Verteilte Systeme

Teil 1: Einführung

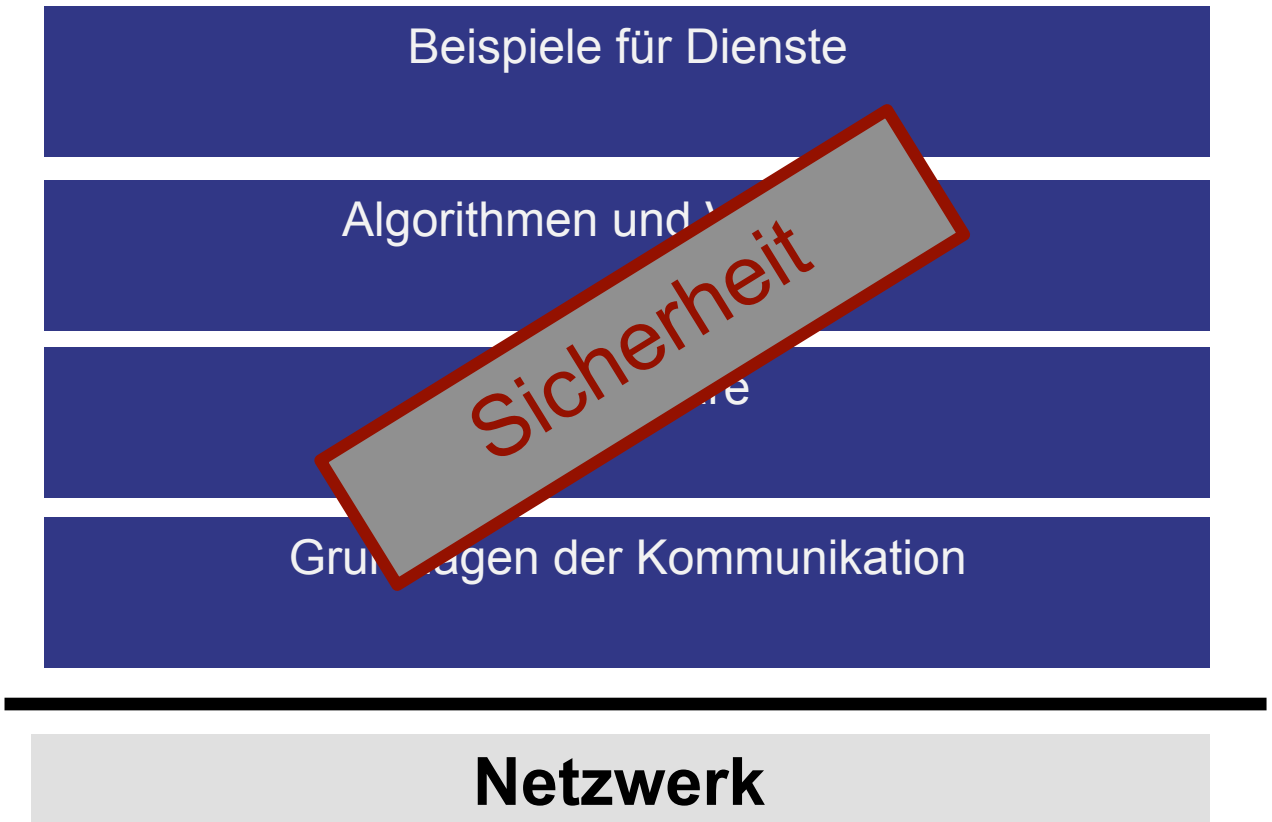
Inhalte der Vorlesung

- Einführung
- Netzwerkkommunikation
- Middleware
- Algorithmen und Verfahren
  - Synchronisation
  - Skalierbarkeit
  - Konsistenz / Replikation
  - Fehlerbehandlung
- Dienstbeispiele
  - Verteilte Dateisysteme
- Sicherheit

Inhalte der Vorlesung

- Einführung
- Netzwerkkommunikation
- Middleware
- Algorithmen und Verfahren
  - Synchronisation
  - Skalierbarkeit
  - Konsistenz / Replikation
  - Fehlerbehandlung
- Dienstbeispiele
  - Verteilte Dateisysteme
- Sicherheit

Aufbau der Vorlesung



Teil 1: Einführung

- Definition "Verteiltes System"
- Beispiele
- Qualitätsmerkmale
- Architekturen
  - Hardware
  - Software
  - System
- Grundannahmen über die Umgebung eines verteilten Systems
- Übersicht über weitere Themen der Vorlesung

Was ist ein verteiltes System?

- Beispiele ?
  - Multicore?
  - in der *Cloud* gemietete Rechner?
  - email ?
  - Webclient und –server?
- Problem: Fast alles fällt in diese Kategorie, ist irgendwie verteilt
  - von Multicore bis zu Großrechnern und Cloud-Diensten

Wie schränkt man ein?

- Definition (Tanenbaum, van Steen)
  - "Ein verteiltes System ist eine Ansammlung *unabhängiger* Computer, die den Benutzern wie ein *einzelnes kohärentes* System erscheinen."
- Wir bewegen uns in dieser Vorlesung innerhalb dieser Definition

- Frage: Welche der eben genannten Beispiele fallen nicht unter diese Definition?

Wie schränkt man ein?

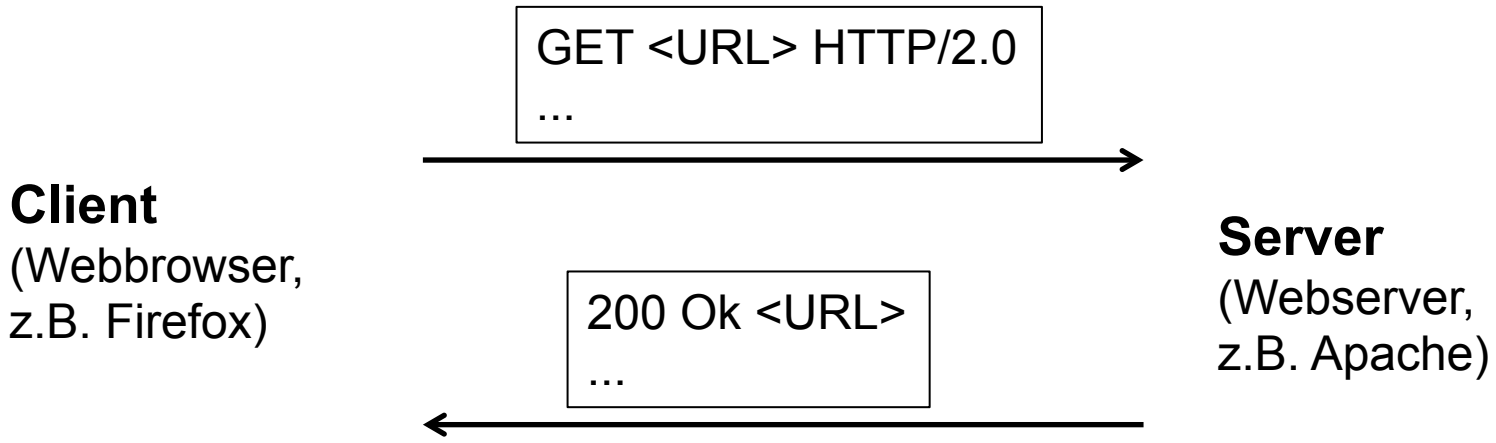
- Wichtige Eigenschaften der hier betrachteten Systeme
  - Kohärenz (Zusammenhang des Gesamtsystems)
  - Unabhängige Computer
- Daraus folgt: Multicore Rechner und Systeme mit gemeinsamem Speicher (z.B. große Parallelrechner) werden hier nicht betrachtet
- Daraus folgt weiterhin: die Verteilung sollte nicht sichtbar für Benutzer sein (verborgen, transparent)

