

PostgreSQLバックアップ 基礎講座

2025-02-21
SRA OSS K.K. 正野裕大

- 正野裕大 (マサノユウタ)
- SRA OSS K.K. データベース技術グループに所属
- PostgreSQL の技術サポートやコンサルティング、PostgreSQL のトレーニング講師が普段の業務
- 好きな言語は  Go 言語
- 最近フロントエンドに興味湧いて  Svelte を勉強中

株式会社SRA OSS

所在地: 東京都豊島区南池袋2-32-8

設立日: 2022年6月17日

株主: 株式会社SRA
株式会社NTTデータグループ

資本金: 7,000万円

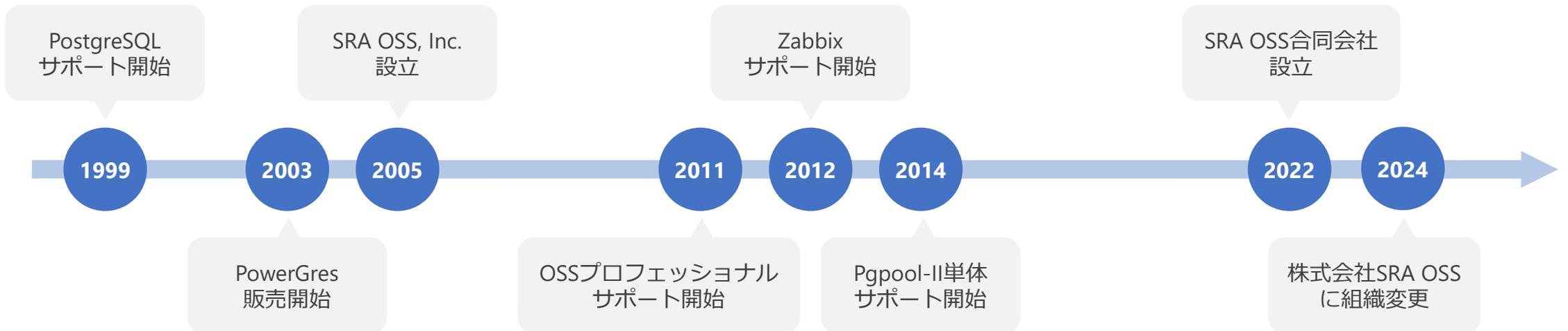
社長: 稲葉 香理

事業内容

- オープンソースソフトウェア (OSS) 関連のサポート、製品開発・販売、構築・コンサル
- OSSの教育、開発、コミュニティ運営支援
- ソフトウェアの研究開発

顧問: 石井 達夫

技術顧問: 増永 良文 (お茶の水女子大学名誉教授)



- PostgreSQL のバックアップの基礎について解説します
- 前提の環境説明
- 論理バックアップとリストア
- 物理バックアップとリストア
- PITR

■ 仮想マシン osc (Rocky Linux 9)

```
[root@osc ~]# cat /etc/redhat-release
Rocky Linux release 9.4 (Blue Onyx)
```

■ PostgreSQL16をインストール(*)

```
# PostgreSQLの公式リポジトリを登録
[root@osc ~]# dnf install -y \
https://download.postgresql.org/pub/repos/yum/repos/EL-9-x86_64/↵ (長いので折返し, 実際は一行)
pgdg-redhat-repo-latest.noarch.rpm

# ディストリビューション側のPostgreSQLモジュールを無効化
[root@osc ~]# dnf -qy module disable postgresql

# PostgreSQL16をインストール
[root@osc ~]# dnf install -y postgresql16-server
```

(*) 開発コミュニティのダウンロードページにOS毎にインストールガイドがまとまっているので参照してください。

■ postgresユーザに変わって環境変数^(*1)を設定

```
[root@osc ~]# su - postgres

[postgres@osc ~]$ cat <<-EOT >.pgsql_profile
export PATH=/usr/pgsql-16/bin/:\$PATH
export PGDATA=/var/lib/pgsql/16/data
EOT
[postgres@osc ~]$ source .pgsql_profile
```

■ PostgreSQLインストール手順の参考情報^(*2)

[2023/01/21]OSS-DB Exam Silver 技術解説セミナー「運用管理：基本的な運用管理作業」
<https://www.youtube.com/watch?v=qeXUdTTW1Uc&t=355s>

- 弊社のYouTubeチャンネルもおすすめ
SRA OSS LLC
<https://www.youtube.com/c/sraoss-official>

(*1) 環境変数: プログラムが動作する際に参照する設定値。RPMパッケージからPostgreSQLをインストールすると、ホームディレクトリ直下の.pgsql_profileから環境変数を読み込めるようになります。
.pgsql_profileを作成せずに.bash_profileで環境変数を設定してもよいです。

(*2) 扱っているPostgreSQLバージョンは16ではありませんが、内容は参考になります。

■ initdbの実行

データベースクラスタ(*)を初期化するPostgreSQLコマンド

```
[postgres@osc ~]$ initdb --encoding=UTF8 --no-locale
```

■ --encoding=ENCODING

データベースのデフォルト文字エンコーディングを指定

■ --no-locale

ロケールを使用しない（慣習的にロケールを使用しないことが推奨されている）

■ データベースクラスタパスの指定方法

\$PGDATAで指定するか--pgdata=《データベースクラスタパス》で指定

■ PostgreSQLの起動

```
[postgres@osc ~]$ pg_ctl start
```

■ 操作対象となるデータベースクラスタをセットで指定する

■ 指定方法は\$PGDATAか--pgdata=《データベースクラスタパス》

(*) データベースクラスタ: ストレージに記録されるPostgreSQLのデータ一式を格納するディレクトリ。

■ データベース(*)benchmarkを作製

```
[postgres@osc ~]$ createdb benchmark # 第一引数に作成したいデータベース名を指定する
```

■ pgbenchでbenchmarkデータベースにデータを作成

```
[postgres@osc ~]$ pgbench --initialize --scale=30 benchmark
```

■ PostgreSQLに同梱されているベンチマークツール

■ --initialize ベンチマークテーブルの初期化

■ --scale=NUM ベンチマークテーブルの規模 (--scale=1でデータベースサイズは15MB程度になる)

■ 第一引数 pgbenchの対象データベース

■ こうしてできた benchmark データベースのデータをバックアップしていきます

(*) データベース: テーブルやインデックスを格納する領域。initdb直後はtemplate0, template1, postgresの3つのデータベースが存在しています。ただし、それらはメンテナンス用データベースなので、アプリのデータを入れてはいけません。

SRA OSS PostgreSQLのバックアップ手法

手法	バックアップ対象	方法
論理バックアップ	PostgreSQLのデータ	PostgreSQLコマンド
物理バックアップ	データベースクラスタ	OSコマンド or PostgreSQLコマンド
PITR (Point In Time Recovery)	データベースクラスタとWAL ファイル ^(*1)	物理バックアップ と WALアーカイブ ^(*2)

(*1) WALファイル: データの書き込み速度と信頼性を担保するファイル。詳細は後述。

(*2) WALアーカイブ: WALファイルをバックアップするPostgreSQLのしくみ。詳細は後述。

- データベースのデータをSQL文としてエクスポートする
- pg_dump
指定のデータベースのデータをバックアップ
- pg_dumpall
全てのデータベースのデータをバックアップ

■ pg_dump

- データベース単位のバックアップを行なうPostgreSQLコマンド
- 実行時に使うデータベースユーザはデータベースの所有者かスーパーユーザ

```
[postgres@osc ~]$ pg_dump --file=benchmark.sql benchmark
```

