PostgreSQLインサイド概要

SRA OSS, Inc. 日本支社
石井達夫

アジェンダ

- PostgreSQLの概要
- PostgreSQLの構造概要
- PostgreSQLの実装と内部構造
- 問い合わせ処理の流れ

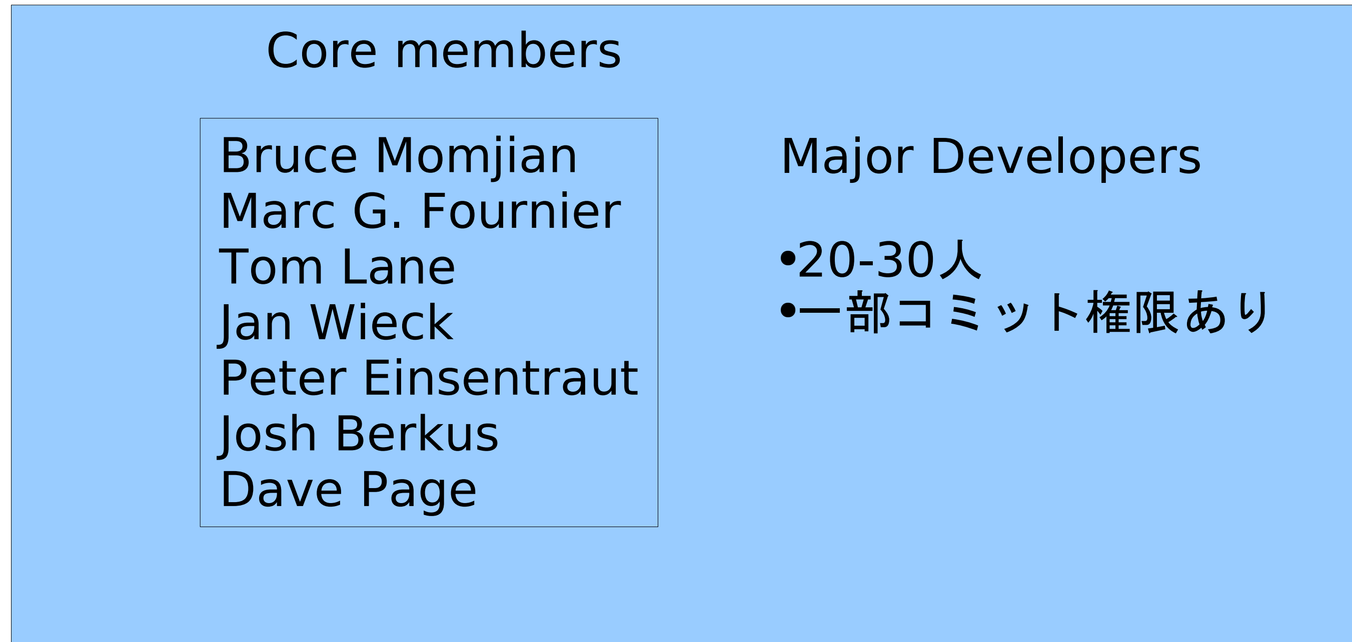
PostgreSQLの概要

PostgreSQLの概要：歴史

１９７７年スタート	１９８６年スタート	１９９４年スタート	１９９６年スタート
最初期の関係データベースの実装 クローズドソース PostgreSQLとのコードの直接の関連性なし	次世代関係データベースシステム フリーソフト Object-Relational Database PostgreSQLの直接の祖先	ＳＱＬ言語の実装 ＡＮＳＩ／Ｃによる書き換え ２５％の高速化 GNU makeの採用 178,976ステップ	PostgreSQL Global Development Groupによるインターネットベースの開発 689,833ステップ

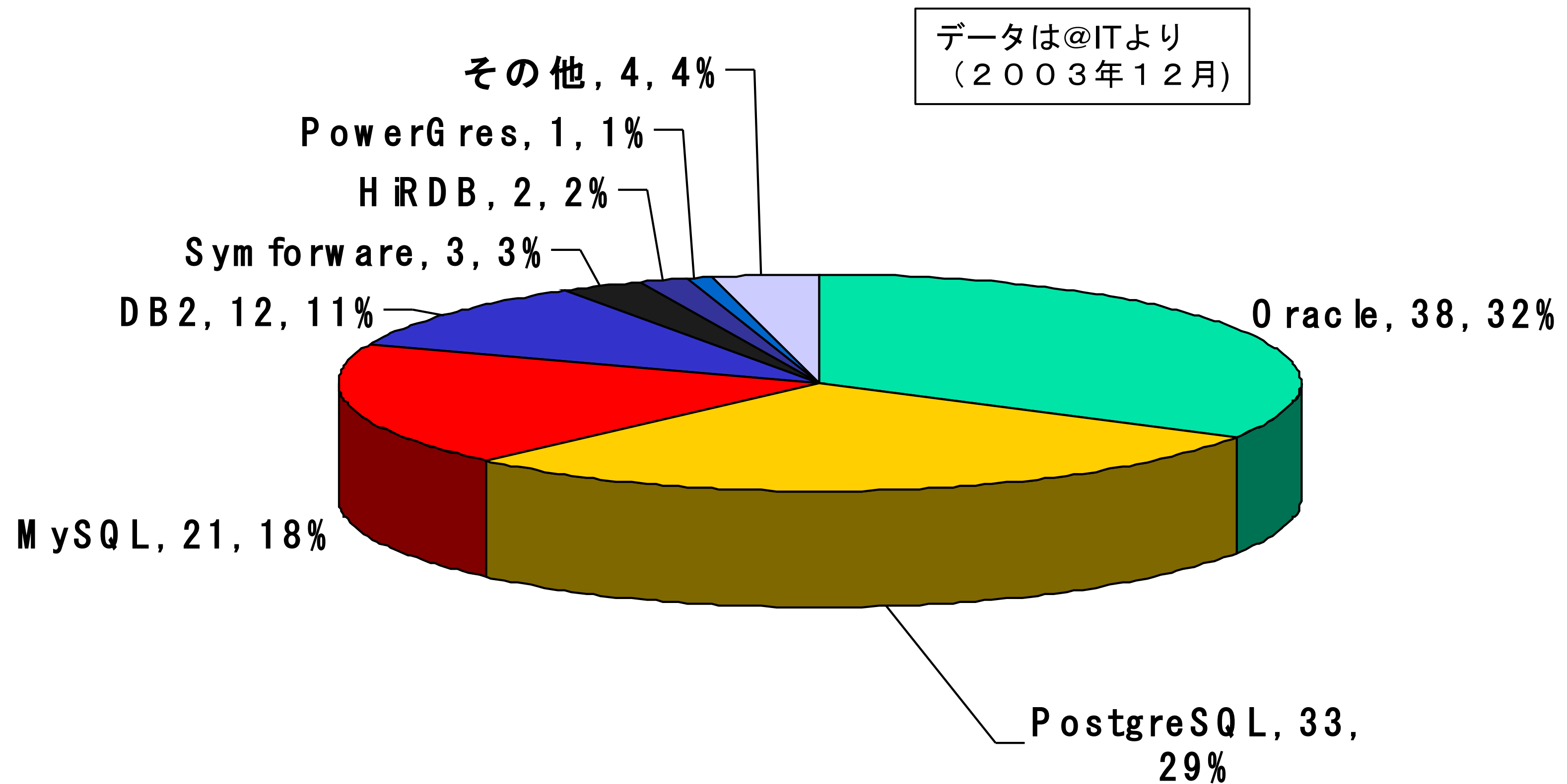
p14

PostgreSQLの概要：開発体制

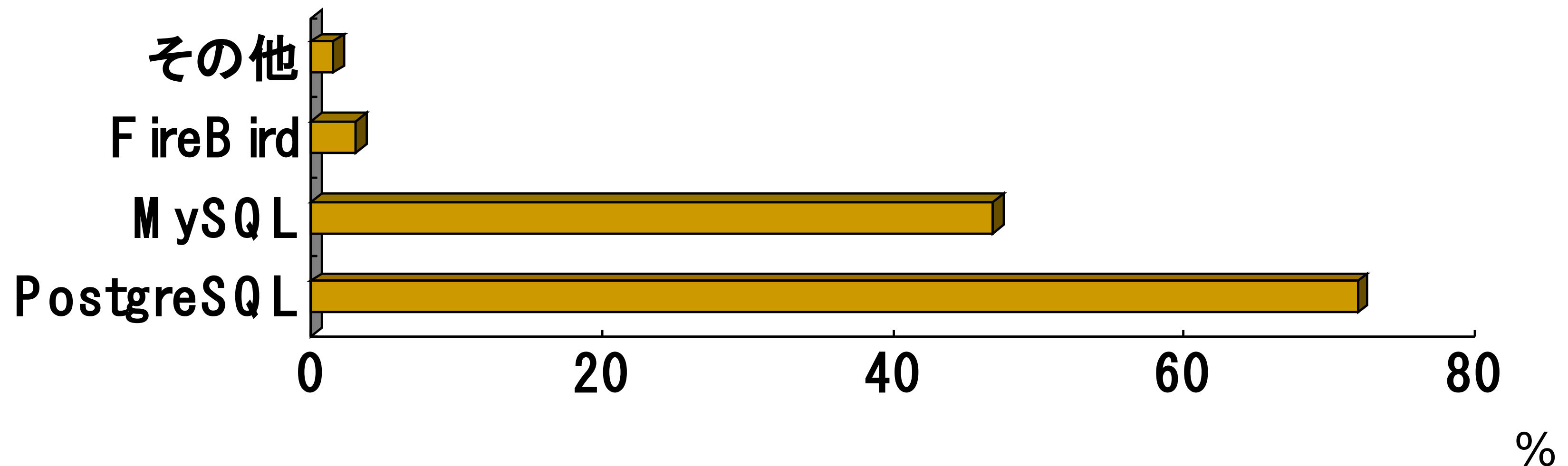


- ボランティアによる自主独立プロジェクト
- サーバ類もボランティアの提供
- メインソースはcvs.postgresql.orgで管理，アプリケーション，関連プログラムはpgfoundry.postgresql.orgでホスティング