Noch ein paar Stichworte

- Internet
  - "Internet der Dinge"
  - Praktisch alle *Cloud* Dienste: Facebook, Twitter, Google, ...
    - Frage: Warum werden Cloud Dienste in aller Regel von verteilten Systemen bereitgestellt?
  - *Grid* Computing (wissenschaftliches Rechnen auf weltweit verteilten Rechnern, z.B. Ergebnisverarbeitung von LHC Experimenten)

Beispiel: WWW

- Client und Server bilden ein verteiltes System
- Basiert auf HTTP



Beispiel: Google

- Google betreibt weltweit Rechenzentren mit 100000en von Rechnern (ähnlich Facebook, Twitter, Microsoft, ...)
- Riesiges verteiltes System mit verschiedenen einzelnen *Diensten*
  - Webserver
  - Load balancer
  - Indexserver
  - Dokumentenserver
  - Adserver
  - Spellchecker
  - ...

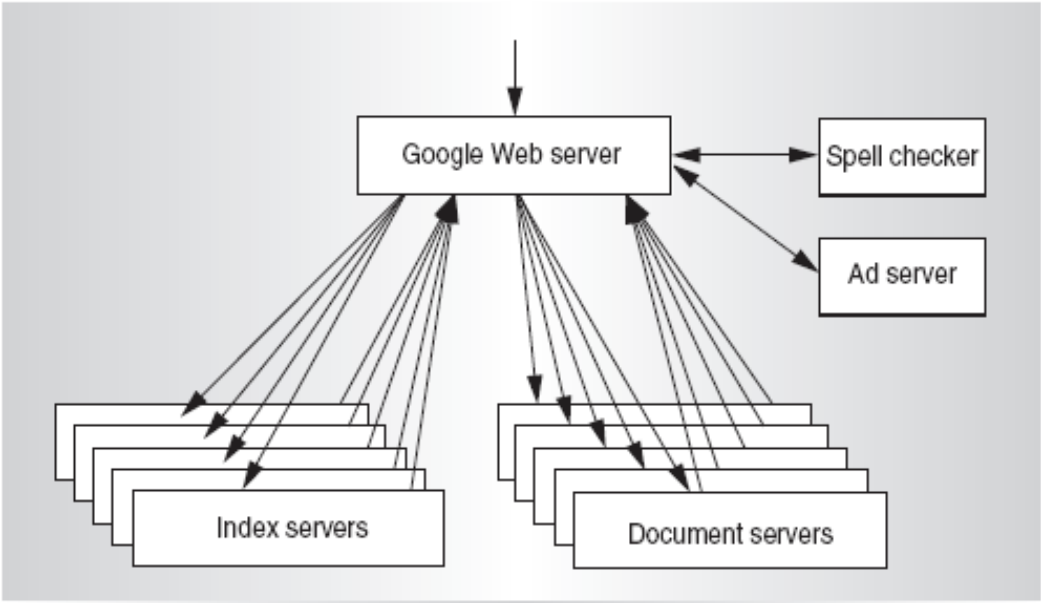
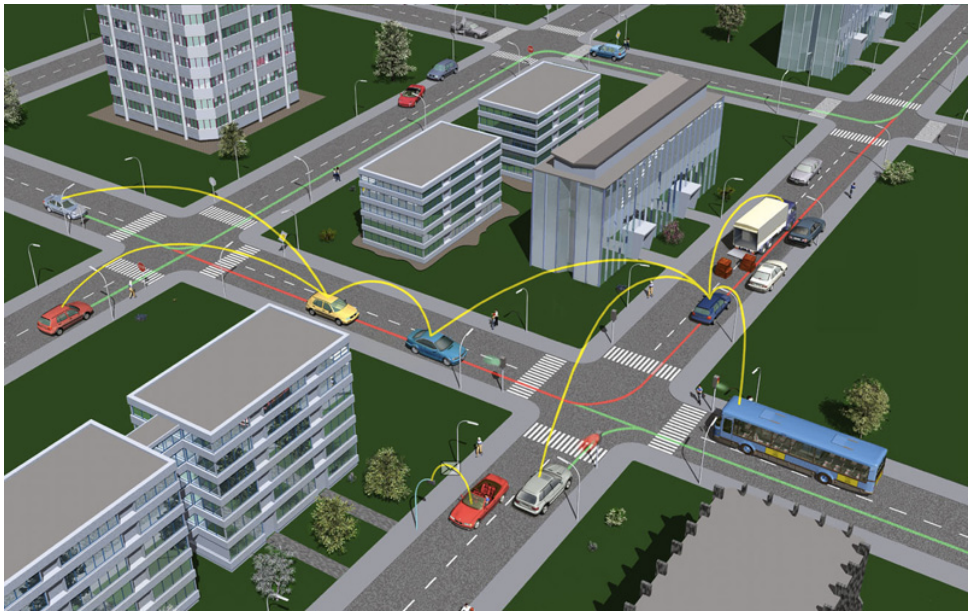


Figure 1. Google query-serving architecture.

