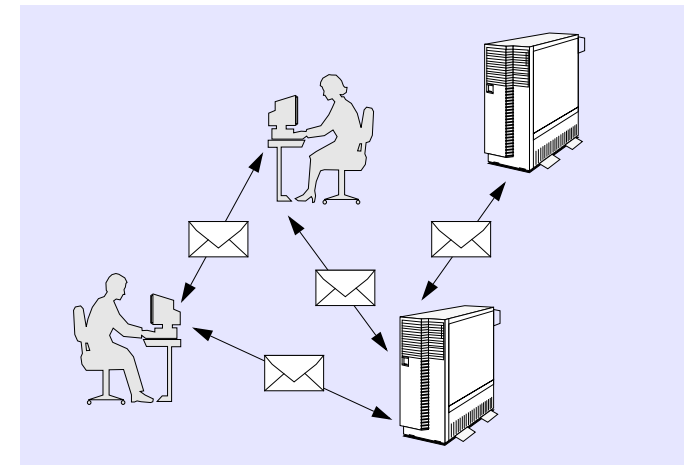


Verteilte Systeme

Friedemann Mattern

Institut für Pervasive Computing
Departement Informatik
ETH Zürich



Folienkopien zur Vorlesung

Verteilte Systeme

Friedemann Mattern

Departement Informatik, ETH Zürich

Herbstsemester 2007 (5-stündig inkl. Übungen)

© F. Mattern, 2007

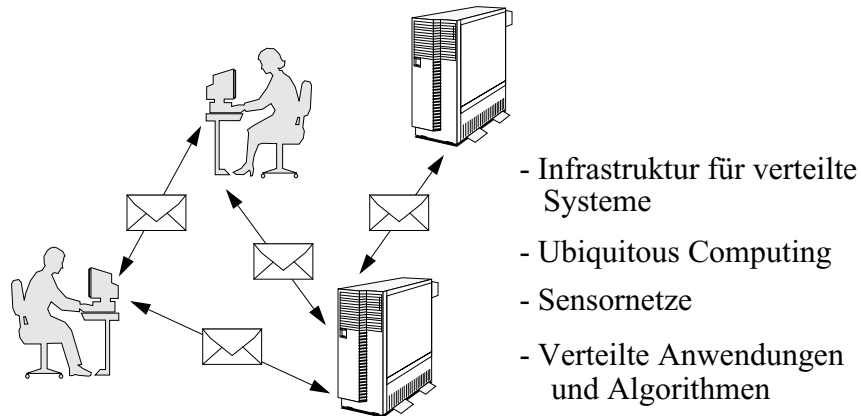
Achtung: Prüfungsrelevant ist der Inhalt der Vorlesung, nicht alleine der Text dieser Foliensammlung!

- 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



- Assistent(inn)en:

- Robert Adelman
- Ruedi Arnold
- Philipp Bolliger
- Steve Hinske
- Moritz Köhler
- Matthias Lampe
- Marc Langheinrich
- Benedikt Ostermaier
- Matthias Ringwald
- Kay Römer
- Christof Roduner
- Silvia Santini
- Vlad Trifa

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

Ansprechperson für organisatorische Aspekte (z.B. Übungsbetrieb)

Organisatorisches zur Vorlesung

5-stündige Veranstaltung (Vorlesung inkl. Übungen)

Sinnvolle Vorkenntnisse:

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

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

- 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

Absicht!

- *Folienkopien* jeweils einige Tage nach der Vorlesung

- im .pdf-Format bei www.vs.inf.ethz.ch/edu

- Vorlesung im Dezember: *Prof. G. Alonso*

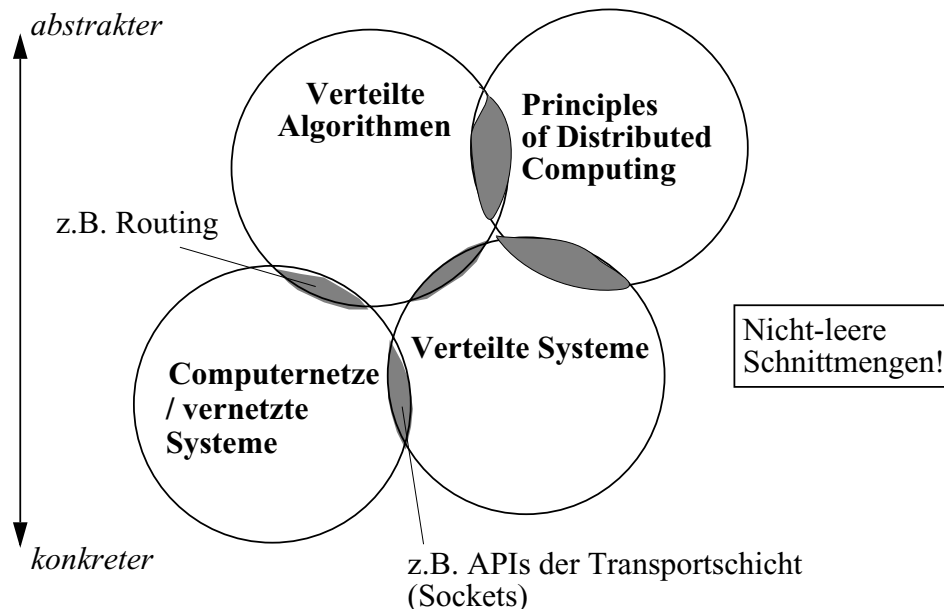
- Prüfung schriftlich

- 20% der bewerteten Übungen gehen in die Prüfungsnote ein

Thematisch verwandte Veranstaltungen im Masterstudium

- Principles of Distributed Computing
- Ubiquitous Computing
- Ad-hoc- und Sensor-Netzwerke / Drahtlose Sensornetze
- Peer-to-Peer and Self-Organizing Networks
- Enterprise Application Integration - Middleware
- Web Services and Service Oriented Architectures
- Verteilte Algorithmen
- ...

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



Literatur

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



fourth edition

DISTRIBUTED SYSTEMS CONCEPTS AND DESIGN

George Coulouris
Jean Dollimore
Tim Kindberg



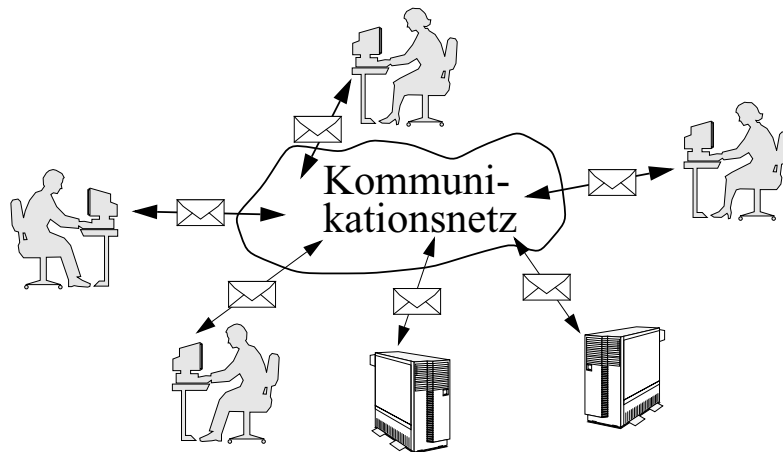
A. Tanenbaum, M. van Steen: *Distributed Systems: Principles and Paradigm (2nd ed.)*. Prentice-Hall, 2007

Alexander Schill, Thomas Springer: *Verteilte Systeme*. Springer, 2007

“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.

