

# PostgreSQL を読むということ

## *Readings on the PostgreSQL*

NTT サイバースペース研究所  
OSS コンピューティングプロジェクト  
坂田 哲夫

1

## 何をお話するのか？

- PostgreSQL を読むことについてお話しします
  - 読み方と実例の両面から
- ソースを読むとは、どういうことか？
  - 読み方の心構え(みたいなもの)
  - 狙い・取り組み・結果のまとめ
- ログのしくみ
  - 実際のソースに即して、読み方・まとめ方をお話し  
す

2

# 何をお話するのか？

- ソースを読むとは、どういうことか？
  - 何がわかればよいのか
  - 何を指すべきなのか
- 実践の仕方はどのようにすべきか？
  - 取り組み方
  - 道具立て
  - 成果にまとめる
- ログのしくみを読む
  - もっとも複雑な XLogInsert を読んでみます

理論編

実践編

3

理論編

# ソースを読むこととは？

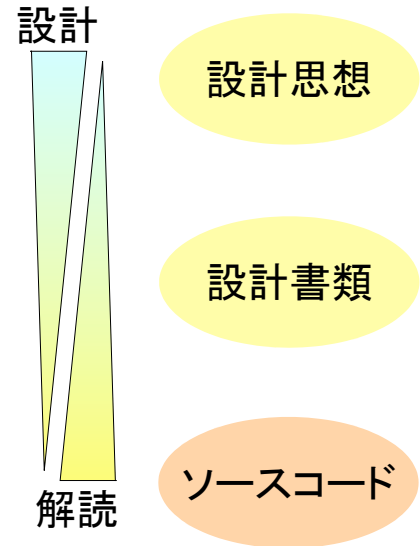
- 以下の点を明らかにしたい
  - ソースを読むとは、いったい何をすることなのか？
  - ソースを理解するとは、いったい何がわかることなのか？
- では、何がわかればいいのだろうか？
  - 設計資料相当の内容がわかればよいはずだ
    - PostgreSQL には設計資料がない
    - たとえ設計資料があったとしても、設計思想は分からない
  - 機能上・実装上の問題点について個々に理解したいはずだ
    - レベルに応じて資料化すべき

4

## 設計と解読のレベル

### • 設計の階層を振り替える

- 要件定義: システムの満たすべき外部からみた条件を確定する
- 機能設計: 要求を満たすための機能上の構造を確定する(データ構造(概要)、モジュール構造 etc.)
- 詳細設計: 機能を満たすべき各関数の仕様を確定する(API、振る舞い、詳細なデータ構造、アルゴリズム)



5

## 詳細設計を理解する

### • 関数単位での理解

- API: 入出力の意味、制限(範囲)
- 振る舞い: 機能(入力⇒出力)、副作用
- データ構造: 変数、構造体定義のレベルで
- アルゴリズム: 無名のアルゴリズムにも注意

6

# 機能設計を理解する

- モジュール単位での理解
  - 機能モジュール: DBMS の機能を充足するための関数群を特定する  
(e.g. タプルマネージャ、ログマネージャ)
    - 入出力・振る舞い・データ構造・アルゴリズムを特定
  - モジュール構成: 個別関数への機能分解
  - 外部仕様との関係を理解
    - モジュール内・モジュール間

7

## どのように取り組むか

- 事前準備
  - 読む対象を決める
    - 機能モジュール単位くらいが適当  
(1関数では小さすぎて却って意味が掴めない)
  - 機能の本質を知る
    - 教科書等でその機能の位置付け・内容を整理しておく  
⇔ソースコードから機能の本質を知るのは難しい
  - 時間を確保する
    - まとまったソースを読むには相当の時間を要する
    - 中断されない時間(1時間以上)が必要
  - ノートを準備する
    - 長期の作業を記録し、サポートする

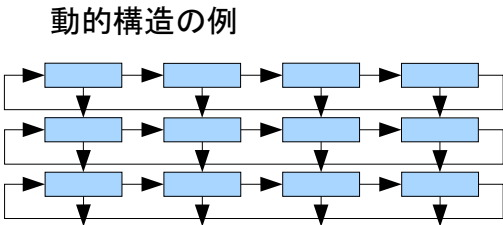
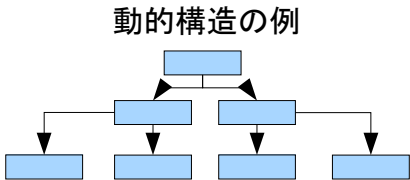
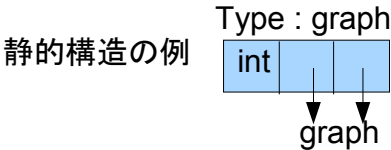
8

# どこから読むか

- アルゴリズム + データ構造 = プログラム (N.Wirth)
  - データ構造
    - 静的構造 \* : 宣言された枠組み (構造体)
    - 動的構造 \* : 複数の構造体のネットワーク (生成手順が手続きに内包されている)
  - アルゴリズムがデータ構造を生成し、データ構造がアルゴリズムを駆動する。
    - アルゴリズムの解読とデータ構造の解読が並行して進む
    - 一度に両方解読できないので、ノートを取りつつ進める
      - (\* : ここでの仮の名前)

# データ構造を読む

- 静的構造を調査する
  - 型 (構造体など) 宣言
    - 各種のツールがあれば効率的に進められる
- 動的構造を調査する
  - 実際に使われる際のデータ構造
    - 生成手順の調査が必要
      - コードを調査してノートを取る



