Logのしくみ

平成十六年七月廿六日

NTTサイバースペース研究所
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

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
    - 固めて書き出す⇒隘路の軽減

ログとはどのようなモノか

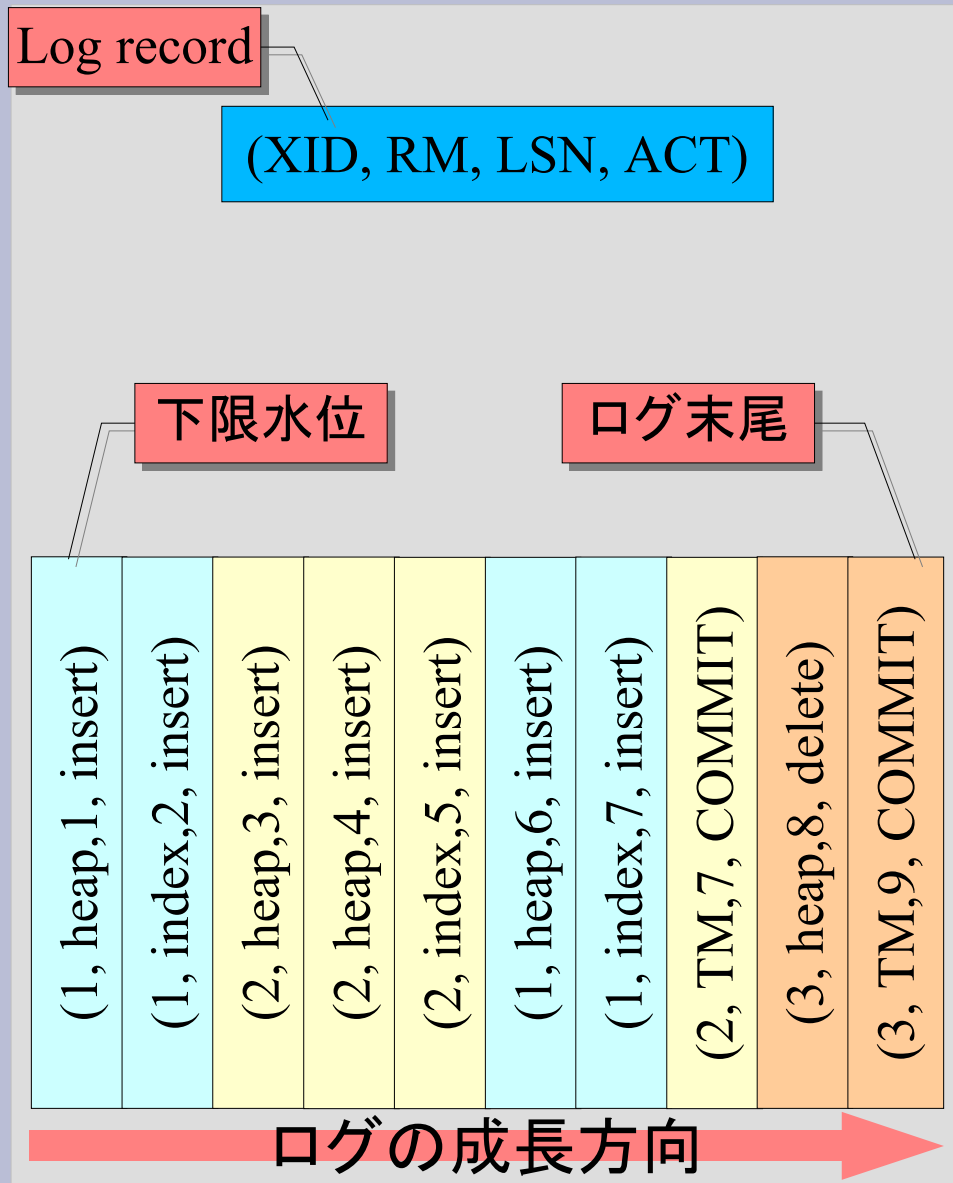
- ログには何が入っているか?
  - どのトランザクションのログか(XID)
  - データベースに対する更新操作(action)
    - 記録方式は、論理・物理の2に大別される
    - 1操作＝1レコード(log record)
  - 更新を実施したサブシステム(RM)
  - それら操作の順序(LSN)

- 取得方式の留意点
  - ログの容量(論理ログは小さく、物理ログは大きい)
  - 巾等(*idempotent*)であること

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
    - これ以上巻き戻さなくても良い地点(下限水位)がある
  - ログを順方向にスキャンする'redoスキャン'
    - コミット済みTrXのログのみ反映する

