OSS-DB

OSS-DB Silver
技術解説無料セミナー
2020/9/5 開催

主題	運用管理（出題範囲 52 %）
副題	基本的な運用管理作業 【重要度：7】の ・VACUUM、ANALYZEの目的と使い方 ・自動バキュームの概念と動作

本日の講師

株式会社NGN-SF
平野 幸児

LPI-JAPAN

講師紹介

■平野　幸児

#OSS-DB

- 株式会社 NGN-SF テクニカルチーフインストラクタ
- VMware 認定インストラクター（"VCI Special Award 2018"受賞）
- 担当研修：OSS-DB 資格対応研修やVMware 社の仮想化ソリューション研修など
- 2012年6月 OSS-DB Gold 資格取得
- 講師から一言：PostgreSQLは毎年のバージョンアップが楽しみ！
  （研修講師としては、ついていくのが大変ですが……）

■株式会社 NGN-SF　https://www.ngn-sf.co.jp/

- JR山手線・都営浅草線「五反田駅」から徒歩5分のIT研修会社
- LPI-Japan アカデミック認定校（OSS-DBとLinuC、OPCEL）
- OSS-DB やLinuCの資格試験対応研修、VMware社・Cisco社の認定トレーニングを実施

プロモーション

OSS-DB Silver 対応研修スケジュール

#OSS-DB

① 9/23～9/25
② 11/4～11/6
③ 12/2～12/4
④ 2/8～2/10

- 詳細は → https://www.ngn-sf.co.jp/course/ossdb01/

OSS-DB Gold 対応研修スケジュール

① 10/20～10/22
② 2/24～2/26

- 詳細は → https://www.ngn-sf.co.jp/course/ossdb02/

OSS-DBとは

#OSS-DB

オープンソースのデータベースソフトウェア「PostgreSQL」を扱うことができる技術力の認定です。様々な分野でPostgreSQLの利用拡大が進む中でOSS-DBの認定を持つことは、自分のキャリアのアピールにもつながります。

- ✓OSS-DB Goldは設計やコンサルティングができる技術力の証明

  PostgreSQLについての深い知識を持ち、データベースの設計や開発のほか、パフォーマンスチューニングやトラブルシューティングまで行えることが証明できます

- ✓OSS-DB Silverは導入や運用ができる技術力の証明

  PostgreSQLについての基本的な知識を持ち、データベースの運用管理が行えるエンジニアとしての証明ができます

- ✓対象のバージョンはPostgreSQL 11

PostgreSQLのVACUUMとANALYZE

#OSS-DB

基本的な運用管理作業 【重要度：7】

- **説明：**
  データベース管理者として実行する基本的な運用管理コマンドに関する知識を問う
- **主要な知識範囲：**
  PostgreSQLの起動・停止方法
  データベースロール / ユーザの概念
  データベースロール / ユーザの追加・削除・変更方法
  VACUUM、ANALYZEの目的と使い方
  自動バキュームの概念と動作
  システム情報関数
  情報スキーマとシステムカタログ
  テーブル単位の権限（GRANT/REVOKE）
- **重要な用語、コマンド、パラメータなど：**
  pg_ctl start / stop
  CREATE/ALTER/DROP ROLE/USER
  VACUUM
  ANALYZE
  vacuumdb
  autovacuum
  current_user

・OSS-DB　公式サイトの
「OSS-DB Silver 出題範囲」より引用
https://oss-db.jp/outline/silver

この資料の注意点

#OSS-DB

PostgreSQL バージョン11に準拠

- 資料内のパラメータ名やそのデフォルト値などは、バージョン11のもの

タプル：行やレコードのこと

- 日本PostgreSQLユーザ会の「PostgreSQL 日本語マニュアル」の表記に準拠
  https://www.postgresql.jp/document/11/html/index.html

