2003.10.28 **修正版**

JPUG しくみ分科会 勉強会

PostgreSQLソースコード解析

～ DBMSアーキテクチャの概要とソースツリーの概要 ～

2003年10月27日

NTTデータ先端技術(株)
井久保 寛明

テーマ

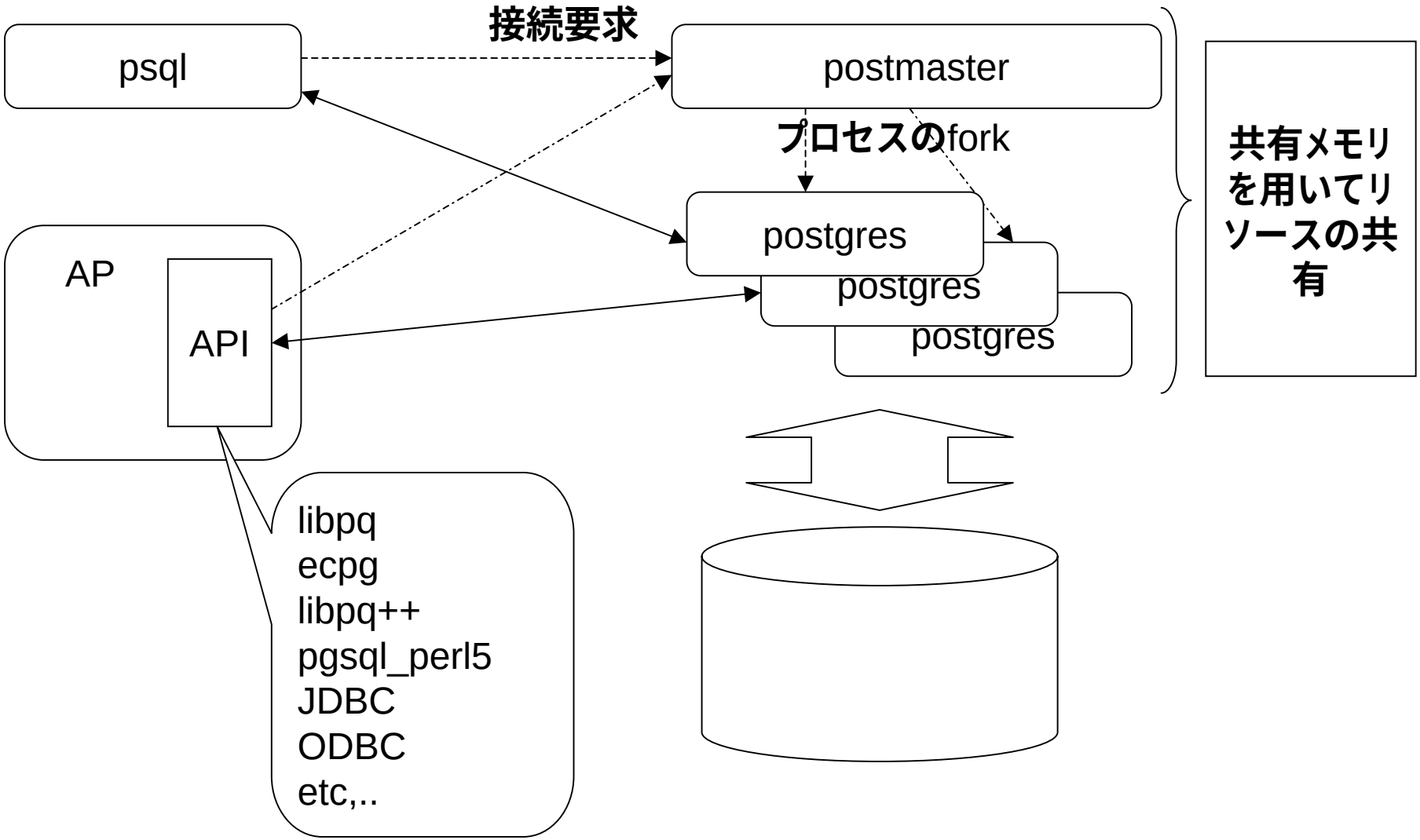
- PostgreSQLのソースコード(backend)を読むための基礎知識を学ぶ
  - DBMSの実装の概要を知る
  - ソースツリーの構造を知る
- データベースに使われている技術の雰囲気に触れる
  - 興味の持てる分野を探す

Agenda

- PostgreSQL のアーキテクチャ
  - PostgreSQLのプロセスの構造
  - DBMSエンジンのアーキテクチャ
    - ストレージ管理
    - キュエリの実行系
    - トランザクション管理
- PostgreSQLのソースツリー
  - ソースツリー直下
  - src
  - src/backend
- ソースコード流れ
  - main()
  - PostgresMain()

PostgreSQLのアーキテクチャ

PostgreSQLのプロセスの構造



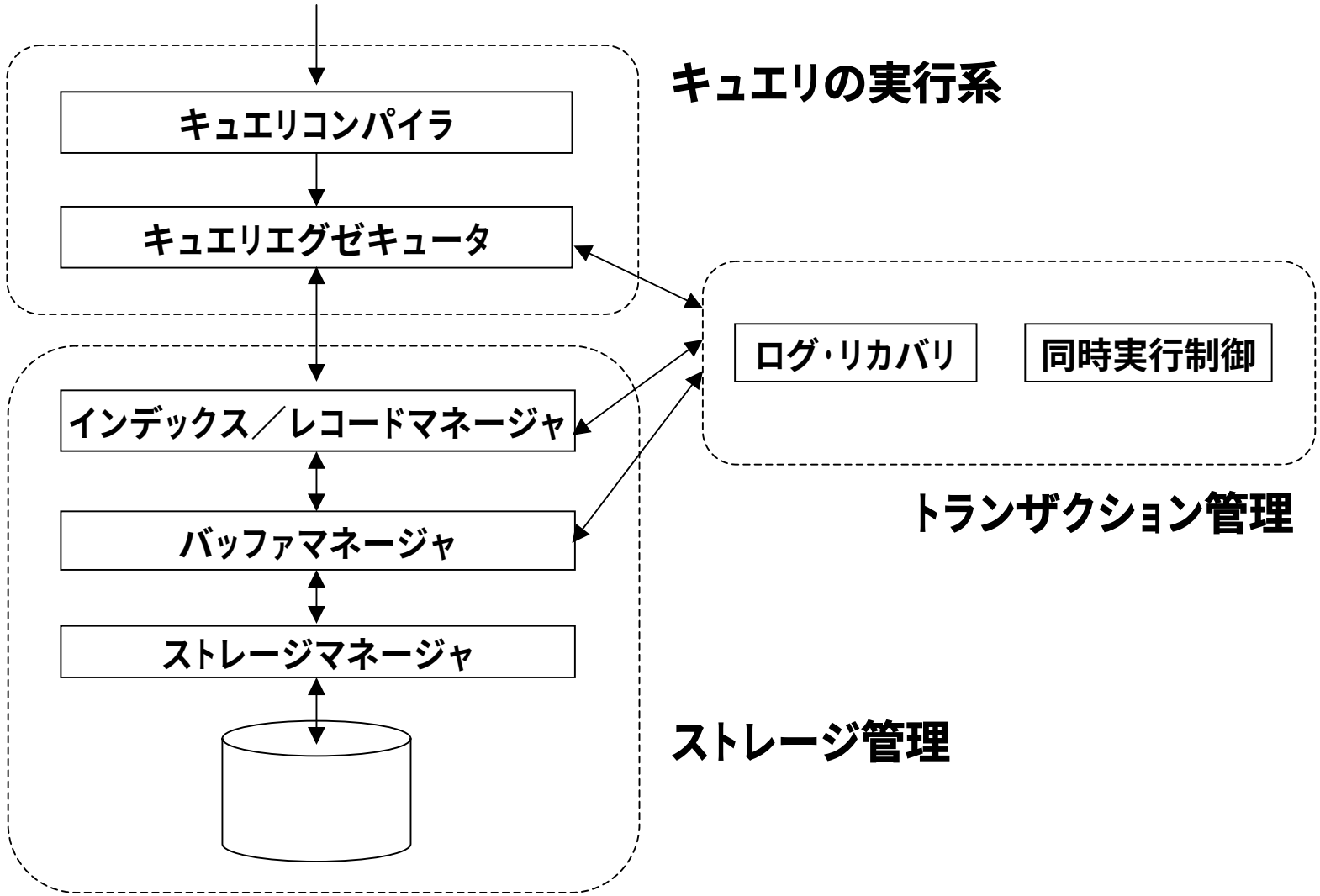
DBMSのアーキテクチャ

次の3つのパーツに分ける (※ 分け方はいろいろある)

