Practical Webobjects
Chapter 3 (Page 61-97):

Managing the Object Graph

WR

WR at Csus4.net

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

目次

- Understanding the Object Graph
  - **オブジェクトグラフと**editing context**の基本**
- Rules of Engagement
  - **ルールは守りましょう**
- Seeing State Changes in the Object Graph
  - **オブジェクトグラフの状態変化を監視する方法**
- Debugging Object Graph Problems
  - **オブジェクトグラフにまつわるトラブルシュート**
- Locking, Concurrency, and EOF
  - **ロックと並行性の問題**
- Managing Object Freshness
  - **キャッシュ問題**

目次

- Understanding the Object Graph
  - オブジェクトグラフとediting contextの基本
- Rules of Engagement
  - ルールは守りましょう
- Seeing State Changes in the Object Graph
  - オブジェクトグラフの状態変化を監視する方法
- Debugging Object Graph Problems
  - オブジェクトグラフにまつわるトラブルシュート
- Locking, Concurrency, and EOF
  - ロックと並行性の問題
- Managing Object Freshness
  - キャッシュ問題

Understanding the Object Graph

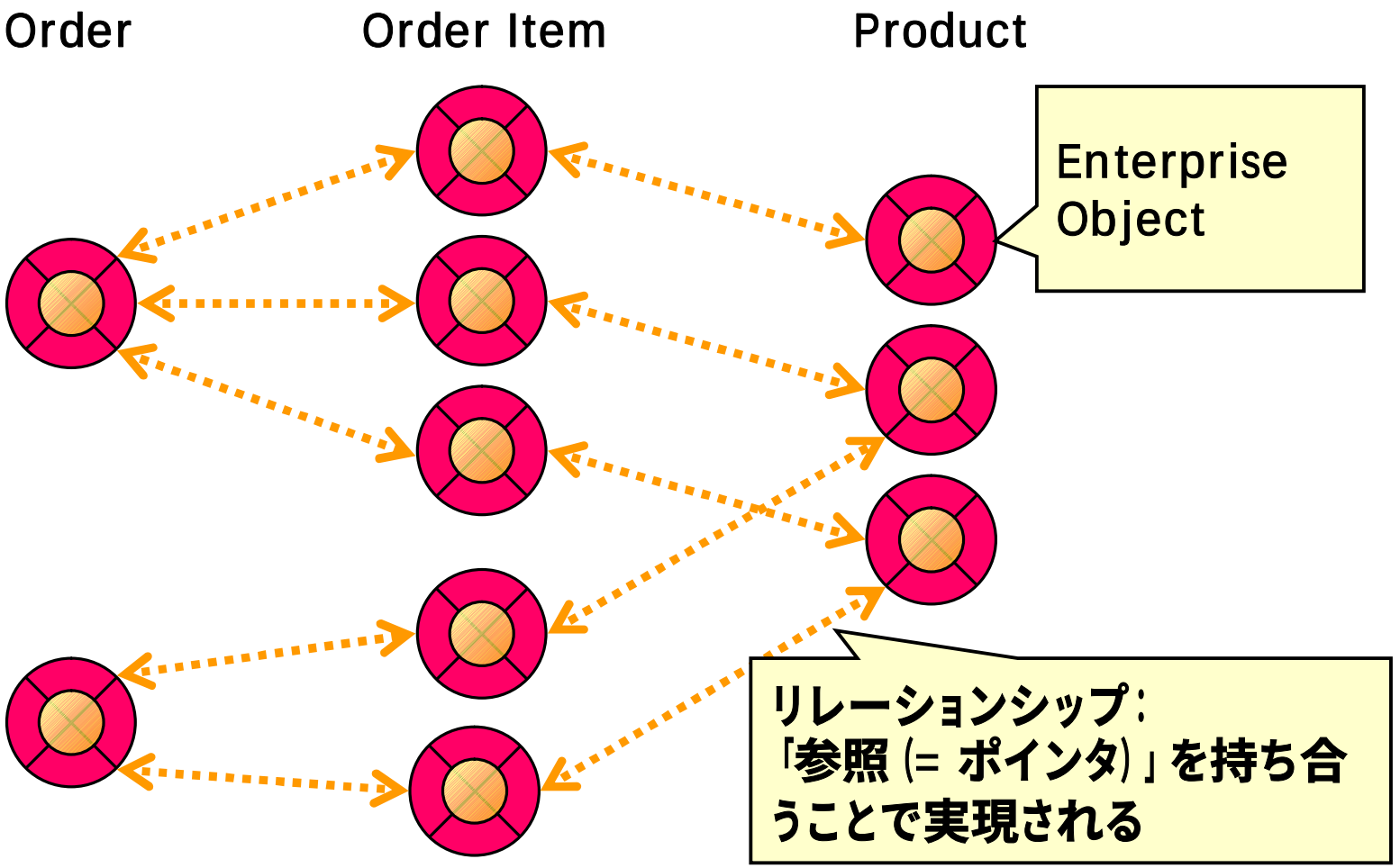
P61-65.

- Viewing the Object Graph in Context
- Understanding the Varieties of Editing Context
  - The Ordinary Editing Context
    - The Peer Editing Context
    - The Default Editing Context
    - Child Editing Contexts
    - Nested Editing Contexts
  - The Shared Editing Context

オブジェクトグラフとは？

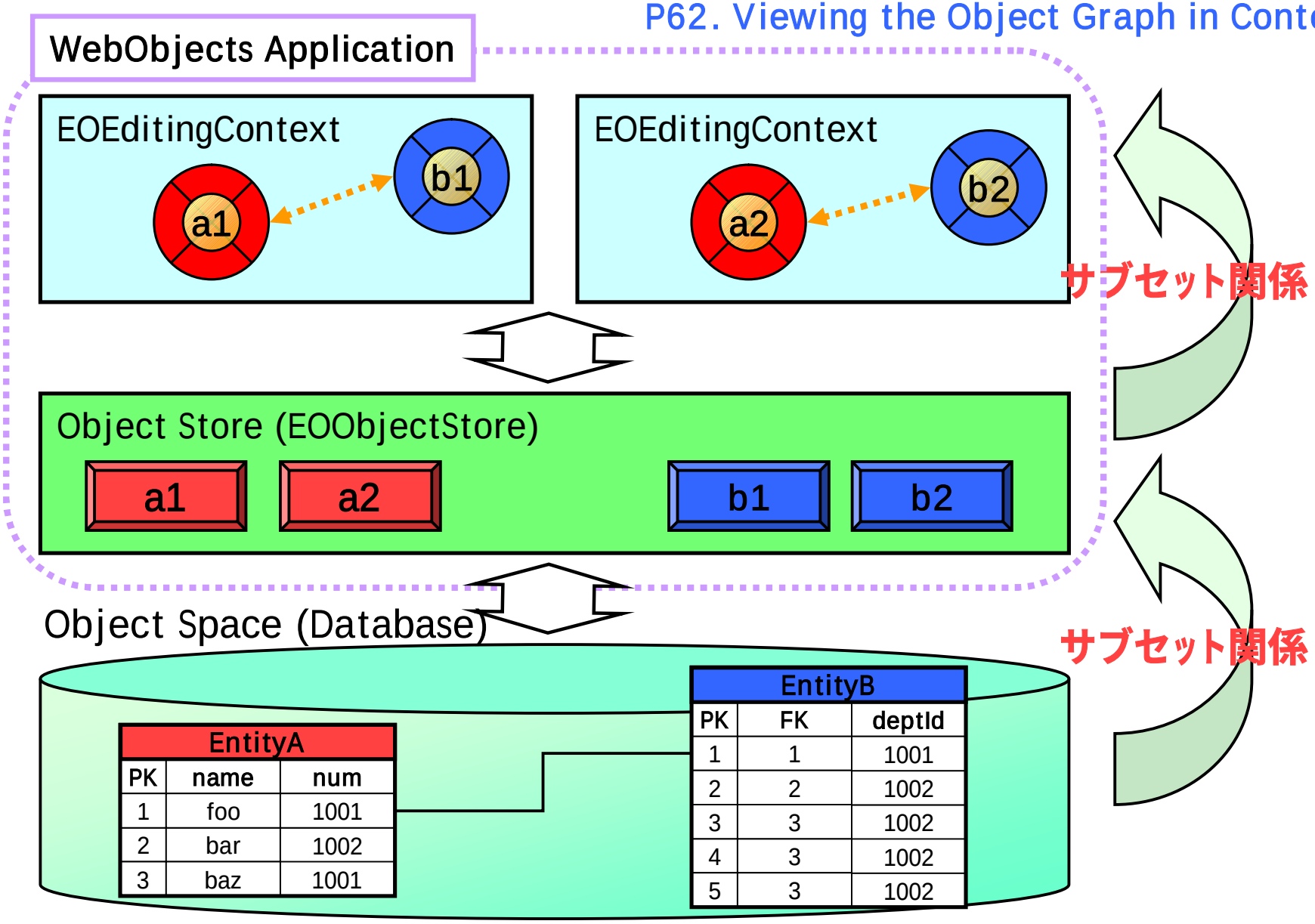
P61. Understanding the Object Graph

- オブジェクトグラフ = オブジェクトのネットワーク



各レイヤが保持するデータ

P62. Viewing the Object Graph in Context



EOEditingContextの役割

P62. Viewing the Object Graph in Context

- Object Storeの保持するデータのサブセットをenterprise objectの形式で保持する
  - WR: 前のページのことね。
  - WR: ついでに言えば、EOのオブジェクトグラフとして
- データ編集のための隔離された、パーソナルな作業領域(sandbox)として振舞う
- Object Storeの内部的に一貫したビューを維持するため、enterprise objectを管理する

