

Einführung in die Java Enterprise Edition

- Das Thema JEE ist durch eine große Anzahl von Ressourcen abgedeckt
 - Online
 - Oracle-Dokumentation
 - Tutorials und Blogs
 - Gedruckte Literatur
- Eine konkrete Empfehlung ist hier schwierig bis unmöglich
- Als Quellen wurden für diese Broschüre vorwiegend die JEE-Spezifikation von Oracle benutzt

- Dies ist ein Theorie-Seminar
 - Keine Programmier-Übungen
 - Statt dessen Diskussion, Analyse von fertigem Programm-Code, Demonstrationen
- Dokumentation und Ressourcen stehen im Internet zur Verfügung
 - Beispiele unter <https://GitHub.com/Javacream/org.javacream.training.jee7>
- Konventionen
 - Befehle werden in `Courier-Schriftart` dargestellt
 - Dateinamen werden in *kursiver Courier-Schriftart* dargestellt
 - Links werden in unterstrichener Courier-Schriftart dargestellt

© Javacream

Javacream

Dr. Rainer Sawitzki

Alois-Gilg-Weg 6

81373 München

eMail: training@rainer-sawitzki.de

Alle Rechte, einschließlich derjenigen des auszugsweisen Abdrucks, der fotomechanischen und elektronischen Wiedergabe vorbehalten.

Übersicht	6
Der Applikationsserver	22
Dienste des Applikationsservers	57
Programmierung von JEE Anwendungen	80
Das Programmiermodell	103
Java Server Faces	132
Datenbank-Zugriff	155
Architektur und Design	179

1

ÜBERSICHT

1.1

BESTANDTEILE

- Die JEE besteht aus:
 - Dem Applikationsserver
 - Dieser ist eine erweiterte Java Virtual Machine, die Dienste zur Verfügung stellt
 - Er enthält weitere Werkzeuge
 - Überwachung
 - Anwendungsinstallation bzw. Deployment
 - Administration
 - Einer Reihe von komplexen technischen Komponenten
 - Inbound und Outbound Connectors
 - Enterprise JavaBeans
 - Servlets

- Programmiermodellen für die Anwendungs-Entwicklung
 - Context & Dependency Injection für die Verwaltung der Fach-Klassen
 - JavaServer Faces für die Erstellung von Web-Anwendungen
 - Java Persistence API für Datenzugriffe
 - Verteilte Anwendungen
 - Scheduling und Batch Processing

- Connectors
 - Über Inbound Connectors kommunizieren entfernte Clients mit dem Applikationsserver
 - Sie öffnen einen abhörenden Netzwerk-Socket
 - Jeder ankommende Request wird der passenden Anwendung zugeordnet
 - Outbound Connectors binden externe Systeme an
 - Dazu baut der Outbound Connector Socket-Verbindungen auf, die in einem Connection Pool gehalten werden
 - Zusätzliche Inbound und Outbound Connectors können installiert werden

- Enterprise JavaBeans
 - Kümmern sich um Authentifizierung und Autorisierung
 - Realisieren horizontale und vertikale Skalierbarkeit
 - Können direkt durch gängige Kommunikationsprotokolle angesprochen werden
 - und machen damit die Anwendung verteilbar

Enterprise JavaBeans sind technische Komponenten, aber mittlerweile so einfach zu definieren, dass sie problemlos auch direkt im Anwendungs-programm benutzt werden können!

- **Servlets**
 - Kümmern sich um Authentifizierung und Autorisierung
 - Realisieren horizontale und vertikale Skalierbarkeit
 - Können durch das http/https-Protokolle angesprochen werden
 - Ursprünglich rein Request-Response-orientiert
 - Ab der JEE 7 auch asynchrone Aufrufe mit Server-Callbacks möglich
 - Unterstützung von Web Sockets

Servlets bieten ein mächtiges Low-Level-API und werden von Frameworks wie JavaServer Faces oder Web Services intern benutzt.

- Context & Dependency Injection (CDI)
 - Der Context erzeugt sämtliche relevante Fachklassen
 - Die Lebensdauer wird durch "Scopes" definiert
 - Setzt deren Abhängigkeiten
 - Macht die Fachlogik transaktionsfähig
 - Realisieren mit Interceptors Anwendungsfallübergreifende Querschnittsfunktionen
 - Aspekt-orientierte Programmierung
 - Stellt einen Event-Bus zur losen Kommunikation zur Verfügung

Mit CDI und Interceptors unterstützt die JEE die neuesten anerkannten Design-Regeln für die Anwendungsprogrammierung!

- **JavaServer Faces**
 - Bilden Browser-basierte Anwendungen in Komponenten ab
 - Abstrahieren und Kapseln die technischen Details der http-Technologie
 - Stellen mit einem Event-Modell, AJAX-Unterstützung und Navigationsregeln ein hochwertiges Programmier-Modell zur Verfügung
 - Erstellen mit Layouts Web-Seiten mit einheitlicher Gestaltung
 - Auch JSF-Komponenten sind automatisch CDI-Konform

Die direkte Verwendung eines Servlets zur Programmierung von Web Anwendungen ist in der Regel viel zu kompliziert!

- Java Persistence API (JPA)
 - Ein EntityManager verwaltet Entity-Objekte
 - Create-, Read-, Update-, Delete-Metoden
 - Synchronisation der Entity-Objekte mit der Datenbank-Einträgen
 - Objektorientierte Abfragen mit Objekt-orientiertem SQL und Criteria-Objekten
 - Auch direkte SQL-Befehle können ausgeführt werden

Das JPA macht die direkte Verwendung des JDBC-APIs `java.sql` überflüssig.

- Verteilte Anwendungen
 - Remote Method Invocation (RMI)
 - Notwendig ist nur ein Remote-kompatibles Interface
 - http/https
 - Servlet-API mit `HttpServletRequest`, `HttpServletResponse`, `HttpSession`, ...
 - Messaging
 - JMS-API
 - SOAP-basierte Web Services
 - JAX-WS
 - RESTful Web Services
 - JAX-RS
 - Anbindung an Mail-Server

- Scheduling
 - Definition von wiederkehrenden Aktionen mit Timer und TimerTask
- Java Batch
 - Ein komplettes Job-basiertes Framework zur Ausführung komplexer Batch-Anwendungen
 - Definition durch die Job Specification Language
 - eine XML-basierte Beschreibungssprache

1.2

SPEZIFIKATION UND HERSTELLER

- Eindeutige Definition des Laufzeitverhaltens und der zur Verfügung gestellten Dienste
- Entwickler und Architekten müssen für unterschiedliche Projekte/Kunden nur eine einzige Umgebung lernen
- JEE-Anwendungen laufen in allen JEE-konformen Applikationsserver
 - 100 % Pure Java läuft in "Java Compatible" Umgebungen
- Garantierte Abwärts-kompatibilität
- Herstellerunabhängigkeit garantiert fruchtbare Konkurrenz zwischen JEE-Anbietern

- Kommerziell
 - IBM WebSphere
 - Oracle WebLogic
 - ...
- Open Source
 - JBoss
 - Apache Geronimo bzw. IBM Developer
 - Oracle Glassfish
 - Apache Tomcat
 - ...

- JEE-Provider müssen einer formalen Spezifikation genügen
 - Damit ist die Frequenz von neuen Versionen und damit die Integration neuer Anforderungen und Features relativ gering
 - JEE 1.4 2001
 - JEE 5 2006 (!)
 - JEE 6 2009
 - JEE 7 2013
 - JEE 8 als JSR 366 in der Findungsphase
- Die Spezifikation erfolgt unter Mitarbeit verschiedener Hersteller und ist deshalb stets als Schnittmenge aktueller Technologien zu verstehen
 - Im Gegensatz dazu können unabhängige, Implementierungs-getriebene Frameworks jederzeit neue Features einbringen
 - Mit allen daraus resultierenden Vor- und Nachteilen!

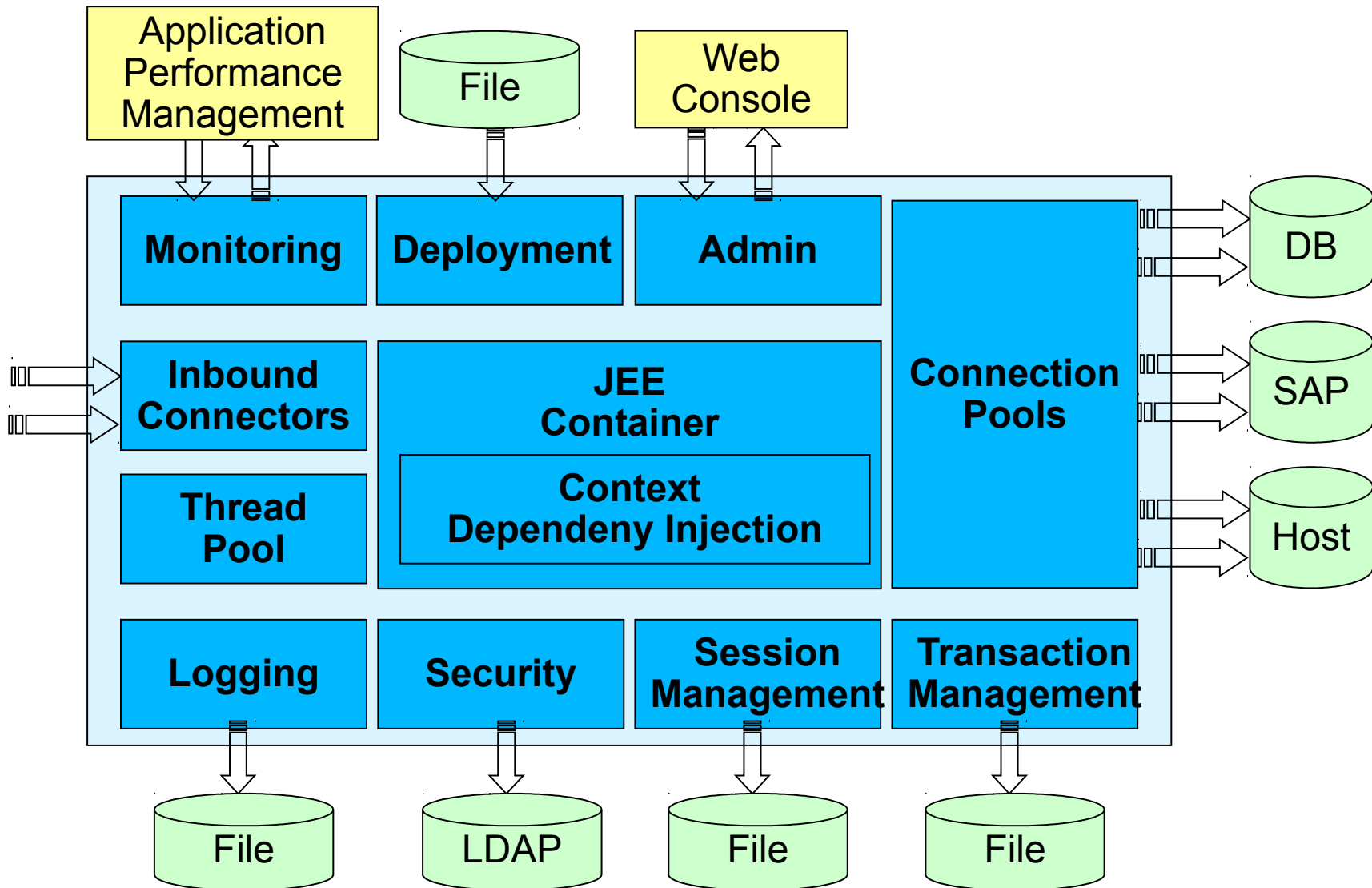
2

DER APPLIKATIONSSERVER

2.1

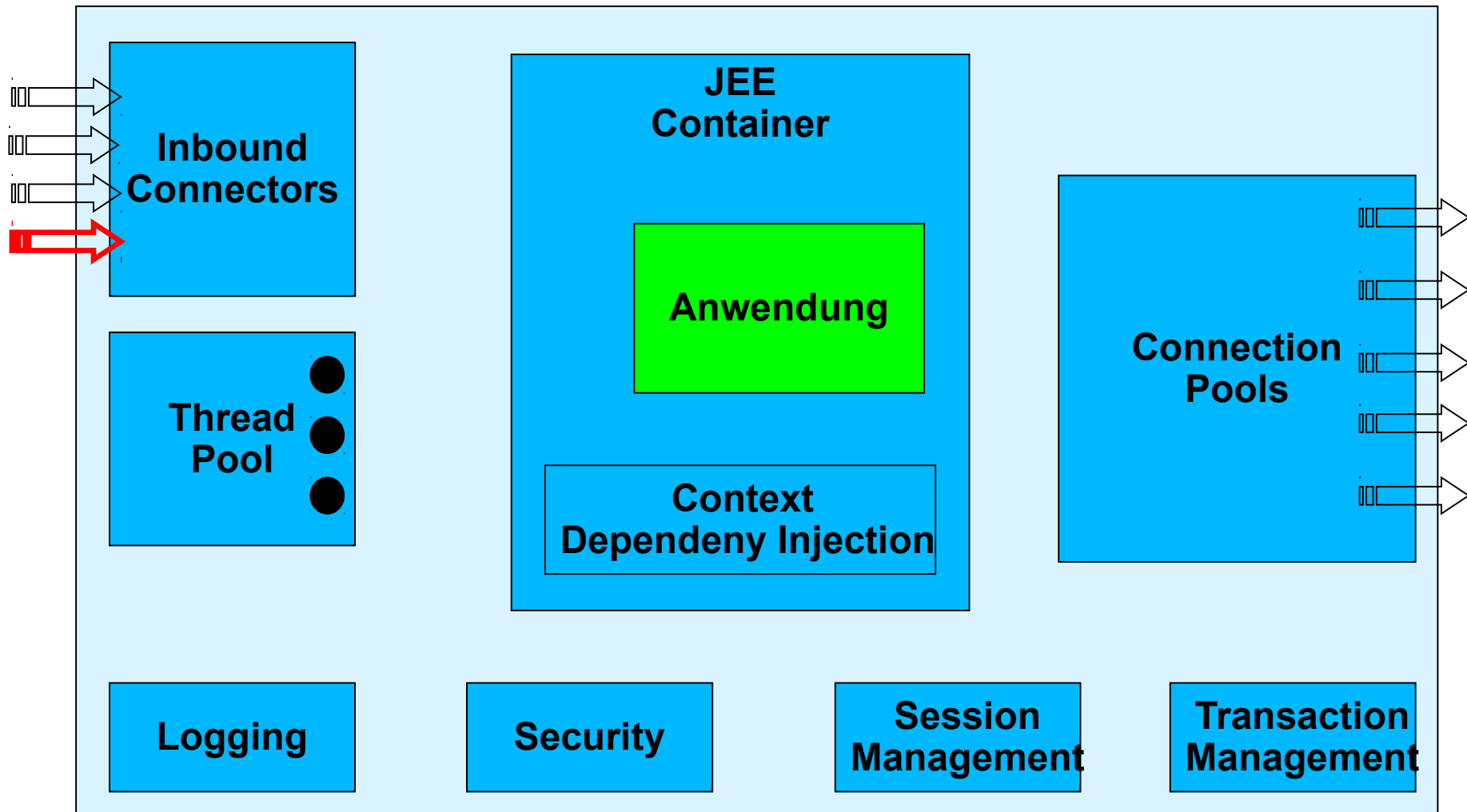
ARBEITSWEISE

- Die Funktionen, die ein Applikationsserver zur Verfügung stellen muss, sind im Rahmen der JEE-Spezifikation definiert
- Die genaue Umsetzung kann von Hersteller zu Hersteller differieren
 - Provider unterscheiden sich beispielsweise
 - im Grad der Modularität
 - in der Art der Konfiguration
 - in der Gestaltung von Administrations-Werkzeugen
- Auf Grund der gleichen technischen Vorbedingungen und den Zwängen der Spezifikation ergibt sich jedoch in der Praxis ein sehr ähnliches Lösungskonzept für zentrale Aufgaben
 - Request-Verarbeitung
 - Multithreading
 - Deployment

