# Database

## 第11回：トランザクション処理
## ～障害回復機能～

上智大学理工学部情報理工学科
高岡詠子



2011/12/22

# トランザクションとは？

- Transact
- Transaction
- 作業の論理単位(logical unit of work)

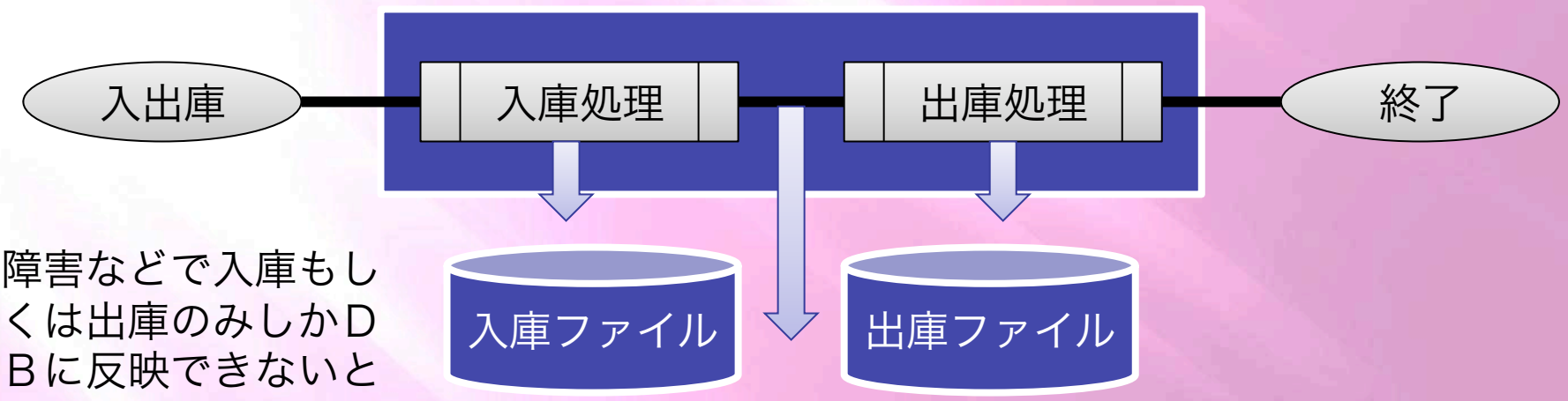
# トランザクションとは？

## ユーザから見たひとまとまりの処理単位

ある企業の在庫管理システムにおける入出庫の処理の一部

仕入れ先から顧客へ直接発送される商品⇒自社で在庫を持たない

入庫と出庫は同時に発生したものとして処理




障害などで入庫もしくは出庫のみしかＤＢに反映できないと大変！

入出庫の登録は切り離すことができない

このような処理の単位（プロセス）をトランザクションという

# トランザクション管理

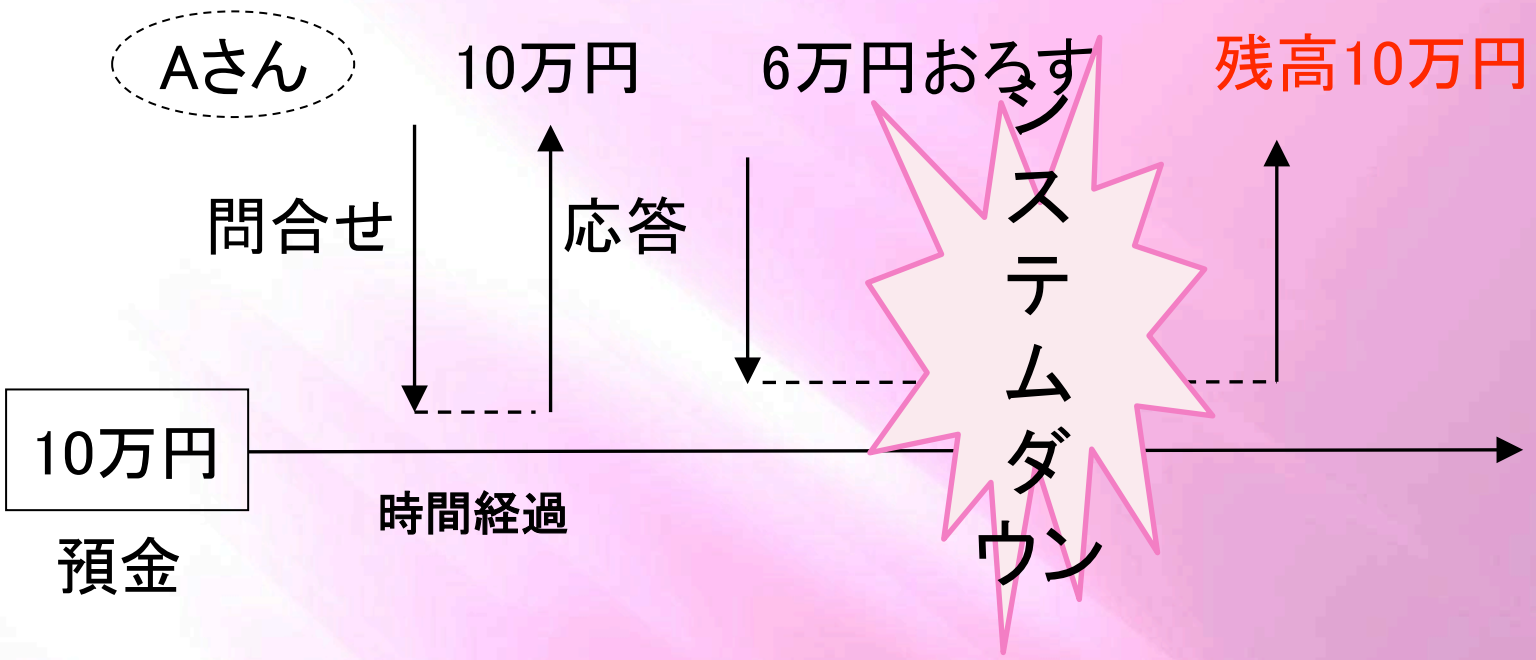