- --file= 《バックアップファイルの出力パス》
- 第一引数
バックアップしたいデータベース名

■ pg_dumpは出力フォーマットを選択可能

- --format=p テキスト形式 (PostgreSQLデータがSQL文のテキストとして出力される) ※デフォルト
- --format=c カスタム形式 (PostgreSQL独自の圧縮形式)
- --format=t tar形式
- --format=d ディレクトリ形式 (テーブル単位のカスタム形式)
--jobs= 《並列数》 で並列バックアップ可能

```
[postgres@osc ~]$ pg_dump --format=c --file=benchmark.dump benchmark # カスタム形式
```

■ pg_dumpall

- データベース全体のバックアップを行なうPostgreSQLコマンド
- 実行時に使うデータベースユーザはスーパーユーザのみ

```
[postgres@osc ~]$ pg_dumpall --file=db_all.sql
```

- --file= 《バックアップファイルの出力パス》

■ pg_dumpallの出力フォーマットはテキスト形式のみ

SRA OSS pg_dumpしたデータのリストア

■ テキスト形式のバックアップデータ

- psqlを使う
- 実行時に使うデータベースユーザはデータベースの所有者かスーパーユーザ

```
[postgres@osc ~]$ createdb benchmark02 # リストア先データベースを新規作成  
[postgres@osc ~]$ psql --file=benchmark.sql benchmark02 # テキスト形式をリストア
```

- --file= 《リストアに使うバックアップファイルのパス》
- 第一引数
リストア先のデータベース名

■ それ以外の形式

- pg_restoreを使う

```
[postgres@osc ~]$ createdb benchmark03 # リストア先データベースを新規作成  
[postgres@osc ~]$ pg_restore --dbname=benchmark03 benchmark.dump # カスタム形式をリストア
```

- --dbname= 《リストア先のデータベース名》
- 第一引数 リストアに使うバックアップファイル、ディレクトリを指定
- ディレクトリ形式はリストア時に--jobs= 《並列数》で並列リストア可能

SRA OSS pg_dumpallしたデータのリストア

■ データベースクラスタを初期化してリストアする

■ 既存のPostgreSQLを停止

```
[postgres@osc ~]$ pg_ctl stop
```

■ 既存のデータベースクラスタを待避（あるいは削除）

```
[postgres@osc ~]$ mv $PGDATA ${PGDATA}.bak
```

■ データベースクラスタの初期化と設定ファイルのリストア(*)

```
[postgres@osc ~]$ initdb --no-locale --encoding=UTF8  
[postgres@osc ~]$ cp ${PGDATA}.bak/*.conf $PGDATA
```

■ 起動してpsqlでリストア

```
[postgres@osc ~]$ pg_ctl start  
[postgres@osc ~]$ psql --file=db_all.sql postgres
```

(*) pg_dump / pg_dumpall がバックアップするのはデータだけです。つまり、設定ファイルはバックアップされません。別の手段で設定ファイルはバックアップする必要があります。また、今回は設定ファイルを編集していないので、設定ファイルのリストアは必要な手順ですがここでは意味はありません。

SRA OSS 物理バックアップ

- データベースクラスタを構成するディレクトリ・ファイルをコピーする
- オフラインバックアップ（コールドバックアップ）
PostgreSQLを停止してOSコマンドで\$PGDATAを物理バックアップ
- オンラインバックアップ
PostgreSQLを稼働したままPostgreSQLコマンド / OSコマンドで物理バックアップ(*)

(*)もう一つ、PostgreSQLが用意している低レベルAPIを使う方法もありますがここでは取り扱いません。詳細はマニュアルを参照してください。

[PostgreSQL: Documentation: 16: 26.3. Continuous Archiving and Point-in-Time Recovery (PITR)]

<https://www.postgresql.org/docs/16/continuous-archiving.html#BACKUP-LOWLEVEL-BASE-BACKUP>

SRA OSS 物理バックアップ > オフラインバックアップ

■ オフラインバックアップ（コールドバックアップ）

PostgreSQLを停止してOSコマンドで\$PGDATAを物理バックアップ

■ PostgreSQLを停止(*)

```
[postgres@osc ~]$ pg_ctl stop
```

■ たとえばcpコマンドでバックアップ（rsync、tar、あるいは仮想マシンならスナップショットを取得する）

```
[postgres@osc ~]$ cp -r $PGDATA ${PGDATA}.offline
```

■ PostgreSQLを起動

```
[postgres@osc ~]$ pg_ctl start
```

(*) オフラインバックアップ、つまり、OSコマンドで物理バックアップを取得する場合は必ずPostgreSQLを停止します。PostgreSQL稼働中にOSコマンドで取得したバックアップは内部が不正な状態になるので、バックアップデータとして使えずリストアもできません。

■ オンラインバックアップ

PostgreSQLを稼働したままPostgreSQLコマンド / OSコマンドで物理バックアップ

■ pg_basebackup

```
[postgres@osc ~]$ pg_ctl status # 起動していることを確認
pg_ctl: server is running (PID: 3332)
/usr/pgsql-16/bin/postgres
[postgres@osc ~]$ pg_basebackup --pgdata=${PGDATA}.online
```

■ --pgdata= 《バックアップディレクトリ出力パス》

■ pg_basebackupはデフォルトではローカルで起動しているPostgreSQLのデータベースクラスタをバックアップ対象とする

■ リモートPostgreSQLのデータベースクラスタのバックアップを取得したい場合は以下を指定する

--host= 《ターゲットのホスト名 (IPアドレス) 》

--port= 《ターゲットPostgreSQLの待受ポート番号》

--user= 《接続に使うデータベースユーザ名》

■ バックアップデータを\$PGDATAに配置してpg_ctl startで起動

```
[postgres@osc ~]$ pg_ctl stop           # PostgreSQLを停止
[postgres@osc ~]$ rm -rf $PGDATA        # データベースクラスタを削除
[postgres@osc ~]$ cp -r $PGDATA.offline $PGDATA  # オフラインバックアップデータを$PGDATAに配置
[postgres@osc ~]$ pg_ctl start         # バックアップデータで起動
```