- ストレージ管理
  - OSのファイルシステムのような技術
  - ファイル管理、I/O管理、バッファ管理
  - レコード単位のデータの供給
- キュエリの実行系
  - キュエリコンパイラ（その名のとおりコンパイラ）
  - キュエリエグゼキュータ（エミュレータのようなもの）
- トランザクション管理
  - ログ管理
  - ロック管理

一度に全部理解するのは大変です。
興味のあるところからはじめましょう。

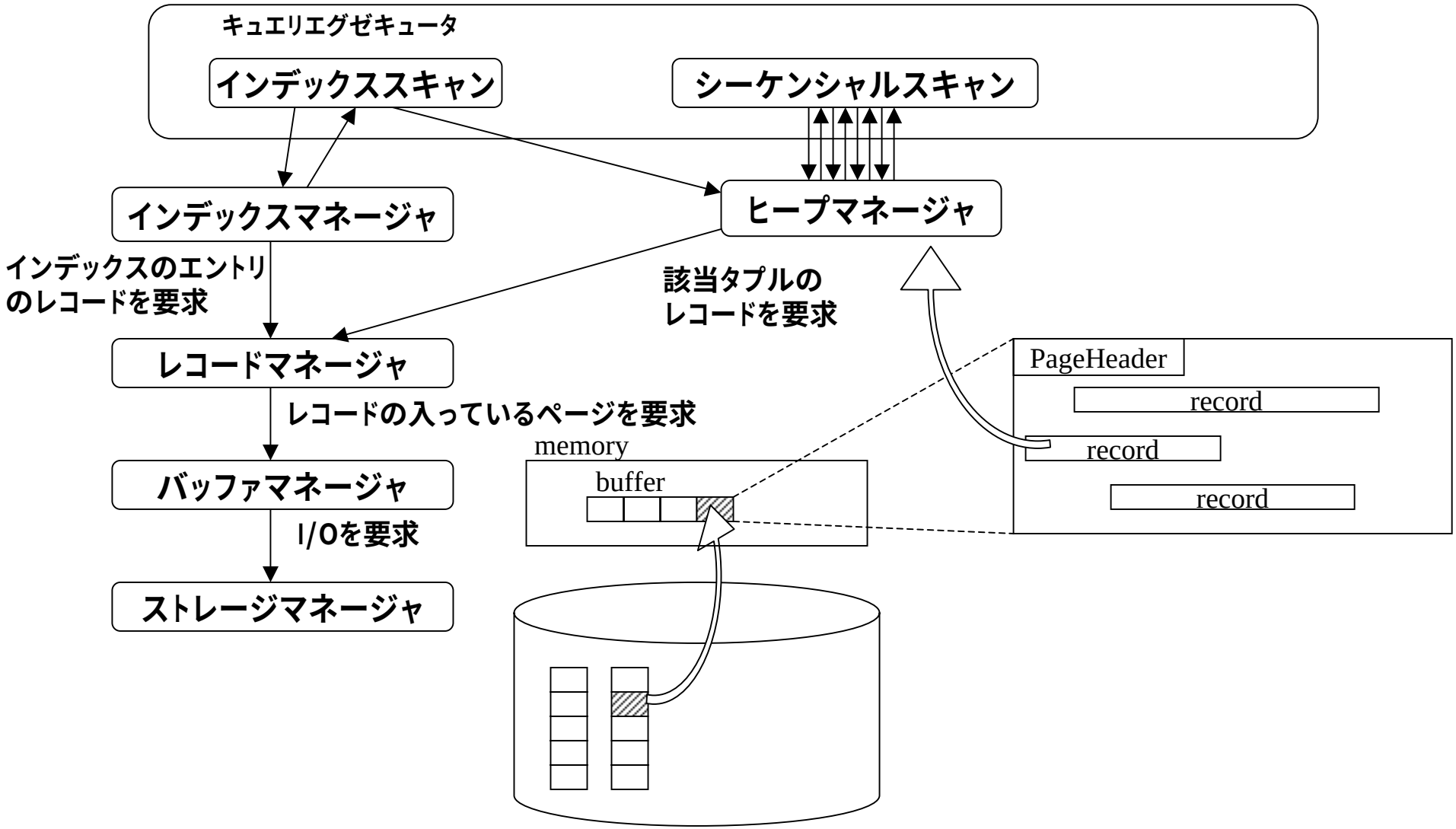
DBMSのアーキテクチャ(図)



ストレージ管理

- ストレージマネージャ
  - ディスクへのI/O の管理、OSファイルの管理など
  - OSのファイルサイズの限界やファイルデスクリプタの限界を吸収
- バッファマネージャ
  - 毎回HDDから読み出すと実行に時間かかかる（HDDは数ms、メモリは数十nsでアクセスできる）ので、HDD中の情報をキャッシュするための機構。
  - 固定長のバッファとして管理する
- レコードマネージャ（バッファ ページ マネージャ）
  - バッファのブロック内の管理を行うモジュール
  - １レコード単位の操作を行う
- ヒープマネージャ
  - 固定長のブロックをつないで、可変長の領域を提供する
- インデックスマネージャ
  - ヒープへのデータへ効率よくアクセスするための、インデックス機構の実装

ストレージ管理 (図)



