FUJITSU

PostgreSQL開発者が語る！ 論理レプリケーションの最前線

PostgreSQL Conference Japan 2023

16:20-17:10 Room B

黒田 隼人

自己紹介

```
# SELECT * FROM me;
-[ RECORD 1 ]-----------------------------------------
name        | 黒田 隼人
affiliation | 富士通
address     | 兵庫県神戸市
job         | フルタイムOSS開発者
etc         | {"趣味": ["旅行", "野球観戦", "ゲーム"]}
```

はじめに

FUJITSU

- 論理レプリケーションの一般的なイメージ

目次

- 論理レプリケーション概論
  - レプリケーション構成の組み方
  - 注意点・制限事項と回避策
- 論理レプリケーションの中身
- 大規模トランザクションに対する論理レプリケーションのアップデート
- PG17に向けて開発中の機能

FUJITSU

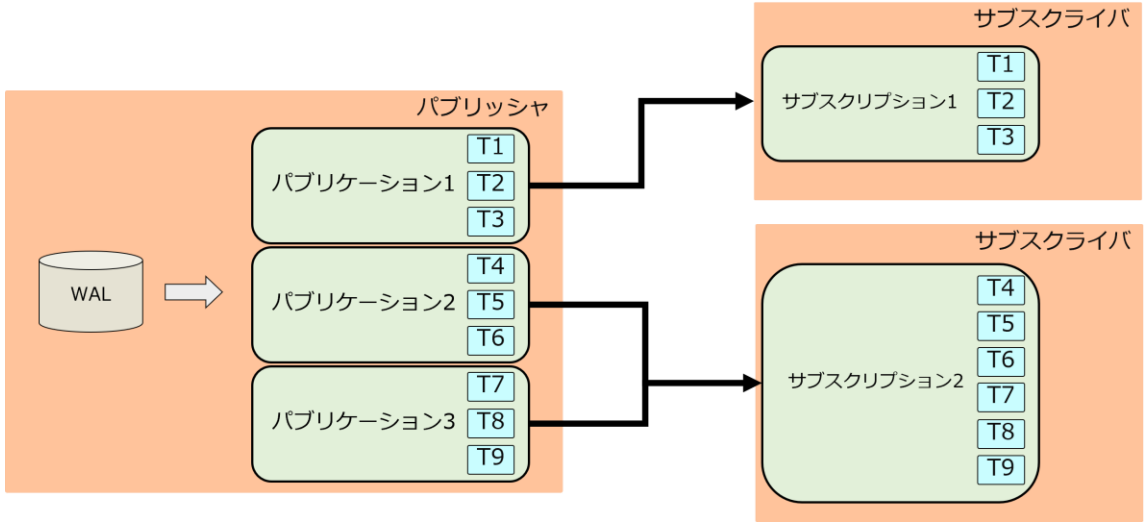
論理レプリケーション概論

論理レプリケーションとは？

FUJITSU

- データとそれに対する更新を転送・複製する仕組み
  - 一部のテーブルやDMLだけを複製することが可能
- 初期のデータ同期と更新追従の2つのフェーズ

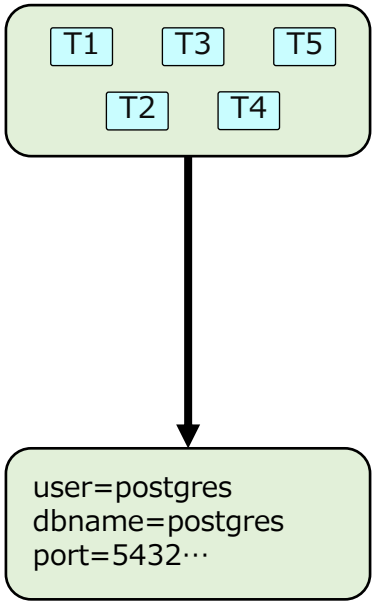
Since PG10



基本用語

- パブリケーション Publication
  - データと更新を送るレプリケーションの上流
  - 論理レプリケーションの対象となるテーブルと複製するDMLの集合
  - パブリケーションを持つノードをパブリッシャと呼ぶ

- サブスクリプション Subscription
  - レプリケーションの下流
  - パブリッシャへの接続文字列と購読するパブリケーションの集合
  - サブスクリプションを持つノードをサブスクライバと呼ぶ



ストリーミングレプリケーションとの違い

FUJITSU

	ストリーミング(物理)レプリケーション	論理レプリケーション
上流/下流ノードの呼び方	プライマリ/スタンバイ	パブリッシャ/サブスクライバ
下流に送る内容	WALそのもの	WALから抽出された更新内容
複製する対象	DBクラスタ全体	DB単位 テーブル単位・DML単位で指定
下流ノードでできること	データ参照のみ	データ参照・更新
設定できる上流ノードの数	1つだけ	複数設定が可能
上流/下流ノードの環境	同一OS・メジャーバージョンのみ	異なるOS・メジャーバージョン間でも可

⇨ **バックアップに特化**
上流と下流で完全に一致

⇨ **より幅広い用途**
上流と下流の状態が異なる

ユースケース

- 業務継続性を高めたアップグレード
- IoT機器が計測したデータをデータセンタへと集約

業務継続性を高めたアップグレード

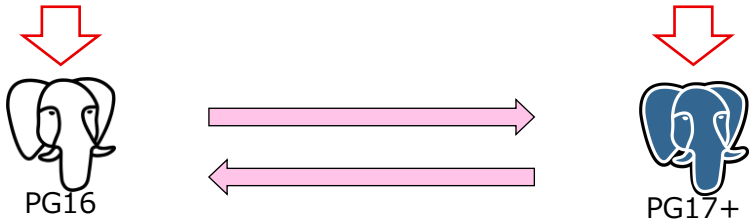
FUJITSU

- 業務を継続したまま、DB内のデータを新環境に移せる
  - 古いDBをパブリッシャ、新しいDBをサブスクライバに設定
  - 初期データ同期と更新の追従が行われ、自動的にデータ移行が完了する
- 同期中に行われたデータの追加や変更も自動的に反映される

- 双方向レプリケーションを設定すれば、切り戻しに対応することも可能

Since PG16

  - データ同期後が完了した後、旧系->新系の論理レプリケーションを構築
  - その後業務アプリケーションの接続先を切り替えることで、新系への変更を旧系に反映できる
  - その後バージョン違いに起因する問題が見つかった場合、再度旧系に接続すればよい



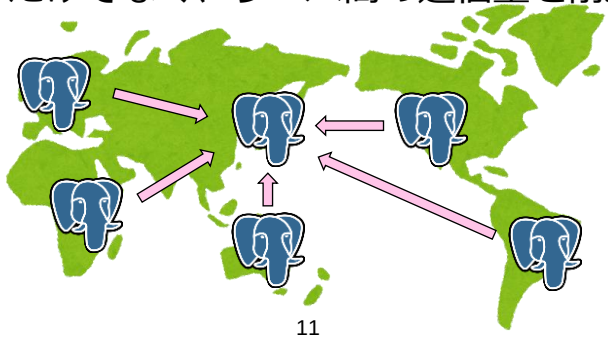
IoT機器が計測したデータをデータセンタへと集約

FUJITSU

- 複数個所で計測されたデータを１か所に集約・分析ができる
  - 複数のデータ収集用DBにパブリケーションを作成する
  - 分析用DBに複数のサブスクリプションを作成し、すべてのパブリケーションを購読する
  - IoT機器は最寄りのデータ収集用DBにデータを書き込む

