




Forschungszentrum  
Informatik




Universität  
Karlsruhe (TH)

# Mehrschichtenarchitekturen und Komponentenumgebungen



Peter Tomczyk  
(FZI/IPE)

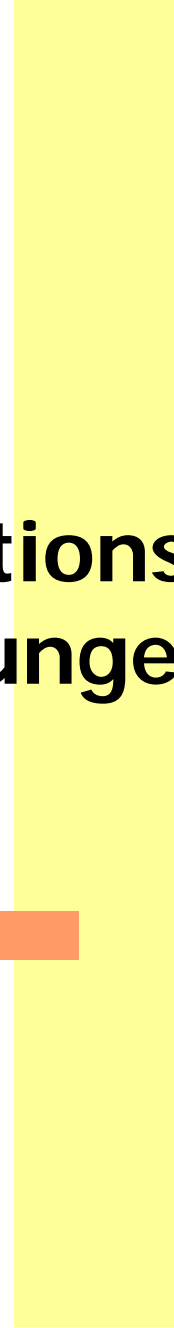


# Geschäftsprozesse und „Enterprise Applications“


- Geschäftsprozess = Verkettungen von Aktivitäten, die zur Erstellung der Unternehmensleistung beitragen
  - Vertriebsprozess, Materialbereitstellungsprozess, ...
- „Gute Geschäftsprozesse“ sind eine wichtige Grundlage für den anhaltenden Erfolg eines Unternehmens
- Ansätze zur Verbesserung von Geschäftsprozessen:
  - Optimierung
  - Automatisierung ⇒ „Enterprise Applications“
- Unsere Frage heute:
  - Wie entwickle ich Software zur Unterstützung von Geschäftsprozessen? → Methodik und Technologie

- „Enterprise Applications“ – Herausforderungen
- Grundlegende Lösungsansätze
- Komponenten und Komponentenumgebungen
- Konkretes Beispiel: Enterprise JavaBeans

# **„Enterprise Applications“ und ihre Herausforderungen**

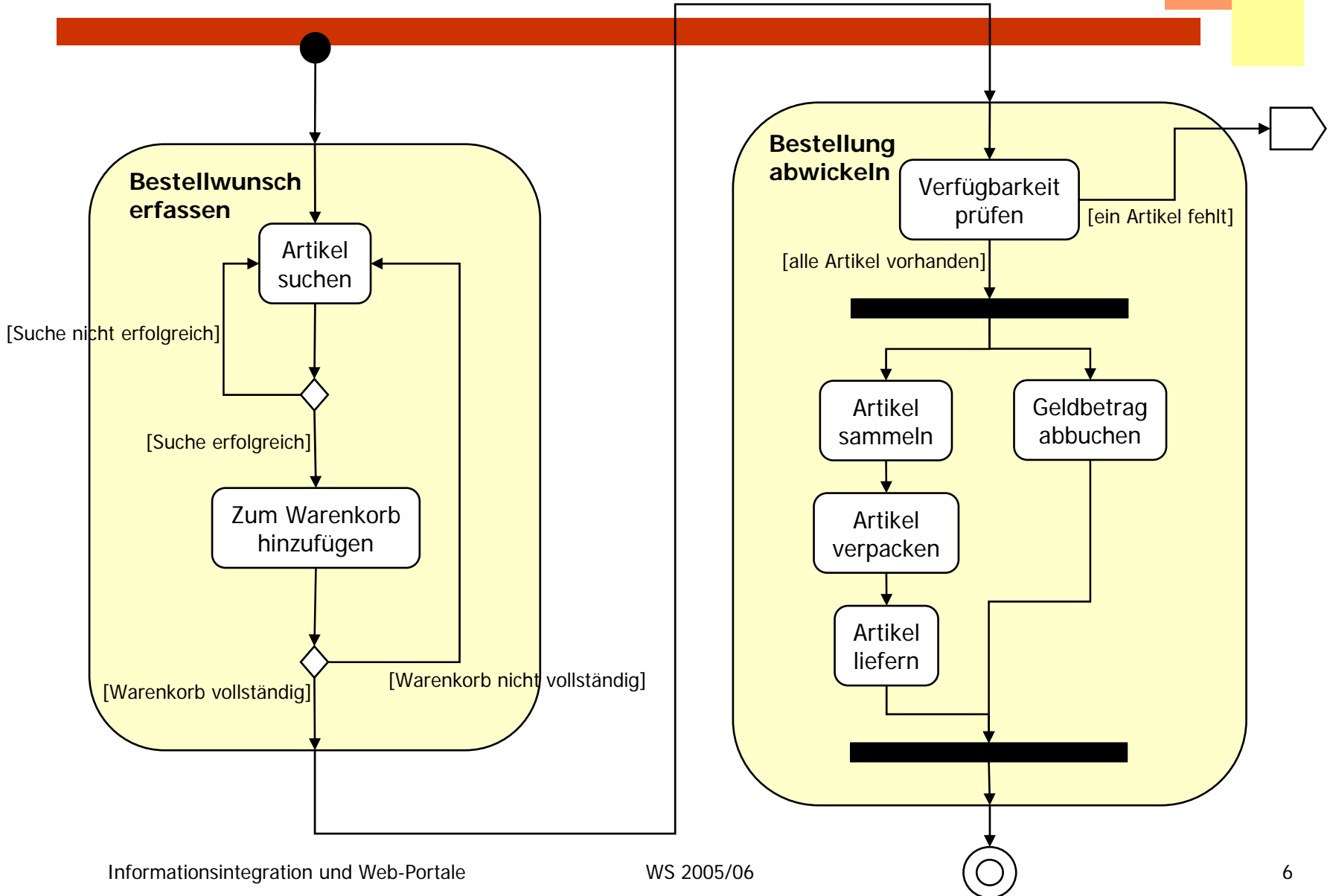


# Ein Stück Geschichte...



Ansatz:	<b>Mainframes</b>	<b>Client-Server-Architektur</b>	→	?
Zeit:	1975 - 1985	1985 - 1995		
Motivation:	Digitale Erfassung und Verarbeitung von Daten	Unterstützung komplexer Arbeitsvorgänge (CAD, CASE, ...)		
Rahmenbedingungen:	Extrem teure Rechnerleistung	Arbeitsplatzrechner werden erschwinglich		
Grundidee:	Ein Großrechner für die Datenverarbeitung + viele einfache Terminals zur Datenerfassung	Ein zentraler Rechner mit DBMS, ERP o.ä. (Server) + leistungsfähige Arbeitsplatzrechner (Clients)		

# Geschäftsprozesse heute: Online-Kauf



# Zentrale Herausforderungen

- Einbindung externer Akteure
  - Kunden, Lieferanten, Bank (B2C, B2B)
  - Ähnliches Problem: global agierende Unternehmen
- Vielzahl und Vielfältigkeit der Akteure
  - Wie viele Kunden erwarte ich? Welche Endgeräte?
- Automatisierung transaktionaler Vorgänge
  - „Geld abbuchen“ und „Artikel liefern“
- Integration existierender Systeme
  - Zugriff auf das ERP in „Verfügbarkeit prüfen“
- Ständige Veränderungen (→ Geschäftsprozessoptimierung)
  - Morgen verkaufen wir auch über eBay...
- Qualitätzusagen

# Rahmenbedingungen



- Knapper Kostenrahmen für
  - die Entwicklung
  - den Betrieb (hohe Wartbarkeit notwendig)

# Und wie die Geschichte weiter geht...

Ansatz:	Mainframes	Client-Server-Architektur	Mehrschichten-architektur	Dienstorientierte Architektur
Zeit:	1975 - 1985	1985 - 1995	1995 - ...	2002 - ...
Motivation:	Digitale Erfassung und Verarbeitung von Daten	Unterstützung komplexer Arbeitsvorgänge (CAD, CASE, ...)		
Rahmenbedingungen:	Extrem teure Rechnerleistung	Arbeitsplatzrechner werden erschwinglich	HTML/HTTP, CGI, Java, Servlets, ...	XML, WebServices, ...
Grundidee:	Ein Großrechner für die Datenverarbeitung + viele einfache Terminals zur Datenerfassung	Ein zentraler Rechner mit DBMS, ERP o.ä. (Server) + leistungsfähige Arbeitsplatzrechner (Clients)		

# Grundlegende Lösungskonzepte



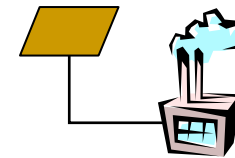
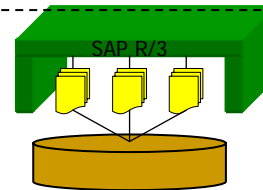
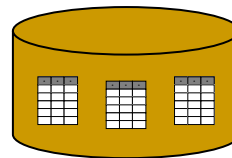
# Ausgangslage

Klienten mit  
unterschiedlichen  
Fähigkeiten



„Enterprise Application“

Rohdaten und -dienste  
(nicht integriert)



# Mehrschichtenarchitekturen

Klienten mit unterschiedlichen Fähigkeiten



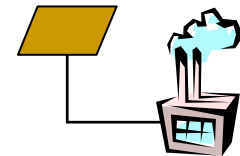
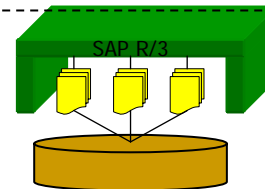
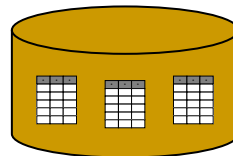
Interaktion

Geschäftsprozesse (auf Basis der Geschäftsobjekte)

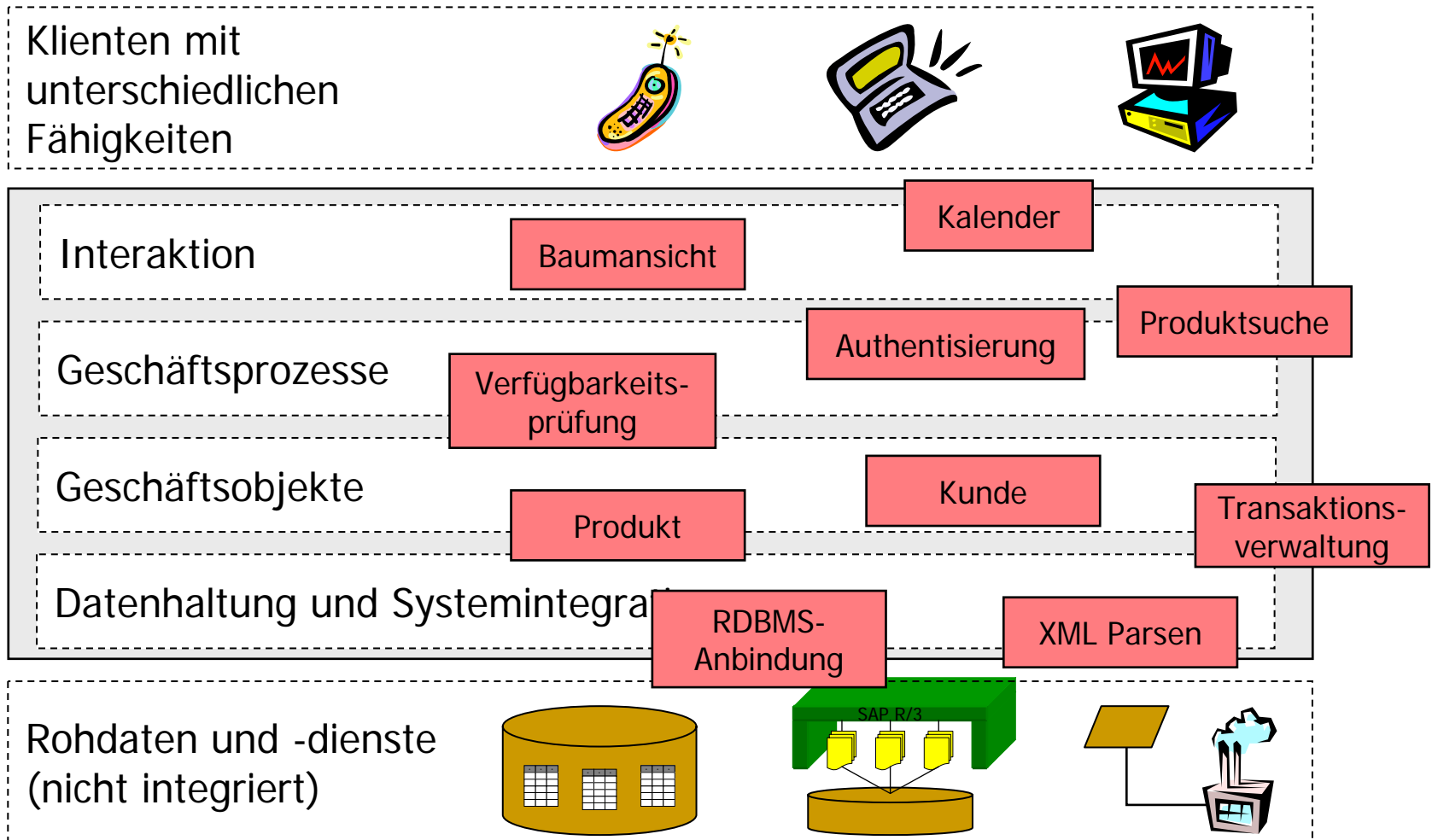
Geschäftsobjekte (technisch und inhaltlich integrierte Daten)

Datenhaltung und Systemintegration

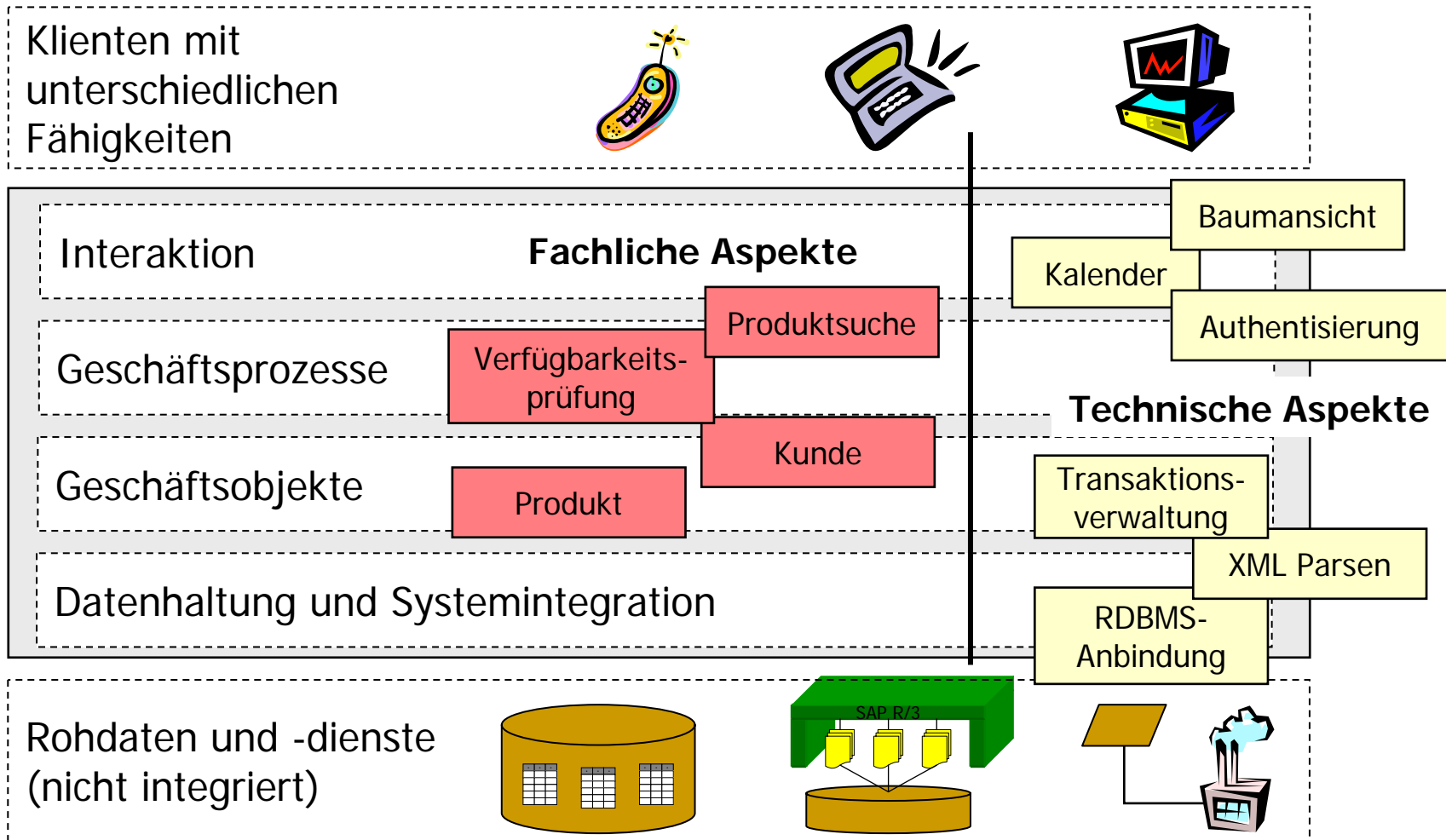
Rohdaten und -dienste (nicht integriert)



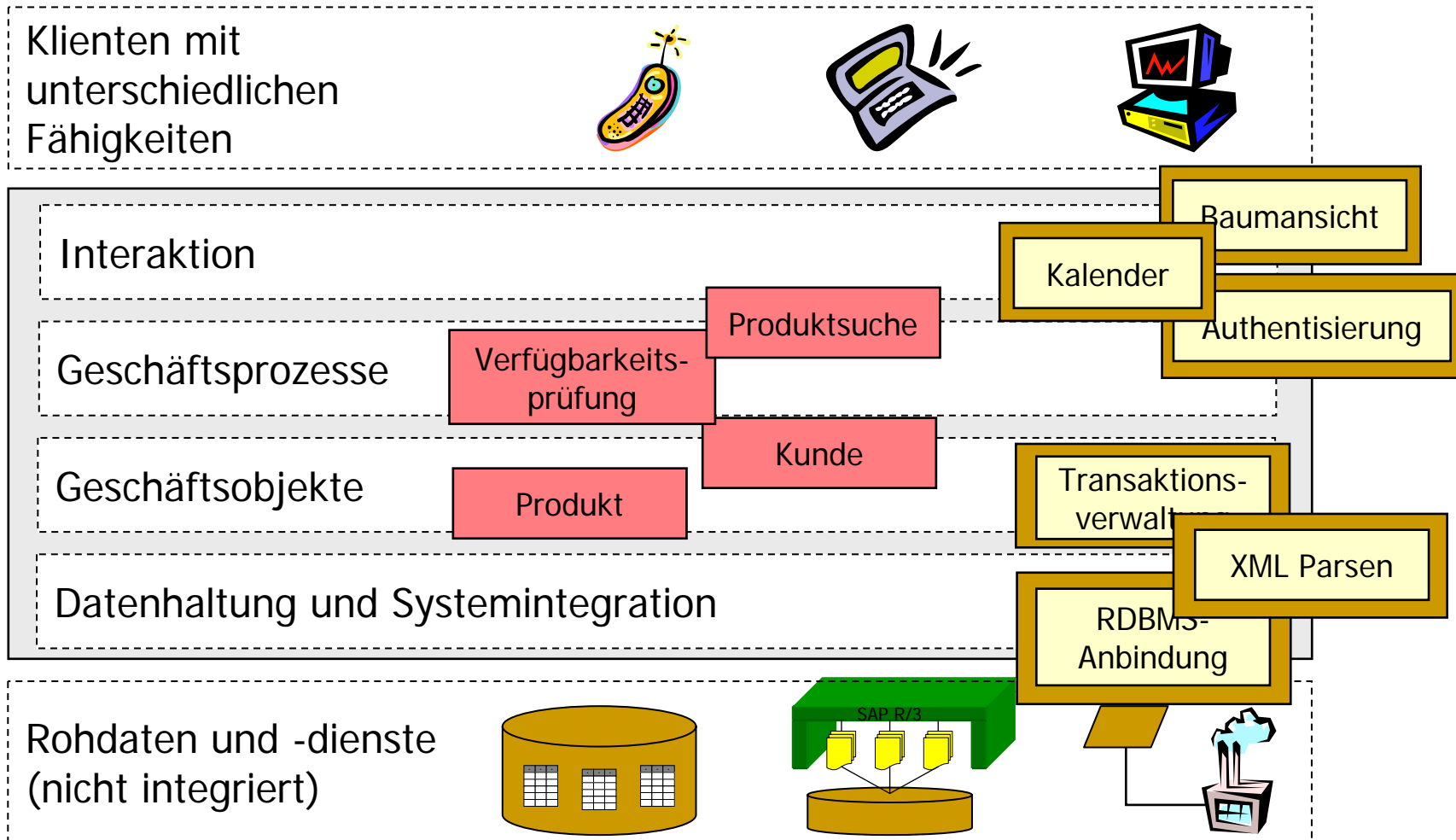
# Modularisierung



# „Separation of Concerns“



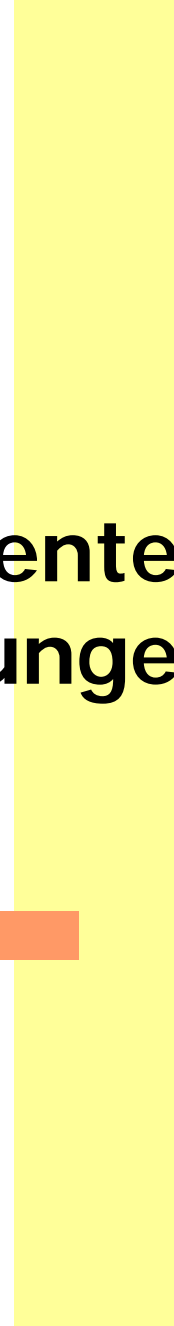
# Standardisierung



# Die Vorteile auf einen Blick

- Mehrschichtenarchitektur
  - Austauschbarkeit einzelner Schichten
  - 1:N-Lösungen (z.B. ein Prozess – mehrere Oberflächen)
- Modularisierung
  - Detaillierte Strukturierung des Entwicklungsprozesses
  - Eine gewisse Wiederverwendbarkeit von Teillösungen
- „Separation of Concerns“
  - Hohe Wiederverwendbarkeit und Austauschbarkeit technischer Module (hier noch unternehmensintern!)
- Standardisierung
  - Offene Märkte für Teillösungen werden möglich
  - Profilierung als Spezialist für Teillösungen möglich

