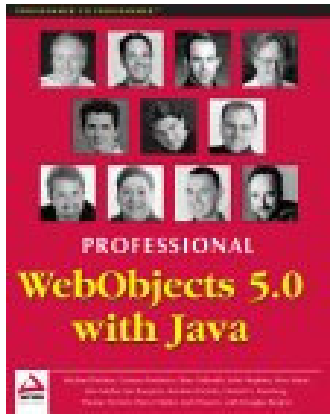
Advanced EOF – Locking

from Professional Webobjects 5.0 With Java



WR

WR@Csus4.net

http://www.csus4.net/WR/

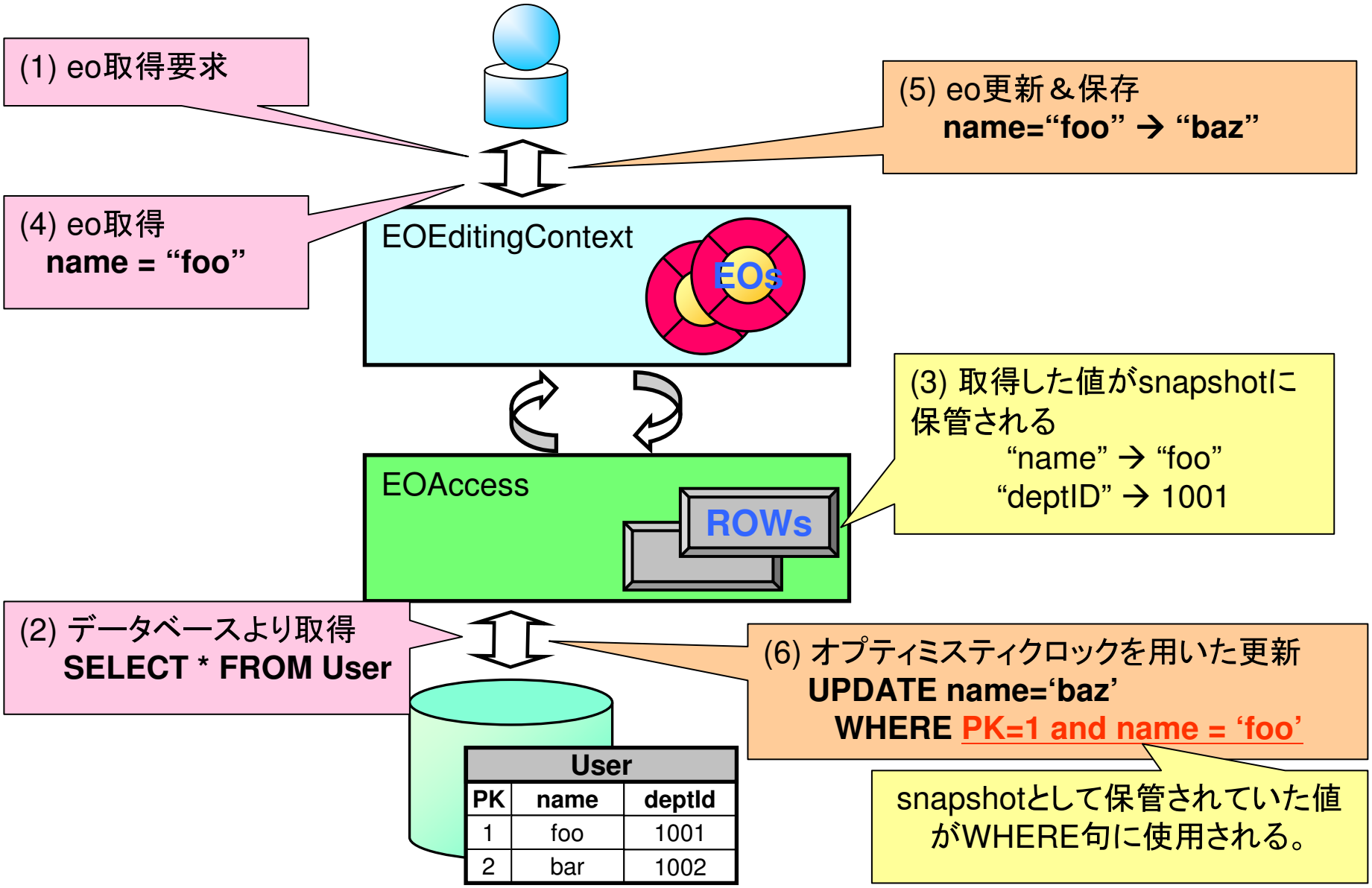
WebObjects/EOFにおけるLockの分類

- Optimistic Locking (楽観的ロック)
- Pessimistic Locking (悲観的ロック)
  - Lock on Select
  - Lock on Update
  - Lock on Demand
- Locking a Column
- Locking By Application Logic

楽観的ロック (Optimistic Locking) [1]

