



Especializada em treinamentos Java e J2EE

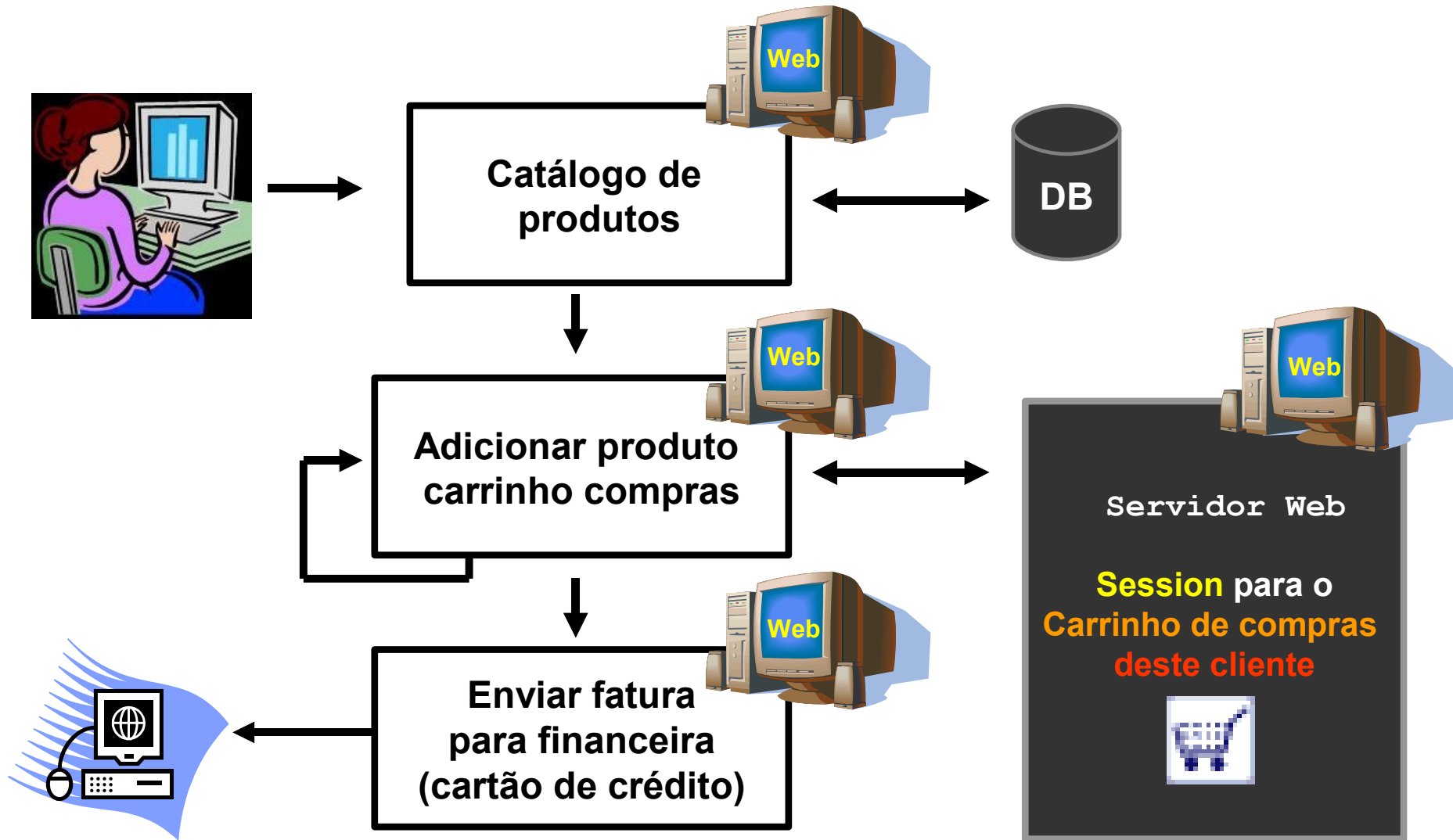
- Mais de 600 alunos treinados
- Mais de 4.000 em palestras e mini-cursos
- Instrutores certificados
- Em Florianópolis, na V.Office – f. (48) 224-8580

Agenda

- **Princípios do EJB: socket, RMI e CORBA**
- **Arquiteturas com e sem EJB**
- **Servidores EJB**
- **JNDI**
- **EJBs**

Palestrante

- **Vinicius M. Senger**
 - Trabalha com desenvolvimento de softwares há mais de 14 anos;
 - Foi instrutor e consultor Java da: Sun do Brasil, Oracle e Microsoft;
 - Palestrante em diversos eventos nacionais e recentemente aprovado no JavaOne (maior evento Java dos Estados Unidos);
 - Certificações: Java Programmer 1.4, Sun Certified Enterprise Architect P1, Sun Official Instructor, Oracle Instructor, Microsoft Certified Professional, Microsoft Certified Trainner;
 - Diretor Técnico e fundador da Globalcode;



Problemas:

- Servidor Web está responsável pela camada de apresentação que está sendo acessado por TODOS os usuários da aplicação .
- Servidor Web não deveria ser responsável por processar lógica de negócios
- O número de carrinhos de compra criado será muito grande.

Solução:

- Criar um **Pool** de objetos do tipo carrinho de compras para diminuir o tempo gasto na criação dos objetos.
- Tirar a lógica de processamento de lógica de negócios da camada web e colocá-la em um servidor intermediário.

Problemas:

- Este envio de dados representa o processamento do **pagamento** da compra do cliente, e portanto não a **tolerância a falhas é muito baixa**, precisamos de garantia de entrega.

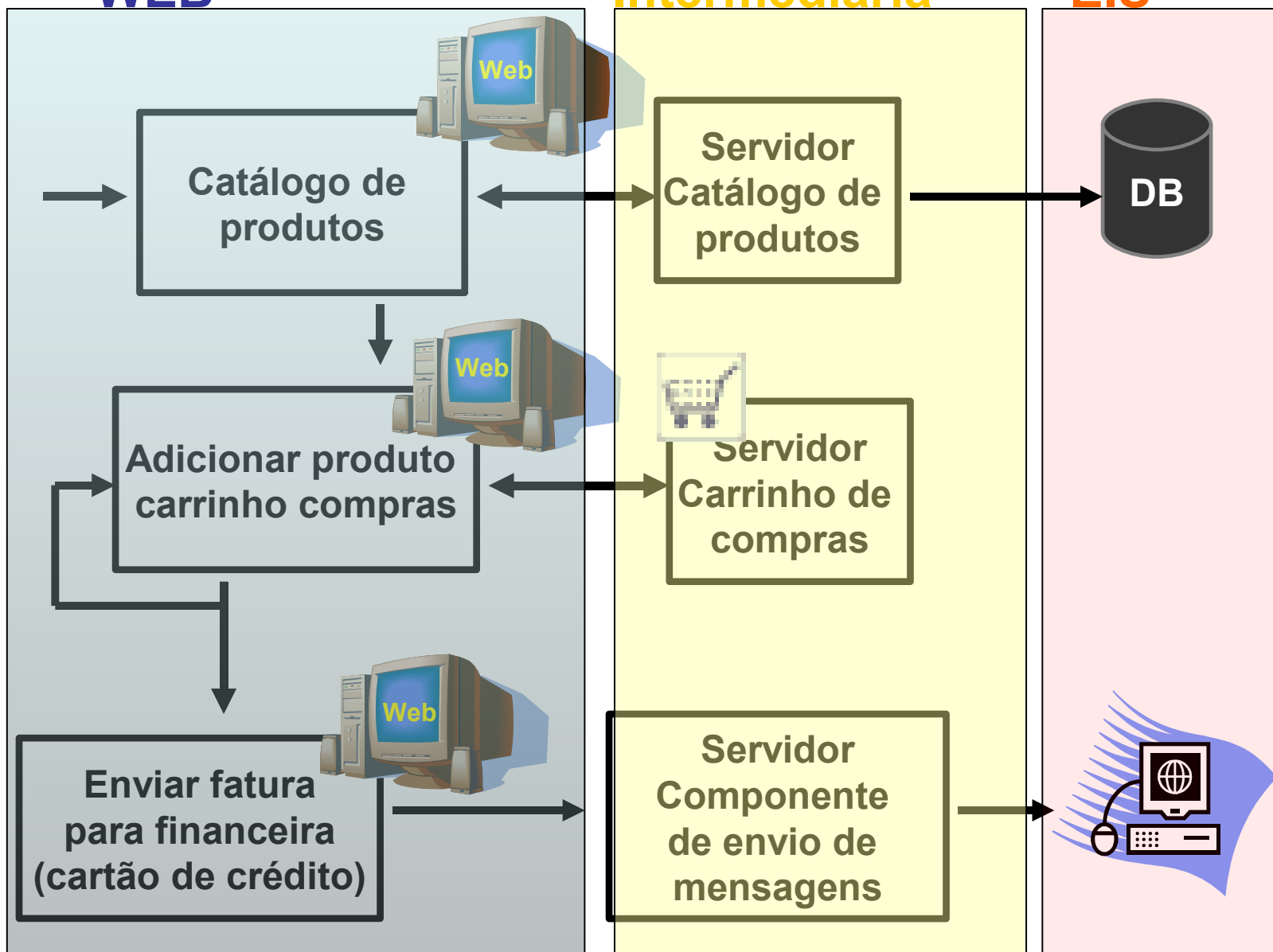
• Solução:

- Utilizar **Message Queue**

WEB

Intermediária

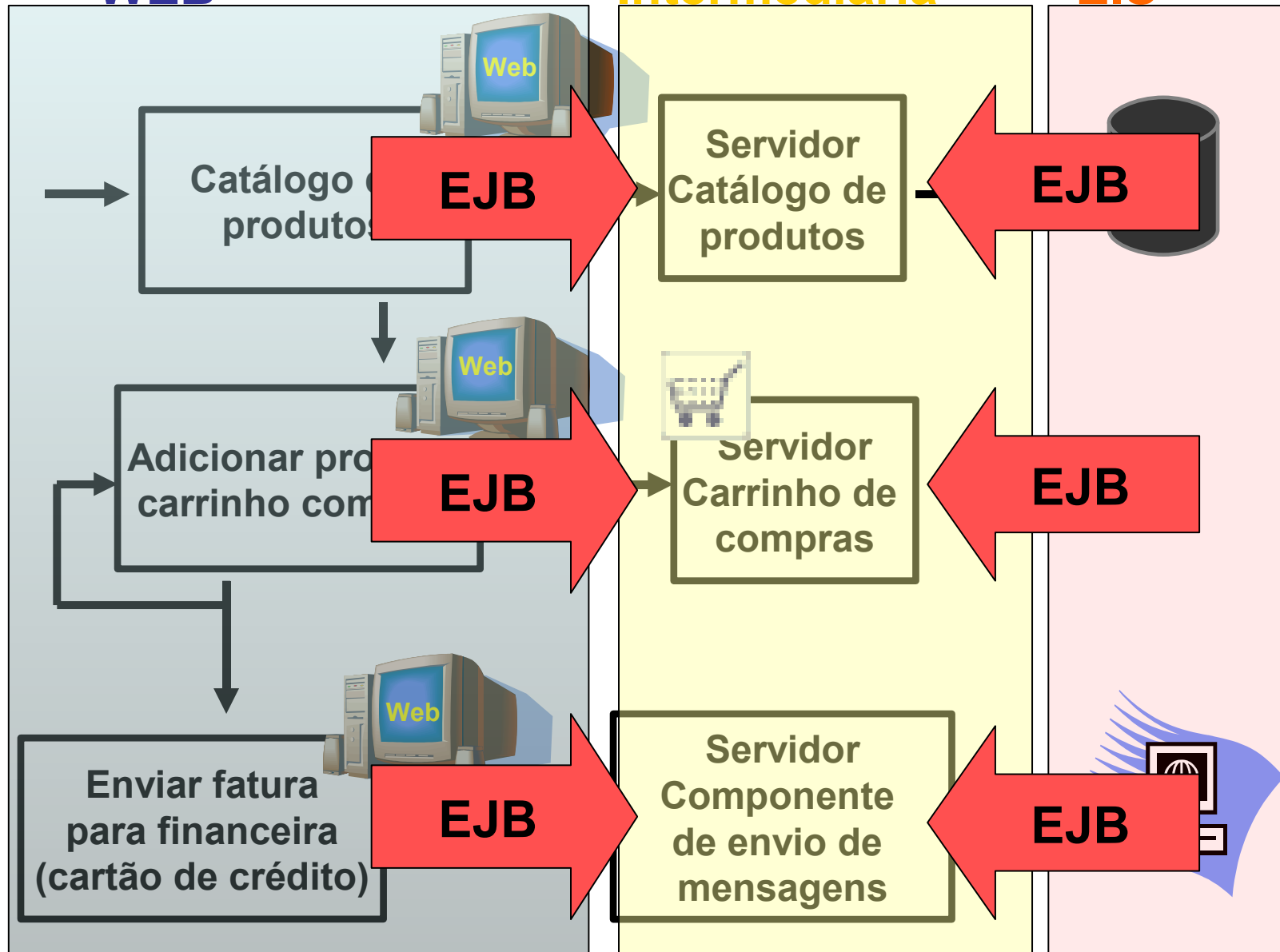
EIS



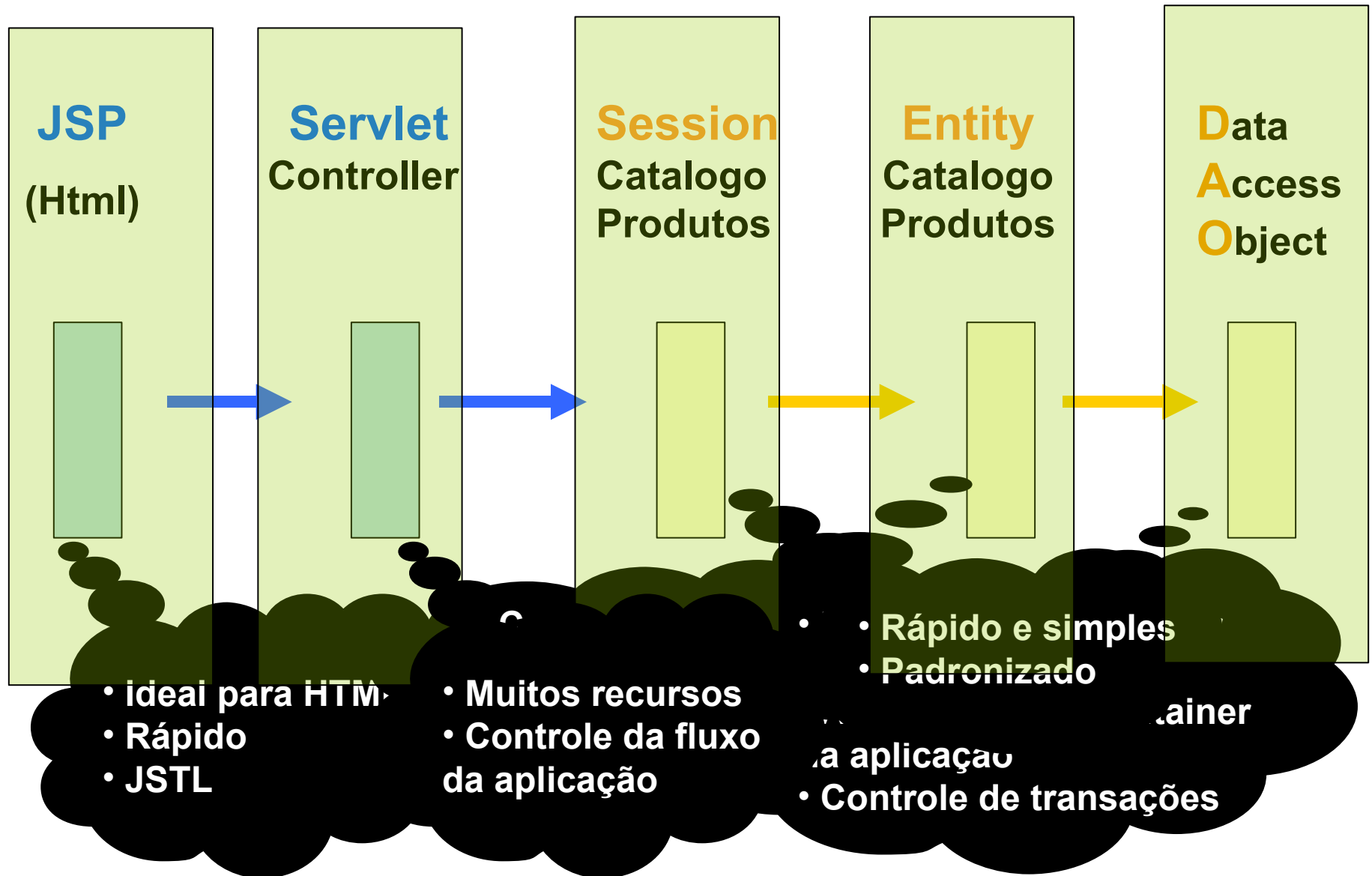
WEB

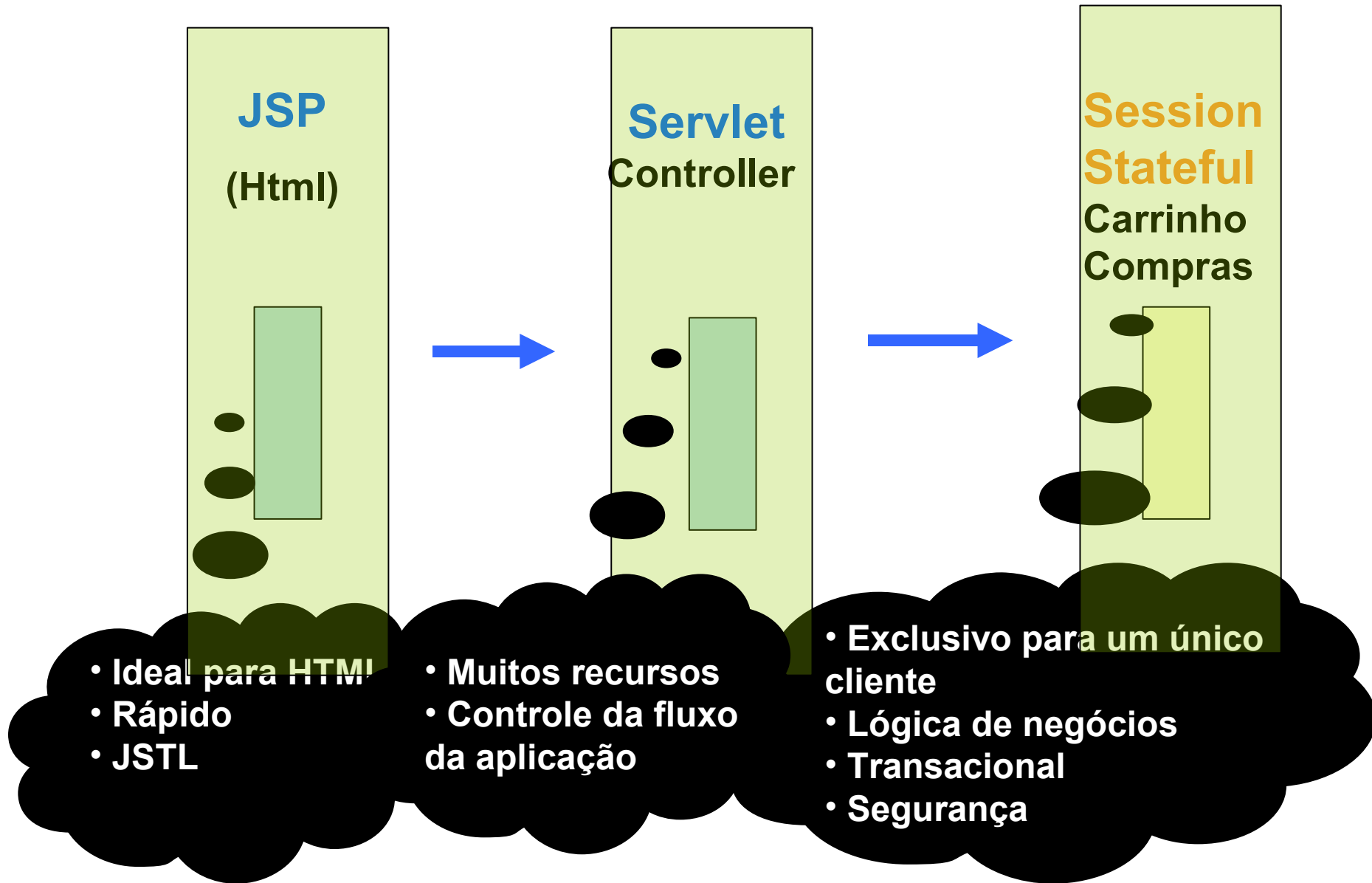
Intermediária

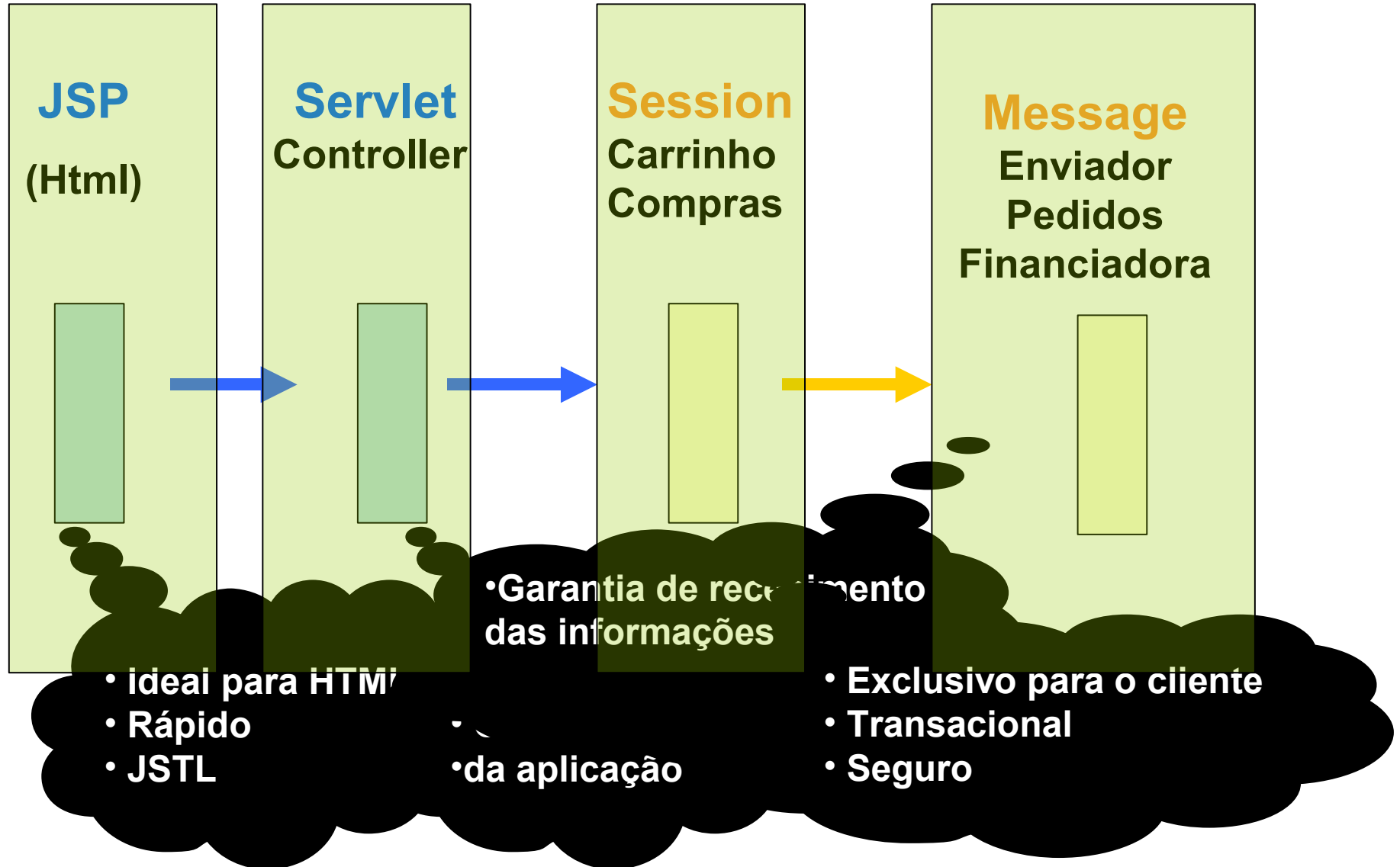
EIS



Montagem do Catálogo Produtos







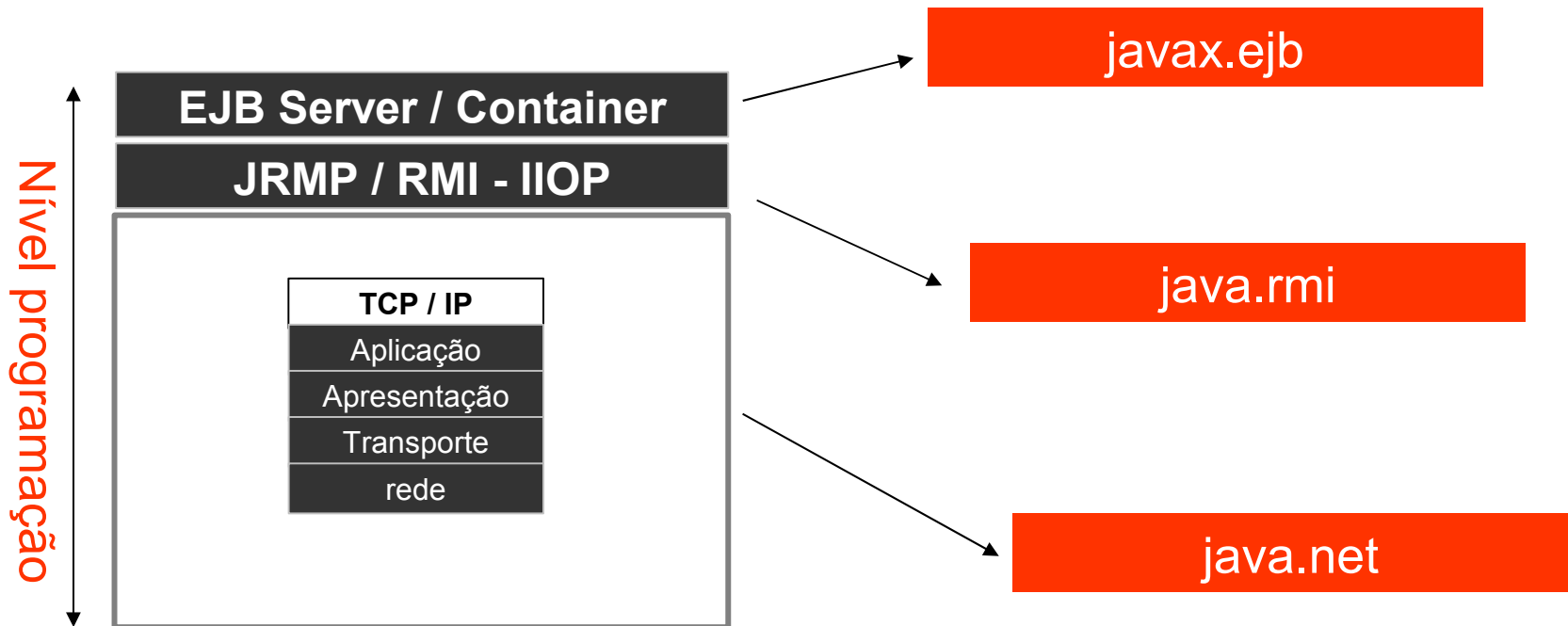
Agenda

- **Princípios do EJB: socket, RMI e CORBA**
- Arquiteturas com e sem EJB
- Servidores EJB
- JNDI
- EJBs - Session Bean
- Conclusões

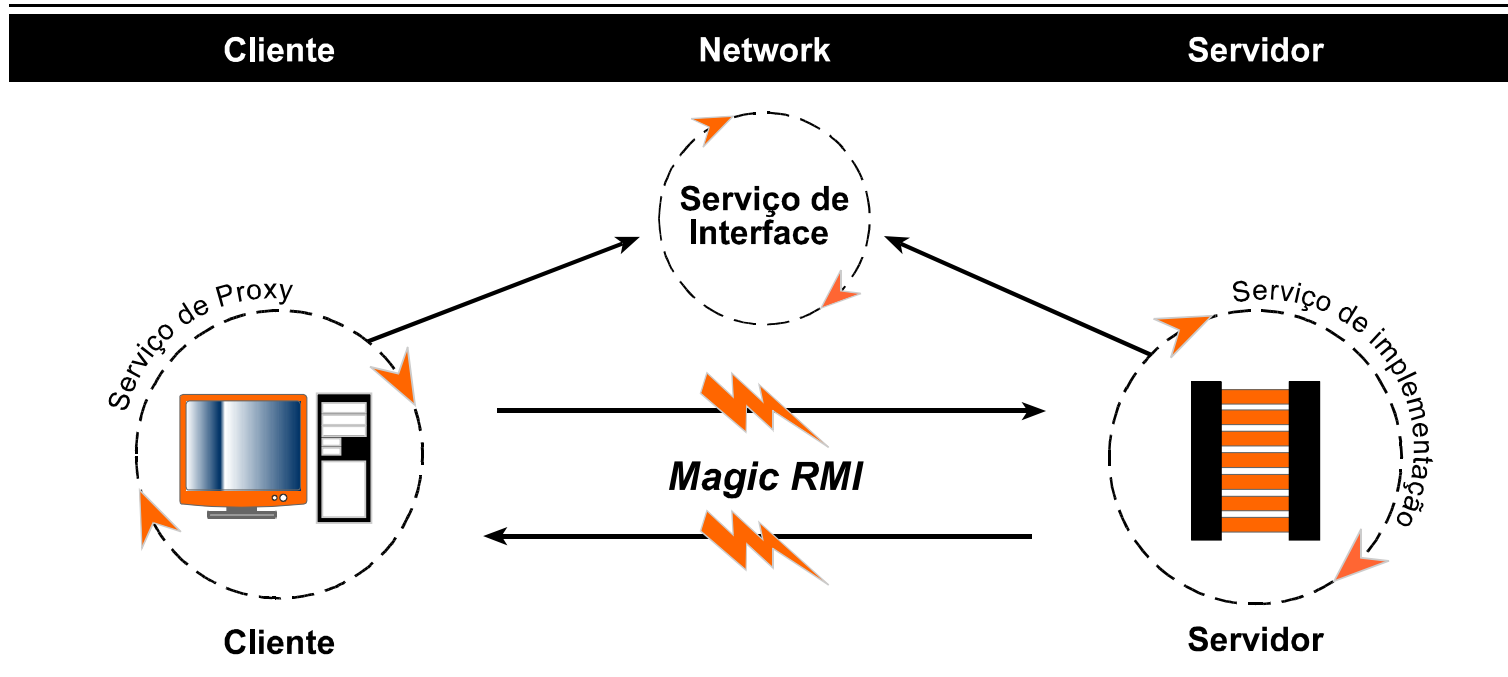
Princípios do EJB

- EJB é um modelo de componentes com foco em arquitetura n-camadas, baseado em princípios básicos de rede;
- Um servidor de EJB's é um serviço TCP/IP;
- EJB é um modelo de alto nível para objetos distribuídos / computação distribuída;
- EJB pode ser utilizado em soluções Web e não Web;
- EJB é o CICS do Java!

Princípios do EJB



Princípios do EJB



Princípios do EJB

- Todos EJB tem...
 - Uma interface que define a forma de acesso;
 - Uma interface que define a forma de construção;
 - Polimorfismo natural;
 - Uma classe que implementa seus “métodos de negócio”;
 - Um servidor que o gerencia;
 - Um nome público no catálogo de objetos;

