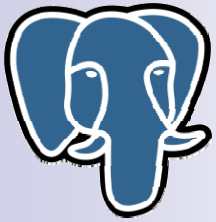
PostgreSQL9.1 同期レプリケーションと Pacemaker による高可用クラスタ化の紹介

PostgreSQL Conference 2012



目次

NTT Open Source Software Center

- PostgreSQL 9.1 同期レプリケーション

- 同期レプリケーションのHAクラスタ化
  - レプリケーション構成のHAクラスタ化 ３大機能
  - 基本動作
  - 運用
  - HAクラスタ化まとめ

NTT Open Source Software Center

PostgreSQL9.1同期レプリケーション

藤井 雅雄
NTT OSSセンタ

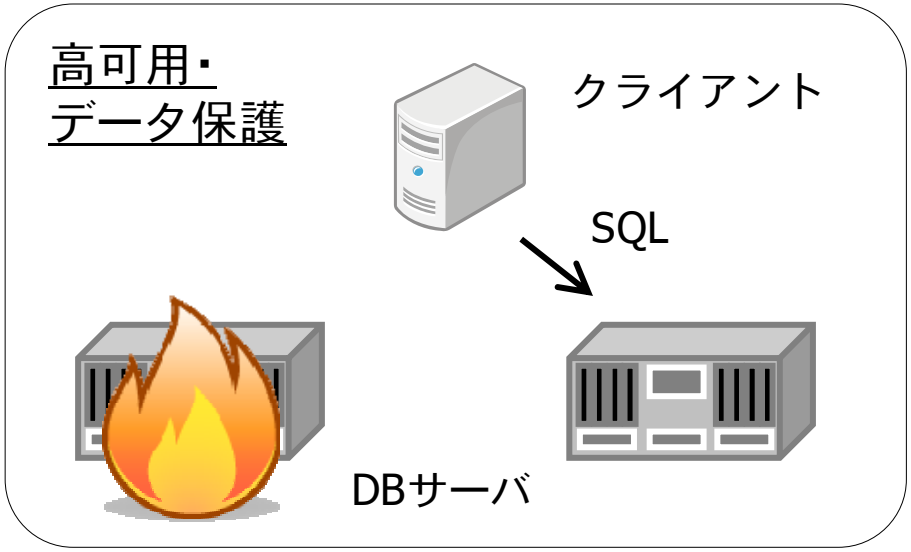
レプリケーションとは?

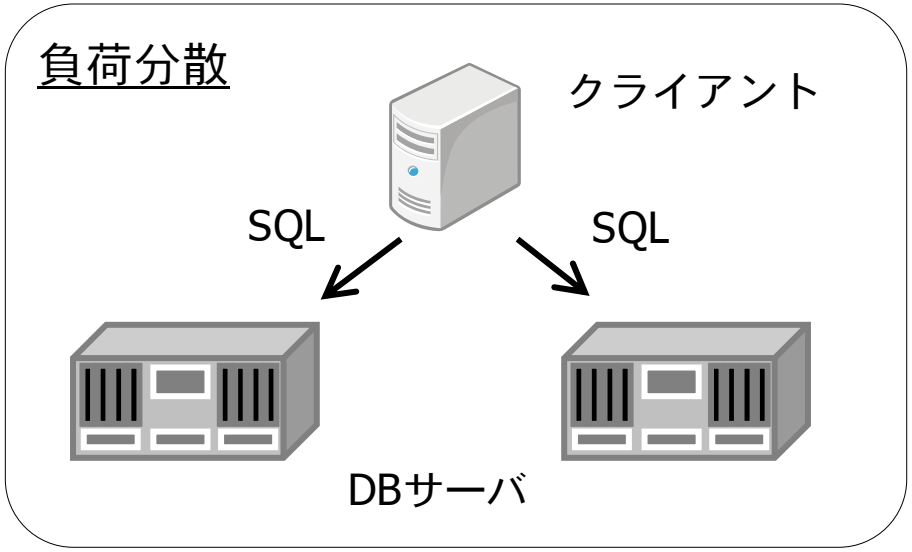
NTT Open Source Software Center

- 複数のサーバにデータベースを自動的に複製する機能

- 用途1: 高可用・データ保護
  - 1台が故障しても、データを失うことなく別サーバが処理を引き継げる
  - システム全体としてデータベースサービスが停止するのを回避できる

- 用途2: 負荷分散
  - SQL実行の負荷を複数のサーバに分散できる
  - 負荷が一箇所に集中しないので、システム全体として性能向上できる





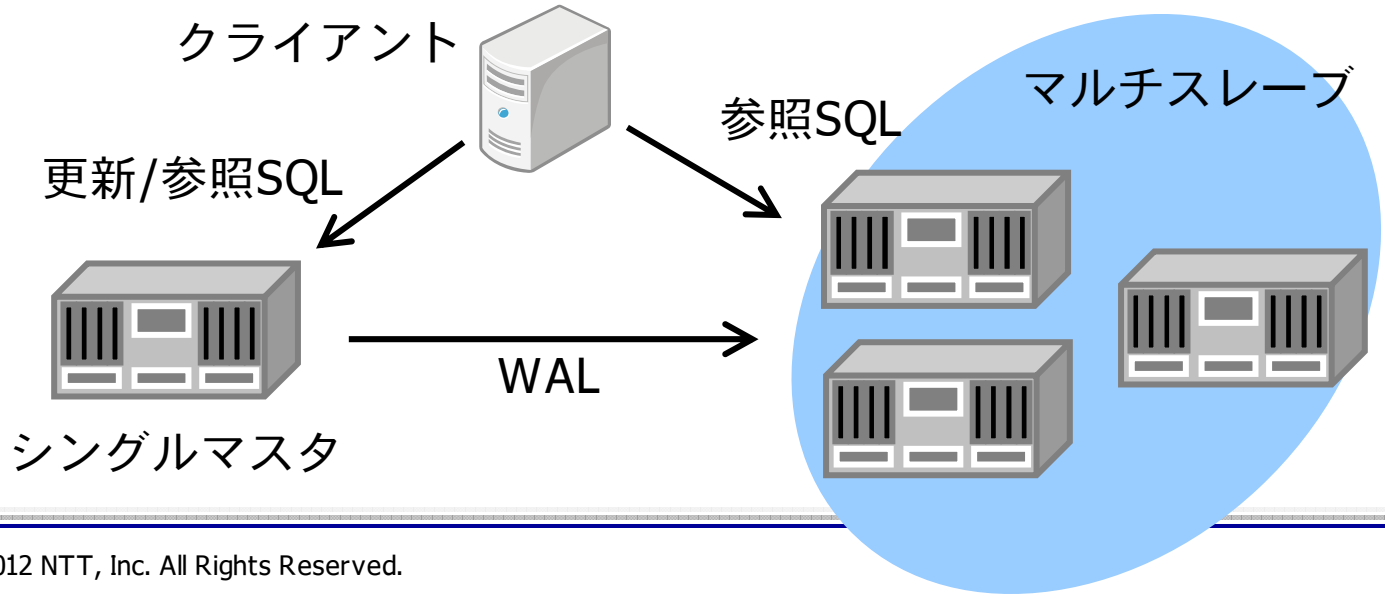
バージョン9.0レプリケーションの復習

NTT Open Source Software Center

- バージョン9.0から本体組み込みのレプリケーションを利用可能

- 特徴1: シングルマスタ/マルチスレーブ構成
  - マスタは更新/参照SQL実行可能
  - スレーブは参照SQLのみ実行可能

- 特徴2: ログシッピング
  - WAL(トランザクションログ)をマスタからスレーブに転送
  - スレーブはWALをリカバリし続けることでデータベースを複製