- Paragraph1
  - デフォルトでは楽観的ロックが使われる
  - 楽観的ロックはactivelyにはDB上のデータをロックしない
- Paragraph2
  - 鍵アイコンが付いたattributeのみを使用する
- Paragraph3
  - フェッチしたデータをsnapshotに保管する
  - 更新時にsnapshot内のデータとDB上のデータを比較する
    - 同じ場合、更新は成功
    - 違う場合、更新は失敗
      - 例外が発行されるので、キチンと拾いましょう

楽観的ロック [2] - snapshotとSQLクエリ



楽観的ロック [3]

- Paragraph4
  - 楽観的ロックを使うのは簡単
  - ただし、保存しようとするまで、DB上のデータの変更に気付かない
  - UPDATEステートメントにWHERE句を含むため、更新処理が若干遅い
- Paragraph5
  - 更新の衝突が起こりにくい場合《に有効》
  - 《他者がデータにアクセスしているかどうかではなく》実際にDB上のデータが更新できているかどうかに関心がある
- Paragraph6
  - どのカラムに鍵マークをつけるべきかは、楽観的ロックの仕組みから判断できる

楽観的ロック [4]

- Paragraph7
  - ロックする属性を増やせば増やすほど、限定子(=WHERE句)に含まれる値の数が増え、DBの更新処理が遅くなる
  - 変更されないとわかっている属性はロック対象から外す
- Paragraph8
  - BLOB (Binary Large OBject)はロック対象から外せ
- Paragraph9
  - データ更新時に必ず更新される属性があれば、この属性のみをロックすることで、楽観的ロックを実現できる
    - 更新カウンタ (update counter) とか
  - 利点：処理が軽い

楽観的ロック [5]

- Paragraph10
  - PKとロック対象属性を含む複合インデックスを作るとDB処理をチューニングできる
    - *いわゆるカバレージインデックス*
  - Our Problem …
    - *わかりません!!!!*
- Paragraph11
  - 別のアイディアがある
    - ロック属性の修正をDB側に持たせる (ex. トリガ)
      - *わからない!!!*
    - 利点: 値の変更のみでよい(???)
    - 欠点: JavaとDBストアドの両方のメンテが面倒

悲観的ロック [1]

- Paragraph1
  - 悲観的ロックはロックしたデータに対する他者のアクセスをactivelyに防ぐ方式
- Paragraph2
  - デフォルトのstrategyを悲観的ロックとすることで利用可能
    - *EODatabaseContext.setUpdateStrategy()のこと??*
  - 楽観的ロックとの併用は不可
- Paragraph3
  - 一旦オブジェクトをロックしたら、保存またはundoするまでロックは解除されない

悲観的ロック [2]

- Paragraph4
  - 注意点いくつか
    - 保存またはrefault(=undo)するのを忘れると、ロックされたままになる
    - ロック失敗時の例外処理を入れ込むこと
- Paragraph5
  - 悲観的ロックはWebObjectsアプリケーションにしばしば適合しない
    - often unsuitable
  - セッションの制御は不可能
    - 偶然、ブラウザを終了されたら???

悲観的ロック [3]

- Paragraph6
  - 悲観的ロックはパワフルであるが、DBMSごとにロック機構が違うため、取り扱いが難しい場合がある
    - ロックの粒度、ロック衝突時の振る舞いなど
  - 悲観的ロックを使うときは、使用するDBMSのマニュアルをちゃんと読め！ RTFM!!
- Paragraph7
  - EOFは3つの悲観的ロック戦略をサポートする

Lock on Select [1]

- Paragraph1
  - グローバルに機能する
  - フェッチしたデータは自動的にロックされる
  - デフォルトのロック戦略を変更する
- Paragraph2
  - savaChanges()またはrevert(), invalidateAll()したときにロックは自動的に解除される
  - 再フェッチすると、再ロックされる

