PostgreSQL 17 最新情報セミナー

2024-10-25

株式会社SRA OSS 高塚 遥

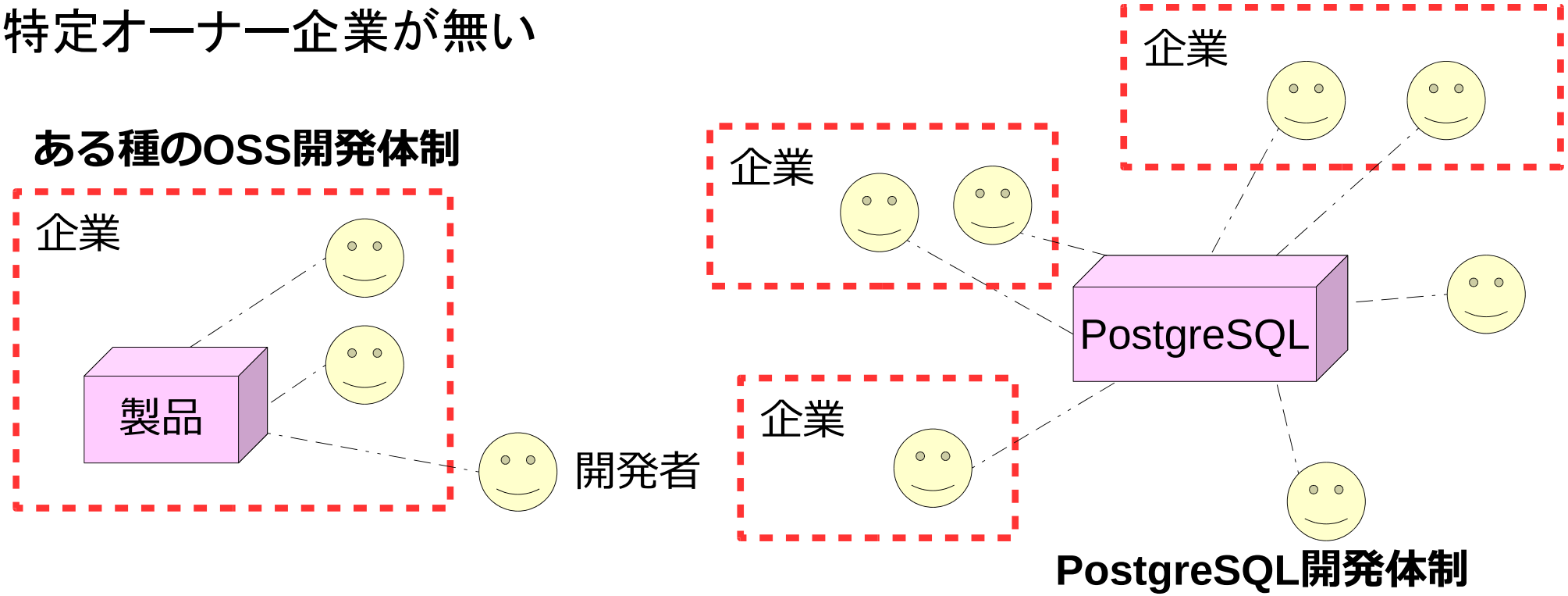
講演内容 / 講演者

- 講演内容
  - PostgreSQL概要
  - PostgreSQL 17 の新機能、非互換変更

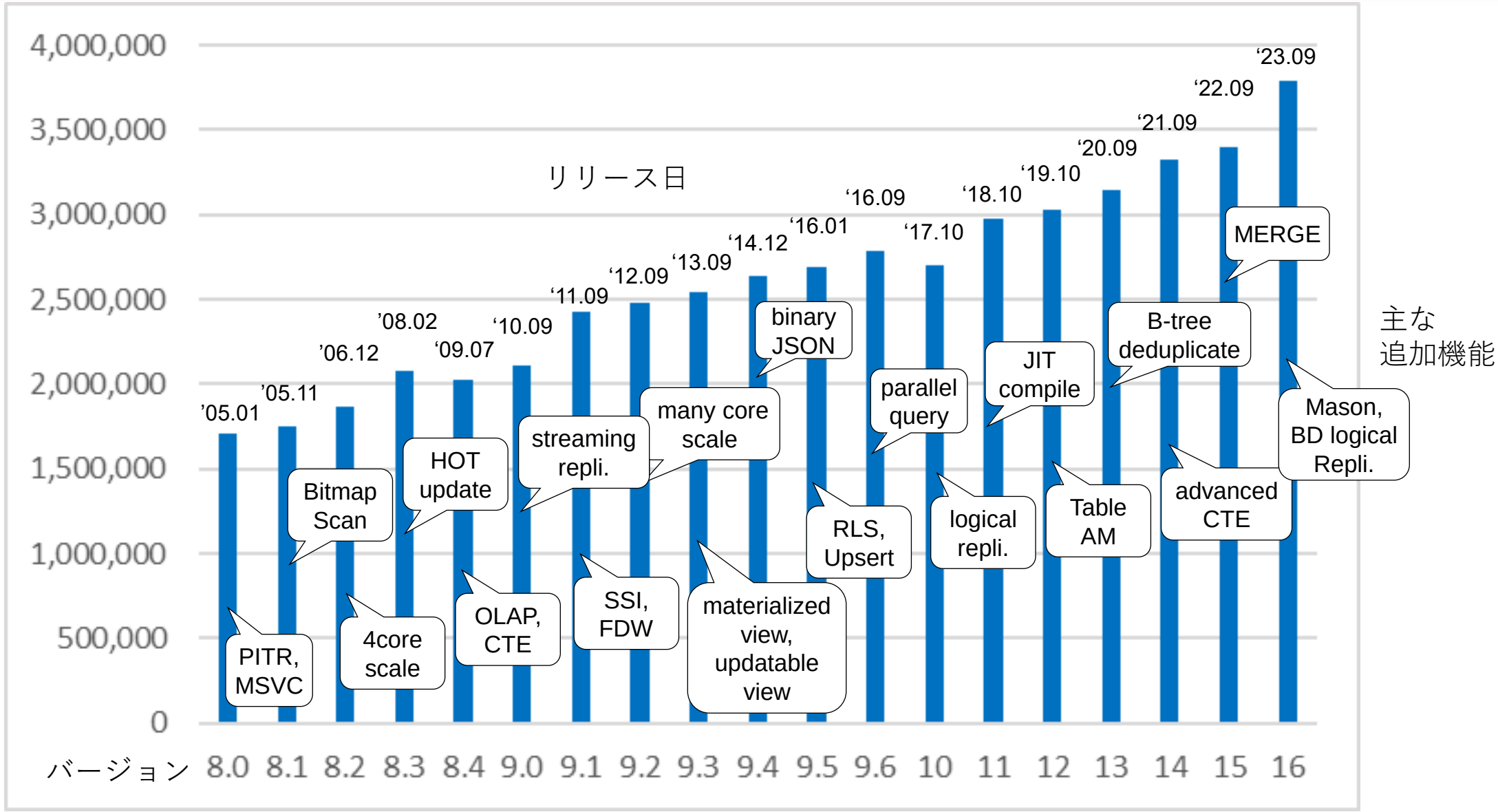
- 講演者
  - 株式会社 SRA OSS 高塚 遥
  - PostgreSQL のヘルプデスク、コンサルティング、設計構築、トレーナー等を１５年以上にわたり担当
  - 特定非営利活動法人 日本PostgreSQLユーザ会 理事

