J2EEの新しい動向

稚内北星学園大学
丸山不二夫

Agenda

1. J2EEとは何か?
2. EJBとコンテナ
3. J2EEとWebサービス / Grid
4. J2EEとビジネスプロセスの統合

1. J2EEとは何か?

- J2EEの誕生
- インターネットの爆発とJ2EE / Webサービス
- n-tier モデルの成立
- サーバサイド・プログラミングのメリット
- コンポーネント / コンテナ　モデル
  Enterprise Java BeanとApplication Server
- コンポーネントの配備 (deploy) と
  宣言的プログラミング
- J2EEを構成する要素技術

J2EEの誕生

Mark Hapner

J2EEの前史

- 1997年4月12日 Sunは、JCP（Java Community Process）を使った企業向けのJavaプラットフォームの開発開始をアナウンス。

- サーバサイドのコンポーネントモデルを定義して、Javaアプリケーションサーバにベンダーに依存しないJavaインターフェースを提供することを、Enterprise JavaBean APIの中心目標に。

J2EEの誕生

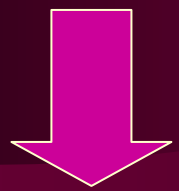
- 二年後の1999年10月、Enterprise Java APIの仕様が完成。多くのベンダの実装が利用可能に。

- 特に、Enterprise JavaBeans(EJB) は、Javaアプリケーションサーバのデファクト・スタンダードなコンポーネントモデルになる。

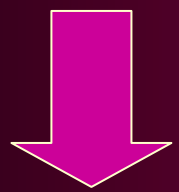
- **J2EE = Java™ 2 SDK, Enterprise Edition**

J2EEの発展

- 1999年　12月17日　　**J2EE 1.2**

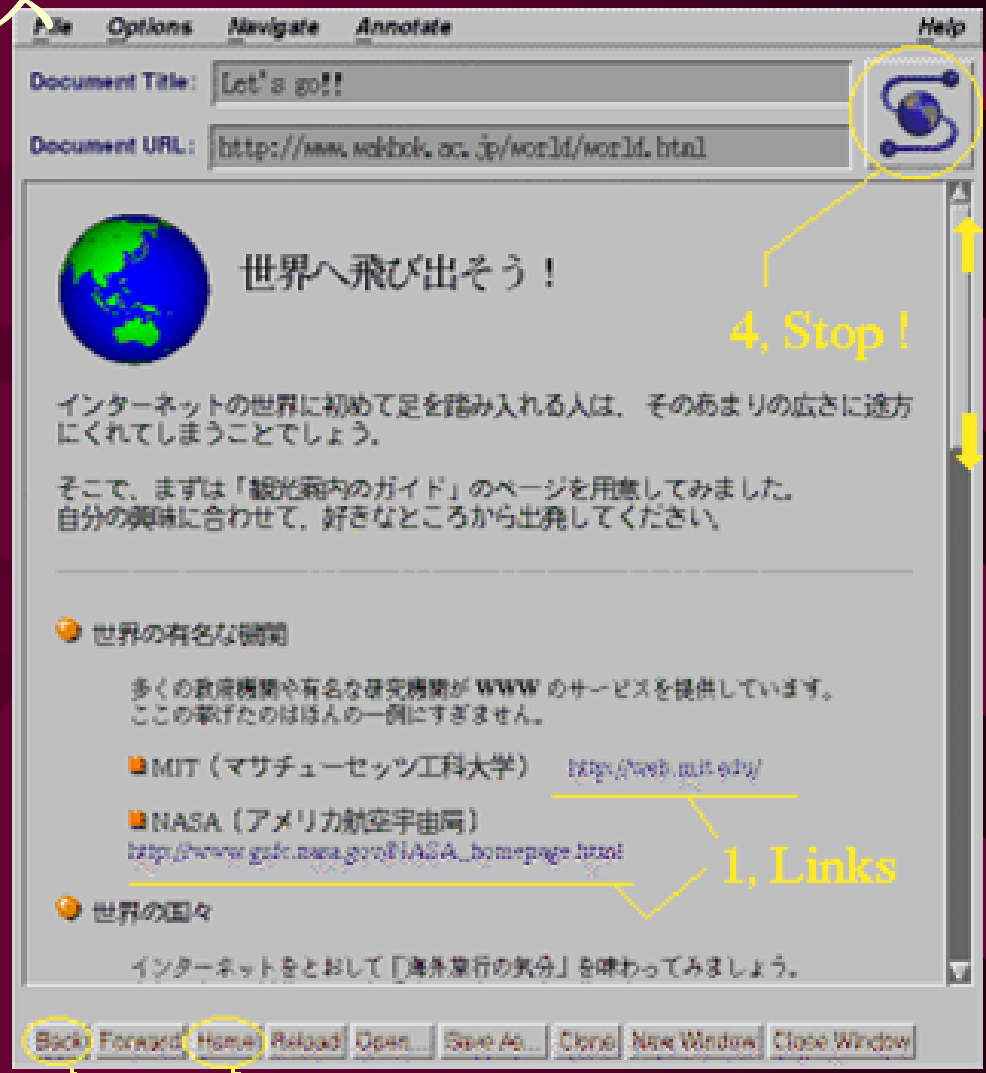


- 2001年　 9月17日　　**J2EE 1.3**



- 2003年　11月24日　　**J2EE 1.4**

インターネットの爆発と
J2EE / Webサービス

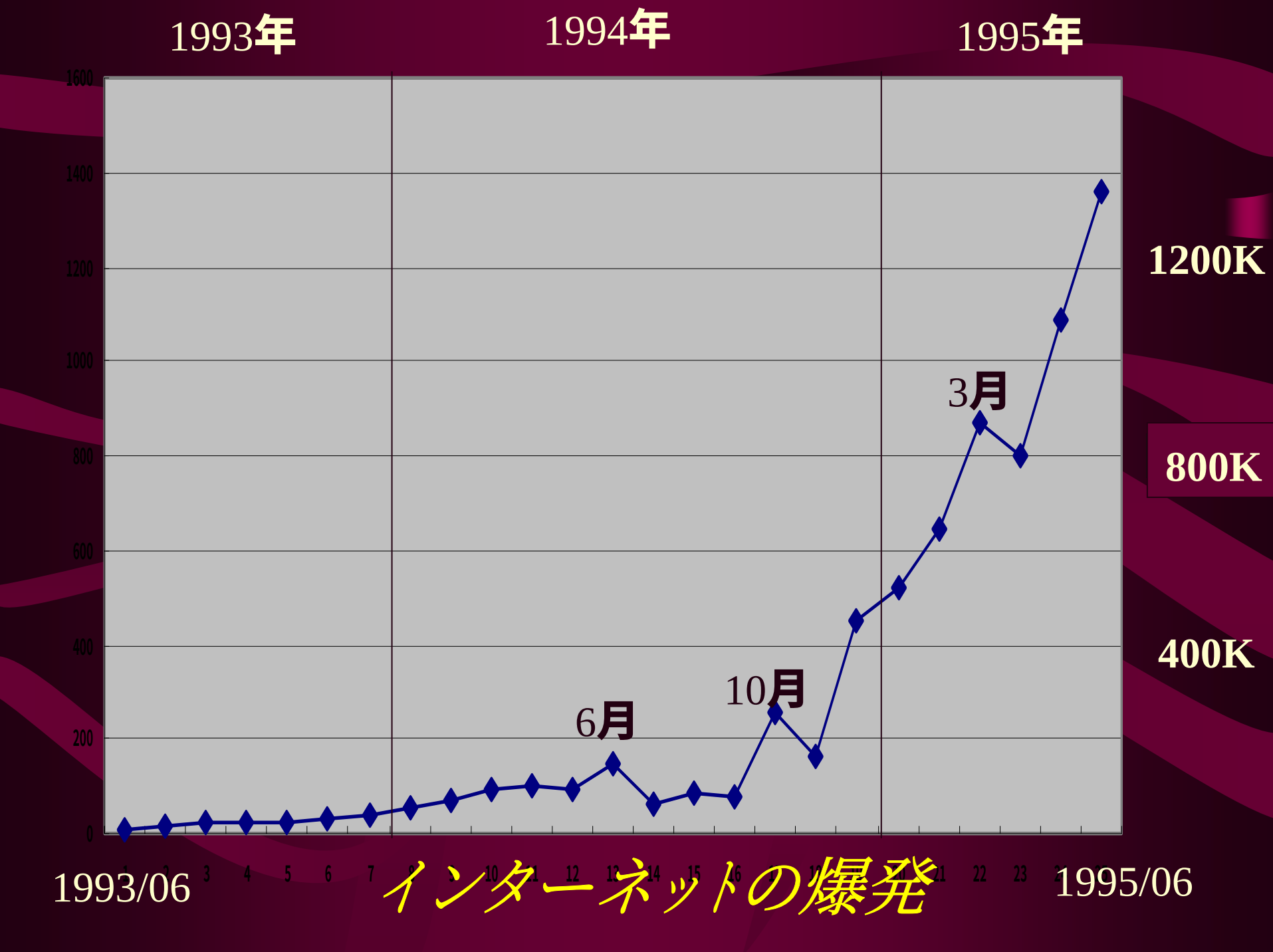


1993年 1月 WWWサービス

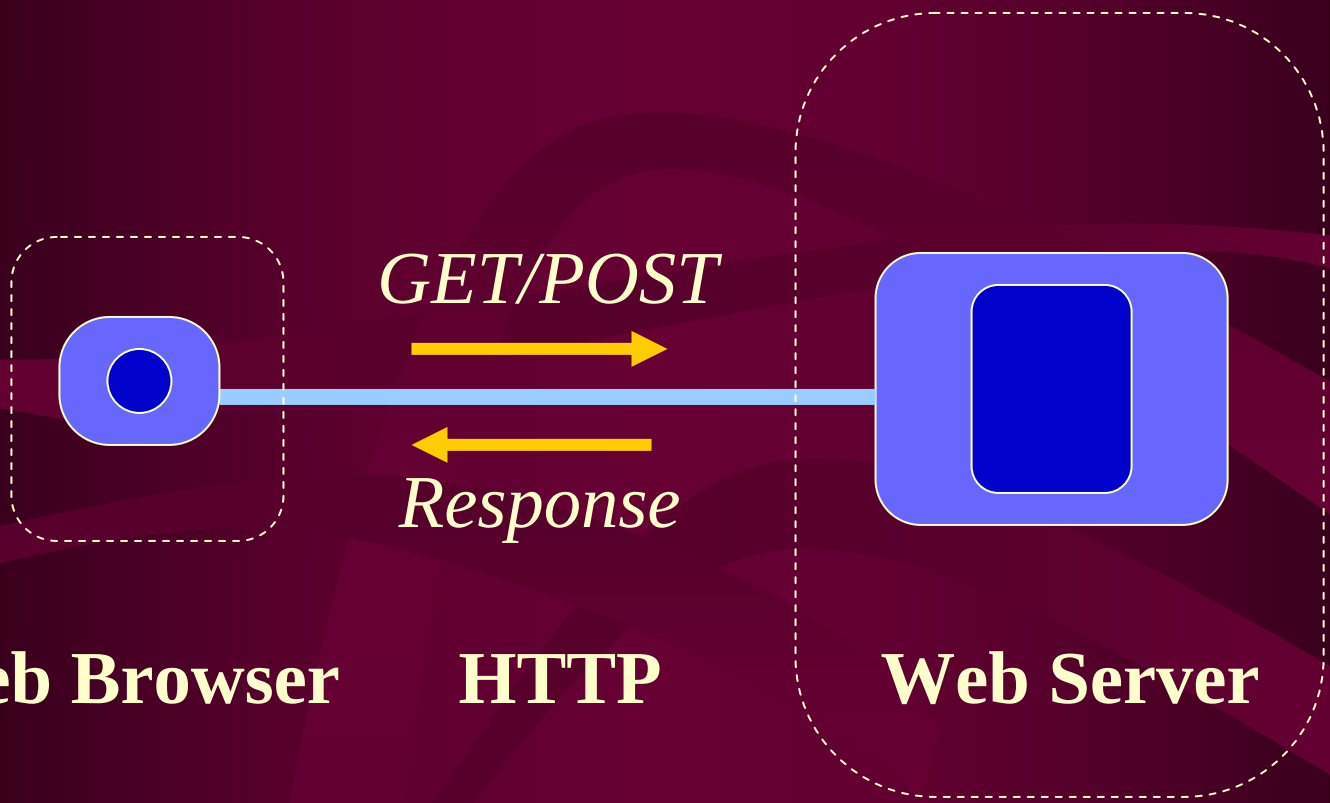
NCSAのWhat‘s New ページの容量

1995年12月	KB	1994年12月	448KB	1993年12月	40KB
1995年11月	KB	1994年11月	163KB	1993年11月	33KB
1995年10月	KB	1994年10月	257KB	1993年10月	21KB
1995年 9月	KB	1994年 9月	76KB	1993年 9月	24KB
1995年 8月	KB	1994年 8月	87KB	1993年 8月	20KB
1995年 7月	KB	1994年 7月	65KB	1993年 7月	18KB
1995年 6月	1363KB	1994年 6月	147KB	1993年 6月	11KB
1995年 5月	1084KB	1994年 5月	97KB		
1995年 4月	797KB	1994年 4月	102KB		
1995年 3月	872KB	1994年 3月	93KB		
1995年 2月	645KB	1994年 2月	71KB		
1995年 1月	522KB	1994年 1月	58KB		

インターネットの爆発


1993年
1994年
1995年
1600
1400
1200
1000
800
600
400
200
0
1200K
800K
400K
3月
10月
6月
1993/06
1995/06
インターネットの爆発

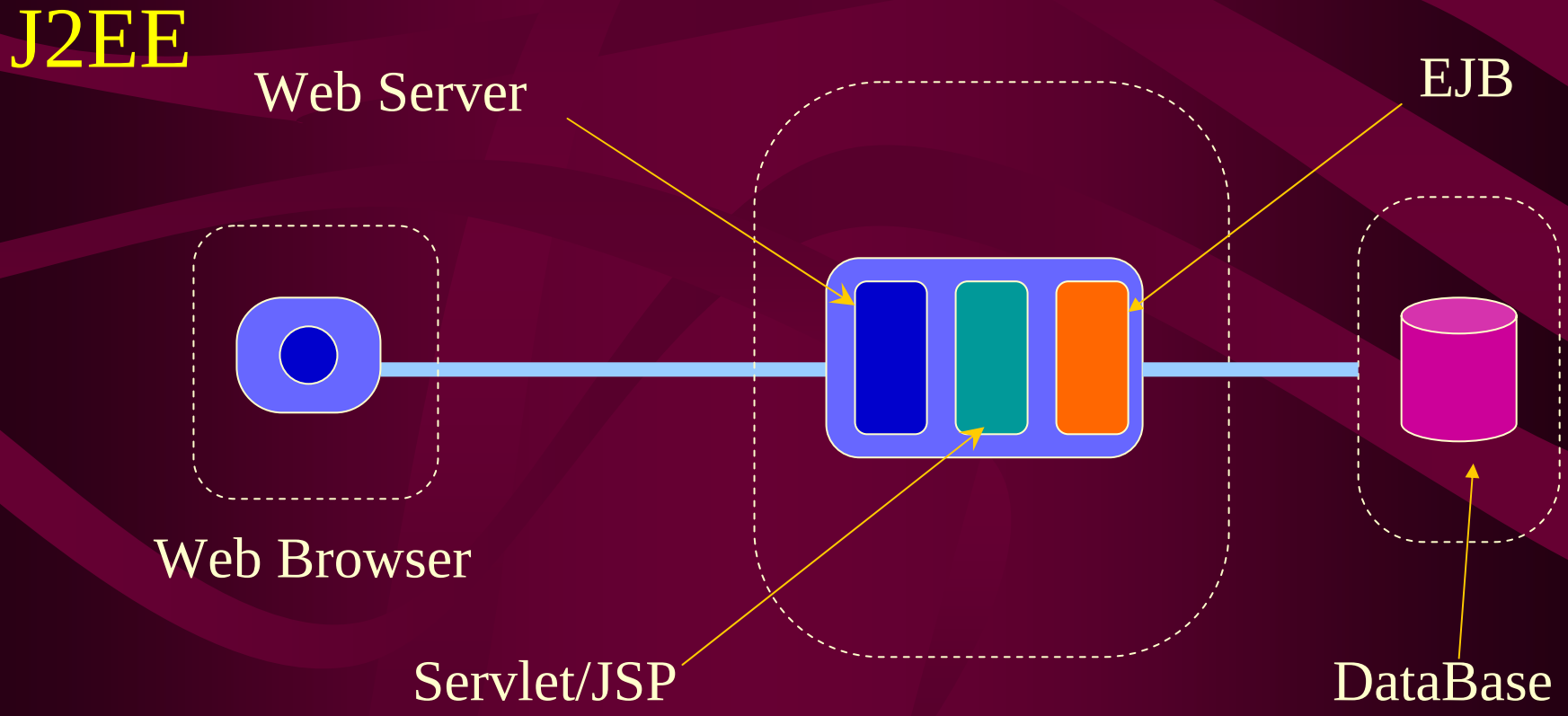
Web Browser とWeb Server



サーバ･クライアントモデル

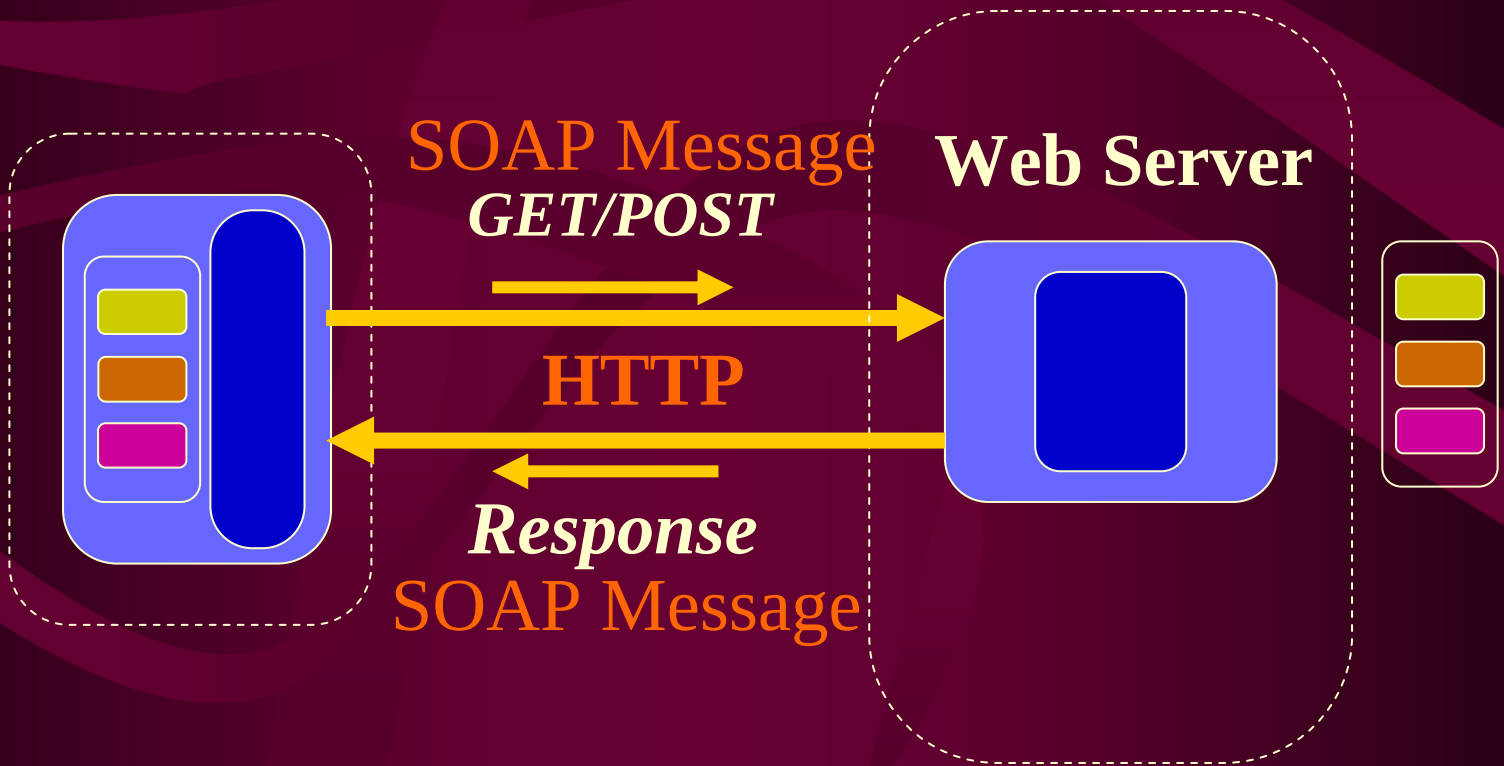
J2EEとは？

汎用クライアントとして
Webブラウザを利用する
サーバ・クライアント・モデル



Webサービスとは？

汎用サーバとして
Webサーバを利用する
サーバ・クライアント・モデル



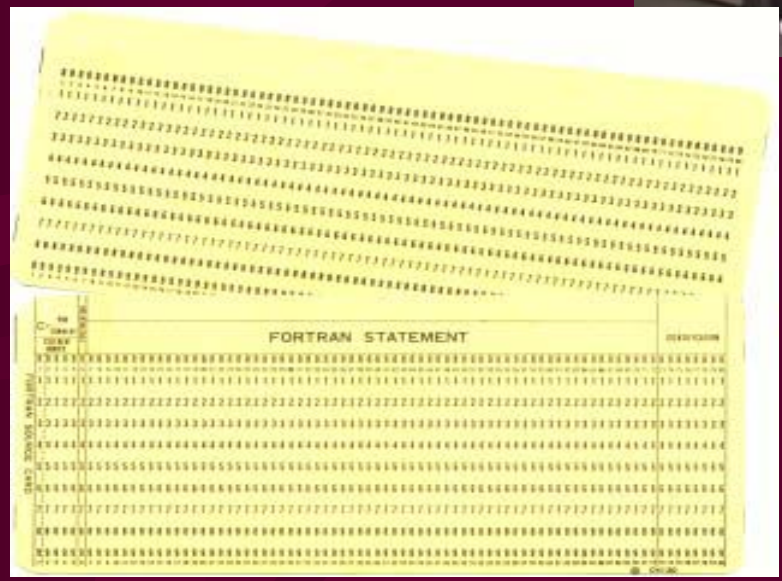
n-tier モデルの成立

入出力チャンネルから
ネットワーク上でのリソースの共有へ

3270端末エミュレータ

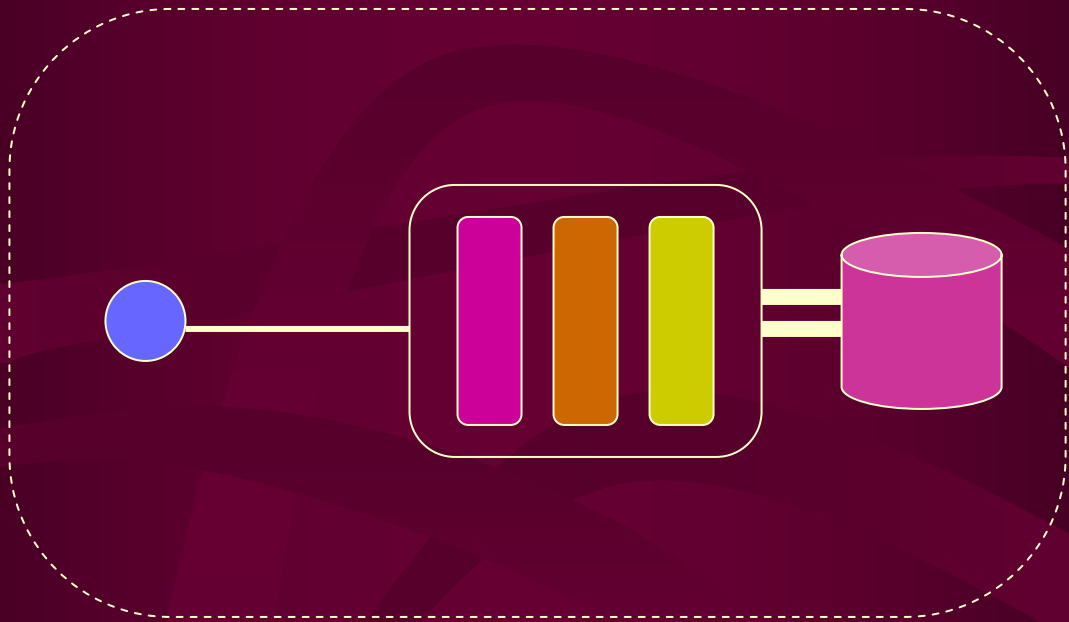
IBM 9672

IBM360



ASR-33

1-Tier モデル



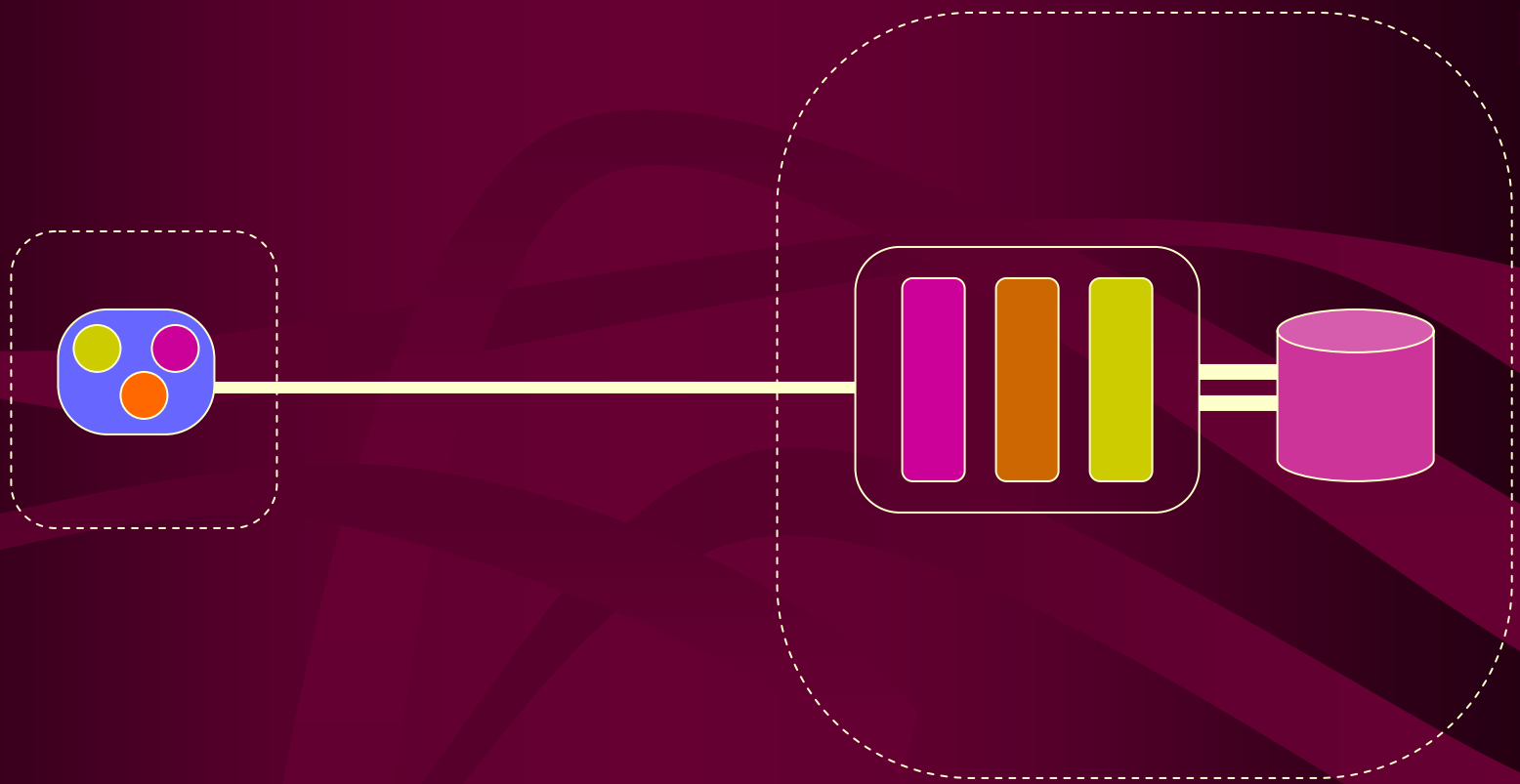
入出力端末･チャンネル

IBM PC、'81年8月12日 誕生

Sun3/260 1987年

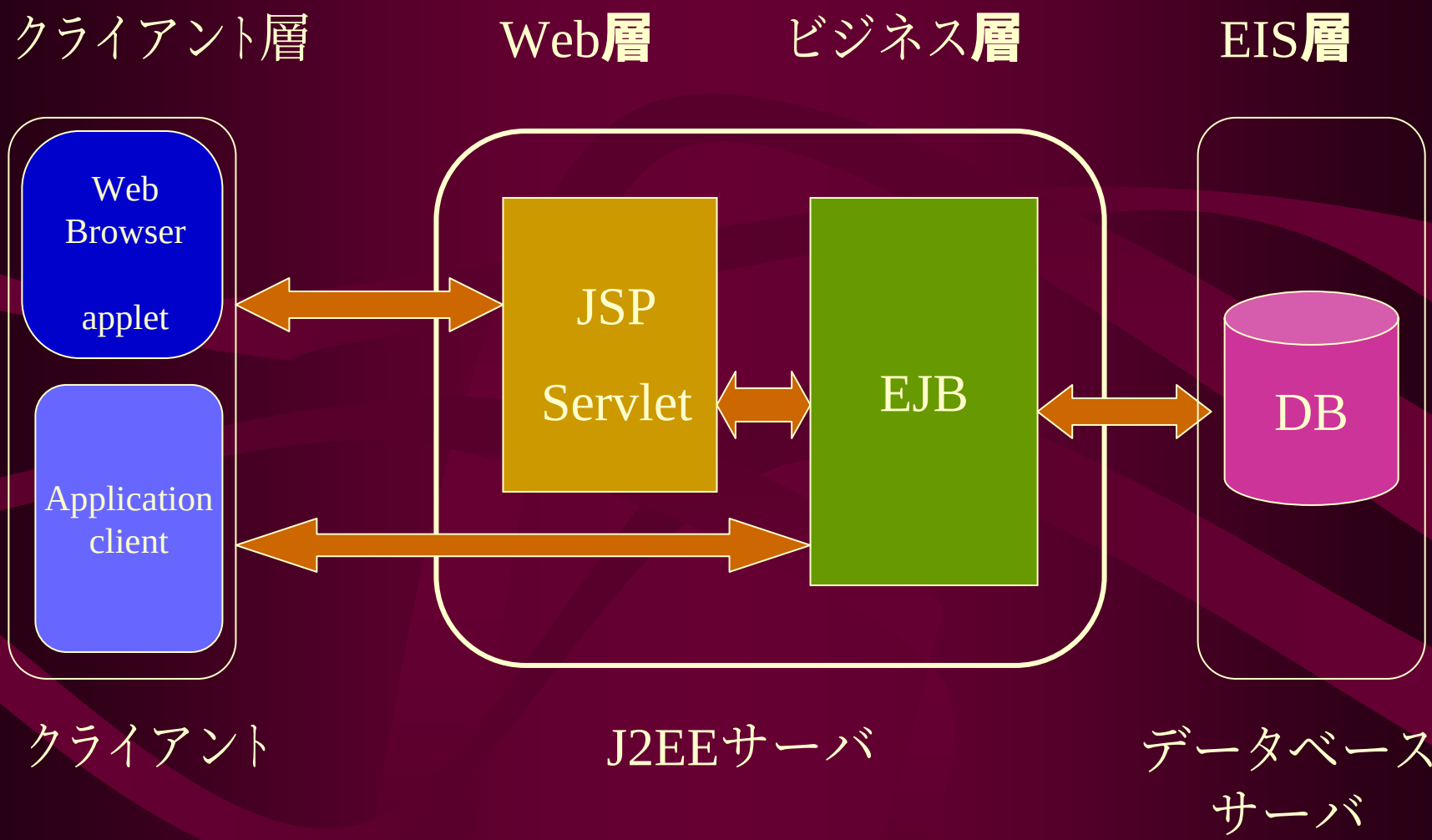
VAX-11/780(1977)

