～徹底検証報告～
次期メジャーバージョン PostgreSQL 9.5 の実力

SRA OSS, Inc. 日本支社　高塚 遙

2015-10-06 14:10 ～ 14:50
【PostgreSQL 最新動向セミナー】

本講演の内容

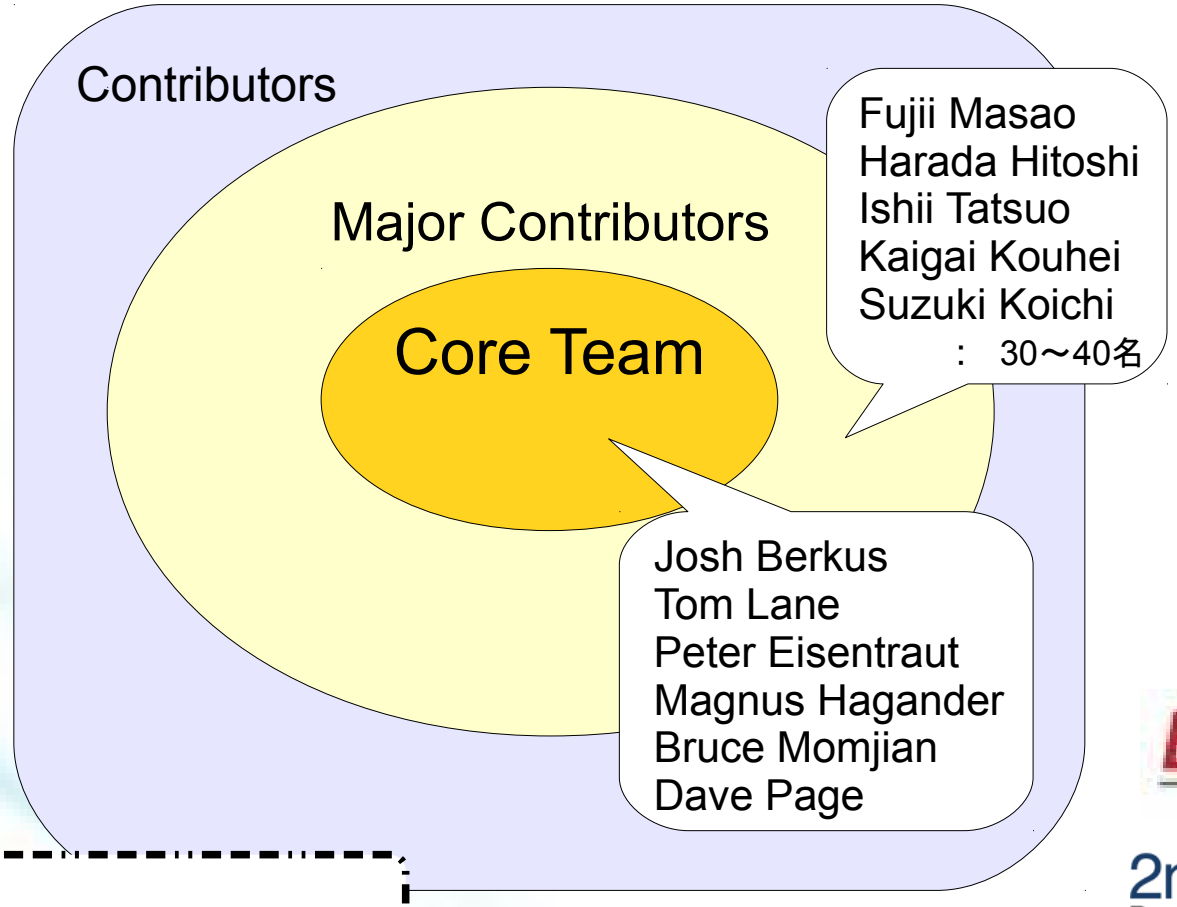
- PostgreSQL開発とリリースの概要
- PostgreSQL 9.5 新機能の検証報告

