Verteilte Systeme

Friedemann Mattern

Prüfungsrelevant ist der Inhalt der Vorlesung, nicht alleine der Text dieser Foliensammlung!

ETH zürich

Wer bin ich? Wer sind wir?

Prof. Friedemann Mattern

+ mehrere Assistenten und Assistentinnen

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

Wer bin ich? Wer sind wir?

Prof. Friedemann Mattern

+ mehrere Assistenten und Assistentinnen

Matthias Kovatsch
Ansprechperson für organisatorische Aspekte
kovatsch@inf.ethz.ch

Fachgebiet „Verteilte Systeme“
im Departement Informatik,
Institut für Pervasive Computing

Mit was beschäftigen wir uns?

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

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

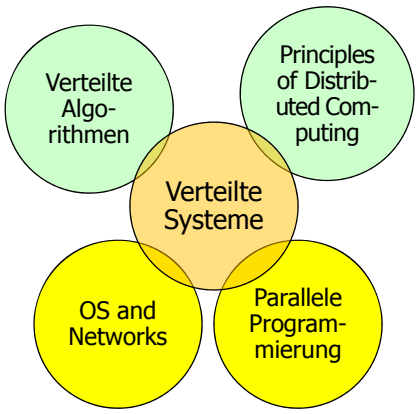
Organisatorisches zur Vorlesung

- Format: 6G+1A: Vorlesung und Praktikum integriert
  - Mo 9:15 - 12:00, Fr 9:15 - 12:00, jew. NO C 6
  - Praktikum ist inhaltlich *komplementär zur Vorlesung* (mobile Kommunikationsplattformen: Android, HTC Desire etc.)
  - Gelegentliche *Assistentenstunden (zu den "Vorlesungsterminen")* zur Besprechung der Praktikumsaufgaben und Vertiefung des Stoffes
  - Gelegentliche *Denkaufgaben (ohne Lösung...) in der Vorlesung*
- Sinnvolle Vorkenntnisse (Grundlagen)
  - 4 Semester der Bachelorstufe Informatik
  - Grundkenntnisse Computernetze und Betriebssysteme (z.B. Prozessbegriff, Synchronisation)
  - UNIX, Java ist hilfreich

Organisatorisches (2)

- Folienkopien jeweils einige Tage nach der Vorlesung
  - im pdf-Format bei www.vs.inf.ethz.ch/edu
- Prüfung schriftlich
  - bewertete Praktikumsaufgaben gehen in die Prüfungsnote ein
- Vorlesung ab November: Prof. Roger Wattenhofer

Einordnung der Vorlesung



- „Verteilte Systeme" ist ein Querschnittsthema
- Gewisse Überschneidungen mit anderen Vorlesungen unvermeidlich

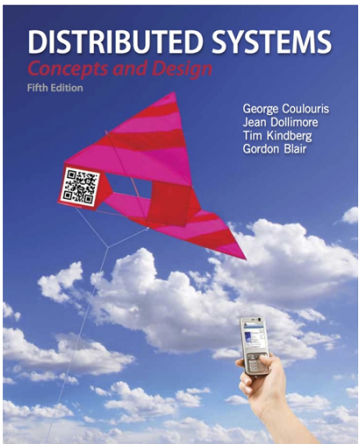
Thematisch verwandte Veranstaltungen (Masterstufe)

- Ubiquitous Computing
- Enterprise Application Integration - Middleware
- Web Services and Service Oriented Architectures
- Verteilte Algorithmen
- Principles of Distributed Computing
- Distributed Information Systems
- Advanced Operating Systems
- Fehlertoleranz in Verteilten Systemen

Es werden nicht immer alle Vorlesungen angeboten!

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

Literatur



- G. Coulouris, J. Dollimore, T. Kindberg, G. Blair: Distributed Systems: Concepts and Design (5th ed.). Addison-Wesley, 2011
- A. Tanenbaum, M. van Steen: Distributed Systems: Principles and Paradigms (2nd ed.). Prentice-Hall, 2007
- O. Haase: Kommunikation in verteilten Anwendungen (2. Aufl.). R. Oldenbourg Verlag, 2008
- A. Schill, T. Springer: Verteilte Systeme (2. Aufl.). Springer Vieweg, 2011

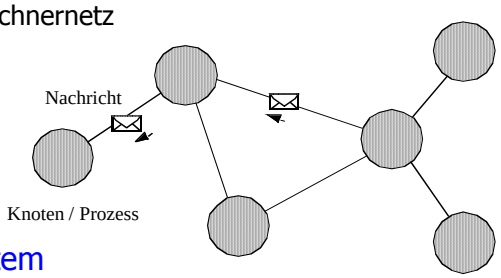
„Verteiltes System“ – zwei Definitionen

- 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?

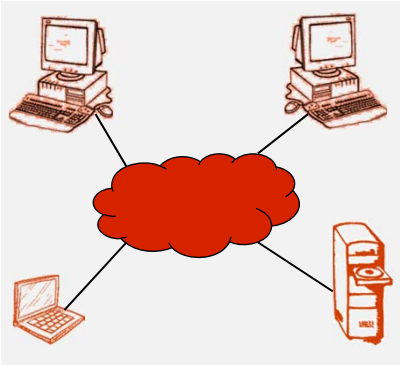
„Verteiltes System“

- Physisch verteiltes System
  - Compute-Cluster ... Rechnernetz



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

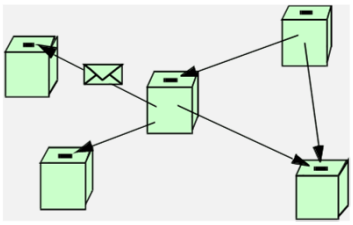
Sichten verteilter Systeme (1)



