Folienkopien zur Vorlesung

# Verteilte Systeme

Friedemann Mattern

Departement Informatik, ETH Zürich

Wintersemester 2006/07 (5-stündig inkl. Übungen)



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

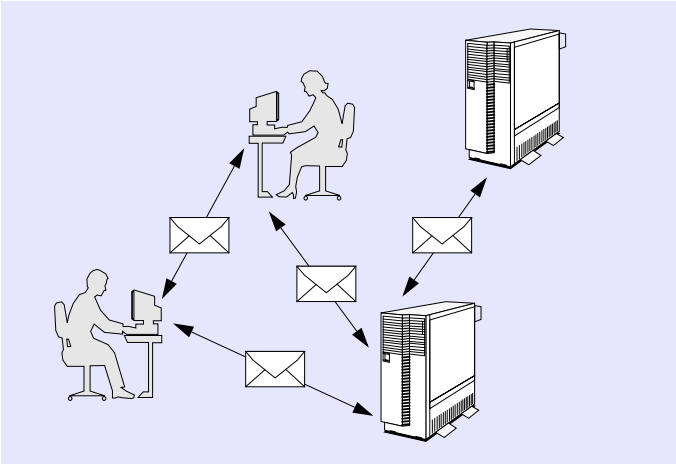




# Verteilte Systeme

**Friedemann Mattern**

Institut für Pervasive Computing
Departement Informatik
ETH Zürich

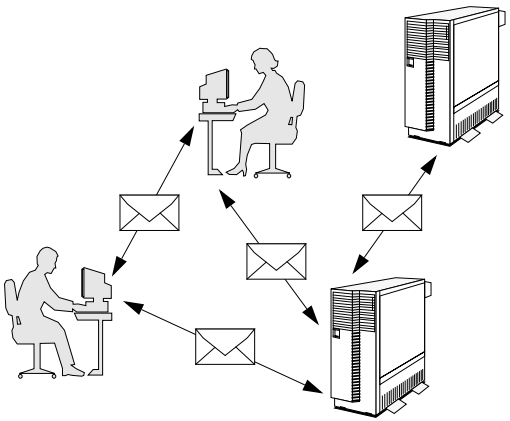


- Rechner, Personen, Prozesse, “Agenten” sind an *verschiedenen Orten*.
- Autonome Handlungsträger, die jedoch gelegentlich *kooperieren* (und dazu über Nachrichten *kommunizieren*).

# Wer bin ich? Wer sind wir?

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

Seit Herbst 1999 an der ETH Zürich



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

- Assistent(inn)en:
  - Robert Adelmann
  - Ruedi Arnold
  - Philipp Bolliger
  - Christian Flörkemeier
  - Christian Frank
  - Steve Hinske
  - Matthias Lampe
  - Marc Langheinrich
  - Benedikt Ostermaier
  - Matthias Ringwald
  - Kay Römer
  - Christof Roduner
  - Silvia Santini
  - Jonas Wolf

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

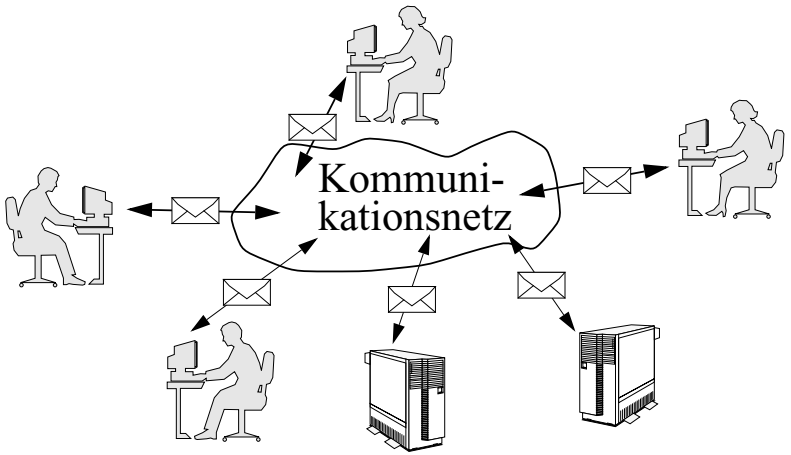
Ansprechperson für organisatorische Aspekte (z.B. Übungsbetrieb)



# "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?

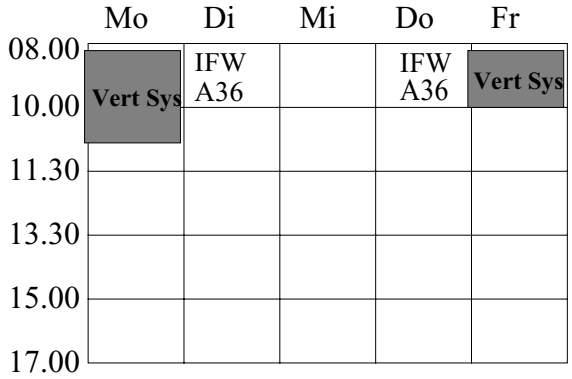
# Organisatorisches zur Vorlesung

5-stündige Veranstaltung (Vorlesung inkl. Übungen)

Sinnvolle Vorkenntnisse:

- Betriebssysteme (Prozessbegriff, Synchronisation)...
- UNIX / C / Java
- Grundkenntnisse der Informatik und Mathematik

---




Mo 08:15 - 11:00, IFW A36
Fr 08:15 - 10:00, IFW A36
} *Vorlesung inkl. Übung*

Absicht!

- Gelegentliche *Denkaufgaben* in der Vorlesung
- Praktische Übungen korrelieren gelegentlich nur schwach mit dem Inhalt der Vorlesung *(komplementieren* die Vorlesung)
- Gelegentliche *Übungsstunden* (zu den "Vorlesungsterminen)" zur Besprechung der Aufgaben und Vertiefung des Stoffes

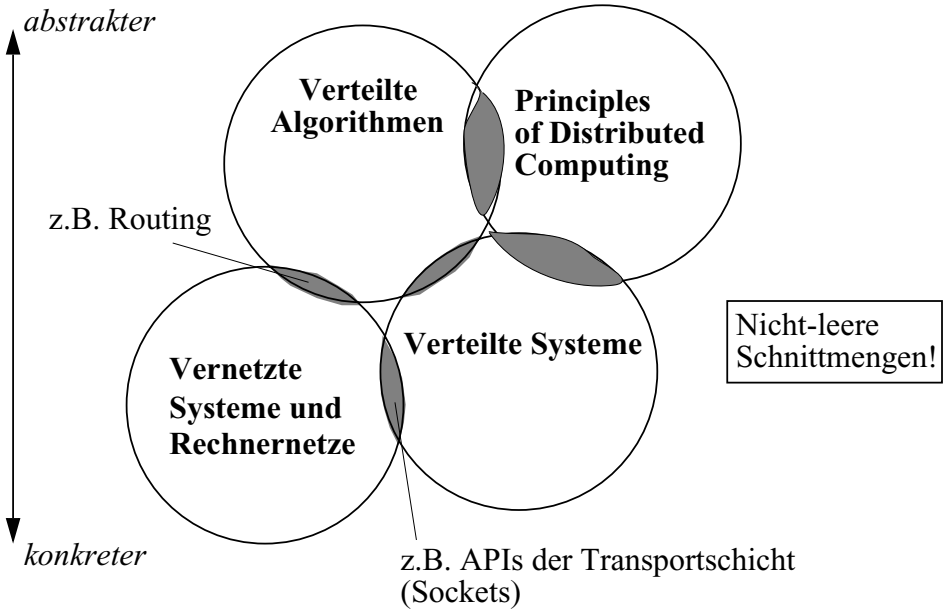
---

- *Folienkopien* jeweils einige Tage nach der Vorlesung im Web im .pdf-Format: www.vs.inf.ethz.ch/edu

