SRA OSS, INC.

PostgreSQL 本体の高速化の軌跡

～ これまでとこれから ～

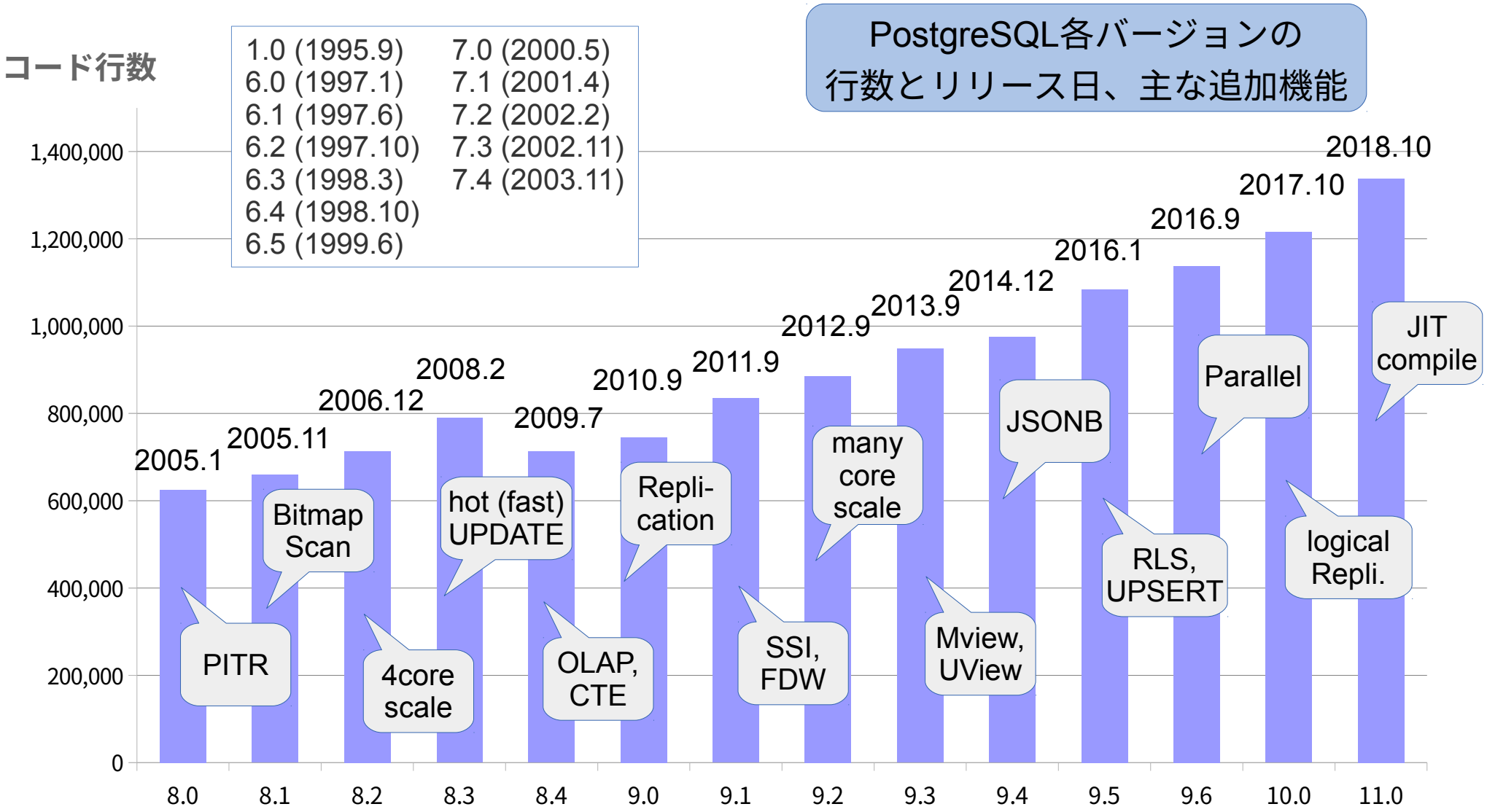
長田悠吾
SRA OSS, Inc. 日本支社　技術開発室

2019/4/18

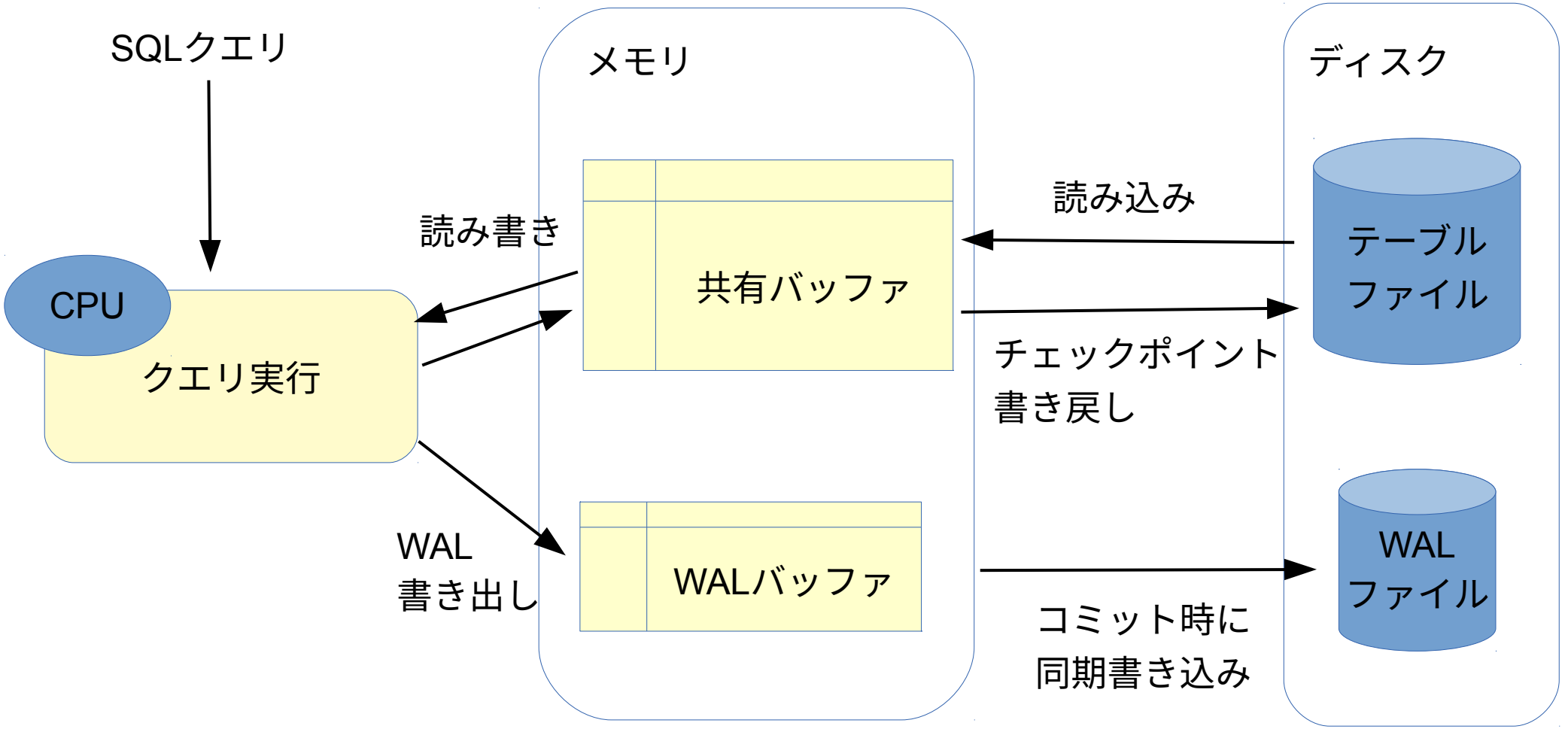
PostgreSQLとは

- 代表的なオープンソースのRDBMSの１つ
  - カリフォルニア大学で研究・開発されたIngres(‘70), Postgres(‘80)がルーツ
- PostgreSQLライセンスで配布
  - BSDタイプの緩いライセンス
- 特定のオーナー企業を持たず、コミュニティによる開発が続けられている
  - 年１回のメジャーバージョンアップ

PostgreSQLの軌跡



基本的なアーキテクチャ



高速化のアプローチ

- ディスク I/O
  - テーブル保存場所の分散
  - バッファ管理の改善
  - テーブルサイズの肥大化防止
- クエリ処理
  - マルチCPU同時実行性能の向上
  - SQL 機能の強化
  - プランナやエクゼキュータの改善
- クラスタ構成
  - レプリケーションの利用
- 大規模データ
  - パーティショニング
  - パラレルクエリ
  - Just-in-Time コンパイル

