PostgreSQLのLogのしくみ

三訂版
平成廿一年二月十四日

NTT OSSセンタ
坂田 哲夫

あらまし

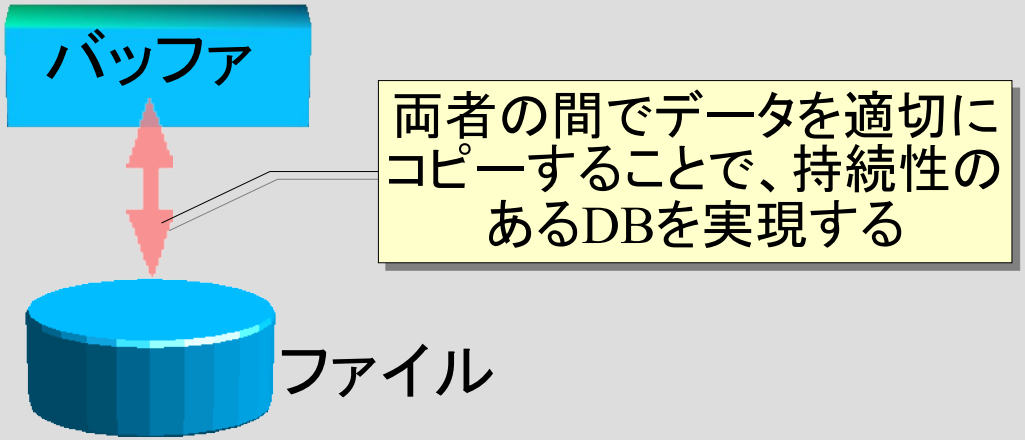
- ログの概要
  - ログとはどのようなモノか
  - ログによるリカバリ
  - WALについて
  - 論理物理ロギング
  - ログデータの構造
- PostgreSQL7.4でのログのしくみ
  - 大きな枠組み：ロギングサブシステムの位置付け
  - 内部構造：関数とデータ構造レベルでの説明
- PITRについて
  - 追加機能の概要
  - RPITについて
- まとめ、文献

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

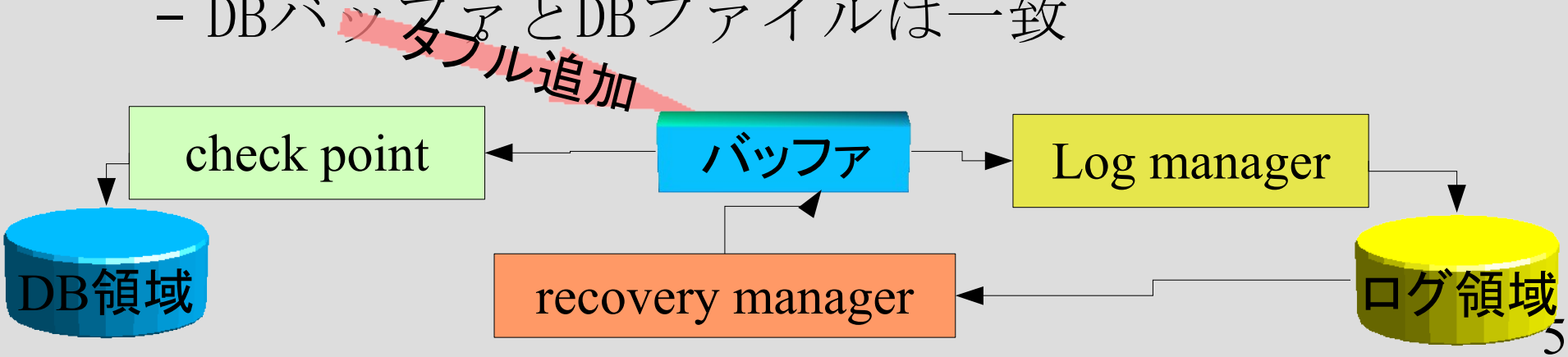
データベースのストレージ

- 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)



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



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

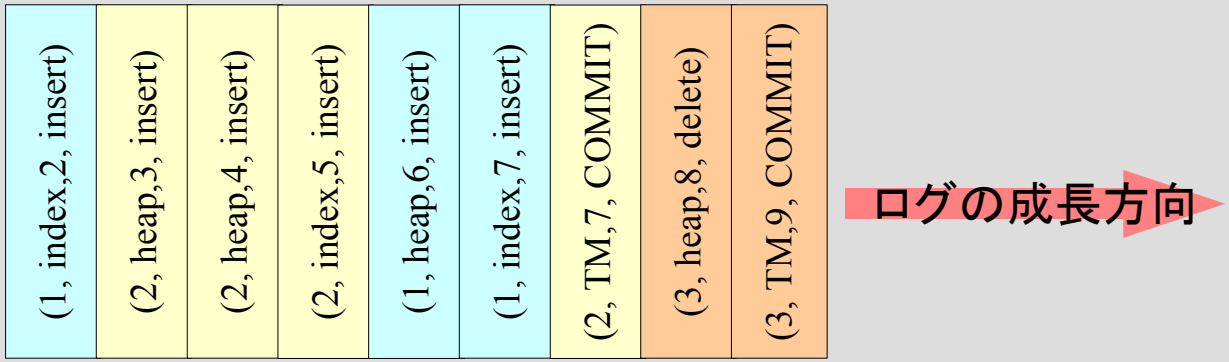
- ログファイルを別に持っている意義
  - コミットの際に全ての結果をDBファイルへ書き出す（原子性と持続性を満たすため）
    - コミット処理時にダウンするとどうなるか？
    - ディスクが隘路になる
  - ディスク上の別の場所に一旦書き出す
    - システムダウンしてしても記録は残る⇒リカバリ可能
    - 固めて書き出す⇒隘路の軽減

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)

Log recordの例

(XID, RM, LSN, ACT)

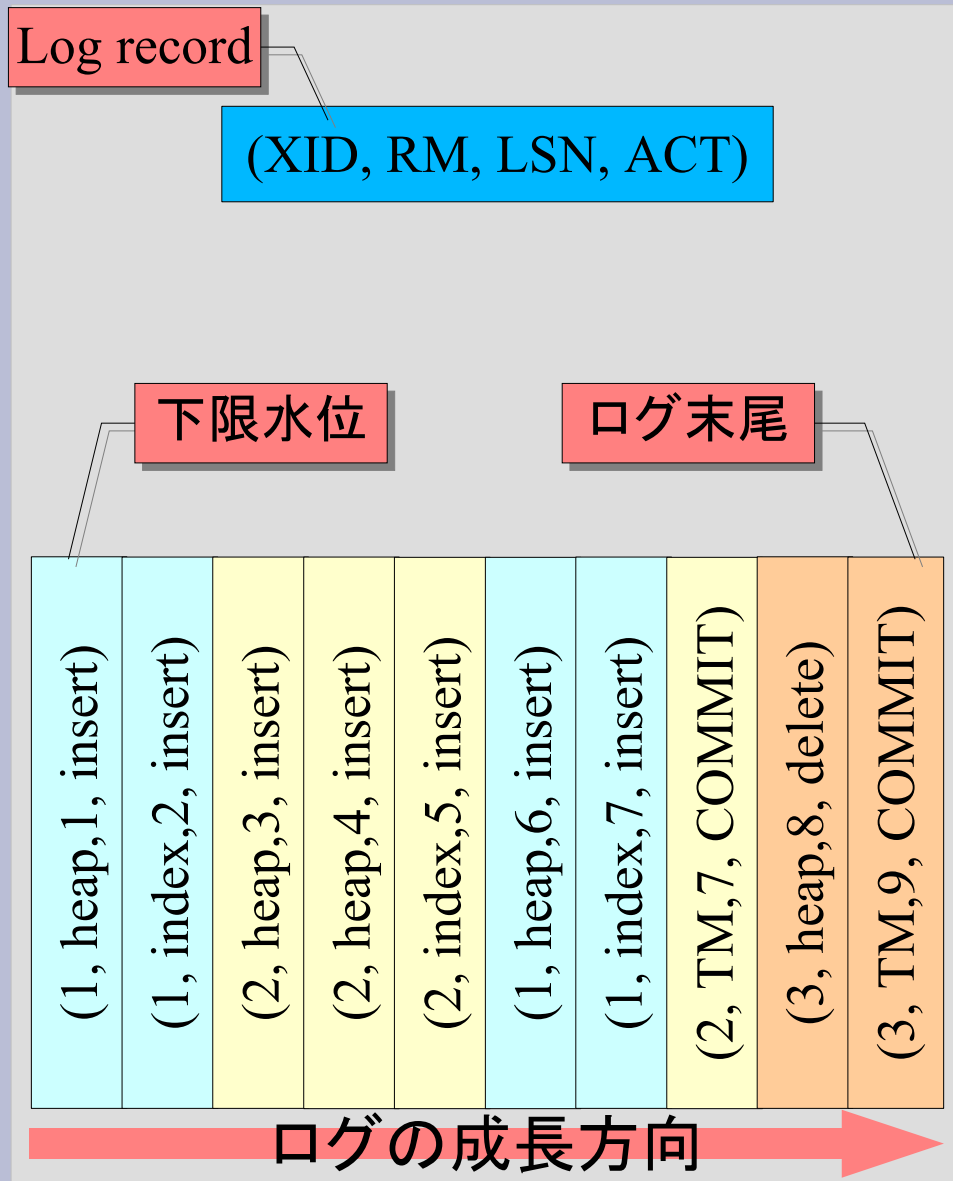


ログによるリカバリ

- 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
  - 再生開始時点まで巻き戻す'undoスキャン'
    - 未コミットTrXのログを参照して、DBを元に戻すロールバック処理
    - ロールバックセグメントがあるので、そこまで実行する
  - 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

