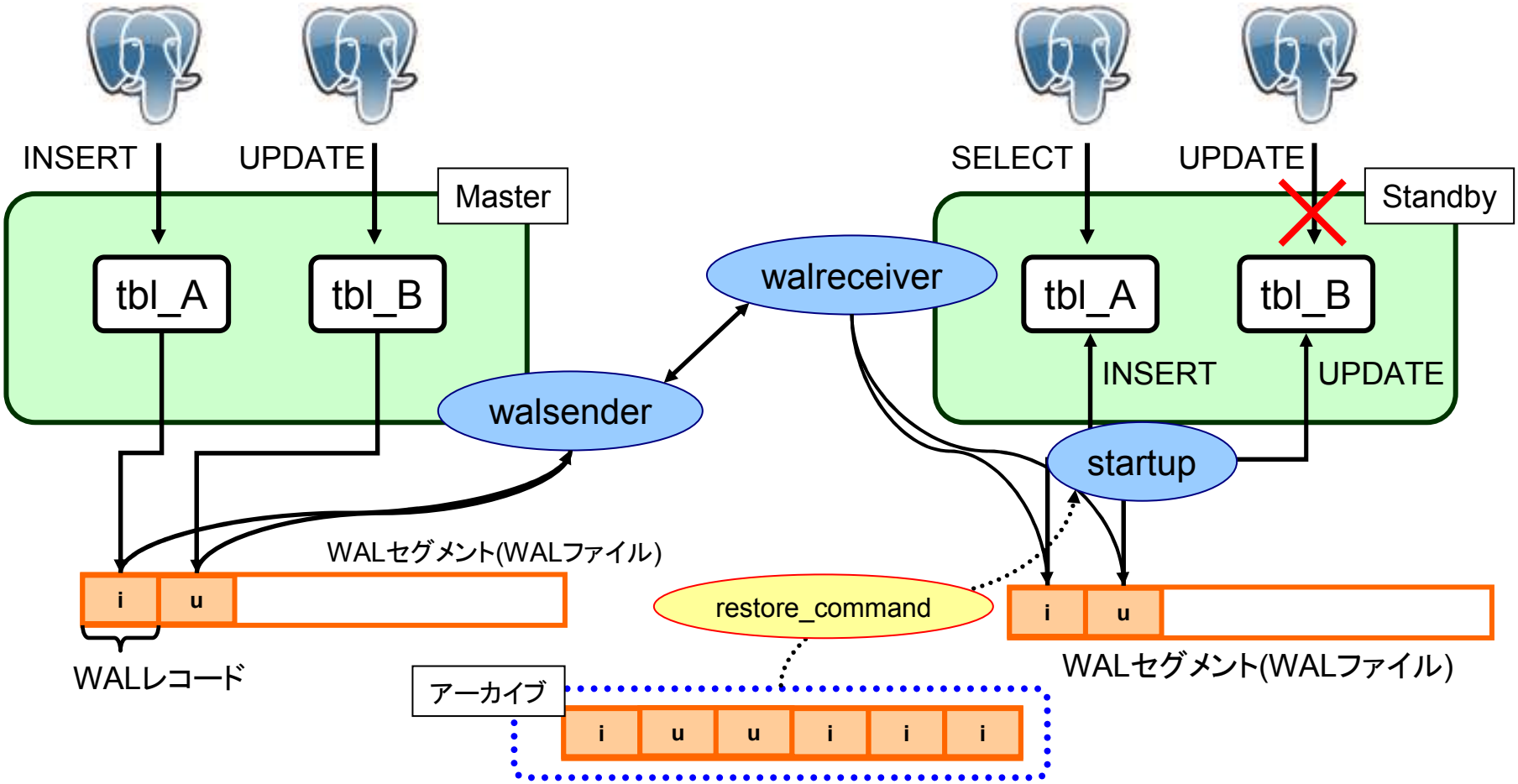
SR+HSの処理概要

- 従来(ウォームスタンバイ)
  - WALセグメント単位にスタンバイ側へ転送され、replayされる
    - セグメントが埋まる or archive_timeout 契機でないと転送されない
  - スタンバイではアクセス不可