PostgreSQLのLinux上のシェア



Linux市場でのオープンソースデータベースのシェア



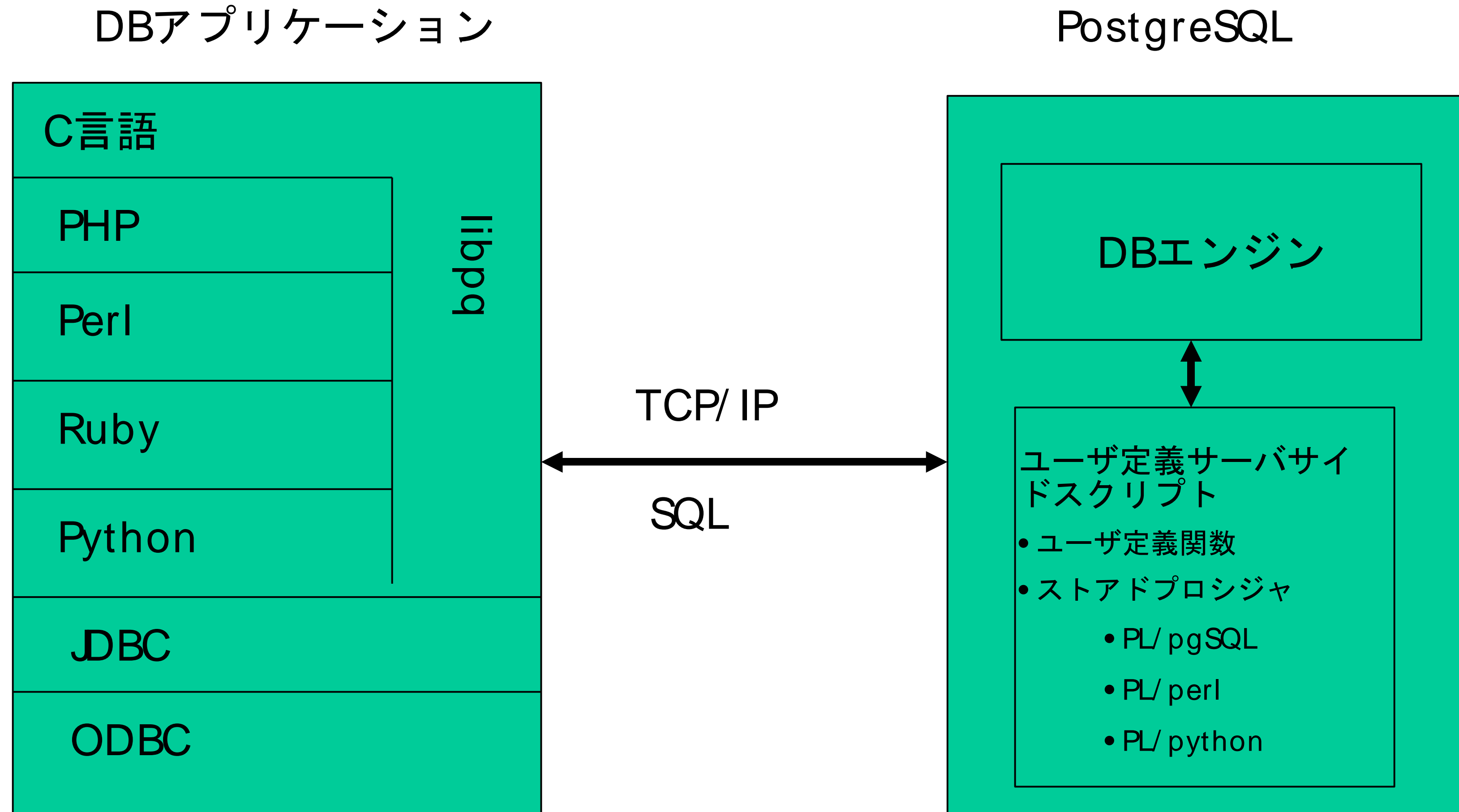
データは2005年の矢野経済研究所のアンケート調査より

PostgreSQLの主な機能

- SQL:2003の主要部分を実装
  - DDL, DML, Schema
  - 副問い合わせ，トリガー，外部キー，ビュー
- 高度なトランザクション機能
  - 2レベルのトランザクション分離レベル
  - トランザクションログによるリカバリ
  - アーカイブログ(PITR: Point In Time Recovery)
  - セーブポイント
  - 2相コミット
  - 行ロック，テーブルロック
  - MVCC(Multi Version Concurrency Control)