目次

#OSS-DB

VACUUMの前に：PostgreSQLのテーブルファイル構造1

#OSS-DB

テーブルファイルのイメージ：本と本棚

1冊の本＝1タプルデータ

- **1つの棚に収まるように格納**

※棚に収まりきらないような巨大な本は、
"TOAST"という特殊な本棚に保存

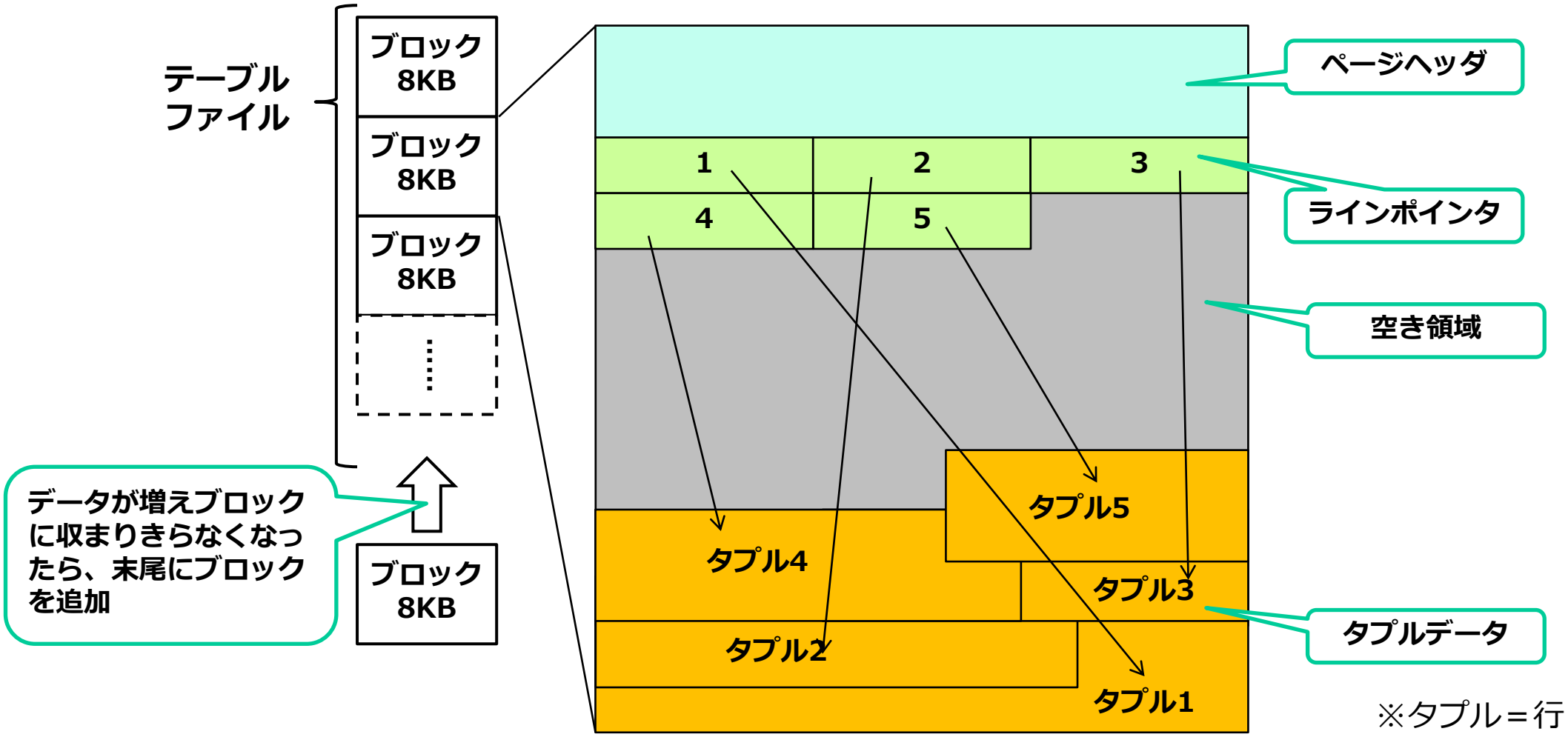
1つの棚＝1ブロック

- **本が棚に収まりきらなくなったら、新しく棚を増やす**

VACUUMの前に：PostgreSQLのテーブルファイル構造2

テーブルファイルの内部構造1：ブロック

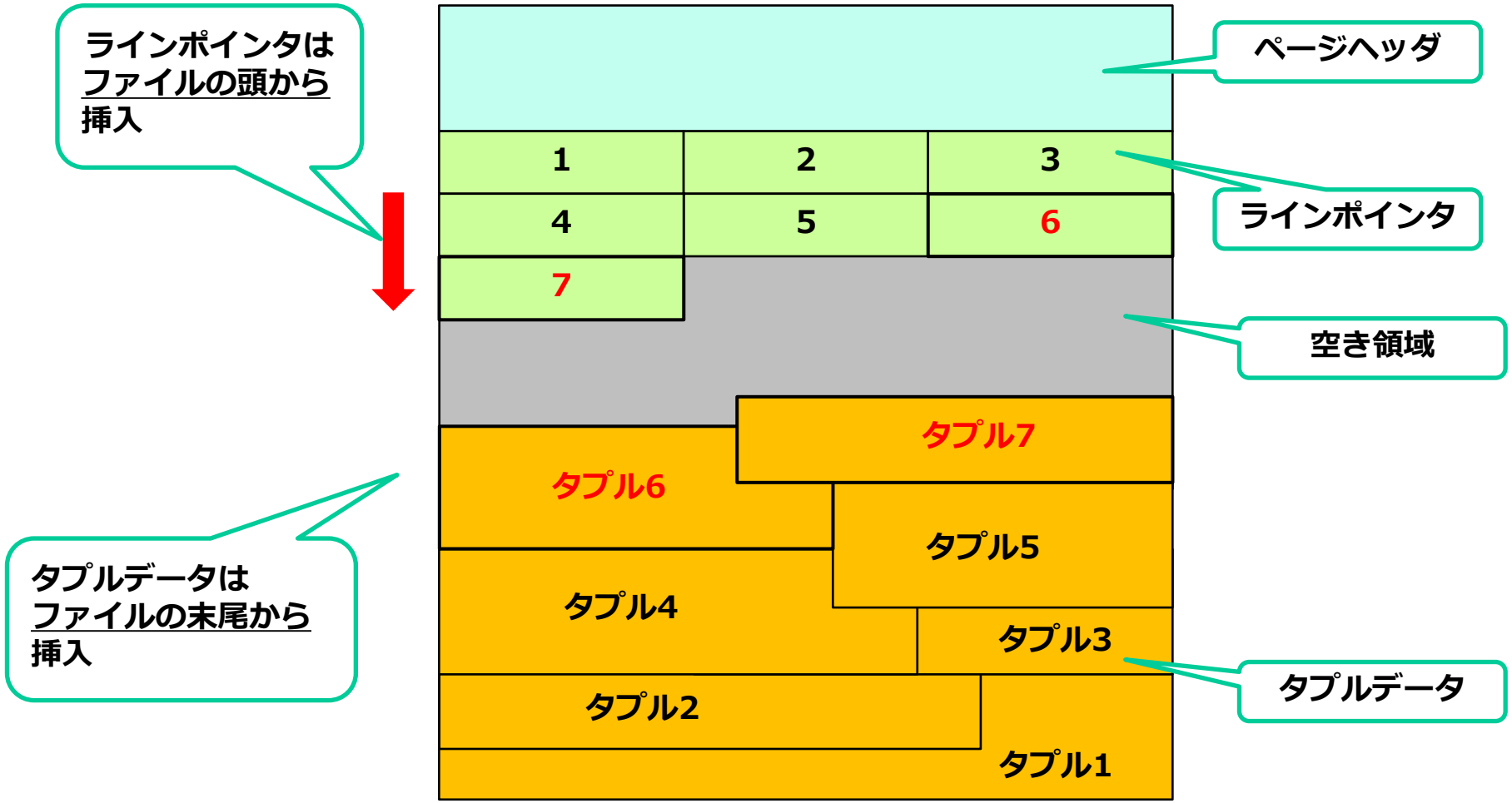
#OSS-DB



VACUUMの前に：PostgreSQLのテーブルファイル構造3

テーブルファイルの内部構造2：データの挿入

#OSS-DB



VACUUMの役割

#OSS-DB

■VACUUMの役割①：不要領域の回収（このセミナー内で説明）

- 不可視化されたデータ(=不要領域)が残ったままだと
  データ容量が肥大化し、キャッシュのヒット率も悪くなる
  → VACUUMで不要領域を回収

■VACUUMの役割②：XIDの周回問題回避（※OSS-DB Goldの出題範囲　付録で説明）

- XID：トランザクションの実行順番を表す32bitの符号なし整数で、
  約42億9000万の整数を巡回して使用
- 巡回で値が初期値に戻るとデータの可視化/不可視化処理が破たん

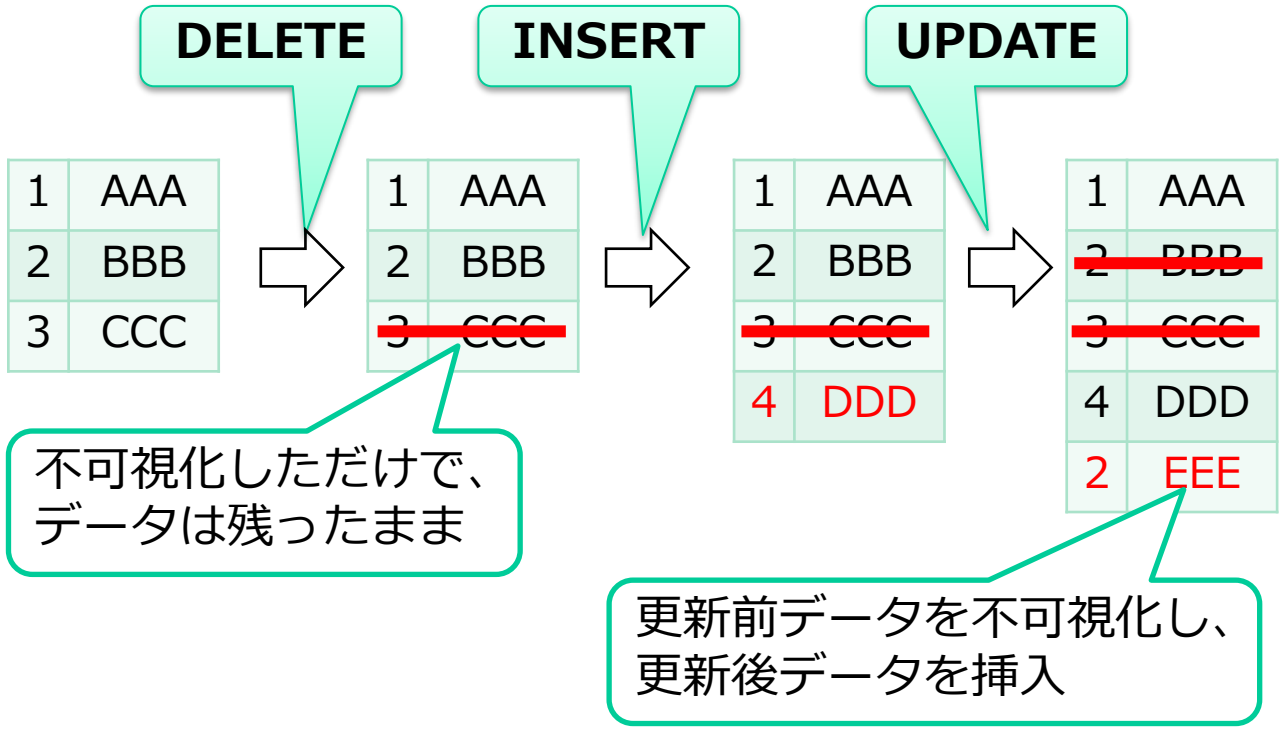
→ VACUUMで十分に古いXIDに対して「フリーズ処理」を施し、
周回問題を回避

追記型MVCCアーキテクチャとVACUUM1

#OSS-DB

PostgreSQLは追記型MVCCアーキテクチャ

- 「不要な本を廃棄せず残しておき、本がだんだん増えていく」イメージ
- DELETE：該当タプルを不可視化しただけで、データは残ったまま
- UPDATE：DELETE + INSERT処理を行い、更新前データを不可視化
- 不可視化したデータ＝**不要領域**

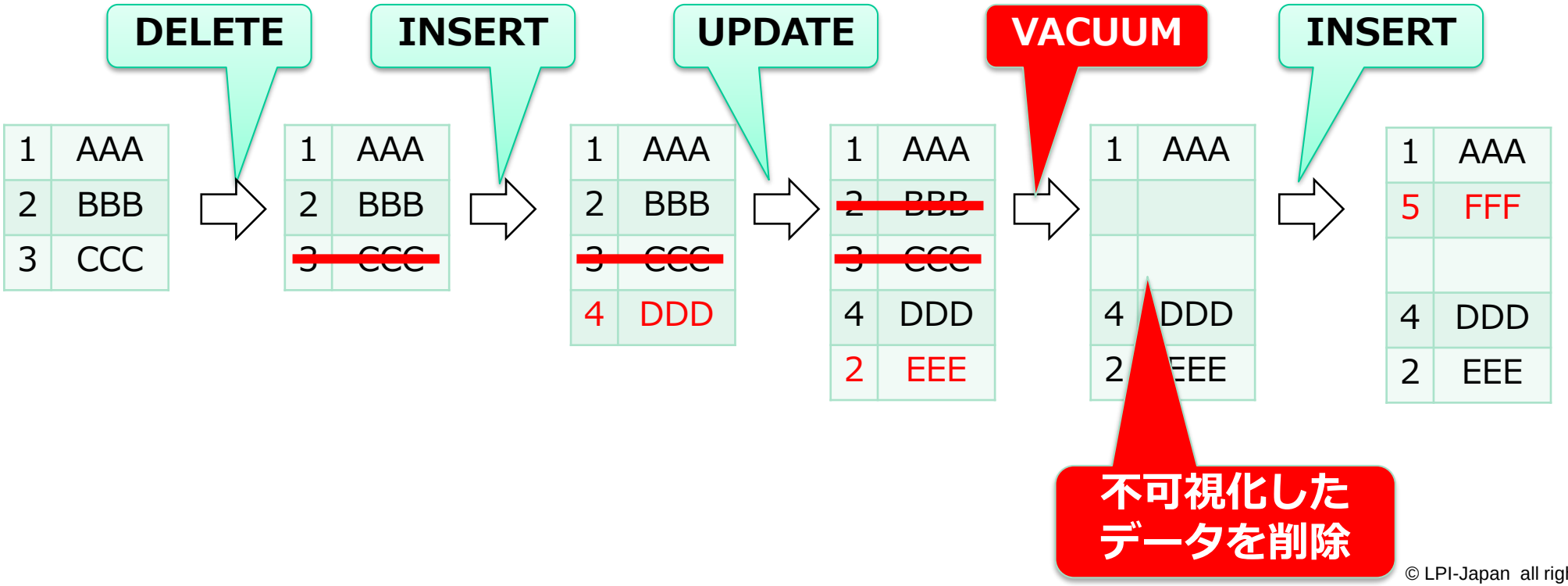


追記型MVCCアーキテクチャとVACUUM2

#OSS-DB

VACUUMで不要領域を回収

- **VACUUM**：不可視化されたデータを削除して、再利用可能にする処理
- 「本棚の不要な本を破棄し、空きスペースを作る」イメージ



2種類のVACUUM

#OSS-DB

	①VACUUM	②VACUUM FULL
処理方法	ブロック内で不要領域を削除 →　ブロック内で並び替え	テーブルファイルの再作成
テーブルロック	SHARE UPDATE EXCLUSIVE ※SELECTとINSRT、UPDATE、DELETE可能	ACCESS EXCLUSIVE ※最も厳しい排他ロックで、SELECTすら不可
テーブルファイルサイズ	基本的に、ファイルサイズは変わらない	一般的に、ファイルサイズが大きく削減される
備考	・別名：コンカレントVACUUM ・テーブルファイルの末尾に空のブロックがあれば、その分のサイズは削減される	・実行中は、対象テーブルへの全てのアクセスが待機 → サービス停止に直結 ・ディスクの空き容量が必要