Beispiel: Automobile

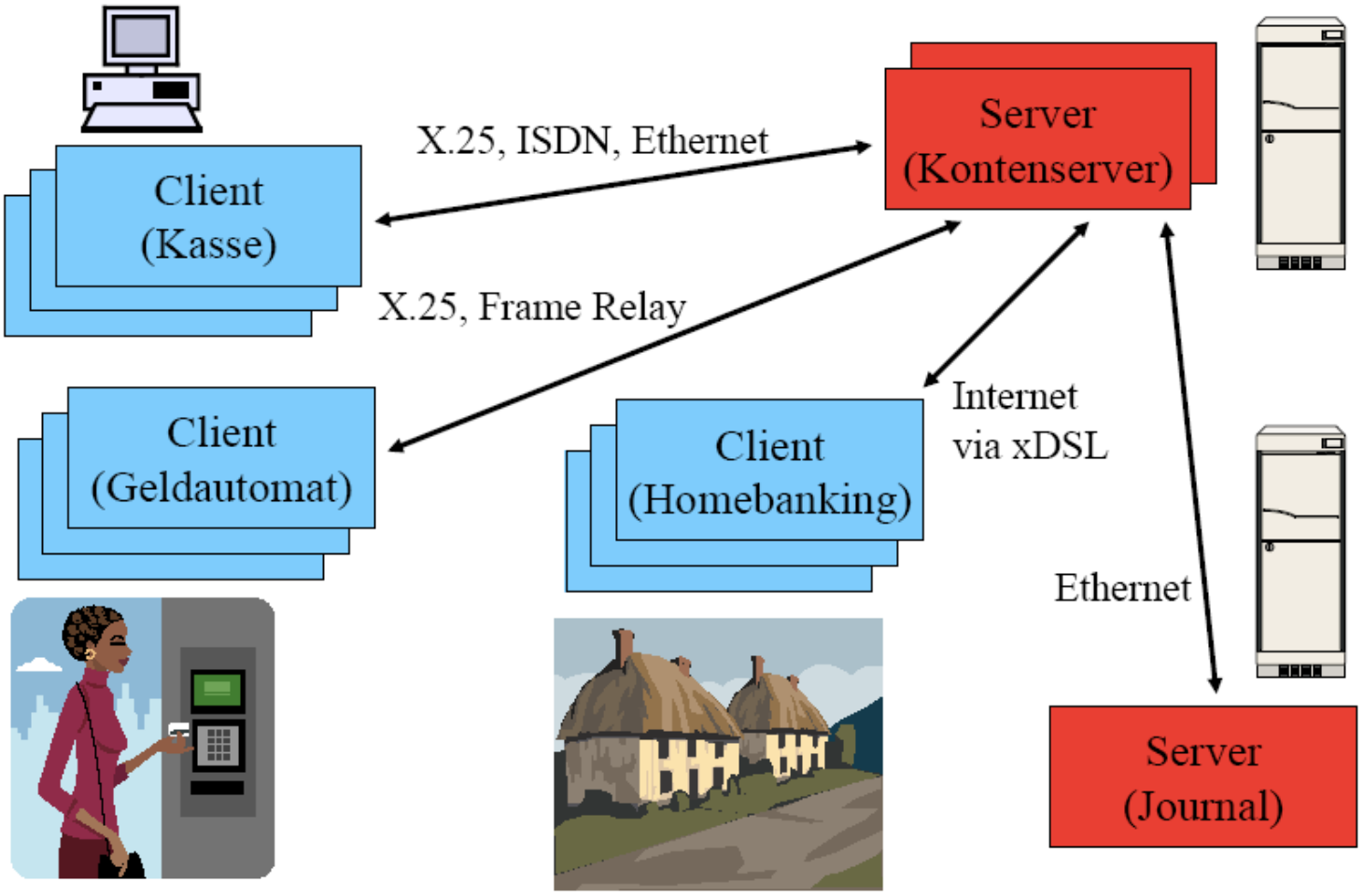
- Übermittlung von Verkehrsdaten (z.B. Staus) an andere Fahrzeuge und zentrale Server
- Ähnlich: Sensornetzwerke
- Probleme
  - Ad-hoc Netze
  - Topologie ändert sich ständig
  - Datenschutz



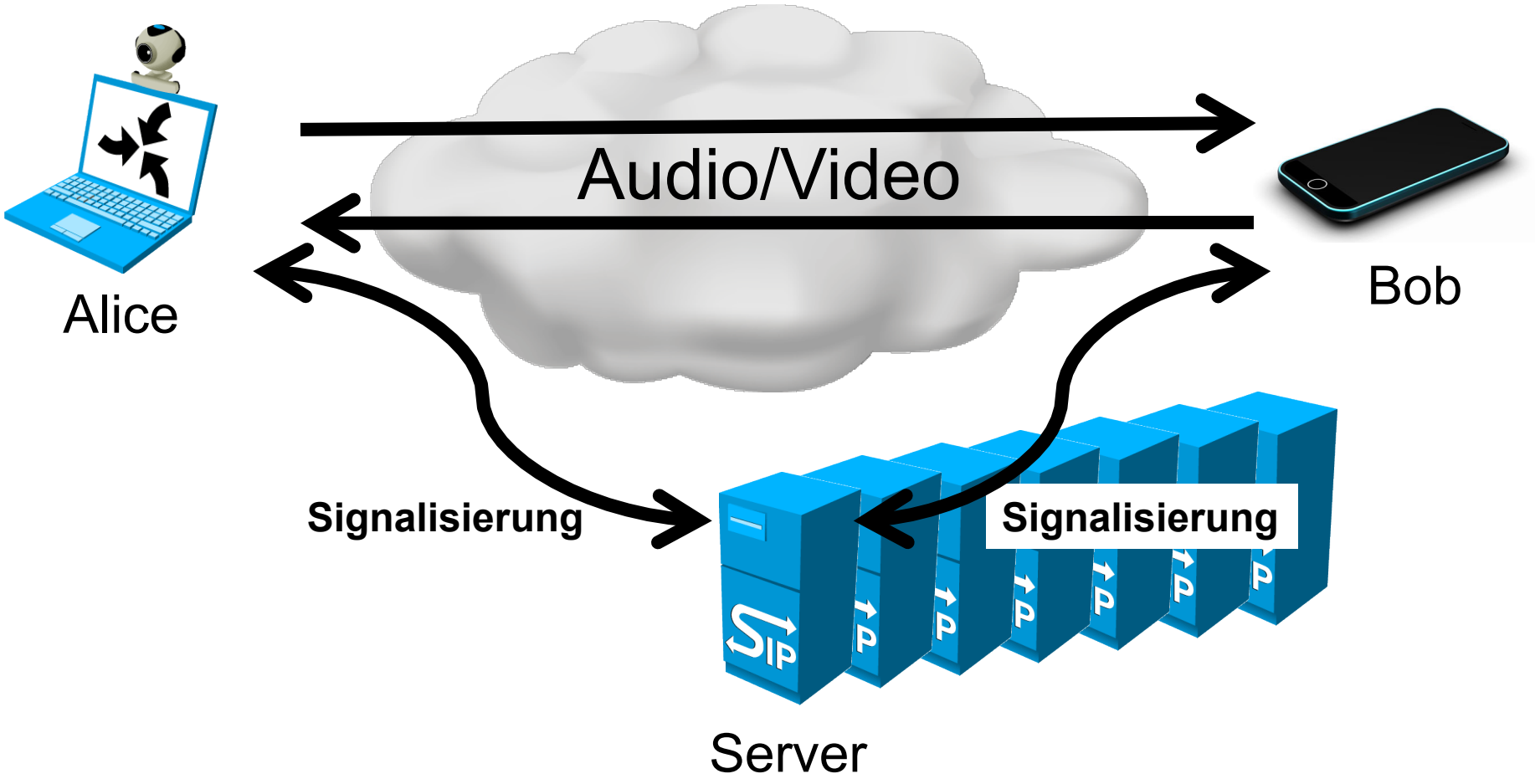
- Quellen
  - BBC: http://news.bbc.co.uk/2/hi/technology/6274974.stm
  - Car-to-X (Mercedes Benz)
  - Manuell: http://www.waze.com/

