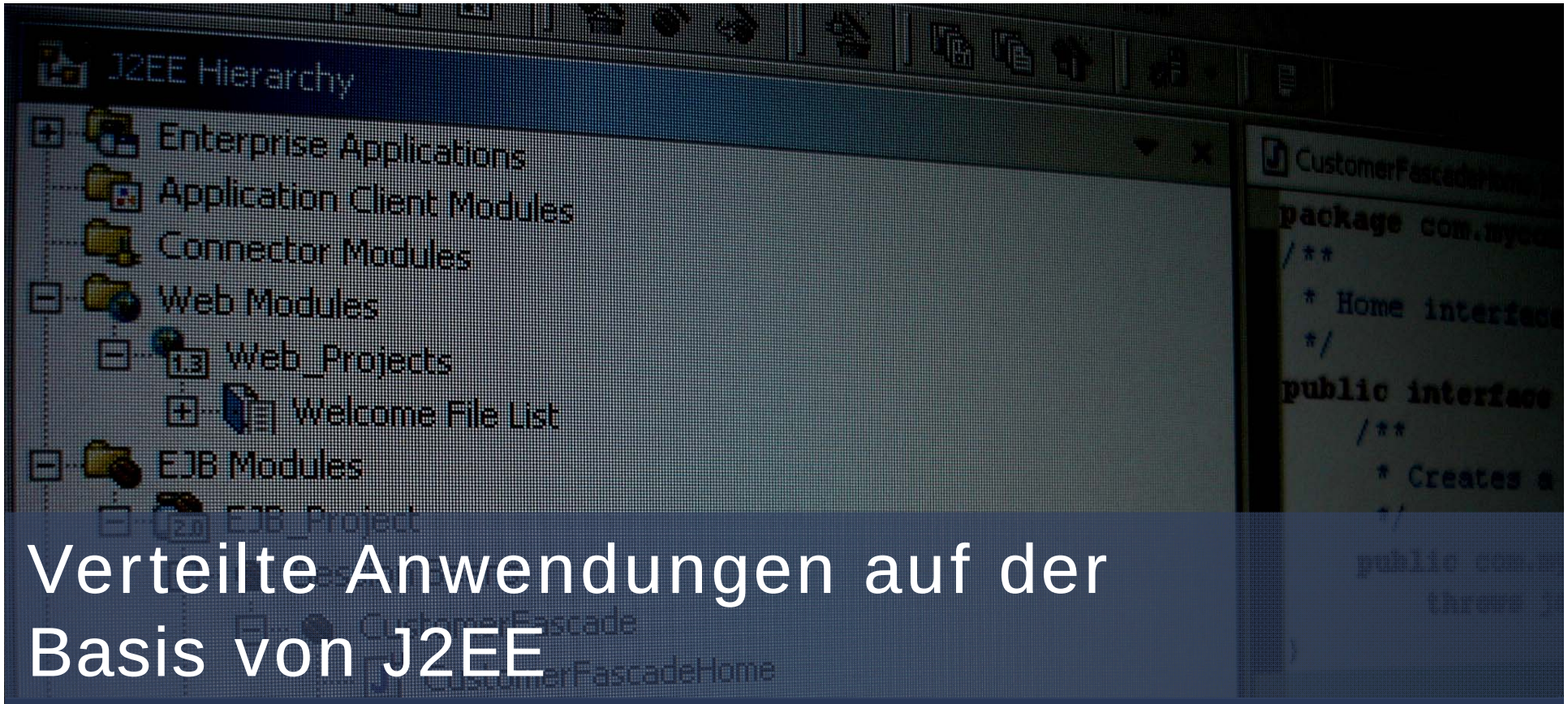


Verteilte Anwendungen auf der Basis von J2EE

Stefan Malich

Wintersemester 2004/2005

Version 1.0

Lehrstuhl für Wirtschaftsinformatik und
Softwaretechnik
Prof. Dr. Stefan Eicker

Agenda

Einführung und Motivation

Verteilte Objekte und Komponenten

Verteilte Softwarearchitekturen

J2EE-Plattform

J2EE-basierte Softwarearchitektur

Aspekte der Verteilung von J2EE-Anwendungen

Agenda

- Einführung und Motivation
- Verteilte Objekte und Komponenten
- Verteilte Softwarearchitekturen
- J2EE-Plattform
- J2EE-basierte Softwarearchitektur
- Aspekte der Verteilung von J2EE-Anwendungen

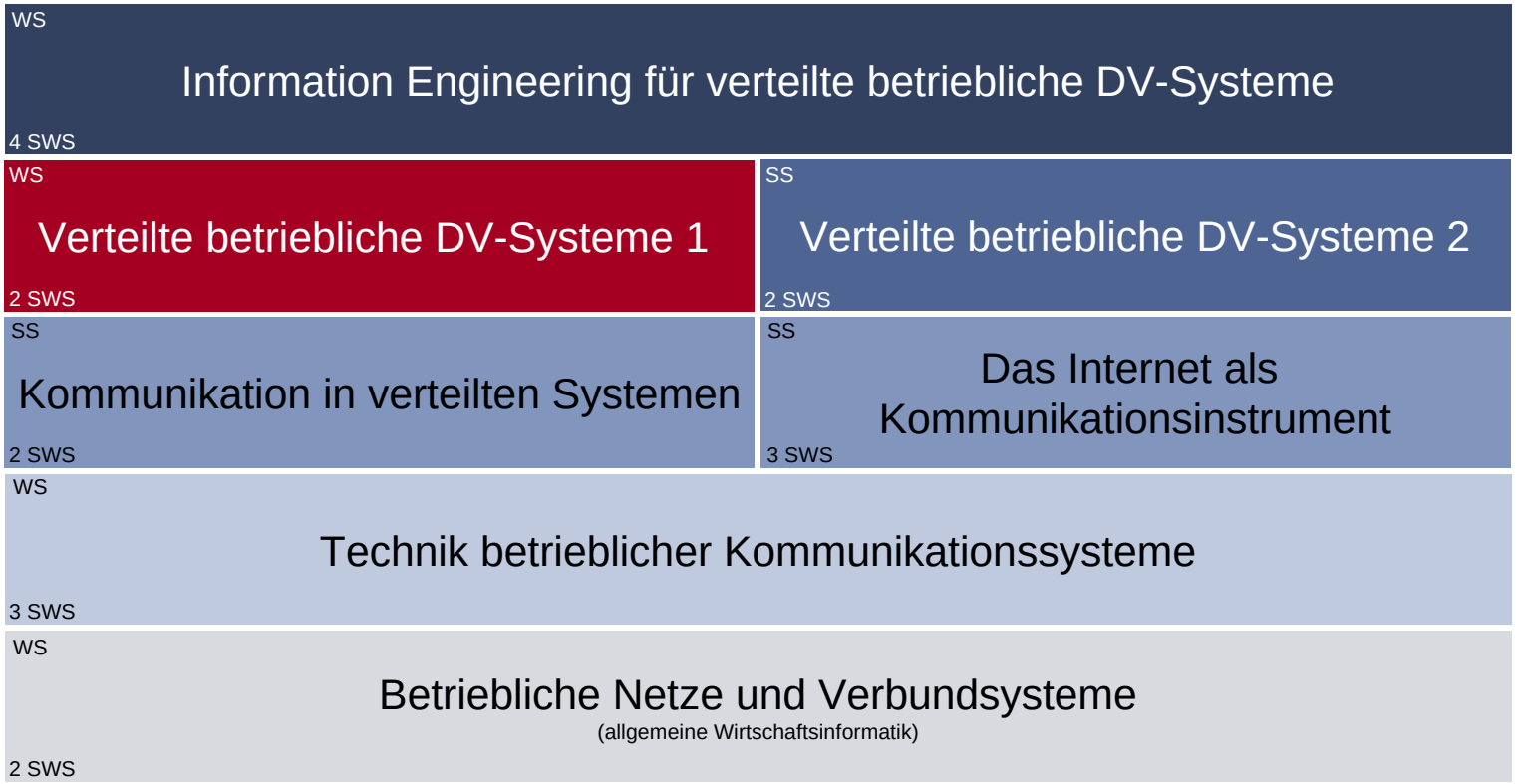
Organisatorisches

- Folien
  - Bereitstellung erfolgt sukzessive vor jeder Vorlesungseinheit
- Kommunikation
  - Veranstaltungsbegleitendes Forum
  - E-Mail: stefan.malich@icb.uni-essen.de
  - Innerhalb der Sprechstunde des Lehrstuhls: montags, 14:00 Uhr
- Literatur und Quellen
  - Quellenangaben in den Folien
  - Zahlreiche Dokumente, Spezifikationen und Bücher sind frei verfügbar und somit im engeren Fokus
- Englische Begriffe und Bezeichnungen
  - keine konsequente Übersetzung

Literaturempfehlungen

- Armstrong, Eric et al. - The J2EE Tutorial v1.4
  - Sehr gute und umfassende Anleitung zur Entwicklung von Anwendungen auf der Basis von J2EE
  - http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
- Sun Microsystems - Java 2 Platform, Enterprise Edition (J2EE) Specification, v1.4
  - Die führende Spezifikation zur J2EE-Plattform
  - http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf
- Sun Microsystems - Enterprise JavaBeans Specification, Version 2.1
  - Die führende Spezifikation zur Enterprise JavaBeans
  - http://java.sun.com/products/ejb/docs.html
- Aufgrund des Umfangs wird ein gezieltes Studium einzelner Kapitel empfohlen!

Positionierung der Vorlesung



Einführung und Motivation

Data Warehousing und verteilte Anwendungen

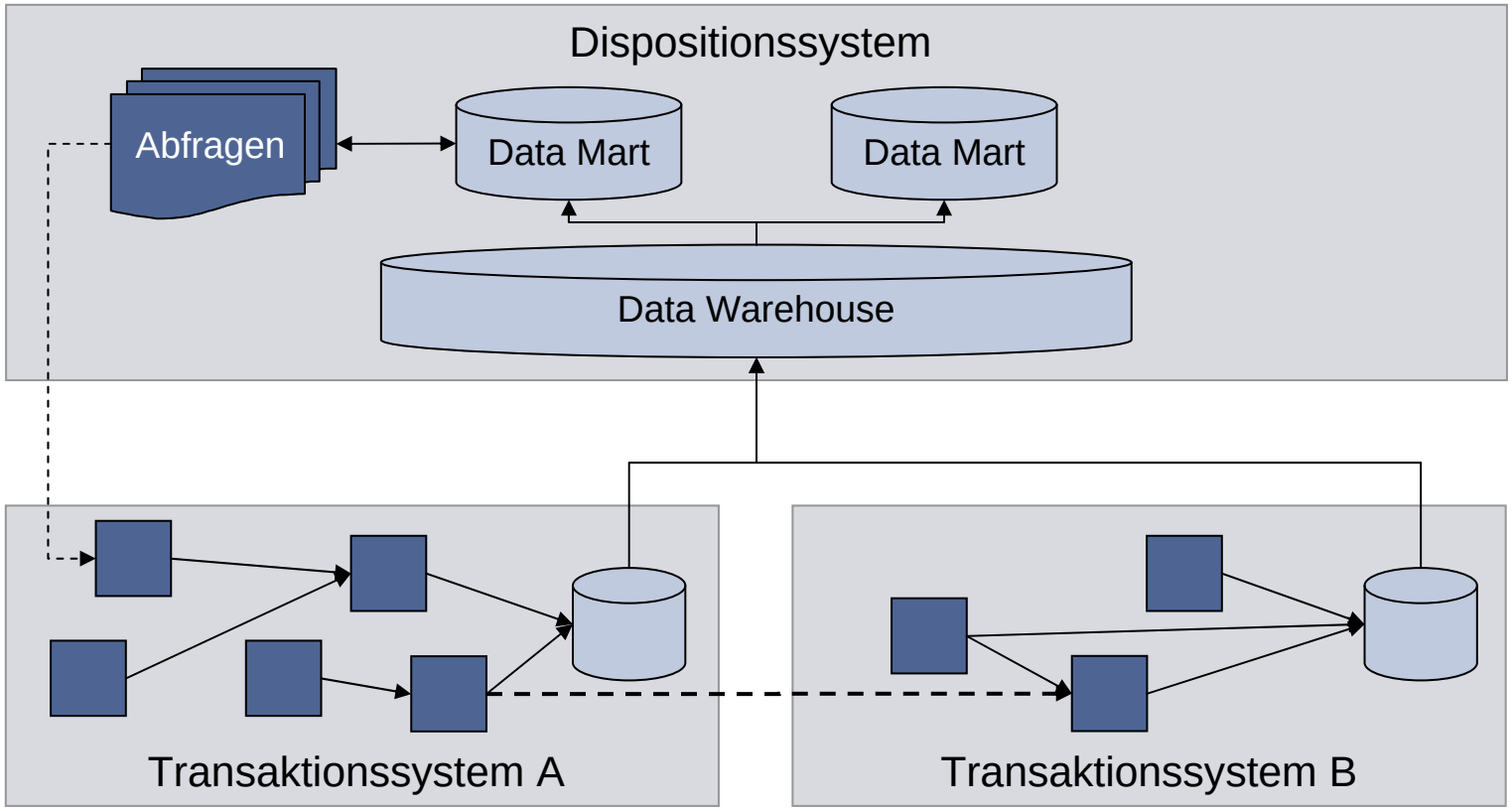
Teil 1:
Data Warehousing

- Dispositionssysteme
  - datenorientiert
- Primärer Fokus
  - Datenmodell
  - Abfragen
  - Datenbeschaffung, Daten-transformation und Datenanalyse

Teil 2:
Verteilte Anwendungen auf der Basis von J2EE

- Transaktionssysteme
  - transaktionsbasiert
- Primärer Fokus
  - Softwarearchitektur
  - Softwarekomponenten
  - Transaktionen

Praxisrelevanz der Themen



Begriffsdefinitionen

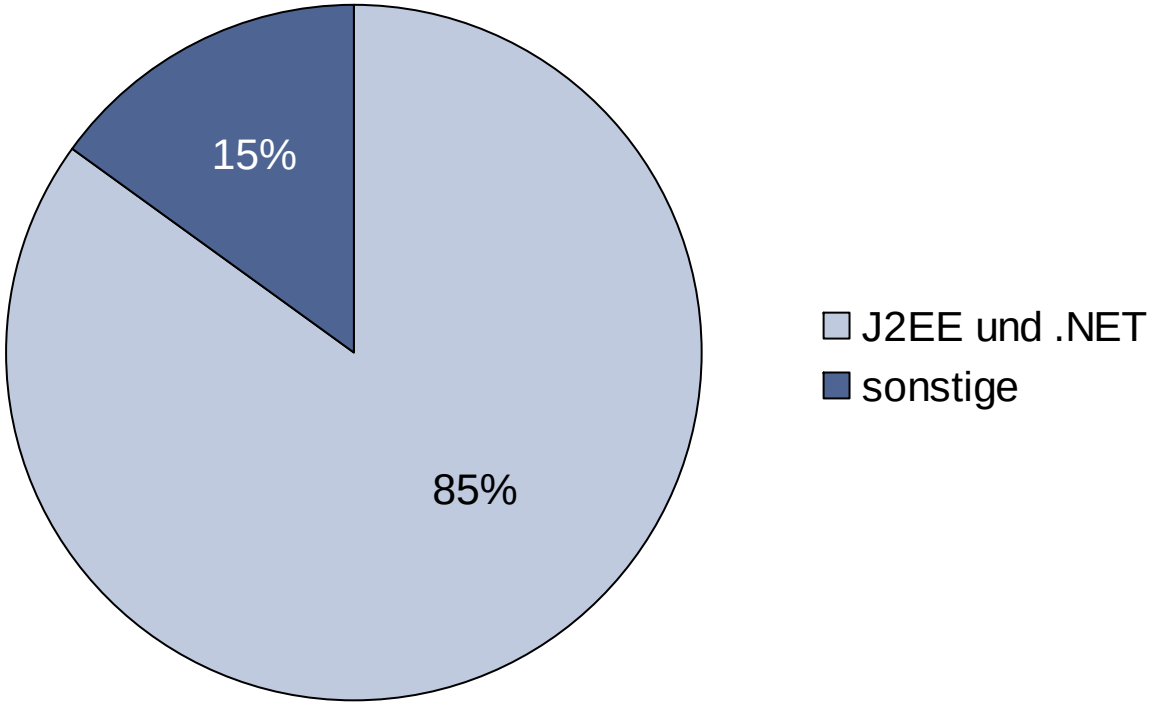
- Plattform
  - Wikipedia: "In computing, a platform describes some sort of framework, either in hardware or software, which allows software to run. Typical platforms include a computer's architecture, operating system, or programming languages and their runtime libraries." (http://en.wikipedia.org/wiki/Platform_(computing))
  - Entwicklung und Ausführung von Software
- Entwicklungsplattform
  - Engerer Fokus auf die Entwicklung von Software
  - Beispiele: J2EE, .NET, COBOL, usw.

Relevanz der Vorlesungsinhalte

Die Plattformen J2EE und .NET gehören zu den Grundkenntnissen eines (Wirtschafts-)Informatikers, da diese Plattformen bei einem Großteil der Entwicklungen von transaktionsbasierten Anwendungen eingesetzt werden.

Anteile bei Neuentwicklungen

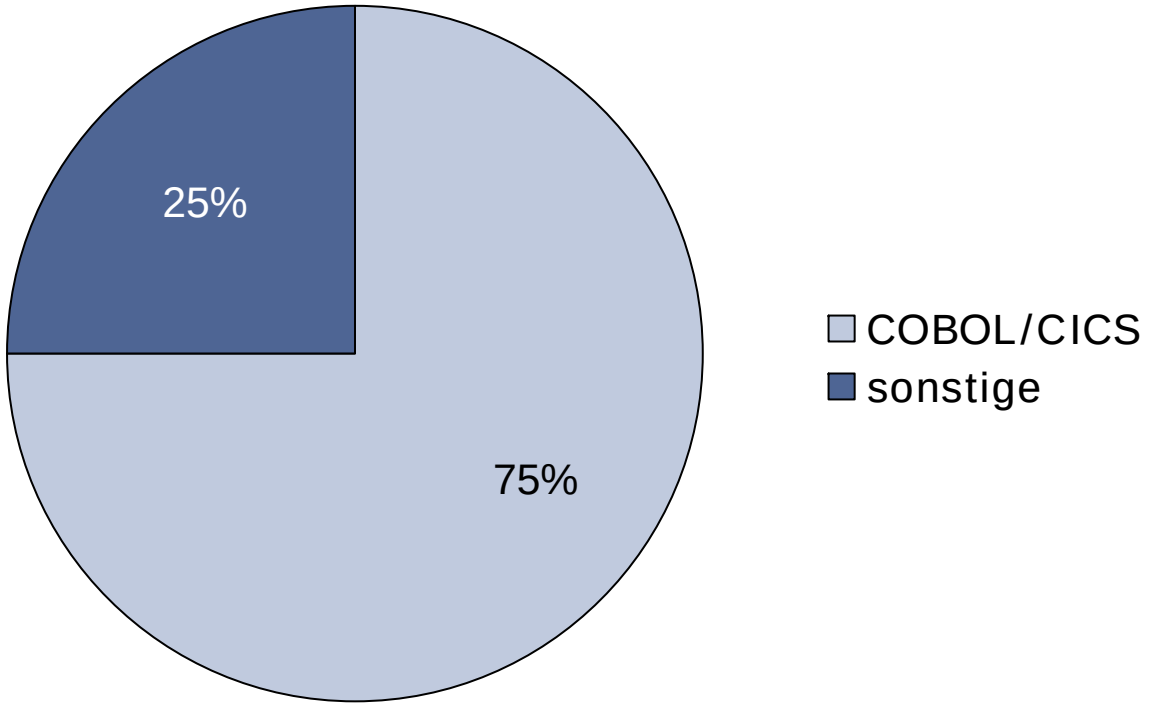
Relative Anteile der Entwicklungsplattformen bei transaktionsbasierten Neuentwicklungen



Quelle:Gartner/namics ag

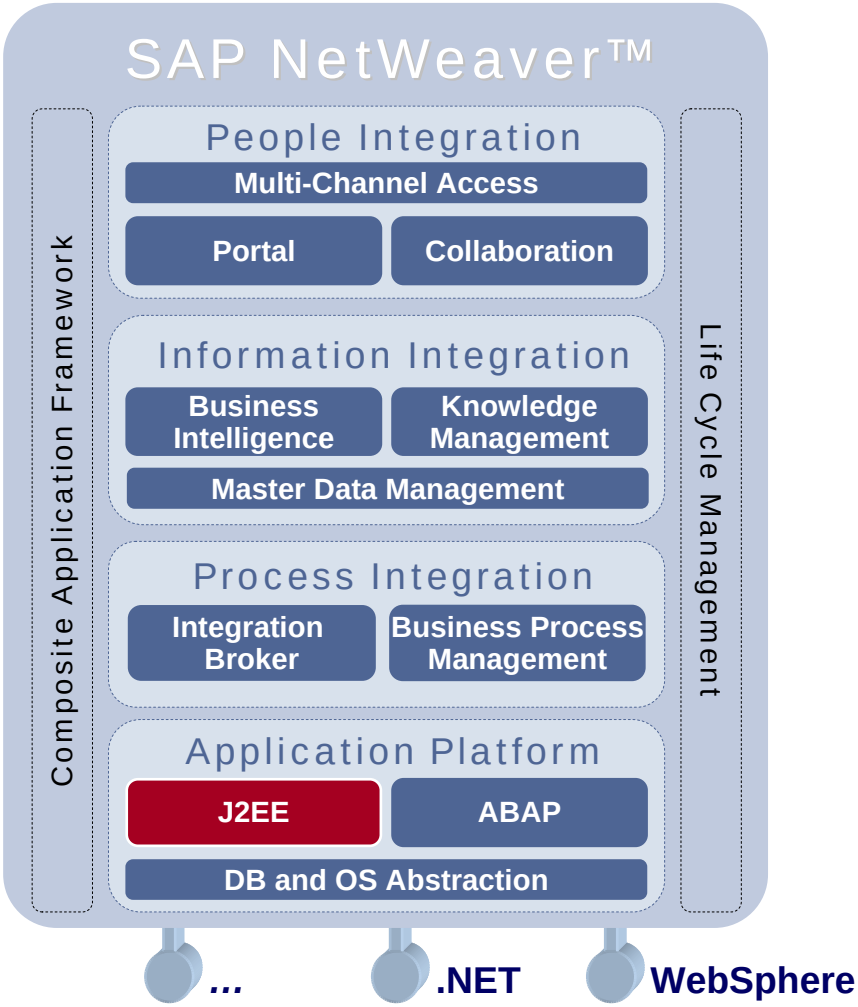
Anteile bei existierenden transaktionsbasierten Anwendungen

Relative Anteile der Entwicklungsplattformen existierenden transaktionsbasierten Anwendungen



Quelle:Ovum

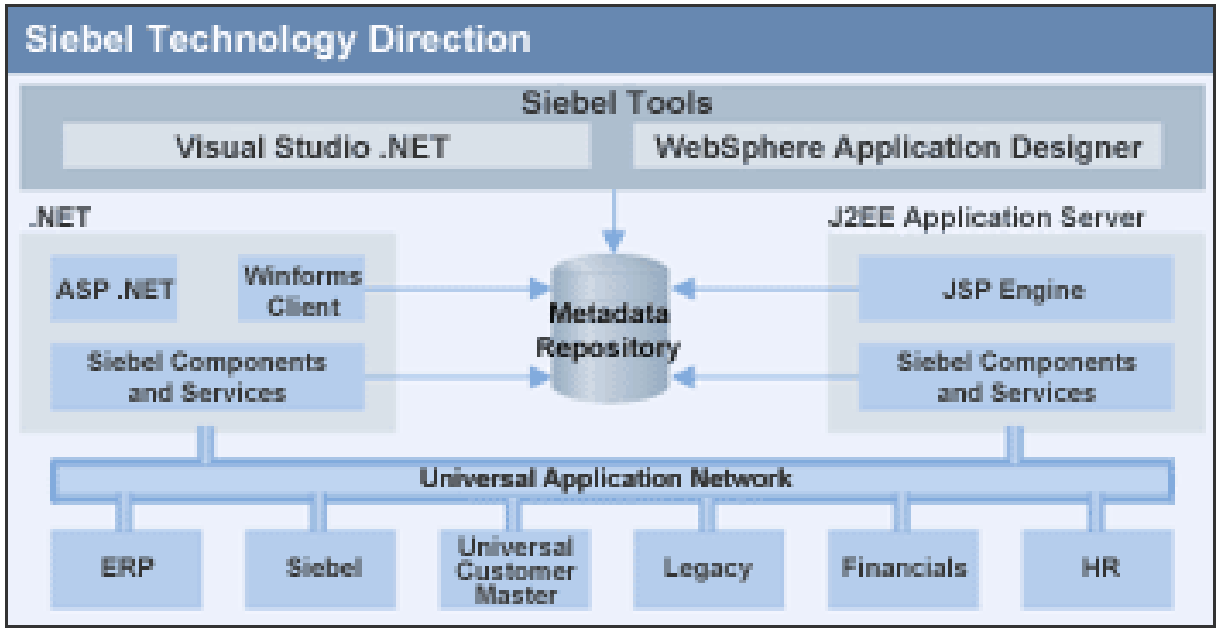
Beispiel: SAP



- NetWeaver-Plattform und interne Entwicklungen basieren auf J2EE und ABAP
- J2EE ist die Plattform für Neuentwicklungen
- .NET wird „unterstützt“

Quelle: SAP

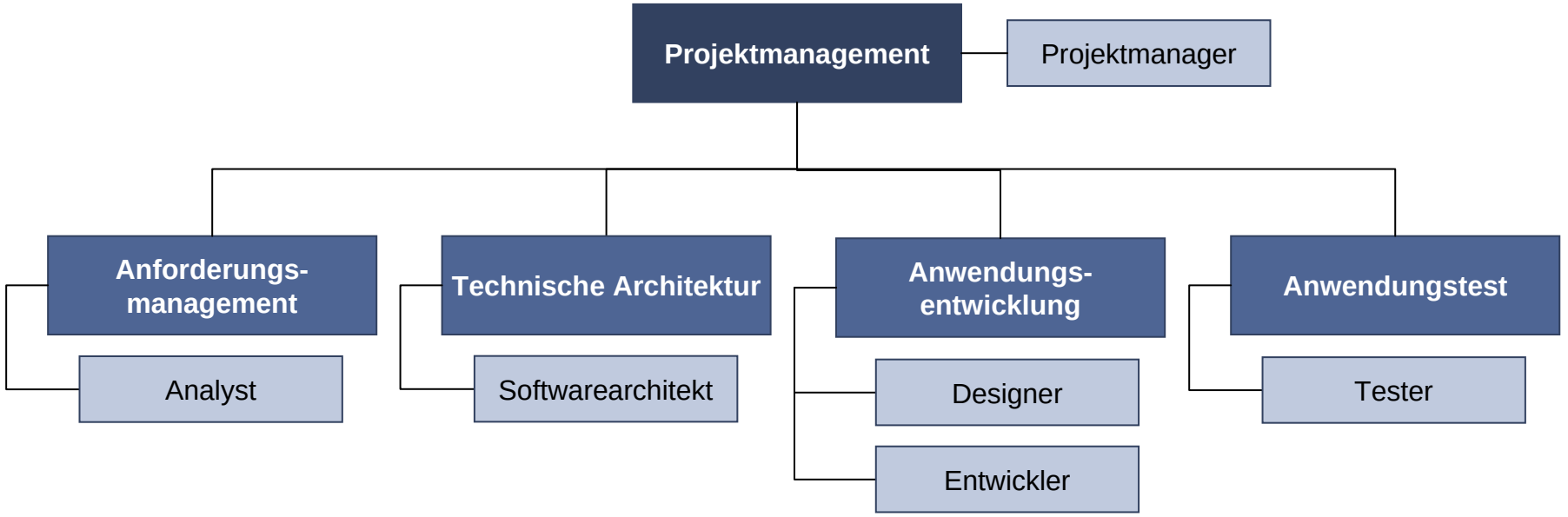
Beispiel: Siebel



- Aktuelle Versionen unterstützen J2EE und .NET
- Ältere Versionen basierten nur auf .NET
- Web Services ermöglichen Interoperabilität zwischen den beiden Plattformen

Quelle: Siebel

Rollen in einer Projektorganisation



Verantwortlichkeiten und beispielhafte Aufgaben (1/3)

- Analyst
  - Analyse und Definition der fachlichen Anforderungen
  - Erstellung des Fachkonzepts
- Softwarearchitekt
  - Entwurf der Softwarearchitektur
  - Definition der grund-legenden technischen Entscheidungen
  - Definition des projekt-weiten Rahmens für die einzelnen Entwurfs-entscheidungen

- Analyst
  - Erstellung eines HTML-basierten Prototypen
- Softwarearchitekt
  - Identifikation der Komponenten der Softwarearchitektur
  - In welcher Schicht der Softwarearchitektur wird die Validierung der Benutzereingaben ausgeführt?

