

# Verteilte Systeme

Organisation

# Informatik



# Anforderungen

- ◆ Betriebssysteme (BS/BSP)  
da in diesen Umgebungen die Verteilten Systeme arbeiten
- ◆ Rechnernetze (RN/RNP)  
da dies die notwendigen Basisstrukturen sind auf/in denen Verteilte Systeme ablaufen
- ◆ Kenntnisse der Lehrveranstaltungen des 1-ten bis 4 -ten Semesters  
da Verteilte Systeme *die Informatik vereint*

# Vorlesungszyklus des Fachgebiets

Eigene  
Werke

Bachelor Projekt

WPs  
&  
POs

Peer-2-Peer Netzwerke

Webarchitekturen

Netzwerkmanagement

Aktuelle Internettechnol.

Verteilte Systeme

Grundlagen

Rechnernetze

Betriebssysteme



## Arbeitsgruppe Internet-Technologien

- Arbeitsgebiete:
  - IPv6, Mobiles und Future Internet, P2P Netze
  - Multimediakommunikation, Hypermedia, Web 2.0
- Sie finden uns in:
  - Raum 580 & Raum 780
- Messen & Ausstellungen:
  - CeBIT, LEARNTEC, NdW
- Auslandskooperationen
  - Wir vermitteln gerne



# Mitarbeit in aktuellen Projekten (1)

als studentischer Mitarbeiter, Bacheland, Masterand oder Doktorand ...

HVMcast

Industrie-Partner



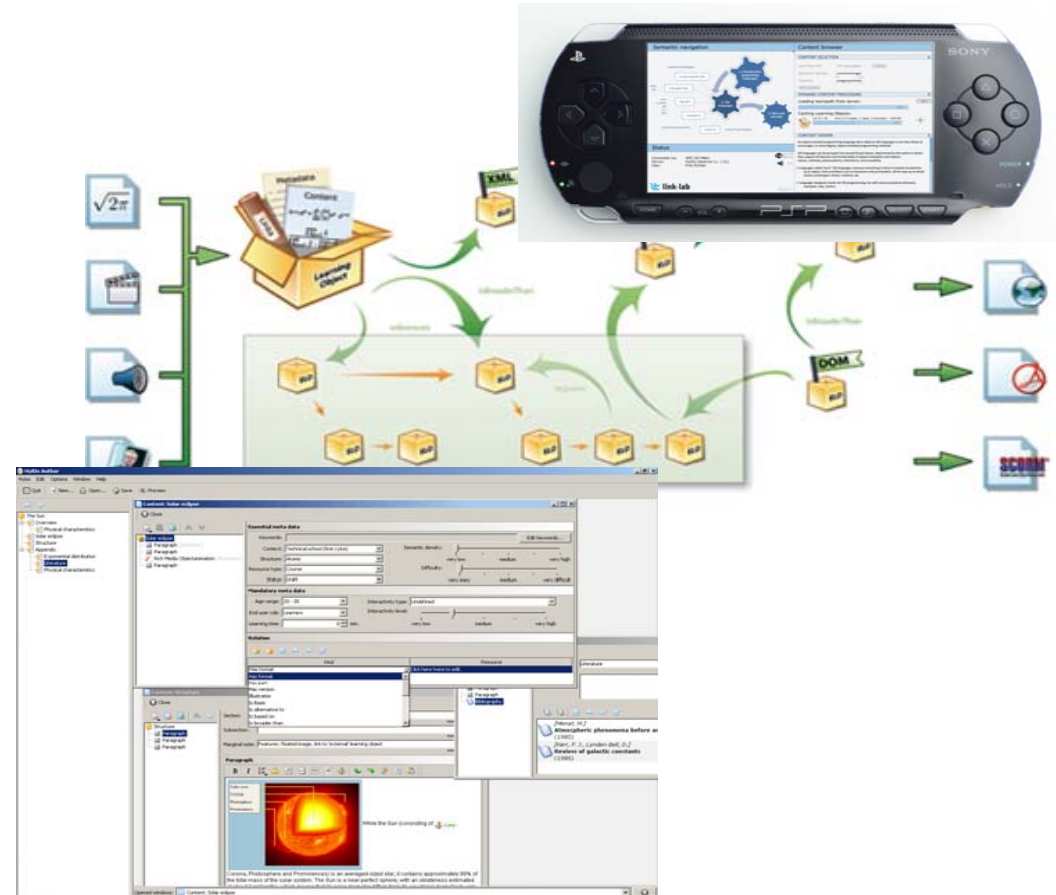
# Mitarbeit in aktuellen Projekten (2)

als studentischer Mitarbeiter, Bacheland, Masterand oder Doktorand ...

## Moviecast



## Mindstone / hyIOs



# Basis-Literatur

- G. Coulouris, J. Dollimore, T. Kindberg:  
*Verteilte Systeme*,  
Pearson Studium
- Andrew S. Tanenbaum, Maarten van Steen:  
*Verteilte Systeme: Grundlagen und Paradigmen*,  
Pearson Studium
- U. Hammerschall: *Verteilte Systeme und Anwendungen*, Pearson Studium
- Claudia Eckert: *IT-Sicherheit*, Oldenbourg
- N. A. Lynch: *Distributed Algorithms*, Morgan Kaufmann
- U. Lang, R. Schreiner: *Developing Secure Distributed Systems with CORBA*, Artech House
- A. S. Tanenbaum: *Moderne Betriebssysteme*, Hanser Verlag
- A. S. Tanenbaum: *Computernetzwerke*, Prentice Hall
- D. Baum, M. Gasperi, R. Hempel, L. Villa: *Extreme Mindstorms: An Advanced Guide to Lego Mindstorms*, Apress Verlag



Zum Kauf  
empfohlen!



# Praktikum / PVL

- ◆ 4 Praktikumsaufgaben
  - RMI mittels Corba: message of the day
  - Verteilte Algorithmen: Konzeption und Realisierung
  - 2mal Lego Mindstroms
- ◆ Gruppenaufteilung: **2 Studierende** in einer Gruppe
- ◆ **PVL-Bedingungen**
  - **Baut stark auf **vorbereitender Arbeit** auf!**
  - **Details** siehe Aufgabenstellungen!
    - ◆ Bis **Freitag Abend vor** dem Praktikumstermin:  
Entwurf an [abgabe\\_tivsp@informatik.haw-hamburg.de](mailto:abgabe_tivsp@informatik.haw-hamburg.de) senden.
    - ◆ Am **Anfang des Praktikumstermins**: Konzeptgespräch
    - ◆ Am **Ende des Praktikumstermins**: Ergebnisbesprechung
    - ◆ Bis **Freitag der selben Woche**: Abgabe des Codes
  - Anwesenheitspflicht (gesamte Praktikumszeit!)
  - Erfolgreiche Bearbeitung **aller** Aufgaben: siehe Aufgabenstellungen

# Mündliche Prüfung

- ◆ Finden in der 2. Klausurwoche statt (1. – 5. Februar '10)
- ◆ Erste Anmeldung wie zu den Klausuren
- ◆ Zweite Anmeldung für die (**verbindlichen!**) Prüfungstermine
- ◆ Prüfungszeit: ca. 30 Minuten
- ◆ Bewertung: Bekanntgabe am Ende der Prüfung
- ◆ Ablauf:
  - Nennung eines Themas, mit dem **begonnen** werden soll.
  - Weiteres Prüfungsgespräch umfasst alle Vorlesungsthemen

## Hinweis zu den Folien

- ◆ Die Folien sind **kein vollständiges Skript** und genügen normalerweise nicht zur Prüfungsvorbereitung oder als Nachschlagewerk!
- ◆ Sie sollten sich deshalb auf jeden Fall zumindest mit der aufgeführten **Basis-Literatur beschäftigen**, insbesondere mit den zum Kauf empfohlenen Büchern, und sich von Zeit zu Zeit auch **weiterführende Literatur** und aktuelle **Zeitschriftenartikel anschauen**.
- ◆ Bemerkung am Rande: Diese Folien sind zum großen Teil aus Folien anderer Kollegen (auch anderer Hochschulen) zusammengestellt!