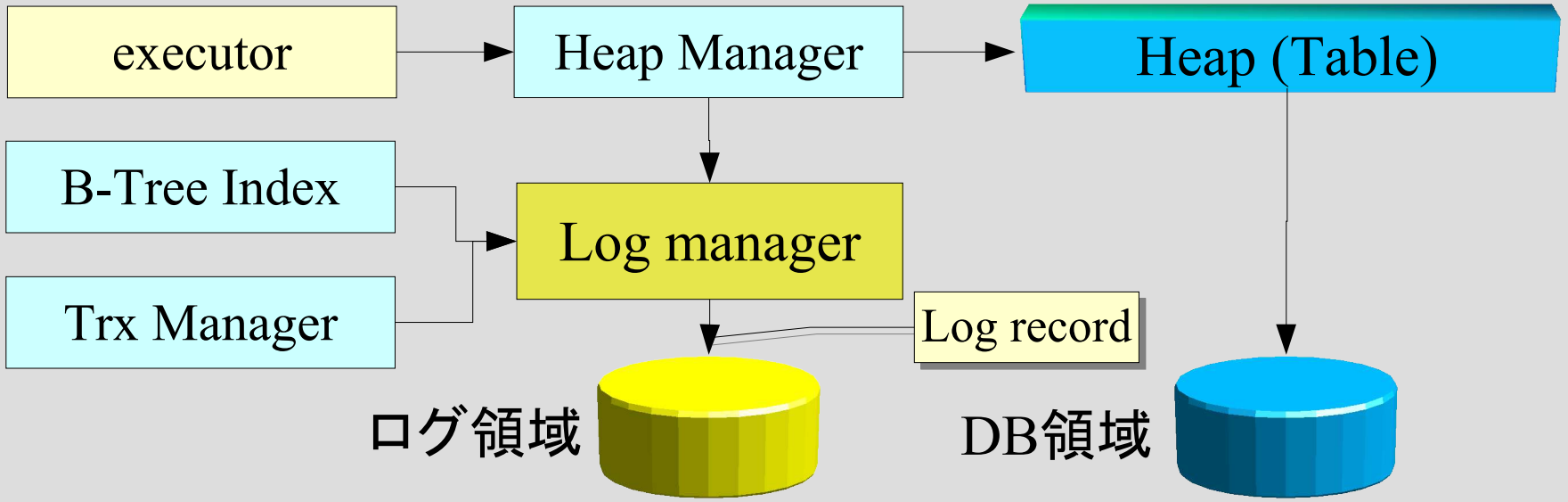
リカバリ動作のイメージ



- 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

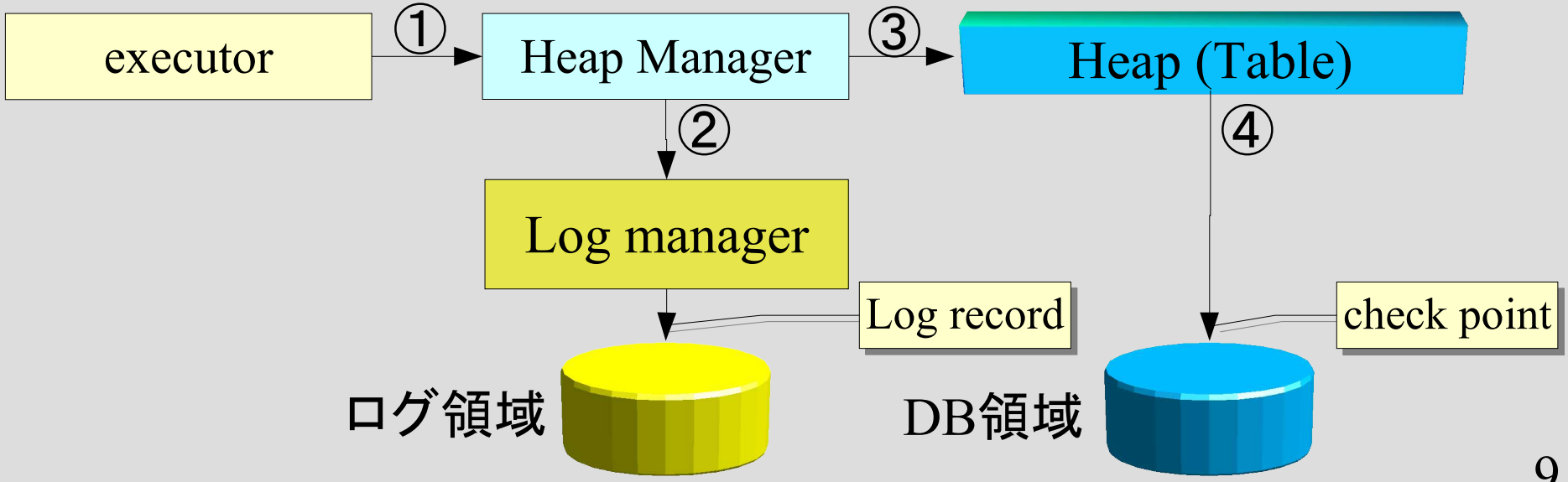
ログへの書き込み

- 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



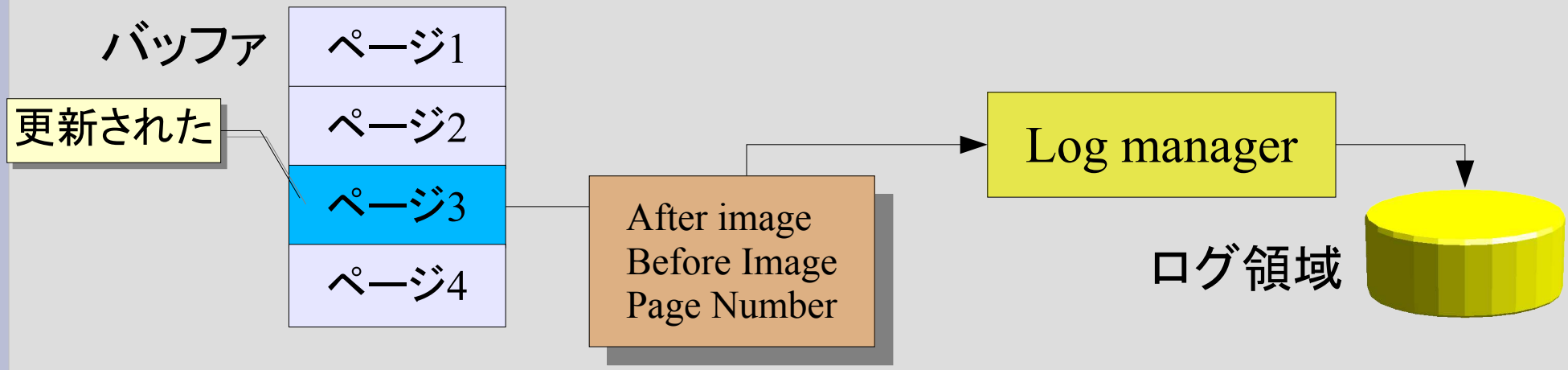
ログへの書き込み：WAL

- いつログに書き込むか?
  - 変更を実際のDB(heap)に反映する前にログに書く
    - 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)

