

バックアップとPITR

日本PostgreSQLユーザ会北海道支部
株式会社サイクル・オブ・フィフス
石田朗雄

バックアップの方法

- pg_dump
 - SQLレベルのオンライン・フルバックアップ
- PITR(Point In Time Recovery)
 - トランザクションログを使った差分バックアップ
- コールドバックアップ
- レプリケーション

pg_dump

- 標準のオンラインバックアップ(libpqアプリケーション)
- 基本的に、SQL形式で出力する
 - PostgreSQLのメジャーバージョンアップや、他のRDBMSへの移行にも
- pg_dumpには大量のオプションがある(version8.2では29個くらい)。⇒とても柔軟なツール
 - 接続に関するオプション
 - 出力形式に関するオプション
 - 出力対象に関するオプション
 - 等々

pg_dumpからのリストア

- プレインテキスト形式のダンプであれば、psqlに与えるだけ
- 通常は、PostgreSQLのスーパーユーザ権限で実行する

例1)

```
$ pg_dump -f db1.dump db1
```

別のマシンにリストアする場合

この時点ではdb1が存在しない

```
$ createdb -T template0 db1
```

```
$ psql -f db.dump db1
```

例2)

```
$ pg_dump --create -f db1.dump db1
```

CREATE DATABASEも出力してくれる

```
$ psql -f db.dump template1
```

pg_dump 練習1

- 1つのデータベースをダンプする

```
$ pg_dump db1
```

- 1つのテーブルだけダンプする

```
$ pg_dump -t accounts db1  
$ pg_dump --table=accounts db1
```

- 1つのテーブルのスキーマ(定義)のみダンプする

```
$ pg_dump -s -t accounts db1  
$ pg_dump --schema-only --table=accounts db1
```

pg_dump 練習2

- COPY形式ではなくINSERT形式で

```
$ pg_dump -t branches --inserts db1  
$ pg_dump -t branches --column-inserts db1
```

- グローバルオブジェクト(ROLE、TABLESPACE)のみダンプする

```
$ pg_dumpall -g
```

カスタム形式のpg_dump

- pg_dumpの出力するファイル形式
 - -Fp プレインテキスト形式(デフォルト)
 - -Ft tar形式
 - -Fc カスタム形式
- tar形式、カスタム形式のダンプをリストアする時は、pg_restoreコマンドを使う
 - pg_dumpではなくpg_restoreで、出力範囲に関する各種オプションを指定する
 - ダンプファイルに含まれるオブジェクト一覧を出力、編集した一覧からのダンプ

pg_dump 練習3

- 1つのテーブルのスキーマのみダンプする(カスタム形式版)

```
$ pg_dump -Fc -f db1.dump db1  
$ pg_restore -t branches --schema-only db1.dump
```

- ダンプファイルからリストを出力する

```
$ pg_restore -l db1.dump > db1.list  
db1.listを編集  
$ pg_restore -L db1.list db1.dump
```

その他注意点

- バージョンアップに使う場合は、新しいバージョンの pg_dump を使う
- pg_dump でバックアップされない物に注意
 - postgresql.conf や pg_hba.conf
 - 追加した拡張モジュール(\$PGDATA/lib)
 - テーブルスペース

PITR(Point In Time Recovery)