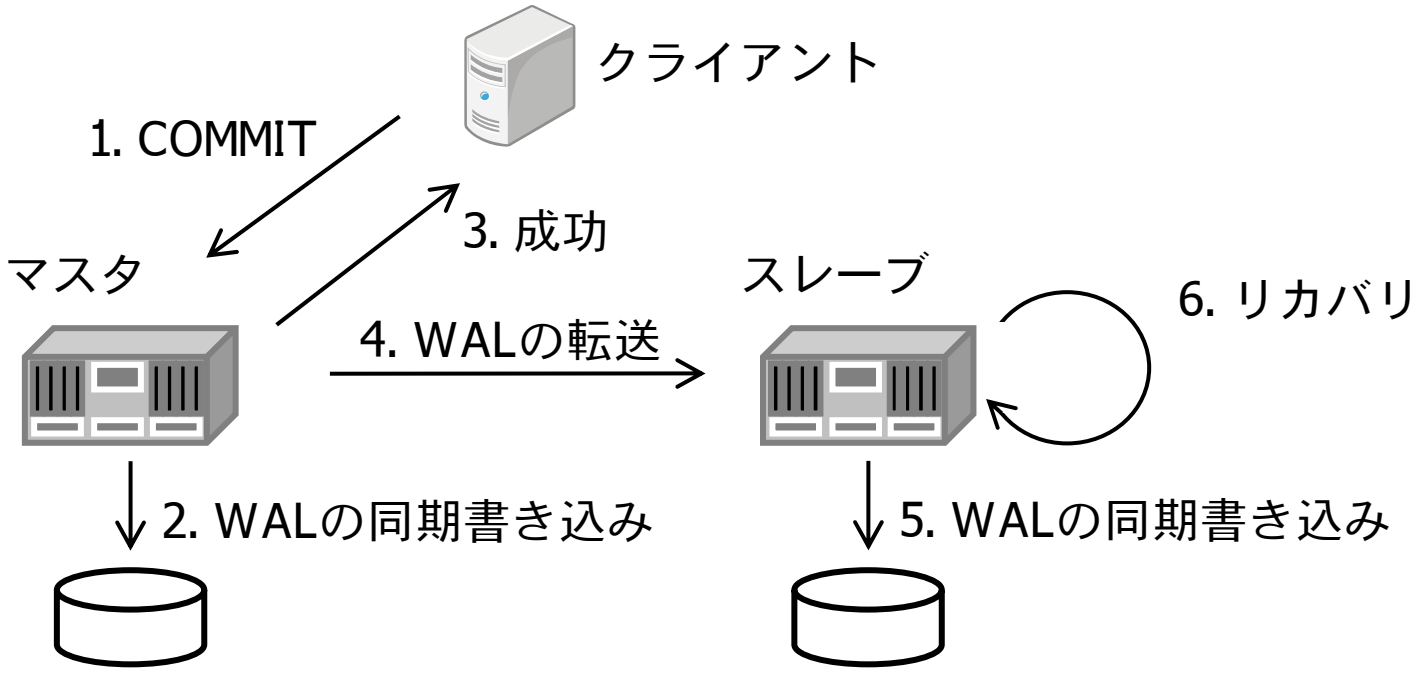


バージョン9.0レプリケーションの復習

NTT Open Source Software Center

- 特徴3: 非同期レプリケーション
  - マスタは、WALスレーブに届いたのを確認せずにCOMMITの成功をクライアントに返却する
  - レプリケーションの性能オーバーヘッドは小さい
  - フェイルオーバ時に直前のCOMMIT済のデータを失う可能性がある
  - スレーブで参照SQL実行時、古いデータが返却される可能性がある

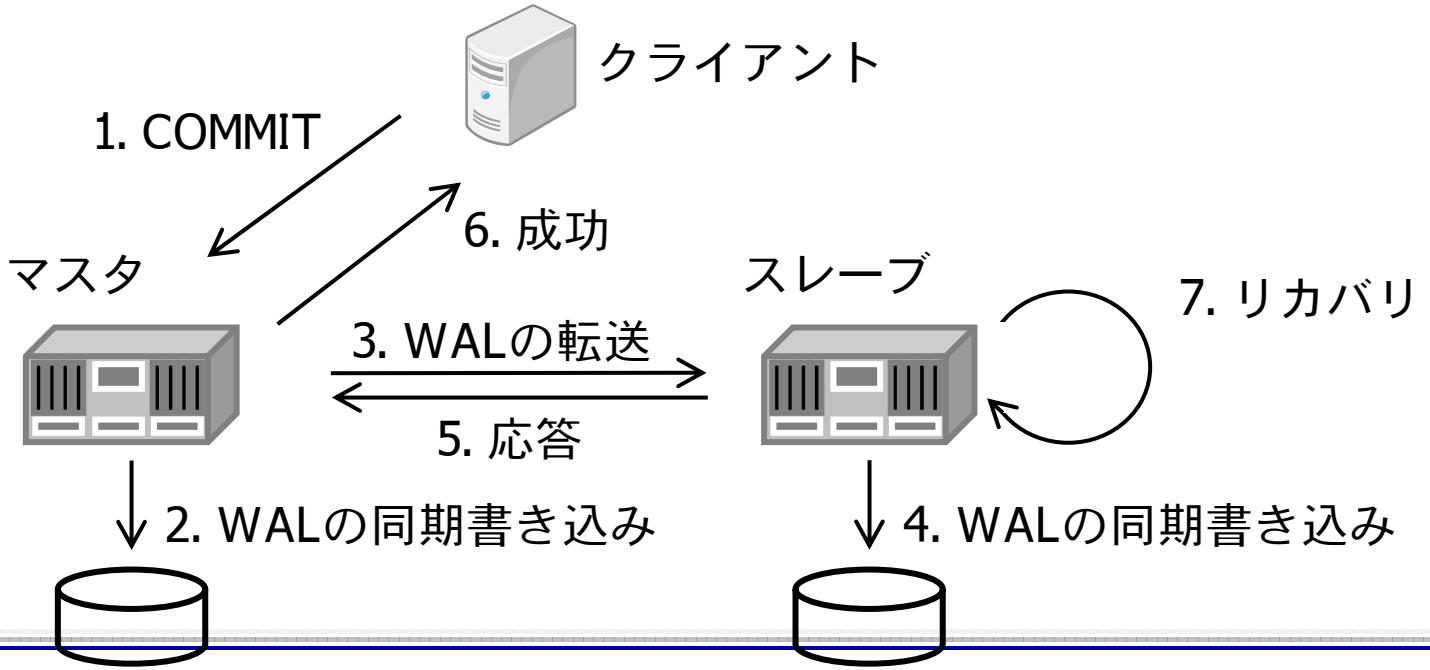
  ➔ 高可用・データ保護の用途には使いづらい☹



バージョン9.1レプリケーション

- レプリケーションのモードを「非同期」「同期」から選択可能
- 同期レプリケーション
  - マスタは、WALがスレーブに届いた(WALをディスクに同期書き込み)のを確認してからCOMMITの成功をクライアントに返却する
  - レプリケーションの性能オーバーヘッドは大きい
  - フェイルオーバ時にCOMMIT済のデータは失われない
  - スレーブで参照SQL実行時、古いデータが返却される可能性がある