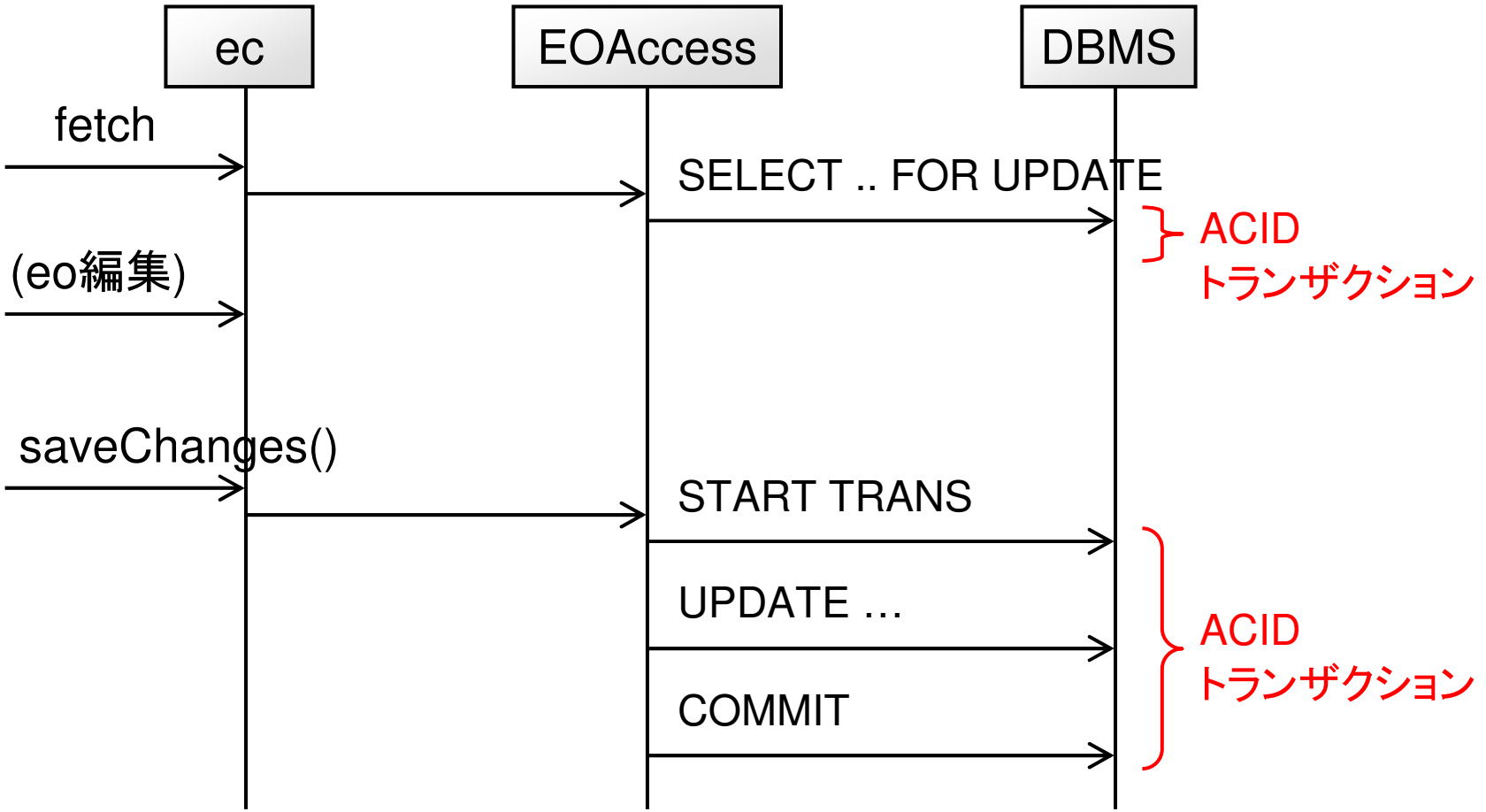
Lock on Select [2]

- Paragraph3

```
EOFetchSpecification fs;
EOObjectsStoreCoodinator osc =
    EOObjectStoreCoodinator.defaultCoodinator();
EODatabaseContext dbc =
    osc.objectStoreForFetchSpecification(fs);
dbc.setUpdateStrategy(
    EODatabaseContext.UpdateWithPessimisticLocking
    )
```

- Paragraph4
  - 複数DBコネクションを使用している場合は、コネクション毎に上記処理を行うこと

Lock on Select [3] –シーケンスとSQLクエリ



SELECT … FOR UPDATE ～ UPDATEが同一のACID
トランザクション内にカプセル化されるべきでは???
※ (WOMeeting:792)から始まるスレッドを見よ

Lock on Select [4]

