PostgreSQL 16 最新情報セミナー

2023-09-29

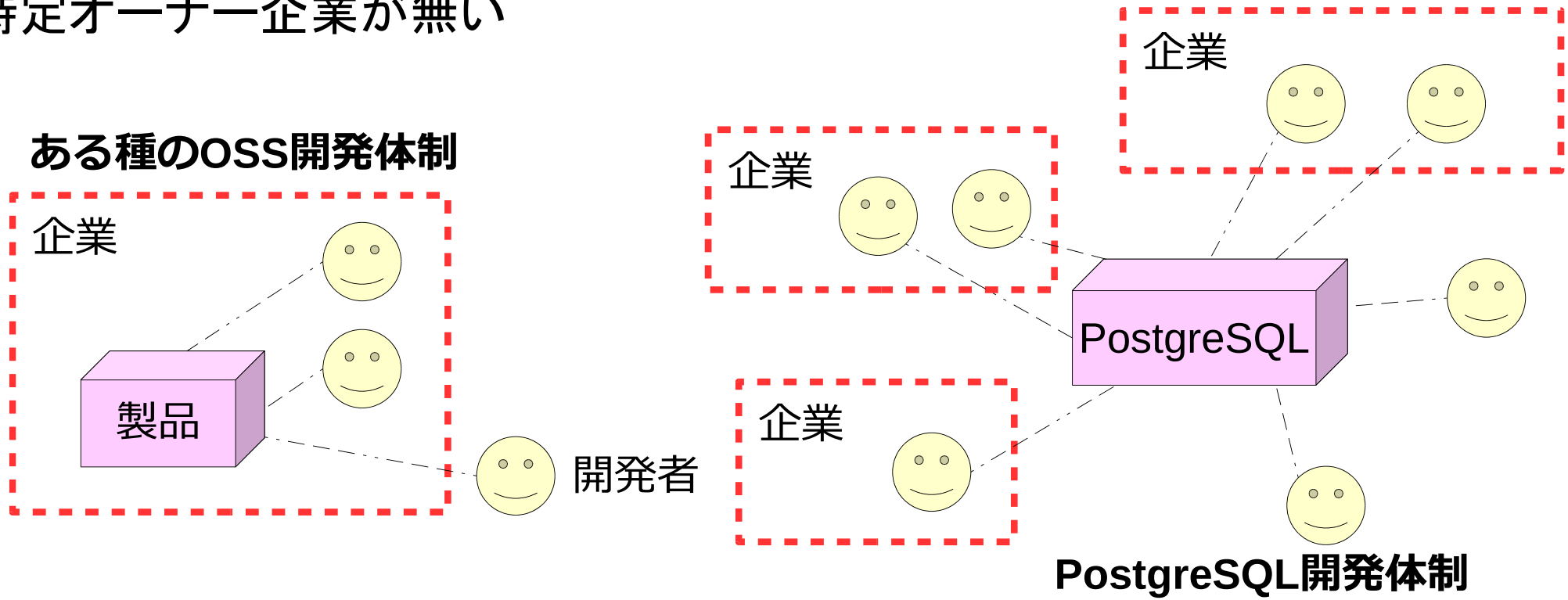
SRA OSS合同会社 三和陽菜

講演内容

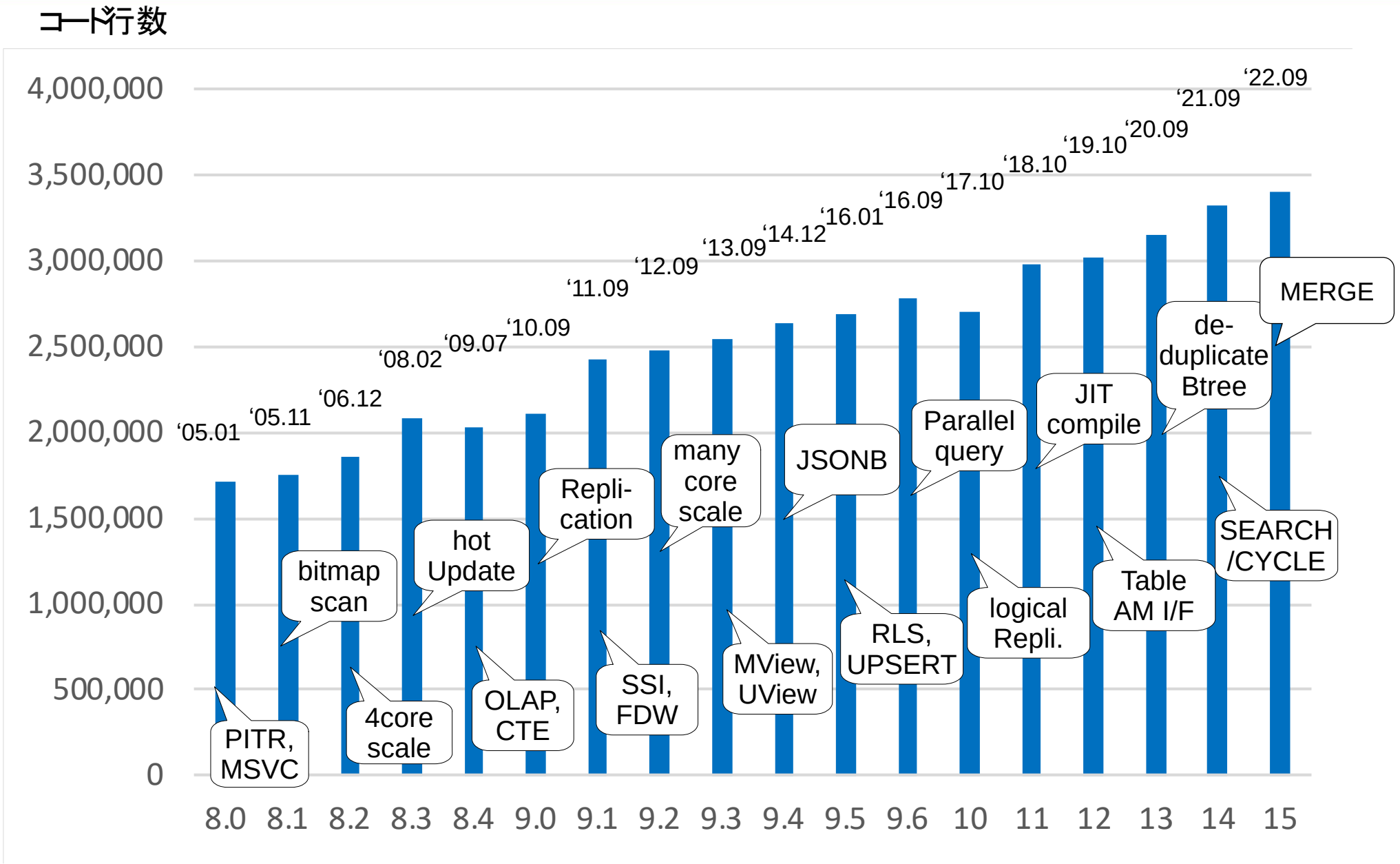
- 講演内容
  - PostgreSQL概要
  - PostgreSQL 16 の新機能、非互換変更