2-Tier モデル

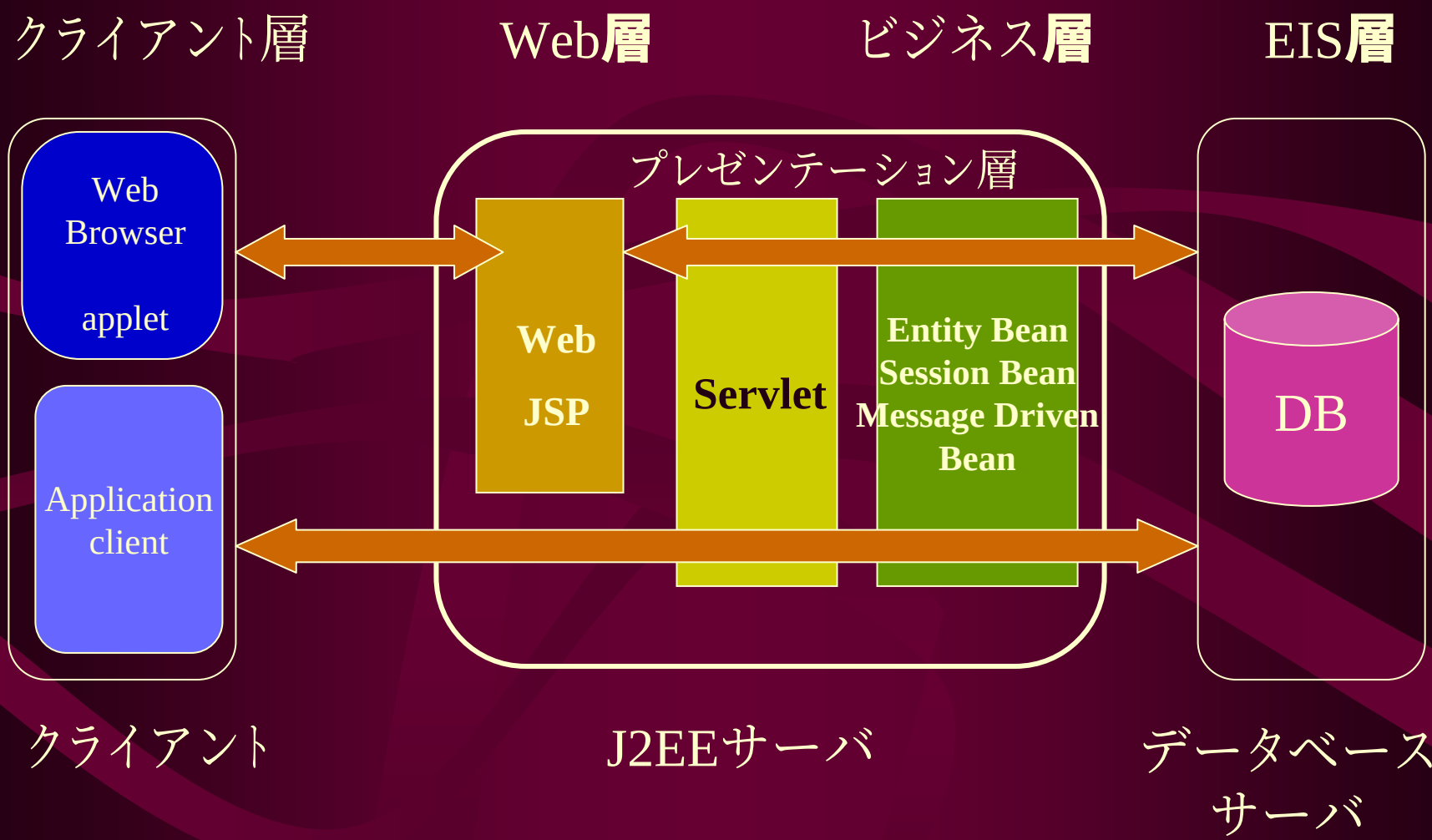


サーバ･クライアントモデル

3-Tier モデル　J2EE



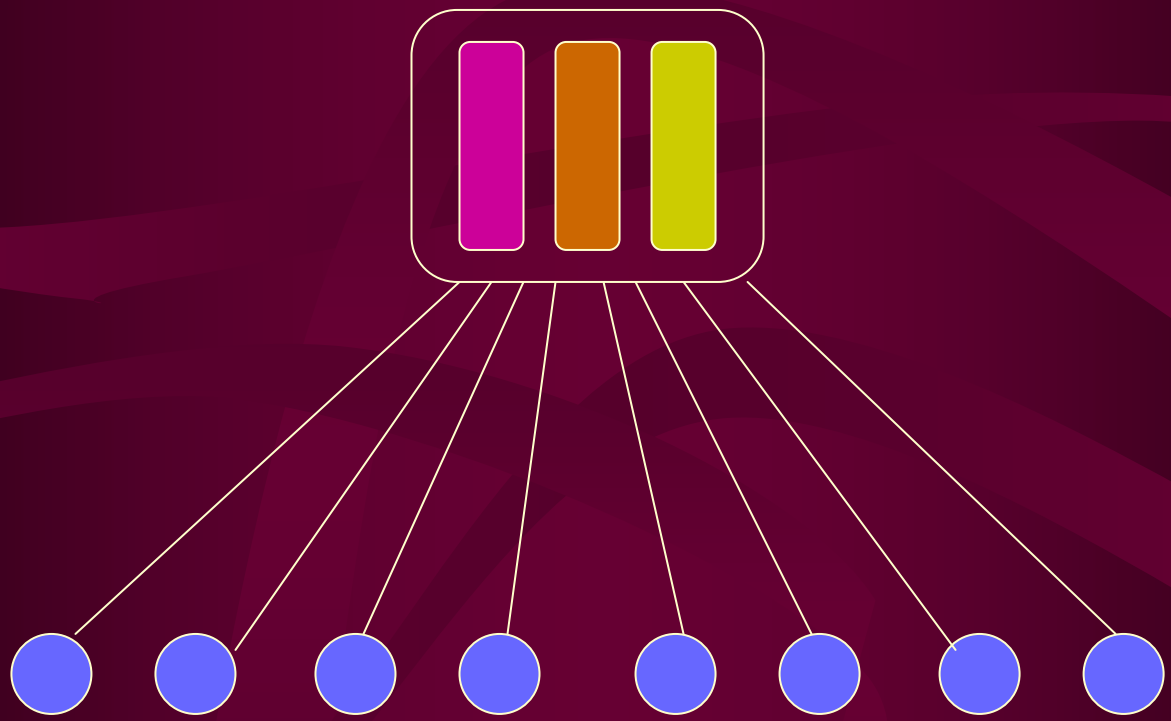
n-Tier モデル　J2EE



サーバサイド・プログラミングのメリット

1-Tier モデル

センターホスト型

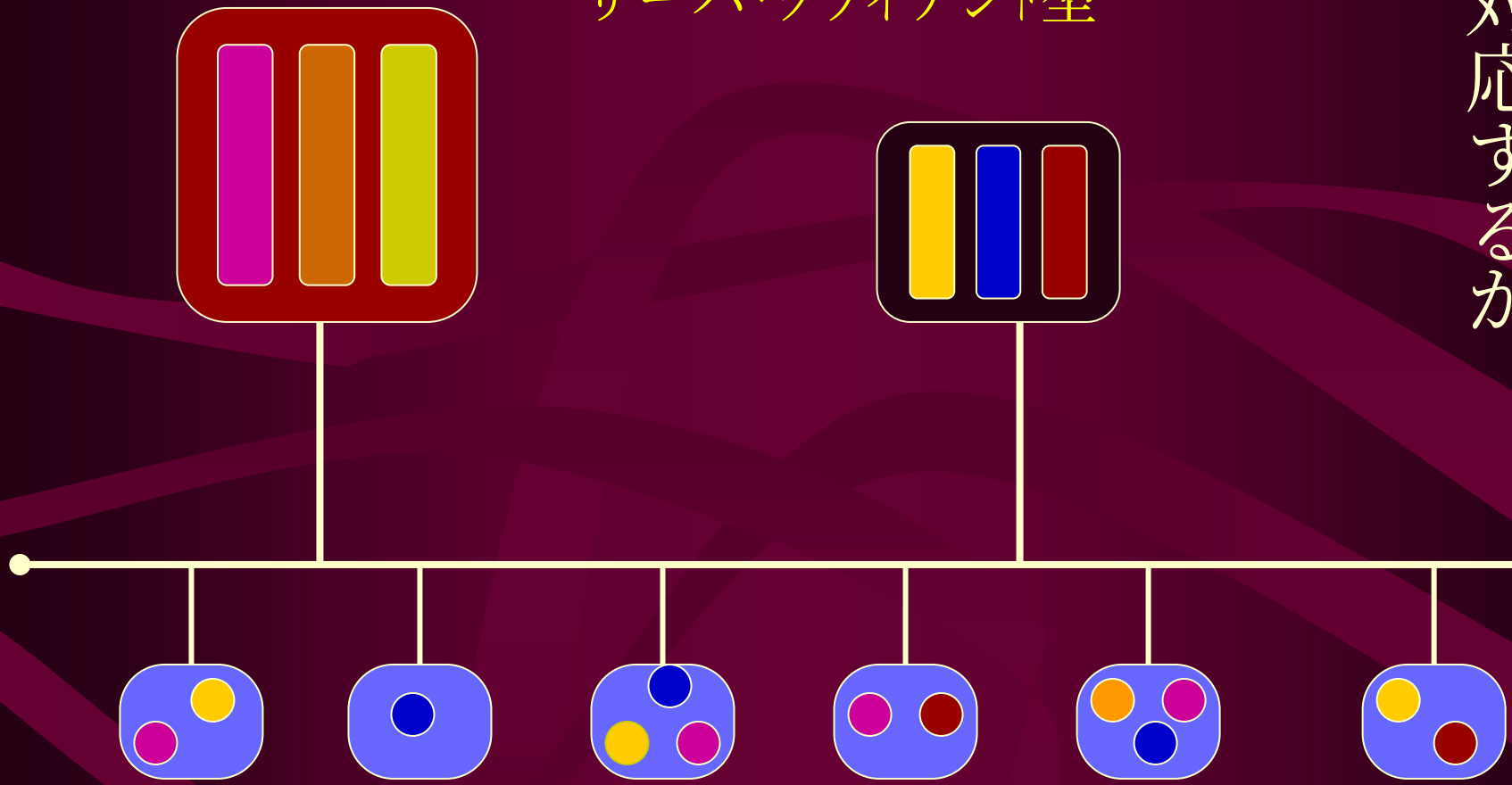


アプリケーションの更新にどう対応するか

ホストの側がアプリケーションを更新すればいい

2-Tier モデル

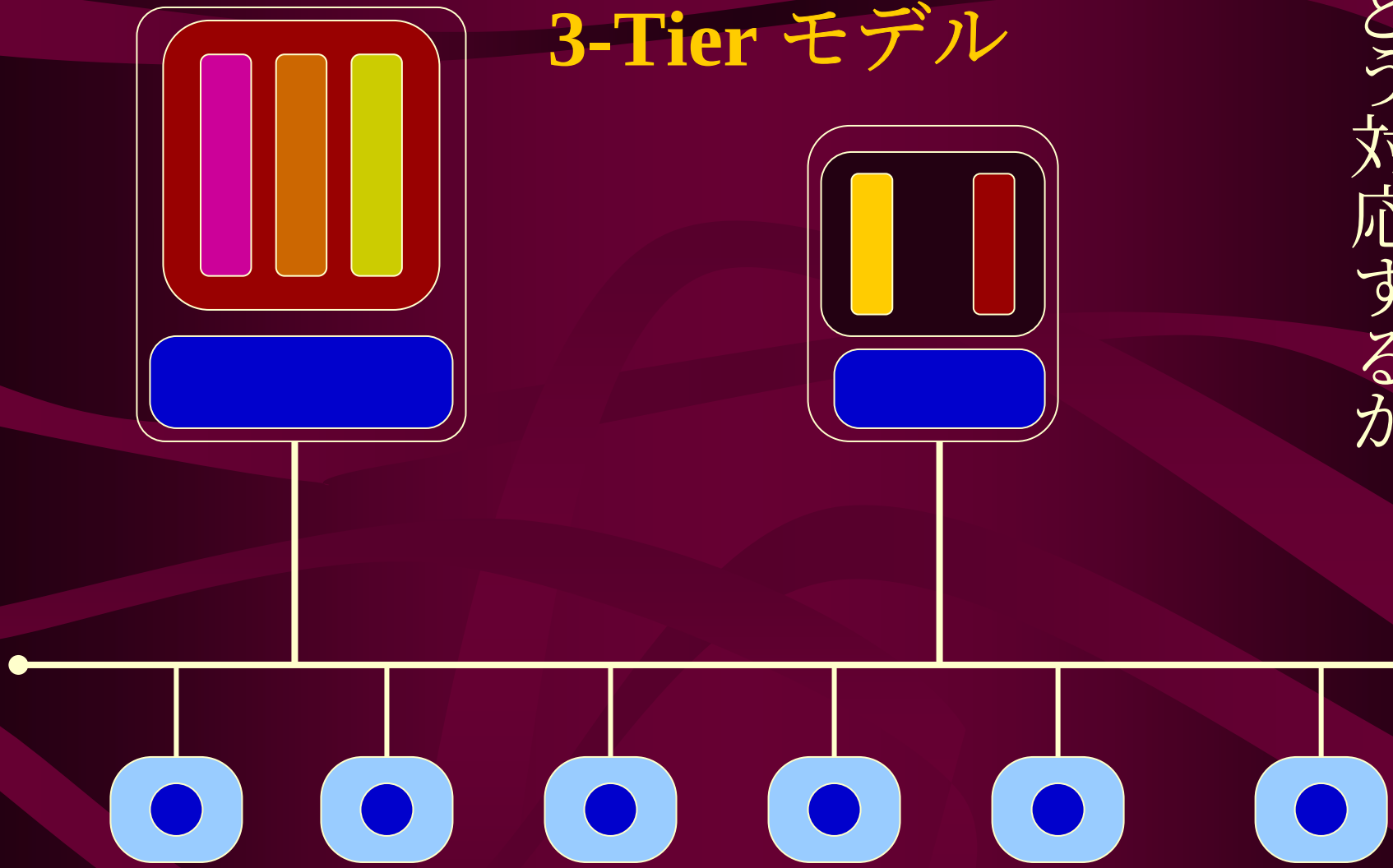
サーバ･クライアント型



アプリケーションの更新にどう対応するか

サーバ･クライアント双方の変更が必要

3-Tier モデル



アプリケーションの更新にどう対応するか

アプリケーション・サーバ側の変更だけでいい

コンポーネント / コンテナ　モデル
Enterprise Java Beanと Application Server

ComponentとContainerの役割

- **Component**
  - 「ビジネス・ロジック」への集中
  - コンポーネントの分離と連携
  - コンポーネントの「再利用」
- 開発者の役割分担
  - Creation / Component Provider
  - Assembly / Application Assenbler
  - Deployment / Deployer
- **Container**
  - ネットワーク接続
  - トランザクション処理
  - ライフサイクル管理
    等の定型的な処理の自動化

J2EEの表面には
見えない土台としての
J2EEコンテナ

コンポーネントのメリット
「ビジネス・ロジック」への集中

```
public class BankBean implements SessionBean {
  public void transferToSaving(double amount) {
      checkingBalance -= amount;
      savingBalance += amount;
      try {
        updateChecking(checkingBalance);
        if (checkingBalance < 0.00) {
          context.setRollbackOnly();
          throw new InsufficientBalanceException();
        }
        updateSaving(savingBalance);
      } catch (SQLException ex) {
         throw new EJBException (……);
      }
   }
```

多様なタイプのコンポーネント
Enterprise Java Bean

- Stateless Session Bean
- Statefull Session Bean
- Entity Bean

J2EE1.2

- Message Driven Bean

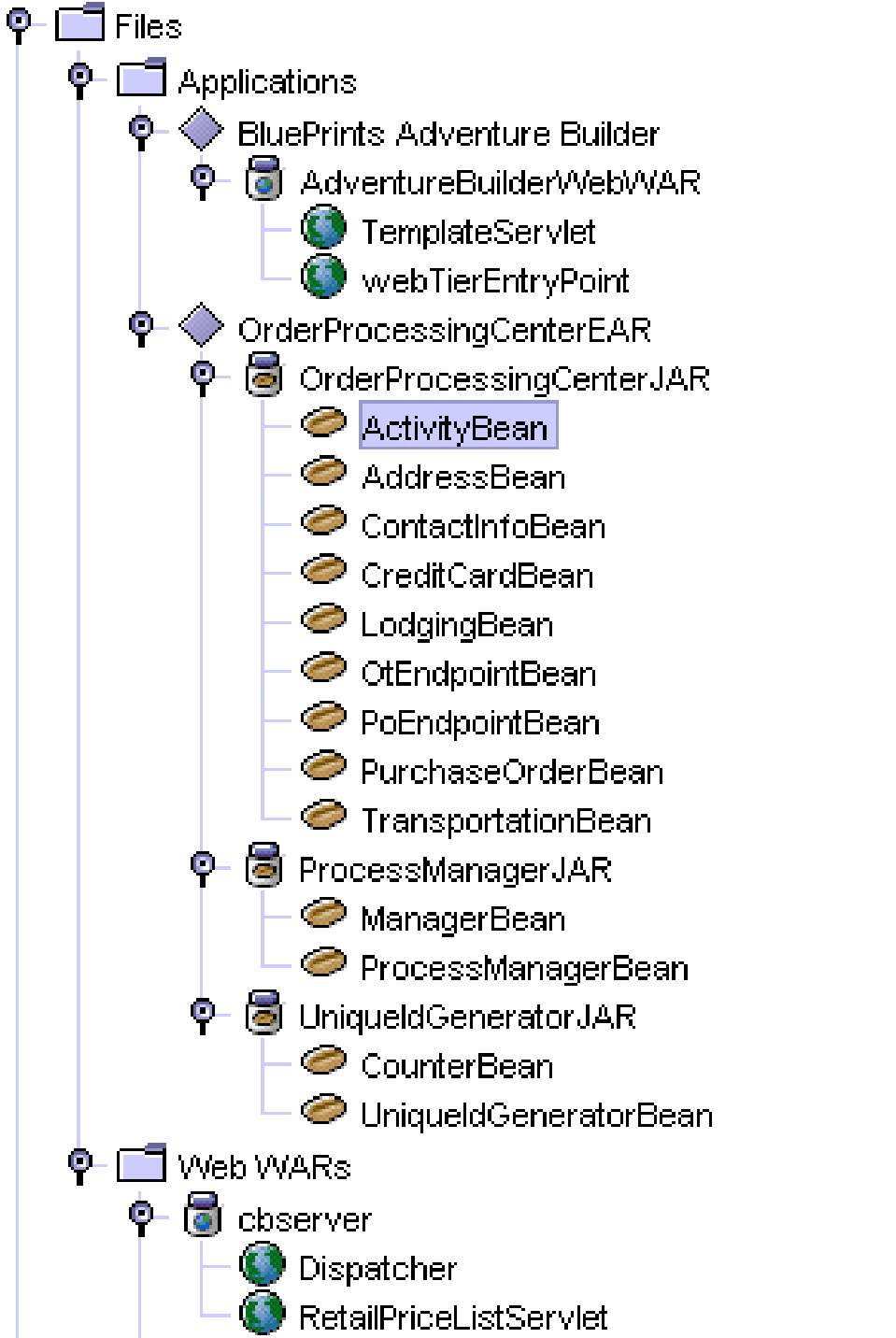
J2EE1.3

- Stateless Session Bean with Web Service Endpoint

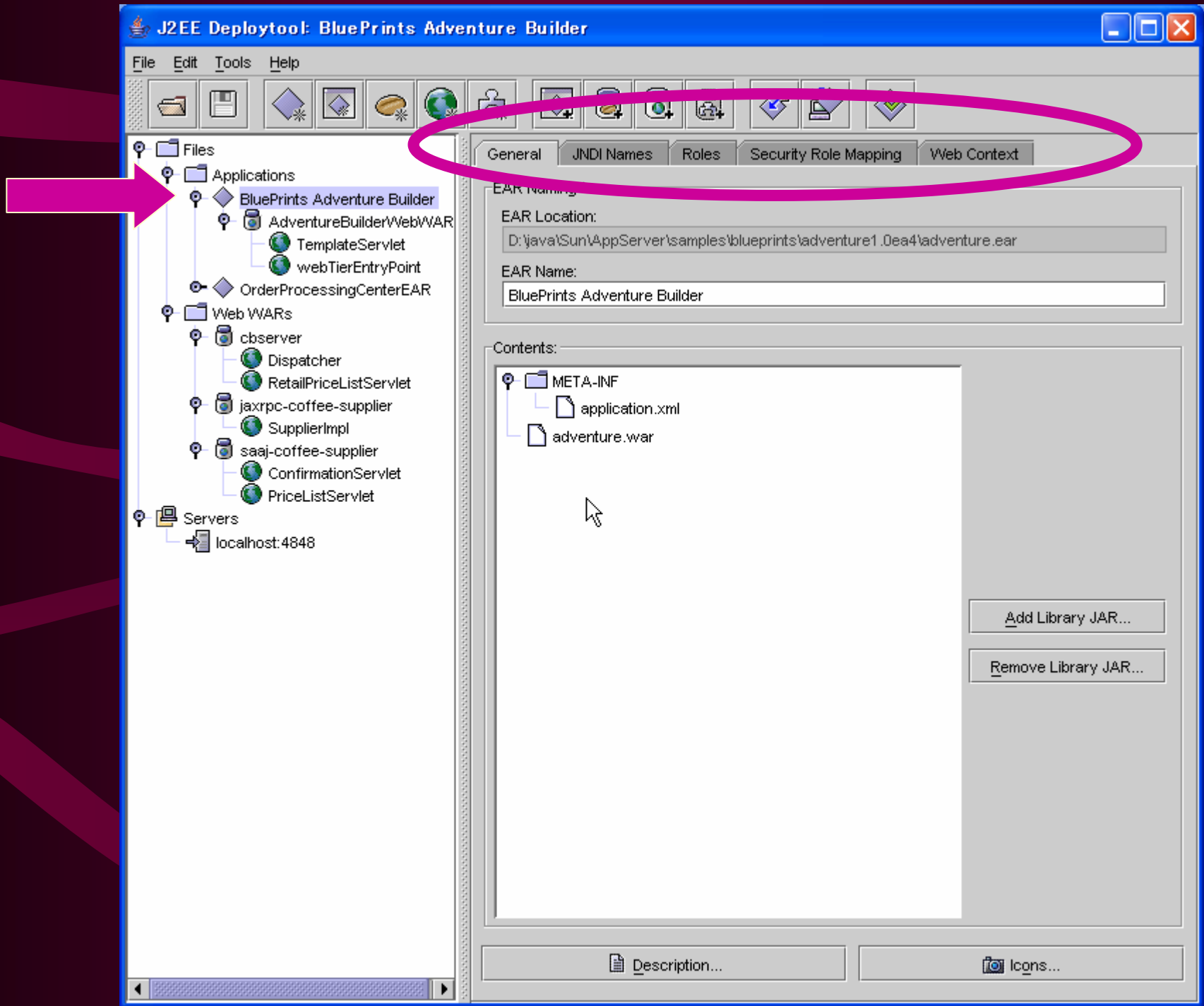
J2EE1.4

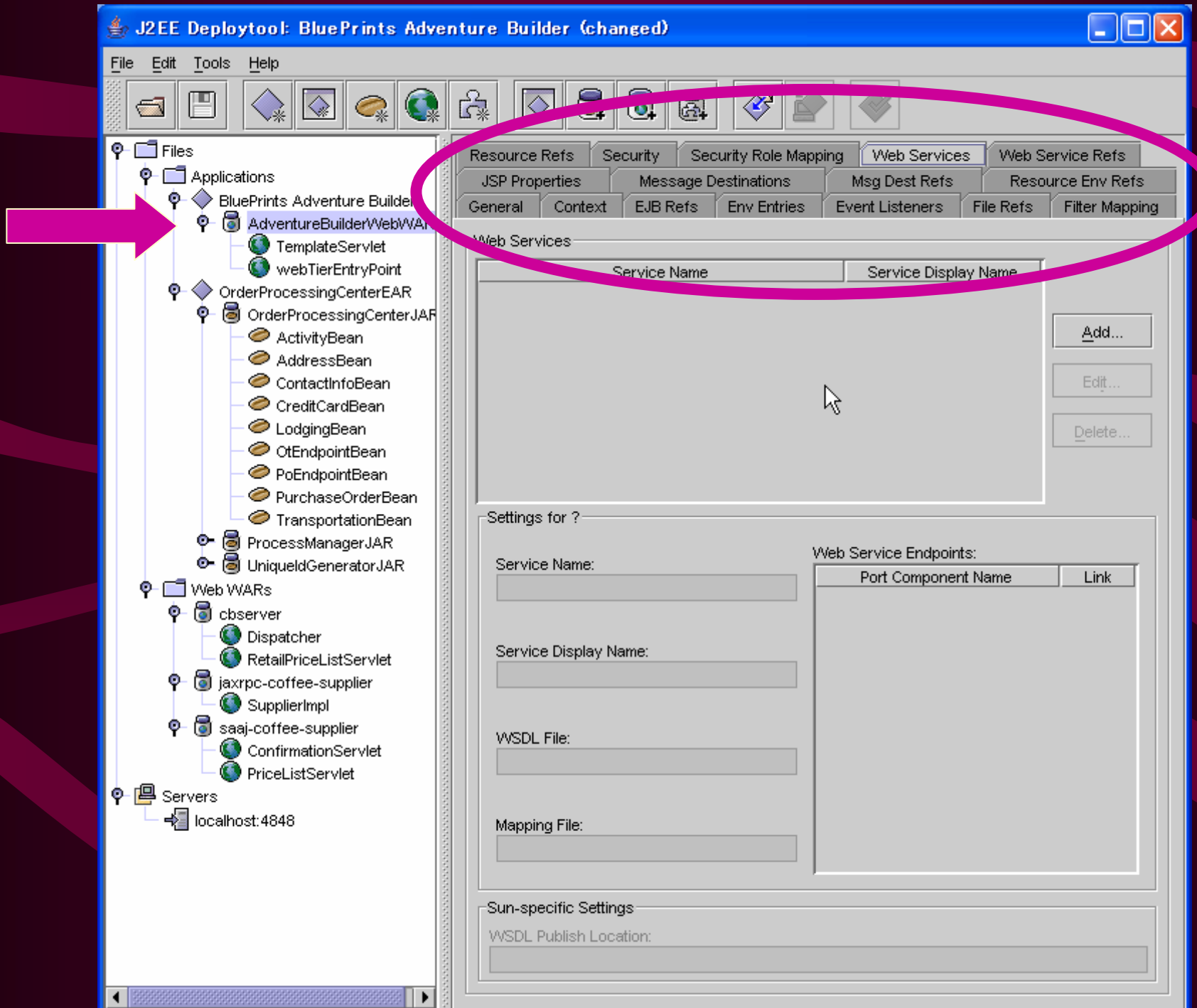
多様なデザイン・パターン
J2EE Design Pattern

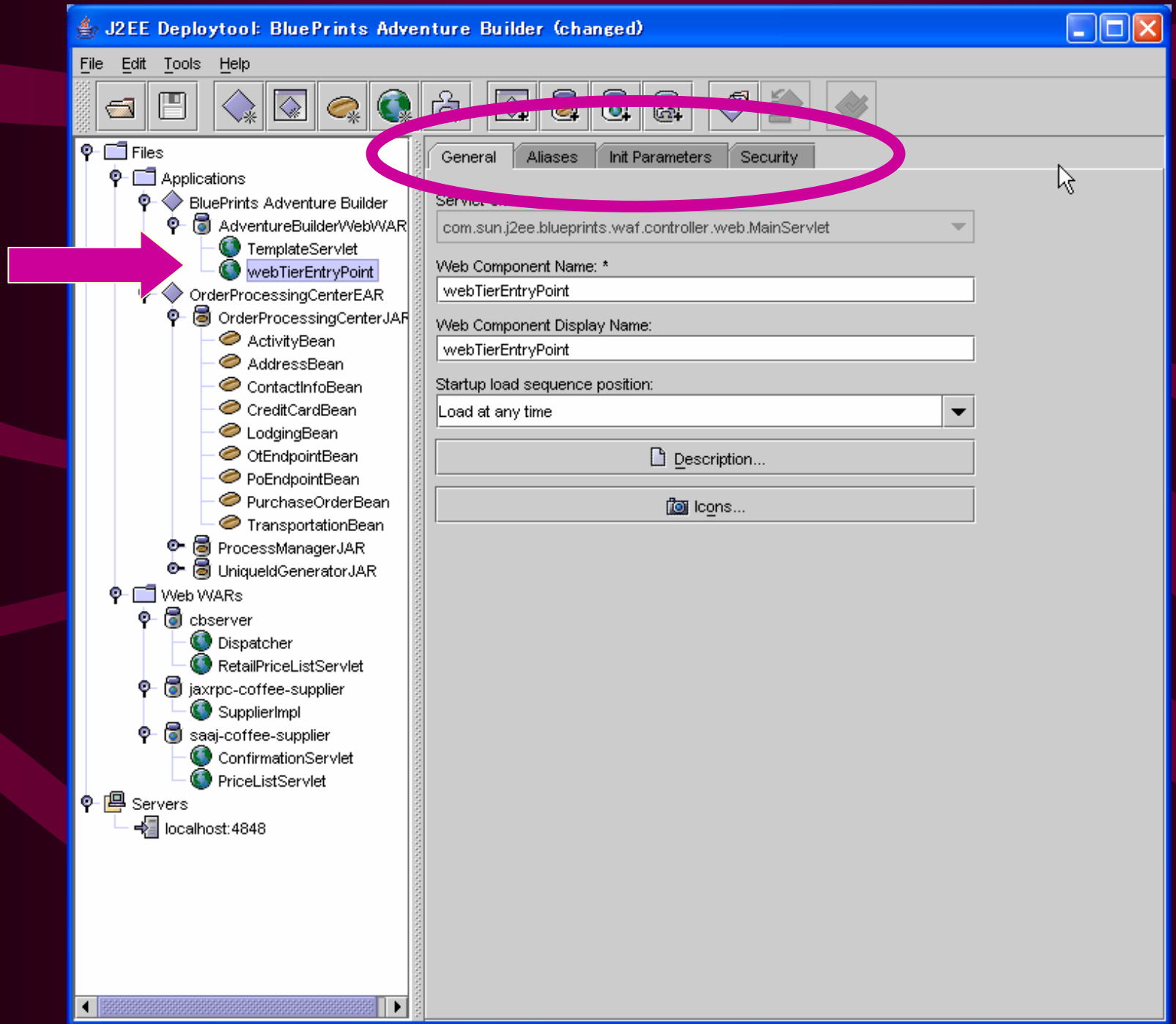
- Front Controller
- Intercepting Filter
- Composite View
- View Helper
- Dispatcher View
- Business Delegate
- Session Façade
- Data Access Object
- Service Locator
- Transfer Object Assembler
- Value List Handler
- Composite Entry
- Transfer Object
- Service Activator
- …….

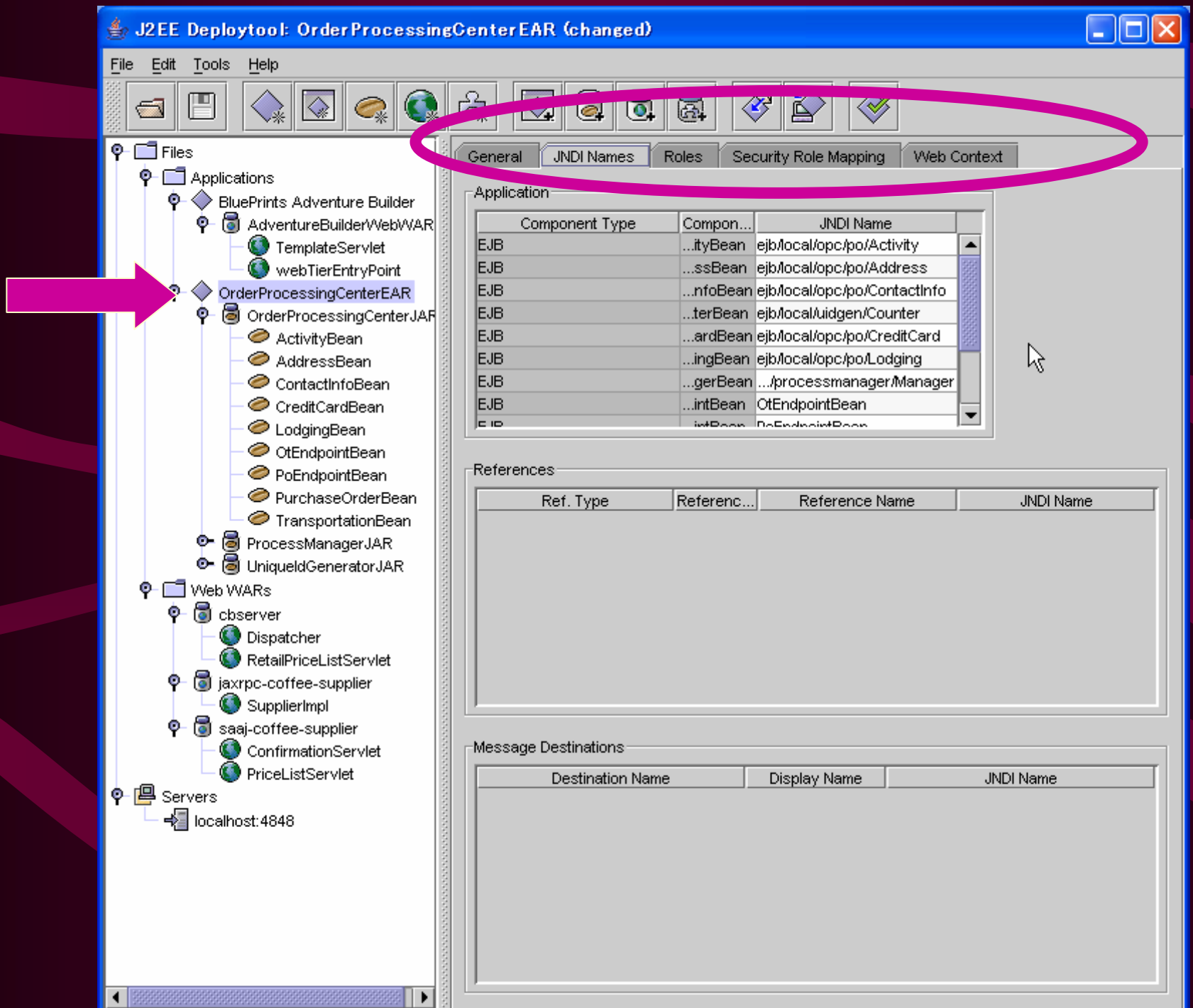


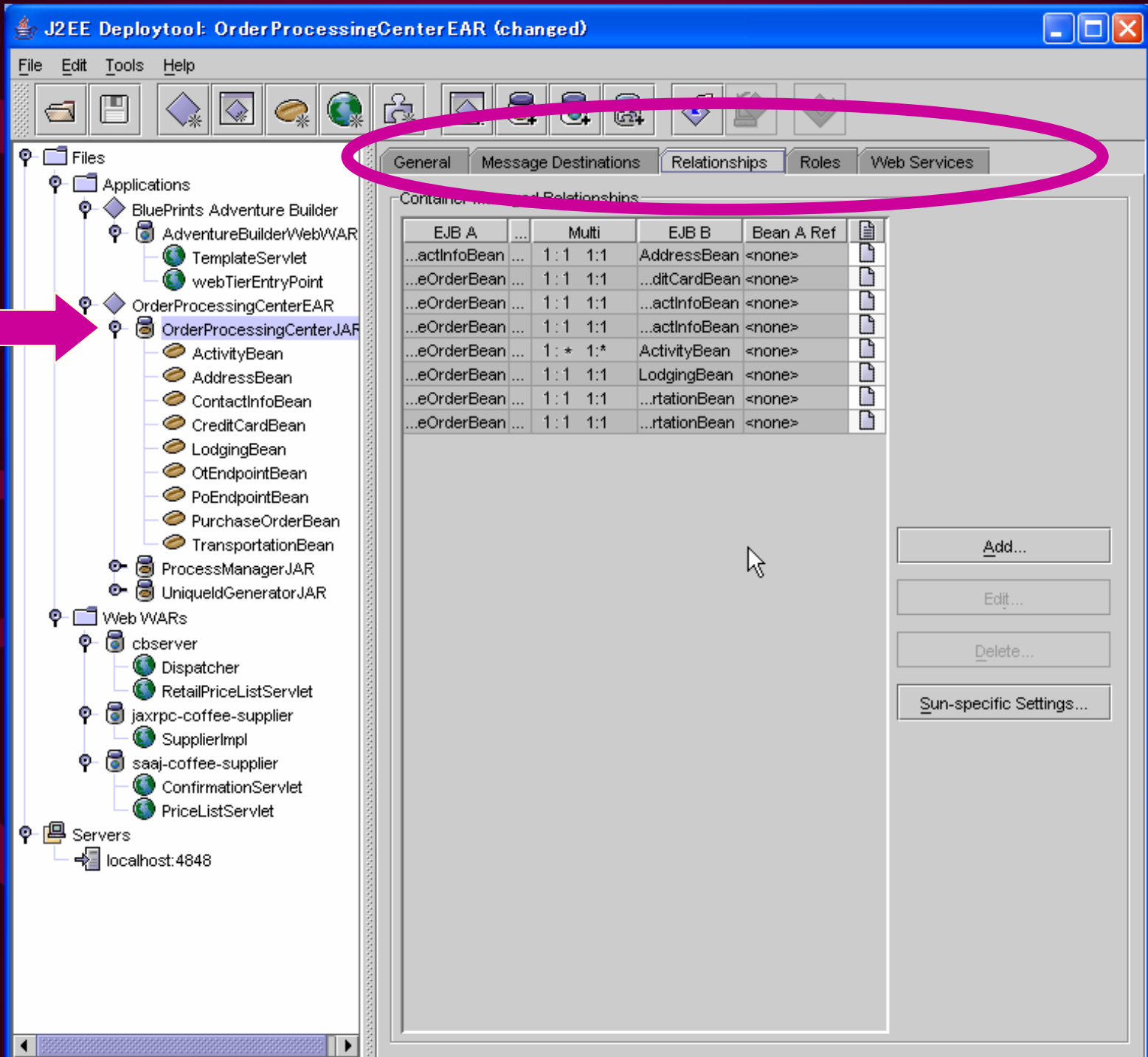
コンポーネントの配備（deploy）と宣言的プログラミング

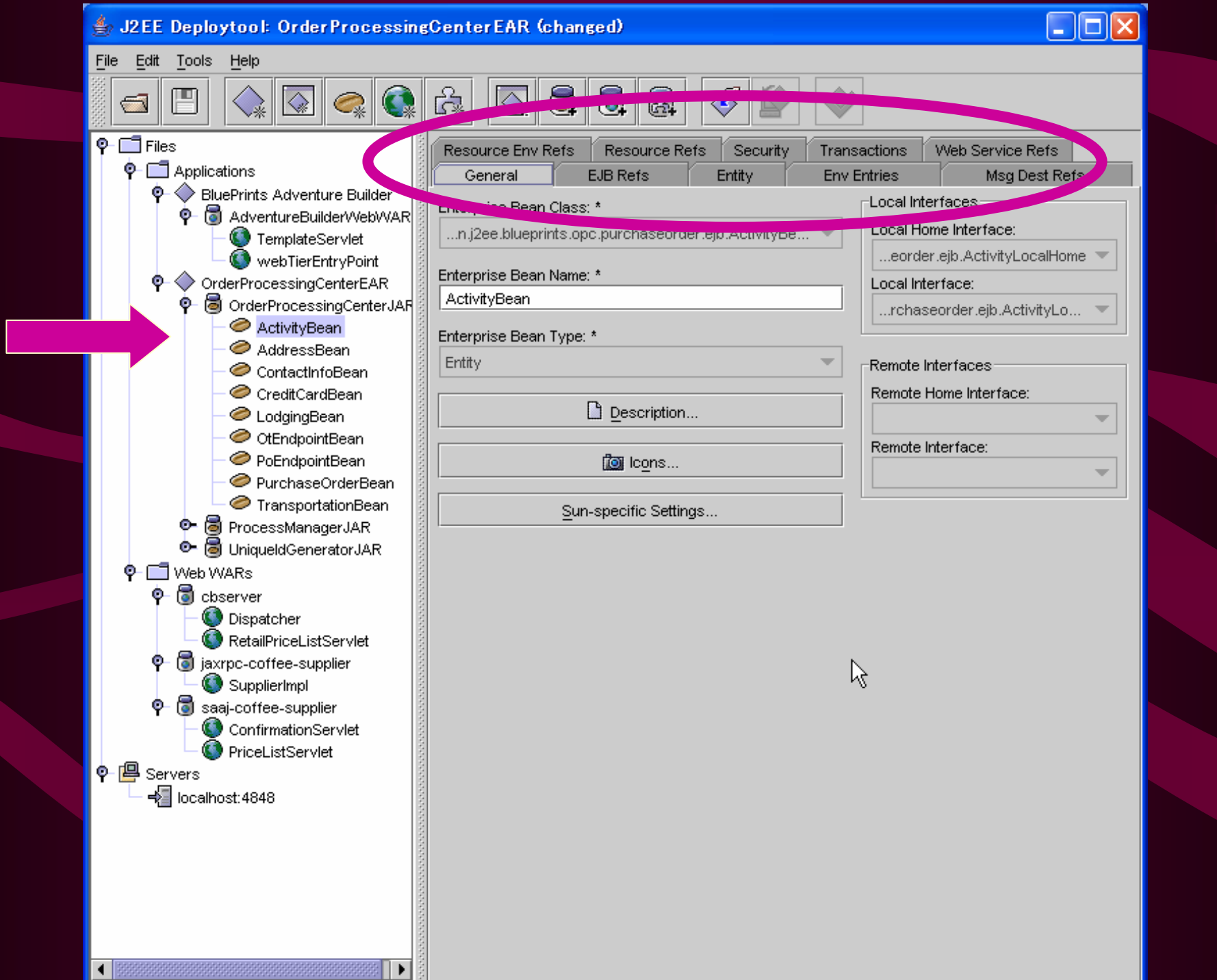

J2EE Deploytool: BluePrints Adventure Builder
File
Edit
Tools
Help
Files
Applications
BluePrints Adventure Builder
AdventureBuilderWebWAR
TemplateServlet
webTierEntryPoint
OrderProcessingCenterEAR
Web WARs
cbserver
Dispatcher
RetailPriceListServlet
jaxrpc-coffee-supplier
SupplierImpl
saaj-coffee-supplier
ConfirmationServlet
PriceListServlet
Servers
localhost:4848
General
JNDI Names
Roles
Security Role Mapping
Web Context
EAR Location:
D:\java\Sun\AppServer\samples\blueprints\adventure1.0ea4\adventure.ear
EAR Name:
BluePrints Adventure Builder
Contents:
META-INF
application.xml
adventure.war
Add Library JAR...
Remove Library JAR...
Description...
Icons...