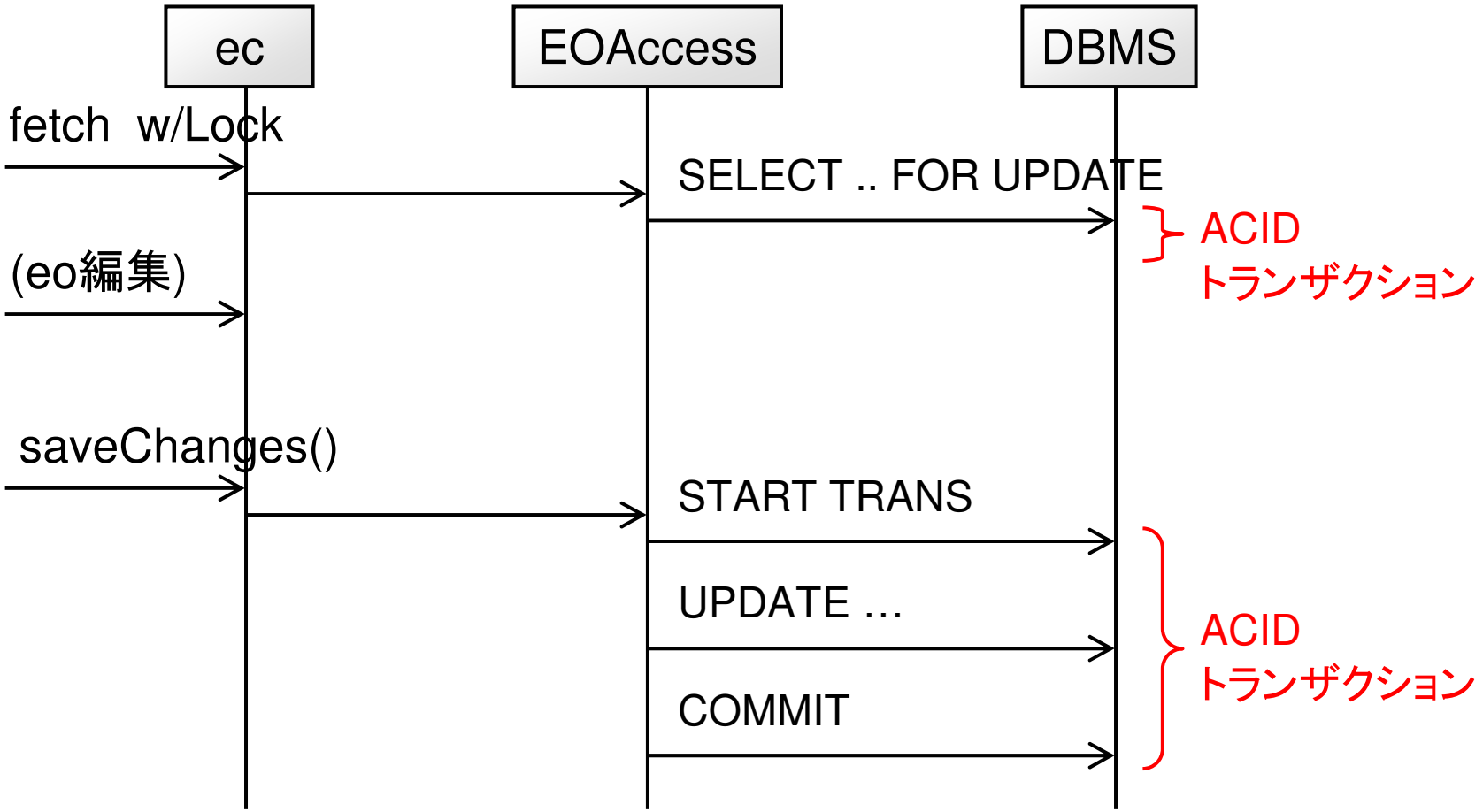
- Paragraph5
  - It works directly when …
    - **よくわからない**
  - デッドロック発生時は例外があがる
  - 書き込み処理より読み出し処理が多い場合は、ロックを必要以上にかけることとなる
- Paragraph6, 7
  - 特定のフェッチにおいて、特定のオブジェクトのみをロックしたい場合、フェッチスペシフィケーションにその旨指定可能

```
fs.setLocksObjects(true);
fs.setRefreshedRefetchedObjects(true);
```

Lock on Select [5]

- Paragraph8
  - internal cache (=snapshot)をクリアするためにsetRefreshesRefetchedObjects()を呼べ
  - 定義済みフェッチスペシフィケーションがある場合は、Optionを設定せよ
- Paragraph9
  - この戦略はEOModelerを用いてフェッチスペシフィケーションを定義する場合は、最も便利なやり方である
  - しかし、ロックの解除を忘れないようにせよ

Lock on Select [6] - シーケンスとSQLクエリ



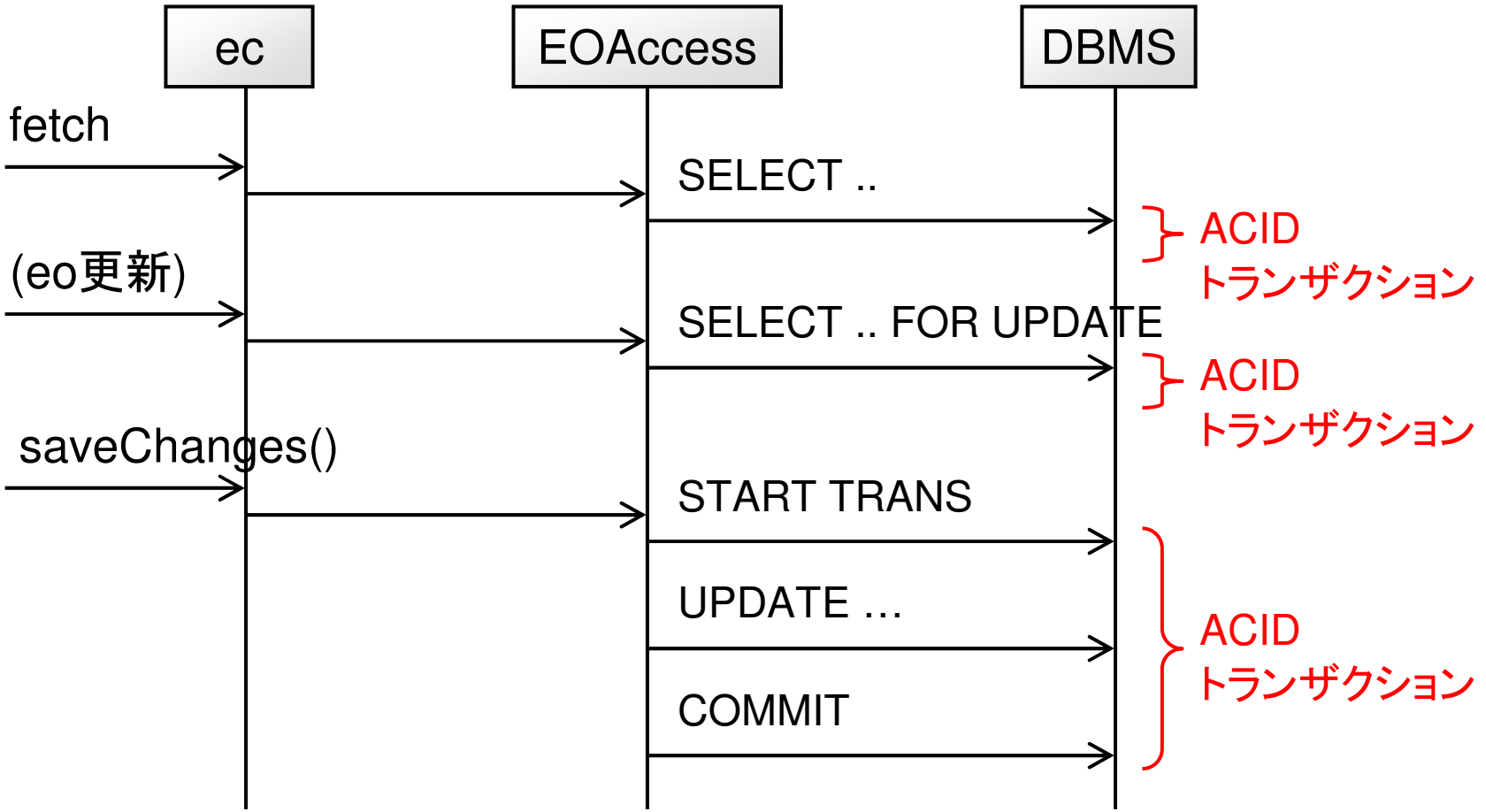
Lock on Update [1]