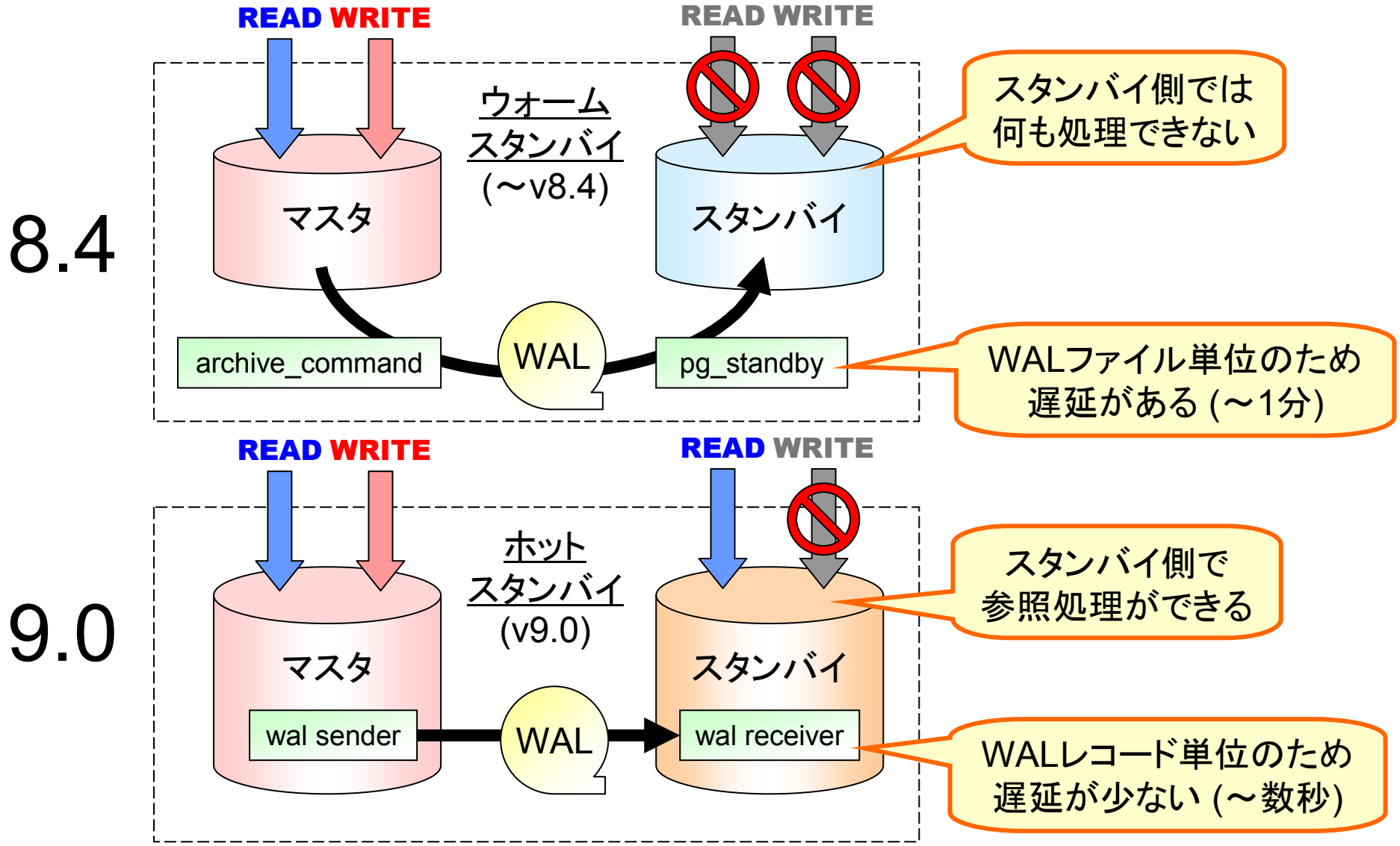


SR+HSの処理概要

- SR+HSの場合
  - WALレコード単位にスタンバイへ非同期的に転送され、replayされる
  - スタンバイではアクセス&参照SQL可能

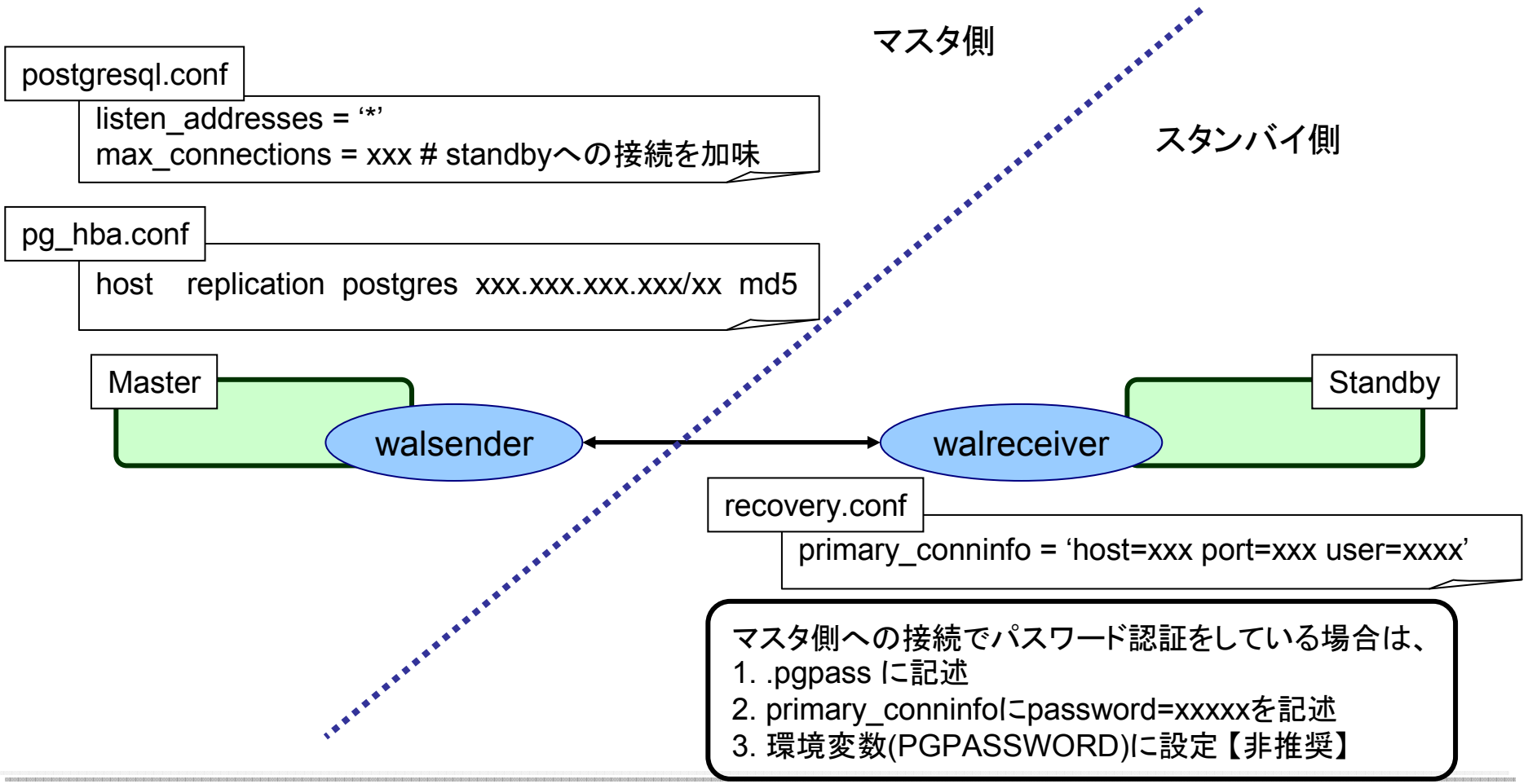