- トランザクションが正常に終了したかしないかでDBに値を反映
- トランザクションが完了すれば正しいデータとしてDBに反映→
- 異常終了したら：途中まで行われた処理はなかったことにする→トランザクション開始以前の状態に戻す→

# コミットとロールバック

- 前回のcommitから現在までの操作をコミットする
- 一度commitすると操作を取り消せない
- MySQL標準は自動コミット（テーブルを更新（変更）する命令を実行すると、直ちにその更新がディスクに格納される）

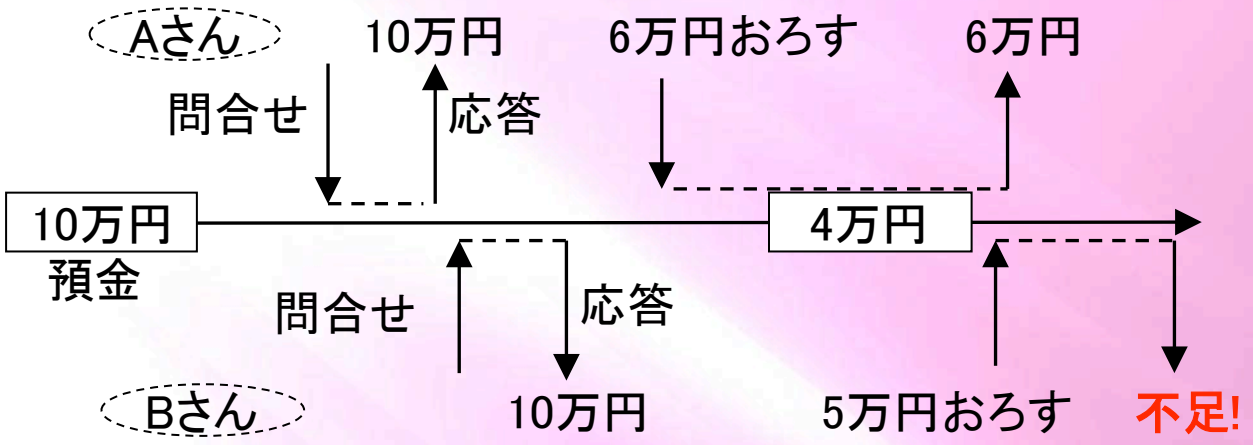
- 前回のcommitから現在までの操作をキャンセルし、データの更新をなかったことにする

# システムダウン




◆システムが復旧したとしても6万円支払ったにも関わらず,
残高は更新されない．つまり10万円のまま
◆障害が回復してもＤＢ内容を元の状態に戻せない
◆障害が回復してもプロセスをどの状態に戻すべきかわからない

