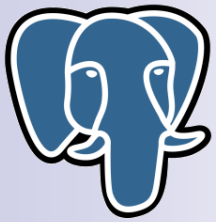
HAクラスタで PostgreSQLレプリケーション構成の 高可用化

2012/11/30 PostgreSQL Day 2012

NTT OSSセンタ
近藤光正　松尾隆利

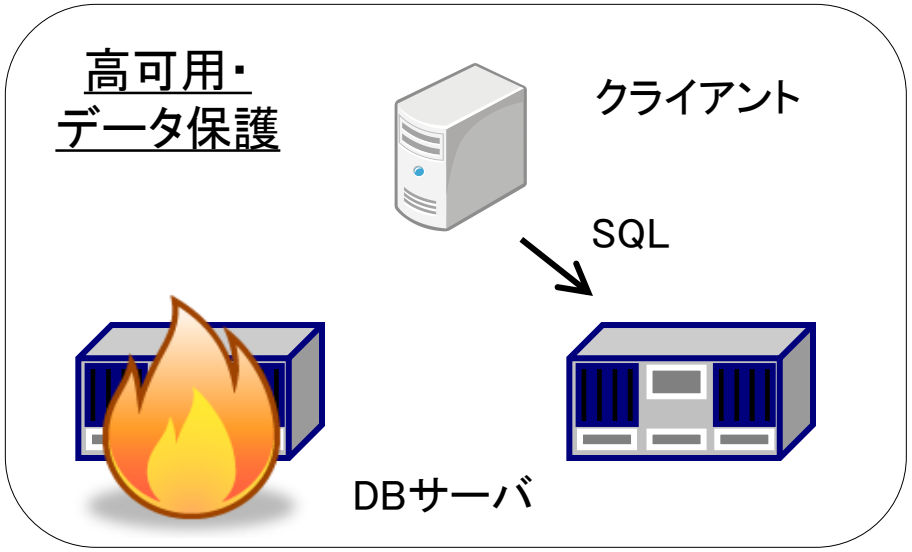


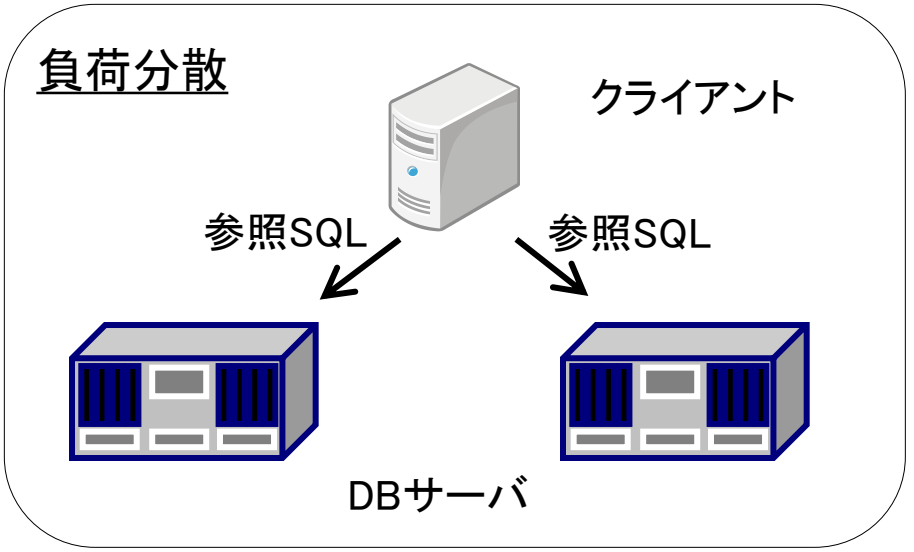
レプリケーションとは?

- 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
  - 現用系が故障しても、データを失うことなく待機系が現用系の処理を引き継げる
  - 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
  - 参照 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



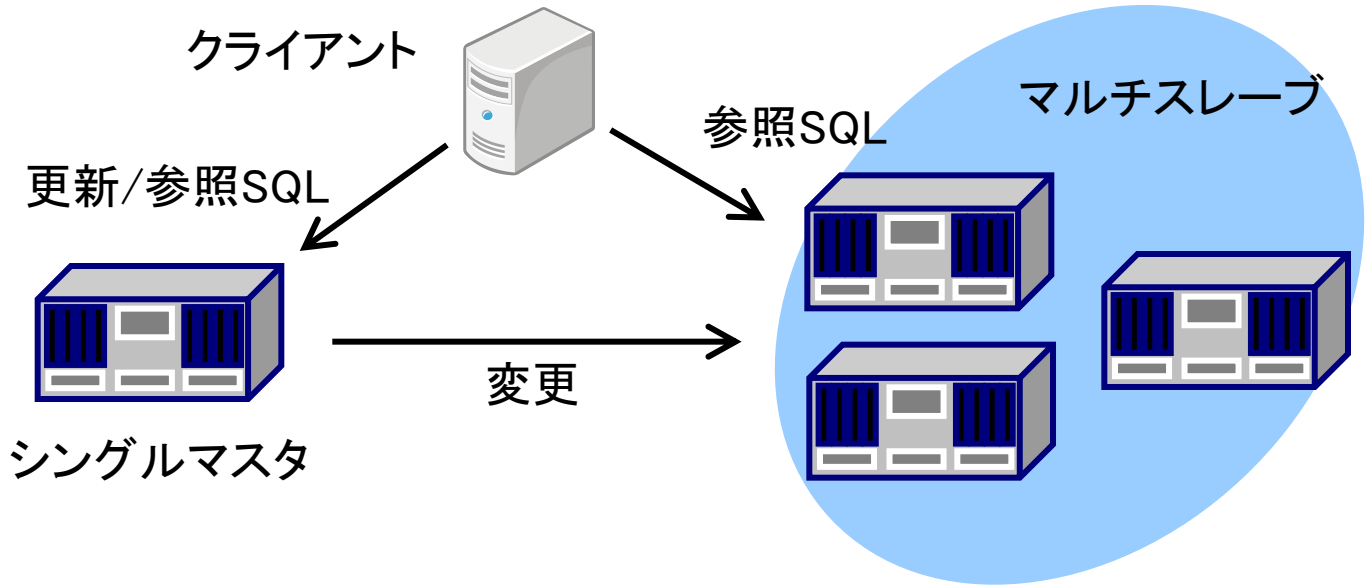


PG本体機能のレプリケーション構成

□ シングルマスタ/マルチスレーブ構成

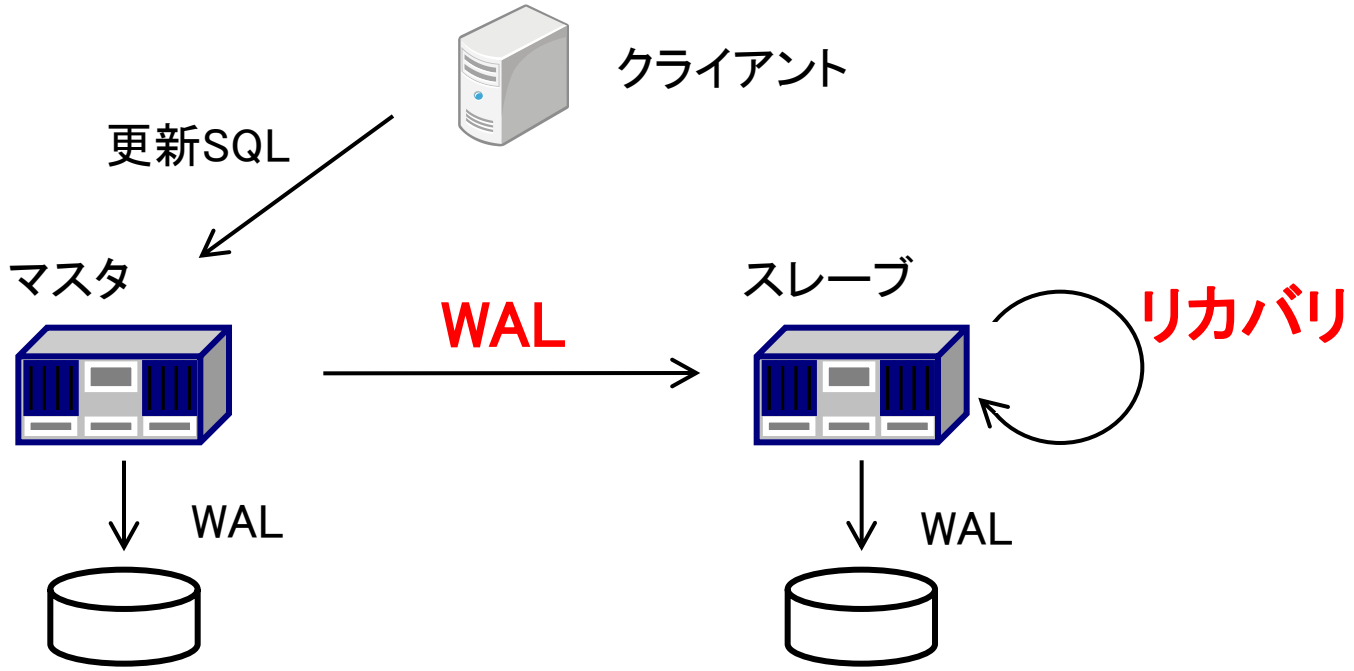
- マスタは、更新SQLと参照SQLを実行可能
- スレーブは、参照SQLのみ実行可能

➔参照系処理のスケールアウトに利用可能



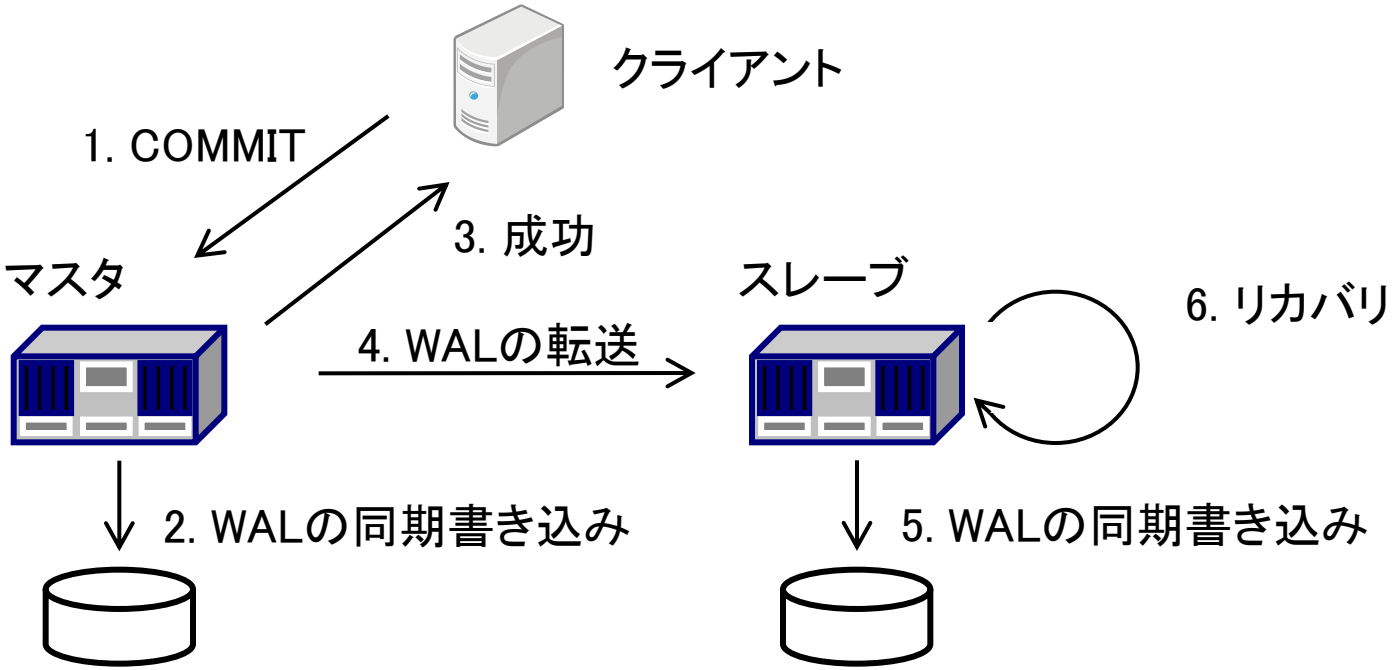
PG本体機能レプリケーション方式

- ログシッピング方式
  - PG用語でストリーミングレプリケーションと呼んでいます
- データベースの変更情報としてWALをマスタからスレーブにレコード単位で送信
- 送信されたWALをリカバリすることで、スレーブはデータベースを複製