Princípios do EJB

- EJB Vs. RMI Vs. CORBA
 - EJB é um servidor de objetos de mais alto nível que RMI e CORBA;
 - EJB é totalmente baseado em RMI;
 - EJB é mais fácil que CORBA;
 - EJB está mais pronto que RMI;

Agenda

- Princípios do EJB: socket, RMI e CORBA
- **Arquiteturas com e sem EJB**
- Introdução ao JBOSS
- JNDI
- EJBs - Session Bean
- Conclusões

Arquiteturas com EJB

- Quando usar EJB?
 - Alta escalabilidade;
 - Integração e compartilhamento de componente;
 - Uso de rich clients sem Web Container;
 - Clientes não-Java;
 - Flexibilidade na arquitetura;
 - Afunilamento e distribuição de requests;
 - Controle de threads de alto nível;
 - Pode poupar banco de dados e transações RDBMS;

Arquiteturas com EJB

- EJB: padrão de componentes escaláveis da espec J2EE, um hotel de objetos 5 estrelas;
- Ofertas de um container:
 - Escalabilidade, gestão de memória, ciclo de vida de objetos e estado de objetos;
 - Conexões, Transações, Serviço de nomes;
 - Segurança;
 - Prova de falhas para beans de entidade;
 - Integração e WebServices;
 - Agendamento (Session Timed Objects)
 - Clustering, alta disponibilidade e confiabilidade