PostgreSQLにはございません

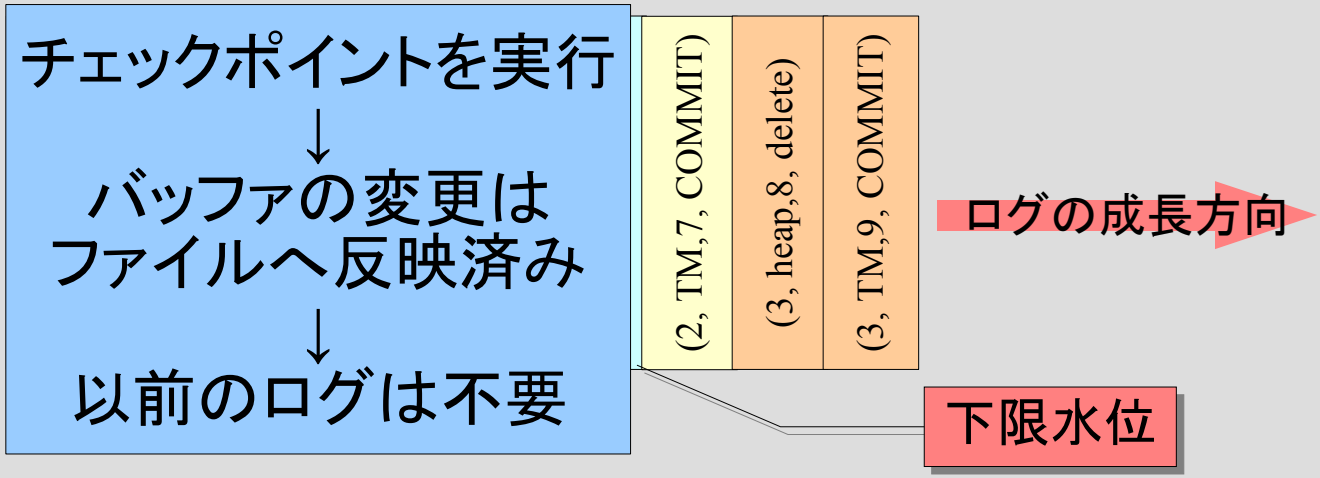
リカバリ動作のイメージ



- 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
- undoスキャン
  - 下限水位まで逆走査
  - PosgreSQLでは不要
- redoスキャン
  - 下限水位から走査
  - ログレコードを順次適用

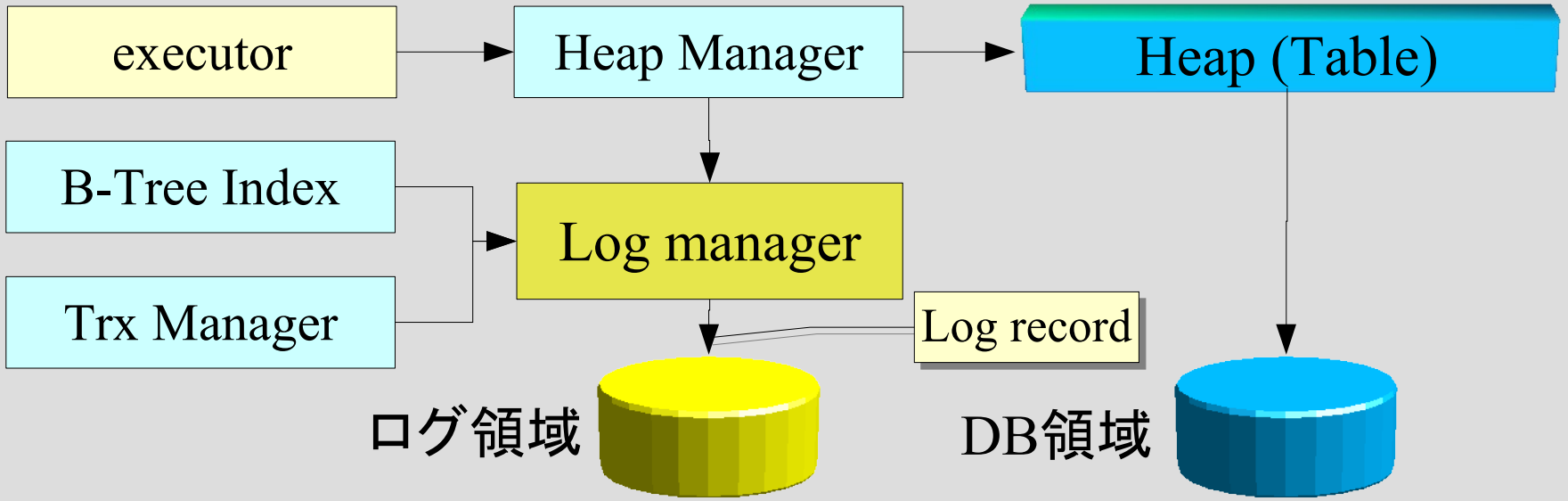
下限水位 (low water mark)

- リカバリでは、全てのログを反映するべきか？
  - リカバリの際には、バッファの不整合(ファイルとの不一致)を修正できれば良い
  - チェックポイントを実行すると、その時点以前の操作についてはバッファとファイルは一致している
    - それより古いログは不要
  - ログ上のその点を下限水位と呼ぶ



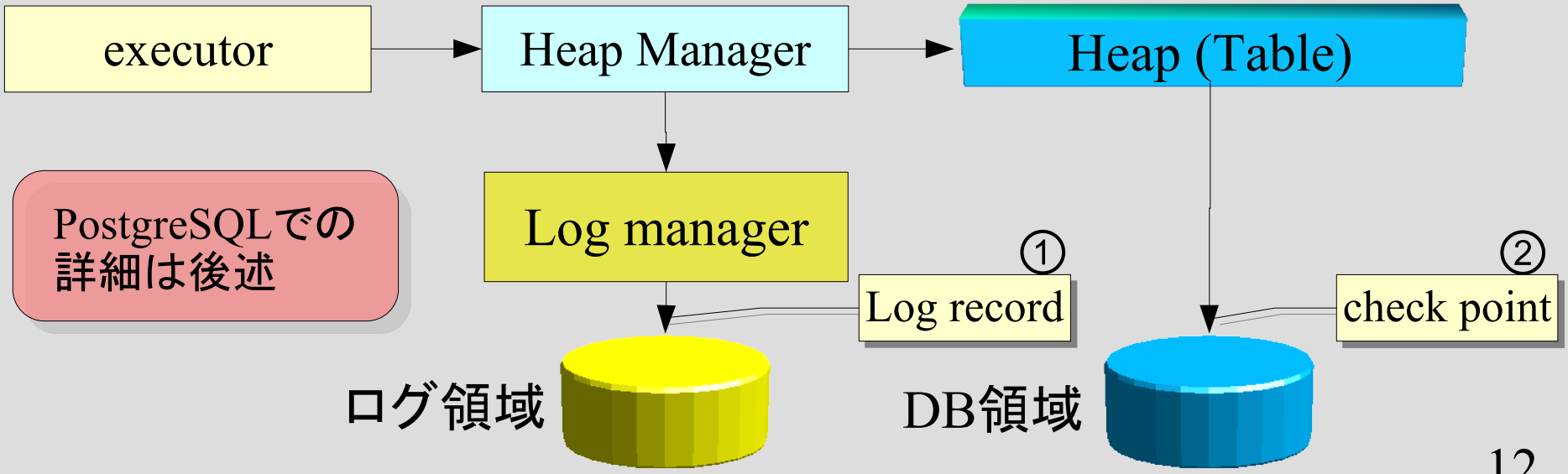
ログへの書き込み

- ログに対する書き込み
  - リソースマネージャ(RM)からのデータをログに記録
  - RM：ログに格納すべきデータを管理しているサブシステム
    - Heap, B-Tree, xact, など



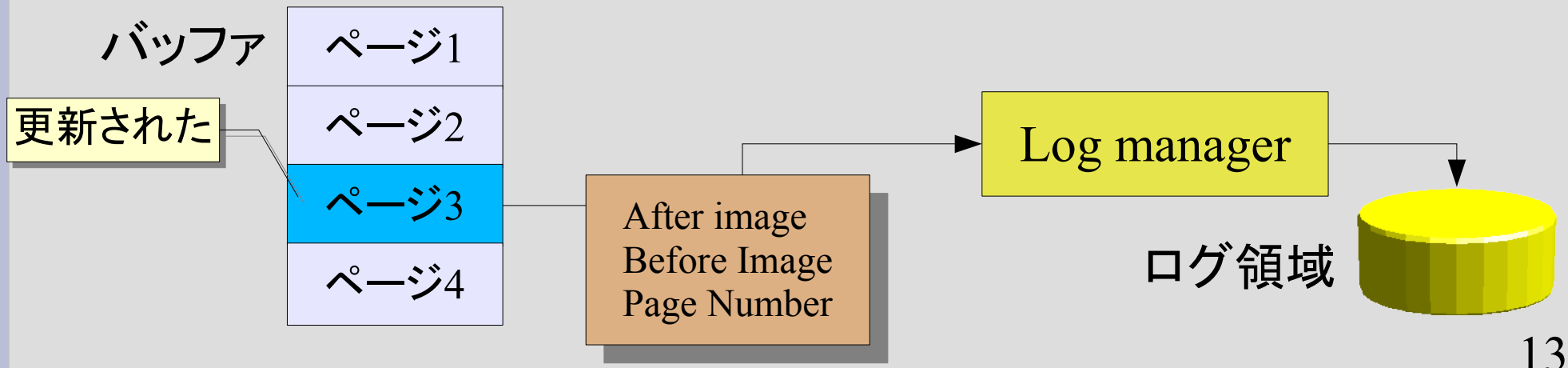
ログへの書き込み：WAL

- 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



何がログへ書き込まれるか

- 各種のログ方式に共通
  - 何番目のログレコードか(log serial number; LSN)
    log managerが付与する
- 物理ロギングの場合
  - どのページを出力したか
  - 更新後のページが出力される(after image)
  - 更新前のページも出力される(before image)