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
- 豊富なプラットフォーム選択肢

弱み

- 伝統的な設計による性能限界
  - ストレージ主体
  - 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
  - スケールアウト
  - 監査ログ、プランナヒント、監視、など

PostgreSQL 16 新機能

- 性能向上
  - プランナ／エグゼキュータ改善
  - 同時データ投入の改善
  - SIMD CPU機能対応
- SQL機能
  - SQL/JSON
  - 数値リテラル表現
  - ICU照合順序ルールに対応
- クライアント
  - libpqロードバランス
  - psqlで拡張問い合わせ
- ロジカルレプリケーション
  - スタンバイからパブリケーション
  - 並列適用
  - 双方向レプリケーション
- 運用管理
  - 新たなモニタリング項目
  - 新たな定義済ロール
  - VACUUM／凍結処理の改良
  - Mesonビルド

プランナ／エグゼキュータ(1) パラレル対応の拡大

- Hash Full Join／Hash Right Join がパラレル対応
  - これまで外部結合での Hash結合は Hash Left Join のみがパラレル対応

- string_agg 、 array_agg でパラレル対応

```
db1=# explain SELECT array_agg(special_flg), string_agg(special_flg, ',')
        FROM t_agg_test WHERE special_flg IS NOT NULL;
                                QUERY PLAN
-------------------------------------------------------------------------
 Finalize Aggregate  (cost=859.80..859.81 rows=1 width=32)
   ->  Gather  (cost=859.78..859.79 rows=2 width=32)
         Workers Planned: 2
         ->  Partial Aggregate  (cost=859.78..859.79 rows=1 width=32)
               ->  Parallel Seq Scan on t_agg_test
                     (cost=0.00..859.67 rows=43 width=2)
                     Filter: (special_flg IS NOT NULL)
```

プランナ／エグゼキュータ(2) インクリメンタルソート対応

- SELECT DISTINCT でもインクリメンタルソートに対応

```
db1=# explain SELECT DISTINCT ON (num1, num2, num3) * FROM t_sort;
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------
 Unique  (cost=0.78..7082.01 rows=10000 width=14)
   ->  Incremental Sort  (cost=0.78..6332.01 rows=100000 width=14)
         Sort Key: num1, num2, num3
         Presorted Key: num1, num2
         ->  Index Scan using t_sort_num1_num2_idx on t_sort
             (cost=0.29..3221.04 rows=100000 width=14)
```

プランナ／エグゼキュータ(3) ウィンドウ関数の最適化

- フレーム化オプションの最適化
- Run Condition 適用拡大

```
=# explain (ANALYZE, COSTS OFF) SELECT sid, examid, score,
    row_number() OVER (PARTITION BY examid ORDER BY score DESC) rn,
    rank() OVER (PARTITION BY examid ORDER BY score DESC
       ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) rnk,
    dense_rank() OVER (PARTITION BY examid ORDER BY score DESC
       RANGE BETWEEN CURRENT ROW AND CURRENT ROW) drnk
    FROM t_score ORDER BY sid, examid;
```

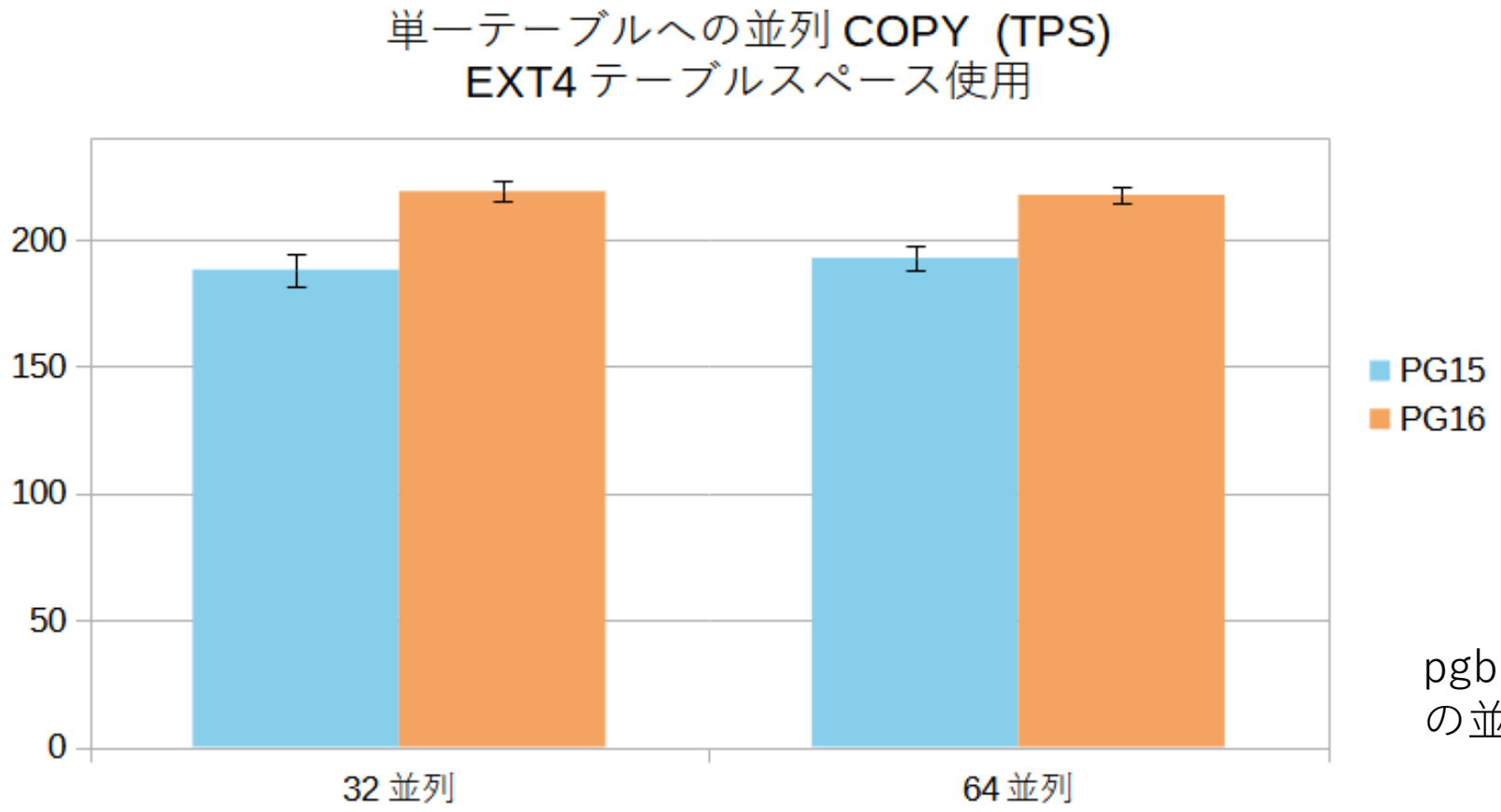
```
                                        QUERY PLAN
-----------------------------------------------------------------------------
Sort (actual time=1.214..1.232 rows=300 loops=1)
  Sort Key: sid, examid
  Sort Method: quicksort  Memory: 48kB
  ->  WindowAgg (actual time=0.238..1.027 rows=300
        ->  WindowAgg (actual time=0.227..0.643 rows
              ->  WindowAgg (actual time=0.224..0.42
                    ->  Sort (actual time=0.215..0.23
                          Sort Key: examid, score DESC
                          Sort Method: quicksort  Memo
                          ->  Seq Scan on t_score
                                (actual time=0.008..0.1
```