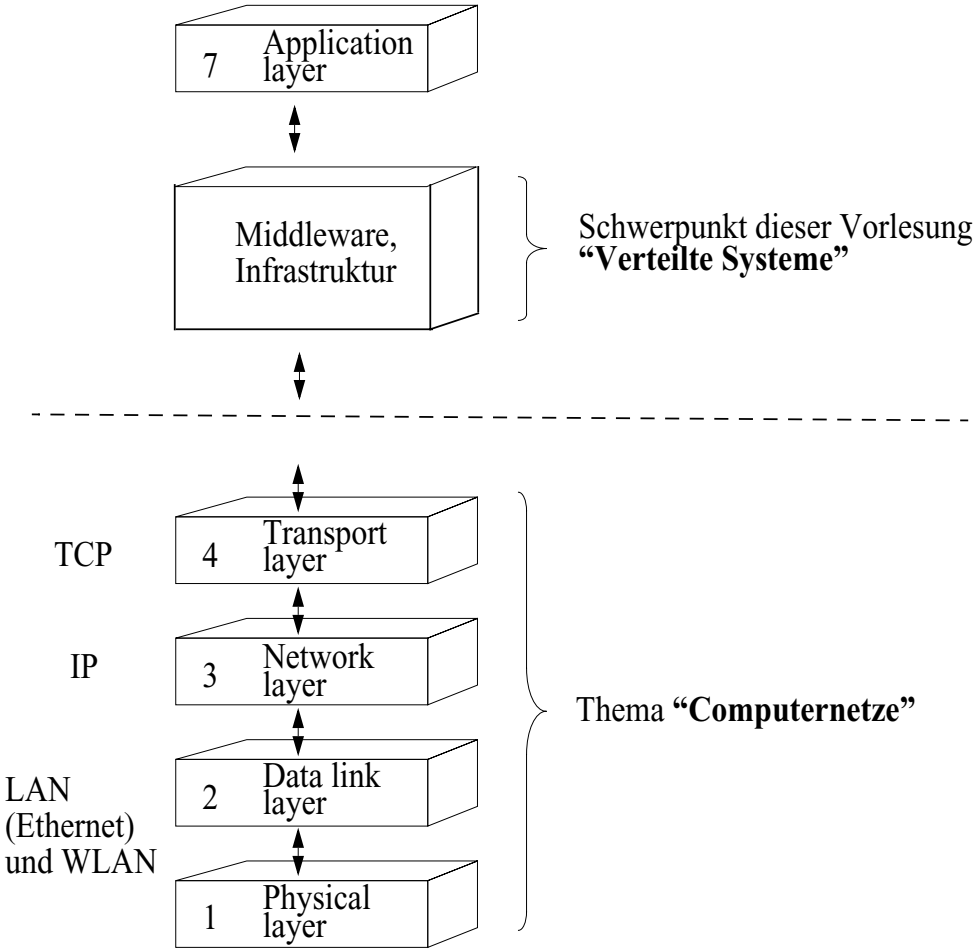


# Thematisch verwandte Veranstaltungen im Master-/Fachstudium

- *Principles of Distributed Computing*
- *Ubiquitous Computing*
- *Mobile Computing*
- Einschlägige *Seminare*
- *Semester- und Diplomarbeit / Masterarbeit*
- *Praktikum ("Labor")*
- ...



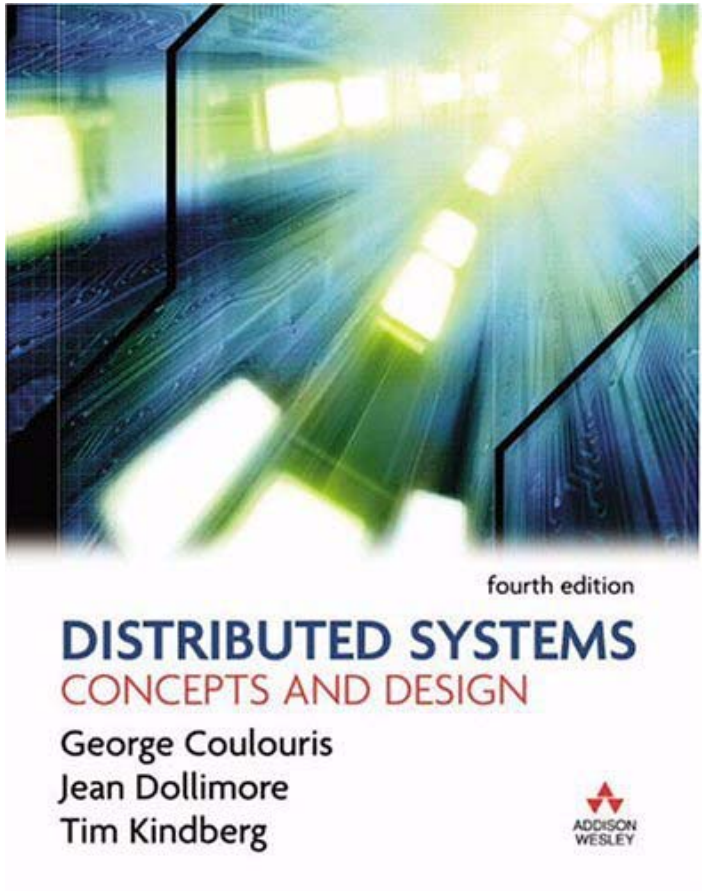
# Netze, Anwendungen, Verteilte Systeme






# Literatur

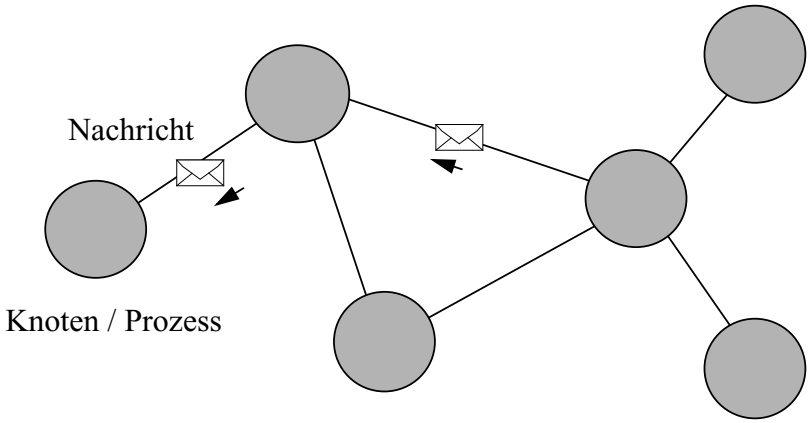
G. Coulouris, J. Dollimore, T. Kindberg: *Distributed Systems: Concepts and Design (4th ed.).* Addison-Wesley, 2005




---

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

# “Verteiltes System”

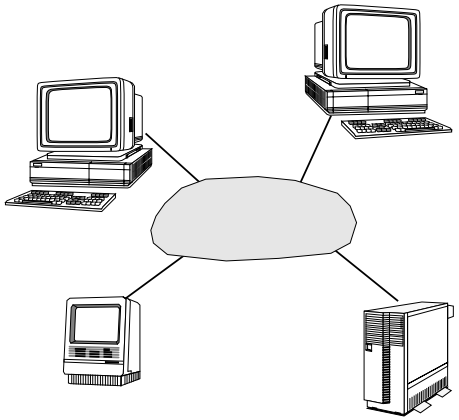



*Physisch verteiltes System:*
Mehrrechnersystem ... Rechnernetze

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

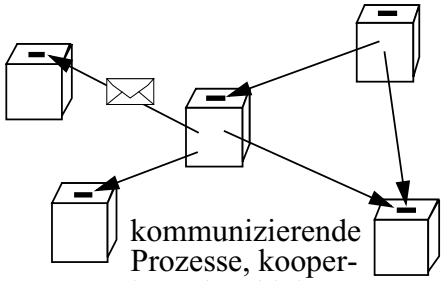


# Sichten verteilter Systeme



*Rechnernetz* mit Rechenknoten:
- Compute-Cluster
- LAN (Local Area Network)
- WAN (Wide Area Network)
- Routing, Adressierung,..