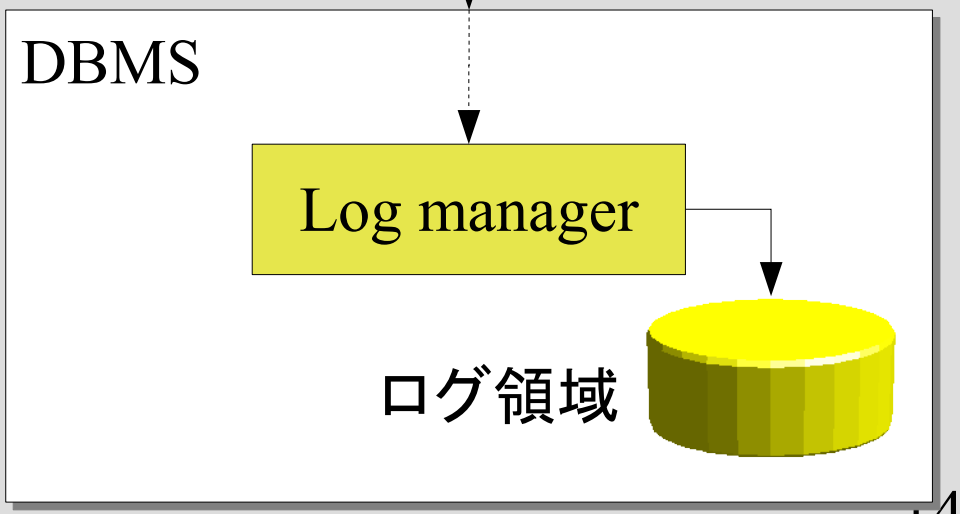


何がログへ書き込まれるか

- 論理ロギングの場合
  - DBMSに対する操作を記録する

```
INSERT INTO table1 value (1,2,3);
INSERT INTO table2 value (4,5,6);
```

  - 容量が小さいという利点がある
  - だが、問題もある…
    ⇒次スライド



冪等 (idempotent)ということ

- リカバリでのredo操作を考える
  - 同じredo操作を2度以上実行しても、常に同じ結果になることが望ましい
    この性質を冪等(べきとう;巾等)と呼ぶ
    - チェックポイント済みか否かに関わらず、常にredoすれば、バッファの内容はログを正しく反映する
- 巾等な操作の例
  - ページイメージを決まった位置へ上書きする
- 巾等でない操作の例
  - SQLのINSERT文をn回実行すると、n行追加される

何がログへ書き込まれるか

- 各方式の得失を考える観点
  - 巾等性：同じ操作を何度実行しても、常に同じ結果を得ること
  - ログデータの量：少ない方がよい

観点	***物理***	***論理***
べき等性	あり。	実現困難 例えば、insert を考えよ
ログの容量	大	物理ログより小

両者を組み合わせた方式が必要

何がログへ書き込まれるか

- 物理論理ロギング
  - ページ内の特定の箇所に対する
  - 論理的な操作を記録する
    - 詳細はGrayの教科書を参照
- PostgreSQLでも採用されている

物理論理ログの例

```
Opcode          -- 操作種別
Page            -- 物理ページ番号
Offset          -- ページ内オフセット
length          -- データ長
record[length]  -- データ本体
```

論理的部分

物理的部分

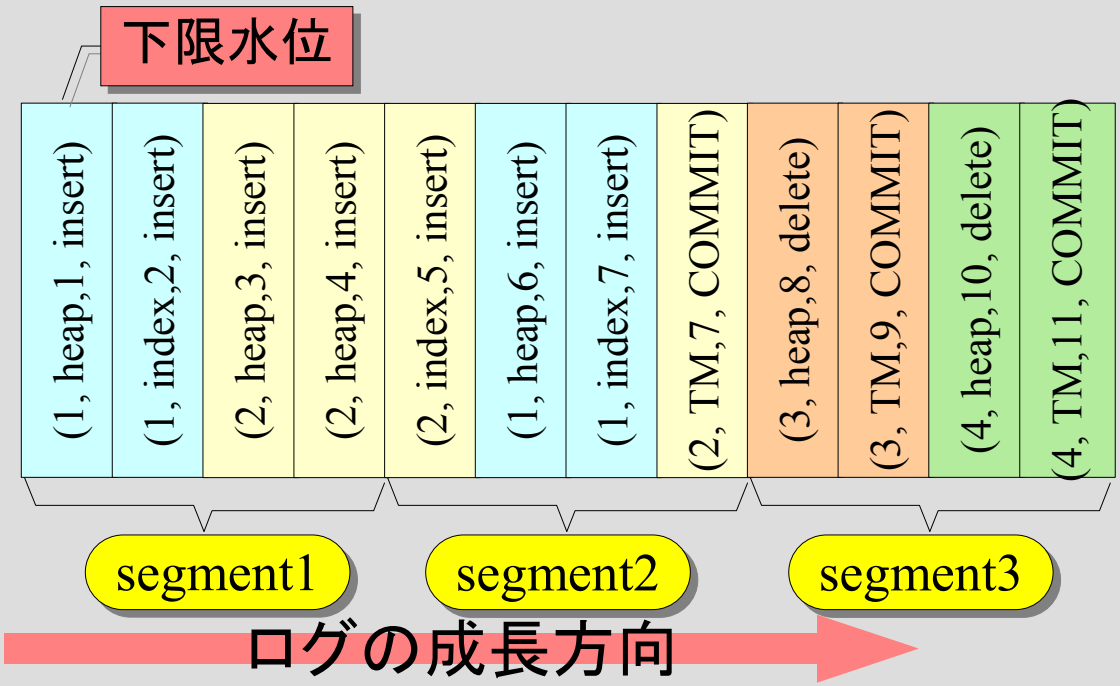
force-at-commit：コミット時の処理

- 通常のファイル書き込み(write)
  - データは直ちにディスクに書き込まれるわけではない(バッファに残っている)
  - コミット時にlogをwriteした直後にシステムダウンすると、ファイルに記録が残らない

- コミット時にはディスクに確実に書き込む
  - この種の書き込みを force-at-commit(コミット時ログ強制書き出し)と呼ぶ

ログデータの全体的な構造

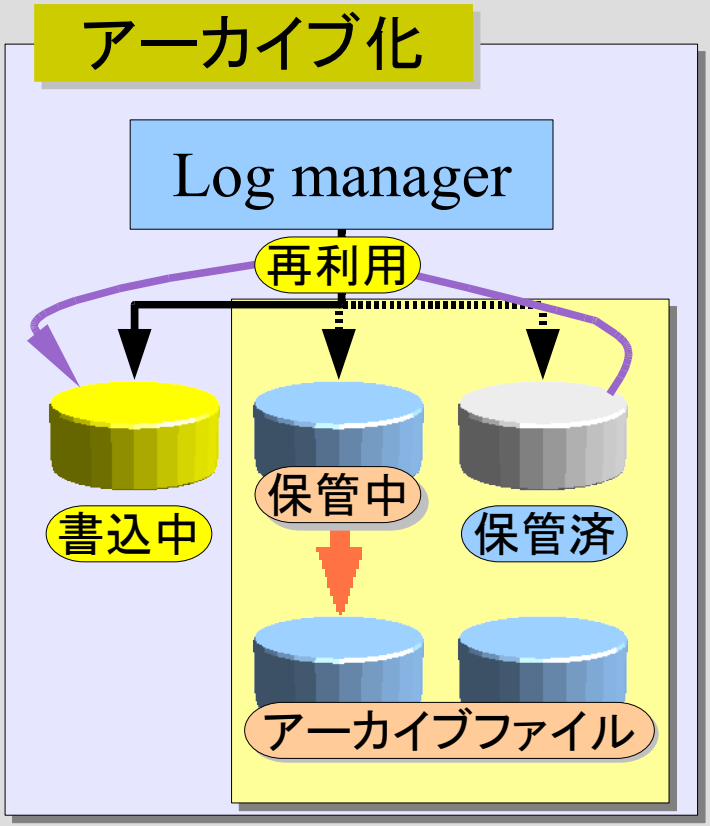
- 基本的には、無限に伸びるログレコードの列データ
  - 現実にはセグメント化＆リサイクル
  - セグメントのアーカイブ化(PITR)
- ログレコードにはLSNが付与される
- 下限水位が(ログとは別に)設定される



- 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)

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)