# Komponenten und Komponentenumgebungen



# Komponentenumgebungen

Klienten mit unterschiedlichen Fähigkeiten



Interaktion

Geschäftsprozesse

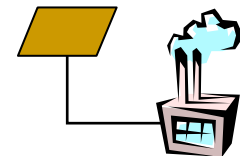
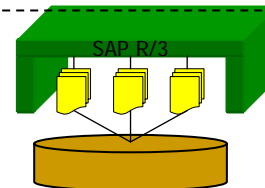
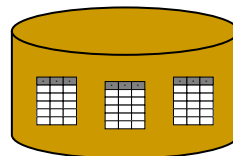


Geschäftsobjekte



Datenhaltung und Systemintegration

Rohdaten und -dienste  
(nicht integriert)



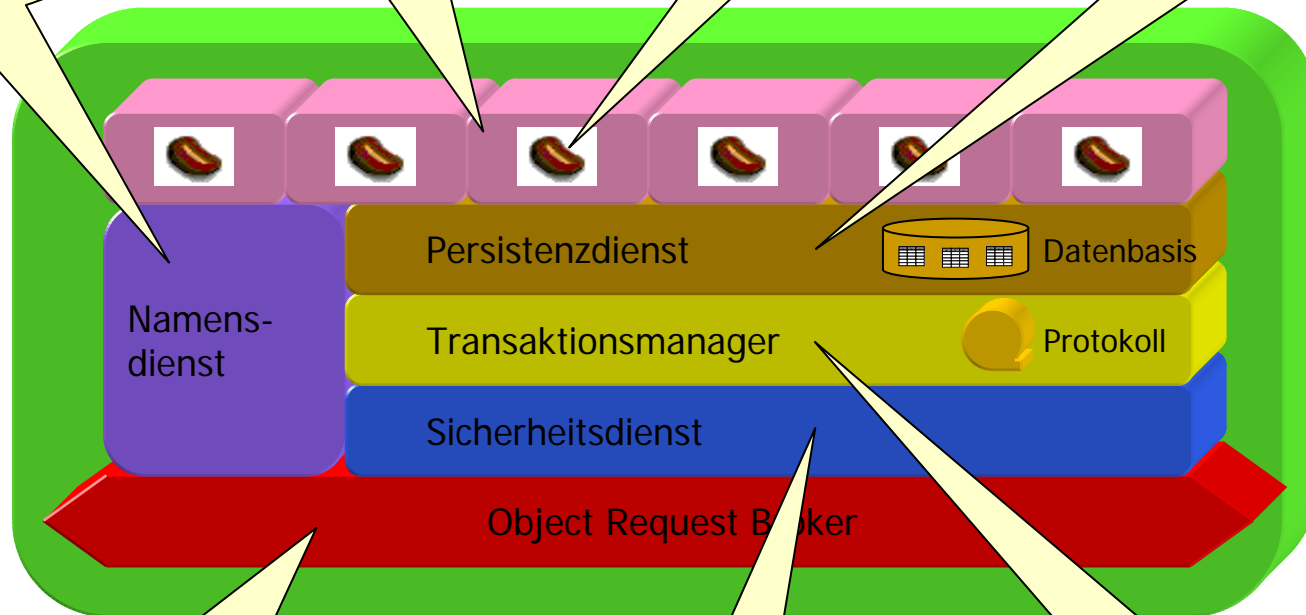
# Dienste einer Komponentenumgebung

5. Auffinden von Komponenten und Diensten

2. Container für Komponenten

1. Geschäfts-komponenten

3. Transparente Zustandsspeicherung



6. Transparente Kommunikation zwischen verteilten Objekten

7. Transparente Zugriffskontrolle

4. Transparente Transaktionsverwaltung

# Was sind Komponenten?



*"A component is a piece of software that is small enough to create and maintain, big enough to deploy and support, and with standard interfaces for interoperability."*

(J. Harris, President, CI Labs, 1995)

*"A component is a reusable, self-contained piece of software that is independent of any application."*

(R. Orfali, D. Harkey, J. Edwards:  
The Essential Distributed Objects Survival Guide)

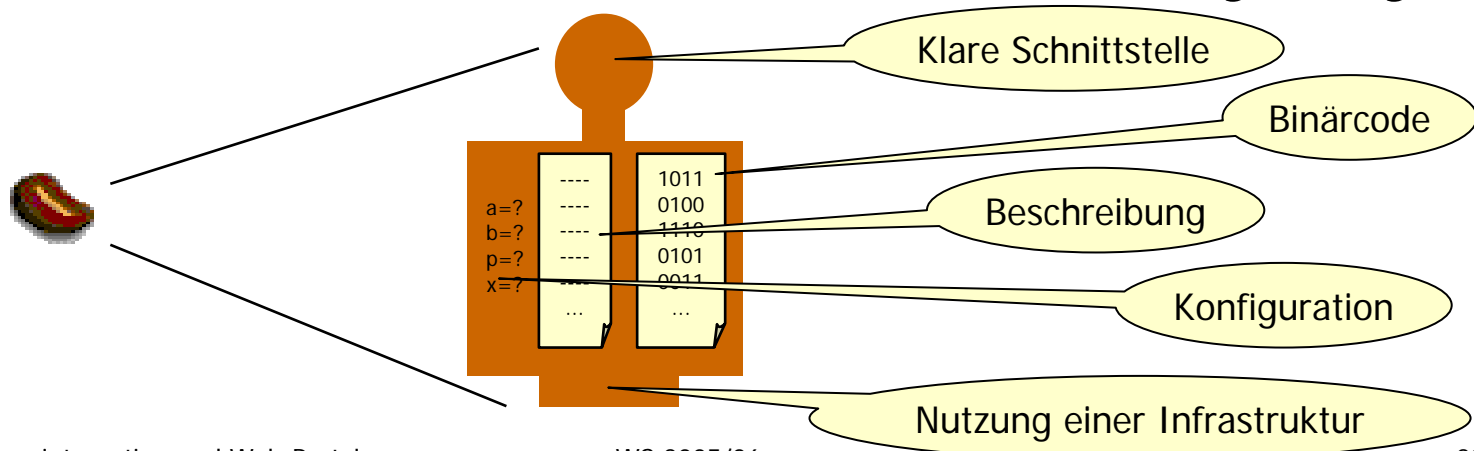
Zur Definition siehe auch C. Szyperski oder F. Griffel

# Komponenten: Wesentliche Eigenschaften

- Komponenten sind vielfältig einsetzbare, vermarktbar, in sich abgeschlossene Funktionseinheiten.
  - Komponenten interagieren nur über wohldefinierte Schnittstellen mit ihrer Umgebung.
  - Komponenten können leicht und flexibel zu größeren Funktionseinheiten gruppiert werden.
  - Beispiele:
    - Kunde (Geschäftsobjekt)
    - Bestellung (Geschäftsobjekt)
    - Kundenverwaltung (komplexe Komponente)
    - Produktkatalog (komplexe Komponente)
- ⇒ Es gibt keine optimale Granularität aus technischer Sicht!

# Komponenten: Softwaretechnische Sicht

- Komponenten sind Softwaremodule, die auf *Binärecodeebene* wiederverwendbar sind.
- Komponenten sind *selbstbeschreibend* sowohl hinsichtlich ihrer exportierten als auch ihrer importierten Dienste.
- Komponenten sind möglichst umfassend *konfigurierbar*.
- Komponenten implementieren Infrastrukturdienste nicht selbst, sondern beziehen sie aus ihrer *Einsatzumgebung*.



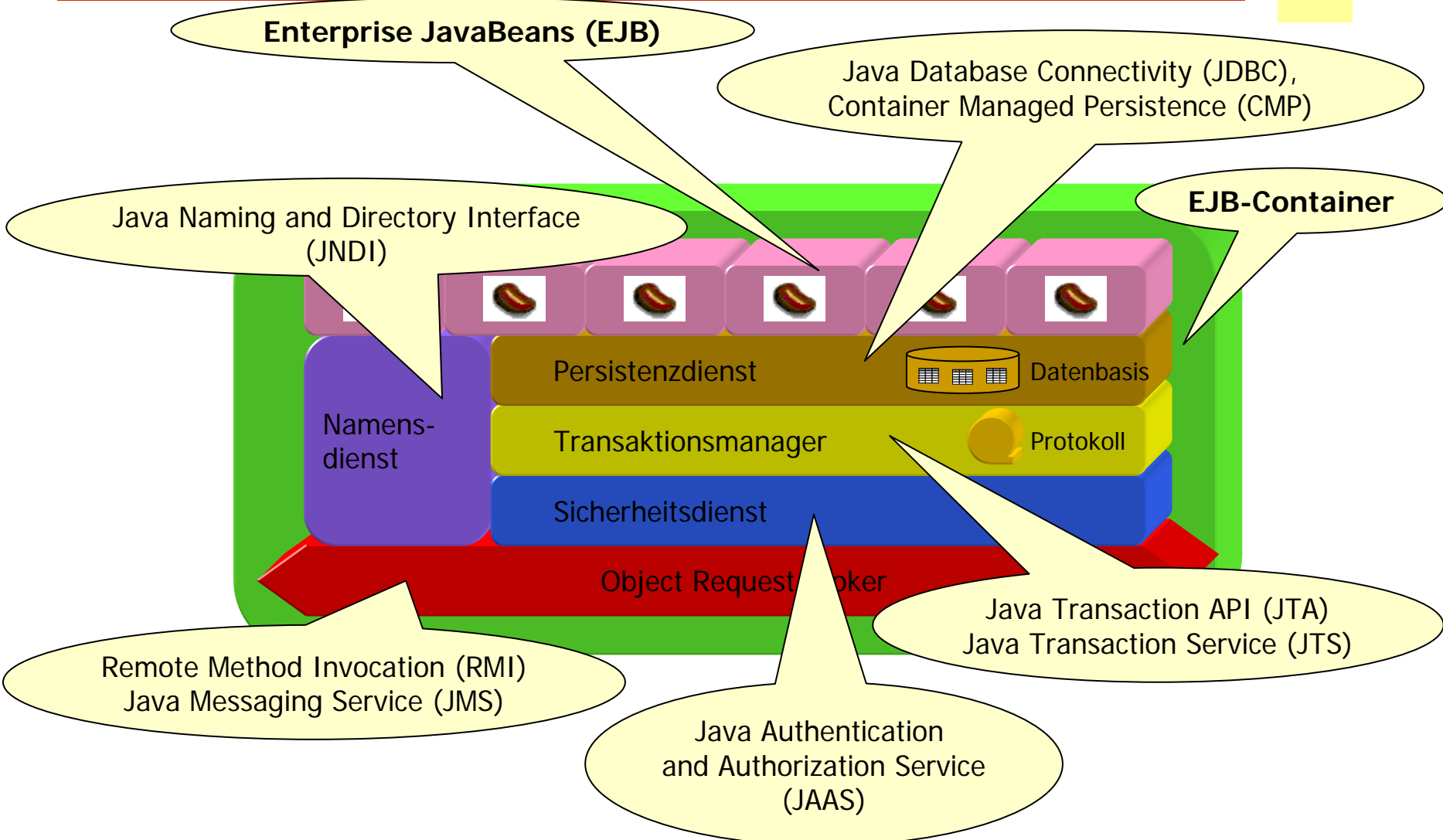
# **Enterprise JavaBeans: Der offene Standard für Komponentenumgebungen**



# Java 2 Enterprise Edition (J2EE) und die Enterprise JavaBeans (EJB)

- Java 2 Enterprise Edition:
  - „J2EE defines the standard for developing component-based multitier enterprise applications.“ [[java.sun.com/j2ee/](http://java.sun.com/j2ee/)]
  - Entwickelt von Sun Microsystems (1999)
- Enterprise JavaBeans:
  - „...stellt in J2EE die zentrale Technologie zur Abbildung von Geschäftslogik und Geschäftsdaten dar.“ [Backschat/Gardon]
  - aktuell ist EJB 2.1
  - gerade im Kommen: EJB 3.0

# J2EE-Komponentenumgebung

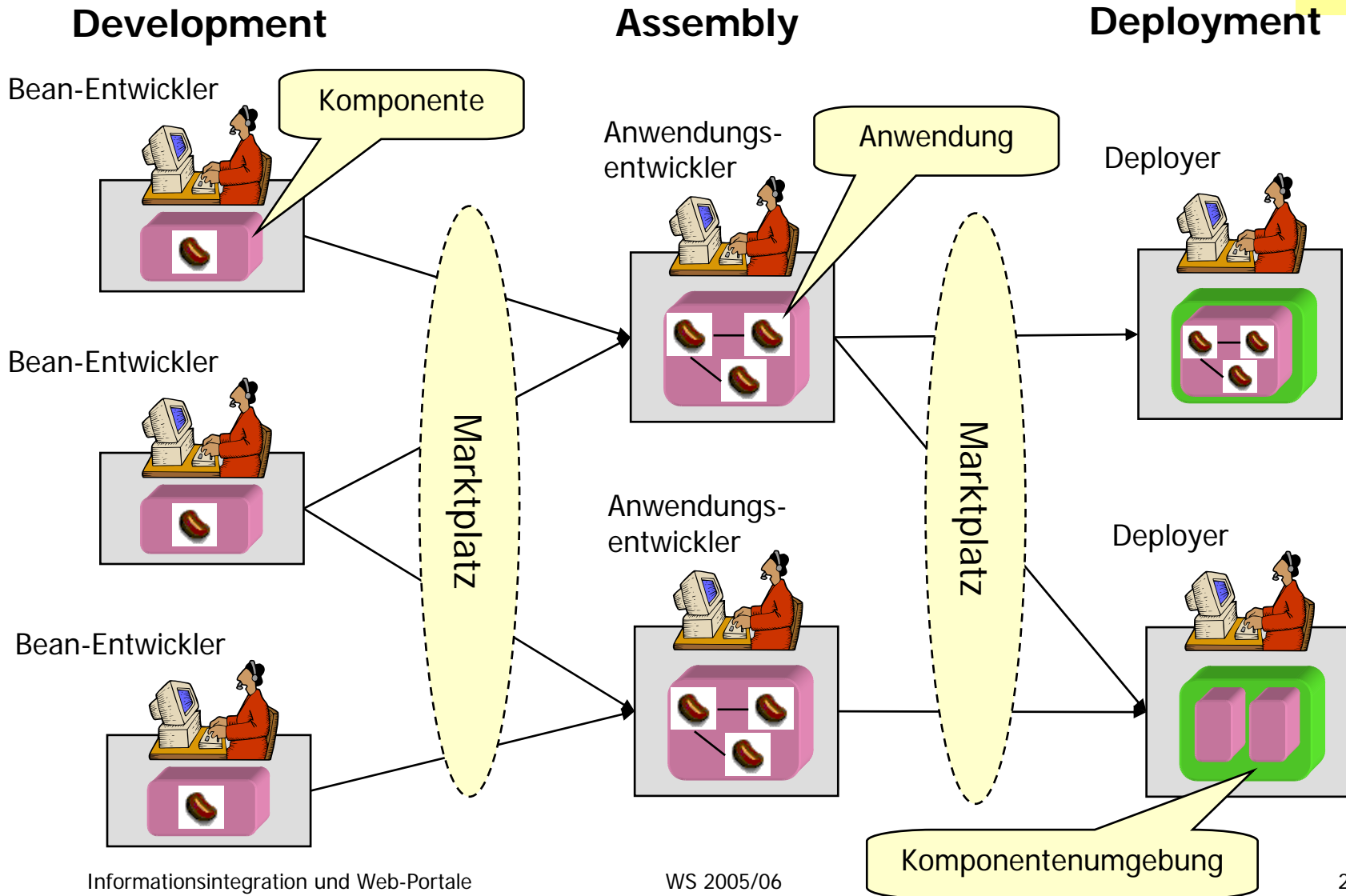


# Wesentliche Inhalte des EJB-Standards



- EJB spezifiziert
  - einen Komponentenprozess mit diversen Rollen
  - die grundsätzlichen Arten von Komponenten
  - die Klientensicht von Komponenten
  - die Regeln für die Implementierung von Komponenten
  - den Lebenszyklus von Komponenten und die Interaktion zwischen Komponente und Container
  - die Beschreibung und Konfiguration von Komponenten
  - Anfragesprache zum Zugriff auf persistente Komponenten

# EJB-Prozess



# Komponentenarten in EJB

- *Entity Beans* modellieren *Geschäftsobjekte*:  
Kunden, Lieferanten, Produkte etc.
  - Langlebige, von mehreren Klienten genutzte Objekte
  - Bean und Container Managed Persistence möglich
  - Identifikation durch Objektreferenz und Primärschlüssel
  - Sinnvoll zur objektorientierten Kapselung von Datensätzen
- *Session Beans* modellieren *Geschäftsprozesse*:  
Kaufvorgang, Buchung, Bonitätsprüfung etc.
  - Kurzlebige, nur von einem Klienten genutzte Objekte
  - Zustand existiert nur für die Dauer des Geschäftsprozesses
  - Identifikation nur durch Objektreferenz
  - Sinnvoll zur Kapselung von Geschäftslogik und Diensten
- *Message Driven Beans* modellieren *ereignisgetriebene Prozesse*

# Bestandteile einer Entity Bean

- **Remote Interface:** Definiert die für Klienten nutzbaren Methoden ("Geschäftsmethoden") der Bean.
- **Home Interface:** Definiert Methoden zum Erzeugen, Aufsuchen und Zerstören von Entity Beans.
- **Primary Key:** Ein serialisierbares Objekt, das den Datenbankschlüssel für die Zustandsdaten der Bean kapselt.
- **Bean:** Das eigentliche Bean-Objekt mit Zustandsdaten und den Implementierungen der Geschäftsmethoden.
- **Deployment Descriptor:** XML-Datei, die Struktur, Persistenzanforderungen, Zugriffsrechte, transaktionales Verhalten und referenzierte Namen der Bean beschreibt.

# Bestandteile einer Session Bean



- **Remote Interface:** Definiert die für Klienten nutzbaren Methoden ("Geschäftsmethoden") der Bean.
- **Home Interface:** Definiert Methoden zum Erzeugen und Zerstören von Session Beans.
- **Bean:** Das eigentliche Bean-Objekt mit den Implementierungen der Geschäftsmethoden.
- **Deployment Descriptor:** XML-Datei, die Struktur, Zugriffsrechte, transaktionales Verhalten und referenzierte Namen der Bean beschreibt.