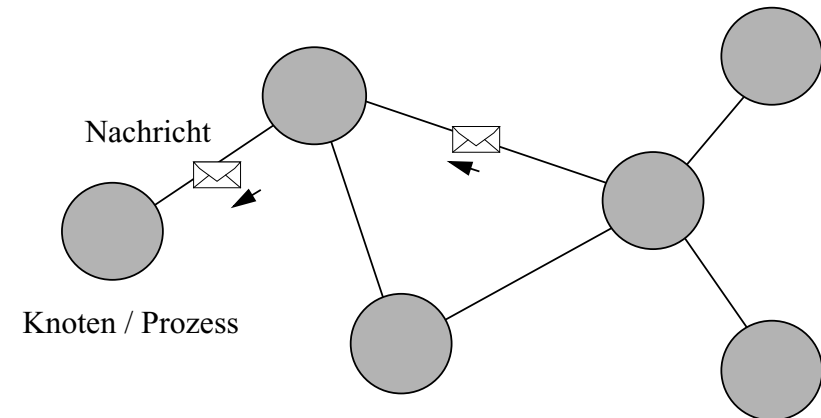
-- H. Bal



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

“Verteiltes System”



Physisch verteiltes System:

Mehrrechnersystem ... Rechnernetze

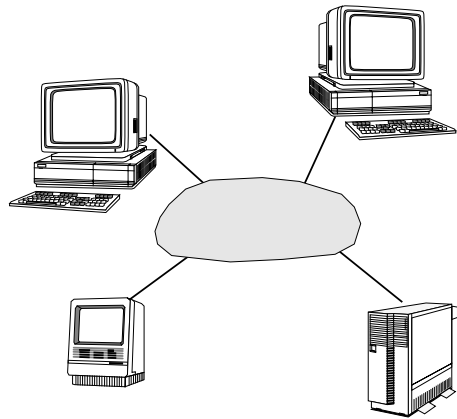
Logisch verteiltes System: Prozesse (Objekte, Agenten)

- Verteilung des Zustandes (keine globale Sicht)
- Keine gemeinsame Zeit (globale, genaue "Uhr")

- welche Problemaspekte stecken hinter Lamports Charakterisierung?

Sichten verteilter Systeme

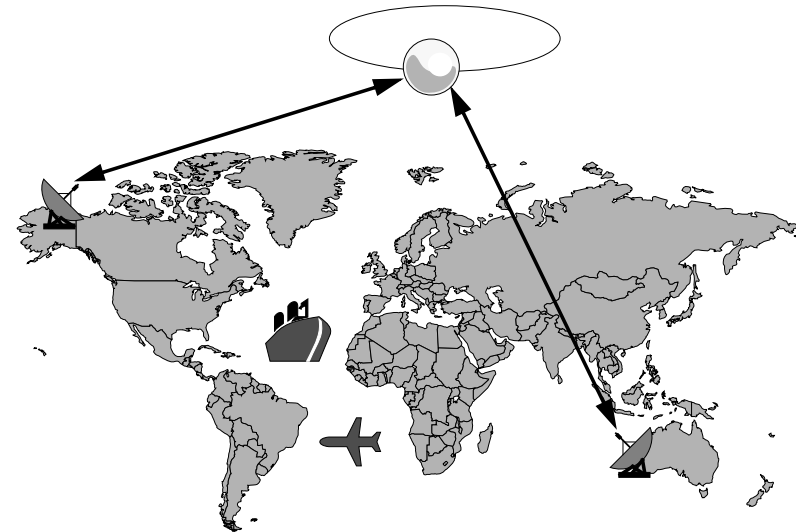
Die verteilte Welt



Rechnernetz mit Rechenknoten:

- Compute-Cluster
- Local Area Network
- Internet

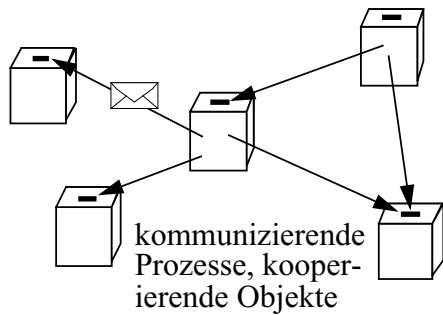
⇒ Routing, Adressierung,...



zunehmende Abstraktion

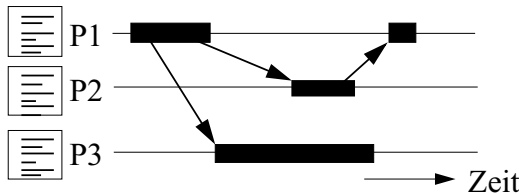
Auch die "reale Welt" ist ein verteiltes System:

- 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



Objekte in
Betriebssystemen,
Middleware,
Programmiersprachen

⇒ "Programmierersicht"
(Client, Server...)

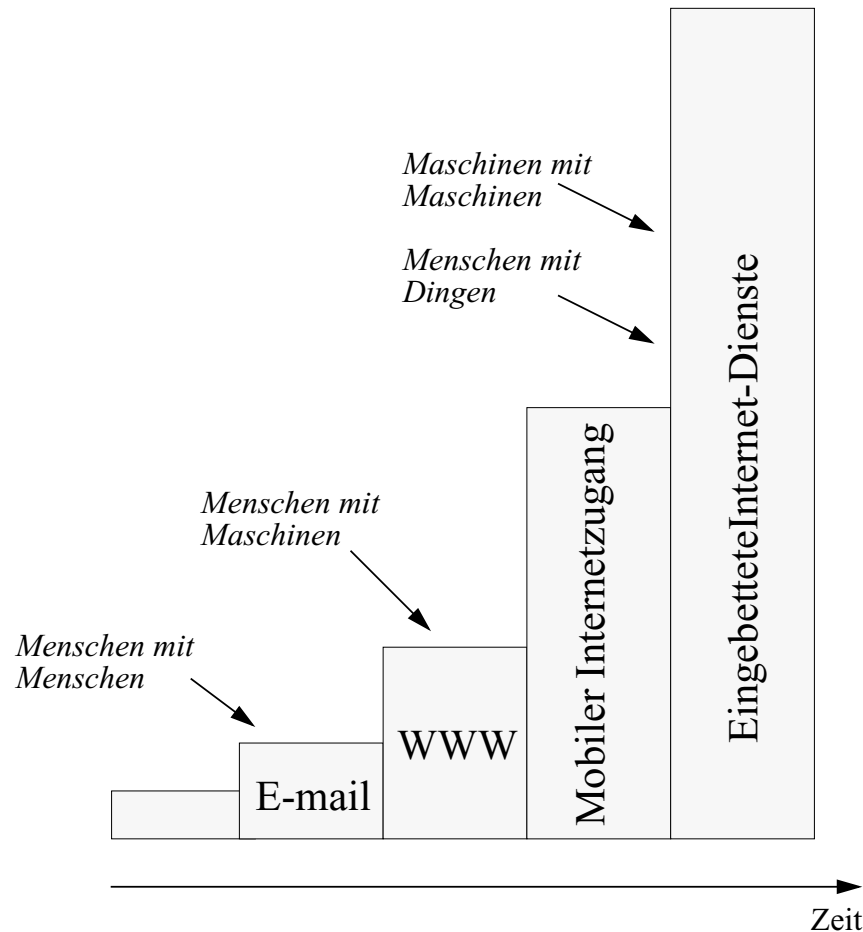


Algorithmen-
und Protokoll-
ebene

- Aktionen, Ereignisfolgen
- Konsistenz, Korrektheit

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, bis an die Person (Handy)
 - immer "exotischere" Endgeräte (TV, 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...

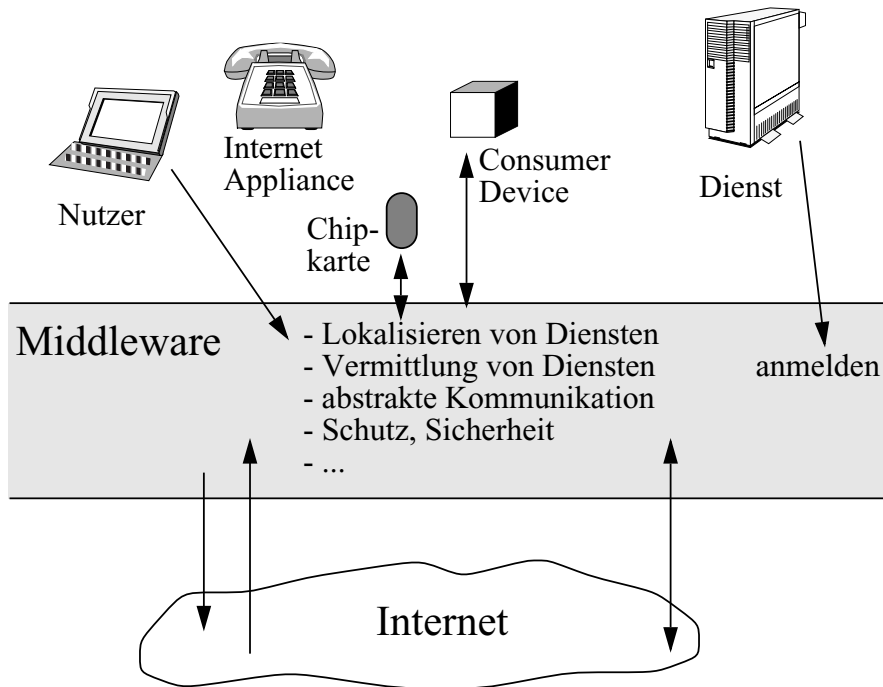
- Markt erfordert sehr schnelle Reaktion

- 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 ("Repository")
- "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
- 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 Ressourcen
 - 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...
 - war bereits klassisches Thema bei Datenbanken und Betriebssystemen
- Programmiersprachen
 - kommunizierende Objekte
- Physische Parallelität
 - z.B. Multiprozessoren
- Parallele und verteilte Algorithmen
- Semantik
 - mathematische 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,...

