Hochschule Darmstadt
Fachbereich Informatik

Daten- und Systemintegration

Legacy Software / Anwendungsintegration mit Enterprise JavaBeans (EJBs)

(v1.0)

Prof. Dr. F. Bühler

Vorlesung „ Daten- und Systemintegration“

Kapitel 3 Legacy Software / Anwendungsintegration mit Enterprise JavaBeans (EJBs)

3.1 Legacy Software

Überblick

Während der technologische Fortschritt der Informationstechnologie (IT) stetig voranschreitet, setzen viele Unternehmen „*technologisch veraltete Systeme*“ in ihrer Anwendungslandschaft ein.

In den meisten Fällen handelt es sich dabei um Unternehmenssoftware aus der Zeit der Mainframe-Anwendungen (z. B. IBM Mainframe zSeries), die weder eine grafische Benutzeroberfläche, noch aktuelle und bekannte Benutzerkonzepte zur Verfügung stellen.

Diese Software wird als Legacy Software (Altsysteme) bezeichnet.

3.1 Legacy Software

Definition/Begriffsklärung

Legacy Systeme sind in einem Bereich der Unternehmenssoftware existierende Anwendungen, die fest im Unternehmen verankert und historisch gewachsen sind. Sie wurden mit früheren Technologien (z. B. COBOL, CICS) umgesetzt und sind nicht oder nur schwer modifizierbar.

In vielen Fällen handelt es sich dabei um hostbasierte Mainframe Systeme, die sich von modernen Systemarchitekturen (zum Beispiel Microsoft .NET oder Java) deutlich unterscheiden und in der Regel monolithisch aufgebaut sind.

Nach [MALE-2006] wird eine Legacy System als *„ein soziotechnisches System, welches Legacysoftware enthält“* verstanden. Damit lässt sich ein solches System nicht losgelöst und ohne Kenntnis der dazugehörigen Organisation und der enthaltenen Geschäftsprozesse analysieren.

Jedes Software-System, das operativ betrieben wird, ist somit ein Kandidat für Legacy Software.

[MALE-2006]
Masak, Dieter: *Legacysoftware: Das lange Leben der Altsysteme*, Springer Verlag, 2006.

3.1 Legacy Software

Die „harte“ Wirklichkeit

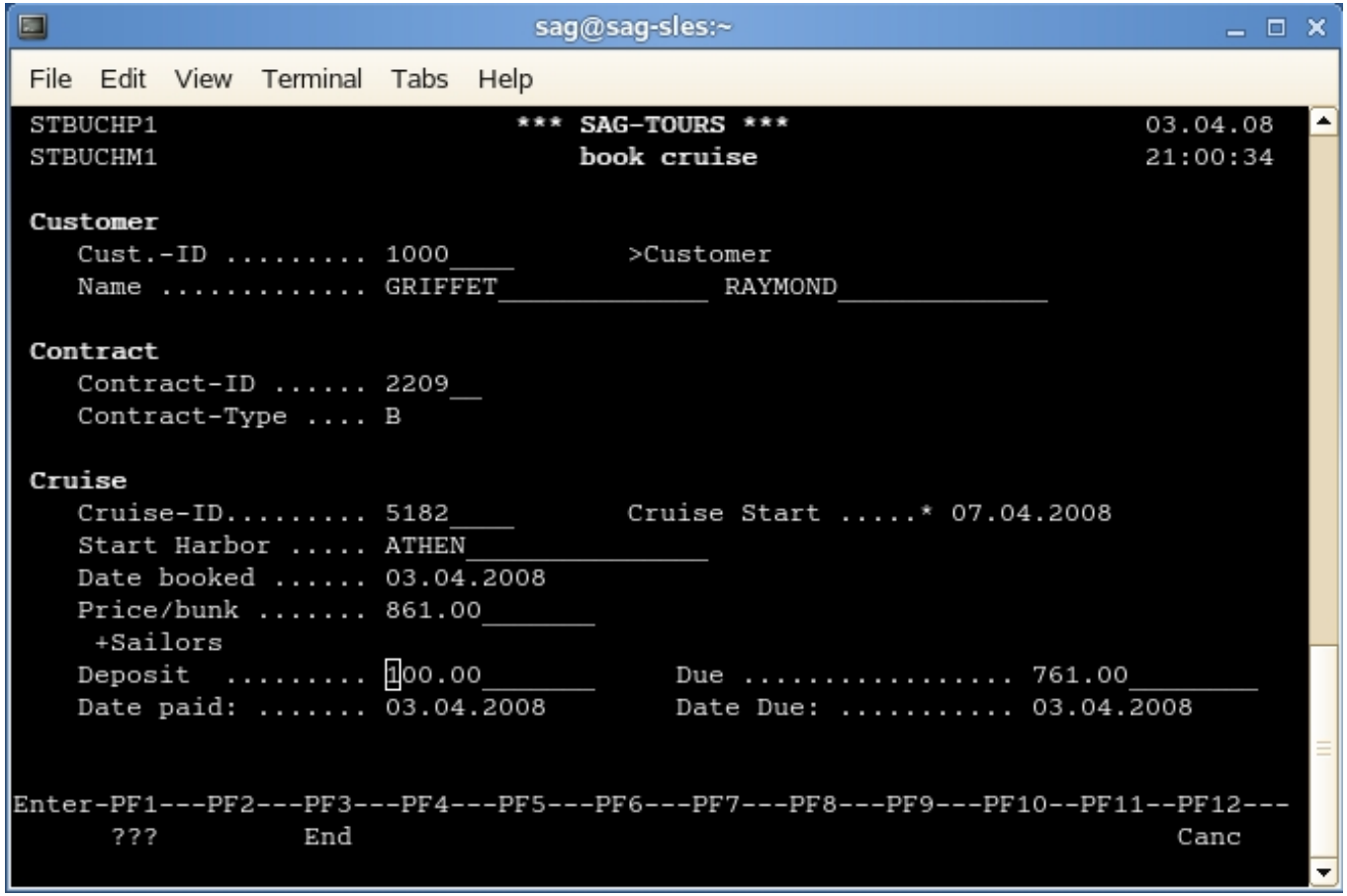
’More than 90% of enterprise applications in production are monolithic. [They are]... not externally accessible in a modular form that allows easy reuse in other applications.’

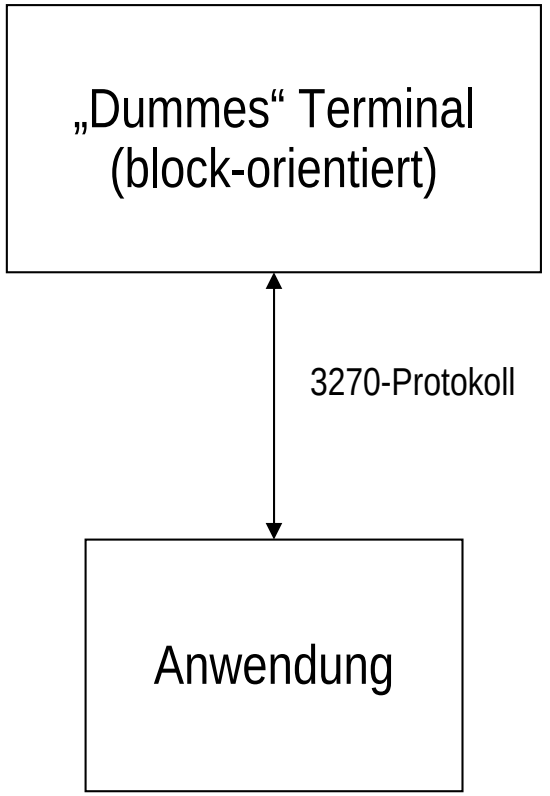
[GARTNER-2003]
Gartner, The Agile Enterprise, Oktober 2003.

Im Rahmen von „Legacy Modernisierungen“ wird versucht, diese Anwendungen in neuer Form (z. B. über Web-Frontends) weiter zu nutzen.

3.1 Legacy Software

Beispiel: Terminal-Anwendung „SAG-Tours“ (Software AG)

