



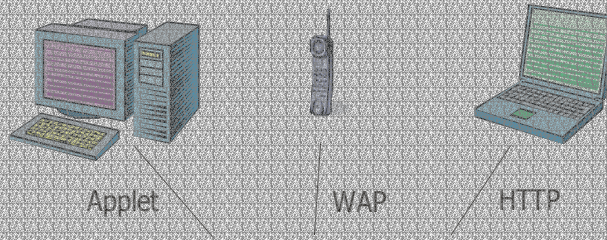
**Forschungszentrum
Informatik**



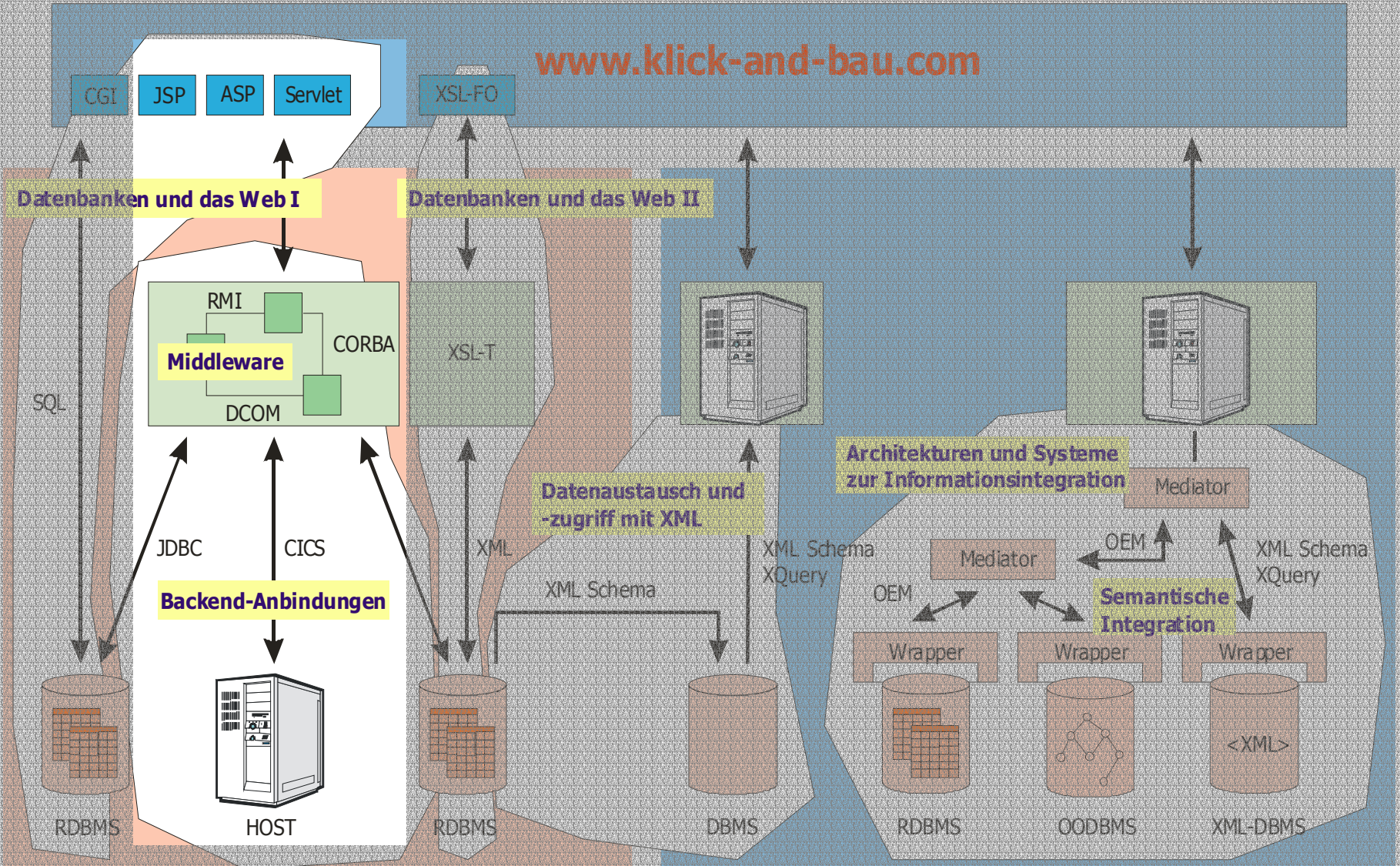
**Universität
Karlsruhe (TH)**

Mehrschichtenarchitekturen und Komponentenumgebungen

P. Tomczyk



www.klick-and-bau.com



Datenbanken und das Web I

Datenbanken und das Web II

Middleware

Backend-Anbindungen

Datenaustausch und -zugriff mit XML

Architekturen und Systeme zur Informationsintegration

Semantische Integration

RDBMS

HOST

RDBMS

DBMS

RDBMS

OODBMS

XML-DBMS

Mediator

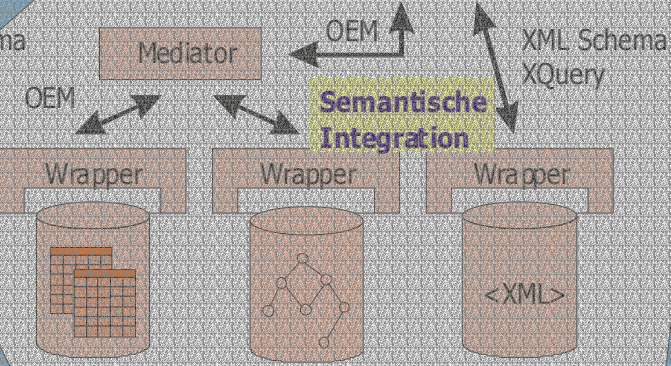
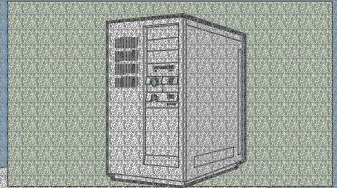
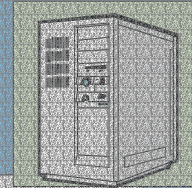
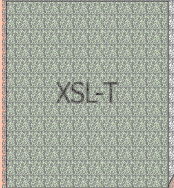
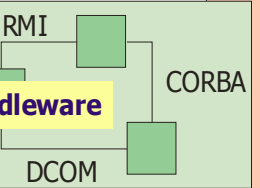
Mediator

Mediator

Wrapper

Wrapper

Wrapper

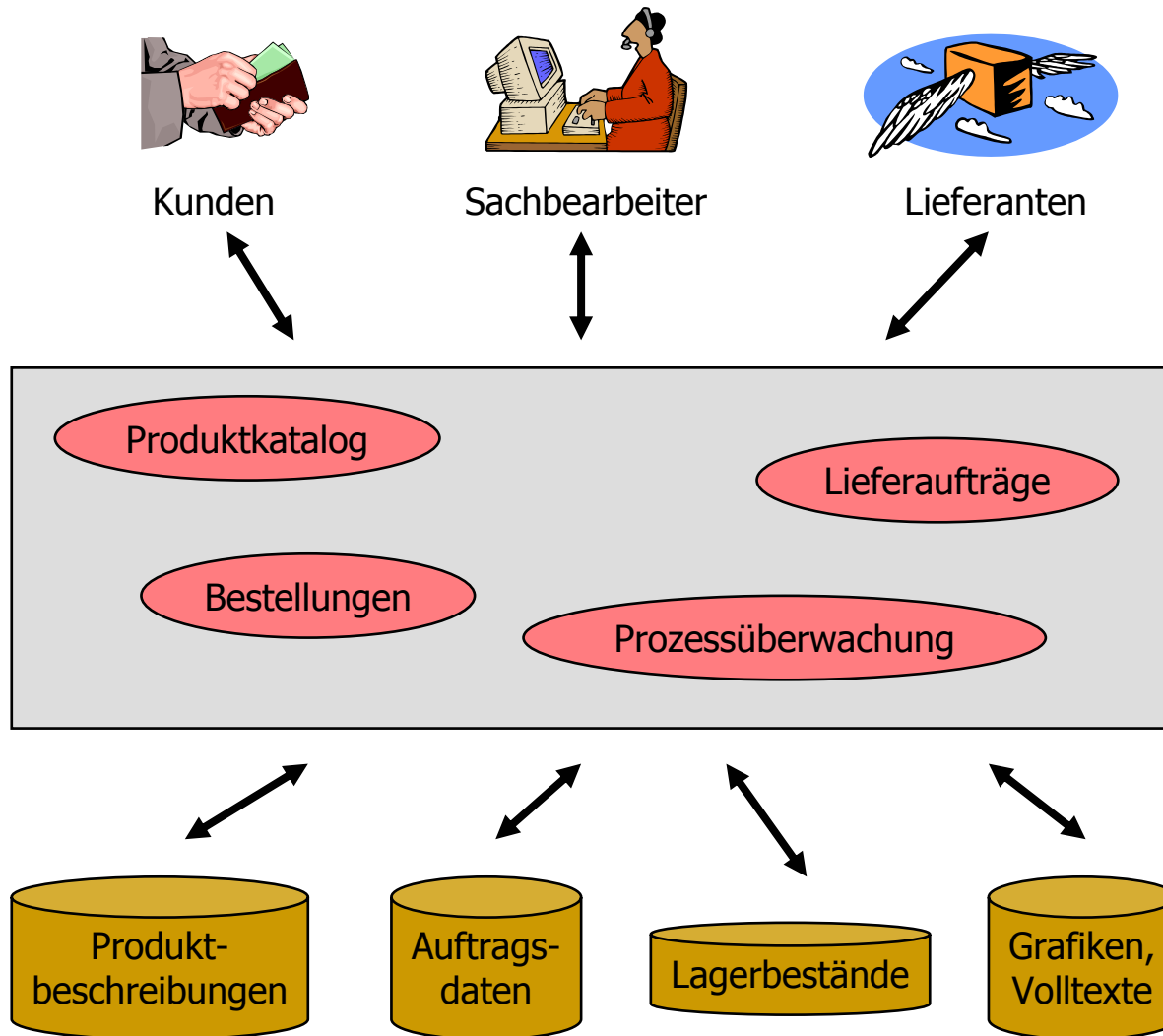


Programm heute



- Die Herausforderung: Systemintegration
- Mehrschichtenarchitekturen
- Komponenten und Komponentenumgebungen
- Konkretes Beispiel: Enterprise JavaBeans

Die Herausforderung: Systemintegration



Integrationsarchitekturen: Randbedingungen

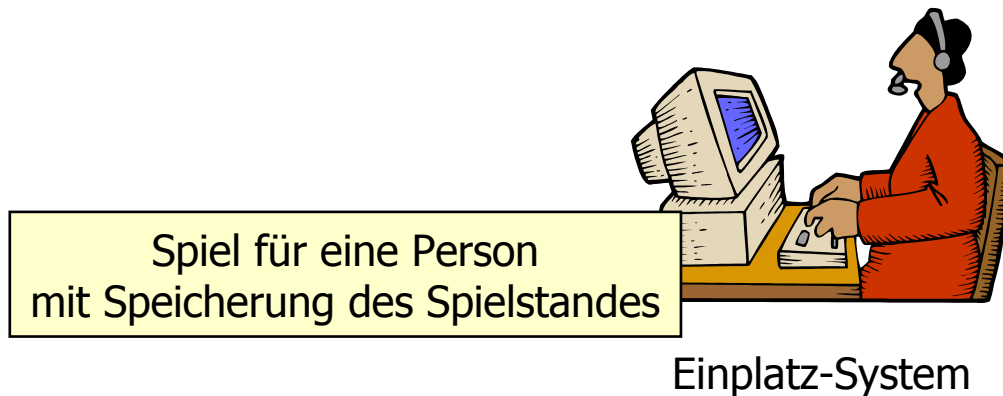
- Viele und unterschiedliche Systemkomponenten
 - ✍ DBMS, ERP-Systeme (z.B. SAP R/3), TP-Monitore, ...
 - ✍ Unterschiedlichste Plattformen, Sprachen u. Protokolle
- Hohe Skalierbarkeit erforderlich
 - ✍ Nutzung durch Tausende bis Zehntausende Mitarbeiter
 - ✍ Millionen bis Milliarden Seitenabrufe pro Tag
- Hohe Installations- und Wartungsfreundlichkeit erforderlich
 - ✍ Pro installierter Programmkopie ca. 2 Wartungen / Jahr
- Hohe Flexibilität erforderlich
 - ✍ Operationale Datenbestände überdauern ca. 20 Jahre
 - ✍ Geschäftsprozesse überdauern ca. 2 Jahre
 - ✍ Präsentationslogik überdauert ca. 6 Monate