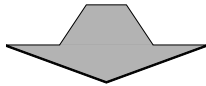
Eingesetzt zur Realisierung von Leistungs- und Ausfalltransparenz

- 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, Authentizität, 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 → vieles gleichzeitig
- Nebenläufigkeit → morgen anders als heute...
- Nichtdeterminismus → niemand weiss alles
- Zustandsverteilung



- 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

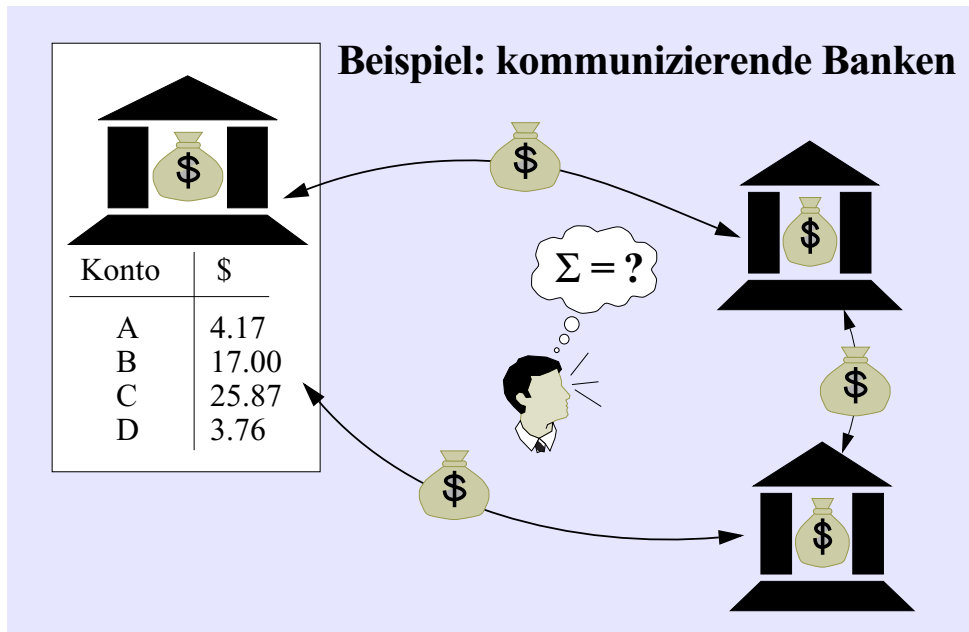
Einige *konzeptionelle* Probleme und Phänomene verteilter Systeme

- 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:

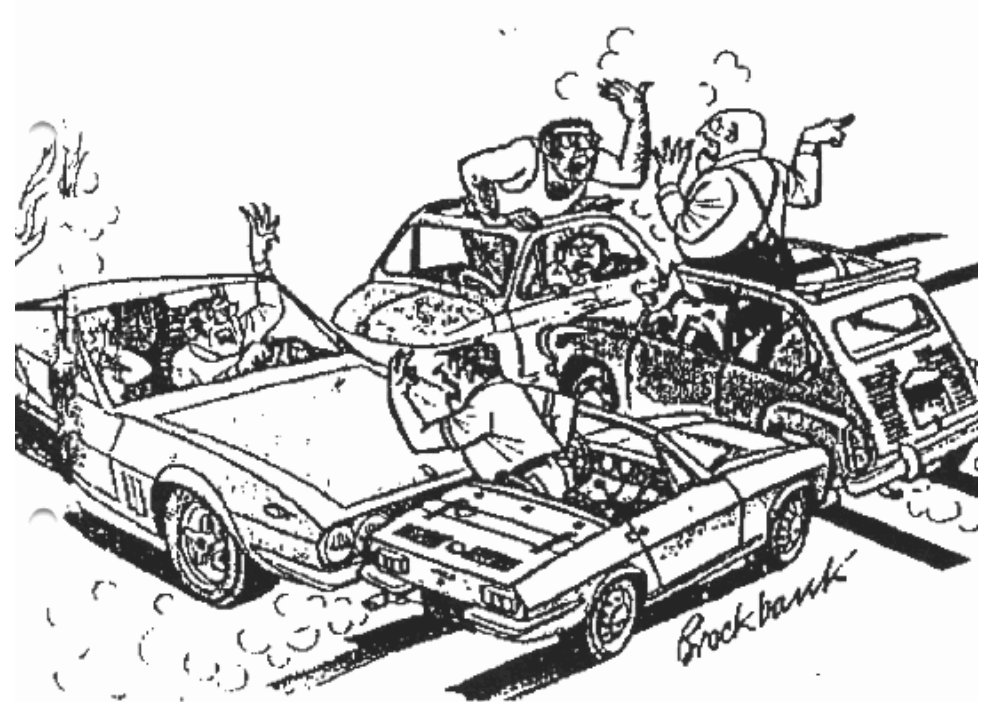
- verteilte Geldkonten
- ständige Transfers zwischen den Konten

- Erschwerte Bedingungen:

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

- Anwendung: z.B. verteilte DB-Sicherungspunkte

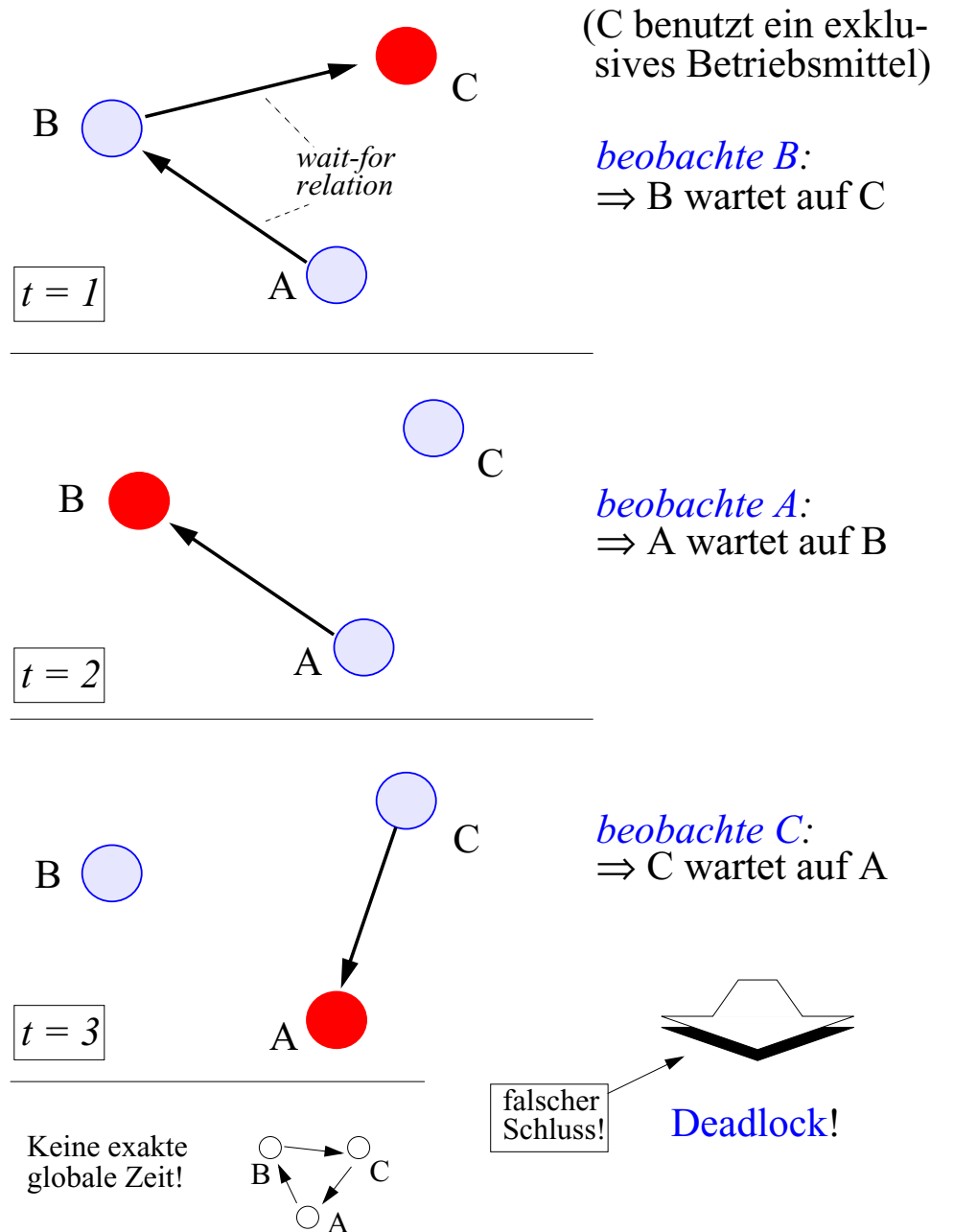
Ein zweites Beispiel: Das Deadlock-Problem