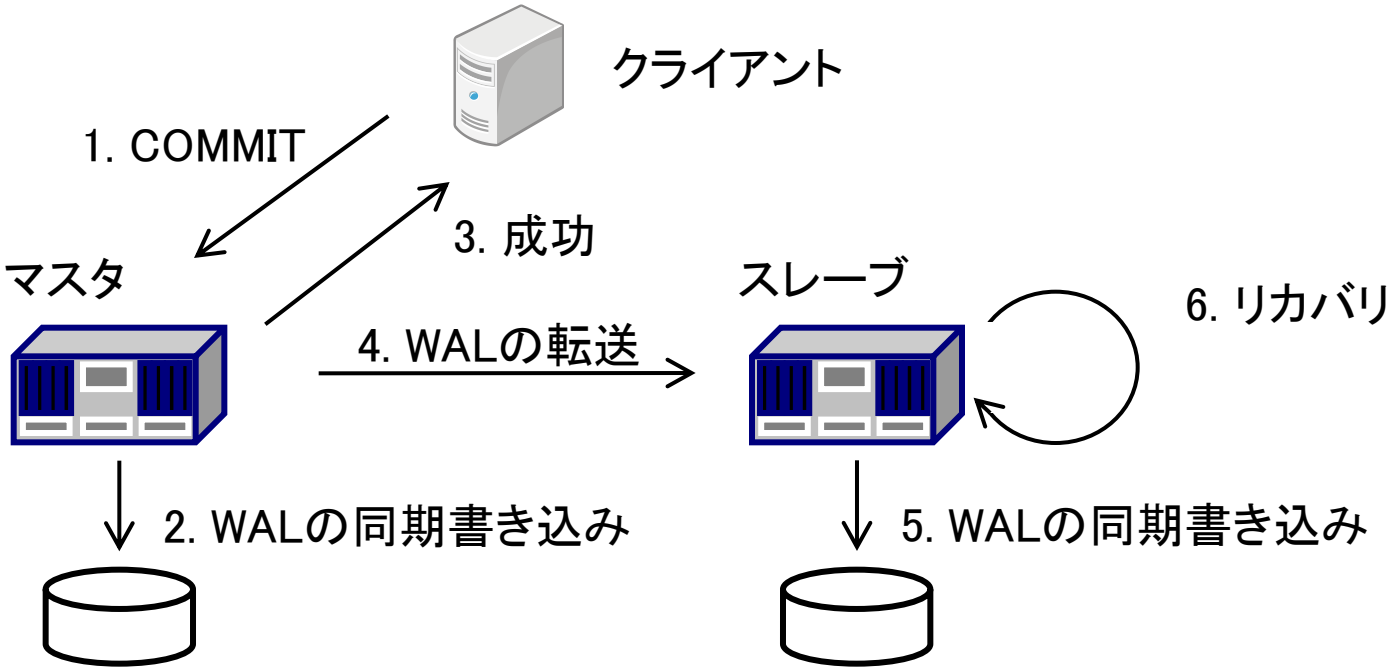
非同期レプリケーション　1/2

- トランザクションは、WALがマスタに書き込まれた時点で完了
  - スレーブへのWALの転送やリカバリは、トランザクションの完了とは非同期的に実行
- レプリケーションの性能オーバーヘッドは小さい
  - トランザクションは、レプリケーションの完了を待つ必要がないため



非同期レプリケーション 2/2

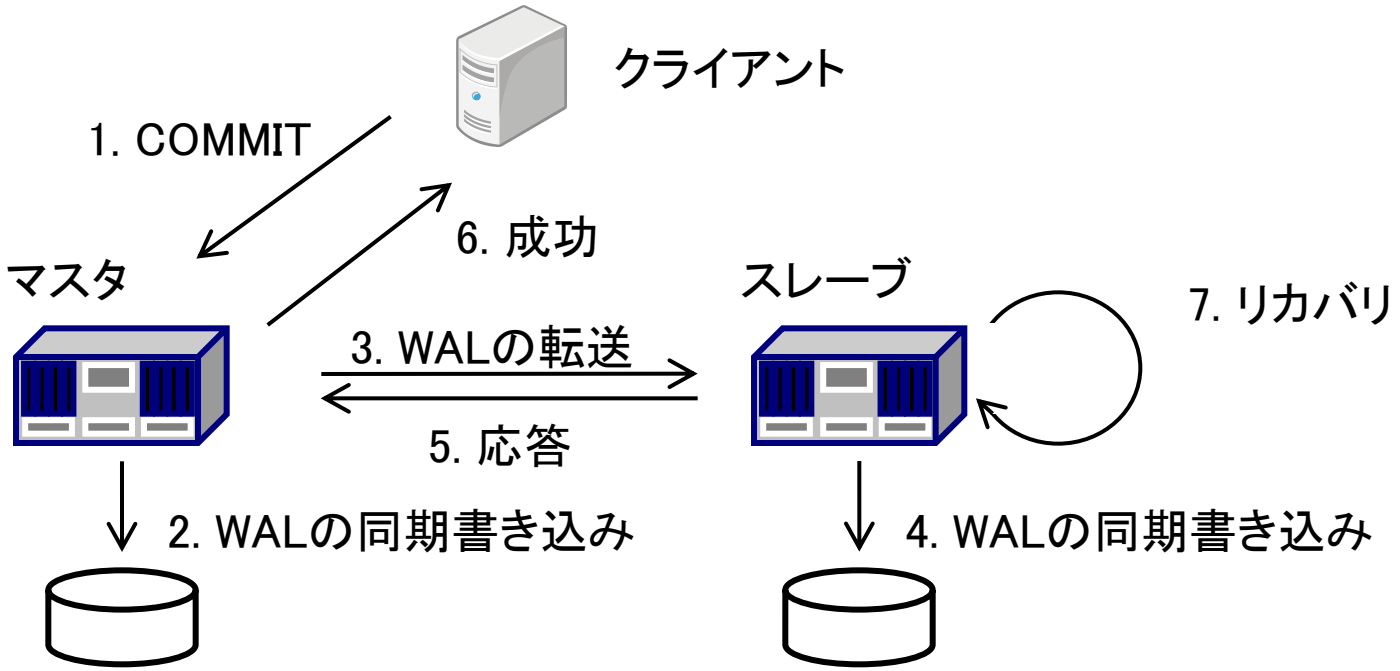
- フェイルオーバ時にCOMMIT済のデータを失う可能性がある
  - COMMIT成功時、そのトランザクションのWALがスレーブに届いている保証はないため
  - ➔ 高可用・データ保護の用途には利用しにくい☹
- スレーブで参照SQLを実行したとき、古いデータが見える可能性がある
  - COMMIT成功時、そのトランザクションのWALがリカバリされている保証はないため



同期レプリケーション 1/2

- 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、同期から選択可能

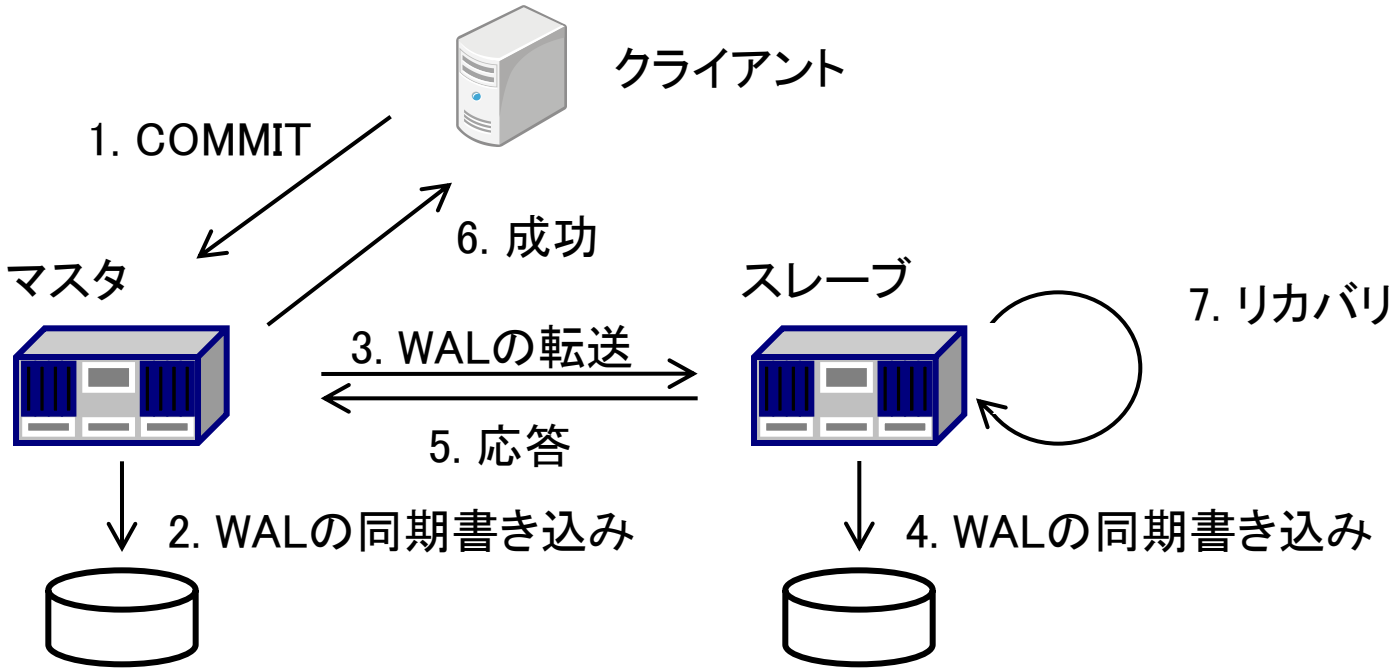
- トランザクションは、WALがマスタとスレーブの両方に書き込まれた時点で完了
  - WAL書き込みを同期させている
  - スレーブへのWAL転送は同期的だが、リカバリは非同期的に実行