Mehrschichtenarchitekturen



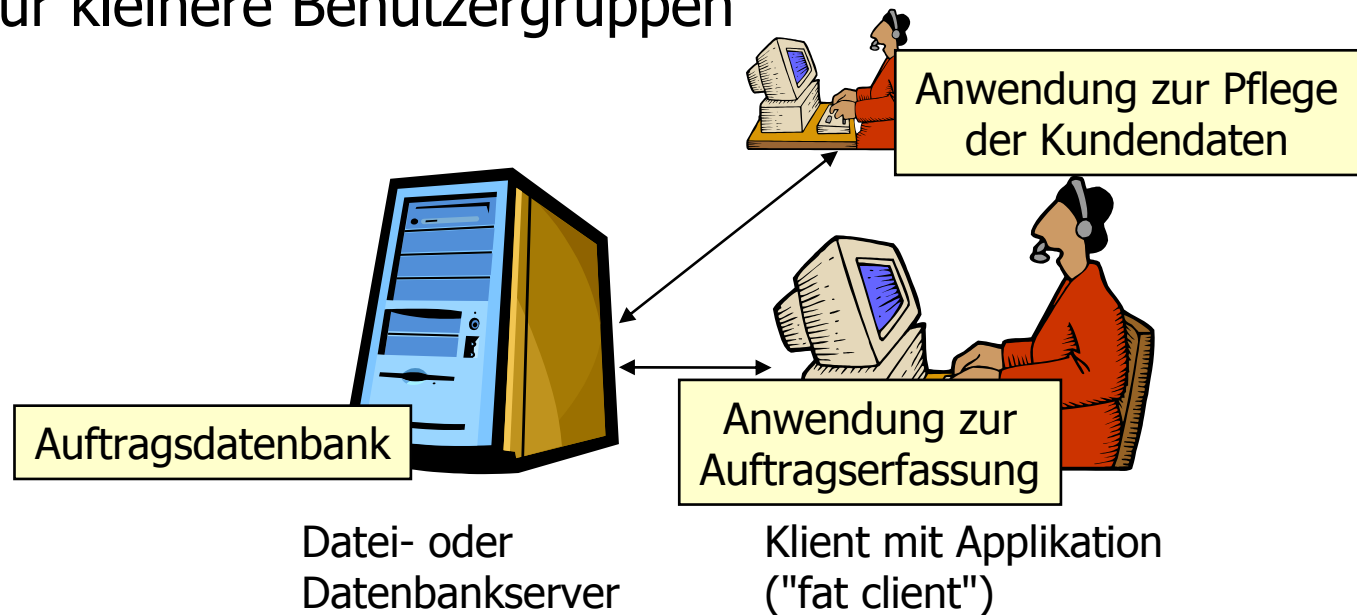
Ein-Schichten-Architektur

- Monolithisches System: integrierte Datenhaltung, Applikationslogik und Präsentation
- Beispiele: Textverarbeitung, Programmierumgebungen, Spiele, ...
- Geeignet für einzelne Benutzer



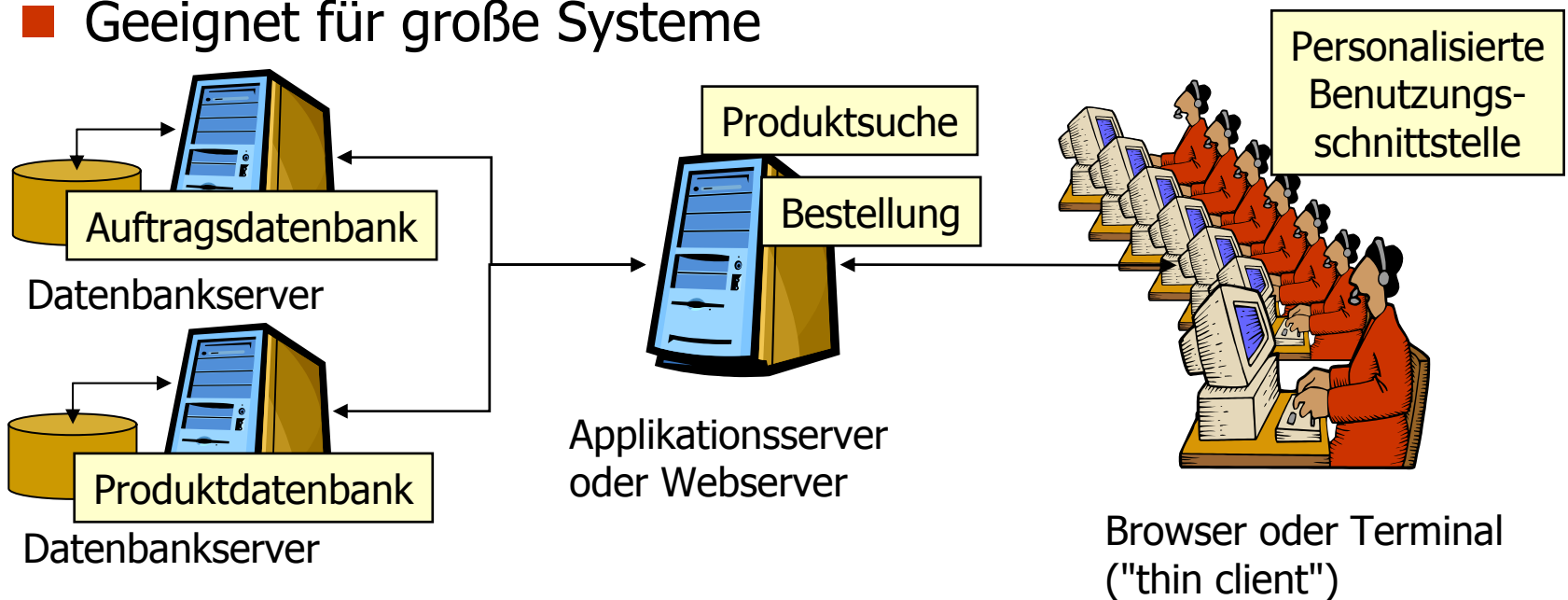
Zwei-Schichten-Architektur

- Datenhaltung erfolgt getrennt von Applikationen
- Beispiele: Vernetzter PC mit Datei-Server, Warenwirtschafts-Anwendung mit unterliegendem DBMS
- Geeignet für kleinere Benutzergruppen



Drei-Schichten-Architektur

- Datenhaltung, Applikationslogik und Präsentation getrennt
- Beispiele: Webseiten mit DB-Anbindung, viele TP-Monitor-gestützte Systeme (Flugbuchung, Kontoverwaltung, ...)
- Geeignet für große Systeme

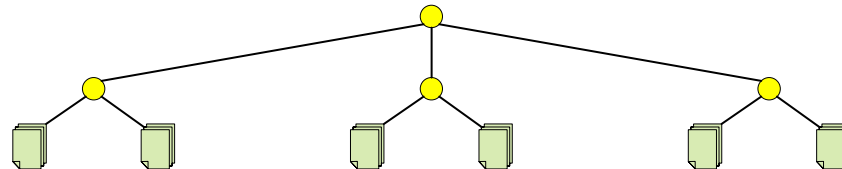


Realität: Viele Schichten

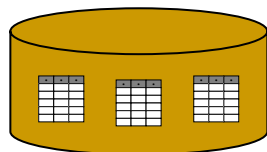
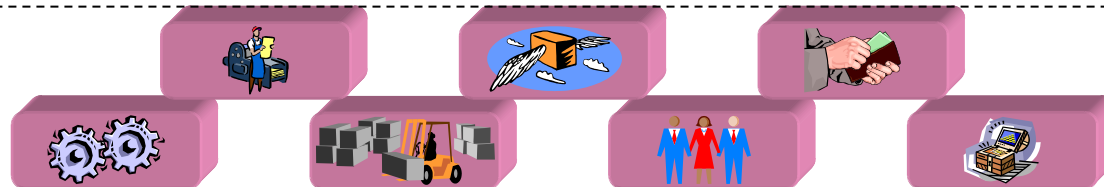
Klienten mit unterschiedlichen Fähigkeiten



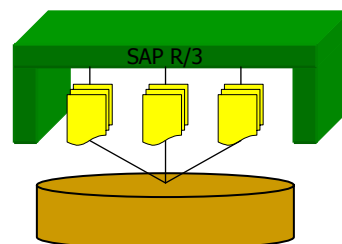
Webserver mit Skripting- und Personalisierungsschicht



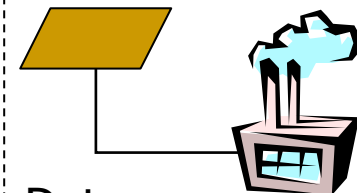
Aufeinander aufbauende Applikationen



DBMS



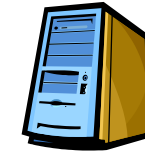
ERP-System



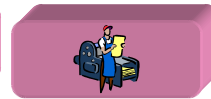
Daten von Zulieferfirma

Referenzmodell eines integrierten Informationssystems

Klienten
(Endnutzer oder
weitere Schichten)



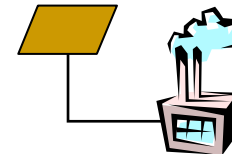
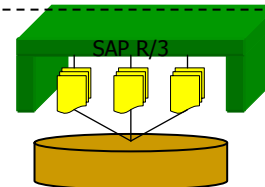
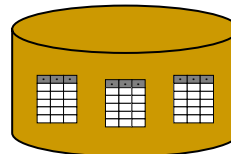
Geschäftsprozesse
(auf Basis der
Geschäftsobjekte)



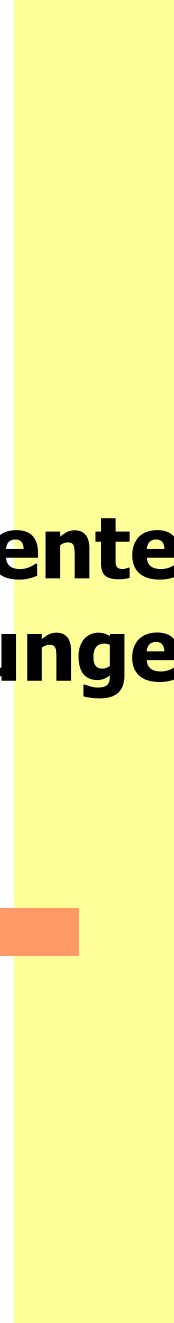
Geschäftsobjekte
(technisch und inhaltlich
integrierte Daten)



Rohdaten und -dienste
(nicht integriert)

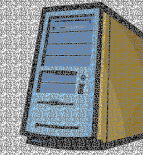


Komponenten und Komponentenumgebungen



Referenzmodell eines integrierten Informationssystems

Klienten
(Endnutzer oder
weitere Schichten)



Wiederverwendbare
Geschäftsprozess-
komponenten

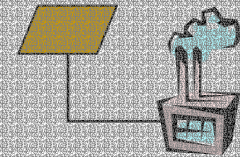
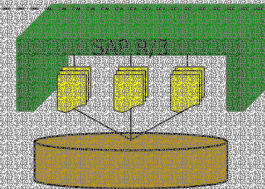
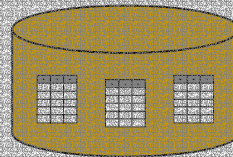


Komponenten-Umgebung

Wiederverwendbare
Geschäftsobjekt-
komponenten



Rohdaten und -dienste
(nicht integriert)



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)

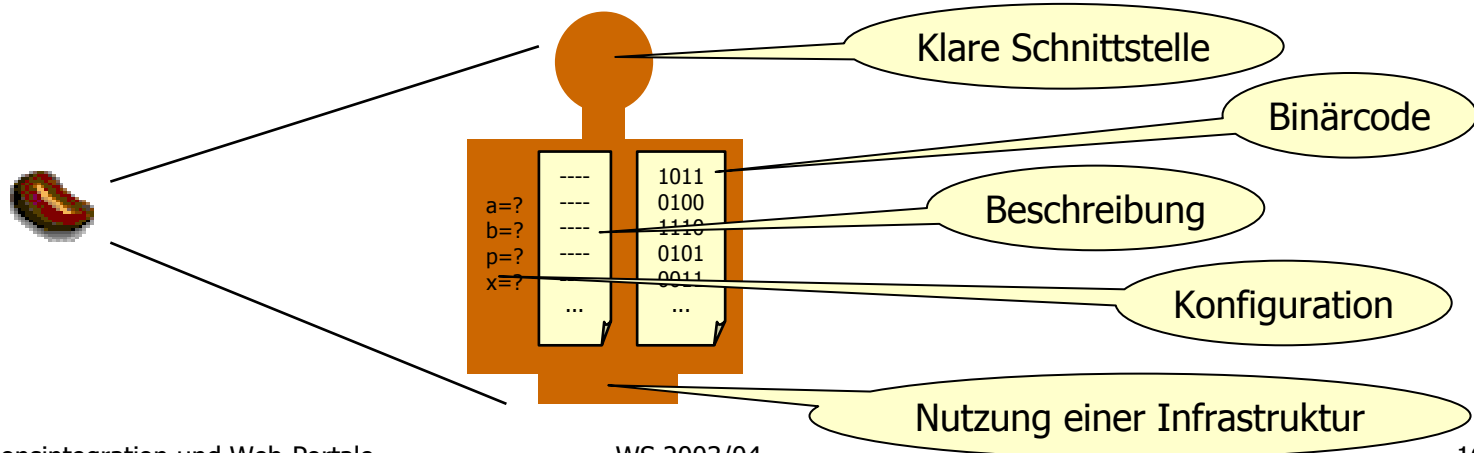
Zur Definition siehe auch C. Szyperski oder F. Griffel

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)
 - ⇒ optimale Granularität bis heute umstritten

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*.