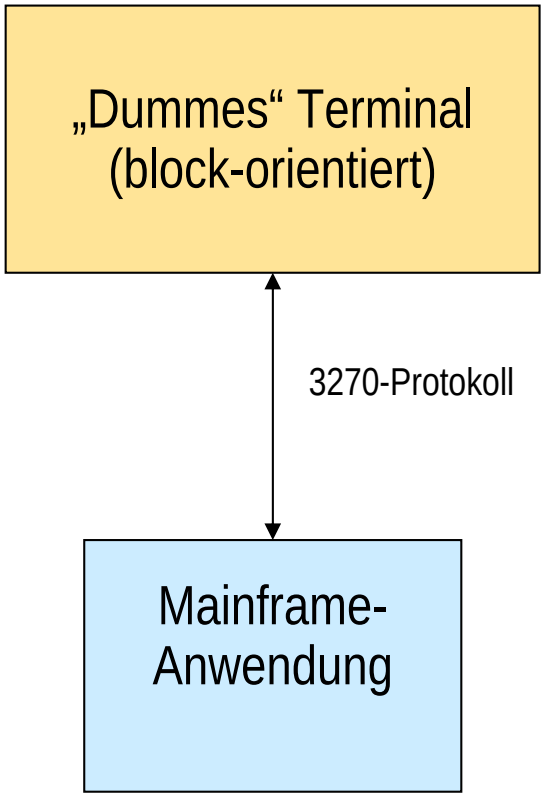




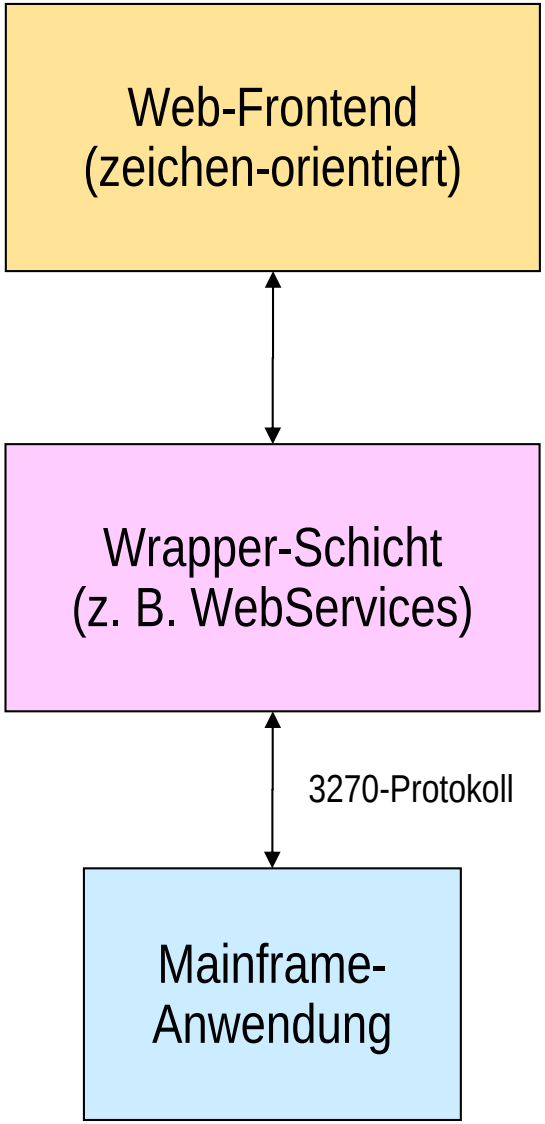
Client-/Server-Architektur

3.1 Legacy Software

Oberflächenintegration



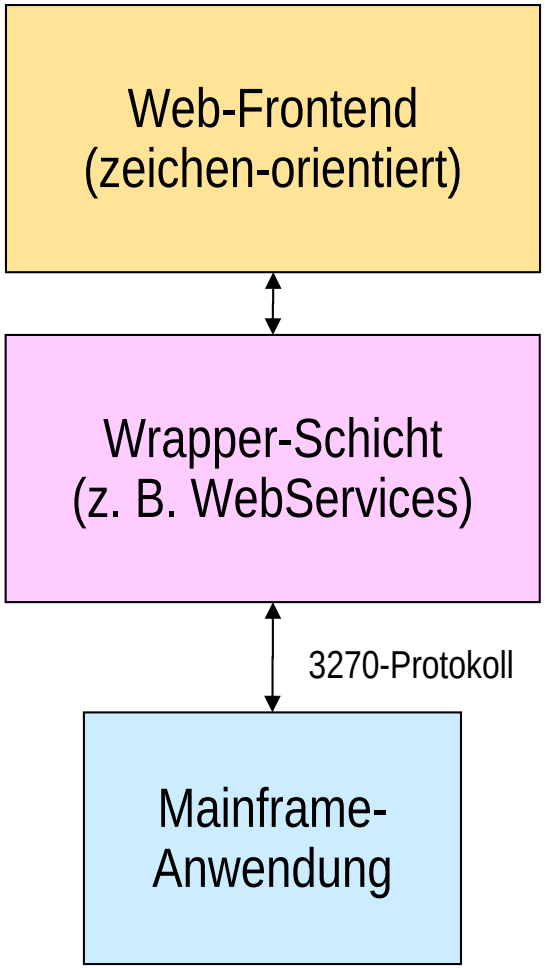
Client-/Server-Architektur



Mehrschichtige Architektur

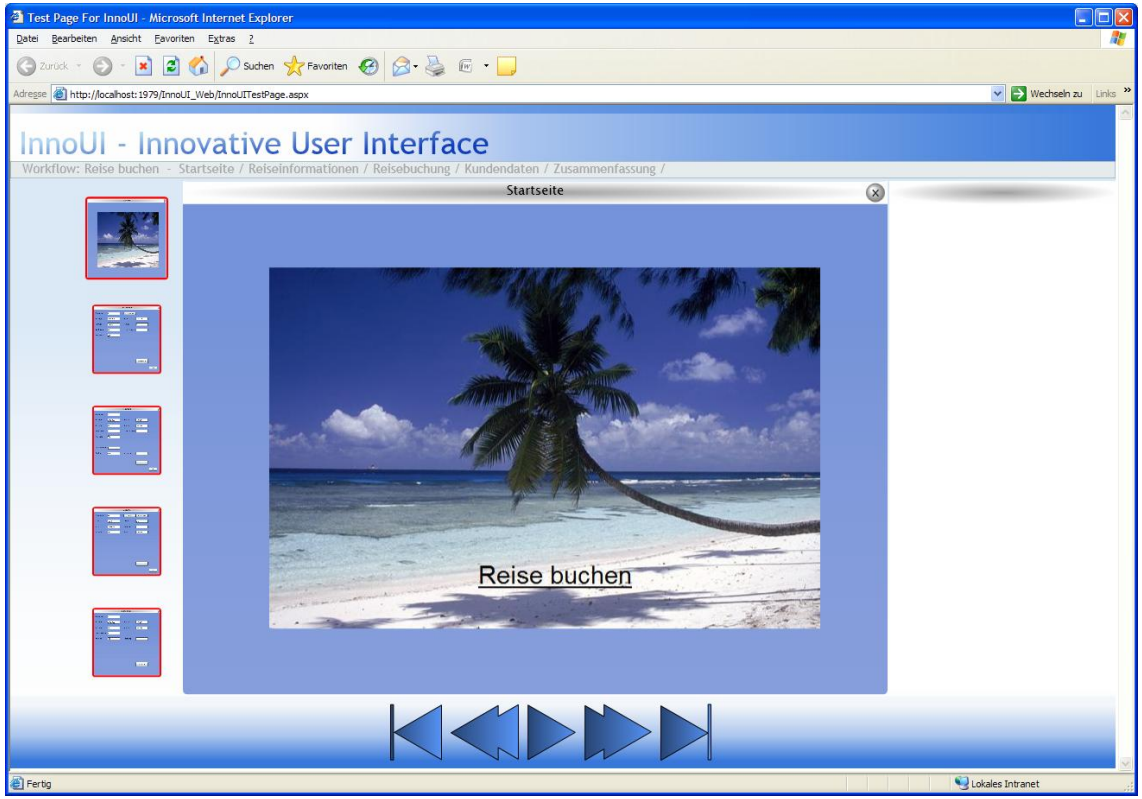
3.1 Legacy Software

Oberflächenintegration



Mehrschichtige Architektur

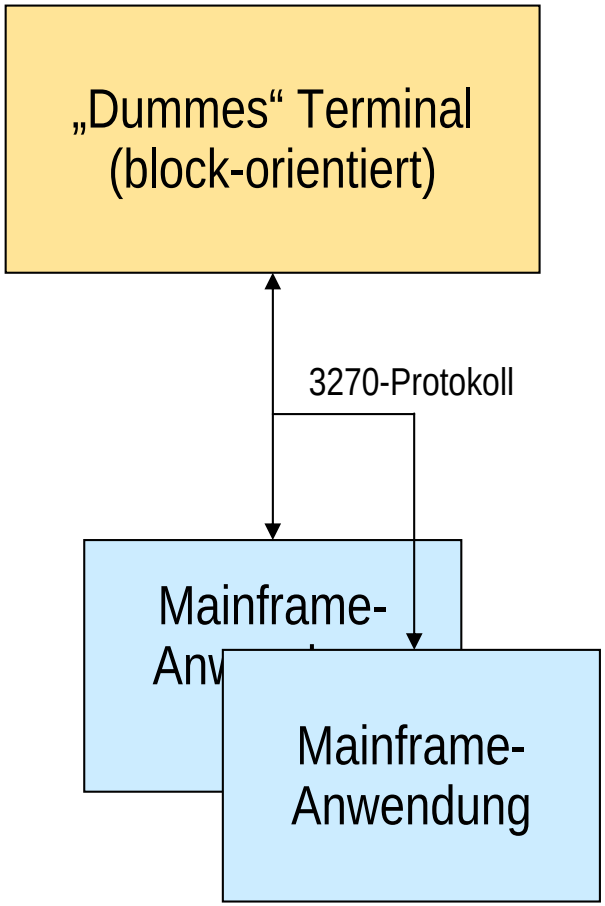
Neue Web-Oberfläche: RIA-Client



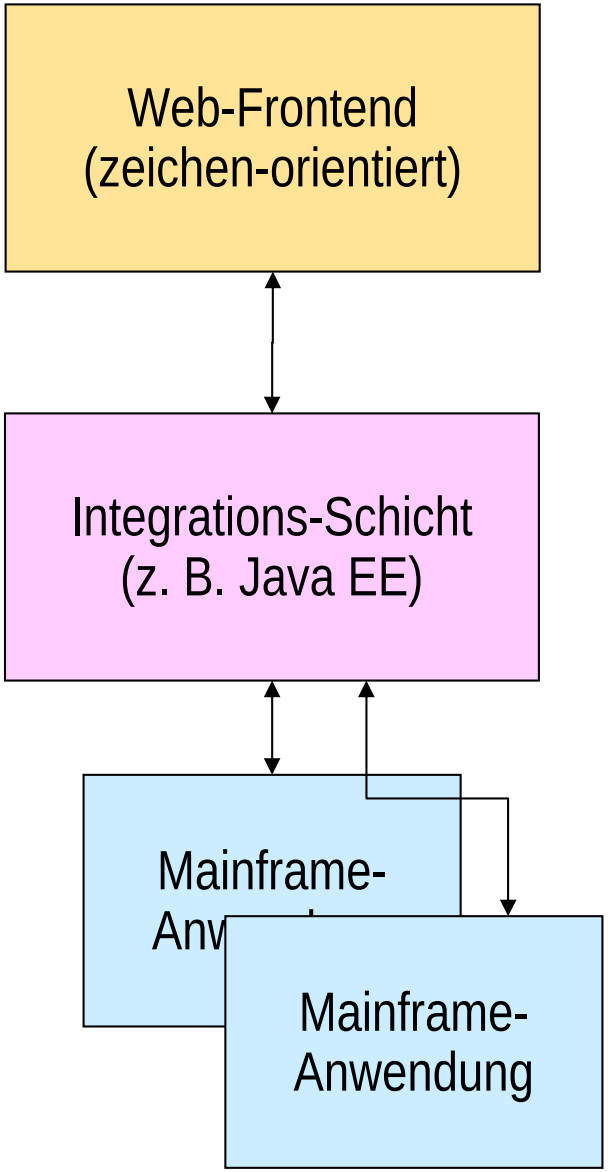
Masterarbeit: Alen Lazarov

3.1 Legacy Software

Funktionsintegration



Client-/Server-Architektur



Mehrschichtige Architektur

Vorlesung „ Daten- und Systemintegration“

Kapitel 3 Legacy Software / Anwendungsintegration mit Enterprise JavaBeans (EJBs)

3.2 Einstieg/Überblick Java-Plattform

Java-Plattform im Überblick

- Java Sprache, Java Virtual Machine (JVM)

- Entwicklungswerkzeuge und APIs (java, javac, javadoc, jar, Security, IDL, …)

- Bibliotheken (Beans, I/O, Logging, Concurrency, Reflection, ..)