- Paragraph1
  - ロックするオブジェクトの数を減らしたい場合はLock on Updateが有効である
  - フェッチしたオブジェクトに修正が加えられたタイミングでロックをかける
  - ロックをかける前に再フェッチする
- Paragraph2

```
ec.setLocksObjectsBeforeFirstModification(true)
```

- Paragraph3
  - Lock on Selectよりオーバーヘッドが少ない
    - 具体的には??
  - デッドロック発生時には例外が投げられる
  - 修正前にDBにクエリが飛ぶ

Lock on Update [2] - シーケンスとSQLクエリ



SELECT … FOR UPDATE ～ UPDATEが同一のACID
トランザクション内にカプセル化されるべきでは???
※ (WOMeeting:792)から始まるスレッドを見よ

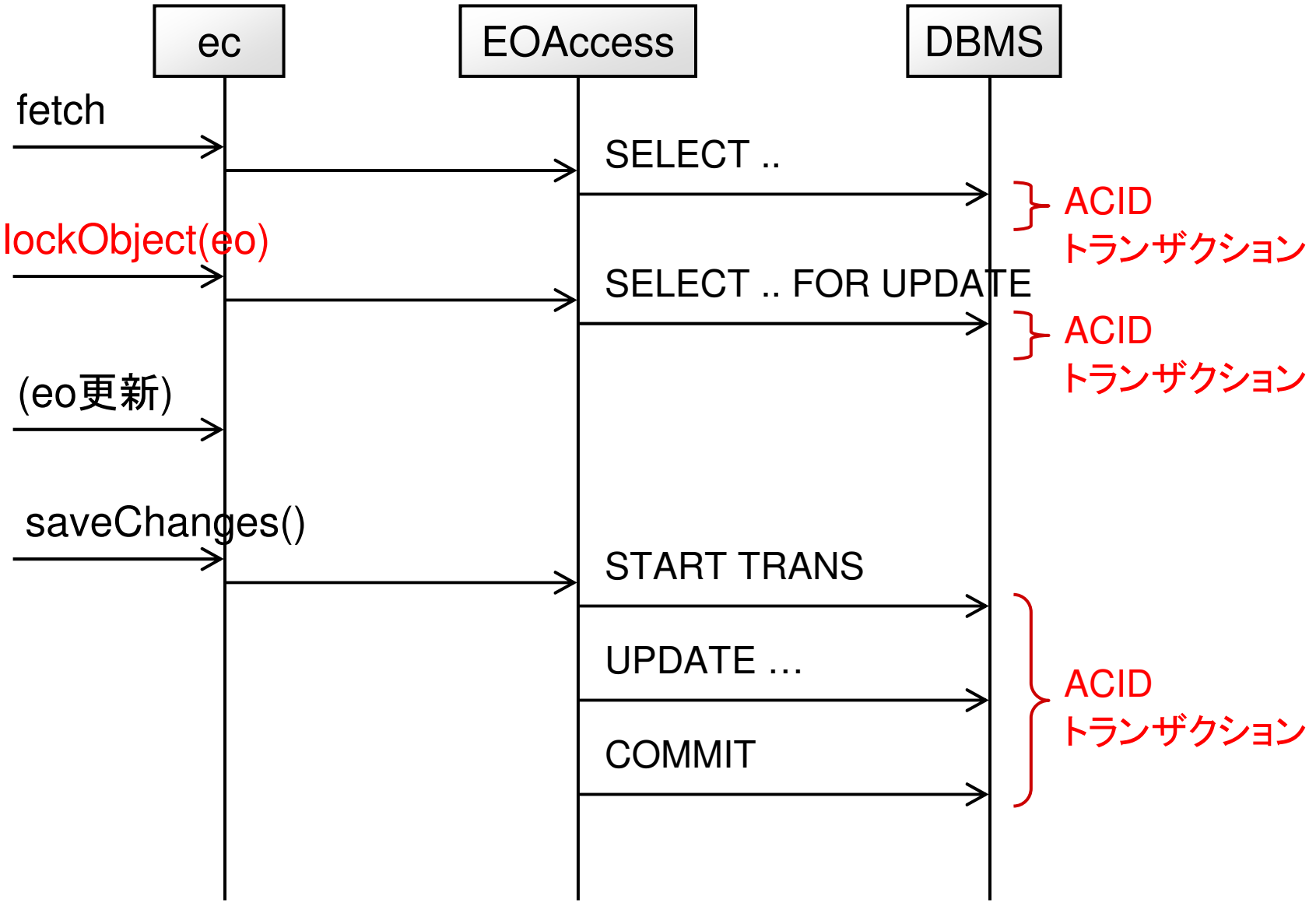
Lock on Demand [1]