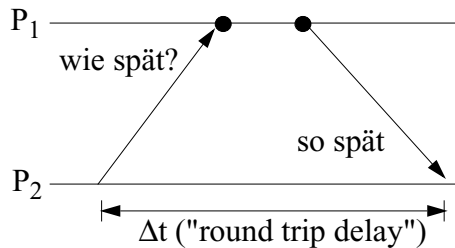
Das Deadlock-Problem



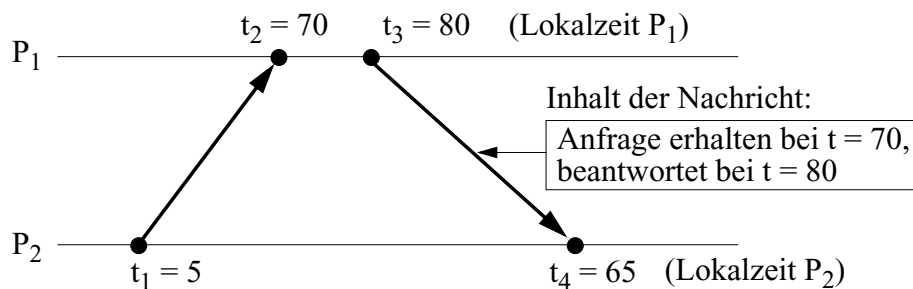
Phantom-Deadlocks



Ein drittes Problem: Uhrensynchronisation



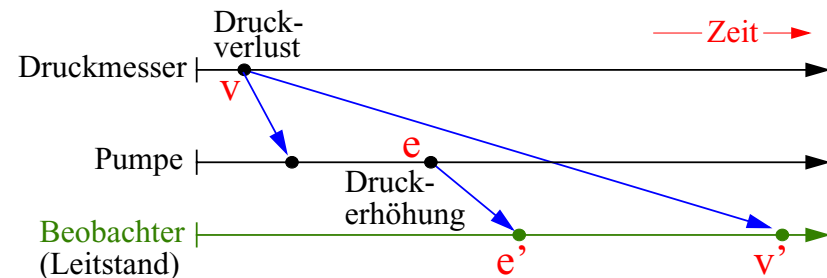
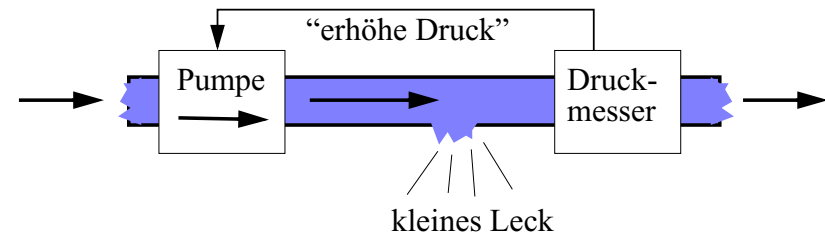
- Lastabhängige Laufzeiten von Nachrichten
- Unsymmetrische Laufzeiten
- Wie erfährt man die Laufzeit?



- 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?

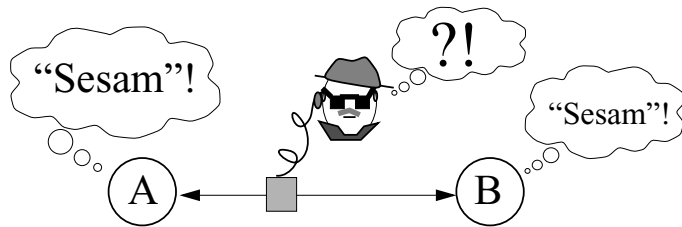
Ein viertes Problem: (nicht) kausaltreue Beobachtungen

- 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.

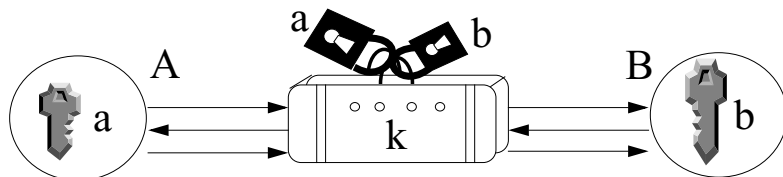
Und noch ein Problem: Verteilte Geheimnisvereinbarung



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

Multiprozessoren und Compute-Cluster

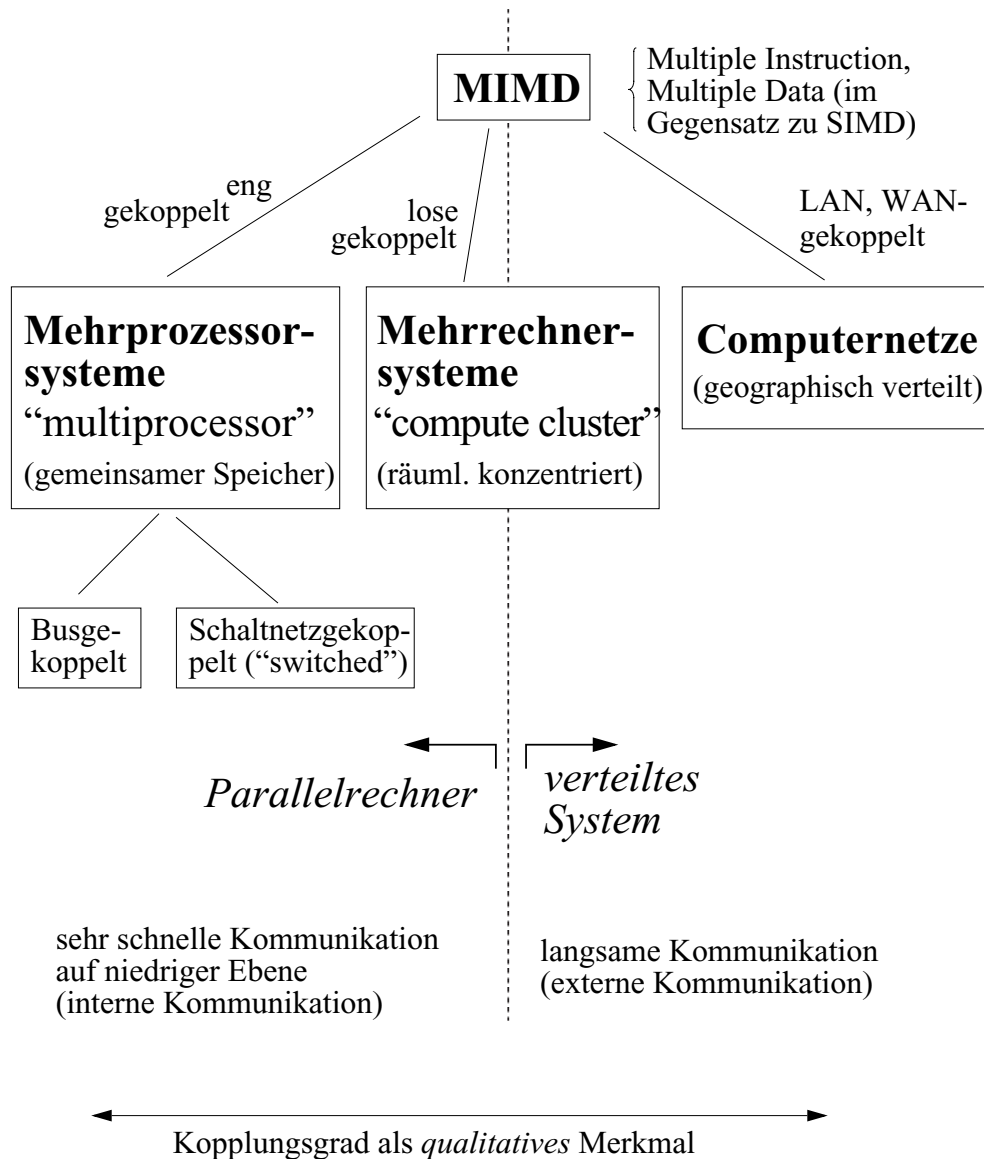
- Idee: Vorhängeschlösser um eine sichere Truhe:



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?