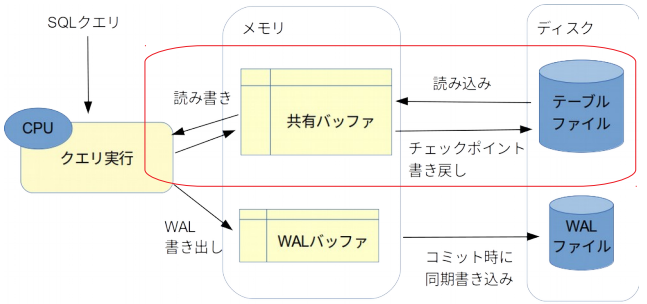
ディスク I/O の改善

WAL

- WAL (Write Ahead Log)
  - PostgreSQL のトランザクションログ
    - 追加、更新、削除などの操作を記録
  - 実装されたのは PostgreSQL 7.1
- それ以前は、トランザクションをコミットする度に、変更されたデータを全てディスクに同期書き込みしていた
  - 書き込み性能は非常に低かった
  - 1000万件のデータロードに12時間以上

→ WALの導入により書き込み性能が大幅に改善

バッファ管理



- ページ置き換えアルゴリズム
  - キャッシュヒット率の改善（LRU → ARC / 2Q) (8.0)
  - 共有バッファへの同時アクセス改良（clock アルゴリズム）(8.1)
- ディスクへの同期
  - バックグラウンドライタによる定期的な書き込み（8.0）
  - 負荷分散チェックポイント（8.3）
- アクセス戦略
  - 大規模なシーケンススキャン (8.3)
    - 使用頻度の高いページがバッファから押し出されるのを防止
  - 大量挿入の性能改善（8.4）

不要領域の回収（VACUUM）

- 多版型同時実行制御（Multi Version Concurrent Control: MVCC) (6.5)
  - 更新時：古い行を残したまま新しい行を追記（追記型）
  - 読み込みと書き込みが互いをブロックしない

- 古い行が残ったままだと、テーブルサイズの肥大化が発生し性能が劣化

  → VACUUM で不要領域を回収する必要がある

- VACUUM の改善
  - コンカレント VACUUM (7.2)
  - 自動VACUUM (7.4)
  - VACUUM の遅延実行 (8.0)
  - VACUUM 高速化 (8.2, 8.4, 9.6, 10, 11)

ページの効率的な利用

- フィルファクタ (8.2)
  - テーブルページの空き領域に余裕をもたせる機能
  - 同じページの中で更新操作が出来るため効率が良くなる

- HOT (Heap Only Tuple) (8.3)
  - 不要なインデックスの更新を抑制

    → 更新性能が向上
  - テーブルページ内の不要領域が再利用される機会を増加

    → VACUUM の必要性が減り、テーブルサイズの肥大化が抑制

ディスク I/O 関連その他

- テーブル空間 (8.0)
  - テーブルやインデックスなどを格納する記憶領域
  - 用途に合わせて複数のディスクシステムを使い分けることで性能向上
- UNLOGGED テーブル（9.0）
  - トランザクションログ（WAL）に記録しないテーブル
  - 書き込みが高速
- pg_prewarm (9.4)
  - 指定したテーブルを事前にメモリバッファに載せることでアクセスを高速化
  - サーバ起動時に前回の共有バッファの内容を復元できるように（11）

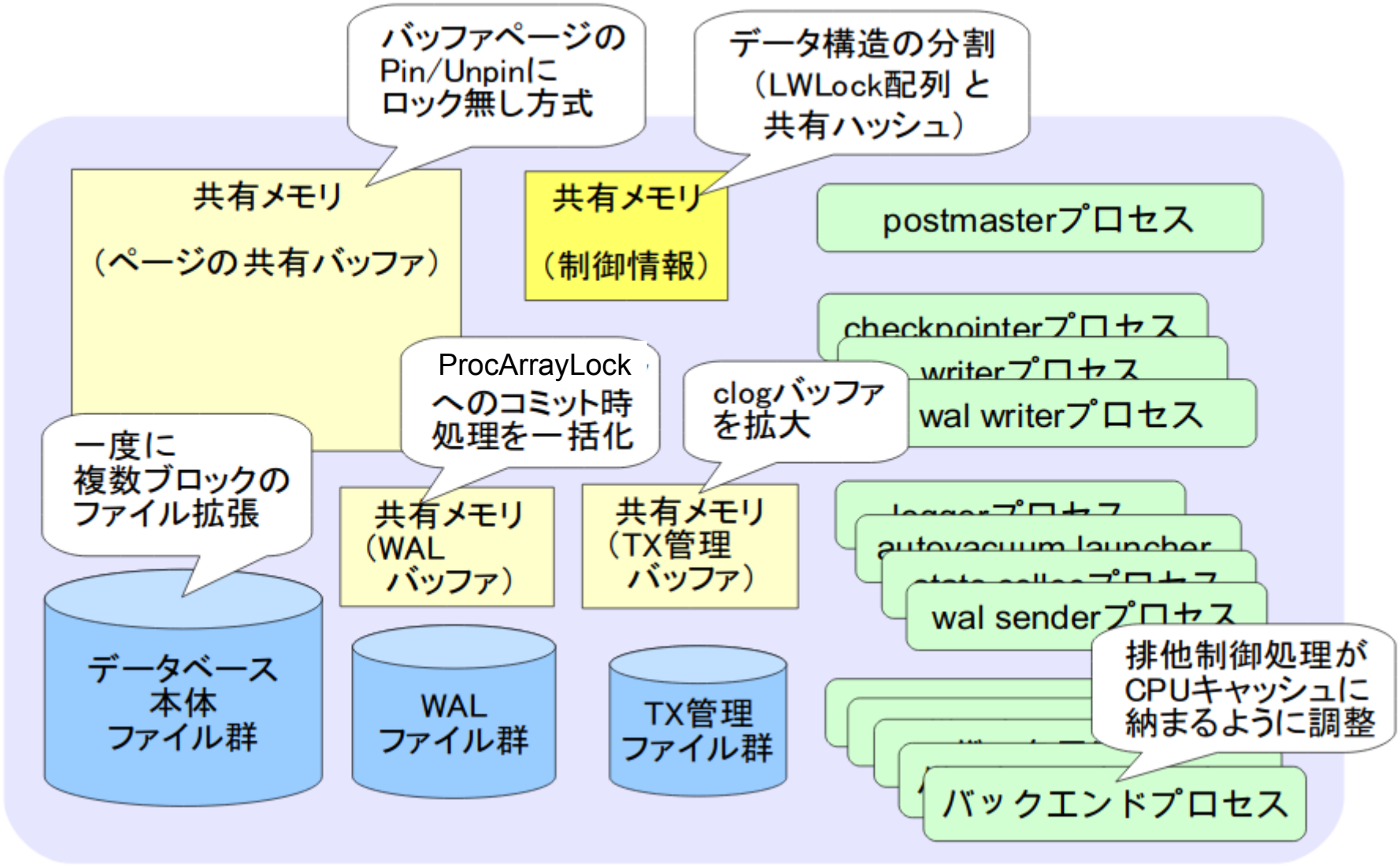
クエリ実行の高速化

多CPUでの同時実行性能の改善

- 全体的なロックの見直しによる競合の削減 (8.2)
- 同時実行性能の大幅に改善（32〜64 コア）（9.2）
  - グループコミットの効率化

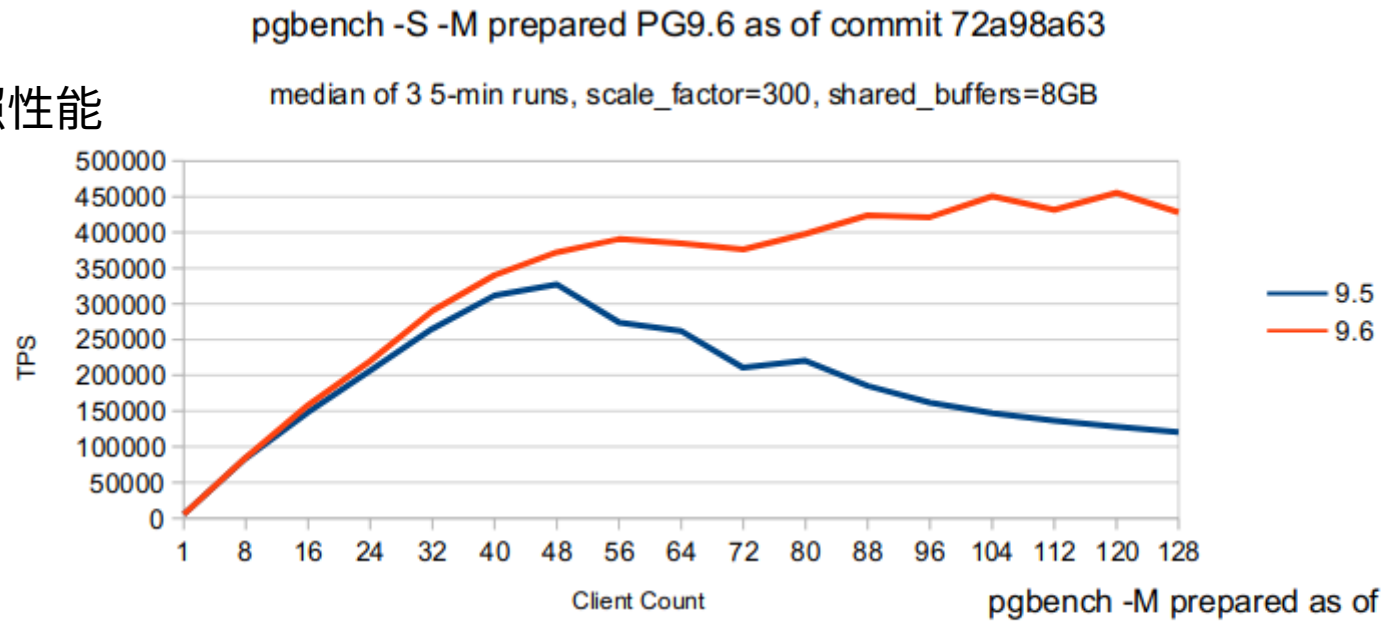
    ロック負荷軽減
  - 頻繁にアクセスされる構造を分割してキャッシュに乗りやすく
- WALバッファ同時書き込みによる性能向上（9.4）
- 内部的なロック実装の改訂によるスピンロック軽減 (9.5)
- 管理用ハッシュテーブルのパーティション数の増加 (9.5)

