PostgreSQL入門
～アーキテクチャ編～

Masahiko Sawada

PostgreSQLテクニカルウェビナー

2022/2/2

EDB

Agenda

- OSから見たPostgreSQL
- PostgreSQLの内部構造
- テーブル、インデックス
- 共有メモリ
- WAL
- まとめ

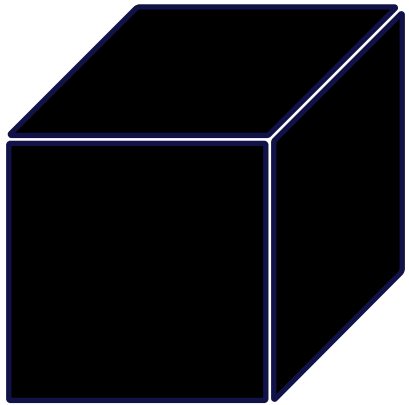
本Webinarについて

- PostgreSQL14を元に資料を作成しています
- 再来週に「クエリ編」を行います
- 本日のWebinarでは、PostgreSQLのアーキテクチャに焦点を当て、PostgreSQLを構成するコンポーネント、テーブル、インデックス等を解説します
- 目指すゴール
    - PostgreSQLの用語が理解できる
    - PostgreSQLを構成するコンポーネントとその役割が理解できる
    - 「インストールされたばかりのPostgreSQLがあるとして、特定テーブルに1件のレコードを最初にINSERTした場合、アクセスが発生するファイルとその理由をすべて教えてください」
        - 引用元：MySQLエキスパートyoku0825が目指す、DBAとしての未来像 https://engineering.linecorp.com/ja/interview/mysql-yoku0825/

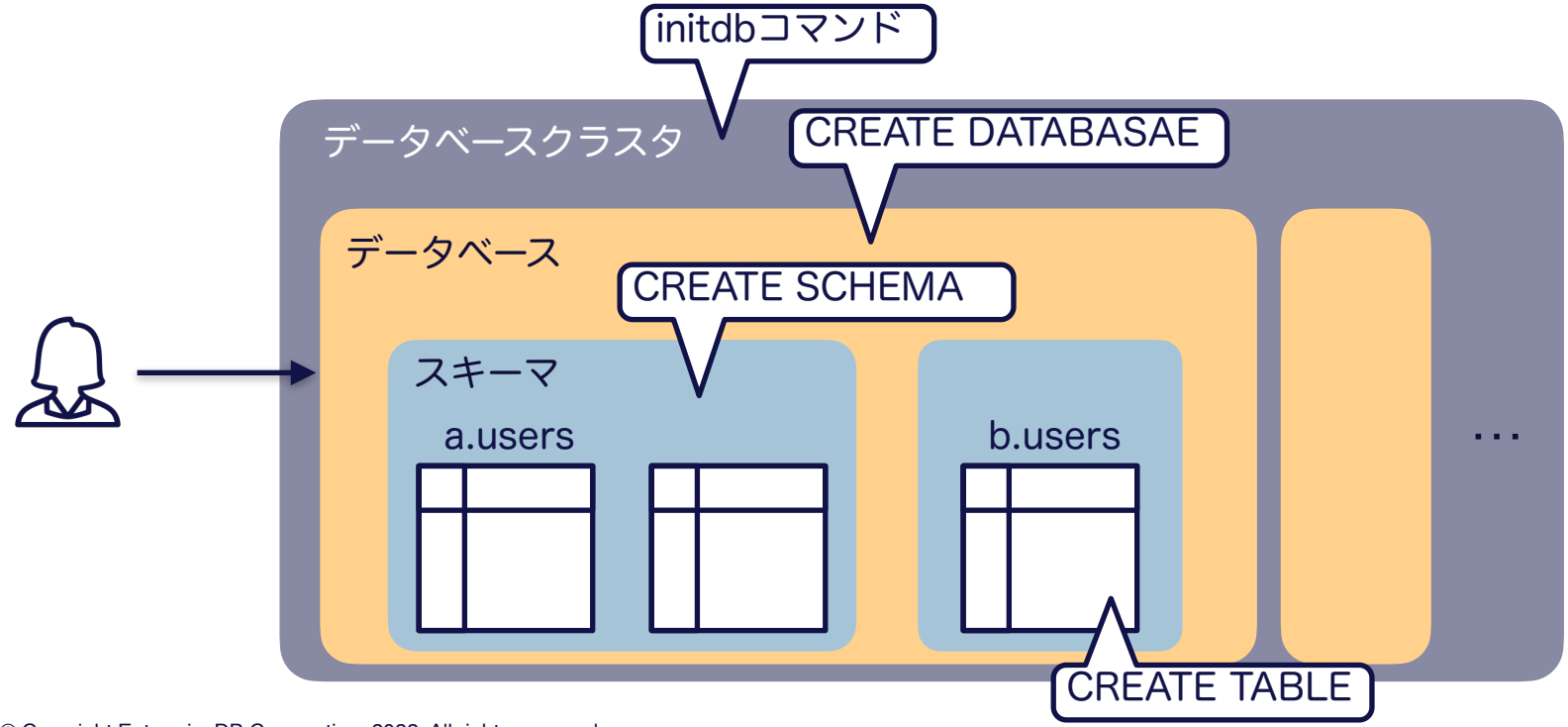
内部構造を知ることは必要？

Feel the breath of PostgreSQL

- トラブルシューティング、パフォーマンス・チューニング
  - クエリのレスポンスが悪いときはどうする？
  - 仮説をどう立てる？
- RDBMSの選定
  - 各RDBMS毎に特徴がある
  - PostgreSQLの得意、不得意を理解する



用語紹介と概要



EDB

OSから見た PostgreSQL

PostgreSQLのプロセス群

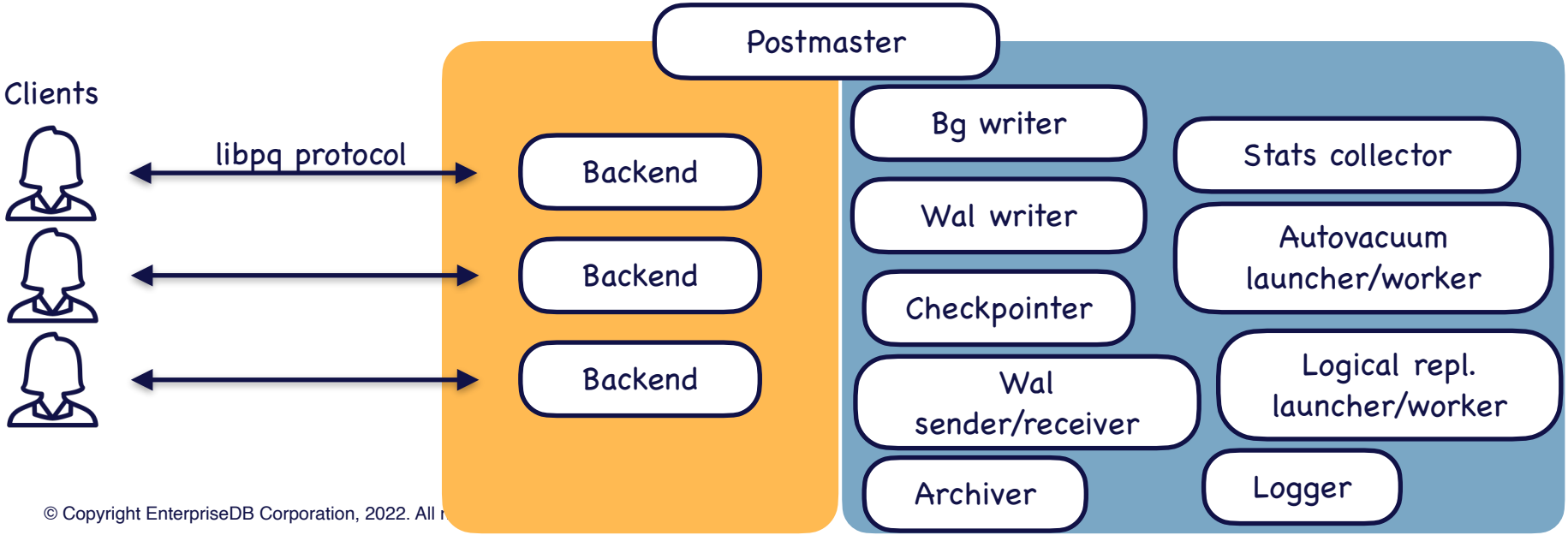
OSから見れば一つのプログラム

```
$ ll -h bin/postgres
-rwxr-xr-x  1 masahiko  masahiko    25M Oct 28 18:10 bin/postgres

$ ps x | grep postgres
72572  -  Ss   0:00.89 /usr/home/masahiko/pgsql/master/bin/postgres -D data -p 5432
72574  -  Ss   0:00.39 postgres: checkpointer  (postgres)
72575  -  Ss   0:01.82 postgres: background writer  (postgres)
72576  -  Ss   0:01.95 postgres: walwriter  (postgres)
72577  -  Is   0:00.34 postgres: autovacuum launcher  (postgres)
72578  -  Ss   0:01.15 postgres: stats collector  (postgres)
72579  -  Is   0:00.03 postgres: logical replication launcher  (postgres)
75547  -  Ss   0:00.00 postgres: masahiko postgres [local] idle (postgres)
```