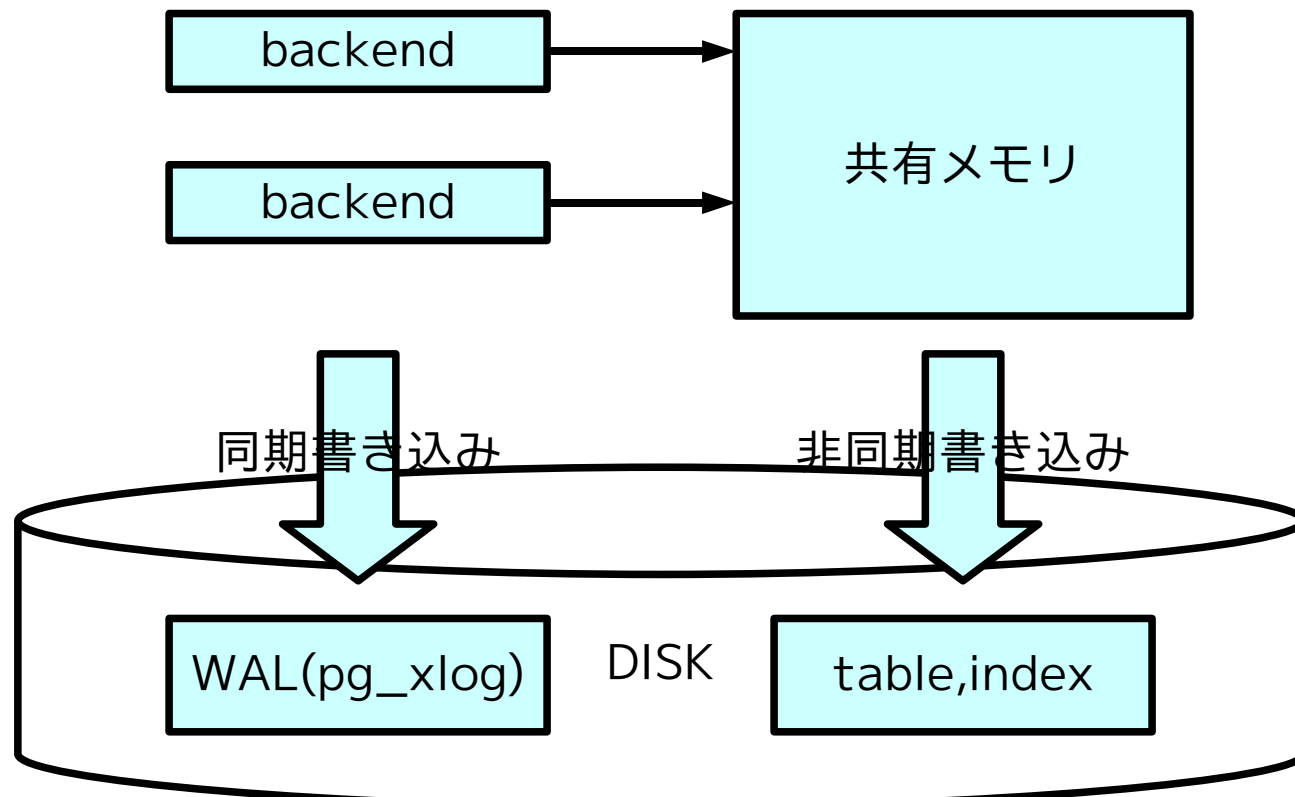
- 非常に大きなデータベースでは、pg_dumpに時間がかかりすぎる
- リストアにも時間がかかりすぎる
- ベースとなるフルバックアップと、それに対する差分バックアップ
- 7.1 WAL(Write Ahead Log)
- 8.0 PITR(Point Time In Recovery)
- 8.2 WARM STANDBY

WAL(Write Ahead Log)

- データ更新時、直接ファイルに同期書き込みすると遅くなる
 - ファイルではなく共有メモリに書く
- しかしそれだけだと、PostgreSQLが停止してしまうとデータが失われてしまう
 - どのファイルにどのような変更を行なったかをログに同期書き込みする
- どのみち同期書き込みしてるから一緒じゃないの？
 - あちこちのファイルにランダムアクセスするより、一つのファイルに追記した方が速い
- 実体はpg_xlogにあるバイナリファイル。トランザクションログとも呼ばれる

ファイルへの書き込み

- WALには同期書き込み(commitのタイミング等)
- テーブルには非同期書き込み(checkpoint、background writer等)



PITR(Point In Time Recovery)

- ある時点でのフルバックアップと、そこ以降のWALファイルがあれば、差分バックアップを実現できる
 - WALの増加がディスクを使いすぎないように、WALはローテーションする(古いものは捨てられる)
 - 満杯になったWALを別のところへコピーしておく

PITRのための準備

- archive_command(postgresql.conf)
 - pg_xlogの下の満杯になったWALをコピーするためのコマンド
 - 失敗ステータスを返した場合はリトライする
 - %p=ログの絶対パス、%f=ログのファイル名

```
archive_command = 'cp %p /path/to/archive_dir/%f'
```

```
$ ps ax | grep post
...
 953  ??  Ss      0:00.00 postmaster: archiver process      (postgres)
```

アーカイブされるタイミング

- WALが満杯になった時にアーカイブ
 - 満杯になる前のWALはリカバリできない
 - 直近までリカバリするにはpg_xlogをどうにかして守る
- 8.2からは
 - archive_timeout(postgresql.conf)
 - pg_switch_xlog()