フレーム化オプション最適化でプラン要素を集約

```
                                        QUERY PLAN
-----------------------------------------------------------------------------
Sort (actual time=0.588..0.607 rows=300 loops=1)
  Sort Key: sid, examid
  Sort Method: quicksort  Memory: 43kB
  ->  WindowAgg (actual time=0.179..0.483 rows=300
        ->  Sort (actual time=0.164..0.182 rows=300
              Sort Key: examid, score DESC
              Sort Method: quicksort  Memory: 36kB
              ->  Seq Scan on t_score
                    (actual time=0.021..0.066 rows=3
```

性能改善(1) 同時データ投入の改善

- テーブルのページ拡張を複数ページまとめて行う
  - 並列でデータ投入する場合に効果的

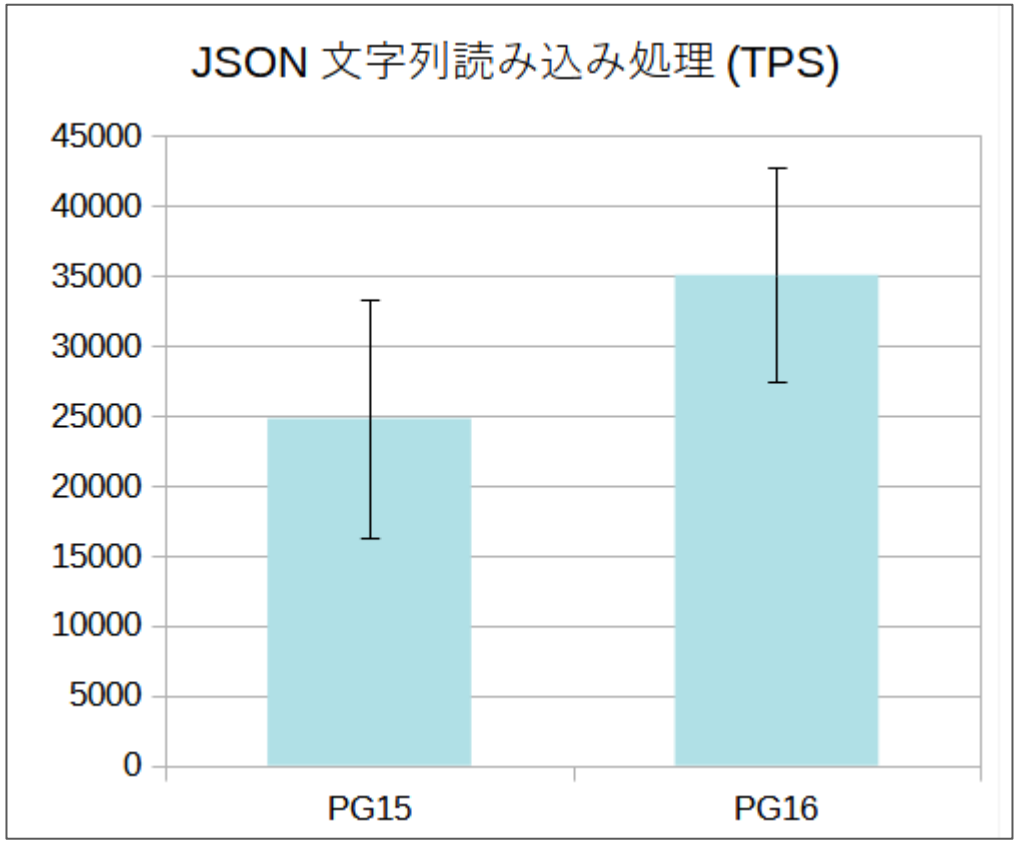


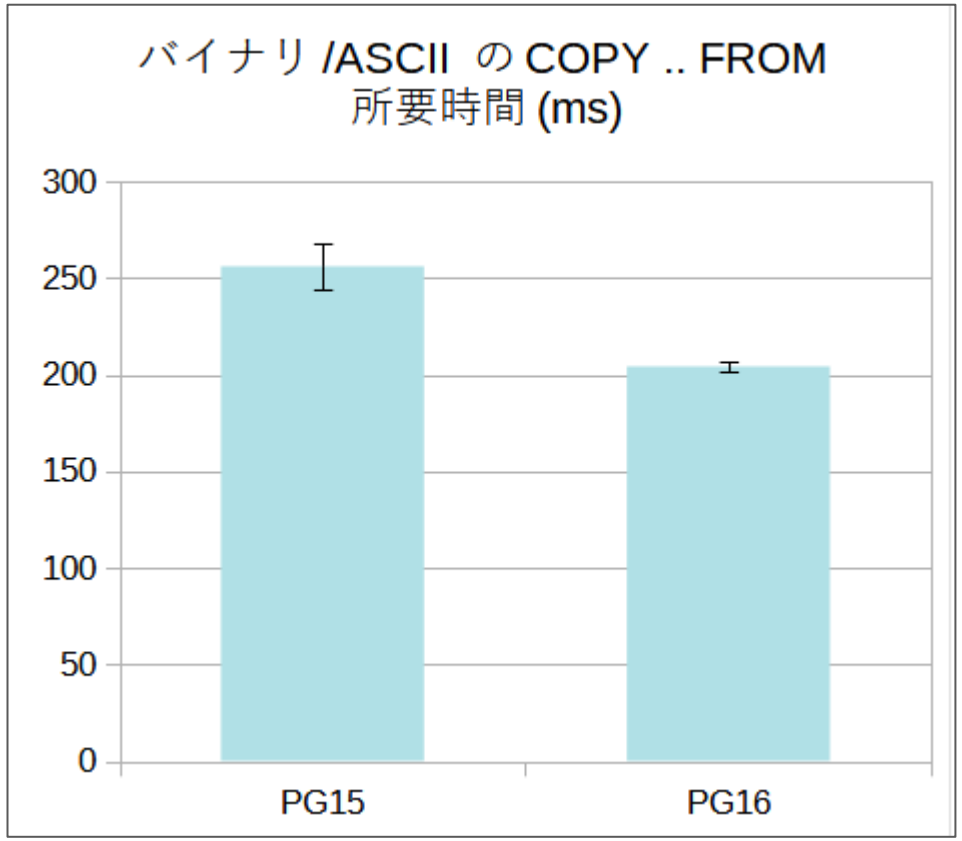
ファイルシステムによる性能差あり

pgbench による COPY FROM の並列同時実行

性能改善(2)　SIMD CPUアクセラレータ対応

- SIMD (Single Instruction Multiple Data)
- x86_64 アーキテクチャCPU では SSE2 命令に対応
  - 内部のプロセス制御処理、JSON文字列入力、ASCII文字列入力