PostgreSQLのプロセス群

OSから見れば一つのプログラム



PostgreSQLのディレクトリ構造

データベースクラスタの実体は一つのディレクトリ

```
$ ls ${PGDATA}
PG_VERSION     pg_dynshmem    pg_notify     pg_stat_tmp  pg_xact
base           pg_hba.conf    pg_replslot   pg_subtrans  postgresql.auto.conf
global         pg_ident.conf  pg_serial     pg_tblspc    postgresql.conf
pg_commit_ts   pg_logical     pg_snapshots  pg_twophase  postmaster.opts
pg_multixact   pg_stat        pg_wal        postmaster.pid
```

データベースクラスタの中身（1）

- base　　：テーブルやインデックスのデータ。データベース毎にディレクトリが切られている
- global　　：データベースクラスタ全体で共有するシステムカタログ
- pg_commit_ts　：各トランザクションのCOMMIT時刻を記録する
- pg_dynshmem：動的に確保する共有メモリで使う
- pg_logical　　：logical decoding/replicationで使う
- pg_multixact　：行レベルの共有ロックで使う
- pg_notify　　：LISTEN/NOTIFYで使う
- pg_serial　　：Serializableトランザクション分離レベルの時に使う
- pg_snapshots　：スナップショット共有機能で柄う
- pg_stat　　：稼働統計情報を格納するために使う

データベースクラスタの中身（2）

- pg_subtrans　：SAVEPOINTの時に使う
- pg_tblspc　　：テーブルスペース先へのシンボリックリンクに使う
- pg_twophase　：2相コミットで使う
- pg_wal　　：トランザクションログを格納する
- pg_xact　　：各トランザクションのCOMMIT/ABORT情報を格納する

少しずつ見えてきた！

・クライアントが一つのデータベースに接続すると、一つのプロセスがSQLを処理する

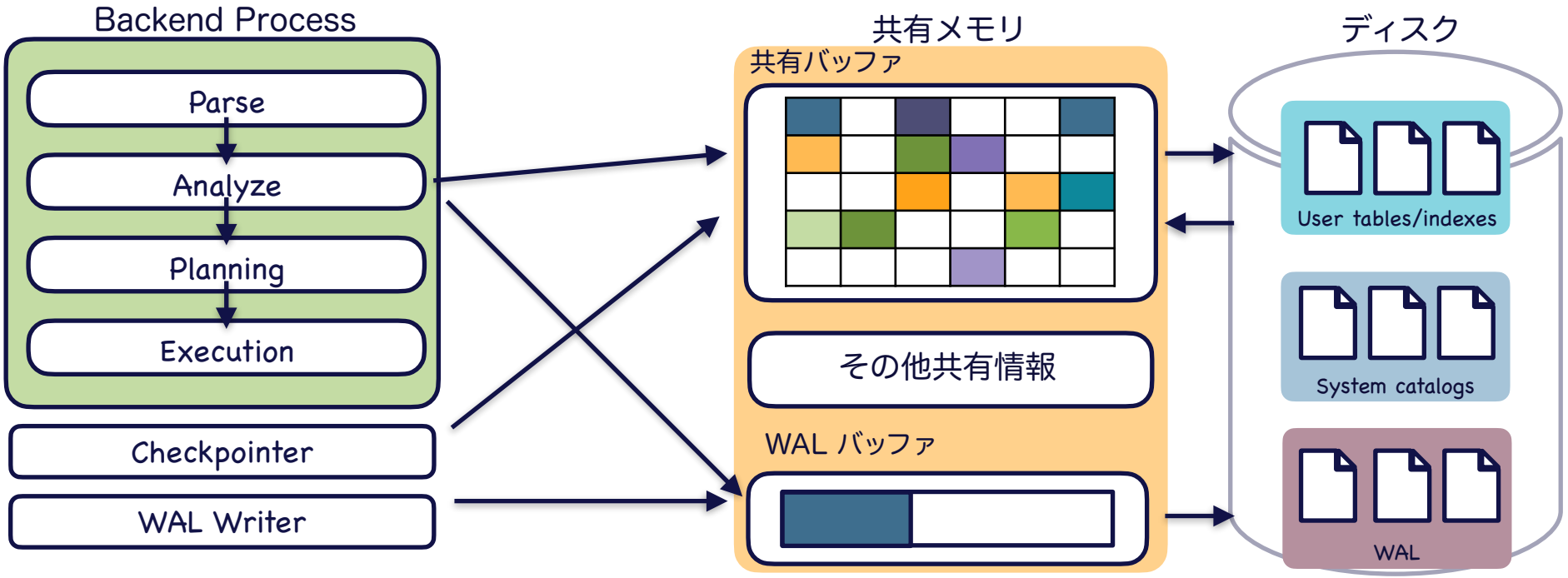
・PostgreSQLは様々な役割を持ったプロセスの集合

・データベースクラスタはOSからみると1つのディレクトリ

EDB

PostgreSQL内部から見たアーキテクチャ

PostgreSQLのアーキテクチャ



Backendプロセスによるクエリ処理

- 受信したクエリを解析し、”最適”な実行計画を作成する
- 作成した実行計画を実行する
- 詳細は再来週のWebinarで解説します

データベース

CREATE DATABASE …;

- 複数のテーブルやインデックスを格納する「箱」のようなもの
- 1つのインスタンスに複数のデータベースを作成することが可能
- クライアントは特定のデータベースに接続する
- データベースの実体は「ディレクトリ」
- template0, template1, postgresの3つが作成される

```
$ ls -l ${PGDATA}/base/
total 0
drwx------ 296 masahiko staff 9472 Jan 31 22:55 1
drwx------ 296 masahiko staff 9472 Jan 31 22:55 13237
drwx------ 296 masahiko staff 9472 Jan 31 22:55 13238
```

テーブルとインデックス

CREATE TABLE …;　CREATE INDEX …;