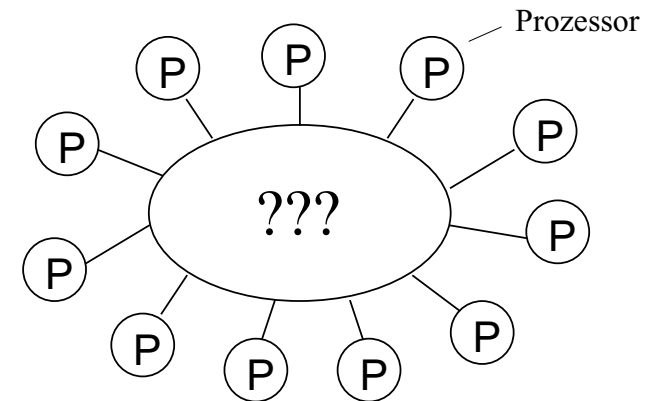
Wie wäre es damit?: k sei eine Zahl. “Verschliessen” und “aufschliessen” eines Schlosses entspricht dem Hinzuaddieren oder Subtrahieren einer beliebig ausgedachten (geheimgehaltenen) Zahl a bzw. b .

Abgrenzung Parallelrechner



Prozessorverbund

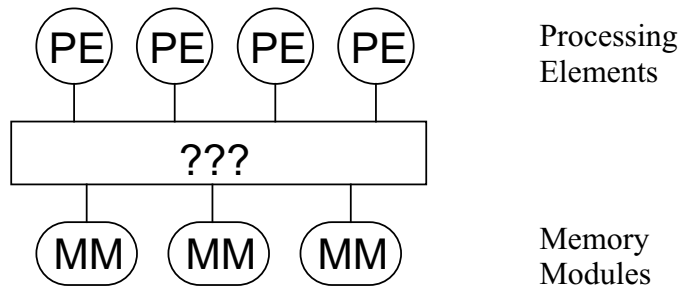
- Autonome Prozessoren + „Kommunikationsnetz“
- Je nach Kopplungsgrad und Grad der Autonomie ergibt sich daraus ein
 - Mehrprozessorsystem
 - Compute Cluster
 - Computernetz



Speicherkopplung

- Shared Memory

- Kommunikation über gemeinsamen Speicher



- n Processing Elements teilen sich k Memory Modules

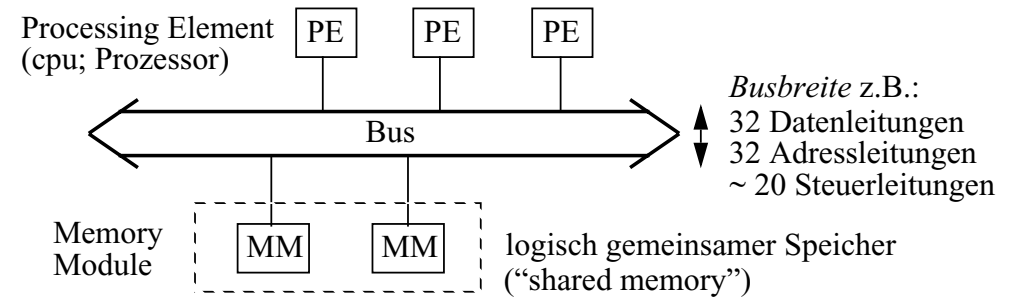
- Kopplung zwischen PE und MM, z.B.

- 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

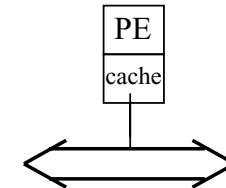


Problem:

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

Lösung:

Lokale Caches
zwischen PE und Bus:



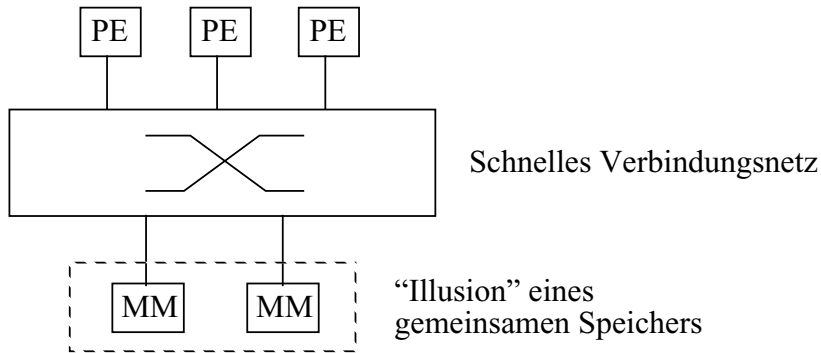
Cache gross genug wählen, um Hitraten > 90% zu erzielen (abhängig von der Hauptspeichergrösse)!

Probleme:

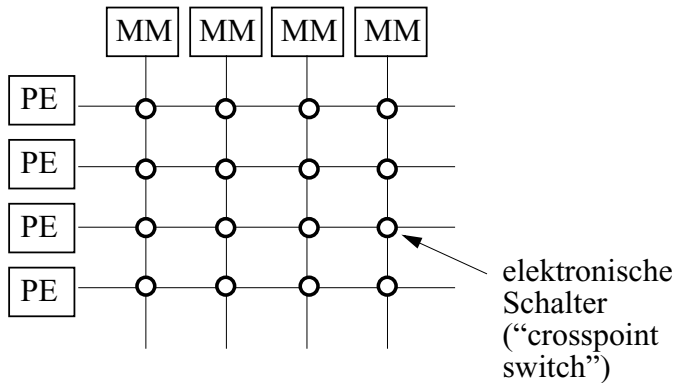
- 1) Kohärenzproblem der caches
- 2) Damit Problem nur verschoben (ca. 10 Mal mehr Prozessoren möglich)

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

Schaltnetzgekoppelte Multiprozessoren



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

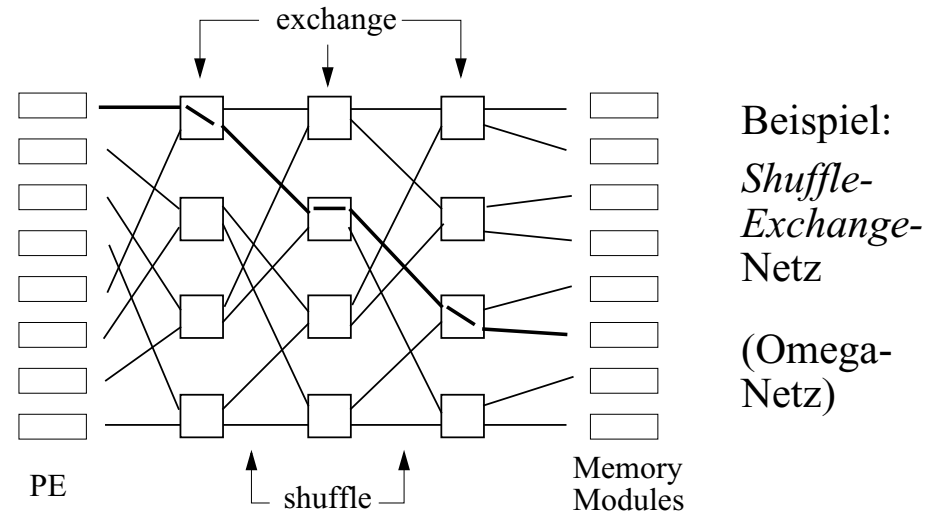
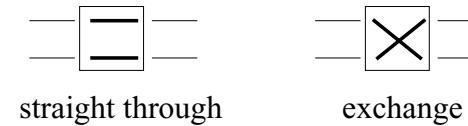


- Mehrere PEs können teilweise gleichzeitig auf verschiedene Speichermodule zugreifen
- Schlecht skalierbar (quadratisch viele Schalter)
(Vermeidung von hot spots durch interleaving, Randomisierung...)

