Kapitel 9

eBusiness-Plattformen am Beispiel von J2EE

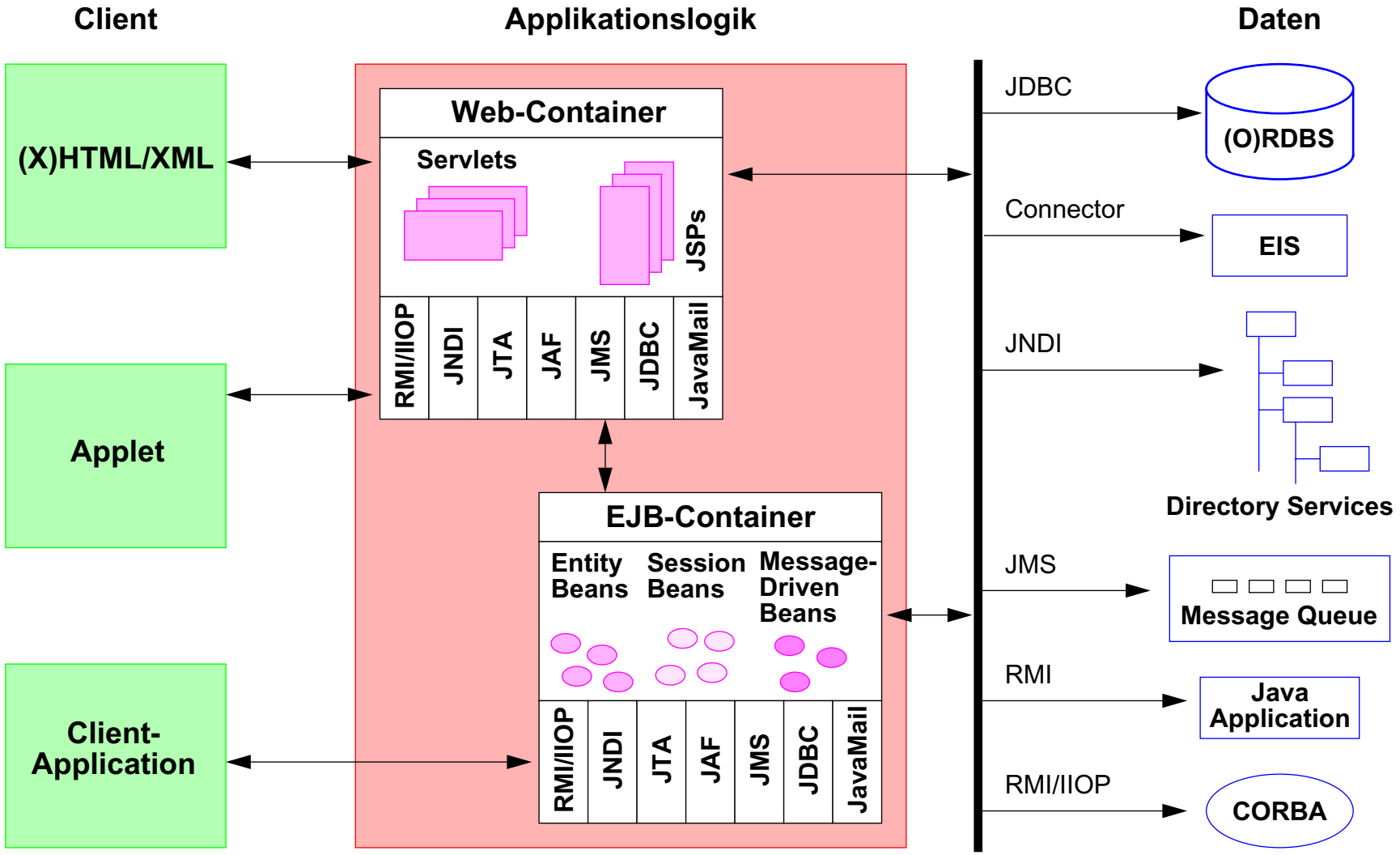
Inhalt

- Einführung
- **J2EE**
  - Überblick
  - Architekturmodelle
  - Enterprise Java Beans (EJBs)
  - Web-Komponenten
- Zusammenfassung

Einführung

- Anforderungen an eBusiness-Systeme:
  - eCollaboration
    - Interoperabilität
    - Verfügbarkeit
    - Durchsatz
  - geringe Kosten
    - Systementwicklung
    - Verwaltung/Administration/Wartung
    - *Unit-of-Work*
    - Skalierung
- Vertreter
  - Microsoft's ***.NET***
  - Sun's ***Java 2 Enterprise Edition*** (J2EE)
    - dient im folgenden als Beispiel
    - offene Spezifikation
    - bedeutendste Produkte (zusammen etwa 60% Marktanteil):
      - IBM's WebSphere und BEA's WebLogic
  - **.NET/MTS** vs. **J2EE/EJB** Vergleichsinfo:
    - Roger Sessions: *Java 2 Enterprise Edition (J2EE) vs. .NET Platform - Two Visions for eBusiness* **(WP!)**
    - Gopalan Suresh Raj: *A Detailed Comparison of Enterprise JavaBeans (EJB) & The Microsoft Transaction Server (MTS) Models*

Überblick - J2EE (1)



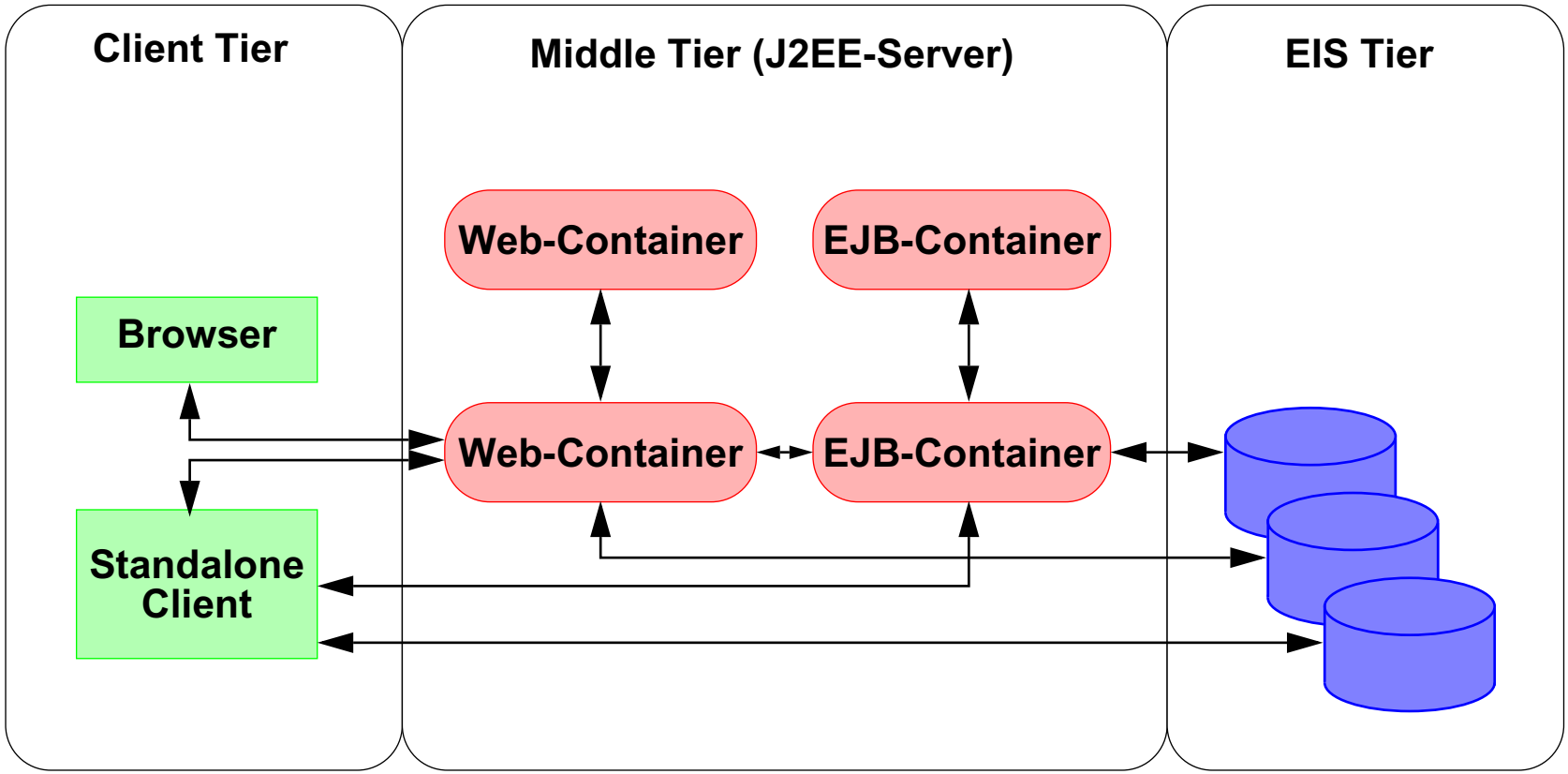
Überblick - J2EE (2)