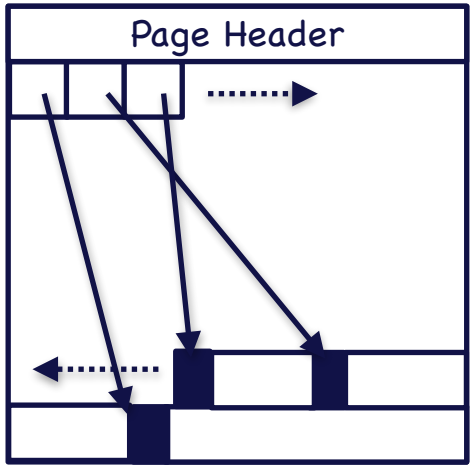
- OID（Object ID）が割り当てられる
- テーブルやインデックスの実体はファイル
  - 1テーブルは複数のセグメントファイル（最大1GB）で構成される
- 各テーブルは8kBのブロックから構成される
  - PostgreSQLのディスクI/O単位は8kB

```
=# select pg_relation_filepath('test');
 pg_relation_filepath
----------------------
 base/13238/16384
(1 row)
```

テーブルの中身

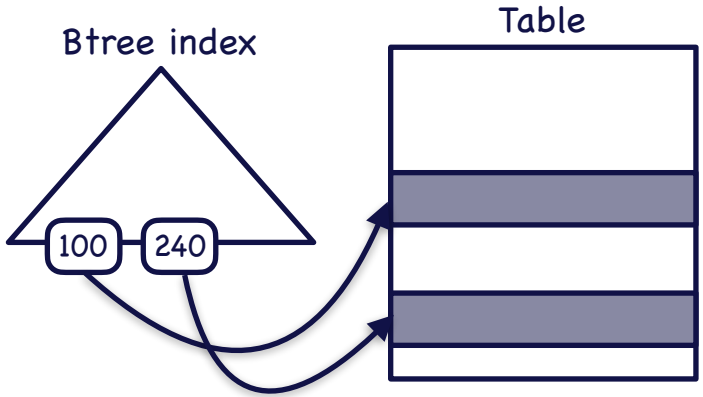
行指向のテーブル、別名Heap

- ページ＝ブロック
- 複数のページから構成される
- タプルはTID(ブロック番号＋オフセット番号）で一意に表現できる
- 固定長のItemIdはページ上部から埋められる
- 可変長のタプルはページ下部から埋められる
- 各テーブルには、各ページ空き領域を管理する
  空き領域マップ（Free Space Map）がある
- FSMを使って挿入するページを探す



インデックスの中身

Btree、GIN、BRINなど



・複数のインデックスタイプをサポート

・Btree

　・各テーブルタプルに対して1つのインデックスが対応する

　・ツリーの各ノードはページに対応

・BRIN（Block Range INdex)

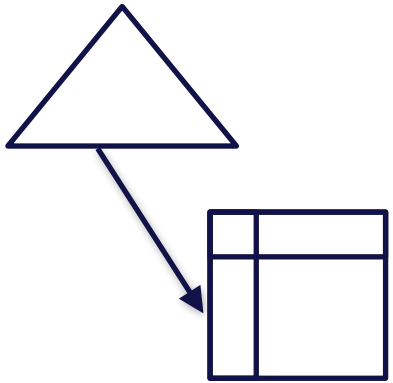
　・複数ブロックのサマリ情報に対して1つインデックスを対応させる

[少し脱線] Table/Index Access Method

- Heap、Btreeインデックスなどの特定のデータ構造を総称してTable/Index Access Method
- PostgreSQLでは、ユーザが任意のTable/Index Access Methodを実装可能
- 現在はテーブルは、Heapのみが同梱
- インデックスは、Btree、BRIN、Hash、GIN、Gist、SPGist、Bloomなどが同梱

代表的なアクセス方法

- Seq Scan
  - テーブルの（物理的な）先頭から順番にスキャンする
  - テーブル内の多くの割合のデータを取得する時に高速
- Index Scan
  - インデックスをスキャンしてから、テーブルをスキャンする
  - テーブル内の一部のデータを取得する時に高速



システムカタログ

PostgreSQLのメタデータが詰まった内部テーブル

- 実体は通常のテーブルと同じ
- PostgreSQL14には62個ある
- テーブル、テーブルにある列、インデックス、データ型、演算子、トリガー、関数、ユーザ、型変換など、内部的な情報はすべてシステムカタログにある
- テーブルとしてpg_catalogスキーマに存在するので、SQLでアクセスできる
    - 例）テーブルを検索する時：
        - SELECT relname FROM pg_class WHERE relname = ‘user’ and relkind = ‘r’;

[少し脱線] 共有する／しないシステムカタログ

- データベース、ユーザなどはデータベースクラスタ全体で共有するので対応するカタログ（pg_database, pg_authidなど）は共有するシステムカタログ(pg_class.isshared = 't')
- その他の定義情報（テーブル、トリガなど）は、データベース毎に存在する

例）テーブルを作った時

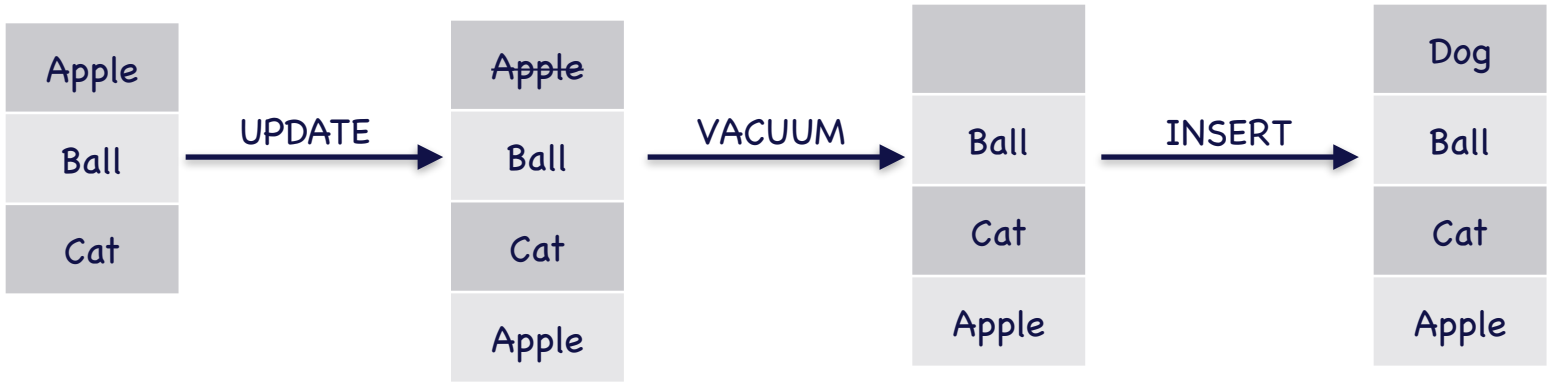
```
CREATE TABLE users (id INT PRIMARY KEY, name TEXT);
```

- テーブル、インデックス(主キー) → pg_class
- インデックス(主キー) → pg_index
- id列、name列 → pg_attribute
- 主キー制約 → pg_constraint
- 依存関係（テーブルを削除したら主キーも削除する） → pg_depend