Permutationsnetze

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

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



Beispiel:
Shuffle-Exchange-Netz
(Omega-Netz)

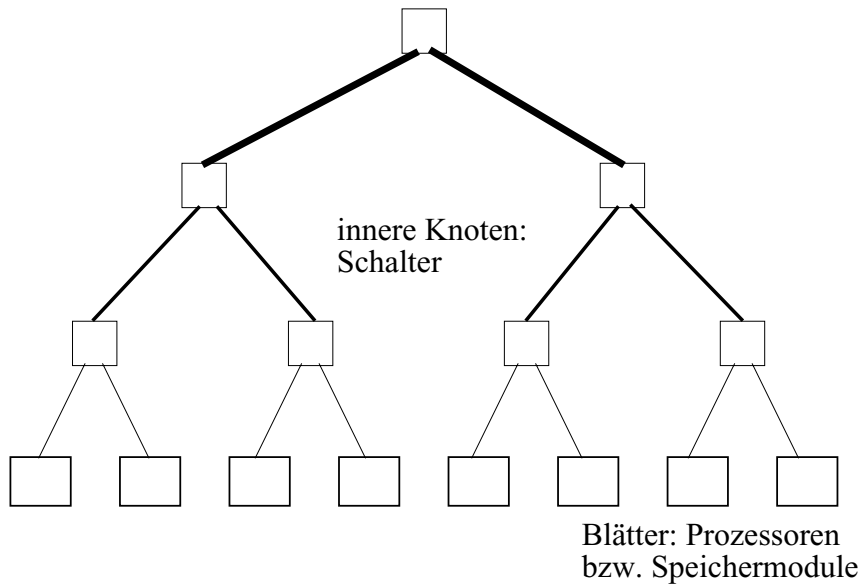
Hier: $\log n$ (identische!) Stufen mit je $n/2$ Schaltern.

Es gibt weitere ähnliche dynamisch schaltbare Netze.
Designkriterien:

z.B. Butterfly-Netze

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

Fat-Tree-Netze



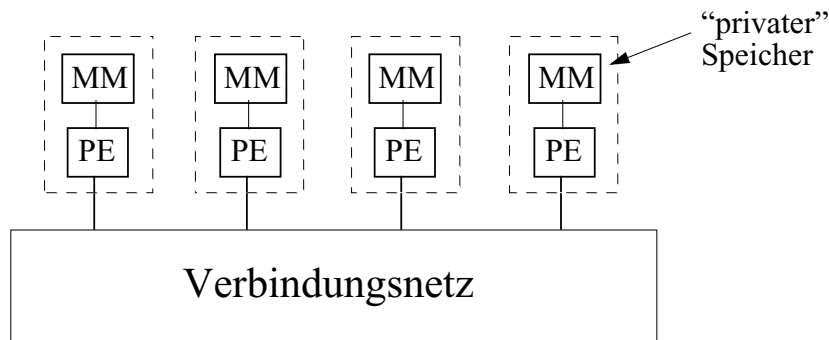
Verbindungsleitungen höherer Bandbreite bzw. mehrere parallele Leitungen auf Niveaus, die näher an der Wurzel liegen.

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")

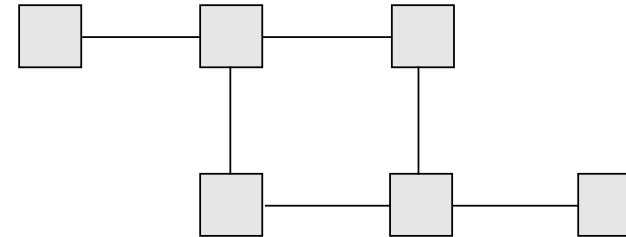
Vernetzung vollständiger Einzelrechner:



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 (Prozessor mit privatem Speicher)
- Kante = dedizierte Kommunikationsleitung

Ausdehnung: i.a. nur wenige Meter

Bewertungskriterien:

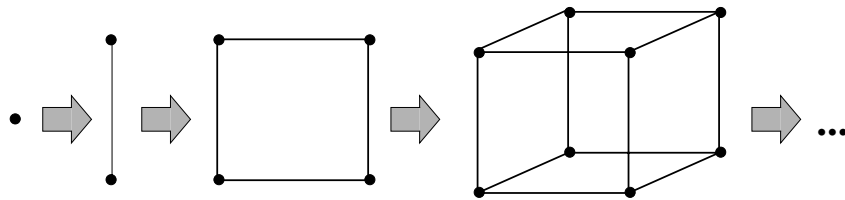
- Gesamtzahl der Verbindungen (bei n Knoten)
- maximale Entfernung zweier Knoten
- durchschnittliche Entfernung
- Anzahl der Nachbarn eines Knotens ("fan out")
- Symmetrie, Homogenität, Skalierbarkeit...
- Routingkomplexität
- Zahl der alternativ bzw. parallel verfügbaren Wege

Technologische Faktoren:

- Geschwindigkeit, Durchsatz, Verzögerung, spezifische Kommunikationsprozessoren...

Hypercube

- Hypercube = "Würfel der Dimension d"



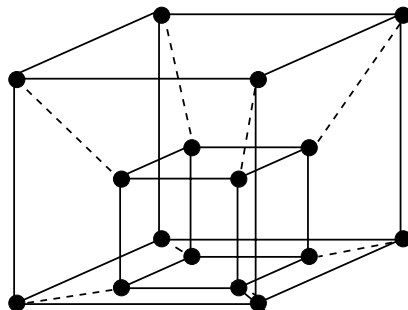
← Draufsicht von der Seite liefert jeweils niedrigere Dimension

→ 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:

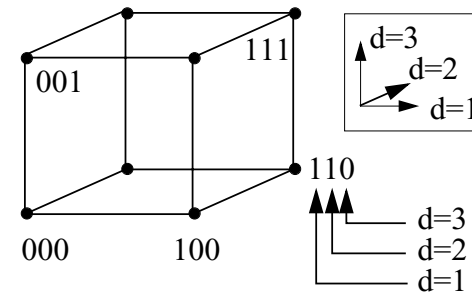
„Nimm zwei Würfel der Dimension d und verbinde korrespondierende Ecken“



4-dimensionaler Würfel

Hypercube der Dimension d

- $n = 2^d$ Knoten
- Anzahl der Nachbarn eines Knotens = d
(Anzahl der "ports" in der Hardware)
- Gesamtzahl der Kanten (= Verbindungen): $d \cdot 2^d / 2 = d \cdot 2^{d-1}$
(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:

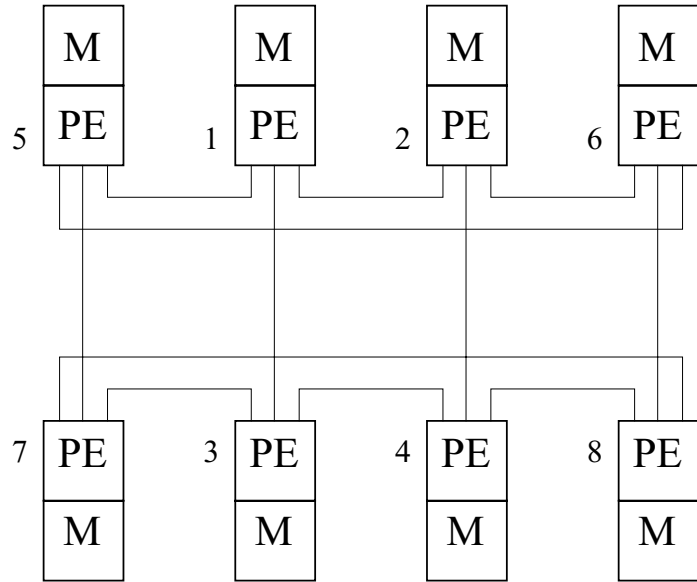
- kurze Weglängen (max. $\log n$)
- einfaches Routing
- viele Wegalternativen (Fehlertoleranz, Parallelität!)