- Benutzerinterface (AWT, Swing, Java 2D, Sound, …)

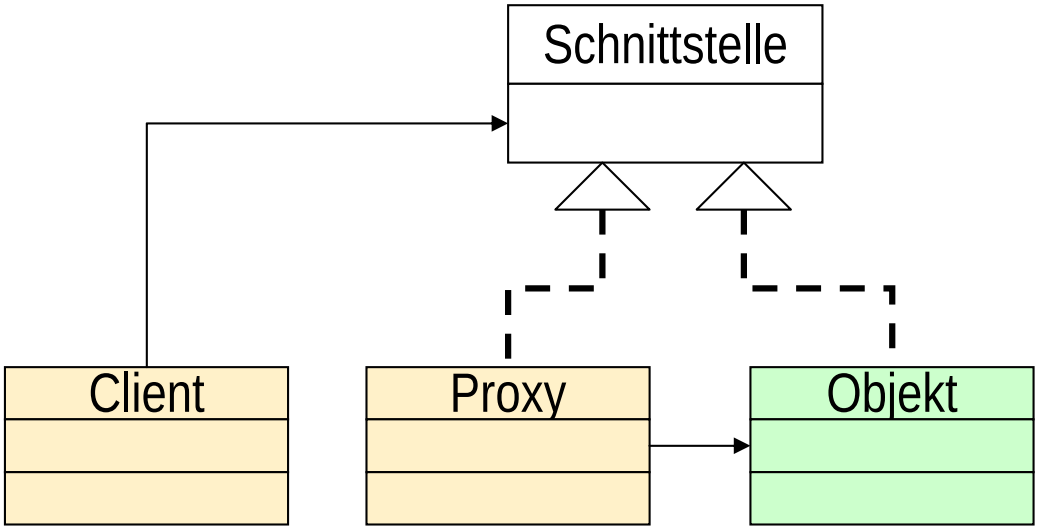
- Integrationstechnologie (IDL, JDBC, JNDI, RMI, RMI-IIOP)

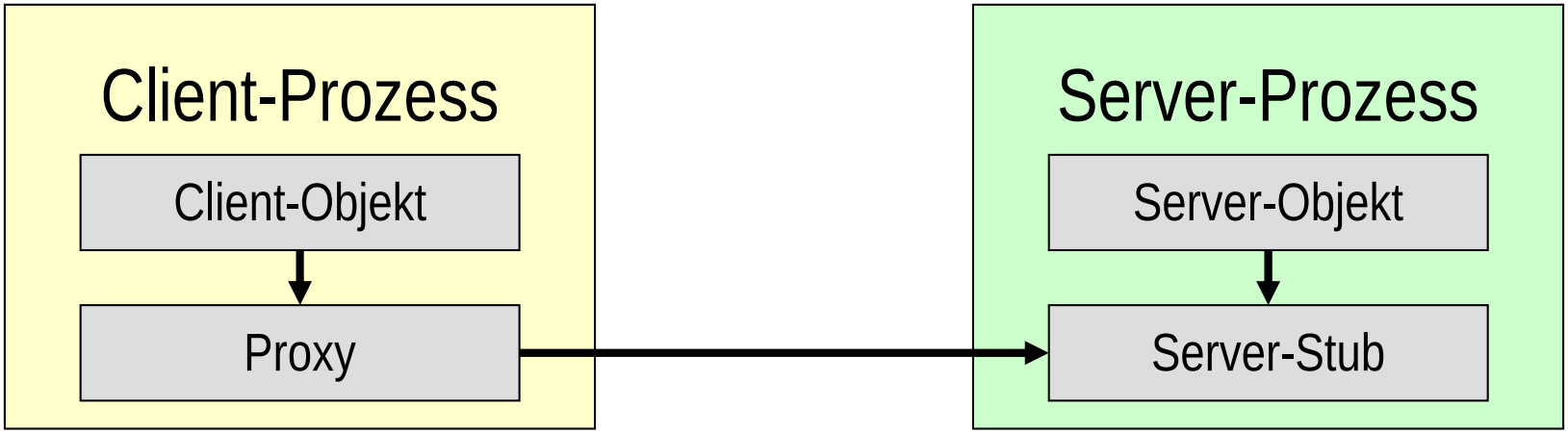
==> Mittels Java EE können unterschiedlichste Integrationsszenarien realisiert werden.

3.2 Einstieg/Überblick Java-Plattform

RMI (Remote Method Invocation)

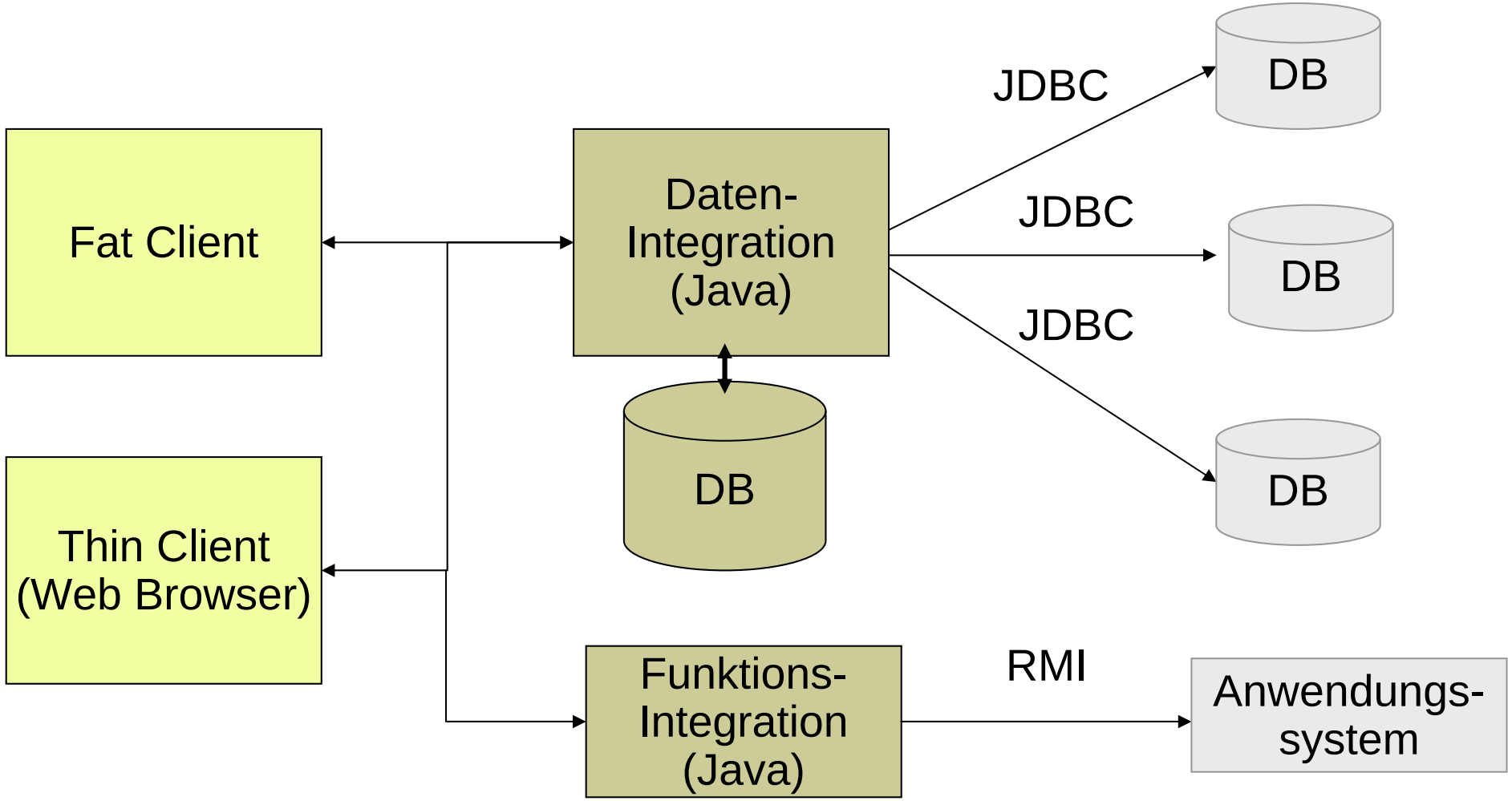
Aufruf von Methoden auf einem entfernten Objekt nach dem Anfrage/Antwort-Prinzip
(-> Proxy Pattern).