Verantwortlichkeiten und beispielhafte Aufgaben (2/3)

- Designer
  - Entwurf eines Teils der Anwendung bzw. des Systems unter Berücksichtigung der Anforderungen, der Softwarearchitektur und den projektweiten Entwurfsrichtlinien
- Entwickler
  - Implementierung und Test der einzelnen Systemkomponenten

- Designer
  - Entwurf einer Komponente mit ihren Schnittstellen im Kontext der Softwarearchitektur
- Entwickler
  - Test im Kontext der Softwarearchitektur
  - Test in den verschiedenen Umgebungen

Verantwortlichkeiten und beispielhafte Aufgaben (3/3)

- Tester
  - Planung, Koordination und Durchführung des Anwendungstests
- Projektmanager
  - Planung und Steuerung des Projektes
  - Koordination der Projektressourcen
  - Abstimmung der fachlichen Anforderungen und Projektprioritäten unter Berücksichtigung der technischen Rahmenbedingungen

- Tester
  - Definition und Koordination der Testumgebungen
- Projektmanager
  - Erstellung des Arbeitsplans
  - Erstellung der Projektorganisation
  - Anforderungs-management

Agenda

Einführung und Motivation

Verteilte Objekte und Komponenten

Verteilte Softwarearchitekturen

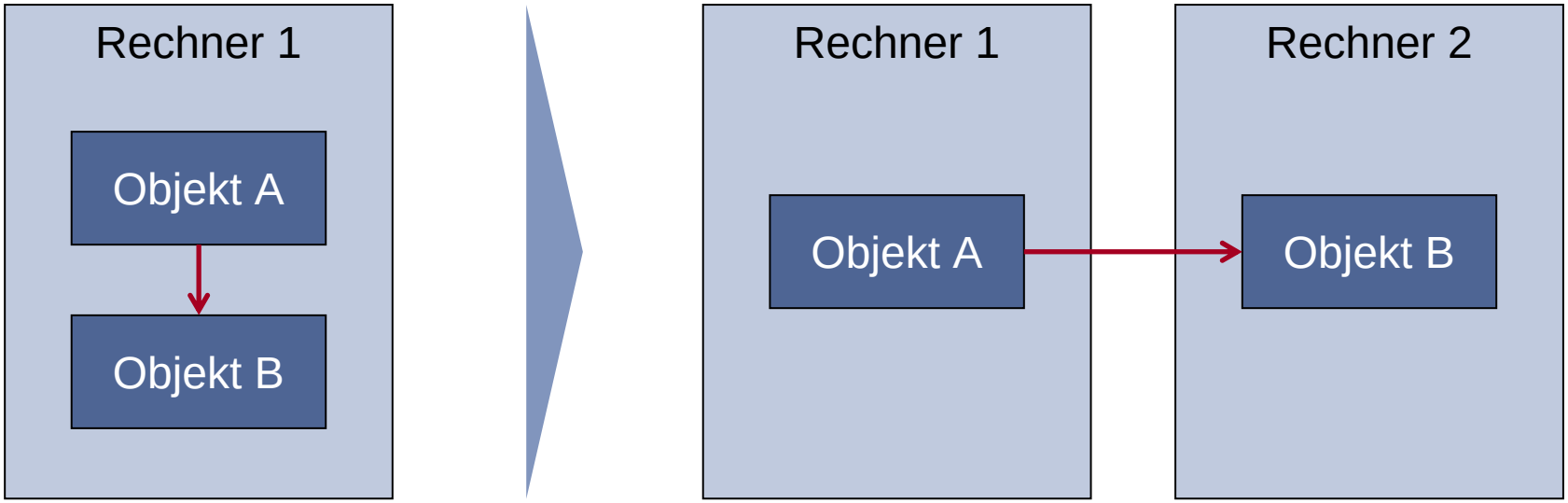
J2EE-Plattform

J2EE-basierte Softwarearchitektur

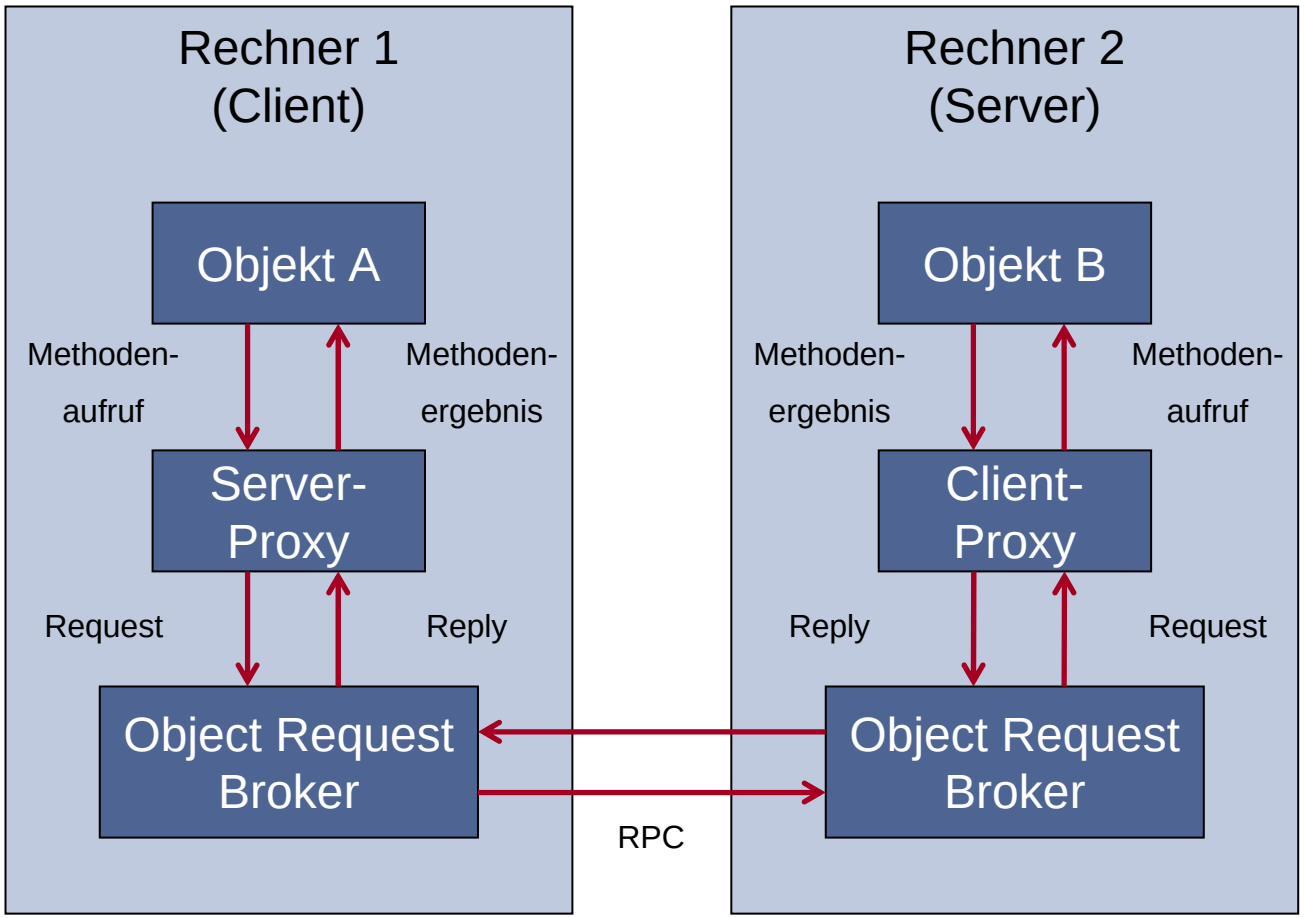
Aspekte der Verteilung von J2EE-Anwendungen

Motivation

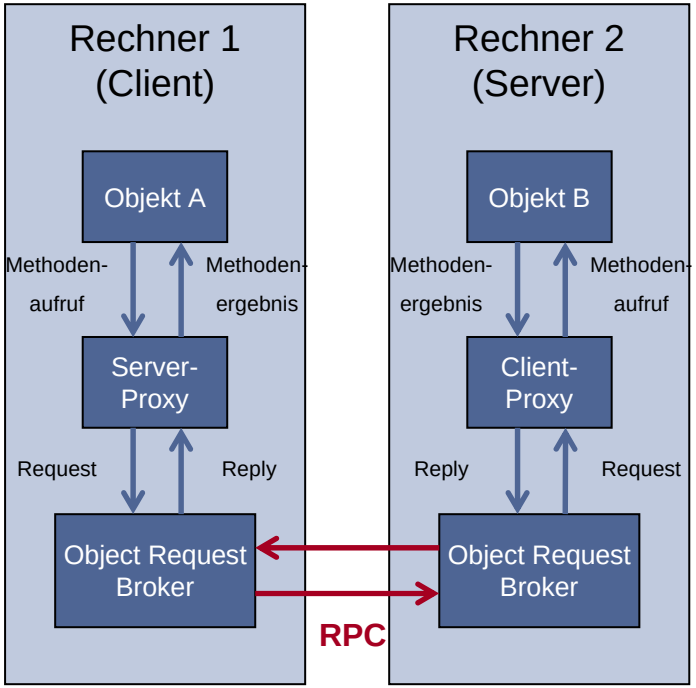
- Verbindung des objekt-orientierten Paradigmas mit den Konzepten verteilter Systeme
- Rechner- und betriebssystemübergreifender Zugriff eines Objekts auf die Methoden eines anderen Objekts



Object Broker Pattern



Implikationen der Verteilung



- Performance
  - bei einem verteilten Methodenaufruf erfolgt eine RPC-Kommunikation über das Netzwerk
  - Granularität der Objekte: Objekt vs. Komponente
- Zuverlässigkeit
  - das Netzwerk ist eine zusätzliche Fehlerquelle
- Sicherheit
  - die RPC-Kommunikation muss gesichert werden
- Komplexität
  - höher: RPC-Kommunikation und Ortstransparenz

Agenda

Einführung und Motivation

Verteilte Objekte und Komponenten

Verteilte Softwarearchitekturen

J2EE-Plattform

J2EE-basierte Softwarearchitektur

Aspekte der Verteilung von J2EE-Anwendungen

Begriffsdefinition: Softwarearchitektur

- Keine einheitliche, allgemein anerkannte Definition
- Beispiele:
  - Balzert, Helmut:
    „Eine Software-Architektur ist eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Komponenten.“
    ([Balz96], Seite 639)
  - Bass, Len; Clements, Paul; Kazman, Rick:
    „The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.“
    ([BaCK03], Seite 21)

Eigenschaften einer Softwarearchitektur (1/2)

- Systeme können durch mehrere Strukturen bzw. Anordnungen beschrieben werden
  - jede Struktur bzw. Anordnung definiert eine Sicht auf das identische System
- Kontext der Vorlesung:
  - Softwarearchitektur = Anwendungsarchitektur
  - Vergleiche: Anwendungsarchitektur vs. Datenarchitektur

Quelle: in Anlehnung an [BaCK03]

Eigenschaften einer Softwarearchitektur (2/2)

- Definition von Softwareelementen und deren Beziehungen
  - interne Informationen über die Softwareelemente sind nicht relevant, sofern sie die Beziehung zwischen zwei Elementen nicht beeinflussen (Prinzip der Abstraktion)
  - Softwareelemente interagieren über Schnittstellen
  - Schnittstellen trennen öffentliche Informationen über Softwareelemente von privaten (internen) Informationen (Prinzip des Information Hiding)

Quelle: in Anlehnung an [BaCK03]

Ziele einer Softwarearchitektur (1/2)

- Entwurf einer Anwendung bzw. eines Softwaresystems auf einem hohen Abstraktionsniveau
  - Instrument zur Kommunikation
  - Grundlage und Rahmen für die detaillierten Entwürfe der einzelnen Softwareelemente
- Identifikation der grundsätzlichen Typen von Softwareelementen
  - Beispiele: Masken/Webseiten, Module, Datenbanken
- Strukturierung und Gruppierung der Softwareelemente
  - Orientierungshilfe in der Menge der konkreten Softwareelemente
  - Grundlage für die Planung innerhalb des Projektmanagements

Quelle: in Anlehnung an [BaCK03] und [Balz96]

Ziele einer Softwarearchitektur (2/2)

- Identifikation von Schnittstellen
  - innerhalb der Anwendung bzw. des Softwaresystems
  - zu anderen Anwendungen bzw. Softwaresystemen
- Definition von grundsätzlichen Entwurfsrichtlinien und -entscheidungen
  - Frühzeitige Entscheidungen, welche nur schwierig und/oder mit großem Aufwand zu ändern sind
  - Beispiel: Technologie der Benutzeroberfläche: Windows-Client, Java-Client, Web-basierter Client

Quelle: in Anlehnung an [BaCK03] und [Balz96]

Sichten auf Softwarearchitekturen (1/2)

- Zahlreiche Sichten auf die identische Softwarearchitektur möglich
  - Beispiel: Rational Unified Process (RUP) empfiehlt 5 Sichten (sog. 4+1 Model) (vgl. [Kruc95])
- Jede Sicht beschreibt bestimmte Aspekte der Softwarearchitektur
  - unterschiedliche Zielgruppen: Projektmanager, Designer, Entwickler, Tester

Sichten auf Softwarearchitekturen (2/2)

- Notation der Beschreibung
  - Unterschiede: objektorientierter vs. strukturierter Entwurf
  - Beispiel: Verwendung der Unified Modelling Language (UML) im Rational Unified Process (RUP)
  - Beispiel: Verwendung von Datenflussdiagrammen und Funktionsbäumen innerhalb der Strukturierte Analyse (nach DeMarco)

Beispiel: Sichten des RUP (1/2)

- Logical View
  - die wichtigsten Klassen/Komponenten auf Entwurfsebene und ihre Organisation in Packages und Subsystems
  - Beziehungen zwischen den Klassen/Komponenten (Nutzung, Vererbung, usw.)
  - Organisation der Packages und Subsystems in Schichten
- Implementation View
  - Verfeinerung und Detaillierung der logischen Sicht
  - Klassen/Komponenten auf Implementierungsebene und ihre Organisation in Packages und Subsystems
  - Beziehungen zwischen den Klassen/Komponenten
  - Organisation der Packages und Subsystems in Schichten

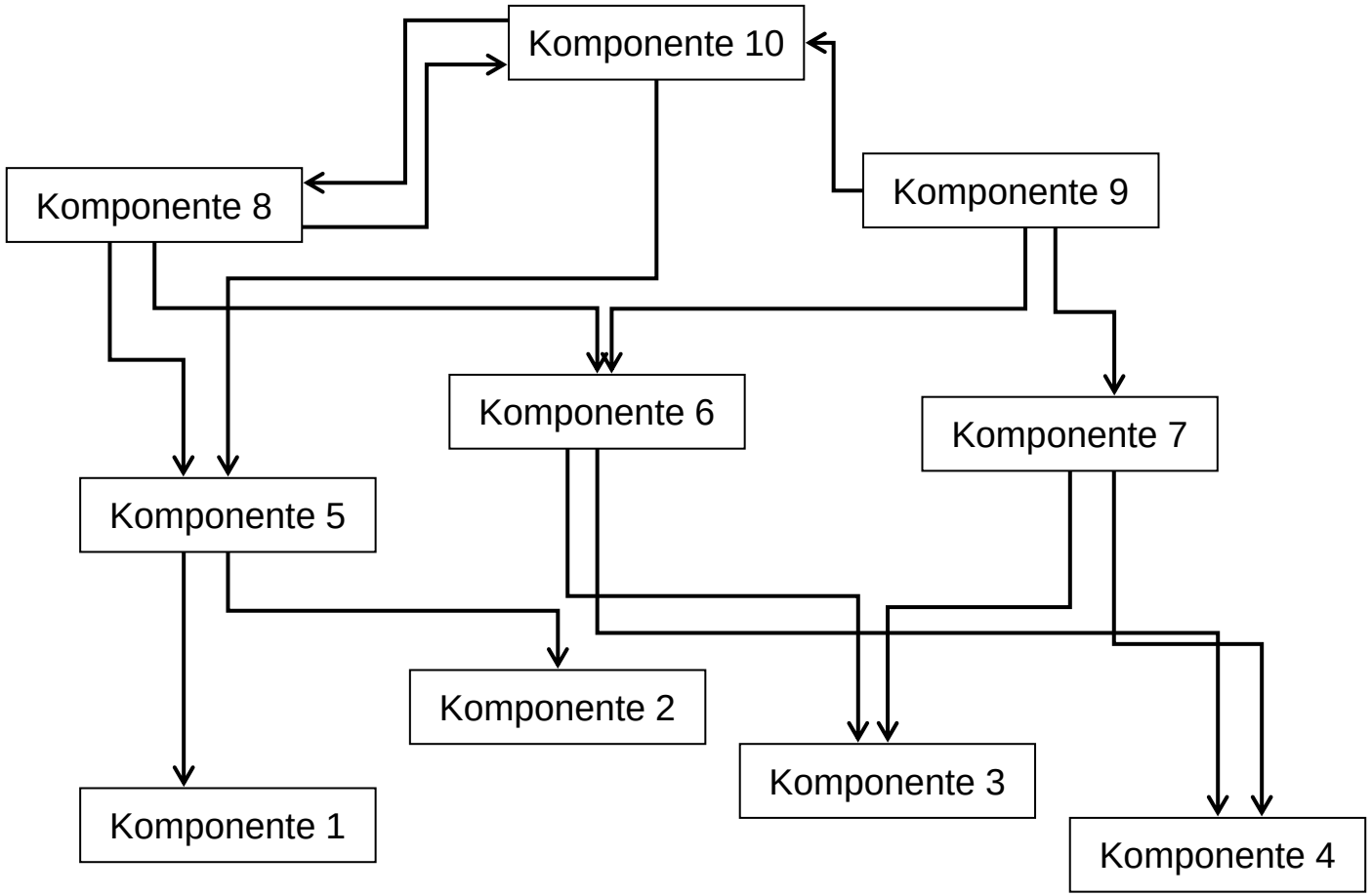
Quelle: in Anlehnung an [Rati03]

Beispiel: Sichten des RUP (2/2)

- Process View
  - Beschreibung der Tasks (Prozesse, Threads), deren Interaktionen und Konfigurationen
  - Allokation der Objekte und Komponenten zu Tasks
- Deployment View
  - Beschreibung der physischen Rechnerknoten (Nodes) und ihre Konfiguration
  - Allokation der Tasks zu Rechnerknoten
- Use Case View
  - Use Cases und Scenarios, welche das wesentliche Verhalten der Softwarearchitektur beschreiben
  - Beispiel: Was passiert innerhalb der Softwarearchitektur, wenn der Benutzer den Button X auswählt?

Quelle: in Anlehnung an [Rati03]

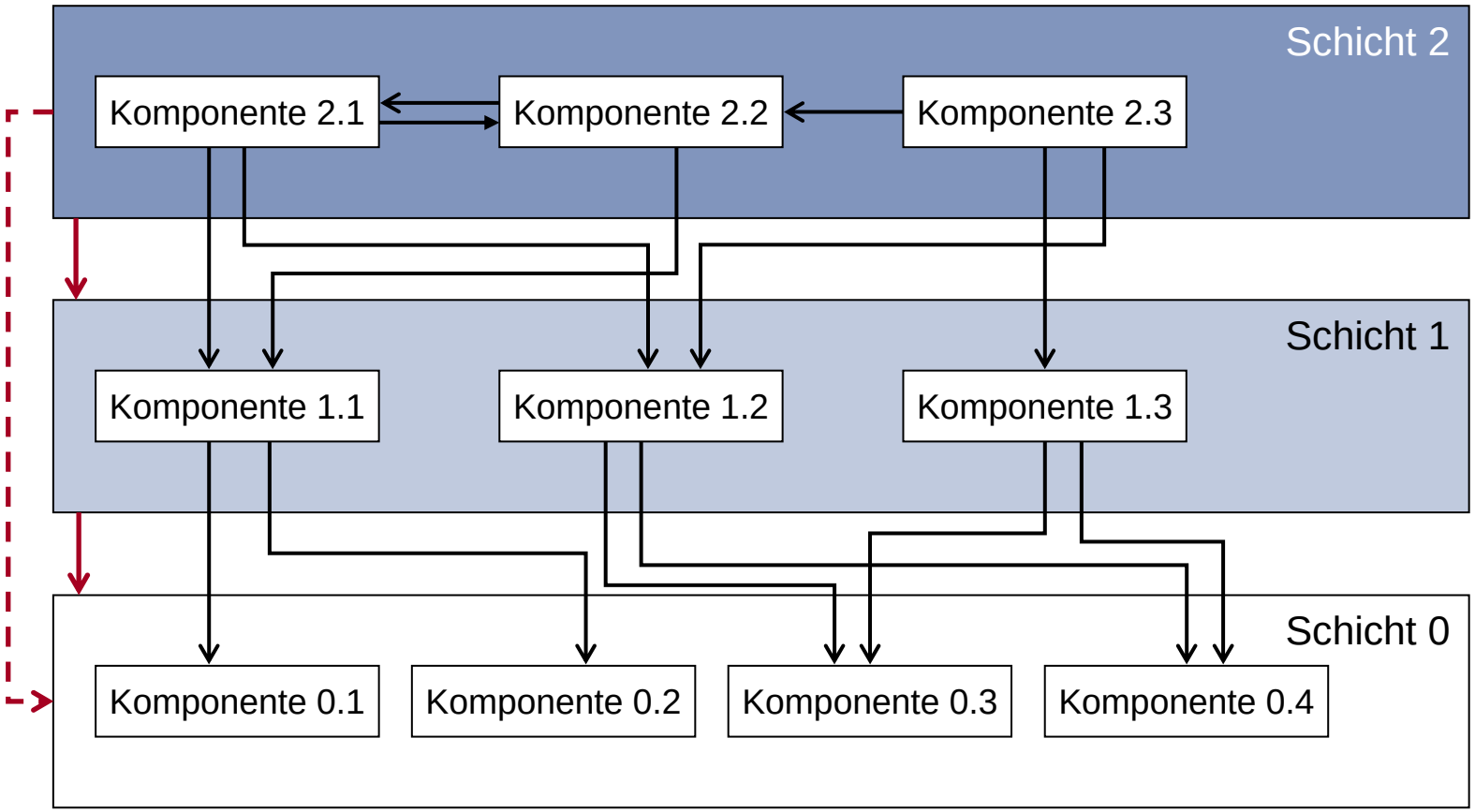
Softwarearchitektur: logische Sicht



Quelle: in Anlehnung an [Balz96]

→ Komponente X nutzt Komponente Y

Softwarearchitektur: logische Sicht mit Schichten



Quelle: in Anlehnung an [Balz96]

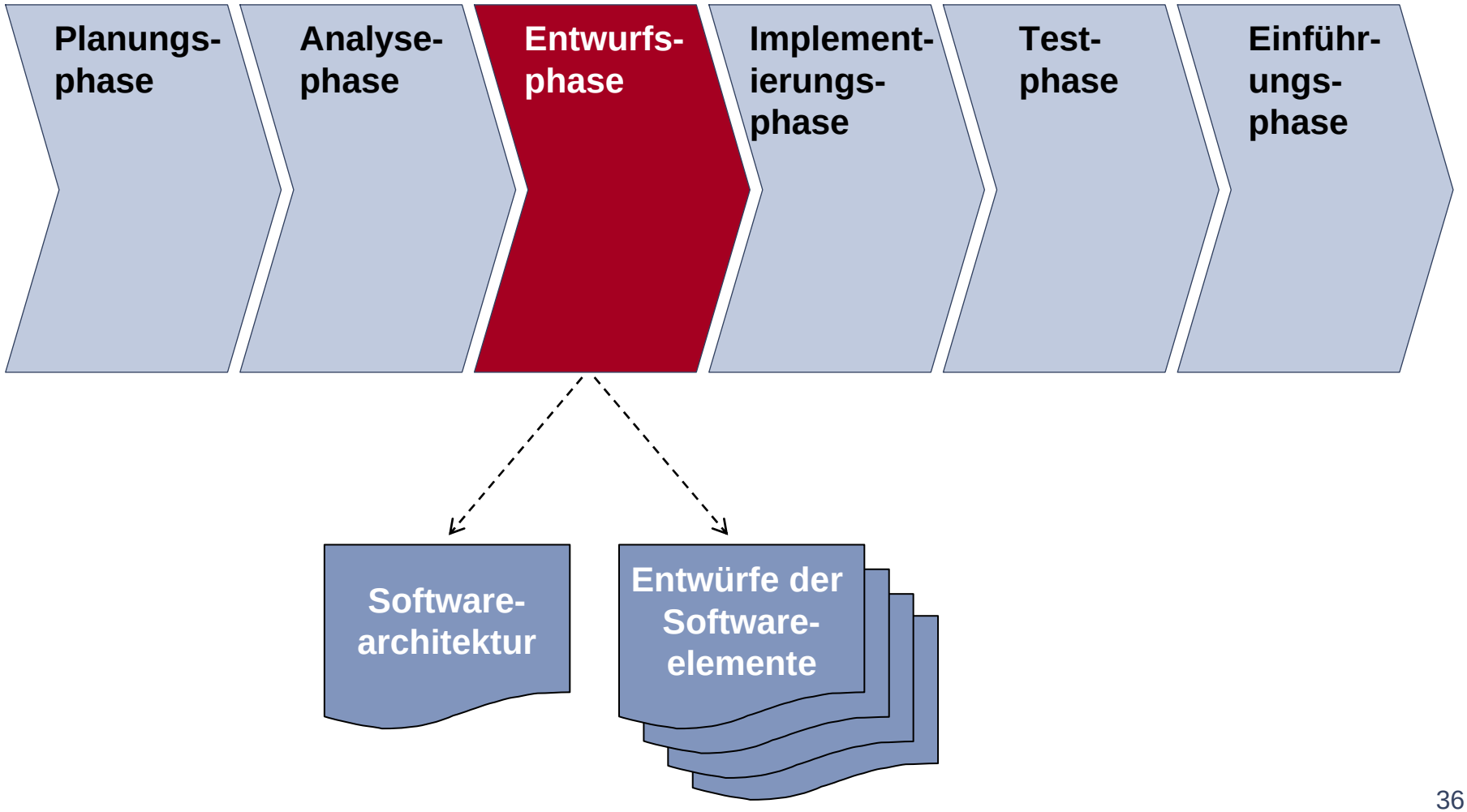
→ Komponente X nutzt Komponente Y

Schichtenarchitekturen

- Gruppierung der Softwareelemente in Schichten
- Softwareelemente können innerhalb einer Schicht beliebig aufeinander zugreifen
- Zwischen den Schichten gelten strengere Zugriffsregeln
- Schichtenarchitektur mit linearer Ordnung:
  - Anordnung der Schichten nach ihrem Abstraktionsniveau
  - Jede Schicht kann nur auf die nächst niedrigere Schicht zugreifen
- Schichtenarchitektur mit strikter Ordnung:
  - Von Schichten mit höherem Abstraktionsniveau kann auf alle Schichten mit niedrigerem Abstraktionsniveau zugegriffen werden – allerdings nicht umgekehrt

Quelle: in Anlehnung an [Balz96]

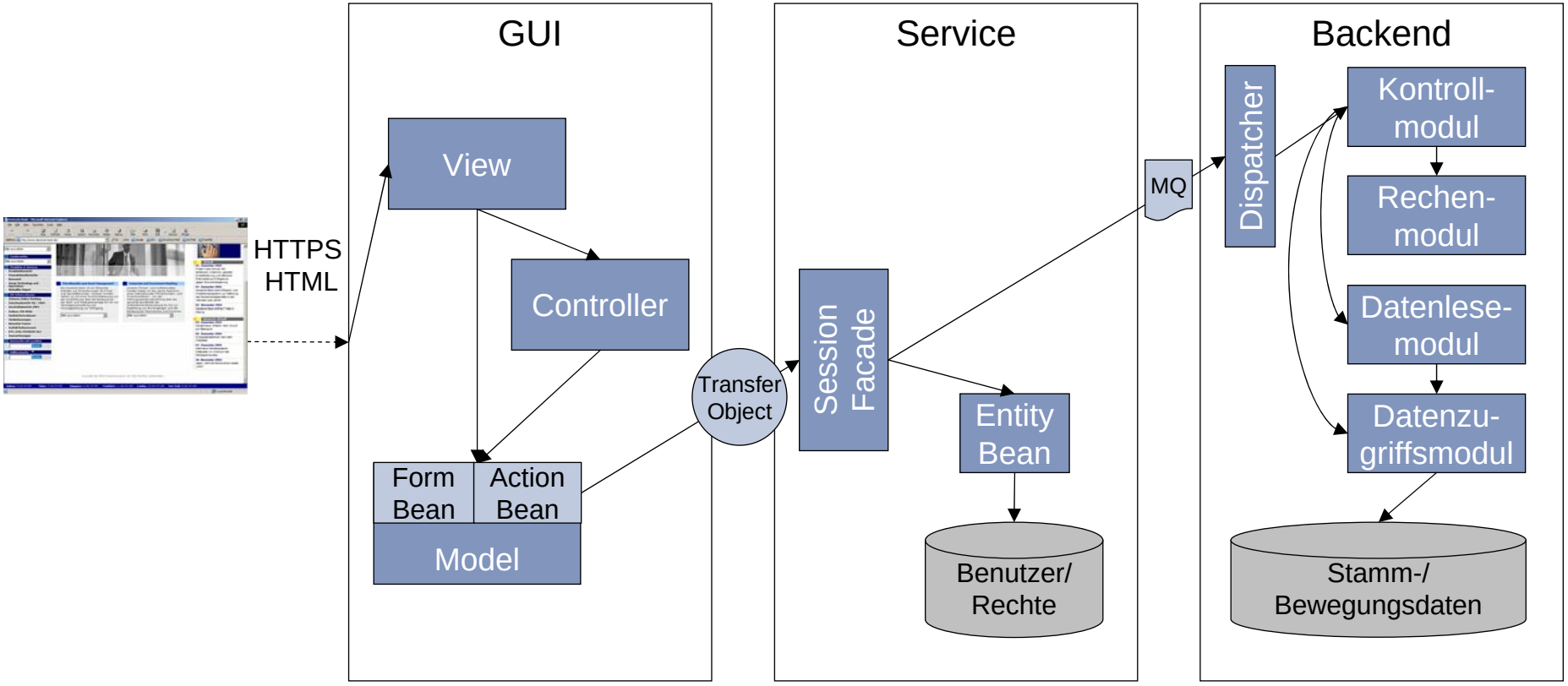
Definition der Softwarearchitektur in der Entwurfsphase



Beispiel: Vorgehensweise in der Entwurfsphase

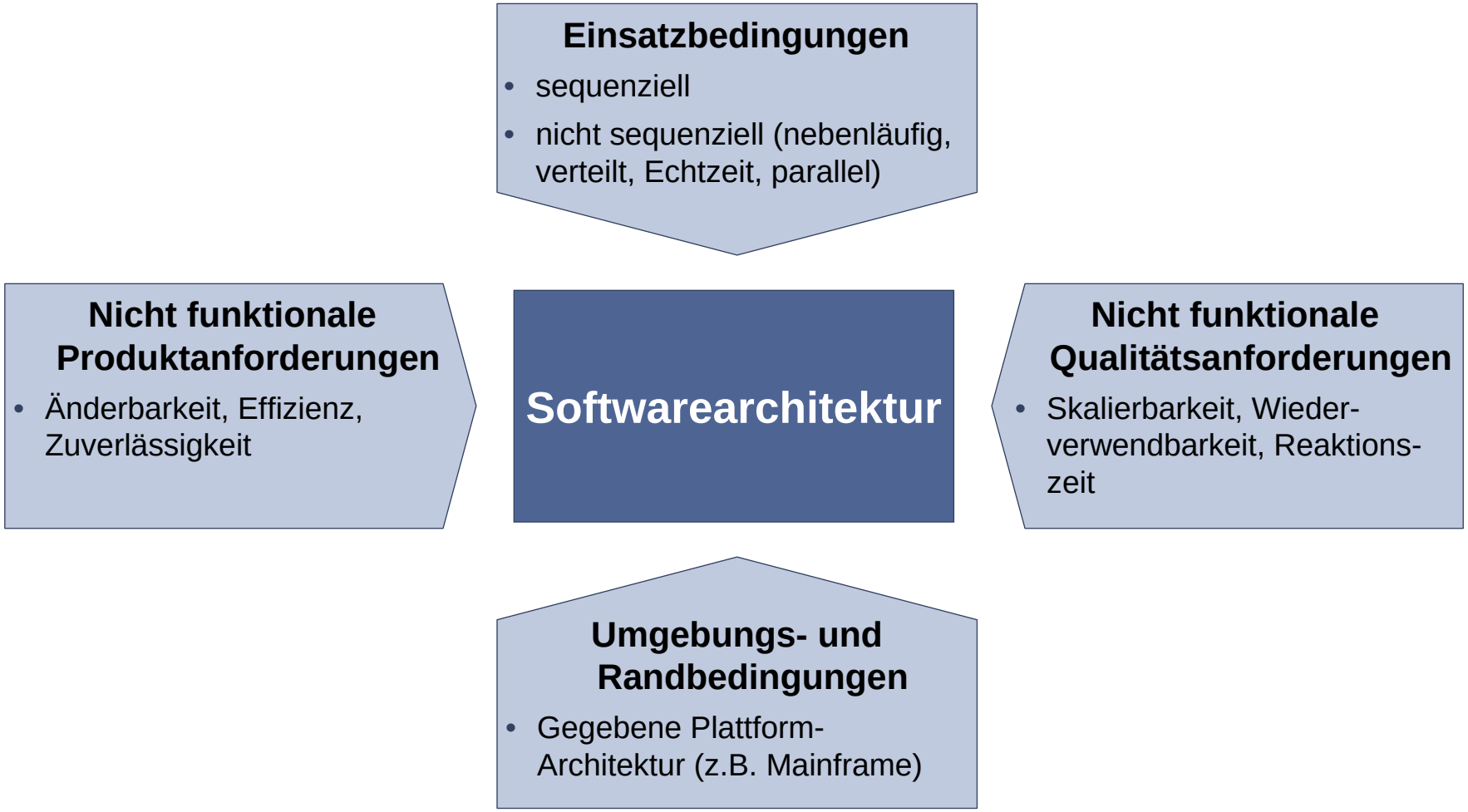
Abbildung ist bewusst nicht in den Folien enthalten!

Beispiel: Entwurf einer Softwarearchitektur



→ Softwareelement X nutzt Softwareelement Y

Einflussfaktoren auf eine Softwarearchitektur



Quelle: in Anlehnung an [Balz96]

Funktionsorientierte Strukturierung in drei Schichten

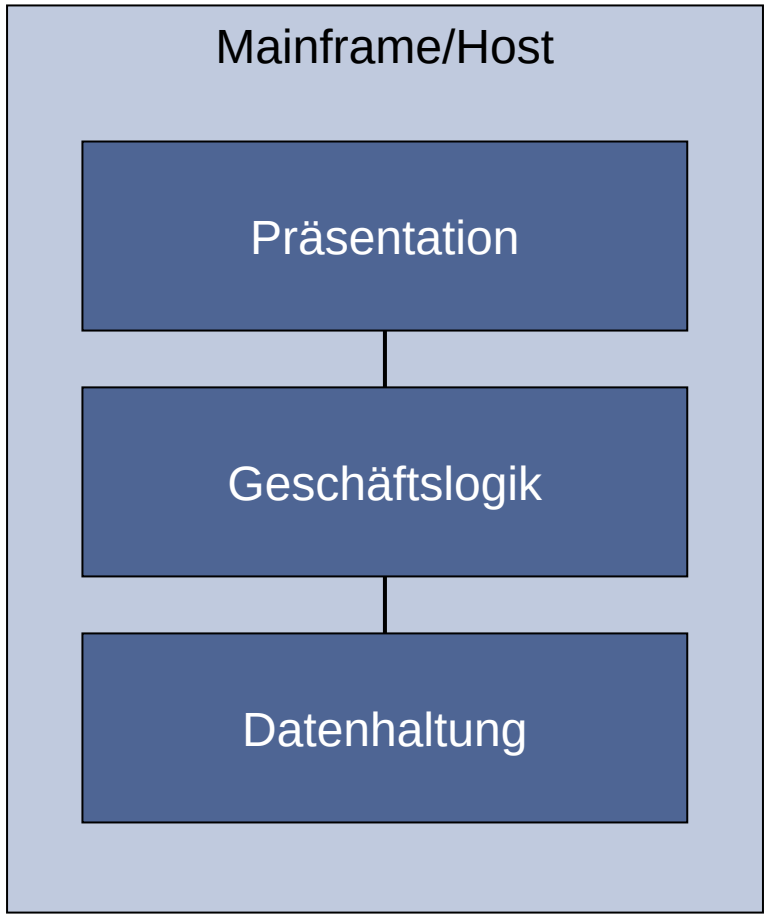
- Präsentation
  - Menüstrukturen, Bearbeitungsmasken, Berichte
  - Beispiel: Web-Client, Windows-Client
- Geschäftslogik
  - Geschäftprozesse, -objekte und -funktionen
  - Beispiel: Verbuchung einer Wareneingangsrechnung, Kalkulation eines Verkaufspreises
- Datenhaltung
  - Verwaltung und Organisation der Daten
  - Beispiel: relationales Datenbankmanagementsystem, Dateien

Verteilung der Schichten auf unterschiedliche Rechnerknoten?