# アルゴリズム

- 無名のアルゴリズムに注意せよ
  - 大事なものは、PostgreSQLに固有の処理、教科書が書かないアルゴリズム
- 隠れたアルゴリズムもある
  - アルゴリズムの断片が各所に埋め込まれている  
⇒積み上げてみて初めて姿を表す

11

むだ話

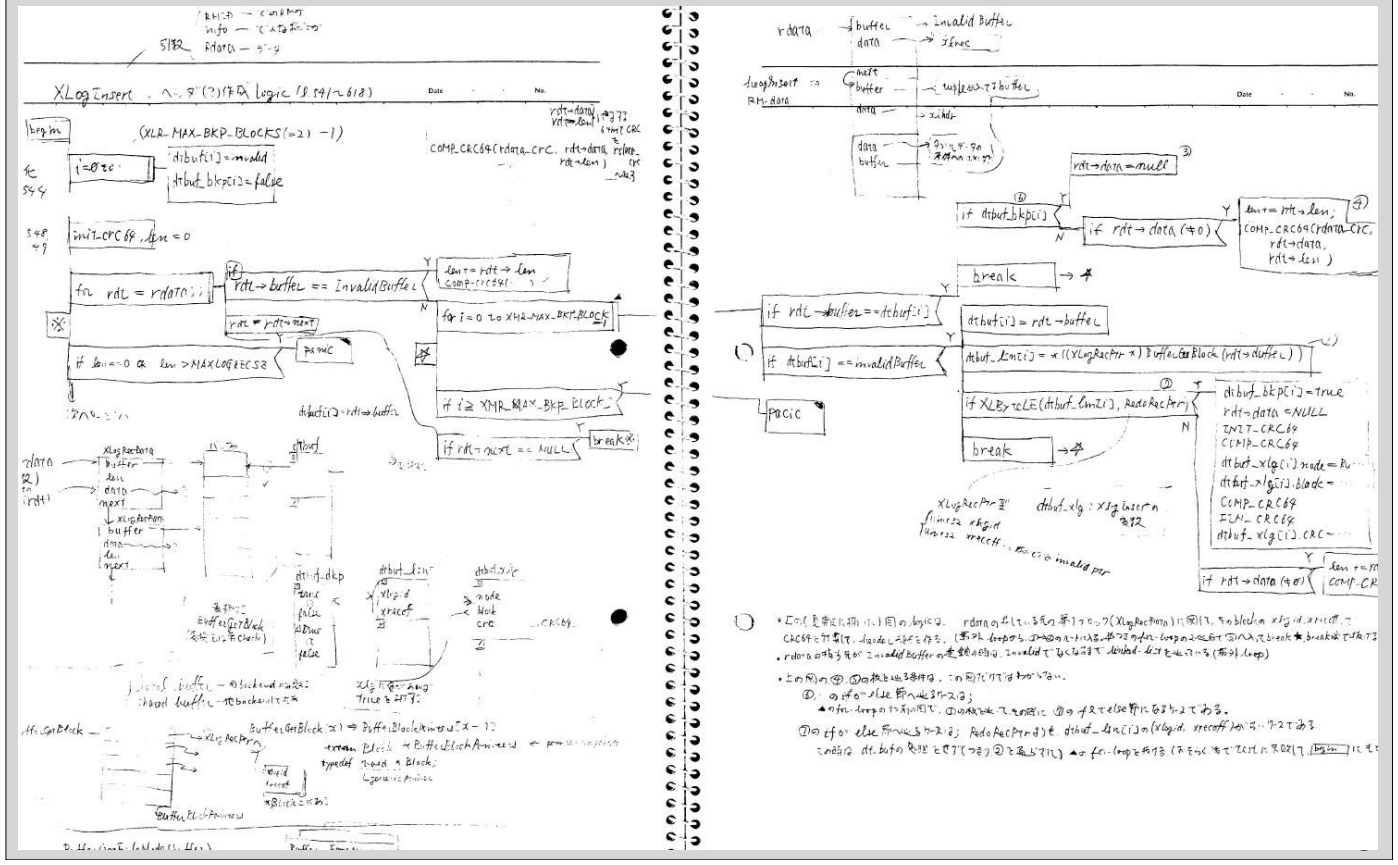
## 道具立て：ノートの三原則

- 第1条：紙切れに書かない
  - 紙切れは散逸する。もったいないのは紙ではない
- 第2条：詰めて書かない
  - 分かったことは追記する。転記は無駄、ミスのもと
- 第3条：専用の大判ノートに書く
  - 多くの頁を参照する際のロスになる
  - 1頁に書ける量が多いほうが良い⇒見開きを使う