PostgreSQL とは

- 多機能、高性能、かつオープンソースの RDBMS
  - 伝統的な RDBMS設計
  - INGRES(‘70)、POSTGRES(‘80) から続く長い歴史
  - BSDタイプのライセンス
  - 特定オーナー企業が無い



PostgreSQLのリリース履歴



現代における PostgreSQLの実力

強み

- 各種用途での圧倒的な実績
  - 各種ソフトウェアからの対応
- 安定した開発体制
- SQL標準準拠の方針
- 様々な互換製品、拡張製品

弱み

- 伝統的な設計による性能限界
  - ディスク主体、少CPU・小メモリ想定
- 保守的な開発方針
- 小さい本体と外部機能依存
  - 高可用性
  - バックアップ管理
  - 監査ログ、プランナヒント、監視 ・・・

PostgreSQL 17 新機能

- 性能向上
  - VACUUM性能改善
  - WAL排他処理の改善
  - COPY高速化、ストリームI/O
  - 各種プランナ改善
- SQL機能
  - SQL/JSON 対応拡充
  - MERGE文の拡張
  - COPY に ON_ERROR 指定
- パーティショニング
  - 排他制約対応
- ロジカルレプリケーション
  - pg_createsubscriber
  - フェイルオーバ対応
  - pg_upgrade対応
- 運用管理
  - インクリメンタルバックアップ
  - pg_dump/pg_restore --filter
  - 新モニタリングビュー
  - MAINTAIN権限と pg_maintainロール

本講演で取り上げるもの
以外にも多数あります

PostgreSQL 17 新機能

性能向上

VACUUM性能改善(1)

- 大テーブルVACUUM – 課題点と改善
  - maintenance_work_mem が 1GB までしか使われない
    - デッドタプルを記憶する領域不足だと、無駄な処理ステップ繰り返しが発生

