J2EE展望
---EJB3.0によるEoDの世界---

稚内北星学園大学
丸山不二夫

EoDとしてのJ2EE

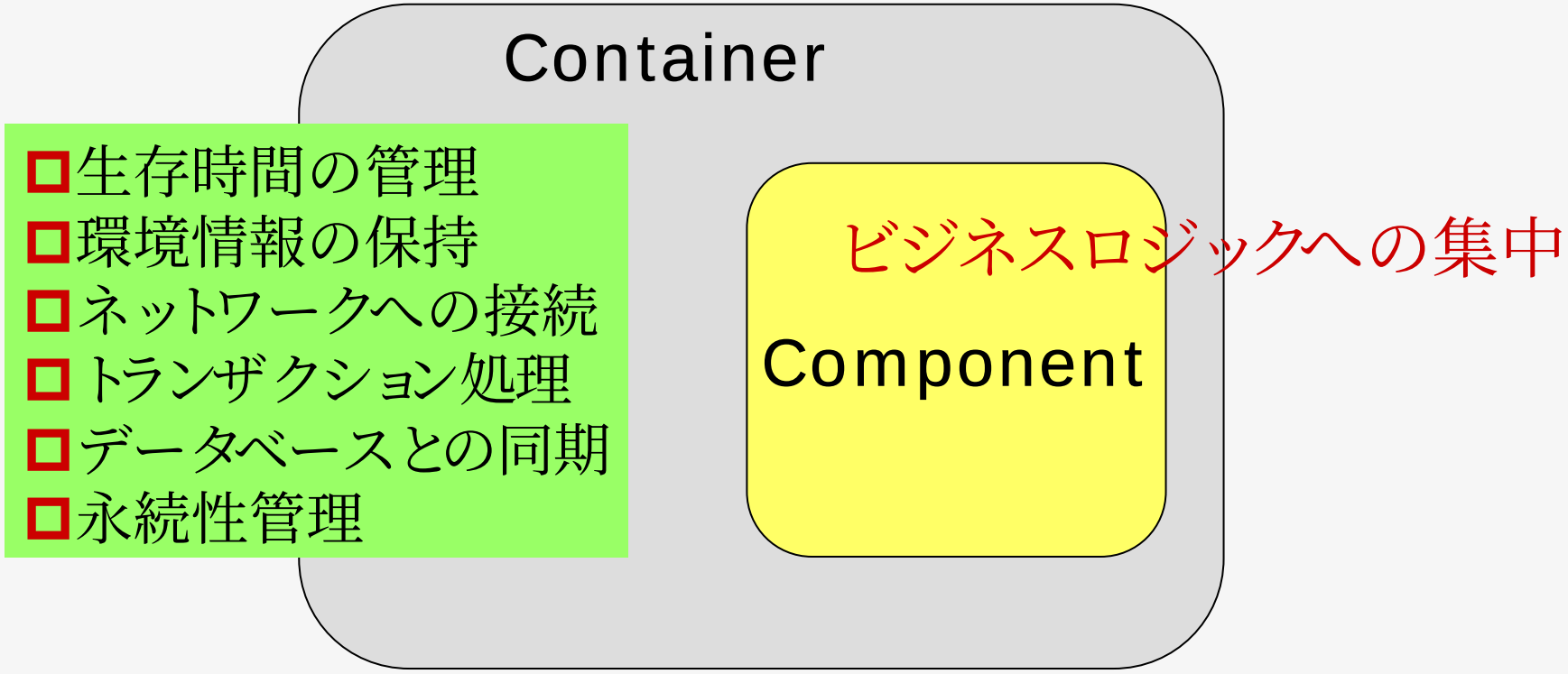
EoDとしてのJ2EE

コンテナ・コンポーネント・モデル

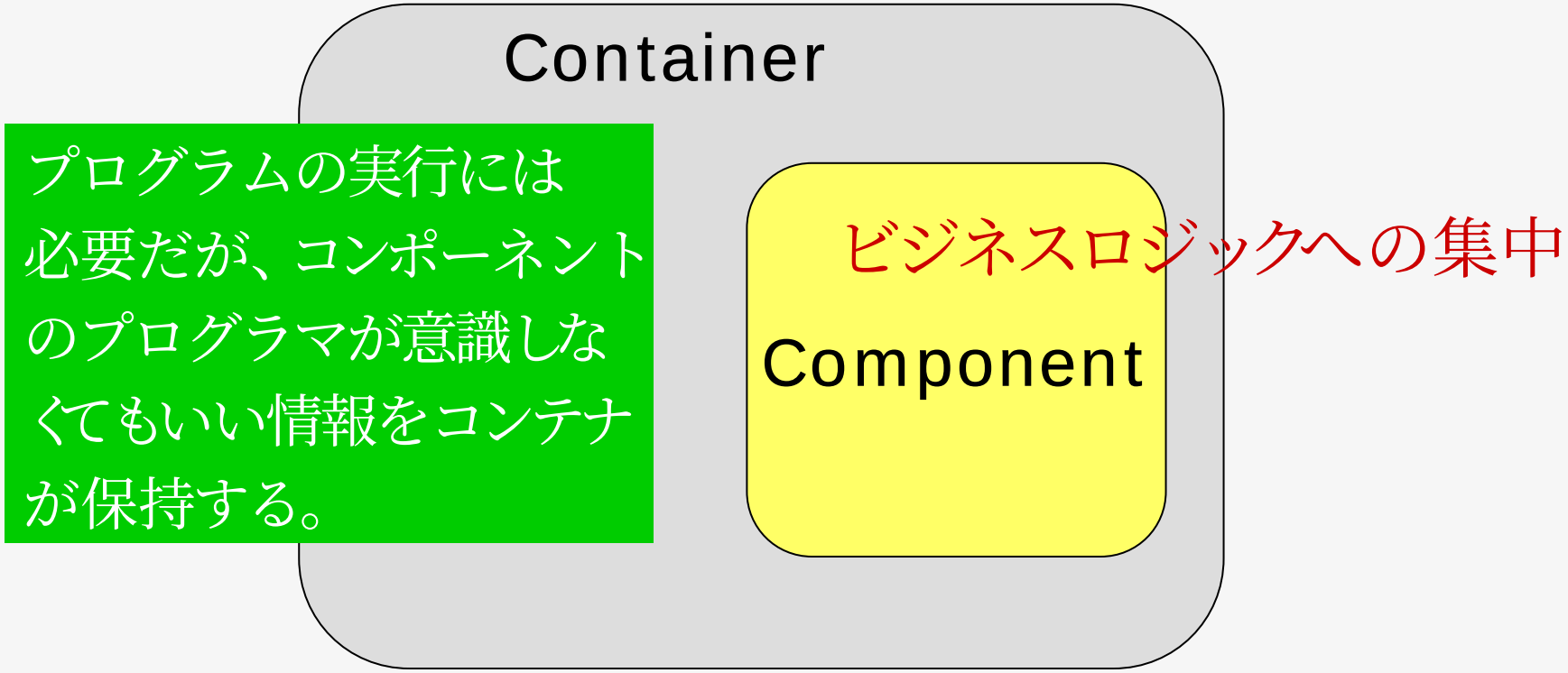
- ビジネスロジックへの集中
- 「三つ組」によるプログラミング
- コンポーネントとコンテナの役割分担

