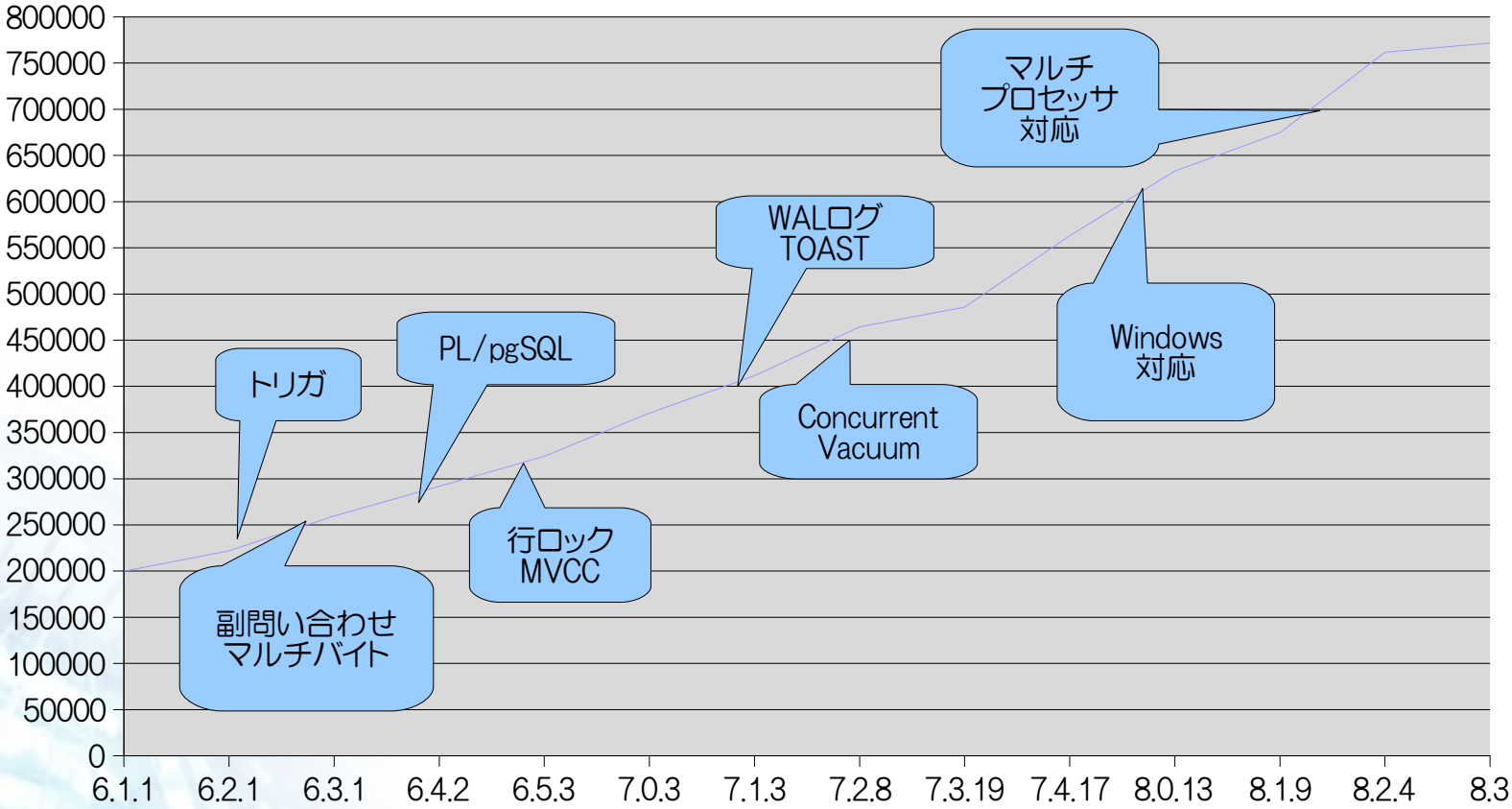
SRA OSS, INC.

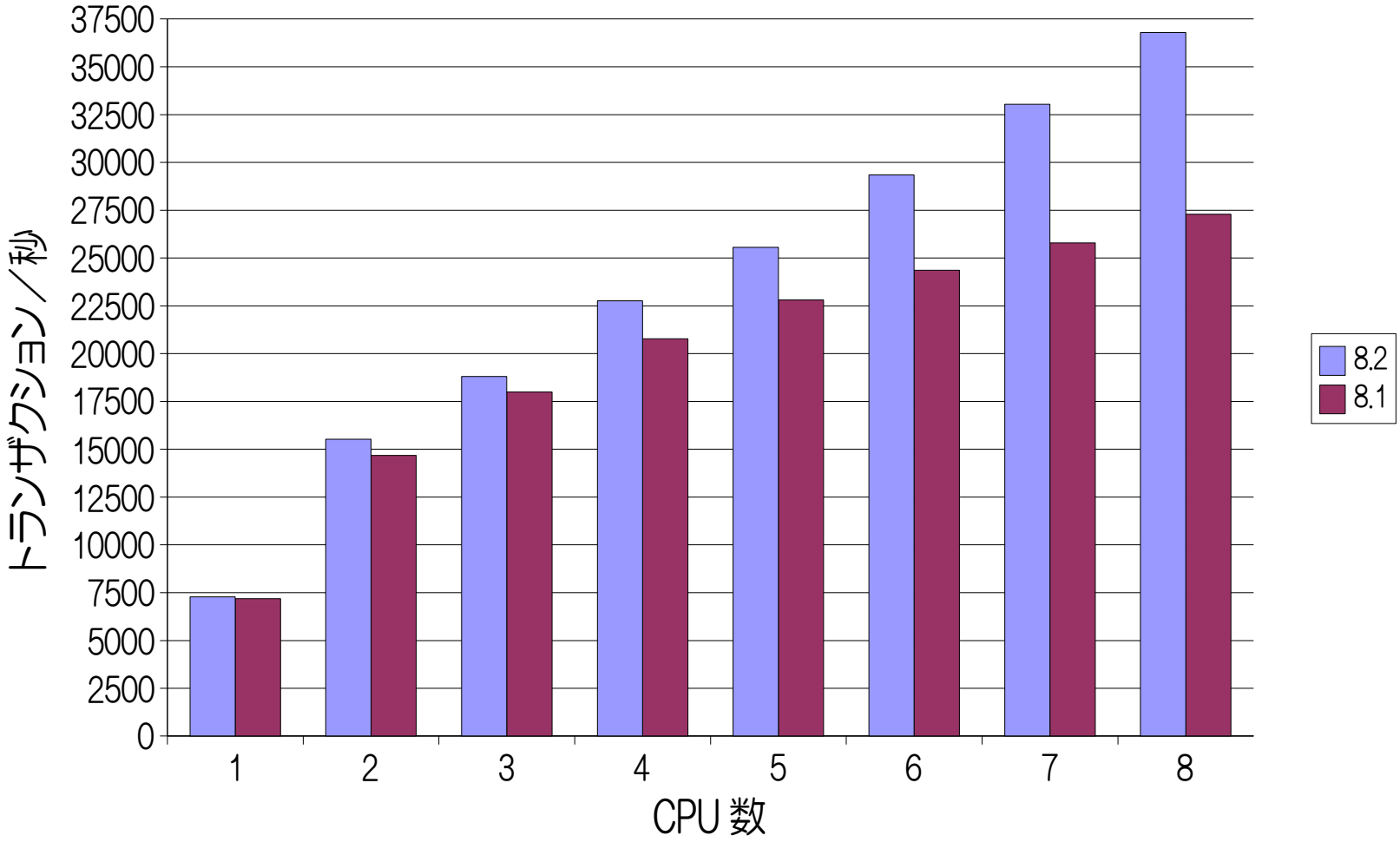
PostgreSQL最新情報
－ 待望のPostgreSQL8.3完全ガイド －

