SRA OSS, INC.

PostgreSQLの最新情報

日本PostgreSQLユーザ会
石井 達夫

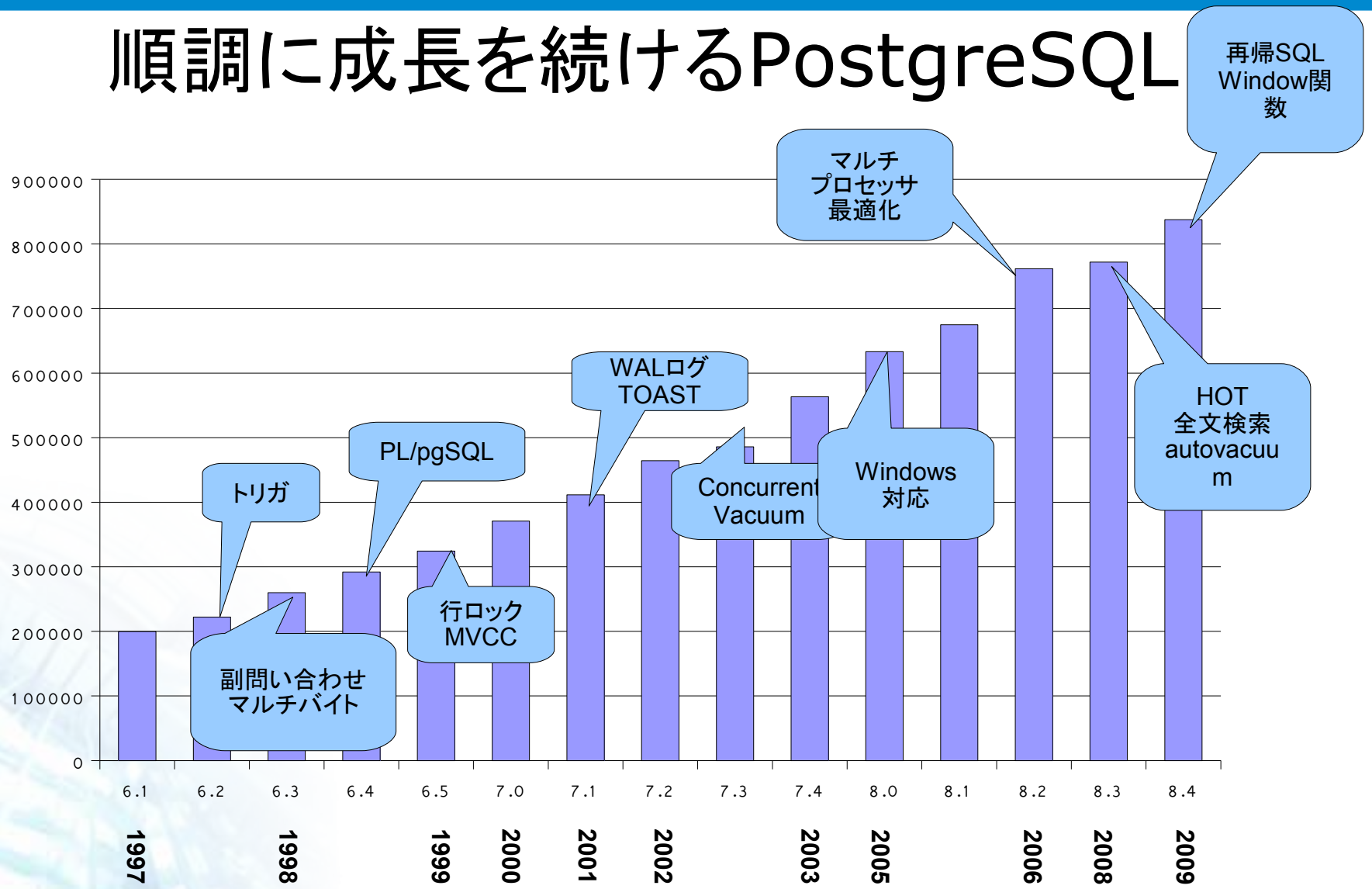
すみません. カナダに行けませんでした

コンファレンスについて

- スケジュールはこちら
  - http://www.pgcon.org/2009/schedule/
- 今回の目玉は右の方だったようです
- “How to Get Your PostgreSQL Patch Accepted”


Tom Lane Rejected My Patch and
All I Got Was This Stupid T-Shirt
In short: -1 from me.
regards, tom lane

順調に成長を続けるPostgreSQL



PostgreSQLの主要な機能

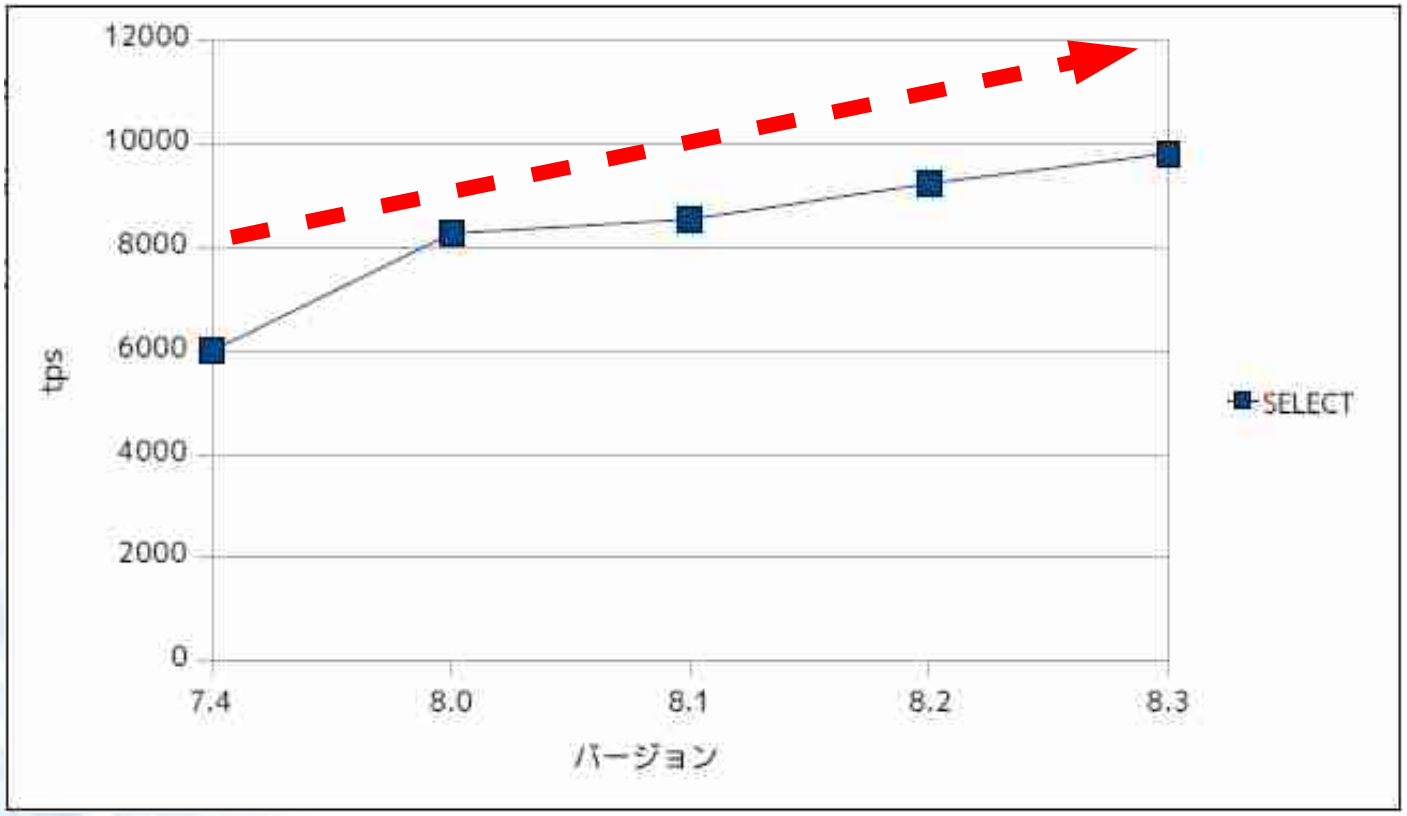
- 標準装備の機能
  - 行ロック
  - 読み取り一貫性(MVCC)
  - コストベース・オプティマイザ
  - パーティショニング
  - ストアドプロシジャ，トリガ
  - オンライン・バックアップ
  - アーカイブログ
  - 全文検索
- オプション機能
  - レプリケーション，クラスタリング，GIS対応