- ふーむ、「内部的に一貫した」とは？？？

「内部的に一貫した」とは？

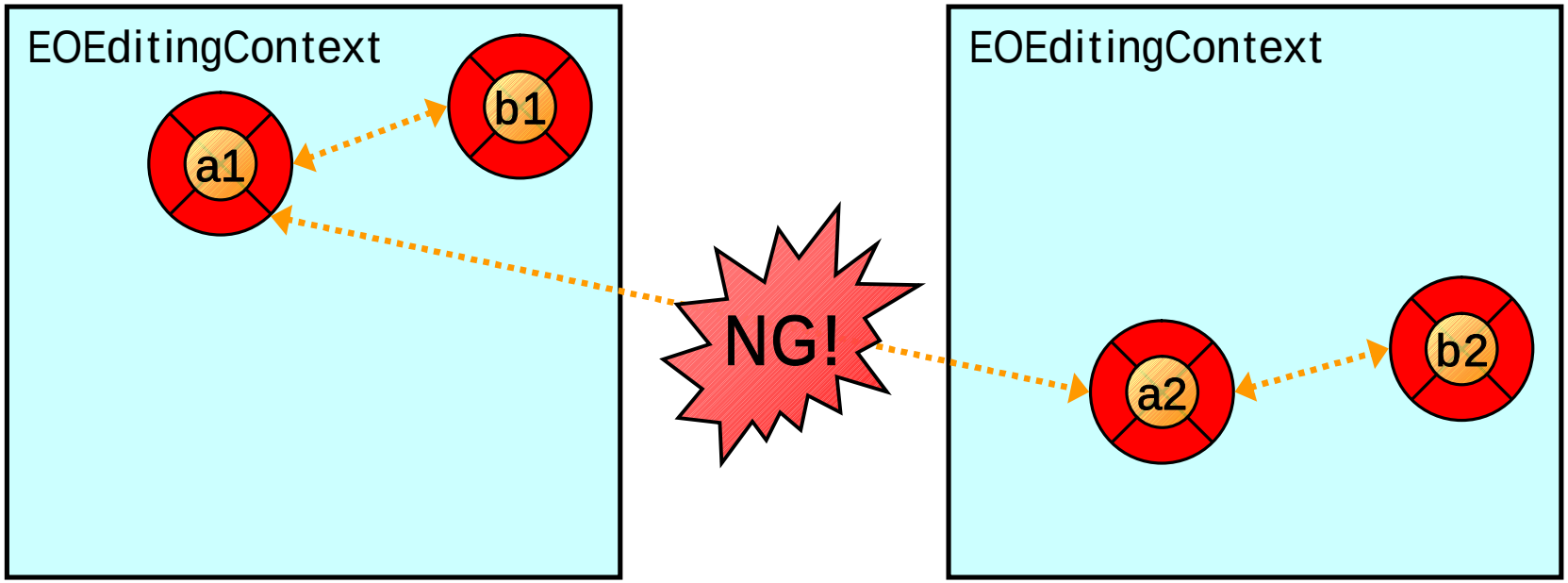
P62. Viewing the Object Graph in Context

- EC内のEOは、同じEC内のEOへの参照のみを持つ
  - WR: とりあえず、今はEOSharedEditingContextのことを考えない
- EC内のオブジェクトグラフ (のサブセット) は、グラフ内のどのオブジェクトからたどっても同じに見える
  - EOの状態は、オブジェクトグラフのたどり方に依存しない
- EC内に存在しないEOは、参照された時点でobject storeから透過的に取り出される
- Object storeにおける変更はECに反映される

- 具体的には以下のスライドで・・・

EC内のEOは、同じEC内のEOへの参照のみを持つ

P62. Viewing the Object Graph in Context

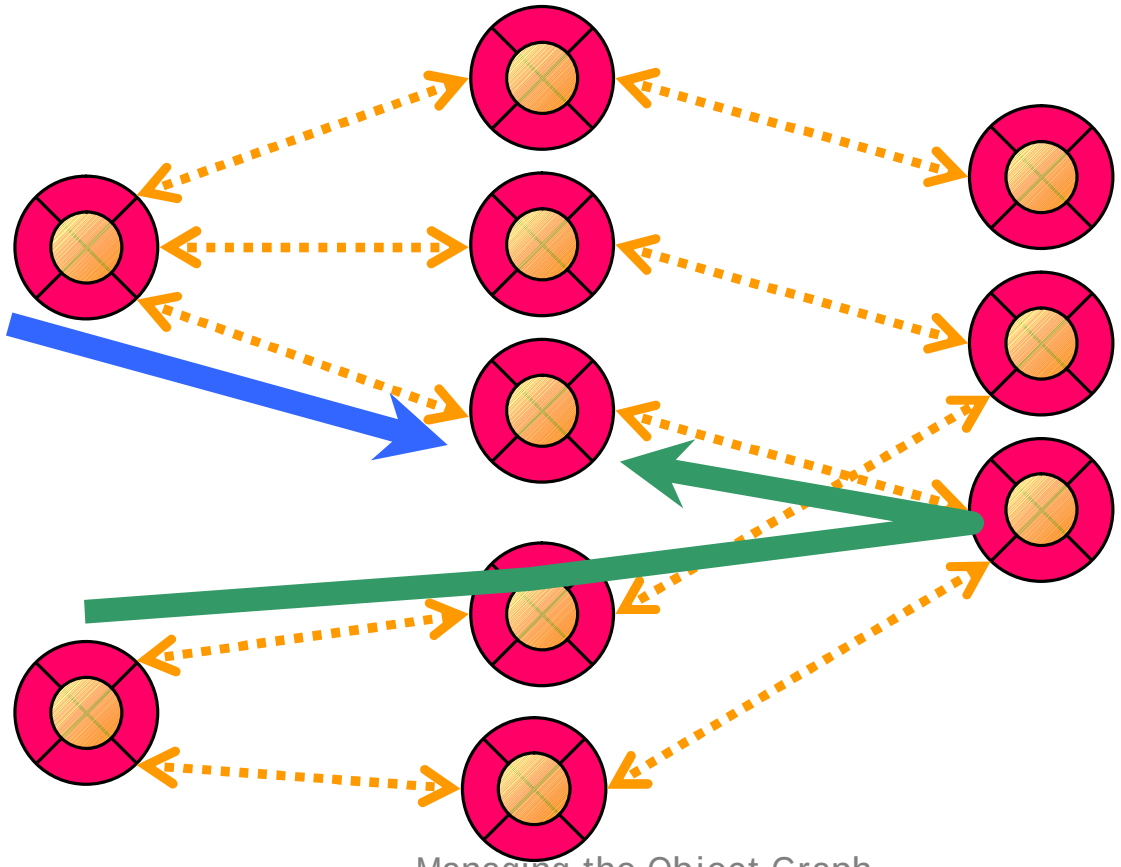


- EC間を跨いでリレーションシップを張ってはダメ！ということ

オブジェクトグラフのたどり方とEOの状態

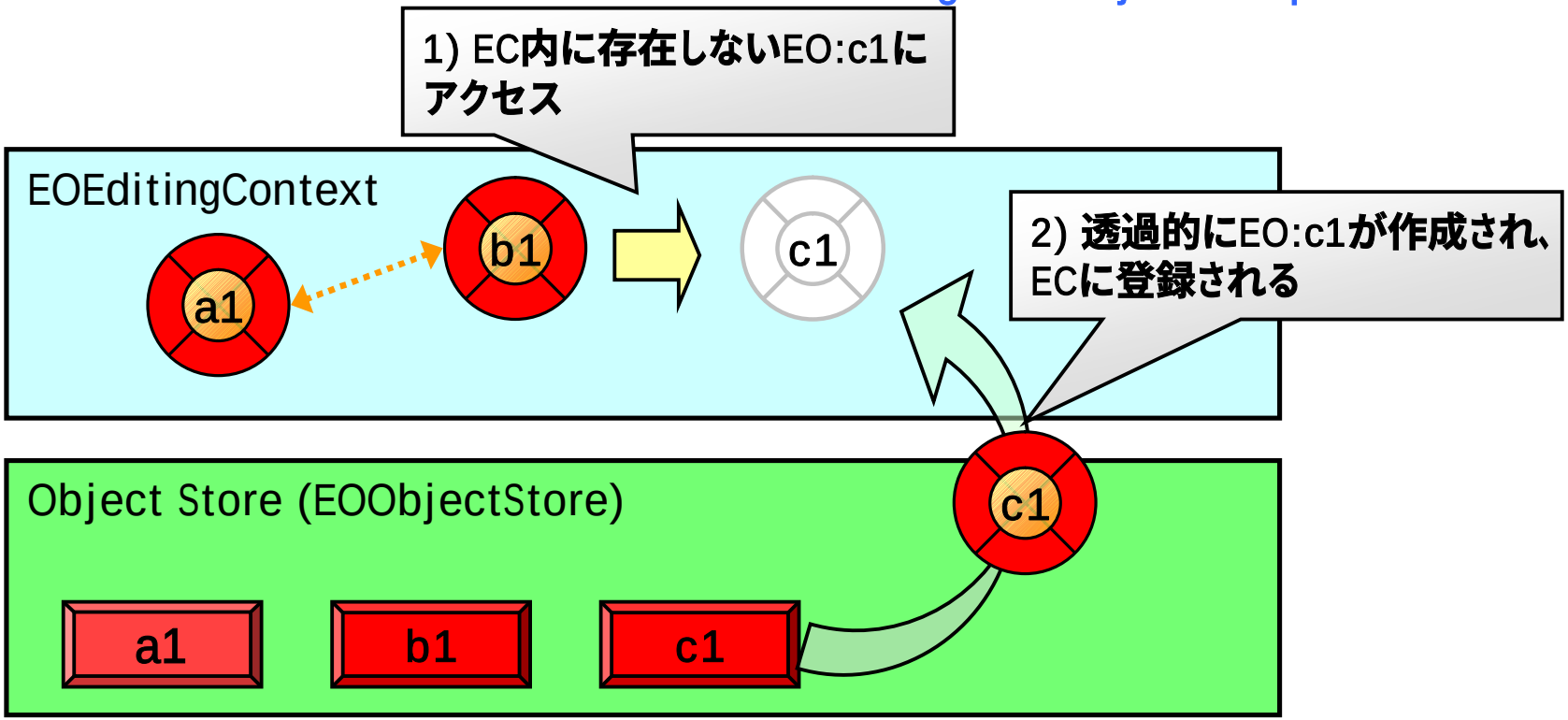
P62. Viewing the Object Graph in Context