- Computernetz mit “Rechenknoten”, z.B.
  - Compute-Cluster
  - Local Area Network
  - Internet
- Relevante Aspekte:
  - Routing, Adressierung,…

Zunehmende Abstraktion

Sichten verteilter Systeme (2)

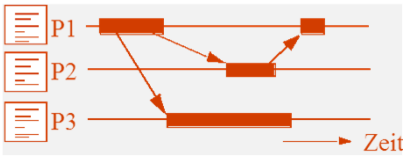


kommunizierende Prozesse,
kooperierende Objekte

- Objekte in Betriebs-systemen, Middleware, Programmiersprachen
- "Programmierersicht"
  - z.B. Client mit API zu Server

Zunehmende Abstraktion ➡

Sichten verteilter Systeme (3)



- Algorithmen- und Protokollebene
  - Aktionen, Ereignisfolgen
  - Konsistenz, Korrektheit

- Man kann verteilte Systeme auf verschiedenen Abstraktionsstufen betrachten
- Es sind dabei jeweils unterschiedliche Aspekte relevant und interessant

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 physische Systeme
  - z.B. Steuerung einer Fabrik, Zweigstellennetz einer Bank (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

- Outsourcing von Diensten, Verlagerung in eine „Cloud", kann günstiger sein als eine lokal-zentralisierte Lösung
- Compute-Cluster manchmal besseres Preis-Leistungs-verhältnis als Supercomputer

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

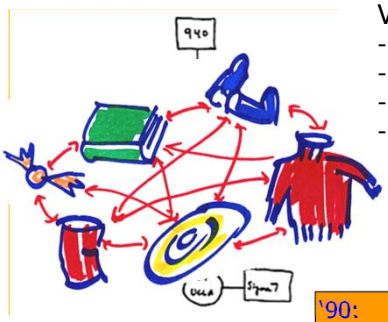
Historische Entwicklung (1)

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

Historische Entwicklung (2)

- Workstation- und PC-Netze (LAN)
  - bahnbrechende, frühe Ideen bei XEROX-PARC (XEROX-Star als erste Workstation, Desktop-Benutzerinterface, Ethernet, RPC, verteilte Dateisysteme,...)
- Kommerzielle Pionierprojekte als Treiber (WAN)
  - z.B. Reservierungssysteme, Banken, Kreditkarten
- Web / Internet als Plattform
  - für electronic commerce etc.
  - web services
  - neue, darauf aufbauende Dienste
- Mobile Geräte
  - z.B. Smartphones
- Internet der Dinge

Änderung der Vernetzungsqualität aus historischer Sicht



Vernetzung von:
- Computern (ftp)
- Dokumenten (WWW)
- Menschen (soz. Netze)
- Dingen (Internet der Dinge)

'80: E-Mail

'90: WWW

'00: Mobiler Zugang

Eingebettete Internet-Dienste

Zeit

TCP http Web 2.0

Historie von Konzepten

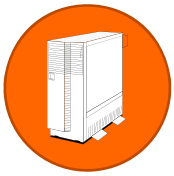
- Concurrency, Synchronisation
  - war bereits klassisches Thema bei Datenbanken und Betriebssystemen
- Programmiersprachen
  - „kommunizierende" Objekte
- Parallele und verteilte Algorithmen
- Semantik von Kooperation / Kommunikation
  - mathematische Modelle für Verteiltheit (z.B. CCS, Petri-Netze)
- 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!

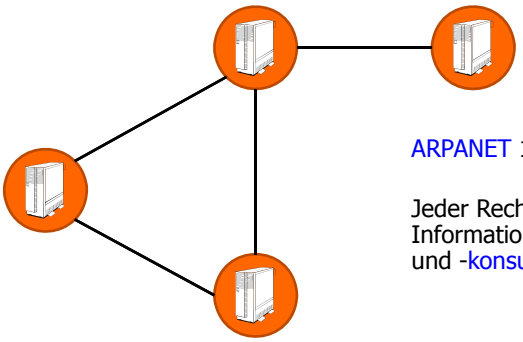
Architekturen verteilter Systeme

- Zu Anfang waren Systeme monolithisch



- Nicht verteilt / vernetzt
- Mainframes
- Terminals als angeschlossene „Datensichtgeräte" („Datenendgerät": Fernschreiber [tty], ASCII)

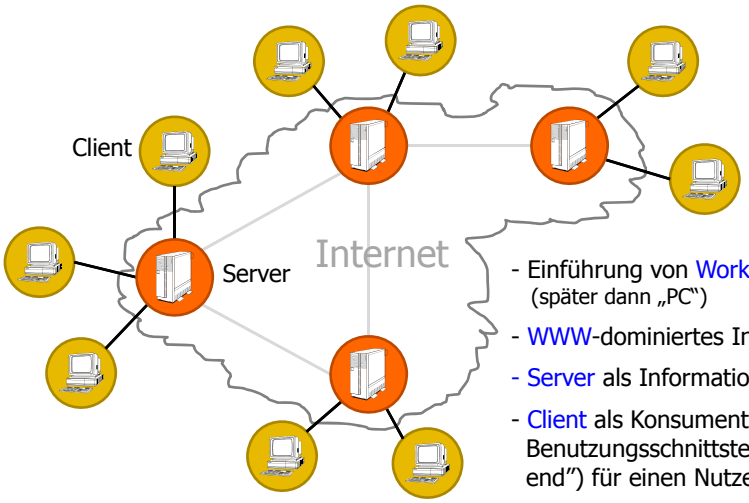
Architekturen verteilter Systeme: Peer-to-Peer