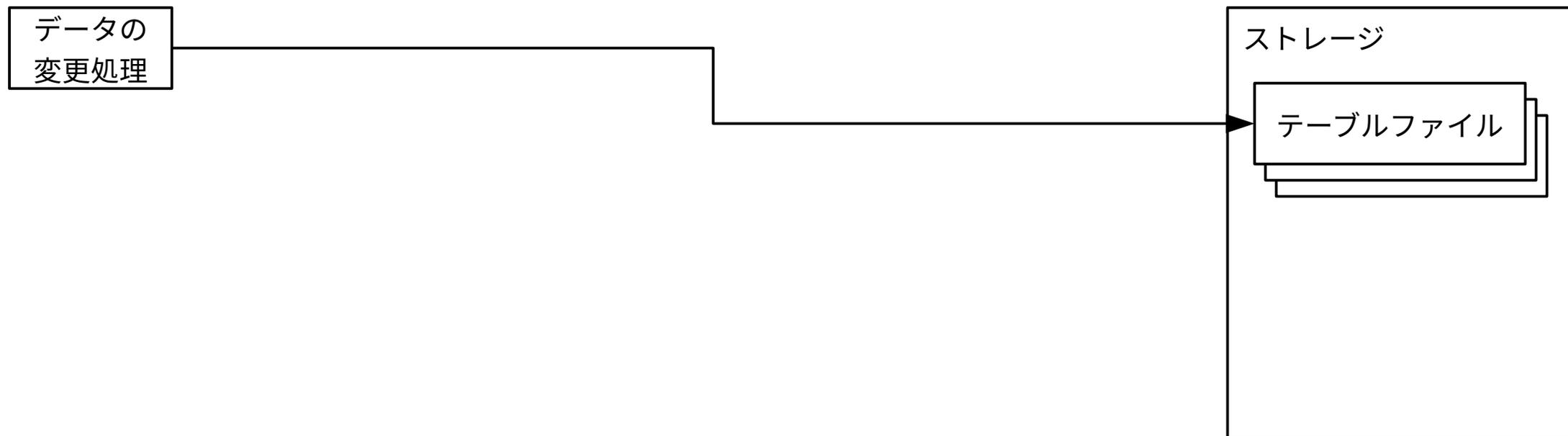
■ PITRとは

- 物理バックアップに加えて**WAL**のバックアップ（WALアーカイブ）も使ったバックアップ手法

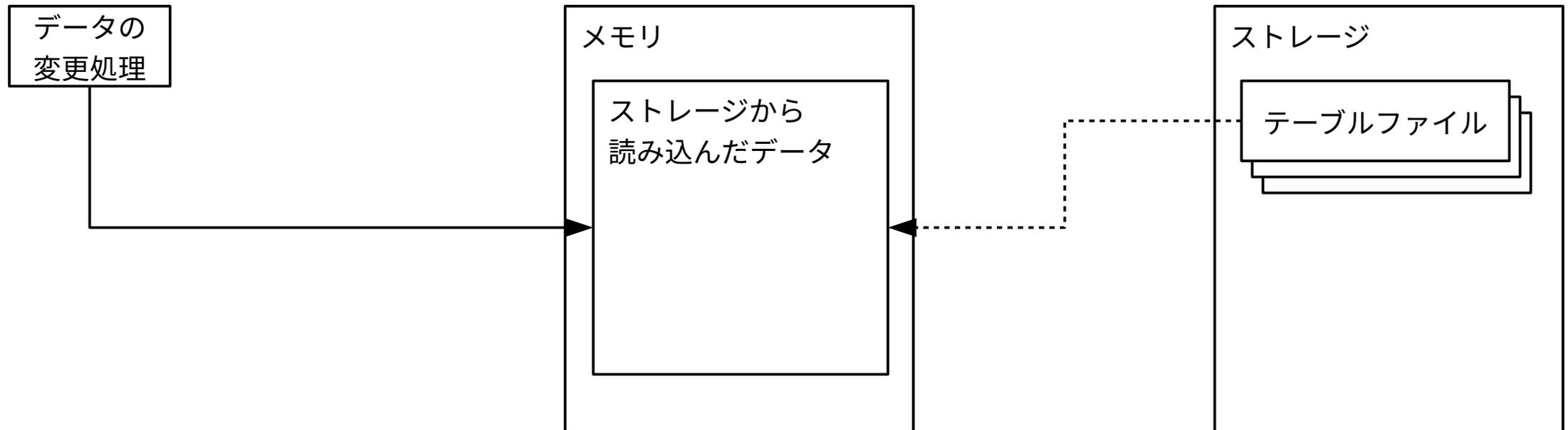
■ WALとは

- PITRの理解にはWALの理解が必要なので先に解説

- 前提: PostgreSQLデータのデータ書き込みについて
 - ストレージに直接書き込むのは高コストなのでやりたくない

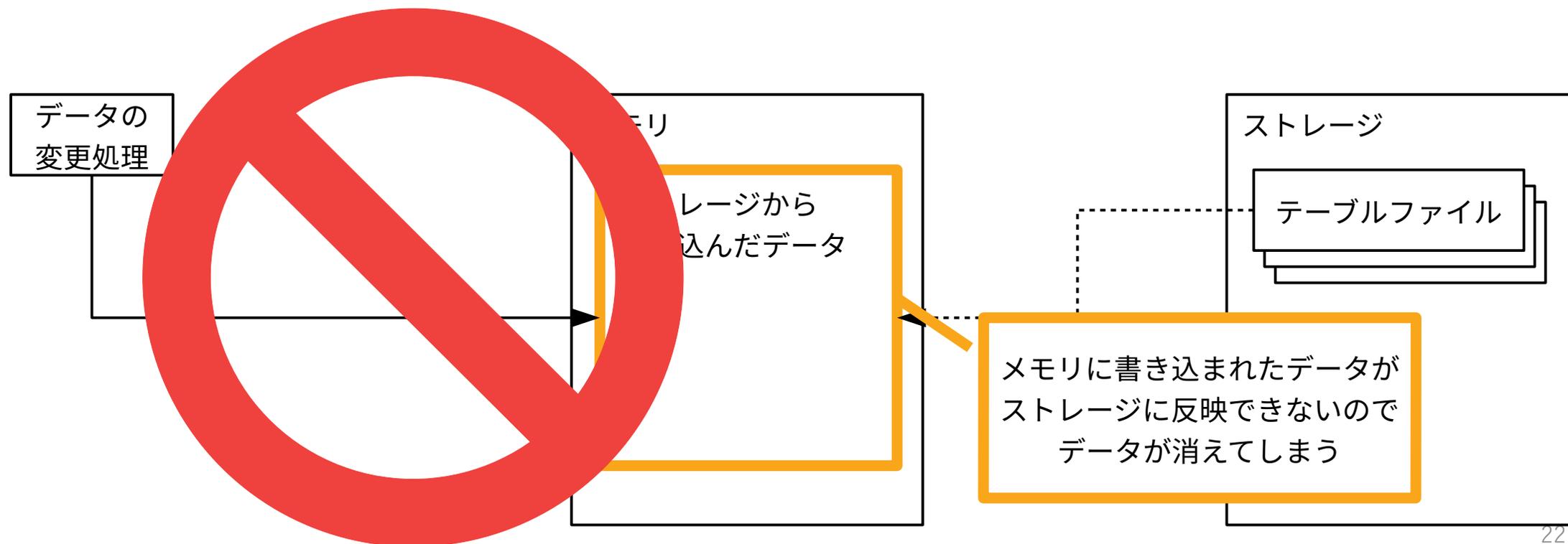


- 前提: PostgreSQLデータのデータ書き込みについて
 - ストレージに直接書き込むのは高コストなのでやりたくない
 - そこで: メモリを介する



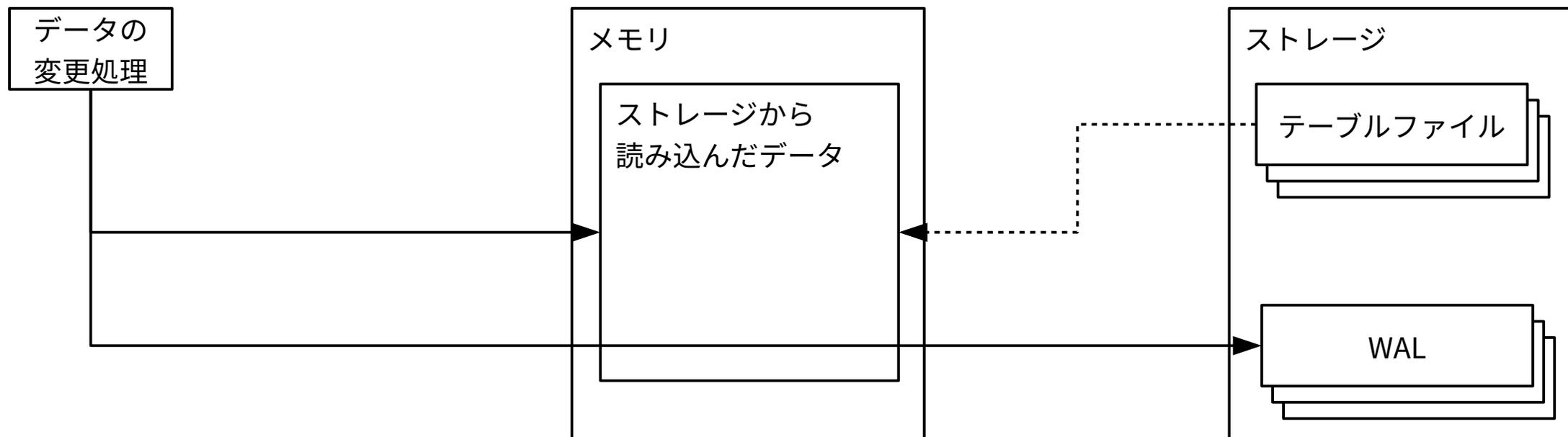
■前提: PostgreSQLデータのデータ書き込みについて

- ストレージに直接書き込むのは高コストなのでやりたくない
- そこで: メモリを介する **しかし: メモリにデータを書き込むだけではPostgreSQLがクラッシュするとデータが消えてしまう**