- コンポーネントの「再利用」
- コンポーネントの連携とデザイン・パターン
- Assemble/Deploy 開発者の役割分担

コンポーネントとコンテナの機能の分担

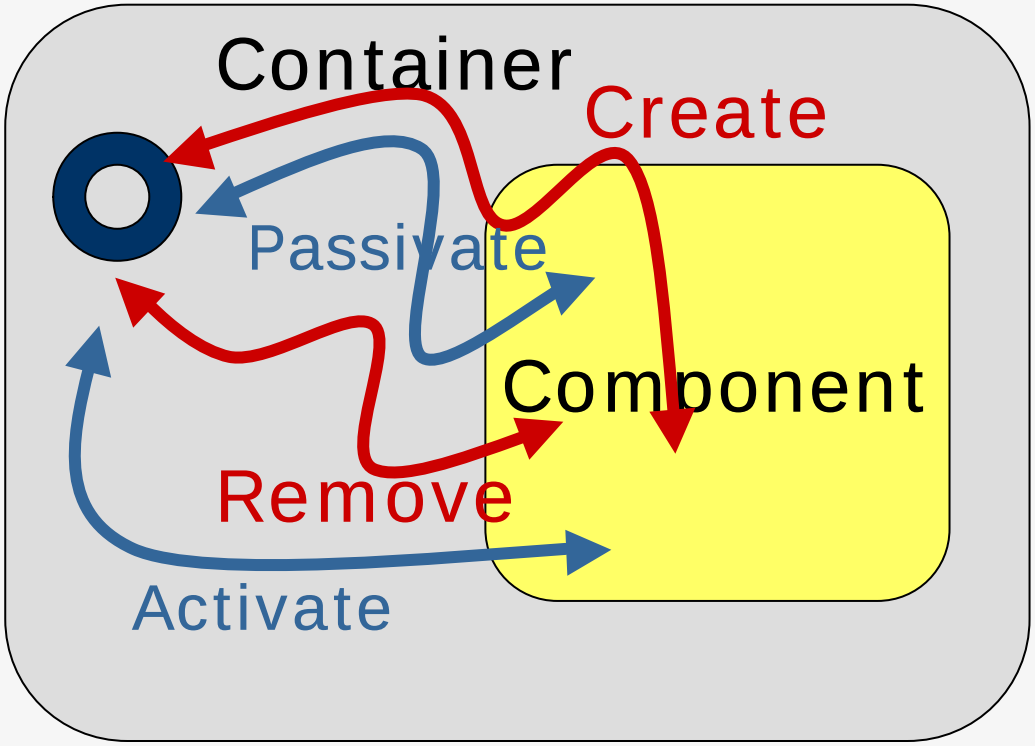


コンポーネントとコンテナの情報の分担



DIの手法、Annotationの手法の基礎

対象プログラムとしてのコンポーネント
メタプログラムとしてのコンテナ



対象プログラムの生存時間管理は、
コンテナの基本的な機能の一つ

「三つ組み」プログラミングの役割

別プログラムを生成する
Syntax Sugar

EJBの三つの定義ファイル

Remoteインターフェース： Bank.java

```
public interface Bank extends EJBObject {
    public void transferToSaving(double amount)
        throws RemoteException, InsufficientBalanceException;
    ……
}
```

コンポーネントの
インターフェース

Homeインターフェース： BankHome.java

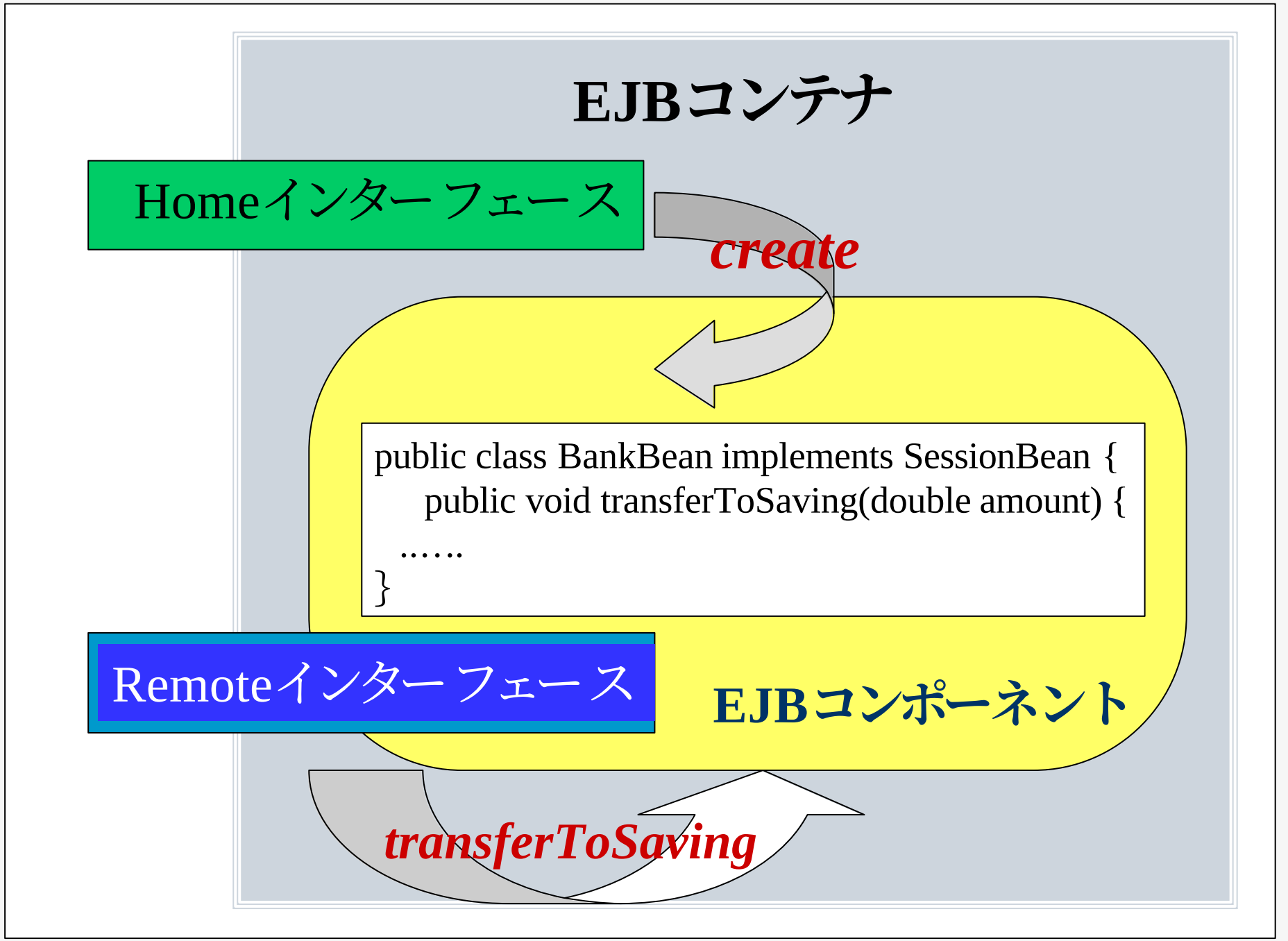
```
public interface BankHome extends EJBHome {
    public Bank create(String id)
        throws RemoteException, CreateException;
}
```