PostgreSQL 7.4 でのLogのしくみ

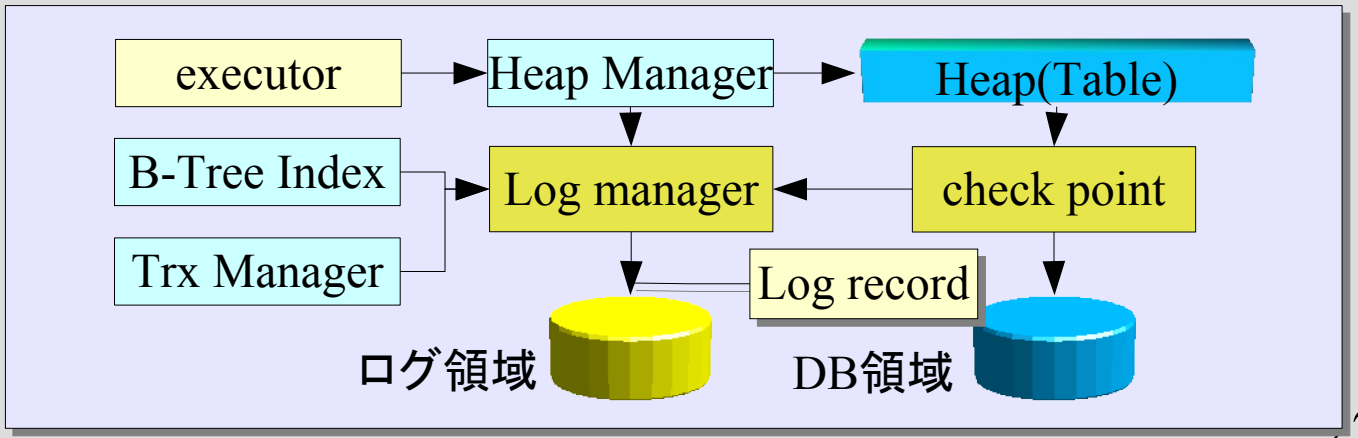
PostgreSQL7.4でのログの
実装の説明をします

大きな枠組み
内部構造
主要データ構造

大きな枠組み

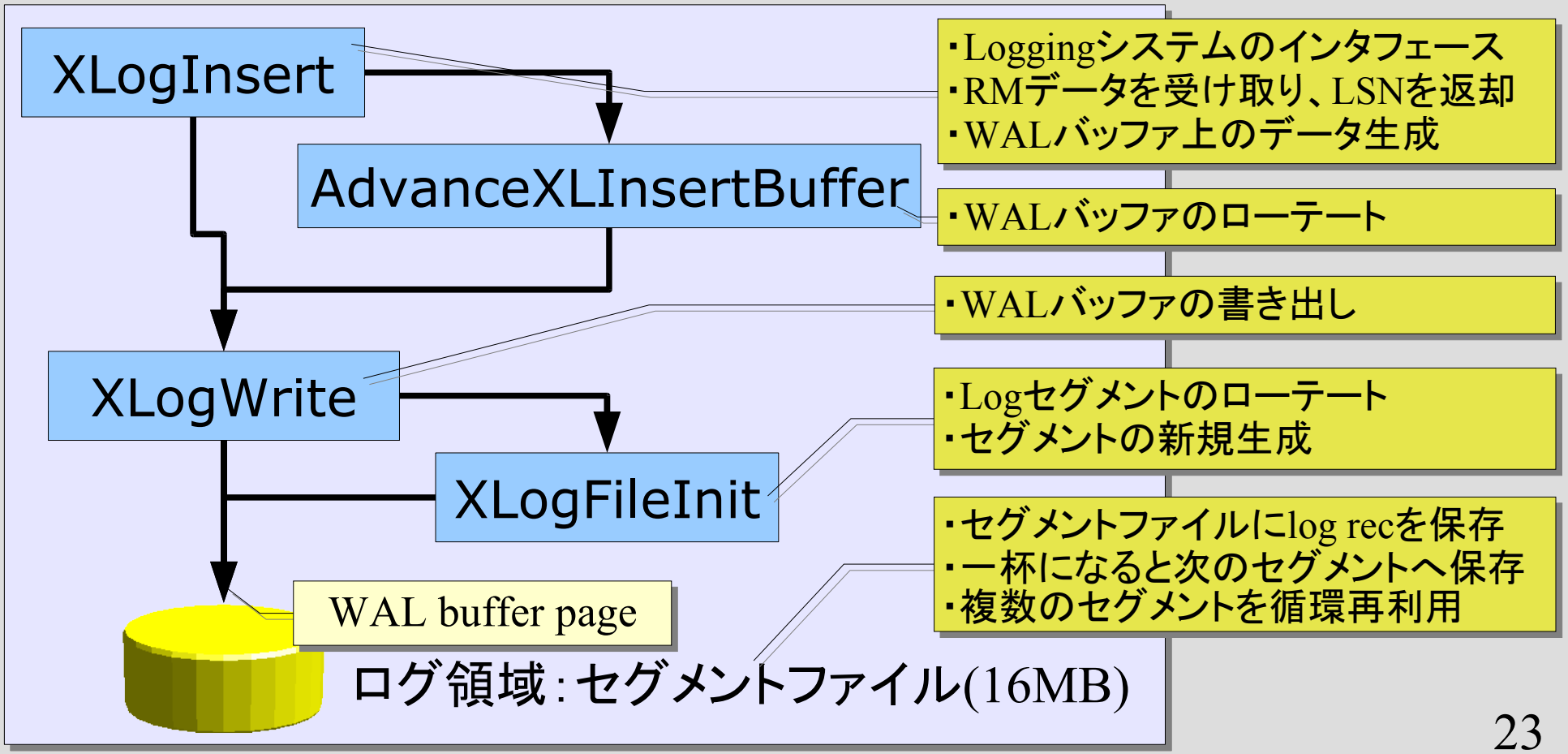
- PostgreSQL内でのLogging systemの位置付け
  - リソースマネージャ(RM)とheapの間にある
  - check pointと協調動作する
- Log資源の管理
  - 物理ログと物理論理ログの混在
  - 可能なら早期に書き込む
  - 複数backendの協調動作(排他制御)