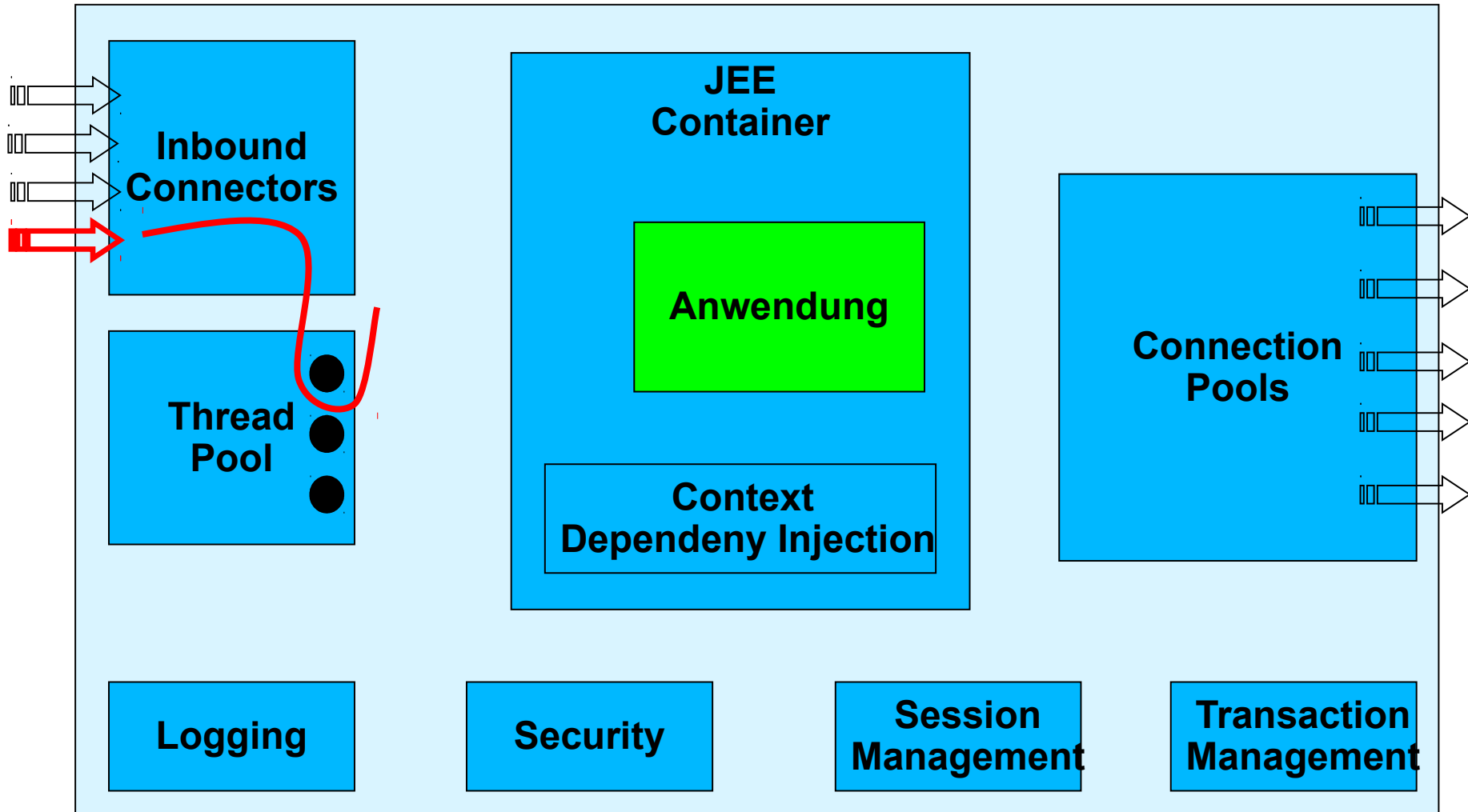


2.2

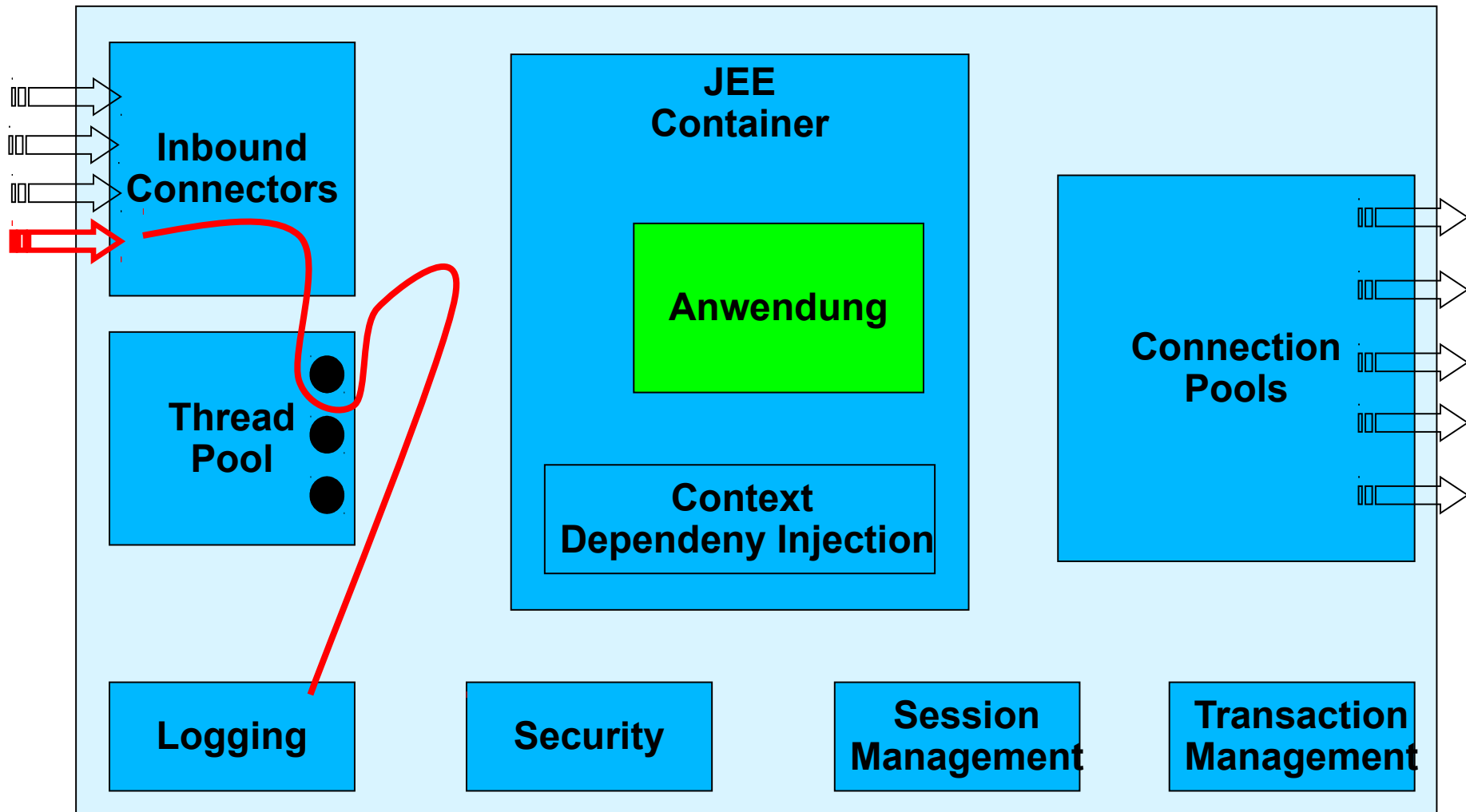
REQUEST VERARBEITUNG

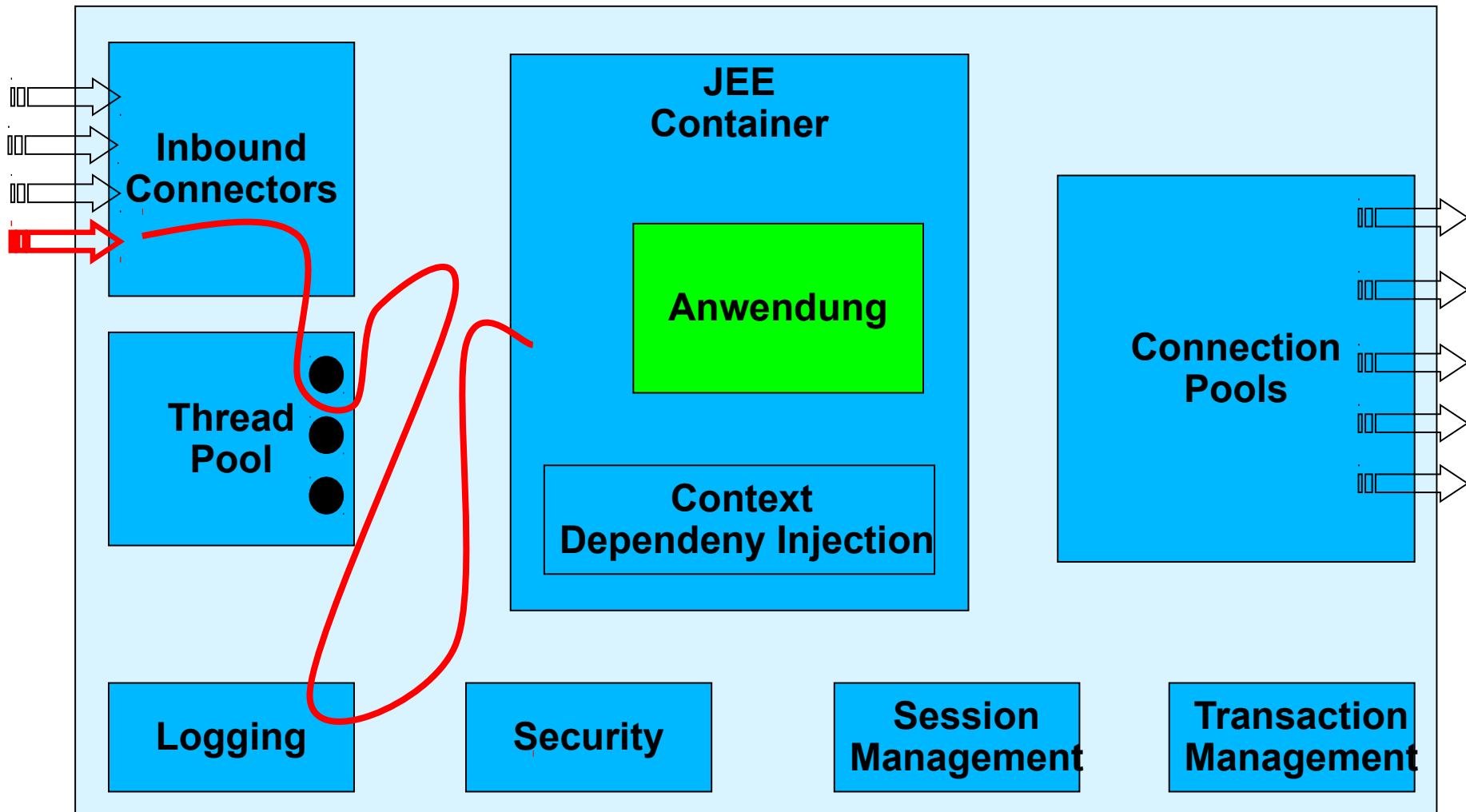


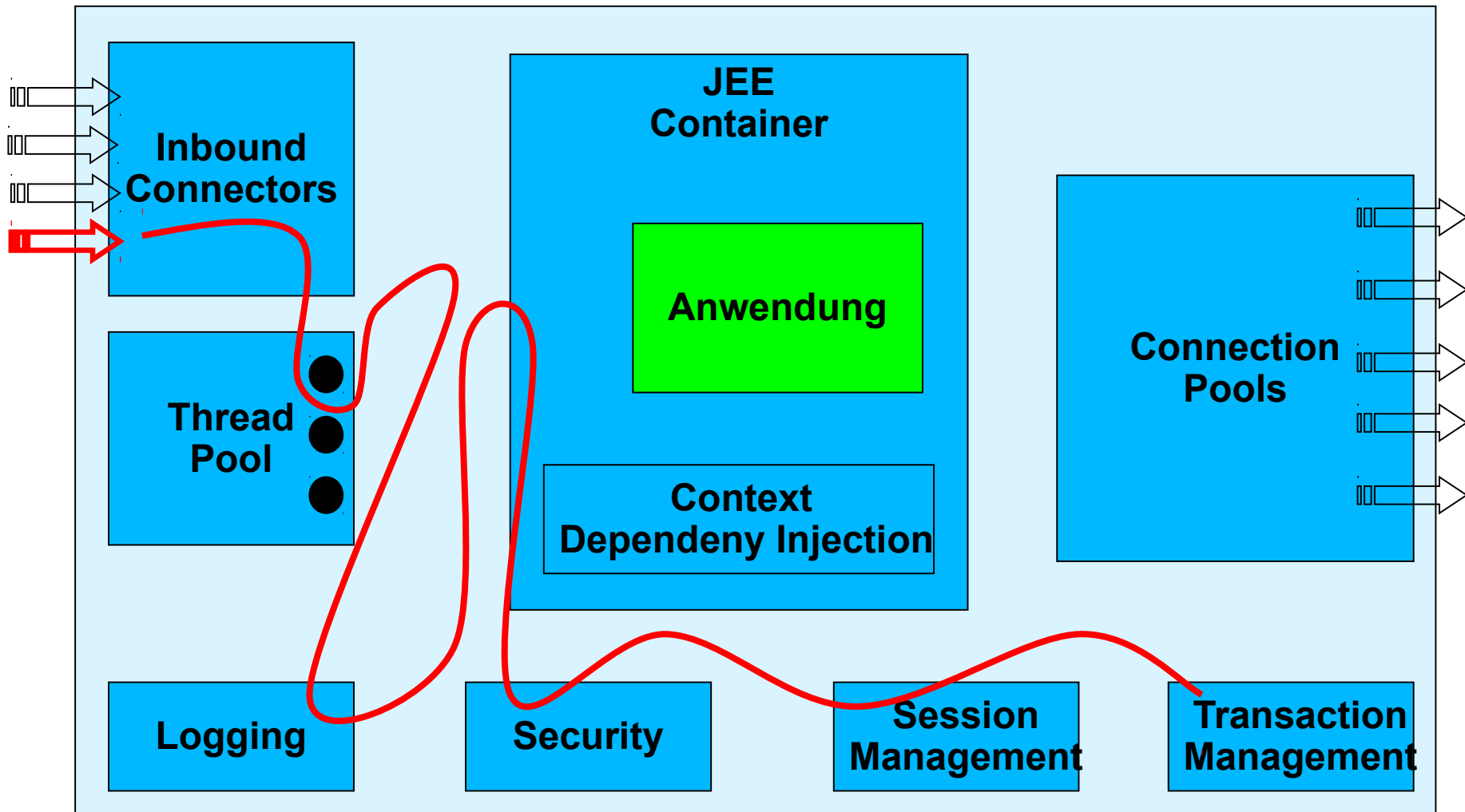
Verarbeitung durch Thread aus Pool

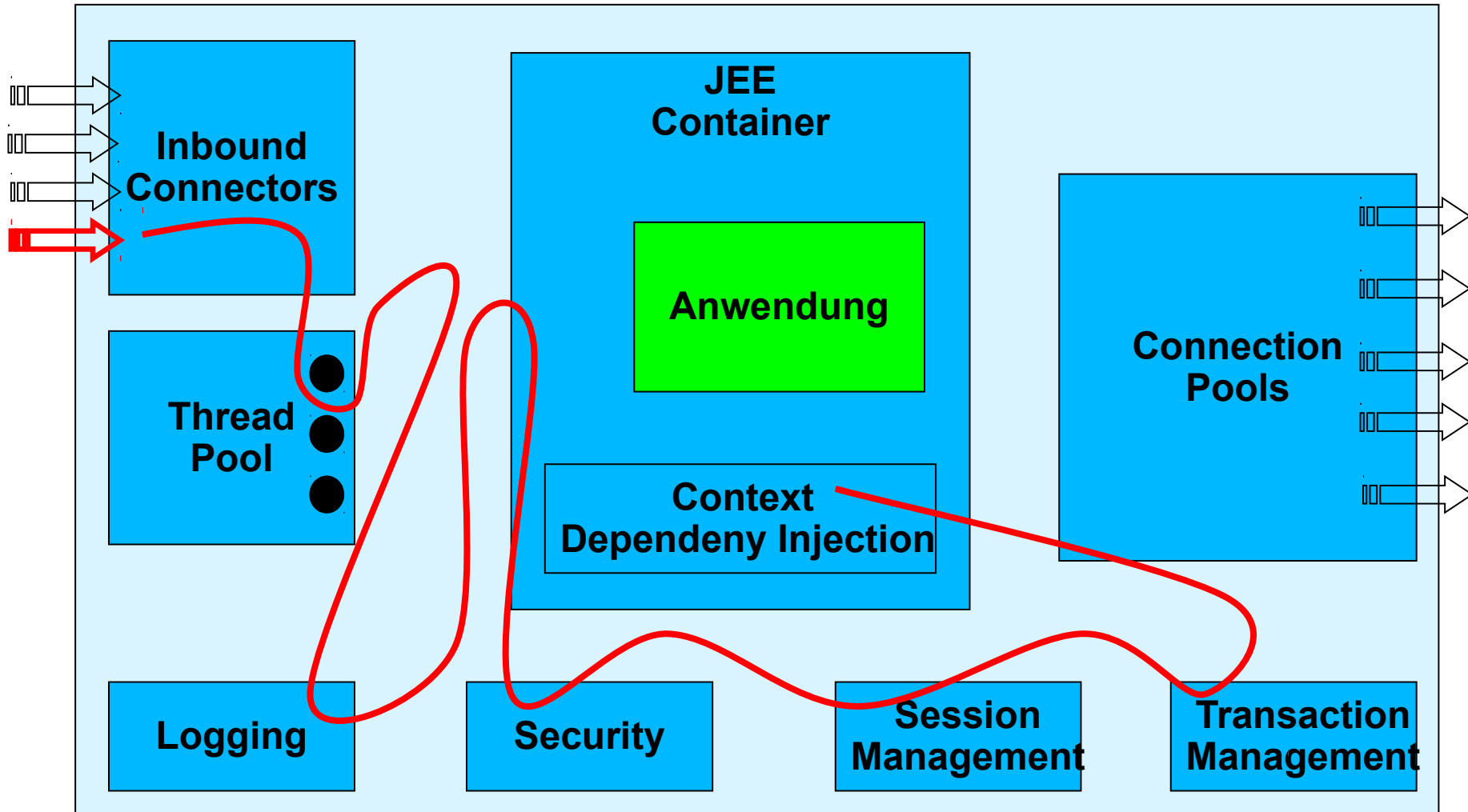


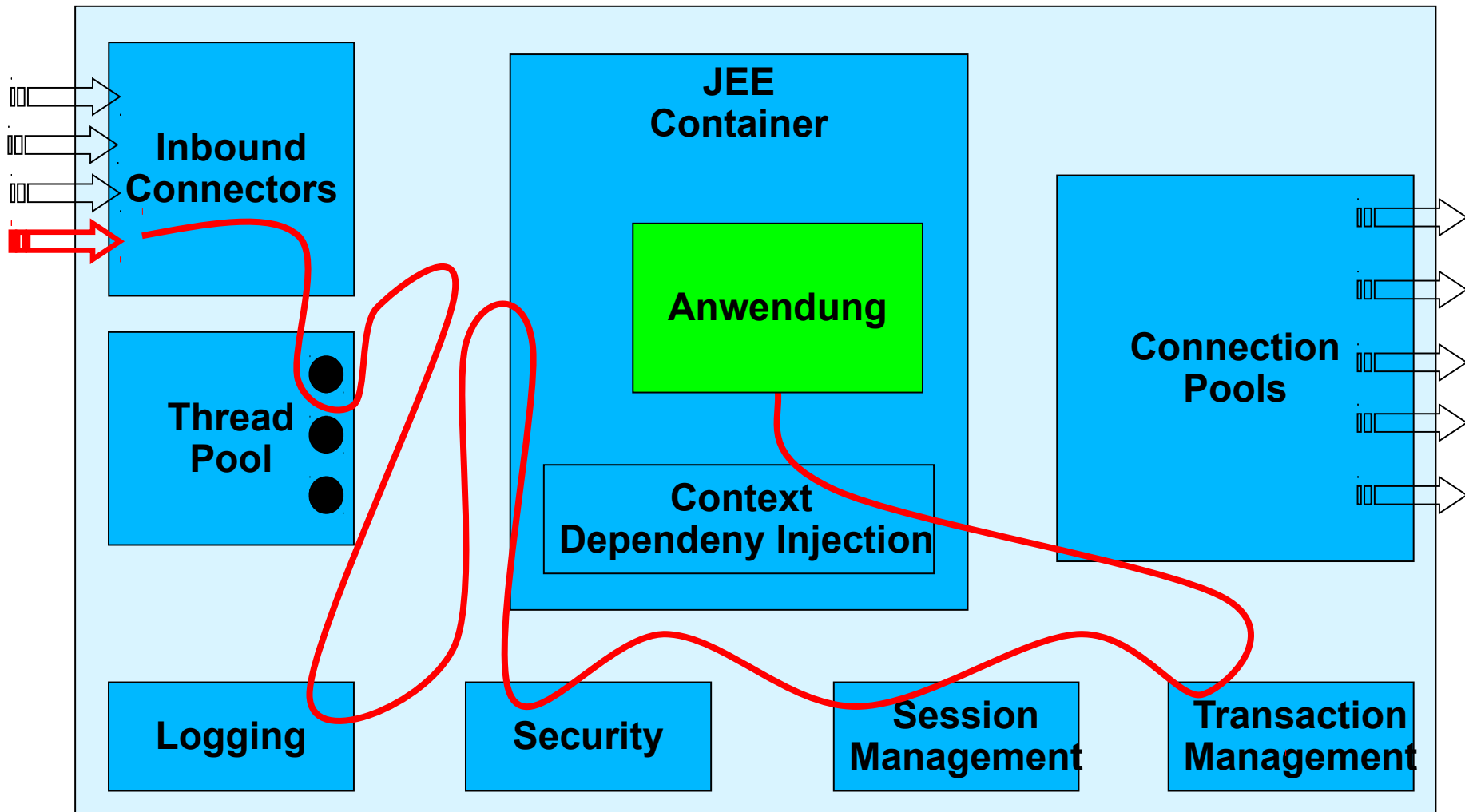
Aufruf des zentralen Loggings



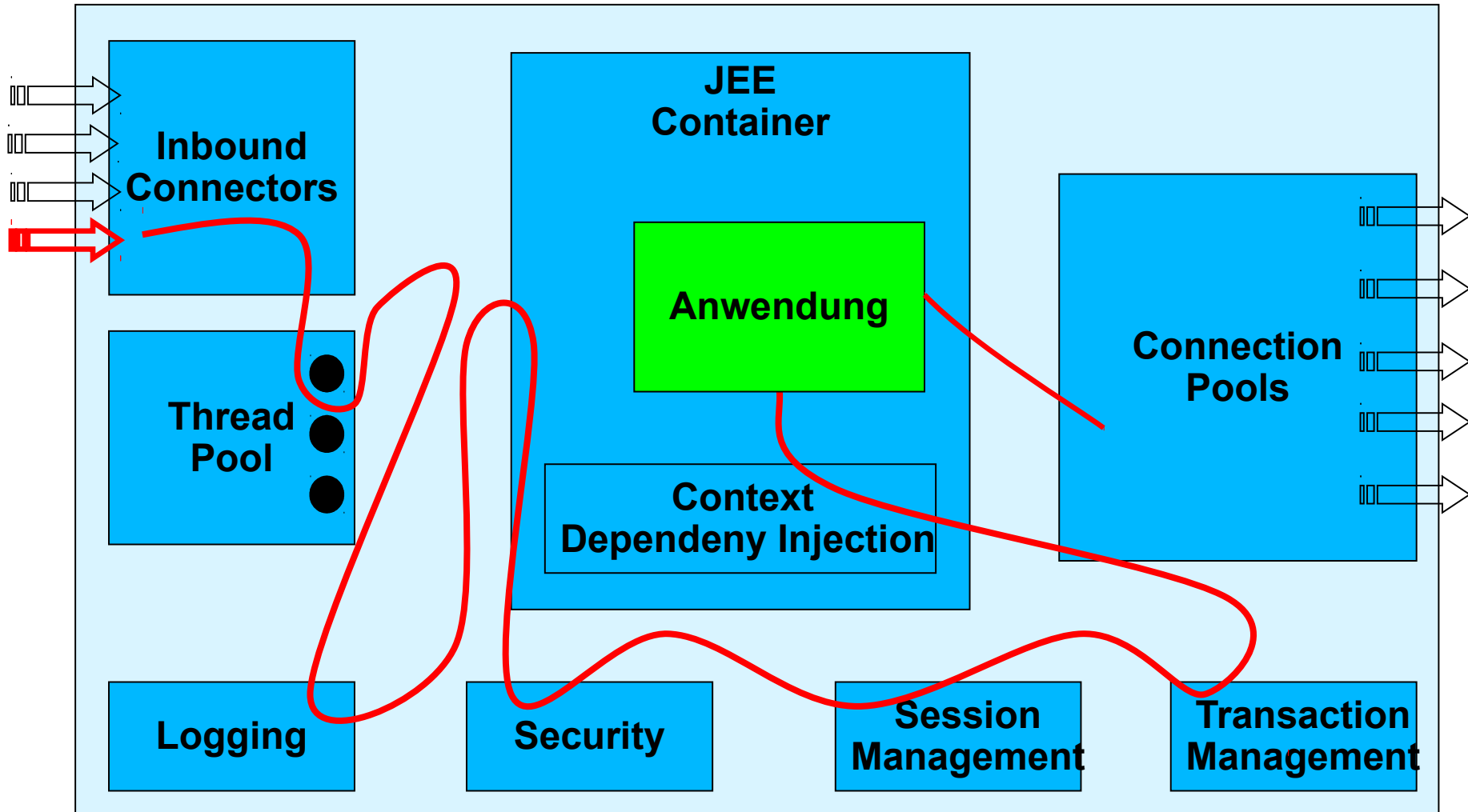




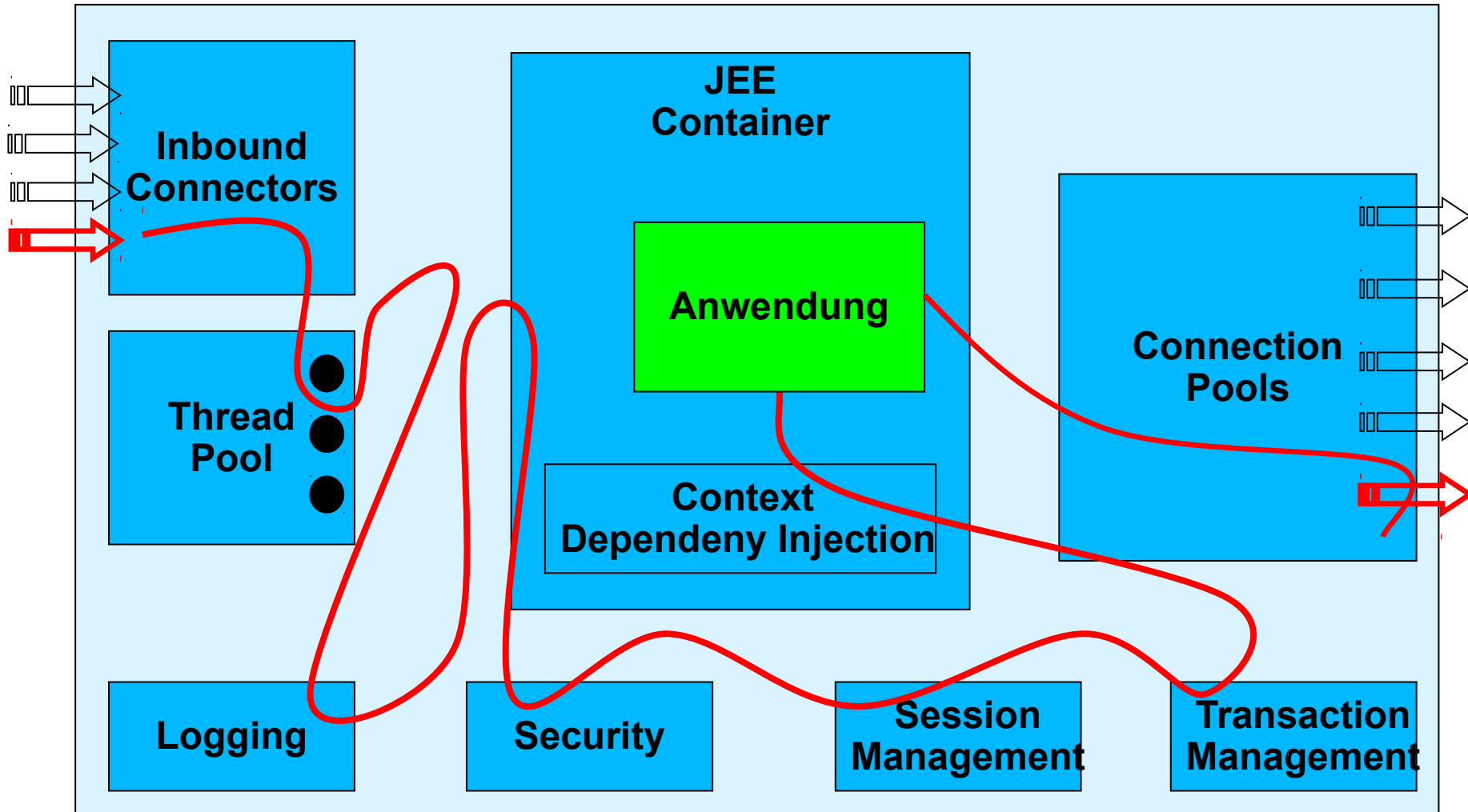


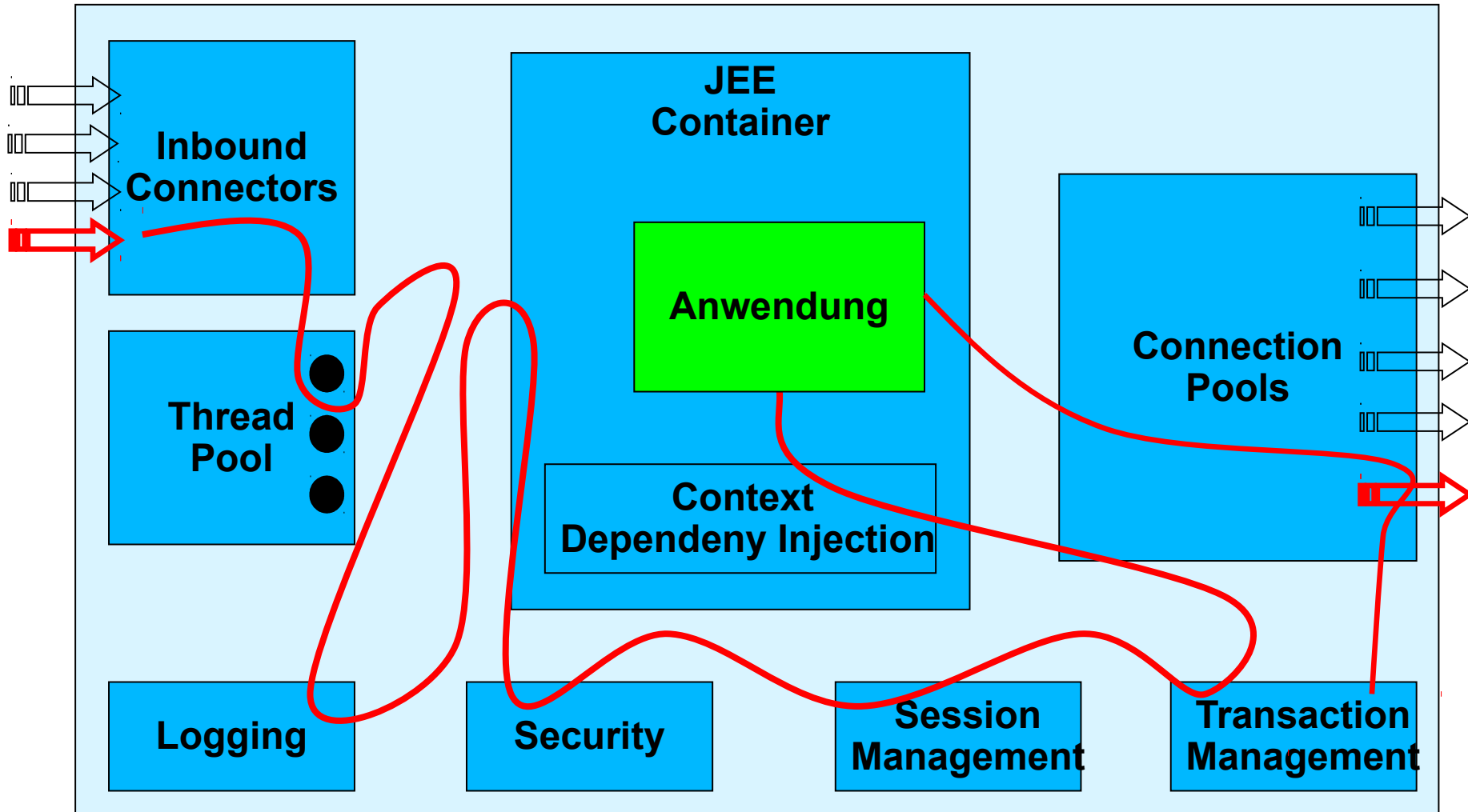


Optional: Anforderung einer Connection

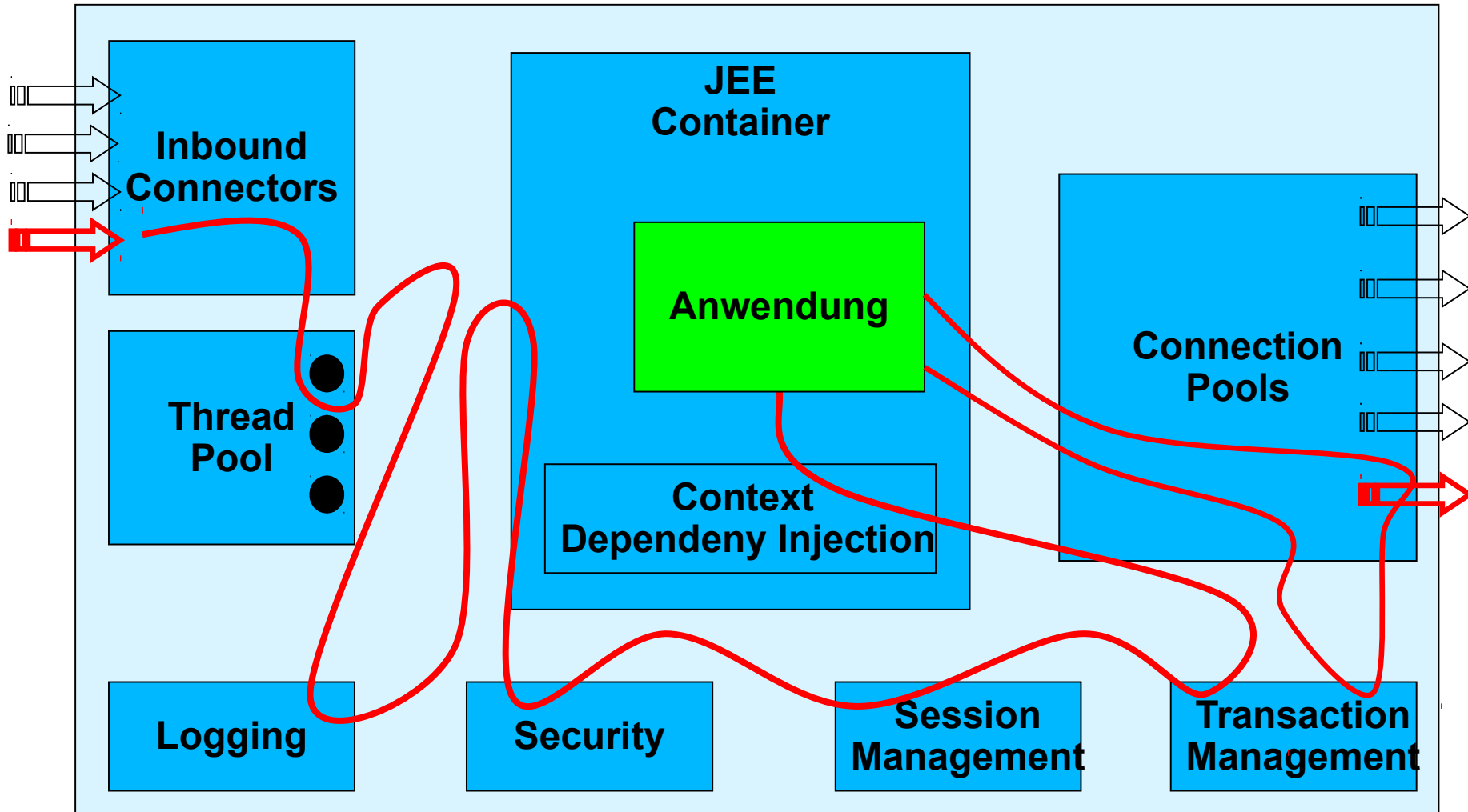


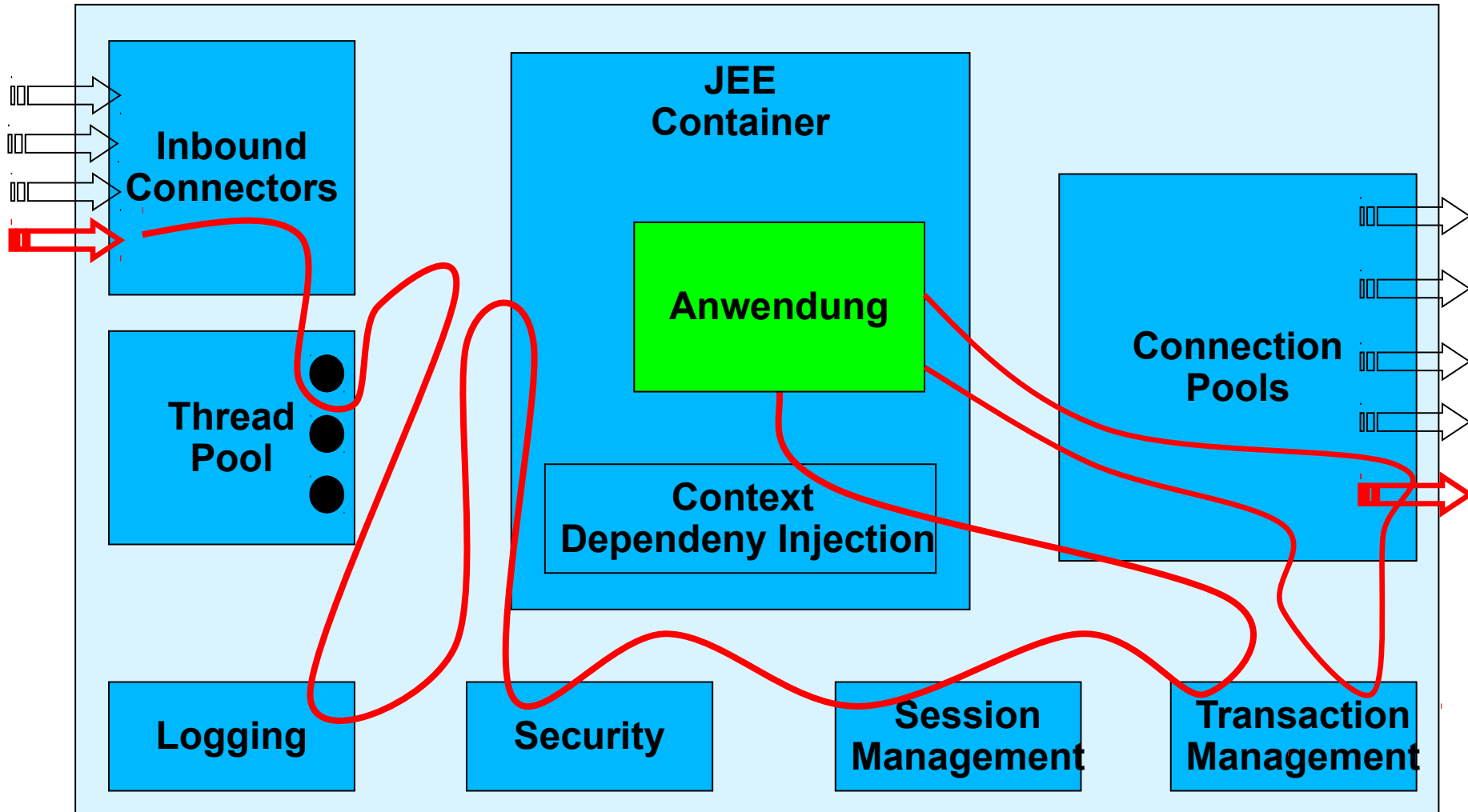
Reservierung der Connection





Rückgabe der Connection an Anwendung





2.3

DEPLOYMENT

- JEE Anwendungen sind gepackte Java-Archive
 - Optionale Zusatz-Informationen werden in XML-Deskriptoren hinzugefügt
- Anwendungs-Typen:
 - Anwendungen mit Enterprise JavaBeans
 - Endung `.jar` („Java Archive“)
 - Web-Applikationen
 - Endung `.war` („Web Archive“)
 - Spezielle Connector-Implementierungen
 - Endung `.rar` („Resource Archive“)
 - Gesamt-Anwendungen
 - Endung `.ear` („Enterprise Archive“)
 - Darin möglich sind Web-Archive, EJB-Archive und Resource Archive
- Die JEE 7 präferenziert als Packaging war-Dateien

- Das Deployment kann durch verschiedene Verfahren initiiert werden:
 - Auto-Deployment
 - Kopieren des Archivs in ein vom Hersteller definiertes Verzeichnis
 - Deployment-Tools, z.B. eine Web-Oberfläche
 - Wizard-gesteuerter File-Upload
 - Scripting
 - Hersteller-abhängige Skripte, die direkt mit dem laufenden Applikationsserver kommunizieren
 - Programmatisch
 - JEE definiert ein Deployment-API für Java-Anwendungen

- Das Deployment kann im laufenden Betrieb erfolgen
- Dabei können bereits installierte Anwendungen durch neue Versionen ersetzt werden
 - Interessant für Entwicklung und Test
 - Im Produktiv-Betrieb häufig nicht genutzt
- Hersteller implementieren teilweise eine ausgereifte Deployment-Prozedur
 - Parallele Bereitstellung verschiedener Versionen
 - Rollback-Mechanismus

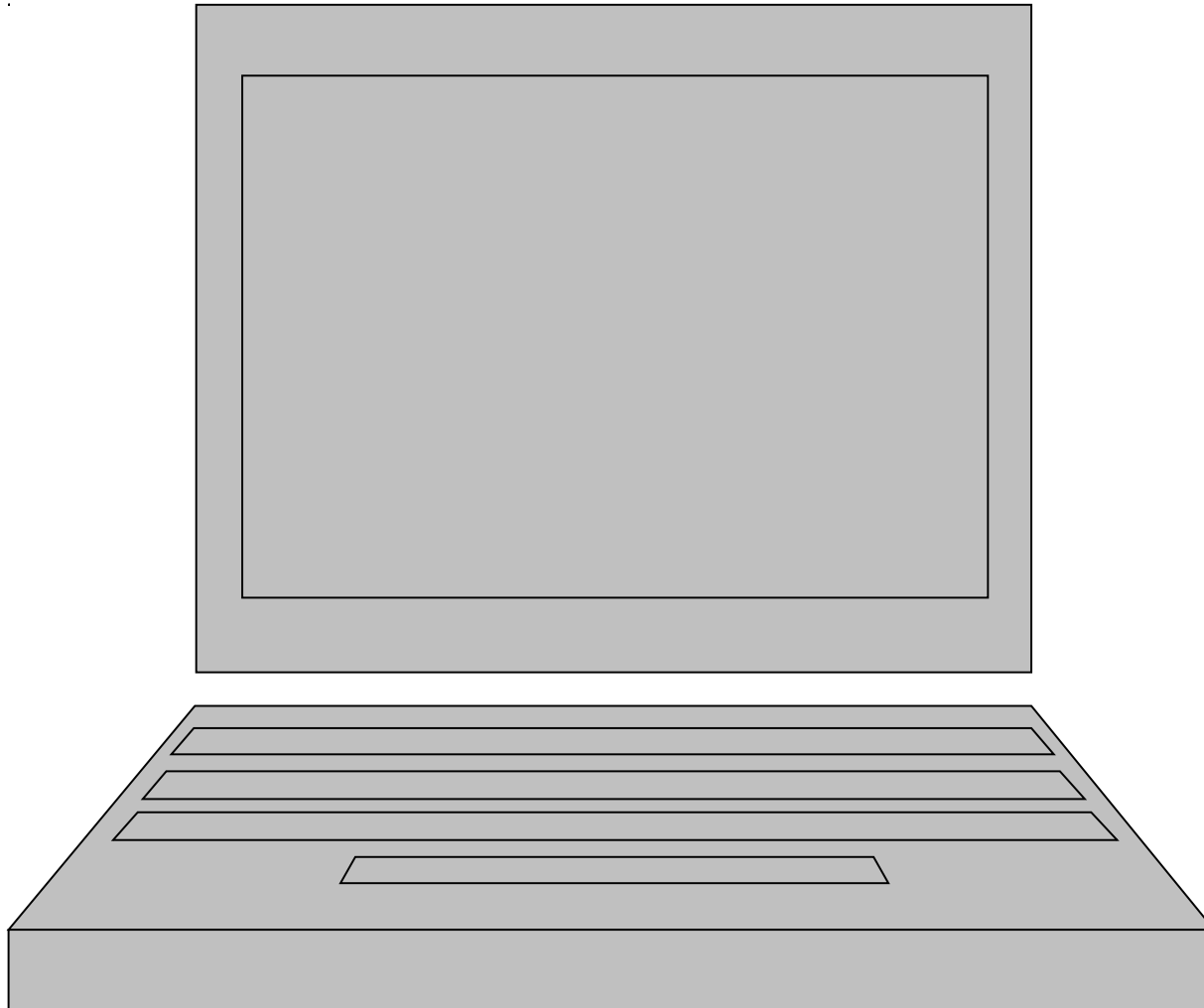
- Wird eine neue Anwendung installiert wird eine komplexe Deployment-Sequenz durchgeführt:
 - Bereitstellung eines eigenen Klassenladers zur Kapselung verschiedener Anwendungen
 - Auslesen der im Archiv vorhandenen Klassen und Deskriptoren, um Meta-Informationen zu erhalten:
 - Transaktionsverhalten
 - Security
 - Anbindung an Inbound Connectors und damit Netzwerk-Protokolle
 - Abhängigkeiten und benötigte Ressourcen
 - Erzeugung des Containers und konkrete Anbindung an die Dienste des Applikationsservers
 - Instanziierung der Anwendungsklassen und Auflösen der Abhängigkeiten

2.4

ADMINISTRATION

- Die Dienste des Applikationsservers müssen konfiguriert werden
 - Verzeichnisse
 - Pool-Größen
 - Timeouts
 - ...
- Konfiguration erfolgt über
 - Editieren von Konfigurationsdateien
 - Aufruf von Skripten
 - Benutzung einer Web-basierten Administrations-Oberfläche

Demo: Administrations-Oberfläche eines Applikationsservers



2.5

ÜBERWACHUNG

- Betriebssystem
 - Standard-Werkzeuge der System-Überwachung
 - CPU
 - Speicher
 - Verfügbarkeit
- Java Virtual Machine
 - Java Management Extension (JMX)
 - Threads
 - Interne Speicherorganisation und Garbage Collection
 - Geladene Klassen

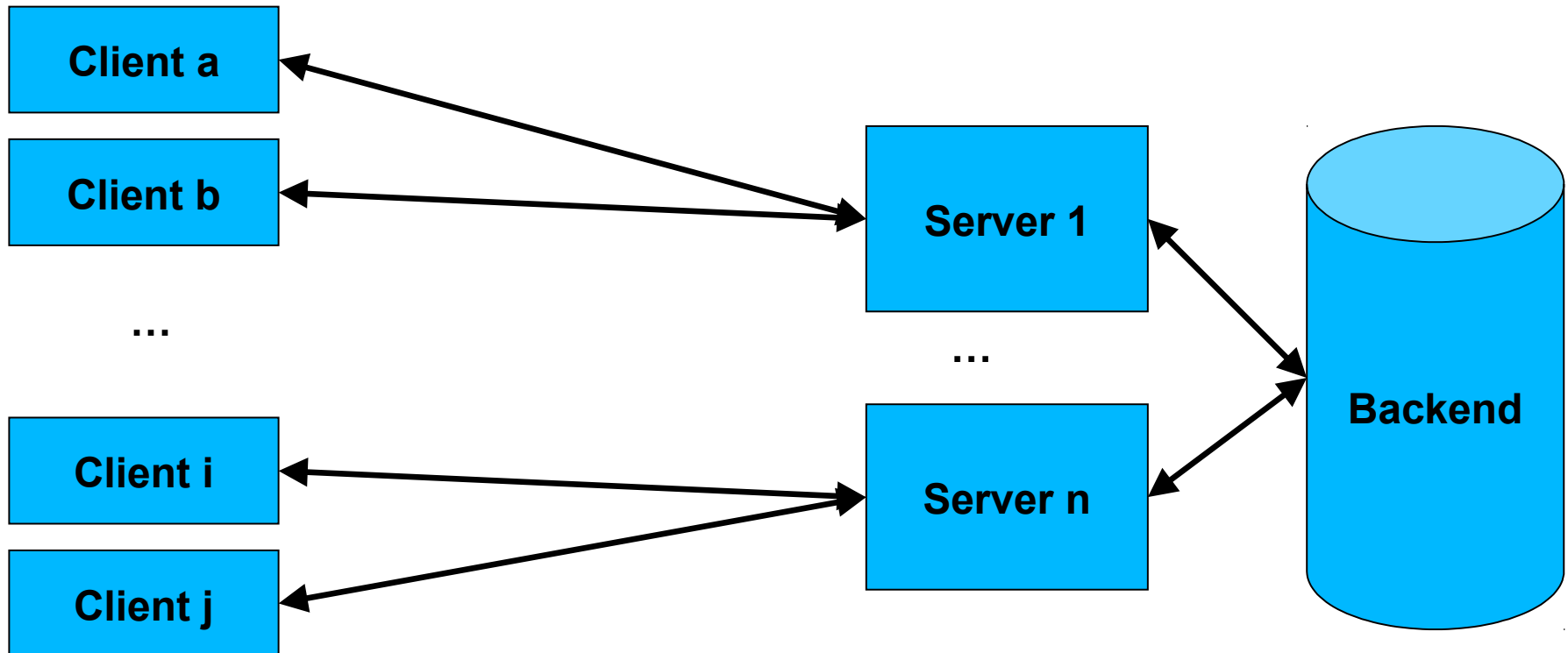
- Logging
 - Editor für Log-Dateien
 - Allgemeine Informationsausgaben
 - Fehlermeldungen
 - Detaillierte Debug-Ausgaben für Entwicklung und Test
- JEE Metriken
 - Zugriff über JMX oder Administrationswerkzeuge
 - Zugriffszeiten
 - Pool-Größen
 - Timeouts

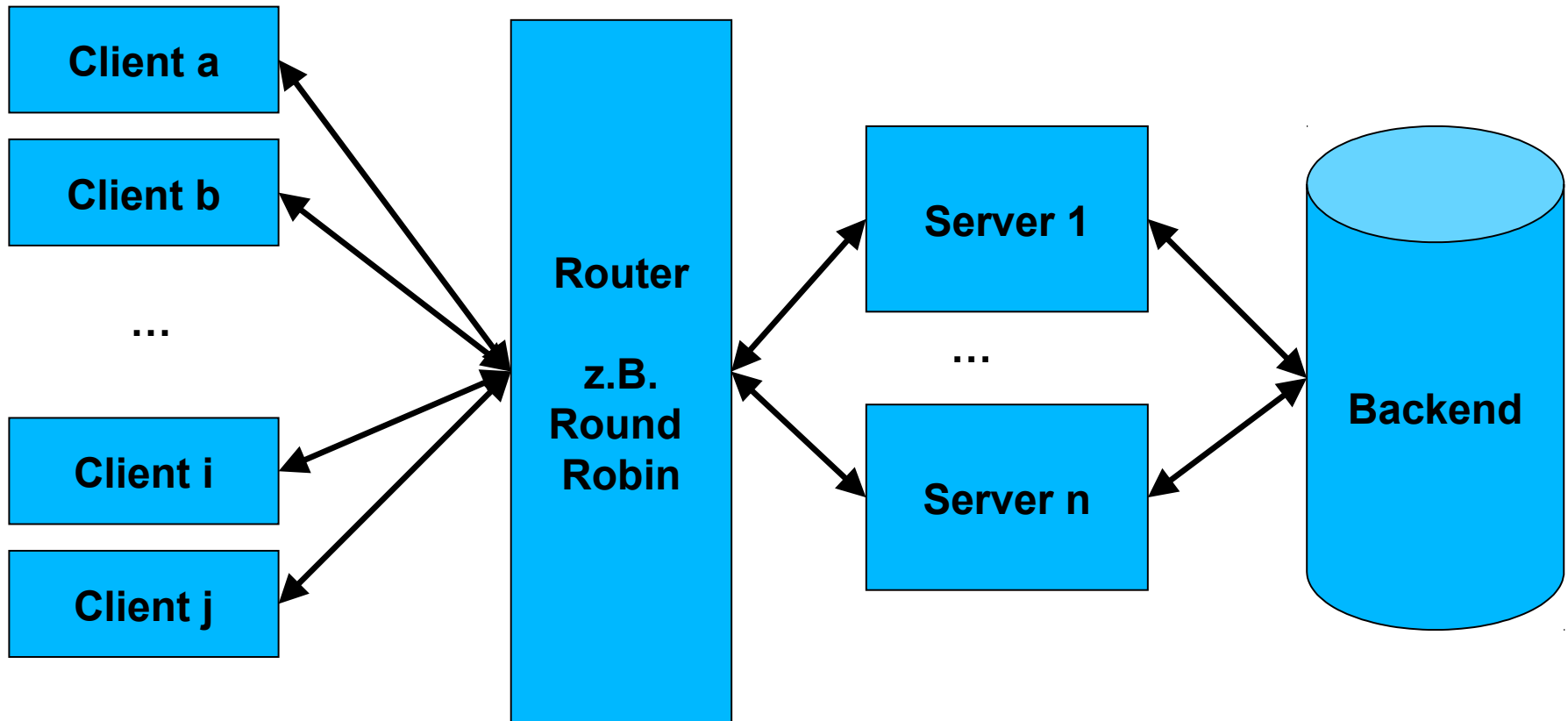
- Der Applikationsserver schreibt eine Log-Datei
 - Enthält sämtliche Fehler-Meldungen sowie Status-Informationen
 - Gruppierung in einzelne „Log Level“

