Database

第12回:トランザクション処理
～同時実行制御～

上智大学理工学部情報理工学科
高岡詠子

No reproduction or republication without written *permission*.
許可のない転載、再発行を禁止します

©2012 Eiko Takaoka All Rights Reserved.

今日の授業

- 障害回復機能の復習
- 同時実行制御

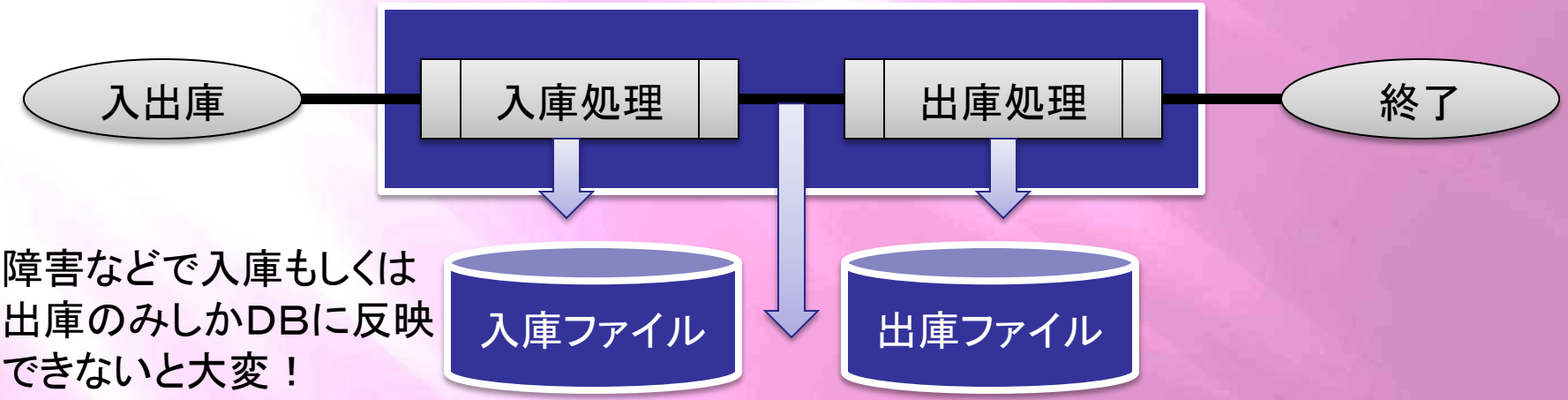
©2012 Eiko Takaoka All Rights Reserved.

トランザクションとは？

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

©2012 Eiko Takaoka All Rights Reserved.

トランザクション管理

トランザクションとはなんでしたか？

ユーザから見たひとまとまりの処理単位

- 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→　**コミット**
- 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→　**ロールバック**

©2012 Eiko Takaoka All Rights Reserved.

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
データの利用簡素化	機密保護	同時実行制御 機密保護機能
	障害時復旧	障害回復機能

©2012 Eiko Takaoka All Rights Reserved.

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
  - 分割できない処理
  - 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

原子性(Atomicity)

一貫性(Consistency)

- 複数の処理間の関係についての制約
  - 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
  - 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

独立性(Isolation)：

耐久性(Durability)：

©2012 Eiko Takaoka All Rights Reserved.

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

©2012 Eiko Takaoka All Rights Reserved.

ACID特性を実現する制御機能

- 障害回復制御
  - 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
  - 「原子性」「耐久性」を確保

復習しましょう

©2012 Eiko Takaoka All Rights Reserved.

トランザクション処理のとき

- 実行前と実行後のデータをログとして残しておく⇒Write Ahead Logging
  データベースバッファをディスクに書き出すのではなくログのみをHDに書き出す
- コミットされたトランザクションの更新情報は、バッファ内に保持され、あるタイミングでデータベースへ書き込まれる
  ⇒これがチェックポイント

©2012 Eiko Takaoka All Rights Reserved.

ログによる障害回復

- DBへの書き込み発生時
  - データの更新前後の内容を必ずログに書き込む
- REDO：データ更新後の内容
- UNDO：データ更新前の内容
- ログをいつとるか？　Write Ahead Logging
  - ログを取る前にデータの更新をすると？
    UNDOできない
  - コミットしたトランザクションのログがないと？
    REDOできない

©2012 Eiko Takaoka All Rights Reserved.