これらの特徴のため
内部の動作は複雑



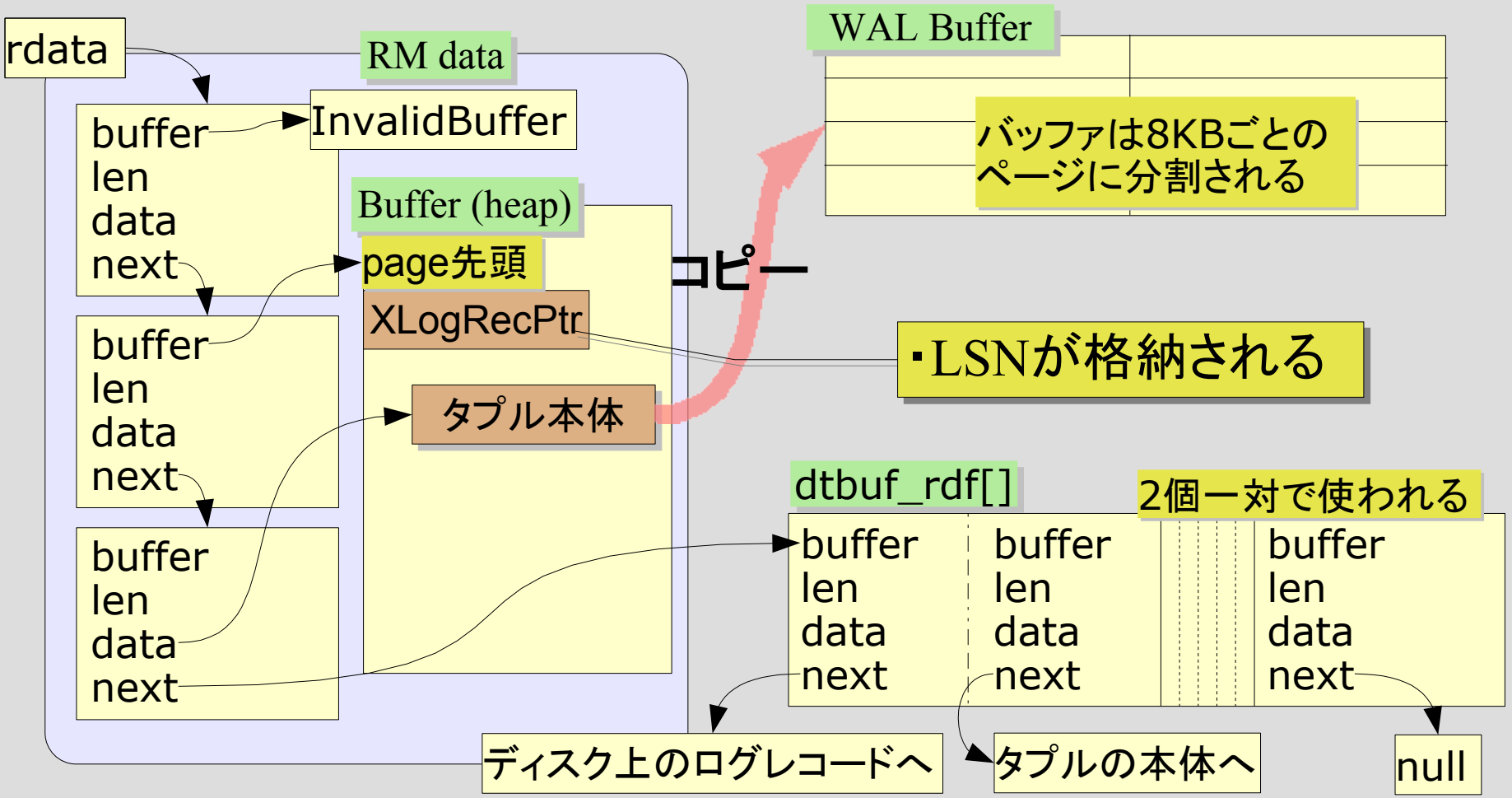
Loggingシステム内部構造

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



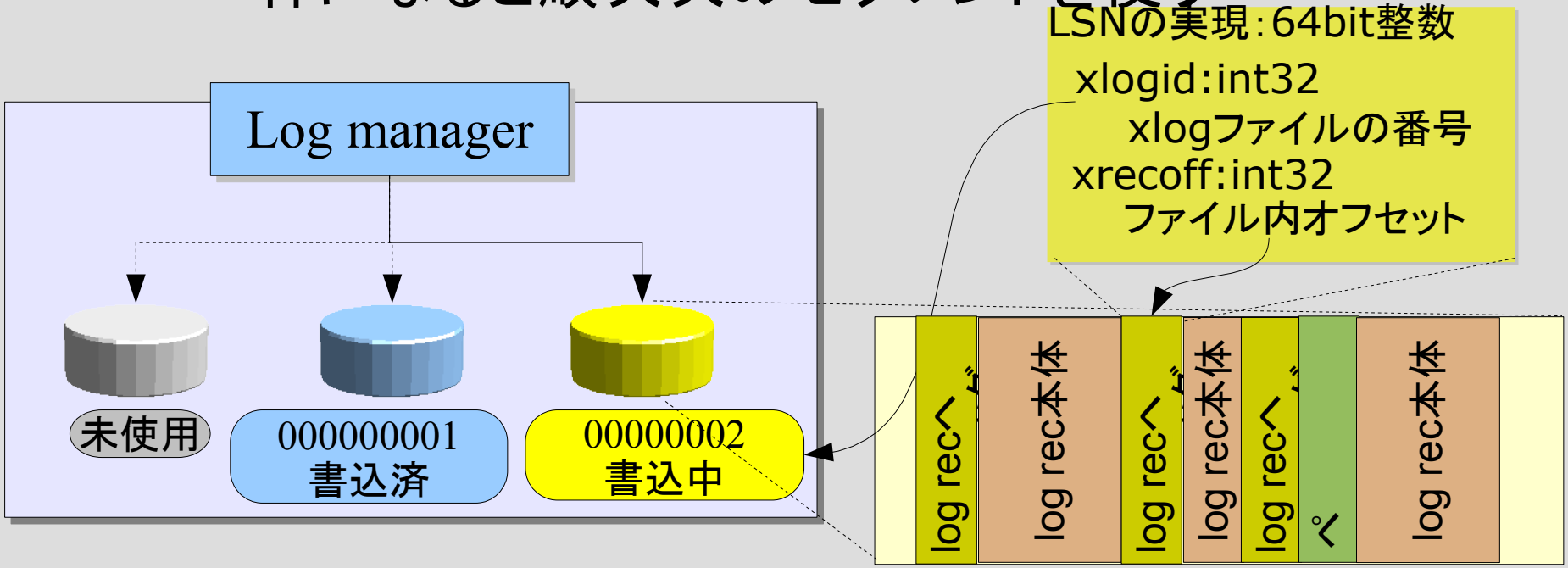
Loggingシステムのデータ構造

- Log関連の主要データ(その1)
  - RMとWALバッファのデータ構造概要



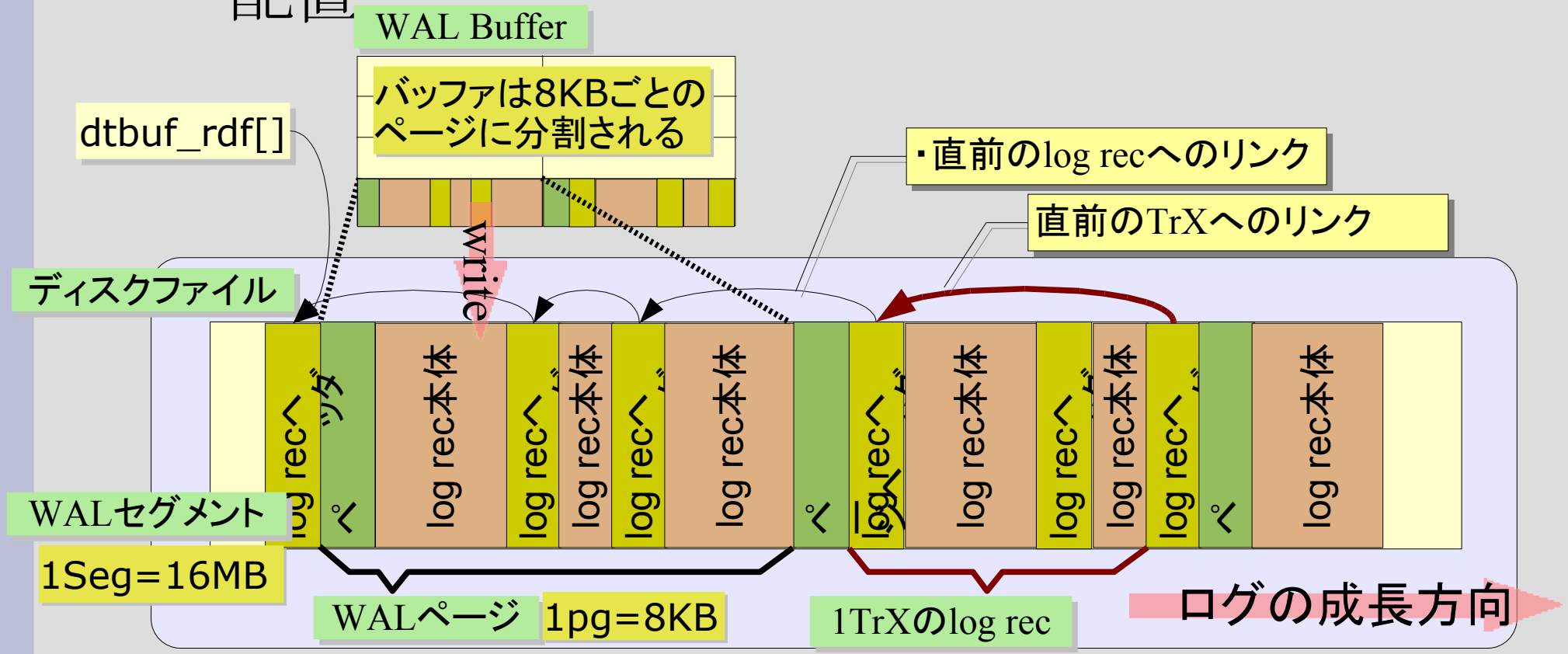
Loggingシステムのデータ構造

- Log関連の主要データ(その3)
  - ディスク上のログファイルの配置
  - 再利用(ローテート)
    - ログセグメントを用意する
    - 一杯になると順次次のセグメントを使う



Loggingシステムのデータ構造

- Log関連の主要データ(その2)
  - WALバッファとディスク上のログデータの配置



Loggingシステムのデータ構造

- Log関連の主要データ(その4)
  - ヘッダほか

ログレコードの破損を検出する

XLogRecord

xl_crc	pg_crc32	誤り訂正コード
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
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

XLogInsertで設定

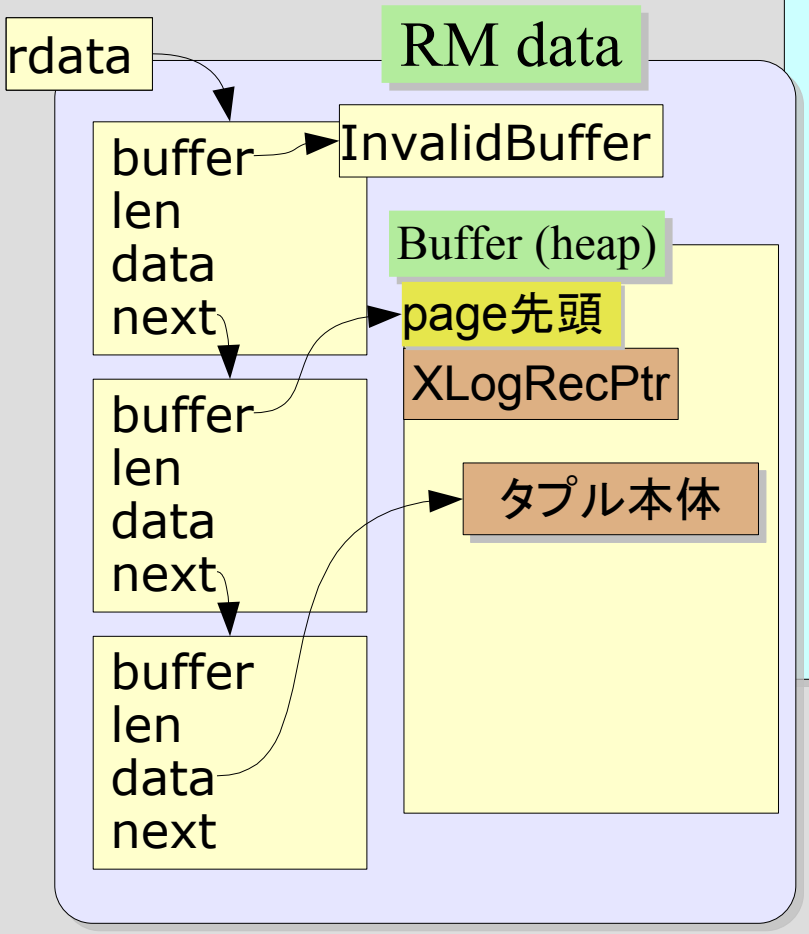
XLogPageHeaderData

xlp_magic	uint16	0xD05A; WALの版を示す
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 全てのbackendで同じ値
xlp_pageaddr	XLogRecPtr	このページヘッダの位置

AdvanceXLogInsertで設定

XLogInsertの概要

- 入出力Ｉ／Ｆ



戻り値：ログレコード挿入箇所

関数仕様

- XLogRecPtr

XLogInsert(RmgrID rmid,

どのRMからのリクエストかを示す

uint8 info,

RMによる操作の種類を示す

XLogRecData *rdata)