➔ 高可用・データ保護の用途に使える☺

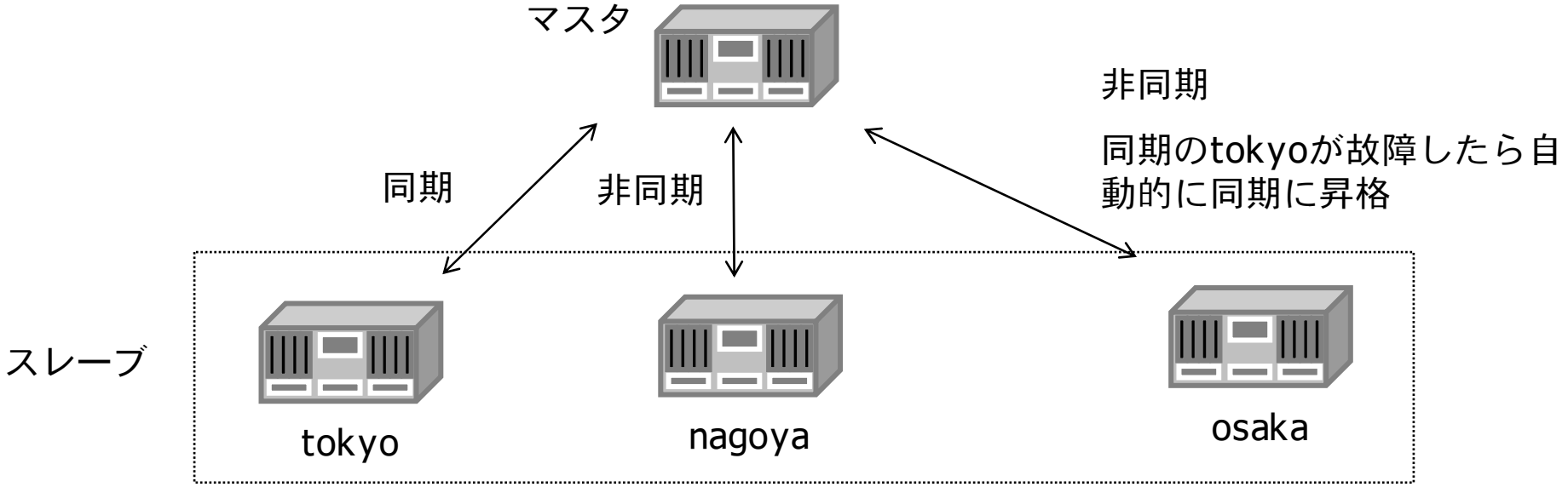


同期レプリケーションの設定

- スタンバイごとにレプリケーションのモードを設定
  - 同期レプリケーションを行うスレーブの名前をsynchronous_standby_namesに設定
  - 同期レプリケーションを実行できるスレーブは同時に1台のみ
  - より先頭に設定されている名前のスレーブが優先的に同期レプリケーションを行う

postgresql.conf

synchronous_standby_names = ‘tokyo, osaka’



recovery.conf

primary_conninfo = ‘application_name=“tokyo”’

トランザクション単位のデータ保護レベルの設定

NTT Open Source Software Center

- synchronous_commitでトランザクション単位にデータ保護レベルを設定可能
  - SET synchronous_commit TO xxx;

設定値	スレーブのモード	マスタのWAL書き込み	スレーブのWAL書き込み
off	ALL		
local	ALL	待つ	
on	非同期	待つ	
	同期	待つ	待つ

- トランザクションの重要度に応じてデータ保護レベルを変更
  - お金を扱うトランザクションでは設定値 on
  - 故障時に一部更新結果が消えてもよいトランザクションでは local / off に設定して性能向上

スレーブ故障時の挙動

NTT Open Source Software Center

- 非同期レプリケーション
  - トランザクションはマスタ単独で処理
- 同期レプリケーション
  - スレーブにWALを書けるまでトランザクションは停止!!
  - 同期レプリケーションの原則「マスタとスレーブ両方にWALを書くまでCOMMITを成功させない」を保証するため
  - マスタ単独でトランザクションを完了するには、レプリケーションのモードを「非同期」に設定変更する必要がある
    synchronous_standby_namesの設定値を空にして設定ファイルをリロード(pg_ctl reload)
    注意: synchronous_commitによる「非同期」への設定変更では、故障発生当時に実行中だったトランザクションを再開できないため、synchronous_standby_namesで設定変更すること
  - スレーブ故障時に自動的にマスタを単独稼働させる機能はPostgreSQLにはない

  ➔ Pacemakerで自動的にそれをやってくれる機能については後述
  - マスタ故障でフェイルオーバが発生した場合も同じ挙動になることに注意
  - 同期レプリケーションのスレーブが複数ある場合は、トランザクションは停止しない
    別のスレーブが「同期」に自動的に昇格し、そのスレーブにWALを書けた時点でトランザクションは完了するため

pg_stat_replication

NTT Open Source Software Center

- レプリケーションの様子を確認できるビュー
  - ビューを確認できるのはマスタのみ
  - スタンバイからの接続ごとに1レコード表示

```
=# SELECT * FROM pg_stat_replication;
-[ RECORD 1 ]----+---------------------------
procpid          | 26531
usesysid         | 10
usename          | postgres          ←スレーブがレプリケーション接続に使ったユーザの名前
application_name | tokyo             ←スレーブの名前
client_addr      | 192.168.1.2       ←スレーブのIPアドレス
client_hostname  |
client_port      | 39654             ←スレーブのポート番号
backend_start    | 2012-02-01 18:54:49.429459+09 ←レプリケーションの開始時刻
state            | streaming         ←レプリケーションの状態
                                       startup: 接続中、catchup: スレーブ追いつき途中、
                                       streaming: スレーブ追いつき済
sent_location    | 0/406E7CC         ←マスタがどこまでWALを送信したかの位置
write_location   | 0/406E7CC         ←スレーブがどこまでWALを受信したかの位置
flush_location   | 0/406E7CC         ←スレーブがどこまでWALを同期書き込みしたかの位置
replay_location  | 0/406E1B0         ←スレーブがどこまでWALをリカバリしたかの位置
sync_priority    | 0                 ←同期レプリケーションの優先度
                                       0: 非同期、1〜?: 同期(値が小さいほど優先度高)
sync_state       | async             ←スレーブのレプリケーションモード
                                       async: 非同期、sync: 同期、
                                       potential: 非同期だが、同期に昇格するかも
```

v9.1のレプリケーション関連の機能(同期レプリケーション以外)