コンテナの
インターフェース

EJBクラス： BankBean.java

```
public class BankBean implements SessionBean {
    public void transferToSaving(double amount) {
    ……
}
```

コンポーネントの
実装


EJBコンテナ
Homeインターフェース
create
public class BankBean implements SessionBean {
public void transferToSaving(double amount) {
……
}
Remoteインターフェース
EJBコンポーネント
transferToSaving

```
public String sayHello() throws RemoteException   {
        .............
      // 元のEJBクラスのインスタンス
       HelloEJB helloejb = (HelloEJB) ..... ;
       .............
      // メソッド呼び出しの前処理
       container.preInvoke(...);
      // 元のEJBクラスのメソッド呼び出し
      String s = helloejb.sayHello();
      // メソッド呼び出しの後処理
       container.postInvoke(...);
       .............
       return s;
   }
```

メソッドのはさみ込み

EJB2.1の「三つ組み」は、挟み込みで メソッドを書き換えるための枠組み に他ならない

- コードの書き換えメカニズムとしてAOPの先駆としての「三つ組み」
- コード・ジェネレータのSyntax Sugarとしての「三つ組み」

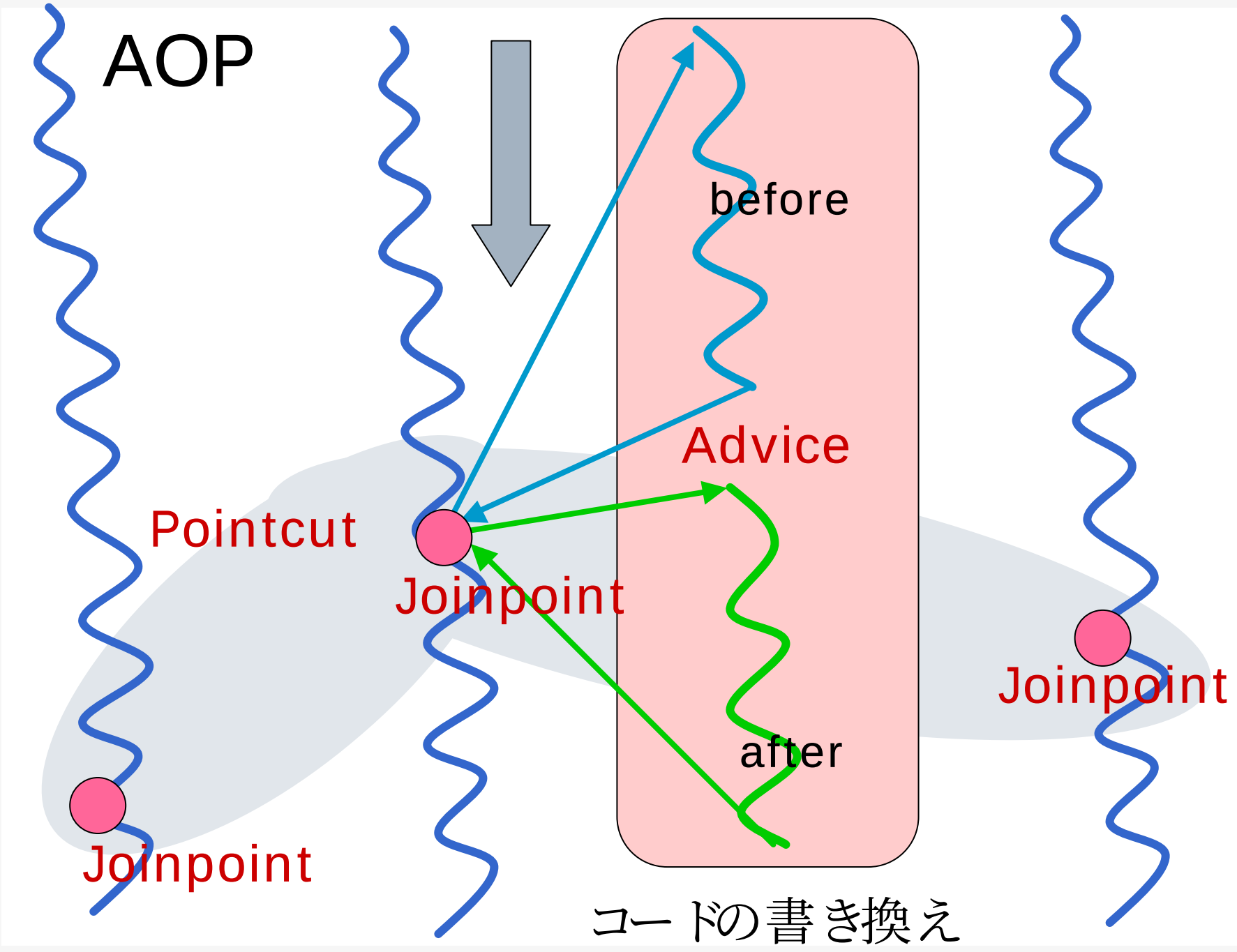
メタプログラミングとしてのAOP

対象プログラムのコード書き換え

プログラムとしてのAOP

AOPの基本的な概念

- Joinpoint　実行中のプログラムのある点
- Advice　あるJointpointで実行される動作
- Pointcut　あるAdviceが呼び出されることを指定した、Jointpointの集合


AOP
before
Advice
Pointcut
Joinpoint
after
Joinpoint
Joinpoint
コードの書き換え

EJBプログラミングの難しさ

EJB2.1でのCMP
(Container Managed Persistency)

EJB2.1のCMP

- Abstractクラスで 実装」?
- プログラマが書くコードと、生成されるコードとの関係が見えにくい。
- HomeインターフェースのFinder
- EJBコード以外に、Deployment Descriptor内にSQLを書くというスタイル。