J2EE Deploytool: BluePrints Adventure Builder (changed)
File Edit Tools Help
Files
Applications
BluePrints Adventure Builder
AdventureBuilderWebWAR
TemplateServlet
webTierEntryPoint
OrderProcessingCenterEAR
OrderProcessingCenterJAR
ActivityBean
AddressBean
ContactInfoBean
CreditCardBean
LodgingBean
OtEndpointBean
PoEndpointBean
PurchaseOrderBean
TransportationBean
ProcessManagerJAR
UniqueIdGeneratorJAR
Web WARs
cbserver
Dispatcher
RetailPriceListServlet
jaxrpc-coffee-supplier
SupplierImpl
saaj-coffee-supplier
ConfirmationServlet
PriceListServlet
Servers
localhost:4848
Resource Refs
Security
Security Role Mapping
Web Services
Web Service Refs
JSP Properties
Message Destinations
Msg Dest Refs
Resource Env Refs
General
Context
EJB Refs
Env Entries
Event Listeners
File Refs
Filter Mapping
Web Services
Service Name
Service Display Name
Add...
Edit...
Delete...
Settings for ?
Service Name:
Service Display Name:
WSDL File:
Mapping File:
Web Service Endpoints:
Port Component Name
Link
Sun-specific Settings
WSDL Publish Location:


J2EE Deploytool: BluePrints Adventure Builder (changed)
File
Edit
Tools
Help
Files
Applications
BluePrints Adventure Builder
AdventureBuilderWebWAR
TemplateServlet
webTierEntryPoint
OrderProcessingCenterEAR
OrderProcessingCenterJAR
ActivityBean
AddressBean
ContactInfoBean
CreditCardBean
LodgingBean
OtEndpointBean
PoEndpointBean
PurchaseOrderBean
TransportationBean
ProcessManagerJAR
UniqueIdGeneratorJAR
Web WARs
cbserver
Dispatcher
RetailPriceListServlet
jaxrpc-coffee-supplier
SupplierImpl
saaj-coffee-supplier
ConfirmationServlet
PriceListServlet
Servers
localhost:4848
General
Aliases
Init Parameters
Security
com.sun.j2ee.blueprints.waf.controller.web.MainServlet
Web Component Name: *
webTierEntryPoint
Web Component Display Name:
webTierEntryPoint
Startup load sequence position:
Load at any time
Description...
Icons...


J2EE Deploytool: OrderProcessingCenterEAR (changed)
File Edit Tools Help
Files
Applications
BluePrints Adventure Builder
AdventureBuilderWebWAR
TemplateServlet
webTierEntryPoint
OrderProcessingCenterEAR
OrderProcessingCenterJAR
ActivityBean
AddressBean
ContactInfoBean
CreditCardBean
LodgingBean
OtEndpointBean
PoEndpointBean
PurchaseOrderBean
TransportationBean
ProcessManagerJAR
UniqueIdGeneratorJAR
Web WARs
cbserver
Dispatcher
RetailPriceListServlet
jaxrpc-coffee-supplier
SupplierImpl
saaj-coffee-supplier
ConfirmationServlet
PriceListServlet
Servers
localhost:4848
General
JNDI Names
Roles
Security Role Mapping
Web Context
Application
Component Type
Compon...
JNDI Name
EJB ...ityBean ejb/local/opc/po/Activity
EJB ...ssBean ejb/local/opc/po/Address
EJB ...nfoBean ejb/local/opc/po/ContactInfo
EJB ...terBean ejb/local/uidgen/Counter
EJB ...ardBean ejb/local/opc/po/CreditCard
EJB ...ingBean ejb/local/opc/po/Lodging
EJB ...gerBean .../processmanager/Manager
EJB ...intBean OtEndpointBean
References
Ref. Type
Referenc...
Reference Name
JNDI Name
Message Destinations
Destination Name
Display Name
JNDI Name

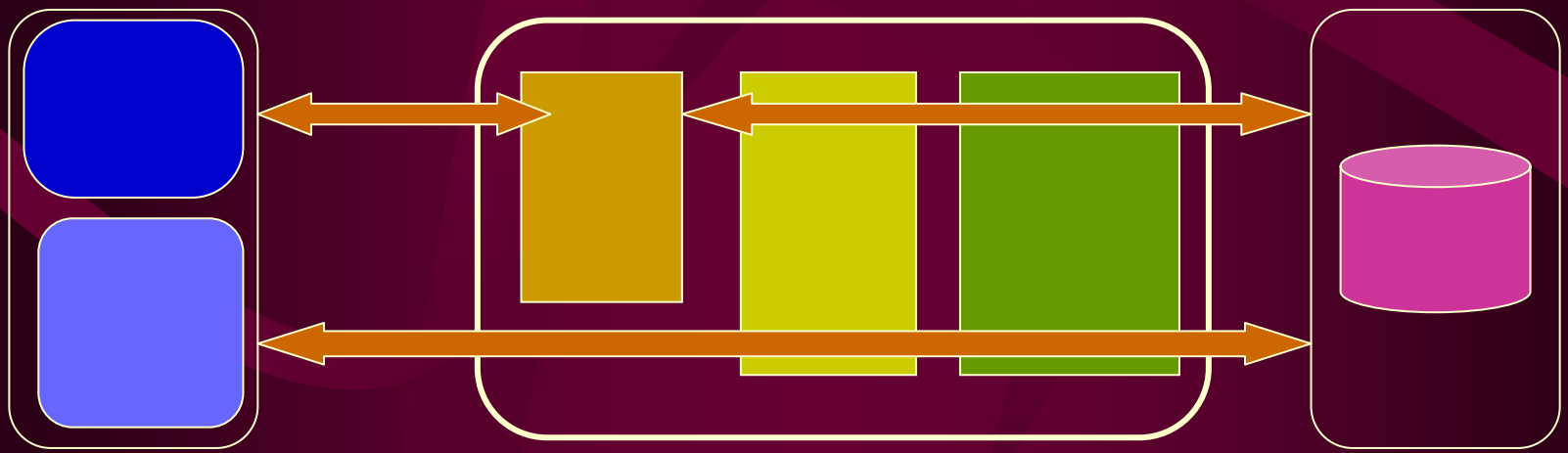

J2EE Deploytool: OrderProcessingCenterEAR (changed)
File
Edit
Tools
Help
Files
Applications
BluePrints Adventure Builder
AdventureBuilderWebWAR
TemplateServlet
webTierEntryPoint
OrderProcessingCenterEAR
OrderProcessingCenterJAR
ActivityBean
AddressBean
ContactInfoBean
CreditCardBean
LodgingBean
OtEndpointBean
PoEndpointBean
PurchaseOrderBean
TransportationBean
ProcessManagerJAR
UniqueIdGeneratorJAR
Web WARs
cbserver
Dispatcher
RetailPriceListServlet
jaxrpc-coffee-supplier
SupplierImpl
saaj-coffee-supplier
ConfirmationServlet
PriceListServlet
Servers
localhost:4848
General
Message Destinations
Relationships
Roles
Web Services
EJB A
Multi
EJB B
Bean A Ref
...actInfoBean 1 : 1 1:1 AddressBean <none>
...eOrderBean 1 : 1 1:1 ...ditCardBean <none>
...eOrderBean 1 : 1 1:1 ...actInfoBean <none>
...eOrderBean 1 : 1 1:1 ...actInfoBean <none>
...eOrderBean 1 : * 1:* ActivityBean <none>
...eOrderBean 1 : 1 1:1 LodgingBean <none>
...eOrderBean 1 : 1 1:1 ...rtationBean <none>
...eOrderBean 1 : 1 1:1 ...rtationBean <none>
Add...
Edit...
Delete...
Sun-specific Settings...


J2EE Deploytool: OrderProcessingCenterEAR (changed)
File
Edit
Tools
Help
Files
Applications
BluePrints Adventure Builder
AdventureBuilderWebWAR
TemplateServlet
webTierEntryPoint
OrderProcessingCenterEAR
OrderProcessingCenterJAR
ActivityBean
AddressBean
ContactInfoBean
CreditCardBean
LodgingBean
OtEndpointBean
PoEndpointBean
PurchaseOrderBean
TransportationBean
ProcessManagerJAR
UniqueIdGeneratorJAR
Web WARs
cbserver
Dispatcher
RetailPriceListServlet
jaxrpc-coffee-supplier
SupplierImpl
saaj-coffee-supplier
ConfirmationServlet
PriceListServlet
Servers
localhost:4848
Resource Env Refs
Resource Refs
Security
Transactions
Web Service Refs
General
EJB Refs
Entity
Env Entries
Msg Dest Refs
Enterprise Bean Class: *
...n.j2ee.blueprints.opc.purchaseorder.ejb.ActivityBe...
Enterprise Bean Name: *
ActivityBean
Enterprise Bean Type: *
Entity
Description...
Icons...
Sun-specific Settings...
Local Interfaces
Local Home Interface:
...eorder.ejb.ActivityLocalHome
Local Interface:
...rchaseorder.ejb.ActivityLo...
Remote Interfaces
Remote Home Interface:
Remote Interface:

J2EEを構成する要素技術

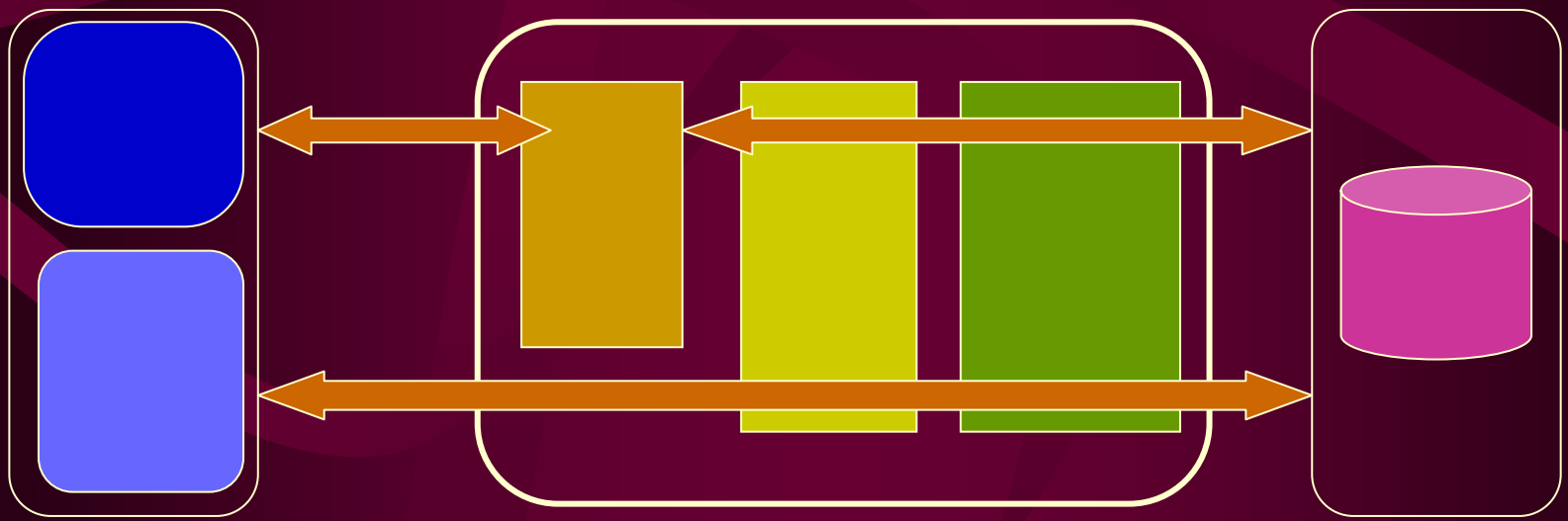
J2EEを構成する要素技術 (1)

- **Enterprise JavaBeans Technology**
- **JDBC™ API**
- **Java Servlet Technology**
- **JavaServer Pages (JSP) Technology**
- **Java Message Service (JMS)**
- **Java Transaction API (JTA)**



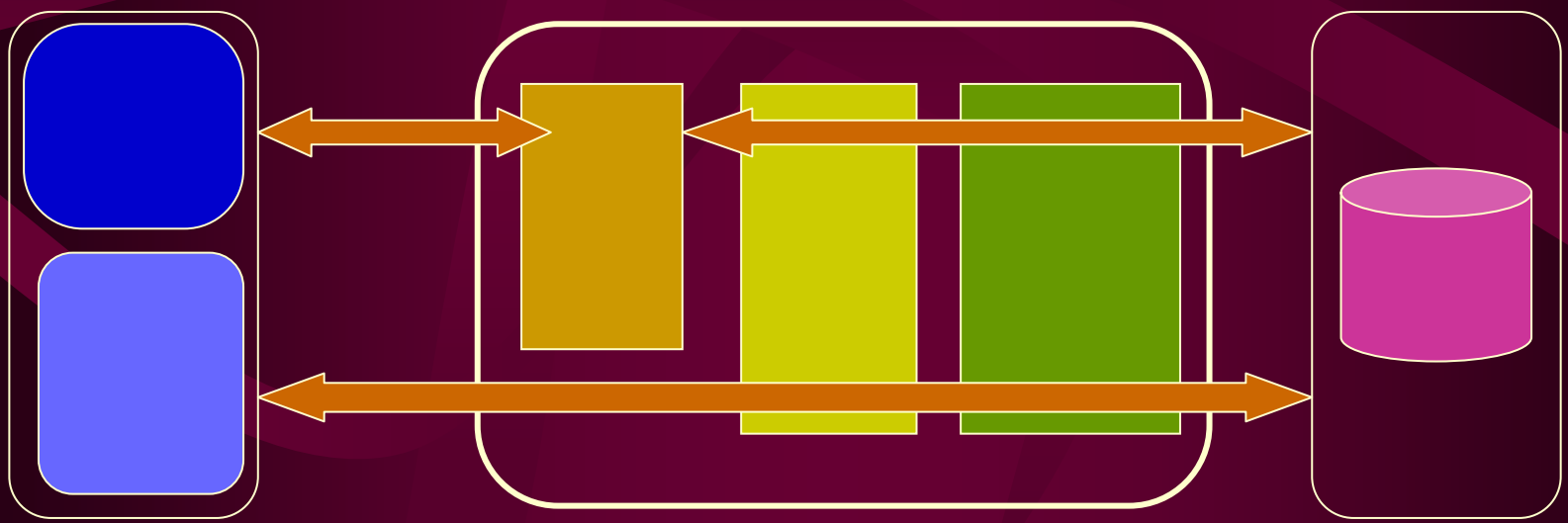
J2EEを構成する要素技術 (2)

- **JavaMail™ Technology**
- **JavaBeans Activation Framework**
- **Java API for XML Processing (JAXP)**
- **J2EE Connector Architecture**
- **Java Authentication and Authorization Service (JAAS)**



J2EEを構成する要素技術 (3)

- **Web Service**
- **Java APIs for XML based RPC(JAX-RPC)**
- **Java APIs for XML Messaging(JAXM)**
- **Java APIs for XML Registry (JAXR)**
- **Java Server Faces**
- **……**



2.　EJBとコンテナ

- **J2EEのContainerの役割**
- **EJBの3種類の定義ファイル**
- **J2EE Containerのメカニズムを考える**
  -- **EJBの3種類の定義ファイルの問題** --
- **EJB : Entity Bean の役割**

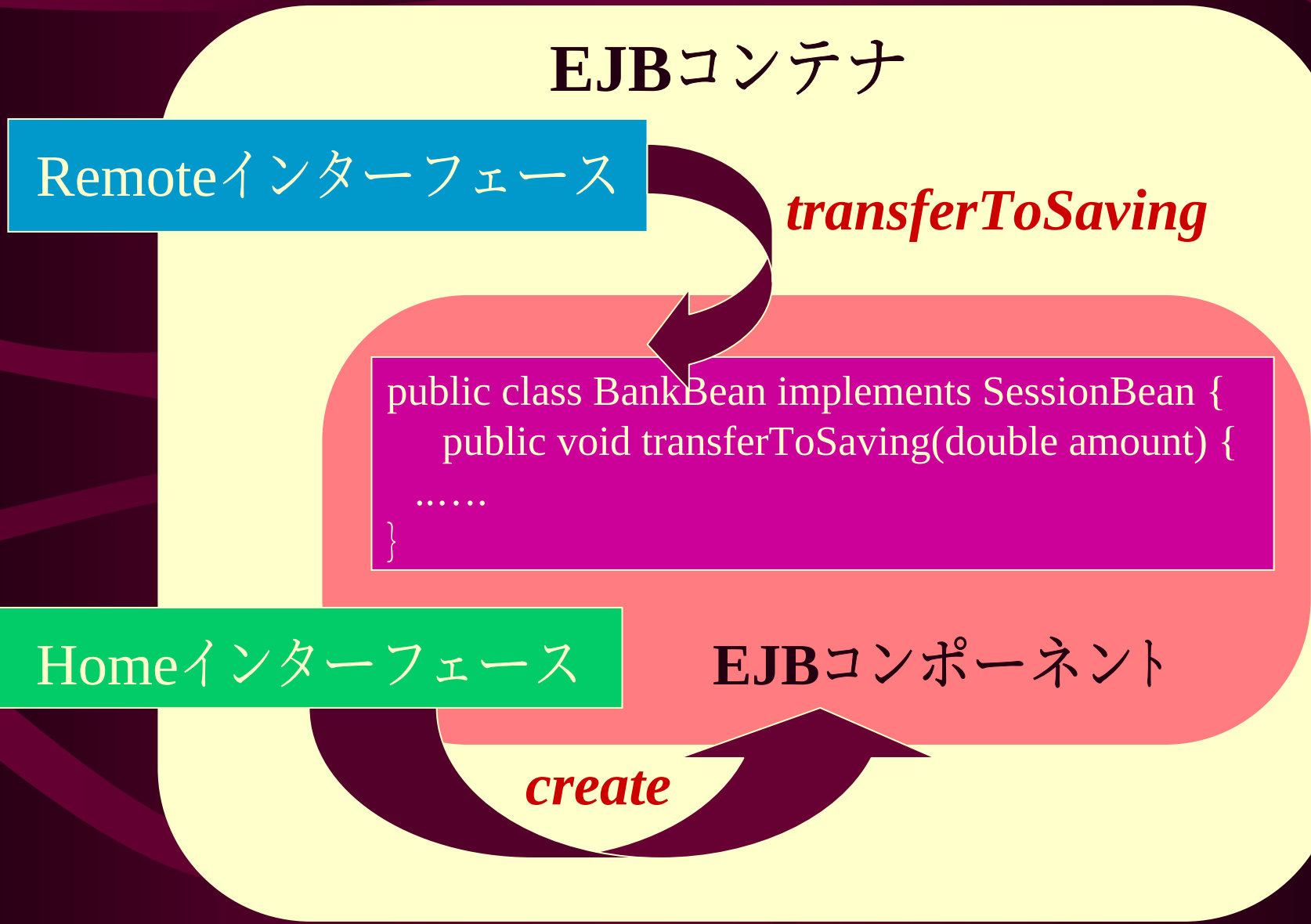
- **J2EE1.3 EJB2.0での永続性管理の自動化**
- **J2EE1.3 EJB2.0 CMPサンプル**
- **J2EE1.3 JMSメッセージングの導入**

J2EEのContainerの役割

J2EEでのコンテナのメタファー

- コンテナは、コンポーネントの「いれもの」。
  コンテナは、最初は空。
- コンテナは、外部との境界に、**Home**と**Remote**という二つのインターフェースを持っている。
- **Home**インターフェースを通じて、コンテナにコンポーネントを作ったり、消したりすることが可能。
- コンポーネント内のメソッドは、**EJB**クラスで定義され、**Remote**インターフェースを通じて呼び出される。

EJB コンテナとEJBコンポーネント



EJBの3種類の定義ファイル

EJBの3種類の定義ファイル

Remoteインターフェース： Bank.java

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

Homeインターフェース： BankHome.java

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

EJBクラス： BankEJB.java

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

EJBの三つの定義ファイル(1)

- Remote**インターフェース**

```
public interface Hello extends javax.ejb.EJBObject {
        public String sayHello() throws
                        java.rmi.RemoteException;
}
```

EJB実装クラスの外部へのインターフェース

EJBの三つの定義ファイル(2)

- **Home**インターフェース　(**これも**Remote**である**)
  コンテナへのインターフェース

```
public interface HelloHome
        extends javax.ejb.EJBHome {
    public Hello create() throws
            java.rmi.RemoteException,
            javax.ejb.CreateException;
}
```

*Homeインターフェースの*create*メソッドは,*
Remote インターフェースを返す。

EJBの三つの定義ファイル(3)

- EJBクラス

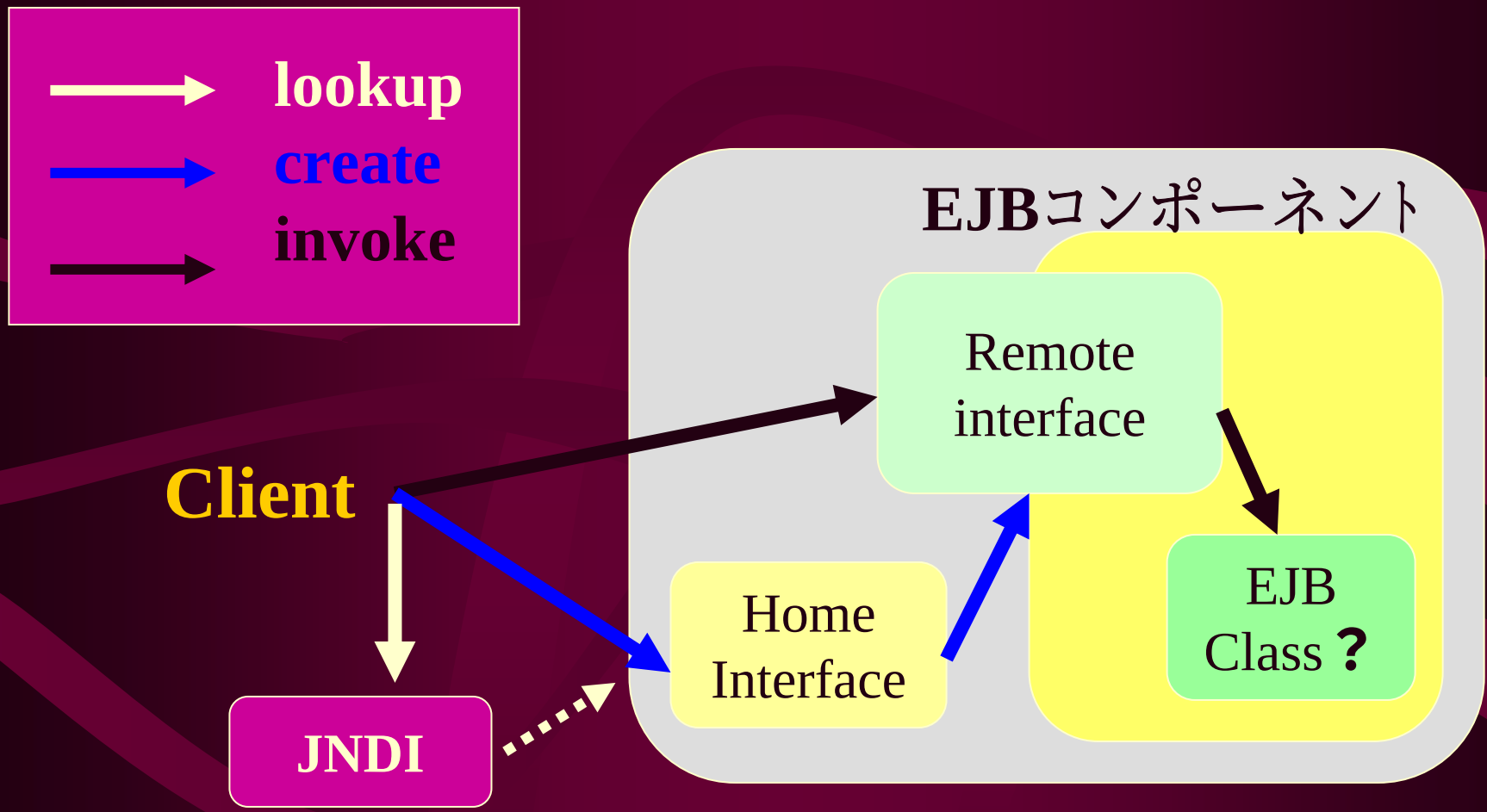
```
public class HelloEJB  implements javax.ejb.SessionBean {
        public String sayHello() {
            return  "Hello World!";
        }
        public void ejbCreate() {}
        public void setSessionContext(SessionContext sc) {}
        public void ejbRemove() {}
        ……
}
```

EJBの実装クラス

クライアントからの呼び出し

1. **JNDIでHome**インスタンスを獲得する。
2. **Home**インスタンス上で、**create**メソッドを呼び出してコンテナ内にコンポーネントを生成する。
3. **create**の返り値が**remote**インターフェースを実装したコンポーネントのインスタンスである。
4. コンポーネントのインスタンス上で、**remote**インターフェースのメソッドを**invoke**する。

Clientから見たJ2EEのコンテナとEJBの三つの定義ファイル



J2EE Containerのメカニズムを考える

-- EJBの3種類の定義ファイルの問題 --

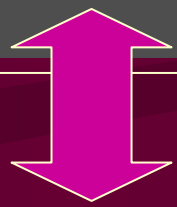
EJBの三つの定義ファイルの問題　1

- 二つのRemoteインターフェースがあるが、クラスは一つしかない。
- 二つのRemoteインターフェースを実装したクラスは、どこに定義されているか？

二つのインターフェースに対応する、実装クラスは、自動的に生成されている。

Homeインターフェース

```
public interface HelloHome extends javax.ejb.EJBHome {
        public Hello create() throws
                        java.rmi.RemoteException, …….;
}
```

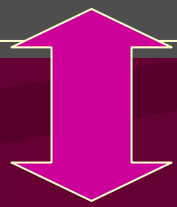


RMI/IIOPでの実装

```
public class HelloHomeImpl extends
        PortableRemoteObject implements HelloHome {
    public Hello create() throws RemoteException,….. {
        return  (Hello)…….. ;
    }
    ……
}
```

Remoteインターフェース

```
public interface Hello extends javax.ejb.EJBObject {
        public String sayHello() throws
                        java.rmi.RemoteException;
}
```



RMI/IIOPでの実装

```
public class HelloEJB_EJBObjectImpl extends
        PortableRemoteObject implements Hello {
    public String sayHello() throws RemoteException {
        return  “Hello World!”;
    }
    ……
}
```

これでは駄目！

EJBクラスの役割は?

EJBの三つの定義ファイルの問題　2

- EJBクラスは、Remoteインターフェースを持っていない。
- RemoteインターフェースのsayHelloメソッドを実装したように見えるEJBクラスは、形式的には、そうなっていないのは何故か？

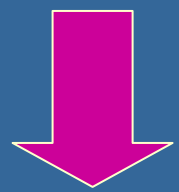
Remoteの実装クラスは、EJBクラスから生成されている

EJBクラス

```
public class HelloEJB  implements javax.ejb.SessionBean {
        public String sayHello() {
            return  "Hello World!";
        }
```

javax.ejb.SessionBeanはRemoteか？

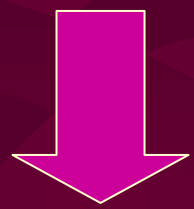
```
public interface SessionBean  extends EnterpriseBean {
    .........
}
```



```
public interface EnterpriseBean  extends Serializable {
    .........
}
```

EJBの三つの定義ファイルの問題　3

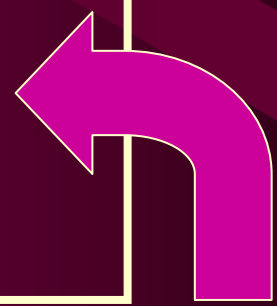
- **Remote**インターフェースの**sayHello**メソッドを、**EJB**クラスの**sayHello**メソッドが実装したものでないなら、二つの**sayHello**メソッドはどういう関係なのか？



- **EJB**クラスのメソッドは、自動生成された**Remote**インターフェースの実装クラスの同じ名前のメソッドの中で、そのまま利用される。

同名メソッドの書き換え

```
public String sayHello() throws RemoteException   {
// Remoteメソッドである。
      .............
// 元のEJBクラスのインスタンス
      HelloEJB helloejb = (HelloEJB) ..... ;
      .............
// ejbLoad等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();
// ejbStore等メソッド呼び出しの後処理
      container.postInvoke(...);
      .............
      return s;
```



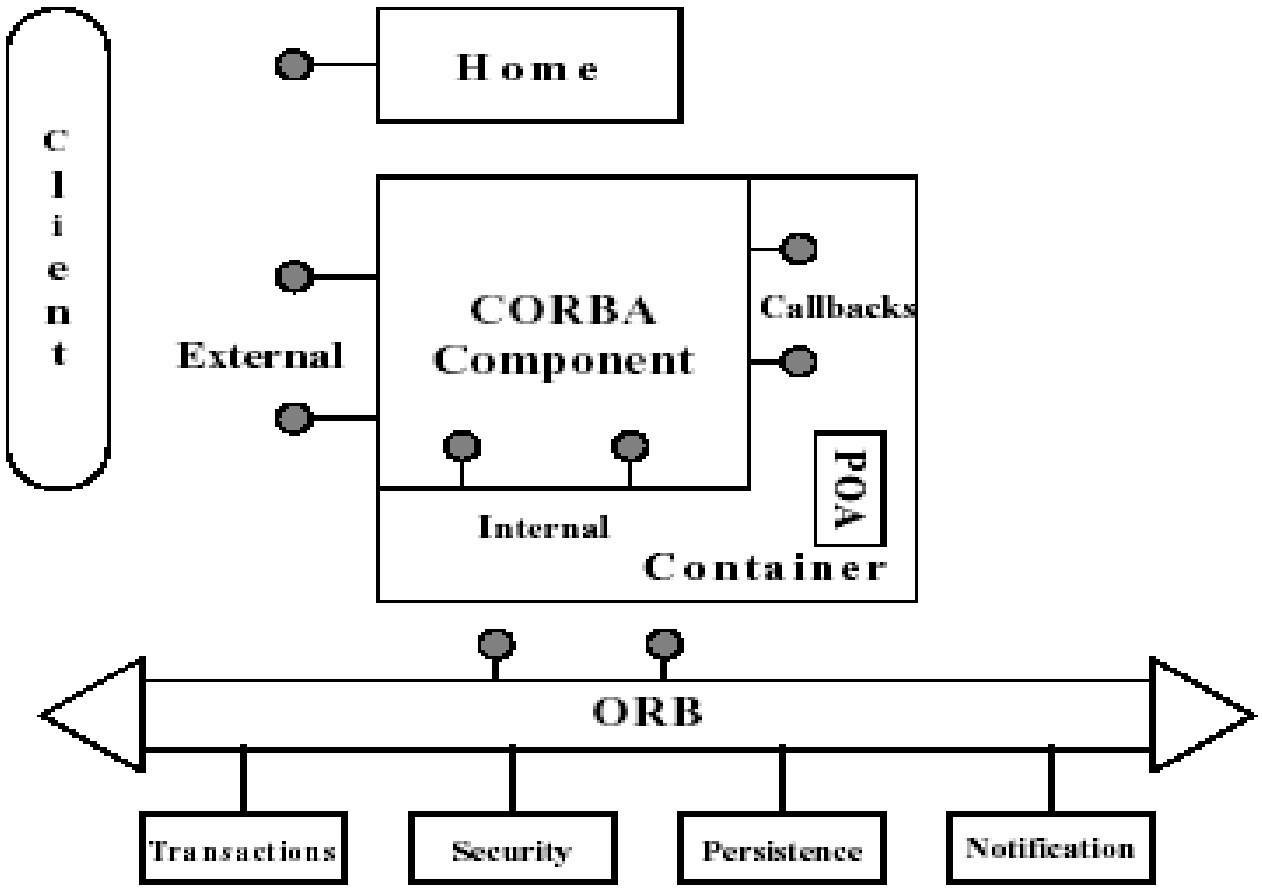
EJBクラスのメソッドのはさみ込み

J2EE コンテナの特徴的手法

- ソースからの別のソースの生成
  - **CORBA : IDL**
  - **J2EE : Java Interface**
  - **JAX-RPC : WSDL / Java Interface**
  - **GT3 OGSI : GWSDL**
- メソッドの書き換え
- 元のメソッドの挟み込み
  - **Transaction**
  - **Database との同期**
  - **Life Cycle 管理**

CORBA Container

The overall architecture is depicted in Figure 4-1.



