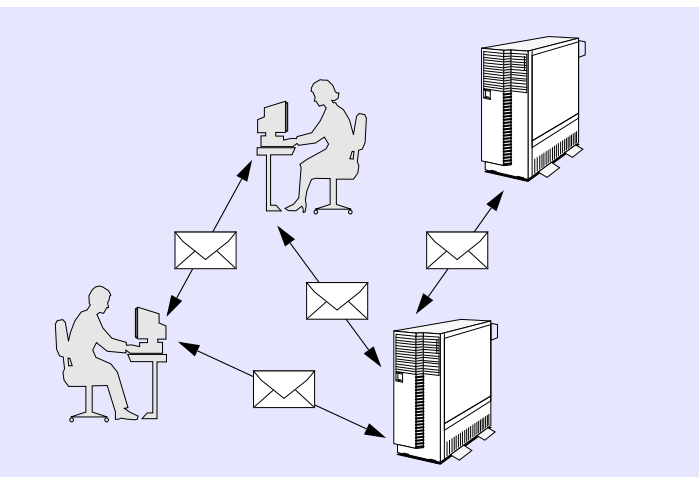
Verteilte Systeme

Teil 1

Friedemann Mattern

Institut für Pervasive Computing
Departement Informatik
ETH Zürich



- Rechner, Personen, Prozesse, "Agenten" sind an *verschiedenen Orten*.
- Autonome Handlungsträger, die jedoch gelegentlich *kooperieren* (und dazu über Nachrichten *kommunizieren*).

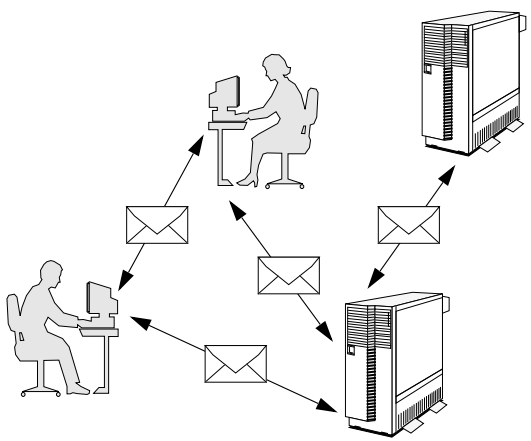
Wer bin ich? Wer sind wir?

Fachgebiet "**Verteilte Systeme**" im Departement Informatik, Institut für Pervasive Computing

- 12 Assistent(inn)en
- Forschungsprojekte

Mehr zu uns:
www.vs.inf.ethz.ch

- Infrastruktur für verteilte Systeme
- Internet der Dinge
- Ubiquitous Computing
- Sensornetze
- Verteilte Anwendungen und Algorithmen



Organisatorisches zur Vorlesung

5-stündige Veranstaltung (Vorlesung inkl. Übungen)

Sinnvolle Vorkenntnisse (Grundlagen):

- 4 Semester der Bachelorstufe Informatik (inkl. Mathematik-Anteil)
- Computernetze
- Grundkenntnisse Betriebssysteme (z.B. Prozessbegriff, Synchronisation)
- UNIX, Java

Mo 8:15 - 11:00, IFW A36
Fr 8:15 - 10:00, IFW A36
} *Vorlesung inkl. Übung*

Absicht!

- Gelegentliche *Denkaufgaben* in der Vorlesung
- Praktische Übungen korrelieren im Allgemeinen nur schwach mit dem Inhalt der Vorlesung *(komplementieren* die Vorlesung): "Kommunikation mit Nokia N95-Geräten"
- Gelegentliche *Übungsstunden* (zu den "Vorlesungsterminen") zur Besprechung der Aufgaben und Vertiefung des Stoffes

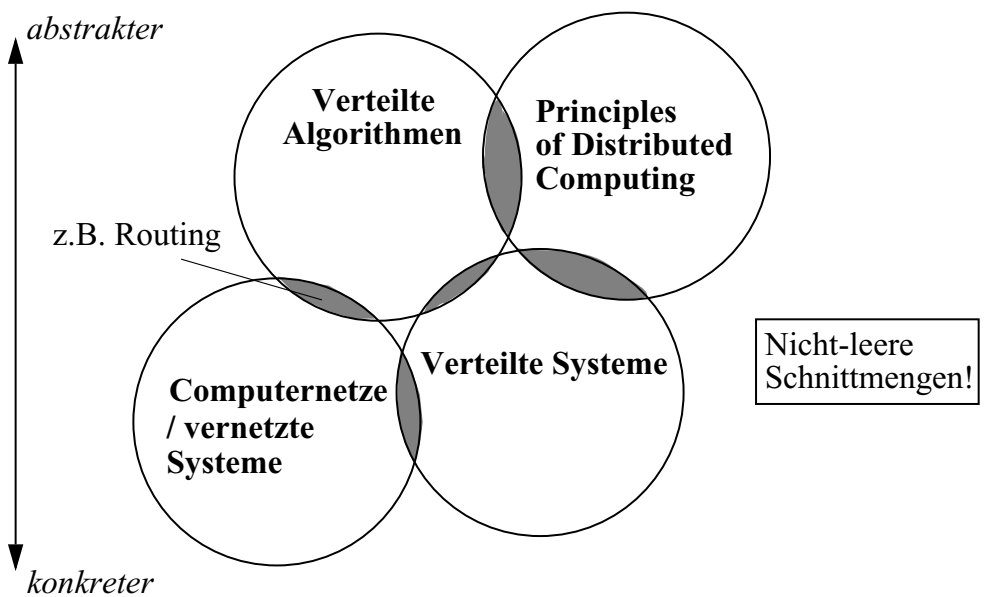
---

- *Folienkopien* jeweils einige Tage nach der Vorlesung
  - im .pdf-Format bei www.vs.inf.ethz.ch/edu
- Vorlesung ab November: *Prof. Roger Wattenhofer*
- Prüfung schriftlich
  - bewertete Übungen gehen zu kleinem Teil in die Prüfungsnote ein
- Ansprechperson für organisatorische Aspekte:
  - Matthias Kovatsch, kovatsch@inf.ethz.ch