同期レプリケーション 2/2

◘ 同期レプリケーションの性能オーバーヘッド

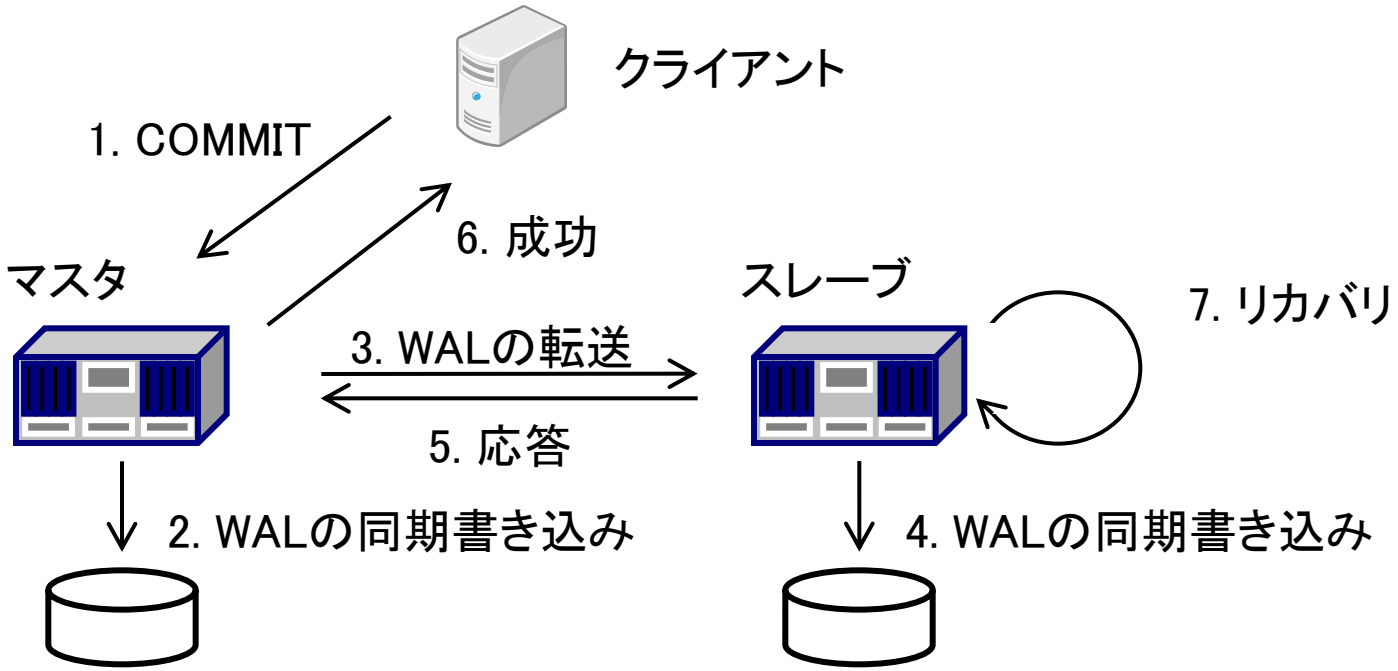
- トランザクションは、WALの転送、スレーブによるWALの同期書き込みを待たなければならない
  - WriteBack機能付きのRAIDカードを推奨
- 両系間のN/W性能やスレーブのディスクI/O性能が低いとオーバーヘッドは増大
- 高頻度でfsyncが走るため、N/Wよりディスク I/Oの方が性能に影響しやすい



同期レプリケーションのデータ保護レベル

- フェイルオーバ時にCOMMIT済のデータが失われることはない
  - COMMIT成功時、そのトランザクションのWALがマスタとスレーブ両方のディスクに存在することが保証されるため
  - COMMIT済のデータが失われるのは、両系のディスクが破壊されたという稀な場合のみ
  - ➔ 高可用・データ保護の用途に利用しやすい

- スレーブで参照SQLを実行したとき、古いデータが見える可能性がある
  - **「同期レプリケーション = COMMIT済のデータがすぐにスレーブで参照できる」ではないことに注意!**

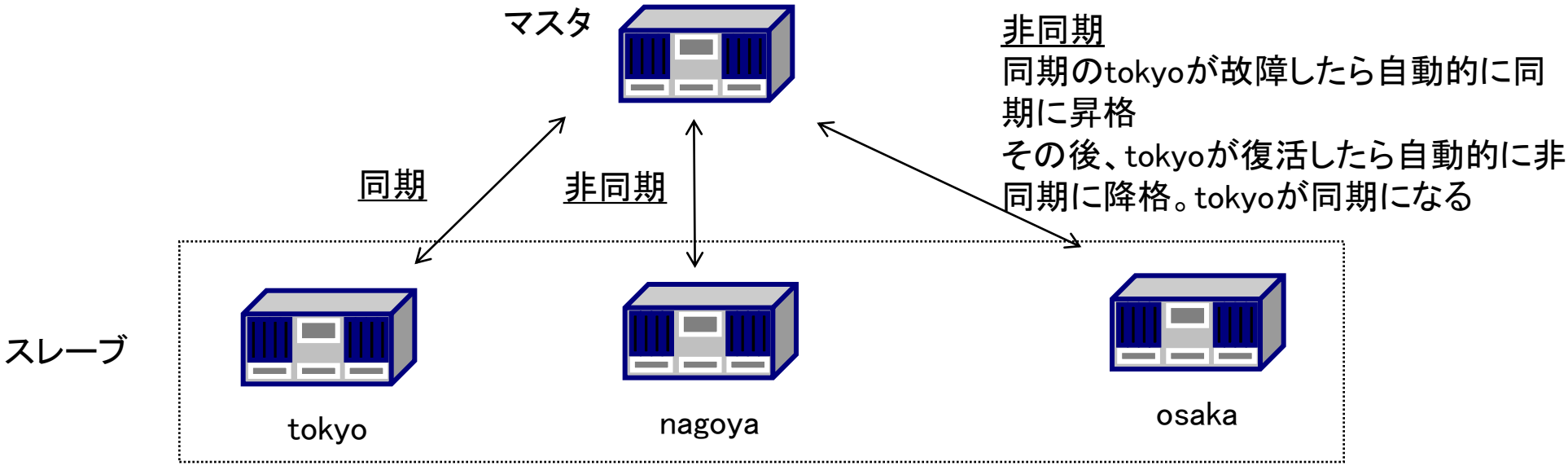


同期レプリケーションの設定

- スレーブ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”’

同期レプリケーションの監視

- pg_stat_replicationビューからスレーブごとにレプリケーション情報を取得可能
  - ビューを確認できるのはマスタのみ。スレーブではこのビューは常に空

```
=# 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    | 1                 ←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   | sync                ←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
```

トランザクション単位のデータ保護レベルの制御

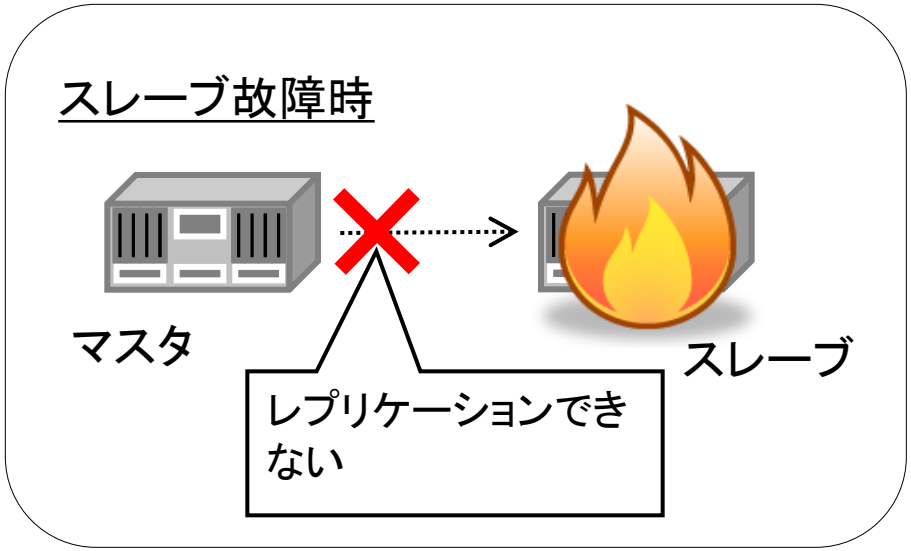
- トランザクションの種類によって、データ保護レベルよりも速度を重視したい or 速度よりもデータ保護レベルを重視したい
- この要望に応えるのがsynchronous_commit
  - SET文を使って、トランザクションごとにデータ保護レベルを設定可能

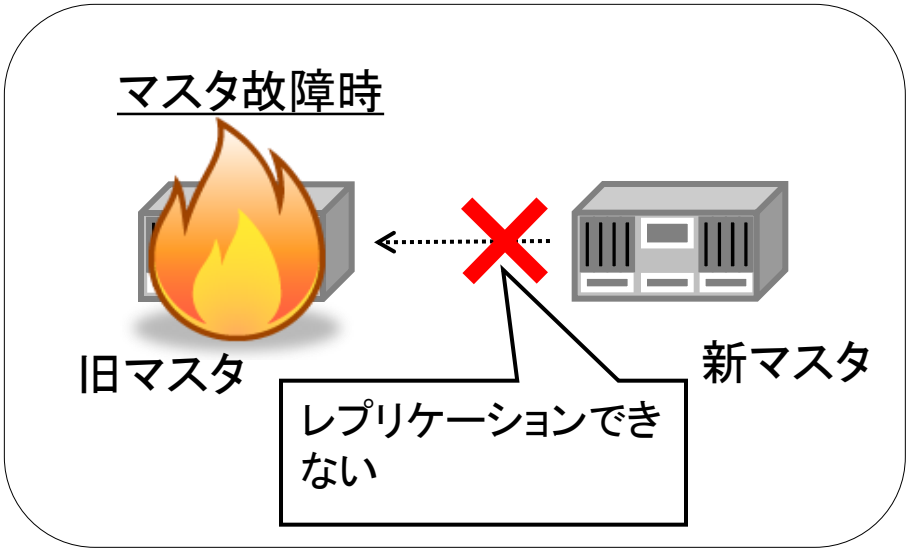
スレーブのモード	synchronous_commitの設定値	トランザクションがWAL書き込みを待つか?	
		マスタ	スレーブ
同期	on	待つ	ディスク書き込みまで待つ
	remote_write	待つ	メモリ書き込みまで待つ
	local	待つ	
	off		
非同期	on	待つ	
	local	待つ	
	off		

レプリケーションを継続できないときの挙動

- レプリケーションを継続できないのは、マスタとスレーブの2台構成で、以下の場合
  - マスタ or スレーブの故障時
  - マスタ/スレーブ間のN/W故障時

- 非同期レプリケーション
  - マスタは、単独でトランザクションを処理
  - スレーブ故障やネットワーク故障によってトランザクションが停止することはない





レプリケーションを継続できないときの挙動

- レプリケーション 故障の場合
  - マスタは、スレーブからの届くことのないレプリケーション完了の応答を待ち続ける
  - **故障から復旧するまでマスタのトランザクションが停止する！！**
    - **replication_timeout の設定値を超えても停止します！！**

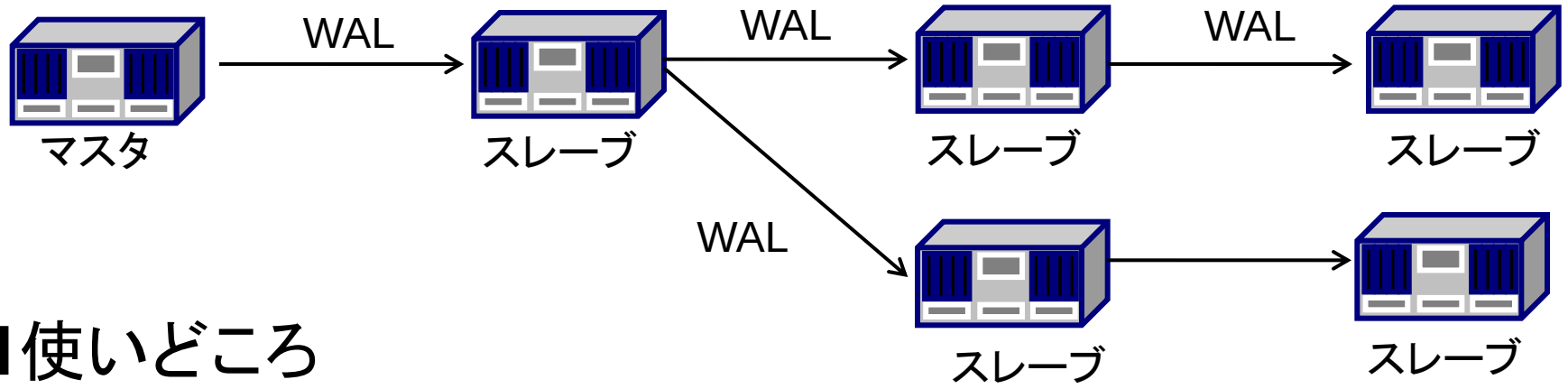
- トランザクションを再開させるには？
  - スレーブを故障から復旧させ、再びレプリケーションを開始
  - synchronous_standby_names から、故障した Slave を削除し、pg_ctl reload

- **データ保護を最優先**するための措置
  - マスタ単独でのトランザクション処理を許すと、そのトランザクションはマスタのディスクにしか記録されていないため、マスタのディスク故障時にCOMMIT済のデータが消える
  - トランザクションを停止することで、マスタにだけ存在するようなCOMMIT済データが発生するのを回避

