Practical Webobjects
Chapter 2 (Page 31-60):

EO Modeling Techniques

WR

WR@Csus4.net

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

目次

- Inheritance
- Relationships
- Attribute Value Types
- Prototypes
- Debugging JDBC Connection Problem
- EOModeling with Eclipse
- Summary

はじめに

- “EnterpriseObjects”と “Using EOModeler” **を読んでおくこと！**

Inheritance

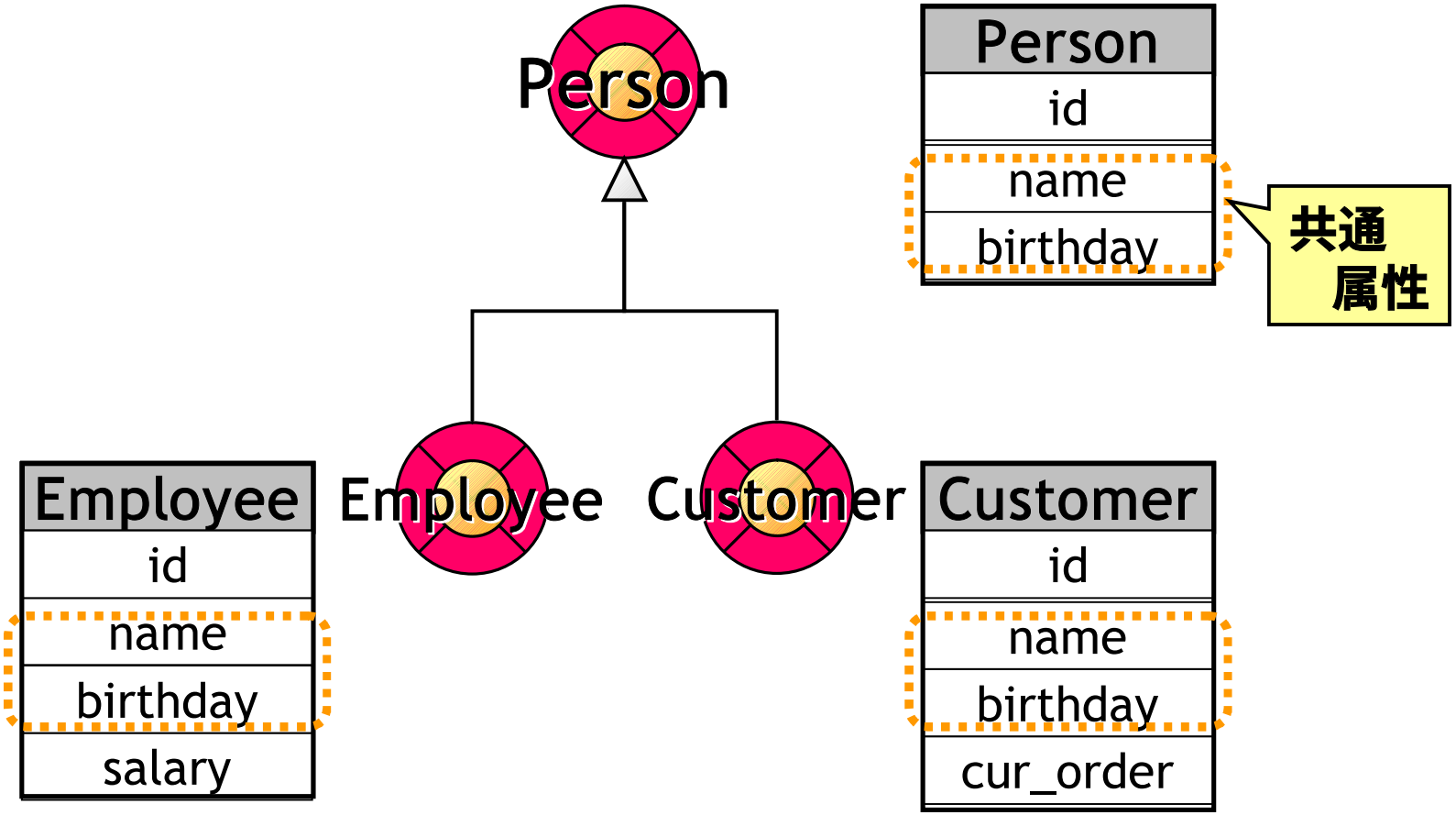
- Inheritance Implementation Details
- Inheritance and Primary Keys
- Inheritance and the Overly Required Attribute
- Inheritance and the Impossible Constraint
- Inheritance and Multiple Models
- Inheritance, Multiple Models, and SQL Generation

Inheritance Implementation Details

- 継承を用いるならば、パフォーマンス上の問題を回避するため、**継承の実現方式**を理解しておかなくてはならない。
- EOFは**継承の実現方式**として、以下の3つの方式を提供する。
  - 単一テーブル継承
  - 垂直継承
  - 水平継承

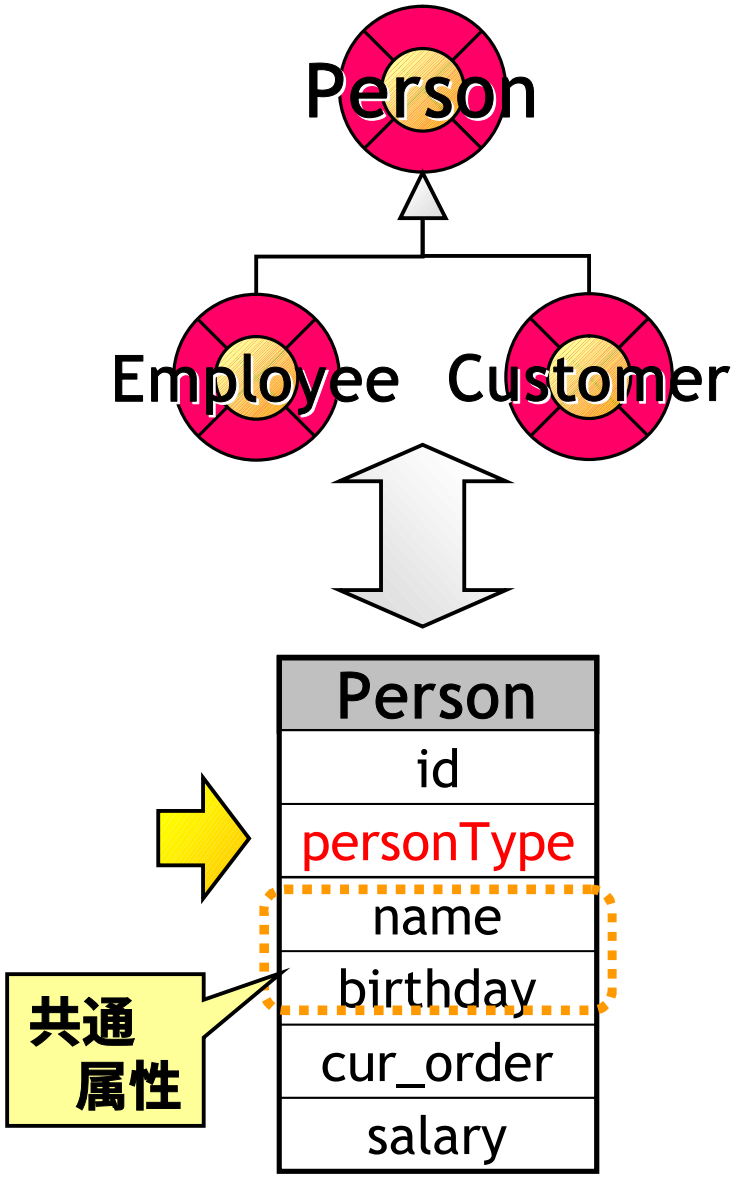
継承関係サンプル

- 以下の継承関係をモデリングする。



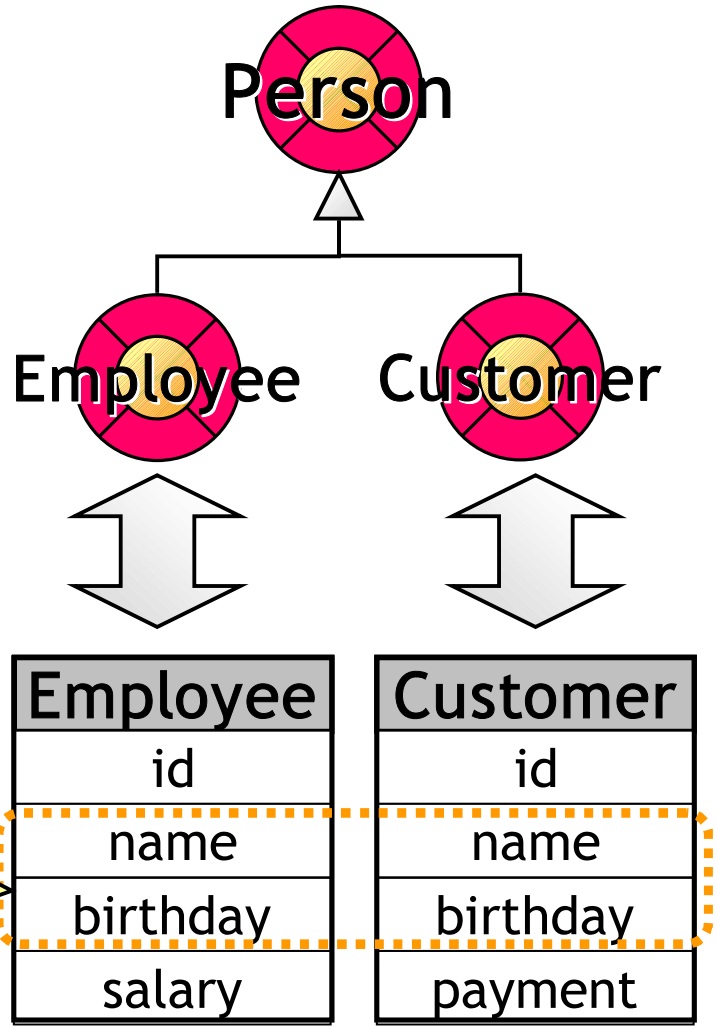
Single-Table Inheritance

- 親Entityと子Entityを単一のテーブルに格納する
- パフォーマンス良い
- 継承されたアトリビュートは複製されない
- サブクラス（やスーパークラス）の識別に”type”を用いる
- 注意点
  - 使うRDBMSのnull値の格納方式をチェックしておけ