Thematisch verwandte Veranstaltungen im Masterstudium

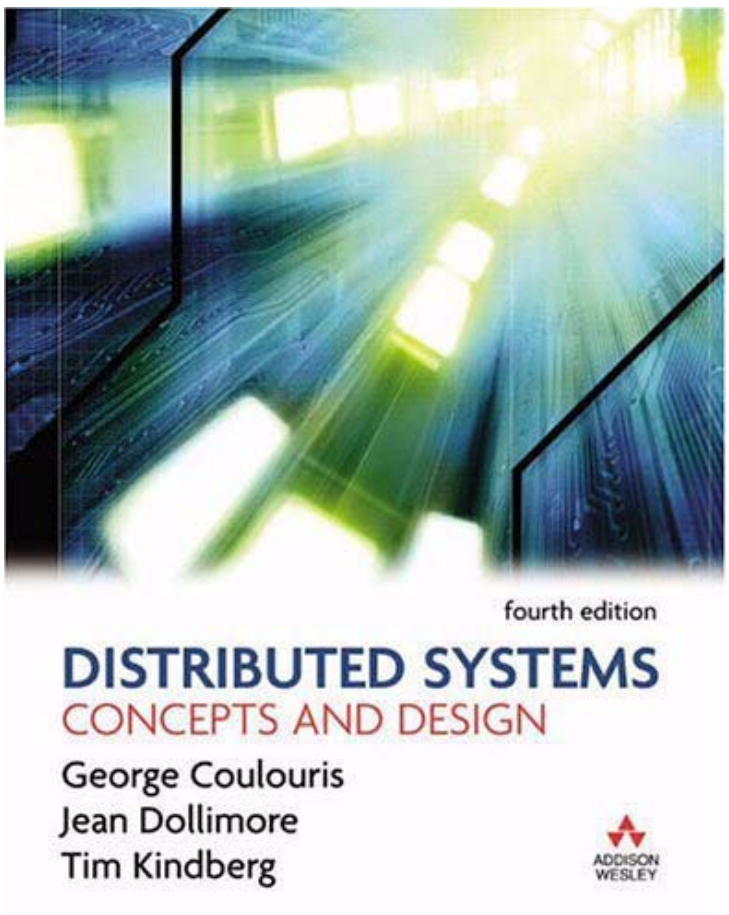
- Ubiquitous Computing
- Enterprise Application Integration - Middleware
- Web Services and Service Oriented Architectures
- Verteilte Algorithmen
- Principles of Distributed Computing
- ...

- Einschlägige Seminare
- Semester- und Masterarbeit
- Praktikum ("Lab")



Literatur

G. Coulouris, J. Dollimore, T. Kindberg: *Distributed Systems: Concepts and Design (4th ed.).* Addison-Wesley, 2005



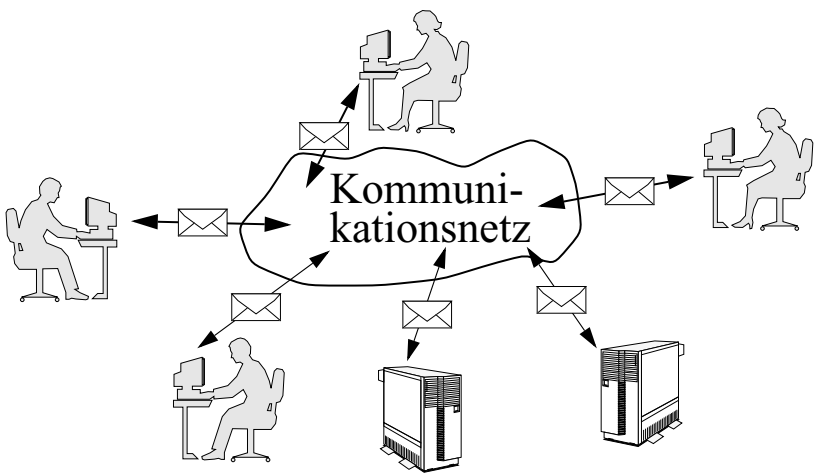
A. Tanenbaum, M. van Steen: *Distributed Systems: Principles and Paradigm (2nd ed.).* Prentice-Hall, 2007

Oliver Haase: *Kommunikation in verteilten Anwendungen (2. Auflage).* R. Oldenbourg Verlag, 2008

"Verteiltes System" - zwei Definitionen

A distributed computing system consists of multiple autonomous processors that do not share primary memory, but cooperate by sending messages over a communication network.

-- H. Bal

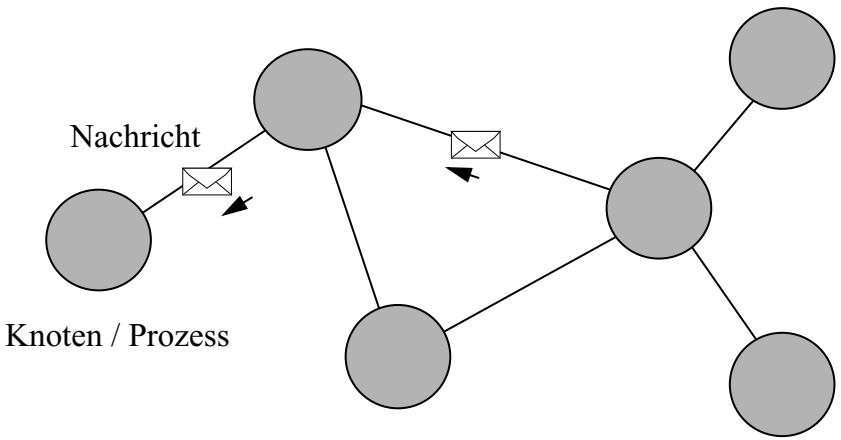


A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

-- Leslie Lamport

- welche Problemaspekte stecken hinter Lamports Charakterisierung?

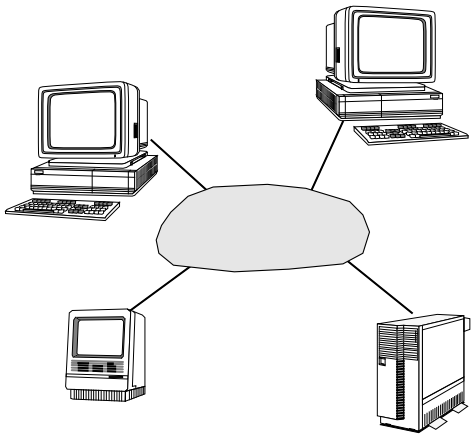
“Verteiltes System”



Physisch verteiltes System:
Mehrrechnersystem ... Rechnernetze

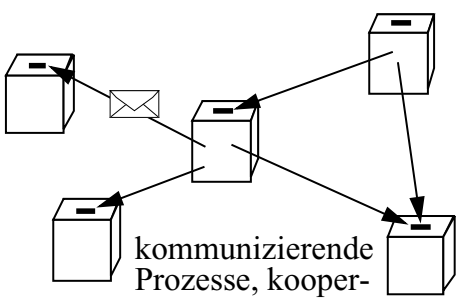
Logisch verteiltes System: Prozesse (Objekte, Agenten)
- Verteilung des Zustandes (keine globale Sicht)
- Keine gemeinsame Zeit (globale, genaue Uhr)

Sichten verteilter Systeme



Computernetz mit “Rechenknoten”:
- Compute-Cluster
- Local Area Network
- Internet

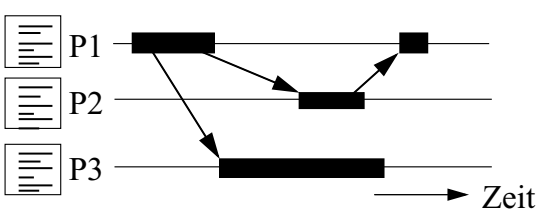
$\Rightarrow$ Routing, Adressierung,..



Objekte in *Betriebssystemen, Middleware, Programmiersprachen*

kommunizierende Prozesse, kooperierende Objekte

$\Rightarrow$ “Programmierersicht” (Client, Server,...)

