

Verteilte Systeme

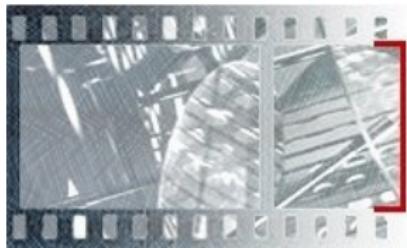
R. Kaiser, R. Kröger, O. Hahm

(HTTP: <http://www.cs.hs-rm.de/~kaiser>
E-Mail: eobert.kaiser@hs-rm.de)

Kai Beckmann
Sebastian Flothow
Alexander Schönborn

Sommersemester 2021

0. Vorspann



<http://www.interaktiv-narrativ.org/media/vorspann.jpg>

Vorspann

- ① Einordnung der Veranstaltung
- ② Organisation der Veranstaltung
- ③ Materialien

Einordnung der Veranstaltung

- Pflichtveranstaltung des Bachelor-Studiums
- Leistungsnachweis:
 - ▶ Prüfungsleistung
 - ▶ Praktikum: separate Studienleistung
- Inhaltliche Voraussetzungen:
 - ▶ Rechnernetze und Telekommunikation
 - ▶ Programmierung: C und Java fürs Praktikum

Organisation der Veranstaltung

- Vorlesung LV 4131

- ▶ 2-stündig (Mi 08:15)
- ▶ online via ZAPP / BigBlueButton (<https://zapp.mi.hs-rm.de/>)
- ▶ Videos auf AMIGO (<https://video.cs.hs-rm.de>)

- Gliederung

- ① Einführung
- ② Netzwerkprogrammierung
- ③ Remote Procedure Calls
- ④ Anwendungsarchitektur
- ⑤ Namens- und Verzeichnisdienste
- ⑥ Sicherheit
- ⑦ Global Time
- ⑧ Verteilte Transaktionsmechanismen
- ⑨ Verteilte Dateisysteme

Details siehe auch auf der Homepage von Prof. Kaiser:

<http://www.cs.hs-rm.de/~kaiser>

Organisation der Veranstaltung (2)

- Praktikum LV 4132

- ▶ 2-stündig
- ▶ 5 Gruppen
 - ★ Vgl. Stundenplan

- Inhalt

- ▶ Hamsterasyl
- ▶ RPC
- ▶ SunRPC
- ▶ HamsterIoT
- ▶ REST

Organisation der Veranstaltung (3)

Termine:

KW	Vorlesung Mittwochs	Gruppen A-B Mittwochs	Gruppen C-F Freitags	Montags	Abgabe Blatt	Vorrechnen Theo. Blatt	Abnahme Prak. Blatt	Vorstellen Prak. Blatt	Fragen zu Prakt. Blatt
15	14.04.20	-	-						
16	21.04.20	21.04.20	23.04.20					1: Hamsterlib	1: Hamsterlib
17	28.04.20	28.04.20	30.04.20						1: Hamsterlib
18	05.05.20	05.05.20	07.05.20	03.05.20	1: Hamsterlib	2: Kap. 1+2		3. Hamster-RPC	3. Hamster-RPC
19	12.05.20	12.05.20	14.05.20				1: Hamsterlib		3. Hamster-RPC
20	19.05.20	19.05.20	21.05.20	17.05.20	3. Hamster-RPC	4: Kap. 2-4		5: SunRPC	5: SunRPC
21	26.05.20	26.05.20	28.05.20				3. Hamster-RPC		5: SunRPC
22	02.06.20	02.06.20	04.06.20	31.05.20	5: SunRPC	6: Kap 3+4		7: HamsterIoT	7: HamsterIoT
23	09.06.20	09.06.20	11.06.20				5: SunRPC		7: HamsterIoT
24	16.06.20	16.06.20	18.06.20	14.06.20	7: HamsterIoT	8: Kap 5+6		9: HamsterREST	9: HamsterREST
25	23.06.20	23.06.20	25.06.20				7: HamsterIoT		9: HamsterREST
26	30.06.20	30.06.20	02.07.20			10: Kap 8			9: HamsterREST
27	07.07.20	07.07.20	09.07.20	05.07.20	9: HamsterREST		9: HamsterREST		

Organisation der Veranstaltung (4)

● Leistungsnachweis

► Prüfungsleistung LV 4131:

- ★ Präsenzlausur (90 min.)
- ★ Zum Bestehen müssen mind. 50% der möglichen Punkte erreicht werden

► Praktikum LV 4132:

- ★ Bewertung von 5 ausgewählten Praktikumsaufgaben mit Punkten
- ★ Abgabe für alle Teilnehmer zu fixem Termin (Nacht zu Montag 4:00)
- ★ Automatisierte Plagiatsprüfung (genehmigtes Verfahren)
- ★ Punkte werden addiert und ergeben die Praktikumsnote
- ★ Präsentation von Papierübungen zum Erwerb von Bonuspunkten

Materialien

- ① Folien zur Vorlesung
- ② Übungsblätter
- ③ Lehrbücher (vgl. Modulbeschreibung)

Tanenbaum, van Steen: *Verteilte Systeme* (++)

Pearson

ISBN 978-3-8273-7293-2

49,95 €



Coulouris, Dollimore, Kindberg, Blair: *Distributed Systems*

Pearson

ISBN 978-0132143011

147,95 €



- ④ E-Learning-Material

Details siehe auch auf der Homepage von Prof. Kaiser: <http://www.cs.hs-rm.de/~kaiser>

1. Einführung



<http://gymnasium-blomberg.de/wp-content/uploads/2013/05/Wegweiser.jpg>

Inhalt

1. Einführung

1.1 Geschichtliche Entwicklung

- Halbleitertechnologie
- Kommunikationstechnologie
- Systemtechnologie

1.2 Grundbegriffe verteilter Systeme

- Definitionen und Beispiele
- Transparenz
- Betriebssysteme

Motivation und Geschichte

Technologische Veränderungen:

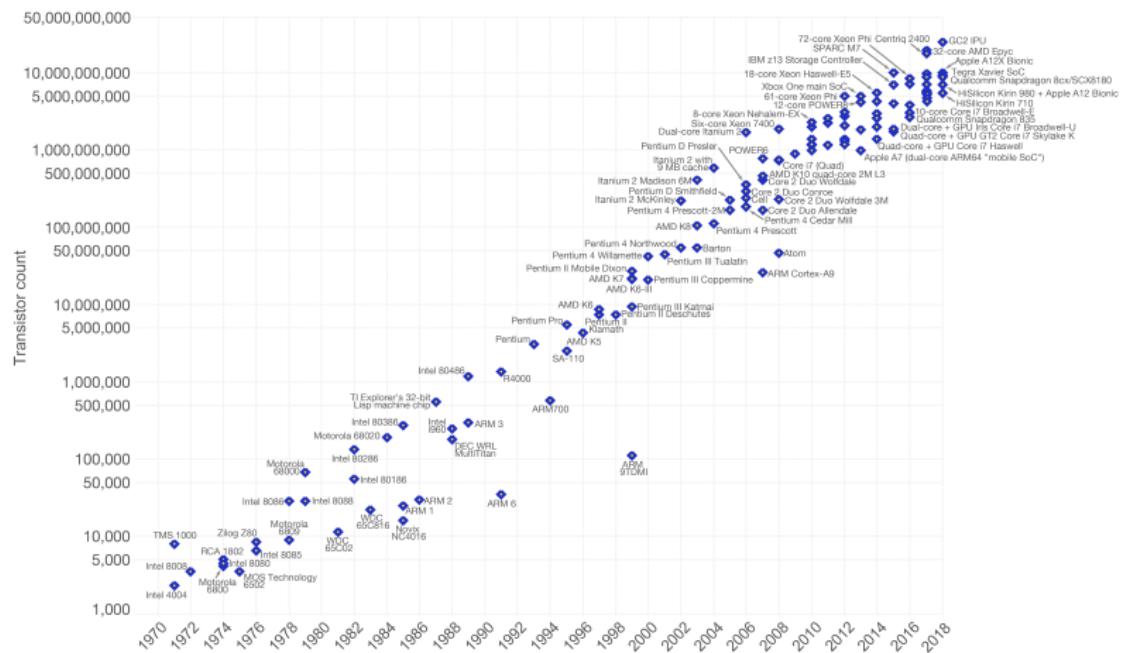
- ① Halbleitertechnologie
- ② Kommunikationstechnologie
- ③ Systemtechnologie

Halbleitertechnologie: Leistung und Kosten

- Speicherchips:
 - ▶ 1973: 4 kBit
 - ▶ 1985: 64 kBit
 - ▶ 1998: 64 MBit
 - ▶ 2008: 16 GBit
 - ▶ 2018: 128 GBit (Samsung DDR4 DRAM)
- Gesetz von Moore (1965): Alle zwei Jahre (18 Monate) verdoppelt sich die Zahl der Transistorfunktionen auf der gleichen Grundfläche.
- Entwicklung der Kosten je Transistorfunktion auf ca. 1/10 alle vier Jahre.
- Immer wieder wurde das Ende des Gesetzes vorausgesagt.
- Neuere Technologien: Z-RAMs, MRAMs, FeRAMs (nicht-flüchtig),
...

CPU-Komplexität

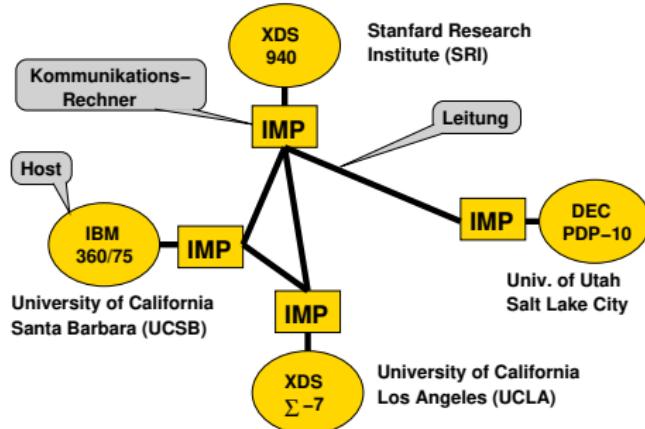
Entwicklung der CPU-Komplexität



https://upload.wikimedia.org/wikipedia/commons/8/8b/Moore's_Law_Transistor_Count_1971-2018.png

Vom ARPANET zum Internet

- 1958 Gründung der Advanced Research Projects Agency (ARPA) durch US Dept of Defense (DoD) als Reaktion auf Sputnik
- 1968 Idee des „Internet“ als „tool to create critical mass of intellectual resources“ (Licklider, Taylor)
Hauptplaner: *Vinton Cerf, Bob Kahn*
- 1969 erstes funktionsfähiges Netz, gemietete 50 kBit/sec Leitungen, Interface Message Processors von BBN (RFC 1)



Vom ARPANET zum Internet (2)

1972 erste öffentliche Demo (remote login)

- Network Control Protocol (NCP) als Protokoll (Menge von Regeln für die Kommunikation)
- Hauptnutzung: Terminal-Sitzungen, Dateitransfer, Electronic Mail

1974 Grundzüge der TCP/IP-Protokolle in Papier von Cerf/Kahn
IP=Internet Protocol, TCP=Transmission Control Protocol,
Standardisierung in den Folgejahren

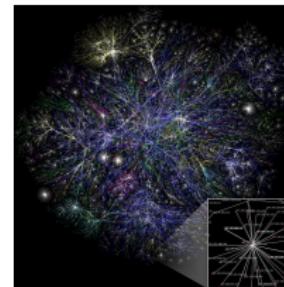
1982 Übergang zu den heutigen Internet Protokollen TCP/IP Version 4

ab 1983 Verbreitung von TCP/IP durch Berkeley UNIX 4.2 BSD, freie Verfügbarkeit des Quellcodes

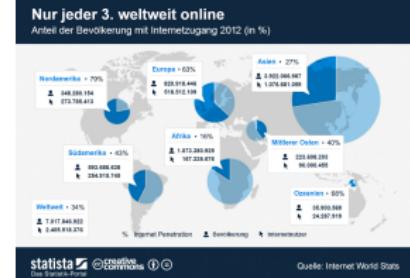
1986 Erstes Treffen der Internet Engineering Task Force (IETF) in San Diego

Wachstum des Internets

- Anzahl AS (Autonomer Systeme, admin. Routing-Domäne)
 - ▶ Verdopplung alle 5 Jahre (vgl. PhysOrg.com)
 - ▶ Kern stabil
 - ▶ Wachstum an der Peripherie
- Verkehrsdaten
 - ▶ Wachstumsraten von ca. 2,8 pro Jahr angenommen (Caspian Networks)
- Internet-Nutzer
 - ▶ 2018: die Hälfte der Menschen ist online
 - ▶ (beinahe) Verdopplung in letzten zehn Jahren
 - ▶ Stärkstes Wachstum außerhalb EU, Japan, USA



https://en.wikipedia.org/wiki/File:Internet_map_1024.jpg



<https://de.statista.com/infografik/1049/weltbevoelkerung-mit-mit-internetzugang>

Das World-Wide Web (WWW)

- ab 1970 Arbeiten zu Hypertext-Systemen (durch Zeiger verbundenes verteiltes Geflecht von Knotendokumenten mit einfachen Navigationsmöglichkeiten) von Ted Nelson (Project Xanadu)
- 1990 Vorschlag für ein Hypertext-Projekt am CERN in Genf durch Tim Berners-Lee und Robert Cailliau: Wiege des World-Wide Web
- 1991 Entwicklung einer ersten Version auf NeXT-Rechner
Präsentation auf Hypertext-Konferenz
- 1992 Herausgabe einer freien Version von Web-Server und Browser (Unix-basiert) durch CERN, Ende des Jahres: weltweit ca. 50 Web-Server
- 1993 Marc Andreessen, Eric Bira (NCSA, Univ. of Illinois) geben erste Version des Mosaic Browsers heraus, gründen später Netscape
- 1994 Für Microsoft ist WWW noch kein Thema. Bill Gates: „... an Internet Browser is a trivial piece of software. There are at least 30 companies that have written creditable Internet browsers, so that's nothing...“
- Ende '95 Microsoft greift ein...

Ubiquitous Networks

- 1982 An der Carnegie Mellon Universität wird ein Getränkeautomat mit dem Internet verbunden
- 1995 Veröffentlichung der ersten IPv6-Spezifikation
- 1996 Hewlett-Packard und Nokia veröffentlichen mit dem OmniGo 700LX und dem 9000 Communicator erste Smartphone-Vorläufer
- 1997 Kristofer S. J. Pister, Joe Kahn und Bernhard Boser (Berkeley) präsentieren einen Forschungsantrag zum Thema *Smart Dust*
- 1998 Gründung von Google
- 1999 Kevin Ashton prägt den Begriff des *Internet of Things*
- 2001 Wikipedia geht online
- 2004 Gründung von Facebook
- 2007 Apple veröffentlicht das erste iPhone
- 2014 Die IETF-Arbeitsgruppe *CORE* veröffentlicht die erste Spezifikation zum Constrained Application Protocol (CoAP)

Heutige Klassen von Rechensystemen

- Personalcomputer (PC, Desktop), Workstations
- Server, Großrechner (Mainframes)
 - ▶ hochverlässliche Verarbeitung von Massendaten
 - ▶ Hoch- bis Höchstleistungs-Ein-/Ausgabe-Einheiten
 - ▶ Server erbringen Dienstleistungsfunktionen in Rechnernetzen
 - ▶ Mainframes sind z.T. wegen nicht mehr wartbarer Altprogramme erforderlich (Legacy-Systeme)
- Supercomputer
 - ▶ Vielzahl von Prozessoren/Knoten
 - ▶ hohe Verarbeitungsleistung
 - ▶ Beispiel: numerische Berechnungen zur Wettervorhersage
- Embedded Computer (eingebetteter Rechner)
 - ▶ Teil von Maschinen, Geräten oder Anlagen
 - ▶ Rechensystem steht gegenüber der Funktionalität des umgebenden Systems im Hintergrund
 - ▶ Cyber-Physical Systems/Industrie 4.0

Aktuelle Entwicklung

- Heutige Rechner werden zwar immer leistungsfähiger und besitzen ein immer besseres Preis-/Leistungsverhältnis, erreicht wird dies aber nur durch graduelle Verbesserungen bekannter Techniken.
- Ebenen
 - ▶ Prozessoren
 - ★ kürzere Entwicklungszyklen durch verbesserte Design-Werkzeuge
 - ★ Konzentration auf Prozessoren mit Intel-Befehlssatz im Office-Bereich
 - ★ ursprünglich viele μ Controller-Typen in Embedded Systemen, mittlerweile Konzentration auf ARM-Architektur
 - ★ Multicore-Prozessoren
 - ▶ Systeme
 - ★ verstärkter Einsatz von Systemen mit vielen Knoten
 - ★ z.B. Blade-Server, HPC-Cluster
 - ▶ Netzwerke
 - ★ steigende Kommunikationsbandbreiten
 - ★ verschiedenartige Dienstgüteanforderungen
 - ★ mobile Knoten

Aktuelle Entwicklung (2)

- Virtualisierung
 - ▶ Virtuelle Maschinen (VMs)
 - ▶ Speichervirtualisierung (Software Defined Storage)
 - ▶ Virtuelle Netze (Software Defined Networks, SDNs)
- Virtuelle Infrastrukturen (Cloud Computing)
- Internet der Dinge und der Dienste, Industrie 4.0/Industrial Internet
- Big Data

Definitionen

- „*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
- „*A distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system.*“
Maarten van Steen, Andrew S. Tanenbaum

Grundbegriffe verteilter Systeme

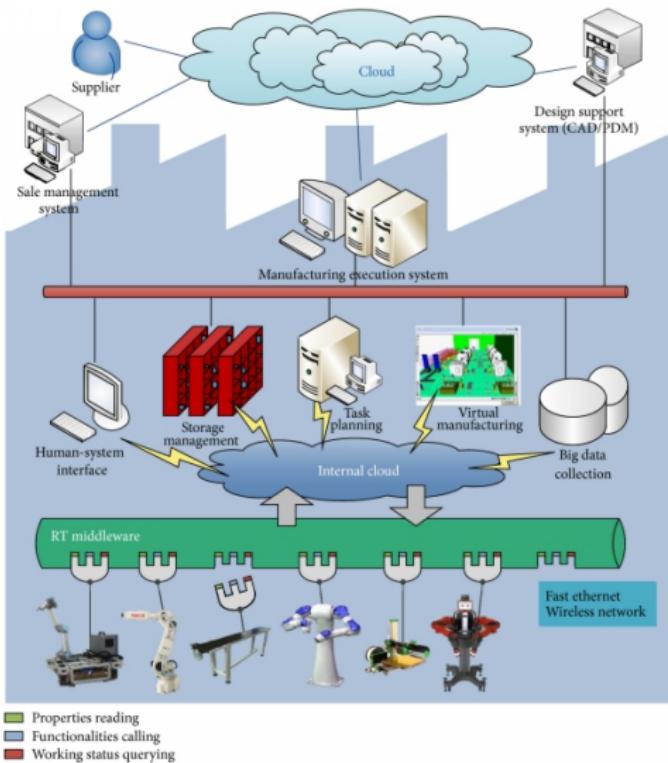
- Enge Kopplung: Zwei Software-Komponenten sollen **eng gekoppelt** heißen, wenn sie durch gemeinsame Nutzung von Betriebsmitteln (Sharing) kommunizieren. Diese können z.B. sein:
 - ▶ gemeinsam genutzte typisierte Objekte
 - ▶ gemeinsam genutzte Speichersegmente
- Lose Kopplung: Zwei Software-Komponenten sollen **lose gekoppelt** heißen, wenn sie durch Nachrichtenaustausch (Message Passing) kommunizieren (höhere Autonomie der Komponenten).
- Analog existieren auf der Ebene der Anwendungsprogrammierung entsprechende Paradigmen, die auf Sharing oder auf Message Passing angelegt sind.

Verteiltes Programm/Verteiltes System

- Ein **verteiltes Programm** besteht aus einer Menge von lose gekoppelten Software-Komponenten, die (durch Nachrichtenaustausch) bzgl. einer gemeinsamen Problemlösung kooperieren.
- Ein verteiltes Programm beinhaltet
 - ▶ einen **verteilten Zustand** (in den jeweiligen Software-Komponenten).
 - ▶ **verteilte Kontrolle/Koordination**, um die gemeinsame Problemlösung zu bewerkstelligen.
- Ein **verteiltes System** ist ein Rechensystem, das ein verteiltes Programm ausführt.

Beispiele verteilter Systeme

- Das Internet
- Peer-to-peer Netze
- Automatisierte Fertigungsanlagen



Transparenzarten

- Transparenz bedeutet, anders als im üblichen Sprachgebrauch, die *Unsichtbarkeit* einer Eigenschaft eines Mehrrechner-Systems.
- Transparenzarten gemäß International Standards Organization (ISO) und Advanced Network Systems Architecture (ANSA)
 - ▶ **Ortstransparenz:** keine Kenntnis des Ortes einer Komponente notwendig, insbesondere ist der Ort nicht Teil des Namens.
 - ▶ **Zugriffstransparenz:** Form des Zugriffs ist unabhängig, ob Komponente lokal oder entfernt ist.
 - ▶ **Migrationstransparenz:** Komponente kann verlagert werden, ohne dass sich die Schnittstelle zu den Nutzern ändert.
 - ▶ **Replikationstransparenz:** Benutzungsschnittstelle ist unabhängig von der Anzahl der Kopien einer Komponente, Verfügbarkeit als qualitative Eigenschaft kann verbessert werden.

Transparenzarten (2)

- Weitere Transparenzarten:

- ▶ **Nebenläufigkeitstransparenz:** Nebenl. Nutzung von Komponenten so, als ob diese privat wären, z.B. durch autom. Sperren, vgl. Datenbanken.
- ▶ **Fehlertransparenz:** Ein eingetretener Fehlzustand wird an der Benutzungsstelle nicht sichtbar, sondern durch Redundanz im unterlagerten System maskiert.
- ▶ **Skalierungstransparenz:** System und Applikationen können sich vergrößern, ohne dass die Systemstruktur oder die Applikationsalgorithmen geändert werden müssen.
- ▶ **Leistungstransparenz:** Neukonfiguration des Systems bei Veränderung der Last, um die Leistung zu verbessern.

Welche Transparenzart ist relevant?

- Transparenz trägt zur Vereinfachung des Umgangs und der Programmierung des Systems bei, da der betreffende Aspekt vom Nutzer des Systems nicht beachtet werden muss.
- Ein verteiltes System sollte möglichst alle angegebenen Transparenzarten verwirklichen, um eine möglichst einheitliche Systemsicht zu erreichen.
- Vollkommene verteilte Systeme, die von allen Aspekten abstrahieren, existieren derzeit nicht.
- Die Unterstützung einzelner Transparenzarten (z.B. Ortstransparenz) ist fortgeschritten.

LOS - NOS - DOS

- Stufen der Entwicklung können an folgenden BS- Architekturklassen festgemacht werden:
- **LOS (Local Operating System):**
übliches Knoten-Betriebssystem ohne Unterstützung für Verteiltheit.
Beispiele:
 - ▶ IBM MVS,
 - ▶ UNIX System III,
 - ▶ DOS, Windows 3.1,
 - ▶ ...

LOS - NOS - DOS (2)

- **NOS (Network Operating System):**

Betriebssystemerweiterung von ev. verschiedenen LOS' für ein Multicomputer-System, um gewisse Funktionen bzgl.

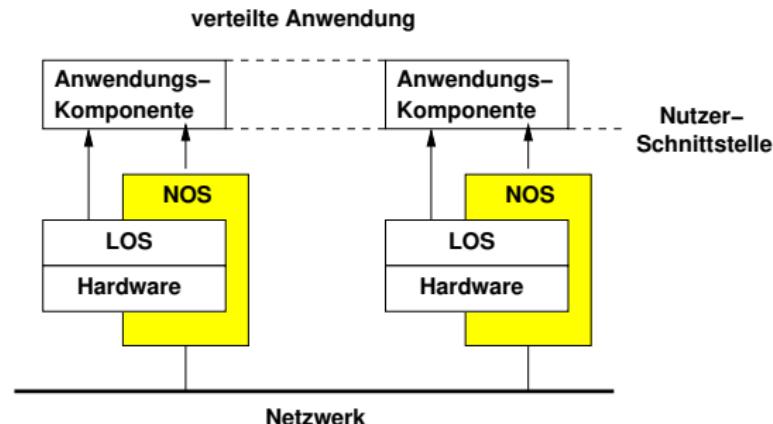
- ▶ Dateisystem,
- ▶ Schutz (Benutzerverwaltung),
- ▶ entfernter Programmausführung

systemweit, mehr oder weniger transparent, bereitzustellen.
Beispiele:

- ▶ Novell NetWare,
- ▶ Windows for Workgroups,
- ▶ UNIX Yellow Pages (NIS) und Network File System (NFS)

LOS - NOS - DOS (3)

- Prinzipielle Struktur eines Netzwerkbetriebssystems (NOS):



- Die unterlagerten lokalen Knoten-Betriebssysteme (LOS) können gleich oder verschieden sein.

Beispiele:

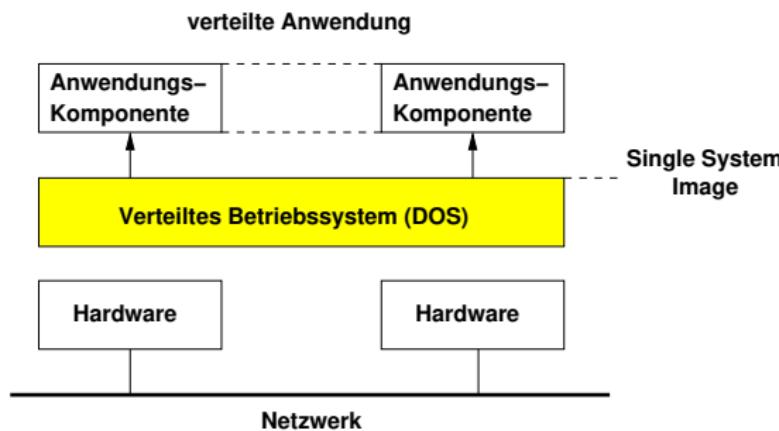
- Netware Client für DOS, NT, ... ; Netware Server ist spezielles BS.
- NFS Client für UNIX, NT, ...

LOS - NOS - DOS (4)

• DOS (Distributed Operating System):

Ein verteiltes Betriebssystem ist ein grundständiges BS, das

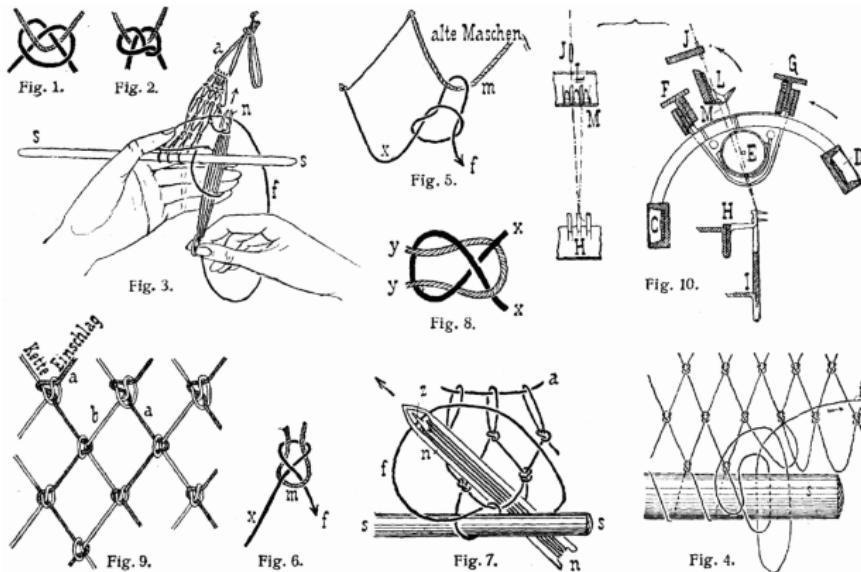
- ▶ seinen Nutzern eine einheitliche Systemsicht auf ein Multicomputer-System bietet (s.o.),
- ▶ zur Realisierung der Transparenzeigenschaften auf Algorithmen basiert, die unter verteilter Kontrolle unter Austausch von Nachrichten ablaufen.



Zusammenfassung

- Physikalische Grenzen in der Halbleitertechnologien erfordern neue Ansätze zur Steigerung der Leistungsfähigkeit.
- Die Allgegenwärtigkeit des Internets lässt verteilte Systeme immer wichtiger werden.
- Die zugrundeliegende Verteiltheit der Komponenten bleibt für den Nutzer und Programmierer eines verteilten Systems unsichtbar (→ Transparenz).

2. Netzwerkprogrammierung



<https://de.wikipedia.org/wiki/Fischernetz#/media/File:L-Netze.png>

Inhalt

2. Netzwerkprogrammierung

2.1 Einführung

2.2 Grundlagen der Kommunikation

- Anzahl der Teilnehmer
- Adressierung
- Pufferung
- Synchronisation
- Kommunikationsmuster
- Semantik von Nachrichten

2.3 Internet-Protokolle

2.4 Grundlagen der Socket-Programmierung

2.5 BSD Sockets

2.6 Java Sockets

2.7 Server-Architektur

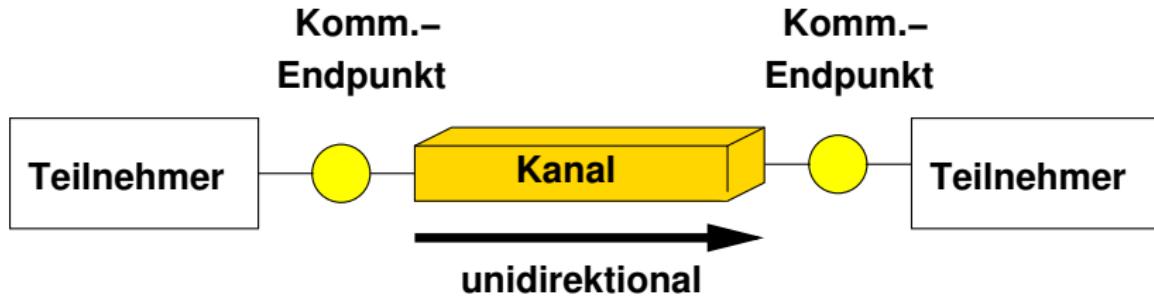
Einführung

Ziele

- Kennenlernen eines allgemeinen nachrichtenorientierten Kommunikationsdienstes hoher praktischer Relevanz (Internet mit TCP/IP-Protokollen) und dessen Programmierschnittstelle (Sockets)
- API zur TCP/IP-Funktionalität der LV „Rechnernetze“ (vgl. 2. Sem.)
- Höhere Kommunikationsdienste (Remote Procedure Calls (RPCs), Web Services) bauen hierauf auf und werden z.T. im weiteren dieser LV behandelt.
- **Merke:** Höhere Kommunikationsdienste oder Middleware-Plattformen offerieren eine abstraktere, auf das jeweilige Kooperations-Paradigma ausgerichtete Schnittstelle, basieren aber intern auf den hier zunächst besprochenen Konzepten eines unterlagerten Kommunikationssystems.

Grundlagen der Kommunikation

- Jegliche Interaktion zwischen Beteiligten verlangt eine zugrunde liegende Kommunikationsmöglichkeit.
- Kommunikationskanal
 - ▶ Einrichtung zur Verbindung/Kopplung der Kommunikationspartner heisst Kommunikationskanal oder Kanal.
- Richtung des Nachrichtenflusses eines Kanals
 - ▶ Ein Kanal heisst gerichtet oder unidirektional, wenn ein Prozess ausschließlich die Sender-Rolle, der andere ausschließlich die Empfänger-Rolle ausübt, ansonsten heißt er ungerichtet oder bidirektional



Aspekte der Kommunikation

Wichtige Aspekte der Kommunikation

- ① Anzahl Kommunikationspartner
- ② Adressierung
- ③ Pufferung
- ④ Synchronisation
- ⑤ Kommunikationsmuster
- ⑥ Nachrichtenstruktur

Anzahl der Teilnehmer

Anzahl der Kommunikationsteilnehmer eines Kanals

- Genau zwei:
 - ▶ einfacher Fall
 - ▶ Regelfall
- Mehr als zwei:
 - ▶ Für bestimmte Anwendungen können Gruppenkommunikationsformen sinnvoll sein.
 - ▶ sogenannter Multicast-Dienst
 - ▶ Spezialfall: Broadcast
 - ▶ vgl. 2.3.

Adressierung

Direkte Adressierung

- Kommunikationspartner haben eindeutige Adressen.
- Benennung ist **explizit** und **symmetrisch**: ein Sender muss den Empfänger benennen und umgekehrt.

Beispiel:

SEND (P, message) - Sende Nachricht an Prozess P.

RECEIVE (Q, message) - Empfange Nachricht von Prozess Q.