Verteilung der Schichten: Architekturstile und Muster

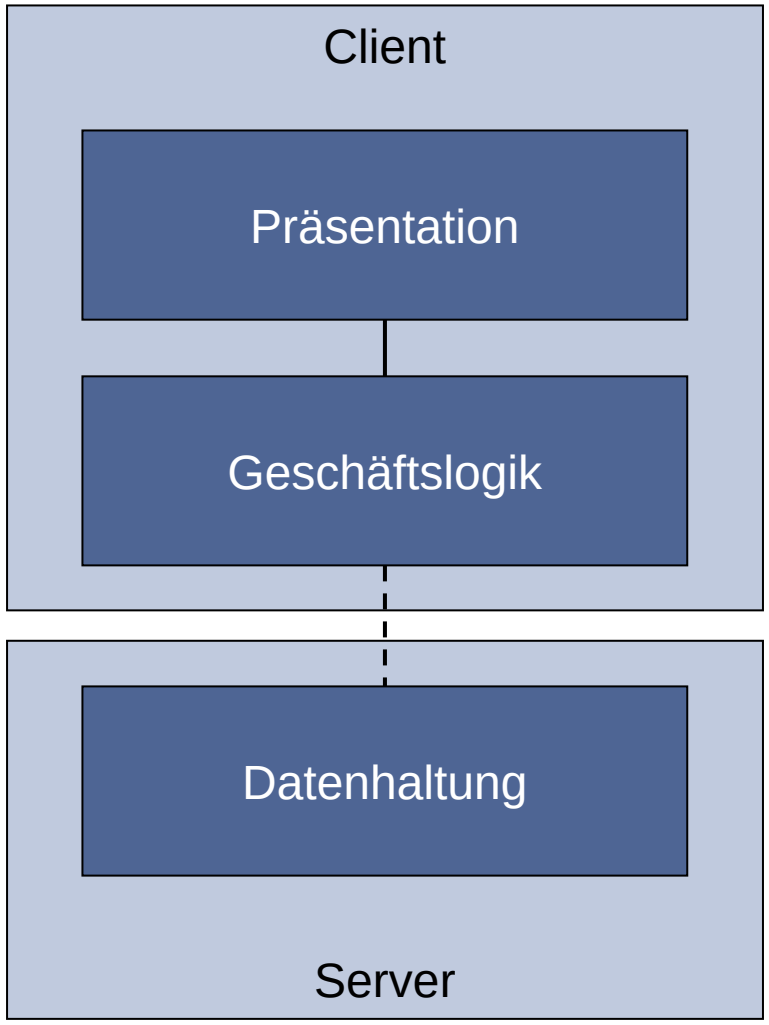
- Definition der Verteilung von Präsentation, Geschäftslogik und Datenhaltung
  - Abhängig von der Entwicklungsplattform, welche bestimmte Parameter der Verteilung definiert
- Grundlegende Architekturstile:
  - Host/Mainframe-Softwarearchitektur
  - Client/Server-Softwarearchitektur
  - Mehrschichtige Softwarearchitektur

Host/Mainframe-Softwarearchitektur



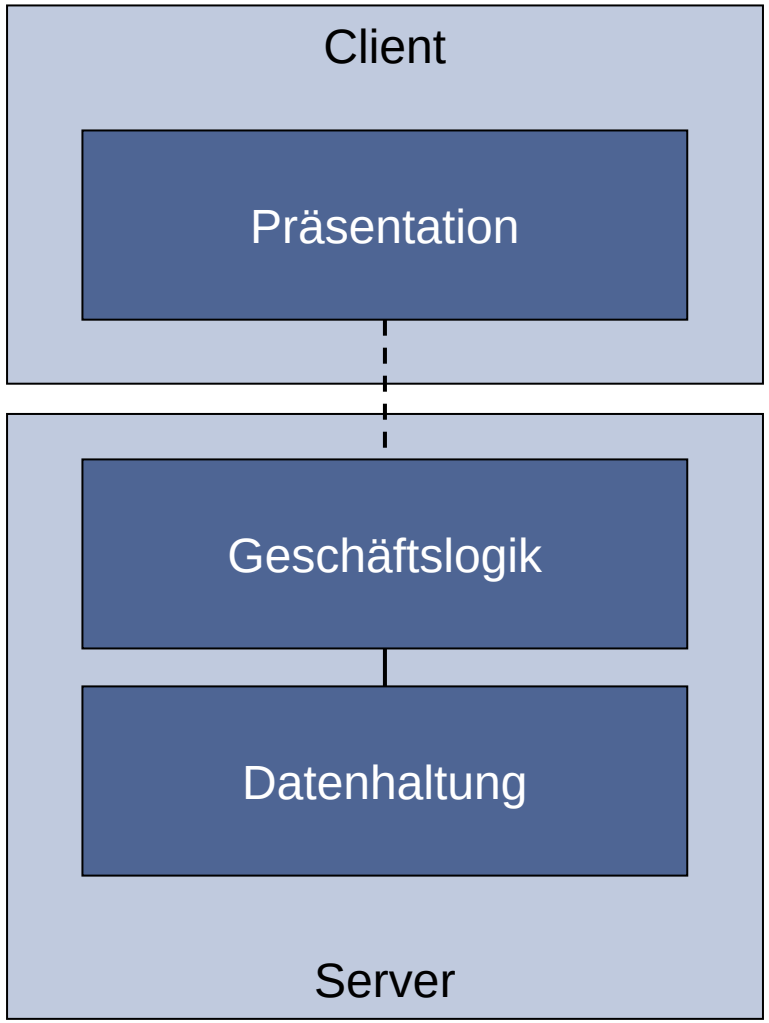
- Präsentations-, Geschäftslogik- und Datenhaltungsschicht liegen auf einem Rechnerknoten
- Beachte: Strukturierung in Schichten ist insbesondere bei Altsystemen nicht immer erfolgt

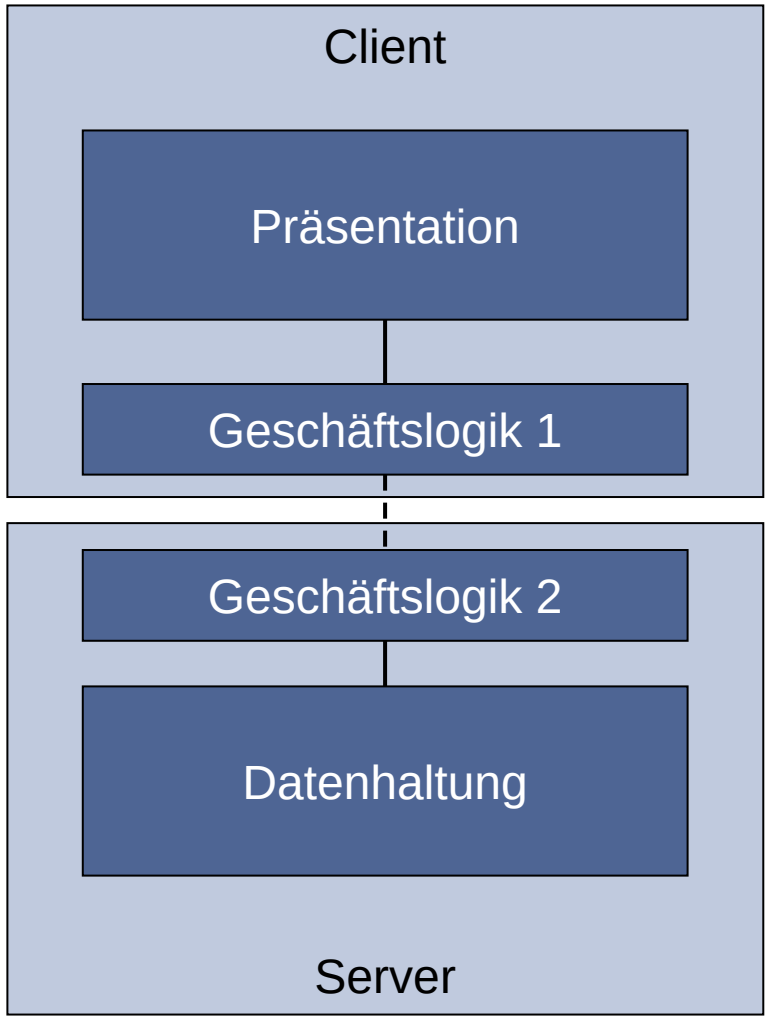
Client/Server-Softwarearchitektur



- Präsentations- und Datenhaltungsschicht auf unterschiedlichen Rechner-knoten
- Verteilung der Geschäftslogik kann variieren
- „Fat Client“

Variationen der Client/Server-Softwarearchitektur

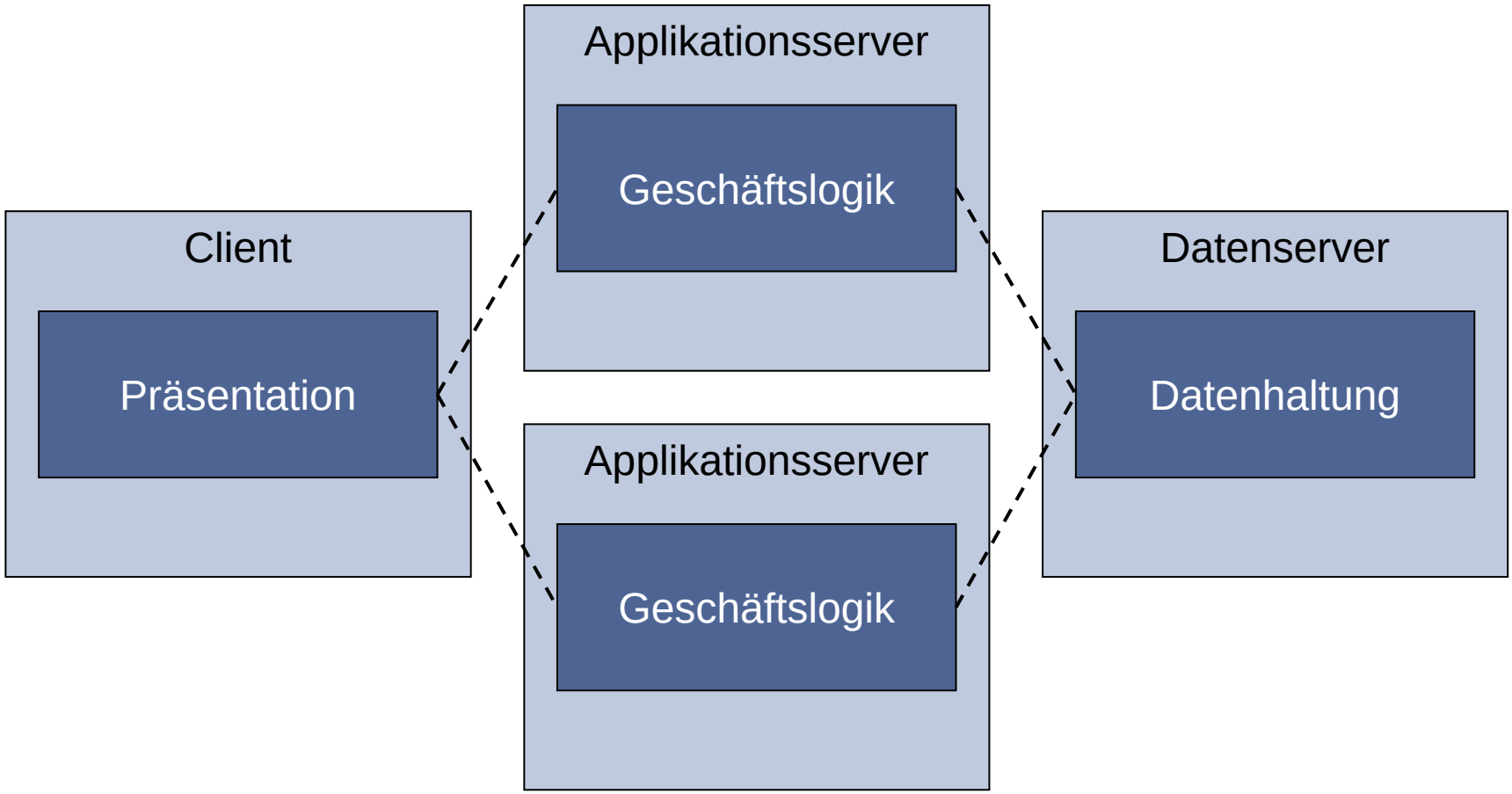




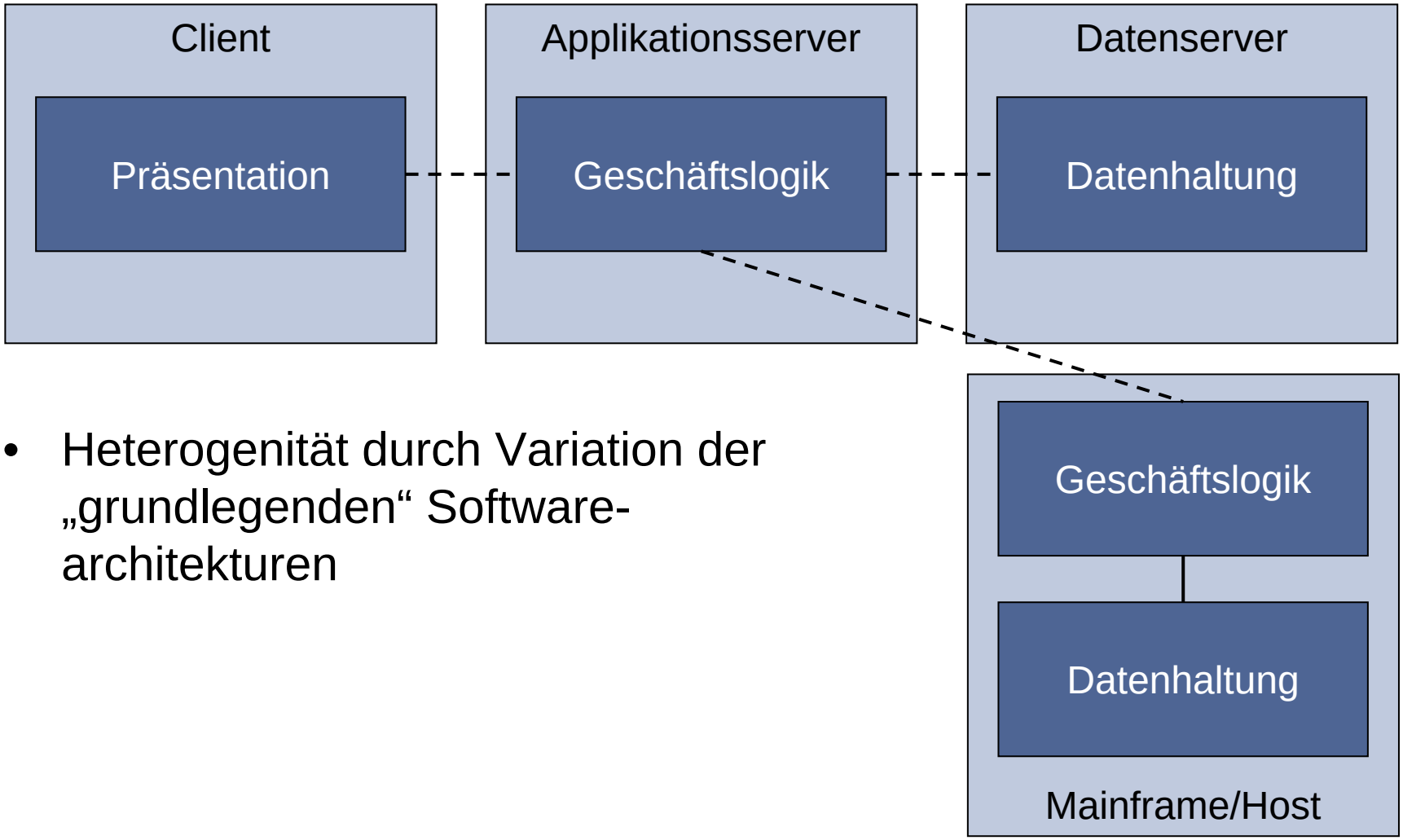
Mehrschichtige Softwarearchitektur

- Verteilung der Schichten auf mehreren Rechnerknoten
- „Thin Client“

Variation der mehrschichtigen Softwarearchitektur



Softwarearchitekturen in der Praxis



- Heterogenität durch Variation der „grundlegenden“ Software-architekturen

Best Practices und Patterns

- J2EE
  - Mehrschichtige Softwarearchitektur
  - „Core J2EE Patterns: Best Practices and Design Strategies“
  - http://www.corej2eepatterns.com/index.htm
- .NET
  - Mehrschichtige Softwarearchitektur
  - „Patterns & Practices“
  - http://www.microsoft.com/resources/practices/default.mspx

Anforderungen und Probleme

- Reduzierung der Entwicklungs- und Betriebskosten
- Schnelle Reaktions-geschwindigkeit
- Öffnung der Unternehmen nach außen
  - Kunden, Lieferanten, Partner
- Ganzheitliche Geschäftsprozesse
- Kurze Lebenszyklen der Produkte- bzw. Dienstleistungen

- Günstige und schnelle Entwicklung von Anwendungen
- Vollständige Integration der unternehmens-internen Anwendungen
- Skalierbarkeit der Anwendungen
- Differenzierte Definition von Sicherheits-eigenschaften
- Hohe Komplexität

Agenda

Einführung und Motivation

Verteilte Objekte und Komponenten

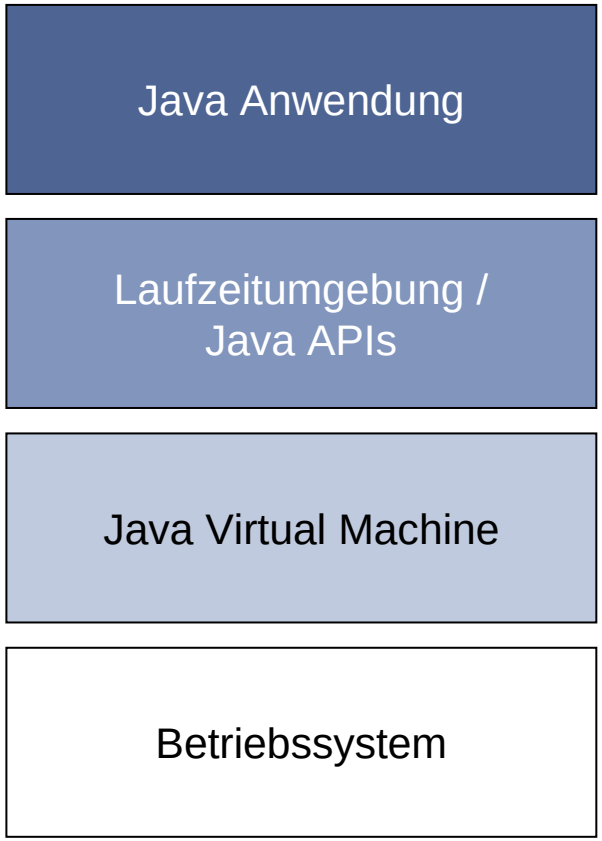
Verteilte Softwarearchitekturen

J2EE-Plattform

J2EE-basierte Softwarearchitektur

Aspekte der Verteilung von J2EE-Anwendungen

Java-Plattform

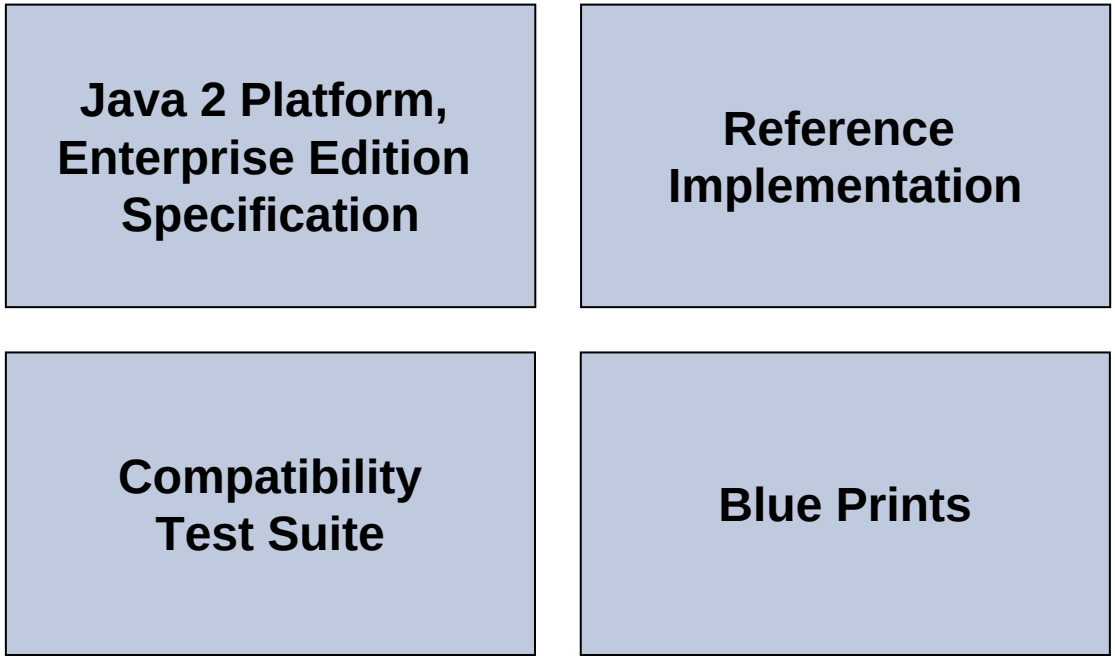


- software-basierte Plattform, welche auf verschiedenen Betriebssystemen implementiert ist
- Java Virtual Machine
  - Windows, Unix, Mainframe
- Laufzeitumgebung / Java APIs
  - grundlegende Klassen und Schnittstellen
  - Abstraktion vom Betriebssystem (z.B. Dateien, Threads)

Quelle: in Anlehnung an http://java.sun.com/

Begriffsdefinition: J2EE

- Definition einer Plattform, welche Kosten und Komplexität von verteilten, mehrschichtigen Anwendungen reduziert
- Definition einer standardisierten Softwarearchitektur, welche aus vier Elementen besteht



Quelle: in Anlehnung an [Sun04a]

Positionierung von J2EE

- Java 2 Platform, Standard Edition (J2SE)
  - APIs, Compiler, Tools, Runtimes zur Entwicklung und Ausführung von Applets und PC-Anwendungen
- Java 2 Platform, Enterprise Edition (J2EE)
  - basiert auf J2SE
  - weitere APIs, Tools und Services
  - Entwicklung und Ausführung von komponenten-basierten, mehrschichtigen Anwendungen
- Java 2 Platform, Micro Edition (J2ME)
  - Technologien und Spezifikationen im Umfeld von eingebetteten Systemen und Multimedia-Geräten
  - PDAs, Mobiltelefone, TV Set-top boxes usw.

Quelle: siehe http://java.sun.com/

Java 2 Platform, Enterprise Edition Specification

- Kein Produkt, sondern eine Spezifikation
  - Spezifikation vs. Standard
- Veröffentlichung und Weiterentwicklung innerhalb des Java Community Process
  - Konsortium von IT-Herstellern
  - Industrie-Standard bzw. De-facto-Standard
  - http://jcp.org
  - aktuelle Version ist v1.4
- Definition einer standardisierten Softwarearchitektur
- Umsetzung in einer Referenzimplementierung und konkreten Produkten

Referenzimplementation

- Sun Java System Application Server Platform Edition
  - frei verfügbarer, J2EE-kompatibler Applikationsserver
  - ab Version 8 wird die Version 1.4 der Spezifikation unterstützt
  - http://java.sun.com/j2ee/1.4/download.html

J2EE-kompatible Produkte (Beispiele)

- WebSphere Application Server
  - kommerzieller Applikationsserver
  - http://www.ibm.com/software/webservers/appserv/was/
  - ab Version 6.0 wird die Version 1.4 der Spezifikation unterstützt
- BEA WebLogic Server
  - kommerzieller Applikationsserver
  - http://www.bea.com/
  - Version 8.1 unterstützt die Version 1.3 der Spezifikation

J2EE-kompatible Produkte (Beispiele)

- Oracle Application Server
  - kommerzieller Applikationsserver
  - Version 10g wird die Version 1.4 der Spezifikation unterstützen
  - http://www.oracle.com/
- SAP Netweaver / SAP Web Application Server
  - kommerzieller Applikationsserver
  - Version 6.30 unterstützt die Version 1.3 der Spezifikation
  - http://www.sap.com/

J2EE-kompatible Produkte (Beispiele)

- JBoss
  - Open-Source Applikationsserver
  - Version 4.0 unterstützt die Version 1.4 der Spezifikation
  - http://www.jboss.org/

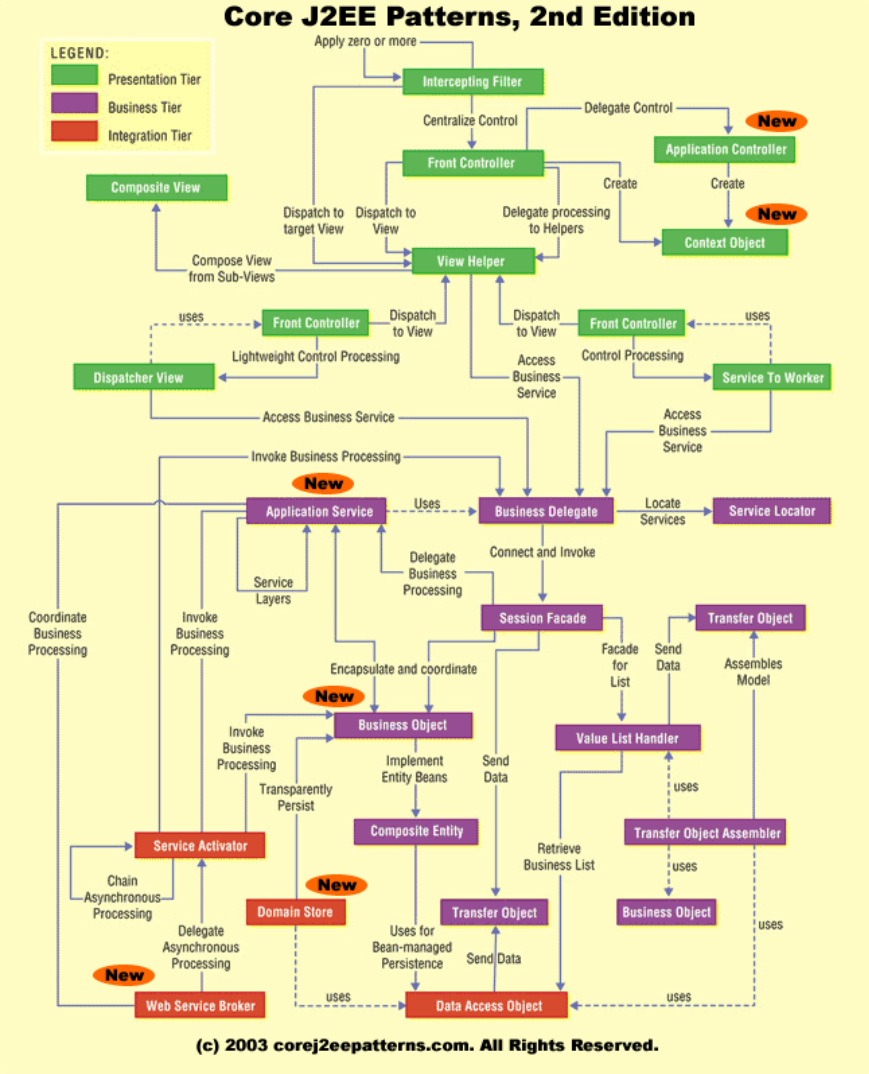
Compatibility Test Suite

- Überprüfung einer Anwendung
  - Java Application Verification Kit (AVK) for the Enterprise
  - Werkzeug zur Überprüfung der Kompatibilität von Anwendungen
  - Kompatibilität = Anwendung läuft in allen Applikationsservern
- Lizenzierung eines Applikationsservers
  - Attribut "J2EE-kompatibel" muss lizenziert und die Compatibility Test Suite muss durchlaufen werden
  - http://java.sun.com/j2ee/verified/

Blue Prints

- auch: Java Blue Prints Program
- Richtlinien
  - u.a. Konventionen für Dateinamen und Verzeichnisstrukturen
- Patterns
  - auf der Ebene der Softwarearchitektur
  - J2EE-Pattern-Katalog
  - Singh, I.; Stearns, B.; Johnson, M. et al. - Designing Enterprise Applications with the J2EE Platform, Second Edition
- Beispielanwendungen
  - z.B. Java Pet Store Sample Application

J2EE-Pattern-Katalog



Quelle: http://www.corej2eepatterns.com/

Agenda

Einführung und Motivation

Verteilte Objekte und Komponenten

Verteilte Softwarearchitekturen

J2EE-Plattform

J2EE-basierte Softwarearchitektur

Aspekte der Verteilung von J2EE-Anwendungen

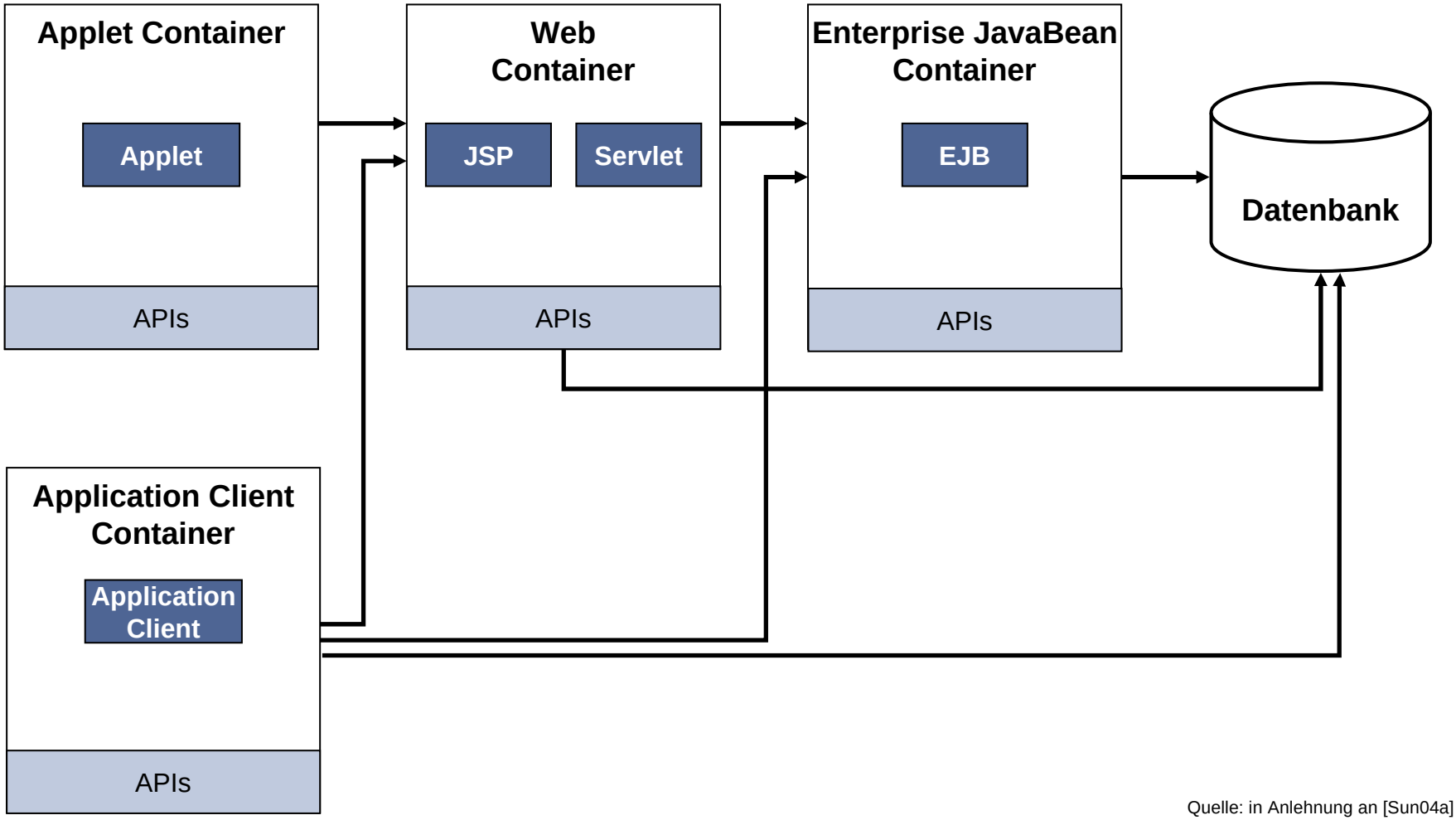
Begriffsdefinition: Komponente

- Komponente = Komponente bzw. Softwareelement einer Softwarearchitektur
- Beispiele (siehe Kapitel ‚verteilte Softwarearchitekturen'):
  - Balzert, Helmut:
    „Eine Software-Architektur ist eine strukturierte oder hierarchische Anordnung der Systemkomponenten sowie Beschreibung ihrer Komponenten."
    ([Balz96], Seite 639)
  - Bass, Len; Clements, Paul; Kazman, Rick:
    „The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them."
    ([BaCK03], Seite 21)

Komponenten einer J2EE-basierten Anwendung

- Anwendungskomponenten
  - Komponenten, welche anwendungsspezifische Funktionen implementieren
- Container
  - Laufzeitumgebung für Anwendungskomponenten einer J2EE-Architektur
- Dienste
  - Komponenten, welche allgemeine Funktionen implementieren und von einem Container bereitgestellt werden
  - Zugriff über ein Application Programming Interface (API)
- Server
  - Kann ein oder mehrere Container verwalten

Standardisierte Softwarearchitektur



Quelle: in Anlehnung an [Sun04a]

Anwendungskomponenten

- Applets
  - werden in einem Web Browser ausgeführt
  - besitzen eine grafische Oberfläche
- Application Clients
  - Java-Anwendungen, welche auf einem PC ausgeführt werden
  - besitzen i.d.R. eine fenster-basierte, grafische Oberfläche
- Web-Komponenten
  - Servlets und JavaServer-Pages
  - können HTTP-basierte Requests bearbeiten
- Enterprise JavaBeans
  - server-seitige Komponenten, welche i.d.R. die Geschäftslogik implementieren und im Kontext von Transaktionen aufgerufen werden