- http://www.omg.org/docs/formal/02-06-65.pdf

EJB : Entity Bean の役割

Entity Beanの発想

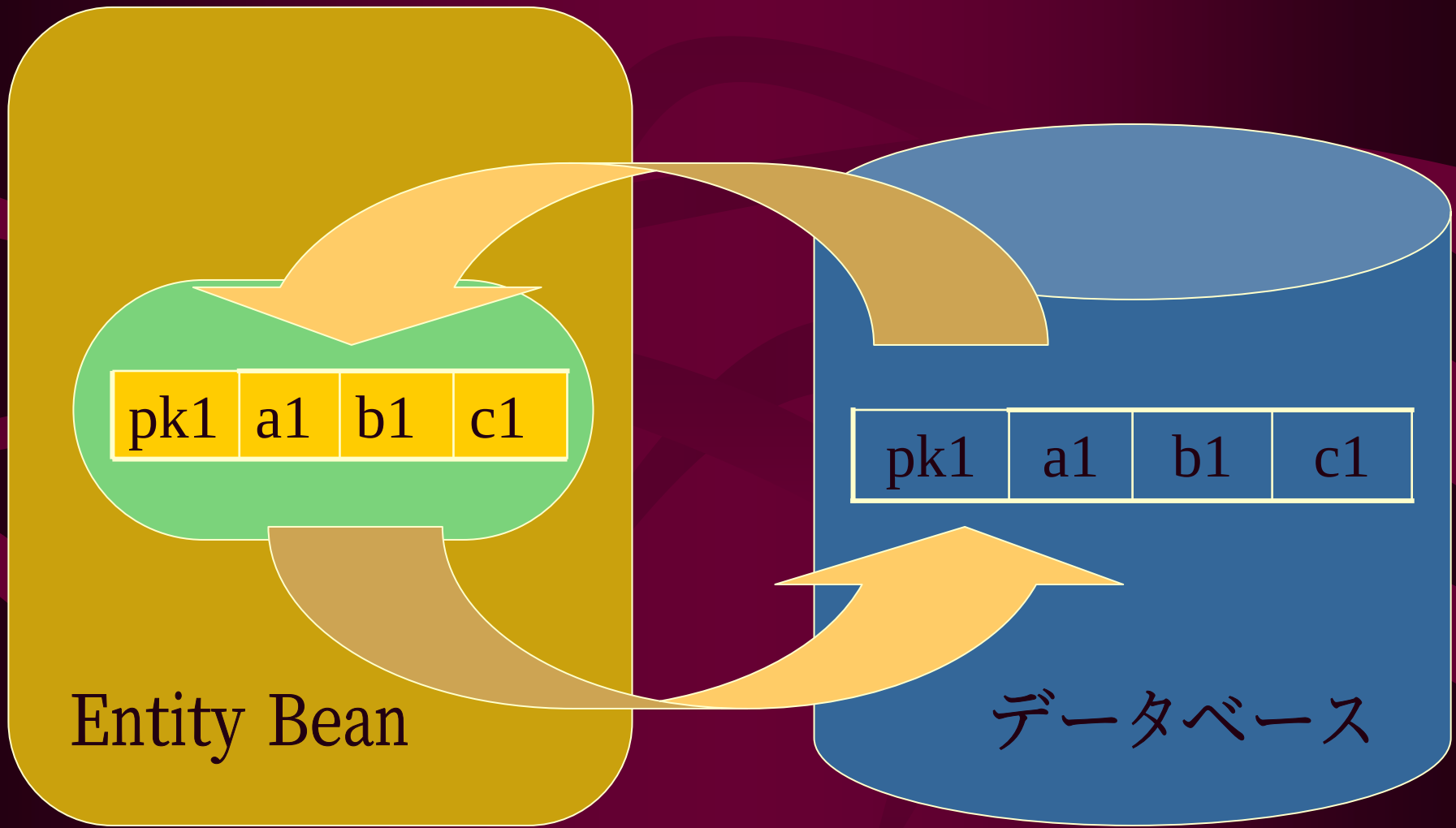
メモリー上の一時的な情報である
Entity Bean インスタンスと

メモリー外部の永続する情報である
データベースの一行とを

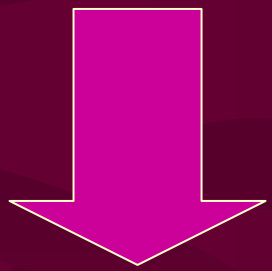
システマティックに結びつける

一時的な情報

永続する情報



Entity Bean インスタンスと
データベースの一行とが
システマティックに結びついている



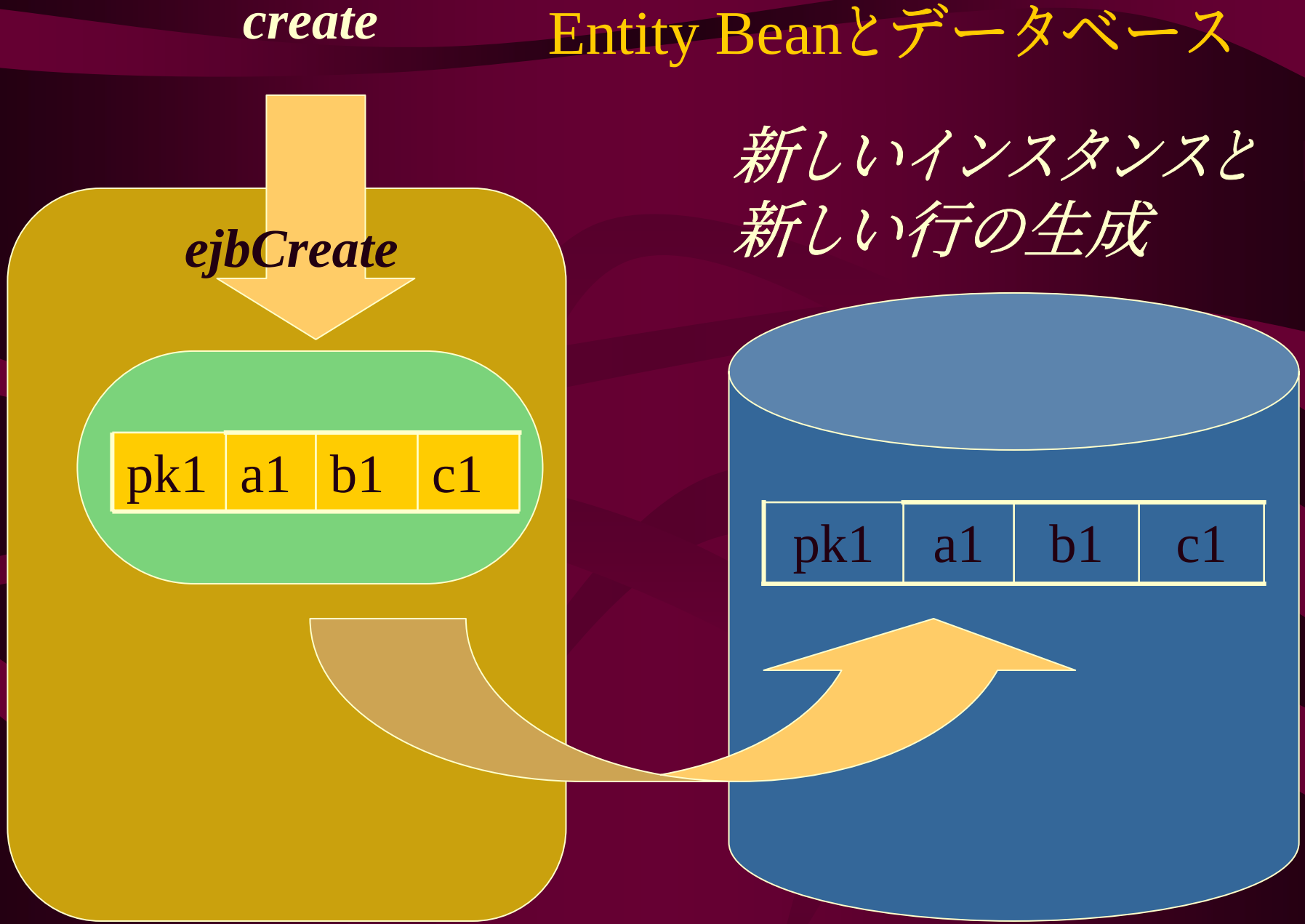
データベースを意識せずに
Beanのインスタンスを考えていればいいので、
ビジネス・ロジックに集中できる。

EJB:　create

メモリー上に新しいインスタンスを生成し、
データベース上に新しい行を挿入する。

SQL：INSERT

Entity Beanとデータベース



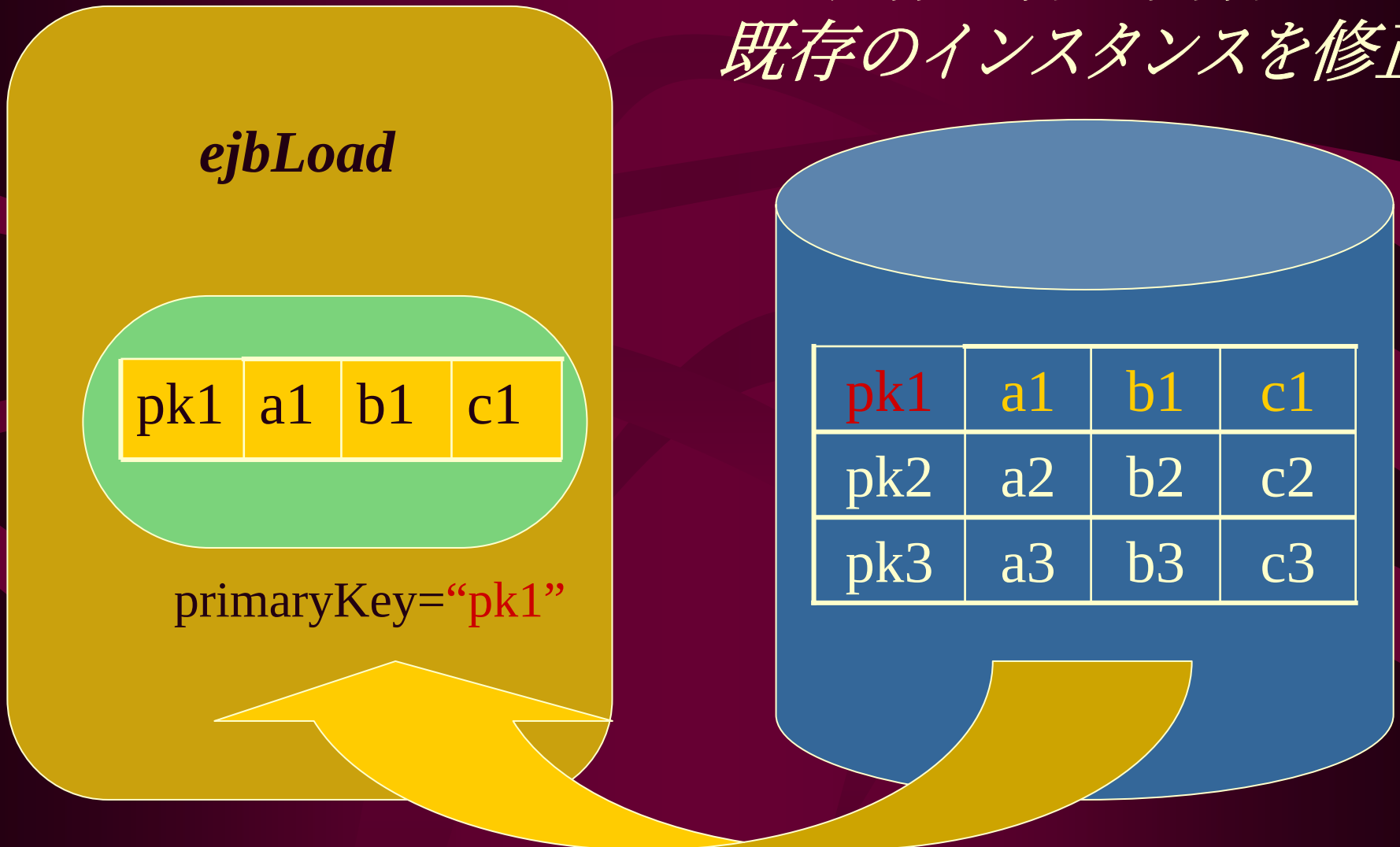
ejbLoad と *ejbStore*

ビジネス・メソッドの呼び出しの前後に
メモリー上のインスタンスと、
データベース上の行との同期を行う。

ejbLoad ***SQL: SELECT***
ejbStore ***SQL: UPDATE***

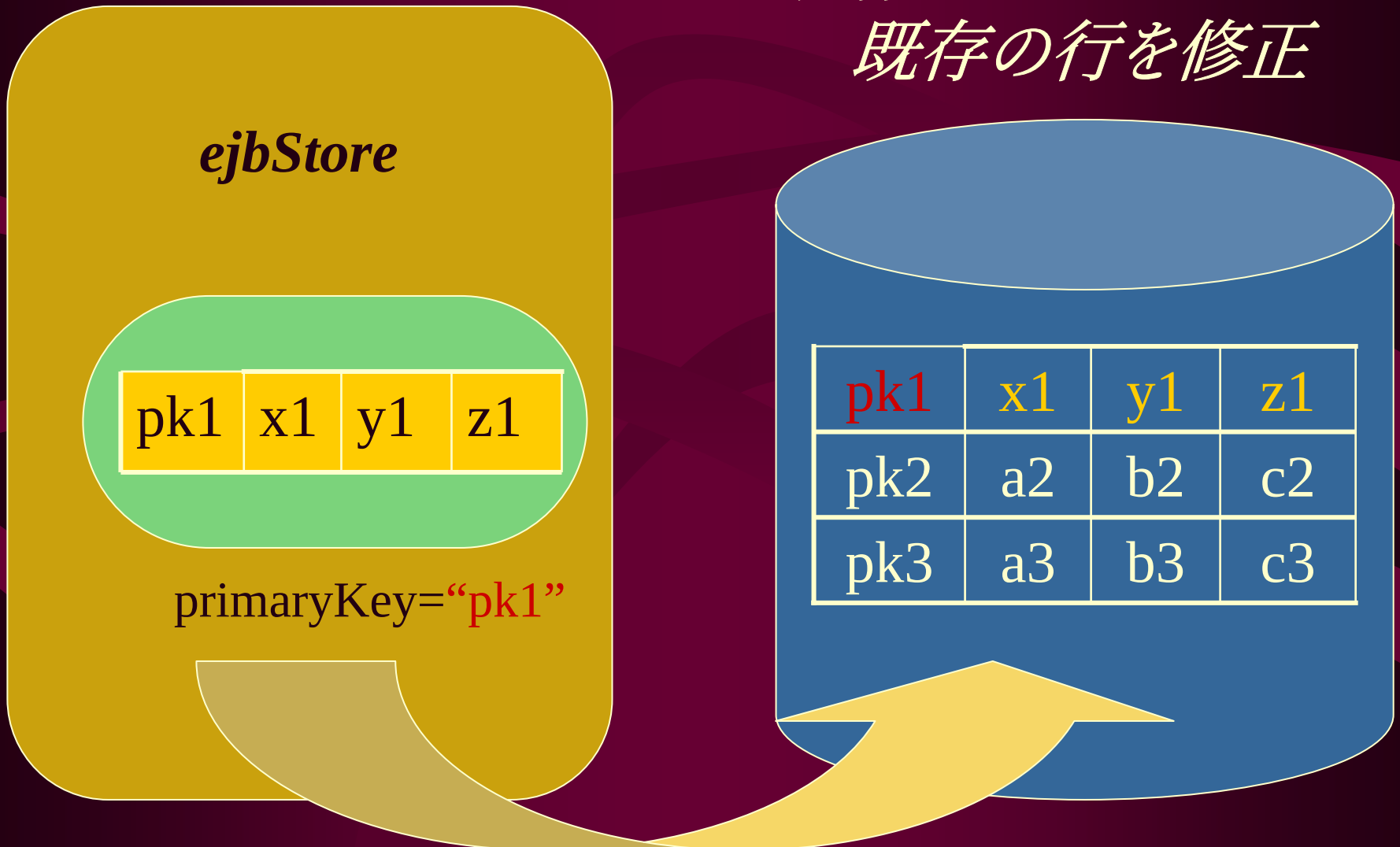
Entity Beanとデータベース

既存の行の内容に
既存のインスタンスを修正



Entity Beanとデータベース

既存のインスタンスで
既存の行を修正



EJBのメソッドとSQLの命令の対応

- ejbCreate　　insert
- ejbRemove　　delete
- ejbLoad　　select
- ejbStore　　update
- ejbFindByPrimaryKey　　select

BMP (Bean Managed Persistency) とは？

- ejbCreate　　insert
- ejbRemove　　delete
- ejbLoad　　select
- ejbStore　　update
- ejbFindByPrimaryKey　　select

これらの対応を、
プログラマが実装するのが
BMP。

永続性管理の自動化

J2EE1.3

J2EE1.3 EJB2.0でのCMP (Container Managed Persistency)

現実のプロジェクトで、EJBコンポーネントの永続性を管理する最良の方策は、永続性管理を自動化することである。

J2EE1.3 EJB2.0でのCMPの特徴

- Abstract Class
- Abstrct Accessor
  - テーブルの項目を表現するPersistentフィールド（cmpフィールド）は、abstract なsetter/getterによってアクセスされる。
  - テーブル間の関係を表現するRelationフィールド（cmrフィールド）も、abstract なsetter/getterによってアクセスされる。

J2EE1.3 EJB2.0でのFinder/Select

- Finderメソッド
  - (Local)Homeインターフェース上で"find"で始まる名前をもつ
- Select メソッド
  - EntityBeanの定義で、"ejbSelect"で始まる名前をもつabstractメソッド
- ともに、EJB-QLを通じて、Deploy時にDeployment Descriptorに定義される。
- Homeメソッド
  - Beanのインスタンスに共通するビジネス・メソッドを定義する。(static method と似ている)

J2EE1.2 EJB1.1から J2EE1.3 EJB2.0への移行

- EntityBeanをabstract Classとして実装する
- Localインターフェースを利用して実装する
- cmpへのsetter/getterメソッドをabstractにする
- テーブルの関係を表現するSQLコードを、cmrのabstractなsetter/getterで置き換える
- FinderとSelectメソッドを、EJB-QLを利用して追加する
- Deploy時に、テーブル間の関係を定義する

J2EE1.3
EJB CMPサンプル

「スポーツ名簿」サンプル

- **リーグテーブル** (LeagueBeanTable)

leagueId	name	sport

- **チームテーブル** (TeamBeanTable)

teamId	name	city

- **選手テーブル** (PlayerBeanTable)

playerId	name	position	salary

テーブル間の関係

- **リーグテーブル** (LeagueBeanTable)

1 : m  **一つのリーグに複数のチーム**

- **チームテーブル** (TeamBeanTable)

m : n  **一つのチームに複数の選手**
一人の選手が複数のチームに所属？

- **選手テーブル** (PlayerBeanTable)

「選手テーブル」に対応したCMP Field

```
public abstract class PlayerBean implements EntityBean {
    // cmp：テーブル項目を定義するSetter/Getterのペア
    // 項目「選手ID (playerId)」の定義
    public abstract String getPlayerId();
    public abstract void setPlayerId(String id);
    // 項目「選手名 (name)」の定義
    public abstract String getName();
    public abstract void setName(String name);
    // 項目「ポジション (position)」の定義
    public abstract String getPosition();
    public abstract void setPosition(String position);
    // 項目「年俸 (salary)」の定義
    public abstract double getSalary();
    public abstract void setSalary(double salary);
```

CMP Field

playerId	name	position	salary

「選手テーブル」が関係するCMR Field

```
// テーブル間の関係を表現している、cmrフィールド teams
// を定義するSetter/Getterのペア
// この選手が所蔵するチーム（複数ありうる）の情報
// このcmr   teamsは、EJB-QLのIN operatorで、IN(p.teams)
// の形で使われる
 public abstract Collection getTeams();
 public abstract void setTeams(Collection teams);
 // セレクト・メソッド
 public abstract Collection ejbSelectLeagues(LocalPlayer player)
     throws FinderException;
 public abstract Collection ejbSelectSports(LocalPlayer player)
     throws FinderException;
     ..........
}
```

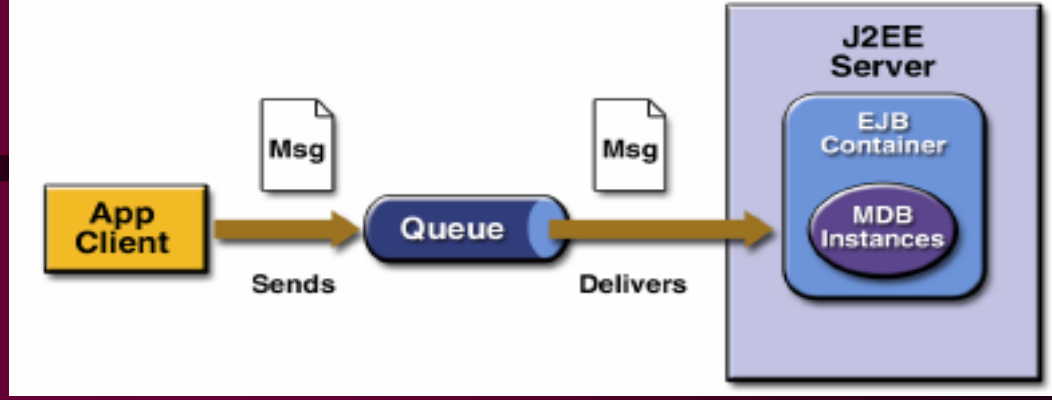
CMR Field

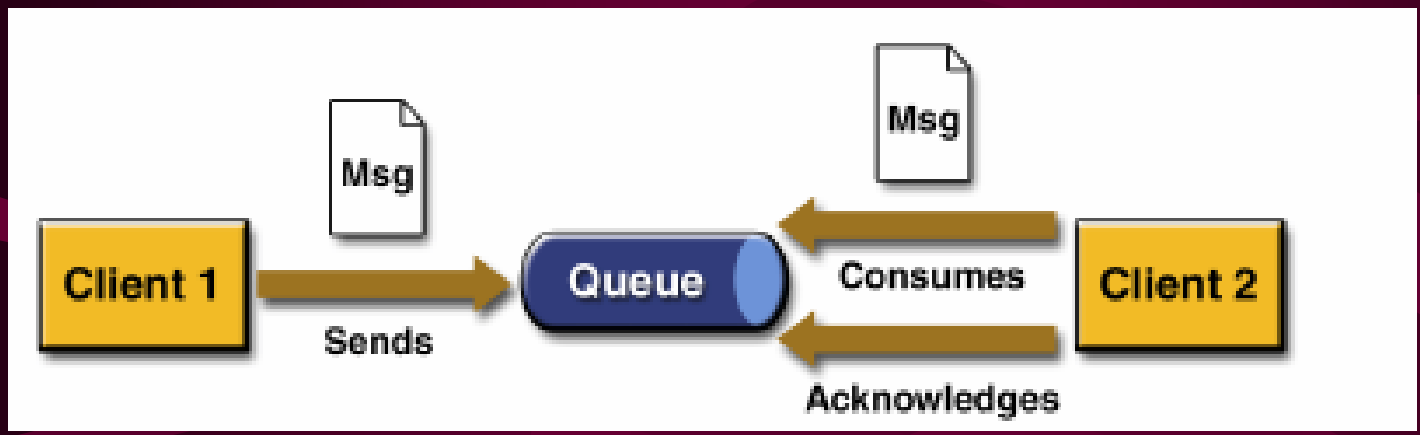
teamId	name	city

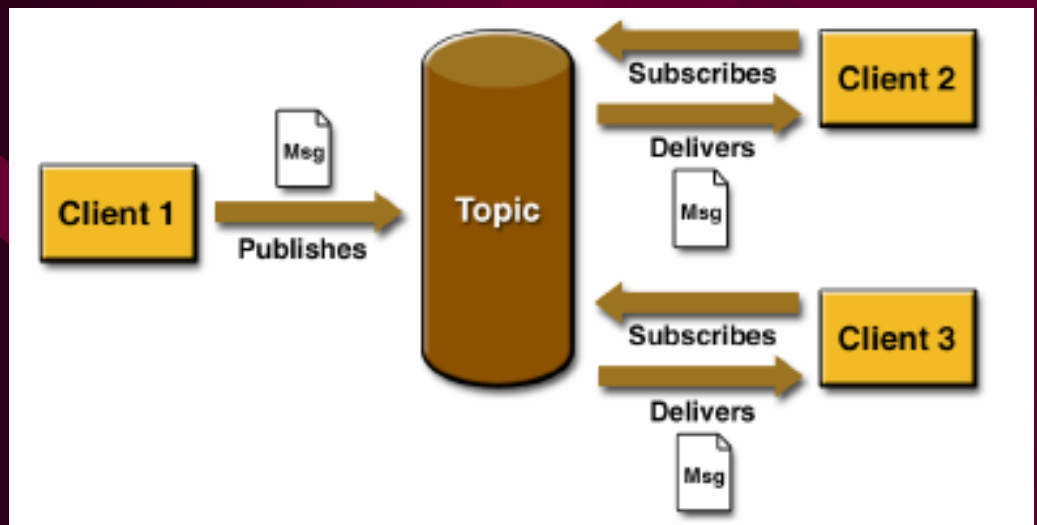
JMSメッセージングの導入

J2EE1.3

省略します。


J2EE Server
EJB Container
MDB Instances
App Client
Msg
Sends
Queue
Msg
Delivers


Client 1
Msg
Sends
Queue
Msg
Consumes
Client 2
Acknowledges


Client 1
Msg
Publishes
Topic
Subscribes
Client 2
Delivers
Msg
Subscribes
Client 3
Delivers
Msg

3. J2EEとWebサービス / Grid

- J2EE1.4でのWebサービスの導入
- SOAP-RPCからJAX-RPCへ
- Endpointインターフェースを持つ Session Beans
- J2EE1.4　Webサービスの利用イメージ
- J2EE1.4とEoD: Java Server Faces
- J2EE1.4とJAIN SLEE

- GridとWebサービスの接近
- GridとWebサービスの統合
  -- OGSIからWS-RFへ --

J2EEへのWebサービスの導入

J2EE1.4

J2EEとWebサービスとの統合の課題

Web層でJ2EEとWebサービスとを統合することはそれほど難しくはない。問題はEJB層。

EJB層のEntityビーンもSessionビーンも、コンテナの外部とは、基本的にはRMIを使って通信しなければならない。

一方、Webサービスは、SOAPのプロトコルを使わなければならない。

SOAP-RPCからJAX-RPCへ

JAX-RPC = RMI / SOAP

Rahul Sharma

JAX-RPC の特徴

- JavaとWSDLの対応付け
- Javaの型とXMLの型の対応付け
- JavaでのWebサービスの記述
- 多様なクライアントモデル

Java Remote Interface → WSDL

WSDL → Java クラス群

JAX-RPC

Java Remote Interface ⇒ WSDL

WSDL ⇒ Java クラス群

Java Interface から、WSDLを生成する
WSDLから、Javaクラスを生成する

WSDLタグ	対応して生成されるファイルの種類
portType	サービス･エンドポイント･インターフェース
binding	Stub クラス サービス実装クラス
service	サービス･インターフェース サービス･ロケータ

JAX-RPC WSDL2Javaコマンドで生成されるファイル / メソッド達

WSDLタグ	生成されるファイル名	例
portType	portTypeタグのname 属性 + ".java"	Hello.java
binding	binding タグのname 属性 + Stub.java“ binding タグのname 属性 + "Impl.java"	HelloBindingStub.java HelloBindingImpl.java
service	service タグのname 属性 + ".java" service タグのname 属性 + "Locator.java"	HelloWorld.java HelloWorldLocator.java

WSDLタグ	生成されるメソッド名	例
portType	portTypeタグのoperationタグのname 属性	sayHello
binding	binding タグのoperationタグのname 属性	
service	"get" + serviceのportタグのname 属性 "get" + serviceのportタグのname 属性	getHello

サービス定義インターフェース Hello.java

```
public interface Hello extends java.rmi.Remote {
    public String sayHello(String name) throws
                java.rmi.RemoteException;
}
```

サービス実装クラス HelloImpl.java

```
public class HelloImpl implements Hello{
    public String sayHello(String name) throws
                java.rmi.RemoteException {
        return "ServiceImpl was re-defined by " + name + "!" ;
    }
}
```

サービス･インターフェース **HelloService**.java

```
public interface HelloService extends javax.xml.rpc.Service {
    public String getHelloAddress();
    public Hello getHello()
          throws javax.xml.rpc.ServiceException;
    public Hello getHello(java.net.URL portAddress)
          throws javax.xml.rpc.ServiceException;
}
```

WSDL service Element

```
<service name="HelloService">
  <port name="Hello" binding="tns:HelloBinding">
    <soap:address
    location="http://localhost:8080/axis/services/Hello" />
  </port>
</service>
```

サービス･ロケータ HelloServiceLocator.java

```
public class HelloServiceLocator
    extends org.apache.axis.client.Service
    implements HelloService {
    ......
    public Hello getHello()
          throws javax.xml.rpc.ServiceException {
        ........
    }
   ......
    public Hello getHello(java.net.URL portAddress)
          throws javax.xml.rpc.ServiceException {
        ........
    }
    ......
}
```

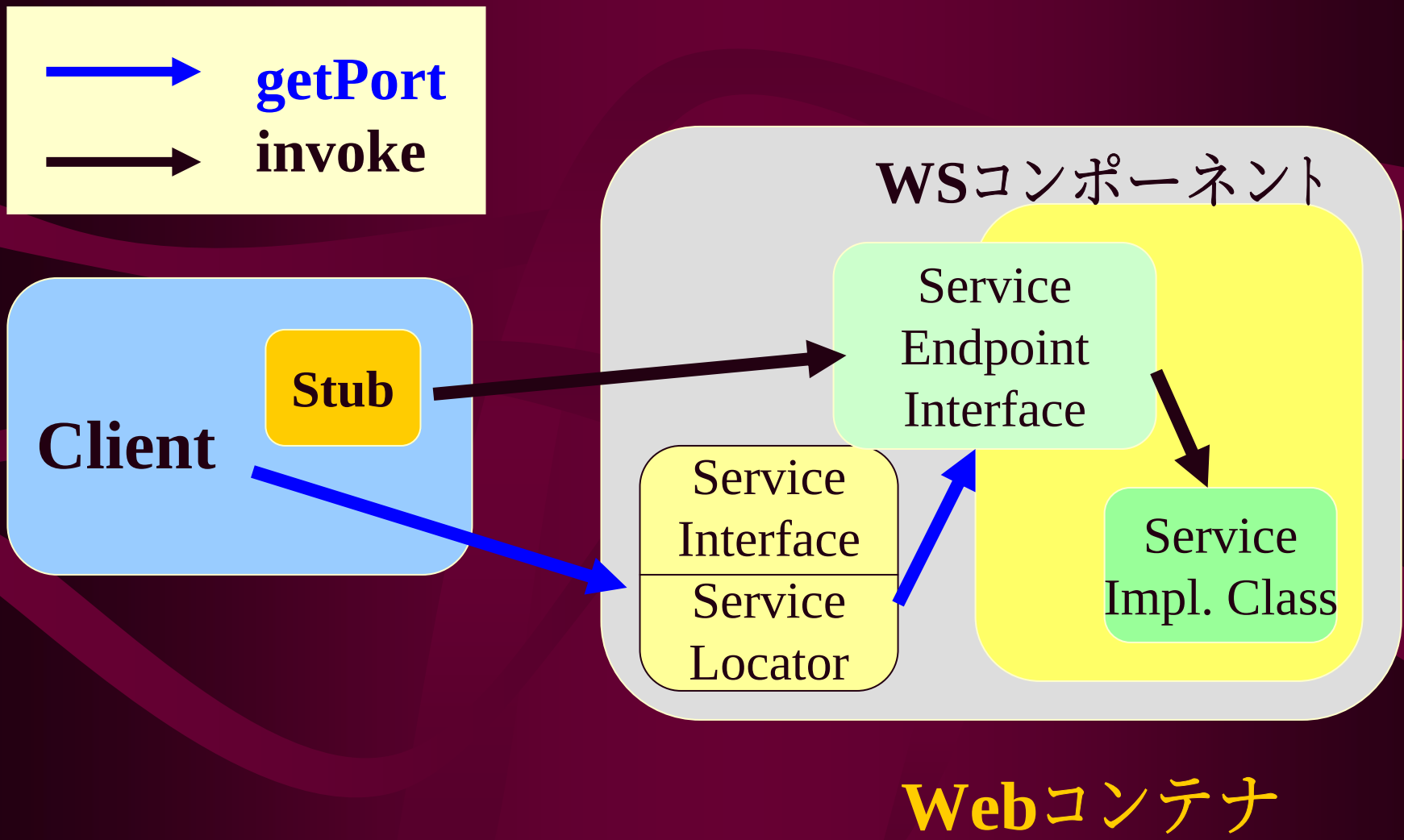
サービス･インターフェースを実装

サービス･インターフェースがコンテナへのインターフェースを提供する

クライアント・プログラム Main.java

```
public class Main {
    public static void main (String[] args) throws Exception {
        Hello port = new HelloServiceLocator().getHello();
        try {
            String str = ((HelloStub) port).sayHello("Fujio");
            System.out.println( str );
        } catch (java.rmi.RemoteException re) {
            System.err.println("Remote Exception caught: " + re );
        }
    }
}
```

JAX-RPCがWSDLから生成するクラスとコンテナ



Endpointインターフェースを持つ Session Beans (EJB2.1)

- (Local)Homeインターフェースを持たず、
- EJB(Local)Objectを継承した、Local/Remoteインターフェースを持たない。

Serviceインターフェースを拡大した Serviceインターフェースをもち、

- 直接、Remoteインターフェースを拡大した Endpointインターフェースをもち、
- そのEndpointインターフェースを実装したクラスを持つ

Endpointインターフェースを持つ Session Beans (EJB2.1)

```
public Interface BookPriceService extends
        javax.xml.rpc.Service{
  public BookPrice getBookPrice( )
      throws RemoteException;
}
public interface BookPrice extends
        javax.rmi.Remote {
  public String getBookPrice(String isbn)
      throws javax.rmi.RemoteException;
}
public class BookPriceWS  implements
        BookPrice, javax.ejb.SessionBean {
  public float getBookPrice(String isbn){….}
    …….
}
```