キュエリの実行系

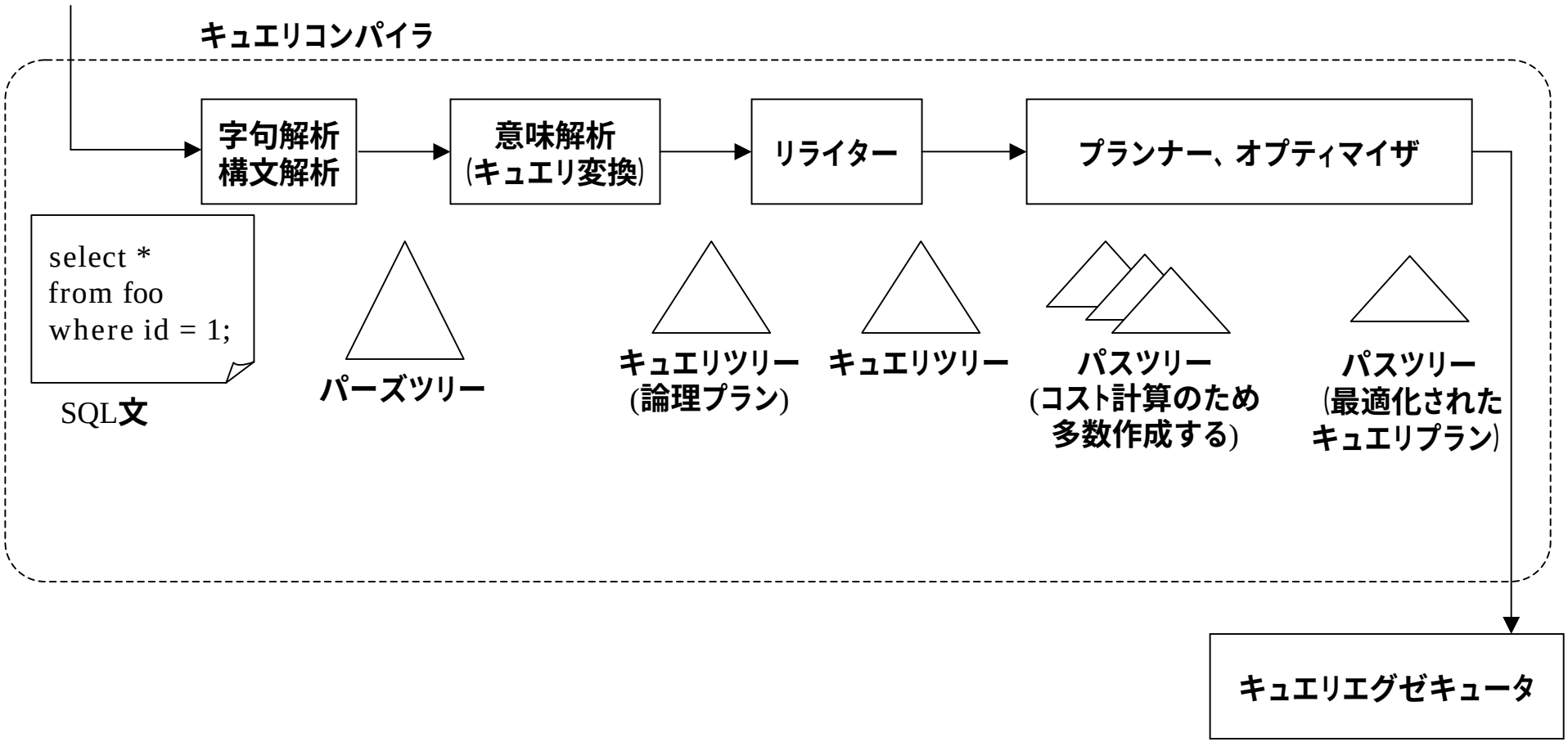
■ キュエリエグゼキュータ

- ◆ 各種のキュエリの実行アルゴリズムの実装

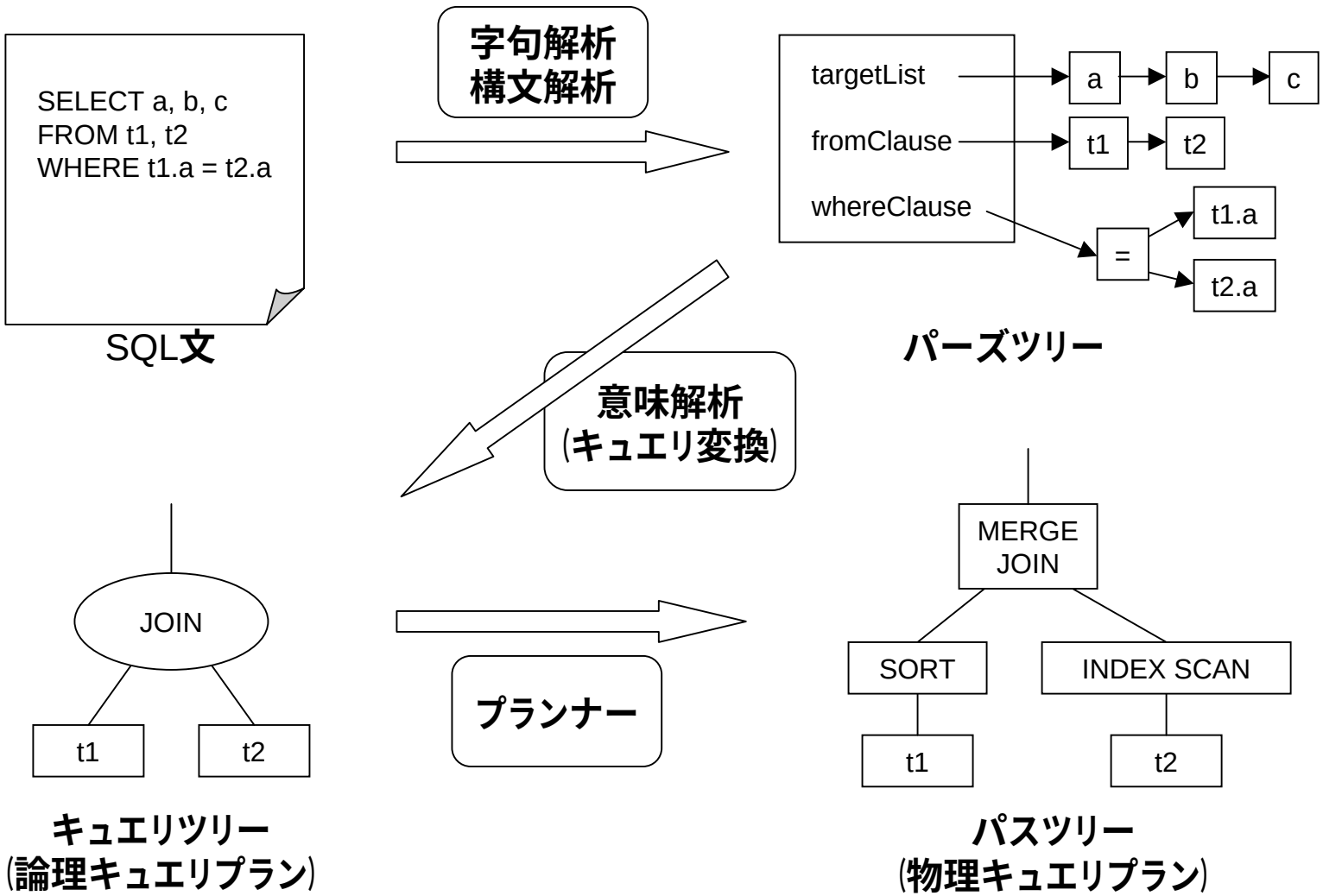
■ キュエリコンパイラ

- ◆ 字句解析
  - トークンの切り出しを行う
- ◆ 構文解析
  - パーズツリーの作成を行う
- ◆ 意味解析 (キュエリ変換)
  - パーズツリーからキュエリツリーに変換する
- ◆ リライター
  - VIEW、サブキュエリなどの書換えルールを適用する
- ◆ プランナ、オプティマイザ
  - 論理キュエリプラン (キュエリツリー) から物理キュエリプラン (パスツリー) を作成する
  - 多数のパスツリーを作成し、コストの一番小さいものを採用する
  - JOINの数が多い場合、GEQO (遺伝的アルゴリズム) を使用する

キュエリコンパイラ（1／2）



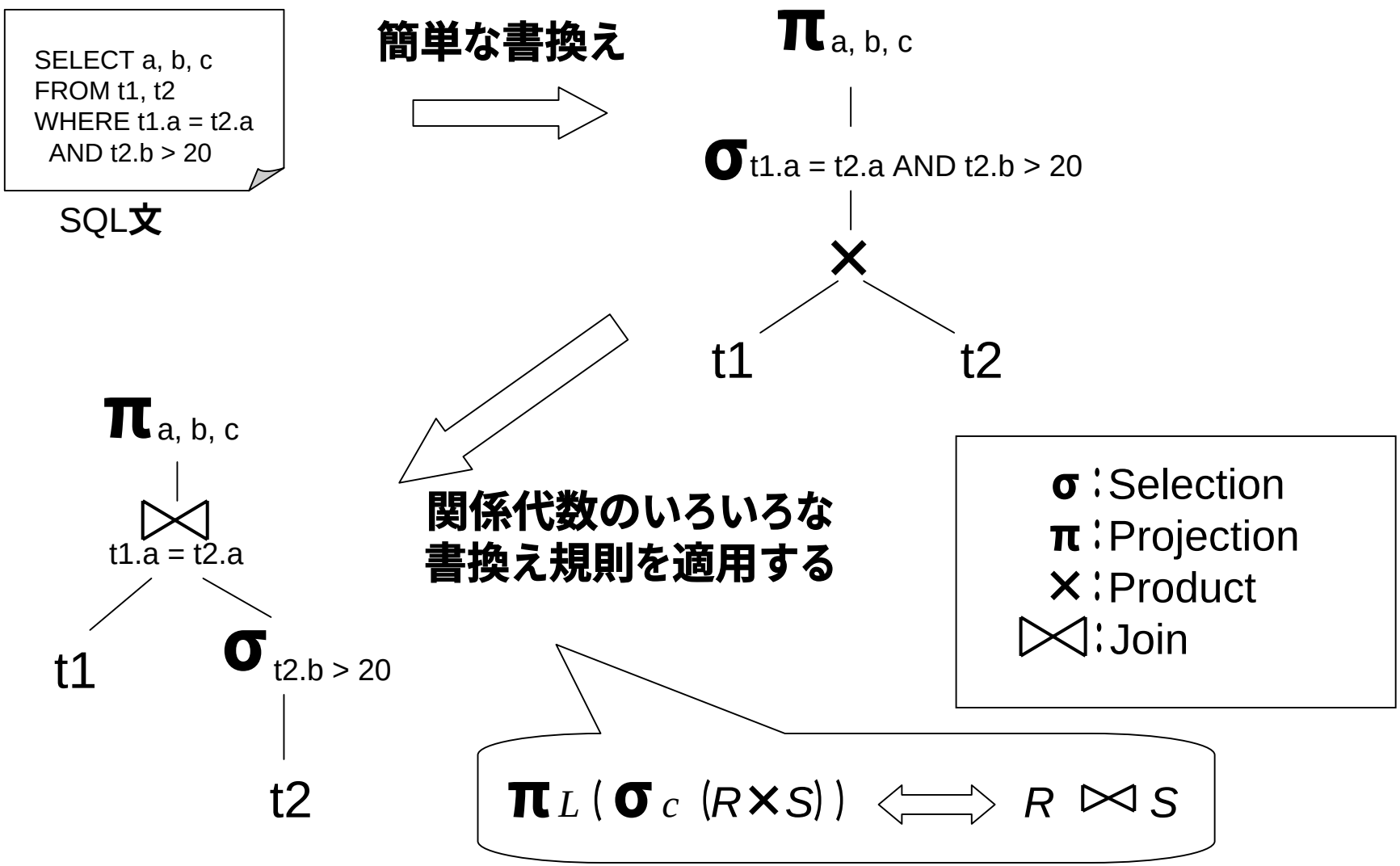
キュエリコンパイラ (2／2)