ウォーム・スタンバイ vs. ホット・スタンバイ



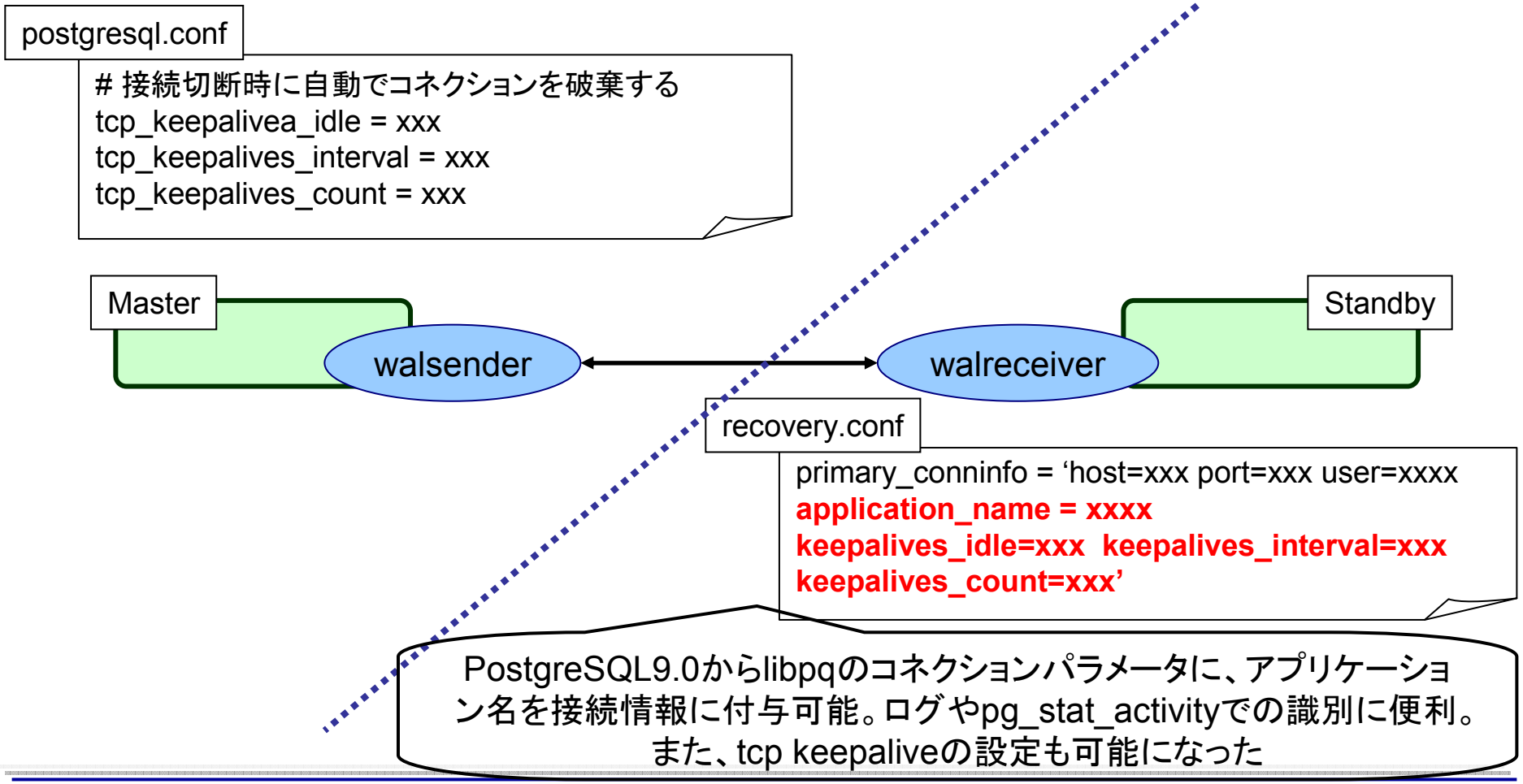
SRのコネクションと認証

- スタンバイ側のwalreceiverは、基本的に普通のクライアントと同じように扱われる
  - パラメータ設定も、通常のクライアントに即した形