PITRのためのバックアップ

1. `select pg_start_backup('任意の文字列');`

→ `$PGDATA/backup_label`が作られる

```
START WAL LOCATION: 0/14C9900 (file 00000001000000000000000001)
CHECKPOINT LOCATION: 0/14C9900
START TIME: 2007-07-03 20:47:39 JST
LABEL: backup.tar
```

2. `tar cf /tmp/backup.tar $PGDATA`

3. `select pg_stop_backup();`

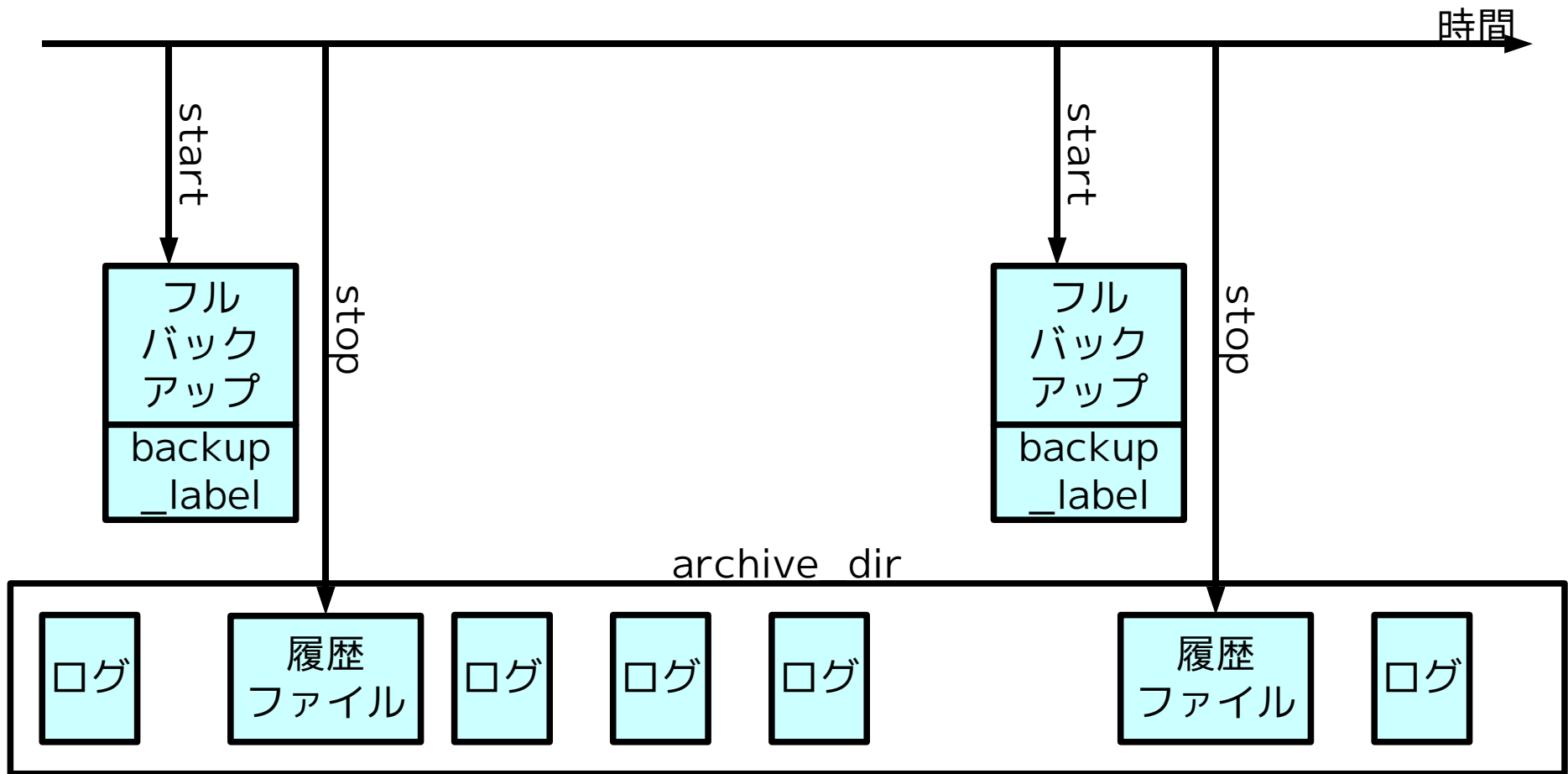
→ バックアップ履歴ファイル

`$PGDATA/pg_xlog/0000.....004C9900.backup`が作られ、`archive_dir`にコピーされる

```
START WAL LOCATION: 0/14C9900 (file 00000001000000000000000001)
STOP WAL LOCATION: 0/14C9964 (file 00000001000000000000000001)
CHECKPOINT LOCATION: 0/14C9900
START TIME: 2007-07-03 20:47:39 JST
LABEL: backup.tar