- Technologien
  - Komponenten-Technologien
    - ***Web-Komponenten***
      - Java Server Pages (JSPs)
      - Servlets
    - ***Enterprise Java Beans (EJBs)***
  - Service-Technologien
    - Java Transaction API (JTA)
    - Java DataBase Connectivity (JDBC)
    - Java Naming and Directory Service (JNDI)
    - Java Message Service (JMS)
    - JavaMail
    - Java Connector Architecture
    - Java Authentication and Authorization Service (JAAS)
  - Kommunikationstechnologien
    - Internet Protokolle
      - HTTP, TCP/IP, SSL
    - Remote Object Protocols
      - Java RMI, RMI/IIOP, Java IDL

Überblick - J2EE (3)

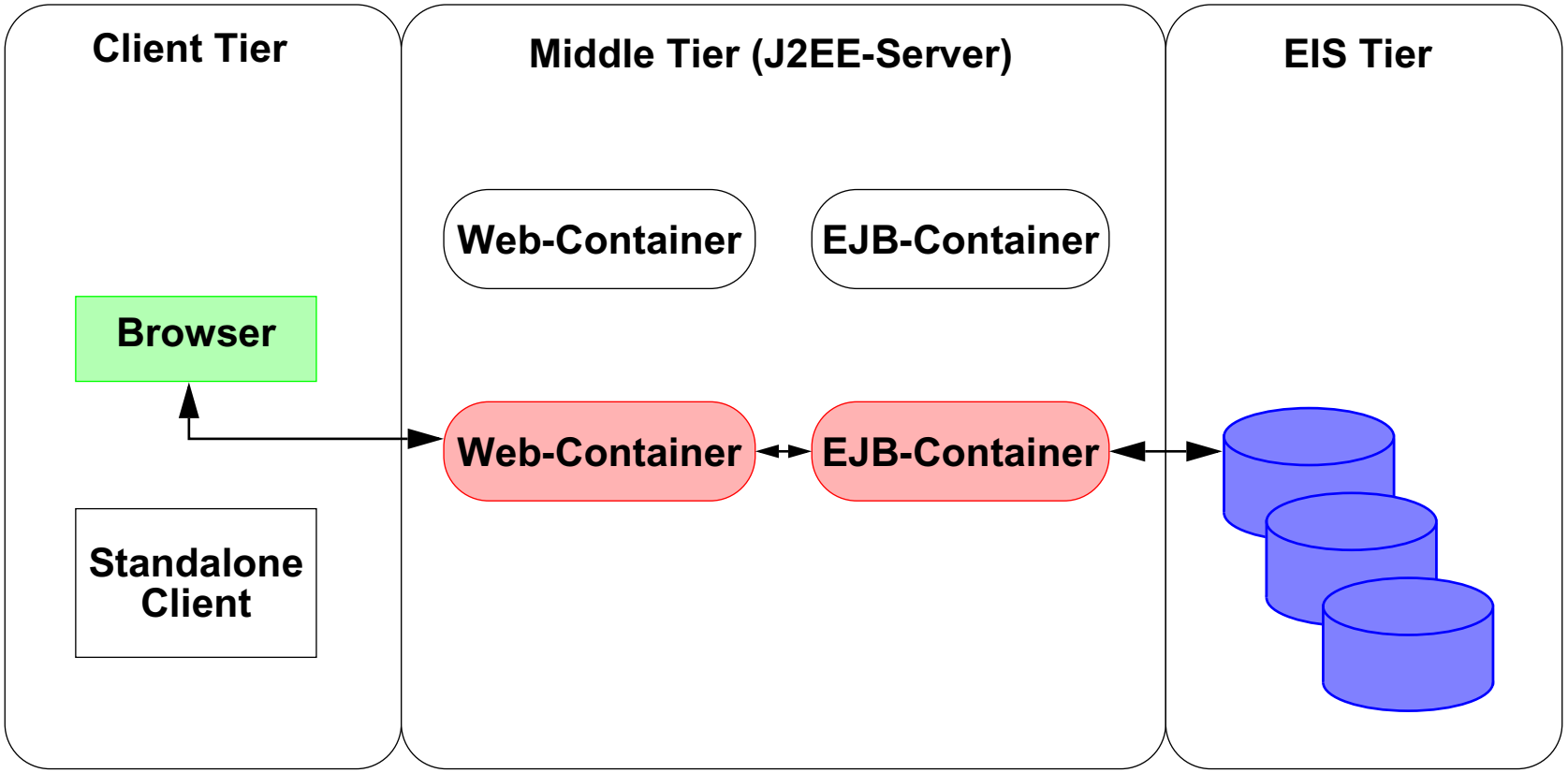
- Container-Begriff
  - allgemeine Dienste einer Laufzeitumgebung
  - Weiterleitung von Client-Anfragen an Komponenten
  - einheitlicher Zugang der Komponenten zu Technologien und APIs
  - ***Declarative Services***
    - generische Implementierung von Komponenten
    - Konfiguration bei Einbindung (Deployment)
      - Beispiel: transaktionales Verhalten
- Deployment-Begriff
  - für EJBs
  - Implementierung einer EJB betrifft lediglich Aspekte der Geschäftslokgik
  - deklarative Spezifikation der technischen Aspekte
  - Konfiguration der technischen Aspekte hinsichtlich einer konkreten Anwendung durch *Deployment Descriptor*
    - transaktionales Verhalten
    - Sicherheit/Zugriffssschutz
    - Bezeichner/Typeigenschaften
    - Beziehungen
    - ...