■前提: PostgreSQLデータのデータ書き込みについて

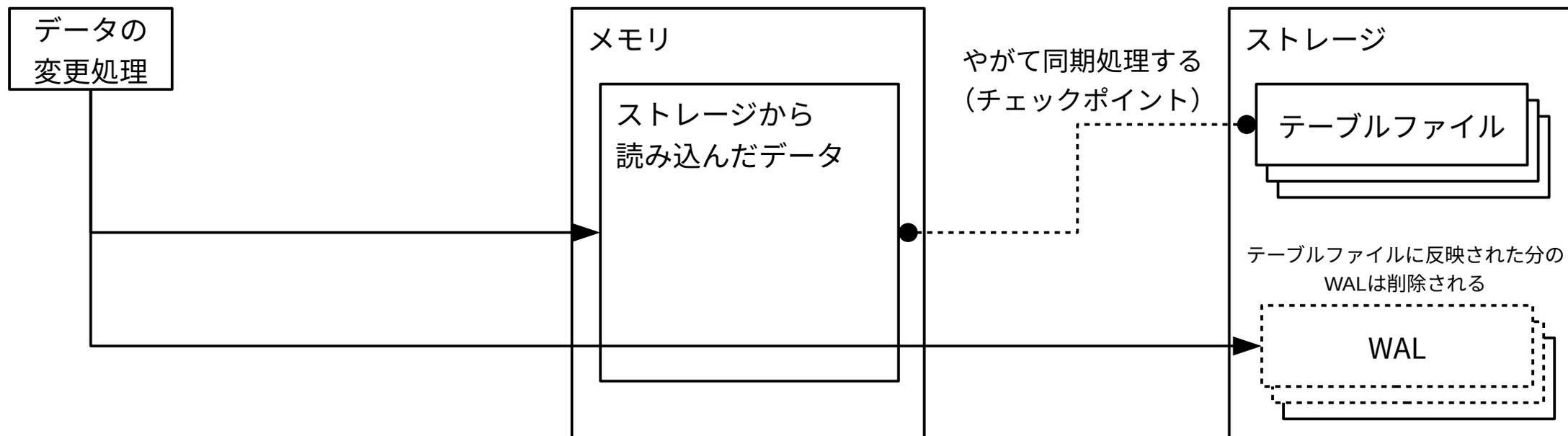
- ストレージに直接書き込むのは高コストなのでやりたくない
- そこで: メモリを介する しかし: メモリにデータを書き込むだけではPostgreSQLがクラッシュするとデータが消えてしまう
- そこで: データの変更内容をWALファイルとメモリに書き込む(*)



(*) 結局ストレージに書き込んでいますがシーケンシャルに追記しているだけなのでテーブルファイルに書き込むよりずっと高速です。

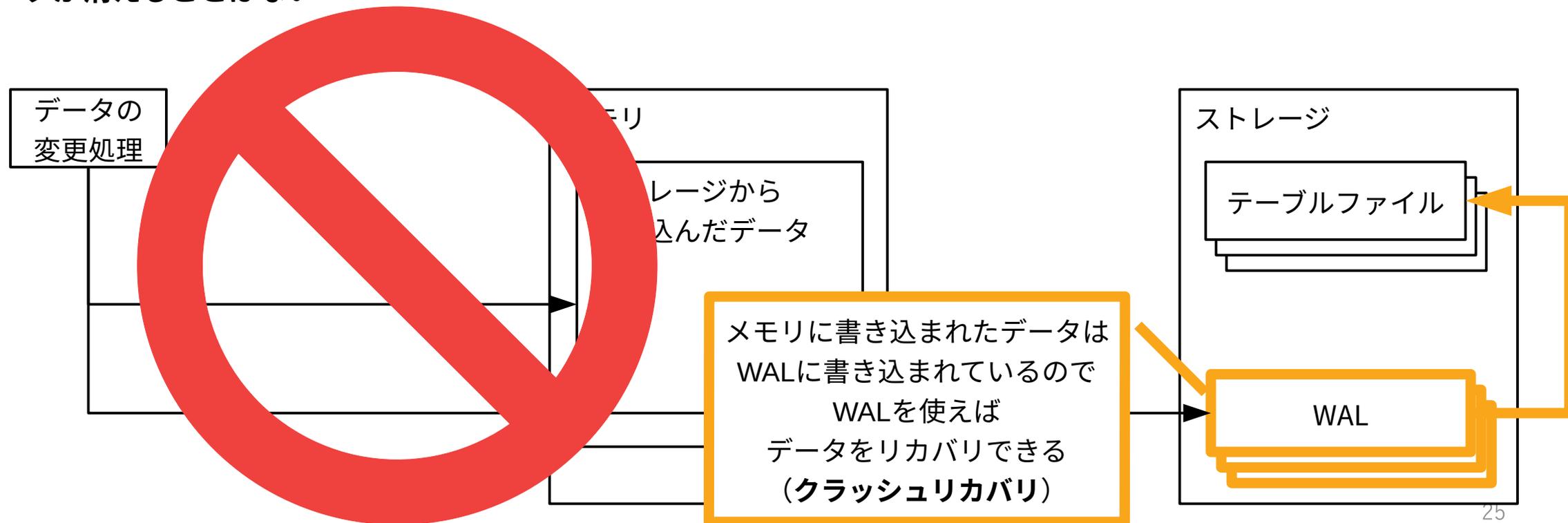
■前提: PostgreSQLデータのデータ書き込みについて

- ストレージに直接書き込むのは高コストなのでやりたくない
- そこで: メモリを介する しかし: メモリにデータを書き込むだけではPostgreSQLがクラッシュするとデータが消えてしまう
- そこで: データの変更内容をWALファイルとメモリに書き込む(*)
- メモリとテーブルファイルが同期 (チェックポイント) したら不要になったWALは削除される



■前提: PostgreSQLデータのデータ書き込みについて

- ストレージに直接書き込むのは高コストなのでやりたくない
- そこで: メモリを介する しかし: メモリにデータを書き込むだけではPostgreSQLがクラッシュするとデータが消えてしまう
- そこで: データの変更内容をWALファイルとメモリに書き込む(*)
- メモリとテーブルファイルが同期 (チェックポイント) したら不要になったWALは削除される
- **これなら: チェックポイント前にPostgreSQLがクラッシュしてもデータの変更内容はストレージに記録されているのでデータが消えることはない**



■ PITRとは

- 物理バックアップに加えてWALのバックアップ（WALアーカイブ）も使ったバックアップ手法
- 物理バックアップにバックアップしたWALを適用してデータのリカバリができる
- バックアップしたWALが適用できる限り任意の時点にデータをリカバリできる

■ WALとは

- データの書き込み速度と信頼性を担保する仕組み
- 不要になると削除される
- PostgreSQLがダウンしてもWALを使ってデータのリカバリができる（クラッシュリカバリ）

SRA OSS PITR (Point In Time Recovery)