この5年間でPostgreSQLの性能はどれだけ向上したか？

- PostgreSQL 7.4 (2003年)から PostgreSQL8.3 (2008年)の間に...
  - 検索性能164%向上
  - 一括ロード性能200%向上
  - 更新性能300%向上

検索性能の向上

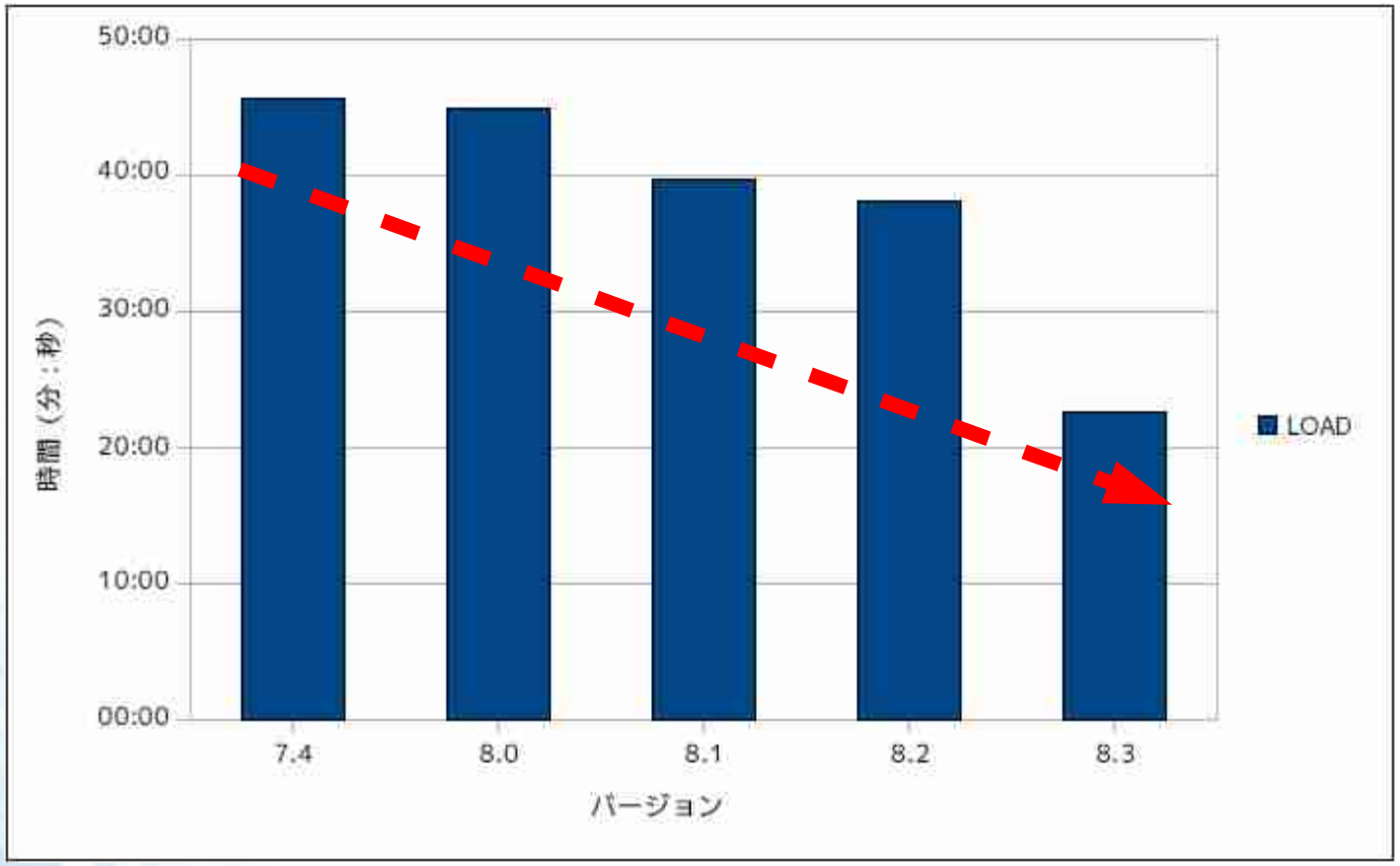
7.4(2003/11)から8.3(2008/2)の間に**164%**の性能向上



