



## ***Netzprogrammierung Organisation und Einleitung***

Prof. Dr.-Ing. Robert Tolksdorf  
Freie Universität Berlin  
Institut für Informatik  
Netzbasierte Informationssysteme  
mailto: [tolk@inf.fu-berlin.de](mailto:tolk@inf.fu-berlin.de)  
<http://www.robert-tolksdorf.de>



## Organisatorisches

## Veranstaltungsinhalt

- „Die Vorlesung stellt Prinzipien, Sprachen und Middleware für die Entwicklung verteilter, insbesondere Web-basierter Anwendungssystemen dar“
- Ausrichtung praktisch auf *Netzprogrammierung*, weniger auf
  - Netztechnik
  - Konzepte Verteilter Systeme
  - Eigenschaften Verteilter Algorithmen
  - Anwendung verteilter Systeme

# Unterlagen

- Folien im Netz unter [http://www.ag-nbi.de/lehre/0607/V\\_NP](http://www.ag-nbi.de/lehre/0607/V_NP)
- In den Foliensätzen Verweise auf Quellen

- Titel mit vorlesungsbezogenen Inhalten:
  - Heinzl, Steffen, Mathes, Markus. Middleware in Java. ISBN: 3-528-05912-5. Vieweg Verlag 2005.
  - Heiko Wöhr. Web-Technologien. ISBN 3-89864-247-X dpunkt.verlag 2004
  - Bengel, Günther. Grundkurs Verteilte Systeme. ISBN: 3-528-25738-5. Vieweg 2004.
  - Andrew S. Tanenbaum, Maarten van Stehen. Verteilte Systeme. ISBN 3-8273-7057-4 Pearson 2004
  - Mathias Lothar, Cornelius Wille, Fritz Zbrog, Reiner Dumke. Web Engineering, ISBN 3-8273-7080-9-Pearson 2004
- Verbindlich ist Inhalt der Vorlesungsfolien

- Zwei Übungen bei
  - Konstantin Clemens (zacharov@inf.fu-berlin.de) und
  - Jan Sebastian Siwy (siwy@inf.fu-berlin.de):
    - Mi 16-18h
    - Mi 18-20h
- Dort:
  - Weitere Fragen
  - Hinweise untereinander zu Übungsaufgaben
- Übungsblätter
  - Enthalten praktische Aufgaben, die bewertet werden
  - Enthalten Fragen zur weiteren Diskussion und zur eigenen Beantwortung

- Leistungsnachweis
  - Note des Leistungsnachweises ist die individuelle Klausurnote
  - Voraussetzung für Klausur:  
Regelmäßige und aktive Teilnahme an Veranstaltung
  - Aktive Teilnahme:
    - Regelmäßige Teilnahme
    - Abgabe aller Übungsblätter in Gruppen
    - Bestehen von n-1 Übungsblättern

## Formalitäten

- Für BSc Studierende ist eine *verbindliche* Anmeldung zur Veranstaltung notwendig
- Ohne diese Anmeldung dürfen *keine* Leistungen erbracht werden
- Verbindliche Anmeldung mit Unterschrift in der nächsten Woche

- Mailingliste
  - [nbi\\_v\\_np@lists.spline.inf.fu-berlin.de](mailto:nbi_v_np@lists.spline.inf.fu-berlin.de)
  - Eintragung über [http://lists.spline.inf.fu-berlin.de/mailman/listinfo/nbi\\_v\\_np](http://lists.spline.inf.fu-berlin.de/mailman/listinfo/nbi_v_np)
  - *Verbindliche* Ankündigungen zur Veranstaltung
  - Allgemeine Nachfragen der Teilnehmer
  - Gegenseitige Kommunikation unter Teilnehmern
- Mit Robert Tolksdorf
  - [tolk@inf.fu-berlin.de](mailto:tolk@inf.fu-berlin.de)
  - <http://www.robert-tolksdorf.de/sprechstunde>
  - Bevorzugt: *Elektronische* Nachfrage

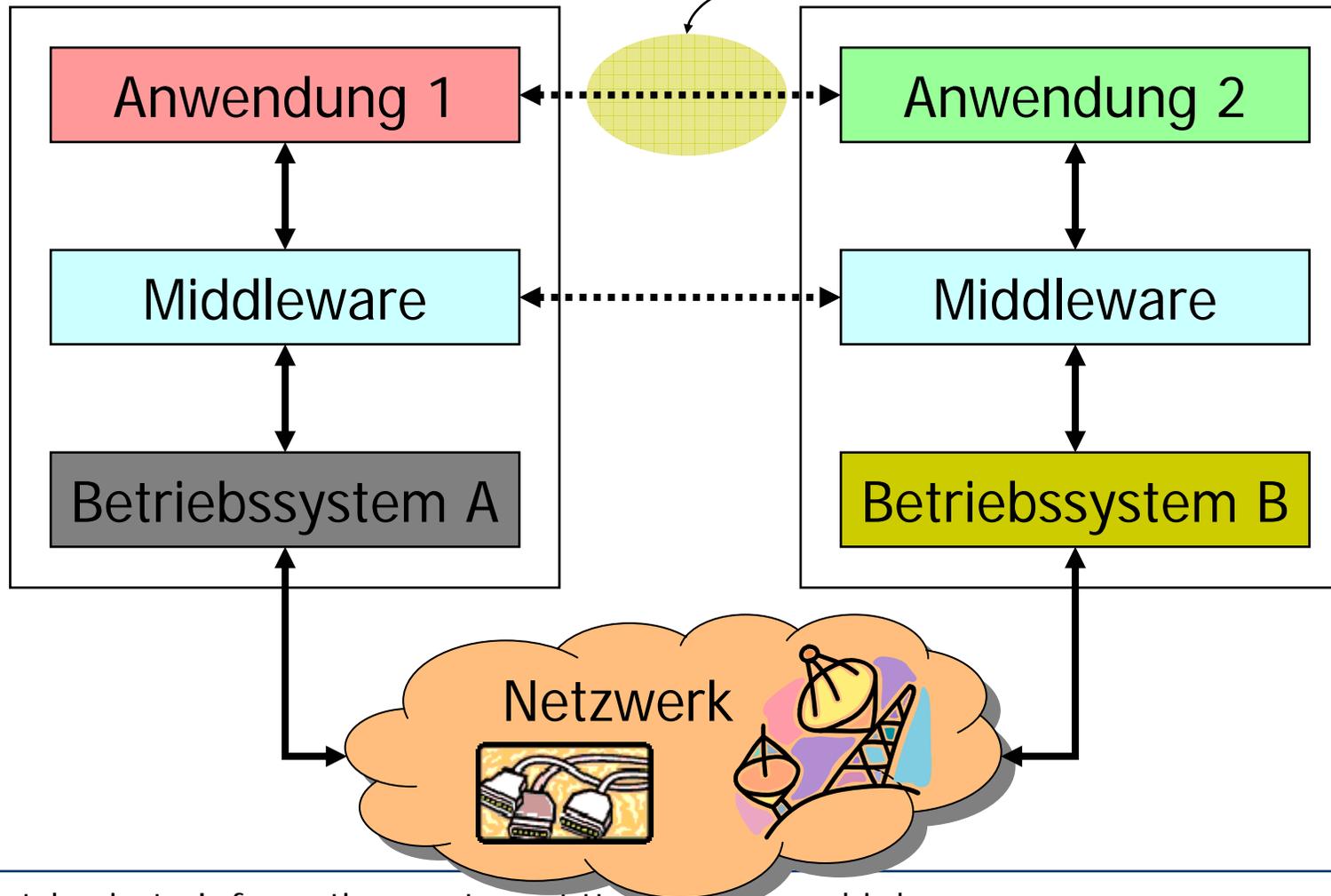


## Netzprogrammierung

- „Die Vorlesung stellt Prinzipien, Sprachen und **Middleware** für die Entwicklung verteilter, insbesondere Web-basierter Anwendungssystemen dar“
- Middleware stellt Technologien, Abstraktionen, Schnittstellen bereit mit der netzbasierte Systeme realisiert werden können