12

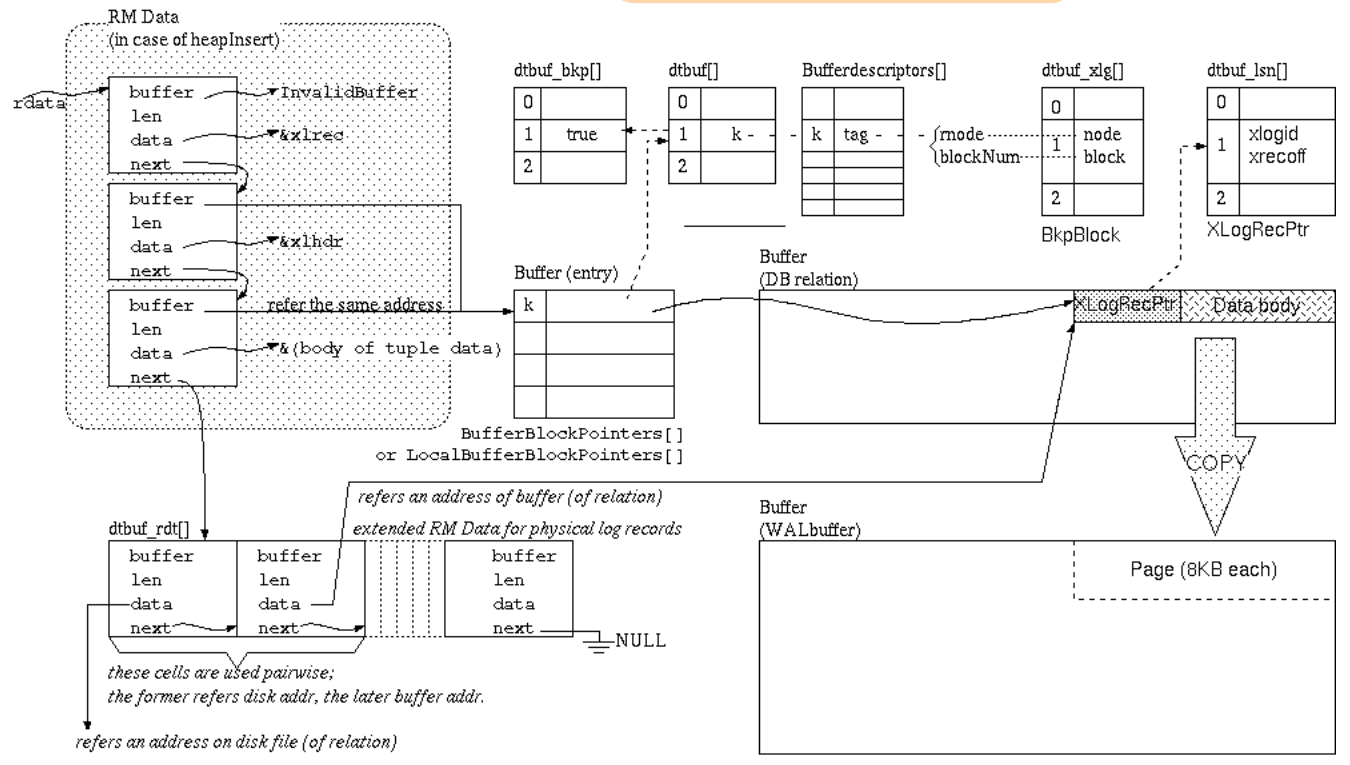
# ノートの例

XLogInsert の例  
 ・A4 見開き、0.3 のシャープペン  
 ・約 70 行分



# 整理したデータ構造の図

XlogInsert の主要データ構造



## 解読の視点

- 本質と偶有
  - 教科書に載っていないレベルの情報が重要
- めそな視点を持つ
  - ミクロ = ソースレベル、マクロ = ブロック図  
(両方とも既にあるだろう) ⇒ メソ (meso) は中間
  - 各種のマネージャ (log, lock, storage, buffer...) 単  
位にまとめる
  - マネージャ間 (inter)/ マネージャ内 (intra) の情報

15

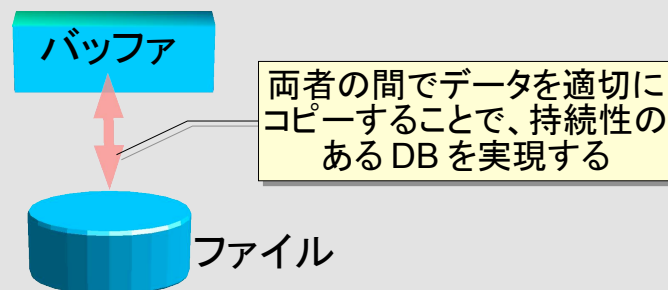
# Log の概要

PostgreSQL のログの説明をする前に  
DBMS 一般でのログの説明をします

16

# データベースのストレージ

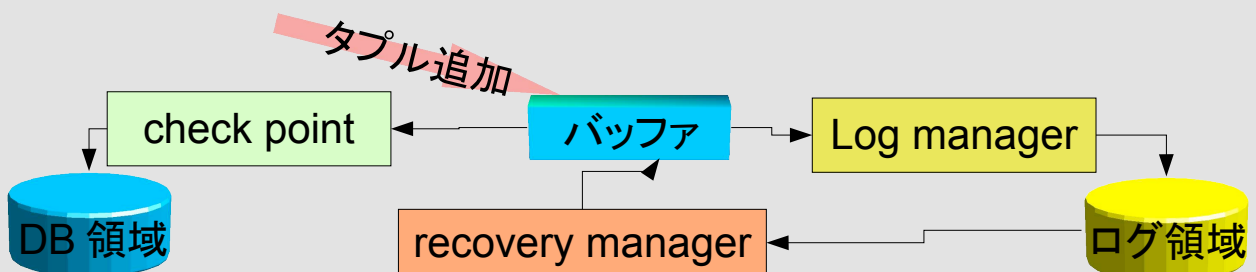
- データベースの利用するストレージ(記憶装置)についておさらいします
  - バッファ:
    - 主記憶上にある。ランダムアクセスできる
    - 電源が切れると内容を失う(揮発性)
  - ファイル:
    - I/O 経由でページ単位でアクセス
    - 電源が切れても内容は保存される(不揮発性)



17

# ストレージ管理のイメージ

- タプル追加のリクエスト
  - 追加するタプルの情報をログファイルへ書き込む
  - バッファ上の表の領域にタプルを追加
- コミット
  - ログをディスクへ書き込む
  - DB バッファとDB ファイルは不一致かもしれない
- チェックポイント
  - ダーティなDB バッファをファイルへ書き出す
  - DB バッファとDB ファイルは一致