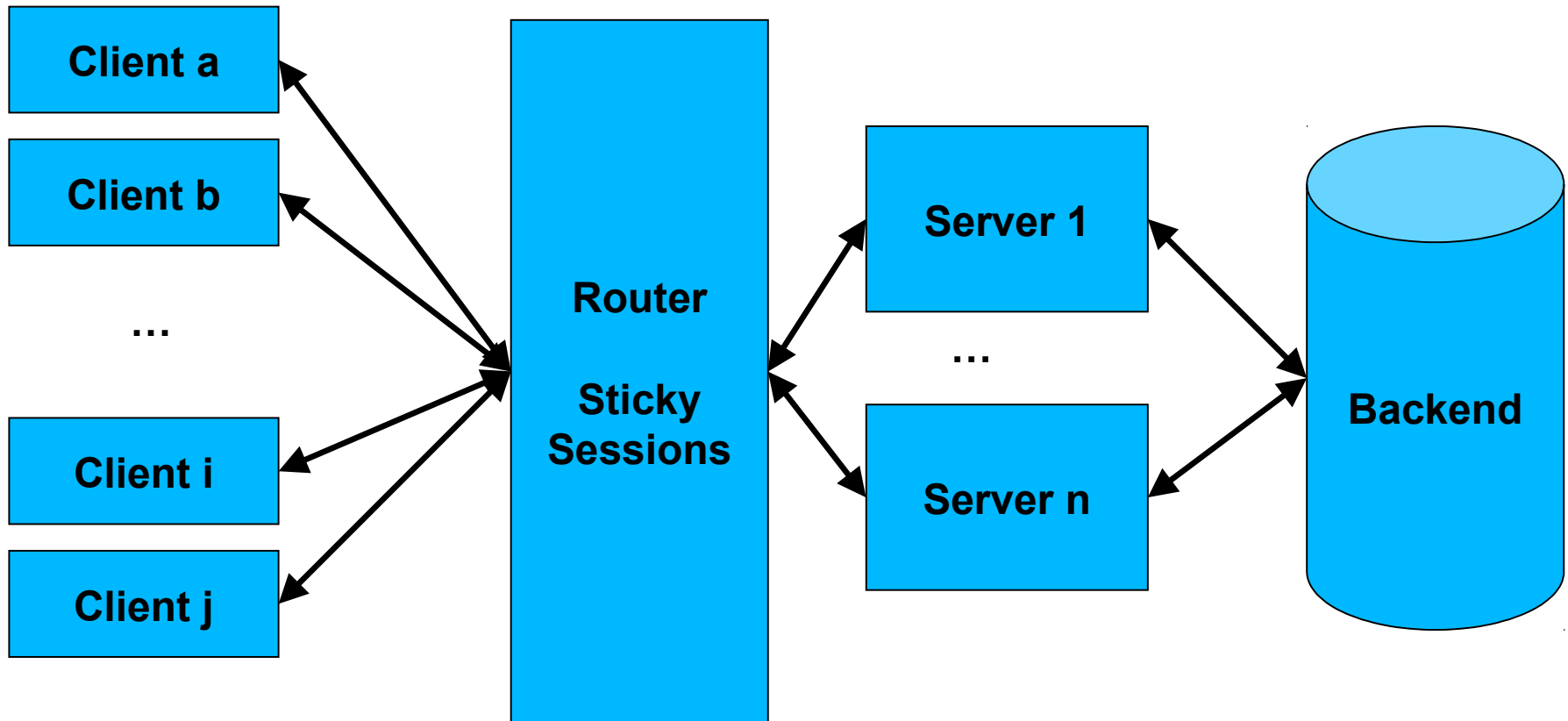
- Sämtliche relevanten System-Parameter werden überwacht
 - Pool-Größen
 - Speicher-Verbrauch
- Ebenso Zugriffszeiten auf die Anwendung

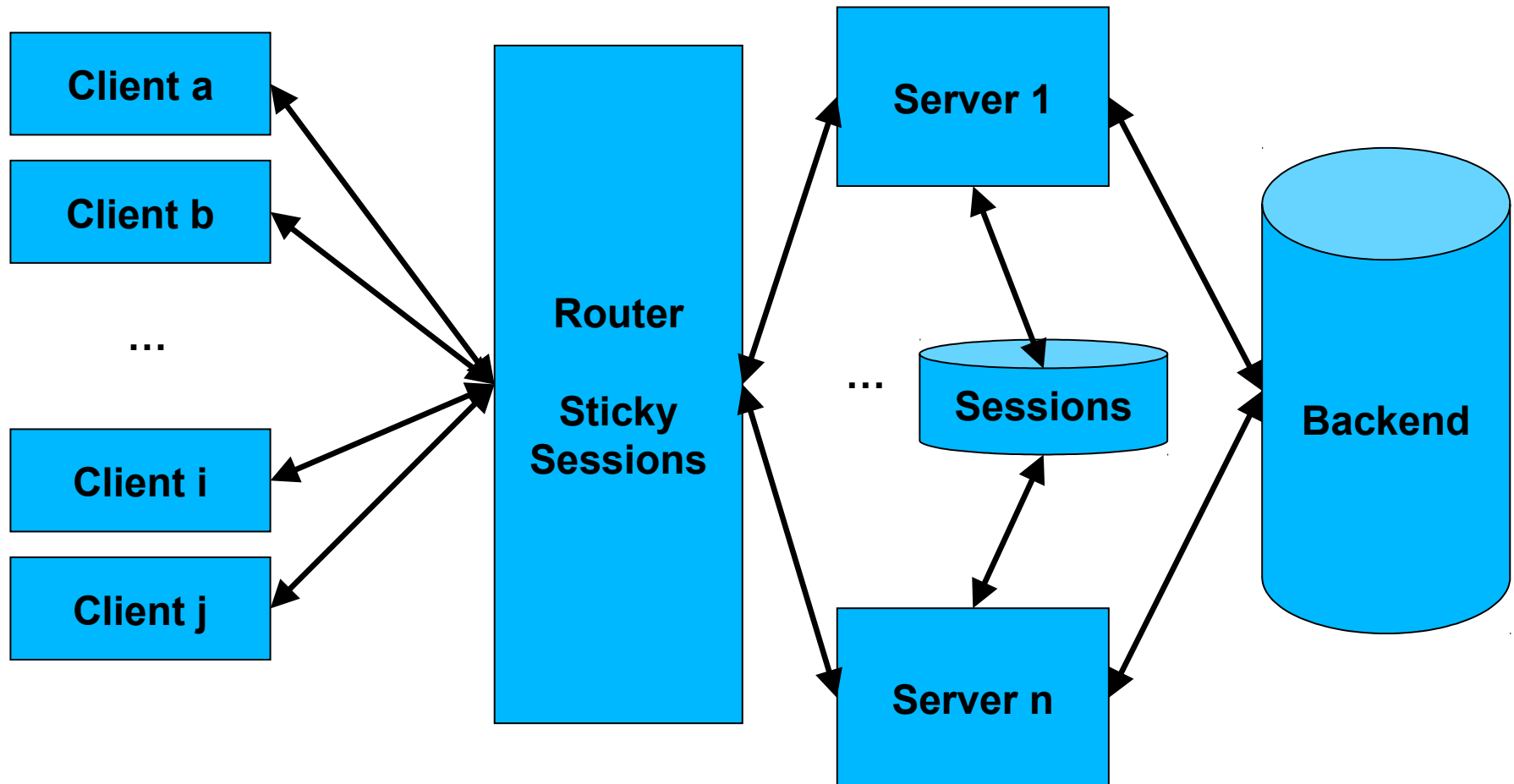
2.6

CLUSTER-SYSTEME









3

DIENSTE DES APPLIKATIONSSERVERS

3.1

CONNECTORS

- Empfängt Requests über eine Netzwerk-Verbindung
 - Auch Datei-basiertes Connectors sind möglich
- Benutzt für die Verarbeitung einen freien Thread aus dem Thread Pool
 - Durch den Pool wird eine Überlastung der Server-Ressourcen vermieden
- Delegiert die Aufrufe an einen JEE Container weiter
- Schreibt das Ergebnis zurück in das Netzwerk-Protokoll

- Die folgenden Protokolle müssen von einem JEE Applikationsserver durch Inbound Connectors unterstützt werden:
 - http/https
 - Das Internet-Protokoll für Web-Anwendungen
 - Java RMI
 - Ein Protokoll für die Kopplung zweier Java-Prozesse
 - JMS
 - Messaging

- SOAP/http
 - SOAP-basierte Web Services
- IIOP
 - Kommunikation mit CORBA-Anwendungen
- REST
 - RESTful Web Services

- Es können jederzeit eigene Inbound Connectors realisiert werden
 - Die JEE legt ein Programmier-API und ein Deployment-Verfahren fest
- Beispiele:
 - Raw TCP/IP
 - Datei-basierte Connectors
 - Mail-gesteuerte Connectors
 - ...

- Bindet ein Backend-System in den Applikationsserver ein
 - Zugriffe auf Backend Systeme erfolgen nie direkt aus dem Programm heraus
- Hält einen Pool von Connections und stellt diese der aufrufenden Anwendung zur Verfügung
 - Diese Connections werden in der Regel bereits beim Start des Servers angelegt
 - Die Pool-Größe kann dynamisch der Last angepasst werden
- Initiiert und steuert Transaktionen im Backend-System
- Unterstützt das vom Backend-System geforderte Authentifizierungsprotokoll

- Die folgenden Backend-Systeme müssen von Applikationsservern eingebunden werden können:
 - Datenbanken
 - Die `javax.sql.DataSource`
 - Messaging-Systeme
 - Die `javax.jms.ConnectionFactory`

- Es können jederzeit eigene Outbound Connectors realisiert werden
 - Die JEE legt ein Programmier-API und ein Deployment-Verfahren fest
- Im Gegensatz zu Inbound Connectors sind Outbound-Connectors deutlich aufwändiger
 - Allerdings existiert ein breiter Markt von Drittanbietern
 - Auch JEE Provider statten ihre Produkte häufig mit besonderen Connectors aus
 - IBM WebSphere mit Host-Anbindung
- Beispiele:
 - Raw TCP/IP
 - File Connectors
 - SAP Connector
 - CICS-Connector
 - ...

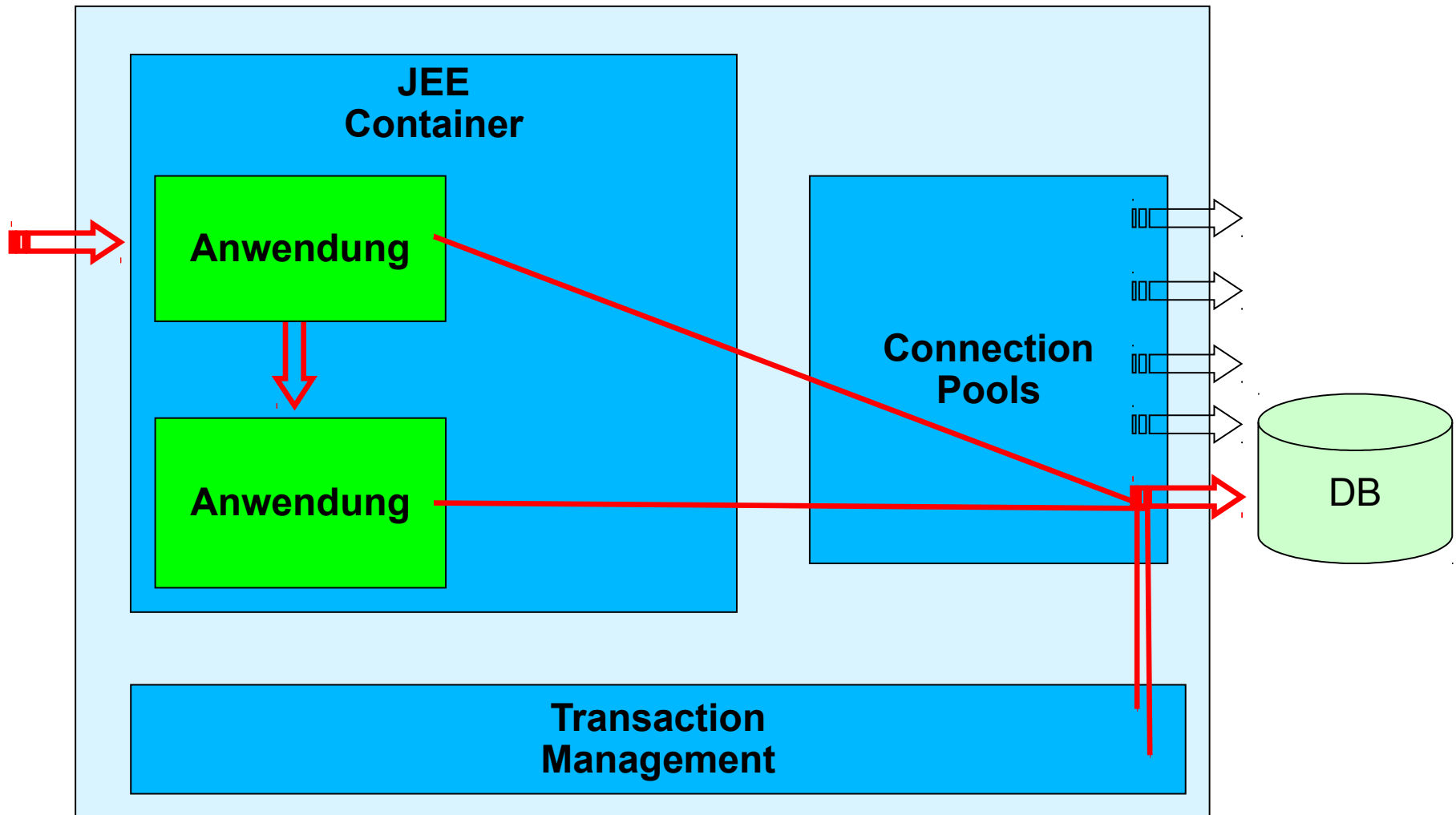
3.2

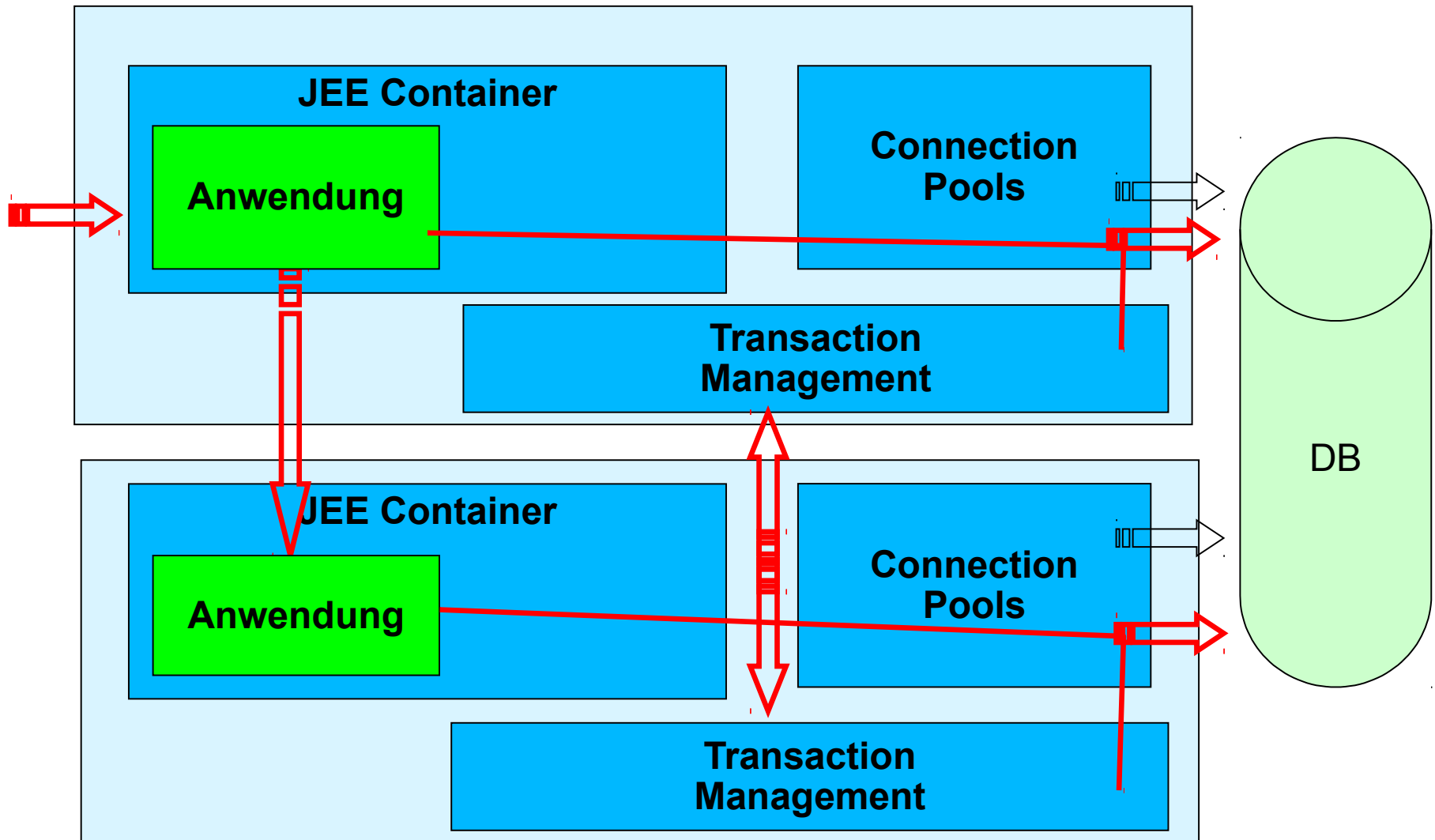
DER TRANSACTION MANAGER

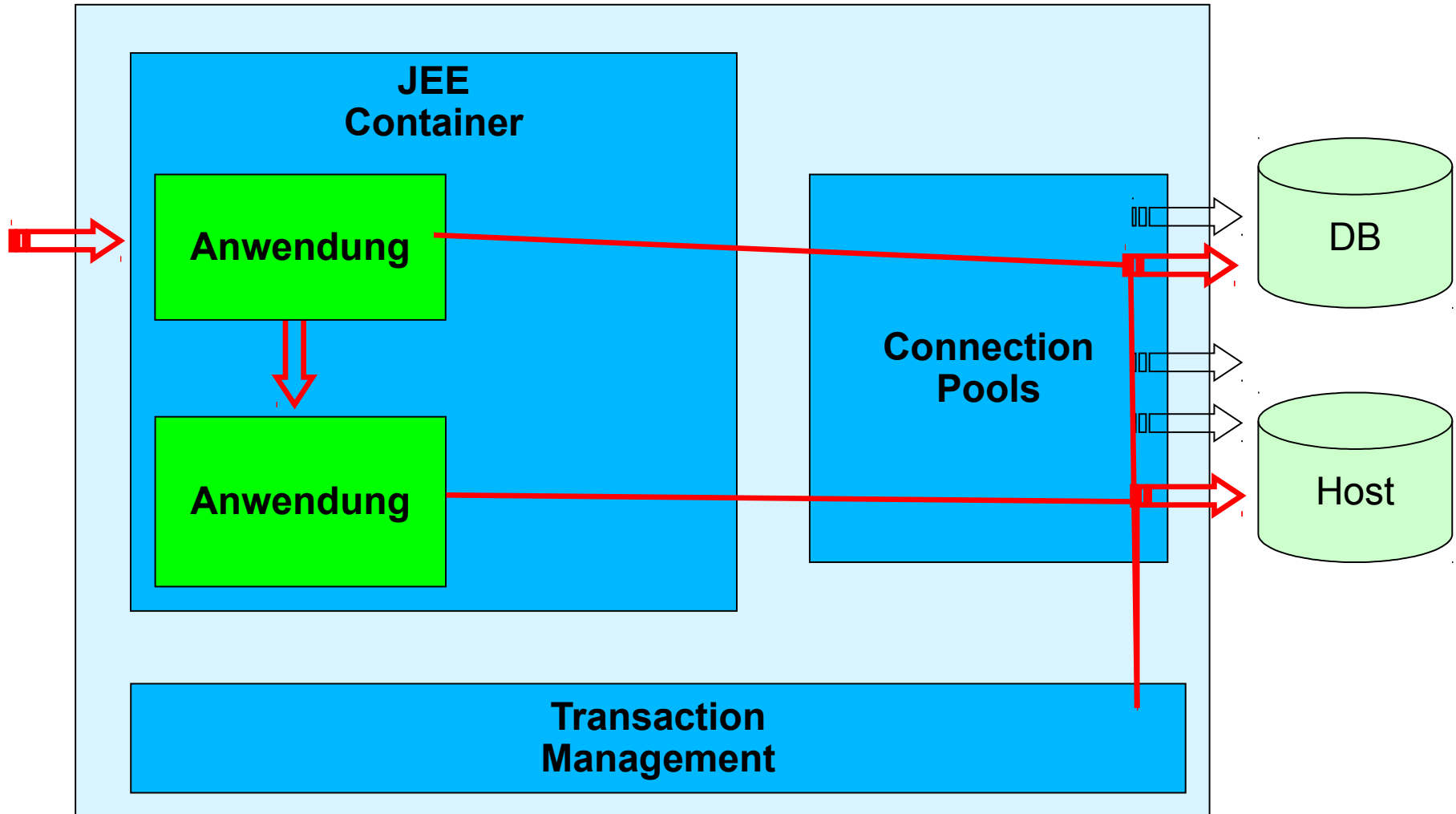
- Der Applikationsserver ist in der Lage, Transaktionen in Backend-Systemen zu starten und zu koordinieren
 - Dies ist die eigentliche Kern-Funktion des JEE Applikationsservers
 - Reine Web Server müssen keinen Transaktionsmanager zur Verfügung stellen
- Über Inbound Connectors kann auch eine Transaktion übernommen werden
 - Dies ist allerdings abhängig vom zu Grunde liegenden Protokoll
- Sämtliche Ressourcen-Zugriffe über Outbound Connectors werden vom Transaktionsmanager kontrolliert

- Lokale Transaktionen
 - Umspannen Aktionen innerhalb eines Applikationsservers

- XA Transaktionen sind
 - Verteilt über mehrere Server-Instanzen oder
 - Ressourcen-übergreifend







3.3

SESSION MANAGEMENT

- Ein Client kann bei Bedarf Zustand auf dem Server ablegen
 - Dazu definiert er eine Session
- Gründe für Serverseitigen State sind:
 - Entlastung der Client-Ressourcen
 - Verringerung der Netzwerklast
- Der Client muss den Zustand auf dem Server auch definiert wieder freigeben
 - Sonst droht eine Überlastung des Servers!
 - Auslagerung von Sessions aus dem Speicher
 - Timeout

- Web Anwendungen
 - Senden pro http-Request eine Session-ID
 - Als Request-Parameter
 - Als Cookie-Parameter
- RMI-Clients
 - Können eine stehende TCP/IP-Verbindung benutzen
 - Alternativ kann auch hier eine Session ID benutzt werden
- JMS und SOAP
 - Können im aktuellen Stand der Spezifikation keinen Serverseitigen State benutzen

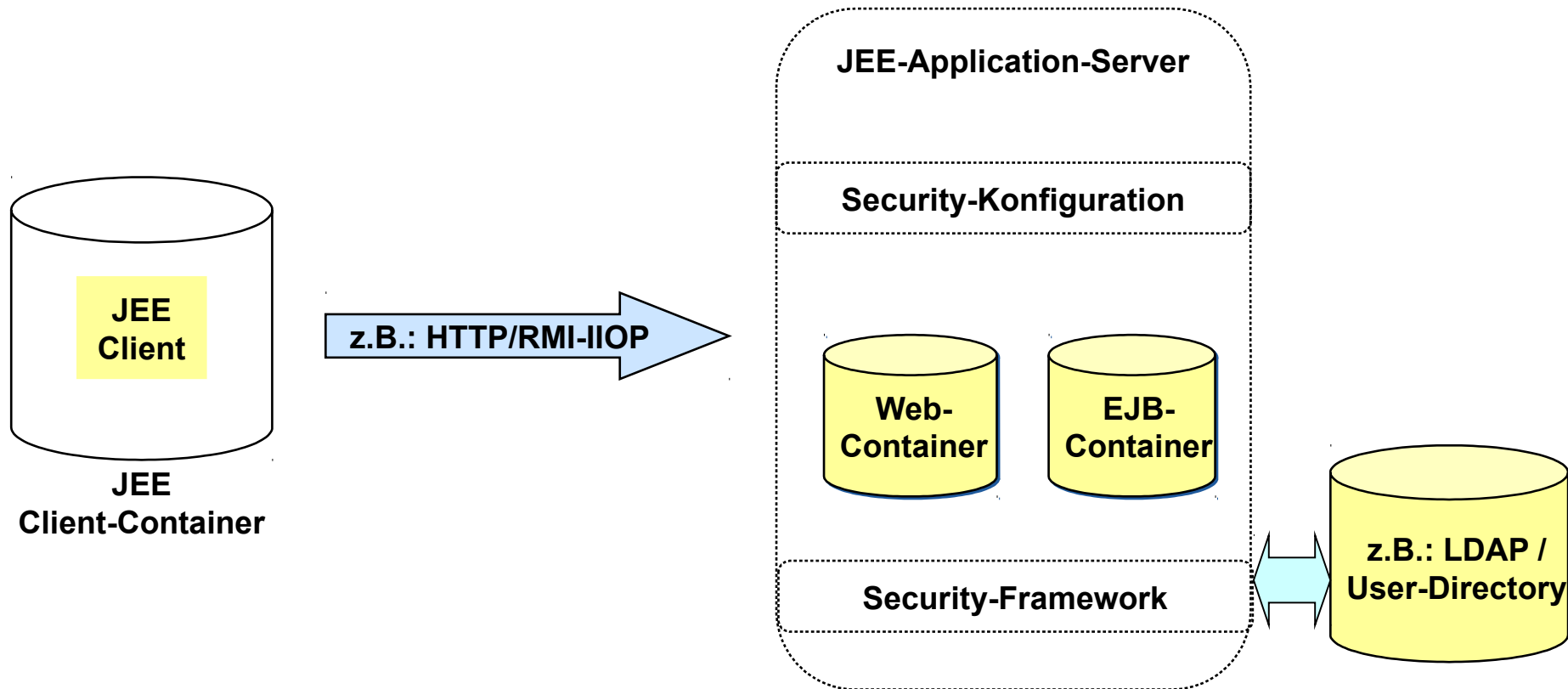
3.4

AUTHENTIFIZIERUNG UND AUTORISIERUNG

- Authentifizierung
 - Ein Benutzer muss seine Identität nachprüfen lassen
- Zugriffskontrolle
 - Zuordnung von Benutzern zu Rechten in einem System
- Sichere Kommunikation
 - Die Verbindung zwischen Client und Server wird verschlüsselt

- JEE Anwendungen definieren eigene Anwendungs-Rollen
 - PowerUser
 - User
 - Guest
- Ein Benutzerverwaltungssystem definiert User und Gruppen
 - Administrator
 - Sachbearbeiter
 - „Georg Metzger“
- Ein Realm mapped die Anwender-Rollen auf User und Gruppen

- Jede Anwendung kann so konfiguriert werden, dass eine Anmeldung erforderlich ist
 - Dazu definiert sie das zu benutzende Realm
 - Das eigentliche Authentifizierungs-Verfahren wird komplett vom Applikationsserver übernommen
- Die Autorisierung erfolgt auf funktionaler Ebene
 - Seiten einer Web Anwendung
 - Methodenaufruf einer Business-Klasse



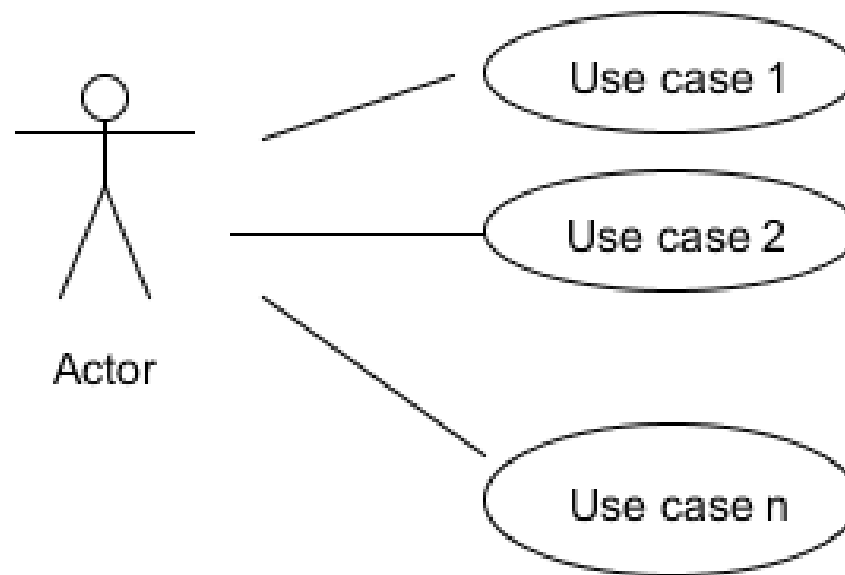
4

PROGRAMMIERUNG VON JEE ANWENDUNGEN

4.1

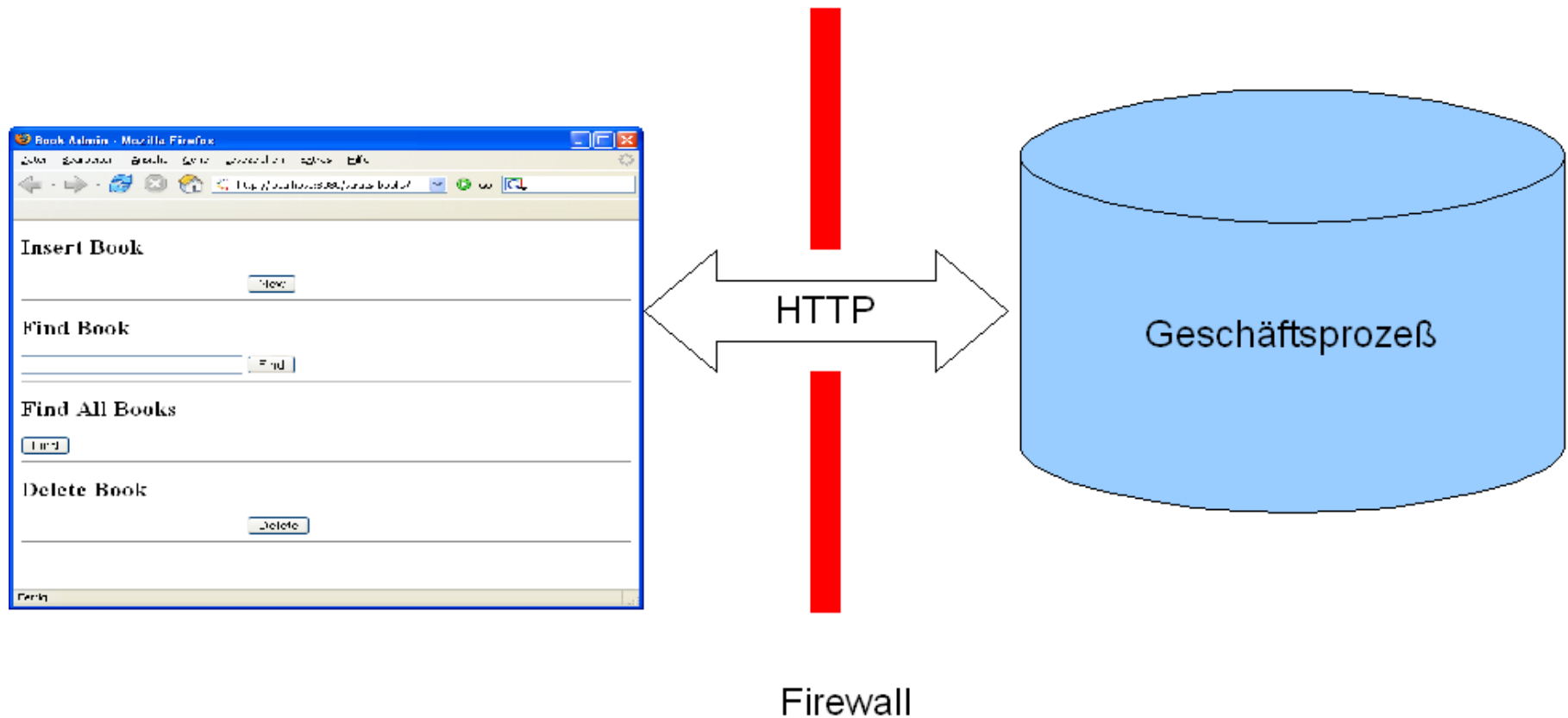
FOKUS UND TYPISCHE BEISPIELE

- Die JEE ermöglicht keine völlig neue Art von Anwendungen
 - Weder technisch noch fachlich
- Auch mit JEE bleibt die klassische Aufgabenstellung:
 - „Eine Fachvorgabe muss in eine funktionierende Anwendung umgesetzt werden!“

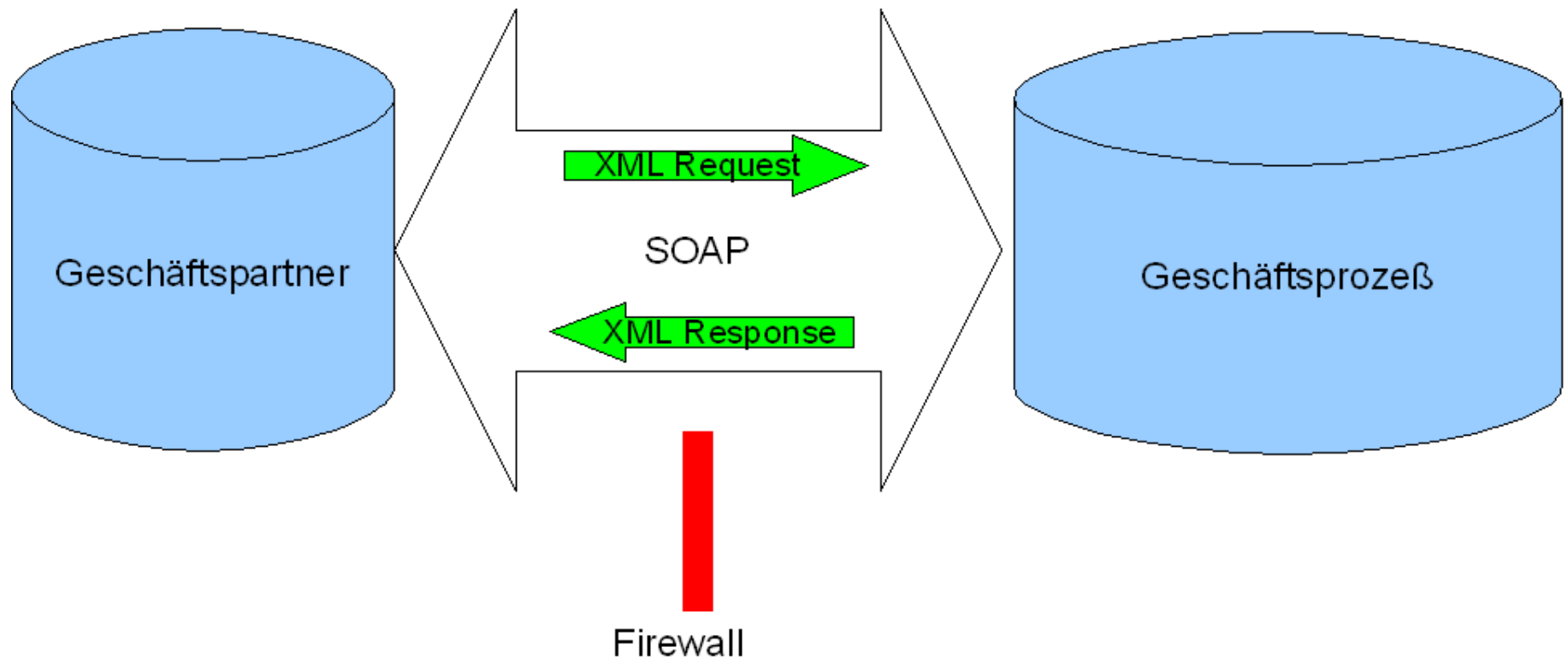


- Die JEE ist auf verschiedene typische Anwendungen hin ausgerichtet, z. B.:
 - Web Applikation mit Browser-basiertem Front End
 - Unternehmens-übergreifende Prozesse
 - Web und Rich Clients
 - Komplexe Transaktionssteuerung
 - ...
- Es können aber auch komplexe Spezial-Aufgaben umgesetzt werden!
 - Workflow-Engines
 - Bus-Systeme
 - Applikationsserver als Content Management System
 - ...
- Auf Basis der JEE hat sich eine breite Produkt-Palette entwickelt
 - jBPM Workflow Engine
 - IBM Process Manager
 - Liferay Portal Server
 - ...