ARPANET 1969

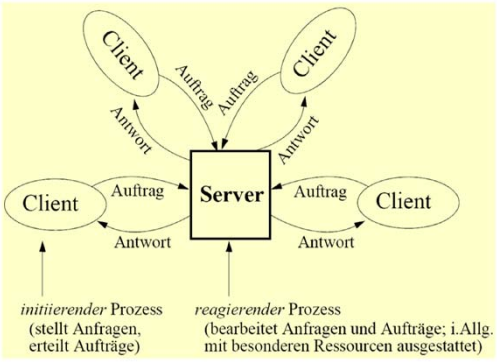
Jeder Rechner gleichzeitig Informationsanbieter und -konsument

Architekturen verteilter Systeme: Client-Server



- Einführung von Workstations (später dann „PC")
- WWW-dominiertes Internet
- Server als Informationsanbieter
- Client als Konsument; gleichzeitig Benutzungsschnittstelle ("front end") für einen Nutzer

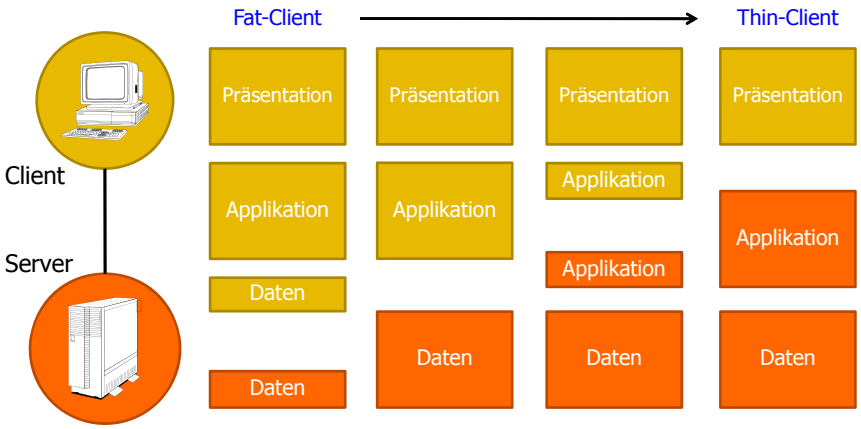
Client-Server



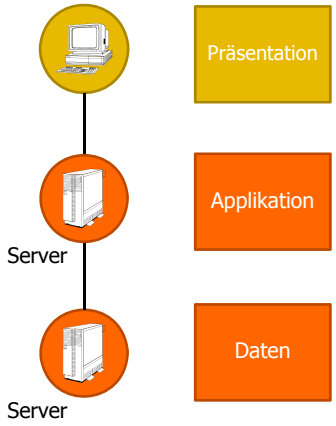
Populär auch wegen des eingängigen Modells

- entspricht Geschäftsvorgängen in unserer Dienstleistungsgesellschaft
- gewohntes Muster → intuitive Struktur, gute Überschaubarkeit

Architekturen verteilter Systeme: Fat- und Thin-Client



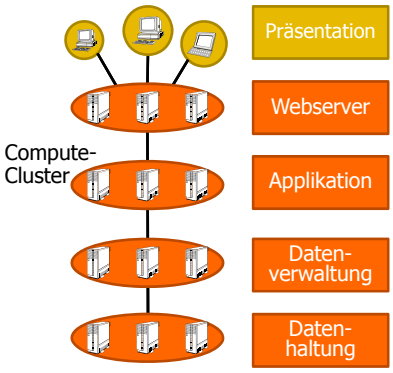
Architekturen verteilter Systeme: 3-Tier



Verarbeitung wird auf mehrere physikalische Einheiten verteilt

- Logische Schichten mit minimierten Abhängigkeiten
- Leichtere Wartung
- Einfaches Austauschen

Architekturen verteilter Systeme: Multi-Tier

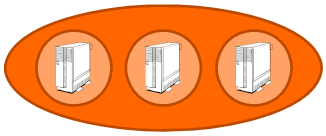


Weitere Schichten sowie mehrere physikalische Einheiten pro Schicht („Compute-Cluster") erhöhen die Skalierbarkeit und Flexibilität

Mehrere Webserver ermöglichen z.B. Lastverteilung

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

Architekturen verteilter Systeme: Compute-Cluster



Vernetzung kompletter Einzelrechner

Räumlich konzentriert (wenige Meter)

Sehr schnelles Verbindungsnetz

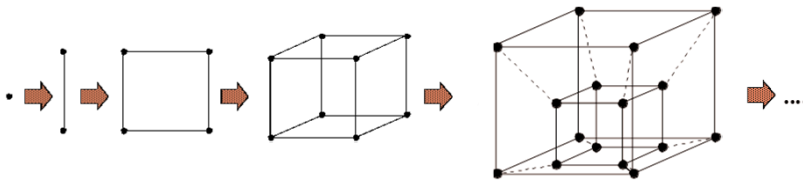
Es gibt diverse Netztopologien, um die Einzelrechner *(als Knoten in einem Graphen)* miteinander zu verbinden – diese sind unterschiedlich hinsichtlich

- Skalierbarkeit der Topologie
- Routingkomplexität
- Gesamtzahl der Einzelverbindungen
- maximale bzw. durchschnittliche Entfernung zweier Knoten
- Anzahl der Nachbarn eines Knotens
- …

Bestimmt mit die Kosten und die Leistungsfähigkeit eines Systems

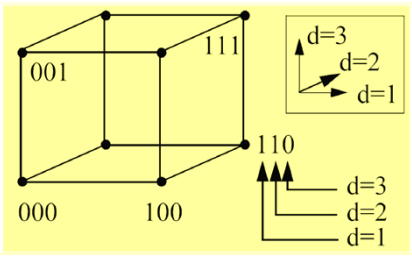
Beispiel: Hypercube-Verbindungstopologie