- Asymmetrische Variante (z.B. für Server-Prozesse): Nur der Sender benennt den Empfänger, dem Empfänger (Server) wird mit dem Empfang i.d.R. die Identität des Senders bekannt.

Beispiel:

SEND (P, message)

RECEIVE (sender_id , message)

Adressierung (2)

Indirekte Adressierung

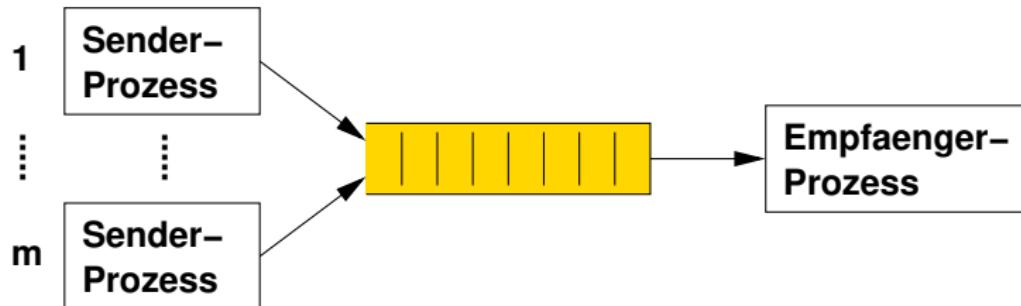
- Kommunikation erfolgt indirekt über zwischengeschaltete Instanzen.
- Vorteile:
 - ▶ Verbesserte Modularität
 - ▶ Menge der Partner kann transparent restrukturiert werden, z.B. nach Ausfall eines Beteiligten.
 - ▶ Erweiterte Zuordnungsmöglichkeiten von Sendern und Empfängern, wie z.B. m:1, 1:n, m:n.
 - ▶ Zwischeninstanz kann
 - ★ nur weiterleiten
 - ★ speichern und weiterleiten (store-and-forward)
 - ★ Nachrichten transformieren

Beispiel indirekte Adressierung

Mailbox:

SEND (mbox, message) - Sende eine Nachricht an Mailbox mbox.

RECEIVE (mbox, message) - Empfange Nachricht von Mailbox mbox.



Pufferung

- Kapazität eines Kanals:

Anzahl der Nachrichten, die vorübergehend in einem Kanal gespeichert werden können, um Sender und Empfänger zeitlich zu entkoppeln.

- Die Pufferungsfähigkeit eines Kanals wird i.d.R. durch einen Warteraum/Warteschlange erreicht.
- In verteilten Systemen wird ein Warteraum i.d.R. auf der Empfängerseite gehalten (Rendezvous-Seite).
- Pufferung kann ordnungserhaltend sein oder auch die Sendeordnung verändern

Pufferung (2)

Keine Pufferung (Kapazität Null)

- Ungepufferte Kommunikationsform
- Sender und Empfänger werden zeitlich sehr eng koppelt
- auch als *Rendezvous* bezeichnet.
- Rendezvous wird häufig als zu inflexibel angesehen.
- Beispiel:
 - ▶ Ein Sender wird blockiert, wenn seine SEND-Operation vor einer entsprechenden RECEIVE-Operation stattfindet. Wird dann die entsprechende RECEIVE-Operation ausgeführt, wird die Nachricht ohne Zwischenspeicherung unmittelbar vom Sender- zum Empfänger-Prozess kopiert.
 - ▶ Findet umgekehrt RECEIVE zuerst statt, so wird der Empfänger bis zum Aufruf der SEND-Operation blockiert.
- Beispiel: Hoare: Communicating Sequential Processes (CSP).
- Beispiel: Kommunikation zwischen Ada-Tasks.
- Beispiel: Kommunikation zwischen Threads in L4 Mikrokern.

Pufferung (3)

Beschränkte Kapazität

- Kanal kann zu einem Zeitpunkt maximal N Nachrichten enthalten (Warteraum der Kapazität N).
- SEND-Operation bei nicht-vollem Warteraum:
 - ▶ Nachricht wird im Warteraum abgelegt,
 - ▶ Sendeprozess fährt in seiner Berechnung fort.
- Warteraum voll (er enthält N gesendete aber noch nicht empfangene Nachrichten):
 - ▶ Sender wird blockiert, bis ein freier Warteplatz vorhanden ist, oder die Nachricht wird verworfen.
 - ▶ Analog wird ein Empfänger bei Ausführung einer RECEIVE- Operation blockiert, wenn der Warteraum leer ist.

Pufferung (4)

Konsequenzen

- Gepufferte Kommunikation bewirkt zeitlich lose Kopplung der Kommunikationspartner.
- Übergabe der Nachricht an das Kommunikationssystem bedeutet für den Sender nicht, dass der Empfänger die Nachricht erhalten hat. Er kennt i.d.R. auch keine maximale Zeitdauer, bis dieses Ereignis eintritt.
- Wenn dieses Wissen wesentlich für den Sender ist, muss dazu eine explizite Kommunikation zwischen Sender und Empfänger durchgeführt werden:

Prozess P (Sender):

...

send (Q, message);

receive (Q, reply);

...

Prozess Q:

...

receive (P, message);

send (P, "acknowledgement");

...

Synchronisation

Blocking oder Blockierend

- Sender kann beim Aufruf blockiert werden

Non-blocking oder Nicht-blockierend

- Sender kann beim Aufruf nicht blockiert werden
- kann damit unmittelbar weiterarbeiten

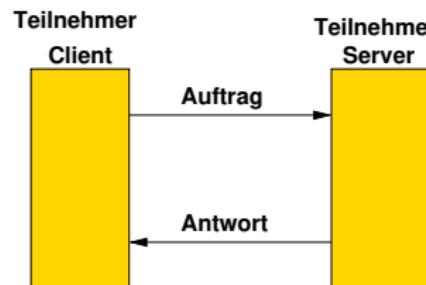
Kommunikationsmuster

One-Way

- Einzelnachricht ohne Antwort oder Quittung

Request/Response oder Auftrag/Antwort

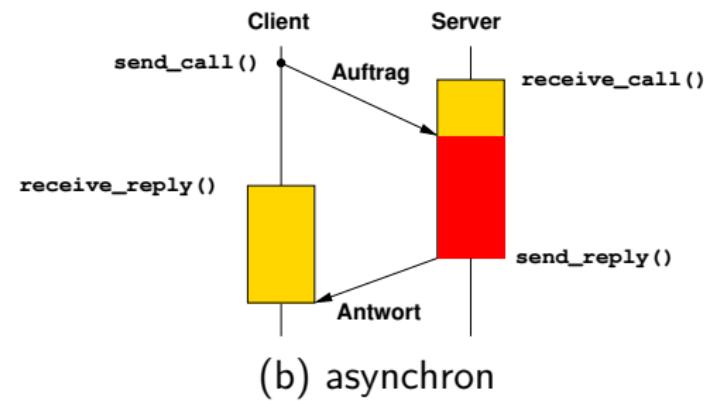
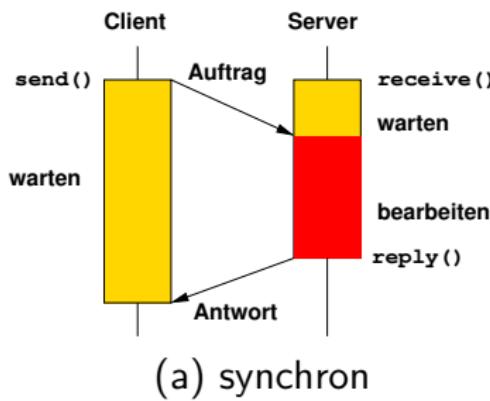
- Client-Rolle (Auftraggeber)
- Server-Rolle (Auftragnehmer)
- häufig blockierend beim Auftraggeber (vgl. Standard-RPC)



Erweiterung

Unterschiedliche Synchronität bei Request/Response:

- synchroner Aufruf: Sender blockiert bis zum Abschluss des Kommunikationsvorgangs (hier: Eintreffen der Antwort).
⇒ keine Parallelität.
- asynchroner Aufruf: Sender wird nur für die Einleitung des Kommunikationsvorgangs (Übergabe der Nachricht an das Kommunikationssystem) verzögert.



Kommunikationsmuster (2)

Publisher / Subscriber-Modell

- Nachrichten klassifiziert in Topics oder Event Channel
- Empfänger abonnieren Topics (Subscriber)
- Sender publizieren Nachrichten oder Events (Publisher)
- Modell erlaubt transparentes Senden einer Nachricht an mehrere Empfänger!
- Beispiele: CORBA Notification Service, OMG DDS, MQTT

Kommunikationsmuster (3)

komplexere Kommunikationsmuster

- nicht besonders üblich mit einfachen Kommunikationssystemen
- Ausnahmebeispiel: Three-Way Handshake zwischen zwei Teilnehmern zum zuverlässigen Verbindungsaufbau
- komplexere Formen entstehen bei Gruppenkommunikation
- sehr verbreitet auf höheren Ebenen
- Beispiel: Geschäftsprozesse

Semantik von Nachrichten

Bytestrom

- Übergebene Nachrichten verschiedener SEND-Operationen sind als Einheiten nicht mehr identifizierbar.
Nachrichtengrenzen gehen verloren.
- Der Empfänger (und das Nachrichtensystem) sehen ausschließlich eine Folge von Zeichen (Bytestrom).
- Beispiel: UNIX pipes

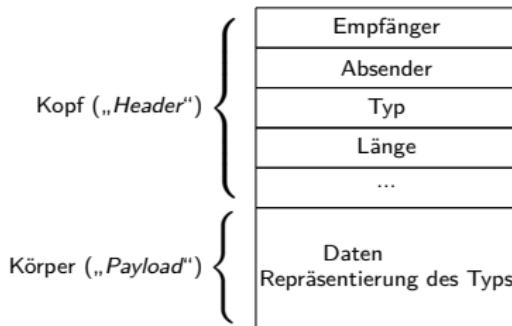
Nachrichtencontainer

- Nachrichten sind für Sender und Empfänger identifizierbare Einheiten fester oder variabler Länge.
Nachrichtengrenzen bleiben erhalten.
- Die korrekte Interpretation der internen Struktur einer Nachricht obliegt den Kommunikationspartnern.
- Beispiel: UNIX message queues

Nachrichtenstruktur

Typisierte Nachrichten

- Nachrichten haben eine typisierte Struktur.
- Typ ist Sender und Empfänger und z.T. dem Kommunikationssystem bekannt und wird in Operationen verwendet.
- Beispielhafter Aufbau einer Nachricht:



- Nachrichtenkörper kann typisierte Objekte (im Sinne der Objektorientierung) enthalten.

Nachrichtenserialisierung

Beispiel

- Java object serialization überführt Objekt in Bytestrom und zurück (deserialization)
- Header enthält Information über Typ, Layout usw., Körper die Daten
- Java Klasse implementiert Interface `java.io.Serializable`
- Alle Attribute der Klasse müssen selbst `Serializable` sein oder als `transient` markiert sein.
- Operationen `writeObject()`, `readObject()`

Semantik von Nachrichten (2)

Nachrichten mit Dokumentcharakter

- Beispiel: HTML über HTTP
- XML-Dokumente
 - ▶ heute stark favorisiert
 - ▶ Typbeschreibung durch Schema
- Beispiel: SOAP (*Simple Object Access Protocol*)

```
1 <soap-env:Envelope
2   xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
3   soap-env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
6     <soap-env:Body>
7       <tns:getFlaeche xmlns:tns="urn:tns:beispiel">
8         <tns:seite1 xsi:type="xsd:double">8.0</tns:seite1>
9         <tns:seite2 xsi:type="xsd:double">4.0</tns:seite2>
10        </tns:getFlaeche>
11      </soap-env:Body>
12    </soap-env:Envelope>
```

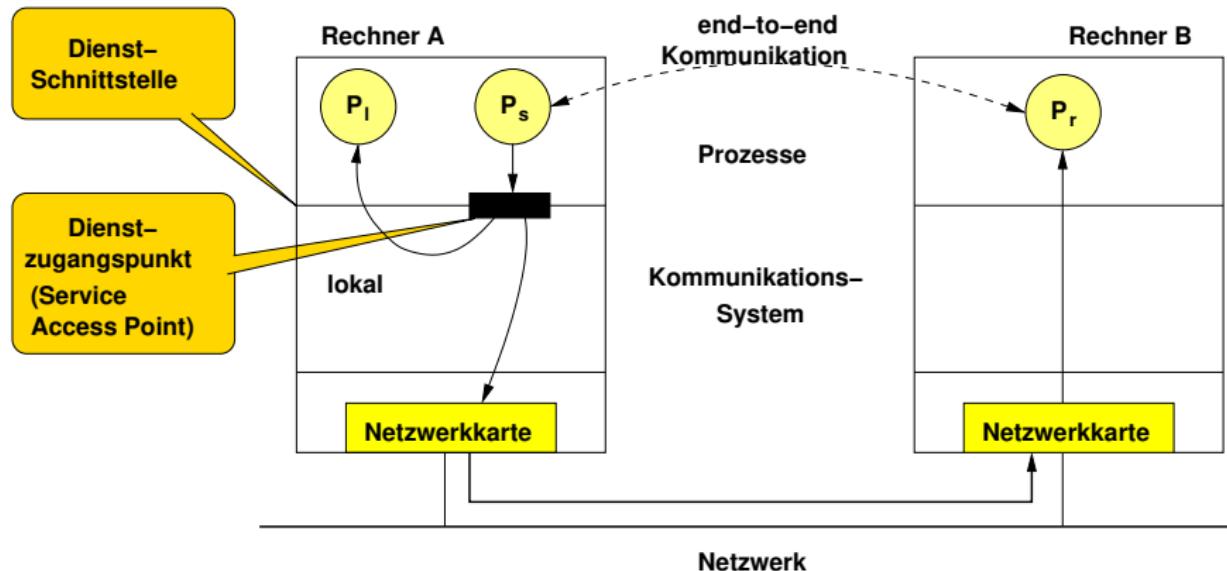
Internet-Protokolle (Wiederholung 2.Sem)

Typische Eigenschaften eines Kommunikationsdienstes

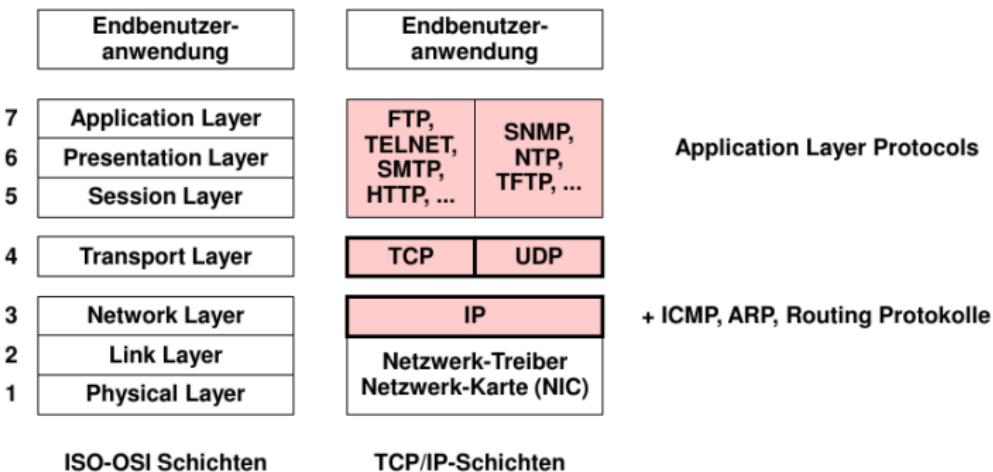
- End-to-End-Kommunikation zwischen Prozessen
- Quality-of-Service (QoS)-Parameter
 - ▶ Merkmale bzgl. der Güte der Diensterbringung, z.B.
 - ★ Leistungseigenschaften (Antwortzeit, Durchsatz)
 - ★ Echtzeiteigenschaften
 - ★ Fehlereigenschaften
- Transparenz bzgl. Netzwerktopologie und Übertragungsverfahren
- Wichtige interne Aspekte:
 - ▶ Fehlerkontrolle
 - ▶ Flusskontrolle
 - ▶ Routing (Wegfindung)

Kommunikationsmodell

Allgemeines Modell der systemweiten Interprozesskommunikation:



Einordnung der Internet-Protokolle



IP:	Internet Protocol	SMTP:	Simple Mail Transfer Protocol
TCP:	Transmission Control Protocol	HTTP:	HyperText Transfer Protocol
UDP:	User Datagram Protocol	SNMP:	Simple Network Management Prot.
FTP:	File Transfer Protocol	NTP:	Network Time Protocol
TELNET:	Remote Terminal Protocol	TFTP:	Trivial File Transfer Protocol

Hauptmerkmale der TCP/IP-Architektur

- Verbindungsloses Protokoll (IP) auf der Netzwerkebene
- Netzketten zur Paketvermittlung
- Statisches und dynamisches Routing
- Transportprotokoll (TCP) mit Ende-zu-Ende-Sicherungsfunktionen

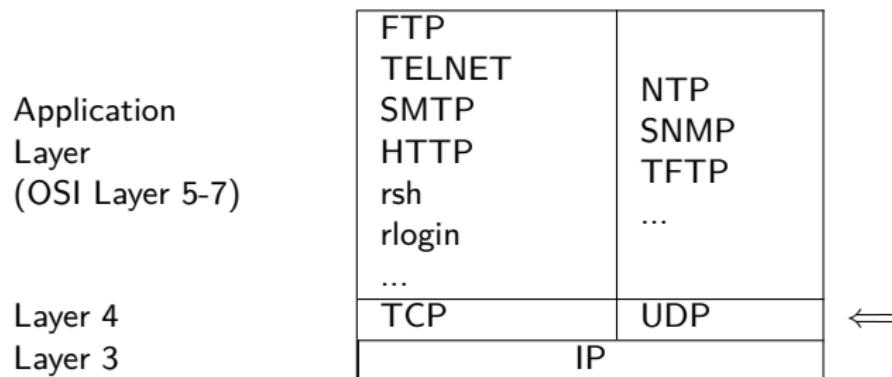
Zusammenfassung IP

Wesentliche Eigenschaften des Internet-Protokolls (IPv4):

- verbindungsloses Protokoll
- „best-effort“-Beförderung von Einzelnachrichten
(Datagram, =Paket)
- Adressierung von Hosts durch 32-bit Internet-Adressen
- Fragmentierung von Paketen bei Bedarf
- Prüfsumme nur für den Kopfteil, nicht über den Dateninhalt
- nicht oft benutzte Protokollfelder sind optional enthalten
- Lebensdauer eines Pakets im Netz begrenzt
- neue Version IPv6

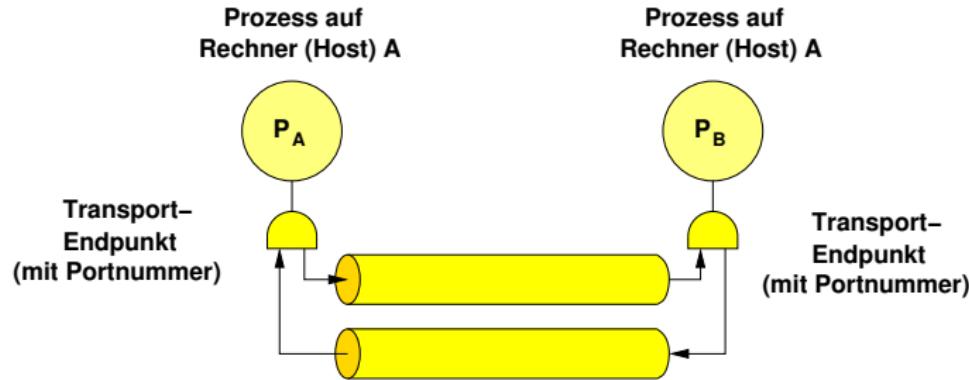
TCP, UDP und die Transportebene

- Aufgabe des Transmission Control Protocols (TCP):
 - ▶ zuverlässiger bidirektionaler Punkt-zu-Punkt-Transport eines Bytestroms zwischen Prozessen in Endsystemen
- Aufgabe des User Datagram Protocols (UDP):
 - ▶ Best-effort Datagram-Dienst der IP-Netzwerkebene wird Prozessen in Endsystemen zugänglich gemacht
- Einordnung:



TCP-Kommunikationsmodell

- verbindungsorientiert
- virtuelle, bidirektionale, voll-duplex-fähige Verbindung zwischen Transport-Endpunkten, die von Prozessen genutzt werden
- Adressierung der Transportdienst-Endpunkte durch 16-bit-Portnummern (in Ergänzung der IP-Adresse des Knotens)
- Bytestrom-orientiert, nicht blockweise. Ein in einem Paket versendeter Abschnitt des Bytestroms wird Segment genannt



Zusammenfassung TCP

Weitere Eigenschaften des TCP-Protokolls:

- Sicherungsfunktion der Übertragung durch
 - ▶ Sequenznummern
 - ▶ Prüfsummenbildung (Algorithmus wie IP)
 - ▶ Empfangsquittungen und Timeouts
 - ▶ Wiederholung nach Timeout
- Sliding Window-Prinzip zur Flusskontrolle
- Urgent Data und Push-Funktion für vorrangige Daten

Zusammenfassung UDP

Wesentliche Eigenschaften des UDP-Protokolls:

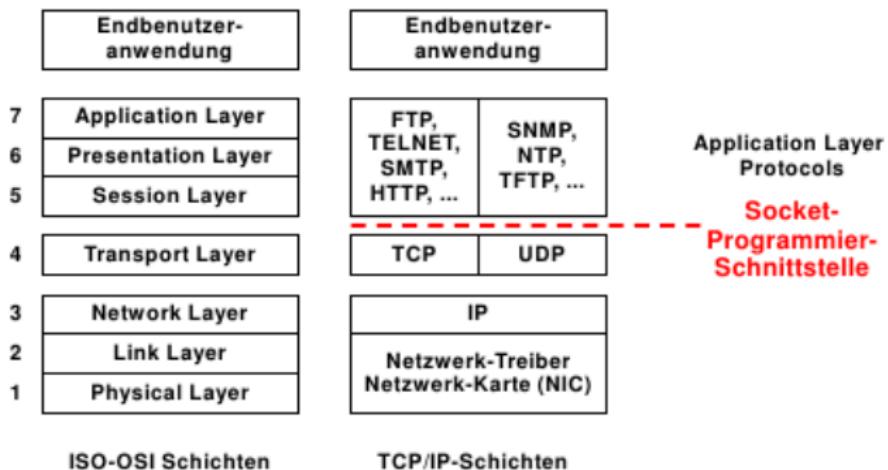
- verbindungsloses Protokoll
- Adressierung der Nutzer durch 16-bit-Portnummern
- „best-effort“-Beförderung von Datagrammen (Einelnachrichten),
i.w. wird der einfache IP-Dienst der Netzwerkebene
Anwendungsprozessen zugänglich gemacht (User Datagrams im
Unterschied zu IP Datagrams, 1:1-Zuordnung)
- Multicast / Broadcast-fähig (1:n-Kommunikation), direkte
Umsetzung bei Multicast-fähigen Netzen wie z.B. Ethernet
- Integritätsprüfung durch Prüfsumme
(reale Durchführung kann von der Implementierung abhängen)
- keinerlei Quittungen oder Sicherungsmechanismen,
d.h. Datagramme desselben Senders können verloren gehen, in
anderer Reihenfolge oder doppelt ankommen
- keine Flusskontrolle, d.h. auf Empfängerseite aufgrund von
Puffermangel nicht annehmbare Datagramme werden vom
Protokollmodul ohne Mitteilung an den Empfänger verworfen
- gut geeignet zur Implementierung einfacher
Auftrag/Antwort-Protokolle

Grundlagen der Socket-Programmierung

Netzwerk-Programmierung mittels Sockets:

- Ziel: Nachrichtenorientierte Interprozesskommunikation zwischen Anwendungsteilen auf verschiedenen Rechnern
- Eingeführt in 4.x BSD UNIX mit C API, heute in nahezu allen Betriebssystemen bereitgestellt (z.B Windows NT, BSe für eingebettete Systeme)
- Heute immer noch die am weitesten verbreitete Schnittstelle für die Entwicklung von Netzwerk-Anwendungen auf der Basis von TCP/IP-Protokollen
- Allen höheren Anwendungsprotokollen (z.B. HTTP) unterlagert
- Sockets als Kommunikationsendpunkte
- Unterstützt Client/Server-Beziehung zwischen Programm-Komponenten:
 - ▶ Server: Diensterbringer
 - ▶ Client: Dienstnutzer, Kunde, Klient
- Java Sockets bilden BSD Sockets in Menge von Klassen nach

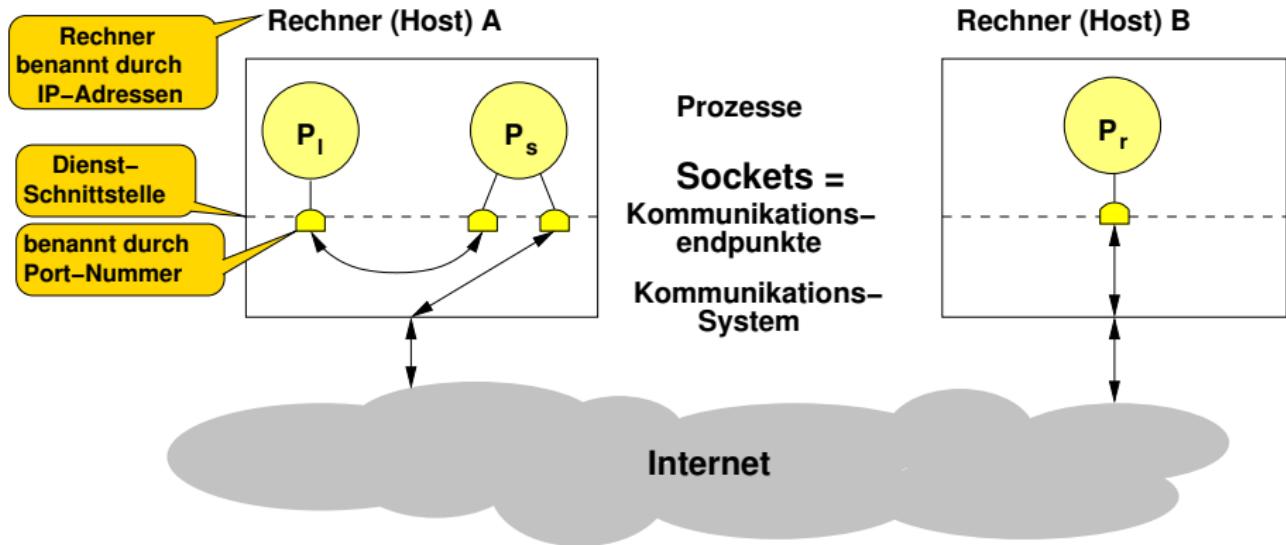
Einordnung



IP:	Internet Protocol	SMTP:	Simple Mail Transfer Protocol
TCP:	Transmission Control Protocol	HTTP:	HyperText Transfer Protocol
UDP:	User Datagram Protocol	SNMP:	Simple Network Management Prot.
FTP:	File Transfer Protocol	NTP:	Network Time Protocol
TELNET:	Remote Terminal Protocol	TFTP:	Trivial File Transfer Protocol

Kommunikationsmodell

Socket-basiertes Modell (als Verfeinerung s.o.):



Globaler Kommunikationsendpunkt benannt durch: (Host IP-Adresse, Portnummer)

Internet-Adressen (IPv4)

- Die Repräsentierung einer IP-Adresse geschieht in einem 32-bit-Wort
- In der dezimalen Schreibweise für IP-Adressen (dotted decimal) werden die Inhalte der Bytes einer Adresse in absteigender Wertigkeit, durch Punkte getrennt, als Zeichenkette angegeben:
- Beispiele:
 - ▶ 193.175.36.224
 - ▶ 127.0.0.1 (loopback-Netz von localhost)
- Subnetze: CIDR (Classless Inter-Domain Routing): Feste Zuordnung einer IP-Adresse zu Netzklasse (A,B,C) entfällt. Suffix gibt Anzahl der 1-Bits der Netzmaske an. Bsp: /20 mit Netzmaske 255.255.240.0
- Bzgl. IPv6 und der leicht modifizierten Netzwerkprogrammierung sei z.B. verwiesen auf „Beej's Guide to Network Programming“
<http://beej.us/guide/bgnet/>

Byte Ordering

Für die Datendarstellung von Unsigned Integers im Speicher werden unterschieden:

- big-endian Rechner (höherwertige Stelle in Byte mit niederer Adresse)
Beispiele: Sun SPARC, Motorola, IBM Mainframe
- little-endian Rechner (umgekehrt)
Beispiele: Intel x86, DEC VAX

Beispiel: IP-Adresse 193.175.36.224

So ...	0	1	2	3	„Big endian“
	193	175	36	224	
...oder so	224	36	175	193	„Little endian“

Netz-Datendarstellung der TCP/IP-Protokoll-Familie ist big-endian

Port-Nummern

Regeln für die Benutzung von Port-Nummern:

- Portnummern werden von der IANA (Internet Assigned Numbers Authority) vergeben.
- Die Verwaltungsprozedur ist beschrieben in RFC 6335
 - ▶ 0–1023: reservierte System-Ports, well-known Port-Nummern der Standarddienste
 - ▶ 1024–49151: registrierte oder User-Ports, können von der IANA zugewiesen werden
 - ▶ 49152–65535: private oder Ephemeral Port-Nummern von benutzerdefinierten Diensten

Beispiele:

- | | |
|------------|------------|
| ● 22: SSH | ● 53: DNS |
| ● 25: SMTP | ● 80: HTTP |

Zuordnungen werden in Datei */etc/services* angegeben

Netzwerk-Verbindungen

Viele gleichzeitige Verbindungen zwischen Rechnern möglich.