- 有効なデータだけを複製することで、同時にデータクレンジングも可能
  - 例：計測された電圧が3V以上の場合のみ複製
  - パブリケーションに行フィルタを設定することで、分析用DBに送るデータを絞る
  - DBクラスタの容量を節約するだけでなく、サーバ間の通信量を削減できる

Since PG15



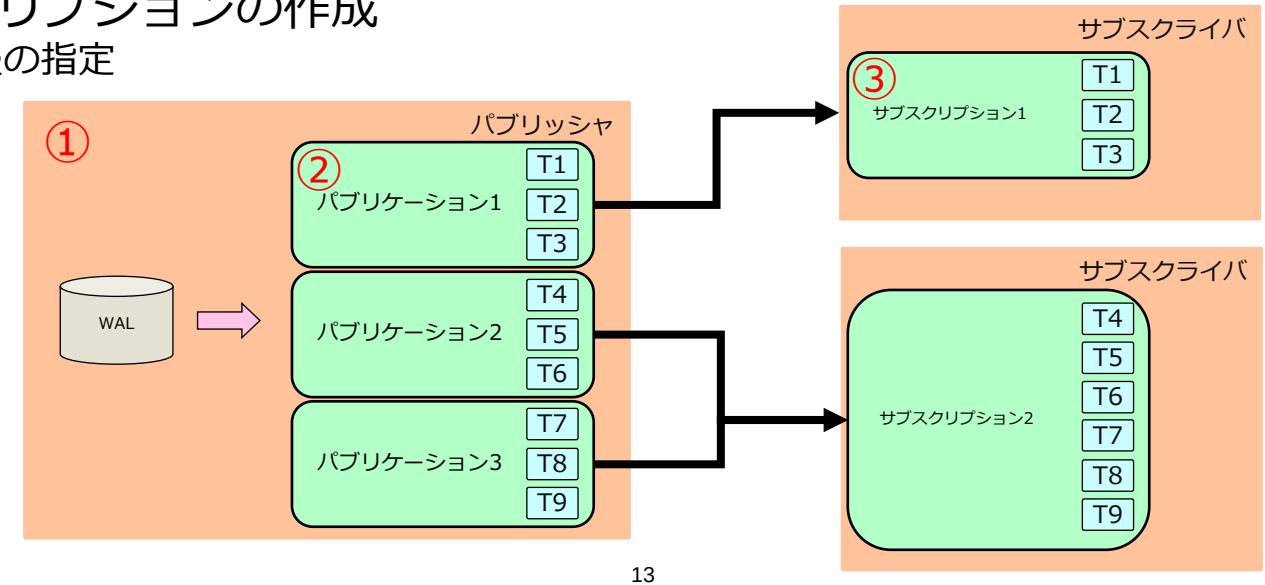
レプリケーション構成の組み方

レプリケーション構成の組み方

① パブリッシャ・サブスクライバのパラメータ設定
… 論理レプリケーションを開始するための準備

② パブリケーションの作成
… レプリケーション対象の決定

③ サブスクリプションの作成
… 接続情報の指定



手順① パラメータ設定

- **パブリッシャ上で`wal_level`を`logical`に変更する**
  - デフォルト値は`replica`

- その他のパラメータはデフォルトでも動作する
  - マシンスペックやネットワーク環境、レプリケーション構成に応じて調整

手順② パブリケーションの作成

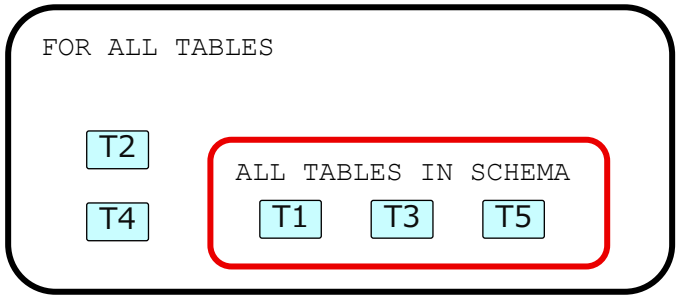
FUJITSU

- 複製の対象となるテーブルを指定する
  - 全テーブル（FOR ALL TABLES）
  - 特定スキーマ内の全テーブル（FOR TABLES IN SCHEMA schema）
  - 特定テーブルの条件を満たす行のみ（FOR TABLE tbl WHEHR…）

Since PG15

Since PG15

- 複製するDMLも指定できる
  - デフォルトではデータの挿入/更新/削除すべてが対象
  - データの挿入だけを対象にすることも可能



tbl

Id	Val
1	6.923085
2	2.0701537
3	3.796526
8	1.4896693

手順② パブリケーションの作成

FUJITSU

- CREATE PUBLICATION文でパブリケーションを作成できる
- 定義内容はpg_publicationカタログに記録される

```
publisher=# CREATE PUBLICATION pub FOR ALL TABLES;
CREATE PUBLICATION
publisher=# SELECT pubname, puballtables, pubinsert, pubupdate, pubdelete, pubtruncate
              FROM pg_publication;
 pubname | puballtables | pubinsert | pubupdate | pubdelete | pubtruncate
---------+--------------+-----------+-----------+-----------+-------------
 pub     | t            | t         | t         | t         | t
(1 row)
```

- パブリケーションに含まれるテーブルはpg_publication_relで確認可能

手順③ サブスクリプションの作成

FUJITSU

- パブリッシャへの接続情報を指定する
  - 指定された内容を元にパブリッシャへと接続することで、論理レプリケーションが開始される

- 一部論理レプリケーションに関するオプションも指定できる
  - 例：初期データ同期を行わない
  - 例：大規模トランザクションをストリームする 

  - 例：エラー時に自動無効化 

手順③ サブスクリプションの作成

FUJITSU

- CREATE SUBSCRIPTION文でパブリケーションを作成できる
  - 実行できるのはsuperuserまたはpg_create_subscriptionロール

Since PG16

- 定義内容はpg_subscriptionカタログに記録される

```
subscriber=# CREATE SUBSCRIPTION sub CONNECTION 'user=postgres host=localhost port=5431'
             PUBLICATION pub;
NOTICE:  created replication slot "sub" on publisher
CREATE SUBSCRIPTION

subscriber=# SELECT subdbid, subname, subconninfo, subpublications FROM pg_subscription;
 subdbid | subname |              subconninfo               | subpublications
---------+---------+----------------------------------------+-----------------
       5 | sub     | user=postgres host=localhost port=5431 | {pub}
(1 row)
```

- 更新されるテーブルはpg_subcription_relカタログで確認可能

注意点・制限事項と回避策

注意点・制限事項

FUJITSU

1. データ競合が発生するとレプリケーションは停止する
2. DDLは複製できない

注意点・制限事項①

- データ競合が発生するとレプリケーションは停止する
  - 競合の自動解決はできないので、ユーザが自分で直す必要がある
  - デフォルト設定の場合、繰り返し適用を試みてエラーになる

Time1

```
subscriber=# INSERT INTO employee VALUES (1, 'kuroda');
INSERT 0 1
```

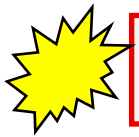
subscriber

Time2

```
publisher=# INSERT INTO employee VALUES (1, 'fujitsu');
INSERT 0 1
```

publisher

Time3



```
ERROR: duplicate key value violates unique constraint "employee_id_key"
DETAIL: Key (id)=(1) already exists.
…
```

注意点・制限事項① | 回避策・解決策

FUJITSU

回避策

- 競合しうるデータの挿入を避ける
  - 例：サブスクライバ側では参照のみを行う
  - 例：ノード毎に担当する主キーの範囲を定め、その範囲のデータのみ挿入する