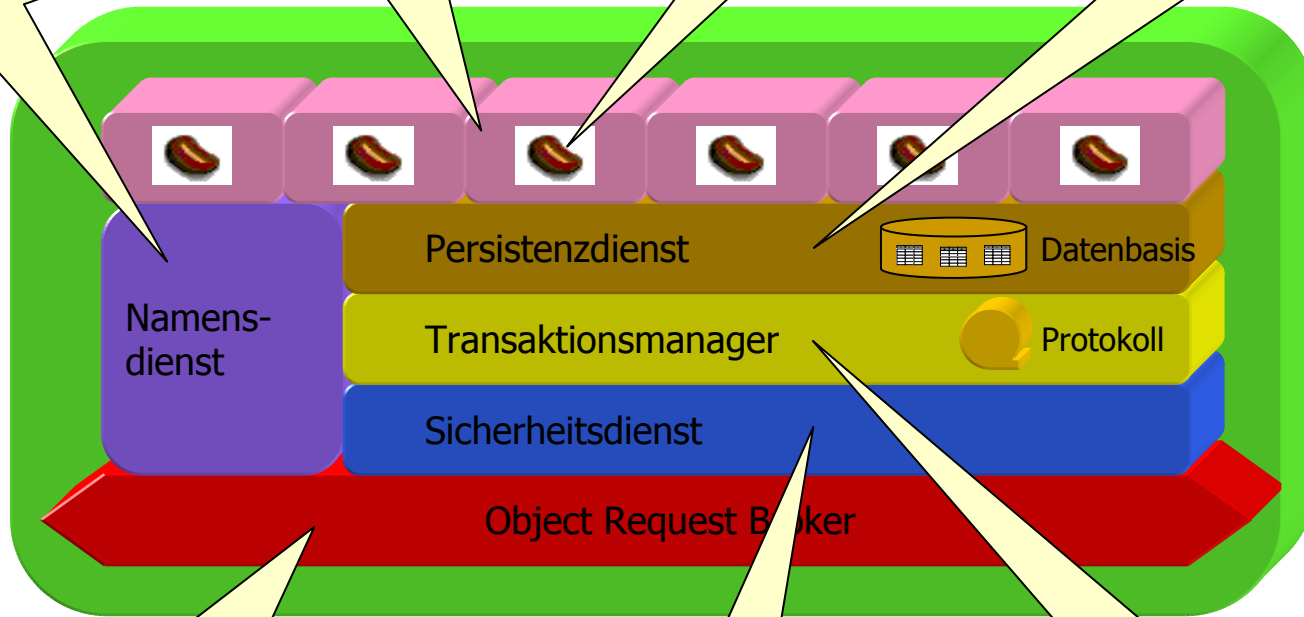
Komponentenumgebungen: Applikationsserver

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

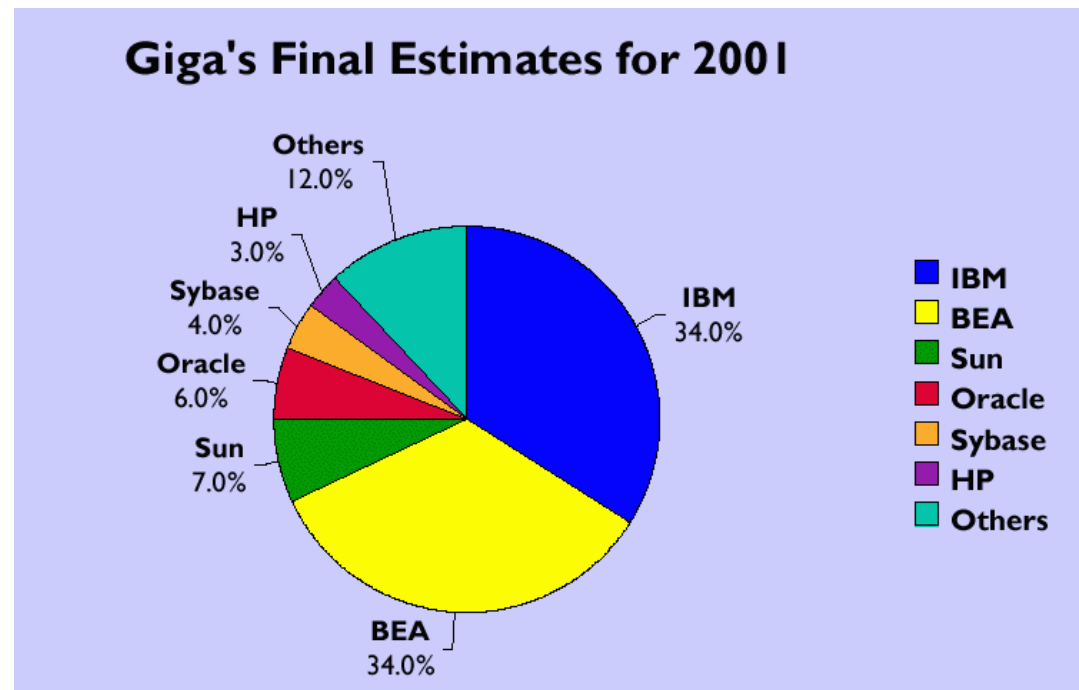
Applikationsserver: Beispiele

■ Kommerzielle Applikationsserver:

- ✍ BEA WebLogic, IBM WebSphere, Oracle9i Application Server, Sun ONE (iPlanet), Sybase EAServer, HP, Iona, Microsoft .NET, ...

■ Open Source:

- ✍ JBoss
- ✍ JOnAS
- ✍ ...



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

„Separation of Concerns“: Funktionale und technische Belange

- Komponenten sind für *funktionale* (anwendungsspezifische) Belange zuständig:
 - ✍ Welche Daten beschreiben einen Kunden?
 - ✍ Wie sieht der Bestellvorgang aus?
 - ✍ Wie wird eine Kreditkarte geprüft?
- Komponentenumgebungen sind für *technische* (anwendungsunabhängige) Belange zuständig:
 - ✍ Wie wird der Zustand einer Komponente persistent gemacht?
 - ✍ Wie findet man eine Komponente wieder?
 - ✍ Wie kommunizieren Komponenten miteinander?

Enterprise JavaBeans: Der offene Standard für Komponentenumgebungen



Der EJB-Standard

- Entwickelt von Sun Microsystems als Java-Pendant zum Microsoft Transaction Server
 - ✍ Erste Fassung EJB 1.0 (Dezember 1997)
 - ✍ „JSR Final Release“ ist EJB 2.0 (April 2001)
 - ✍ „JSR Proposed Final Draft 2“ ist EJB 2.1 (Juni 2003)
- Breite Akzeptanz in der Industrie
- Eng abgestimmt mit dem CORBA-Standard für verteilte Objektsysteme
- Referenzimplementierung von Sun, zahlreiche kommerzielle Implementierungen, einige Implementierungen frei verfügbar (praktisch alle genannten Applikationsserver bis auf Microsoft unterstützen EJB)

Wesentliche Inhalte des EJB-Standards

■ EJB spezifiziert

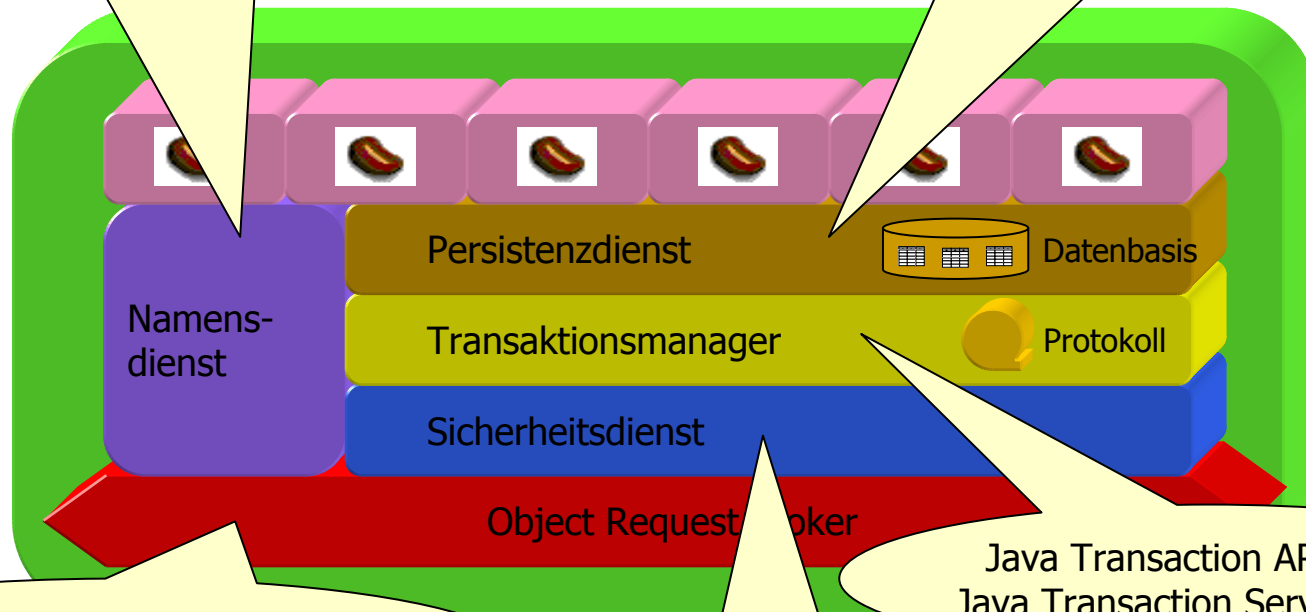
- ✍ die grundsätzlichen Arten von Komponenten
- ✍ die Klientensicht von Komponenten
- ✍ die Regeln für die Implementierung von Komponenten (Server)
- ✍ den Lebenszyklus von Komponenten und die Interaktion zwischen Komponente und Container
- ✍ die Beschreibung und Konfiguration von Komponenten
- ✍ Anfragesprache zum Zugriff auf persistente Komponenten



J2EE Infrastrukturdienste

Java Naming and Directory Interface (JNDI)

Java Database Connectivity (JDBC),
Container Managed Persistence (CMP)



Namens-
dienst

Persistenzdienst



Datenbasis

Transaktionsmanager



Protokoll

Sicherheitsdienst

Object Request Broker

Remote Method Invocation (RMI)
Java Messaging Service (JMS)

Java Transaction API (JTA)
Java Transaction Service (JTS)

Java Authentication
and Authorization Service
(JAAS)

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*:
Warenkorb füllen, Preis berechnen, 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(String id)
        throws RemoteException, FinderException;
    // remove() Methoden werden von EJBHome geerbt
}
```

Artikel-Bean: Primärschlüsselklasse

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

public class ArtikelPK implements java.io.Serializable {

    // Schl,sselfelder - hier muss eine Untermenge der Felder
    // der Beanklasse stehen.

    public String id;

    // Spezielle Hash- und Vergleichsmethoden, da die Schl,sselwerte
    // 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 (Forts.)

```
// Initialisierungsmethode - wird vom Container aufgerufen,  
// wenn Klient create()-Methode des Home Interface aufruft.  
// F,r 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: ÑINSERT INTO ARTIKEL VALUES (?,?,...)ì  
    return new ArtikelPK(id);  
}
```

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

```
public void ejbPostCreate(String id) {  
    // nichts zu tun  
}
```

Artikel-Bean: Beanklasse (Forts.)

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

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

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

```
// etc.
```

Artikel-Bean: Beanklasse (Forts.)

```
// Lebenszyklusmethoden - werden vom Container aufgerufen.

public void ejbLoad() {
    // Zustandsvariablen werden aus der Datenbank gelesen.
    // CMP: nichts zu tun.
    // BMP: ÑSELECT * FROM ARTIKEL WHERE Id = ?î
}

public void ejbStore() {
    // Zustandsvariablen werden in die Datenbank geschrieben.
    // CMP: nichts zu tun.
    // BMP: ÑUPDATE ARTIKEL SET Name = ? ... WHERE Id = ?î
}

public void ejbRemove() {
    // Bean und Datensatz werden gel^scht.
    // CMP: nichts zu tun.
    // BMP: ÑDELETE FROM ARTIKEL WHERE Id = ?î
}

// Einige weitere Lebenszyklusmethoden sind nicht gezeigt.
}
```

Artikel-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">
```