WALの基本動作

A) トランザクションログをWALバッファに書き込み、DBバッファ内のデータを変更
チェックポイント が実行されるまでHDには反映されない

B) コミットされた時点でWALバッファの内容をWALログファイルに書き込む

C) チェックポイント時
1. ログバッファをディスクに書き込む
2. DBバッファ内のデータをHDに書き込む
3. WALログファイルを新規作成

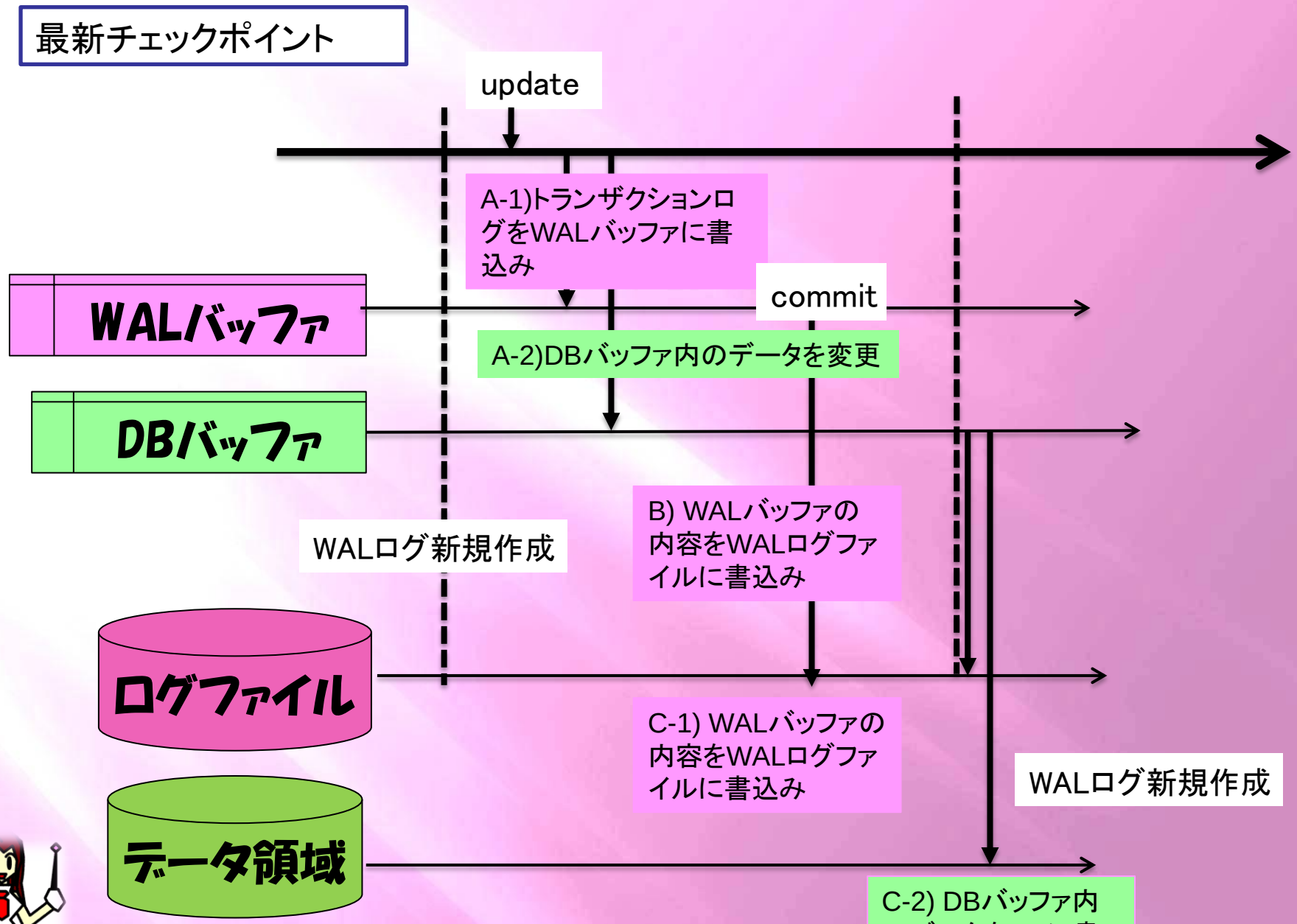
文献：WALとは： http://www.interdb.jp/techinfo/postgresql/p-2-09.html

©2012 Eiko Takaoka All Rights Reserved.

UNDO/REDO方式でのトランザクション処理

- トランザクション開始をログバッファに書きこむ
- 書き込みが起こったらログを書いてからデータを書き込む
- コミット時はWALバッファの内容をWALログファイルに書き込んでからDBバッファをHDに書き込む

©2012 Eiko Takaoka All Rights Reserved.