In TCP/IP eindeutig identifizierbar durch:

- Transportschicht-Protokol
- IP-Adresse des lokalen Hosts
- lokale Portnummer
- IP-Adresse des entfernten Hosts
- entfernte Portnummer

Arten von Sockets

Stream Sockets: (SOCK_STREAM)

- Verlässliche Kommunikation (i.d.R. eines Byte-Stroms) zwischen zwei Endpunkten
- Verbindungsorientierter Transportdienst
- Im Falle von Internet-domain sockets ist TCP das benutzte Default-Protokoll

Datagram Sockets: (SOCK_DGRAM)

- Unzuverlässige Kommunikation von Einzelnachrichten (best-effort delivery)
- Verbindungsloser Datagram-Dienst
- Im Falle von Internet-domain sockets ist UDP das benutzte Default-Protokoll

Raw Sockets: (SOCK_RAW)

- erlauben Zugriff auf unterlagerte Protokolle, wie IP, ICMP, ...
- hier nicht weiter betrachtet

BSD Sockets

Socket-Programmierung:

- Eingeführt in 4.x BSD UNIX
- API realisiert durch BSD System Calls
- Einbettung in UNIX I/O-Funktionen
- TCP/IP-Protokolle als Default, aber Nutzung anderer Protokolle prinzipiell möglich
- Schnittstelle auch für die Kommunikation von Prozessen auf *einem* UNIX-Rechner nutzbar (UNIX domain sockets)

Socket-Adressen Datentypen

- Include-Dateien:

```
#include <sys/types.h>
#include <sys/socket.h>
```

- Internet-Adresse:

```
struct in_addr { uint32_t s_addr; };
```

- Socket-Adresse (allg. Typ, in System Calls benutzt):

```
struct sockaddr {
    u_short sa_family; /* hier AF_xxxx */
    char    sa_data[14]; /* bis 14B typ-spez. Adresse */
};
```

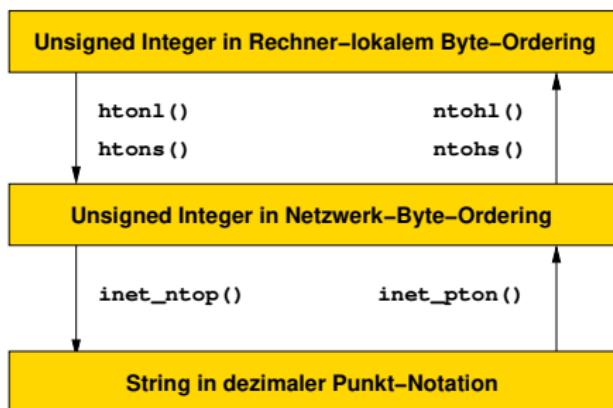
- Socket-Adresse (Internet-Typ):

```
struct sockaddr_in {
    u_short sin_family; /* hier AF_INET, o. AF_UNIX */
    u_short sin_port;   /* Port-Numer in */
    /* network byte order */
    struct in_addr sin_addr; /* IP-Adresse in */
    /* network byte order */
    char sin_zero[8];    /* unbenutzt */
};
```

- Cast:

```
struct sockaddr_in my_addr;
... (struct sockaddr*) &my_addr ...
```

Hilfsfunktionen: Adress-Konvertierung



Funktionen definiert in
<sys/types.h>
<netinet/in.h>

Funktionen definiert in
<sys/types.h>
<netinet/in.h>
<arpa/inet.h>

- | | |
|------------------|-----------------------------------|
| htonl()/htons(): | host to network long/short |
| ntohl()/ntohs(): | network to host long/short |
| inet_ntop(): | network to presentation/printable |
| inet_pton(): | presentation/printable to network |

Hilfsfunktion: Adress-Übersetzung

getaddrinfo()

```
struct addrinfo {
    int          ai_flags;      // AI_PASSIVE, AI_CANONNAME, etc.
    int          ai_family;     // AF_INET, AF_INET6, AF_UNSPEC
    int          ai_socktype;   // SOCK_STREAM, SOCK_DGRAM
    int          ai_protocol;   // use 0 for "any"
    size_t       ai_addrlen;    // size of ai_addr in bytes
    struct sockaddr *ai_addr;   // struct sockaddr_in or _in6
    char         *ai_canonname; // full canonical hostname
    struct addrinfo *ai_next;   // linked list, next node
};
```

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
int getaddrinfo(
    const char *node,
    const char *service,
    const struct addrinfo *hints, (⇒ ersetzt gethostbyname(), getservbyname())
    struct addrinfo **res);
```

„Given node and service, which identify an Internet host and a service, getaddrinfo() returns one or more addrinfo structures, each of which contains an Internet address that can be specified in a call to bind(2) or connect(2).“

Beispiel

```
int main(int argc, char *argv[])
{
    struct addrinfo hints;
    struct addrinfo *result;
    int s;

    ...
    memset(&hints, 0, sizeof(struct addrinfo));
    hints.ai_family = AF_UNSPEC;      /* Allow IPv4 or IPv6 */
    hints.ai_socktype = SOCK_DGRAM;   /* Datagram socket */
    hints.ai_flags = AI_PASSIVE;     /* For wildcard IP address */
    hints.ai_protocol = 0;           /* Any protocol */
    hints.ai_canonname = NULL;
    hints.ai_addr = NULL;
    hints.ai_next = NULL;
    s = getaddrinfo(NULL, argv[1], &hints, &result);
    if (s != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
        exit(EXIT_FAILURE);
    }
}
```

Weitere Hilfsfunktionen

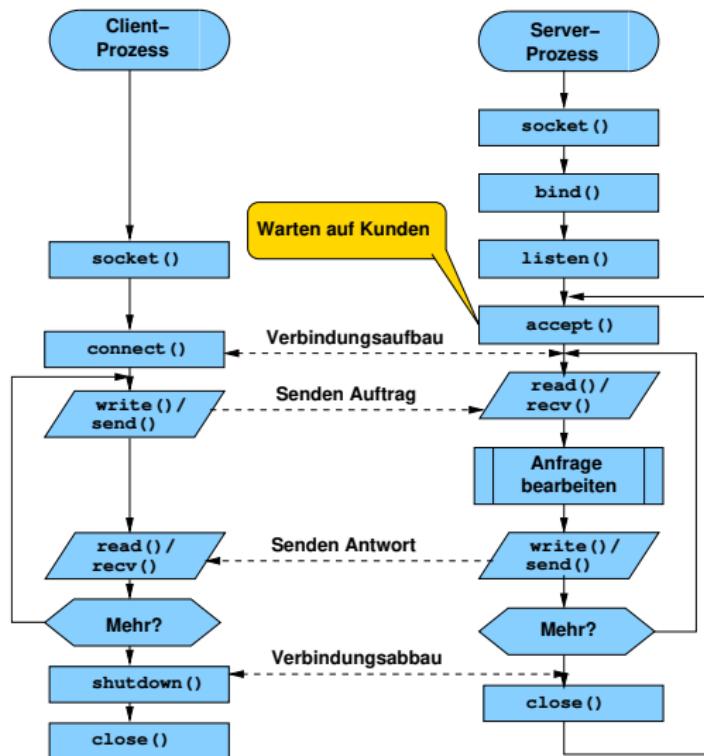
`gethostname()`
`gethostid()`
`getsockopt()`
`setsockopt()`

Ermitteln des eigenen Host-Namens
Ermitteln des eigenen unique Host-IDs
Ermitteln der Parameter eines Sockets
Setzen der Parameter eines Sockets

Socket-Funktionen im Überblick

socket()	Erzeugen
bind()	Zuordnung einer Socket-Adresse / eines -Namens
listen()	Server: Socket vorbereiten auf Akzeptieren von Klienten
accept()	Server: Warten auf Verbindungsanfrage
connect()	Client: Verbindung aufbauen
send() / write()	Senden
recv() / read()	Empfangen
shutdown()	Verbindung schließen
close()	Socket zerstören
sendto() / recvfrom()	Senden/Empfangen UDP
select()	Warten auf das Eintreffen eines von mehreren I/O-Ereignissen

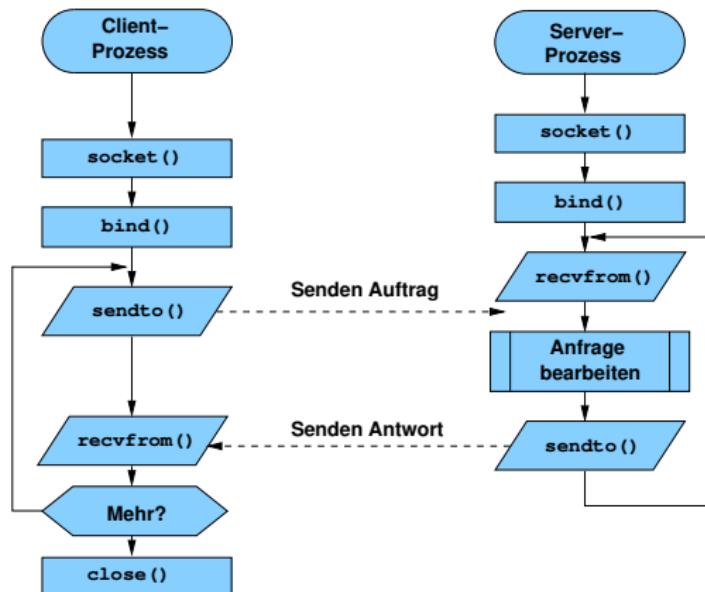
Vereinfachtes Zusammenspiel für TCP



Verallgemeinerung:

- Ein Server hält i.a. gleichzeitig mehrere Verbindungen zu Klienten.

Vereinfachtes Zusammenspiel für UDP



socket()

Erzeugen eines Sockets

```
int socket(int family, int type, int protocol)
```

erzeugt einen Socket in der Internet-Domäne (family=AF_INET) oder der UNIX-Domäne (AF_UNIX) vom Typ Stream-Socket (type=SOCK_STREAM), Datagram-Socket (SOCK_DGRAM) oder Raw-Socket (SOCK_RAW) zur Verwendung mit dem Protokoll protocol und liefert einen Deskriptor für den erzeugten Socket.

Für protocol wird i.d.R. der Wert 0 übergeben. Dann wird das Default-Protokoll gewählt, welches in der Internet-Domäne TCP für einen Stream- Socket bzw. UDP für einen Datagram-Socket ist. Es ist noch keine Socket-Adresse zugeordnet, der Socket ist *ungebunden*.

Beispiel:

```
sd1 = socket(AF_INET, SOCK_STREAM, 0)
sd2 = socket(AF_INET, SOCK_DGRAM, 0)
```

bind()

Binden einer Socket-Adresse

```
int bind(int sd, struct sockaddr *addr, int addrlen)
```

bindet den in der struct sockaddr übergebenen, von der Domäne des betrachteten Sockets abhängigen Adresse an den Socket. Im Falle der Internet-Domäne entspricht diese Struktur der struct sockaddr_in, für einen Socket in der UNIX-Domäne wird im Effekt ein Dateiname übergeben. Der Socket wird im Kommunikationssystem (TCP/IP Protokoll-Modul) registriert. Für Klienten in verbindungsorientierter Kommunikation nicht notwendig.

Beispiel:

```
rc = bind(sd, (struct sockaddr *) &my_addr, sizeof(my_addr))
```

listen()

Socket vorbereiten auf Verbindungsanfragen

```
int listen(int sd, int qlength)
```

zeigt dem TCP/IP-Modul an, dass TCP-Verbindungen über den Socket `sd` angenommen werden sollen; `qlength` gibt die maximale Anzahl wartender Verbindungswünsche an, für die ein `accept` aussteht (nicht die Anzahl der insgesamt möglichen Klienten).

Nur für Server-Seite in verbindungsorientierter Kommunikation notwendig.

Beispiel:

```
rc = listen(sd, 5)
```

accept()

Warten auf Verbindungsanfragen

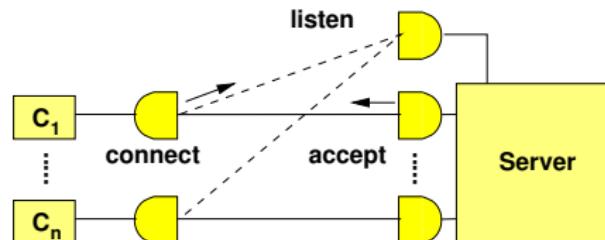
```
int accept(int sd, struct sockaddr *claddr, int *addrulen)
```

blockiert, bis eine Verbindungsanfrage eines Klienten am Socket `sd` vorliegt. Dann wird ein neuer Socket erzeugt, und dessen Deskriptor zurückgegeben. Damit entsteht eine private Verbindung zwischen Klient und Server. Der Socket `sd` steht für weitere Verbindungsanforderungen zur Verfügung. Die Identität des Klienten (entfernte Socket-Adresse) wird in der übergebenen Struktur `claddr` abgelegt und die Längen-Variablen `addrulen` entsprechend gesetzt.

Nur für Server-Seite in verbindungsorientierter Kommunikation notwendig.

Beispiel:

```
snew = accept(sd, &clientaddr, &clientaddrlen)
```



connect()

Verbindungsanfrage

```
int connect(int sd, struct sockaddr *saddr, int saddrlen)
```

Aktive Verbindungsanfrage eines Klienten über seinen Socket `sd` zu einem Server, dessen Adresse in `saddr` mit der Länge `saddrlen` übergeben wird.

Nur für Client-Seite in verbindungsorientierter Kommunikation notwendig.

Beispiel:

```
rc = connect(sd, &saddr, sizeof(saddr))
```

write()/send() und read()/recv()

Senden

```
int write(int sd, char *buf, int len)  
int send(int sd, char *buf, int len, int flag)
```

Der `write`-Aufruf wird wie bei Datei-Deskriptoren verwendet. Der `send`-Aufruf besitzt ein Flag als zusätzliches Argument zur Spezifikation spezieller Optionen.

Empfangen

```
int read(int sd, char *buf, int nbytes)  
int recv(int sd, char *buf, int nbytes, int flag)
```

Der `read`-Aufruf wird wie bei Datei-Deskriptoren verwendet. Der `recv`-Aufruf besitzt ein Flag als zusätzliches Argument zur Spezifikation spezieller Optionen.

Beispiele:

```
charcount = write(sd, buf, len)  
charcount = send(sd, buf, len, sendflag)  
charcount = read(sd, buf, len)  
charcount = recv(sd, buf, len, recvflag)
```

shutdown()

Schließen einer Verbindung

```
int shutdown(int sd, int how)
```

Geordnetes Schließen einer Verbindung, `how` gibt an, ob sich das TCP/IP-Modul auch nach dem Schließen noch um die Verbindung kümmern soll.

Der Socket-Deskriptor bleibt bestehen und muss, falls gewünscht, mittels `close()` zerstört werden.

Beispiel:

```
rc = shutdown(sd, 2)
```

select()

Warten auf das Eintreffen eines von mehreren I/O-Ereignissen

```
#include <sys/time.h>
int select(int nfds, int *readmask, int *writemask,
           int *exceptmask, struct timeval *timeout)
```

bietet die Möglichkeit zum Umgang mit mehreren Socket/File-Deskriptoren in einem Prozess, um so lange zu warten, bis an einem Deskriptor einer vorgebbaren Menge ein Ereignis eintritt (z.B. Socket wird lesbar) oder ein Timeout abgelaufen ist. Die Wartezeit kann zeitlich begrenzt werden oder unbefristet sein.

Die Mengen werden über Bitmasken spezifiziert, wozu z.B. die FD_SET- Makros mit Vorteil eingesetzt werden können.

Bei Rückkehr ist `readmask` verändert und enthält die Bitmaske der Deskriptoren, für die Ereignisse eingetreten sind, der Rückgabewert gibt die Anzahl dieser Deskriptoren an.

Beispiel:

```
int sd1, sd2;                      FD_ZERO(&fds);
fd_set fds;                         FD_SET(sd1,&fds);
sd1 = socket(AF_INET,...);          FD_SET(sd2,&fds);
sd2 = socket(AF_INET,...);          rc=select(FD_SETSIZE,&fds,
...                                NULL,NULL,timeout);
```

Java Sockets

Verkleidung der BSD Sockets hinter mehreren Interfaces/Klassen des Package `java.net`

Adressierung

- InetAddress mit Subklassen Inet4Address und Inet6Address
- SocketAddress mit Subklasse InetSocketAddress

TCP-Verbindungen

- ServerSocket
- Socket
- Für etablierte Verbindung getInputStream()/getOutputStream()

Datagram-Kommunikation über UDP

- DatagramPacket
- DatagramSocket – MulticastSocket

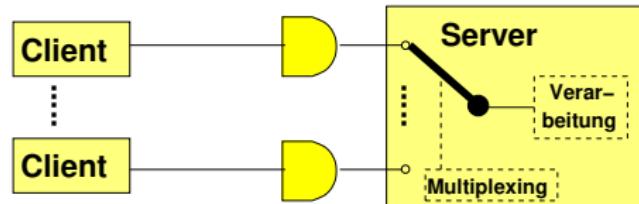
Vertiefung im Praktikum

Server-Architektur

- **Architektur-Prinzipien für die interne Struktur von Server-Prozessen**

- **Problem:**

Server muss i.d.R. mit mehreren Klienten kommunizieren können.



Modelle

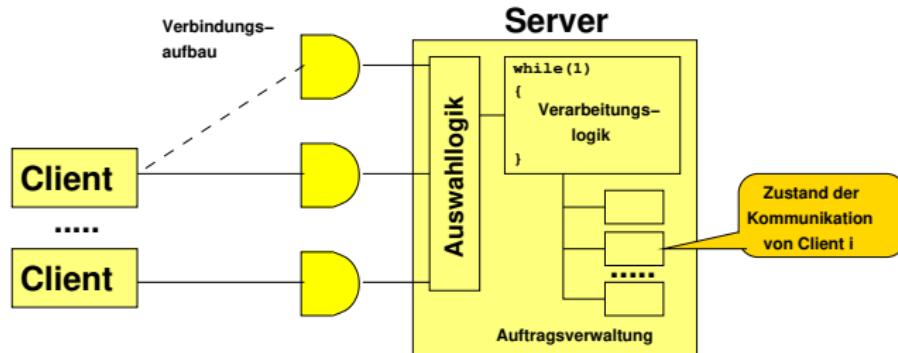
- Einfacher sequentieller Prozess
- Klassischer sequentieller Server als Zustandsautomat
- Parallelle Server-Prozesse
- Multithreaded Server

Einfacher sequentieller Prozess

- Ein Prozess bearbeitet nacheinander die Anfragen aller Klienten
- Problem, wenn Server während der Bearbeitung selbst zum Client gegenüber einem weiteren Server wird:
⇒ der gesamte Server ist blockiert!
- Nachteile:
 - ▶ keine Nebenläufigkeit im Server
 - ▶ keine Nutzung einer unterlagerten Multiprozessor-Architektur durch einen einzelnen Server möglich.
- Ansatz ist in kommerzieller Umgebung selten akzeptabel!

Klassischer sequentieller Server

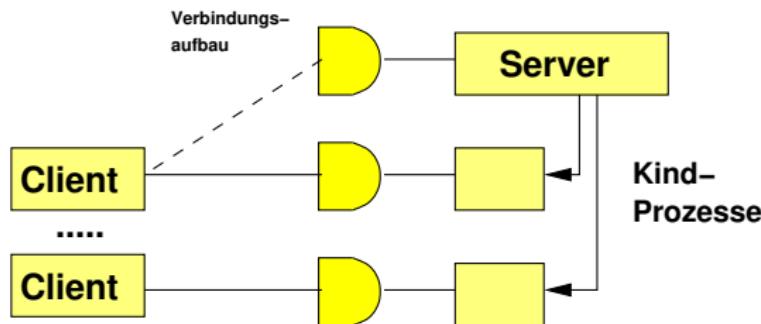
Architektur: Server als Zustandsautomat



- keine interne Blockierung:
mehrere Aufträge können überlappend ausgeführt werden
- Multiplexing „von Hand“ ⇒ komplex zu programmieren
- Auswahllogik in UNIX:
 - ▶ nicht-blockierende Aufrufe (Option `O_NDELAY`) und Polling
 - ▶ `select()`

Parallele Server-Prozesse

Architektur:



- Kindprozesse enthalten Gedächtnis für den Zustand der Kommunikation mit jeweils einem Client
- Vorteil: Multiprozessor-Architektur kann genutzt werden
- aufwändiges Prozess-Handling aus Performance-Sicht

Multithreaded Server

- **Automatisches Lösen des Multiplexing-Problems**

- ▶ jedem Auftrag wird bei Beginn der Verarbeitung ein Thread fest zugeordnet
- ▶ jeder einzelne Thread kann innerhalb des Servers blockieren, aber Nebenläufigkeit bleibt insgesamt erhalten.
- ▶ Thread Pool

- **Anwendbar für alle Paradigmen verteilter Anwendungen
(vgl. Kap. 3)**

- **Synchronisation erforderlich**

Multithreaded Server (2)

- **Threads sind mittlerweile in allen neueren Betriebssystemen und Runtime-Systemen verfügbar**
- **Implementierung der User-Level Threads durch Kernel Threads oder in Thread Libraries möglich (vgl. LV Betriebssysteme)**
- **Verbreitete Programmierschnittstellen**
 - ▶ Pthreads POSIX 1003.4 (C/C++)
 - ▶ Boost.Threads (C++)
 - ▶ Java Concurrency ab SE 5: java.util.concurrent
- **Wird im Praktikum vertieft**

Zusammenfassung

- Netzwerkprogrammierung ist grundlegend für die Umsetzung verteilter Systeme.
- Quasi jedwede Netzwerkprogrammierung basiert letztlich auf dem Konzept der Socket-Schnittstellen.
- BSD-Sockets stellen die ursprüngliche und bis heute wichtigste Socket-API dar.

3. Remote Procedure Calls



http://walker.countynewstoday.com/wp-content/uploads/2015/03/logo_marshall.jpg

Inhalt

3. Remote Procedure Calls

- 3.1 Einführung und Motivation
- 3.2 Grundprinzip
- 3.3 Binding
- 3.4 Behandlung von Parametern
- 3.5 Semantik im Fehlerfall
- 3.6 RPC-Protokoll

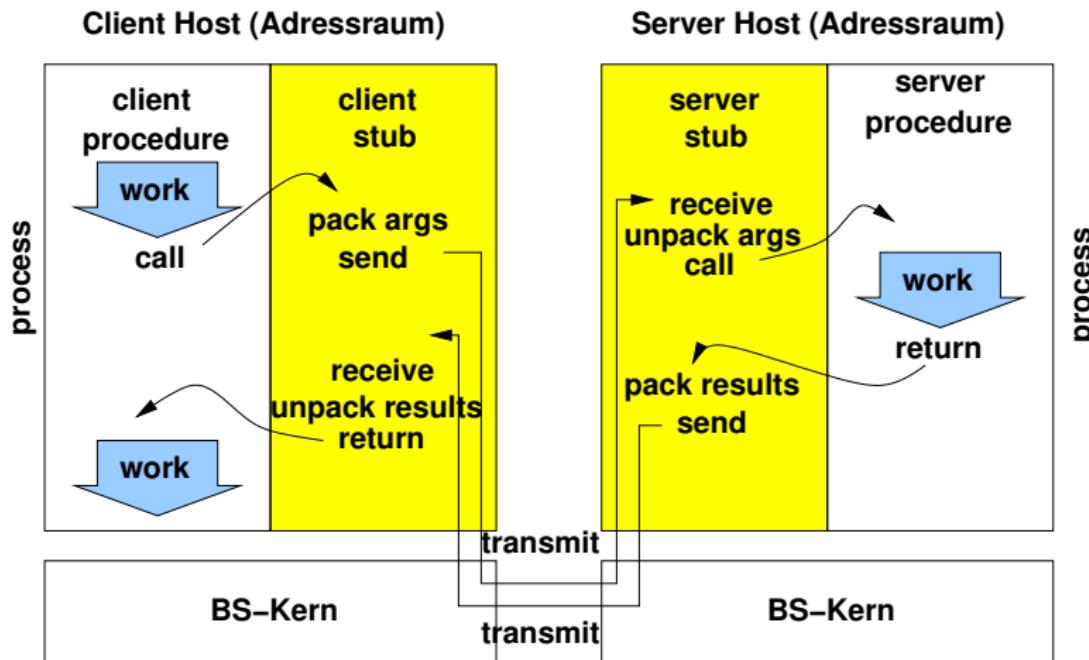
Einführung und Motivation

- Nachrichtenorientierte Kommunikation
 - ▶ asynchroner Nachrichtenaustausch
 - ▶ explizit mit send()/receive()-Operationen
 - ▶ Fazit:
 - + sehr flexibel, alle Kommunikationsmuster implementierbar
 - explizit, I/O-Paradigma
- Ziel des Remote Procedure Call (RPC)
 - ▶ deutsch: Fernaufruf (wenig verbreitet)
 - ▶ Transparenz der Kommunikation
 - ▶ Erscheinungsbild wie üblicher lokaler Prozederaufruf
- Unterstützung für
 - ▶ Dienstorientierung: Dienst = Service = Menge von Funktionen
 - ▶ RPC für Funktionsaufruf
 - ▶ Objektorientierung: RPC genutzt für Methodenaufrufe

Historie

- Erste umfassende Darstellung:
 - ▶ Dissertation Nelson (1981, XPARC)
 - ▶ abgeleitetes Paper Birrel/Nelson (1984, ACM ToCS)
- Definition:
 - ▶ „RPC (Fernaufruf) ist der synchrone Transfer von Kontrolle und Daten zwischen Teilen eines in verschiedenen Adressräumen ablaufenden Programms“
- Nelson's These:
 - ▶ RPC ist ein leistungsfähiges Konzept zur Konstruktion verteilter Anwendungen
 - ▶ RPC vereinfacht die Programmierung verteilter Systeme
- Heute:
 - ▶ Nelson's Sicht allgemein akzeptiert
 - ▶ RPC-Systeme in vielen Produkten
 - ▶ Typ. Beispiele: SunRPC und NFS, OSF DCE RPC, (aktuell) Apache Thrift, D-Bus

Grundprinzip

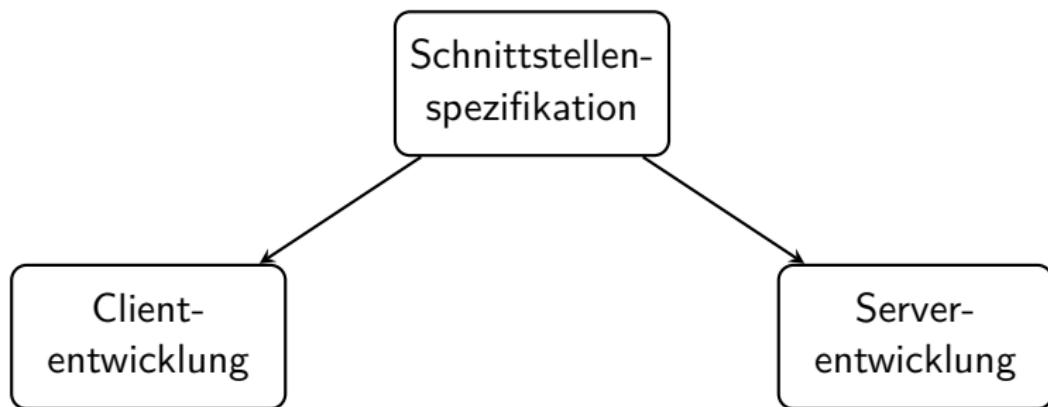


pack/unpack = marshalling/unmarshalling

Stellvertreterkomponenten: stub, proxy, skeleton

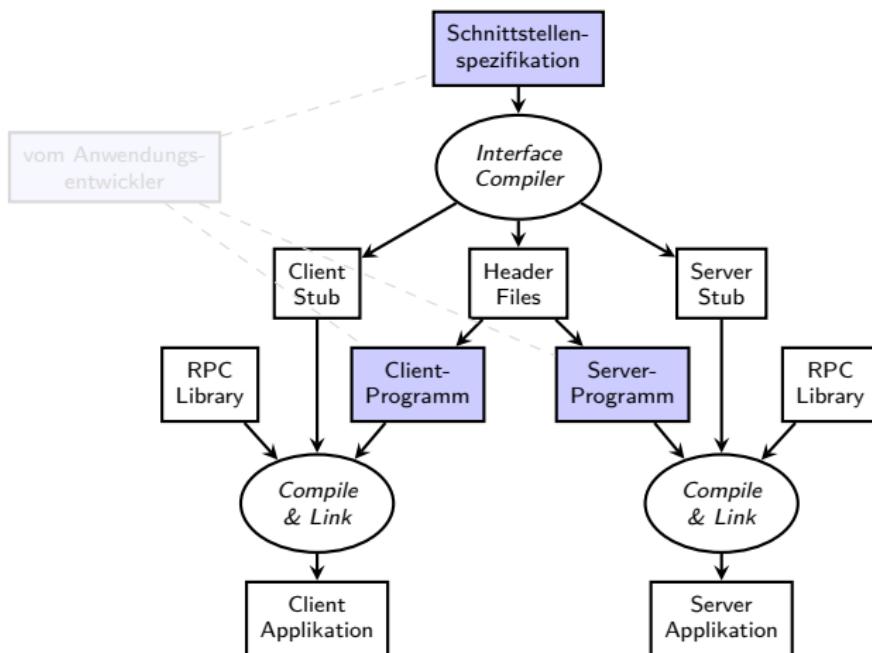
Programmentwicklung

Grobstruktur:



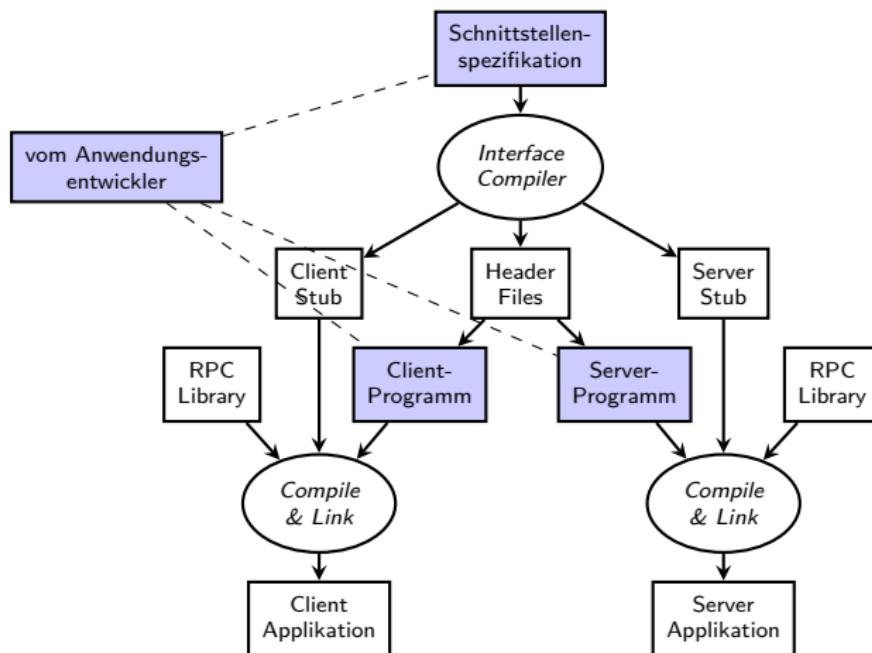
Programmentwicklung (2)

genauer, aber immer noch unabhängig von speziellem RPC-System:

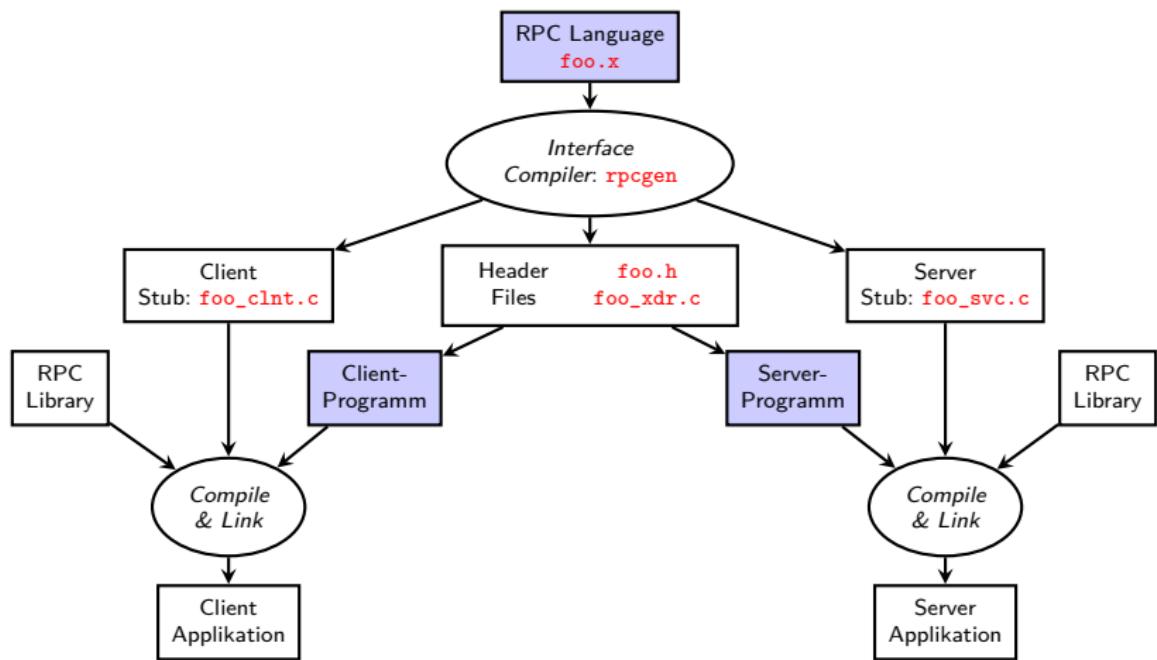


Programmentwicklung (2)

genauer, aber immer noch unabhängig von speziellem RPC-System:



Beispiel: SunRPC



Beispiel Schnittstellenbeschreibung SunRPC (1) *

```
const MAX_FILENAME_LEN = 255;
typedef string t_filename<MAX_FILENAME_LEN>;
const MAX_CONTENT_LEN = 255;
typedef string t_content<MAX_CONTENT_LEN>;  
  
struct s_fstat {
    long dev;
    long ino;
    long mode;
    long nlink;
    long uid;
    long gid;
    long rdev;
    long size;
    long blksize;
    long blocks;
    long atime;
    long mtime;
    long ctime;
}  
  
struct s_filewrite {
    t_filename filename;
    t_content content;
};  
struct s_chmod {
    t_filename filename;
    long mods;
};
```

Beispiel Schnittstellenbeschreibung SunRPC (2)

```
program fileservice {
    version fsrv {
        int fsrv_mkdir(string) = 1;
        int fsrv_rmdir(string) = 2;
        int fsrv_chdir(string) = 3;
        int fsrv_writefile(s_filewrite) = 4;
        string fsrv_readfile(string) = 5;
        s_fstat fsrv_fileattr(string) = 6;
        int fsrv_chmod(s_chmod) = 7;
    } = 1;
} = 0x30000001;
```

Beispiel Schnittstellenbeschreibung DCE

```
[ uuid(5ab2e9b4-3d48-11d2-9ea4-80c5140aaa77),
    version(1.0), pointer_default(ptr)
]
interface echo {
    typedef [ptr, string] char * string_t;
    typedef struct {
        unsigned32 argc;
        [size_is(argc)] string_t argv[];
    } args;
    boolean ReverseIt(
        [in] handle_t h,
        [in] args* in_text,
        [out] args** out_text,
        [out,ref] error_status_t* status
    );
}
```

Beispiel Schnittstellenbeschreibung Thrift