追記型とVacuum

PostgreSQLのガーベージコレクション機能

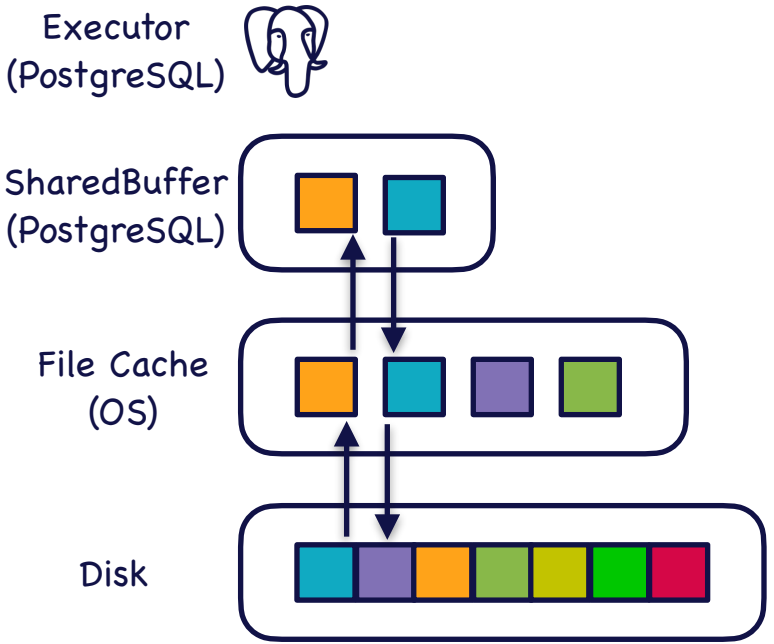
- Vacuumで不要になった領域を回収し再利用できるようにする
- 物理的なサイズは小さくならない



共有バッファ

PostgreSQLはデータをどのように管理するのか？

- テーブルやインデックスの実体は1つのファイル
  - 8kB（ブロック）単位で読み書きする
- 共有バッファに読んだデータをキャッシュする
  - OSのファイルキャッシュも利用する
  - shared_buffersで調整
  - 満杯になったら入れ替える（Clock Sweep）
  - CheckpointerやBgwriterが変更したデータをディスクに書き込む



PostgreSQLのメモリ

共有するメモリとしないメモリ

- プロセス間で共有するデータは共有メモリに確保
  - 共有バッファ、ロック、管理情報など
  - 例）shared_buffersは共有メモリとして一つ確保
- プロセス内だけで使うメモリはプロセス毎に確保
  - プランニング時に使うメモリ、ソート等のSQL実行時に使うメモリ、システムカタログのキャッシュなど
  - 例）work_memはプロセス毎に確保する

よくある質問

- 大きなテーブルを全件SELECTすると共有バッファはそのテーブルで一杯になる？
    - リングバッファを使うのでそのテーブルデータで一杯にはならない
- バッファ上のページはどのようなタイミングでディスクに書かれる？
    1. Background Writerプロセスが定期的に書く
    2. CheckpointerプロセスがCHECKPOINT実行時に書く
    3. Backendプロセスが別のページを読み込むために書く

トランザクションログ、WAL

・メモリには最新のデータが乗っている

・メモリ上のデータは揮発

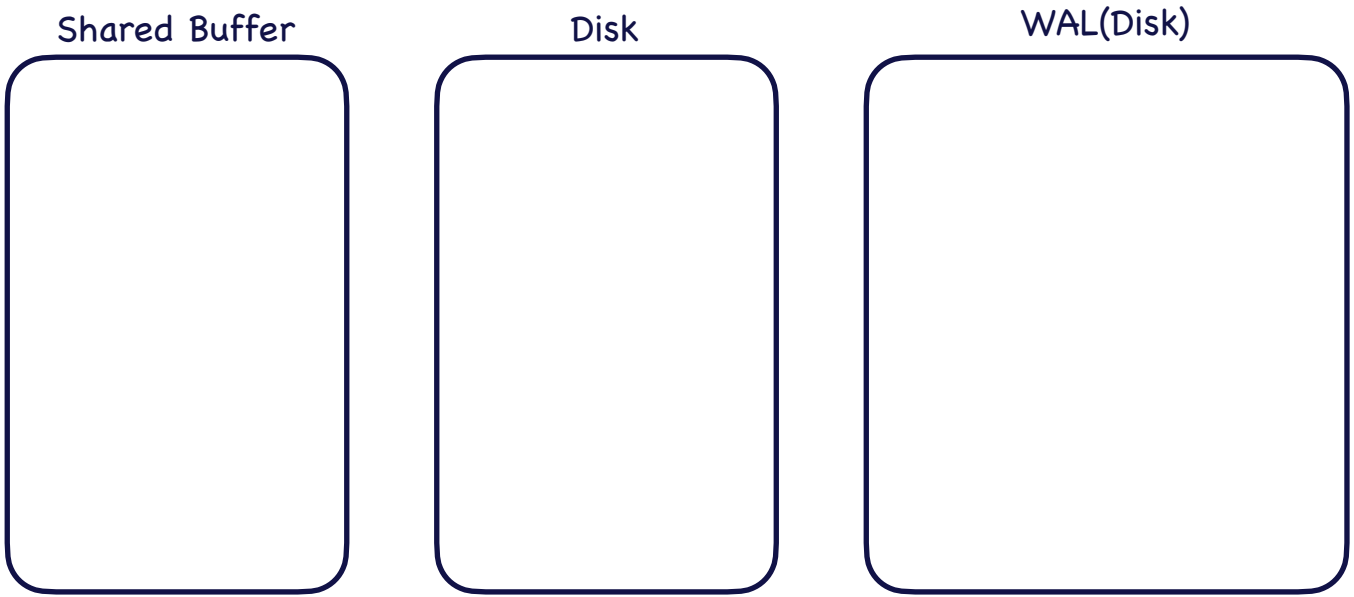
・サーバの突然の電源断、停電、故障時等でもデータを失ってもいけない

すべてはログに書いてある

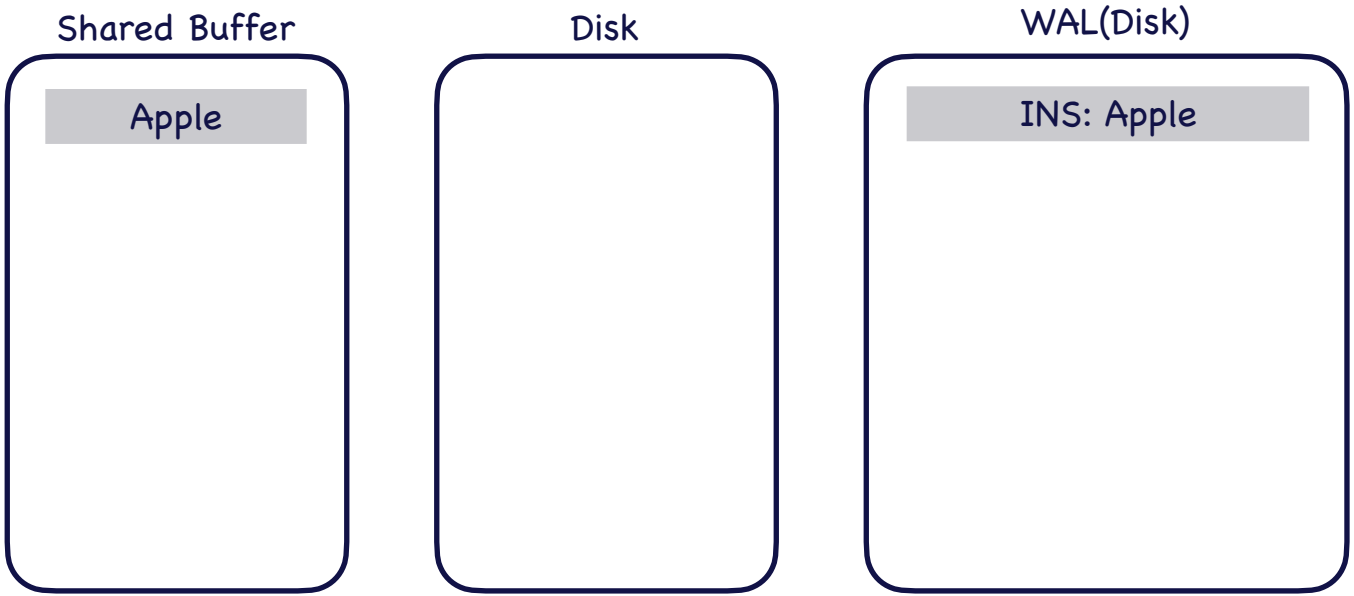
- コミット済みのデータを失わない！
- ACIDのAとD（Durability）
- WAL (Write Ahead Logging)
  - データ（テーブルやインデックス）の変更をディスクに反映する前に、必ず対応するログをディスクに反映する
  - COMMIT後にクラッシュしてもログを再生すれば元通り（ロールフォワード）