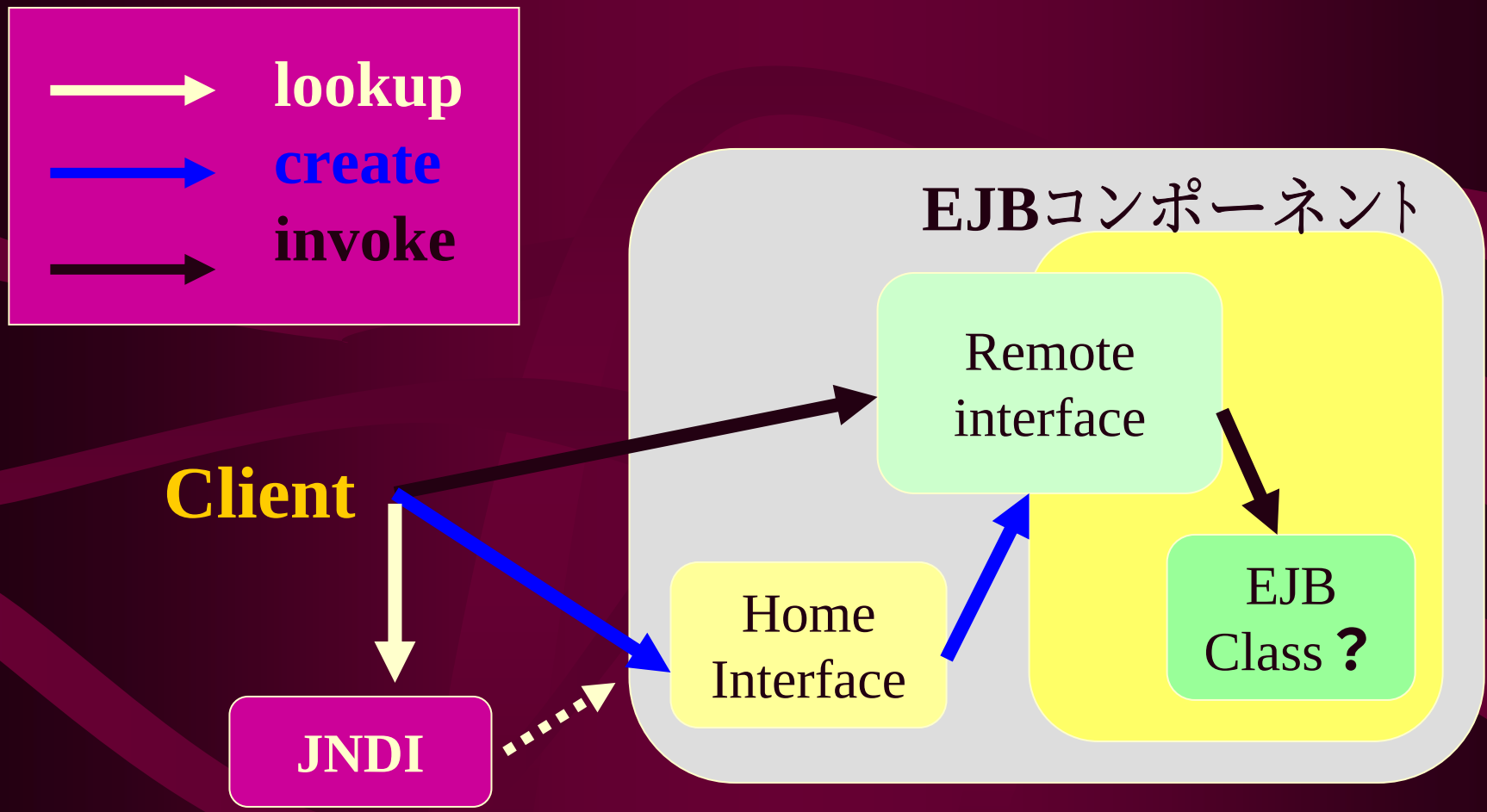
通常のSession Beans

```
public Interface HelloHome extends
        javax.ejb.EJBHome {
  public Hello create( )
      throws RemoteException, CreateException;
}
```

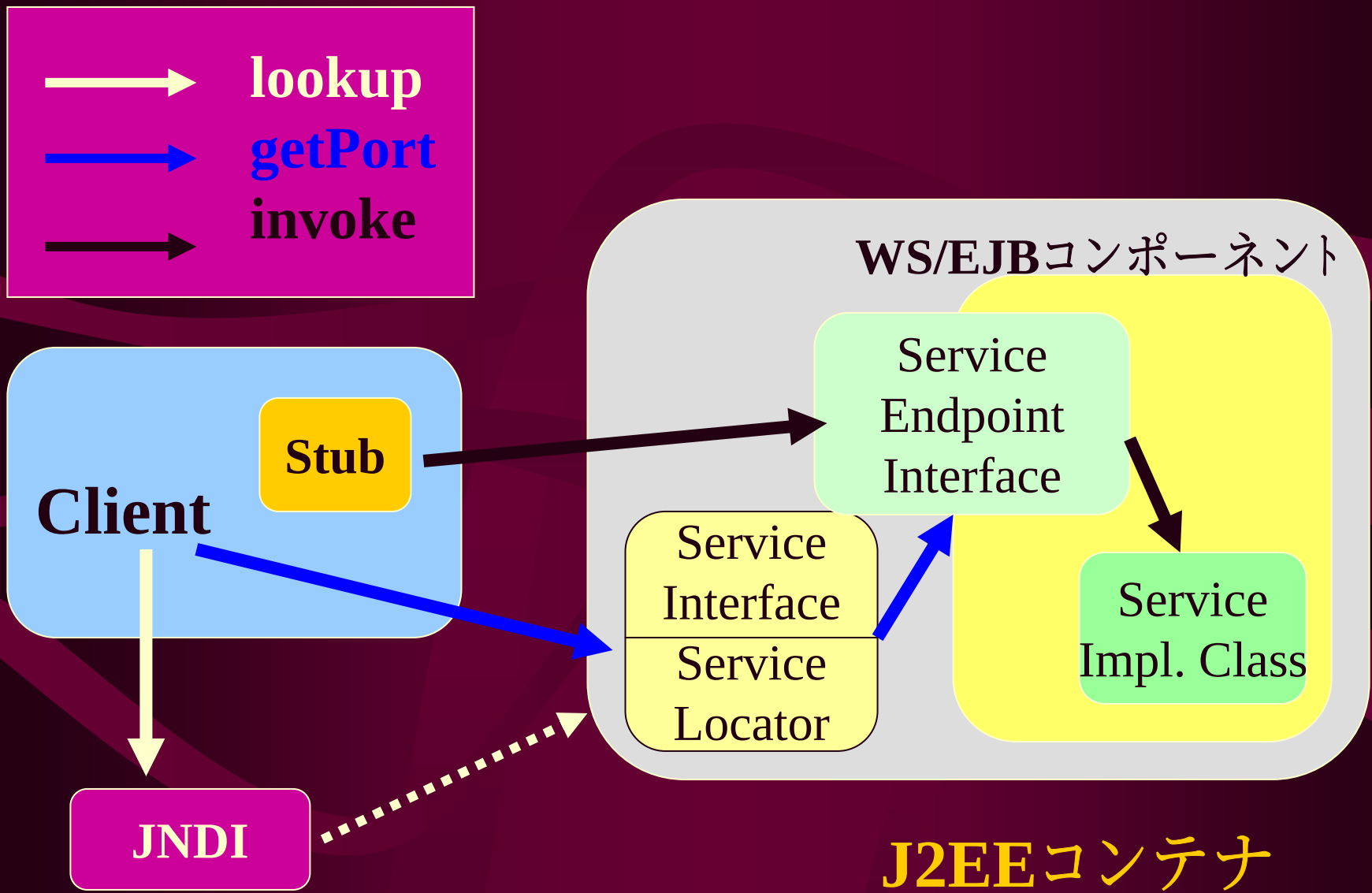
```
public interface Hello extends
        javax.ejb.EJBObject{
  public String sayHello(Stringname)
      throws javax.rmi.RemoteException;
}
```

```
public class HelloBean  implements
        javax.ejb.SessionBean {
  public String sayHello(String name) {….}
    …….
}
```

Clientから見たJ2EEのコンテナとEJBの三つの定義ファイル

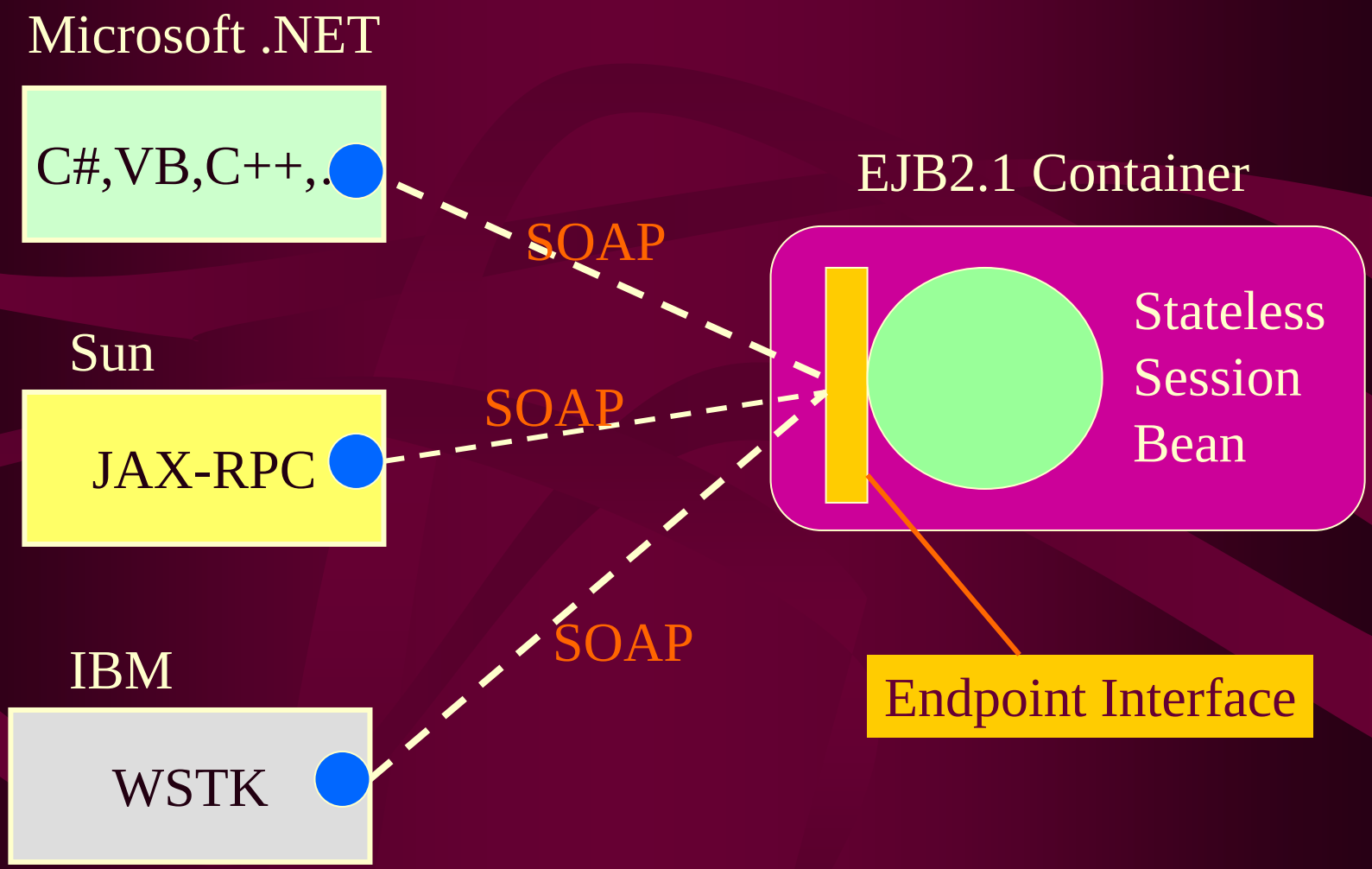


J2EE1.4のWebサービス対応

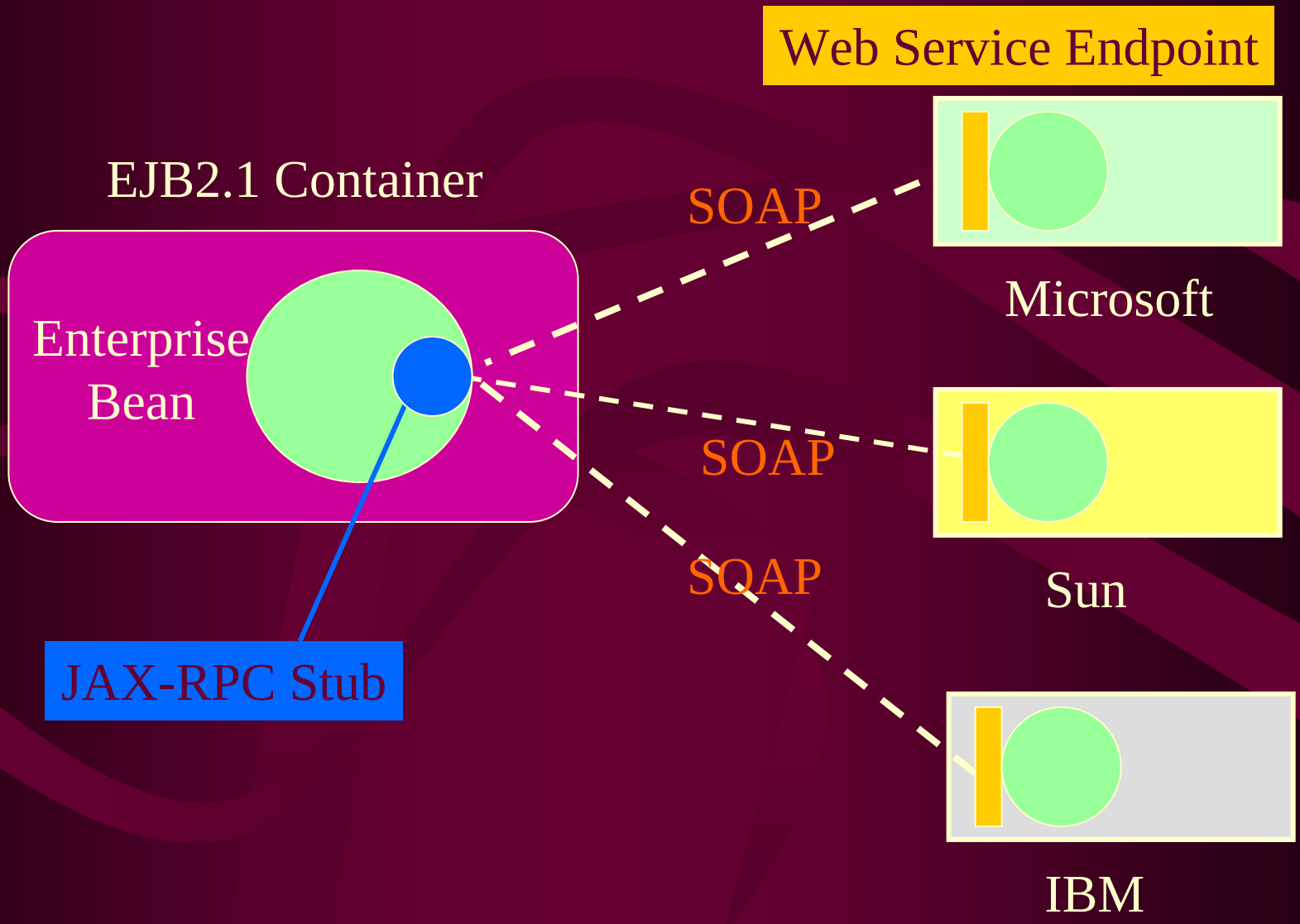


J2EE1.4　**Web**サービスの利用イメージ

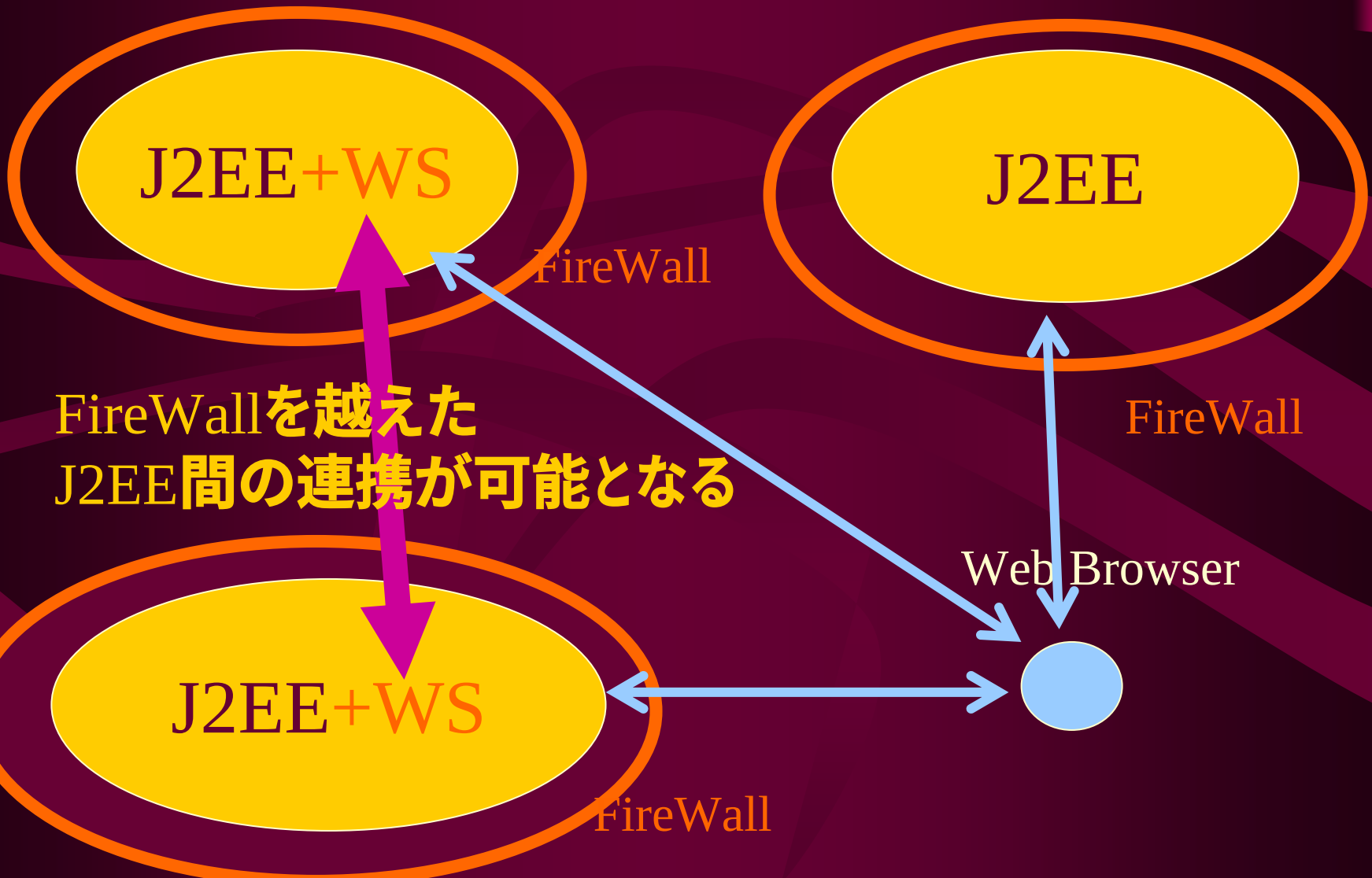
SOAP互換であれば、誰でも、セッション・ビーンのメソッドを呼び出せる



J2EEのビーンのJAX-RPCのStubからどんなWebサービスも呼び出せる



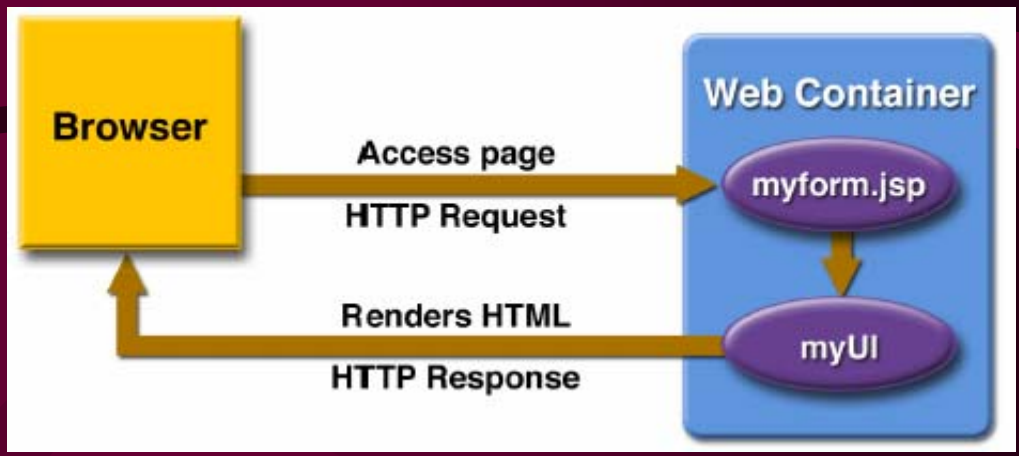
FireWallを越えたシステム間の連携
Enterprise Gridの基礎

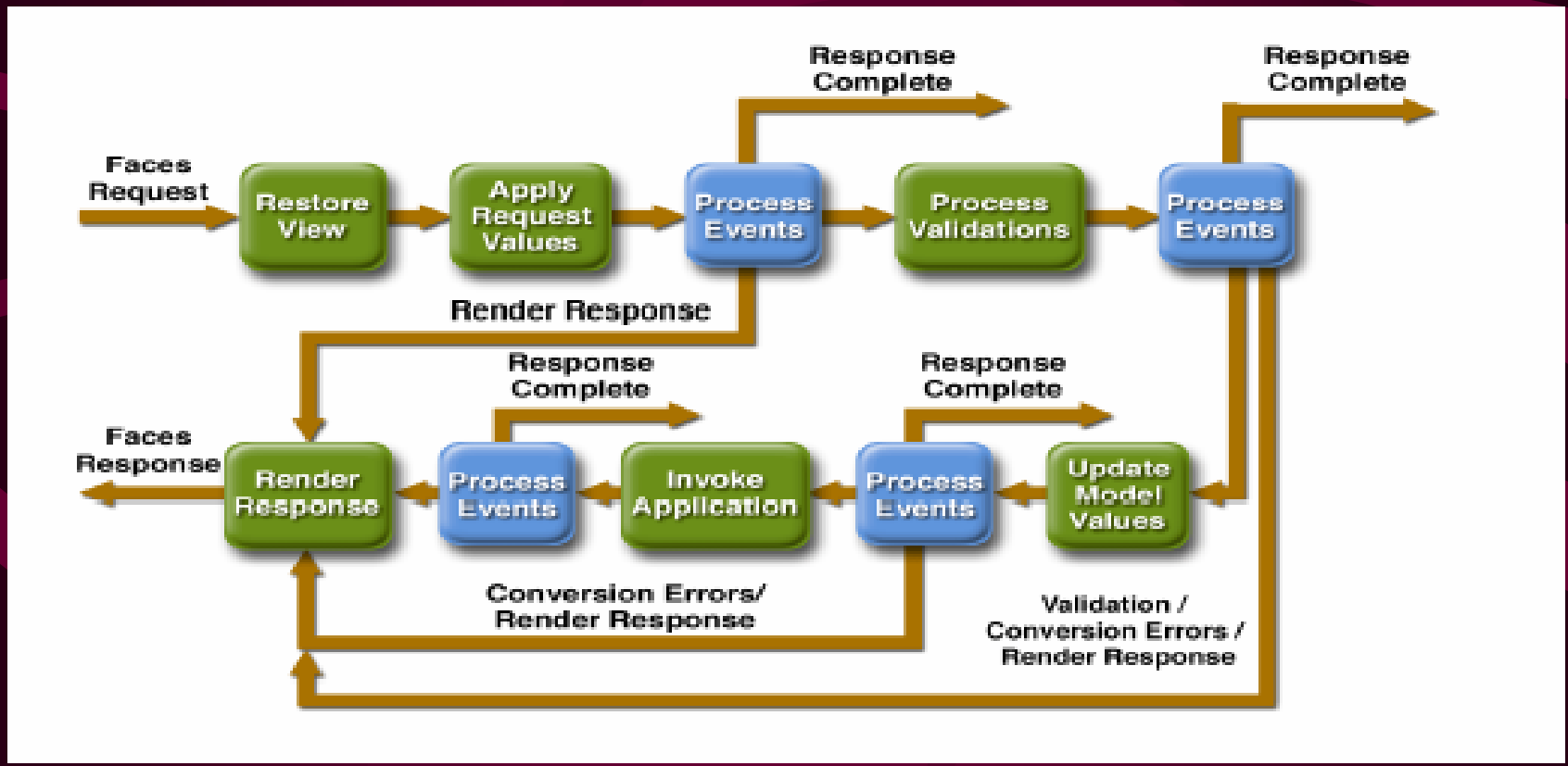


Craig R.McClanahan

J2EE1.4とEase of Development : Java Server Faces

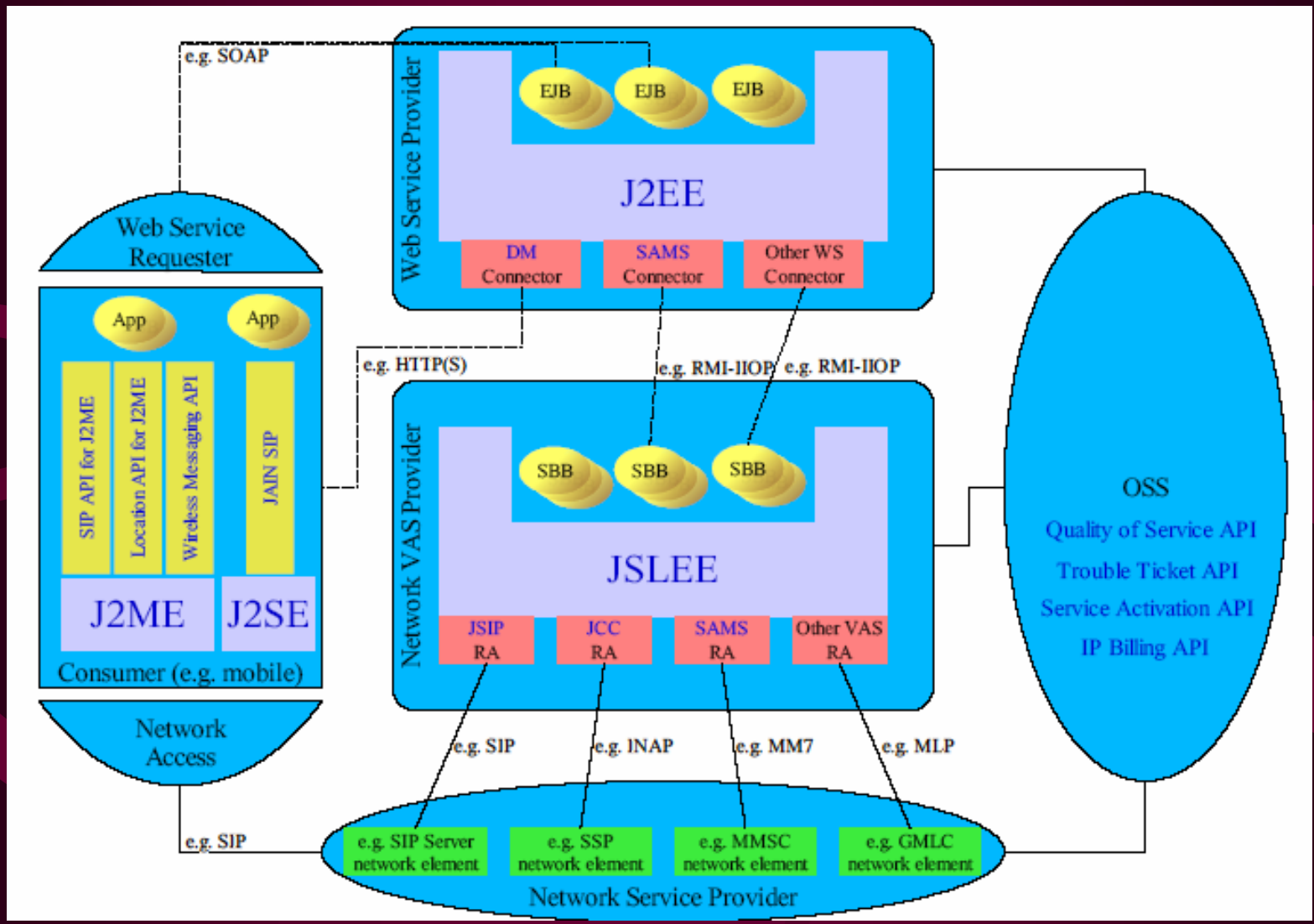
省略します。


Browser
Web Container
Access page
HTTP Request
myform.jsp
Renders HTML
HTTP Response
myUI


Response Complete
Response Complete
Faces Request
Restore View
Apply Request Values
Process Events
Process Validations
Process Events
Render Response
Response Complete
Response Complete
Faces Response
Render Response
Process Events
Invoke Application
Process Events
Update Model Values
Conversion Errors/ Render Response
Validation / Conversion Errors / Render Response

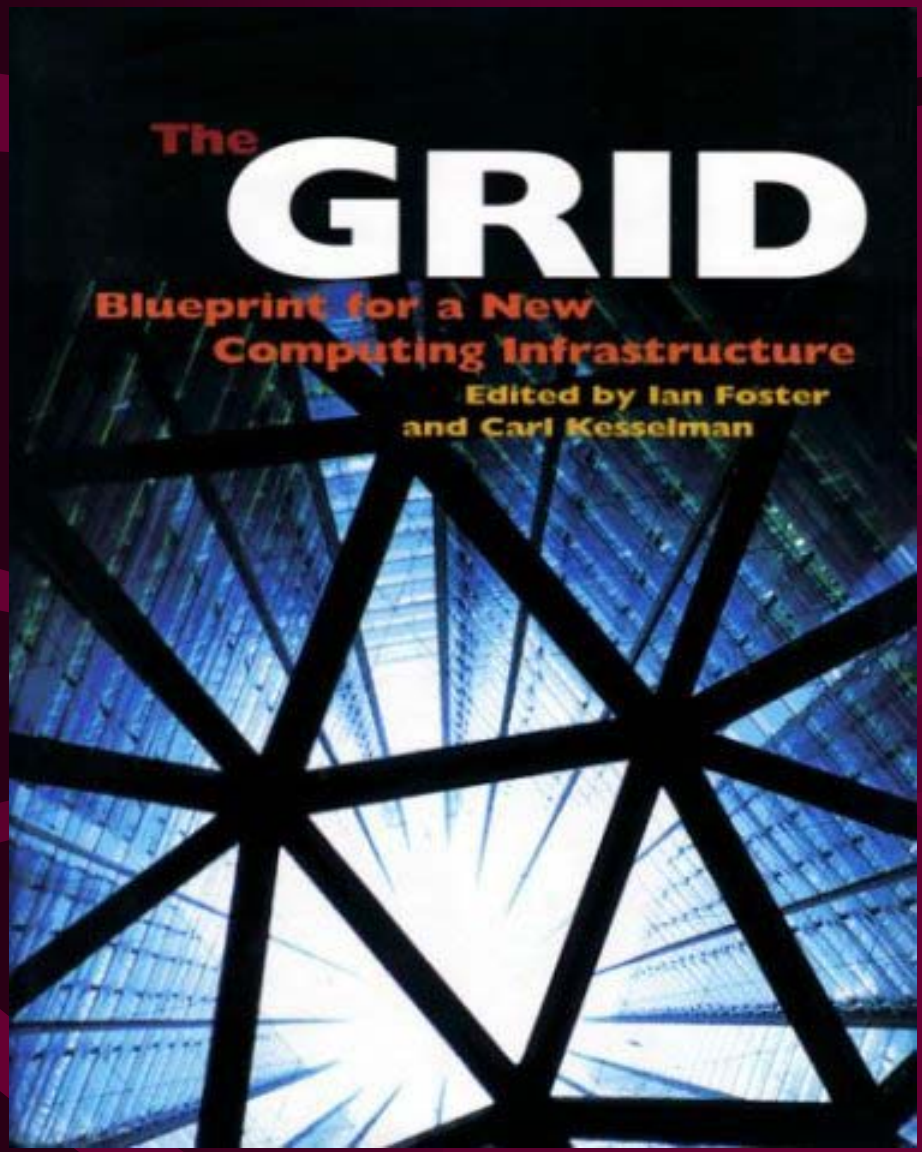
J2EE1.4とJAIN SLEE
(Service Logic Execution Environment)

省略します。


e.g. SOAP
Web Service Provider
EJB
EJB
EJB
J2EE
DM Connector
SAMS Connector
Other WS Connector
Web Service Requester
App
App
SIP API for J2ME
Location API for J2ME
Wireless Messaging API
JAIN SIP
J2ME
J2SE
Consumer (e.g. mobile)
e.g. HTTP(S)
e.g. RMI-IIOP
e.g. RMI-IIOP
Network VAS Provider
SBB
SBB
SBB
JSLEE
JSIP RA
JCC RA
SAMS RA
Other VAS RA
OSS
Quality of Service API
Trouble Ticket API
Service Activation API
IP Billing API
Network Access
e.g. SIP
e.g. INAP
e.g. MM7
e.g. MLP
e.g. SIP
e.g. SIP Server network element
e.g. SSP network element
e.g. MMSC network element
e.g. GMLC network element
Network Service Provider

GridとWebサービスの接近

GGF / Globus



Ian Foster

1998年

The Grid: Blueprint for a New Computing Infrastructure

“The Physiology of the Grid”

http://www-nix.globus.org/ogsa/docs/alpha/physiology.pdf

2002/06/22

GridへのWebサービスの全面的な導入

この論文が画期的だったのは、Gridサービスの実現に、e-Businessの世界で急速に標準的な地位を占めつつあるWebサービスを利用することを明確に宣言したこと。

“The Physiology of the Grid”

http://www-nix.globus.org/ogsa/docs/alpha/physiology.pdf

2002/06/22

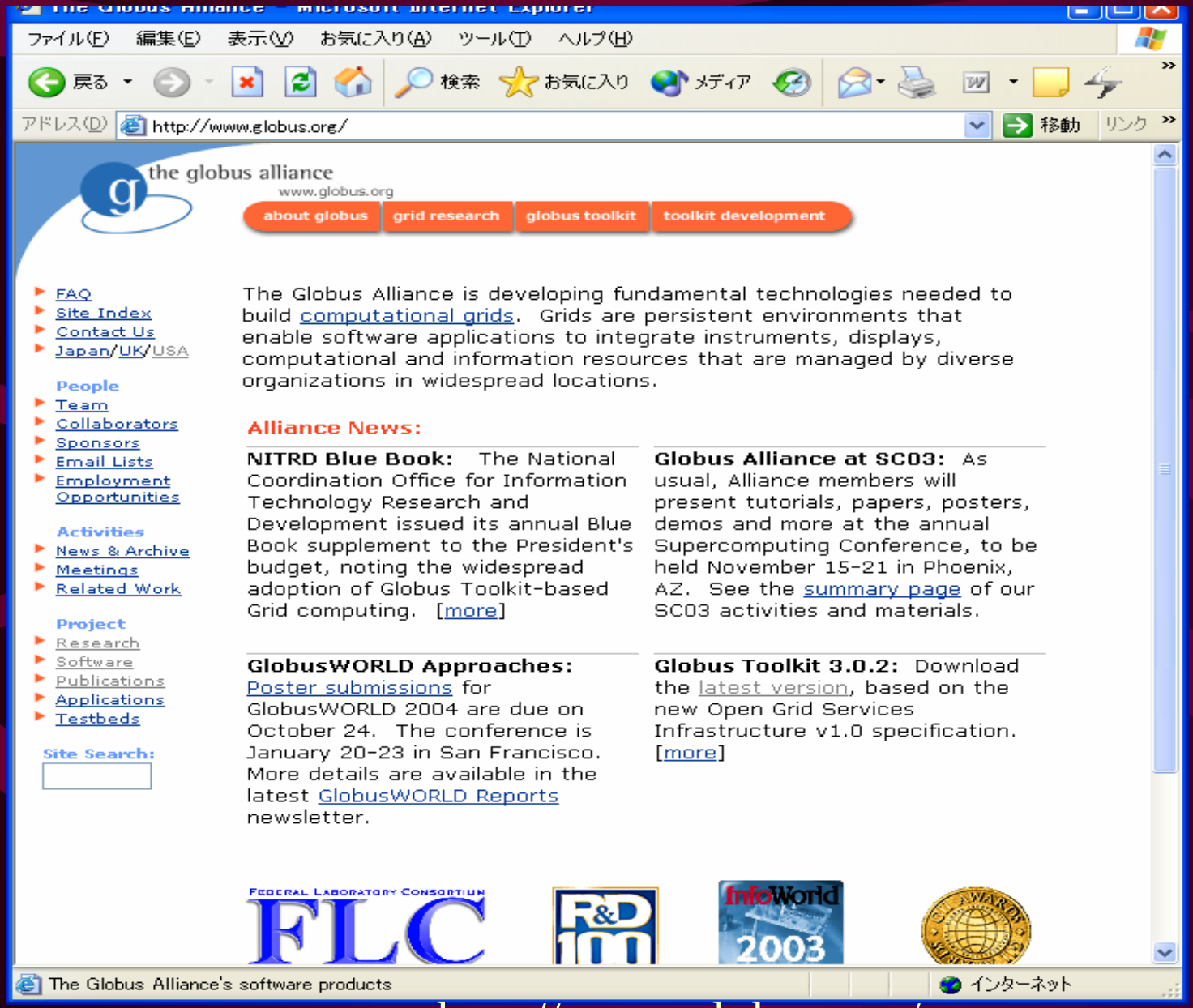
OGSAの提案

Gridと**Web**サービスの提携と、それによる双方の能力の拡大を、“Open Grid Services Architecture (OGSA)”と呼ぶことが提案される。