講演者について

高塚 遙
SRA OSS, Inc. 日本支社にて
PostgreSQLサポート、各種クラスタ構築技術支援、
マイグレーション技術支援、などを担当

PostgreSQL開発とリリース（1）

- 開発体制 - 特定のオーナー企業を持たない方式



支援企業

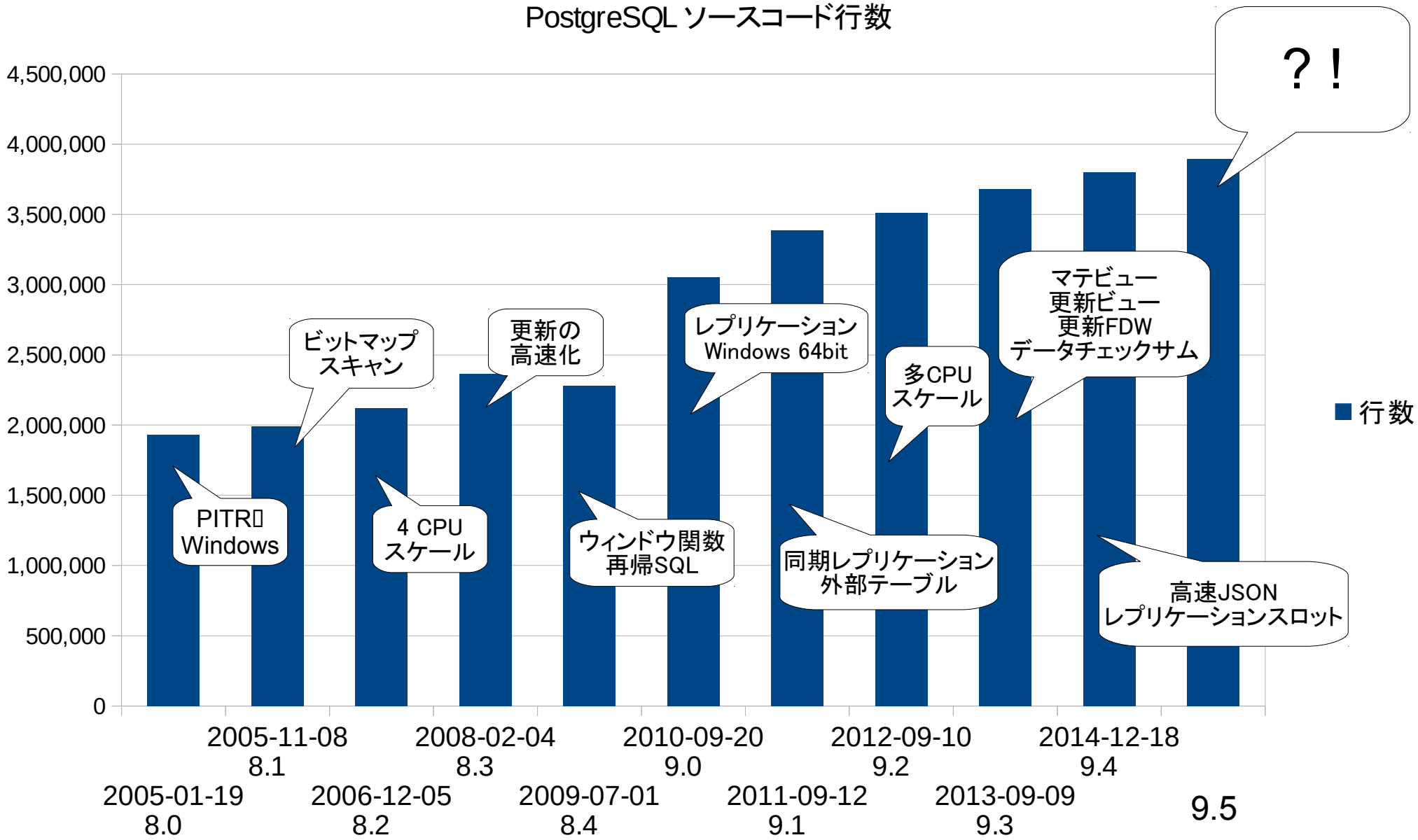
NTT
NEC
DALIBO
PGX POSTGRESQL EXPERTS, INC.
salesforce
meetme where new friends meet
Redpill Linpro
OmniTI
EnterpriseDB
SRA OSS, INC.
2ndQuadrant Professional PostgreSQL

etc...