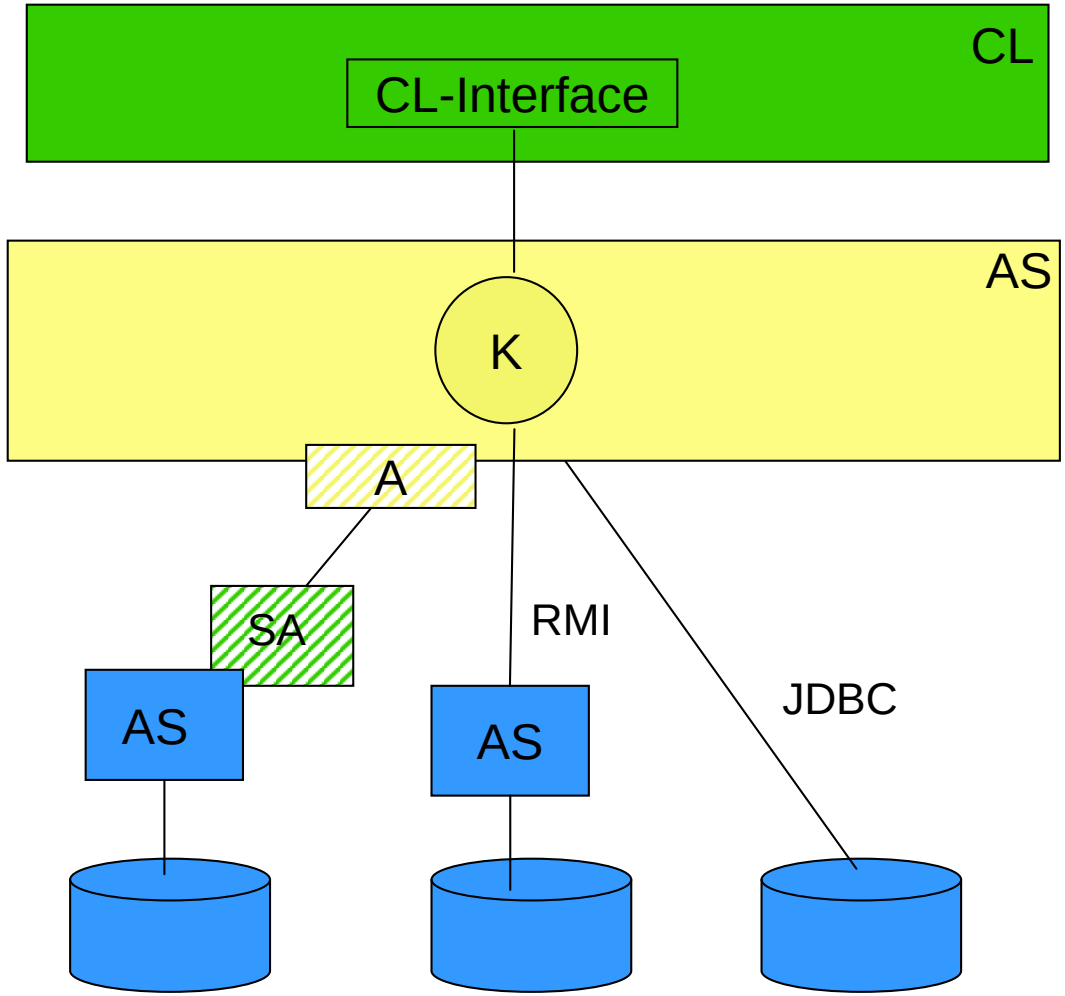
3.2 Einstieg/Überblick Java-Plattform

RMI (Remote Method Invovation) / JDBC (Java Database Connectivity)



3.2 Einstieg/Überblick Java-Plattform

Integrationsarchitektur auf Basis von Jave EE



CL: Client

AS: (Web) Application Server

K: Business-Komponente (Java EE/EJB)

A: Adapter

SA: Spezifischer Ressourcen-Adapter

AS: (Backend-) Anwendungssysteme

(Datenbanken, Legacy-Systeme, ERP, …)

3.2 Einstieg/Überblick Java-Plattform

Ziele der EJB-Architektur

- Reduzierung der Komplexität (Vereinfachung) der Entwicklung von Enterprise Applikationen, da Anwendungsentwickler nicht “Low-level Transaktionen” und “State Management Details”, “Multi-Threading”, “Ressourcen-Pooling” und andere komplexe Low-level APIs verstehen müssen.
- Erfahrene, professionelle Entwickler können jedoch auch direkt auf Low-Level APIs zugreifen.
- Vollständige Definition einer Umgebung, die die Entwicklung und Verteilung sowie Laufzeitaspekte berücksichtigt
  (-> „Enterprise Application‘s Life Cycle“).

3.2 Einstieg/Überblick Java-Plattform

Die EJB-Architektur ist ein wesentlicher Bestandteil von Java EE

Integriert sind verschiedene Java-Standards, z. B.
- JTS (Java Transaction Service)
- JNDI (Java Naming and Directory)
- JDBC (Java Database Connectivity)
- JMS (Java Messaging Service)
- RMI (Remote Method Invocation)
- CORBA (Common Object Request Broker Architecture)

3.2 Einstieg/Überblick Java-Plattform

Hauptprinzipien

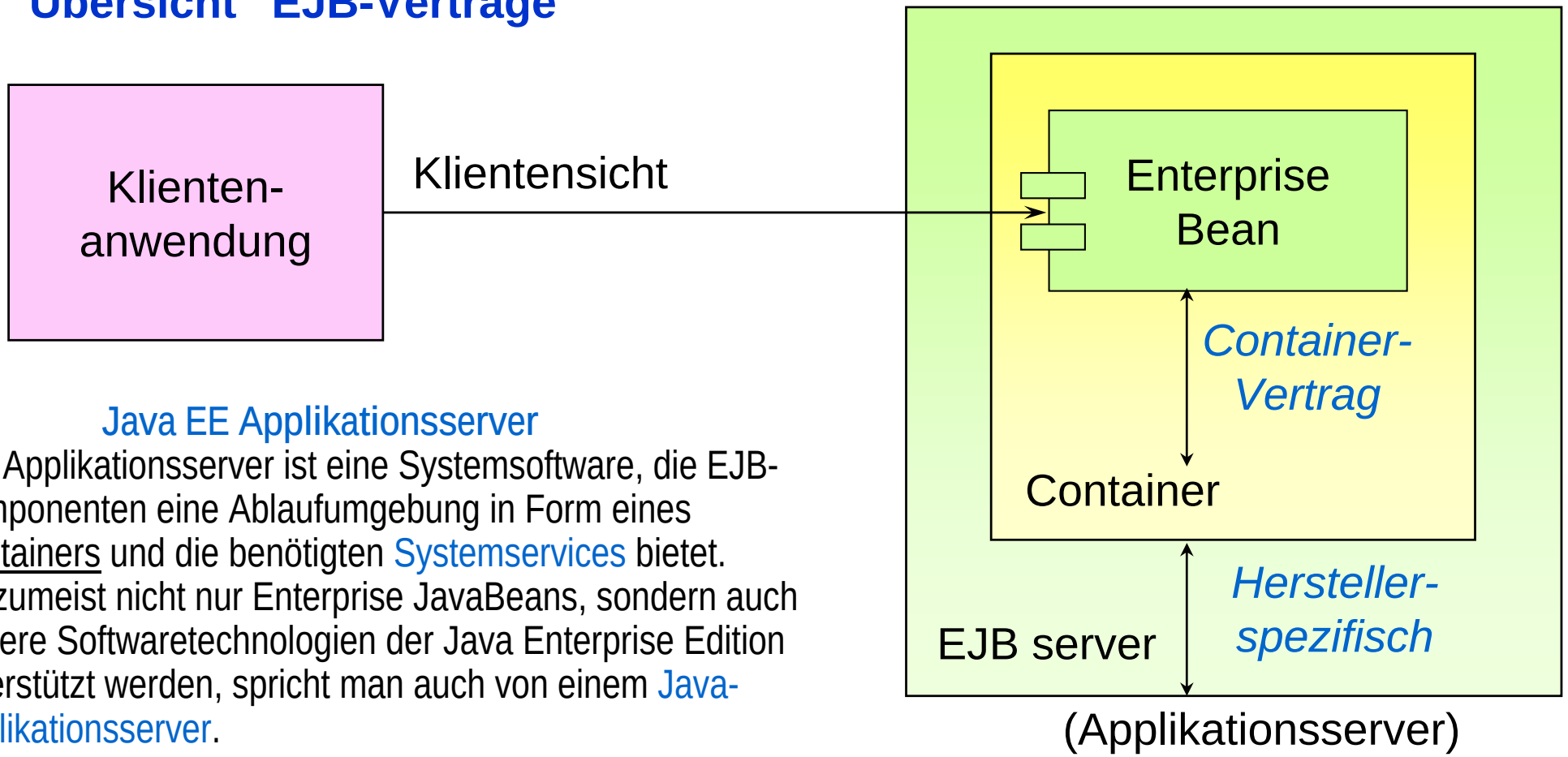
- **Trennung technischer, funktionaler Belange** („separation of concerns“)
  Technisch: Verteilung, Nebenläufigkeit, Ressourcen u.a.
  Funktional: fachliche Anforderungen mittels Geschäftskomponenten realisiert

- **Komponenteninfrastruktur**
  Die Infrastruktur (Container) realisiert die Implementierung technischer Belange.
  Anwendung: Geschäftsanwendungen mit speziellen Anforderungen im Hinblick auf Sicherheit, Persistenz und Transaktionen, Ausfallsicherheit, Lastverteilung, Skalierbarkeit.

EJB-Komponenten besitzen keinerlei GUI-Funktionalität (= sie sind “unsichtbar”)
und benötigen eine spezielle Laufzeitumgebung (-> EJB-Applikations-Server).

3.2 Einstieg/Überblick Java-Plattform

Übersicht “EJB-Verträge”