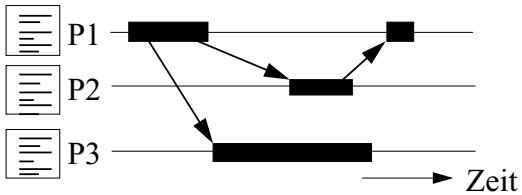
zunehmende Abstraktion



kommunizierende Prozesse, kooperierende Objekte

Objekte in *Betriebssystemen, Middleware, Programmiersprachen*

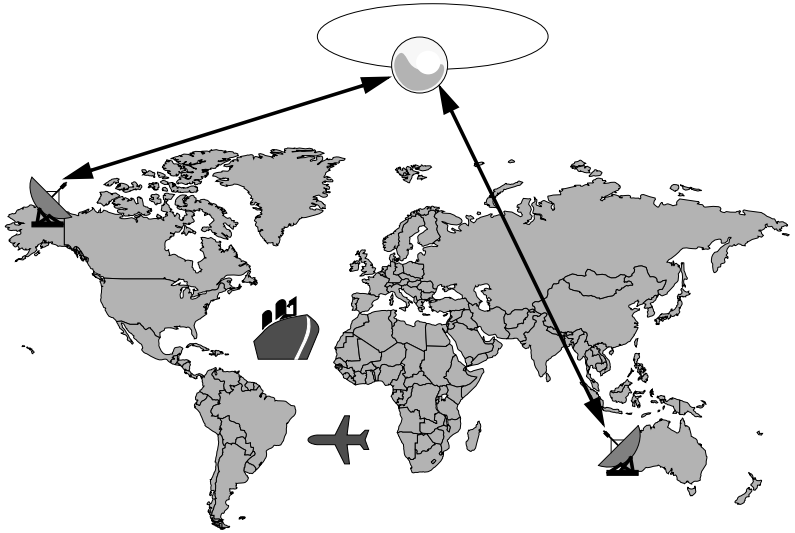
⇒ “Programmierersicht” (Client, Server...)




*Algorithmen-* und Protokoll-ebene

- Aktionen, Ereignisfolgen
- Konsistenz, Korrektheit

# Die verteilte Welt

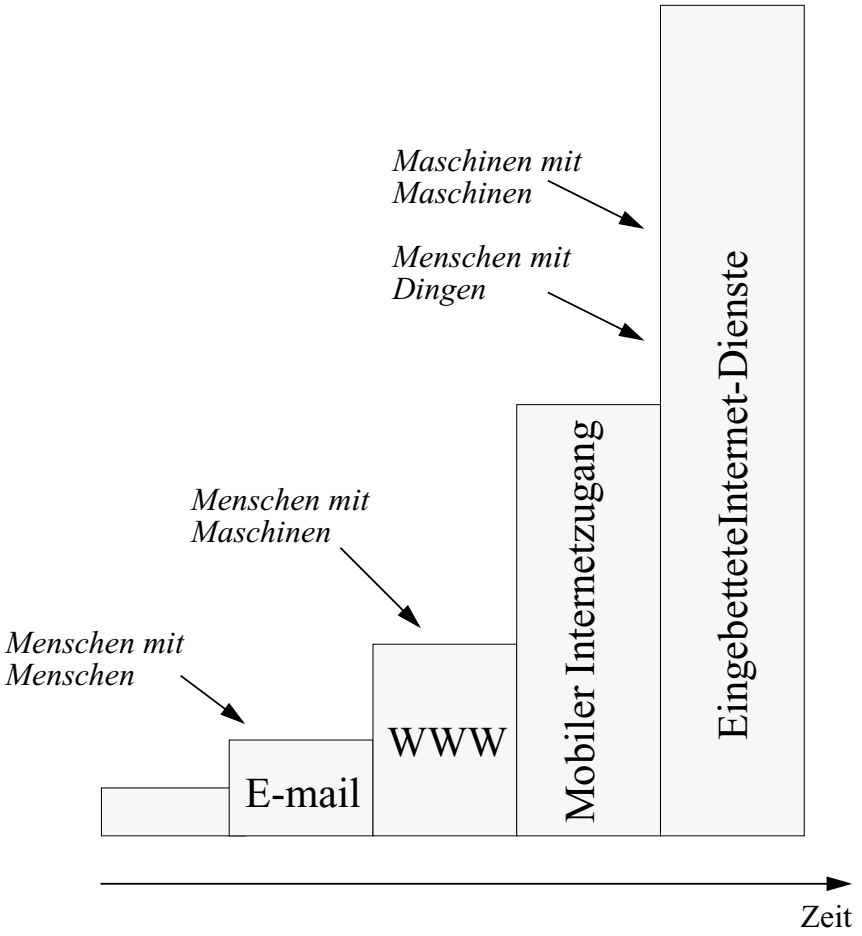


Auch die "reale Welt" ist ein verteiltes System:

- 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



# Das qualitative Internet-Wachstum



# Motivation: Software-Infrastruktur für Internet-basierte Anwendungen

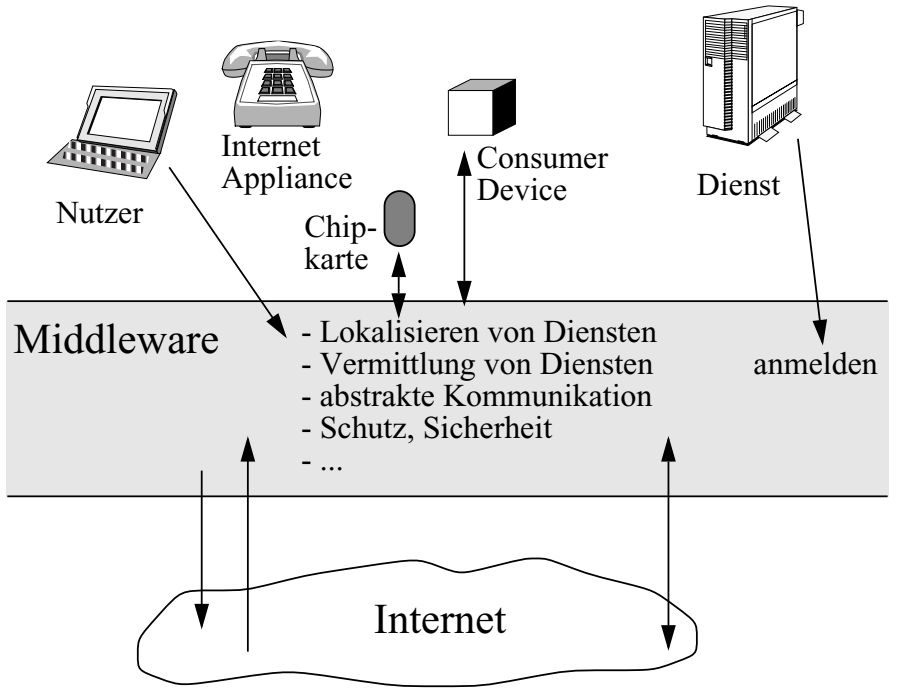
- Phänomen: das Internet verbreitet sich immer weiter
  - mehr Nutzer, Popularisierung
  - bis in die Haushalte
  - immer exotischere Endgeräte (PDA, Handy, Kühlschrank, Chipkarte)
  - bald enthalten vielleicht auch Briefmarken, Kleidungsstücke etc. kommunikationsfähige Chips
  - Sensoren
- Mobile Geräte, 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?
  - wichtig auch für Electronic Commerce-Szenarien