    → 使用メモリサイズ上限を撤廃

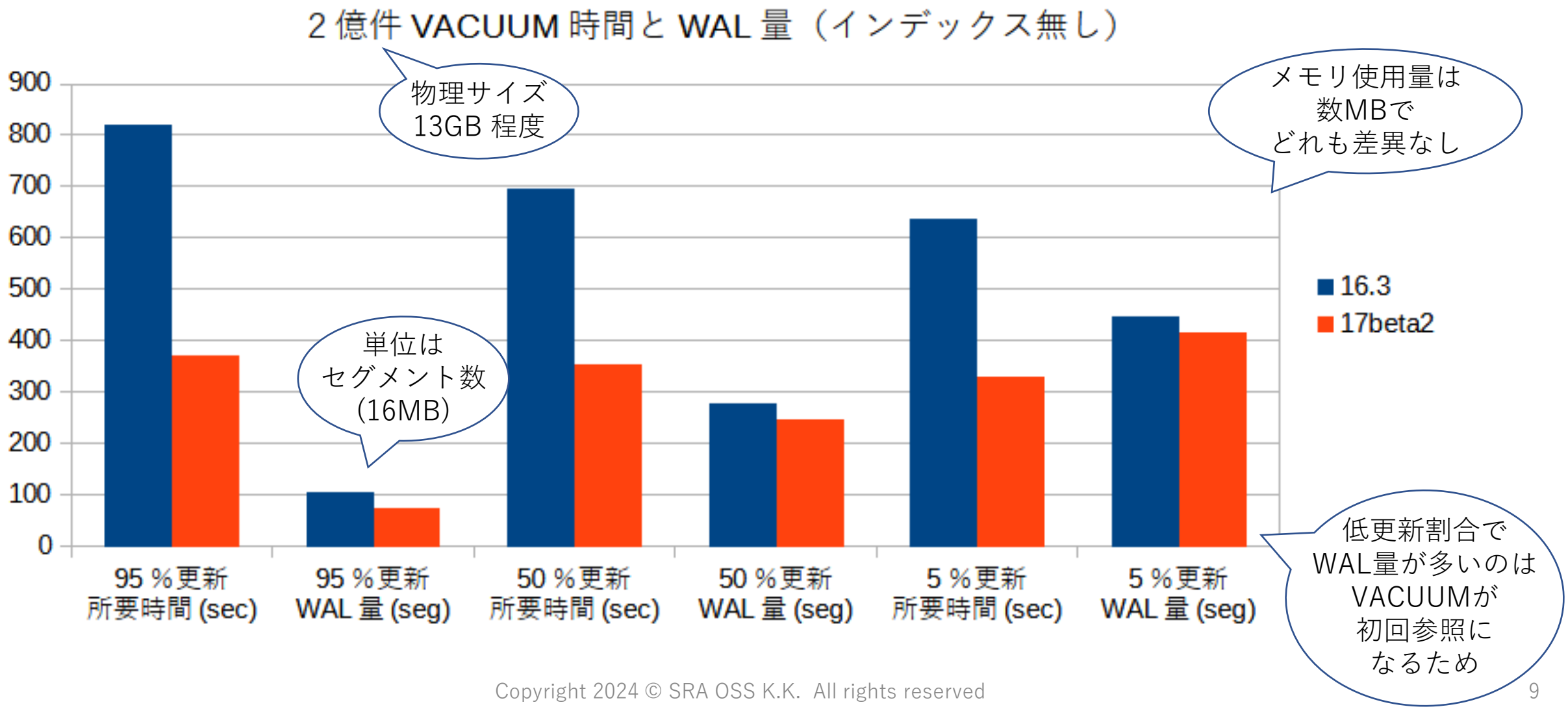
  - デッドタプルを記憶するデータ構造が単純配列で効率的でない

    → Adaptive Radix Tree データ構造を導入
    必要な領域サイズも節減

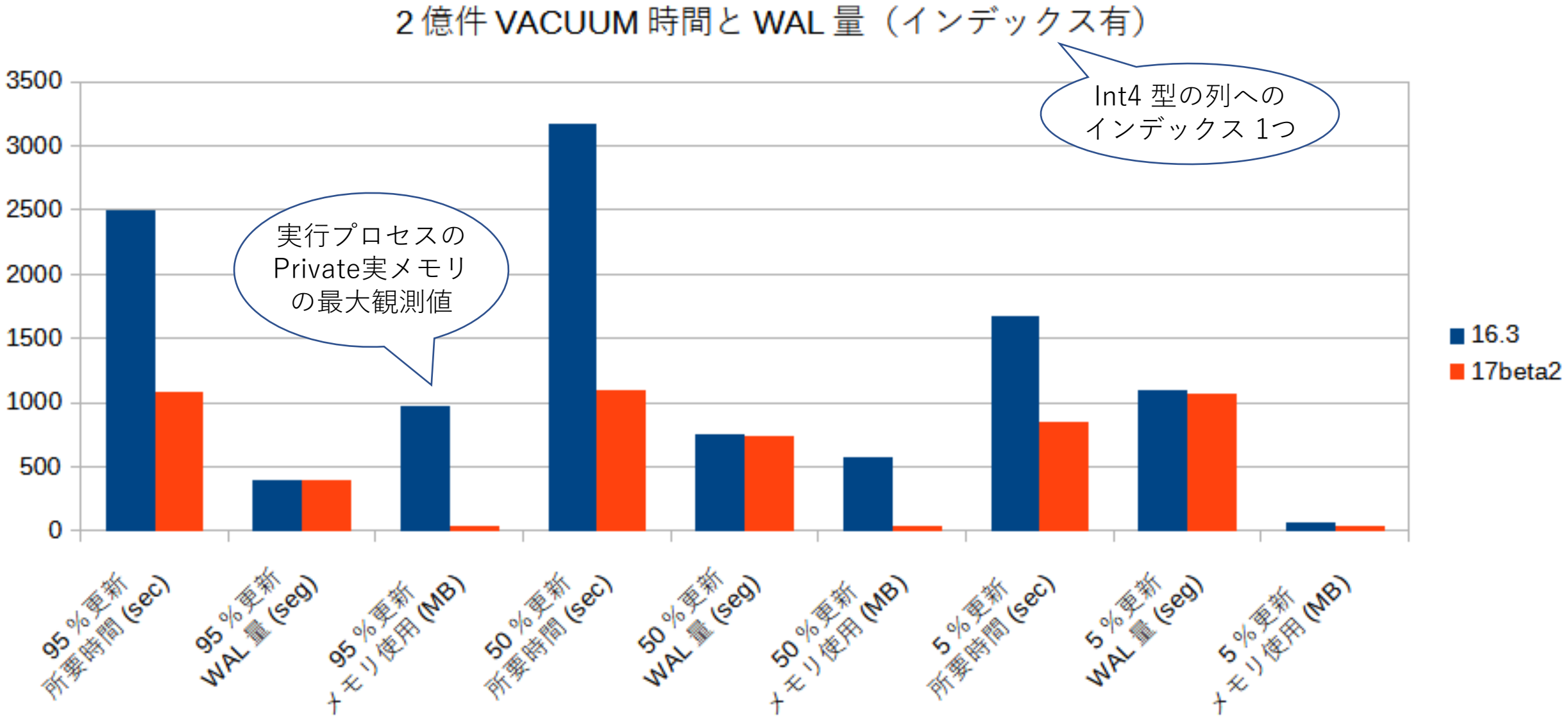
  - それ以外に:

    → インデックス無しテーブルの場合にWAL出力量を節減する改修

VACUUM性能改善(2)



VACUUM性能改善(3)