チェックポイント

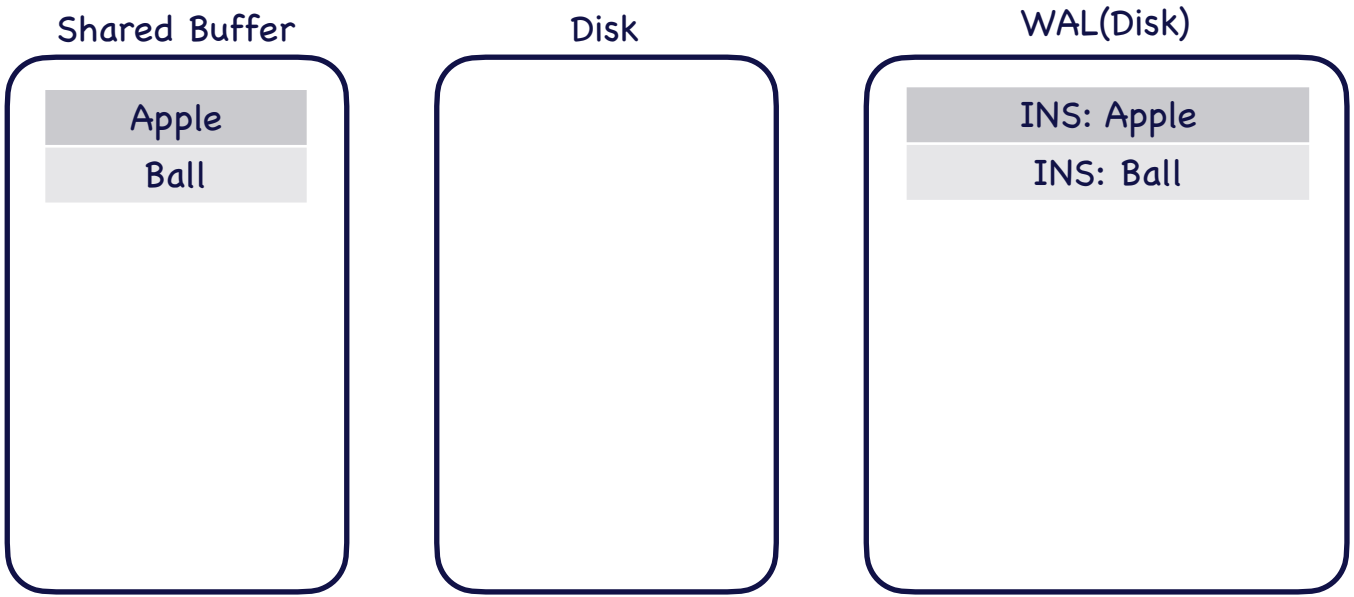
・共有バッファ上のデータをディスクに書き出す
　　・チェックポイント中もデータの読み書きは可能
・CHECKPOINTコマンドで手動実行
・max_wal_size、checkpoint_timeoutパラメータによって自動実行
・サーバクラッシュ後は、前回のチェックポイントから復旧する（リカバリ）する
　　・チェックポイントは復旧（リカバリ）時間を短縮する
・Log Sequence Number（LSN）：各WALレコードに割り当てられる一意の数字
　　・実際はWALレコードのバイトオフセット



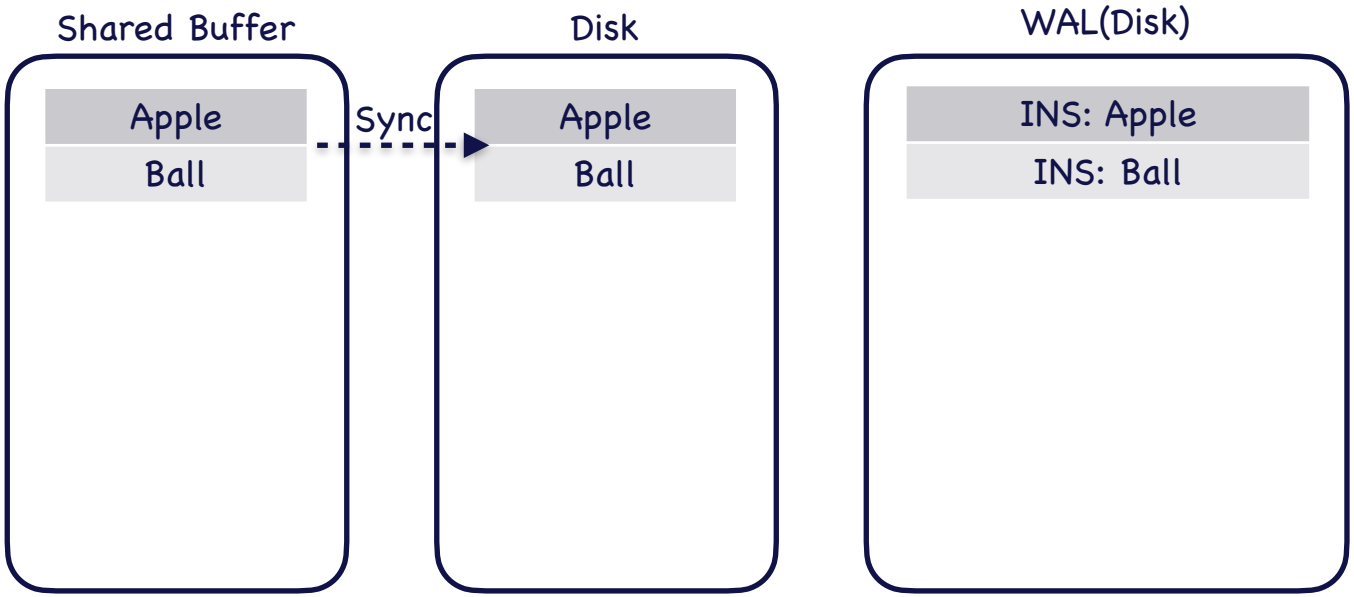
INSERT 'Apple';