STOP TIME: 2007-07-03 20:49:25 JST
```

backup_labelと履歴ファイル

- backup_labelとバックアップ履歴ファイルを見比べることで、どこまでのアーカイブログが必要かを判断できる



PITRのリカバリ

- バックアップを展開(パーミッションに注意)
 - \$PGDATA/pg_xlogは不要
- \$PGDATAにrecovery.confを配置
 - share/recovery.conf.sampleを参照
- restore_command

```
restore_command = 'cp /path/to/archive_dir/%f %p'
```

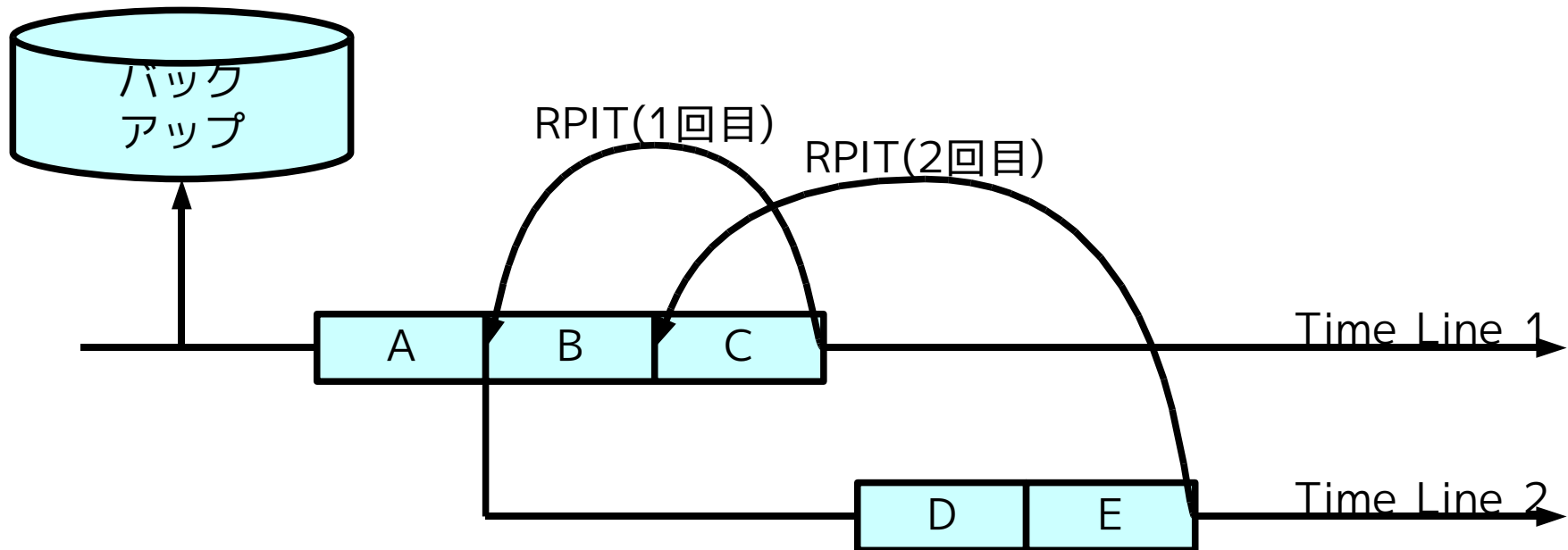
- postmasterを起動
- リカバリが終わると、recovery.confはrecovery.doneという名前にリネームされる

RPIT(Recovery to Point In Time)

- 通常のPITRのリカバリは、アーカイブログの最後までを再生する
- アーカイブログの途中の任意の時点までリカバリするのがRPIT
- 例えば間違っってデータを消してしまった時(オペレーション、プログラム)
- recovery.confの
 - recovery_target_time(時間で指定)
 - recovery_target_xid(トランザクションIDで指定)