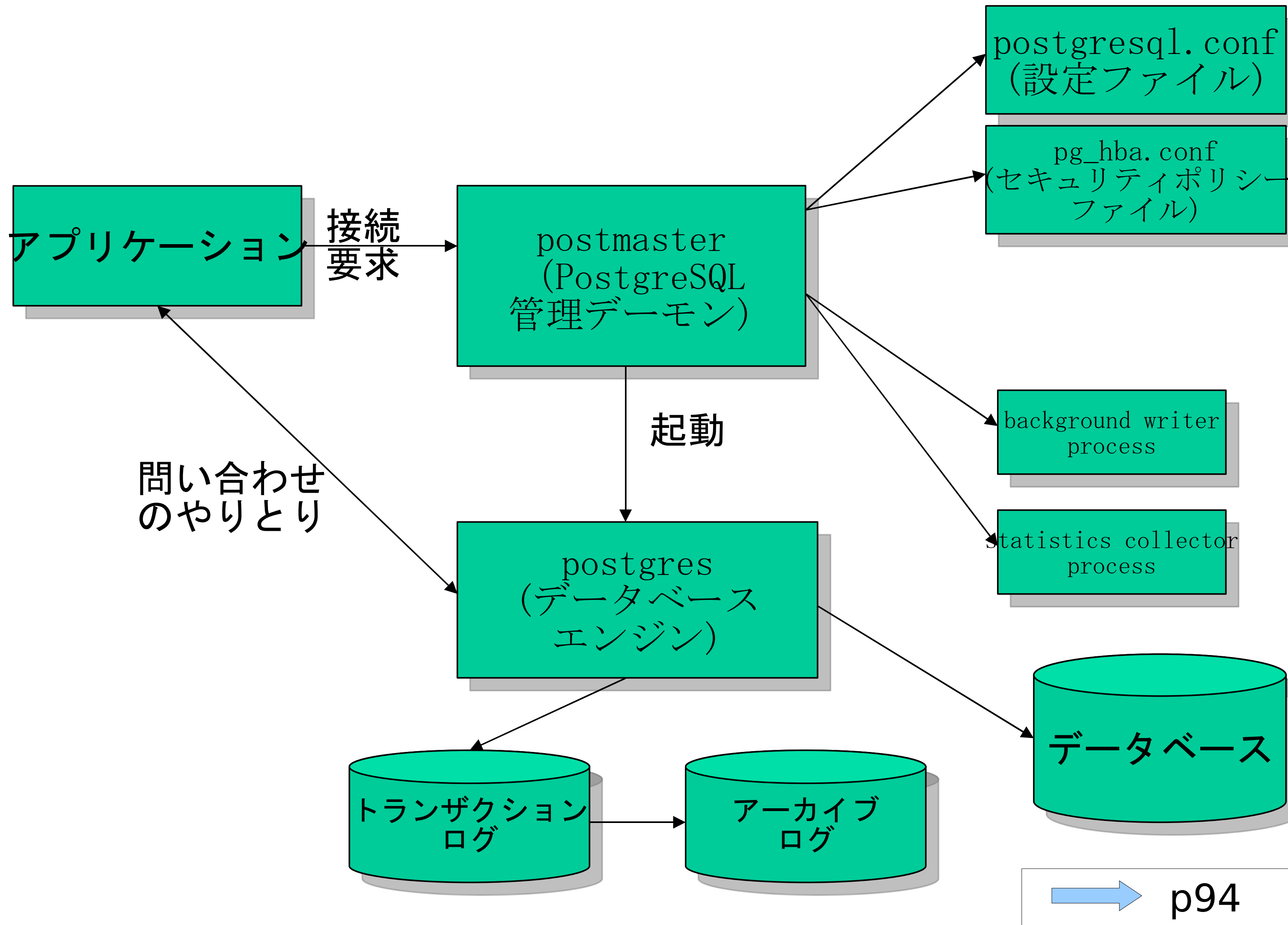
PostgreSQLの構造概要

PostgreSQLの利用形態



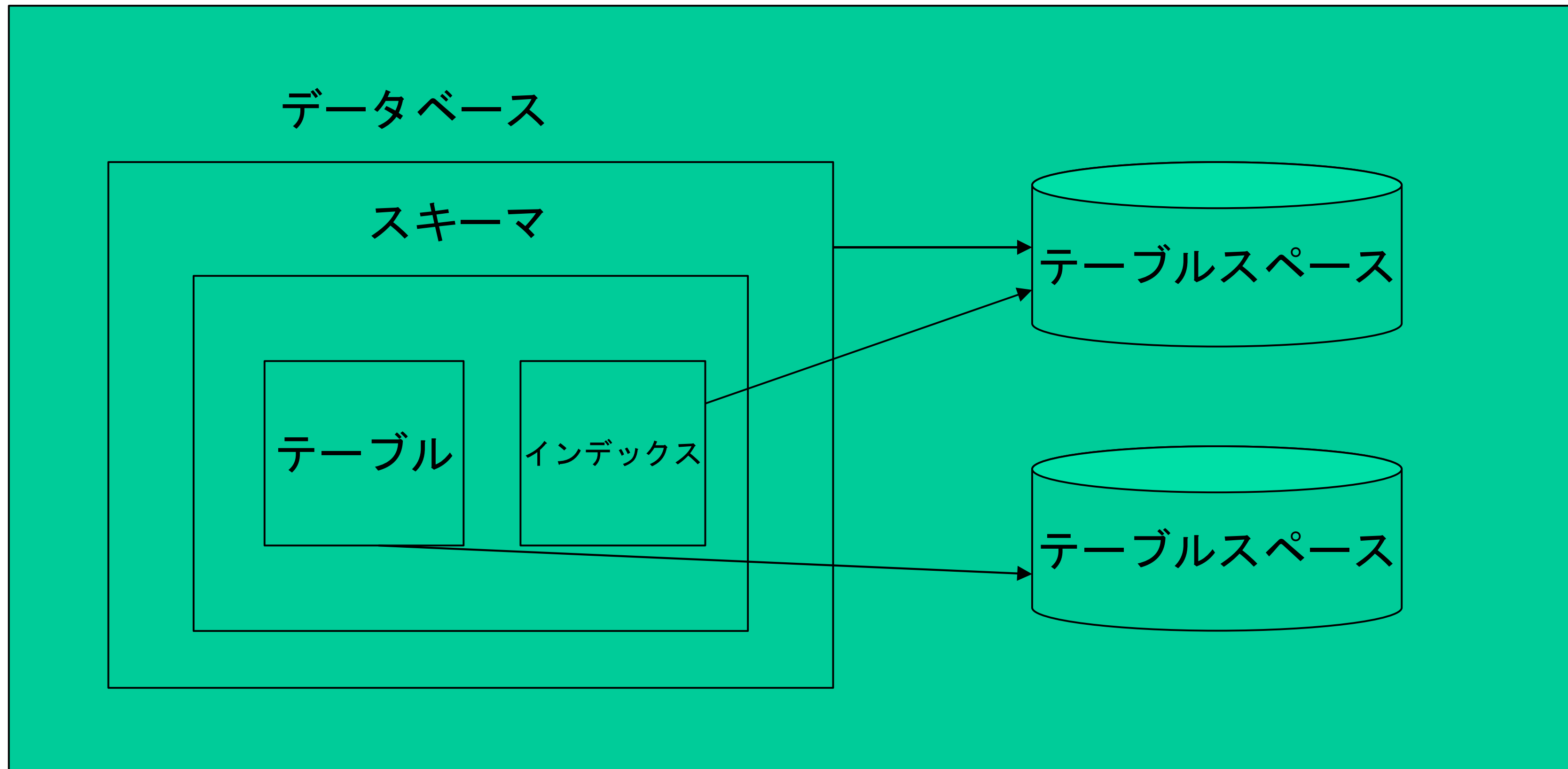
p93

PostgreSQLのプロセス構造



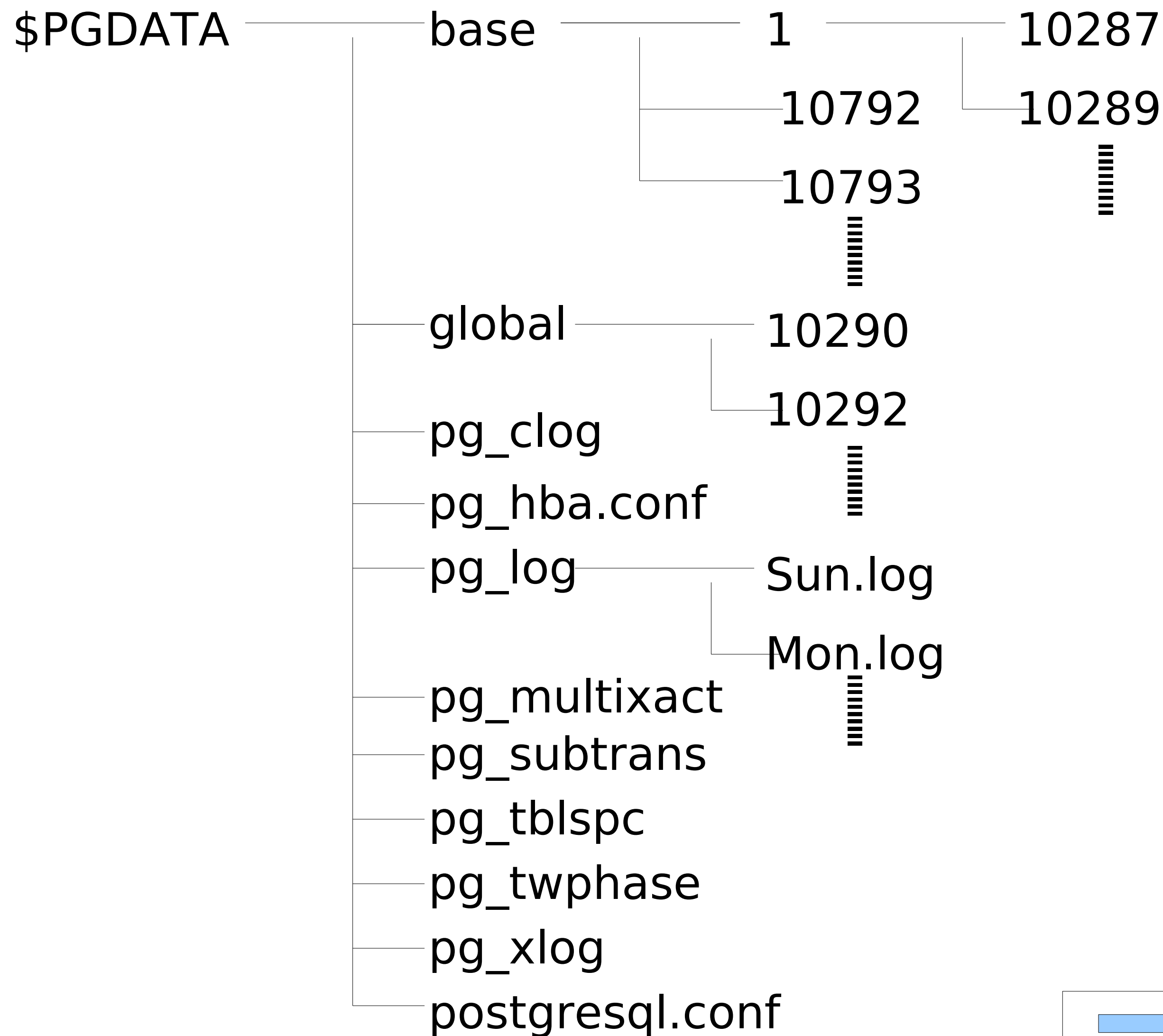
p94

PostgreSQLのデータ構造



p96

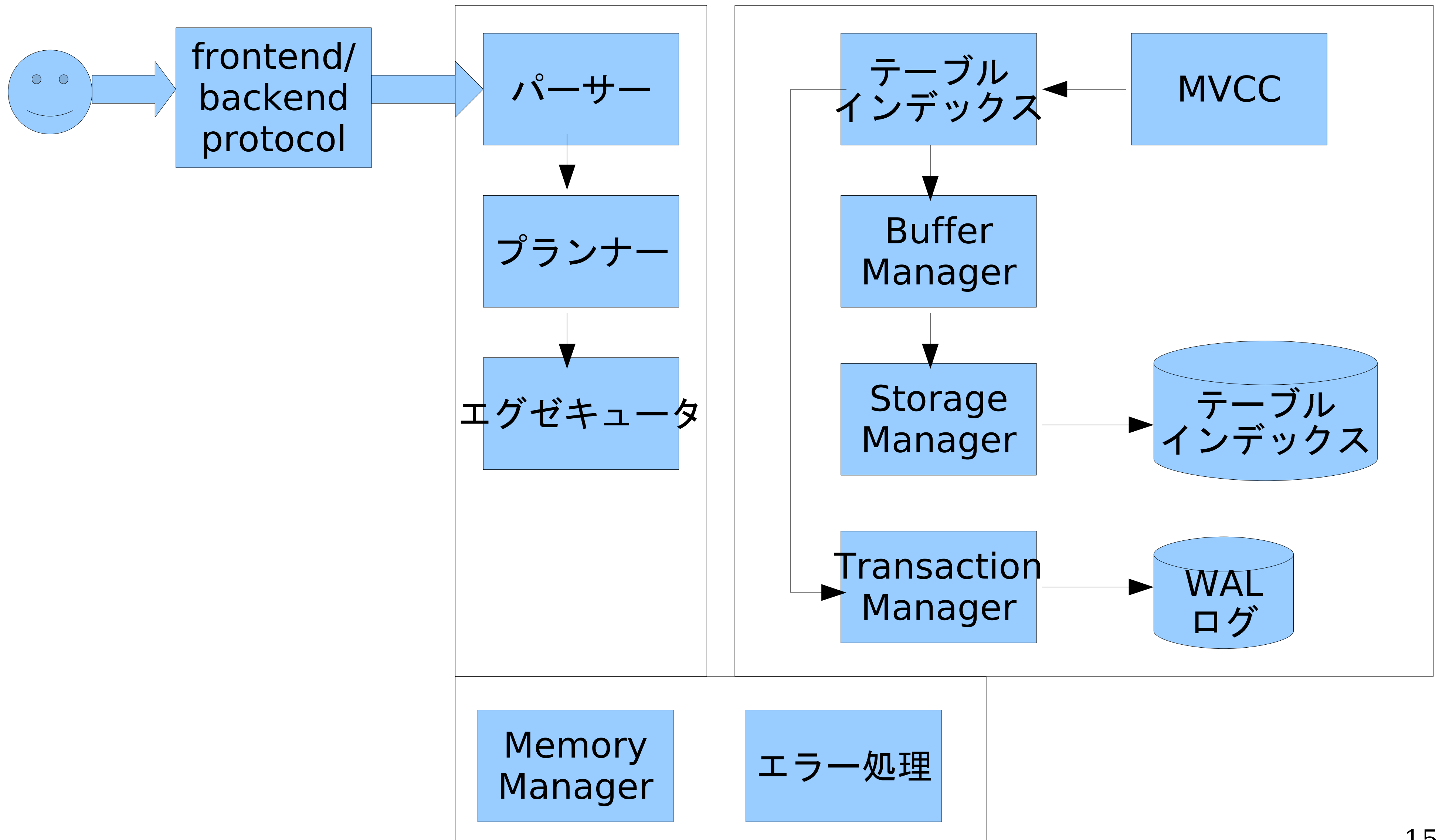
データベースクラスタの構造



p98

PostgreSQLの実装と内部構造

サブシステムとその関係



メモリーマネージャ

- メモリーリークの防止
  - 「メモリコンテキスト」(memory context)
    - コンテキスト単位で一括メモリ解放
      - トランザクション単位のメモリコンテキスト
      - SQL文単位のメモリコンテキスト
- メモリーアロケーション効率の向上
  - 8KB単位でアロケーション
- API
  - mallocの代わりにpalloc，freeの代わりにpfreeを使用

エラー/例外処理

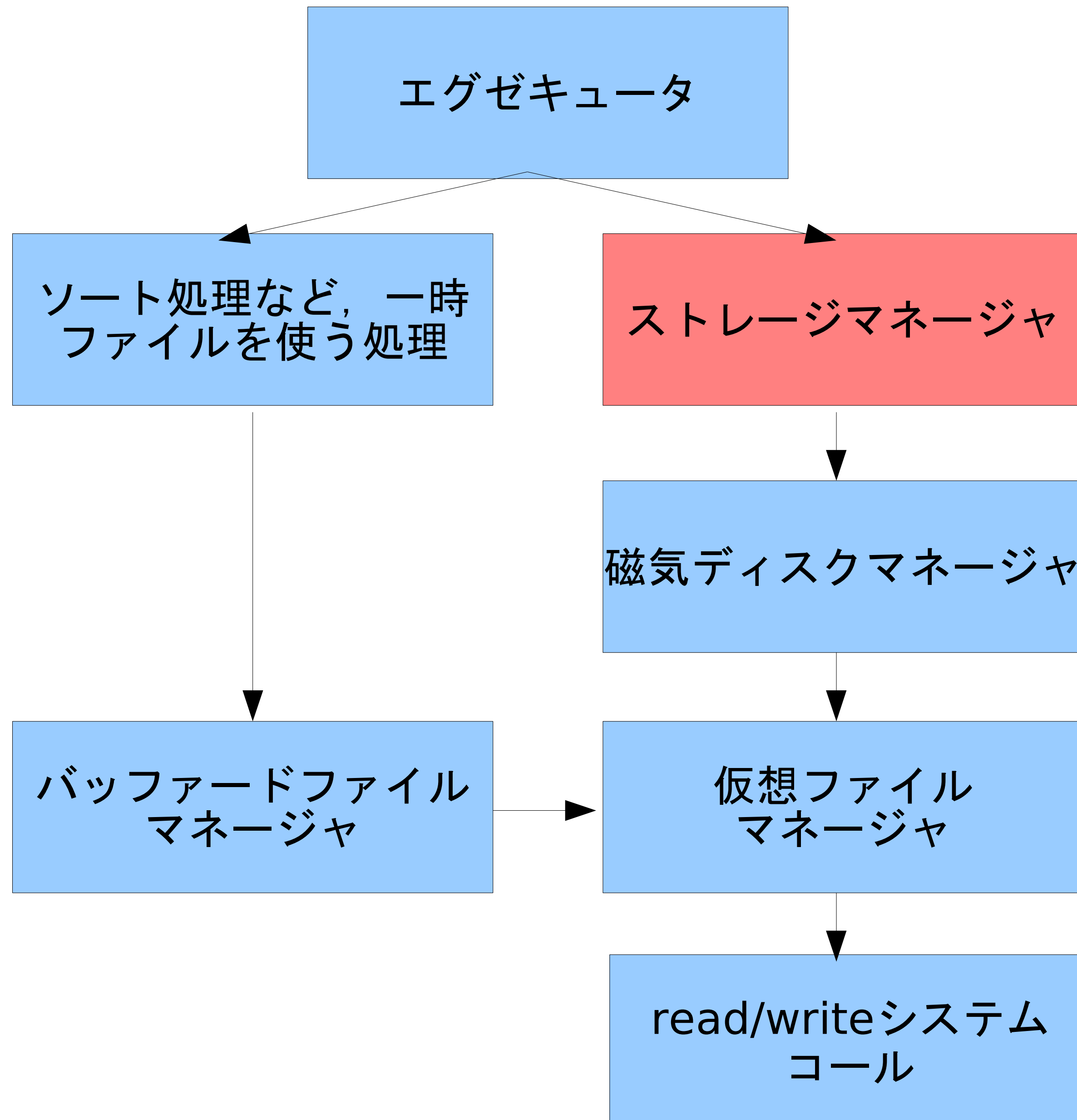
- long jumpを利用して例外処理を実装
  - エラー発生時にトランザクションをアボートし，メモリを解放する
- コードの簡素化
- エラー処理の集中管理
  - エラー番号(SQL標準のエラー番号+PostgreSQL独自)
  - エラーメッセージ
    - gettextによるメッセージカタログ
    - メッセージ出力先の柔軟な切り替え
- API: elog()

ストレージマネージャ

- ストレージ層を抽象化
- 実際には磁気ディスクのみサポート
- API
  - smgr_init
  - smgr_shutdown
  - smgr_close
  - smgr_create
  -

- API
  - smgr_unlink
  - smgr_extend
  - smgr_read
  - smgr_write
  - smgr_nblocks
  - smgr_truncate
  - smgr_immdesync
  - smgr_commit
  - smgr_abort
  - smgr_sync

ストレージマネージャの位置づけ



アクセスメソッドの種類(1)