- Paragraph1
  - ロックする対象(=eo)、タイミングを明示的に指定できる
- Paragraph2

```
ec.lockObject(eo)
```

- Paragraph3
  - 欠点：余計なフェッチが走る

Lock on Demand [2] - シーケンスとSQLクエリ



Locking a Column [1]

- Paragraph1
  - 楽観的/悲観的ロックの1つの欠点
    → save/revert時にロックが解除される
  - 複数のsave操作にまたがってロックを維持したい場合や、長時間ロックを維持したい場合はどうする?
- Paragraph2
  - ロック用のカラムを付け加えるのが良い
  - ただし、上記カラムの処理をアプリケーション側に実装する必要がある
- Paragraph3
  - 余計に保存領域が必要
  - Boolean型のカラムが最も単純
    - ロック:true / アンロック:false

User		
PK	**name**	**locked?**
1	foo	true
2	bar	false

Locking a Column [2]

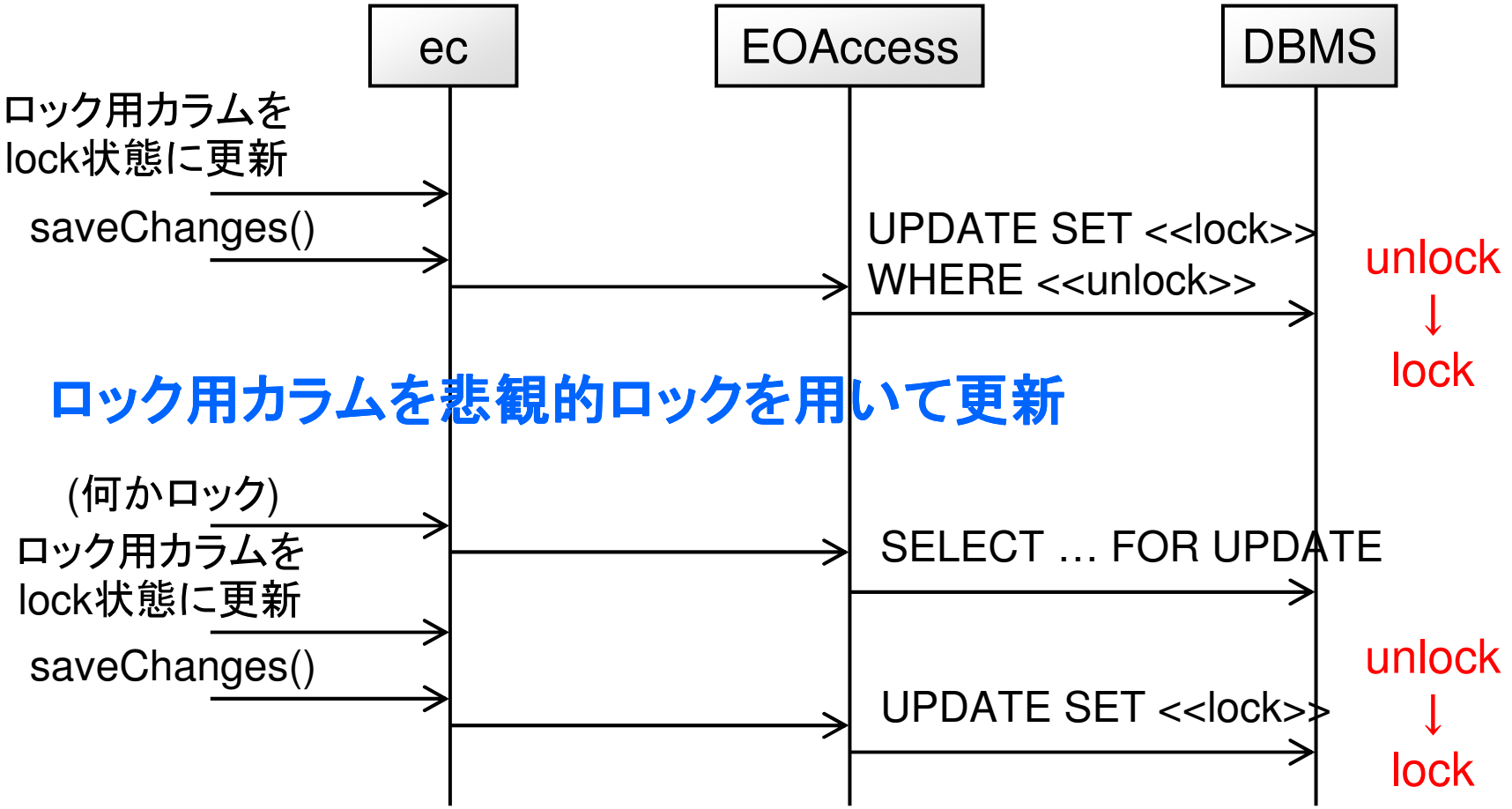
- Paragraph4
  - 「誰がカラムをロックしたか?」を追跡できるよう、sessionIDやuserIDを保存するのも良い
    - 異なるアプリケーションインスタンス間でアクセスを共有できる
    - この場合、「null = アンロック状態」となる
- Paragraph5
  - ロック用カラムにどんな情報を保存するかは要件による
  - 複合的な情報でも良い
    - タイムスタンプとユーザー情報とか