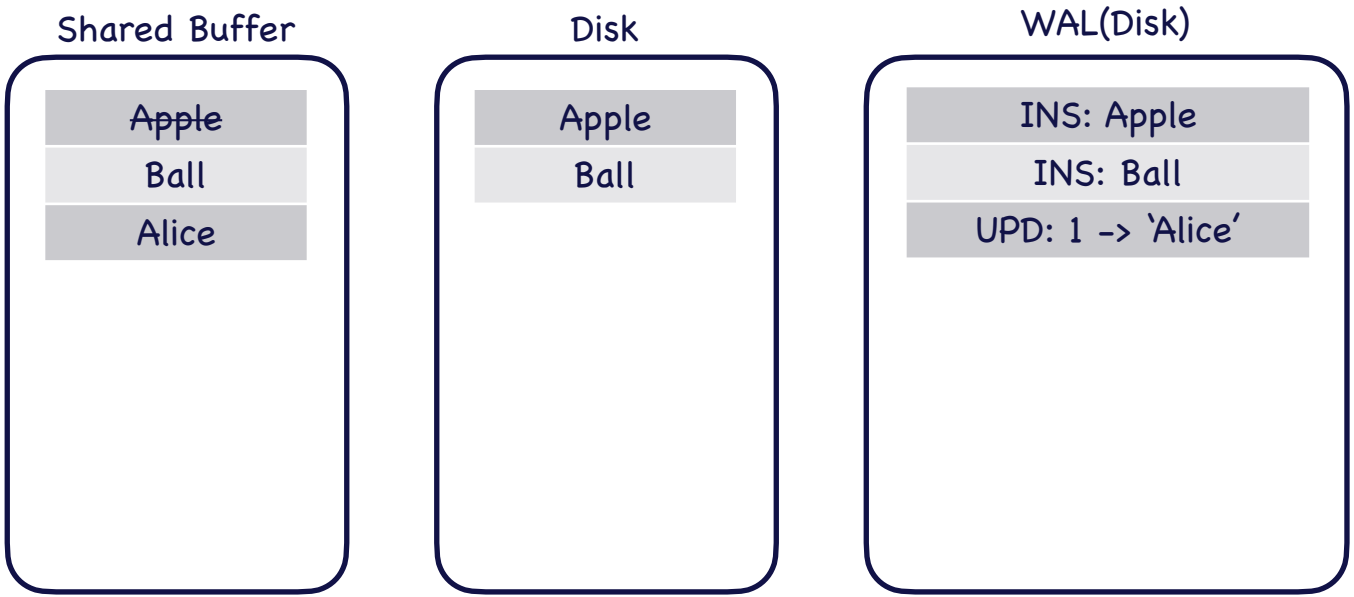
INSERT 'Apple';
INSERT 'Ball';



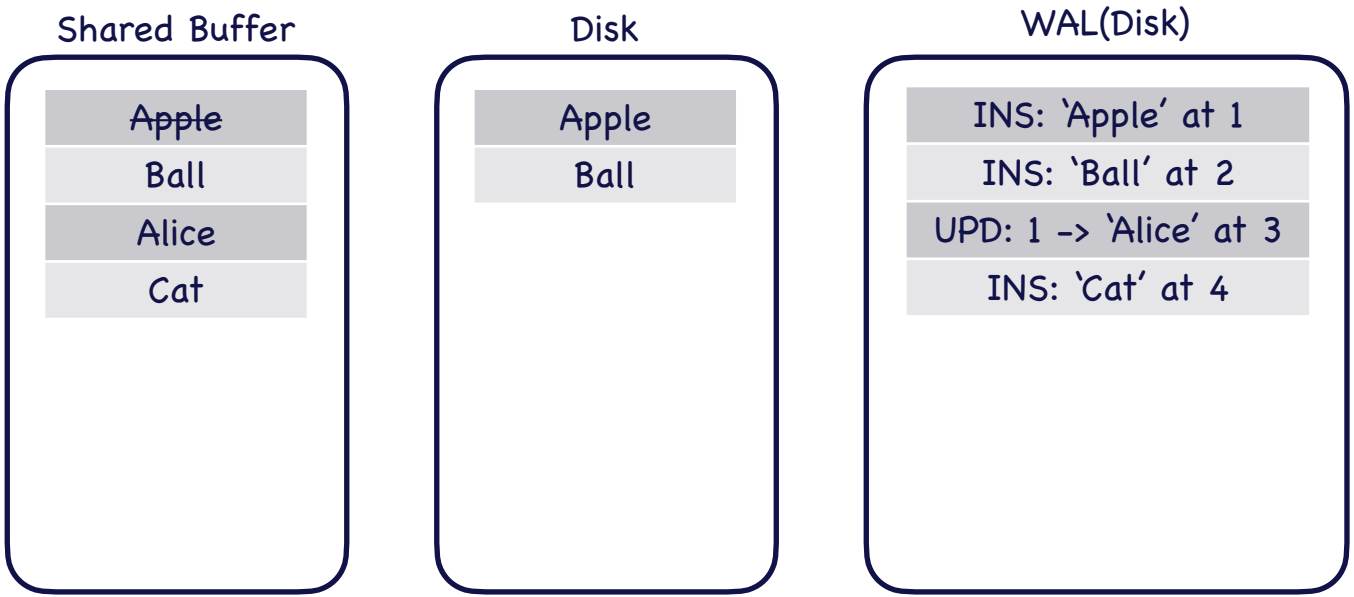
INSERT 'Apple';
INSERT 'Ball';



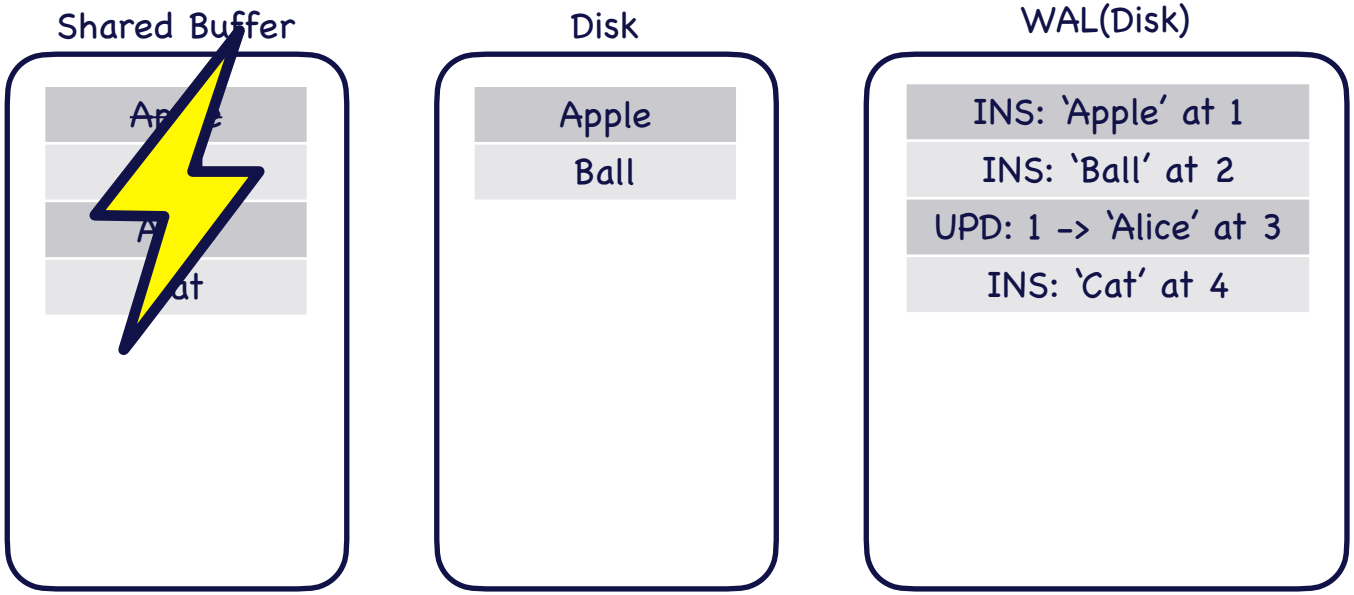
INSERT 'Apple';
INSERT 'Ball';
UPDATE 'Apple' -> 'Alice';