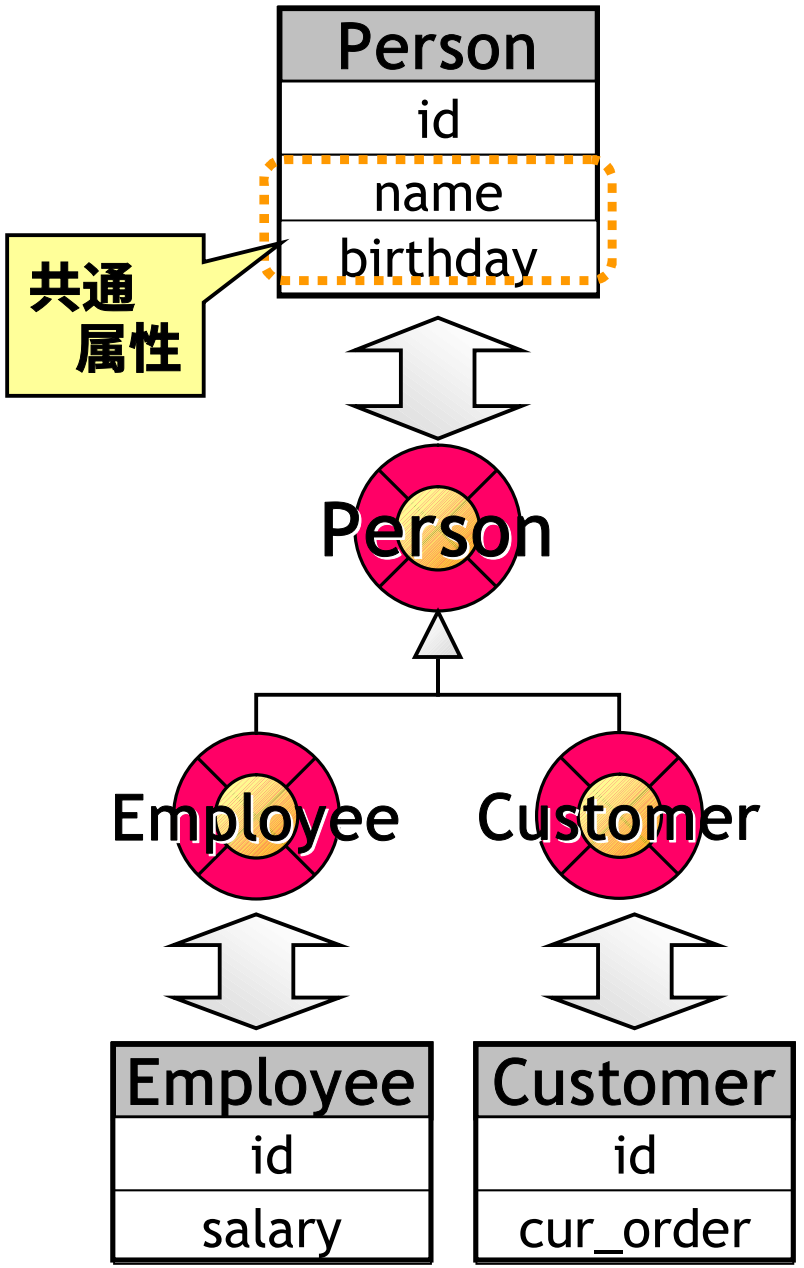
Horizontal Inheritance

- **継承アトリビュートを含め、すべての属性が複製される**
- **問題点**
  - **親Entity取得のために複数回fetchが必要**(deep fetch)
- **注意点**
  - **例はPerson がabstractなエンティティの場合**
  - **非abstractな場合、別途Personテーブルが必要**



Vertical Inheritance

- **継承ツリー間で属性が共有されない**
- **ほとんど使用されない**
- **問題点**
  - **子Entity取得のためにJOINが必要**
  - **親Entity取得のために複数回fetchが必要**
    (deep fetch)



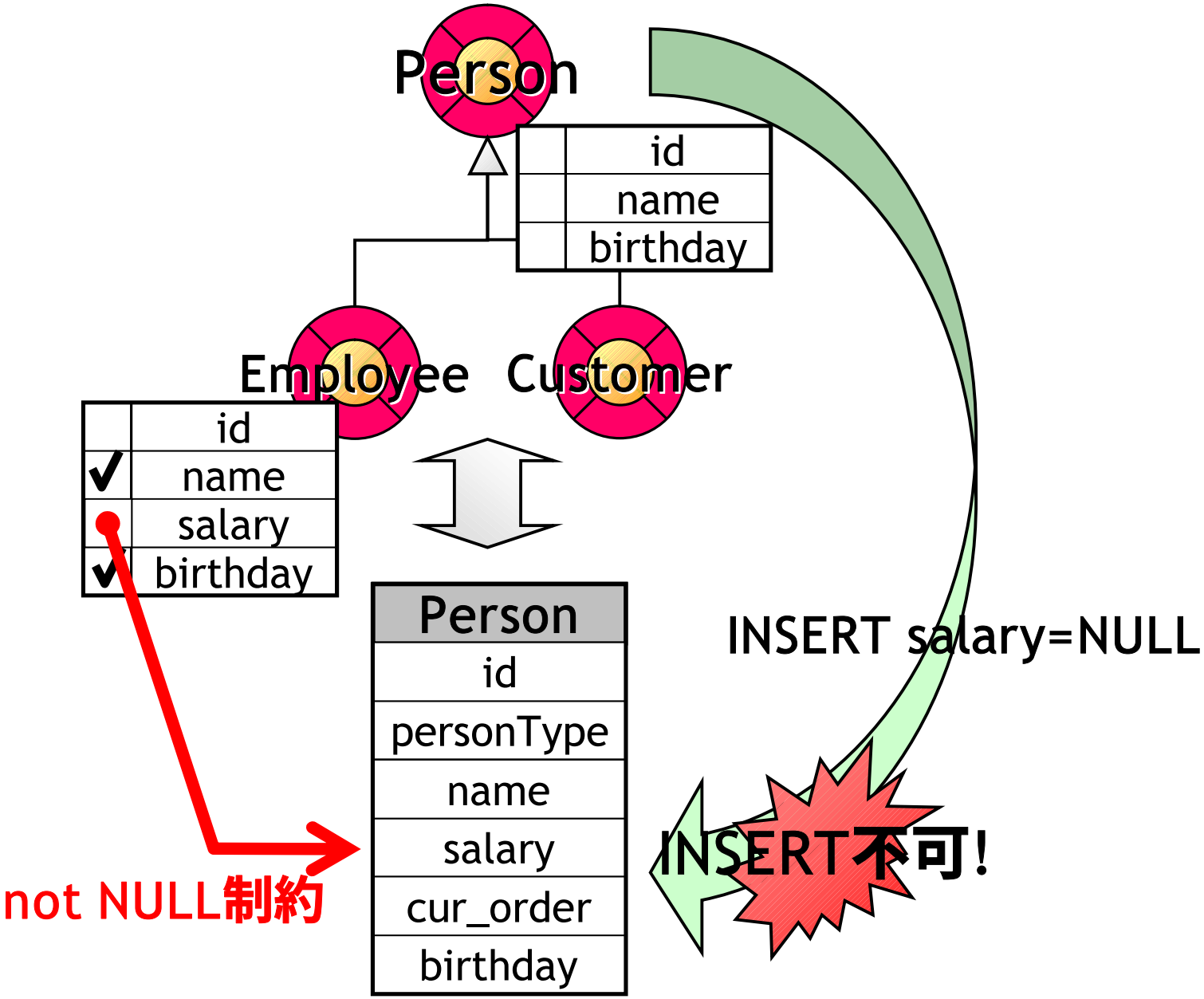
Inheritance and Primary Keys

- keyの一意性に関する制約
  - RDB: テーブル内で一意
  - EOF: 継承ツリー内で一意
- primary key generator
  - 継承ツリー内で一意なkeyを生成する
  - 具体的には????

Inheritance and the Overly Required Attribute

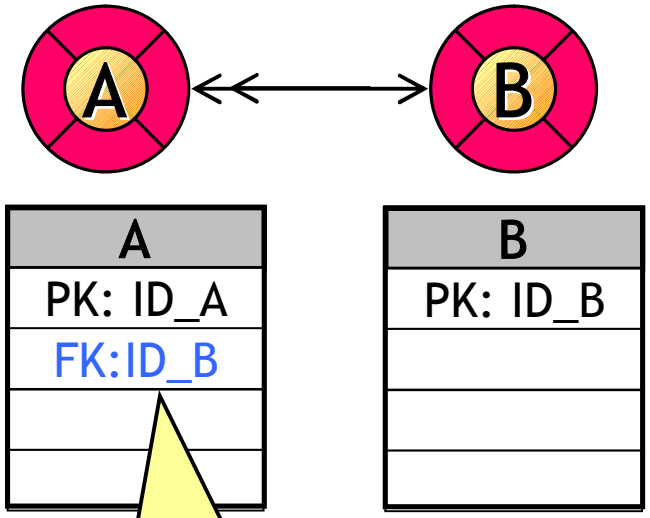
- **継承を使う場合の注意点**
  - **必須(=Required)属性に関連して、誤ったSQLが生成される場合がある。**
  - **INSERTできないテーブルが生成される恐れあり！**

ex) INSERTできないテーブル



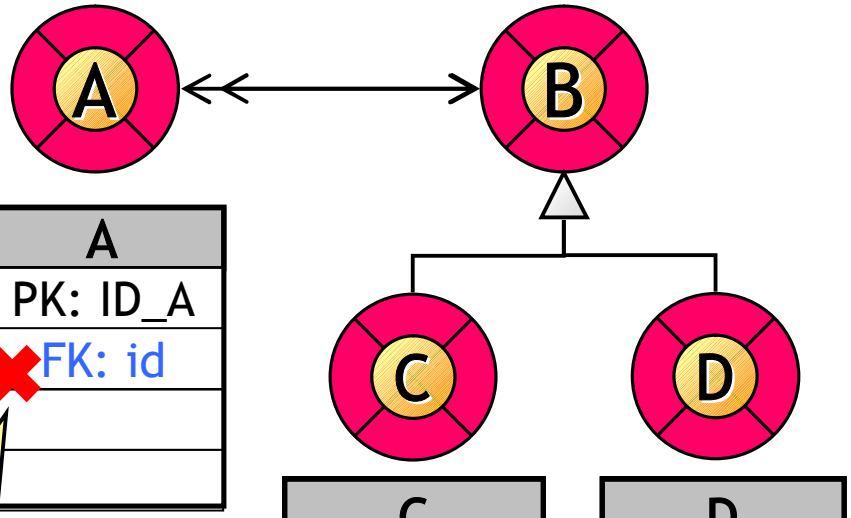
Inheritance and the Impossible Constraint