一括ロード性能の向上

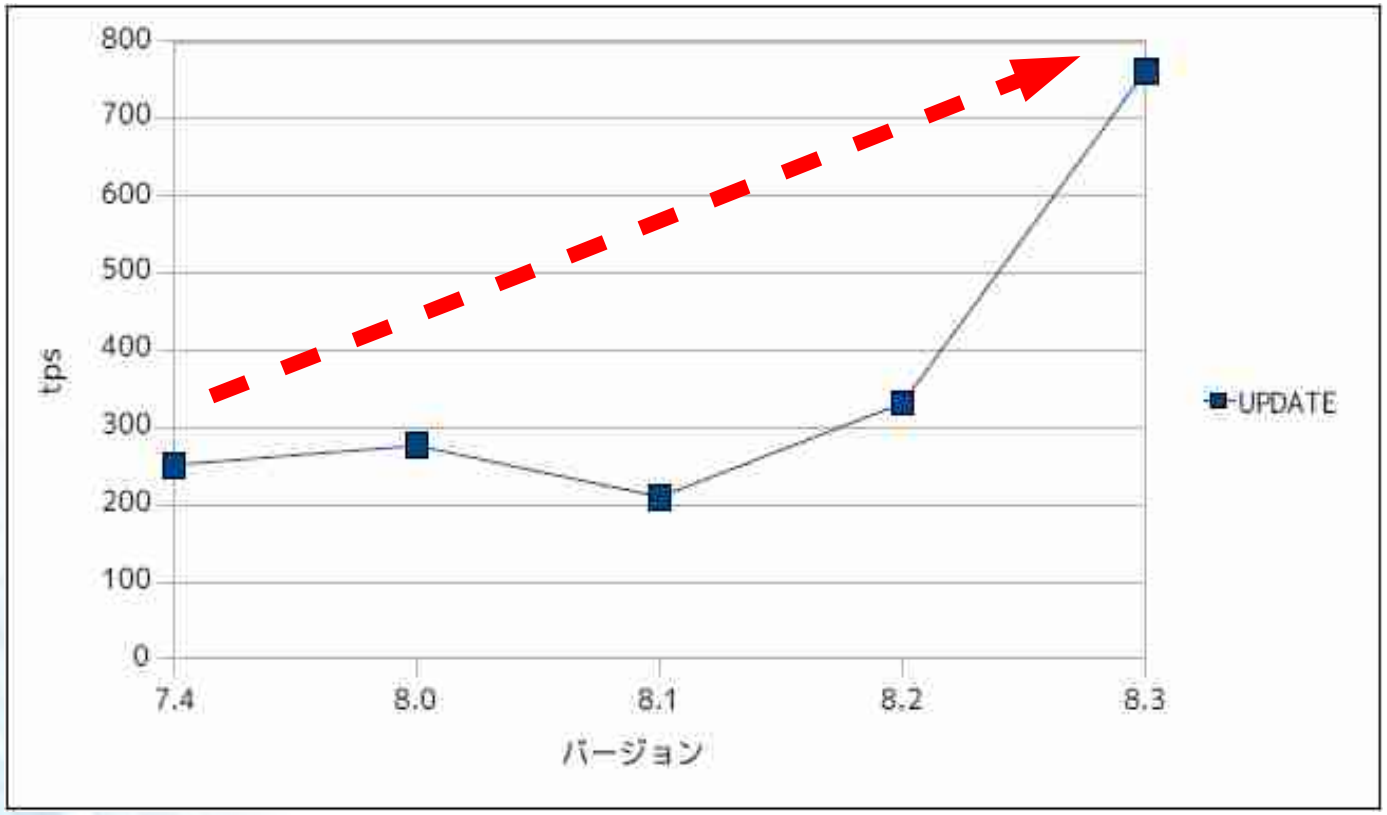
7.4(2003/11)から8.3(2008/2)の間に**200%**の性能向上

1億件(15GB)のロード



更新性能の向上

7.4(2003/11)から8.3(2008/2)の間に**300%**の性能向上



8.4注目の新機能：再帰SQLのサポート

- リスト，木構造などのデータ構造から再帰的にデータを取得できる
  - 今はアプリ側で処理するか，PL/pgSQLなどの関数を使わなければならず，不便で効率も悪かった
- 住友電工情報システムとSRA OSSの協力により開発
  - ユーザ企業が積極的にPostgreSQLの開発に関与する新しい形態(エコシステム)
- 実装はSQL標準の共通SQL式(Common Table Expression: CE)句を採用

単純な再帰SQL

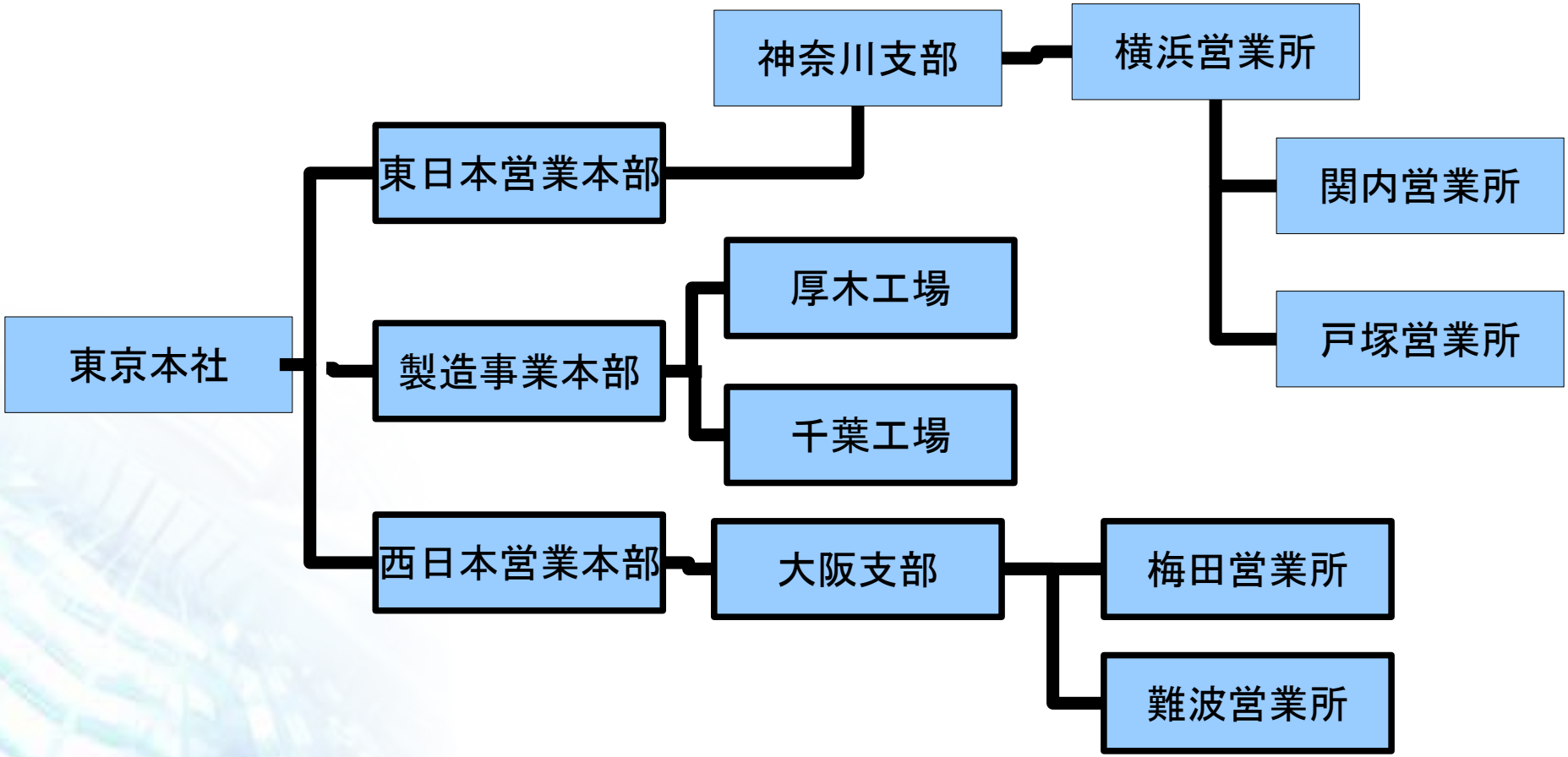
```
WITH RECURSIVE t(n) AS (
    VALUES (1) – 非再帰項

UNION ALL

    SELECT n+1 FROM t WHERE n < 100 -- 再帰項
)
SELECT sum(n) FROM t;

 sum
------
 5050
```