CPUスケーラビリティ性能改善点（9.6）

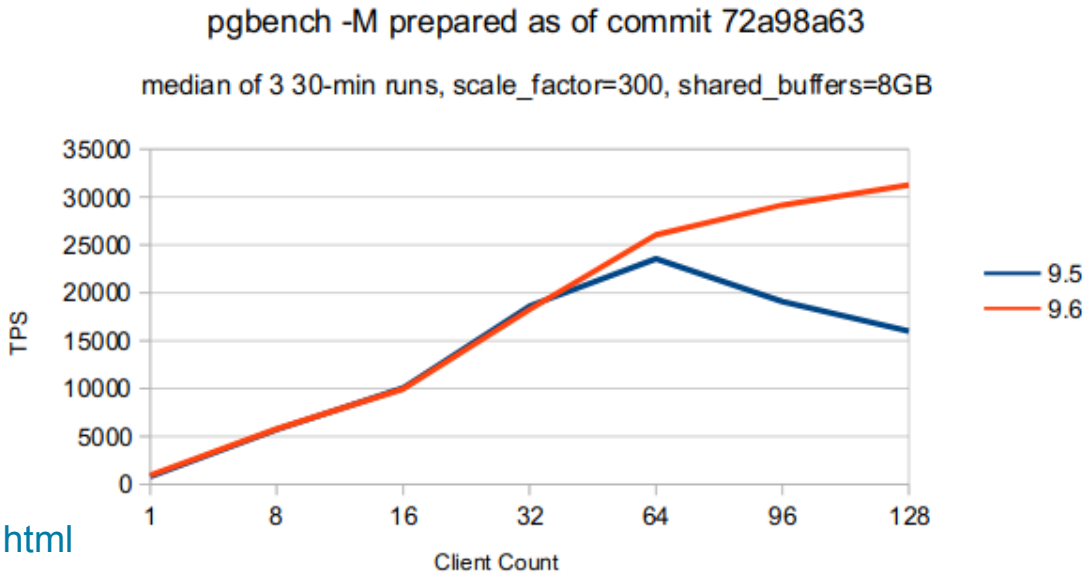


9.5 - 9.6 性能改善

参照性能



読み書き性能



Intel x86, 8 sockets, 64 cores, 128 hardware threads, 500GB RAM

PGCon2016
Faster PostgreSQL: Improved Writes, Amit Kapila
https://www.pgcon.org/2016/schedule/events/915.en.html

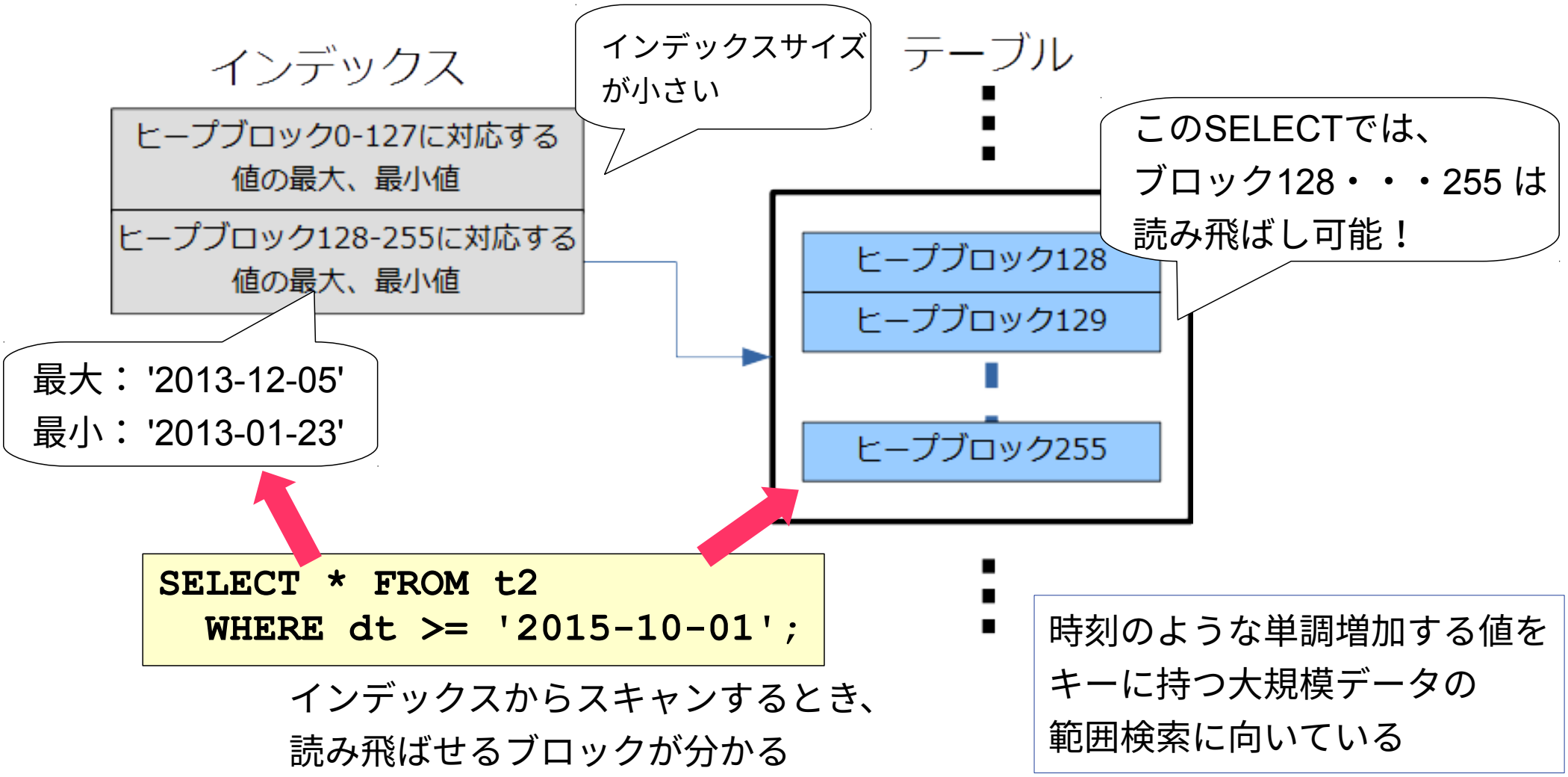
インデックス機能の強化

- ビットマップインデックス (8.1)
  - and / or で並べられた複数条件を、複数のインデックスの同時使用で検索
- GINインデックス（8.2）
  - 全文検索などで利用

- インデックスオンリースキャン（9.2）
  - テーブルデータにアクセスせずに、インデックスだけを読む検索
  - インデックスが付与されたキーの値の取得が高速
- カバリングインデックス (11)
  - インデックスが付与されていない列の値もインデックスに含めることができる

BRIN インデックス (9.6)