# Middleware: Kleber zur Verbindung von Anwendungen

Gewinn: Anwendungen kommunizieren



- Für Anwendung erscheint verteiltes System homogener als es tatsächlich ist
- Beispiel:  
Objekte können genutzt werden unabhängig von ihrem Ort
- Notwendig:  
Ortsunterschiede maskieren, transparent machen
- *Ortstransparenz* durch Middleware-Dienste, die Methodenaufruf zum richtigen Ort transportieren ohne dass der Nutzer das tun muss

- „*Transparency*: The property of masking from applications the details and the differences in mechanisms used to overcome problems caused by distribution.“ [ISO Referenzmodell Open Distributed Processing, Part1]
- Aspekte die ganz oder teilweise transparent sein können:
  - Heterogenität der beteiligten Software
  - Heterogenität der beteiligten Hardware
  - Ort und Bewegung der beteiligten Komponenten
  - Mechanismen zur Fehlerbehandlung relativ zu Qualitätserfordernissen
- „*Distribution transparencies are used to hide aspects of ODP systems that arise through their distribution.* Within an ODP system, the ODP infrastructure supports a set of distribution transparencies. The applications determine those transparencies that they need and that must be provided by the infrastructure. Other aspects of distribution are handled by the applications themselves.“ [ODP Part1]

- Zugriffstransparenz (access transparency)
  - Auf alle Ressourcen und Dienste wird gleich zugegriffen
    - Maskiert unterschiedliche Datenrepräsentationen
    - Maskiert unterschiedliche Aufrufmechanismen
    - Maskiert unterschiedliche genutzte Programmiersprachen
  - Wichtigste und notwendige Transparenz
- Ortstransparenz (location transparency)
  - Ort von Ressourcen und Diensten ist nicht bekannt
    - Maskiert die Nutzung von Ortsinformationen beim Auffinden anderer Objekte
  - Ermöglicht rein logische Sicht auf Namen im System
  - Keine Ortsinformationen zur Nutzung von Namen nötig

- Nebenläufigkeitstransparenz
  - Auf Ressourcen kann nebenläufig zugegriffen werden
    - Maskiert die Nutzung von Synchronisationsmechanismen bei der Implementierung der Zugriffe
- Skalierungstransparenz
  - Das System ist erweiterbar
    - Maskiert Rekonfigurationen der Hard- und Softwarebasis
- System ist für Benutzer also
  - homogen                      statt heterogen
  - lokal                            statt verteilt
  - exklusiv genutzt            statt gemeinsam genutzt
  - immer gleich                 statt rekonfiguriert

- Persistenztransparenz (persistence transparency)
  - Objekte erscheinen immer zugänglich
    - Maskiert die Aktivierung und Deaktivierung von Objekten zu Klienten und zum Objekt selber
  - Durch Persistenz überlebt ein Objekt Zeiten in denen ein System nicht ausführt, speichert, kommuniziert etc.
  - Objekte erscheinen immer verfügbar
- Transaktionstransparenz (transaction transparency)
  - Transaktionen funktionieren immer ganz oder gar nicht
    - Maskiert Koordination von Aktivitäten auf Objekten durch die Konsistenz erhalten bleibt
  - Beinhaltet Planung, Überwachung, Wiederrufen von Aktivitäten
  - Transparenz erfordert die Verfeinerung der Spezifikation von Objekten um transaktionale Eigenschaften

- Migrationstransparenz (migration transparency)
  - Prozesse und Ressourcen verschiebbar
    - Maskiert Ortsänderungen von Objekten
  - Macht beispielsweise Lastausgleich transparent
  - Mit Fehler- und Persistenztransparenz kombinierbar
- Relokationstransparenz (relocation transparency)
  - Maskiert Änderungen des Ortes gebundener Schnittstellen
  - Objekte können während der Interaktion mit Klienten verschoben oder ersetzt werden
  - Sichert Weiterarbeit auch trotz temporär inkonsistenter Sichten

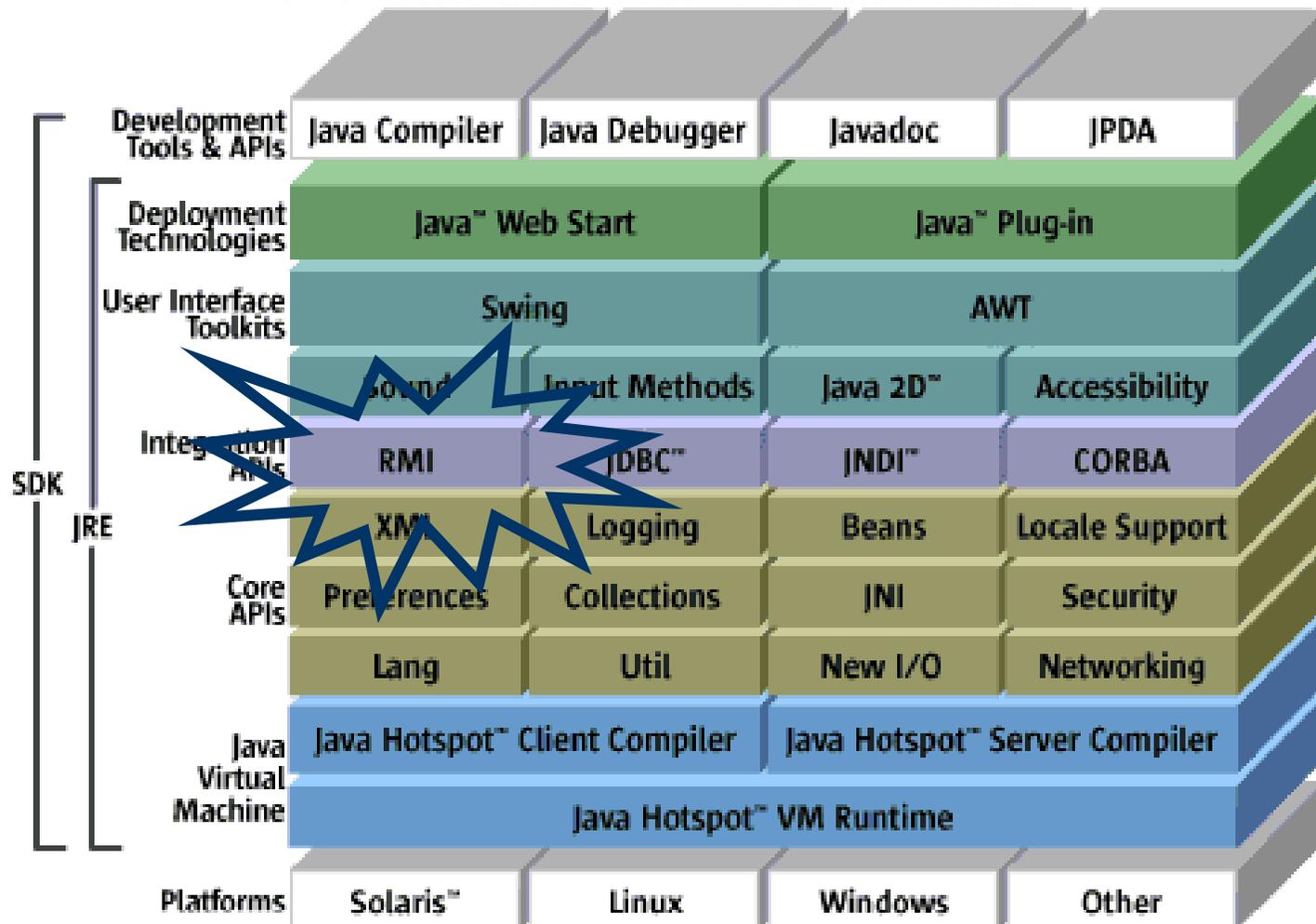
- Replikationstransparenz (replication transparency)
  - Mehrfach vorhandene Objekte werden so zugegriffen als existieren nur ein Exemplar
    - Maskiert den Einsatz einer Gruppe von Objekten an einer Schnittstelle
  - Erhöht Performanz und Verfügbarkeit
- Fehlertransparenz (failure transparency)
  - System erscheint trotz Fehlern intakt
    - Maskiert Fehler und Wiederherstellung anderer Objekte
  - Beispielmechanismen: Checkpointing, Transaktionen
  - Falls vorhanden, ideale, fehlerfreie Welt für Programmierer
- System ist für Benutzer also auch
  - unterbrechungsfrei      statt fehlerbehaftet
  - leistungsoptimiert      statt nur zusammengesaltet