サンプルデータ



RDBでの表現

事業所名	管轄事業所名	人員数
東京本社		１００
東日本営業本部	東京本社	５０
神奈川支部	東日本営業本部	５０
横浜営業所	神奈川支部	２０
関内営業所	横浜営業所	１５
戸塚営業所	横浜営業所	１０
製造事業本部	東京本社	５０
厚木工場	製造事業本部	２００
千葉工場	製造事業本部	２００
西日本営業本部	東京本社	５０
大阪支部	西日本営業本部	５０
梅田営業所	大阪支部	２０
難波営業所	大阪支部	２０

東日本営業本部以下の人員数を求める

再帰SQLの定義

```
WITH RECURSIVE 東日本営業人員(事業所名, 人員数) AS
 (SELECT 事業所名, 人員数 FROM 人員構成表
  WHERE 事業所名 = '東日本営業本部'

  UNION ALL

  SELECT PEN.事業所名, PEN.人員数 FROM 人員構成表 AS
PEN,
    東日本営業人員 AS EBP
    WHERE PEN.管轄事業所名 = EBP.事業所名)

SELECT SUM(人員数) FROM 東日本営業人員;
```

SELECT本体

組織階層の「深さ」を表示

```
WITH RECURSIVE 東日本営業人員(事業所名, レベル, 人員数) AS
  (SELECT 事業所名, 1, 人員数 FROM 人員構成表
    WHERE 事業所名 = '東日本営業本部'

   UNION ALL

   SELECT PEN.事業所名, EBP.レベル+1, PEN.人員数 FROM 人員構成表 AS
PEN,
東日本営業人員 AS EBP
    WHERE PEN.管轄事業所名 = EBP.事業所名)

SELECT * FROM 東日本営業人員;
```

事業所名	レベル	人員数
東日本営業本部	1	50
神奈川支部	2	50
横浜営業所	3	20
関内営業所	4	15
戸塚営業所	4	10

(5 rows)

PostgreSQLの新機能：Window関数

- SQL標準のデータ解析関数
  - 順序付けなどに利用
  - 行を集約しない(集約関数の一種ではない)
- 主な関数
  - row_number: 行番号
  - rank, dense_rank, percent_rank, cume_dist: 順位
  - ntile: N個のグループに分類
  - first_value, last_value, nth_vaue: ある順位のデータ
- OVER
  - 関数() OVER(PARTITION BY...): 区間に分割
  - 関数() OVER(ORDER BY...): 区間毎にソート

Window関数: row_number

```
test=# SELECT row_number() OVER (), * FROM empsalary ;
 row_number |  depname  | empno | salary | enroll_date
------------+-----------+-------+--------+-------------
          1 | develop   |    10 |   5200 | 2007-08-01
          2 | sales     |     1 |   5000 | 2006-10-01
          3 | personnel |     5 |   3500 | 2007-12-10
          4 | sales     |     4 |   4800 | 2007-08-08
          5 | personnel |     2 |   3900 | 2006-12-23
          6 | develop   |     7 |   4200 | 2008-01-01
          7 | develop   |     9 |   4500 | 2008-01-01
          8 | sales     |     3 |   4800 | 2007-08-01
          9 | develop   |     8 |   6000 | 2006-10-01
         10 | develop   |    11 |   5200 | 2007-08-15
(10 rows)
```

出力結果に対して，1から連番を振る

Window関数: rank

```
test=# select * from empsalary order by empno;
  depname   | empno | salary | enroll_date
------------+-------+--------+-------------
 sales      |     1 |   5000 | 2006-10-01
 personnel  |     2 |   3900 | 2006-12-23
 sales      |     3 |   4800 | 2007-08-01
 sales      |     4 |   4800 | 2007-08-08
 personnel  |     5 |   3500 | 2007-12-10
 develop    |     7 |   4200 | 2008-01-01
 develop    |     8 |   6000 | 2006-10-01
 develop    |     9 |   4500 | 2008-01-01
 develop    |    10 |   5200 | 2007-08-01
 develop    |    11 |   5200 | 2007-08-15
(10 rows)
```

元データ

```
test=# SELECT depname, empno, salary, rank() OVER (PARTITION BY depname ORDER BY salary DESC) FROM empsalary;
  depname   | empno | salary | rank
------------+-------+--------+------
 develop    |     8 |   6000 |    1
 develop    |    10 |   5200 |    2
 develop    |    11 |   5200 |    2
 develop    |     9 |   4500 |    4
 develop    |     7 |   4200 |    5
 personnel  |     2 |   3900 |    1
 personnel  |     5 |   3500 |    2
 sales      |     1 |   5000 |    1
 sales      |     4 |   4800 |    2
 sales      |     3 |   4800 |    2
(10 rows)
```

部門の中で，給料の高い順に並べる

Window関数: dense_rank

```
test=# SELECT depname, empno, salary,
dense_rank() OVER (PARTITION BY depname ORDER BY salary DESC)
 FROM empsalary;
  depname   | empno | salary | dense_rank
------------+-------+--------+------------
 develop    |     8 |   6000 |          1
 develop    |    10 |   5200 |          2
 develop    |    11 |   5200 |          2
 develop    |     9 |   4500 |          3
 develop    |     7 |   4200 |          4
 personnel  |     2 |   3900 |          1
 personnel  |     5 |   3500 |          2
 sales      |     1 |   5000 |          1
 sales      |     4 |   4800 |          2
 sales      |     3 |   4800 |          2
(10 rows)
```

rankと違って，順位番号の抜けがない

Window関数: percent_rank

```
test=# SELECT depname, empno, salary,
percent_rank() OVER (PARTITION BY depname ORDER BY salary DESC)
FROM empsalary;
  depname  | empno | salary | percent_rank
-----------+-------+--------+--------------
 develop   |     8 |   6000 |            0
 develop   |    10 |   5200 |         0.25
 develop   |    11 |   5200 |         0.25
 develop   |     9 |   4500 |         0.75
 develop   |     7 |   4200 |            1
 personnel |     2 |   3900 |            0
 personnel |     5 |   3500 |            1
 sales     |     1 |   5000 |            0
 sales     |     4 |   4800 |          0.5
 sales     |     3 |   4800 |          0.5
(10 rows)
```