# 同一データの同時更新




◆複数のプロセスが並行処理しているとき障害が発生すると他の処理の結果に影響を与える場合が生じる
◆複数のプロセスが同一データの更新を並行的に処理すると，実行順序によってＤＢの内容が異なる（一貫性が保てなくなる）場合が生じる

# トランザクション管理の位置づけ

| DBMSに必須 | DBMSの基本機能 | 機能実現のために |
| --- | --- | --- |
| データ重複排除 | データ独立性の確保 | 正規化 |
| | | ER図 |
| データ管理機能の充実 | データの一貫性 | 外部スキーマ |
| | データ操作の簡素化 | SQL |
| | 機密保護 | |
| データの利用簡素化 | | 機密保護機能 |
| | 障害時復旧 | |

2011/12/22

# トランザクション処理の内容：ACID特性

- 1つの処理についての制約
  - 性(A )：分割できない処理
  - 性(C )：処理内容が処理の順序や終了状態に関係なく保証されること
- 複数の処理間の関係についての制約
  - 性(I )：他の処理から独立していること
    - 同時処理したときと逐次処理したときとの結果が同じ
    - 処理途中のデータが他の処理から見えないこと
  - 性(D )：障害等に対して耐久性があること
    - 処理終了後のDBの内容は障害等に影響されずに不変に保たれる
    - 一度コミットした処理は絶対に消えない