SQL機能(1)　SQL/JSON

PostgreSQL 15で１回取り下げとなったもの。若干変更あり。

- 述語とJSONデータ構築関数が追加

構文	説明
IS JSON [VALUE]	文字列が JSON として解釈できるなら真を返す。
IS JSON ARRAY	文字列が JSON 配列として解釈できるなら真を返す。
IS JSON OBJECT	文字列が JSON オブジェクト（いくつかのキーと値）として解釈できるなら真を返す。
IS JSON SCALAR	文字列が JSON のスカラ値として解釈できるなら真を返す。

関数名	説明
JSON_OBJECT	テキスト配列から JSON オブジェクトを構築します。
JSON_OBJECTAGG	提供されたデータを JSON オブジェクトに集約します。
JSON_ARRAY	提供された SQL または JSON データから JSON 配列を構築します。
JSON_ARRAYAGG	提供された SQL または JSON データを JSON 配列に集約します。

SQL機能(2)　数値リテラル表現

- 16進数、8進数、2進数の整数値

```
db1=# SELECT 0x42F, 0o273, -0b100101;
 ?column? | ?column? | ?column?
----------+----------+----------
     1071 |      187 |      -37
(1 row)
```

マイナスはN進数
指定の手前に書く。

- 桁区切り付きの数値

```
db1=# SELECT 1_000_000_000.000;
    ?column?
----------------
 1000000000.000
(1 row)
```

「_」の使い方は自由。
4桁区切りでも良い。
何であれ無視される。

SQL機能(3)　ICU照合順序

- ICUライブラリはビルド時に含めるのがデフォルトになった
- 定義済み照合順序
  - 各言語地域のICU照合順序、ucs_basic（UNICODEコード順）、unicode（言語地域未定時の妥当そうなソート順）
- ルール機能

ソート順序に
英米式トランプ
の並び順を指定

```
db1=# CREATE COLLATION col_card_rule (provider = icu, locale = 'und',
      rules = '& 1 < A  & 9 < 10  & 10 < J  & J < Q  & Q < K');
```

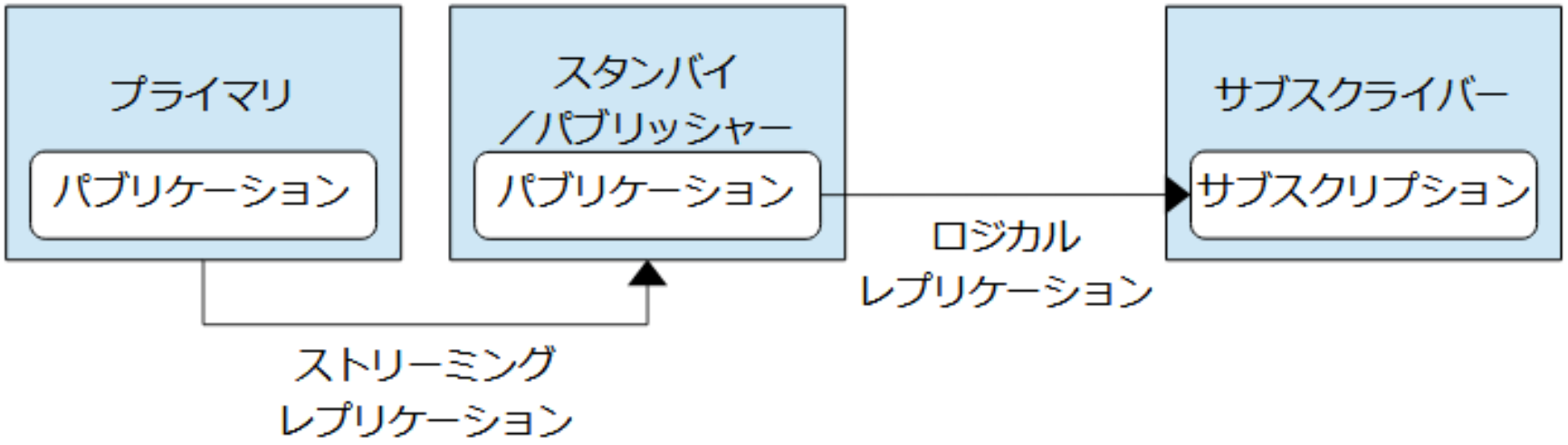
「高」と
「髙」を
同じとみなす
ように指定

```
db1=# CREATE COLLATION col_my_ident_rule (provider = icu, locale = 'und',
      deterministic = false, rules = '& 高 = 髙');

db1=# SELECT '高' = '髙' COLLATE col_my_ident_rule;
 ?column?
----------
 t
```