-Nachteile:

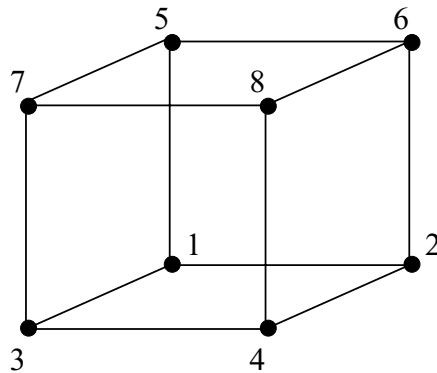
↑ wieviele verschiedene Wege der Länge k gibt es insgesamt?

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

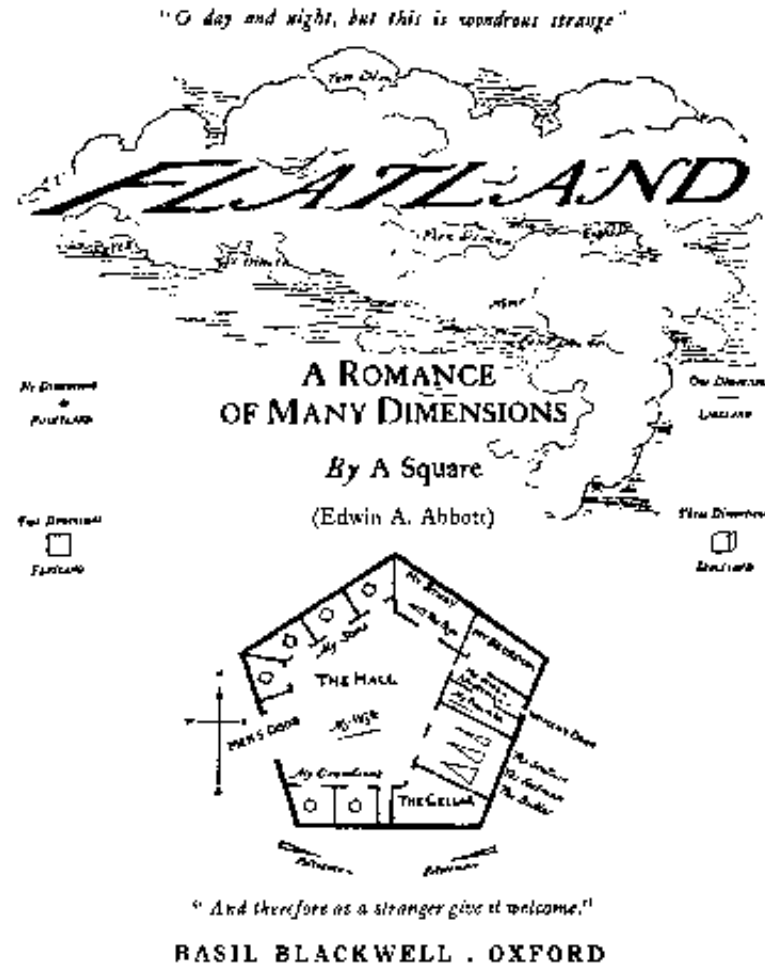
Layout eines Hypercube



Obiger Topologie sieht man zunächst nicht an, dass es sich dabei um einen 3-dimensionalen Würfel handelt!



Flatland (1884)



- Diskussion über Hypercubes und höhere geometrische Dimensionen zwischen "A. Square" und "Sphere"
- Gleichzeitig soziale Satire

§ 16 ---How the Stranger vainly endeavoured to reveal to me in words the mysteries of Spaceland

...

Sphere. ... We began with a single Point, which of course -- being itself a Point -- has only ONE terminal Point. One Point produces a Line with TWO terminal Points.

One Line produces a Square with FOUR terminal Points.

Now you can give yourself the answer to your own question: 1, 2, 4, are evidently in Geometrical Progression. What is the next number?

I. Eight.

Sphere. Exactly. The one Square produces a SOMETHING-WHICH-YOU-DO-NOT-AS-YET-KNOW-A-NAME-FOR-BUT-WHICH-WE-CALL-A-CUBE with EIGHT terminal Points. Now are you convinced?

...

Sphere. How can you ask? And you a mathematician! The side of anything is always, if I may so say, one Dimension behind the thing. Consequently, as there is no Dimension behind a Point, a Point has 0 sides; a Line, if I may so say, has 2 sides (for the points of a Line may be called by courtesy, its sides); a Square has 4 sides; 0, 2, 4; what Progression do you call that?

I. Arithmetical.

Sphere. And what is the next number?

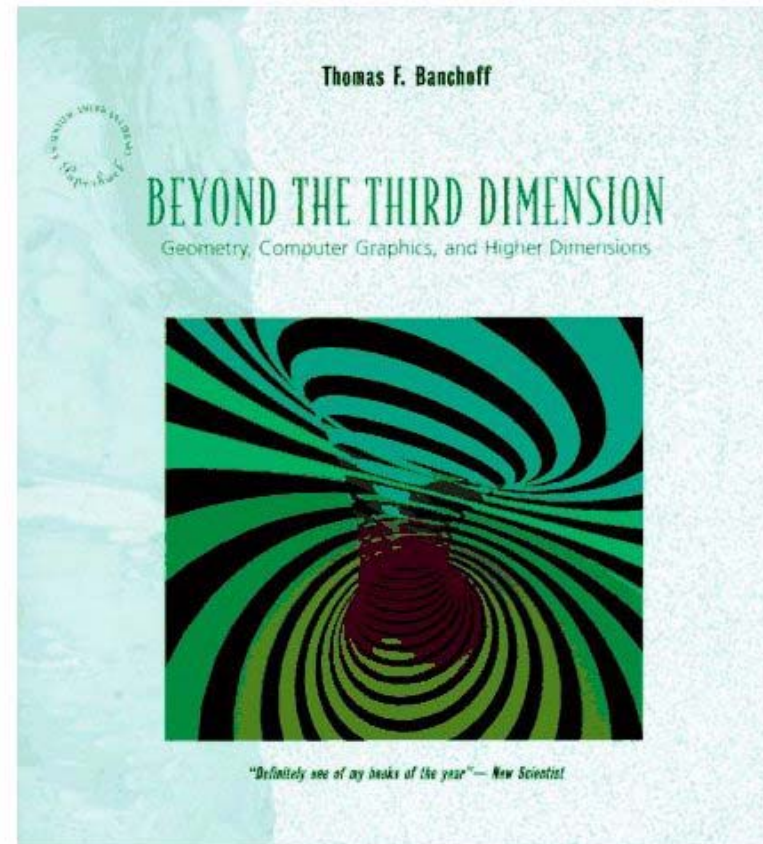
I. Six.

Sphere. Exactly. Then you see you have answered your own question. The Cube which you will generate will be bounded by six sides, that is to say, six of your insides. You see it all now, eh?

“Monster,” I shrieked, “be thou juggler, enchanter, dream, or devil, no more will I endure thy mockeries. Either thou or I must perish.” And saying these words I precipitated myself upon him.

Online-Text erhältlich bei: <http://www.geom.umn.edu/~banchoff/Flatland/> oder <http://www.alcyone.com/max/lit/flatland/>

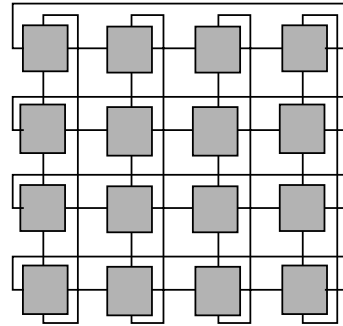
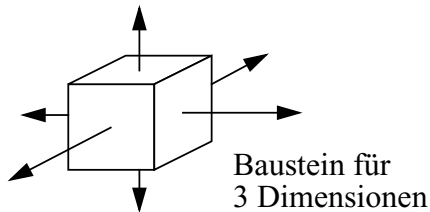
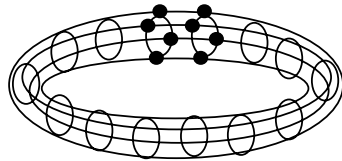
Das *Buch* ist erhältlich in mehreren Ausgaben; z.B.: Abbott, Edwin A.: Flatland. Penguin, 1987, ISBN: 0140076158, DM 11,90



T. F. Banchoff: Beyond the Third Dimension. Scientific American Library

Eine andere Verbindungstopologie: der d-dimensionale Torus

= d-dimensionales "wrap-around Gitter"