キュエリコンパイラ（関係代数とは？）

- キュエリを構築する基本的な動作は、関係代数で表せる
- 基本的な関係代数の演算子
  - Union, Intersection, Difference
    - SQLでは、UNION, INTERSECT, EXCEPT で記述される
  - Selection（選択）
    - WHERE句で書かれる条件
  - Projection（射影）
    - SELECT句で書かれている、属性のリストで表されている
  - Product（直積）
    - FROM句に書かれているテーブルのリストである
  - Join
    - Product、Projection、Selection の組み合わせでも表現できる
- 関係代数の拡張
  - Dulicate elimination （SQL の DISTINCT）
  - Grouping（SQL の GROUP BY 〜 HAVING）
  - Sorting（SQL の ORDER BY）

キュエリコンパイラ（関係代数を使うと何ができる？）



キュエリエグゼキュータ（DBMS固有のアルゴリズムとは？）

■ 通常のプログラムの世界でのアルゴリズム

- ◆ 全てのデータがメモリに乗っていることが前提
- ◆ つまり、どんなにメモリを使用しても構わないことになっている

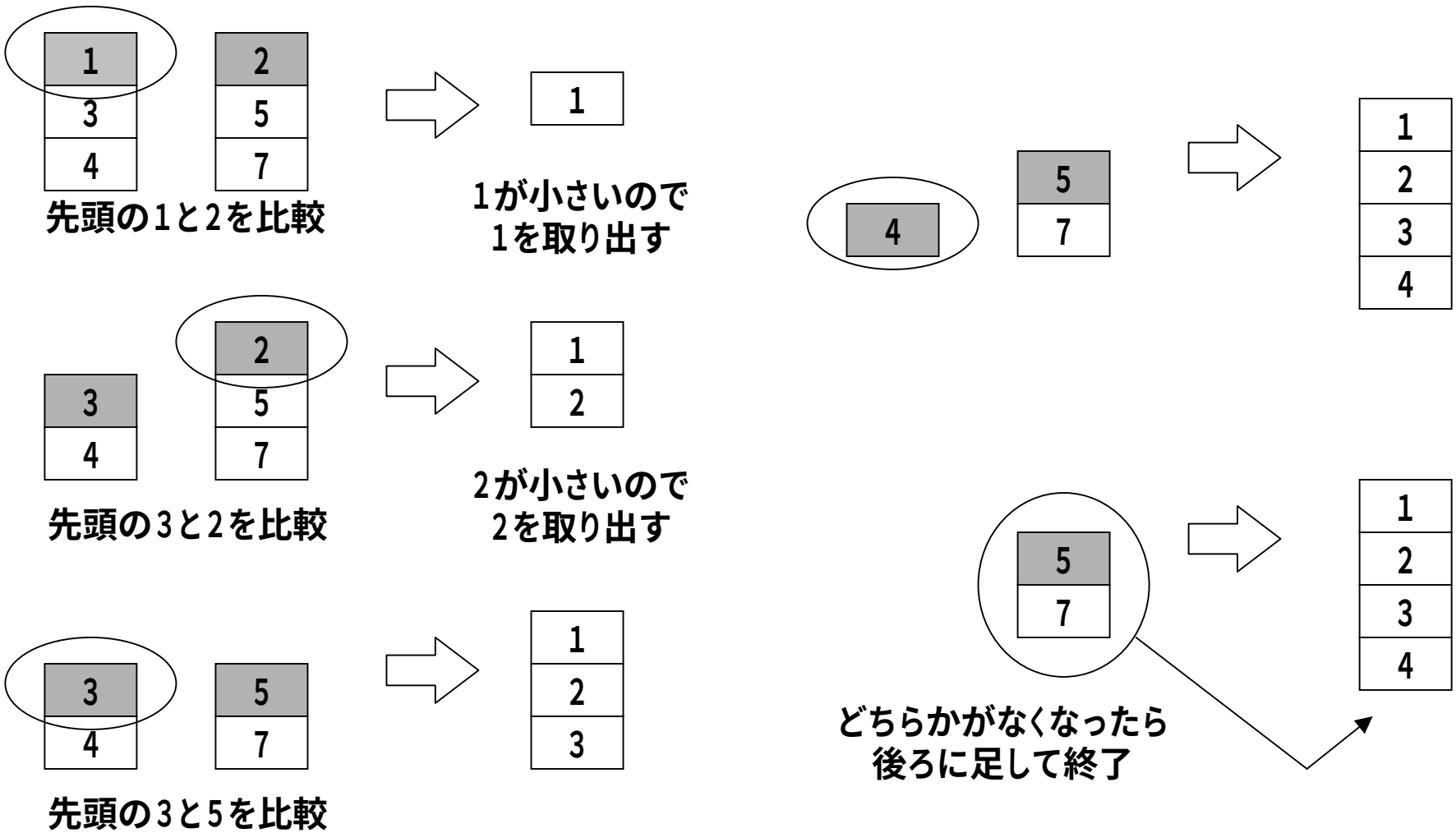
■ DBMSの世界でのアルゴリズム

- ◆ メモリとディスクでは、アクセス速度が1万～100万倍程度異なる
- ◆ データ量が多い場合、いかにディスクI/Oを減らすかということが重要になる
- ◆ アルゴリズムの基本は次の3つ、あとはその応用
  - ソートを基本にしたアルゴリズム
  - ハッシュを基本にしたアルゴリズム
  - インデックスを基本としたアルゴリズム
- ◆ メモリの上限により、1段で終わらないものは2段、3段と多段で実行する