Beispiel: Bankanwendung

- Schema (stark vereinfacht)



Beispiel: Voice-over-IP



Beispiel: Grid- oder Cloud Computing

- Grid: Hochleistungsrechnen
  - Großrechner
  - Speicher
  - Geräte (z.B. LHC)
  - Entwickler verwalten
    einzelne Rechner
  - häufig nicht profitorientiert, Zielgruppe: Wissenschaft

Quelle: www.gridcafe.org

- Cloud: Vermieten von Rechnerkapazität o.ä. Ressourcen
  - Einzelne (virtualisierte) Rechner
  - I.d.R. existiert Middleware zur Vereinfachung der Nutzung (Programmierung)
  - Zielgruppe: kommerzielle Nutzer
- Nutzung: u.a. für *big data* Anwendungen

Interessante Aspekte

- In einem verteilten System sind die folgenden Eigenschaften/Funktionalitäten sehr schwer zu realisieren

  1. Skalierbarkeit
  2. Replikation / Synchronisation
  3. Fehlertoleranz

- Frage: Warum benötigt man diese Eigenschaften?

Ziele

- Wie sollte ein gutes verteiltes System aussehen?
- Mögliche Antworten:

Themen dieser Veranstaltung

1. Verteilungstransparenz: Die Verteiltheit ist verborgen
2. Offenheit: Das Gesamtsystem darf aus heterogenen Komponenten bestehen
3. Skalierbarkeit: Das System muss einfach erweiterbar (vergrößerbar) sein
4. Zuverlässigkeit: Das System sollte (fast) immer zur Verfügung stehen
5. Einfacher Zugriff auf Ressourcen

Verteilungstransparenz

- Das System stellt sich dem Benutzer als ein einzelner Dienst dar (kohärentes System)
- Der Benutzer weiß nicht, dass der verwendete Dienst verteilt ist
- Man weiß z.B. nicht
  - wie viele Komponenten beteiligt sind
  - wo diese Komponenten sich befinden (Ortstransparenz)

- Beispiel: Suche mit Google, Facebook Posting, ...

Verteilungstransparenz

- Arten von Transparenz
  - Zugriff: Auf Ressourcen wird mit gleichen Mechanismen zugegriffen
  - Ort: Verbirgt, wo sich eine Ressource befindet
  - Migration: Verbirgt, dass eine Ressource verschoben wird / wurde
  - Relokation: Verbirgt, dass eine Ressource während der Nutzung verschoben werden kann
  - Nebenläufigkeit: Verbirgt gleichzeitigen Zugriff mehrerer Nutzer
  - Fehler: Verbirgt Ausfall und Wiederherstellung einer Ressource

Quelle: Tanenbaum, van Steen, S. 22

Verteilungstransparenz

- Frage: Welche der genannten Eigenschaften kann man realistischerweise implementieren?

- Frage: Gibt es Fälle, in denen Verteilungs-transparenz gar nicht wünschenswert ist?

Offenheit

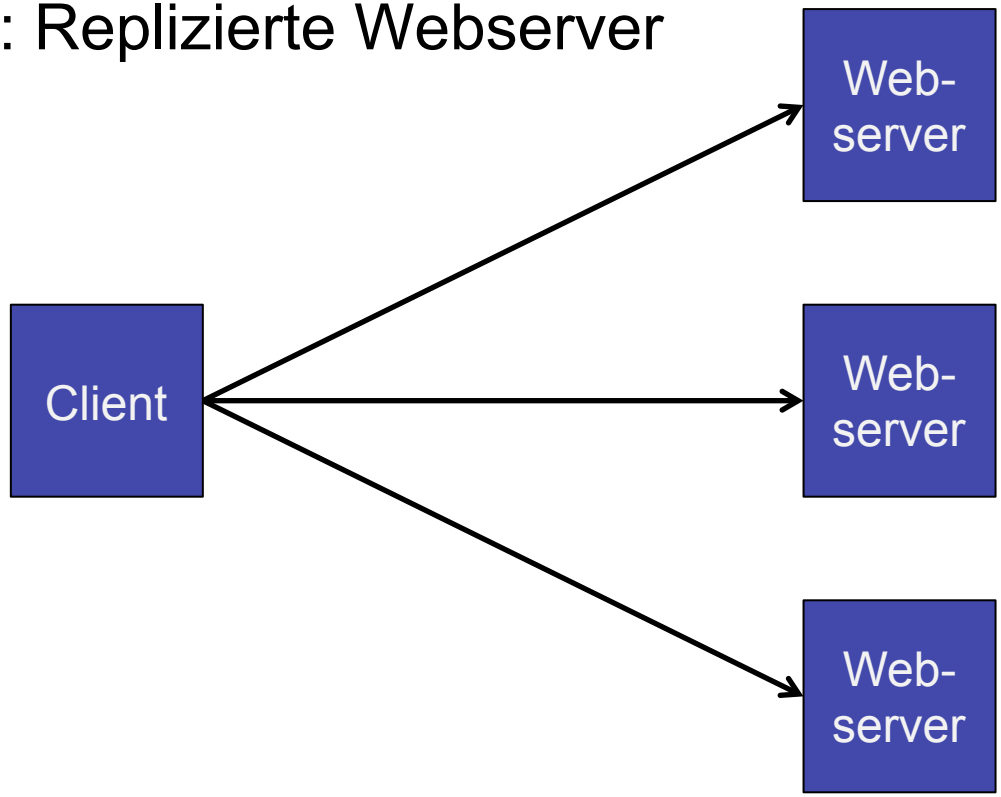
- Schnittstellen
  - Offengelegt
  - Kommunikation zwischen heterogenen Systemen
  - Interoperabilität zwischen Komponenten erwünscht
  - Realisiert z.B. mittels Schnittstellenbeschreibungssprachen
  - Heterogene Komponenten
  - Beispiel Betriebssystem: Komponenten können auf unterschiedlichen Betriebssystemen laufen
- Portabilität: Austausch von Komponenten möglich
- Erweiterbarkeit: zusätzliche Komponenten können einfach integriert werden

Skalierbarkeit

- Ein verteiltes System soll einfach erweiterbar sein
- Zusätzliche Komponenten bei
  - zusätzlicher Rechenlast
  - zusätzlichen Nutzern
  - zusätzlichen Aufgaben
  - ...
- Eingeschränkt häufig durch zentralisierte Komponenten, z.B.
  - nur 1 Webserver
  - nur 1 Datenbank
  - nur 1 ...