解決策

- 競合データの削除
  - DELETE文やUPDATE文により、競合の原因となったデータを書き換える
  - パブリッシャ側のデータを優先したい場合にはこれで十分
- ALTER SUBSCRIPTION SKIPコマンド
  - 特定のトランザクションの適用をスキップする
  - サブスクライバ側のデータを優先したい場合にはこちらを使用する
  - 詳しい説明は次ページ

Since PG15

注意点・制限事項① | 解決策

FUJITSU

- ALTER SUBSCRIPTION SKIPコマンド

Since PG15

  - 始めにログを確認し、競合を引き起こしたトランザクションのLSNを確認する

```
ERROR:  duplicate key value violates unique constraint "employee_pkey"
DETAIL:  Key (id)=(1) already exists.
CONTEXT:  processing remote data for replication origin "pg_16394" during message type "INSERT" for
replication target relation "public.employee" in transaction 736, finished at 0/150FD60
```

  - その後ALTER SUBUSRIPTION SKIPで対象のLSNを指定し、該当トランザクションを読み飛ばす

```
subscriber=# ALTER SUBSCRIPTION mysub DISABLE ;
ALTER SUBSCRIPTION
subscriber=# ALTER SUBSCRIPTION mysub SKIP ( lsn = '0/150FD60' );
ALTER SUBSCRIPTION
subscriber=# ALTER SUBSCRIPTION mysub ENABLE ;
ALTER SUBSCRIPTION
```

  - ログを再度確認すると、該当トランザクションのスキップを確認できる

```
LOG:  logical replication starts skipping transaction at LSN 0/150FD60
CONTEXT:  processing remote data for replication origin "pg_16394" during message type "BEGIN" in
transaction 736, finished at 0/150FD60
```

注意点・制限事項②

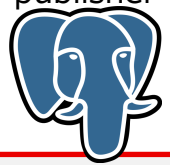
FUJITSU

- DDLは複製できない
  - 全テーブルを対象としてパブリケーションを作成し、その後にテーブルを定義すると…

Time1

```
publisher=# CREATE PUBLICATION pub FOR ALL TABLES;
CREATE PUBLICATION
```

publisher



Time2

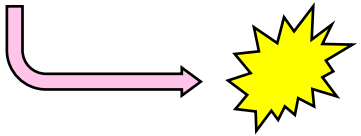
```
subscriber=# CREATE SUBSCRIPTION sub CONNECTION…;
NOTICE: created replication slot "sub" on publisher
CREATE SUBSCRIPTION
```

subscriber

Time3

```
publisher=# CREATE TABLE songs (id int PRIMARY KEY, name text);
CREATE TABLE
publisher=# INSERT INTO record VALUES (1, 'We are the world');
INSERT 0 1
```

Time4



```
ERROR:  logical replication target relation "public.songs"
does not exist
```

注意点・制限事項② | 回避策・解決策

FUJITSU

回避策

- 複製の対象をテーブルをスキーマ・個別単位で指定する
  - 「全テーブル」を対象としないことで、勝手に対象となることを防ぐ
  - 両ノードにテーブルを作成後、新たなテーブルも複製対象に追加する
    (ALTER PUBLICATION・ALTER SUBSCRITION)
- スキーマ定義を一致させてから論理レプリケーションを開始する
  - 基本的にはこちらを推奨

解決策

- 手動でサブスクライバ側にもテーブルを作成する

目次

- 論理レプリケーション概論
  - レプリケーション構成の組み方
  - 注意点・制限事項と回避策
- 論理レプリケーションの中身
- 大規模トランザクションに対する論理レプリケーションのアップデート
- PG17に向けて開発中の機能

FUJITSU

論理レプリケーションの中身

論理レプリケーションとは…

FUJITSU

論理デコーディング

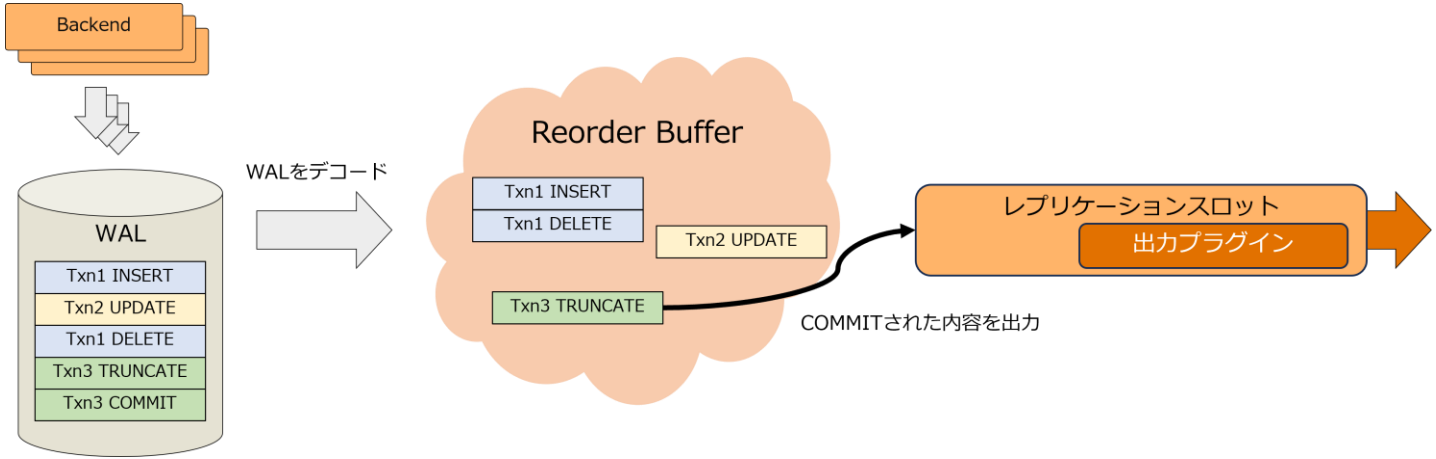
＋

レプリケーション

論理デコーディング

FUJITSU

- データベース内の更新内容を外部に出力する仕組み
- WALをデコードして関係する更新だけを抽出する
- 出力フォーマットはプラグインとして設定可能
  - 公式リポジトリには2種のサンプルプラグインが同梱
  - APIは公開されているので自作も可能：https://github.com/HUUTFJ/hayadecoder



論理デコーディング | 論理レプリケーションスロット

FUJITSU

- どこまでのWALを読み取って出力したのかを記録
  - Checkpoint時に未デコードのWALファイルが削除されるのを防ぐ
- スロット名と出力プラグイン名を指定して作成

不要なスロットを放置していると、WALファイルがリサイクル削除されずにディスクが一杯になるかも……

```
postgres=# SELECT * FROM pg_create_logical_replication_slot('test', 'test_decoding');
 slot_name |    lsn
-----------+-----------
 test      | 0/1506180
(1 row)
postgres=# SELECT slot_name, plugin, database, confirmed_flush_lsn
           FROM pg_replication_slots;
 slot_name |    plugin     | database | confirmed_flush_lsn
-----------+---------------+----------+---------------------
 test      | test_decoding | postgres | 0/1506180
(1 row)
```

論理デコーディング | 更新内容の出力

FUJITSU

- トランザクションがコミットされると、更新内容を抽出できる

```
postgres=# INSERT INTO tbl VALUES (generate_series(1, 2));
INSERT 0 2
postgres=# SELECT * FROM pg_logical_slot_get_changes('test', NULL, NULL);
    lsn     | xid |                  data
------------+-----+----------------------------------------
 0/15061B8 | 734 | BEGIN 734
 0/15061B8 | 734 | table public.tbl: INSERT: id[integer]:1
 0/1506298 | 734 | table public.tbl: INSERT: id[integer]:2
 0/1506348 | 734 | COMMIT 734
(4 rows)
```