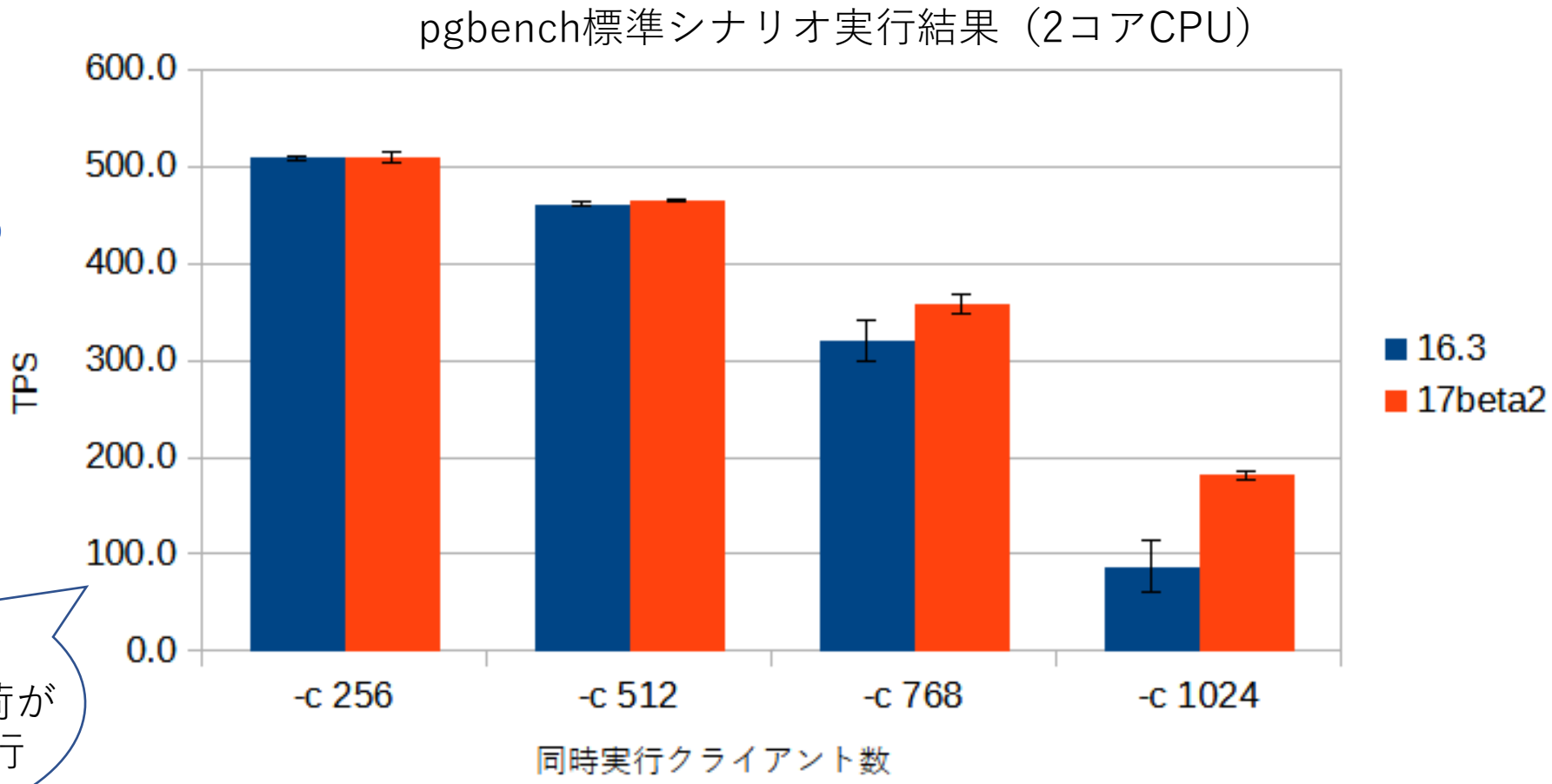


WAL排他処理の改善

- 更新の同時実行処理で性能改善
  - アトミックな値交換CPU命令を採用

→ 多接続での
性能ダウンが
緩和されている

→ 少コア数の
サーバでも
効果あり

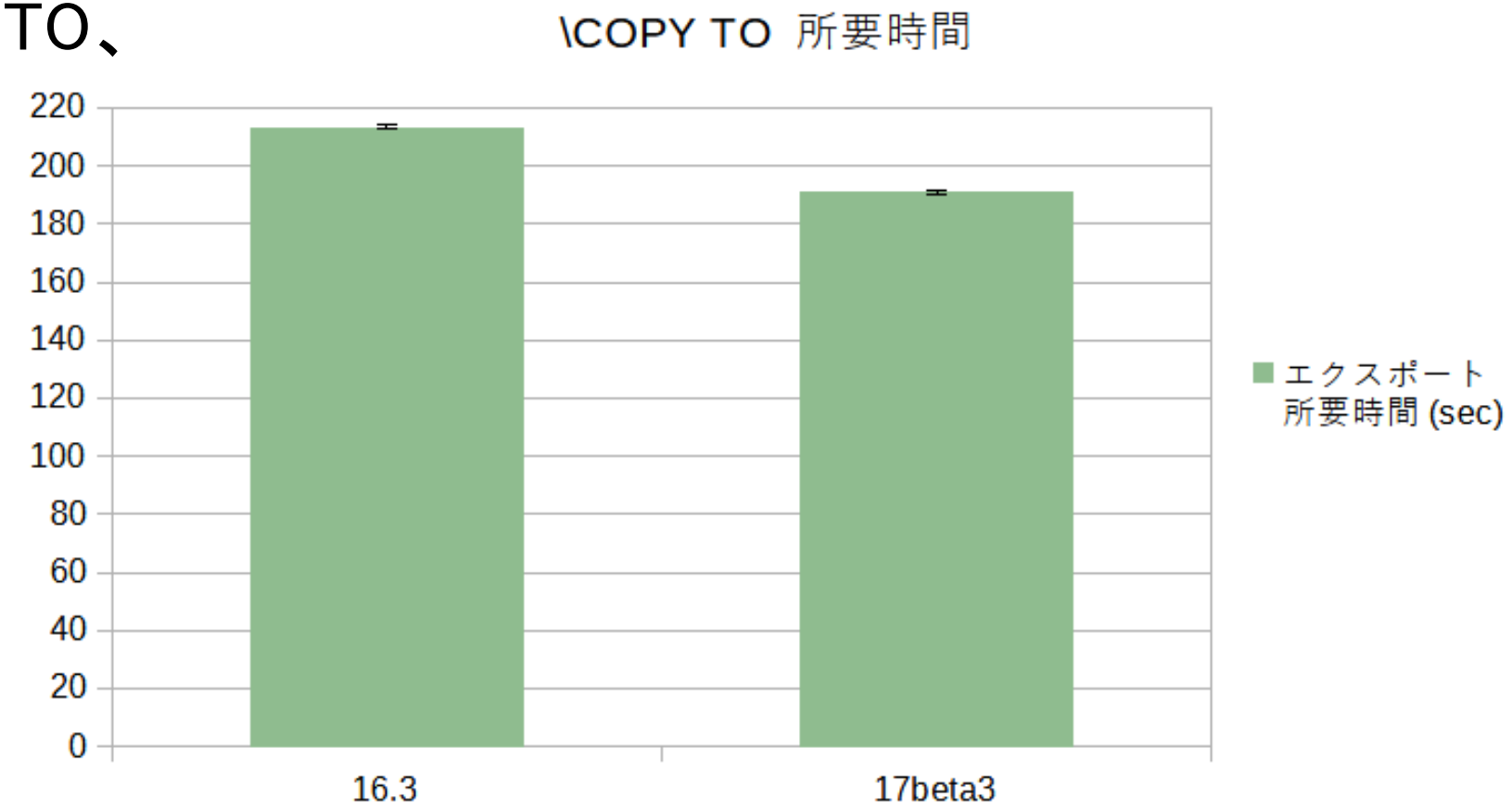


WAL以外の
ストレージI/O負荷が
小さい構成で実行

COPYなどエクスポート高速化

- データ送出処理を最適化

- 1行が大きい COPY TO、
  pg_basebackup
  で性能向上



1行 30KB弱 × 100万行
のテーブルを
バッファに載った状態から
psql ¥COPY TO 実行
（結果は読み捨て）

ストリーム I/O対応

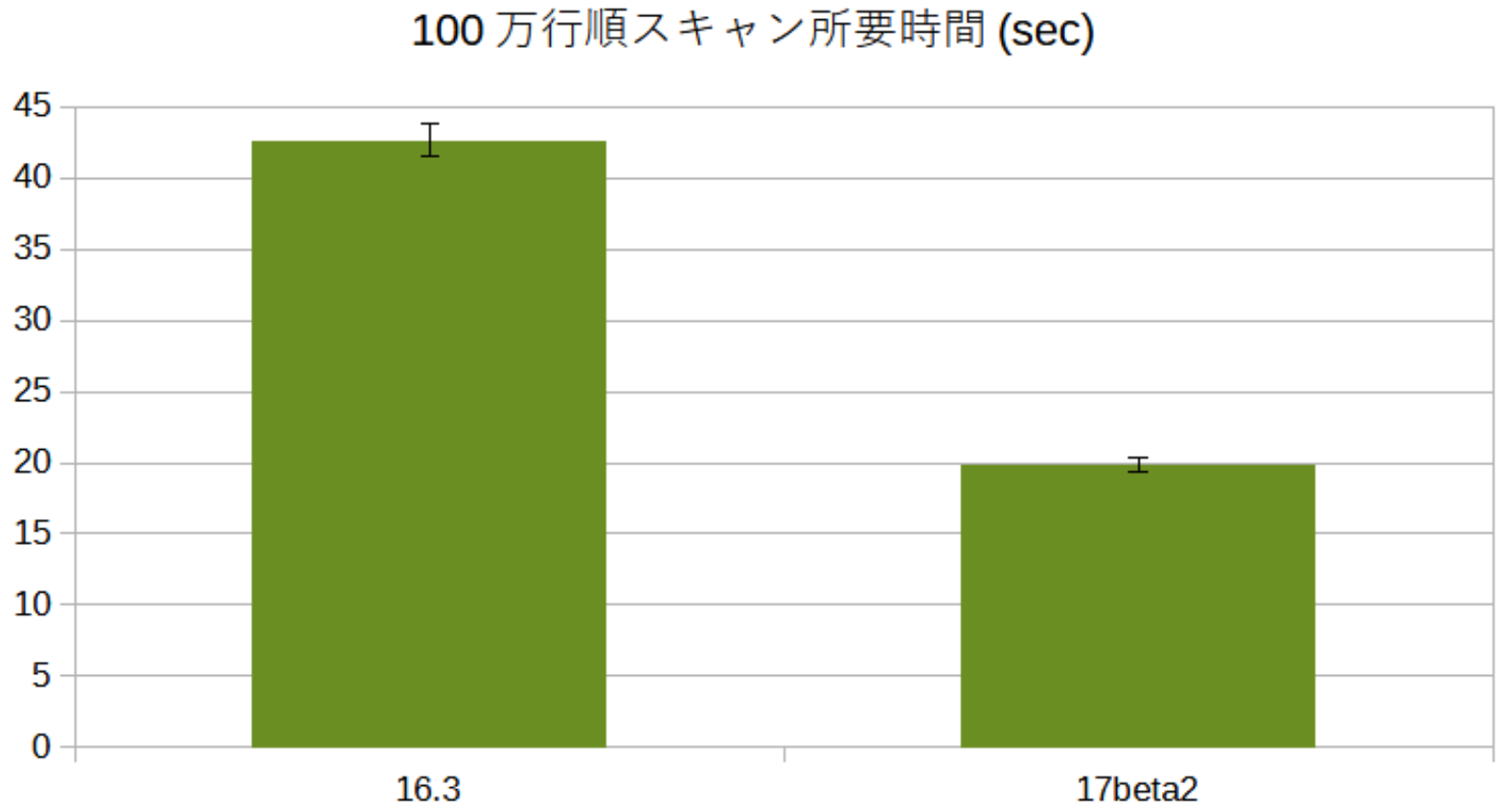
- 連続ブロック読み取りに適した Linuxシステムコールを利用

- ダンプ取得など、大サイズを順スキャンするときに効果

- 右グラフは

  SELECT count(1)
  FROM tbl1m

  の所要時間



キャッシュヒットしない状況を作って試験している

各種プランナ/エグゼキュータ 改善

- 「... WHERE i1 IN (...) AND i2 IN (...) 」で Btree利用を最適化
  - (i1, i2) という複合インデックスがある場合
- WITH句の CTEプラン結果への無駄な処理を除く
  - CTE結果の特性(「ソート済み」「ユニーク」など)を活用
- IN句相関サブクエリを Subplan ではなく Join に展開
- 範囲データとの @> 演算に Btreeインデックスを適用
  - 範囲データ型のプランナ統計情報が拡充
- GROUP BY a, b, c を インデックスやORDER BY 列順に合わせる
- NOT NULL制約を活かした最適化

PostgreSQL 17 新機能

SQL機能

SQL/JSON 対応拡充(1)

- SQL/JSON 標準に準じた関数がいくつか追加
  - JSON_TABLE – jsonデータ内容を表に転換

全体、および、列ごと、
JSON から表として取り出す
箇所を jsonpath で指定

（jsonb 列の内容を JSON_TABLE で表に展開）

```
db1=# SELECT id, jt.* FROM t_ext, LATERAL JSON_TABLE(js, '$.basic' COLUMNS (
        name text PATH '$.name' ,
        main_authors text[] PATH '$."main-authors"',
        "1st_rel" date PATH '$."first-release"')) as jt;
 id |      name      |                  main_authors                  |  1st_rel