- ヒープ

  - データが順に並ぶ単純な構造，テーブルに使用
- B+tree

  - PostgreSQLの主要なインデックス，性能や効率が優れている．スカラー値の範囲検索
- Hash

  - 値の一致だけを検索条件にする
  - 効率が悪く，ログが記録されないので非推奨
- R-tree

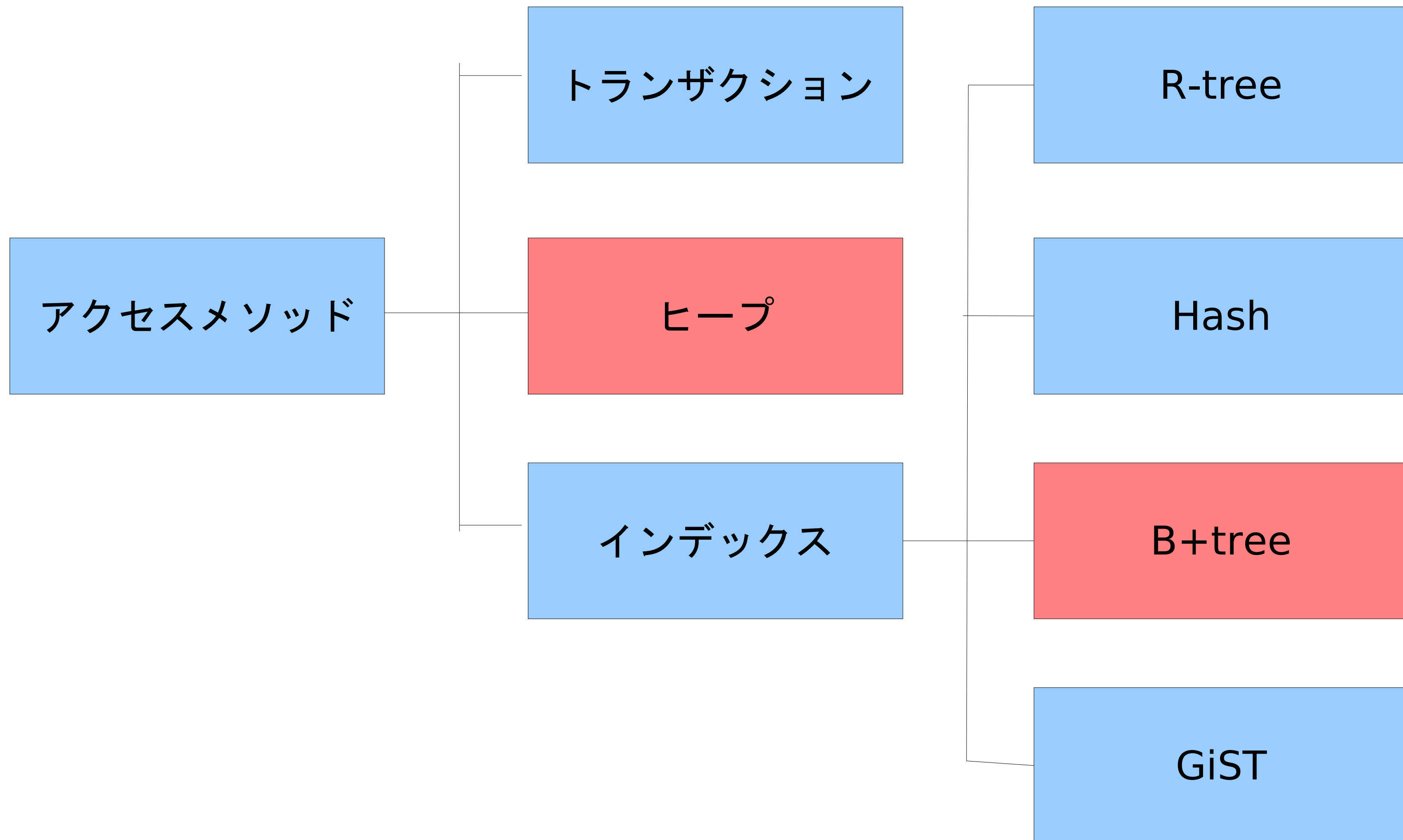
  - 空間範囲検索(「含むかどうか」など)

アクセスメソッドの種類(2)

- GiSTインデックス
  - 汎用的かつ抽象化されたインデックス
    - consistent, union, compress, decompress, penalty, picksplit, sameの7つの演算子クラス用メソッドを定義すればどのようなデータもインデックス可能
  - 応用例
    - B-tree，全文検索，木構造探索など
  - R-treeと同等の実装を含む
    - R-treeの実装は，ログが記録されないなどの問題あり
    - R-treeは今後サポートされない予定

access/gist/README

アクセスメソッドの体系



ヒープの物理構造

テーブルファイル

８１９２バイト
固定長のブロック
(ページ)

ラインポインター

空き領域

ヒープタプル

ヒープタプル

ヒープタプルの構造

個々のタプルは「タプルID」(Tuple ID:TID)で識別される．タプルIDはOID (Object Identifier)と違って更新などで変化する.

タプルID(ヒープタプルの物理的な位置) =

ブロックID(32bit)+ブロック内のオフセット(16bit)

32TBまでの大きさのテーブルを扱うことができる(ブロックサイズが標準の8KBの場合)

include/access/htup.h

タプルヘッダーの構造

t_xmin	4	行を挿入したトランザクションID
t_cmin	4	行を挿入したコマンドID
t_xmax	4	行を削除したトランザクションID
t_cmax/t_xvac	4	行を削除したコマンドIDまたは VACUUMによって移動された行のバージョン
t_ctid	6	この行あるいは新しい行のTID(タプルID)
t_natts	2	列の数
t_infomask	2	フラグビット
t_hoff	1	行データへのオフセット

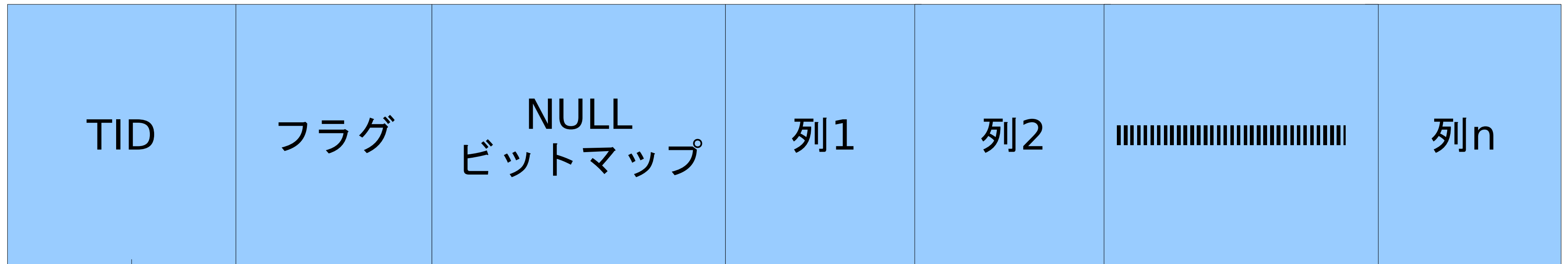
include/access/htup.h

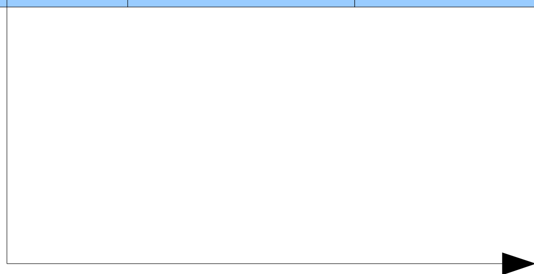
インデックスタプルの構造

最大32列のマルチカラム
インデックス作成可能

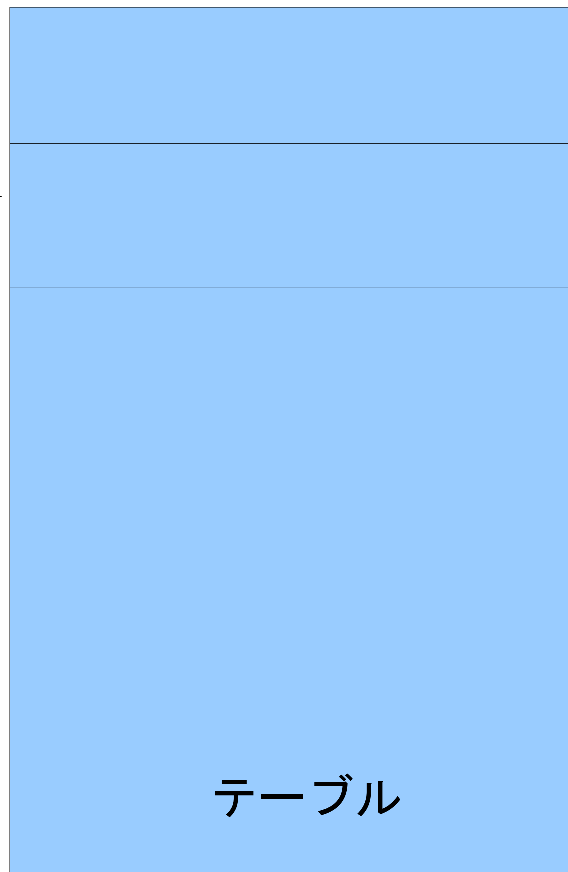
include/access/itup.h

テーブルとインデックスの関係

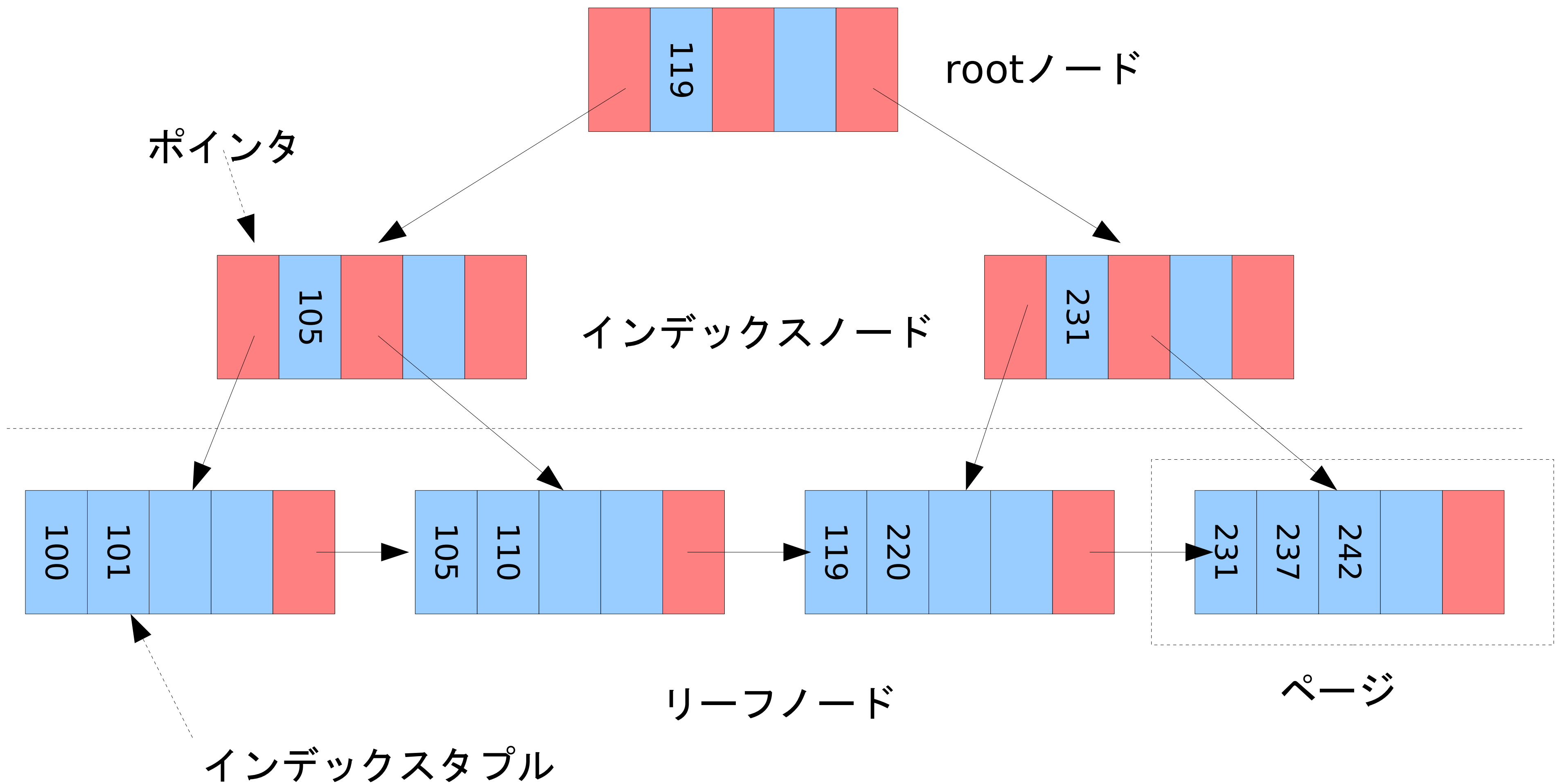




インデックスタプル



B+treeインデックスの構造



MVCC(Multi Version Concurrency Control)