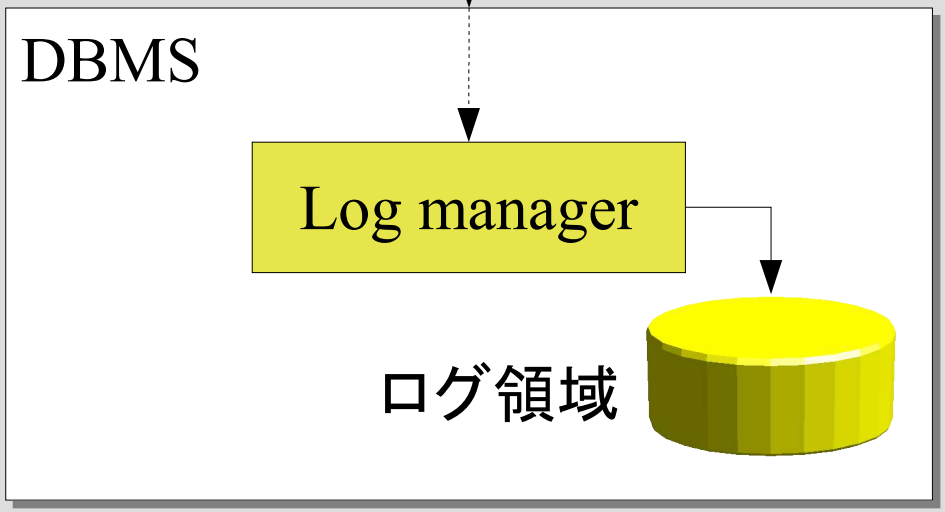


何がログへ書き込まれるか

- 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



何がログへ書き込まれるか

- 各方式の得失を考える観点
  - べき等性：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

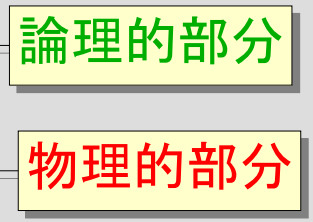
両者を組み合わせた方式が必要

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
- PostgreSQLでも採用されている

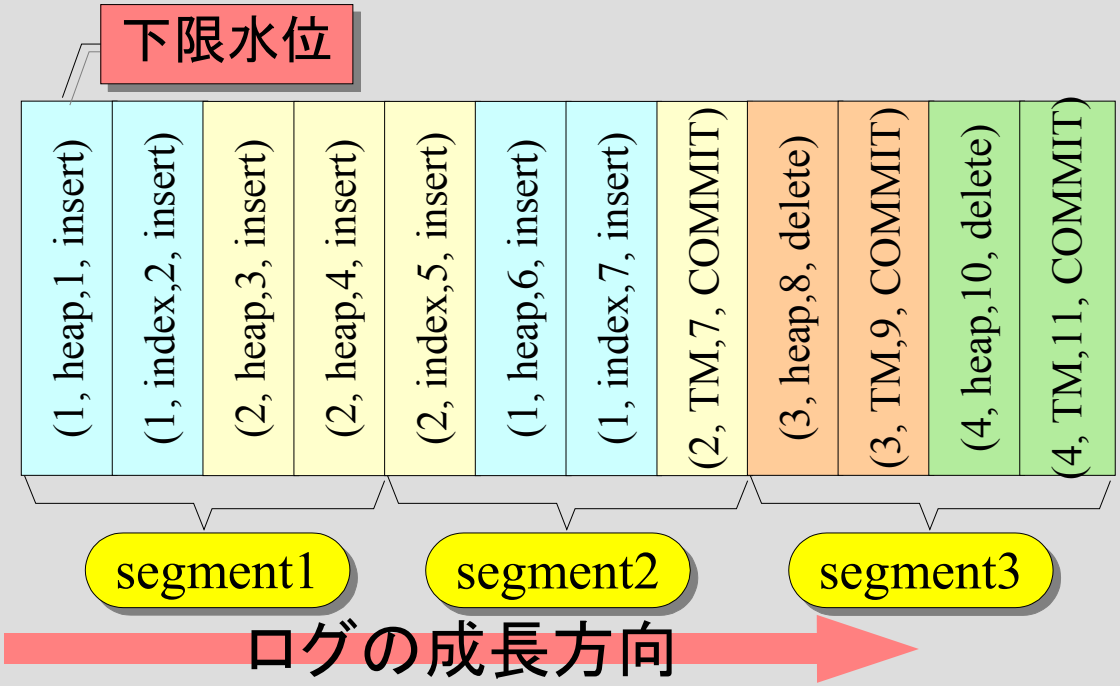
物理論理ログの例

```
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
```



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)