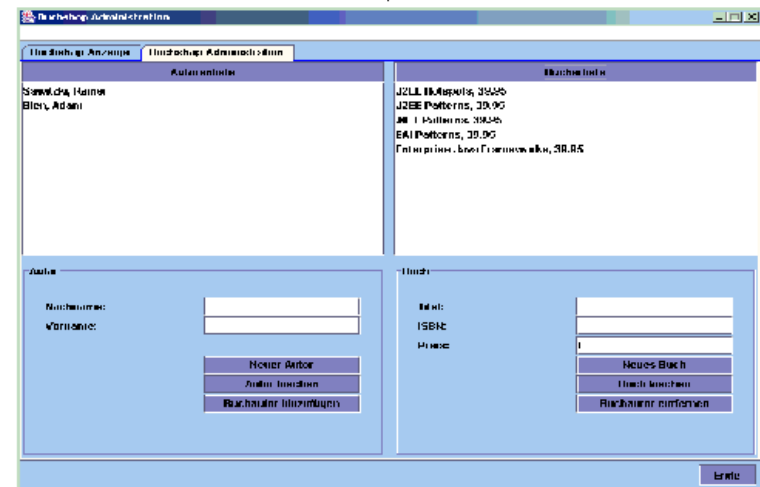
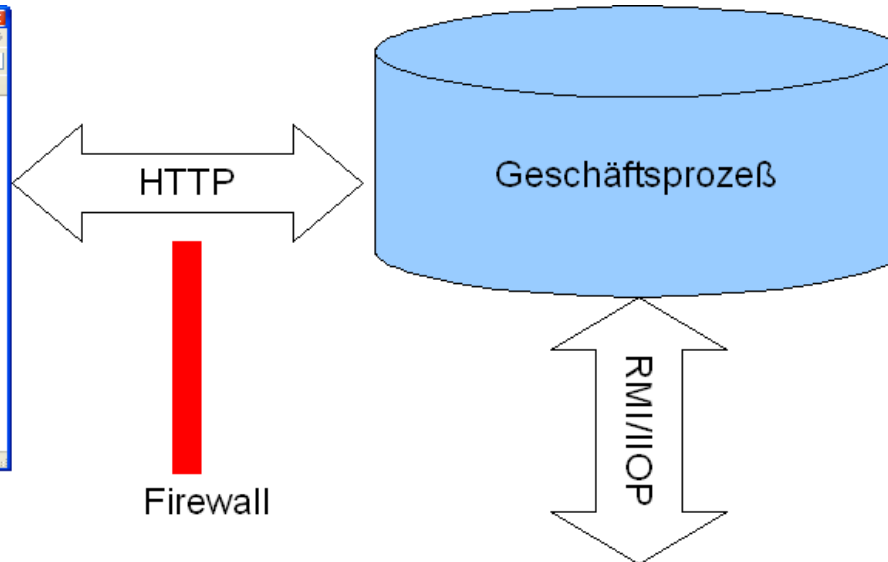
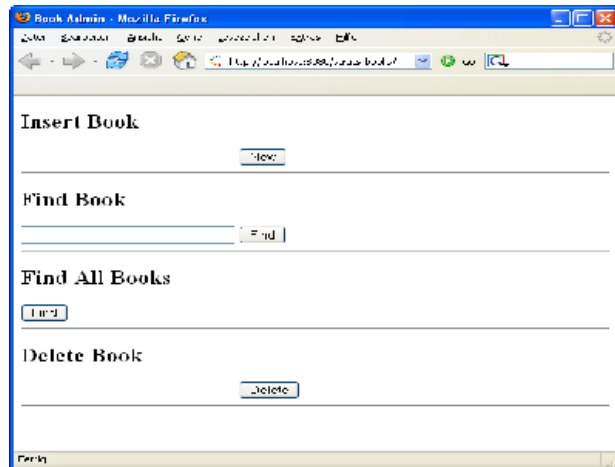
Beispiel 1: Business to Consumer



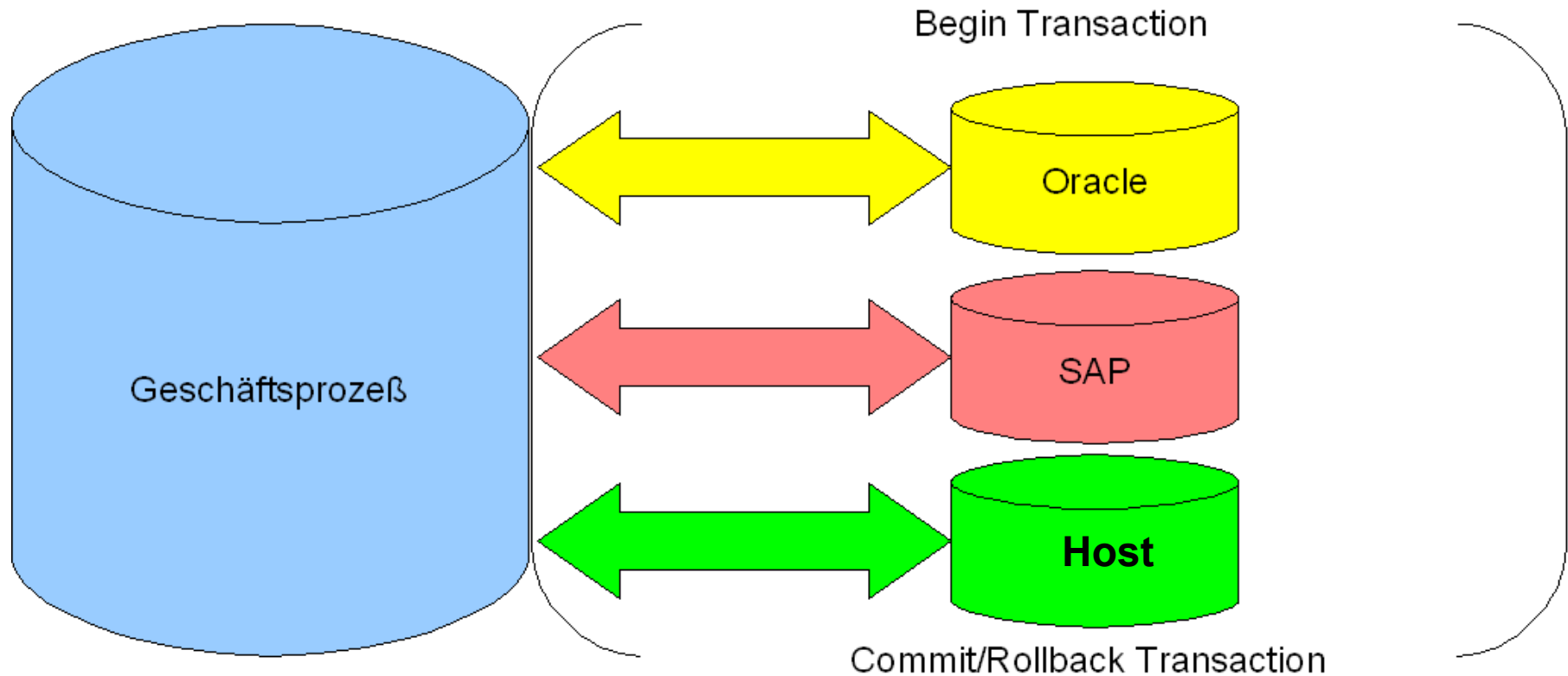
Beispiel 2: Business to Business



Beispiel 3: Interne und externe Anwendungen

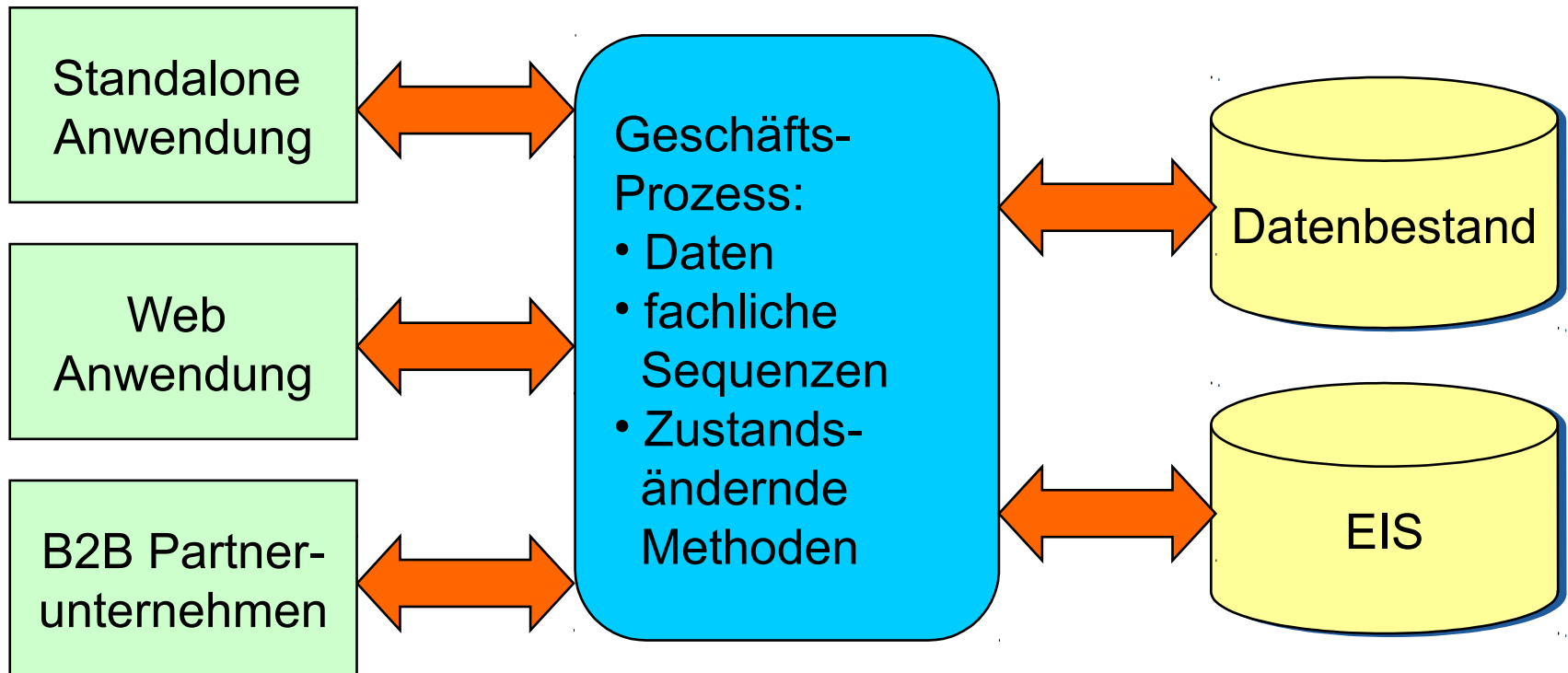


Beispiel 4: XA-Transaktionen



4.2

DIE VISION DER IDEALEN UMSETZUNG



- enthält möglichst wenige Code-Zeilen
- ist modular und flexibel aufgebaut
- ist Fehler-tolerant und ausfallsicher
- ist einfach zu testen
- kann von verschiedensten Anwendungen aufgerufen werden
- kann beliebig verteilt werden und ist 100%ig skalierbar
- ist selbst Transaktionsfähig und orchestriert Transaktionen konsistent über beliebige Backend-Systeme hinweg

- Der Applikationsserver nimmt dem Entwickler viele Aufgaben ab
 - Netzwerk
 - Skalierung/Cluster-Betrieb
 - Verwaltung von Ressourcen
- Deklarative Programmierung
 - Transaktionssteuerung
 - Security
 - Seiten-Navigation

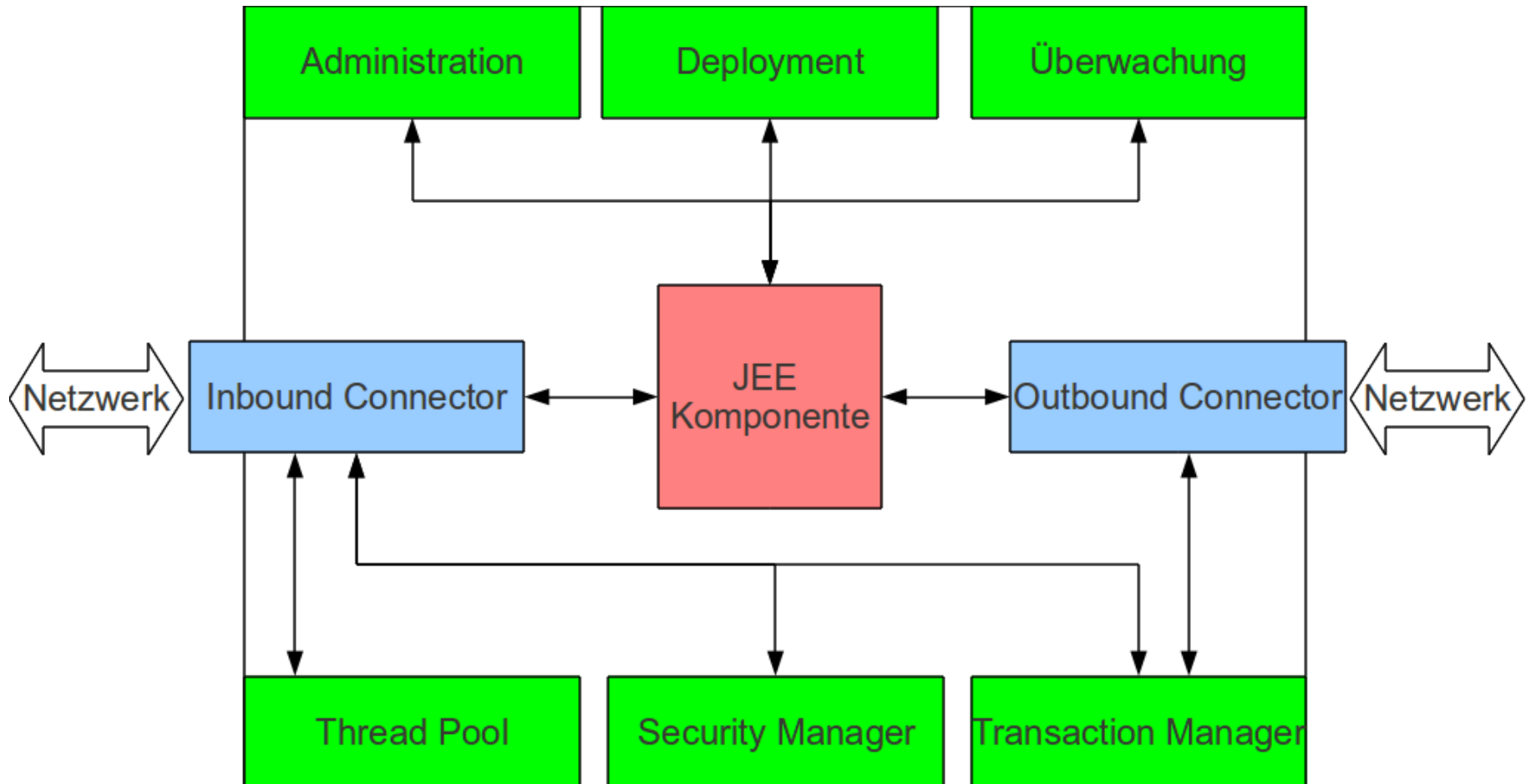
Die JEE mag nicht perfekt sein, ist aber definitiv ein großer Schritt in die richtige Richtung!

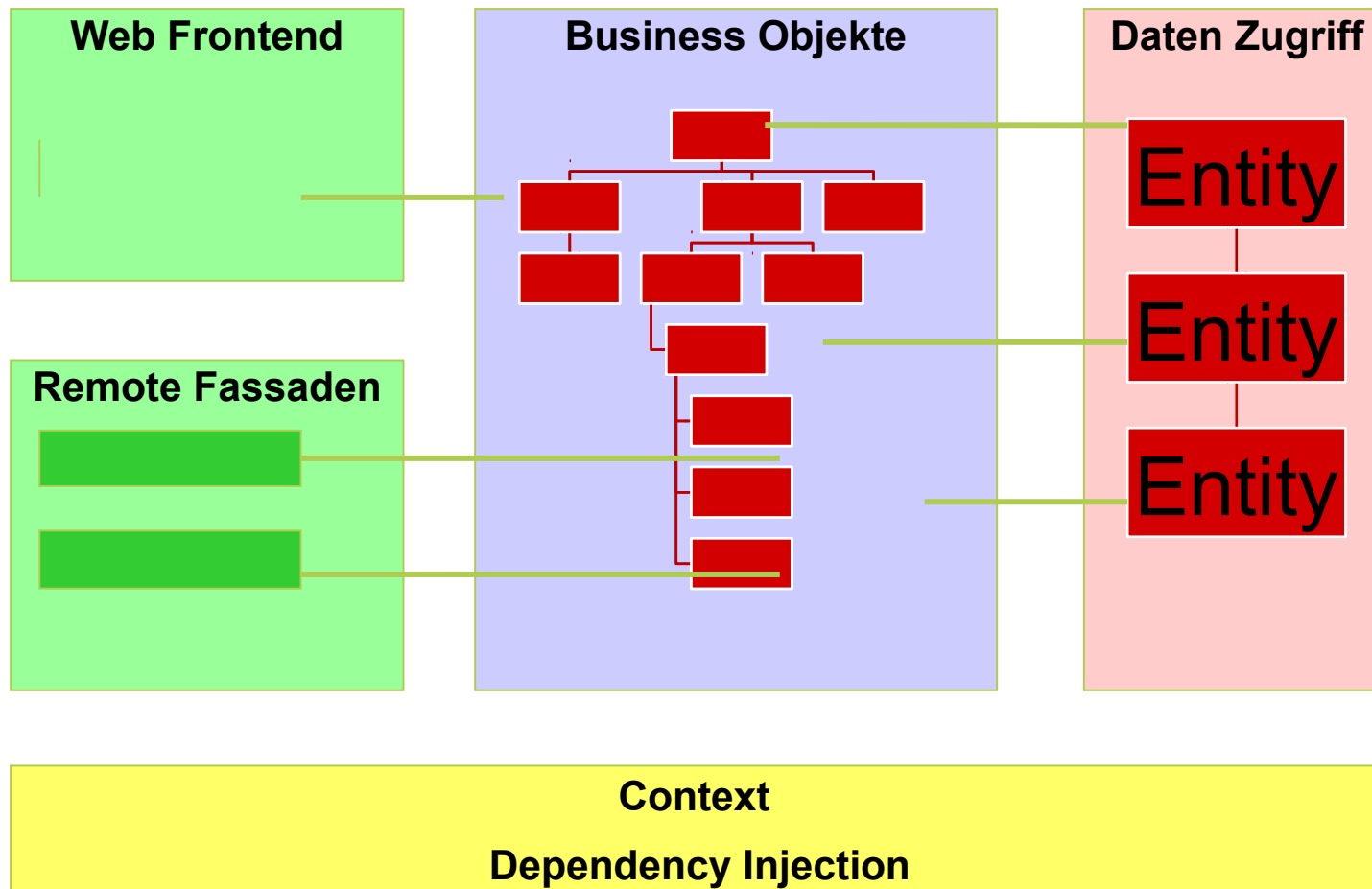
4.3

KRISE UND WIEDERAUFERSTEHUNG: DIE JEE IM WANDEL DER ZEIT

- Die Zahl der Projekt-Anfragen/Seminar-Teilnehmer, die den vollen Umfang der JEE benötigen, stagnierte bis 2009 auf etwa 25% des Maximalwerts von 2002
 - Unter Berücksichtigung allgemeiner Marktschwankungen
- Die JEE in der Version 1.4 wurde schon zum Zeitpunkt ihrer Veröffentlichung im November 2003 massiv kritisiert
 - „Umständlich, unzeitgemäß, unbrauchbar, ...“
 - Effiziente Anwendungsentwicklung war mit reinen JEE-Mitteln de facto unmöglich!
 - JEE Projekte benutzten große Mengen an proprietären Werkzeugen zur Erleichterung des Entwicklungsprozesses

JEE ist ursprünglich eine Laufzeitumgebung

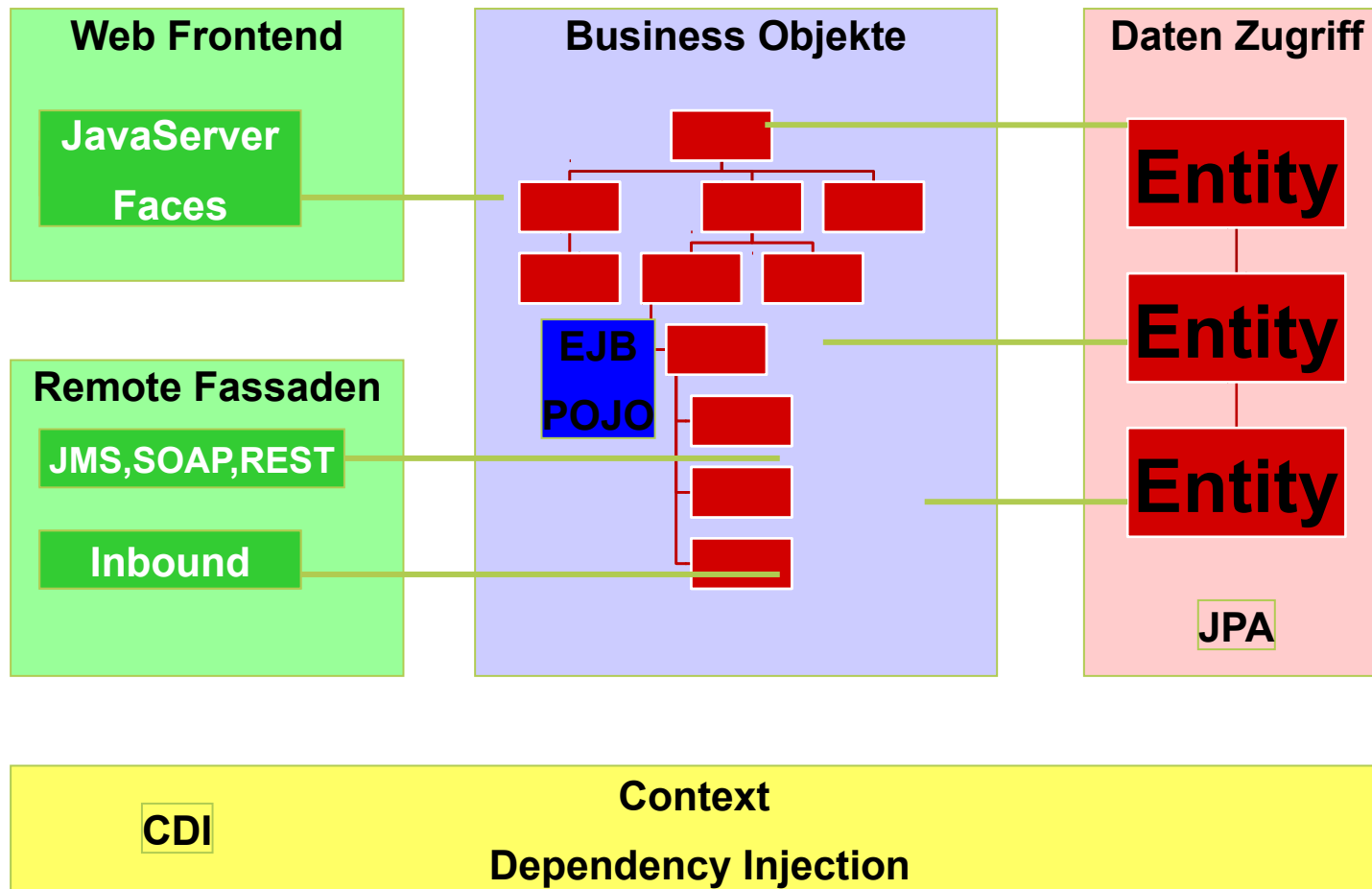




- Seit 2003 wurden eine Vielzahl von Frameworks entwickelt, deren Fokus alleine auf der Vereinfachung des JEE-Programmiermodells liegt
 - Web Frameworks wie Apache Struts, Code-Generatoren wie XDoclet
- Seit 2004 steht mit dem Spring Framework ein alternatives Programmiermodell zur Verfügung
 - In der Entwicklergemeinde als deutlich einfacher empfunden
- JBoss Seam übernahm auf Grund der Stagnation und langen Release-Zyklen der JEE die Weiterentwicklung
 - Allerdings stets darauf Bedacht, auf Standards zu achten!
 - JBoss Seam ist mittlerweile größtenteils in die JEE aufgegangen und wird nicht mehr aktiv weiterentwickelt
- Seit 2009 steht CDI (Context & Dependency Injection) zur Verfügung
 - Eigentlich eine neue Version der Java Enterprise Edition
 - Allerdings kompatibel mit den alten Versionen

- Das Spring-Framework der springsource.org ist ein Open Source Framework, das aus sehr vielen Modulen besteht
- Die Core-Komponente ist ein hervorragendes Dependency-Injection- und AOP-Framework
 - Und deshalb prinzipiell zur Ergänzung der JEE wunderbar geeignet!
- Spring positioniert sich selbst jedoch recht aggressiv als Alternative zur JEE
 - Ein eigenes Web Framework
 - Mit dem Spring tc-Server (einem erweiterten Server) und dem dm-Server (OSGi-konformes Deployment) werden eigene Laufzeitumgebungen definiert.
- Spring ist Implementierungsgetrieben, es gibt keine Spezifikation!
 - Ein erster (?) Versuch der Springsource zur Kommerzialisierung durch die Einführung einer geschlossenen „Supported Version“ ist am massiven Protest der Community gescheitert
 - Spring enthält wie das Negativ-Beispiel Struts 1.x bereits eine ganze Menge von „historisch gewachsenen“ Bibliotheken, die nicht mehr benutzt werden sollten

- Spätestens mit der im Jahre 2009 veröffentlichten JEE 6 ist es jedoch gelungen, einen Großteil der proprietären Frameworks wieder einzufangen
 - Datenbankzugriffe und O/R-Mapping mit dem **Java Persistence API 2**
 - Das Web Framework **JavaServer Faces 2** mit den Facelets Templates
 - Mit den Interceptors können Querschnittsfunktionen realisiert werden
 - Dependency Injection mit der **Context and Dependency Injection-Bibliothek**
- Damit wird die ursprüngliche Spezifikation der Laufzeitumgebung um ein komfortables Programmiermodell ergänzt!
 - Dies wird auch von den Entwicklern honoriert: Es ist ein deutlicher Trend zurück zur JEE zu beobachten!



- 2003
 - Java Enterprise Edition, Version 1.4
 - Direkte Datenzugriffe mit „Bean Managed Persistence“ und Apache Torque
 - Web Frontend mit Servlets und JavaServer Pages
- 2004
 - Erster Umstieg auf Hibernate
 - Code-Generierung mit XDoclet
 - Gescheiterter Umstieg auf JavaServer Faces, statt dessen Apache Struts
- 2005
 - Design-Änderung auf Dependency Injection, erste Integration von Spring
 - Generische SessionBeans als Fassaden
 - Web Services mit Apache Axis
- 2006
 - Migration auf JEE 5
 - Umstellung auf MyFaces
 - Kompletter Umstieg auf Hibernate

- 2007
 - Aufsplittung in eine Variante mit EJBs und eine Variante mit Spring
 - Teilweiser Rückbau von Hibernate auf Java Persistence API
- 2008
 - Integration von AJAX-Funktionalität mit Ajax4JSF bzw. RichFaces
 - Umstellung auf Apache CXF
- 2009
 - Weitere Aufsplittung in eine JBoss Seam-Variante
 - Teilweise Umstellung auf JAX-WS

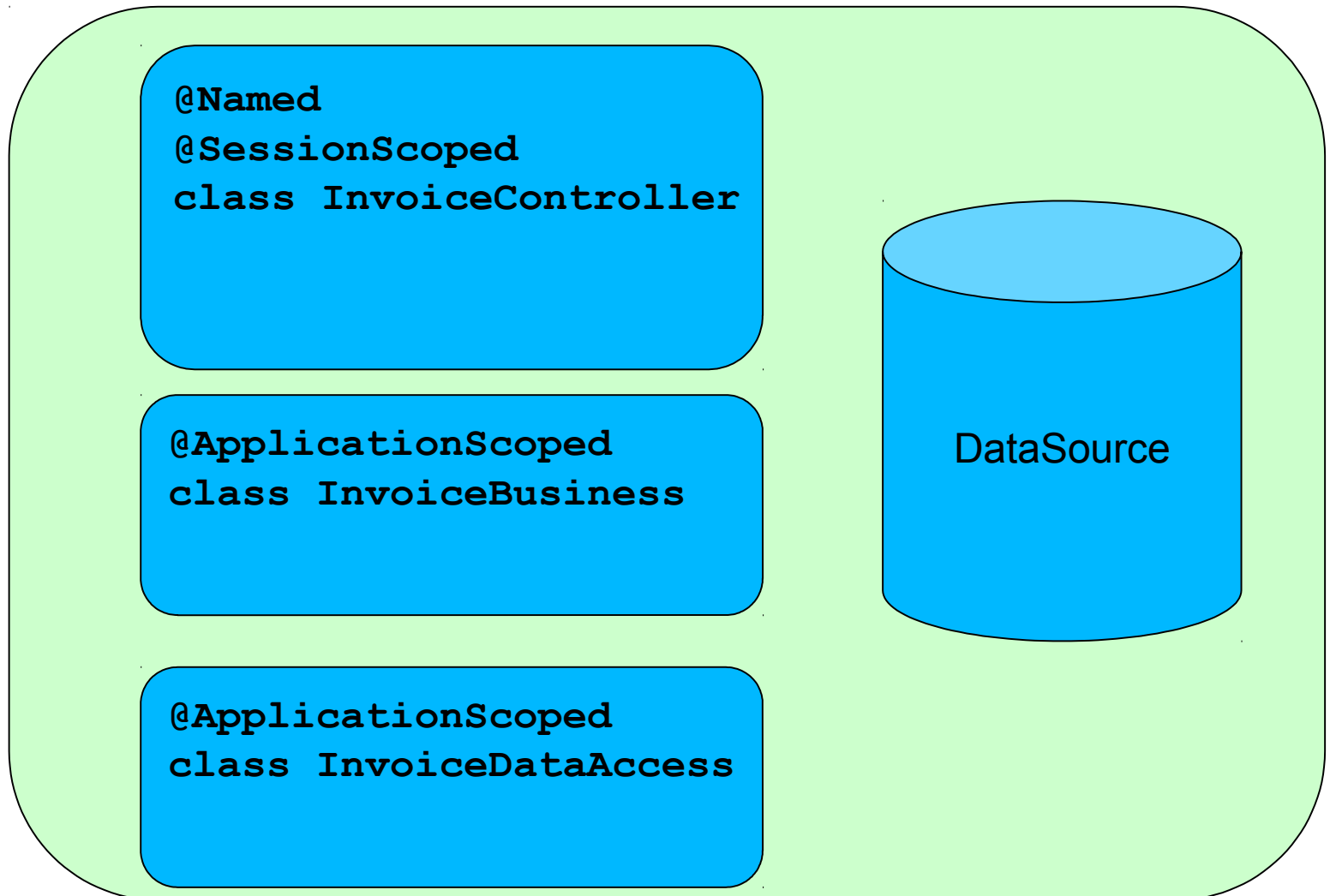
- 2010
 - Umbau auf JEE 6
 - Kompletter Verzicht auf Hibernate API
 - Teilweiser Rückbau von RichFaces auf JSF 2.0
 - RESTful Aufrufe mit JAX-RS
 - Die Anwendung ist zum ersten mal praktisch unabhängig von externen Bibliotheken!
- 2013
 - Konsequente Benutzung von CDI auch zur Transaktionssteuerung
 - Fast alle EJBs sind verschwunden