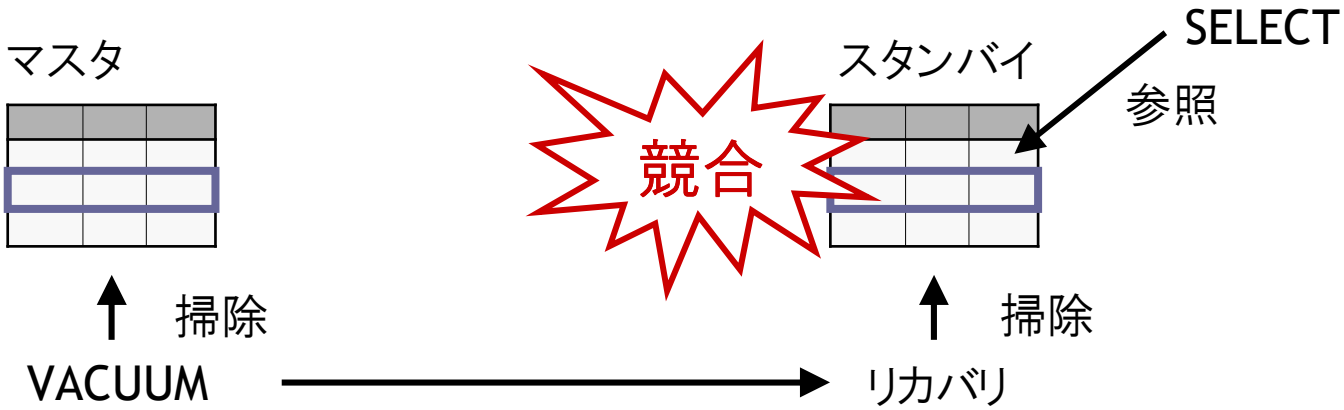
SRのコネクションと認証

- その他、接続系のパラメータも活用可能
  - tcp keepalive 設定はマスタ、スタンバイの双方で可能



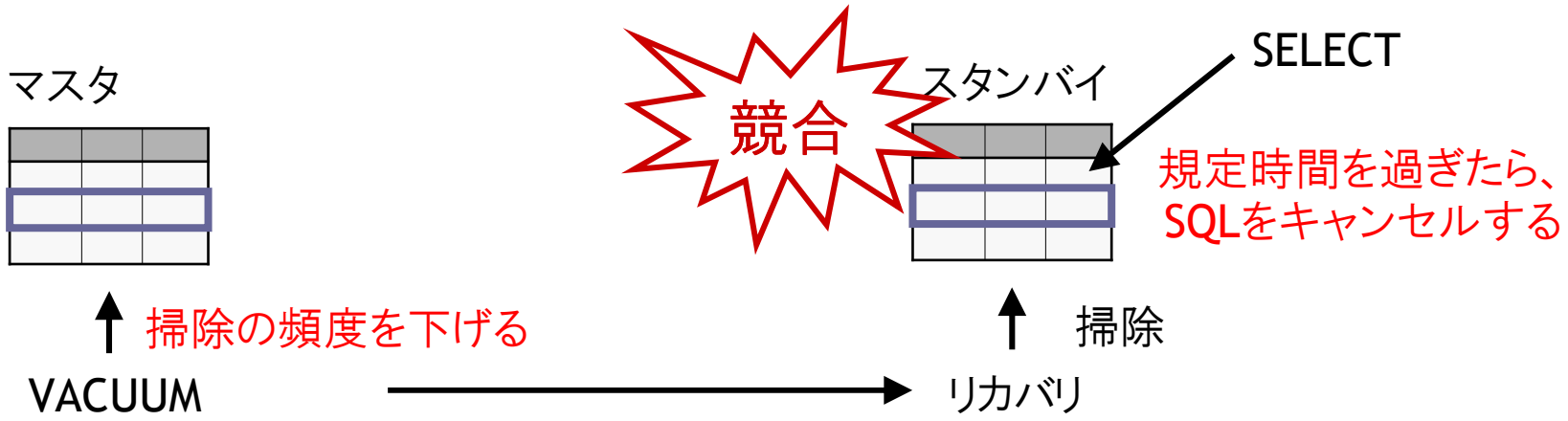
HSの注意点

- スタンバイの参照SQLとリカバリが競合する
  - マスタがVACUUMやHOTでゴミ掃除したデータを、スタンバイの参照SQLがまだ見ていた
    - HOTによるゴミ掃除の頻度は高いため、競合は高頻度で起こる可能性あり
  - スタンバイでアクセス中のデータベースを、マスタがDROPした..なども