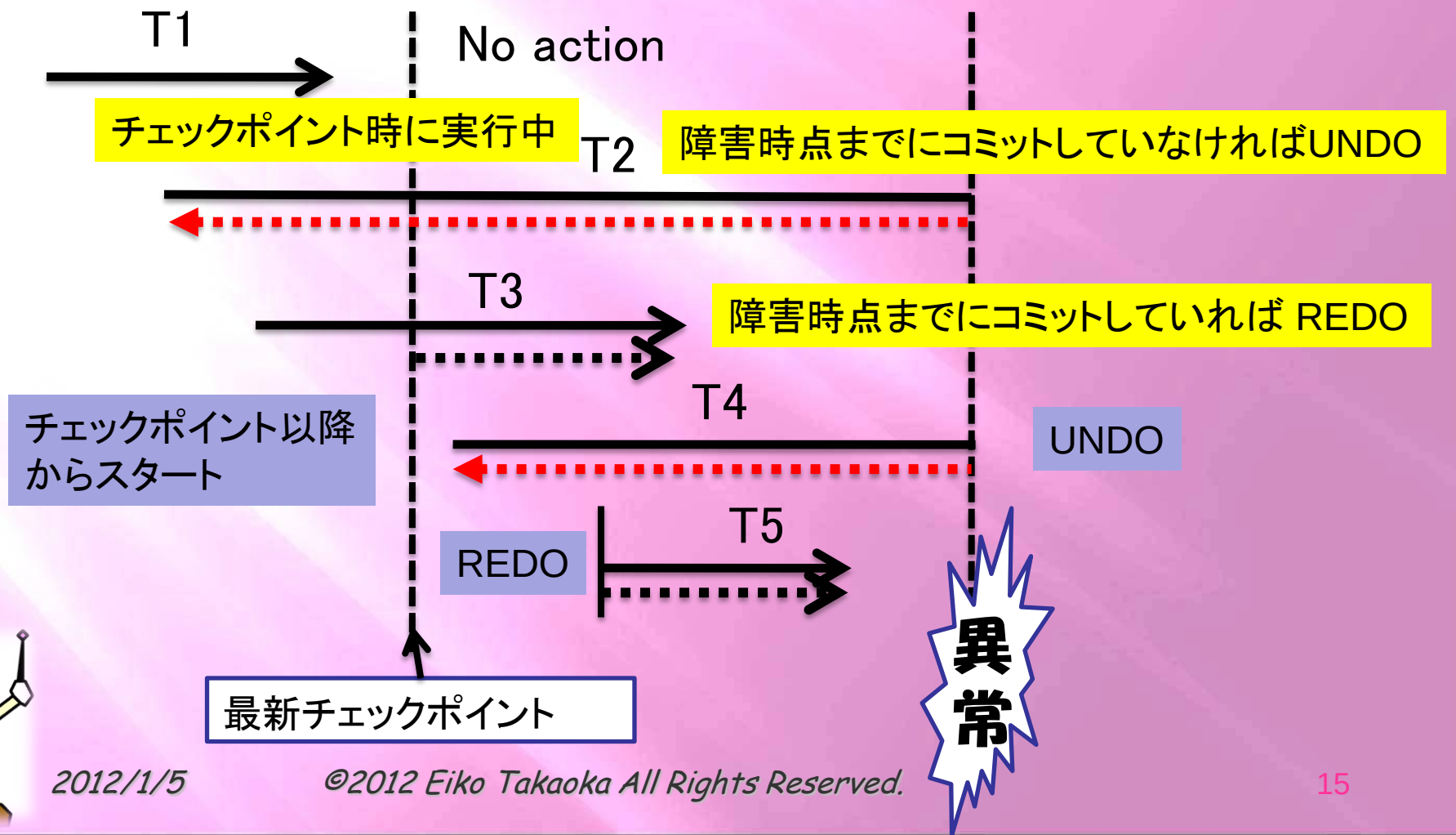
©2012 Eiko Takaoka All Rights Reserved.

障害回復の方針

- チェックポイント時点で実行中のトランザクションは，チェックポイント以前のログが必要
- チェックポイント時に実行中のトランザクションは，障害時点までにコミットしていれば REDO され，それ以外は UNDO
- チェックポイント以降からスタートしたトランザクションチェックポイント以降のログをチェック

©2012 Eiko Takaoka All Rights Reserved.

トランザクションの種類ごとに対する処理



©2012 Eiko Takaoka All Rights Reserved.

今日の授業

- 障害回復機能の復習
- 同時実行制御

©2012 Eiko Takaoka All Rights Reserved.