```
typedef i32 MyInteger
enum Operation { ADD = 1,
                 SUBTRACT = 2,
                 MULTIPLY = 3,
                 DIVIDE = 4
}
struct Work {
    1: MyInteger num1 = 0,
    2: MyInteger num2,
    3: Operation op,
    4: optional string comment,
}
exception InvalidOperation { 1: i32 what, 2: string why }
service Calculator {
    void ping(),
    i32 add(1:i32 num1, 2:i32 num2),
    i32 calculate(1:i32 logid, 2:Work w)
        throws (1:InvalidOperation ouch),
    oneway void quit()
}
```

Binding/Trading

- Binding/Trading:

- ▶ Problem: Binden eines Clients an einen Server notwendig
- ▶ Problem gilt analog auch für andere Paradigmen
- ▶ Aspekte: Naming & Locating

⇒ Naming

- ▶ Wie benennt der Client, an was er gebunden werden will (Service)
- ▶ Interface-Name allgemein aus systemweitem Namensraum
- ▶ Trading als Verallgemeinerung: zusätzlich Interface-Attribute
- ▶ vgl. Kap. 5: Allg. Namensdienste

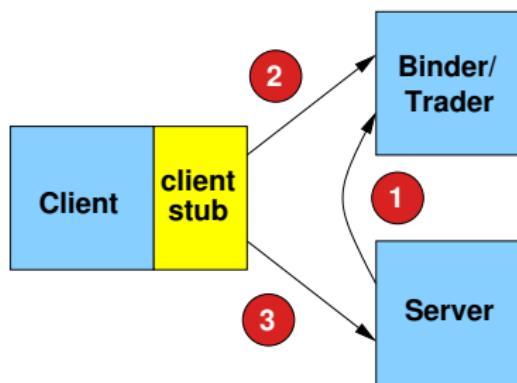
⇒ Locating

- ▶ Bestimmen der (ortsabhängigen) Adresse eines Servers, der das gewünschte Interface exportiert und zur Diensterbringung benutzt wird
- ▶ typisch: (IP-Adresse des Hosts, Portnummer).

Locating-Alternativen

- Adresse statisch im Anwendungsprogramm
 - ▶ kein Suchvorgang erforderlich
 - ▶ i.d.R. nicht flexibel genug
 - ⇒ zu frühes Binden
- Suchen nach Exporteuren zur Laufzeit, z.B. durch Broadcast
 - ▶ hoher Laufzeitaufwand
 - ▶ Broadcast über Netze hinweg problematisch
 - ⇒ i.d.R. zu spätes Binden
- Verwaltung von Zuordnungsinformation durch zwischengeschaltete Instanz
 - ▶ vermittelnde Instanz wird Binder, Trader oder Broker genannt
 - ▶ Exporteur lässt angebotenes Interface (mit allen Attributen) registrieren
 - ▶ Bindeforderung eines Importeurs bewirkt Zuordnung durch Binder/Trader

Prinzipielle Vorgehensweise



① Exportieren des Interface

- ▶ Registrieren eines Interfaces bei Binder
- ▶ Binder hat bekannte Adresse

② Importieren

- ▶ bei erster Inanspruchnahme des Dienstes aus stub heraus
- ▶ liefert handle mit Adresse

③ Fernaufruf

- ▶ client stub benutzt Adresse für Aufruf an Server

Binder / Trader

Typische Schnittstelle

Register(Dienstname, Version, Adresse, evtl. Attribute)

Deregister(Dienstname, Version, Adresse)

Lookup(Name, Version, evtl. Attribute) \Rightarrow Adresse

- Vorteile:

- ▶ sehr flexibel
- ▶ kann mehrere gleichartige Server berücksichtigen
- ▶ Basis für Lastausgleich zwischen äquivalenten Servern

- Nachteile:

- ▶ zusätzlicher Aufwand beim Exportieren und Importieren eines Interfaces
- ▶ problematisch bei kurzlebigen Servern und Clients

Beispiel: SunRPC

- Namen
 - ▶ Paare (Programmnummer, Versionsnummer)
- Adressen
 - ▶ Paare (IP-Adresse des Hosts, Portnummer)
- Binder: Portmapper
 - ▶ Abbildung von Namen auf Portnummern
 - ▶ IP-Adresse des Hosts muss bekannt sein,
der dort lokale Portmapper wird befragt
 - ▶ Portmapper ist selbst SunRPC-Dienst (Port 111)

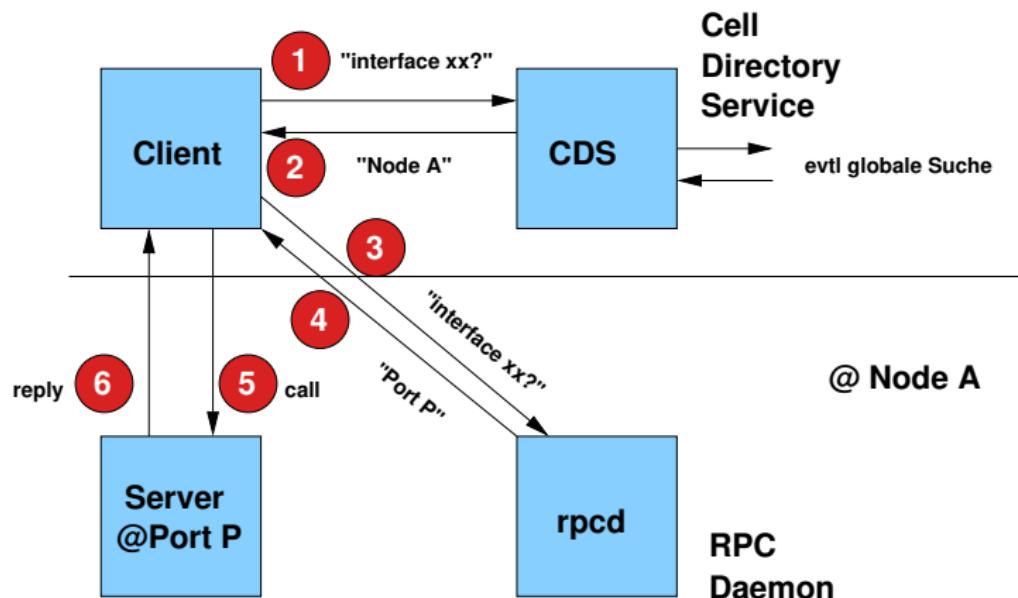
Beispiel: DCE RPC

- Namen
 - ▶ UUID (Universal Unique Identifier)
 - ▶ weltweit eindeutiger String
 - ▶ enthält Netzwerk-Adressinformationen
(z.B. Ethernet MAC-Adresse) und Zeitmarke
 - ▶ generiert durch Tool uuidgen

- Adressen
 - ▶ Paare (IP-Adresse des Hosts, Portnummer)

- Binding
 - ▶ zweistufig innerhalb einer DCE-Zelle
 - ▶ kein zusätzliches Wissen notwendig
 - ▶ Binder heisst RPC Daemon

Beispiel: DCE RPC (2)



Behandlung der Parameterübergabe

- Heterogenitätsproblem
 - ▶ verschiedene Codes (z.B. ASCII - EBCDIC)
 - ▶ Little Endian - Big Endian
 - ▶ unterschiedliche Zahlenformate
- Lösungsmöglichkeiten
 - ▶ Abbildungen zwischen lokalen Datendarstellungen
 - ★ Sender sendet in seiner lokalen Darstellung, Empfänger transformiert
 - ★ erfordert $n \cdot n$ Abbildungen
 - ▶ kanonische Netzdatendarstellung für alle Typen
 - ★ erfordert $2n$ Abbildungen (bei n lokalen Darstellungen)
 - ★ evtl. unnötige Codierung

Verbreitete Netzdatendarstellungen

• XDR (External Data Representation)

- ▶ definiert durch Sun im Rahmen von SunRPC
- ▶ i.w. Motorola 68000 Datenformate: ASCII; Big-Endian, 2-Komplement; IEEE-Gleitpunktzahlen, ...
- ▶ zusammengesetzte Typen: Arrays, Structures, Unions
- ▶ keine explizite Typisierung der Daten, d.h. keine sich selbst beschreibenden Daten
- ▶ für RPC-Systeme sind die Parameter-Typen aber beim Generieren des Stub Codes für beide Seiten bekannt

Beispiel

```
struct {  
    string author<>;  
    int year;  
    string publisher<>;  
}
```

5
Stee
n___
2002
6
Wesl
ey__

}

jeweils 4 Bytes lang)

Verbreitete Netzdatendarstellungen (2)

- ASN.1 BER (ISO Abstract Syntax Notation Number 1, Basic Encoding Rules, ISO 8824, 8825, ITU X.409)

- explizite Typisierung der übertragenen Daten, d.h. allen Datenfeldern geht die Typinformation voraus.
- verbreitet: CANopen, LDAP, UMTS/LTE, VoIP, Encryption
- Standard-Repräsentierung: (Type, Länge, Inhalt)
- Nachteil: laufzeitaufwändig (Bitzugriffe)

Beispiel

Type Identifier:				
7	6	5	4	0
Class	Type	Tag		
0 2	1	Boolean		
0 1	2	Integer, ...		
1 A	16	Sequence		
	Type:	0	Primitive	
		1	Constructed	
	Class:	00	Universal	
		01	Application...	
jeweils 1 Byte (hex)				

Verbreitete Netzdatendarstellungen (3)

• CDR (Common Data Representation)

- ▶ Definition in OMG CORBA 2.0
- ▶ Nutzung im CORBA IIOP-Protokoll
- ▶ Versenden im eigenen Format, „Receiver makes it right“
- ▶ Simple types (short, long, float, char, ...)
- ▶ Complex types (sequence, string, union, struct, ...)
- ▶ Alignment/Padding entsprechend Mehrfachem der Elementlänge
- ▶ Big-endian

Beispiel

struct <string, unsigned long>																
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
5				'S'	'T'	'E'	'E'	'N'					2002			
05	00	00	00	53	54	45	45	4E	00	00	00	00	00	07	D2	
← Länge →									← Padding →							

Verbreitete Netzdatendarstellungen (4)

- JSON (JavaScript Object Notation) Data Interchange Format
 - ▶ Schlankes, textbasiertes Austauschformat
 - ▶ Unabhängig von Programmiersprachen
 - ▶ RFC 7159, abgeleitet von ECMAScript
 - ▶ leicht zu parsen, viele Parser verfügbar
 - ▶ Simple types (string, number, boolean, null)
 - ▶ Complex types (object, array)
 - ★ Objekt ist ungeordnete Liste von Name/Wert-Paaren, wobei Name ein String und Wert ein simple Type, ein Object oder ein Array sein kann
 - ★ Array ist geordnete Folge von Werten

Beispiel

```
{  
    "AUTHOR" : "Steen",  
    "YEAR" : 2002,  
    "PUBLISHER" : "Wesley"  
}
```

Probleme

- komplexe, zusammengesetzte Parametertypen
 - ▶ z.B. structs, arrays, erfordern Regeln zur Serialisierung
- Adressen in Parametern
 - ▶ keine Bedeutung im Zieladressraum !
 - ▶ einfachste Lösung: Verbieten, nur call-by-value zulassen (i.W. SunRPC)
 - ▶ Nutzung eines gemeinsamen globalen Adressraums, falls vorhanden
 - ▶ Ersetzen von Zeigern durch Marker, Rekonstruktion
zusammengesetzter Datenstrukturen auf Empfängerseite durch dort lokale Zeiger (z.B. DCE RPC)

Sicherheit

- Probleme

- ▶ gegenseitige Authentisierung
- ▶ Autorisierung bzgl. ausführbarer Funktionen auf Server-Seite
- ▶ Verschlüsselung der übertragenen Daten

- ausführliche Betrachtung in separatem Kapitel

Semantik des RPC im Fehlerfall

● Fehler-Problematik

- ▶ lokaler Funktionsaufruf:
Rufer und Gerufener werden gleichzeitig abgebrochen
- ▶ RPC:
Ausfall einzelner Komponenten in verteilter Umgebung möglich
- ▶ Zusätzlich Fehlerfälle des Nachrichtensystems berücksichtigen
 - ★ Nachrichtenverlust
 - ★ unbekannte Übertragungszeiten
 - ★ Out-of-order-Ankunft von Nachrichten (Überholen)

Semantik des RPC im Fehlerfall (2)

● *at-least-once*-Semantik

- ▶ erfolgreiche Ausführung des RPC
 - ⇒ aufgerufene Prozedur mindestens einmal ausgeführt, d.h. Mehrfachaufruf kann passieren
 - ▶ beliebiger Effekt im Fehlerfall möglich
 - ▶ i.a. nur für idempotente Operationen geeignet, d.h. mehrfacher Aufruf ändert nicht Zustand und Ergebnis

● Realisierung

- ▶ einfachste Form
- ▶ kommt innerhalb eines Timeouts kein Ergebnis auf Client-Seite an, wird Aufruf vom Stub wiederholt
- ▶ keine Vorkehrungen auf Server-Seite

Semantik des RPC im Fehlerfall (3)

- *at-most-once-Semantik*

- ▶ erfolgreiche Ausführung des RPC
 - ⇒ aufgerufene Prozedur genau einmal ausgeführt
- ▶ nicht-erfolgreiche Ausführung des RPC
 - ⇒ aufgerufene Prozedur erscheint als niemals ausgeführt
- ▶ es können keine partiellen Fehlerauswirkungen zurückbleiben

- Realisierung

- ▶ komplexer
- ▶ Duplikaterkennung erforderlich

Semantik des RPC im Fehlerfall (4)

- *exactly-once-Semantik*
 - ▶ erfolgreiche Ausführung des RPC
⇒ aufgerufene Prozedur genau einmal ausgeführt
 - ▶ nicht-erfolgreiche Ausführung des RPC
⇒ aufgerufene Prozedur erscheint als niemals ausgeführt
 - ▶ entspricht im Normalfall lokalem Funktionsaufruf
- Realisierung
 - ▶ sehr komplex (unmöglich?)

Orphan-Problem

- Orphan = Waise
- Problem: Client stirbt nach Absetzen des RPC
- erzeugter RPC kann weitere Aktivität nach sich ziehen, obwohl niemand darauf wartet
- nach Restart Eintreffen von Antworten aus „früherem Leben“
- Lösungsansätze:
 - ▶ *Extermination: gezielter Abbruch verwaister RPCs basierend auf stabilem Speicher (praktisch unbrauchbar)*
 - ▶ (Gentle) Reincarnation: Einführung von Epochen auf Client-Seite
 - ▶ *Expiration: RPCs werden mit Timeout versehen*

RPC-Protokoll

- RPC-Protokoll: Regeln zur Abwicklung von RPCs
- abhängig von unterlagertem Transportdienst
 - ▶ Datagrammdienst (z. B. UDP)
 - + resourcenschonend, niedrige Latenz
 - Duplikate (durch Timeouts), Vertauschungen und Verlust sind möglich
 - ▶ zuverlässiger Transportdienst (z. B. TCP)
 - + weniger Fehlerfälle auf den höheren Schichten
 - ggf. leistungsmindernd
- ⇒ die Auswahl erfolgt je nach Dienstanforderung

Beispiel: SunRPC

- auch: Open Network Computing (ONC) RPC
- C-Spracheinbettung
- Unterlagerter Transportdienst
 - ▶ TCP oder UDP
 - ▶ RPC fügt keine die Zuverlässigkeit steigernde Maßnahmen hinzu
 - ⇒ UDP und timeouts auf Applikationsebene führen zu „*at-least-once*“-Semantik
 - ⇒ TCP und message transaction ids auf Applikationsebene führen zu „*at-most-once*“-Semantik
- Binding durch Portmapper
 - ▶ Portmapper-Protokoll ist selbst RPC-basiert
- Parameter
 - ▶ i.W. nur call-by-value
- Sicherheit
 - ▶ Authentifizierung: Null, UNIX, DES

Beispiel: SunRPC

- auch: Open Network Computing (ONC) RPC
- C-Spracheinbettung
- Unterlagerter Transportdienst
 - ▶ TCP oder UDP
 - ▶ RPC fügt keine die Zuverlässigkeit steigernde Maßnahmen hinzu
 - ⇒ UDP und timeouts auf Applikationsebene führen zu „*at-least-once*“-Semantik
 - ⇒ TCP und message transaction ids auf Applikationsebene führen zu „*at-most-once*“-Semantik
- Binding durch Portmapper
 - ▶ Portmapper-Protokoll ist selbst RPC-basiert
- Parameter
 - ▶ i.W. nur call-by-value
- Sicherheit
 - ▶ Authentifizierung: Null, UNIX, DES, RPCSEC_GSS

OSF DCE/RPC

- Teil des OSF Distributed Computing Environments
- Grundlage für Microsofts DCOM und ActiveX
- C/C++-Spracheinbettung
- verschiedene Semantiken wählbar mit „*at-most-once*“ als default
- beliebige Parameter-Typen,
„lange“ Parameter über „Pipe“-Mechanismus
- Sicherheit basierend auf Kerberos-Framework
- Bedeutung stark gesunken

Aktuelles RPC-System: Apache Thrift

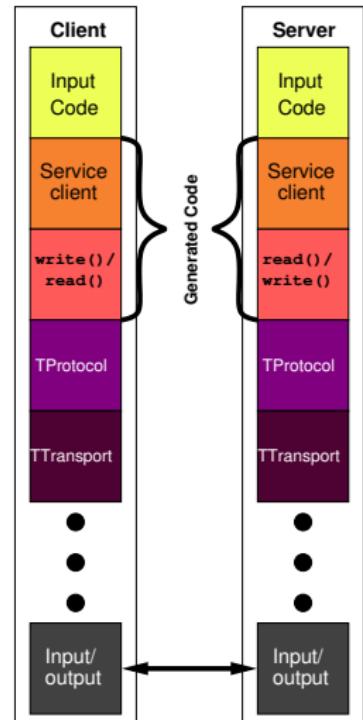
- Apache Thrift Projekt

(<http://thrift.apache.org/>)

- Ursprung Facebook, veröffentlicht 2007
- Unterstützung für alle gängigen Programmiersprachen
- Einfache Thrift-IDL (vgl. Folie 3-13)
- IDL Compiler generiert Client- und Server-Hüllen
- Verschiedene Server-Architekturen nutzbar:
TNonBlockingServer, TThreadedServer,
TThreadPoolServer, TForkingServer, ...
- Verschiedene Protokolle und Transports konfigurierbar
- Protokolle: binäre und textbasierte (u.a. JSON) ⇒ geringer Overhead
- Transports: Tsocket, TMemoryTransport, ...

- Bekannte Nutzer

- Facebook, last.fm, Pinterest, Uber, NSA



Zusammenfassung

- Remote Procedure Calls bieten die Möglichkeit, Funktionen so auf einem entfernten Rechner aufzurufen als würde dies lokal geschehen.
- Wichtige Elemente eines RPC-Systems sind die Schnittstellenbeschreibungssprache (IDL) und deren Compiler, der Binder sowie das Netzdatendarstellungsformat.
- Es existieren verschiedene Fehlersemantiken, die über- oder unterhalb des RPC-Protokolls behandelt werden können.

4. Anwendungsarchitektur



<http://www.mappenvorbereitungskurs.de/berufsbilder/architektur.html>

Inhalt

4. Anwendungsarchitektur

4.1 Einführung

4.2 Middleware-basierte Architekturen

- Nachrichtenorientierung
- Dienstorientierung
- Objektorientierung
- Komponentenorientierung
- Service-Orientierung

4.3 Grundlegende Architekturmodelle

- Client/Server-Modell
- P2P-Modell
- Multi-Tier-Modell
- SOA-Modell

Haupttreiber kommerzieller IT-Produkte

- Hohe Anpassungsfähigkeit

- ▶ Flexibles Abbilden heutiger und künftiger Geschäftsprozesse
- ▶ Verringerung der Entwicklungszeit (time-to-market)
- ▶ Integration existierender (Teil)-Lösungen
- ▶ Interoperabilität mit Fremdsystemen
- ▶ Berücksichtigung der aktuellen technologischen Trends:
 - ★ Internet of Things
 - ★ Cloud Computing
 - ★ Big Data

- Niedrige Kosten

- ▶ Verringerung der Entstehungs-/Entwicklungskosten
- ▶ Verringerung der Betriebs- und Management-Kosten
(total cost of ownership)

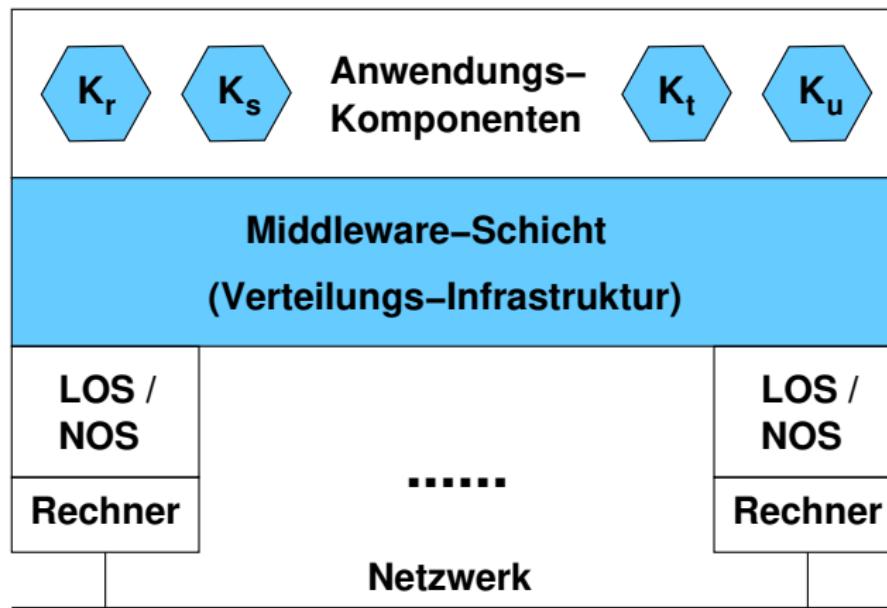
Lösungsansätze

- Offene Systeme (Hersteller-Unabhängigkeit)
- Standardlösungen statt Individual-Entwicklung
- Client/Server, Distributed Computing
- Middleware
- Web-Anwendungen
- Application Server
- Wiederverwendung von Software
(software reuse, componentware)
- Wiederverwendung von Diensten/
Service-Orientierte Architekturen

Middleware-basierte Anw.-Architekturen

- Aufgabe der Middleware:

Standard-Software-Schicht als Verteilungsplattform zur Integration von Programm-Komponenten



Middleware-Architekturen

Jede Middleware ist charakterisiert durch ein Architektur-Paradigma mit Strukturmodell und Ablaufmodell

- Strukturmodell definiert
 - ▶ die verteilbaren Einheiten (Programmkomponenten)
 - ▶ ihre Benennung und Adressierung
 - ▶ eventuelle Hilfskomponenten
- Aktivitätsmodell (Dynamik) definiert
 - ▶ Akteure
 - ▶ Interaktionsmuster
 - ▶ kommunizierte Einheiten
 - ▶ Synchronisation
- Implementierung der Middleware erfordert Rückgriff auf Komponenten der untergelagerten Schichten (speziell NOS/LOS)

Middleware-Architekturen (2)

- Grad der Spezialisierung: kann sehr unterschiedlich sein, z.B.
 - ▶ Unterstützung eines allgemeingültigen Kooperationsansatzes (hier im Vordergrund)
 - ▶ Datenbank-zentriert (SQL-Middleware, Transaktionsmonitore)
 - ▶ Dokumenten/Workflow-orientiert
- Abhängigkeiten von Programmiersprachen
 - ▶ manchmal sehr hoch (z.B. nur mit Java nutzbar)
- Abhängigkeiten von unterlagerten Betriebssystemen
 - ▶ oft weniger stark
- Abhängigkeiten von unterlagerter Hardware
 - ▶ i.d.R. sehr gering

Historische Entwicklung

- Nachrichtenorientierung
 - Dienstorientierung
 - Objektorientierung
 - Komponenten-Orientierung
 - Service-Orientierung (Dokumenten-Orientierung)
- werden im folgenden überblicksweise besprochen
ausgewählte Ansätze in folgenden Kapiteln detaillierter

Paradigma: Nachrichtenorientierung

- Grundmodell kommunizierender Prozesse klassischer Betriebssysteme übertragen auf verteilte Systemumgebung
 - ▶ Prozesse als verteilbare Einheiten
 - ▶ Nachrichten als kommunizierte Einheiten
- Programmierung paralleler Anwendungen
(hier nicht behandelt)
 - ▶ Programmierung von Parallelrechner-Anwendungen
 - ▶ Basis bilden spezielle nachrichtenorientierte Library-Schnittstellen
 - ▶ Message Passing Interface (MPI) (Quasi-Standard)

Paradigma: Nachrichtenorientierung (2)



- Beispiel: Socket-Programmierung (vgl. Kap.2)
 - ▶ Berkeley Sockets (UNIX)
 - ▶ Winsock (MS Windows sockets API)
 - ★ Library mit i.w. Übernahme der UNIX/BSD-Funktionen
 - ▶ Transport Layer Interface (TLI)
 - ★ API für Netzwerkprogrammierung auf Transport-Ebene: `t_xxx`
 - ★ In UNIX SVR4 auf STREAMS implementiert.
 - ★ Kaum noch relevant und nicht mehr empfohlen.
 - ▶ Sockets heute noch de-facto-Standard, z.T. über Libraries oder Klassen verkleidet
 - ▶ Java Sockets (`java.net`)
entspricht weitgehend Modell der Berkeley Sockets.

Paradigma: Nachrichtenorientierung (3)

- Message-oriented Middleware (MOM)
 - ▶ häufig Unterstützung für Persistenz, Transaktionen
 - ▶ Beispiele:
 - ★ IBM Websphere MQ
 - ★ Java Messaging Service (JMS) (Teil von J2EE)
 - ★ RabbitMQ

Paradigma: Dienstorientierung

Basis: Remote Procedure Call (RPC)

- Dienste als verteilbare Einheiten
- Dienst = Service: Menge von offerierten Operationen/Funktionen
- Nutzung entfernter Dienste durch Prozeduraufrufe
- i.d.R. synchrone Verarbeitung
- kommunizierte Einheiten sind Requests und Responses, enthalten typisierte Parameter usw. in einer Netzdendatendarstellung
- Basis für Client/Server-Anwendungen
- Bindung von Client und Server relativ statisch

Details im Kap. 3

Verbreitete RPC-Plattformen

SunRPC

- public domain, auf vielen Systemen lauffähig
- Bedeutung weniger durch Nutzung allgemeiner Anwendungen
- aber Netzwerkdateisystem NFS basiert auf SunRPC

OSF DCE RPC, Microsoft RPC

- DCE Distributed Computing Environment:
erste reiche Dienstumgebung
- DCE RPC: ursprünglich aus Apollo NCS RPC entstanden
- zu komplex in der Nutzung
- Microsoft RPC weitgehend kompatibel zu DCE RPC
- heute kaum noch genutzt

Apache Thrift

- sehr flexibles RPC-System
- Unterstützung aller relevanten Programmiersprachen
- Weit verbreitet

Paradigma: Objektorientierung

- Objekte (im Sinne der OO Programmierung) als verteilbare Einheiten
- Anwendung = verteiltes Objekt-„Geflecht“
- Interaktion durch Methodenaufrufe (mit Ortstransparenz, Zugriffstransparenz) auf Basis eines RPC-Mechanismus
- Wiederverwendung von Klassen auf Quellcodeebene
- Wesentliche Plattformen
 - ▶ OMG CORBA
 - ▶ Microsoft DCOM
 - ▶ Java RMI

Beispiel: RMI

- Java Remote Method Invocation (RMI) (Sun/Oracle)
 - ▶ Jüngste Plattform
 - ▶ Einfachste Nutzung
 - ▶ unterstützt ausschließlich homogene „Welt“ von verteilten Java-Objekten

Beispiel: Microsoft DCOM

- Microsoft DCOM

- ▶ Erweiterung von COM/OLE unter Nutzung von Microsoft RPC
- ▶ weitgehend proprietäre Plattform
- ▶ 1999 an Open Group übergeben
- ▶ Microsoft's Folge-Basis war .NET
- ▶ Bedeutung gesunken, aber Nutzung noch in Automatisierungstechnik

Beispiel: OMG CORBA

- Object Management Group (OMG)

- ▶ internationale Non-Profit Organisation von Herstellern, Software Häusern und Anwendern
- ▶ gegründet 1989
 - ★ 3Com, American Airlines, Canon, Data General, HP, Philips, Sun, Unisys, ...
- ▶ 1.000+ Mitglieder (Firmen, Organisationen, Hochschulen, ...)
- ▶ In der Vergangenheit hohes Tempo für Standardisierungsgremium
- ▶ offener, formaler Standardisierungsprozess basierend auf Request for Proposals (RFPs)
- ▶ Ziel: Definition von Schnittstellen, nicht Produktentwicklung
- ▶ <http://www.omg.org>: frei verfügbare Dokumente
- ▶ heute auch relevant bzgl.
 - ★ UML-Standardisierung
 - ★ Model Driven Architecture (MDA)

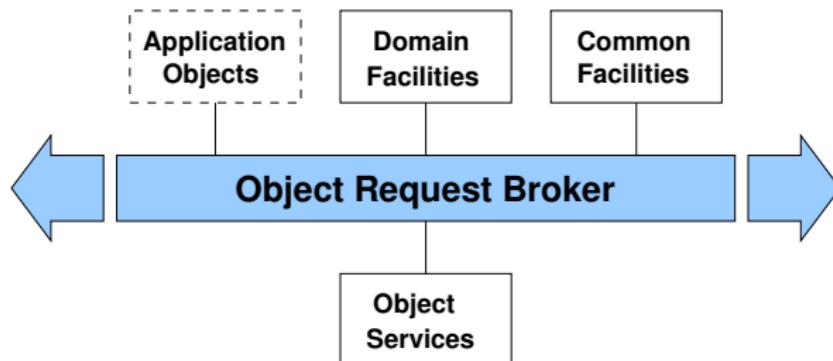
CORBA

- CORBA (Common Object Request Broker Architecture)
 - ▶ Architektur-, Betriebssystem- und Programmiersprachen-unabhängig
 - ▶ CORBA IDL an C++ angelehnte Schnittstellenbeschreibungssprache
 - ▶ IOR als systemweite Objektreferenz, GIOP/IIOP als Aufrufprotokoll
 - ▶ zahlreiche objektorientierte Dienste und Implementierungen verfügbar
- Kaum noch Bedeutung für neue Geschäftsanwendungen, aber noch Pflege

Object Management Architecture

OMA = Object Management Architecture

- Referenzmodell für verteilte, objektorientierte Anwendungen in heterogenen Umgebungen



ORB = Object Request Boker

- „Objekt-Bus“ = Kern der OMA
- vermittelt Aufrufe zwischen Objekten (stellenübergreifend, plattformübergreifend, Programmiersprachen-unabhängig)
- Interoperabilität zwischen verschiedenen ORBs

OMA-Objektmodell

Objekt

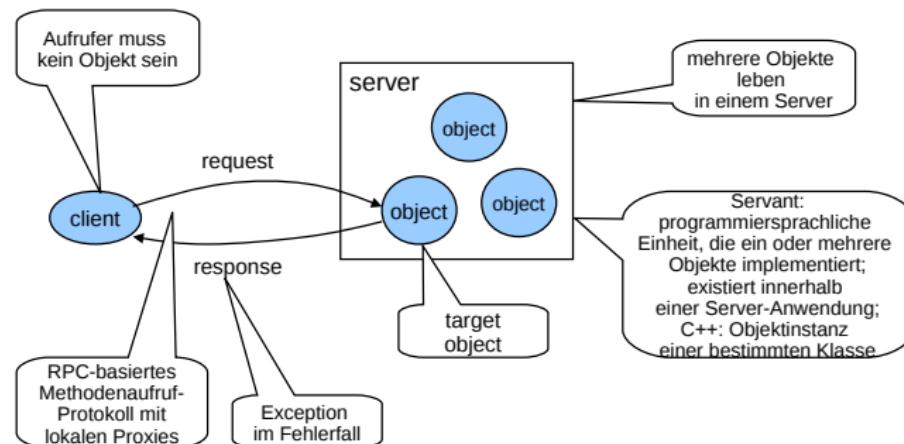
- „gedachte“, gekapselte Einheit auf einem System.
- wird „real“, wenn Implementierung in einer Programmiersprache dazu existiert.
- muss nicht einem Objekt auf Programmiersprachenebene entsprechen.
- besitzt unveränderbare Identität (identity).
- hat Zustand.
- kann durch ORB lokalisiert werden.
- hat Attribute (von außen zugreifbar)
- offeriert Operationen (Methoden), die durch Anfragen von Klienten (client requests) in Anspruch genommen werden.

OMA-Objektmodell (2)

Objektreferenz

- „handle“, um Objekt zu identifizieren, zu adressieren und zu lokalisieren
- interne Struktur für Klienten verborgen (opaque)
- bezieht sich auf ein bestimmtes Objekt

Zusammenhang



Arten von Requests

- synchron (*synchronous*)
 - ▶ Client blockiert, bis Response ankommt.
- verzögert synchron (*deferred synchronous*)
 - ▶ Client arbeitet nach Abschicken des Requests weiter, fragt später nach der Antwort (derzeit nur über DII möglich).
- Einwegauftrag (*oneway request*)
 - ▶ Best-effort-Zustellung ohne Response, muss nicht beim target object ankommen.
- asynchrone Aufrufe (*asynchronous requests*)
 - ▶ definiert im Rahmen von CORBA Messaging, Teil der Version CORBA 2.5 (2001).

Anwendungsentwicklung

CORBA Interface Definition Language (IDL)

- deskriptive Sprache zur Definition von Objektschnittstellen
(keine Kontrollkonstrukte)
- streng typisiert
- any-Typ erlaubt Flexibilität
- ISO 14750
- Beschreibung ist unabhängig von bestimmter Implementierungssprache
- syntaktisch an C++ angelehnt

Ein einfaches CORBA IDL-Beispiel