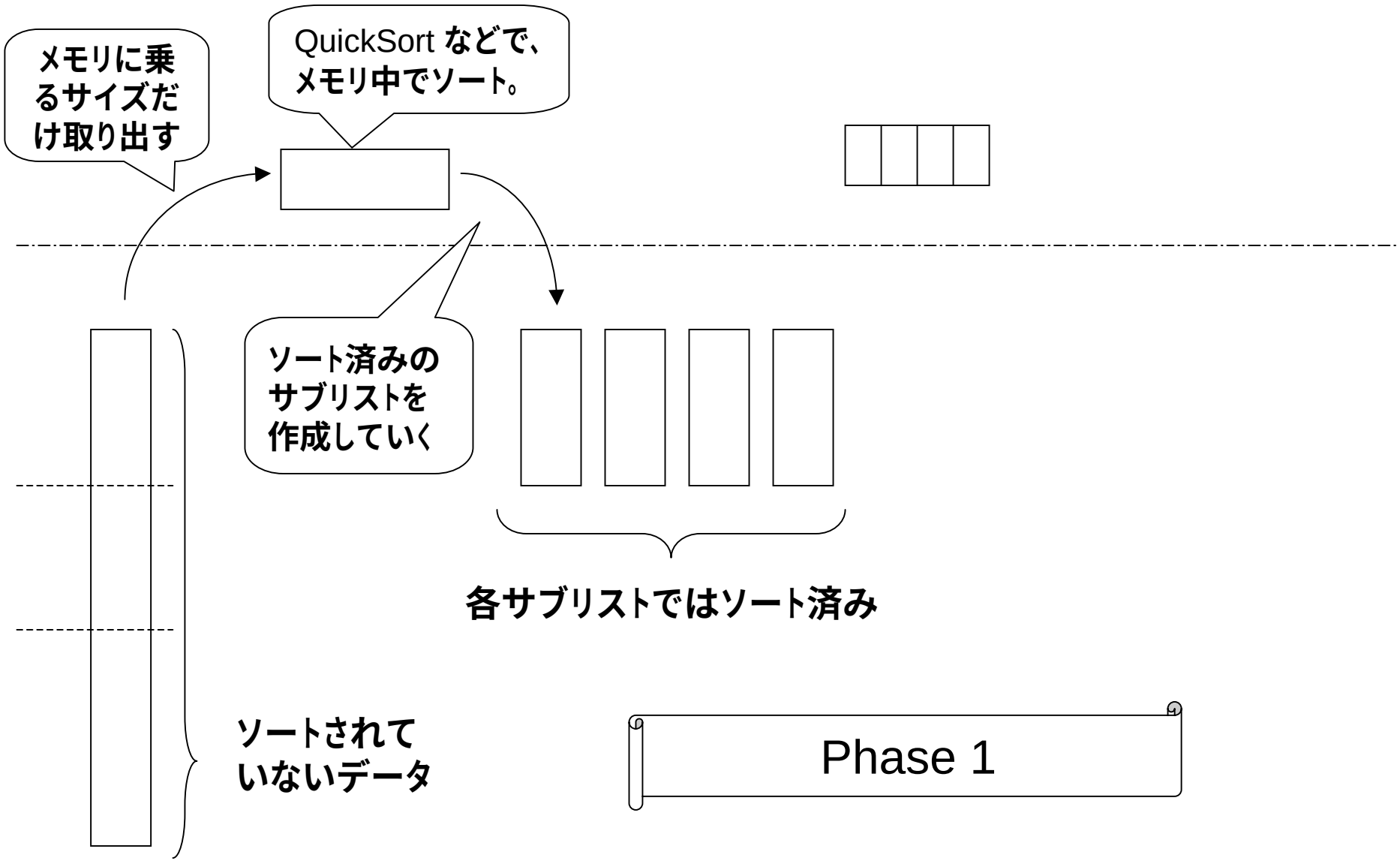
■ Merge Sort（例）

- ◆ ソート済みの2つのリストから新しいソート結果を得るアルゴリズム

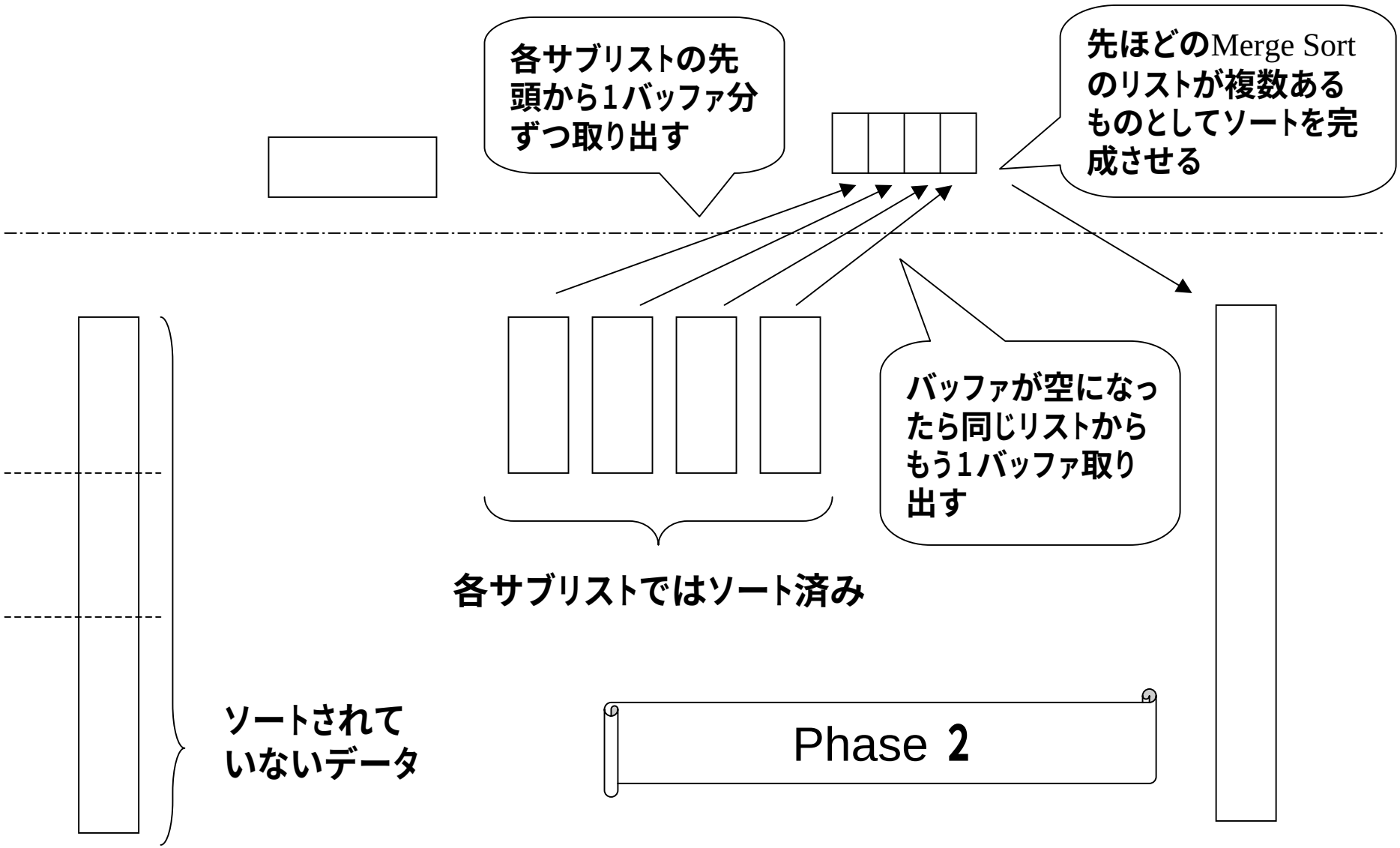
キュエリエグゼキュータ（Merge Sort のアルゴリズム）



キュエリエグゼキュータ（2Phase Multi-way Merge Sort) (1/3)



キュエリエグゼキュータ（2Phase Multi-way Merge Sort) (2/3)



キュエリエグゼキュータ（2Phase Multi-way Merge Sort) (3/3)

■ Merge Sort

- ◆ **利用できるバッファのサイズをMとした場合、M×Mのサイズのテーブルまで、2段でソートできる**
- ◆ **M×M のサブリストを作っていくことで、3段目の**Merge Sort **が可能**（M×M×Mまでソート可能)
- ◆ **そのまま拡張して、多段のソートが可能**

- ◆ **2Phase目のソート済みのサブリストの先頭から1バッファずつとりだして、処理を行うところが肝！**
- ◆ **サブリストが別のテーブルになっているものが、**Merge JOIN（**正確には、**Merge Sort Join) **のアルゴリズムである**

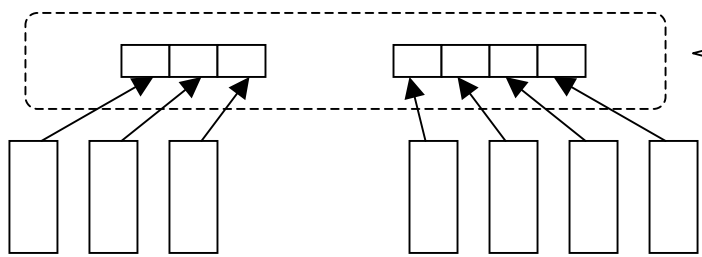


Table1 のソート済みサブリスト　　**Table2 のソート済みサブリスト**

それぞれのソート結果をディスクに書き込まず、Merge Sort の 2Phase 目のところで、2つのテーブルを一気にJOIN すると高速なMerge Joinになる。

キュエリエグゼキュータ（PostgreSQLのエグゼキュータ)