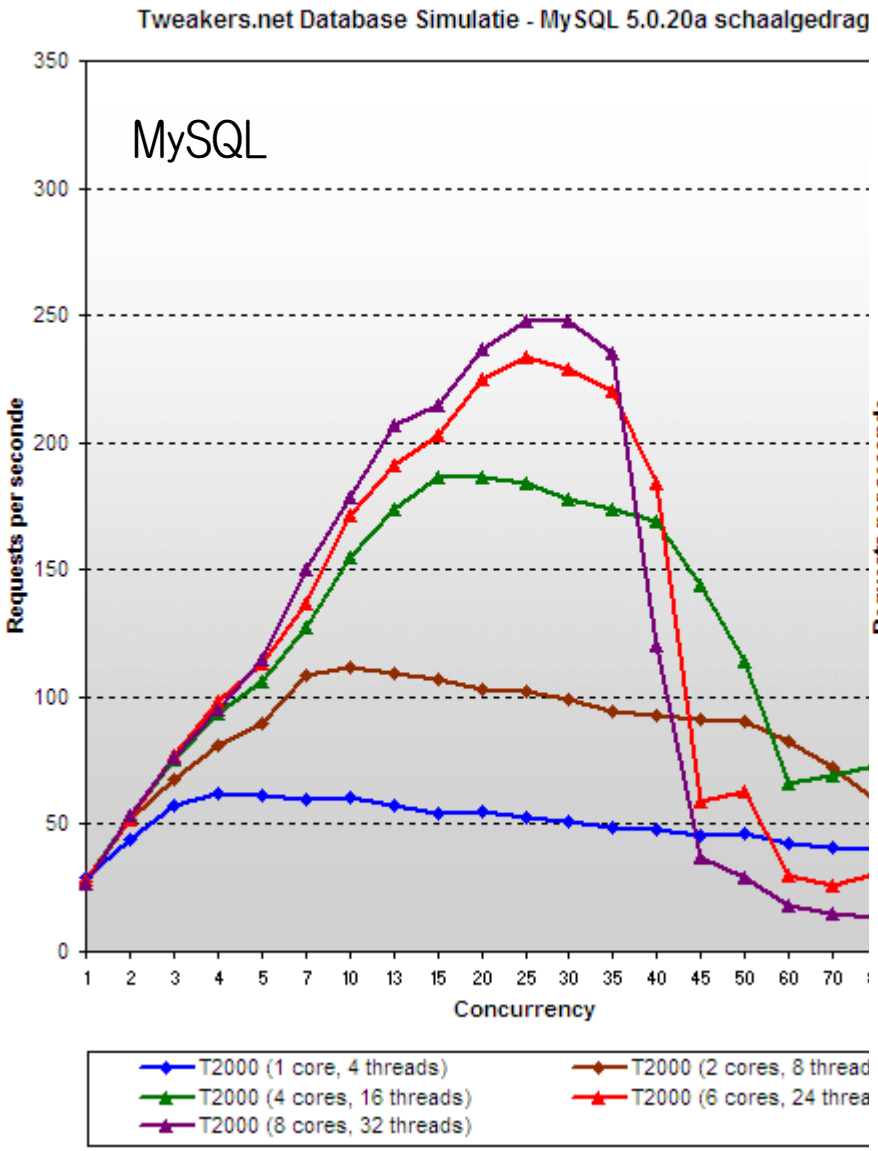
SRA OSS, Inc. 日本支社
石井 達夫

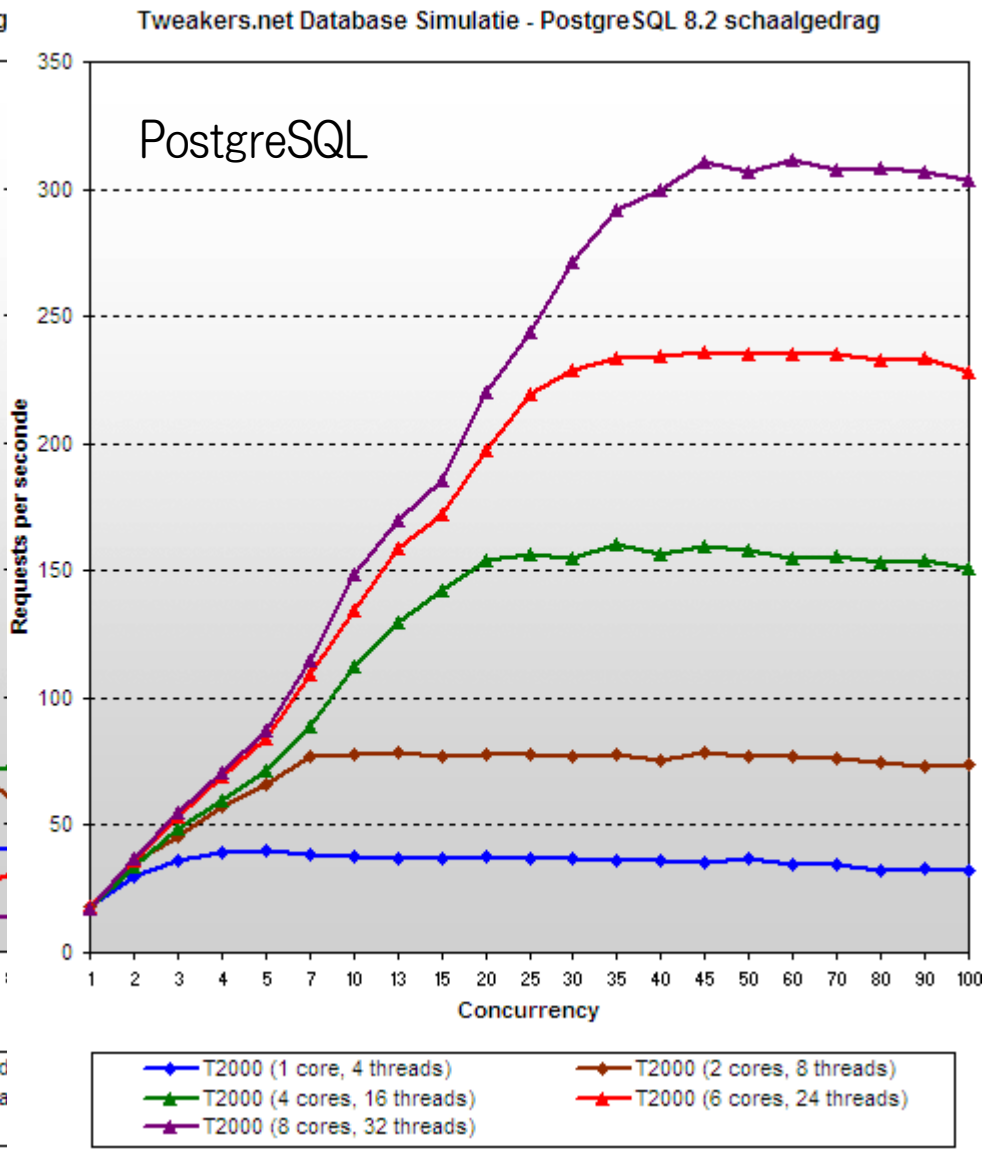
PostgreSQLの歴史



PostgreSQL 8.1 と 8.2 の CPU スケーラビリティの比較





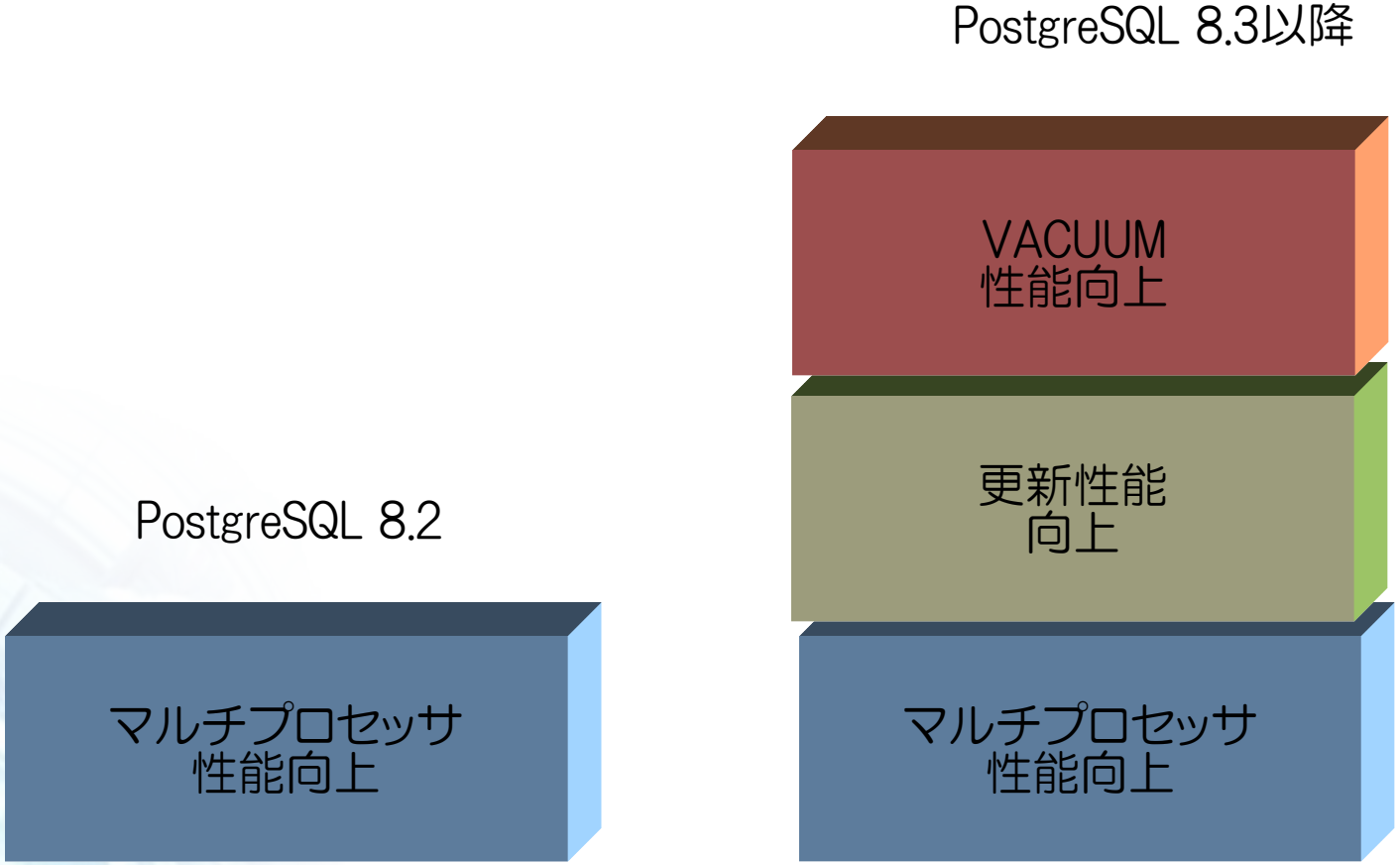


http://tweakers.net/より引用

PostgreSQL 8.3でどこが良くなっている？

- 性能
  - 更新性能の改善
  - 順スキャンの改善
  - 特定のSQLの高速化
- 管理機能
  - ソート処理のモニタリング
  - インデックスアドバイザー
  - ログ項目の追加
- SQL機能
  - 更新可能カーソル