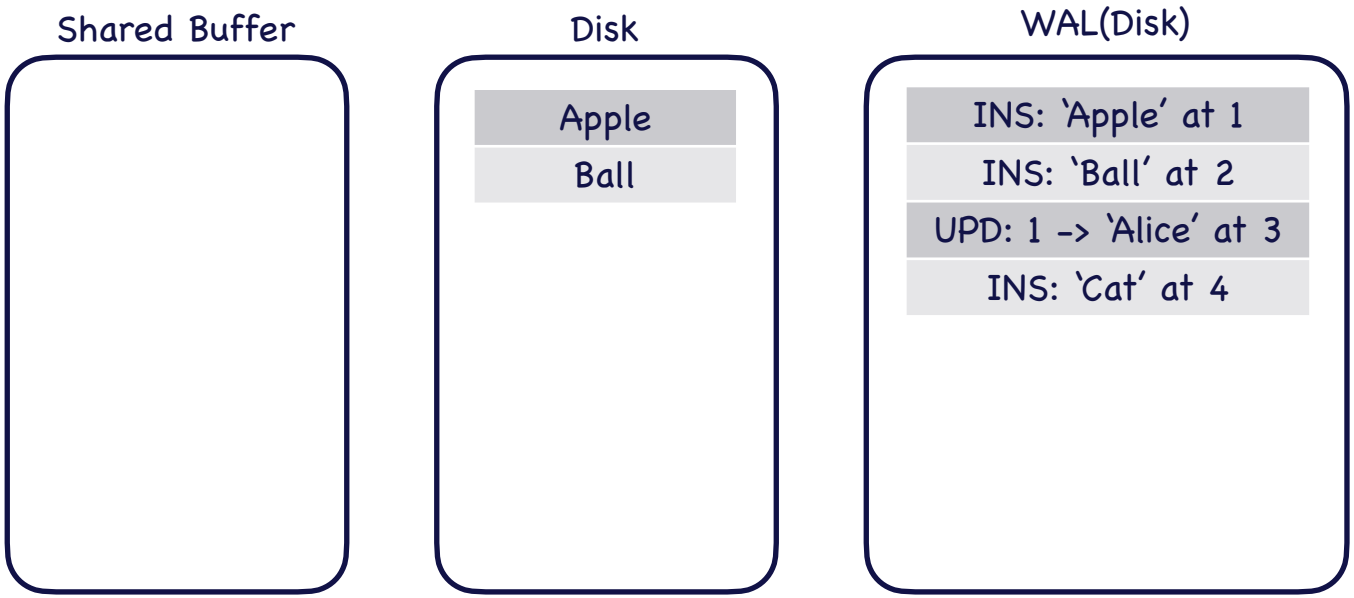
INSERT 'Apple';
INSERT 'Ball';
UPDATE 'Apple' -> 'Alice';
INSERT 'Cat';



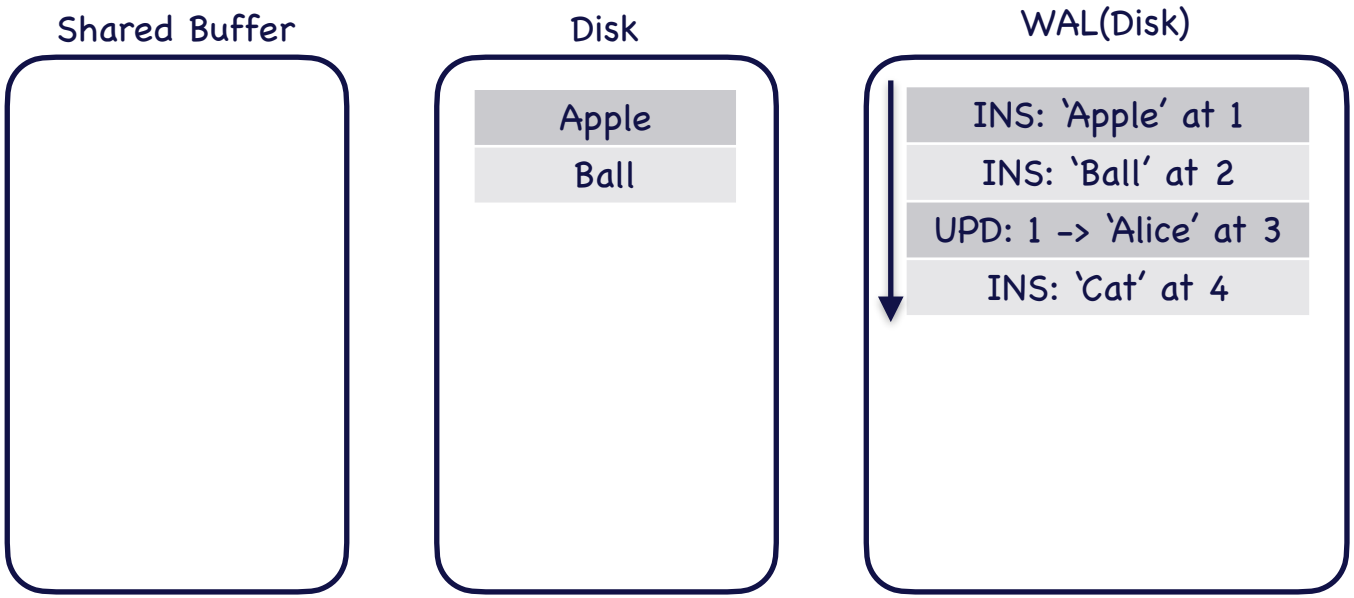
INSERT 'Apple';
INSERT 'Ball';
UPDATE 'Apple' -> 'Alice';
INSERT 'Cat';



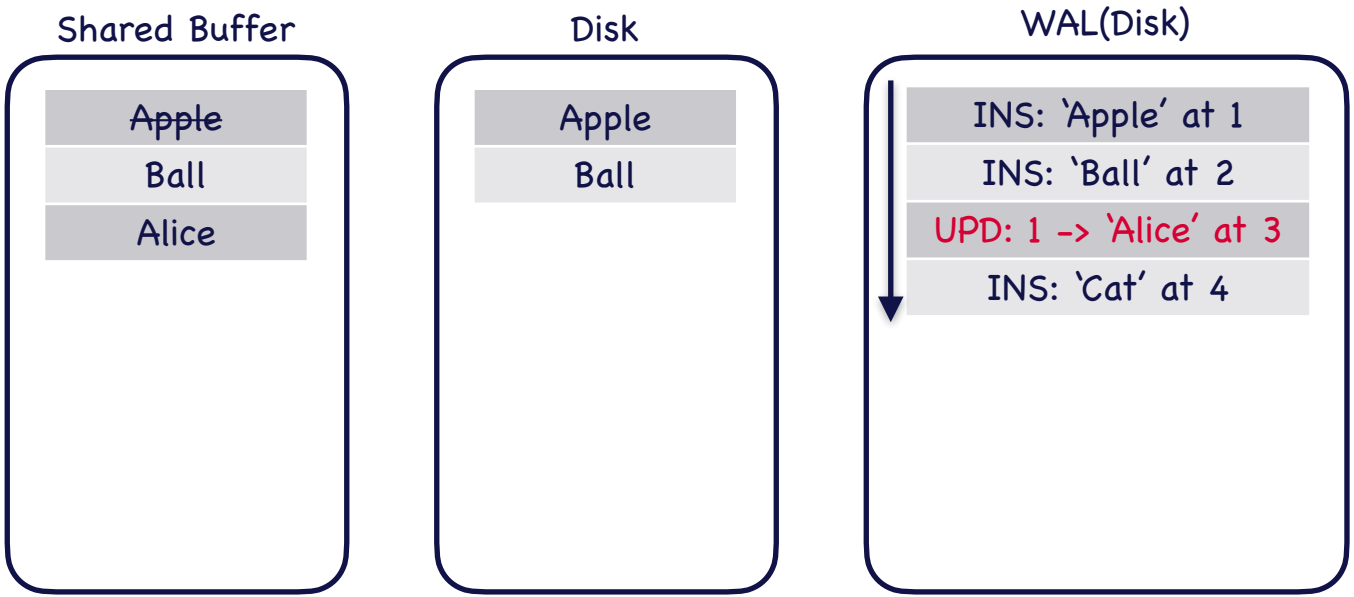
```
INSERT 'Apple';
INSERT 'Ball';
UPDATE 'Apple' -> 'Alice';
INSERT 'Cat';
```