# Infrastruktur für verteilte Anwendungen




- Beispiel: *Jini*
  - Zweck: Interaktion mit dem Netz und mit Diensten vereinfachen
  - Lookup-Service (“Bulletin-Board”)
  - “Leasing” von Objekten (Freigabe bei Ablauf des Vertrages)
  - hot plugging von Objekten, Teildiensten etc.
  - Garbage-Collection von Objekten im Netz
  - Vermittlung von Ereignissen (events); API für events
  - Unterstützung mobiler Geräte und Dienste
  - Föderation kooperierender Java-VMs (Gruppenkonzept)
  - Mobiler Code (Java-Bytecode, Applet); z.B. Druckertreiber als “Proxy”
  - Kommunikation über entfernter Methodenaufruf oder (persistente) Tupel-Räume

# Transparenz

Transparenz = unsichtbar (“durchsichtig”) sein

Verteiltheit wird vor dem Benutzer / Programmierer verborgen, so dass das System als Ganzes gesehen wird (statt als Menge von Einzelkomponenten)

→ Umgang mit der Verteiltheit wird einfacher

→ Abstraktion von “internen” Aspekten

---

*Verschiedene Arten der Transparenz, z.B.:*

Ortstransparenz

Ort, an dem sich Daten befinden oder an dem ein Programm ausgeführt wird, ist unsichtbar

Replikationstransparenz

Unsichtbar, wieviele Replikate eines Objektes (z.B. Datei) existieren

Concurrency-Transparenz

Mehrere Benutzer / Prozesse können gemeinsame Objekte (z.B. Dateien) benutzen, ohne dass es zu Inkonsistenzen kommt

Leistungstransparenz

Kein (spürbarer) Leistungsunterschied zwischen lokaler und entfernter Bearbeitung

Ausfalltransparenz

Ausfall einzelner Komponenten ist unsichtbar → Fehlertoleranz



# Transparenz (2)

*Aufwand* zur Realisierung von Transparenz ist hoch!

*Implementierung* von Transparenz auf verschiedenen Ebenen möglich, z.B.::

- Betriebssystem (→ alle Anwendungen profitieren davon)
- Anwendungsprogramm (Nutzung der Semantik)

---

Transparenz ist *graduelle / qualitative Eigenschaft*

- oft nicht nur einfach “vorhanden” / “nicht vorhanden”

Transparenz lässt sich nicht immer (einfach) erreichen

- Beispiel: Fehlertransparenz, Leistungstransparenz
- Sollte daher nicht in jedem Fall perfekt angestrebt werden

# 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*
    - i.a. nur Ausfall von Teilfunktionalität
    - Redundanz durch Replikation



# Weitere Gründe für verteilte Systeme

- Nutzung entfernt / global verfügbarer Resourcen
  - WWW, Internet, Web-Services
  - ökonomischer Effekt, ermöglicht “Globalisierung”

- *Wirtschaftlichkeit*: Vernetzte PCs haben i.a. besseres Preis-Leistungsverhältnis als Supercomputer
  → Anwendung falls möglich “verteilen” auf mehrere kleine Rechner

- *Geschwindigkeit*: Falls Anwendung “gut” parallelisierbar, ist eine sonst unerreichbare Leistung möglich
  - ggf. (dynamische) Lastverteilung beachten
  - “Grid computing”

---

- Es gibt *inhärent geographisch verteilte* Systeme
  → z.B. Zweigstellennetz einer Bank; Steuerung einer Fabrik
  → verteilte Informationsdienste (vgl. WWW)

- *Electronic commerce*
  → kooperative Datenverarbeitung räumlich getrennter Institutionen
  → z.B. Reisebüros, Kreditkarten,...

- Mensch-Mensch-*Kommunikation*
  - Plattformen für Gruppenarbeit (z.B. Web-basiert)
  - E-Mail, Diskussionsforen, Blogs, IP-Telefonie,...

# Historische Entwicklung (“Systeme”)

Rechner-zu-Rechner-Kommunikation

- Zugriff auf entfernte Daten (DFÜ)
- Dezentrale Informationsverarbeitung zunächst ökonomisch nicht sinnvoll (zu teuer, Fachpersonal nötig)
  → Master-Slave-Beziehung (RJE, 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...)
- Heute Standard bei PC-Anwendungen im Betriebssystem:
  - Kommunikation über LAN (Resource-Sharing)
  - Software für “Gruppenarbeit” (E-Mail, gem. Dateisystem...)

Forschungsprojekte

- z.B. X-Server, Kerberos,...

Kommerzielle Projekte

- z.B. Reservierungssysteme, Banken, Kreditkarten

WWW (und Internet) als Plattform

- für electronic commerce etc.
- XML, web services, peer to peer,...



# Historische Entwicklung (“Konzepte”)

- Concurrency, Synchronisation...
  - bereits klassisches Thema bei Datenbanken und Betriebssystemen
- Programmiersprachen
  - kommunizierende Objekte
- Physische Parallelität
  - z.B. Multiprozessoren
- Parallele und verteilte Algorithmen
- Semantik
  - math. Modelle, 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!

# 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 / globale Zeit
  - Inkonsistenzen, z.B. zwischen Datei und Verzeichnis
  - Konkurrenter Zugriff, Replikate, Cache,...

Eingesetzt zur Realisierung von Leistungs- und Ausfalltoleranz

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

- Dynamik, Offenheit
  - “Interoperabilität” zu gewährleisten ist nicht einfach

- Komplexität
  - Verteilte Systeme schwierig zu entwickeln, betreiben, beherrschen
  - Abstraktion als Mittel zur Beherrschung von 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 Schwachstellen)



# Aspekte verteilter Systeme

im Vergleich zu *sequentiellen* Systemen:

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

- Programmierung *komplexer*
- Test und Verifikation *aufwendiger*
- 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