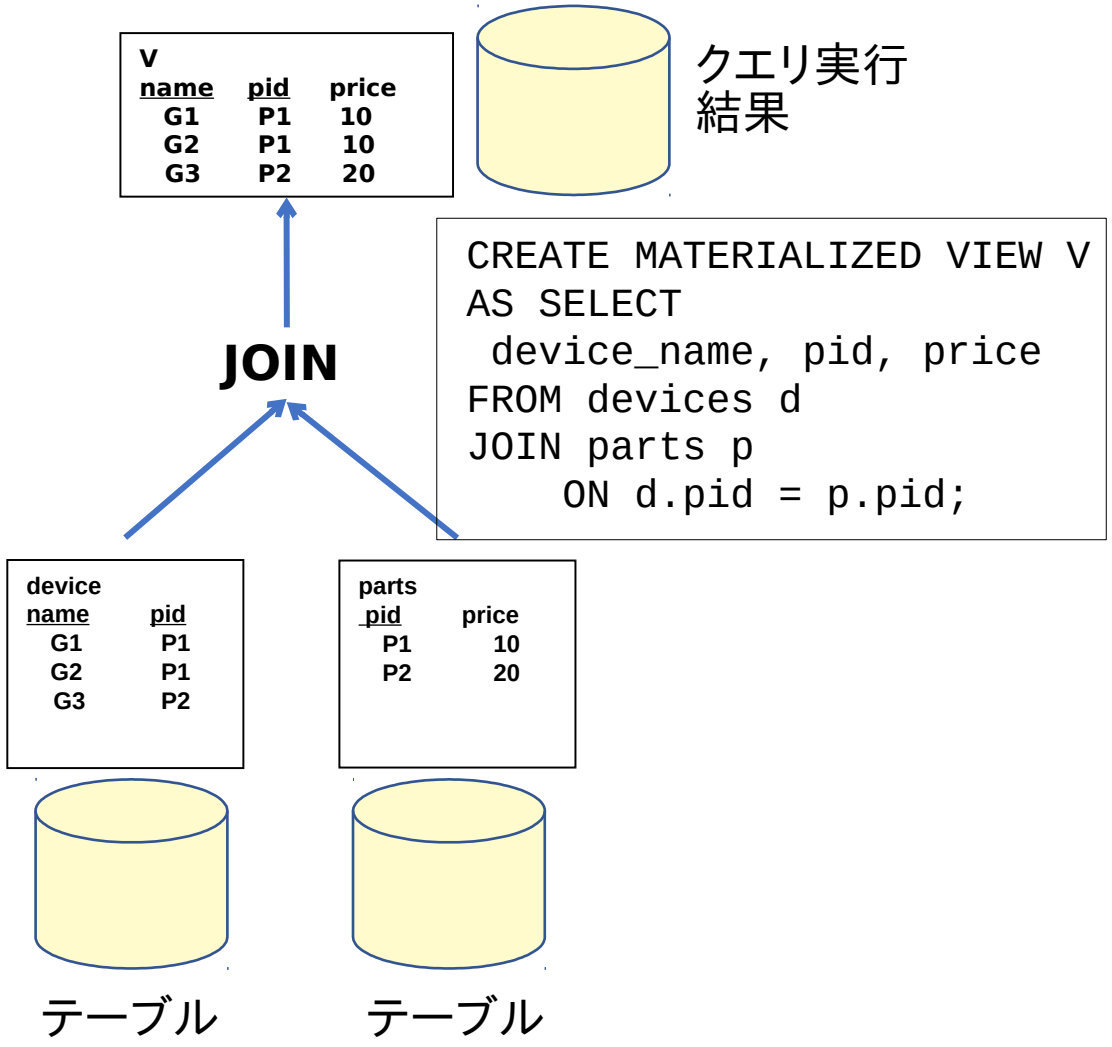
- いくつかのブロック（＝ページ）の塊ごとに最大値・最小値を保持する



マテリアライズドビュー (9.3)

- クエリの結果がデータベースに格納されている
- 通常のビューよりも高速な応答が可能
- テーブルが更新された時には、クエリとデータが不一致となる
- 手動の「リフレッシュ」でクエリの再実行が必要
  - 自動更新、差分更新の機能はない

```
REFRESH MATERIALIZED VIEW V;
```



プランナ・エクゼキュータの改善

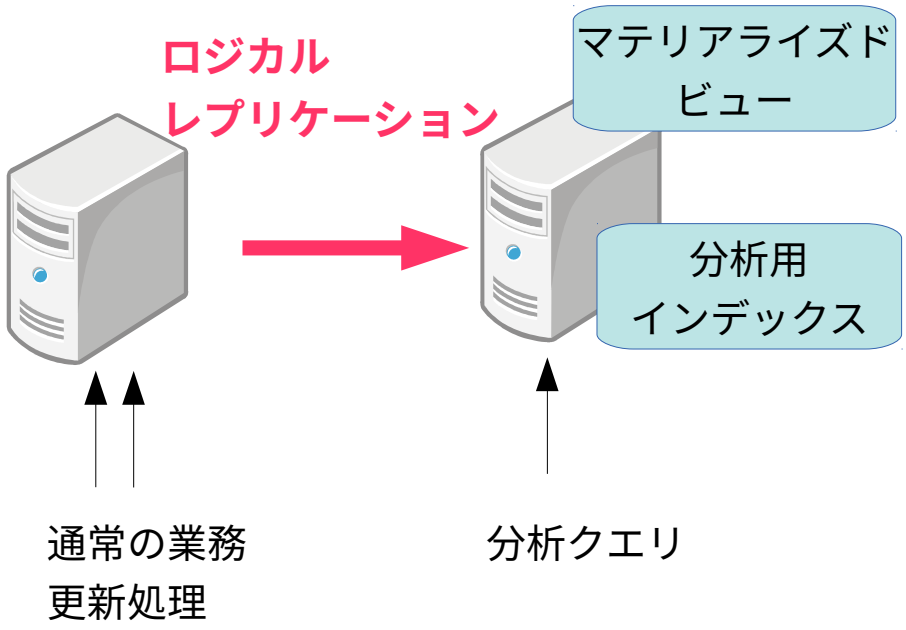
- リリースの毎に改善が重ねられている（以下はその抜粋）

- IN/NOT IN 副問い合わせの大幅な高速化 (7.4)
- ハッシュバケットを使用したGROUP BY処理の改良 (7.4)
- JOIN構文による結合順序の最適化 (7.4)
- インデックスを使用した MIN() とMAX() の最適化 (8.1)
- ソート処理の性能向上 (8.2)
- ORDER BY ... LIMIT をソートせずに処理 (8.3)
- DISTINCT, UNION / INTERSECT / EXCEPT でハッシュが利用可能に（8.4）
- 不要な OUTER JOIN の除去（9.0）
- FULL OUTER JONI でハッシュジョインが使用可能に（9.1）
- 内側にインデックススキャンを使う入れ子ループの改善（9.2）
- インデックススキャンのコスト推定改善 (9.3)
- 部分インデックスを使った場合のコスト推定改善（9.5)
- 複数列に関する統計情報の計算 (10)
- 集計計算の速度改善 (11)

レプリケーションの利用

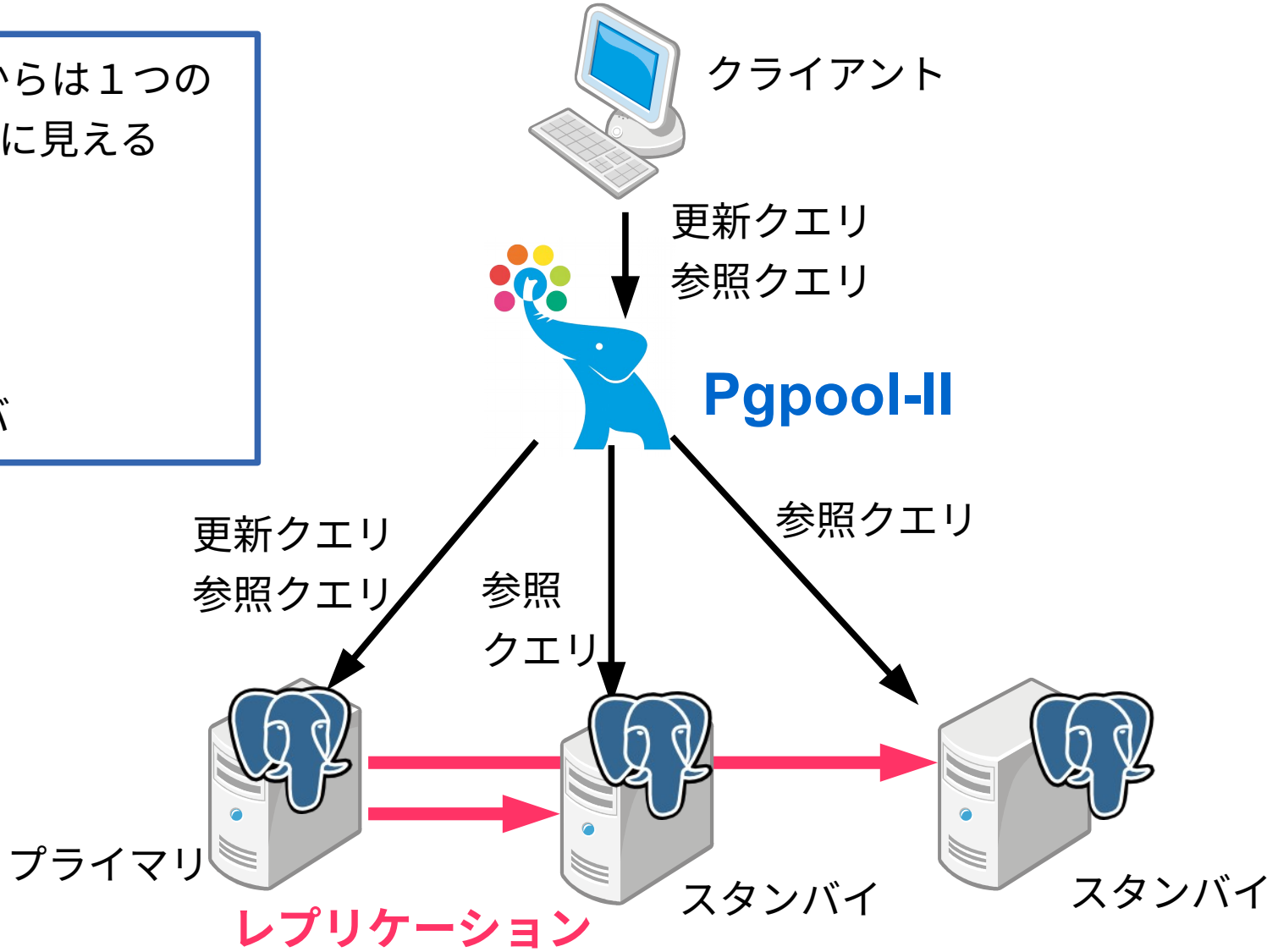
PostgreSQL のレプリケーション

- ストリーミングレプリケーション (9.0)
  - WAL を利用した組み込みのレプリケーション機能
    - 同期レプリケーション(9.1)、カスケードレプリケーション (9.2), 複数サーバ対応 (9.6)、Quorum コミット (10)
- ロジカルレプリケーション (10)
  - テーブル単位のレプリケーション
  - 複製先でのデータ加工が可能
- レプリケーションを利用した性能改善
  - 業務・更新用サーバと分析用サーバの分離
  - 複数サーバ間での参照負荷分散