PostgreSQL 7.4 でのLogのしくみ

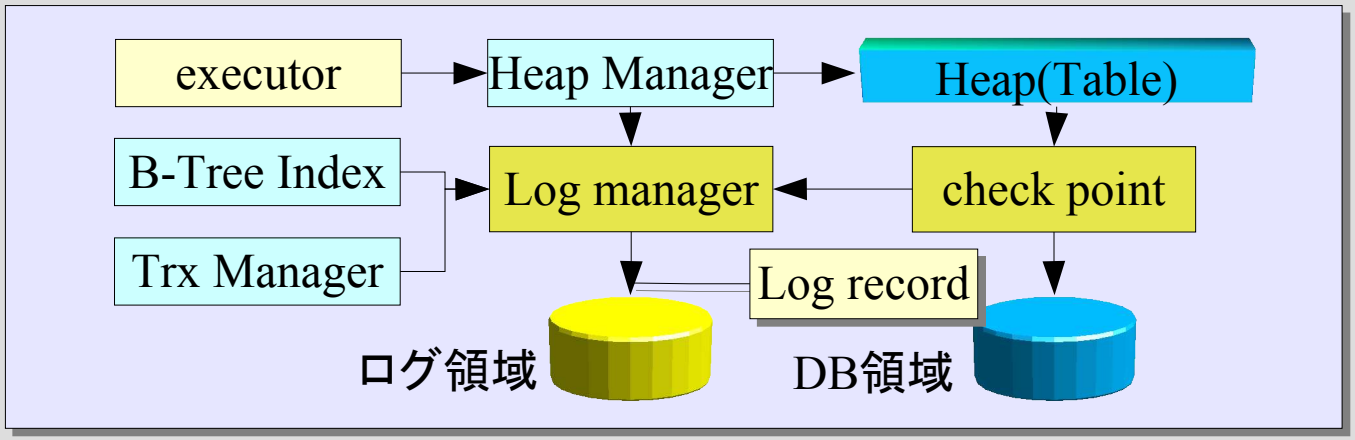
PostgreSQL7.4でのログの
実装の説明をします

大きな枠組み
内部構造
主要データ構造

大きな枠組み

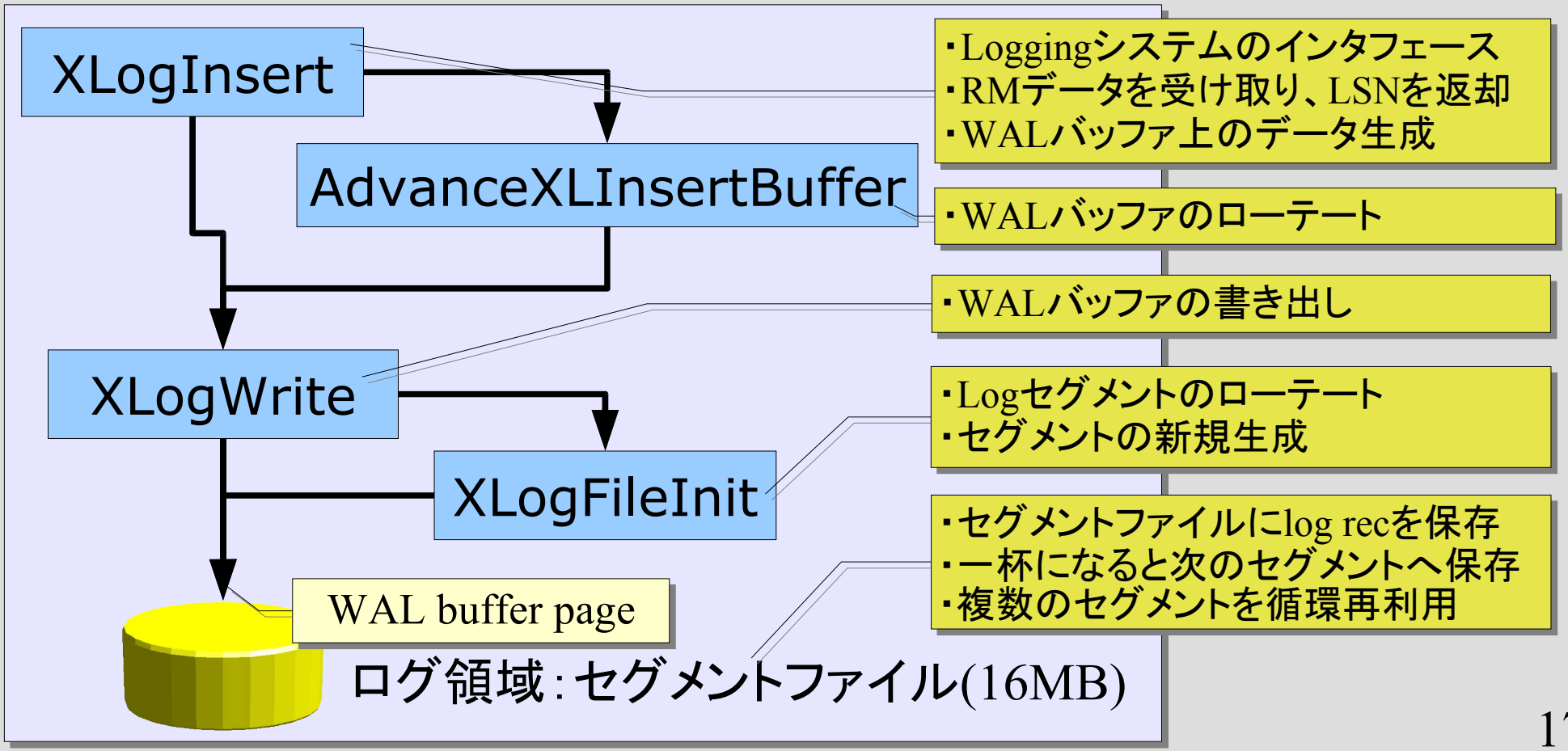
- PostgreSQL内でのLogging systemの位置付け
  - リソースマネージャ(RM)とheapの間にある
  - check pointと協調動作する
- Log資源の管理
  - 物理ログと物理論理ログの混在
  - 可能なら早期に書き込む
  - 複数backendの協調動作(排他制御)