Syntax Sugar の複雑化

ejb-jar.xml

```
<query>
    <query-method>
        <method-name>ejbSelectSports
        </method-name>
        <method-params>
          <method-param>team.LocalPlayer
          </method-param>
        </method-params>
    </query-method>
    <ejb-ql> select distinct t.league.sport
             from Player p, in (p.teams) as t
             where p = ?1
    </ejb-ql>
</query>
```

J2EEでの開発の難しさ

- マルチティアでの複雑なデザインパターン
  - Webアプリケーション化
- Entity Beansの利用の難しさ
  - Session Beansだけを使う
- CMPコーディングの複雑さ
  - BMPだけを使う

J2EE批判とAlternativeの提案

Expert One-on-One J2EE Design and Development

by Rod Johnson

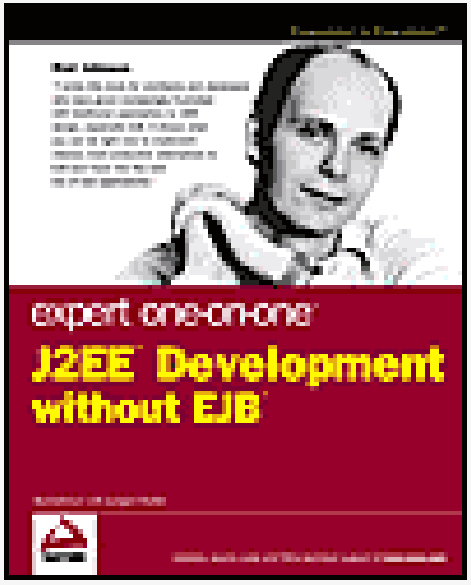
2002/10

Implications of Using EJB

- **Using EJB makes applications harder to test**
- **Using EJB makes applications harder to deploy**
- **Using EJB with remote interfaces may hamper practicing OO design**
- **Using EJB may make simple things hard**
- **Reduced choice of application servers**

Expert One-on-One
J2EE Development without EJB by Rod Johnson

2004/07



J2EE at a Crossroads

- **J2EE applications are usually too expensive to develop.**
- **J2EE has significant issues with ease of development.**
- **J2EE applications tend to be unnecessarily complex.**
- **There is a growing movement in the J2EE community toward simpler solutions and less use of EJB.**

Inversion of Control Containers and the Dependency Injection pattern

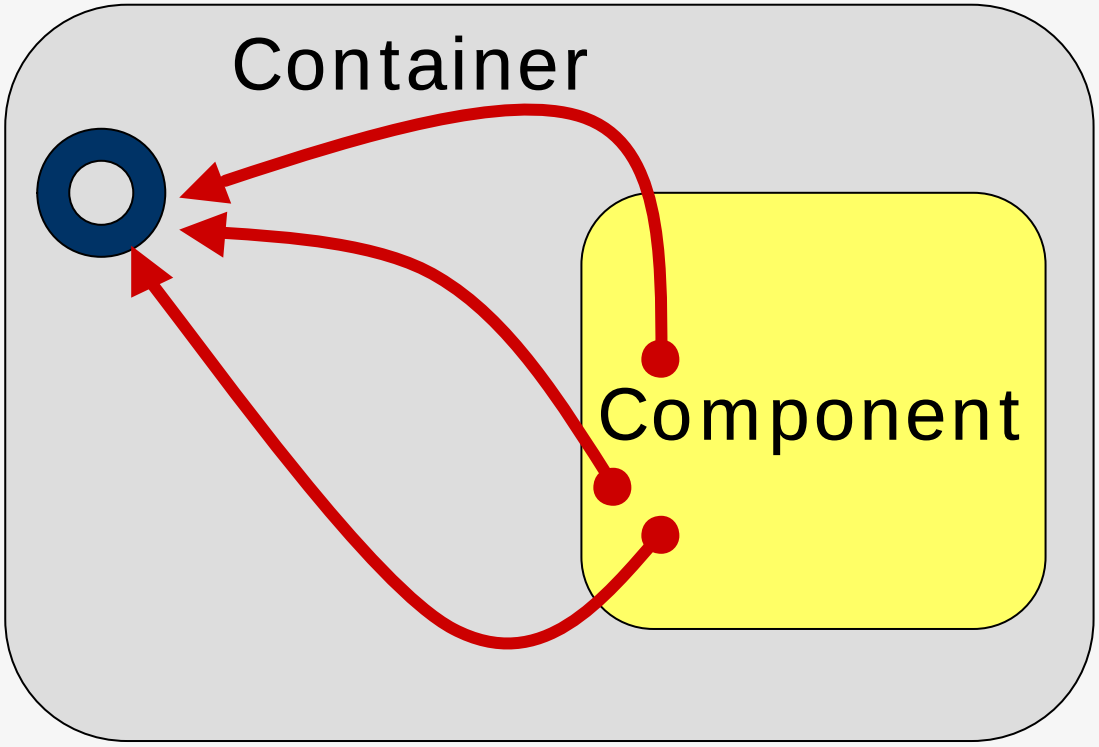
Martin Fowler
2004/01

One of the entertaining things about the enterprise Java world is the huge amount of activity in building alternatives to the mainstream J2EE technologies, much of it happening in open source. Much of this is a reaction to the heavyweight complexity of much in the mainstream J2EE world, but much of it is also exploring alternatives and coming up with creative ideas.