参照負荷分散

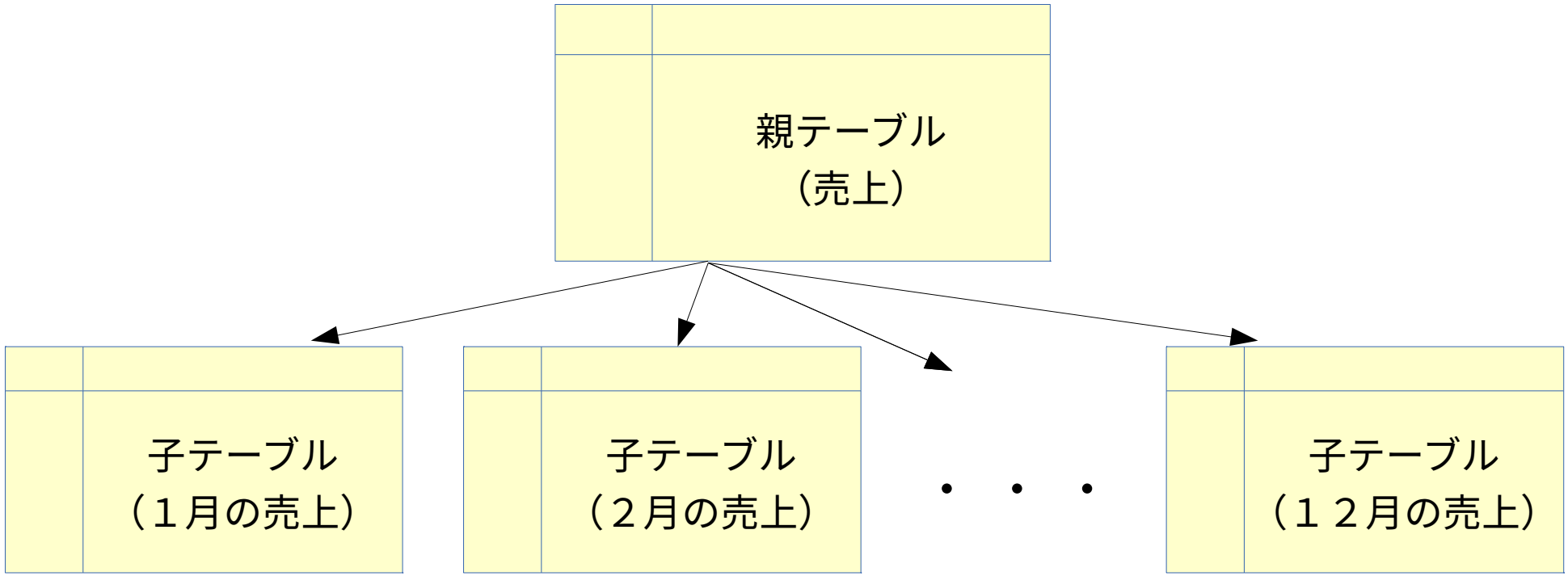
- アプリケーションからは１つの PostgreSQL のように見える
- クエリ振り分け
- 負荷分散
- 自動フェイルオーバ



パーティショニング

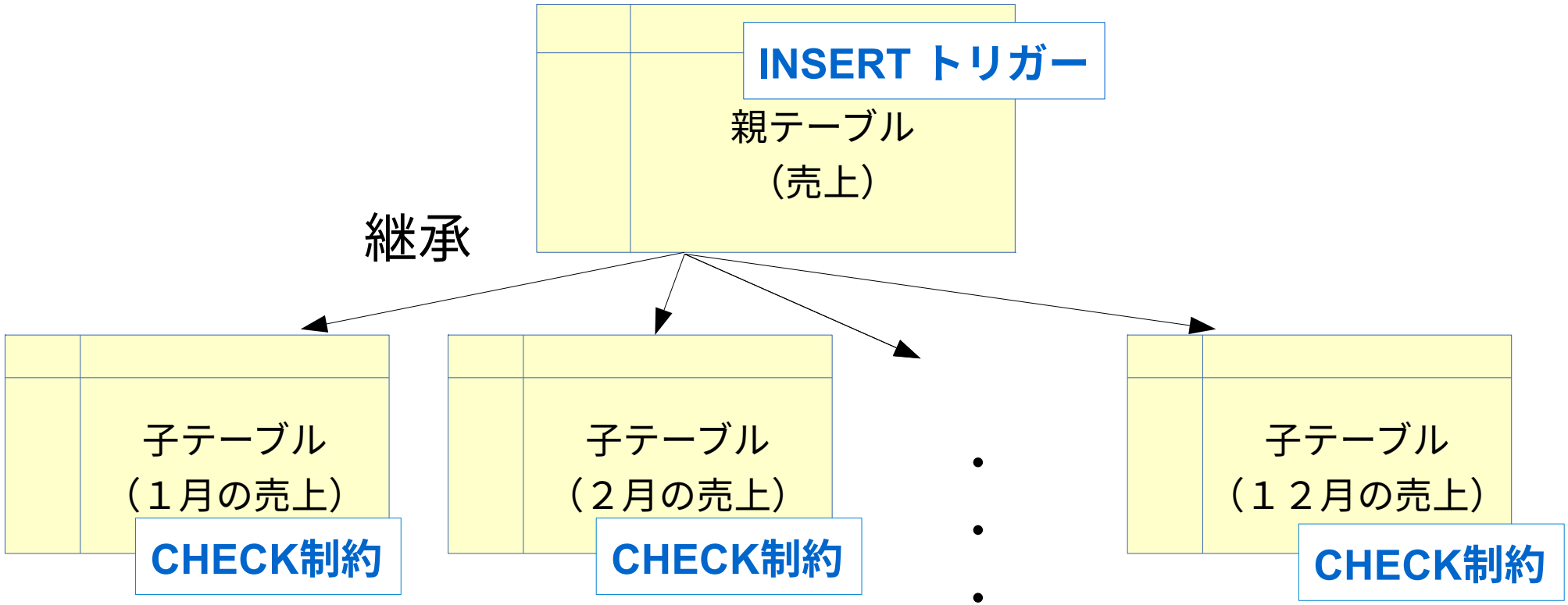
テーブルパーティショニング

- 巨大な親テーブルを複数の子テーブルに分割
  - 親テーブルへの検索時、必要な子テーブルのみを検索
  - 挿入時には適切な子テーブルに振り分け



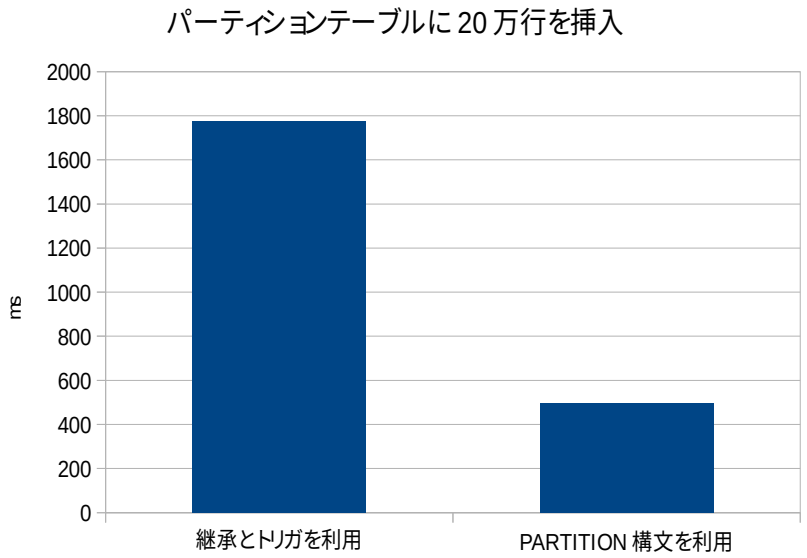
継承と制約によるパーティショニング

- 子テーブルは親テーブルを継承して作成
  - 検索時は「制約による除外」により、必要な子テーブルのみが検索される (8.1)
  - 挿入時はトリガー関数でレコードを振り分ける