RMによる操作に付随するデータ

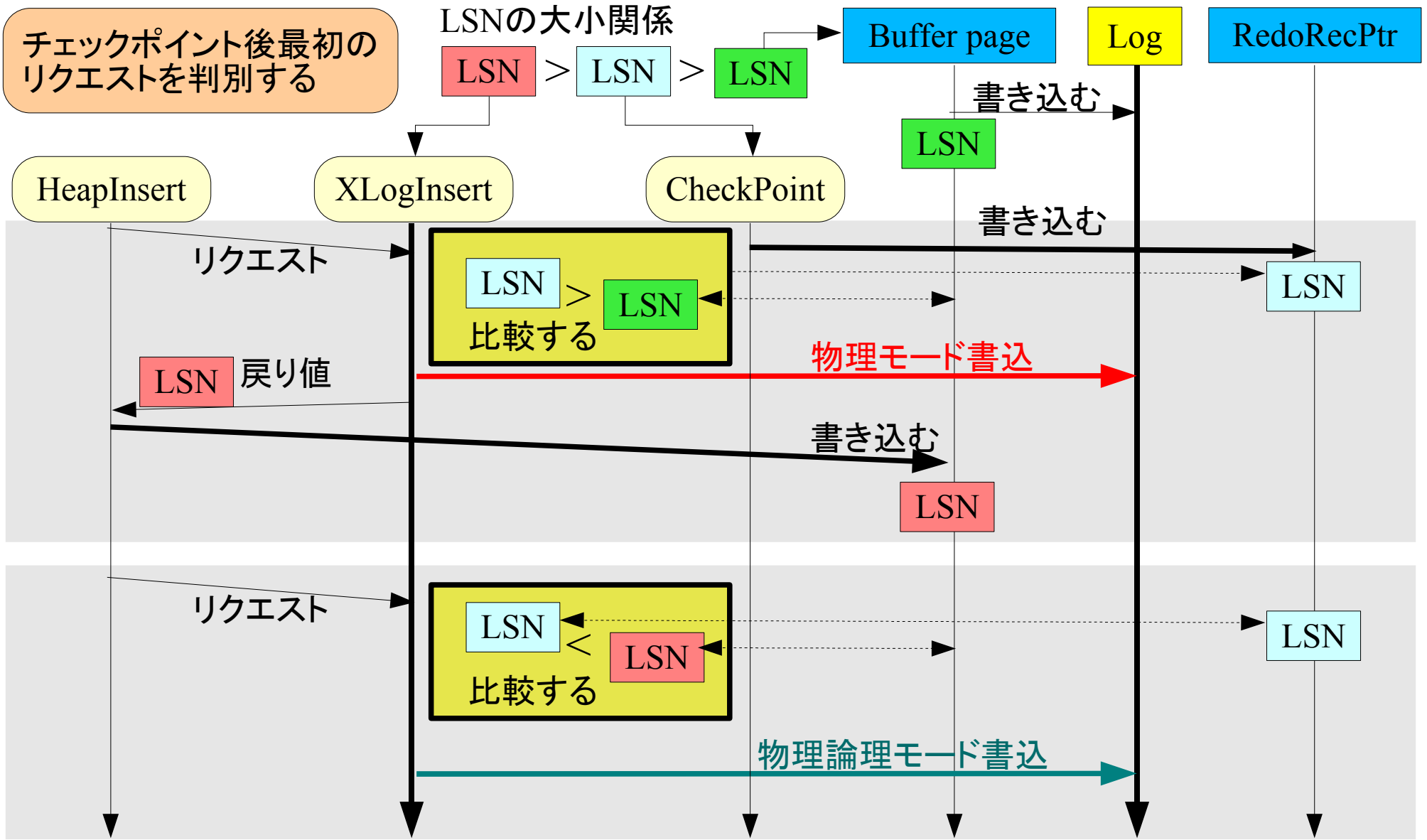
XLogInsertの概要

- 主な機能
  - WALバッファに書き込む前のデータ構造の準備
    - データ構造(その1)でのdt_buf[]など
    - CRC(巡回冗長コード)の生成：誤り検出&訂正
  - 物理ログ/物理論理ログの判断(後述)
    - RedoRecPtrの最新性の確認
  - 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

XLogInsertの概要

- 物理モード/物理論理モードの動作切り替え
  - 通常は物理論理モードで動作する
  - チェックポイント後、そのページの内容が初めてログに書かれる時は、ページ全体を出力(物理モード)
    ⇒詳細は次のスライドで

物理/物理論理の切り替えロジック



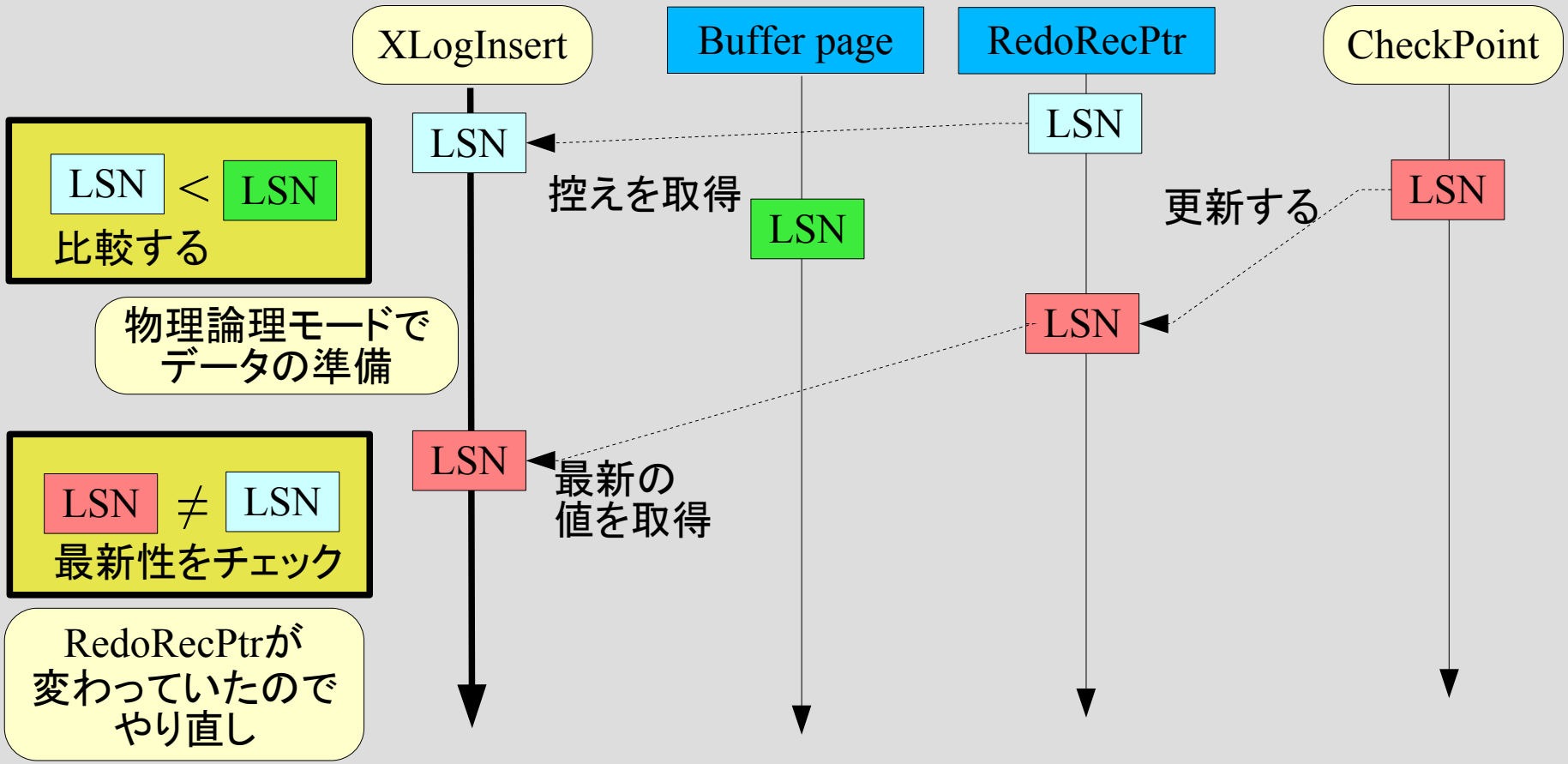
LSN: log serial number

XLogInsertの概要

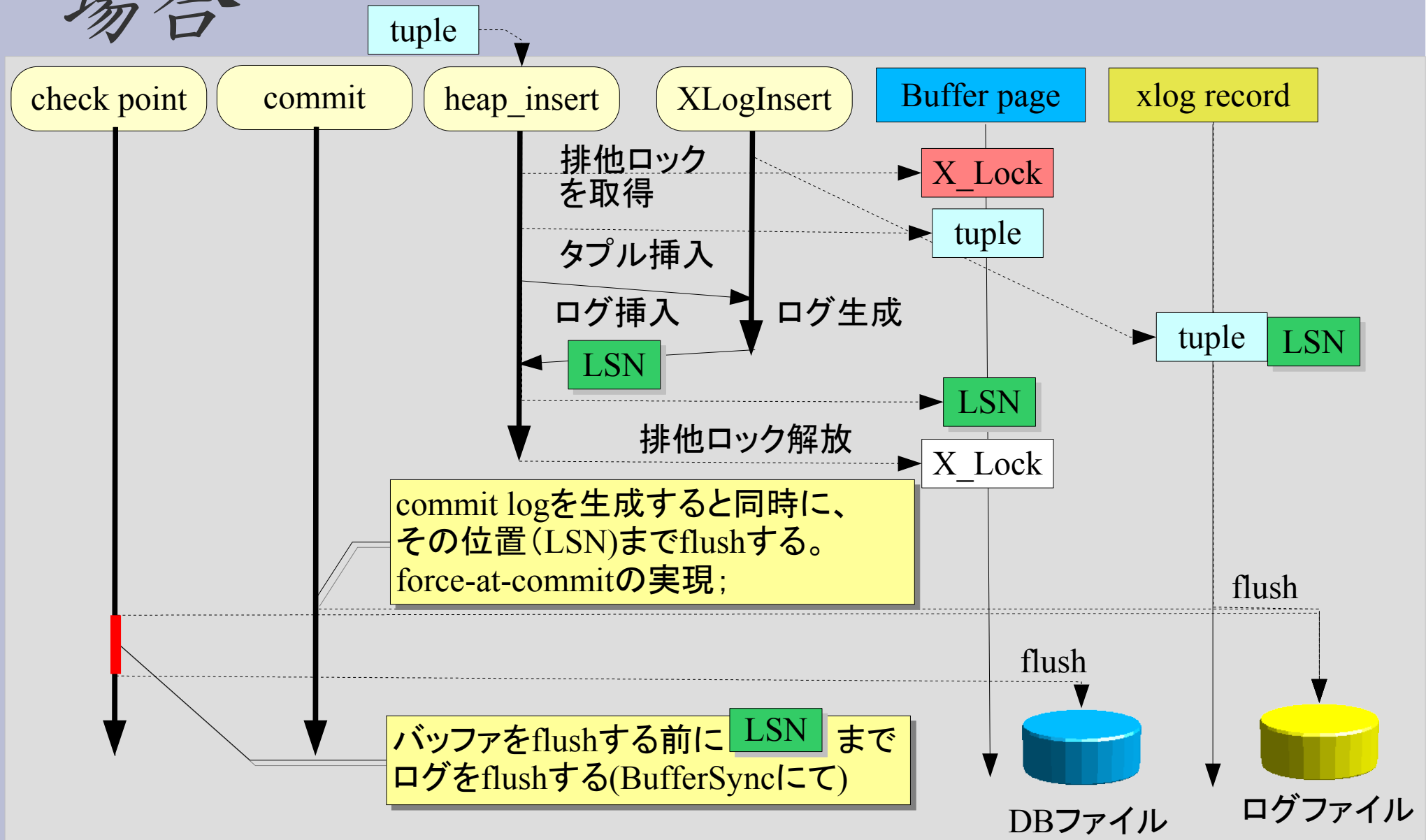
- 物理/物理論理の切り替え(続き)
  - RedoRecPtrの最新性の確認
    - checkpointは非同期的に動作するので、backendが処理中にRedoRecPtrが更新されることがある
    - RedoRecPtrは排他制御される資源
  - RedoRecPtrのロック時間の最小化
    - 各backendが控えの値を持っている
    - 先の判断は控えの値を用いて処理する
      - その後、ログヘッダなどを生成する
  - 楽観的な排他制御:
    - 最新性の確認後にWALバッファ上へのデータのコピーを行う
    - 控えの値が最新でなければ、最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