----+----------------+------------------------------------------------+------------
  1 | pg_store_plans | {"Kyotaro Horiguchi"}                          | 2015-10-09
  2 | pgaudit        | {"David Steele"}                               | 2015-12-12
  3 | pg_repack      | {"Daniele Varrazzo","Josh Kupershmidt"}        | 2008-12-08
(3 rows)
```

SQL/JSON 対応拡充(2)

```
db1=# CREATE TABLE t_ext(id serial, js jsonb);
db1=# INSERT INTO t_ext (js) VALUES
      ($$ { "basic": {
         "name": "pg_store_plans",
         "repository": "https://github.com/ossc-db/pg_store_plans",
         "first-release": "2015-10-09",
         "main-authors": ["Kyotaro Horiguchi"]}}$$),
      ($$ { "basic": {
         "name": "pgaudit",
         "repository": "https://github.com/pgaudit/pgaudit",
         "website": "https://www.pgaudit.org",
         "first-release": "2015-12-12",
         "main-authors": ["David Steele"]}}$$),
      ($$ { "basic": {
         "name": "pg_repack",
```

前ページ
JSON_TABLE 使用例：
t_ext テーブルの
JSONデータ

SQL/JSON 対応拡充(2)

- JSON_EXISTS – jsonpath にて該当する要素があるか問い合わせ
- JSON_QUERY – jsonpath 問い合わせ結果を取り出し
- JSON_VALUE – JSON要素を SQLのデータ型の値に変換して取り出し
- JSON_SCALAR – SQLのスカラ値からJSONデータを構成
- JSON_SERIALIZE – JSONデータをバイナリ列などに変換

```
db1=# SELECT json_serialize('{"A":100, "B":200} '::json RETURNING bytea);
           json_serialize
----------------------------------------
 \x7b2241223a3130302c202242223a3230307d20
(1 row)
```