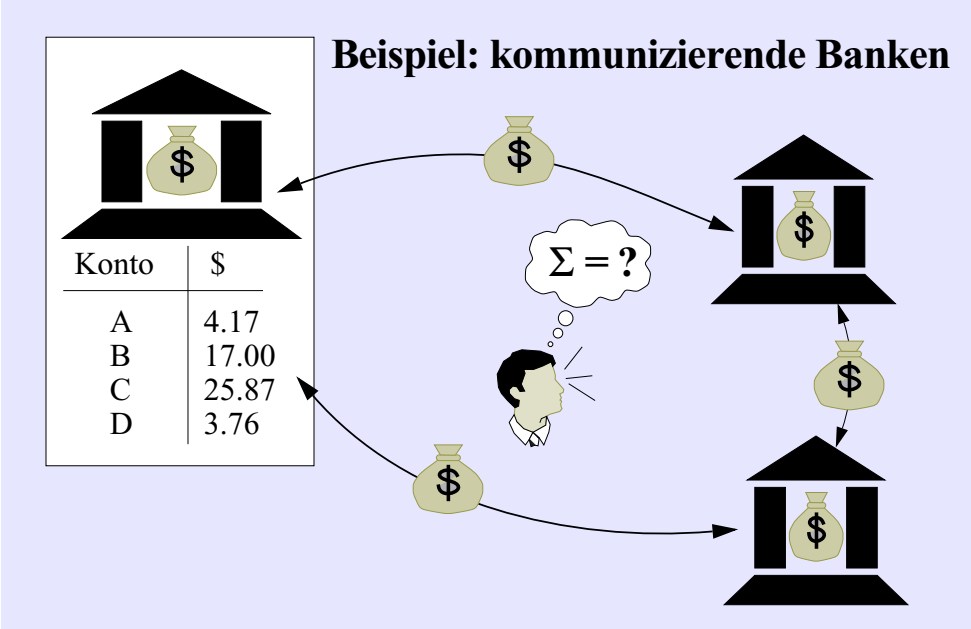
---

- 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!



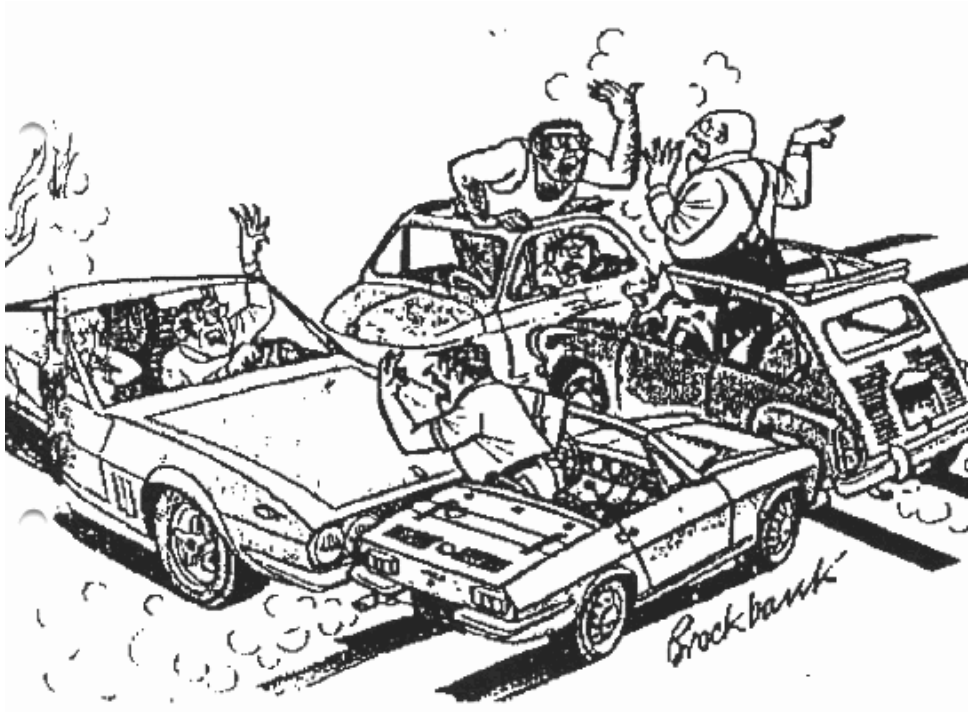
# Ein erstes Beispiel: Wieviel Geld ist in Umlauf?

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



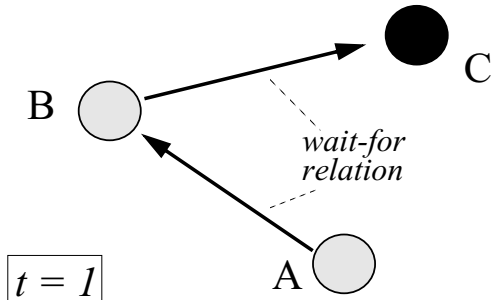

- Modellierung:
  - verteilte Geldkonten
  - ständige Transfers zwischen den Konten
- Erschwerte Bedingungen:
  - niemand hat eine globale Sicht
  - es gibt keine gemeinsame Zeit ("Stichtag")
- Anwendung: z.B. verteilte DB-Sicherungspunkte

# Ein zweites Beispiel: Das Deadlock-Problem





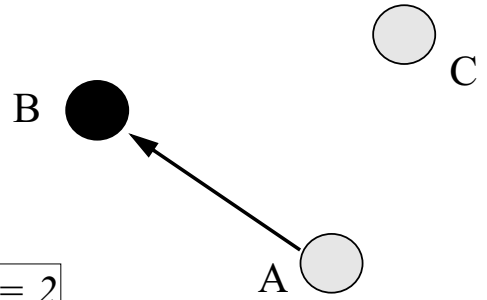

# Phantom-Deadlocks




(C benutzt ein exklusives Betriebsmittel)

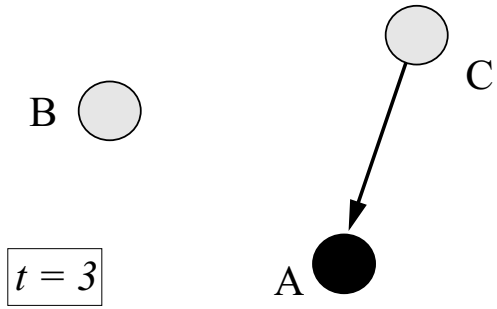
*t = 1*

*beobachte B:*
==> B wartet auf C




*t = 2*

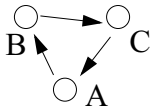
*beobachte A:*
==> A wartet auf B




*t = 3*

*beobachte C:*
==> C wartet auf A

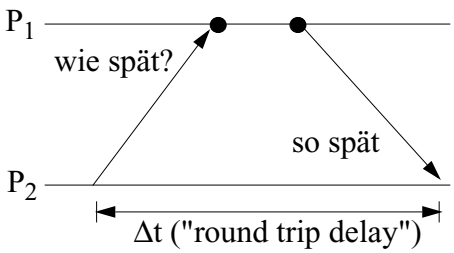
Keine exakte globale Zeit!



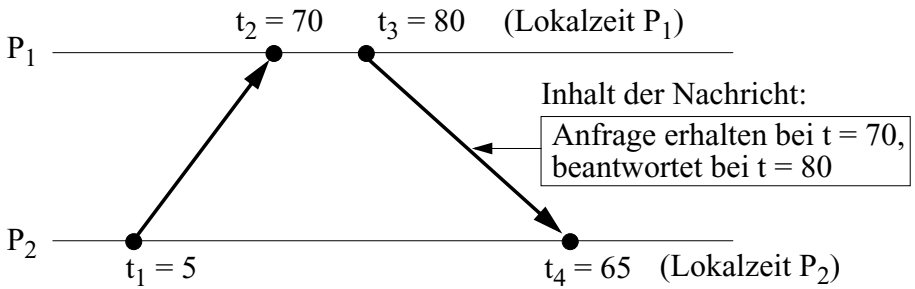

falscher Schluss!

Deadlock!