- Würfel der Dimension d
  - Vorteil: einfaches Routing, kurze Weglängen
  - Nachteil: Viele Einzelverbindungen (O(n log n) bei n Knoten)
- Rekursives Konstruktionsprinzip
  - Hypercube der Dimension 0: Einzelrechner
  - Hypercube der Dimension d+1: *„Nimm zwei Würfel der Dimension d und verbinde korrespondierende Ecken"*



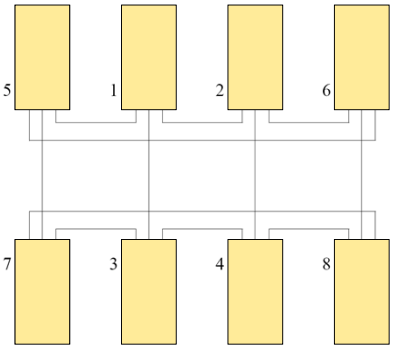
Routing beim Hypercube

- Knoten systematisch nummerieren (entspr. rekursivem Aufbau)
- Zieladresse bitweise xor mit Absenderadresse
- Wo sich eine "1" findet, in diese Dimension muss gewechselt werden

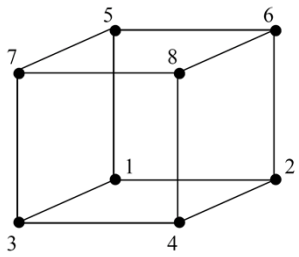


- Maximale Weglänge: d; durchschnittliche Weglänge = d/2
  (Induktionsbeweis als einfache Übung)

Hypercube

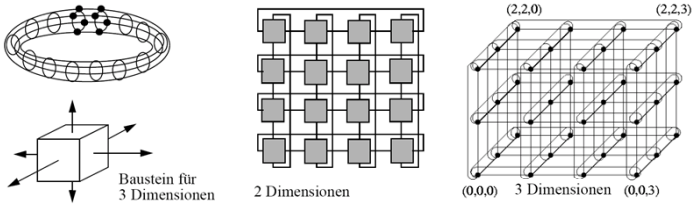


- Dies stellt auch einen Hypercube dar
  - ist nur nicht als Würfel gezeichnet



Eine andere Verbindungstopologie: Der d-dimensionale Torus

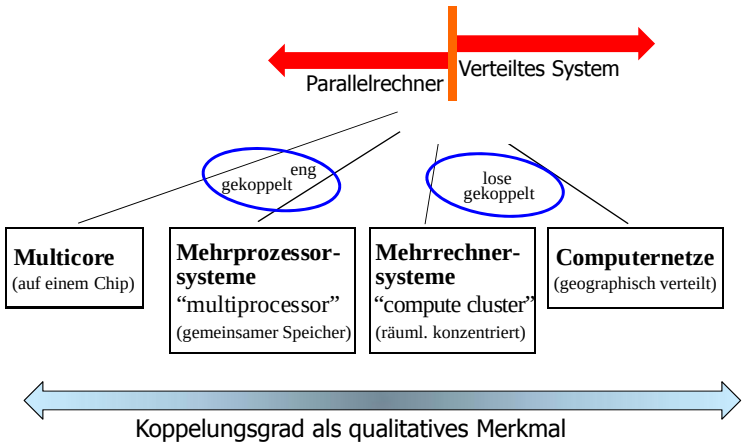
- Gitter in d Dimensionen mit "wrap-around"



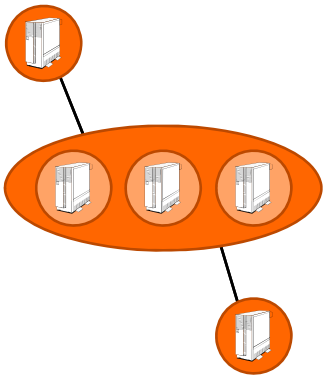
- Rekursives Konstruktionsprinzip: Nimm w_d gleiche Exemplare der Dimension d-1 und verbinde korrespondierende Elemente zu einem Ring
  - Sonderfall einfacher Ring: d = 1
  - Sonderfall Hypercube: d-dimensionaler Torus mit $w_i = 2$ für alle Dimensionen i

Es gibt noch einige andere sinnvolle Verbindungstopologien (auf die wir nicht eingehen)

Parallelrechner ↔ verteiltes System



Architekturen verteilter Systeme: Service-Oriented Architecture (SOA)



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

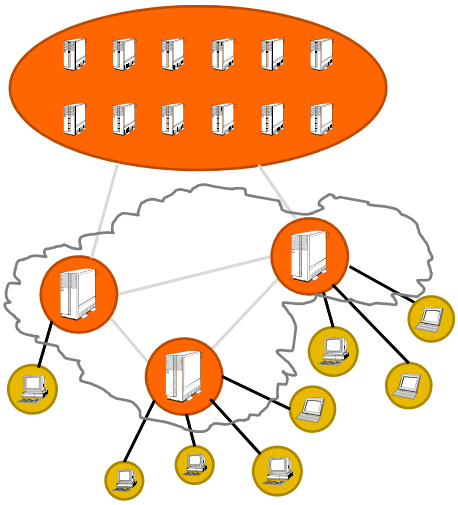
Lose Kopplung zwischen Services über Nachrichten und events (statt RPC = Remote Procedure Call)