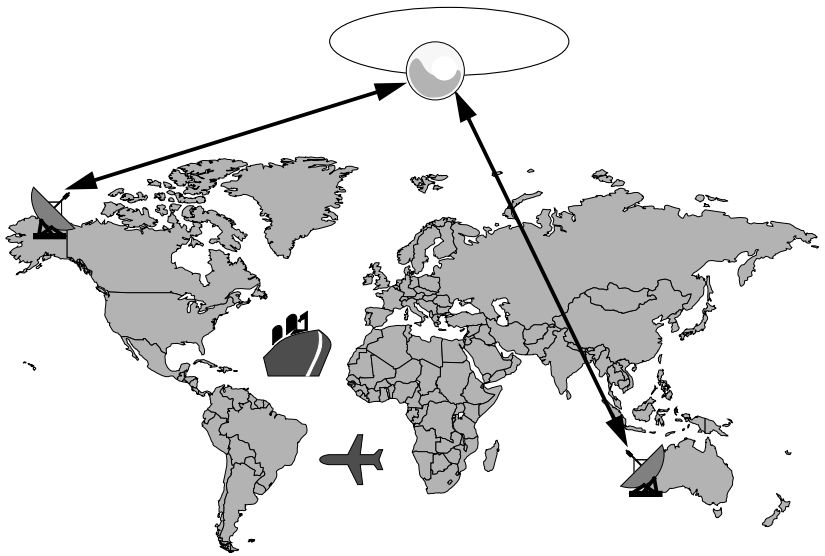


Algorithmen- und Protokollebene

- Aktionen, Ereignisfolgen
- Konsistenz, Korrektheit

zunehmende Abstraktion

Die verteilte Welt



Auch die “reale Welt” ist ein verteiltes System:

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

Warum verteilte Systeme?

- Es gibt *inhärent geographisch verteilte* Systeme
  → z.B. Zweigstellennetz einer Bank; Steuerung einer Fabrik
  (Zusammenführen / Verteilen von Information)

- *Electronic commerce*
  → kooperative Informationsverarbeitung räumlich getrennter Institutionen
  (z.B. Reisebüros, Kreditkarten,...)

- Mensch-Mensch-*Telekommunikation*
  - E-Mail, Diskussionsforen, Blogs, digitale soz. Netze, IP-Telefonie,...

- *Globalisierung* von Diensten
  - Skaleneffekte, Outsourcing,...

- Wirtschaftliche Aspekte
  - Cluster von einfachen Computern oder Netz von PCs manchmal besseres Preis-Leistungsverhältnis als Grosscomputer
  - Outsourcing von Diensten manchmal wirtschaftlicher
  - vgl.: Virtualisierung, dynamische Lastverteilung, Cloud Computing

Verteilte Systeme als “Verbunde”

- Verteilte Systeme *verbinden* räumlich (oder logisch) getrennte Komponenten zu einem bestimmten *Zweck*

  - *Systemverbund*
    - gemeinsame Nutzung von Betriebsmitteln, Geräten,....
    - einfache inkrementelle Erweiterbarkeit

  - *Funktionsverbund*
    - Kooperation bzgl. Nutzung jeweils spezifischer Eigenschaften

  - *Lastverbund*
    - Zusammenfassung der Kapazitäten

  - *Datenverbund*
    - allgemeine Bereitstellung von Daten

  - *Überlebensverbund*
    - Redundanz durch Replikation
    - dann i.Allg. nur Ausfall von Teilfunktionalität

Historische Entwicklung (“Systeme”)

Rechner-zu-Rechner-Kommunikation

- Zugriff auf entfernte Daten (“Datenfernübertragung”, DFÜ)
- dezentrale Informationsverarbeitung war zunächst ökonomisch nicht sinnvoll (zu teuer, Fachpersonal nötig)
  → Master-Slave-Beziehung (“Remote Job Entry”, Terminals)

ARPA-Netz (Prototyp des Internet)

- “symmetrische” Kommunikationsbeziehung (“peer to peer”)
- Internet-Protokollfamilie (TCP/IP,...)
- file transfer (ftp), remote login, E-Mail

Workstation-Netze (LAN)

- bahnbrechende, frühe Ideen bei XEROX-PARC
  (XEROX-Star als erste Workstation, Desktop-Benutzerinterface, Ethernet, RPC, verteilte Dateisysteme,...)

Kommerzielle Pionierprojekte als Treiber

- z.B. Reservierungssysteme, Banken, Kreditkarten

WWW (und Internet) als Plattform

- für electronic commerce etc.
- XML, web services, peer to peer,...
- neue, darauf aufbauende Dienste

Historische Entwicklung (“Konzepte”)

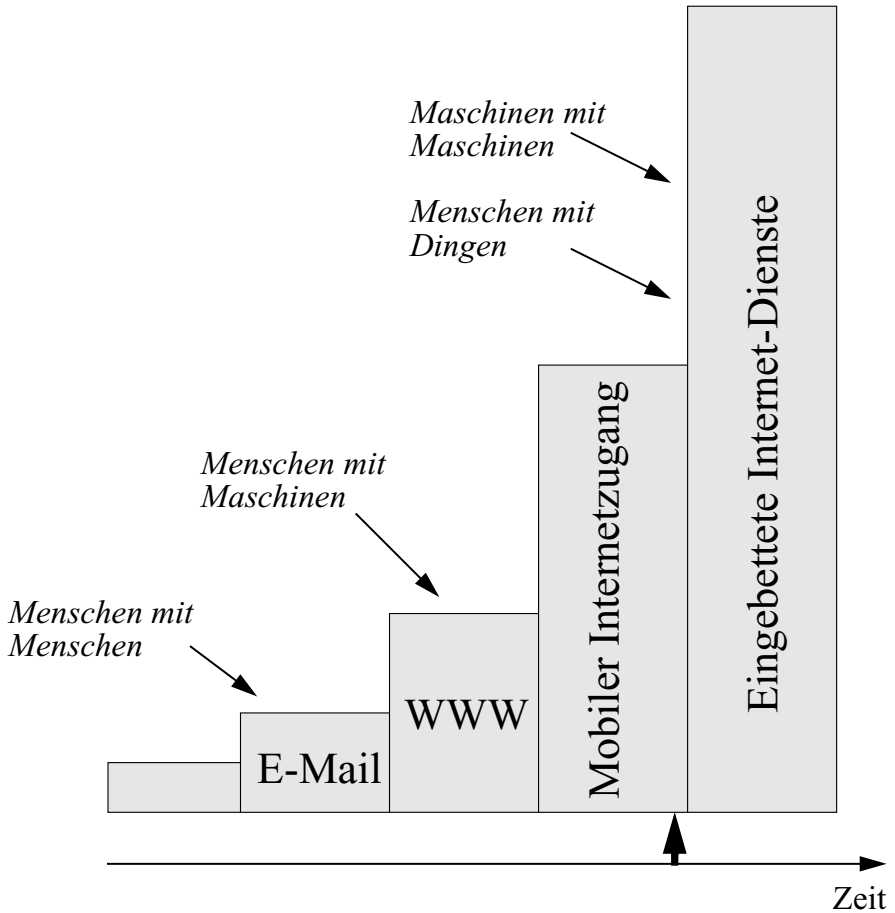
- Concurrency, Synchronisation,...
  - war bereits klassisches Thema bei Datenbanken und Betriebssystemen
- Programmiersprachen
  - kommunizierende Objekte
- Physische Parallelität
  - z.B. Multiprozessoren
- Parallele und verteilte Algorithmen
- Semantik
  - mathematische Modelle für Verteiltheit (z.B. CCS, Petri-Netze)
- Abstraktionsprinzipien
  - Schichten, Dienstprimitive,...
- Verständnis grundlegender Phänomene der Verteiltheit
  - Konsistenz, Zeit, Zustand,...