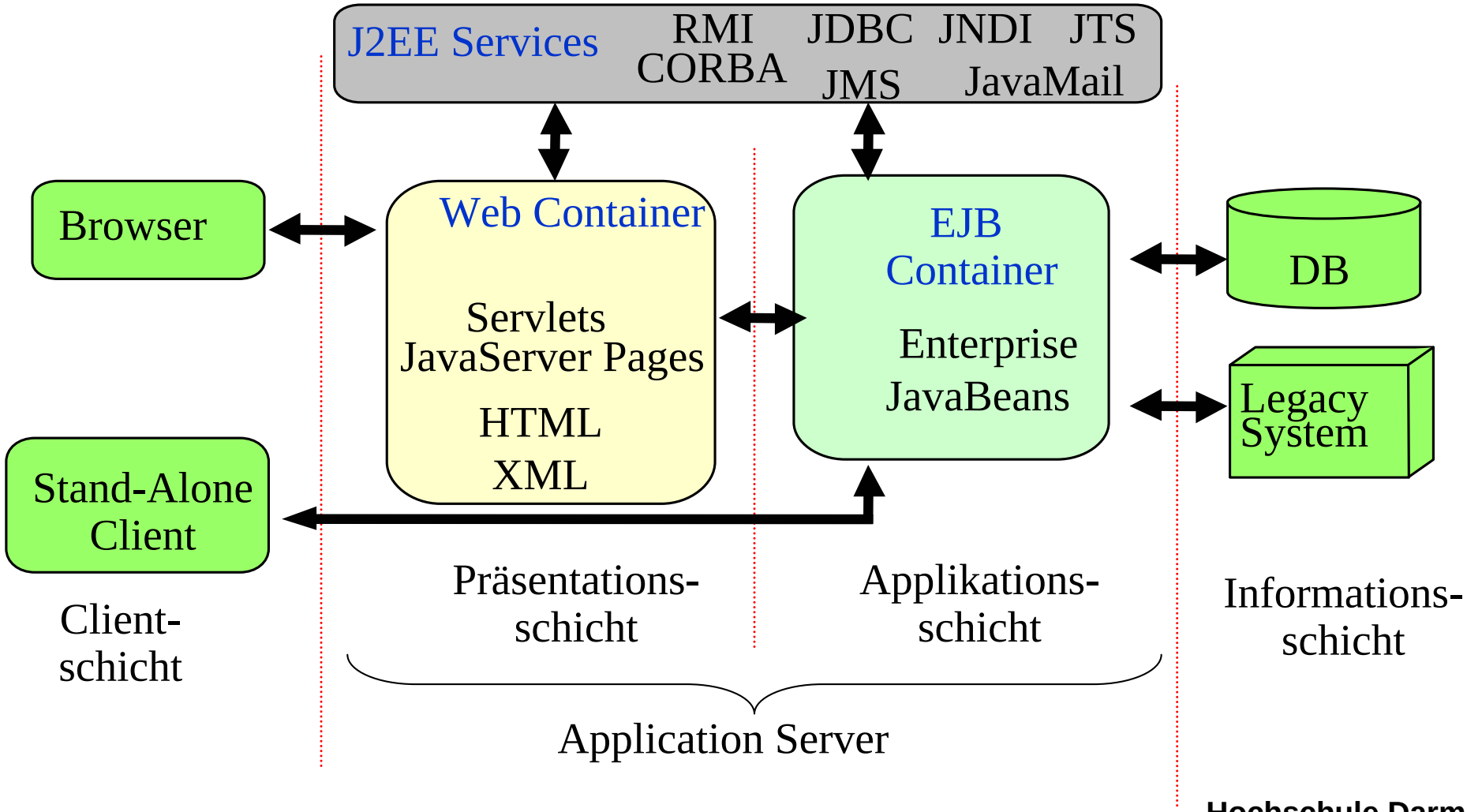


Java EE Applikationsserver

Der Applikationsserver ist eine Systemsoftware, die EJB-Komponenten eine Ablaufumgebung in Form eines Containers und die benötigten Systemservices bietet.
Da zumeist nicht nur Enterprise JavaBeans, sondern auch weitere Softwaretechnologien der Java Enterprise Edition unterstützt werden, spricht man auch von einem Java-Applikationsserver.
Zumeist wird die Einheit von „EJB–Server und EJB–Container“ als Applikationsserver bezeichnet.

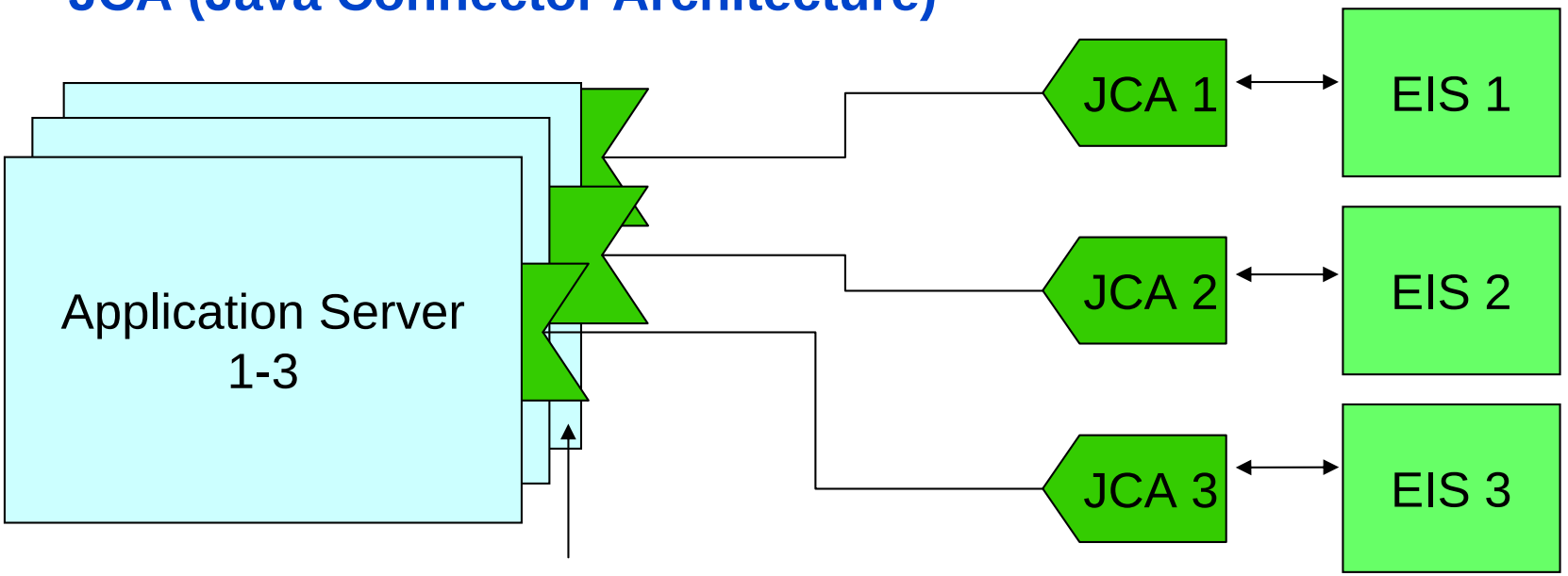
3.2 Einstieg/Überblick Java-Plattform

Einordnung der EJB–Architektur



3.2 Einstieg/Überblick Java-Plattform

JCA (Java Connector Architecture)



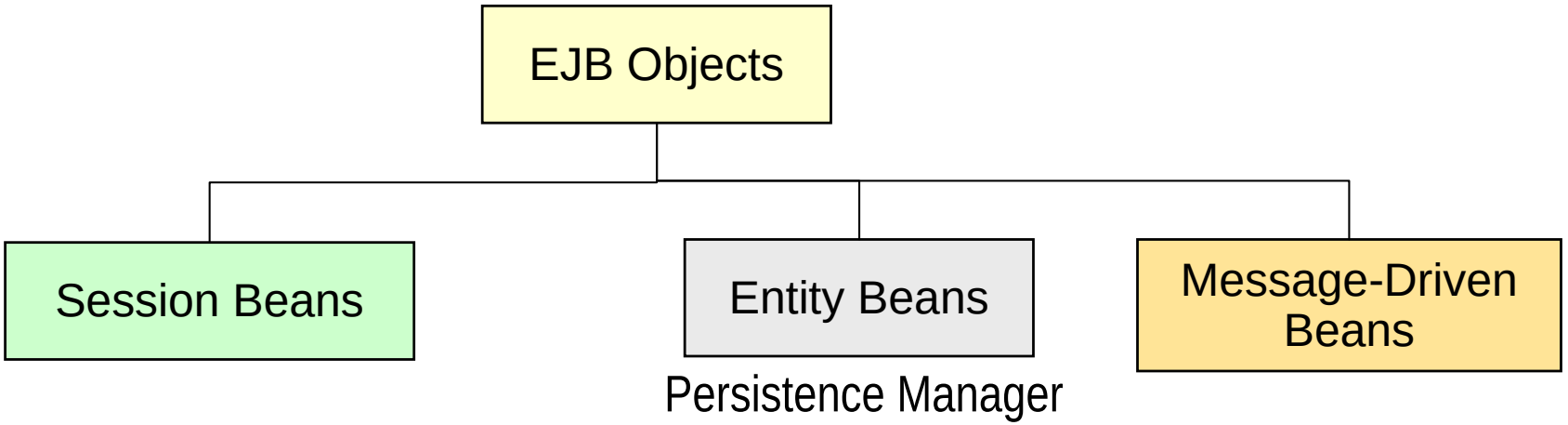
Erweiterung für die Einbindung von Adaptern

Die Java Connector Architektur wird verwendet, um Resource-Adapter zu erzeugen, die den Zugriff auf Enterprise Information Systems (EIS) ermöglichen.

Beispiel: Zugriff auf eine CICS/COBOL-Anwendung über EJB-Schicht.

3.2 Überblick EJB-Entwicklung

EJB-Typen

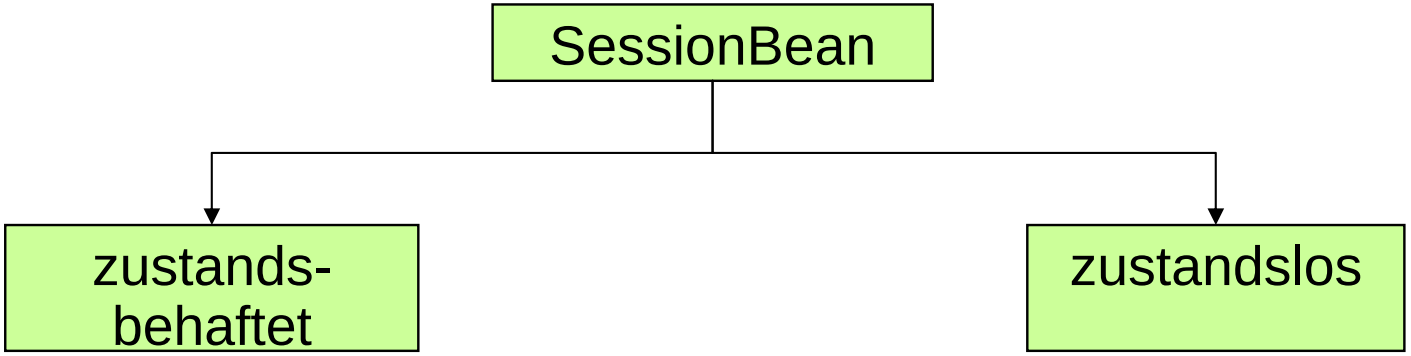


Vorlesung „ Daten- und Systemintegration“

Kapitel 3 Legacy Software / Anwendungsintegration mit Enterprise JavaBeans (EJBs)

3.3 SessionBeans

Es gibt zwei verschiedene Varianten einer Session Bean. Sie unterscheiden sich in der Fähigkeit, den Zustand ihrer Attribute zu erhalten.