◆ 継承ナシ



外部キー制約を用いて、「親表(=B)に必ず値が存在しなくてはならない」制約を実装できる

◆ 継承アリ



親表がC or Dとなるため、単なる外部キー制約では制約を実装できない

パス

- **あまりないケースと思われるのでパス**
  - Inheritance and Multiple Models
  - Inheritance, Multiple Models, and SQL Generation

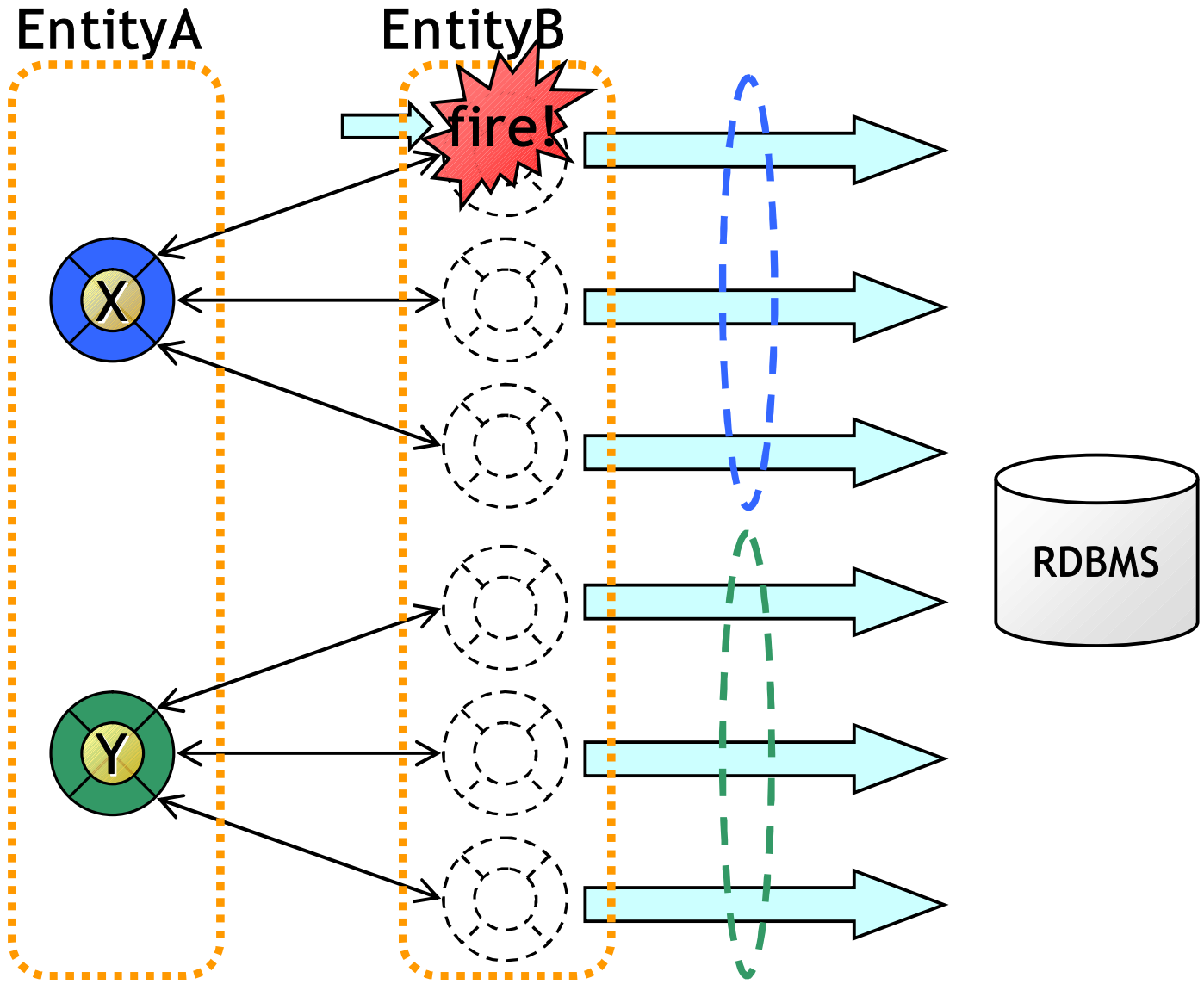
Relationships

- Optimizing Relationships with Batch Faulting
- One-Sided Relationships
- AddObjectsToBothSidesOfRelationshiopWithKey()

Optimizing Relationships with Batch Faulting

- Faultingとフェッチ回数
  - eo1ヶあたりフェッチ回数1回
  - toManyリレーションシップをiterateすると、フェッチがガンガン飛ぶ
  - Batch Faultingを活用することで、フェッチ回数を削減できる
- Batch FaultingでフェッチされるEOの個数を制限する方法
  - リレーションシップに対して最大数を設定する
  - エンティティに対して最大数を設定する