Entwicklung “guter” Konzepte, Modelle, Abstraktionen etc. zum Verständnis der Phänomene dauert oft lange

- notwendige Ordnung und Sichtung des verfügbaren Gedankenguts

Diese sind jedoch für die Lösung praktischer Probleme hilfreich, oft sogar notwendig!

Das qualitative Internet-Wachstum



Vernetzung von:

- Computern (ftp)
- Dokumenten (WWW)
- Menschen (digitale soziale Netze)
- Dingen (“Internet der Dinge”)

Software-Infrastruktur für Internet-basierte Anwendungen?

- Phänomen: das Internet verbreitet sich immer weiter
  - mehr Nutzer, Popularisierung
  - bis an die Person (Handy)
  - immer “exotischere” Endgeräte (TV, Kühlschrank, Chipkarte)
  - bald enthalten vielleicht auch viele Supermarktprodukte, Kleidungsstücke etc. kommunikationsfähige Chips
  - Sensoren

- Mobile Geräte, dynamische Umgebungen

- Es entstehen neue Dienste im Netz

- Dienste interagieren miteinander
  - Kompatibilität, Standards, Protokolle, offene Schnittstellen,...

- Markt erfordert sehr schnelle Reaktion
  - schnelle Implementierung neuer Dienste
  - Update über das Netz

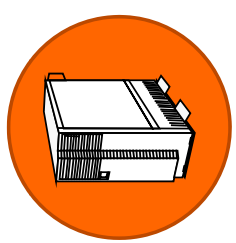
- Anschluss neuer Geräte muss “von selbst” erfolgen
  - Integration in eine Infrastruktur und Umgebung von Ressourcen

---

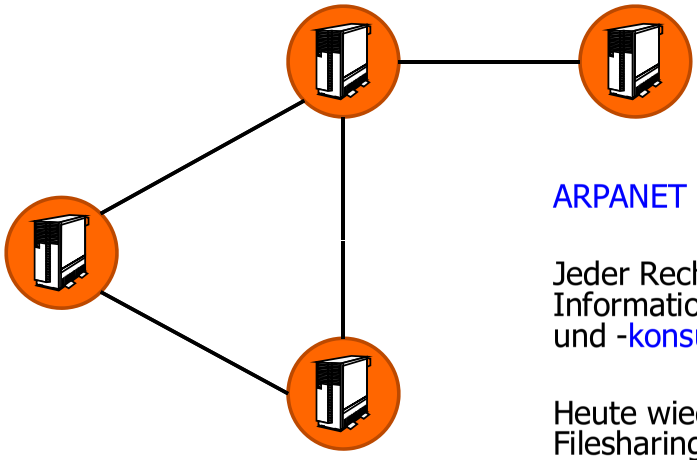
- Kann man eine Infrastruktur schaffen, die das unterstützt?
- Welche Systemarchitektur ist hierfür geeignet?

Architekturen verteilter Systeme

- Zu Anfang waren Systeme monolithisch
  - Mainframes
  - Nicht verteilt / vernetzt
  - Terminals
  - (Fernschreiber, ASCII)



Architekturen verteilter Systeme: Peer-to-Peer

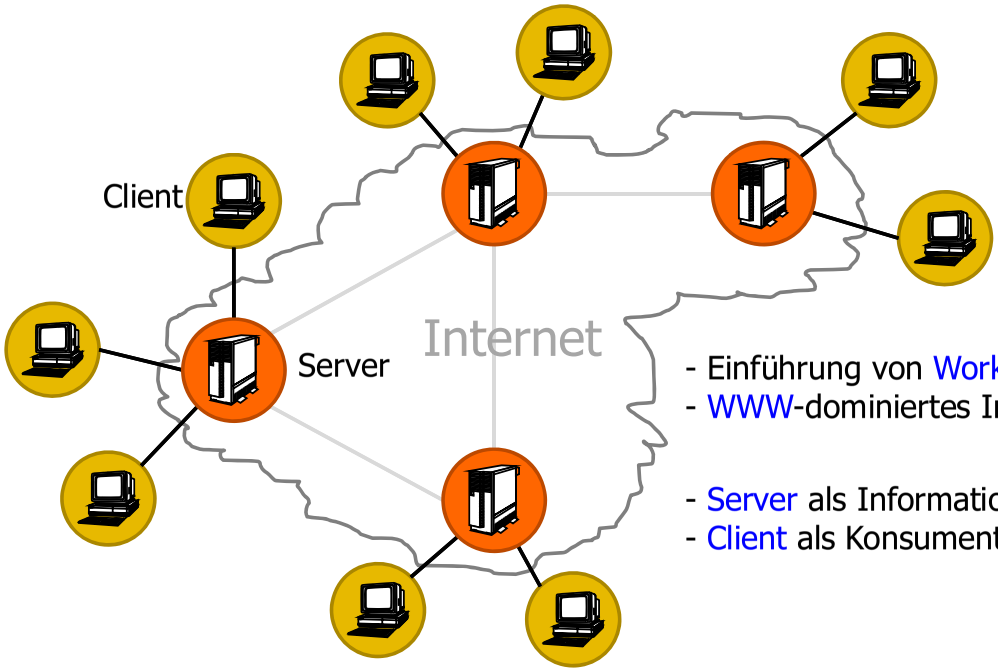


ARPANET 1969

Jeder Rechner gleichzeitig Informationsanbieter und -konsument

Heute wieder aktuell für Filesharing → Overlay-Netze

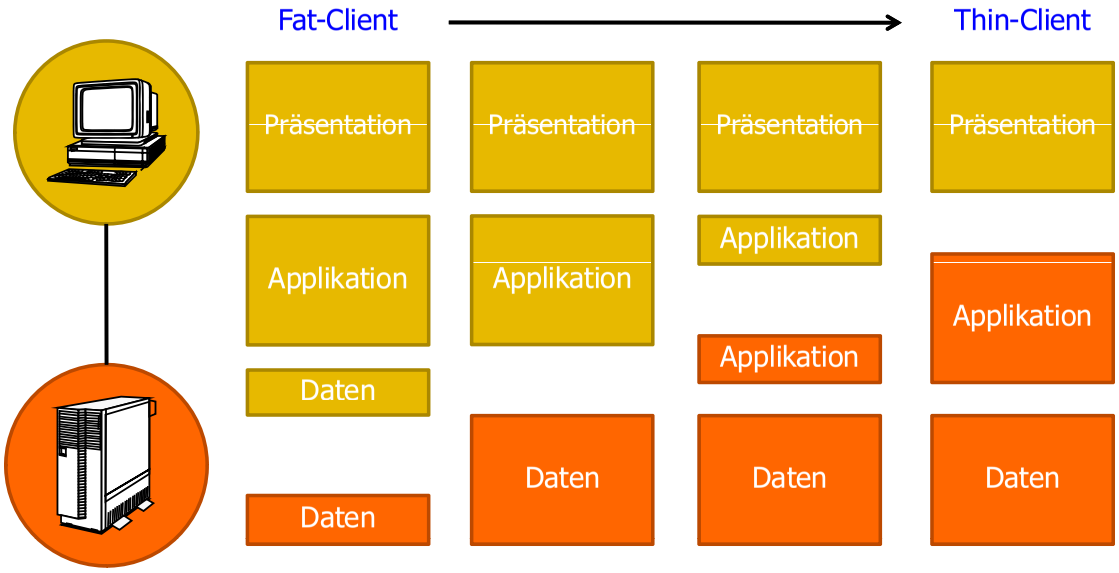
Architekturen verteilter Systeme: Client-Server