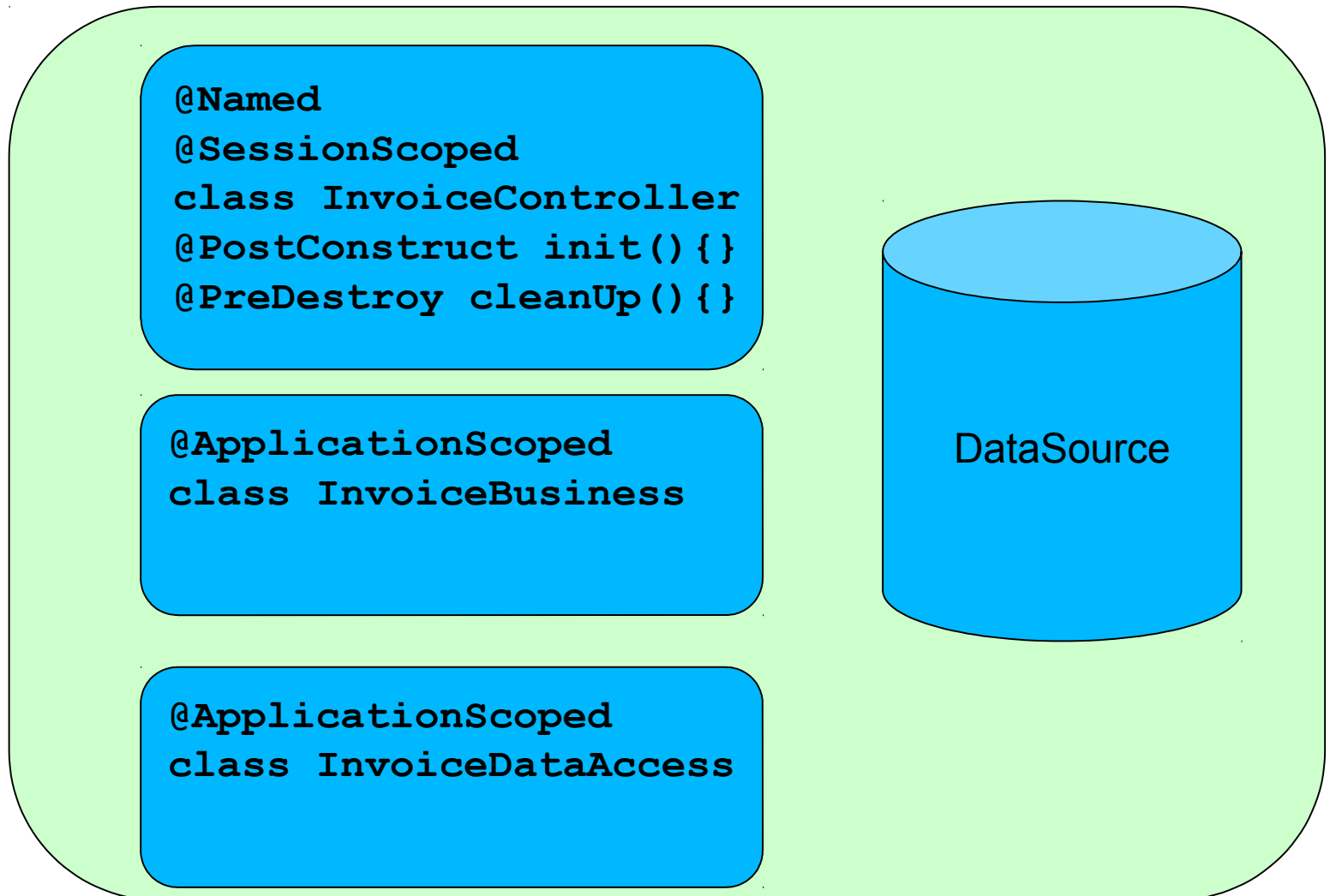
5

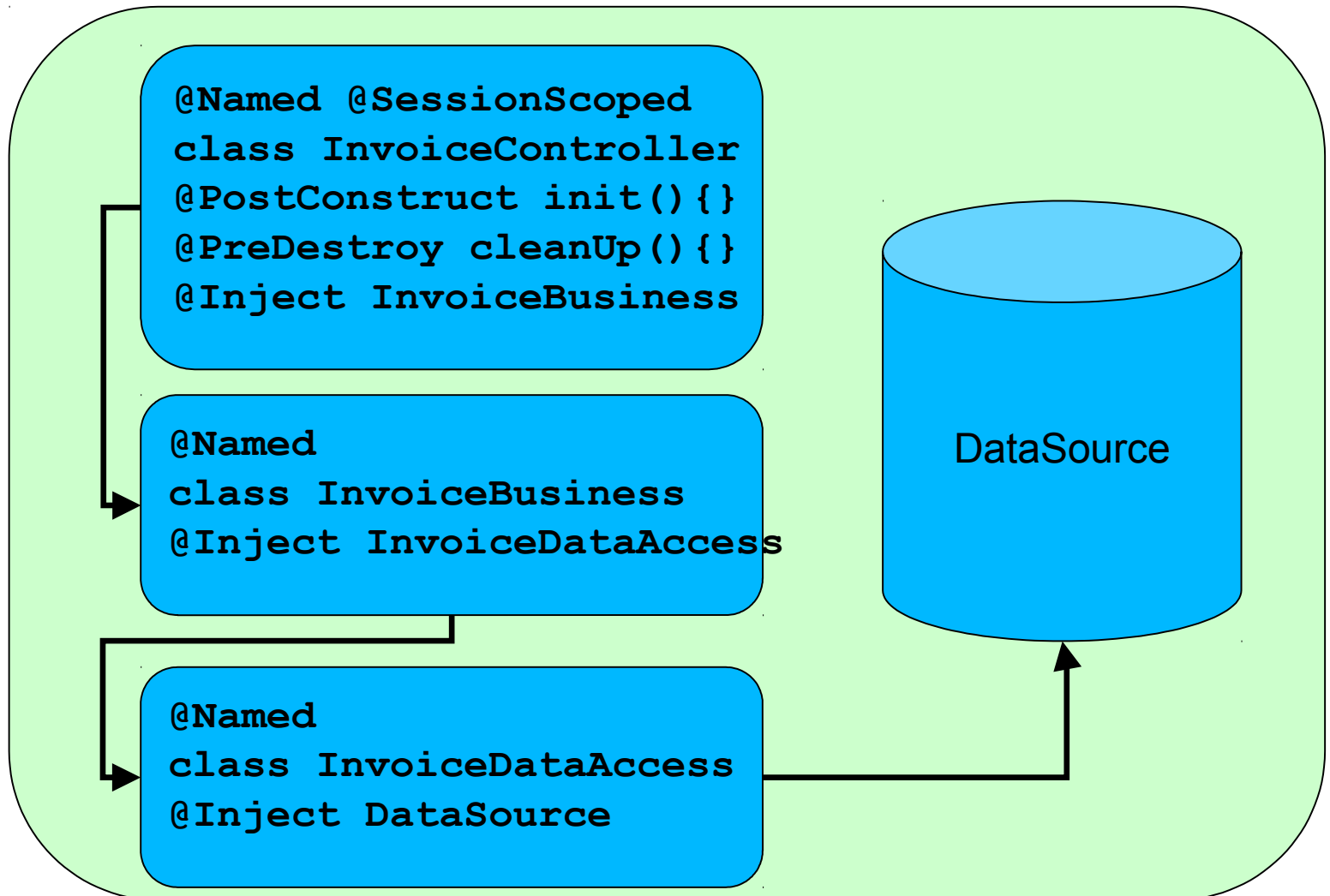
DAS PROGRAMMIERMODELL

5.1

CONTEXT & DEPENDENCY INJECTION

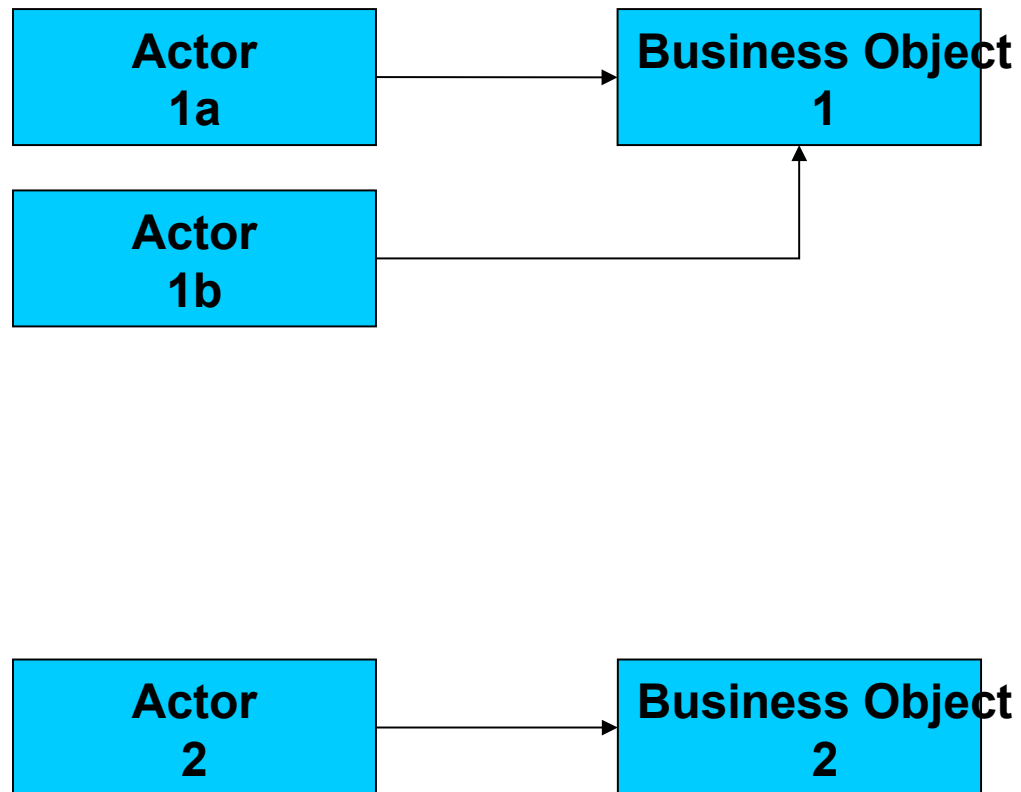


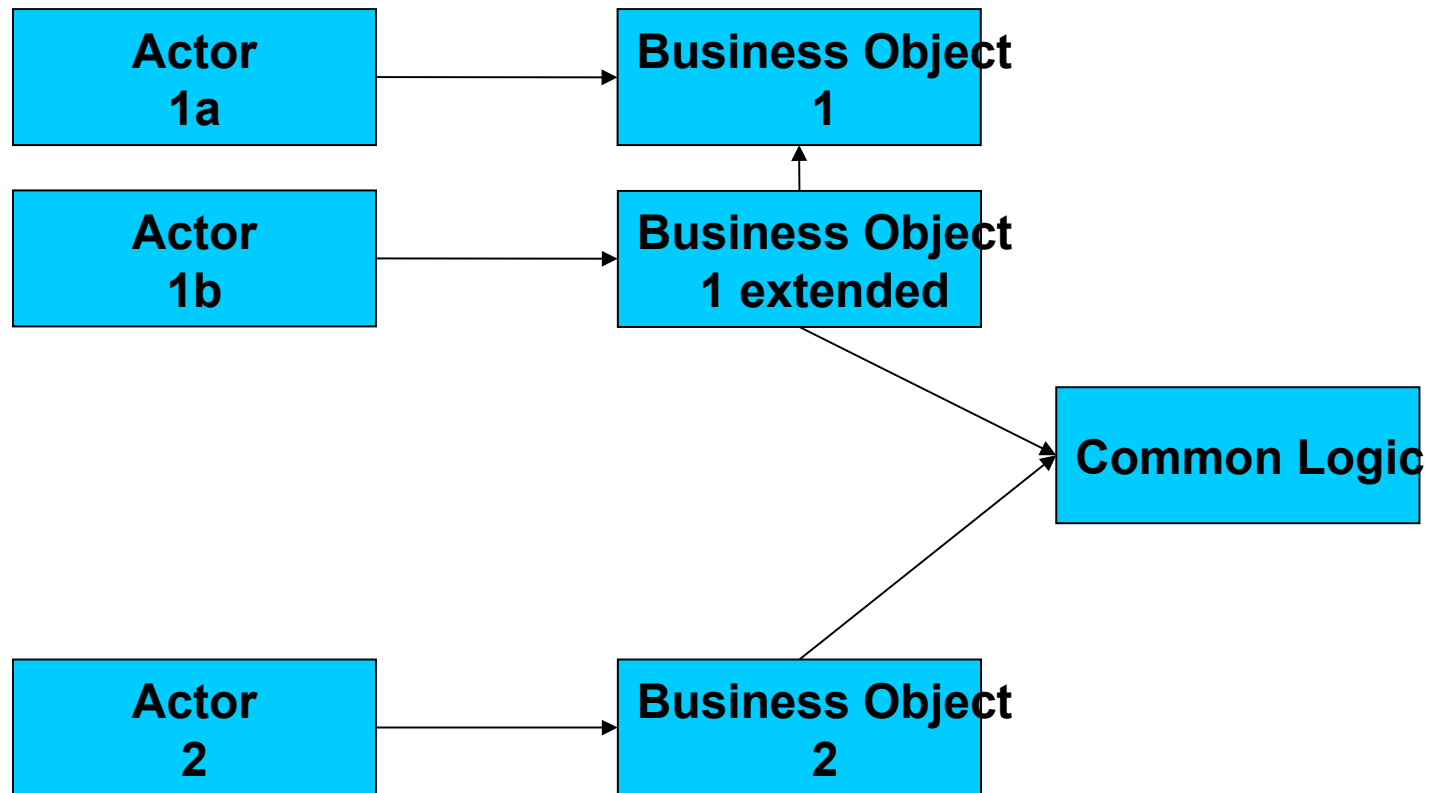


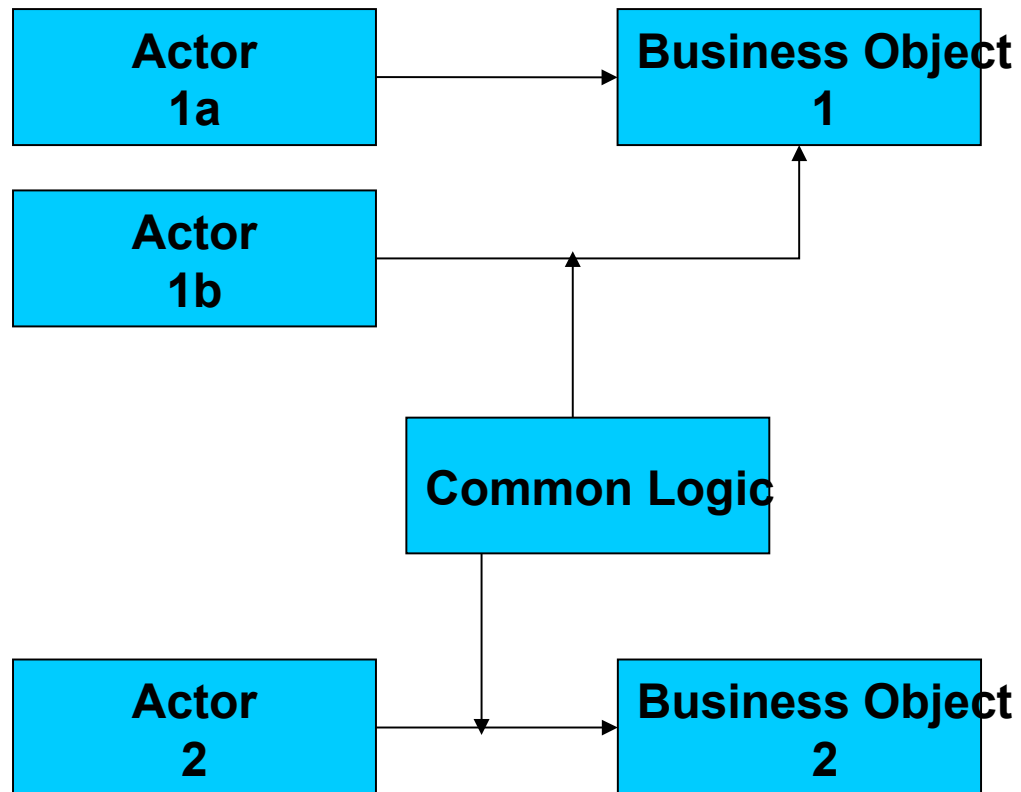


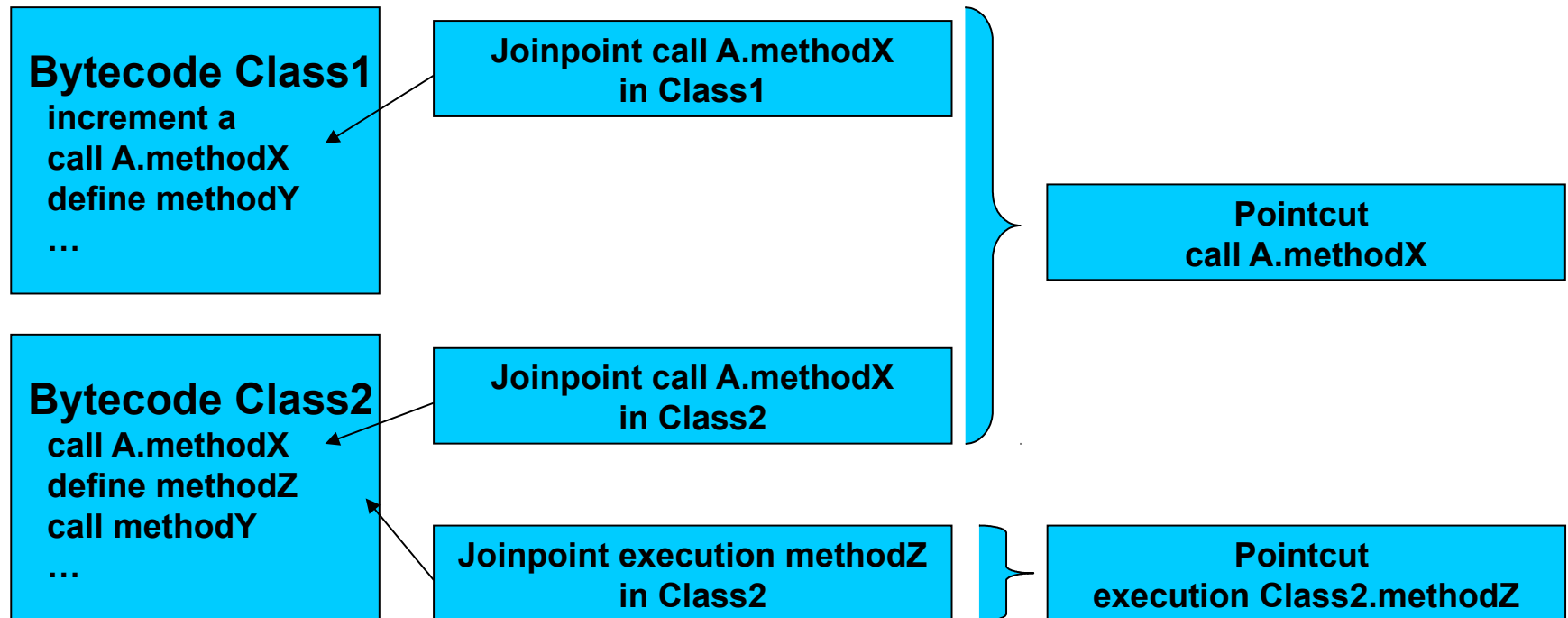
5.2

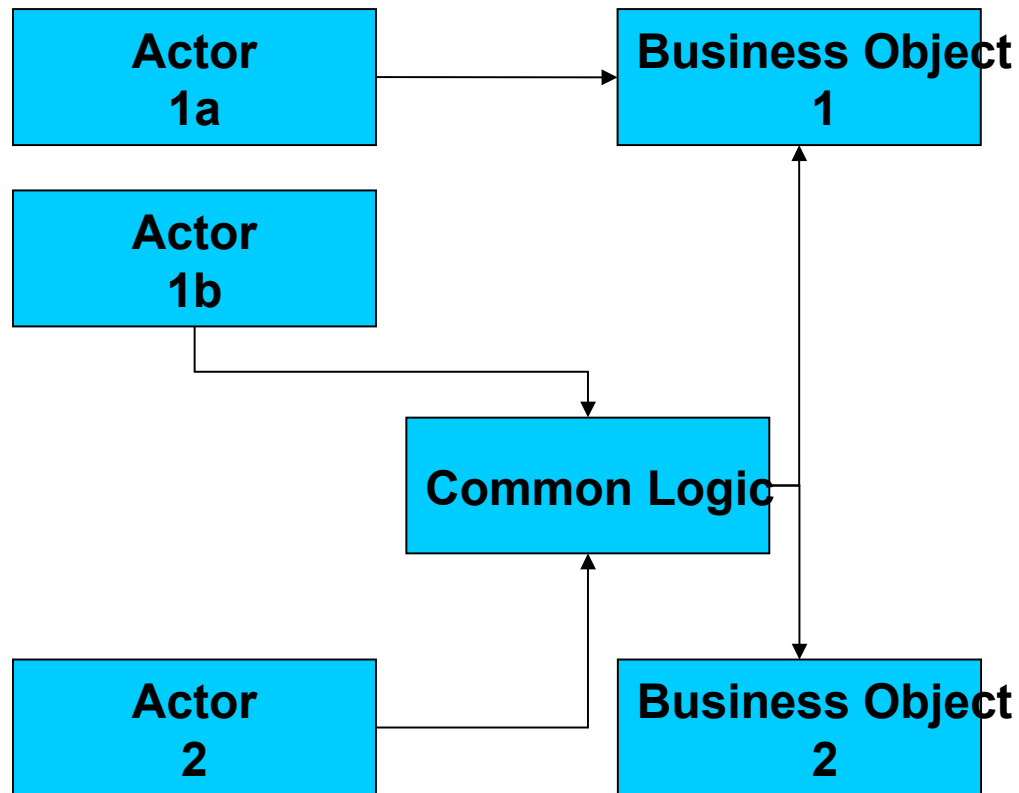
ASPEKTORIENTIERTE PROGRAMMIERUNG











- Servlet Filter
 - Definition in der web.xml
 - Filter werden über ein Servlet- oder URL-Mapping den Seiten der Web Anwendung zugeordnet
- EJB Interceptors
 - Formulierung mit AOP-Pointcut-Syntax
 - Konkrete Realisierung ist abhängig vom Provider
- CDI-Interceptors
 - Analog zu den EJB Interceptors
 - Keine Abhängigkeit zur JEE

5.3

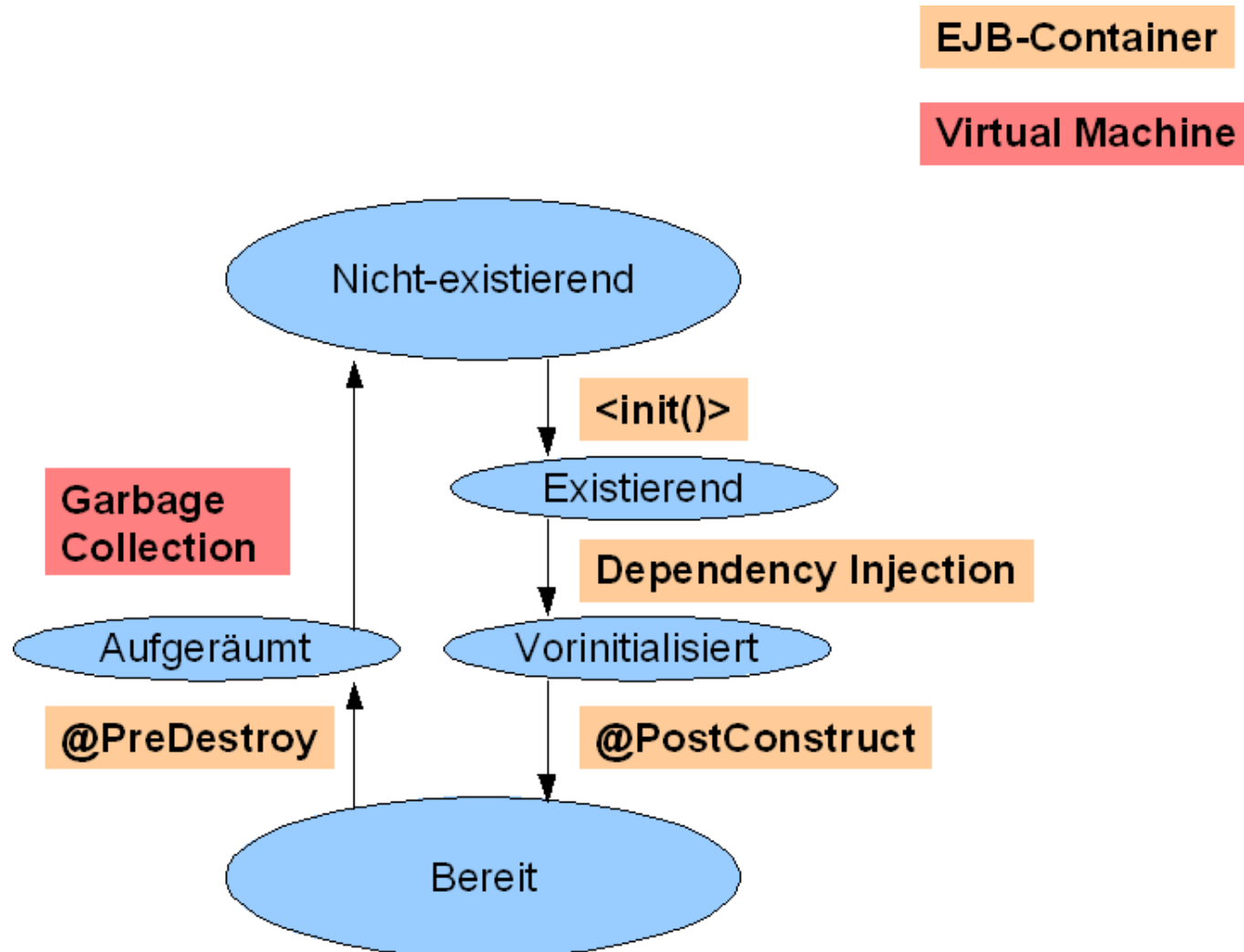
ENTERPRISE JAVABEANS ALS FACHOBJEKTE

- Definieren das Transaktionsverhalten der Anwendung
- Definieren und Prüfen die Anwendungs-Rollen
- Rufen die eigentliche Geschäftslogik auf
- Können bei Bedarf an einen Inbound Connector gekoppelt und so über das Netzwerk aufgerufen werden
- Eine Stateful SessionBean kann eingesetzt werden, um bei Bedarf Client-Zustand im Server zu halten

Die Aufgaben von Enterprise JavaBeans überschneiden sich mit CDI. Eine Interoperabilität ist jedoch gewährleistet!

- Stateless SessionBeans
 - Gruppieren Funktionen
 - Können bei Bedarf über Java RMI oder SOAP angesprochen werden
 - Mehrere Instanzen werden in einem Pool verwaltet

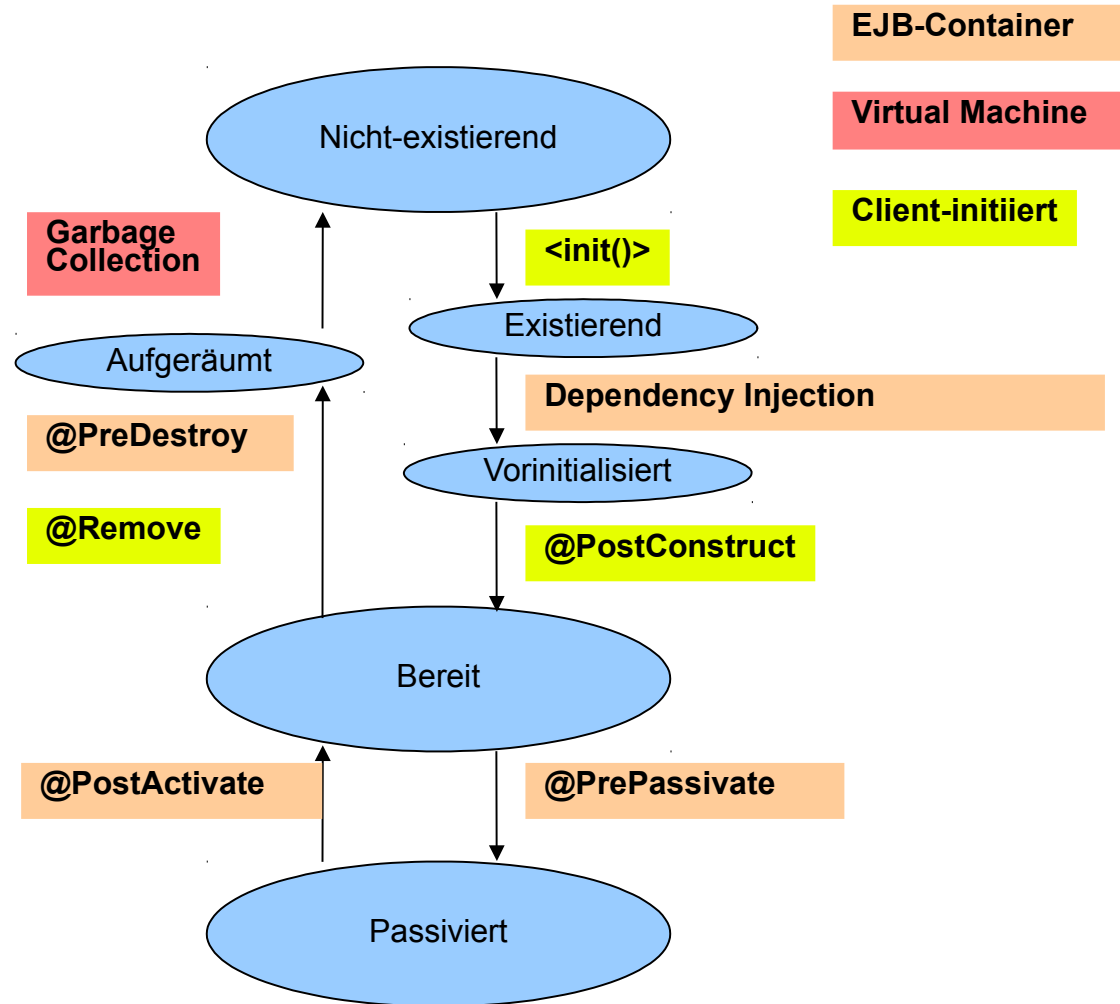
- MessageDriven Beans
 - Sind Listener an einer JMS-Destination
 - Mehrere Instanzen werden in einem Pool verwaltet



- Singleton Beans
 - Gruppieren Funktionen
 - Existieren exakt einmal pro Anwendung und Applikationsserver
 - Können von verschiedenen Clients gleichzeitig benutzt werden

- Stateful SessionBeans
 - Gruppieren über ein Interface Funktionen
 - Können bei Bedarf über Java RMI oder SOAP angesprochen werden
 - Haben einen Lebenszyklus, der vom Client gesteuert wird
 - Gleichzeitiger Zugriff mehrere Clients gleichzeitig ist nicht möglich

Lebenszyklus einer Stateful SessionBean



5.4

ENTERPRISE JAVABEANS UND SERVLETS IM DETAIL



- Begriffe
 - Instance Swapping
 - Instance Pooling
- Aufruf:
 - lokal
 - RMI
- Verwendung
 - Aufruf von Geschäftsprozessen, die alle notwendigen Informationen als Parameter bekommen
- Einsatz
 - Uneingeschränkt geeignet
 - Sofort Cluster-fähig und ausfallsicher

- Begriffe
 - Instance Swapping
 - Instance Pooling
- Aufruf:
 - JMS
- Verwendung
 - Aufruf von Geschäftsprozessen, die alle notwendigen Informationen als Nachricht bekommen
 - Client erwartet (wenn überhaupt) Antwort über Callback
- Einsatz
 - Uneingeschränkt geeignet
 - Sofort Clusterfähig und Ausfallsicher

- Begriffe
 - Eine Instanz pro Anwendung
 - Potenziell gleichzeitiger Zugriff muss berücksichtigt werden
- Aufruf:
 - lokal
 - RMI
- Verwendung
 - Als globaler Zwischenspeicher (Daten-Cache) auf der Serverseite
- Einsatz
 - Weniger Overhead als Stateless SessionBeans
 - Multithreading muss beachtet werden

- Begriffe
 - Aktivierung und Passivierung
 - Timeout
 - Session Replizierung
- Aufruf:
 - RMI
 - lokale
- Verwendung
 - Als Zwischenspeicher (Daten-Cache) auf der Serverseite
- Einsatz
 - Eingeschränkt geeignet: Session-Problematik im Cluster
 - Relativ hoher Konfigurationsaufwand
 - Geringere Effizienz als Stateless SessionBeans: Eine Instanz pro Client erforderlich

- Begriffe
 - Session
 - Timeout
 - Session Replizierung
- Aufruf:
 - http
- Verwendung
 - Als Controller für Web Anwendungen
- Einsatz
 - Als technische Komponente für Web Anwendungen unabdingbar

5.5

WEB ANWENDUNGEN

- Alle benötigten Klassen müssen in einem Java-Archiv spezieller Struktur abgelegt werden
 - .war-Dateien
- Spezielles Verzeichnis: WEB-INF
 - Im classes-Verzeichnis sind alle nichtgepackten Klasse
 - lib-Verzeichnis für benötigte Bibliotheken
 - Web-Deskriptor web.xml
- Zusätzlich an beliebiger Stelle weitere Ressourcen
 - Statische Elemente wie HTML-Seiten, Bilder...
 - Dynamische JSP-Seiten

- Eine Start-Seite
- Ein Servlet mit der Verarbeitungslogik
- Eine JavaBean, die das Ergebnis hält
- Eine JSP zur Darstellung der Ergebnisse
- Deskriptor
 - web.xml

Für komplexere Anwendungen ist dieser Ansatz nicht mehr zeitgemäß!

6

JAVA SERVER FACES

6.1

ÜBERBLICK

- Historie
 - Beginn der Spezifikation im Jahr 2001 (JSR-127) (ASF, BEA, Borland, HP, IBM, Novell, Oracle, Sun)
 - März 2004: Final Release Version 1.0 für J2EE 1.4
 - Mai 2006: JSF Version 1.2 als Bestandteil von Java EE 5
 - Dez. 2009: JSF Version 2.0 als Bestandteil von Java EE 6
 - Zwischen Java EE 6 und 7 wurde ein "maintenance-release" JSF 2.1 freigegeben.
 - 2014: JSF Version 2.2 als Bestandteil der JEE 7

- Die Hersteller von Applikationsservern und Komponentenbibliotheken verwenden mittlerweile bevorzugt die Referenzimplementierung

Mojarra

- Oracle
 - GlassFish
 - WebLogic
- Erweiterungen sind jedoch immer noch sinnvoll!
 - JBoss RichFaces 4
 - ICEFaces 2
 - PrimeFaces
 - Apache Tomahawk/Trinidad
 - ...