I.FosterによるGridの定義

- ネットワーク上のノードを……集中管理するのではなく、異なった管理のもとにあるリモート・リソースやリモート・ユーザを、ネットワーク上で統合し協調させることを目指したシステムをGridと呼ぶ

Globus Toolkit 3.0



GT2 から GT3 へ

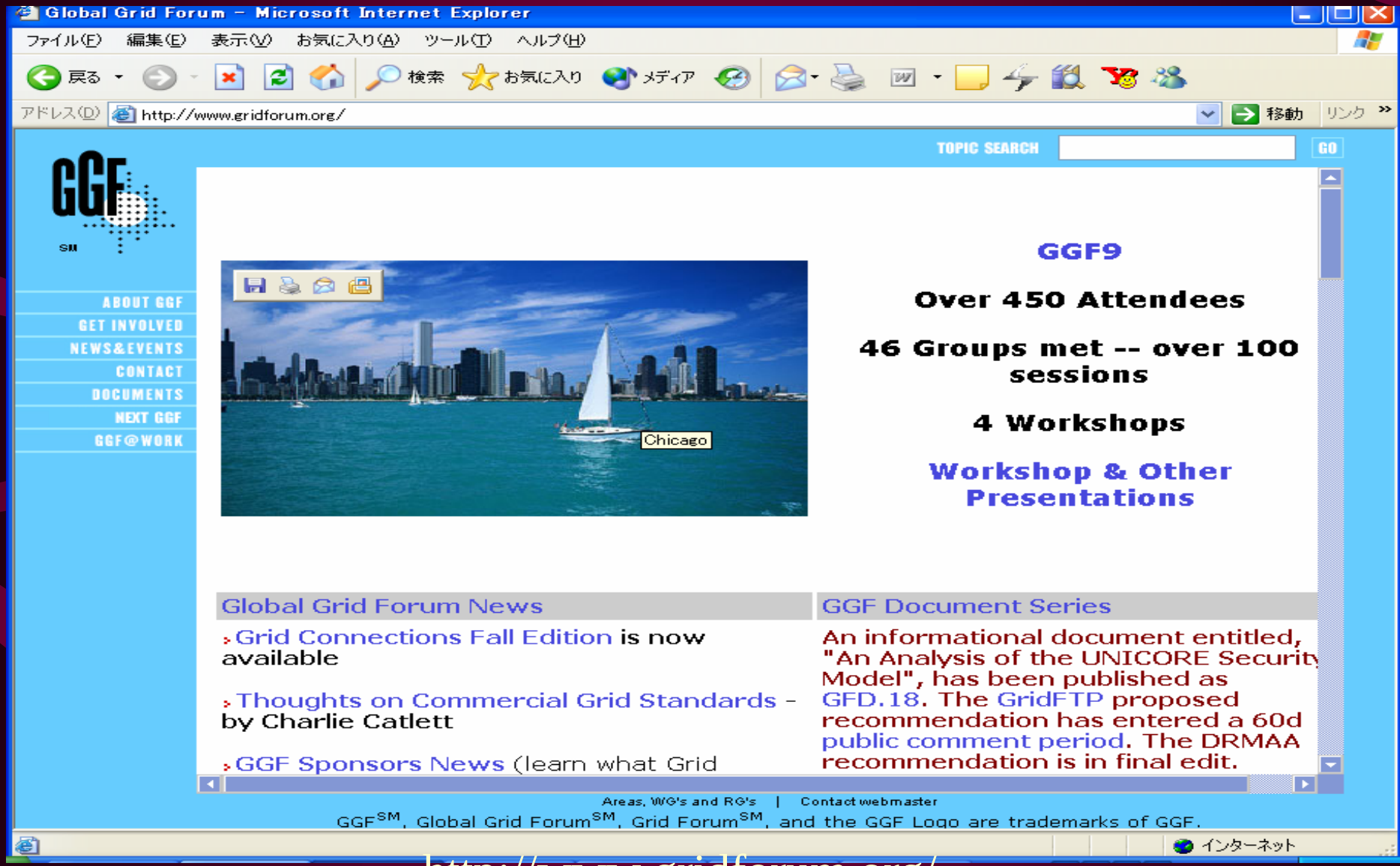
http://www.globus.org/

Globus / GT3でのGrid概念の新しさ

- マシンを、PlatformやOSの違いを超えて、ネットワーク上で、結合・連携させる。
- その結合・連携には、汎用的なネットワークのリソースが利用可能なWebサービスを利用して、マシンを「疎結合」で結ぶ。
- 従来のコンピューティング・パワー指向のGridに対して、リソース共有を指向するGrid

GGF OGSA/OGSI

Open Grid Service Architecture / Open Grid Service Infrastructure



http://www.gridforum.org/

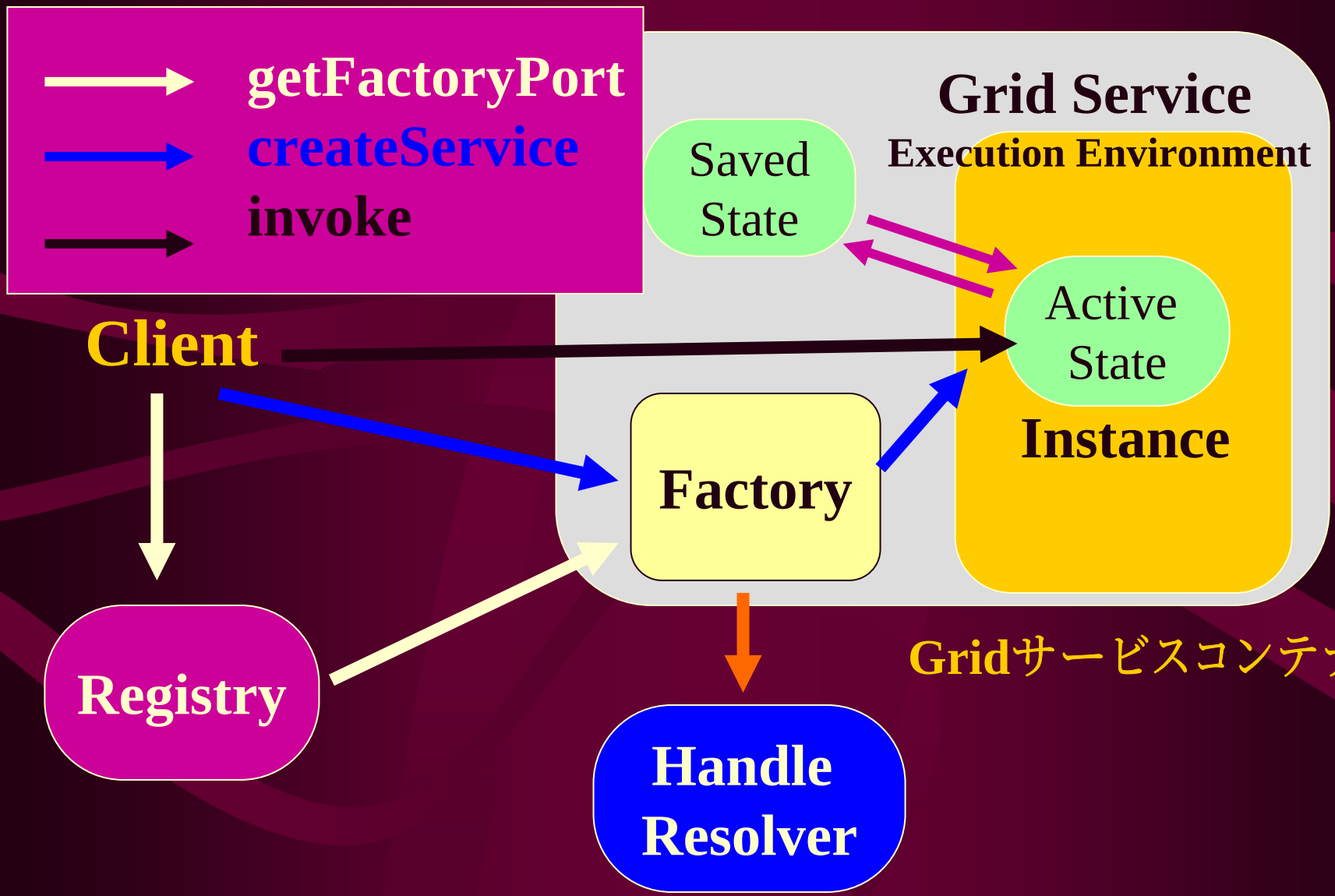
GGF / OGSAの設計目標

- アプリケーションのデザインやコードの再利用を容易にすることができるように、様々な状況の元でも、標準的な方法で共通に利用されるうるソフトウェア・コンポーネントをつくること。
- 上位のサービスが、下位のサービスから構成できるように、コンポーネントの組み合わせを容易にすること。

サービス志向アーキテクチャ

- すべての要素がサービスから構成されているようなアーキテクチャを、「サービス志向アーキテクチャ」と呼ぶ。
- このアークテクチャ上では、全てのoperationは、メッセージ交換の結果として理解される。

Gridサービスコンテナ



J2EE,Web Service, Grid (OGSI) のコンテナ / コンポーネントの比較

	J2EE	Web Service	Grid Service OGSI
Containerの発見	JNDI	UDDI / WSIL	**Registry**
Containerのインターフェース	Home Intf.	Service Intf.	**Factory**
Componentの生成	**create**	**get<Port>**	**createService**
Componentのインターフェース	Remote Intf.	Remote Endpoint	**Handle / Reference**

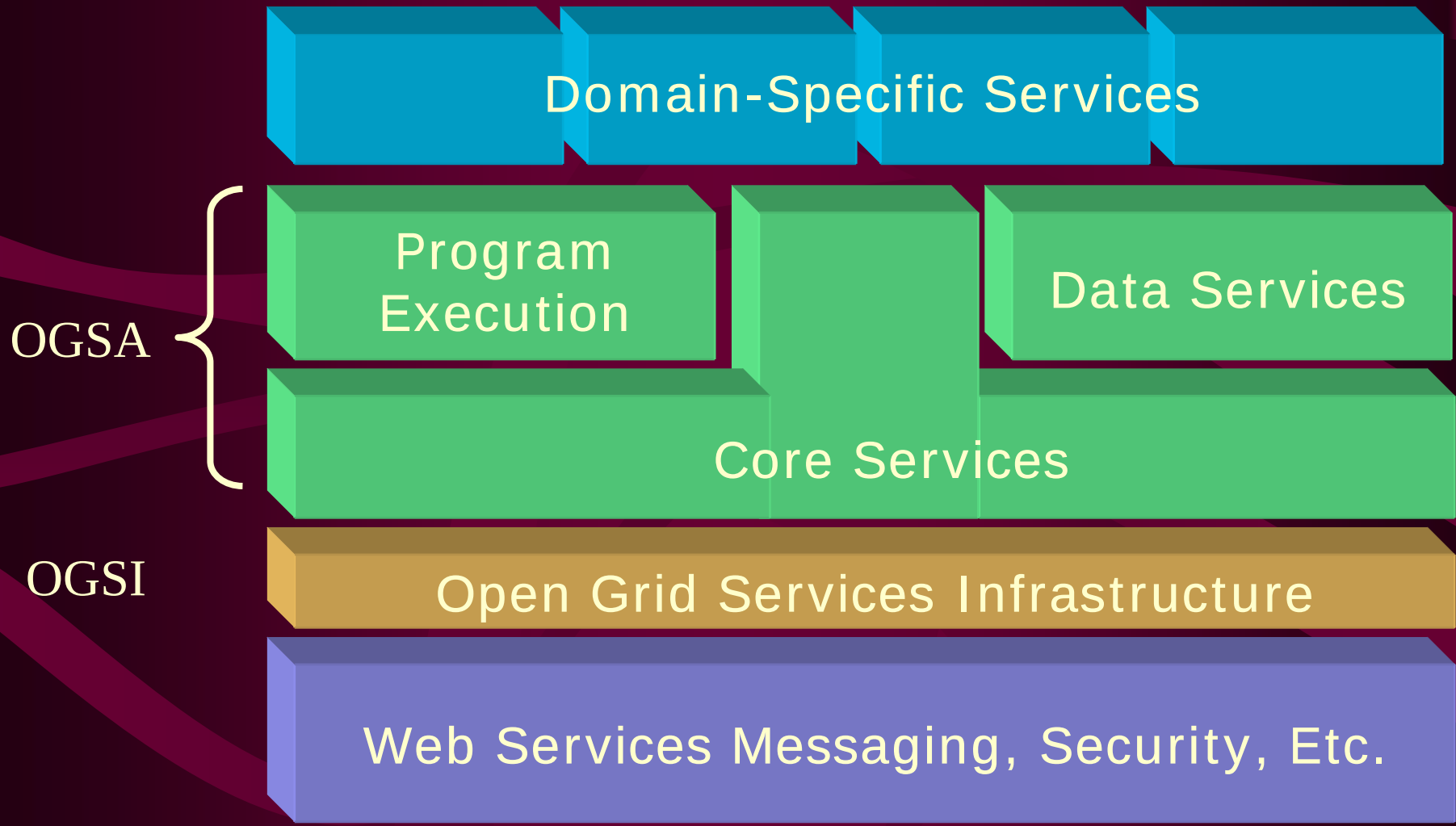
GridとWebサービスの統合

OGSIからWS-RFへ

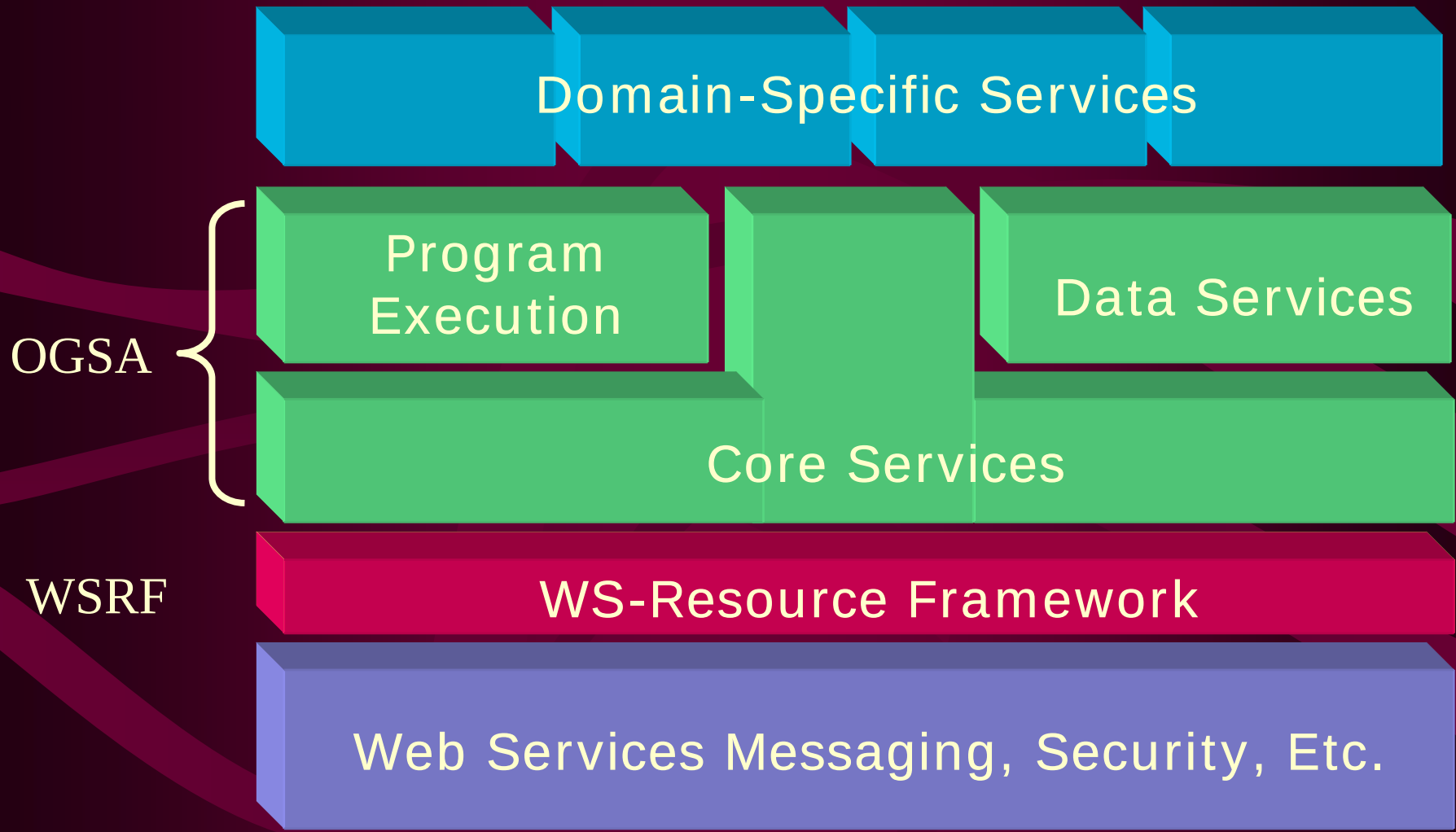
GridとWeb Servicesは接近してきた。

しかし、すれ違って別の道を
進む可能性が出てきた。

OGSA / OGSI


Domain-Specific Services
Program Execution
Data Services
Core Services
OGSA
OGSI
Open Grid Services Infrastructure
Web Services Messaging, Security, Etc.

OGSA / WSRF

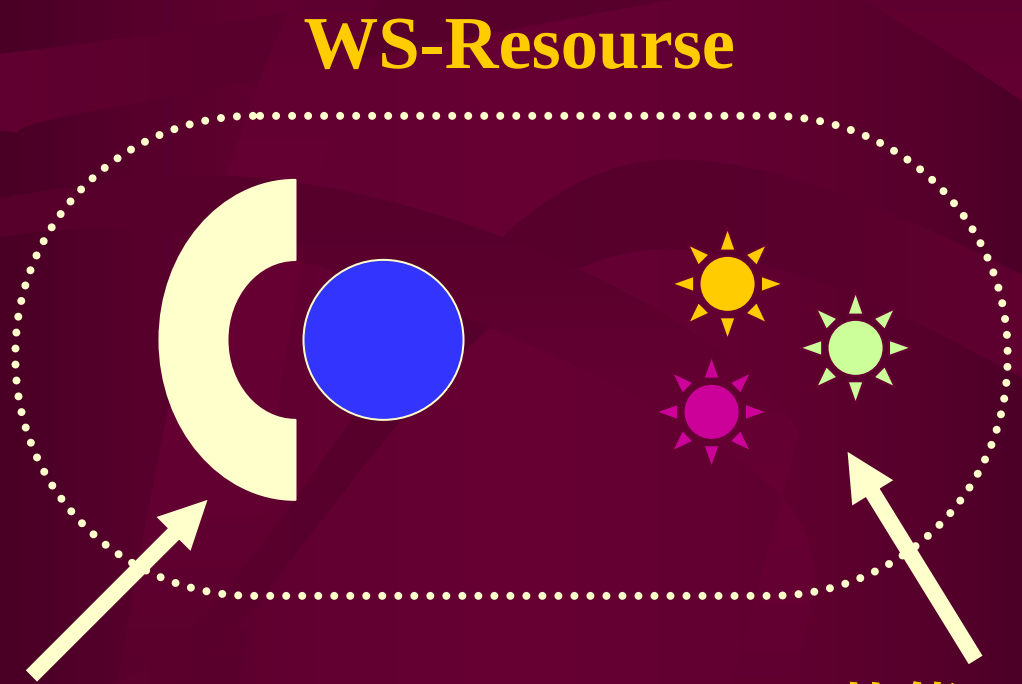

Domain-Specific Services
Program Execution
Data Services
OGSA
Core Services
WSRF
WS-Resource Framework
Web Services Messaging, Security, Etc.

Grid と Web Servicesは WSRFで、一体化する。

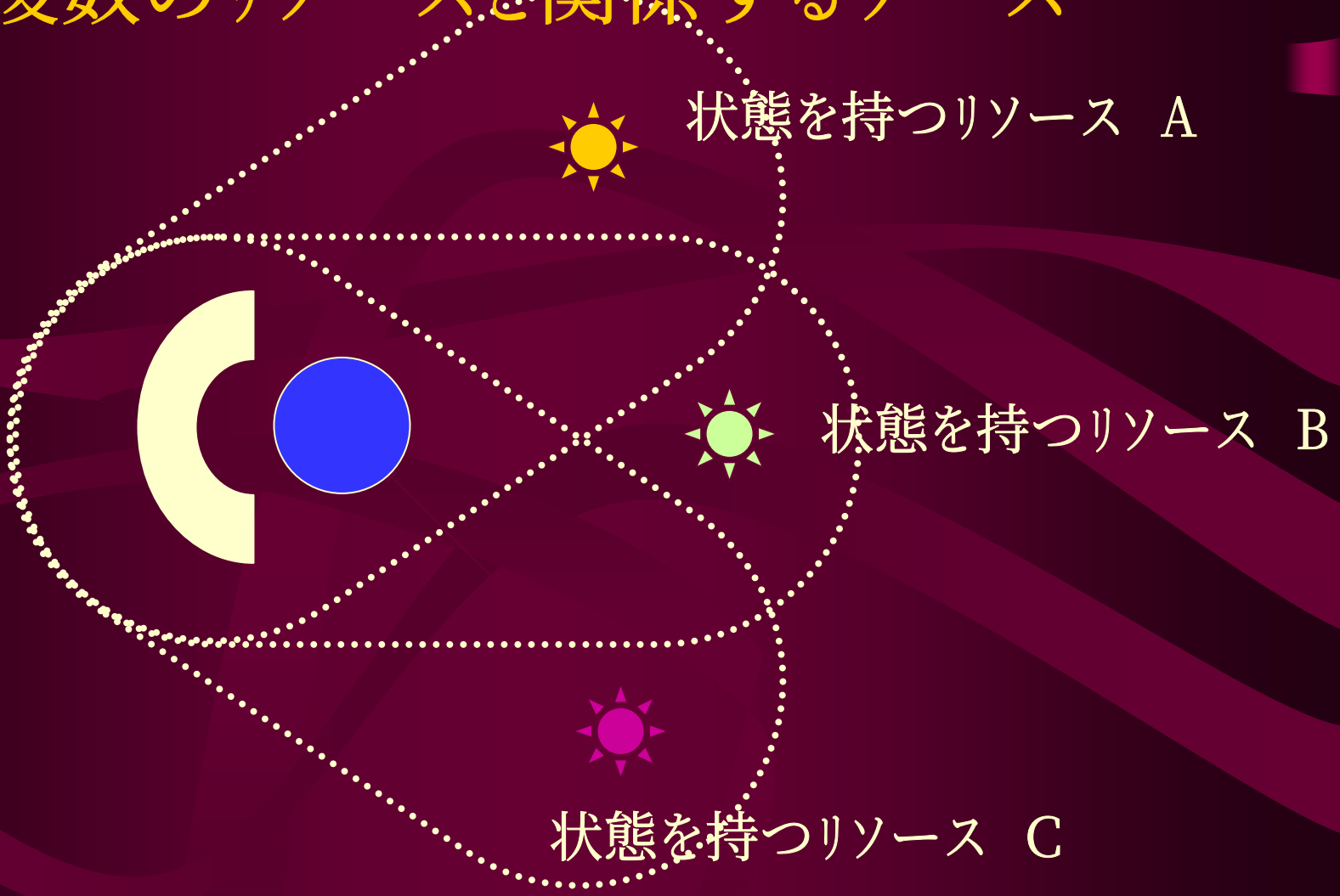
WSRFの定義は、GridとWebサービスのコミュニティが、共通の土台の上に前進できるということを意味している。

WS-Resourseとは何か?

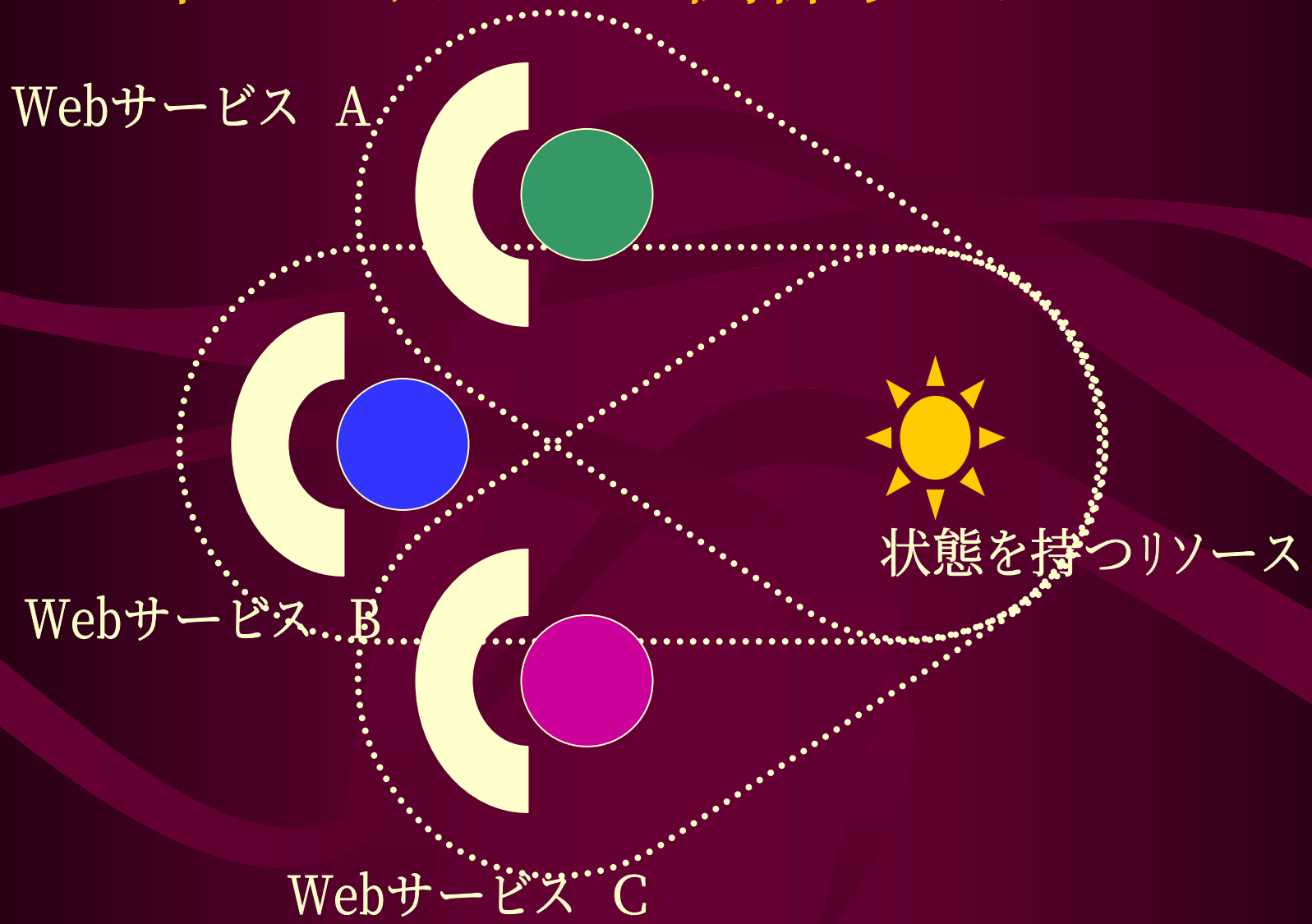
「状態を持たないWebサービス」と
「状態を持つリソース」を
分離したうえで、組み合わせたもの



単一のWebサービスが
複数のリソースと関係するケース



複数のWebサービスが単一のリソースと関係するケース



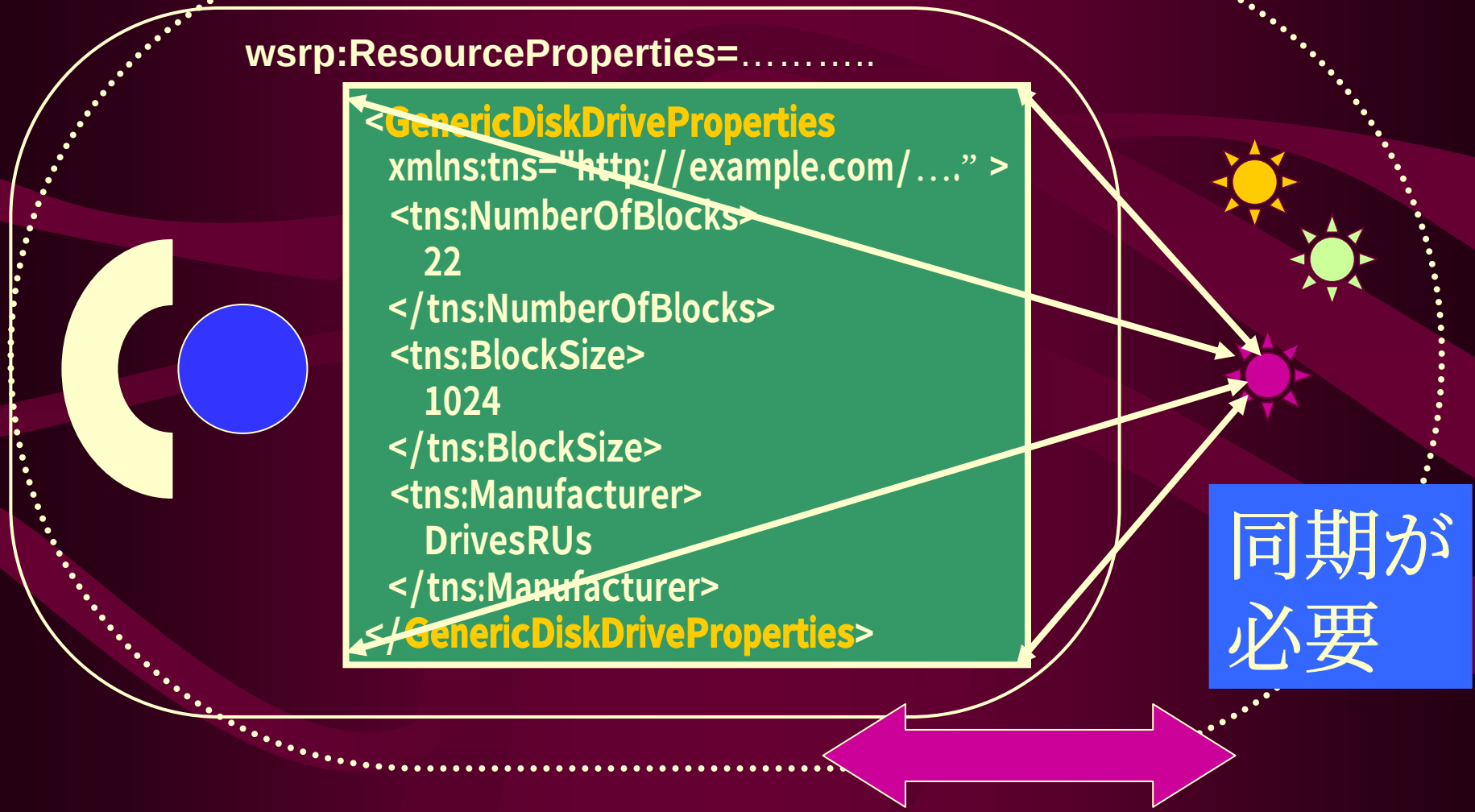
WS-Resourseの「状態」としての WS-Resource Properties Document

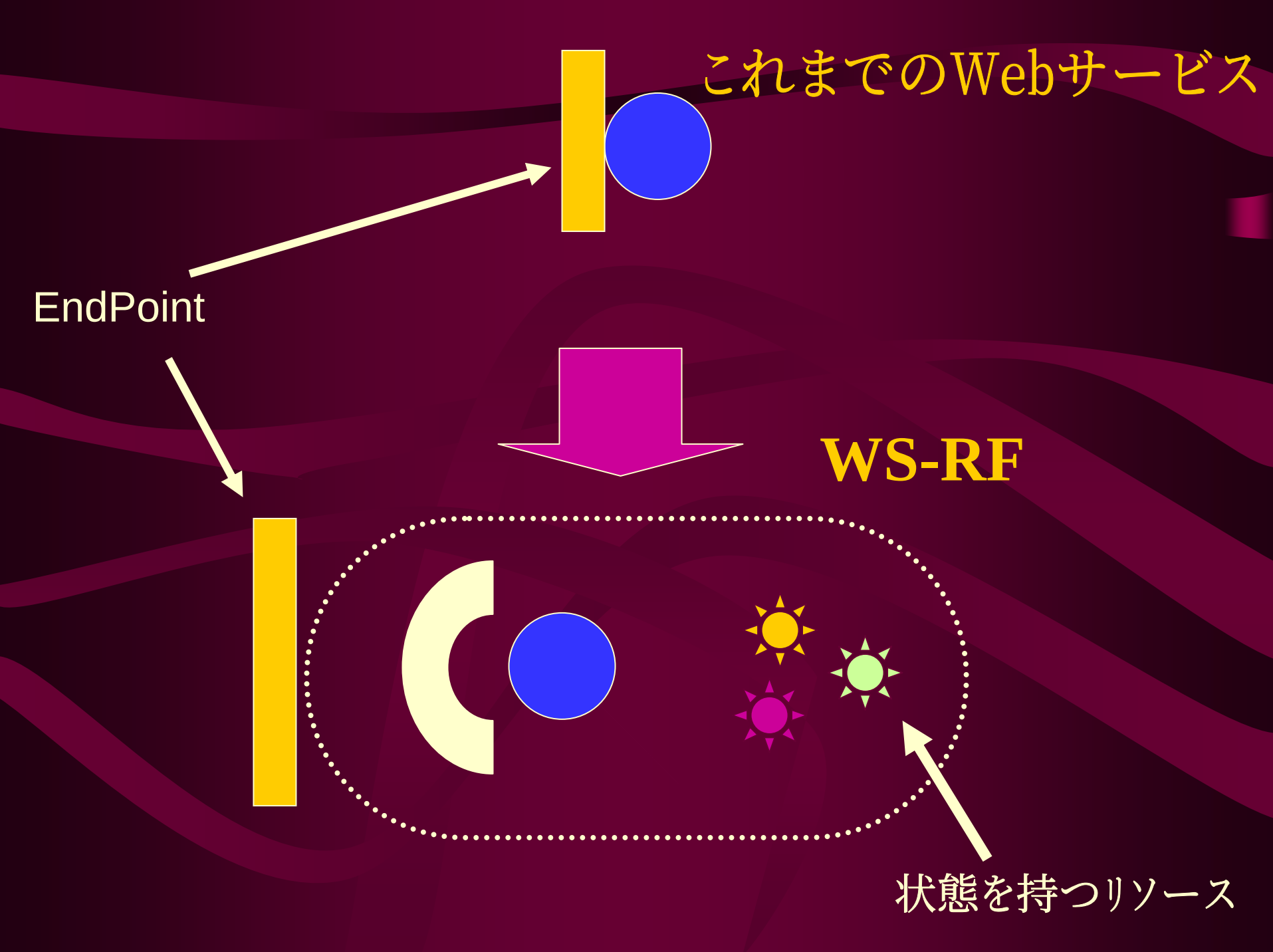
WSDLのportTypeでResource Properties Documentoを定義