# Systemdienste

- Transparenzen werden vom System erzeugt
- Dazu sind Dienste unterschiedlichster Art notwendig
- Middleware umfasst:
  - Rahmenwerk für Dienste
  - Basisdienste
  - APIs
  - Implementierungen
  - ...

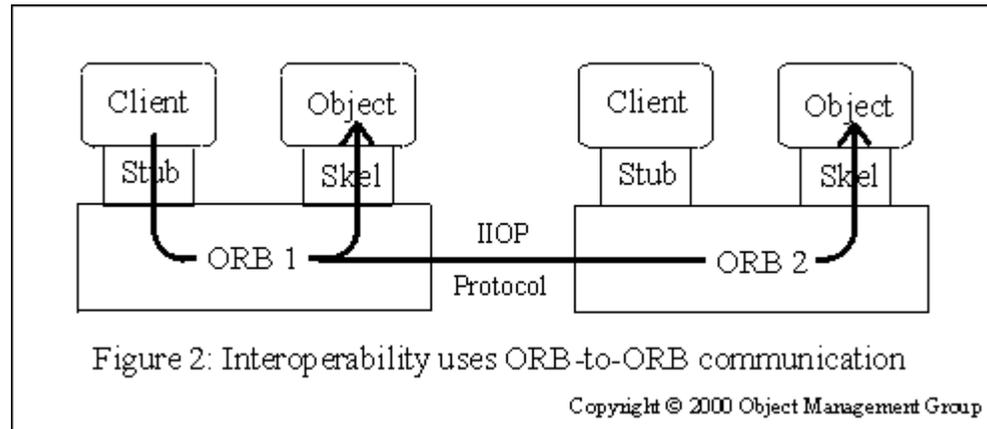
# Beispiel: Java RMI

## Java™ 2 Platform, Standard Edition v 1.4



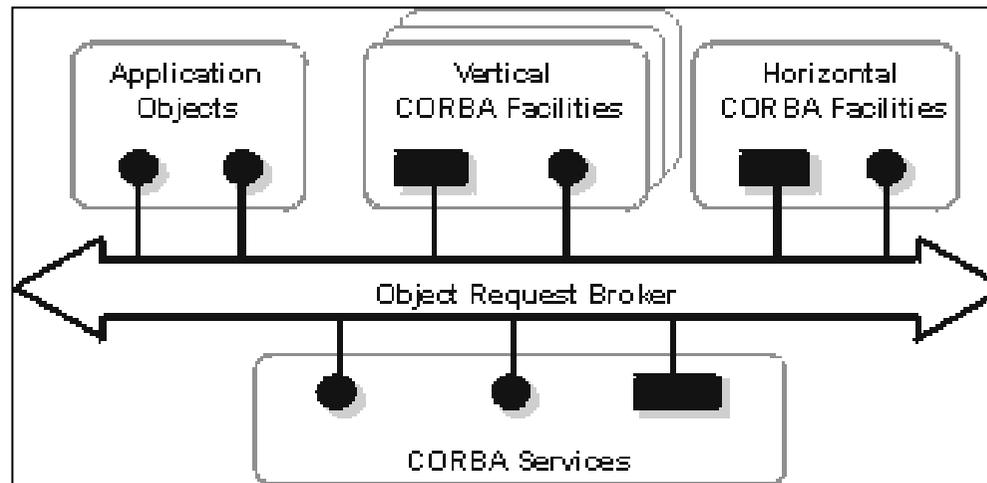
# Beispiel: OMG CORBA/OMA

- Grundlegendes Prinzip:



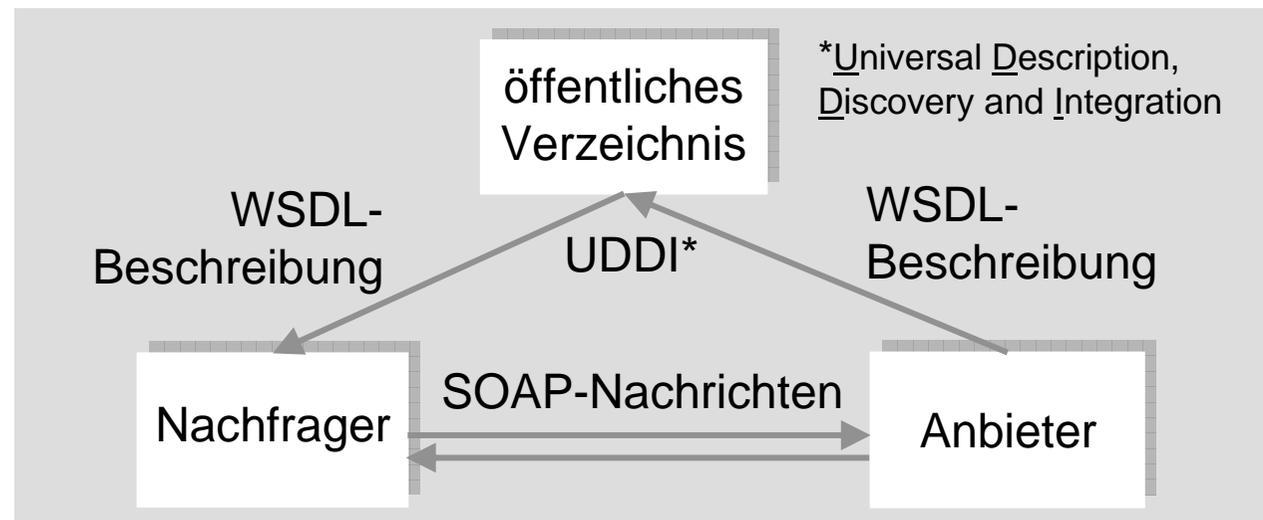
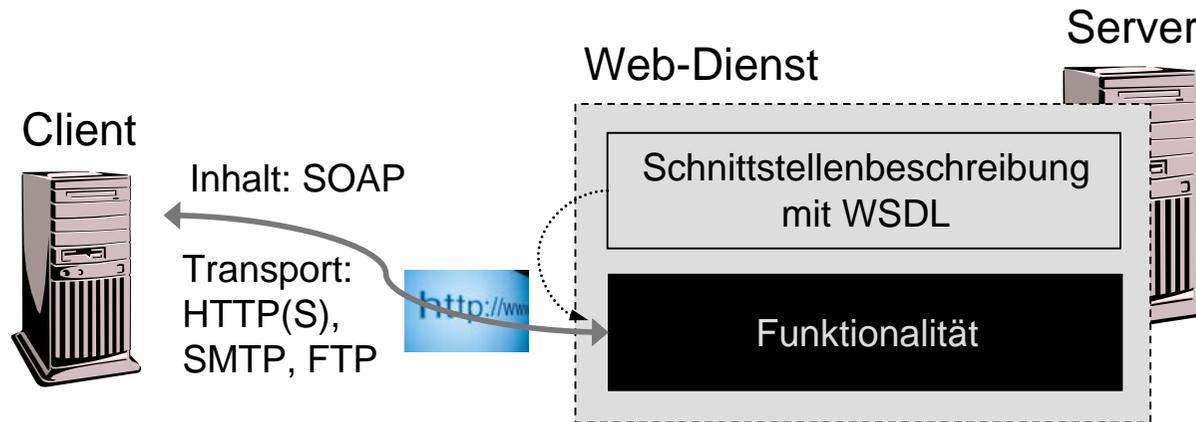
[<http://www.omg.org/gettingstarted/corbafaq.htm>]