順位を割合（パーセント）で表します

Window関数: ntile

```
test=# SELECT depname, empno, salary,
ntile(3) OVER (PARTITION BY depname ORDER BY salary DESC)
FROM empsalary;
  depname   | empno | salary | ntile
------------+-------+--------+-------
 develop    |     8 |   6000 |     1
 develop    |    10 |   5200 |     1
 develop    |    11 |   5200 |     2
 develop    |     9 |   4500 |     2
 develop    |     7 |   4200 |     3
 personnel  |     2 |   3900 |     1
 personnel  |     5 |   3500 |     2
 sales      |     1 |   5000 |     1
 sales      |     4 |   4800 |     2
 sales      |     3 |   4800 |     3
(10 rows)
```

部門毎に給与を
できるだけ
3分割します

Window関数: cume_dist

```
test=# SELECT depname, empno, salary FROM
 (SELECT *, cume_dist() OVER (ORDER BY salary DESC) AS rank FROM
empsalary)
 AS foo WHERE rank <= 0.3;
 depname | empno | salary
---------+-------+--------
 develop |     8 |   6000
 develop |    10 |   5200
 develop |    11 |   5200
(3 rows)
```

cume_distは「累積分散（パーセンタイル）」を表します．上の問合わせは，給与の上位3割を示します．

Window関数：集約関数との組み合わせ

```
SELECT depname, empno, salary, sum(salary)
OVER w FROM empsalary WINDOW w AS (PARTITION BY
depname);
  depname  | empno | salary |  sum
-----------+-------+--------+-------
 develop   |    11 |   5200 | 25100
 develop   |     7 |   4200 | 25100
 develop   |     9 |   4500 | 25100
 develop   |     8 |   6000 | 25100
 develop   |    10 |   5200 | 25100
 personnel |     5 |   3500 |  7400
 personnel |     2 |   3900 |  7400
 sales     |     3 |   4800 | 14600
 sales     |     1 |   5000 | 14600
 sales     |     4 |   4800 | 14600
(10 rows)
```

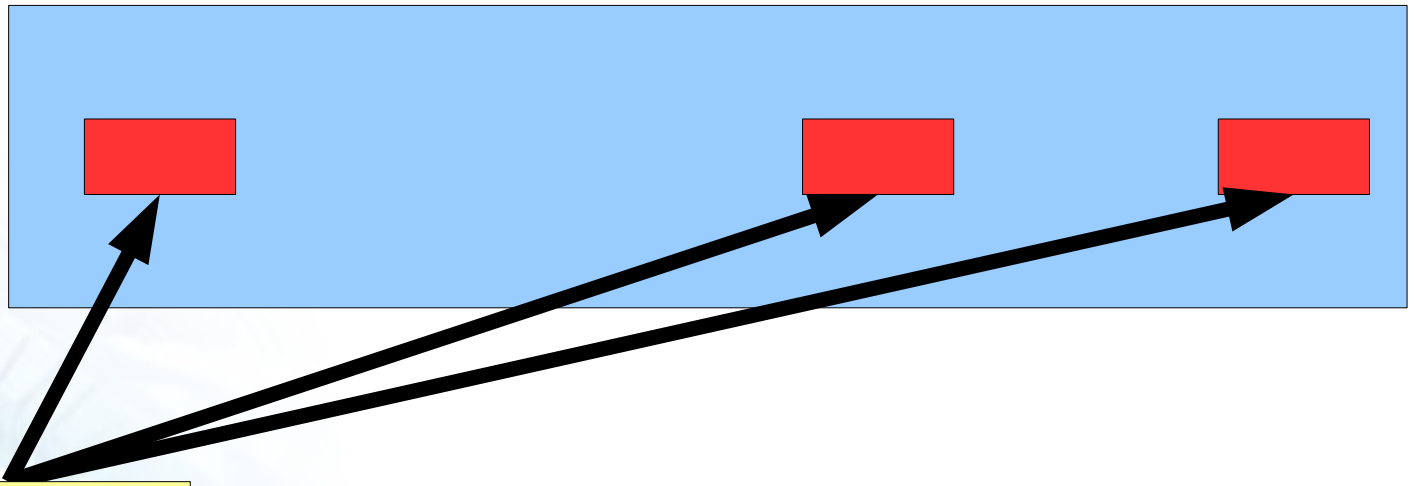
sumは部門内の給料の合計を示します

PostgreSQL 8.4: その他の改良

- VACUUMの高速化
  - Visibility Mapにテーブルの不用領域の位置を記録し，無駄なテーブルのスキャンを防ぐ
- Free Space Map(FSM)のオンディスク化
  - ディスク上にFSMを持つことにより，FSMの溢れが発生しない->追跡不能な再利用可能領域が発生しなくなる
- ディスク先読みの並列実行
  - effective_io_concurrencyで並列度を指定(=ドライブの物理台数)
  - 一部プランのみ．Linux/UNIXで利用可能
- 並列リストア
  - DBのリストア処理を並列に実行できるようになった

VACUUMの効率化 (Visibility Map)