これらの特徴のため
内部の動作は複雑



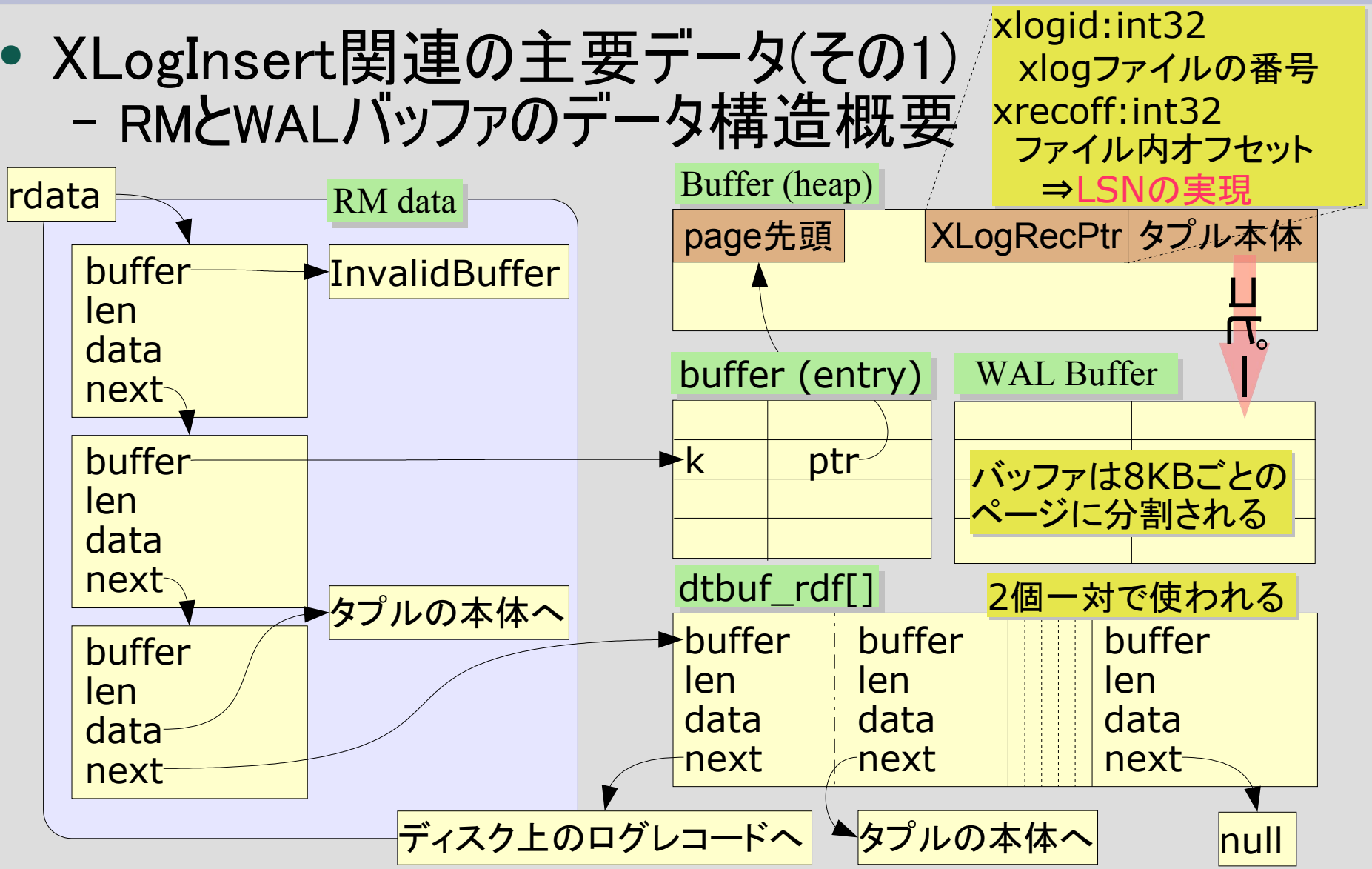
Loggingシステム内部構造

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



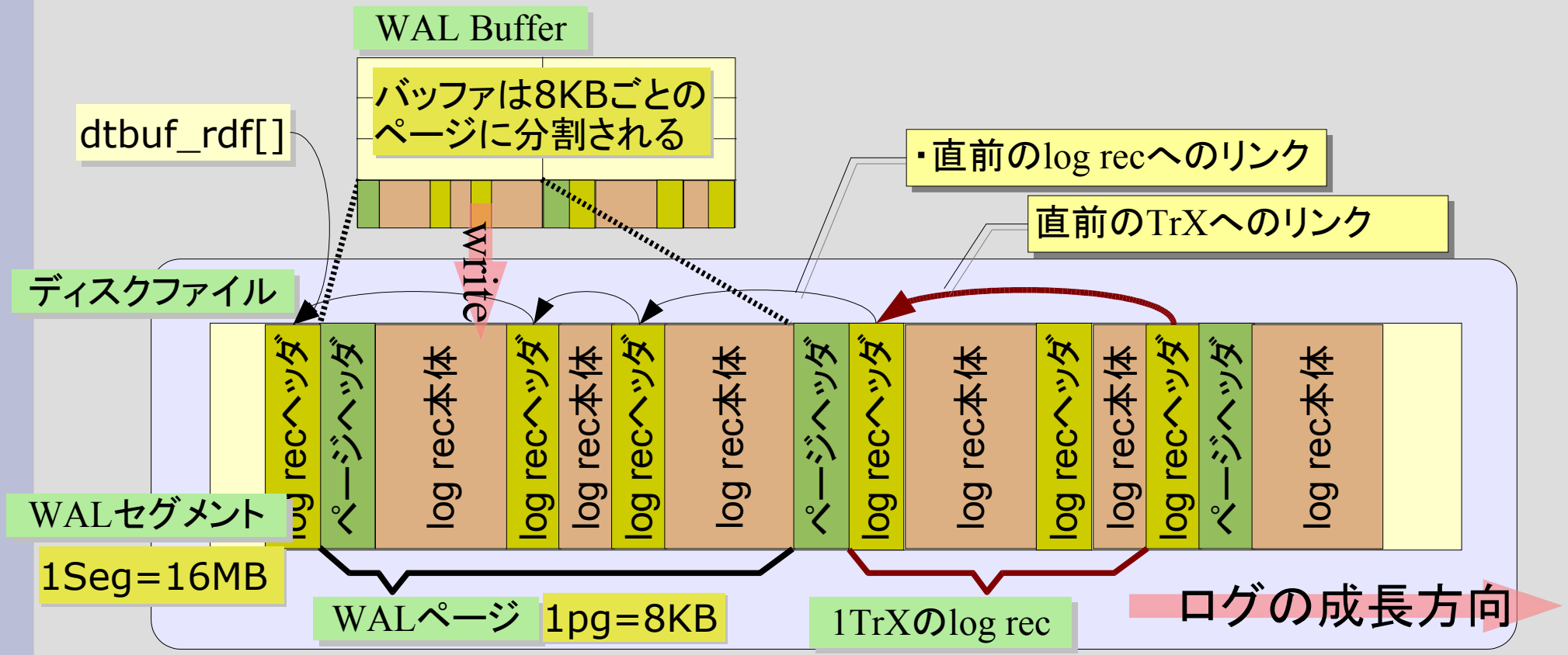
Loggingシステムのデータ構造

- XLogInsert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

- XLogInsert関連の主要データ(その2)
  - 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

- XLogInsert関連の主要データ(その3)
  - ヘッダほか

ログレコードの破損を検出する

XLogRecord

xl_crc	crc64	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

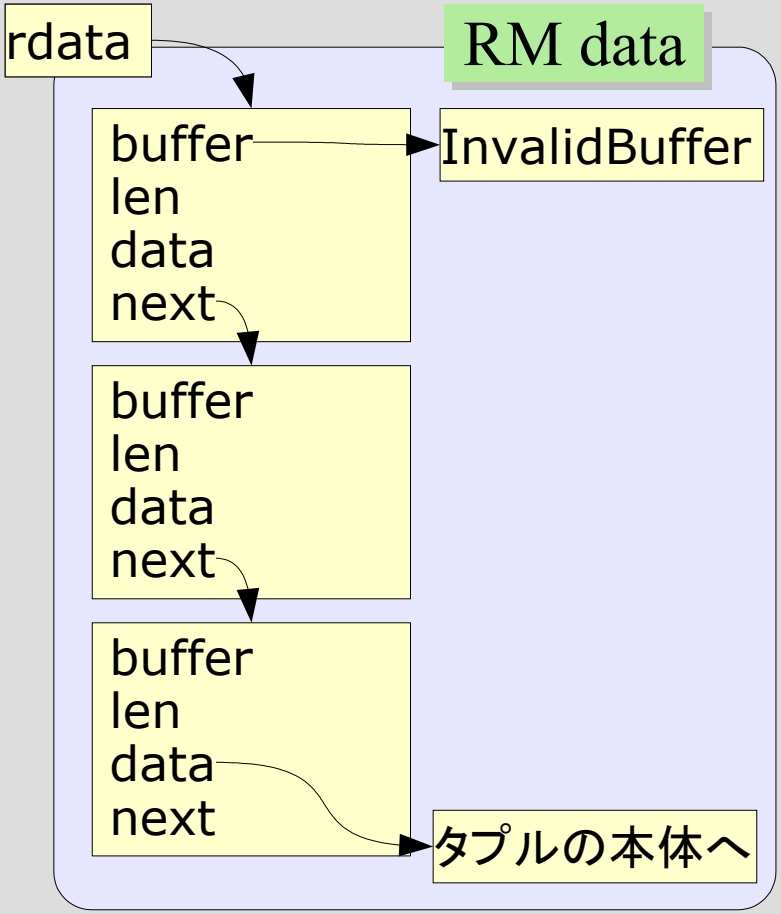
XLogPageHeaderData

xlp_magic	uint16	0xD05A; WALの版を示す
xlp_info	uint16	0; 使われていない?
StartUpId	xlp_sui	0; 全てのbackendで同じ
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