ACID特性を実現する制御機能

- 同時実行制御
  - 複数のプロセスが並行処理を行い(データ同時実行性)
  - 同じデータを更新しても異常が生じないようにする(データ整合性)
  - 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
  - 「独立性」「一貫性」を確保する

今日はこれ

同時実行の利点

- CPUとI/Oは並列に実行できるのでディスクとCPUが暇な時間の量を減らせる.
- 短いトランザクションと長いトランザクションを交互に実行すると，通常は短いトランザクションのほうが早く終わり，長いトランザクションの後回しになることは少ない

©2012 Eiko Takaoka All Rights Reserved.

同時実行とパフォーマンス

- パフォーマンスとデーター貫性のトレードオフを考える必要がある
- パフォーマンスを表す用語
  - スループット：与えられた時間の中で処理されるトランザクションの数
  - 応答時間：トランザクションの実行に平均的に要する時間

©2012 Eiko Takaoka All Rights Reserved.

直列可能性

- トランザクションの分離モデル
- 複数のトランザクションが同時ではなく1つずつ(直列に)実行された結果と同じになること
  - あるトランザクションが問い合わせ実行中の表に対して他のトランザクションは挿入はできない
- 異常回避のために単純に順番に実行(直列実行)するか？

↓

- 結果は正しくても並列性は向上しない

↓

- 直列可能性を持つ(直列実行した場合の結果のどれかになる)ように複数のトランザクションを並列実行させる必要がある

©2012 Eiko Takaoka All Rights Reserved.