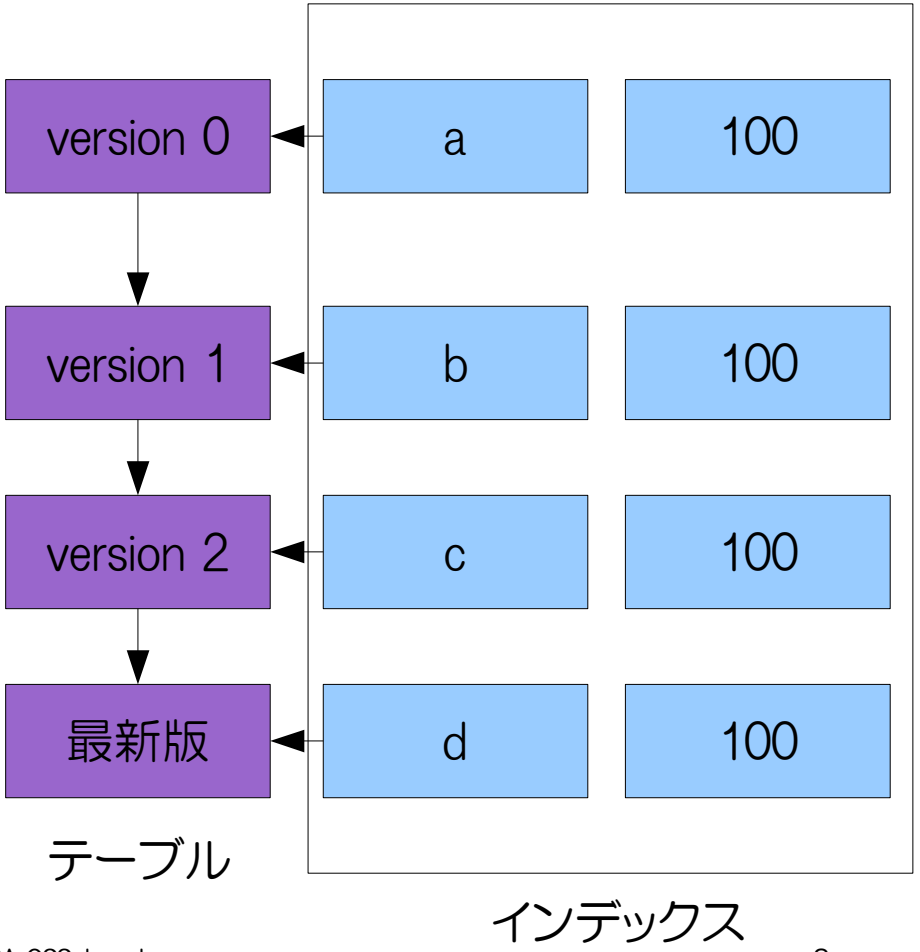
PostgreSQLの性能向上



PostgreSQL 8.3の改良点：性能

HOT(1)
従来の更新処理の問題点

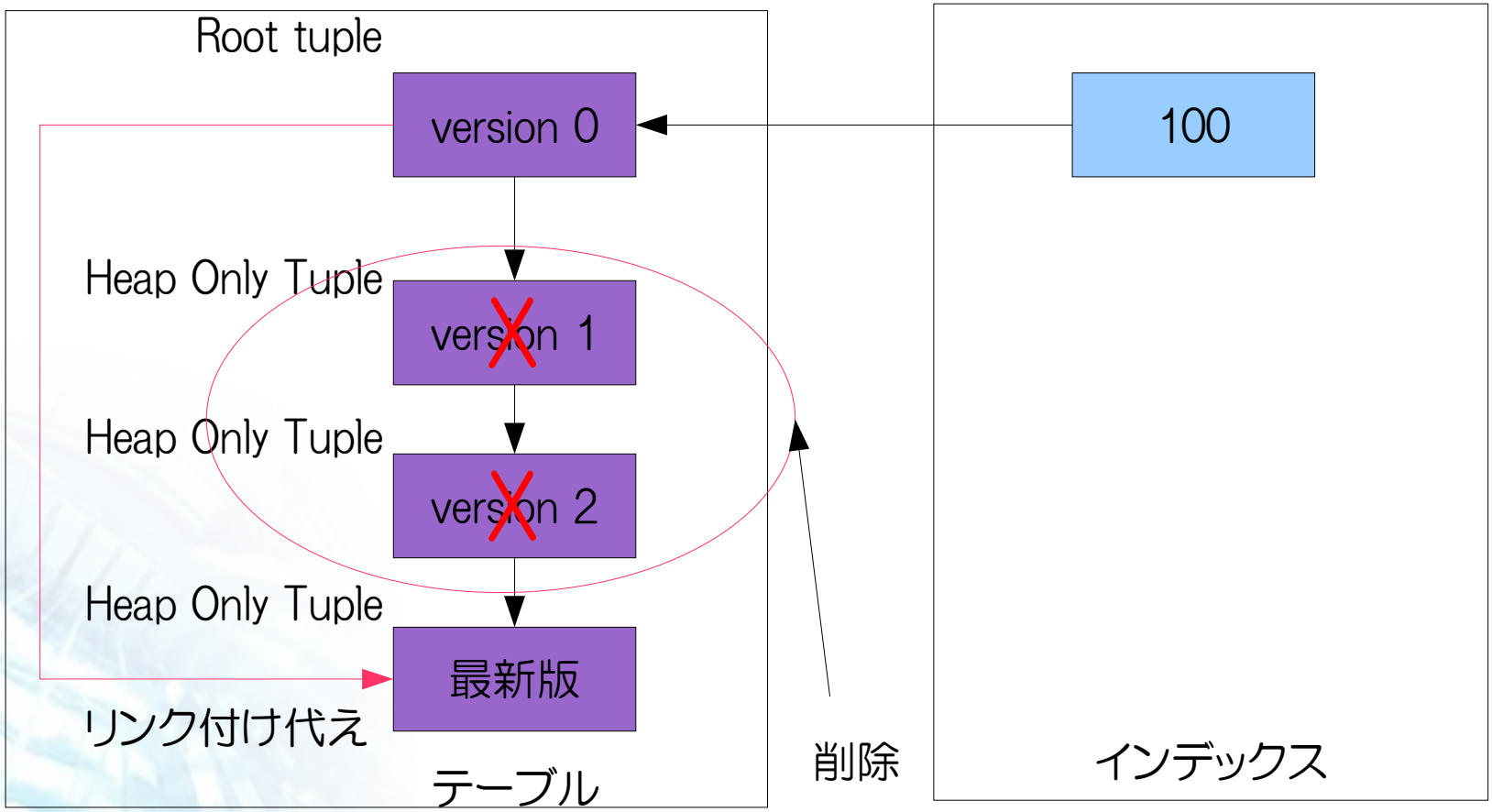
- 更新を繰り返すと更新連鎖(update chain)が長くなる
- 更新されないインデックスも追加される
- VACUUMをしないとどんどん遅くなる
  - 巨大なテーブルでは頻繁なVACUUMは困難



HOT(2): HOTとは

- HOT: Heap Only Tupleの略
- 更新対象列にインデックスカラムが含まれていない場合に有効
- VACUUMの必要性を減らす
  - 局所的なVACUUM処理をリアルタイムで実行
- UPDATEを繰り返してもテーブル，インデックスが肥大化しない