XLogInsertの概要

- RedoRecPtrの最新性の確認



WAL的挙動ー heap insert の場合



XLogWriteの概要

- 機能
  - WALバッファの内容を実際にファイルに書き出す
  - 書き込み位置の管理を行う

関数仕様

- API

```
void XLogWrite(
        XLogwrtRqst WriteRqst)
```

戻り値：なし

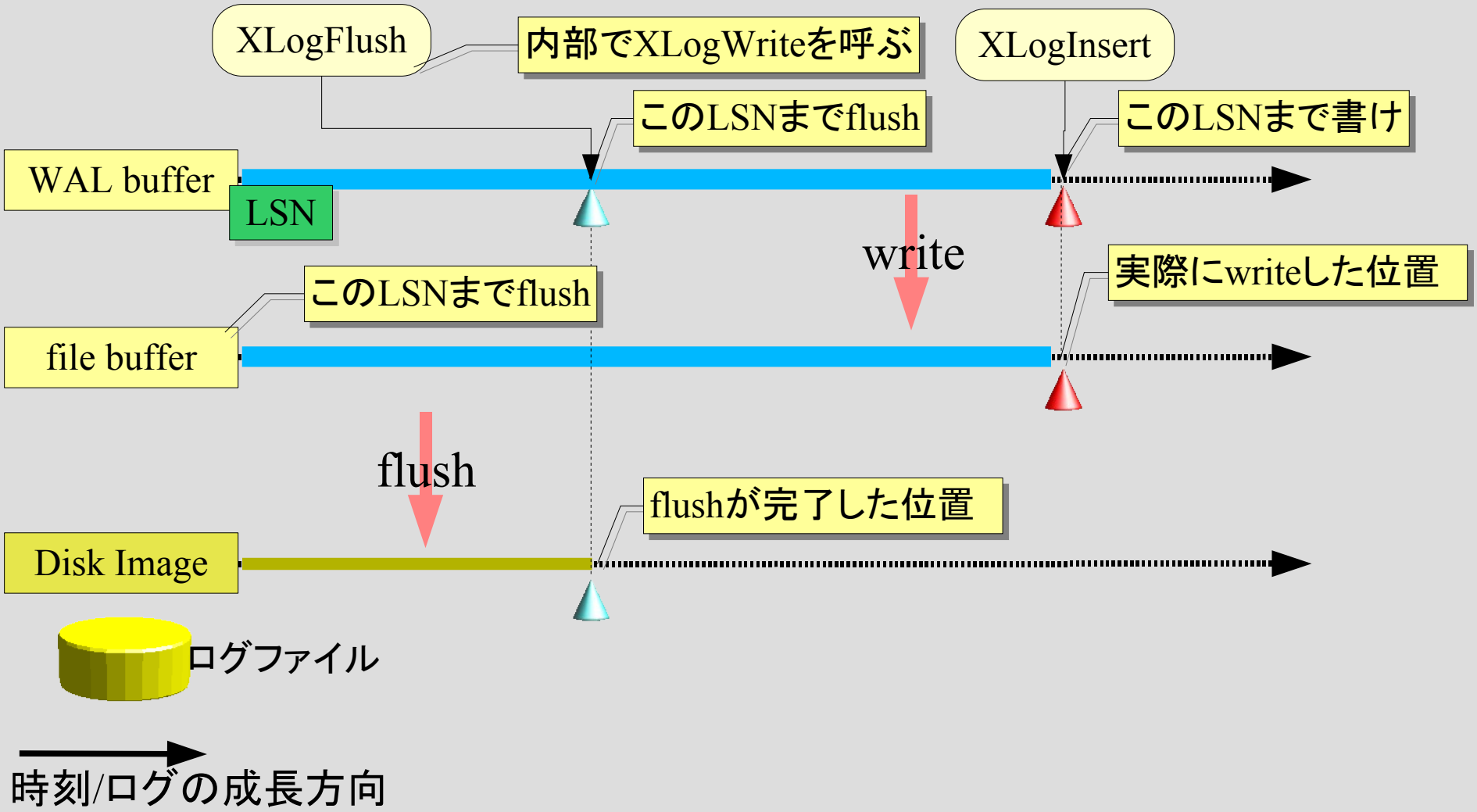
その位置まで書き込むべきLSN;
・writeすべき位置
・flushすべき位置

ログのアドレス(LSN)を示す

XLogWriteの概要

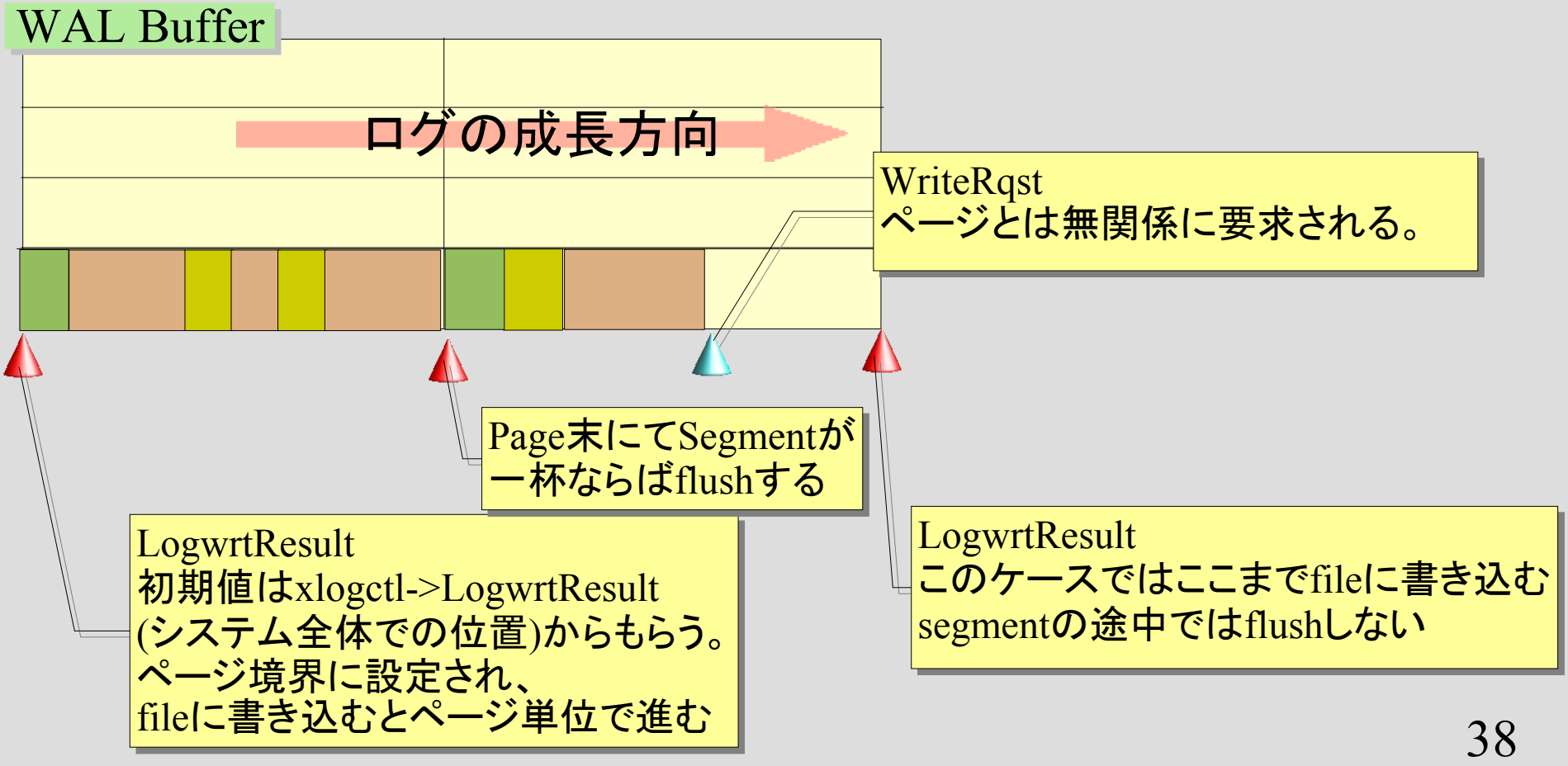
- WALバッファの書き込み管理
  - 同じ型を持つ3つの変数
    - WriteRqst
      書き込みたい先端の位置(引数)
    - LogwrtResult
      実際に書き込んだ位置の各backendでの控え
    - xlogctl->LogwrtResult
      システム全体でのログの書き込み位置
  - メンバの型
    - Write型　write()で書き込んだ位置の先端
    - Flush型　fsync()などでディスク上に確実に書き込んだ位置の先端
  - flushする条件
    - segmentが一杯で最終ページも完結している