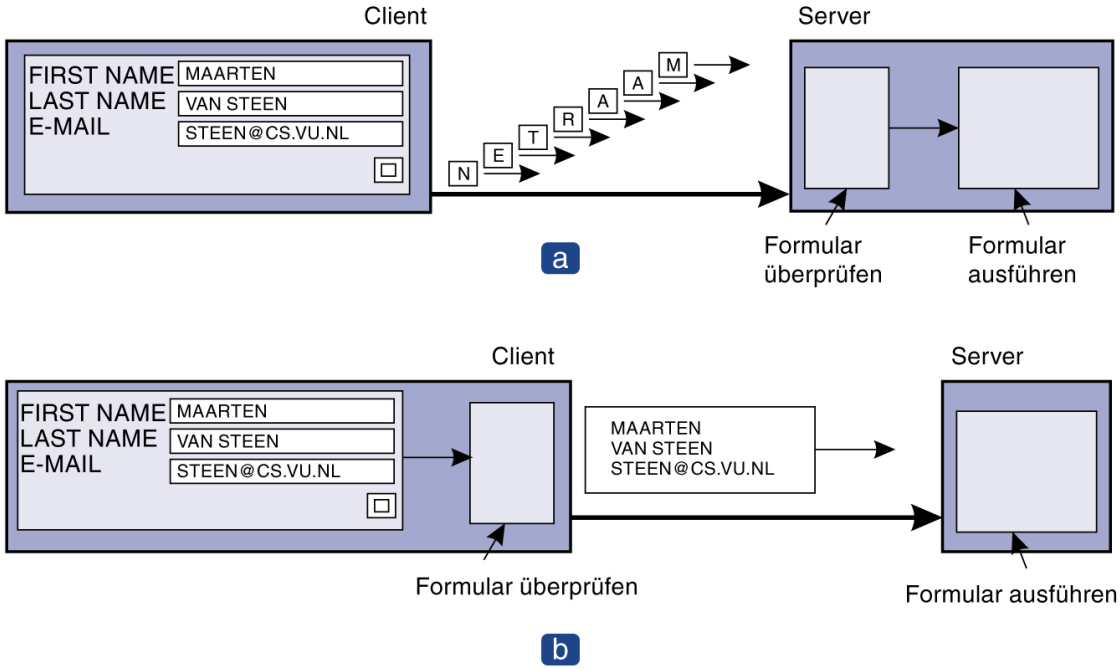
Skalierbarkeit

- Idee: Mehr als eine Komponente einsetzen
- Beispiel: Replizierte Webserver



- Frage: Welches Problem kann dabei entstehen?
- Hinweis: Wie realisiert man gemeinsamen Status?

Skalierbarkeit

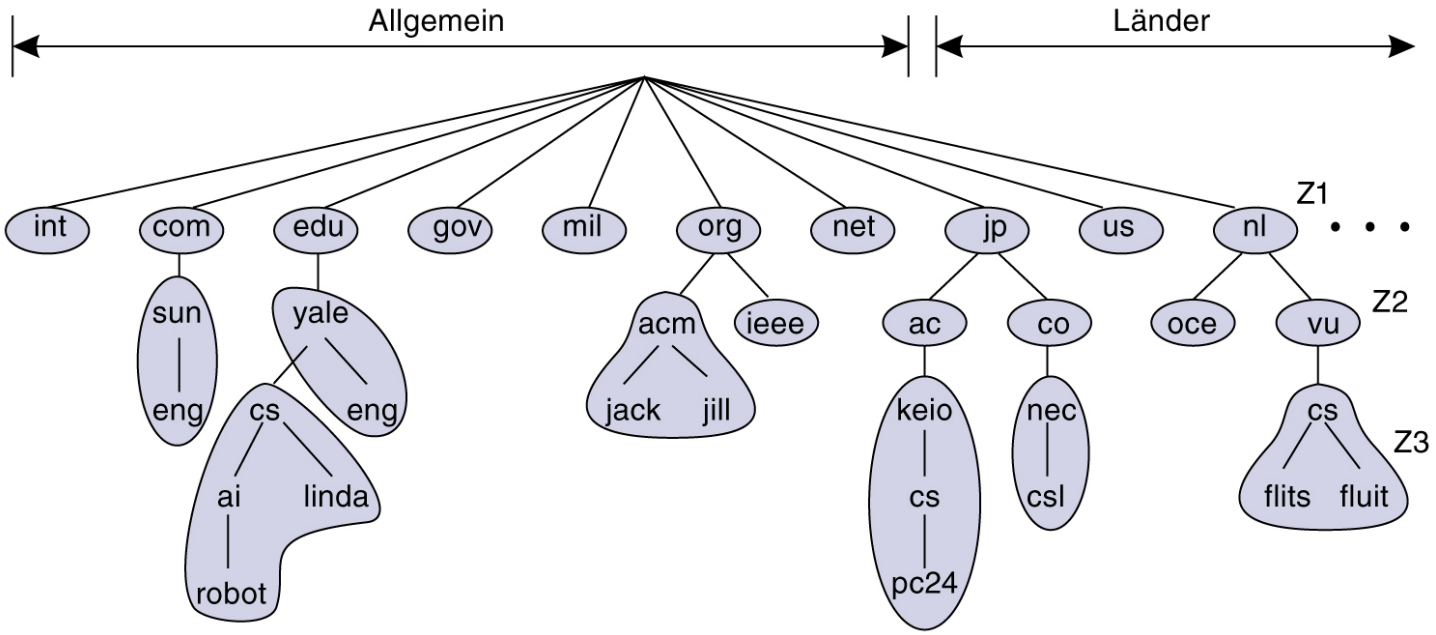


Quelle: Tanenbaum, v. Steen

- Beispiel Webserver: Ausfüllen von Formularen
- Skalierbarkeit kann hier verbessert werden durch
  - Überprüfung auf Client und Übertragung von Daten "am Stück" statt einzeln

Skalierbarkeit

- Beispiel: DNS



Quelle: Tanenbaum, v. Steen

- Frage: Wie erreicht DNS Skalierbarkeit?
- Antwort: Einteilung des Namensraumes in Zonen

Zuverlässigkeit

- Das System soll möglichst immer zur Verfügung stehen
- Fehlertoleranz: durch Verteilung und Redundanz kann Ausfall einzelner Komponenten kompensiert werden
- Keine *single points of failure* (s. Skalierbarkeit)

Zuverlässigkeit

- "*A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable*"