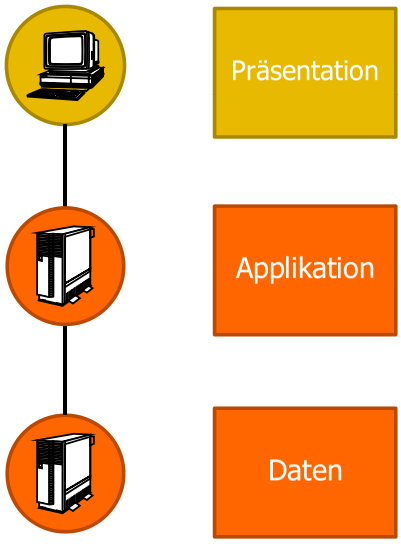
- Einführung von Workstations
- WWW-dominiertes Internet

- Server als Informationsanbieter
- Client als Konsument

Architekturen verteilter Systeme: Fat- undThin-Client



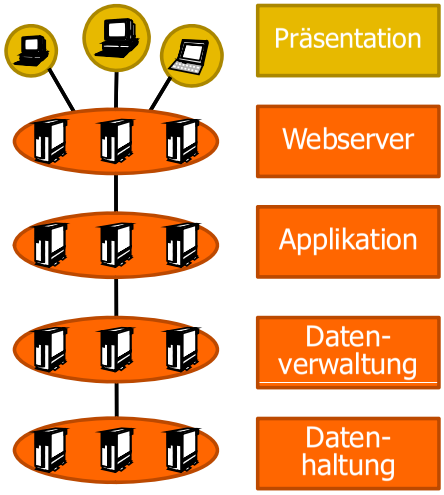
Architekturen verteilter Systeme: 3-Tier



Verarbeitung wird auf mehrere physikalische Einheiten verteilt

Logische Schichten minimieren die Abhängigkeiten
Leichtere Wartung
Einfaches Austauschen

Architekturen verteilter Systeme: Multi-Tier

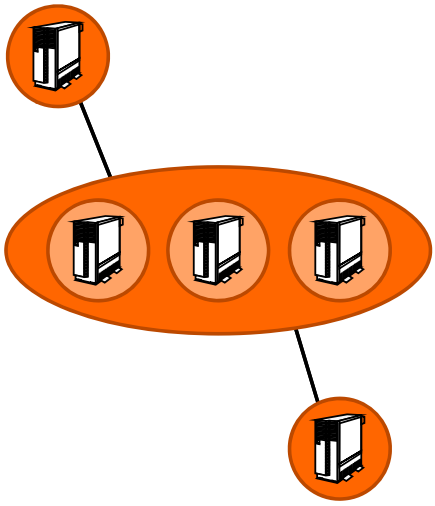


Weitere Schichten sowie mehrere physikalische Einheiten pro Schicht erhöhen die Skalierbarkeit und Flexibilität

Mehrere Webserver ermöglichen z.B. Lastverteilung

Verteilte Datenbanken in der Datenhaltungsschicht bietet Sicherheit durch Replikation und hohen Durchsatz

Architekturen verteilter Systeme: Service-Oriented Architecture (SOA)



Eine Unterteilung der Applikation in einzelne, unabhängige Abläufe innerhalb eines Geschäftsprozesses erhöht die Flexibilität weiter

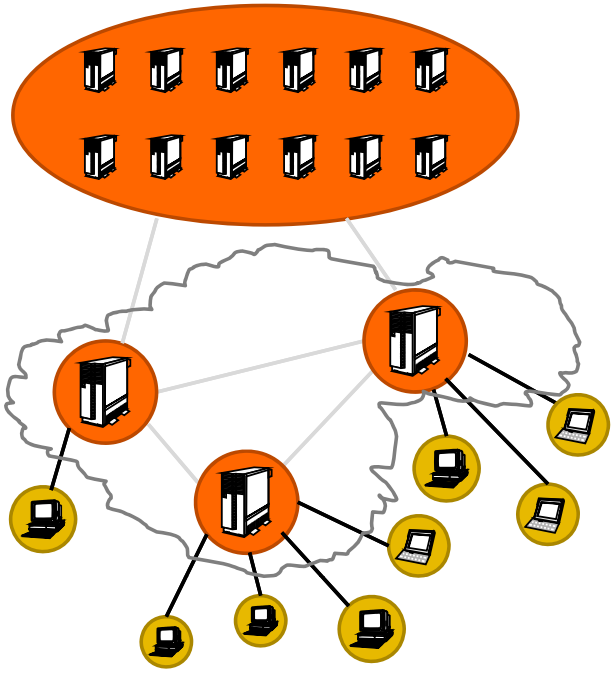
Lose Kopplung zwischen Services über Nachrichten und events (statt RPC)

Services können bei Änderungen der Prozesse einfach neu zusammengestellt werden („development by composition")

Services können auch von externen Anbietern bezogen werden

Oft in Zusammenhang mit Web-Services

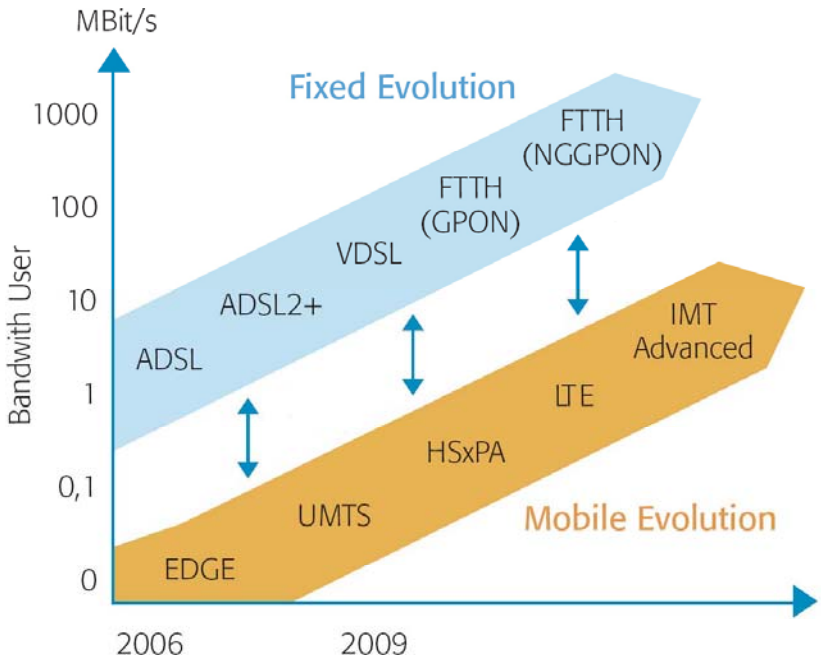
Architekturen verteilter Systeme: Cloud-Computing