XLogへの書き込みの概要



XLogWriteの概要

- WALバッファの書き込み管理
  - WriteRqst, LogwrtResultのイメージ



AdvanceXLInsertBufferの概要

- 機能
  - WALバッファのページを1つ送る
  - 送り先ページが使用中なら書き出す

- API

関数仕様

戻り値：論理型
WAL管理の共有変数の更新の要否

```
static bool

AdvanceXLInsertBuffer(void)
```

引数はない

AdvanceXLInsertBufferの概要

- 動作概要
  - WALバッファのページを1つ送る
    - WALバッファの現在の書き込み位置はWALの書き込み位置を示す共有変数XLogCtl->xlblocks[nextidx] から取得する
  - 送り先ページが使用中なら書き出す
    - 書き出し自体はXLogWrite()で実行
    - 書き出したら、WALの書き込み位置を示す共有変数を更新する
    - 使用中でなくとも共有変数の更新が必要だが、ここでは行わずに必要性を戻り値で返す(他の処理でもっと前に進める可能性が高いため)
  - 新しいページを初期化する

Log-Recoveryのチューニング

- checkpointの間隔の最適化
  - checkpointの間隔広い⇒リカバリ時間長い
  - checkpointの間隔狭い⇒リカバリ時間短い
- force-at-commitの負担軽減
  - 遅延コミットの実施
    - コミット時に直ちにforceするのではなく、ちょっと待ってみる(その間に他のbackendがflushする可能性が有る)
- 適値を探る
  - 常に妥当な最適値はなく、事例ごとに異なる
  - benchmarkツールなどを使用してチューニング

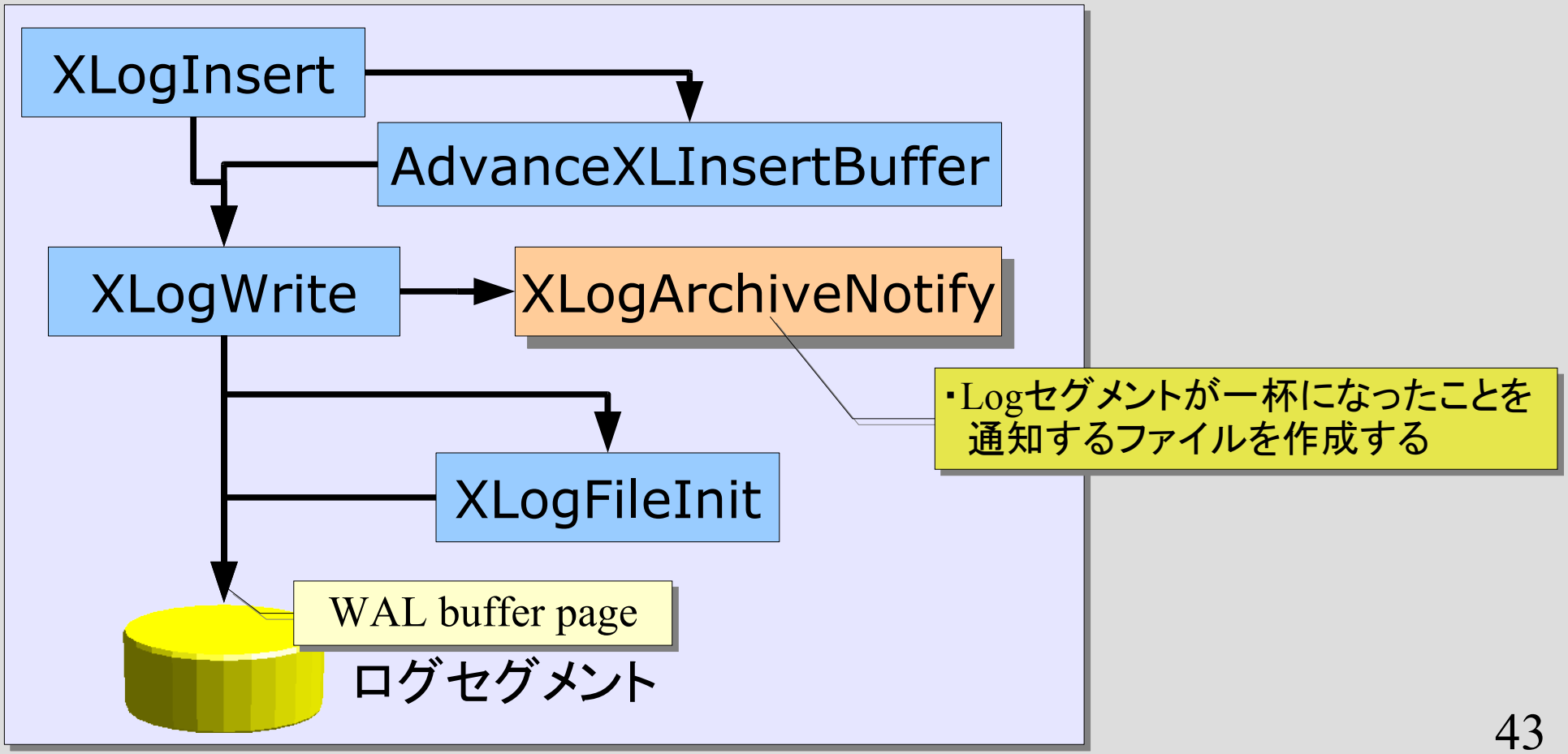
PostgreSQL でのPITRのしくみ

PostgreSQL7.5のLogの中での
Point In Time Recoveryの
改造個所を説明をします

主な改造個所
課題

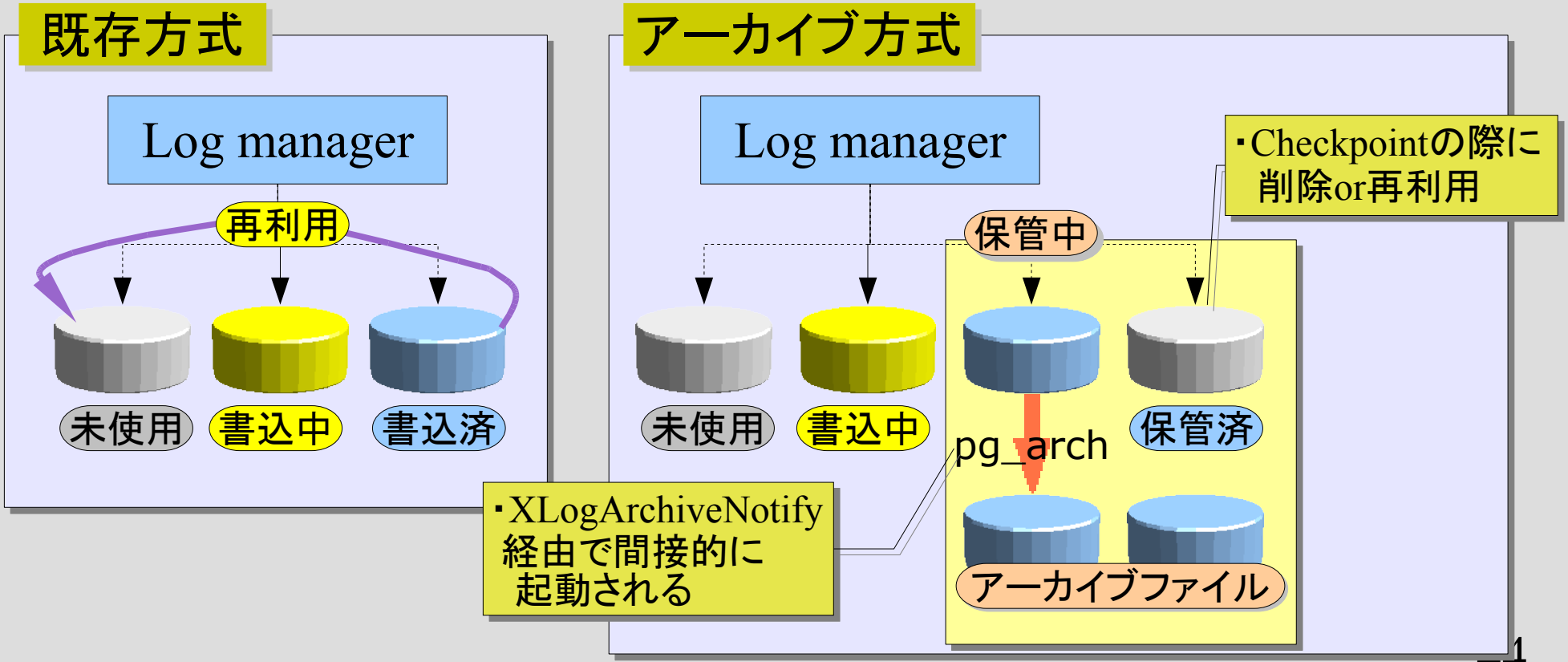
PITRのための改造個所

- 関数レベルでの主な改造個所(下図)
  - 在り処 src/backend/access/transam/xlog.c



PITRのための改造個所

- 関数レベルでの主な改造個所(続き)
  - セグメントファイルの管理方針の変更
    - 単純なリサイクルからアーカイブ方式へ



Log-Recoveryシステムでの課題

- PostgreSQL 7.5では、PITRによって懸案だったroll forward loggingが可能となった
- 以下の課題が残っていると考えられる
  - 重要なファイルの2重化
    - トランザクションログ
    - コントロールファイル
  - バックアップ方式の充実
    - 差分/増分バックアップ
  - RPIT(recovery to point in time)
    - 復元時点の指定方法
      - 時刻指定・TID指定とも分かりにくい
      - タイムライン管理が必要(議論中?)

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

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

お疲れさまでした