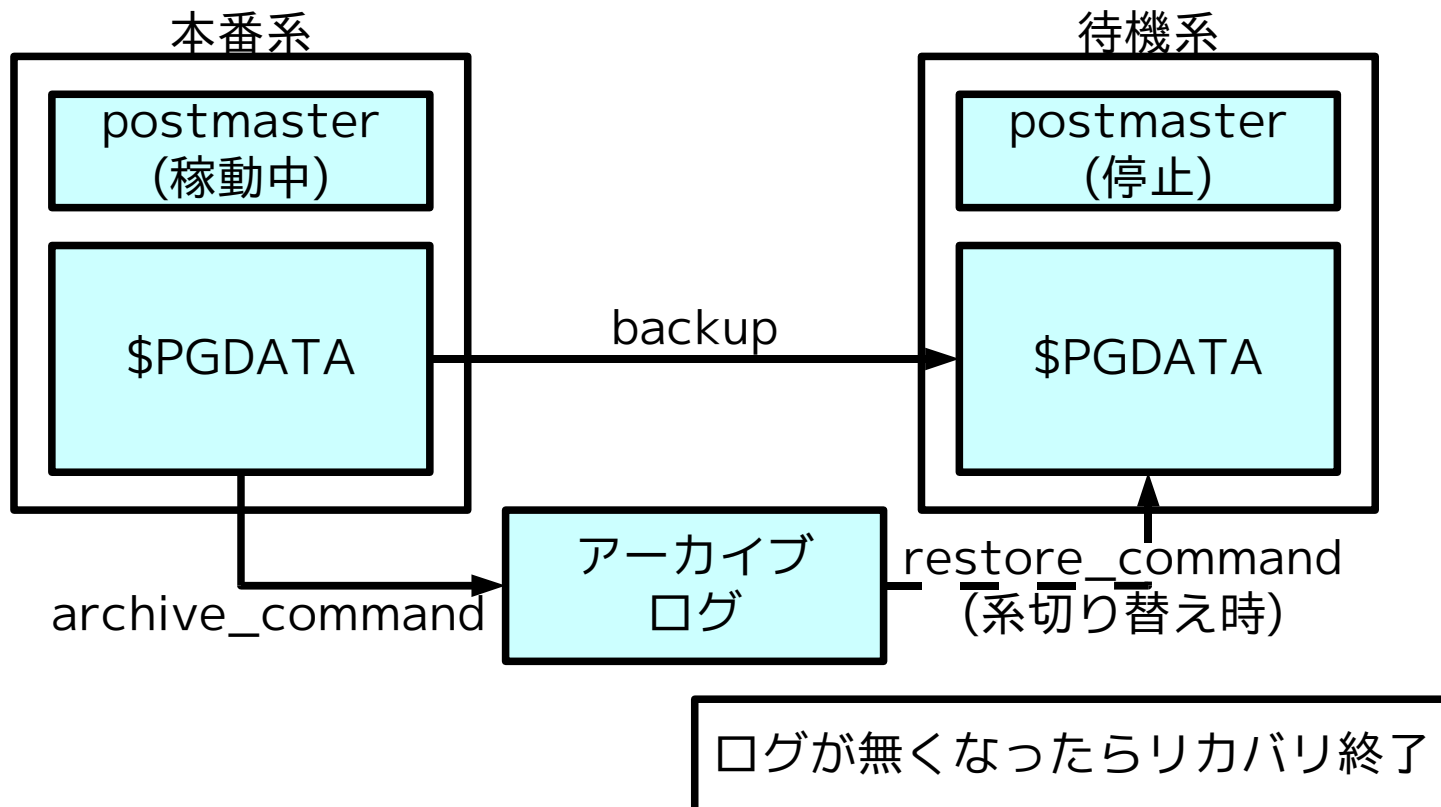
Time Line(時系列)