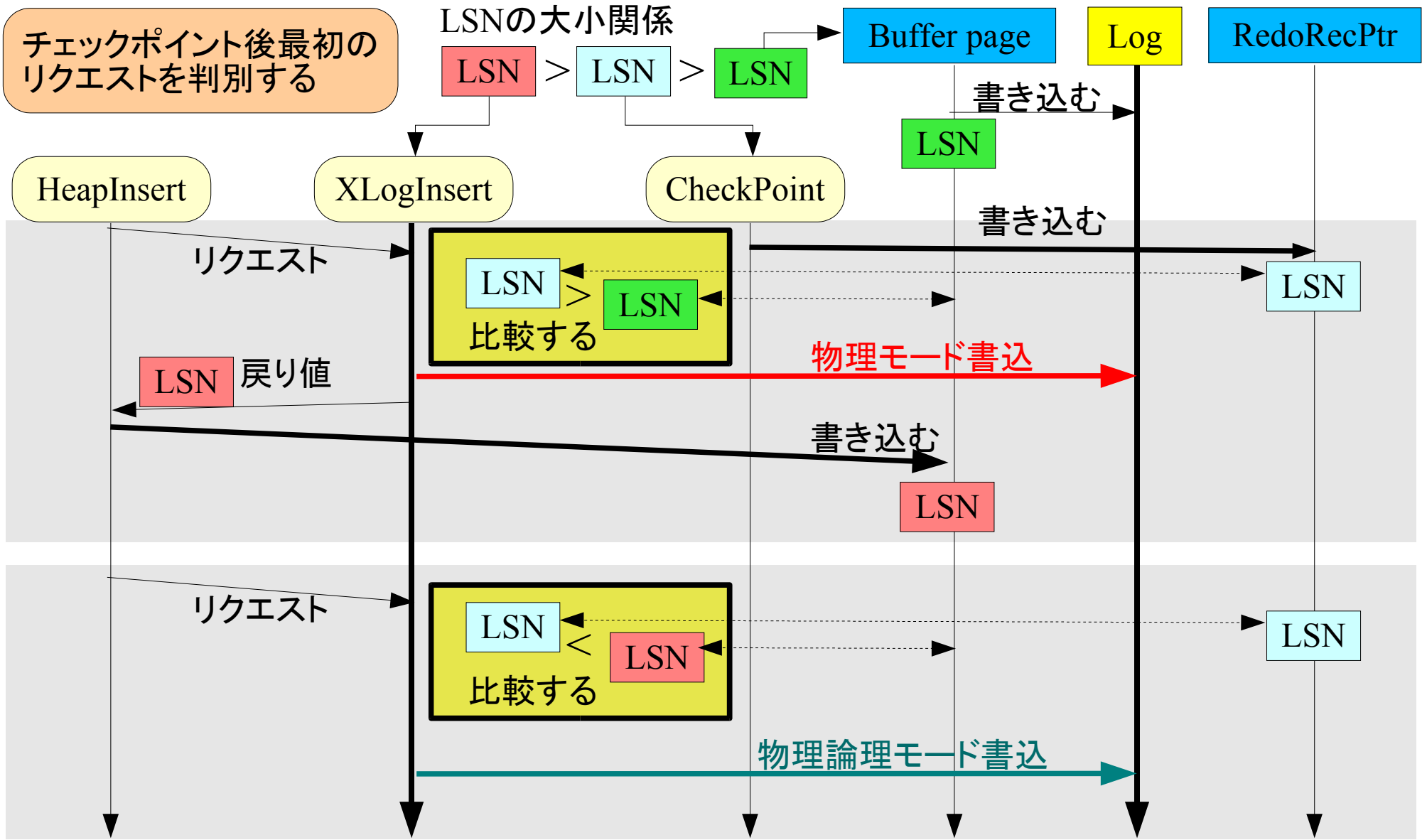
XLogInsertの概要

- 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dr_buf[]など
    - CRC(巡回冗長コード)の生成：誤り訂正&検出
  - 書き込みデータの最新性の確認(後述)
  - 物理ログ/論理ログの判断(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