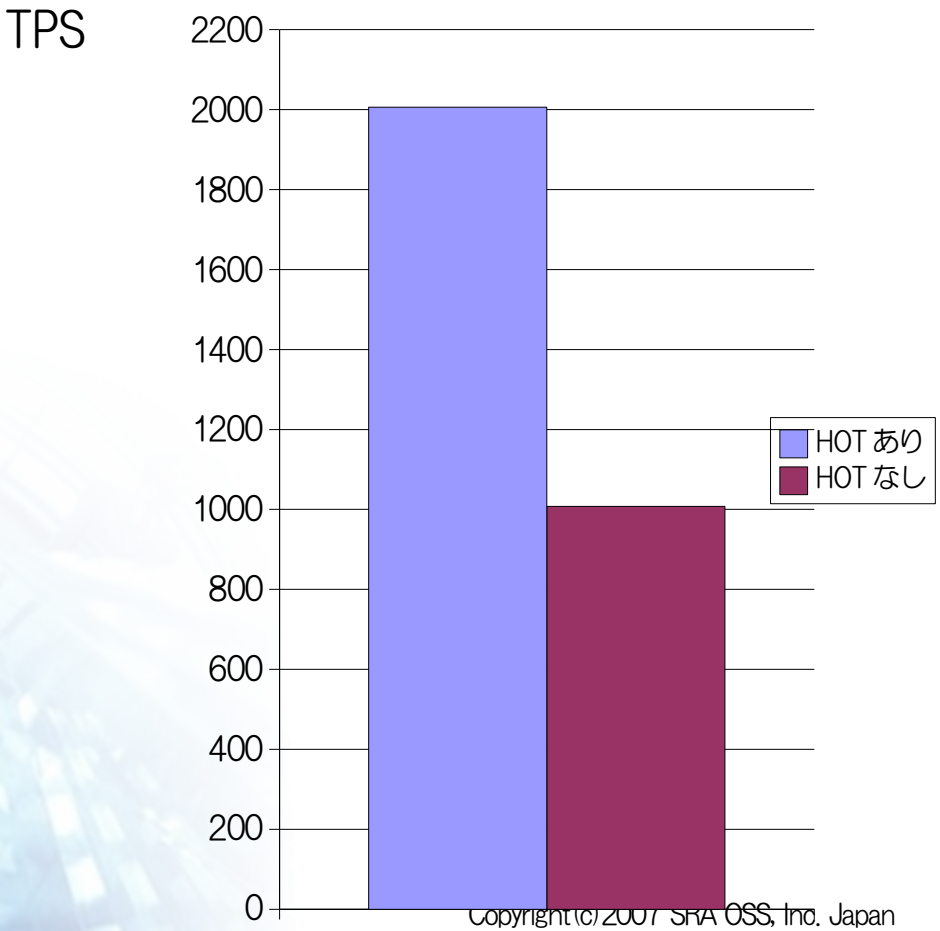
HOT(3): HOTの動作原理



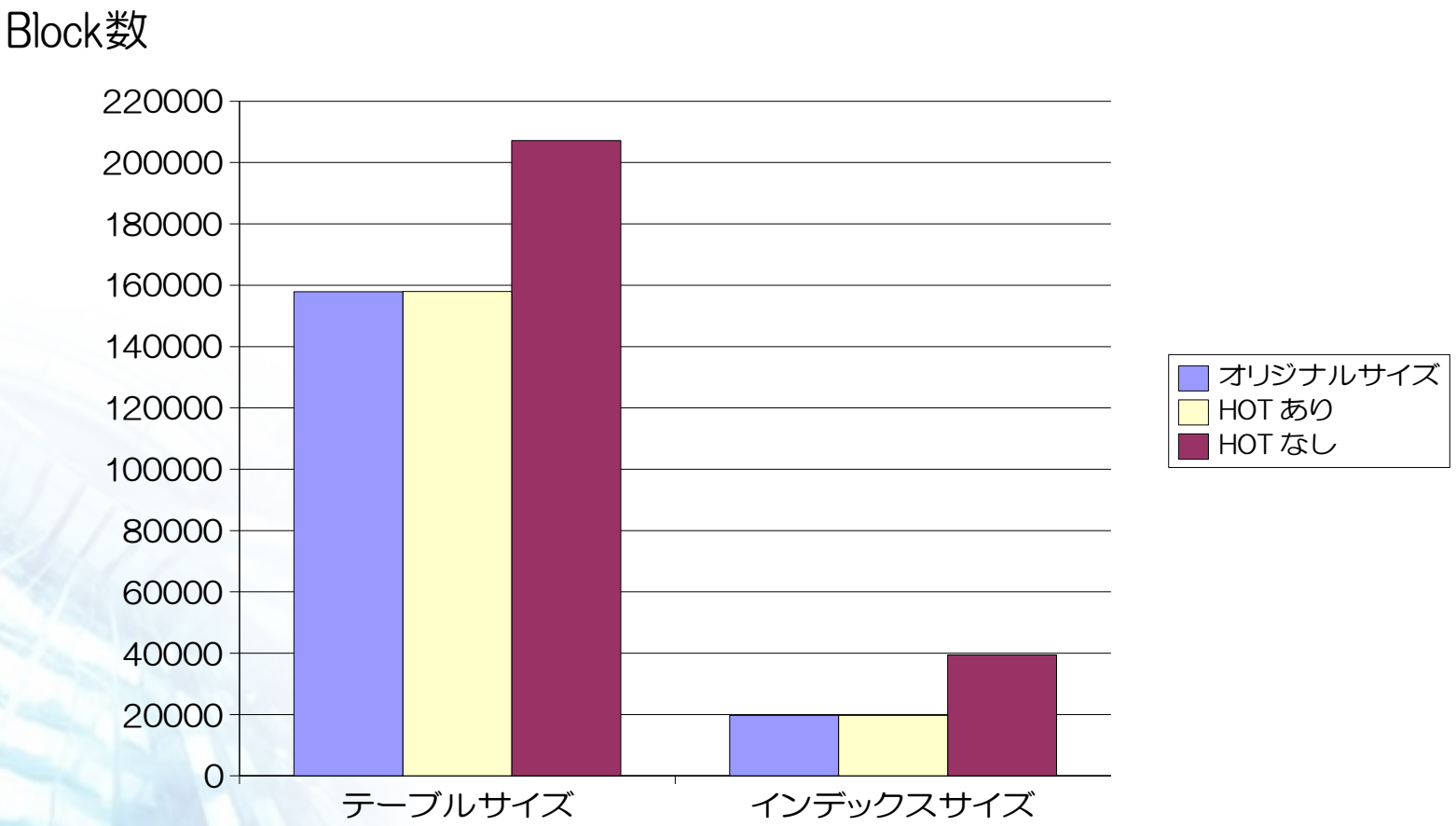
HOT(4): HOTの効果

- MLに流れた開発者自身によるベンチマーク
- pgbenchによる900万行のデータ
- メモリ2GB, 共有バッファ128MB
- 更新＋挿入
- autovacuumあり