NTT Open Source Software Center

- pg_basebackup
  - オンライン物理バックアップを取得するコマンド
- pg_ctl promote
  - スレーブをマスタに昇格させるコマンド
  - 9.0まではtrigger_fileを作成する必要があったが、9.1からはコマンド1発
- リカバリの停止・再開
  - SELECT pg_xlog_replay_pause()　　：リカバリを一時停止
  - SELECT pg_xlog_replay_resume()　　：リカバリを再開
  - 停止するのはスレーブのリカバリであり、WALの転送は停止しないことに注意
- hot_standby_feedback
  - スレーブで実行された参照SQLはリカバリと競合する可能性がある
  - 例えば、参照SQLがまだ見ているデータをリカバリが削除しようとする場合など
  - 競合が発生すると、参照SQLがキャンセルされたり、リカバリが停止したりする
  - 最も発生確率の高い競合を発生させなくするパラメータ。onに設定するだけ
  - ただし、スレーブのトランザクション実行状況がマスタでのVACUUMやHOTに影響する
- replication_timeout
  - マスタがスレーブの故障を検知するためのタイムアウト
  - 9.0ではkeepaliveの設定である程度検知できたが、完璧でなかった

v9.2のレプリケーション関係機能

NTT Open Source Software Center

- より高速な同期レプリケーション
  - スレーブがWALを受信した(正確にはwrite(2)で書き込んだ)時点で、レプリケーション完了の応答をマスタに返却するモード
  - 重い処理である「スレーブでのWALの同期書き込み(fsync)」をマスタは待つ必要がなくなり、レプリケーションのオーバーヘッドが小さくなる
- カスケードレプリケーション
  - スレーブからスレーブにレプリケーション
  - マスタに接続するスレーブ台数を減らし、マスタの負荷を小さくできる
  - 非同期のみ
- スレーブからのオンライン物理バックアップ
  - pg_basebackupでマスタからだけでなくスレーブからバックアップを取得
  - マスタに集中していたバックアップの負荷をスレーブに負荷分散できる
- pg_receivexlog
  - マスタからWALを受信しディスクに書き続けるツール
  - オペミス対策でWALの二重化などの用途

まとめ

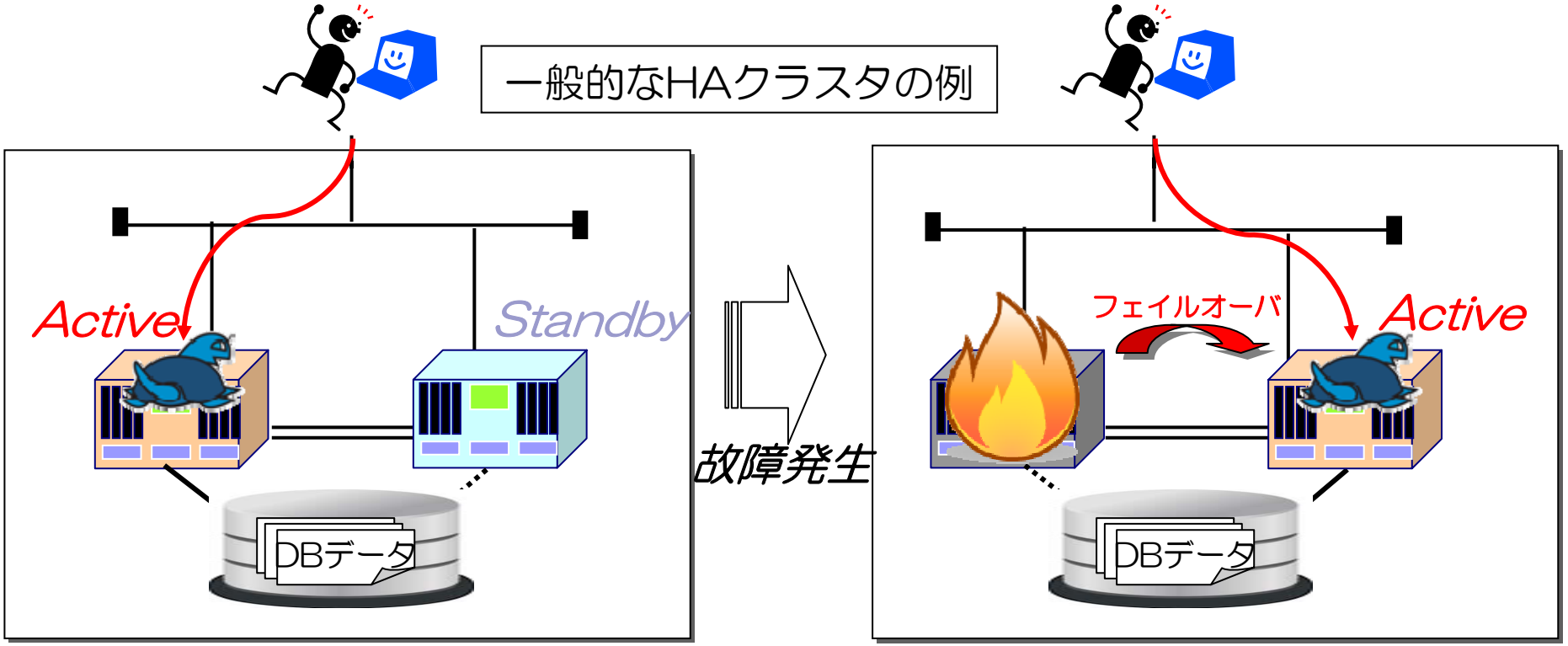
NTT Open Source Software Center

- バージョン9.1からレプリケーションのモードを「非同期」「同期」選択可能
- レプリケーションのモードはスレーブ単位、トランザクション単位で設定
- 同期レプリケーション実行中にサーバが故障すると、トランザクションは停止する
- レプリケーションの様子はpg_stat_replicationで確認できる

PostgreSQL同期レプリケーション
+
Pacemaker
による高可用クラスタ化

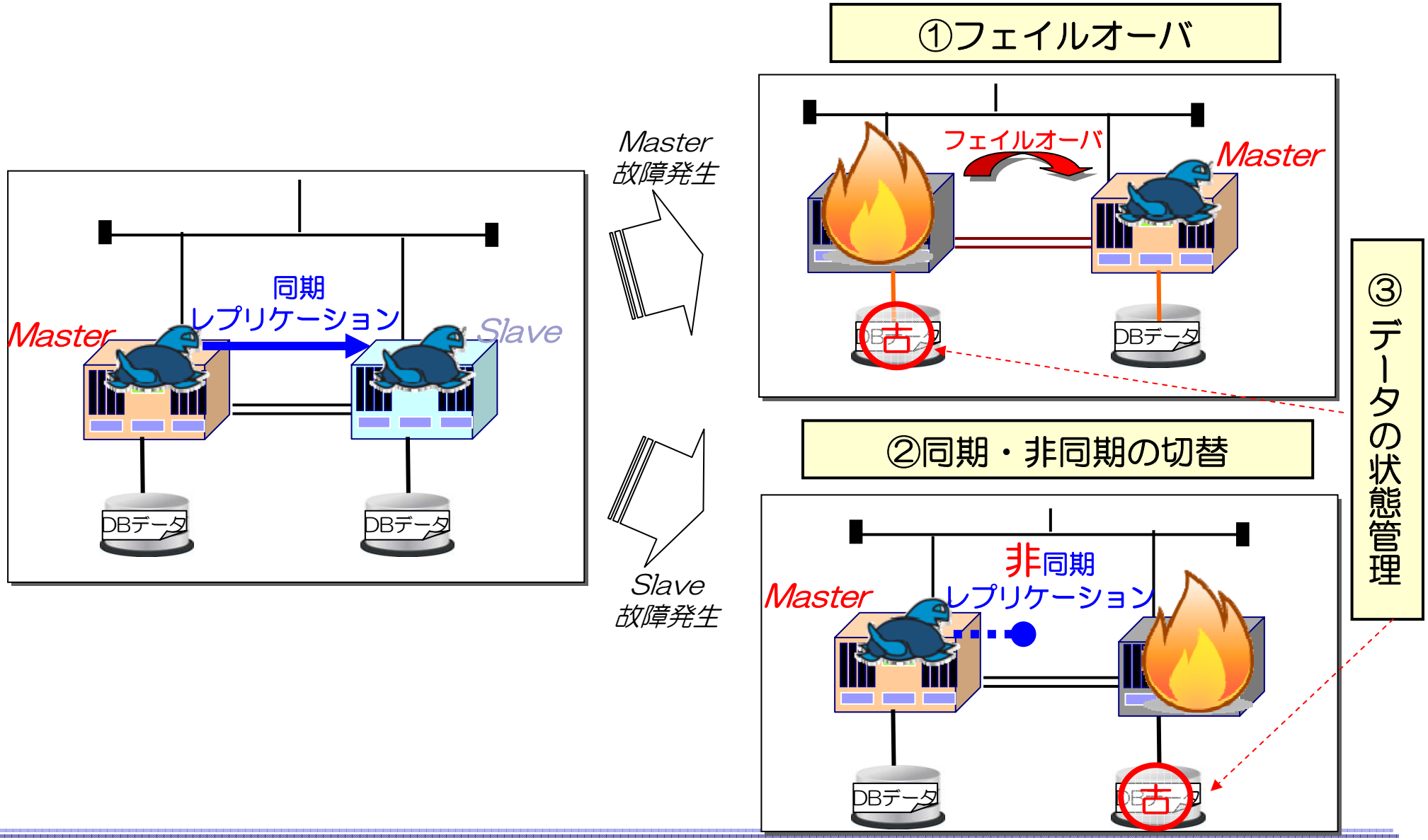
高可用(HA)クラスタ？ Pacemaker?