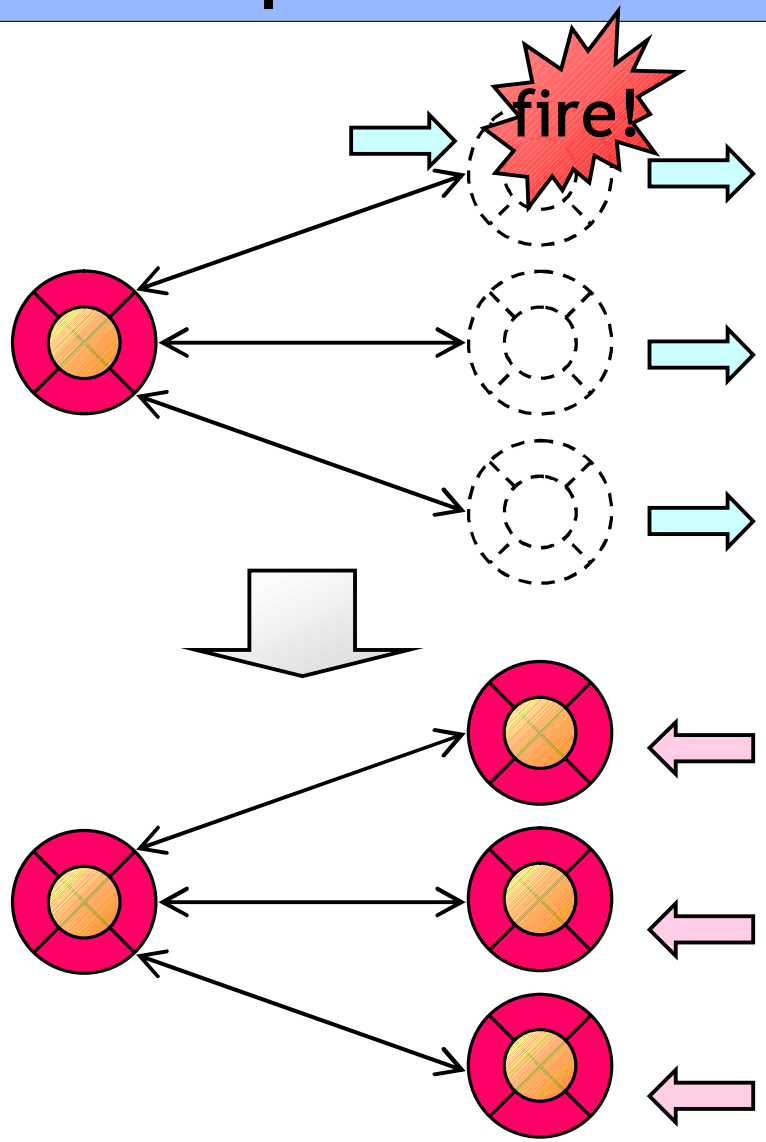
Batch Faulting Concepts



SELECT … FROM EntityB as B WHERE B.FK= x OR B.FK = y

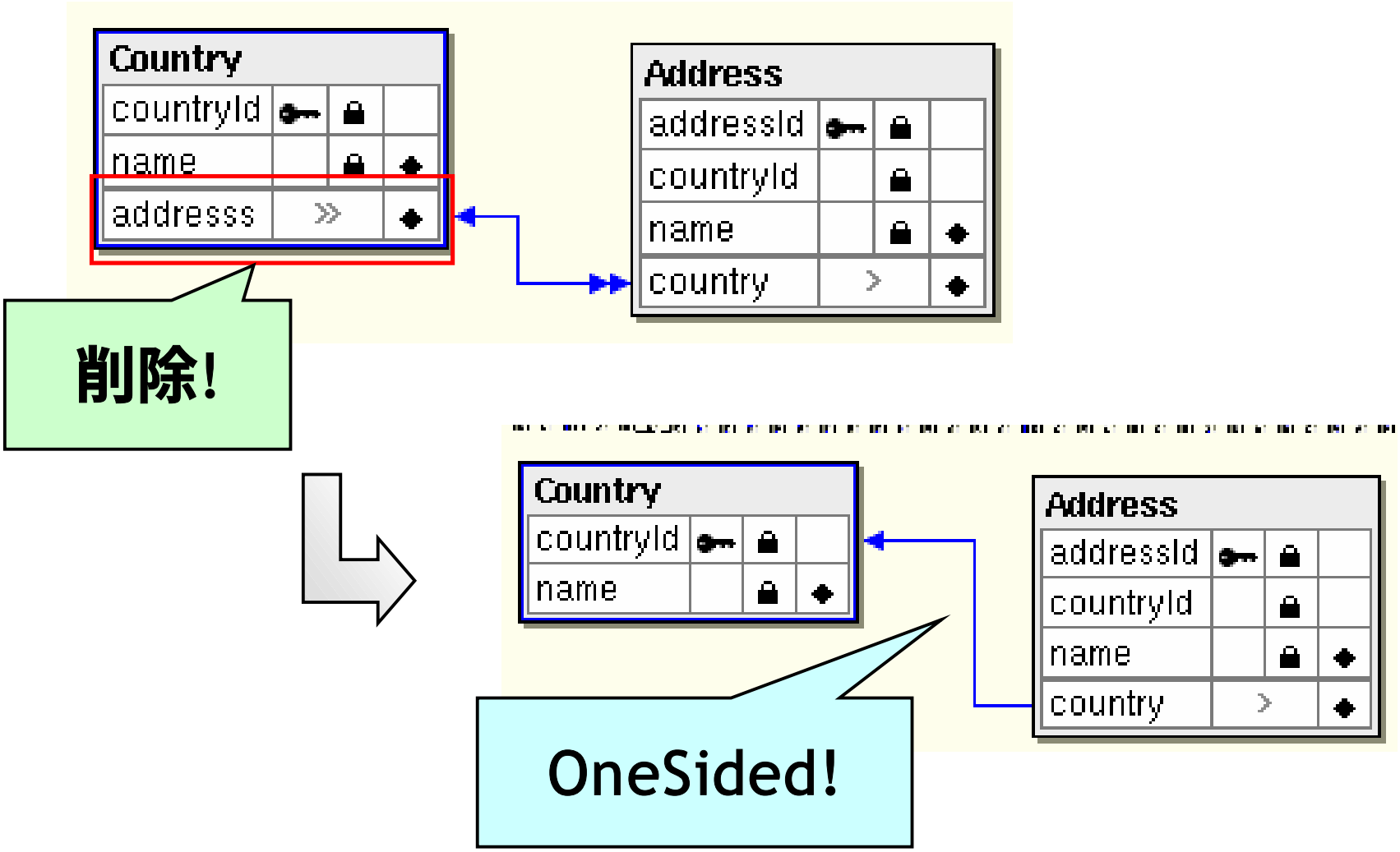
One-Sided Relationships

- **通常の場合**(Both-Sided Relationship)
  - to-manyリレーションシップがfireされると、関連するEOがすべてフェッチされる
  - to-manyリレーションシップに関連付けられたEOの数が**極端に多い場合**は、これが問題となる。



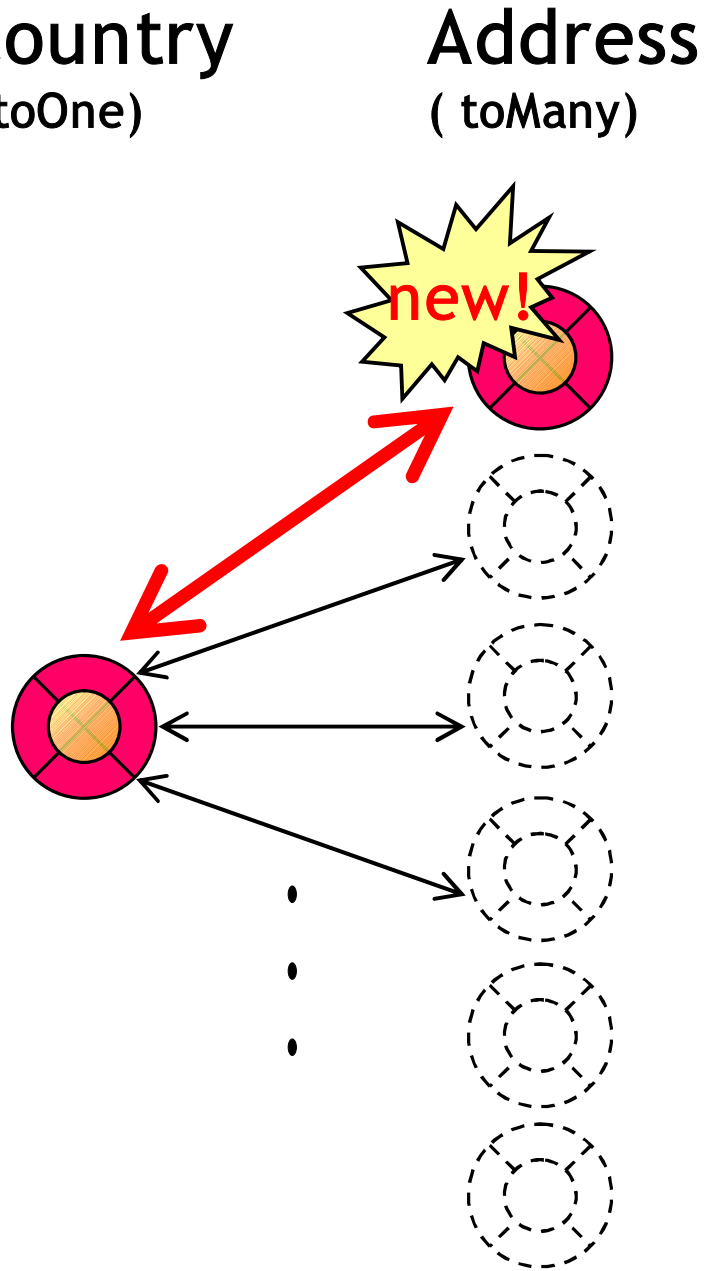
One-Sided Relationship

- **対処** → One-Sided Relationship



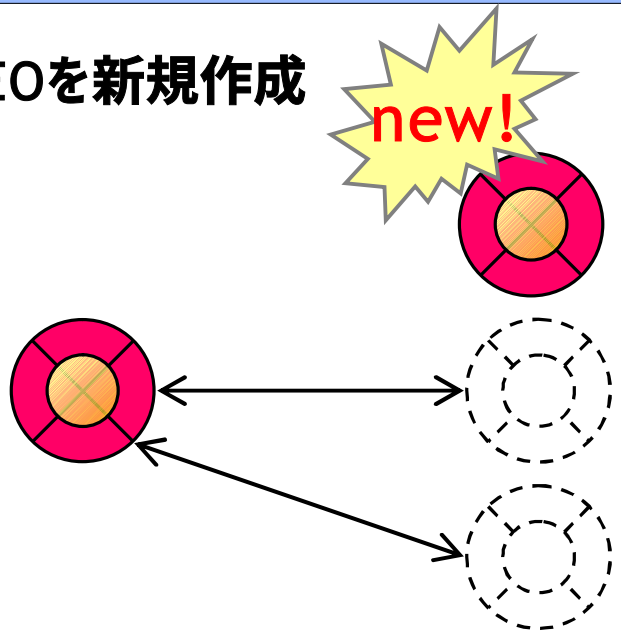
振る舞いの比較 ~事例~

- toMany側にEOを追加した時の振る舞いを比較する
- モデル例
  - Both-Sided Relationshipの場合
  - One-Sided Relationshipの場合