# Entity Beans: Beispiel

- Angenommen, klick-and-bau.com verwaltet Artikelstammdaten in der folgenden Tabelle:

```
ARTIKEL(  
  Id          CHAR(12),  
  Name       VARCHAR(25),  
  Beschreibung VARCHAR(512),  
  Photo_URL  VARCHAR(512),  
  Hersteller_Id CHAR(12)  
  Kategorien VARCHAR(256),  
  Status     INT,  
  Preis     NUMERIC(8,2)  
)
```

- Die Datensätze sollen nun als "Artikel"-Entity Beans gekapselt werden.

# Artikel-Bean: Remote Interface

```
package com.klick-and-bau;

import java.rmi.RemoteException;

public interface Artikel extends javax.ejb.EJBObject {

    public String getName() throws RemoteException;
    public void setName(String name) throws RemoteException;

    public String getBeschreibung() throws RemoteException;
    public void setBeschreibung(String beschr) throws RemoteException;

    ...

    public double getPreis() throws RemoteException;
    public void setPreis(double preis) throws RemoteException;

}
```

# Artikel-Bean: Home Interface

```
package com.klick-and-bau;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.FinderException;

public interface ArtikelHome extends javax.ejb.EJBHome {

    public Artikel create(String id)
        throws RemoteException, CreateException;

    public Artikel findByPrimaryKey(ArtikelPK artikelPk)
        throws RemoteException, FinderException;

    // remove() Methoden werden von EJBHome geerbt
}
```

# Artikel-Bean: Primärschlüsselklasse

```
package com.klick-and-bau;

public class ArtikelPK implements java.io.Serializable {

    // Schlüsselfelder - hier muss eine Untermenge der Felder
    // der Beanklasse stehen.

    public String id;

    // Spezielle Hash- und Vergleichsmethoden, da die Schlüsselwerte
    // und nicht die Java-Objekte gehasht bzw. verglichen werden sollen.

    public int hashCode() {
        return id.hashCode();
    }

    public boolean equals(Object obj) {
        return (obj instanceof ArtikelPK) && id.equals(((ArtikelPK)obj).id);
    }
}
```

# Artikel-Bean: Beanklasse

```
package com.klick-and-bau;

public class ArtikelBean implements javax.ejb.EntityBean {

    // Zustandsvariablen - werden vom Container automatisch
    // anhand der Informationen im Deployment Descriptor
    // mit den entsprechenden Tabellenfeldern abgeglichen.

    public String id;
    public String name;
    public String beschreibung;
    public String photoURL;
    public String herstellerId;
    public String kategorien;
    public int    status;
    public double preis;
```

# Artikel-Bean: Beanklasse

```
// Initialisierungsmethode - wird vom Container aufgerufen,  
// wenn Klient create()-Methode des Home Interface aufruft.  
// Für jede create()-Methode des Home Interface muss eine  
// entsprechende ejbCreate()-Methode der Beanklasse existieren.
```

```
public ArtikelPK ejbCreate(String id) {  
    this.id = id;  
    // CMP: sonst nichts zu tun  
    // BMP: „INSERT INTO ARTIKEL VALUES (?,?,...)“  
    return new ArtikelPK(id);  
}
```

```
// Zusätzliche Initialisierungsmethode - wird vom Container  
// aufgerufen, nachdem Datensatz in der Datenbank angelegt  
// wurde.
```

```
public void ejbPostCreate(String id) {  
    // Dient vor allem zur Initialisierung von Bean-Referenzen.  
}
```

# Artikel-Bean: Beanklasse

```
// Implementierungen der Geschäftsmethoden.
```

```
public String getName() {  
    return name;  
}
```

```
public void setName(String name) {  
    this.name = name;  
}
```

```
// etc.
```

# Artikel-Bean: Beanklasse

```
// Lebenszyklusmethoden - werden vom Container aufgerufen.  
  
public void ejbLoad() {  
    // Zustandsvariablen werden aus der Datenbank gelesen.  
    // CMP: nichts zu tun.  
    // BMP: „SELECT * FROM ARTIKEL WHERE Id = ?“  
}  
  
public void ejbStore() {  
    // Zustandsvariablen werden in die Datenbank geschrieben.  
    // CMP: nichts zu tun.  
    // BMP: „UPDATE ARTIKEL SET Name = ? ... WHERE Id = ?“  
}  
  
public void ejbRemove() {  
    // Bean und Datensatz werden gelöscht.  
    // CMP: nichts zu tun.  
    // BMP: „DELETE FROM ARTIKEL WHERE Id = ?“  
}  
  
// Einige weitere Lebenszyklusmethoden sind nicht gezeigt.  
}
```

# Artikel-Bean: Deployment Descriptor

```
<?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">
```

# Artikel-Bean: Deployment Descriptor

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <description>Bean für Artikel-Stammdaten</description>
      <ejb-name>Artikel</ejb-name>
      <home>com.klick-and-bau.ArtikelHome</home>
      <remote>com.klick-and-bau.Artikel</remote>
      <ejb-class>com.klick-and-bau.ArtikelBean</ejb-class>
      <persistence-type>Container</persistence-type>
      <prim-key-class>com.klick-and-bau.ArtikelPK</prim-key-class>
      <reentrant>False</reentrant>
      <cmp-field><field-name>id</field-name></cmp-field>
      <cmp-field><field-name>name</field-name></cmp-field>
      <cmp-field><field-name>beschreibung</field-name></cmp-field>
      <cmp-field><field-name>photoURL</field-name></cmp-field>
      <cmp-field><field-name>herstellerId</field-name></cmp-field>
      <cmp-field><field-name>kategorien</field-name></cmp-field>
      <cmp-field><field-name>status</field-name></cmp-field>
      <cmp-field><field-name>preis</field-name></cmp-field>
    </entity>
  </enterprise-beans>
```

Allgemeine  
Angaben

Persistenz

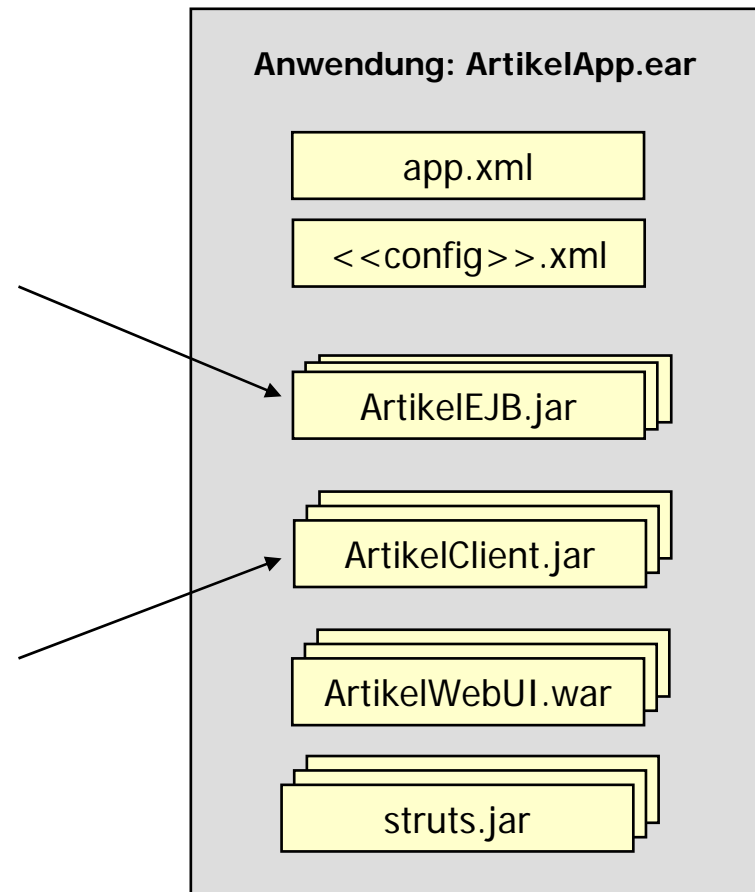
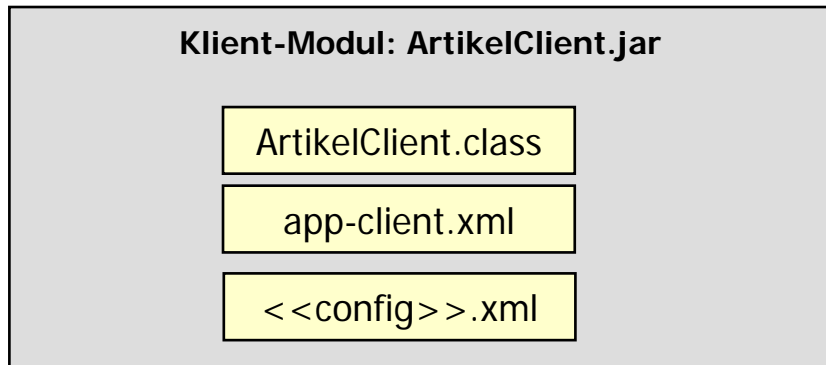
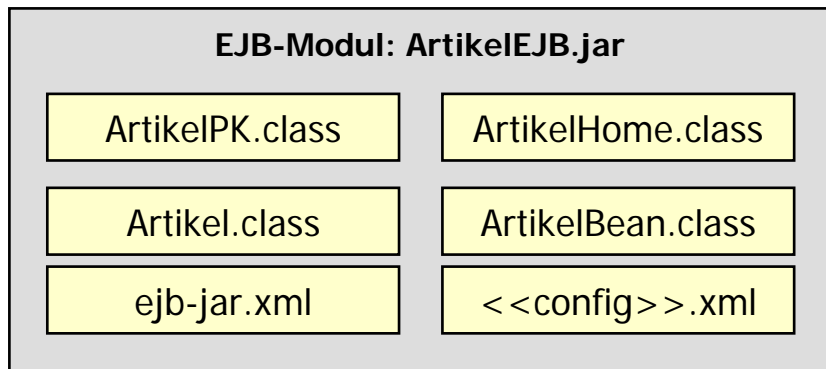
# Artikel-Bean: Deployment Descriptor

```
<assembly-descriptor>
  <security-role>
    <description>Benutzer mit Vollzugriff</description>
    <role-name>Vollzugriff</role-name>
  </security-role>
  <method-permission>
    <role-name>Vollzugriff</role-name>
    <method>
      <ejb-name>Artikel</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
  <container-transaction>
    <method>
      <ejb-name>Artikel</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