※www.postgresql.org 記載より

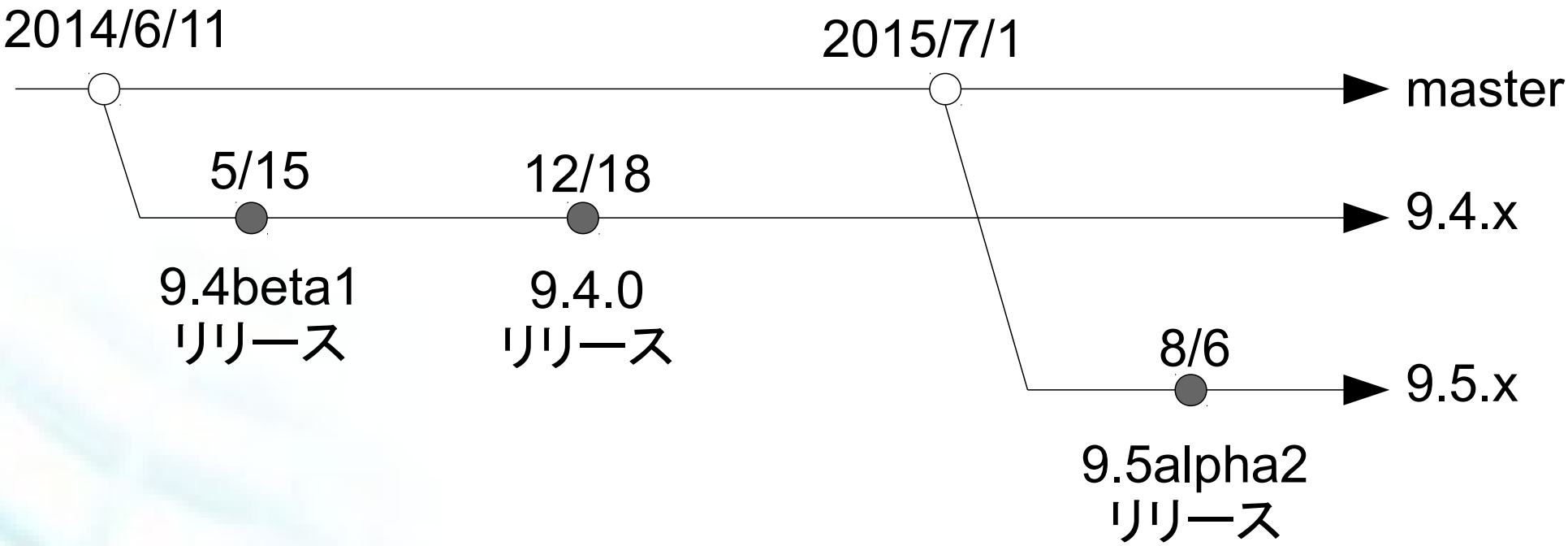
PostgreSQL開発とリリース（2）

- リリースの歴史



PostgreSQL開発とリリース（3）

- いつリリースされる？
  - 9.5.x は、仕様確定済み、9.5 alpha2 がリリース済み
  - 9.5.0 リリース予定は？

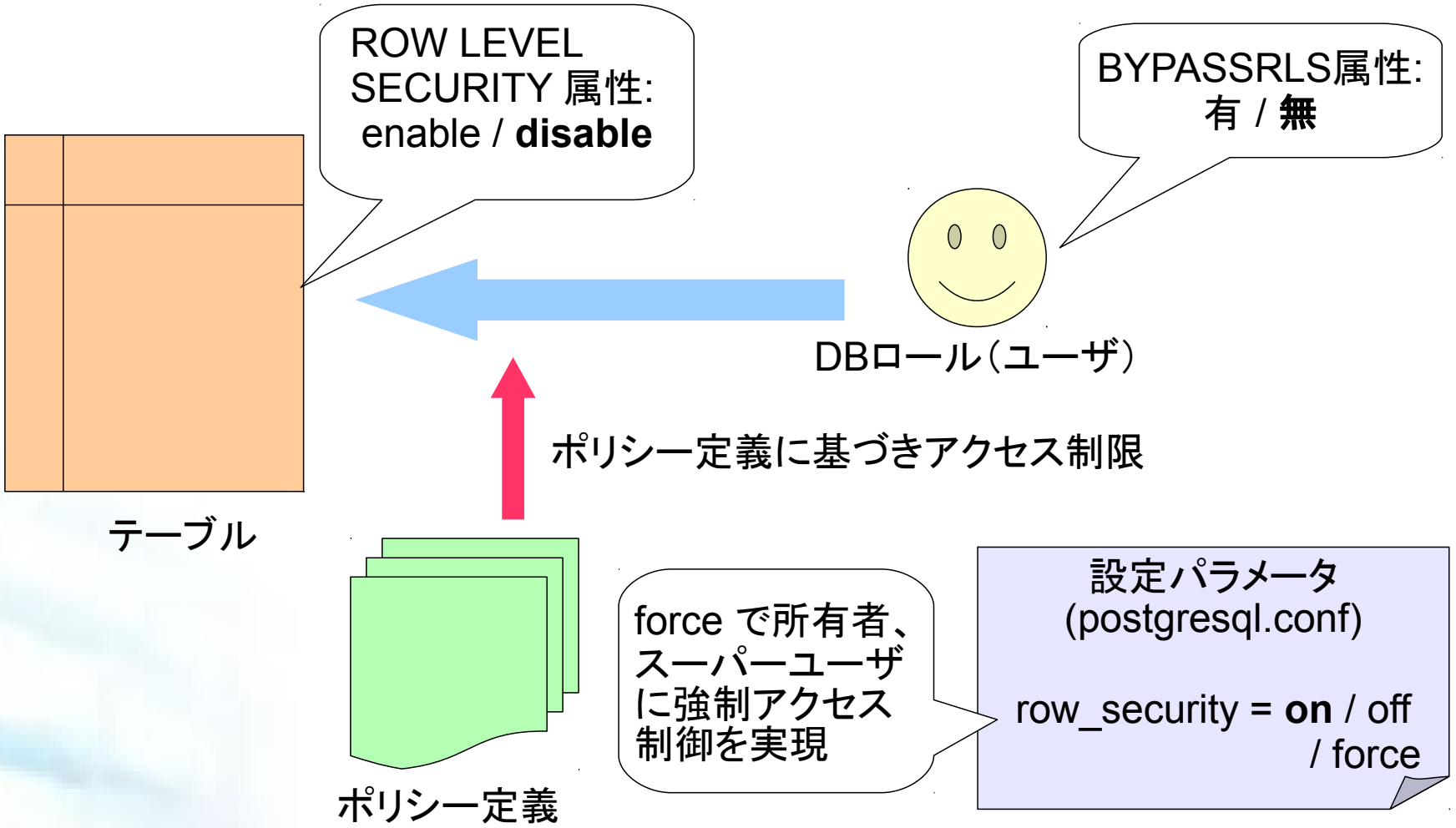


主要な新機能

- 行単位セキュリティ
- BRINインデックス
- トランザクションの巻き戻し
  pg_rewind コマンド
- 「挿入または更新」に対応
  INSERT ... ON CONFLICT ...
- 外部テーブルのインポート／継承
- 集計構文 に対応
  GROUPING SETS、CUBE、ROLLUP
- JSONB型 むけの関数・演算子の拡充

行単位セキュリティ（1）

- Row Level Security (RLS) と呼ばれる
  - ポリシー定義に基づきテーブルの行アクセスを制御



行単位セキュリティ(2)

```
GRANT ALL on t1 to user1;
ALTER TABLE t1 ENABLE ROW LEVEL SECURITY;
```

→ 従来のアクセス権限は全て「可」に設定し、
テーブルに対する RLS を有効に設定。

テーブル:t1

id	dat	lv
101	データ1（機密）	5
102	データ2	1
103	データ3（公開）	0
104	データ4	1
105	データ5（機密）	5

```
CREATE POLICY policy1 ON t1
  FOR SELECT TO user1
  USING (lv <= 2);
```

USING 指定条件が暗黙に適用

```
db1=> SELECT * FROM t1
        WHERE id <= 103;
  id | dat           | lv
 ----+---------------+----
  102 | データ2         | 1
  103 | データ3 (公開) | 0
(2 rows)
```