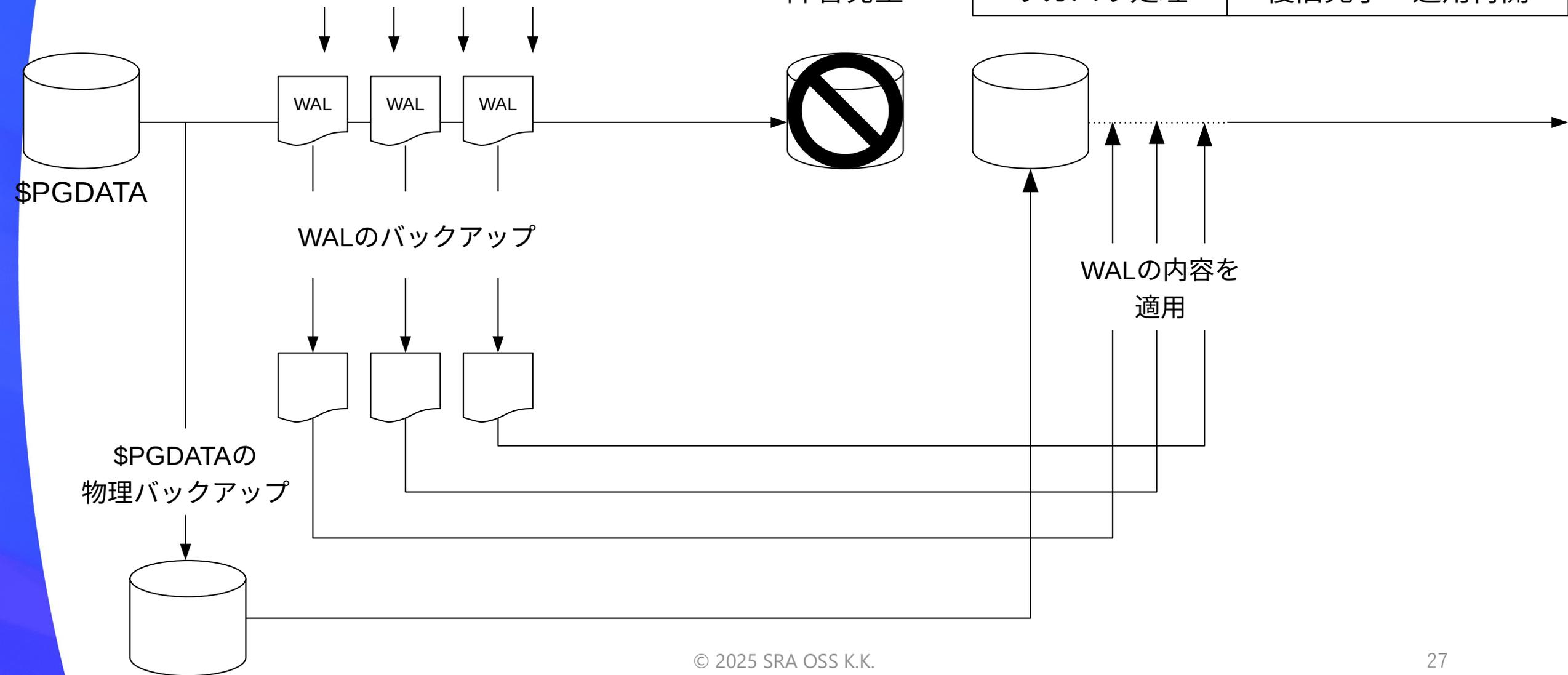
■イメージ図

データの変更

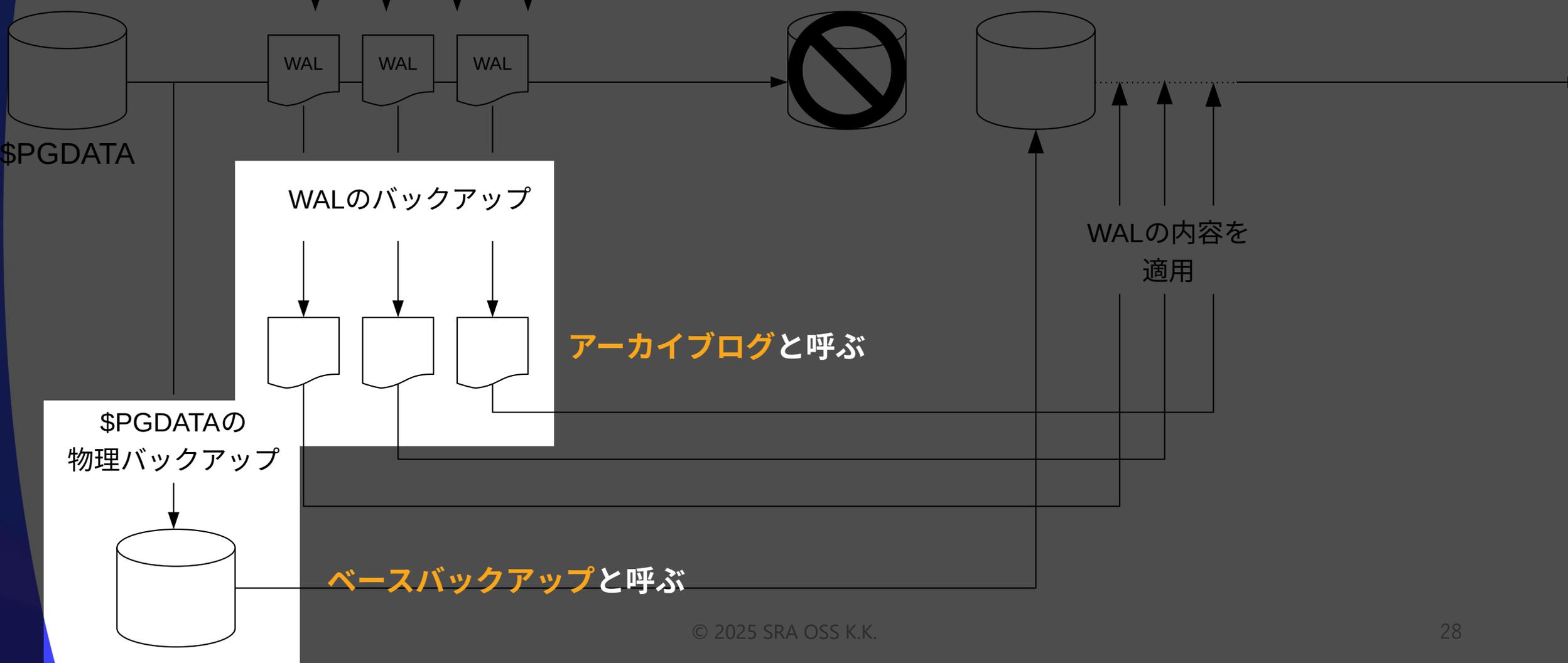
障害発生

リカバリ処理

復旧完了・運用再開



1. ベースバックアップとアーカイブログを格納する領域を作成
2. WALのバックアップ（WALアーカイブ）が行われるように設定変更



SRA OSS PITR (Point In Time Recovery) > 設定

■ ベースバックアップとアーカイブログの格納領域を作製

```
[postgres@osc ~]$ mkdir $HOME/16/backups/base # ベースバックアップ格納ディレクトリを作成  
[postgres@osc ~]$ mkdir $HOME/16/backups/arc # アーカイブログ格納ディレクトリを作成
```

■ WALアーカイブの設定（説明は次のページ）

■ \$PGDATA/postgresql.confの編集と適用

```
@@ -255,12 +255,12 @@  
  
# - Archiving -  
  
-#archive_mode = off # enables archiving; off, on, or always  
+archive_mode = on # enables archiving; off, on, or always  
# (change requires restart)  
#archive_library = '' # library to use to archive a WAL file  
# (empty string indicates archive_command should  
# be used)  
-#archive_command = '' # command to use to archive a logfile segment  
+archive_command = 'cp "%p" "/var/lib/pgsql/16/backups/arc/%f"' # command to use to ...
```