- 行をバージョン管理する
- 更新中もその行を読み出せるのでトランザクションの並列実行性に優れている
  - 実際には一つ前のバージョンを読み出す
- Oracleで「読み取り一貫性」と呼ばれている機能に相当
  - 実装は全く異なる
- どのトランザクションからも参照されなくなった行はVACUUMコマンドで削除

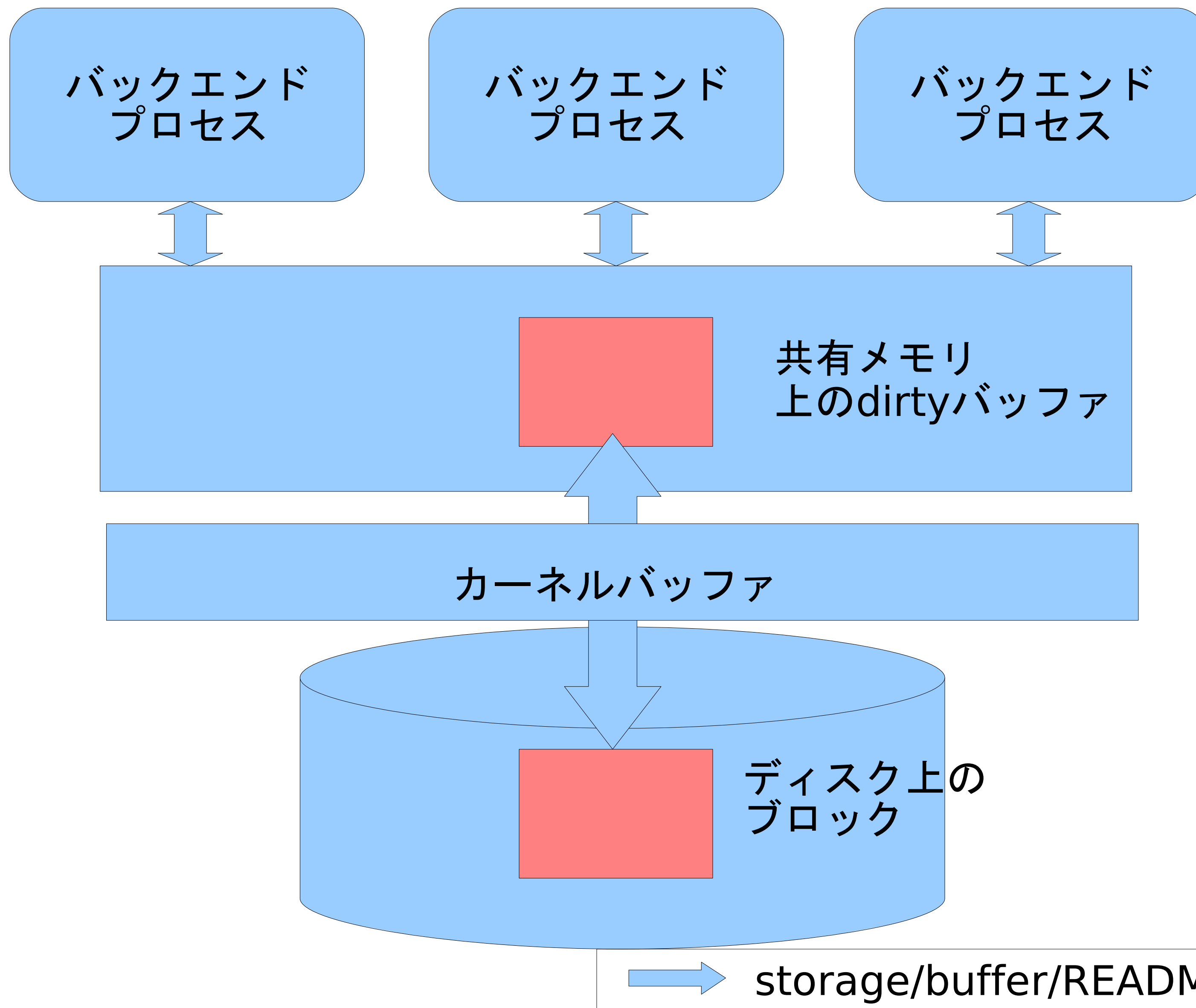
MVCCとタプルの可視性ルールの例

t_xmin	4	行を挿入したトランザクションID
t_cmin	4	行を挿入したコマンドID
t_xmax	4	行を削除したトランザクションID
t_cmax/t_xvac	4	行を削除したコマンドIDまたは VACUUMによって移動された行のバージョン
t_ctid	6	この行あるいは新しい行のTID(タプルID)

- 同一トランザクション内では
  - 自分よりもコマンドIDが小さければ可視(ハロウィーン問題の回避)
- 異なるトランザクションでは
  - 自分よりもxminが小さければ可視(削除，更新された行を除く)

utils/time/tqual.c

バッファキャッシュ管理



storage/buffer/README

バッファキャッシュからディスクへの書き込みタイミング

- dirtyバッファの再利用が必要になったとき
- バックグラウンドライタープロセスが書き込んだとき
- チェックポイントが発生したとき
  - このときに同期書き込み

システムカタログ

- DBのメタ情報を管理
  - DBの一覧,テーブルの一覧…
- DBローカルなシステムカタログとDB共有のシステムカタログがある
- DB共有システムカタログ
  - pg_authid, pg_tablespace, pg_shdepened, pg_database
- システムカタログを動的に作ることはできない

→ include/catalog

キャッシュマネージャ

- ヒープメモリ上のローカルなキャッシュ
  - 主にシステムカタログアクセスの高速化目的
- キャッシュの種類
  - カタログキャッシュ(システムカタログのキャッシュ)
  - リレーションディスクリプタキャッシュ(pg_classのキャッシュ)
  - 型キャッシュ
- キャッシュの無効化
  - 共有メモリを使った「メッセージ」を各バックエンドに送る

utils/cache/

ロックマネージャ

- 3種類のロック
  - スピンロック
  - 軽量ロック
  - 重量ロック
- 重量ロックの機能
  - SQLのLOCK文
  - 行ロック,テーブルロックの管理
  - デッドロックの検出
    - waiting graphの利用
  - トランザクション終了時に自動的にロックを解放

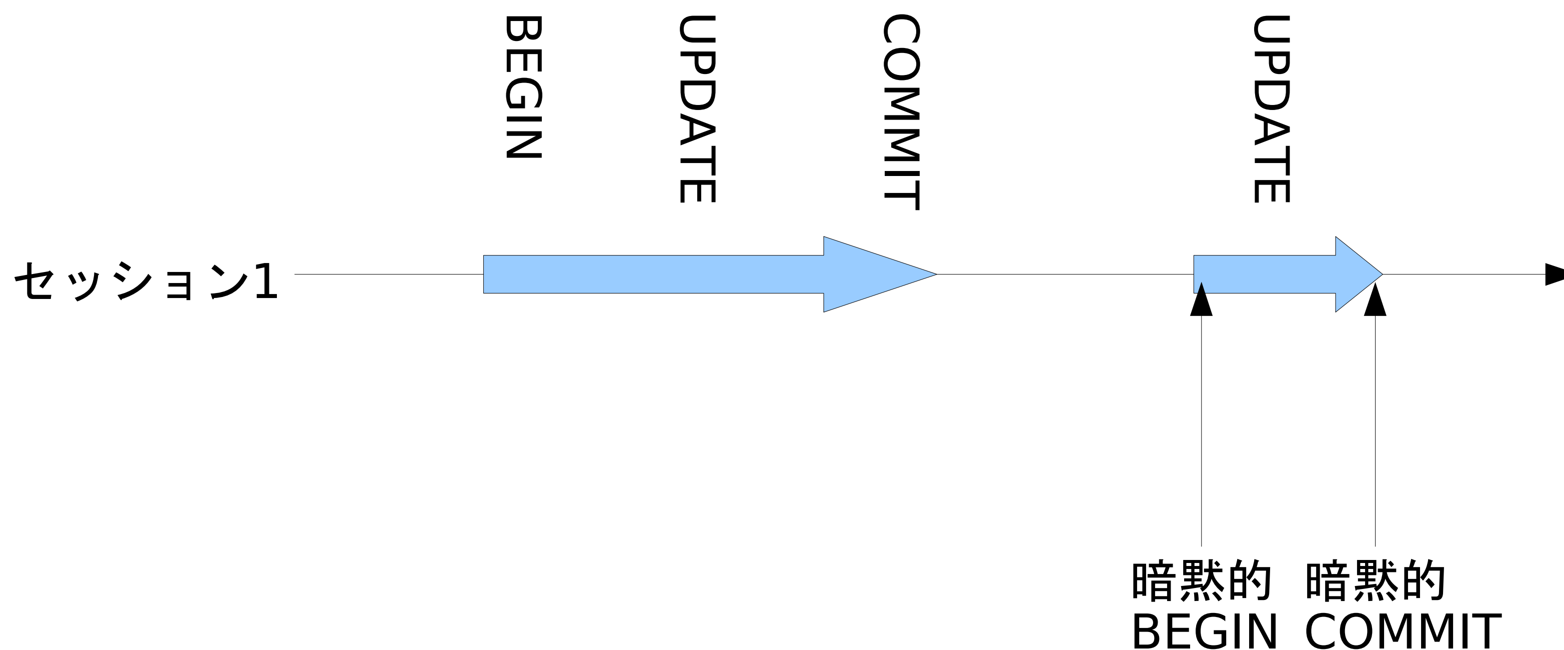
storage/lmgr/README

トランザクション管理

- トランザクションの競合の調停
  - ロック,MVCCの利用
- デッドロックの検出
  - waiting graphの利用
- トランザクションとデータの可視性の管理
  - タプルヘッダーのxmin, xmaxなどを利用
  - SERIALIZABLEとREAD COMMITTEDのサポート
  - MVCCとの兼ね合い
- トランザクションログの管理