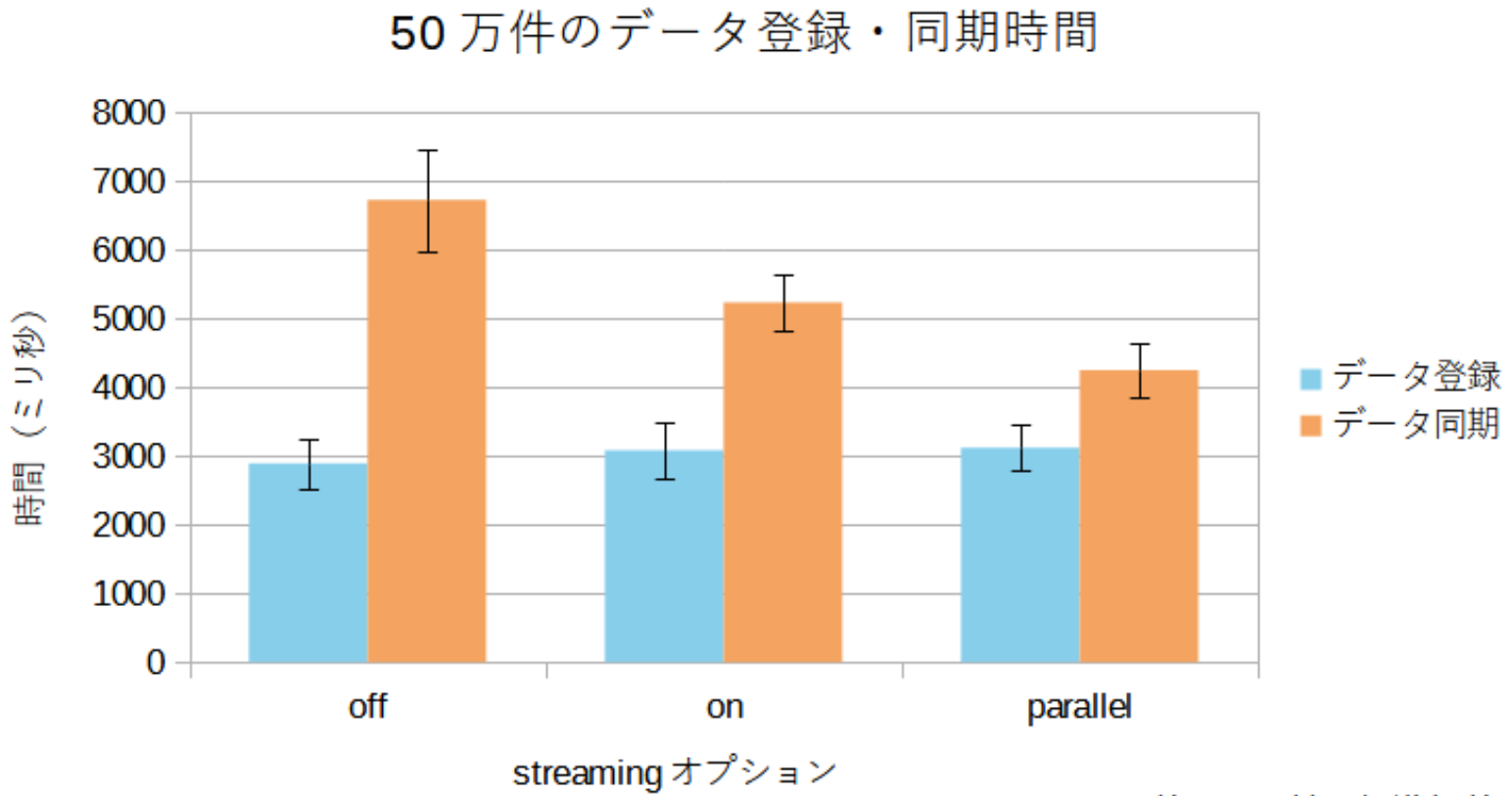
ロジカルレプリケーション（1）スタンバイでパブリケーション

- ストリーミングレプリケーションのスタンバイから、
  ロジカルレプリケーションが可能になった
  - CREATE PUBLICATION は プライマリで行う
  - CREATE SUBSCRIPTION で接続先にスタンバイサーバを指定する
  - 補助関数 pg_log_standby_snapshot()



ロジカルレプリケーション(2) 並列適用

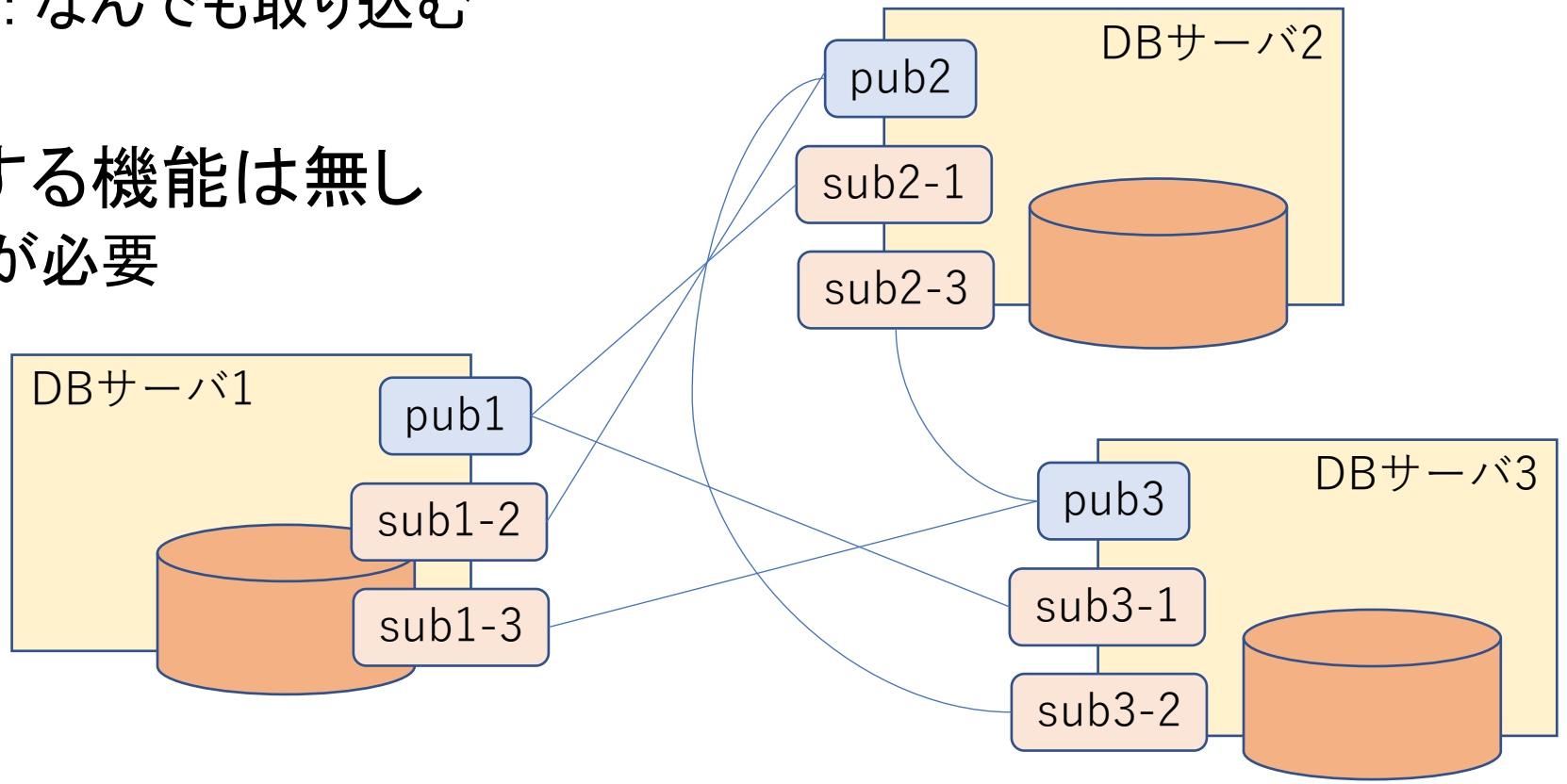
- サブスクリプションの新オプション streaming = parallel
  - コミットを待たずに送信し、さらに並列に適用する
  - max_parallel_apply_workers_per_subscription で最大並列度を指定