コンカレントVACUUMの処理イメージ

#OSS-DB

■定期的な本棚の整理

- 1つの棚ごと不要な本を廃棄し、整理整頓

■不要な本＝不要領域

- **棚ごとに不要な本を廃棄し、空きスペースを確保**

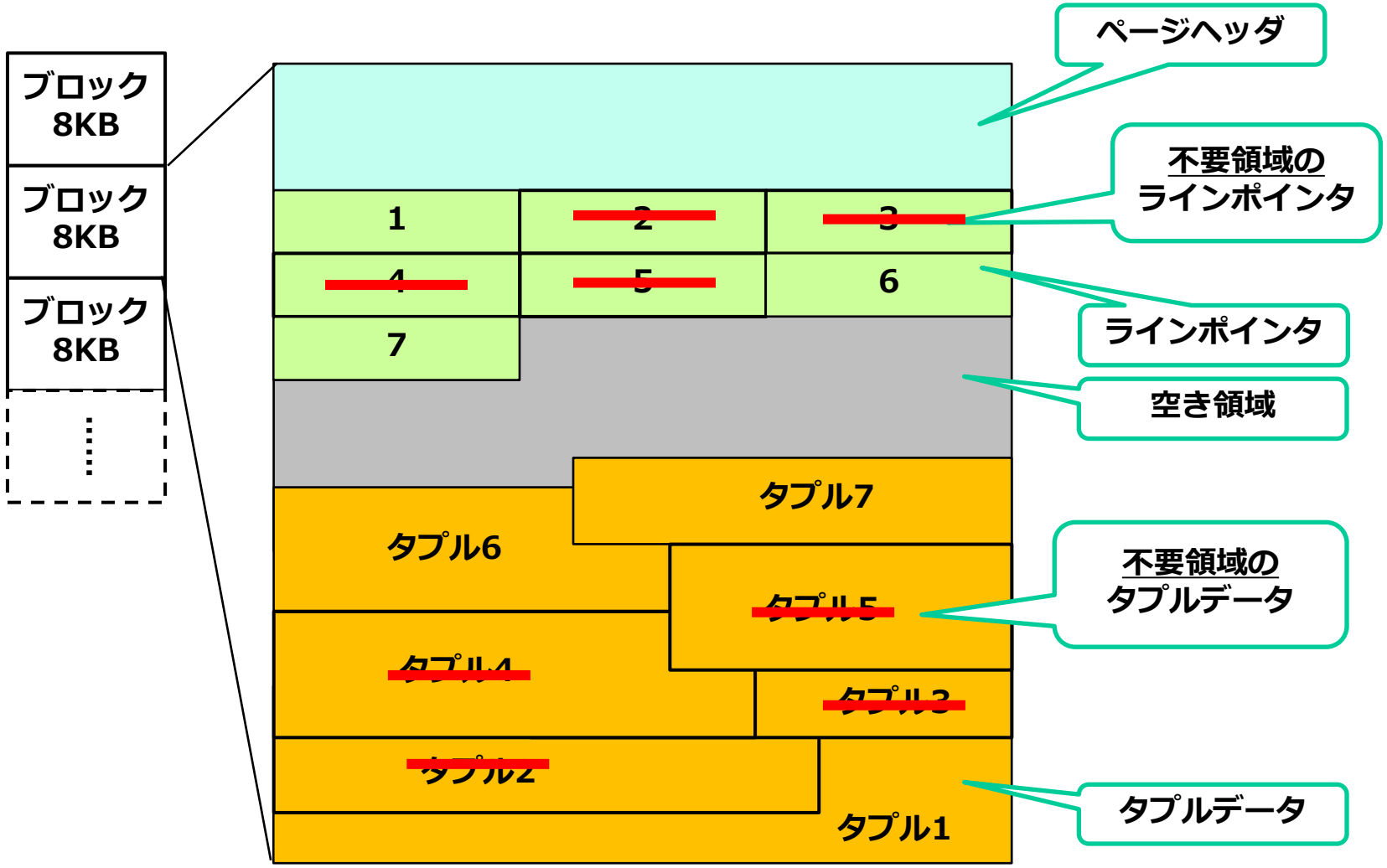
■1つの棚（＝1ブロック）ごとに実施

- **棚をまたいでの整理整頓はしない**
- **基本的に、本棚自体の処分はしない**

コンカレントVACUUMの処理（1/3）

■不要領域を回収し、ブロック内で並び替え①

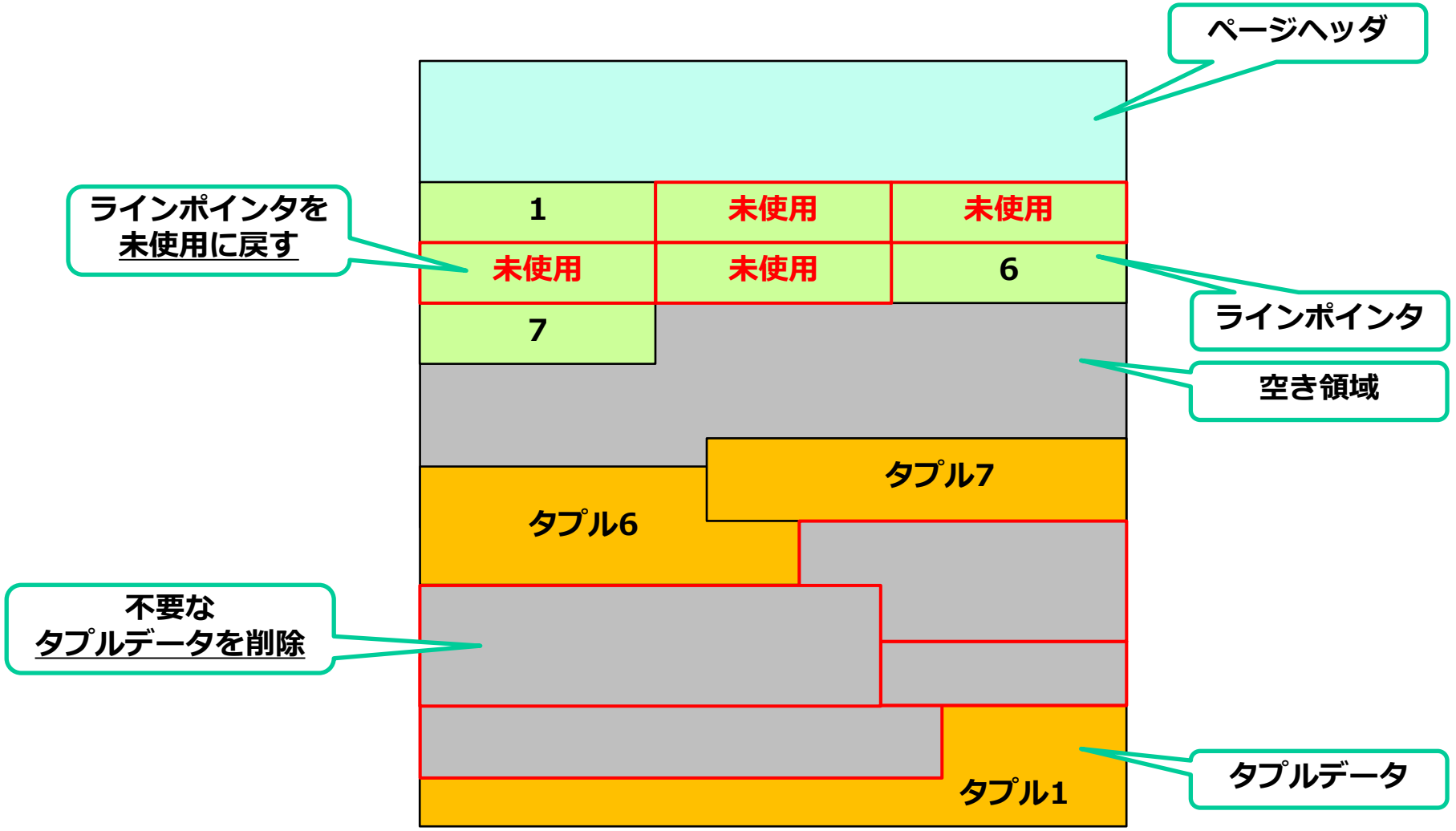
#OSS-DB



コンカレントVACUUMの処理（2/3）

■不要領域を回収し、ブロック内で並び替え②

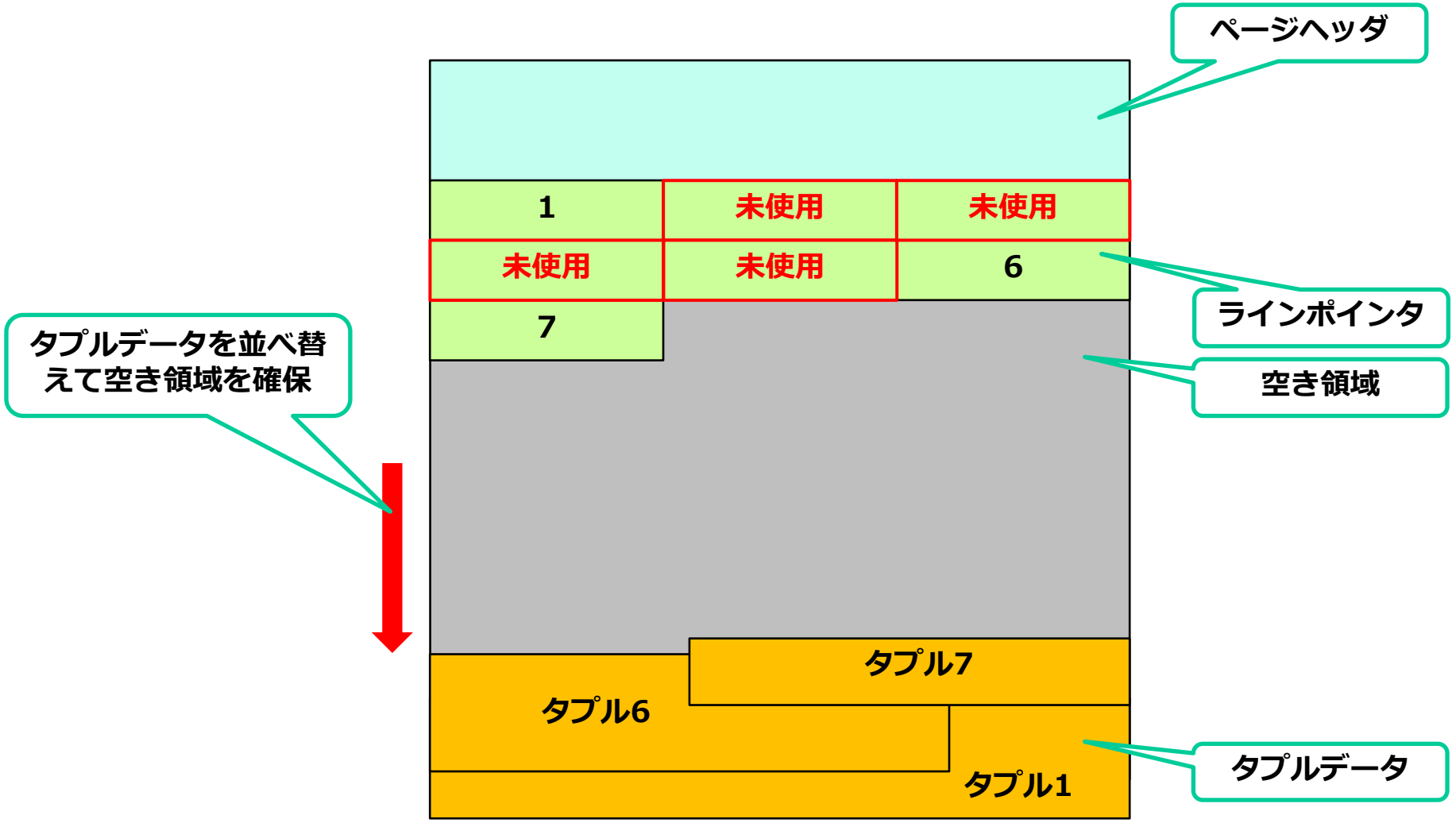
#OSS-DB



コンカレントVACUUMの処理（3/3）

■不要領域を回収し、ブロック内で並び替え③

#OSS-DB



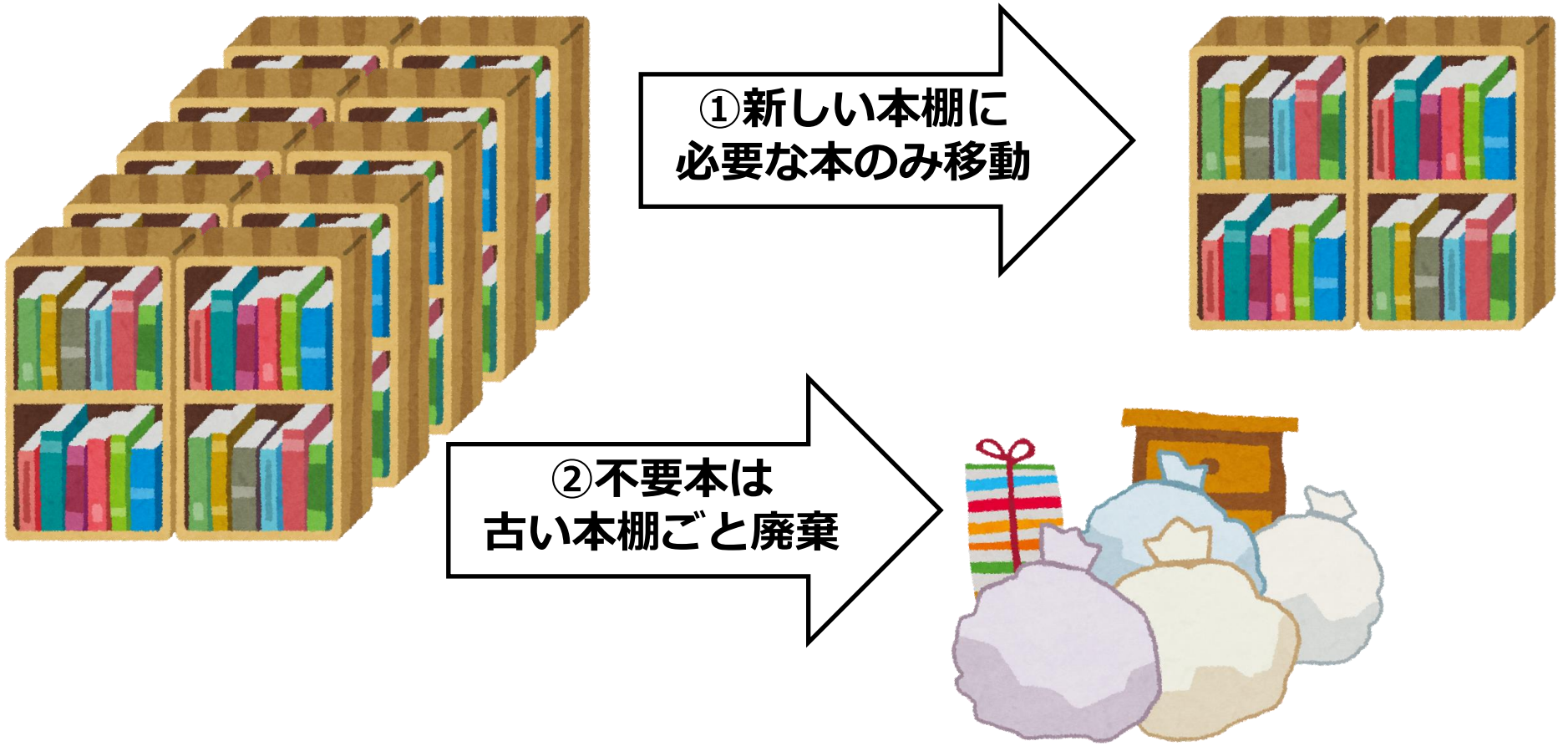
VACUUM FULLの処理イメージ

■大掃除で、本棚丸ごとの整理

#OSS-DB

① 本棚を新しく用意し、必要な本のみ移動

② 不要な本は、古い本棚ごと廃棄

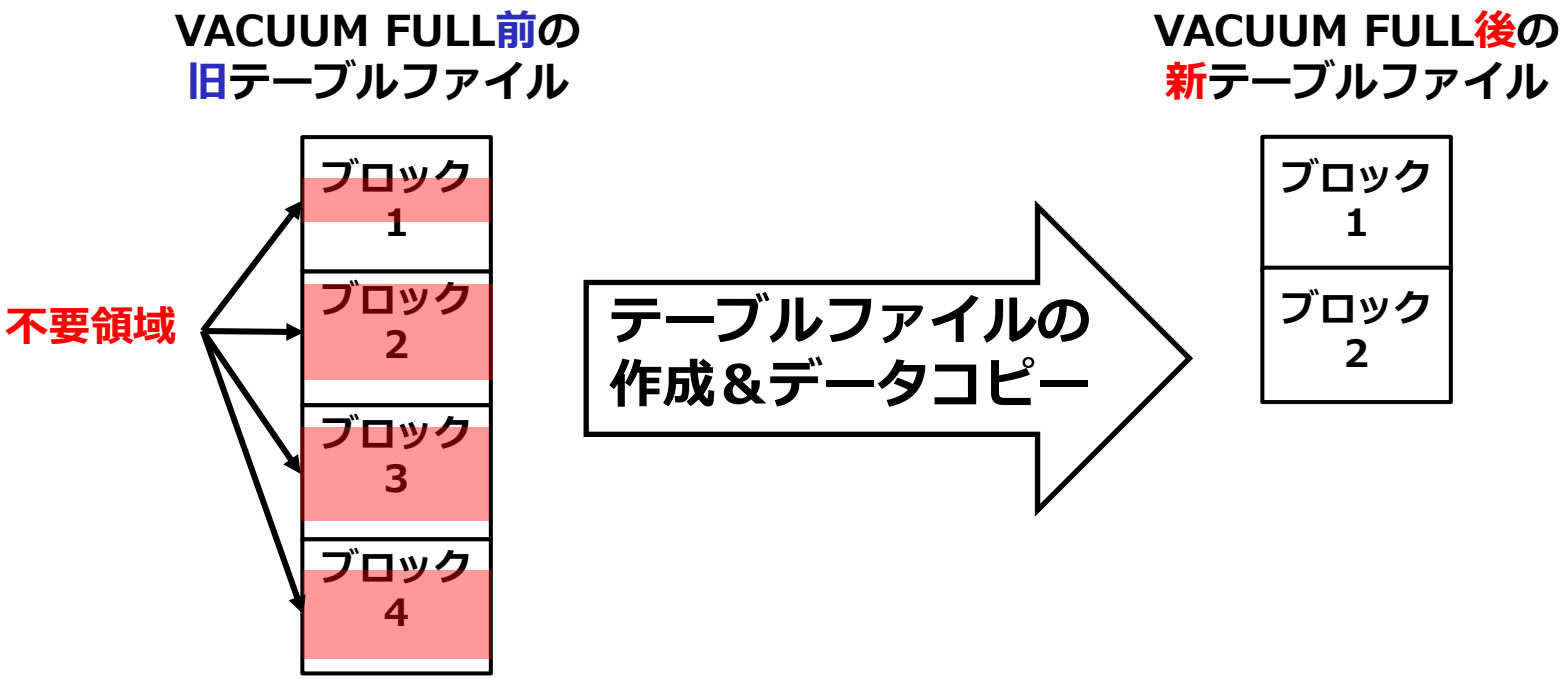


VACUUM FULLの処理

■**テーブルファイルを新規作成し、有効タプルのみをコピーした後、旧テーブルファイルを削除**

#OSS-DB

- 実行中は対象テーブルに**排他ロック**を取得
- 一般的にブロック数が削減されるため、VACUUM FULL 後はファイルサイズが小さくなる



不要領域の調査方法

#OSS-DB

pgstattuple:テーブルの統計情報表示関数

- 「CREATE EXTENSION pgstattuple;」でインストール
- 一般的に、dead_tuple_percent が20%以上ならVACUUMすべき
  例）下記のテーブルでは不要タプルが約60%を占めるため、VACUUMすべき

```
SELECT * FROM pgstattuple('pgbench_accounts');
-[ RECORD 1 ]------+-----------
table_len          | 1342955520     テーブルの物理的な大きさ（バイト）
tuple_count        | 10000          有効タプル数
tuple_len          | 1210000        有効タプルの合計物理長（バイト）
tuple_percent      | 0.09           有効タプルの割合　tupple_len/table_len * 100
dead_tuple_count   | 6594919        不要なタプル数（＝VACUUMの対象タプル）
dead_tuple_len     | 797985199      不要タプルの合計物理長（バイト）
dead_tuple_percent | 59.42          不要タプルの割合
free_space         | 452935708      空き領域の合計物理長（バイト）
free_percent       | 33.73          空き領域の割合
```