- 故障時にトランザクションを停止させたくない場合
  - 同期レプリケーションのスレーブを複数台用意
    - 別のスレーブが「同期」に自動的に昇格し、そのスレーブにレプリケーションできるとトランザクション再開
  - Pacemaker と合わせて使う
    - RA が自動的に制御します。詳細は後半で紹介します！

カスケードレプリケーション

ロスレーブから更に非同期レプリケーションが可能

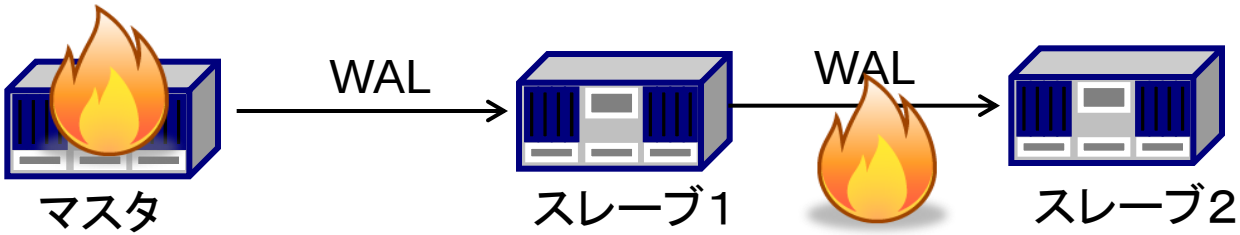


ロ使いどころ

- DR構成時の遠隔地でのレプリケーション
- マスタの負荷軽減
- 参照負荷分散のレプリケーション構成の更なるスケールアウト
  - 通常だと 1マスタ 10スレーブ 辺りが限界
    - ネットワーク I/O がネックになるため
- カスケードレプリケーションを用いたHA構成はRAが未対応

カスケード構成の注意

- Masterが故障し、フェイルオーバした際に、カスケード先のレプリケーションも切断される



- 原因
  - Promoteしたスレーブ1のTimeLineIDが上がるため
    - TimeLineIDが揃っていないとレプリケーションはできない
- 対策
  - 故障後はスレーブ1から手動でベースバックアップを取得し、再組込みする必要あり

まとめ

- 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」と「同期」から選択可能
  - WALの同期を保証し、参照可能データの同期は保証しない
- 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可能
  - [9.1～] synchronous_commit = local が追加
  - [9.2～] synchronous_commit = remote_write が追加
- レプリケーションの様子は pg_stat_replication で確認できる
- 同期レプリケーションはデータ保護を最優先し、レプリケーションを継続できないときはトランザクションを停止させる
  - 高可用構成の場合は、Pacemakerと組み合わせて使用することを推奨

Pacemakerを用いた
レプリケーション構成の高可用化

NTT OSSセンタ
松尾 隆利

レプリケーション対応機能開発経緯

～ 2009年

- PG-REX というプロジェクトが存在
  - PostgreSQL 8.3 + 独自パッチで同期レプリケーションを実装
  - HeartbeatでHAクラスタ化

2010年

- PostgreSQL9.0 + Pacemakerで開発
  - 同期レプリケーション実現の独自パッチと、HAクラスタ化用独自パッチあり

    → 制御スクリプト(pgsql RA)をコミュニティへ投稿するもリジェクト

2011年

- PostgreSQL9.1 + Pacemakerで開発（社外からも応援あり）
  - RAほぼ作り直し　→ コミュニティ絶賛！(tremendous job !!)

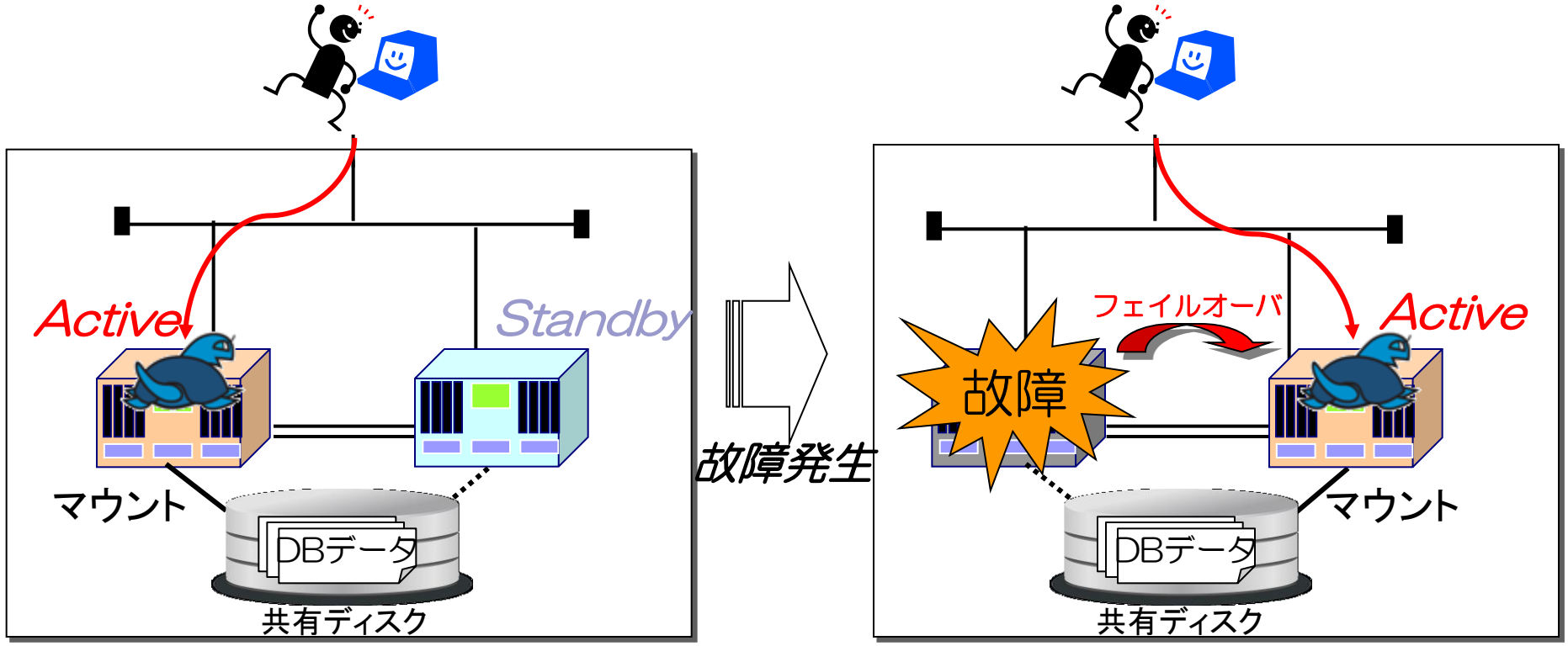
2012年

- 4月13日　コミュニティのリポジトリにマージ！
- 5月16日　resource-agents 3.9.3 としてリリース

  Pacemakerのコンポーネント名

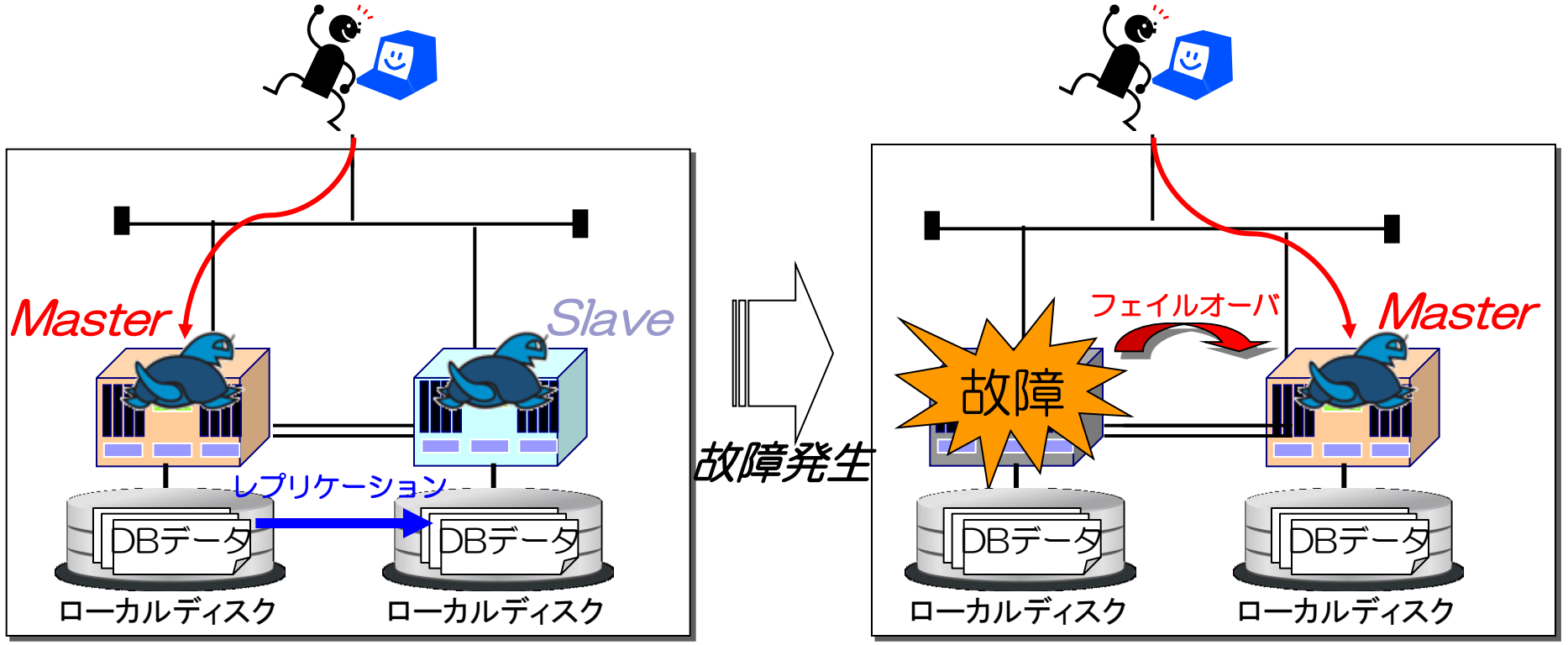
HAクラスタの今までの構成　～Active/Standby 構成～

- PostgreSQLのデータは共有ディスク上に配置し、2台のサーバ間で共有
- 通常はActiveサーバでサービスを提供し、Activeサーバ故障時はStandbyサーバがActiveとなりサービスを提供（フェイルオーバ）



レプリケーション構成　～Master/Slave 構成～

- PostgreSQLのデータはローカルディスク上に配置し、PostgreSQLのストリーミングレプリケーション機能を用いて共有
- 通常はMasterサーバでサービスを提供し、Masterサーバ故障時はSlaveサーバがMasterに昇格しサービスを継続