- 複数台のサーバーを用意し、壊れた時に他のサーバーでサービスを引き継ぐことでシステムの可用性(Availability)を高める手段
  - Pacemakerは"Linux-HA Japan"コミュニティ提供のソフトウェア
  - 市販製品では、ClusterPro, LifeKeeper, Red Hat High-Availabilityアドオン 等



レプリケーション構成のHAクラスタ化 ３大機能

NTT Open Source Software Center

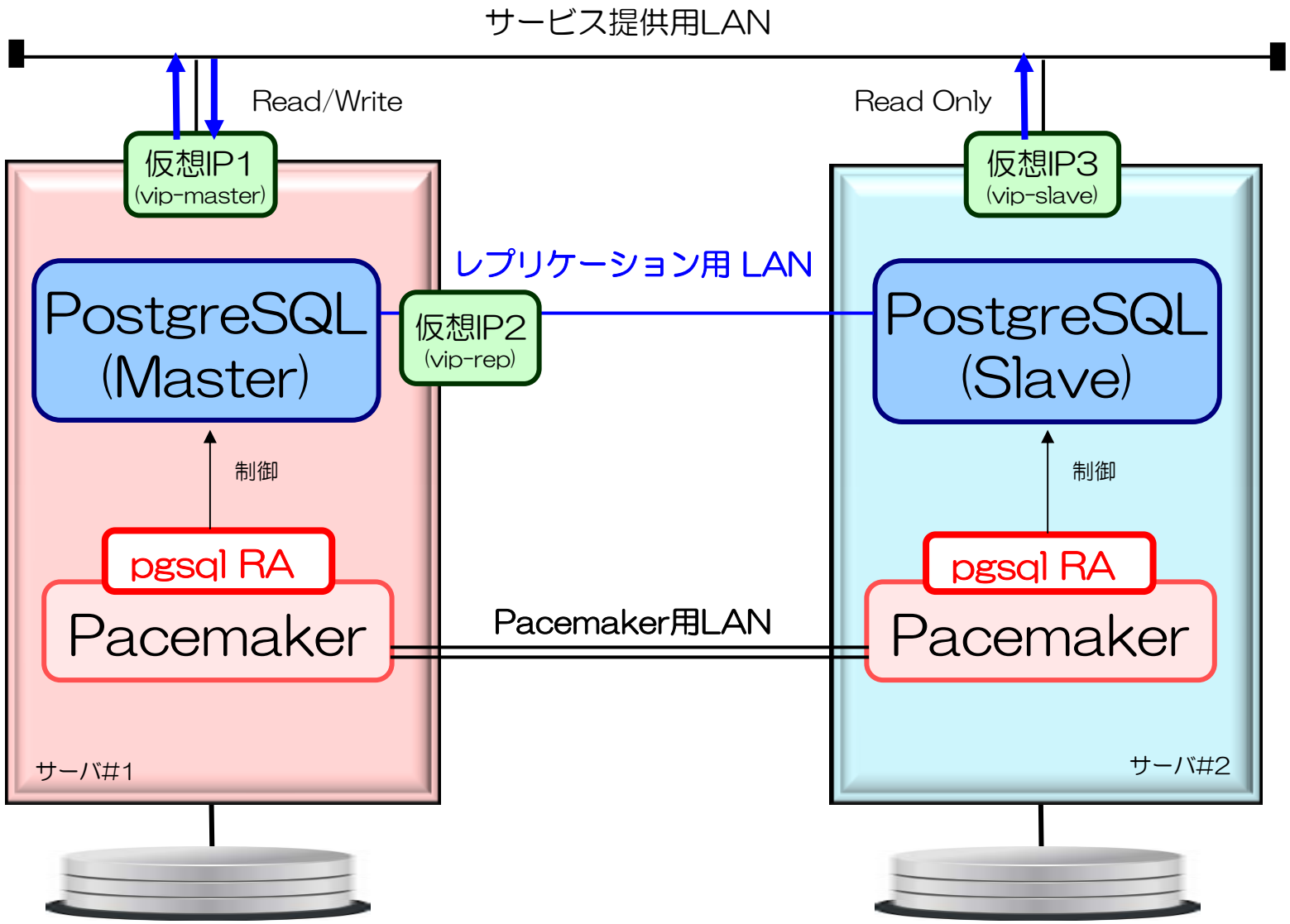


想定している適用箇所

NTT Open Source Software Center

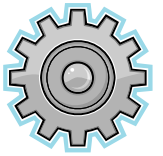
- クラスタ化に費用を割けられない
  - NAS, 共有ディスク, NFSサーバ等が不要
- 参照クエリを負荷分散したいがSlaveの故障にも備えたい
  - Slave故障時にMasterで参照クエリを処理させる
    - アプリケーションは接続先を意識しなくてよくなる
- 高負荷時のフェイルオーバ時間を出来る限り短くしたい
  - フェイルオーバ後のクラッシュリカバリが(ほぼ)必要なくなる
- 某高速ストレージを使いたい
  - PCI Express接続ではデータを共有できない
  - 性能を求めるにはレプリケーション用のネットワークもそれなりの性能の物を揃える必要あり