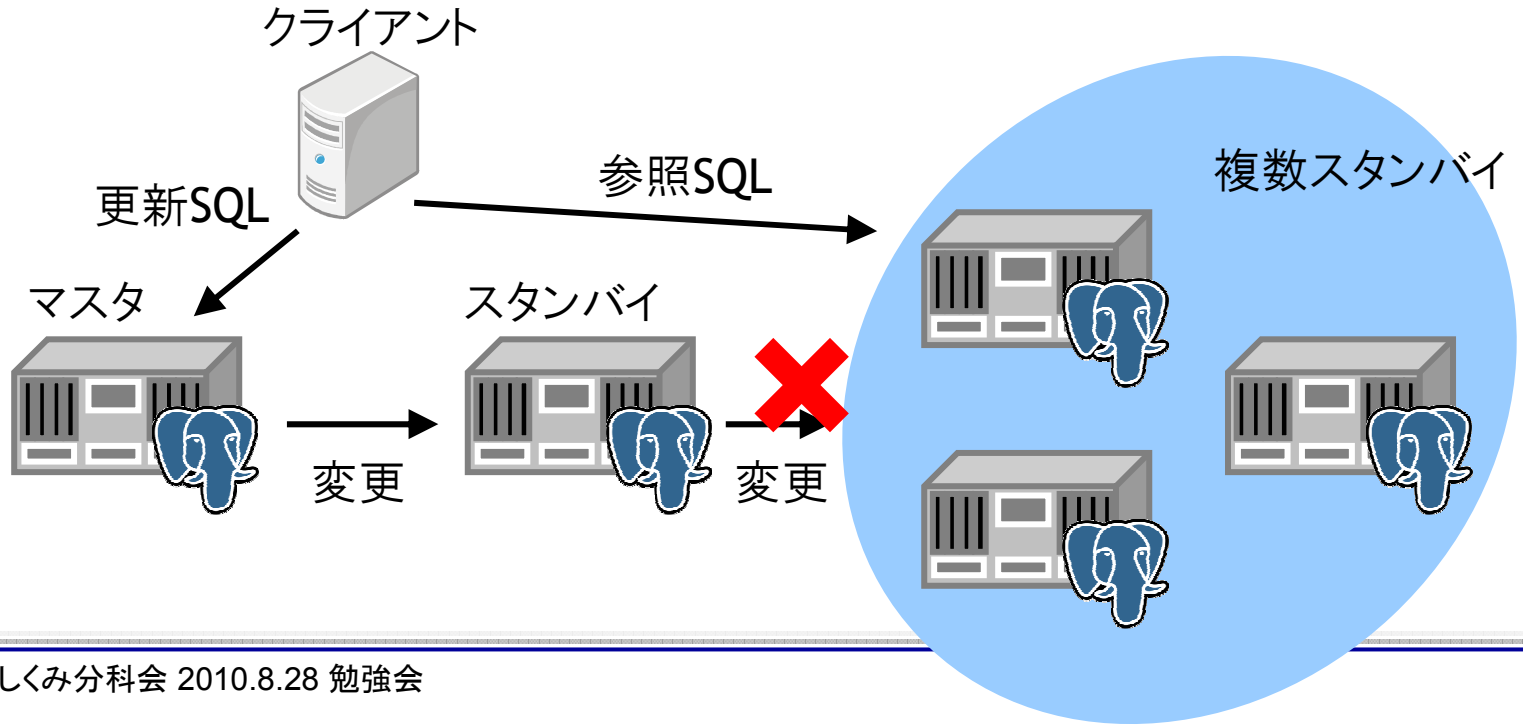
HSの注意点

- 競合中はリカバリが止まる
- 対策はパラメータ設定で可能
  - マスタ側(vacuum_defer_cleanup_age)
    - 競合を抑制するために、VACUUMやHOTによるゴミ掃除の頻度を下げる設定が可能
  - スタンバイ側(max_standby_streaming_delay, max_standby_archive_delay)
    - 何秒競合したら参照SQLをキャンセルして、リカバリを進ませるかをパラメータで設定可能
    - リカバリを優先したい場合は、この値を0に設定しておく



レプリケーションで、できることできないこと

- ⭕ 複数のスタンバイサーバ
  - 参照スケールアウトが可能
  - スタンバイ台数は無限に増やせるが、マスタのオーバーヘッドも増える
- ⭕ 遠隔地へのスタンバイサーバの配置
  - ネットワークで接続されていればOK = 共有ディスク不要
- ✖ スタンバイサーバのカスケード構成
  - スタンバイからスタンバイへのWAL転送はできない



レプリケーションで、できることできないこと

- データベース・クラスタ一括のレプリケーション
- 非同期での変更差分ログ転送
  - フェイルオーバ時にコミット済のデータが消えるかも
- 既存DBのレプリケーション
  - レプリケーション用にテーブル構成を変える必要がない
  - Slony-Iだとテーブルに主キーを追加する必要あり
- 既存APの再利用
  - マスタはすべてのSQLを実行可能
  - pgpool-IIだとノード間でずれるSQLがあり、使用できない
- ✖ レプリケーションするデータベースやテーブルの選択
- ✖ WALに記録されない操作の伝播
  - テーブルスペース用のディレクトリ作成など
- ✖ レプリケーションを使ったローリング・アップグレード
  - 異なるメジャーバージョン(v9.0 vs v9.1)、アーキテクチャ(32bitOS vs 64bitOS)間のレプリケーションは不可