8.3以前では，テーブルをすべて読まないと，不用領域を見つけることができなかった



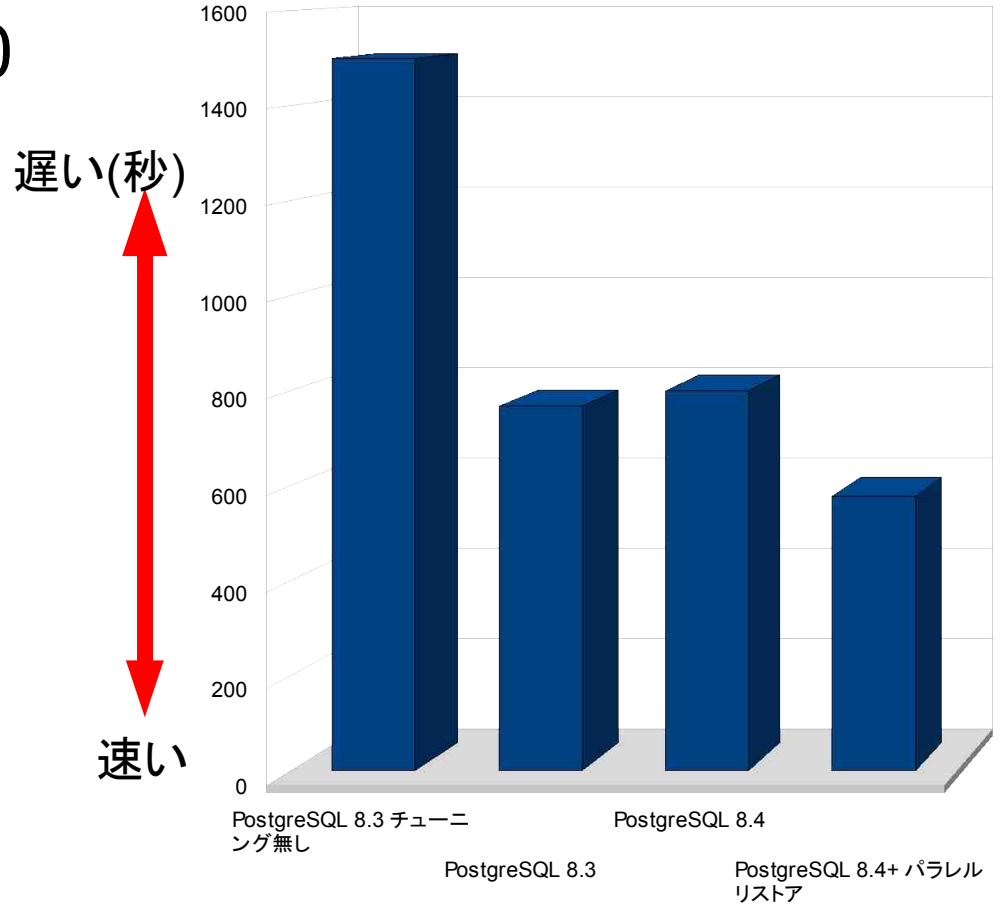
8.4では，Visibility Mapを使って必要なところだけを読んで不要領域を発見できる

PostgreSQL 8.4: その他の改良

- 列単位でのアクセス権限設定
- 可変引数関数，デフォルト引数関数
- 前方一致による全文検索
- ハッシュインデックスの高速化
- SELECT DISTINCTの高速化(GROUP BYへの書き換え必要なし)
- EXISTS, NOT EXISTSの高速化
- 統計情報ファイルの指定が可能（メモリファイル利用による高速化も可能)
- デッドロックの際に原因となったSQLを表示
- 関数単位の実行回数，実行時間のログ

パラレルリストアの効果

- PostgreSQL 8.4(4/10 版)
- DBサイズ2.7GB
- チューニングあり
- チューニング内容
  - shared_buffers = 256MB
  - wal_buffers = 256
  - check_point_segments = 16



pgpool-II 2.2リリース！

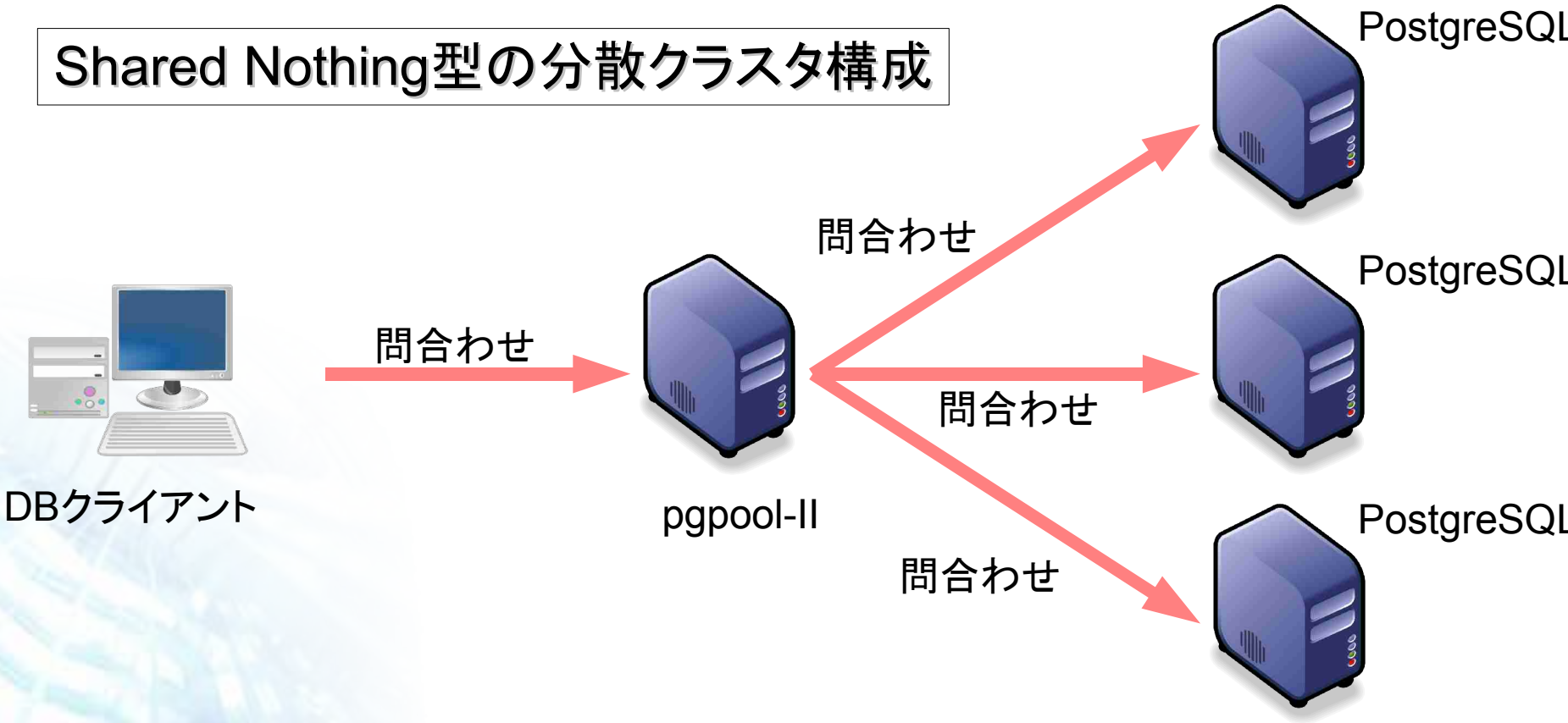
- pgpool-IIとは
  - PostgreSQL専用のオープンソースクラスタソフト
  - PostgreSQL 7.4から8.3まで対応
  - レプリケーション，負荷分散，パラレルクエリなどの機能
  - 管理用GUIツールあり
  - 20万のダウンロード実績
  - 他のレプリケーションソフトとの連携も可能
    - Slony-I, warm standbye
- pgpool-II 2.2
  - SERIALIZABLEトランザクションへの対応
  - オンラインリカバリの改善
  - 信頼性の向上

pgpool-IIを導入するメリット

- レプリケーションによりデータのリアルタイムバックアップ
- 故障DBの自動切り放し
- HAよりも短いフェイルオーバ時間
- 同期型レプリケーションなので，複数サーバ間でデータの一時的なずれが起きない
- 負荷分散機能により，検索系の性能アップが可能
- PostgreSQLアプリケーションの修正は最小限
- 運用を止めずにDBの切り離し，復帰，追加が可能
- PostgreSQLの可用性，性能を向上可能