```
interface balance {
    exception out_of_tolerance {};
    readonly attribute long mode;
    long getweight_in_grams() raises (out_of_tolerance);
    void set_ref_weight_mode(in long ref_weight);
    unsigned short get_weight_in_percent();
    void reset_ref_weight_mode();
};

interface ext_balance : balance {
    exception out_of_tolerance {long difference};
    readonly attribute long mode;
    long getweight_in_carat() raises (out_of_tolerance);
    void set_tare_weight_mode(in long tare_weight);
    void reset_tare_weight_mode();
    void set_tolerance_weight_mode(in long min, in long max);
    void reset_tolerance_weight_mode();
};
```

Ein einfaches CORBA IDL-Beispiel

Schnittstellen-
definition

```
interface balance {
    exception out_of_tolerance {};
    readonly attribute long mode;
    long getweight_in_grams() raises (out_of_tolerance);
    void set_ref_weight_mode(in long ref_weight);
    unsigned short get_weight_in_percent();
    void reset_ref_weight_mode();
};

interface ext_balance : balance {
    exception out_of_tolerance {long difference};
    readonly attribute long mode;
    long getweight_in_carat() raises (out_of_tolerance);
    void set_tare_weight_mode(in long tare_weight);
    void reset_tare_weight_mode();
    void set_tolerance_weight_mode(in long min, in long max);
    void reset_tolerance_weight_mode();
};
```

Ein einfaches CORBA IDL-Beispiel

Schnittstellen-
definition Typen,
sichtbare Attribute

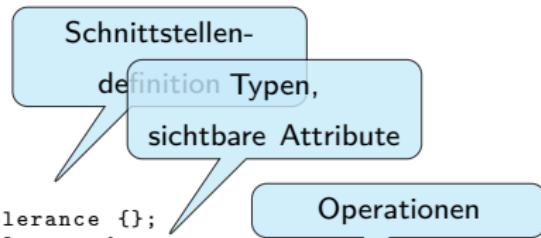
```
interface balance {
    exception out_of_tolerance {};
    readonly attribute long mode;
    long getweight_in_grams() raises (out_of_tolerance);
    void set_ref_weight_mode(in long ref_weight);
    unsigned short get_weight_in_percent();
    void reset_ref_weight_mode();
};

interface ext_balance : balance {
    exception out_of_tolerance {long difference};
    readonly attribute long mode;
    long getweight_in_carat() raises (out_of_tolerance);
    void set_tare_weight_mode(in long tare_weight);
    void reset_tare_weight_mode();
    void set_tolerance_weight_mode(in long min, in long max);
    void reset_tolerance_weight_mode();
};
```

Ein einfaches CORBA IDL-Beispiel

```
interface balance {
    exception out_of_tolerance {};
    readonly attribute long mode;
    long getweight_in_grams() raises (out_of_tolerance);
    void set_ref_weight_mode(in long ref_weight);
    unsigned short get_weight_in_percent();
    void reset_ref_weight_mode();
};

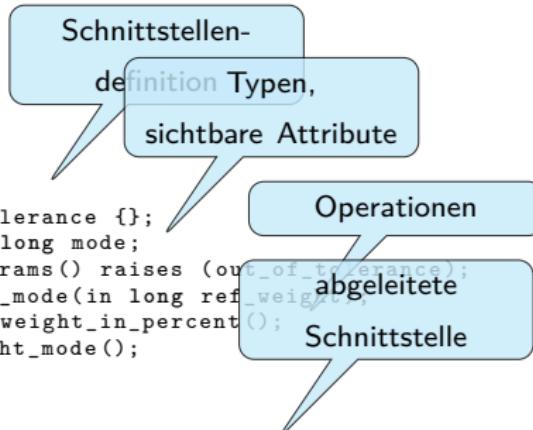
interface ext_balance : balance {
    exception out_of_tolerance {long difference};
    readonly attribute long mode;
    long getweight_in_carat() raises (out_of_tolerance);
    void set_tare_weight_mode(in long tare_weight);
    void reset_tare_weight_mode();
    void set_tolerance_weight_mode(in long min, in long max);
    void reset_tolerance_weight_mode();
};
```



Ein einfaches CORBA IDL-Beispiel

```
interface balance {
    exception out_of_tolerance {};
    readonly attribute long mode;
    long get_weight_in_grams() raises (out_of_tolerance);
    void set_ref_weight_mode(in long ref_weight);
    unsigned short get_weight_in_percent();
    void reset_ref_weight_mode();
};

interface ext_balance : balance {
    exception out_of_tolerance {long difference};
    readonly attribute long mode;
    long get_weight_in_carat() raises (out_of_tolerance);
    void set_tare_weight_mode(in long tare_weight);
    void reset_tare_weight_mode();
    void set_tolerance_weight_mode(in long min, in long max);
    void reset_tolerance_weight_mode();
};
```



- Schnittstellen-
definition Typen,
sichtbare Attribute
- Operationen
- abgeleitete
Schnittstelle

Language Mappings

- Spezifikation, wie IDL in verschiedene Programmiersprachen abgebildet wird
 - ▶ z.B. IDL Module auf C++ Namensraum oder Java package,
 - ▶ IDL Interface auf C++-Klasse,
 - ▶ IDL-Operationen auf deren Member-Funktionen.
- Standardisierte Language Mappings für
 - ▶ C, C++, Java, Smalltalk, COBOL, Ada, Lisp, PL/1, Python, IDLscript
- Andere definierte Language Mappings für:
 - ▶ Tcl, Perl, Eiffel, ...
- Konsequenz:
 - ▶ Unterschiedliche Teile einer verteilten Anwendung können mit verschiedenen Sprachen entwickelt sein
 - ▶ z.B. Server-Applikation in C++, Clients in Java.

Produkte

Wichtige kommerzielle ORBs:

- BEA M3 (Teil von BEA Tuxedo) (BEA gekauft von Oracle, 2008)
- IONA Orbix (IONA gekauft von Progress, 2008)
- ORBexpress RT, Orbriver RT, PrismTech OpenFusion
(für Echtzeitanwendungen)

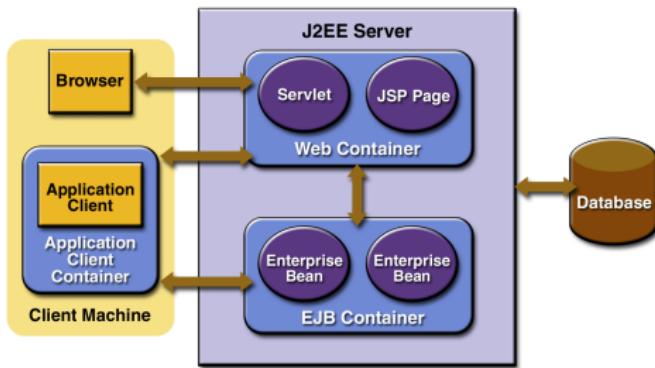
Wichtige freie ORBs:

- OOC ORBacus (noch weitgehend frei verfügbar, 2001 aufgekauft von IONA)
- MICO (Open Source Projekt, ursprünglich Uni Frankfurt)
- JacORB (FU Berlin, jetzt PrismTech OpenFusion)
- TAO (WUSTL) (Echtzeitverarbeitung)
- ORBit (Middleware für GNOME)

Paradigma: Komponentenorientierung

- Komponenten als verteilbare Einheiten
- Starke Unabhängigkeit und Austauschbarkeit der Komponenten
- Interaktion durch Methodenaufrufe (basierend auf RPC)
- Enterprise Java Beans (EJB) als am weitesten verbreitetes Komponentenmodell, daneben Microsoft .NET
 - ▶ Teil der Spezifikation von Java-Schnittstellen für Server-seitige Komponenten (J2EE, jetzt JEE)
 - ▶ enger Bezug zu CORBA
 - ▶ Ziel: Vereinfachung der Anwendungsentwicklung
 - ▶ Application Server als integrierte Infrastruktur für transaktionsorientierte Geschäftsanwendungen
 - ▶ Schnittstellen zu standardisierten Diensten (Persistenz, Transaction Management, Directory-Dienste, Messaging) zum Deployment-Zeitpunkt gebunden
 - ▶ Hohe Skalierbarkeit für Server-seitige Webanwendungen

Enterprise Java Beans



<http://www.rizzimichele.it/enterprise-java-beans-and-all-j2ee/>

• Komponenten

- ▶ Stateless und Stateful Session Beans (Ausführung einer Task für Client ohne bzw. mit Gedächtnis für denselben Client)
- ▶ Entity Beans (Repräsentierung von Geschäftsobjekten im persistenten Speicher, Unterstützung für Transaktionen)
- ▶ Message-driven Beans (asynch. Verarbeitung von Nachrichten, JMS-API)

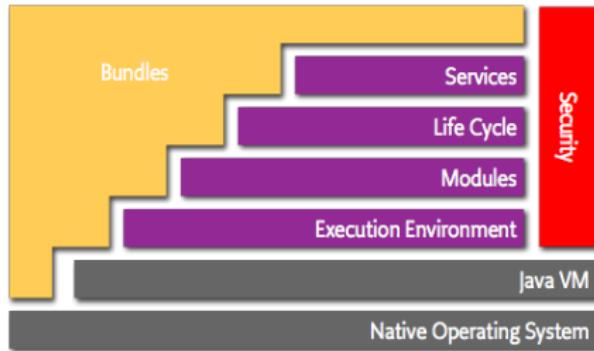
Verbreitete Produkte

- Freie:
 - ▶ JBoss Application Server (steil aufgestiegen, jetzt Red Hat)
 - ▶ Geronimo (Apache)
 - ▶ JOnAS (Object Web, Bull)
- IBM Websphere
- Oracle/BEA Weblogic
- SAP NetWeaver
- Sun GlassFish

OSGi - Komponentenmodell

- Aktuelles verbreitetes Komponentenmodell mit Java-Bezug für große verteilte Systeme bis zum Embedded-Bereich
- Von Entwicklern erzeugte Komponenten heißen „Bundles“
- Dynamisches Management von Komponenten (Lifecycle, incl. Updates, Remote Management)
- Unterstützung für Versionierung
- Einsatz: Technologie auch z.B. enthalten
 - ▶ als Equinox Plattform in Eclipse für dyn. Plugin-Management
 - ▶ zur internen Modularisierung in vielen Applikationsservern
 - ▶ Ursprung Home Automation, auch dort noch sehr aktiv (Smart Home, Residential Gateways, z.B. Telekom Qivicon)
 - ▶ Automotive / Telematik u.a.

OSGi - Architektur



<https://www.osgi.org/developer/architecture/>

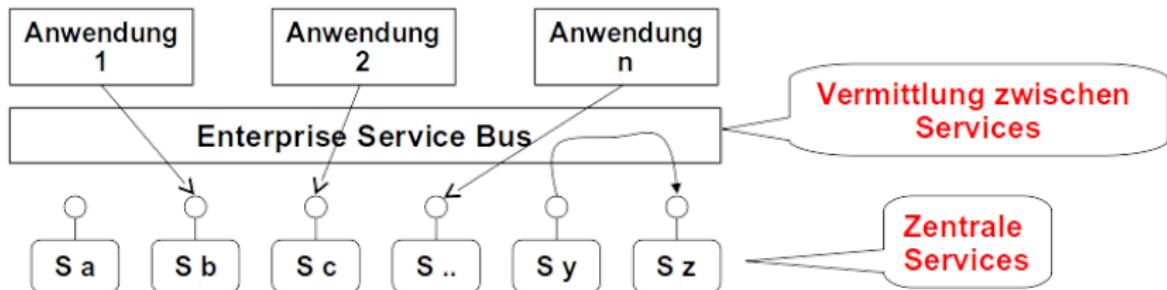
- Services verbinden Bundles dynamisch
- Life-Cycle - API für install, start, stop, update, uninstall
- Modules - Layer, der Import/Export von Code definiert
- Execution Environment - definiert Klassen und Methoden der Plattform

Paradigma: Service-Orientierung

- Service-Orientierte Architekturen (SOA)
- Architekturansatz für Geschäftsanwendungen zur Strukturierung und Nutzung von verteilten Diensten, die möglicherweise unter Kontrolle verschiedener Eigentümer stehen, mit dem Ziel, eine fachliche Strukturierung von Anwendungsmengen zu erreichen.
- Erwartete Vorteile:
 - ▶ Definition von Diensten anhand der Geschäftsprozesse
 - ▶ Gleichzeitige mehrfache Nutzung von Diensten in verschiedenen Anwendungen
 - ▶ Dadurch Reduktion ansonsten mehrfach zu pflegender Funktionalität
 - ▶ Zentrale Integration verschiedener Anwendungen statt paarweiser Schnittstellen

SOA: Dienste – Anwendungen

- Idealvorstellung:



- Zunehmende Kritik wegen Problemen:

- ▶ Komplette Dekomposition bestehender Anwendungen ist schwierig, aufwendig und für Nutzer nicht sichtbar
- ▶ Veränderung an zentralen Diensten betreffen viele Anwendungen
- ▶ Formalisierung der Geschäftsprozesse aus Diensten für Fachabteilungen schwierig

SOA: Technische Sichtweise

- Technische Sichtweise:

- ▶ Dienste autonom mit formalen Schnittstellen (Service Contracts) beschrieben in XML Schema Dokumenten
- ▶ Dienste halten möglichst keinen Zustand
- ▶ XML-Dokumente als kommunizierte Einheiten (Messages)
- ▶ Dienstbeschreibungen (Metadaten) in Verzeichnis (Service Registry)
- ▶ Dienste können über ihre Beschreibung dynamisch erkannt und angesprochen werden (kein Linking notwendig)
- ▶ Implementierungssprache / -technologie irrelevant
- ▶ Web Services als aktueller Hype zur Implementierung von SOA-Diensten
- ▶ Enterprise Service Bus (ESB) zur losen Kopplung der Dienste

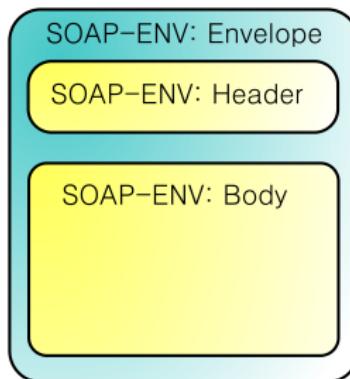
WSDL

- WSDL (Web Service Description Language)
 - ▶ Schnittstellen/Contract-Beschreibungssprache (abstrakt): Types, Messages, Interfaces, Services
 - ▶ W3C-Standard
 - ▶ XML-basiert

SOAP

- SOAP (ehemals Simple Object Access Protocol)
 - ▶ W3C-Standard
 - ▶ Objekte im Sinne der Objektorientierung existieren aber nicht
 - ▶ XML-Dokumenten-basiertes Interaktions-Framework für Web Services
 - ★ SOAP Messages (Envelopes aus opt. Header und Body)
 - ★ asynchrone Verarbeitung prinzipiell möglich
 - ★ SOAP Request/Response-Nachrichten für RPC-Stil
 - ▶ Protocol Binding Framework sieht verschiedene unterlagerte Transport-Dienste vor, neben HTTP/HTTPS auch z.B. SMTP, JMS
 - ▶ Java API for XML Web Services (JAX-WS) Teil von Java SE

SOAP: Beispiel



<https://commons.wikimedia.org/wiki/File:SOAP.svg>

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
    <s:Body>
        <m:TitleInDatabase xmlns:m="http://www.lecture-db.de/soap">
            DOM, SAX und SOAP
        </m:TitleInDatabase>
    </s:Body>
</s:Envelope>
```

Geschäftsprozesse

• Modellierung von Geschäftsprozessen

- ▶ Geschäftsprozess = komplexe Interaktion zwischen Diensten
- ▶ auch Web Services Orchestrierung genannt
- ▶ Programmieren im Großen
 - ★ Web Services als elementare Einheiten
- ▶ WS-BPEL (Business Process Execution Language)
 - ★ OASIS Standard
 - ★ Programm ist selbst XML-Dokument
 - ★ Mittlerweile untergeordnete Bedeutung
- ▶ BPMN (Business Process Model and Notation)
 - ★ Bisher genannt: Business Process Modelling Notation
 - ★ OMG Standard, verwandt zu UML-Aktivitätsdiagr., aktuell 2.0.2 (2013)
 - ★ = ISO/IEC 19510
 - ★ Soll Verständnis zwischen Technikern und Managern fördern

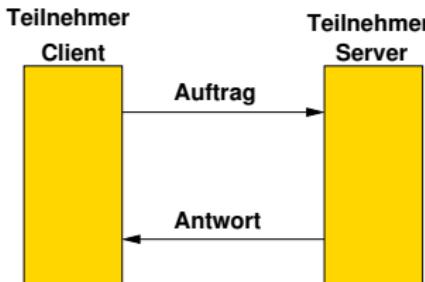
Grundlegende Architekturmödelle

Grundlegende Strukturmodelle für komplexe verteilte Anwendungen

- ① Client/Server-Modell
- ② Peer-to-Peer-Modell
- ③ Multi-Tier-Modell
- ④ SOA Modell

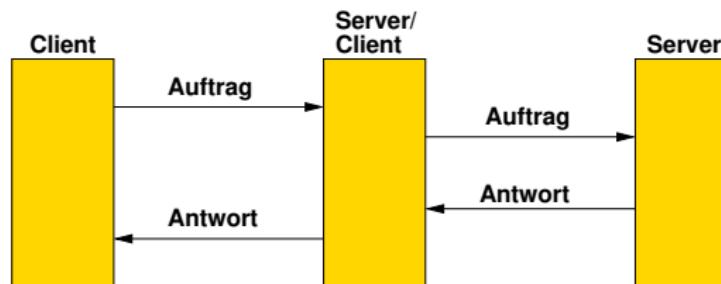
Client/Server-Modell

- Das Client/Server-Modell ist ein Software-Architekturnmodell für verteilte Anwendungen
- Es unterscheidet Rollen:
 - Server: Erbringer eines Dienstes (Service), z.B. Web-Server liefert Seiten
 - Client: Dienstnutzer, Kunde, Klient z.B. Browser fordert Seiten an
- Client und Server i.d.R. auf verschiedenen Rechnern



Client/Server-Modell (2)

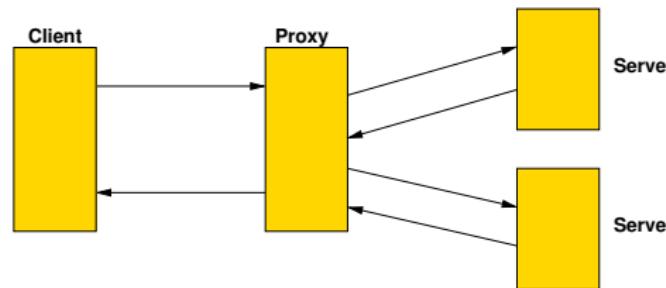
- Kommunikationsvorgänge basieren auf Auftrag/Antwort-Interaktionsmuster und werden vom Client initiiert.
- Ein Client kann im zeitlichen Verlauf mit mehreren Servern arbeiten
- Ein Server kann Aufträge für verschiedene Klienten ausführen.
- Ein Server kann als Client gegenüber weiteren Servern auftreten (Wechsel der Rolle):



Client/Server-Modell (3)

Proxy

- zwischengeschaltete Instanz
- Server-Rolle gegenüber Client,
- Client-Rolle gegenüber eigentlichen Servern
- Aufgaben z.B. Caching, Modifikation der Anfragen, ...
- Beispiel: Proxy-Server für Web-Seiten



Peer-to-Peer-Modell (P2P)

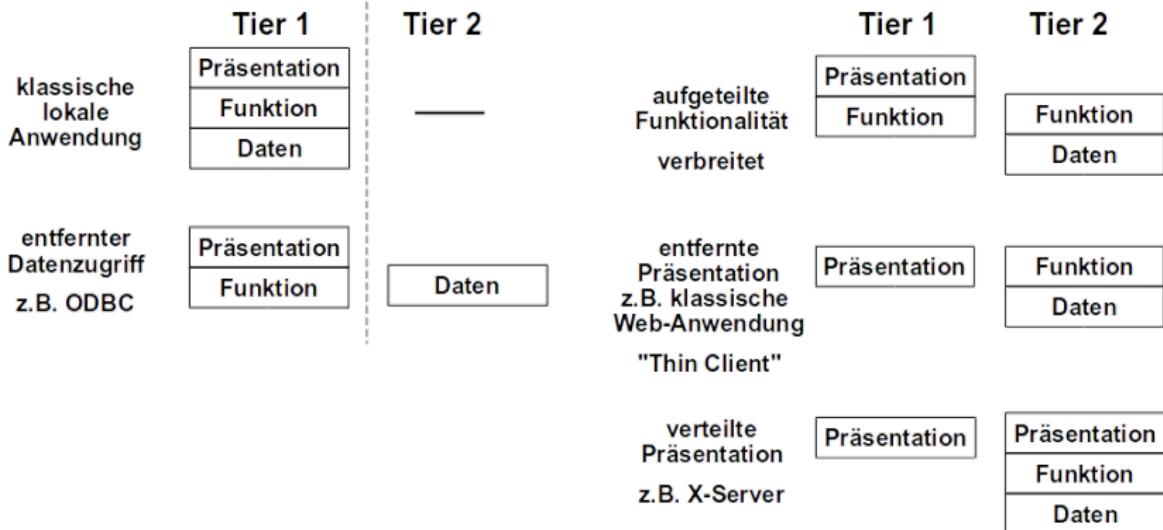
- dezentrale Kommunikation zwischen Gleichrangigen (= Peer)
- keine zusätzliche Infrastruktur (z.B. Server)
- Basis für Ad-hoc-Kommunikation
- Netzwerk- oder Anwendungsebene
- beliebige nachrichtenorientierte Interaktion
- Beispiele
 - ▶ File-Sharing, z.B. BitTorrent, Gnutella, eMule
 - ▶ P2P-Entwicklungsplattformen JXTA, MSP2P

Multi-Tier-Modell

- *Tier* = Reihe, Strang
- Stränge eher orthogonal zu (Abstraktions)-Schichten, i.d.R. orientiert an
 - ▶ Benutzerschnittstelle / Präsentation
 - ▶ Anwendungslogik / Funktion
 - ▶ Datenhaltung
- enthält keine Festlegung über verwendete Middleware
- heute sehr verbreitet
- üblich
 - ▶ Two-Tier-Architektur
 - ▶ 3-Tier-Architektur
 - ▶ N-Tier-Architektur

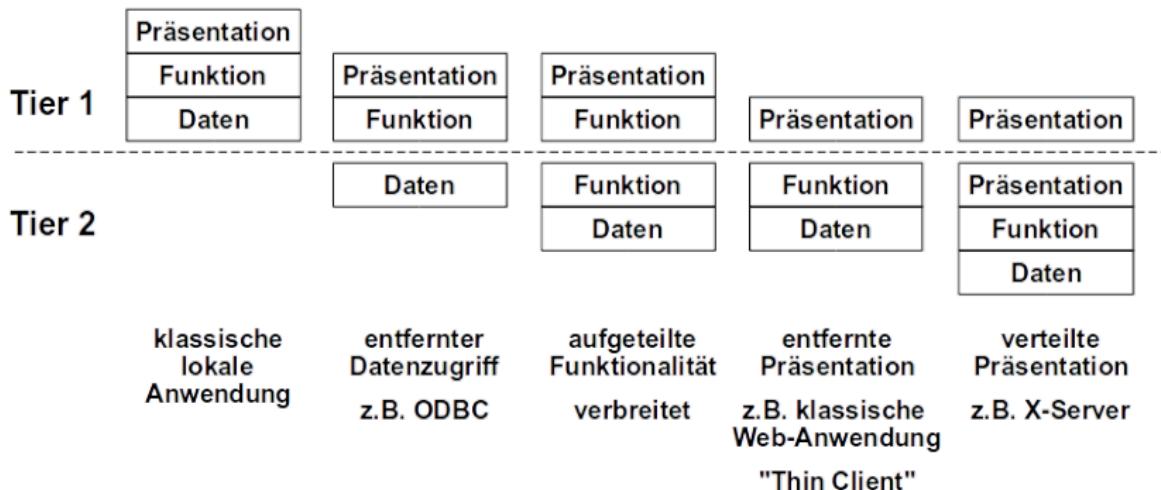
Two-Tier-Architektur

- Enthält Client-Strang (Tier 1) und Server-Strang (Tier 2)
- Einfache mögliche Aufteilungen



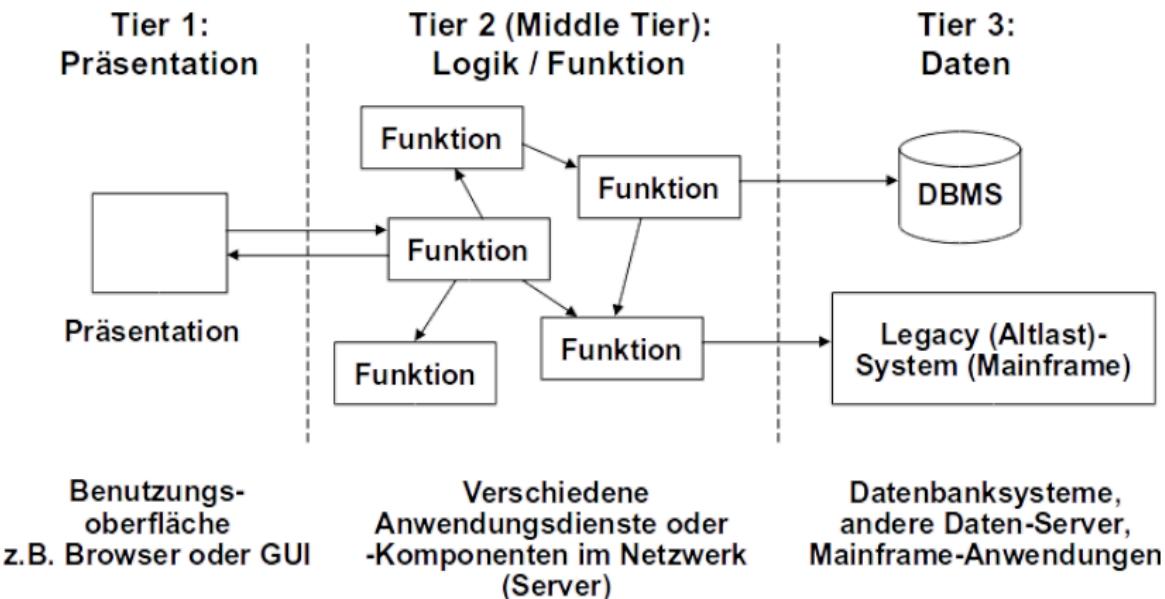
Two-Tier-Architektur (2)

- Übersichtlicher, aber Tier-Anordnung falsch.



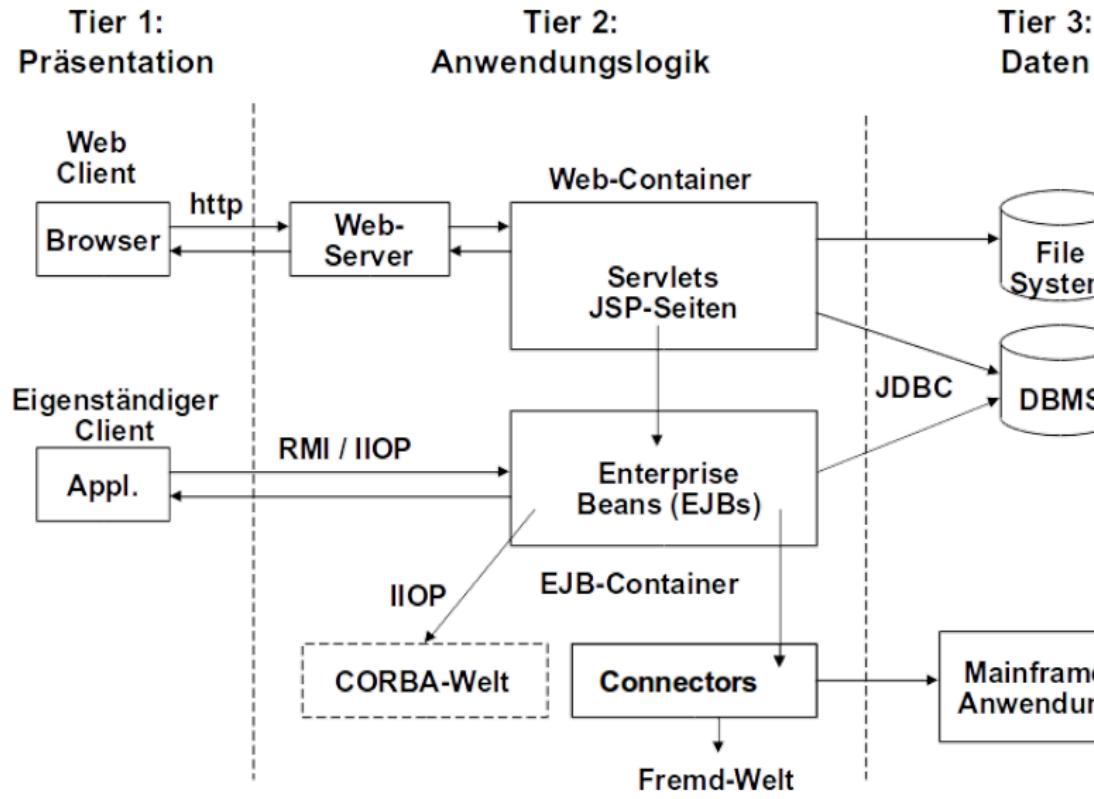
3-Tier-Architektur

- aktuelles Strukturmodell für komplexe Anwendungen

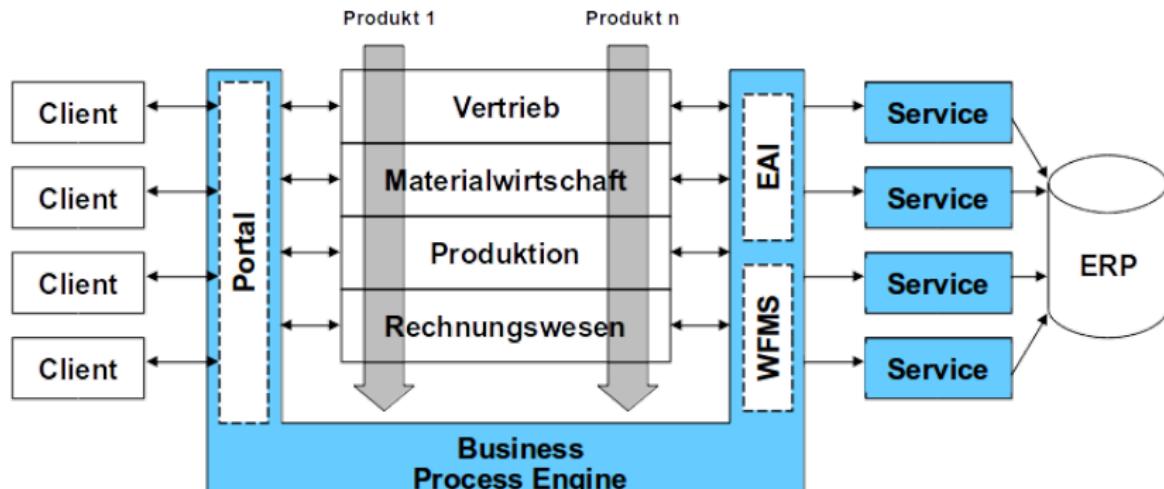


- Erweiterung auf N-Tier-Architektur
Spalten primär des Middle Tiers

Beispiel: J2EE-Anwendung (vereinfacht)



SOA-Modell



Web

Choreography

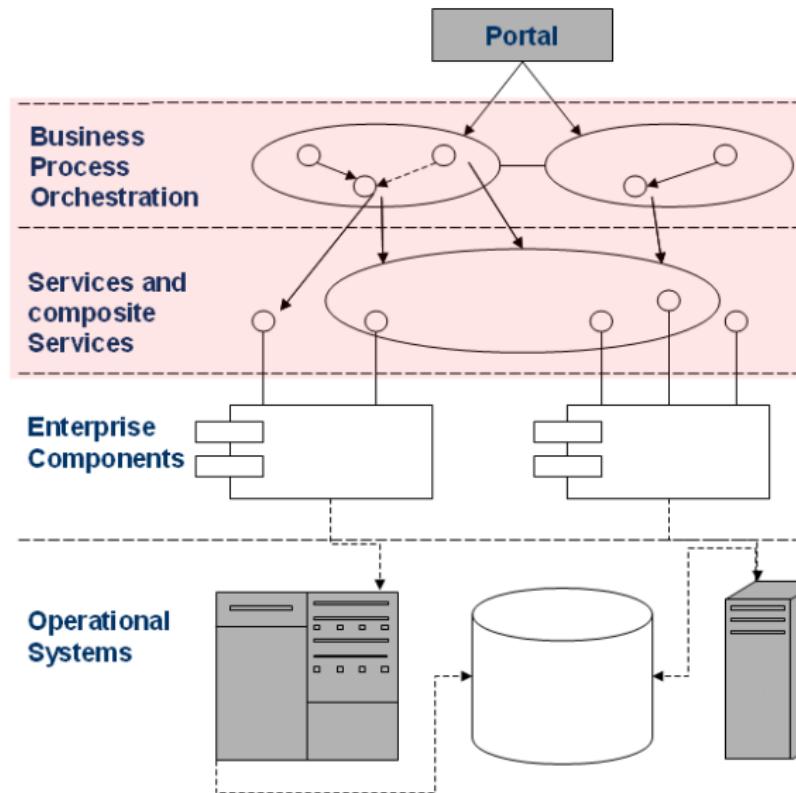
Web
Services

Nach Scheer

SOA aus technischer Sicht

Geschichtetes System

- IT-Geschäftskomponenten nutzen Ressourcen
- Komponenten stellen Teilfunktionalität als Dienst bereit
- Komplexe Dienste können aus einzelnen Basisdiensten kombiniert werden
- Geschäftsprozesse verknüpfen Dienste zu Anwendungen (Choreography / Orchestration)



Optional

- Enterprise Service Bus zur Kommunikation über Protokollgrenzen hinweg

Zusammenfassung

- Eine Middleware stellt eine Schicht zwischen Betriebssystem und Anwendung bereit, um verteilte Anwendungen von den darunter liegenden Schichten abstrahieren zu können.
- Middleware-Architekturen beschreiben die verteilbaren Einheiten und Interaktionsmodelle.
- Bei dem Entwurf eines verteilten Systems können je nach Anforderung unterschiedliche Architekturmodelle zum Einsatz kommen.