Artikel-Bean: Deployment Descriptor (Forts.)

```
<ejb-jar>
  <enterprise-beans>
    <entity>
      <description>Bean f,r 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 (Forts.)

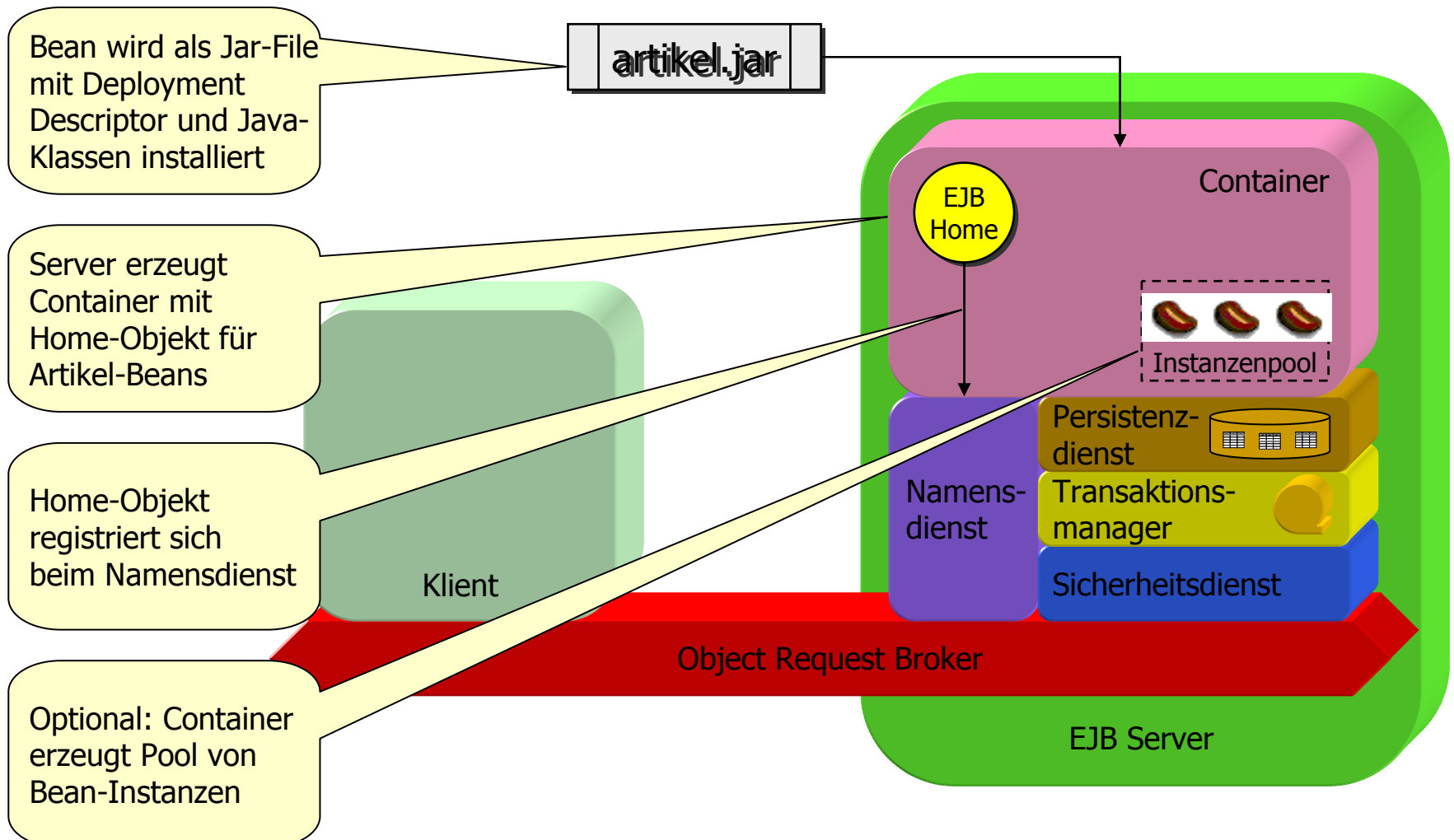
Sicherheit

```
<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>
```

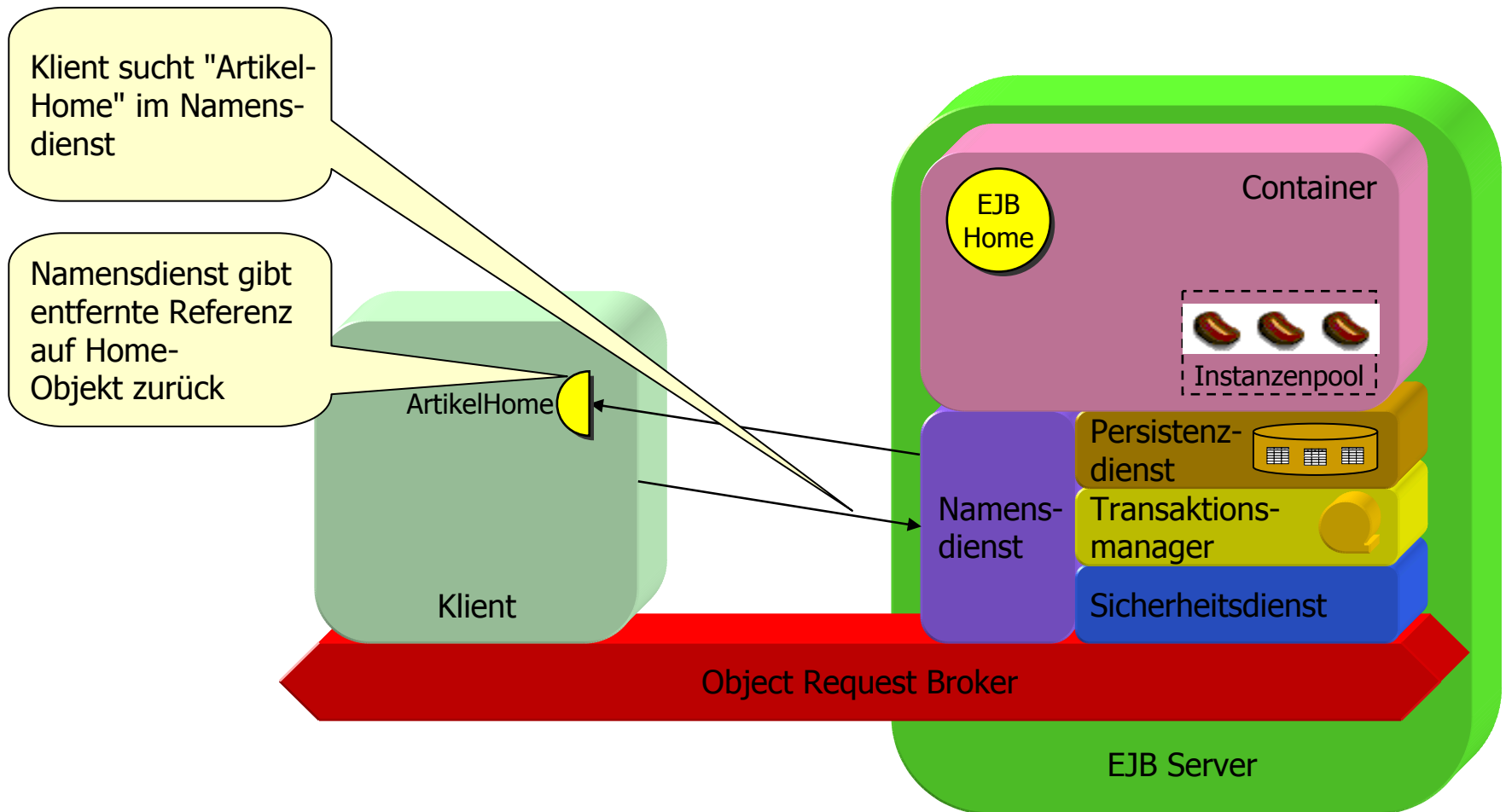
Transaktionales
Verhalten