- オブジェクトグラフをどこからたどっても、EOの状態は同じに見える



透過的に取り出されるEO

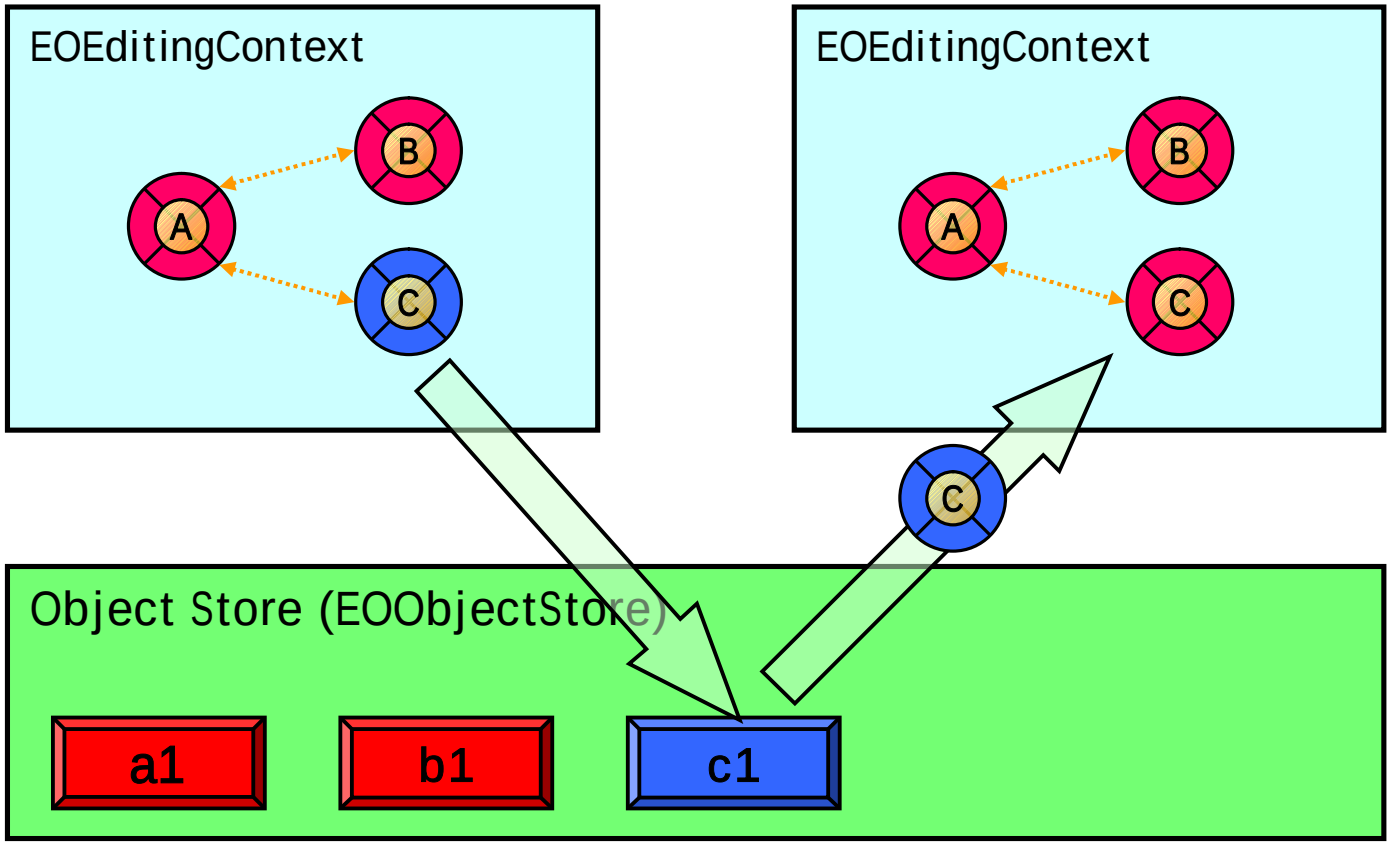
P62. Viewing the Object Graph in Context



- ECに存在しないEO:c1へのアクセスが発生
  → 透過的に（=明示的に処理をさせることなく）EO:c1が作成され、ECに登録される

Object storeにおける変更がECに反映

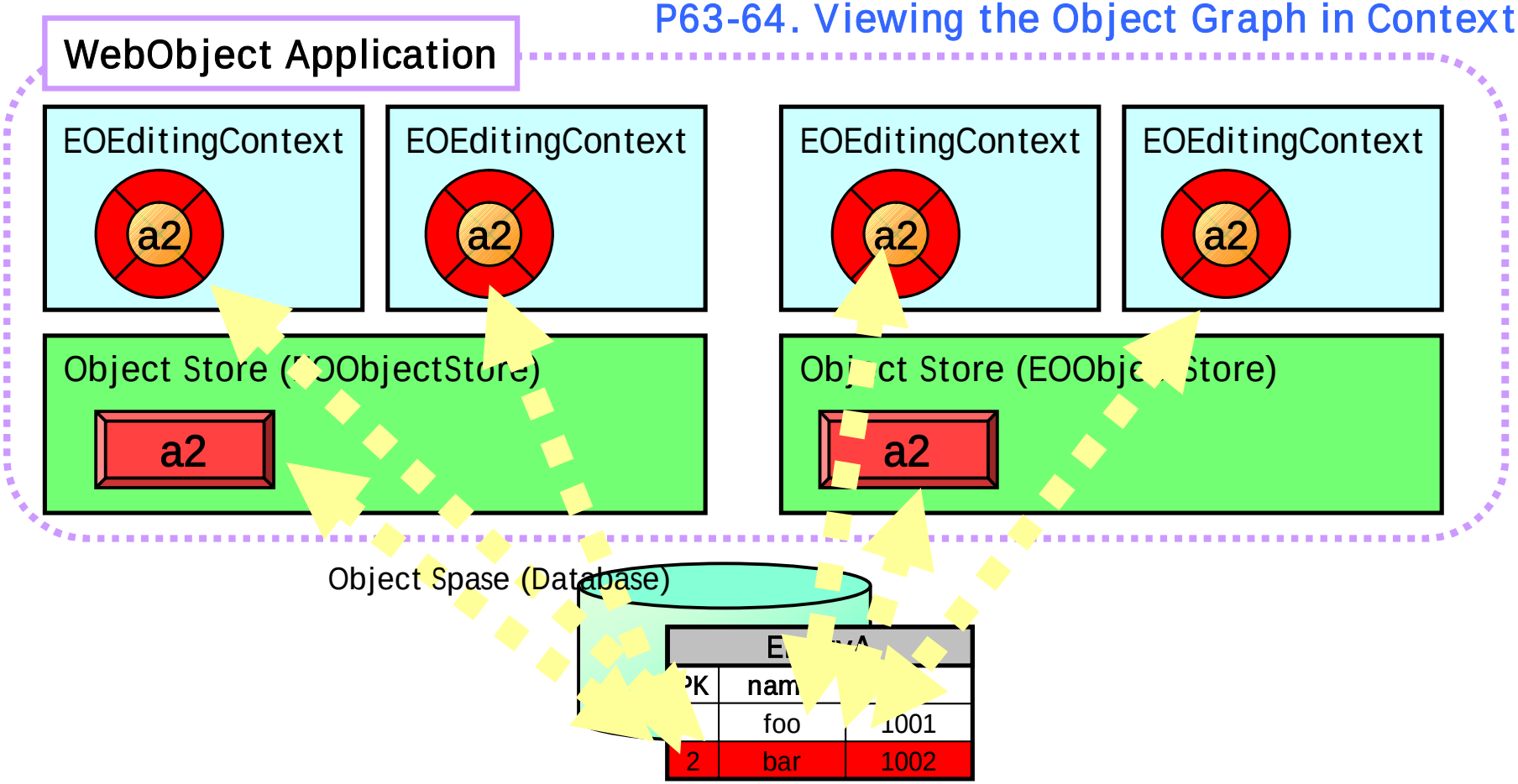
P62-63. Viewing the Object Graph in Context