5. Namens- und Verzeichnisdienste



<http://www.sa-23e.com/wordpress/wp-content/uploads/2013/05/telefonbuch.jpeg>

Inhalt

5. Namens- und Verzeichnisdienste

5.1 Einführung

5.2 Namen

5.3 Namensdienste

5.4 Verzeichnisdienste

5.5 Lokationsdienste

Einführung

- Numerische Identifier und Adressen (z. B. IP-Adressen) sind für den Menschen schwer zu merken.
- Namen dienen zur Abstraktion von konkreten Diensten oder Objekten
- Aber wie komme ich von einem Namen zu einer Adresse?
- In Kapitel 3 (RPC) wurde besprochen:
 - ▶ Naming und Locating/Addressing von Diensten durch Binder
- Hier Verallgemeinerung:
 - ▶ Namensdienste
 - ▶ Verzeichnisdienste
- Auch in anderen Kontexten haben wir es mit Namensauflösung zu tun (z. B. Variablenname → Speicheradresse)

Namen

- Namen:

- ▶ Namen werden genutzt, um Objekte (z.B. eine Ressource oder einen Service) zu identifizieren.
- ▶ Ein Name ist ein Bit- oder Zeichenstring
- ▶ Binding: der Prozess, der einen Namen an ein Objekt bindet

- Eigenschaften von Namen

- ▶ *unique*: ein Name identifiziert eindeutig (höchstens) ein Objekt
- ▶ *pure/rein*: ein Name ist nur ein Bit-Muster und enthält keinerlei weitere Information
- ▶ *impure/unrein*: ein Name impliziert zusätzliche Information über das bezeichnete Objekt

Beispiele

- unique

- ▶ „Erika Mustermann“ ist nicht unique
 - ★ Name mit Geburtstag und Geburtsort sollte für Menschen unique sein
- ▶ UUID (Universally Unique Identifier) sind unique
 - ★ 128 Bit-Zahl
 - ★ spezifiziert in RFC 4122 und ITU-T Rec. X.667 | ISO/IEC 9834-8:2005
 - ★ unterschiedliche Versionen
 - ★ generiert durch Tool `uuidgen`
 - ★ in der Microsoft-Welt auch *Global Unique Identifier (GUID)*
 - ★ Beispiel: 123e4567-e89b-12d3-a456-426614174000

- pure

- ▶ UUIDs als Namen von DCOM-Objekten oder Klassen sind pure

- impure

- ▶ DNS-Namen implizieren oft zusätzliche Information
 - ★ z. B. `mail.hs-rm.de`

URI, URL und URN

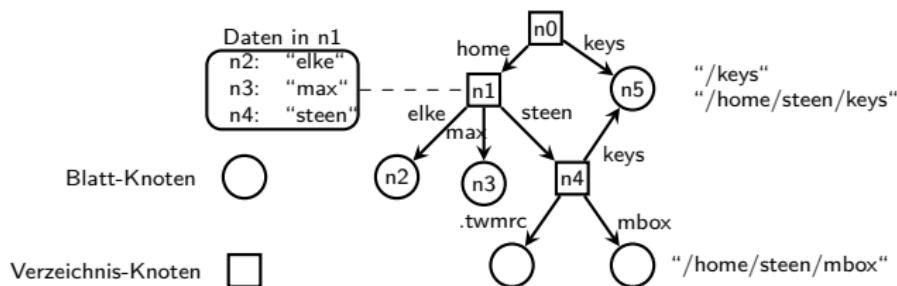
- Ein *Uniform Resource Identifier* (URI) identifiziert eine konkrete Ressource,
z. B. ISBN 978-1543057386.
- Ein *Uniform Resource Locator* (URL) gibt darüber hinaus an wie auf die Ressource zugegriffen werden kann,
z. B.
<https://www.libra-buchhandlung.de/shop/item/9780131217867>.
- Ein *Uniform Resource Name* (URN) identifiziert eine konkrete Ressource persistent und ortsunabhängig,
z. B. urn:isbn:978-1543057386.

Namensräume (1)

- Namen haben nur in einem bestimmten Kontext eine Bedeutung.
- Namen werden in Namensräumen strukturiert.
- Namensräume geben die Syntaxregeln für die Namen vor.
- Beispiele: Namespaces in C++, DNS, ISBN...

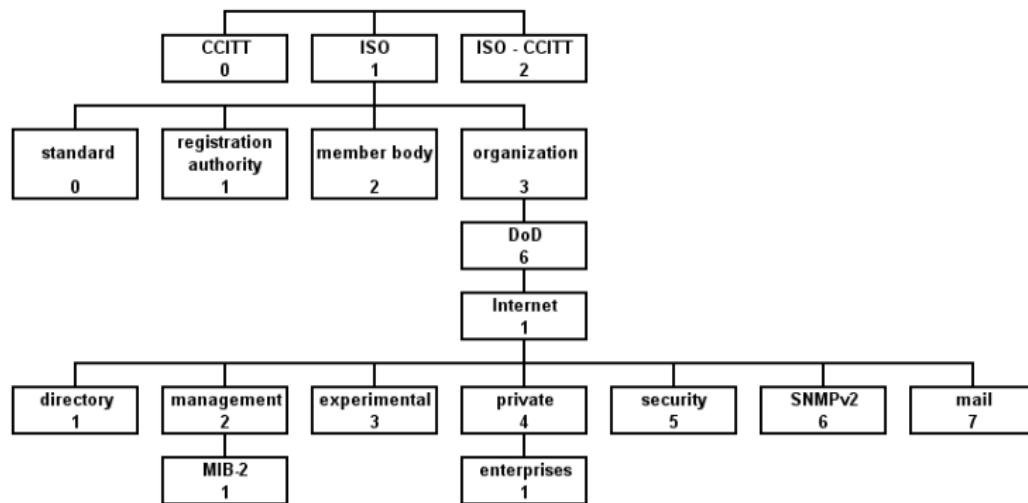
Namensräume (2)

- Flache Namensräume (heute eher selten, z.B. Unix UIDs)
- Hierarchische Namensräume werden üblicherweise als gerichtete Graphen mit Labels organisiert.
- In solchen Namensräumen liefert dann der Präfix den Kontext.
 - ▶ Verzeichnisknoten und Blätter
 - ▶ Absolute und relative Pfade
 - ▶ Globale und lokale Namen
- Beispiel: Unix-Dateinamensraum



Beispiel: MIB-2 - Namensraum

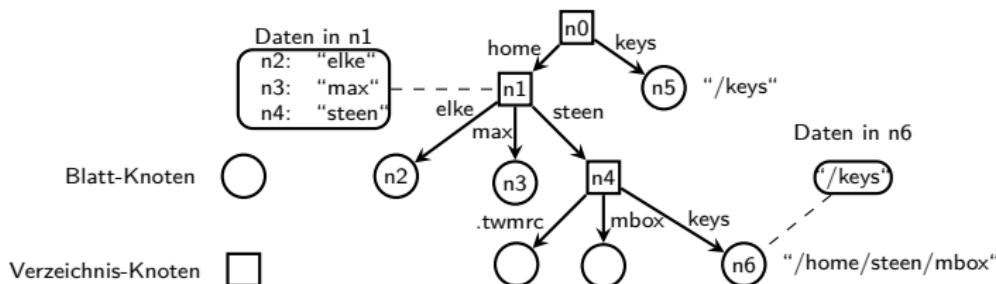
- MIB: Management Information Base



<https://upload.wikimedia.org/wikipedia/commons/1/1c/SNMP.MIB-Tree.PNG>

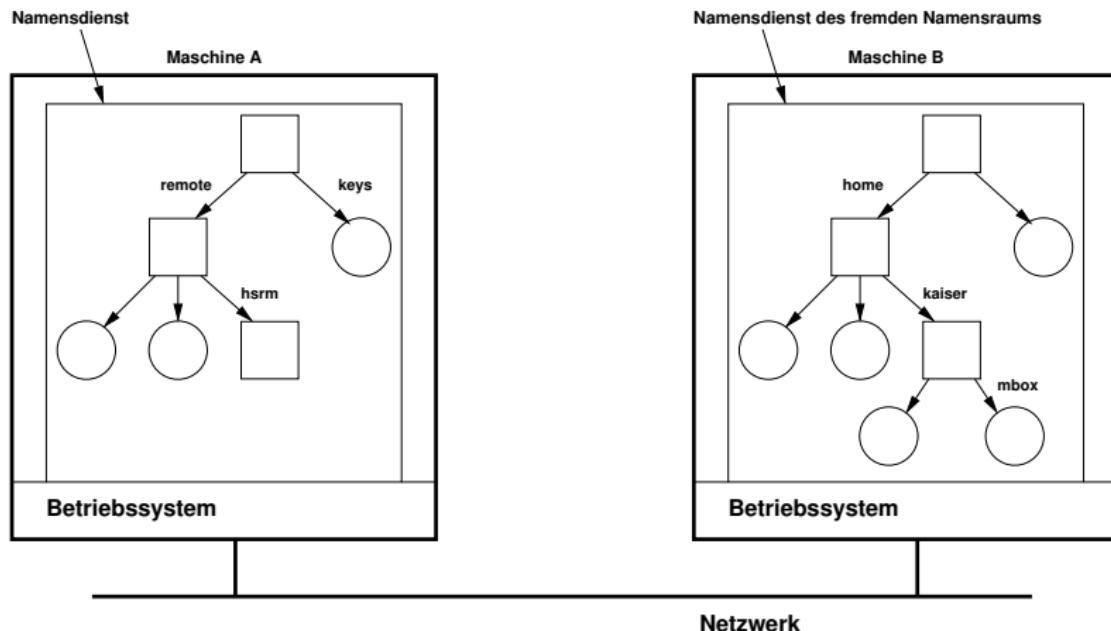
Links in einem Namensraum

- Das Objekt eines Namens ist ein weiterer Name
- Weiterleitung bzw. Abbildung eines Namens auf einen anderen Namen
- Beispiel: Unix Soft-Link



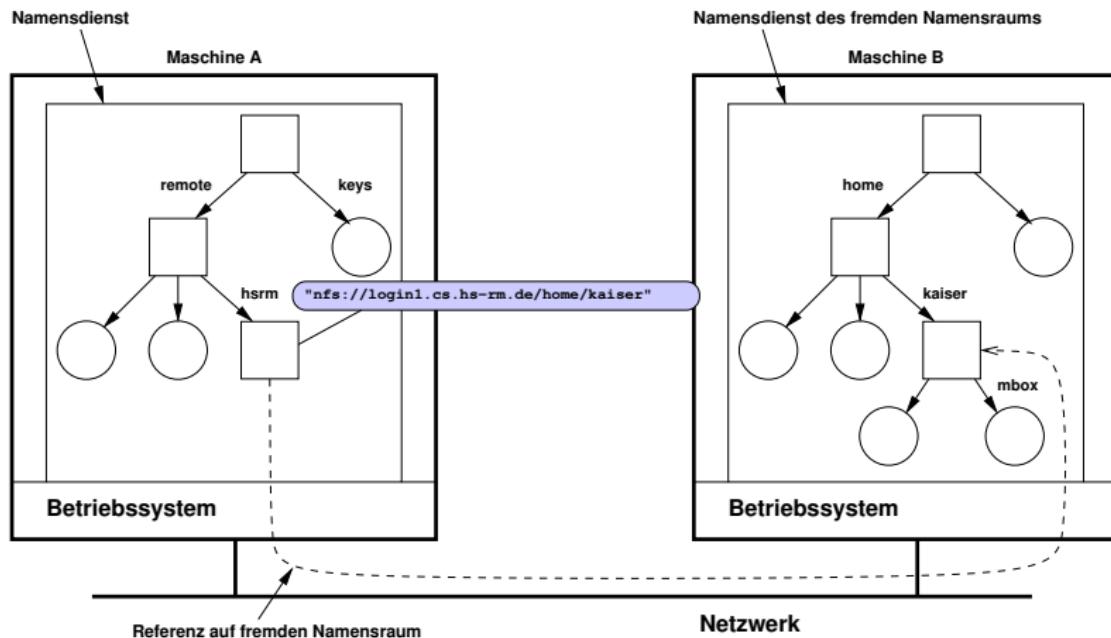
Links in einen anderen Namensraum

• Mounting



Links in einen anderen Namensraum

• Mounting



Organisation von großen Namensräumen

- Um große Namensräume effektiv verwalten zu können, sind diese typischerweise in drei Layer geteilt:
 - ▶ Global Layer
 - ★ High-level Knoten (Einstiegspunkte)
 - ▶ Administrative Layer
 - ★ Namensräume innerhalb einer Organisation
 - ▶ Managerial Layer
 - ★ Namensräume mit Namen, die sich häufig ändern

● Eigenschaften:

	Global	Administrational	Managerial
Geographical scale of network	Worldwide	Organization	Department
Total number of nodes	Few	Many	Vast numbers
Responsiveness to lookups	Seconds	Milliseconds	Immediate
Update propagation	Lazy	Immediate	Immediate
Number of replicas	Many	None or few	None
Is client-side caching applied?	Yes	Yes	Sometimes

Beispiel: DNS (Domain Name Service)

Vgl. LV Rechnernetze

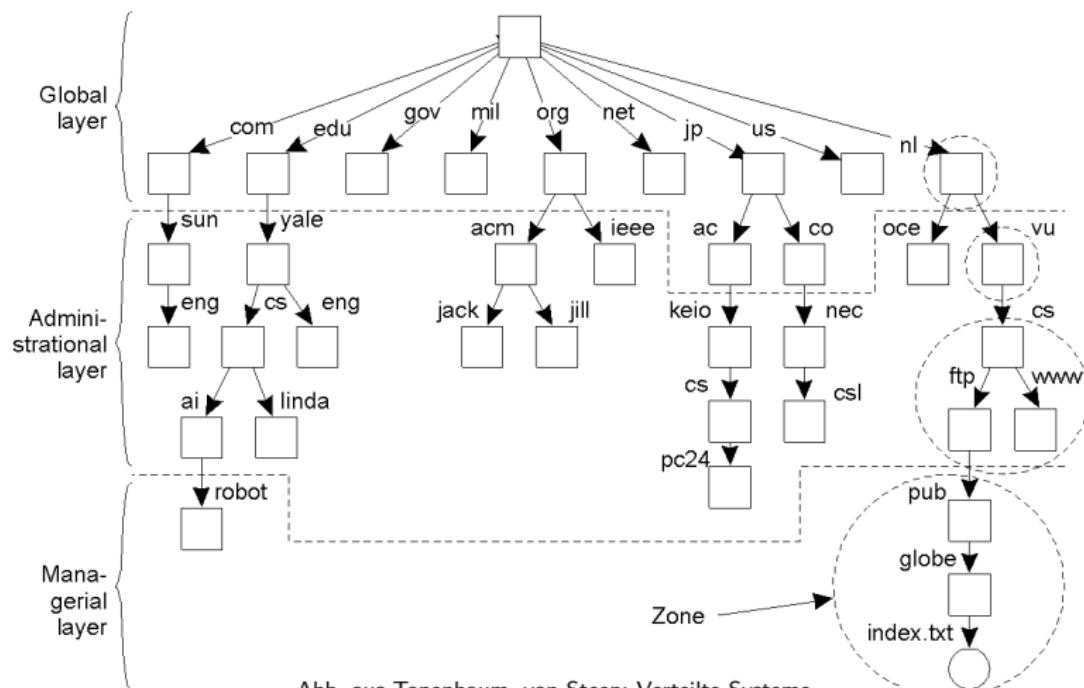


Abb. aus Tanenbaum, van Steen: Verteilte Systeme

Adressen

- Adressen sind Attribute von Namen, die genutzt werden können, um mit dem Objekt zu interagieren / zuzugreifen
 - ▶ Beispiele von Adressen
 - ★ Straße, Hausnr., Ort
 - ★ Telefonnr.
 - ★ (IP-Adresse, Portnummer)
 - ★ Speicheradresse
- Vorteile der Nutzung von Namen gegenüber Adressen
 - ▶ Ortsunabhängig (wünschenswert)
 - ▶ Besser zu merken
 - ▶ Abstrahiert von vielen (Protokoll)-Details der Adresse

Namensdienste

- Namensauflösung:

Vorgang, um von gegebenem Namen eines Objekts zu seinem Adress-Attribut zu kommen

- Namensdienst (Name Server):

Realisierung der Namensauflösung für anfragende Clients

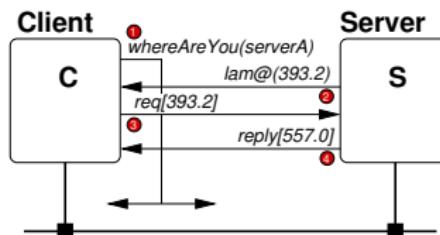
- ▶ Im Falle von RPC-Systemen auch Binder genannt
- ▶ Typische Operationen:

- ★ Register / Bind
- ★ Deregister / Unbind
- ★ Resolve / Lookup

Arten der Namensauflösung

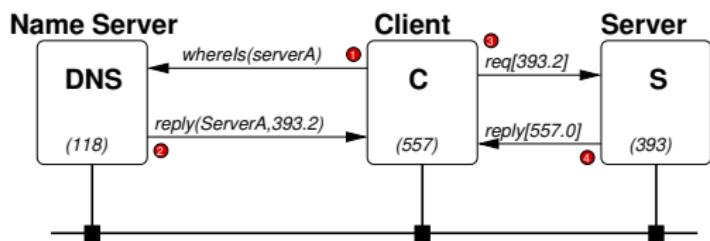
- Suche durch Broadcast

- Anfrage wird an alle gesendet; nur die Einheit antwortet, die den Namen auflösen kann.
- Nachteil: Skaliert nicht
- Beispiel: ARP zur Auflösung von IP-Adressen (Namen) in MAC-Adressen



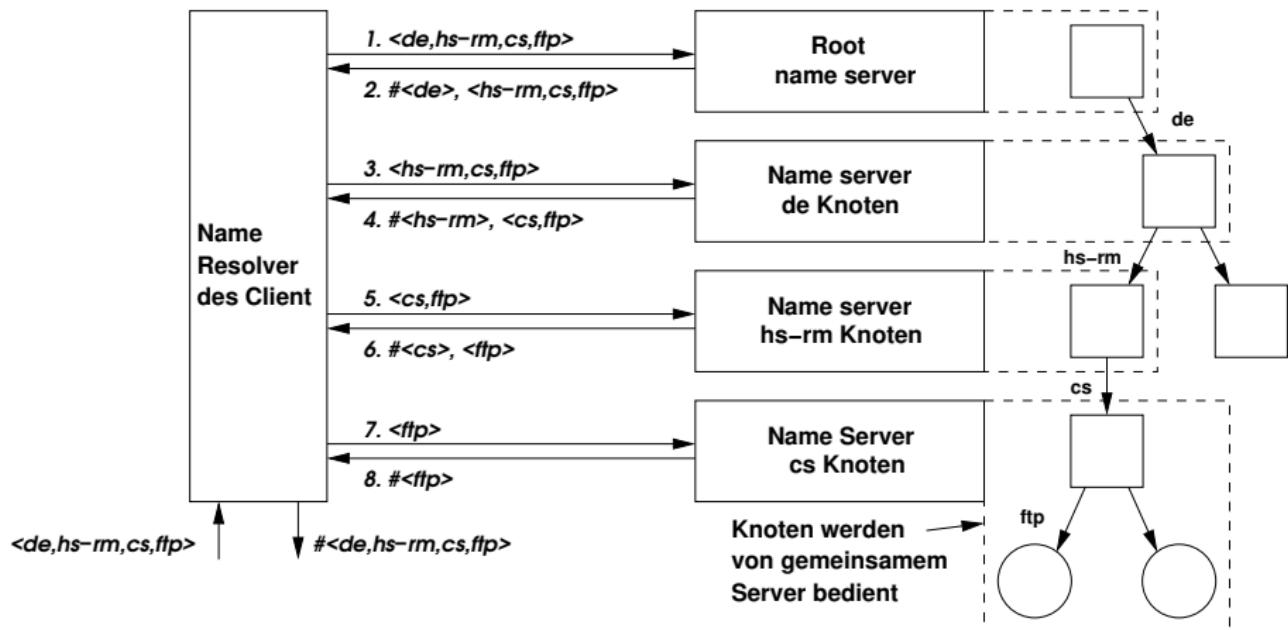
- Nutzung Name Server

- Es wird ein dedizierter Server gefragt, der die Abbildung hält
- Nachteil: benötigt well-known Adresse
- Beispiel: DNS



Namensauflösung mit Name-Servers (1)

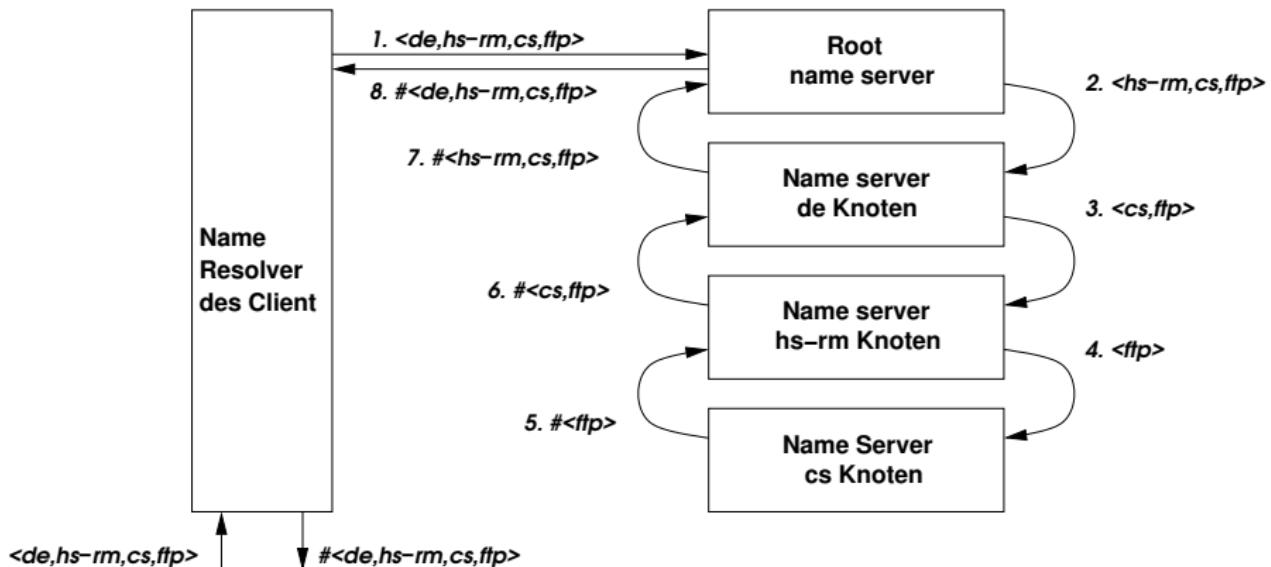
- Iterative Namensauflösung
 - ▶ vom Client aus
 - ▶ Caching nur beim Client



Namensauffölung mit Name-Servern (2)

• Rekursive Namensauffölung

- ▶ Caching auf den Servern möglich
- ▶ Weniger Kommunikation
- ▶ Mehr Last auf den Root-Servern



Verbreitete Namensdienste

- DNS (Internet Domain Name Service)
 - ▶ Vgl. LV Rechnernetze
- JNDI (Java Naming and Directory Interface)
- Java RMI Registry
 - ▶ Vgl. Praktikum
- CORBA INS (Interoperable Naming Service)
 - ▶ URLs als Namen für CORBA-Objekte

Verzeichnisdienste

- Verzeichnisdienst = Directory Service
- Unterschied zu Namensdienst
 - ▶ Erweiterung
 - ▶ Analogie: „Gelben Seiten“ zu Telefonbuch
 - ▶ Im Verzeichnisdienst werden Einträge nicht in erster Linie über ihren Namen gesucht, sondern über Eigenschaften
- Standards
 - ▶ X.500 (ITU-T ehem. CCITT)
 - ★ Komplex, nutzte ISO/OSI-Stack und Directory Access Protocol (DAP)
 - ▶ LDAP (Lightweight Directory Access Protocol)
 - ★ Implementiert nur einen Teil des X.500-Standards
 - ★ Setzt auf TCP/IP auf
 - ★ LDAP = Lightweight-Version von DAP
 - ★ Heute versteht man unter LDAP nicht nur das Zugriffsprotokoll sondern auch den Server des Verzeichnisses (LDAP-Server)

LDAP (Protokoll)

- Aktuelle Version 3 ist spezifiziert im RFC 4511.
- Unterstützung unter vielen Betriebssystemen.
- LDAP - Server beherrschen Replikation und Delegation (Referral).
- LDAPv3 unterstützt SSL, SASL und Kerberos-Authentisierung.
- Die meisten LDAP-Daten sind Textstrings (einfache Kodierung bei Netzübertragung), Binärdaten können verarbeitet werden.

LDAP (Verzeichnis)

- Hierarchischer Namensraum: Directory Information Tree (DIT)
- Einträge (Knoten des Baums) können beliebige LDAP-Objekte sein
- LDAP-Objekt besteht aus Menge von <Attribut, Wert>-Paaren.
- Klassen definieren Objekttypen mit bestimmten Attributmengen und Wertmengen.
- Jedes Objekt gehört zu mindestens einer Klasse.
- Es gibt Schemata für vordefinierte Klassen (z.B. Person, Organisation)
- Vererbung möglich
- Anwendungsspezifisch erweiterbar

Beispiel

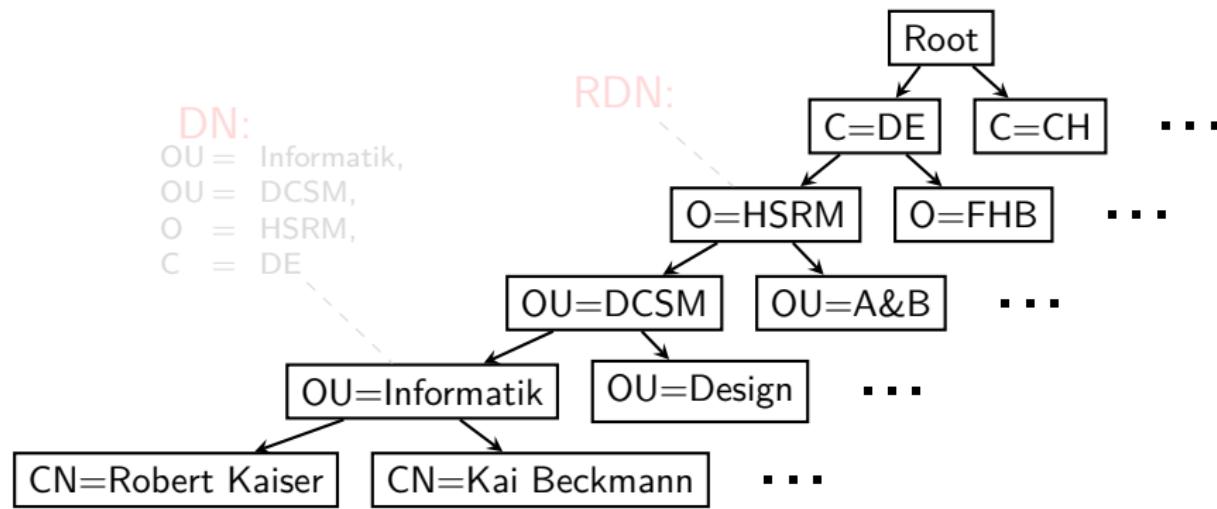
Attribut	Kürzel	Wert
Country	C	DE
Locality	L	Wiesbaden
Organization	O	HSRM
OrganizationalUnit	OU	DCSM
OrganizationalUnit	OU	Informatik
CommonName	CN	Robert Kaiser

RDN und DN

- Ausgangspunkt: DIT Wurzel → Base Object
- Jeder Knoten hat in seiner Ebene einen eindeutigen Namen, genannt **Relative Distinguished Name (RDN)**
- Zusammensetzung der RDNs von Knoten bis zur Wurzel heisst **Distinguished Name (DN)** (vgl. Pfadnamen)

DN:
 OU = Informatik,
 OU = DCSM,
 O = HSRM,
 C = DE

RDN:

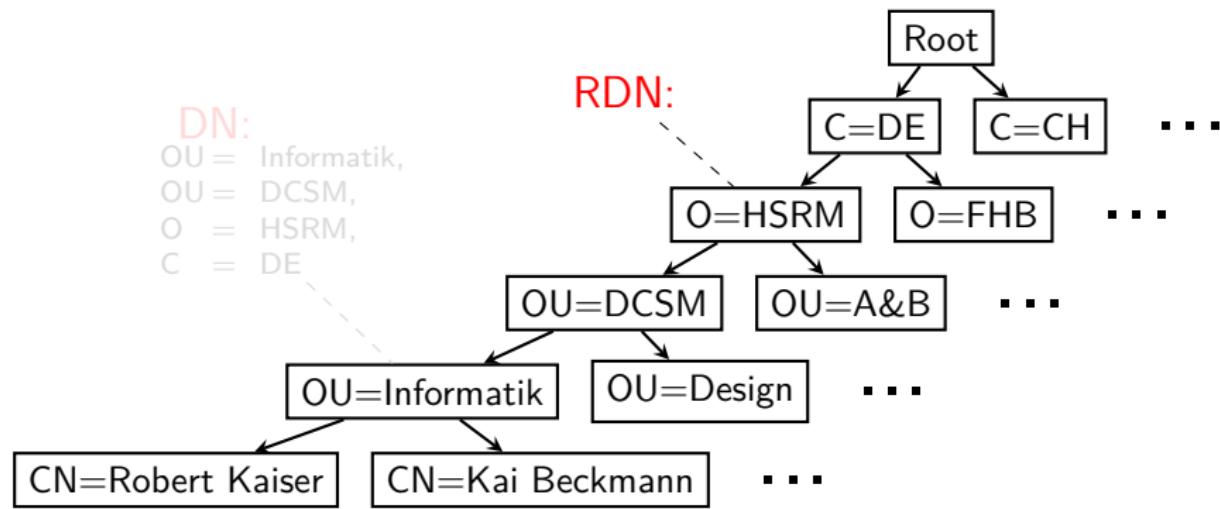


RDN und DN

- Ausgangspunkt: DIT Wurzel → Base Object
- Jeder Knoten hat in seiner Ebene einen eindeutigen Namen, genannt **Relative Distinguished Name (RDN)**
- Zusammensetzung der RDNs von Knoten bis zur Wurzel heisst **Distinguished Name (DN)** (vgl. Pfadnamen)

DN:
 OU = Informatik,
 OU = DCSM,
 O = HSRM,
 C = DE

RDN:

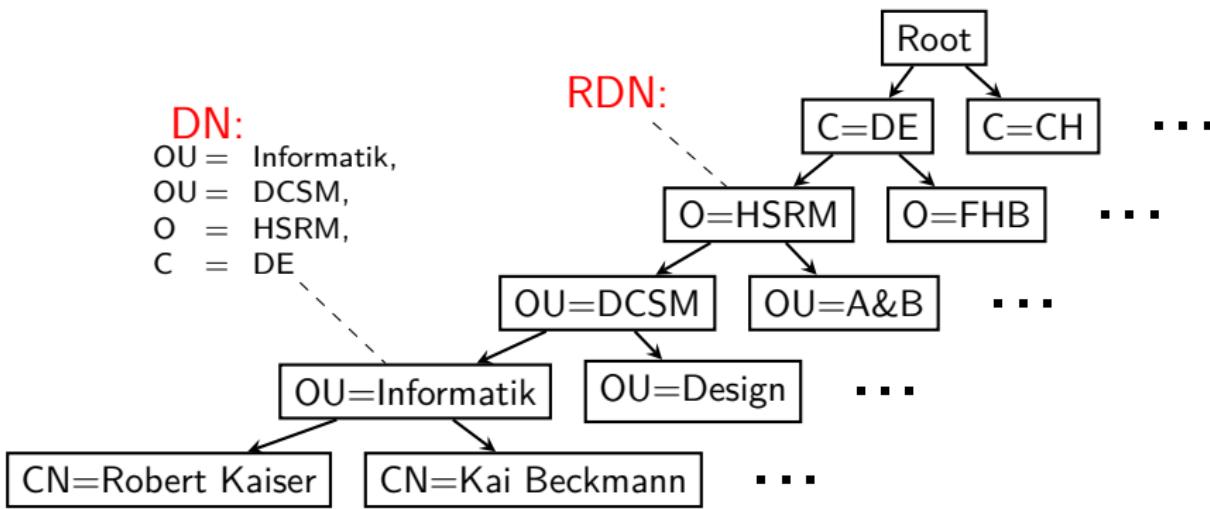


RDN und DN

- Ausgangspunkt: DIT Wurzel → Base Object
- Jeder Knoten hat in seiner Ebene einen eindeutigen Namen, genannt **Relative Distinguished Name (RDN)**
- Zusammensetzung der RDNs von Knoten bis zur Wurzel heisst **Distinguished Name (DN)** (vgl. Pfadnamen)

DN:
 OU = Informatik,
 OU = DCSM,
 O = HSRM,
 C = DE

RDN:



Operationen

- Bind — Authentifizierung
- Add — Hinzufügen eines Eintrags
- Delete — Löschen eines Eintrags
- Search — Suchen von Einträgen
- Compare — Vergleichen von LDAP-Objekten
- Modify — Bearbeiten eines LDAP-Objekts
- ModifyRDN — Verschieben/umbenennen eines Objekts
- Abandon — Abbrechen einer laufenden Anfrage
- Unbind — Abmeldung eines Clients

Anfragen

- Nutzung als Namensdienst

- ▶ Finde Objekt bei gegebenem Distinguished Name.
- ▶ z.B. `read(/C=DE/O=HSRM/OU=Informatik/CN=Robert Kaiser)`, damit Zugriff auf alle Attribute des Objekts.