基本構成



※STONITH用LANの記述は省略しているが設定を強く推奨

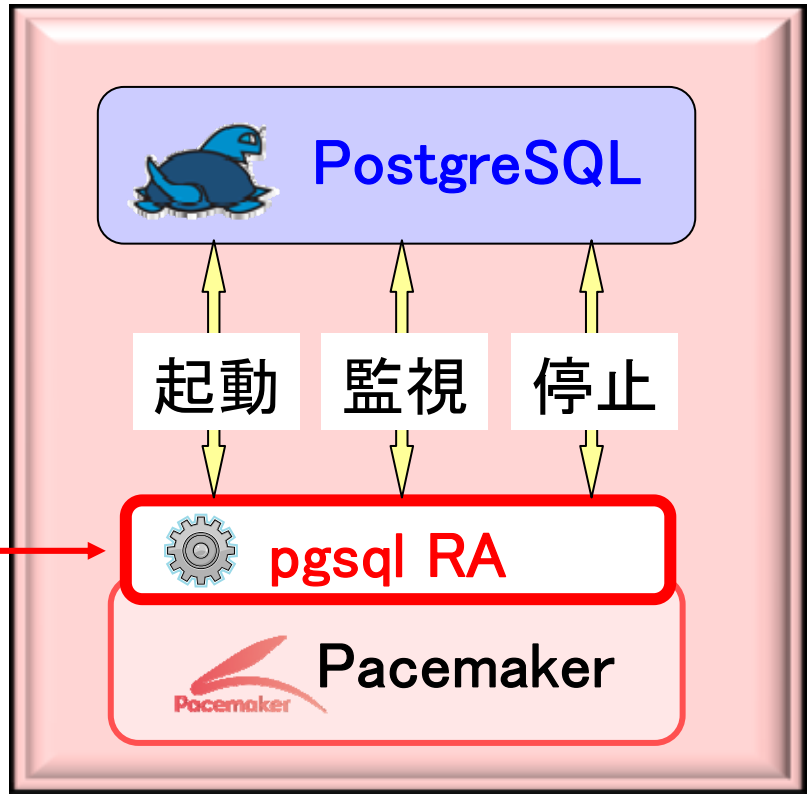
リソースエージェント(RA)



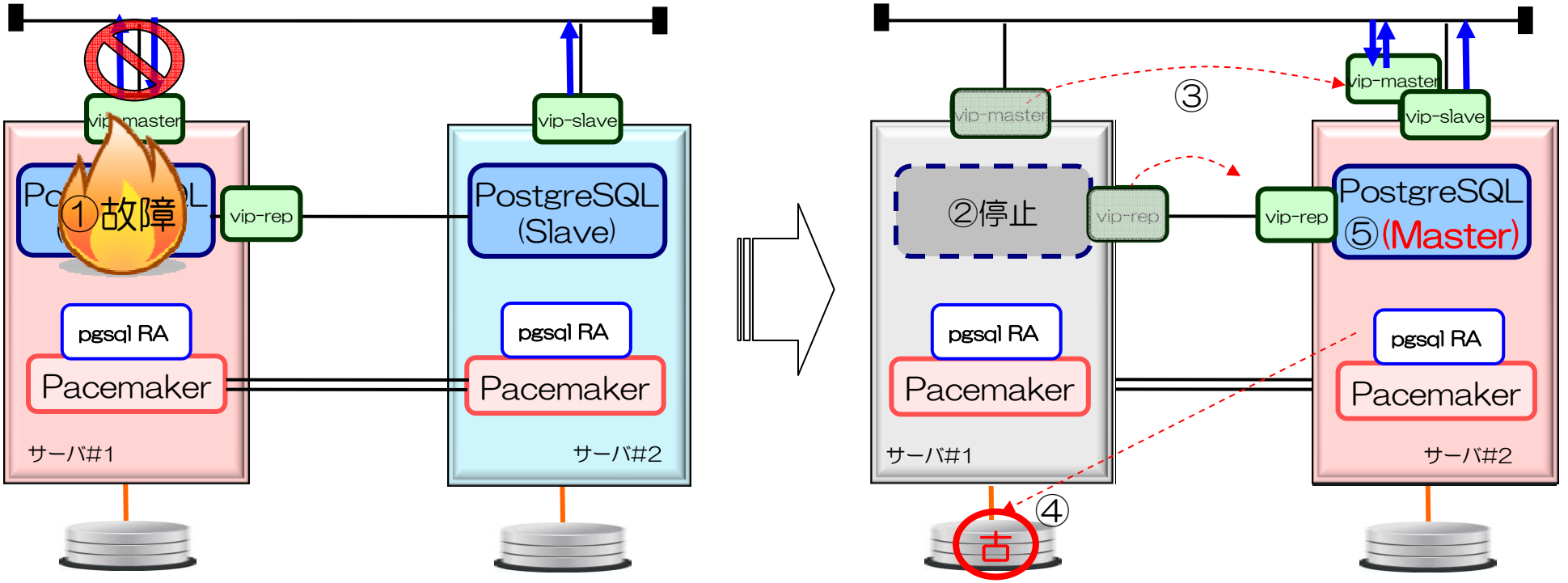
Pacemakerと制御対象のアプリケーション間を
仲介するプログラム。主にシェルスクリプトで記述。

既存のPostgreSQL用RA(pgsql)は、
Active-Standby構成に必要な関数
(起動(start)・監視(monitor)・停止(stop))を
実装