空白文字も正規化されず
単純にバイナリ化
されるだけ

SQL/JSON 対応拡充(3)

- jsonpath のメソッド追加
  - 値の精度調整： .decimal(precision, scale)、
  - 比較式のためのデータ型指定:
    - .bigint()、.boolean()、.date()、.integer()、 .number()、.string()、.time()、.time_tz()、.timestamp()、.timestamp_tz()

```
db1=# SELECT JSON_QUERY('{"a": "123.45"}', '$.a.decimal(3, -1)');
 json_query
------------
 120
(1 row)
```

```
db1=# SELECT id, JSON_QUERY(js, '$.basic.name') FROM t_ext
        WHERE JSON_EXISTS (js,
            '$.basic."first-release" ? (@.date() < "2010-01-01".date())');
```

MERGE文の拡張(1)

- ターゲットに更新可能ビューを指定可能に
- RETURNING句に対応
  - merge_action() 関数で操作種別も取り出し

```
db1=# MERGE INTO t_tgt t USING t_src s ON s.id = t.id
        WHEN MATCHED THEN UPDATE SET v = s.v
        WHEN NOT MATCHED THEN INSERT (id, v) VALUES (s.id, s.v)
        RETURNING merge_action(), t.* ;
 merge_action | id |       v
--------------+----+---------------
 UPDATE       |  1 | data1 updated
 INSERT       |  3 | data3
(2 rows)
```

MERGE文の拡張(2)

- ソース側の行が無い場合のアクションを記述可能に
  - WHEN NOT MATCHED [BY TARGET]:
    → ソース側の行に対応するターゲット側の行が無い場合（従来から対応）
  - WHEN NOT MATCHED BY SOURCE
    → ターゲット側の行に対応するソース側の行が無い場合

```
db1=# MERGE INTO t_tgt t USING t_src s ON s.id = t.id
        WHEN NOT MATCHED BY SOURCE THEN DELETE
        WHEN NOT MATCHED THEN INSERT (id, v) VALUES (s.id, s.v)
        RETURNING merge_action(), t.* ;
```

t_src に無く
t_tgt にある行を
t_tgt から削除

t_tgt に無く
t_src にある行を
t_tgt に挿入

COPY FROM の ON_ERROR オプション

- これまで COPY FROM のエラー時は全て取消し
  - 大きいテーブルのデータ読み込みで破損個所があると対応しにくい

→ ON_ERROR ignore 指定でエラー行をスキップして継続

特に破損が
複数ある場合

  - スキップしたデータ行を把握するには？

→ LOG_VERBOSITY verbose 指定でスキップ行を報告

  - スキップ対象は入力行のデータが壊れている場合のみ
    - 列データが当該列のデータ型で受け入れられる入力文字列になっていない等

データ書式としては正しいが制約違反でテーブルに投入できない場合：

→ スキップされず、従来通りエラーになって全て取り消される

PostgreSQL 17 新機能

パーティショニングと
ロジカルレプリケーション

パーティションテーブルで排他制約

（HASH によるパーティションテーブルを定義）

```
db1=# CREATE TABLE t_meeting (id int, channel int, tsr tsrange, uid int)
        PARTITION BY HASH (channel);
db1=# CREATE TABLE t_meeting_0 PARTITION OF t_meeting
        FOR VALUES WITH (MODULUS 3, REMAINDER 0);
db1=# CREATE TABLE t_meeting_1 PARTITION OF t_meeting
        FOR VALUES WITH (MODULUS 3, REMAINDER 1);
db1=# CREATE TABLE t_meeting_2 PARTITION OF t_meeting
        FOR VALUES WITH (MODULUS 3, REMAINDER 2);
```

（パーティションテーブル全体に主キー制約と排他制約を追加）

```
db1=# ALTER TABLE t_meeting ADD PRIMARY KEY (id, channel);
db1=# ALTER TABLE t_meeting
        ADD EXCLUDE USING gist (channel WITH =, tsr WITH &&);
```

チャンネルと時間範囲で
オンラインミーティングの
予定を管理するテーブル

パーティションキー
の = 条件を含めれば、
&& による排他制約を
指定できる

同チャンネルで
時間が重なったら
制約違反

ロジカルレプリケーション改善(1)

- pg_createsubscriberコマンド

物理ストリーミング
レプリケーションの
スタンバイ

↓ 変換

ロジカルレプリケーション
のサブスクライバサーバ

```
( -d 対象データベース、-P パブリケーションサーバへの接続文字列、
 -p サブスクライバポート、-D サブスクライバディレクトリ、
 --publication パブリケーション名、--subscription サブスクリプション名 )
$ pg_createsubscriber -d db1 -P "dbname=db1 user=postgres port=5433" \
    -p 5434 -D ./pgdata2 --publication=pub1 --subscription=sub1
```

- 初期同期の COPY処理を pg_basebackup に置き換えて高速化
- プライマリ(＝パブリッシャ)無停止で適用 – 実行中の変更差分も反映
- データベース内全テーブルをロジカルレプリケーションする場合むけ

ロジカルレプリケーション改善(2)