不要領域の割合を確認

※ pgstattuple はOSS-DB Gold の出題範囲

VACUUMの実行と運用

#OSS-DB

■VACUUM実行方法

① 自動VACUUM（後述）

② 手動でのVACUUM実行

■VACUUMの運用

- 基本的に自動VACUUM任せで問題ない
- 自動VACUUMを使用しない場合：1日1回全データベースを対象に手動でのVACUUMを実行を推奨
- 適切に自動VACUUMを実施していれば、VACUUM FULLは不要
- 古いデータの大量削除後や、性能トラブル等で必要な場合にだけVACUUM FULLを使用

VACUUMの手動実行1

#OSS-DB

■手動でのVACUUM実行

①SQLの「VACUUM」文

②または「vacuumdb」コマンド

■VACUUM文

- デフォルトのVACUUMの対象はデータベース内全てのテーブル
- 特定のテーブルだけを対象にすることもできる

主なオプション	処理内容
FULL	VACUUM FULL を実行
ANALYZE	VACUUM 後にANALYZE も実行
VERBOSE	処理の詳細な内容を出力
FREEZE	フリーズ処理を実行 ※付録で説明

```
例)    =# VACUUM VERBOSE table1;
```

VACUUMの手動実行2

#OSS-DB

vacuumdbコマンド

- VACUUM文のラッパーで、実際に行われる処理に違いはない
- psql と同様の接続オプションを使用できる
  （-dで接続先データベース、-pでポート番号の指定など）
- それ以外のオプションについては、下記表を参照

主なオプション	処理内容
-a　（--all）	全てのデータベースに対して実行
-t　テーブル名	対象テーブルを指定
-f	VACUUM FULL を実行
-F	VACUUM FREEZE を実行
-z	VACUUM 後にANALYZE も実行
-Z	ANALYZE のみ実行
-v	処理の詳細な内容を出力

例）　`$ vacuumdb -v -d test-db -t table1`

手動VACUUM実行の注意点

#OSS-DB

ピーク時のVACUUM 実行は避ける

- VACUUM による不要領域の回収は、ディスクI/Oに負荷を与える
- 巨大なテーブルでは回収されるタプル数も多く、負荷も大きいため特に注意

→ 性能トラブル発生防止のため、ピーク時の手動VACUUMは避けるべき

トランザクション内でのVACUUM実行は不可

- 下記のエラーで実行不可

例）
```
=# BEGIN;
=# VACUUM table1;
ERROR: VACUUM cannot run inside a transaction block
```

- バッチ処理内で手動VACUUMを実行する際は、トランザクション外で実行

デモ環境の構築

#OSS-DB

- ホストOS：CentOS 7.8
- PostgreSQL本体とcontrib モジュールをyumでインストール済み
  - 自動作成される"postgres"ユーザでPATH環境変数などを設定済み
- PostgreSQL バージョン：11.8
  - バージョン9.1以上ならどのバージョンでも可
- データベースクラスタを作成し、PostgreSQLを起動

```
$ initdb --no-locale --encoding=UTF8
$ pg_ctl start
```

- demoデータベース作成

```
$ createdb demo
```

- カレントディレクトリにデモ用のSQLファイルを配置していることを確認

```
$ ls *.sql
```

VACUUMの実行デモ1

#OSS-DB

VACUUMデモ用のdemo1テーブル

- 5タプルを INSERT したテーブル

demo_id	memo
1	OSS-DB_1
2	OSS-DB_2
3	OSS-DB_3
4	OSS-DB_4
5	OSS-DB_5

psql でdemoデータベースに接続し、demo1テーブルを作成

```
$ psql demo
=# \i create_demo1.sql;
```

※メタコマンド \i
→ ファイルからコマンドを読み実行

demo1テーブルの内容とpgstattupleの結果を表示

```
=# \i stat_demo1.sql;
```

VACUUMの実行デモ2

#OSS-DB

DELETE 文で1タプル削除した後の不要領域を確認

```
=# DELETE FROM demo1 WHERE demo_id = 1;
=# \i stat_demo1.sql;
```

UPDATE 文で1タプル更新した後の不要領域を確認

```
=# UPDATE demo1 SET memo = 'Silver_5' WHERE demo_id = 5;
=# \i stat_demo1.sql;
```

INSERT 文で1タプル挿入した後の不要領域を確認

```
=# INSERT into demo1 (memo) VALUES ('OSS-DB_6');
=# \i stat_demo1.sql;
```

VACUUM 文で不要領域を回収した後の不要領域を確認

```
=# VACUUM VERBOSE demo1 ;
=# \i stat_demo1.sql;
```

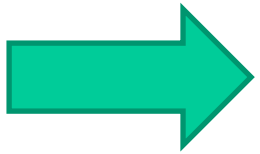
VACUUM FULL の実行デモ1

#OSS-DB

VACUUM FULL デモ用のdemo2テーブル

- 50万タプルを INSERTし、49万9995タプルを DELETE した後にVACUUM

demo_id	memo
100000	OSS-DB_100000
200000	OSS-DB_200000
300000	OSS-DB_300000
400000	OSS-DB_400000
500000	OSS-DB_500000



demo2テーブルのイメージ
（49万9995タプル分の空き領域）

demo2テーブルを作成

```
=# \i create_demo2.sql;
```

demo2テーブルの内容とpgstattupleの結果を表示

```
=# \i stat_demo2.sql;
```

VACUUM FULLの実行デモ2

#OSS-DB

■UPDATE 文で全タプルを更新した後の不要領域を確認

```
=# UPDATE demo2 SET memo = 'Silver_'|| demo_id::text ;
=# \i stat_demo2.sql;
```

■コンカレントVACUUM で不要領域を回収した後の不要領域を確認

```
=# VACUUM VERBOSE demo2 ;
=# \i stat_demo2.sql;
```

■もう一度UPDATE 文で全タプルを更新した後の不要領域を確認

```
=# UPDATE demo2 SET memo = 'Gold_'|| demo_id::text ;
=# \i stat_demo2.sql;
```

■VACUUM FULL でテーブルファイルを再作成した後の不要領域を確認

```
=# VACUUM FULL VERBOSE demo2 ;
=# \i stat_demo2.sql;
```

例題を解いてみよう1

#OSS-DB

次の説明のうち、正しいものをすべて選びなさい。

A. VACUUM はテーブルの排他的ロックを取得する。
B. VACUUM ANALYZE はテーブルの排他的ロックを取得する。
C. VACUUM FULL はテーブルの排他的ロックを取得する。
D. ANALYZE はテーブルの排他的ロックを取得する。
E. 上記はいずれも誤りである。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

・OSS-DB　公式サイトの「Silverの例題解説」より引用
https://oss-db.jp/sample/silver_management_05/87_180709

例題を解いてみよう1

#OSS-DB

次の説明のうち、正しいものをすべて選びなさい。

A. VACUUM はテーブルの排他的ロックを取得する。
B. VACUUM ANALYZE はテーブルの排他的ロックを取得する。
C. VACUUM FULL はテーブルの排他的ロックを取得する。
D. ANALYZE はテーブルの排他的ロックを取得する。
E. 上記はいずれも誤りである。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

・OSS-DB　公式サイトの「Silverの例題解説」より引用
https://oss-db.jp/sample/silver_management_05/87_180709