```
<container-transaction>
  <method>
    <ejb-name>Artikel</ejb-name>
    <method-name>*</method-name>
  </method>
  <trans-attribute>Required</trans-attribute>
</container-transaction>
</assembly-descriptor>
</ejb-jar>
```

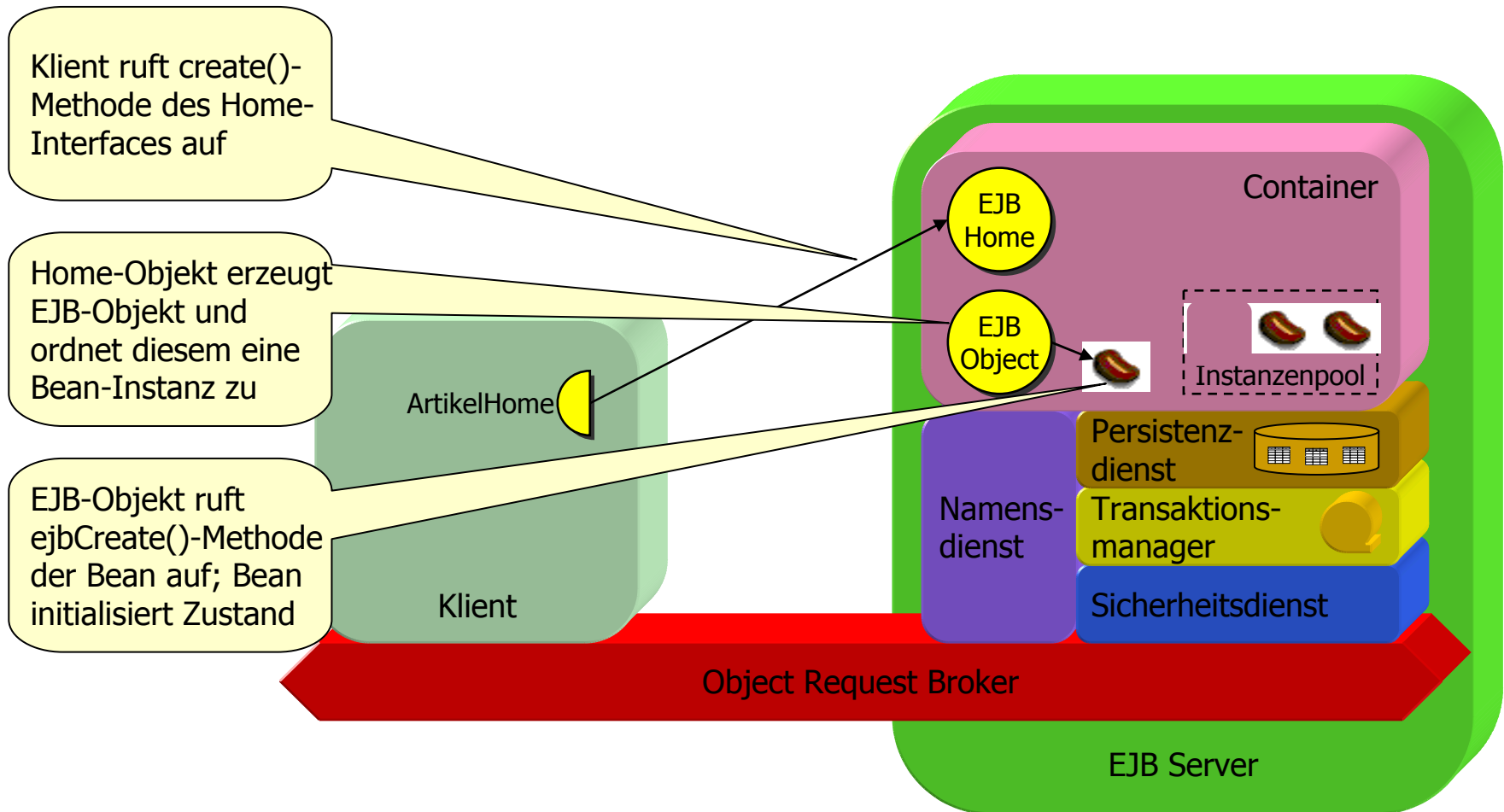
Lebenszyklus einer Entity Bean: Installation



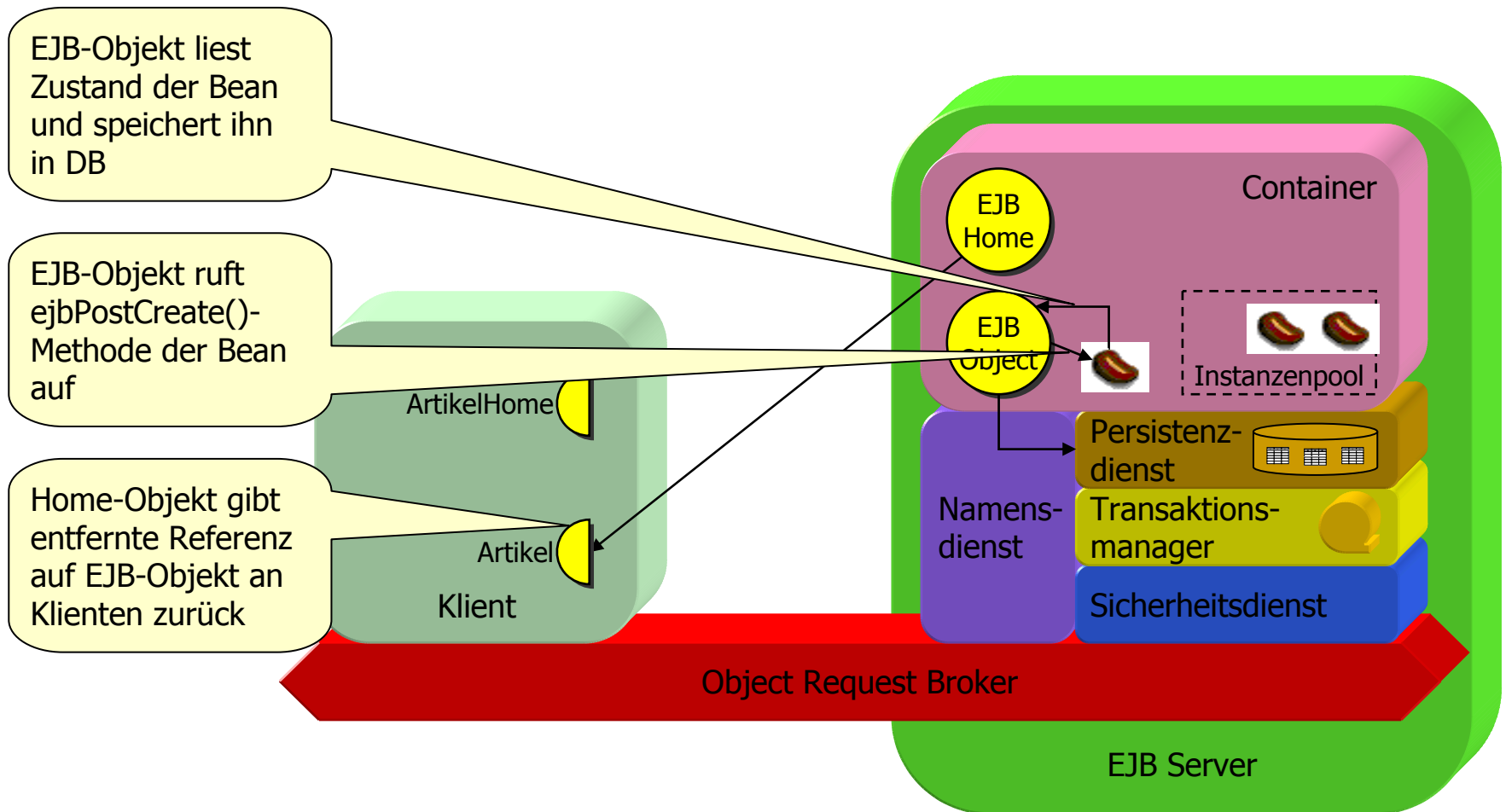
Lebenszyklus einer Entity Bean: Erzeugen einer Bean



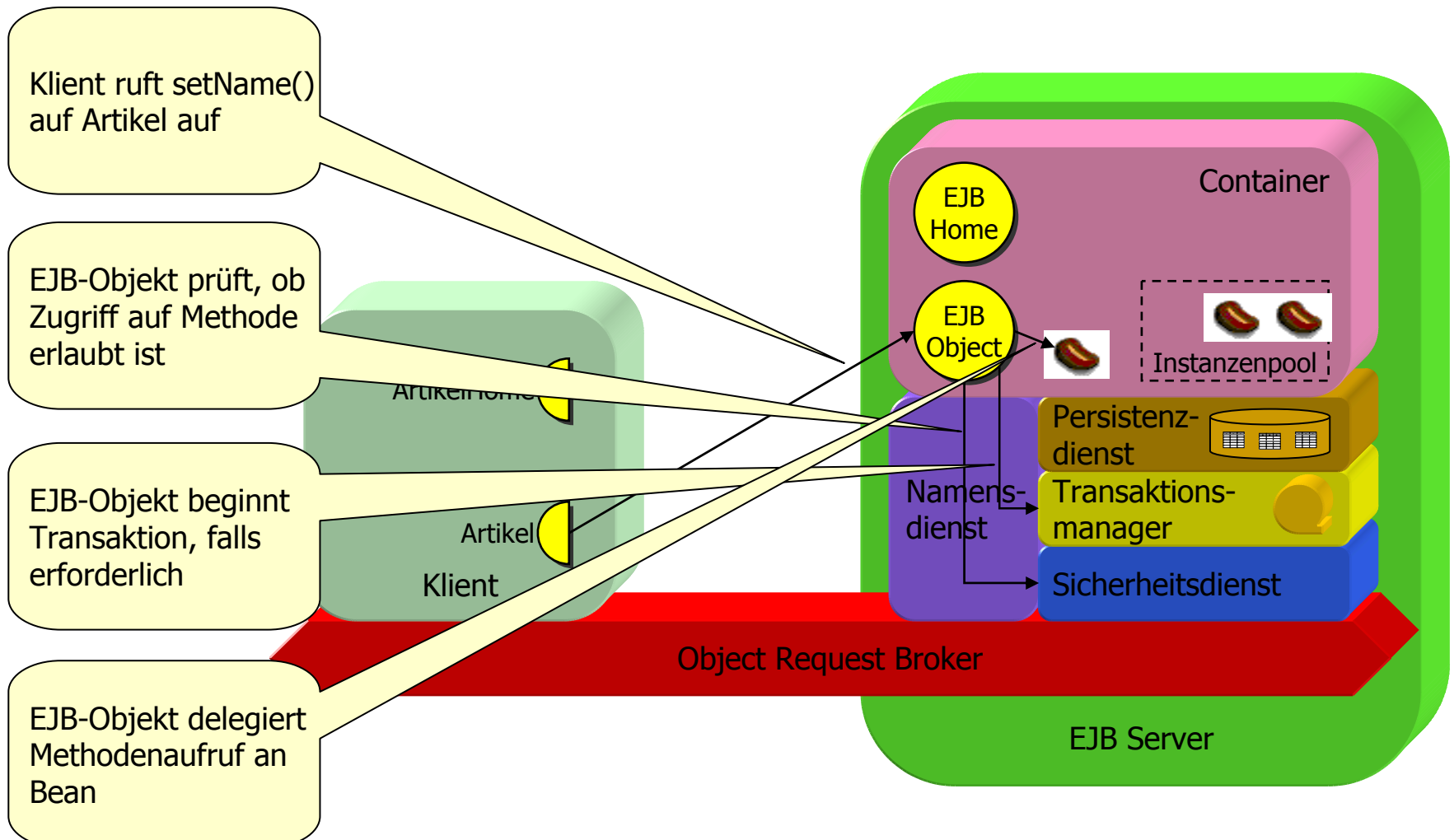
Lebenszyklus einer Entity Bean: Erzeugen einer Bean (Forts.)



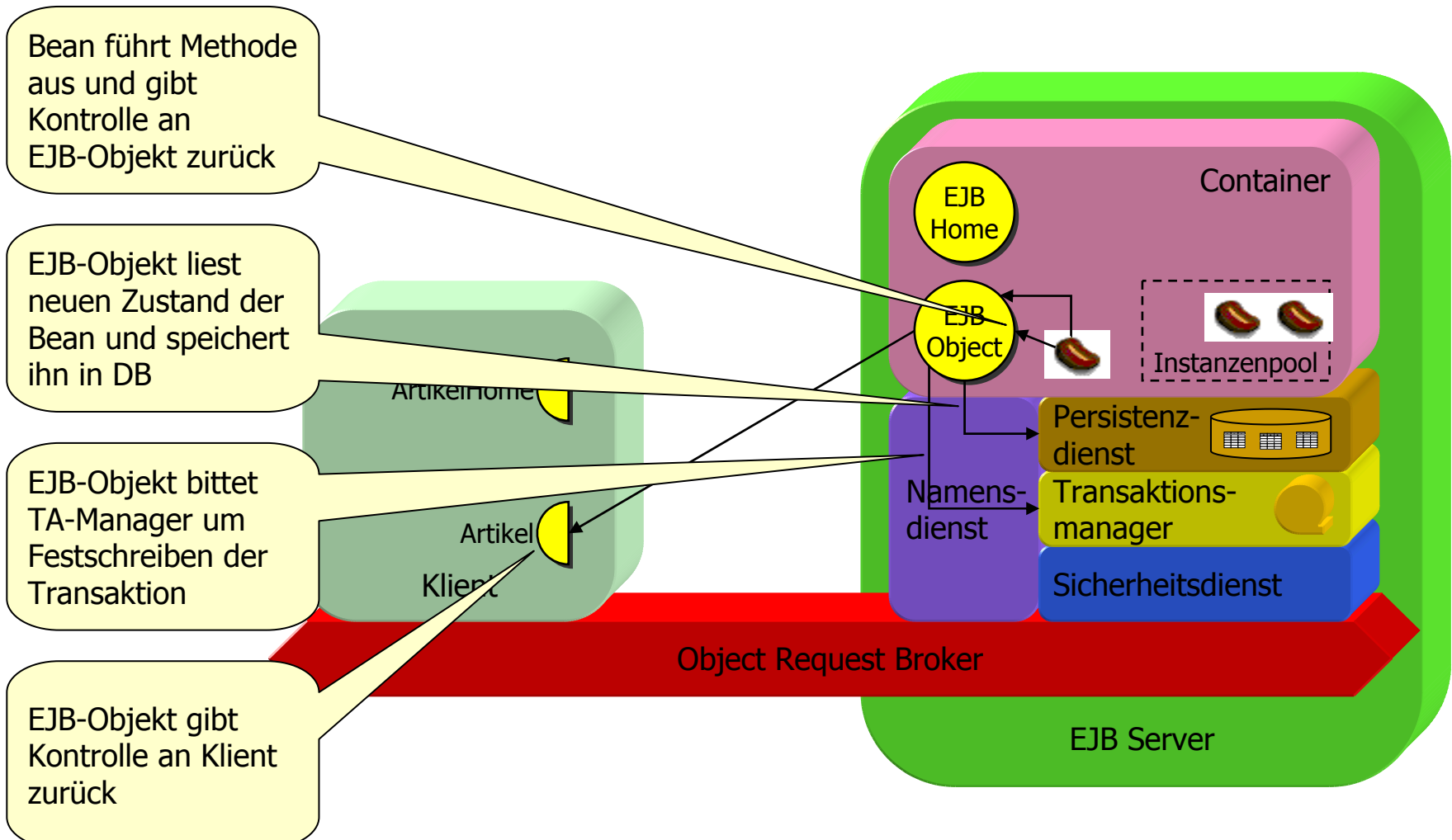
Lebenszyklus einer Entity Bean: Erzeugen einer Bean (Forts.)



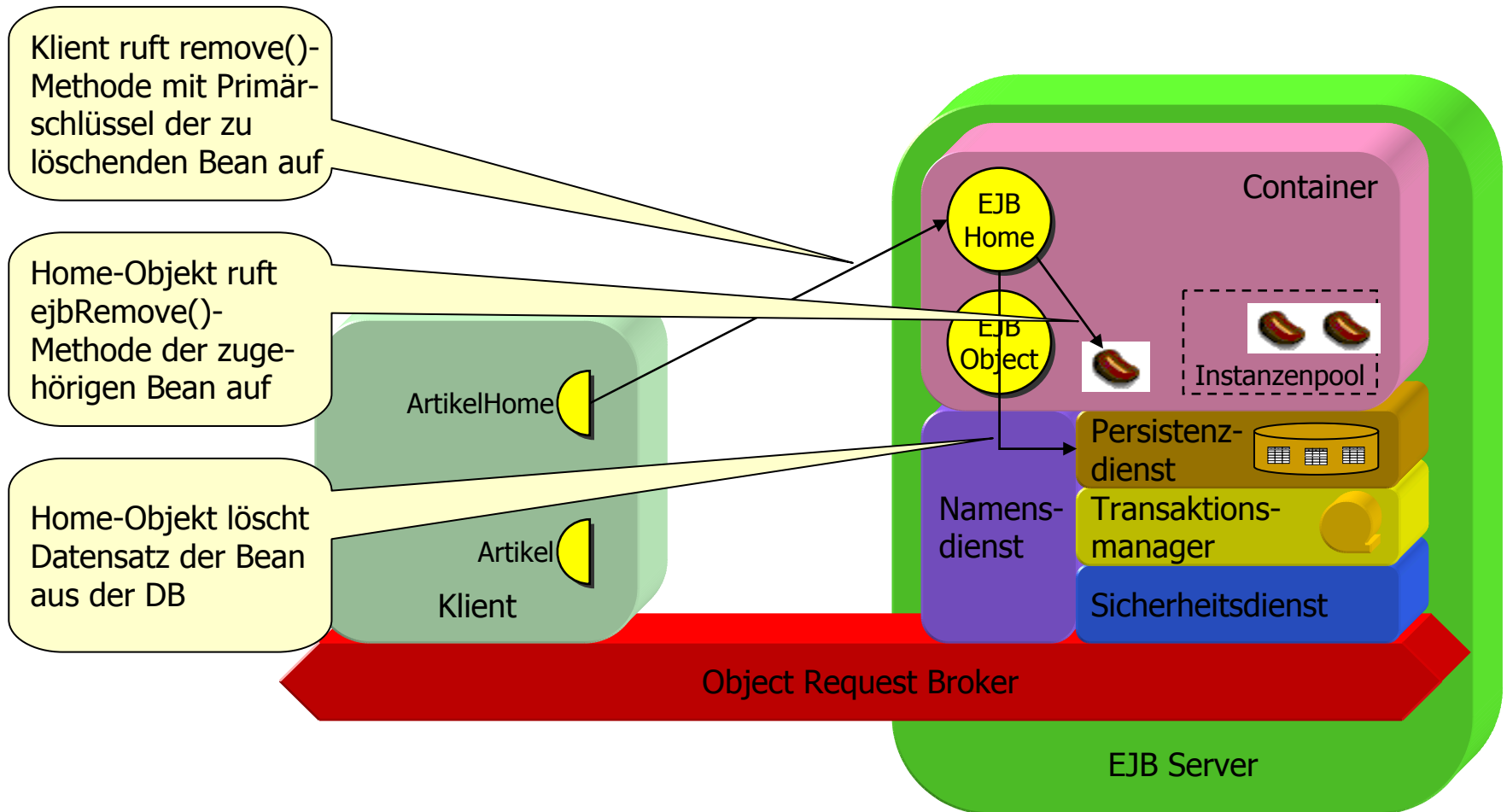
Lebenszyklus einer Entity Bean: Aufruf einer Geschäftsmethode



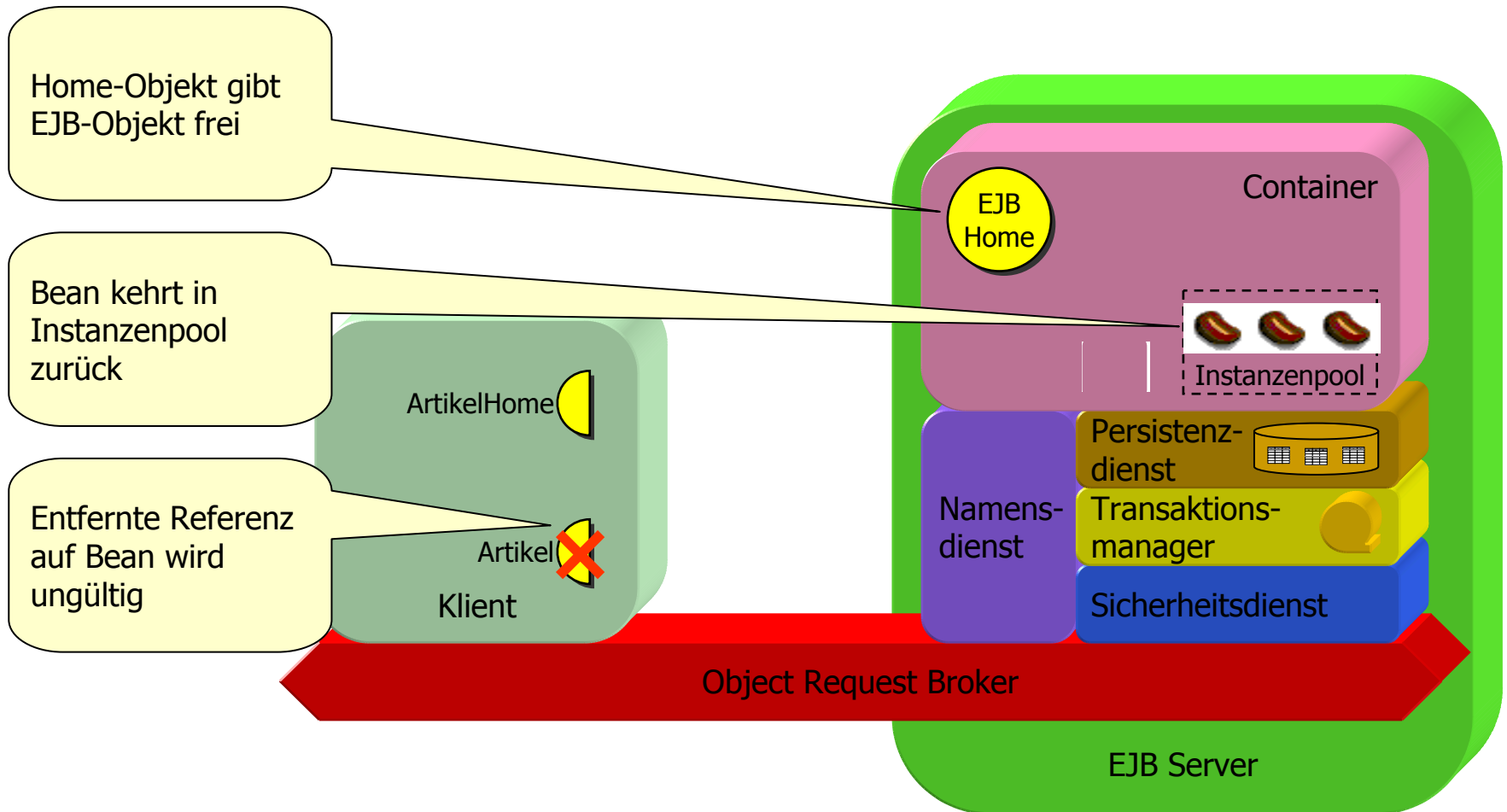
Lebenszyklus einer Entity Bean: Aufruf einer Geschäftsmethode (Forts.)



Lebenszyklus einer Entity Bean: Zerstören einer Bean



Lebenszyklus einer Entity Bean: Zerstören einer Bean (Forts.)



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 Methodenaufufe erstrecken ("Konversationen").
- Beispiel: Adressprüfer