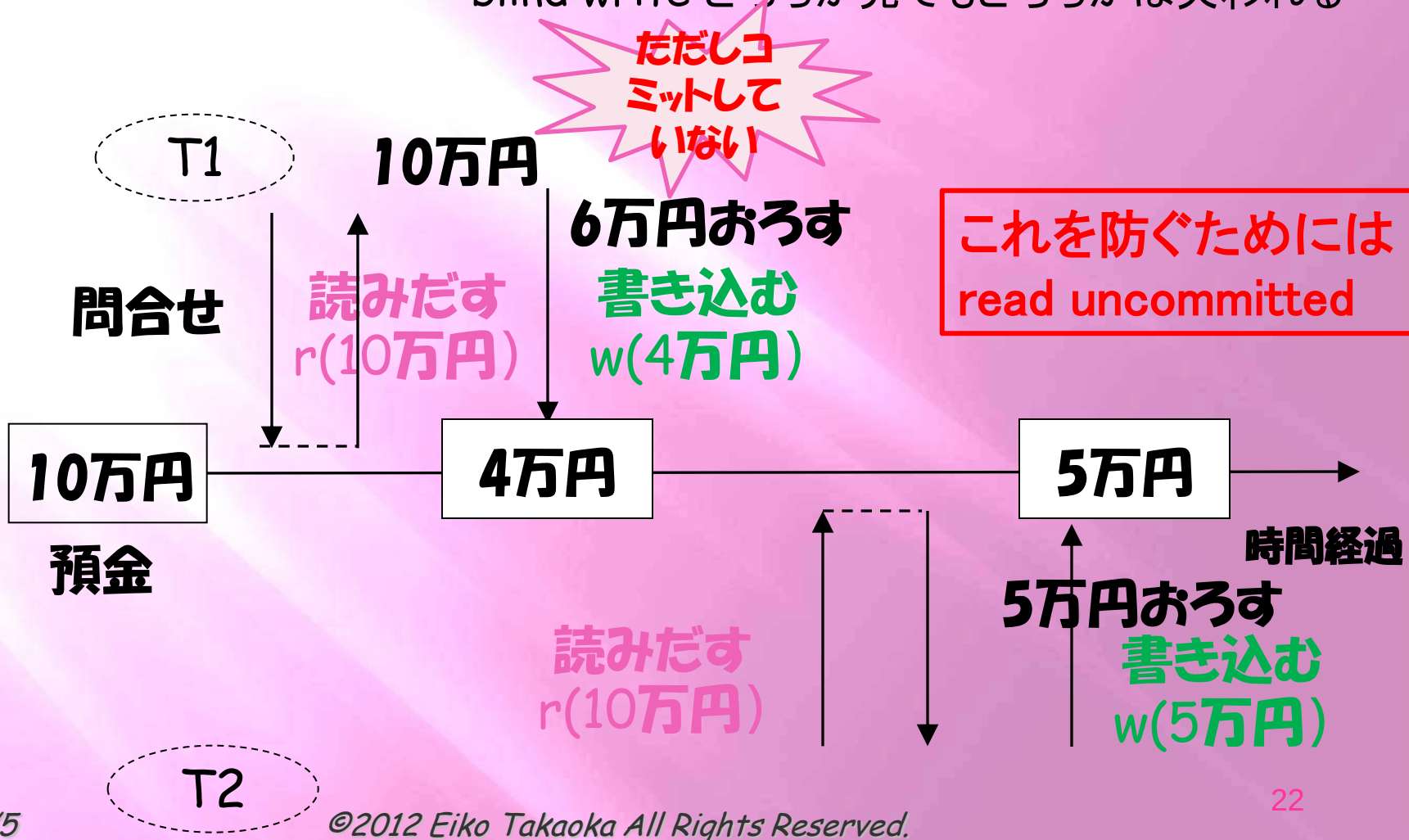
同時実行に伴って発生する異常と回避方法

©2012 Eiko Takaoka All Rights Reserved.

同時実行に伴って生じる異常①

T1とT2が同じデータを読み出してそれぞれ別の値に更新する

このような異常を → lost update(先の更新値が失われる)
blind write どちらが先でもどちらかは失われる



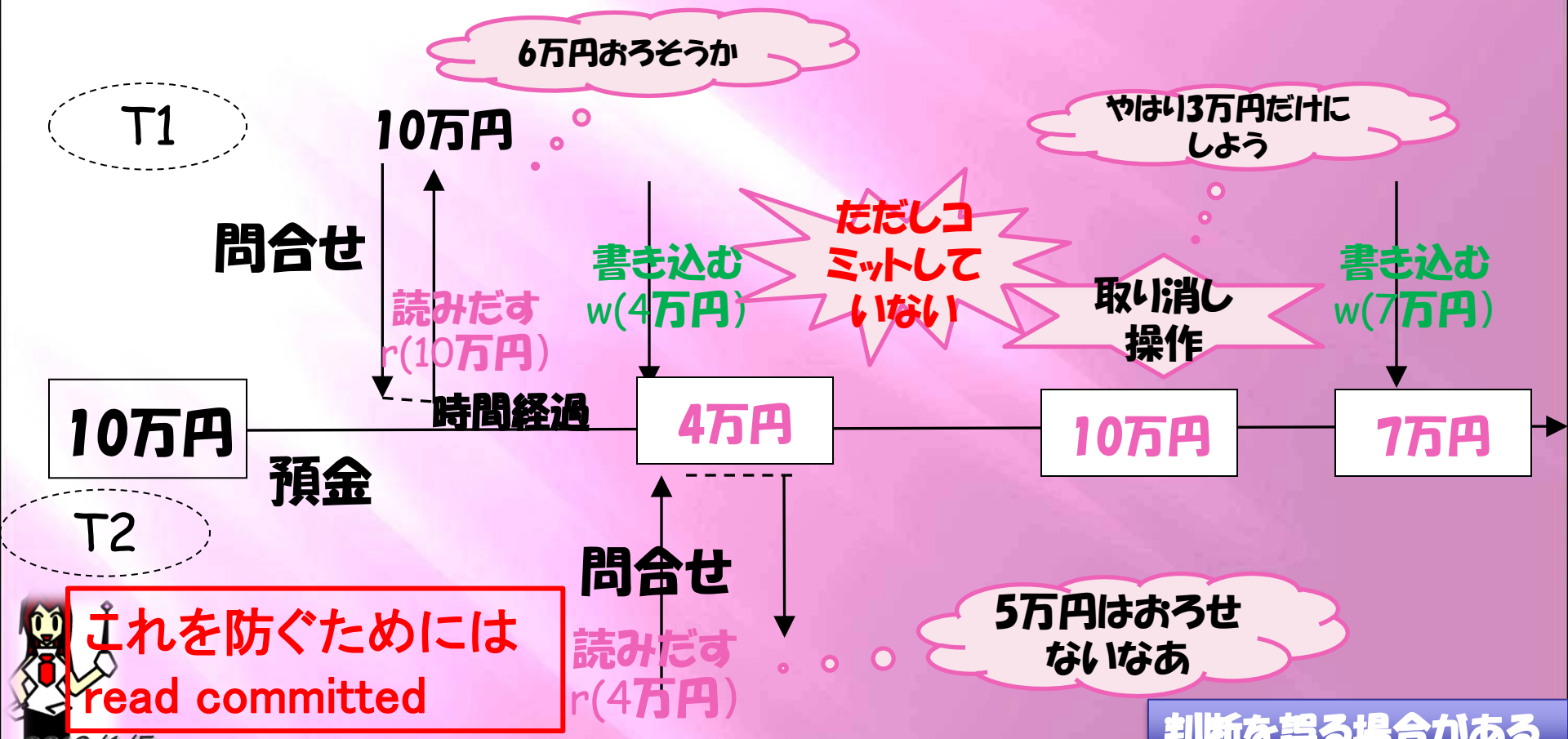
これを防ぐためには
read uncommitted

©2012 Eiko Takaoka All Rights Reserved.