HOT(5): トランザクション性能比較

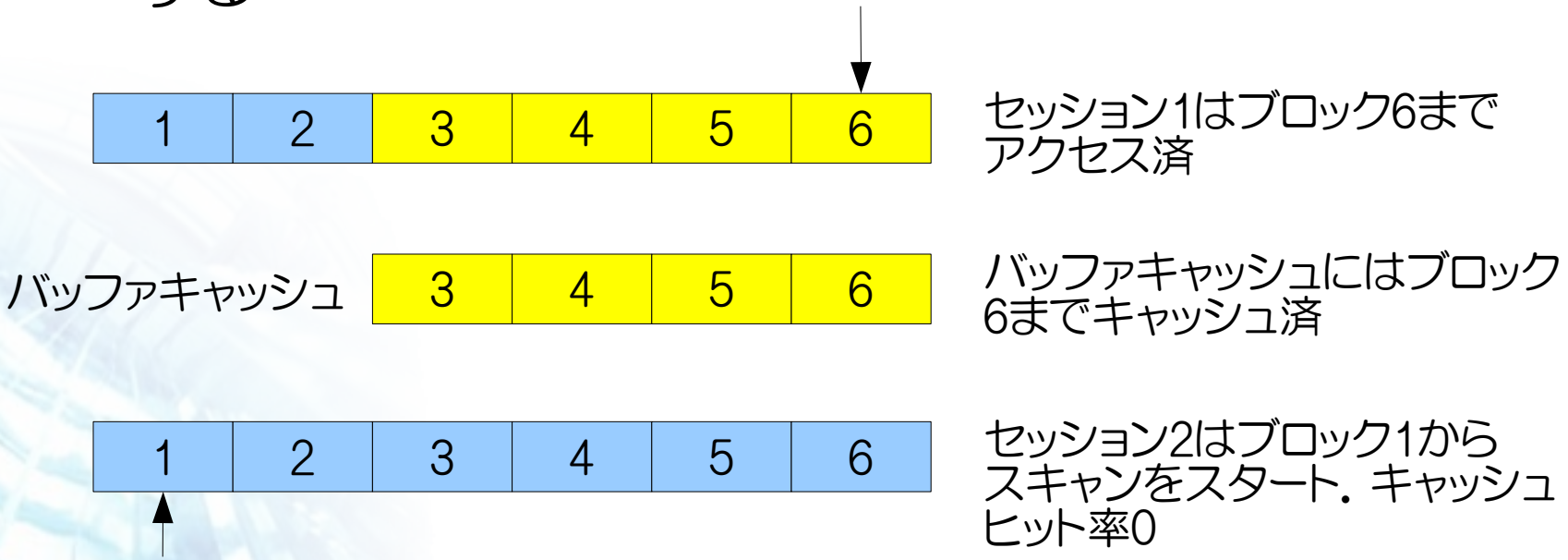


HOT(6): DBサイズの比較



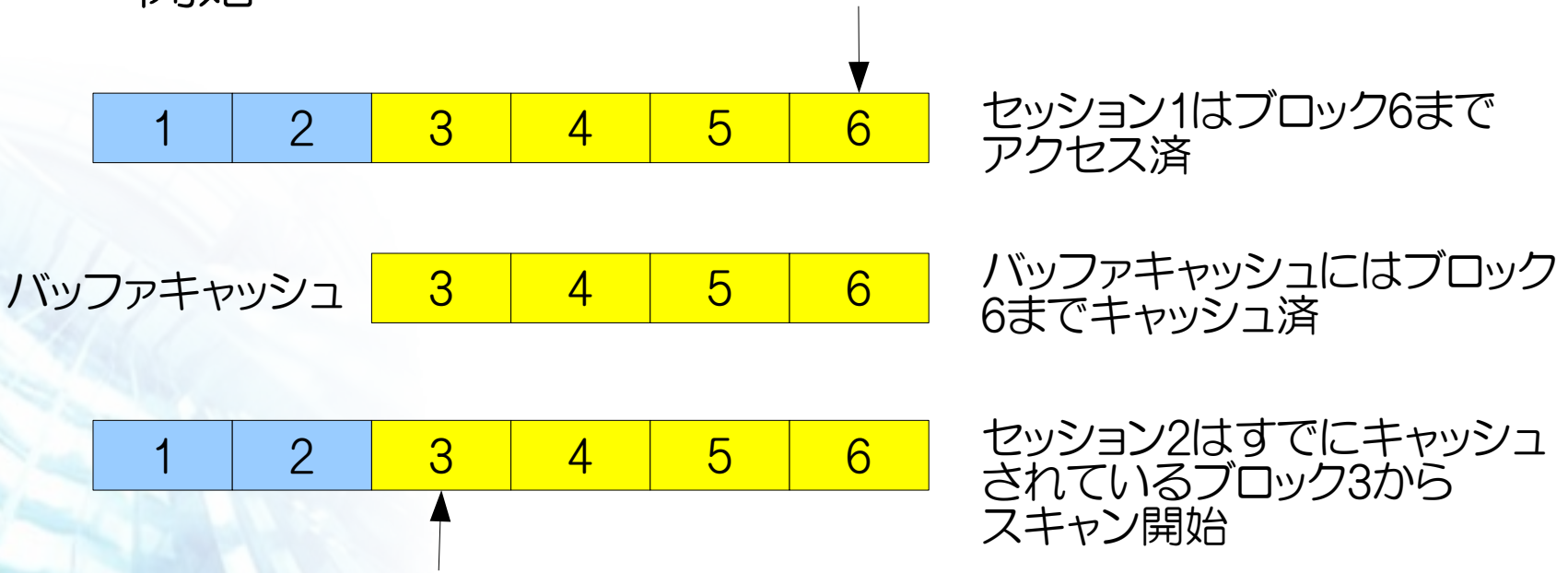
Synchronized Scan(1)

- 問題点
  - 共有バッファに入りきらないような大きなテーブルを複数セッションが順スキャンすると，キャッシュヒット率が低下する



Synchronized Scan(2)

- 解決策
  - 常にテーブルの先頭からスキャンするのではなく，バッファキャッシュにキャッシュされているブロックからスキャン開始



Synchronized Scan(3)

- Synchronized Scan利用上の注意点
  - 常にテーブルの先頭からスキャンするわけではないので，行の返却順序が一定にならない
  - 行の返却順序が問題になる場合は，ORDER BYを使って明示的にソートする
    - 実際にregression testの一部に変更が加えられている

Async Commit

- コミットした後同期書き込みが終わるまで待たずにクライアントにコミット完了を返す
- fsync=offと同等の高速化
- 再投入可能なデータやセンサーデータを扱う場合に効果的
- fsync=offとの違い
  - GUCの「synchronous_commit」でセッション中にオン・オフ可能
  - クラッシュしても最近のトランザクションが消えるだけで，データ不整合は起きない
  - fsync=offではどこまでデータが消えるか予測できない

ディスク領域の節約

- 可変長データの内部表現(varlena)の変更によるデータ領域の削減
  - 126バイト長以下の場合にサイズを表す領域を4バイトから1バイトに削減
- タプルヘッダーを4バイト縮小(27->23バイト，15%の削減)
- スキーマ定義や投入データにもよるが，場合によっては大きな効果

マージジョインの高速化

- 不必要なソートを避ける

explain select * from accounts t1, accounts t2 where t1.aid = t2.aid order by t1.aid desc;

8.2のプラン

```
Sort  (cost=6012239.09..6037239.09 rows=10000000 width=200)
  Sort Key: t1.aid
  ->  Merge Join  (cost=0.00..953070.25 rows=10000000 width=200)
        Merge Cond: (t1.aid = t2.aid)
        ->  Index Scan using accounts_pkey on accounts t1  (cost=0.00..401535.13 rows=10000000 width=100)
        ->  Index Scan using accounts_pkey on accounts t2  (cost=0.00..401535.13 rows=10000000 width=100)
```

8.3のプラン

```
Merge Join  (cost=0.00..942662.25 rows=10000000 width=194)
  Merge Cond: (t1.aid = t2.aid)
  ->  Index Scan Backward using accounts_pkey on accounts t1  (cost=0.00..396331.13 rows=10000000 width=97)
  ->  Index Scan Backward using accounts_pkey on accounts t2  (cost=0.00..396331.13 rows=10000000 width=97)
```

ORDER BY ... LIMITの高速化

- Webアプリケーションに良く使われるクエリ
- 入力行全体をソートするのではなく，LIMIT分だけソートする
- 場合によっては非常に効果的
  - SELECT * FROM accounts ORDER BY abalance LIMIT 10; (1000万件)
    - PostgreSQL 8.2: 113.7秒
    - PostgreSQL 8.3: 7.8秒
- “OFFSET”付には適用されないので注意