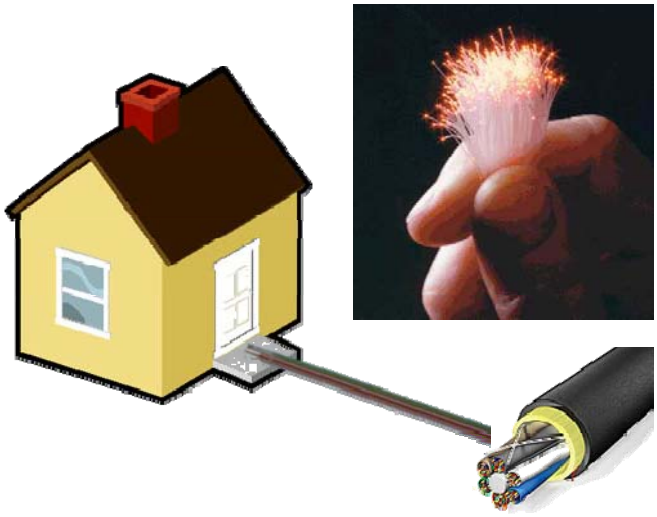
Massive Bündelung der Rechenleistung an zentraler Stelle

Internet als Vermittlungsinstanz

Motivierender Trend: Stetige Erhöhung der Bandbreite für Endnutzer



Hochgeschwindigkeit ins Haus



- Konvergenz TV, Telekommunikation und Internet
- Technologiewechsel → erhebliche Investitionen
  - wirtschaftliche Faktoren und Bedingungen

- Telefondraht → Internet → TV
- TV-Kabel → Telefon → Internet

→ Glasfaser (x1, x5, x10,…?)

„Tripleplay" (Sprache, Daten, Video)

Cloud-Computing

E-Mail wird beim Provider gespeichert

Google Mail

Cloud-Computing

Fotos werden bei **flickr** gespeichert

Cloud-Computing

Videos bei YouTube

Cloud-Computing

Private Dokumente werden bei einem Storage Provider abgelegt

Das Tagebuch wird öffentlich „im Netz" als Blog geführt

Cloud-Computing

Informieren tut man sich im Netz

Google

WIKIPEDIA
Die freie Enzyklopädie

Cloud-Computing

Vernetzen tut man sich bei „social net-works" oder „digital communities" im Netz

Cloud-Computing

Plattformen im Netz
nutzt man zum
- Kaufen
- Spielen
- Kommunizieren
- …

Google Romance BETA

Cloud-Computing

Vorteile für Nutzer:
- von überall zugreifbar
- keine Datensicherung
- keine Softwarepflege

← Kein PC, sondern billiges Web-Terminal, Smartphone etc.

Wie Wasserleitungen einst den eigenen Brunnen überflüssig machten…

Cloud-Computing

Voraussetzungen?

- Überall Breitband (fest & mobil)
- Netz-Verlässlichkeit (Versorgungssicherheit, Datenschutz,…)
- Wirtschaftlichkeit

Cloud-Computing

Plattformen:

- Wer betreibt sie? Wo?
- Wer verdient daran?
- Wer bestimmt?
- Wer kontrolliert?
- Welche Nationen profitieren davon?

Cloud-Computing

Auch gewöhnliche Dinge werden die Dienste in der „cloud" nutzen

„Breitband ist die Elektrizität des 21. Jahrhunderts“

- Wo aber sind dann die „Kraftwerke"?

Beispiel: Google-Datenzentren

- Jedes Datenzentrum hat 10 000 – 100 000 Computer
- Kostet über 500 Mio $ (Bau, Infrastruktur, Computer)
- Verbraucht 50 – 100 MW Energie (Strom, Kühlung)
- Neben Google weitere (z.B. Amazon, Microsoft, Ebay,…)

Google Data Center Groningen

Google Data Center Columbia River

Rückansicht: Energiezufuhr

Innenansicht

- Effizient wie Fabriken
  - Produkt: Internet-Dienste
- Kostenvorteil durch Skaleneffekt
  - Faktor 5 – 7 gegenüber traditionellen „kleinen" Rechenzentren
- Angebot nicht benötigter Leistung auf einem Spot-Markt

Das entwickelt sich zum eigentlichen Geschäft!

Zukünftige Container-Datenzentren

- Hunderte von Containern aus je einigen tausend Compute-Servern
  - mit Anschlüssen für Strom und Kühlung
- Nahe an Kraftwerken
  - Transport von Daten billiger als Strom

- Anmerkung zum Thema „Green IT":
  - IT-Infrastruktur verursachte 2007 weltweit ca. 830 Mio t CO_2 (2%)
  - IT-gestütztes Energie- und Gebäudemanagement sollte aber bis zu 4 Mal so viel einsparen

Cloud-Computing für die Industrie und Wirtschaft

- Spontanes Outsourcen von IT inklusive Geschäftsprozesse
  - Datenverarbeitung als Commodity
  - Software und Datenspeicher als Service
- Keine Bindung von Eigenkapital
  - Kosten nach „Verbrauch"
- Elastizität: Sofortiges Hinzufügen weiterer Ressourcen bei Bedarf
  - virtualisierte Hardware

Markt für „utility computing"
2010: ca. 95 Milliarden EUR

Charakteristika und "praktische" Probleme verteilter Systeme

- Räumliche Separation, autonome Komponenten

  → Zwang zur Kommunikation per Nachrichtenaustausch

  → neue Probleme:

  - partielles Fehlverhalten (statt totaler "Absturz")
  - fehlender globaler Zustand / exakt synchronisierte Zeit
  - Inkonsistenzen, z.B. zwischen Datei und Verzeichnis
  - konkurrenter Zugriff, Replikate, Cache,...

  Eingesetzt zur Realisierung von Leistungs- und Ausfalltransparenz

- Heterogenität
  - ist in gewachsenen Informationsumgebungen eine Tatsache
  - findet sich in Hard- und Software

- Dynamik, Offenheit
  - Gewährleistung von "Interoperabilität" ist nicht einfach

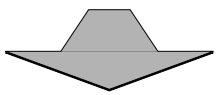
- Komplexität
  - Abstraktion als Mittel zur Beherrschung der Komplexität wichtig:
    a) Schichten (Kapselung, virtuelle Maschinen)
    b) Modularisierung (z.B. Services)
    c) "Transparenz"-Prinzip

- Sicherheit
  - Vertraulichkeit, Authenzitität, Integrität, Verfügbarkeit,...
  - notwendiger als in klassischen Einzelsystemen
  - aber schwieriger zu gewährleisten (mehr Angriffspunkte)

Aspekte verteilter Systeme

im Vergleich zu *sequentiellen* Systemen:

- Grösse und Komplexität → jede(r) ist anders
- Heterogenität → jede(r) ist anders
- Nebenläufigkeit → vieles gleichzeitig
- Nichtdeterminismus → morgen anders als heute
- Zustandsverteilung → niemand weiss alles