18

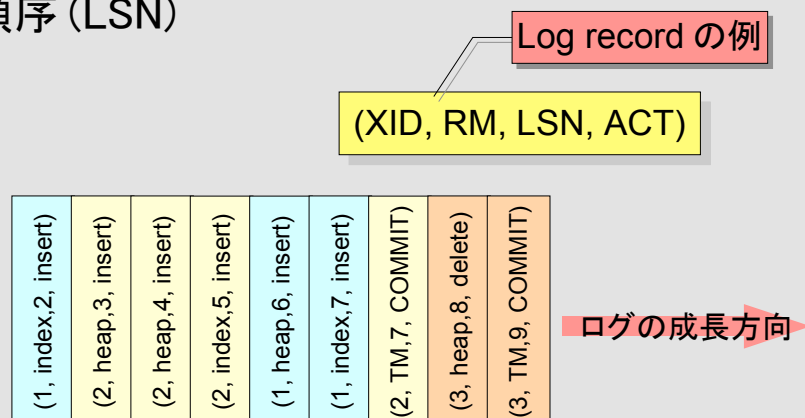
# ログとはどのようなモノか

- トランザクションの原子性と持続性
  - 原子性: All or Nothing
  - 持続性: コミットされた TrX の結果を確実に保存
- ログファイルを別に持っている意義
  - コミットの際に全ての結果を DB ファイルへ書き出す (原子性と持続性を満たすため)
    - コミット処理時にダウンするとどうなるか?
    - ディスクが処理のボトルネックになる
  - ディスク上の別の場所に一旦書き出す
    - システムダウンしてしても記録は残る ⇒ リカバリ可能
    - 固めて書き出す ⇒ ボトルネックの軽減

19

# ログとはどのようなモノか

- ログには何が入っているか - **更新の逐次的な記録**
  - どのトランザクションのログか (XID)
  - データベースに対する更新操作 (action)
    - 記録方式は、論理・物理の 2 に大別される
    - 1 操作 = 1 レコード (log record)
  - 更新を実施したサブシステム (RM)
  - それら操作の順序 (LSN)



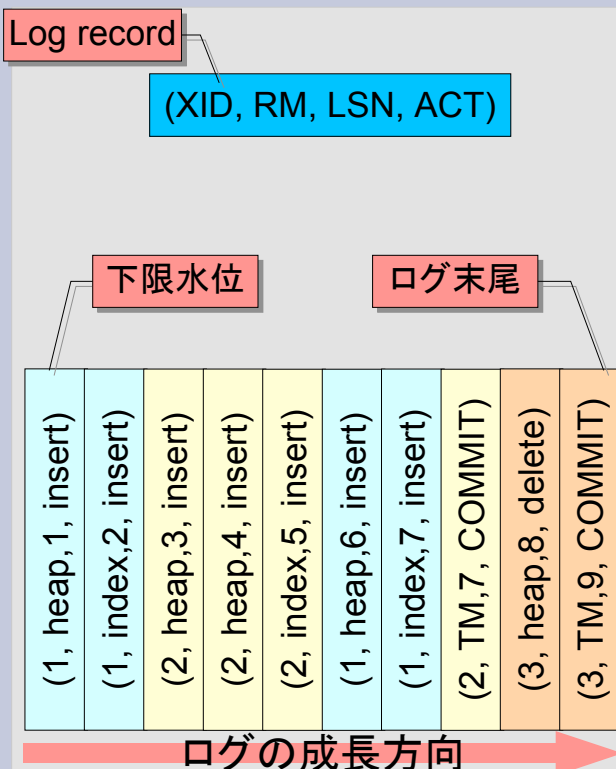
20

# ログによるリカバリ

- ログの記録を時系列にそって「再生」して、データベースに「反映」する
  - ログを順方向にスキャンする 'redo スキャン'
    - 下限水位から実行
    - コミット済み TrX のログのみ反映する
    - 必要となる時点までリカバリ(反映)する
  - このほか、ログを逆方向に見ていく 'undo スキャン' もあるが、PostgreSQL では実装されていないので割愛

21

# リカバリ動作のイメージ

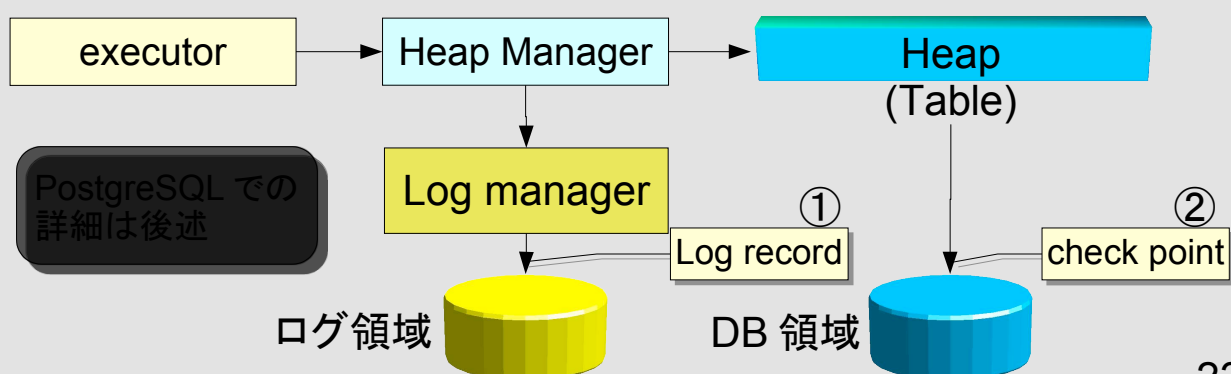


- 下限水位の決定
  - それ以上古いログは不要
  - チェックポイントの時点で、それ以前のコミット済み TrX の内容は DK 上にある
- redo スキャン
  - 下限水位から走査
  - ログレコードを順次適用