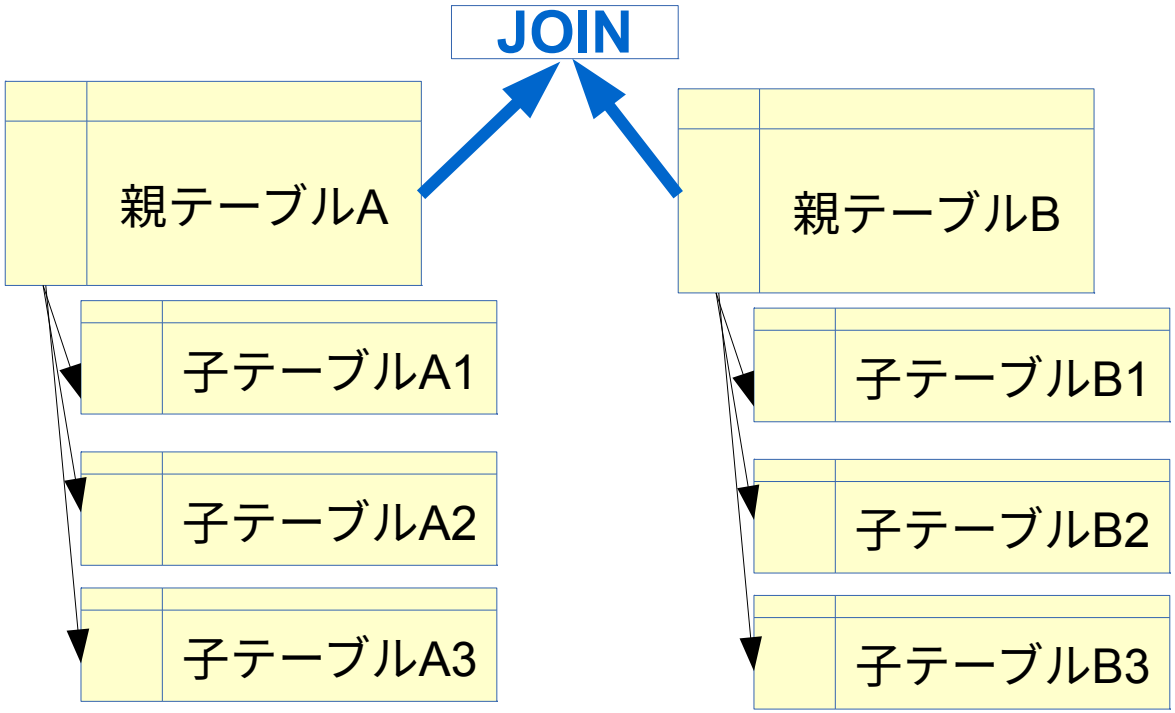
宣言的パーティショニング

- 組み込みのテーブルパーティショニング機能 (10)
  - CREATE TABLE 文でパーティショニングが構築可能
    - プランナがパーティショニング情報を考慮
    - 挿入の高速化
  - 他の機能との連携強化
    - パーティション単位の結合 (11)
    - パーティション単位の集約 (11)



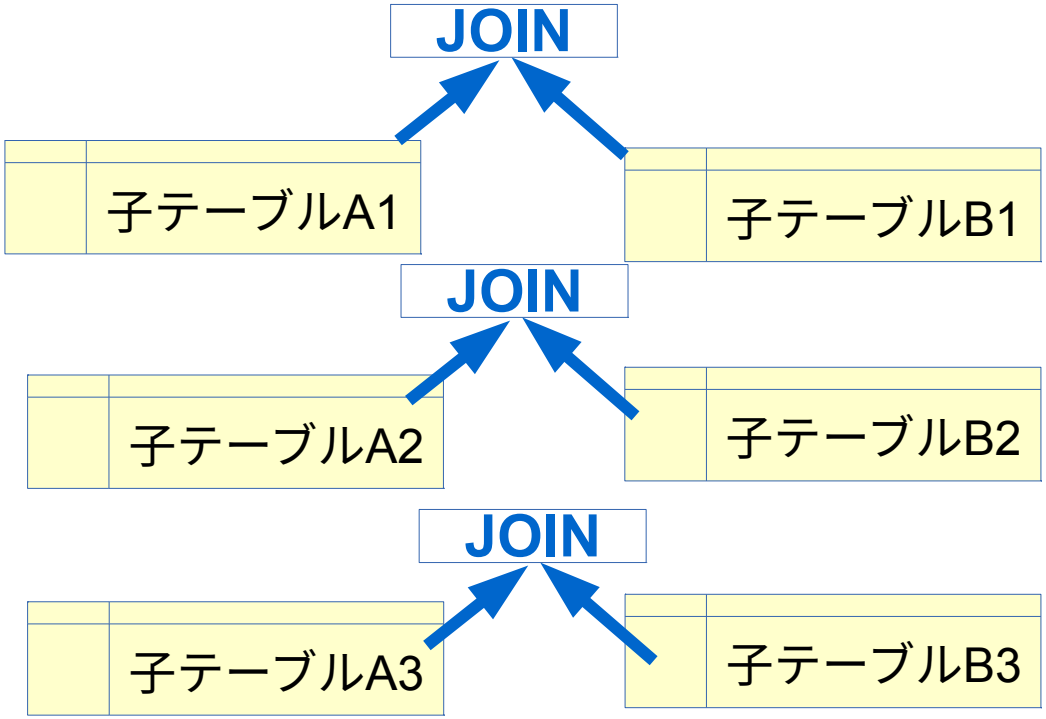
パーティション単位の結合

- 共通するパーティションキーを用いてテーブルを結合する場合：
  - パーティショニングのされ方が同じであるならば
  - テーブル全体を結合するのではなく・・・



パーティション単位の結合

- 共通するパーティションキーを用いてテーブルを結合する場合：
  - パーティショニングのされ方が同じであるならば
  - 子テーブル同士を結合した結果を連結する

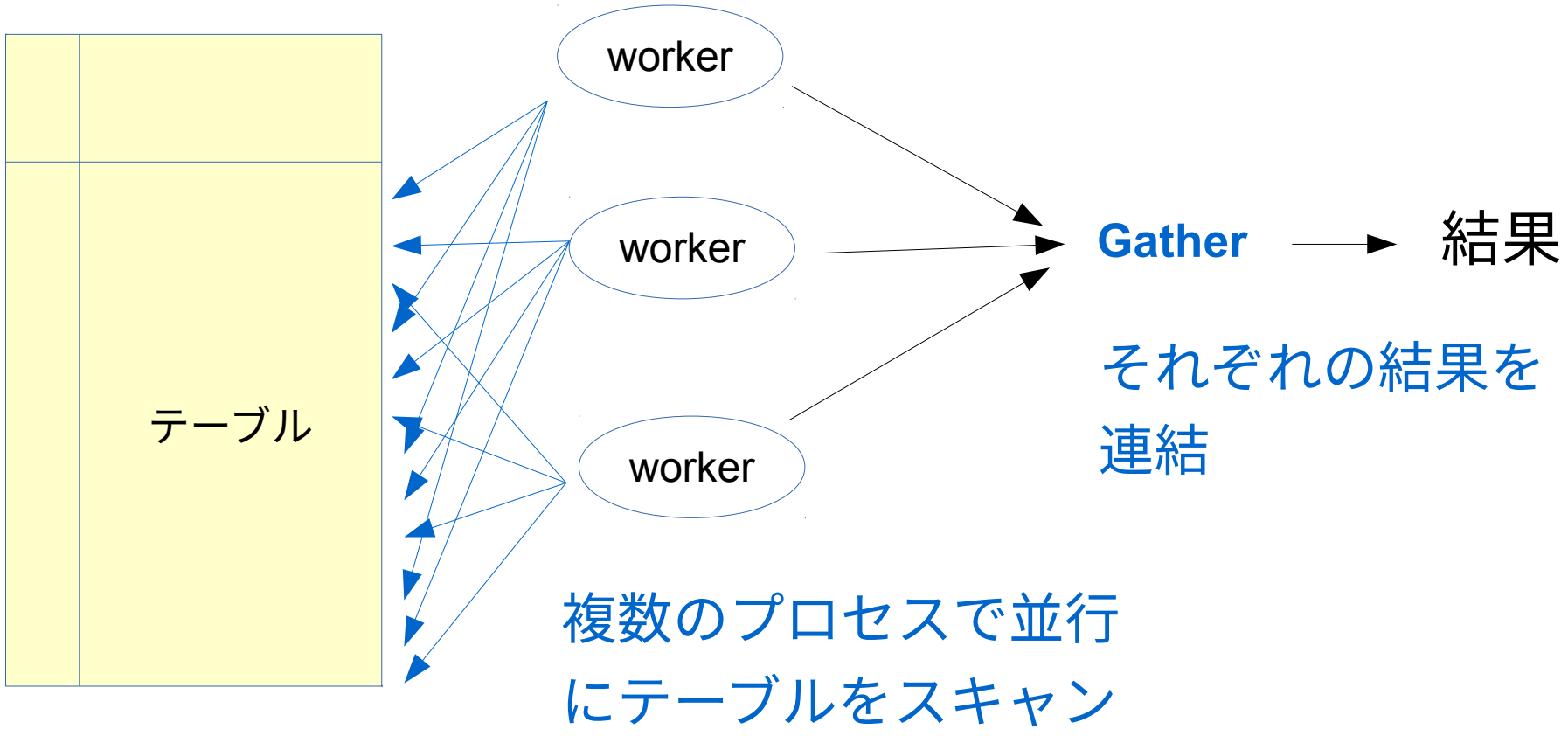


例えば、A1 と B2 ではキーの範囲が違うので結合する必要がない

パラレルクエリ

パラレルクエリ (9.6)