- Objects Storeが複数存在する場合は・・・気をつけてね。
- 「EOFの同期とキャッシュ」
  (http://www.csus4.net/WO/)も参照のこと

オブジェクトの同一性

P63-64. Viewing the Object Graph in Context

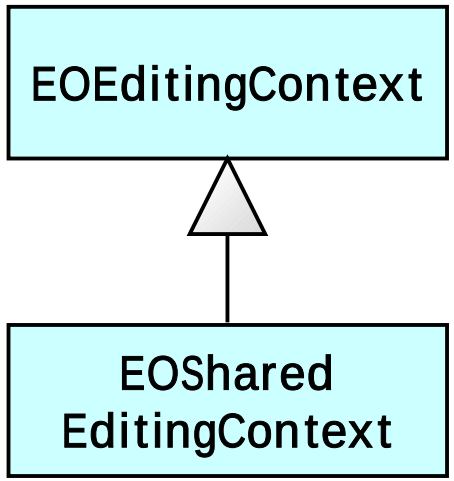


- **同じモノ(=レコード)はEOGlobalIDで一意に識別できる**
- **同じEOGlobalIDに対応するオブジェクトはレイヤ毎に複数存在できる。**

EditingContextの分類

P64-65. Understanding the Varieties of Editing Contexts

- クラスによる分類
  - EOEditingContext
  - EOSharedEditingContext
- 関係性による分類
  - 普通のEditing Context
  - Sessionが保持するdefaultのEditing Context
  - Parent-Child関係にあるEditing Context
  - (Nested：Parent-Child 関係と同じ意味)



  - Peer関係にあるEditing Context

クラスによる分類

P64-65. Understanding the Varieties of Editing Contexts

クラス	EOEditingContext	EOSharedEditingContext
EOを保持する目的	編集または参照	参照専用
EC内のEO編集可能?	OK	NG (参照専用)
他ECとのオブジェクト共有が可能?	NG	OK (※)
lock() / unlock() 必要?	必要	不要 ($)

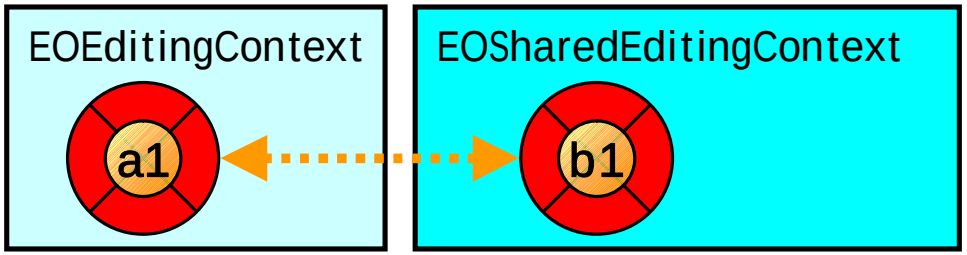
※：sharedEC内のEO→他EC内のEOなるリレーションシップを設定してはいけない (詳細は次ページ)

$：参照専用なんだから、当たり前・・・

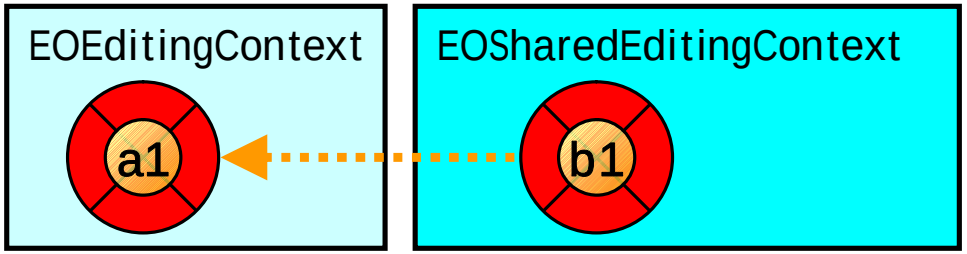
EOSharedEditingContextとリレーションシップ

P65. The Shared Editing Context

- NG



- OK

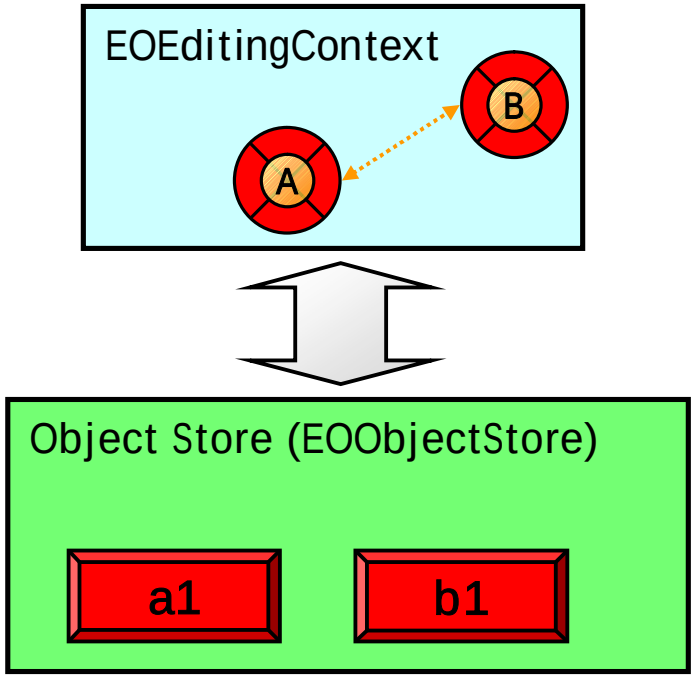


  - One-Sided Relationshipについては2章を参照のこと
- EOSharedEditingContextの使い方
  - Look upテーブルを置くと良いかもね
- WR:識者の意見を伺いたいところではある・・・

関係性による分類：普通のEC

P64. Understanding the Varieties of Editing Contexts

- new EditingContext()して**生成する**
- **通常、親**object store**は**EOObjectStoreCoodinator
- **プログラマが明示的に**lock()/unlock()**する必要あり**



関係性による分類：SessionのDefault

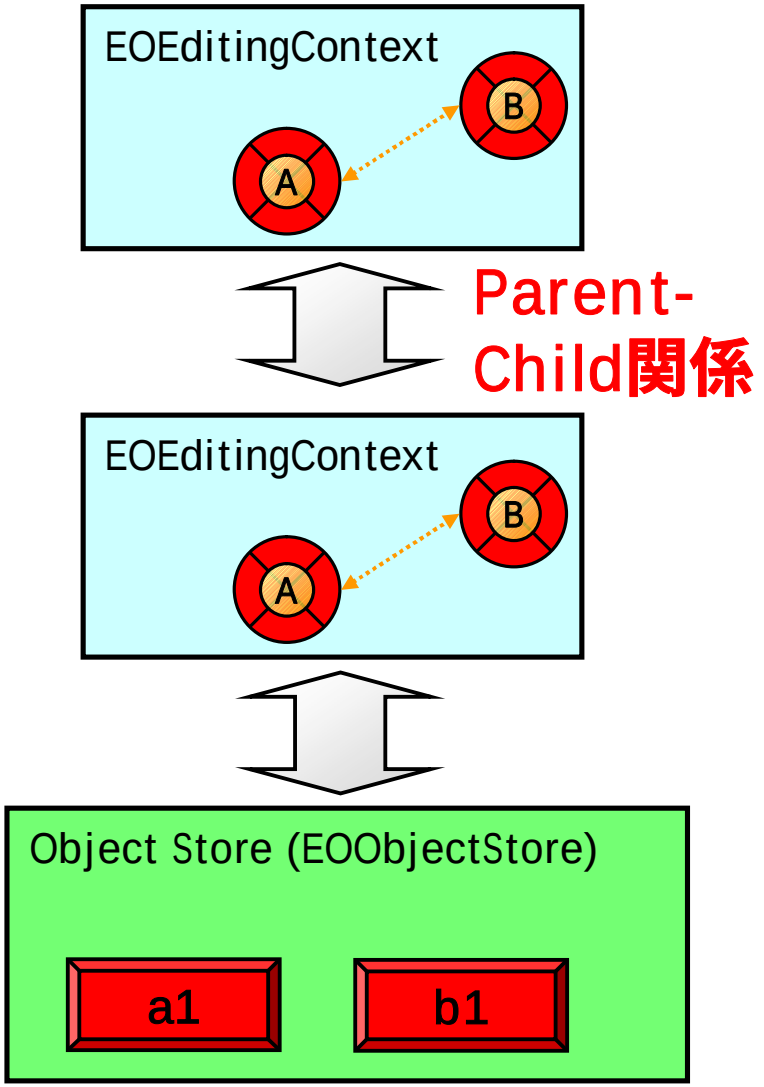
P64. Understanding the Varieties of Editing Contexts