# Verteilte Systeme

Einführung

# Inhalt der Vorlesung

1. Einführung und Systemmodelle
2. Interprozesskommunikation (IP, Client/Server, ..., RMI)
3. Namensdienste und Internet Standardanwendungen
4. Zeit, Synchronisation und globale Systemzustände
5. Übereinstimmung und Koordination
6. Peer-to-Peer Systeme
7. Verteilte Transaktionen
8. Replikation
9. Sicherheit in verteilten Systemen

## Warum bilden „**Verteilte Systeme**“ ein **eigenständiges Thema** ?

- ◆ Es gibt **keinen gemeinsamen Speicher**  
(Interaktion durch Nachrichtenaustausch)
- ◆ Es gibt **nebenläufige/parallele Aktivitäten**  
(Koordination, Synchronisation)
- ◆ Fehler und **Ausfälle sind wahrscheinlich**  
(Transparenz)
- ◆ **Komponenten** (Hardware und Software) sind **heterogen**  
(Standardisierung von Schnittstellen)
- ◆ **Systeme** können **sehr groß** sein  
(Großsystemeffekte, Umschlag von der Quantität in die Qualität)

## Wozu braucht man ein „**Verteiltes System**“?

- ◆ **Kommunikationsverbund** (Übertragung von Daten, insbesondere Nachrichten, an verschiedene, räumlich getrennte Stellen; z.B. E-Mail)
- ◆ **Informationsverbund** (Verbreiten von Information an interessierte Personen/Systeme; z.B. WWW)
- ◆ **Datenverbund** (Speicherung von Daten an verschiedenen Stellen: bessere Speicherauslastung, erhöhte Verfügbarkeit, erhöhte Sicherheit)
- ◆ **Lastverbund** (Aufteilung stoßweise anfallender Lasten auf verschiedene Rechner: gleichmäßige Auslastung verschiedener Ressourcen)
- ◆ **Leistungsverbund** (Aufteilung einer Aufgabe in Teilaufgaben: Verringerte Antwortzeiten)
- ◆ **Wartungsverbund** (Zentrale Störungserkennung und –behebung: schnellere und billigere Wartung verschiedener Rechner)
- ◆ **Funktionsverbund** (Verteilung spezieller Aufgaben auf spezielle Rechner; Bereitstellung verschiedener Funktionen an verschiedenen Orten)
- ◆ **Kapazitätsverbund** (Ausnutzung sämtlicher zur Verfügung stehender Rechenkapazität)

# Verteilte Welt und Probleme



- Viele gleichzeitige („parallele“) Aktivitäten
- Exakte globale Zeit nicht erfahrbar/vorhanden
- Keine konsistente Sicht des Gesamtzustandes
- Kooperation durch Kommunikation
- *Ursache* und *Wirkung* zeitlich getrennt

> Räumliche Separation,  
autonome Komponenten

> Heterogenität

> Dynamik, Offenheit

> Komplexität

> Sicherheit

+ Probleme sequentieller  
Systeme

+ Nebenläufigkeit

+ Nichtdeterminismus

– Synchronisation  
schwieriger

– Programmierung  
komplexer

(Threads, kritische Abschnitte, etc.) gelernt. Insbesondere habe ich erkannt, dass das Debugging und die Fehlersuche in solchen Systemen ungleich schwerer als in anderen Systemen ist. Das liegt sicher zum einen daran, dass es auch deutlich mehr Fehlermöglichkeiten gibt. Hinzu kam aber noch meine Unerfahrenheit in diesem Bereich



# Was ist ein verteiltes System ?

- ◆ Definition: Verteiltes System (nach *Leslie Lamport*)

*Ein verteiltes System ist ein System, mit dem ich nicht arbeiten kann, weil irgendein Rechner abgestürzt ist, von dem ich nicht einmal weiß, daß es ihn überhaupt gibt.*

- oft die Realität
- wird aber besser (hoffentlich auch durch diese Vorlesung)

- ◆ Definition: Verteiltes System (nach *Andrew S. Tanenbaum*)

*Ein verteiltes System ist eine Kollektion unabhängiger Computer, die den Benutzern als ein Einzelcomputer erscheinen.*

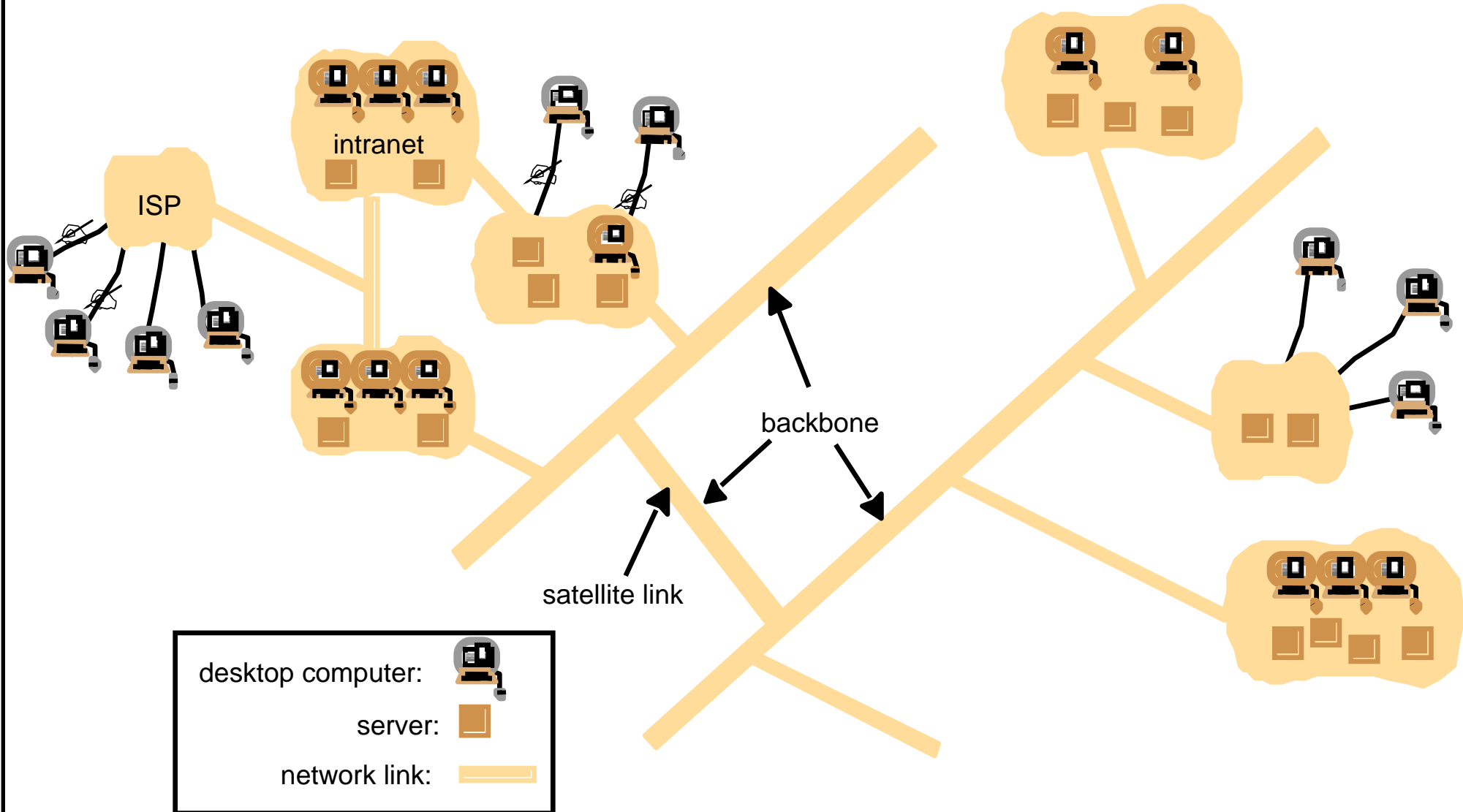
- impliziert, daß die Computer miteinander verbunden sind und
- die Ressourcen wie Hardware, Software und Daten gemeinsam benutzt werden.
- Es herrscht eine einheitliche Sicht auf das Gesamtsystem vor.