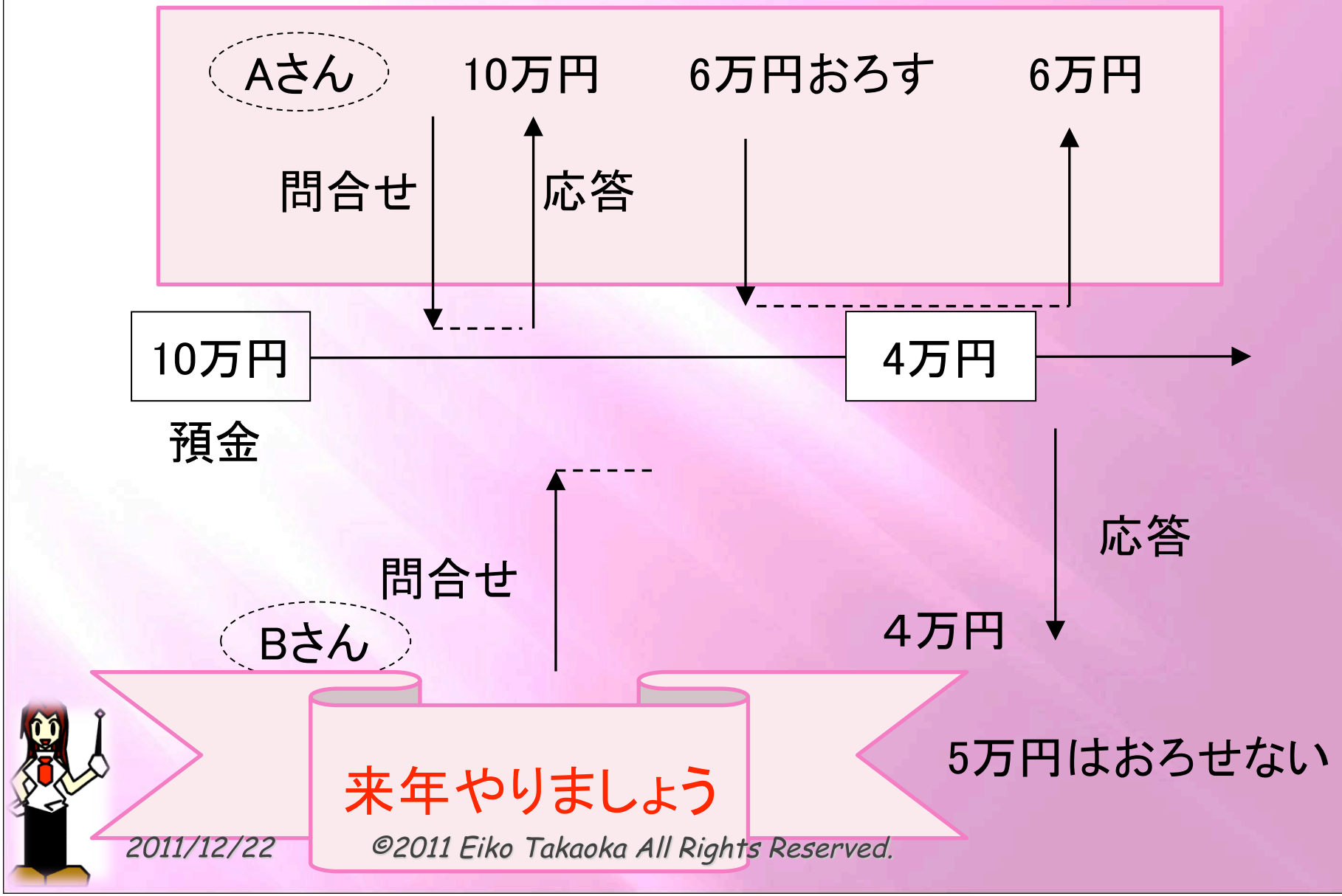
# ACID特性を満たすために

- 各トランザクションにコミュニケーション機能をもたせる？
  →不特定多数のトランザクションが走るときコミュニケーション制御が複雑になる。
  →共有されるDB側に制御機能をもたせる

# ACID特性を実現する制御機能

- 制御
  - 複数のプロセスが並行処理を行い
  - 同じデータを更新しても異常が生じないようにする
  - DB側に排他制御（他のトランザクションのアクセスを排除する制御）を設ける
  - 「　　性」「　　」を確保する