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



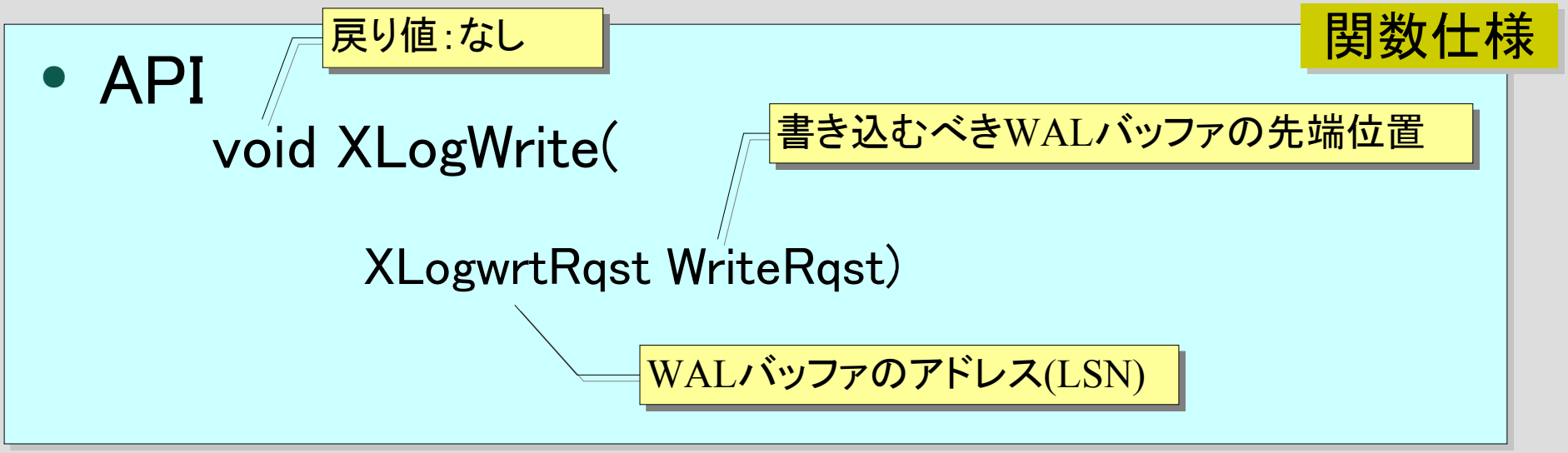
LSN: log serial number

XLogInsertの概要

- ログ書き込みの排他制御
  - DBバッファは排他資源
  - WALポインタ自身も排他資源
    - WALの書き込みオーバヘッドを減らすため極力ロックを獲得しない
    - 各backendはWALポインタのコピーを持つ
      - ロックせずに読めるがDBバッファ/WALポインタの最新性の保証はない
        - 他のbackendがWALポインタを更新して、何時の間にか古くなることがある
        - 書き込むべきページが他のbackendによって更新されることもある
  - 途中までデータを準備し、最新性を確認してからWALバッファに書き込む(楽観的制御)

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



- API

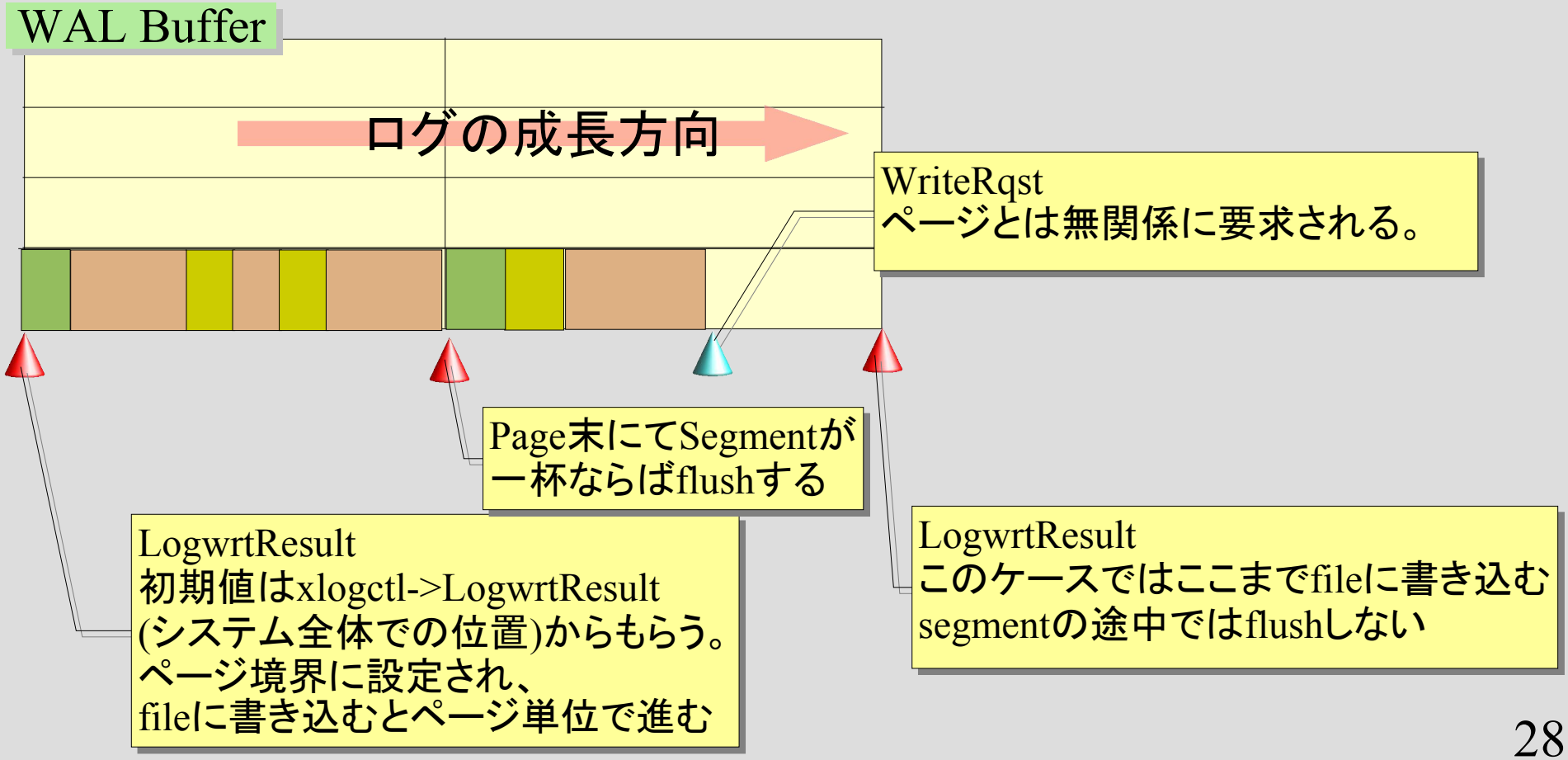
```
void XLogWrite(
        XLogwrtRqst WriteRqst)
```