# Was ist ein verteiltes System ?

Eine *allgemeinere* Beschreibung:

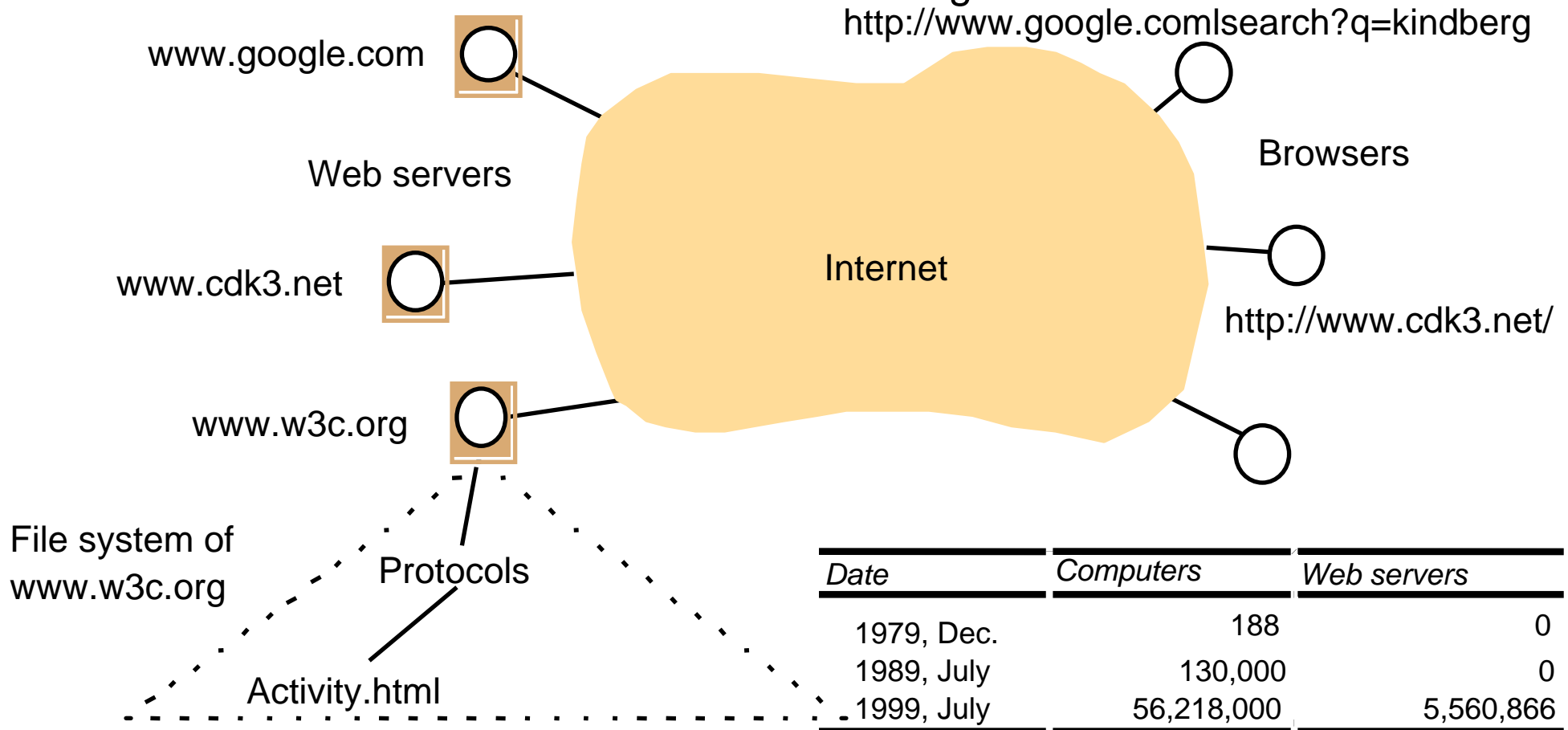
- ◆ Ein **verteiltes System** ist ein System, in dem
  - Hard-und Softwarekomponenten,
  - die sich auf miteinander vernetzten Computern befinden,
  - miteinander kommunizieren und ihre Aktionen koordinieren,
  - indem sie Nachrichten austauschen.
  
- ◆ Eine **verteilte Anwendung** ist eine Anwendung, die ein verteiltes System zur Lösung eines Anwendungsproblems nutzt. Sie besteht aus verschiedenen Komponenten, die mit den Komponenten des VS sowie den Anwendern kommuniziert.

# Beispiel Nr1: Das Internet

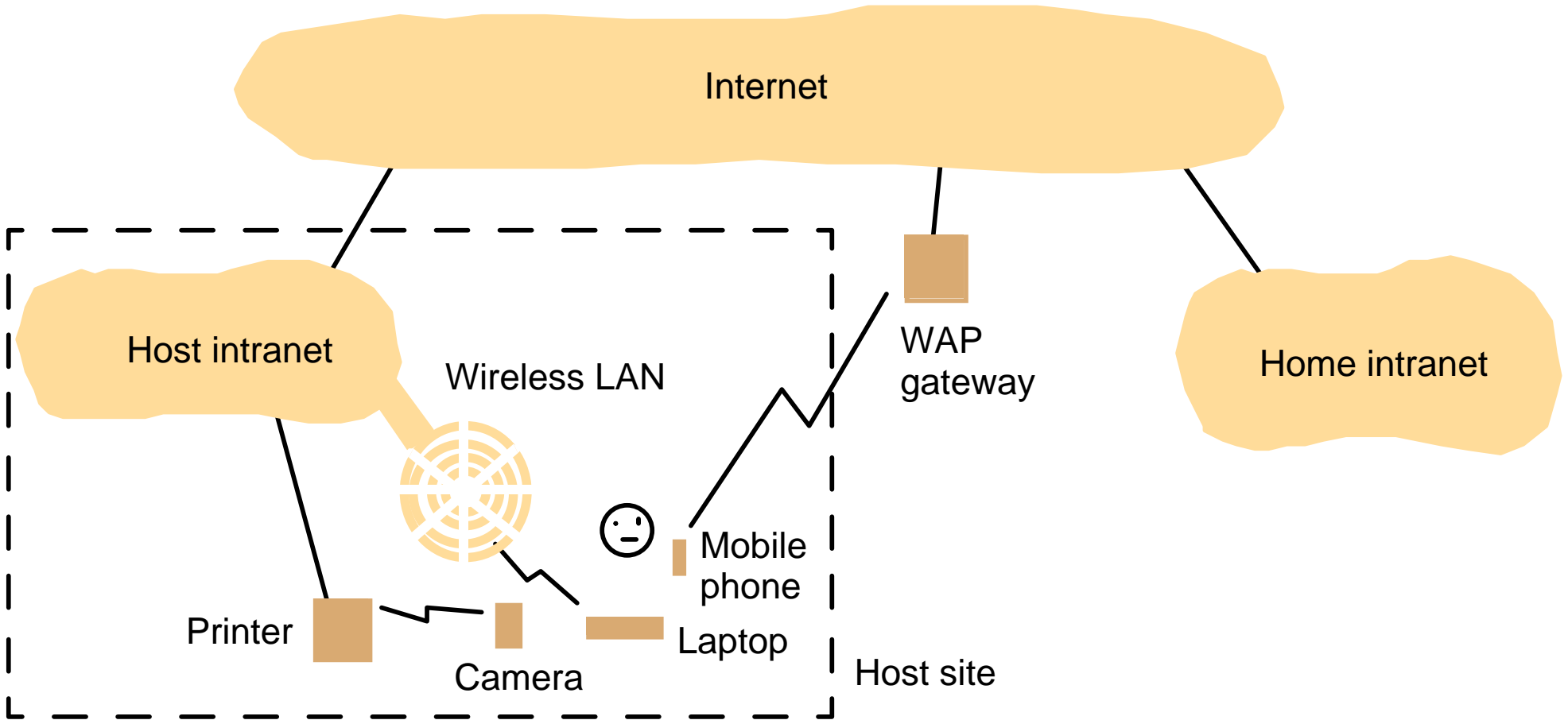


# Beispiel Nr2: Das World Wide Web

- Sicherlich die populärste verteilte Anwendung.
- Basiert auf dem Internet als verteiltes System.
- Interessant: WWW als Basis für neue Anwendungen.