- Rekursives Konstruktionsprinzip: „Nimm w_d gleiche Exemplare eines Torus der Dimension $d-1$ und verbinde korrespondierende Elemente zu einem Ring“

- Bei Ausdehnung w_i in Dimension i :

$$n = w_1 \times w_2 \times \dots \times w_d \text{ Knoten;}$$

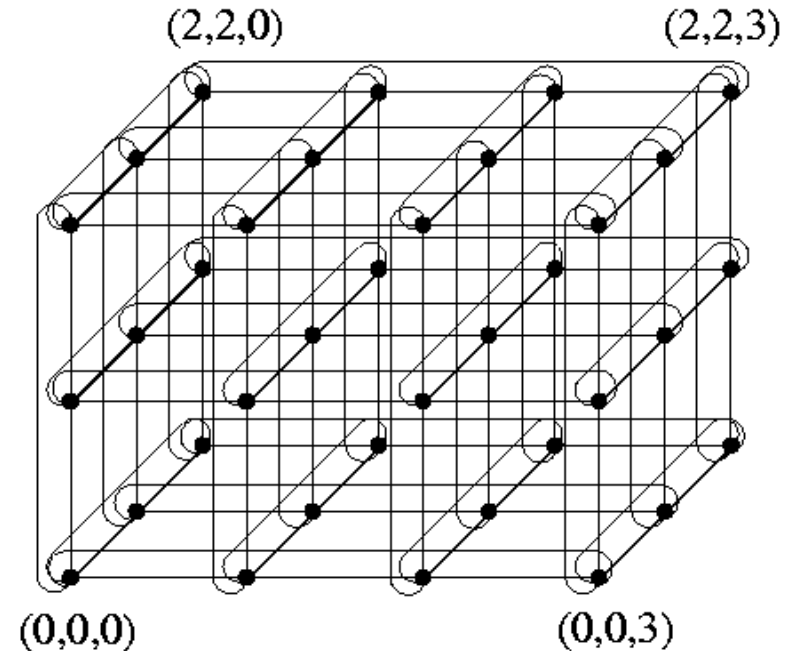
$$\text{mittlere Entfernung zw. 2 Knoten: } \Delta \approx \frac{1}{4} \sum w_i$$

- Ring als Sonderfall $d = 1$!

- Hypercube der Dimension d ist d-dimensionaler Torus mit $w_i = 2$ für alle Dimensionen!

$$\rightarrow \Delta = \frac{1}{4} \sum_d 2 = \frac{1}{4} (2 d) = \frac{d}{2} = \frac{1}{2} \log_2 n$$

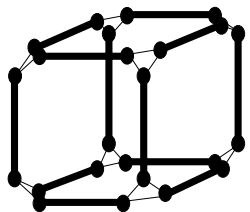
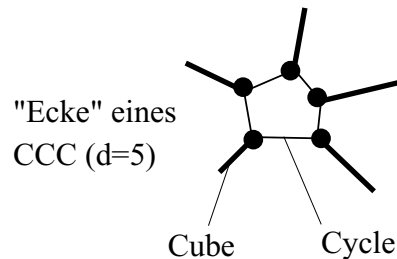
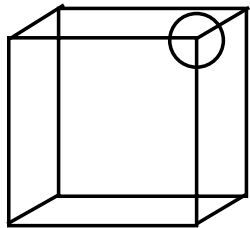
3-dimensionaler Torus



Mit $w_1 = 4, w_2 = 3, w_3 = 3$

Cube Connected Cycles (CCC)

d-dimensionaler Hypercube mit aufgeschnittenen Ecken, die durch Gruppen von d ringförmig verbundenen Knoten ersetzt sind.



Beachte: Jeder Knoten hat immer *drei* Anschlüsse!

Bei Dimension d: $n = d \cdot 2^d$

Maximale / mittlerer Weglänge? ←

Denkübung!
(nicht ganz einfach)

Anzahl der Verbindungen = $3n / 2$
(statt $O(n \log n)$ wie beim Hypercube)

Es gibt viele weitere Verbindungstopologien
(wollen wir hier aber nicht betrachten)

Zufallstopologien (mit max. Knotengrad = k)

Verfahren A:

- 1) Starte mit einem Graphen ohne Kanten.
- 2) Wähle zwei zufällige Knoten, die noch weniger als k Kanten haben, und verbinde diese.
- 3) Wiederhole Schritt 2 solange wie möglich.
- 4) Falls ein unzusammenhängender Graph entsteht, beginne von vorne.

Verfahren B ("greedy graphs"):

Motivation:
Kurze Zyklen vermeiden

- 2) Wähle zwei bel. Knoten *mit maximaler Entfernung* (einschliesslich ∞), die noch weniger als k Kanten haben, und verbinde diese.

Wie gut sind diese Zufallsgraphen bzgl. mittlerer Knotenentfernung und Routingbelastung?

Verzögerungszeiten,
Routingoverhead

Bottleneck

Bem.: Explizites Routing nötig!

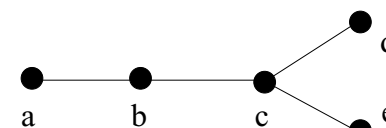
Routing-Belastung eines Knotens:

Zahl von Routen, die durch den Knoten gehen

Routing-Belastung einer Verbindung:

Zahl von Routen, die durch die Verbindung gehen

Beispiel:

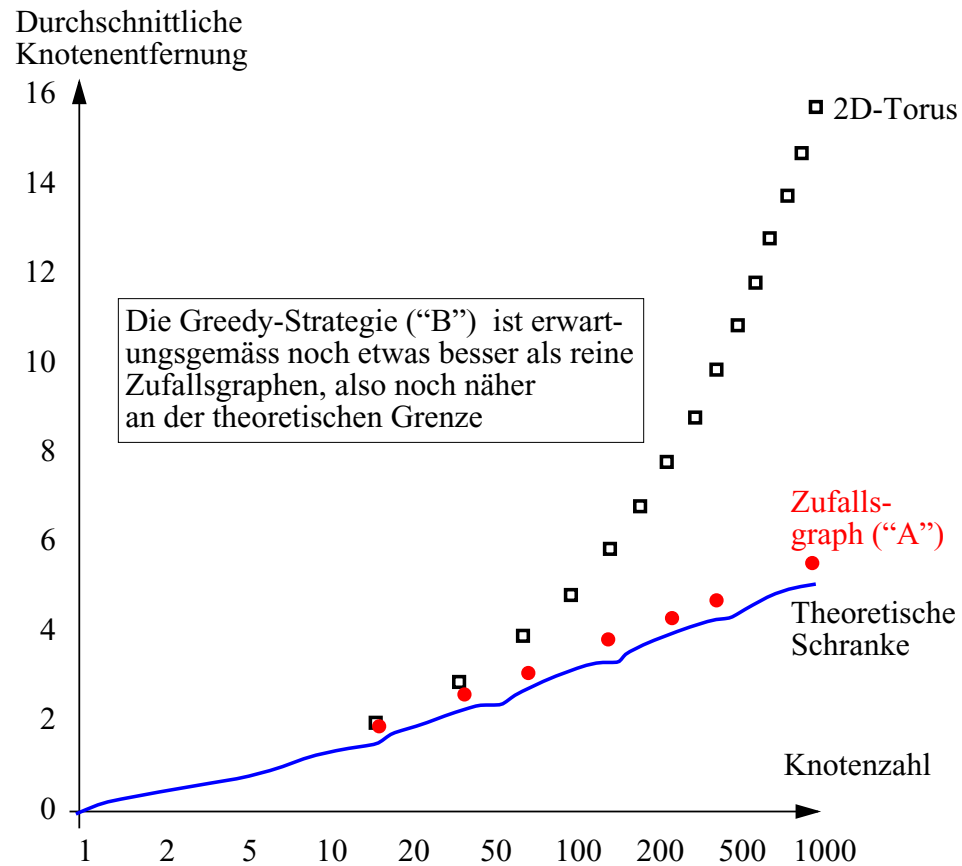


10 Routen von 20 gehen durch c

12 Routen von 20 gehen durch bc

Mittlerer Knotenabstand

- Hier: für Knoten mit max. Grad 4



Quelle:

D. M. N. Prior, M. G. Norman, N. J. Radcliffe, L. J. Clarke: *What Price Regularity?*, Concurrency, Practice and Experience, Vol 2 No 1, pp. 55-78, 1990 (Dort auch Ergebnisse zur Routing-Belastung)