- JOIN
  - Nestloop JOIN
  - Merge JOIN
  - Hash JOIN
- SCAN
  - Sequential SCAN
  - Index SCAN
  - Subquery SCAN
  - Function SCAN
  - Tid SCAN
- **その他の物理演算子**
  - Hash, Sort
  - Aggregate, Unique, Group, Limit, SetOp
  - Result, Material
  - Append, Subplan

executor **の下の** nodeXXX.c**ファイルより想像できる物理演算子（まだ調べてないです)**

トランザクション管理（ACID特性）

トランザクション管理は、ACID特性を満たすための機構

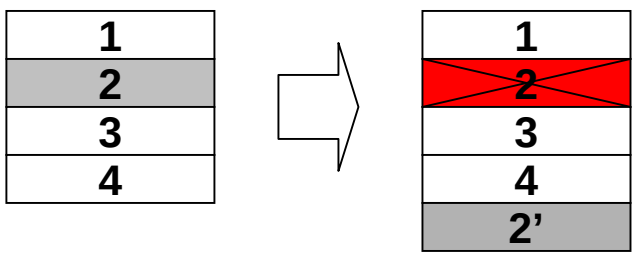
■ ACID特性

- ◆ Atomicity（原子性）
  - トランザクションは、全て実行が完了するか、または全く実行されていない状態でなければならない。
  - ログをとることで実現する。トランザクションが途中で止まった場合、更新前のデータを書き戻すことでアボート処理を行い、全く実行されていない状態にする。
- ◆ Consistency（一貫性）… △ここでいうトランザクション管理ではあまり関係がない
  - データベースのインテグリティ制約が保たれる。おおざっぱにいうと、意味的に正しい状態が保てるということ。
- ◆ Isolation（独立性）
  - 他のトランザクションと干渉してはならない。つまり、同時実行している他のトランザクションによって、結果が変わってはならないということ。
  - 実際は、Transaction Isolation Level の設定によって、ユーザがトランザクションの干渉の度合いを選択できる
  - ロックによって実現されている
- ◆ Durability（永続性）
  - 一度コミットした変更は、永遠に失われてはならない。

トランザクション管理 (WAL と MVCC)

- WAL（Write Ahead Logging）
  - データをディスクに書き込む際に、必ずデータバッファより先に、ログを書き込む方式のこと
  - これによって、データの保証を行う

- MVCC (Multi Version Concurrency Control)
  - 追記型DBMSの特性を利用して、同時実行性をあげるための仕掛け



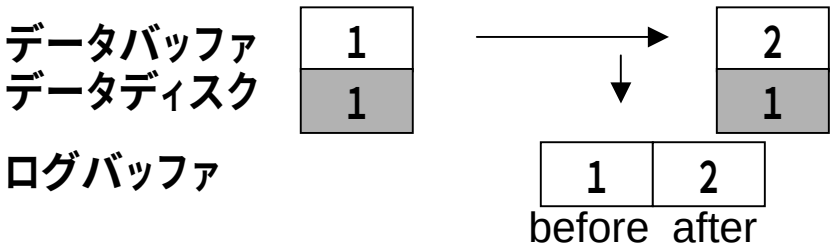
追記型DBMSとは、左のように2を2'に更新するときに、元の領域ではなく、新しい領域に更新データを書き込む方式である。

追記型でない場合、2の領域はコミットされるまでロックされている。**MVCC**とは、トランザクションの順序性を保てる場合は、元の2の領域を別のトランザクションで読ませる技術である。

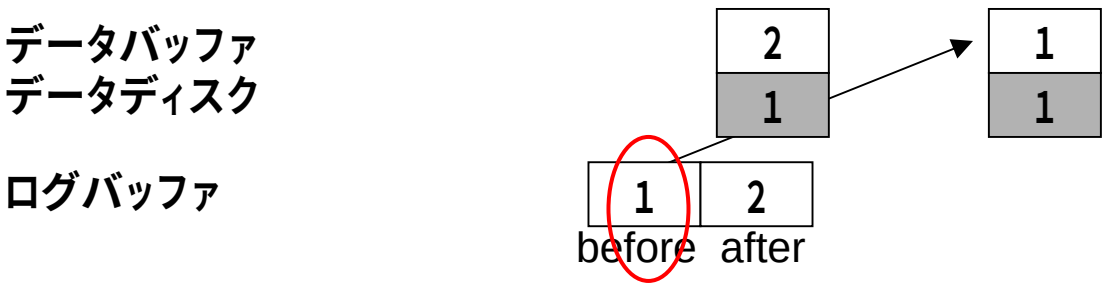
トランザクション管理 (ログとリカバリ)

※ PostgreSQLは追記型DBMSなので、更新前のデータはログに書かなくてよい

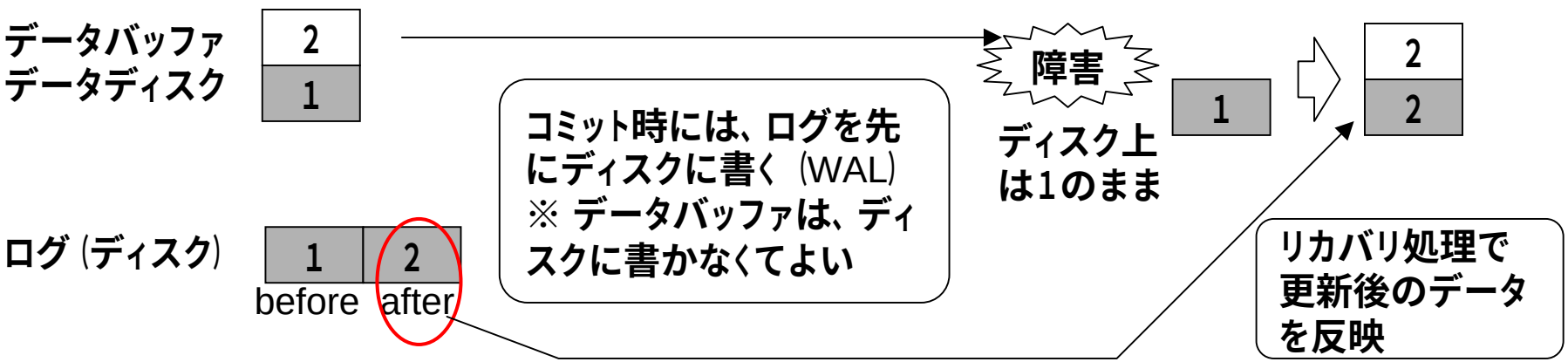
■ 一般的な物理ロギングの例



データバッファを更新する際は、
更新前、更新後のデータをログに書く



アボート時は、更新前のデータを書き戻す。



コミット時には、ログを先にディスクに書く (WAL)
※ データバッファは、ディスクに書かなくてよい

リカバリ処理で更新後のデータを反映

PostgreSQLのソースツリー

※ PostgreSQL 7.3.4をベースにしています。

トップディレクトリ

```
frisk(4)% ls -F
COPYRIGHT       INSTALL     config/         configure.in  src/
GNUmakefile     Makefile    config.log      contrib/
GNUmakefile.in  README      config.status*  doc/
HISTORY         aclocal.m4  configure*      register.txt
frisk(5)%
```

※ 下線付きは、configure 実行後に作成されるファイル

- src 以下にソースコードが入っている
- contrib 以下に、様々な人から提供されたプログラムが入っている
- HISTORY に、Postgres95 からの改変履歴が全て書かれている
- ドキュメント類（大文字のファイル名）
  - COPYRIGHT, HISTORY, INSTALL, README, register.txt, docディレクトリ以下
- configre, コンパイルに必要なファイル
  - GNUmake, GNUmake.in, Makefile, configure, configure.in, config ディレクトリ

src ディレクトリ (1/2)

DEVELOPERS	FAQ が pgsql/doc/FAQ_DEV にあり、開発支援ツールが pgsql/src/tools にあるというメモ
Makefile	各サブディレクトリに対して、make の実行を引き継がせる定義が書いてあるだけ。
Makefile.global	make の suffix などのルールを書いてあるファイル。Makefile などから読み込まれる。configure 実行時に Makefile.global.in から作成される
Makefile.global.in	Makefile.global の元ファイル。
Makefile.port	プラットフォームごとの定義の書かれた makefiles 以下のファイルへのシンボリックリンク。Makefile.global から include される。configure 実行時に作成される。
Makefile.shlib	Shared ライブラリ作成のルールが書かれた Makefile。Shared ライブラリが使えるプラットフォームの Makefile で include される。
nls-global.mk	NLS のルールが定義されているファイル。Makefile.global から include される。
win32.mak	Windows 用の makefile。ココには、ODBC などでクライアントとしてアクセスするのに必要な部分しか含まれていない。