Locking a Column [3]

- Paragraph6
  - 先のアプローチは便利で強力に思えるが、《ロック以外の》機能を入れ込むと、実装がトリッキーになる可能性がある
    - ロック以外の制御情報を入れ込まないほうがいいよ！と言っているようだ。
- Paragraph7
  - ロック《用カラム》を変更する際の同時実行制御のため、楽観的 / 悲観的ロックを使用する必要がある
  - 《ロック用カラムを変更するために》楽観的 / 悲観的ロックを使用することに賛否両論ある(???)
  - 楽観的ロックを使用する場合は、ロック用カラムのみに鍵マークをつける

Locking a Column [4] – シーケンス

ロック用カラムを楽観的ロックを用いて更新



ロック用カラムを悲観的ロックを用いて更新

通常の悲観的ロックと同様に、ロックのかけ方にはいろいろな方法がある点に注意されたい。(Lock on Select, etc …)

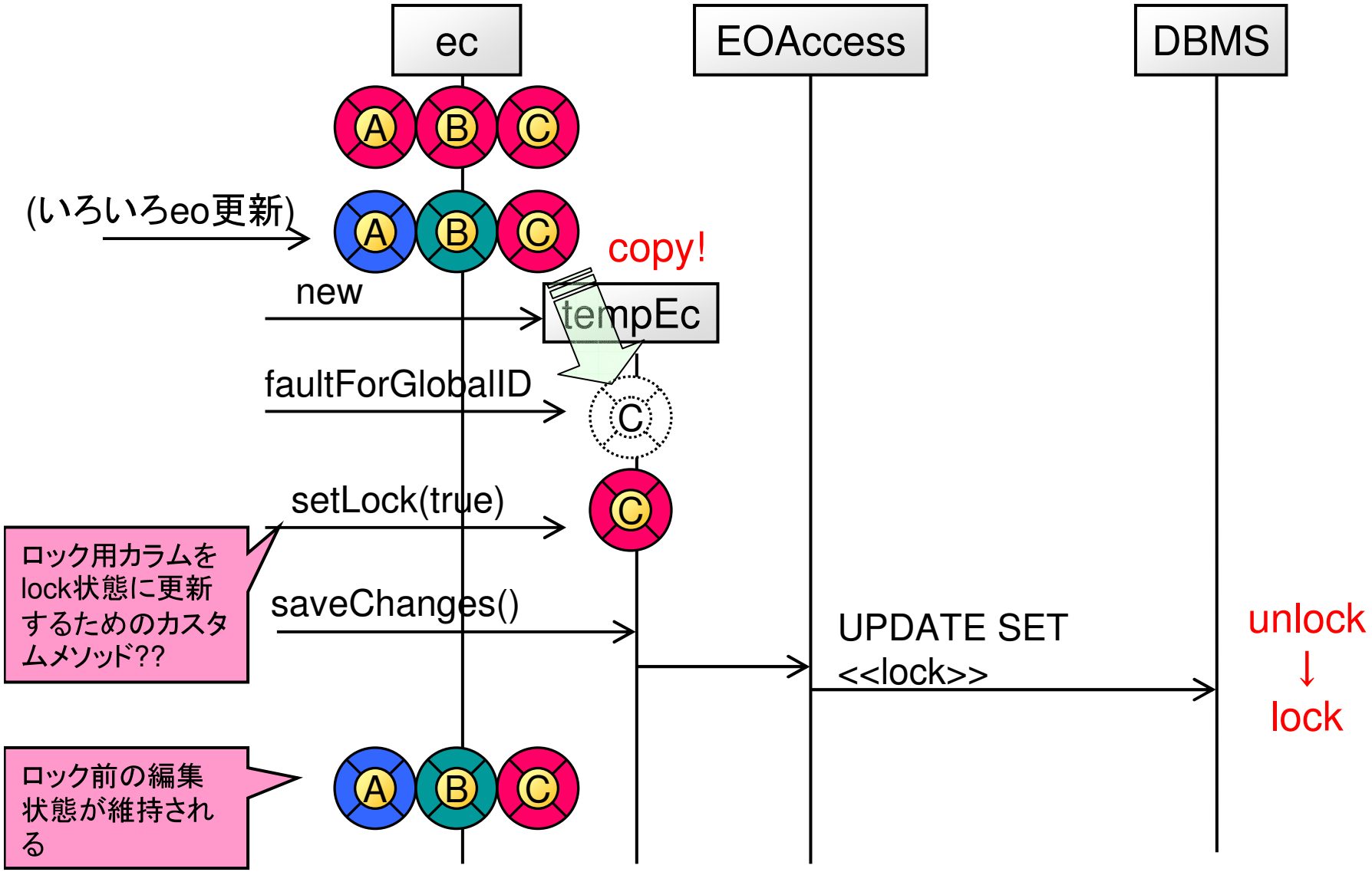
Locking a Column [5]