- Sessionのデフォルトの EditingContext
  - WOSessionにより生成され、保持される
  - session().defaultEditingContext()で参照可能
- 親object storeはEOObjectStoreCoodinator
  - WR:ですよね?
- WOSessionがlock()/unlock()の面倒を見てくれる。
  - プログラマはlock()/unlock()を気にする必要なし！
- 以下のようなデータを置くのに適する
  - Sessionのライフサイクルと対応したライフサイクルを持つデータ
  - Sessionの同一性(　利用するユーザーの同一性) に関連するデータ

関係性による分類：Parent-Child関係

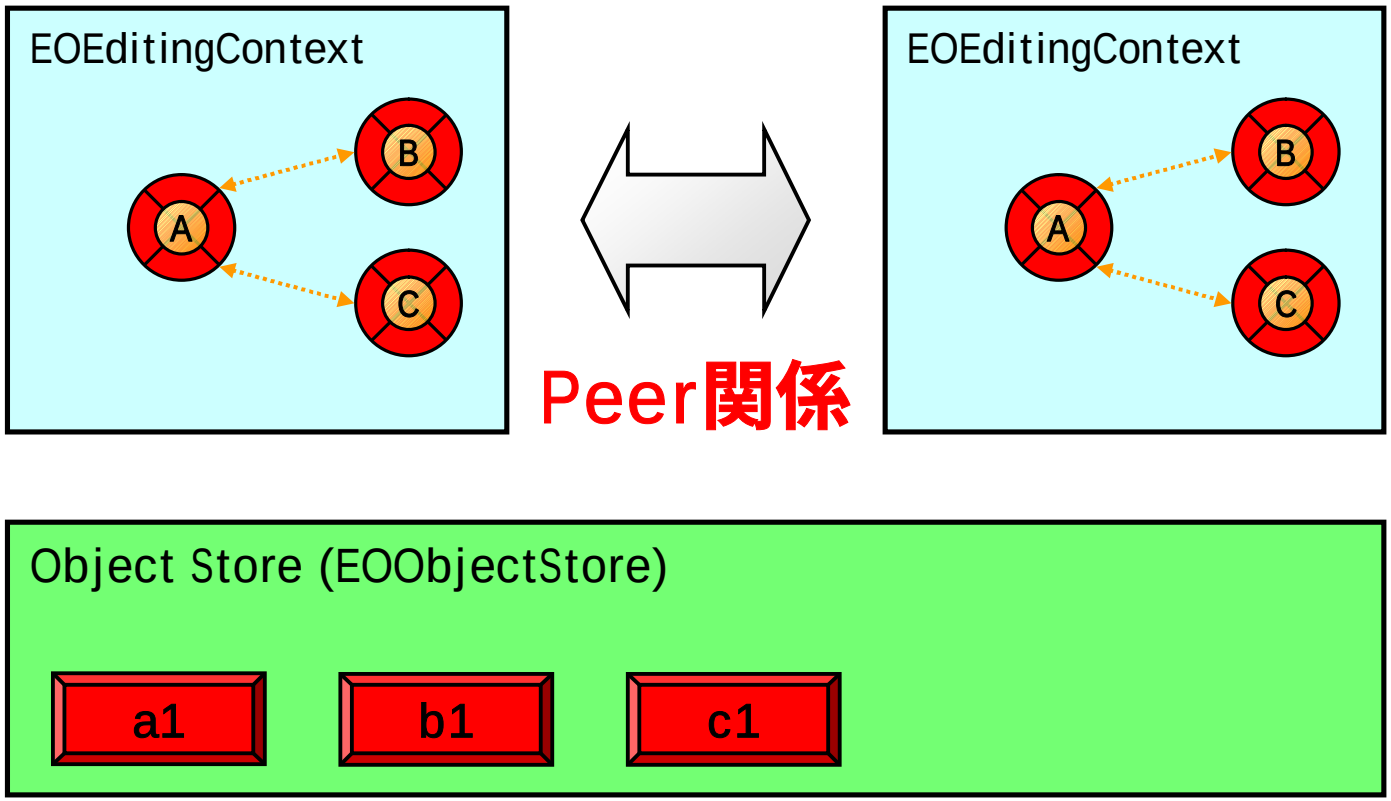
P64. Understanding the Varieties of Editing Contexts

- **親objects storeがECである場合の関係性**
  - new EOEditingContext(parentEC)
- **更新時の振る舞い**
  - **子ECでコミット→親ECにのみ反映**
- **プログラマはlock()/unlock()する必要がある**
- **入れ子になったダイアログ、ウィザード的を機能を実現する際に有用（かもしれない・・・）**



関係性による分類：Peer関係

P64. Understanding the Varieties of Editing Contexts



- Peer**関係** = **親**Object store**が同一の**object store **インスタンスである**EC**同士の関係**

関係性による分類：比較

	(普通の)	Session default	Child
親Object Store	EOObject Store Coodinator	EOObject Store Coodinator	EOEditing Context (parent)
ロック必要?	**必要**	**不要**	**必要**
アクセス/生成方法	session(). defaultEditingContext()	new EOEditingContext()	new EOEditingContext(parentEC)
用途		**Sessionのライフサイクルに対応したデータの編集・参照**	**入れ子型の編集機能**

※：Peerに関しては、位置づけが他と異なるため、比較対象から除外した

目次

- Understanding the Object Graph
  - オブジェクトグラフとediting contextの基本
- Rules of Engagement
  - ルールは守りましょう
- Seeing State Changes in the Object Graph
  - オブジェクトグラフの状態変化を監視する方法
- Debugging Object Graph Problems
  - オブジェクトグラフにまつわるトラブルシュート
- Locking, Concurrency, and EOF
  - ロックと並行性の問題
- Managing Object Freshness
  - キャッシュ問題

Rules of Engagement

P65-67.

- Keep object relationships in the same editing context
- Work with objects only in an editing context
- Modify objects at appropriate times

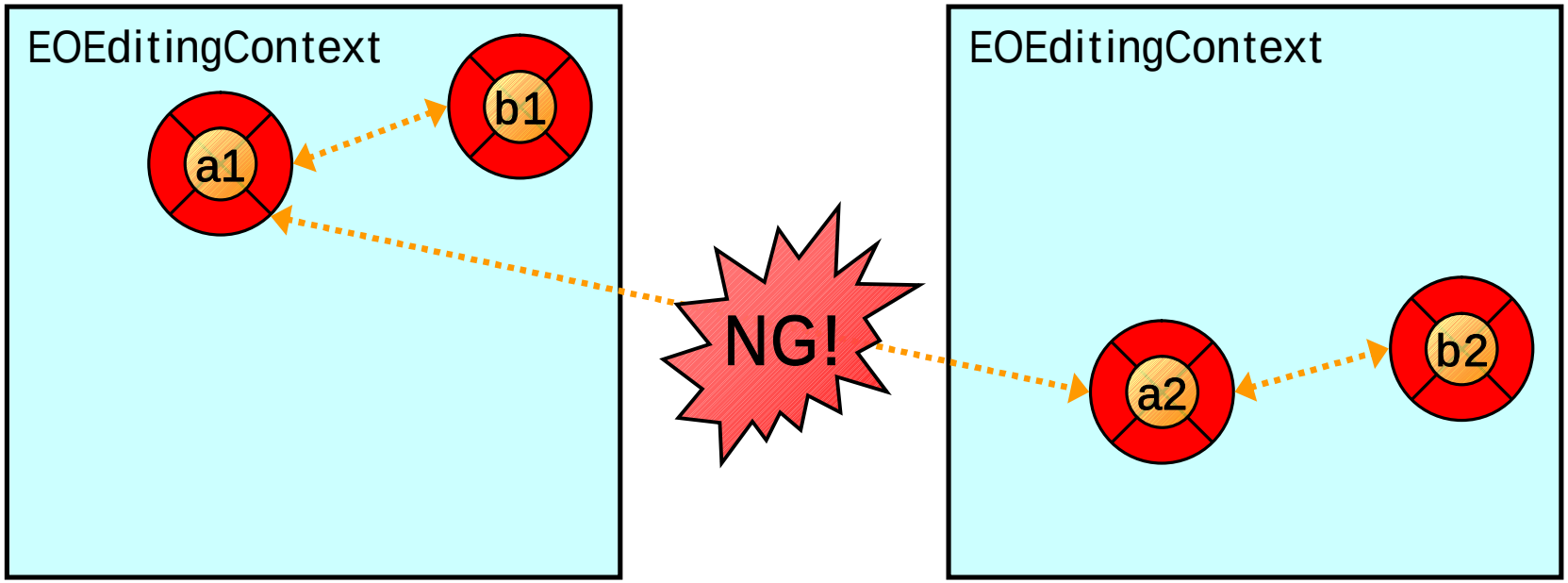
使用上のルール

P65-67. Rule of Engagement

- リレーションシップは、あるEC内のオブジェクト間で張りなさい
  - 逆に言えば・・・「ECを跨いでリレーションシップを張ってはいけません」
- 1つのEC内でオブジェクトを扱いなさい
  - 逆に言えば・・・「あるEC内のオブジェクトを他のECに登録してはいけません」とか
- オブジェクトの修正は適切なタイミングで
  - EOのvalidation処理との絡みに要注意！