Architekturmodelle (1)



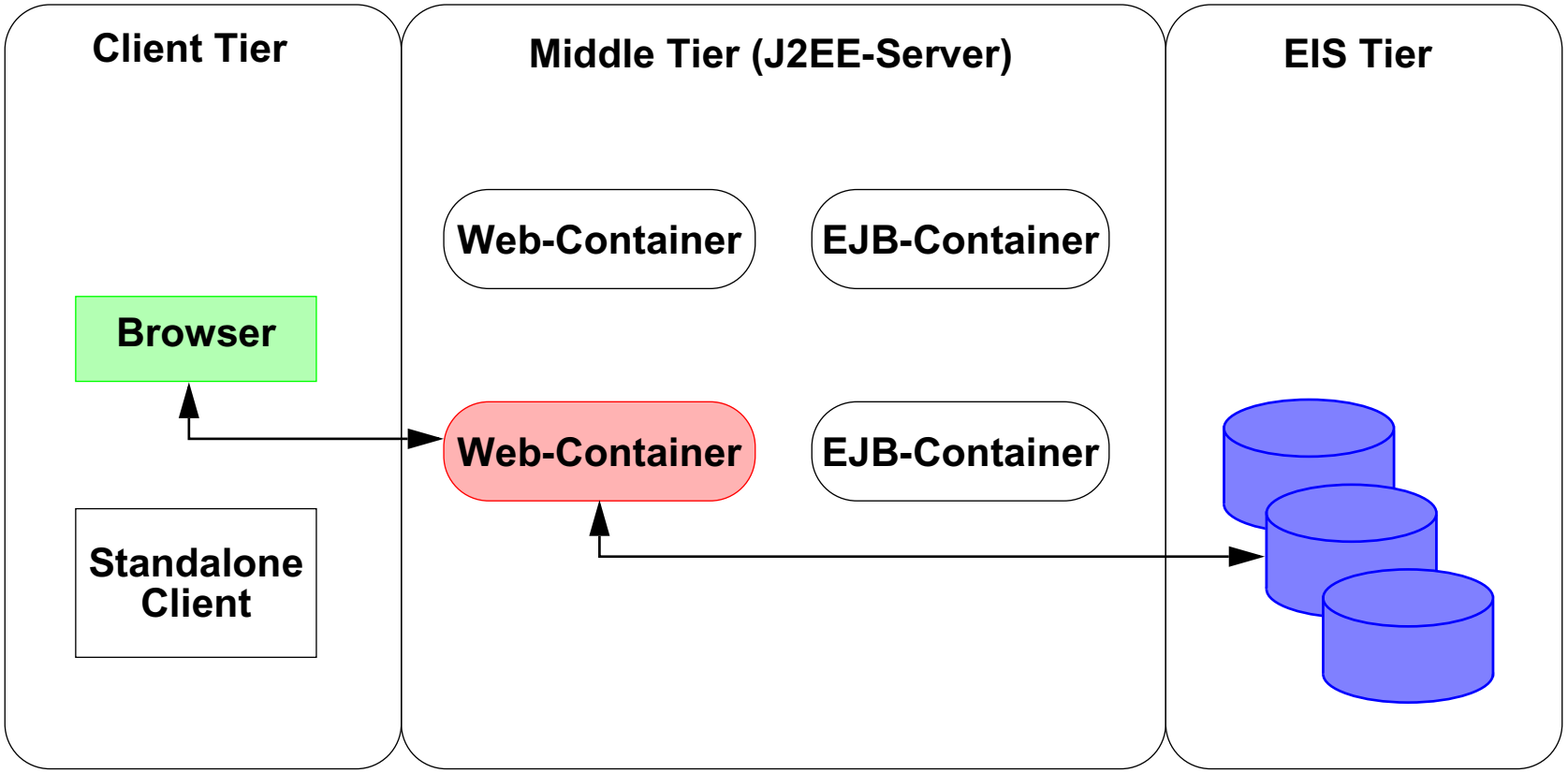
Architekturmodelle (2)

- Mehrebenen-Anwendung



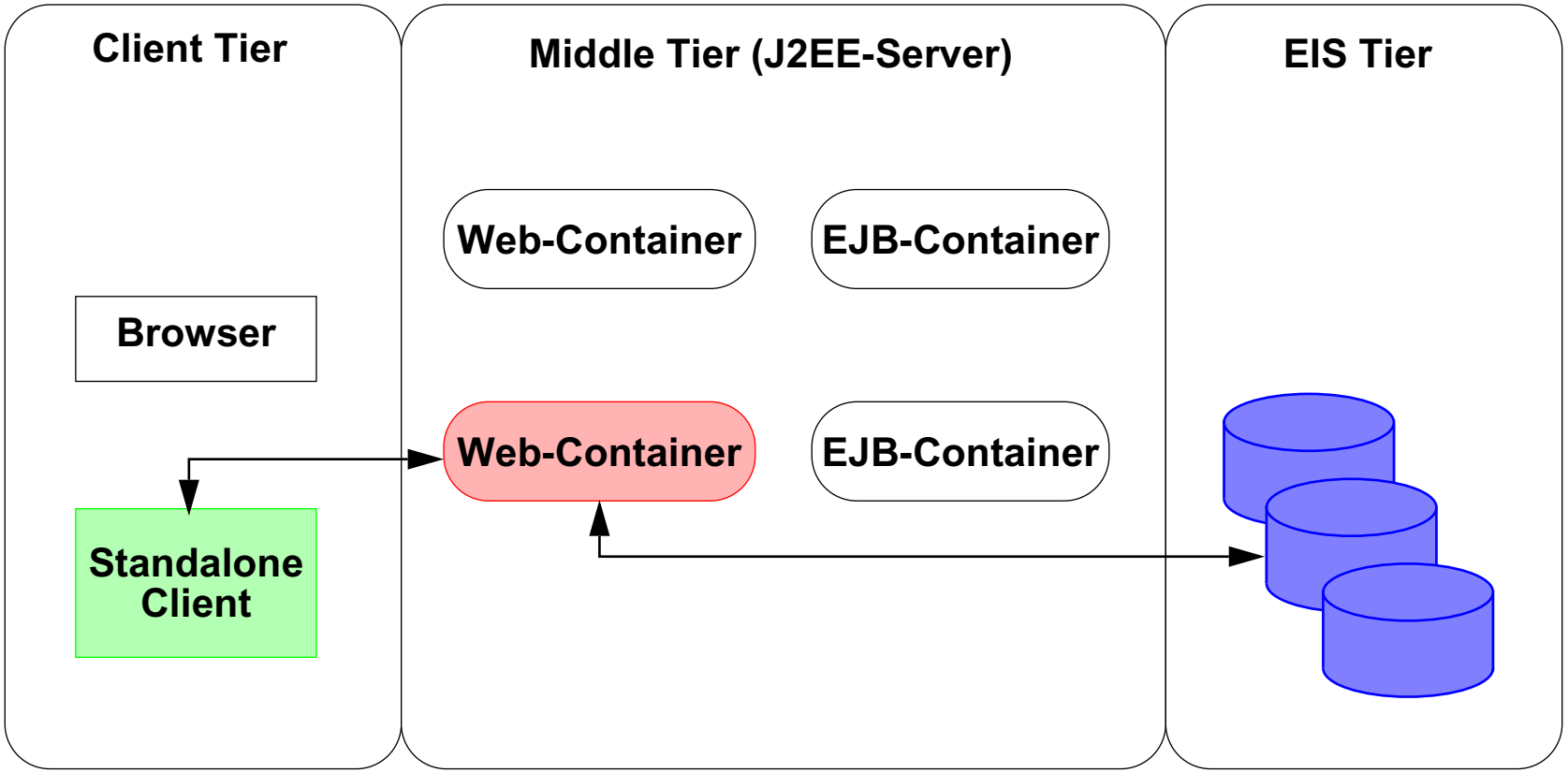
Architekturmodelle (3)