- 大きなテーブルに対するクエリを複数のプロセス（CPU）で並列に実行することでパフォーマンスを向上



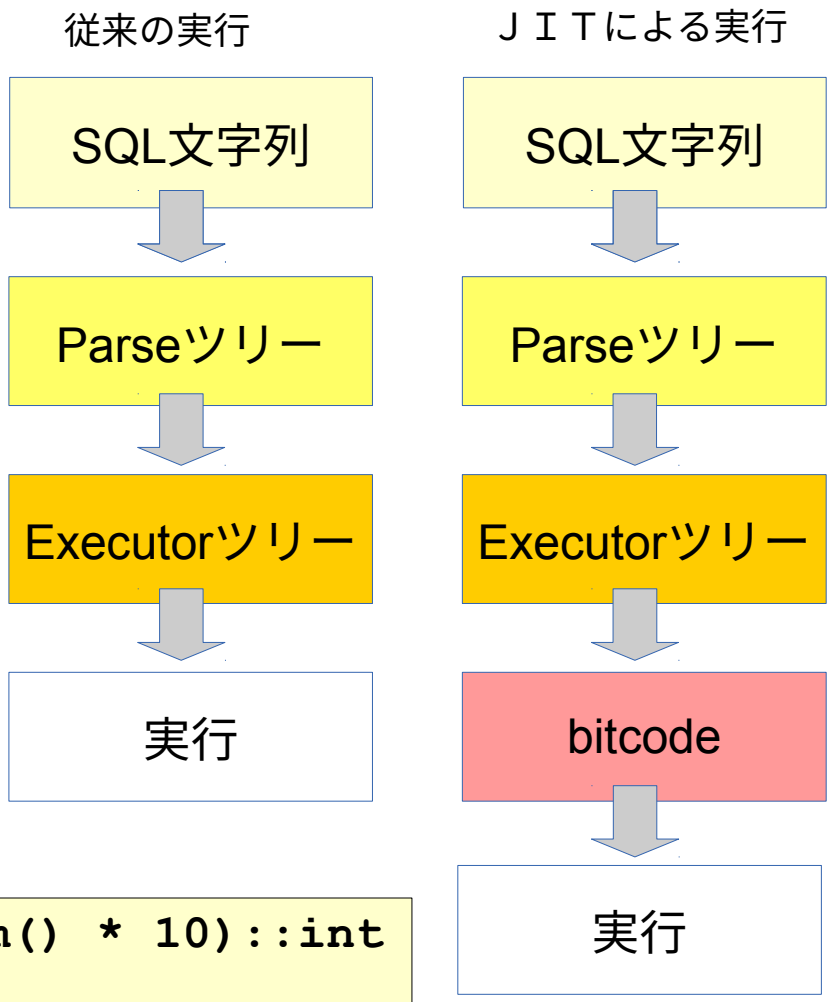
パラレルクエリの強化

- パラレル化ができる対象の広がり
  - シーケンシャルスキャン、集約、
    ネストループ結合、ハッシュ結合 (9.6)
  - B-tree インデックス、ビットマップヒープスキャン、
    マージ、副問い合わせ (10)
  - CREATE INDEX、CREATE TABLE AS、
    CREATE MATERIALIZED VIEW、UNION (11)

JIT コンパイラ基盤

JITコンパイル基盤 (11)

- SQL実行にあたって Just In Time コンパイルを行い、ネイティブ実行を実現
- LLVMコンパイラ基盤を使用
- 繰り返し処理回数が多い場合に有効
- プランナコストでJIT適用を判断
- 行データ取り出し、SELECTリストの式、WHERE句の式に適用される




```
SELECT id, c1, c2, c3, (c4 * random() * 10)::int
  FROM t1 WHERE typ = 101;
```

これからの高速化

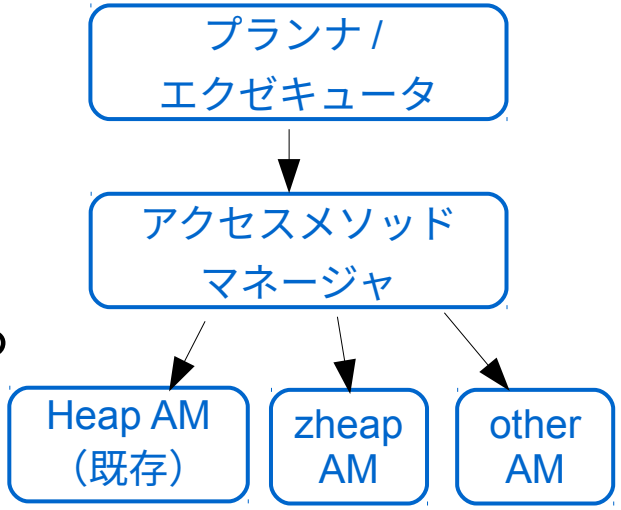
PostgreSQL 12 の予定

- Generated column
  - 「式の計算結果」をカラムとしてレコード内に保存できるSQL標準の機能
  - マテリアライズドビューのカラム版のようなもの
- Pluggable storage API（次のスライド）
  - PostgreSQL のストレージシステムの選択肢を増やすための基盤
  - 追記型 MVCC 以外のストレージによる高速化の可能性
- パーティショニングテーブルの性能改善
  - プランニングや検索の高速化
  - テーブル更新時のロック改善
- CTE（WITH句）のインライン化
  - 現行では WITH 句のクエリは「書かれた通りに」実行される（最適化バリア）
  - それを通常のサブクエリのようにクエリ内に展開して最適化の対象にできるように

Pluggable Storage

- PostgreSQL で使用可能なストレージシステムの選択肢を増やすための基盤
  - テーブル作成時に、USING 句を使ってアクセスメソッド（AM）を指定できるようになる

```
CREATE TABLE tbl(...) USING heap;
```



    - 省略時は default_table_access_method の値（デフォルト ‘heap’）が使われる
  - 新しいアクセスメソッドは CREATE ACCESS METHOD 文 (9.6) で作成できる

```
CREATE ACCESS METHOD myheap
  TYPE TABLE HANDLER my_heap_tableam_handler;
```

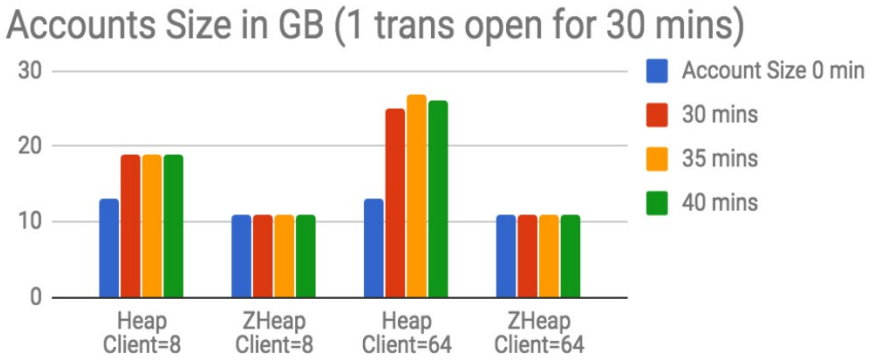
さらに PostgreSQL 13 以降…

- Pluggable storage API を利用した新しいストレージの提案
  - UNDOログに基づくストレージ形式（zheap）
  - 組み込みのインメモリ列指向ストレージ（Zedstore）
- 組み込みシャーディング
  - テーブルパーティショニングと FDW を利用したシャーディング
- 組み込みのコネクションプール
  - 現在はクライアント接続の度にプロセスが生成されており性能に影響を与えている
  - プロセスを複数の接続で使いまわす仕組みが提案されている
- より効率的なプランの生成
  - Loose index scan, Incremental sort, ...
- マテリアライズドビューの差分更新
  - 全体をリフレッシュするのではなく、必要な差分だけを計算して更新する
  - SRA OSS, Inc. Japan が現在取り組んでいるテーマ