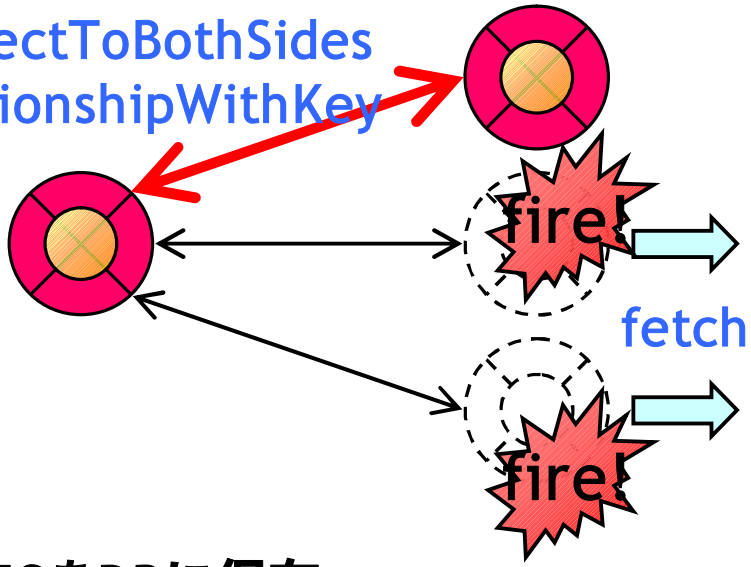


Both-Sided Relationshipの場合

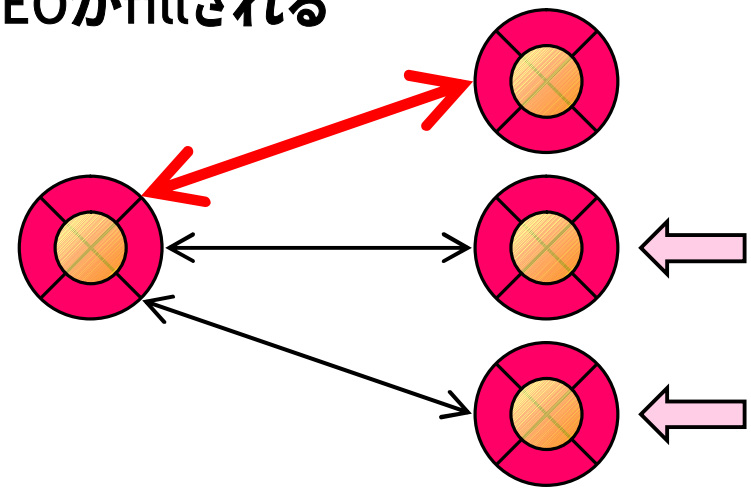
1) EOを新規作成



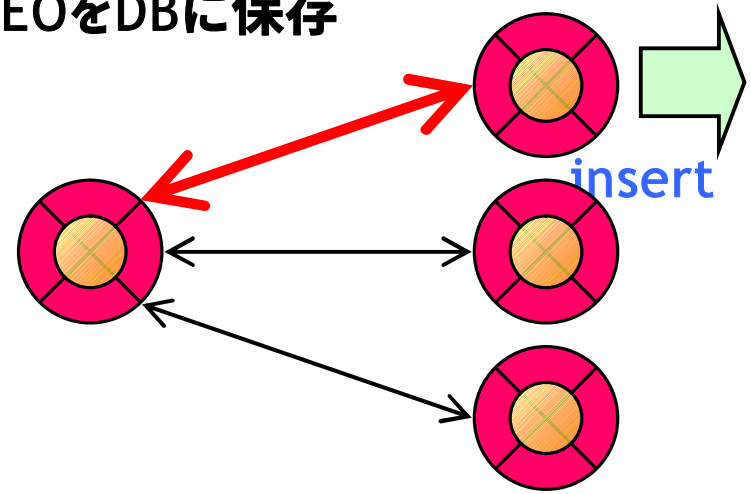
2) リレーションシップを設定



3) EOがfillされる

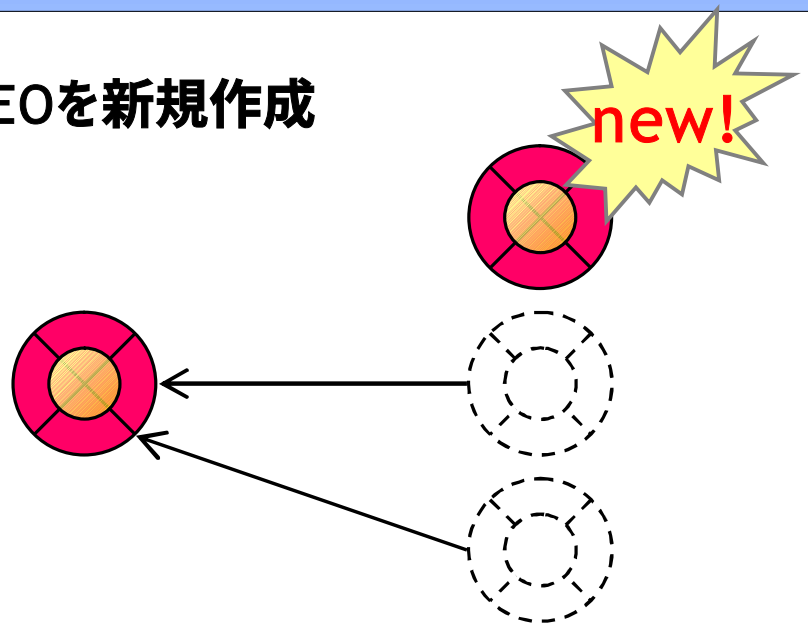


4) EOをDBに保存

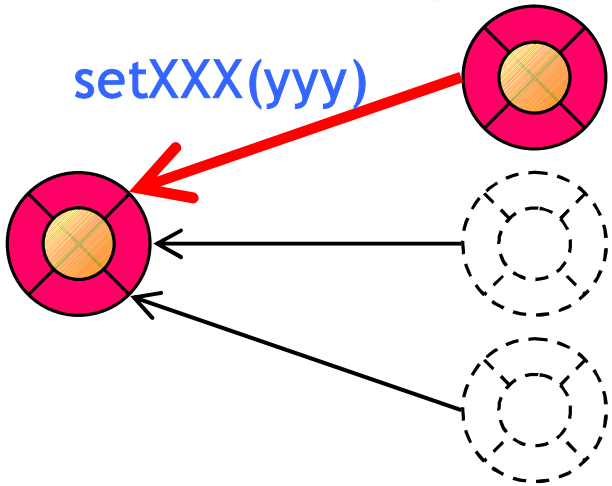


One-Sided Relationshipの場合

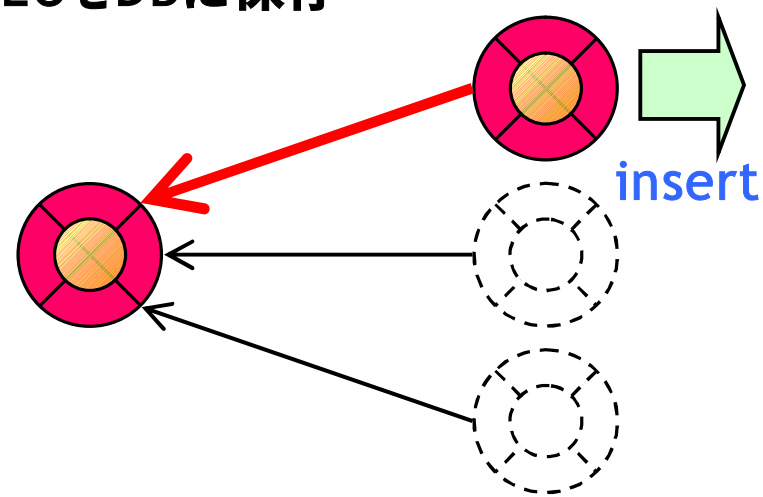
1) EOを新規作成



2) リレーションシップ設定

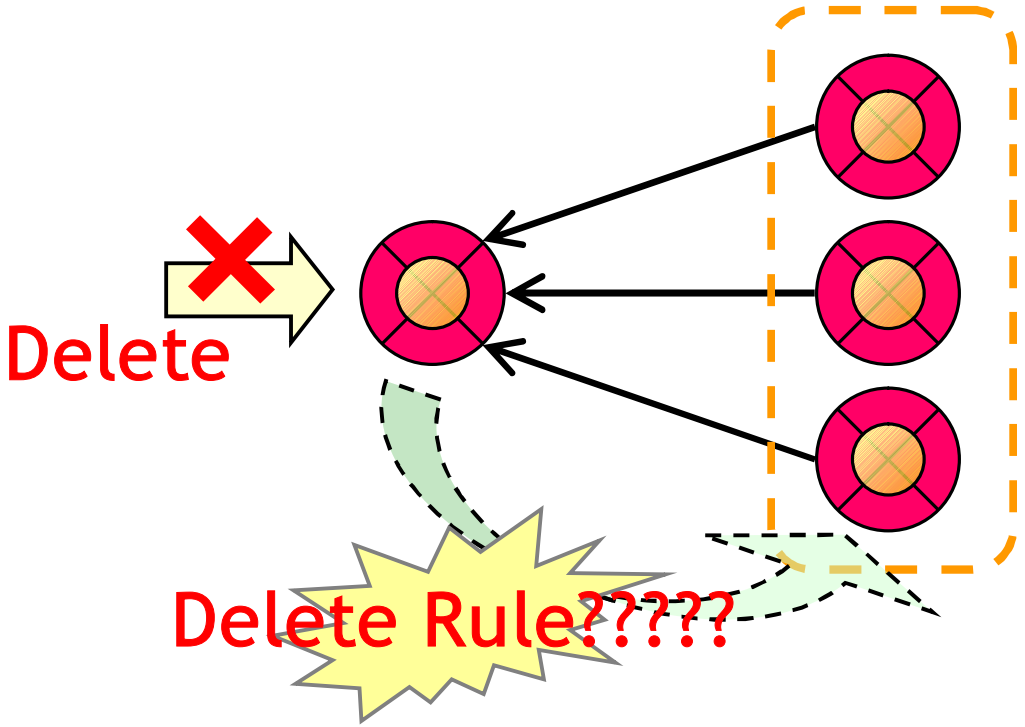


3) EOをDBに保存



Delete Rules and One-Sided Relationships

- Relationshipが片方向
  - → もう逆方向にはRelationshipが存在しない
  - → Delete Ruleが機能しない
  - → 自前で用意する必要がある



AddObjectsToBothSidesOfRelationshipWithKey()

- `a.setB(b)`
- `b.setA(a)`
  - **このリレーションシップ設定法は、**Both-Sided Relationship**の場合のみうまくゆく**
- `a.addObjectsBothSidesOfRelationshipWithKey(b, “to_b”)`
  - **これは、**Both-Sided / One-Sided Relationship **のどちらの場合でもうまくゆく**

Attribute Value Types

- NonInteger Numbers
- Column: Value Classes and Valuve Types
- Custom Types
- Column: Using NSArray and NSDictionary as Attributes
- Handling LOB Data Efficiently

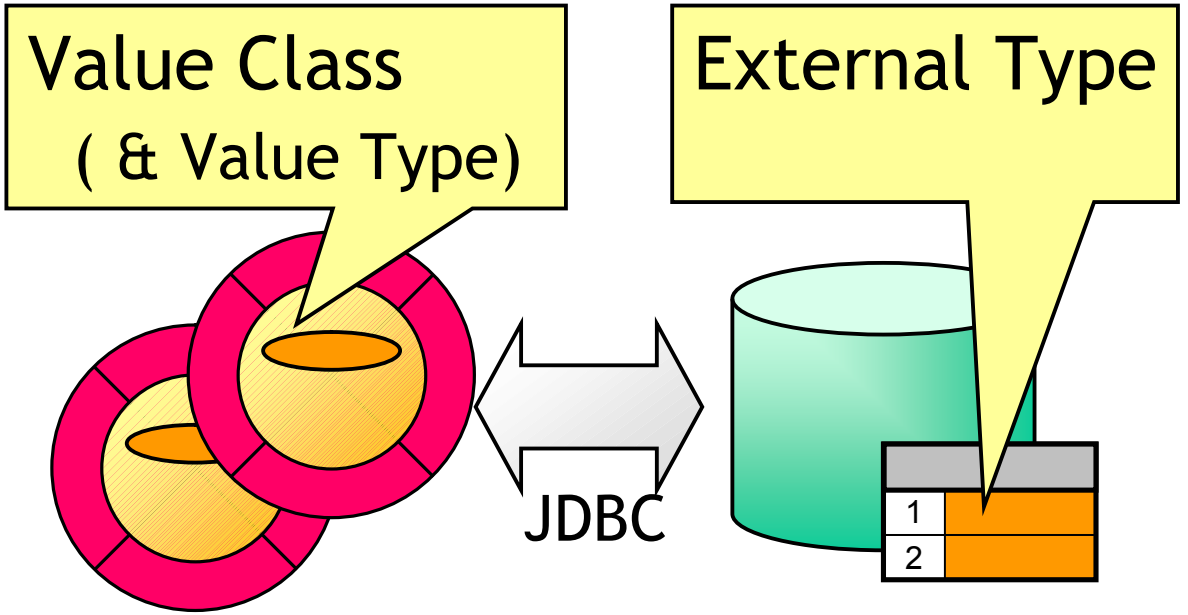
NonInteger Numbers

- お金を扱う場合、BigDecimalを使いなさい。
  - 小数を含む
  - 「丸め誤差」が許されない
- BigDecimalの欠点
  - floatの40倍遅い