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

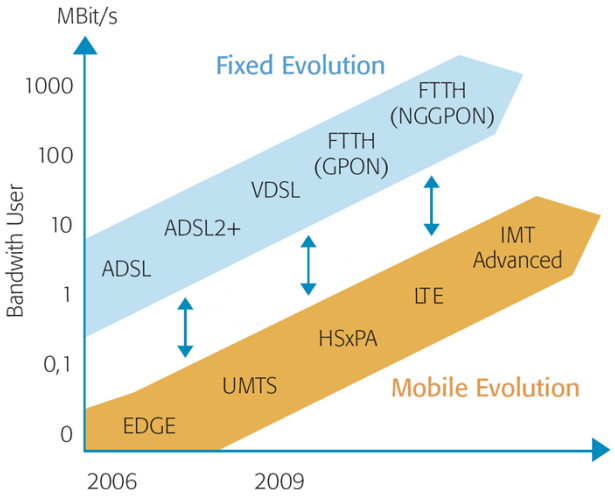


Massive Bündelung der Rechen-leistung an zentraler Stelle

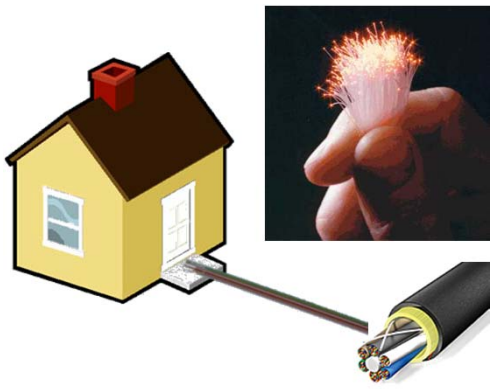
Outsourcen von Applikationen in die Cloud

Internet im Wesentlichen nur noch 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

Cloud-Computing …für Privatnutzer

E-Mail wird beim Provider gespeichert

Google Mail

Cloud-Computing …für Privatnutzer

Fotos werden bei **flickr** gespeichert

Videos bei YouTube

Cloud-Computing …für Privatnutzer

Private Dokumente werden bei einem Storage Provider abgelegt

Das Tagebuch wird öffentlich „im Netz“ als Blog geführt

Cloud-Computing …für Privatnutzer

Informieren tut man sich im Netz

Google

WIKIPEDIA
Die freie Enzyklopädie

Cloud-Computing …für Privatnutzer

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

Cloud-Computing …für Privatnutzer

Plattformen im Netz nutzt man zum

- Kaufen
- Spielen
- Kommunizieren
- …

Google Romance BETA

Cloud-Computing …für Privatnutzer

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?

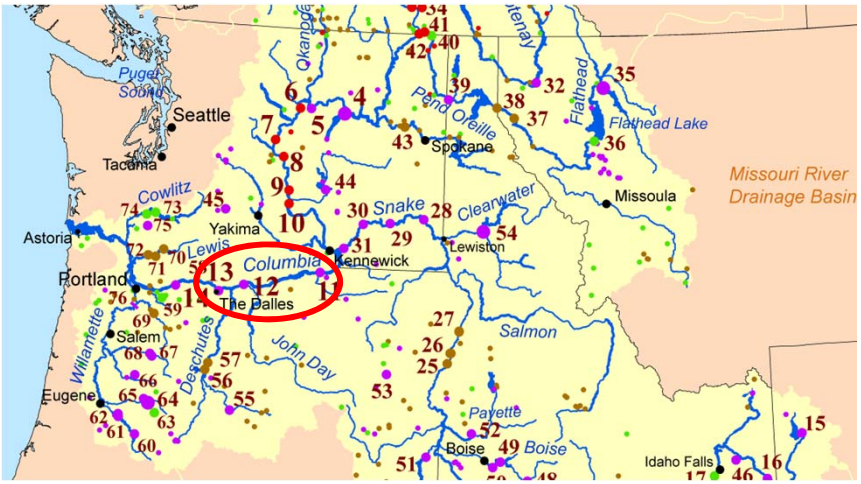
Beispiel: Google-Datenzentren

- 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

The Dalles, OR, Columbia River



Google Data Center Columbia River

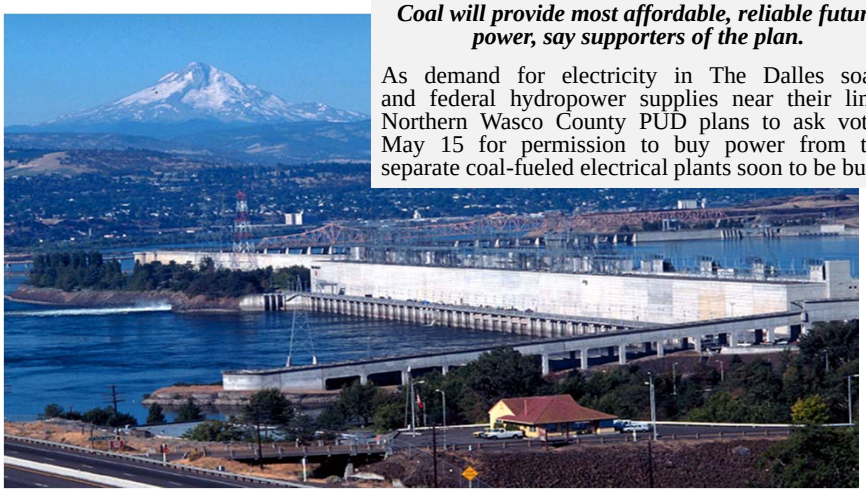
Energiezufuhr „Discovery Substation“ : 115 kV / 13.8 kV

Nahes Kraftwerk: Dalles Dam Power Station, Columbia River

The Chronicle, March 1, 2007 –
PUD to seek vote on coal power

Coal will provide most affordable, reliable future power, say supporters of the plan.

As demand for electricity in The Dalles soars, and federal hydropower supplies near their limit, Northern Wasco County PUD plans to ask voters May 15 for permission to buy power from two separate coal-fueled electrical plants soon to be built.