- DDLを抽出することはできない

```
postgres=# CREATE TABLE foo (id int);
CREATE TABLE
postgres=# SELECT * FROM pg_logical_slot_get_changes('test', NULL, NULL);
    lsn     | xid |    data
------------+-----+------------
 0/15063B0 | 735 | BEGIN 735
 0/150D950 | 735 | COMMIT 735
(2 rows)
```

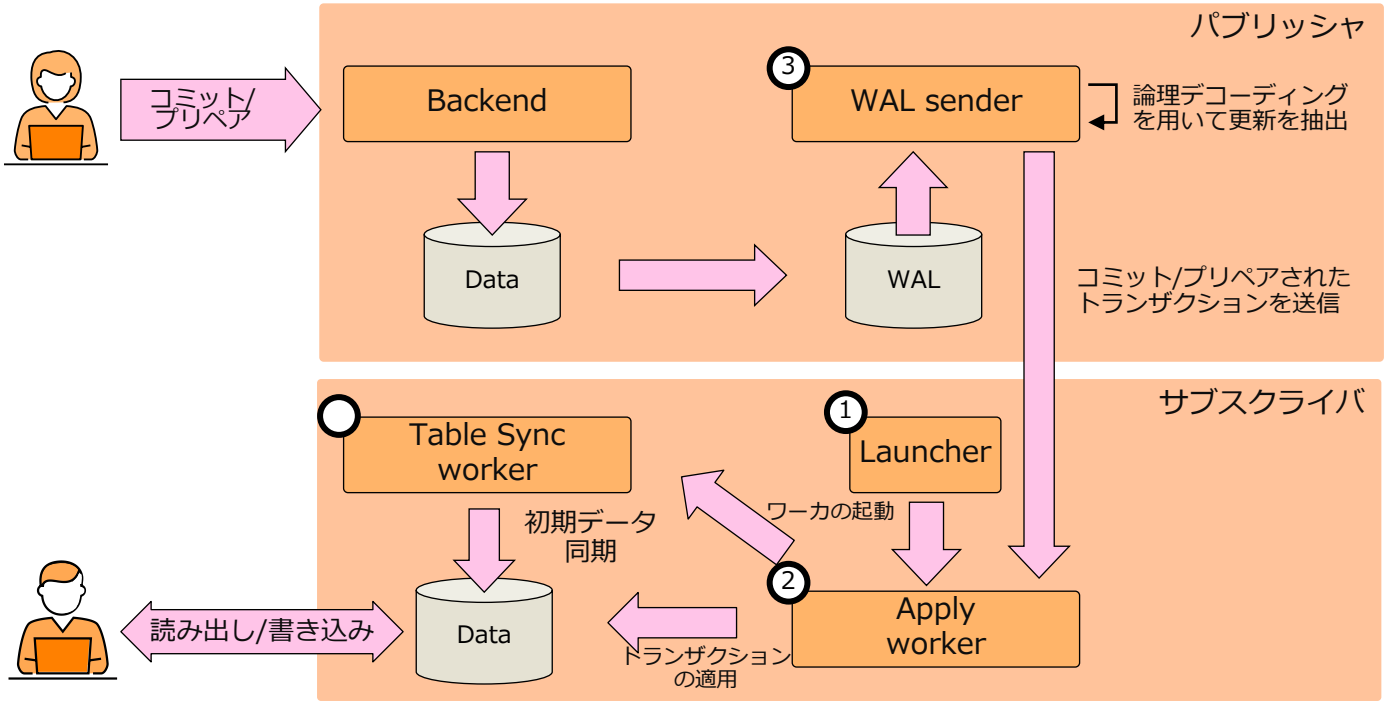
論理デコーディングのインターフェース

FUJITSU

- SQL関数
  - ……前スライドまでの例
  - バックエンドプロセスがWALを読み出す
  - 更新内容は関数の返り値として出力される
- レプリケーションプロトコル
  - ……論理レプリケーションの基盤
  - WAL senderがWALを読み出す
  - 更新内容は随時ストリームされる

論理レプリケーション | プロセスアーキテクチャ

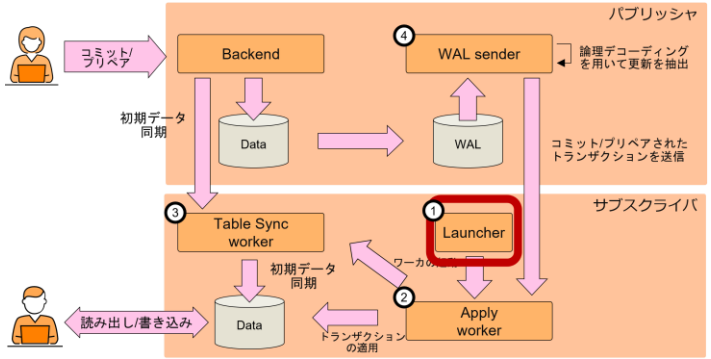
FUJITSU



Launcher

- サーバプロセス起動と同時に起動するバックグラウンドワーカ
- 定期的にpg_subscriptionを確認し、各行に対応するapply workerを生成

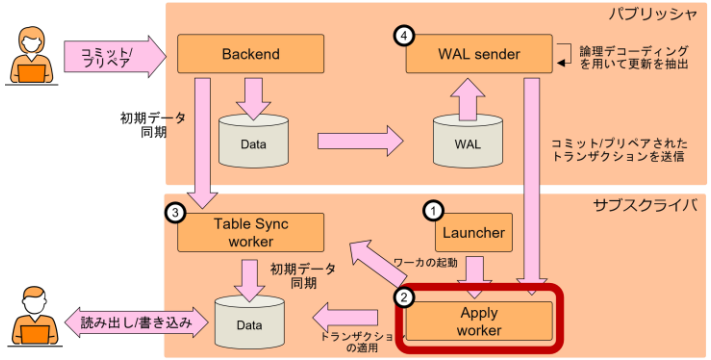
```
postgres=# SELECT datid, pid, backend_type FROM pg_stat_activity
           WHERE backend_type LIKE 'logical%';
 datid |  pid  |         backend_type
-------+-------+------------------------------
       | 22427 | logical replication launcher
(1 row)
```