並列適用なので
行の物理順序は
一致しなくなる

ロジカルレプリケーション(3) 双方向レプリケーション

- サブスクリプション origin = none 指定
  - オリジンがある更新データは取り込まない – 更新のピンポンを防止
    ⇒ マルチマスタ構成が可能
  - デフォルトは any : なんでも取り込む
- 更新の衝突に関する機能は無し
  - 利用者側で対策が必要

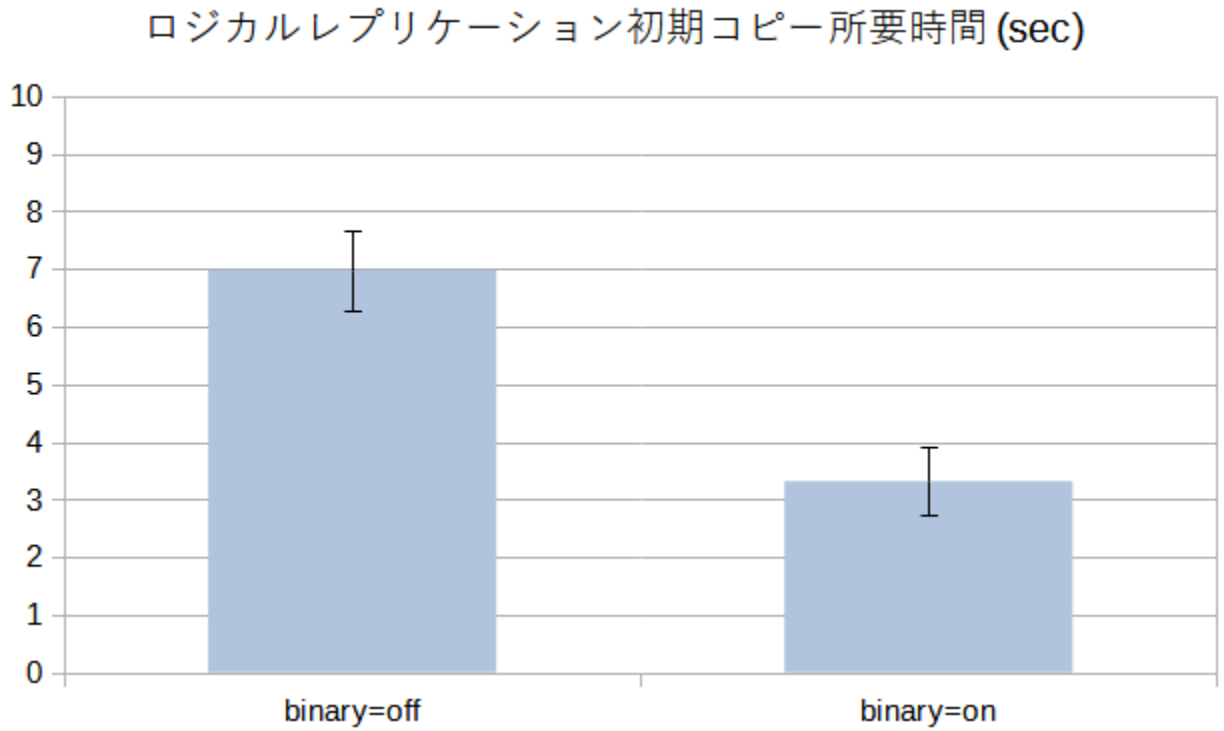


ロジカルレプリケーション(4) 様々な高速化

- 主キー以外のインデックス使用
  - 主キーが無く、REPLICA IDENTITY FULL が指定されているテーブルの
    レプリケーションで、UPDATE / DELETE で行を特定するときに、
    主キー以外のインデックスがあれば活用する

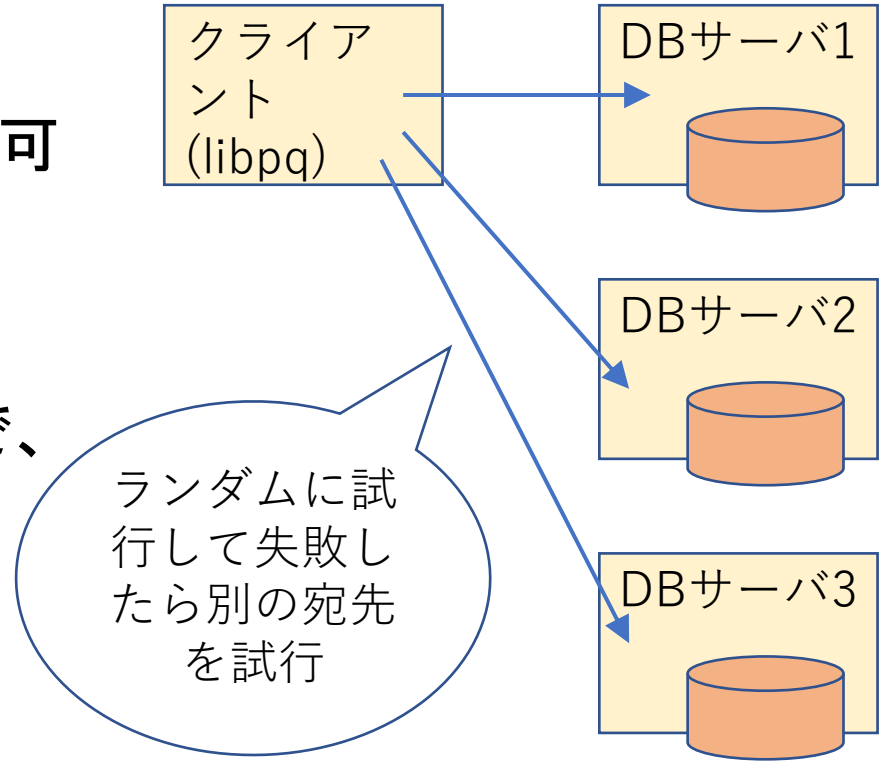
- 初期コピーでバイナリ転送
  - これまで binary=on でも、
    初期コピーには適用されなかった

bytea型と
timestamp型の列
を持つテーブルで
検証



クライアント機能(1) libpqロードバランス

- 接続オプション load_balance_hosts = random
  - デフォルトは disable
  - 環境変数PGLOADBALANCEHOSTS でも指定可
  - 以前から複数接続先ホスト指定は可能
    （必ず先頭から試行する）
  - 以前からある target_session_attrs オプションで、
    接続時プライマリ・スタンバイ指定も可



クライアント機能(2) psql / pg_dump

- psql で拡張プロトコル
  - parse → bind → execute で実行させる

```
db1=# \bind 'foo' 'bar'


db1=# SELECT $1, $2;
 ?column? | ?column?
----------+----------
 foo      | bar
(1 row)
```

- pg_dump 圧縮オプションに lz4、zstd 追加
  - 従来は gzip のみ
  - lz4 : （圧縮時）速い、圧縮率はやや劣る
  - zstd: （圧縮時）遅い、高い圧縮率