zheap

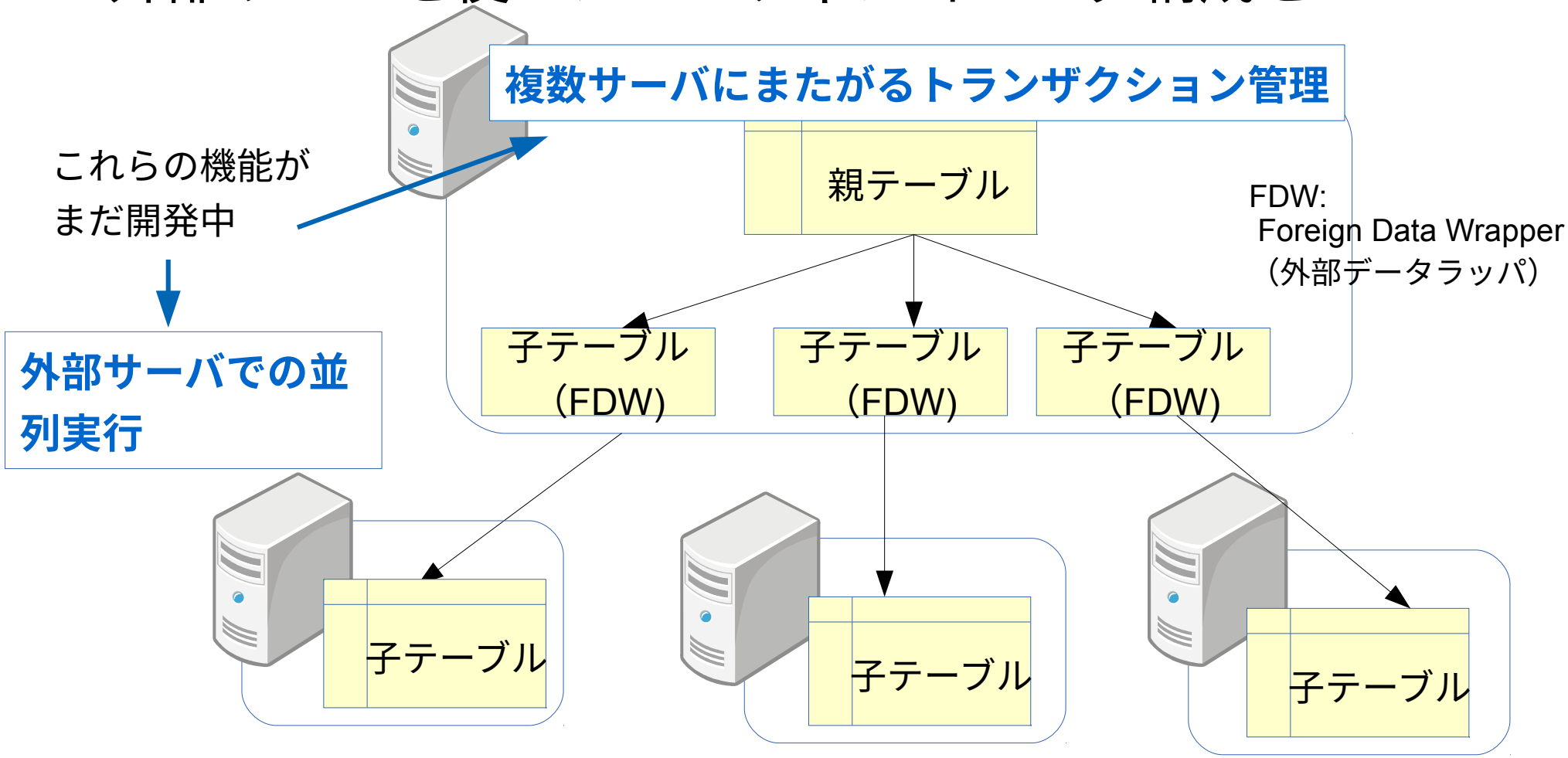
- UNDOログに基づくストレージ形式
  - EnterpriseDB 社で開発が進められている
  - UNDO ログ
    - データを「元の古い状態に戻す」ためのトランザクションログ
    - PostgreSQL では古い行を残したまま更新を行う（追記型）ので、 UNDOログがない。
    - 古い行が残ったままだと、テーブルサイズの肥大化が発生する

      → VACUUM の必要性
  - UNDO ログを使うと
    - テーブルサイズが小さく抑えられ、性能も向上する
    - VACUUM も必要なくなる可能性
    - ただし、以下のデメリットもある
      - ロールバックが遅くなる
      - 他のトランザクションが存在するときの検索が遅くなる



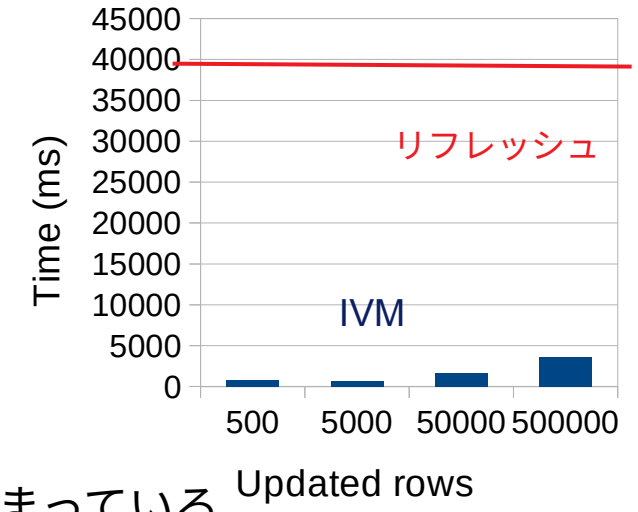
組み込みシャーディングの構想

- 外部サーバを使ったパーティショニング構成をベース



マテリアライズドビューの差分更新

- Incremental View Maintenance (IVM)
  - マテリアライズビューの更新方法の１つ
  - 通常の REFRESH では、「全データ」の再計算が必要だが、IVM では「差分」のみを計算して適用
- PGConf.EU 2018
  - テーブルの行 OID を使用した方法を提案
    - 増永先生（お茶の水大学名誉教授）のアイデアが元
  - PoC（Proof of Concept）実装
    - アイデアの実現可能性評価
    - 非常に簡約な実装で、単純なJOINビューのみ対応
- しかし、行 OID は PostgreSQL 12 で廃止されることが決まっている
  - 別の実装方法で現在開発中
    - PGCon 2019 で発表予定



おわりに

- PostgreSQL の高速化の歴史
  - ディスク負荷軽減から、マルチCPU同時実行性能の向上へ
  - インデックス、マテリアライズドビューなどの SQL 機能強化
  - プランナ / エクゼキュータ、VACUUM の改善
- 大規模データ対応 / 分析系機能
  - BRIN インデックス、パーティショニング（シャーディング）、パラレルクエリ、JIT コンパイル
- 今後の可能性
  - 《拡張性の活用》：Pluggable Storage、FDW、新しいインデックス、・・・
  - 《新機能の実装》：組み込みコネクションプーリング、IVM、・・・

参考資料

- CommitFest
  - https://commitfest.postgresql.org/
- pgsql-hackers
  - https://www.postgresql.org/list/pgsql-hackers/
- Performance Improvements in PostgreSQL 9.2 (Robert Haas, PGCon 2012)
  - https://www.pgcon.org/2012/schedule/events/416.en.html
- Faster PostgreSQL: Improved Writes (Amit Kapila, PGCon 2016)
  - https://www.pgcon.org/2016/schedule/events/915.en.html
- Towards Built-in Sharding in Community PostgreSQL (Amit Langote, Etsuro Fujita, Kyotaro Horiguchi, Masahiko Sawada , PGCon 2017)
  - https://www.pgcon.org/2017/schedule/events/1069.en.html
- Pluggable Storage in PostgreSQL (Andres Freund, PGConf.EU 2018)
- https://www.postgresql.eu/events/pgconfeu2018/schedule/session/2207-pluggable-storage-in-postgresql/
- zheap: An answer to PostgreSQL bloat woes (Amit, Kapila, PGConf.EU 2018)
  - **https://www.postgresql.eu/events/pgconfeu2018/sessions/session/2104-zheap-an-answer-to-postgresql-bloat-woes/**
- Towards more efficient query plans: PostgreSQL 11 and beyond (Alexander Kuzmenkov, PGConf.EU 2018)
  - https://www.postgresql.eu/events/pgconfeu2018/schedule/session/2124-towards-more-efficient-query-plans-postgresql-11-and-beyond/
- Implementing Incremental View Maintenance on PostgreSQL (Yugo Nagata, PGConf.EU 2018)
  - https://www.postgresql.eu/events/pgconfeu2018/schedule/session/2195-implementing-incremental-view-maintenance-on-postgresql/
- Built-in Sharding for PostgreSQL (Robert Haas blog)
  - http://rhaas.blogspot.com/2018/05/built-in-sharding-for-postgresql.html
- SRA OSS Tech Blog
  - https://www.sraoss.co.jp/tech-blog/
- PostgreSQL 技術情報 (SRA OSS, Inc. 日本支社)
  - https://www.sraoss.co.jp/technology/postgresql/
- セミナー資料 (SRA OSS, Inc. 日本支社)
  - https://www.sraoss.co.jp/event_seminar/material.php