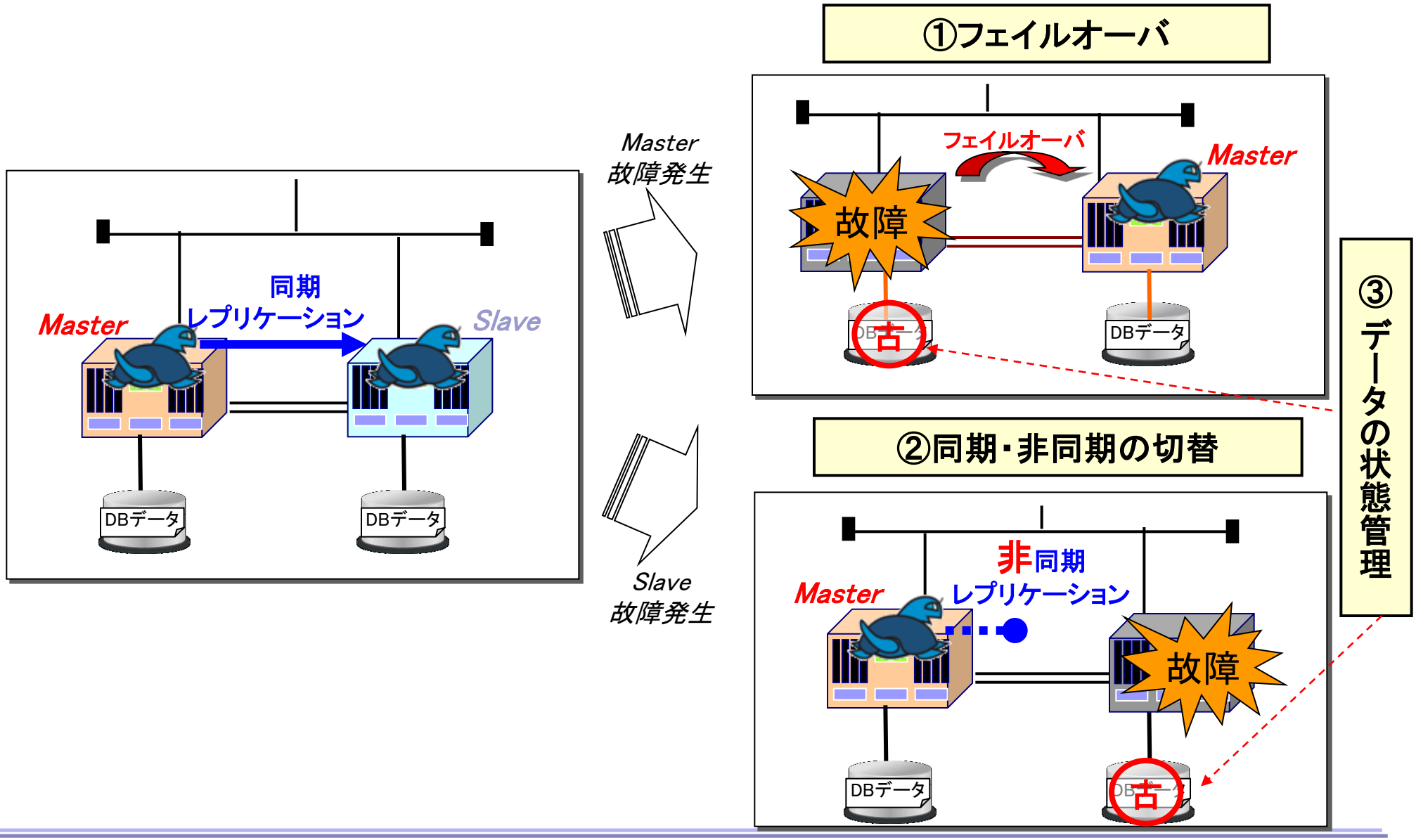


Active/Standby vs Master/Slave

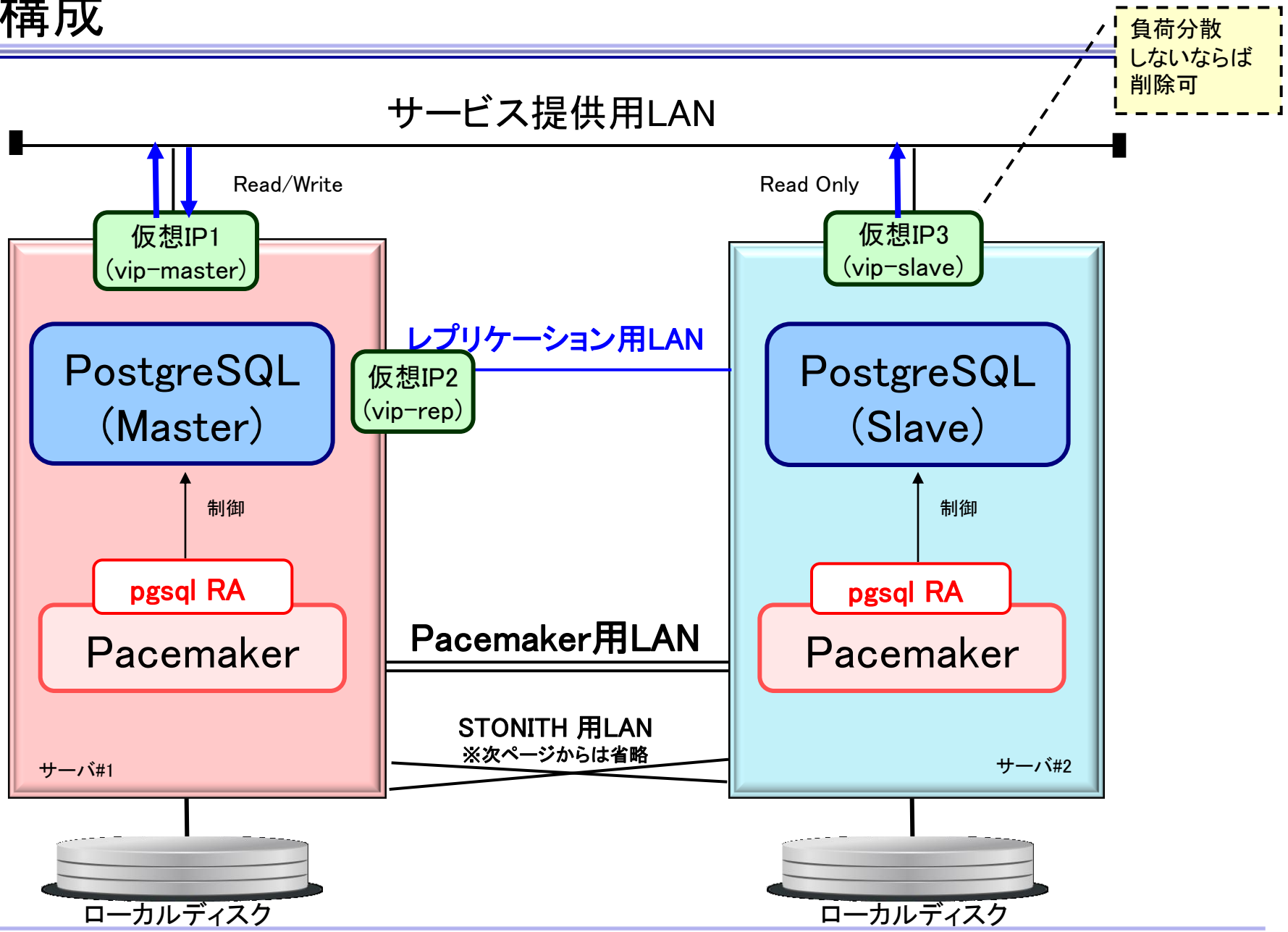
	Active/Standby	Master/Slave
ハードウェア費用	共有ディスク(相当のもの)必須	
運用のしやすさ	データは1箇所のみ	2箇所のデータの整合性を考慮
データの安全性	最新データは 共有ディスク上のみ	最新データは2箇所に分散
サービス継続性	フェイルオーバ時に リカバリに時間を要する	Slave故障がサービスに影響 (同期レプリケーション使用時)
DB性能	レプリケーションの オーバヘッドなし	共有ディスク構成をとれない 高速ストレージを活用可
負荷分散	構成上不可能	ReadOnlyクエリを Slaveで処理可能
実績		これから・・・・

それぞれ一長一短。サービスの要件に応じて選択すること。

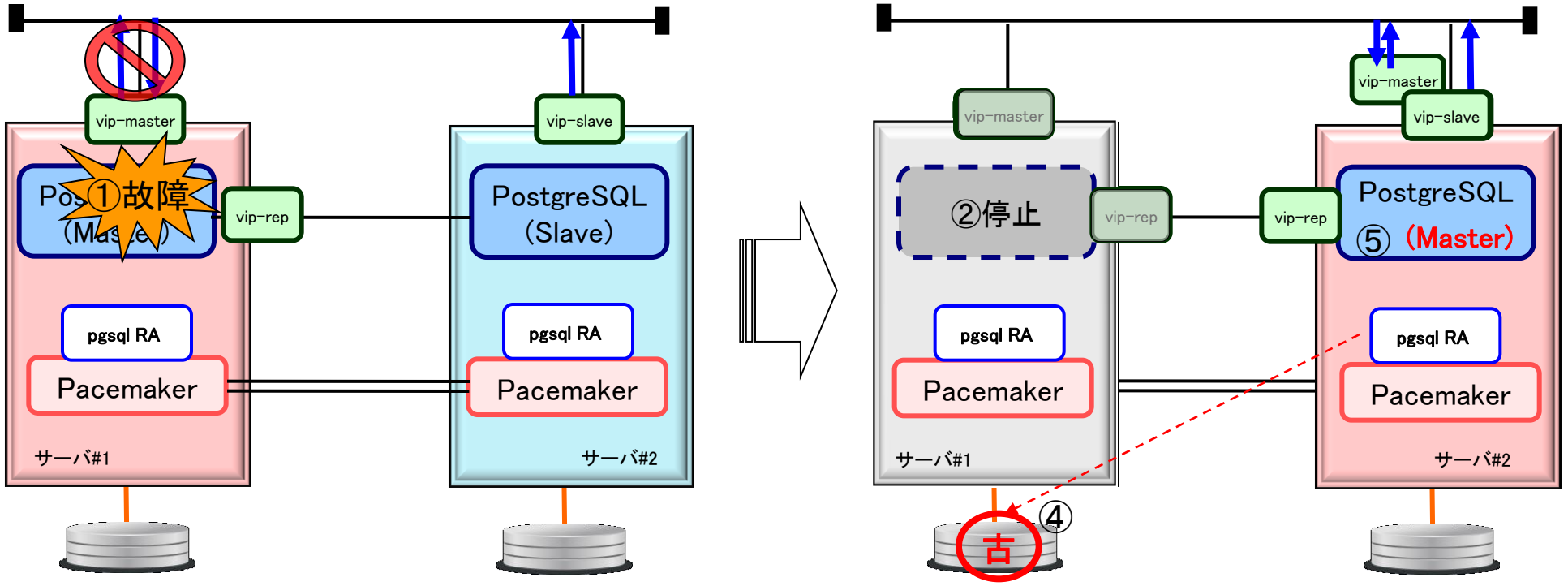
レプリケーション構成のHAクラスタ化 3大機能



基本構成



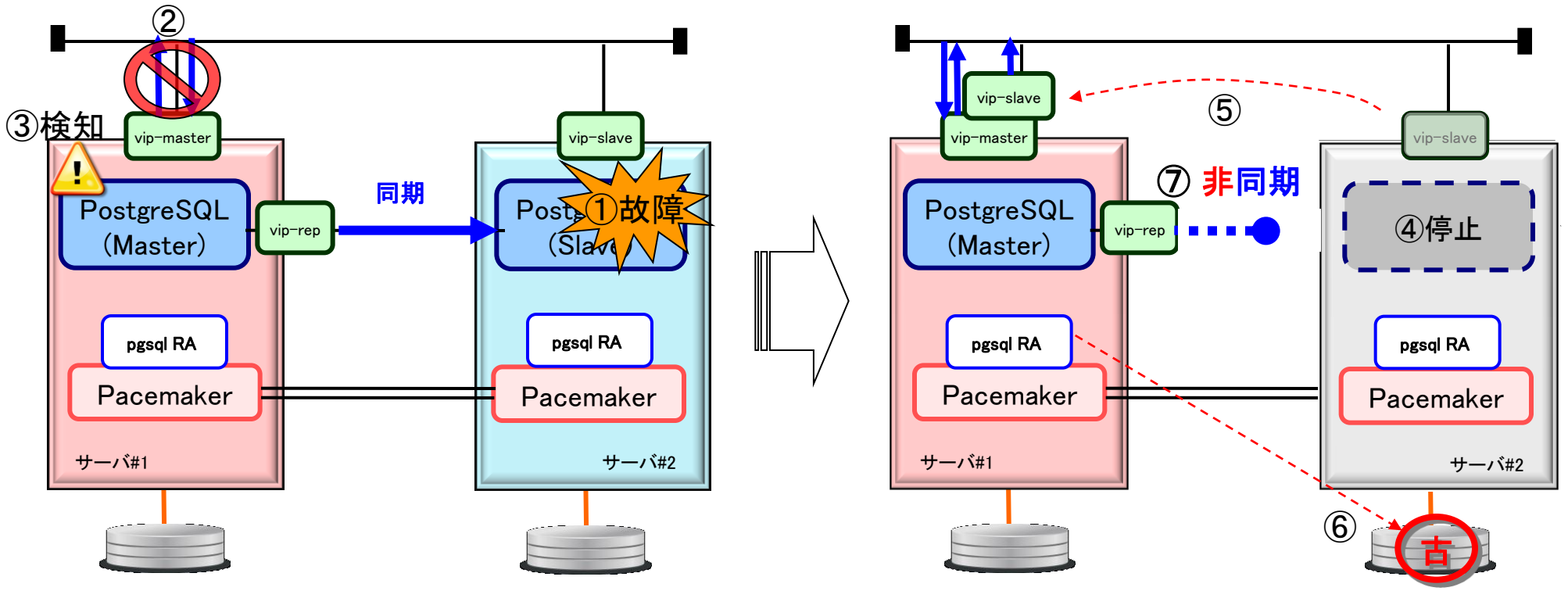
基本動作1：Masterのフェイルオーバ



① #1のPostgreSQLの故障を検知

② #1のPostgreSQLを停止
③ 仮想IP(vip-master, vip-rep, vip-slave)を停止
④ #1のデータが古いことを記録
⑤ #2のPostgreSQLをMasterに昇格(promote)
⑥ #2で仮想IP(vip-master, vip-rep, vip-slave)を起動