- CVSなどのブランチに似た機能
- 一回RPITしたんだけど、やっぱりリカバリをやりなおしたい？
- 再生されるログはABC？ABDE？



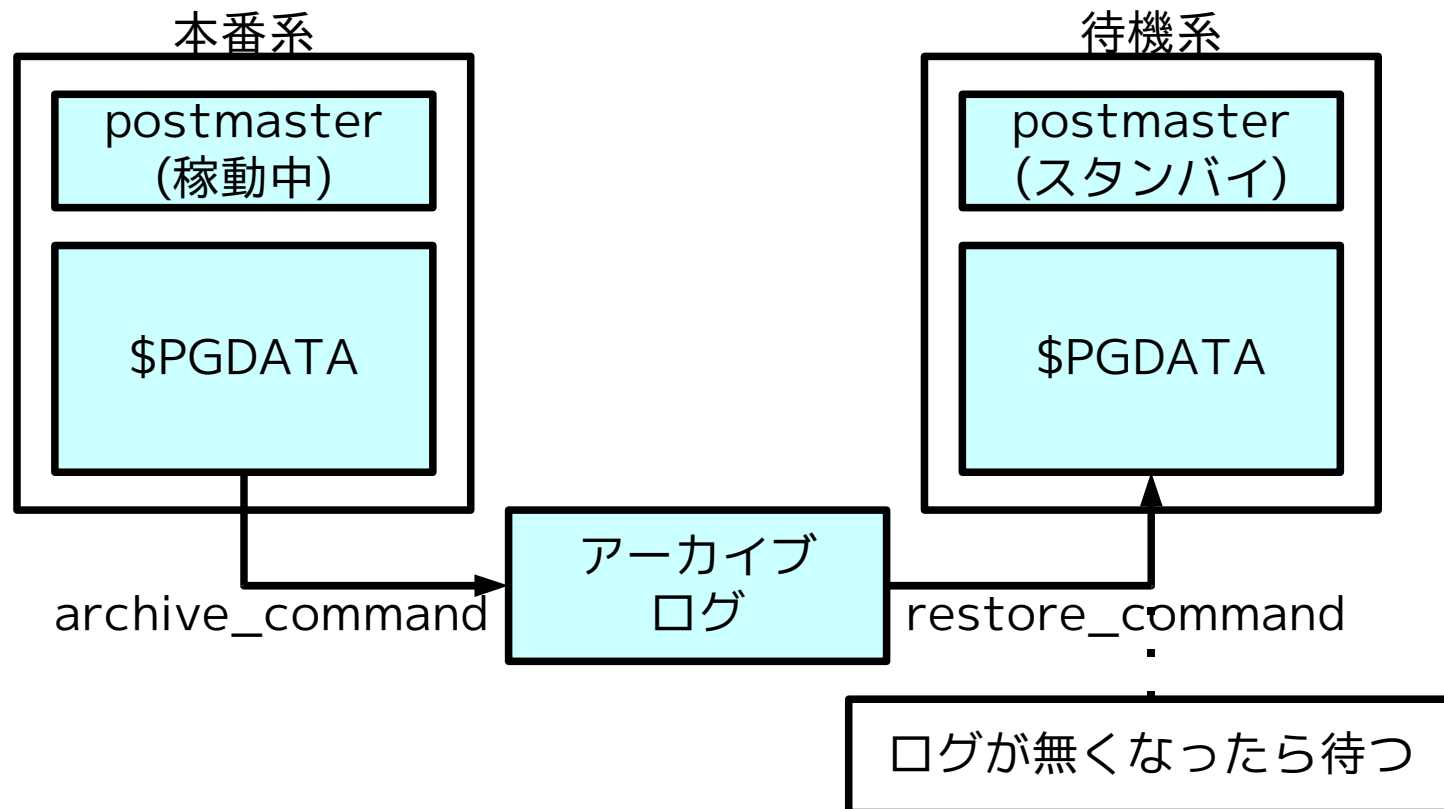
COLD STANDBY

- 本番系がダウンした時に待機系を起動(restore_commandが実行される)
- 待機系の起動時に大量のアーカイブログを読み込むととても起動に時間がかかる



WARM STANDBY

- アーカイブされたものを待機系が次々読み込む
- 8.2ではシェルスクリプト等を自作する必要がある
- 8.3からはpg_standbyがcontribに追加された



pg_standby

- 8.3のcontribに収録
- -t [トリガファイル名] ⇒ トリガファイルが存在すれば、新しいアーカイブログを待つのをやめて、待機系を起動する

```
restore_command = 'pg_standby -c -d -t "C:\\path\\to\\trigger"  
"C:\\path\\to\\archive_dir" "%f" "%p" > standby.log'
```

ご静聴ありがとうございました