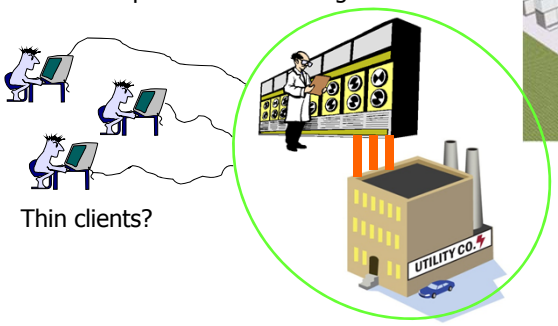
Innenansicht eines Cloud-Zentrums

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

Das entwickelt sich zu einem wesentlichen Geschäft!

Zukünftige Container-Datenzentren

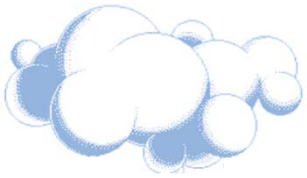
- Hunderte von Containern aus je einigen tausend Compute-Servern
  - mit Anschlüssen für Strom und Kühlung
- Nahe an Kraftwerken
  - Transport von Daten billiger als Strom



Thin clients?

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



Architekturen verteilter Systeme: Zusammenfassung

- Peer-to-Peer
- Client-Server (Fat-Client vs. Thin Client)
- 3-Tier
- Multi-Tier
- Service-Oriented Architecture (SOA)
- Compute-Cluster
- Cloud-Computing

Charakteristika und Problemaspekte verteilter Systeme

- Räumliche Separation und Autonomie der Komponenten führen (relativ zu zentralen Systemen) zu neuen Problemen:
  - partielles Fehlverhalten möglich (statt totaler "Absturz")
  - fehlender globaler Zustand / exakt synchronisierte Zeit
  - mögliche Inkonsistenzen (z.B. zwischen Datei und Verzeichnis / Index)
- Typischerweise Heterogenität in Hard- und Software
- Hohe Komplexität
- Sicherheit (Vertraulichkeit, Authentizität, Integrität, Verfügbarkeit,...)
  - notwendiger als in isolierten Einzelsystemen
  - aber schwieriger zu gewährleisten (mehr Angriffspunkte)

Gegenmittel?

- Gute Werkzeuge ("Tools") und Methoden
  - z.B. Frameworks und Middleware als Software-Infrastruktur
- Abstraktion als Mittel zur Beherrschung von Komplexität
  - z.B. Schichten (Kapselung, virtuelle Maschinen) oder
  - Modularisierung (z.B. Services)
- Adäquate Modelle, Algorithmen, Konzepte
  - zur Beherrschung der Phänomene rund um die Verteiltheit

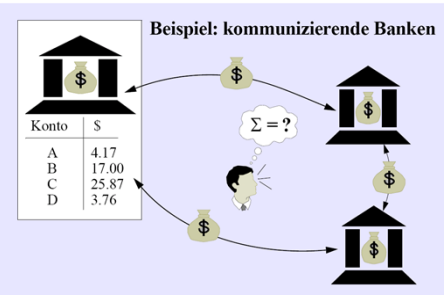
- Ziel der Vorlesung
  - Verständnis der grundlegenden Phänomene
  - Kenntnis von geeigneten Konzepte und Methoden

Einige konzeptionelle Probleme und Phänomene verteilter Systeme

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

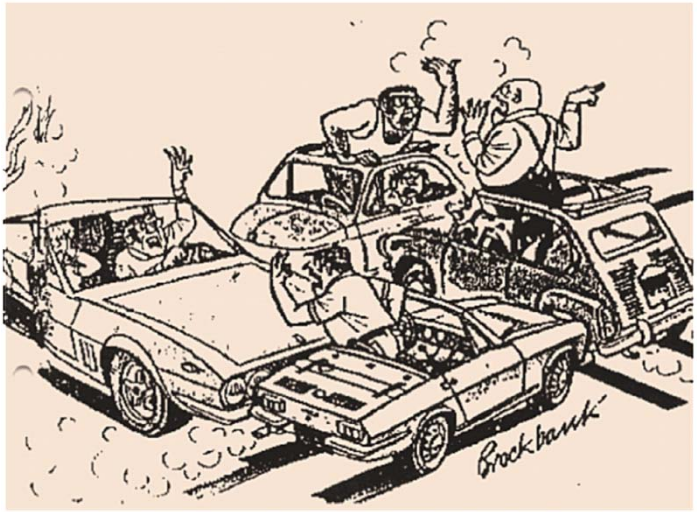
- Dies sind einige einfach zu erläuternde Probleme und Phänomene – natürlich gibt es noch viel mehr und viel komplexere Probleme konzeptioneller wie praktischer Art
- Achtung: Manches davon wird nicht hier, sondern in anderen Vorlesungen (z.B. "Verteilte Algorithmen") eingehender behandelt!

Ein erstes Beispiel: Wieviel Geld ist in Umlauf?

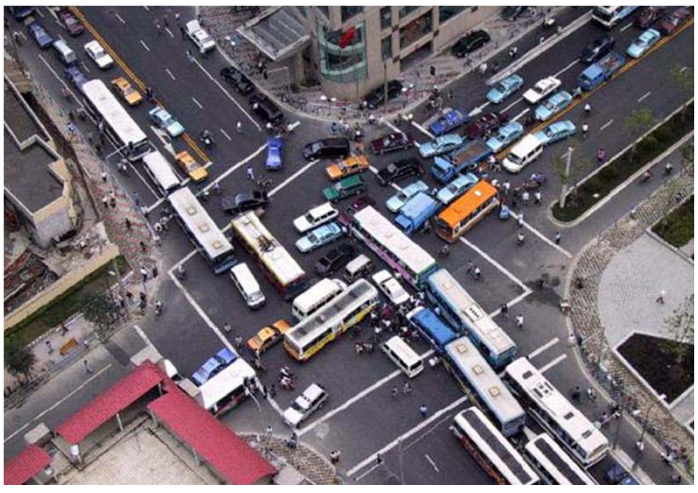


- Hier: konstante Geldmenge
- Ständige Transfers zwischen den Banken
- Niemand hat eine globale Sicht
- Es gibt keine gemeinsame Zeit ("Stichtag")
- Anwendung: z.B. verteilte Datenbank-Sicherungspunkte