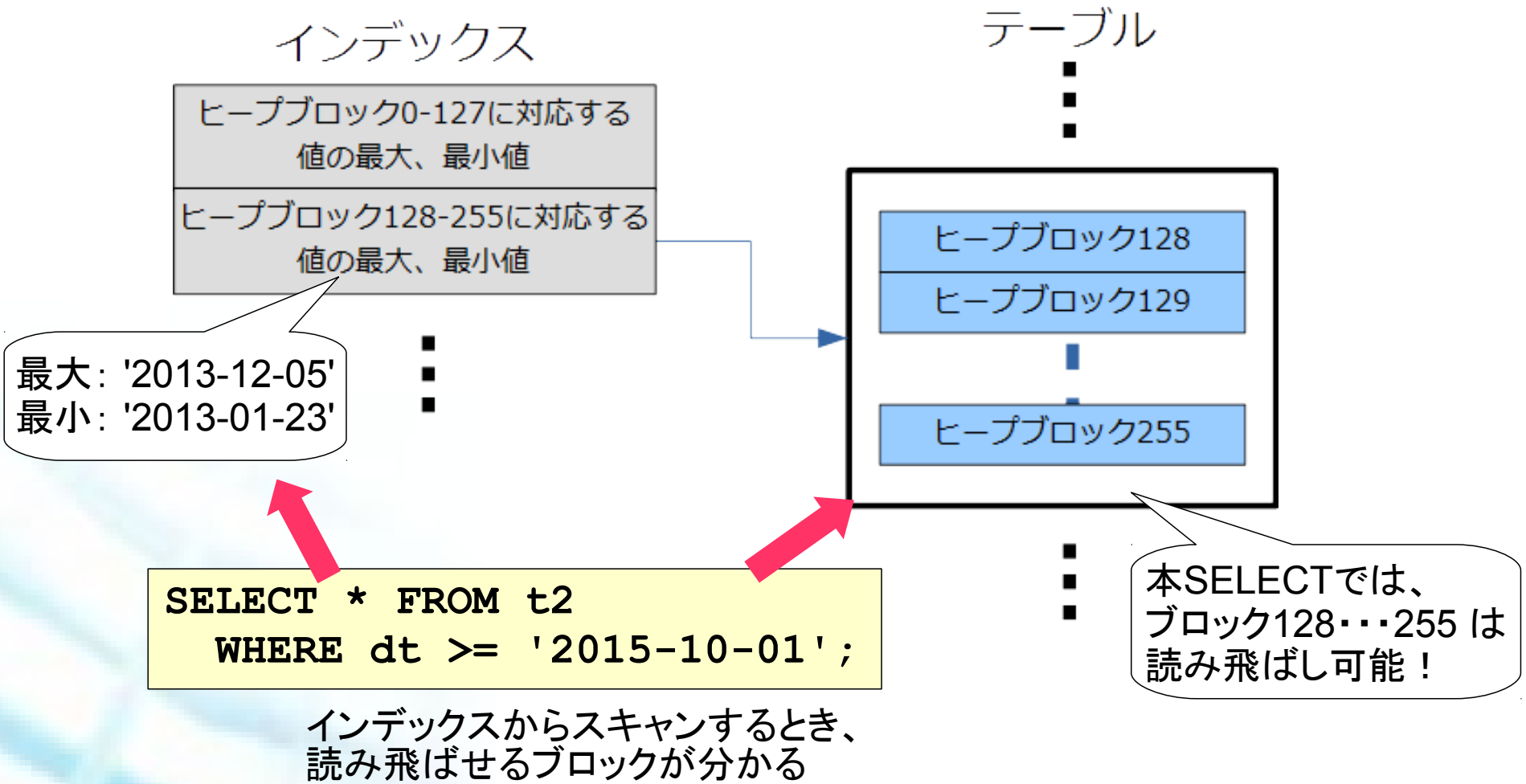
```
CREATE POLICY policy2 ON t1
  FOR INSERT TO user1
  WITH CHECK (lv = 1);
```

CHECK 指定条件を満たさないのでエラーになる

```
db1=> INSERT INTO t1 VALUES
      (106, 'データ6', 0);
ERROR:  new row violates
 row level security policy
 for "t1"
```

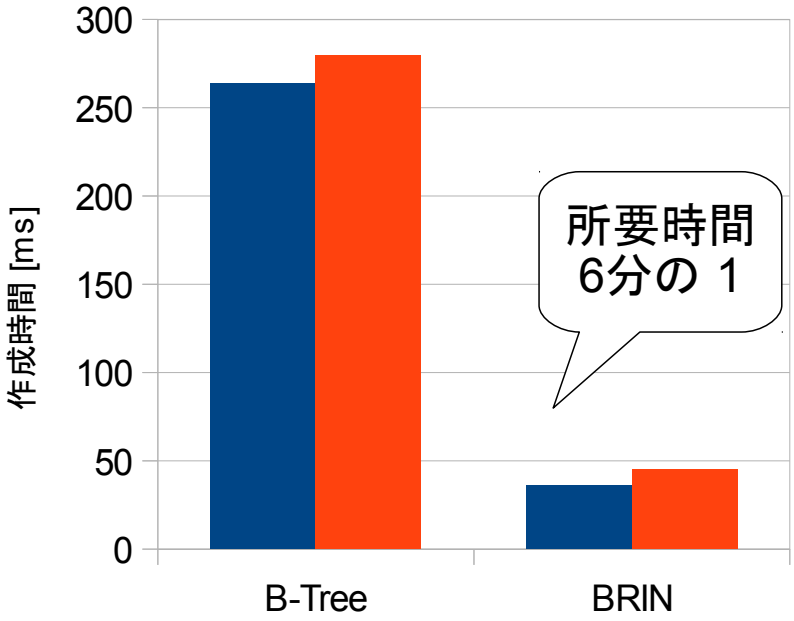
BRIN インデックス（1）

- 新しいインデックス方式「brin」が追加された
  - いくつかのブロック（＝ページ）の塊ごとに最大値・最小値を保持する

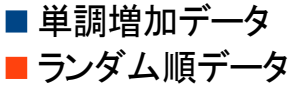


BRINインデックス(2)