いずれも
圧縮レベルの
オプションが
デフォルトの
場合

運用管理(1)　新たなモニタリング項目

関数／ビュー	説明
ビュー pg_stat_io	ストレージ I/Oに関する統計情報を出力。 → ストレージ使用量把握に
ビュー pg_stat_*_tables/indexes の列 last_seq_scan、last_idx_scan	最後にシーケンシャルスキャン、インデックススキャンが実行された時刻を出力。 → 使用状況把握に
ビューpg_stat_*_tables の列 n_tup_newpage_upd	新しいページへの行バージョン作成を伴って更新された行数を出力。 → FILLFACTOR調整の目安に
関数 pg_stat_get_backend_subxact (backendid)	バックエンドのサブトランザクションキャッシュ内のサブトランザクション数、そのキャッシュがオーバーフローしているかを出力。 → サブトランザクション多用による性能劣化把握

運用管理(2)　新たな定義済ロール

- pg_maintain
  - 全てのテーブルにメンテナンス操作ができるロール
  - VACUUM、ANALYZE、CLUSTER、REFRESH MATERIALIZED VIEW、REINDEX、LOCK TABLE
- pg_create_subscription
  - サブスクリプションを作成できるロール
  - データベースに対する CREATE権限も必要
- pg_use_reserved_connections
  - 新パラメータ reserved_connectionsによる予約枠を使えるロール
  - superuser_reserved_connections とは別に予約枠を設定

いずれも以前は
スーパーユーザ
限定であった

運用管理(3)　mesonビルド

- これまで
  - UNIX、Linux むけ：autoconf / make
  - Windows むけ：独自perlスクリプト / Visual Studio
- これらに加えて meson / ninja によるビルドをサポート
  - いまのところ従来ビルド方式のすぐの廃止が宣言されたわけではない

VACUUM処理(1) ページ凍結の改善

- ページ単位でのタプル凍結
  - 「積極的」でないVACUUM処理であっても、WALにフルページ書き込みをするページについては、凍結処理も行う

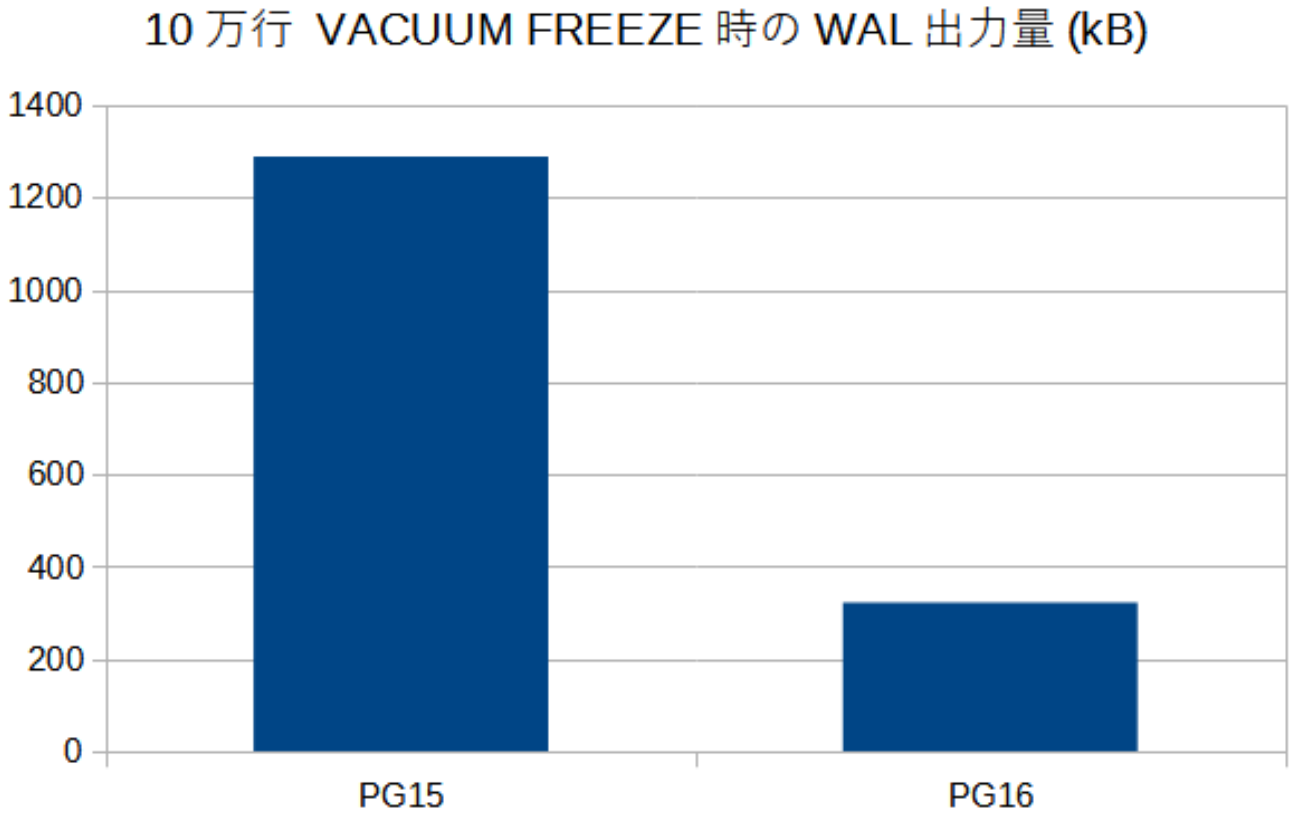
凍結処理を行う弊害は WAL が多く出ること
↓
元々 WALが多く出るタイミングで一緒に凍結処理もしてしまおう

タプル凍結とは？
- タプル（≒行）には作成/削除時の XID（トランザクションID）が記録される
- XID は今のところ 32bit整数なので、値を周回して使うために VACUUM時にタプルの XID値を「最も古い値」を意味する予約された値に書き換える