# Ein drittes Problem: Uhrensynchronisation




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



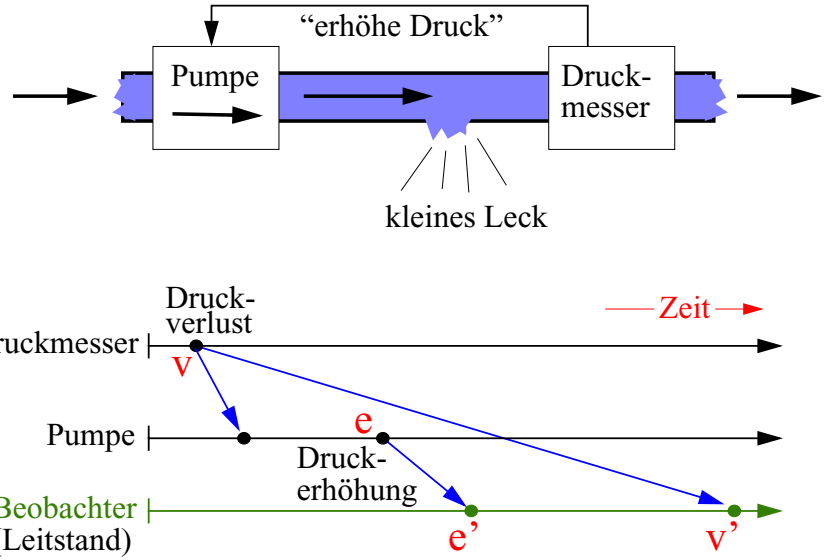

- Uhren gehen nicht unbedingt gleich schnell!
  (wenigstens "Beschleunigung ≈ 0", d.h. konstanter Drift gerechtfertigt?)

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



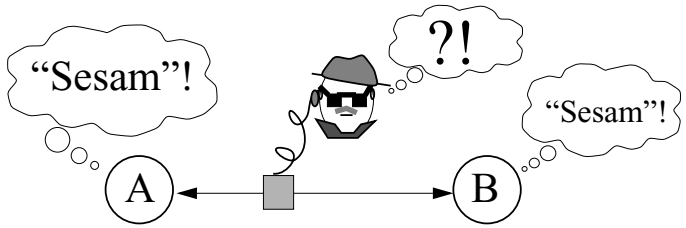
# Ein viertes Problem: Kausal (in)konsistente Beobachtungen

- 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.

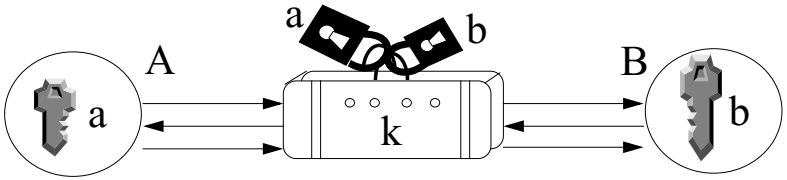
# Und noch ein Problem: Verteilte Geheimnisvereinbarung




- 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.

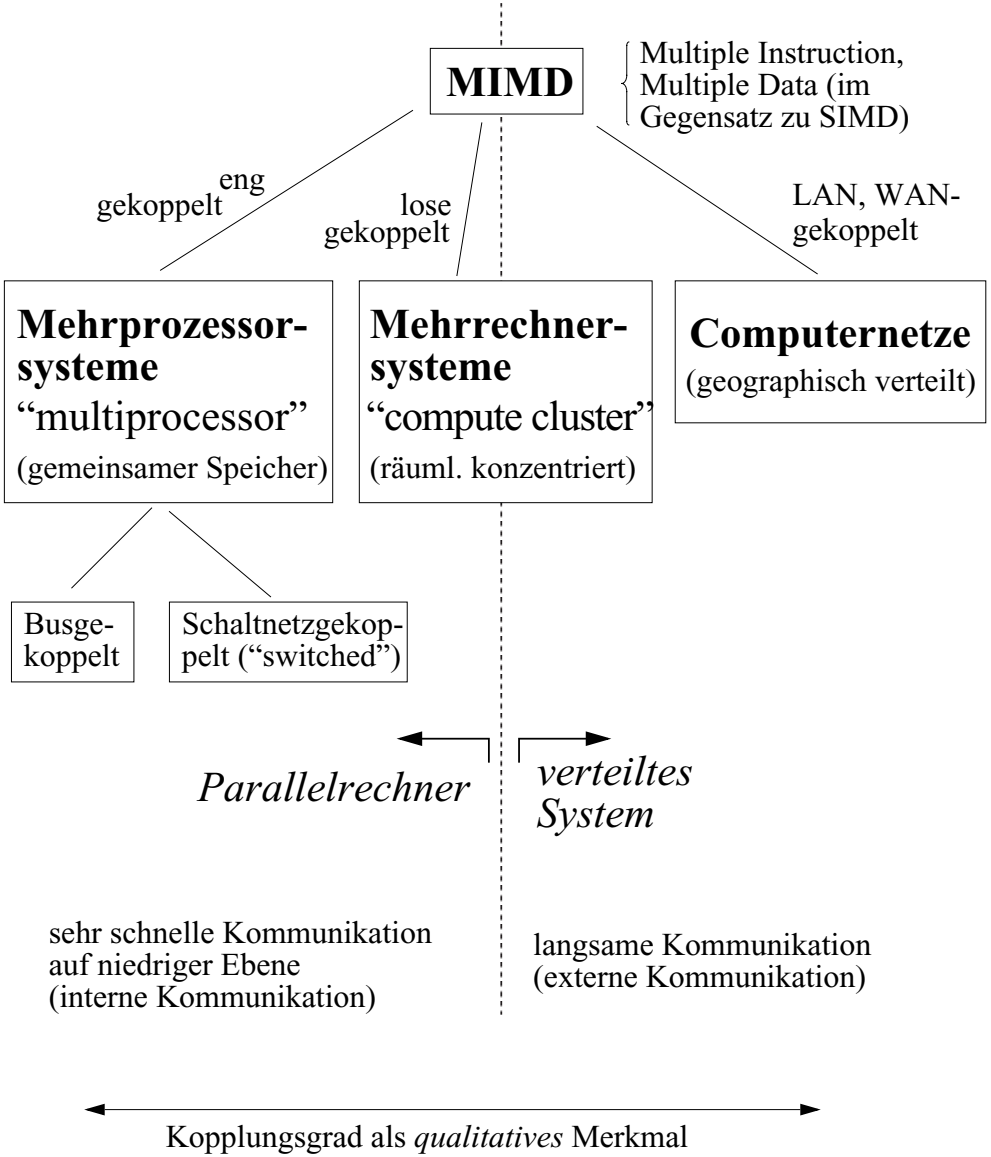
- Problem: Lässt sich das so softwaretechnisch realisieren?

  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.