- BigDecimalの代替案
  - 桁上げしてlongを使う
    - Ex) $127.33 を $12,733として扱う
    - オーバーフロー、割り算、掛け算には気をつけよ！

Column: Value Classes and Value Types

- Concepts



- Value ClassとValue Typeの組み合わせ関係

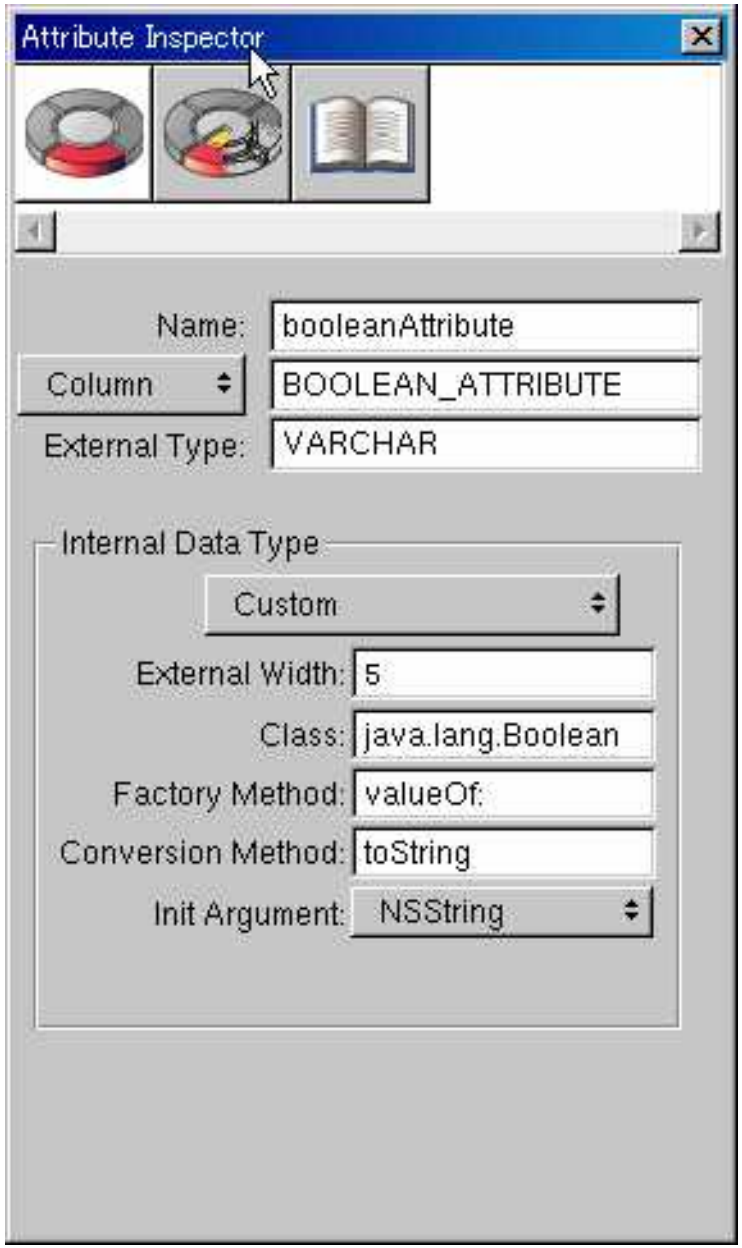
Value Class	Value Type	
	M/O	役割
Number	M	Numberのサブクラスを指定
String	O	文字列の変換ルールを指定
NSTimeStamp	O	Date or Time or TimeStamp or ...を指定

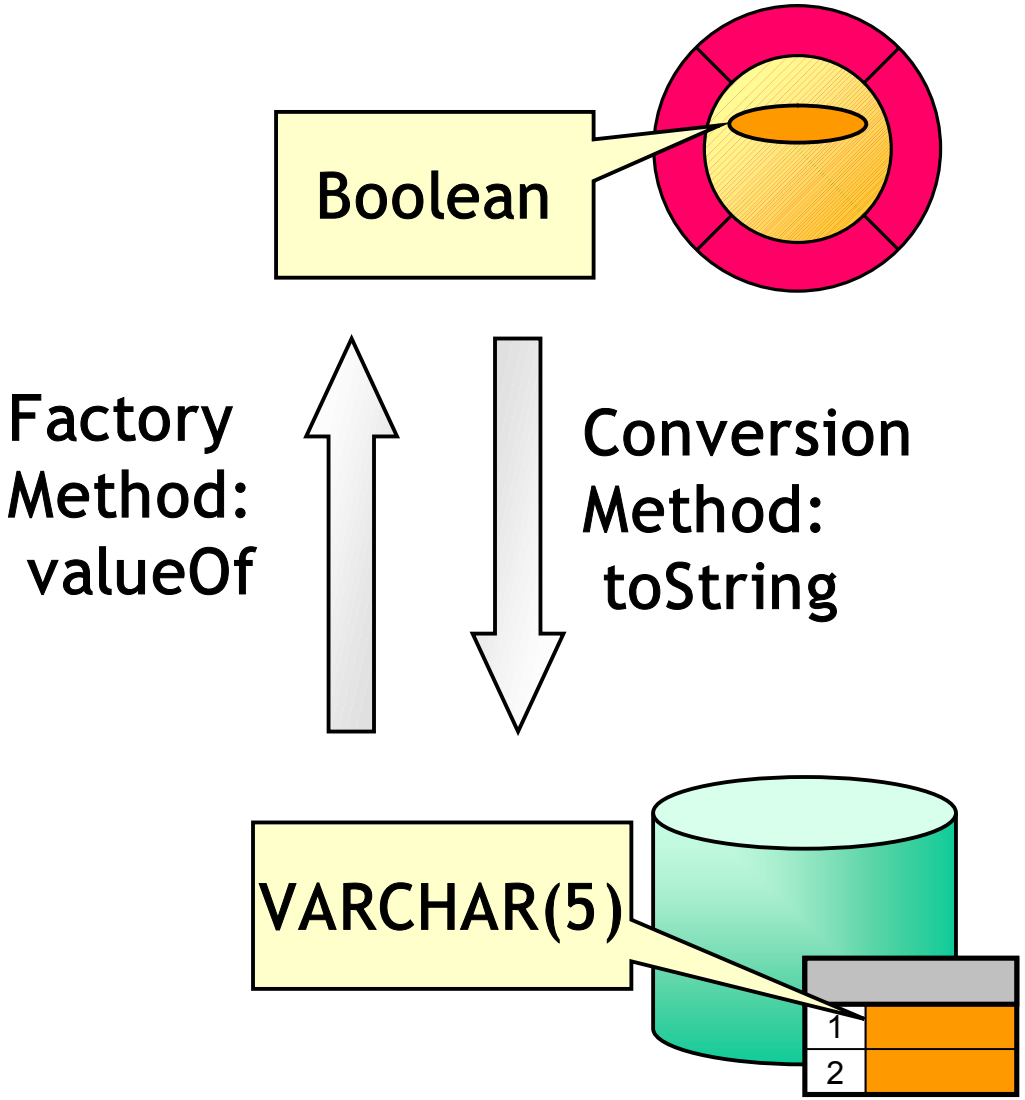
Custom Types

- **アトリビュートとして利用可能なjavaクラス**
  - **定義済みType（Number, String, ...)**
  - **カスタムtype（コレクションクラス, Boolean, ...)**
- **カスタムtypeの要件**

要実装メソッド	説明
clone() メソッド	Faulting時、osc→ec or ec→ecの間でeoをコピーする際に使用する
isEqual() メソッド	Optimisitcロックの変更有無チェックに使用する
Factory Method	DBからフェッチ時、RDBのカラム値からカスタムTypeのインスタンスを生成するために使用する
Conversion Method	DB保存時、カスタムTypeからRDBのカラム値に変換するために使用する

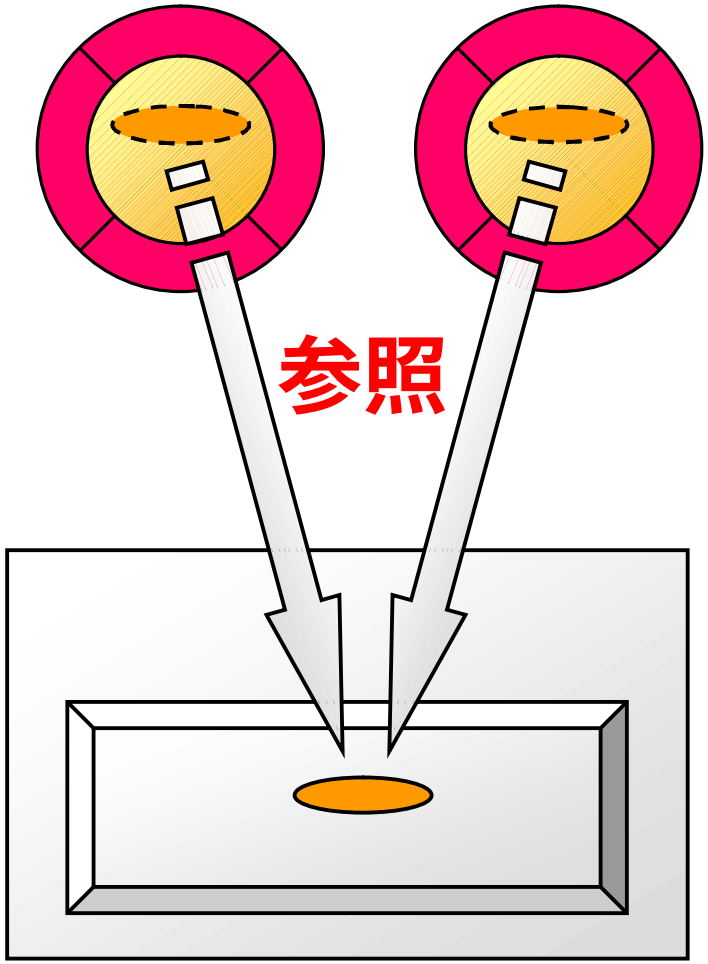
An EOF Boolean





No Beach for You!