⇒ これを「凍結する」と呼ぶ

VACUUM処理(2) ページ凍結のWAL量削減

- ページ凍結を行う VACUUM による WAL出力量を抑制
  - WAL出力の重複をなくす



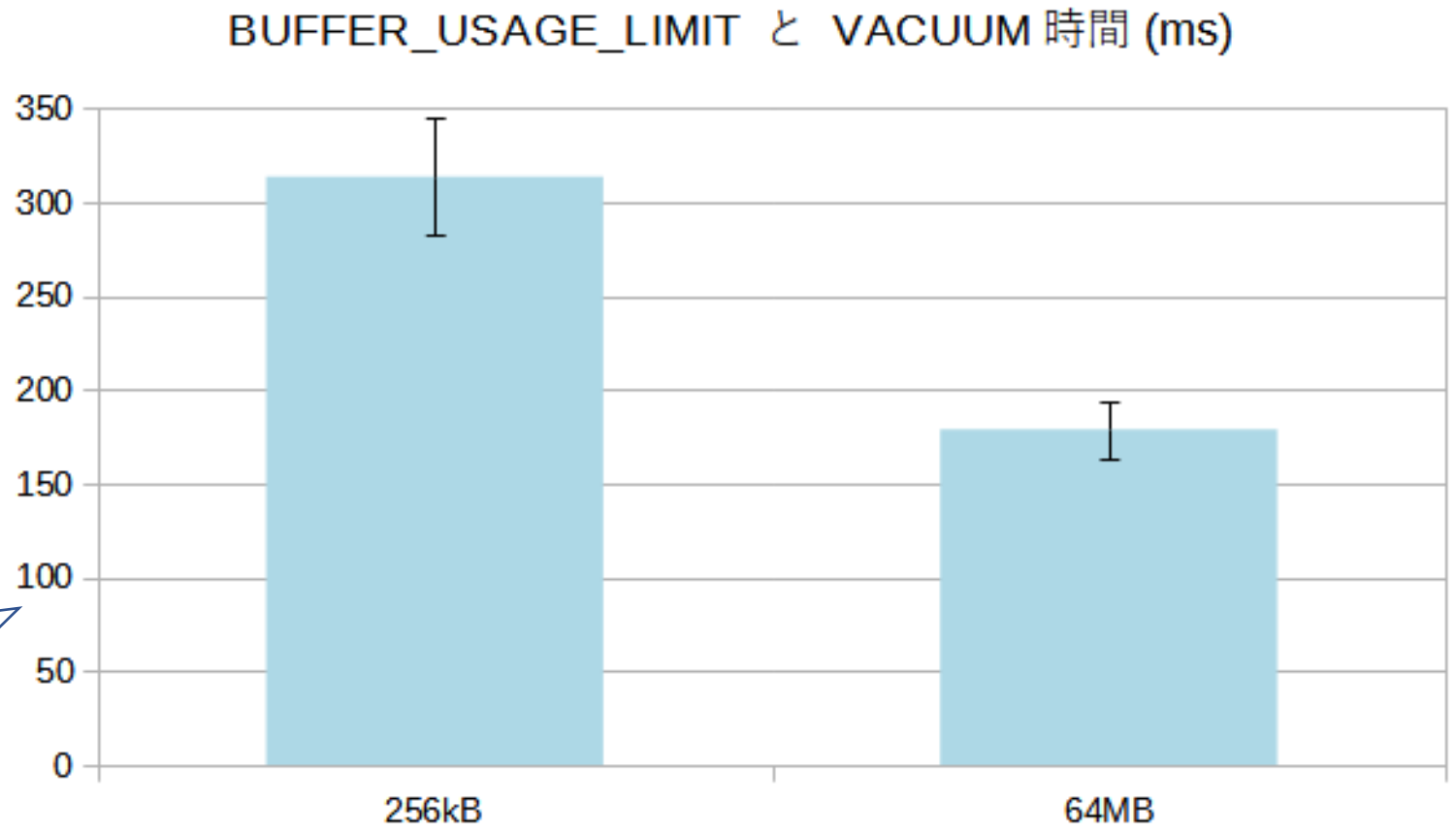
テーブルをTRUNCATEで空にする
↓
CHECKPOINT を実行
↓
10万行をINSERT
↓
無関係なトランザクションでXID進行
↓
VACUUM FREEZE 実行

VACUUM FREEZEコマンド前後で
WAL出力量を計測

VACUUM処理(3) リングバッファ指定パラメータ

- VACUUM コマンドに新オプション BUFFER_USAGE_LIMIT
  - デフォルト 256kB
  - この値の範囲内だけ、共有バッファを使用
  - １回のVACUUM で 共有バッファがすっかり書き換えられるのを防止

```
db1=# VACUUM (BUFFER_USAGE_LIMIT '1MB');
VACUUM
```



50MBほどのテーブルのVACUUM

非互換の変更(1)

- PL/pgSQL カーソル仕様変更
  - これまで: カーソル変数名がそのまま PostgreSQLカーソル名（ポータル名）
    - → ポータル名はセッション内が一つの名前空間なのでしばしば衝突していた
      「ERROR: cursor “c1” already in use」
  - これから: カーソル変数名とは無関係なポータル名が付与される
    - 「<unnamed portal 3>」のような名前になる
    - カーソル変数に文字列を代入してポータル名を明示的に指定することもできる

```
DECLARE c1 CURSOR FOR SELECT * FROM pg_users:
 BEGIN
     c1 := 'c1';
     OPEN c1;
     ...
```

非互換の変更(2)

- 主キー用のインデックスに NULLS NOT DISTINCT 指定禁止
  - これまで指定可能だったがリストア不能のダンプが生成されていた
- REINDEX DATABASE コマンドでシステムテーブルが対象外になる
  - これからは REINDEX SYSTEM を実行する必要がある
- 継承テーブル／パーティションテーブルで、
  親子間で生成列と通常列の混在禁止
  - 生成列の式が異なることは引き続き許容される
- ロールの CREATEROLE属性の制限
  - CREATEDB、REPLICATION、BYPASSRLS 変更にはその属性が必要に
  - ロールのメンバ変更には自身の所属と ADMIN OPTION が必要に

非互換の変更(3)

- contrib/pg_walinspect で関数廃止
  - pg_get_wal_records_info_till_end_of_wal()
  - pg_get_wal_stats_till_end_of_wal()
  - 「end_lsn」引数を持つ同機能の関数に「FFFFFFFF/FFFFFFFF」を指定すれば代替できる。

```
db1=# SELECT * FROM
        pg_get_wal_records_info('0/03000000', 'FFFFFFFF/FFFFFFFF');
```

- CREATE RULE “_RETURN“ によるビュー作成が廃止
- postmasterコマンド廃止
  - 以前から postgres へのシンボリックリンクにすぎない

非互換の変更(4)

- 既存設定パラメータの廃止、名称変更
  - force_parallel_mode → debug_parallel_query に名称変更
  - vacuum_defer_cleanup_age → 廃止
    - レプリケーションスロットや hot_standby_feedback = on で代替
  - promote_trigger_file → 廃止
    - トリガファイルを置くことでスタンバイサーバを昇格させる機能自体が廃止
  - lc_collate → 廃止
  - lc_ctype → 廃止
    - 接続先データベースの collate と ctype を返していた。以下問い合わせで代替。

```
SELECT datcollate collate, datctype ctype FROM
   pg_database WHERE datname = current_database();
```

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
  - 引き続き活発な開発活動

- PostgreSQL 16
  - 2023年9月14日リリース
  - 追加項目の多いリリース：機能/性能の両面で様々な拡張がされている

ご清聴ありがとうございました

- PostgreSQL 16検証報告

https://www.sraoss.co.jp/tech-blog/pgsql/pg16report/