- Programmierung *komplexer*
- Test und Verifikation *aufwändiger*
- Verständnis der Phänomene *schwieriger*

⇒ gute Werkzeuge ("Tools") und Methoden
- z.B. Middleware als Software-Infrastruktur

⇒ adäquate Modelle, Algorithmen, Konzepte
- zur Beherrschung der "neuen" Phänomene

Ziel: Verständnis der grundlegenden Phänomene, Kenntnis der geeigneten Konzepte und Verfahren

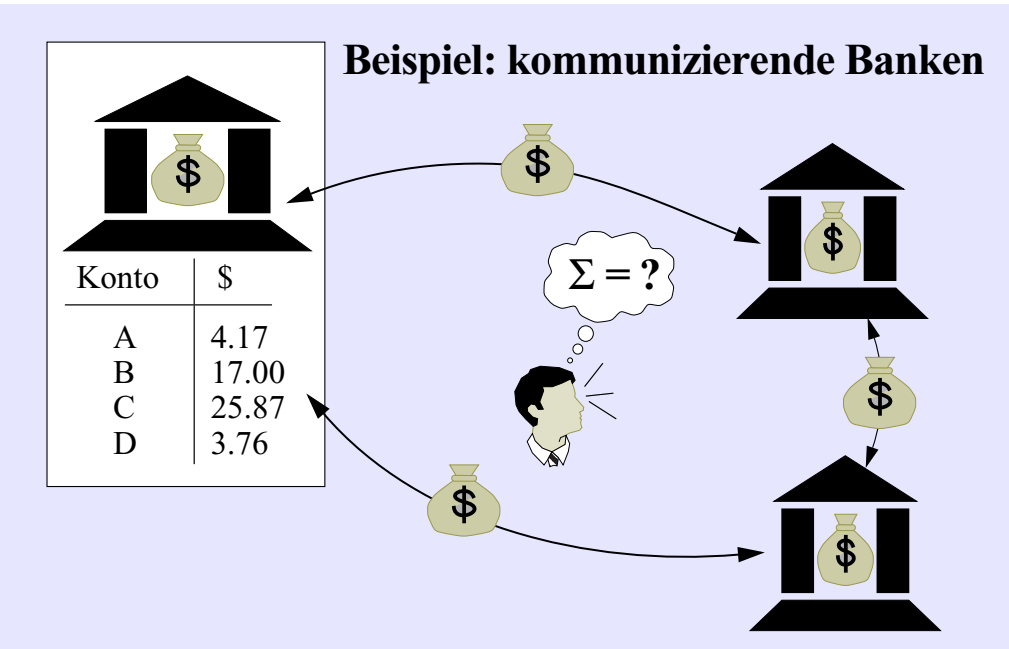
Einige *konzeptionelle* Probleme und Phänomene verteilter Systeme

1) Schnappschussproblem
2) Phantom-Deadlocks
3) Uhrensynchronisation
4) Kausaltreue Beobachtungen
5) Geheimnisvereinbarung über unsichere Kanäle

- Dies sind einige einfach zu erläuternde Probleme und Phänomene
- Es gibt noch viel mehr und viel komplexere Probleme
  - konzeptioneller Art
  - praktischer Art
- Achtung: Manches davon wird nicht hier, sondern in der Vorlesung "Verteilte Algorithmen" eingehender behandelt!

Ein erstes Beispiel: Wieviel Geld ist in Umlauf?

- konstante Geldmenge, oder
- monotone Inflation (→ Untergrenze)

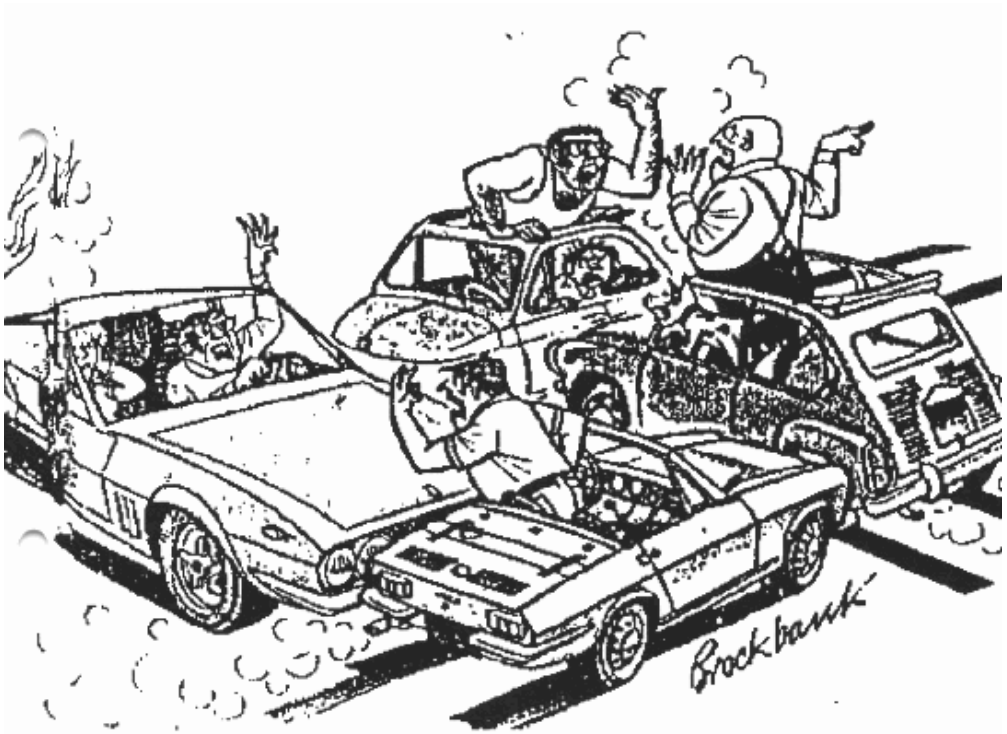


- Modellierung:
  - verteilte Geldkonten
  - ständige Transfers zwischen den Konten

- Erschwerte Bedingungen:
  - niemand hat eine globale Sicht
  - es gibt keine gemeinsame Zeit ("Stichtag")

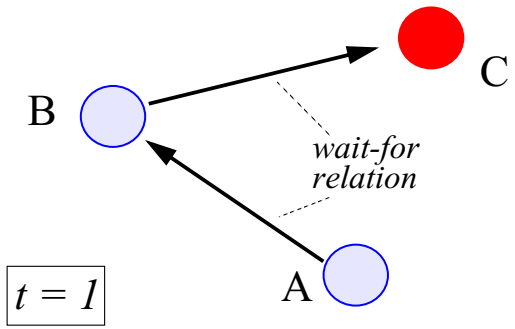
- Anwendung: z.B. verteilte DB-Sicherungspunkte