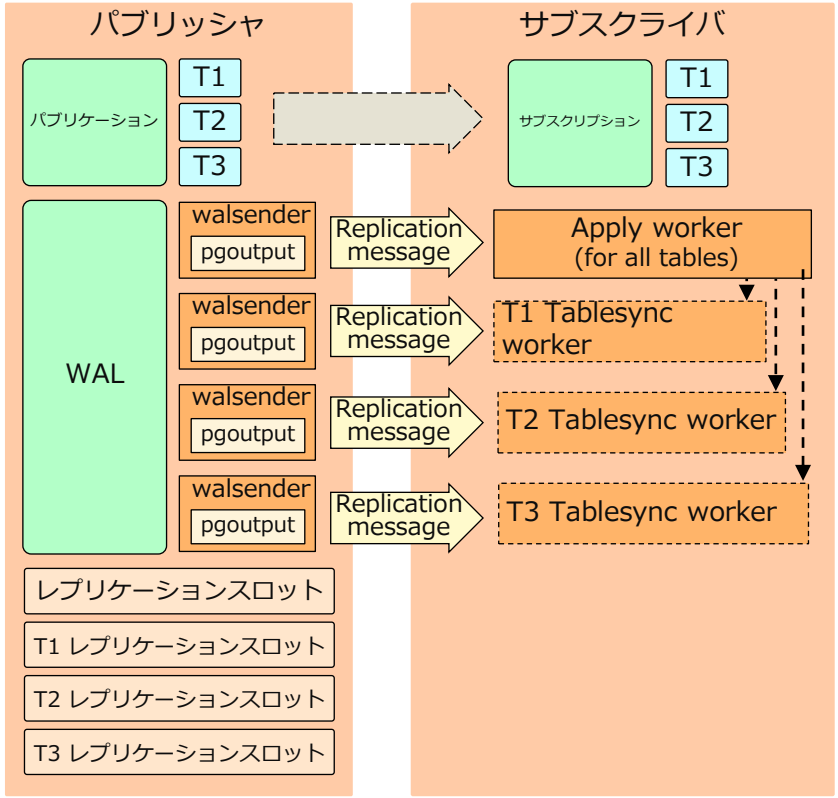
Apply worker

- サブスクライバ側にのみ存在するバックグラウンドワーカ
- pg_subscriptionの毎行に1つ存在
- パブリッシャに接続し、WAL senderからの出力内容を受け取って適用する
  - パブリッシャ上に論理レプリケーションスロットを作成し、そこから論理デコーディングを用いて更新内容を抽出する
- pg_stat_subscriptionビューでワーカの状態を参照可能

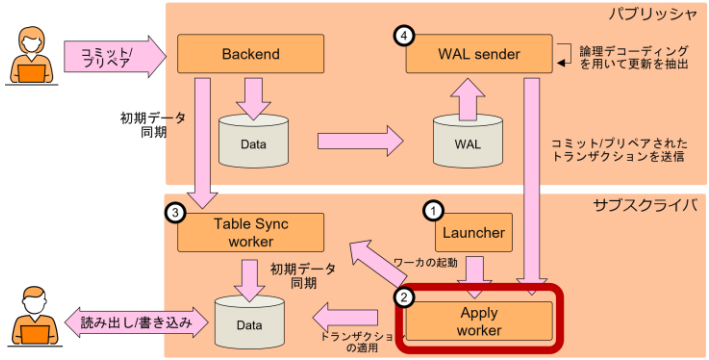
```
postgres=# SELECT subname, received_lsn
           FROM pg_stat_subscription;
 subname | received_lsn
---------+--------------
 sub     | 0/1546788
(1 row)
```



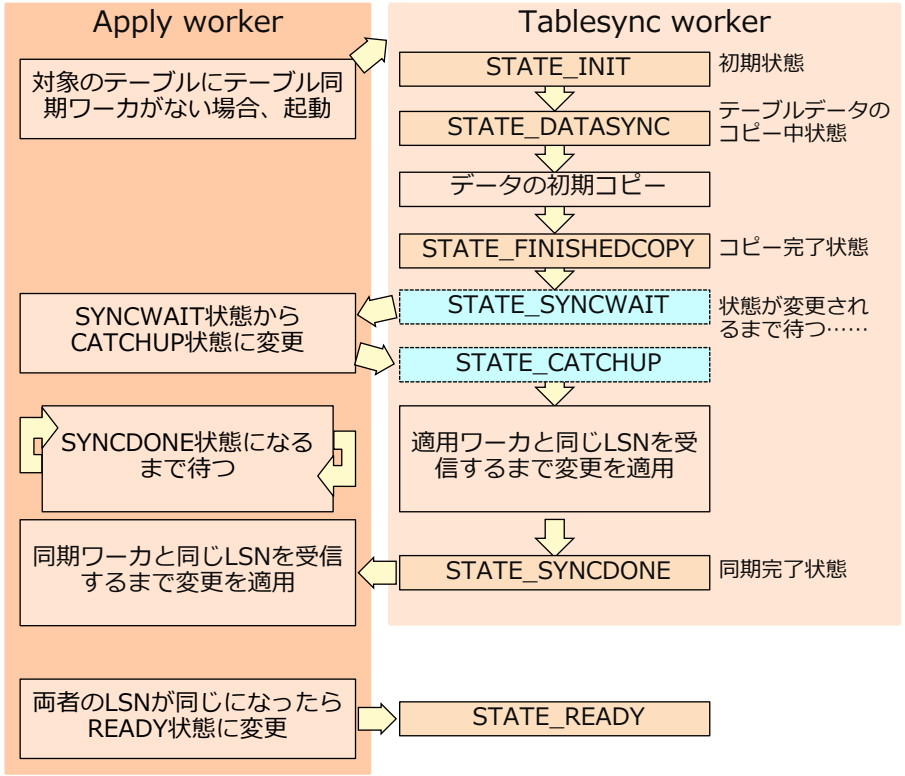
Apply worker



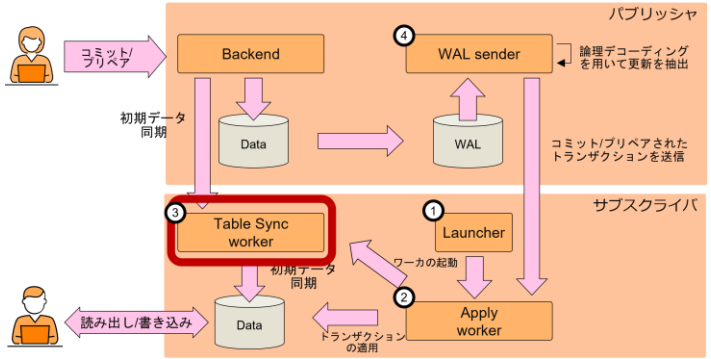
- 初期データ同期時には、対象となったテーブル毎にTablesync workerを起動する
- それぞれのTablesync workerがレプリケーションスロットを作成し、対応するWAL senderから更新を受け取って適用する



Tablesync worker



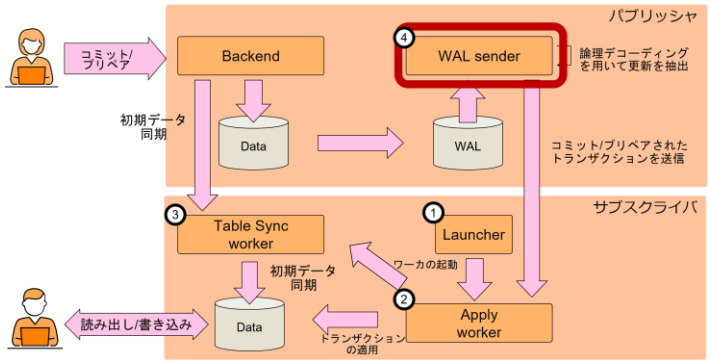
- サブスクライバ側にのみ存在するバックグラウンドワーカ
- テーブル毎に生成される
- COPY文による初期コピーと、ストリーミングプロトコルによる更新の受信・適用の2つのフェーズからなる



WAL sender

- パブリッシャ側にのみ存在するバックエンドプロセス
- 適用ワーカ/同期ワーカの要求に応じて起動
- 論理デコーディングの仕組みを用いてWALをデコードし、出力内容をワーカに送信し続ける
- pg_stat_replication_slotsビューでスロットの使用状況を参照可能

```
postgres=# SELECT slot_name, total_txns, total_bytes
           FROM pg_stat_replication_slots;
 slot_name | total_txns | total_bytes
-----------+------------+-------------
 sub       |         56 |        7392
(1 row)
```