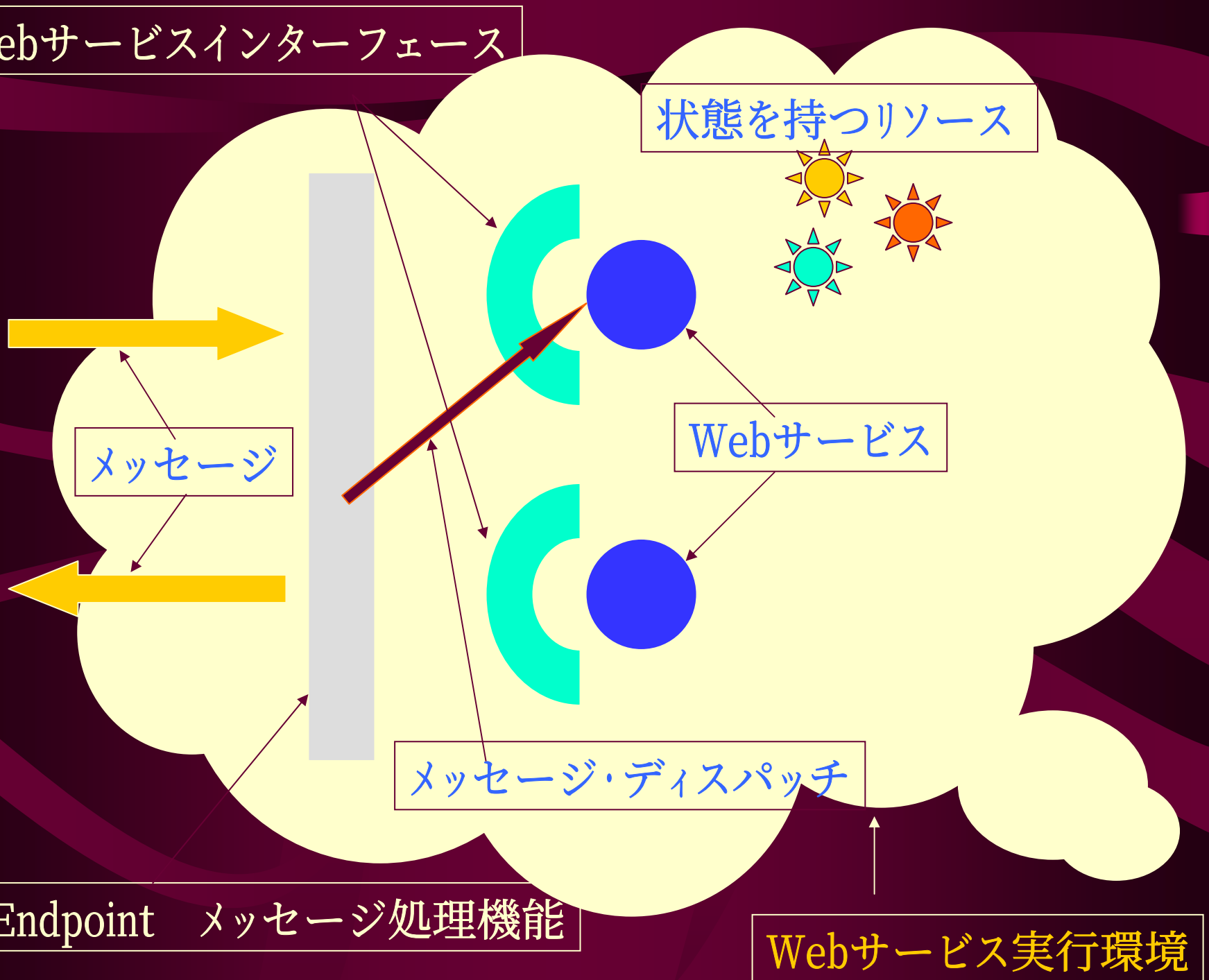
```
<GenericDiskDriveProperties
  xmlns:tns="http://example.com/….” >
  <tns:NumberOfBlocks>
     22
  </tns:NumberOfBlocks>
  <tns:BlockSize>
     1024
  </tns:BlockSize>
  <tns:Manufacturer>
     DrivesRUs
  </tns:Manufacturer>
</GenericDiskDriveProperties>
```

```
<!-- Association of resource properties document to a portType -->
<wsdl:portType name="GenericDiskDrive"
wsrp:ResourceProperties="tns:GenericDiskDriveProperties" >
```

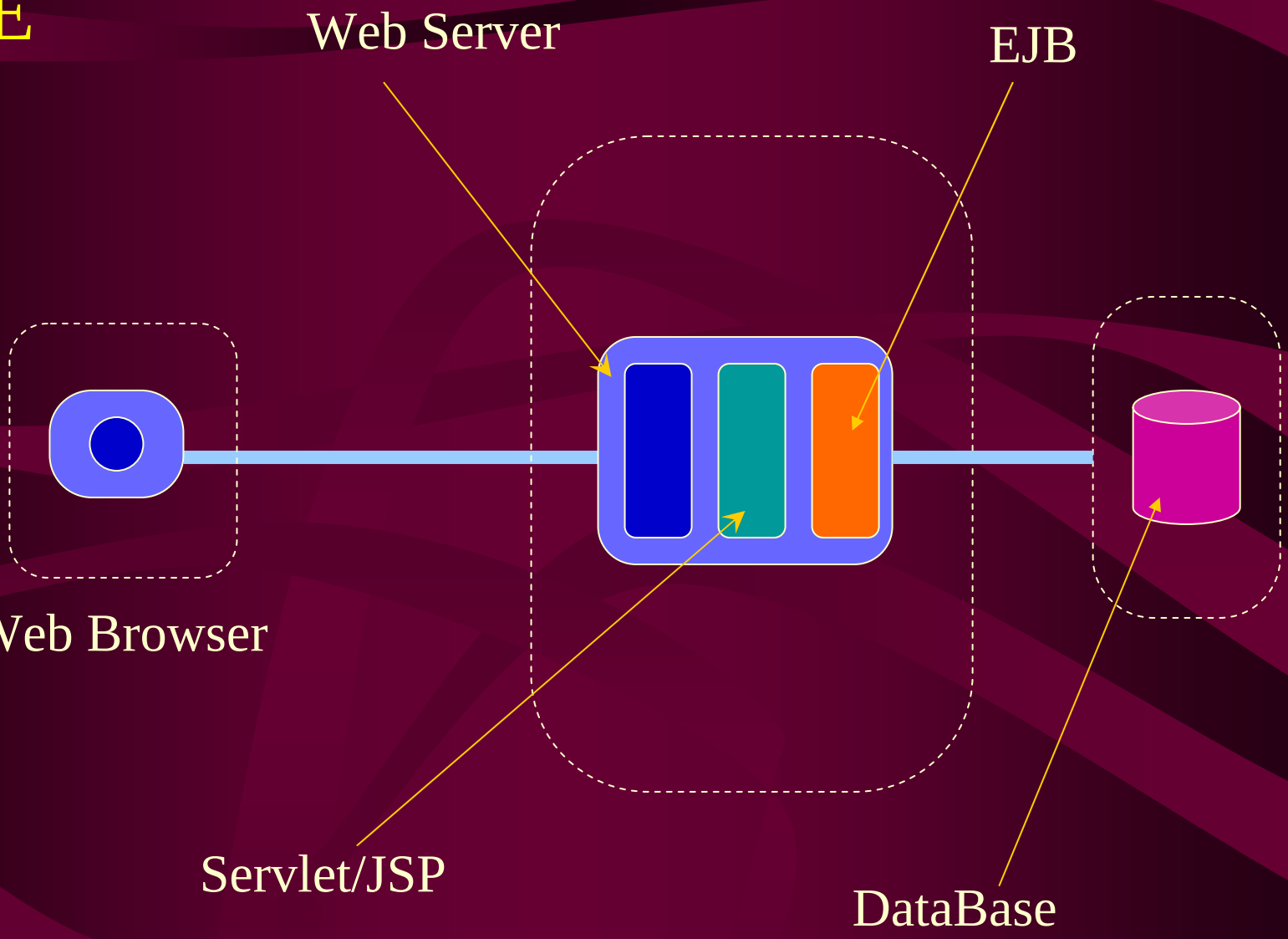
Statefull-Resourseの「**Projection**」としての WS-Resource Properties Document



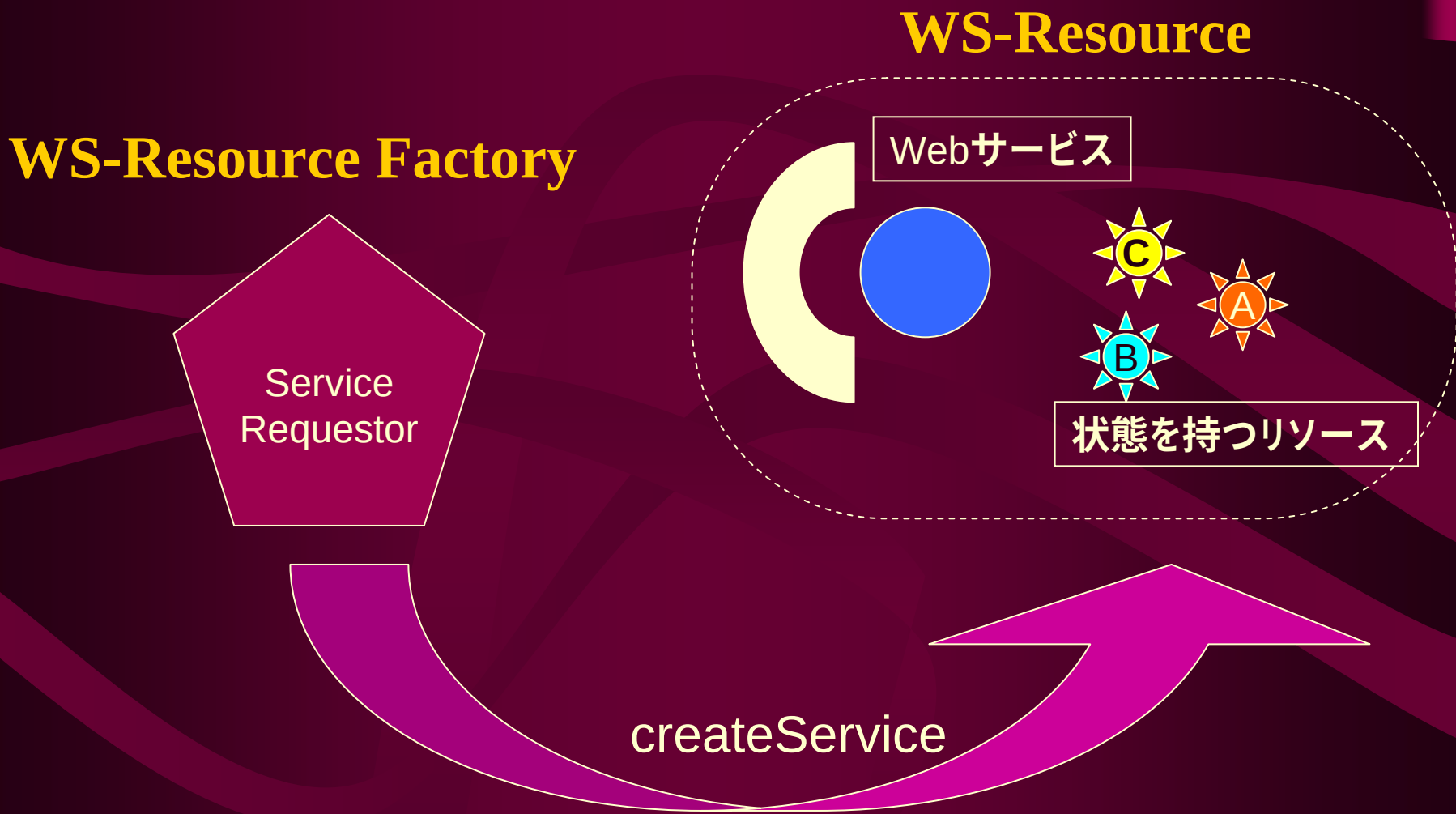

これまでのWebサービス
EndPoint
WS-RF
状態を持つリソース

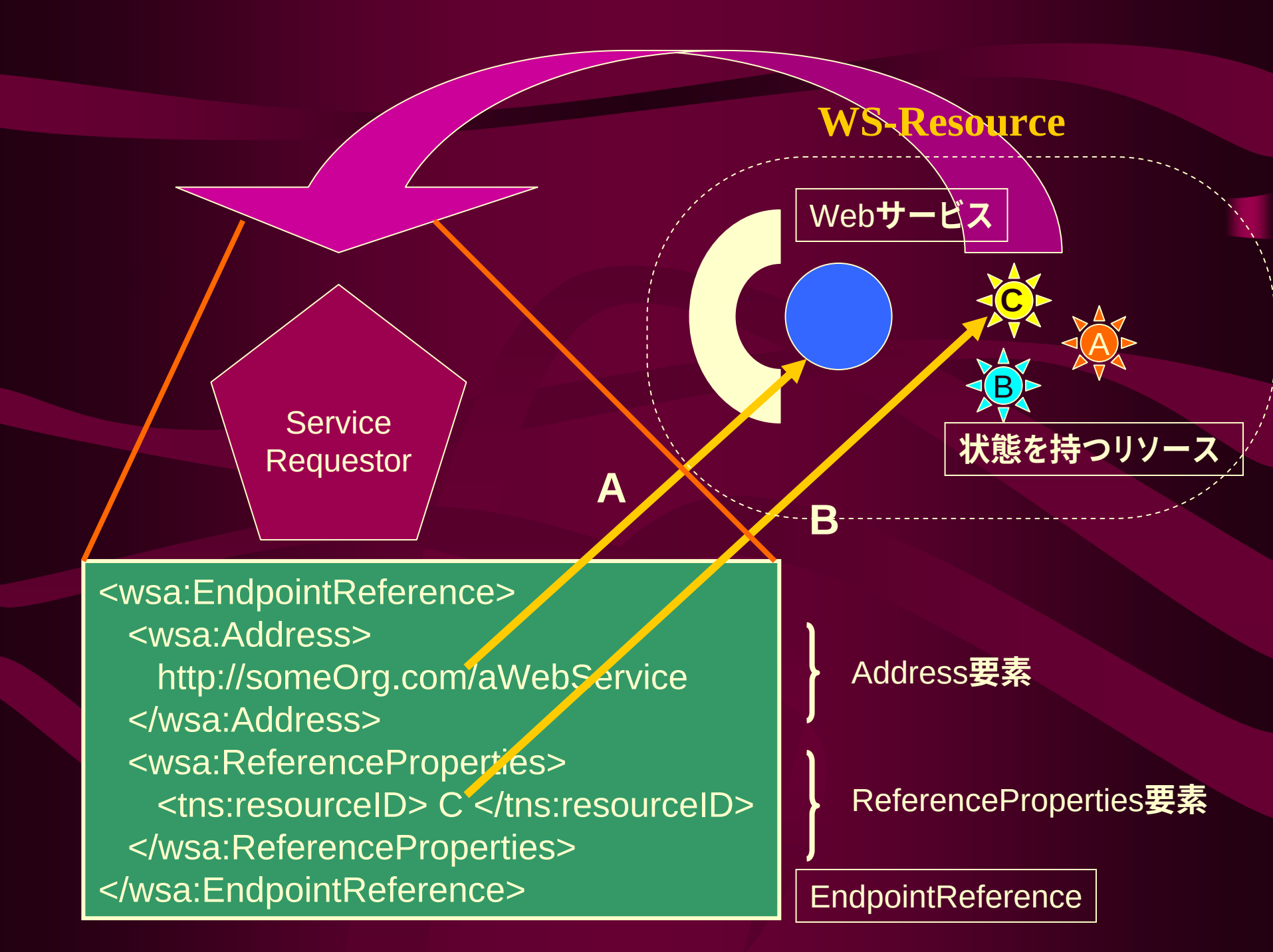

Webサービスインターフェース
状態を持つリソース
メッセージ
Webサービス
メッセージ・ディスパッチ
Endpoint　メッセージ処理機能
Webサービス実行環境

J2EE

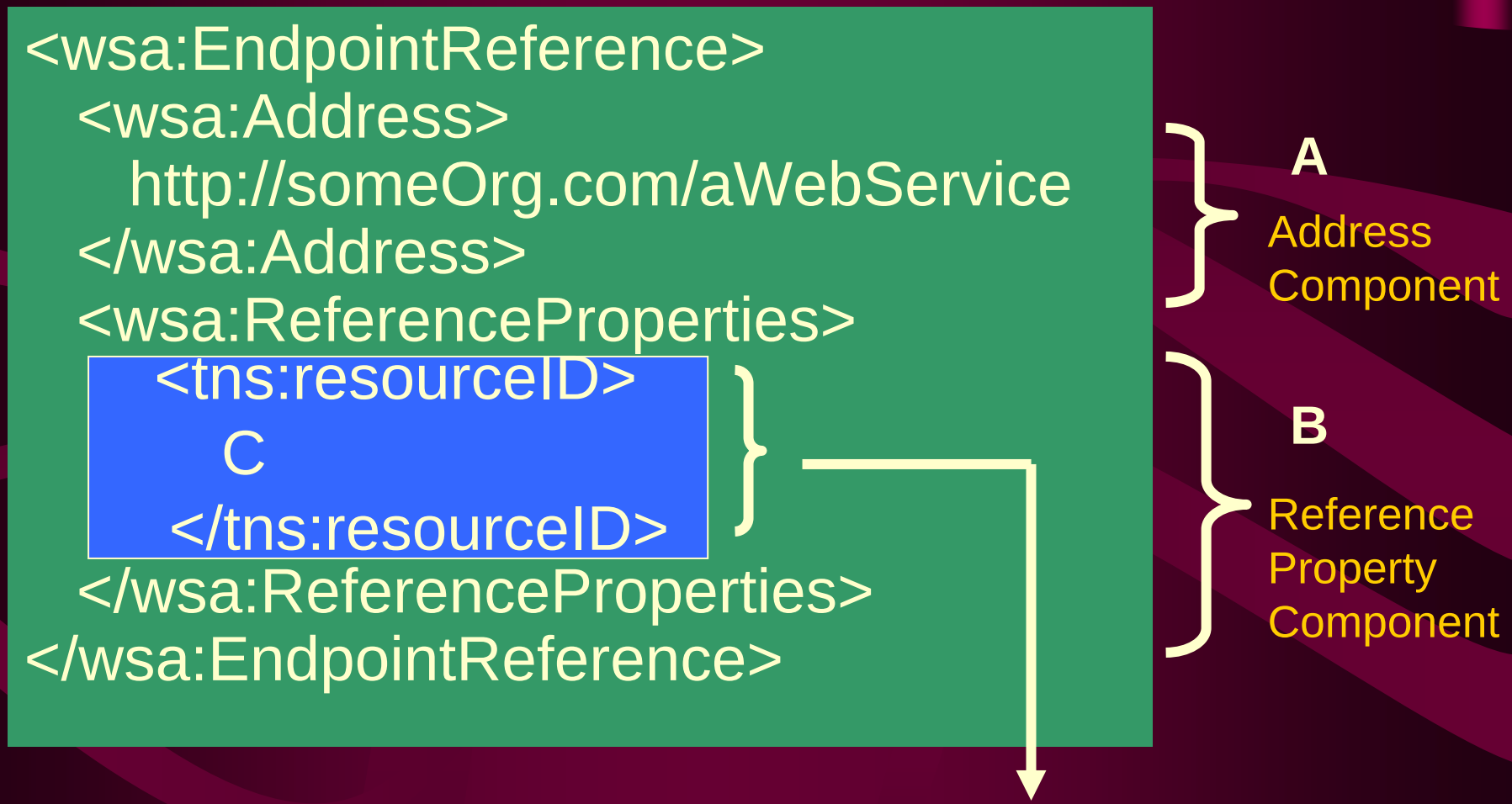


FactoryによるWS-Resourceの生成は、EndpointReferenceを返す




WS-Resource
Webサービス
C
A
B
状態を持つリソース
Service
Requestor
A
B
<wsa:EndpointReference>
<wsa:Address>
http://someOrg.com/aWebService
</wsa:Address>
<wsa:ReferenceProperties>
<tns:resourceID> C </tns:resourceID>
</wsa:ReferenceProperties>
</wsa:EndpointReference>
Address要素
ReferenceProperties要素
EndpointReference

WS-Resource-qualified EndpointReferenceと WS-Resource Context



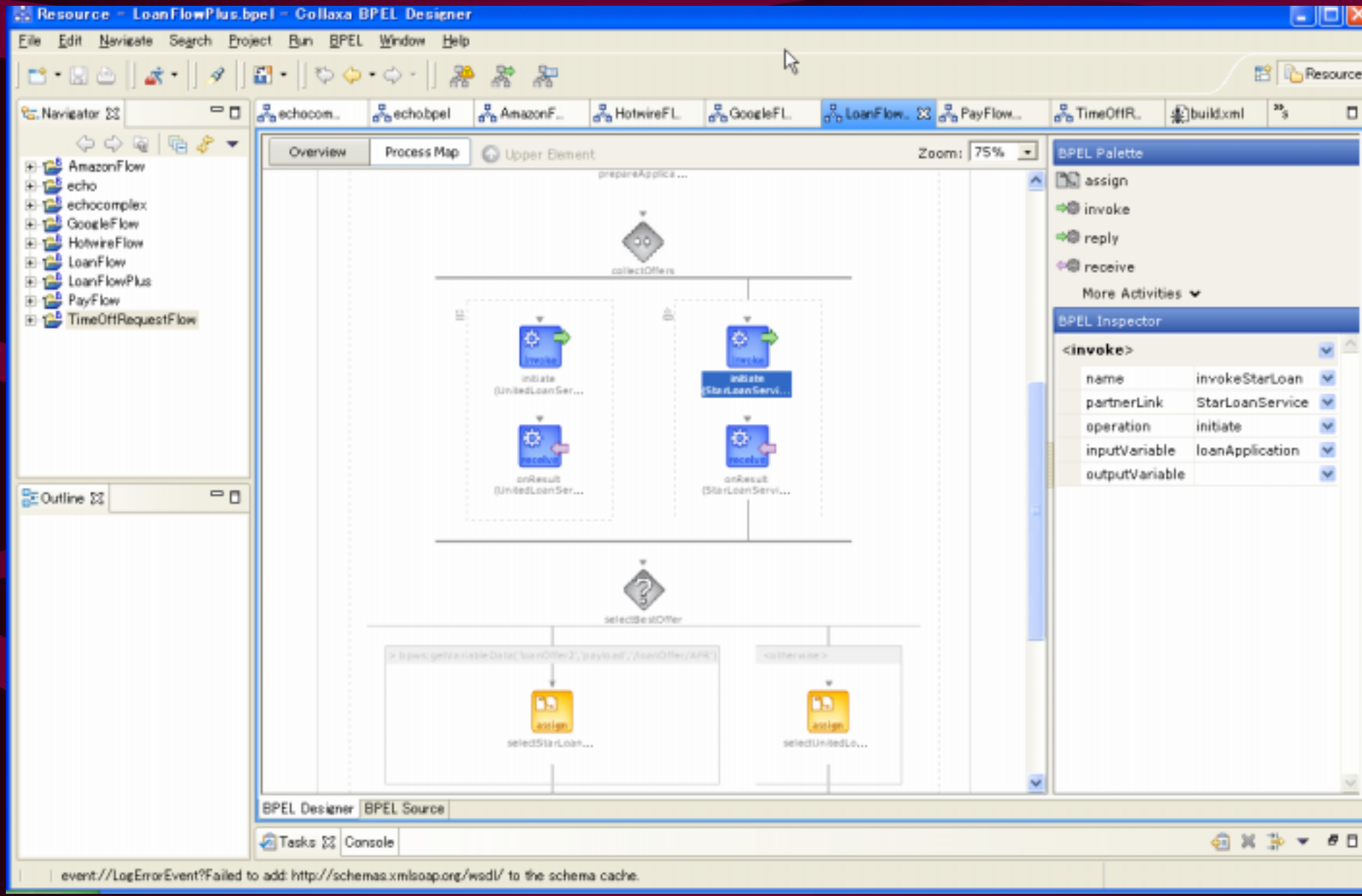
4. J2EEとビジネスプロセスの統合
J2EE / Webサービス / Grid

- **Webサービスとビジネス･プロセスの統合**
- **Business Processの記述**
- **BPEL4WS**
- **J2EEとビジネスプロセスの統合**
  -- Java Business Integration **JSR208** –
- まとめ

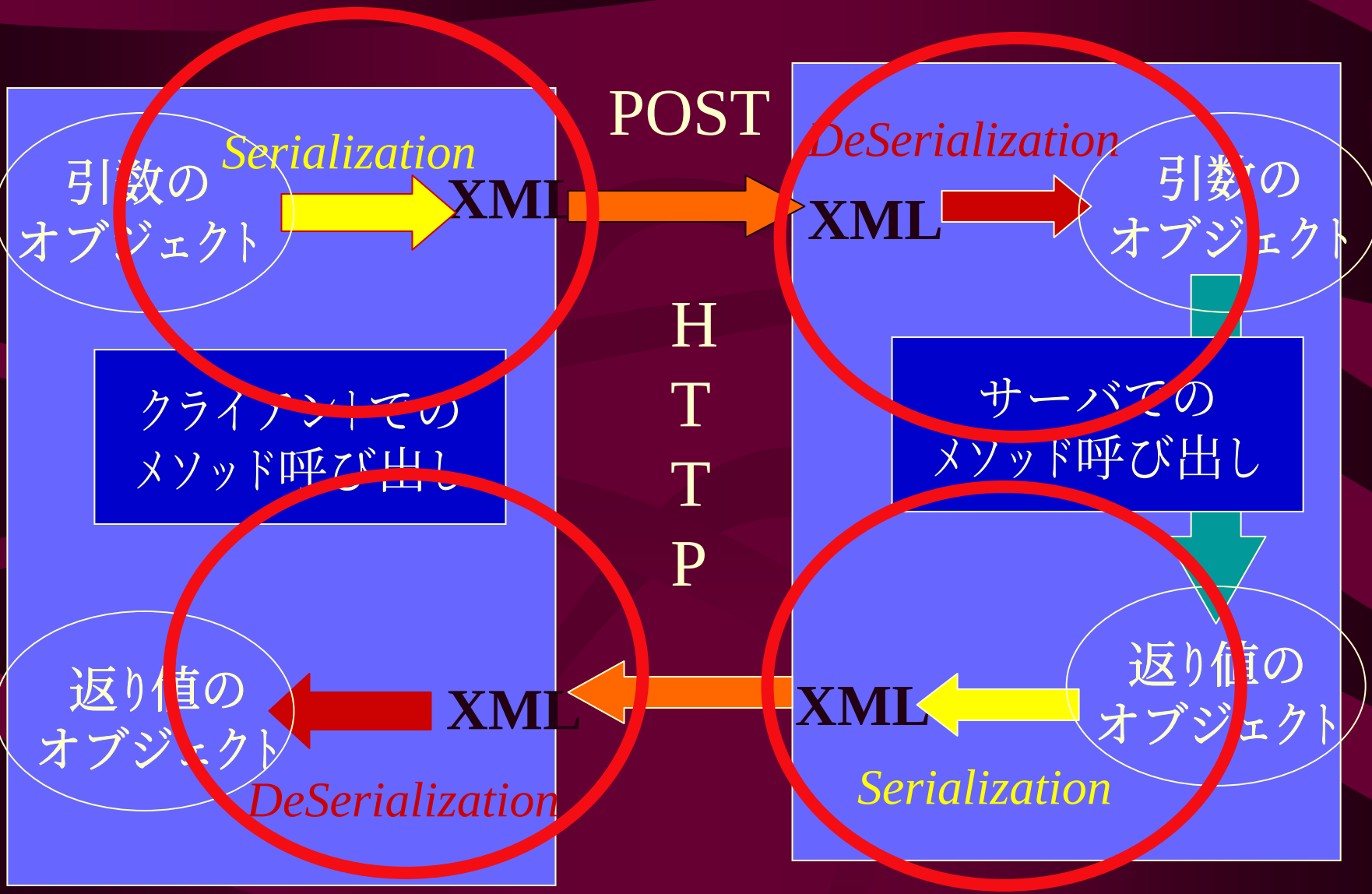
Webサービスと
ビジネス･プロセスの統合

Webサービスの「統合」の技術的背景

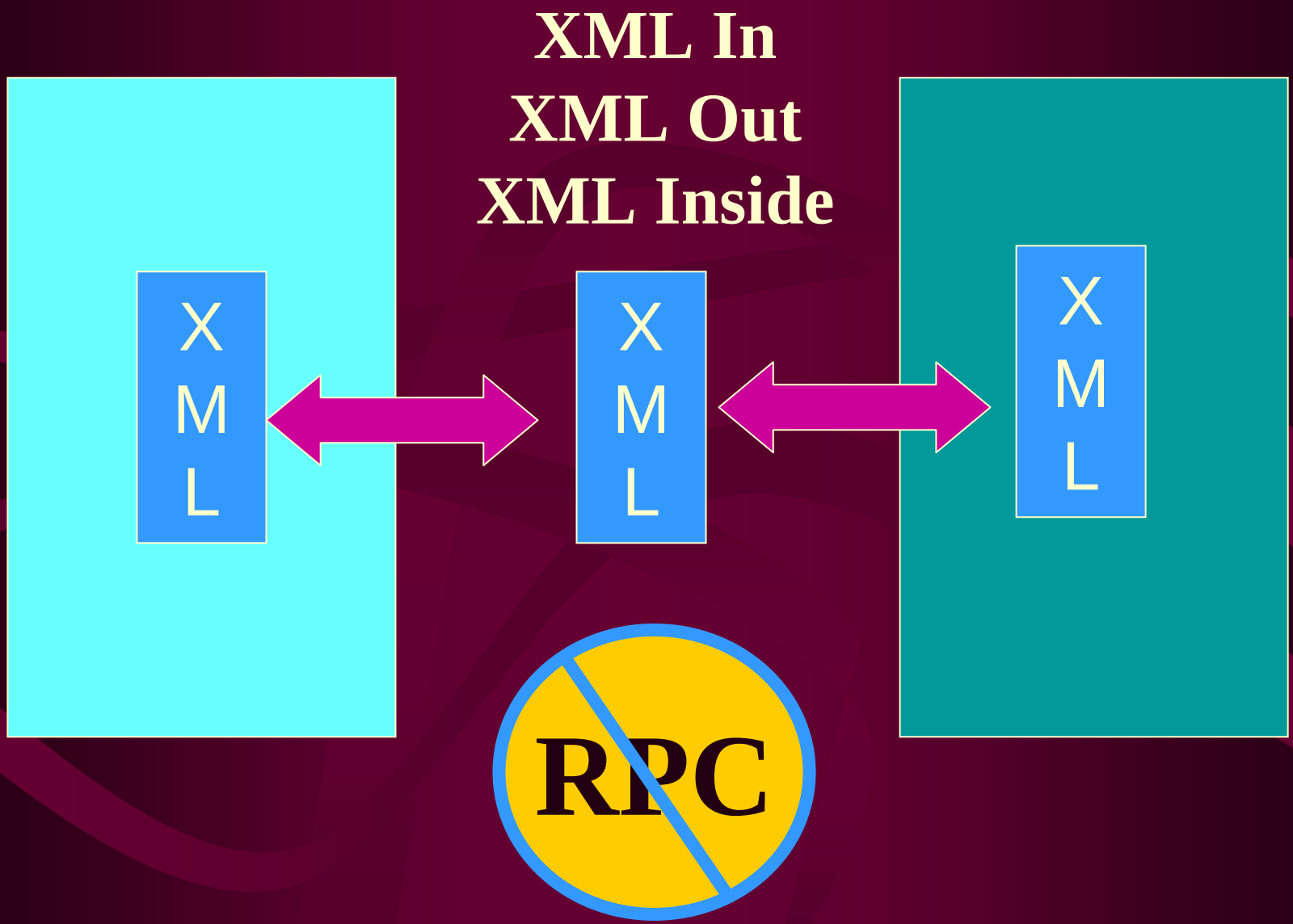
- テクニカルには、Webサービス間の複雑な相互関係を記述する必要が生まれている。
- Webサービスのアーキテクチャが、単なるRPCを超えて発展しつつあること
- Document centricな処理モデルが、信頼性の高い疎結合の「ビジネス統合」のメカニズムとして登場しつつあること


Resource - LoanFlowPlus.bpel - Collaxa BPEL Designer
File Edit Navigate Search Project Run BPEL Window Help
Resource
Navigator
AmazonFlow
echo
echocomplex
GoogleFlow
HotwireFlow
LoanFlow
LoanFlowPlus
PayFlow
TimeOffRequestFlow
Outline
echocom...
echo.bpel
AmazonF...
HotwireFL...
GoogleFL...
LoanFlow...
PayFlow...
TimeOffR...
build.xml
Overview
Process Map
Upper Element
Zoom: 75%
collectOffers
initiate
(UnitedLoanSer...
initiate
(StarLoanServi...
onResult
(UnitedLoanSer...
onResult
(StarLoanServi...
selectBestOffer
assign
selectStarLoan...
assign
selectUnitedLo...
BPEL Palette
assign
invoke
reply
receive
More Activities
BPEL Inspector
<invoke>
name invokeStarLoan
partnerLink StarLoanService
operation initiate
inputVariable loanApplication
outputVariable
BPEL Designer
BPEL Source
Tasks
Console
event://LogErrorEvent?Failed to add http://schemas.xmlsoap.org/wsdl/ to the schema cache.