Ein zweites Beispiel: Das Deadlock-Problem



Das Deadlock-Problem

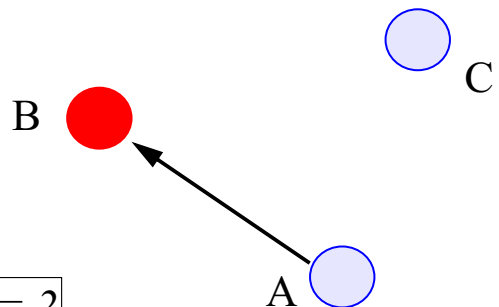
Phantom-Deadlocks



$t = 1$

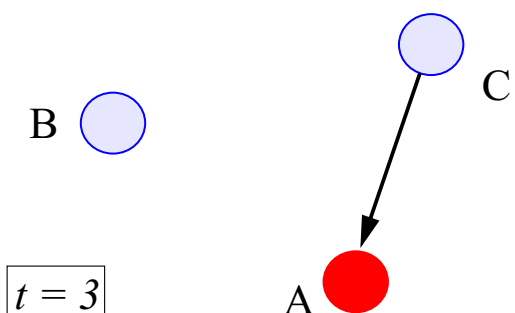
(C benutzt ein exklusives Betriebsmittel)

beobachte B:
⇒ B wartet auf C



$t = 2$

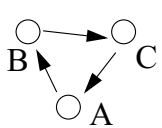
beobachte A:
⇒ A wartet auf B



$t = 3$

beobachte C:
⇒ C wartet auf A

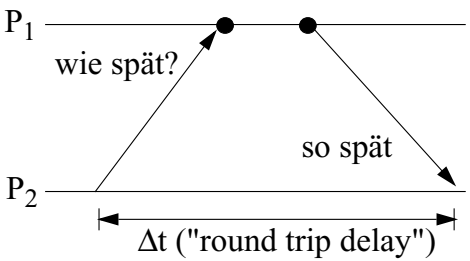
Keine exakte globale Zeit!



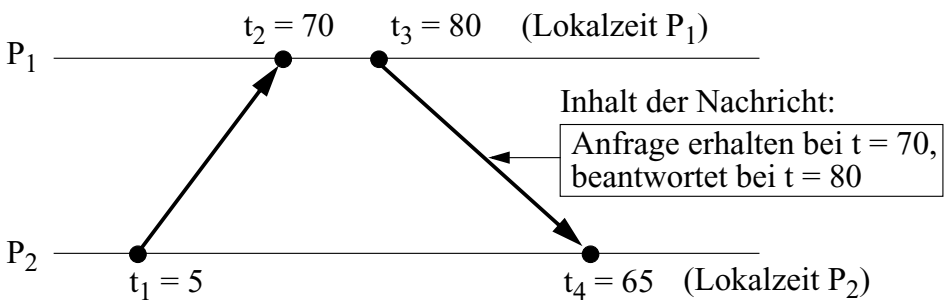
falscher Schluss!

Deadlock!

Ein drittes Problem: Uhrensynchronisation



- Lastabhängige Laufzeiten von Nachrichten
- Unsymmetrische Laufzeiten
- Wie erfährt man die Laufzeit?

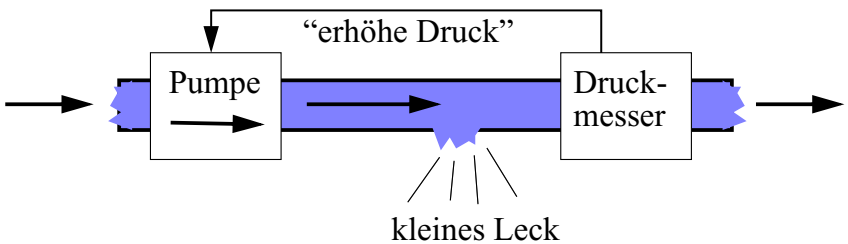


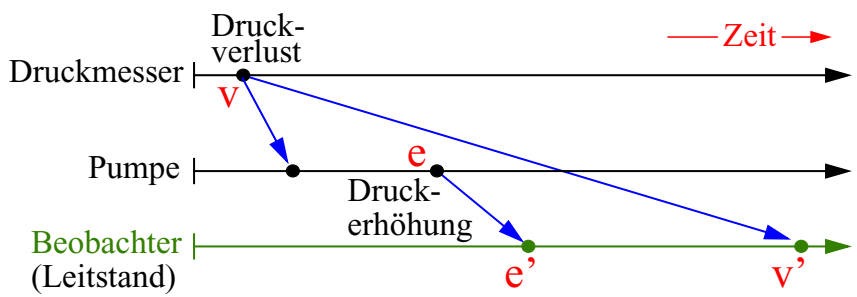
- Uhren gehen nicht unbedingt gleich schnell!
  (wenigstens "Beschleunigung ≈ 0", d.h. konstanter Drift gerechtfertigt?)

- Wie kann man den Offset der Uhren ermitteln oder zumindest approximieren?

Ein viertes Problem: (nicht) kausaltreue Beobachtungen

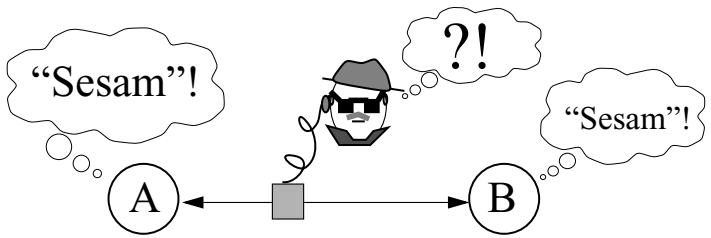
- Gewünscht: Eine Ursache stets vor ihrer (u.U. indirekter) Wirkung beobachten





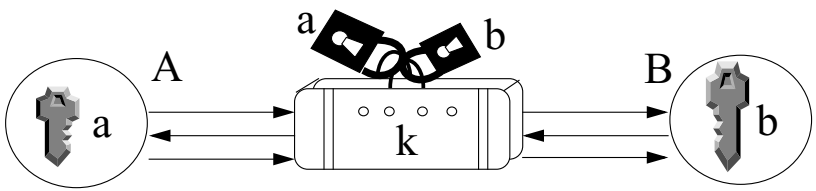
Falsche Schlussfolgerung des Beobachters:
Es erhöhte sich der Druck (aufgrund einer unbegründeten Aktivität der Pumpe), es kam zu einem Leck, was durch den abfallenden Druck angezeigt wird.

Und noch ein Problem: Verteilte Geheimnisvereinbarung



- Problem: A und B wollen sich über einen unsicheren Kanal auf ein gemeinsames geheimes Passwort einigen.

- Idee: Vorhängeschlösser um eine sichere Truhe:



1. A denkt sich Passwort k aus und tut es in die Truhe.
2. A verschliesst die Truhe mit einem Schloss a.
3. A sendet die so verschlossene Truhe an B.
4. B umschliesst das ganze mit seinem Schloss b.
5. B sendet alles doppelt verschlossen an A zurück.
6. A entfernt Schloss a.
7. A sendet die mit b verschlossene Truhe wieder an B.
8. B entfernt sein Schloss b.

- Problem: Lässt sich das so softwaretechnisch realisieren?

Wie wäre es damit?: k sei eine Zahl. “Verschliessen” und “aufschliessen” eines Schlosses entspricht dem Hinzuaddieren oder Subtrahieren einer beliebig ausgedachten (geheimgehaltenen) Zahl a bzw. b.