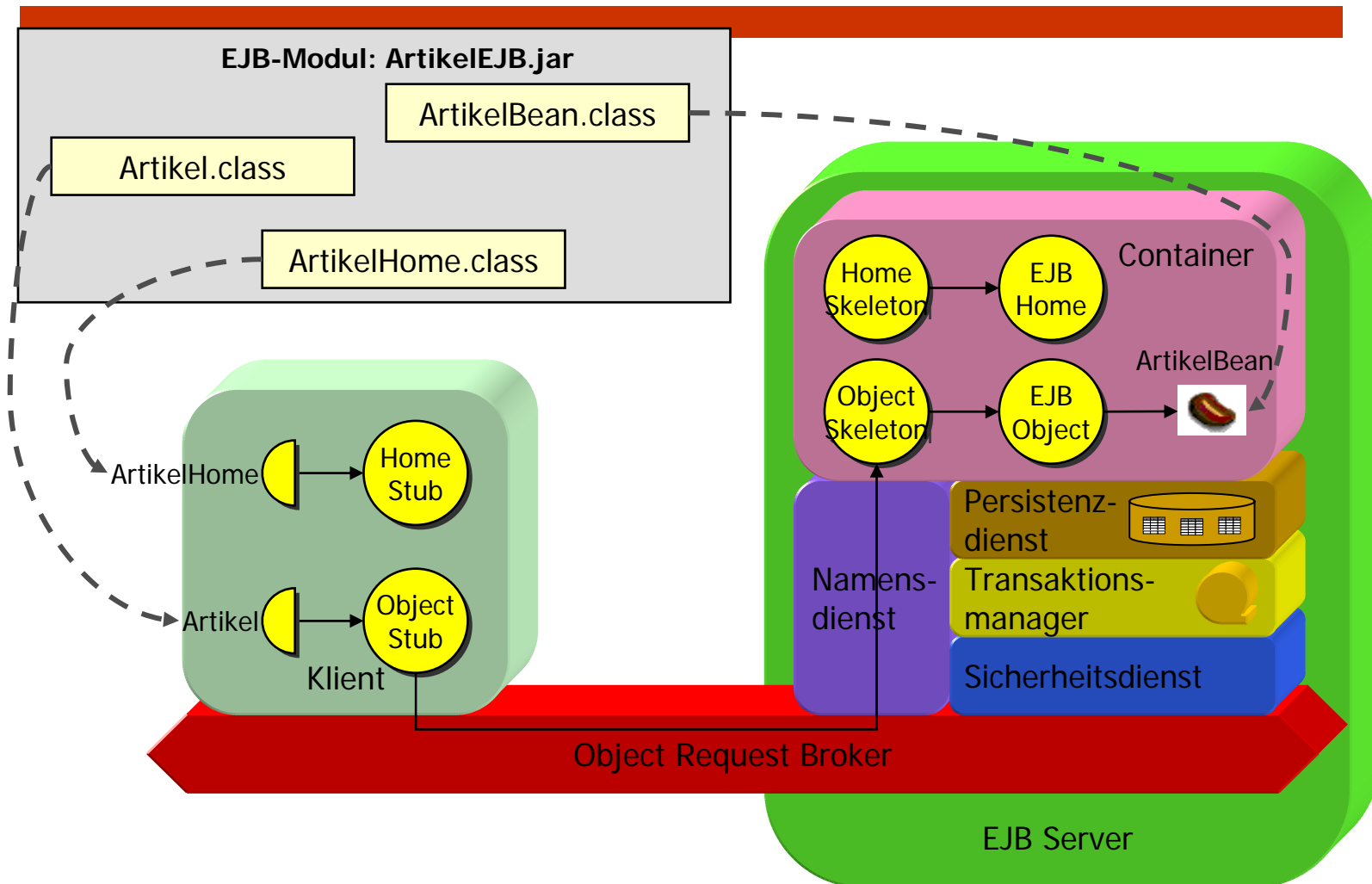
Sicherheit

Transaktionales Verhalten

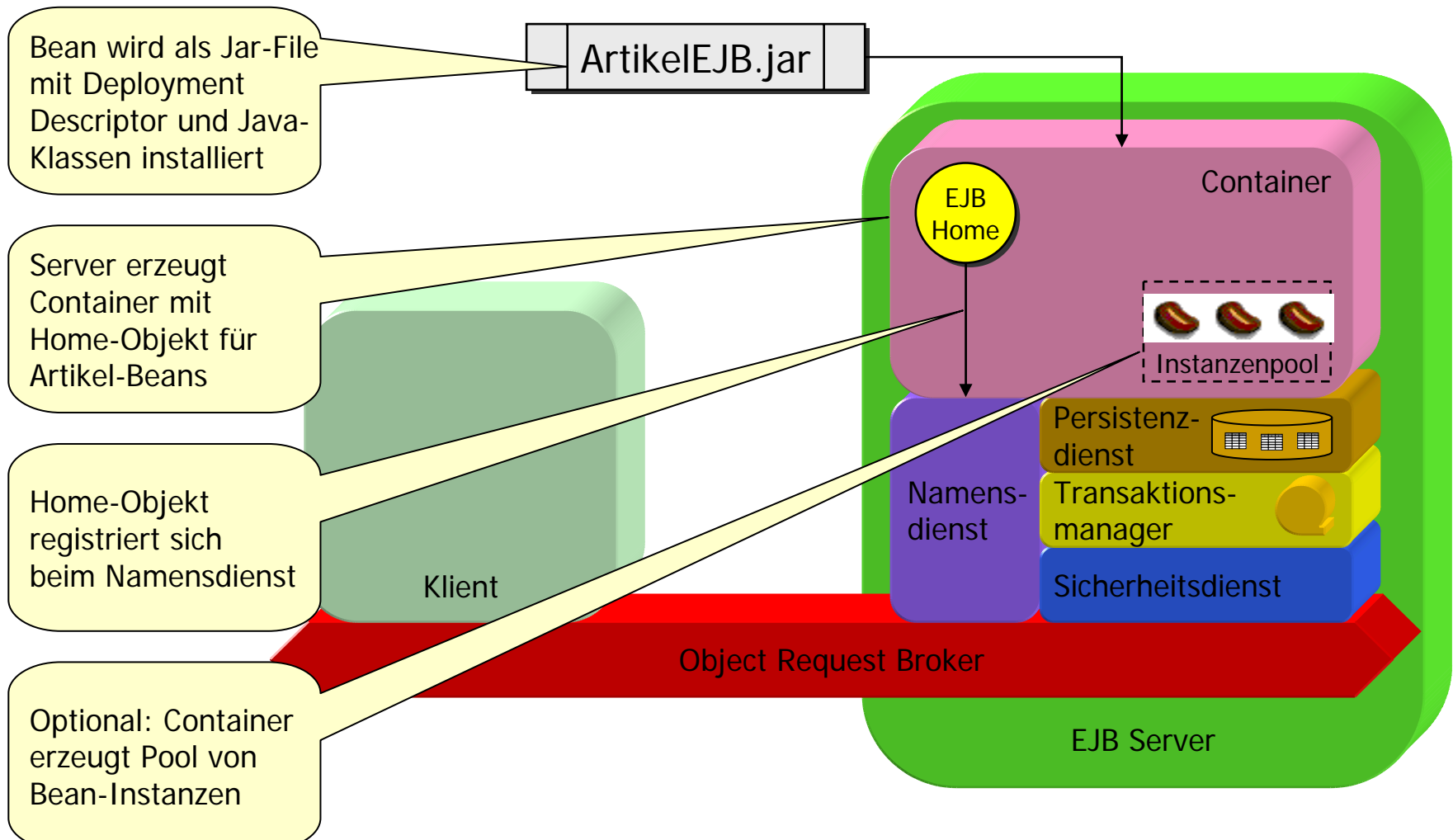
# EJB-Archive



# „Interception“



# Lebenszyklus einer Entity Bean: Installation



# Lebenszyklus einer Entity Bean: Erzeugen einer Bean

```
package com.klick-and-bau;

import java.rmi.RemoteException;
...

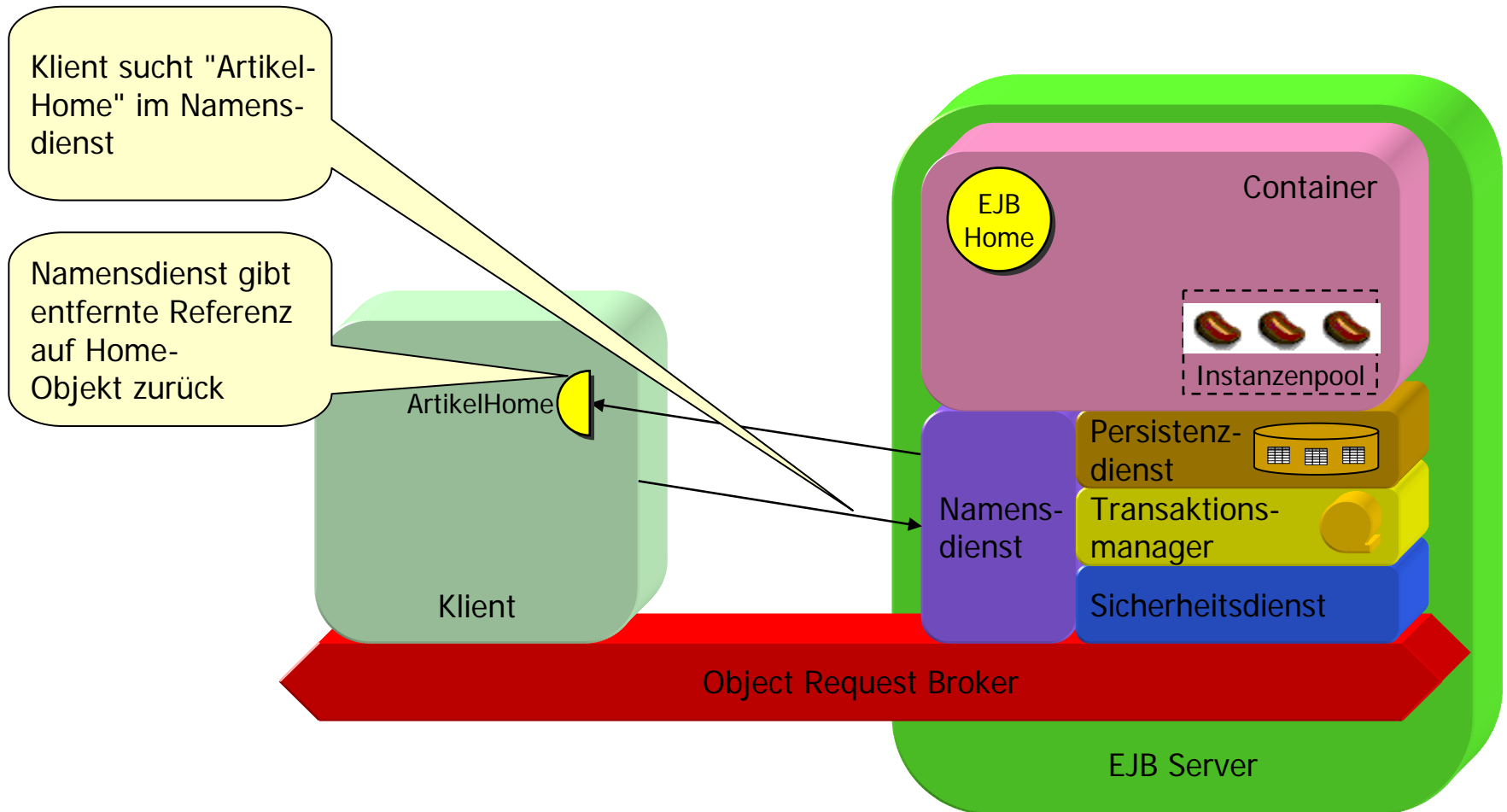
public class ArtikelClient {

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

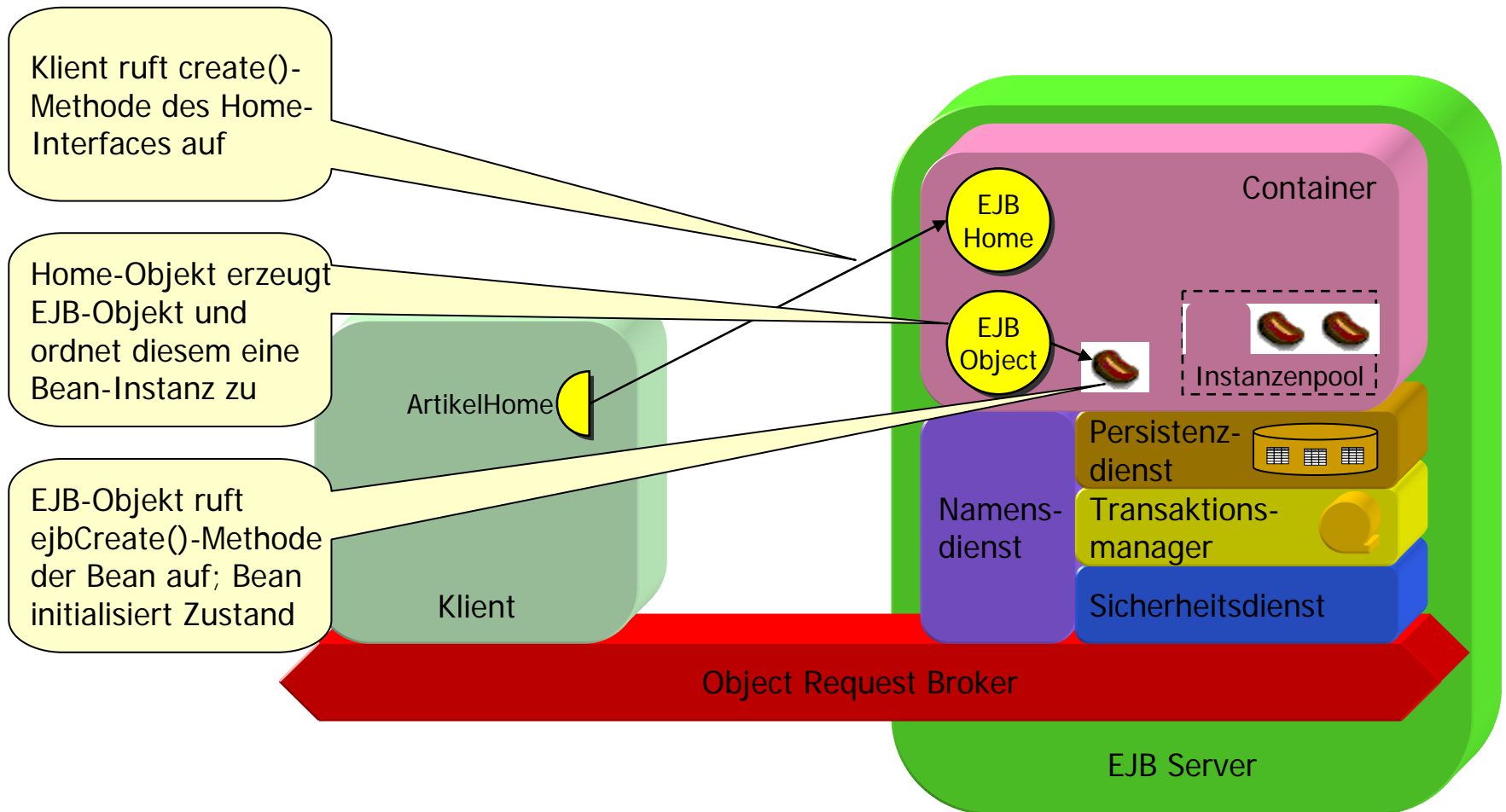
        // Eine Referenz auf ArtikelHome besorgen.
        InitialContext ctx = new InitialContext();
        Object artikelHomeRef = ctx.lookup(„ArtikelHomeInterface“);
        ArtikelHome artikelHome = (ArtikelHome)PortableRemoteObject.narrow(
            ref, ArtikelHome.class);

        // Einen Artikel erzeugen.
        Artikel artikel = artikelHome.create(„77“);
        ...
    }
}
```

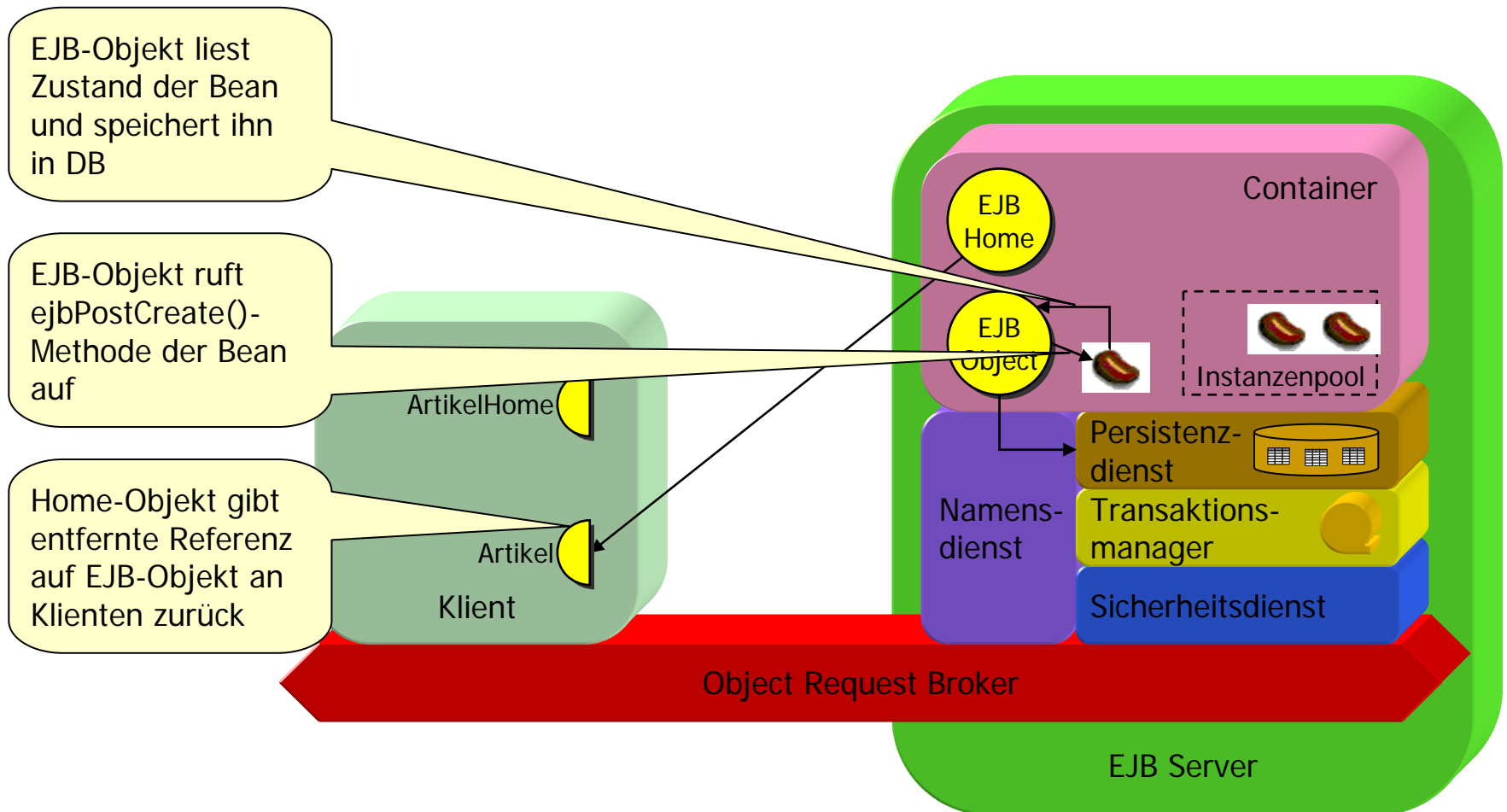
# Lebenszyklus einer Entity Bean: Erzeugen einer Bean



# Lebenszyklus einer Entity Bean: Erzeugen einer Bean



# Lebenszyklus einer Entity Bean: Erzeugen einer Bean



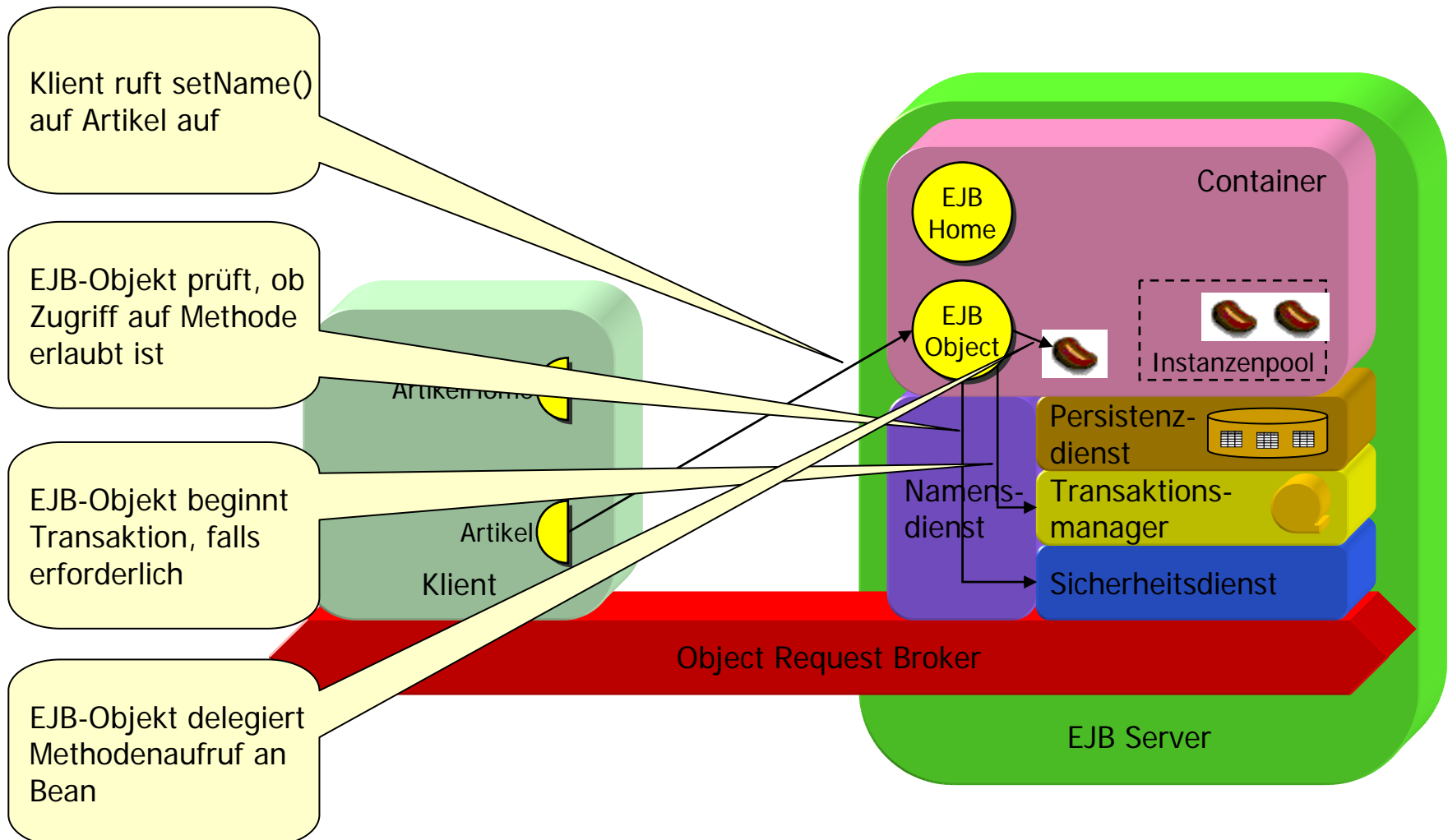
# Lebenszyklus einer Entity Bean: Aufruf einer Geschäftsmethode

...

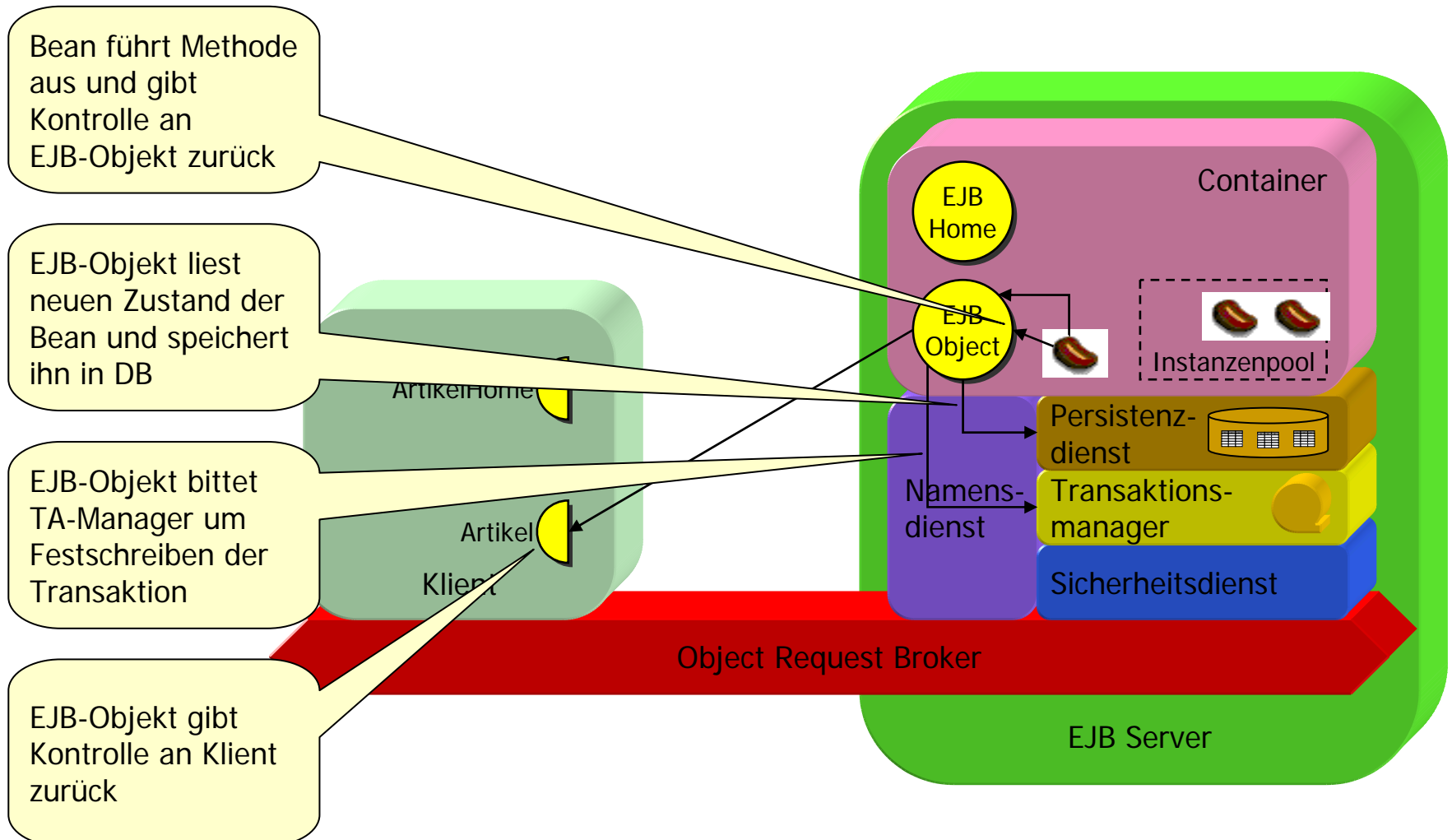
```
// Eine Geschäftsmethode aufrufen.  
String name = artikel.getName();
```

...

# Lebenszyklus einer Entity Bean: Aufruf einer Geschäftsmethode



# Lebenszyklus einer Entity Bean: Aufruf einer Geschäftsmethode



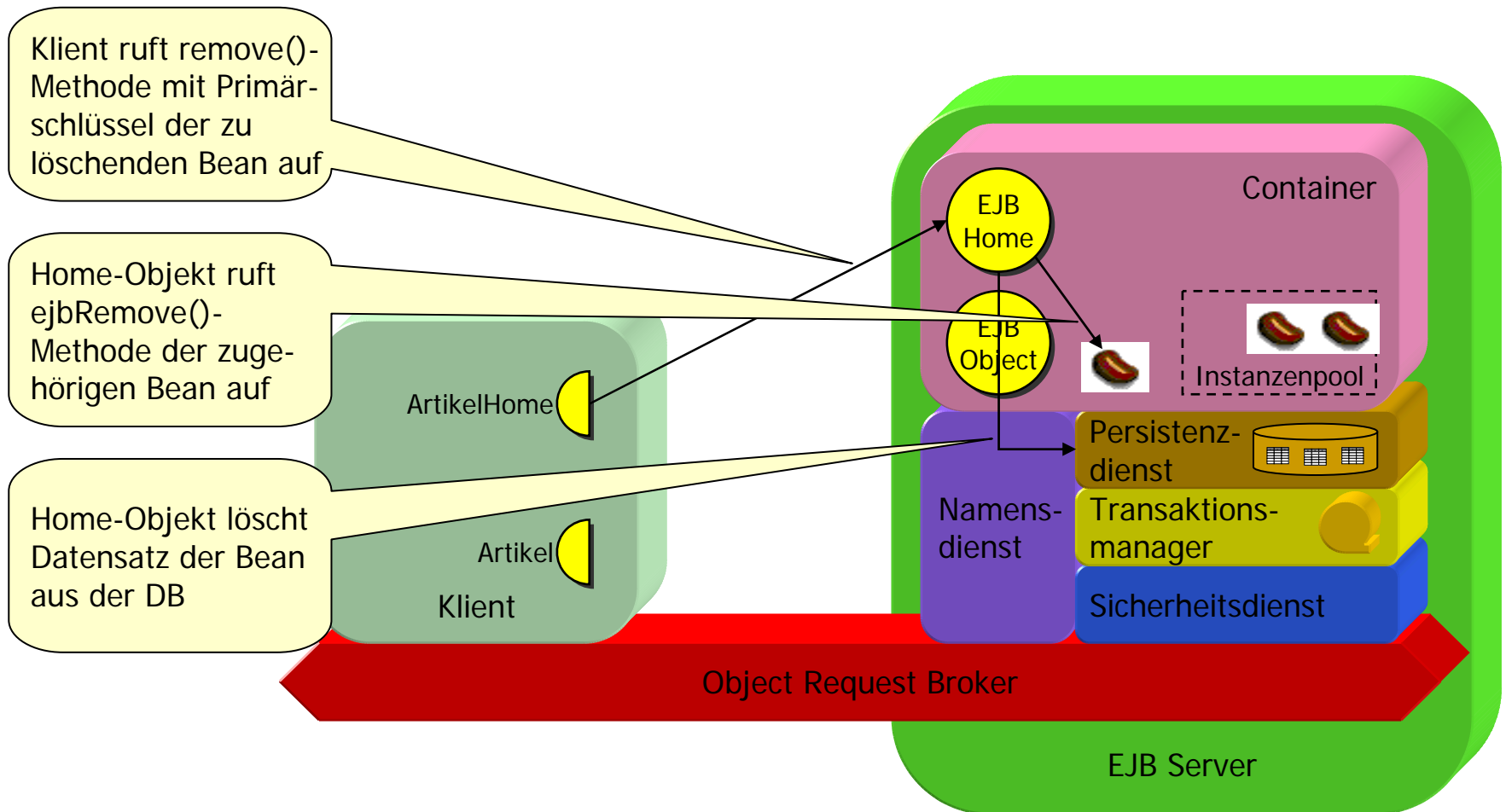
# Lebenszyklus einer Entity Bean: Zerstören einer Bean

...