目次

#OSS-DB

ANALYZEとテーブル統計情報

■**ANALYZE** ：テーブル統計情報の更新処理

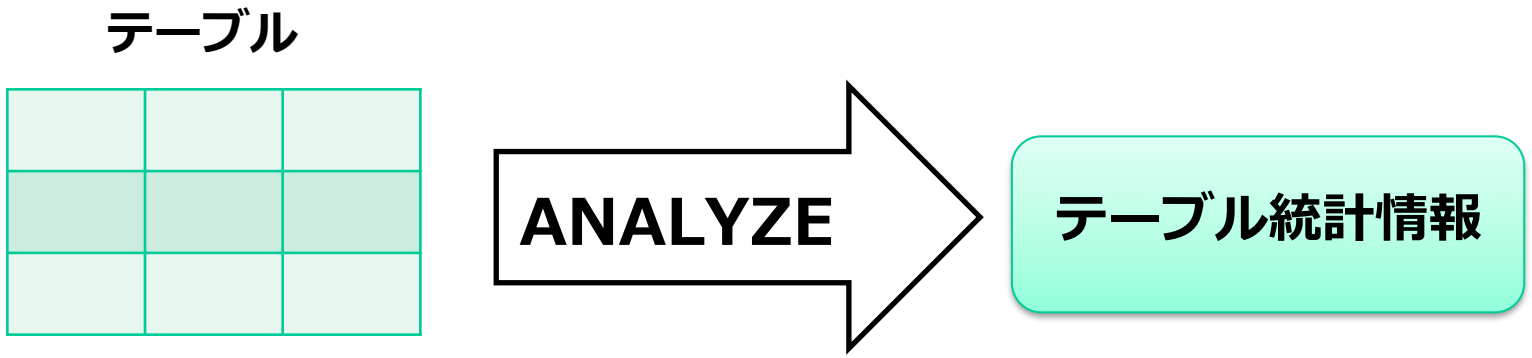
#OSS-DB

■**テーブル統計情報：**「テーブルにどのようなデータが格納されているか？」という情報が保存されたシステムカタログ

■テーブル統計情報が実際のテーブルデータと異なっていると

→　SQLの実行効率が下がり、PostgreSQLの性能が低下

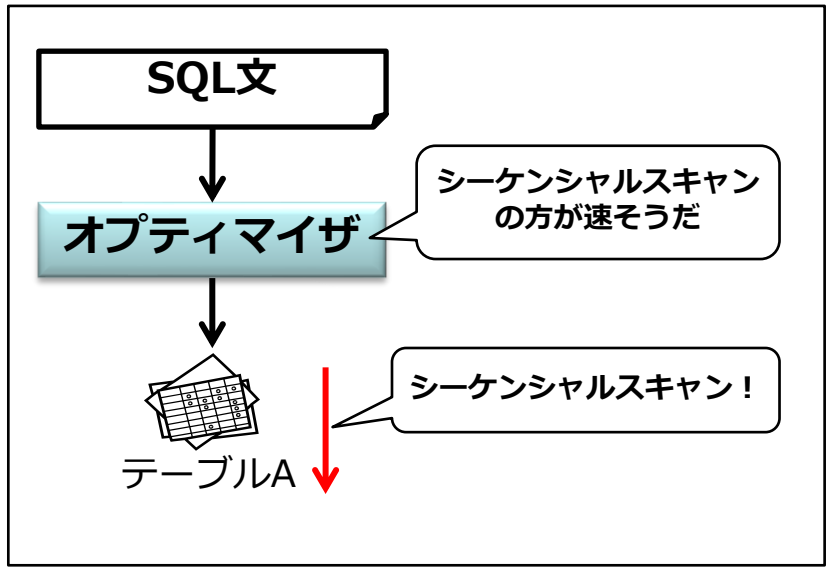
- データの更新や削除に応じて、テーブル統計情報の更新が必要

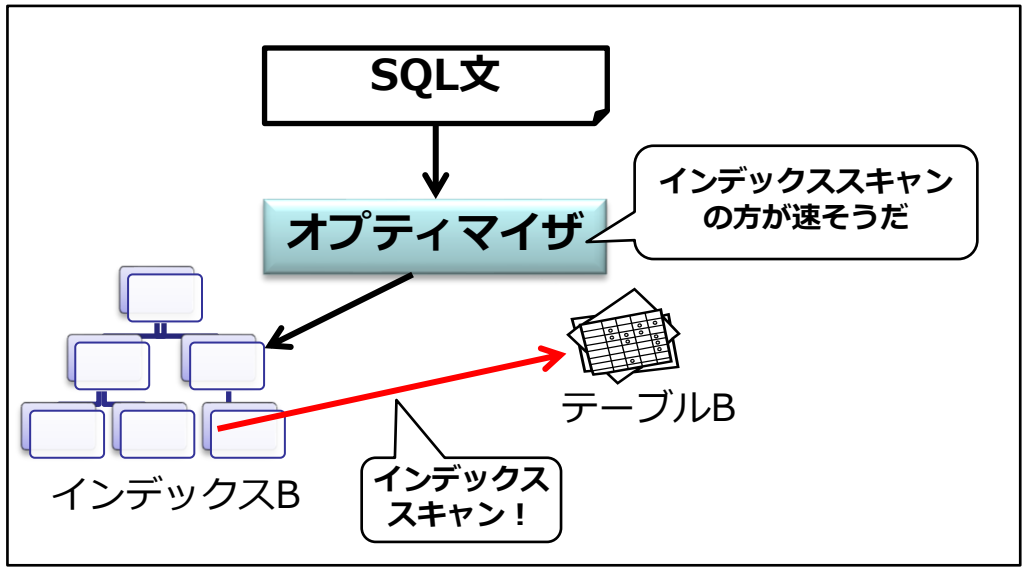


実行計画

#OSS-DB

- SQLは問い合わせ言語
  - SQL文にはデータの「処理内容」のみを記述
  - データの「処理方法」は記述しない
    （処理方法：データへのアクセス方法、結合順序、結合方法etc…）
- 「データをどう処理するか？」（＝実行計画）はDBMSに一任されている
- DBMSのオプティマイザが複数の実行計画を生成し、その中から最も効率の良いものを自動選択





オプティマイザの判断基準

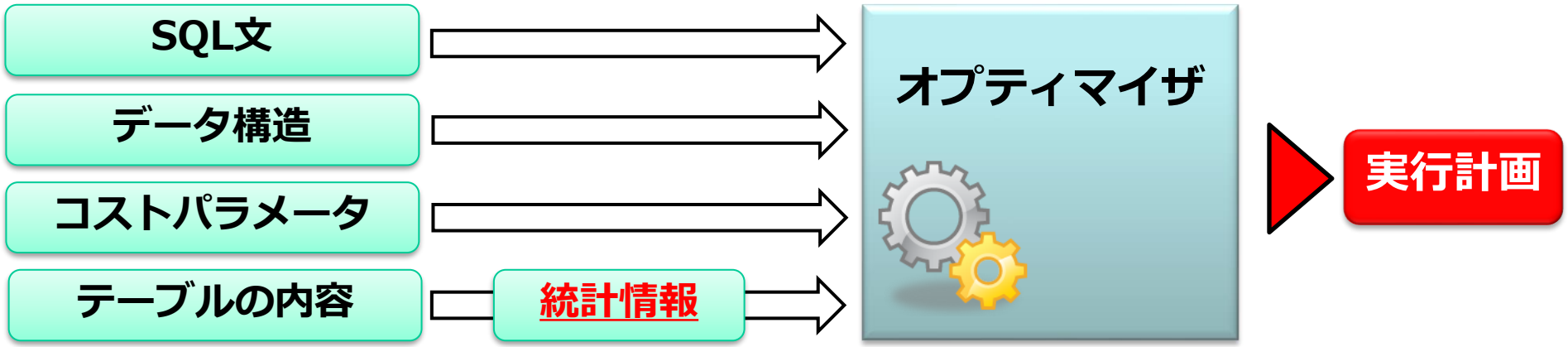
#OSS-DB

■オプティマイザ

複数の実行計画を生成し、その中からSQL文を最も効率よく実行する方法（= 最も低い”コスト”の方法）を判断・選択

■オプティマイザが”コスト”を計算する基準となる情報

- SQL文自体
- データ構造（テーブル構成、インデックス構成など）
- コストパラメータ（コスト決定のためのpostgresql.confのパラメータ）
- テーブルの内容（タプル数やどの値が多いかなど＝テーブル統計情報）



統計情報とは

#OSS-DB

■**テーブル統計情報**

- テーブルのタプル数やカラムごとの値の特徴（頻出値やnullの割合など）を収集・集計
- テーブルの全スキャンではなくランダムサンプリングでデータ収集
- 内容が同じテーブルでも収集結果は常に同じとは限らない

■**テーブル統計情報の精度が悪いと**

→　適切な実行計画を作成できない

→　SQLの**実行性能が劣化**

■テーブルが更新されて値の分布状況が変更された場合、統計情報の更新（＝ANALYZE）も必要

ANALYZEの実行と運用

#OSS-DB

ANALYZE実行方法

① 自動VACUUM（後述）
自動VACUUM 用のプロセスがANALYZEも自動実行

② 手動でのANALYZE 実行

ANALYZEの運用

- VACUUMと同様に、基本的に自動VACUUM任せで問題ない
- 自動VACUUMを使用しない場合：1日1回全データベースを対象に手動でのANALYZEを実行を推奨

ANALYZEの手動実行

#OSS-DB

■手動でのANALYZE実行

① SQLの「ANALYZE」文
② または「vacuumdb」コマンド

■ANALYZE文

- デフォルトの対象はデータベース内全てのテーブル
- 特定のテーブルのみ、特定のテーブルの特定のカラムのみANALYZEも可能

```
例）　=# ANALYZE VERBOSE table1 (column1);
```

■vacuumdbコマンド

- 「-z」または「-Z」オプションでANALYZEを実行

```
例）　$ vacuumdb -Z -v -d test-db -t 'table1(column1)'
```

ANALYZEのよくある質問1

■**Q1**：巨大なテーブルでは、ANALYZEの処理時間は長くなる？

#OSS-DB

■**A1**：ANALYZEのデータ収集はランダムサンプリングのため、
巨大なテーブルでも処理時間は長くなりません

■**Q2**：ランダムサンプリングする量は調整できるの？

■**A2**：**default_statistics_target** パラメータで調整できます

- 設定値×300タプルがサンプリングの対象。デフォルト値は100
- 小さすぎる　→　統計情報の精度が悪化　→　不適切な実行計画の原因
- 大きすぎる　→　統計情報の精度が高くなるが、ANALYZEに時間がかかる
- ALTER 文でテーブルごと、テーブルのカラムごとに値を設定可能

例）　`=# ALTER TABLE table1 ALTER column1 SET STATISTICS 500;`

ANALYZEのよくある質問2

#OSS-DB

- **Q3**：テーブル統計情報の保存先は？
- **A3**：PostgreSQLのシステムカタログに保存されます
  - システムカタログ：PostgreSQLのシステム情報や稼働状況、テーブルのメタ情報などが保存された特殊なテーブル群
  - システムカタログの
    pg_class テーブル
    pg_statistic テーブル
    にテーブル統計情報を保存
  - 具体的な内容については、OSS-DB Gold 試験の出題範囲のため省略

- **Q4**：ANALYZE 処理では、対象テーブルにロックをかけるの？
- **A4**：コンカレントVACUUMと同じ「SHARE UPDATE EXCLUSIVE」というロックを、かけます
  - 対象テーブルで、SELECTとINSRT、UPDATE、DELETE可能

ANALYZEのよくある質問3

■**Q5**：Oracle Database のように、テーブル統計情報の固定化（ロック）はできないの？

#OSS-DB

※テーブル統計情報の固定化：テーブル統計情報の変動による実行計画の変化を避けるチューニング手法

■**A5**：PostgreSQL自体には固定化の機能はありませんが、**pg_dbms_stats** という外部ツールを導入すれば、固定化やエクスポート・インポートが可能です

- **pg_dbmas_stat**
  https://pgdbmsstats.osdn.jp/

例題を解いてみよう2

#OSS-DB

ANALYZEについて述べたものとして、適切なものを2つ選びなさい。

A. テーブルがアクセスされる頻度、更新される頻度といった統計情報を取得する。
B. デフォルトではテーブル全体を解析するため、巨大なテーブルのANALYZEには時間がかかる。
C. 実行時のパラメータにより、データベース全体、データベース内の特定のテーブルのみ、特定のテーブルの特定の列のみ、などANALYZEの対象を制御できる。
D. 実行時のパラメータにより、統計情報の取得の目標値を変更できるので、これにより、統計情報の正確さやANALYZEに要する時間を制御できる。
E. 自動バキュームの実行時に自動的に実行される。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

・OSS-DB　公式サイトの「Silverの例題解説」より引用
https://oss-db.jp/sample/silver_management_04/66_160808