# Multiprozessoren und Compute-Cluster

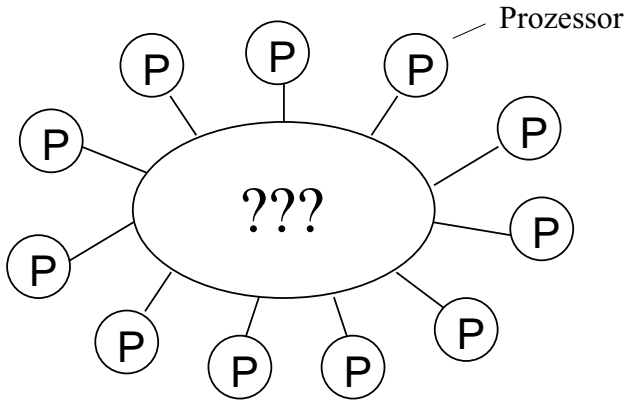
# Abgrenzung Parallelrechner





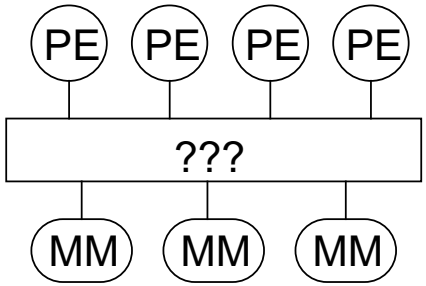

# Prozessorverbund

- Autonome Prozessoren + „Kommunikationsnetz“
- Je nach Kopplungsgrad und Grad der Autonomie ergibt sich daraus ein
  - Mehrprozessorsystem
  - Mehrrechnersystem
  - Computernetz



# Speicherkopplung

- Shared Memory
  - Kommunikation über gemeinsamen Speicher




Processing Elements

Memory Modules

- n Processing Elements teilen sich k Memory Modules

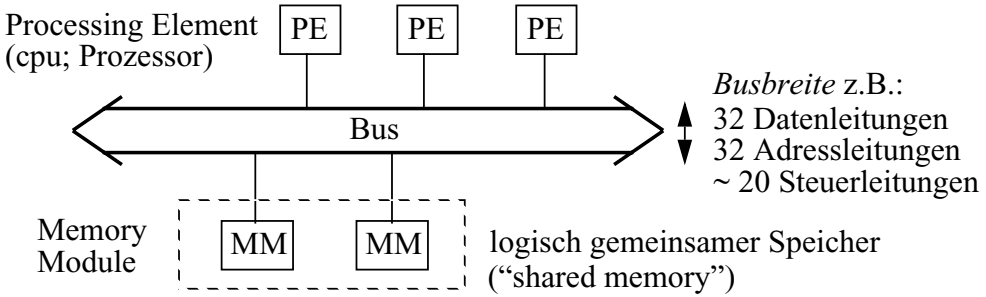
- Kopplung zwischen PE und MM, z.B.
  - Bus
  - Schaltnetz
  - Permutationsnetz

- UMA-Architektur (Uniform Memory Access) oder NUMA (Non-Uniform Memory Access)

wenn es “nahe” und “ferne” Speicher gibt: z.B. schneller Zugriff auf den “eigenen” Speicher, langsamer auf fremden



# Busgekoppelte Multiprozessoren



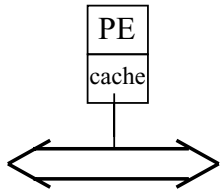

---

*Problem:*

Bus i.a. bereits bei wenigen (3 - 5) PEs überlastet

*Lösung:*

Lokale Caches
zwischen PE und Bus:



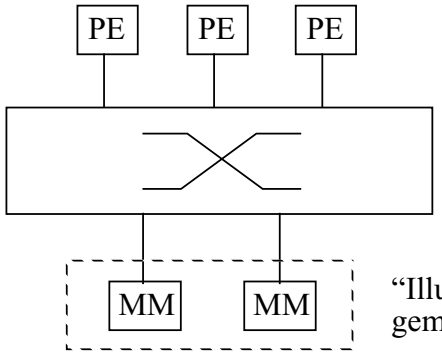

Cache gross genug wählen, um Hitraten > 90% zu erzielen (abhängig von der Hauptspeichergrösse)!

*Probleme:*

1) Kohärenzproblem der caches
2) Damit Problem nur verschoben
   (ca. 10 Mal mehr Prozessoren möglich)

Generell: Busgekoppelte Systeme schlecht skalierbar!
(Übertragungsbandbreite bleibt “konstant” bei Erweiterung um Knoten)

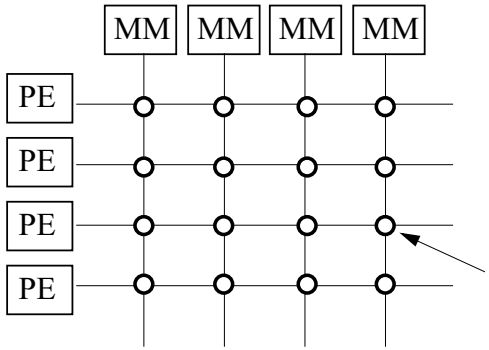
# Schaltnetzgekoppelte Multiprozessoren




Schnelles Verbindungsnetz

"Illusion" eines gemeinsamen Speichers

*Z.B. Crossbar-switch* (Kreuzschienenverteiler):




elektronische Schalter ("crosspoint switch")

- Mehrere PEs können gleichzeitig auf verschiedene Speichermodule zugreifen
- Schlecht skalierbar (quadratisch viele Schalter)
  (Vermeidung von hot spots durch interleaving, Randomisierung...)

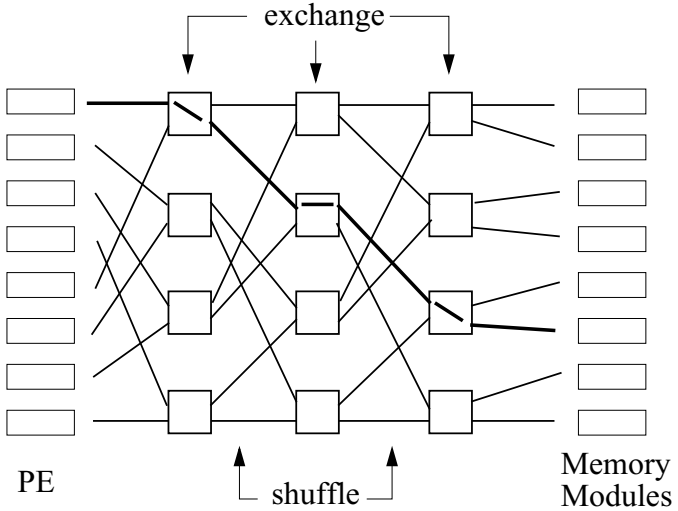


# Permutationsnetze

Mehrere Stufen von Schaltelementen ermöglichen die Verbindung jedes Einganges zu jedem Ausgang.

Schaltelement ("interchange box") kann zwei Zustände annehmen (durch ein Bit ansteuerbar):

straight through exchange




Beispiel:

*Shuffle-Exchange-*Netz

(Omega-Netz)

Hier: log n (identische!) Stufen mit je n/2 Schaltern.

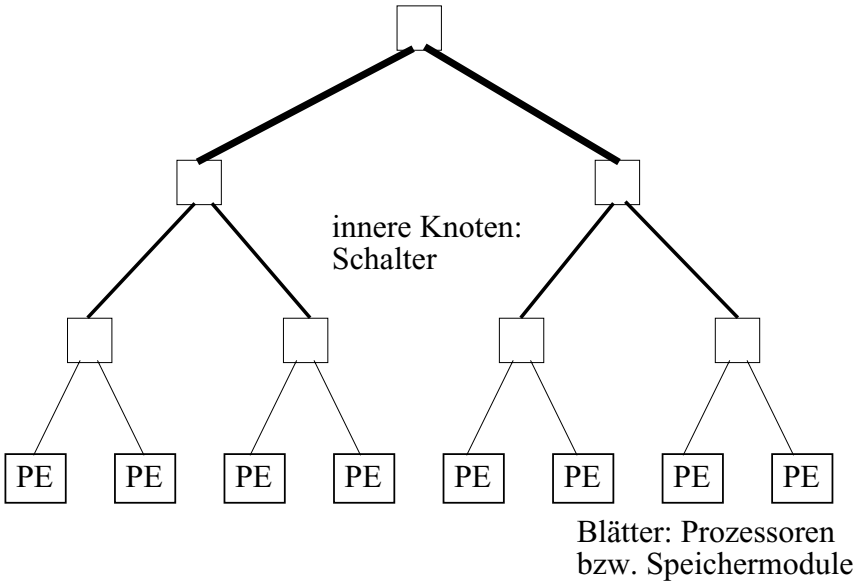
Es gibt weitere ähnliche dynamisch schaltbare Netze.