- Btreeインデックスとの比較
  - 作成時間とインデックスサイズを 10万行程度のデータで実測

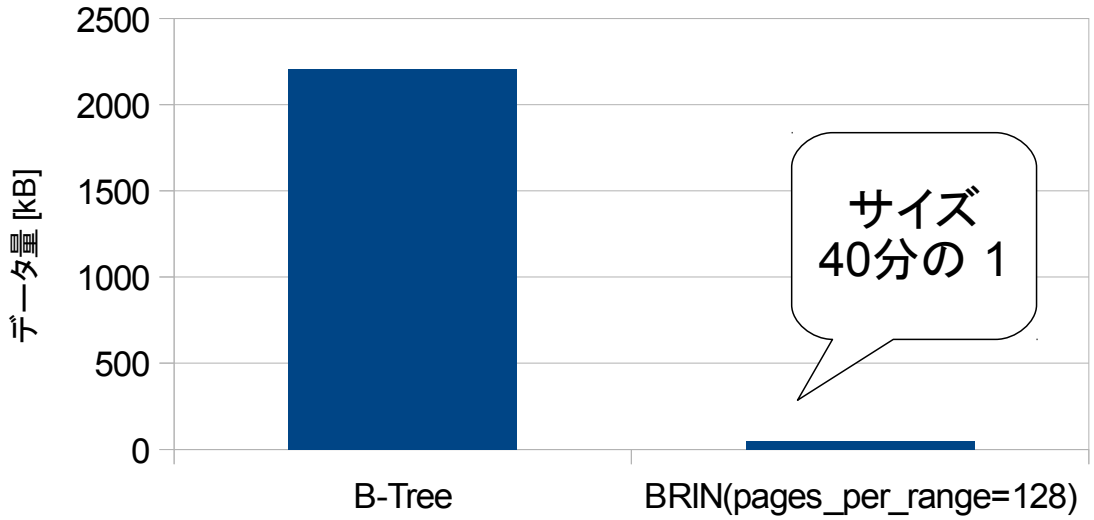


インデックス作成時間を比較



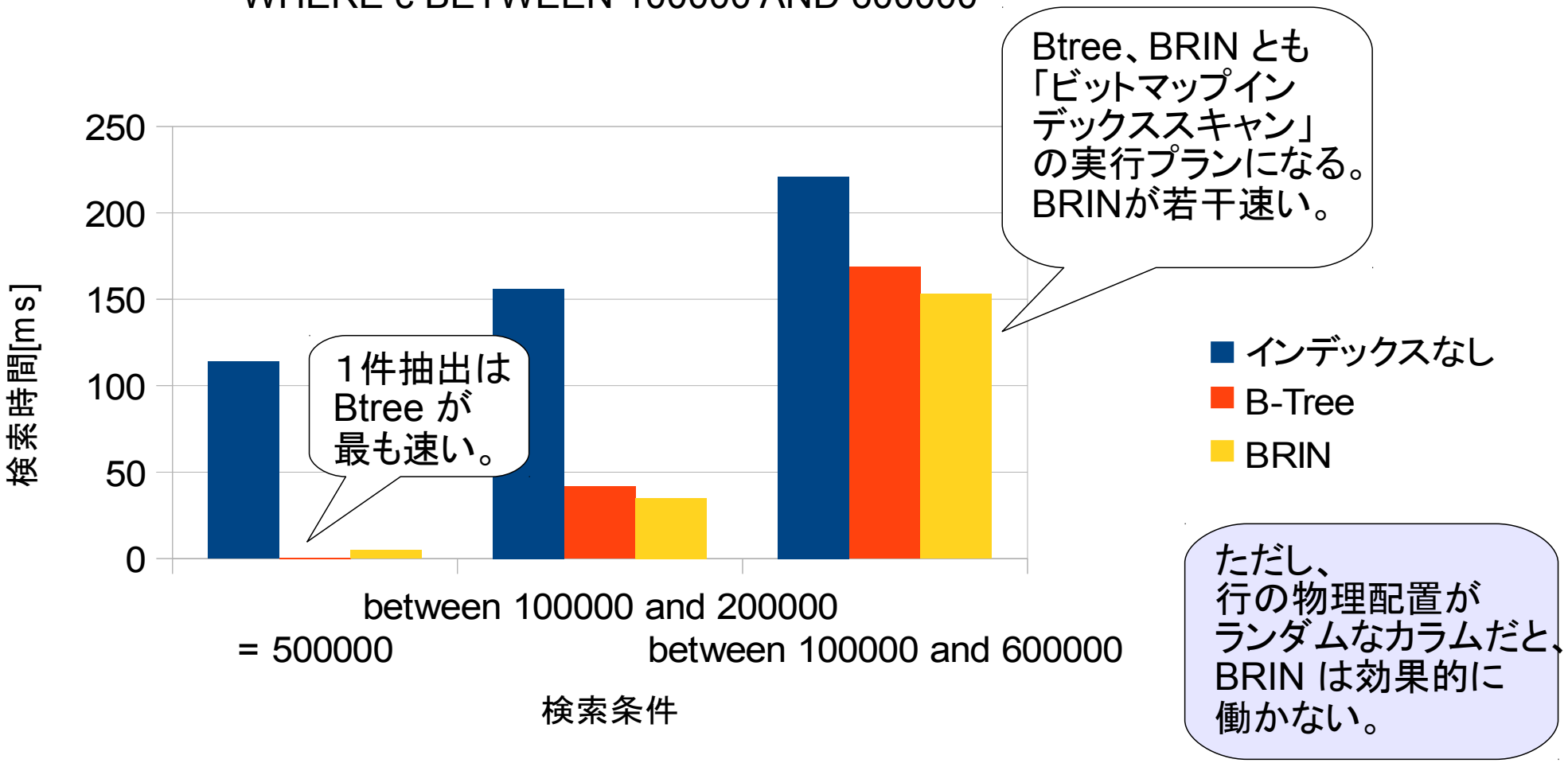
インデックス作成時オプション
pages_per_range にて、
何ブロックをインデックス上の
1項目とするかを指定。
128 はデフォルト。