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
```

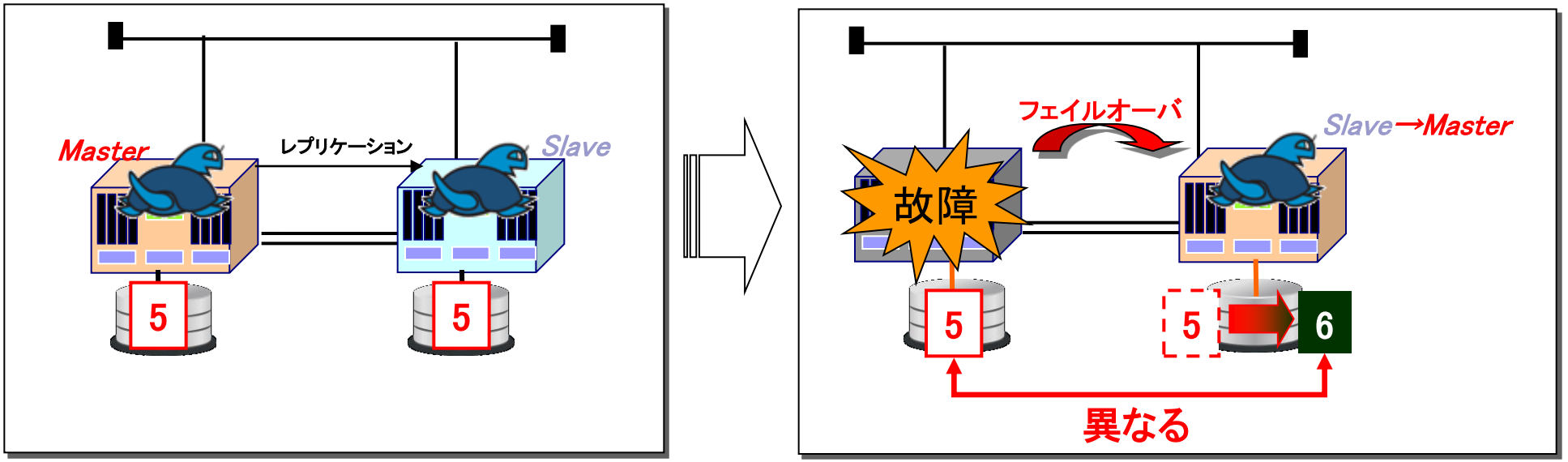
④ #2のPostgreSQLを停止
⑤ #2の仮想IP(vip-slave)を#1に付け替え
⑥ #2のデータが古いことを記録
⑦ #1のPostgreSQLを非同期設定に変更
→ トランザクション再開

ここまでが基本動作
次はフェイルオーバ後の復旧

TimelineID

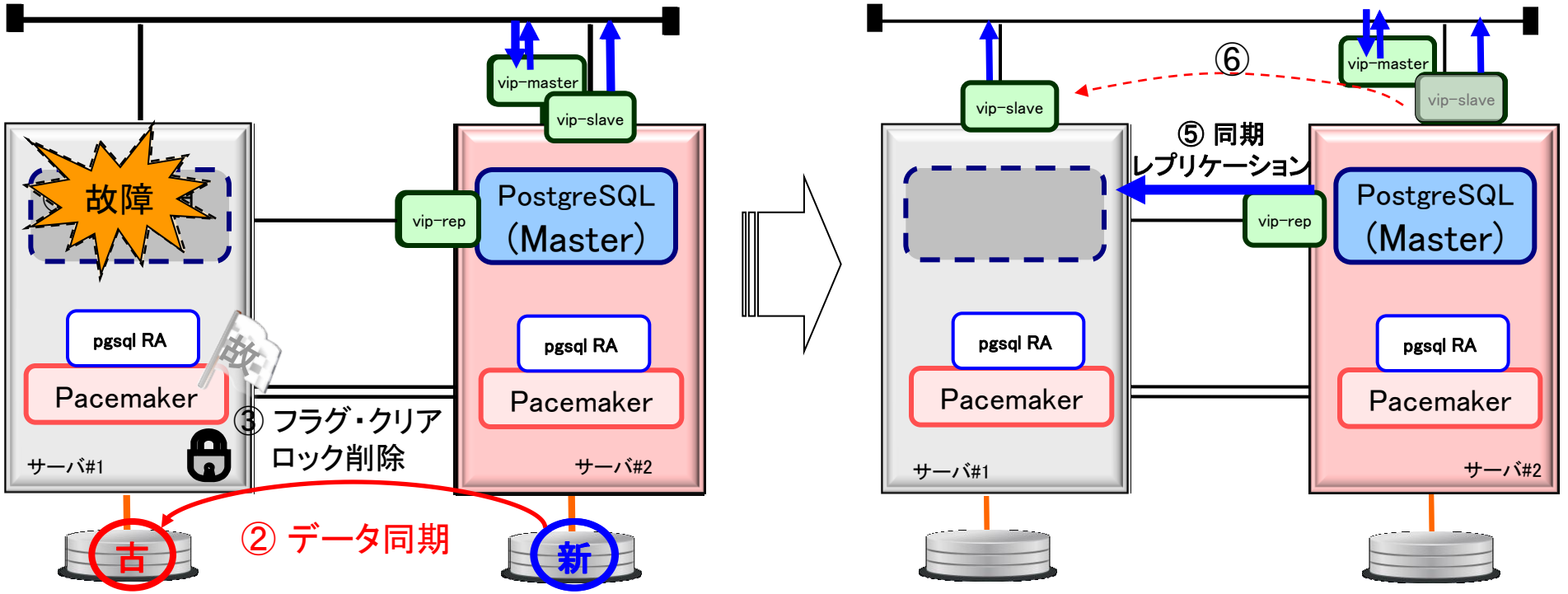
- PostgreSQLがSlaveからMasterへ昇格した際インクリメントされる数値
- TimelineIDが異なるとレプリケーション接続ができない

フェイルオーバ時



TimelineIDをそろえるには
新Masterのデータを旧Masterへコピーする必要あり

運用1: フェイルオーバ後の復旧



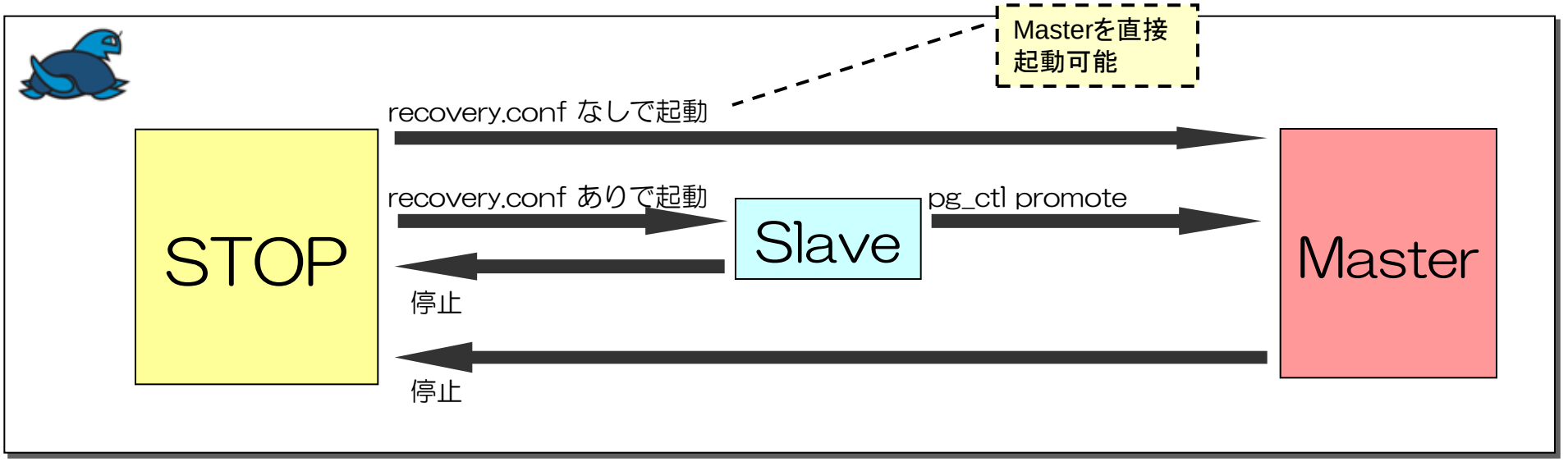
① 故障の復旧
② #1のデータを#2と同期
　→ TimelineIDがそろう
③ #1のロックファイル削除と
　Pacemaker上の故障フラグを
　クリア

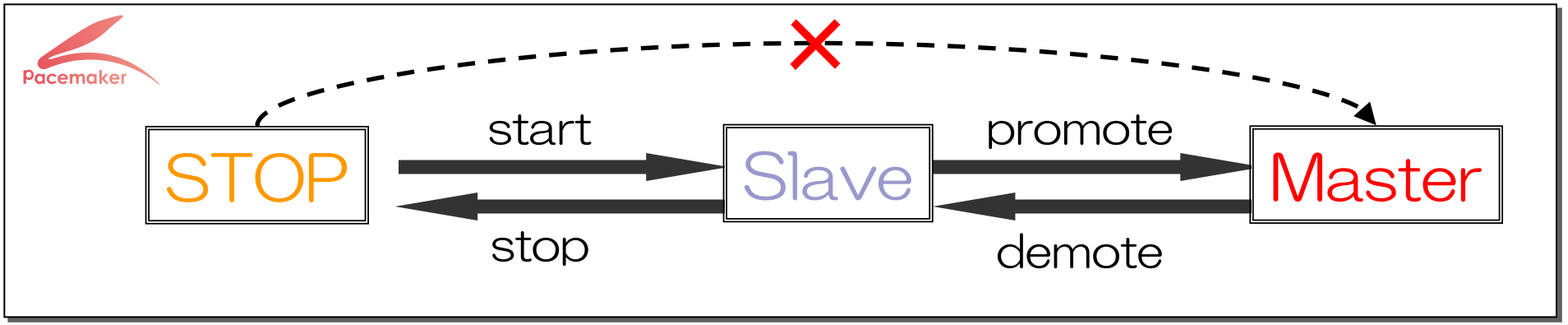
（手動）

④ #1のPostgreSQLをSlaveで起動
⑤ レプリケーション開始
　→ 非同期で接続→同期設定に切替
⑥ #2の仮想IP(vip-slave)を#1に付け替え

次は起動方法

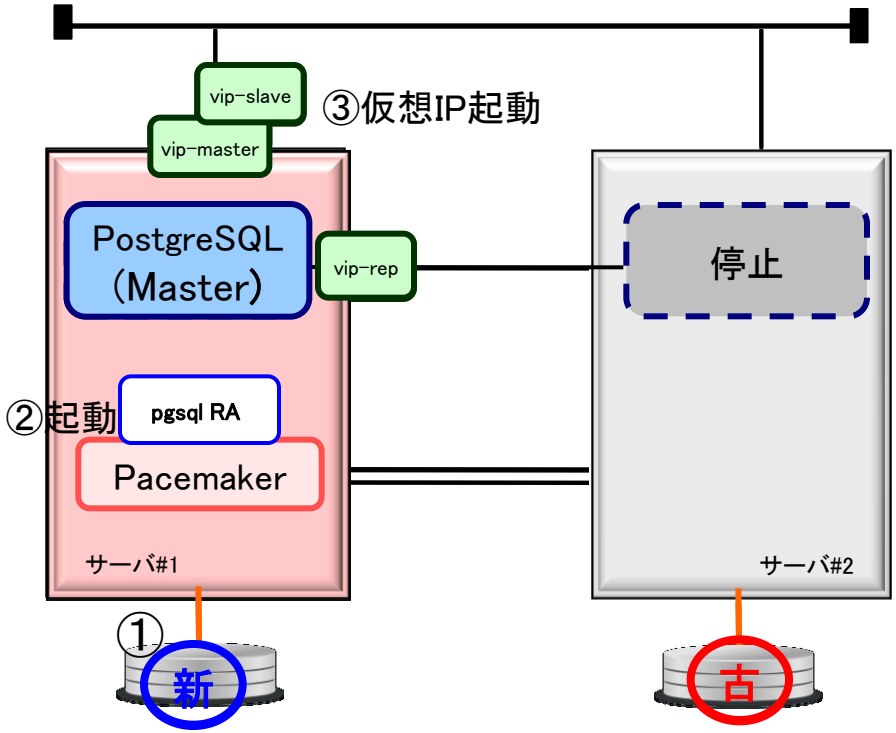
PostgreSQLとPacemakerの状態遷移

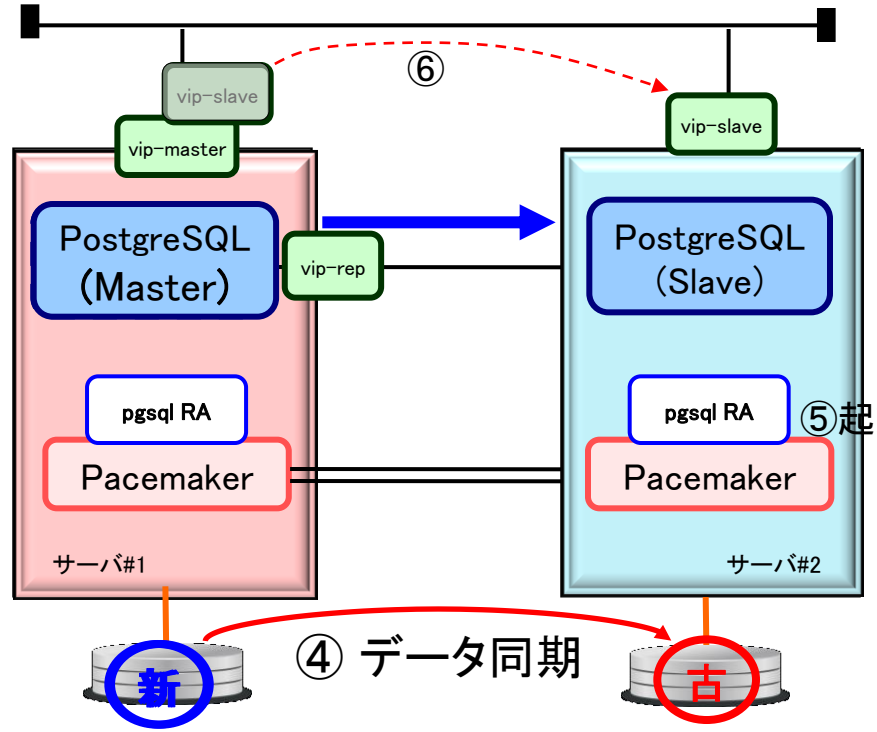




PostgreSQLのMasterは必ずSlaveを経由して起動される
→ Master起動時もTimelineIDがインクリメントされる

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
→ TimelineIDがずれる
③ 仮想IPが#1で起動

（手動）

④ #2のデータを#1と同期
→ TimelineIDがそろう
⑤ Pacemaker起動
→ レプリケーション開始
⑥ #1の仮想IP(vip-slave) を#2に付け替え

（手動）

高可用化まとめ

□3大機能

- Masterのフェイルオーバ
- レプリケーションの同期・非同期の切替
- データの状態管理

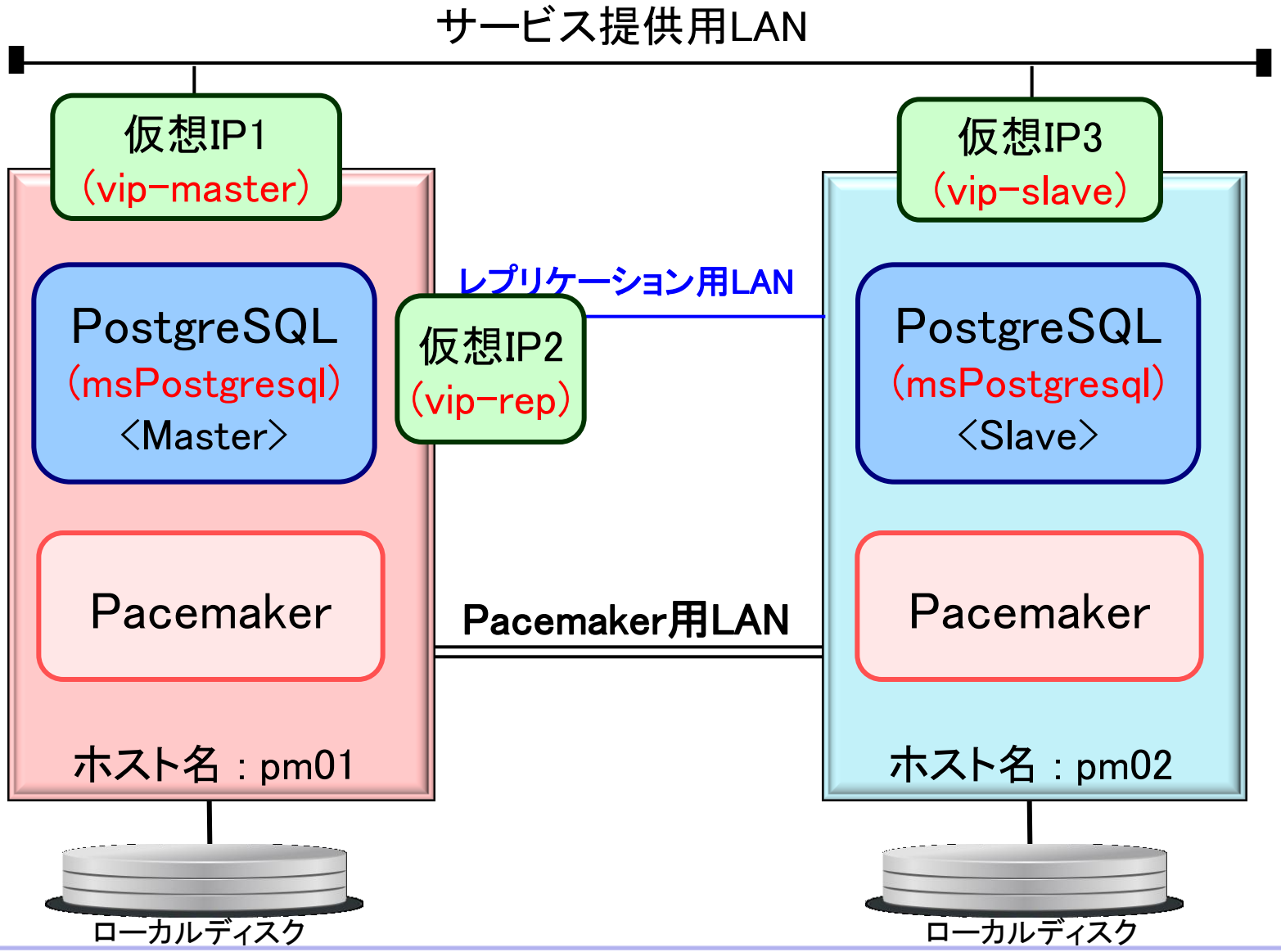
□運用時の注意

- TimelineIDがずれているとレプリケーションできないため注意
  - 難しくてよくわからない場合は、
    『Slave 起動前に Master データのコピーが必要』と覚えておけばOK

    ※ 単純にTimelineID を合わせるためには、WALアーカイブのみをコピーすれば可能だが、
    フェイルオーバするとサーバ間のデータの整合性が崩れる可能性があり、これを避けるためにも
    全データのコピーを推奨

デモ

デモ環境



Pacemaker状態表示例(crm_mon -Af 実行時)