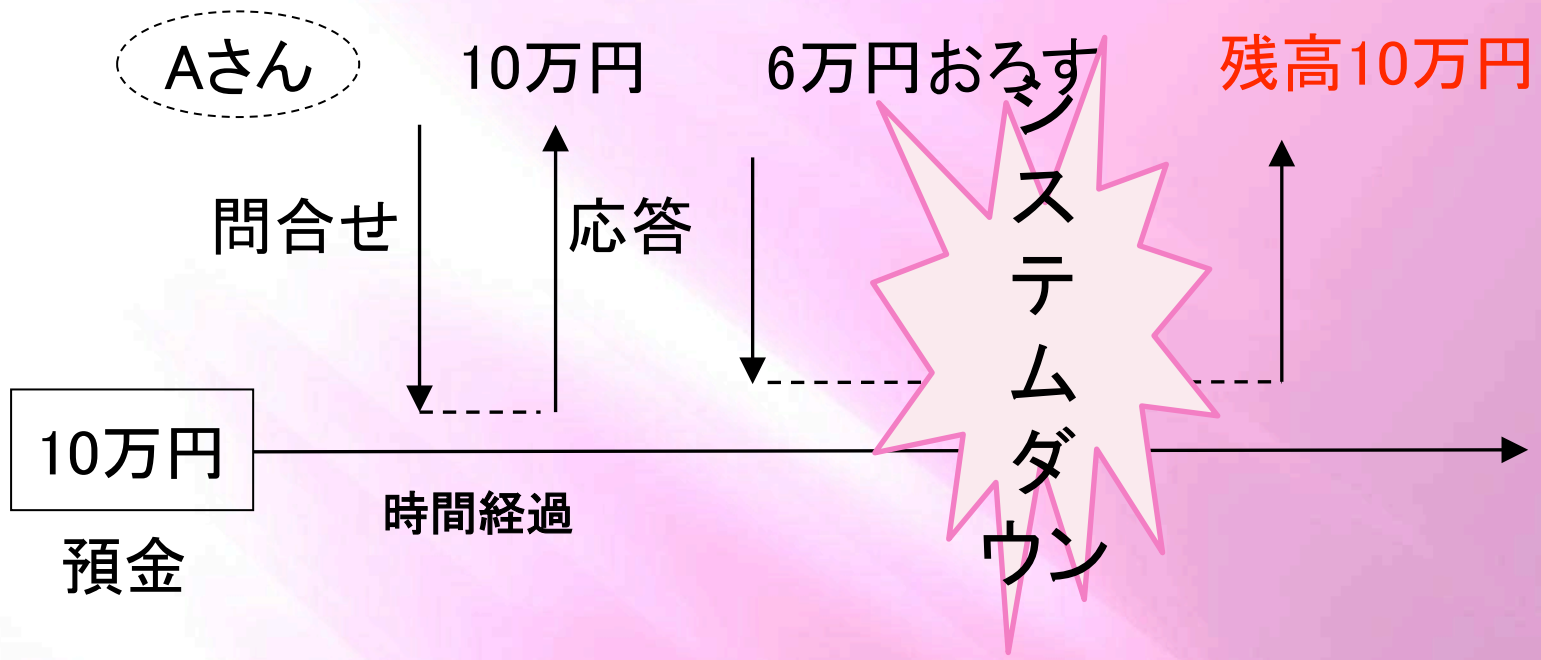
# 同一データの同時更新



# ACID特性を実現する制御機能

- 制御
  - 障害が発生したときにもとの状態に復旧するための制御
  - ログなどでアクセス記録を残す
  - 「　　性」「　　性」を確保

# 例：システムダウン




◆処理が正常に終了しないのでＤＢに反映しなければよい

# 障害の種類とリカバリ(障害復旧)

- 障害
  - トランザクション障害
    - アプリケーションプログラムのエラー
    - 例外処理
  - システム障害
    - ハードウェア，ソフトウェア，ネットワーク等の問題によるシステムダウン
  - メディア障害
- リカバリとは:

# リカバリのための方式

- リカバリのための方式
  - ログにしたがってリカバリを行う方式
    ⇒
  - ログにしたがったリカバリを行わない方式
    ⇒
- ログ：リカバリのための情報格納
- 実行前と実行後のデータをログとして残しておく
  ⇒データベースバッファがディスクに書き出されるまえにログを書き出す

# Undo/redo方式

- 保管しているログを使ってどうリカバリするか？
  - ログをトランザクションの始めから終わりまで読む
  - バックアップからログを適応してチェックポイントの状態まで戻して，そこから再度障害直前まで処理をやりなおす
  - もう一度
  - ログを元に，処理を元に戻す
  - 実行した操作を

# Undo/Redoの違い

- REDO（ロールフォワード）のためのログ
  - 更新命令そのもの（どのような更新が行われたか）が保存される
  - レコードのコピーではないことに注意！！
  - 定期的なバックアップと合わせて使う
- UNDO（ロールバック）のためのログ
  - そのレコードの更新前の状態を保存（その時点でのレコードのコピー）
  - トランザクションが完了しなかった場合この「更新前の状態」に戻す

# データの更新手法

- 方式
  - ディスクへDBの書き込みはコミットの前後いずれでもよい
  - ディスクへDBの書き込みの前にログをディスクに書き込む（write ahead logging）
- 方式
  - トランザクションがコミットするまで、更新情報をすぐにはディスクに反映しない手法
  - トランザクションが短時間で更新の少ない場合に適する
- 方式
  - すべての更新をDBに反映した後にのみコミット
  - redoは不要
  - トランザクションが長時間で更新が多い場合に適する

# チェックポイント

- ログのためのエントリ
  - トランザクションを一時的に停止する
  - ログバッファをディスクに書きだす
  - データベースバッファをディスクに書き出す
  - チェックポイントというログを書き込み，ログバッファをディスクに書き出す
  - 一時的に停止していたトランザクション再開
- 更新された全てのブロックを定期的にログとして記録する
- チェックポイント以前にコミットされたトランザクションに対しては障害発生時にredo不要
- チェックポイントの期間設定について
  - ○分毎　あるいは
  - ○個のトランザクションがコミットする毎