 - ✍ Die Adressprüfer-Bean ist stateless und dient zur Prüfung von Adressen anhand einer Postleitzahl-Datenbasis.

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 Adresspr,fer extends javax.ejb.EJBObject {

    public boolean g,ltig(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 Adresspr,ferHome extends javax.ejb.EJBHome {

    public Adresspr,fer create() throws RemoteException, CreateException;

    // find()-Methoden gibt es f,r 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 Adresspr_ferBean 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 g_ltig(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 zur,ck, 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 Adresspr_fung</description>
      <ejb-name>Adresspr_fer</ejb-name>
      <home>com.klick-and-bau.Adresspr_ferHome</home>
      <remote>com.klick-and-bau.Adresspr_fer</remote>
      <ejb-class>com.klick-and-bau.Adresspr_ferBean</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

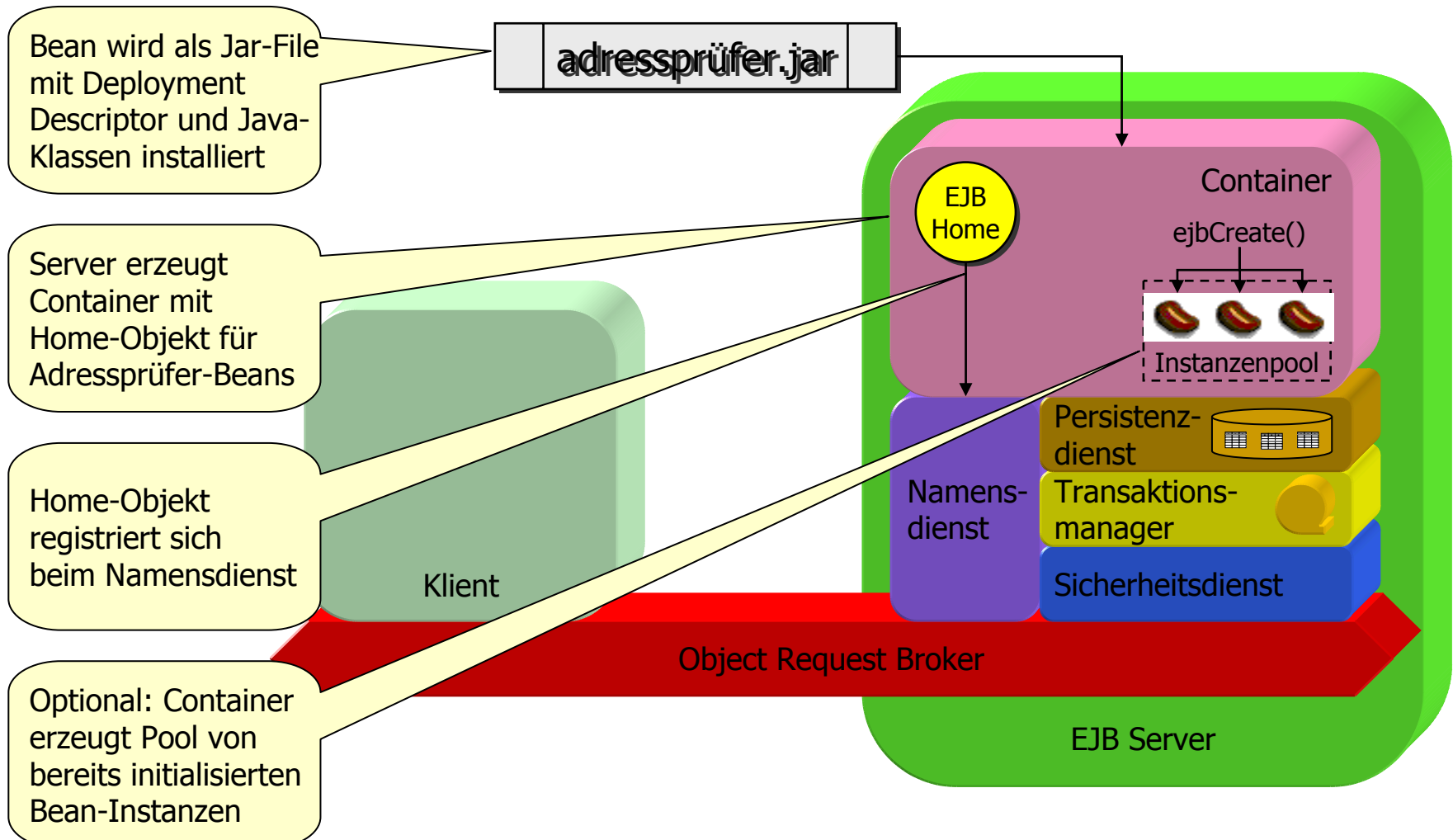
Adressprüfer-Bean: Deployment Descriptor (Forts.)