レプリケーションで、できることできないこと

- 手動フェイルオーバ
  - スタンバイからマスタへの昇格
  - ウォームスタンバイと同じようにトリガファイルを作成するだけ
- マスタ/スタンバイをそれぞれ個別に停止/再開
- スタンバイサーバの動的な追加、削除
  - スタンバイサーバの追加・削除時にマスタを停止させる必要がない
- Windowsでのレプリケーション
  - PostgreSQLが動作するプラットホームであればどこでも
- スタンバイサーバの認証
  - 通常のDB接続と同じレベルで認証可能
- ✖ 自動フェイルオーバ
  - PacemakerやpgpooI-II等のHA製品と連携させる必要がある

スタンバイサーバで、できることとできないこと

- ⭕ スタンバイサーバでの参照処理
  - SELECT、SET、BEGIN/COMMIT、pg_dump
- ⭕ スタンバイサーバでのオンライン・バックアップ
  - pg_start/stop_backup()を使えないので、手順は面倒
- ✖ スタンバイサーバでの更新処理
  - INSERT/UPDATE/DELETE, DDL, VACUUM, ANALYZE
  - VACUUM等のメンテナンスはマスタで行い、それがスタンバイに伝播される
  - 更新スケールアウトには使えない
- ✖ 一時テーブルの使用
  - ソート用の一時ファイルの作成は可能
- ✖ スタンバイサーバでのアーカイブログの保存

非同期 vs. 同期

- 非同期 (9.0)
  - マスタの更新処理のオーバーヘッドが小さい
  - マスタサーバでコミットしても、スタンバイサーバにはまだその情報が届いていない
    - 切り替え直前 (数秒) の更新は失われる可能性あり
    - 直前の更新をスタンバイで参照できない
- 同期 (定義はさまざま)
  - 更新処理の伝播のみ同期 (次期バージョン9.1で予定)
    - 高信頼:フェイルオーバーしても更新結果を失わない
    - cf. MySQL 5.5 : Semi-Synchronous Replication
    - cf. DRBD : protocolB, C
  - データベース状態の一致まで同期
    - 多大なオーバーヘッド? スタンバイが先行してしまう?

9.0非同期レプリケーションの使いどころ

- ディザスタリカバリ
  - サイト間の故障検知と自動的な切替等が不要であれば、HA製品との連携は不要
- 参照スケールアウト
  - 少し古いデータを参照するシステム
    - 最新のデータはスタンバイでは見えないかもしれない
    - 例えば、数時間前～1ヶ月前のデータを集計するバッチ等
- warm-standbyを使用するシステムの置き換え

パラメータあれこれ

パラメータあれこれ :マスタサーバ

- postgresql.conf
  - **wal_level** → hot_standby, **archive_mode** → on
    - 利用可能な機能の宣言
    - 性能とトレードオフあり
  - **archive_command**
    - WALをアーカイブするコマンド
  - **vacuum_defer_cleanup_age**
    - 競合を減らすため、どれだけVACUUMを遅らせるか?
  - **max_wal_senders** → 正数
    - スタンバイサーバの数。max_connections も増やすべし
  - **wal_sender_delay**
    - 更新差分の転送遅延

パラメータあれこれ：スタンバイサーバ

- postgresql.conf
  - **hot_standby** → on
    - リカバリ中に参照を許すか?
  - **max_standby_archive_delay** = 30s
  - **max_standby_streaming_delay** = 30s
    - リカバリが参照処理が競合した場合の待ち時間
    - archive_delayは、restoreコマンドでアーカイブからWALを呼ぶ時に使う
- recovery.conf
  - **standby_mode** = 'on'
    - スタンバイサーバとして動作させるか?
    - offの場合はアーカイブリカバリ (WALが尽きたら通常モードに移行)
  - **primary_conninfo**
    - マスタサーバへの接続文字列
  - **restore_command**
    - WALをアーカイブから取り出すコマンド
  - **trigger_file**
    - フェイルオーバー用。マスタから切り離して通常運用を開始するシグナル
    - 設定し忘れても pg_ctl stop → start での再開は可能

パラメータ : wal_level (マスタ側)