22

# ログへの書き込み手順: WAL

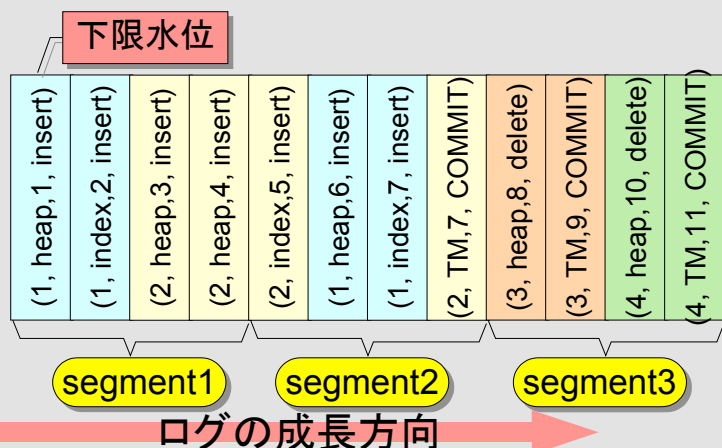
- いつログに書き込むか?
  - DB ファイルを更新する前にログファイルに書く
    - コミット前に DB を更新できる (可視性は別として)
    - 変更が確実に記録される
    - この手順を write ahead log(WAL) と呼ぶ
- Heap と DB ファイルの同期は checkpoint で行う



23

# ログデータの全体的な構造

- 基本的には、無限に伸びるログレコードの列データ
  - 現実にはセグメント化 & リサイクル
  - セグメントのアーカイブ化 (PITR)
- ログレコードには LSN が付与される
- 下限水位が (ログとは別に) 設定される

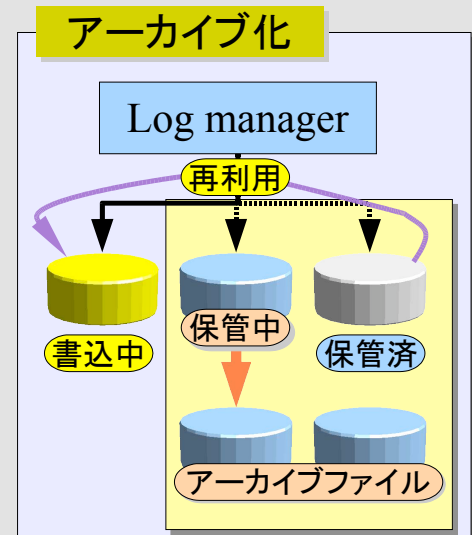


- ログは segment という名前のファイルに分割されて格納される
- segment ファイルは順次再利用される
- 書き込みが完了した segment は別のディスクへコピーする ⇒ アーカイブ化 (PITR)

24

# ログデータのアーカイブ化

- ログファイルはセグメント化する
  - セグメントは3つは必要
    - アクティブ、アーカイブ待ち、予備
    - PostgreSQL のデフォルトも3つ
- 書き込みが終わったセグメントはリサイクルする
- リサイクルする前にセグメントを別の安全な場所へ移す (アーカイブ化)