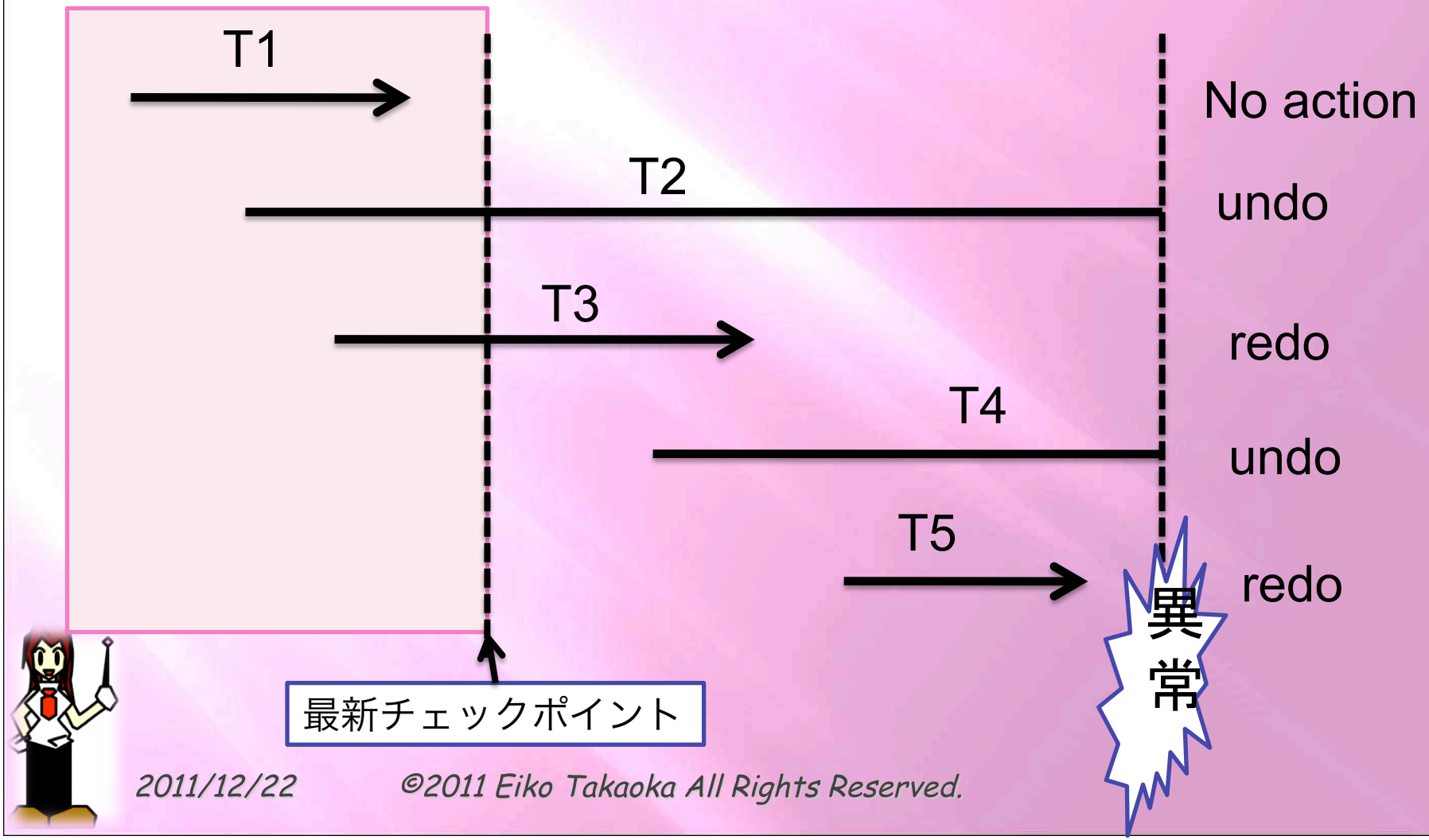
# ページバッファをいつディスクへ書き出すかを指定するモード

- steal/ no steal
  - コミット前に更新をDBに書く:
  - コミット前に更新をDBに書かない:
- force/ no force
  - コミット時にDBをすぐ更新:
  - 更新しなくても良い:

- リカバリの容易さを考えると
- 通常のDBMSは
  - にしないと巨大なバッファ空間が必要
  - にすると、更新頻度の高いページを毎回ディスクに書き込む必要がない
  - I/Oコストを低減できる