storage/access/transam

明示的なトランザクションと暗黙的なトランザクション



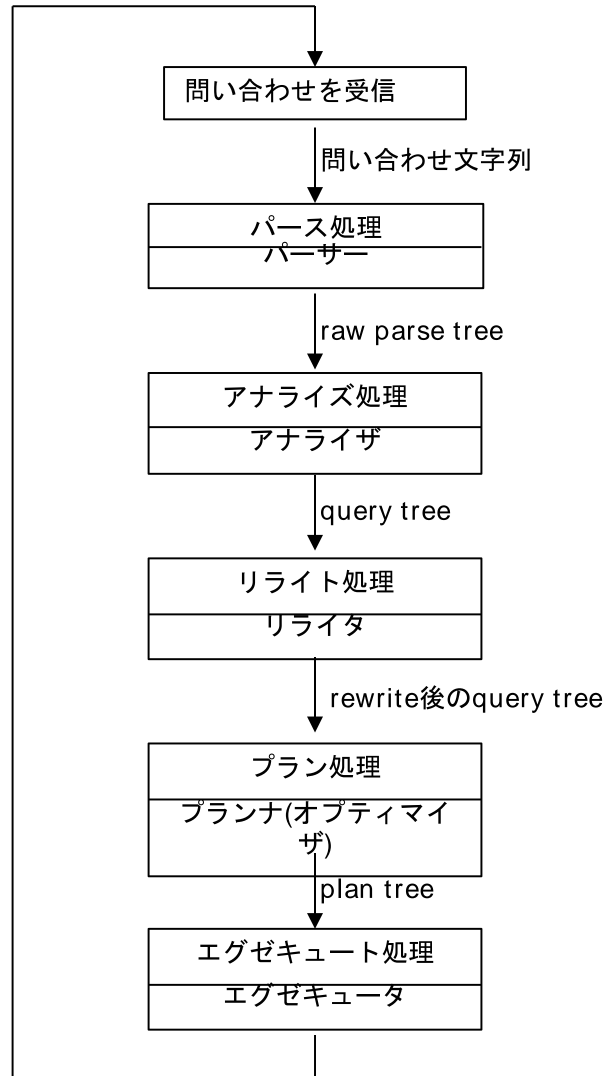
フロントエンド・バックエンドプロトコル

- TCP/IP上のアプリケーションレベルのプロトコル
  - 接続開始,問い合わせの実行要求,結果の返却など
  - 行データは基本的に文字列でやりとり
- フロントエンド用,バックエンド用のC言語API(libpq)
- libpqを使わずに独自実装も可能
  - JDBC, ODBC, pgpool

→ libpq

問い合わせ処理の流れ

問い合わせ処理の流れ

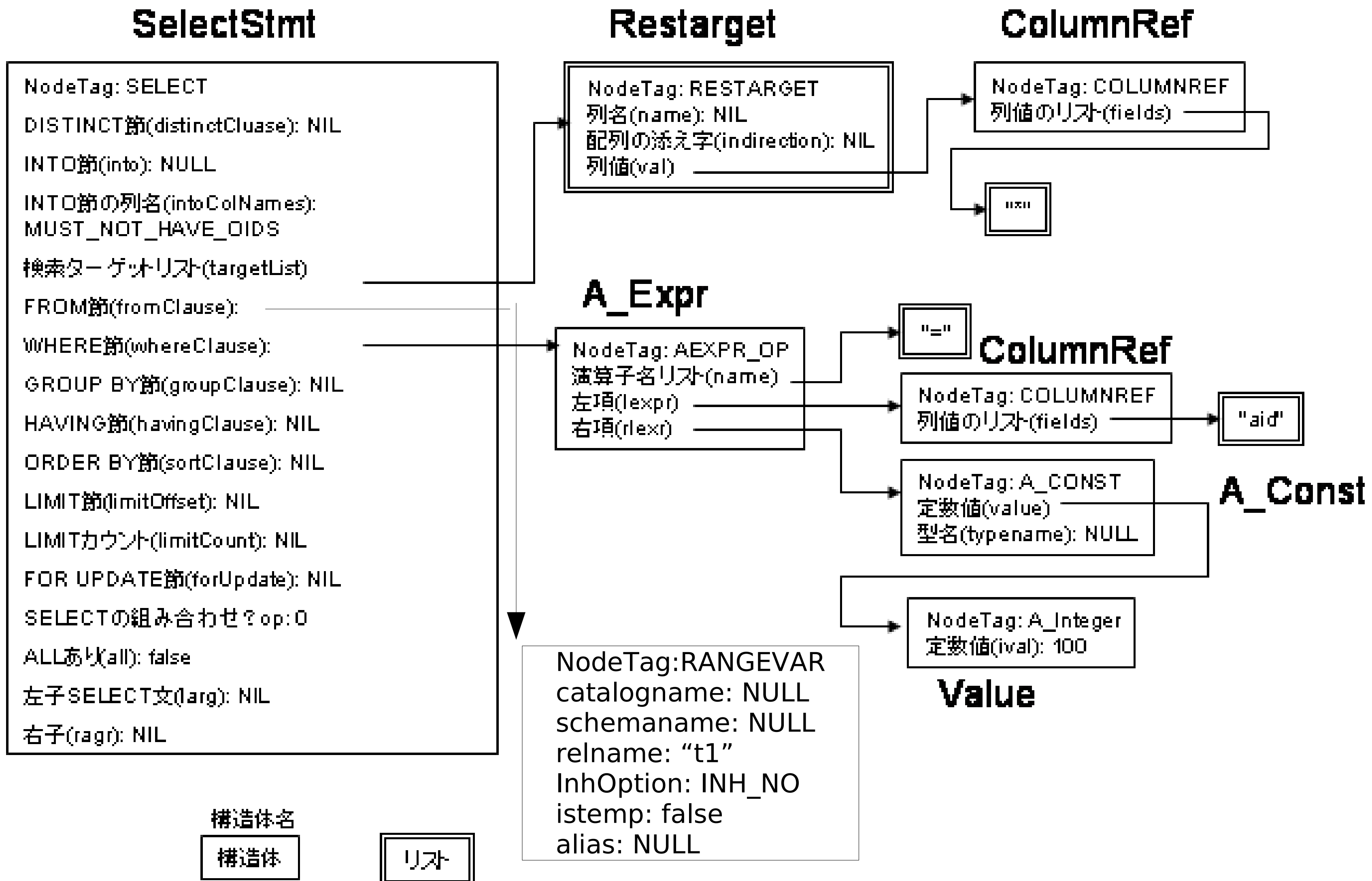


パース処理

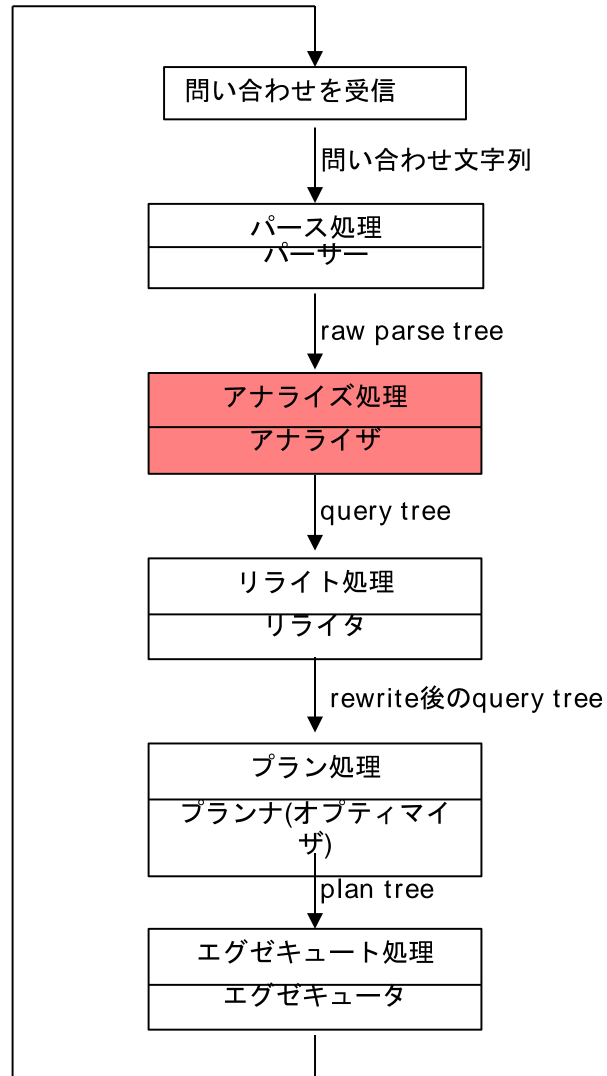
- 文字列のSQL文を解析してパースツリーを作成
- この段階ではシステムカタログは参照しない
- 字句解析
  - flexを利用
  - 数字は[0-9],整数は数字が1個以上つらなったもの,などの定義を列挙
- 構文解析
  - bisonを利用
  - SQL文の構造を再帰的に定義

paraser/gram.y

SELECT * FROM accounts WHERE aid=100のパースツリー



問い合わせ処理の流れ



アナライズ処理

- パースツリーからクエリーツリーを作成
- ステムカタログを参照してオブジェクトを決定,情報を補足
  - テーブルOID
  - 列名リスト
  - 型OID
  - オペレータOID

parser/analyze.c

Query

NodeTag: QUERY
SQLの種類(commandType): SELECT
問い合わせの出所(querySource): 0(オリジナル)
コマンドタグ設定可能？(cansetTag): true
ユーティリティコマンド(utilityStmt): NULL
rtableへのインデックス(resultRelation): 0
SELECT INTOのターゲット(into): NULL
集約あり？(hasAggs): false
副問い合わせあり？(hasSubLinks): false
レンジテーブルリスト(rtable):
FROM句とWHERE句(jointree):
FOR UPDATE対象のテーブルへのインデックス(rowMarks): NIL
ターゲットリスト(tagetList):
GROUP BY句リスト(groupClause): NIL
HAVING句(havingQual):
DISTINCT句リスト: NIL
ORDER BY句リスト: NIL
リミット句: NULL
オフセット数: NULL
UNION/INTERSECT/EXCEPTツリー(setOperations): NULL

RangeTableEntry

RTE種類(rtekind): RTE_RELATION
ユーザ指定別名(alias): NULL
別名リスト
継承？(inh): false
INあり？(inFromCL): false
アクセス権(requiredPerms): ACL_SELECT
アクセス権チェックユーザ(checkAsUser): 0

NodeTag: ALIAS
別名(aliasname): "accounts"
列名リスト(colnames):