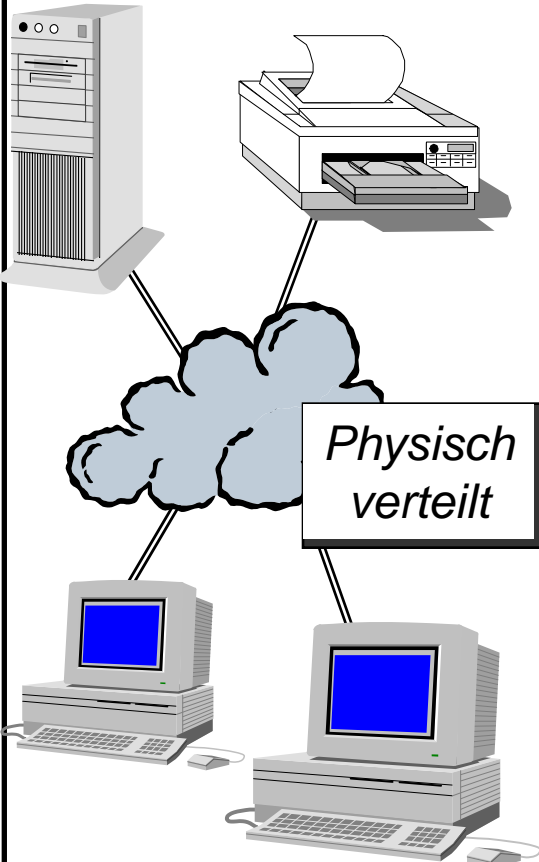


# Beispiel Nr3: mobile computing

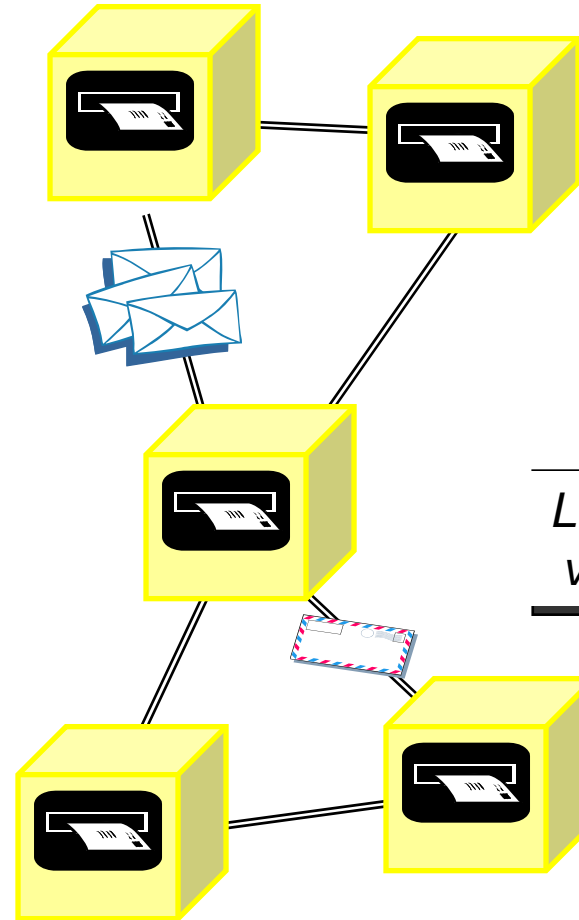


# Sichten verteilter Systeme

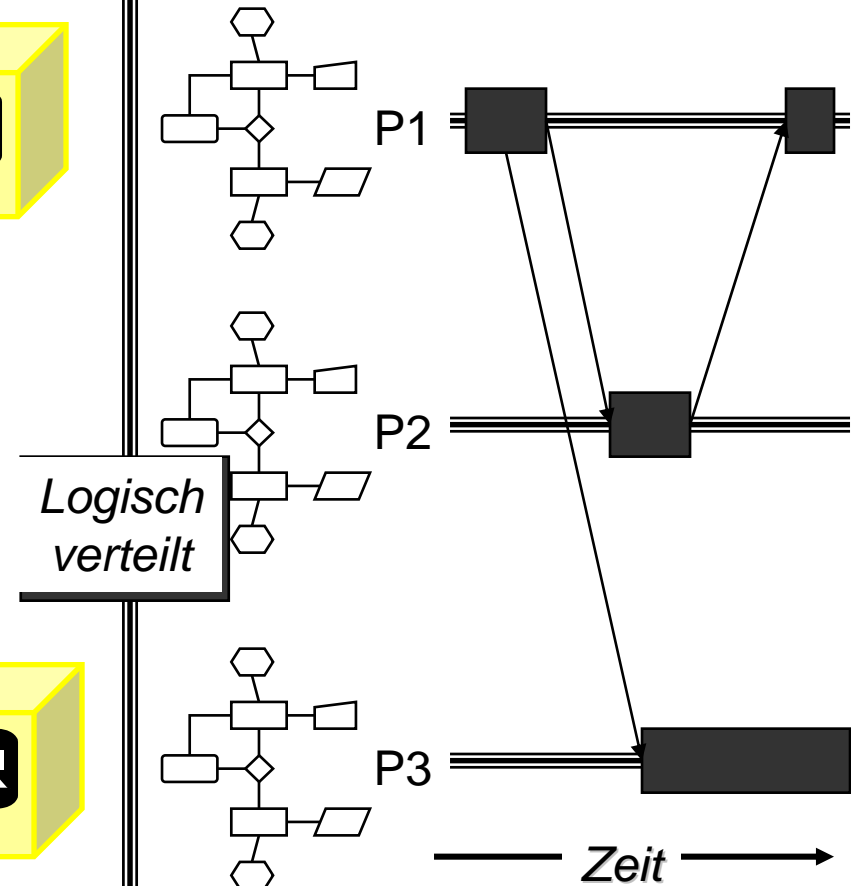
Rechnernetz mit  
Rechnerknoten



Objekte



Algorithmen u.  
Protokolle



Logisch  
verteilt

# Wünschenswerte Eigenschaften

- ◆ **Gemeinsame Ressourcennutzung:** Hardware, Daten, Dienste etc. gemeinsam nutzen
- ◆ **Offenheit:** Schlüsselschnittstellen (einheitlich) offen legen
- ◆ **Nebenläufigkeit:** Mehrere gleichzeitig existierende Prozesse
- ◆ **Skalierbarkeit:** auch mit vielen Komponenten gut funktionieren können
- ◆ **Sicherheit:** Verfügbarkeit, Vertraulichkeit, Integrität, Authentizität, etc
- ◆ **Fehlertoleranz:** Fehler erkennen, maskieren, tolerieren
- ◆ **Transparenz:** hier im Sinne, **etwas nicht sehen** bzw. durch etwas hindurch sehen können

# Transparenz

**Transparenz** wird definiert als das Verbergen der Separation der einzelnen Komponenten in einem verteilten System vor dem Benutzer und dem Applikationsprogrammierer, so dass das System als Ganzes wahrgenommen wird, und nicht als Sammlung voneinander unabhängiger Komponenten.

ISO (International Standards Organization) und ANSA (Advanced Network Systems Architecture) identifizieren **acht Formen** der Transparenz:

1. **Zugriffstransparenz** ermöglicht den Zugriff auf lokale und entfernte Ressourcen unter Verwendung identischer Operationen.
2. **Positionstransparenz** (Ortstransparenz) erlaubt den Zugriff auf die Ressourcen, ohne dass man ihre Position/ihren Ort kennen muss.
3. **Nebenläufigkeitstransparenz** erlaubt, dass mehrere Prozesse gleichzeitig mit denselben gemeinsam genutzten Ressourcen arbeiten, ohne sich gegenseitig zu stören.