```
Online: [ pm01 pm02 ]

vip-slave   (ocf::heartbeat:IPaddr2):       Started pm02
 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):   Started pm01
     vip-rep    (ocf::heartbeat:IPaddr2):   Started pm01
 Master/Slave Set: msPostgresql
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
 Clone Set: clnPingd
     Started: [ pm01 pm02 ]

Node Attributes:
* Node pm01:
    + default_ping_set                  : 100
    + master-pgsql:0                    : 1000
    + pgsql-data-status                 : LATEST
    + pgsql-status                      : PRI
    + pgsql-master-baseline             : 00000000150000B0
    + pm02-eth1                         : up
    + pm02-eth2                         : up
* Node pm02:
    + default_ping_set                  : 100
    + master-pgsql:1                    : 100
    + pgsql-data-status                 : STREAMING|SYNC
    + pgsql-status                      : HS:sync
    + pm01-eth1                         : up
    + pm01-eth2                         : up

Migration summary:
* Node pm01:
* Node pm02:
```

仮想IPの状態

PostgreSQLのMaster/Slaveの状態

pm01ノードのPostgreSQLとデータの状態

promote直前のxlogの位置

pm02ノードのPostgreSQLとデータの状態

Pacemaker状態表示 省略後(今回のデモ用表示)