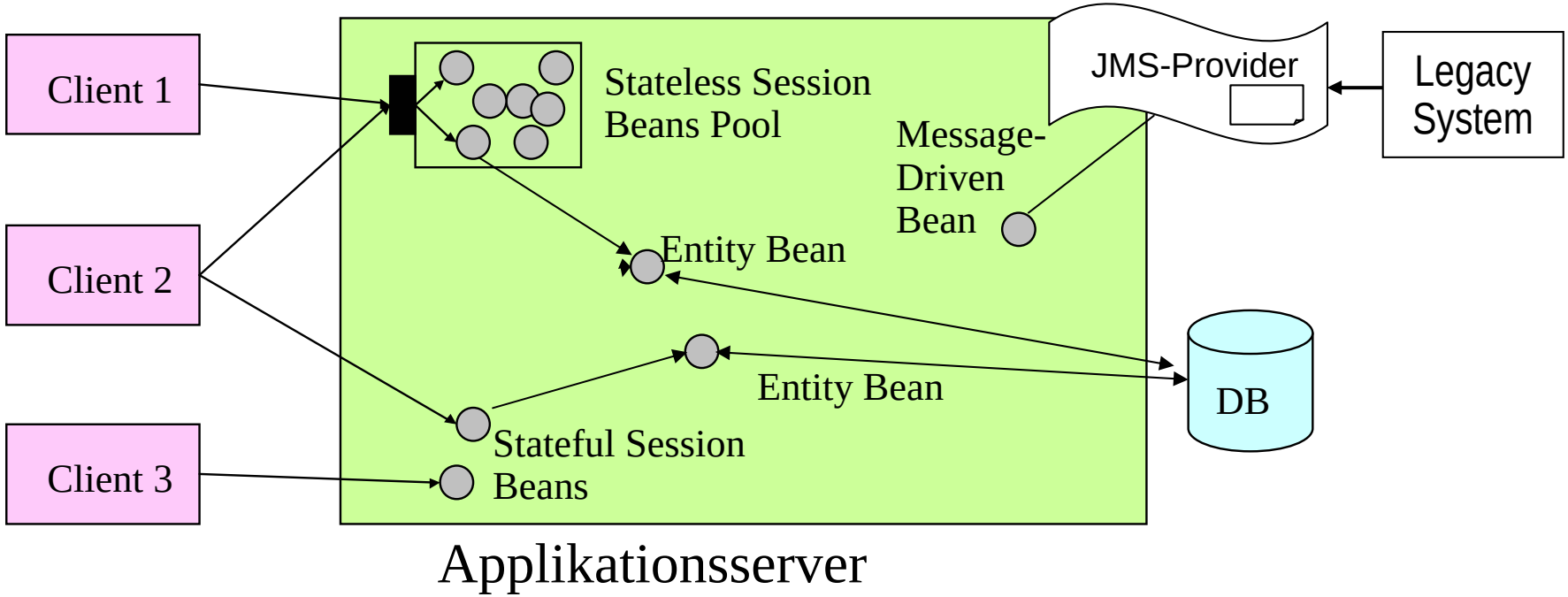
Zuordnung zu Client-Session bis diese wieder freigegeben wird. Keine andere Clients können auf diese Instanz zugreifen (-> exklusiver Zugriff).

Zuordnung für Methoden-aufrufe des Clients, keine Speicherung von Daten möglich. Objekte werden dynamisch aus Pool ge-nommen.

3.3 SessionBeans

Session Beans

Session Beans repräsentieren die Geschäftsprozesse in einem Anwendungssystem. Sie dienen der Implementierung der Anwendungslogik. Eine Session Bean ist geeignet, eine durch einen Benutzer ausgelöste Verarbeitung im System durchzuführen. Es gibt zwei verschiedene Varianten einer Session Bean. Sie unterscheiden sich in der Fähigkeit, den Zustand ihrer Attribute zu erhalten.



EJB Implementierung – SessionBean

- Zuerst müssen die benötigten SessionBeans und deren Typ (stateless oder stateful) analysiert werden.

- Im nächsten Schritt müssen die Business-Methoden ermittelt und implementiert werden (Remote (Component) Interface). Die Bean-Klasse entspricht weitestgehend einer gewöhnlichen Java-Klasse und dient der Programmierung individueller Anwendungslogik.

- Zusätzlich zu den Business Methoden kann diese Bean-Klasse auch noch einige Lifecycle-Methoden implementieren. Diese Methoden werden vom Container zu bestimmten Ereignissen im Lebenszyklus der Bean aufgerufen. Dazu gehören z. B. die Erzeugung und Zerstörung der Bean.

EJB Implementierung – SessionBean

Für jede Bean-Klasse müssen zwei Java-Interfaces vorhanden sein. Der Client kennt nur die Interfaces einer Bean und nicht deren Implementierung.

Remote Interface

... beschreibt die Sicht eines Clients auf die Bean-Komponente. Darin sind alle Business-Methoden, die der Client nutzen kann, definiert.

Home Interface

... stellt dem Client Methoden zur Erzeugung von Beans zur Verfügung.

Beispiel für Remote-Interface einer EJB-Komponente „Account“

```
import javax.ejb.*;

@Remote
public interface AccountRemote
{
    public float makeLodgement(float amount);

    public float makeWithdrawal(float amount)
            throws WithdrawalException;

}
```

Annotation für Remote Interface

Business-Methoden für Remote-Interface (-> Delegation der Operationsaufrufe zu Implementierungs-klasse)

Beispiel für Bean-Klasse einer EJB-Komponente „Account“

```
import javax.ejb.*;

@Stateless
public class Account implements SessionBean
{
    public AccountBean() {...}
    public float makeLodgement(float amount)
            throws RemoteException {...}

    public float makeWithdrawal(float amount)
            throws WithdrawalException {...}

}
```

beinhaltet u.a.
javax.ejb.SessionBean
javax.ejb.EntityBean

Implementierung der Business-Methoden

3.3 SessionBeans

Deployment Deskriptor

Um Beans verteilen zu können, wird vom Entwickler ein Deployment Deskriptor angelegt.

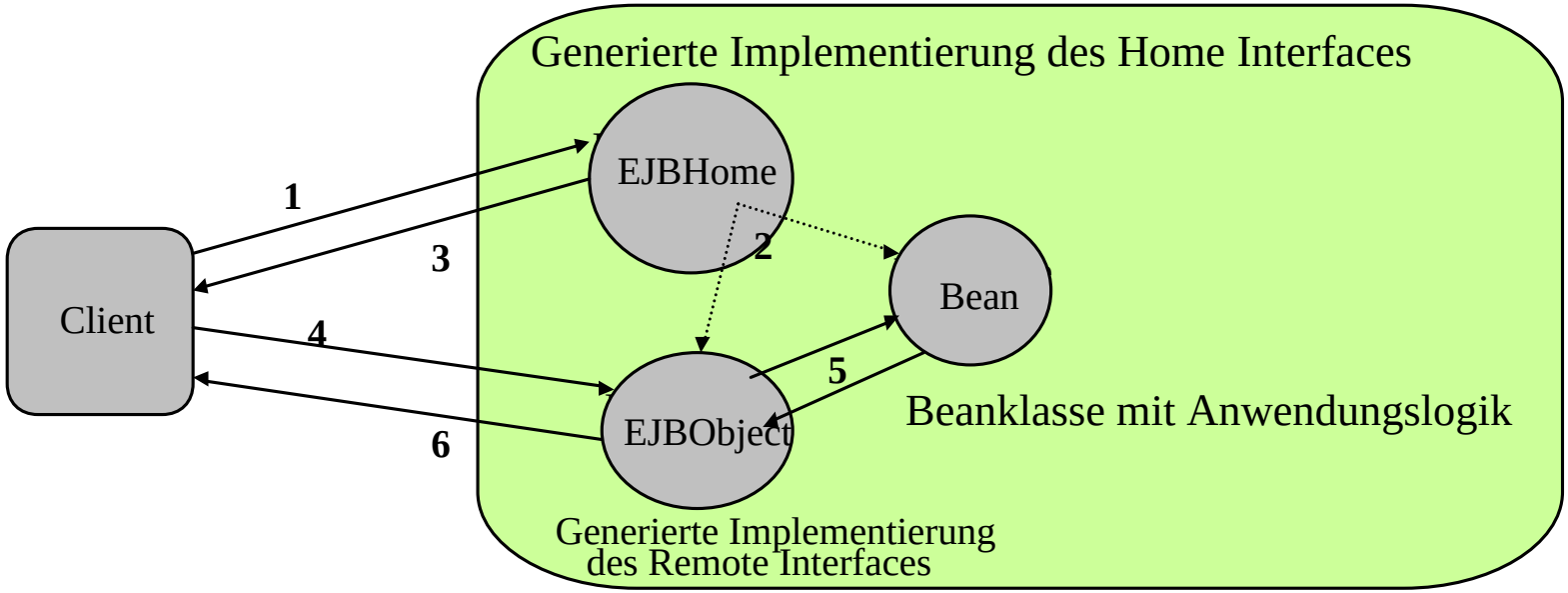
Da in Enterprise JavaBeans viele Komponenteneigenschaften nicht im Programmcode festgelegt werden müssen, sondern zur Zeit des Deployments beschrieben werden können, müssen diese in einem Deployment Deskriptor definiert werden.

Beim Deployment-Deskriptor handelt sich um eine Datei im XML Format. Darin werden u.a. Angaben zu dem Komponentennamen, den benötigten Transaktionen, dem Zugriffsschutz von Methoden und den verwendeten Ressourcen gemacht.

3.3 SessionBeans

EJB Implementierung

1) Client fordert eine Beaninstanz an.
2) EJBHome erzeugt ein EJBObject und assoziiert es mit der Bean.
3) Der Client bekommt eine Referenz vom Typ des Remote Interfaces auf das EJBObject zurückgeliefert.
4) Über das Remote Interface ruft der Client eine Business-Methode auf.
5) EJBObject führt Systemdienste aus. Anschließend leitet es den Aufruf an die Bean weiter und empfängt den Rückgabewert.
6) Der Client bekommt die Antwort der Bean übermittelt.