# Transparenz

4. **Replikationstransparenz** erlaubt, dass mehrere Instanzen von Ressourcen verwendet werden, um die Zuverlässigkeit und die Leistung zu verbessern, ohne dass die Benutzer oder Applikationsprogrammierer wissen, dass Repliken verwendet werden.
5. **Fehlertransparenz** erlaubt das Verbergen von Fehlern, so dass Benutzer und Applikationsprogrammierer ihre Aufgaben erledigen können, auch wenn Hardware- oder Softwarekomponenten ausgefallen sind.
6. **Mobilitätstransparenz** erlaubt das Verschieben von Ressourcen und Clients innerhalb eines Systems, ohne dass die Arbeit von Benutzern oder Programmen dadurch beeinträchtigt wird.
7. **Leistungstransparenz** erlaubt, dass das System neu konfiguriert wird, um die Leistung zu verbessern, wenn die Last variiert.
8. **Skalierungstransparenz** erlaubt, dass sich System und Applikationen vergrößern, ohne dass die Systemstruktur oder die Applikationsalgorithmen geändert werden müssen.

# Verteilte Systeme

## Systemmodelle

# Systemmodelle

- ◆ Beschreibung der allgemeinen Eigenschaften und des Designs eines Systems
- ◆ Das **Modell** sollte abdecken:
  - Die wichtigsten Komponenten des Systems
  - Die Art ihrer Interaktion
  - Wie deren individuelles und kollektives Verhalten beeinflusst werden kann
- ◆ Ein **Architekturmodell**
  - *vereinfacht und abstrahiert* zunächst die Funktionen der individuellen Komponenten eines verteilten Systems, um dann
  - die *Verteilung* der Komponenten auf ein Netzwerk von Computern und
  - die *Beziehung* der Komponenten (Rolle in der Kommunikation mit anderen, Kommunikationsmuster) untereinander zu beschreiben.
- ◆ Weitere Modelle: Interaktionsmodell, Fehlermodell, Sicherheitsmodell

# Hardware- und Software-Serviceschichten

- Plattformunabhängig
- Middlewareabhängig

Applikationen, Dienste

Middleware

Betriebssystem

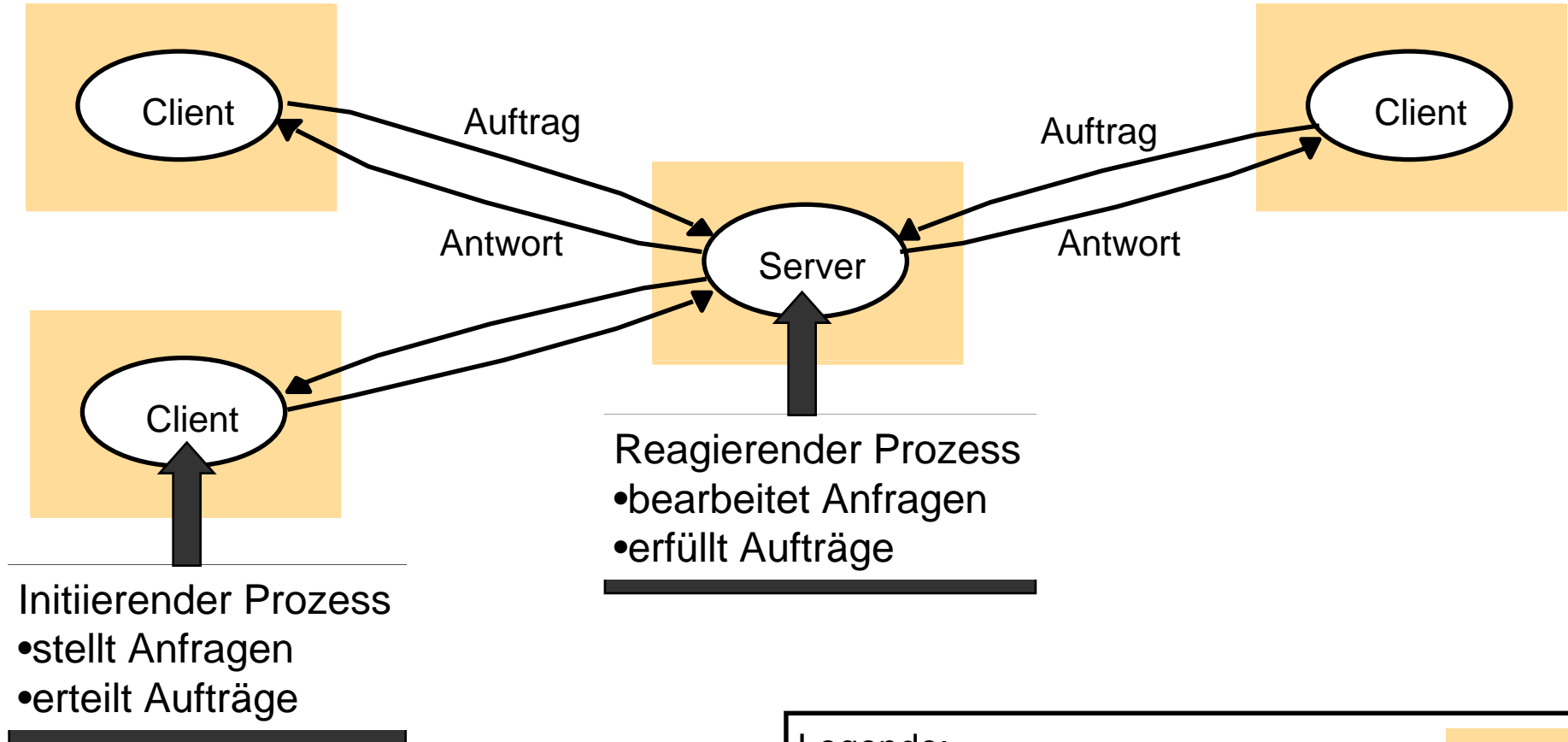
Computer- und Netzwerkhardware

**Middleware** (Verteilungsplattform) :  
Transparenz der

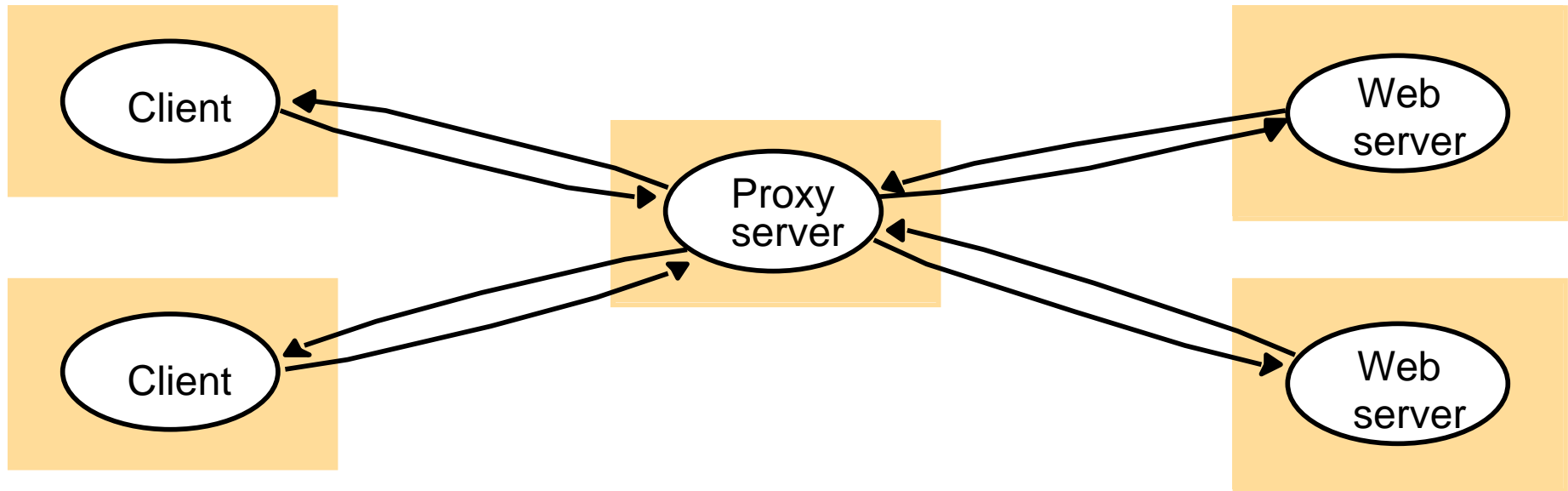
- *Heterogenität* existierender Hardware und Betriebssysteme
- *Verteilung*