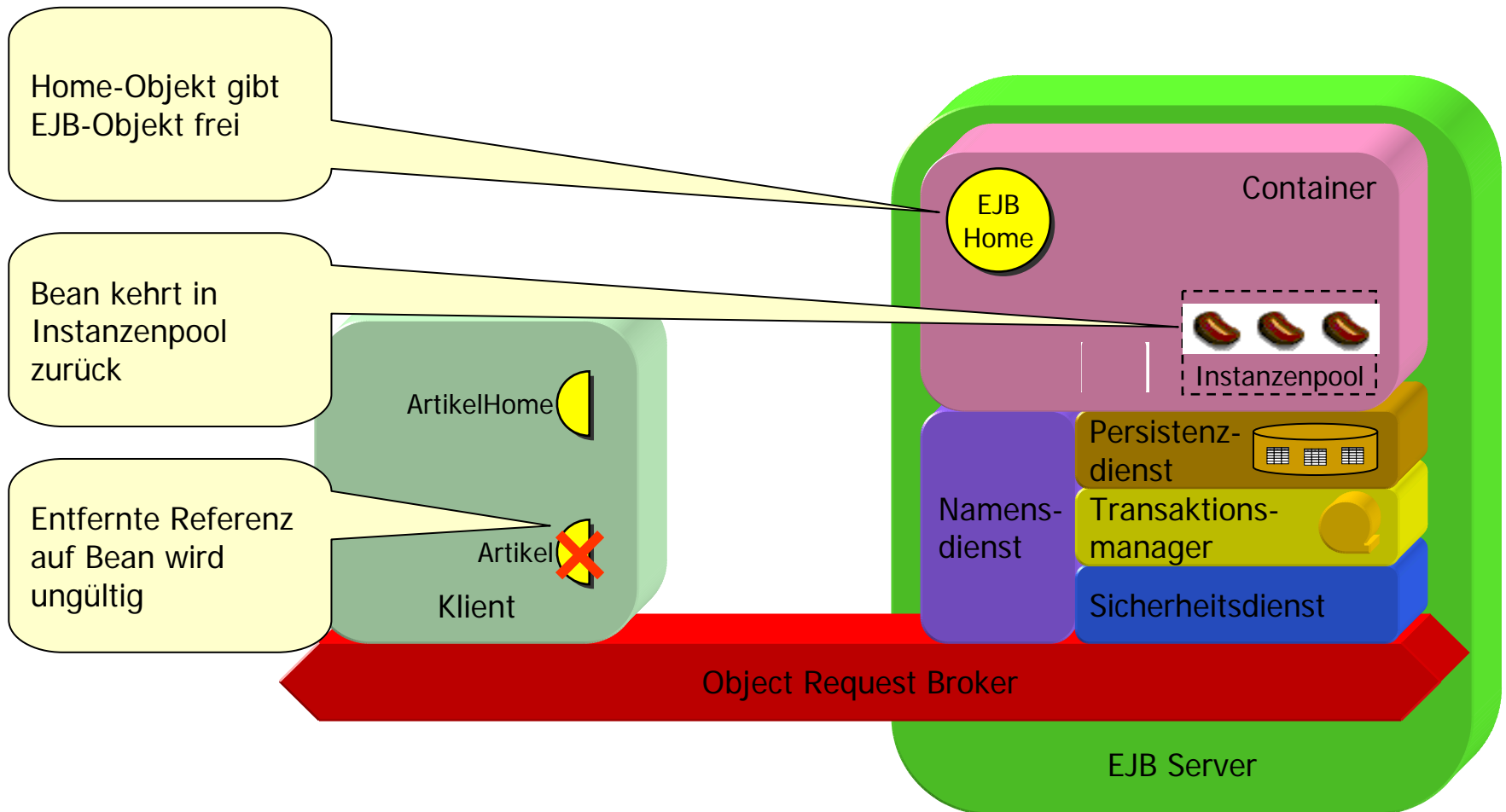
```
// Den Artikel löschen.  
artikel.remove();
```

```
}  
}
```

# Lebenszyklus einer Entity Bean: Zerstören einer Bean



# Lebenszyklus einer Entity Bean: Zerstören einer Bean



# Session Beans: Beispiel

- Session Beans kommen in zwei Varianten:
  - *Stateless* Session Beans beschreiben Prozesse, die mit einem einzigen Methodenaufruf abgewickelt werden können.
  - *Stateful* Session Beans beschreiben Prozesse, die sich über mehrere Methodenaufrufe erstrecken ("Konversationen").
- Beispiel: Adressprüfer
  - Die Adressprüfer-Bean ist stateless und dient zur Prüfung von Adressen anhand einer Postleitzahl-Datenbasis.
- Beispiel: Kaufvorgang
  - Die Kaufvorgang-Bean ist stateful. Sie implementiert Methoden zum Füllen des Warenkorbs, Festlegen der Zahlungsmodalität, Auslösen der Zahlung etc.

# Adressprüfer-Bean: Prinzip

- Die folgende Postleitzahl-Tabelle stehe zur Verfügung:

```
POSTLEITZAHLEN(  
  Postleitzahl NUMERIC(5,0),  
  Stadt        VARCHAR(50),  
  Strasse      VARCHAR(100),  
  Min_Nummer   NUMERIC(5,0) DEFAULT 0,  
  Max_Nummer   NUMERIC(5,0) DEFAULT 99999  
)
```

- Zur (naiven) Prüfung einer Adresse (plz, std, str, num) wird geschaut, ob ein passender Datensatz existiert:

```
select * from POSTLEITZAHLEN  
where Postleitzahl = plz  
and Stadt = std and Strasse = str  
and Min_Nummer <= num and Max_Nummer >= num
```

# Adressprüfer-Bean: Remote Interface

```
package com.klick-and-bau;  
  
import java.rmi.RemoteException;  
  
public interface Adressprüfer extends javax.ejb.EJBObject {  
  
    public boolean gültig(int plz, String std, String str, int num)  
        throws RemoteException;  
  
}
```

# Adressprüfer-Bean: Home Interface

```
package com.klick-and-bau;

import java.rmi.RemoteException;
import javax.ejb.CreateException;

public interface AdressprüferHome extends javax.ejb.EJBHome {

    public Adressprüfer create() throws RemoteException, CreateException;

    // find()-Methoden gibt es für Session Beans nicht,
    // remove()-Methoden werden von EJBHome geerbt.
}
```

# Adressprüfer-Bean: Beanklasse

```
package com.klick-and-bau;

import java.sql.*;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;
import javax.ejb.EJBException;

public class AdressprüferBean implements javax.ejb.SessionBean {

    // Zustandsvariablen (auch in stateless Beans erlaubt, solange
    // nicht versucht wird, klientenspezifischen Zustand zu halten)

    // JDBC-Verbindung zur PLZ-Datenbasis
    Connection con = null;

    // Vorbereitete SQL-Suchanfrage
    PreparedStatement ps = null;
```

# Adressprüfer-Bean: Beanklasse (Forts.)

```
// Initialisierungsmethode - wird vom Container irgendwann zwischen
// dem Anlegen der Instanz und dem Aufruf der ersten Geschäftsmethode
// aufgerufen.
// Hier: Öffne Verbindung zu PLZ-Datenbasis und bereite SQL-Anfrage vor.

public void ejbCreate() {
    try {
        InitialContext ctx = new InitialContext();
        DataSource ds = (DataSource) ctx.lookup("java:comp/env/jdbc/plzDB");
        con = ds.getConnection();
    } catch (NamingException ne) {
        throw new EJBException(ne);
    }
    ps = con.prepareStatement(
        "select * from POSTLEITZAHLEN"
        + "where Postleitzahl = ?"
        + "and Stadt = ? and Strasse = ?"
        + "and Min_Nummer <= ? and Max_Nummer >= ?"
    );
}
```

# Adressprüfer-Bean: Beanklasse (Forts.)

```
// Implementierung der Geschäftsmethode.  
  
public boolean gültig(int plz, String std, String str, int num) {  
    try {  
        ps.setInt(1,plz);  
        ps.setString(2,std);  
        ps.setString(3,str);  
        ps.setInt(4,num);  
        ps.setInt(5,num);  
        ResultSet res = ps.executeQuery();  
        // res.next() gibt false zurück, wenn kein Datensatz vorliegt.  
        return res.next();  
    } catch (SQLException se) {  
        throw new EJBException(se);  
    }  
}
```

# Adressprüfer-Bean: Beanklasse (Forts.)

```
// Lebenszyklusmethoden - werden vom Container aufgerufen.  
  
public void ejbRemove() {  
    // Bean wird gleich gelöscht - gebe Ressourcen frei.  
    try {  
        if (ps != null) ps.close();  
        if (con != null) con.close();  
    } catch (SQLException se) {  
    }  
}  
  
// Einige weitere Lebenszyklusmethoden sind nicht gezeigt.  
}
```

# Adressprüfer-Bean: Deployment Descriptor

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <description>Bean zur Adressprüfung</description>
      <ejb-name>Adressprüfer</ejb-name>
      <home>com.klick-and-bau.AdressprüferHome</home>
      <remote>com.klick-and-bau.Adressprüfer</remote>
      <ejb-class>com.klick-and-bau.AdressprüferBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Container</transaction-type>
      <resource-ref>
        <description>Postleitzahl-Datenbasis</description>
        <res-ref-name>jdbc/plzDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
      </resource-ref>
    </session>
  </enterprise-beans>
```

Allgemeine  
Angaben

Benötigte  
Ressourcen

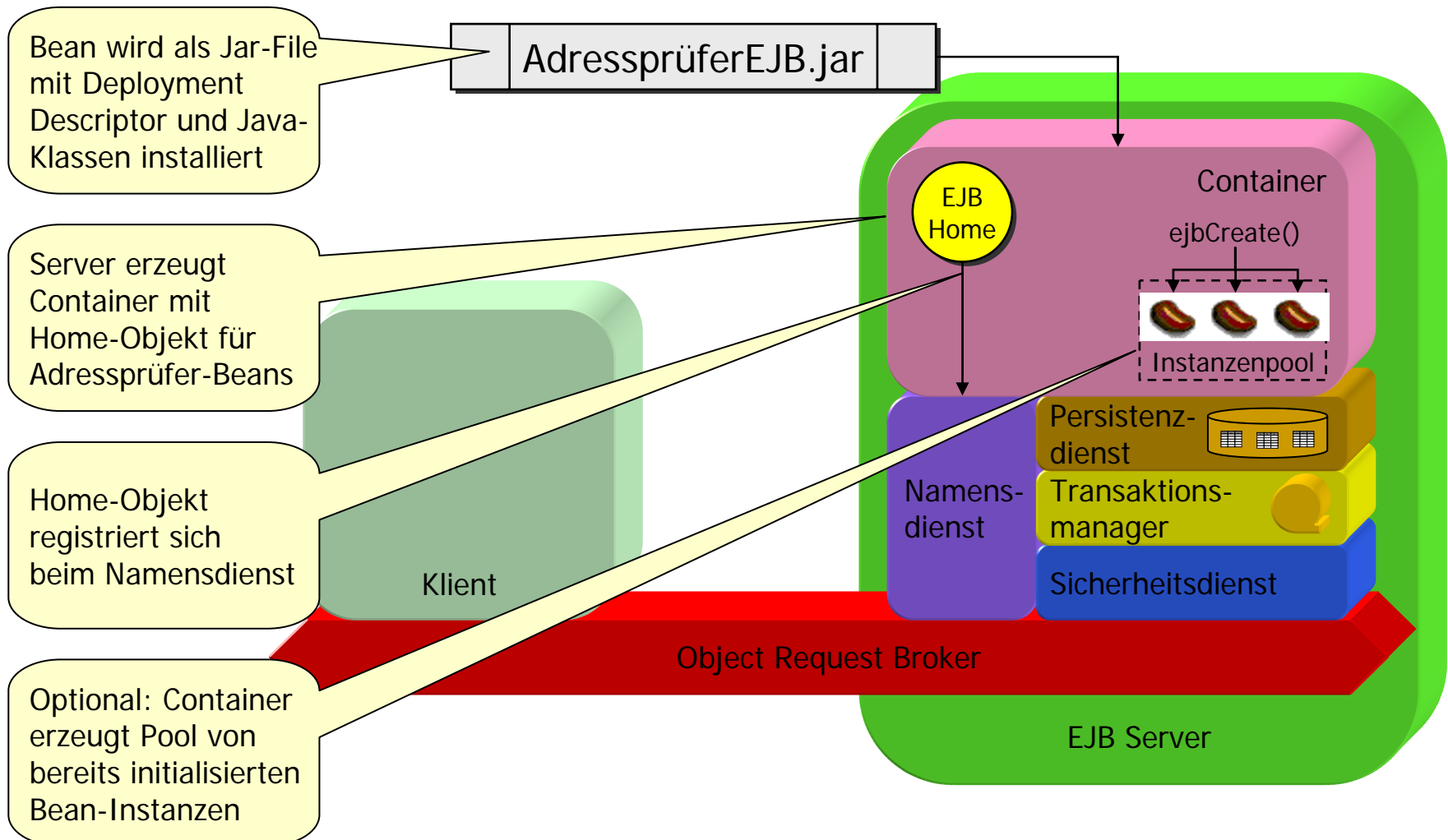
# Adressprüfer-Bean: Deployment Descriptor (Forts.)

```
<assembly-descriptor>
  <security-role>
    <description>Benutzer mit Vollzugriff</description>
    <role-name>Vollzugriff</role-name>
  </security-role>
  <method-permission>
    <role-name>Vollzugriff</role-name>
    <method>
      <ejb-name>Adressprüfer</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
  <container-transaction>
    <method>
      <ejb-name>Adressprüfer</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

Sicherheit

Transaktionales Verhalten

# Lebenszyklus einer Stateless Sess. Bean: Installation



# Lebenszyklus einer Stateless Sess. Bean: Erzeugen einer Bean

```
package com.klick-and-bau;

import java.rmi.RemoteException;
...

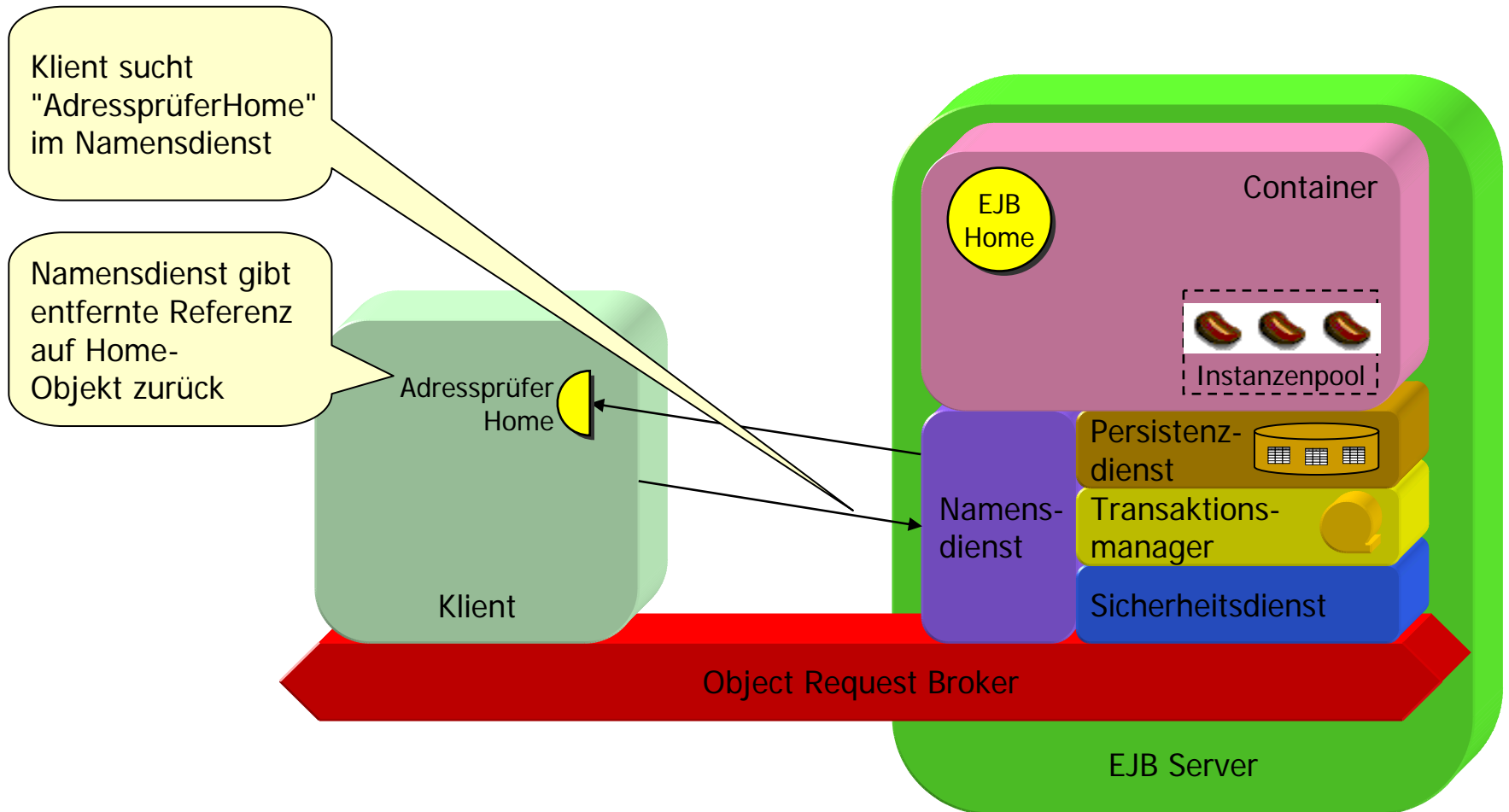
public class AdressprüferClient {

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

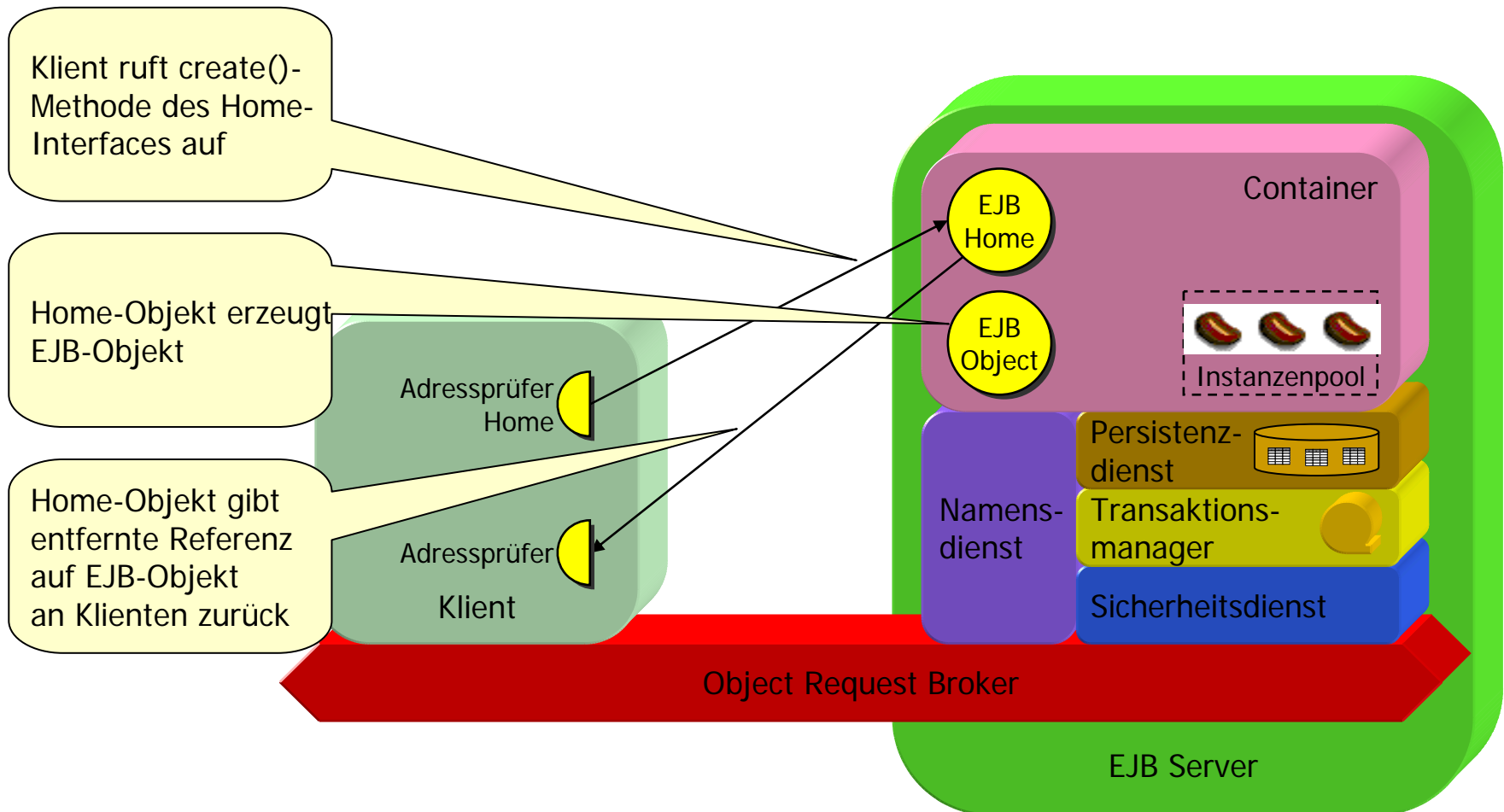
        // Eine Referenz auf AdressprüferHome besorgen.
        InitialContext ctx = new InitialContext();
        Object prüferHomeRef = ctx.lookup(„AdressprüferHomeInterface“);
        AdressprüferHome prüferHome =
            (AdressprüferHome)PortableRemoteObject.narrow(ref, ArtikelHome.class);

        // Einen Adressprüfer erzeugen.
        Adressprüfer prüfer = prüferHome.create();
        ...
    }
}
```