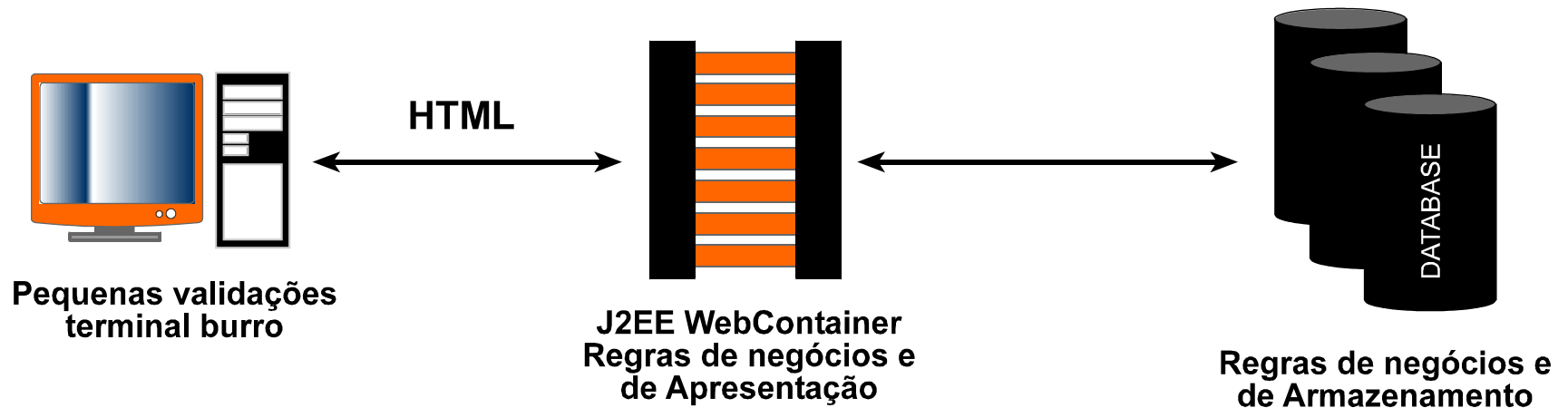
Arquiteturas sem EJB

- Web, thin client

Thin-client

HTTP Server

Database Server



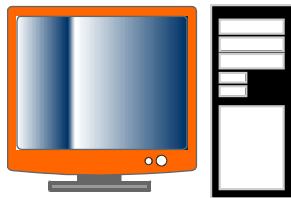
Arquiteturas sem EJB

- Web, rich client / rich internet application

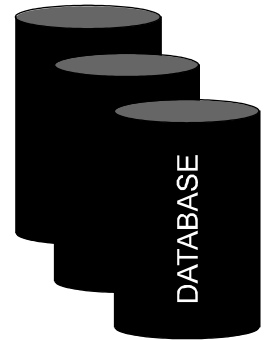
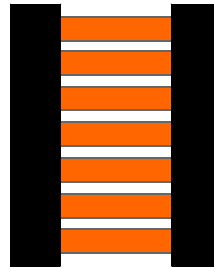
Fat-client

HTTP Server

Database Server



XML



Grande dinâmica na interface
Usuários exigentes:
Swing, Flash, .net entre outros

J2EE WebContainer
Regras de negócios e
de Apresentação

Regras de negócios e
de Armazenamento

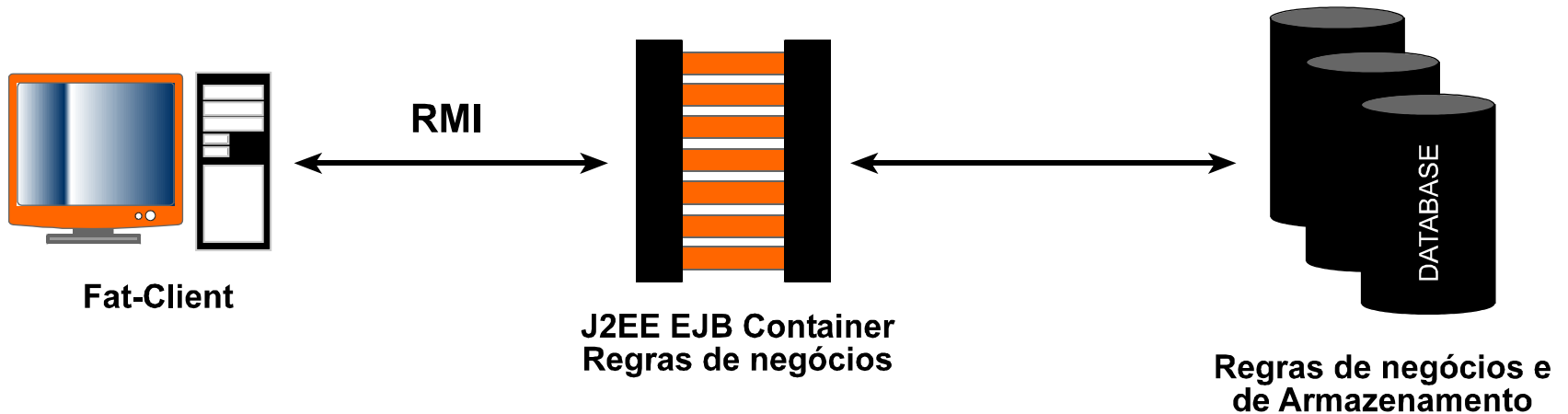
Arquiteturas com EJB

- 3 camadas não Web, com rich-client

Fat-client

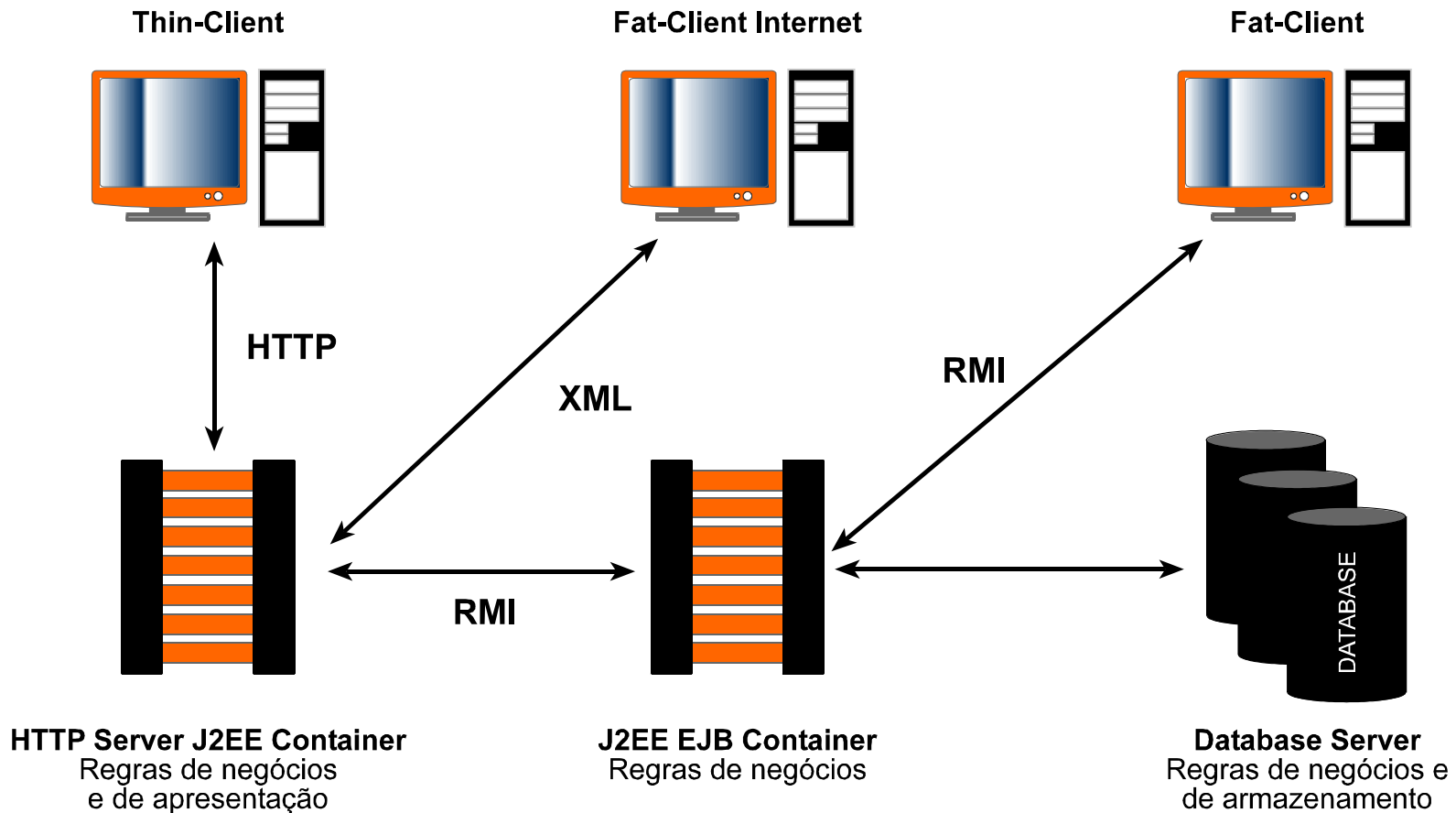
HTTP Server

Database Server



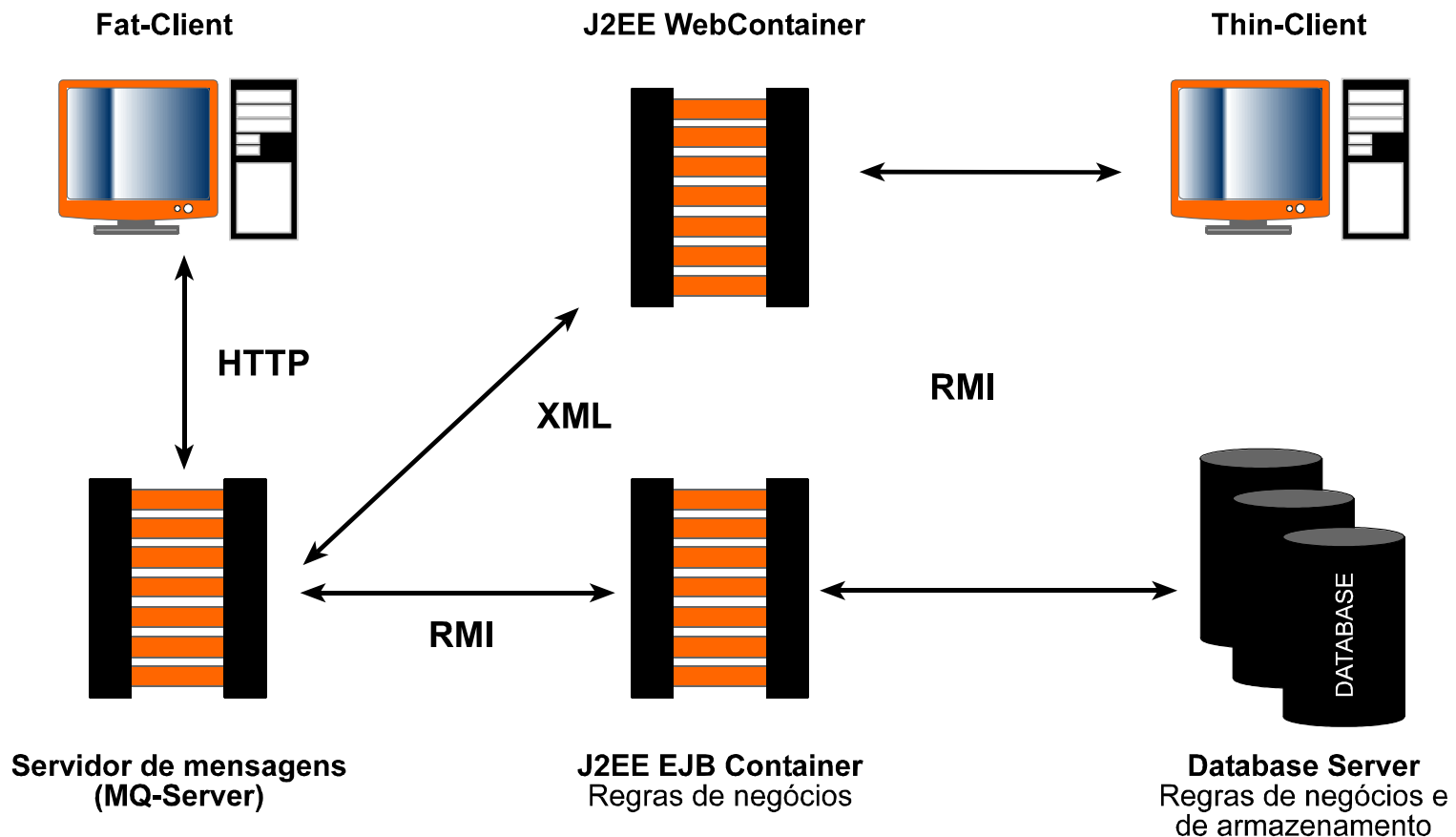
Arquiteturas com EJB

- Web + EJB, Thin-client + Rich-client



Arquiteturas com EJB

- Mensagens + Web + EJB, Thin-client + Rich-client



Agenda

- Princípios do EJB: socket, RMI e CORBA
- Arquiteturas com e sem EJB
- **Servidores EJB**
- JNDI
- EJBs - Session Bean
- Conclusões