3.3 SessionBeans

Client-Zugriff auf SessionBean

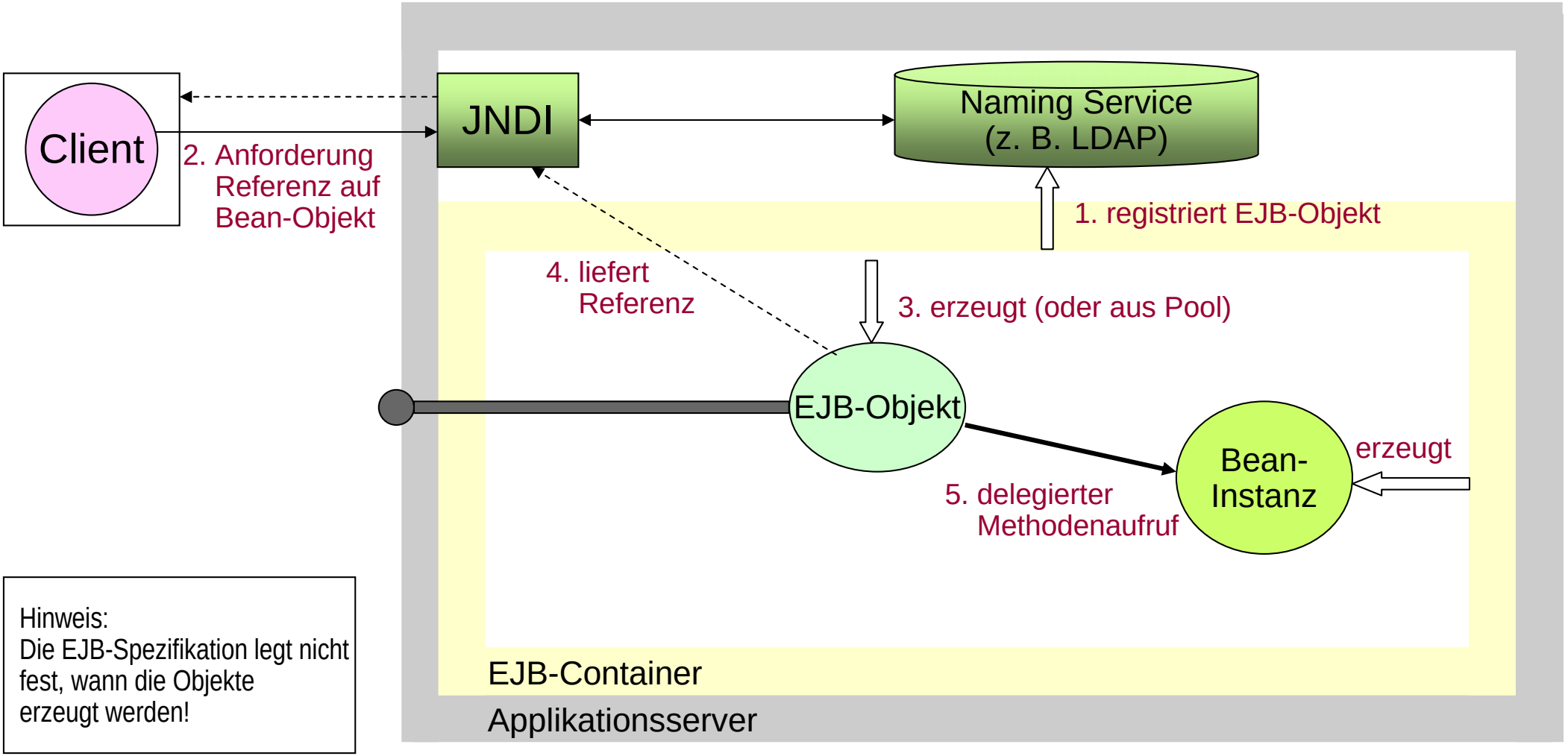
1. GUI-Eingaben bzw. Parameter für Taskflow ermitteln
2. RMI-Security-Manager initialisieren
3. JNDI-Context ermitteln und EJB-Referenz über JNDI holen
   (-> EJB-Object wird evtl. instanziiert und Client zugeordnet)
4. Operation(en) über ***Remote-Interface*** durchführen
5. EJB-Objekt freigeben (@Remove-Annotation für Methode)

3.3 SessionBeans

Beispiel: Client-Zugriff auf SessionBean

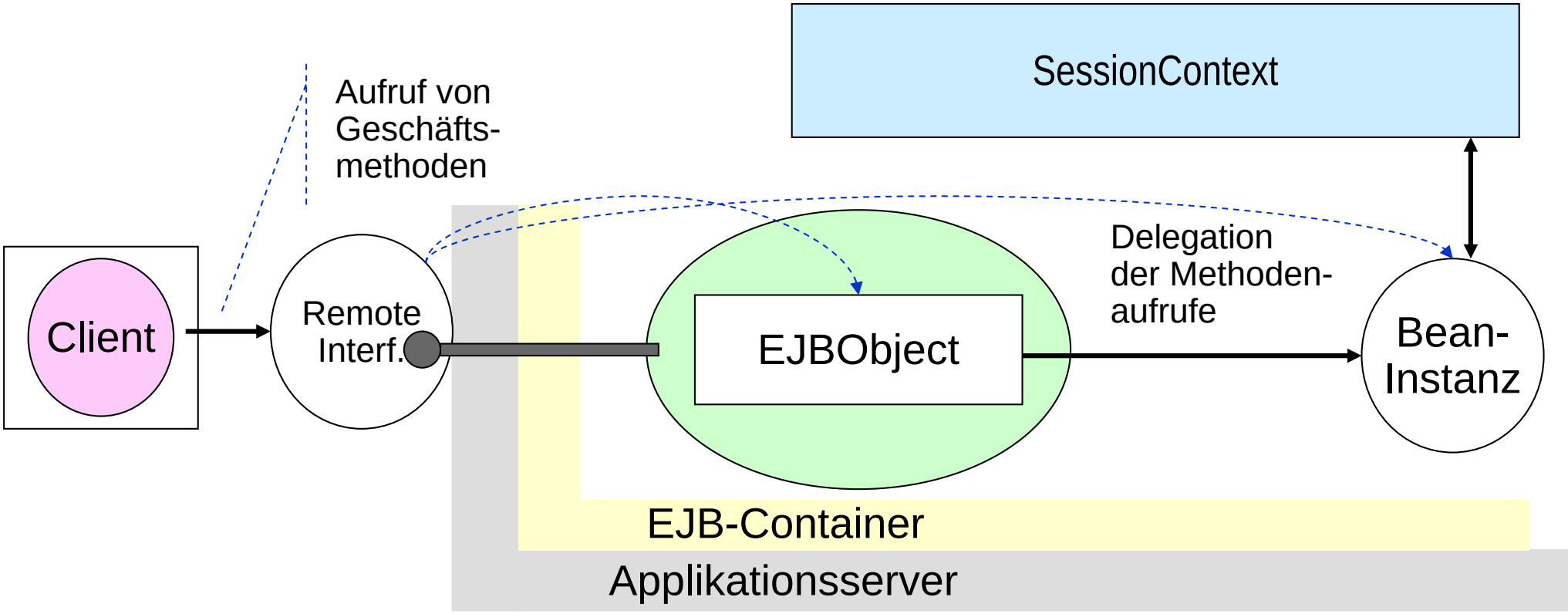
```
    // Verbindung zum JNDI Server aufbauen
    try {
            lContext = new InitialContext(lProperties);

            // Beans initialisieren
            xBean = (xBeanRemoteIF) lContext.lookup(xBean.JNDINAME_REMOTE);

    } catch (NamingException e) {
            logger.error("Fehler bei Verbindung zum JBoss JNDI Server aufgetreten");
            logger.error(e.getMessage());
            System.exit(0);

    } finally {
            if (xBean == null) {
               logger.error("Fehler bei Bean Deklaration");
            }
    }

    // Durchführung der Logik
   xBean.BusinessMethod(Parameter);
    }
}
```

Synchrones Aufrufmodell (Initialisierung) – 1(2)



Hinweis:
Die EJB-Spezifikation legt nicht fest, wann die Objekte erzeugt werden!

Synchrones Aufrufmodell (Remote Interface) – 2(2)



*Die Methoden des **Remote Interface** werden von EJBObject und der Bean-Klasse implementiert.*

3.3 SessionBeans

Grundsätzlicher Aufbau einer lauffähigen EJB-Komponente

Eine EJB-Komponente vom Typ SessionBean enthält folgende SW-Artefakte:

- Component Interface (öffentliche Schnittstelle für Client, fachliche Methoden, Trennung der Schnittstelle von Implementierung, zwei Varianten: **Remote Interface** über RMI/IIOP-Protokoll und Local Interface)
- WebService Endpoint Interface (nur bei WebService, Stateless SessionBean)
- Bean-Klasse (Implementierung der Geschäftslogik analog dem Component Interface, Aktivierung über entsprechenden Delegations-Mechanismus unter Einsatz des Deployment-Deskriptors)
- Deployment-Deskriptor

3.3 SessionBeans

Grundsätzlicher Ablauf für Entwicklung

1. SessionBean erzeugen
2. Applikationsserver starten
3. SessionBean deployen
4. Client-Anwendung entwickeln und ausführen
5. Client-Anwendung beenden
6. SessionBean weiter entwickeln
7. SessionBean „deployen“
8. Client-Anwendung evtl. anpassen und starten
9. Client-Anwendung beenden
10. Applikationsserver herunterfahren („shutdown“)