# Lebenszyklus einer Stateless Sess. Bean: Erzeugen einer Bean



# Lebenszyklus einer Stateless Sess. Bean: Erzeugen einer Bean (Forts.)



# Lebenszyklus einer Stateless Sess. Bean: Aufruf einer Geschäftsmethode

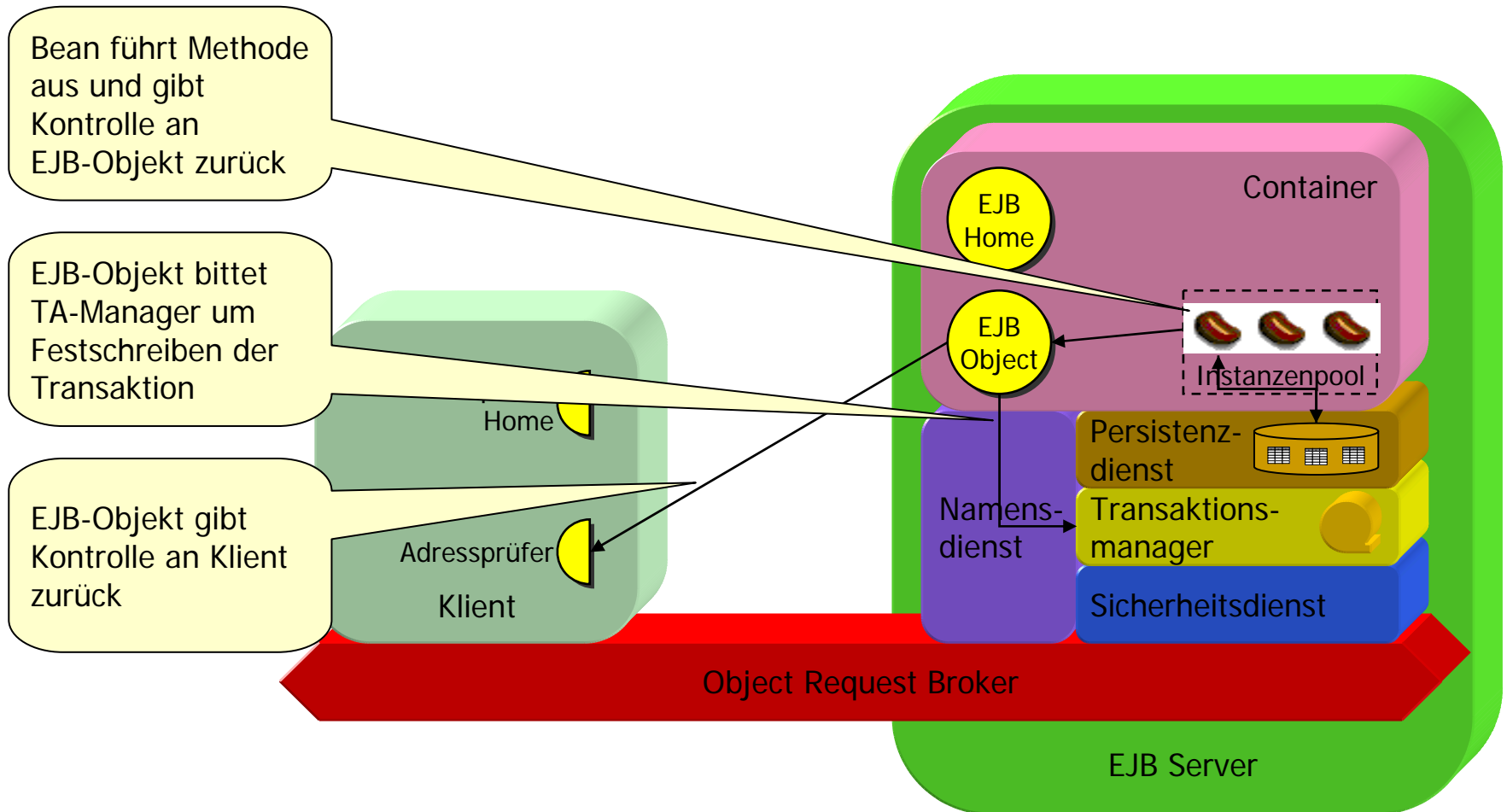
...

```
// Eine Adresse auf Gültigkeit prüfen.  
boolean adresseIstOk = prüfer.gültig(76131, „Karlsruhe“,  
    „Haid-und-Neu-Straße“, 10);
```

...



# Lebenszyklus einer Stateless Sess. Bean: Aufruf einer Geschäftsmethode (Forts.)



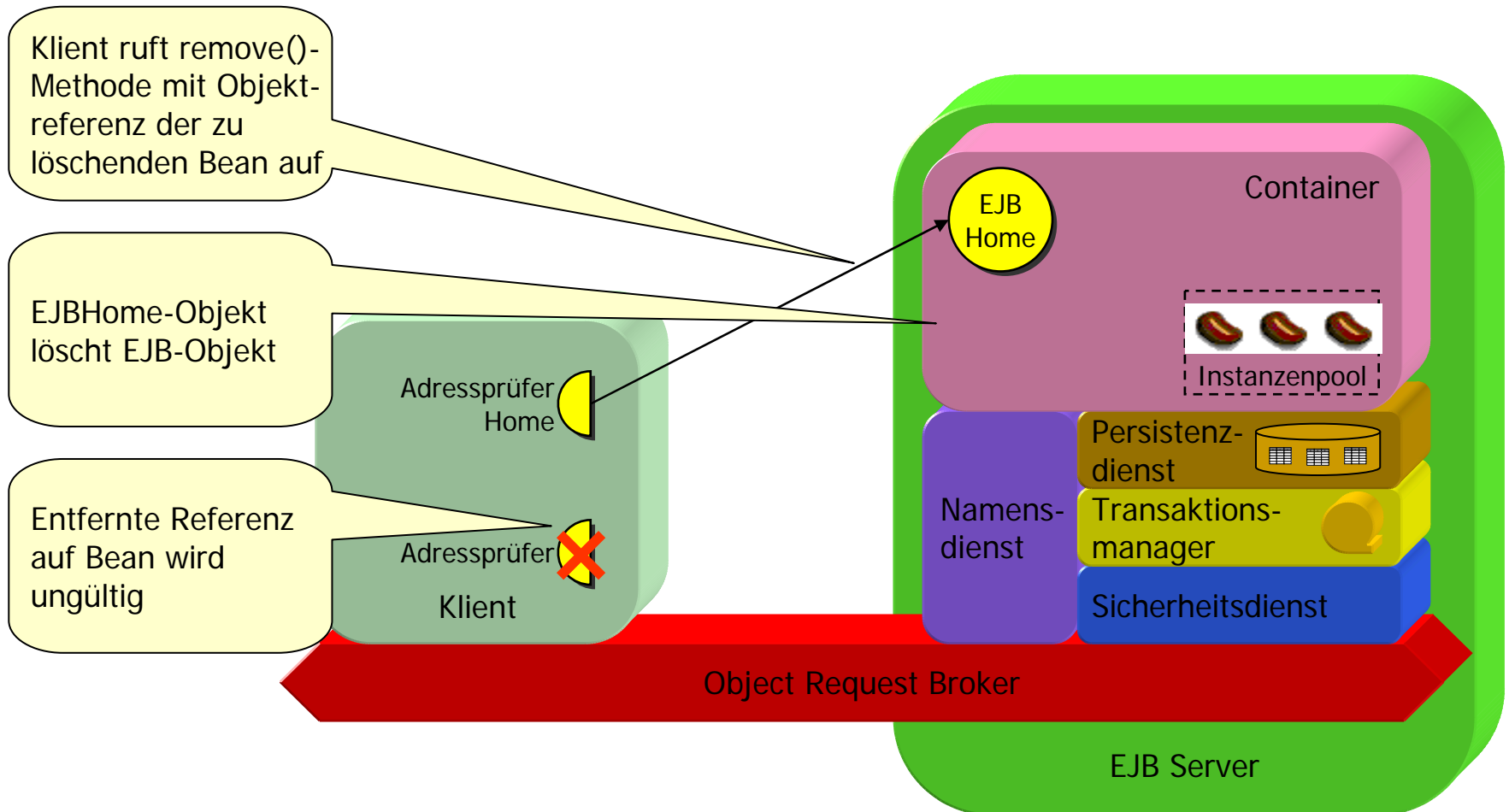
# Lebenszyklus einer Stateless Sess. Bean: Zerstören einer Bean

...

```
// Den Adressprüfer löschen.  
prüfer.remove();
```

```
}  
}
```

# Lebenszyklus einer Stateless Sess. Bean: Zerstören einer Bean



# **Einsatz und Kritik von Enterprise JavaBeans**



# Vorteile von EJB



---

- Bewährte Vorgehensweisen wurden öffentlich wiederverwendbar gemacht
- EJB-Server bieten eine fertige Infrastruktur für umfangreiche, verteilte Anwendungen
- Know-how ist „portabel“ geworden:
  - leichter Umgang mit mehreren Plattformen
  - stärkerer Erfahrungsaustausch unter den Entwicklern
  - erhöhtes Verständnis fremder Software
- Deutlich vereinfachte Portierung von Anwendungen

# Nachteile von EJB



- Hoher Lernaufwand
- Aufwändige Entwicklung
- Schlechte Handhabung und Fehleranfälligkeit
  - Deployment Descriptoren, Testen, Debugging
- Konzeptuelle Schwächen
  - Technische und fachliche Aspekte z.T. stark verzahnt
  - Warum braucht man sonst so viele J2EE-Patterns?
- Begrenzte Integrierbarkeit von und in bestehende IT-Landschaften
  - siehe später: Dienstorientierte Architektur und WebServices

# Lösungen: Effiziente Kommunikation

## ■ Problem:

- Bis EJB 2.0 grundsätzlich entfernter Zugriff auf EJBs  
⇒ Feingranulare Zugriffe ineffizient (getter/setter etc.)

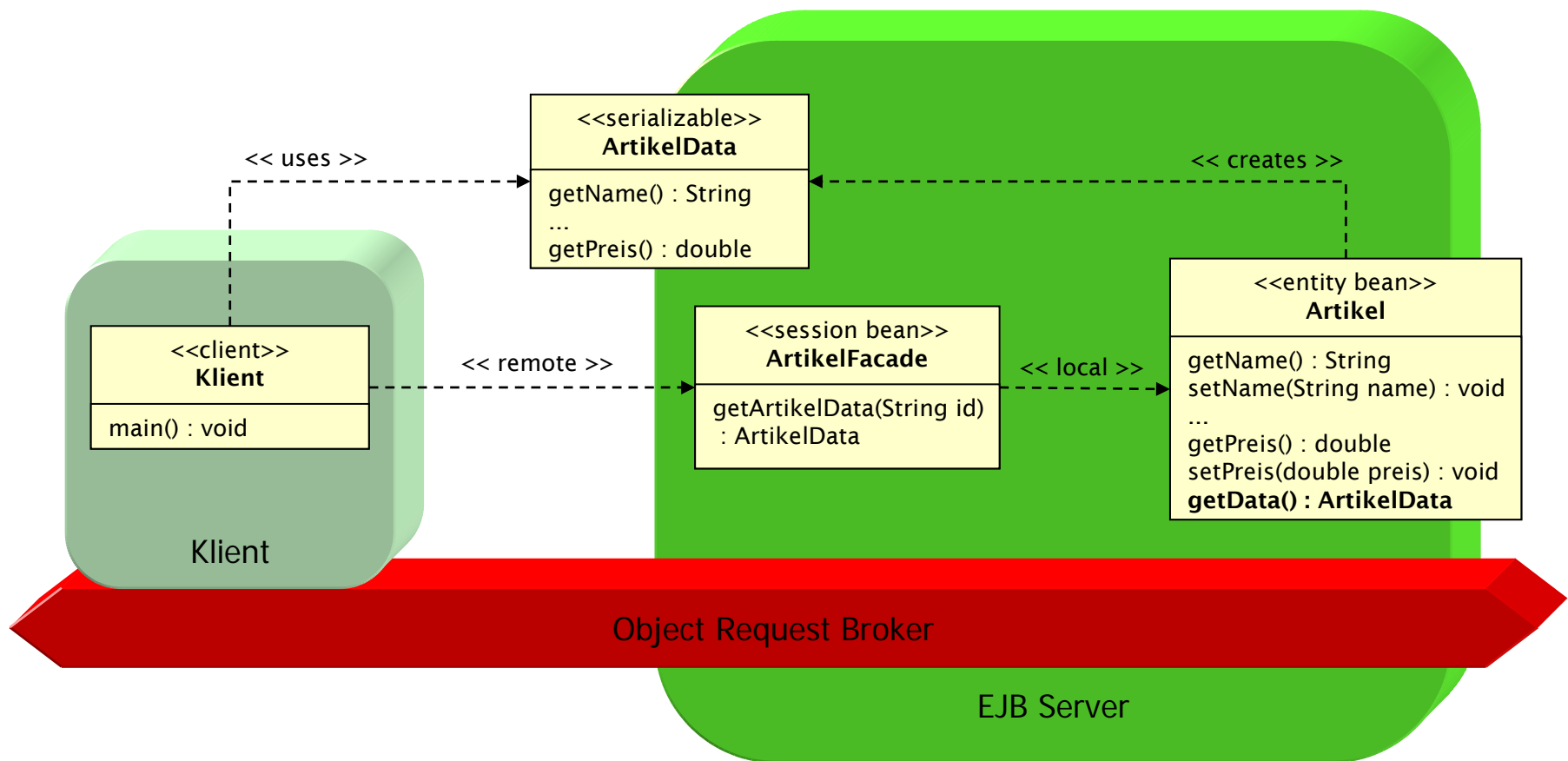
## ■ Lösungsansatz:

- Einführung lokaler Schnittstellen in EJB 2.0 zur Kommunikation innerhalb des Servers
  - `javax.ejb.EJBLocalObject`, `javax.ejb.EJBLocalHome`
- J2EE-Muster „Facade“ und „TransferObject“ für den Datenaustausch zwischen Server und Klient

## ■ Nebenwirkungen:

- Unerwünschte Kopplung von fachlichen und technischen Aspekten
- Ein nochmals erhöhter Implementierungsaufwand

# Muster „Facade“ und „TransferObject“



# Lösungen: Aufwandsreduktion

## ■ Problem:

- Redundante/überflüssige Code-Passagen sind zu schreiben
- Komplexe und fehleranfällige Elemente (→ DD)

## ■ Lösungsansätze:

- xdoclet – Generierung von Schnittstellen und DD aus einer annotierten Bean-Implementierung
- Model-Driven Architecture (MDA) – Generierung kompletter EJBs aus einem annotierten UML-Klassendiagramm
- EJB 3.0 – EJBs sind Java-5-annotierte POJOs und POJIs

```
@Remote @Stateless public class HelloWorldBean {  
    public String sayHello(String s) {  
        System.out.println("Hello:" + s);  
    }  
}
```

# Zusammenfassung

- Unterstützung moderner Geschäftsprozesse ist eine sehr komplexe und kostspielige Angelegenheit
- Grundlegende Lösungsansätze
  - Mehrschichtenarchitekturen
  - Modularisierung
  - „Separation of Concerns“
  - Standardisierung
- Komponentenumgebungen sind ein Versuch, diese Ansätze in ein ganzheitliches Rahmenwerk zu gießen
- J2EE und Enterprise JavaBeans sind Java-basierte Standards für Komponentenumgebungen



Applet

WAP

HTTP

www.klick-and-bau.com

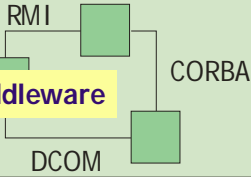
CGI JSP ASP Servlet

XSL-FO

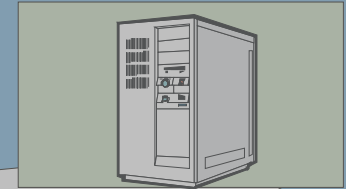
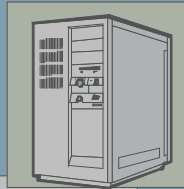
Datenbanken und das Web I

Datenbanken und das Web II

Middleware



XSL-T



Architekturen und Systeme zur Informationsintegration

Mediator

Datenaustausch und -zugriff mit XML

XML Schema XQuery

Semantische Integration

OEM OEM XML Schema XQuery

Backend-Anbindungen

JDBC

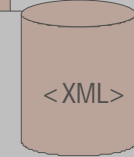
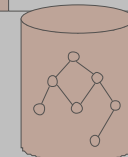
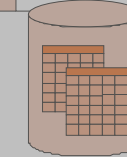
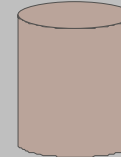
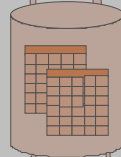
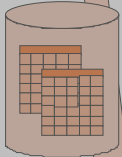
CICS

XML Schema

XML

OEM

XML Schema XQuery



RDBMS

HOST

RDBMS

DBMS

RDBMS

OODBMS

XML-DBMS

## ■ Bücher:

- Backschat, M., Gardon, O. (2002): *Enterprise JavaBeans*. Spektrum Akademischer Verlag.
- Alur, D., Crupi, J., Malks, D. (2002): *Core J2EE Patterns*. Prentice Hall.
- Bien, A. (2003): *J2EE Patterns*. Addison-Wesley.
- Johnson, R. (2004): *J2EE Development without EJB*. Wiley Publishing.