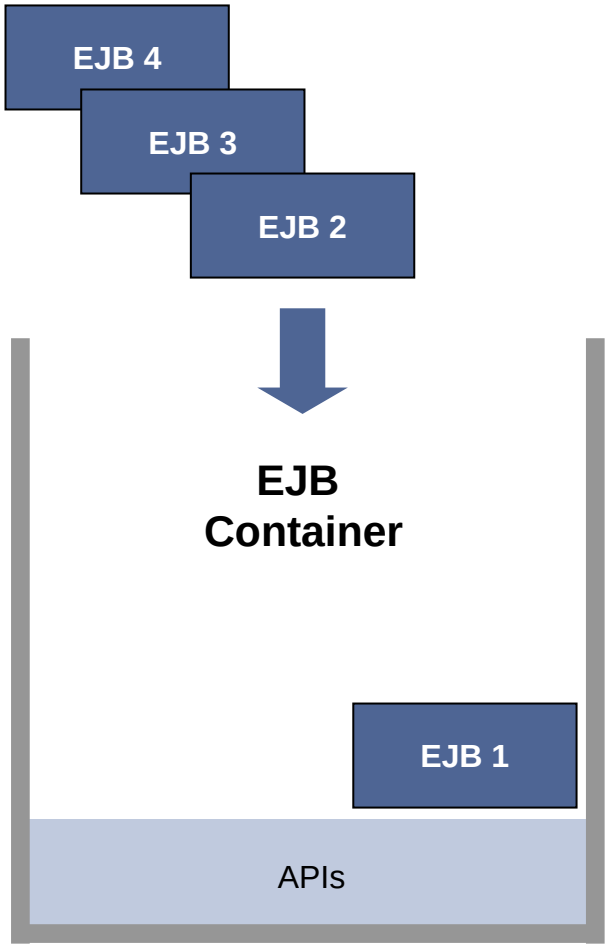
Quelle: in Anlehnung an [Sun04a]

Container-Konzept

- Laufzeitumgebung für Anwendungskomponenten einer J2EE-Architektur
- Bereitstellung von APIs, welche von den Anwendungs-komponenten genutzt können
- Container nutzen Protokolle, um miteinander zu kommunizieren
- Bereitstellung von Diensten
  - Verzeichnis- und Namensdienste, Datenbankverbindungen, Transaktionen, Nachrichtendienste usw.
- Trennung der Geschäftslogik von den Diensten der J2EE-Plattform

Quelle: in Anlehnung an [Sun04a]

Eigenschaften eines Containers



Beispiel: EJB Container

- J2EE-Spezifikation definiert, welche APIs von den Containern angeboten werden müssen
- Container-Werkzeuge zur Verwaltung des Containers und der Anwendungs-komponenten
- Interpretation eines standardisierten Dateiformats, um die Anwendungskomponenten im Container installieren zu können

Vorteile des Container-Konzepts

- Implementation von grundlegenden Klassen und Funktionen
  - APIs der J2SE und J2EE
- Implementation und Kapselung von Kommunikations-protokollen
- Bereitstellung und Wiederverwendung von Diensten
  - Management von Verbindungen (Connection Pooling)
  - Unterstützung von Transaktionen
  - Sicherheitsfunktionen
  - Lebenszyklusmanagement
  - Verzeichnisdienste
- Plattform-Unabhängigkeit
- Skalierbarkeit
  - Kapselung der benötigten Funktionen durch die Container

Nachteile des Container-Konzepts

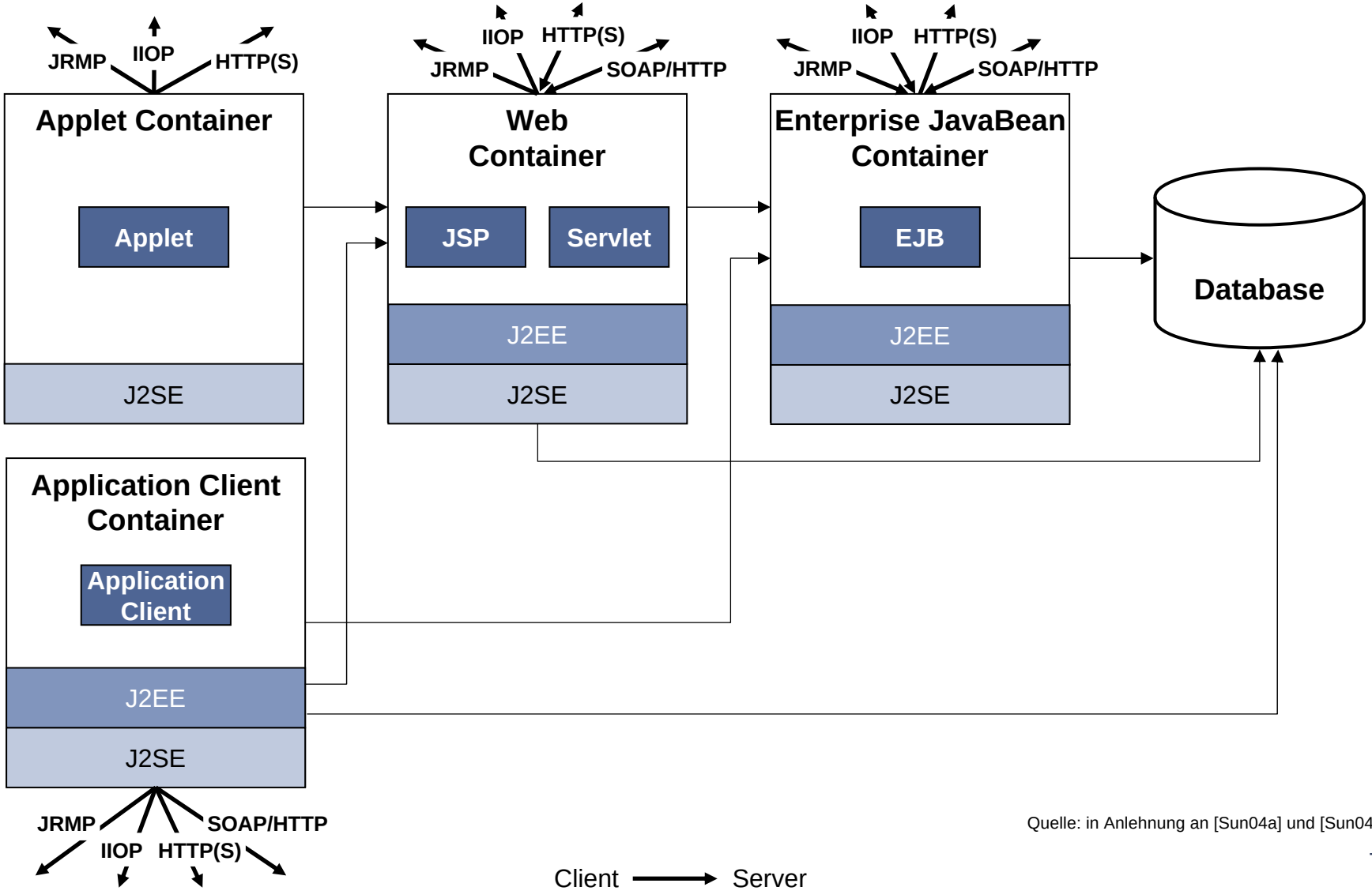
- Abhängigkeit der Anwendung vom Container
  - Container implementiert die Laufzeitumgebung und Dienste
  - Absturz des Containers führt zum Absturz der Anwendung
- Abhängigkeit der Anwendung von anderen Anwendungen
  - Anwendung A kann die Stabilität von Anwendung B gefährden, wenn beide Anwendungen im identischen Container ausgeführt werden
- Implementation der vom Container geforderten Schnittstellen notwendig
  - Flexibilität vs. Wiederverwendung

Klassifikation der Dienste

- Protokolle und Datenformate
  - Ziel ist die Interoperabilität zwischen verschiedenen Anwendungskomponenten und Containern
  - Gruppen von unterstützten Protokollen und Datenformaten:
    - Internet und WWW-Protokolle
    - Protokolle der Object Management Group (OMG)
    - Protokolle der Java Technology
    - Diverse Datenformate
- Application Programming Interfaces (APIs)
  - auch: Programmierschnittstelle
  - analog zum Konzept der Protokolle und der ausgetauschten Nachrichten
  - Fokus auf Java-basierte APIs

Quelle: in Anlehnung an [Sun04a]

Dienste der Container - Überblick



Quelle: in Anlehnung an [Sun04a] und [Sun04b]

Internet und WWW-Protokolle (1/2)

- TCP/IP-Protokollfamilie
  - TCP und UDP über IP
  - Unterstützung durch die J2SE, welche die jeweilige Betriebssystemfunktionalität kapselt
  - siehe http://www.ietf.org und http://www.rfc-editor.org
- HTTP 1.1
  - Unterstützung durch die J2SE, welche die jeweilige Betriebssystemfunktionalität kapselt
  - Client-seitiges API: java.net Package; Server-seitig: Servlet und JSP-Interfaces
  - Web Container muss HTTP-basierte Dienste auf dem Port 80 bereitstellen können
  - siehe http://www.ietf.org und http://www.rfc-editor.org

Internet und WWW-Protokolle (2/2)

- SSL 3.0, TLS 1.0
  - Web Container muss HTTPS-basierte Dienste auf dem Port 443 bereitstellen können
  - SSL 3.0: siehe http://wp.netscape.com/eng/ssl3/
  - TLS 1.0: siehe http://www.ietf.org und http://www.rfc-editor.org
- SOAP 1.1
  - Unterstützung der SOAP-Spezifikation auf der Basis eines HTTP-basierten Transportprotokolls
  - siehe http://www.w3.org/2000/xp/Group/
- WS-I Basic Profile 1.0
  - Detaillierte Parametrisierung des „Web Service-Protokollstack“ (SOAP, WSDL, UDDI) zur Sicherstellung der Interoperabilität zwischen verschiedenen Web Service-Implementationen
  - siehe http://www.ws-i.org

Protokolle der Object Management Group (OMG) (1/2)

- Internet Inter-ORB Protocol (IIOP) 1.2
  - Teil der CORBA-Spezifikation der OMG: Kommunikationsprotokoll zwischen Object Request Brokern
  - Verwendung des IIOP als Protokoll für Java Remote Method Invocation (Java RMI)
  - siehe http://www.omg.org/cgi-bin/doc?formal/99-10-07
- EJB Interoperability Protocol
  - Detaillierte Parametrisierung des IIOP zur Nutzung als Kommunikationsprotokoll zwischen Enterprise JavaBeans (EJB)
  - siehe Sun Microsystems - Enterprise JavaBeans Specification, Version 2.1 ([Sun04b])

Protokolle der Object Management Group (OMG) (2/2)

- CORBA Interoperable Naming Service Protocol
  - IIOP-basiertes Protokoll zur Implementation eines verteilten Namens- und Verzeichnisdienstes
  - siehe http://www.omg.org/cgi-bin/doc?formal/00-06-19

Protokolle der Java Technology

- Java Remote Method Protocol (JRMP)
  - Protokoll der ersten Versionen der Java Remote Method Invocation (Java RMI) APIs
  - siehe http://java.sun.com/j2se/1.4.2/docs/guide/rmi/

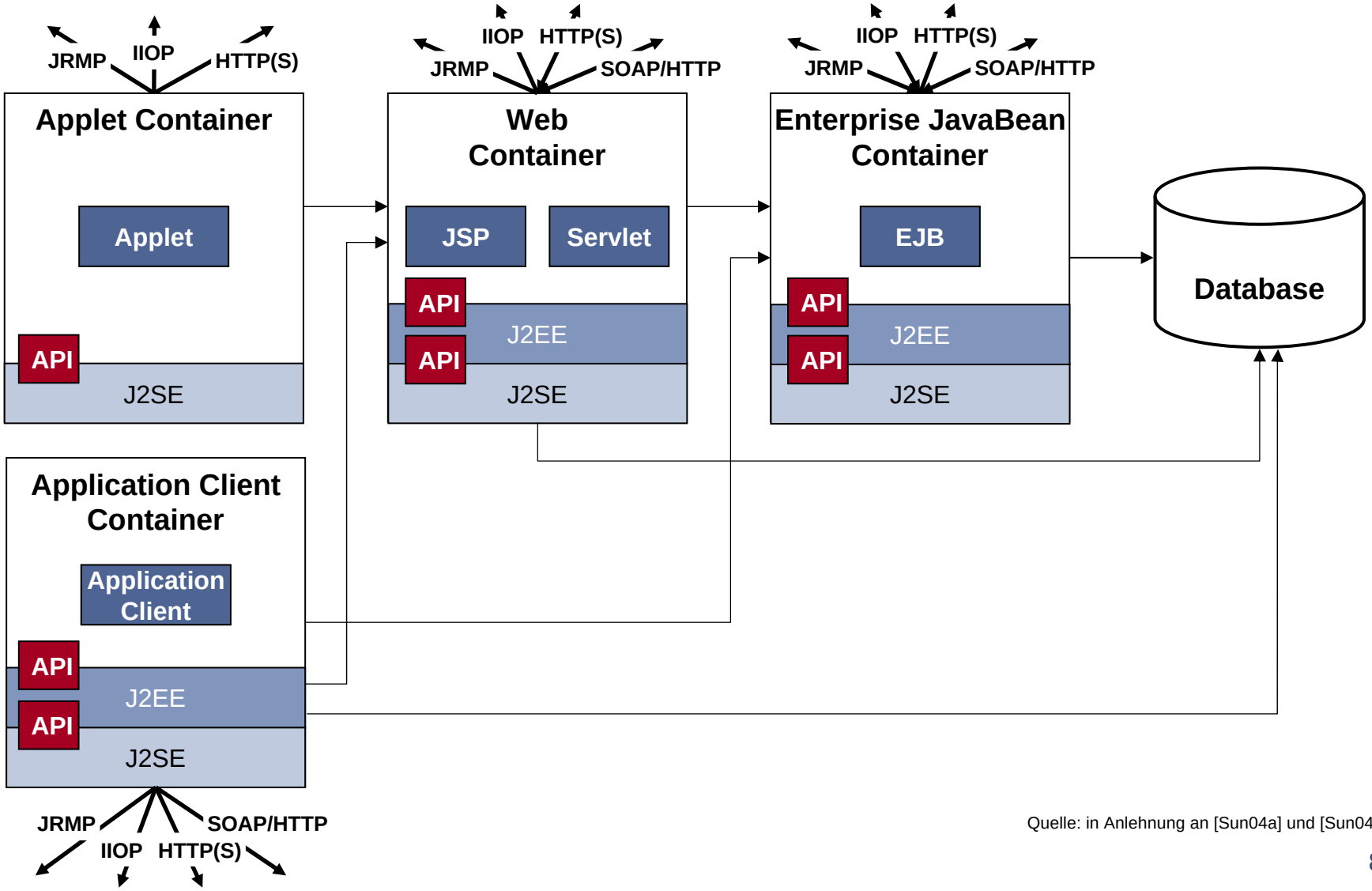
Datenformate (1/2)

- XML 1.0
  - Basis für XML-basierte Dokumente und Nachrichten
  - siehe http://www.w3.org/XML/Core/
- HTML 3.2
  - Keine direkte Unterstützung durch APIs, aber Web Clients müssen HTML darstellen können
  - siehe http://www.w3.org/MarkUp/
- Bildformate
  - Unterstützung der Formate GIF und JPEG entsprechend der APIs im Package java.awt.image
  - http://java.sun.com/j2se/1.4.2/docs/api/java/awt/package-summary.html

Datenformate (2/2)

- Java Archives (JAR)
  - Standardformat zur Paketierung und Installation von Java-basierten Anwendungskomponenten
  - Installation von Applets vs. Installation im Container
  - siehe http://java.sun.com/j2se/1.4.2/docs/guide/jar/
- Java Class File Format
  - Datenformat zur Ausführung von Klassen in der Java Virtual Machine
  - siehe http://java.sun.com/docs/books/vmspec/

Dienste der Container - Überblick



Quelle: in Anlehnung an [Sun04a] und [Sun04b]

APIs der J2SE

- Erforderliche Unterstützung der APIs der Java 2 Standard Edition, Version 1.4 durch alle Containertypen
  - siehe http://java.sun.com/j2se/1.4.2/docs/
- Folgende APIs sind in Bezug auf die J2SE optional, für die J2EE allerdings verbindlich
  - Java RMI over IIOP (RMI-IIOP)
  - Java IDL
  - Java Database Connectivity (JDBC) API
  - Java Naming and Directory Interface (JNDI)
  - Java API For XML Processing (JAXP)
  - Java Authentication and Authorization Service (JAAS)

APIs der J2EE (1/5)

API	Beschreibung der Dienste/Schnittstellen
Servlet 2.4	Erweiterung der Funktionalität eines Web Servers, um dynamische Webseiten auf der Basis der Java-Plattform zu erstellen; Gegenstück zu Technologien wie CGI und ASP
JavaServer Pages (JSP) Specification 2.0	Einbettung von Java-Code und vordefinierten Befehlen in statische HTML-Seiten; JSPs werden mit einem JSP-Compiler in Servlets übersetzt
Enterprise JavaBeans (EJB) Specification 2.1	Server-seitige, verteilte Komponentenarchitektur
Java Messaging Service (JMS) Specification 1.1	Programmierschnittstelle zwischen einem Container und einer Nachrichten-basierten Middleware

APIs der J2EE (2/5)

API	Beschreibung der Dienste/Schnittstellen
Java Transaction API (JTA) 1.0	Programmierschnittstelle zwischen den beteiligten Parteien einer verteilten Transaktion (Anwendungen, Ressourcen Manager und Application Server)
JavaMail 1.3	Ermöglicht E-Mail-Nachrichten zu erstellen, zu versenden und zu empfangen (MIME, E-Mail Message Stores)
JavaBeans Activation Framework (JAF) 1.0	Unterstützung der Konvertierung von MIME-Datentypen in Java-Objekte

APIs der J2EE (3/5)

API	Beschreibung der Dienste/Schnittstellen
Java API for XML Processing (JAXP) 1.2	Erstellung und Bearbeitung von XML-Dokumenten auf der Basis der Programmiermodelle SAX und DOM; Schnitt-stellen zu einer XSLT-Transformationsengine
J2EE Connector Architecture 1.5	Programmierschnittstelle zu Resource Adaptern, welche die Ausführung von Transaktionen unterstützen
Web Services for J2EE 1.1	Beschreibung der Implementierung und Veröffentlichung eines Web Service Endpoints
Java API for XML-based RPC (JAX-RPC) 1.1	RPC auf der Basis von SOAP

APIs der J2EE (4/5)

API	Beschreibung der Dienste/Schnittstellen
SOAP with Attachments API for Java (SAAJ) 1.2	Erstellung und Bearbeitung von SOAP-basierten Nachrichten
Java API for XML Registries (JAXR) 1.0	Programmierstelle für den Zugriff auf einen Verzeichnisdienst für Web Services entsprechend der Spezifikationen ebXML Registry und/oder UDDI
J2EE Management 1.0	Steuerung und Überwachung von Komponenten der J2EE-Plattform (J2EE Management Model)
Java Management Extensions (JMX) 1.2	Einbindung von Komponenten der J2EE-Architektur in das J2EE Management Model

APIs der J2EE (5/5)

API	Beschreibung der Dienste/Schnittstellen
J2EE Deployment 1.1	Schnittstelle zwischen einem Deployment Tool und einem J2EE-Server
Java Authorisation Service Provider Contract for Containers (JACC) 1.0	Definition von neuen java.security.Permission Klassen zur Implementation des Autorisierungsmodells der J2EE-Plattform

API-Anforderungen an die Container (1/4)

API	Applicat-ion Client	Applet	Web Container	EJB Container
EJB 2.1	Ja	Nein	Ja	Ja
Servlet 2.4	Nein	Nein	Ja	Nein
JSP 2.0	Nein	Nein	Ja	Nein
JMS 1.1	Ja	Nein	Ja	Ja
JTA 1.0	Nein	Nein	Ja	Ja

Quelle: in Anlehnung an [Sun04a]

API-Anforderungen an die Container (2/4)

API	Applicat-ion Client	Applet	Web Container	EJB Container
JavaMail 1.3	Ja	Nein	Ja	Ja
JAF 1.0	Ja	Nein	Ja	Ja
JAXP 1.2	Ja	Nein	Ja	Ja
Connector Architecture 1.5	Nein	Nein	Ja	Ja
Web Services for J2EE 1.1	Ja	Nein	Ja	Ja

Quelle: in Anlehnung an [Sun04a]

API-Anforderungen an die Container (3/4)

API	Applicat-ion Client	Applet	Web Container	EJB Container
JAX-RPC 1.1	Ja	Nein	Ja	Ja
SAAJ 1.2	Ja	Nein	Ja	Ja
JAXR 1.0	Ja	Nein	Ja	Ja
Management 1.0	Ja	Nein	Ja	Ja
JMX 1.2	Ja	Nein	Ja	Ja

Quelle: in Anlehnung an [Sun04a]

API-Anforderungen an die Container (4/4)

API	Applicat-ion Client	Applet	Web Container	EJB Container
Deployment 1.1	Nein	Nein	Nein	Nein
JACC 1.0	Nein	Nein	Ja	Ja

Quelle: in Anlehnung an [Sun04a]

Rollen in der Anwendungsentwicklung

- Große Anzahl und großer Umfang der APIs
- Hohe Anforderungen an Anwendungsentwickler

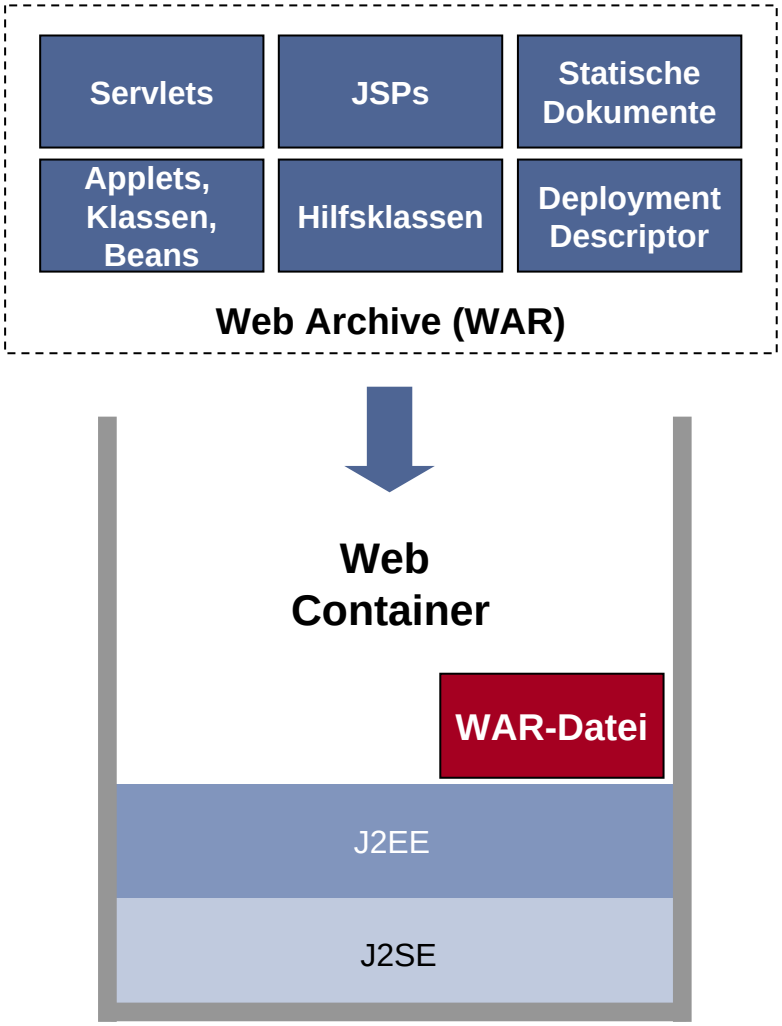
- Dedizierte Rollen in der Anwendungs-entwicklung, z.B.
  - Web Designer
  - Web Entwickler
  - EJB Entwickler
  - Datenbank Entwickler
- Querschnittsrollen, z.B.
  - Technischer Architekt
  - Anwendungsarchitekt

Web-Komponenten einer Anwendung

- Servlets
- JavaServer Pages (JSP)
- Client-seitige Applets, Klassen und Beans
  - vor der Ausführung: Übertragung vom Web-Server auf den Client (Web Browser)
  - Beans (auch: JavaBeans): Client-seitige Komponentenarchitektur
- Hilfsklassen
  - für Server-seitige Ausführung (Servlets)
- Statische Dokumente (HTML, Bilder, Videos, usw.)
- Metainformationen
  - Deployment Descriptor: Konfiguration und Verbindung der einzelnen Web-Komponenten

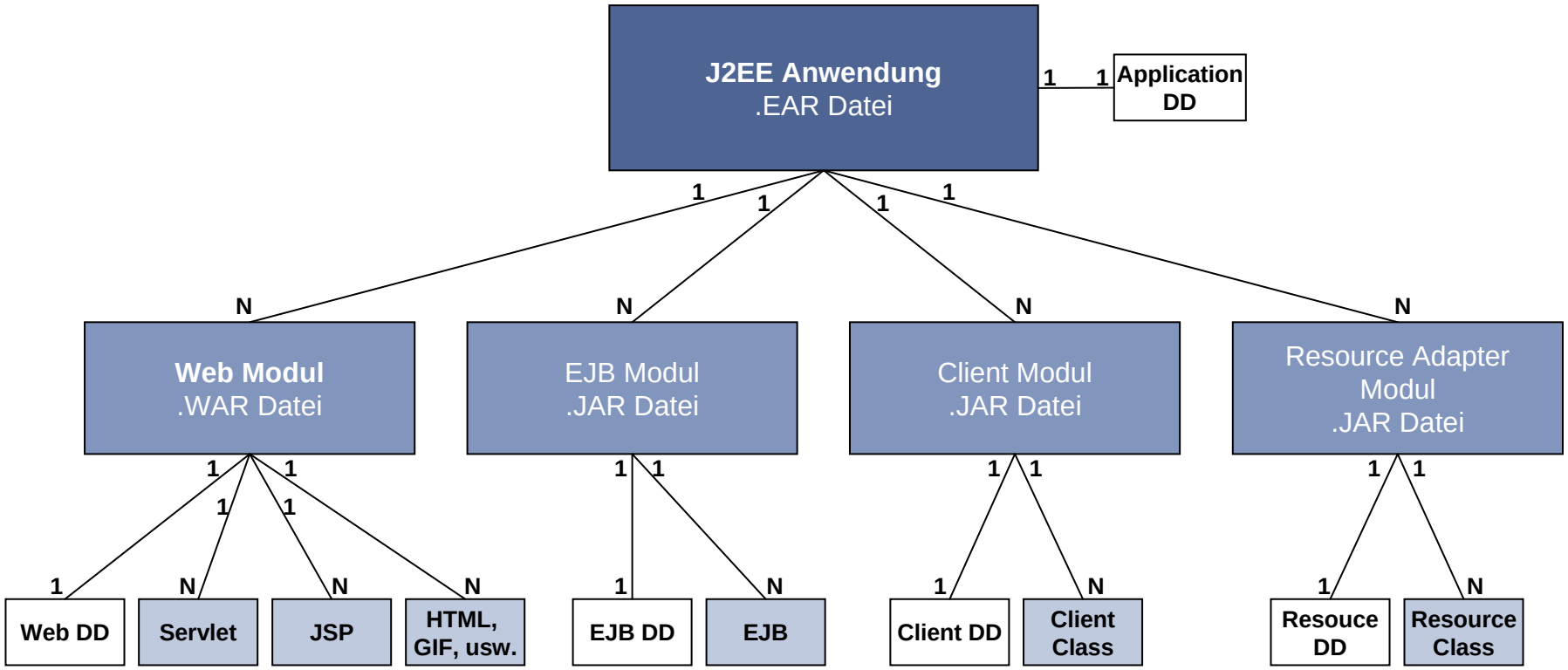
Quelle: in Anlehnung an [Sun04c]

Paketierung und Installation

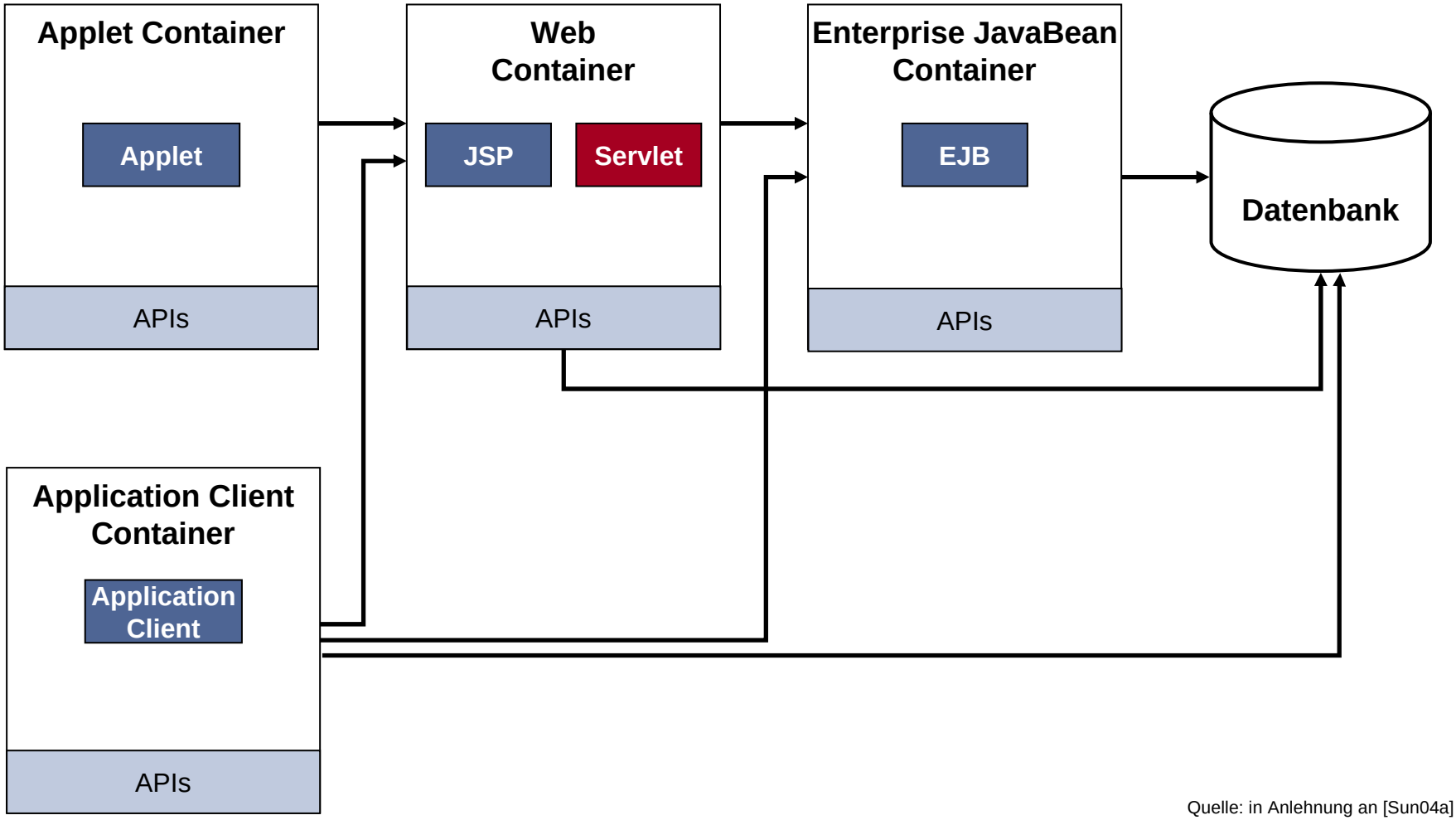


- Paketierung der einzelnen Web-Komponenten zur Installation in einem Web-Container
- Datenformat: Web ARchive Format (WAR)
- Vorteile:
  - Installation eines WAR vs. Installation einzelner Web-Komponenten
  - Konsistenz und Integrität
  - Paketierung und Versionierung

Module einer J2EE-Anwendung

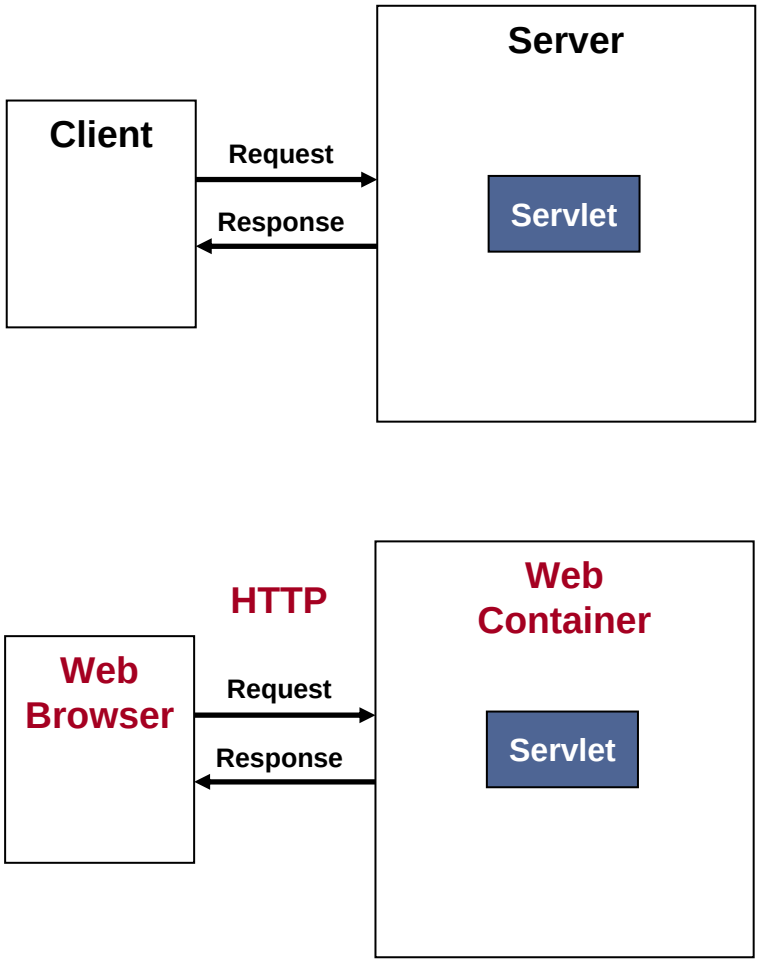


Positionierung in der Softwarearchitektur



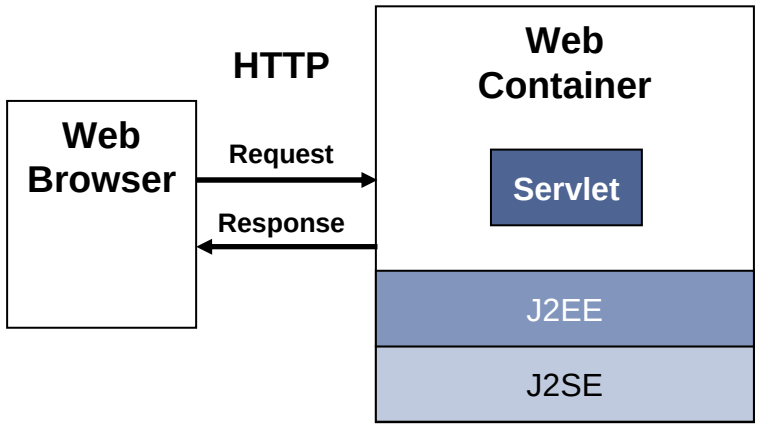
Quelle: in Anlehnung an [Sun04a]