```
<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>Adresspr , fer</ejb-name>
      <method-name>*</method-name>
    </method>
  </method-permission>
  <container-transaction>
    <method>
      <ejb-name>Adresspr , fer</ejb-name>
      <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
  </container-transaction>
</assembly-descriptor>
</ejb-jar>
```

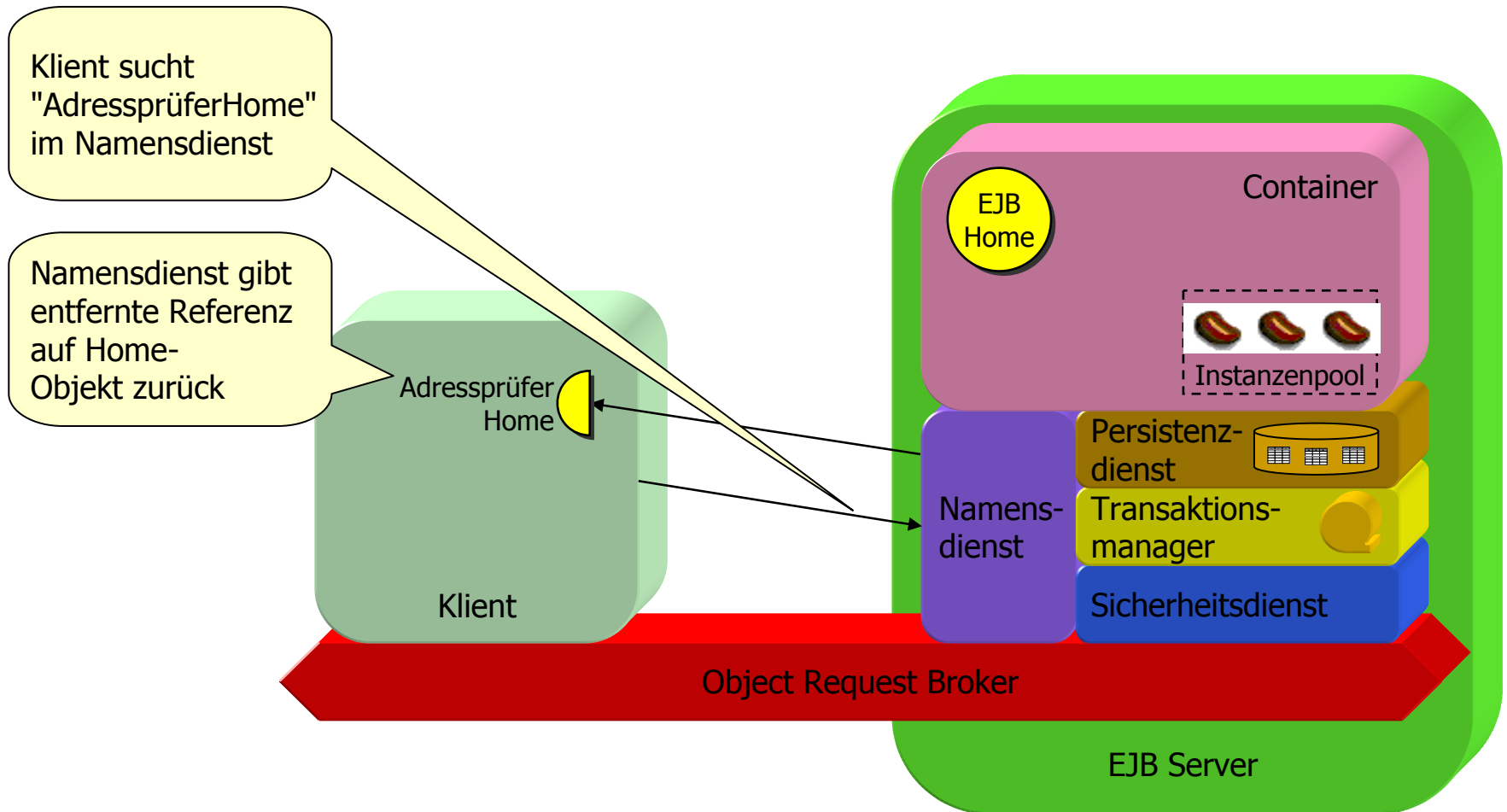
Sicherheit

Transaktionales Verhalten

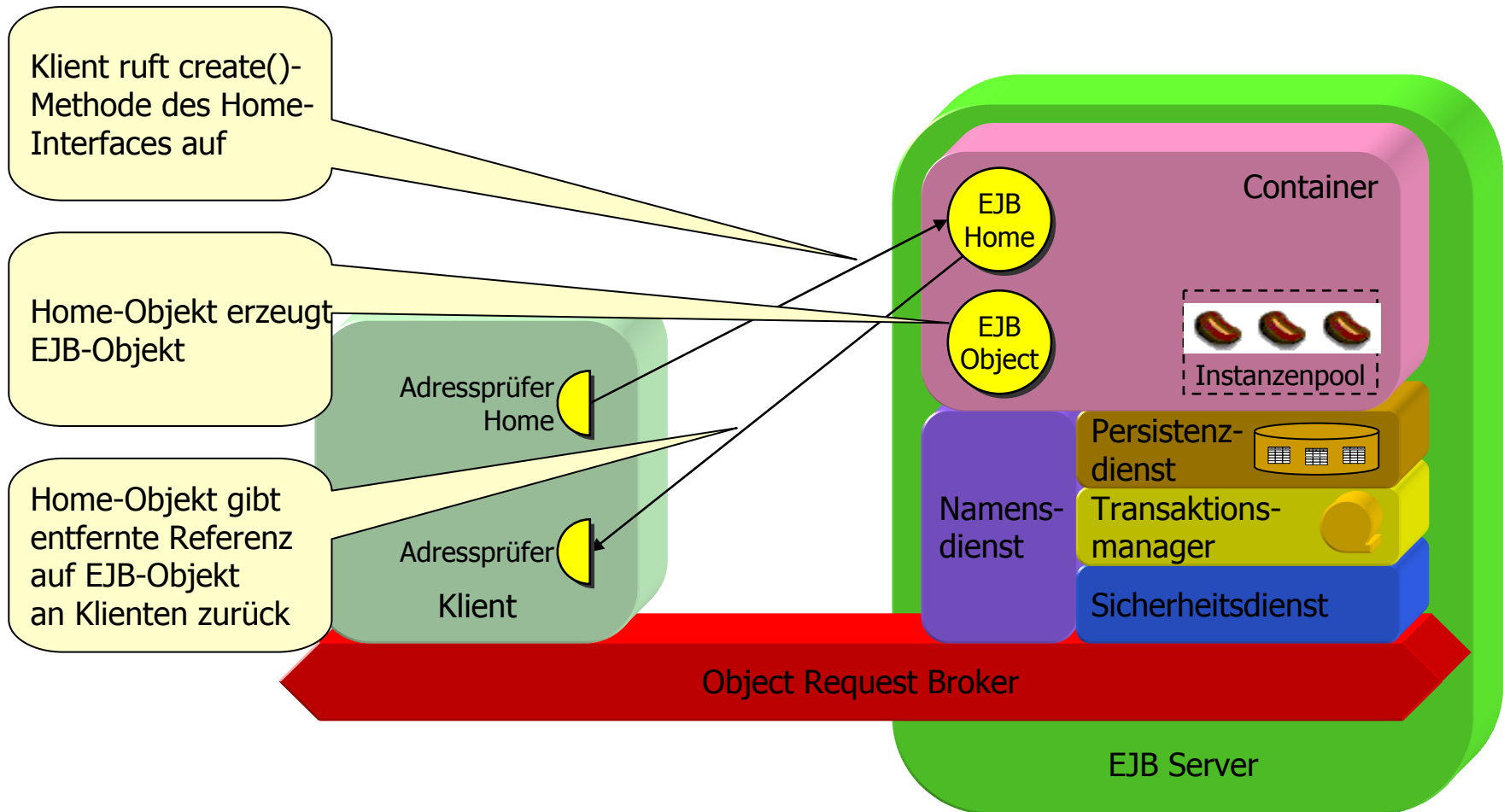
Lebenszyklus einer Stateless Sess. Bean: Installation



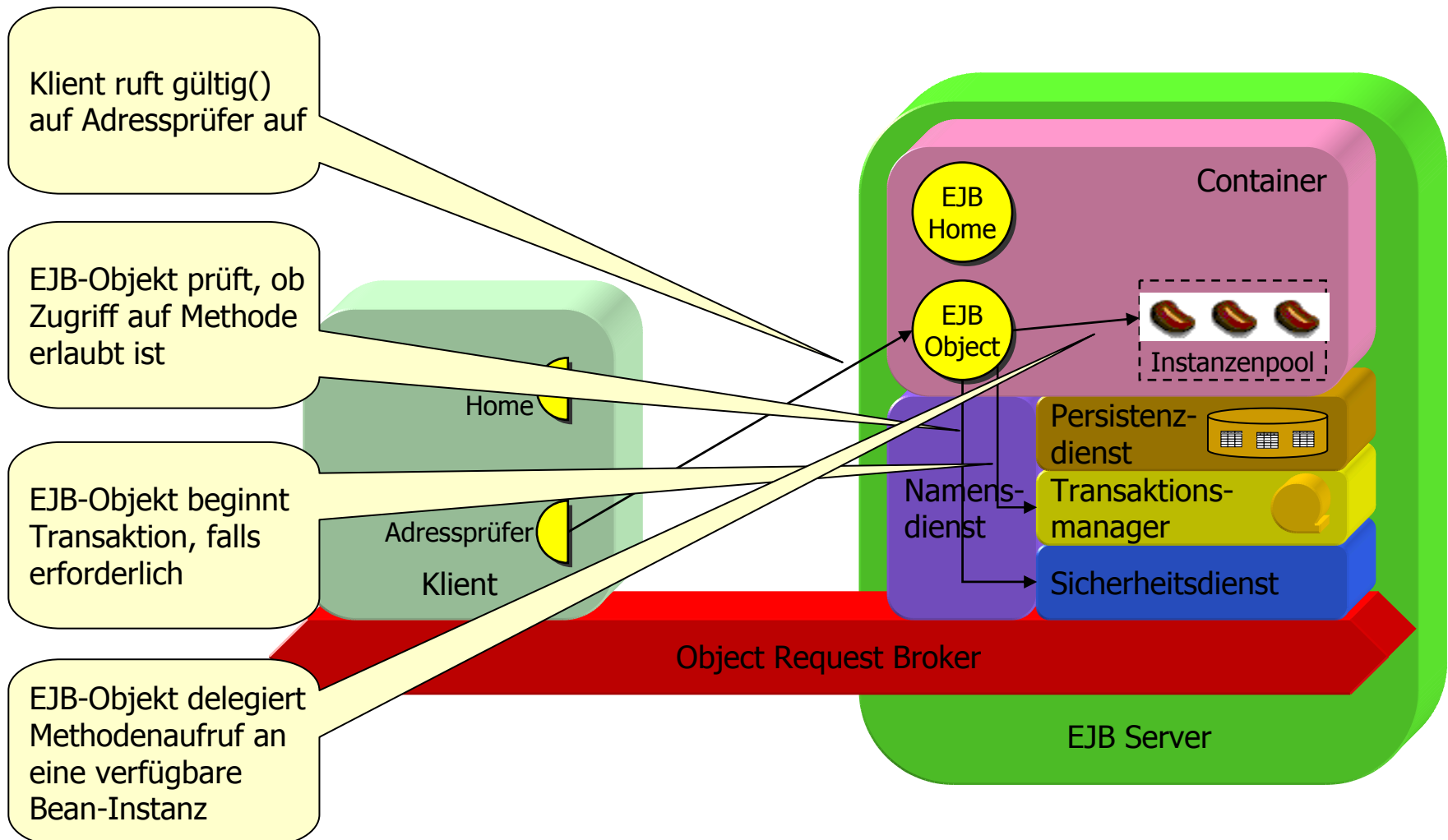
Lebenszyklus einer Stateless Sess. Bean: Erzeugen einer Bean



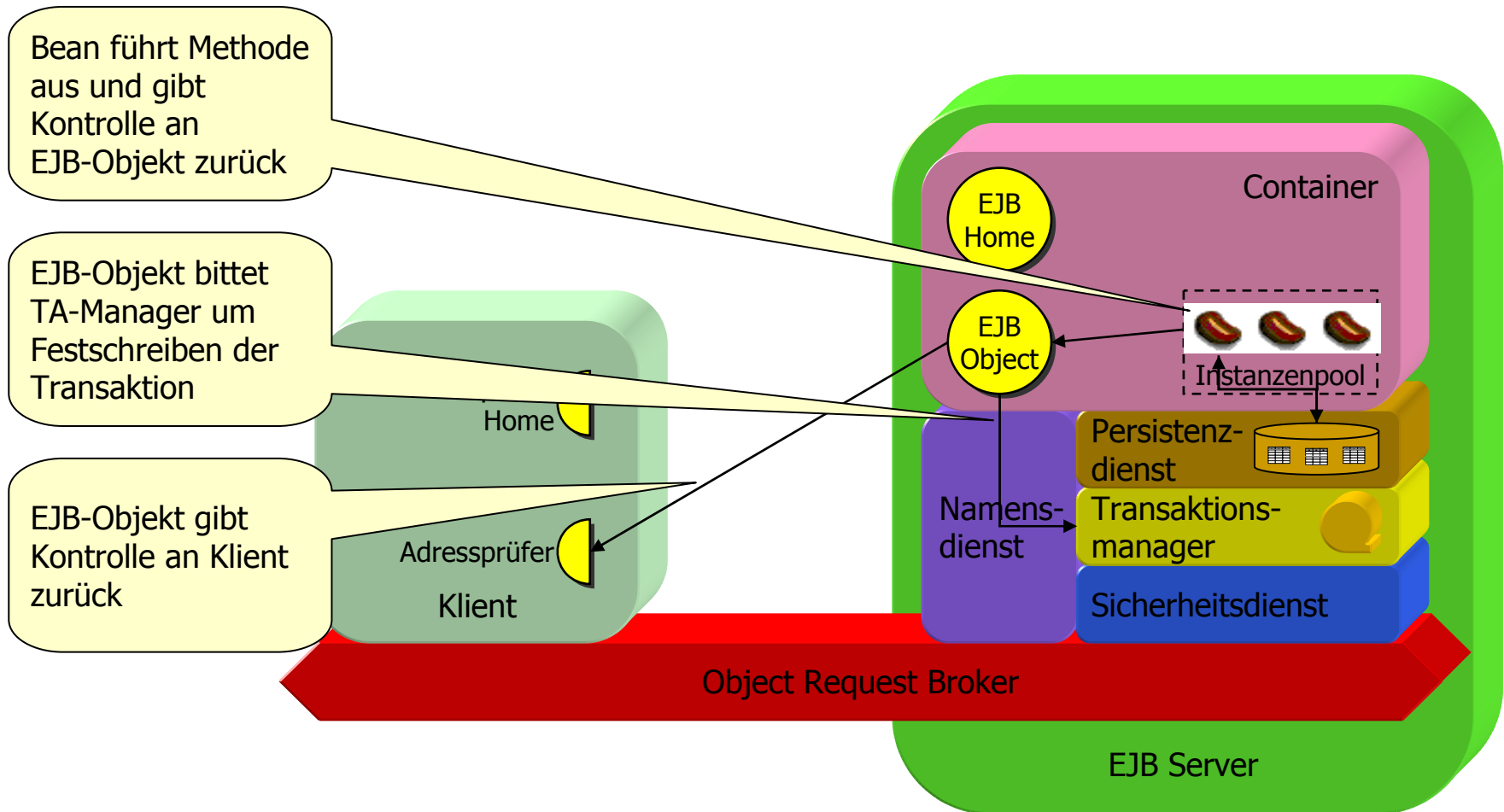
Lebenszyklus einer Stateless Sess. Bean: Erzeugen einer Bean (Forts.)



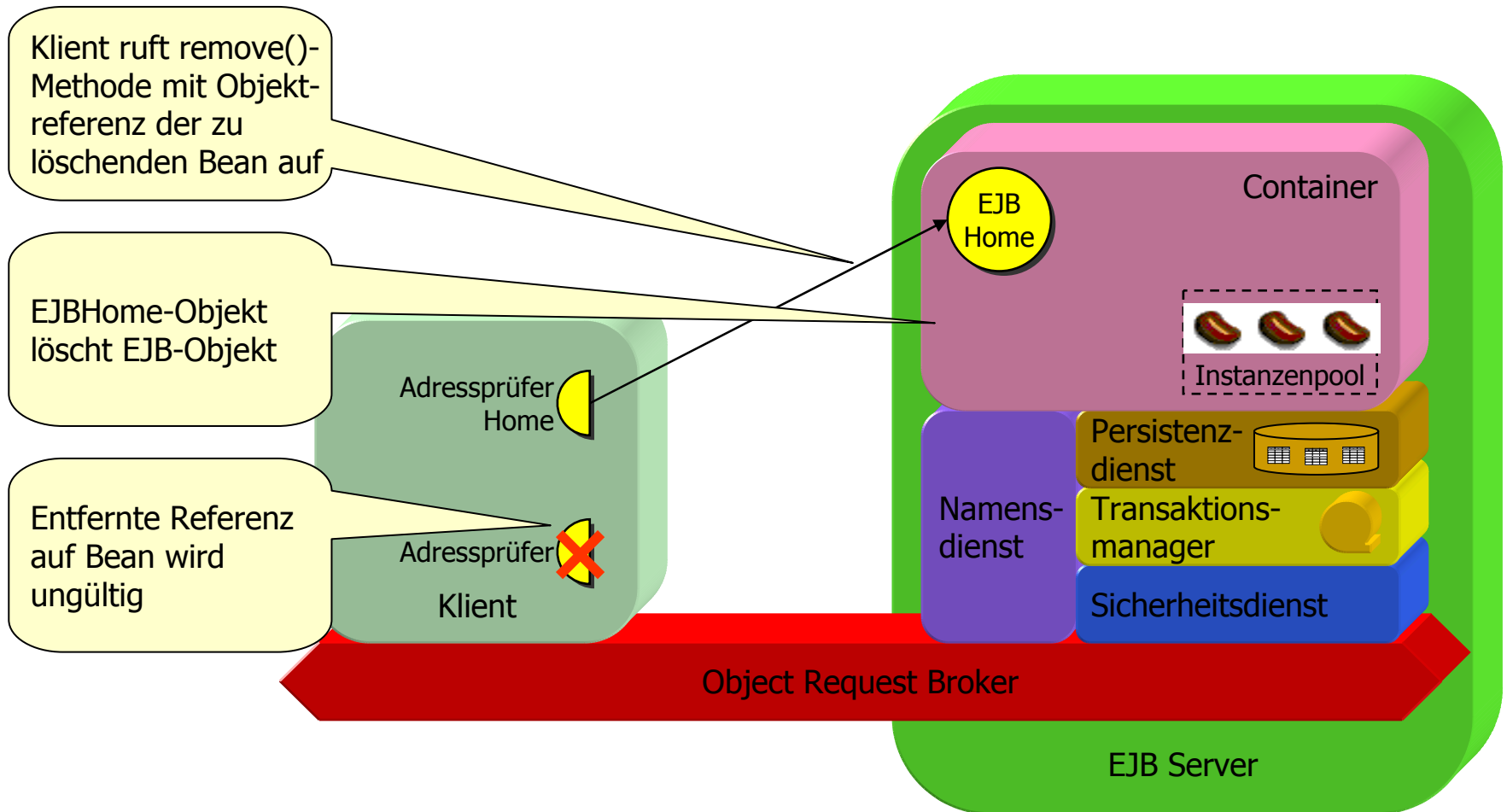
Lebenszyklus einer Stateless Sess. Bean: Aufruf einer Geschäftsmethode