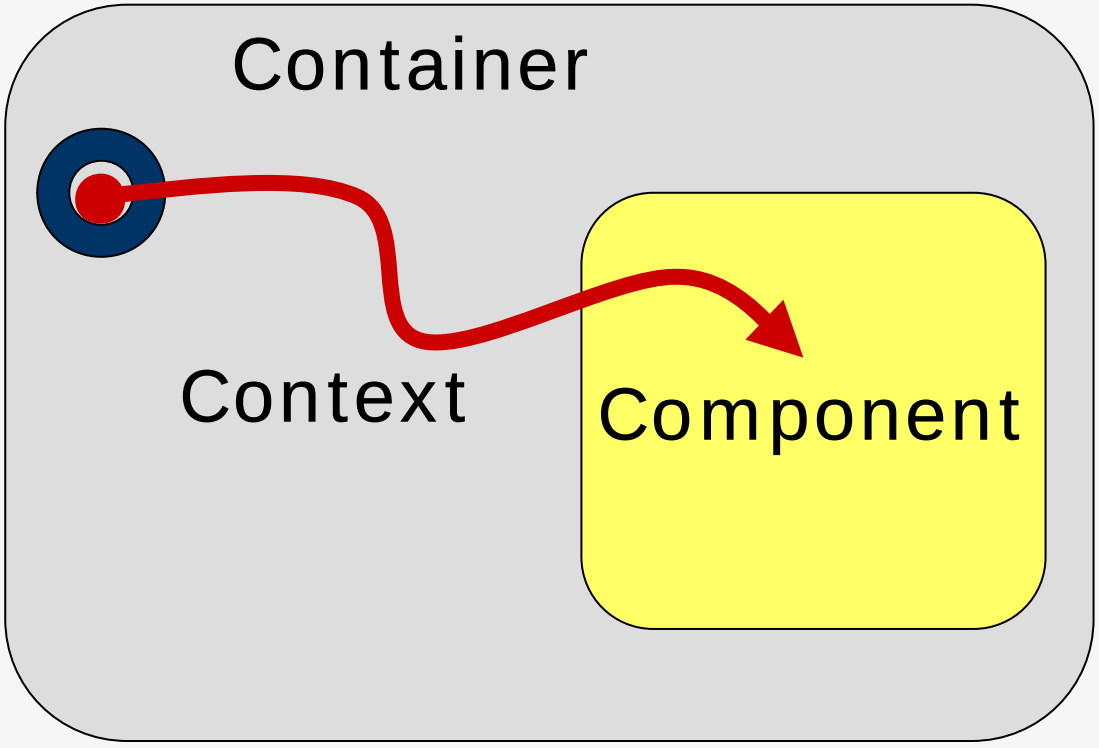
J2EE批判とAlternativeの提案

- 軽量コンテナ　Spring
- O/Rマッピングツール　Hibernate
- AOPの利用　JBoss

EJB2.1
コンポーネントがコンテナから情報を取得する



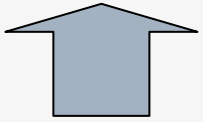
EJB3.0
コンテナがコンポーネントに情報を注入する



制御の逆転（Inverse of Control）と
従属性注入（Dependency Injection）

EJB3.0
DI Annotaion on Setter Methods

```
@Inject
public void setGreeting(String greeting){
   _greeting = greeting;
}
```



```
<ejb-server jndi-name="java:comp/env/ejb">
   <bean type="qa.TestBean">
      <init greeting="Hello, World from web.xml"/>
   </bean>
</ejb-server>
```

Resin configuration-file

J2EE5.0の登場

J2EE5.0の登場

- J2EEに対する批判を受け止め、プログラミングスタイルを大幅に改善する。
  - Ease of Development
  - Ease of Development
  - Ease of Development
  - Ease of Development
  - Ease of Development
  - .....

J2EE5.0の登場

- J2EE5.0 = JSF1.0 + EJB3.0
  (+ JSTL1.1 + JSP2.1
  + JAXB2.0 + JAX-RPC2.0)

- J2EE5.0のEoD =
  GUIを利用したJ2EEアプリの開発
  ＋ Annotationを利用したプログラミング

EJB3.0の特徴

- Annotation
- Configuration by exception
- Dependency injection mechanisms
- POJO / POJI
- Elimination of Home interface
- Simplified CMP
- .....

EJB3.0プログラミング

- Homeインターフェースがいらない
- Local（Remote）インターフェースがいらない
- 特別なcreateやremoveメソッドがいらない
- 特別なfindやselectメソッドがいらない
- Deployment descriptorがいらない

Entity Beans

科目担当テーブル　courses

キー id	科目 course	教員 teacher
1	Java	丸山
2	XML	植田
3	SQL	安藤

テーブルcoursesを生成するSQL

```
CREATE TABLE courses (
   id INTEGER PRIMARY KEY auto_increment,
   course VARCHAR(250),
   teacher VARCHAR(250) );

INSERT INTO courses VALUES('Java','丸山');
INSERT INTO courses VALUES('XML','植田');
INSERT INTO courses VALUES('SQL','安藤');
```

テーブルcoursesに EntityBeanを対応させる

- EntityBeanに対応させる ===> @Entity
- テーブルの名前は courses ===> @Table
- キーは自動生成 ===> @Id
- キー項目の名前は id ===> @Column
- 基本的な項目 course ===> @Basic
- 基本的な項目 teacher ===> @Basic

テーブルcoursesに EntityBeanを対応させる Annotation

```
@Entity
@Table( name="course“ )
@Id( generator=javax.ejb.GeneratorType.
   AUTO )
@Column( name="id” )
@Basic
@Basic
```

EJB3.0 CourseBean.java

```
package example;
@javax.ejb.Entity
@javax.ejb.Table (name="courses“ )
public class CourseBean {
    private int _id;
    private String _course;
    private String _teacher;
    @javax.ejb.Id
    ( generator=GeneratorType.AUTO
    @javax.ejb.Column (name="id")
    public int getId() { return _id; }
    public void setId(int id) { _id = id; }
```

EntityBeanに対応させる

テーブルの名前は courses

キーは自動生成
キー項目の名前はid

EJB3.0 CourseBean.java

```
    @javax.ejb.Basic
    public String getCourse() {
       return _course;
    }
    public void setCourse(String course)
      {
        _course = course;
    }
    @javax.ejb.Basic
    public String getTeacher() {
       return _teacher;
    }
    public void setTeacher(String teacher)
      {
        _teacher = teacher;
     }
}
```