- Objekt-orientierter Programmier-Ansatz
 - „Managed Beans“ halten den Zustand der Web-Anwendung und definieren über Actions ihr Verhalten
- Feingranulare Scopes für die Verwaltung des Datenmodells der Web Anwendung
 - Damit vereinfachte Verwendung der Session
- Ausgefeilter Zyklus zur Verarbeitung eines Requests
 - Integration von Validierung und Konvertierung
 - Die Anwendungsprogramme benötigen das http-nahe Servlet-API nur noch in Ausnahmefällen
- Server-seitige UI-Komponenten mit Event-basiertes Programmiermodell
 - Die im Browser dargestellte Oberfläche steht auf Serverseite ebenfalls zur Verfügung

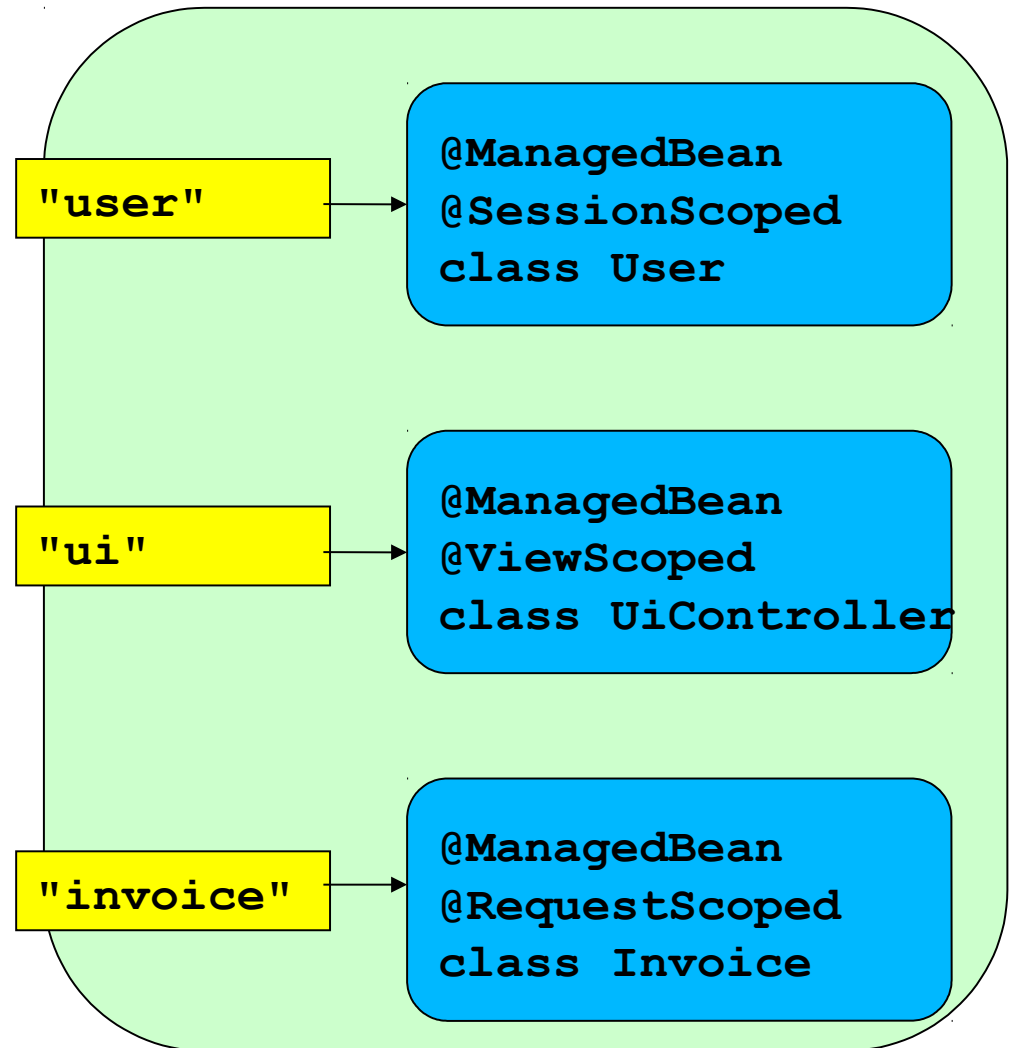
- Page Flows
 - Komplexe Anwendungen benutzen mächtige Navigation Rules
 - Einfachere Anwendungen definieren im Programm direkt die nächste darzustellende Seite
- Seiten-Rendering mit Facelets-Templating
 - Ein Facelet-Layout besteht aus einer festen Seiten-Definition mit eingefügten Platzhaltern
 - Eine konkrete Facelet-Seite wählt ein Layout und setzt die Platzhalter
- Transparente AJAX-Unterstützung durch JSF-Tags
 - Interaktive Benutzerelemente können ohne jede JavaScript-Programmierung eingesetzt werden
 - AJAX-JSF-Komponenten generieren das benötigte JavaScript intern selber
 - Mapping von JavaScript-Events auf JSF-Aktionen
- Direkte Aufrufe von JSF-Aktionen
 - Bei Bedarf kann ein JavaScript-Entwickler eine JSF-Bibliothek benutzen und so einen JavaScript-Java-Aufruf durchführen

6.2

DAS WEB DATENMODELL

- Die Clients der allermeisten Web-Anwendungen legen Informationen auf dem Server ab
 - Die Alternative der Client-seitigen Speicherung ist noch unbefriedigend
 - Cookies
 - Hidden Fields
 - Interessant: Das HTML5-Repository
- Managed Beans halten diesen Zustand auf dem Server
 - JSF synchronisiert automatisch Benutzer-Eingaben mit den Managed Beans
- Jede Managed Bean wird durch einen eindeutigen Namen identifiziert
 - Wichtig für das Rendern einer Seite!
- Die Lebensdauer von Managed Beans wird durch die Angabe von Scopes definiert

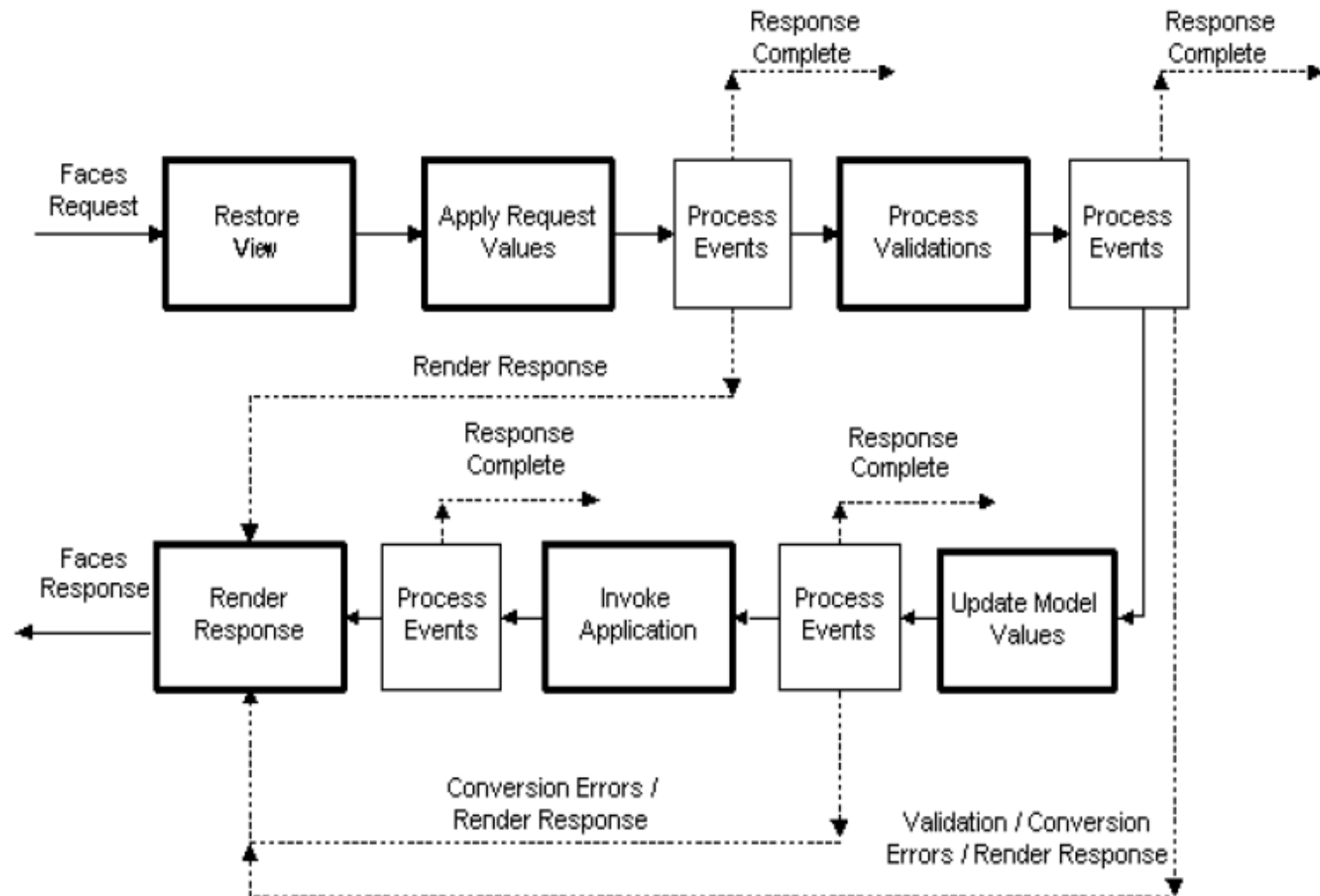
- Application Scope
 - Gültig für die gesamte Zeit der Anwendung
- Session Scope
 - Gültig für die Benutzer-Session
 - „Login – Logout“
- Conversation Scope
 - Gültig für einen Arbeitsablauf
 - „Rechnung erstellen“, „Kunden-Daten aktualisieren“
- View Scope
 - Gültig, so lange keine Seiten-Navigation erfolgt
- Flash Scope
 - Gültig für einen Request-Redirect-Request-Zyklus
- Request Scope
 - Gültig für einen Request



- Die eben dargestellten Abläufe erinnern sehr stark an CDI
 - JSF enthält seine eigene Implementierung eines CDI-Frameworks
- JSF ist CDI-konform und interoperabel
- Mittelfristig werden die Managed Beans verschwinden

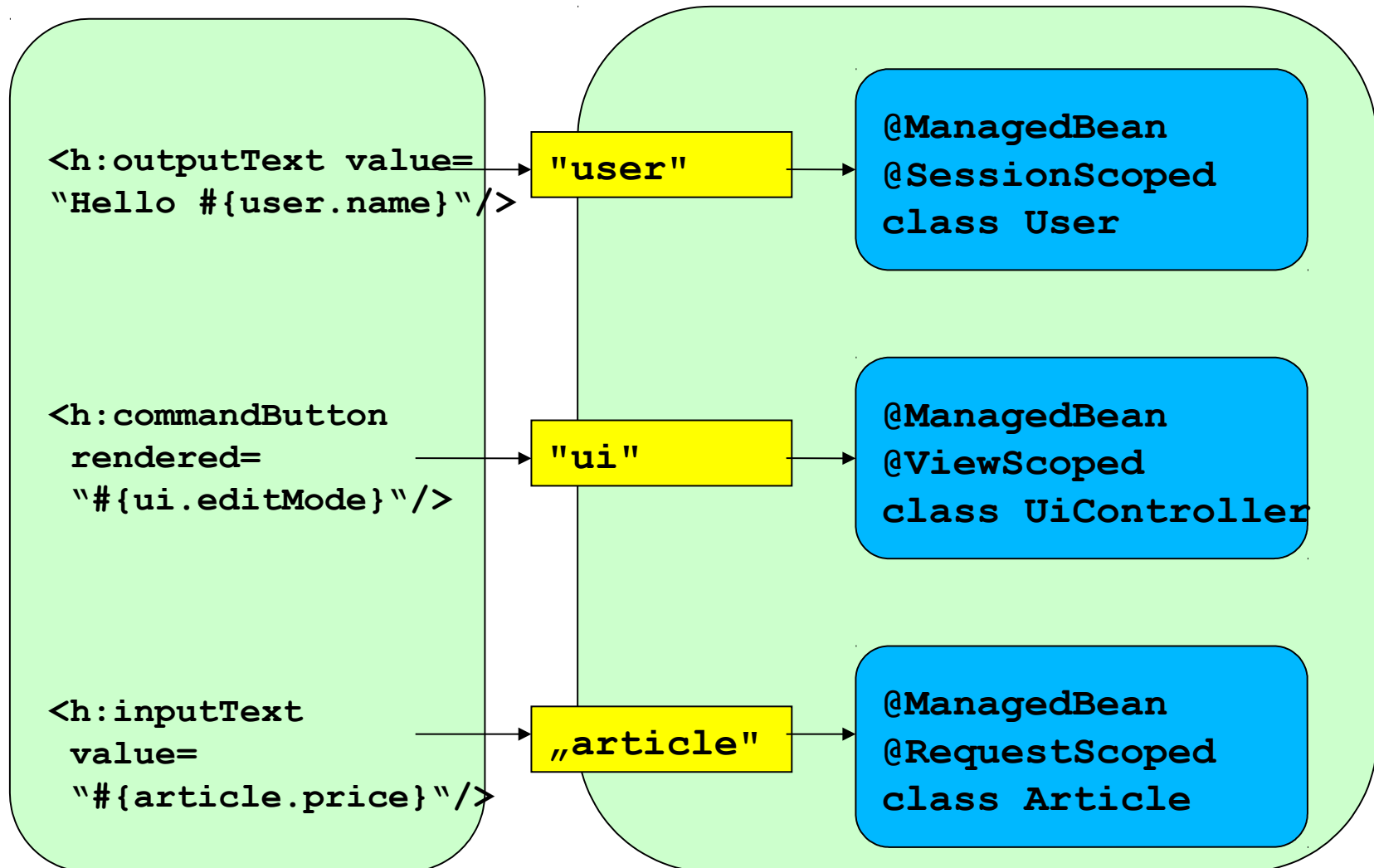
6.3

REQUEST-VERARBEITUNG



6.4

RENDERING



Hello Hugo

`<input type="button"/>`

`<input type="text"
value="1.99"/>`

`<h:outputText value=
"Hello #{user.name}"/>`

`<h:commandButton
rendered=
"#{ui.editMode}"/>`

`<h:inputText
value=
"#{article.price}"/>`

"user"

@ManagedBean
@SessionScoped
class User

"ui"

@ManagedBean
@ViewScoped
class UIController

"article"

@ManagedBean
@RequestScoped
class Article

6.5

FACELETS

- Eine normale HTML-Seite mit Facelet-Platzhaltern
 - Diese Seite ist „abstrakt“ und sollte nicht direkt dargestellt werden

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ui="http://java.sun.com/jsf/facelets">
<head>
  <title>
    <ui:insert name="title">Default title</ui:insert>
  </title>
  <link rel="stylesheet" type="text/css" href="./css/main.css"/>
</body>
  <ui:insert name="content"></ui:insert>
</body>
</head>
```

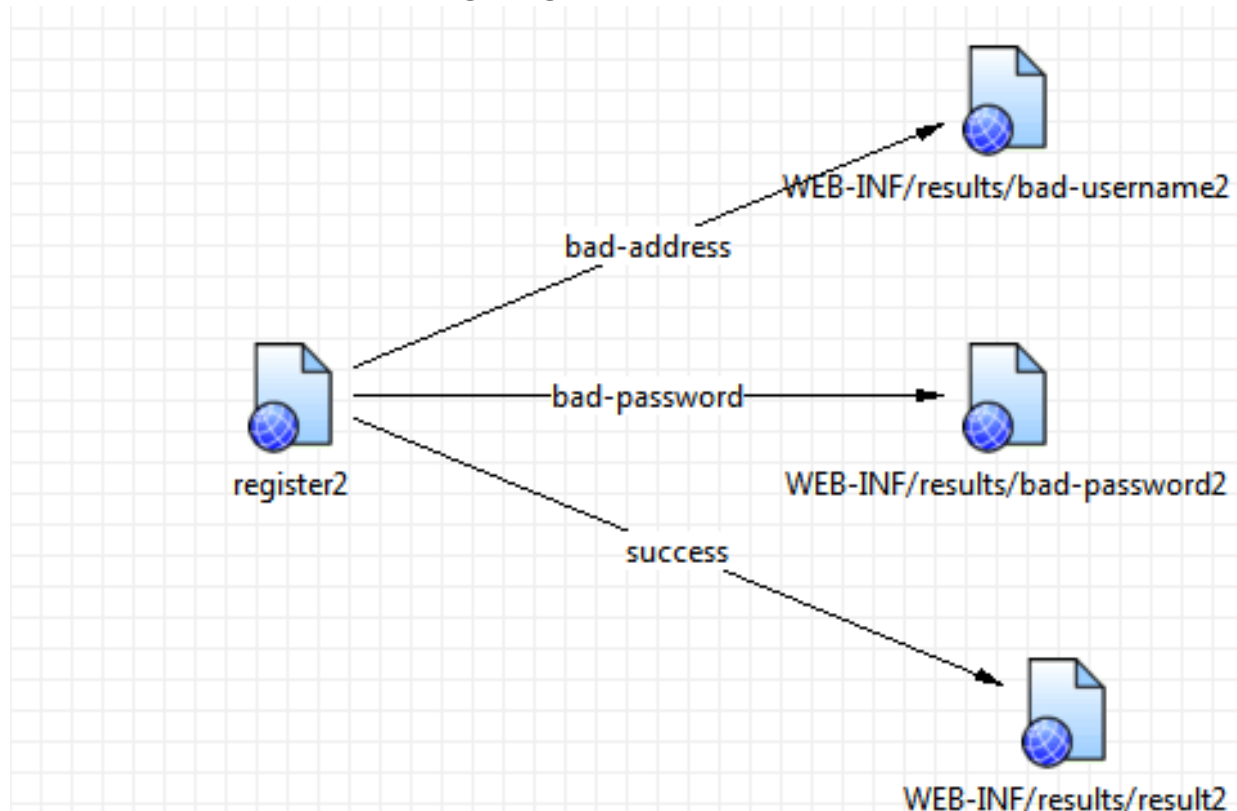
- Eine Ui-Composition definiert die Platzhalter
 - Und wird dann zum Browser gerendered