Begriffsdefinition: Servlet



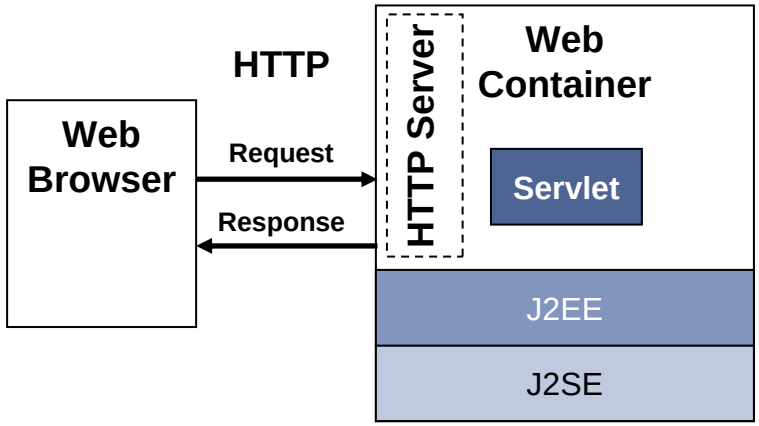
- Java-basierte Klasse, welche die Funktionalität eines Web Servers erweitert
- Erweiterung: Bereitstellung von dynamischen Inhalten auf der Basis eines Request-and-Response-Programmiermodells
- Unabhängig vom Kommunikationsprotokoll
- Abgeleitete Klassen zur Unterstützung des HTTP

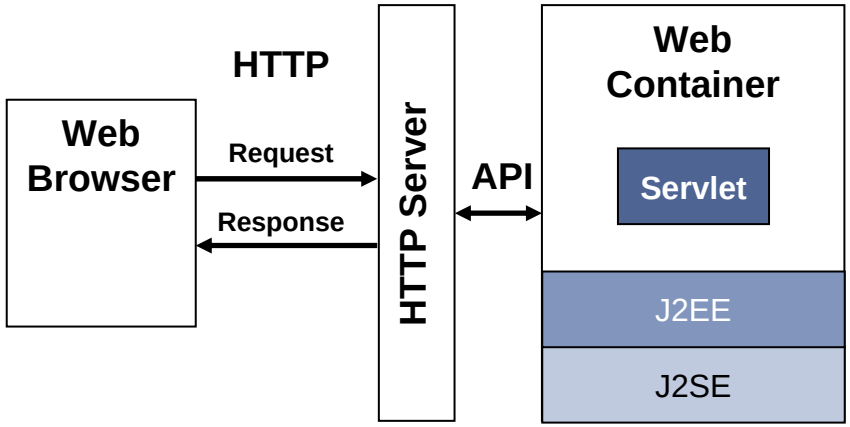
Begriffsdefinition: Servlet Engine / Servlet Container



- Laufzeitumgebung für Servlets
- Steuerung und Verwaltung der Servlets
  - dynamisches Laden und Instanziieren von Servlets
  - Management des Lebenszyklus von Servlets
- Implementation und Kapselung des HTTP
  - Bereitstellung von HTTP-Nachrichten durch Java-Objekte für Servlets

Verteilungsaspekte von Web Servern



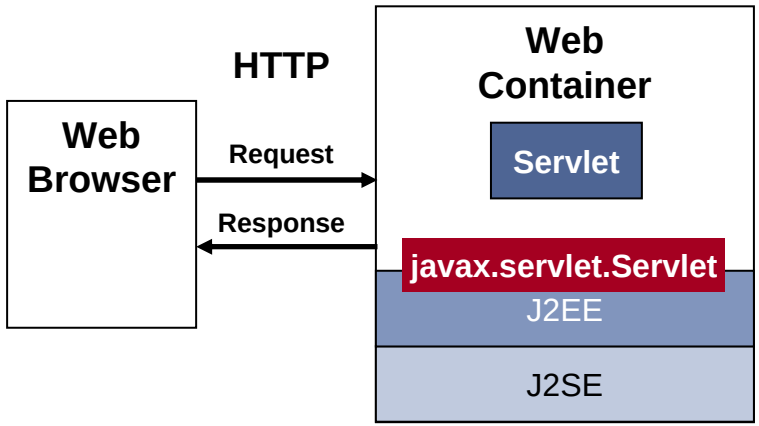


- Der HTTP-Server kann im Prozess des Web Containers laufen oder als eigenständiger Prozess ausgeführt werden
  - Prozess vs. Thread
  - Bereitstellung von statischen vs. dynamischen Inhalten
- Eigenständiger HTTP-Server wird über ein API integriert

Interface Servlet

- Schnittstelle, die von allen Servlets implementiert wird
  - javax.servlet.Servlet
- Methoden zum Management des Lebenszyklus
  - void init(ServletConfig config)
  - void destroy()
- Methode zur Verarbeitung von Requests
  - void service(ServletRequest req, ServletResponse res)
- Methoden zur Ermittlung von Informationen über das Servlet
  - ServletConfig getServletConfig()
  - String getServletInfo()

Steuerung und Verwaltung der Servlets



- Aufruf der Methoden der Schnittstelle Servlet durch den Servlet Container
- Schnittstelle Servlet ist ein Vertrag zwischen dem Servlet Container und dem Servlet
  - dynamisches Laden und Instanziieren von Servlets
  - Management des Lebenszyklus von Servlets

Klasse HTTPServlet (1/2)

- Abstrakte Klasse zur Verarbeitung von Requests auf der Basis des HTTP
- Abgeleitet von der abstrakten Klasse GenericServlet
  - GenericServlet wiederum implementiert die Schnittstelle Servlet
- Service-Methode wird überschrieben (Overriding)
  - void service(HttpServletRequest req, HttpServletResponse resp)

Klasse HTTPServlet (2/2)

- Aufruf einer speziellen Methode entsprechend des Typs des HTTP-Requests durch die Service-Methode
  - doGet für HTTP GET Requests
  - doPost für HTTP POST Requests
  - doPut für HTTP PUT Requests
  - doDelete für HTTP DELETE Requests
  - doHead für HTTP HEAD Requests
  - doOptions für HTTP OPTIONS Requests
  - doTrace für HTTP TRACE Requests
- doGet- und doPost-Methode werden typischerweise von einem Anwendungsentwickler überschrieben

Lebenszyklus eines Servlets

- Management des Lebenszyklus erfolgt durch den Servlet Container
- Wenn noch keine Instanz des Servlets existiert
  - Laden der Servlet-Klasse
  - Erstellung einer Instanz der Servlet-Klasse
  - Aufruf der init-Methode des Servlets
- Wenn ein HTTP-Request eintrifft
  - Aufruf der service-Methode mit den entsprechenden Request- und Response-Objekten

Mapping von Requests zu Servlets

- Konfiguration von Web-Komponenten durch Deployment Descriptors
- Mappings von Requests zu Servlets erfolgen auf der Basis von URL-Pfaden und Patterns
- Beispiel:
  - Mappings im Deployment Descriptor
    - /foo/bar/* entspricht servlet1
    - /baz/* entspricht servlet2
    - /catalog entspricht servlet3
  - Servlet-Aufrufe
    - Pfad: /foo/bar/index.html - Aufruf: servlet1
    - Pfad: /baz - Aufruf: servlet2
    - Pfad: /baz/index.html - Aufruf: servlet2
    - Pfad: /catalog - Aufruf: servlet3

HTTP-Request

- Kapselung des HTTP-basierten Client-Requests in einem Java-Objekt
  - javax.servlet.http.HttpServletRequest
  - Instanziierung des Objekts durch den Servlet Container
- Zugriff auf die Parameter
  - HTTP GET Request: Parameter werden an URL gehängt (Query-String)
  - HTTP POST Request: Parameter werden im Body transportiert
- Zugriff auf die Header
  - Bsp. User-Agent, Server, Last-Modified
- Zugriff auf Cookies
  - Typischerweise nur Namen und Wert des Cookies

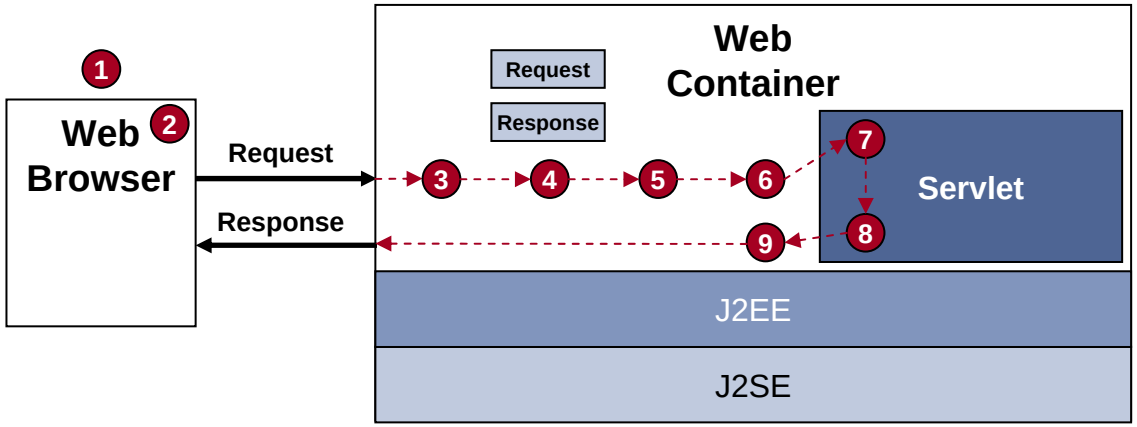
HTTP-Response

- Kapselung der HTTP-basierten Server-Response in einem Java-Objekt
  - javax.servlet.http.HttpServletResponse
- Instanziierung des Objekts durch den Servlet Container
- Zugriff auf den Response-Buffer des Servlet Containers
  - Bsp. getBufferSize, setBufferSize, resetBuffer
- Zugriff auf die Header
  - Bsp. Server, Last-Modified, Content-Type
- Zugriff auf den Body
  - Bsp. HTML, XML, MIME
  - Content-Type Header muss entsprechend gefüllt sein!

Beispiel: Überschreiben der doGet-Methode

```
public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException
{
      System.out.println("VTBDV1 - Methode doGet() wird aufgerufen ...");

      // Setzen des HTTP-Headers 'Content Type'
      resp.setContentType("text/html");

      // Hole eine Referenz zum Character Stream
      PrintWriter out = resp.getWriter();

      // Schreibe die HTML-Tags in den Character Stream
      out.println("<HEAD><TITLE>Verteilte betriebliche DV-Systeme 1</TITLE></HEAD>");

      out.println("<BODY>");

      out.println("<H1>Teil 1: Verteilte Anwendungen auf der Basis von J2EE</H1>");

      out.println("<H2>Kapitel Servlets</H2>");

      // Ausgabe zur Indentifikation der Seite
      out.println("<P>Das ist die Ausgabe eines Servlets!</P>");

      out.println("<BODY>");

      // Schließe den Character Streams
      out.close();
}
```

Sequenz eines Servlet-Aufrufs

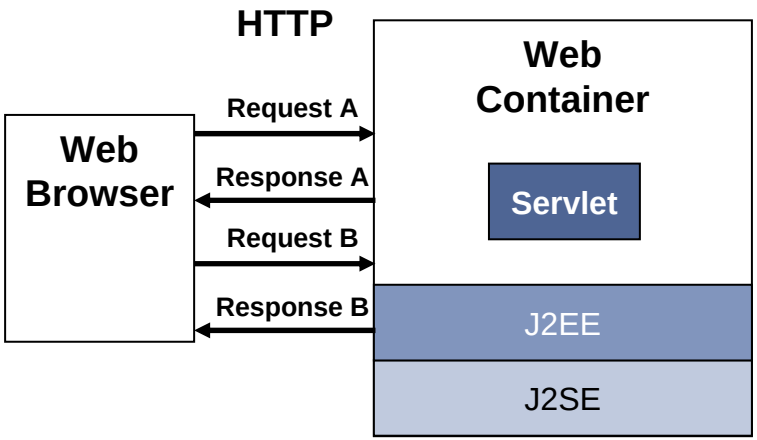


1. **Benutzer: Eingabe der URL**
2. **Web Browser: HTTP Request wird an Web Server/Web Container gesendet**
3. **Web Container: Überprüfung der URL und Mapping auf das Servlet**
4. **Web Container: Kapselung des Requests und Responses in jeweils einem Objekt**
5. **Web Container: Wenn noch keine Instanz des Servlets existiert, Laden und Erstellung einer Instanz der Servlet-Klasse, Aufruf der init-Methode des Servlets**
6. **Web Container: Aufruf der service-Methode des Servlets**
7. **Servlet: Aufruf entsprechenden do-Methode**
8. **Servlet: Schreiben der HTML-Tags in ein Buffer des Reponse-Objekts**
9. **Web Container: HTTP Response wird auf der Basis des Response-Objektes erstellt und versendet**

Filtering

- Transformation eines HTTP-Requests und/oder -Responses
- Manipulation und Anpassung eines Requests und/oder Responses bevor oder nachdem ein Servlet aufgerufen wird
- Einsatzszenarios
  - Logging und Auditing
  - Kompression
  - Verschlüsselung
  - Trigger
- Filter implementieren die Schnittstelle javax.servlet.Filter
  - doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
- Verkettung von mehreren Filtern
  - chain.doFilter()

Sessions



- HTTP ist ein zustandloses Protokoll
- Anforderung: mehrere Requests müssen einem Client zugeordnet und in einer Session gruppiert werden
- Mehrere Mechanismen zum Aufbau und der Verfolgung von Session verfügbar
  - Session Tracking Mechanisms

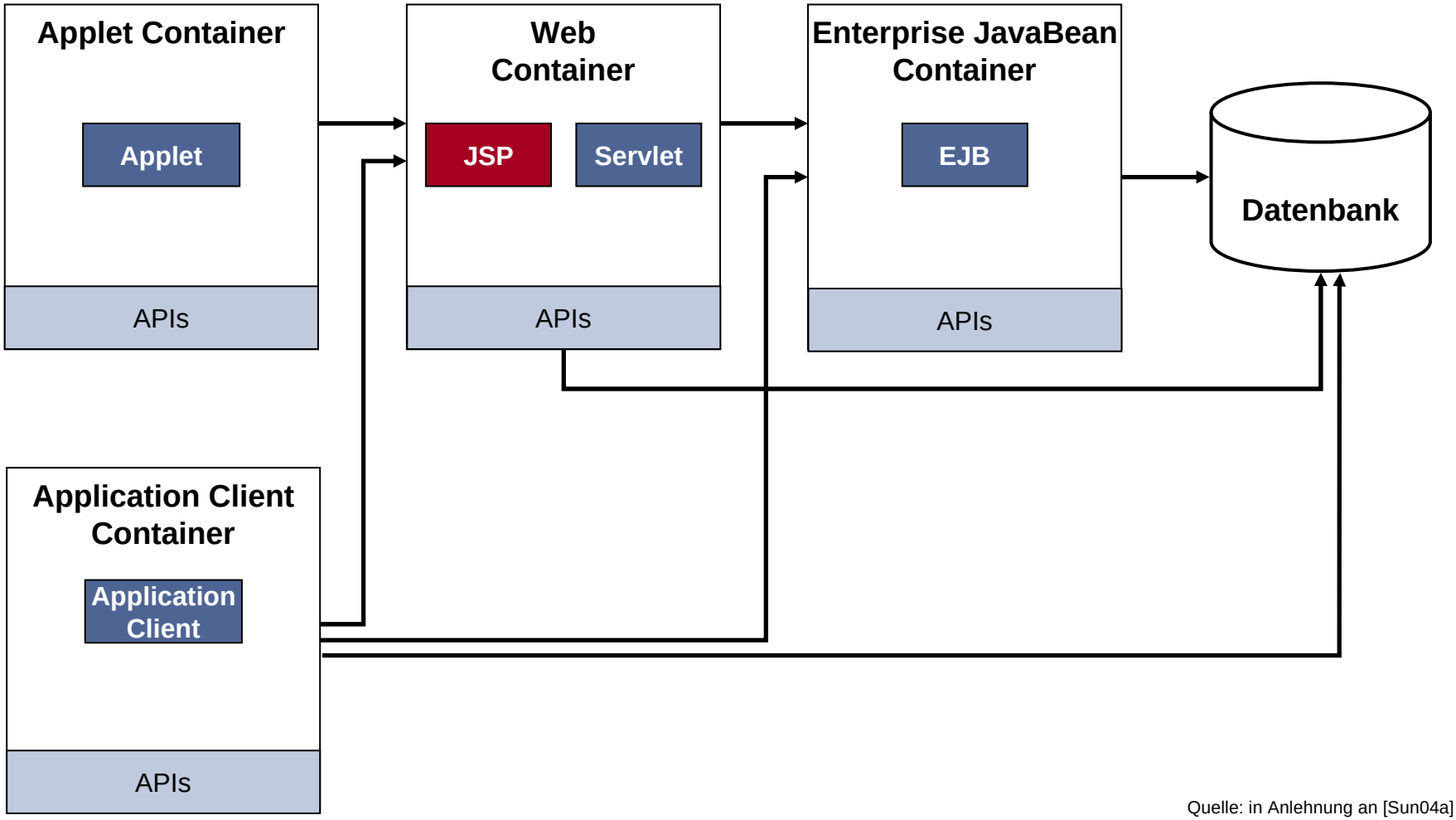
Session Tracking Mechanismen

- Cookies
  - Servlet Container sendet einen Cookie an den Client und der Client sendet bei jedem nachfolgenden Request den Cookie mit
- URL Rewriting
  - Erweiterung des URL-Pfads um eine Session-ID, welche vom Servlet Container initial erstellt und bei nachfolgenden Requests ausgewertet wird
  - Bsp. http://www.shop.com/index.html;jsessionid=1234
- Secure Sockets Layer (SSL) Sessions
  - Aufbau und Verfolgung einer Session innerhalb der SSL
  - Servlet Container nutzt diese Session zur Verfolgung der HTTP-Requests

Eigenschaften von Sessions

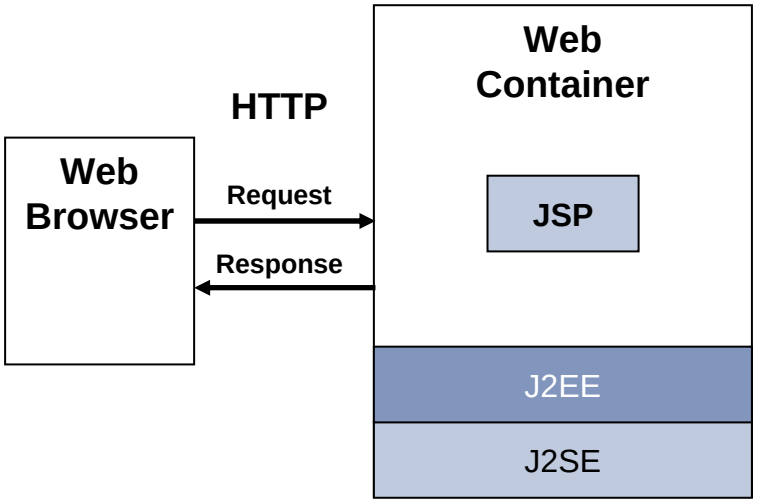
- Attribute
  - Zugriff von Servlets auf Attribute im Session-Kontext
  - Lesen und Schreiben von Attributen
- Timeouts
  - HTTP unterstützt keine Nachricht zum Beenden einer Session
  - Verwendung eines Timeouts zur Identifikation einer beendeten Session
  - Timeout Period: Konfigurationsparameter des Servlet Containers

Positionierung in der Softwarearchitektur



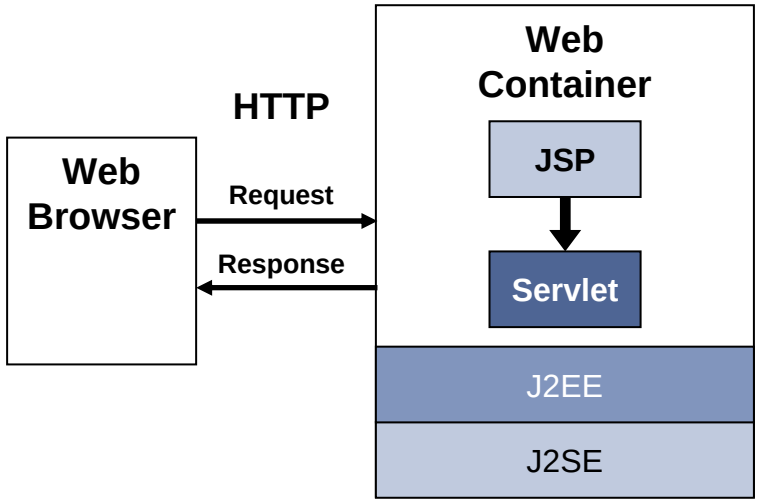
Quelle: in Anlehnung an [Sun04a]

Begriffsdefinition: JavaServer Page (JSP)



- Text-basiertes Dokument, welches beschreibt wie dynamische Inhalte für einen Client bereitgestellt werden
- Enthält zwei Arten von Inhalten
  - statische Inhalte wie z.B. HTML und XML
  - dynamische Inhalte, welche mit sog. JSP Elementen erstellt werden
- Empfohlene Dateiendung
  - *.jsp

JavaServer Pages vs. Servlets



- JavaServer Pages werden (spätestens) beim ersten Aufruf in Servlets übersetzt
- Alle folgende Requests führen lediglich zu einem Aufruf des Servlets
- JavaServer Pages Spezifikation basiert auf der Servlet Spezifikation
  - Steuerung und Verwaltung der Servlets
  - HTTP-Request- und -Response-Objekte
  - usw.

Beispiel: JavaServer Page

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>

<TITLE>Verteilte betriebliche DV-Systeme 1</TITLE>
</HEAD>
<BODY>
<H1>Teil 1: Verteilte Anwendungen auf der Basis von J2EE</H1>
<H2>Kapitel JavaServer Pages</H2>
<H3>Dynamische HTML-Seite auf der Basis einer JSP</H3>
<P>Das ist eine dynamische Seite!</P>

<% for (int i=0; i < 20; i++ ) {
    out.println("<P>Das ist die " + i + ". Ausgabe einer JavaServer Page!</P>");
}
%>

</BODY>
</HTML>
```

Begriffsdefinition: JSP Element

- Steuerung der Übersetzung in ein Servlet
- Erstellung und Zugriff auf Objekte
- Definition von Methoden und des Kontrollflusses
- Zwei Syntaxen möglich
  - Standard
    - Syntax an XML angelehnt, aber nicht „well-formed“
  - XML
    - Zugriff und Validierung mit XML-Parser möglich

JSP Elementtypen

- Directives
  - Anweisungen, welche bei der Übersetzung in ein Servlet durch die JSP Engine verarbeitet werden
- Scripting Elements
  - Declarations
    - Deklaration von Variablen und Methoden innerhalb einer JSP-Seite
  - Scriplets
    - Code-Fragmente, welche während der Verarbeitung des Requests ausgeführt werden
  - Expressions
    - Ausdrücke, welche in eine Zeichenkette konvertiert und (über den sog. JSP Writer) ausgegeben werden
- Actions
  - Erstellen, benutzen und/oder verändern Objekte

Directives

- Erzeugen keine Ausgaben
- Standard Syntax
  - <%@ page page_directive_attr_list %>
- XML-basierte Syntax
  - <jsp:directive.page page_directive_attr_list />
- Bsp. Standard Syntax
  - <%@ page contentType=”text/html” %>
  - Konfiguration des MIME-Types in einer Response-Nachricht

Scripting Elements - Declarations

- Deklaration von Variablen und Methoden, welche innerhalb der JSP-Seite verwendet werden
- Erzeugen keine Ausgaben
- Standard Syntax
  - <%! declaration(s) %>
- XML-basierte Syntax
  - <jsp:declaration> declaration </jsp:declaration>
- Bsp. Standard Syntax
  - <%! int i = 0; %>
  - Deklaration und Initialisierung der Variablen i

Scripting Elements - Scriplets

- Code-Fragmente, welche während der Verarbeitung des Requests ausgeführt werden
- Scriplets können zuvor deklarierte Variablen modifizieren
- Scriplets können Ausgaben generieren
- Standard Syntax
  - <% scriptlet %>
- XML-basierte Syntax
  - <jsp:scriptlet> code fragment </jsp:scriptlet>
- Bsp. Standard Syntax
  - <% i++; %>
  - Inkrement der Variablen i

Scripting Elements - Expressions

- Ausdrücke, welche in eine Zeichenkette konvertiert und (über den sog. JSP Writer) ausgegeben werden
- Standard Syntax
  - <%= expression %>
- XML-basierte Syntax
  - <jsp:expression> expression </jsp:expression>
- Bsp. Standard Syntax
  - <%= (new java.util.Date()).toString() %>
  - Ausgabe des Datums als Zeichenkette

Actions

- Erstellen, benutzen und verändern Objekte
- Standard Actions
  - in der Spezifikation vordefiniert
- Custom Actions
  - benutzerdefinierte Actions, welche über einen Erweiterungsmechanismus (sog. Tag Libraries) integriert werden
- Nur eine XML-basierte Syntax
  - <tag attr1=”attribute value”... >body</tag>

Beispiel: Action <jsp:useBean>

- Zuweisung eines Objekts mit einer entsprechenden ID und innerhalb eines definierten Scopes
- Klasse des Objekts muss der JavaBean-Spezifikation entsprechen, d.h. eine JavaBean sein
- Flexible Semantik
  - Wenn das Objekt noch nicht unter der ID und im definierten Scope existiert, wird es vor der Zuweisung erstellt
- Syntax
  - <jsp:useBean id="name" scope="page|request|session|application" typeSpec />
- Beispiel
  - <jsp:useBean id=”customer” class=”com.myco.Customer” />

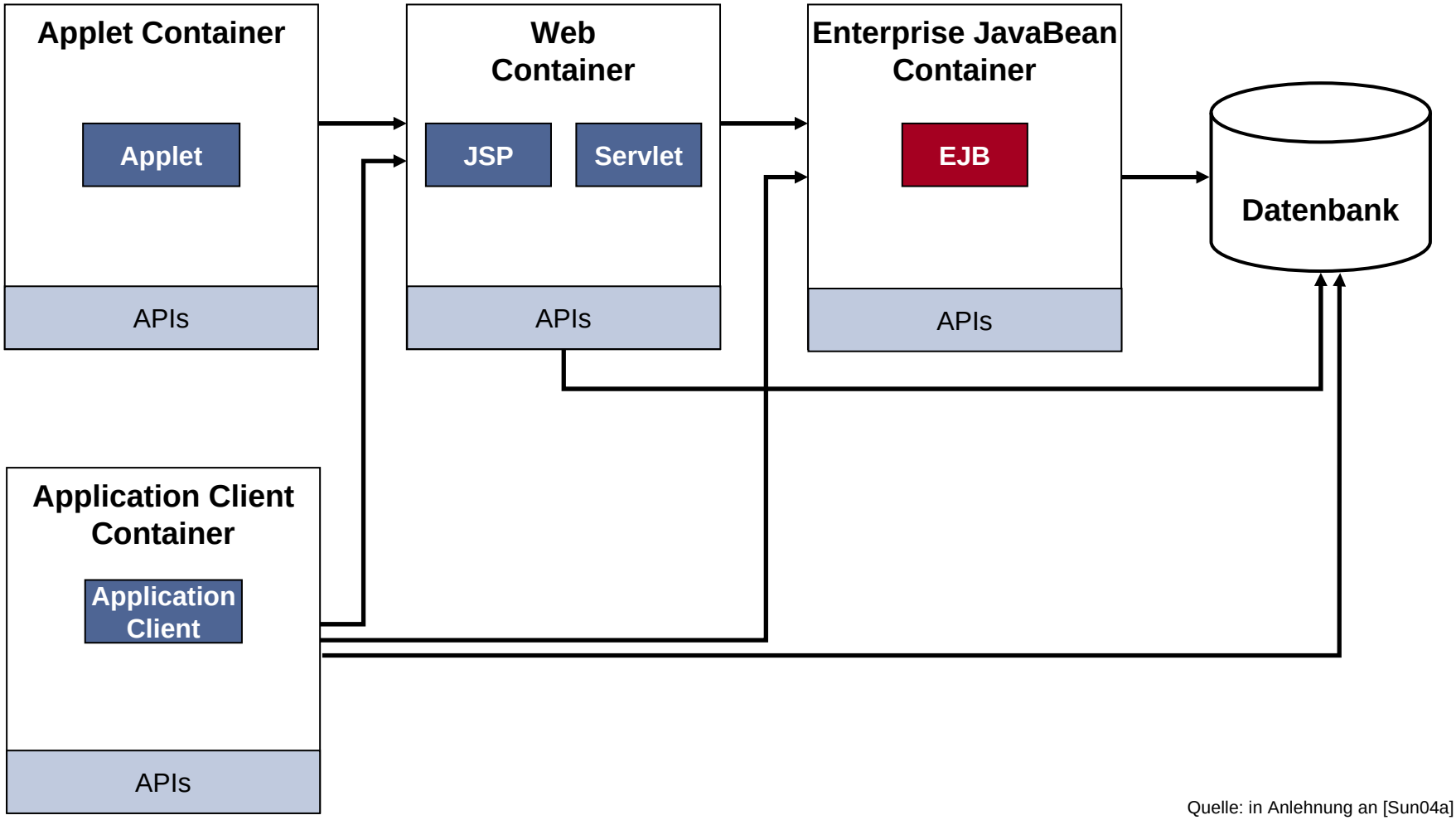
Beispiel: JavaServer Page mit JSP Elementen (1/2)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<jsp:useBean id="cust" class="com.softec.vtbdv1.beans.Customer"
    scope="session">
    <jsp:setProperty name="cust" property="surname" value="Müller"/>
    <jsp:setProperty name="cust" property="firstname" value="Karl-Heinz"/>
</jsp:useBean>

<HTML>
<HEAD>

<%-- Directives  --%>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
session="true"%>

<TITLE>Verteilte betriebliche DV-Systeme 1</TITLE>
</HEAD>
```

Beispiel: JavaServer Page mit JSP Elementen (2/2)

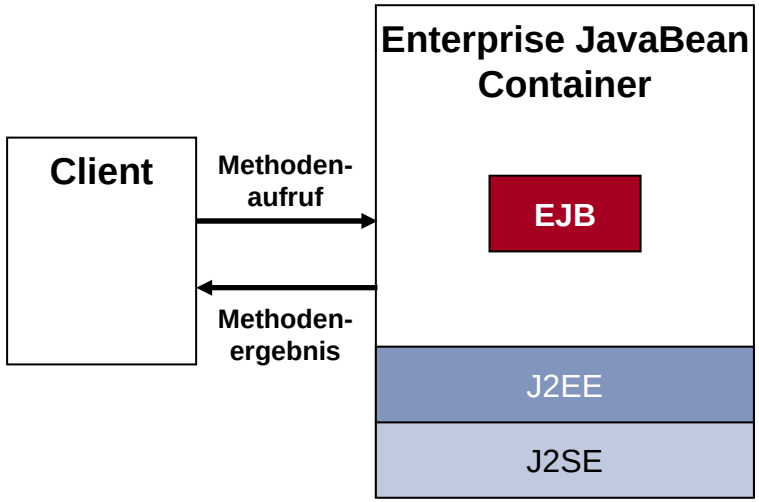
```
<BODY>
<H1>Teil 1: Verteilte Anwendungen auf der Basis von J2EE</H1>
<H2>Kapitel JavaServer Pages</H2>
<H3>JSP Elementtypen - Beispiele</H3>

<%-- Declarations --%>
<%! int i = 1; String str = new String("Zeichenkette"); %>

<%-- Scriplets --%>
<% for (; i < 5; i++) {
    out.println("Zeile Nummer: " + i + "<BR><BR>"); } %>
<% out.println("Der Wert der Variablen 'str' ist: " + str + "<BR><BR>"); %>

<%-- Expressions --%>
<%= (new java.util.Date()).toString() %>
<BR><BR>

<%= cust.getSurname() + ", " + cust.getFirstname() %>
</BODY>
</HTML>
```