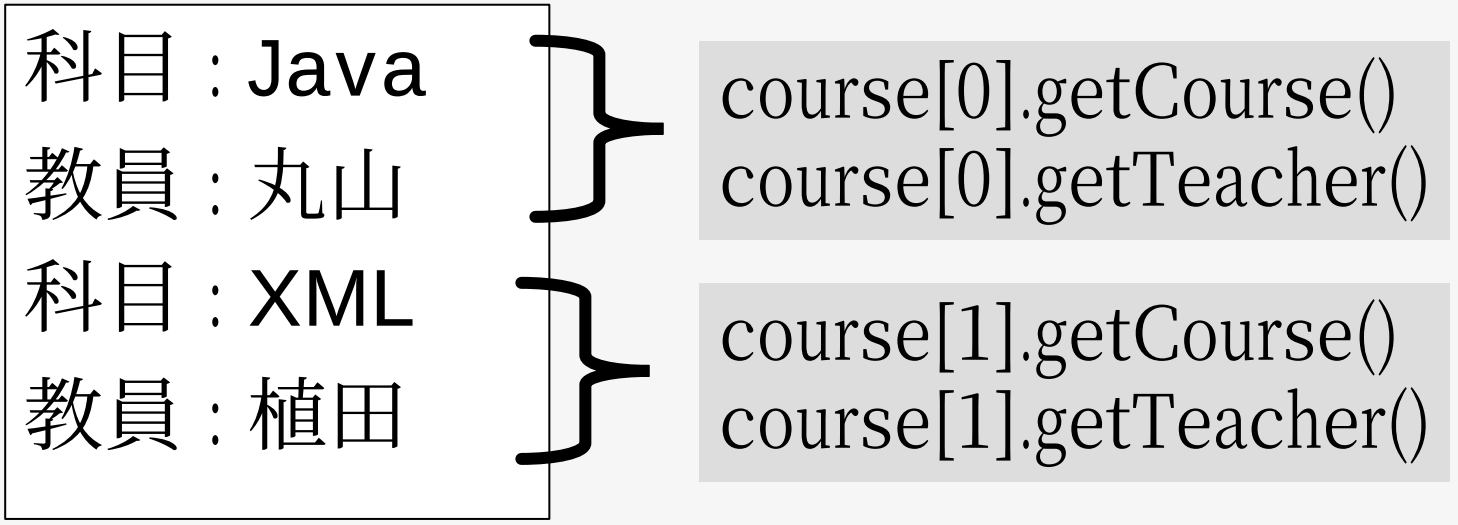
基本的な項目
course

基本的な項目
teacher

EJB3.0 クライアント側での呼び出し行の発見　find

```
private EntityManager _manager;
CourseBean [] course = new CourseBean[2];
course[0] = (CourseBean)
  _manager.find("CourseBean", new Integer(1));
course[1] = (CourseBean)
  _manager.find("CourseBean", new Integer(2));
for (int i = 0; i < course.length; i++) {
  out.println(“科目 : " + course[i].getCourse());
  out.println(“教員 : " + course[i].getTeacher());
}
```

先のプログラムの出力


科目 : Java
教員 : 丸山
科目 : XML
教員 : 植田
course[0].getCourse()
course[0].getTeacher()
course[1].getCourse()
course[1].getTeacher()

EJB3.0で、テーブル間の関係はどのように表現されるか?

多対１ Many-to-One の関係

ゼミのテーブル Seminar

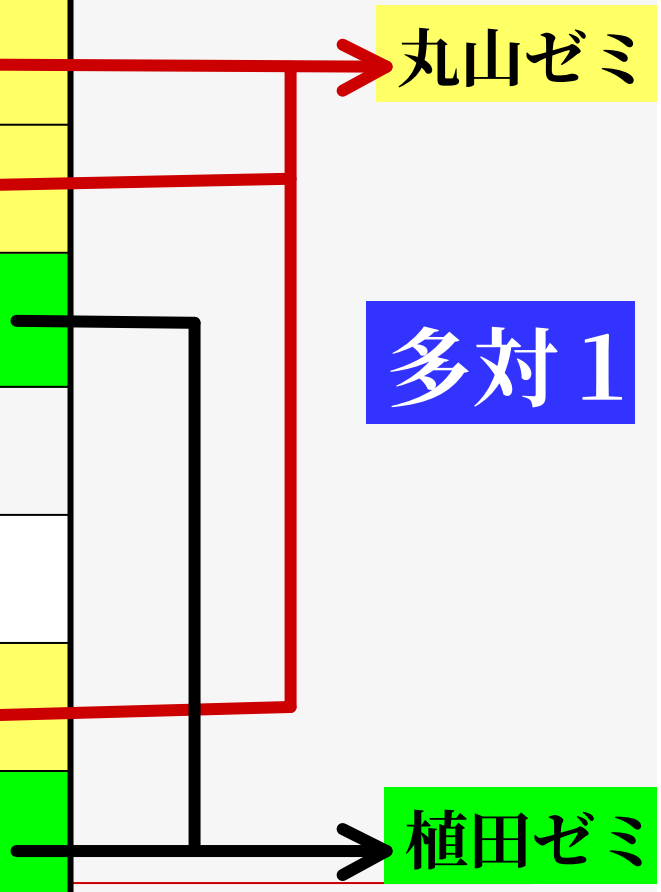
キー seminar_id	ゼミ名 name
1	丸山ゼミ
2	植田ゼミ
3	安藤ゼミ
4	門間ゼミ
5	金山ゼミ

EJB3.0
Seminar.java

```
@Entity
public class Seminar {
   @Id
   @Column ( name= "seminar_id")
   public long getId() ;
   @Basic
   public String getName() ;
}
```

学生テーブル Student

キー student_id	学生 name	ゼミid seminar_id
1	藤木文彦	1
2	岩本和久	1
3	坂本寛	2
4	五十嵐理佳	4
5	塚本智宏	3
6	加藤潔	1
7	佐々木政憲	2
8	斉藤吉広	5



EJB3.0
Student.java

```
@Entity
public class Student {
   @Id
   @Column ( name = "student_id“ )
   public long getId() ;
   @Basic
   public String getName() ;
   @ManyToOne
   @JoinColumn ( name = "seminar_id“ )
   public Seminar getSeminar()
}
```

EJB3.0
クライアント側での呼び出し

```
Query allStudent = _entityManager.createQuery(
    "SELECT o FROM Student o");
List students = allStudent.listResults();
for (int i = 0; i < students.size(); i++) {
    Student student = (Student) students.get(i);
    out.println(student.getName() + “ は、” +
    student.getSeminar().getName() + “所属です。”);
}
```

先のプログラムの出力

藤木文彦は、丸山ゼミ所属です。
岩本和久は、丸山ゼミ所属です。
坂本寛は、植田ゼミ所属です。
五十嵐理佳は、門間ゼミ所属です。
塚本智宏は、安藤ゼミ所属です。
加藤潔は、丸山ゼミ所属です。
佐々木政憲は、植田ゼミ所属です。
斉藤吉広は、金山ゼミ所属です。