- Web-basierte Anwendung



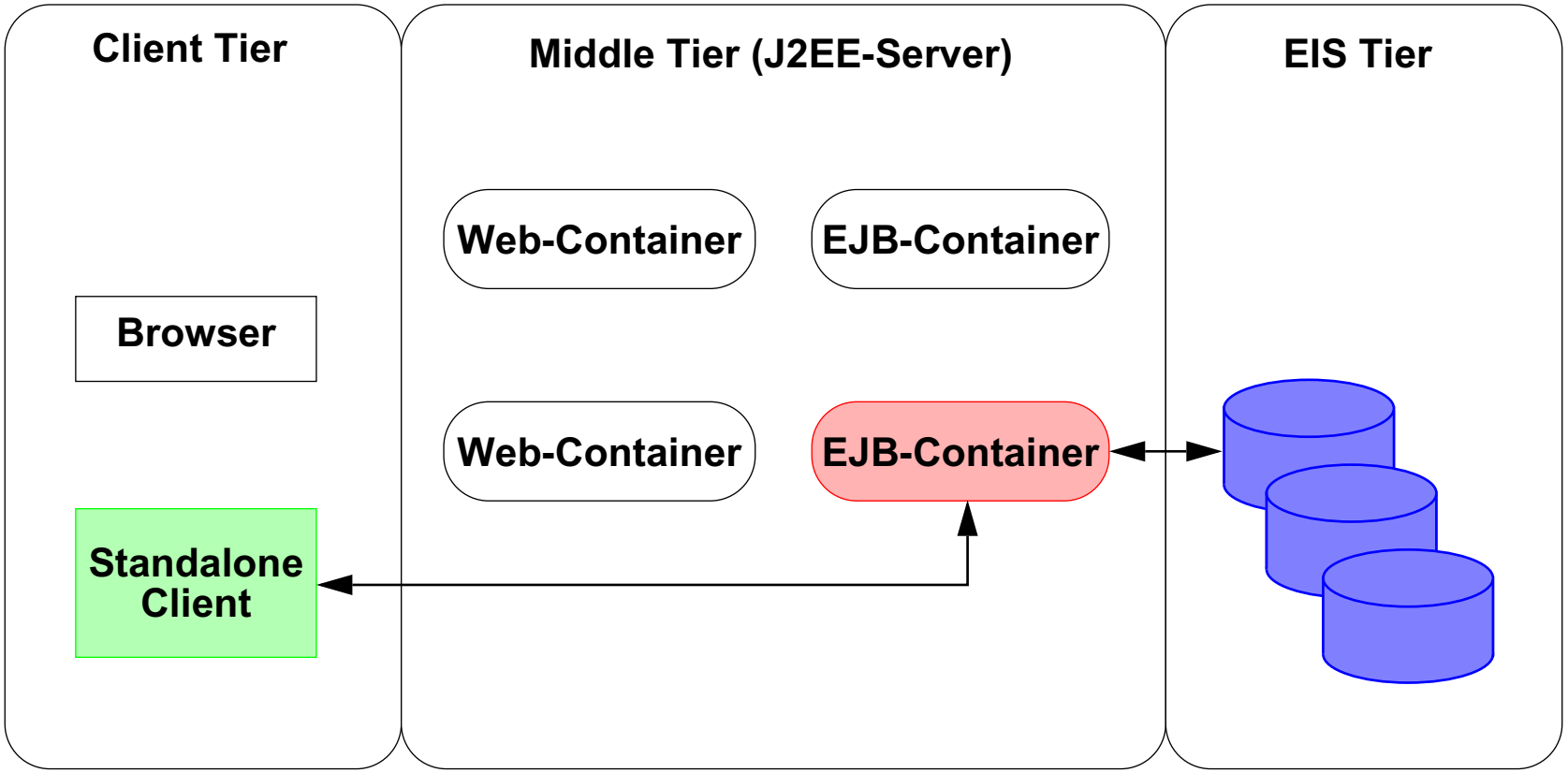
Architekturmodelle (4)

- Standalone-Client I



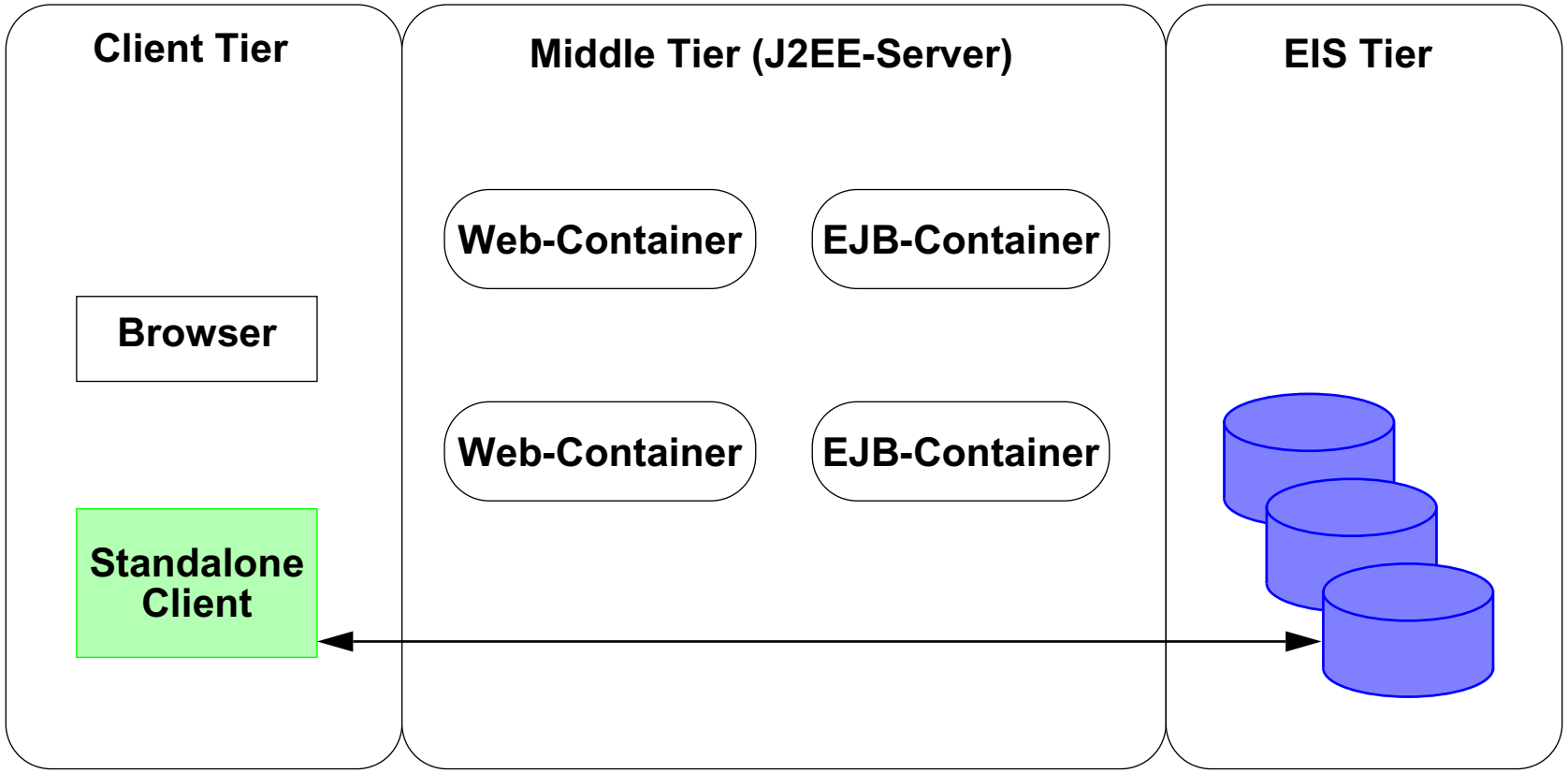
Architekturmodelle (5)