負荷分散チェックポイント

- 問題点
  - チェックポイント時に一時的に性能が低下する
    - dirty bufferの掃出し
  - background writerでは代用できない
    - トータルのI/Oが増えてしまうため
- 解決策
  - チェックポイントを負荷分散し，少しずつdirty bufferを掃出す
  - checkpoint_completion_target
- 効果
  - 性能が一定に

WALログの省略

- 不必要なWALログの出力がなくなったことにより，ログ領域の削減，処理の高速化
  - pgbenchでは，COPYの前にTRUNCATEを実行することにより，COPYでログ出力されない
    - 1000万件COPYの場合
      - 8.2では6分32秒
      - 8.3では3分18秒

そのほかの性能改良

- リカバリの高速化
  - 上書きされるページのデータを読まない

PostgreSQL 8.3の改良点：DB運用管理機能

インデックスアドバイザー

- 「インデックスアドバイザー」とは？
  - 必要なインデックスがあれば指摘してくれるような機能
  - 多くの商用DBMSに見られるツール
- PostgreSQLでの実装
  - 「インデックスアドバイザー」の機能自体はユーザが実装
    - 実際にはベンダーやpgfoundryで開発されることを期待？
    - 実装的にはPostgreSQLからダイナミックロードされるモジュールになる
  - PostgreSQLにはそのためのインターフェイス(フック)を設ける
    - EXPLAINコマンドから呼び出されるフックなど

インデックスアドバイザーの実装例

```
regression=# load '/home/tgl/pgsql/advisor';
LOAD
regression=# explain select * from fooey order by unique2,unique1;
                                QUERY PLAN
-----------------------------------------------------------------------
 Sort  (cost=809.39..834.39 rows=10000 width=8)
   Sort Key: unique2, unique1
   ->  Seq Scan on fooey  (cost=0.00..145.00 rows=10000 width=8)

 Plan with hypothetical indexes:
 Index Scan using <hypothetical index> on fooey  (cost=0.00..376.00 rows=10000 width=8)
(6 rows)
```

ユーザ定義インデックス
アドバイザーのロード

ユーザ定義インデックスアドバイザーの出力

ソート処理のモニタリング

```
test=# set trace_sort to on;
test=# explain analyze select * from accounts order by abalance;
LOG:  begin tuple sort: nkeys = 1, workMem = 1024, randomAccess = f
LOG:  switching to external sort with 7 tapes: CPU 0.02s/0.00u sec elapsed 0.03 sec
LOG:  performsort starting: CPU 0.51s/0.00u sec elapsed 0.52 sec
LOG:  finished writing final run 1 to tape 0: CPU 0.52s/0.00u sec elapsed 0.52 sec
LOG:  performsort done: CPU 0.52s/0.00u sec elapsed 0.52 sec
LOG:  external sort ended, 1331 disk blocks used: CPU 0.96s/0.00u sec elapsed 0.96 sec

                                                    QUERY PLAN

-----------------------------------------------------------------------------------------------------------
 Sort  (cost=16191.82..16441.82 rows=100000 width=97) (actual time=527.619..771.426
rows=100000 loops=1)
   Sort Key: abalance
   Sort Method:  external sort  Disk: 10648kB
   ->  Seq Scan on accounts  (cost=0.00..2588.00 rows=100000 width=97) (actual
time=0.021..213.239 rows=100000 loops=1)
 Total runtime: 969.161 ms
(5 rows)
```

Auto vacuum

- Auto vacuumとは
  - VACUUMを自動実行
- 8.2までは
  - デフォルトではauto vacuum無効
  - 統計情報の収集が必要だったため
- 8.3からは
  - デフォルトでauto vacuum有効
  - 追加された設定項目
    - log_autovacuum_min_duration
    - track_counts

ログ項目の追加

- log_lock_waits
  - ロック待ち状態のトランザクションを表示

  LOG: process 12274 still waiting for AccessExclusiveLock on relation 16467 of database 16384 after 1003.966 ms
  STATEMENT: lock table t1;
- checkpointログ
  - log_checkpoints

  2007-09-24 21:35:45 JST LOG: checkpoint starting: time
  2007-09-24 21:38:15 JST LOG: checkpoint complete: wrote 1693 buffers (55.1%); 0 transaction log file(s) added, 0 removed, 0 recycled; write=150.021 s, sync=0.014 s, total=150.062 s

ヒープページ調査ツール(contrib)

```
test=# SELECT * FROM heap_page_items(get_raw_page('pg_class',0));
-[ RECORD
1 ]-------------------------------------------------------------------------
lp           | 1
lp_off       | 5744
lp_flags     | 1
lp_len       | 205
t_xmin       | 2
t_xmax       | 0
t_field3     | 497
t_ctid       | (0,1)
t_infomask2  | 27
t_infomask   | 10507
t_hoff       | 32
t_bits       |
11111111111111111111111100000000000000011010001001101000000000000000
t_oid        | 11403
```

PostgreSQL 8.3の改良点：SQL機能

更新可能カーソル(1)

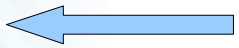
- 従来のカーソルは「更新不可能」だった

```
BEGIN
DECLARE c CURSOR FOR SELECT * FROM T1;
FETCH 2 FROM c;
 i |  j
---+-----
 1 | foo
 2 | bar
(2 rows)

DELETE FROM t1 WHERE i = 1;
MOVE BACKWARD ALL IN c;
FETCH ALL FROM c;
 i |  j
---+-----
 1 | foo
 2 | bar
(2 rows)
```

削除されていないように見える

```
SELECT * FROM t1;
 i |  j
---+-----
 1 | foo
(1 row)
```



実際には削除されている

更新可能カーソル(2)

- 「FOR UPDATE」を付けて更新可能カーソルへ

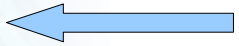
```
BEGIN
DECLARE c CURSOR FOR SELECT * FROM T1 FOR UPDATE;
FETCH 2 FROM c;
 i |  j
---+-----
 1 | foo
 2 | bar
(2 rows)

DELETE FROM t1 WHERE CURRENT OF c;
MOVE BACKWARD ALL IN c;
FETCH ALL FROM c;
 i |  j
---+-----
 1 | foo
(1 rows)

SELECT * FROM t1;
 i |  j
---+-----
 1 | foo
(1 row)
```

新しい構文

削除されている



実際に削除されている

JIS 2004とは？

- 従来の日本語文字コード規格(JIS X 0208+JIS X 0212)を拡張した文字コード規格(JIS X 0213)の通称
  - ただし，JIS 2004はJIS X 0208+JIS X 0212の完全上位互換ではない
- 多数の漢字や符合を追加
  - 一部の「機種依存文字」も取り込み
- Windows Vistaなどが採用し，注目される
- 人名などが充実しているため，官公庁や地方公共団体での採用が期待される

JIS 2004で使えるようになった機種依存文字の一部(NEC特殊文字)

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳

Ⅰ Ⅱ Ⅲ Ⅳ Ⅴ Ⅵ Ⅶ Ⅷ Ⅸ Ⅹ

ミリ キロ センチ メートル グラム トン アール ヘクタール リットル ワット カロリー ドル セント パーセント ミリバール ページ mm cm km mg kg cc ㎡ 平成 〃 〃