- パブリッシャのフェイルオーバに対応（切り替えは外部から駆動）

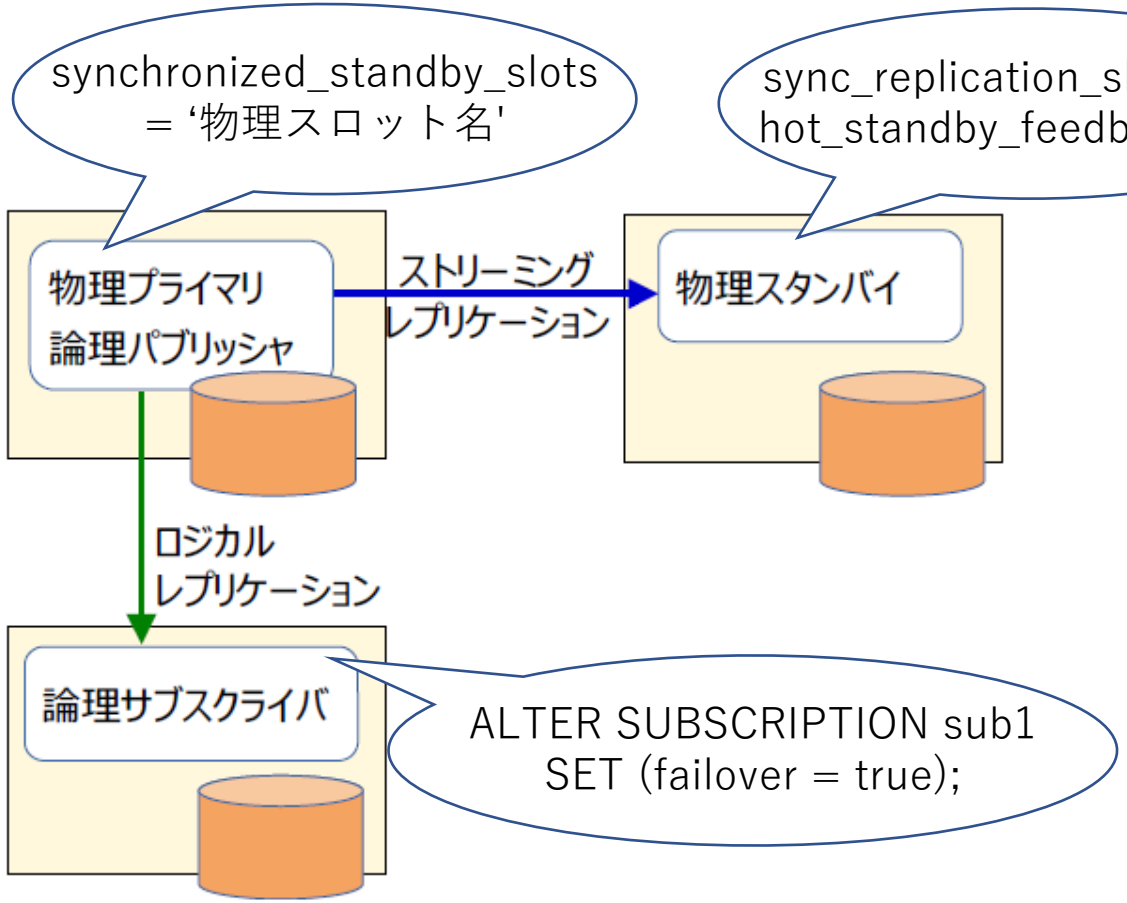


図 1: レプリケーションの組み合わせ

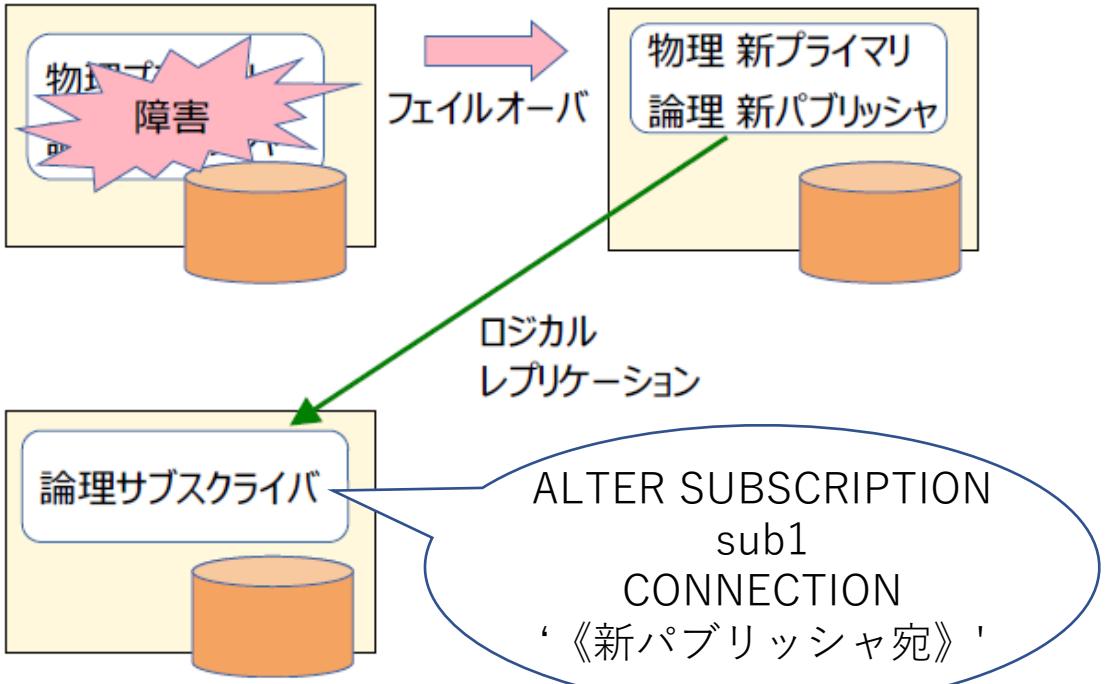


図 2: フェイルオーバした場合

ロジカルレプリケーション改善(3)

- ロジカルレプリケーションの pg_upgrade対応
  - PostgreSQL 17以降 → 18以降 で対応 ・・・ 使われるのは次バージョンから
  - パブリッシャとサブスクライバの各々に pg_upgrade をかけると、
    新バージョンでロジカルレプリケーションが継続できる

PostgreSQL 17 新機能

運用管理

インクリメンタルバックアップ(1)

- pg_basebackup で --incremental オプション
  - 前回からの変更差分のみを含むベースバックアップを取得
  - WALサマライズ機能を有効にする必要がある
- 新コマンド pg_combinebackup
  - 起点のベースバックアップと差分バックアップ群を組み合わせて、リカバリ可能な完全なベースバックアップを構成する
  - ファイルコピーを伴う: Linuxむけのコピー負荷を軽減するオプションが利用可能
    - --copy-file-range　　copy_file_range システムコールを使用
    - --clone　　Reflinkが有効なファイルシステムで利用可能（RHEL8.x の mkfs.xfs のデフォルトではない）

両者とも
カーネル 4.5以降
(RHEL8.x以降)
で利用可能

インクリメンタルバックアップ(2)

```
$ ls $PGDATA/pg_wal/summaries/
000000010000000001000028000000000010B1D38.summary
000000010000000000010B1D3800000000000152F750.summary
000000010000000000152F75000000000000152F850.summary
```

wal_summarize = on としたときの WALサマリファイル

インクリメンタルバックアップ例

```
$ ls backup_1/base/16384
1247_fsm    4157                INCREMENTAL.2650    INCREMENTAL.3380
1249_fsm    4159                INCREMENTAL.2651    INCREMENTAL.3394
1255_fsm    4163                INCREMENTAL.2652    INCREMENTAL.3394_vm
1259_fsm    4165                INCREMENTAL.2653    INCREMENTAL.3395
```