同時実行に伴って生じる異常②

T1が更新中のデータ(最終的には別の値になる可能性がある)をT2が参照してしまうこと
ACID特性のうち一貫性と矛盾する

このような異常を → Dirty read



これを防ぐためには
read committed

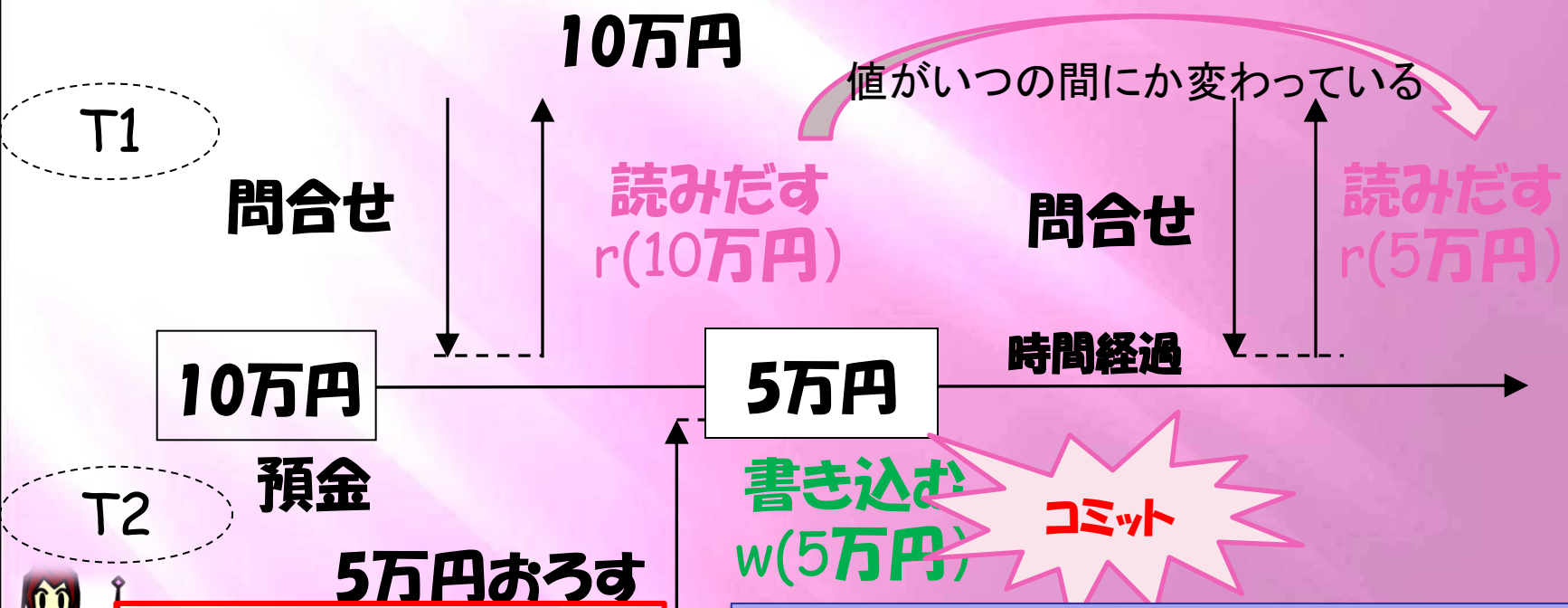
判断を誤る場合がある

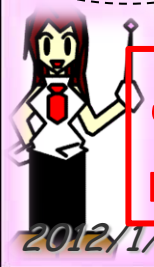
©2012 Eiko Takaoka All Rights Reserved.

同時実行に伴って生じる異常③

T1が読んだデータの値をT2が修正か削除したあと,
T1がもう一度その値を読むと値がいつの間にか変わったり消えたりする

このような異常を → non-repeatable read
（反復不可能読み取り）





これを防ぐためには
repeatable read

同じ問い合わせを二回連続して繰り返し(リピート)ても同じ結果になると「保証できません」という意味

©2012 Eiko Takaoka All Rights Reserved.