Positionierung in der Softwarearchitektur



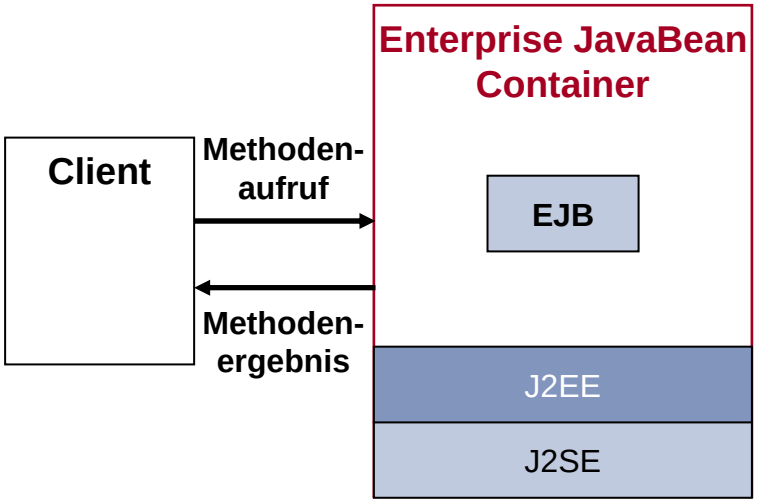
Quelle: in Anlehnung an [Sun04a]

Begriffsdefinition: Enterprise JavaBean



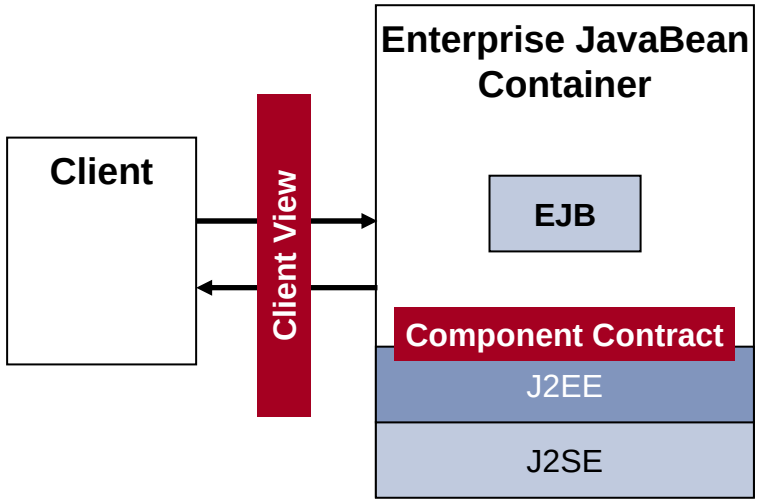
- Server-seitige Architektur zur Entwicklung von komponenten-basierten und verteilten Anwendungen
- Ausführung innerhalb eines Enterprise JavaBean (EJB) Containers
- Aufruf durch verschiedene potenzielle Clients wie z.B.
  - Applet Container
  - Application Client Container
  - Web Container
  - EJB Container

Begriffsdefinition: Enterprise JavaBean Container



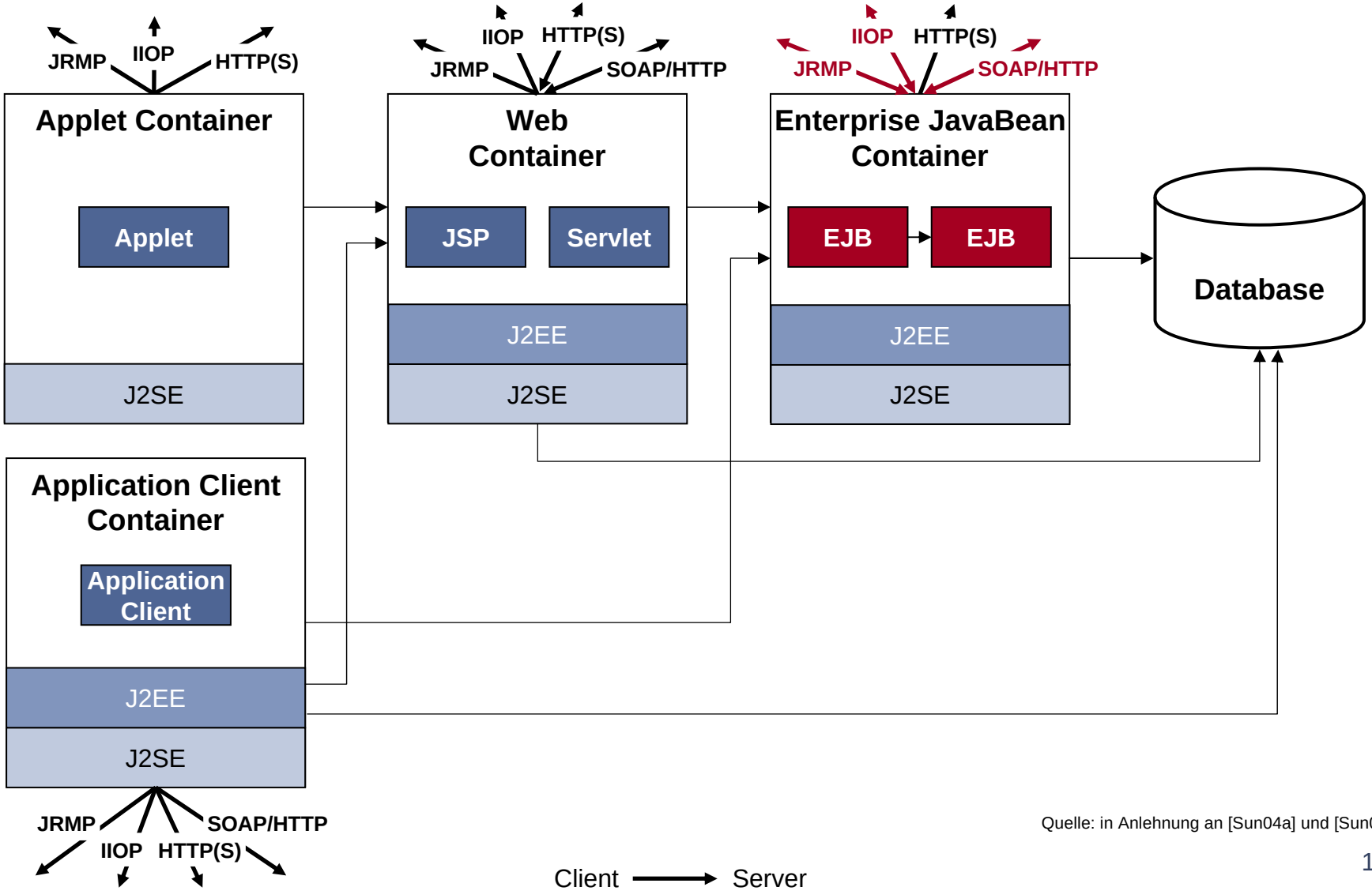
- Laufzeitumgebung für EJBs
- Steuerung und Verwaltung der EJBs
  - dynamisches Laden und Instanziieren von EJBs
  - Management des Lebenszyklus von EJBs
- Bereitstellung von Diensten wie z.B.
  - Kapselung der Kommunikation mit Clients
  - Transaktionen
  - Persistenz

Eigenschaften der Enterprise JavaBeans



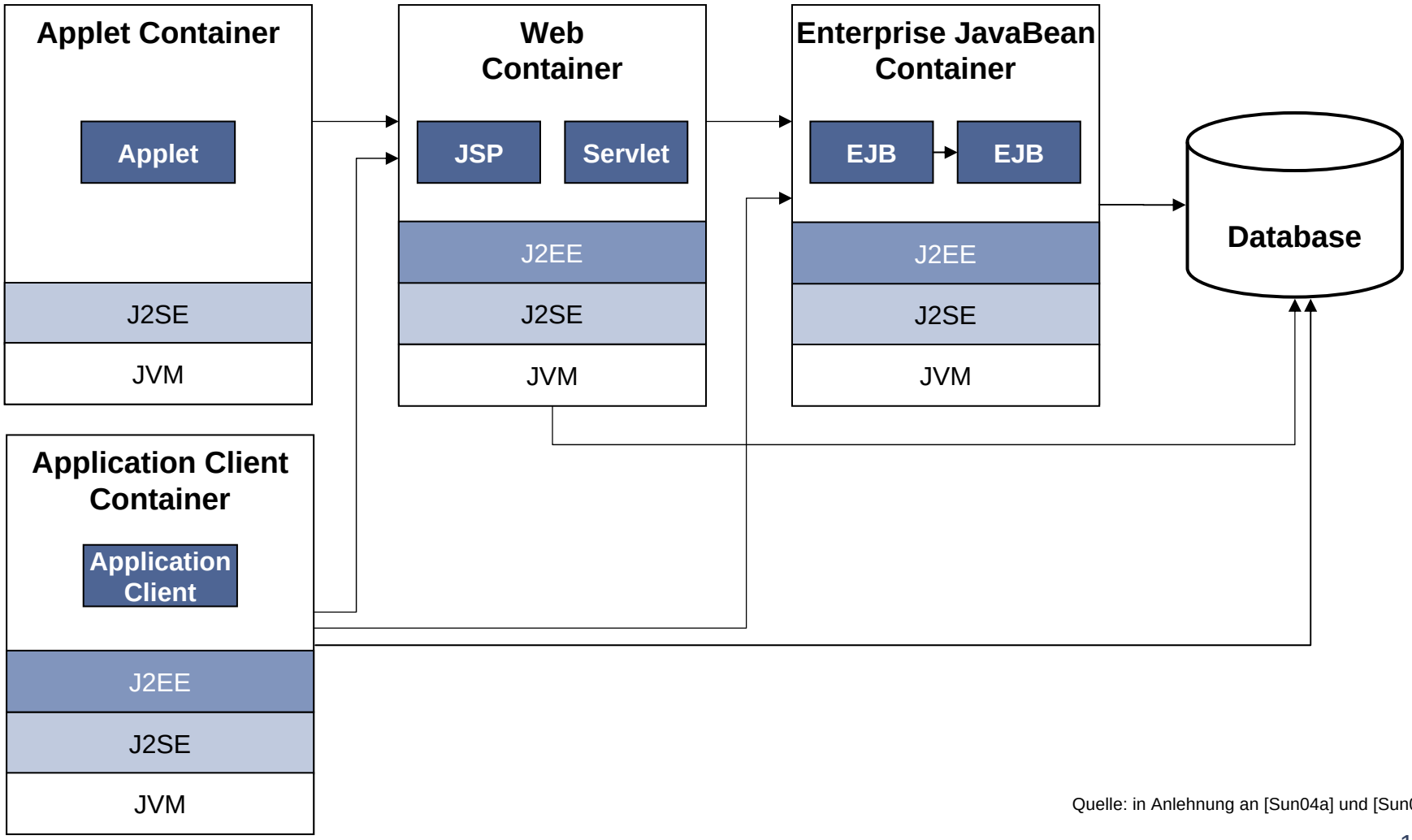
- EJBs enthalten i.d.R. die Geschäftslogik einer Anwendung
- Konfiguration zum Zeitpunkt der Installation im Container
- Zugriff der Clients wird durch den Container vermittelt
- Zwei wesentliche Schnittstellen
  - zwischen Client und Container
  - zwischen EJB und Container

Clients der EJBs



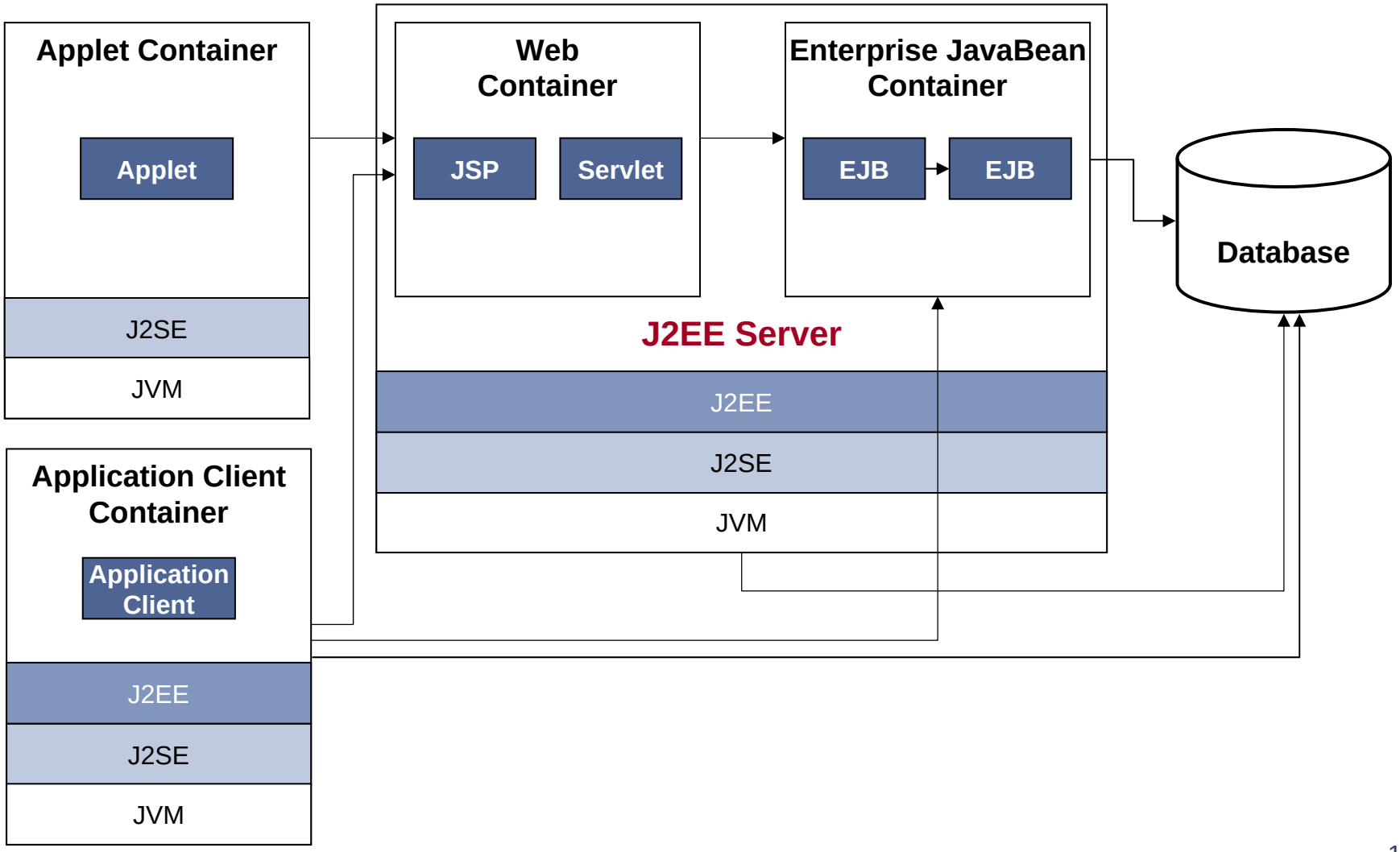
Quelle: in Anlehnung an [Sun04a] und [Sun04b]

Verteilung der Clients



Quelle: in Anlehnung an [Sun04a] und [Sun04b]

J2EE Server mit Web und EJB Container



Local vs. Remote Clients

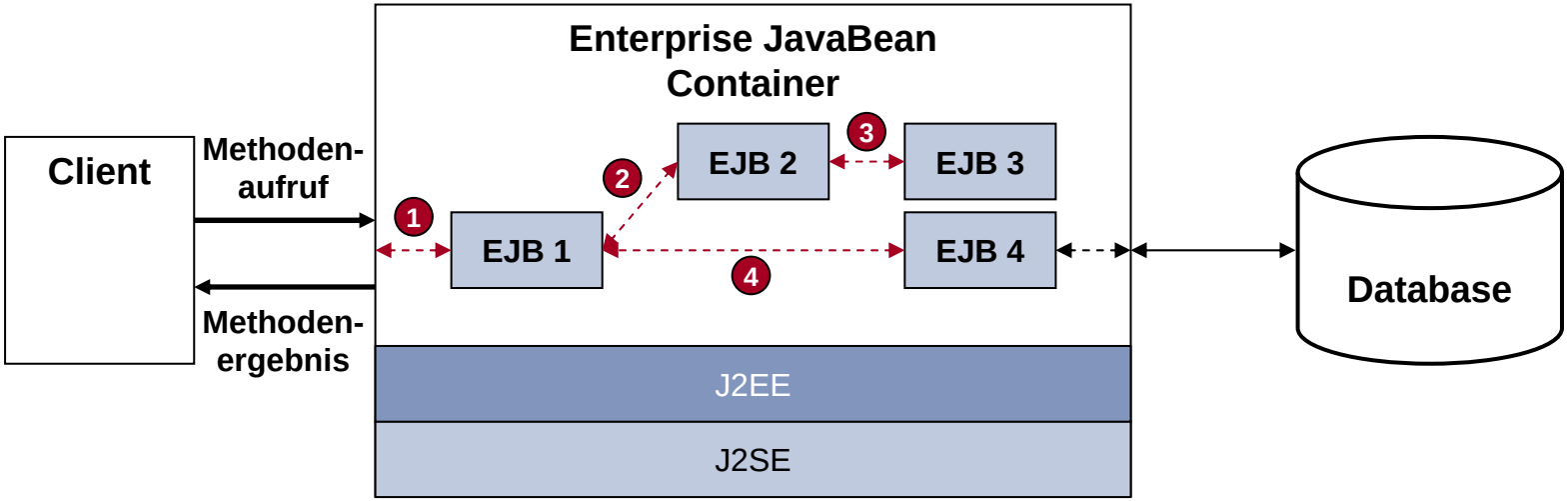
Kriterium	Local Client	Remote Client
Ortstransparenz	Nein	ja
Ort der EJB	in der identischen JVM	in einer anderen JVM (auf dem identischen oder einem anderen Rechner)
Typ der Schnittstelle	Java Interface	Java Remote Interface
Methodenparameter und -ergebnis	Pass by Reference	Pass by Value

Aspekte des Entwurfs

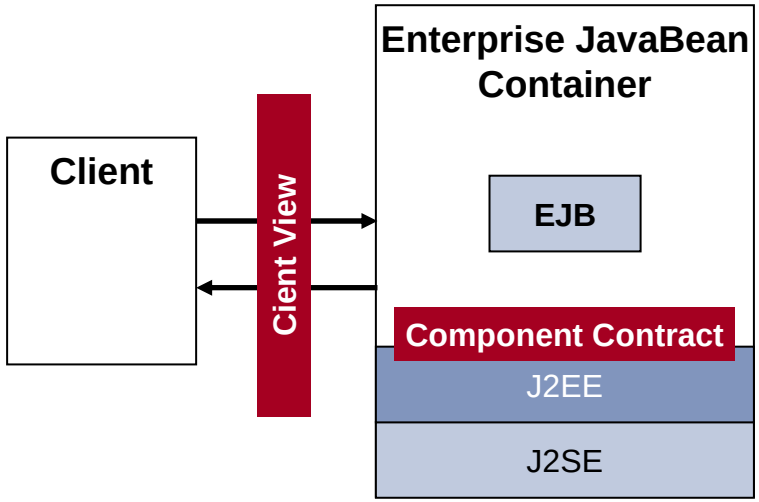
- Flexibilität der Verteilung
  - Entfernte Schnittstellen ermöglichen Ortstransparenz und unterstützen somit auch Local Clients
- Effizienz der Methodenaufrufe
  - Entfernte Methodenaufrufe sind „teuer“ (Marshalling, Latenzzeit, Kopieren von Methodenparametern und Ergebnissen, usw.)
- Isolation der Objekte
  - Keine Seiteneffekte durch Manipulation von gemeinsam genutzten Objekten (Entkopplung von Client und EJB)
- Umfang der Fehlerbehandlung
  - Netzwerkkommunikation als zusätzliche Fehlerquelle

Quelle: in Anlehnung an [Sun04b]

Granularität der EJBs



Schnittstellen zwischen Client und Container



- Local und Remote Client View umfassen
  - Home Interface
  - Component Interface
- Remote Client View enthält zusätzlich
  - Metadata Interface

Home Interface

- Definition von Methoden zum Erstellen, Löschen und Auffinden von EJBs eines Typs
- Definition von Methoden, welche nicht an eine spezielle EJB-Instanz gebunden sind
- Spezifikation der Schnittstelle durch den EJB-Entwickler
  - Container generiert eine Klasse / Klassen, welche die Schnittstelle implementiert / implementieren
- Typ des Client Views bestimmt von welcher Schnittstelle das Home Interface abgeleitet werden muss
  - Remote Client View: javax.ejb.EJBHome
  - Local Client View: javax.ejb.EJBLocalHome

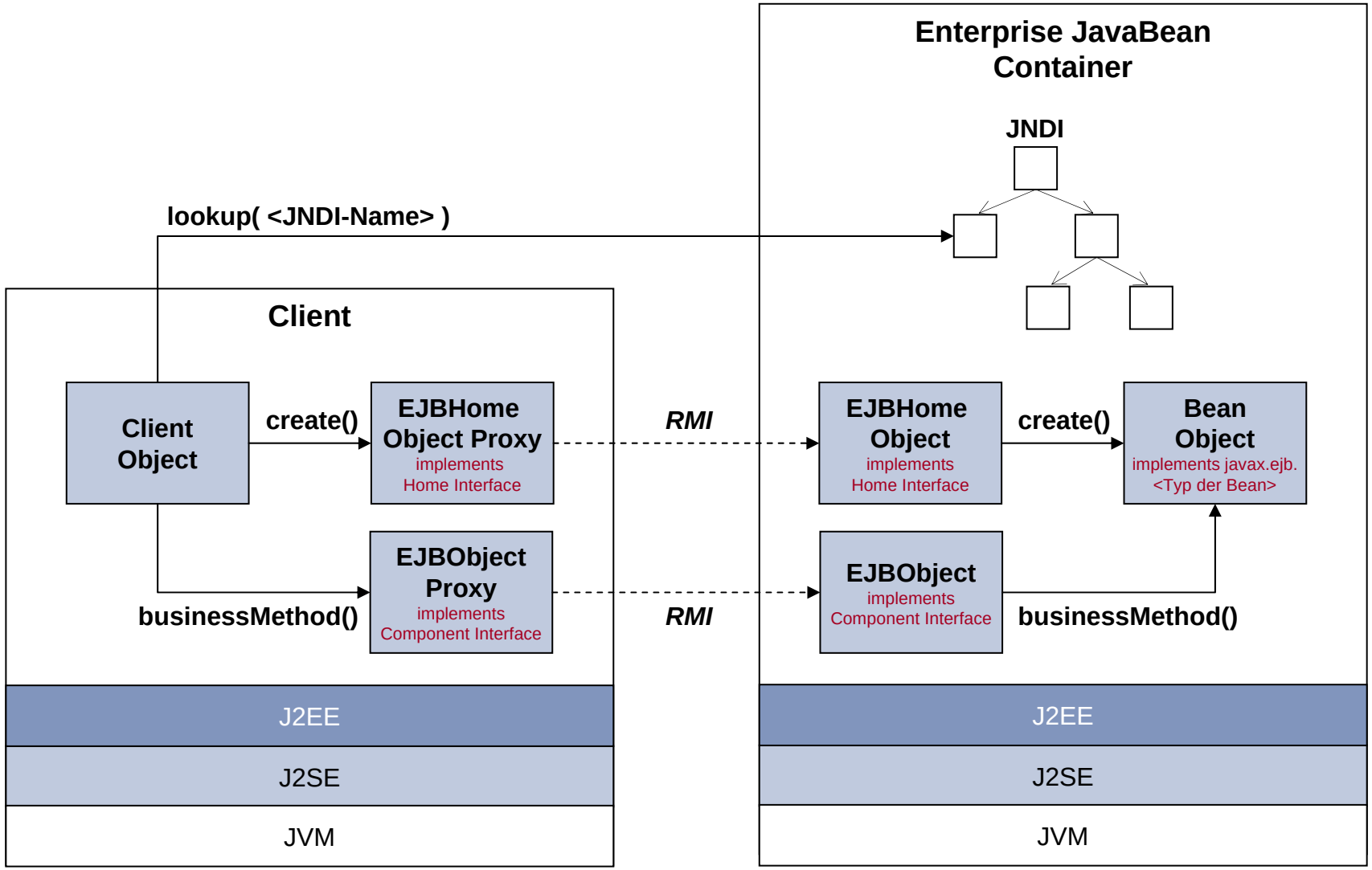
Component Interface

- Definition von Methoden, welche die Geschäftslogik implementieren und vom Client aufgerufen werden können
- Spezifikation der Schnittstelle durch den EJB-Entwickler
  - Container generiert eine Klasse / Klassen, welche die Schnittstelle implementiert / implementieren
- Typ des Client Views bestimmt von welcher Schnittstelle das Component Interface abgeleitet werden muss
  - Remote Client View: javax.ejb.EJBObject
  - Local Client View: javax.ejb.EJBLocalObject

Metadata Interface

- Definition von Methoden zur Abfrage von Metadaten einer EJB
  - Metadaten bezüglich des Home und Component Interface
- Nutzung der Metadaten durch
  - Werkzeuge, die bereits im Container installierte EJBs analysieren und nutzen
  - Clients, die eine Scripting Language verwenden
  - Interface: javax.ejb.EJBMetaData

Nutzung einer entfernten EJB (1/2)



Nutzung einer entfernten EJB (2/2)

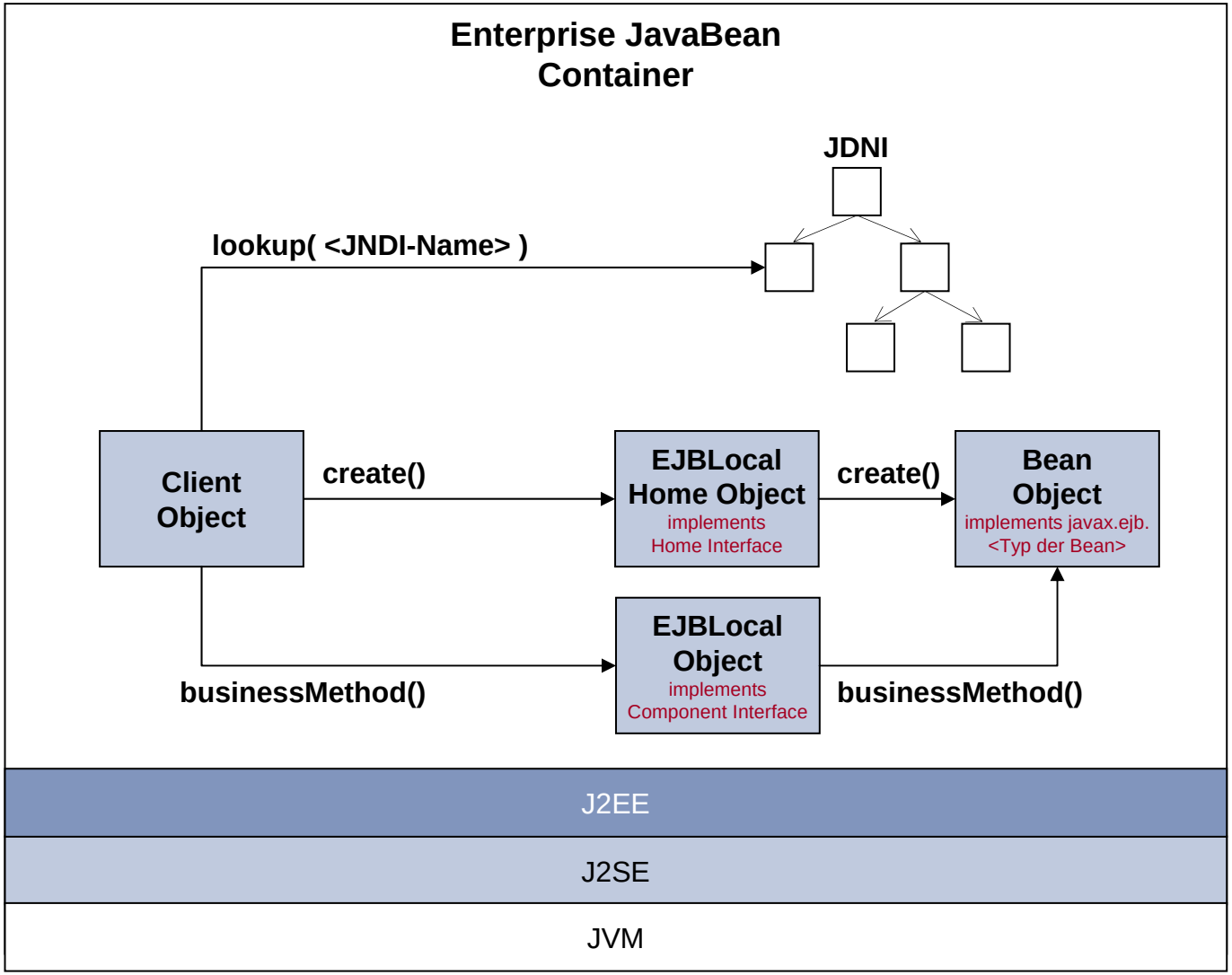
```
// Initialisierung des Kontextes für die
// folgende JNDI-Operation
Context initialContext = new
InitialContext();

// Ermittlung des Objektes
Object object = initialContext.lookup(
"ejb/com/softec/vtbdv1/CustomerFacadeHome");

// Type Cast des Objektes auf Remote Home
// Interface
CustomerFacadeHome facadeHome =
(CustomerFacadeHome)javax.rmi.PortableRemoteO
bject.narrow( object,
CustomerFacadeHome.class );
```

- Ermittlung des Objektes, welches das Remote Home Interface implementiert
  - JNDI wird als Objektspeicher verwendet
  - Type Cast erfolgt über statische Methode narrow(), um CORBA-Kompatibilität zu erhalten

```
// Erstellung der EJB-Instanz
CustomerFacade facade = facadeHome.create();

// Aufruf der Methode (Geschäftslogik)
int i = facade.getNumOfOpenOrders( customerID
);
```

- Erstellung und Nutzung der EJB-Instanz

Nutzung einer lokalen EJB (1/2)

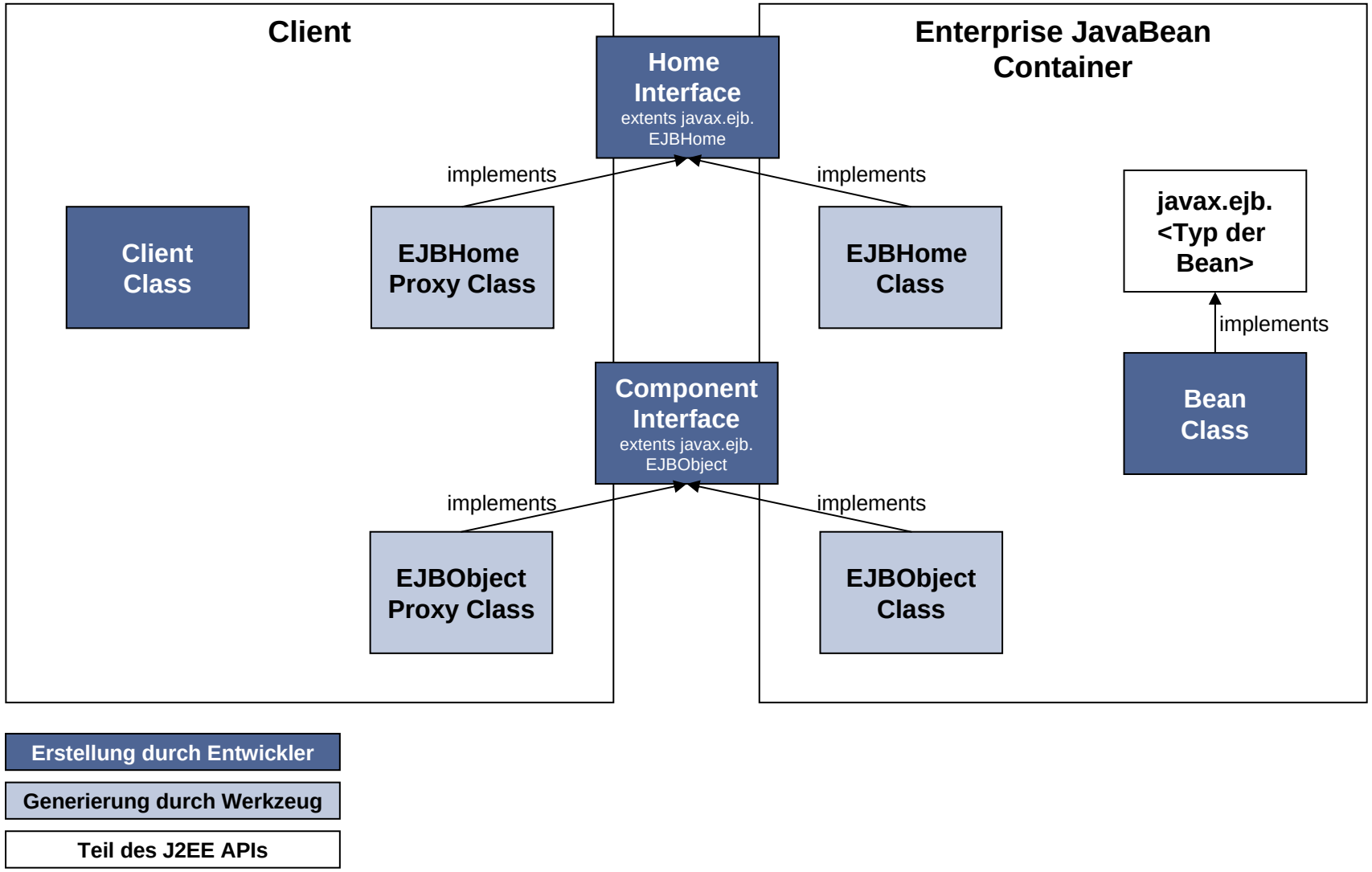


Nutzung einer lokalen EJB (2/2)