- Standalone-Client II



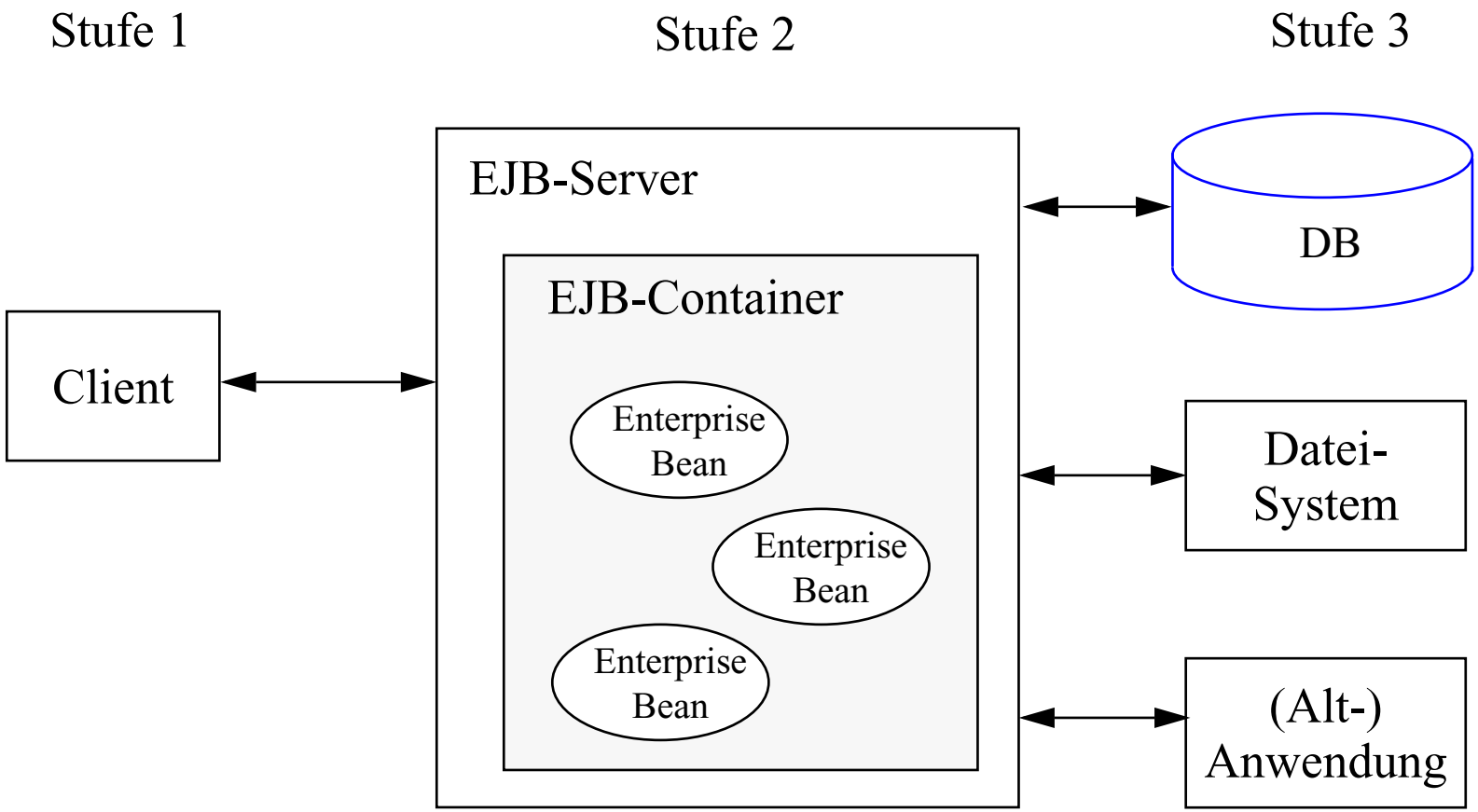
Architekturmodelle (6)