RPC　Java<->XMLマッピング



Document Centric



Business Processの記述

OrchestrationとChoreography

Web Services Orchestration

- Webサービスの相互作用の実行順序を含む
- プロセスのフローを記述する
- プロセスは、常に一つの主体によってコントロールされる

Web Services Choreography

- パブリックなメッセージの交換に関連していて、実行可能なプロセスには、直接関係しない
- 複数の主体を含んだメッセージのシーケンスを跡付ける

Orchestration：BPEL4WS

Business Process Execution Language for Web Services

IBM, Microsoft, BEAが仕様策定

Choreography : WSCI

Web Services Choreography Interface

Sun, SAP, Intalio, BEAが仕様策定

OASIS Web BPEL TC

IBM, Microsoft, BEA, Oracle, Sun, SAP

BPEL4WS

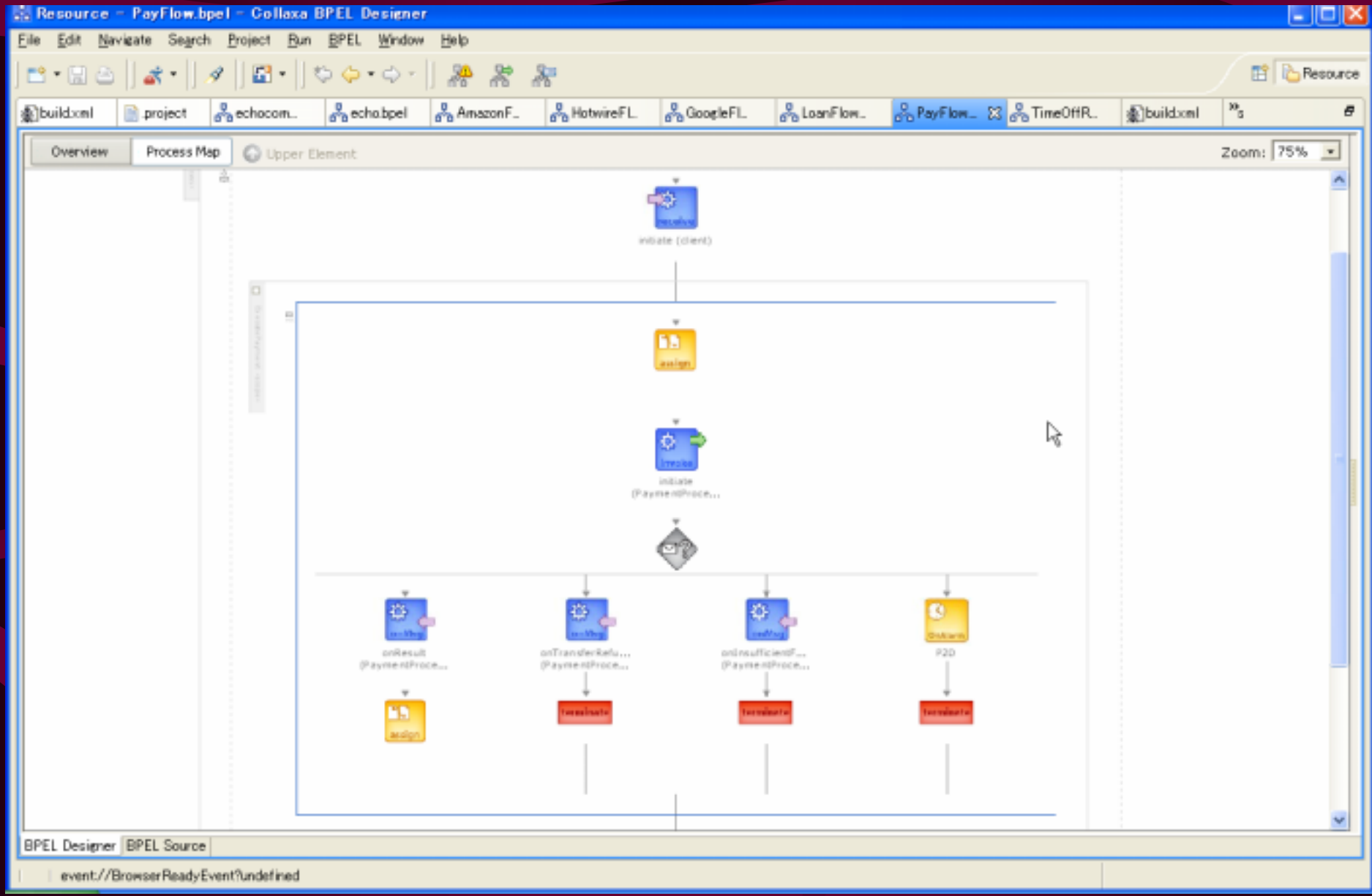
Business Process Execution Language for Web Services

http://www-106.ibm.com/developerworks/library/ws-bpel/

BPEL4WSとは？

- 複数のWebサービスをいかに結合して新しいWebサービスを提供するかを規定する

- 同じ言語が、実行可能なプロセスと抽象的なプロセスの双方を定義するために定義されている
  - 実行可能なプロセスは、ワークフローを実行するのに必要なすべてのものを記述している
  - 抽象的なプロセスは、メッセージ交換に基づいたワークフローに必要とされる観察可能な振る舞いを記述する（ビジネスパートナー間の契約の妥当性をチェックすることが出来る）

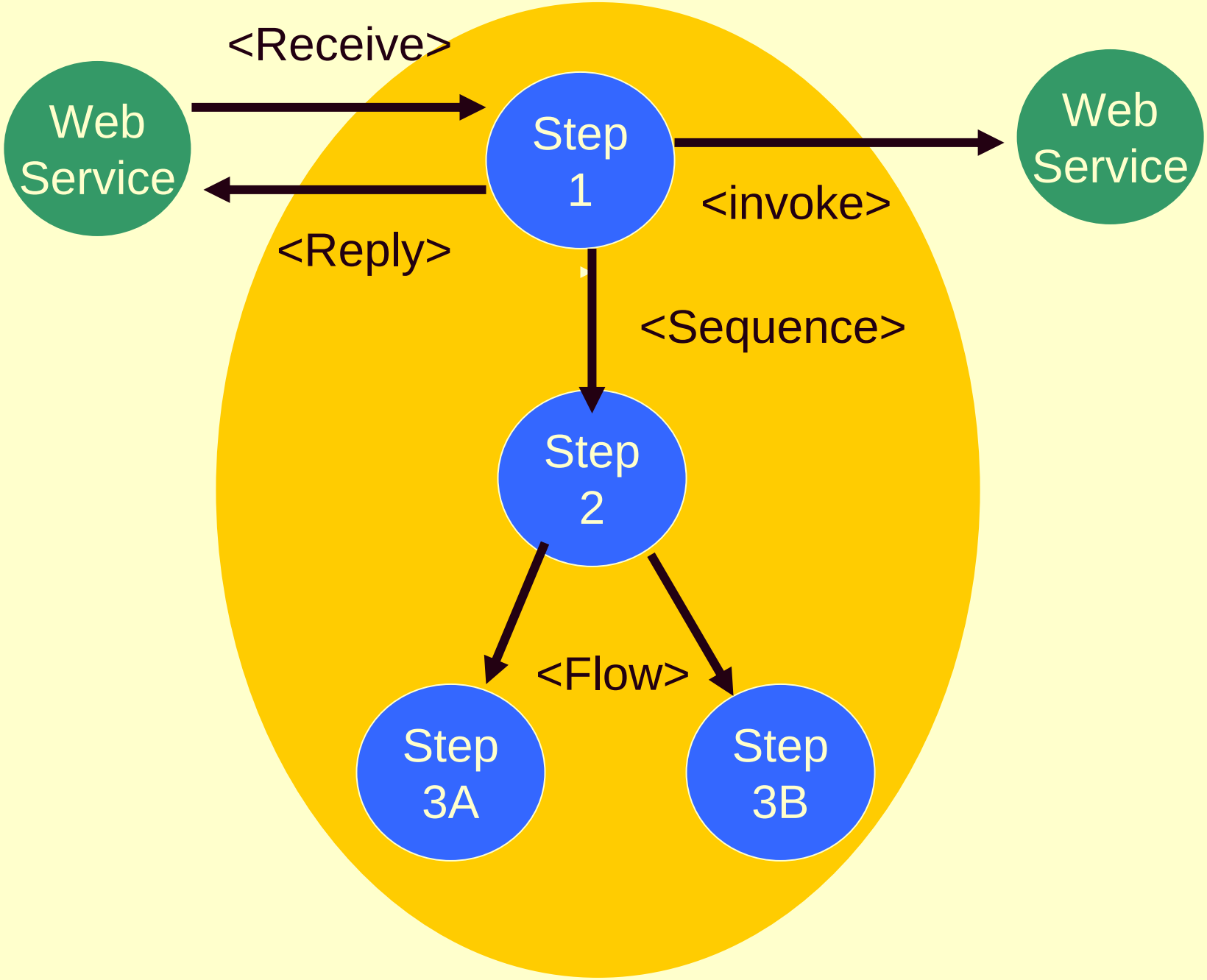

Resource - PayFlow.bpel - Collaxa BPEL Designer
File Edit Navigate Search Project Run BPEL Window Help
Resource
build.xml project echocom... echo.bpel AmazonF... HotwireFL... GoogleFL... LoanFlow... PayFlow... TimeOffR... build.xml
Overview Process Map Upper Element Zoom: 75%
initiate (client)
assign
invoke
initiate (PaymentProce...
onResult (PaymentProce...
onTransferRefu... (PaymentProce...
onInsufficientF... (PaymentProce...
P2D
assign
terminate
terminate
terminate
BPEL Designer BPEL Source
event://BrowserReadyEvent?undefined

BPEL4WSとは？

- 基本的なWebサービスの活動をサポートする：invoke, receive, reply
- Implicit lifecycle: ワークフローのインスタンスはメッセージが「スタート」とマークされワークフローエンジンに到達した時に生成される

BPELのActivityの例

- **Primitive Activities**
  - **<receive>** パートナーからのメッセージを待つ
  - **<invoke>** メッセージを発する
  - **<reply>** パートナーにメッセージを返す
  - ……
- **Structured Activities**
  - **<sequence>** 順番に実行する
  - **<flow>** 並行して実行する
  - **<pick>** メッセージに応じて一つを実行する
  - ……..


<Receive>
Web Service
Step 1
Web Service
<invoke>
<Reply>
<Sequence>
Step 2
<Flow>
Step 3A
Step 3B

<process>

<!– Definition and roles of process participants -->

<partnerlinks> ... **</partnerlinks>**

<!- Data/state used within the process -->

<variables> ... **</variables>**

<!- Properties that enable conversations -->

<correlationSets> ... </correlationSets>

<!- Exception handling -->

<faultHandlers> ... </faultHandlers>

<!- Error recovery – undoing actions -->

<compensationHandlers> ...
</compensationHandlers>

<!- Concurrent events with process itself -->

<eventHandlers> ... </eventHandlers>

<!- Business process flow -->

(activities)*

</process>

BPEL Structure

J2EEとビジネスプロセスの統合

Java Business Integration

JSR208

N.Kassem　TS3740.pdf

M.Hapner JA-SIG-12-08-Keynote.pdf

WhitePaper JBIwp70903.fm.pdf

Java Business Integration

- Java プラットフォームをSOAで拡大したもの
  - サービス間の組み合わせ
  - サービス内部の組み合わせ
- JBI アプリケーションは、統合コンポーネントの組み合わせ
- 標準的なパッケージングとデプロイのモデル

Java Business Integration

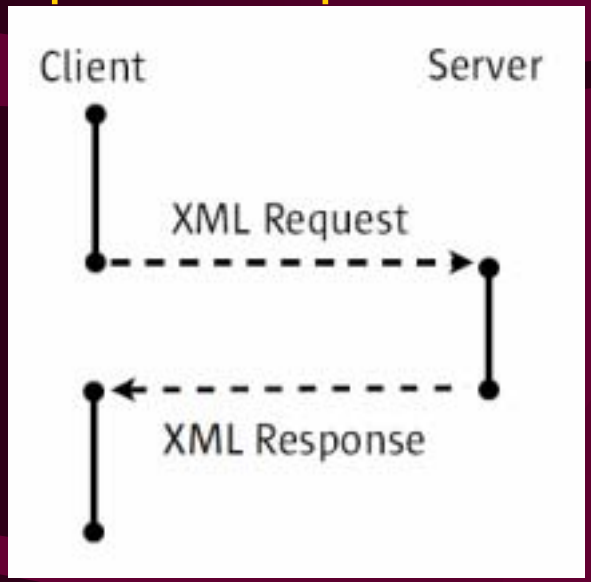
プラットフォームを拡大するための一群のSPI（システムプログラムインターフェース）の集まり

- ビジネス・プロセス・エンジン（Machine SPI）
  - ビジネス・プロセス・マシンの為の標準的な実行環境を整備する
- バインディング・フレームワーク(Binding SPI)
  - 外部のサービスとデータを交換するための拡張可能なプロトコルバインディングのセットをコンテナに提供する
  - メッセージをプロトコルバインディングに関連付ける

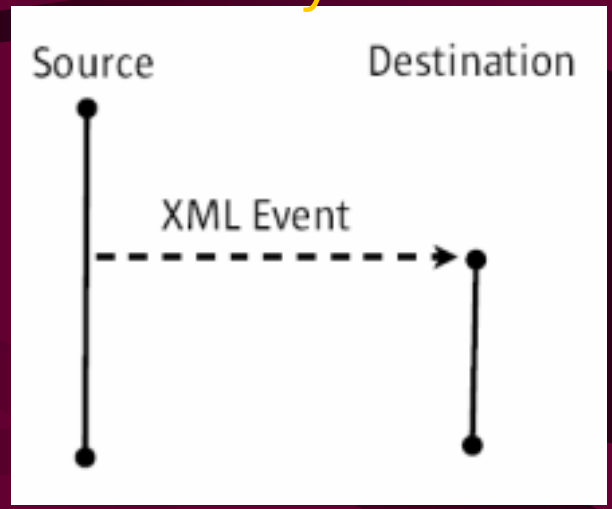
J2EE with JBI

- 拡大可能な統合プラットフォーム
- 多くのコンポーネント･モデル
- 多くのバインディング
- すべてがWSDL MEPsと、組み合わせのメカニズムを共有する

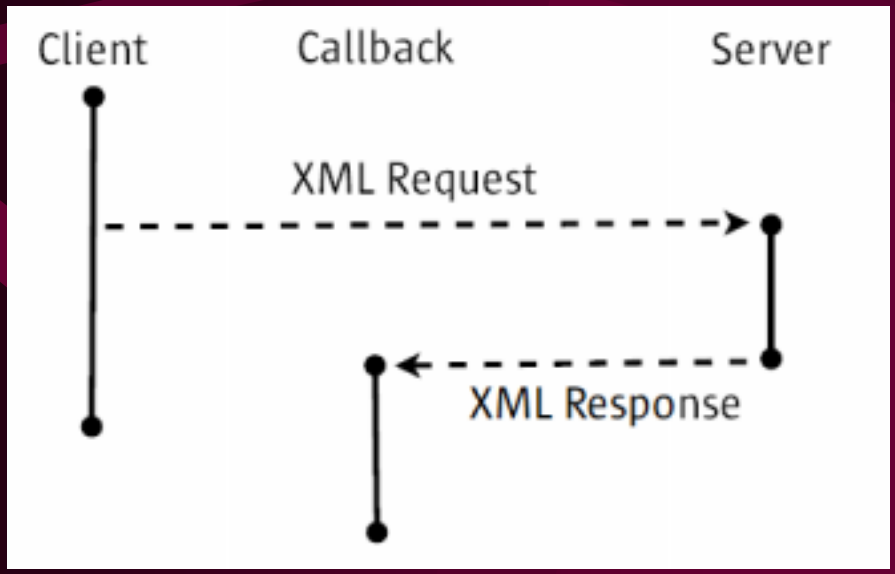
Request-Response MEP



One-Way MEP

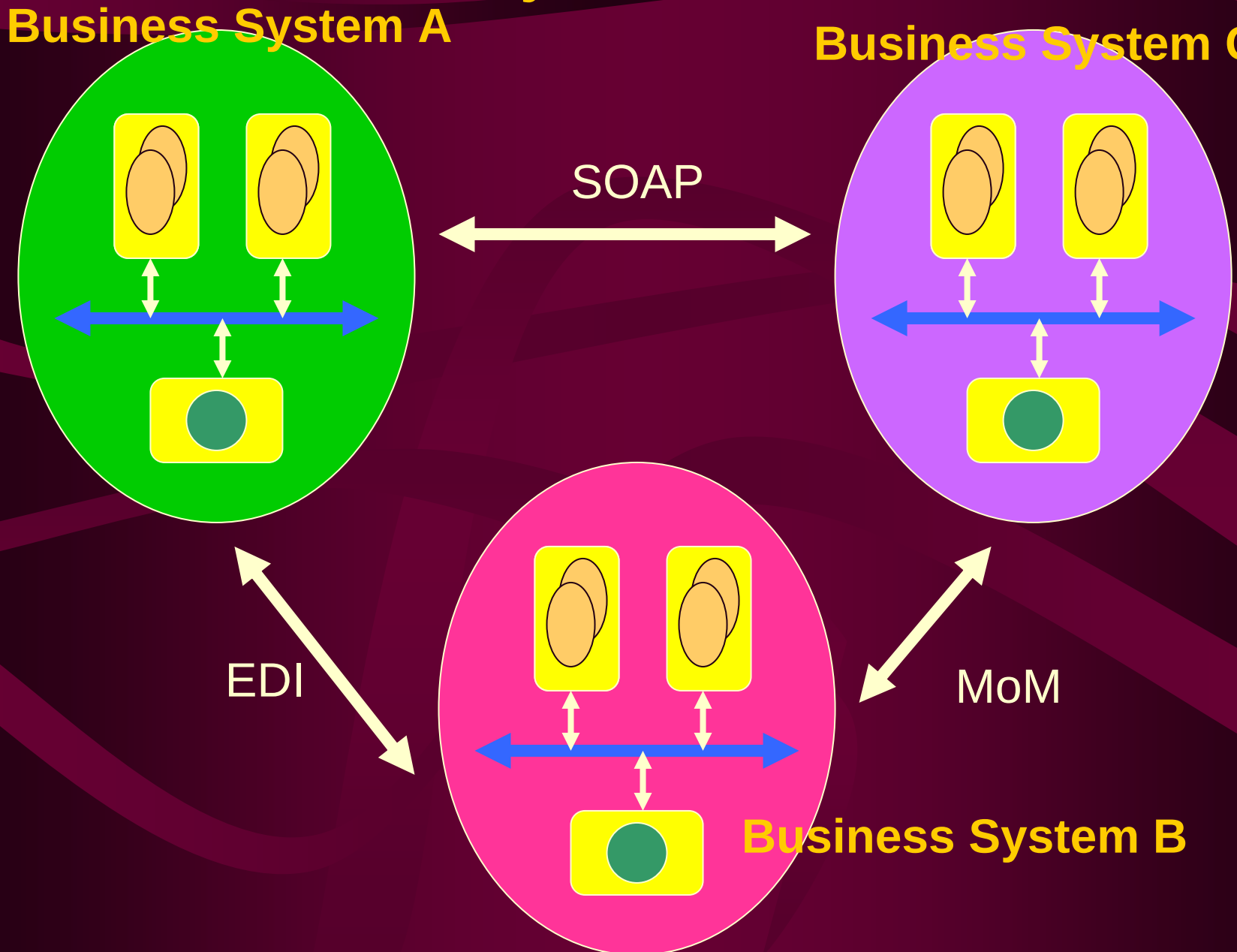


Callback MEP

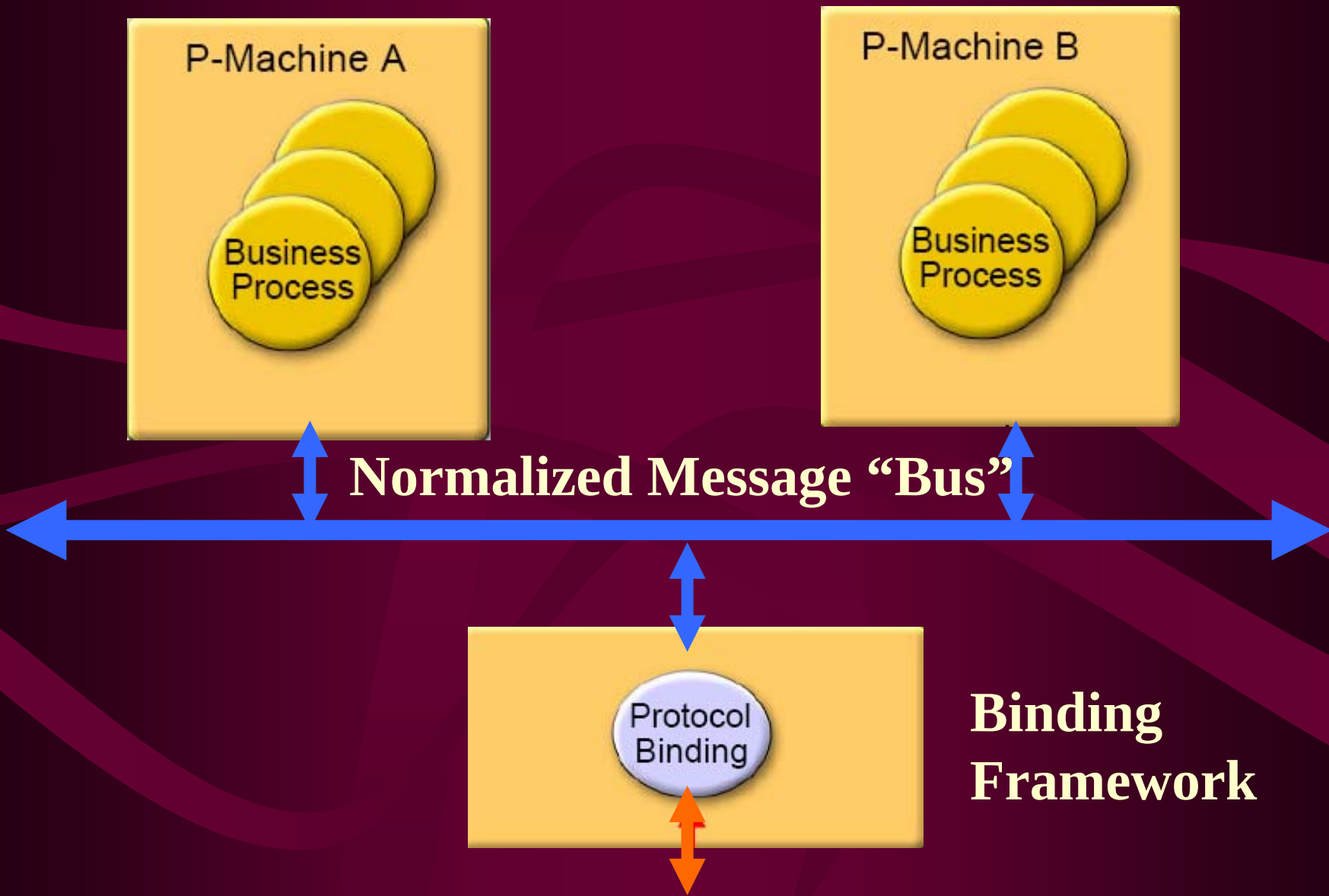


WSDL Message Exchange Patterns

JBI, System View



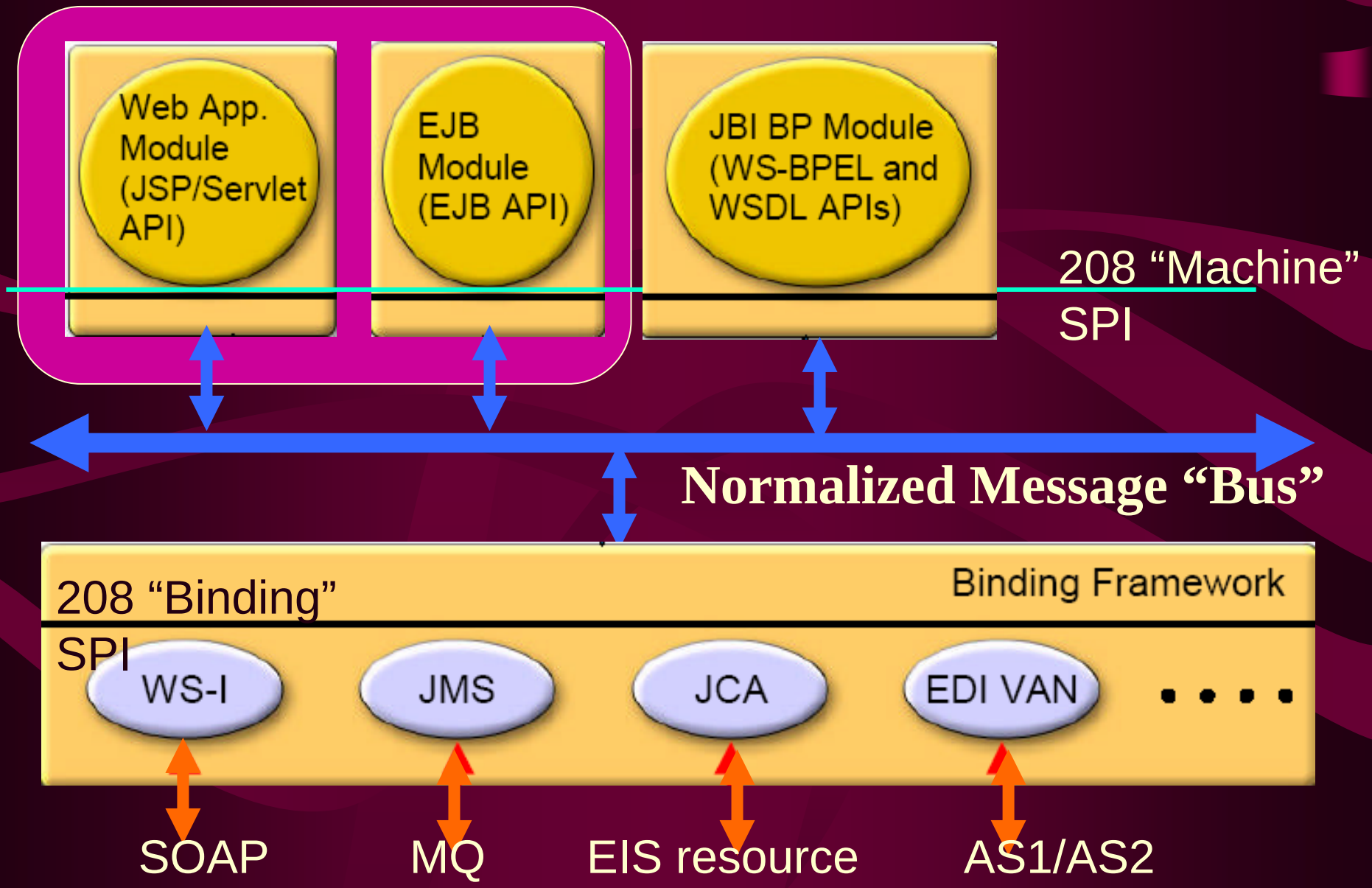
JBI, Process View



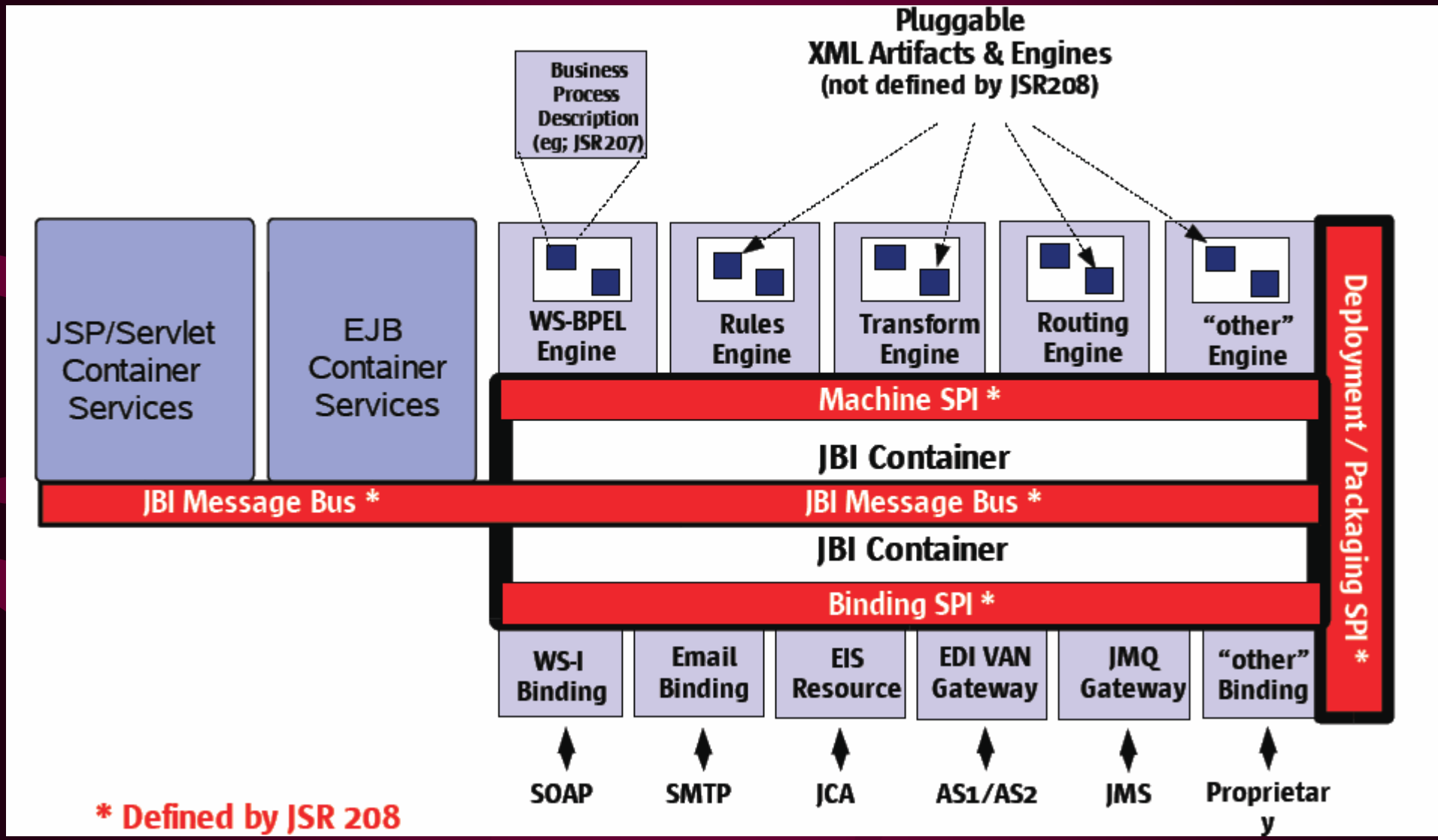
Business Process Machines（BPM)

- **BPM**は、「ビジネス・プロセス・インスタンス」をサポートし、その生存サイクルを管理する
- **BPM**は、「正規化されたメッセージ」のレベルで機能する
- 「正規化されたメッセージ」という概念は、ビジネスメッセージを効率的に処理する上で重要な役割を演ずる
- JBIは、BPMと正規化されたメッセージバスとの間の取り決めを形式化したもの

JBI and J2EE

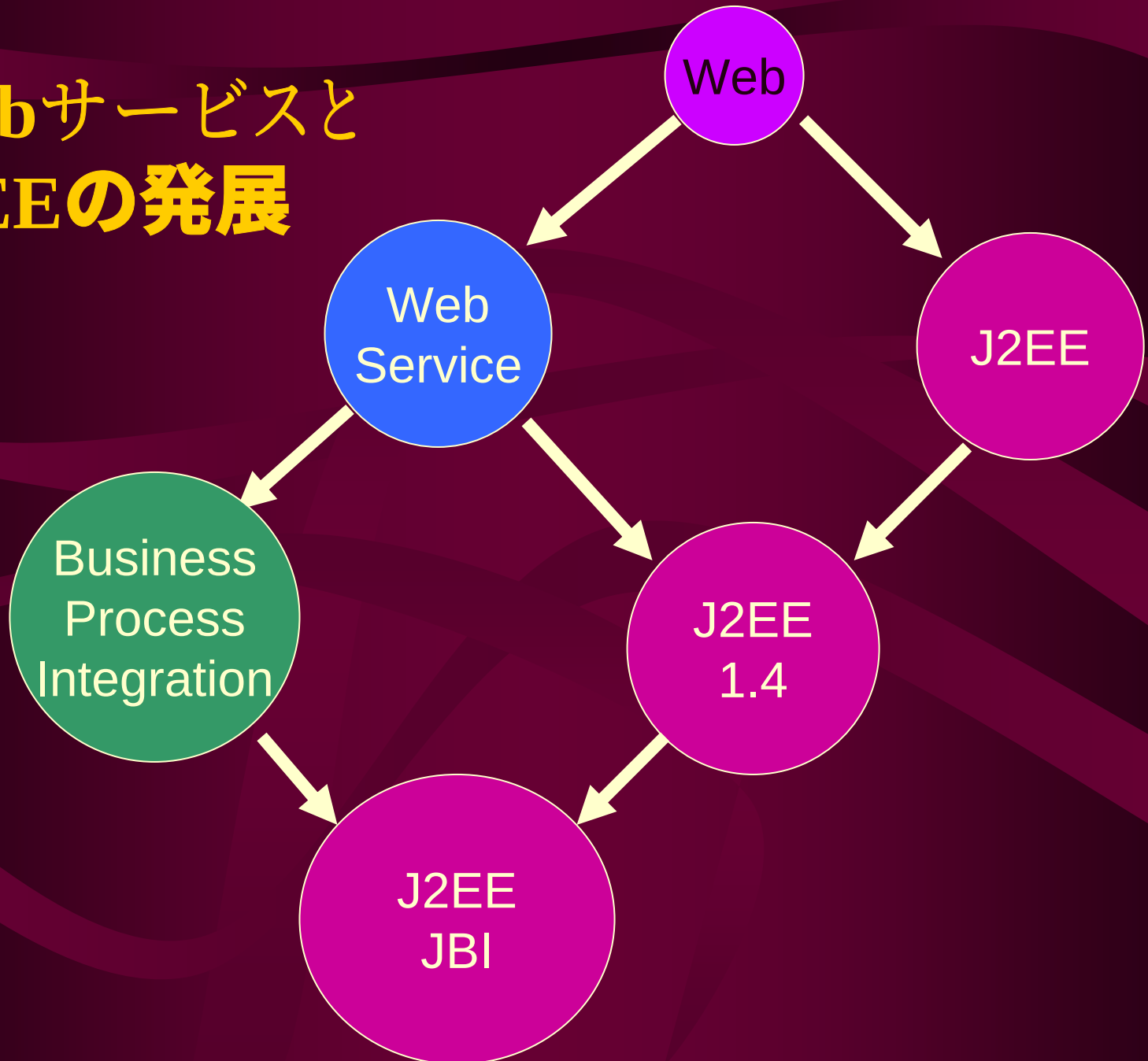

Web App. Module (JSP/Servlet API)
EJB Module (EJB API)
JBI BP Module (WS-BPEL and WSDL APIs)
208 "Machine" SPI
Normalized Message "Bus"
208 "Binding" SPI
Binding Framework
WS-I
JMS
JCA
EDI VAN
. . . .
SOAP
MQ
EIS resource
AS1/AS2

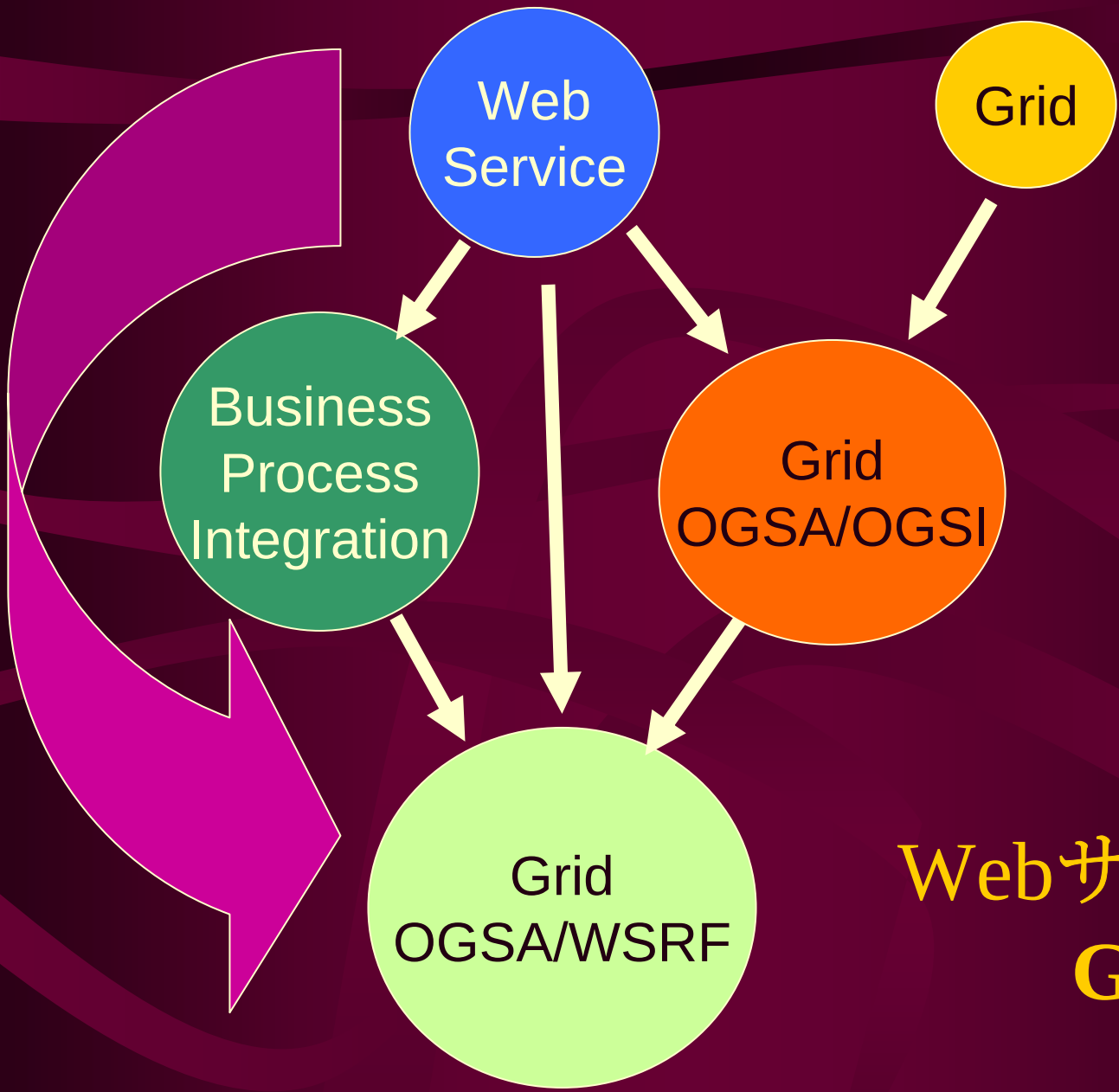
JBI Container Model



まとめ

Webサービスと J2EEの発展





Webサービスと **Grid**

Grid,WebサービスとJ2EE