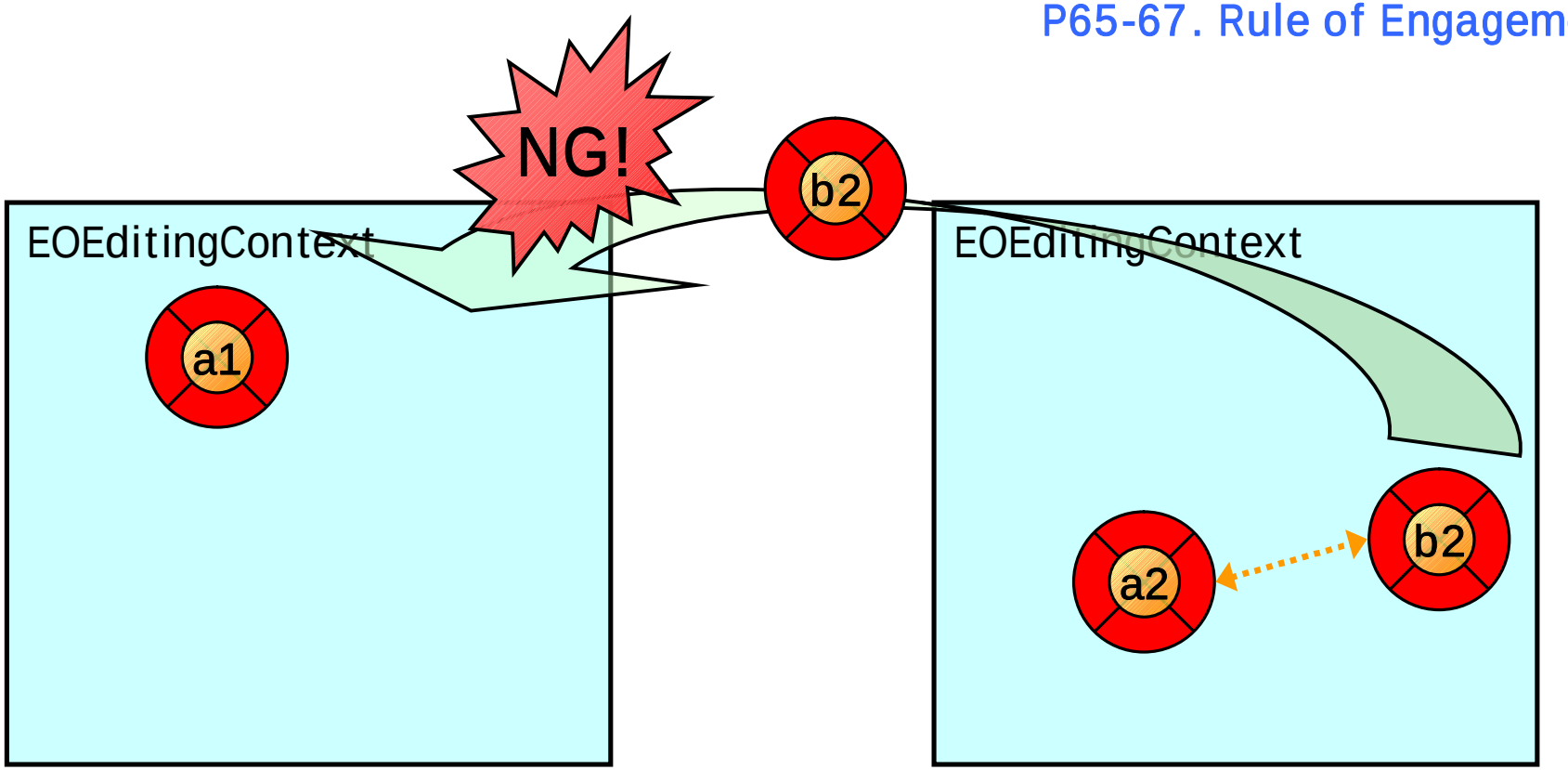
リレーションシップは、あるEC内のオブジェクト間で張りなさい

P65-67. Rule of Engagement



1つのEC内でオブジェクトを扱いなさい

P65-67. Rule of Engagement



- あるEC内のEOを別のECに持って行ってはイケナイ。

オブジェクトの修正は適切なタイミングで

- Validateされた後に属性値を変更したりしちゃダメ
- ということだと思う・・・

目次

- Understanding the Object Graph
  - オブジェクトグラフとediting contextの基本
- Rules of Engagement
  - ルールは守りましょう
- Seeing State Changes in the Object Graph
  - オブジェクトグラフの状態変化を監視する方法
- Debugging Object Graph Problems
  - オブジェクトグラフにまつわるトラブルシュート
- Locking, Concurrency, and EOF
  - ロックと並行性の問題
- Managing Object Freshness
  - キャッシュ問題

Seeing State Changes in the Object Graph

P67-78.

- Introducing the EOStateTransition Interface
- The CooperatingEditingContext
  - Recording the Object Graph Changes
  - Handling Deletions
  - Handling Insertions
  - Handling Updates
  - Processing a Save
  - Sending the “has” Notifications
  - Sending the willUpdate Notifications
  - Working with Nesting Editing Contexts
  - How to Use CooperationgEditingContext

オブジェクトグラフの変化を見る

P67-. Seeing State Changes in the Object Graph

- CooperationgEditingContext in PracticalWebObject.frameworkを使いなさい
  - EOEditingContextのサブクラス
  - 特徴
    - オブジェクトグラフの状態を監視できる
      - 内部でinsert/delete/update処理をフックしている
  - 使用方法
    - Sessionクラスのコンストラクタを修正する
    - もしくは普通にnewする
  - 詳細は書籍を参照あれ・・・

目次

- Understanding the Object Graph
  - オブジェクトグラフとediting contextの基本
- Rules of Engagement
  - ルールは守りましょう
- Seeing State Changes in the Object Graph
  - オブジェクトグラフの状態変化を監視する方法
- Debugging Object Graph Problems
  - オブジェクトグラフにまつわるトラブルシュート
- Locking, Concurrency, and EOF
  - ロックと並行性の問題
- Managing Object Freshness
  - キャッシュ問題

Debugging Object Graph Problems

P78-83.

- Finding and Preventing Cross-Editing Context Relationships
  - Using DBC to Prevent Cross-Editing Context Relationships
- Defeating Referential Integrity Problems
  - Logging Referential Integrity Problems
  - Detecting Referential Integrity Problems

オブジェクトグラフ関連の問題デバッグ方法

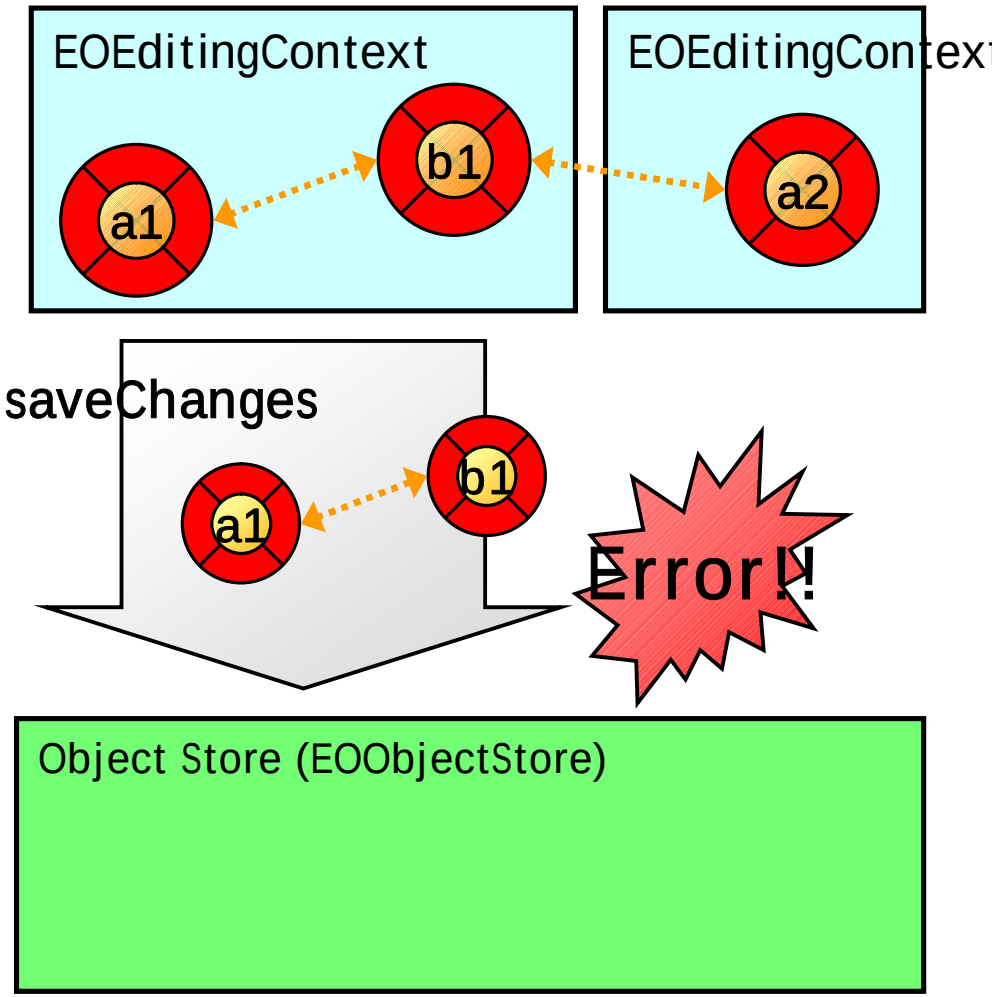
P78-. Debugging Object Graph Problems

- Finding and Preventing Cross-Editing Context Relationships
  - ECを跨いだEO間のリレーションシップを実行時に検出する方法
- Defeating Referential Integrity Problem
  - モデルレベルで定義された参照整合性にまつわる問題に対処する