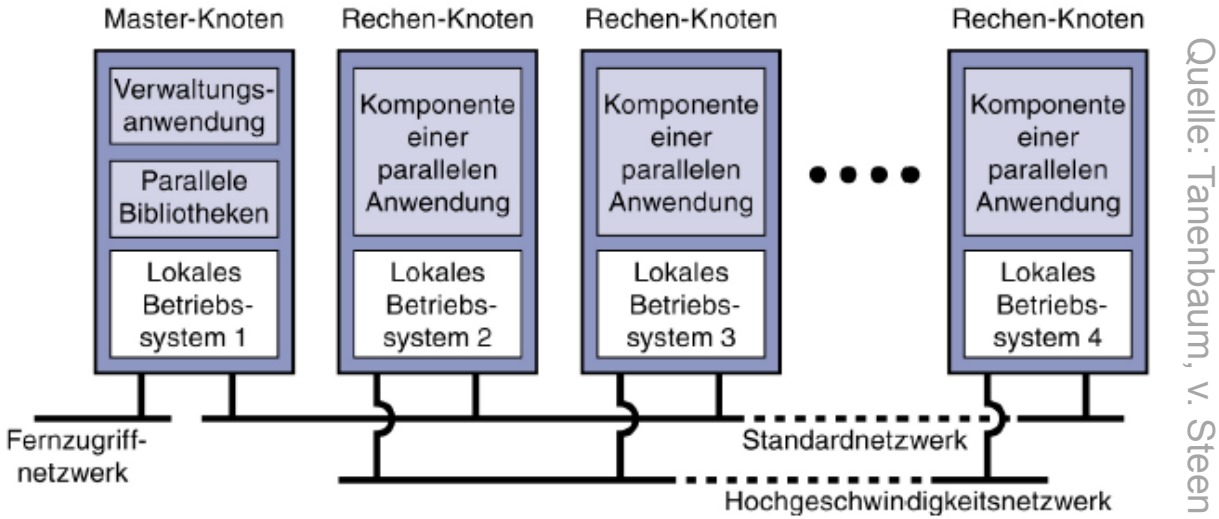
  - Leslie Lamport

- Beispiel: Network File System (NFS). Was passiert mit dem Dateisystem, wenn der Dateiserver ausfällt?

Hardware

- Einzelne Rechner bestehen maximal aus mehreren CPUs oder Kernen
- In einem verteilten System werden mehrere solcher Rechner durch ein Netzwerk (oder mehrere) verbunden

- Beispiel Clusterrechner

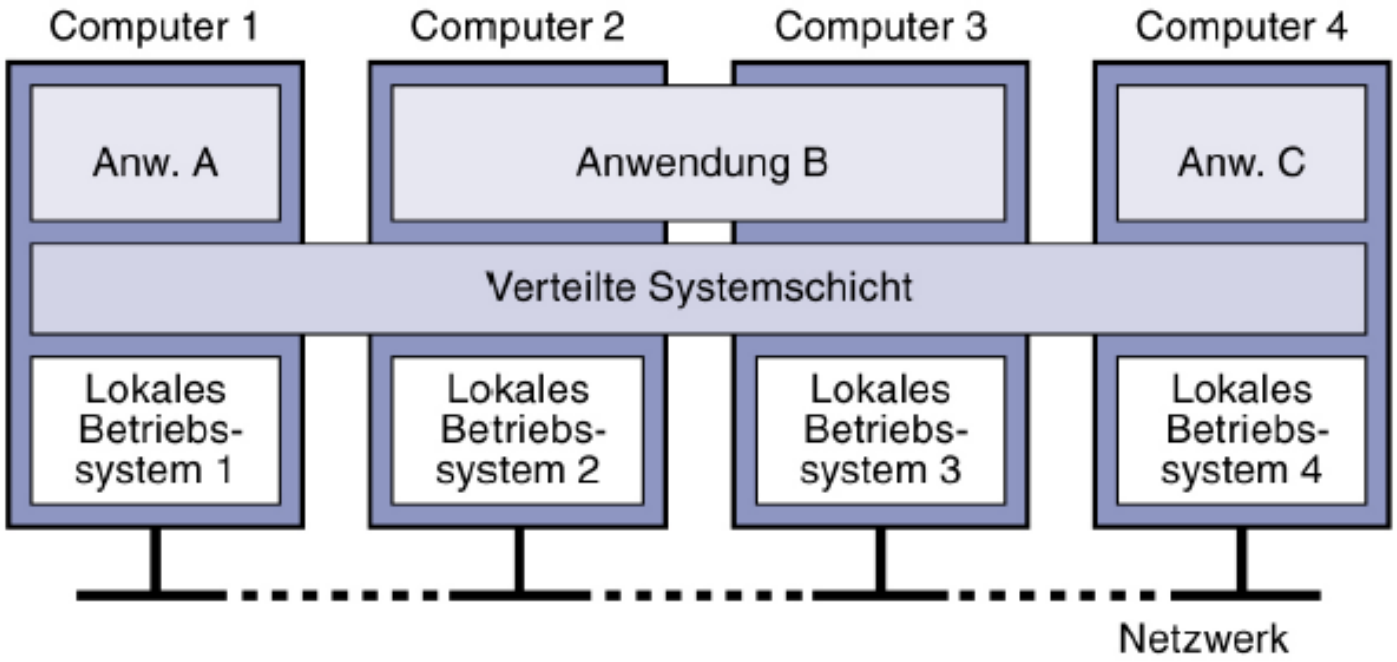


Quelle: Tanenbaum, v. Steen

- Systeme fangen oft homogen an, werden dann heterogen durch Upgrades und Erweiterungen

Software

- Heterogenität durch *Middleware* verborgen
- Middleware ist i.d.R. *kein* Betriebssystem
- Ziel ist Transparenz: einheitliche Schnittstelle, die Verteilung verbirgt



Quelle: Tanenbaum, v. Steen

System Architektur

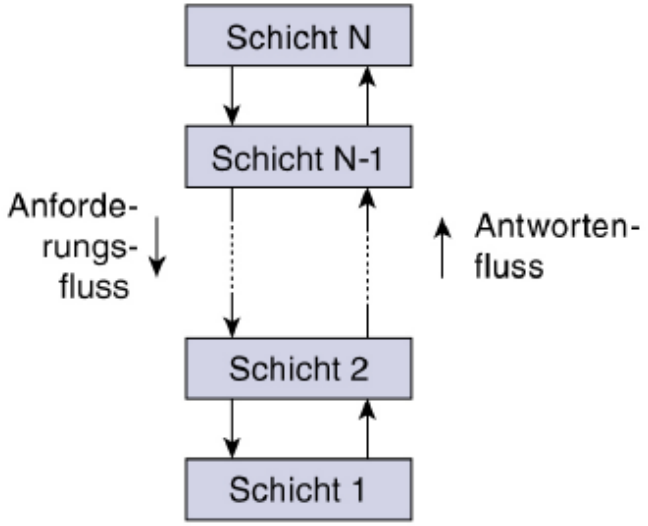
4 Architekturvarianten

1. Geschichtet
2. Objektbasiert (*service oriented*)
3. Ereignisbasiert (*event driven*)
4. Datenzentiert (*data centric*)

Architektur: Geschichtet

1. Client-Server
2. Multitier / Multiserver

- Verteiltes System besteht aus Schichten, die einzelne Aufgaben bearbeiten

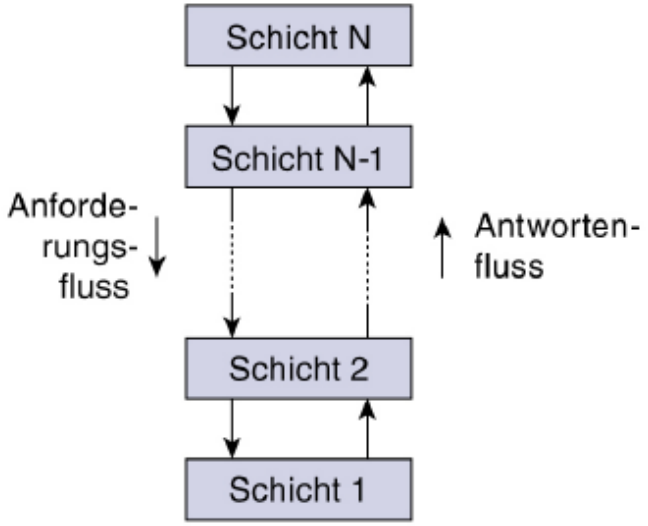


Quelle: Tanenbaum, v. Steen

Architektur: Geschichtet

1. Client-Server
2. Multitier / Multiserver

- Verteiltes System besteht aus Schichten, die einzelne Aufgaben bearbeiten
- Die Schichten können verteilt sein, d.h. jede Schicht kann sich auf einem anderen Rechner befinden