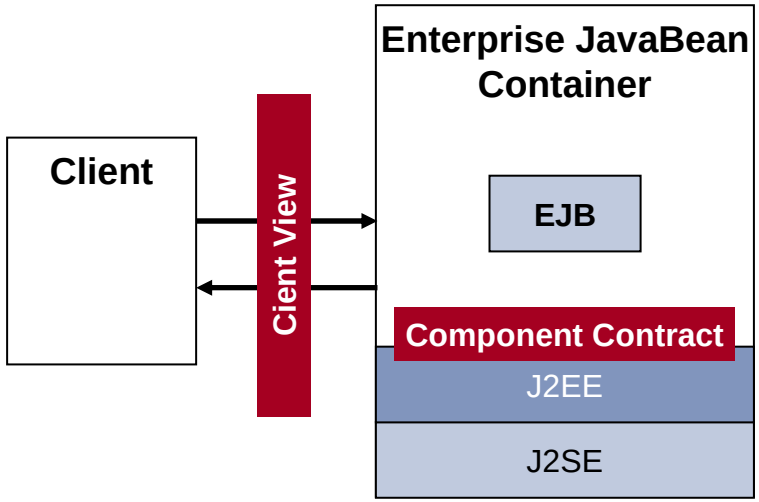
```
// Initialisierung des Kontextes für die
// folgende JNDI-Operation
Context initialContext = new
InitialContext();

// Ermittlung des Objektes und Java Type Cast
OrderListLocalHome orderListLocalHome =
(OrderListLocalHome)initialContext.lookup(
"local:ejb/com/softec/vtbdv1/
OrderListLocalHome");



// Erstellung der EJB-Instanz
OrderListLocal orderList =
orderListLocalHome.create();

// Aufruf der Methode (Geschäftslogik)
int i = orderList.getNumOfOrders( customerID
, OPEN_ORDERS );
```

- Ermittlung des Objektes, welches das Local Home Interface des Objektes implementiert
  - JNDI wird als Objektspeicher verwendet
  - Type Cast auf der Basis des standard Java-Mechanismus
- Erstellung und Nutzung der EJB-Instanz

Erstellung vs. Generierung von Klassen und Interfaces



Schnittstellen zwischen EJB und Container

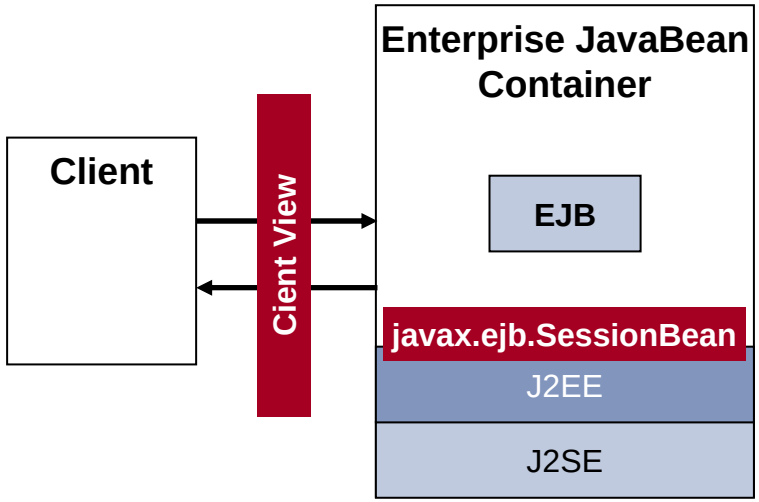


- Vertrag zwischen einer EJB und dem Container, welcher aus zwei Schnittstellen besteht
  - Methoden, welche die Geschäftslogik implementieren (vgl. Component Interface)
  - Callback-Methoden, welche vom Typ der EJB abhängig sind
- Implementation der Schnittstellen erfolgt durch die sog. Bean Class

Typen von Enterprise JavaBeans - Überblick

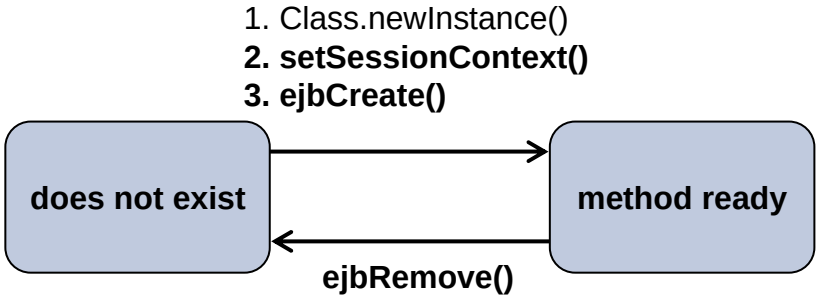
- Session Beans
  - implementieren Funktionen, welche i.d.R. von einem Client innerhalb eines Geschäftsprozesses aufgerufen werden
  - kurzlebig
- Entity Beans
  - objekt-orientierte Sicht auf Daten in einer Datenbank
  - langlebig
- Message-driven Beans
  - implementieren Funktionen, welche asynchron beim Eintreffen einer Client Message ausgeführt werden
  - kurzlebig

Eigenschaften von Session Beans



- Implementation von Funktionen, die von einem Client aufgerufen werden
- Funktionen können im Kontext einer Transaktion ausgeführt werden
  - EJB Container verwaltet und steuert die Transaktionen (Dienst)
- Zwei Subtypen von Session Beans
  - stateless
  - stateful

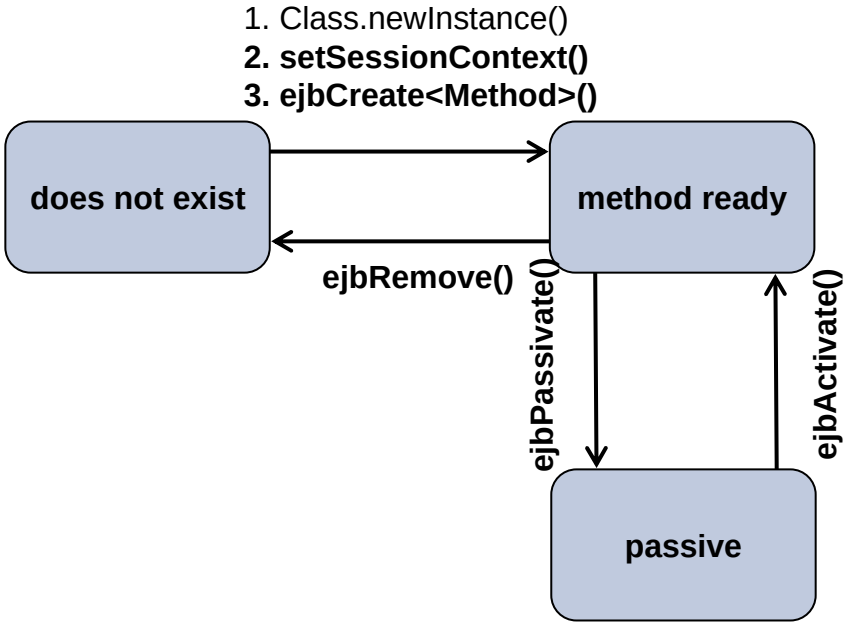
Lebenszyklus von Stateless Session Beans



- Verwaltung des Lebenszyklus erfolgt durch den Container
  - Container ruft die Callback-Methoden der Schnittstelle javax.ejb.SessionBean auf
- Keine Zuweisung zu einem bestimmten Client
  - Wiederverwendung von Instanzen ist möglich
- Keine Speicherung von Zustandsinformationen
  - Effizienter Speicherverbrauch

Quelle: in Anlehnung an [Sun04b]

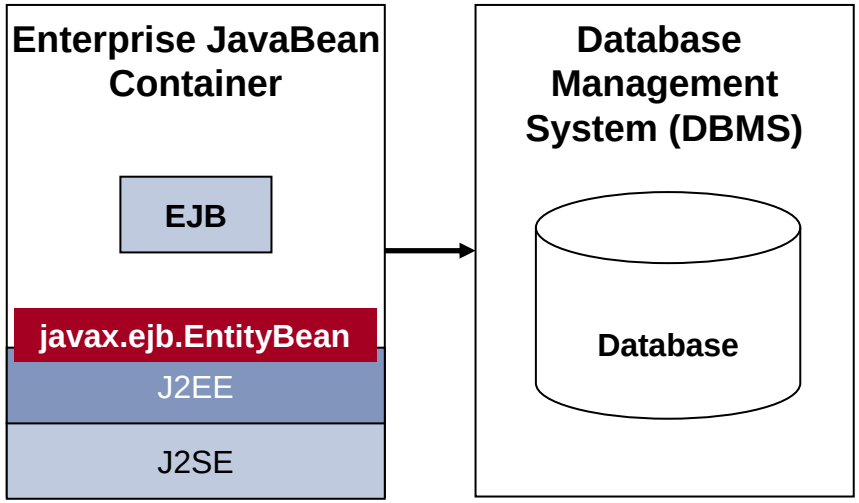
Lebenszyklus von Stateful Session Beans



- Verwaltung des Lebenszyklus erfolgt durch Client und Container
  - Container ruft die Callback-Methoden der Schnittstelle javax.ejb.SessionBean auf
- Stateful Session Beans können in den Zustand „passive“ übergehen
  - Serialisierung des Zustands der Bean auf einen Sekundärspeicher
  - Freigabe des Speicherbereichs der Bean

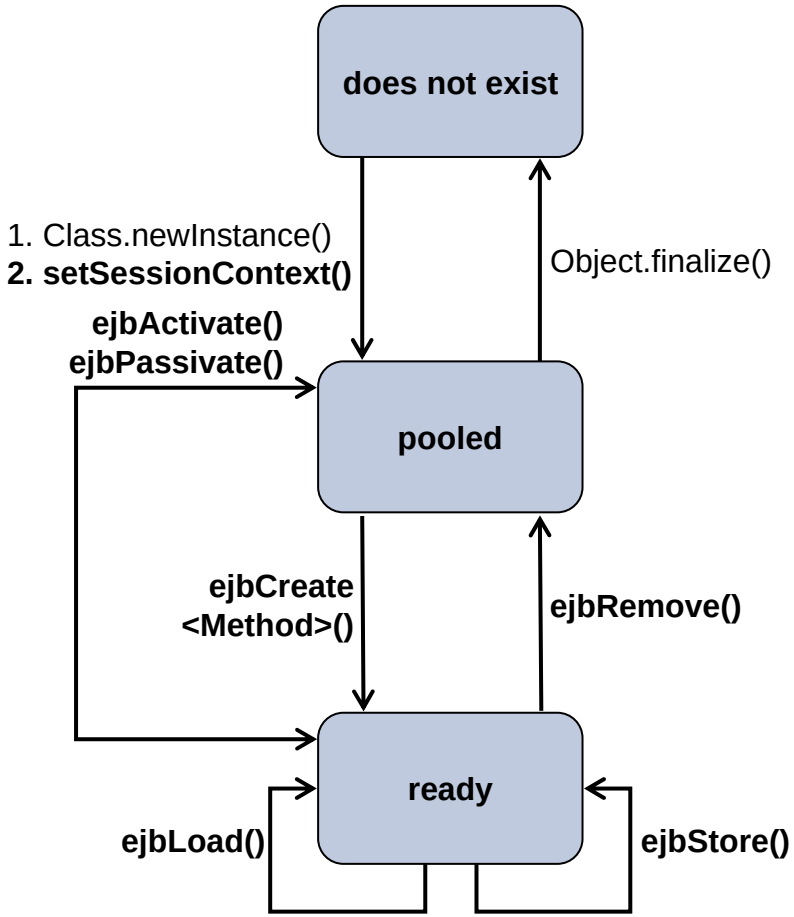
Quelle: in Anlehnung an [Sun04b]

Eigenschaften von Entity Beans



- Objekt-orientierte Sicht auf Datenstrukturen in einer Datenbank
  - Primär: relationale Datenbanken
- Sicht des Clients: für jede Entität in der Datenbank existiert ein Entity Objekt
- Mehrere Clients können gleichzeitig auf ein Entity Objekt zugreifen
  - Container synchronisiert den Zugriff mit Hilfe von Transaktionen

Lebenszyklus von Entity Beans



Quelle: in Anlehnung an [Sun04b]

- Verwaltung des Lebenszyklus erfolgt durch Container
  - Container ruft die Callback-Methoden der Schnittstelle javax.ejb.EntityBean auf
- Zustand „pooled“
  - keine aktuelle Assoziation mit einem Datensatz in der Datenbank
  - es wurden keine Daten geladen („leere Instanz“)

Finder Methods

- Bestandteil des Home Interfaces einer Entity Bean
  - jede Methode definiert einen Weg um ein Entity Objekt oder eine Menge von Entity-Objekten zu finden
- Methodennamen beginnen mit dem Prefix „find“
- Verwendung der Methodenargumente bei der Suche
  - Abfragen werden deklarativ auf Basis der sog. EJB Query Language im Deployment Descriptor definiert
- Ergebnis ist das EJBObject oder eine Menge der EJBObjects in einer Collection
  - Collection findByName(String name)
  - EJB QL: SELECT object(o) FROM CustomerCMP o WHERE o.name = ?1
- Verbindlich ist die Finder Method findByPrimaryKey()
  - <Component Interface> findByPrimary(<Primary Key Class>)

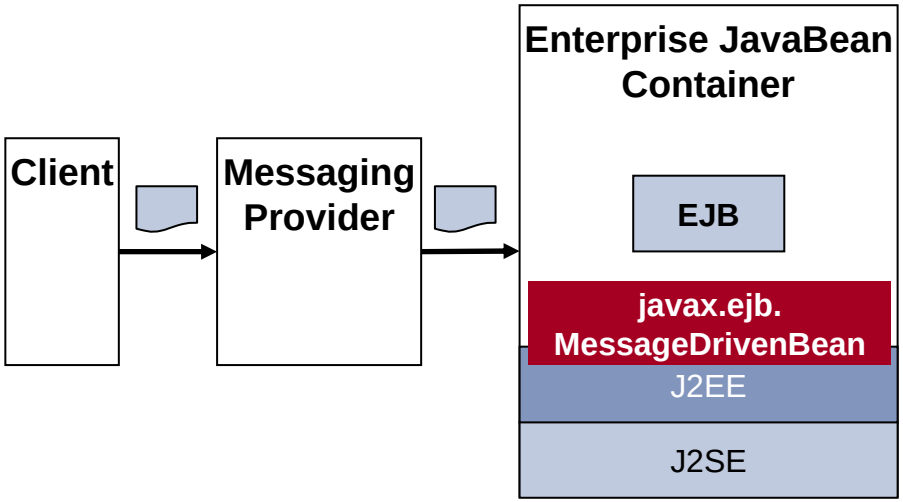
Container-Managed Persistence (CMP) Entity Bean

- Entwickler beschreibt persistente Attribute der Bean
- Werkzeuge generieren zum Installationszeitpunkt den Code
  - Lese-, Speicher- und Löschoperationen auf den Daten in der Datenbank
- Unterschiedliche relationale DBMS können bei Generierung berücksichtigt werden
  - Beachte: unterschiedliche Dialekte und Erweiterungen des SQL Standards

Bean-Managed Persistence (BMP) Entity Bean

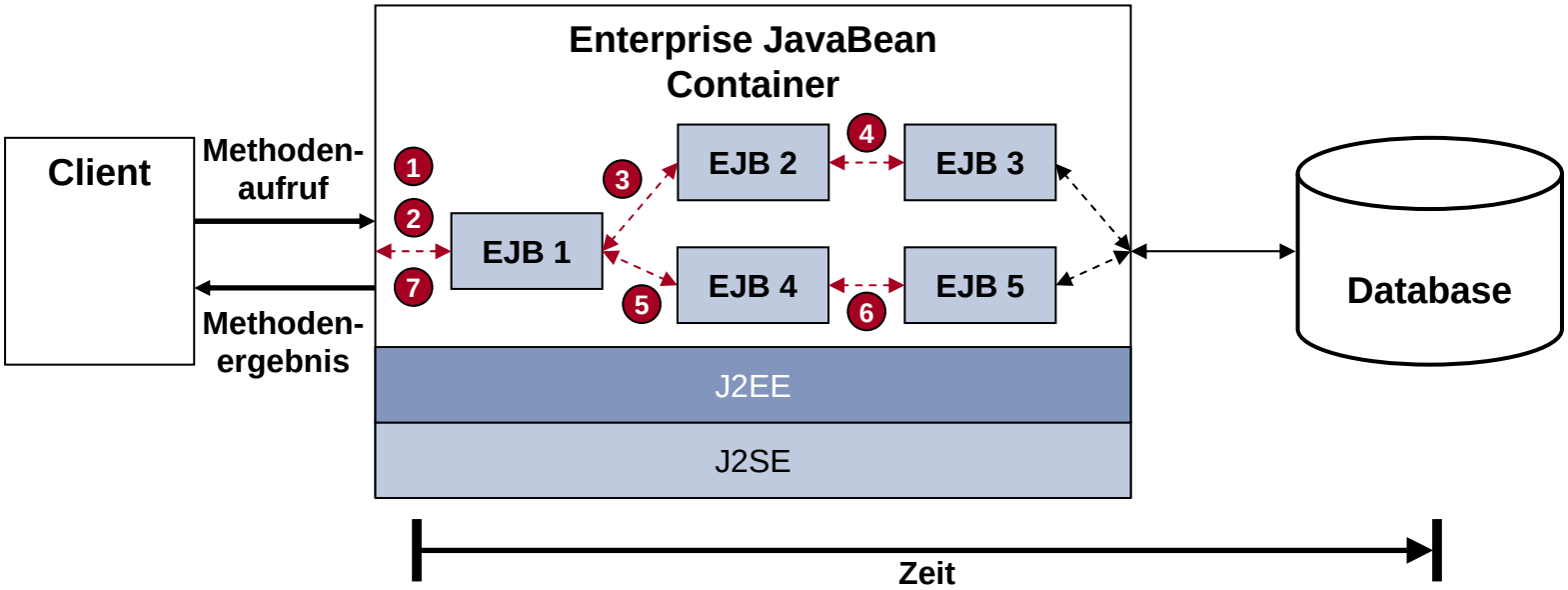
- Entwickler ist für die Definition der persistenten Attribute und des Codes verantwortlich
  - Implementierung von Code zum Lesen, Speichern und Löschen von Daten in der Datenbank
  - Schnittstelle zwischen Container und DBMS basiert i.d.R. auf SQL
- Unterschiedliche DBMS müssen vom Entwickler berücksichtigt werden
  - auch nicht-relationale DBMS können verwendet werden (hierarchisch, XML-basiert)

Eigenschaften von Message-Driven Beans



- Konsument einer asynchronen Nachricht, welche zum Aufruf der Message-Driven Bean führt
- Nutzung durch den Client erfolgt durch das Versenden einer Nachricht an einen sog. Messaging Destination
  - Keine Implementation des Home, Component oder Metadata Interfaces

Transaktionen



1. Container: beginTransaction()
2. EJB 1: executeOrder()
3. EJB 2: calculateInvoice()
4. EJB 3: createInvoice()
5. EJB 4: calculateActualStock()
6. EJB 5: updateStock()
7. EJB Container: EndTransaction()

Agenda

Einführung und Motivation

Verteilte Objekte und Komponenten

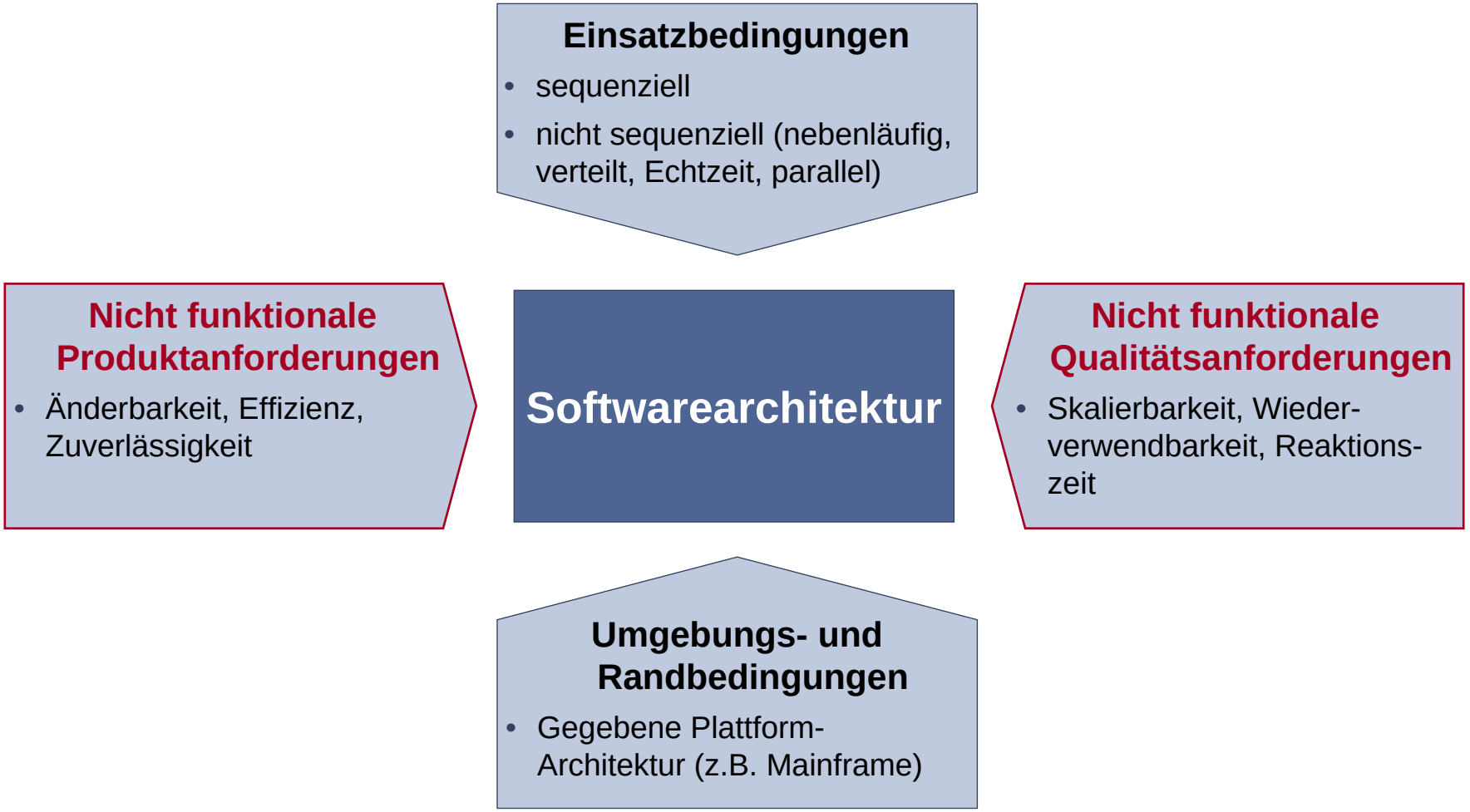
Verteilte Softwarearchitekturen

J2EE-Plattform

J2EE-basierte Softwarearchitektur

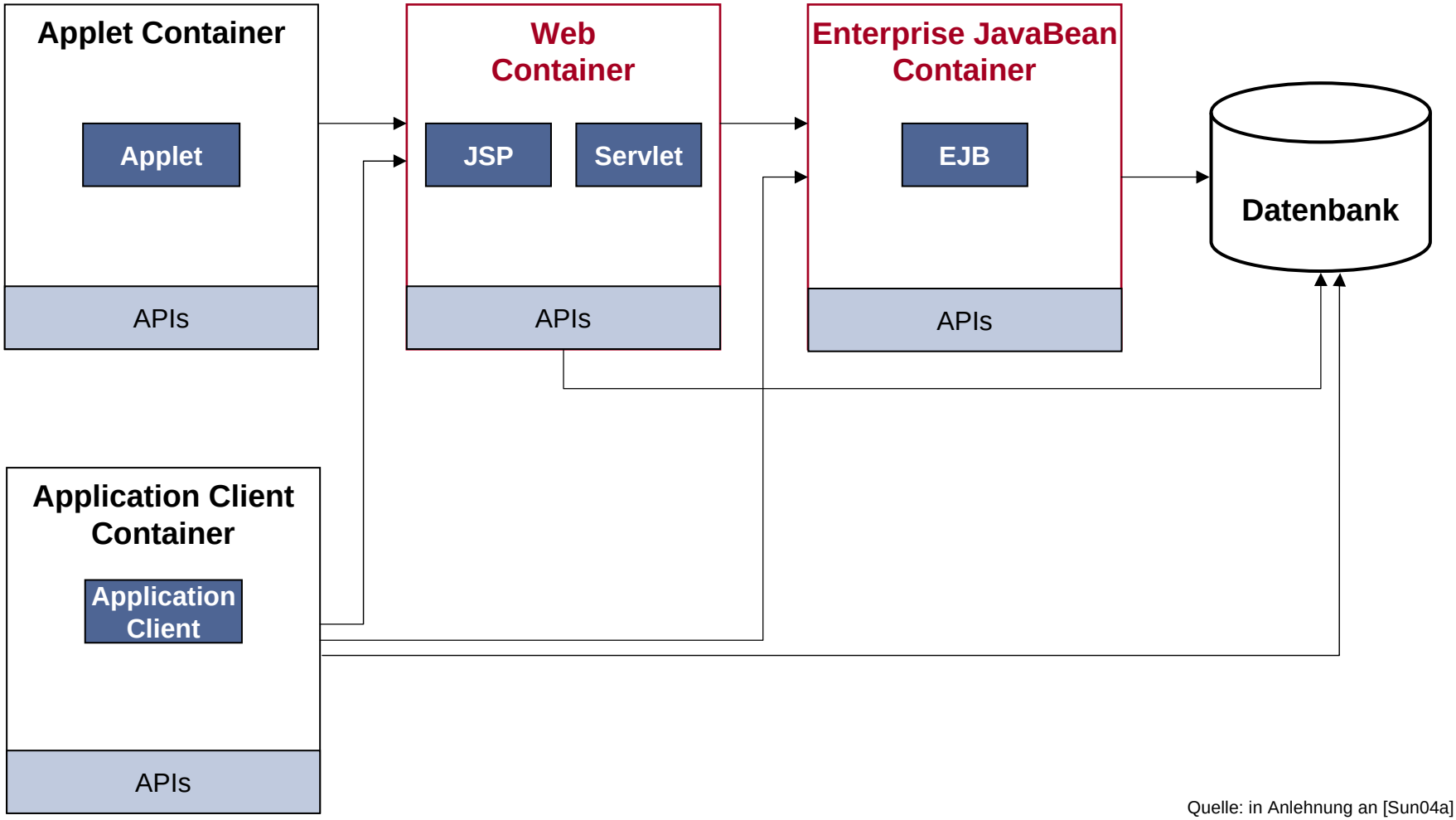
Aspekte der Verteilung von J2EE-Anwendungen

Einflussfaktoren auf eine Softwarearchitektur



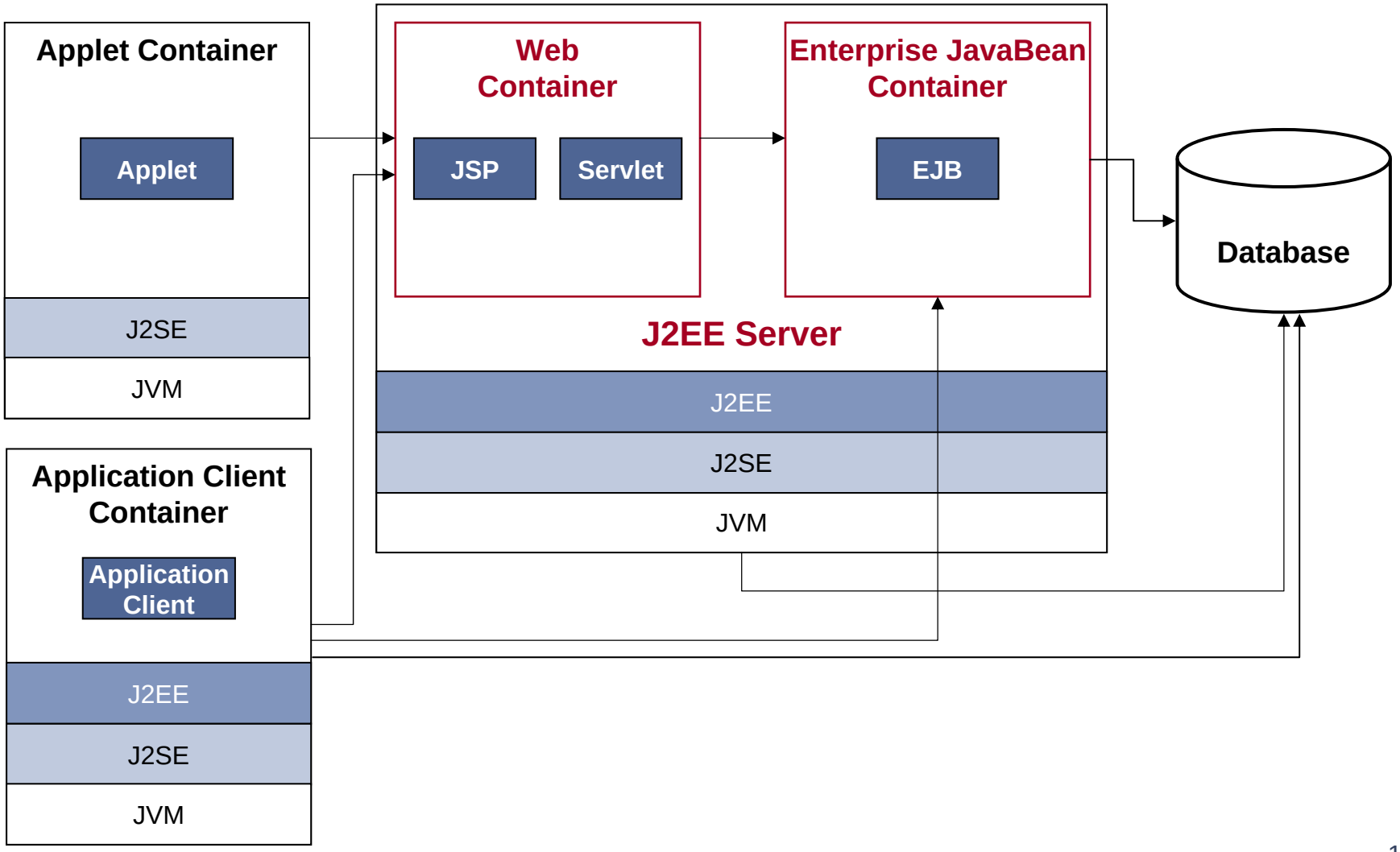
Quelle: in Anlehnung an [Balz96]

Verteilung des Web und EJB Containers



Quelle: in Anlehnung an [Sun04a]

Konfiguration der J2EE Server und Container



Kriterien der Konfiguration (1/3)

- Sicherheit
  - Trennung des Web Servers vom Applikationsserver durch eine oder mehrere Firewalls
    - Screening Router (Protocol Firewall)
    - Application Gateway (Domain Firewall)
- Geschwindigkeit
  - Antwortzeit für eine Transaktion unter Berücksichtigung einer bestimmten Last
- Durchsatz
  - Anzahl der ausgeführten Transaktionen pro Zeiteinheit innerhalb eines bestimmten Zeitraums

Quelle: vgl. [ABBC04] und [SCHI04]

Kriterien der Konfiguration (2/3)

- Skalierbarkeit
  - Erweiterung einer Konfiguration durch Hinzufügen von Rechnerkapazität
    - Hard- und Softwareelemente
  - Vertikale Skalierung
    - zusätzliche Prozesse, identischer Rechnerknoten
    - ermöglicht Ausfallsicherheit bezüglich der relevanten Softwareelemente (z.B. Betriebssystem) und des/der Serverprozesses/Serverprozesse
  - Horizontale Skalierung
    - zusätzliche Prozesse auf unterschiedlichen Rechnerknoten
    - ermöglicht zusätzlich Ausfallsicherheit bezüglich der relevanten Hardwareelemente

Quelle: vgl. [ABBC04] und [SCHI04]

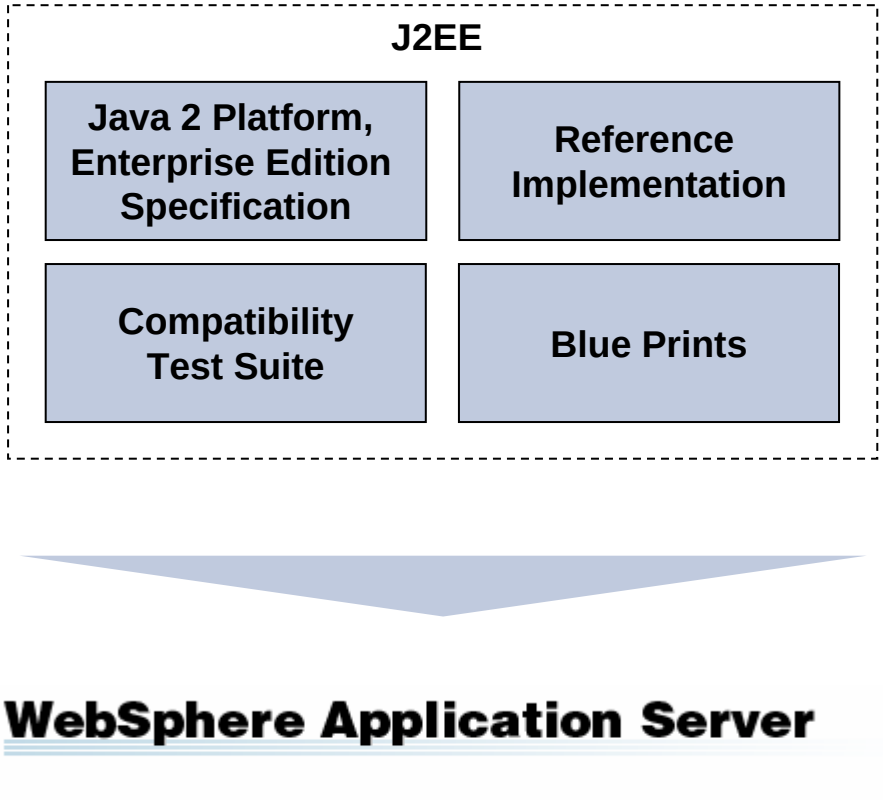
Kriterien der Konfiguration (3/3)

- Verfügbarkeit
  - Wahrscheinlichkeit, dass die Funktionen einer Anwendung nutzbar sind, wenn sie benötigt werden
    - Betriebszeit (Operating Time) vs. Stillstandzeit (Downtime)
    - Failure vs. Fault
  - Maßnahme: durch Redundanz kann sog. Single Point of Failure vermieden werden
- Wartbarkeit
  - Aktualisierung von Hard- und Softwareelementen
    - Ist abhängig von der Verfügbarkeit

Quelle: vgl. [ABBC04] und [SCHI04]

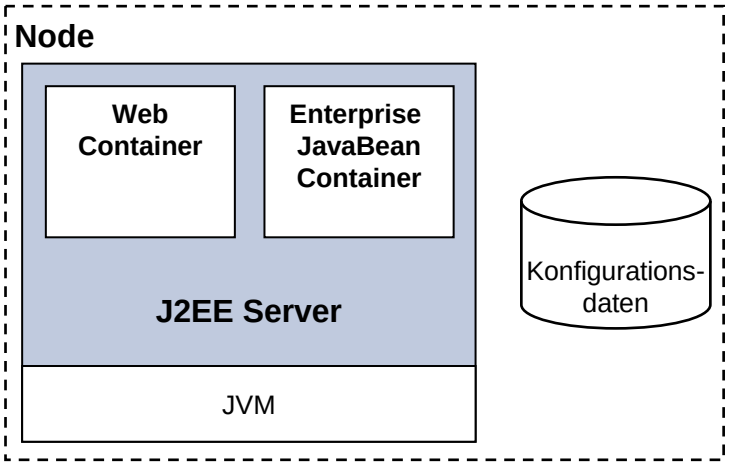
Aspekte der Verteilung von J2EE-Anwendungen

Konfiguration am Beispiel des WebSphere Application Server



- Anbieter eines J2EE-kompatiblen Produkts implementieren entsprechende Werkzeuge und auch weiterführende Funktionen
- Anlehnung der Begriffe und Konfigurationsbeispiele an das ausgewählte Produkt
  - aber: die Begriffe und Konzepte sind i.d.R. auf andere Produkte übertragbar!