- Standalone-Client III



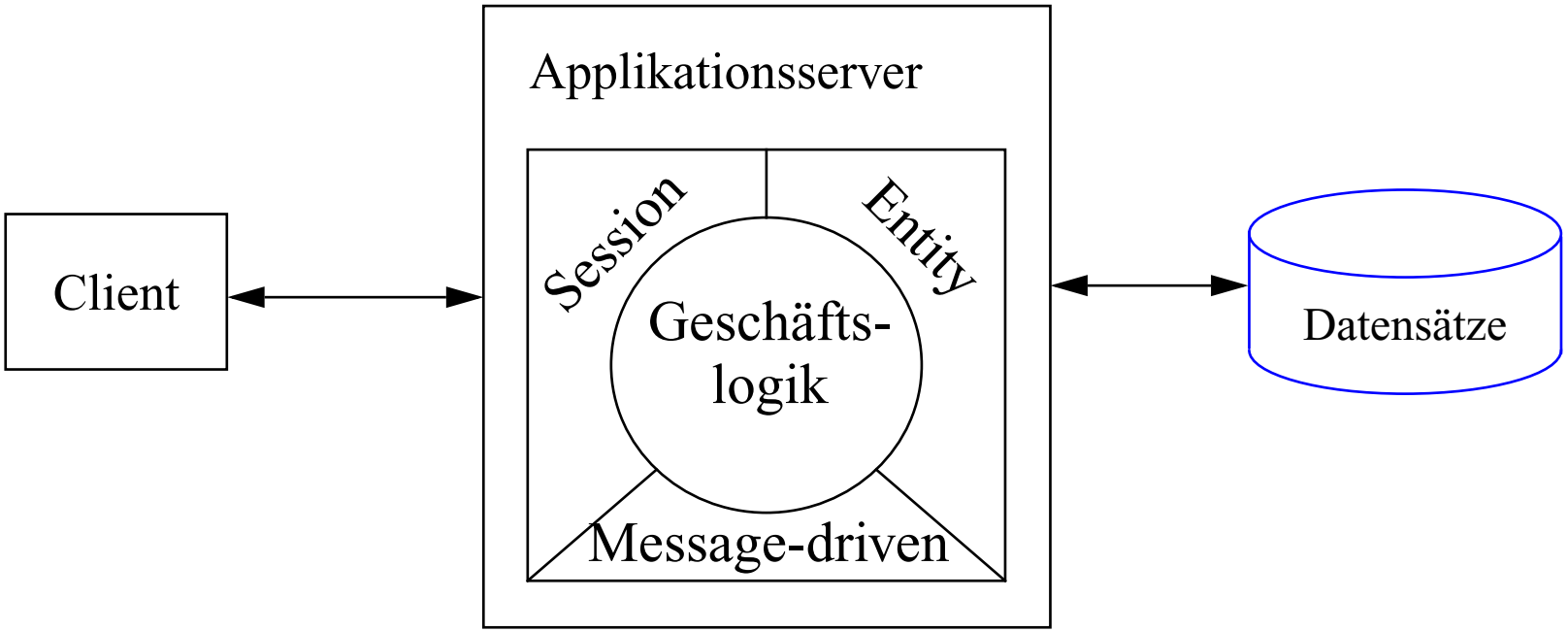
Enterprise JavaBeans (1)

- Mehrebenen Client/Server-Modell



Enterprise JavaBeans (2)

- Session, Entity und Message-driven Beans



Enterprise JavaBeans (3)

- Enterprise Java Bean (EJB)
  - nach EJB-Spezifikation entworfene Komponente
  - besteht aus (*ejb-jar file*):
    - einer Klasse, die die Geschäftslogik implementiert
    - einem Remote-Interface, das die Methoden beschreibt
    - einem Life-Cycle-Interface
    - einem Deployment-Deskriptor
    - einer Primary-Key-Klasse, die persistente Bean-Objekte eindeutig referenziert
- Grundtypen
  - *Session Beans*
    - realisiert geschäftl. Aktivität/Prozess
    - zustandslos (stateless), bzw. flüchtiger Zustand (stateful) für die Dauer einer Session (conversational state)
  - *Entity Beans*
    - repräsentiert (dauerhaftes) Geschäftsobjekt/-konzept
    - persistenter Zustand
    - Primary-Key ermöglicht eindeutigen Zugriff
  - *Message-driven Beans*
    - asynchron, botschaftenorientiert (JMS)
    - erleichtert Intergration mit existierenden Anwendungen

Enterprise JavaBeans (4)

- Entity Beans
  - Persistenz
    - Bean-Managed Persistence (BMP)
      - Callback-Methode, mit der Container Bean anzeigt, Zustand zu speichern
      - Datenbankzugriffe sind Teil des Beans
      - Nutzung vor allem bei Legacy-Systemen bzw. komplizierter DB-Anbindung
    - Container-Managed Persistence (CMP)
      - Container sichert Zustand zu bestimmten Zeitpunkten
      - Zuordnung von Bean-Attributen zu DB-Strukturen in Deployment-Phase
      - Bean-Programmierer muß sich kaum mit dem Persistenz-mechanismus auseinandersetzen
- Session Beans
  - nicht persistent, können jedoch persistente Daten manipulieren
  - beispielsweise Nutzung zur Reduzierung des Nachrichtenaufkommens sinnvoll
  - zustandslose Session-Bean:
    - Zustand nur für einen Methodenaufruf
  - zustandsbehaftete Session-Bean:
    - Zustand über Methodenaufrufe hinweg
    - Zuordnung von Bean-Instanz zu Client notwendig

Enterprise JavaBeans (5)

- Deployment
  - EJB ist Server-unabhängig
  - Erinnerung: Anpassung an die Spezifika des Servers
    - Bekanntmachung der Klassen und Interfaces
    - Verbinden von Bean-Attributen mit DB-Strukturen
    - Konfiguration bzgl. Transaktionsverwaltung
    - Konfiguration bzgl. Sicherheit
    - Setzen von Umgebungsvariablen
    - Erzeugung von *Glue-Code*
  - *Deployment Descriptor*
    - XML-Datei, entsprechende DTD wird vorgegeben
    - Beschreibung von:
      - Typ, Name
      - Persistenzart
      - Klasse, Interfaces, Primary-Key
      - persistente Felder bei Container-Verantwortlichkeit
      - Umgebungsvariablen, benötigte Ressourcen (DBS, etc.)
      - Referenzen zu den Home-Interfaces von benutzten EJBs
      - Transaktionsparameter der einzelnen Methoden
      - Referenzen auf Sicherheitsrollen

Enterprise JavaBeans (6)

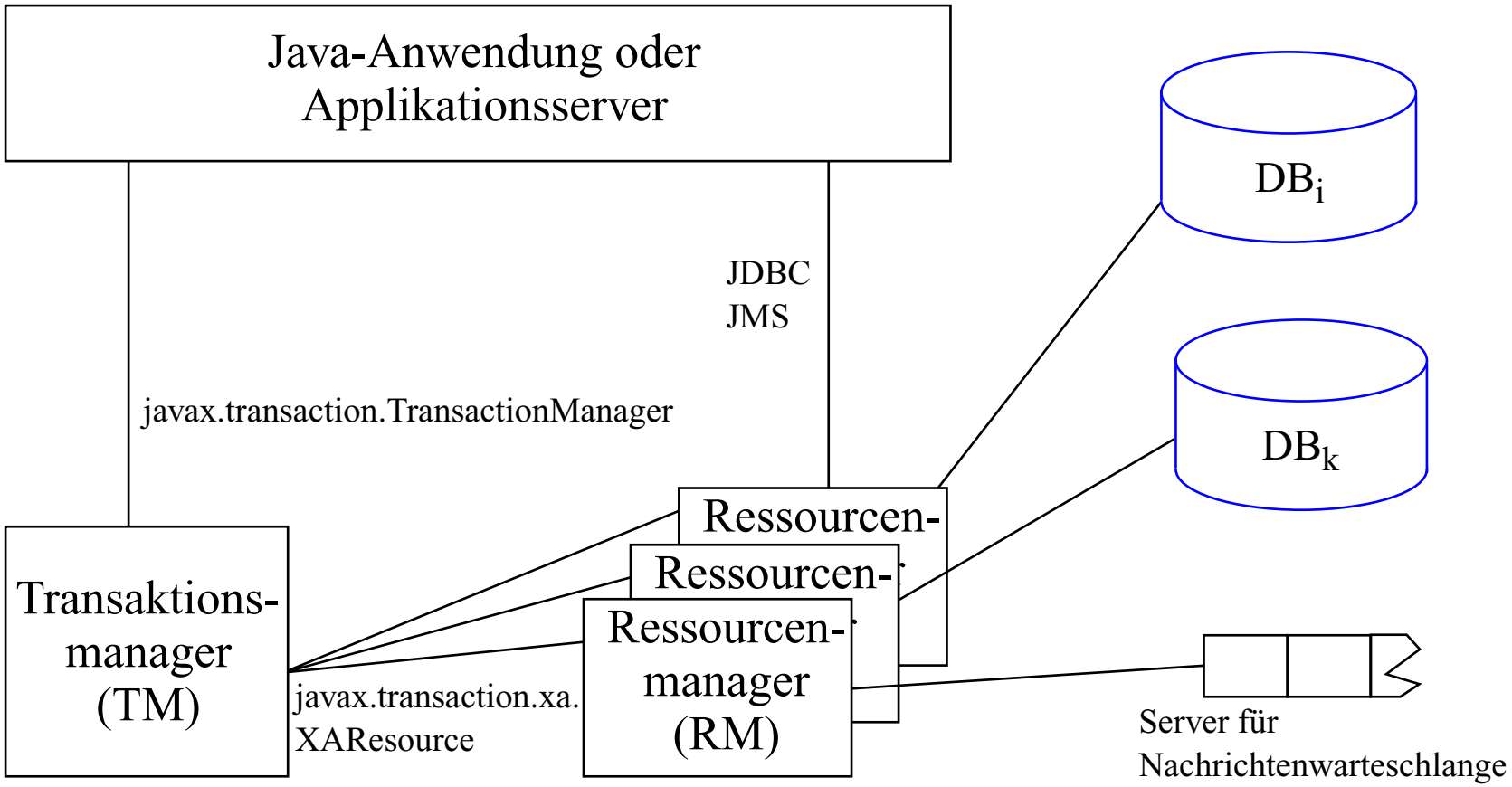
- Transaktionale Eigenschaften
  - programmatic vs. declarative transaction demarcation
  - Transaktionsattribute für Methoden im Deployment Descriptor:

Transaktionsattribute	Client-Transaktion	Transaktion in Methode
NotSupported	keine	keine
	T1	keine
Supports	keine	keine
	T1	T1
Required	keine	T2
	T1	T1
RequiresNew	keine	T2
	T1	T2
Mandatory	keine	Fehler
	T1	T1
Never	keine	keine
	T1	Fehler

  - explizite (programmatische) Demarkation von Transaktionen durch Client, Beans:
    - Nutzung des Java Transaction APIs (JTA)
    - UserTransaction Objekt bereitgestellt durch JNDI (oder EJB Kontext)
    - nicht erlaubt für EntityBeans

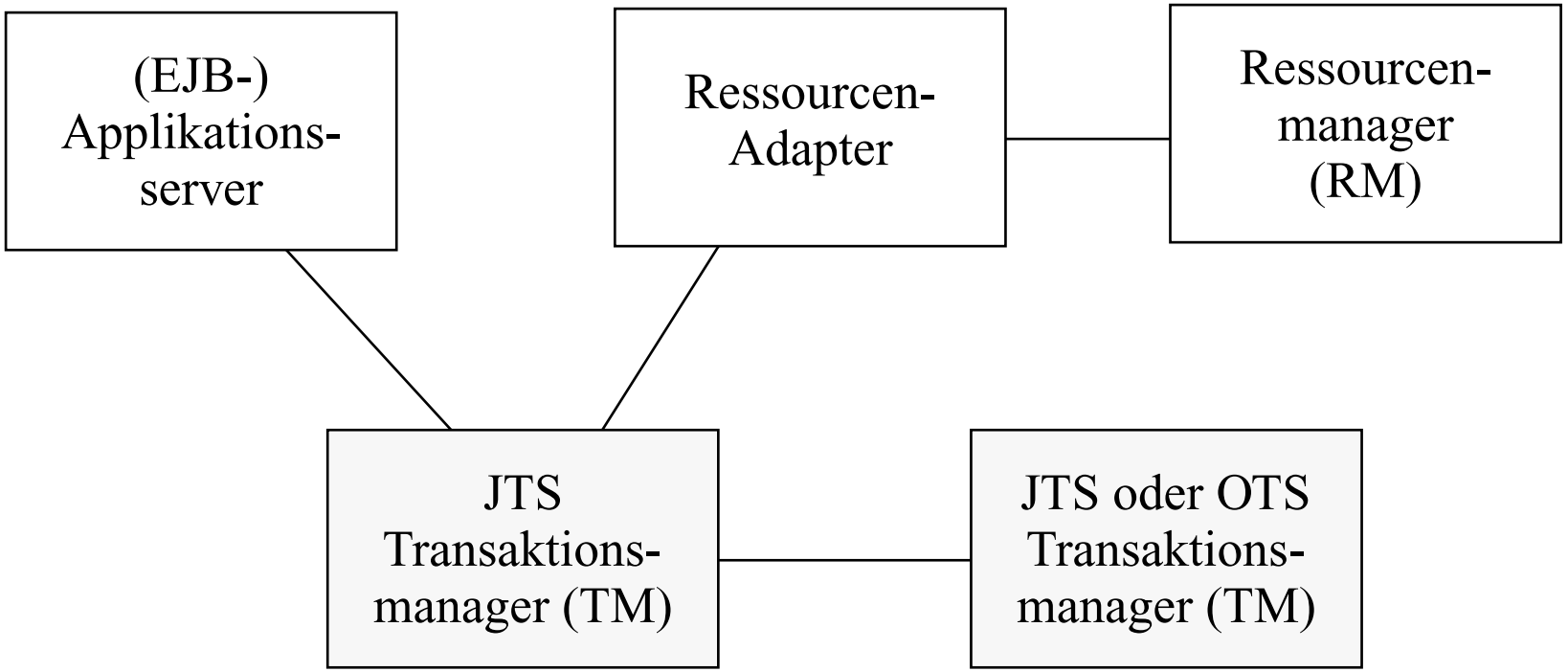
Enterprise JavaBeans (7)