WALが少ない

- **minimal** (デフォルト)
  - クラッシュ・リカバリのみサポート
    - アーカイブ・リカバリやストリーミング・レプリケーションは不可
  - 一部の処理でWAL書き出しをスキップするので高速
    - COPY TO, CREATE INDEX, CLUSTER, VACUUM FULL, ALTER TABLE
  - バージョン 8.4 以前の archive_mode = off に相当
- **archive**
  - アーカイブ・リカバリとストリーミング・レプリケーションが可能
  - ホット・スタンバイはできない
  - バージョン 8.4 以前の archive_mode = on に相当
- **hot_standby**
  - archive の機能加えて、ホット・スタンバイが可能

archive_mode も残っているので忘れずに

WALが多い

パラメータ：max_standby_*_delay (スタンバイ側)

- 何をするパラメータか?
  - スタンバイサーバにて、いつまでもリカバリを待たせている
    長時間かかっている参照処理を強制終了させる
- 設定値を増やす利点
  - 時間がかかる処理を行っても強制ロールバックされない
    - 集計処理
    - pg_dumpによる論理バックアップ
- 設定値を増やす欠点
  - レプリケーションの遅延が増える
- チューニングのきっかけ
  - スタンバイサーバでのエラーメッセージ
    - canceling statement due to conflict with recovery
    - terminating connection due to conflict with recovery

バッサリと
強制ロールバック
or 強制切断

パラメータ：vacuum_defer_cleanup_age (マスタ側)

- 何をするパラメータか?
  - max_standby_delayが動作する引き金になる
    VACUUM等による「行の除去処理」を遅延させる
    - 正確には、競合を起こしやすい「最近にUPDATE/DELETEされて不要になった行」の除去処理をしない
- 設定値を増やす利点
  - レプリケーション遅延を抑えながら、強制ロールバックを減らす
- 設定値を増やす欠点
  - ゴミが増える。テーブルが肥大化する
- チューニングのきっかけ
  - スタンバイ側の短時間の処理でロールバックしている
  - マスタ側で数秒間の更新トランザクション数ほどが適当か？

注意が必要なパラメータ

- マスターとスタンバイで一致(もしくはスタンバイの方を多く)させる必要がある設定
  - max_connections : 接続数
  - max_prepared_xacts : 2 Phase Commit を使う場合
  - max_locks_per_xact : テーブル同時アクセス上限
    - 変更した場合は、ベース・バックアップからやり直し ☹
    - マスタとスタンバイを同じ値にしておくのが最善

```
FATAL:  hot standby is not possible because max_prepared_xacts = 0 is a lower setting than
 on the master server (its value was 12)
CONTEXT:  xlog redo parameter change: max_connections=100 max_prepared_xacts=12 max_locks_
per_xact=64 wal_level=hot_standby
LOG:  startup process (PID 22548) exited with exit code 1
LOG:  terminating any other active server processes
```

- 設定変更にサーバの再起動が必要
  - wal_level, archive_mode, max_wal_senders
    - これ以外の多くはオンラインで変更可能 (pg_ctl reload)

その他のTIPS

Q.スタンバイの遅れ具合はわかの?

- A. スタンバイの遅れ具合はSQL or プロセス情報で取得できる
  - SQLは後述
  - それ以外は、9.0の段階ではpsでの目視が主体になる
    - psで取得できる情報
      - streaming <WAL位置情報> : WAL送受信位置
      - waiting for <WALファイル名> : WAL読み取り
      - recovering <WALファイル名> : WALリプレイ

```
postgres: wal receiver process   streaming 0/9000000
postgres: wal sender process postgres 127.0.0.1(46353) streaming 0/9007BD8
```

Q.スタンバイの遅れ具合はわかるの?

- A. リプレイが必要なWALの量で遅れ具合がわかる
  - マスタ:どこまでWALを作成したか?
    - ① pg_current_xlog_location()
  - スタンバイ:どこまでWALを受け取った / リプレイした
    - ② pg_last_xlog_receive_location()
    - ③ pg_last_xlog_replay_location()

- 遅れの計算
  - 送信遅れ : ① ー ②
  - リプレイ遅れ : ① ー ③
    - マスタとスタンバイの両方にSQLでの問合せが必要
    - 引き算の仕方に工夫が要る (32bit+32bit の非線形アドレス)
  - 遅れの時間は時間当たりのWAL生成量から別途計算