**Plattform:** „unterste“ Hardware- und Softwareschichten (Low-Level) werden häufig als *Plattform* bezeichnet.  
*Beispiele:* Intel x86/{Windows|Linux}, PowerPC/MacOS, SunSPARC/SunOS

# Client/Server Modell



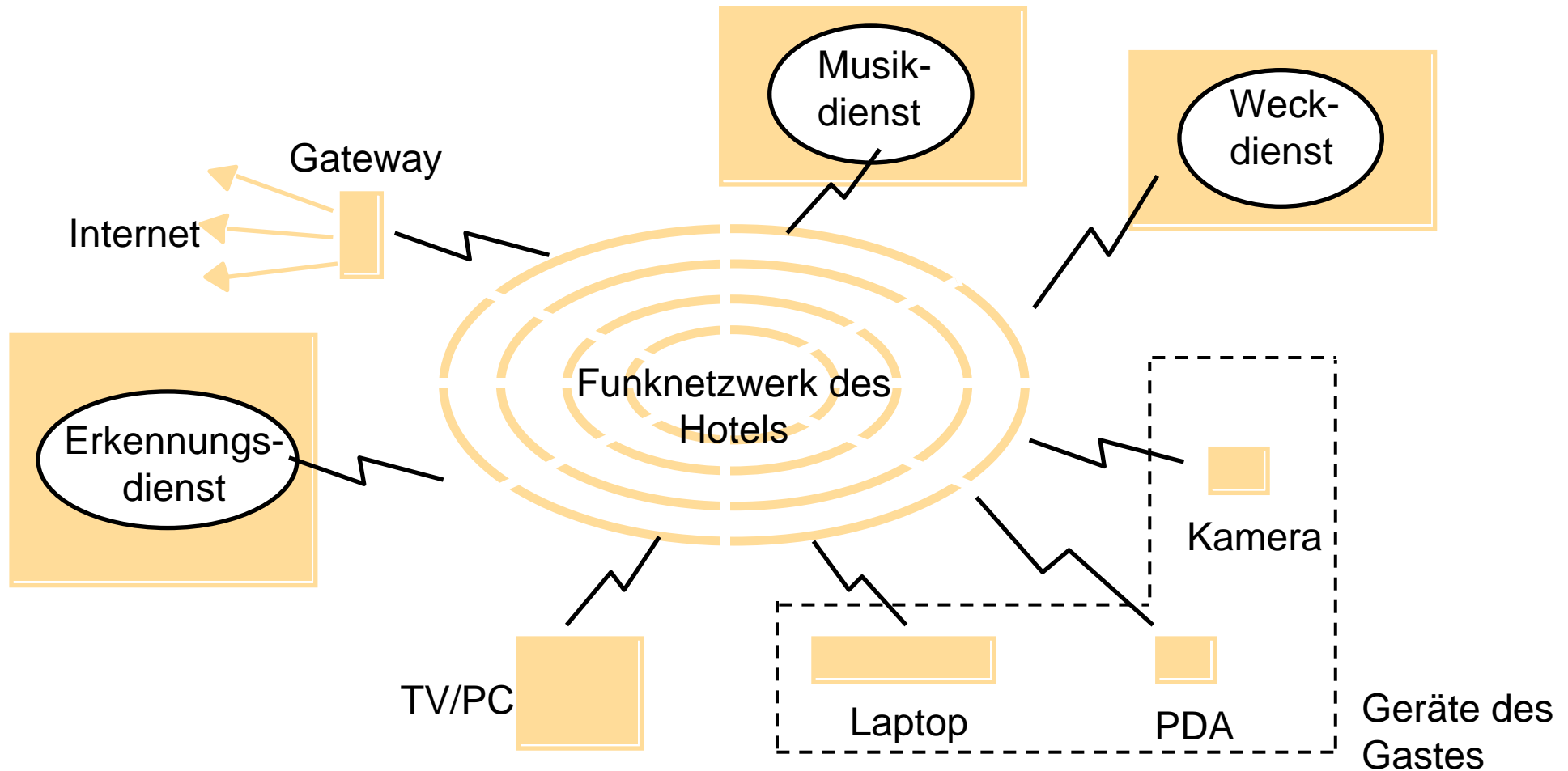
# Proxy-Server und Cache



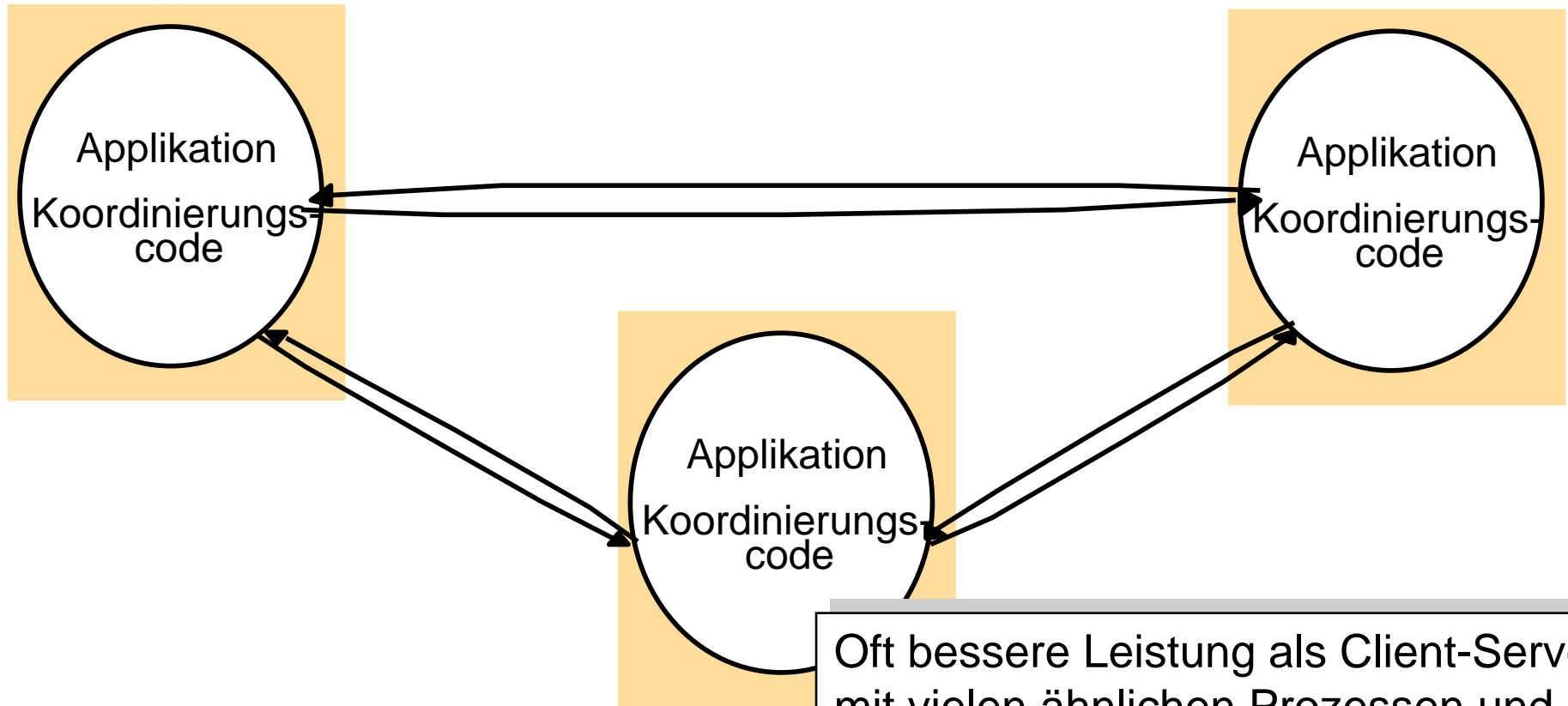