25

# PostgreSQL の Log のしくみ

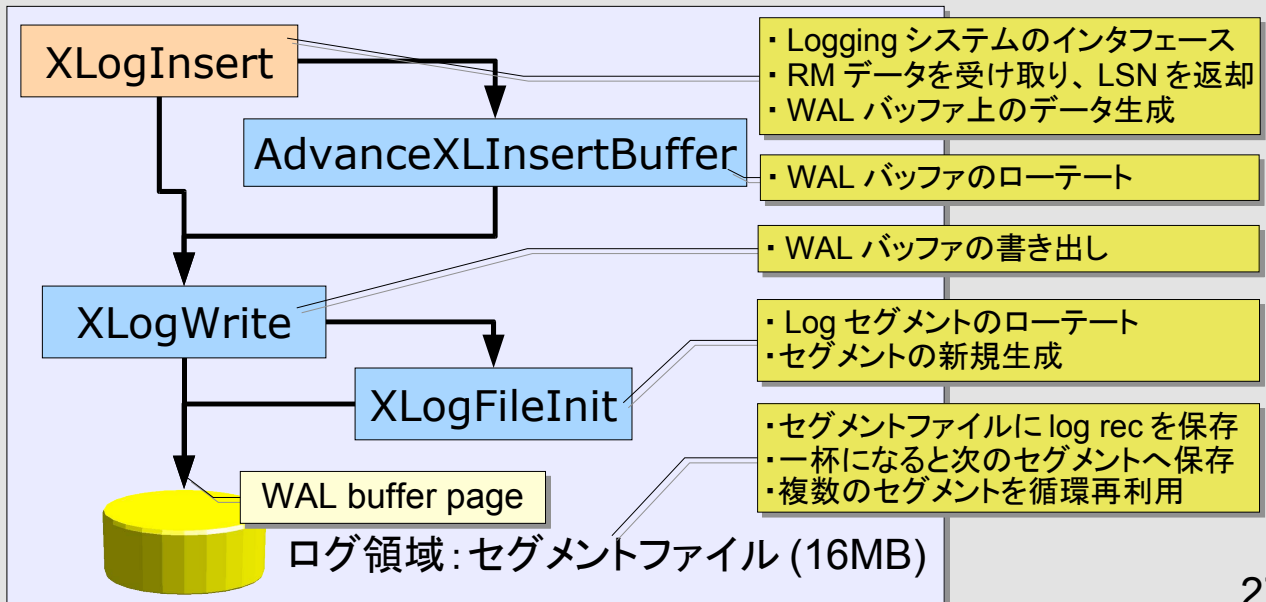
PostgreSQL7.4 でのログの  
実装の説明をします

大きな枠組み  
内部構造  
主要データ構造

26

# Logging システム内部構造

- 関数レベルでの内部構造 (下図)
  - 在り処 src/backend/access/transam/xlog.c



27

# Logging システムのデータ構造

- Log 関連の主要データ (その1)
  - ヘッダほか

## XLogRecord

xl_crc	crc64	誤り訂正コード	ログレコードの破損を検出する
xl_prev	XLogRecPtr	直前 log rec へのリンク	
xl_xact_prev	XLogRecPtr	直前 TrX の log rec へのリンク	
xl_xid	TrID	log rec を書き込んだ TrX	
xl_len	uint16	ログレコード長	XLogInsert で設定
xl_info	uint8	RMでの操作を示すコード	
xl_rmid	RmgrId	どのRMのデータか	

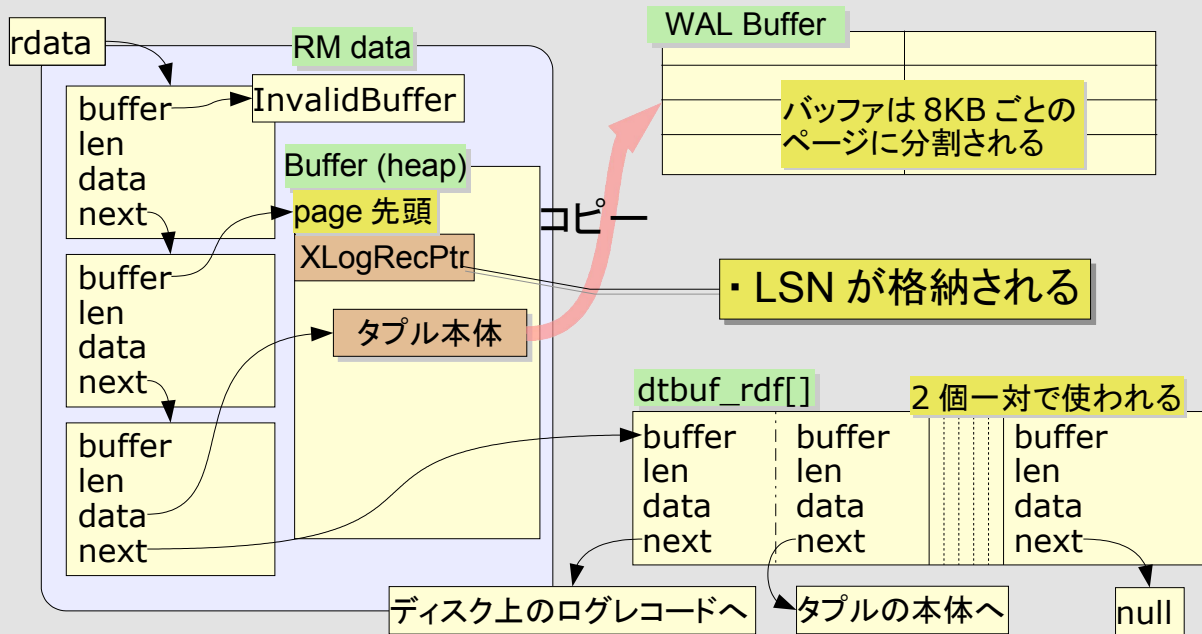
## XLogPageHeaderData

xlp_magic	uint16	0xD05A; WAL の版を示す	AdvanceXLogInsert で設定
xlp_info	uint16	0; 使われていない?	
StartupId	xlp_sui	新規起動毎に 1 ずつ増える	
xlp_pageaddr	XLogRecPtr	全ての backend で同じ値 このページヘッダの位置	