目次

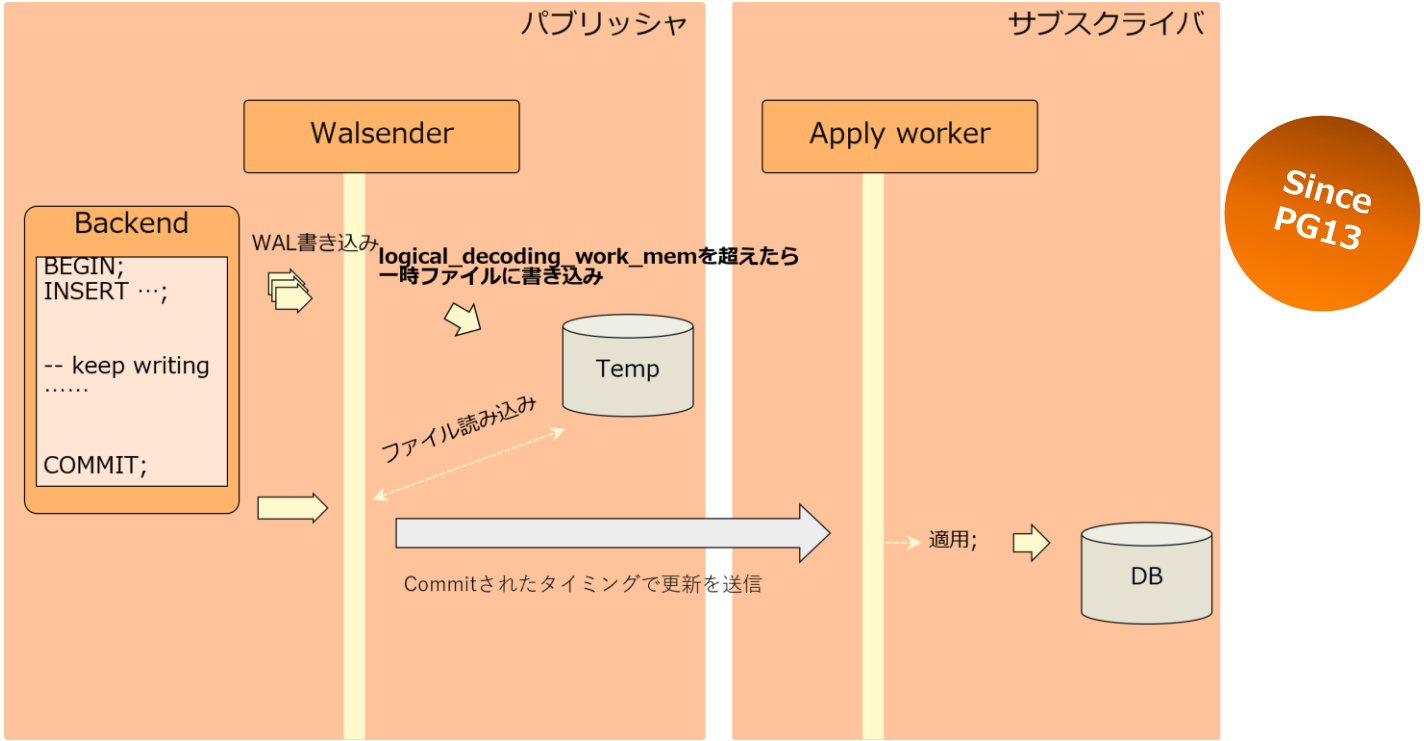
- 論理レプリケーション概論
  - レプリケーション構成の組み方
  - 注意点・制限事項と回避策
- 論理レプリケーションの中身
- 大規模トランザクションに対する論理レプリケーションのアップデート
- PG17に向けて開発中の機能

大規模トランザクションに対する論理レプリケーションのアップデート

大規模トランザクションの取り扱い in PG13

FUJITSU

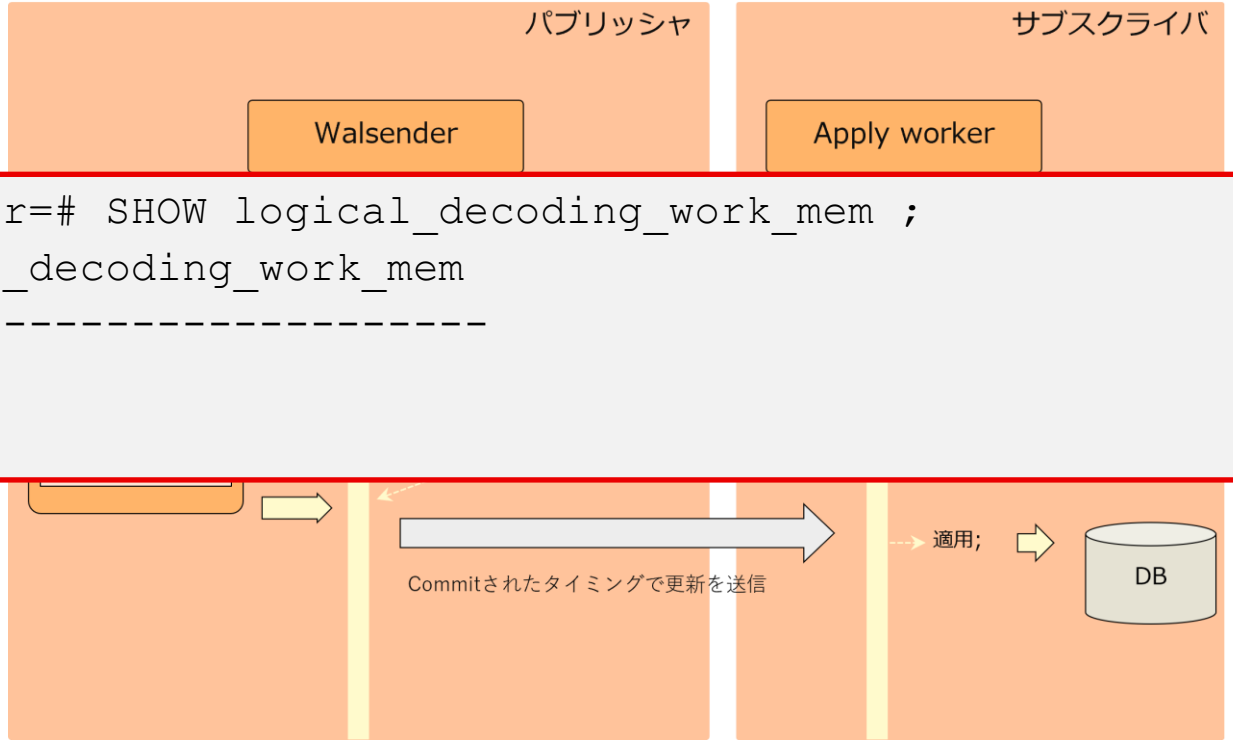
- logical_decoding_work_memを超えた場合、更新を一時ファイルに書き込む