効率が良い

# トランザクションの種類ごとに対する処理

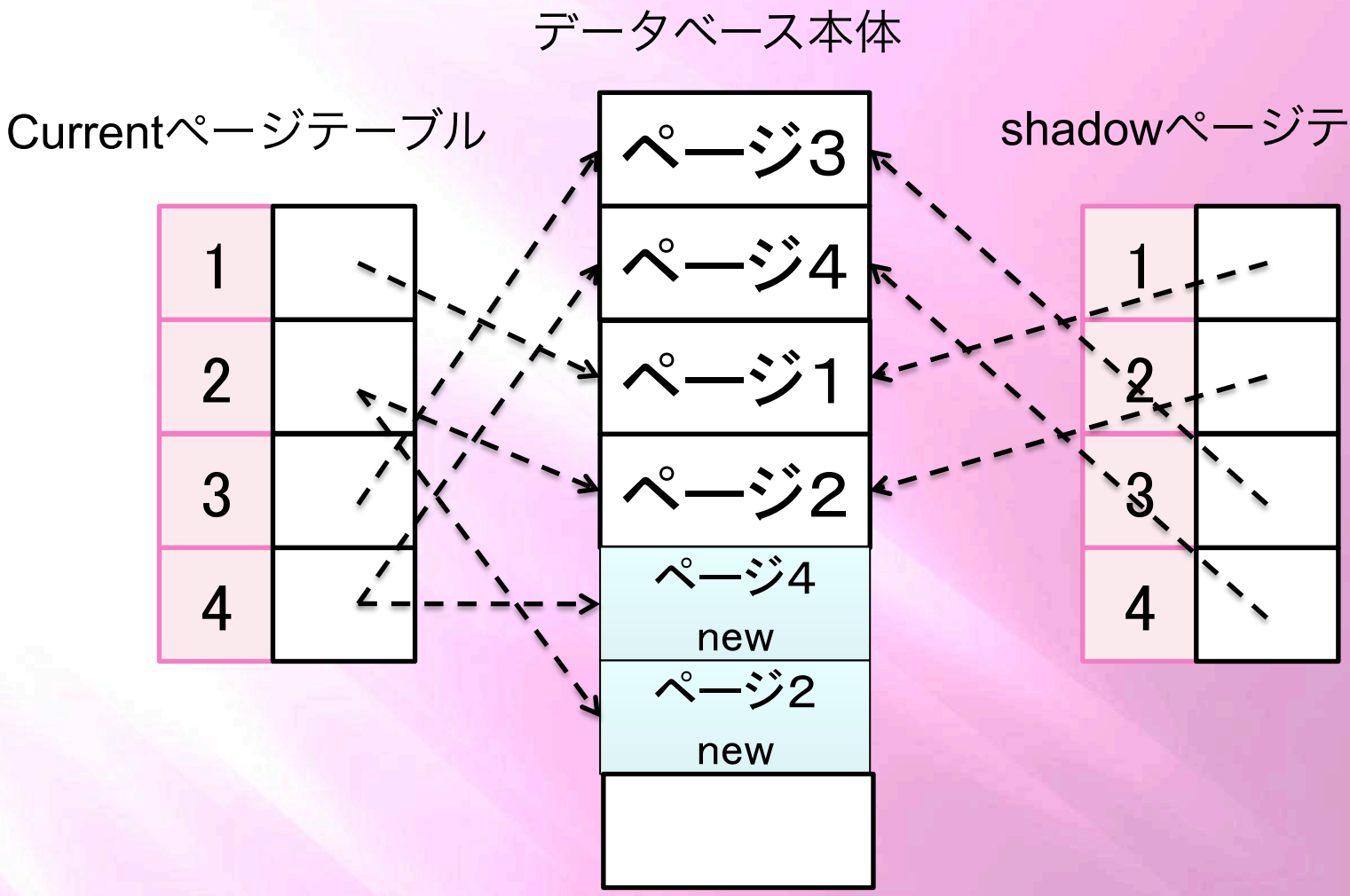


# ログに従ったリカバリをしない方式

- シャドウページング方式
  - ページ：コミットされた後のオリジナルページ（ここまでは戻せる）
  - ページ：新たな更新を行う対象

# Currentページテーブル（新バージョン参照用）<br>shadowページテーブル（旧バージョン参照用）

- トランザクション実行開始時
  - CurrentにShadowをコピー
- 書き込み時：currentのエントリpのさすページ（オリジナルページ）のコピーを作り実際の更新はこのコピーページに対して行われる（オリジナルのページは更新されない）
- 同一DBに対する異なる2つのバージョン
- コミット時
  - DBバッファをディスクへ書き出す
  - Currentを書き出す
  - Currentが新たなShadowページになる
- アボート時
  - コピーページを開放
  - Currentを破棄




# シャドウページ方式の利点／欠点

- 利点：リカバリが簡単
- 欠点
  - ページが断片化してデータアクセスの効率が悪くなる
  - 古いページを再利用するためのガーベッジコレクションが必要
  - マルチユーザ環境に適用する際に同時実行制御が難しい
  - 一時的にディスクを多く使用する

# 即時更新・遅延更新

- 即時更新(immediate update)
  - トランザクションがコミットポイントに達する以前にDBが更新される場合がある
- 遅延更新(deferred update)

# バックアップ

- メディア障害に対しての備え
  - データベース全体
  - ログ
- ログのサイズ＜＜データベース全体のサイズ
  - ログだけのバックアップを頻繁に行う
  - データベースのバックアップ＋ログを利用したredo操作を行う

効率が良い