## ■ News und Artikel im Web:


- [www.theserverside.com](http://www.theserverside.com), [www.ejbsig.de](http://www.ejbsig.de)
- [www.javamagazin.de](http://www.javamagazin.de)

## ■ Zeitschriften:

- JavaMagazin, JavaSpektrum
- Bien, A.: *J2EE Patterns*. In: JavaMagazin 11/2002-08/2003



**Fragen?**



*Übrigens, suche Hiwis für das Projekt CollaBaWue!*

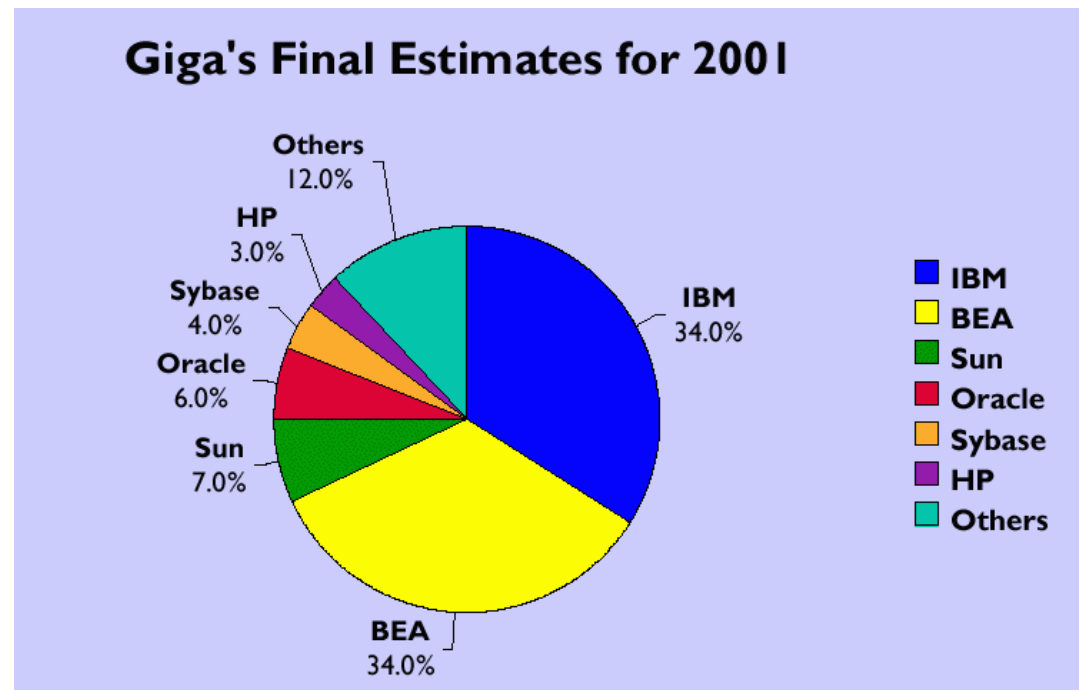


# **Ergänzende Folien für Interessierte**



# Applikationsserver: Beispiele

- Kommerzielle Applikationsserver:
  - BEA WebLogic, IBM WebSphere, Oracle Application Server, Sun ONE, ...
- Open Source:
  - JBoss
  - JOnAS
  - Geronimo
  - ...



Quelle: C. Mohan, „Tutorial: Application Server and Associated Technologies“, VLDB 2002.

# Stateful Session Beans



# Warenkorb-Bean: Prinzip

- Aufgabe: Verwaltung einer Liste von (Artikelnummer, Stückzahl) - Paaren
- Businessmethoden:
  - Einfügen zusätzlicher Artikelnummern und Stückzahlen
  - Ändern von Stückzahlen (und Ausfügen der Artikelnummer, falls Stückzahl auf 0 gesetzt wird)
  - Anzeigen des Inhalts
- Vereinfachende Annahme: Bestellvorgänge werden von separater Bestellauslösung-Bean gehandhabt (hier nicht gezeigt)

# Warenkorb-Bean: Remote Interface

```
package com.klick-and-bau;

import java.rmi.RemoteException;
import com.klick-and-bau.NoSuchArtikelException;
import java.util.Map;

public interface Warenkorb extends javax.ejb.EJBObject {

    public void addArtikel(String artNr, int anzahl)
        throws RemoteException, NoSuchArtikelException;
    public void changeAnzahl(String artNr, int anzahl)
        throws RemoteException, NoSuchArtikelException;
    public Map getInhalt()
        throws RemoteException;

}
```

# Warenkorb-Bean: Home Interface

```
package com.klick-and-bau;

import java.rmi.RemoteException;
import javax.ejb.CreateException;

public interface WarenkorbHome extends javax.ejb.EJBHome {

    public Warenkorb create() throws RemoteException, CreateException;

    // find()-Methoden gibt es für Session Beans nicht,
    // remove()-Methoden werden von EJBHome geerbt.
}
```

# Warenkorb-Bean: Beanklasse

```
package com.klick-and-bau;

import java.util.Map;
import java.util.HashMap;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.rmi.PortableRemoteObject;
import java.rmi.RemoteException;
import javax.ejb.EJBException;
import javax.ejb.FinderException;
```

# Warenkorb-Bean: Beanklasse (Forts.)

```
public class WarenkorbBean implements javax.ejb.SessionBean {

    // Zustandsvariablen

    // Inhalt des Warenkorbes als Hashtabelle (die Artikelnummern sind
    // die Schlüssel, die Anzahlen die Werte).
    // Die Verwendung von Artikelnummern statt Referenzen auf Artikel-Beans
    // macht die Bean effizienter, weil der Container nicht so viele
    // ArtikelBean-Instanzen vorhalten muss.

    HashMap contents = null;

    // Referenz auf ArtikelHome-Objekt, um Existenz von Artikelnummern
    // prüfen zu können.

    ArtikelHome artHome = null;
```

# Warenkorb-Bean: Beanklasse (Forts.)

```
// Initialisierungsmethode - wird vom Container aufgerufen,  
// wenn Klient create()-Methode des Home Interface aufruft.  
// Hier: initialisiere Instanzvariablen.  
  
public void ejbCreate() {  
  
    // Lege leere java.util.HashMap für den Inhalt an.  
    contents = new HashMap();  
  
    // Hole Referenz auf ArtikelHome-Objekt vom Namensdienst.  
    try {  
        InitialContext ctx = new InitialContext();  
        Object ref = ctx.lookup("java:comp/env/ejb/ArtikelHome");  
        // Der Aufruf von narrow() statt eines einfachen Casts ist  
        // aus Gründen der Interoperabilität mit CORBA erforderlich.  
        artHome = (ArtikelHome)  
            PortableRemoteObject.narrow(ref, ArtikelHome.class);  
    } catch (NamingException ne) {  
        throw new EJBException(ne);  
    }  
}
```

# Warenkorb-Bean: Beanklasse (Forts.)

```
// Implementierung der Geschäftsmethoden.

public void addArtikel(String artNr, int anzahl)
    throws NoSuchArtikelException {

    // Prüfe, ob Artikelnummer existiert.

    try {
        artHome.findByPrimaryKey(artNr);
    } catch (FinderException fe) {
        throw new NoSuchArtikelException(fe);
    } catch (RemoteException re) {
        throw new EJBException(re);
    }

    // Füge Artikelnummer und Anzahl in Map ein.

    if (anzahl < 1) anzahl = 1;
    Integer alteAnzahl = (Integer) contents.get(artNr);
    if (alteAnzahl != null) anzahl += alteAnzahl.intValue();
    contents.put(artNr, new Integer(anzahl));
}
```

# Warenkorb-Bean: Beanklasse (Forts.)

```
// Implementierung der Geschäftsmethoden.

public void changeAnzahl(String artNr, int anzahl)
    throws NoSuchArtikelException {

    // Prüfe, ob Artikelnummer bereits im Warenkorb ist.

    if (contents.get(artNr) == null)
        throw new NoSuchArtikelException();

    // Füge Artikelnummer und neue Anzahl in Warenkorb ein.

    if (anzahl < 1) anzahl = 1;
    contents.put(artNr, new Integer(anzahl));
}

public Map getInhalt() {

    return contents.clone();

}
```

# Warenkorb-Bean: Beanklasse (Forts.)

```
// Lebenszyklusmethoden - werden vom Container aufgerufen.  
  
public void ejbRemove() {  
  
    // Bean wird gleich gelöscht - gebe Verweise frei.  
  
    contents = null;  
    artHome = null;  
}  
  
// Einige weitere Lebenszyklusmethoden sind nicht gezeigt.  
}
```

# Warenkorb-Bean: Deployment Descriptor

```
<?xml version="1.0"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise
JavaBeans 1.1//EN" "http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
<ejb-jar>
  <enterprise-beans>
    <session>
      <description>Virtueller Warenkorb</description>
      <ejb-name>Warenkorb</ejb-name>
      <home>com.klick-and-bau.WarenkorbHome</home>
      <remote>com.klick-and-bau.Warenkorb</remote>
      <ejb-class>com.klick-and-bau.WarenkorbBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
      <ejb-ref>
        <ejb-ref-name>ejb/ArtikelHome</res-ref-name>
        <ejb-ref-type>Entity</res-type>
        <home>com.klick-and-bau.ArtikelHome</home>
        <remote>com.klick-and-bau.Artikel</remote>
      </ejb-ref>
    </session>
  </enterprise-beans>
```