- Ergänzt mit Diensten:



[<http://www.omg.org/gettingstarted/specintro.htm>]

# Beispiel: W3C Web Services



[Diagramme: Schild]

- Klassen von Middleware
  - Remote Procedure Call:  
Synchroner, entfernter Prozeduraufruf
  - Distributed Object Middleware:  
Synchroner, entfernter Methodenaufruf an Objekten
  - Distributed Tuples:  
Tuplespace als asynchrones Blackboard-artiges Koordinationsmedium
  - Message-Oriented Middleware:  
Mailbox-Metapher für asynchrone Entkoppelung

[David E. Bakken. *Middleware*. Encyclopedia of Distributed Computing, Kluwer 2001]

- „Die Vorlesung stellt Prinzipien, Sprachen und Middleware für die Entwicklung **verteilter**, insbesondere Web-basierter Anwendungssystemen dar“
- In Verteilten Systeme gelten eine Reihe eigener
  - semantischer Eigenschaften  
(Insbesondere Fehlersemantik)
  - technologischer Herausforderungen  
(Beispielsweise Skalierbarkeit)
  - Anwendungsanforderungen  
(Beispielsweise Integration in organisationsübergreifenden verteilten Systemen)
  - ...

# Verteilte Systeme



Nicht verteilte Systeme

Server-zentrierte Systeme

Klienten-zentrierte Client/Server Systeme

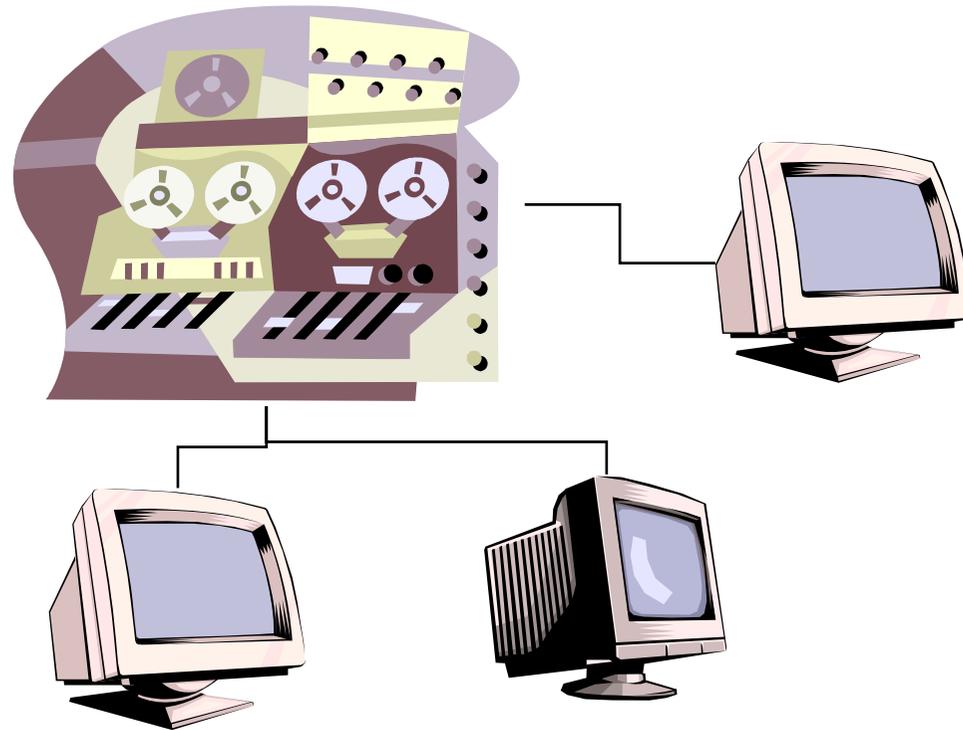
Föderierte Systeme Peer-to-Peer

Verteilte, kollaborative Systeme Peer-to-Peer

Skala: Zeit, Potential, Komplexität, Anforderungen

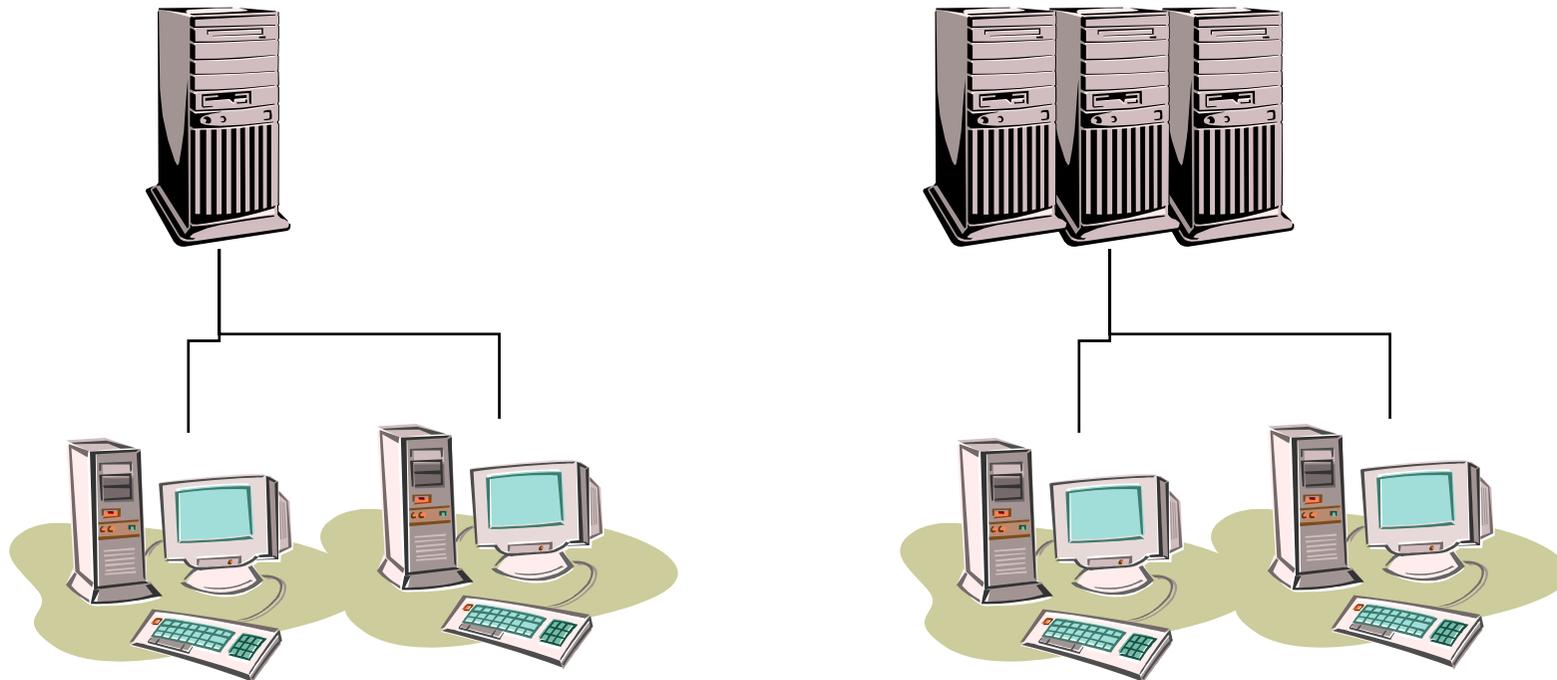
# Server Zentrierung

- „Dumme“ Terminals nutzen einen Server mit Datenbank etc.
- Programme auf Server nebenläufig
- Terminals sind nur Anzeigegeräte
- „Netzwerk“ ist lediglich Draht zur Zeichenübermittlung (seriell, Telefon)
- „Null Client“, Host-basiertes Computing

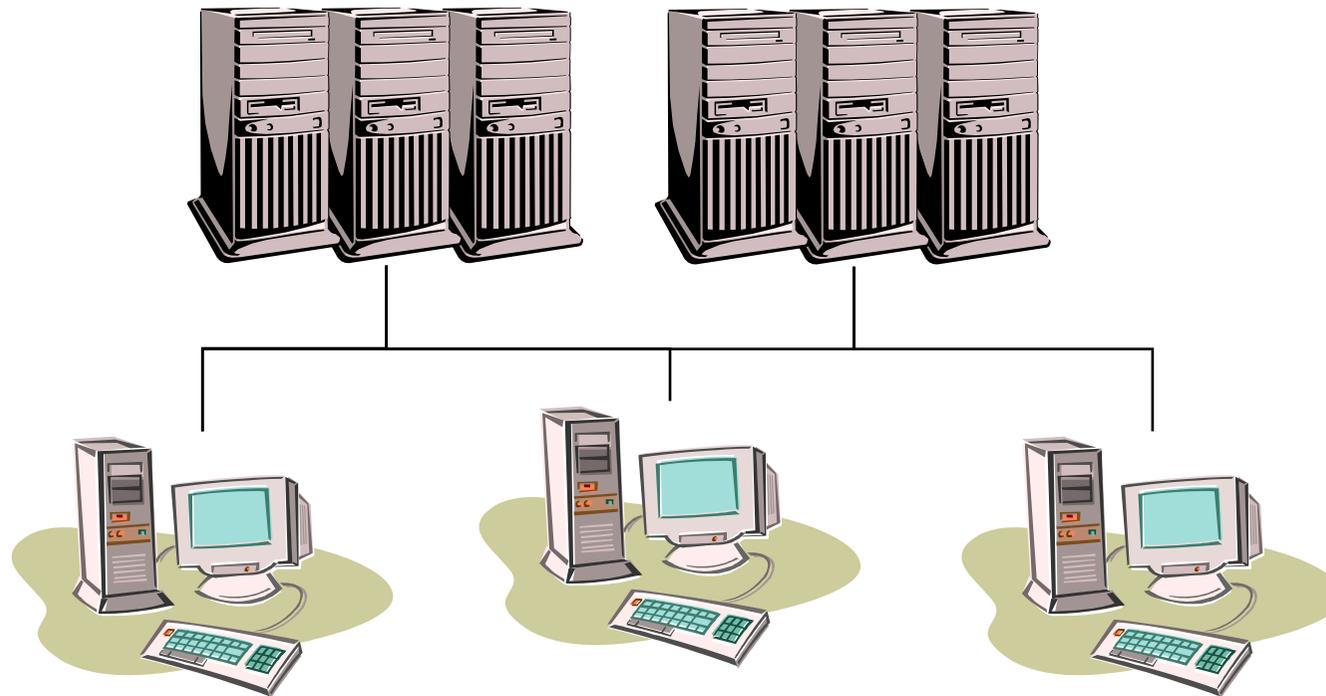


# Client Server

- Ein Server bietet Dienste an
- Alternative: Mehrere durch Middleware verbundene Server bilden einen Server
- Dünne Klienten nutzen diese durch Middleware
- Netzwerk ist ein WAN oder LAN