レプリケーション構成を制御するために
この pgsql RA の機能を拡張



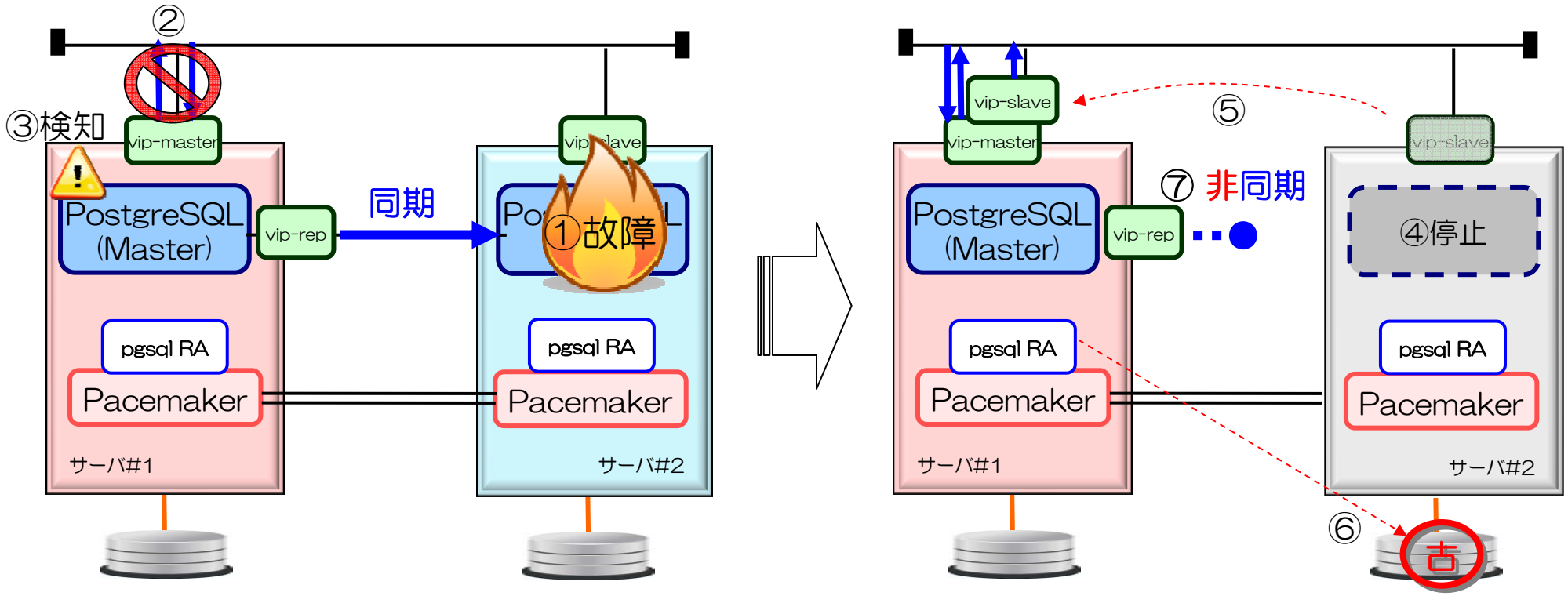
基本動作1：Masterのフェイルオーバ



① #1のPostgreSQLの故障を検知

② #1のPostgreSQLを停止
③ #1の仮想IPを#2に付け替え
④ #1のデータが古いことを記録
⑤ #2のPostgreSQLをMasterに昇格

基本動作2：同期・非同期の切替



① Slaveの故障発生
② Masterのトランザクション停止
～ レプリケーションのタイムアウト待ち～
③ #1でレプリケーション切断を検知

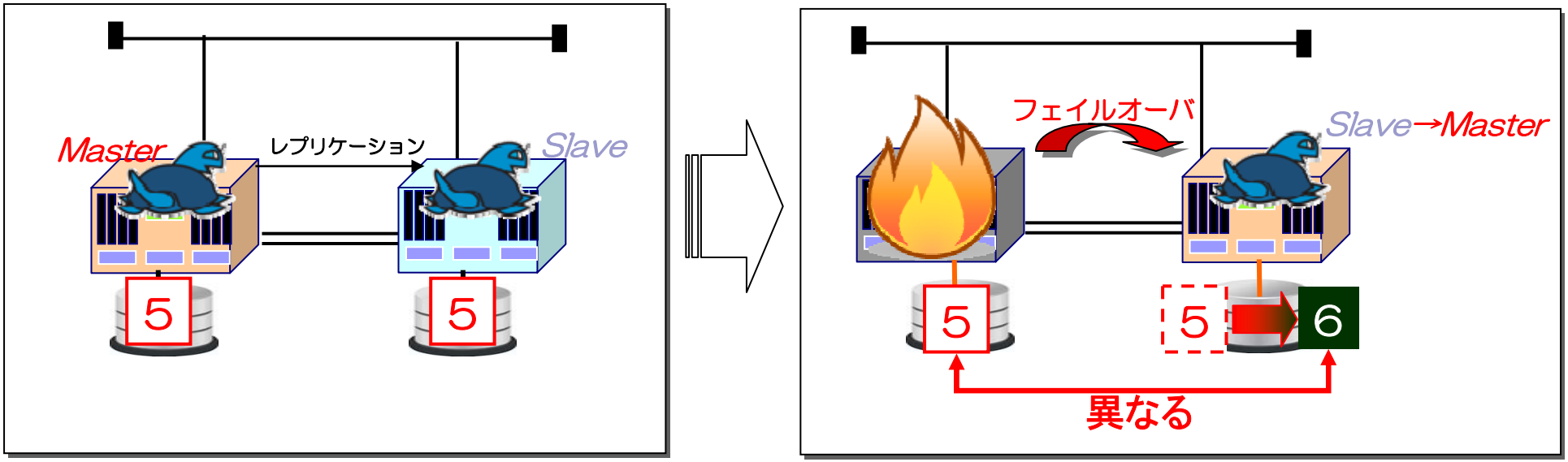
SELECT * from pg_stat_replication

④ #2のPostgreSQLを停止
⑤ #2の仮想IPを#1に付け替え
⑥ #2のデータが古いことを記録
⑦ #1のPostgreSQLを非同期に変更
→ トランザクション再開

TimelineID

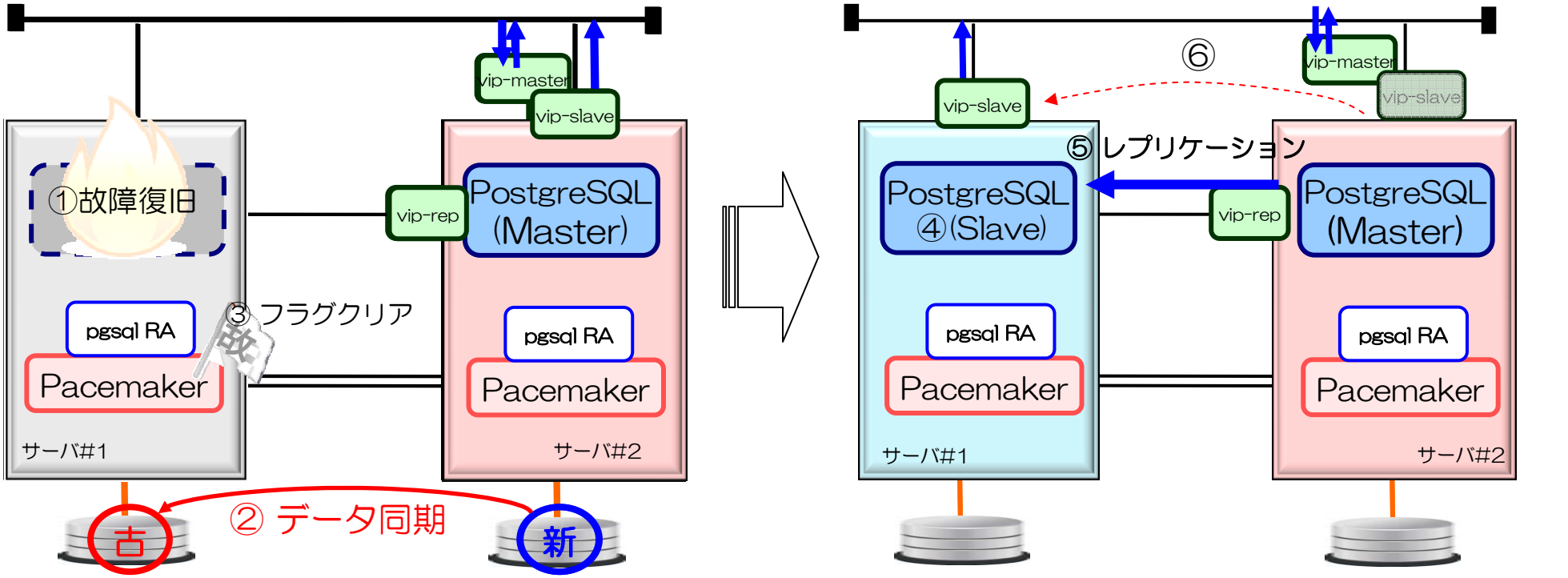
- SlaveからMasterへ昇格した際インクリメントされる数値
- TimelineIDが異なるとレプリケーションできない