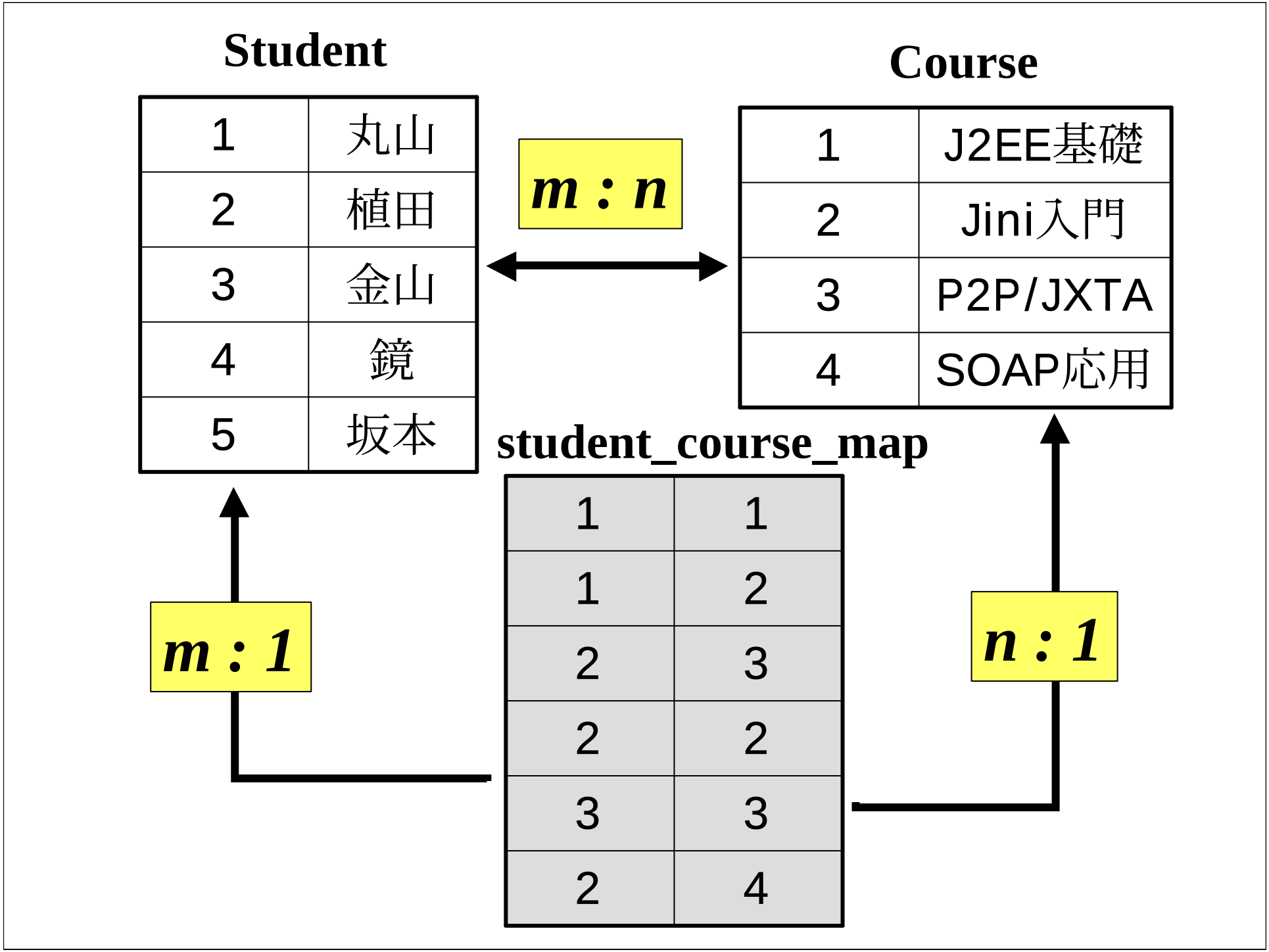
EJB3.0
クライアント側での呼び出し

```
Query semiStudent = _entityManager.createQuery(
    "SELECT s FROM Student s WHERE s.seminar.name =?1");
semiStudents.setParameter(1, "丸山ゼミ");
List students = semiStudent.listResults();
out.println("丸山ゼミ学生リスト");
for (int i = 0; i < students.size(); i++) {
    Student student = (Student) students.get(i);
    out.println(student.getName());
}
```

プログラムの出力

丸山ゼミ学生リスト
藤木文彦
岩本和久
加藤潔

多対多 Many-to-Many の関係


Student
1 丸山
2 植田
3 金山
4 鏡
5 坂本
m : n
Course
1 J2EE基礎
2 Jini入門
3 P2P/JXTA
4 SOAP応用
student_course_map
1 1
1 2
2 3
2 2
3 3
2 4
m : 1
n : 1

Course.java

```
@Entity
public class Course {
   @Id
   @Column ( name = "course_id“ )
   public long getId() ;
   @Basic
   public String getName() ;
}
```

Student.java

```
@Entity
public class Student {
   @Id
   @Column ( name = "student_id“ )
   public long getId() ;
   @Basic
   public String getName() ;
   @ManyToMany( targetEntity="Course")
   @AssociationTable (
      table=@Table (name="student_course_map"),
      joinColumns=@JoinColumn (name="student_id"),
      inverseJoinColumns=@JoinColumn (name="course_id")
   )
   public Collection getCourses() ;
}
```

クライアントからの呼び出し

```
Query allStudent = _entityManager.createQuery(
    "SELECT o FROM Student o“ );
List students = allStudent.listResults();
for (int i = 0; i < students.size(); i++) {
    Student student = (Student) students.get(i);
    out.println(student.getName() + “の履修科目 ”);
    Collection courses = student.getCourses();
    Iterator iter = courses.iterator();
    while (iter.hasNext()) {
        Course course = (Course) iter.next();
        out.println( “    “＋course.getName());
    }
}
```

先のプログラムの出力

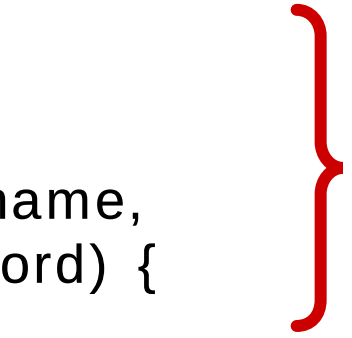
丸山の履修科目
    J2EE基礎
    Jini入門
植田の履修科目
    Jini入門
    P2P/JXTA
金山の履修科目
    P2P/JXTA
    SOAP応用

Annotationの手法

JSR181 -- Web Services Metadata for Java Platform

```
@WebService(
  name="ExampleWebService",
  targetNamespace="http://openuri.org/11/2003/
      ExampleWebService" )
@SOAPBinding(
    style=SOAPBinding.Style.RPC, use=SOAPBinding.Use.LITERAL)
public class ExampleWebServiceImpl {
  @WebMethod( action="urn:login" )
  @WebResult( name="Token" )
  public LoginToken login(
    @WebParam( name="UserName" ) String username,
    @WebParam( name="Password" ) String password) {
    // ...
   }
    // ...
}
```

RPC/literal



BottomUpで、
先のJavaコードから生成されるWSDL

```
.....
 <message name="login">
   <part name="UserName" type="s:string"/>
   <part name=”Password” type=”s:string”/>
 </message>

 <message name="loginResponse">
   <part name="Token" type="LoginToken"/>
 </message>

 <portType name="ExampleWebService">
   <operation name=”login”>
     <input message=”tns:login”/>
     <output message=”tns:loginResponse”/>
   </operation>
.....
```

@WebParam

@WebResult

@WebMethod

Annotationの手法　(JSR181)

- Javaプログラムの持つ情報　＜　WSDLの持つ情報

- Javaプログラムの持つ情報
  ＋annotationの情報　＝　WSDLの持つ情報

JSR220 -- Enterprise JavaBeans 3.0 Specification

```
@javax.ejb.Entity
public class Course {
  @javax.ejb.Id
  @javax.ejb.Column ( name="student_id“ )
  public long getId()

  @javax.ejb.Basic
  public String getName()

  @javax.ejb.ManyToMany( targetEntity="Course“ )
  @javax.ejb.AssociationTable (
      table=@javax.ejb.Table (name="student_course_map"),
      joinColumns=@javax.ejb.JoinColumn (name="student_id")",
      inverseJoinColumns=@javax.ejb.JoinColumn
     (name="course_id")“ )
  public Collection getCourses()
}
```