Cross-Editing Context Relationships

P78-. Finding and Preventing Cross-Editing Context Relationships

- **ECを跨いでリレーションシップを設定してはいけない**
- saveChangesしたタイミングでエラー発生



エラーメッセージ

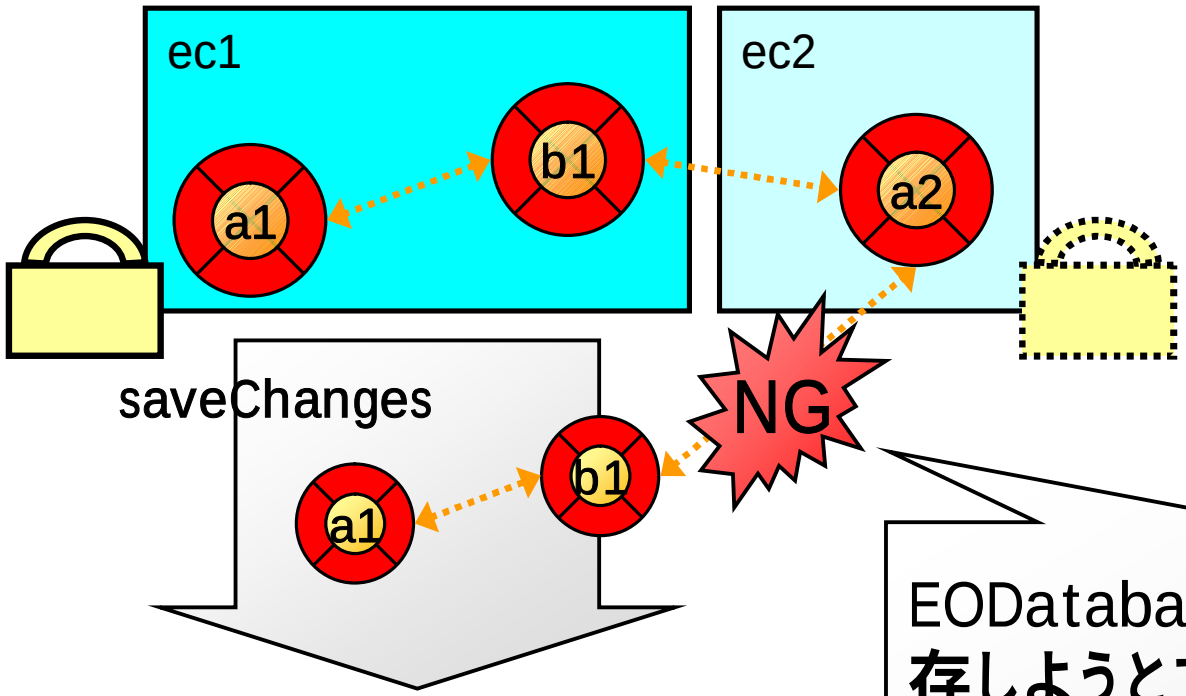
P79. Finding and Preventing Cross-Editing Context Relationships

```
java.lang.IllegalStateException:
Cannot obtain globalId for an
object which is regstered in an
other than the databaseContext's
active editingContext,

object: {values = {stringValue =
"default"; relatedObject = "null";
id = ; etc., this =
"<com.apress.practicalutitiles.test
s.CooperatingEditingContextTestObje
ct etc

databaseContext:
EODatabaseContext@57aa53,
object's editingContext:
EOEditingContext@70c0d3,
databaseContext's active
editingContext:
EOEditingContext@4cc588
```

- **エラーの意味**
  1. **保存対象のEOが、保存処理実行中のEC内のオブジェクトグラフ内に見つからない！**
  2. **誤ったECに登録されているEOの値**
  3. **EOのECと、aciveなECが異なる！**

エラーメッセージ

P79. Finding and Preventing Cross-Editing Context Relationships



EODatabaseContextはa2を保存しようとするが、a2の属するec2はEOObjectStoreCoodinatorのロックを持っていないため、処理を継続できない

防止方法

P78-79. Finding and Preventing Cross-Editing Context Relationships

- DBC(Design by Contract)を使いなさい。
  - リレーションシップ関連のアクセッサに、事前条件Contractを埋め込む
  - 事前条件Contractで、「処理対照のEOがECに登録されてるかどうか？」をチェックする

```
Public void setChildGroups(NSMutableArray newValues){
  /** require [same_ec] (forall i: {0..newValues.count() -1) #
          ((EOEnterpriseObject)newValues.objectAtIndex(i)).
               editingContext() == editingContext());
  **/

  takeStoredValueForKey(newValues, "childGroups");
}
```

など・・・（詳細は書籍を参照）

参照整合性にまつわる問題

P79. Defeating Referential Integrity Problems

- 2種類あることに注意されたい
  - EOModelレベルの参照整合性
    - モデルファイル(*.eomodeled)で定義される
    - cascade on delete, owns destination, …
  - RDBMSレベルの参照整合性
    - DBスキーマ(SQL DDL)で定義される
    - PK→FKの参照整合性, cascade on delete, …
      - RDBMS毎に参照整合性の機能が異なる
- 両者で記述できる内容が異なる
  - 故に、ミスマッチが発生する可能性がある
  - たとえば、EOFを使わずにRDB内のデータをいじった場合など・・・

ミスマッチが発生する典型的な例

P79. Defeating Referential Integrity Problems

- vertical inheritance使用時は、RDBのPK→FK制約が使用できない
  - WR: verticalだけでなく、horizontalも同様と思われる
- EOFが発行するSQLステートメントの順番に起因する問題
  - トランザクション途中で参照整合性違反となってしまう
  - 対処方法
    - RDBMS側の参照性整合性チェックをトランザクション終了時点まで遅らせる (DEFFERD)
      - チェック遅延をサポートしていないRDBMSもあり・・・
    - WR: アダプタレベルでステートメントの発行順序を変更する方法もあり？
      - 汎用性を持たせるのが大変そうではある。

to-oneリレーションシップと参照整合性

P79. Defeating Referential Integrity Problems

- EOFのto-oneリレーションシップ処理
  - to-oneリレーションシップのdestination側に対応するレコードがRDBに存在しない場合、EOFはダミーのオブジェクトを作成する
  - ダミーオブジェクトの属性にアクセスしたタイミングでNullPointerExceptionが投げられる
    - dest側レコードが無いことがこのExceptionの原因と、気づくことができるだろうか？ いや、ない。
- ある意味、参照整合性のミスマッチとも言える。かも
  - WR：どうだろう・・・

参照整合性エラーのロギングと修正

P81-82. Logging Referential Integrity Problems

- デレゲートdatabaseContextFailedToFetchObject
  - ここに適切な処理を実装しなさい
- では、何を実装すべきか？
  - ミスマッチを修正しますか？
    - 理論的には、ミスマッチを訂正して、RDB上にある誤ったデータから、正しいオブジェクトグラフを生成する処理を実装することも可能
    - が、それに価値がある場面に出くわしたことが無い。
  - ロギングして例外を投げましょう：書籍 Listing 3-6参照
    - ただし、refaultが必要なことに注意
      - WR：List内のコメントの内容がよくわからない！
- データの修正：悪い臭いは元から断たなきゃダメ!
  - EOレベルで直す方法もあるが・・面倒
  - RDBにSQLを直に発行して地道に直しましょう

参照整合性エラーの検出と対処

P82-83. Detecting Referential Integrity Problems

- 検出方法
  - Listing 3-7を見よ
    - 生成に失敗しているオブジェクトには、snapshotが存在しないことに着目
  - 手順
    - オブジェクトのglobalIdを取得
    - globalIdに対応するsnapshotがあるかどうか？をチェックする
  - 注意点
    - 先のデレゲートと同時に使用できない
      - <eo>.willRead()を呼んだ瞬間に、デレゲートが実行されてしまうため
- 検出後の対処方法
  - そのオブジェクトへの参照をnullにせよ
  - さもないと、snapshot関連の例外が投げられる

目次

- Understanding the Object Graph
  - オブジェクトグラフとediting contextの基本
- Rules of Engagement
  - ルールは守りましょう
- Seeing State Changes in the Object Graph
  - オブジェクトグラフの状態変化を監視する方法
- Debugging Object Graph Problems
  - オブジェクトグラフにまつわるトラブルシュート
- Locking, Concurrency, and EOF
  - ロックと並行性の問題
- Managing Object Freshness
  - キャッシュ問題

Locking, Concurrency, and EOF

P83-83.