- Paragraph8
  - 欠点
    - ロック用カラムの変更だけがコミットされるのではなく、通常のデータ用カラムの修正も一緒にコミットされてしまう！
  - 新しいecを使うことで回避可能
    - 新しいecには、ロック用カラムの変更を行うeoのみが含まれる

Locking a Column [6]

- Paragraph9
  - ロック用カラムを変更するコードを以下に示す
    - 配布の資料を見よ。
    - 次ページのシーケンスも。
  - ロックされたオブジェクトへのアクセスをどのように処理するか？は要件による。
    - あらゆる修正が行われるたびに、ロックの有無をチェックする
    - 保存処理のときのみに、ロックの有無をチェックする

Locking a Column [7] – シーケンス



Locking a Column [8]

- Paragraph10
  - 新しいecにeoを入れ込むのにglobalIDとfaultを使用
  - ec.InsertObject()を使わない理由は、awakeFromInsertion()が実行されることを避けるため
  - ec.setLock()とかをオーバライドするのもいいかも
    - よくわからない
- その他
  - WO4.5のドキュメントにLocking on Columnの記載あり
    - Documentation > WebObjects 4.5 > Programming Tips > Locking on a Column
    - http://developer.apple.com/documentation/LegacyTechnologies/WebObjects/WebObjects_4.5/System/Documentation/Developer/WebObjects/Topics/ProgrammingTopics.2b.html

Locking by Application Logic

- なんかいろいろ書いてありますけど、正直よくわかりません･･･
- つーか、

何いい加減なこと抜かしとんじゃ！

という気もします。

まとめ? [1]

- 楽観的ロック
  - 更新時UPDATE SET <<newValue>> WHERE <<oldValue>>
- 悲観的ロック
  - Lock on Select (1)
    - dbc.setUpdateStrategy(….UpdateWithPessimisticLocking)
    - フェッチ時：SELECT … FOR UPDATE
    - グローバルに有効
  - Lock on Select (2)
    - fs.setLocksObjects(true)
    - フェッチ時：SELECT … FOR UPDATE
    - あるfsのみ有効

まとめ? [2]

- 悲観的ロック (続き)
  - Lock on Update
    - ec.setLocksObjectsBeforeFirstModification(true)
    - ec配下のeoを更新したタイミングで、更新したeoをロック
    - SELECT … FOR UPDATE WHERE <<eo>>
  - Lock on Demand
    - ec.lockObject(eo)
    - lockObjectを実行したタイミングで、指定eoをロック
    - SELECT … FOR UPDATE WHERE <<eo>>

まとめ？[3]

- Locking a Column
  - 別途、ロック用カラムを設ける
  - ロック用カラムの更新には若干工夫が必要な場合がある
- Locking By Application Logic
  - よくわかりません
  - そもそも、アプリケーションで頑張らないために、DB/ACIDトランザクションを使うんでは？とか思ったりする

補遺

- 同一osc内のec間でロックが機能しない問題
  - 楽観的ロック
    - 同一osc配下のec間でsnapshotを共有していることに起因
  - 悲観的ロック
    - 同一osc配下のec間でDBコネクションを共有していることに起因
- つーか、悲観的ロックをまともに使えるように・・・
  - Lock on XXXの利用戦略
  - ACIDトランザクションの利用戦略
    - RDBMS毎のACIDトランザクションの振る舞いの違い
  - 悲観的ロックをサポートしているアダプタはある？
  - やっぱり、悲観的ロックは使わないのが吉？

- 求む！WebObjects実務経験者の意見！