JBOSS

- É um servidor de EJBs compatível com a especificação J2EE (não oficial até 3.x, J2EE certified no 4);
- Gratuito e open-source;
- Fácil, rápido e prático: feito em Java;
- Suporta clustering e vem com servidor de banco de dados, mensagens, utiliza hibernate para entity, Tomcat, implementa AOP, JMX e muito mais;

JBoss

- Instalação: é um zip, basta descompactar
- Inicializando: `run -c profile`
- Deployment: basta copiar para o diretório deploy do profile em questão
- Demo do startup do jboss

Sun Application Server

- É o servidor que acompanha o JDK 1.4 EE;
- É o reference implementation J2EE;
- É o GlassFish do JEE 5;
- Boa console de administração;
- Integração com NetBeans;
- Demo da console adm;

Agenda

- Princípios do EJB: socket, RMI e CORBA
- Arquiteturas com e sem EJB
- Introdução ao JBOSS
- **JNDI**
- EJBs - Session Bean
- Conclusões

JNDI

- É um catálogo de objetos;
- Representa a forma que o cliente encontra um EJB em um servidor;
- Quando implantamos um EJB, registramos o EJB com um nome JNDI;
- O cliente pesquisa por este nome para encontrar o EJB;
- JNDI é um serviço TCP/IP, o default funciona na porta 1099 do servidor;

Agenda

- Princípios do EJB: socket, RMI e CORBA
- Arquiteturas com e sem EJB
- Introdução ao JBOSS
- JNDI
- **EJBs**
- Conclusões

EJBs

- EJB: padrão de componentes escaláveis da espec J2EE, um hotel de objetos 5 estrelas;
- Ofertas de um container:
 - Escalabilidade, gestão de memória, ciclo de vida de objetos e estado de objetos;
 - Conexões, Transações, Serviço de nomes;
 - Segurança;
 - Prova de falhas para beans de entidade;
 - Integração e WebServices;
 - Agendamento (Session Timed Objects)
 - Clustering, alta disponibilidade e confiabilidade

EJBs

- Tipos de EJB:
 - Session Bean Stateless
 - Session Bean Stateful
 - Session Timed Object
 - Entity Bean BMP
 - Entity Bean CMP
 - Message Driven Bean

EJBs

- Uso de EJBs:
 - Aplicativos Web para grande demanda;
 - Aplicativos Swing sem Web;
 - Enterprise Application Integration em geral;
 - Clustering

EJBs – Session Bean

- Clients:
 - Servlets e JSPs;
 - EJB pode acessar outro EJB;
 - Aplicativo Swing;

EJBs

- Vantagens:
 - Compartilhamento de componentes;
 - Processo evolutivo JCP;
 - Escalabilidade;
 - Netbeans 4.1;

EJBs

- Características:
 - Remote / Local Interface
 - Implementação
 - Home Interface
 - Deployment Descriptor J2EE
 - Deployment Descriptor Específico

EJBs

- Empacotamento:
 - É um jar com a classes dentro do seus respectivos pacotes;
 - Contém um diretório META-INF com os deployment descriptors;

EJBs

- Não podemos em EJBs:
 - I/O
 - Threads
 - Métodos nativos JNI
 - Usar static
 - Usar awt, swing etc.
 - Atuar como servidor de rede
 - Utilizar reflection
 - Retornar this

EJBs – Olá Mundo

- Características:
 - Remote Interface
 - Implementação
 - Home Interface
 - Deployment Descriptor J2EE
 - Deployment Descriptor Específico

EJBs – Olá Mundo

- Características: remote Interface

```
package br.com.globalcode.aa.ejb.session;

public interface Calculadora extends javax.ejb.EJBObject {
    public int somaQuantica(int x, int y) throws
        java.rmi.RemoteException;
}
```

EJBs – Olá Mundo

- Características: Implementação

```
package br.com.globalcode.aa.ejb.session;
import javax.ejb.*;
import br.com.globalcode.aa.util.Debug;

public class CalculadoraBean implements SessionBean {
    SessionContext sessionContext;
    public void ejbCreate() throws CreateException {
        Debug.log("Método create", 9);
    }
    public void ejbRemove() {
        Debug.log("Método Remove", 9);
    }
    public void ejbActivate() {
        Debug.log("Método Activate", 9);
    }
    ...
}
```

EJBs – Olá Mundo

- Características: Implementação

```
...  
public void ejbPassivate() {  
    Debug.log("Método Passivate", 9);  
}  
public void setSessionContext(SessionContext sessionContext) {  
    Debug.log("Método setSessionContext", 9);  
    this.sessionContext = sessionContext;  
}  
public int somaQuantica(int x, int y) {  
    Debug.log("Método somaQuantica(int, int)", 9);  
    int r = (int) Math.random() * 100;  
    return x + y - r;  
}  
}
```

EJBs – Olá Mundo

- Características: Home Interface

```
package br.com.globalcode.aa.ejb.session;

import javax.ejb.*;
import java.rmi.*;

public interface CalculadoraHome extends EJBHome {
    public Calculadora create()
        throws CreateException, RemoteException;
}
```

EJBs – Olá Mundo

- Características: deployment Descriptor J2EE

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC
    "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 2.0//EN"
    "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <display-name>Calculadora</display-name>
      <ejb-name>Calculadora</ejb-name>
      <home>
        br.com.globalcode.aa.ejb.session.CalculadoraHome
      </home>
      <remote>
        br.com.globalcode.aa.ejb.session.Calculadora
      </remote>
    ...
```

EJBs – Olá Mundo

- Características: deployment Descriptor J2EE

```
<ejb-class>
  br.com.globalcode.aa.ejb.session.CalculadoraBean
</ejb-class>
<session-type>Stateless</session-type>
<transaction-type>Container</transaction-type>
</session>
</enterprise-beans>
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name>Calculadora</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

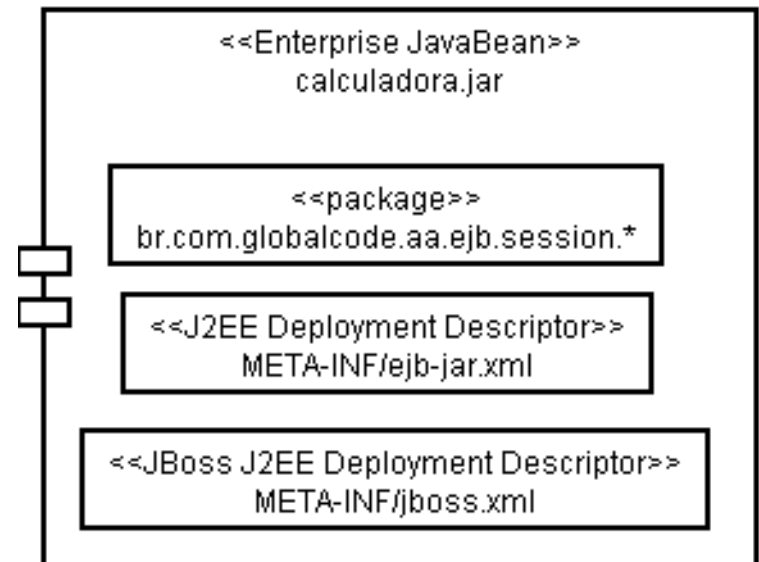
EJBs – Olá Mundo

- Características: Deployment Descriptor Específico

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 3.2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss_3_2.dtd">
<jboss>
  <enterprise-beans>
    <session>
      <ejb-name>Calculadora</ejb-name>
      <jndi-name>Calculadora</jndi-name>
    </session>
  </enterprise-beans>
</jboss>
```


EJBs – Olá Mundo

- Empacotamento
 - É um jar com a classes dentro do seus respectivos pacotes;
 - Contém um diretório META-INF com os deployment descriptors;

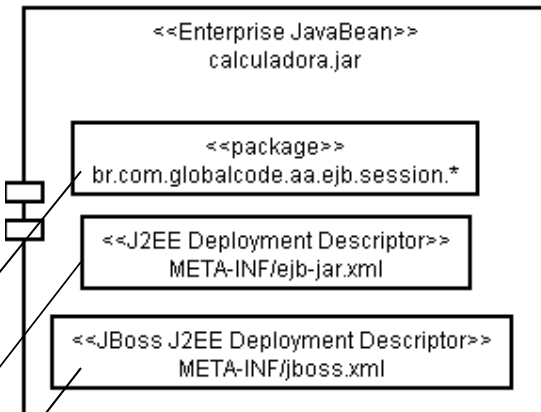


EJBs – Olá Mundo

- Empacotamento – Ant Script

```
<target name="ejb" depends="prepare, compile">
  <jar jarfile="${jars.dir}/calculadora.jar">
    <fileset dir="${deploy.home}">
      <include
        name="br/com/globalcode/aa/ejb/session/*" />
    </fileset>

    <fileset dir="etc/ejbs/session">
      <include name="META-INF/ejb-jar.xml" />
      <include name="META-INF/jboss.xml" />
    </fileset>
  </jar>
</target>
```

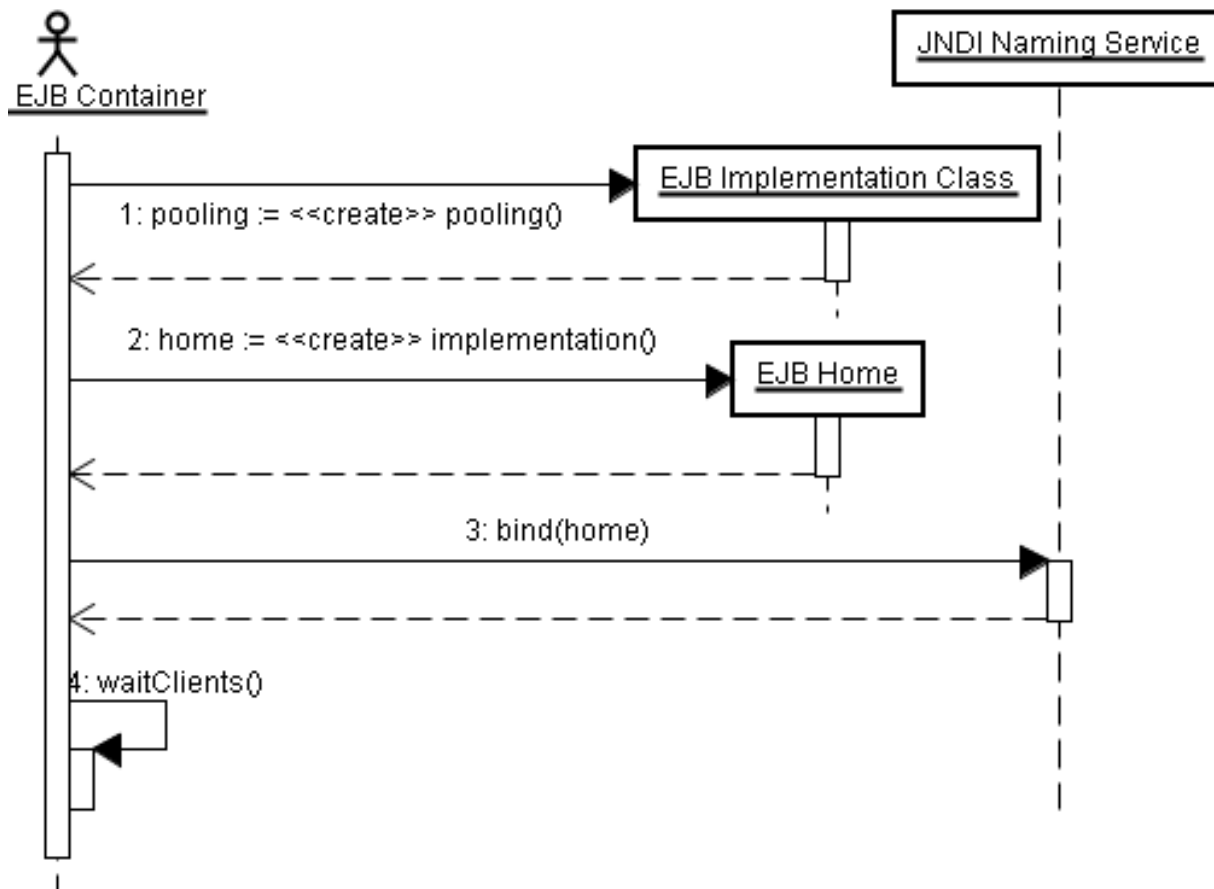


EJBs – Olá Mundo

- Deployment
 - Com cada servidor o processo é diferente;
 - Atualmente, quase todos permitem via aplicativo Web de administração / console administrativa;
 - No JBoss, contamos com hot-deploy: basta copiar o ejb-jar, war ou ear para o diretório deploy;

EJBs – Olá Mundo

- Deployment



- Acessando o EJB