Annotationは 意味」を持つ

@javax.ejb.ManyToMany …

テーブルStudentはテーブルCourseと 多対多」の関係にある。

@javax.ejb.AssociationTable …

この二つのテーブル間の 多対多」の関係は、テーブル student_course_mapで表現され、そのテーブルは、この Studentテーブルのstudent_id項目と、Courseテーブル のcourse_id項目から構成され、次のメソッドgetCoursesは、 この 多対多」の関係の下で、キーstudent_idに対応する、 複数のcourse_id達の集合を返す。

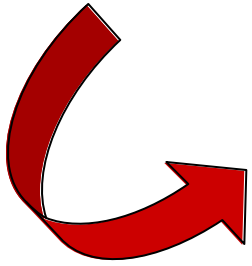
Annotationの手法　(JSR220)

- Javaプログラムの持つ情報　＜
  テーブル間の関係の持つ情報

- Javaプログラムの持つ情報
  ＋annotationの情報　＝
  テーブル間の関係の持つ情報

JSR222 -- The Java™ Architecture for XML Binding (JAXB) 2.0

```
public class Trade {
  @XmlAttribute
  String getSymbol();
  void setSymbol(String);
  @XmlElement
  int getQuantity();
  void setQuantity(int);
}
```



```
<xsd:complexType
  name="trade">
  <xsd:sequence>
    <xsd:attribute
       name="symbol"
       type ="xsd:string"/>
    <xsd:element
       name="quantity"
       type ="xsd:int"/>
  </sequence>
</xsd:complexType>
```

Annotationの手法　(JSR222)

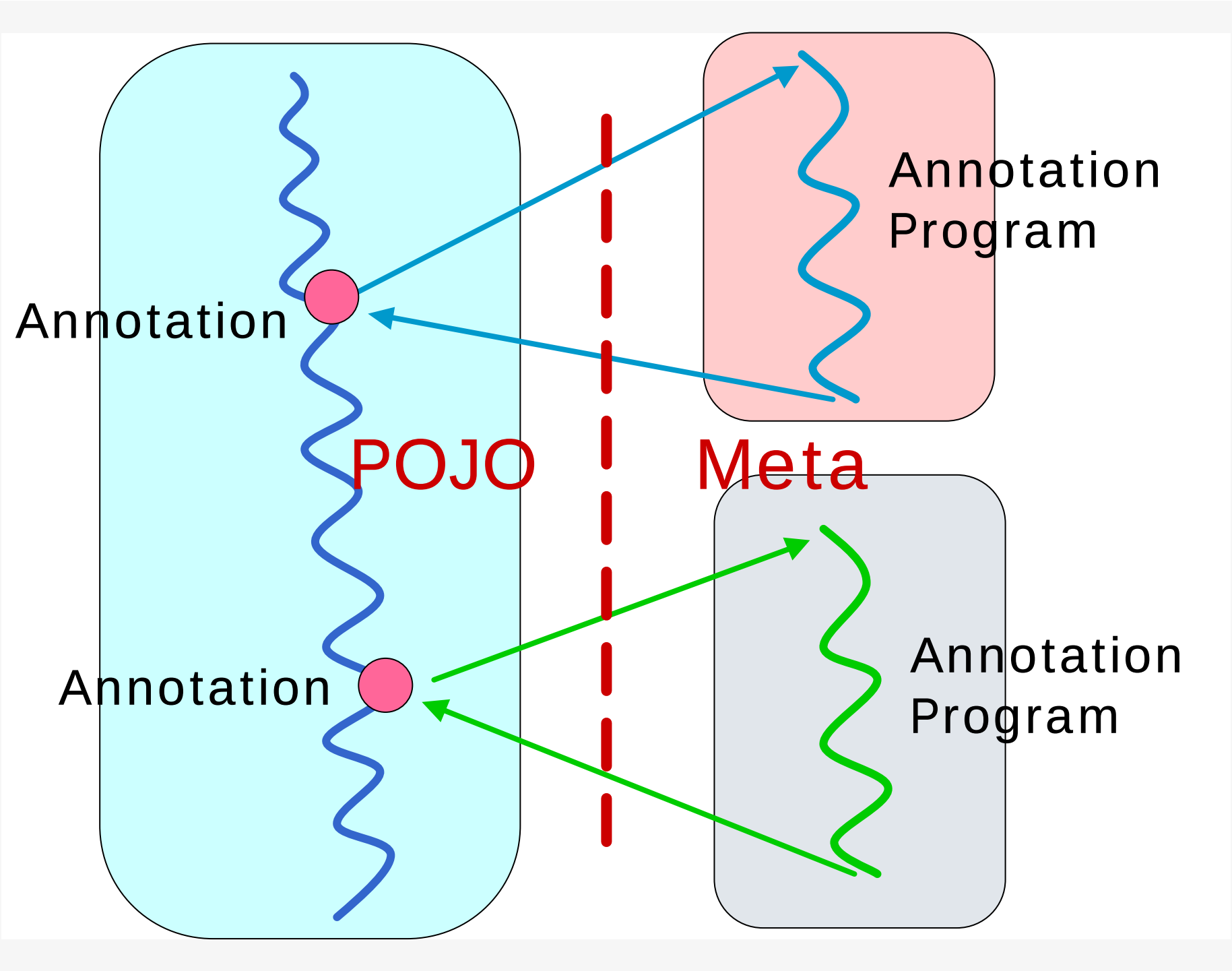
- Javaプログラムの持つ情報　＜
  XML Documentの持つ情報

- Javaプログラムの持つ情報
  ＋annotationの情報　＝
  XML Documentの持つ情報

Annotationの手法

- Javaプログラムの持つ情報　＜
  あるEntityの持つ情報
- Javaプログラムの持つ情報
  ＋annotationの情報　＝
  あるEntityの持つ情報

Annotation　＝　Semantic Sugar


Annotation
Annotation
Program
POJO
Meta
Annotation
Annotation
Program

一般プログラマと Annotationプログラマ

- 一般のプログラマは、Annotationを利用してプログラムを作るが、Annotationプログラムを書く必要はない。

- Annotationプログラマは、具体的な処理を行うプログラムを対象として、その振る舞いを規定するメタ・プログラムとしてAnnotationプログラムを書く。

まとめ

まとめ

- 「EJB=難しい技術」という思い込みを捨てて、J2EEのEoD対応として、EJB3.0を考えることが重要である
- 技術者の再教育を通じて、Annotationを利用した、EJB3.0の新しいプログラミング・スタイルに早く備えることが重要である
- こうした対応とGUIツールの活用を通じて、EoDによる生産性の向上を目標とすべきこと