10万件データの整数カラムにて
インデックスをサイズを比較



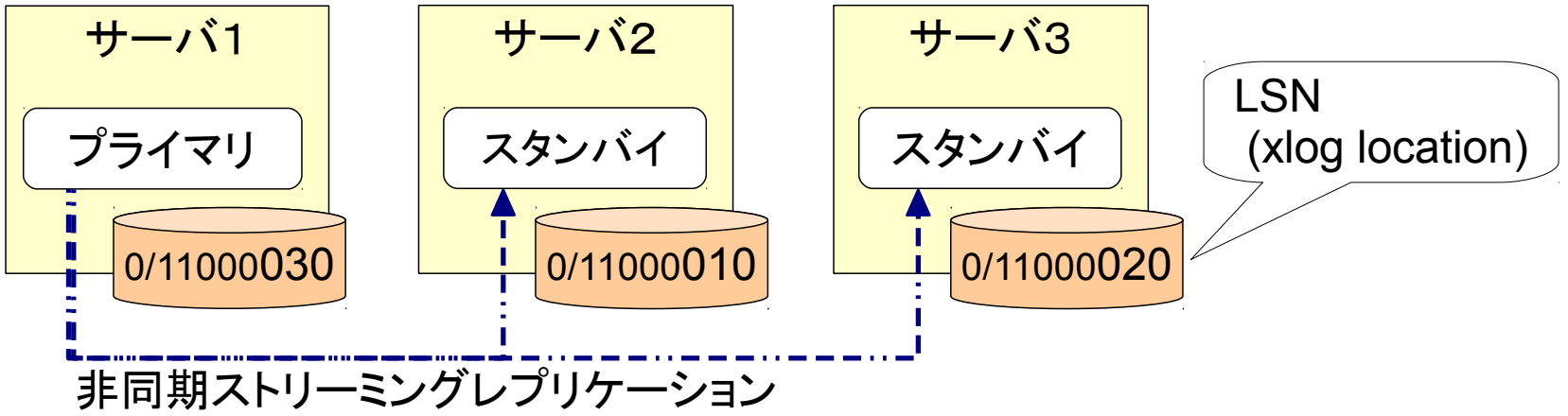
BRINインデックス(2)

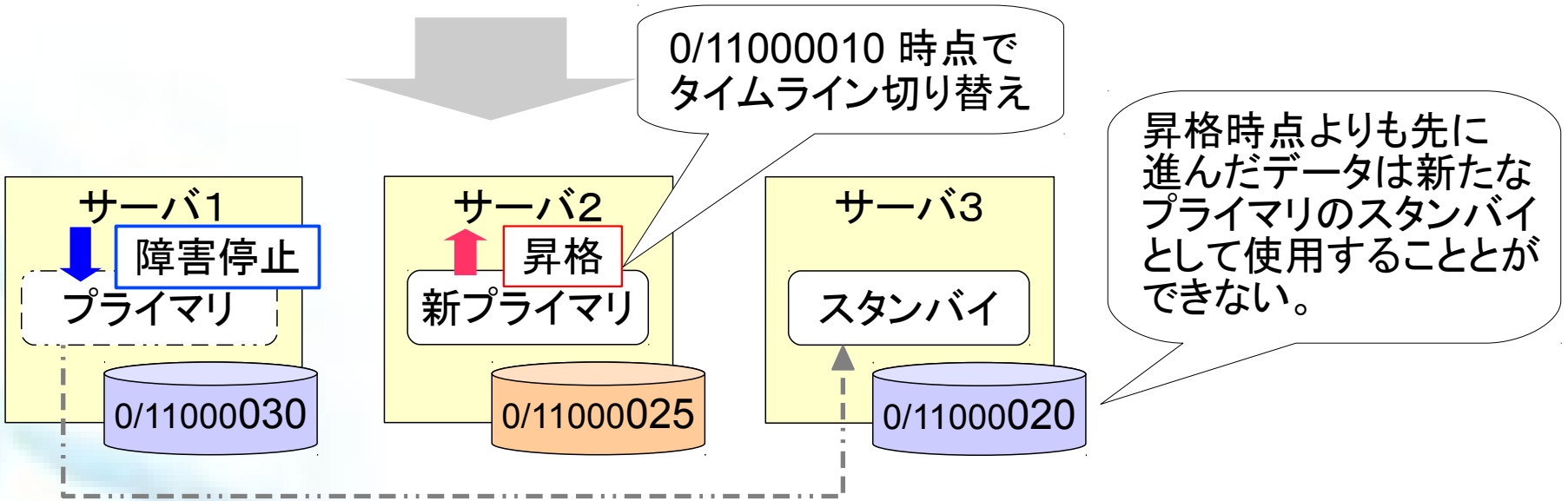
- Btreeインデックスとの比較 - 検索所要時間
  - 1 から 100万の昇順整数データを持つテーブルで条件検索
    - WHERE c = 500000
    - WHERE c BETWEEN 100000 AND 200000
    - WHERE c BETWEEN 100000 AND 600000



トランザクション巻き戻し - pg_rewind (1)

- 昇格時点より先に進んだデータベースクラスタの問題





再度ベースバックアップ取得をするしかなかった

トランザクション巻き戻し - pg_rewind (2)

- pg_rewind コマンド
  - プライマリとしたいサーバのタイムライン切り替え時点まで、ローカル側のデータベースクラスタのトランザクションを巻き戻す

```
[server1]$ pg_ctl start
[server1]$ pg_ctl stop
[server1]$ pg_rewind -D $PGDATA  \
           --source-server='host=server2 port=5432 user=postgres'
[server1]$ vi $PGDATA/recovery.conf
[server1]$ pg_ctl start
```

クラッシュ終了後なら、起動・停止をしておく

```
[server3]$ pg_ctl stop
[server3]$ pg_rewind -D $PGDATA \
           --source-server='host=server2 port=5432 user=postgres'
[server3]$ vi $PGDATA/recovery.conf
[server3]$ pg_ctl start
```

必要な設定パラメータ
(postgresql.conf)

full_page_writes = on（デフォルト）
wal_log_hints = on　（デフォルトは off）

挿入または更新 - INSERT ... ON CONFLICT (1)

- よくある処理パターン
  - 新データ[id=101 hn='TARO' ml='taro@example.org']につき
    id = 101 のレコードが無いなら INSERT
    id = 101 のレコードが在るなら UPDATE
  - 以下のように記述できる

```
INSERT INTO t3 VALUES (101, 'TARO', 'taro@example.org')
  ON CONFLICT (id)
  DO UPDATE SET hn = EXCLUDED.hn, ml = EXCLUDED.ml;
```