このディレクトリにあるファイルは、基本的に Makefile 関連

src ディレクトリ (2/2)

backend/	PostgreSQL のバックエンドの本体が入っているディレクトリ
bin/	インストール後の bin ディレクトリ以下に納められるツール類のスクリプトやソースを格納したディレクトリ。
corba/	CORBA I/F
data/	charset の変換テーブルのサンプル (日本語では使えない？)
include/	C でクライアントを書く際に必要な include ファイルを集めたディレクトリ
interfaces/	各種 API の実装。ecpg, jdbc, libpgtcl, libpq, libpq++, odbc, perl5, python などが実装されている。
makefiles/	Makefile 中の各プラットフォームの固有の定義。Makefile.port としてシンボリックリンクを張って、Makefile.global に読み込まれる。
pl/	Stored Procedure 用の言語の実装。
template/	各種プラットフォーム用のコンパイルの引数のテンプレート
test/	コンパイル後のテストのためのコード類
tools/	PostgreSQL 開発者向けのツール
tutorial/	チュートリアルとしてのサンプルコード
utils/	ユーティリティ。現在、DLL 初期化ユーティリティのみ。

サーバの実装を見る場合、主に backend, include ディレクトリ以下を見ていくことになる。
クライアントは、主に interfaces ディレクトリ以下になる。

src/backend ディレクトリ (1/3)

Makefile		
1 access/		
	1.1 common/	タプル操作、タプルのヘッダ操作の補助関数などが書いてある。
	1.2 gist/	GiST index(多次元インデックス)の実装。
	1.3 hash/	ハッシュインデックスの実装。
	1.4 heap/	heap の実装。catalog 配下の heap.c から呼ばれる。こちらが実ファイルアクセスに近い側。
	1.5 index/	index のインタフェース部分の実装。
	1.6 nbtree/	Btree インデックスの実装。
	1.7 rtree/	2次元インデックス R-tree の実装。
	1.8 transam/	トランザクションマネージャ(ログ管理部分)
2 bootstrap/		初期データベース構築時のモジュール
3 catalog/		カタログの実装
4 commands/		エグゼキュータのうち、コマンド(DML 以外の処理)部分の実装
5 executor/		エグゼキュータのうち、キュエリエグゼキュータの実装
6 lib/		内部で使用するユーティリティ的なライブラリ。
7 libpq/		サーバ側の libpq プロトコルの実装。主にユーザ認証など
8 main/		postmaster, postgres などのプログラムの main() 関数の実装
9 nodes/		SQL コンパイラで使用するノードオブジェクトの実装

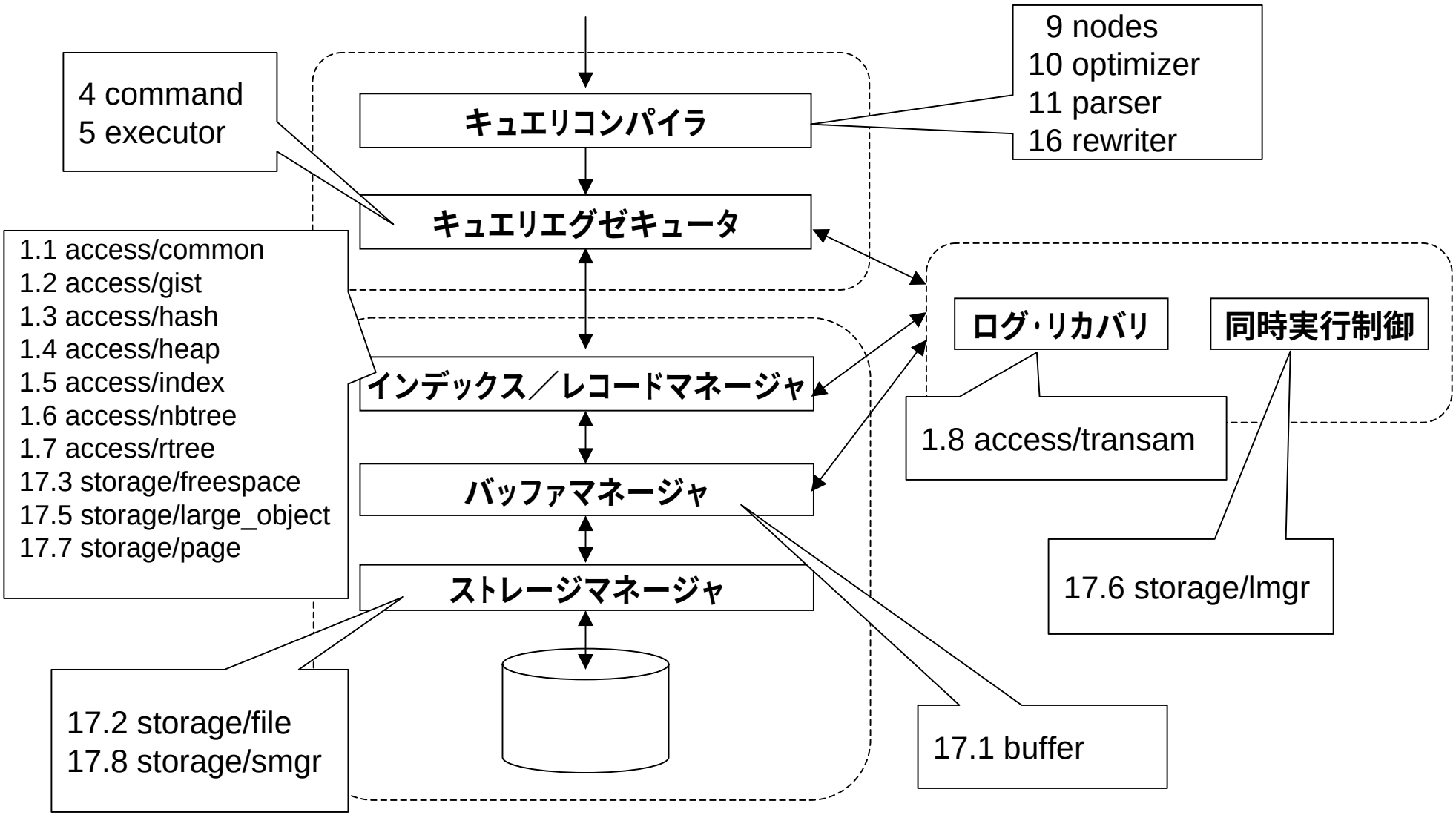
src/backend ディレクトリ (2/3)

10 optimizer/		SQL コンパイラのうち、オプティマイザの実装
	10.1 geqo/	Genetic Query Optimization (遺伝的キュエリ最適化)部分の実装
	10.2 path/	JOIN するときに、可能な全ての順序のパターンを生成する
	10.3 plan/	キュエリプランの生成を行う
	10.4 prep/	特殊なケースに対するプリプロセスフェーズの実装
	10.5 util/	オプティマイザ用の補助関数などの実装
11 parser/		キュエリコンパイラのうちスキャナ、パーザ、意味解析部分の実装
12 po/		メッセージカタログのようなもの。全体としてあまり使われていない？
13 port/		各プラットフォームの移植用コード
14 postmaster/		Postmaster の実装
15 regex/		正規表現関連の実装
16 rewrite/		SQL コンパイラのうち、リライターの実装
17 storage/		ストレージ全般を管理するモジュールを集めたディレクトリ
	17.1 buffer/	バッファ管理。ここで管理するのは、データ部分及びインデックスに使用するバッファだけであり、ログ用のバッファは、別に用意される。
	17.2 file/	仮想ファイルデスクリプタの実装。
	17.3 freespace/	ページ内 (テーブル内) の空スペースを管理する。
	17.4 ipc/	共有メモリの管理
	17.5 large_object/	blob など lo の実装。

src/backend ディレクトリ (3/3)