Allgemeine  
Angaben

Benötigte  
Ressourcen

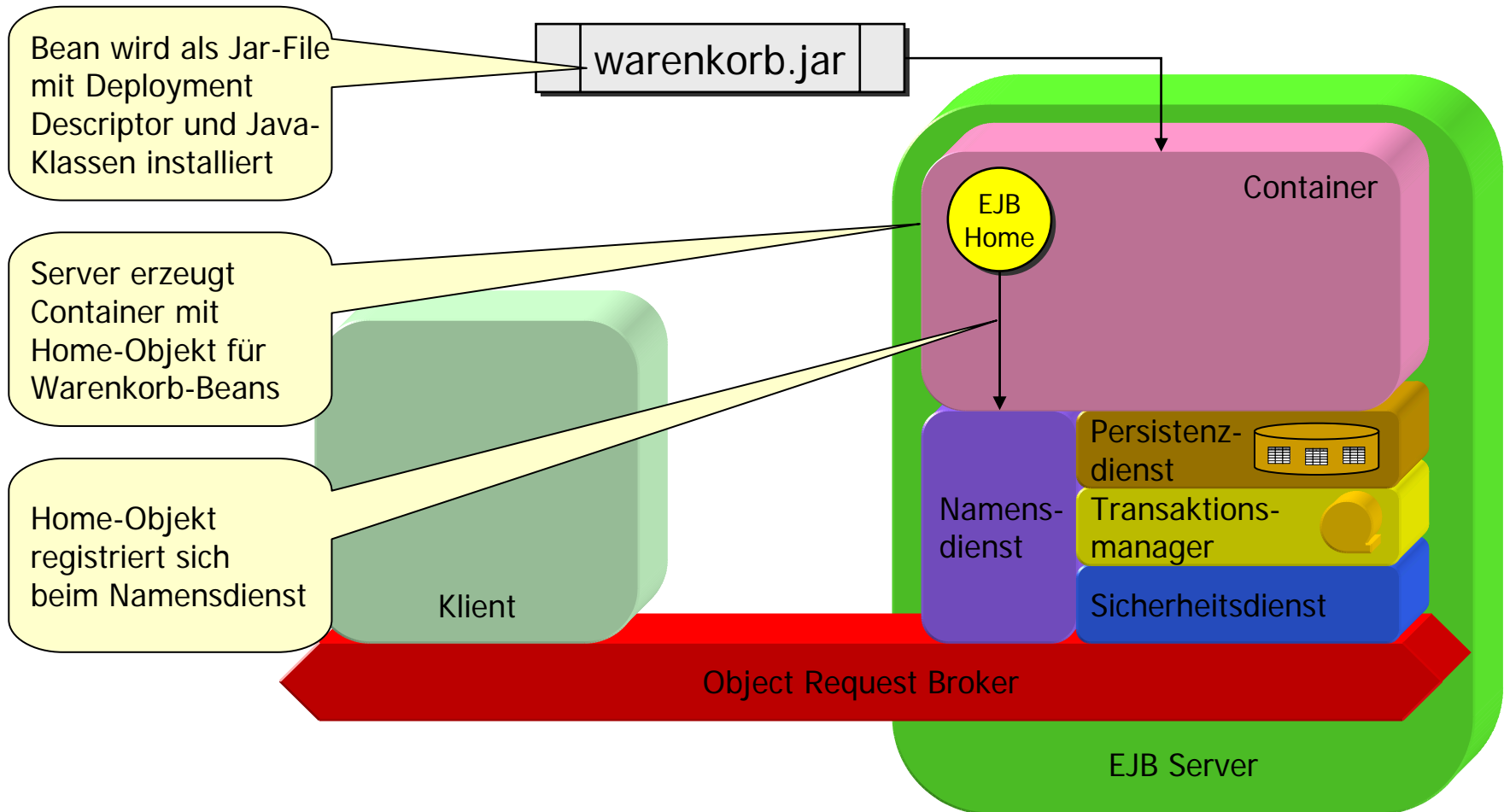
# Warenkorb-Bean: Deployment Descriptor (Forts.)

```
<assembly-descriptor>
  <security-role>
    <description>Benutzer mit Vollzugriff</description>
    <role-name>Vollzugriff</role-name>
  </security-role>
  <method-permission>
    <role-name>Vollzugriff</role-name>
    <method>
      <ejb-name>Warenkorb</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
  <container-transaction>
    <method>
      <ejb-name>Warenkorb</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Supported</trans-attribute>
  </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

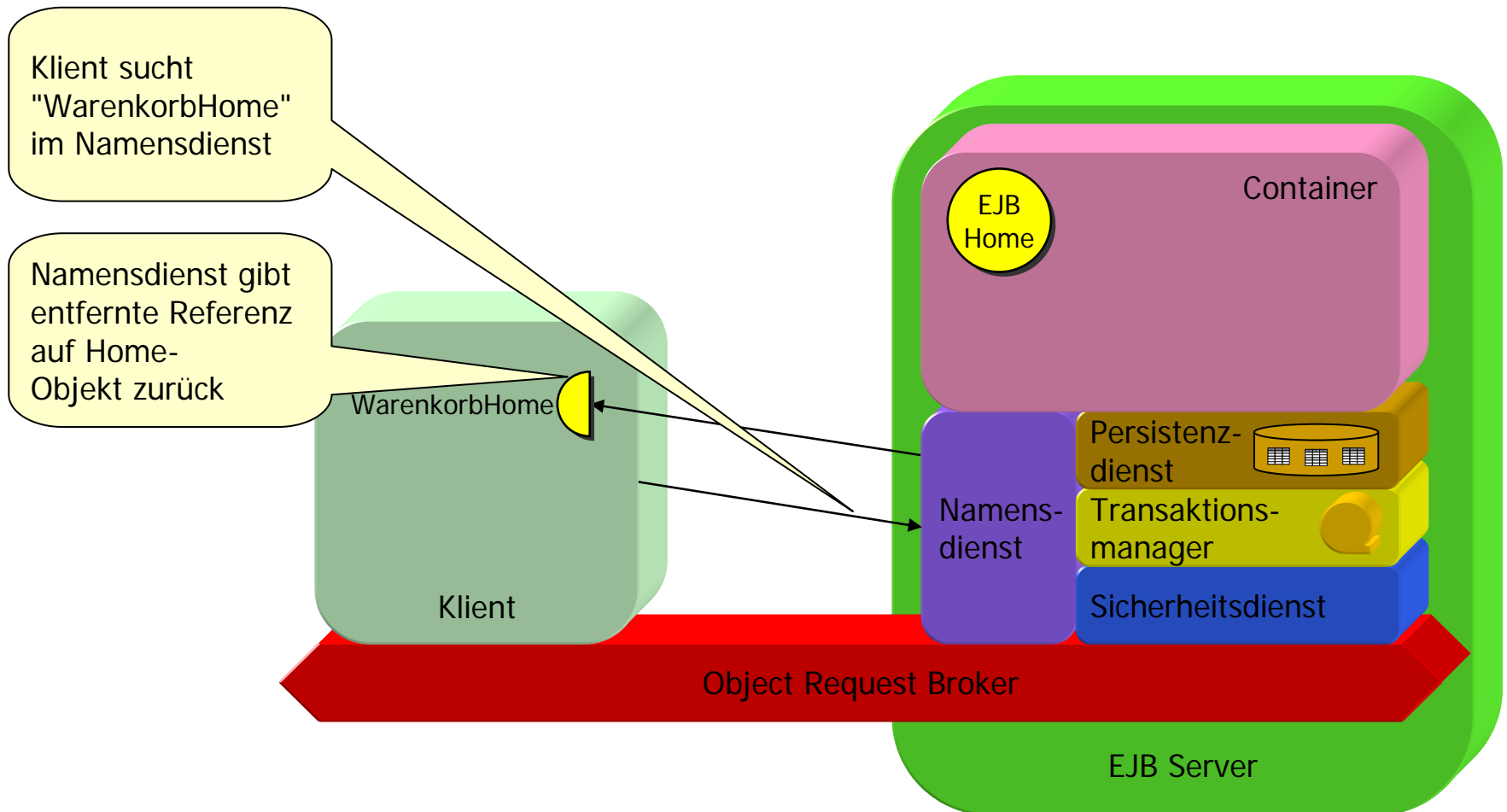
Sicherheit

Transaktionales Verhalten

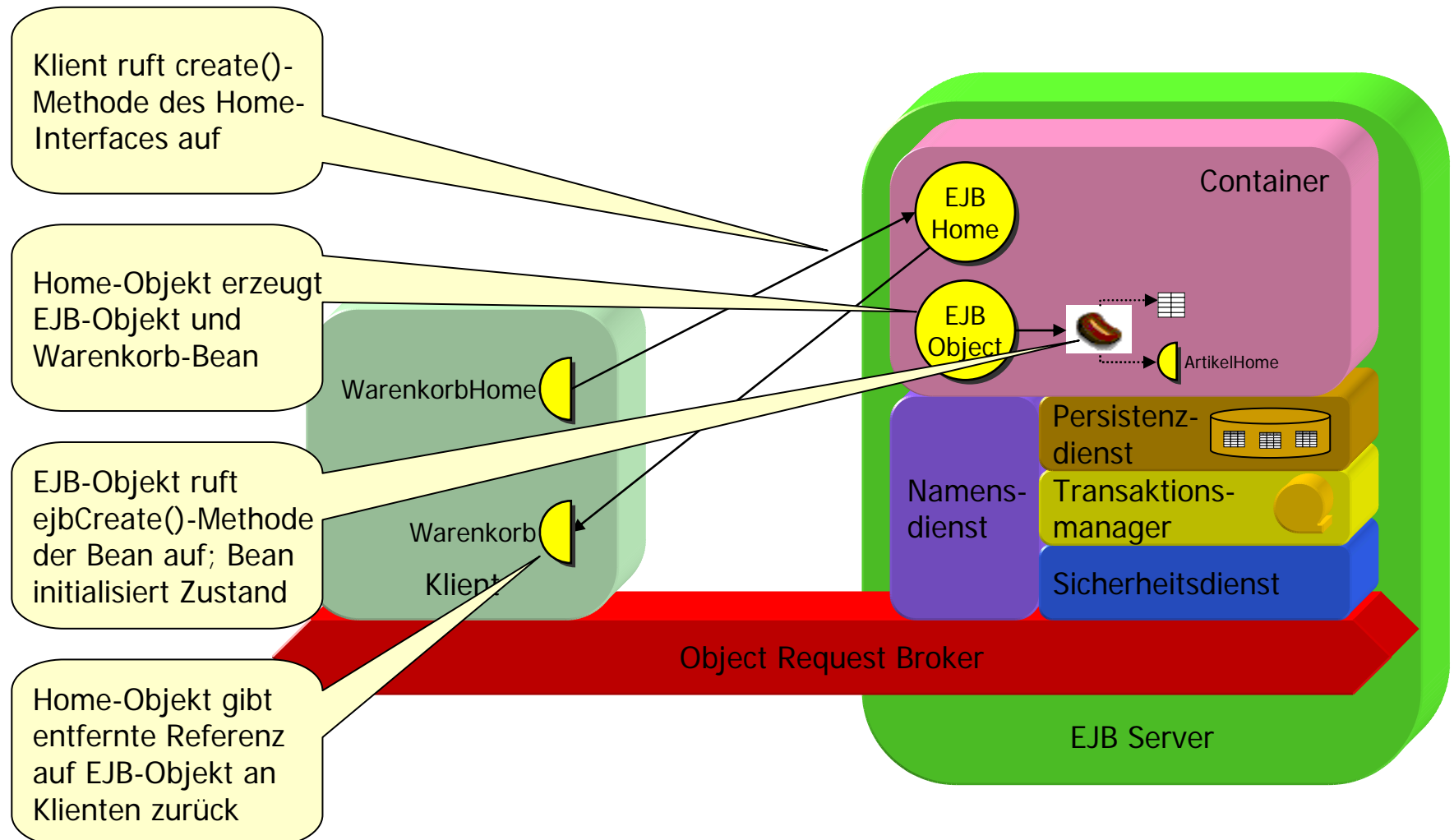
# Lebenszyklus einer Stateful Sess. Bean: Installation



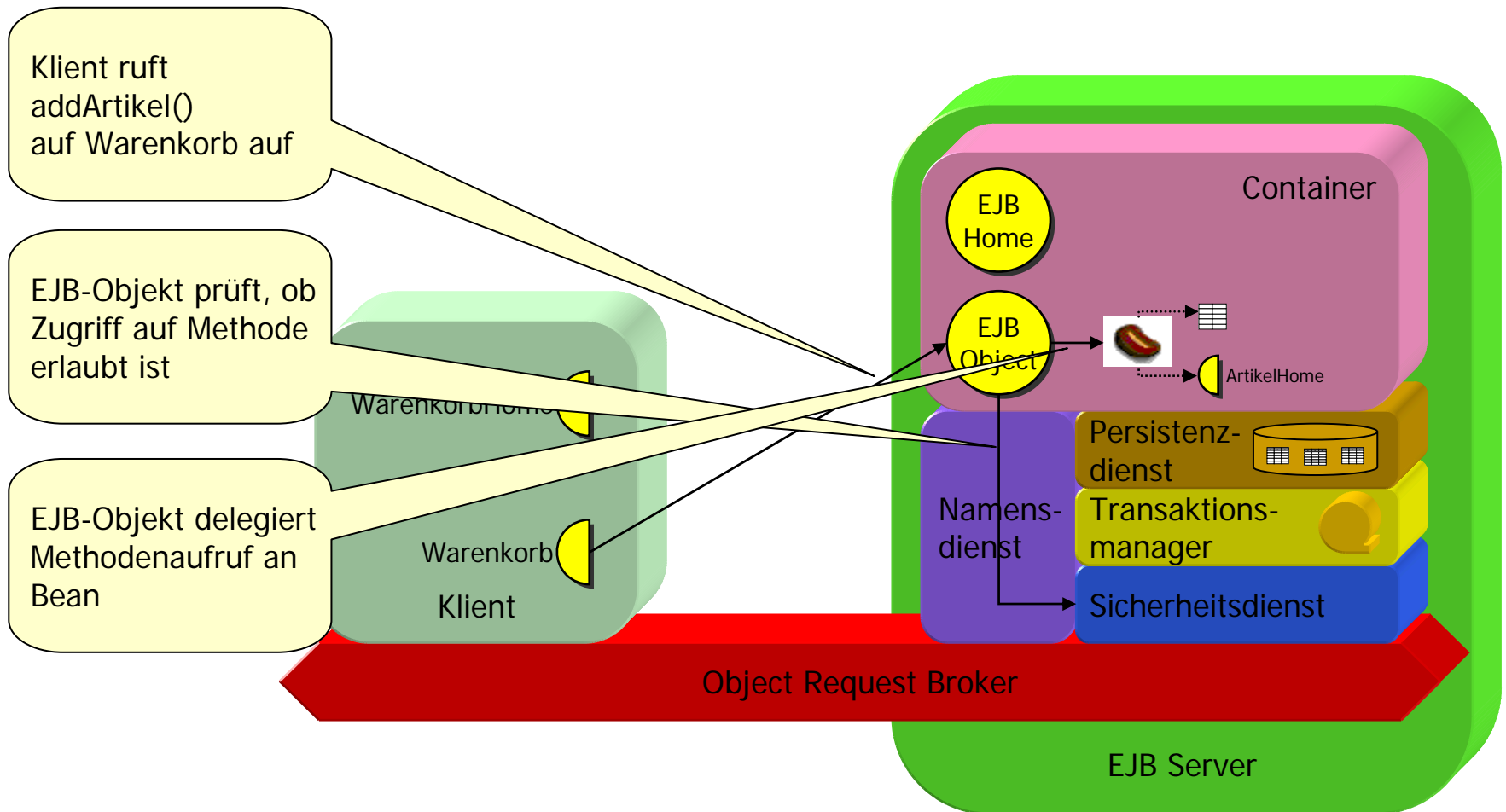
# Lebenszyklus einer Stateful Sess. Bean: Erzeugen einer Bean



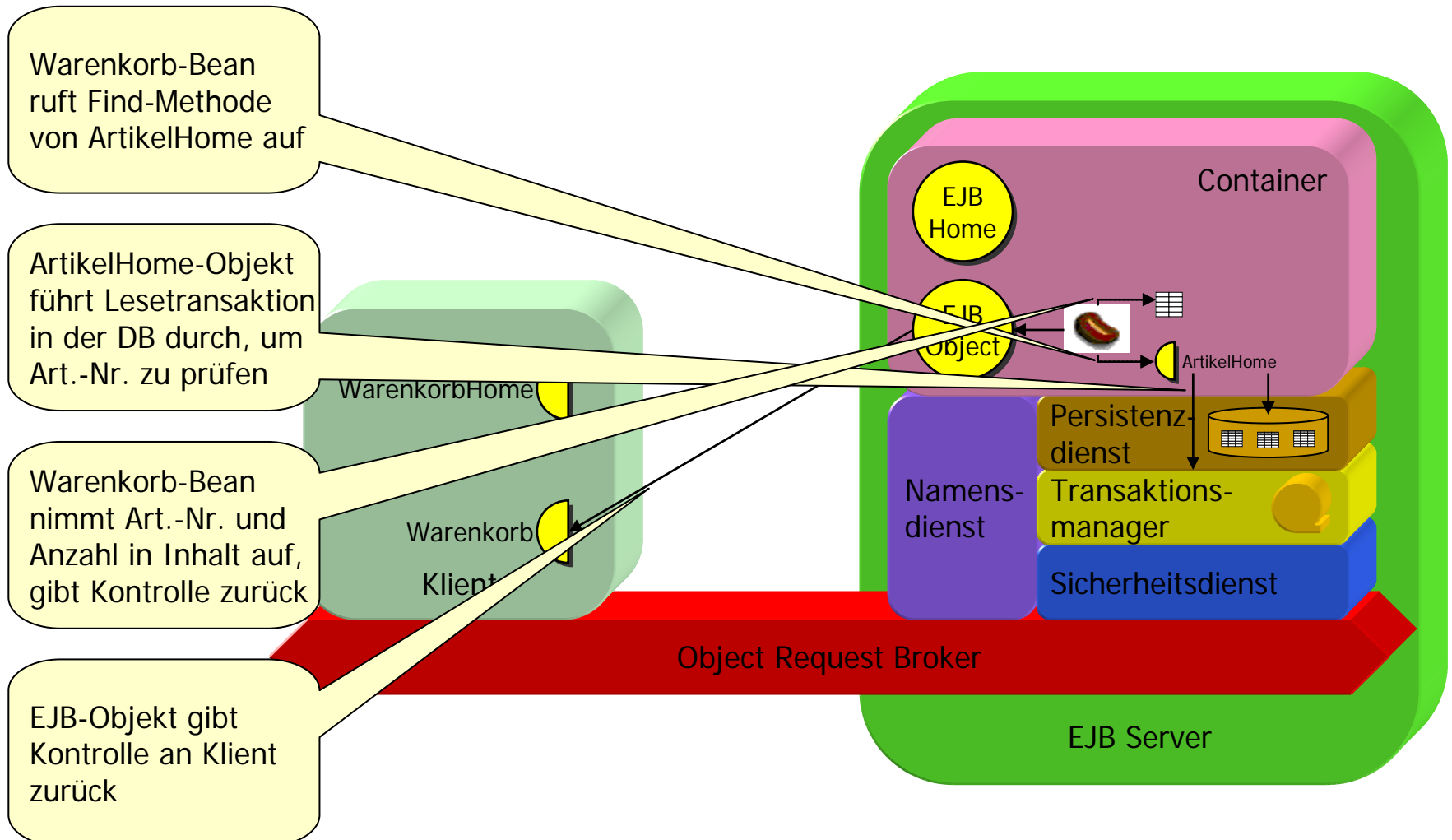
# Lebenszyklus einer Stateful Sess. Bean: Erzeugen einer Bean (Forts.)



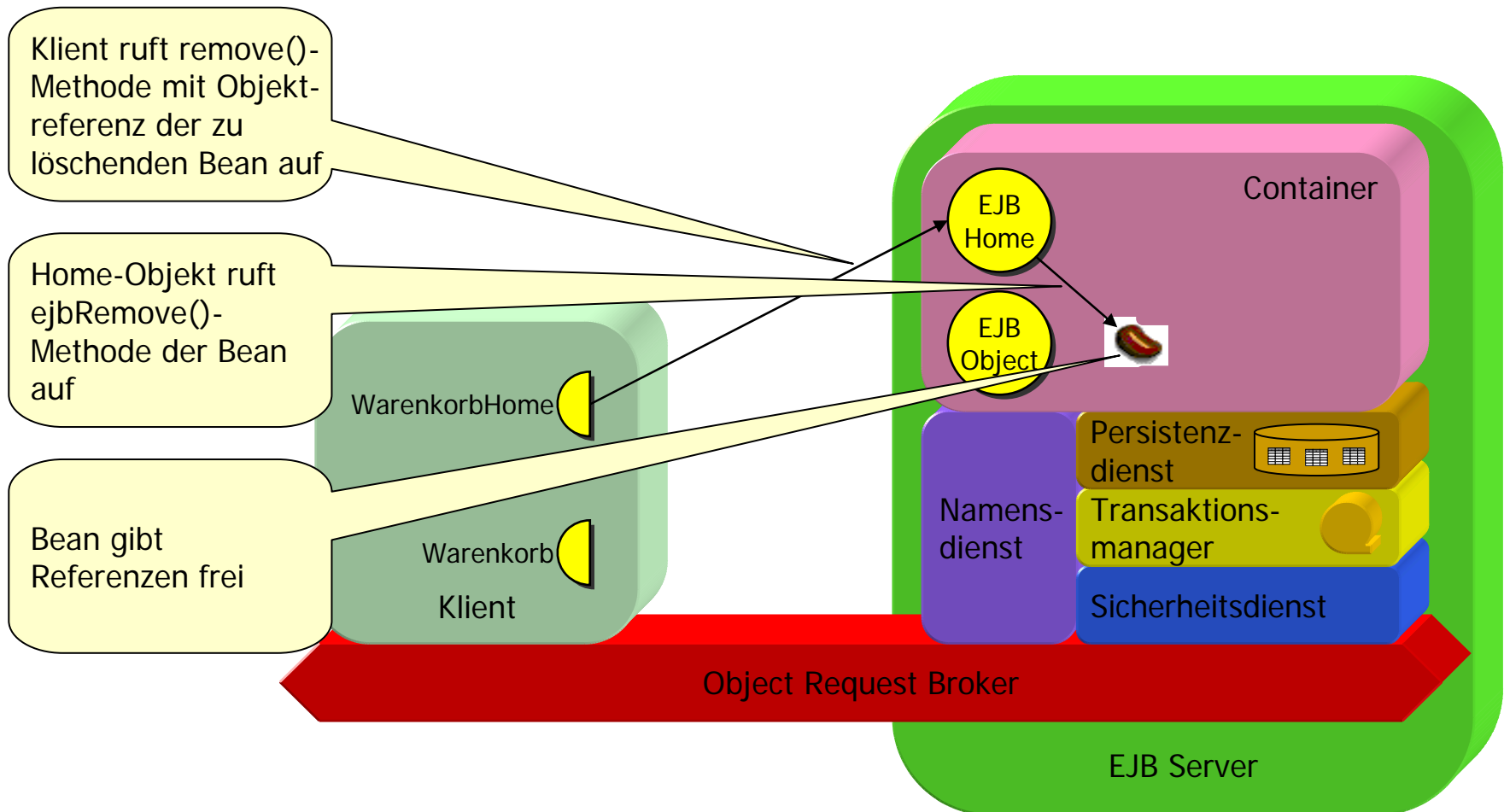
# Lebenszyklus einer Stateful Sess. Bean: Aufruf einer Geschäftsmethode



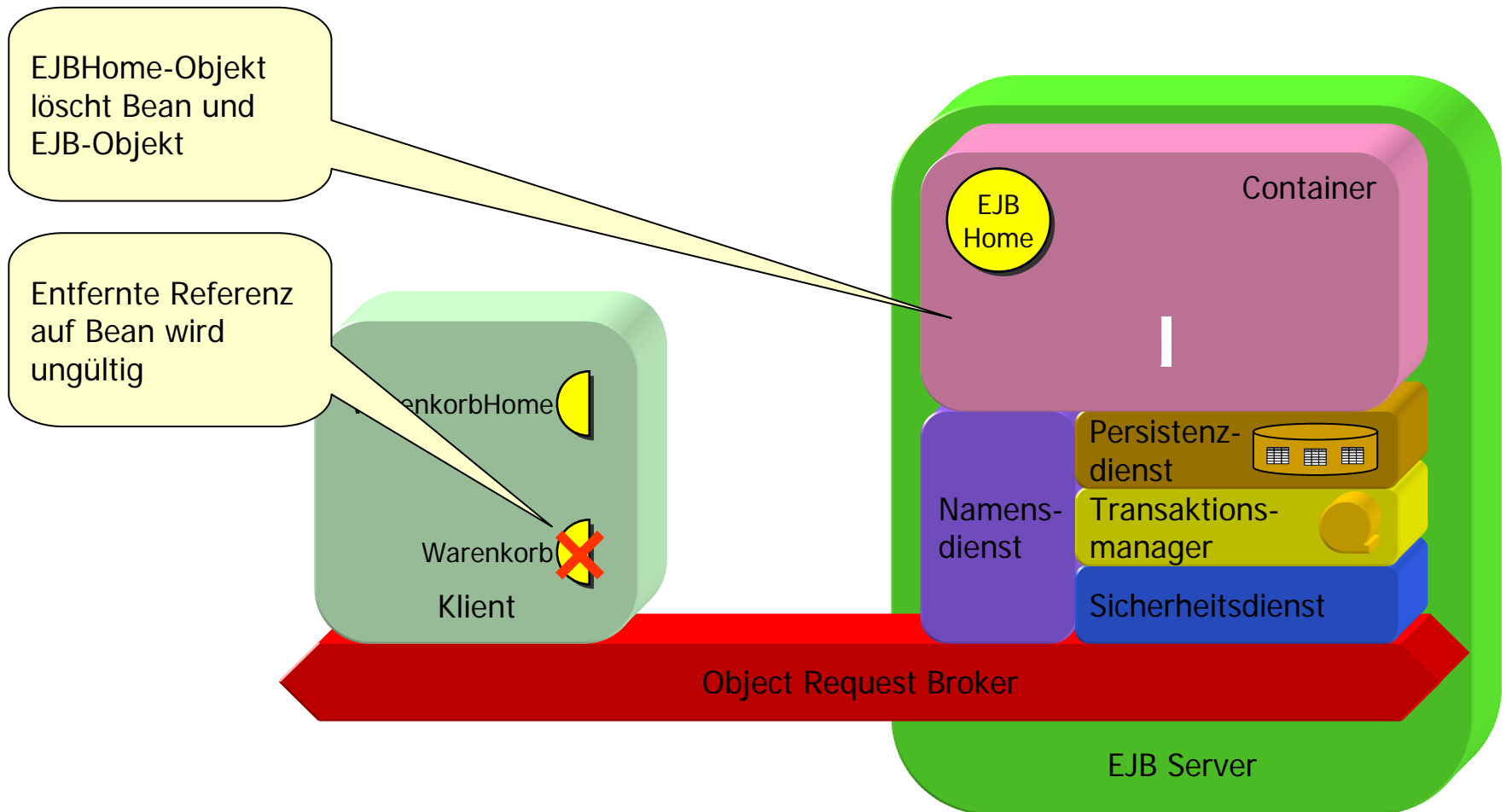
# Lebenszyklus einer Stateful Sess. Bean: Aufruf einer Geschäftsmethode (Forts.)



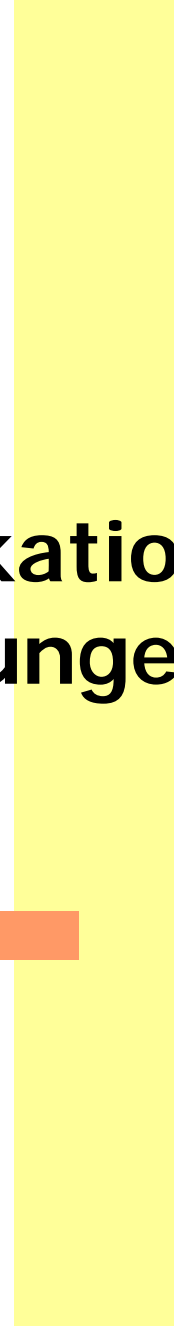
# Lebenszyklus einer Stateful Sess. Bean: Zerstören einer Bean



# Lebenszyklus einer Stateful Sess. Bean: Zerstören einer Bean (Forts.)

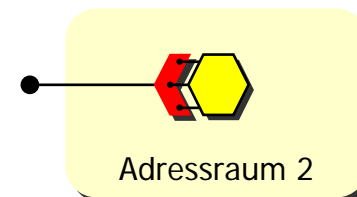
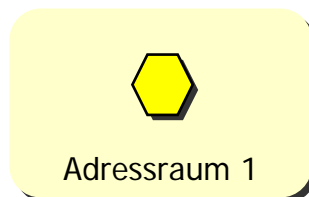


# Kommunikation in verteilten Anwendungen



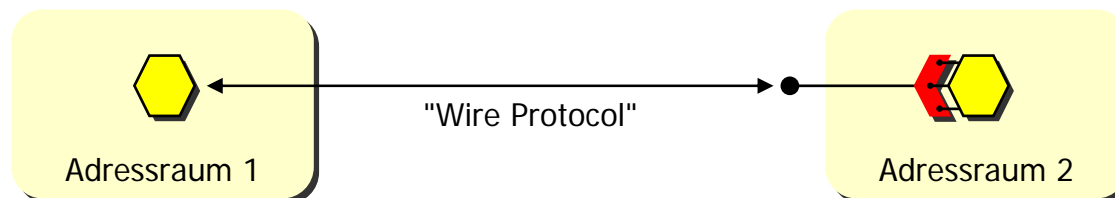
# Lokale und entfernte Objekte

- *Lokales Objekt*: Objekt im eigenen Adressraum
  - Direkt adressierbar
  - Gleiche Datenformate und Aufrufkonventionen
  - Gleiche Fehlerdomäne
- *Entferntes Objekt*: Objekt in einem fremden Adressraum, das (evt. über einen Adapter) Methodenaufrufe aus dem Netzwerk entgegennehmen kann
  - Adressierbar über Netzwerkadresse + fremder Identifikator
  - Unterschiedliche Datenformate und Aufrufkonventionen
  - Andere Fehlerdomäne



# Entfernte Methodenaufrufe

- *Entfernter Methodenaufruf* = Aufruf einer Methode auf einem entfernten Objekt
- Voraussetzungen:
  - Aufrufer kennt Schnittstelle, Netzwerkadresse und Identifikator des entfernten Objekts
  - Aufrufer und entferntes Objekt haben sich auf ein gemeinsames Datenformat und Protokoll für die Übermittlung von Methodenaufrufen und -resultaten geeinigt (sog. *Wire Protocol*)



# Ablauf eines entfernten Methodenaufrufs

- Ggf. Herstellung einer Netzwerkverbindung mit dem entfernten Objekt
- Verpacken von Aufrufparametern in einen Bytestrom gemäß Wire Protocol (sog. *Marshalling*)
- Übermittlung von Methodennamen und verpackten Parametern an das Empfängerobjekt
- Auspacken der Parameter auf der Empfängerseite (sog. *De-Marshalling*)
- Ausführen der Methode
- Ggf. Verpacken von Ergebnis oder Ausnahmebedingung
- Rückübermittlung des verpackten Resultats an Aufrufer
- Ggf. Beenden der Netzwerkverbindung

# Stub, Skeleton und Servant