  - 従来はアプリ側で場合分けを実装
    - あるいは以下のような WITH構文による技巧により実現

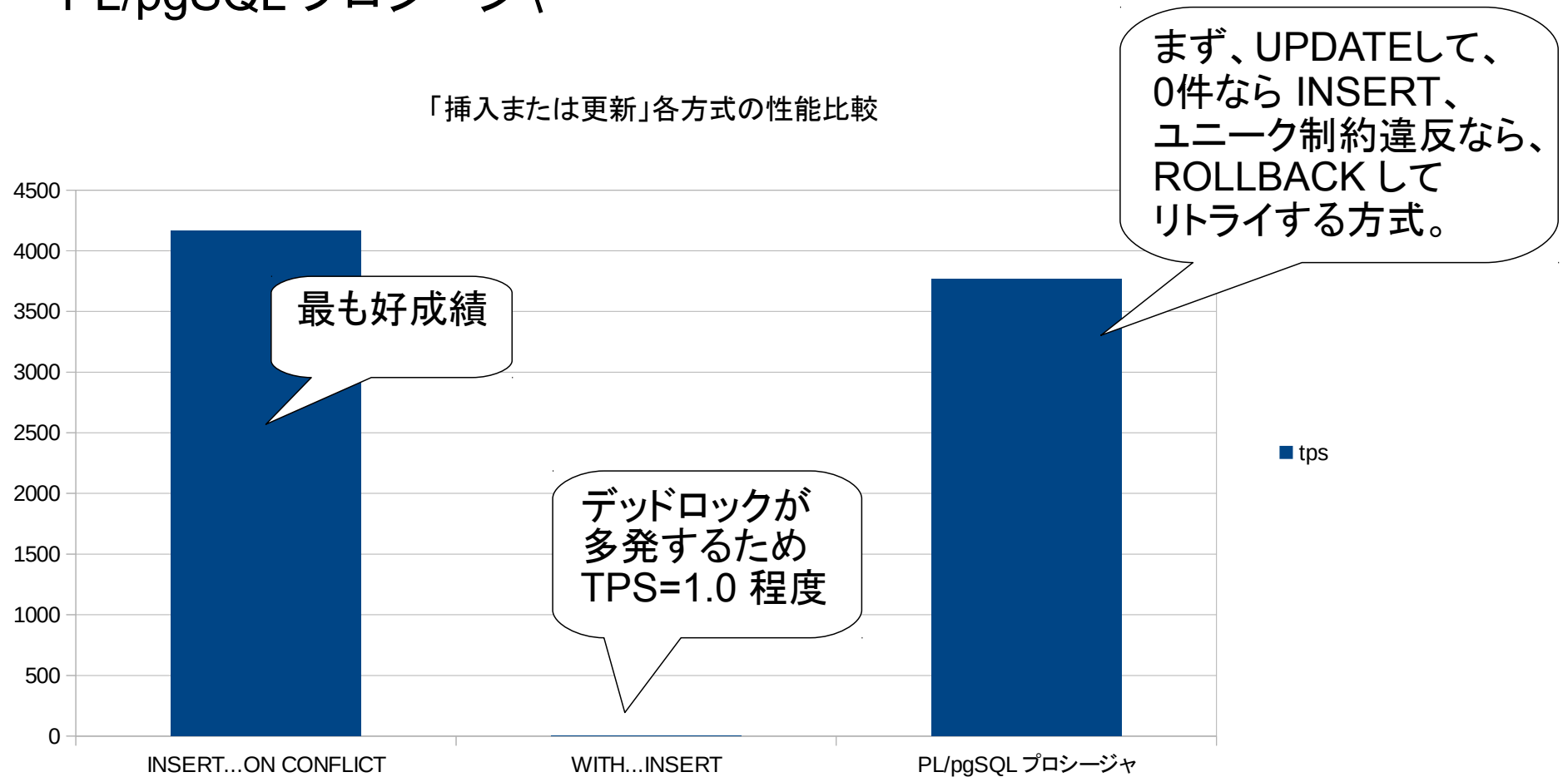
```
WITH val AS (SELECT ((101, 'TARO', 'taro@example.org')::t3).*),
     upd AS (UPDATE t3 SET hn = val.hn, ml='taro@example.org'
                FROM val RETURNING t3.id)
INSERT INTO t3
  SELECT * FROM val WHERE id NOT IN (SELECT id FROM upd);
```

挿入または更新 - INSERT ... ON CONFLICT (2)

- 3つの方式を pgbench で比較
  - INSERT...ON CONFLICT
  - WITH...INSERT
  - PL/pgSQL プロシージャ