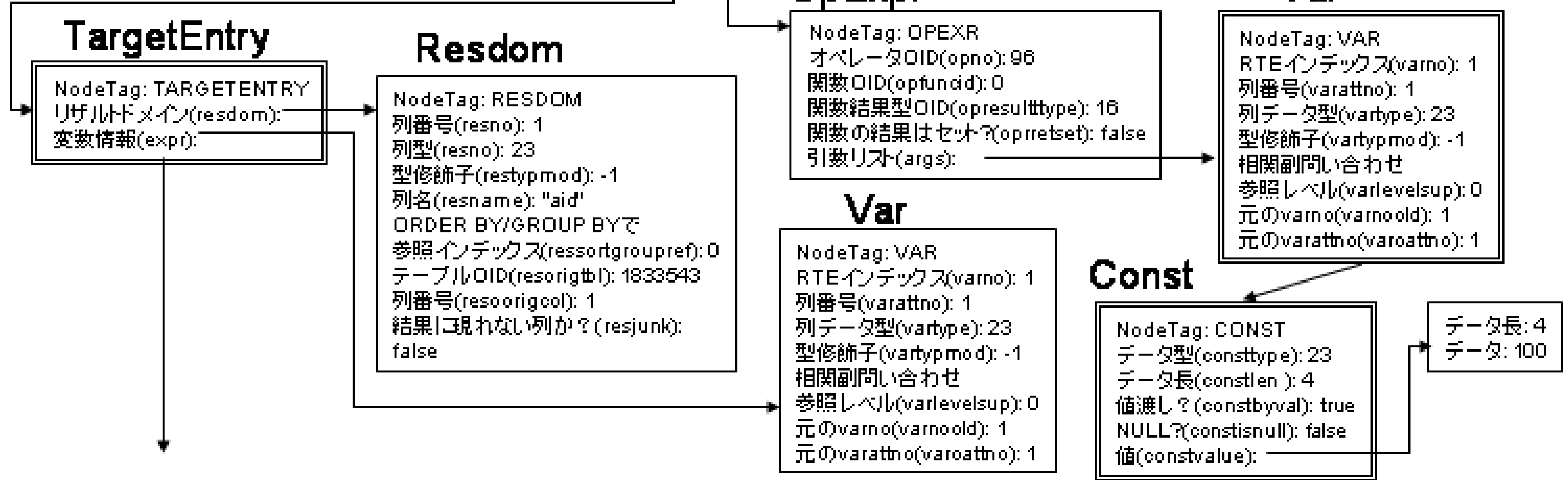
aid → bid → abalance → filler

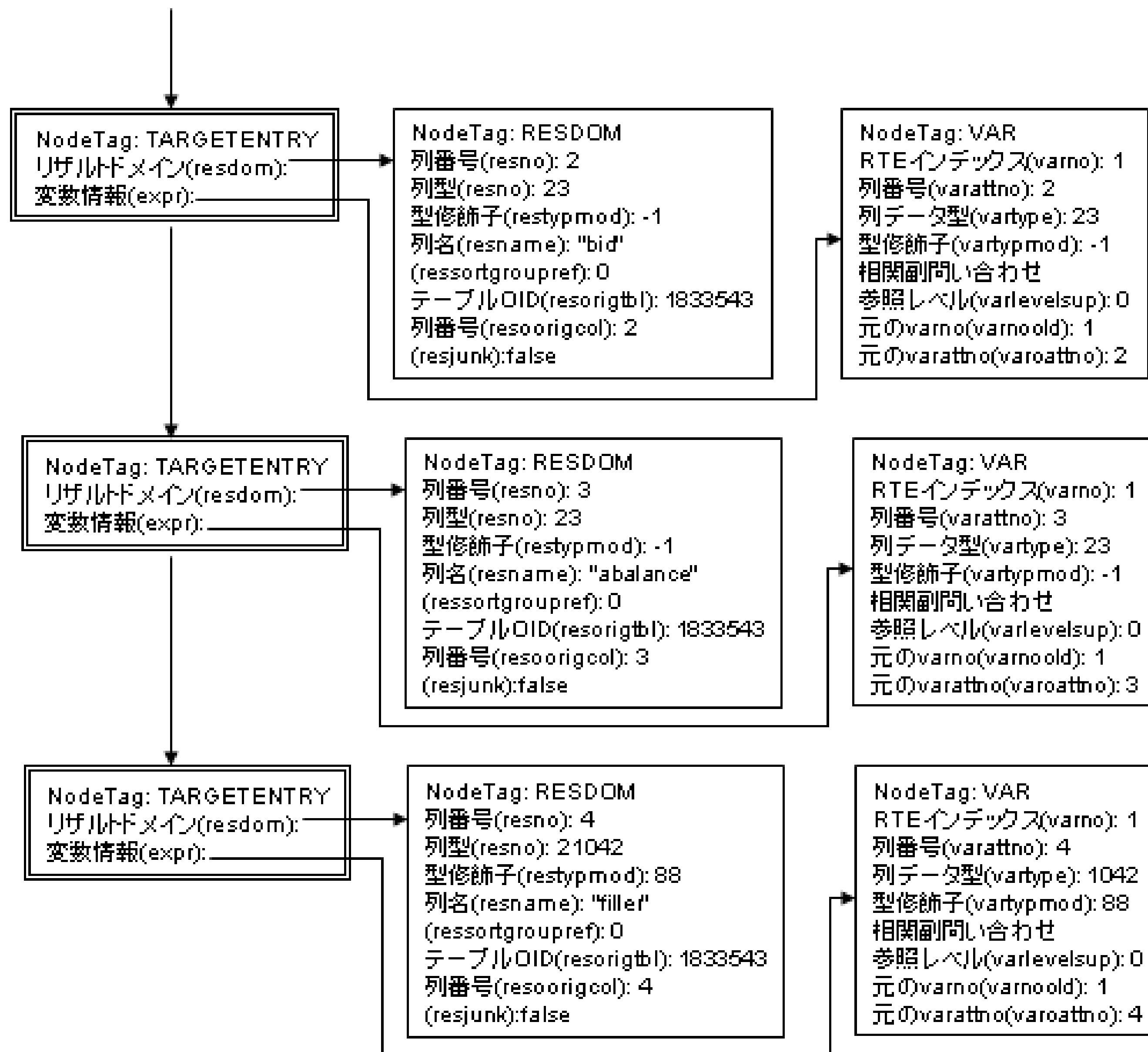
FromExpr

NodeTag: FROMEXPR
FROMリスト(fromlist):
検索条件(quals)

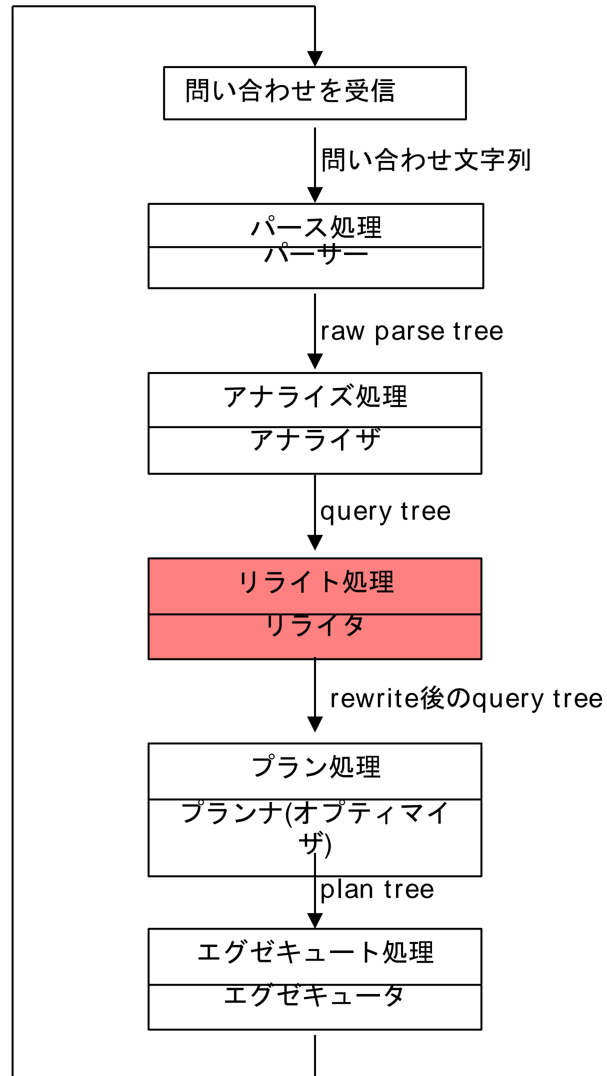
RangeTableRef

NodeTag: RANGETABLEREF
RTEインデックス(rtindex): 1




NodeTag: TARGETENTRY
リザルトドメイン(resdom):
変数情報(expr):
NodeTag: RESDOM
列番号(resno): 2
列型(resno): 23
型修飾子(restypmod): -1
列名(resname): "bid"
(ressortgroupref): 0
テーブルOID(resorigtbl): 1833543
列番号(resoorigcol): 2
(resjunk):false
NodeTag: VAR
RTEインデックス(varno): 1
列番号(varattno): 2
列データ型(vartype): 23
型修飾子(vartypmod): -1
相関副問い合わせ
参照レベル(varlevelsup): 0
元のvarno(varnoold): 1
元のvarattno(varoattno): 2
NodeTag: TARGETENTRY
リザルトドメイン(resdom):
変数情報(expr):
NodeTag: RESDOM
列番号(resno): 3
列型(resno): 23
型修飾子(restypmod): -1
列名(resname): "abalance"
(ressortgroupref): 0
テーブルOID(resorigtbl): 1833543
列番号(resoorigcol): 3
(resjunk):false
NodeTag: VAR
RTEインデックス(varno): 1
列番号(varattno): 3
列データ型(vartype): 23
型修飾子(vartypmod): -1
相関副問い合わせ
参照レベル(varlevelsup): 0
元のvarno(varnoold): 1
元のvarattno(varoattno): 3
NodeTag: TARGETENTRY
リザルトドメイン(resdom):
変数情報(expr):
NodeTag: RESDOM
列番号(resno): 4
列型(resno): 21042
型修飾子(restypmod): 88
列名(resname): "filler"
(ressortgroupref): 0
テーブルOID(resorigtbl): 1833543
列番号(resoorigcol): 4
(resjunk):false
NodeTag: VAR
RTEインデックス(varno): 1
列番号(varattno): 4
列データ型(vartype): 1042
型修飾子(vartypmod): 88
相関副問い合わせ
参照レベル(varlevelsup): 0
元のvarno(varnoold): 1
元のvarattno(varoattno): 4

問い合わせ処理の流れ

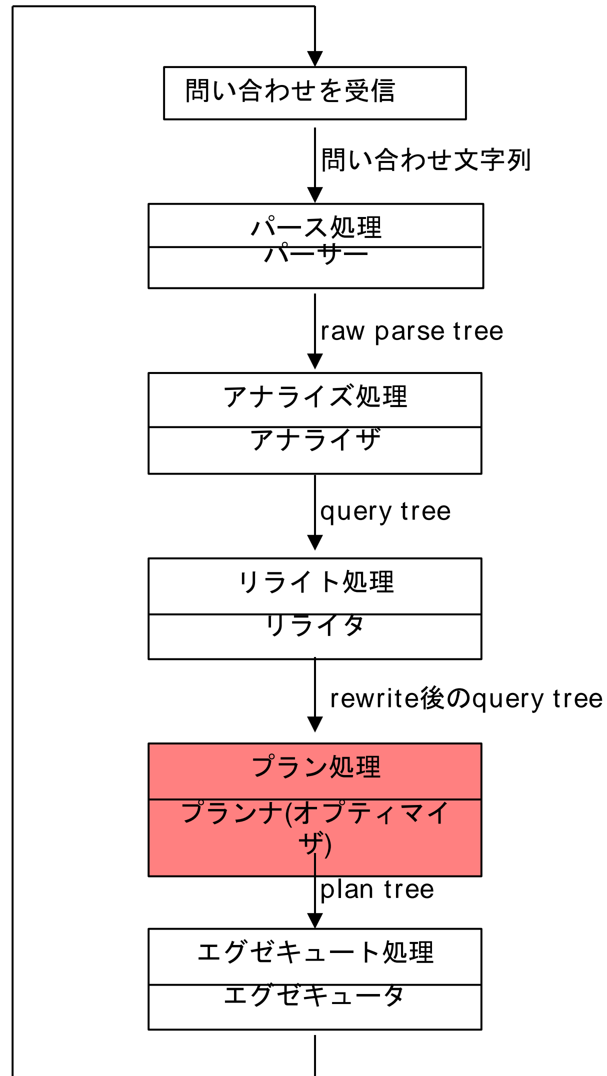


リライト処理

- クエリーツリーを一定のルールによって書き換える
- RULEシステムやVIEWの実装に利用
- RULEの例
  - CREATE RULE t1rule1 AS ON INSERT TO t1 WHERE NEW.i < 100 DO INSTEAD INSERT INTO t1 VALUES(NEW.i, NEW.j);
- RULEのパースツリーもシステムカタログに格納
- RULEパースツリーを追加,置き換え