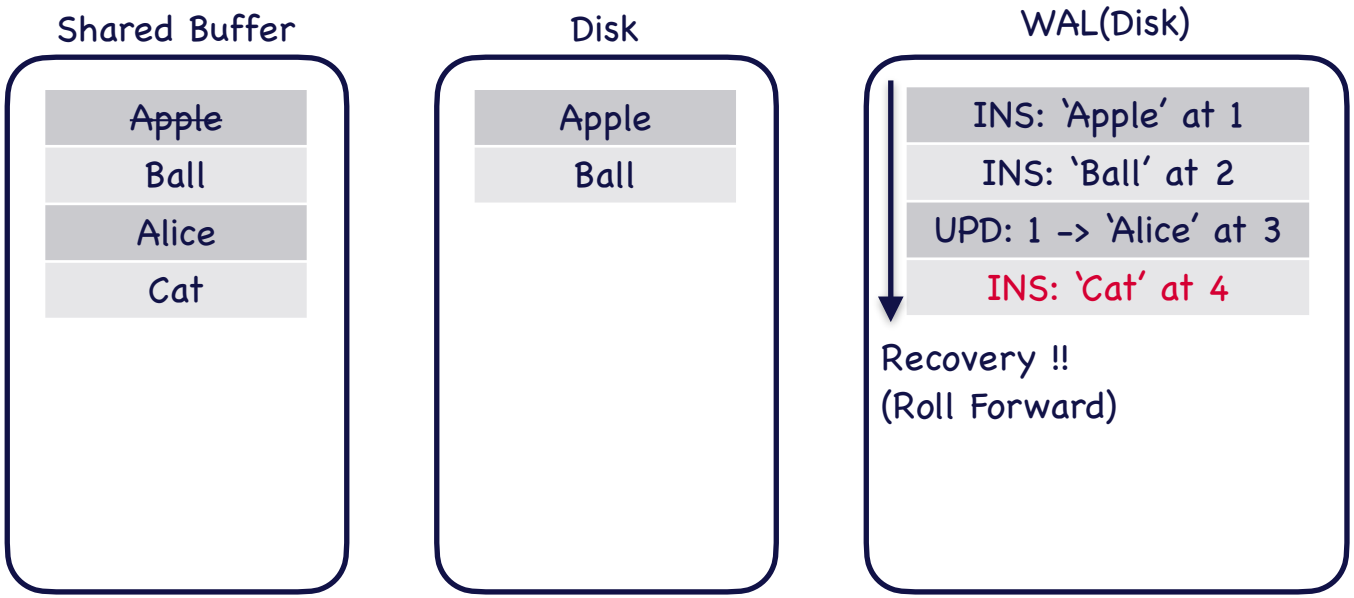
```
INSERT 'Apple';
INSERT 'Ball';
UPDATE 'Apple' -> 'Alice';
INSERT 'Cat';
```



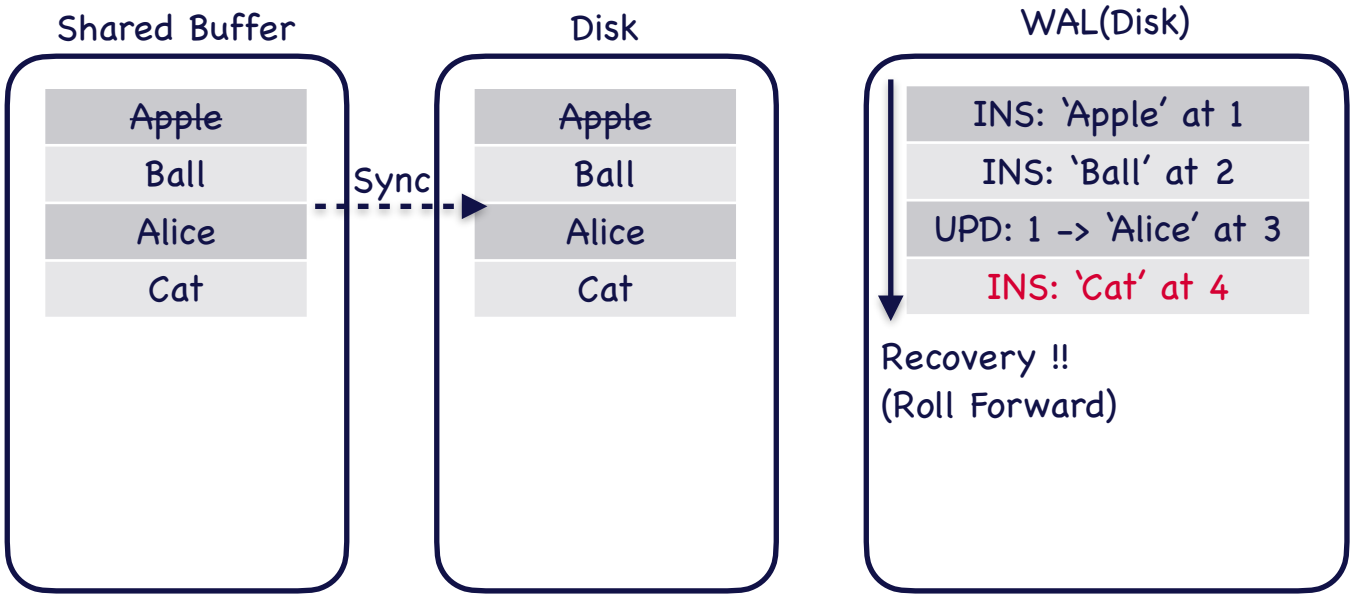
```
INSERT 'Apple';
INSERT 'Ball';
UPDATE 'Apple' -> 'Alice';
INSERT 'Cat';
```



INSERT 'Apple';
INSERT 'Ball';
UPDATE 'Apple' -> 'Alice';
INSERT 'Cat';



INSERT 'Apple';
INSERT 'Ball';
UPDATE 'Apple' -> 'Alice';
INSERT 'Cat';



すべてはログに書いてある

- クラッシュしてもCOMMIT済みのデータは失われない！
- WALはSequential Writeになる
- WALはpg_wal配下に格納される。絶対消さないで！

```
$ ls ${PGDATA}/pg_wal/
000000010000000000000002B  000000010000000000000032 archive_status
000000010000000000000002C  000000010000000000000033
000000010000000000000002D  000000010000000000000034
000000010000000000000002E  000000010000000000000035
000000010000000000000002F  000000010000000000000036
000000010000000000000030  000000010000000000000037
000000010000000000000031  000000010000000000000038
```

まとめ

- PostgreSQLの基本的なアーキテクチャを解説しました
  - プロセス構成、共有バッファ、テーブル、インデックス、Vacuum、システムカタログ、WALなど
- 2/19のクエリ処理編にもぜひご参加ください！
  - どうやってクエリを処理しているのか？
  - どのように実行計画を作っているのか？
  - どのように同時実行制御しているのか？

EDB

Thank you

Masahiko Sawada

masahiko.sawada@enterprisedb.com