■ archive_mode

onにすると不要になって削除されるWALをアーカイブ領域に転送するようになる

■ archive_command

archive_mode = onのとき、実行するOSコマンド

→ データベースクラスタ内に生成されるWALをアーカイブ領域へ転送する方法はDB管理者が決める

- %p: アーカイブ対象のWALファイルのパス名

- %f: アーカイブ対象のWALファイル名

■ \$PGDATA/postgresql.confの反映

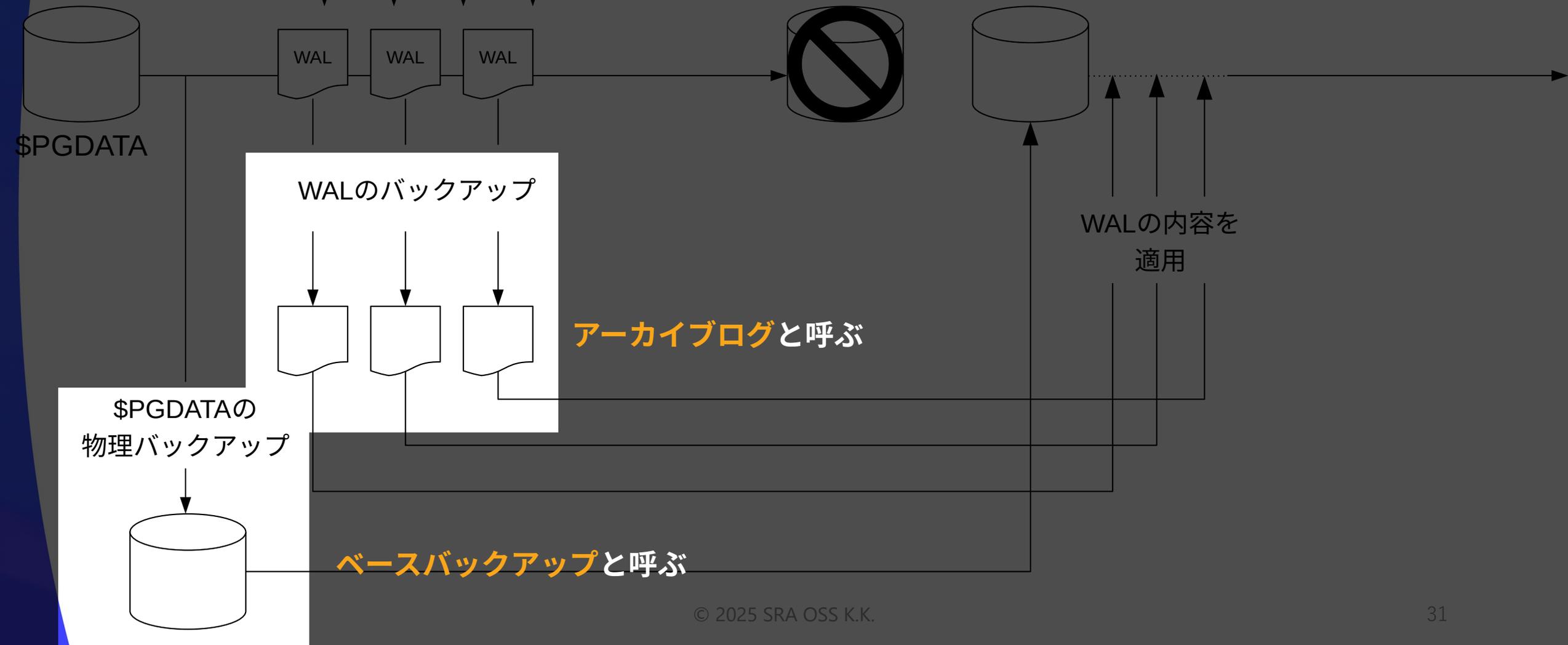
```
[postgres@osc ~]$ pg_ctl restart
```

- 設定ファイルの変更反映はrestartかreload

- 変更したパラメータによって決まる

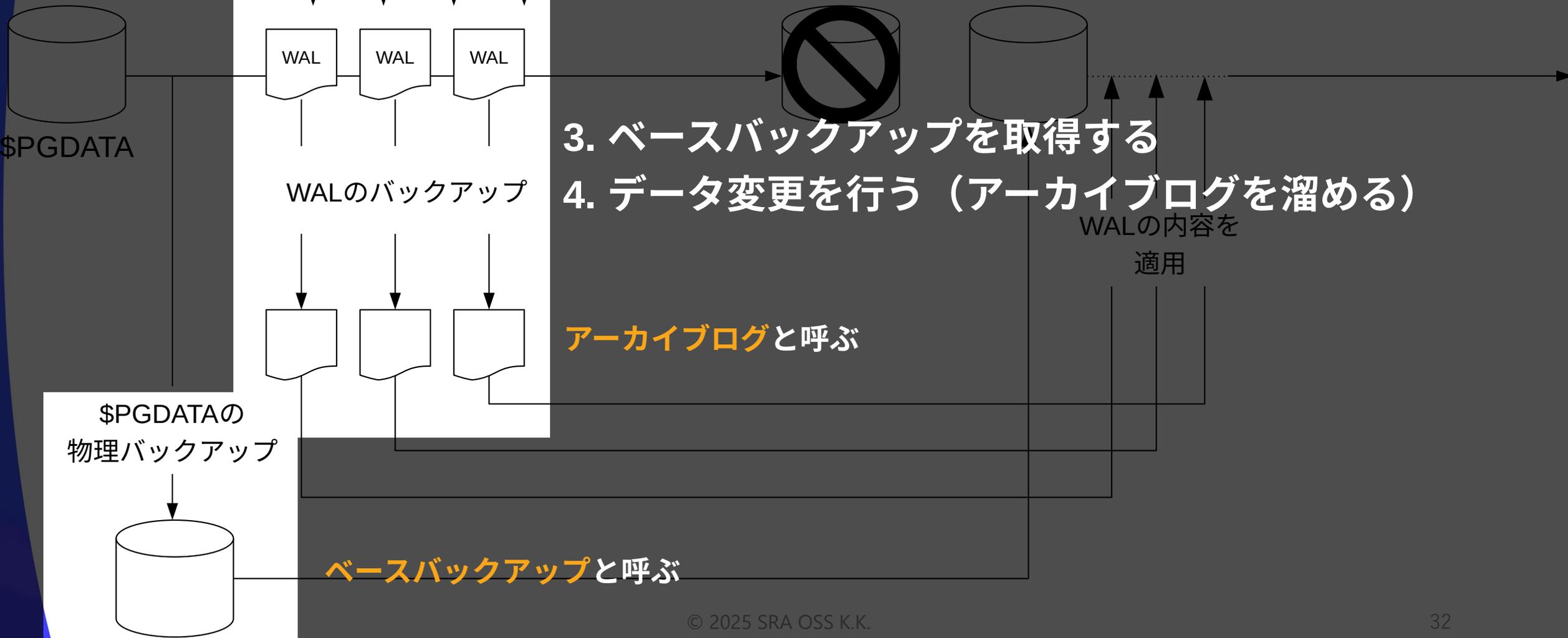
SRA OSS PITR (Point In Time Recovery)

- (済) 1. ベースバックアップとアーカイブログを格納する領域を作成
- (済) 2. WALのバックアップ (WALアーカイブ) が行われるように設定変更



■イメージ図

データの変更



SRA OSS PITR (Point In Time Recovery) > 運用開始

■ ベースバックアップの取得

ベースバックアップ = \$PGDATAの物理バックアップ

■ pg_basebackupでOK

```
[postgres@osc ~]$ pg_basebackup --pgdata=$HOME/16/backups/base/001 --wal-method=none
```

■ --wal-method=none

物理バックアップ対象にWALは含めない
(WALアーカイブしているので含める必要がない)

SRA OSS PITR (Point In Time Recovery) > 運用開始

- benchmarkデータベースにデータ書き込み
pgbenchでベンチマークを実行してデータ書き込みを発生させる

```
[postgres@osc ~]$ pgbench --client=10 --time=30 benchmark
```