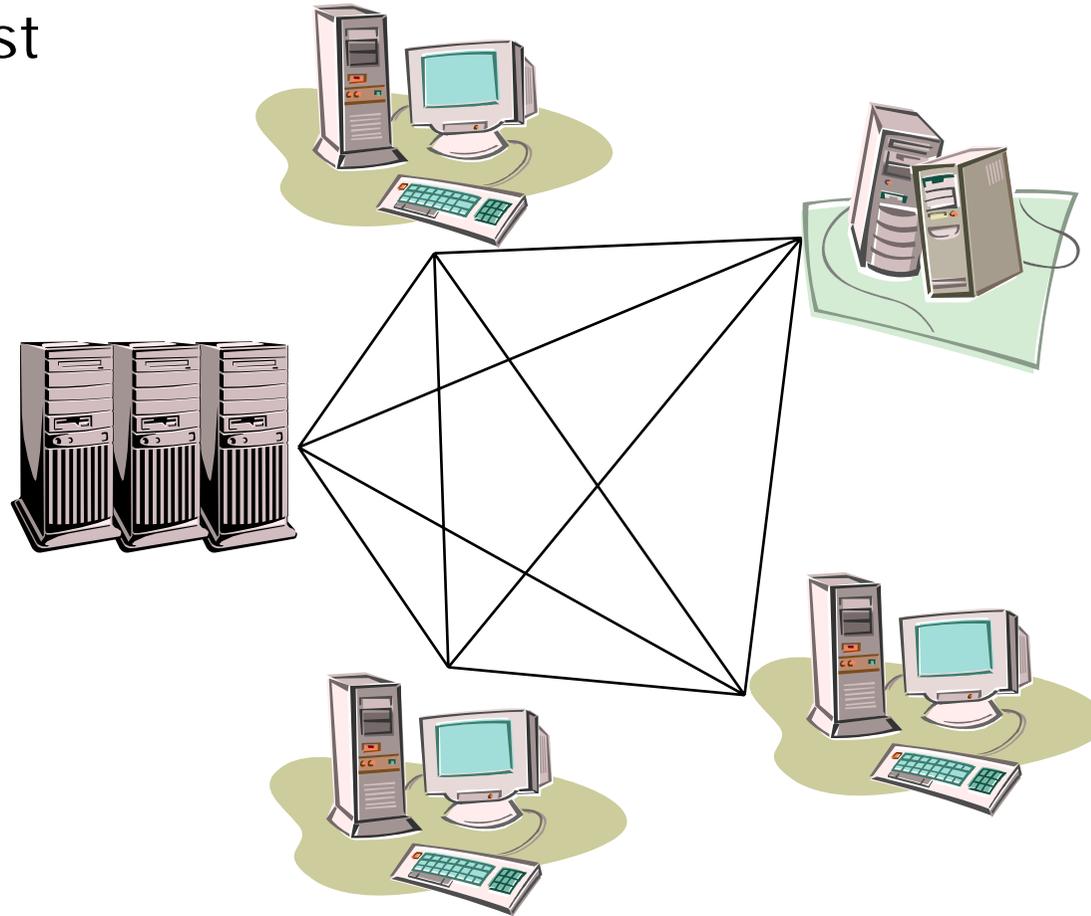


- Durch Middleware verbundene Server bleiben autonom und arbeiten zusammen
- Schlaue Klienten nutzen diese durch Middleware
- Applikationslogik teilweise im Client
- Middleware ist „etwas symmetrischer“

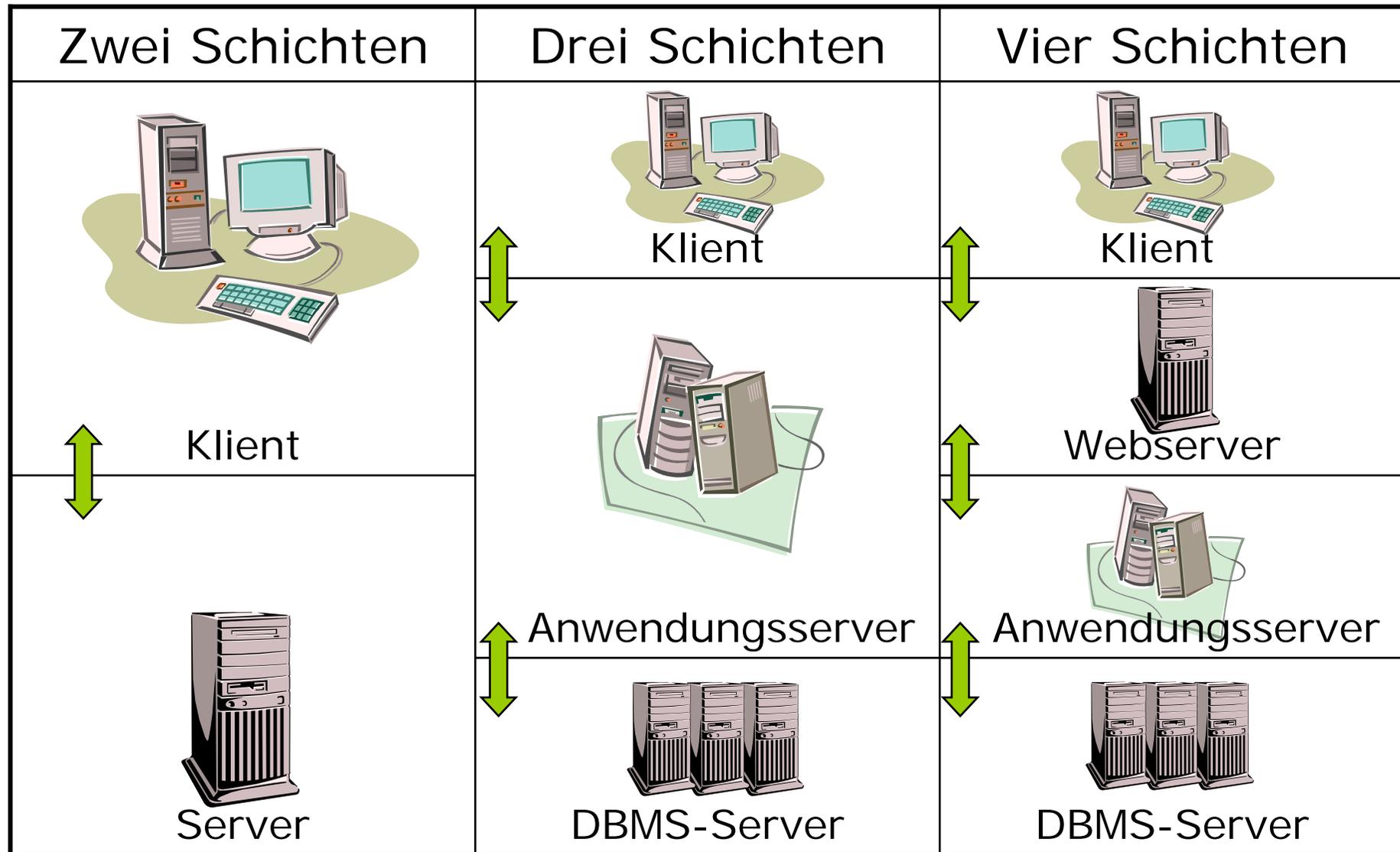


# Peer-to-Peer

- Jeder arbeitet mit jedem zusammen
- Jeder ist gleichberechtigt
- Middleware ist erheblich komplexer



# Schichtensicht (n-Tier)



# Gründe für verteilte Systeme

- Technisch:
  - Schlechte und unvorhersagbare Reaktionszeit zentralisierter Systeme, insbesondere in Interaktion mit Nutzer
  - Gute Verfügbarkeit von Vernetzung
  - Geringe Kosten der Vernetzung
    - > Hoher Nutzen der Integration von Systemen
- Organisatorisch:
  - Größere Integration von Organisationen
  - Virtuelle Organisationen mit „natürlicher“ Verteiltheit von Anwendungen
  - Ad-hoc Zusammenarbeit erlaubt keine zentralisierten Strukturen
- ...

- Systeme sind immer aus verschiedenen, oft unabhängigen Komponenten zusammengesetzt
  - vgl.: PC – Komponenten nicht unabhängig
- Räumliche Verteilung dieser Komponenten
  - Bedingt Vernetzung und Kommunikation
- Komponenten sind einzeln von Fehlern betroffen
  - Fehler und Auswirkungen sind für andere sichtbar
- Komponenten müssen integriert sein, so dass sie möglichst homogen erscheinen
  - Bedingt zusätzliche Software („Middleware“)
- Komponentenheterogenität teilweise transparent
  - Vereinfacht Systemgestaltung
- Komponenten als autonome Partner in Interaktionen
  - Bedingt Nebenläufigkeit

- Probleme/Herausforderungen/Fragen:
  - Interaktionsmodell
    - Wer initiiert eine Interaktion?
    - Wie findet ein Klient einen Server?
  - Skalierbarkeit
    - Wie realisiert man Systeme mit sehr hohen Zugriffszahlen?
    - Wie realisiert man sehr große Systeme mit sehr vielen Komponenten?
  - Entwicklung
    - Wie schreibt man einfach und korrekt Anwendungen dafür?
    - *Standardisierung*
  - Sicherheit
  - ...