例題を解いてみよう2

#OSS-DB

ANALYZEについて述べたものとして、適切なものを2つ選びなさい。

A. テーブルがアクセスされる頻度、更新される頻度といった統計情報を取得する。
B. デフォルトではテーブル全体を解析するため、巨大なテーブルのANALYZEには時間がかかる。
C. 実行時のパラメータにより、データベース全体、データベース内の特定のテーブルのみ、特定のテーブルの特定の列のみ、などANALYZEの対象を制御できる。
D. 実行時のパラメータにより、統計情報の取得の目標値を変更できるので、これにより、統計情報の正確さやANALYZEに要する時間を制御できる。
E. 自動バキュームの実行時に自動的に実行される。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

・OSS-DB　公式サイトの「Silverの例題解説」より引用
https://oss-db.jp/sample/silver_management_04/66_160808

目次

1.VACUUM

2.ANALYZE

3.自動VACUUM

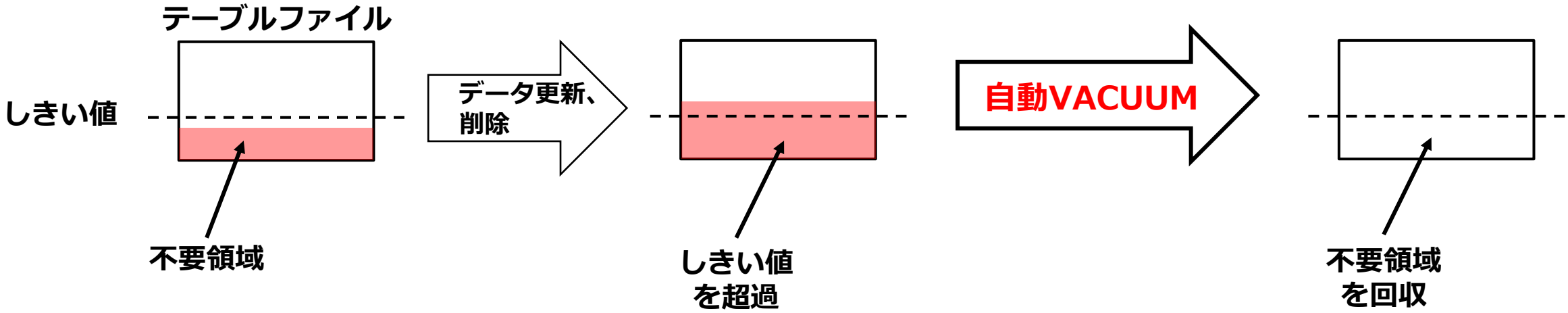
#OSS-DB

自動VACUUM：自動VACUUM処理

■自動VACUUM：VACUUMの自動実行機能

#OSS-DB

- 「本棚全体の不要本の冊数を定期的にチェックし、しきい値を超えたら不要本の整理を開始する」イメージ
- autovaccum lancher プロセスが各テーブル全体の不要領域のタプル数を定期的（デフォルト1分間隔）にチェック
- 不要領域がしきい値を超過すると、autovacuum worker プロセスを起動し、コンカレントVACUUMを自動実行

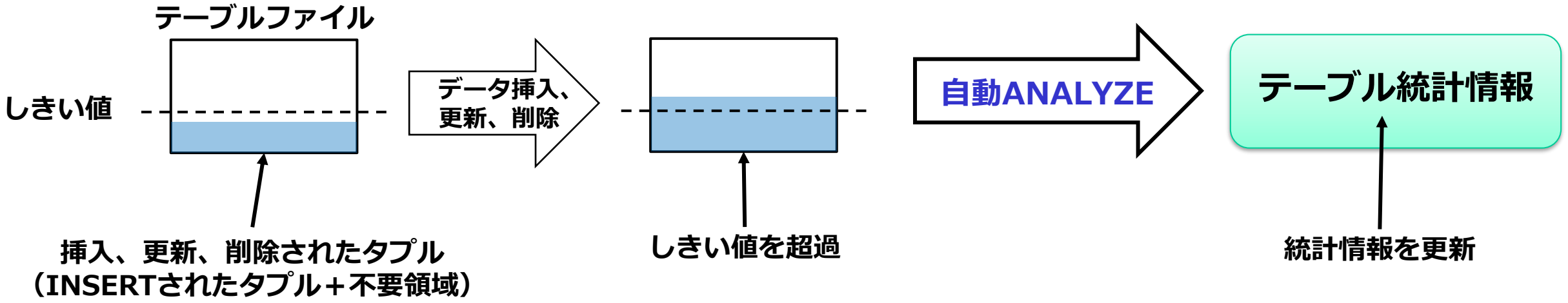


自動VACUUM：ANALYZE処理

#OSS-DB

自動ANALYZE：ANALYZE の自動実行機能

- autovaccum lancher プロセスが各テーブルで挿入、更新、削除されたタプル数を定期的（デフォルト1分間隔）にチェック
- しきい値を超過すると、autovacuum workerプロセスを起動し、ANALYZEを自動実行



自動VACUUMの主な設定パラメータ（1/3）

自動VACUUM プロセスの制御パラメータ

#OSS-DB

パラメータ名	パラメータの意味	デフォルト値
autovacuum	自動VACUUMを使用するかどうかの設定	on
autovacuum_naptime	autovacuum lancher プロセスがテーブルをチェックする周期	1min
autovacuum_max_workers	同時に実行できるautovacuum worker プロセスの最大数 ※	3
log_autovacuum_min_duration	指定した時間以上に実行時間がかかった自動VACUUMをログに記録する	-1 （記録しない）

※PostgreSQL のログに下記内容が出力されたら、値を増やすことを検討

```
LOG: maximum number of autovacuum workers reached
HINT: Consider increasing autovacuum_max_workers (currently 3).
```

自動VACUUMの主な設定パラメータ（2/3）

自動VACUUM しきい値調整パラメータ

#OSS-DB

パラメータ名	パラメータの意味	デフォルト値
autovacuum_vacuum_threshold	自動VACUUMのしきい値パラメータ1 テーブル内の不要領域の最小数を指定	50
autovacuum_vacuum_scale_factor	自動VACUUMのしきい値パラメータ2 テーブル内の不要領域の割合を指定	0.2

- 自動VACUUM しきい値の計算式

```
autovacuum_vacuum_threshold  + autovacuum_vacuum_scale_factor ×（テーブルの有効タプル数）
```

- 例）テーブルの有効タプル数が1万の場合のデフォルトしきい値
  50 + 0.2 * 10,000 = 2,050
  → 不要領域が2,050 タプル以上で自動VACUUM を実行

自動VACUUMの主な設定パラメータ（3/3）

自動ANALYZE しきい値調整パラメータ

#OSS-DB

パラメータ名	パラメータの意味	デフォルト値
autovacuum_analyze_threshold	自動ANALYZEのしきい値パラメータ1 テーブル内の最小数を指定	50
autovacuum_analyze_scale_factor	自動ANALYZEのしきい値パラメータ2 テーブル内の割合を指定	0.1

- 自動ANALYZE しきい値の計算式

```
autovacuum_analyze_threshold  + autovacuum_analyze_scale_factor  ×（テーブルの有効タプル数）
```

- 例）テーブルの有効タプル数が1万の場合のデフォルトしきい値
  50 + 0.1 * 10,000 = 1,050
  → 挿入・更新・削除されたタプルが1,050 タプル以上で自動ANALYZE を実行

自動VACUUM運用の注意点1

■自動VACUUMは、データベースの稼働状況を考慮しない

#OSS-DB

- 自動VACUUM実施の判断可否基準は「不要領域の割合」のみ
- 「特定の時間帯は自動VACUUMを実行しない」という設定も不可

→ I/Oピーク時に自動VACUUMが実行され、トラブルの原因になることもある

例）週次バックアップ取得中に長時間の自動VACUUMが実行された
→ IOPS の限界を超過し、規定時間内にバックアップを終了できなかった

■対応方法1：テーブル単位で自動VACUUMを無効化

- 「ALTER TABLE」文で、特定のテーブルを自動VACUUMの対象から外し、ピーク時間帯を避けて手動VACUUMで運用

```
例）    =# ALTER TABLE table1 SET (autovacuum_enabled = off);
```

■対応方法2：遅延VACUUMを設定

- VACUUMを"一時停止しながらゆっくり実行"することで、I/Oを軽減する機能
- デメリット：VACUUM完了までの時間が長くなる

自動VACUUM運用の注意点2

#OSS-DB

■巨大なテーブルのしきい値は要調整

- 巨大なテーブルの場合、デフォルト設定では自動VACUUMが適切な頻度で実行されず、テーブル肥大化の原因になる

  例）1億タプルのテーブルのデフォルトVACUUMしきい値：2,000万50タプル

- 統計情報も長期間更新されず古いままなので、不適切な実行計画が選択されがち

■対応方法：テーブル単位で適切なしきい値に変更

- 「ALTER TABLE」文で、テーブルのVACUUMとANALYZEのしきい値を調整
- 下記の例では、有効タプル数の2%が更新・削除されると自動VACCUM実施
  有効タプル数の1%が挿入・更新・削除されると自動ANALYZE実施、

```
例）   =# ALTER TABLE table1 SET
          (autovacuum_vacuum_scale_factor = 0.02
           autovacuum_analyze_scale_factor = 0.01);
```

自動VACUUM運用の注意点3

#OSS-DB

■自動VACUUM 実行状況のログ確認

- デフォルトでは、自動VACUUMの情報はログに出力されない

■対応方法：log_autovacuum_min_duration パラメータを設定

- 指定した時間以上に実行時間がかかった自動VACUUMの情報をログに出力
- デフォルトは「-1」で全く出力しない
- 自動VACUUMの基準時間を事前に決めておき、設定する

例）1分以上 自動VACUUM処理に時間がかかったら、ログに出力
log_autovacuum_min_duration = 1min

例題を解いてみよう3

#OSS-DB

自動バキュームの説明として正しいものを3つ選びなさい。

A. 削除済みのタプル領域を回収する。
B. テーブルの統計情報を取得する。
C. 大量のタプルの挿入・削除・更新が行われたテーブルを検査する。
D. データベースの負荷が小さくなったときに自動的に起動する。
E. データベースの負荷が大きいときは起動が抑制される。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