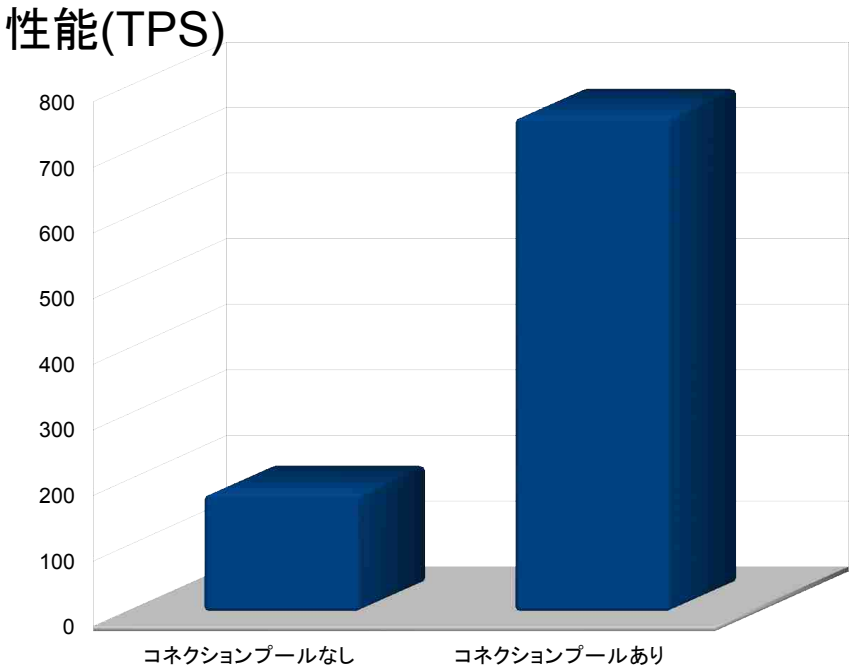
pgpool-IIのアーキテクチャ

Shared Nothing型の分散クラスタ構成



コネクションプールの効果

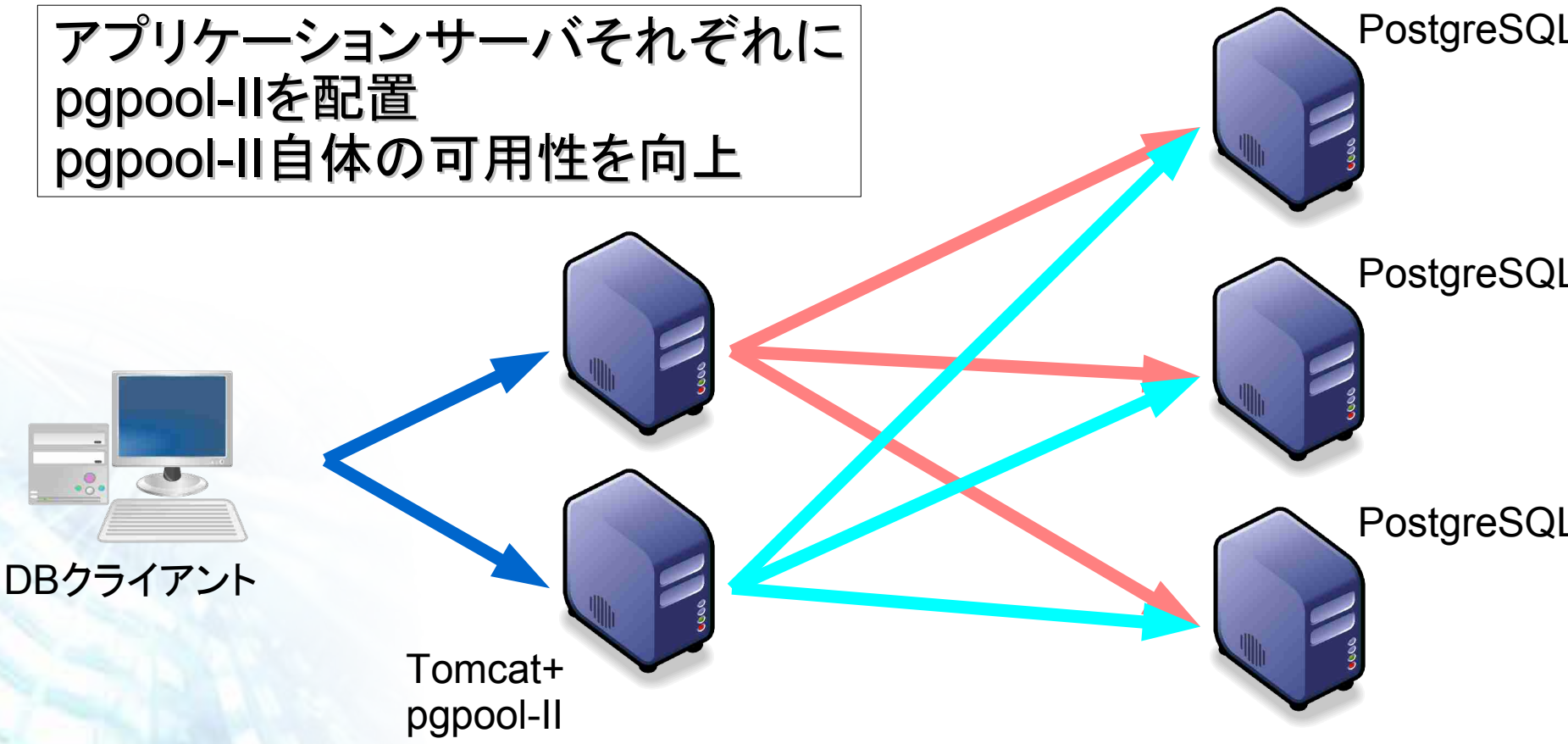
- DBクライアントとpgpool-IIの間の接続が切れても，pgpool-IIとDBの間のコネクションを維持
  - 次回DBクライアントからの接続があったときに再利用するので性能が向上
- コネクションプールをしない場合に比べ，数倍の性能向上が見られるケースも



pgbenchを使い，検索問合わせを10000回実行した結果
データ件数: 10万件
ハードウェア: Centrino 1.1GHz
ソフトウェア: Linux 2.6/PostgreSQL 8.3/pgpool-II 2.1