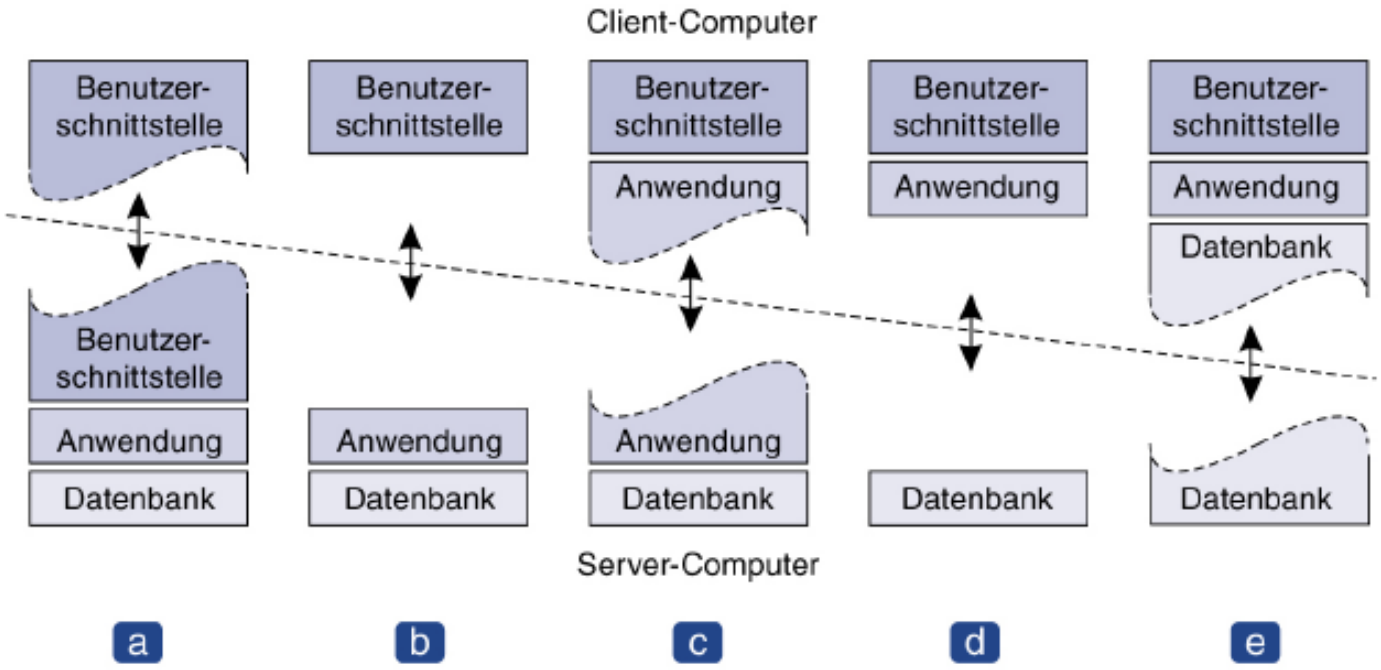


Quelle: Tanenbaum, v. Steen

1. Architekturvariante: Geschichtet

a. Client / Server

- 1-1 Beziehung zwischen einem Client und einem Server
- Trennung kann auf jeder Schicht erfolgen:

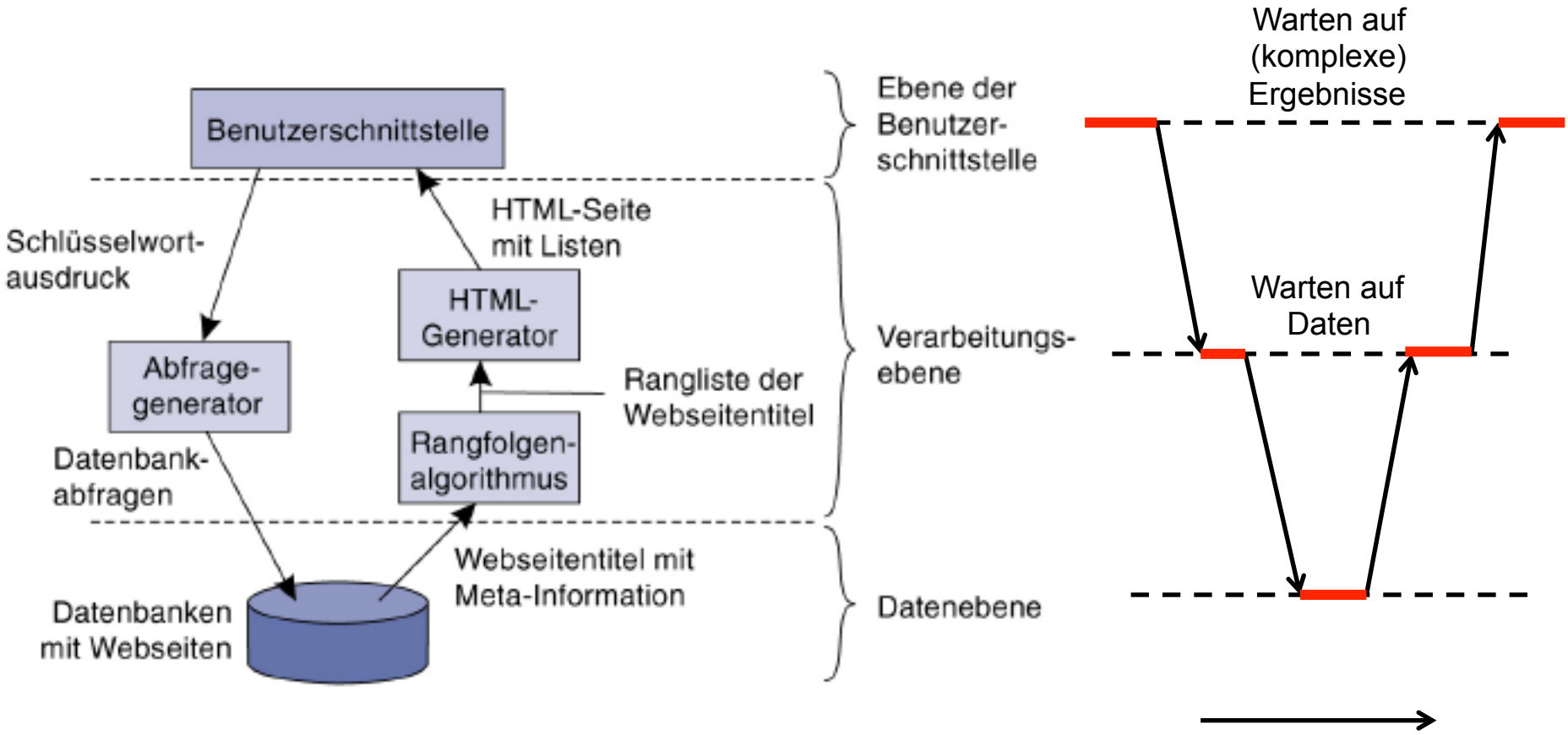


Quelle: Tanenbaum, v. Steen

1. Architekturvariante: Geschichtet

b. Multitier / Multiserver

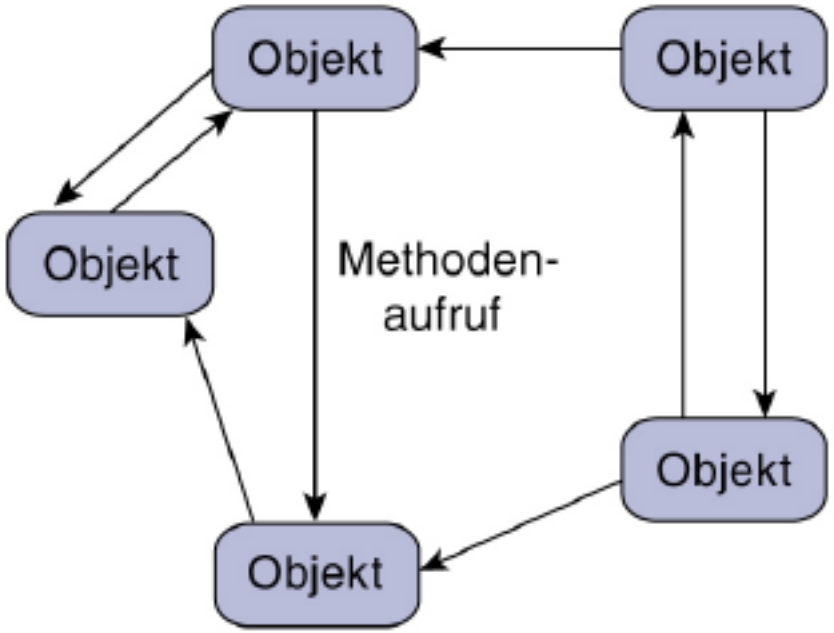
- Verarbeitung auf mehreren (Server-) Ebenen
- Mehrere Schichten physikalisch getrennt:



Quelle: Tanenbaum, v. Steen

2. Architekturvariante: Objektbasiert

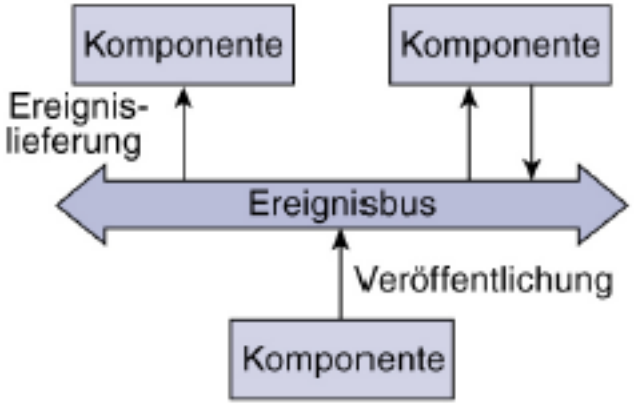
- Aufgaben werden durch Objekte mit klar definierten Schnittstellen erledigt
- Objekte werden verteilt und ihr Standort z.B. durch Dienste (z.B. Namensdienst) publiziert



Quelle: Tanenbaum, v. Steen

3. Architekturvariante: Ereignisbasiert

- Ein Ereignisbus transportiert Anfragen und Antworten zwischen Komponenten

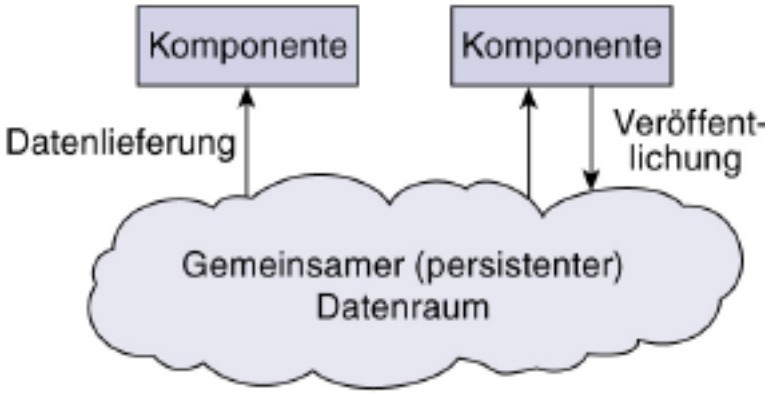


Ereignisbasierte Architektur

Quelle: Tanenbaum, v. Steen

4. Architekturvariante: Datenzentriert

- Ein gemeinsamer Datenraum (persistenter, d.h. dauerhafter Speicher) enthält alle Daten
- Komponenten stellen Anfragen und Antworten in den Datenraum



Datenzentrierte Architektur

Quelle: Tanenbaum, v. Steen

Übung

- Für welches Beispiel ist welche Architekturvariante geeignet?

Architektur	Webserver	Automobile / Sensornetz	Bank-anwendung	Voice-over-IP
Client/Server				
Objektbasiert				
Ereignisbasiert				
Datenzentriert				

Architekturvarianten

- Ein (komplexes) verteiltes System kann praktisch aus beliebigen Varianten bestehen und diese auch kombinieren, z.B. *event-driven* mittels *Client/Server*

- Ein verteiltes System kann sowohl die eine als auch die andere Variante darstellen, je nach Sichtweise

- Beispiel Suchmaschine: Datenzentriert oder Client/Server ?

Umgebung

- Das Netzwerk ist zuverlässig
- Das Netzwerk ist sicher
- Das Netz ist homogen
- Die Zusammensetzung des Systems ändert sich nicht
- Die Latenzzeit ist konstant gleich Null
- Die Bandbreite ist nicht begrenzt
- Es gibt genau einen Administrator

i.d.R. Fehleinschätzungen

- Frage: Welche dieser Annahmen sind korrekt?

Umgebung: Realität

- Netzwerke sind unsicher, nicht zuverlässig, nicht immer verfügbar
  - Firewalls, Paketverlust, Defekte, etc.
- Nachrichten brauchen Zeit um den Empfänger zu erreichen: Zeit zwischen Komponente A und Komponente B ist nicht gleich Zeit zwischen Komponente A und Komponente C
- Reihenfolge von Nachrichten wird im Netzwerk vertauscht
- Es gibt *keine* gemeinsame Zeit
- Komponenten können ausfallen (s. Zitat Lamport)