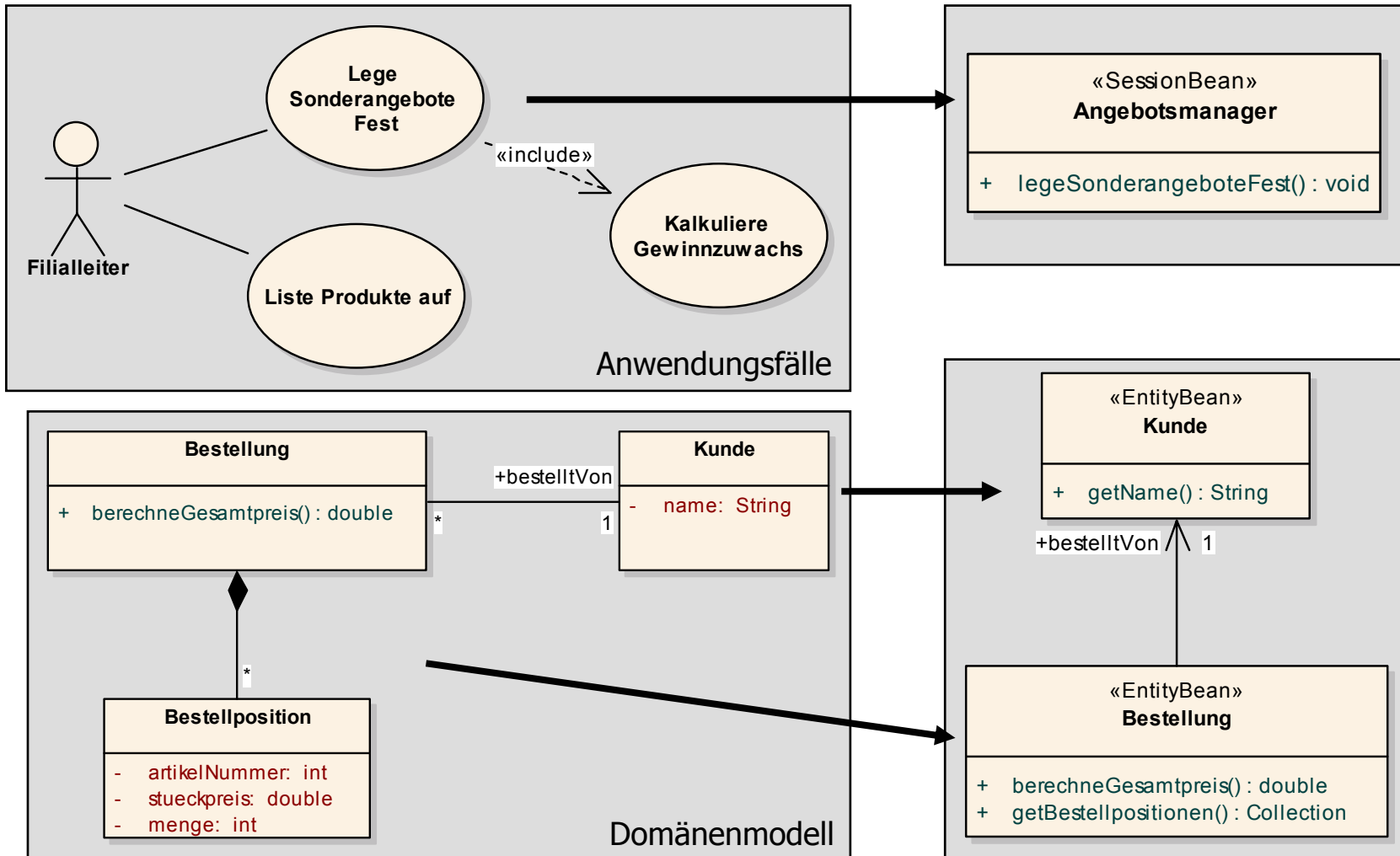
Lebenszyklus einer Stateless Sess. Bean: Aufruf einer Geschäftsmethode (Forts.)



Lebenszyklus einer Stateless Sess. Bean: Zerstören einer Bean

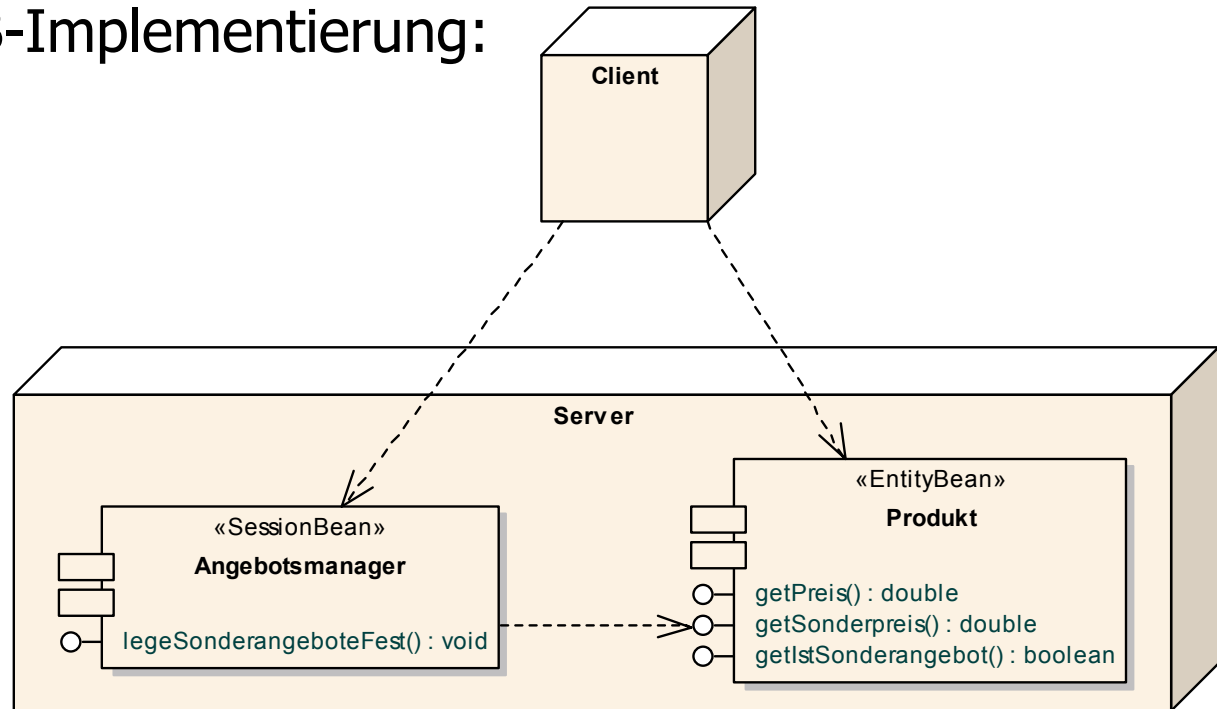


Anwendungsentwicklung mit EJB: Softwaredesign & EJB



Anwendungsentwicklung mit EJB: EJB-Patterns

■ Naive EJB-Implementierung:



■ Problem:

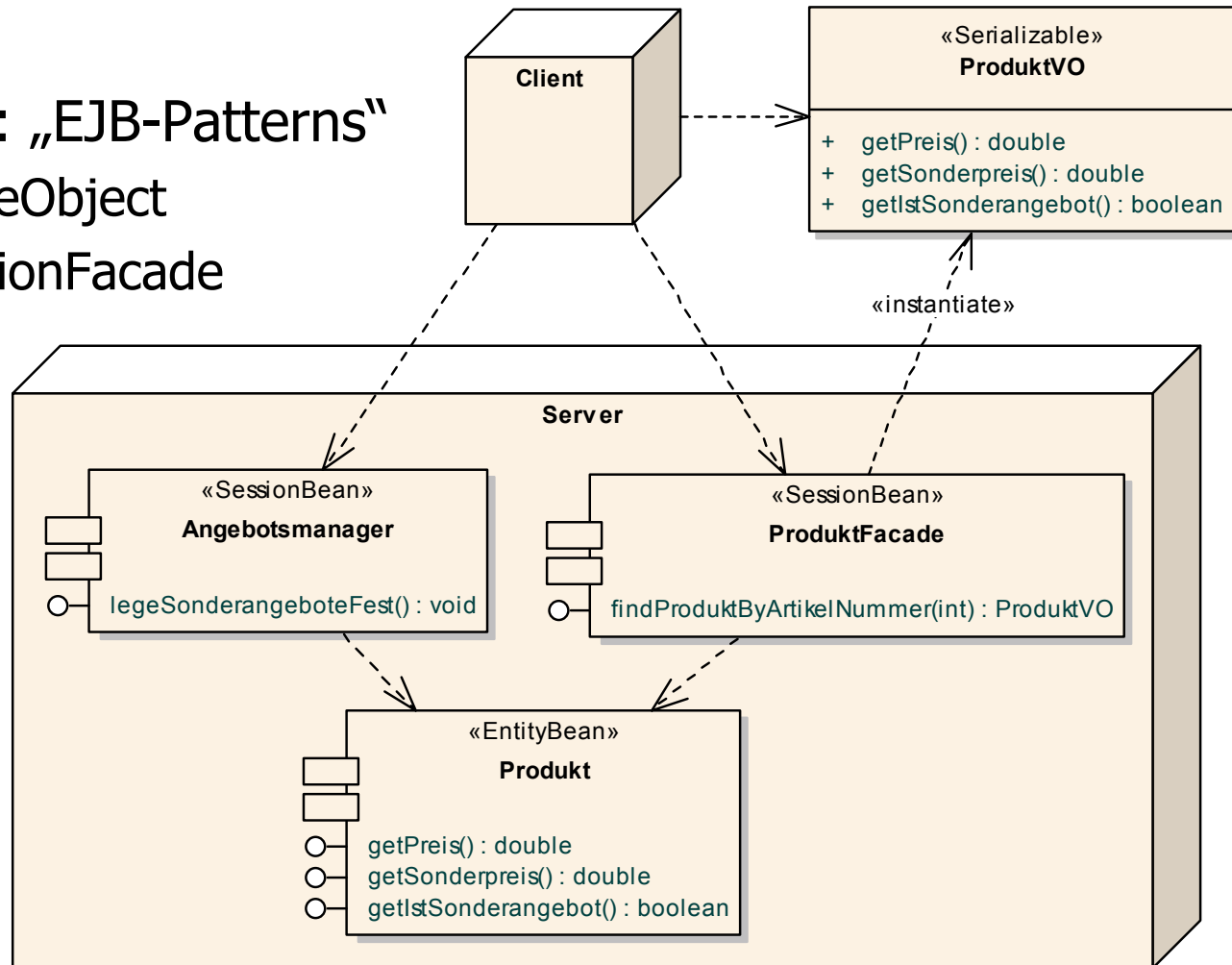
✂ Hohe Latenz durch viele entfernte get()-Aufrufe

Anwendungsentwicklung mit EJB: EJB-Patterns

■ Lösung: „EJB-Patterns“

✍ ValueObject

✍ SessionFacade



■ News und Informationen im Web

 www.theserverside.com

 www.ejbsig.de


 www.javamagazin.de

■ Präsentationen:

 C. Mohan: Tutorial auf der VLDB 2002

www.almaden.ibm.com/u/mohan/AppServersTutorial_VLDB2002_Slides.pdf

■ Zeitschriften:

 JavaMagazin, JavaSpektrum

 A. Bien: „J2EE-Patterns“, in: JavaMagazin 11/2002-08/2003