- Suche von Objekten mit bestimmten Attributwerten.

- ▶ Anfragen können mehrere Ergebnisse liefern.

- Anfragen können komplex sein:

- ▶ Wildcards, Logische Ausdrücke: z.B. `&(C=DE)(CN=Kaiser*)`

Anfragen

- Nutzung als Namensdienst

- ▶ Finde Objekt bei gegebenem Distinguished Name.
- ▶ z.B. `read(/C=DE/O=HSRM/OU=Informatik/CN=Robert Kaiser)`, damit Zugriff auf alle Attribute des Objekts.

- Suche von Objekten mit bestimmten Attributwerten.

- ▶ Anfragen können mehrere Ergebnisse liefern.

- Anfragen können komplex sein:

- ▶ Wildcards, Logische Ausdrücke: z.B. `&(C=DE)(CN=Kaiser*)`

Replikation

- Teile des Namensraums sind i.d.R. auf mehreren Servern repliziert
 - ▶ Aus Gründen der Fehlertoleranz und der Performance
 - ▶ Insbesondere die zentralen Teile
 - ▶ Replikation kann Stunden dauern
 - ▶ Master-Slave Konfiguration
 - ★ Änderungen nur auf Master
 - ★ Propagation an Slaves
- Problem: Update-Zeiten bei Änderungen
 - ▶ Updates sind nicht sofort global sichtbar
 - ▶ Vertretbar nur, wenn
 - ★ Read/Write-Verhältnis groß
 - ★ Das Lesen veralteter Einträge unkritisch ist

Anwendungen

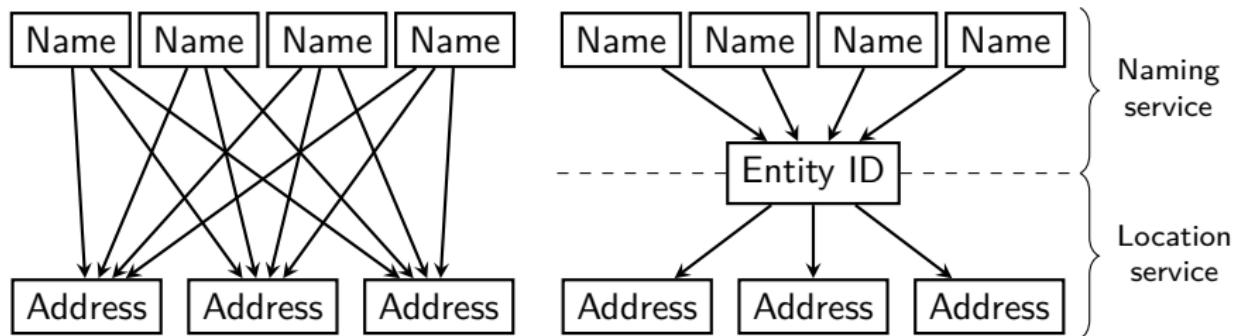
- Benutzerverwaltung (Identity Management)
 - ▶ Schema: inetOrgPerson (RFC 2798)
- Adressbücher von Mail-Systemen
 - ▶ Z.B. Thunderbird-Schnittstelle zu LDAP
- Unternehmensorganisation
 - ▶ Information entsprechend Organigrammen
- Inventarsysteme / Infrastrukturverwaltung

Produkte LDAP/X.500

- OpenLDAP (Open Source)
- NetIQ eDirectory (früher Novell eDirectory, davor Novell Directory Services NDS)
- MS Active Directory (mit LDAP Interface)
- Atos DirX (früher Siemens DirX)
- Oracle Directory Server (früher Sun Directory Server)
- ...

Lokationsdienste

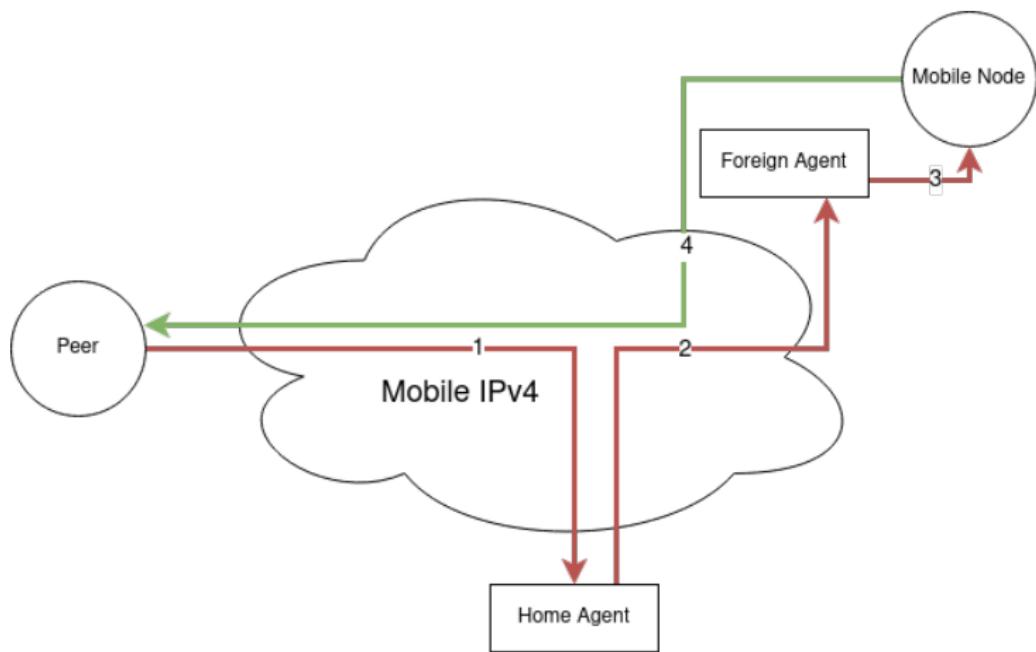
- Bisherige Architektur kritisch, wenn Objekte schnell ihre (physische) Adresse ändern können
 - ▶ Jedes Mal müssten die Einträge im Name Server geändert werden (Problem Replikation/Caching)
- Lösung:
 - ▶ Aufteilung in Naming und Location-Service
 - ▶ Abbildung: Name → unique Entity ID → Location
 - ▶ Nur ein Update erforderlich



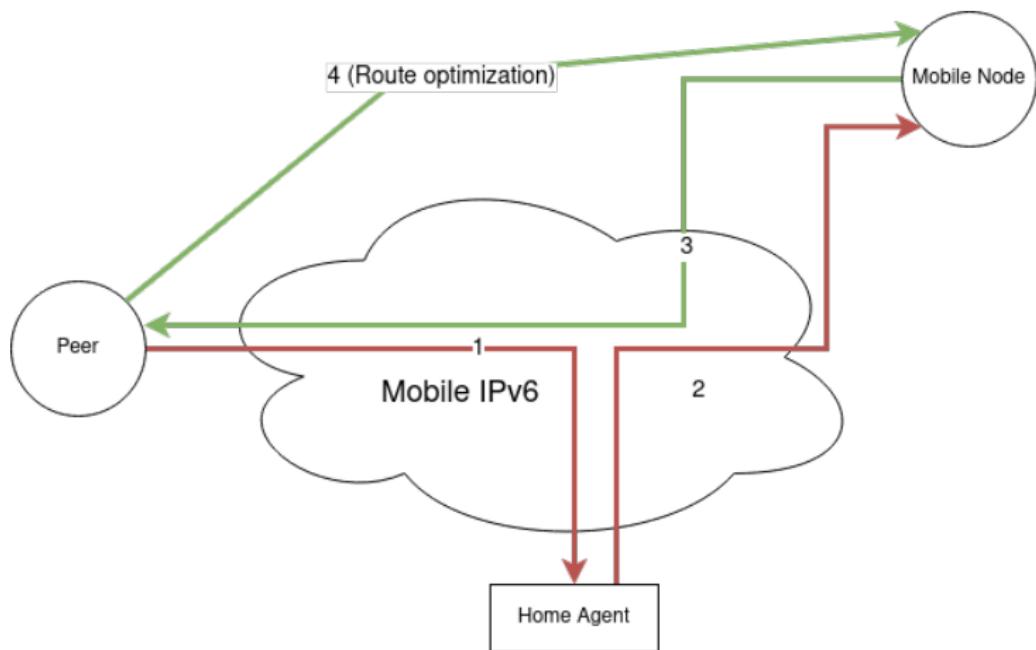
Identifier-Locator Split

- Jeder Teilnehmer benötigt zwei Bezeichner (Adressen)
 - ▶ Ein Identifier (ID) bezeichnet, **wer** er ist.
 - ▶ Ein Locator (Loc) bezeichnet, **wo** er ist.
- In statischen Netzen sind beide Bezeichner statisch:
 - ▶ Können sich deshalb auf eine Adresse reduzieren.
 - ▶ Im Internet ist die ‚reguläre‘ IP Adresse ID und Loc
- In mobilen Netzen ändert sich der Locator
 - ▶ ID und Loc divergieren: ID-Loc Split
 - ▶ Netzwerk und Endsysteme müssen Dualität handhaben

Mobile IPv4



Mobile IPv6



Zusammenfassung

- Namen können unique und pure/impure sein und beziehen ihre Bedeutung aus ihrem Kontext.
- Namensdienste dienen zur Auflösung von Namen zu Adressen.
- Verzeichnisdienste erweitern diesen Ansatz und ermöglichen eine Suche über weitere Eigenschaften.

6. Sicherheit



<https://blog.hackerrank.com/new-domain-tuesday-hackerrank-reveals-security-distributed-systems-domains/>

Inhalt

6. Sicherheit

- 6.1 Einführung
- 6.2 Verschlüsselungsverfahren
- 6.3 Kryptographische Hash-Funktionen
- 6.4 Authentifizierung
- 6.5 Digitale Signaturen
- 6.6 Schlüsselverwaltung
- 6.7 Protokolle und Anwendungen
- 6.8 Firewalls

Einführung

- Sicherheit

- ▶ Bedeutung hier nur im Sinne von Security
- ▶ nicht betrachtet: Sicherheit im Sinne von Safety

- Informationssicherheit (*)

- ▶ „Informationssicherheit hat zum Ziel, die Verarbeitung, Speicherung und Kommunikation von Informationen so zu gestalten, dass die *Vertraulichkeit, Verfügbarkeit* und *Integrität* der Informationen und Systeme in ausreichendem Maß sichergestellt wird. Zur Zielerreichung müssen verschiedene Teilaspekte integriert betrachtet werden.
- Informationssicherheit bezeichnet in diesem Zusammenhang das Ziel, diese Systeme vor Gefahren bzw. *Bedrohungen* zu schützen, *Schaden* zu vermeiden und Risiken zu minimieren“.
- ▶ Trennung von Strategie (Policy, Politik) und Mechanismus
 - ★ Sicherheitsstrategie (Mengen von Regeln)
 - ★ Sicherheitsmechanismen (Mechanismen zur Durchsetzung)

(*)<http://de.wikipedia.org/wiki/Computersicherheit>

Sichere Systeme

- ... gibt es nicht.
- Die absolut sichere Firewall:



<http://www.brauweisen-historisch.de/seitenschneider.jpeg>

Schutzziele

- Allgemeine Schutzziele (CIA):

- ▶ *Vertraulichkeit (Confidentiality):*
Information wird nur Berechtigten zugänglich
- ▶ *Integrität (Unversehrtheit, Integrity):*
Daten dürfen nicht unbemerkt verändert werden
- ▶ *Verfügbarkeit (Availability):*
Zugriff auf Daten ist mit vereinbarter Güte gewährleistet

- Weitere Schutzziele:

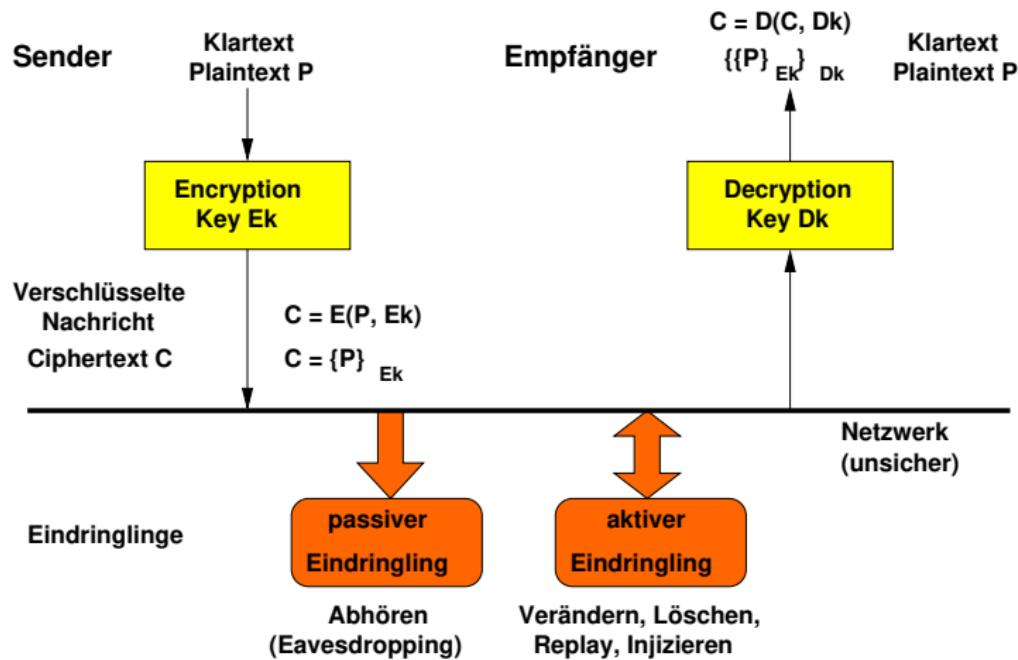
- ▶ *Authentizität (Authenticity):*
Echtheit einer Person oder eines Dienstes ist überprüfbar
- ▶ *Verbindlichkeit (Nicht-Abstreitbarkeit, Non-Repudiation):* Urheber von Daten muss erkennbar sein und kann dies nicht abstreiten
- ▶ *Zurechenbarkeit (Accountability):*
Eine Aktion kann einem Benutzer zugeordnet werden
- ▶ *Privatsphäre (Privacy):*
Personenmerkmale müssen vertraulich bleiben, und Anonymität muss möglichst gewahrt bleiben

Weitere Begriffe

- Authentisierung, Authentifikation (authentication):
 - ▶ Verifikation einer Identität
 - ▶ beidseitige Authentifikation von Kommunikationspartnern
notwendig: z.B. Benutzer - Rechensystem und umgekehrt
- Autorisierung (authorisation):
 - ▶ Ermächtigung: Rechte haben und wahrnehmen können
 - ▶ Security-Modelle
 - ★ Discretionary Access Control
Zugriffsmatrix als abstraktes Modell
Verfahren: Capabilities, Access Control Lists (ACLs)
 - ★ Mandatory Access Control
z.B. militärische Klassifizierung, Restriktionen im Informationsfluss

Weitere Begriffe (2)

- Kryptographie: Lehre von der Übertragung geheimer Nachrichten
- Grundmodell:



Bedrohungen

• STRIDE Modell

- ▶ **S**poofing ↵ Authenticity
- ▶ **T**ampering ↵ Integrity
- ▶ **R**eputation ↵ Non-repudiability
- ▶ **I**nformation disclosure ↵ Confidentiality
- ▶ **D**enial of Service ↵ Availability
- ▶ **E**levation of Privilege ↵ Authorization

Bedrohungen (Beispiele)

- Fehlerhafte Spezifikation von Sicherheitsstrategien
- Fehlerhaftes Design/Spezifikationen von Komponenten
- Fehlerhafte Konfiguration
- Fehlerhafter Code
- Schwache kryptographische Verfahren
- Ausnutzung von Insider-Wissen
- „Social Engineering“
- Lauschen / Abhören / Ausspähen (Eavesdropping)
- Denial-of-Service-Attacken
 - ▶ z.B. durch Erzeugung von Last
 - ▶ Verhinderung, ein vorhandenes Recht wahrnehmen zu können
- Diebstahl von Schlüsseln und Maskerade (Vorgeben einer anderen Identität)
- Aktives Verändern, Löschen, Wiederholen/Replay von Nachrichten
- Injizieren/Infiltrieren von Nachrichten, E-Mails, Viren, Würmern, Trojanischen Pferden, ...

Risikobewertung



- Konflikt zu anderen Charakteristiken der Software-Qualität.
- Aufwand/Nutzen müssen abgewogen werden.
- Pro Bedrohung:
 - ▶ Potentieller Schaden (Leib und Leben, Sachschaden, Image)
 - ▶ Wahrscheinlichkeit des Auftretens
 - ▶ Wahrscheinlichkeit der Erkennung des Auftretens
- Je höher das Risiko, desto wichtiger eine Berücksichtigung in der Sicherheitsstrategie.

Sicherheitsstrategie (Security Policy)

- Gesetzliche Vorgaben
 - ▶ Datenschutzgesetze
 - ▶ Anwendungsfeld-bezogene Gesetze
 - ★ Basel II (EU, Bankenbereich, Teil des Risiko-Managements)
 - ★ Sarbanes-Oxley Act (USA, Unternehmen, Verbesserung der Unternehmensberichterstattung, Forderung nach IT Governance)
 - ★ FDA Regulations (USA, Food and Drug Administration, starke Regulierung)
- Unternehmensorganisation, -prozesse
 - ▶ Identifikation von schützenswerten Vermögenswerte (Daten, Funktionen, Gegenstände)
 - ▶ Identifikation potenzieller Angreifergruppen
 - ▶ Identifikation der Angriffsvektoren
 - ▶ Ableiten von Schutzmaßnahmen (technische und nicht-technische)

Sicherheitsstrategie (Security Policy) (2)

Standards / Best Practices (Beispiele)

- ISO/IEC 27001:2005, „Information technology - Security techniques - Information security management systems Requirements“
 - ▶ spezifiziert Anforderungen für Herstellung, Einführung, Betrieb, Überwachung, Wartung, und Verbesserung eines dokumentierten Informationssicherheits-Managementsystems unter Berücksichtigung der Risiken innerhalb der gesamten Organisation
 - ▶ berücksichtigt sämtliche Arten von Organisationen (z.B. Handelsunternehmen, staatliche Organisationen, Non-Profit-Organisationen).
- ISO 17799:2005, „Information technology – Code of practice for information security management“
 - ▶ Kontrollmechanismen für die Informationssicherheit
 - ▶ 11 Überwachungsbereiche untergliedert in 39 Hauptkategorien, sogenannte Kontrollziele.
 - ▶ insgesamt 133 Sicherheitsmaßnahmen zur Unterstützung

Sicherheitsstrategie (Security Policy) (3)

Standards / Best Practices (Beispiele)

- IETF
 - ▶ Eigene Security Area mit ca. 20 Arbeitsgruppen
 - ▶ Spezifikationen zu TLS, SSH, IPSec ...
 - ▶ Alle RFCs enthalten Security Considerations
- Fast alle Standardisierungsgremien (IEEE, OMA, ...) betreiben Gremien zum Thema Security
- IT-Grundschutzhandbuch des Bundesamtes für Sicherheit in der Informationstechnik (BSI)
 - ▶ in Deutschland verbreitet
 - ▶ „Kochrezept“ für mittleres Schutzniveau
 - ▶ berücksichtigt neben Eintrittswahrscheinlichkeiten und potentieller Schadenshöhe auch Kosten der Umsetzung

Sicherheitsmechanismen

- Überwiegend mit kryptographischen Mechanismen:
 - ▶ Authentisierung
 - ★ von Systemen/Benutzern (entity authentication)
 - ★ von Datenpaketen (data origin authentication)
 - ▶ Integritätssicherung (integrity protection)
 - ★ häufig kombiniert mit Daten-Authentisierung
 - ▶ Verschlüsselung (encryption)
 - ▶ Schlüsselmanagement (key exchange)
 - ▶ ...
- Ohne kryptographische Mechanismen:
 - ▶ Zugriffskontrolle (access control)
 - ▶ Policy-Management
 - ▶ Einbruchserkennung (intrusion detection)
 - ▶ ...

Standardisierte Sicherheitskriterien

- Klassifizierung für sogenannte Sichere Systeme der Informationstechnik basierend auf Kriterienkatalogen.
- Kriterienkatologe finden häufig bei der Beschaffung „Sicherer IT-Systeme“ Anwendung.
- Ursprung US DoD „Orange Book“
 - ▶ TCSEC: Trusted Computer Systems Evaluation Criteria, 1983
- Deutschland:
 - ▶ Zuständigkeit heute: Bundesamt für Sicherheit in der Informationstechnik (BSI)
 - ▶ Deutsche IT-Sicherheitskriterien (Grünbuch), 1989
 - ▶ Konzept der Trennung von Funktionalität und Prüftiefe (Qualitätsstufe)

Sicherheitskriterien (2)

Europa:

- Information Technology Security Evaluation Criteria - ITSEC, 1990
- Vereinigung von TCSEC und ITSEC zu Common Criteria zur Bewertung
(Gegenseitige Anerkennung von Prüfzertifikaten), 1996
- Common Criteria Version 2.1 als weltweiter Standard ISO/IEC 15408
 - ▶ Teil 1: Introduction and General Model
 - ▶ Teil 2: Security Functional Requirements
 - ▶ Teil 3: Security Assurance Requirements
- Unterscheidung Funktionalität und Vertrauenswürdigkeit
- Vordefinierte Beispielklassen (10 Funktionalitätsklassen)
- Vertrauenswürdigkeit unterscheidet zwischen Korrektheit und Wirksamkeit (Stärke niedrig, mittel und hoch)
- Bewertung des Vertrauens in die Korrektheit durch sechs hierarchische Evaluationsstufen E1 (niedrig) bis E6 (formal verifiziert) definiert.

Sicherheitskriterien (3)

Große Zuordnung:

ITSEC	TCSEC	Bedeutung
-, E0	D	kein Schutz; unwirksam
F-C1, E1	C1	Schutz gegen absichtliche Verstöße einfacher Angreifer; einfache Sicherheit informelle Spec., Funktionstest, gezielte Angriffe
F-C2, E2	C2	Schutz gegen absichtliche Verstöße einfacher Angreifer; login-Mech., getrennte User-Daten, logging, informelle Detail-Spec.
F-B1, E3	B1	guter Schutz; Sicherheitsmodell, regelbasierte Schutzstufen, Analyse des Quellcodes bzw. des Hardwarelayouts
F-B2, E4	B2	guter Schutz; formales Sicherheitsmodell, sicherer Datenfluss bei Authentisierung Formales Sicherheitsmodell, semiformale Detailspezifikation
F-B3, E5	B3	sehr guter Schutz; Referenzmonitor-Eigenschaften, Detailspezifikation nachvollziehbar auf Quellcode abbildbar
F-B3, E6	A1	zur Zeit nicht zu überwinden formale Spezifikation und Verifikation

Material

- Viele Quellen, z.B.
 - ▶ Claudia Eckert: IT-Sicherheit. Konzepte, Verfahren, Protokolle, Oldenbourg-Verlag, 2003
 - ▶ J. Plate, J. Holzmann: Sicherheit in Netzen, Skript FH München
 - ▶ RSA Laboratories: RSA Laboratories' Frequently Asked Questions About Today's Cryptography, Version 4.1, 2000,
<http://www.rsasecurity.com/>

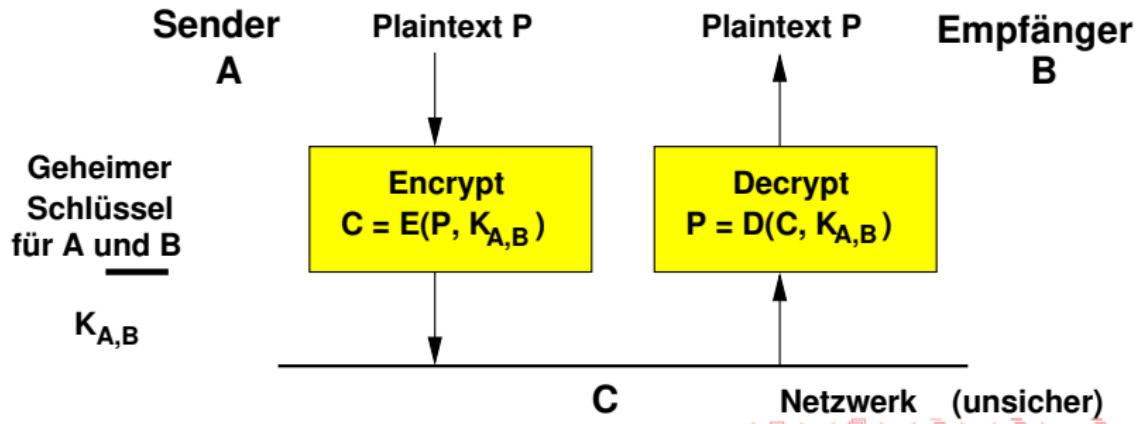
Verschlüsselungsverfahren

- Überblick
 - ▶ Eingeschränkte Algorithmen
 - ▶ Symmetrische Verfahren
 - ▶ Asymmetrische Verfahren

- Eingeschränkte Algorithmen
 - ▶ Geheimhaltung der Arbeitsweise
 - ▶ heute als unbrauchbar angesehen (historisch)
 - ▶ Beispiel: Caesar-Code: Verschieben um n Zeichen im Alphabet

Symmetrische Verfahren (private key encryption)

- ein geheimer Schlüssel für Ver- und Entschlüsselung
- sicherer Kanal zur Verteilung von Schlüsseln notwendig
- Vorteile:
 - ▶ kurze Schlüssel (ab mindestens 128 bit heute als brauchbar anzusehen)
 - ▶ geringer Rechenaufwand (schnell)
- Probleme:
 - ▶ Schlüsselaustausch und -verwaltung (Key Management)
 - ▶ keine Verbindlichkeit



Symmetrische Verfahren (2)

● Block-Algorithmen

- ▶ Verschlüsselung von Daten fester Länge, z.B. 64 Bit
- ▶ Alternativen:
 - ★ Electronic Code Book
 - alle Blöcke werden unabhängig voneinander verschlüsselt
 - ★ Cipher Block Chaining
 - Verschlüsselung hängt vom vorangehenden verschlüsselten Block ab (XOR)

● Strom-Algorithmen

- ▶ Bit/Byte-Strom-orientiert
- ▶ i.d.R. sehr schnell, kaum standardisiert

● Beispiele:

- ▶ DES Data Encryption Standard (US) historisch verbreitetster Vertreter
- ▶ Triple-DES, IDEA, AES
- ▶ RC4 (Strom-Algorithmus)

Asymmetrische Verfahren (public key encryption)

- Schlüssel besteht aus Paar (geheimer Schl., öffentlicher Schl.)
 - ▶ unterschiedliche Schlüssel für Ver- bzw. Entschlüsselung (daher der Name „asymmetrisch“)
 - ▶ Annahme: geheimer Schlüssel kann mit verfügbarem Rechenaufwand nicht aus öffentlichem Schlüssel und Verfahren rekonstruiert werden
- Vorteile:
 - ▶ kein sicherer Kanal zur Verteilung von Schlüsseln notwendig, geheimer Schlüssel wird nie übertragen
 - ▶ öffentliche Schlüssel leicht verteilbar (Verzeichnisdienst)
 - ▶ Verbindlichkeit erreichbar
- Probleme:
 - ▶ relativ lange Schlüssel notwendig
(ab 2048 bit heute als gut angesehen)
 - ▶ hoher Rechenaufwand
 - ▶ Vertrauenwürdiges Schlüsselmanagement

Asymmetrische Verfahren (2)

Vertreter

- RSA-Algorithmus
 - ▶ Rivest, Shamir, Adelman: 1978
 - ▶ basiert auf Primfaktorzerlegung großer Zahlen als schwieriges Einwegproblem
- Diffie-Hellman
 - ▶ Aufbau von sicheren Verbindungen aus einem unsicheren Zustand (ohne Authentifizierung)
- Elliptische Kurven (Elliptic Curve Cryptography – ECC)
 - ▶ anderes/neues math. Verfahren als Basis
 - ▶ geringere Schlüssellänge für gleiche Sicherheit
 - ▶ besonders geeignet für ressourcenbeschränkte Systeme

Typische Verwendung

- Asymmetrische Verschlüsselung (Public Key) für
 - ▶ Authentifizierung
 - ▶ Digitale Signaturen
 - ▶ Schlüsselmanagement
 - Symmetrische Verschlüsselung (Private Key) für
 - ▶ schnelle Verschlüsselung großer Datenmengen
- ⇒ Nutzung von asymmetrischen Verfahren, um Schlüssel für anschließende symmetrische Verschlüsselung auszuhandeln (Session Key)

Beispiel 1: DES (Data Encryption Standard)

- symmetrisch
- US-Standard
- Blockverschlüsselung (64 Bit)
- Schlüssellänge 56 Bit (+ 8 Bit Parity)
- Vorgehensweise
 - ▶ aus 56 Bit-Schlüssel Ableiten von 16 Schlüsseln der Länge 48 Bit
 - ▶ Permutation und inverse Permutation am Anfang und am Ende
 - ▶ 16 Verschlüsselungsrunden (Shift, XOR, Mischfunktion)
- schnelle Implementierung in Hardware
 - ▶ spezielle DES-Chips vorhanden
- gilt heute mit 56 Bit Schlüssellänge nicht mehr als sicher
- Ersatz: Triple DES (3DES)
 - ▶ Mehrfachdurchläufe von DES
 - ▶ effekt. Schlüssellänge 112 Bit
 - ▶ Nachfolger: AES
 - ▶ in Software relativ langsam

Beispiel 2: IDEA (Int. Data Encryption Algorithm)

- symmetrisch
- Ursprung: ETH Zürich, Fa. ASCOM (Schweiz), 1992
- Patent in Europa bis 2011,
für nicht-kommerzielle Zwecke frei verfügbar
- Blockverschlüsselung (64 Bit)
- Schlüssellänge 128 Bit
- 8 iterative Runden, 6 Teilschlüssel je Runde
- in Software effizienter zu berechnen als DES
- gilt als sicherer als DES

Beispiel 3: AES (Advanced Encryption Standard)