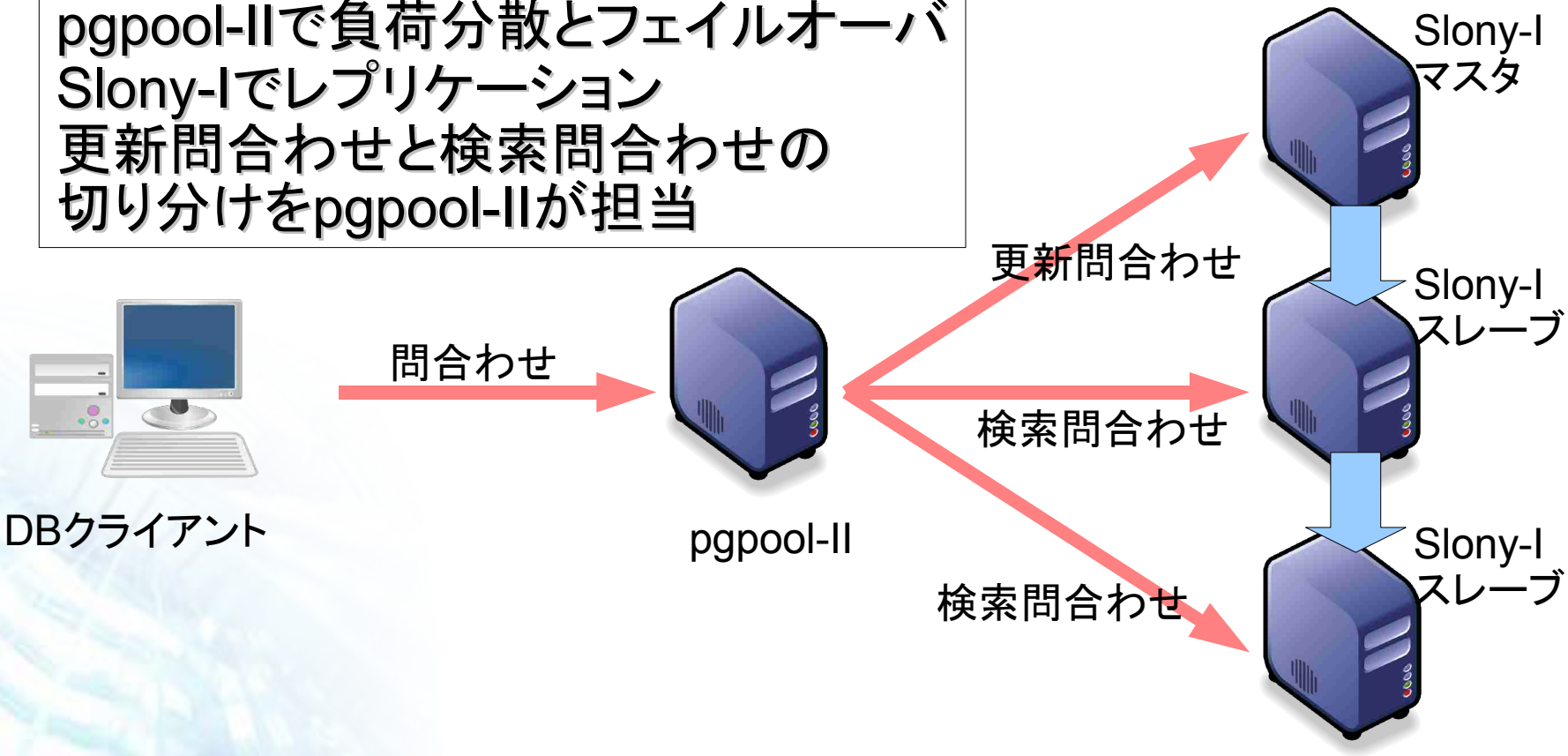
pgpool-IIの構成例（1）

アプリケーションサーバそれぞれに
pgpool-IIを配置
pgpool-II自体の可用性を向上



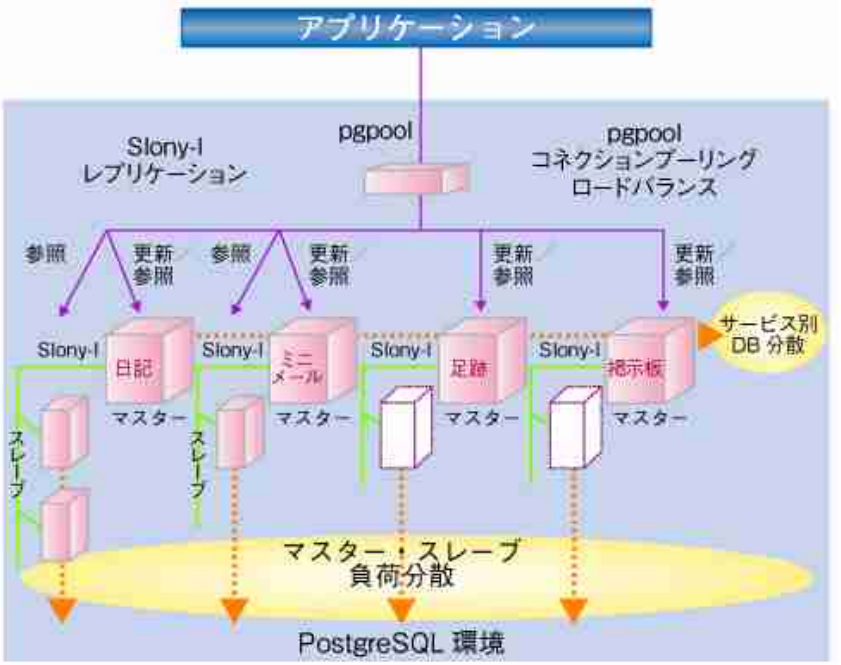
pgpool-IIの構成例（2）

pgpool-IIで負荷分散とフェイルオーバ
Slony-Iでレプリケーション
更新問合わせと検索問合わせの
切り分けをpgpool-IIが担当



pgpool-II事例

- オープンドア様事例
- 国内最大級規模の携帯電話向けSNS/ゲームサイト
- 月間4億PV，トランザクションの2割が更新系
- pgpool-II + Slony-Iで20台以上のPostgreSQLを管理．スケールアウトする大規模システムを構築



http://members.techtarget.itmedia.co.jp/tt/members/0802/28/news01.html

ITMedia techTarget記事より引用

pgpool-IIの関連情報

- pgpool-II開発サイト
  - http://pgfoundry.org/projects/pgpool/
- pgpool-II日本語メーリングリスト
  - http://www.sraoss.jp/mailman/listinfo/pgpool-general-jp
- 「PostgreSQLには絶対pgpool-II」(Think IT)
  - http://www.thinkit.co.jp/article/98/1/

ご清聴ありがとうございました