rewrite

問い合わせ処理の流れ



プランナ

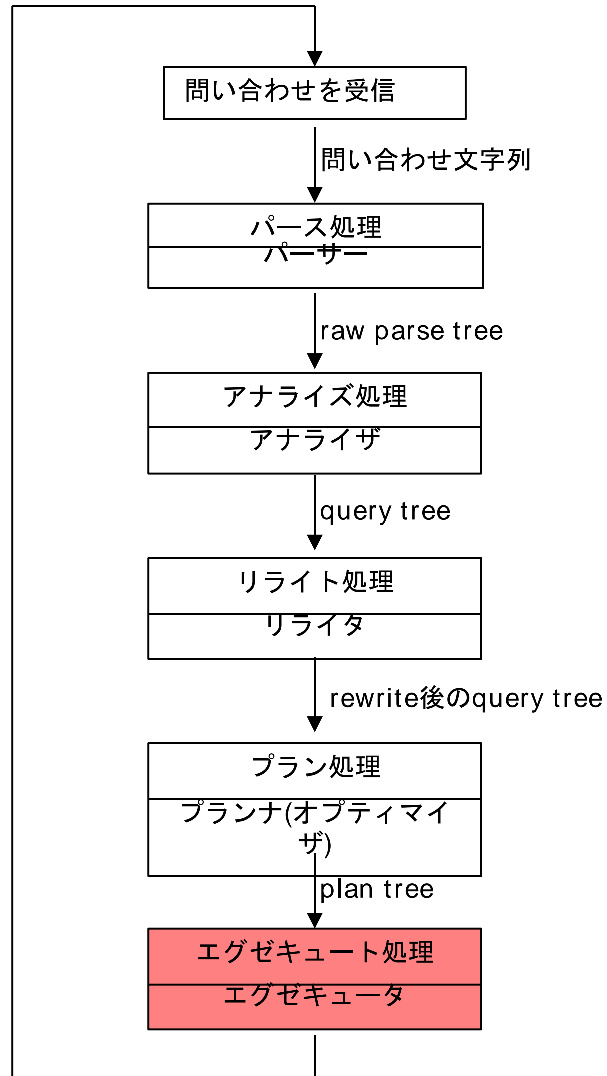
- 一般に「オプティマイザ」と呼ばれている部分
- 「ルールベース」と「コストベース」オプティマイザ
- クエリツリーからプランツリーを生成
  - 組み込みルールを適用してクエリをより効率のよいものに書き換え
  - 可能な実行方法(path)を生成
  - その中から最適なものを選ぶ
  - 選んだpathをプランに変換

optimizer/README

プランナによる書き換えの例

- INをJOINに変換
- foo OR TRUE -> TRUE
- (NOT (A < B)) -> (A >= B)

問い合わせ処理の流れ

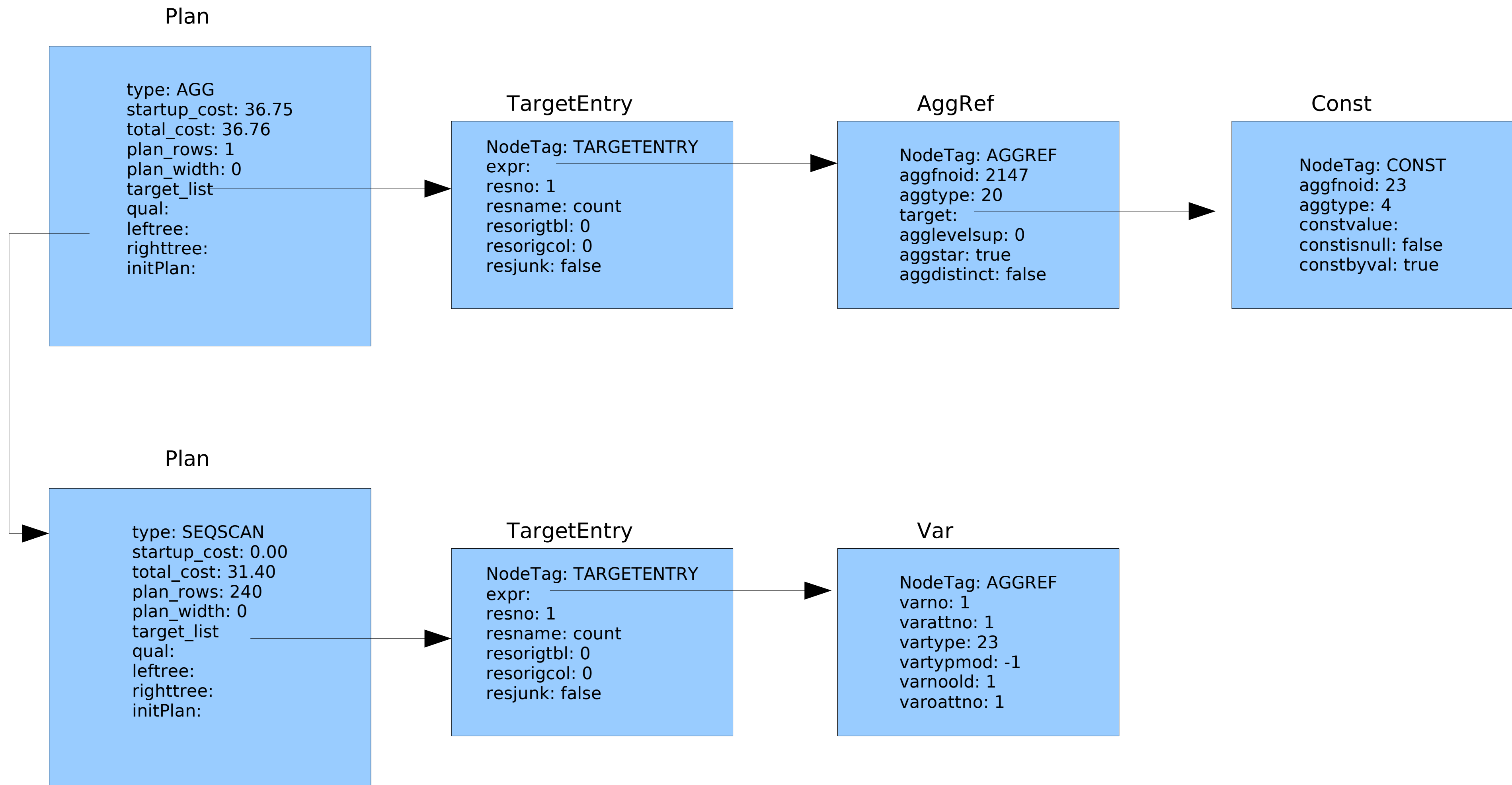


エグゼキュート処理

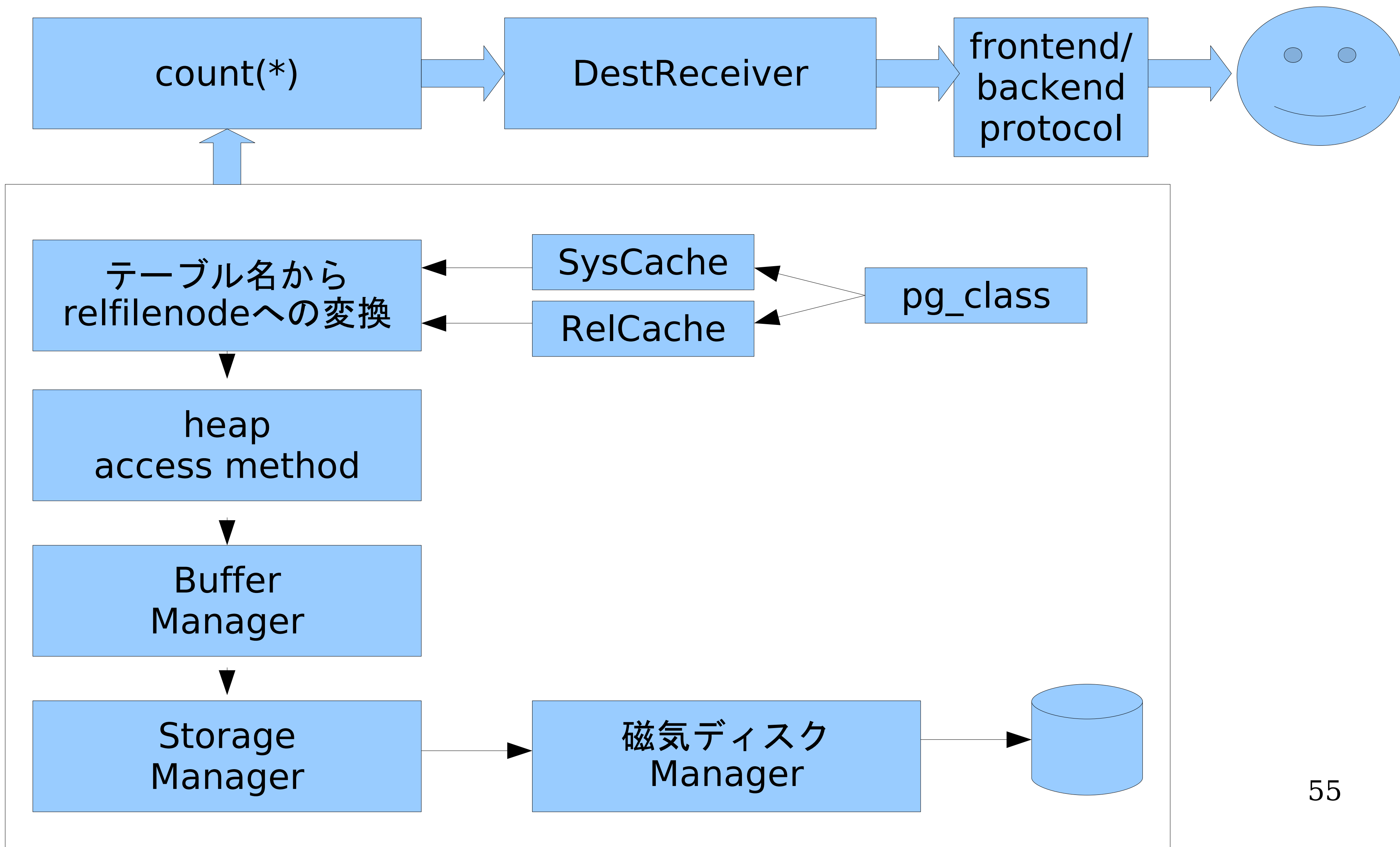
- プランナが作成したプランを実行
- 固有の情報
  - スナップショット
    - 実行中のトランザクションに関する情報
  - DestReceiver
    - 実行結果の送信先

executor/README

SELECT count(*) FROM t1のプランツリー



エグゼキュータの実行



まとめ

- PostgreSQLの概要
  - 歴史,開発体制,市場動向
  - 機能,構造
- 実装と内部構造
  - 各サブシステムの役割と構造
- 問い合わせ処理の流れ
  - パーサー,プランナ,リライト,プランナ,エグゼキュータ

参考文献/URL

- PostgreSQL完全攻略ガイド改訂第5版/石井達夫/技術評論社/2006
- WEB+DB PRESS「徒然PostgreSQL散策」
  - Vol.16「SQLをカスタマイズしよう」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol16_200-211.pdf
  - Vol.20「トランザクションログ」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol20_224-232.pdf
  - Vol.24「テーブルの構造とディスク容量の見積もり」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol24_214-221.pdf

参考文献/URL

- WEB+DB PRESS「徒然PostgreSQL散策」
  - Vol.25「テーブルの構造とディスク容量の見積もり(2)」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol25.pdf
- WEB+DB PRESS「PostgreSQL研究所」
  - Vol.27「パース処理とアナライズ処理」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol27.pdf
  - Vol.28「リライト処理」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol28.pdf
  - Vol.29「プラン処理(1)」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol29.pdf

参考文献/URL

- WEB+DB PRESS「PostgreSQL研究所」
  - Vol.30「プラン処理(2)」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol30.pdf
  - Vol.31「エグゼキュート処理」
    - http://www2b.biglobe.ne.jp/~caco/webdb-pdfs/vol31.pdf