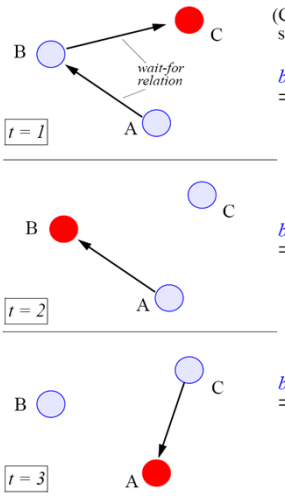
Ein zweites Beispiel: Das Deadlock-Problem



Ein zweites Beispiel: Das Deadlock-Problem



Phantom-Deadlocks



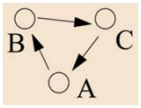
(C benutzt ein exklusives Betriebsmittel)

beobachte B:
⇒ B wartet auf C

beobachte A:
⇒ A wartet auf B

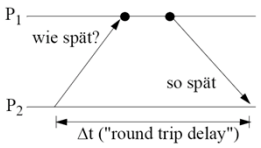
beobachte C:
⇒ C wartet auf A

- Aus den Einzelbeobachtungen darf man nicht schliessen:
- A wartet auf B und B wartet auf C und C wartet auf A

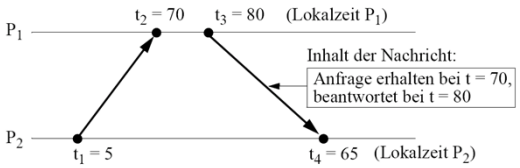


- Diese zyklische Wartebedingung wäre tatsächlich ein Deadlock
- Die Einzelbeobachtungen fanden hier aber zu unterschiedlichen Zeiten statt
- Lösung (nur echte Deadlocks erkennen) ohne Uhren, globale Zeit, Zeitstempel etc.?

Ein drittes Problem: Uhrensynchronisation



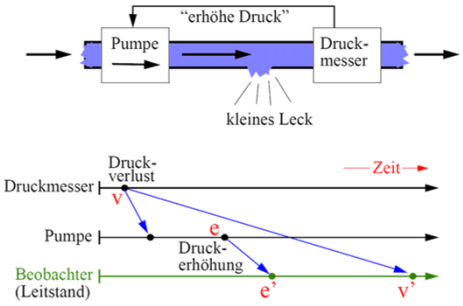
Laufzeit der Nachrichten variabel (und oft verschieden für Hin- und Rückweg)



- Uhren gehen nicht unbedingt gleich schnell!
  - Gilt wenigstens "Beschleunigung ≈ 0", d.h. ist 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. indirekten) Wirkung beobachten

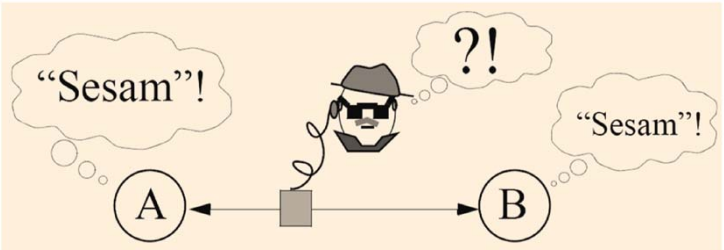


Falsche Schlussfolgerung des Beobachters:

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

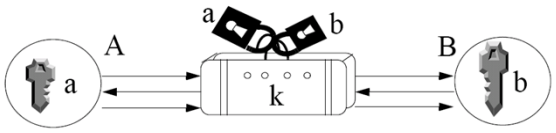
Und noch ein Problem: Verteilte Geheimnisvereinbarung

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



Verteilte Geheimnisvereinbarung (2)

- 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?

Kommunikation

Kooperation durch Informationsaustausch

- Prozesse sollen kooperieren, daher untereinander Information austauschen können
  - falls vorhanden, evtl. über einen gemeinsamen globalen Speicher (dieser kann physisch oder evtl. nur logisch existieren als „virtual shared memory")
  - oder mittels Nachrichten:
    Daten an eine entfernte Stelle kopieren

Kommunikation

Notwendig, damit Kommunikation klappt, ist jedenfalls:

1. ein dazwischenliegendes physikalisches Medium
   - z.B. elektrische Signale in Kupferkabeln (oder der „Äther"?)
2. einheitliche Verhaltensregeln
   - Kommunikationsprotokolle
3. gemeinsame Sprache und gemeinsame Semantik
   - gleiches Verständnis der Bedeutung von Kommunikationskonstrukten und -regeln

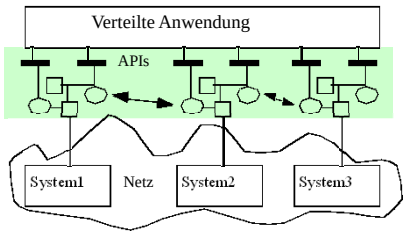
Also trotz Verteiltheit gewisse gemeinsame Aspekte!

Nachrichtenbasierte Kommunikation

- send → receive
- Implizite Synchronisation: Senden vor Empfangen
  - Empfänger erfährt, wie weit der Sender mindestens ist
- Nachrichten sind dynamische Betriebsmittel
  - verursachen Aufwand und müssen verwaltet werden

Message Passing System (1)

- Organisiert den Nachrichtentransport
- Bietet Kommunikationsprimitive (als APIs) an
  - z.B. send (...) bzw. receive (...)
  - evtl. auch ganze Bibliothek verschiedener Kommunikationsdienste
  - verwendbar mit gängigen Programmiersprachen (C, Java,...)