```
<ui:composition template="/WEB-INF/layout/layout.xhtml"> <ui:define  
name="title">Articles</ui:define>  
<ui:define name="content">  
    <h:form id="articleForm">  
        ...  
        ...  
        ...  
    </h:form>  
</ui:define>  
</ui:composition>
```

6.6

SEITEN-NAVIGATION

- Seiten werden im Page-Flow mit verschiedenen Navigation Rules verknüpft
 - Vergleichbar einem Workflow
 - Der konkrete benutzte Ausgang wird zur Laufzeit evaluiert

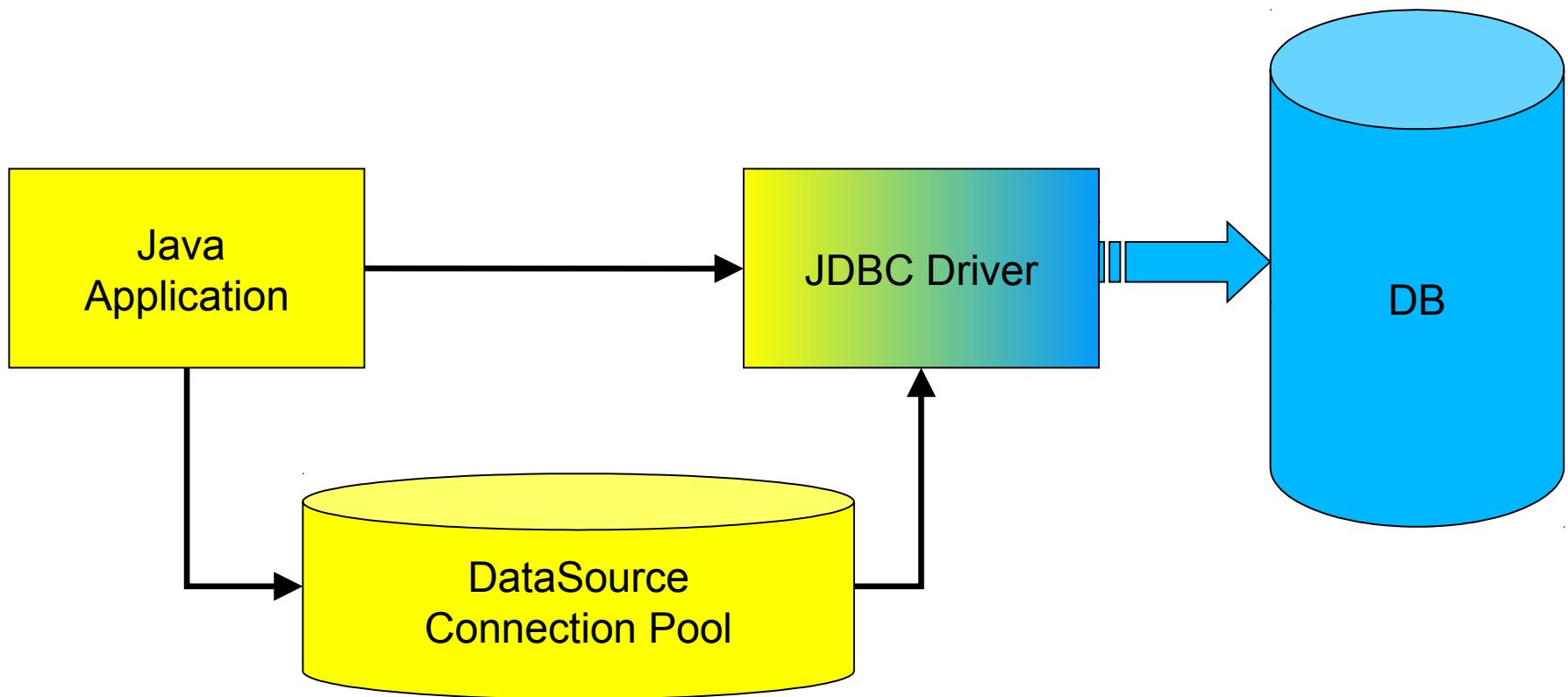


7

DATENBANK-ZUGRIFF

7.1

NATIVER SQL-ZUGRIFF



- Eine konfigurierte `DataSource` enthält einen Connection Pool zur Datenbank
- Die Java-Anwendung
 - holt sich von der `DataSource` eine Connection
 - erzeugt die benötigten Statements
 - setzt diese ab
 - wertet die Ergebnisse aus
 - `ResultSet`
 - `SQLException`
 - schließt die Connection
 - diese wird in den Pool zurückgegeben, nicht real geschlossen!

7.2

O/R-MAPPER

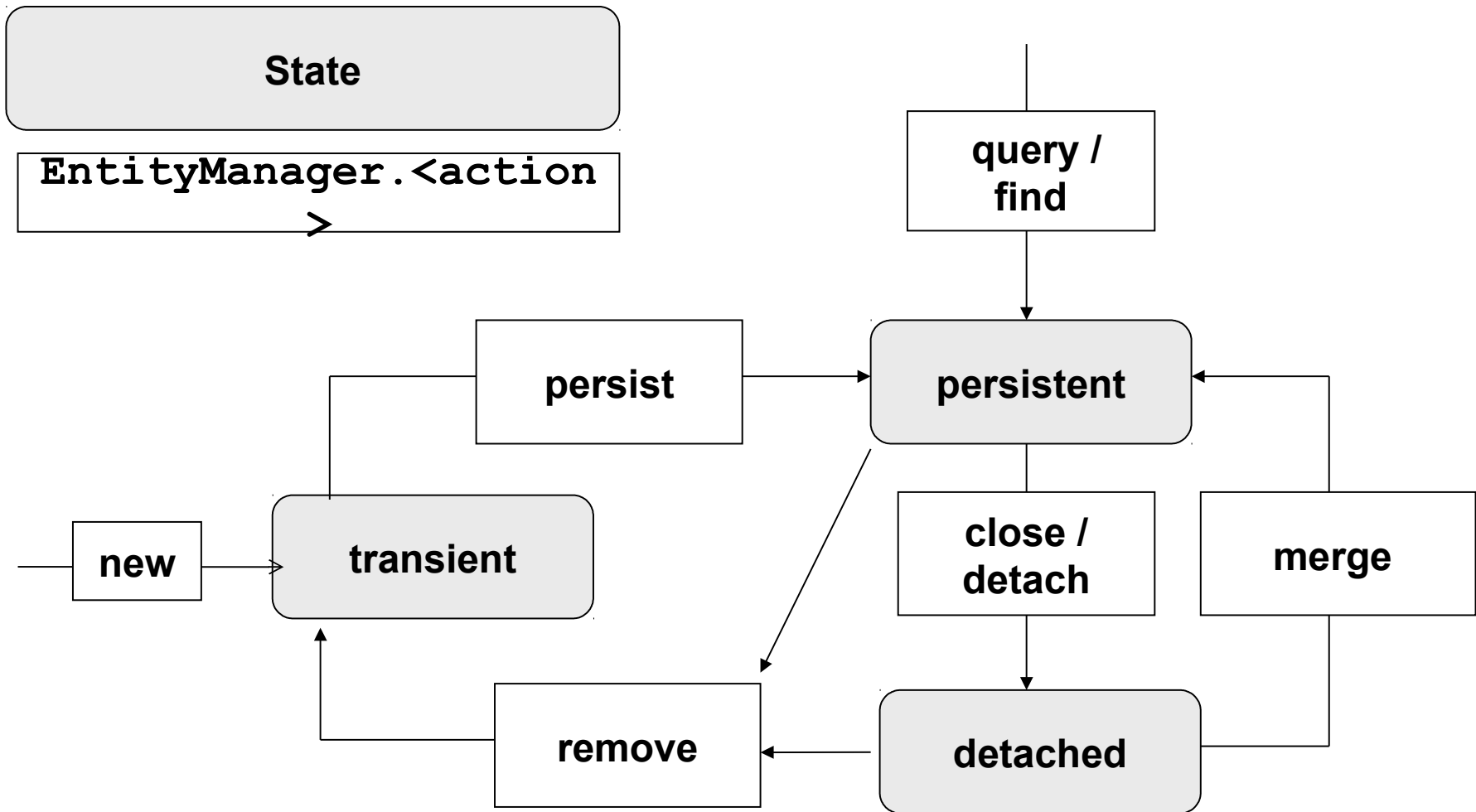
- Die Objekte der Objektorientierung leben im Hauptspeicher und sind somit "flüchtig".
 - Die "Objekte" von Datenbanken sind persistent.
- Die Objektorientierung kennt "intelligente" Objekte – Objekte, die ihren Zustand kapseln. Die Klassen dieser Objekte enthalten Methoden, mittels derer der Zustand der Objekte abfragbar und manipulierbar ist.
 - Relationale Datenbanken dagegen enthalten nur "dumme" Daten.
- Relationale Datenbanken kennen andere Typen als objektorientierte Sprachen.
 - Die Datenbank kennt z.B. die Typen CHAR und VARCHAR; Java dagegen kennt den Typ String.
- In der objektorientierten Welt sind Objekte miteinander über Referenzen (also Pointer) verbunden.
 - In relationalen Datenbanken werden die "Objekte" über Fremdschlüssel-Beziehungen miteinander verbunden. Die Objektorientierung kennt aber keine Fremdschlüssel (und auch keine Primärschlüssel).

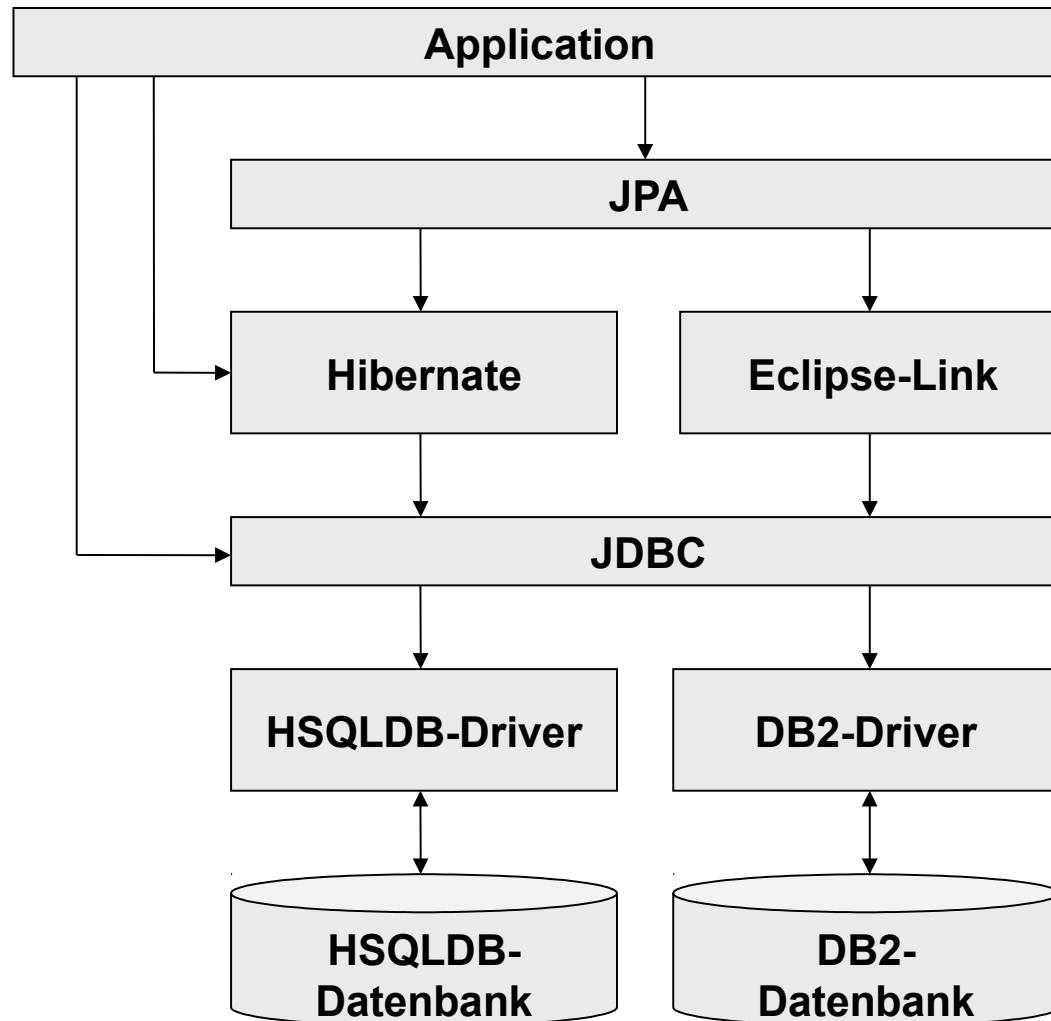
- Die Objektorientierung fokussiert individuelle Objekte; zwischen diesen Objekten kann navigiert werden. (Natürlich lassen sich solche individuellen Objekte auch in Collections zusammenfassen.)
 - Die typische Zugriffsweise von Datenbanken ist dagegen der `SELECT` in Verbindung mit dem `JOIN` – eine Zugriffsweise, die grundsätzlich Mengen von Zeilen liefert. Im Gegensatz zur Objektorientierung operiert die Datenbank also mengenorientiert.
- Die Objektorientierung kennt das Vererbungskonzept.
 - Relationale Datenbanken dagegen kennen mit wenigen Ausnahmen keine Vererbung.
- Relationale Datenbanken beruhen wesentlich auf dem Konzept der referenziellen Integrität;
 - in der Objektorientierung ist dieses Konzept von Natur aus unbekannt.

- Abbildung von Tabellenzeilen auf Objekte und umgekehrt
 - Ein O/R-Mapper muss eine Zeile einer relationalen Tabelle auf ein Objekt abbilden können – und zwar in beide Richtungen: er muss die Spaltenwerte einer Tabellenzeile lesen und dieses dann den Attributen des Objekts zuweisen können; und er muss umgekehrt die Attributwerte eines Objekts auslesen können und diese den Spalten einer Tabellezeile zuweisen können.
- Abbildung von Primär-/Fremdschlüssel-Beziehungen auf Referenzen
 - Die in der Datenbank enthaltenen Primär-/Fremdschlüsselbeziehungen müssen auf referenzielle Beziehungen der Objekte abgebildet werden.
- Generierung von SQL-Statements
 - Wenn dem OR-Mapping die Abbildung zwischen dem Objektmodell und dem Datenbank-Schema über XML-Dateien bekannt gemacht wird, dann ist es natürlich auch möglich, die für das INSERT, das UPDATE und das DELETE erforderlichen SQL-Statements automatisch zur Laufzeit zu generieren.

- Objektorientierte Abfragesprache
 - Für Abfragen sollte ein OR-Mapper eine eigene, objektorientierte Sprache anbieten.
- Identität von Objekten
 - Wird mittels eines OR-Mappers eine Tabellenzeile gelesen und in ein Objekt transformiert, so sollte dieses Objekt das einzige Objekt sein, welches die entsprechende Zeile im Hauptspeicher repräsentiert.
- Natives SQL
 - Für bestimmte Zwecke mag es sinnvoll oder gar notwendig sein, Abfragen oder DML-Befehle direkt in SQL zu formulieren. Der OR-Mapper sollte solche "workarounds" zulassen.

- Die Entity ist eine normale Java-Klasse, die spezielle Annotations besitzt
 - Welches Attribut wird in welcher Tabellenspalte abgelegt
 - Optimierung der Datenbankzugriffe mit Lazy Loading, Fetching...
 - Alternativ kann auch eine XML-basierte externen Konfigurationsdatei benutzt werden
- Das Speichern, suchen etc. übernimmt der `EntityManager`
 - Das Objekt selber ist nicht per se persistent, sondern ist in verschiedenen Zuständen vorhanden:
 - Transient (keine Entsprechung zu einem Datensatz)
 - Persistent (entspricht einem Datensatz, der Entity Manager muss die Objekt-Identität garantieren)
 - Detached (entspricht einem Datensatz, ein detached Objekt ist ein unabhängiger Snapshot des Datenbestandes)
 - **Die Zustandswechsel werden vom `EntityManager` gesteuert**



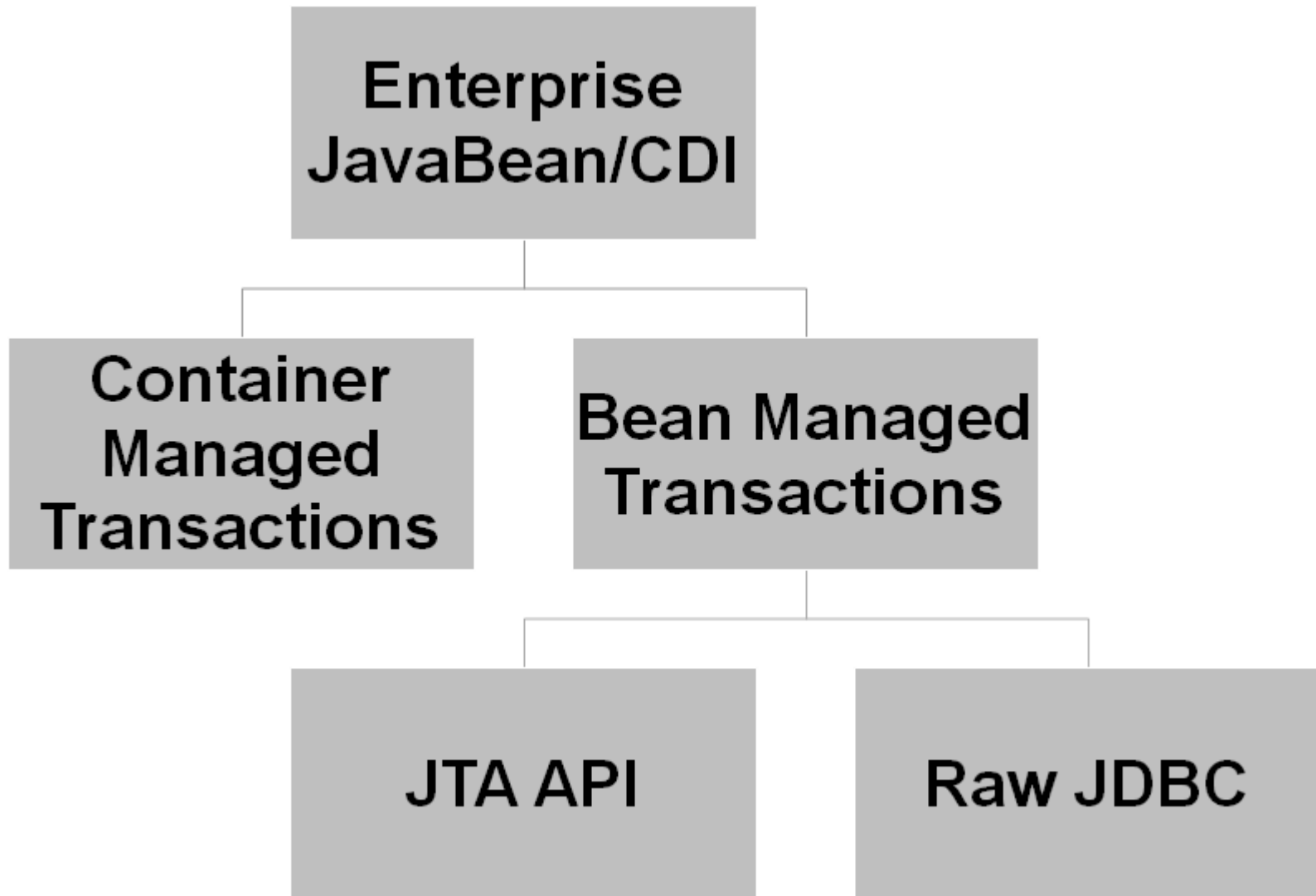


7.3

TRANSAKTIONSSTEUERUNG

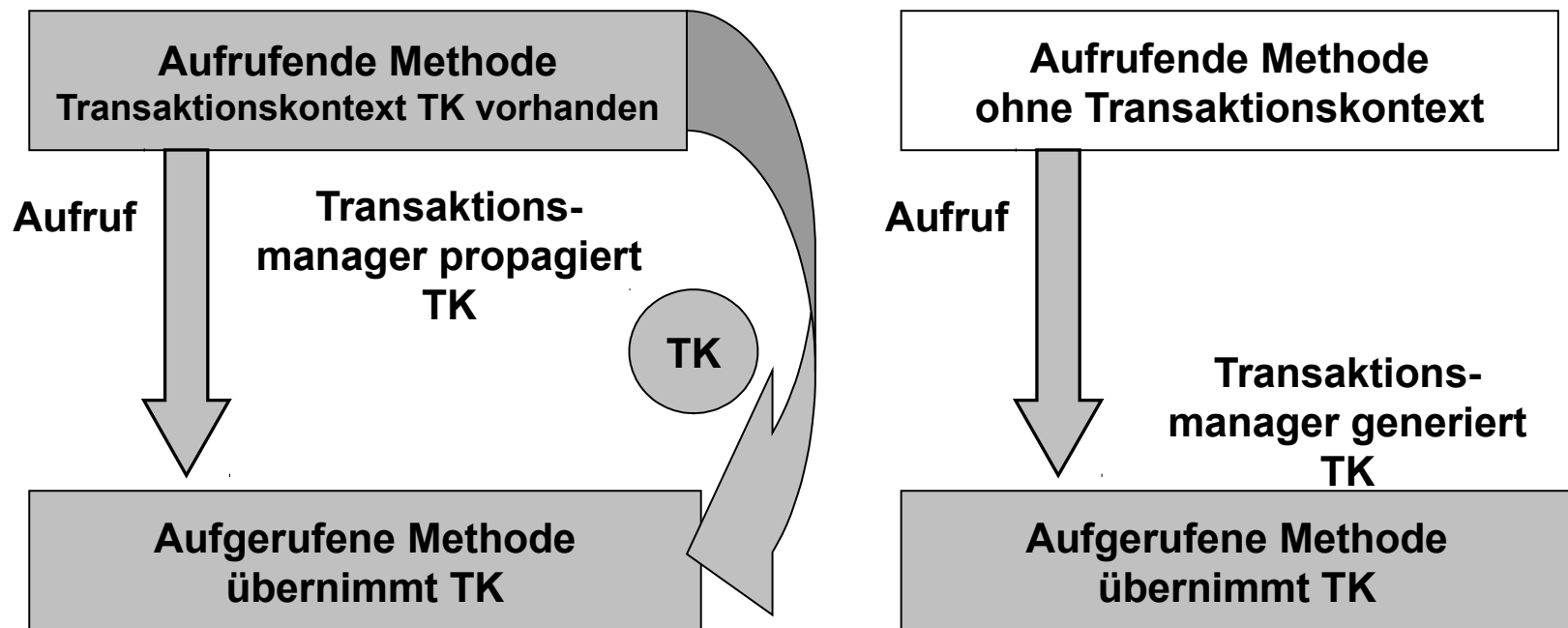
- Ab der JEE 7 unterstützen auch CDI Beans die deklarative Transaktionssteuerung
 - `javax.transaction.Transactional`
- Enterprise JavaBeans sind damit für die Realisierung transaktioneller Fachobjekte nicht mehr notwendig
- Die folgenden Erläuterungen sind für EJBs und CDI gültig

- Deklaratives Transaktions-Management
 - Container übernimmt die gesamte Transaktionsverwaltung
 - Definition eines Transaktions-Attributes auf Methodenebene
 - Die Propagierung der Transaktion wird vom Transaction Manager des Applikationsservers übernommen
- Die Transaktionssteuerung kann auch explizit vom Programmierer vorgenommen werden
 - Dazu wird der Bean vom Container eine `UserTransaction` übergeben
- Die Zuordnung der Transaktionsattribute zu einer Bean-Methode erfolgt durch Annotations
 - Auch eine externe Konfiguration über einen XML-Deskriptor ist möglich

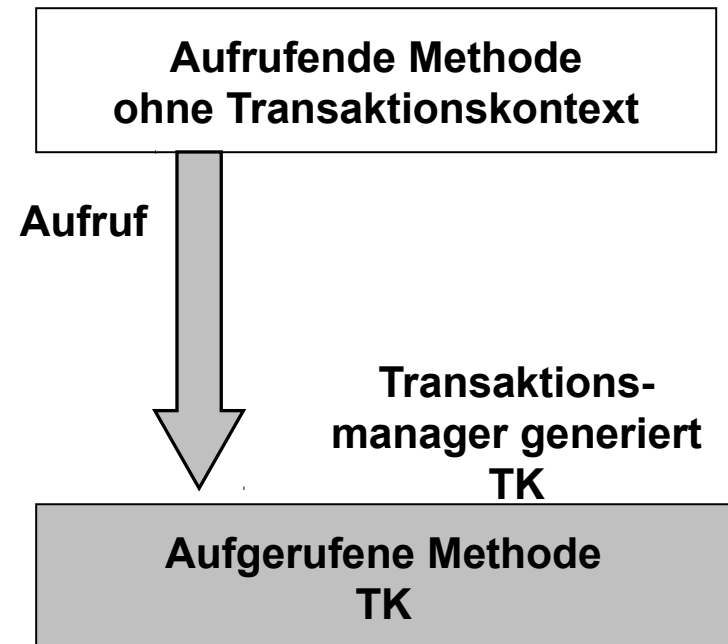
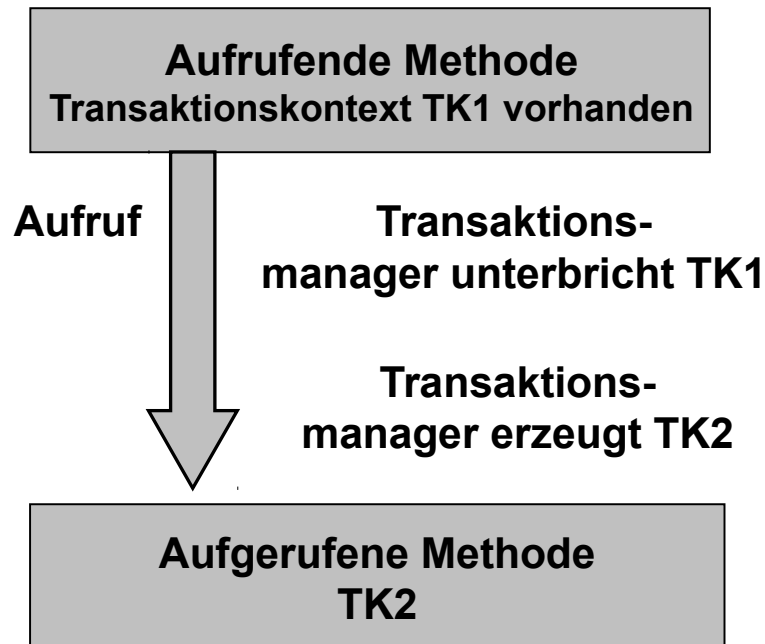


- Es existieren 6 Transaktions-Attribute
 - Attribut:
 - Not Supported
 - Supports
 - Required
 - Requires New
 - Mandatory
 - Never
- Transaktions-Attribute werden auf Methodenebene vergeben
 - Beim Aufruf einer Methode wird an Hand der Attribute eine neue Transaktion gestartet oder ein vorhandener Kontext übernommen

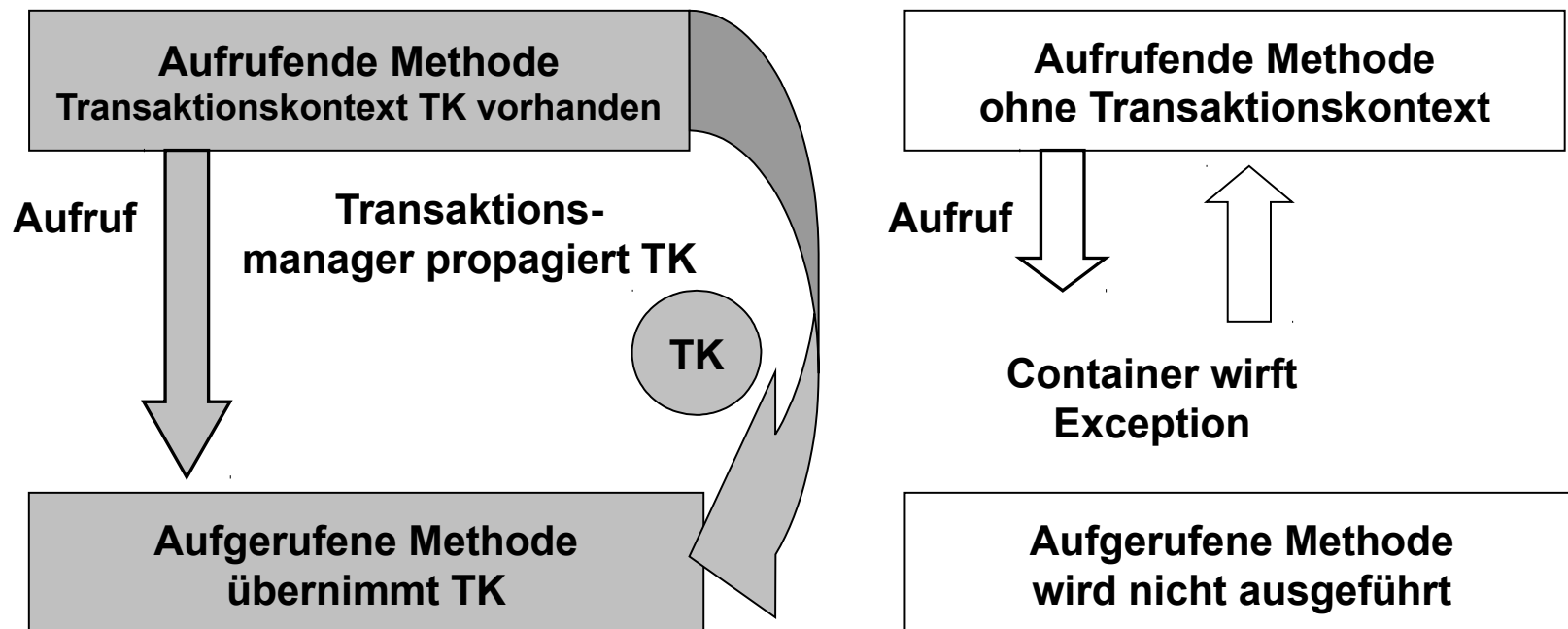
- Die aufgerufene Methode enthält garantiert einen Transaktionskontext



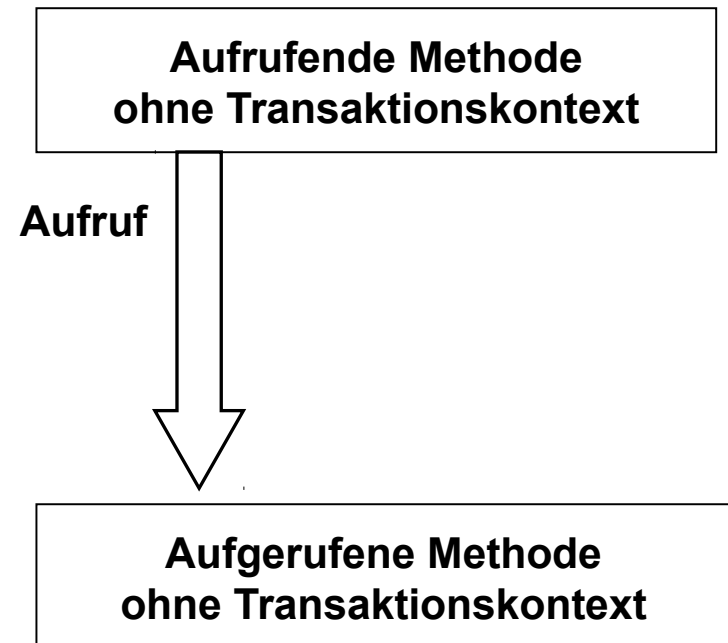
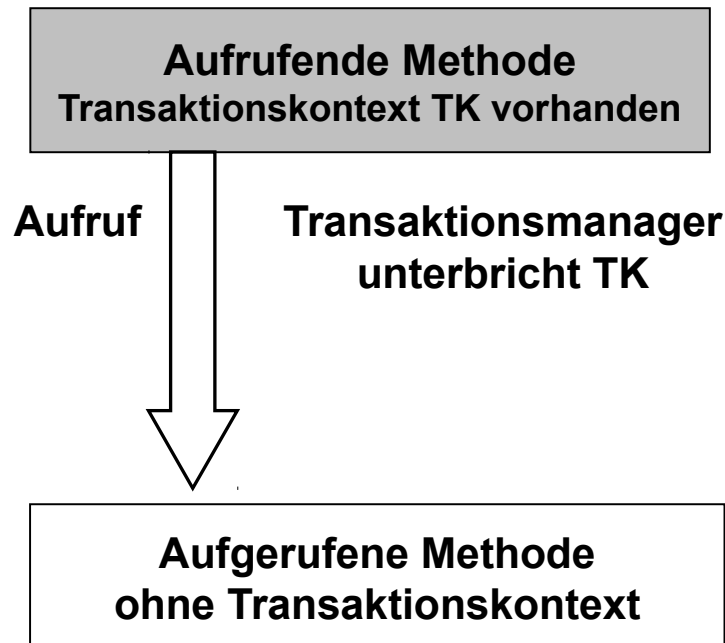
- Die aufgerufene Methode enthält garantiert einen Transaktionskontext



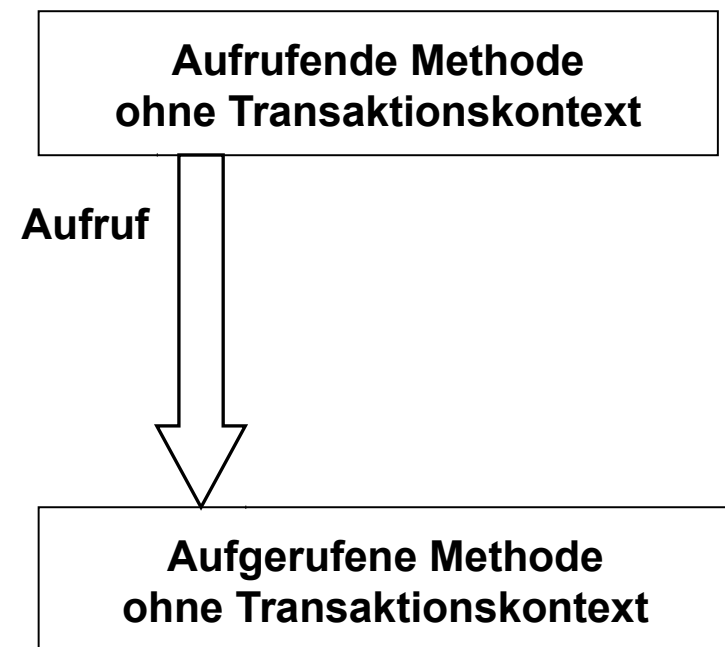
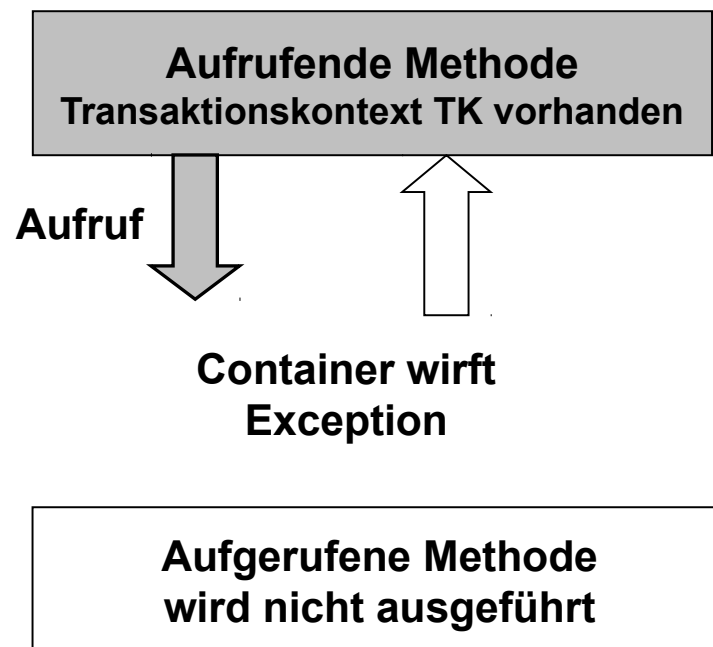
- Der Client muss einen Transaktionskontext besitzen
- Ansonsten: `TransactionRequiredException`



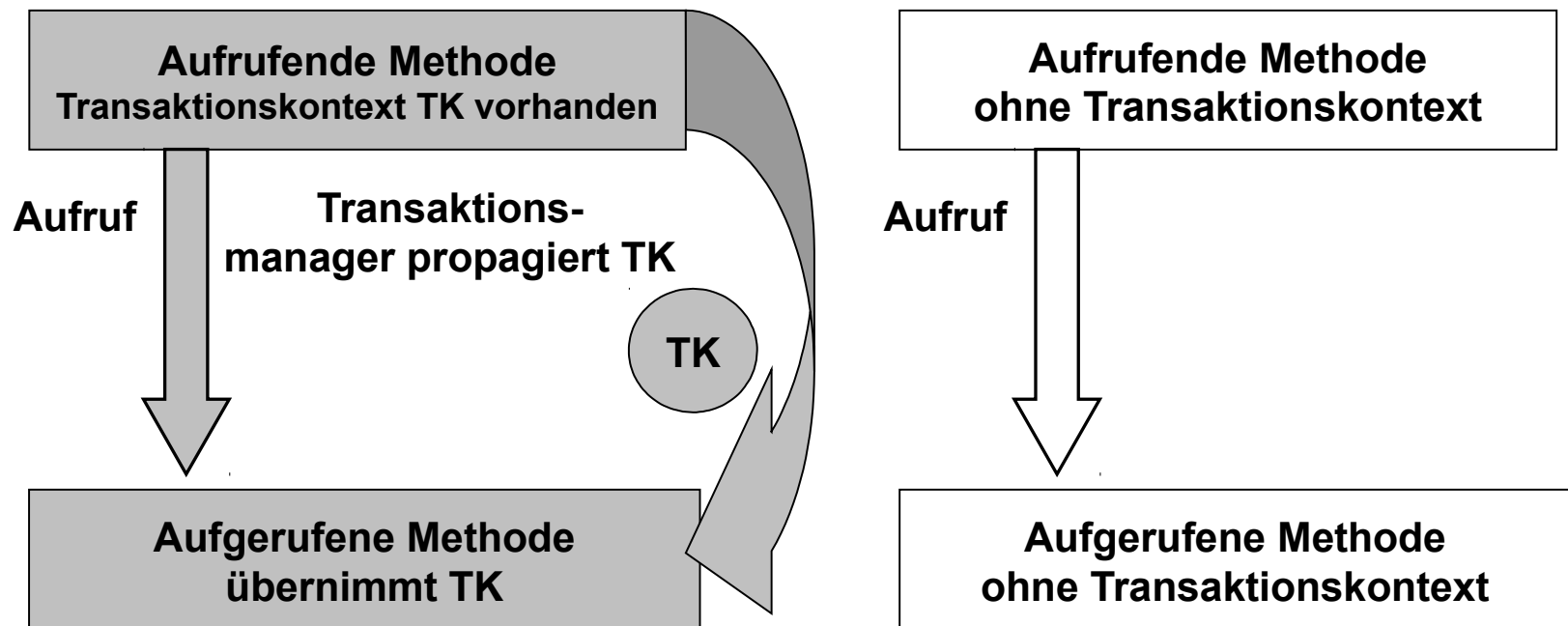
- Die aufgerufene Methode läuft ohne Transaktionskontext



- Der Client darf keinen Transaktionskontext besitzen
- Ansonsten: RemoteException



- Die aufgerufene Methode läuft mit dem Transaktionskontext der aufrufenden Methode



8

ARCHITEKTUR UND DESIGN

8.1

VERTEILTE ANWENDUNGEN

- Kommunikation zwischen zwei virtuellen Maschinen
 - Sehr hochwertig
 - Objekt-orientiert
 - Datenaustausch mit Serialisierten Java-Objekten
 - Sehr effizient und auf Java optimiert
 - Distributed Garbage Collection
- Definition des Server-APIs über ein Java-Interface
- Bereitstellung über eine Stateless oder Stateful SessionBean
 - Zusätzliche Annotation: `@Remote`

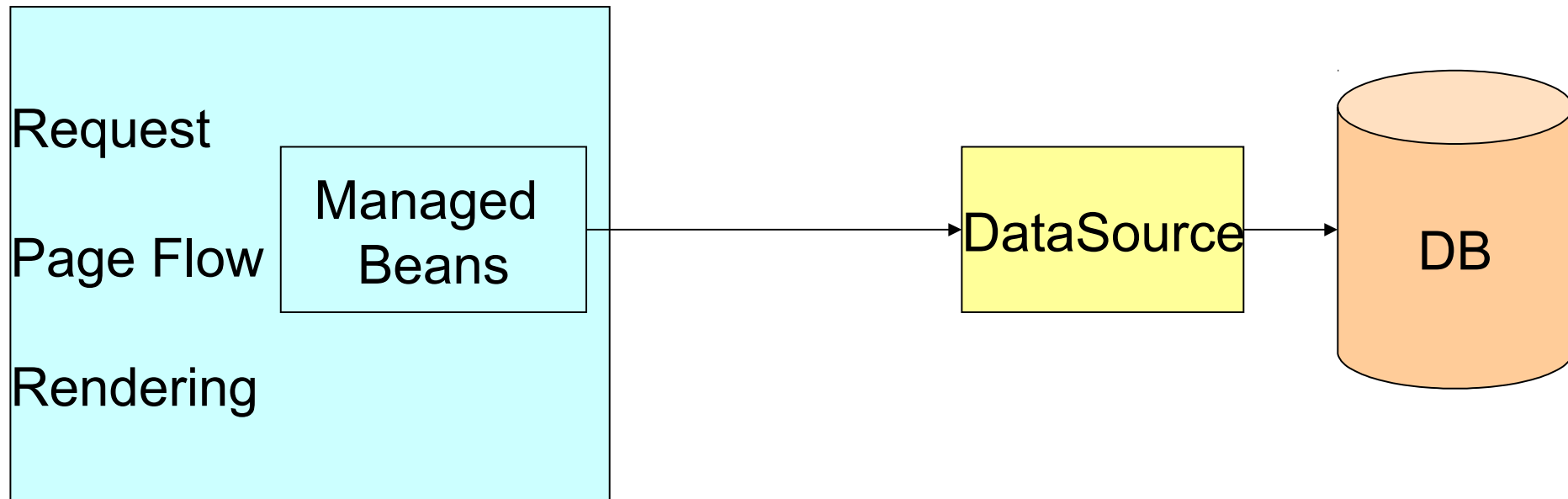
- Notwendig hierzu ist ein Messaging System
 - dazu kann der im Applikationsserver vorhandene genutzt werden
 - häufiger werden jedoch externe Messaging Systeme genutzt
 - z.B. Apache ActiveMQ
- Die Kommunikation erfolgt über das Versenden von Nachrichten an Destinations
 - Sender und Empfänger vereinbaren hierzu eine Nachrichten-Format
 - Die Validierung der Nachrichten bleibt größtenteils Aufgabe der Anwendung
- Bereitstellung über einen MessageListener oder eine MessageDrivenBean
 - Zusätzliche Annotation: `@MessageDriven`

- Kommunikation heterogener Plattformen durch Austausch von XML-Dokumenten
 - Dies sind die SOAP-Envelopes
 - Dies sind XML-Dokumente
- Die Beschreibung des Services erfolgt über eine Schnittstelle formuliert in der Web Services Description Language (WSDL)
 - Contract First
- Alternativ hierzu kann die WSDL auch aus einer Java-Klasse erzeugt werden
 - Code First
 - Hierzu werden eine Vielzahl von Annotationen benutzt
 - JAX-WS, z.B. `@WebService`
 - JAXB, z.B. `@XmlElement`
- Über eine Code-Generierung werden Server-Rümpfe und Client-Stubs erzeugt
- Die Bereitstellung erfolgt über eine annotierte Klasse
 - CDI oder EJB

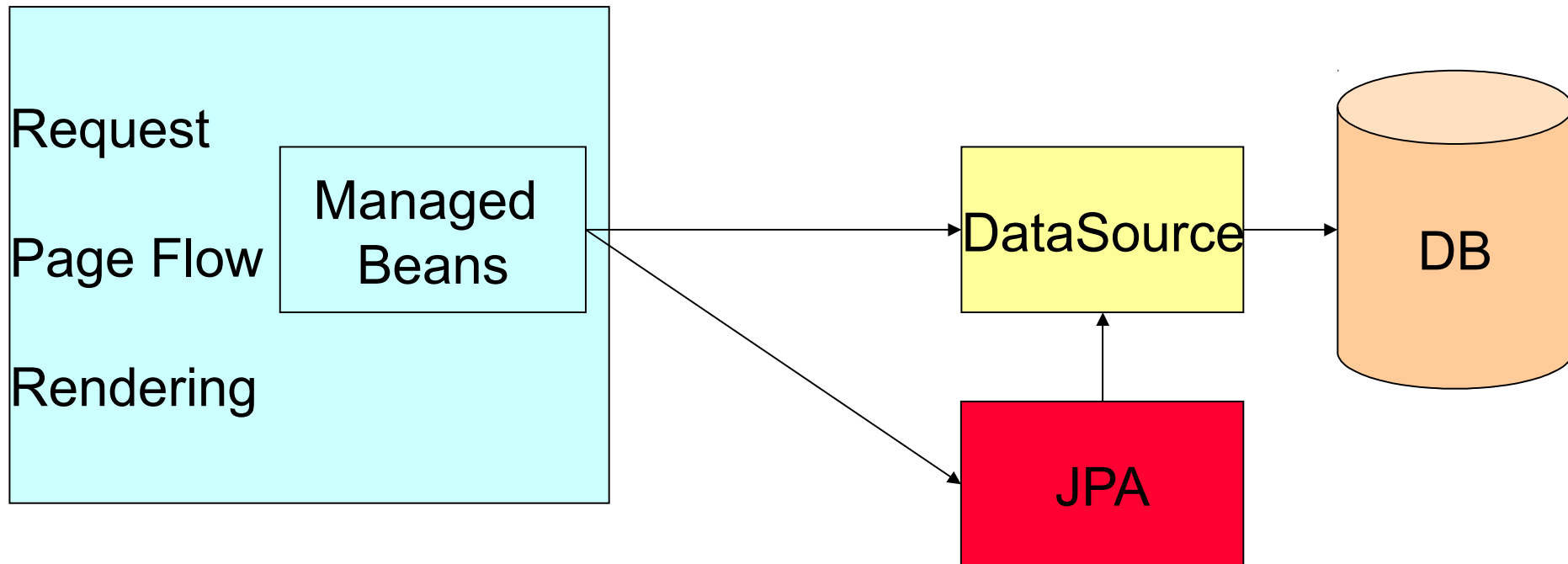
- Interoperable Kommunikation heterogener Plattformen durch Austausch von Standard-Dokumenten
 - Definiert über MIME-Types
- Der Server stellt für den Service eine Reihe von URL-Pfaden zur Verfügung
 - Als Operationen werden die http-Methoden benutzt
 - GET
 - PUT
 - POST
 - ...
- Die Bereitstellung erfolgt durch eine annotierte Klasse
 - CDI
 - EJB
 - Annotationen aus JAX-RS, z.B. `@Path`, `@Produces`, `@Consumes`

8.2

WEB-ARCHITEKTUREN MIT JAVASERVER FACES

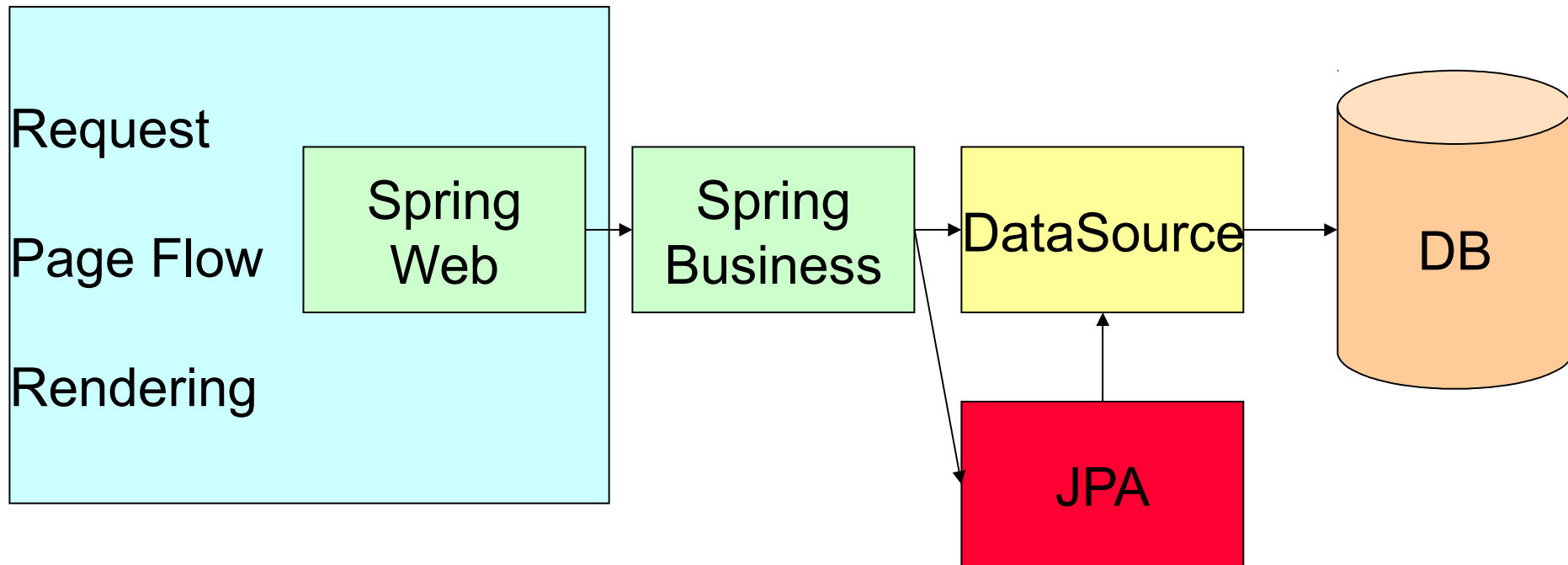


- Die Managed Beans
 - Halten das Datenmodell der Web Anwendung
 - Realisieren die Business-Logik
 - Insbesondere die Datenzugriffe und die Transaktionssteuerung

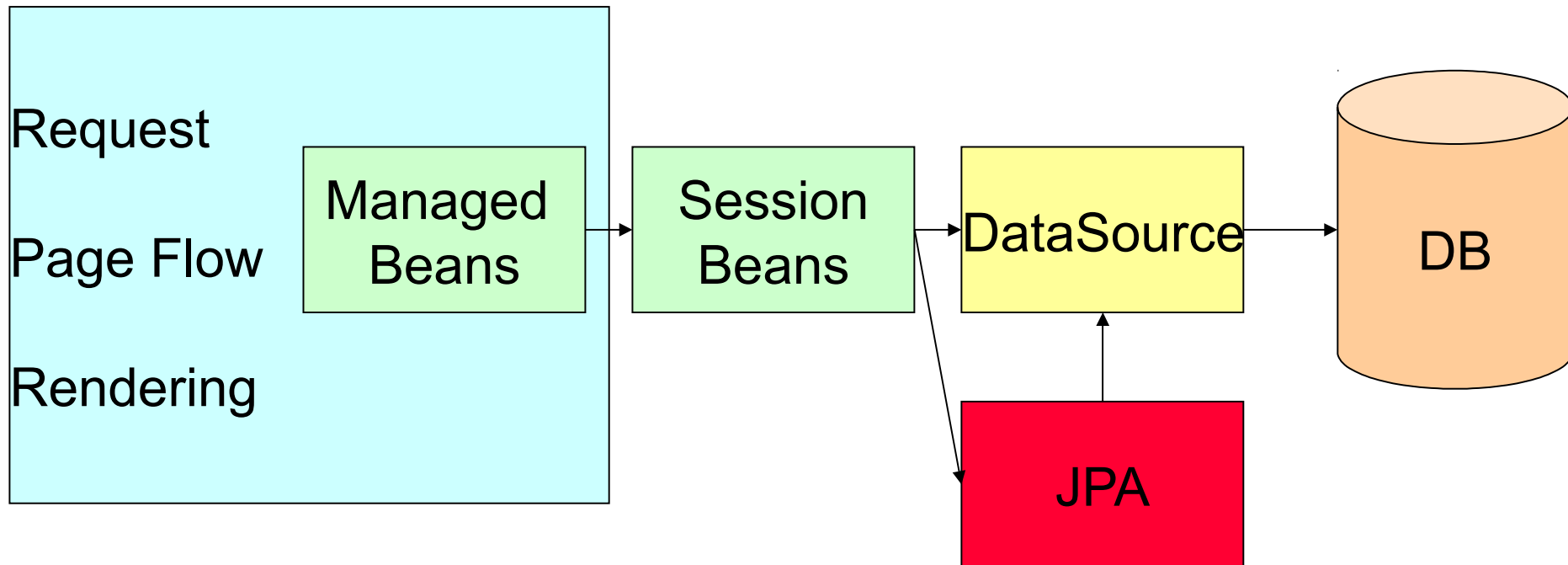


■ Die Managed Beans

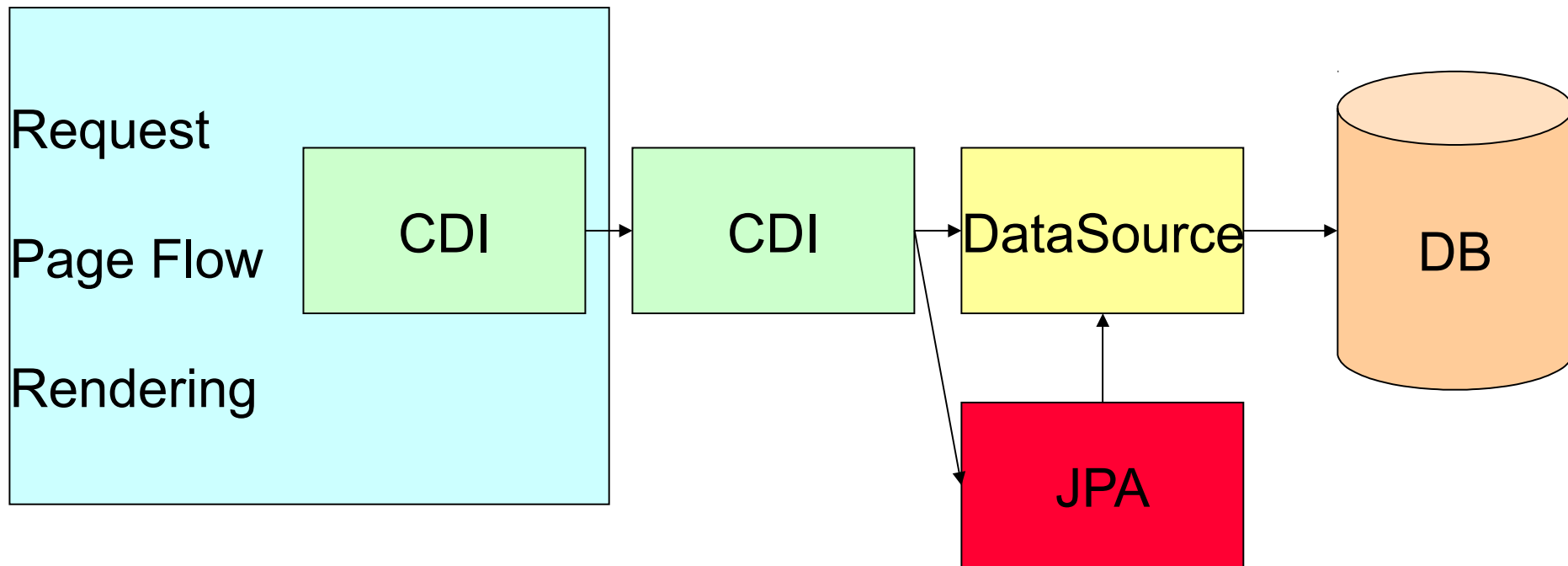
- Halten das Datenmodell der Web Anwendung
- Realisieren die Business-Logik



- Die Managed Beans werden durch Spring-Komponenten ausgetauscht
- Das Datenmodell der Web Anwendung sowie die Business-Logik wird getrennt



- Die Managed Beans halten das Web Datenmodell
- Die SessionBeans realisieren oder delegieren an die Business Logik und übernehmen die Transaktionssteuerung



- Das Web Datenmodell wird von beispielsweise `@RequestScoped` CDI-Beans gehalten
- Die `@ApplicationScoped`-CDI-Beans realisieren die Business Logik und übernehmen die Transaktionssteuerung

8.3

REMOTE FASSADEN MIT ENTERPRISE JAVABEANS

Applikationsserver