同時実行に伴って生じる異常④

T1がデータ検索後，T2の更新によって，T1が再度同じ条件で検索した場合
前にはなかった値まで結果として得られる。

このような異常を → phantom read

T1

日付	操作	残高
1/5	預入	30000
1/7	預入	100000

日付	操作	残高
1/5	預入	30000
1/7	預入	100000
1/7	引出	50000

履歴問合せ

読みだす

再度
問合せ

読みだす

前はなかった行が…

10万円

5万円

預金

時間経過

書き込み

コミット

5万円おろす w(5万円)

T2

これを防ぐためにはserialize

©2012 Eiko Takaoka All Rights Reserved.

non-repeatable readとphantom readの違い

- non-repeatable read
  - 既存のデータが「更新（削除）」されること
- phantom read
  - 前回は存在していなかったものが同じ問い合わせで「出現」すること
- 存在したものの過去の像を見せることと、現在は存在し過去には存在しなかったものを今も存在しないように見せるとの違いはシステム上では大きな差がある。

©2012 Eiko Takaoka All Rights Reserved.

3つの回避可能な現象

1. dirty read
   - まだコミットされていないトランザクションが書き込んだデータを別のトランザクションが読んでしまう現象
2. non-repeatable read
   - 同じ問い合わせを二回連続して繰り返し(リピート)ても同じ結果になると「保証しない」読み取り
   - コミットの前後で別トランザクションが異なるデータを読み取ってしまう現象
3. phantom read(仮読み取り)
   - T1 から問い合わせのあとT2 から新たな登録を行なった場合に T2 のコミットの前後で T1 の問い合わせに以前は存在しなかったものが出現すること。

©2012 Eiko Takaoka All Rights Reserved

分離レベル(isolation level)

- 3つの回避可能な現象という観点から，4つの分離レベルが定義
- トランザクションが別のトランザクションとどの程度に分離されるかの程度を表す
- それぞれトランザクション処理の効率に与える影響の度合いが異なる

©2012 Eiko Takaoka All Rights Reserved.

分離レベルと異常の関係

異常 / 分離レベル	lost update	dirty read 内容を保証しない読み取り	non-repeatable read	phantom read 仮読み取り
read uncommitted	×	○	○	○
read committed	×	×	○	○
repeatable read	×	×	×	○
serializable	×	×	×	×

○発生しえる
×発生しない

©2012 Eiko Takaoka All Rights Reserved.

Serializable

- 厳密な二相ロック
- DataSet の影響を受ける行に、EOT(トランザクションの終了）までロックがかかる
- 変更されたアクセス構造、および問い合わせで使われた構造がすべて EOT までロックされる。

©2012 Eiko Takaoka All Rights Reserved.

Repeatable Read

- データベースの読み取りと変更時にロック
- 変更されたすべてのオブジェクトのロックはEOTまで維持される.
- データの読み取りロックは、EOTまで維持
- 同時実行しているトランザクションがデータに対して行った更新は実行中に読めない
- 新たに挿入されたデータにはロックがかからないので別トランザクションが見ることができる ⇒ phantom readはあり得る

©2012 Eiko Takaoka All Rights Reserved.

Read Committed

- データベースの読み取りと変更時にロック
- 読み取り後にロックは解除
- 変更されたオブジェクトのロックは EOT まで維持
- データに対する並列な更新がコミットされればそれを読むことが可能
- 同じデータを二度読んだり問い合わせをすると，異なる結果が返ってくる

⇒ non-repeatable readとphantom readはあり得る

©2012 Eiko Takaoka All Rights Reserved.

Read Uncommitted

- dirty read を許す
- 読み込みにたいしてロックをかけない
- 書き出しは許されないRead Onlyアクセスモードのトランザクション
- lost updateは免れる
- 一番ゆるい分離レベル

©2012 Eiko Takaoka All Rights Reserved.