28

# Logging システムのデータ構造

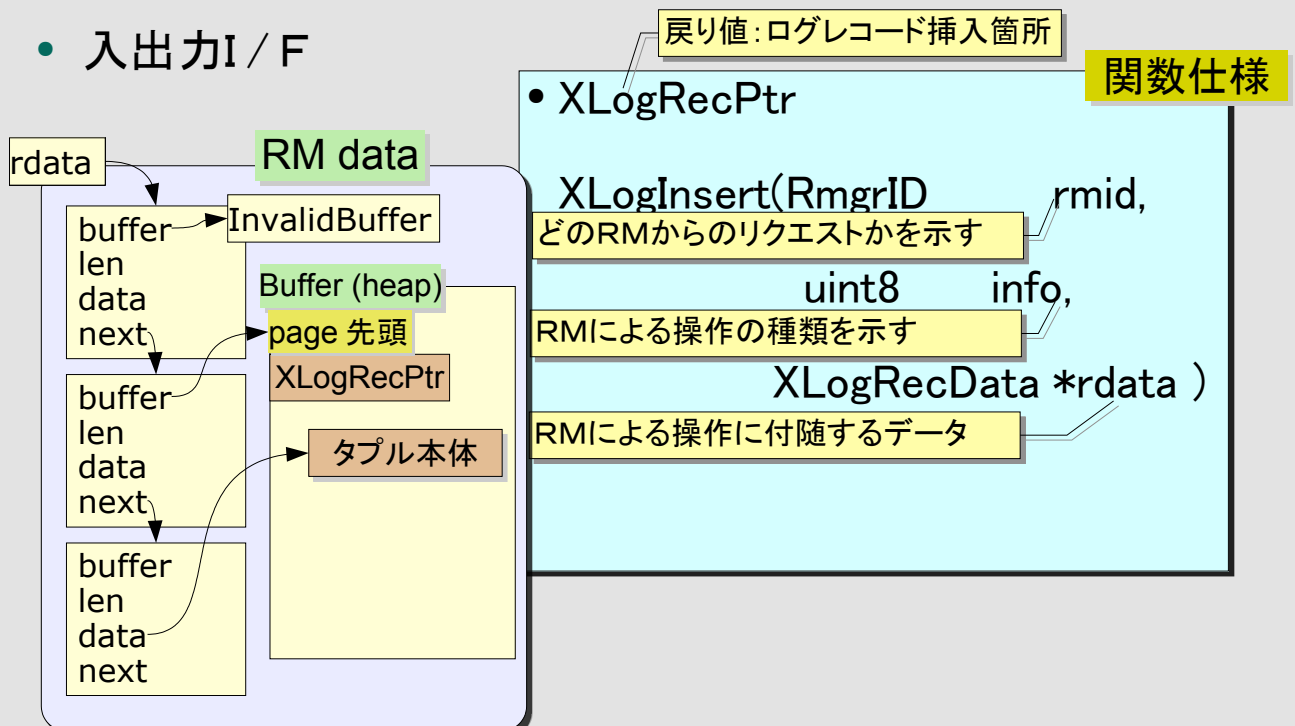
- Log 関連の主要データ (その2)
  - RM と WAL バッファのデータ構造概要



29

# XLogInsert の概要

- 入出力 I / F



30

# XLogInsert の概要

- 主な機能
  - WALバッファに書き込む前のデータ構造の準備
    - ログのデータ構造を生成
    - CRC(巡回冗長コード)の生成: 誤り検出 & 訂正
    - RedoRecPtr の最新性の確認(後述)
  - WALバッファ上へのデータ転送
    - ページが一杯になったらページ送りする  
⇒ AdvanceXLogInsertBuffer() を呼ぶ
    - 半分以上埋まっており、ロックが獲得出来る  
⇒ 早期書き込み XLogWrite() を呼ぶ
  - 後始末

31

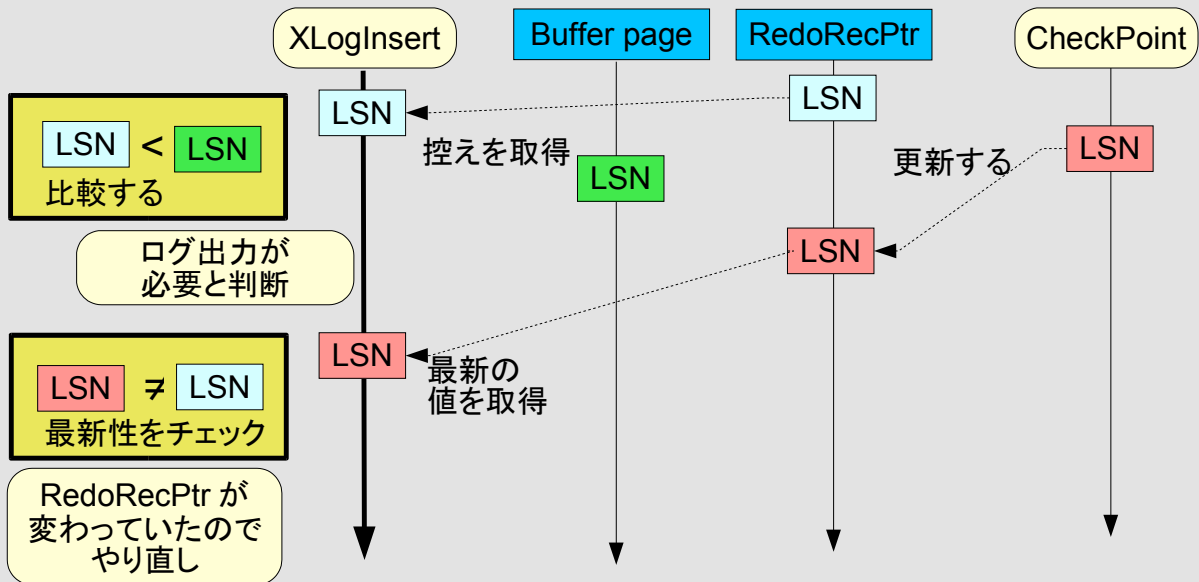
# XLogInsert の概要

- RedoRecPtr の最新性の確認
  - checkpoint は非同期的に動作するので、backend が処理中に RedoRecPtr が更新されることがある
  - RedoRecPtr は排他制御される資源
- RedoRecPtr のロック時間の最小化
  - 各 backend が控えの値を持っている
  - 先の判断は控えの値を用いて処理する
    - その後、ログヘッダなどを生成する
- 楽観的な排他制御:
  - 最新性の確認後に WAL バッファ上へのデータのコピーを行う
  - 控えの値が最新でなければ、最初からやり直し  
⇒ 次のスライド参照