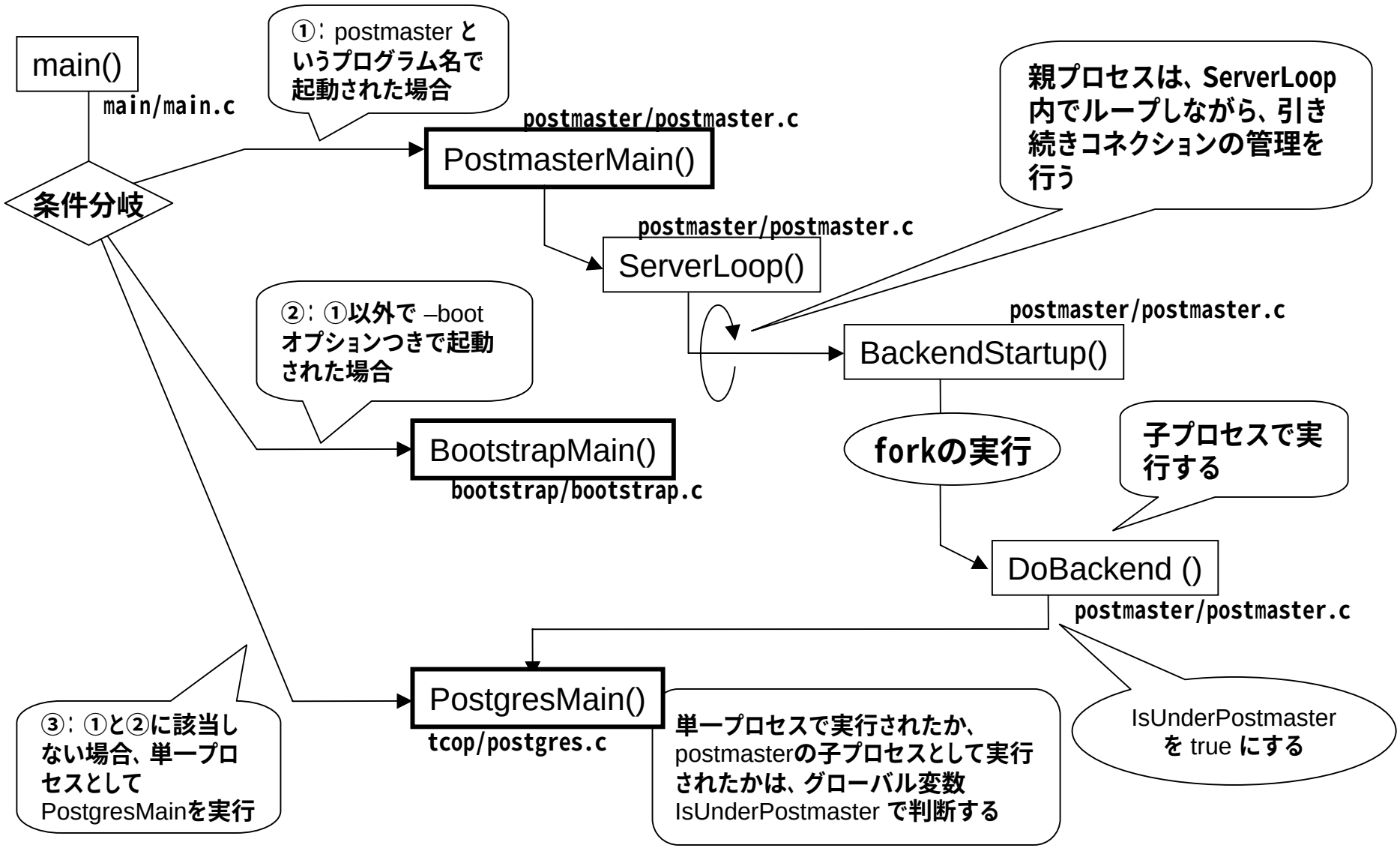
	17.6 lmgr/	ロックマネージャ。
	17.7 page/	ページ中のデータ管理モジュール。レコード単位のデータの取り出しを実装。
	17.8 smgr/	実ストレージへの I/O を実装したモジュール。ディスクへのシステムコールなどを管理。
18 tcop/		PostgresMain などの実装が入っているところ。
19 utils/		
	19.1 adt/	データ型に関する実装。
	19.2 cache/	カタログデータ、リレーション、ファンクションのキャッシュを担当するモジュール。
	19.3 error/	エラーメッセージ出力に関する実装
	19.4 fmgr/	ファンクションマネージャ: ファンクションの実装。
	19.5 hash/	ハッシュ関数の実装
	19.6 init/	postgres または postmaster 起動時の各種初期化モジュール
	19.7 mb/	マルチバイト処理に関するモジュール
	19.8 misc/	実行時のパラメータを扱う GUC やその他汎用モジュールがある場所。
	19.9 mmgr/	メモリマネージャ。階層的にメモリを管理できるモジュールを提供。
	19.10 sort/	ソートの実装。
	19.11 time/	MVCC (タイムスタンプ型の同時実行制御)の実装

DBMSのアーキテクチャ（図） --- ソースコードとの対応 ---

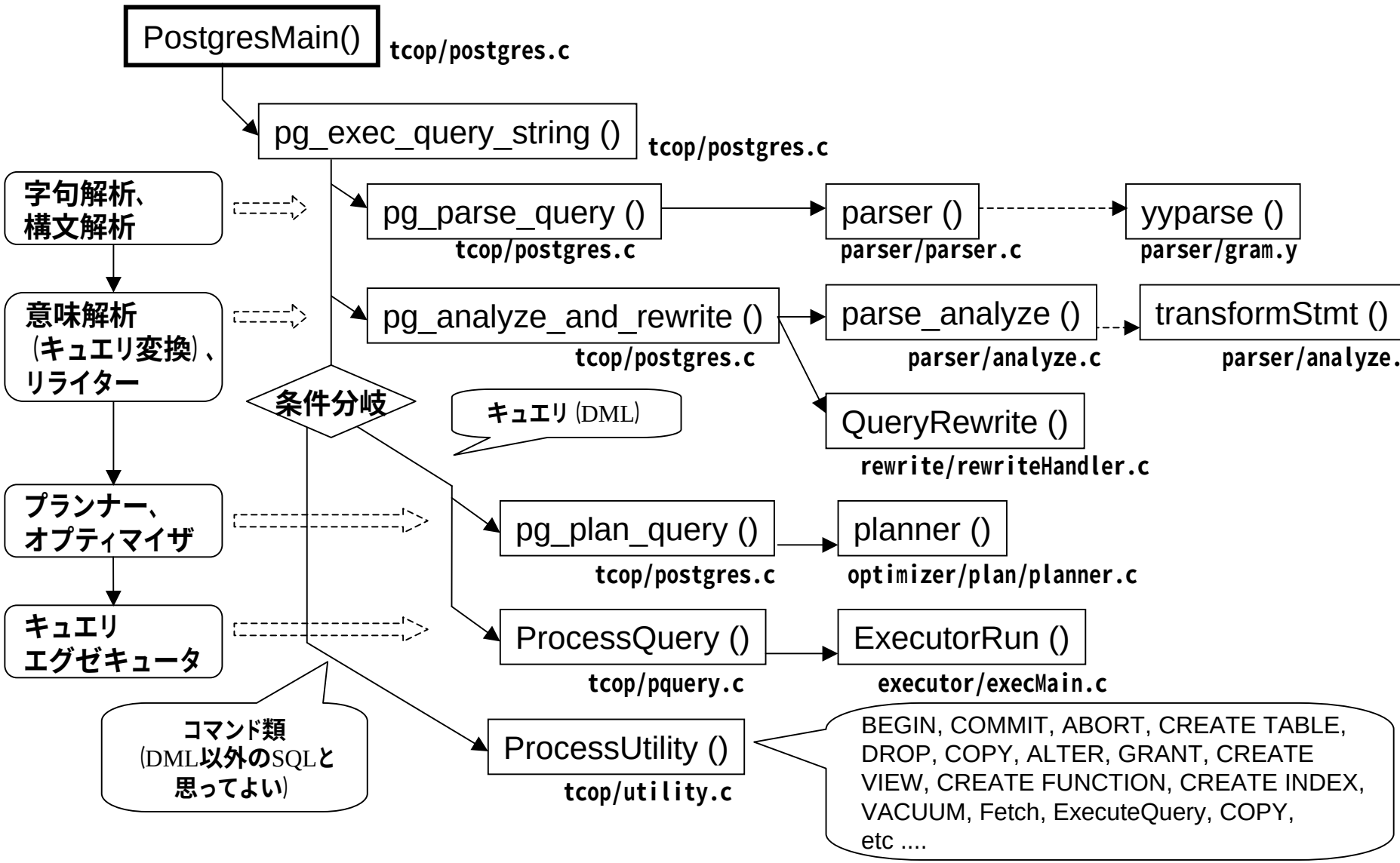


ソースコード流れ

main() からの処理



PostgresMain() からの処理



参考文献

- DATABASE SYSTEM IMPLEMENTATION
  - Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom
  - ISBN: 0130402648
  - **全く同じ内容を含む本**
    - DATABASE SYSTEMS THE COMPLETE BOOK (ISBN: 0130319953)

- Database Management System
  - Raghu Ramakrishnan, Johannes Gehrke
  - ISBN: 0071151109　(ISBN: 0072465638 ?)