分離レベルと性能の関係

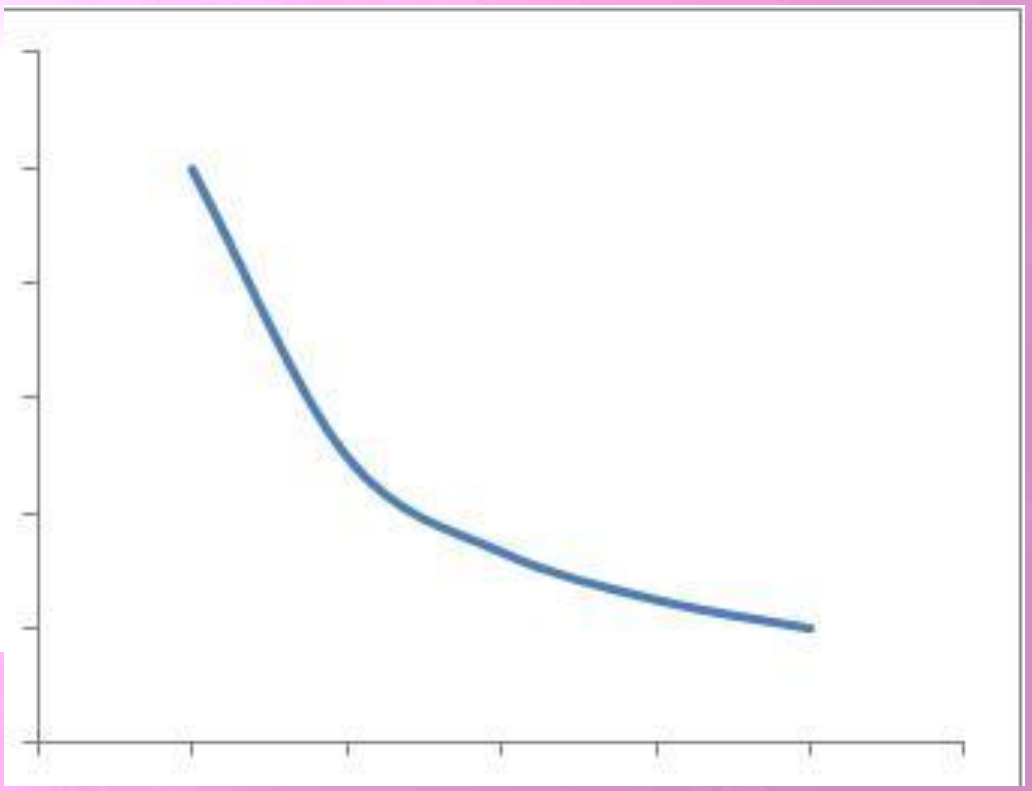
serializable

repeatable read

データ読み出しの統計的な問い合わせならこのあたりでOK

read committed

read uncommitted



低い ←―― 性能 ――→ 高い

©2012 Eiko Takaoka All Rights Reserved.

スケジュールと直列可能性

- 異常回避のために単純に順番に実行(直列実行)するか？

- 結果は正しくても並列性は向上しない

- 直列可能性を持つ(直列実行した場合の結果のどれかになる)ように複数のトランザクションを並列実行させる

©2012 Eiko Takaoka All Rights Reserved.

同時実行制御方式

- 直列可能なスケジュールを得るための手段
  - ロックに基づく方式
  - タイムスタンプに基づく方式
  - 多版制御方式
  - 楽観的制御方式

©2012 Eiko Takaoka All Rights Reserved.

ロック

- ロックとは
  - データを予約しておく(⇒lockをかける)
  - ただ一つのトランザクションだけが更新を行うことを許すための制御のしくみ
  - 他のトランザクションはロックが解放(unlock)されるまでそのデータの更新はできない

©2012 Eiko Takaoka All Rights Reserved.

ロック方式

- 占有ロック(exclusive/write)
  - トランザクションが INSERT 文、UPDATE 文及び DELETE 文であるときのロックのモード
  - 他のトランザクションに参照も更新もさせない
  - データベース更新時に要求
- 共有ロック(shared/read)
  - トランザクションが SELECT 文によりデータを参照するものであるときのロックのモード
  - 他のトランザクションに参照のみを許す．更新は許さない
  - データベース参照時に要求する
- 共有ロック同士は両立
- 占有ロックと他のロック(共有でも占有でも)とは互いに排他的

©2012 Eiko Takaoka All Rights Reserved.

ロックの粒度：どの範囲を単位としてロックをかけるか

- データベース
  - データベース全体に対してのロック
  - システムバージョンアップなどのメンテナンスの場合
- テーブルスペース
- テーブル全体
  - テーブルから既存の項目を削除，新しい項目の追加，すべての行を更新するなど
- ブロック，ページ
- 行：典型的なロックレベル

©2012 Eiko Takaoka All Rights Reserved.

ロックの粒度と実行効率のトレードオフ

- 同時実行性を高めるためには？
  - ロックの粒度は小さいほうがよい
  - トランザクションの衝突する可能性が大きい
  - CPUの負担も増加（ロックをかけたり解除したりする処理が複雑になる）
- 粒度が大きいと？
  - トランザクションが衝突する可能性が減る
  - 同時実行性は低くなってしまう

©2012 Eiko Takaoka All Rights Reserved.