Designkriterien:

z.B. Butterfly-Netze

- wenig Stufen ("delay")
- Parallele Zugriffe; Vermeidung von Blockaden

# Fat-Tree-Netze




Verbindungsleitungen höherer Bandbreite bzw. mehrere parallele Leitungen auf Niveaus, die näher an der Wurzel liegen.

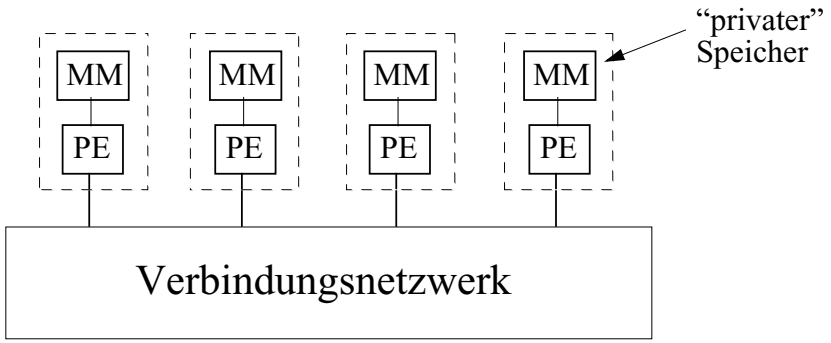


# Multiprozessoren - Fazit

- Gemeinsamer Speicher, über den die Prozessoren Information austauschen (d.h. kommunizieren) können
  - Prozessoren müssen mit dem Speicher (bzw. den einzelnen Speichermodulen) gekoppelt werden
- Speicherkopplung begrenzt Skalierbarkeit und räumliche Ausdehnung
  - Untergliederung des Speichers in mehrere Module (Parallelität)
  - leistungsfähiges Kommunikationsnetz
- Lokale PE-Caches sinnvoll
  - Problem der Cache-Kohärenz
- Bewertungskriterien für Verbindungsnetze
  - Realisierungsaufwand (Fläche, Kosten)
  - Skalierbarkeit (mit wachsender Anzahl PEs und MMs)
  - innere Blockadefreiheit (parallele Kommunikationsvorgänge)
  - Anzahl der Stufen (Verzögerung)
  - Eingangsgrad, Ausgangsgrad der Bauelemente

# Mehrrechnersysteme ("Compute Cluster")

Vernetzung vollständiger Einzelrechner:

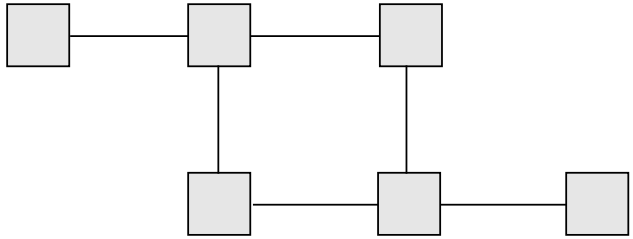



Zugriff auf andere Rechner (bzw. deren private Speicher) nur indirekt über *Nachrichten*

- kein globaler Speicher
- NORMA-Architektur (NO Remote Memory Access)



# Verbindungstopologien für Mehrrechnersysteme



Zusammenhängender Graph mit

Knoten = Rechner

Kante = dedizierte Kommunikationsleitung

Ausdehnung: i.a. nur wenige Meter

---

*Bewertungskriterien:*

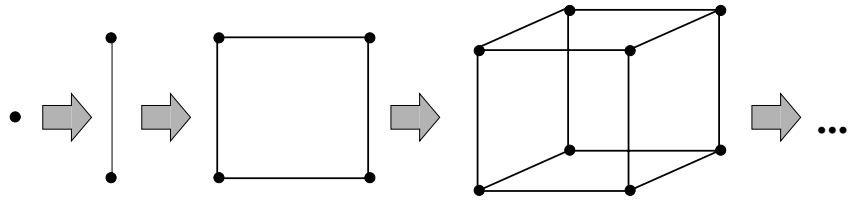
- Gesamtzahl der Verbindungen (bei n Knoten)
- maximale Entfernung zweier Knoten
- durchschnittliche Entfernung
- Anzahl der Nachbarn eines Knotens ("fan out")
- Symmetrie, Homogenität, Skalierbarkeit...
- Routingkomplexität
- Zahl der alternativ bzw. parallel verfügbaren Wege

*Technologische Faktoren:*

- Geschwindigkeit, Durchsatz, Verzögerung, eigene Kommunikationsprozessoren...

# Hypercube

- Hypercube = "Würfel der Dimension d"



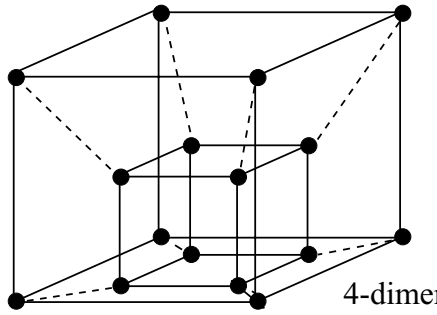
Draufsicht von der Seite liefert jeweils niedrigere Dimension

Entsprechend: Herausdrehen des Objektes aus der Blickebene zeigt, dass es sich "eigentlich" um ein Objekt der Dimension n+1 handelt!

- Rekursives Konstruktionsprinzip

  - Hypercube der Dimension 0: Einzelrechner
  - Hypercube der Dimension d+1:

    *„Nimm zwei Würfel der Dimension d und verbinde korrespondierende Ecken"*



4-dimensionaler Würfel

Man vgl. auch das Buch von T. F. Banchoff: Beyond the Third Dimension (Scientific American Library, 1990)



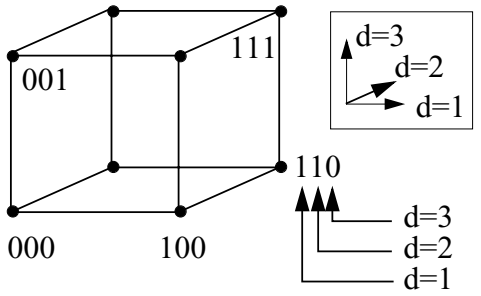
# Hypercube der Dimension d

- $n = 2^d$ Knoten

- Anzahl der Nachbarn eines Knotens = d
  (Anzahl der "ports" in der Hardware)

- Gesamtzahl der Kanten (= Verbindungen): $d\ 2^d/2 = d\ 2^{d-1}$
  (Ordnung O($n$ log $n$))

- Einfaches Routing:
  - Knoten systematisch (entspr. rekursivem Aufbau) numerieren
  - 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 Übung!)

-Vorteile Hypercube:

- kurze Weglängen (max. log n) ← Denkübung: mittlere Weglänge?
- einfaches Routing
- viele Wegalternativen (Fehlertoleranz, Parallelität!)

-Nachteile: ← wieviele verschiedene Wege der Länge k gibt es insgesamt?

- Anzahl der Nachbarknoten eines Knotens wächst mit der Dimension d
- insgesamt relativ viele Verbindungen: O(n log n)
  (eigentlich genügen n-1)