# Vorteile Verteilter Systeme

- Bessere Leistung
  - Mittlerweile besseres Kosten-/Leistungsverhältnis durch
    - Kostengünstige einfache Komponenten (PCs)
    - Günstige Vernetzungs- und Kommunikationskosten
  - Bessere Antwortzeiten durch Lastverteilung
- Bessere Organisation
  - Entsprechung der Systemgestaltung und den Anforderungen verteilter und virtueller Organisationen
  - Gemeinsame Nutzung von Ressourcen (Leistung und Daten)
  - Kommunikation in System erlaubt Kommunikation zwischen Nutzern (Mail, CSCW)
- Bessere Zuverlässigkeit
  - Bessere Zuverlässigkeit durch verteilten Fehlerauftritt und mehrfach vorhandene Ressourcen und Kopien von Daten
  - Deshalb auch Gesamtausfall unwahrscheinlich
  - Inkrementeller Ausbau möglich, da flexibler Entwurf
  - Deshalb auch einfachere Wartung
- ...

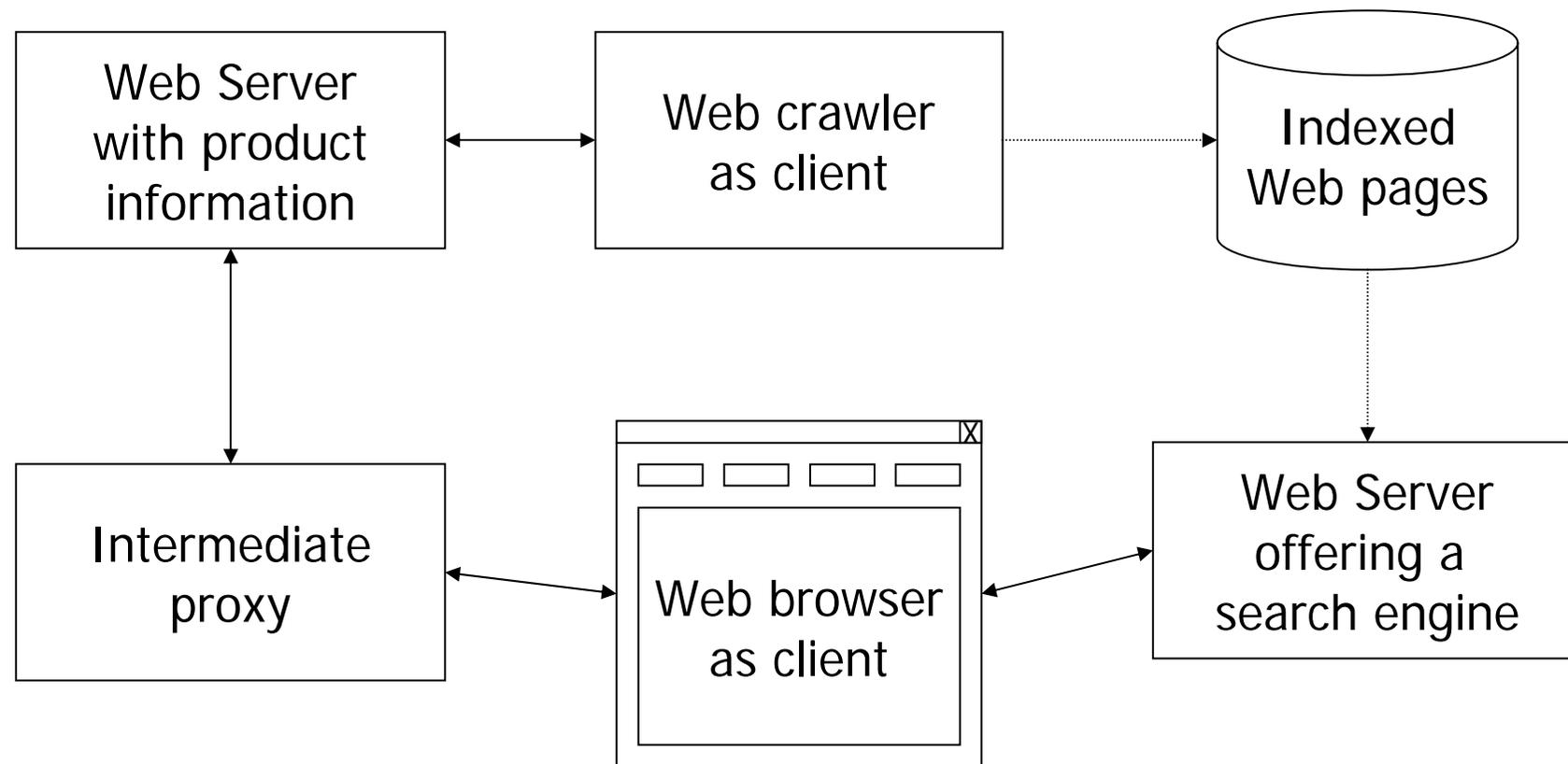
# Nachteile Verteilter Systeme

- Leistung:
  - abhängig von Netzwerkverfügbarkeit
  - abhängig von dessen Qualität (Latenz, Bandbreite etc.)
- Organisation
  - Höhere Komplexität der Software
  - Zusätzlich: Kommunikation und Koordination planen
  - Zusätzlicher Aufwand beim Testen durch nichtdeterministisches Netzverhalten
- Zuverlässigkeit
  - Erheblich mehr logische und physische Angriffspunkte
- ...

- „Die Vorlesung stellt Prinzipien, Sprachen und Middleware für die Entwicklung verteilter, insbesondere **Web-basierter** Anwendungssystemen dar“

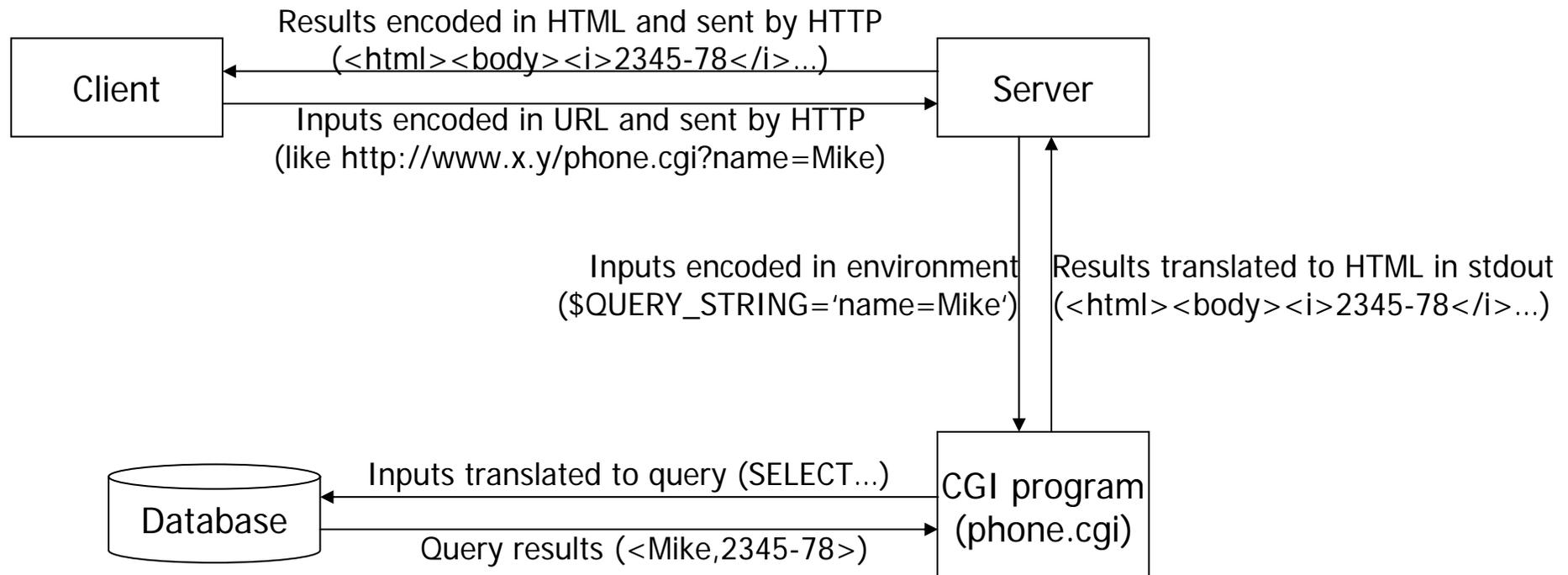
# Web als Verteiltes System

- Eine Web-basierte Anwendung ist verteiltes System
- Jede Komponente kann auf anderem Rechner sein:

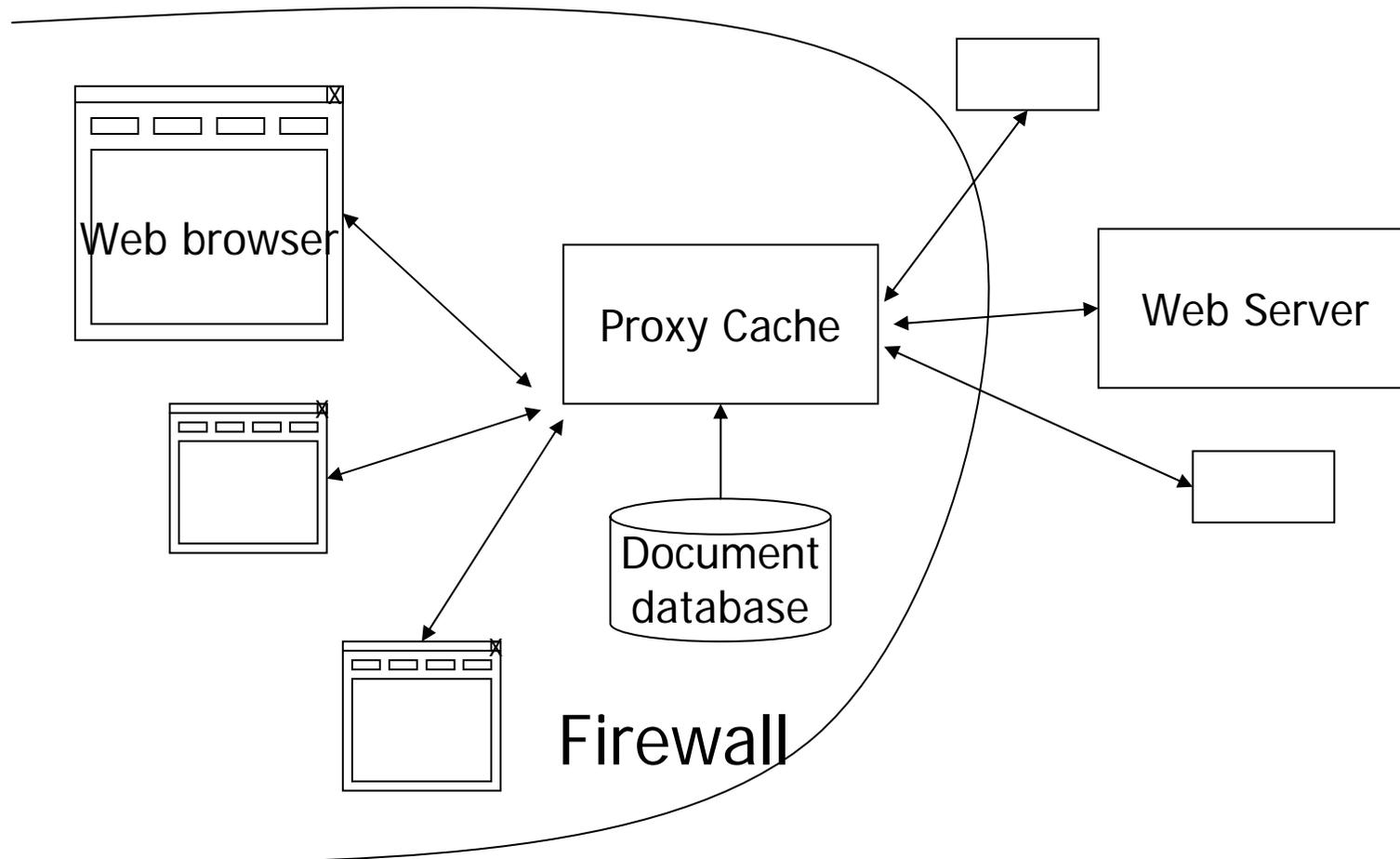


# Middleware für Web-basierte Anwendungen

- HTTP, HTML etc als Middleware
- Allerdings nicht hinreichend integriert



- Weite Bereiche der Web-Technologien beschäftigen sich mit Verteiltheit



- Richard M. Adler. Distributed Coordination Models for Client/Server Computing. IEEE Computer (Vol. 28, No. 4) April 1995. pp. 14-22.
- David E. Bakken. Middleware. Encyclopedia of Distributed Computing, Kluwer 2001.  
<http://www.eecs.wsu.edu/~bakken/middleware.pdf>
- ISO/IEC JTC1/SC21/WG7 ITU-T X.901 | ISO/IEC 10746-x. Basic Reference Model of Open Distributed Processing.
- Ted G. Lewis. Where Is Client/Server Software Headed? IEEE Computer (Vol. 28, No. 4) April 1995. pp. 49-55.
- Milan Milenkovic, Scott H. Robinson, Rob C. Knauerhase, David Barkai, Sharad Garg, Vijay Tewari, Todd A. Anderson, Mic Bowman. Toward Internet Distributed Computing. IEEE Computer (Vol. 36, No. 5) May 2003. pp. 38-46
- Johann Schlichter. Skript zur Vorlesung Distributed Applications. Institut für Informatik. TU München, März 2002  
[http://www11.informatik.tu-muenchen.de/lehre/lectures/ss2002/va/extension/latex/va\\_course\\_student.pdf](http://www11.informatik.tu-muenchen.de/lehre/lectures/ss2002/va/extension/latex/va_course_student.pdf)



## Vorlesungsüberblick

## Vorlesungsblöcke

- Organisation und Einführung
- Fernaufrufe
  - Remote Procedure Call, Konzepte und Grundelemente
  - RMI in Java, entfernter Objektaufruf
  - CORBA, sprachunabhängige Nutzung verteilter Objekte
- Nachrichtenbasierte Kommunikation
  - Sockets, Basis der Kommunikation im Internet
  - Internet Dienste
  - Internet Dienste in Java

- Web Programmierung Server- und Klientenseitig
  - HTTP-Kommunikation in Java (Zustand, Sicherheit etc.)
  - Serverseitige Ausführung: CGI, Servlets, SSI, JSP
  - HTML und Verarbeitung
  - XML, XML-Verarbeitung
  - Clientseitige Ausführung: Javascript, Applets
- Weitere Modelle
  - Koordinationssprachen, entkoppelter Kommunikation und Koordination
  - Peer-to-Peer, Aufhebung der Client/Server Rollen
  - Agenten, autonome netzbasierte Entitäten
- Klausur