これらは、変更が無かったテーブル等のファイル

pg_dump/pg_restore の --filterオプション

- 対象に含めるもの、含めないものを、個別のオプションではなく、フィルタファイルに記述して指定する
  - 「《include か exclude》　《オブジェクトの種類》　《パターン》」を列記

```
include   table         app1.m_*
include   table         app1.tx_*
exclude   table_data    *.tx_*
```

app1スキーマの「m_」または「tx_」で始まる名前のテーブルのみをダンプ。
ただし、「tx_」で始まる名前のテーブルは、定義のみダンプでデータは除外。
「CREATE SCHEMA app1;」も含まれない。

```
exclude   schema          mng*
exclude   table           *.*test*
exclude   foreign_table   *.*
```

excludeのみの場合には、すべてを含み、指定部分のみ除外と解釈される。
「mng」で始まる名前のスキーマは除外。
「test」を含む名前のテーブルは除外。
外部テーブルは全て除外。

新たなモニタリングビュー

- pg_stat_checkpointer … pg_stat_bgwriter から分離

コマンドによる
チェックポイントも
ここに含まれる

スタンバイの
restartpoints_timed 、
restartpoints_req は、
実際の実行数と
ズレがある

```
db1=# SELECT * FROM pg_stat_checkpointer;
-[ RECORD 1 ]-------+------------------------------
num_timed           | 361      《タイムアウトによるチェックポイント実行回数》
num_requested       | 8        《WAL 量によるチェックポイント実行回数》
restartpoints_timed | 0        《タイムアウト等によるリスタートポイント予定数》
restartpoints_req   | 0        《WAL 量によるリスタートポイント要求数》
restartpoints_done  | 0        《リスタートポイント実行回数》
write_time          | 462102   《ファイル書き込みの所要時間累計（ms）》
sync_time           | 1062     《ストレージ同期の所要時間累計（ms）》
buffers_written     | 15823    《書き込みバッファ数累計（ページ）》
stats_reset         | 2024-08-13 14:11:25.131205+09
```

- pg_wait_event … 待機イベントのマスタテーブル

MAINTAIN権限と pg_maintainロール

- MAINTAIN権限
  - テーブルに対して、ロールに GRANT/REVOKEする新たな権限
  - VACUUM、ANALYZE、CLUSTER、REFRESH MATERIALIZED VIEW、REINDEX、LOCK TABLE ができる
    - これまで LOCK TABLE 以外は 所有者か SUPERUSER 限定の操作
- pg_maintainロール
  - 任意のテーブルに MAINTAIN権限を持つ定義済みロール

→ 限定された権限のロールでメンテナンス操作をさせるのに有用
（所有者だと ALTER TABLE もできてしまう）

PostgreSQL 17

非互換の変更点

非互換の変更点(1)

- メンテナンス操作が安全な search_path で実行される
  - VACUUM、ANALYZE、REINDEX、REFRESH MATERIALIZED VIEW で search_path = 'pg_catalog, pg_temp' が暗黙に設定される
  - search_path 優先度の高いスキーマにオーバーライドする関数や演算子を作ることで意図しない処理を実行させる攻撃を防ぐ

  → 既存のユーザ定義関数やマテビュー定義で、
  完全修飾でないオブジェクト名を使っていると、動作しなくなる可能性

- interval型の表現文字列が厳格化
  - ago の繰り返し出現「10 minute ago 3 second ago」→ これからはエラー
  - 数値の無い単位　「1 day month year」→ これからはエラー

非互換の変更点(2)

- SET SESSION AUTHORIZATION の動作変更
  - これまで:　接続時に SUPERUSER ならセッションユーザ変更可能
  - これから:　本コマンド実行時に SUPERUSER であることが必要
    → 特に、接続後に SUPERUSER 属性が剥奪された場合に違いがでる
- 設定パラメータ変更
  - old_snapshot_threshold → 廃止
    - 機能としても廃止（古いトランザクションがあってもVACUUM処理を許す機能）
  - db_user_namespace　→ 廃止
    - 機能としても廃止（データベースごとのユーザをエミュレートする機能）
  - trace_recovery_messages → 廃止
    - on のとき出力されていたメッセージは引き続き DEBUG1 ～ DEBUG4 で出力
  - wal_sync_method → 付与可能な値として fsync_writethrough が廃止
    - Windows では単に fsync を指定すれば同じ意味

非互換の変更点(3)

- LSN から WALファイル名を出す関数: セグメント境界で仕様変更
  - 16.x まで

```
=# SELECT * FROM (VALUES (-1),(0),(1)) v(i), LATERAL
   pg_walfile_name_offset('0/0'::pg_lsn + 16777216 + i);
 i  |         file_name          | file_offset
----+----------------------------+-------------
 -1 | 000000010000000000000000 |    16777215
  0 | 000000010000000000000000 |           0
  1 | 000000010000000000000001 |           1
(3 rows)
```

  - 17.x から

```
=# SELECT * FROM (VALUES (-1),(0),(1)) v(i), LATERAL
   pg_walfile_name_offset('0/0'::pg_lsn + 16777216 + i);
 i  |         file_name          | file_offset
----+----------------------------+-------------
 -1 | 000000010000000000000000 |    16777215
  0 | 000000010000000000000001 |           0
  1 | 000000010000000000000001 |           1
(3 rows)
```

pg_walfile関数も同様に仕様変更

非互換の変更点(4)

- システムテーブル／システムビューの変更
  - pg_stat_bgwriter
    - チェックポイント関連の列は pg_stat_checkpointer に移動
    - buffers_backend列と buffers_backend_fsync列が廃止（pg_stat_io ビューが代替）
  - pg_stat_statement
    - blk_read_time と blk_write_time が shared_blk_read_time、shared_blk_write_time に変名
    - いくつか列追加
  - pg_stat_progress_vacuum
    - max_dead_tuples が max_dead_tuples_bytes に変更、最大格納バイト数を示す
    - num_dead_tuples が num_dead_item_ids に変名
  - pg_stat_slru
    - 出力される name列の値が変更 （「CommitTs」→「commit_timestamp」など）
    - それに伴い pg_stat_reset_slru 関数で指定する項目名も変更

まとめ

- PostgreSQL
  - 長い歴史を持つ、特定オーナーを持たない OSS DB
  - 安定した開発体制と、引き続き活発な開発活動

- PostgreSQL 17
  - 2024年 9月26日にリリース
  - 比較的エンハンスの大きいバージョン
    - VACUUM性能向上
    - ロジカルレプリケーション運用性向上
    - インクリメンタルバックアップ

Tech Blogサイトに
PG17 検証レポート