- Zugriff auf XA-kompatible Ressourcen



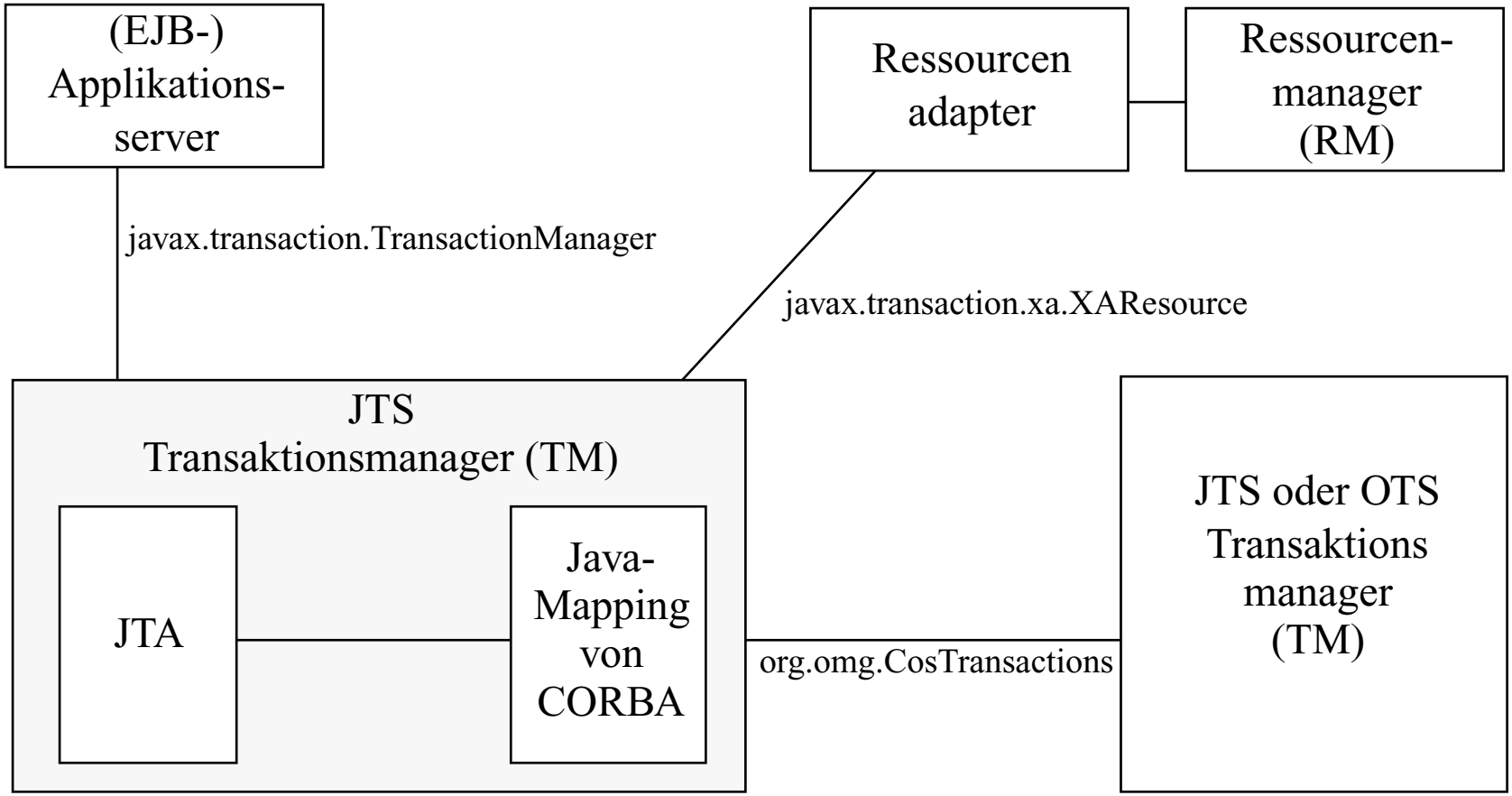
Enterprise JavaBeans (8)

- Grobarchitektur von JTS



Enterprise JavaBeans (9)

- Interoperabilität von JTS und OTS



Enterprise JavaBeans (10)

- Neuerungen in EJB 2.0
  - *EJB-Query-Language*
    - SQL-92-ähnliche Anfragesprache
  - *Message-driven Beans*
    - auch asynchroner Empfang von Nachrichten (im Rahmen von *Point-to-Point-* oder *Publish/Subscribe-Messaging*, die durch JMS unterstützt werden)
  - ...
- CORBA
  - nach Einführung und raschem Markterfolg von EJBs in Zugzwang geraten
  - enthält nun *CORBA Component Model* (***CCM***) als Middle-Tier-Infrastruktur
    - übernimmt Konzepte, die sich bei EJB 'bewährt' haben
    - Unterscheidung zwischen Implementierung und Deployment
    - Container (Transaktionen, Persistenz, Zugriffsschutz, Ereignisse)
    - Interoperabilität mit EJBs
    - Vorteil: CORBA-Komponenten können in mehreren Programmiersprachen realisiert werden

Web-Komponenten (1)

- Dynamische Erzeugung von Web-Seiten
- Servlets
  - Beispiel:

```
import javax.servlet.*;
import javax.servlet.http.*;
public class MyServlet extends HttpServlet {
   public void doGet(HttpServletRequest req,
                     HttpServletResponse resp)
                     throws ServletException, IOException {
      HttpSession ses = req.getSession();
      RequestDispatcher disp = req.getRequestDispatcher();
      if (ses.getAttribute("name")==null) {
         disp.forward(resp.encodeURL("/login");
      }
      else {
         // ...
         resp.getWriter().println(
                           "Hallo "+ses.getAttribute("name"));
         disp.include("/template/banner.html");
      }
   }
}
```

  - Eigenschaften
    - besseres Leistungsverhalten im Vergleich zu CGI
      - leichtgewichtige Prozesse
      - einmaliges Laden und Starten
    - unkomfortable Behandlung der Ausgabe
    - Vermischung von Präsentation und Logik

Web-Komponenten (2)

- Java Server Pages (JSPs)
  - HTML-Seiten mit eingebettetem Java-Kode zur dynamischen Generierung von Inhalten
  - Trennung von Präsentation und Logik
  - einfach nutzbar
  - basiert auf Servlet-Technologie
  - Elemente
    - statischer Text: ***<H1>Welcome</H1>***
    - JSP-Skriptelemente
      - Ausdrücke: ***<%= clock.getDayOfMonth() %>***
      - Skripte: ***<% } else { %>***
      - Deklarationen: ***<%! void doSomething() { doNothing(); } %>***
    - Direktiven: ***<%@ include file="copyright.html" %>***
    - Aktionen: ***<jsp:useBean id="clock" class="calendar.jspCalendar" />***
    - implizite Objekte: ***<% x = session.getAttribute("xValue") %>***

Web-Komponenten (3)

- Java Server Pages (JSPs, Forts.)
  - Beispiel

```
<HTML>
<%@ page language="java" imports="com.wombat.JSP.*" %>
<jsp:useBean id="clock" class="calendar.jspCalendar" />
<H1>Welcome</H1>
<P>Today is </P>
<UL>
   <LI>Day: <%= clock.getDayOfMonth() %>
   <LI>Year: <%= clock.getYear() %>
</UL>
<% Calendar c = Calendar.getInstance();
      if (c.get(Calendar.AM_PM) == Calendar.AM) { %> Good Morning <% }
      else { %> Good Afternoon <% } %>
<%@ include file="copyright.html" %>
</HTML>
```

Zusammenfassung

- 3-Tier-Architekturen setzen sich für eBusiness-Systeme durch
- eBusiness-Plattformen
  - J2EE
    - Familie von Spezifikationen, kontrolliert durch SUN
    - sprachabhängig, auf Java zugeschitten
    - wesentliche Technologien in der Middle-Tier-Infrastruktur: EJBs, Servlets/JSPs
    - plattformunabhängig
  - .NET
    - Familie von Produkten
    - sprachunabhängig(er als J2EE)
    - plattformabhängig (Windows)
  - bislang zu wenige Erfahrungen zur Beurteilung hinsichtlich initial angeführter Anforderungen
    - Entwicklungskosten
    - Verwaltungs/Administrations-/Wartungskosten
    - Skalierbarkeit
    - Interoperabilität