No. K.K. TEL ㊤ ㊥ ㊦ ㊧ ㊨ ㈱ ㈲ ㈹ 明治 大正 昭和 ≒ ≡ ∫ ∮ √ ⊥ ∠ ∟ ⊿ ∵ ∩ ∪

JIS 2004対応の実装

- JIS 2004対応のEUC(EUC_JIS_2004), シフトJIS(SHIFT_JIS_2004)をエンコーディングとして追加
  - JIS 2004はEUC_JP, SJISの上位互換ではないため
- UNICODEはそのまま利用可能
- UNICODE <--> EUC_JIS_2004, SHIFT_JIS_2004の変換が可能

JIS 2004利用上の注意

- クライアントはUNICODE中心
- BMP(2バイトのUCS)の範囲以外のUNICODEが利用できることが必要(Unicode 3.1以降のサポート)
- 字形が変っている文字がある

そのほかのSQLの改良(1)

- CREATE FUNCTIONの改良
  - 実行コスト，想定返却行数の指定が可能に
- DISCARD ALL
  - セッションデータの全初期化をセッションを維持したまま実行可能
  - pgpoolが待っていた機能！
  - 初期化対象データ
    - prepared plan
    - 一時テーブル
    - SETで一時的に変更したデータ
    - カーソル
    - LISTEN/NOTIFY

そのほかのSQLの改良(2)

- tsearch2の本体への取り込み
- CSV形式のログ
- CREATE TABLE LIKE
  - SQL標準, パーティショニング
- warm standby用のcontribコマンド
  - pg_standby
- XML対応
  - 主にSQL:2003対応
  - XMLデータ型
  - テーブル定義やデータのXMLとの相互変換
  - XPath対応

そのほかのSQLの改良点(3)

- 複合型の配列サポート
- ENUMデータ型
  - MySQLからのマイグレーションが容易に

そのほかの改良点

- アーカイブログの圧縮インターフェイス
  - 実際の圧縮はexternal projectにて
- 並列CREATE INDEX

PostgreSQL 8.3のリリース時期

- 10月頃にベータリリースを予定
- 年内には正式リリース？

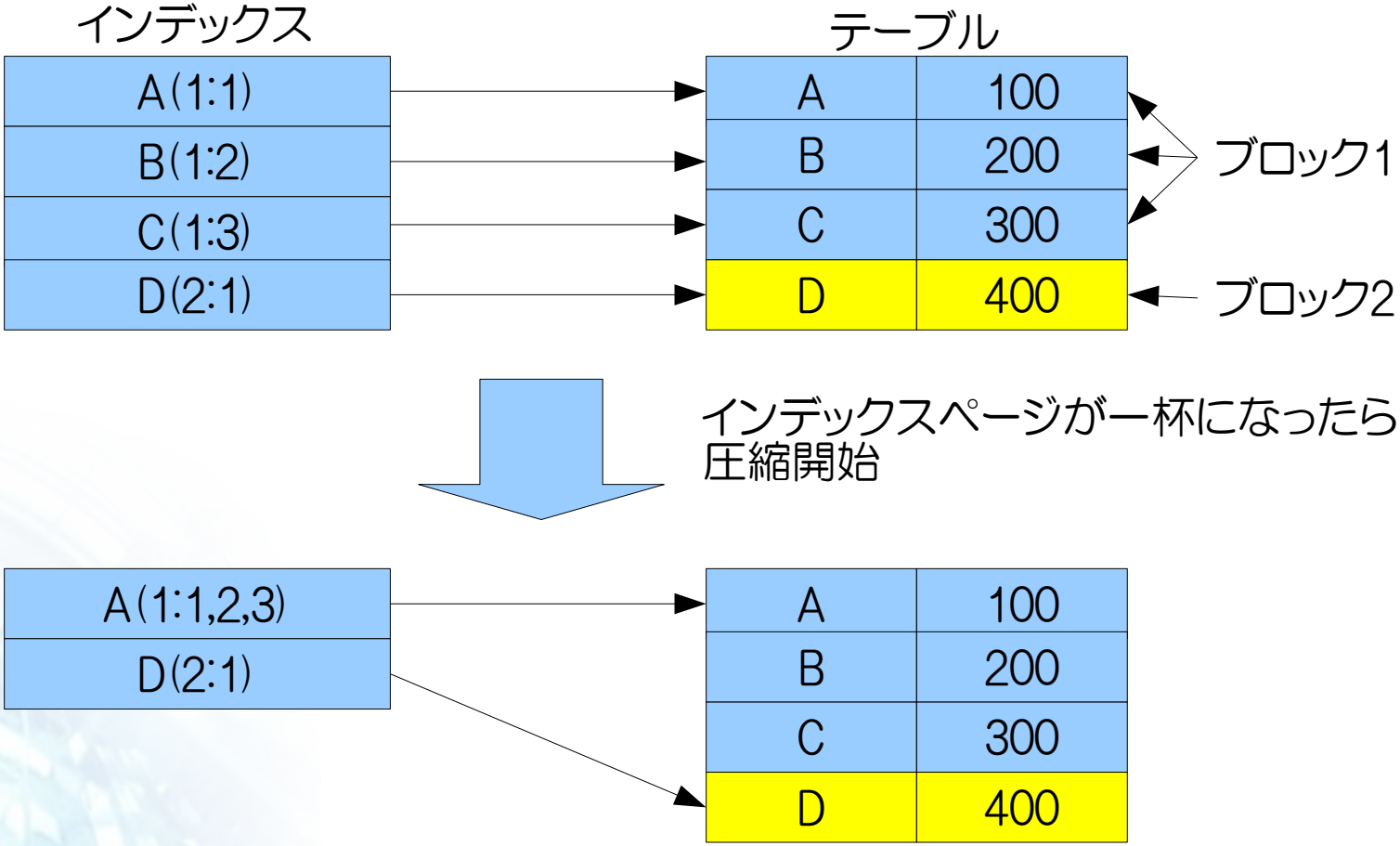
8.4以降で取り込むもの

- Grouped Index Tuples(GIT)
- VACUUM性能の向上
  - DSM(Dead Space Map)
- 検索性能の向上
  - ビットマップインデックス

Grouped Index Tuples(GIT)

- B-Treeインデックスを効果的に圧縮する技術
- CLUSTERコマンドと併用すると効果的
- 時には100倍の圧縮効果も
- CREATE INDEXに新しいオプション
  - CREATE INDEX ... (groupthreshold=2)

GITの動作



まとめ

- 8.3の改良点では，何と言ってもHOTが注目株
  - PostgreSQLが苦手だった，更新の多い大規模データベースへの対応がかなり良くなった
- ストレージ回りにも手を入れた意欲的なバージョン
  - 8.0以来の大きな内部構造変更
- GIT，DSM，ビットマップインデックスなどが8.4以降に先送りになったのは残念
- PostgreSQLの今後
  - 大規模DBへの対応はかなり完成形に近づいた
  - 8.4のリリースは意外と早い？

参考URL

- http://developer.postgresql.org/index.php/Todo:WishlistFor83
- http://developer.postgresql.org/index.php/Feature_Matrix
- http://developer.postgresql.org/index.php/Todo:PatchStatus