```
package br.com.globalcode.aa.ejb.session.client;
```

```
import br.com.globalcode.aa.ejb.session.*;
```

```
import java.util.Hashtable;
```

```
import javax.naming.*;
```

```
import javax.rmi.PortableRemoteObject;
```

JNDI Naming Service

```
public class TesteCalculadora extends Object {
```

```
    public static void main(String[] args) throws Exception {
```

```
        Hashtable props = new Hashtable();
```

```
        props.put(Context.INITIAL_CONTEXT_FACTORY,  
            "org.jnp.interfaces.NamingContextFactory");
```

```
        props.put(Context.PROVIDER_URL, "localhost:1099");
```

```
        props.put(Context.URL_PKG_PREFIXES, "org.jboss.naming");
```

```
        Context context = new InitialContext(props);
```

```
        ...
```

EJBs – Olá Mundo

- Acessando o EJB

Casting Home Object

Lookup no Home Object

```
Context context = new InitialContext(props);
Object ref = context.lookup("Calculadora");

CalculadoraHome calculadoraHome = null;
calculadoraHome = (CalculadoraHome)
    PortableRemoteObject.narrow(ref, CalculadoraHome.class);

Calculadora calculadora = null;
calculadora = calculadoraHome.create();

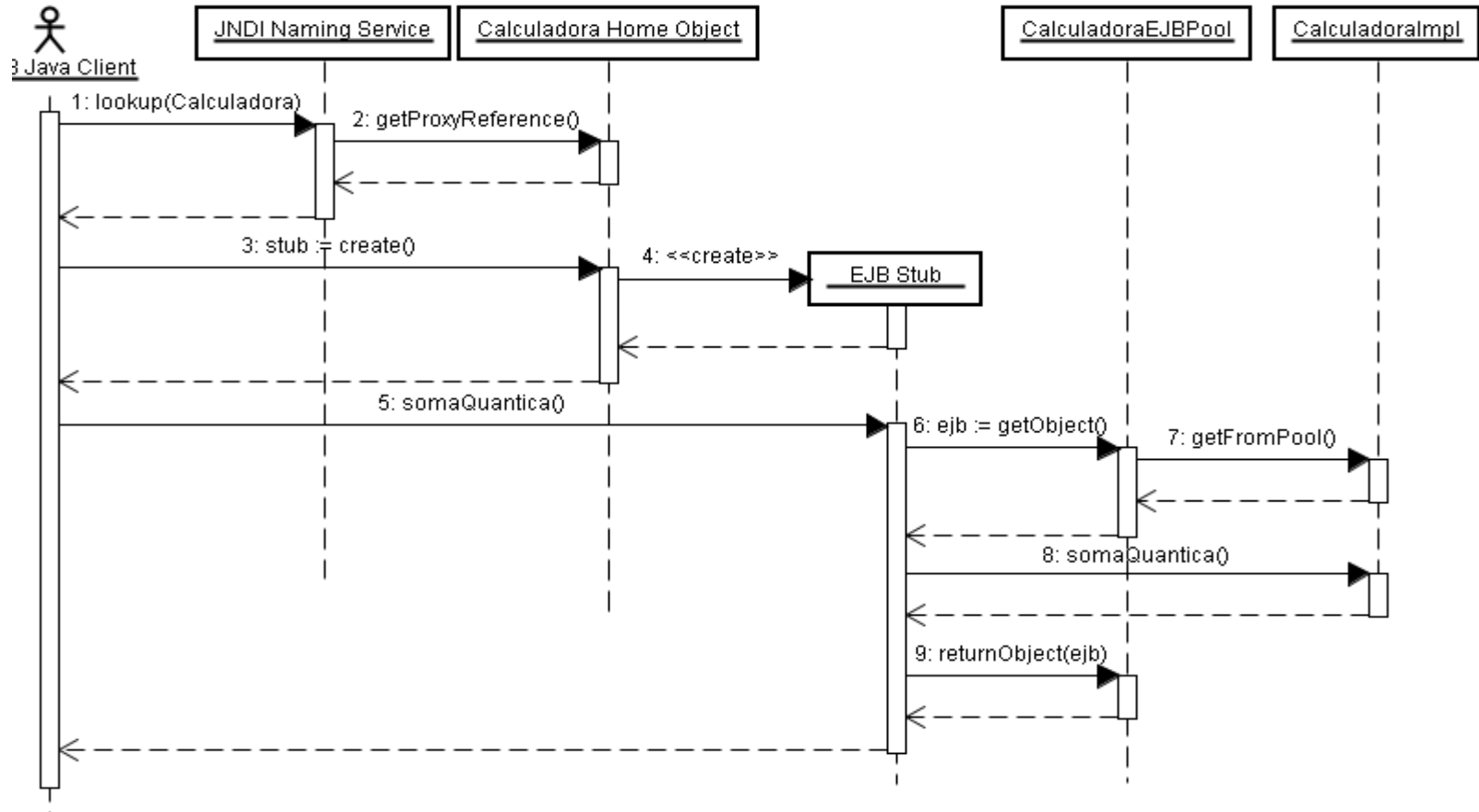
System.out.println("Resultado da soma quantica: " +
    calculadora.somaQuantica(10, 20));
}
```

Diagram illustrating the EJB lookup and execution process:

- Lookup no Home Object**: Points to the `context.lookup("Calculadora")` line.
- Casting Home Object**: Points to the `CalculadoraHome calculadoraHome = null;` and `calculadoraHome = (CalculadoraHome) PortableRemoteObject.narrow(ref, CalculadoraHome.class);` lines.
- Obtendo EJB**: Points to the `calculadora = calculadoraHome.create();` line.
- Executando EJB**: Points to the `calculadora.somaQuantica(10, 20)` line.

EJBs – Olá Mundo

- Acessando o EJB



EJBs

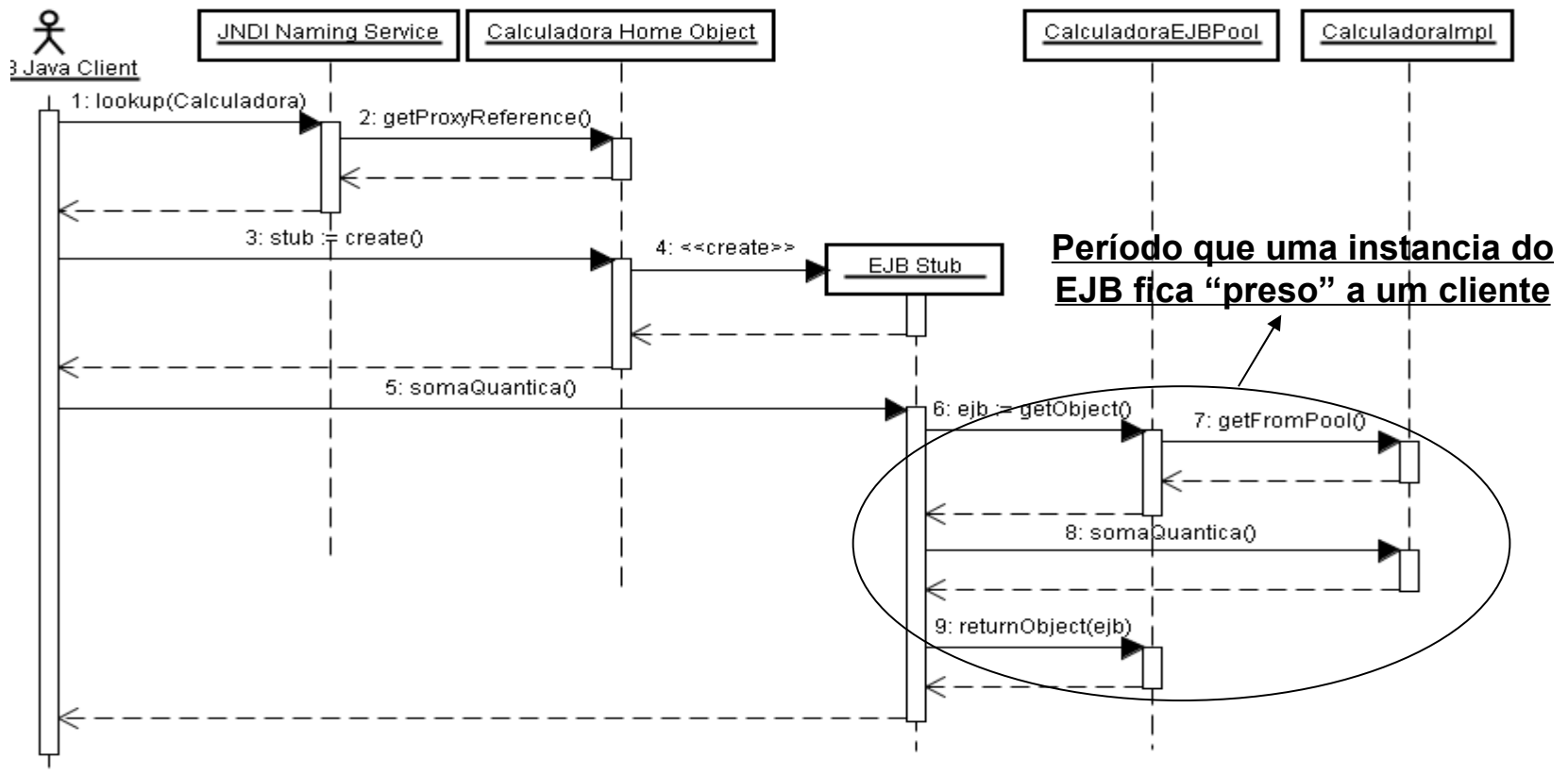
- DEMO

Stateless

- Um EJB Session Bean pode ser Stateless ou Stateful;
- **Stateless** significa que o EJB não terá compromisso de manter uma sessão para client um-para-um;
- Com **Stateless**, o container utiliza pooling de objetos e pode compartilhar 10 instancias de EJB's para 100 usuários simultâneos;
- Não devemos utilizar atributos de negócio, somente atributos técnicos em cache;
- **Stateless** é um EJB econômico;

Stateless

- **Stateless** fica “preso” a um cliente, somente pelo período de execução de um método:

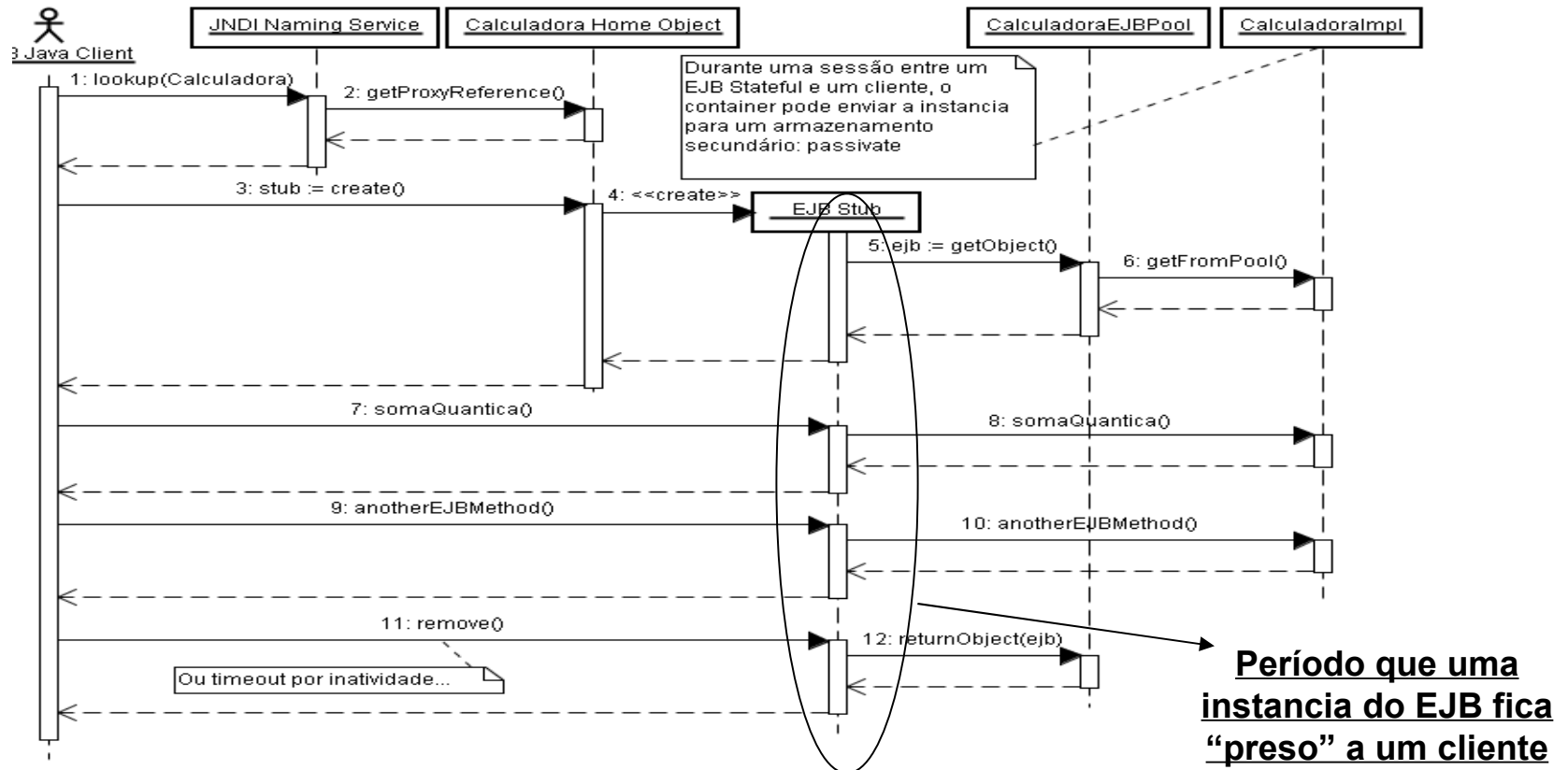


Stateful

- **Stateful** significa que cada cliente terá uma instancia do EJB exclusiva, desde o create da Home, até remove do EJB ou Distributed GC;
- Podemos ter atributos de negócio com métodos getters e setters;
- Pode representar uma sessão de aplicativo, como um carrinho de compras, respostas de telas tipo Wizard, armazenamento pré banco de dados;
- Se prevemos 100 usuários simultâneos, teremos 100 instancias de EJB's Stateful;

Stateful

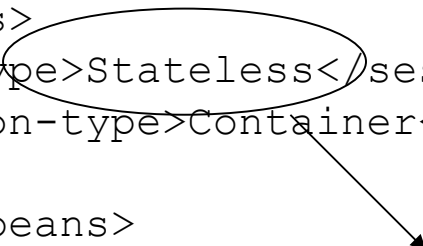
- **Stateful** fica “preso” a um cliente desde o create até o remove ou Distributed Garbage Collection:



Stateless Vs. Stateful

- Na prática, um não é melhor que outro: determinadas situações demandas por comportamentos diferentes;
- Tecnicamente é simples de mudar de stateless para stateful:

```
...  
<ejb-class>  
    br.com.globalcode.aa.ejb.session.CalculadoraBean  
</ejb-class>  
<session-type>Stateless</session-type>  
<transaction-type>Container</transaction-type>  
</session>  
</enterprise-beans>  
...  
</ejb-jar>
```



É só mudar para Stateful

Local Vs. Remote Interface

- Um EJB pode ser acessado por:
 1. Aplicativo Java stand-alone;
 3. Um Servlet, JSP ou Helper Class em um aplicativo Java para Web Container;
 5. Um EJB em um servidor, pode acessar outro EJB em outro servidor;
- Em todos os casos acima, estamos falando em comunicação remota entre dois computadores: client do EJB e Server do EJB;

Local Vs. Remote Interface

- Porém existe um outro tipo de acesso: um EJB pode ser acessado por outro EJB **no mesmo servidor**;
- Exclusivamente para o caso de um EJB acessar outro EJB dentro do mesmo servidor, podemos a partir da versão 2.0 do EJB, especificar um interface local;
- A interface local elimina overhead causado por proxy objects (objetos falsos que encapsulam o objeto remoto);
- Muito utilizada em Entity Beans;
- Um EJB pode ter Local, Remote ou ambas interfaces;

Local Vs. Remote Interface

- Características: Local Interface

```
package br.com.globalcode.aa.ejb.session;
```

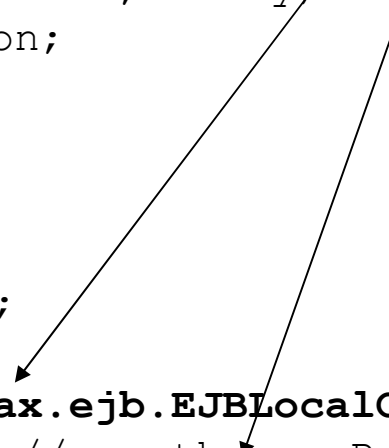
Remote

```
public interface Calculadora extends javax.ejb.EJBObject {  
    public int somaQuantica(int x, int y) throws  
        java.rmi.RemoteException;  
}
```

```
package br.com.globalcode.aa.ejb.session;
```

```
public interface Calculadora extends javax.ejb.EJBLocalObject {  
    public int somaQuantica(int x, int y); //sem throws RemoteExc  
}
```

Local



Local Vs. Remote Interface

- Características: Local Home Interface

```
package br.com.globalcode.aa.ejb.session;  
import javax.ejb.*;  
import java.rmi.*;
```

```
public interface CalculadoraHome extends EJBHome {  
    public Calculadora create()  
        throws CreateException, RemoteException;  
}
```

Remote home

```
package br.com.globalcode.aa.ejb.session;  
  
import javax.ejb.*;  
import java.rmi.*;
```

Local home

```
public interface CalculadoraHome extends EJBLocalHome {  
    public Calculadora create()  
        throws CreateException;  
}
```

Local Vs. Remote Interface

- Características: Local & Deployment Descriptor

...

```
<session>
  <display-name>Calculadora</display-name>
  <ejb-name>Calculadora</ejb-name>
  <home>
    br.com.globalcode.aa.ejb.session.CalculadoraHome
  </home>
  <remote>
    br.com.globalcode.aa.ejb.session.Calculadora
  </remote>
  <local>
    br.com.globalcode.aa.ejb.session.Calculadora
  </local>
  <local-home>
    br.com.globalcode.aa.ejb.session.CalculadoraLocalHome
  </local-home>
```

...

Entity Beans

- EJB's de sessão são especialistas em procedimentos,
- Entity beans em dados de entidades persistentes;
- Entity Bean é a forma padrão J2EE para representar uma entidade persistente em banco preferencialmente relacional;
- Não devemos adotar sem ferramental certo;
- Nível de complexidade alto para “fazer tudo na mão”;
- Um objeto Entity representa uma linha de tabela(s) do DB

Entity Beans

- Temos dois tipos de Entity Beans:
 - BMP: você é responsável pelos métodos e código SQL para C.R.U.D. da entidade;
 - CMP: gera automaticamente código SQL e cuida de relacionamentos entre entidades (CMR);
- Vantagens além do CMP:
 - Concorrência e sincronismo de acesso;
 - Cache de dados do DB em memória;
 - Inicialização inteligente (lazy e easy load);

Entity Beans

- Justifica o uso?
 - Em geral NÃO;
 - Depende também se você tem um bom container de Entity;
 - Domínio total da plataforma J2EE é exigido;
 - Maior flexibilidade: podemos designar um servidor só para Entity Beans e Façades e outro só para Session Beans;

Entity Beans

- Apresentaremos exemplos de BMP e CMP para C.R.U.D. na tabela cursos com os seguintes campos:

| <i>Campo</i> | <i>Tipo</i> |
|---------------------|-----------------------|
| codigo | int – auto-incremento |
| nome | varchar(255) |
| descricao | varchar(255) |
| apelido | varchar(16) |
| carga_horaria | Int |

Entity Bean – BMP DEMO

- Nesta demo vamos apresentar o código de um Entity Bean BMP já pronto:
 1. Código CursoBean.java – Implementação do BMP
 2. Código Curso.java – Interface do BMP
 3. Código CursoHome.java – Home Interface do BMP
 4. Deployment Descriptor ejb-jar.xml e jboss.xml

Entity Bean – CMP DEMO

- Nesta demo vamos apresentar o código de um Entity Bean CMP já pronto:
 1. Código CursoBean.java – Implementação do CMP
 2. Código Curso.java – Interface do CMP
 3. Código CursoHome.java – Home Interface do CMP
 4. Deployment Descriptor ejb-jar.xml e jboss.xml

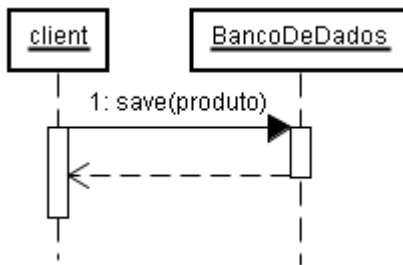
Message-driven Beans

- EJB que acionado de forma assíncrona;
- Trabalha totalmente vinculado ao conceito de Message Queue;
- É uma mistura entre as API's do EJB Session Bean e a API JMS;
- Serviços de mensagens (Message Queue Service), introduzem novos conceitos e arquitetura e aumentam a confiabilidade da solução;

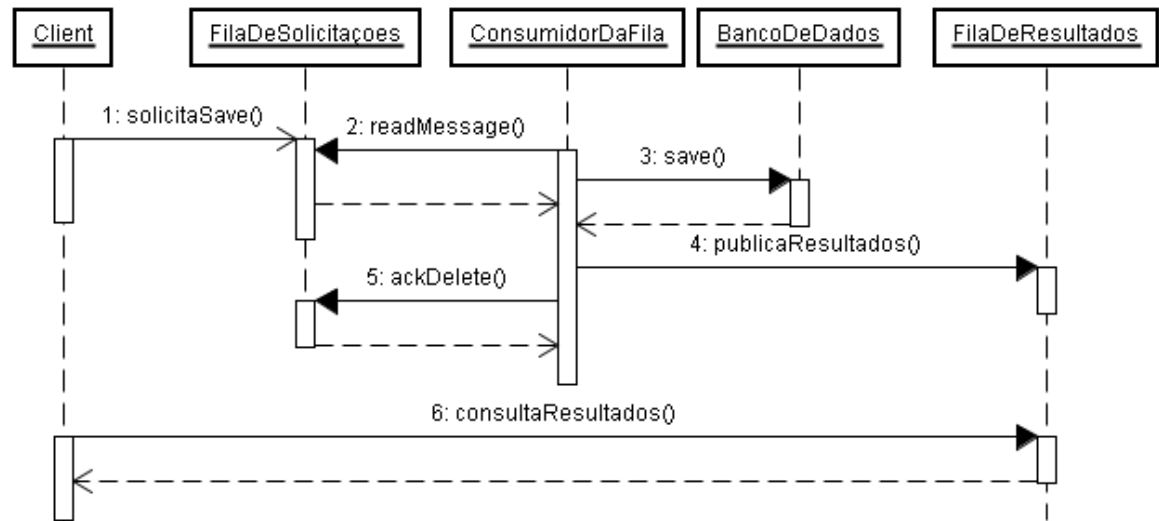
Message-driven Beans

- Vejamos o comparativo de uma chamada síncrona a um método, e uma chamada com Message Queue Service:

Síncrono



Assíncrono



Message-driven Beans

- Para trabalharmos com filas e Message-driven Beans, precisamos de um servidor Message Queue;
- Os mais conhecidos são: IBM MQ-Series, Tibco Rendezvous e Sonic-MQ;
- O JBoss traz um servidor chamado JBoss-MQ bastante funcional;
- Para “ligar” o servidor JBoss-MQ precisamos inicializar o container com o profile “all”;

Message-driven Beans

- Existem dois tipos de filas JMS:
 1. Point-to-point (queue): uma mensagem enviada para a fila será consumida apenas por um cliente;
 3. Publish-subscriber (topic): uma mensagem pode ser recepcionada por múltiplos clientes;

Message-driven Beans

- Cada middleware oferece uma maneira diferente de criamos uma fila ou tópico;
- No caso do servidor JBoss:
 - Arquivo dentro do diretório deploy/jms chamado jbossmq-destinations-service.xml;
 - Para criar filas tipo queue e topic, configuramos tags:

```
<mbean code="org.jboss.mq.server.jmx.Topic"  
      name="jboss.mq.destination:service=Topic,name=nomeDoTopico">
```

```
    <mbean code="org.jboss.mq.server.jmx.Queue"  
          name="jboss.mq.destination:service=Queue,name=nomeFila">
```

Message-driven Beans

- Exemplo de envio de mensagem para fila via JMS / método main:

```
public class SendToQueue {  
    public SendToQueue() {  
    }  
    public static void main(String[] args) throws Exception {  
        String msg="mensagem para publisher";  
        Hashtable props = new Hashtable();  
        props.put(Context.INITIAL_CONTEXT_FACTORY,  
            "org.jnp.interfaces.NamingContextFactory");  
        props.put(Context.PROVIDER_URL, "localhost:1099");  
        props.put("java.naming.rmi.security.manager", "yes");  
        props.put(Context.URL_PKG_PREFIXES, "org.jboss.naming");  
        Context context = new InitialContext(props);  
  
        ...  
    }  
}
```

Message-driven Beans

- Exemplo de envio de mensagem para fila via JMS / método main:

```
QueueConnectionFactory queueFactory =  
    (QueueConnectionFactory)context.lookup("ConnectionFactory");  
QueueConnection queueConnection =  
    queueFactory.createQueueConnection();  
QueueSession queueSession=queueConnection.createQueueSession(  
    false, Session.AUTO_ACKNOWLEDGE);  
Queue queue = (Queue)context.lookup("nomeFila");  
QueueSender queueSender = queueSession.createSender(queue);  
TextMessage message = null;  
message = queueSession.createTextMessage();  
message.setText(msg);  
queueSender.send(queue, message);  
queueConnection.close();  
queueSender.close();  
}  
}
```

Message-driven Beans

- EJB Message-driven Bean é um EJB que fica anexado a uma fila ou tópico;
- Ao receber uma mensagem, o EJB será acionado para consumir a mensagem, o container chama o método `onMessage` do EJB;
- A principal vantagem desta abordagem é que podemos limitar o número de EJBs consumidores, evitando super-consumo do servidor;
- As requisições dos usuários são armazenadas em filas e processadas conforme capacidade do servidor;

Message-driven Beans

- Exemplo de código Message-driven Bean

```
package br.com.globalcode.aa.jms.mdb;
import javax.ejb.*;
import javax.jms.*;
import javax.naming.*;
public class OlaMundoBean implements MessageDrivenBean, MessageListener {
    MessageDrivenContext messageDrivenContext;
    public void ejbCreate() {
    }
    public void ejbRemove() {
    }
    public void onMessage(Message msg) {
        System.out.println("Chegou nova mensagem para EJB");
        System.out.println(msg.toString());
    }
    public void setMessageDrivenContext(MessageDrivenContext messageDrivenContext) {
        this.messageDrivenContext = messageDrivenContext;
    }
}
```

Message-driven Beans

- É um EJB simples, pois não tem Home, Remote e Local interface;
- Vinculamos a fila nos deployment descriptors ejb-jar.xml e jboss.xml:

```
...  
<enterprise-beans>  
  <message-driven>  
    <display-name>OlaMundo</display-name>  
    <ejb-name>OlaMundo</ejb-name>  
    <ejb-class>br.com.globalcode.aa.jms.mdb.OlaMundoBean</ejb-class>  
    <transaction-type>Container</transaction-type>  
    <message-driven-destination>  
      <destination-type>javax.jms.Queue</destination-type>  
    </message-driven-destination>  
  </message-driven>  
</enterprise-beans> ...
```

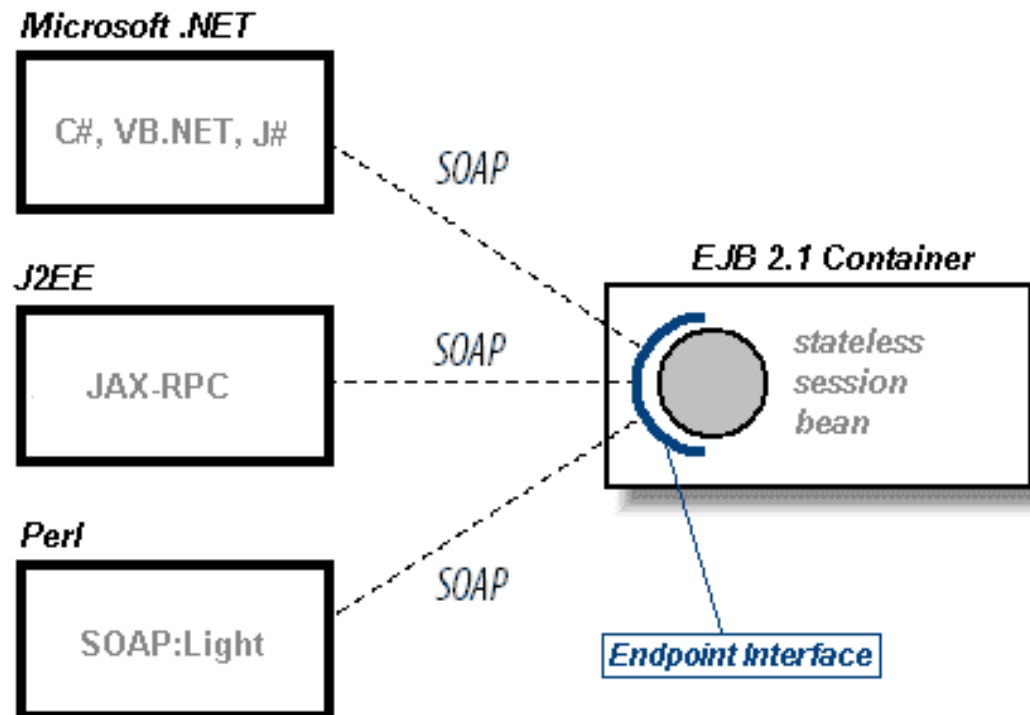
Message-driven Beans

- No jboss.xml indicamos o nome JNDI da fila ou tópico:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jboss PUBLIC "-//JBoss//DTD JBOSS 3.2//EN"
    "http://www.jboss.org/j2ee/dtd/jboss_3_2.dtd">
<jboss>
  <enterprise-beans>
    <message-driven>
      <ejb-name>OlaMundo</ejb-name>
      <destination-jndi-name>nomeFila</destination-jndi-name>
      <connection-factory-name>
        ConnectionFactory
      </connection-factory-name>
    </message-driven>
  </enterprise-beans>
</jboss>
```

EJB 2.1 - WebService

- Três protocolos: IIOP, JRMP / RMI e SOAP



EJB 2.1 - WebService

- Um Session Bean Stateless pode ser acessado por SOAP
- DEMO
 - NetBeans 4.1 criando EJBs com WebServices
 - Microsoft .NET acessando com C\$, ops, C#

Timer Service API

- Timer Service API: suporte a processos agendados;
- Equivalente a crontab, quartz, etc.;
- Podemos agendar processos em todos os EJB's, exceto Stateful session beans;
- Em um Entity Bean, podemos agendar um processo para enviar um e-mail se esgotar o tempo de entrega de um pedido;

Timer Service API

- Basta implementar a seguinte interface no EJB:

```
package javax.ejb;  
  
public interface TimedObject {  
    public void ejbTimeout(Timer timer) ;  
}
```

Timer Service API

- Devemos “ligar” o(s) timer(s) no EJB programaticamente;
- Podemos criar *single-action timers*, ou *interval timers*;
- Podemos criar múltiplos timers para um mesmo EJB;

Timer Service API

```
public class CalculadoraBean implements
    javax.ejb.SessionBean,
    javax.ejb.TimedObject,

    public void ejbCreate () {
        TimerService t = context.getTimerService();
        t.createTimer(5000, "Hi there!");
    }

    System.out.println(timer.getInfo());
}
```

Timer Service API

- Implements da interface TimedObject

```
public class CalculadoraBean implements  
    javax.ejb.SessionBean,  
    javax.ejb.TimedObject,
```

Timer Service API

- Programação do timer no ejbCreate

```
public void ejbCreate () {  
    TimerService t = context.getTimerService();  
    t.createTimer(5000, "Hi there!");  
}
```

Timer Service API

- Implementação do método `ejbTimeout` da interface `TimedObject`;

```
public void ejbTimeout (javax.ejb.Timer timer) {  
    System.out.println ("Chegou a hora..");  
    System.out.println (timer.getInfo ());  
}
```

EJB/QL

- Funções de domínio agregado (COUNT, SUM, MAX, MIN e AVG)
- Adição da cláusula *order by*;

Conclusões

- EJB's:
 - Componentes distribuídos que podem rodar em vários servidores;
 - Clustering;
 - Persistência automática com CMP;
 - Tuning de Threads e entidades de banco;
 - Acesso via WebService;
 - Transações;
 - Segurança;
 - Otimização de memória e threads;
 - Serviços de agendamento de tarefas;
- E você ainda está pensando se vale a pena?