- --client= 《ベンチマーククライアントの同時接続数》
- --time= 《ベンチマーク実行時間 (秒) 》

- 履歴テーブルから最新のデータ行を確認しておく(*)

```
[postgres@osc ~]$ psql benchmark
psql (16.2)
Type "help" for help.

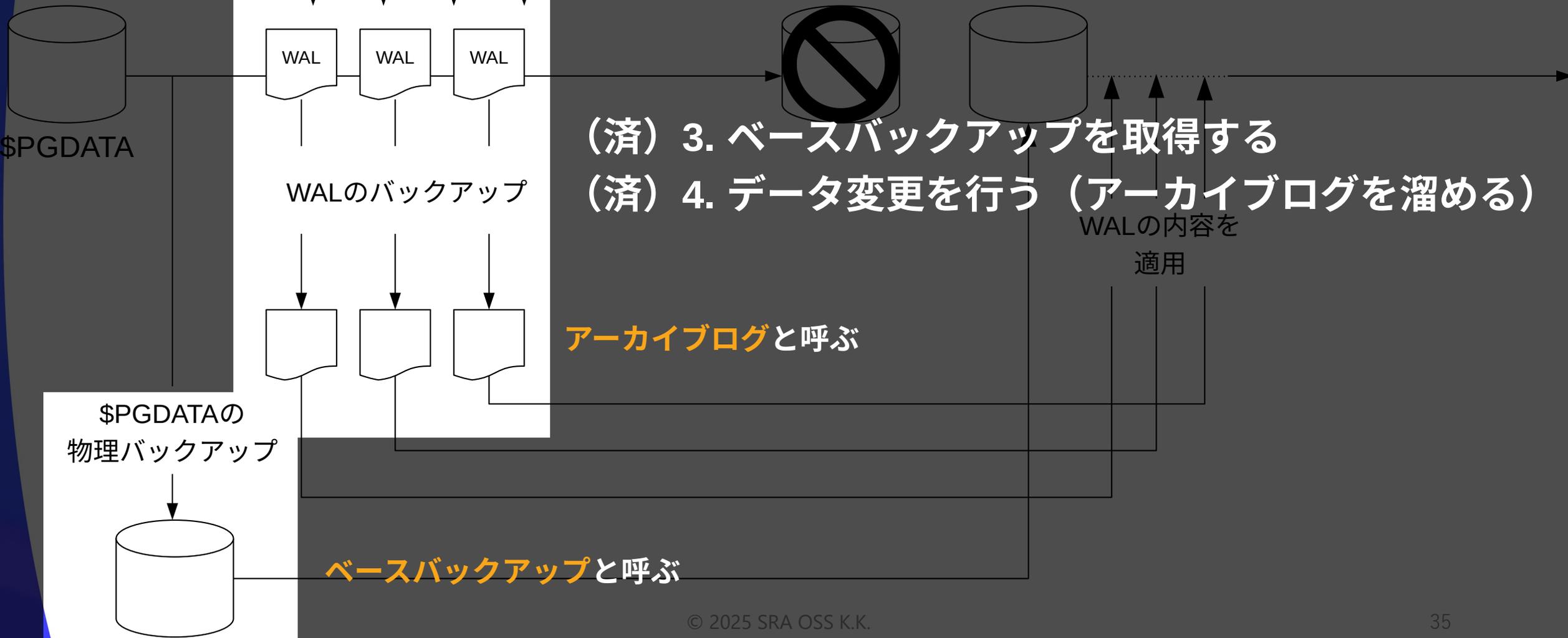
benchmark=# SELECT * FROM pgbench_history ORDER BY mtime LIMIT 1;
 tid | bid |  aid  | delta |          mtime          | filler
-----+-----+-----+-----+-----+-----
   5 |   7 | 2654468 | -4820 | 2024-02-18 13:30:36.983247 |
(1 row)
```

(*) 履歴テーブルのタイムスタンプを、昇順でソート(ORDER BY)して、先頭の1行だけを表示(LIMIT 1)しています。

SRA OSS PITR (Point In Time Recovery)

■イメージ図

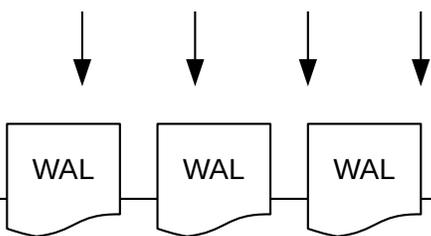
データの変更



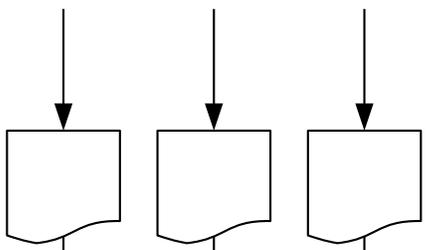
SRA OSS PITR (Point In Time Recovery)

■イメージ図

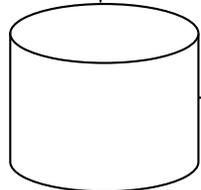
データの変更



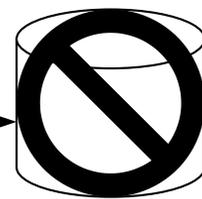
WALのバックアップ



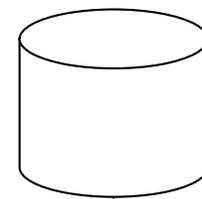
\$PGDATAの物理バックアップ



障害発生



リカバリ処理



WALの内容を適用



復旧完了・運用再開

5. 障害発生
6. 復旧処理

SRA OSS PITR (Point In Time Recovery) > 障害発生

- ベンチマークを実行したデータベースクラスタを強制終了

```
[postgres@osc ~]$ pg_ctl --mode=immediate stop
```

- `--mode=` 《シャットダウンモード》
immediateは強制終了モード（PostgreSQLがクラッシュする）

- 強制終了（障害が発生した）データベースクラスタを待避

```
[postgres@osc ~]$ mv $PGDATA $PGDATA.crash
```

SRA OSS PITR (Point In Time Recovery) > リカバリ処理

- ベースバックアップを\$PGDATAに配置してログを削除(*)

```
[postgres@osc ~]$ cp -a $HOME/16/backups/base/001 $PGDATA  
[postgres@osc ~]$ rm -f $PGDATA/pg_log/*
```

- \$PGDATA/postgresql.confでリカバリ設定

```
@@ -271,7 +271,7 @@  
  
# These are only used in recovery mode.  
  
-#restore_command = ' ' # command to use to restore an archived WAL file  
+restore_command = 'cp "/var/lib/pgsql/16/backups/arc/%f" "%p" ' # command to use to ...
```