同時 500接続で「挿入または更新」を実行

```
pgbench -f test1.sql -c 500 -t 100 -n db1
```



外部テーブルのインポート

- IMPORT FOREIGN SCHEMA 命令が追加
  - 外部テーブルをスキーマ単位で一括登録できる

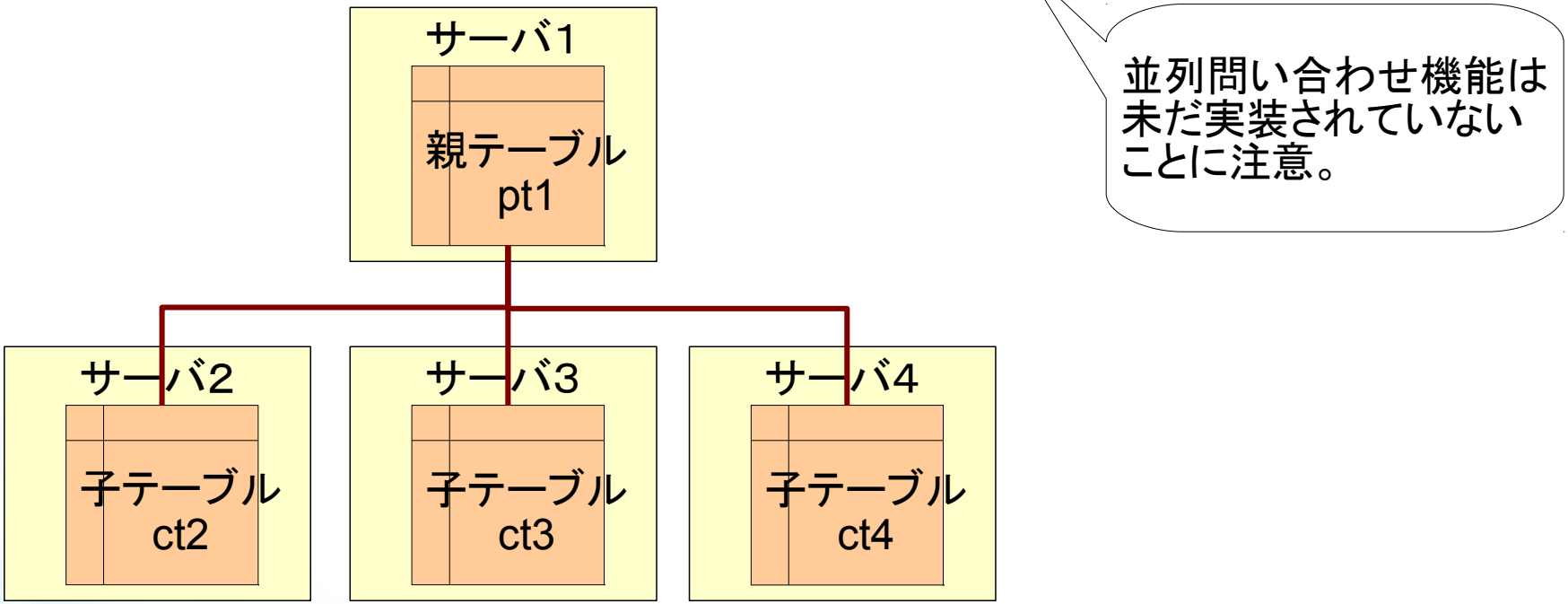
```
db1=> CREATE SCHEMA remote;
db1=> IMPORT FOREIGN SCHEMA apscm
        FROM SERVER foreign_server INTO remote;

db1=> \dE remote.*
                          List of relations
 Schema |     Name     |       Type        | Owner
--------+--------------+-------------------+-------
 remote | customers    | foreign table     | apuser
 remote | products     | foreign table     | apuser
...
```

  - 従来はテーブル毎に CREATE FOREIGN TABLE が必要であった
    - 各カラムの対応するデータ型をデフォルト通りでなく個別に設定するなら、引き続き、CREATE FOREIGN TABLE を使うことになる

外部テーブルの継承

- 外部テーブルとローカルテーブルで継承が可能に
  - パーティショニングと分散格納の組み合わせが可能になる
    - 従来から、テーブル継承はテーブルパーティショニング実現に利用されていた



コマンド例： (s1db1 と s2db1 は各々リモートテーブル用のスキーマ)

```
s2db1=> CREATE TABLE ct2 () INHERITS (s1db1.pt1);

s1db1=> ALTER TABLE s2db1.ct2 INHERIT pt1;
```

集計構文

- 以下の集計構文に対応
  GROUP BY GROUPING SETS (...)
  GROUP BY ROLLUP (...)
  GROUP BY CUBE (...)
- 各カテゴリ組み合わせ毎の小計を一度に出力させることができる
- SQL99 に規定されており、他DBMS の実装と大きな違いはない

テーブル:t_sales

customer	product	num
Tom	egg	10
Tom	apple	20
Jhon	egg	15
Jhon	egg	5

CUBE (c1, c2, ..) :
  指定カラムの全組み合わせ

ROLLUP (c1, c2, ...) :
  指定カラム順に包含する組み合わせ

```
db1=> SELECT customer,product,sum(num)
  FROM t_sales
  GROUP BY GROUPING SETS
  ((customer), (customer, product), ());

 customer | product | sum
----------+---------+-----
 John     | egg     |  20
 John     |         |  20
 Tom      | apple   |  20
 Tom      | egg     |  10
 Tom      |         |  30
          |         |  50
(6 rows)
```

JSONB型むけ関数・演算子の拡充(1)

- JSONB データの部分更新が容易になる
  - 項目が無ければ追加、あれば右辺値で上書きする。

```
db1=> SELECT '{"name": "John", "age": 25}'::jsonb
          || '{"age": 26}'::jsonb;
          ?column?
------------------------------
 {"age": 26, "name": "John"}
```

  - JSONB のオブジェクト要素、配列要素を削除

```
db1=> SELECT '{"name": "John", "age": 25}'::jsonb - 'age';
     ?column?
------------------
 {"name": "John"}

db1=> SELECT '["apple", "orange", "melon"]'::jsonb - 1;
      ?column?
--------------------
 ["apple", "melon"]
```

JSONB型むけ関数・演算子の拡充(2)

- 深い位置にある要素の削除

```
db1=> SELECT '{"name": "John", "contact": {"phone": "0123",
                "email":"John@foohoge.com"}}'::jsonb
               #- '{contact, email}'::text[];
                     ?column?
-----------------------------------------------
 {"name": "John", "contact": {"phone": "0123"}}
```

- 指定パスに値があるかに応じた動作
  - 第4引数 true なら「あれば値の更新」「無ければ挿入」
  - 第4引数 false なら「あれば値の更新」「無ければ何もしない」

```
db1=> SELECT jsonb_set('{"name": "James", "contact":
                          {"phone": "01234 567890",
                           "fax": "01987543210"}}'::jsonb,
            '{contact,skype}',
            '"myskypeid"'::jsonb,
            true);
```

まとめ

- PostgreSQL開発の概要についてお話いたしました。
- PostgreSQL 9.5 の新機能について紹介いたしました。
  - 各機能の詳細については、9.5 alpha2版のマニュアル（英語）を参照いただくことができます。
  - SRA OSS, Inc. Webサイトの技術情報ページで弊社による検証レポートも今月中には公開予定です。

ご清聴ありがとうございました。
ご質問を承ります。

オープンソースとともに

URL：http://www.sraoss.co.jp/
E-mail：sales@sraoss.co.jp
Tel：03-5979-2701