**Proxy-Server:** Gemeinsamer Cache

Zweck von Proxy-Servern: erhöhte Leistung und Verfügbarkeit

# Spontane Netzwerkverbindungen



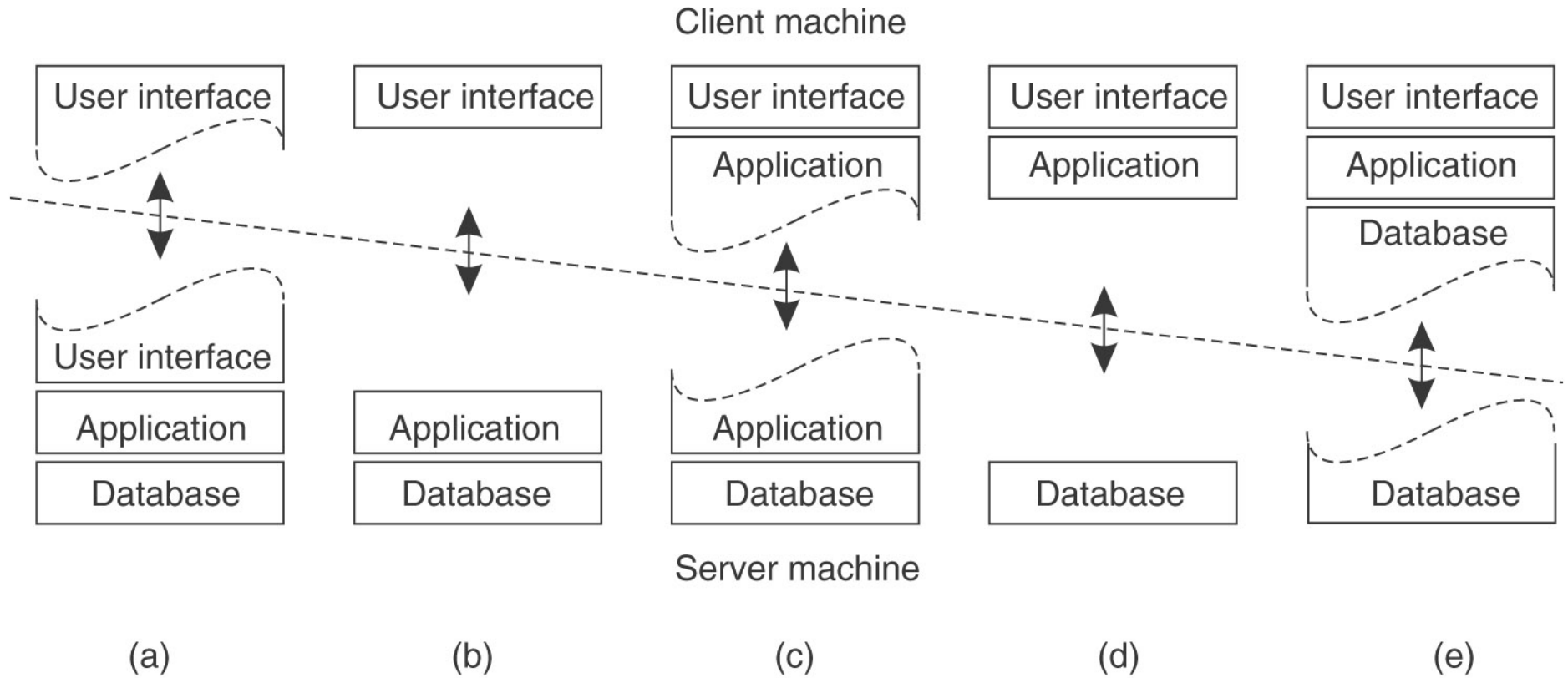
# Gleichrangige Prozesse (Peer Processes)



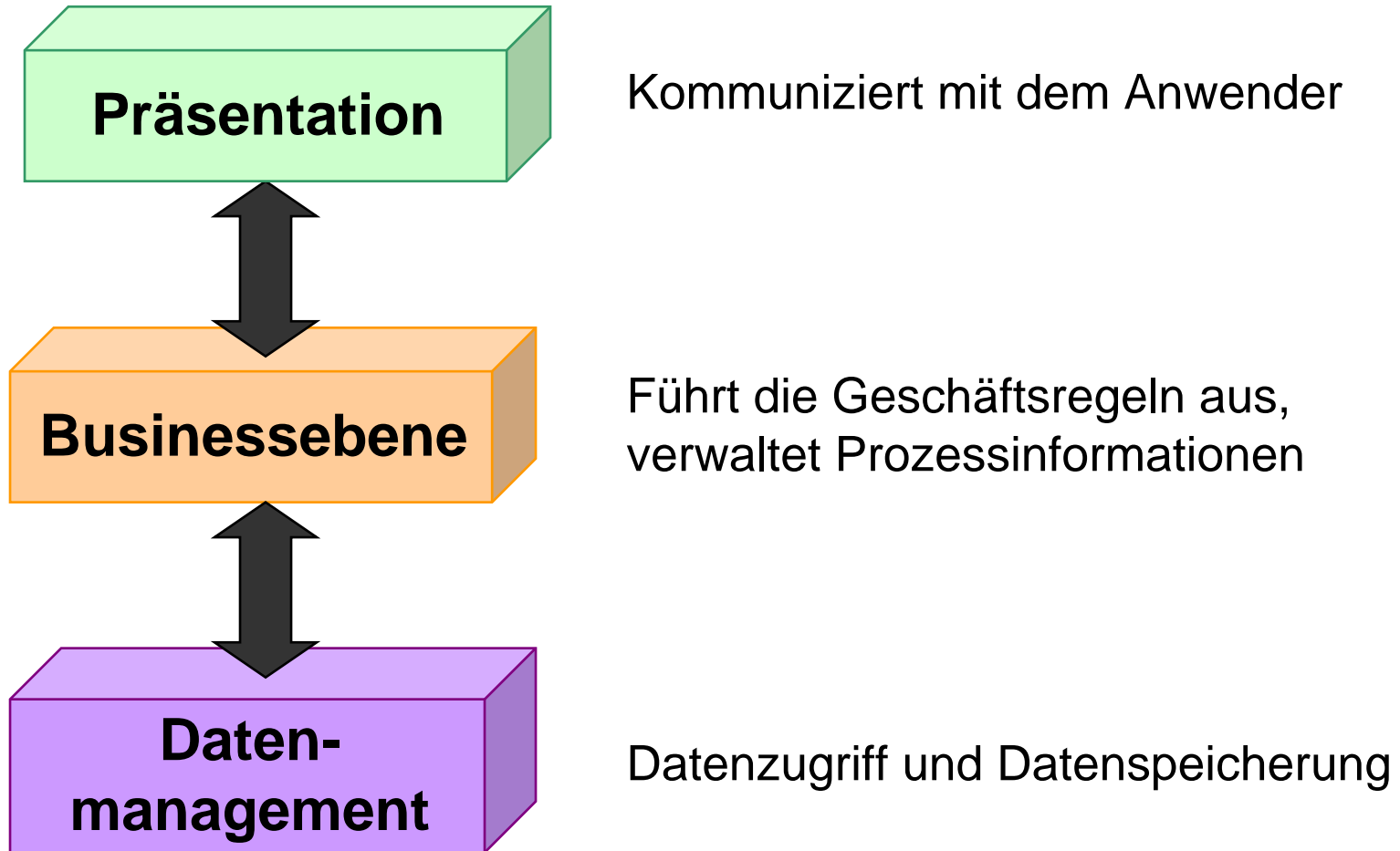
Oft bessere Leistung als Client-Server mit vielen ähnlichen Prozessen und vorwiegend lokaler Kommunikation. Beispiel: Filesharing, Conferencing