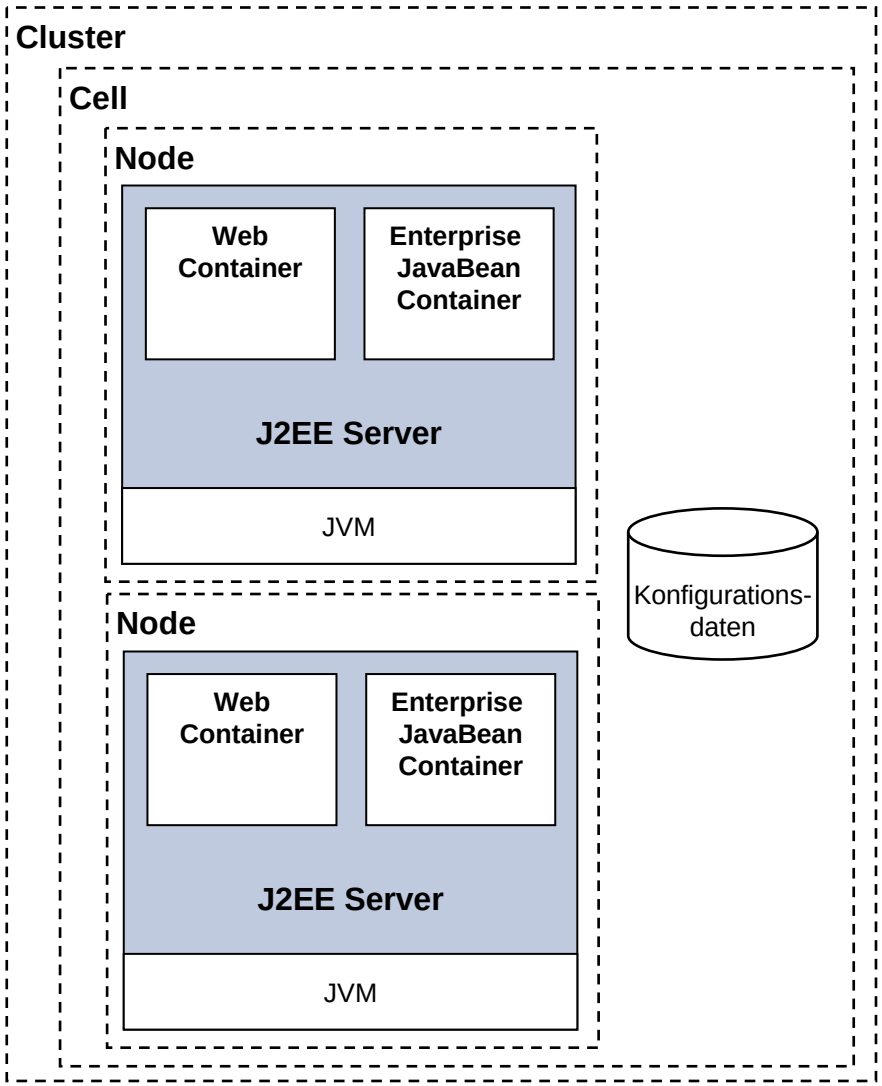
Begriffsdefinitionen: Server und Node



- Server
  - J2EE-kompatibler Server
    - auch: J2EE Server genannt
  - Laufzeitumgebung mit Web und/oder EJB Container
  - Ausführung in eigener JVM
- Node
  - eine Installation eines Servers einschließlich der relevanten Konfigurations-daten

Quelle: in Anlehnung an [SCHI04]

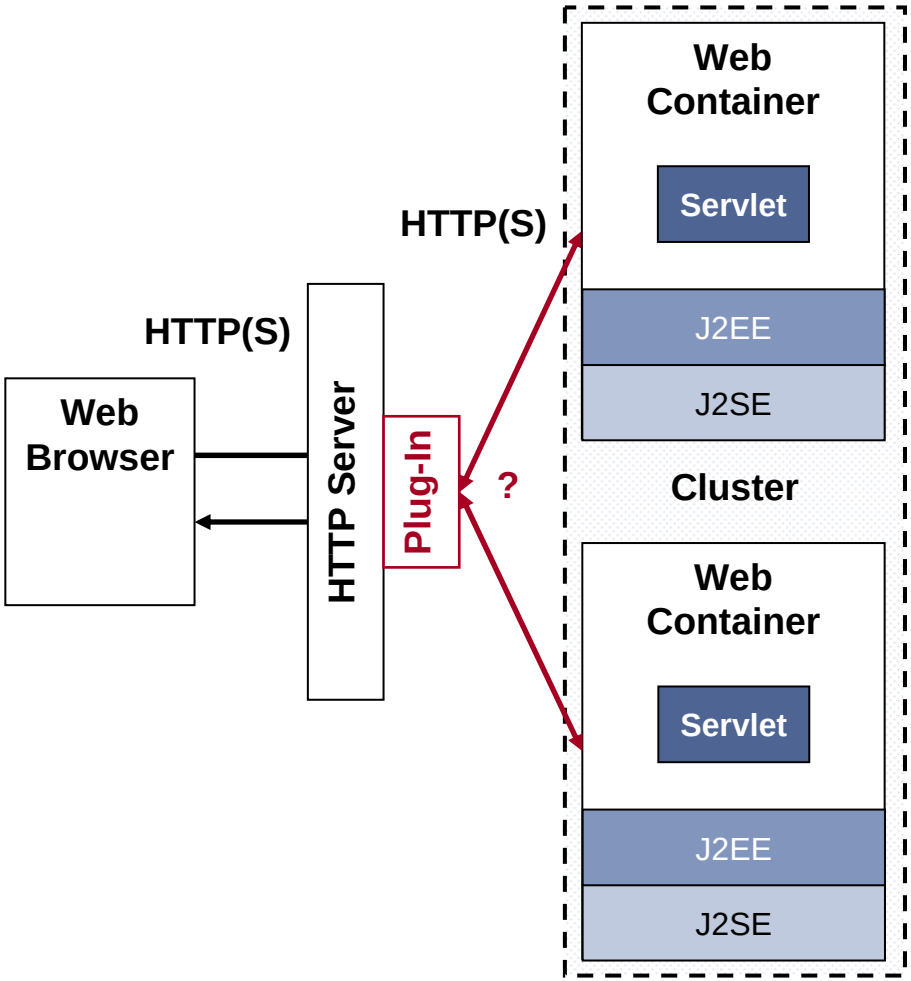
Begriffsdefinitionen: Cell und Cluster



- Cluster
  - logische Gruppierung von Cells zur Lastverteilung
  - kann eine oder mehrere Cells umfassen
- Cell
  - logische Gruppierung von mehreren Nodes zu einer administrativen Einheit
  - unterschiedliche Cells können unterschiedliche Versionen des J2EE Servers und der Anwendung enthalten

Quelle: in Anlehnung an [SCHI04]

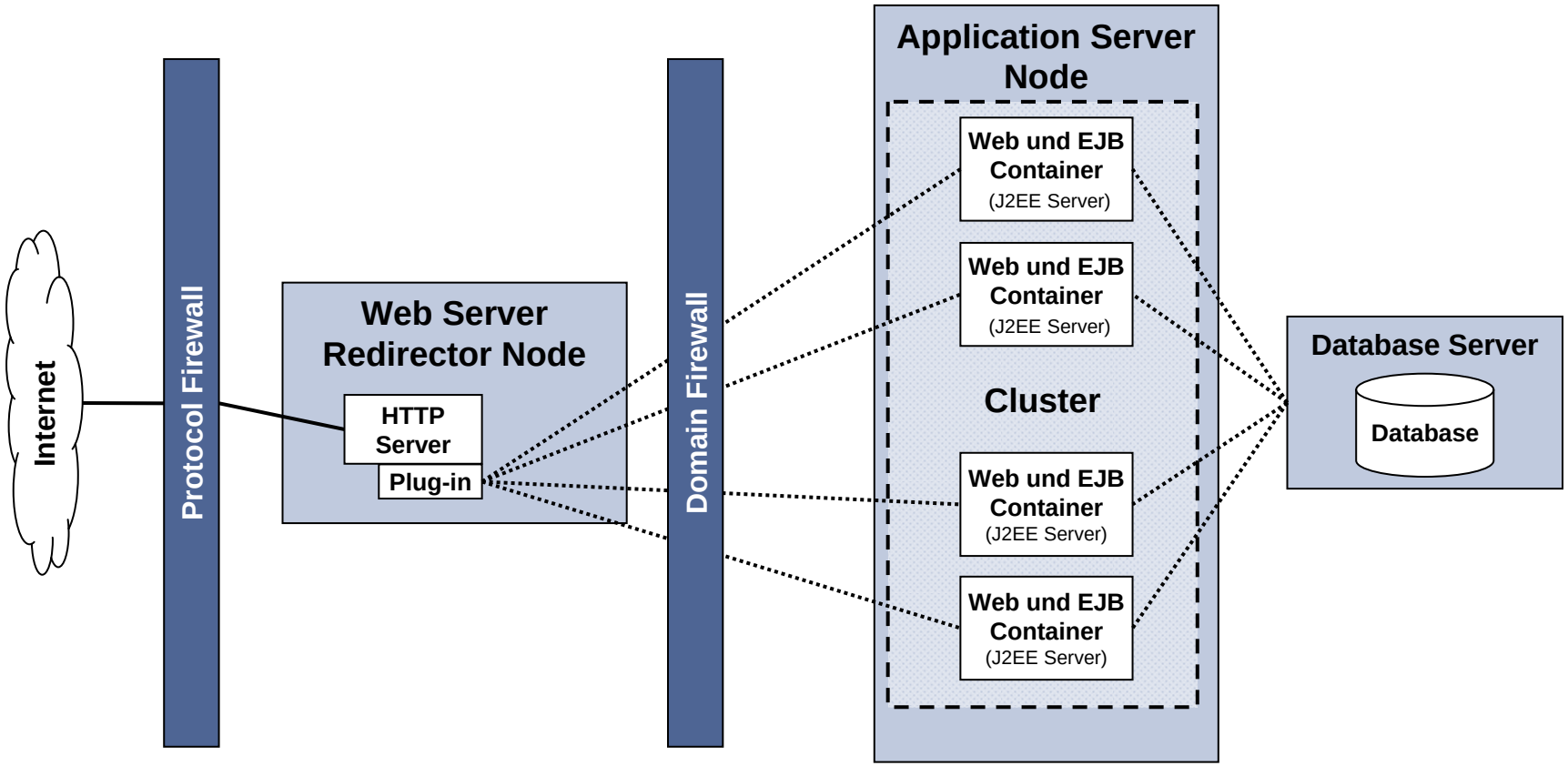
Verteilung der Last



- Auch: Load Balancing oder Workload Management
- Basiert auf Konfiguration eines Clusters
- Beispiel: Routing des HTTP-Requests
  - Anzahl der Cluster Member
  - Gewichtung des Member
  - % routed to Server_1 = weight_1 / (weight_1 + weight_2 + ... + weight_n)
  - Verfügbarkeit eines Cluster Members wird berücksichtigt

Quelle: in Anlehnung an [SCHI04]

Beispiel: Vertikale Skalierung mit Cluster (1/2)

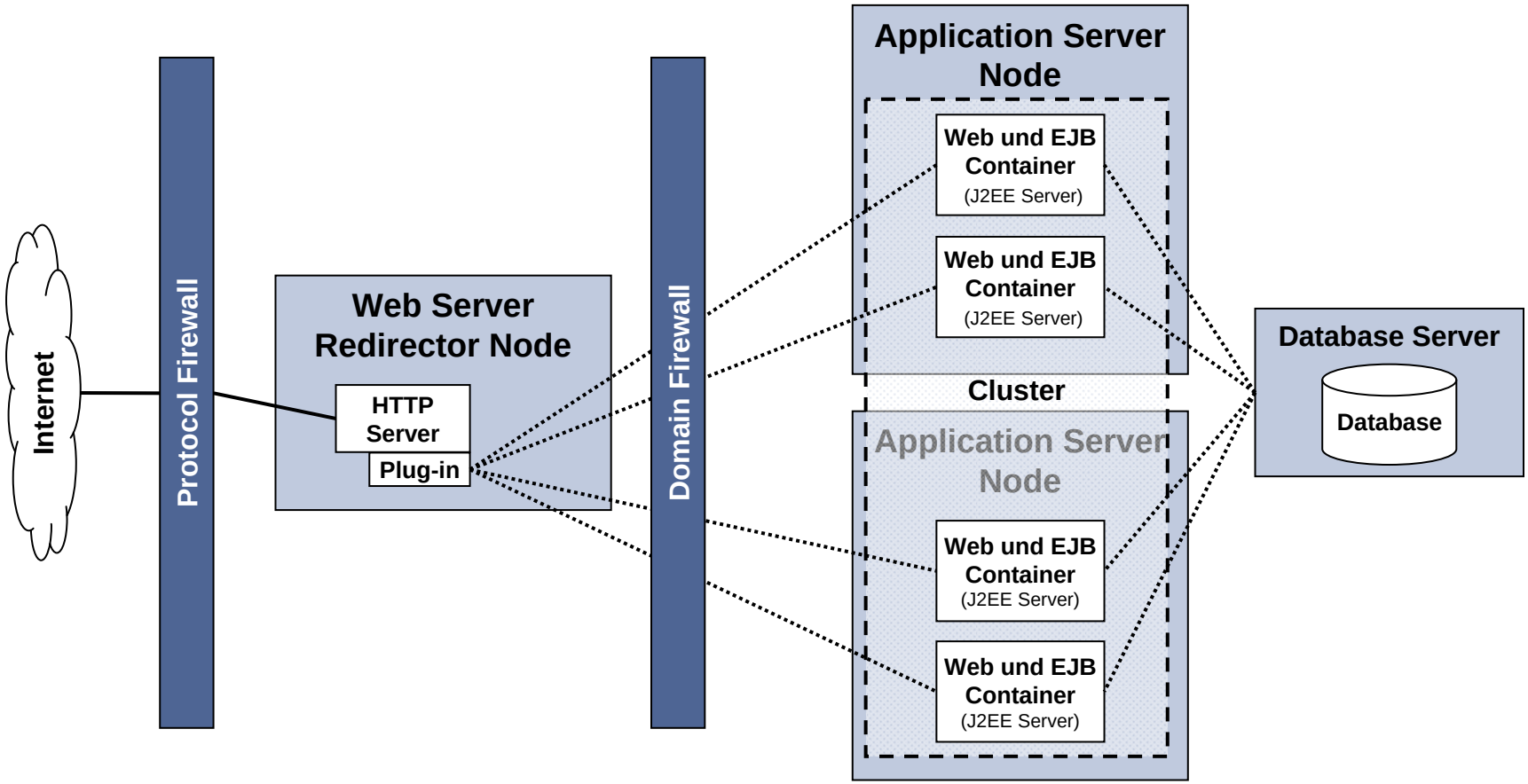


Quelle: in Anlehnung an [SCHI04]

Beispiel: Vertikale Skalierung mit Cluster (2/2)

- Mehrere Nodes werden auf einem Rechnerknoten zu einem Cluster zusammengefasst
- Jeder Node enthält die gleiche(n) Anwendung(en)
- Vorteile
  - Effiziente Nutzung der Rechnerkapazität
  - Lastverteilung innerhalb des Clusters
  - Ausfallsicherheit bezüglich der JVM-Prozesse
- Nachteile
  - Application Server Node ist Single Point of Failure
  - Web Server Redirector Node ist Single Point of Failure

Beispiel: Horizontale Skalierung mit Cluster (1/2)

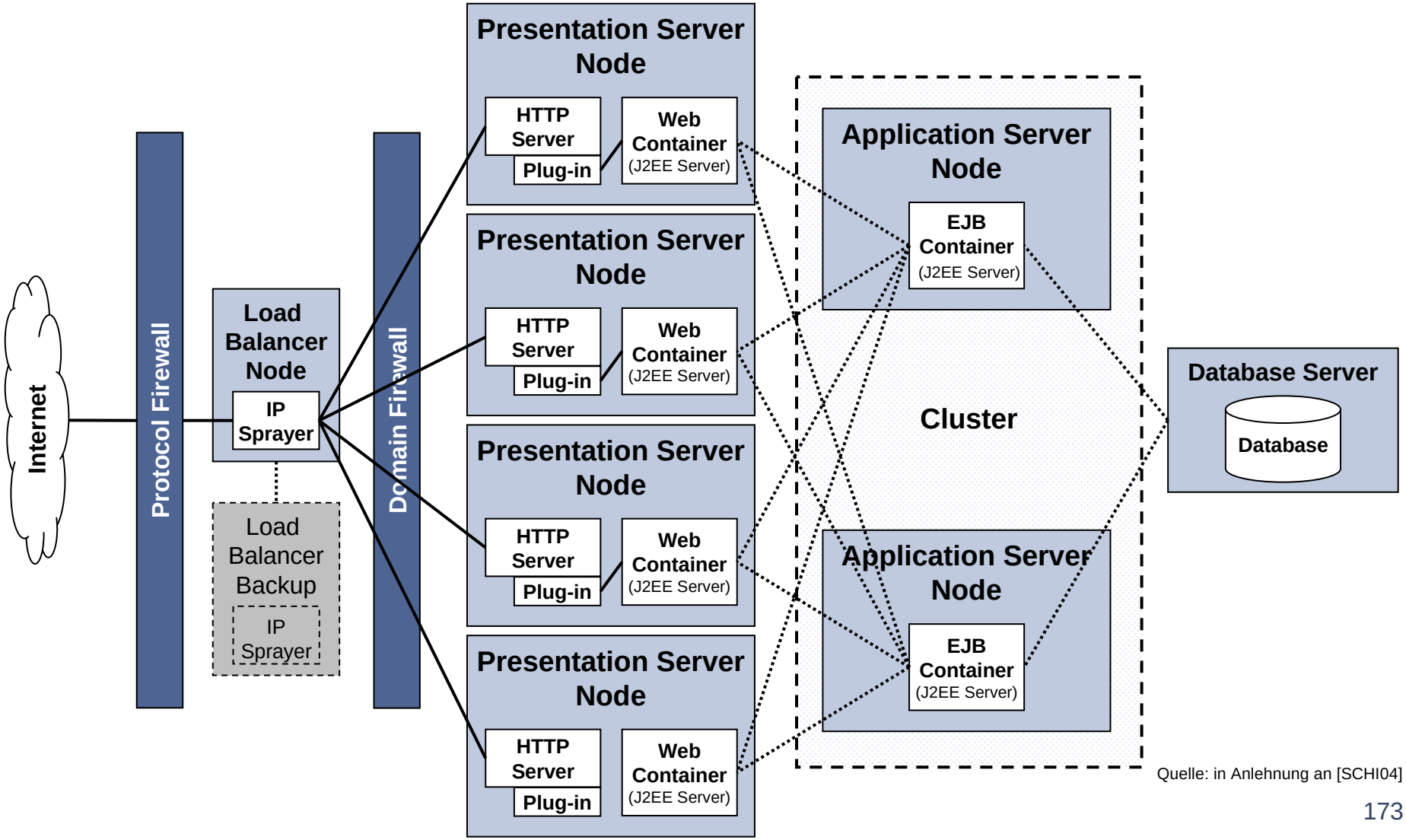


Quelle: in Anlehnung an [SCHI04]

Beispiel: Horizontale Skalierung mit Cluster (2/2)

- Cluster Member sind auf mehrere Rechnerknoten verteilt
- Vorteile
  - Ausfallsicherheit bezüglich der JVM-Prozesse und der Hardware
  - Lastverteilung innerhalb des Clusters (über mehrere Rechnerknoten)
- Nachteile
  - Zusätzliche Hardware
  - Erhöhter Administrationsaufwand durch zusätzliche Rechnerknoten
  - Web Server Redirector Node ist Single Point of Failure

Beispiel: Horizontale Skalierung mit IP Sprayer (1/2)



Quelle: in Anlehnung an [SCHI04]

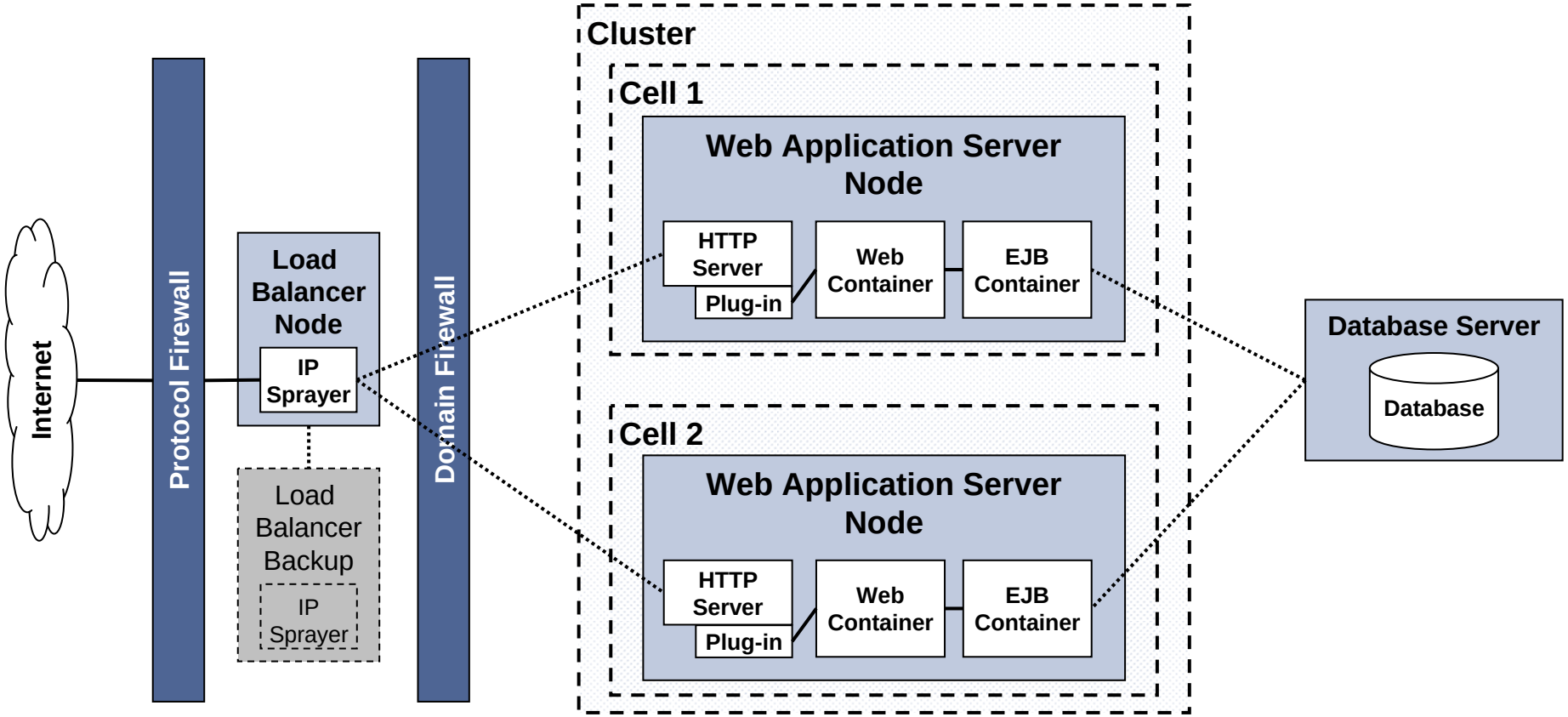
Beispiel: Horizontale Skalierung mit IP Sprayer (2/2)

- IP Sprayer auf dem Load Balancer Node verteilt die HTTP(S)-Requests auf verschiedene Presentation Server Nodes
  - Load Balancer Backup Node eleminiert den IP Sprayer als Single Point of Failure
- Presentation Server Nodes bilden keinen Cluster
  - Web Container erhält einen Request immer vom Web Server Plug-in des identischen Nodes
  - aber: Zusammenfassung in Cluster ist auch möglich
- Application Server Nodes bilden einen Cluster
  - Cluster erscheint als ein logischer J2EE-Server mit EJB Container, welche u.a. die Datenbankzugriffe kapselt

Beispiel: Horizontale Skalierung mit IP Sprayer (2/2)

- Vorteile
  - Erhöhte Web Server Geschwindigkeit durch die Lastverteilung auf mehrere Web Server
  - Erhöhter Durchsatz: mehrere Rechnerknoten verarbeiten die Client Requests
  - Kein Single Point of Failure
    - Beachte: Datenbank Server wird nicht betrachtet
    - Presentation Server Nodes sollten auf mehrere Rechnerknoten verteilt werden
- Nachteile
  - Zusätzliche Hardware (insbesondere Load Balance Nodes)
  - Erhöhter Administrationsaufwand durch zusätzliche Rechnerknoten

Beispiel: Mehrfache Zellen (1/2)



Quelle: in Anlehnung an [SCHI04]

Beispiel: Mehrfache Zellen (2/2)

- Jede Zelle ist eine administrative Einheit und enthält genau eine Version der Anwendung und des J2EE Servers
- IP Sprayer arbeitet auf der Ebene der Zellen
- Vorteile
  - Isolation von Softwarefehlern insbesondere bei der Installation von
    - neuen Versionen der Anwendung
    - neuen Versionen des J2EE Servers
    - Fixes und Patches
  - Erhöhte Geschwindigkeit, da keine Kommunikation zwischen den Prozessen von unterschiedlichen Zellen
- Nachteile
  - Erhöhter Administrationsaufwand, da jede Zelle eine eigene administrative Einheit ist

Zusammenfassung und Ausblick

Agenda

Einführung und Motivation

Verteilte Objekte und Komponenten

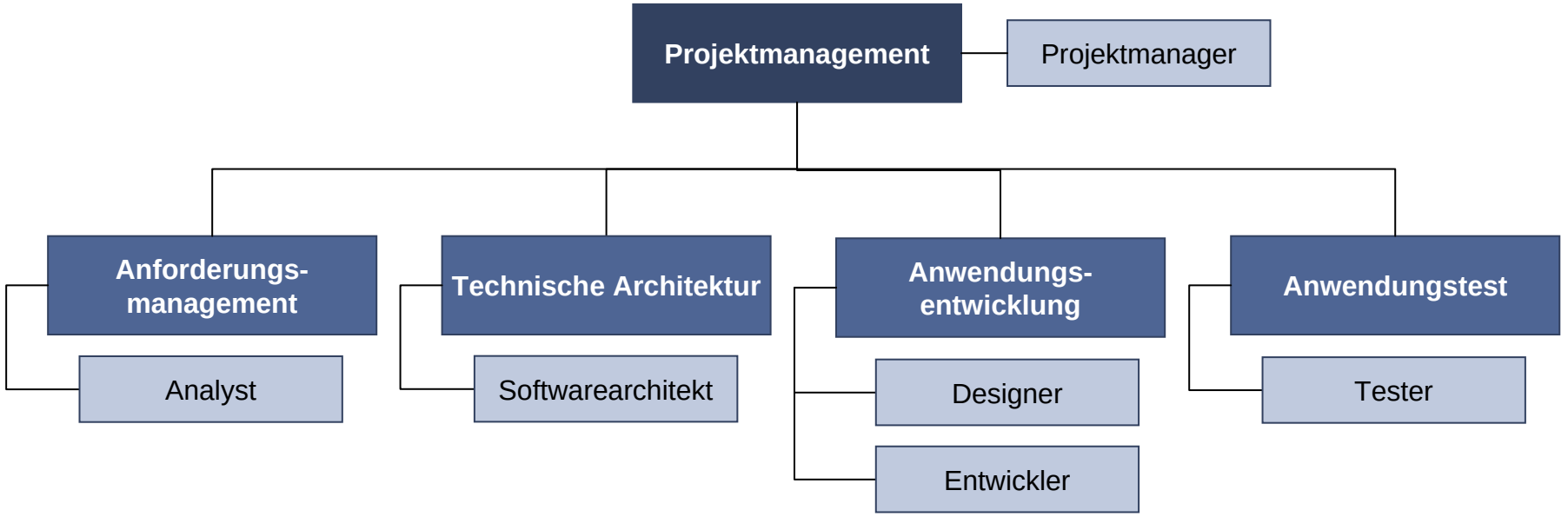
Verteilte Softwarearchitekturen

J2EE-Plattform

J2EE-basierte Softwarearchitektur

Aspekte der Verteilung von J2EE-Anwendungen

Rollen in einer Projektorganisation



Literaturverzeichnis

Verwendete Literatur und Quellen (1/4)

- [ABBC04] Armstrong, Eric; Ball, Jennifer; Bodoff, Stephanie; Carson, Debbie Bode; Evans, Ian; Green, Dale; Haase, Kim; Jendrock, Eric Carson, Debbie Bode; Evans, Ian; Green, Dale; Haase, Kim; Jendrock, Eric; The J2EE Tutorial v1.4; Sun Microsystems; 2004; http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html
- [Balz96] Balzert, Helmut; Lehrbuch der Software-Technik: Software-Entwicklung; Spektrum Akademischer Verlag; Heidelberg, Berlin, Oxford; 1996
- [BaCK03] Bass, Len; Clements, Paul; Kazman, Rick; Software Architecture in Practice; 2. Auflage; Addision-Wesley; Boston; 2003

Verwendete Literatur und Quellen (2/4)

- [Kruc95] Kruchten, Philippe; Architectural Blueprints—The "4+1" View Model of Software Architecture; In, IEEE Software; Vol. 12 No. 6; November 1995; pp. 42-50
- [Rati03] Rational; The Rational Unified Process; Version 2003.06.13; http://www.ibm.com/developerworks/rational/products/rup/

Verwendete Literatur und Quellen (3/4)

- [SCHI04] Sadler, Carla; Clifford, Lee; Heyward, Jeff; Iwamoto, Arihiro; Jakusz, Noelle; Laursen, Lars Bek; Lee, WonYoung; Mauny, Isabell; Rabbi, Shafkat; Sanchez, Ascension; IBM WebSphere Application Server V5.1 System Management and Configuration; WebSphere Handbook Series; 2004; http://www.redbooks.ibm.com/abstracts/sg246195.html?Open
- [Sun04a] Sun Microsystems; Java 2 Platform, Enterprise Edition (J2EE) Specification, v1.4; http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf
- [Sun04b] Sun Microsystems; Enterprise JavaBeans Specification, Version 2.1; http://java.sun.com/products/ejb/docs.html

Verwendete Literatur und Quellen (4/4)

- [Sun04c] Sun Microsystems; Java Servlet Specification, Version 2.4; http://java.sun.com/products/servlet/download.html
- [Sun04d] Sun Microsystems; JavaServer Pages Specification, Version 2.0; http://java.sun.com/products/jsp/reference/api/index.html