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

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

引数はない

```
static bool

AdvanceXLInsertBuffer(void)
```

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

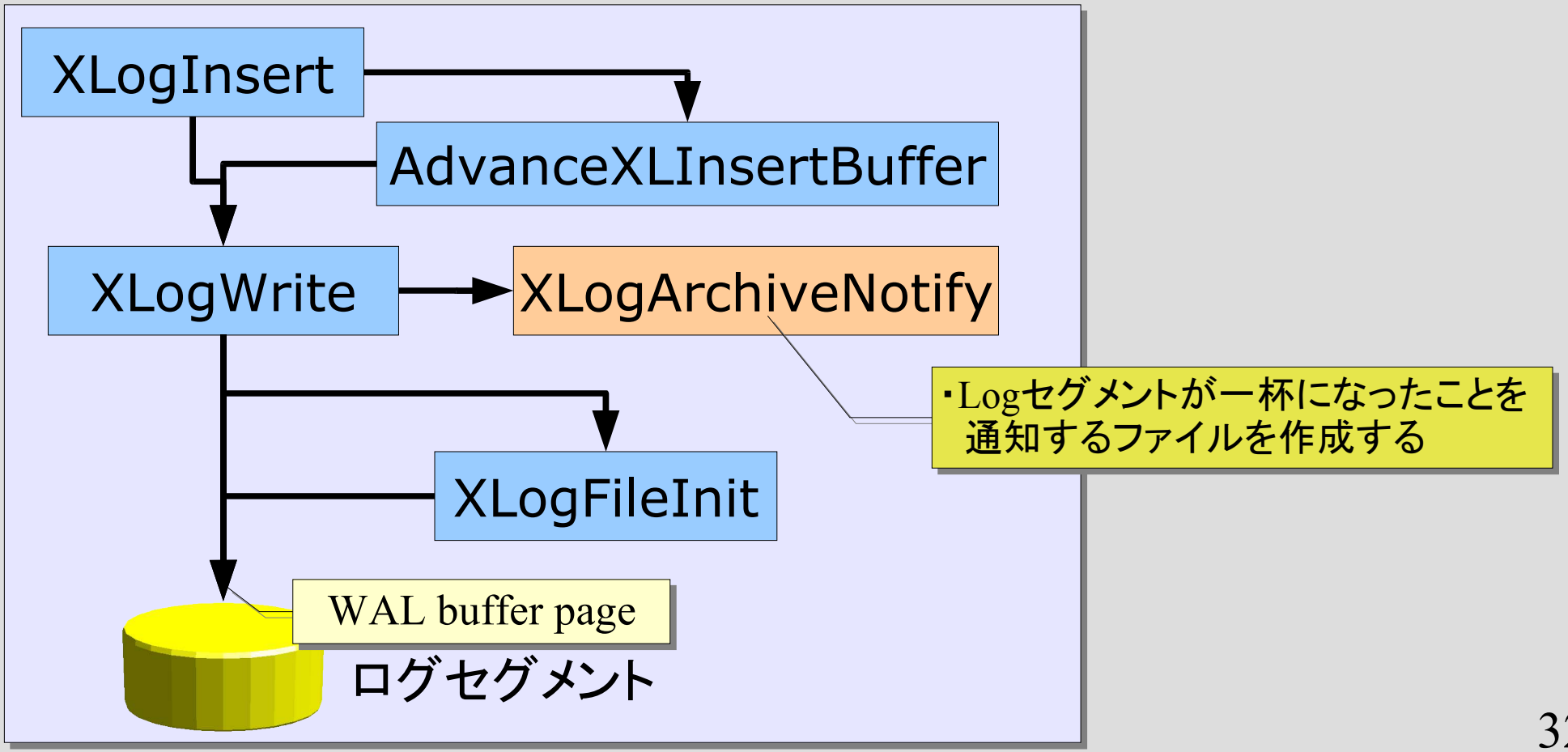
PostgreSQL 7.5 でのPITRのしくみ

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

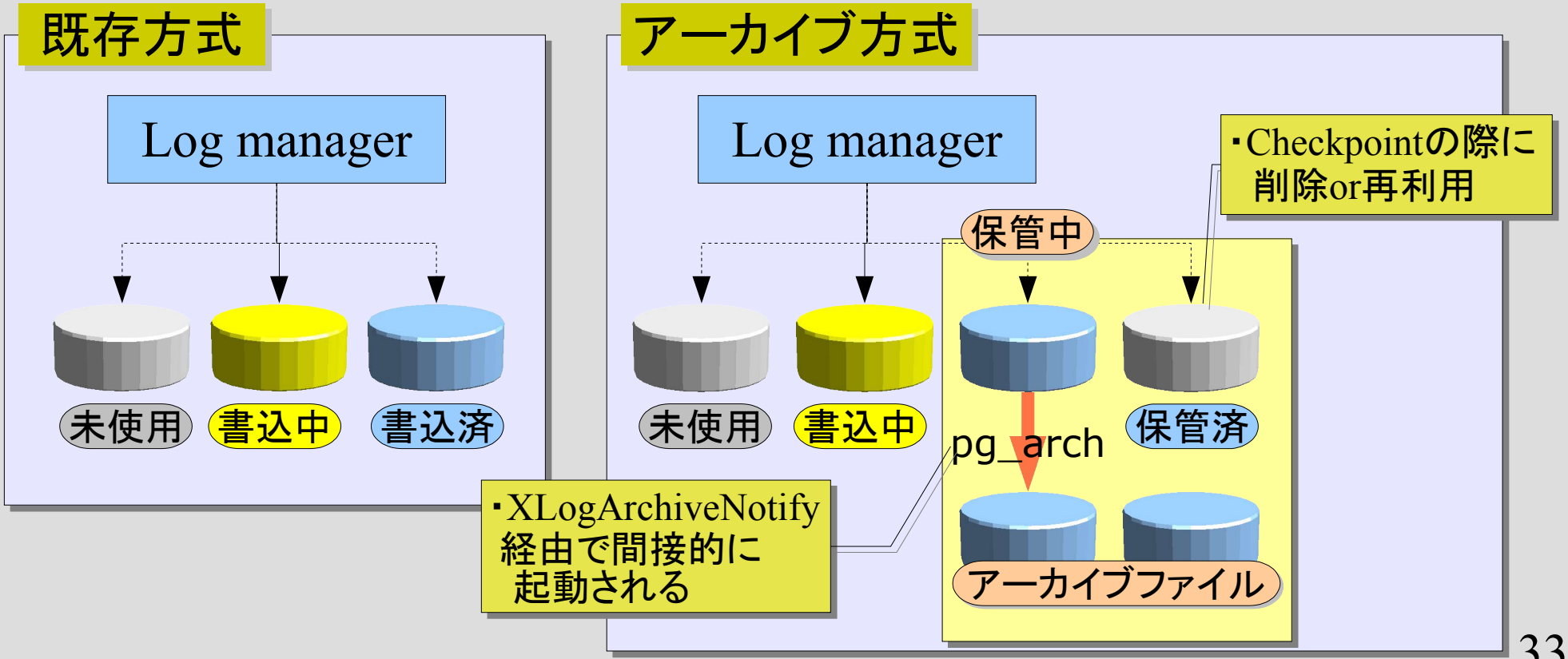
PITRのための改造個所

- 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

参考文献

- 概要レベル
  - 入門書レベルでは以下の本が簡潔
    - 北川 博之, “データベースシステム”, 昭晃堂, 1996.
  - 以下の本は丁寧でお勧めできる
    - Garcia-Molina, J. D. Ullman, J. D. Widom, “Database Systems: The Complete Book”,Prentice Hall, 2001.
- 詳細レベル
  - Gray&Reuterの教科書
    - 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.
    - WALを実装したM. Vadimも読んだらしい

お疲れさまでした

終