```
<<class>>
ApplicationBean

@Stateless
@Local
@Remote
@TransactionalAttribute
@Resource
@PersistenceContext
@WebService
@SoapBinding
@WebMethod
@WebParam
@WebResult
public void business() {
    //...
}
```

Applikationsserver

```
<<class>>  
ApplicationBean
```

```
@Stateless  
@Local  
@Remote  
@TransactionAttribute  
@WebService  
@SoapBinding  
@WebMethod  
@WebParam  
@WebResult  
@Inject Controller
```

```
<<class>>  
Application
```

```
@Named  
@Resource  
@PersistenceContext  
@Inject AnotherApplication  
public void business() {  
    //...  
}
```

Applikationsserver

```
<<class>>  
RemoteApplicationBean  
  
@Stateless  
@Remote  
@Inject ApplicationBean
```

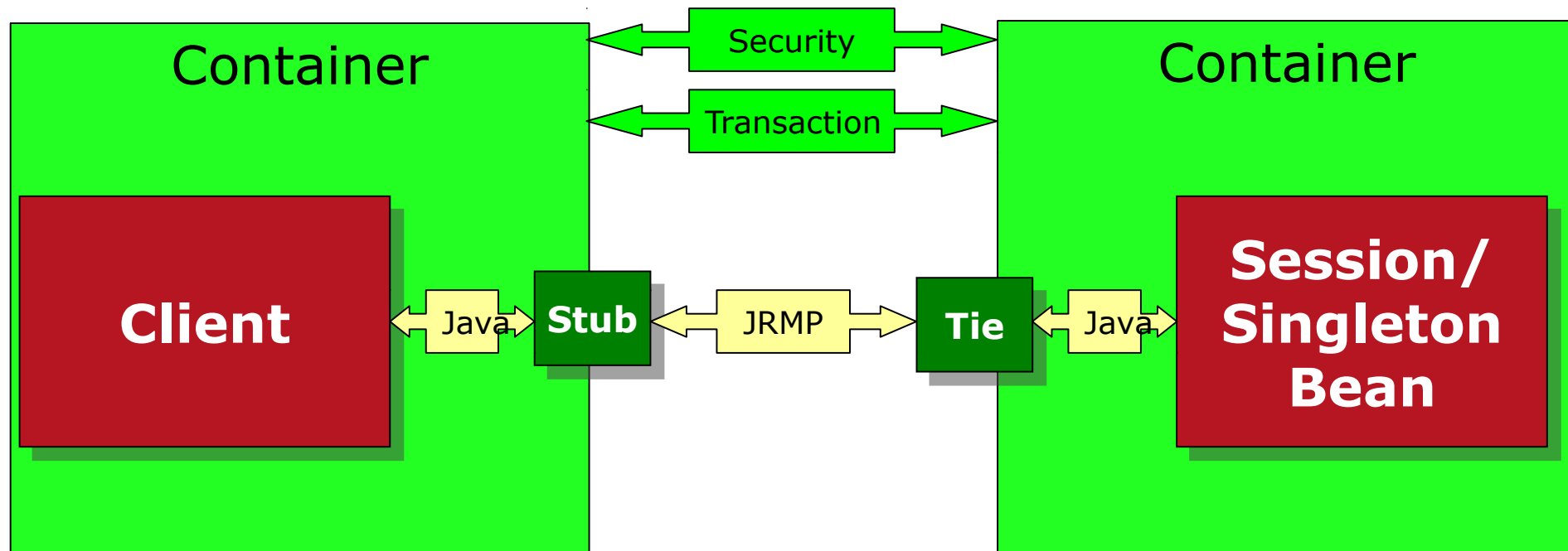
```
<<class>>  
ApplicationWebService  
  
@Stateless  
@WebService  
@SoapBinding  
@WebMethod  
@WebParam  
@WebResult  
@Inject ApplicationBean
```

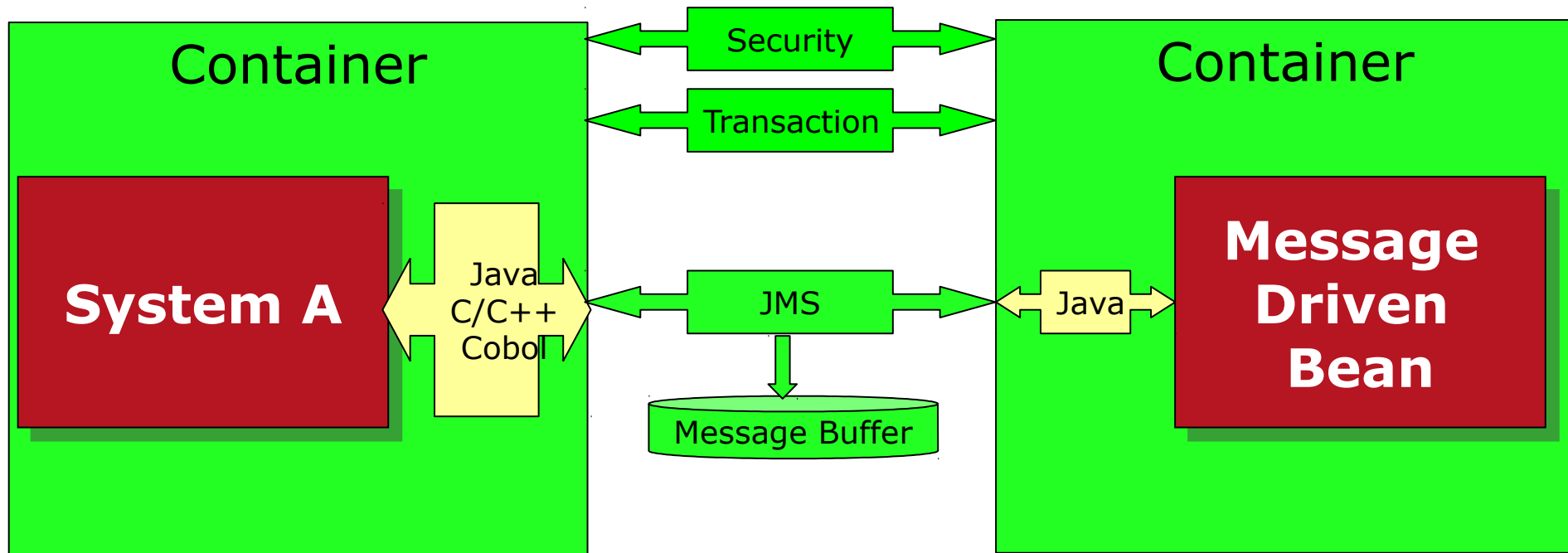
```
<<class>>  
ApplicationBean  
  
@Stateless  
@Local  
@TransactionAttribute  
@Inject Application
```

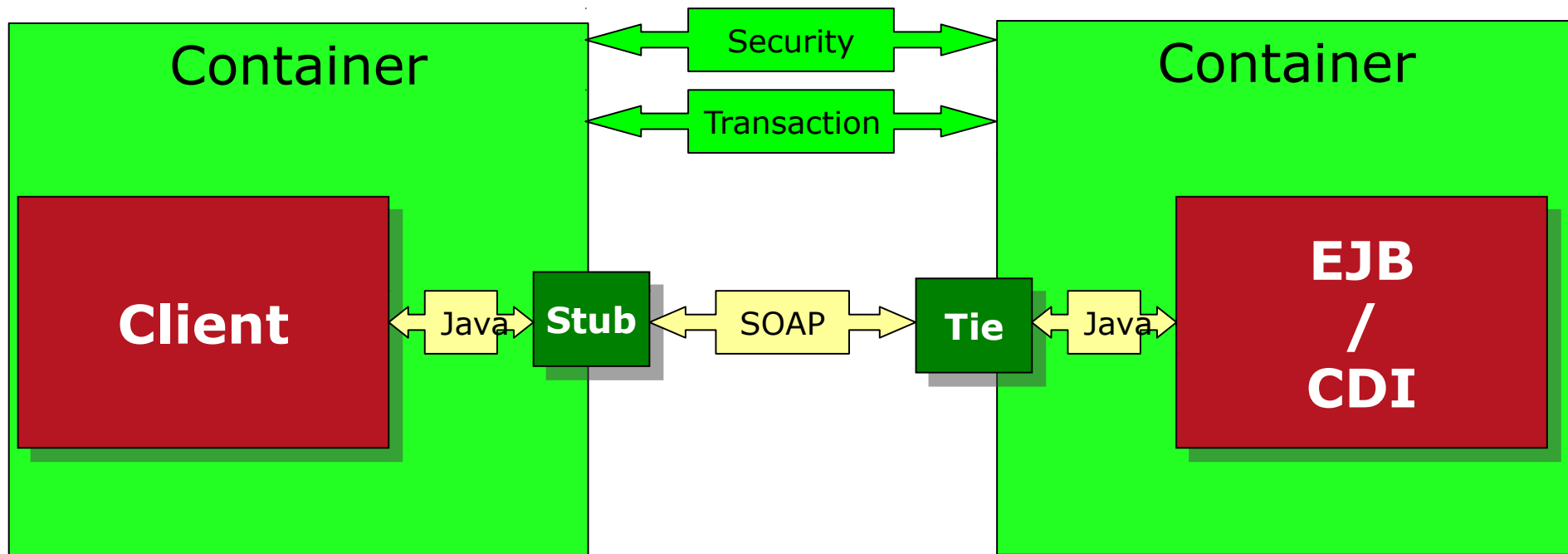
```
<<class>>  
Application  
  
@Named  
@Resource  
@PersistenceContext  
@Inject  
    AnotherApplication  
public void business(){  
    //...  
}
```

8.4

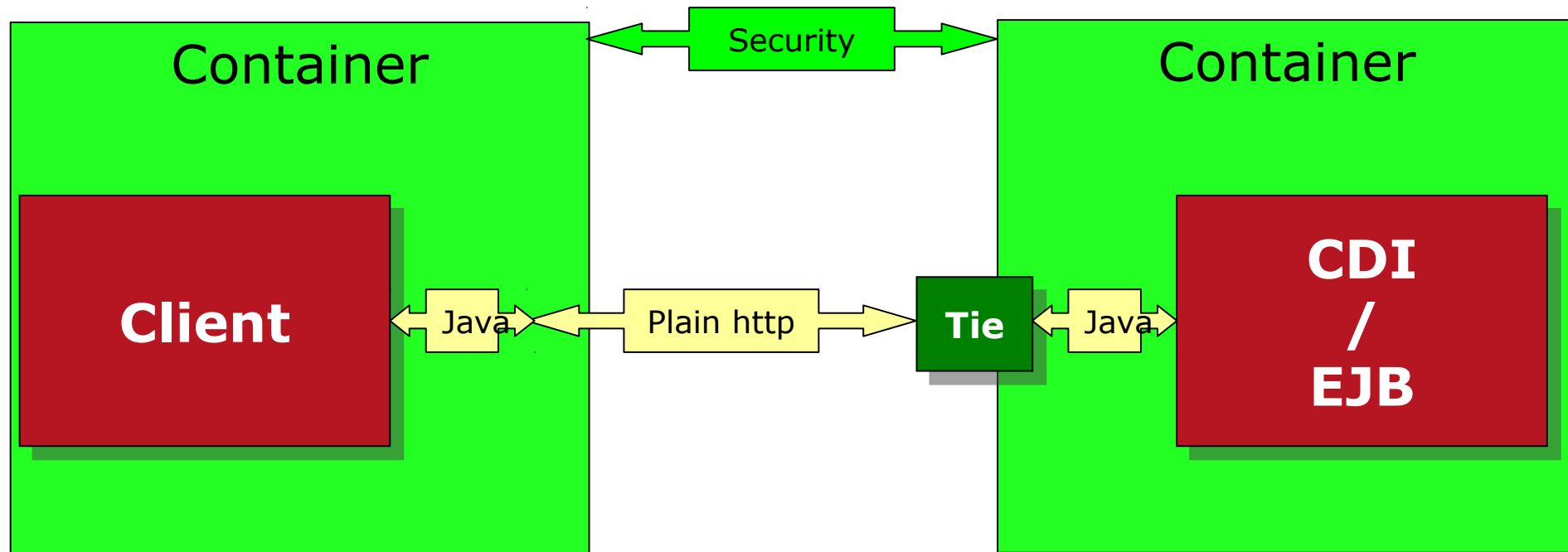
CLIENT SERVER







Security und die Propagierung einer Transaktion sind nicht verpflichtend unterstützt!

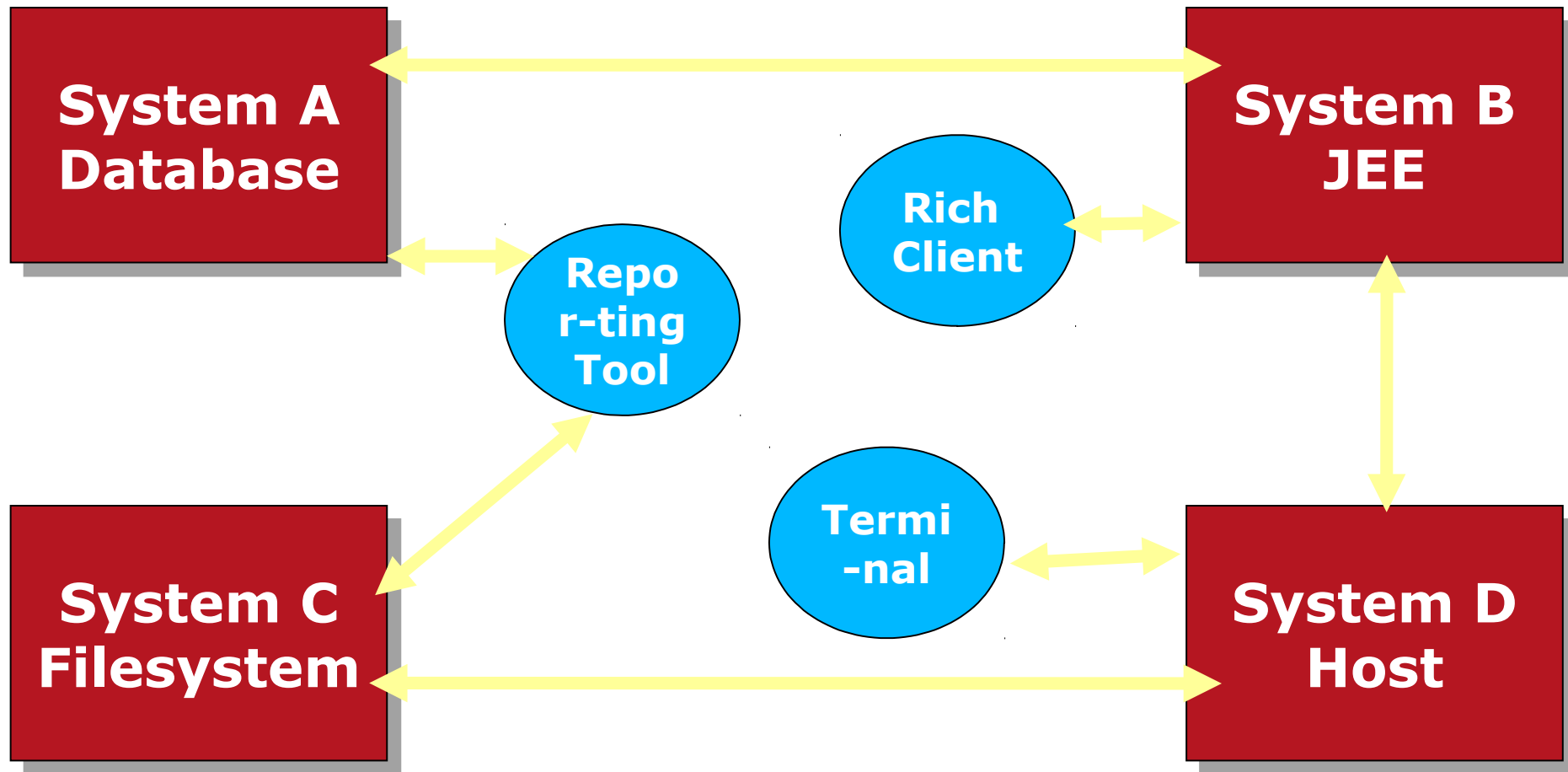


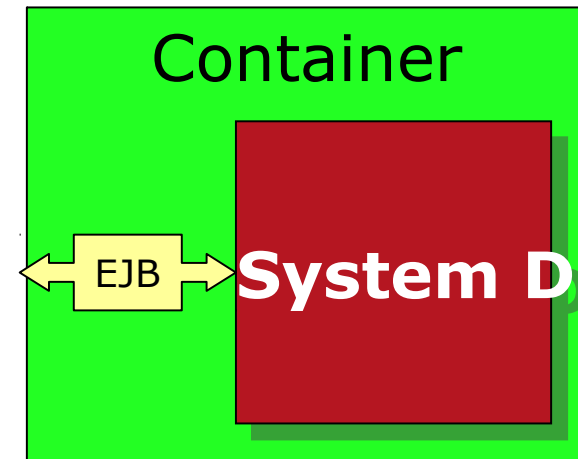
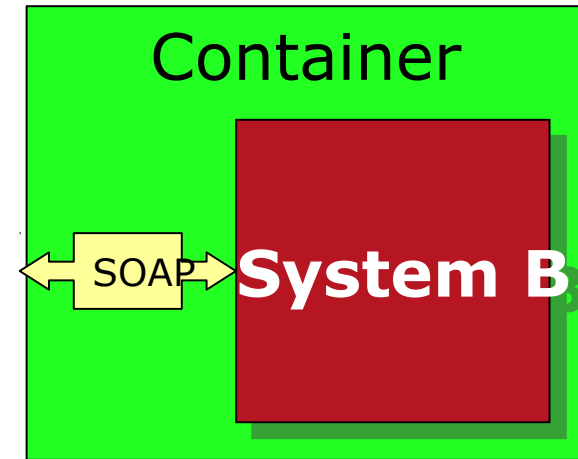
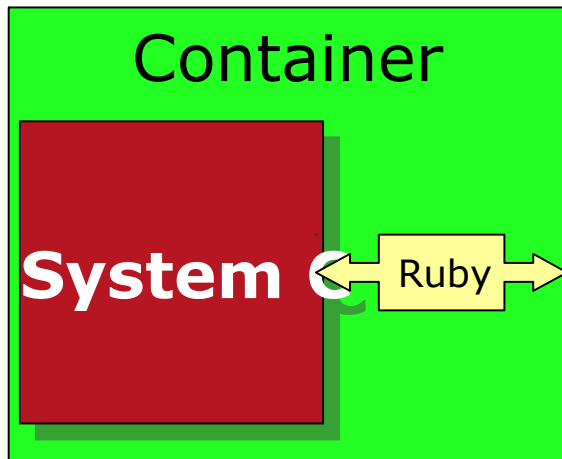
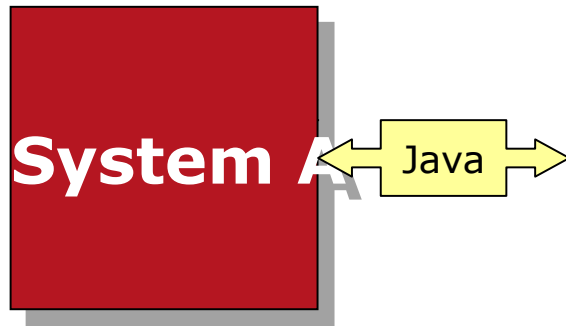
Security und die Propagierung einer Transaktion sind nicht verpflichtend unterstützt!

8.5

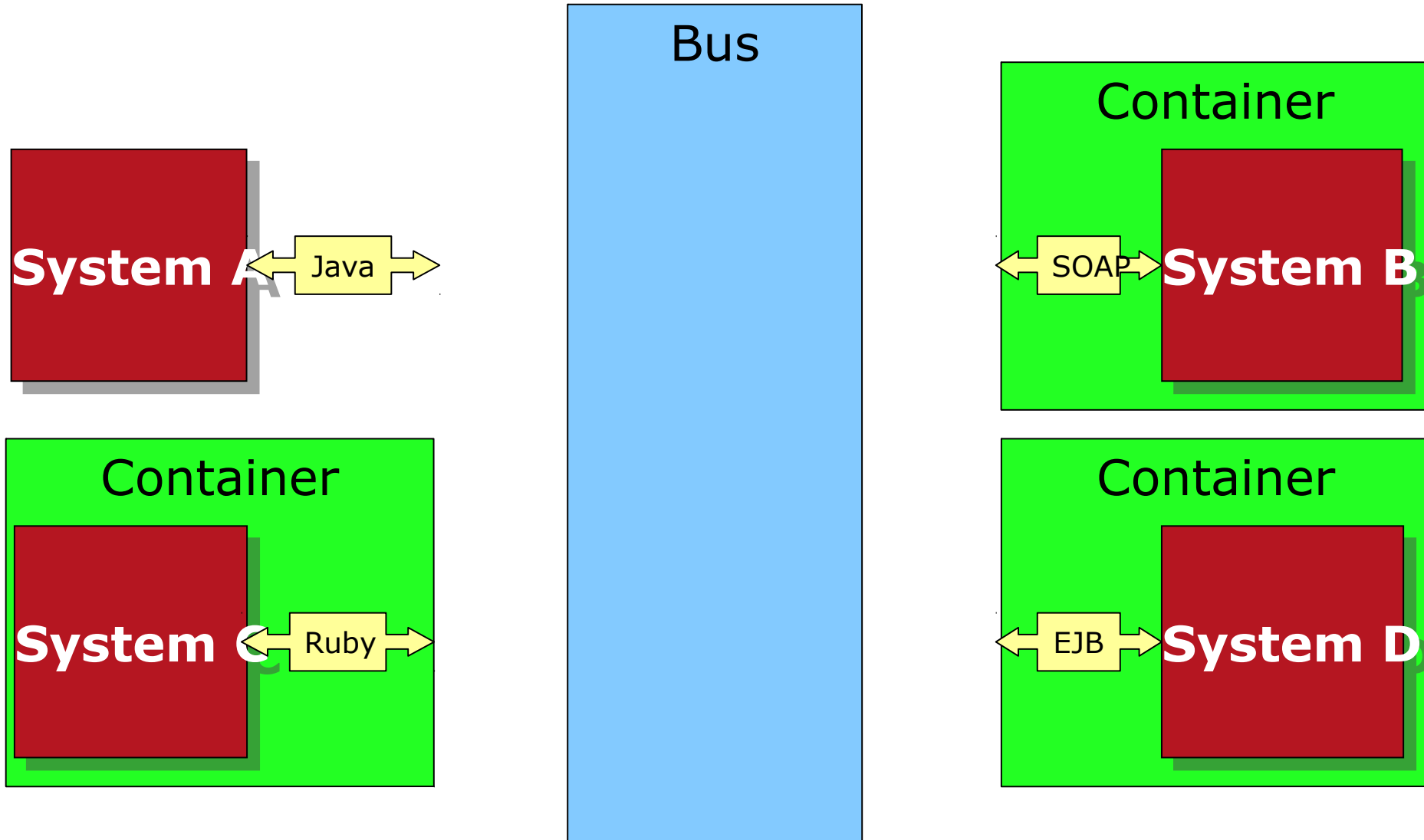
SERVICE ORIENTED ARCHITECTURE

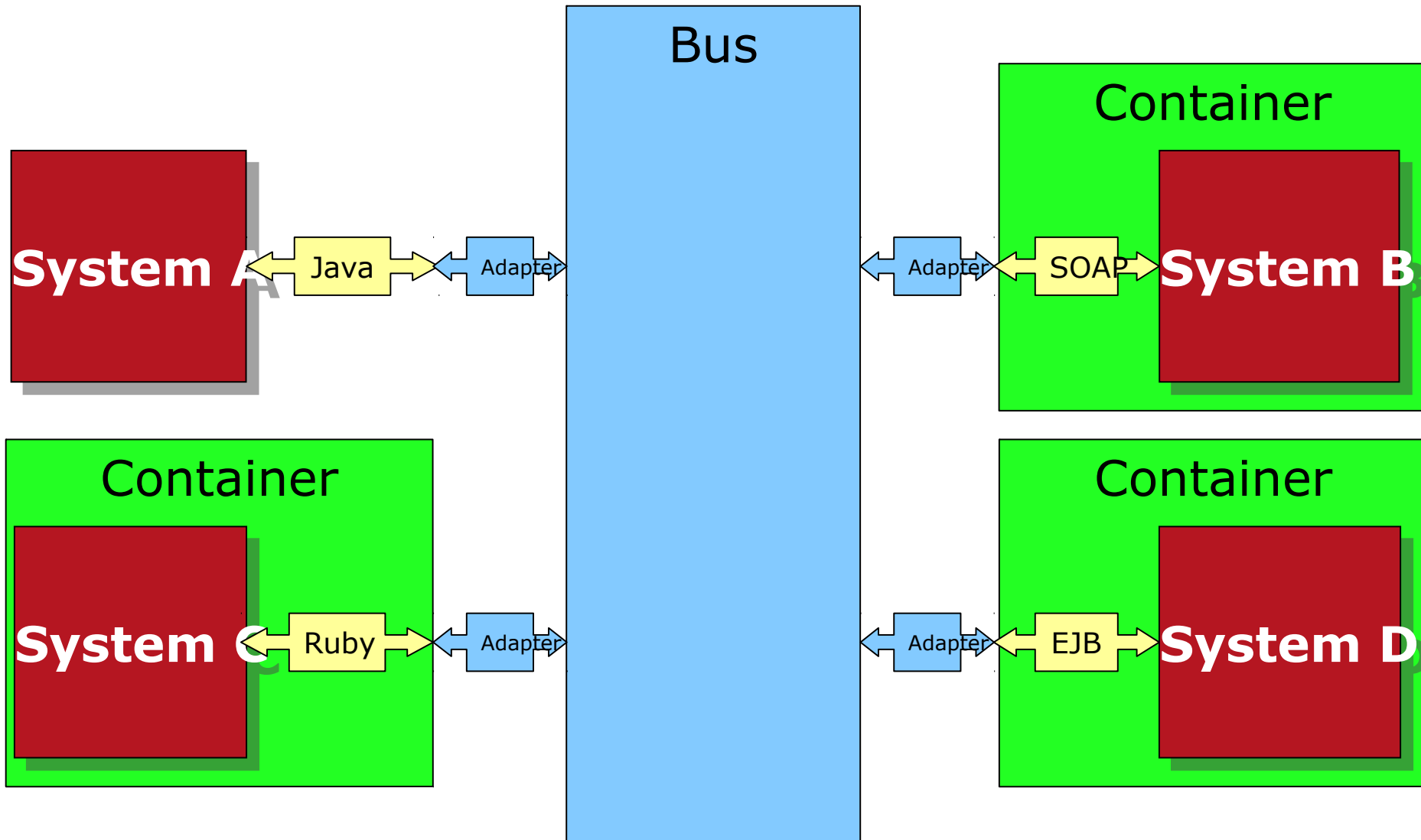
- Service Oriented Architecture ist ein Begriff für eine Best Practice
 - Keine Spezifikation!
 - Keine Plattform!
 - Kein Komponentenmodell!
 - Kein Programmiermodell!
- Häufig auch noch verwechselt mit Web Services
 - Diese sind jedoch nur eine Möglichkeit von vielen, Services zu definieren und aufzurufen

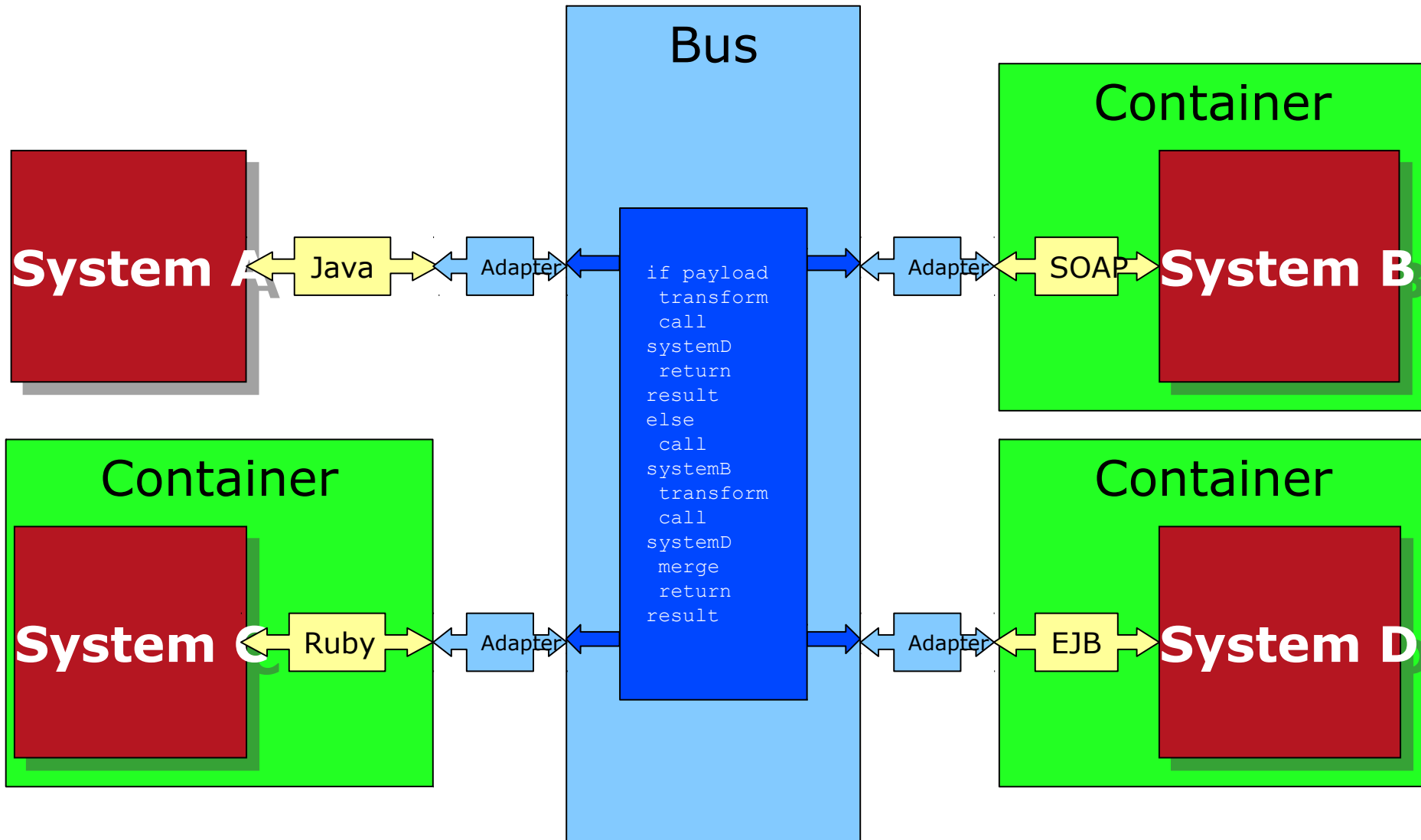




Einführung eines Bus-Systems







- Einführung von Prozess-Variablen.
 - Diese werden automatisch zwischen den einzelnen Prozessschritten persistiert.
- Workflows werden durch einen Graphen repräsentiert.
 - Damit ist Java als Programmiersprache wenig geeignet.
 - Besser: XML-basierte Skript-Sprachen
 - BPEL
 - jBPM
- Die Unterstützung von Workflow-Sprachen ist im Applikationsserver nicht verpflichtend
 - JEE verlangt „nur“ Java
 - Erweiterungen oder Produkt-Lösungen stellen diese Funktionalität jedoch zur Verfügung
 - Enterprise Service Bus (ESB)-Lösungen auf Basis der JEE