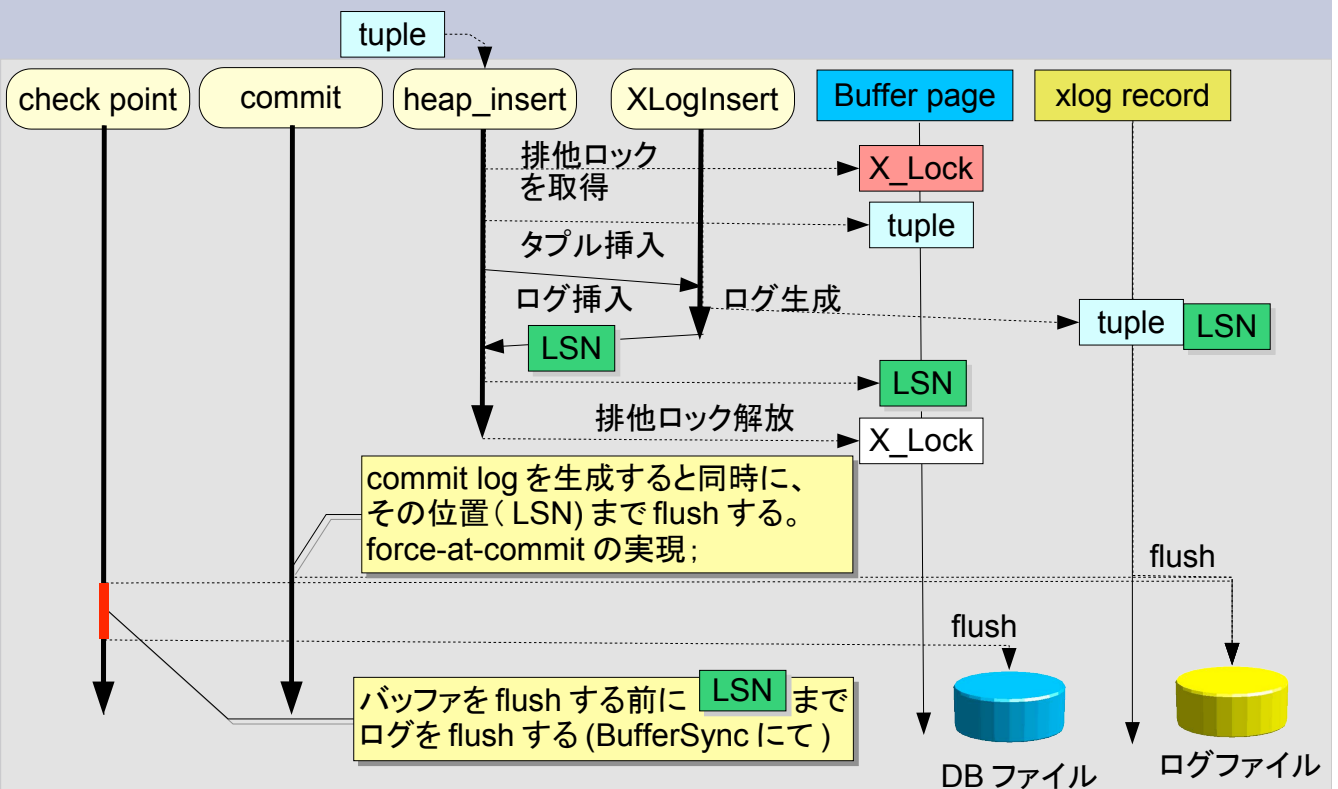
32

# XLogInsert の概要

- RedoRecPtr の最新性の確認



# WAL 的挙動 — heap insert の場合



## 各種まとめ作業

- 継続する必要性⇒次の作業への展開
  - PostgreSQLは大規模。継続は力
  - 一人での読了はムリ。共同が鍵 (他の人が継続)
- 測定と計画：組織的取り組み
  - 実績値に基づく作業計画の必要性
    - 生産性情報の取得;ソフトウェア開発の計画と同じ
- 展開とは共有なり
  - 資料化することが大事 (人間は忘れる葦である)
  - 資料の共有はもっと大事

35

## 構成の例：設計書風に

- 要件 運用・サポートなどを念頭に
  - 分析対象となる機能への要件、分析範囲など
- 機能 / 外部仕様 運用・サポート / 改造作業両方に
  - 分析対象の持っている機能
  - インタフェース (概要)
- 内部構造 改造作業にも使えるレベルで
  - モジュール分割とインタフェース (詳細; API, FAP)
  - データ構造 + アルゴリズム
    - アルゴリズムはソースコードの概要を説明

36

# Fin

ソースを読んで資料化したら、是非、  
仕組み分科会に寄書してください

37

## DB とログの参考文献

- 概要レベル
  - 北川 博之, “データベースシステム”, 昭晃堂, 1996.
    - DB 全般への簡潔な入門書。行間を補う必要が大きい。
- もう少し詳しい本
  - P. Bernstein, E. Newcomer, “トランザクション処理システム入門”, 日経 BP, 1998.(原著 1997)
    - TP 処理に関する教科書。今回の説明の WAL 周りは本書を参照した
  - Garcia-Molina, J. D. Ullman, J. D. Widom, “Database Systems: The Complete Book”, Prentice Hall, 2001.
    - DB 分野全般にわたる詳しい教科書。上記の B 氏本よりは一般的

38

# DB とログの参考文献

- 詳細レベル
  - J. Gray, A. Reuter, “トランザクション処理—概念と技法〈上 / 下〉”, 日経 BP, 2001.(原著 1992)
    - 実装面も詳しく書かれている。今回の説明のログデータの構造は本書を参照した。
    - WAL を実装した M. Vadim も読んだらしい
  - PostgreSQL のマニュアル
    - WAL の解説がある。
    - チューニングについても触れている
  - PITR の解説
    - 坂田 哲夫, “Point In Time Recovery のしくみ”, 'まると PostgreSQL', インプレス, 2004.