- restore_command

アーカイブログを適用するOSコマンドを指定

→ アーカイブログからデータベースクラスタへの転送方法はDB管理者が決める

%p: WALファイルのパス名

%f: アーカイブログのファイル名

(*) これからリカバリして稼働させるデータベースクラスタに、ベースバックアップ取得時点までのログファイルを残しておく意味はありません。

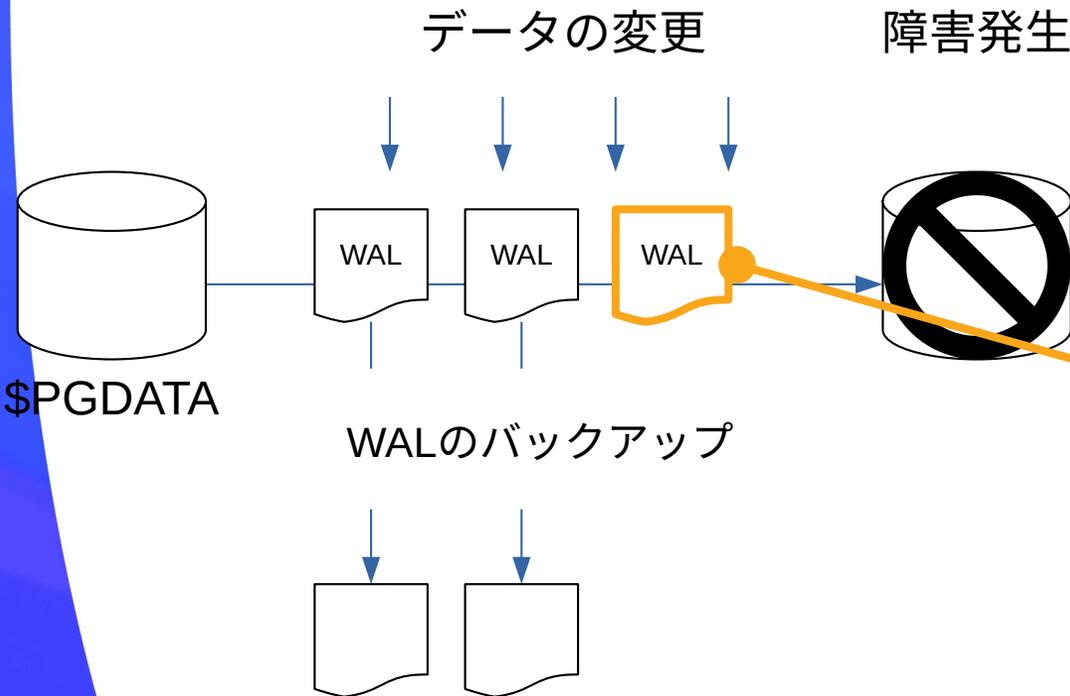
SRA OSS PITR (Point In Time Recovery) > リカバリ処理

■未アーカイブのWALを\$PGDATA/pg_walに配置

```
[postgres@osc ~]$ ls $PGDATA.crash/pg_wal # クラッシュしたデータベースクラスタ内のWALを確認
[postgres@osc ~]$ ls $HOME/16/backups/arc # アーカイブログを確認
[postgres@osc ~]$ cp $PGDATA.crash/pg_wal/* $PGDATA/pg_wal/
```

■なぜやるのか？

- WALがアーカイブされないまま障害が発生するケースもある



■履歴テーブルから最新のデータ行を確認しておく(*)

```
[postgres@osc ~]$ psql benchmark
psql (16.2)
Type "help" for help.

benchmark=# SELECT * FROM pgbench_history ORDER BY mtime LIMIT 1;
 tid | bid |  aid  | delta |          mtime          | filler
-----+-----+-----+-----+-----+-----
   5 |   7 | 2654468 | -4820 | 2024-02-18 13:30:36.983247 |
(1 row)
```

■ \$PGDATA/recovery.signalを作成

```
[postgres@osc ~]$ touch $PGDATA/recovery.signal
```

■ recovery.signal

このファイルがデータベースクラスタ内にある状態でPostgreSQLを起動するとPostgreSQLはリカバリモードで起動する

restore_commandを実行してアーカイブログを適用する

■ リカバリ実行

```
[postgres@osc ~]$ pg_ctl start
```

■ リカバリ中はデータベースに接続できない。

→リカバリするアーカイブログが多いほどデータベースに接続できない時間は長くなる

■ リカバリ完了

- recovery.signalが削除される
- ログファイルに「LOG: archive recovery complete」が出力される
- 障害発生直前の最新データまで復旧できている

```
[postgres@osc ~]$ psql benchmark
psql (16.2)
Type "help" for help.
```

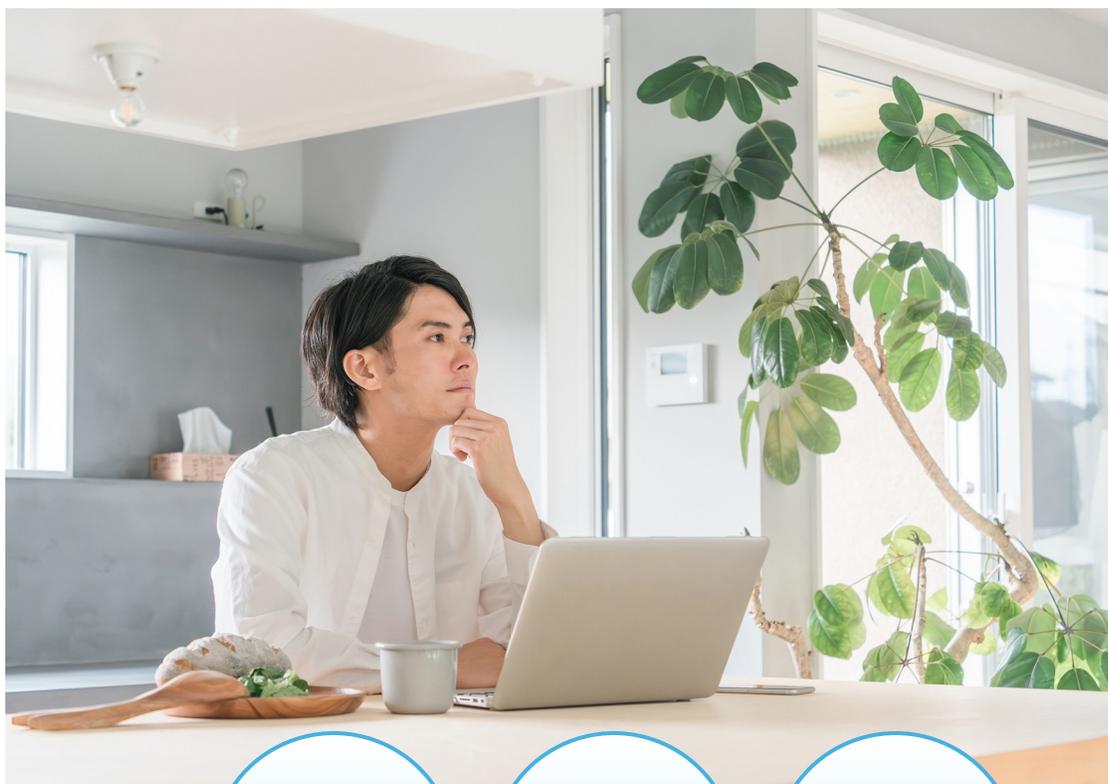
```
benchmark=# SELECT * FROM pgbench_history ORDER BY mtime LIMIT 1;
 tid | bid |  aid  | delta |          mtime          | filler
-----+-----+-----+-----+-----+-----
   5 |   7 | 2654468 | -4820 | 2024-02-18 13:30:36.983247 |
(1 row)
```

手法	バックアップ対象	方法
論理バックアップ	PostgreSQLのデータ	PostgreSQLコマンド
物理バックアップ	データベースクラスタ	OSコマンド or PostgreSQLコマンド
PITR (Point In Time Recovery)	データベースクラスタとWAL ファイル	物理バックアップ と WALアーカイブ

- PostgreSQL 17 からは増分バックアップが可能
PITRのベースバックアップを前回取得したベースバックアップとの差分のみで取得できるようになった
 - バックアップ時間の短縮
 - ストレージ容量の節約
 - リカバリ時間の短縮

- PITRを補助するツールは色々ある
PostgreSQL単体の機能だと煩雑なのでツールに頼るのもよい
 - pg_rman (PostgreSQL のバックアップ/リストア管理ツール)
 - Barman (PostgreSQL PITR 補助ツール)

成長し続けるオープンソースソフトウェアの専門会社で、あなたの可能性を広げてみませんか？



在宅OK
(実績:約8割)

年間休日
125日

【教育】
専門技術
英語

－ 向いている方・SRA OSSの特徴 －



- 特定の分野で力をつけていきたい人
- コミュニティ活動で世界を広げたい方



- ワークライフバランス ◎
(在宅メイン、月平均残業時間：10時間以内)
- コミュニティ活動も業務時間扱い

まずはお気軽に業務内容のお問い合わせや
カジュアル面談からご相談ください。