- What Needs to Be Locked?
- How to Lock Correctly
  - Locking the Java Way
  - The Complexity of Locking
  - The Impact of Direct Action on Locking
  - Toward a Locking Practice
  - Locking Other EO Objects
- Debugging Locking Problems
  - Debugging a Deadlocked Application

ロックすべきか、ロックせぬべきか

P83. Locking, Concurrency, and EOF

- ECをロックしたほうが良いの？
- 本書の主張は・・・「ロックしなさい！」
- その理由
  - そもそも、並行性を考慮しなくて良いJavaアプリケーションなど無い。WebObjectsアプリケーションも例外ではない
    - WR：うーん・・・
  - 運用時に不再現バグに悩むぐらいなら、ロックしておいたほうがいい
    - WR：まぁ、ねぇ・・・

「何を」ロックすればよい？

P84. What Needs to Be Locked?

- editing contextをロックする
  - enterprise objectをロックするのではない
- newしたediting contextをロックする
  - defaultEditingContextとsharedEditingContextのロック処理は、フレームワークが面倒を見てくれる → ロックする必要なし！
  - それ以外のediting contextについては、開発者がロック用のコードを書く必要がある

EOAccessレイヤにおけるロック

P84. What Needs to Be Locked?

- WebObjects 5.2における変更
  - EOAccessレイヤのオブジェクトのロック方法が変更された
  - 詳細は“What’s New in WebObjects 5.2”を見よ
- ロック方法
  - EOAccessレイヤのオブジェクトにアクセスする前に、EOObjectStoreCoodinatorをロックせよ
    - 通常、EOObjectStoreCoodinatorはアプリーケーションに1つだけ存在する
      - EOObjectStoreCoodinatorをnewするコードを書いている場合は、この限りではない
  - すると、EODatabaseContext、EOFスタックの下位オブジェクトがロックされる

ロックの仕方

P84. How to Lock Correctly

- ロックの実現方式
  - NSRecursiveLockで実現される
  - NSRecursiveLockの特徴
    - 複数回ロックしてもデッドロックしない
    - ロックした回数とアンロックした回数が同じ場合、ロックが解除される
- ロック時の注意点
  - アクセスする前にロックせよ：当たり前！
  - ロックしたオブジェクトをアンロックせよ
    - さもないと、デッドロックする

Javaコードでロックする

P85. Locking the Java Way

- **注意点**
  - **新規作成または、参照を取得したら、すぐにロックせよ**
  - **必ずアンロックさせるために、**try-finally**ブロックを活用せよ**
  - EC**の寿命が複数回の**request-response**サイクルに跨る場合は、ロックしたままにしないこと**

```
EOEditingContext myEC = new EOEditingContext();
try {
    myEC.lock();
    // myECについて処理を実行
}
finally {
    myEC.unlock();
    myEC.dispose();
}
```

- Needless to say, WebObjects is complicated enough that this nice, clean solution is not going to get us very far.

- The application for this chapter had a lot of messages tracing editing context locking and requrest-response loop for Component Actions and Direct Actions with sessions.
- These will help to illustrate the problems to be overcome.
- The Fundamental problem has to do with the longevity of editing contexts.
- Unlike the preceding example, they usually live over several cycles of the request-response loop.
- Each request for a paticular session may be handled by a different worker thread.
- If we just create the editing context and leave it locked,the next request for that session will block unless it just happens to get handled by the same worker thread.

ロック/アンロックとrequest-responseループ

P85. The Complexity of Locking

- 望ましいロック/アンロックのタイミング
  - ロック： request-responseループの始めにロック
  - アンロック：ループの終わりにアンロックすべき
- ロック/アンロック実装アイディア
  - WO??? オブジェクトのawake()とsleep()メソッドにロック/アンロックを実装しては？
    - WOApplication?
      - これはさすがにムリ
    - WOSession?
    - WOComponent?
      - 以下、2つのオブジェクトについて検討する

awake() / sleep()の呼び出し順序（1）

P86. The Complexity of Locking - Listing 3-8.

#	オブジェクト	メソッド
1	Application	awake
1	Session	awake
1	aComponent	new
1	aComponent	awake
1	aComponent	appendToResponse
1	aComponent	sleep
1	Session	sleep
1	Application	sleep

aComponentのコンストラクタでECをnewしているならば、
ロック: aComponent.awake()
アンロック:
aComponent.sleep()
で良いように思えるが・・・

#・・・ワーカースレッドのID

リンクを2回連続クリックすると・・・

P86-87. The Complexity of Locking - Listing 3-9.

#	オブジェクト	メソッド
2	(Handler)	(Dispatch Request)
2	Application	awake
2	Session	awake
2	prevPage	awake
2	prevPage	invokeAction
2	nextPage	new
2	nextpage	awake
3	(Handler)	(Dispatch Request)
3	Application	awake
2	nextPage	appendToResponse
2	prevPage	sleep
2	nextPage	sleep
		con'd ...

処理がブロックされている

リンクを2回連続クリックすると・・・

P86-87. The Complexity of Locking - Listing 3-9.

#	オブジェクト	メソッド
2	Session	sleep
2	Application	sleep
3	Session	awake
3	prevPage	awake
3	nextpage	awake
3	nextPage	appendToResponse
3	prevPage	sleep
3	nextPage	sleep
3	Session	sleep
3	Application	sleep

ブロックが解除され、処理を継続する

→ WebObjects(WOF)は同一のSessionへの2つのリクエストを並行して処理しない

セッションを持っているが、それにアクセスしないDirect Actionでは？

P87. The Impact of Direct Action on Locking - Listing 3-10.

#	オブジェクト	メソッド
0	(Handler)	(Dispath Request)
0	Application	awake
0	aDirectAction	(start)
0	aComponent	new
0	aComponent	awake
0	aDirectAction	(end)
0	aComponent	appendToResponse
0	aComponent	sleep
0	Application	sleep

wosid=XXXを持っている

Sessionがawakeされない

セッションを持っていて、それにアクセスするDirect Actionでは？

P88. The Impact of Direct Action on Locking - Listing 3-11.

#	オブジェクト	メソッド
1	(Handler)	(Dispath Request)
1	Application	awake
1	aDirectAction	(start)
1	aComponent	new
1	aComponent	awake
1	aDirectAction	(end)
1	aComponent	appendToResponse
1	Session	awake
1	aComponent	sleep
1	Session	sleep
1	Application	sleep

wosid=XXXを持っている

appendToResponse内でSessionにアクセスしている

Direct Actionでロックしてみる

P88. The Impact of Direct Action on Locking

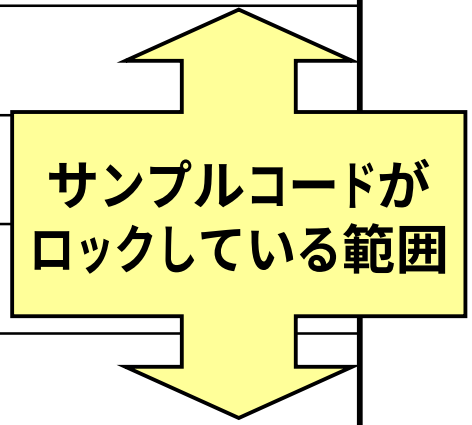
- サンプルコードを見よ。

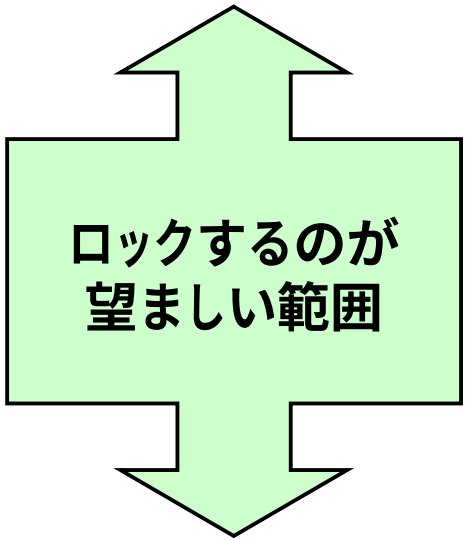
- これは不十分である
  - ページ生成時（generateReponse()実行時）にECが参照されるが、このときはロックが解除されている！
  - WR：「うお、結構厳しい条件だな・・・」とか思いません？

ロックして欲しい範囲・・・

P87. The Impact of Direct Action on Locking - Listing 3-10.

#	オブジェクト	メソッド
0	(Handler)	(Dispath Request)
0	Application	awake
0	aDirectAction	(start)
0	aComponent	new
0	aComponent	awake
0	aDirectAction	(end)
0	aComponent	appendToResponse
0	aComponent	sleep
0	Application	sleep





じゃあ、どーやってロックすんのよ！（怒）

P89. Toward a Locking Practice

- Component Actionしか使用しないアプリケーション
  - WOSessionかWOComponentのawake()とsleep()メソッドがよさそう
- セッション有りのDirect Actionを使用するアプリケーション
  - ワザとSessionにアクセスする処理を追加した上で、WOSessionのawake()とsleep()に。
- セッション無しのDirect Actionを使うアプリケーション
  - 考えないと・・・。うむむ。