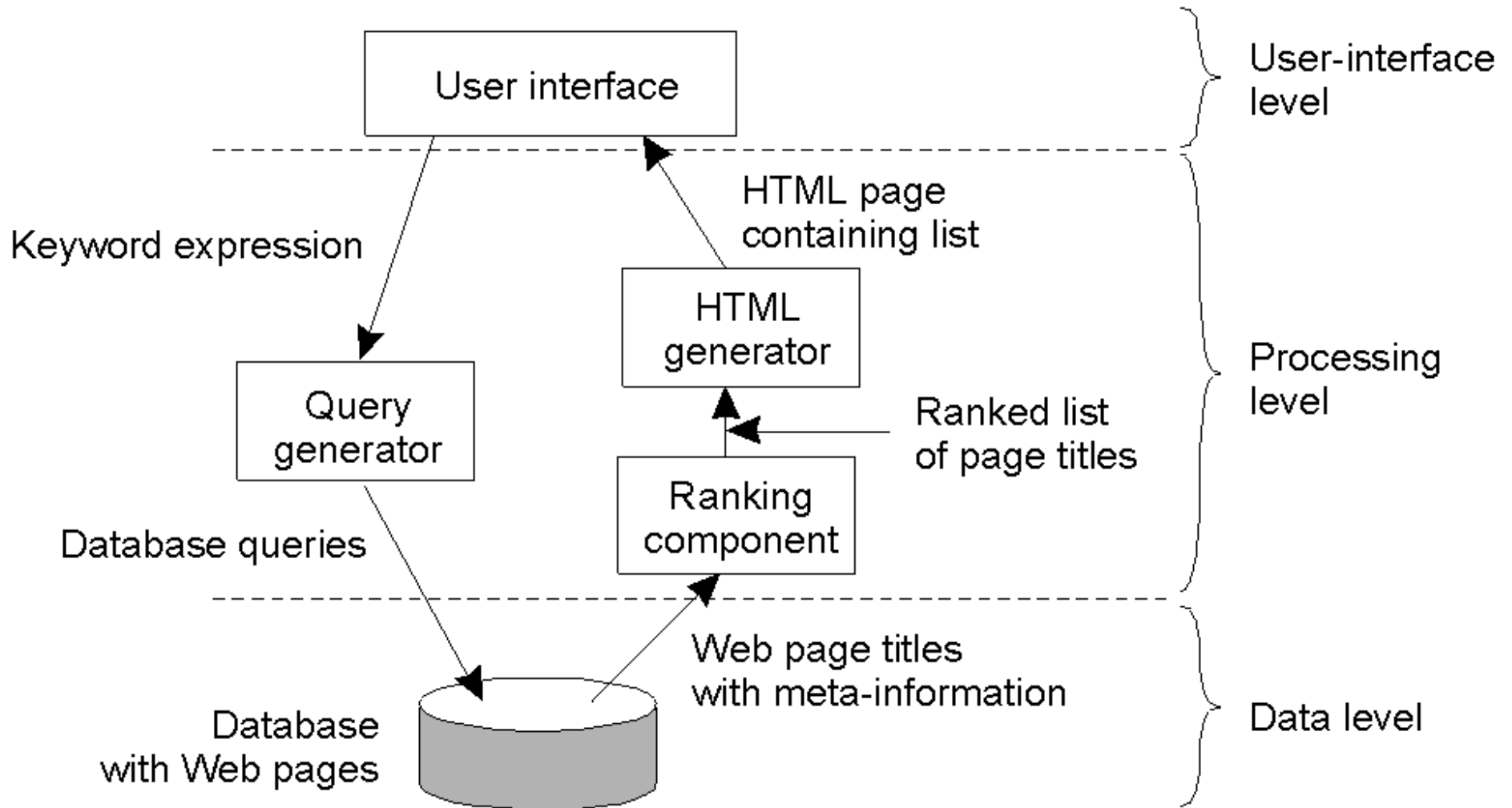
# Thin Client/Fat Server



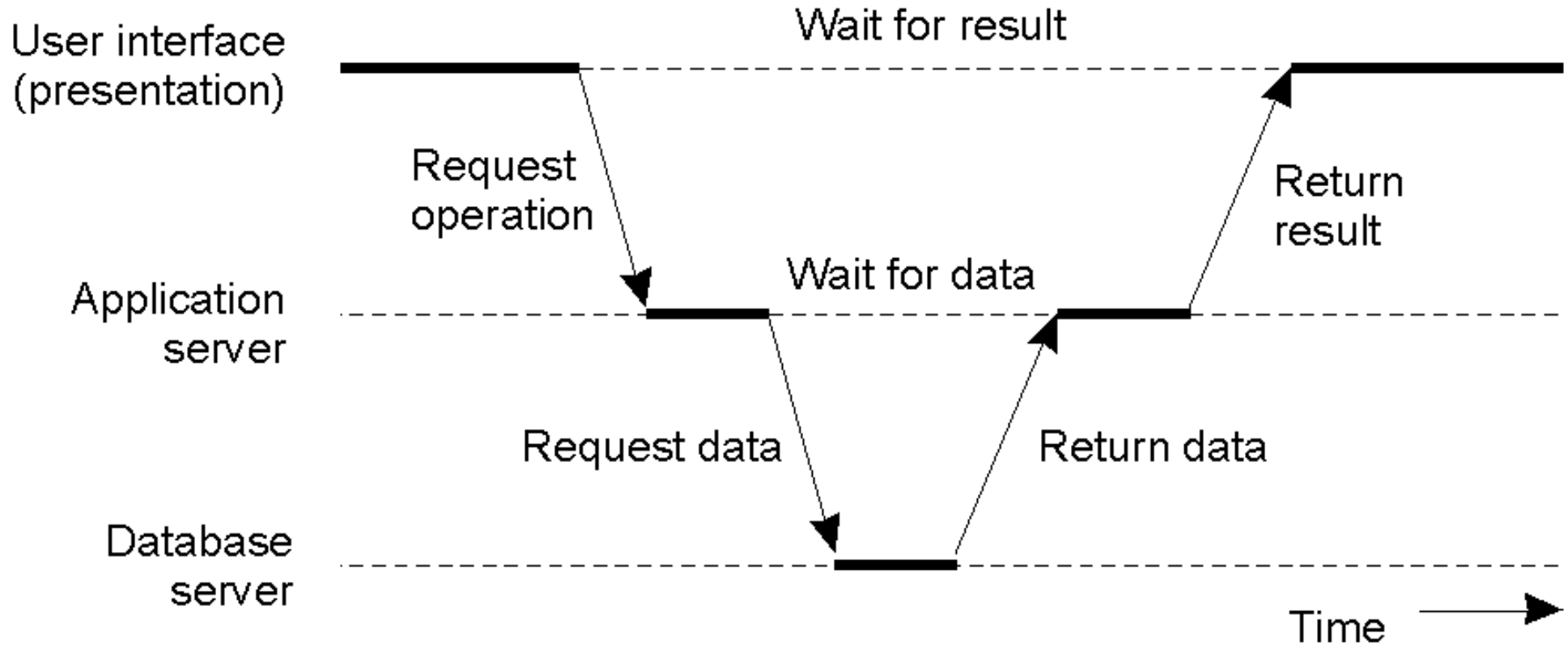
## Drei Ebenen Architektur: „*three-tier*“



# Beispiel: Suchmaschine



# Beispiel: Suchmaschine



# Anforderungen an die Auswahl eines Modells

- ◆ Welches sind die **Einflussfaktoren**
  - bei der **Auswahl** des Modells
  - bei der **Platzierung der Komponenten**
- ◆ **Wichtig** u.a.
  - Performance (Antwortverhalten, Durchsatz, Lastbalancierung, etc.)
  - Quality of Service (Zuverlässigkeit, Sicherheit, Echtzeit, etc.)
  - Inwieweit sollen Caching und Replikation eingesetzt werden (mit welchem Konsistenzmodell?)
  - Benötigter Grad der Verlässlichkeit des Systems (Korrektheit, Sicherheit, Fehlertoleranz, etc.)
  - Kosten (Anschaffung, Schulung, Lizenzen, etc.)