大規模トランザクションの取り扱い in PG13

FUJITSU

- logical_decoding_work_memを超えた場合、更新を一時ファイルに書き込む

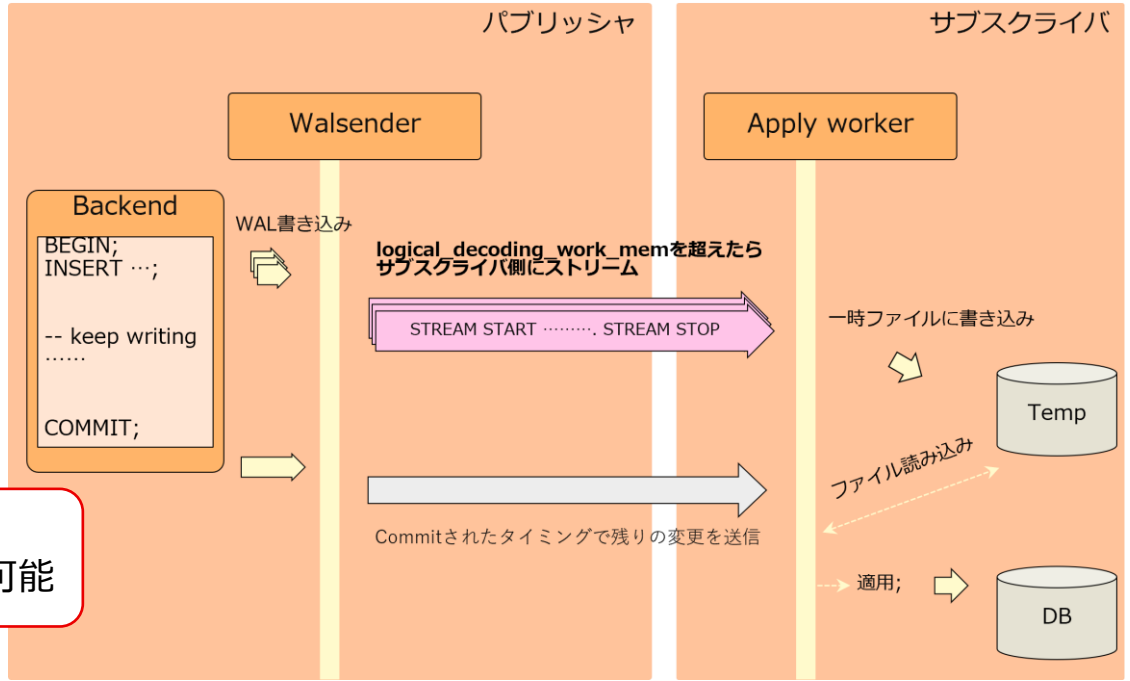



```
publisher=# SHOW logical_decoding_work_mem ;
 logical_decoding_work_mem
---------------------------
 64MB
(1 row)
```

大規模トランザクションの取り扱い in PG14

FUJITSU

- 実行中の大規模トランザクションをストリームできるようになった
- 適用完了までのレイテンシが減少



Since PG14

デフォルトで有効
無効にすることも可能

大規模トランザクションの取り扱い in PG14

FUJITSU

- 実行中の大規模トランザクションをストリームできるようになった
- 通

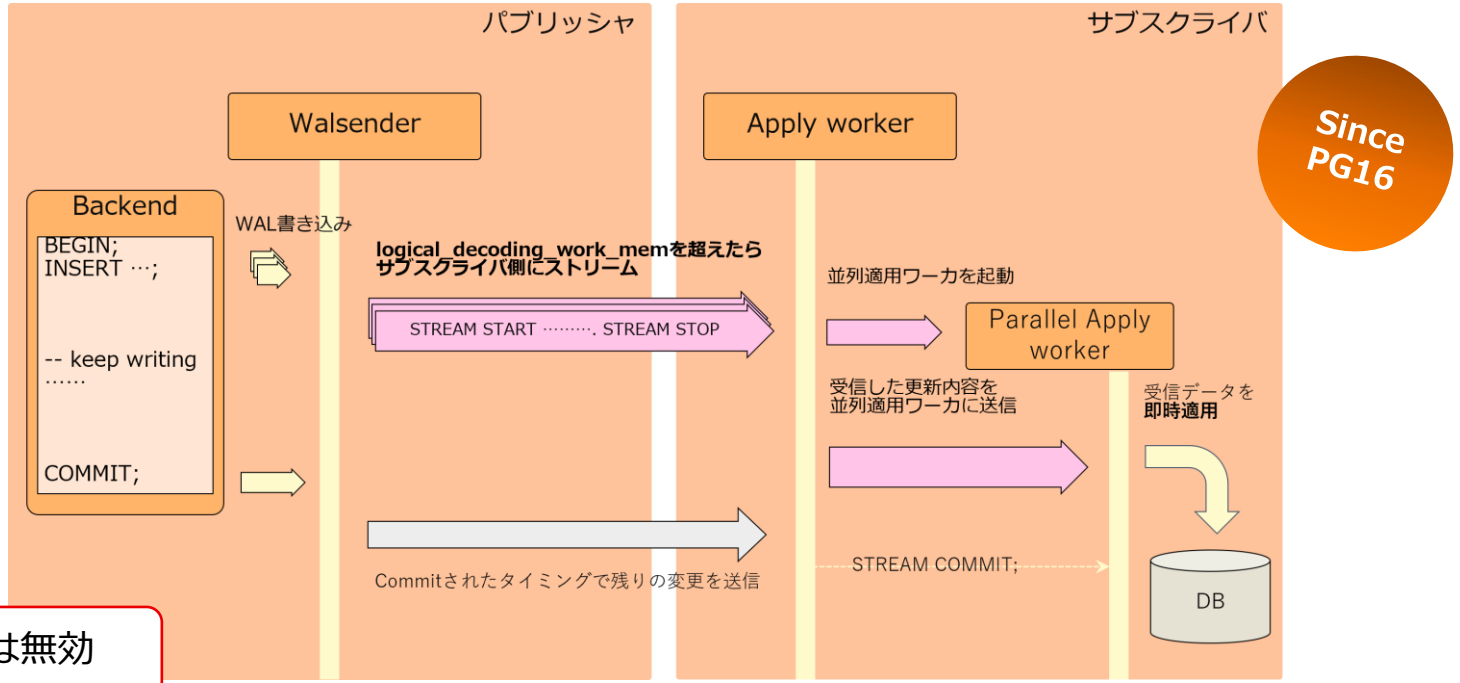
```
subscriber=# CREATE SUBSCRIPTION stream
             CONNECTION 'user=postgres host=localhost port=5431'
             PUBLICATION pub WITH (streaming = 'on');
NOTICE:  created replication slot "stream" on publisher
CREATE SUBSCRIPTION

subscriber=# SELECT subname, substream FROM pg_subscription;
 subname | substream
---------+-----------
 stream  | t
(1 row)
```

デ
無

大規模トランザクションの取り扱い in PG16

FUJITSU

- 実行中の大規模トランザクションを並列適用できるようになった
- 適用完了までのレイテンシがさらに減少



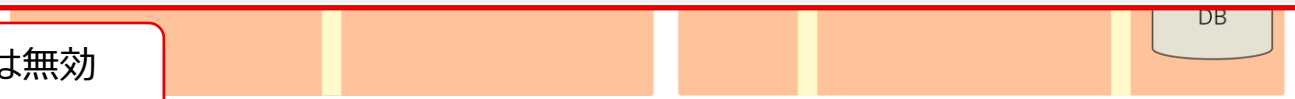
デフォルトでは無効

大規模トランザクションの取り扱い in PG16

FUJITSU

- 実行中の大規模トランザクションを並列適用できるようになった
- 適

```
subscriber=# CREATE SUBSCRIPTION parallel
             CONNECTION 'user=postgres host=localhost port=5431'
             PUBLICATION pub WITH (streaming = 'parallel');
NOTICE:  created replication slot "parallel" on publisher
CREATE SUBSCRIPTION

subscriber=# SELECT subname, substream FROM pg_subscription;
 subname  | substream
----------+-----------
 parallel | p
(1 row)
```