・OSS-DB　公式サイトの「Silverの例題解説」より引用
https://oss-db.jp/sample/silver_management_05/88_180807

例題を解いてみよう3

#OSS-DB

自動バキュームの説明として正しいものを3つ選びなさい。

A. 削除済みのタプル領域を回収する。
B. テーブルの統計情報を取得する。
C. 大量のタプルの挿入・削除・更新が行われたテーブルを検査する。
D. データベースの負荷が小さくなったときに自動的に起動する。
E. データベースの負荷が大きいときは起動が抑制される。

※この例題は実際のOSS-DB技術者認定試験とは異なります。

・**OSS-DB　公式サイトの「Silverの例題解説」より引用**
https://oss-db.jp/sample/silver_management_05/88_180807

#OSS-DB

付録：VACUUMの役割2
XID周回問題の回避

※OSS-DB Gold の出題範囲

XIDとは

#OSS-DB

■XID：トランザクションID

- 符号なし32bitの整数：3~約42億9000万まで単調増加
  - 2^{32}を超えると3に戻る
- トランザクションごとに付与される一意な値で、トランザクションの順番を表す
- タプルヘッダにXIDを保持し、この値によってタプルの新旧（可視性）を判断

■現在のXID ＞＝　タプルのXID　→　過去：可視

- 過去のトランザクション=現在のトランザクションより前のトランザクションで追加・更新されたタプル
- 例）現在のXIDが100の場合、100とそれより前に開始された（XIDが99以下の）トランザクションで追加・更新されたタプルが過去＝可視のタプル

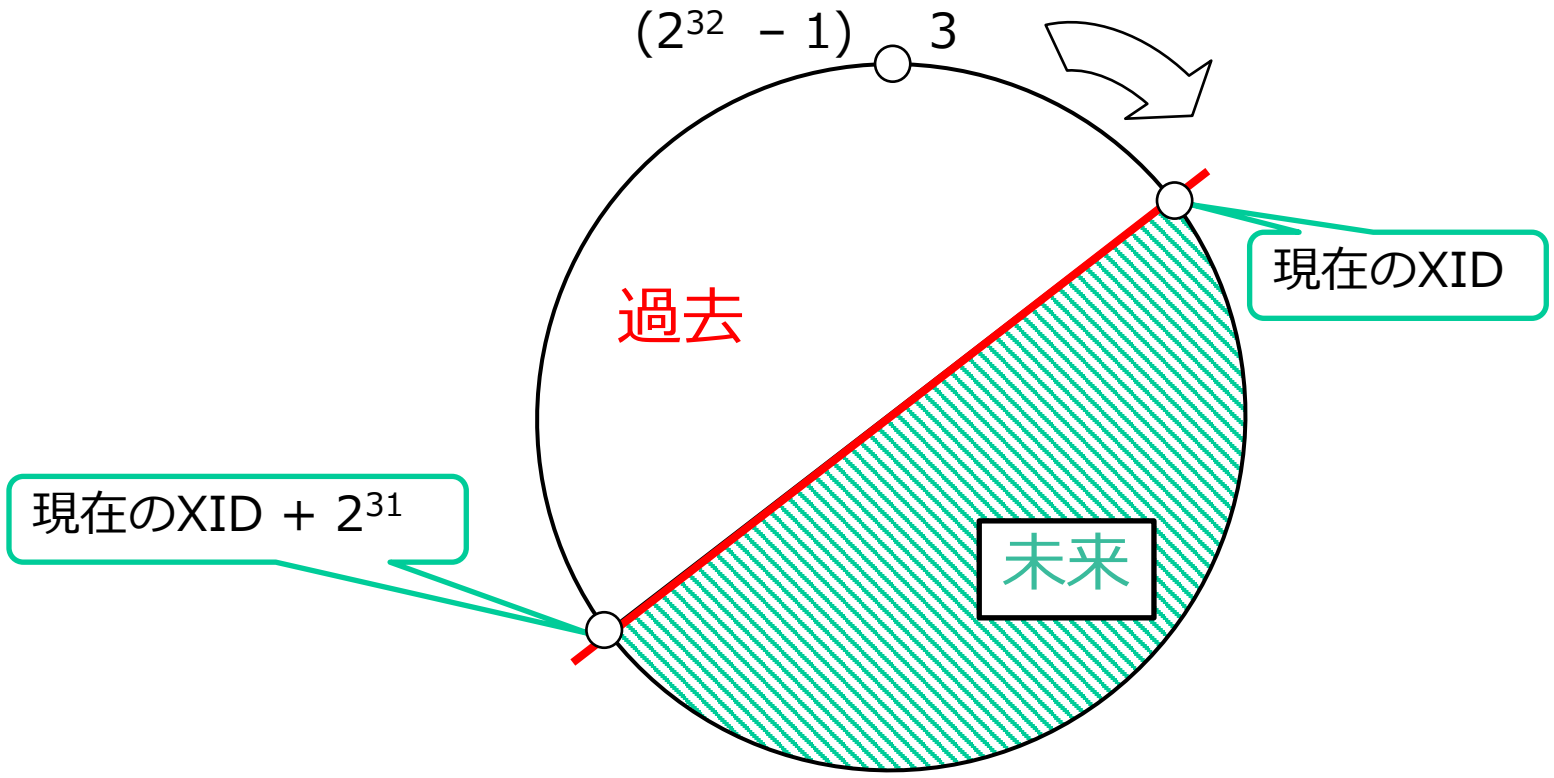
■現在のXID ＜ タプルのXID　→　未来：不可視

- 未来のトランザクション=現在のトランザクションより後のトランザクションで追加・更新されやタプル
- 例）現在のXIDが100の場合、100より後に開始されたトランザクション（XIDが101以上）で追加・更新されたタプルが未来＝不可視のタプル

XIDによる過去・未来識別

#OSS-DB

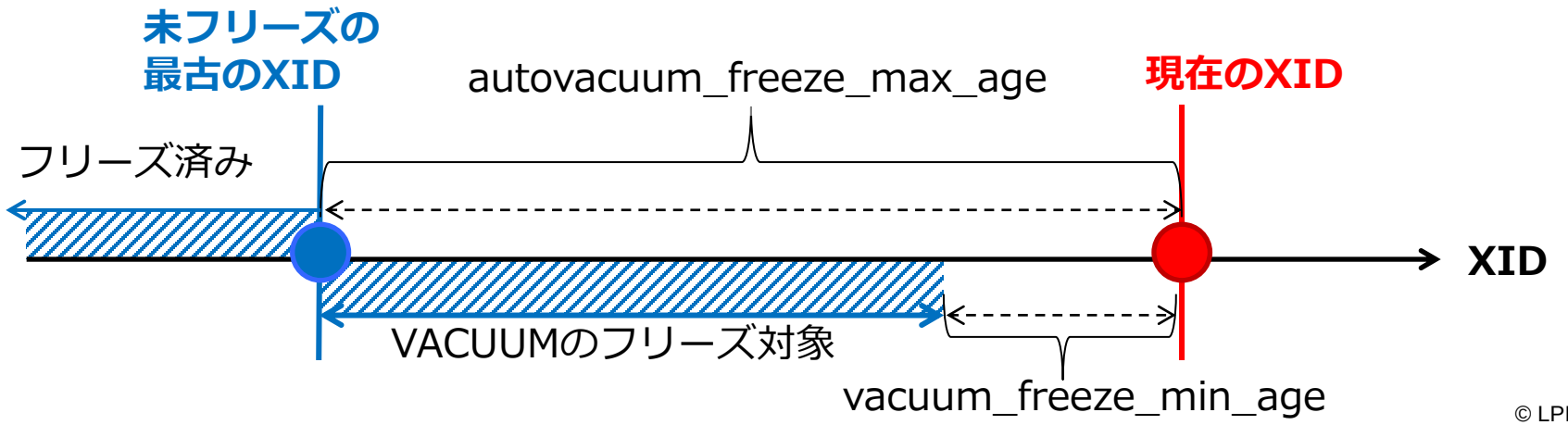
- **過去のXID**：現在のXID - 1　～ 現在のXID – (2^{31} - 1)
  または、現在のXID + (2^{31} + 1)　～　現在のXID + (2^{32} - 1)
- **未来のXID**：現在のXID + 1　～　現在のXID + 2^{31}
- 約21億4500万トランザクション経過で、過去と未来が入れ替わる
  →　**フリーズ処理**で対処



XIDのフリーズ処理

#OSS-DB

- フリーズ処理：十分に古い有効タプルにフリーズ用のフラグを立て、**どのXIDよりも必ず過去のタプル**と認識させる処理
- vacuum_freeze_table_age（デフォルト1.5億） のトランザクションが実行されると、自動VACUUMでフリーズ処理開始
  - 全く不要領域がないテーブルも、フリーズ処理で自動VACUUMされる
- 自動VACUUMが無効でも、フリーズしていない最古のXIDがautovacuum_freeze_max_age（デフォルト2億）よりも古くなったら、現在値よりvacuum_freeze_min_age（デフォルト5千万）以前のXIDを強制的にフリーズ
  - どこまでフリーズしたのかをpg_class.relfrozenxid に記録
  - フリーズしていない最古のXIDは pg_database.datfrozenxid に保存



手動でのフリーズ実行とフリーズ警告

#OSS-DB

■手動でのフリーズ実行

① SQL文の「VACUUM FREEZE」
② vacuumdb コマンドの「-F」オプション
③ VACUUM FULL（テーブルファイル再作成時にフリーズも実行）

■フリーズ未実行による警告とPostgreSQL強制停止

- 自動VACUUMでのフリーズが失敗し続けた場合、XID周回ポイントまで残り1000万トランザクションで、下記 警告が出力

```
WARNING:  database "xxx" must be vacuumed within 178919222 transactions
HINT:  To avoid a database shutdown, execute a database-wide VACUUM in "xxx".
```

- 警告を無視して残り100万トランザクションに達すると、PostgreSQL強制停止
  → シングルユーザモードで起動しなおし、「VACUUM FREEZE」を実行

```
ERROR:  database is not accepting commands to avoid wraparound data loss in database "xxx"
HINT:  Stop the postmaster and vacuum that database in single-user mode.
```