```
vip-slave          (ocf::heartbeat:IPaddr2):   Started pm02
 Resource Group: master-group
     vip-master (ocf::heartbeat:IPaddr2):    Started pm01
     vip-rep    (ocf::heartbeat:IPaddr2):    Started pm01
 Master/Slave Set: msPostgresql
     Masters: [ pm01 ]
     Slaves: [ pm02 ]
--------------------------------------------------------------------
* Node pm01:
    + pgsql-data-status                 : LATEST
    + pgsql-status                      : PRI
* Node pm02:
    + pgsql-data-status                 : STREAMING|SYNC
    + pgsql-status                      : HS:sync
--------------------------------------------------------------------
* Node pm01:
* Node pm02:
```

仮想IPの状態

PostgreSQLのMaster/Slaveの状態

pm01ノードのPostgreSQLとデータの状態

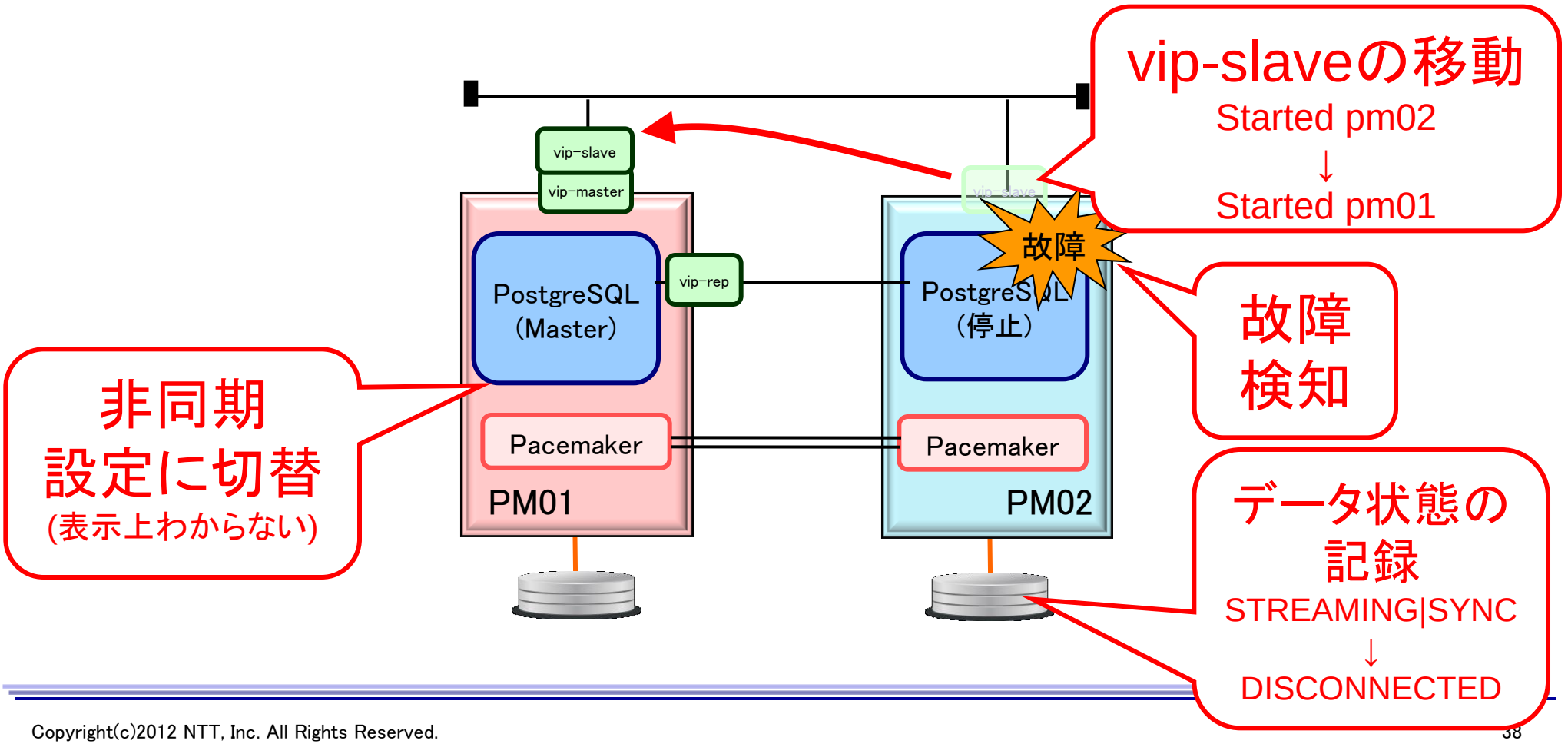
pm02ノードのPostgreSQLとデータの状態

故障状態が表示される

【デモ】 Slaveの故障

□pm02のPostgreSQLのプロセスをkill

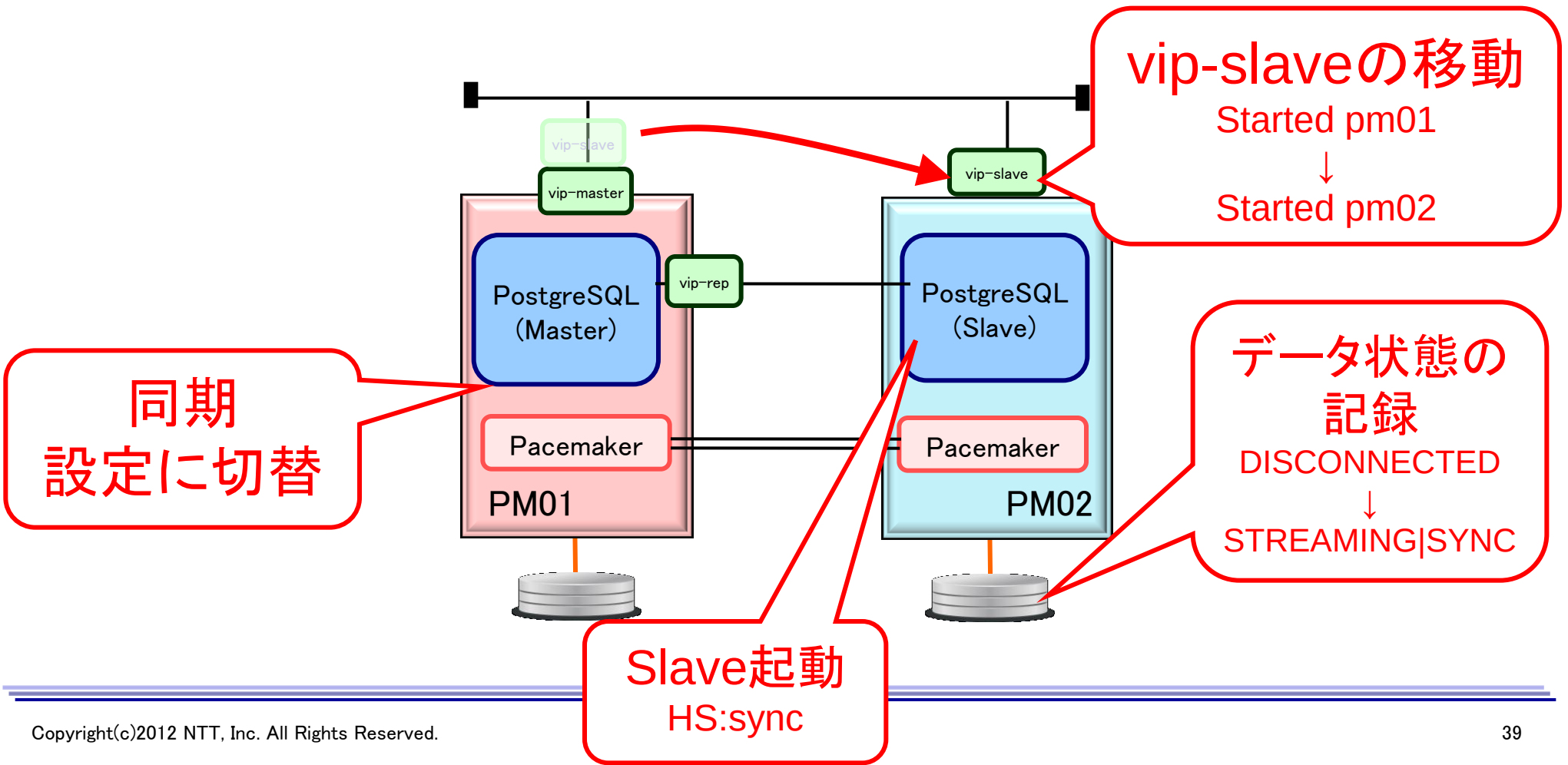
```
# killall -9 postgres
```



【デモ】 Slaveの復旧

□pm02のフェイルカウントクリア

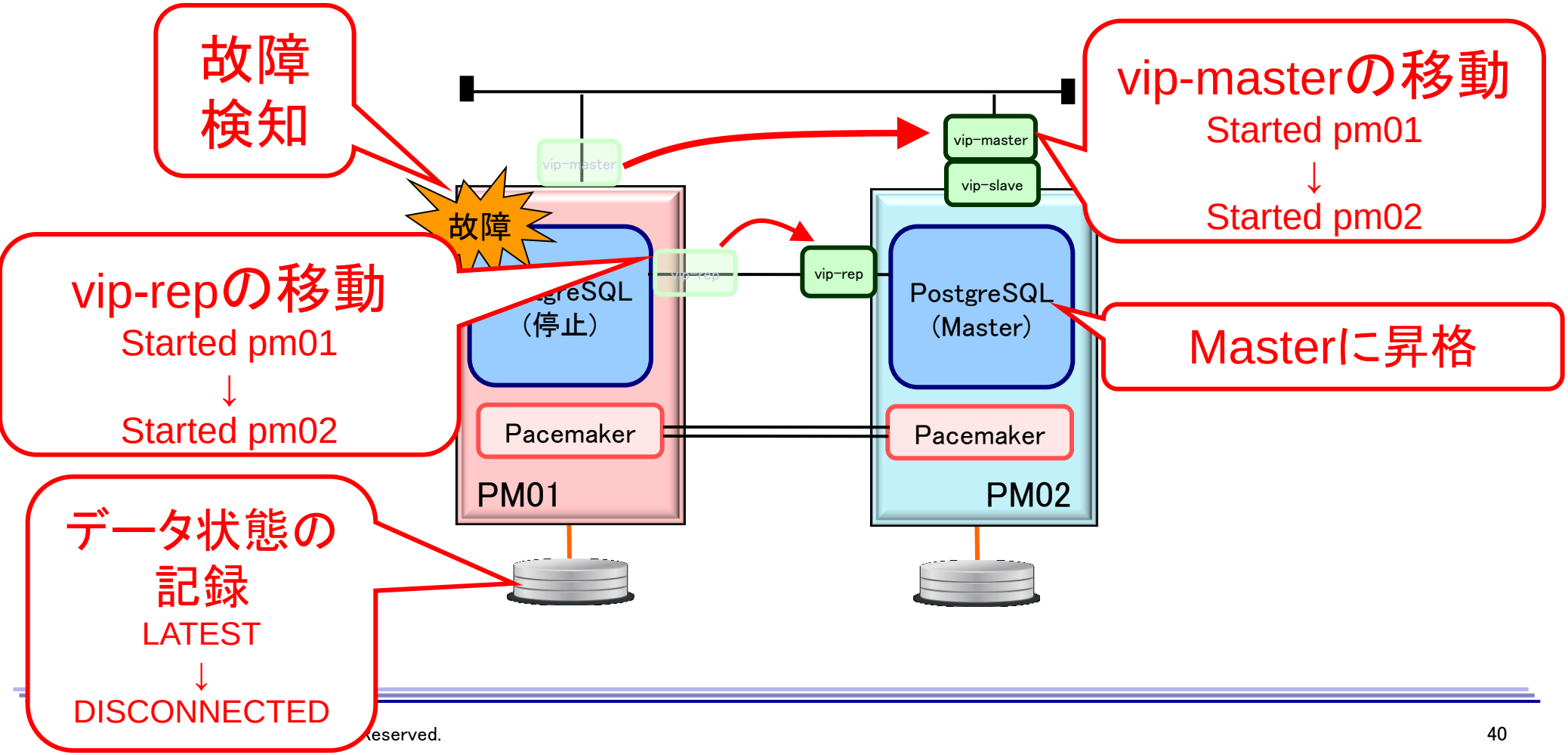
```
# crm resource cleanup msPostgresql pm02
```



【デモ】 Masterの故障

□pm01のPostgreSQLのプロセスをkill

```
# killall -9 postgres
```



PostgreSQL制御スクリプト（pgsql RA）の最近の動き

resource-agent 3.9.4（11/23リリース）での変更内容

- Pacemaker 1.1.x の仕様変更に追従
  - Pacemaker 1.0.x との互換は保持
- recovery.confの、archive_cleanup_commandや recovery_end_commandを設定可能に
- promote時にPostgreSQLをpromoteするのではなく、recovery.confを削除して再起動させることでMaster化可能に
  - Timeline ID のインクリメント防止可能に
    - ただしexperimental 機能
- その他細かなバグ修正

動作環境

- Pacemaker 1.0.12 以上推奨
- resource-agents 3.9.3 以上必須
  - Linux-HA Japan Pacemakerリポジトリパッケージ 1.0.12-1.2 以上に同梱 (2012年7月リリース)

- PostgreSQL 9.1 以上
  - 9.0では動きません

参考

ソースコード (GitHub上のRA直リンク)

- URL : https://github.com/ClusterLabs/resource-agents/blob/master/heartbeat/pgsql

ドキュメントおよび設定例 (GitHubのWiki)

- https://github.com/t-matsuo/resource-agents/wiki/

Pacemakerダウンロード・インストール

- http://linux-ha.sourceforge.jp/

PG-REX(PostgreSQL + Pacemaker)利用マニュアル

- http://sourceforge.jp/projects/pg-rex/releases/55691