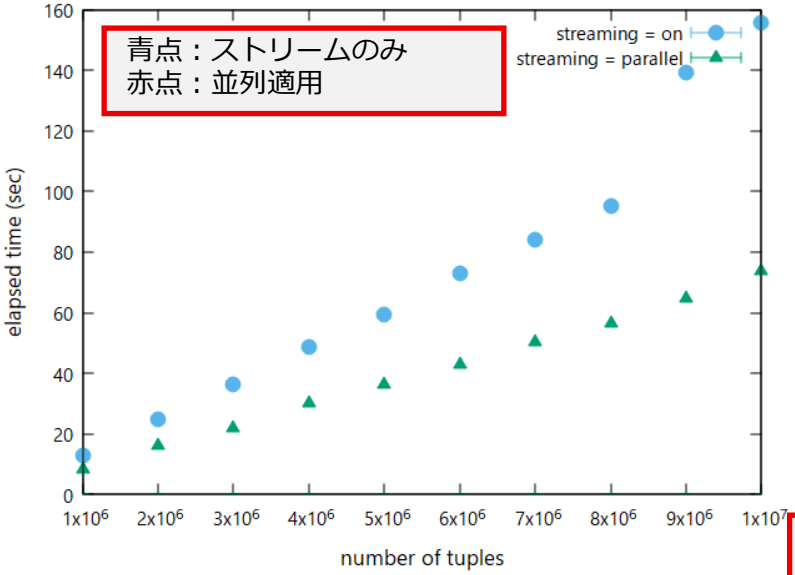


デフォルトでは無効

性能測定 – ストリームのみと並列適用の比較

FUJITSU

- 大規模トランザクションの適用に掛かる時間を測定・比較
- ストリームのみと比較して、並列適用は**20%以上の高速化に成功**




```
shared_buffers = 100GB
Checkpoint_timeout = 30min
max_wal_size = 20GB
min_wal_size = 10GB
autovacuum = off
CREATE TABLE large_test (
      id INTEGER PRIMARY KEY,
      num1 BIGINT,
      num2 DOUBLE PRESICION,
      num3 DOUBLE PRESICION
);
```

```
¥timing
INSERT INTO large_test (id, num1, num2, num3)
       SELECT i, round(random()*10), random(), random()*142
       FROM generate_series(1, 5000000) s(i);
```

PG17に向けて開発中の機能

PG17に向けて開発中の機能（抜粋）

FUJITSU

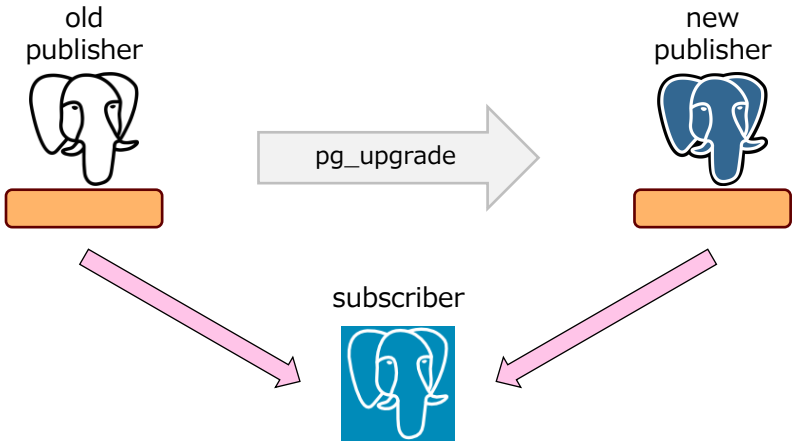
- **論理レプリケーションスロットのpg_upgradeサポート**
- DDL レプリケーション
- 初期スキーマ同期
- シーケンスの論理デコーディングサポート
- スロット同期ワーカの導入

論理レプリケーションスロットのpg_upgradeサポート

FUJITSU

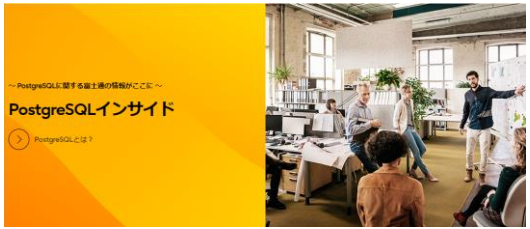
- 現状、レプリケーションスロットはpg_upgradeの対象外
  - スロットにはLSNのようなノード固有の値が含まれており、単純なファイルコピーはNG
- そのため、アップグレード直後に論理レプリケーションの再開はできなかった

- pg_upgradeからSQL関数を実行することで、論理スロットの復元を実現
- masterブランチにpush済、このままいけばPG17から利用可能
  - Authorは黒田です



まとめ

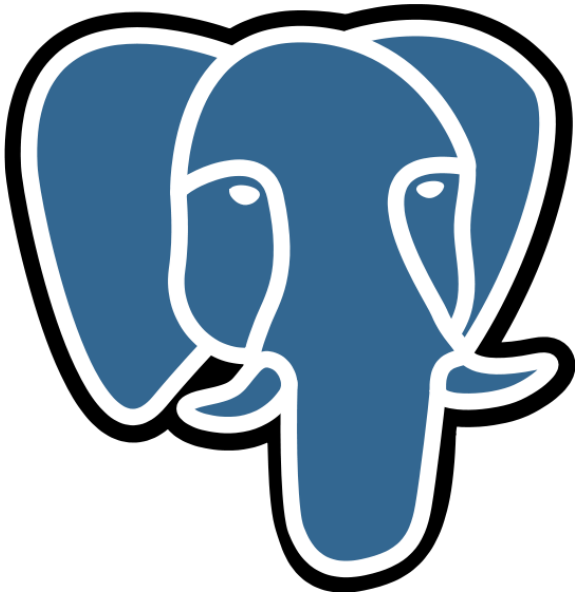
- 論理レプリケーションとはデータと変更を複製する手法
- 単なるバックアップだけではない、様々な活用方法が存在する
- 導入以来活発なアップデートが行われている
- 本日話した内容は弊社のPostgreSQL情報サイトにも掲載



- 技術者Blog：PostgreSQL16とその後
- PostgreSQL技術インデックス
- Inside of logical replication in PostgreSQL: How it works

興味を持った方は

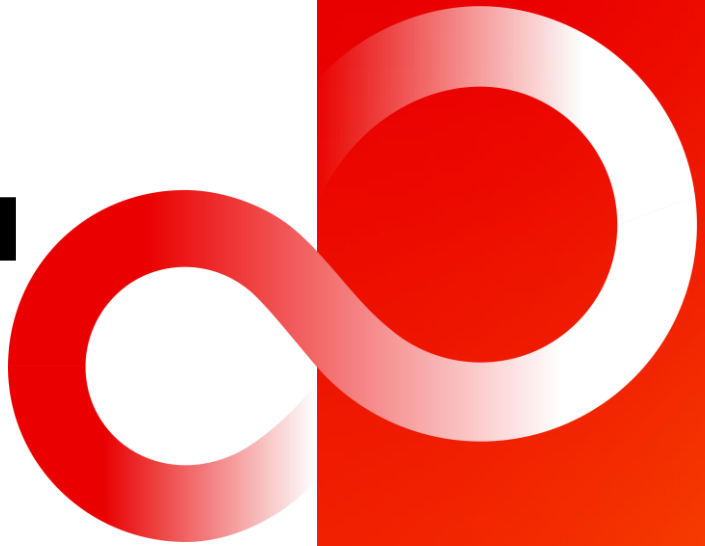
FUJITSU



富士通と一緒にコミュニティで
論理レプリケーションを開発しませんか？

pgsql-hackers

Thank you



FUJITSU