iterative Entwicklung

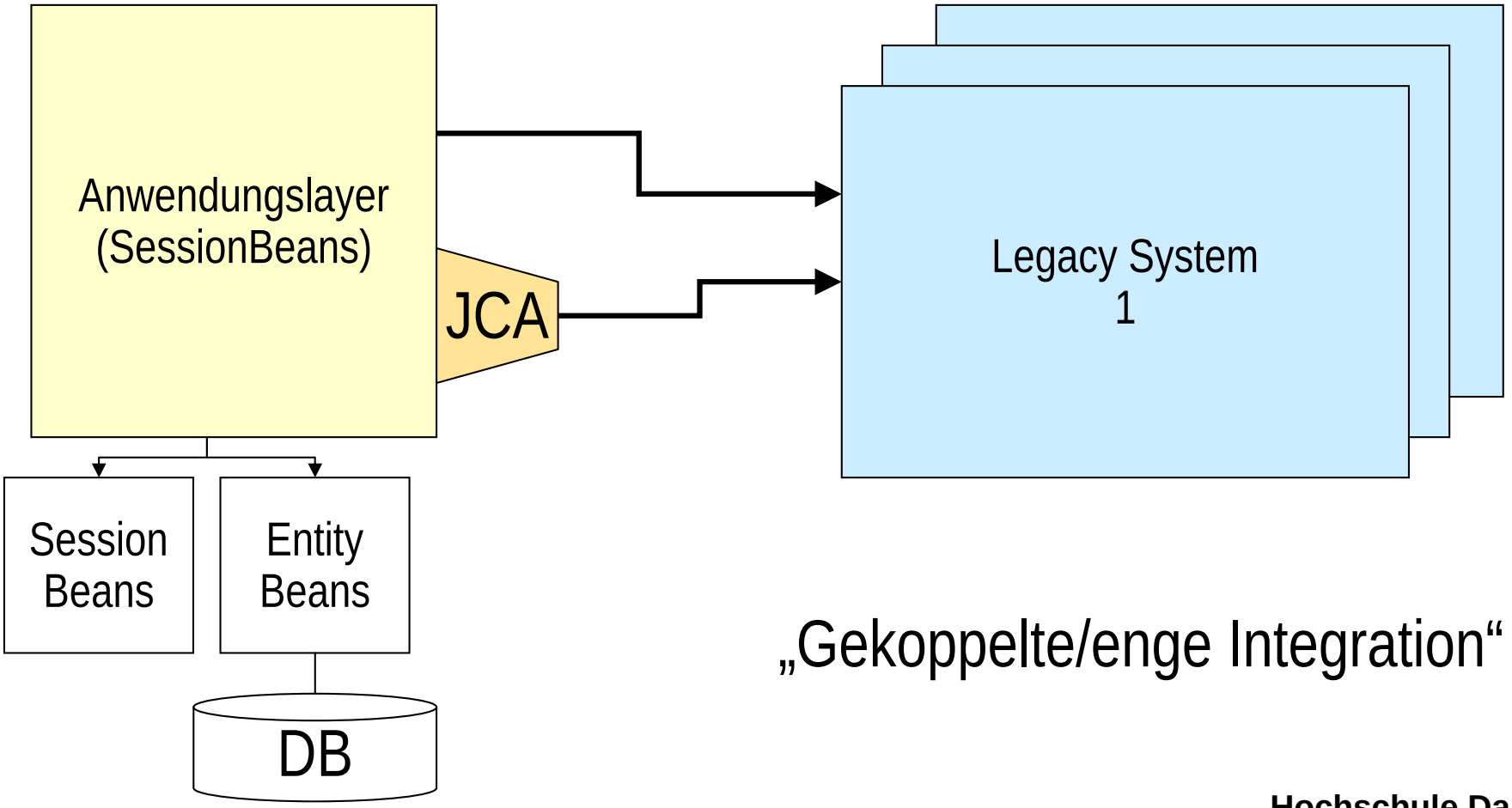
Hinweis: Je nach Applikationsserver muss evtl. dieser **vor** dem Deployen heruntergefahren und dann erneut gestartet werden. Moderne IDEs/Werkzeuge unterstützen den Deployment-Prozess sehr effizient.

Vorlesung „ Daten- und Systemintegration“

Kapitel 3 Legacy Software / Anwendungsintegration mit Enterprise JavaBeans (EJBs)

3.4 Zusammenfassung Integrationsszenarien

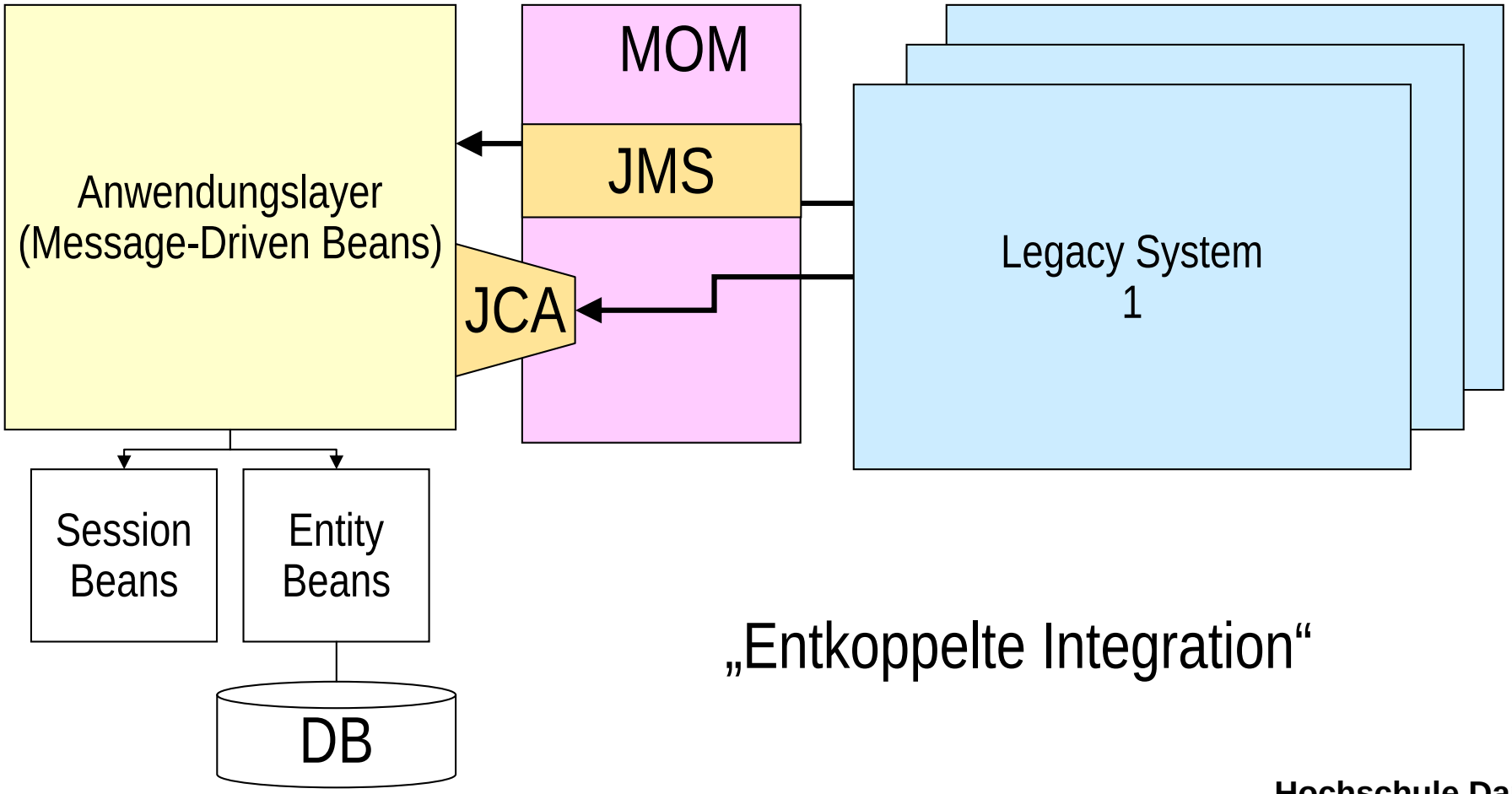
Funktionsintegration (synchron)



„Gekoppelte/enge Integration“

3.4 Zusammenfassung Integrationsszenarien

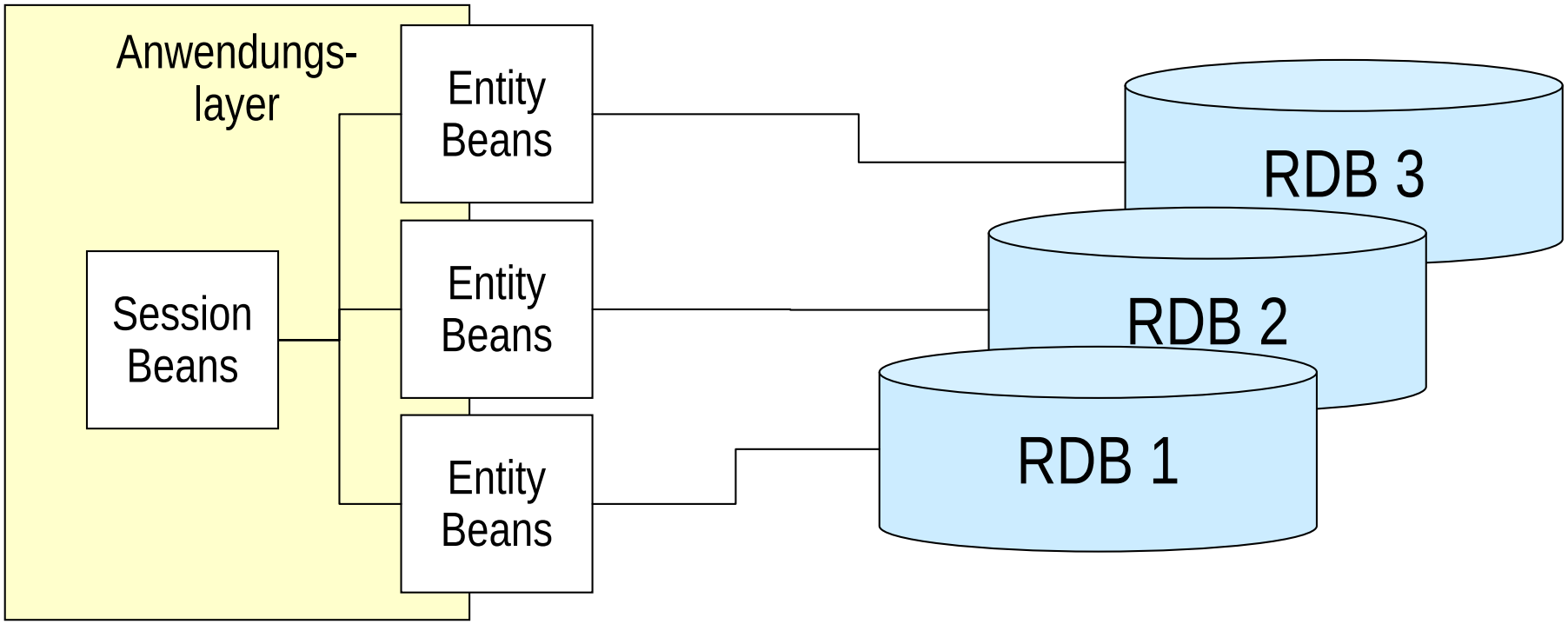
Funktionsintegration (asynchron)



„Entkoppelte Integration“

3.4 Zusammenfassung Integrationsszenarien

Datenintegration



„Object-Relational Mapping“

3.4 Zusammenfassung Integrationsszenarien

Referenzarchitektur für Integrationsprojekte

Fokusierung auf Java EE-konforme Architektur mit MVC-Programmiermodell