- Custom Typeとして定義したクラスは、Immutable(不変)でなければならない
  - ちなみに、定義済みTypeは全てimmutable
- 理由
  - 各eoが、Snapshot内のオブジェクトを参照 (≠コピー) しているため
  - 仮に変更できると、その変更が伝播して問題となる



Column: Using NSArray and NSDictionary as Attributes

- Custom TypeとしてNSArray, NSDictionaryを使いたい
- Custom Typeであるためには、適切なFactory Methodが必要
- しかし、NSArray, NSDictionaryにはない

→ サブクラス化して、FactoryMethodを実装する。

Handling LOB Data Efficiently

- LOBの問題点
  - 読み込み時メモリを喰う
- 対処
  1. 別テーブルに切り出す
     - T字形ERで言うところの、みなしエンティティ?
     - 詳細は書籍参照
  2. RDBからファイルシステムに追い出す
     - 問題:トランザクションのご利益に預かれなくなる。

Prototypes

- JDBC Prototypes (Old vs. New Prototypes)
- Strategies for Mapping Prototypes
- One Last Consideration: The Connection Dictionary

Prototype?

- 正体
  - アトリビュートの設定項目のテンプレート
- ご利益
  - 設定項目の使いまわし、一貫性の確保
  - RDBMS依存性の吸収 (External Typeの差異)
- 実装・実現方法
  - “EOJDBCPrototypes”なる名称のエンティティを作成し、各Prototypeをアトリビュートとして定義すればよい
    - 実は、エンティティ名称の正式な定義は”EO<EOAdaptor name>Prototypes”である (後述)

JDBC Prototypes (Old)

- 実はPrototype定義用エンティティ名称の正確な定義は・・・

  EO<Adaptor name>Prototypes

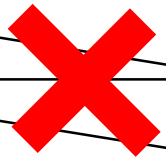
- 昔 (Objective-C時代?) を振り返る
  - RDBMS毎にEOAdaptorクラスが存在
    - 各EOAdaptorクラスでRDBMS依存性を吸収
  - RBBMS毎にPrototype指定用エンティティの名称が異なる

RDBMS	EOAdaptorクラスの名称	Prototype定義用エンティティ名称
Oracle	EOOracleAdaptor	EOOraclePrototypes
OpenBase	EOOpenbaseAdaptor	EOOpenBasePrototypes

JDBC Prototypes (New)

- **現在は、RDBMS毎に異なるEOAdaptorを使用しない!**
  - **→ 全てのRDBMSに対してEOJDBCAdaptorを使用する。**
  - **→ 全てのRDBMSに対してPrototype定義用エンティティの名称は同じ、”EOJDBCProtypes” となる。**

RDBMS	EOAdaptorクラスの名称	Prototype定義用エンティティ名称
Oracle	EOOracleAdaptor	EOOraclePrototypes
OpenBase	EOOpenbaseAdaptor	EOOpenBasePrototypes
全てのRDBMS (OpenBase, Oracle, PostgreSQL,...)	EOJDBCAdaptor	EOJDBCPrototypes



使用しない

全てのRDBMSに同じPrototypeが使用されるのなら、RDBMSの依存性を吸収できないじゃないですか？

Strategies for Mapping Prototypes

#	方式概要	RDBMS依存性の吸収	Prototypeの使いまわし
1)	既存のEOModelにEOJDBC...エンティティを追加する	対応不可	対応不可
2)	EOJDBC... エンティティを持つEOModelをフレームワークに外出しする	対応不可	フレームワークの共有で対応
3)	2)のフレームワークをRDBMS毎に作成	フレームワーク差し替えで対応 (要リビルド)	
4)	プログラムでPrototypeの差し替えを実行 (詳細は書籍を参照・・・)	プログラムで対応	

One Last Consideration: The Connection Dictionary

- Connection Dictionary**変更の必要性**
  - **本番環境とテスト用環境で**RDB**の接続情報**（DB**サーバホスト名、パスワード、**etc...）**を変える**
  - **《**WR:**セキュリティに配慮し、**EOModel**にはわざとパスワードを書かず、後で動的に設定する。とか？？？》**
- Connection Dictionary**の正体**
  - plist
- Connection Dictionary**変更の方法**
  - `aModel.setConnectionDictionary(conDic)` **するだけ**
  - **でも、適切なタイミングで変更するのは難しい。**
  - **⇒** PracticalUtilitis.framework**の** EOModelConnector **クラスを使いなさい。**

Debugging JDBC Connection Problem

- Understanding the Source of the Problem
- Finding and Fixing the Problem

- **典型的なトラブル例**
  - EOModelerからデータベースのデータが見え、コンパイルが上手く行き、ドライバも存在するが、接続が拒否される

Understanding the Source of the Problem

- EOFとJDBCコネクションに関するいくつかの事柄を理解する必要がある。

- EOModeler Muddles　(EOModelerの混乱)
  - ありがちな現象：「EOModelerからデータベースが見えるが、アプリケーションからは見えない」
  - EOModelerの特徴
    - Javaアプリケーションではなく、Objective-Cアプリケーションである。
    - かなり古い
    - Java Bridgeを使用して、JDBC接続する
      - Windows: JDK1.1/JDBC 1.0API を使用する
      - OS X: JDK1.3以降 / JDBC 2.0 API (よって、ちょっと楽)
  - 以後、Windows環境でのトラブルを中心に解説する。

Windows版EOModelerとJava実行環境

EOModeler

JRE1.1

クラスファイル
JARファイル…

JRE1.1用
JDBCドライバ

JavaConfig.plist

CLASSPATH等の
Java関連設定が記載

基本的に
別モノ!!!

WebObjects
アプリケーション

JRE1.3/1.4

クラスファイル
JARファイル…

JRE1.3/1.4用
JDBCドライバ

環境変数など

実行環境の違いから導かれる事柄

- 「EOModelerで接続に成功!」とは・・・
  - アプリケーションの接続可能性とは**ほとんど無関係**
  - 唯一、検証できたことは・・・
    - JDBC URL文字列 (ex) “`jdbc:OpenBase://…`”) の正当性
    - DBMS接続のユーザー情報の正当性
- 「WOAで接続OK、EOModelerで接続NG」の場合

  → 以下をチェックせよ
    - JavaConfig.plistのCLASSPATH設定
    - JDBCドライバ置き場
      - Windows：$NEXT¥Library¥Java
      - OS X：/System/Library/Java

Plug-In Pickiness

- Plug-Inの必要性
  - 現在：どんなDBMSにも同じAdaptor (=JavaJDBCAdaptor) を使用する
  - しかし、DBMSはそれぞれ細かい差異を持つ
  - ⇒ Plug-Inで差異を吸収する
- Plug-In クラスの実装
  - com.webobjects.jdbcadaptor.JDBCPlugInのサブクラス
- Plug-In クラスの選択
  - 1) JDBC URLから類推
    - 詳細はhttp://developer.apple.com/ja/technotes/tn2027.html
  - 2) Connection DictionaryのPlugIn指定
- Plug-In JARの指定
  - デフォルトでサポートされているDBMS ⇒ 指定不要
    - Plug-Inはcom.webobjects.jdbcadaptorパッケージに存在
  - サポート外のDBMSでPlug-Inが不要 ⇒ 指定不要
    (JDBCPluginクラスが使用される)
  - サポート外のDBMSでPlug-Inが必要 ⇒ CLASSPATH指定必要