フェイルオーバ時



TimelineIDをそろえるには手動でのデータ同期(推奨) or WALアーカイブの共有が必要

運用1: フェイルオーバ後の復旧

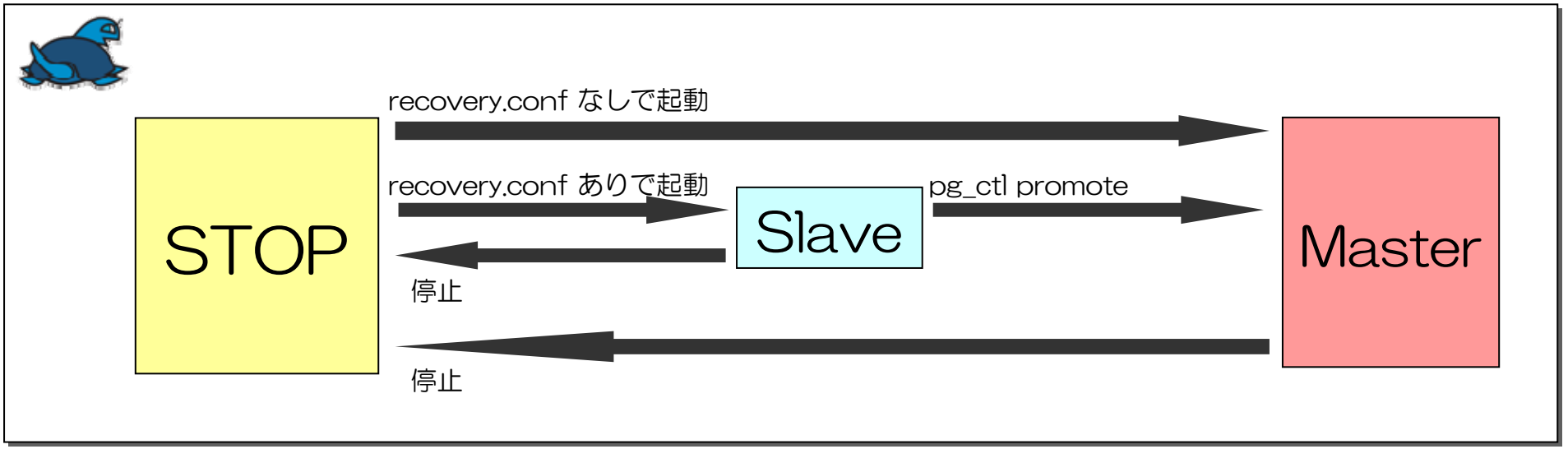


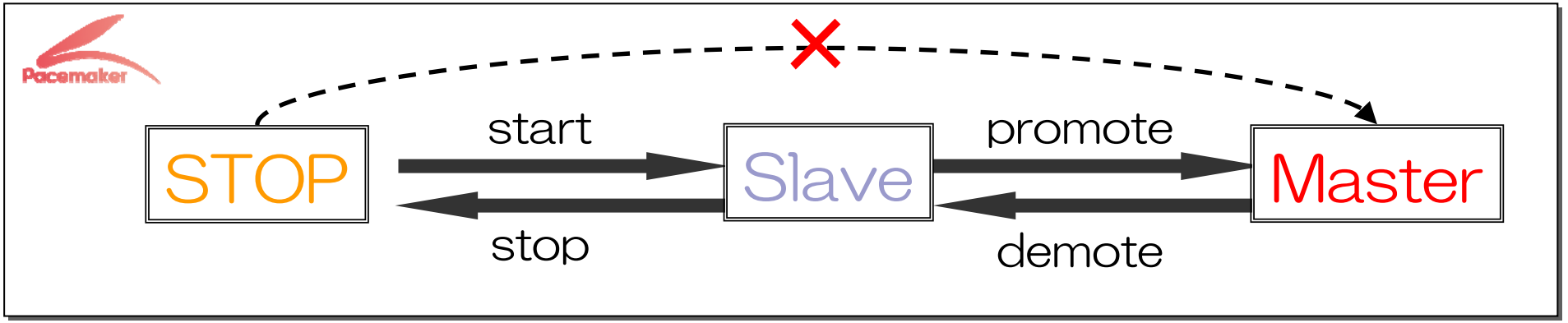
① 故障の復旧
② #1のデータを#2と同期
③ #1のPacemaker上の故障フラグをクリア

(手動)

④ #1のPostgreSQLを起動
⑤ レプリケーション開始
⑥ #2の仮想IP(Slave用)を#1に付け替え

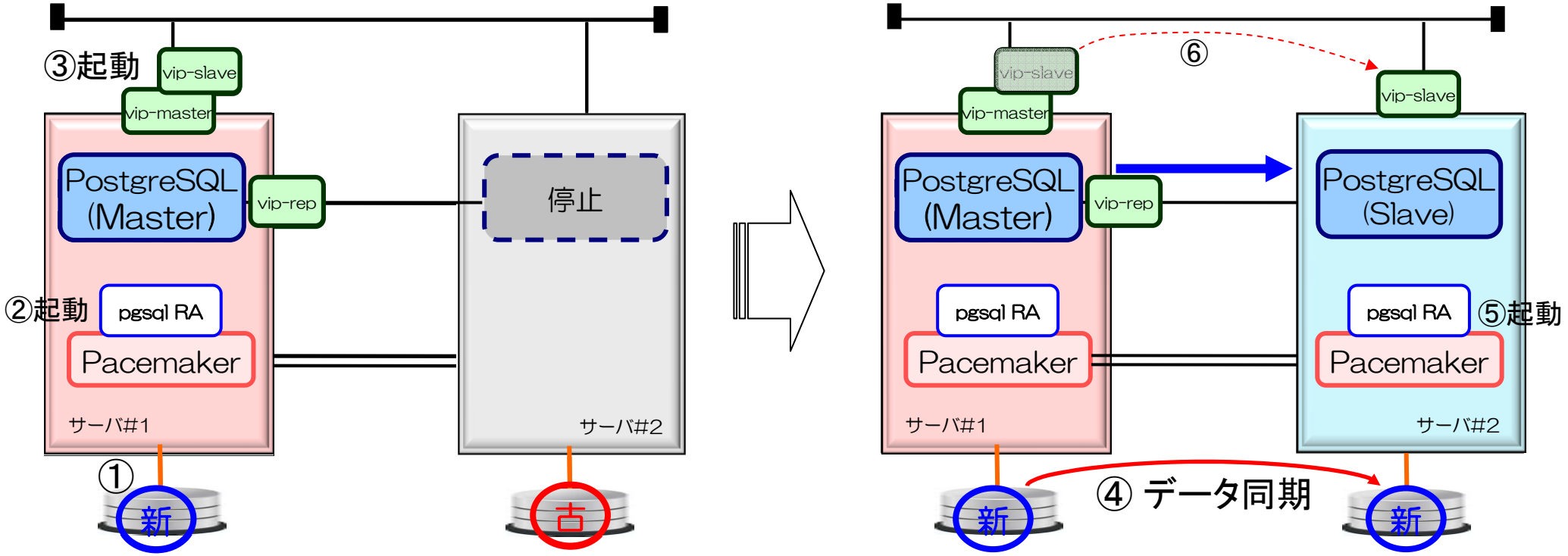
PostgreSQLとPacemakerの状態遷移





PostgreSQLのMasterは必ずSlaveを経由して起動される
→ Master起動時もTimelineIDがインクリメントされる

運用2：起動



① データが新しい方のサーバーを選択
② 選択したサーバのPacemakerを起動
→ Slaveで起動 → Masterに遷移
③ 仮想IPが#1で起動

(手動)

④ #2のデータを#1と同期
⑤ Pacemaker起動
→ レプリケーション開始
⑥ 仮想IP(Slave用)が移動

(手動)

高可用クラスタ化まとめ

NTT Open Source Software Center

- ３大機能
  - PRIのフェイルオーバ
  - 同期・非同期の切替
  - データの状態管理
- TimelineIDがずれるとレプリケーションできないため、手動でのデータの同期(推奨) or WALアーカイブの共有が必要
  - フェイルオーバ時やMaster起動時にずれが発生