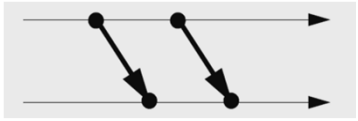
- Besteht aus Hilfsprozessen, Pufferobjekten, ...
- Verbirgt Details des zugrundeliegenden Netzes bzw. Kommunikationssubsystems

Message Passing System (2)

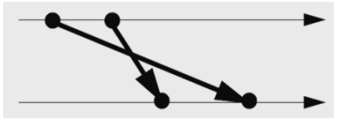
- Verwendet vorhandene Netzprotokolle und implementiert damit eigene, „höhere" Protokolle
- Garantiert (je nach „Semantik") gewisse Eigenschaften
  - z.B. Reihenfolgeerhalt oder Prioritäten von Nachrichten
- Abstrahiert von Implementierungsaspekten
  - z.B. Geräteadressen oder Längenrestriktionen von Nachrichten etc.
- Maskiert gewisse Fehler
  - mit typischen Techniken zur Erhöhung des Zuverlässigkeitsgrades: Timeouts, Quittungen, Sequenznummern, Wiederholungen, Prüfsummen, fehlerkorrigierende Codes,…
- Verbirgt Heterogenität unterschiedlicher Systemplattformen
  - erleichtert damit Portabilität von Anwendungen

Ordnungserhalt von Nachrichten: FIFO

- Manchmal werden vom Kommunikationssystem Garantien bzgl. Nachrichtenreihenfolgen gegeben
- Eine solche Garantie stellt z.B. FIFO (First-In-First-Out) dar: Nachrichten zwischen zwei Prozessen (also auf dem Kommunikationskanal zwischen Sender und Empfänger) überholen sich nicht: Empfangsreihenfolge = Sendereihenfolge



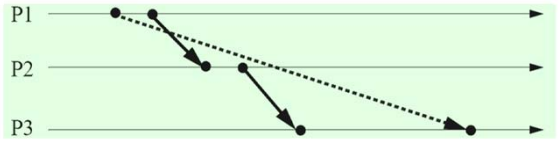
FIFO



kein FIFO

Ordnungserhalt von Nachrichten: kausale Ordnung

- FIFO verbietet allerdings nicht, dass Nachrichten evtl. (über eine Kette anderer Nachrichten) indirekt überholt werden



Zwar FIFO, aber nicht kausal geordnet

- Möchte man auch dies haben, so muss die Kommunikation kausal geordnet sein (Anwendungszweck?)
  - keine Information erreicht Empfänger auf Umwegen schneller als auf direktem Wege („Dreiecksungleichung")
  - entspricht einer „Globalisierung" von FIFO auf mehrere Prozesse
  - Denkübung: Wie garantiert (d.h. implementiert) man kausale Ordnung auf einem System ohne Ordnungsgarantie?

Prioritäten von Nachrichten? (1)

- Achtung: Semantik ist a priori nicht ganz klar:
  - Soll (kann?) das Transportsystem Nachrichten höherer Priorität bevorzugt (=?) befördern?
  - Können (z.B. bei fehlender Pufferkapazität) Nachrichten niedrigerer Priorität überschrieben werden?
  - Wie viele Prioritätsstufen gibt es?
  - Sollen aus einer Mailbox (= Nachrichtenspeicher) immer zuerst Nachrichten mit höherer Priorität geholt werden?

Prioritäten von Nachrichten? (2)

- Mögliche Anwendungen:
  - Unterbrechen / abbrechen laufender Aktionen (→ Interrupt)
  - Aufbrechen von Blockaden
  - Out-of-Band-Signalisierung

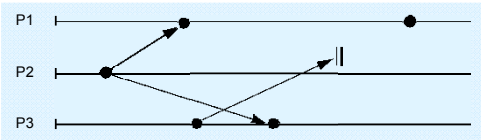
  } Durchbrechung der FIFO-Reihenfolge!

Vgl. auch Service-Klassen in Computernetzen: bei Rückstaus bei den Routern soll z.B. interaktiver Verkehr bevorzugt werden vor FTP etc.

- Vorsicht bei der Anwendung: Nur bei klarer Semantik verwenden; löst oft ein Problem nicht grundsätzlich!
  - Inwiefern ist denn eine (faule) Implementierung, bei der „eilige" Nachrichten (insgeheim) wie normale Nachrichten realisiert werden, tatsächlich „Betrug" am Anwender?

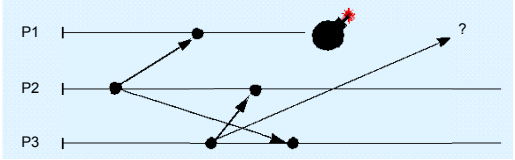
Fehlermodelle (1)

- Zweck: Klassifikation von Fehlermöglichkeiten; Abstraktion von den tieferliegenden spezifischen Ursachen
- Nachrichtenfehler beim Senden / Übertragen / Empfangen:



→ verlorene Nachricht

- Crash / Fail-Stop: Ausfall eines Prozessors:



→ Nicht mehr erreichbarer / mitspielender Prozess

Fehlermodelle (2)

- Zeitfehler: Ereignis geschieht zu spät (oder zu früh)
- „Byzantinische" Fehler: Beliebiges Fehlverhalten, z.B.:
  - verfälschte Nachrichteninhalte
  - Prozess, der unsinnige Nachrichten sendet

(solche Fehler lassen sich nur teilweise, z.B. durch Redundanz, erkennen)

- Fehlertolerante Algorithmen bzw. Systeme müssen das „richtige" Fehlermodell berücksichtigen!
  - adäquate Modellierung der realen Situation / des Einsatzgebietes
  - Algorithmus verhält sich korrekt nur relativ zum Fehlermodell