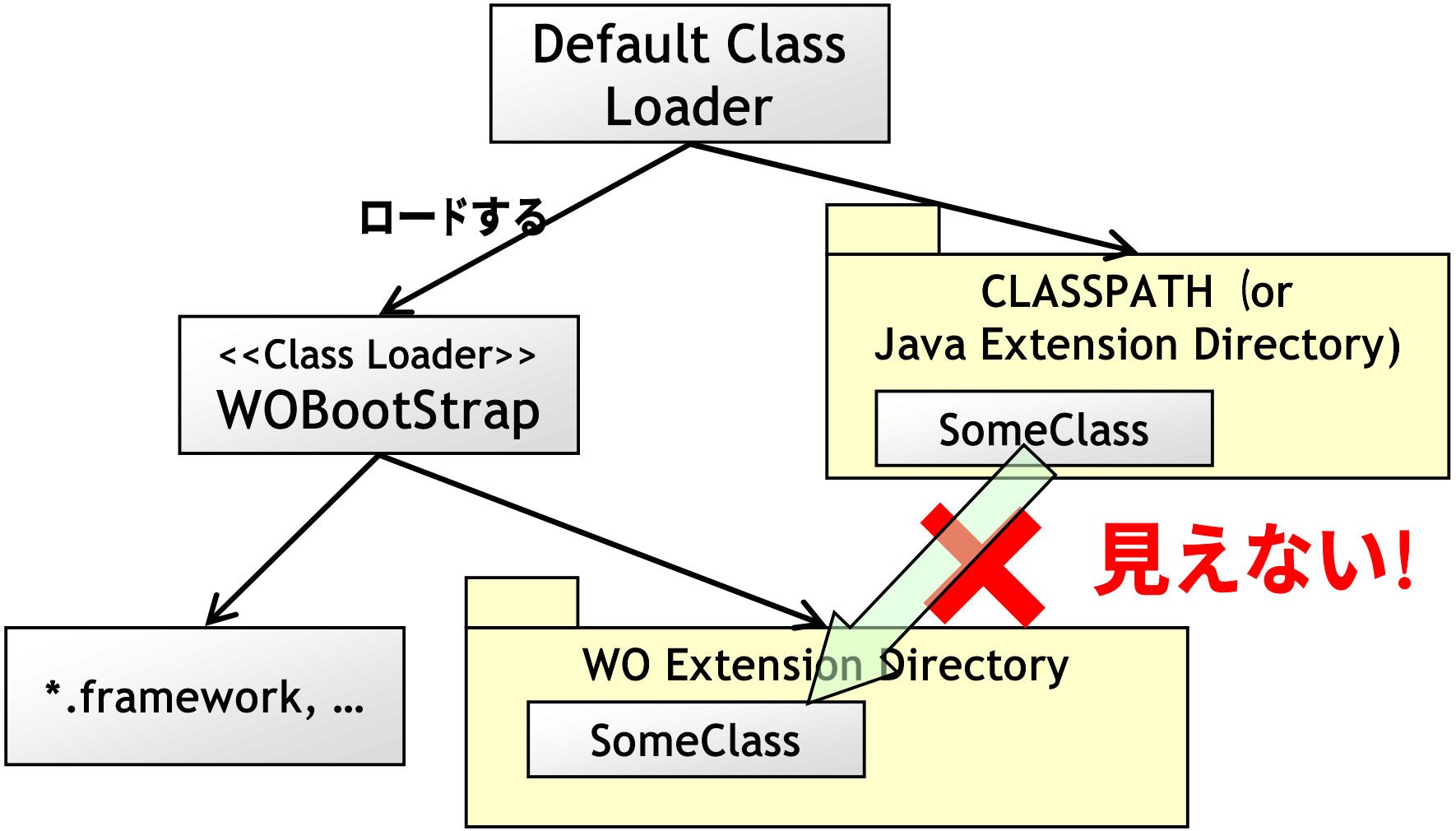
Extension Exertions

- Java拡張メカニズムの概要
  - あるディレクトリ（=拡張ディレクトリ）に置かれたJARsを、VM初期化時に自動でロードする仕組み
    - 対応関係：1つのJVMにつき1つの拡張ディレクトリ
    - ex)C:¥Program Files¥Java¥j2re1.4.2_05¥lib¥ext
- Java拡張メカニズムの問題点
  - 拡張ディレクトリの場所がわかりにくい
    - Windowsだと、たくさんのJRE(　JVM)がインストールされがちなので、かなりワケわからん。
    - OS Xだと・・・《すみませんわからないので省略》
  - WR注) 拡張ディレクトリの場所は、System.getProperty("java.ext.dirs");で取得できる。

Another Extensions Directory

- WebObjects特有の拡張ディレクトリ
  - Windows, Solaris :
    `$NEXT_ROOT/Local/WebObjects/Extensions`
  - OS X : `/Library/WebObjects/Extensions`
- Caution:
  - Plug-Inクラスは**Java拡張ディレクトリに置いてはいけない**
  - **クラスローダについていろいろ書いてありますが、よくわかりません・・・**

こういうこと?

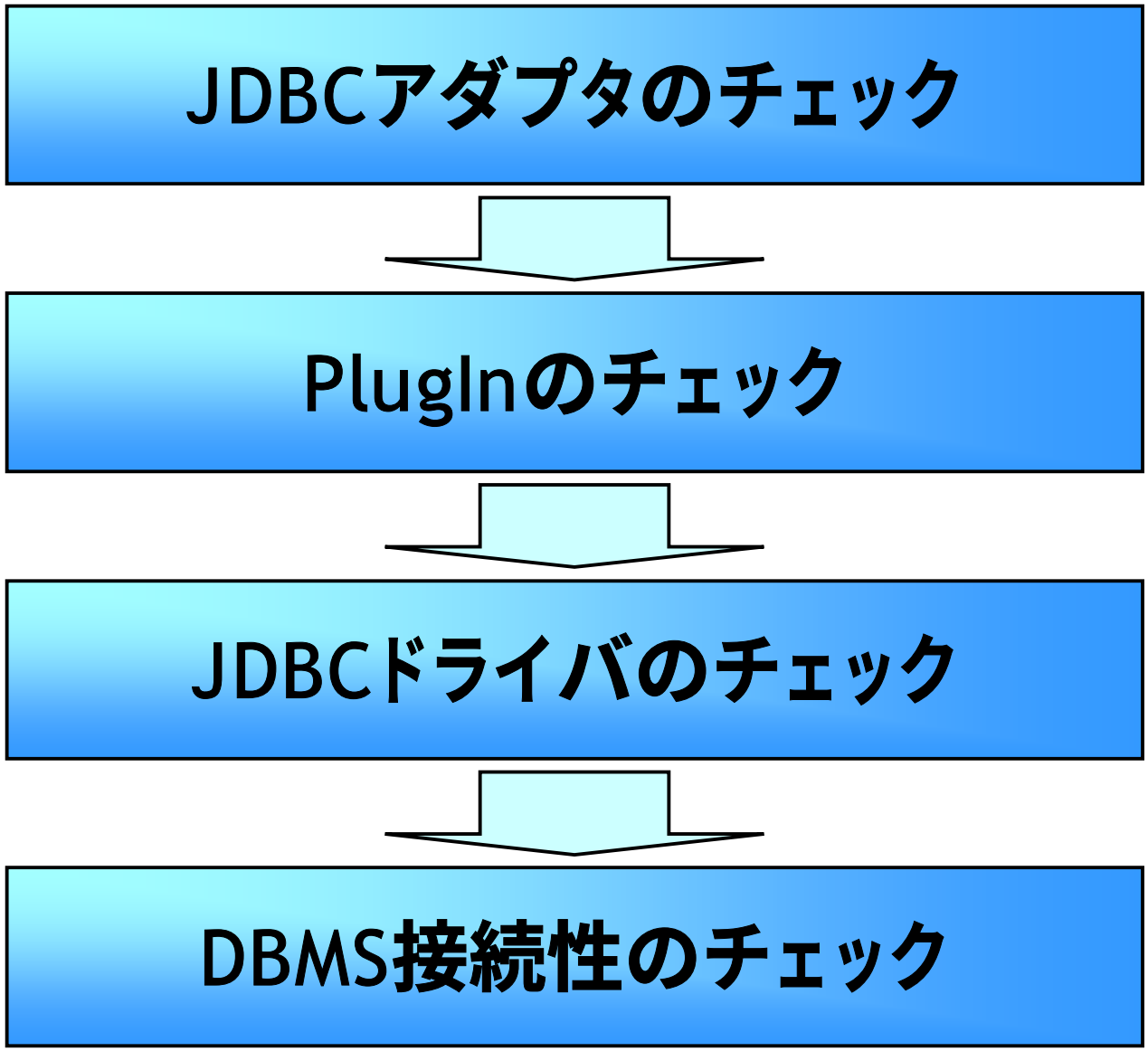


よくわかりません・・・

Finding and Fixing the Problem

- In the Beginning There was a JDBC Adaptor
- A Plug-in Is Born
- The Driver, Please
- Operator, Can You Connect Me?
- The EOFJDBCConnectionAnalyzer Class

トラブルシュートの順序



In the Beginning There was a JDBC Adaptor

- JDBCアダプタのチェック
  - `JDBCAdaptor targetAdaptor = (JDBCAdaptor) EOAdaptor.adaptorWithName("JDBC")`
  - Nullの場合 → 失敗
    - JavaJDBCAdaptor.frameworkがインクルードされていない！
  - JDBCAdapterの場合 → 成功
    - Plug-Inのチェックへ

A Plug-in Is Born

- JDBC Plug-Inのチェック
  - JDBCアダプタにConnection Dictionaryをセットして、Plug-Inが作成されることをチェックする。
    - `jdbcAdaptor.setConnectionDictionary(conDic);`
      `JDBCPlugIn jdbcPlugIn = jdbcAdaptor.plugIn();`
  - jdbcPlugInのクラス名がcom.webobjects.jdbcadaptor.JDBCPlugInの場合
    → DBMS用のカスタムPlugInが使用されない
  - JDBCPlugInのサブクラスの場合
    → 各DBMS用のカスタムPlugInが使用される
    - ex) com.webobjects.jdbcadaptor.OpenBasePlugIn, com.webobjects.jdbcadaptor.FrontbasePlugIn, etc...

The Driver, Please

- JDBCドライバのチェック
  - PlugInがセットされたJDBCAdaptorからドライバのクラス名を取得し、インスタンス化
    - `Class jdbcDriver = Class.forName(jdbcAdaptor.driverName())`
    - 注意) PlugInが適切にセットされていない場合、driverName()でNullPointerExceptionが投げられる。
  - チェックポイント
    - クラス名が取得できているか?
    - クラスのロードがうまくいくか?
      - 失敗した場合、CLASSPATHかJRE拡張ディレクトリにJDBCドライバのJARファイルがあることをチェックせよ。

Operator, Can You Connect Me?

- DBMS接続性のチェック
  - `jdbcAdaptor.assertConnectionDictionaryIsValid()`
  - 接続に失敗すると例外が投げられる
    - `[2004-…] <…> A fatal exception occurred: Database SandBoxx not started on localhost.`
    - `[2004-…] <…> com.webobjects.jdbcadaptor.JDBCAdaptorException: Database SandBoxx not started on localhost.`
    - `at com.webobjects.jdbcadaptor.JDBCContext.connect(JDBCContext.java:244)`
  - ありがちな原因
    - DBサーバのホスト名誤り
    - DBサーバ〜APサーバ間のネットワーク到達性
    - データベースの名前誤り、ユーザ名誤り、パスワード誤り

The EOFJDBCConnectionAnalyzer Class

- PracticalUtilities.frameworkに含まれる接続トラブル解析用クラス
- 接続トラブルが生じた場合は、
  - 1) 上記frameworkをインクルード
  - 2) `new EOFJDBCConnectionAnalyzer("ModelName")`
  - すればよい (らしい・・・)

EOModeling with Eclipse

- EOModelerモデルはPB.projectのFRAMEWORKSEARCHパスを参照する
  - プロトタイプ定義を解決するため
  - 他のモデルのエンティティへのリレーションシップを解決するため
- 問題点
  - WOLipsはFRAMEWORKSEARCHパスをSystem Libraryディレクトリと Local Libraryディレクトリとする
- 対処
  - 上記ディレクトリにフレームワークを置く
  - 全てのモデルを1つのprojectに置く

おわり

- **次回は**Managing the Object Graph…
  - Page 61-97