- symmetrisch
- Name: Rijndael
- Ursprung: Joan Daemen, Vincent Rijmen (Belgien)
- Neuer US-Standard, 2001:
 - Federal Information Processing Standard (FIPS) for the Advanced Encryption Standard, FIPS-197
- DES-Nachfolger
- Informationen: <http://csrc.nist.gov/encryption/aes/>
- Blockverschlüsselung
 - ▶ 3 unterstützte Blocklängen 128, 192 und 256 Bit
- relativ performant
 - ▶ in Software deutlich schneller als 3DES
 - ▶ Realisierung in Hardware möglich, AES-Koprozessoren
 - ▶ auch für Smartcards geeignet
- Verwendung in PGP v2.0 und OpenPGP

Beispiel 4: RC4

- symmetrisch
- Strom-Algorithmus (Byte-orientiert)
- Ursprung: Rivest (RSA Data Security)
- variable Key-Länge bis 2048 Bit
- schnell in Software
- wurde z.B. genutzt für Datei-Verschlüsselung und war in SSL/TLS wählbar
- seit 2015 für TLS verboten, da effiziente Angriffe bekannt geworden sind

Beispiel 5: RSA

- asymmetrisch
- Ursprung: Rivest, Shamir, Adleman, 1977
CACM Vol.21 No.2, Febr. 1978
- Blockverschlüsselung
- Schlüssellänge > 100 Dezimalstellen
z.B. RSA-129 (129 Dezimalstellen \Rightarrow 429 Bit)
- 1024 Bit = 105474-facher Aufwand
2048 Bit = $2.97 \cdot 10^{10}$ -facher Aufwand
- große Schlüssellängen bieten noch Schutz
- sehr hoher Rechenaufwand (ca. 100-1000 mal Aufwand DES)

RSA (2)

Mathematische Basis

- Primfaktorzerlegung großer Zahlen
- Bitkette eines Blocks m des Klartextes wird interpretiert als ganze Zahl $\leq n$
- Verschlüsselung $c = E(m, K^+) = E(m, (e, n)) = m^e \pmod{n}$
- Entschlüsselung $m = D(c, K^-) = D(c, (d, n)) = c^d \pmod{n}$
- für e, d und n muss gelten: $m^{e \cdot d} = m \pmod{n}$ für alle $m < n$
- Bestimmung von e und d :
 - ▶ Bestimme zwei große Primzahlen p und q
 - ▶ $n := p \cdot q, z := (p - 1) \cdot (q - 1)$
 - ▶ Wähle d relativ prim zu z
 - ▶ Bestimme e mit $e \cdot d = 1 \pmod{z}$
 - ▶ öffentlicher Schlüssel: (e, n)
 - ▶ geheimer Schlüssel: (d, n)

Kryptographische Hash-Funktionen

- Bildung eines digitalen Fingerabdrucks über Dokumenten/Nachrichten, genannt Message Digest
- Basis für digitale Signaturen
- Hash-Funktion H
 - ▶ $h = H(P)$
 - ▶ Nachricht P beliebiger Länge
 - ▶ h Bitkette fester Länge (z.B. 128 Bit)
 - ▶ vergleichbar zu CRC zur Fehlererkennung
- Annahmen
 - ▶ Berechnung von H ist einfach
 - ▶ Umkehrung, d.h. Ermittlung einer Ausgangsnachricht bei gegebenem Hashwert, ist schwierig (Einwegfunktion)
 - ▶ Veränderung am Dokument (P) führt zu anderem Hashwert (h)

Beispiel 1: MD5

- MD5 (Message Digest 5)

- ▶ Rivest, 1991
- ▶ 128-Bit Hashwert
- ▶ Vorgehensweise
 - ★ ausgelegt auf 32-Bit-Prozessoren
 - ★ basierend auf MD4 (gilt als nicht sicher)
 - ★ Zerlegung (bzw. Auffüllen) der Nachricht in Stücke von 448 Bit
 - ★ Anfügen der Gesamtlänge als 64 Bit-Zahl
(→ 512 Bit-Blöcke mit je 16 Unterblöcken der Länge 32 Bit)
 - ★ Ausgehend von einem konstanten Digest wird mit jedem neuen Block ein neuer Digest bestimmt (Phase)
 - ★ Jede Phase besteht aus 64 Iterationen mit 32-Bit-Funktionen über (AND, OR, NOT) über den Unterblöcken
- ▶ bekannt, früher von pgp verwendet
- ▶ erste Kollisionen bekannt
- ▶ gilt mittlerweile als unsicher

Beispiel 2: SHA-0, SHA-1, SHA-2, SHA-3

- SHA-1 (Secure Hash Algorithm)

- ▶ Weiterentwicklung vom SHA-0 („Konstruktionsfehler“, 1994)
- ▶ 160-Bit Hashwert
- ▶ Standard ANSI X9.30
- ▶ Berechnung etwas aufwendiger als MD-5
- ▶ Rechenintensive Angriffe bekannt

- SHA-2

- ▶ Weiterentwicklung vom SHA-1 (ab 2001) SHA-224, SHA-256, SHA-384 und SHA-512 (Hashwertlänge in Bit)
- ▶ 512/1024-Bit blockweise Verarbeitung, Dokumente bis 2128 Bit
- ▶ Gilt noch als sicher (NIST)

- SHA-3 (Keccak) (Bertoni, Daemon, Peeters, Van Assche / Italien, Belgien)

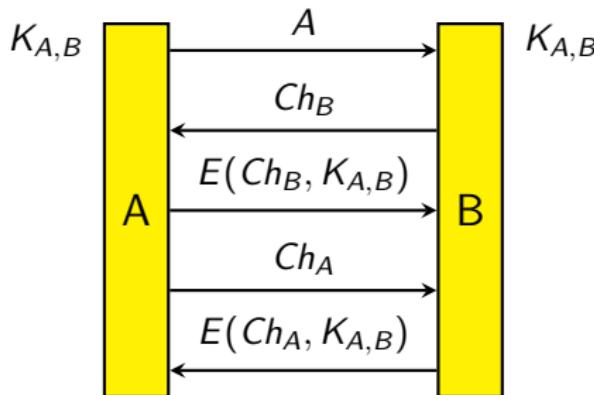
- ▶ Gewinner NIST-Ausschreibung (2012)
- ▶ Alternative zu SHA-2
- ▶ Modifikationen vom NIST vorgeschlagen (kritisch)

Authentifizierung

- Authentifizierung und Nachrichtenintegrität sind nicht voneinander trennbar
 - ▶ Was nützt Authentizität, wenn Nachricht verändert sein kann?
 - ▶ Was nützt Integrität einer Nachricht, wenn sie von jemand anderem kommen kann?
- Vorgehensweise
 - ① zuerst sicheren Kanal einrichten mit gegenseitiger Authentifizierung
 - ② dann geheimen Sitzungsschlüssel verwenden, um Integrität und Vertraulichkeit sicherzustellen

Authentifizierung bei geheimem Schlüssel

- Prinzip eines Challenge-Response-Protokolls



$K_{A,B}$: gemeinsamer geheimer Schlüssel

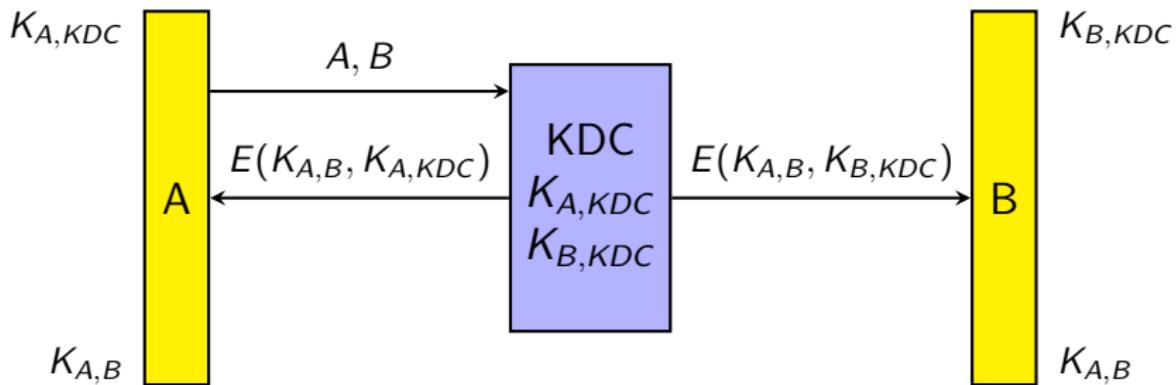
- Kommunikationswunsch A, enthält Identität A
- Challenge Ch_B (z.B. Zufallszahl) gestellt von B
- B kann überprüfen, ob Ch_B in Antwort enthalten (\rightarrow nur A kann Partner sein)
- analog für andere Richtung (\rightarrow nur B kann Partner sein)

- Problem: Verwaltung vieler geheimer Schlüssel

Authentifizierung bei geheimem Schlüssel (2)



- Problem (s.o.): Verwaltung vieler geheimer Schlüssel
- Lösung: Schlüsselverteildienst (Key Distribution Center, KDC)
- Prinzip
 - Jeder Teilnehmer hat geheimen Schlüssel mit KDC
 - KDC generiert geheimen Schlüssel $K_{A,B}$ für gewünschten Kanal und verteilt diesen Schlüssel an beide Partner

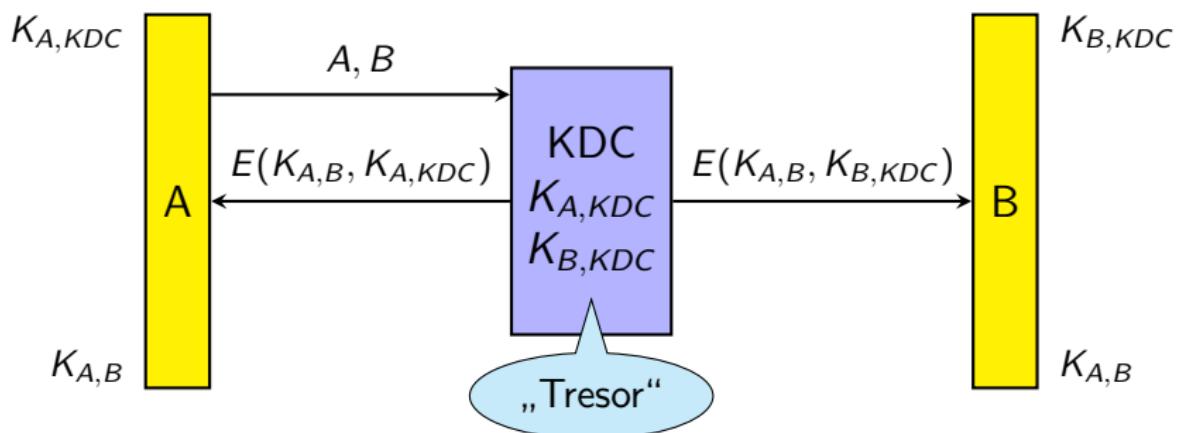


- Problem: Synchronisation zwischen A und B

Authentifizierung bei geheimem Schlüssel (2)



- Problem (s.o.): Verwaltung vieler geheimer Schlüssel
- Lösung: Schlüsselverteildienst (Key Distribution Center, KDC)
- Prinzip
 - ▶ Jeder Teilnehmer hat geheimen Schlüssel mit KDC
 - ▶ KDC generiert geheimen Schlüssel $K_{A,B}$ für gewünschten Kanal und verteilt diesen Schlüssel an beide Partner



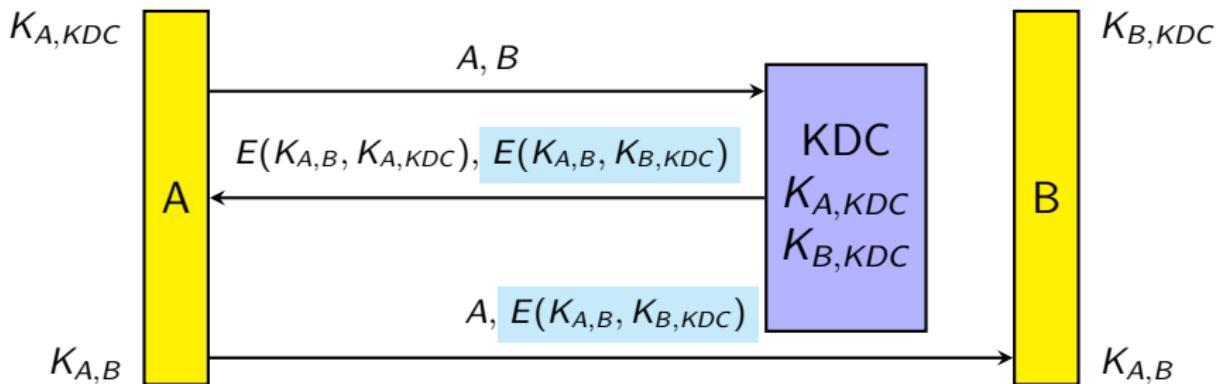
- Problem: Synchronisation zwischen A und B

Authentifizierung bei geheimem Schlüssel (3)



- Lösung:

- ▶ Schlüsselverteildienst (Key Distribution Center, KDC)
- ▶ Einführung von Tickets

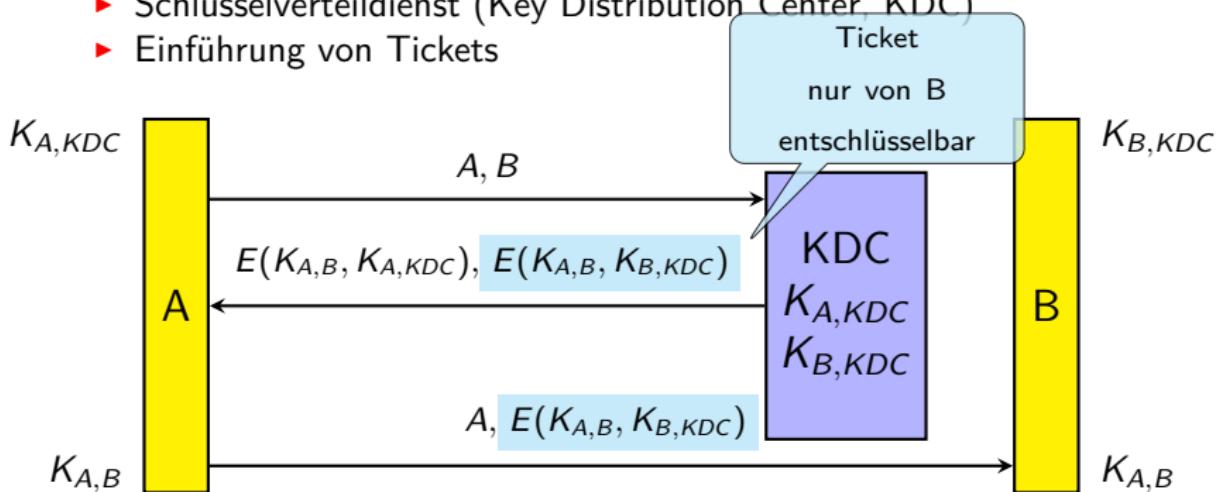


Authentifizierung bei geheimem Schlüssel (3)



- Lösung:

- ▶ Schlüsselverteildienst (Key Distribution Center, KDC)
- ▶ Einführung von Tickets

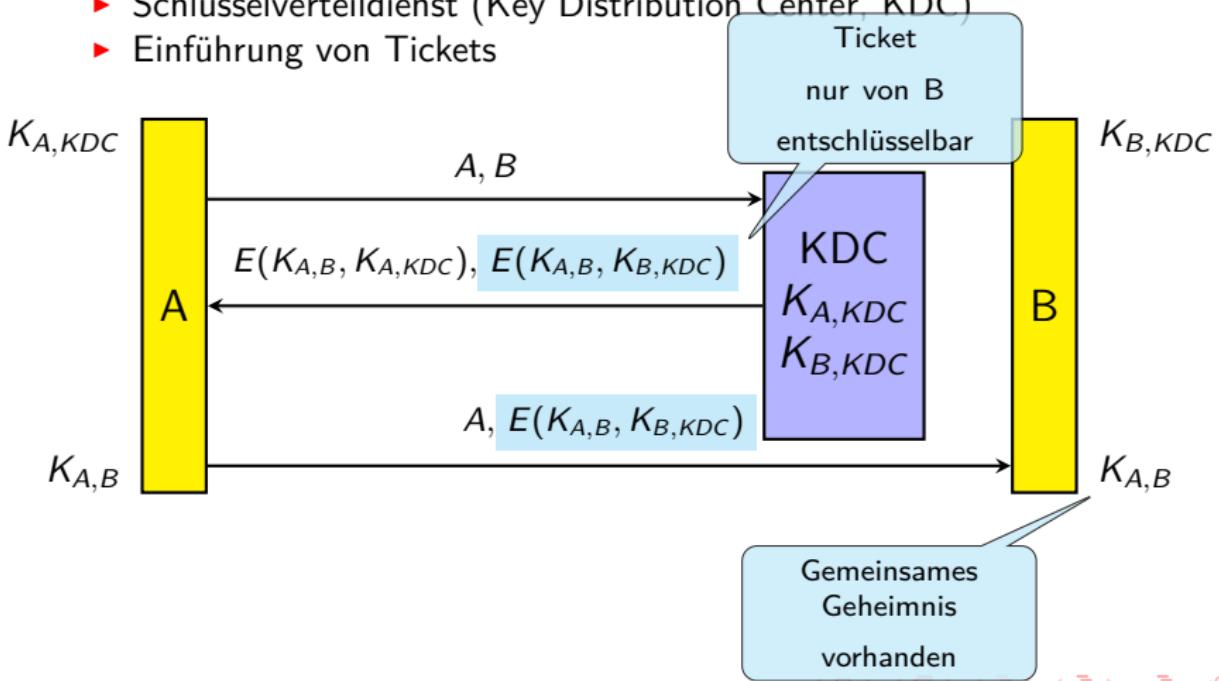


Authentifizierung bei geheimem Schlüssel (3)



- Lösung:

- ▶ Schlüsselverteildienst (Key Distribution Center, KDC)
- ▶ Einführung von Tickets

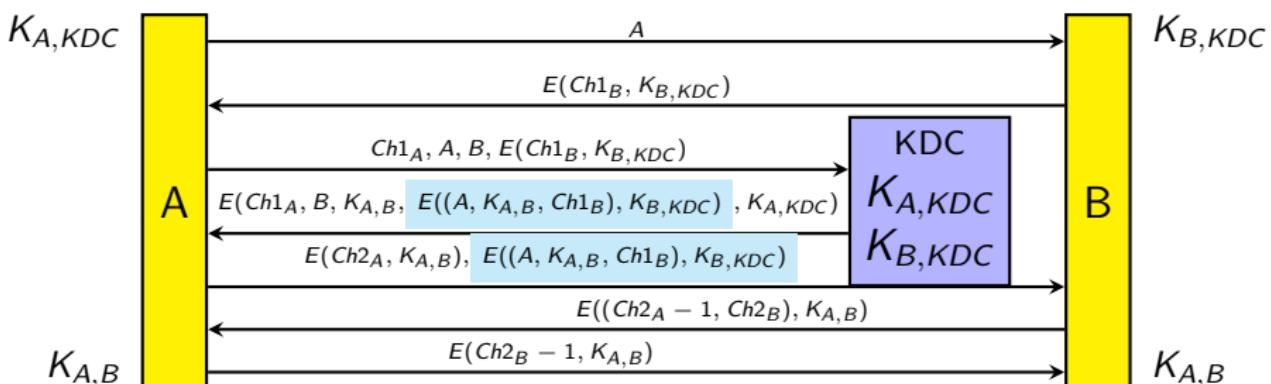


Authentifizierung bei geheimem Schlüssel (4)



- Weiterentwicklung: Needham-Schroeder-Protokoll

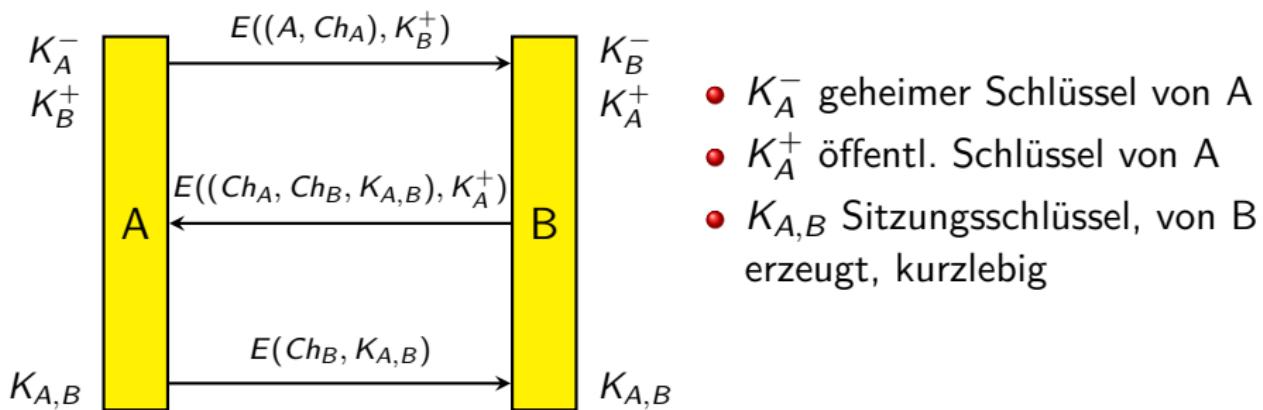
- ▶ Needham, Schröder, 1978
- ▶ Sicherungen gegen wiederholtes Einspielen von Nachrichten
- ▶ Variation dieses Protokolls in Kerberos verwendet (vgl. 6.7)



Authentifizierung mit öffentl. Schlüssel

• Prinzip

- kein KDC erforderlich
- Zuordnung der öffentlichen Schlüssel zu den wahren Personen muss gewährleistet sein

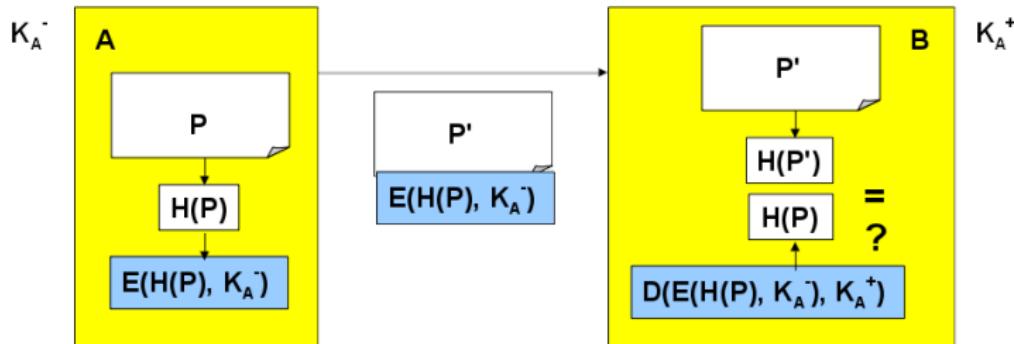


Digitale Signaturen

- Bedeutung wie Unterschrift
 - ▶ nicht vom unterschriebenen Dokument zu trennen
 - ▶ nicht (leicht) fälschbar
- Signatur bietet zuverlässige Feststellung von
 - ▶ Urheberschaft
 - ▶ Nichtabstreitbarkeit
 - ▶ Integrität
 - ▶ Authentizität
- schützt nicht Vertraulichkeit des Inhalts
 - ▶ dazu ist zusätzlich Verschlüsselung notwendig (s.u.)
- Kombination aus
 - ▶ Hash-Algorithmus
 - ▶ Public Key-Infrastruktur
- Europa besitzt bzgl. digitalen Signaturen relativ starke Stellung

Vorgehensweise

- Signieren durch Verschlüsselung des Hash-Werts einer Nachricht mit privatem Schlüssel
- Öffentlicher Schlüssel dient auf Empfänger-Seite zur Überprüfung der Signatur



Verfahren

Ablauf

- ① Teilnehmer A (Sender) bildet über Klartext P mit Hash-Alg. H einen Hashwert $V_A = H(P)$
- ② Teilnehmer A chiffriert Hashwert V_A mit seinem privaten Schlüssel K_A^- (Vorteil Zeitersparnis)

$$VC_A = E(V_A, K_A^-) \text{ (=Signatur)}$$

- ③ Chiffrierter Hashwert wird der (unverschlüsselten) Nachricht angehängt und mit übertragen
- ④ Teilnehmer B (Empfänger) dechiffriert VC_A mit dem öffentlichen Schlüssel K_A^+ von A

$$V = D(VC_A, K_A^+)$$

- ⑤ Neuermittlung des Hashwerts der Nachricht P:

$$V_B = H(P)$$

- ⑥ $V = V_B$?
falls ja: Signatur echt und Nachricht unverändert

Schlüsselverwaltung

- Ziel

- ▶ Sicheres und effizientes Life Cycle Management von Schlüsseln
 - ★ Erzeugen/Einrichten
 - ★ Verteilen
 - ★ Ungültig erklären (Revocation)
- ▶ Vertrauen in Schlüsselverwaltung notwendig!

- Verschiedene Vorgehensweisen

- ▶ bei Umgang mit geheimen Schlüsseln
(Key Distribution Center, KDC)
- ▶ bei Umgang mit öffentlichen Schlüsseln
(Public Key Infrastructure, PKI)
- ▶ alles andere als trivial!

Beispiel für KDC-Ansatz: Kerberos

- Grundlage: Umgang mit geheimen Schlüsseln
- Ursprung: MIT
- basiert auf Needham-Schroeder-Authentifizierung
- erweitert um Zeitstempel
 - ▶ Ticket nur in bestimmtem Zeitintervall gültig
- KDC aufgespalten in
 - ▶ Authentication Server (AS)
 - ▶ Ticket Granting Server (TGS)
- Einsatz
 - ▶ Andrew File System
 - ▶ OSF DCE
 - ▶ Microsoft Active Directory

Kerberos (2)

Annahmen:

- Netzwerk unzuverlässig
- Security Server (Rechner für AS und TGS) ist sicher
 - ▶ in sicherem Raum (durch Kerberos bewacht)
 - ▶ kein Eindringling kann Manipulationen vornehmen
 - ⇒ darf geheime Schlüssel aller Benutzer speichern
- Uhren im verteilten System sind „einigermaßen“ synchron
- Benutzer vergessen Passwörter nicht

Kerberos (3)

Ziele:

- gegenseitige Authentifizierung
- zeitlich befristete Gültigkeit von Schlüsseln, um Schaden zu begrenzen, falls Schlüssel bekannt werden sollte
- Passwörter nie im Klartext auf dem Netzwerk oder auf normalen Servern
- Passwörter auf Client-Rechnern im Klartext nicht länger als einige Mikrosekunden (ständen sonst im Core Dump)

PKI-Systeme

- PKI: Public Key Infrastructure
- Grundlage: Umgang mit öffentlichen Schlüsseln
- Wesentliches Problem
 - ▶ Sichere Verteilung der öffentlichen Schlüssel
 - ▶ Man-in-the-Middle-Attacke beim Schlüsselaustausch möglich
- Basis
 - ▶ Zertifikate
 - ★ Echtheit (Authentizität) öffentlicher Schlüssel
 - ▶ Verzeichnisdienste
 - ★ Auffinden öffentlicher Schlüssel
 - ★ z.B. LDAP

Zertifikate

- Zertifikate
 - ▶ dienen der Bestätigung der Echtheit eines öffentlichen Schlüssels
 - ▶ d.h. der Zugehörigkeit zu einer bestimmten Entität,
wie Person, Dienst, ...

- Zertifizierungsstelle (Certification Authority, CA)

- ▶ bezeichnet ausstellende Instanz
- ▶ Garant der Zuordnung Schlüssel-Person
- ▶ Vertrauenswürdigkeit vorausgesetzt oder öffentliche Schlüssel der Zertifizierungsstellen selbst zertifiziert durch übergeordnete CA
- ▶ von zentraler Instanz (Root CA) überwacht, die die öffentlichen Schlüssel der Zertifizierungsstellen zertifiziert (Vertrauenskette)

- Sperrliste (Certification Revocation List, CRL)

- ▶ identifiziert Seriennummern ungültig gewordener Zertifikate

X.509-Standard für Zertifikate

- Versionen: v1-v3, umfassend X.509v3
- Wesentliche Informationen in einem Zertifikat:
 - ▶ Version
 - ▶ öffentlicher Schlüssel des Zertifikatinhabers
 - ▶ Name des Inhabers (Distinguished Name)
 - ★ Common Name, CN, (Name der Person)
 - ★ Organization, O, (Firma oder Organisation)
 - ★ Organizational Unit, OU, (Abteilung oder Firmenteil)
 - ★ Locality, L, (Stadt, Sitz der Organisation)
 - ★ State, ST, (Staat, Provinz, Gegend)
 - ★ Country, C, (ISO Ländercode)
 - ▶ Name und Land der ausstellenden CA (Distinguished Name)
 - ▶ Gültigkeitszeitraum
 - ▶ verwendete Algorithmen
 - ▶ Extensions

Beispiel: CAs nach Deutschem Signaturgesetz (SigG)

Hochschule RheinMain

- SigG seit 1997, akt. Fassung 2001, zuletzt geändert Juli 2009
- Ziel: fortgeschrittene elektronische Signatur basierend auf qualifizierten Zertifikaten (mit geforderten Informationen)
- SigG bestimmt die Regulierungsbehörde für Telekommunikation und Post (Reg TP) als Wurzelinstanz aller CAs (Reg TP umbenannt in Bundesnetzagentur seit 2005).
- Seit 2007 Verwendung längerer Schlüssel RSA 2048, SHA-512

Beispiel: CAs nach Deutschem Signaturgesetz (SigG)

- Zertifizierungsdiensteanbieter (ZDA) Deutschland (Mai 2020):
 - ▶ Derzeit 13 (vgl. <http://www.bundesnetzagentur.de/>)
 - ★ 1&1 De-Mail GmbH
 - ★ Atos Information Technology GmbH
 - ★ Bank-Verlag GmbH
 - ★ Bundesagentur fuer Arbeit
 - ★ Bundesnetzagentur
 - ★ Bundesnotarkammer
 - ★ D-Trust (Tochter der Bundesdruckerei)
 - ★ DGN Deutsches Gesundheitsnetz Service GmbH
 - ★ Deutsche Telekom AG
 - ★ T-Systems International GmbH
(1994, erstes Trustcenter)
 - ★ exceet Secure Solutions GmbH
 - ★ Deutsche Post AG
 - ★ medisign GmbH
 - ▶ zahlreiche nicht mehr tätige oder untersagte ZDAs

Protokolle und Anwendungen

- hier nicht betrachtet:
 - ▶ Link-Layer Security
 - ▶ sicherheitsbezogene Protokolle auf der Netzwerkebene, wie IPsec
 - ▶ darauf aufbauende Architekturen wie Virtual Private Networks (VPN)
sichere Verbindung von Teilnetzen über unsichere Netze

Sicherheit in Anwendungsprotokollen

Beispiele

- S/MIME (Secure / Multipurpose Internet Mail Extensions)
 - ▶ RFC 2633
 - ▶ Beispiel für Verschlüsselung auf Anwendungsebene
 - ▶ Standard zur Verschlüsselung und Signatur
 - ▶ Kann mit TLS kombiniert werden
 - ▶ basiert auf X.509-Zertifikaten
 - ▶ Benötigt Unterstützung im Mail-Client
- SSH, SSH2
 - ▶ sicherer entfernter Zugriff zu SSH-Server als Ersatz für telnet, rlogin
 - ▶ verschiedene Formen der Authentifizierung und Verschlüsselung möglich (RSA; DES, 3DES, Blowfish)
 - ▶ Tunneln anderer Anwendungen (z.B ftp, Basis für sftp)
 - ▶ hohe praktische Bedeutung

Transport Layer Security (TLS)

- früher: Secure Socket Layer (SSL)
 - ▶ SSL 3.1 = TLS 1.0
 - ▶ TLS 1.3: RFC 8446 (2018)
- Sicherheit auf **Transportebene**
 - ▶ TLS-Protokoll liegt zwischen der Transportschicht und der Anwendungsschicht
 - ▶ Transparenz für alle höheren Anwendungsprotokolle gegeben (HTTP, SMTP, IIOP, ...)
 - ▶ hohe praktische Bedeutung
- Ursprung
 - ▶ Netscape, Einsatz in Browser als Ersatz für unsichere 40-Bit-Verschlüsselung
- Funktionalität
 - ▶ Authentifizierung und Verschlüsselung
 - ▶ Basis:
 - ★ Public Key-Zertifikate nach X.509
 - ★ symmetrische Verschlüsselung mit geheimen Session-Schlüsseln

TLS (2)

Teilprotokolle

- Handshake-Protokoll:
 - ▶ Server-Authentification
 - ★ Server antwortet auf Client-Request mit Zertifikat und Präferenzen für Verschlüsselungsverfahren (RC4, IDEA, DES, 3DES, ...)
 - ★ Client erzeugt Master Key, verschlüsselt diesen mit öffentl. Schlüssel des Servers aus Zertifikat, sendet verschl. Master Key und gewähltes Verfahren
 - ★ Server ermittelt Master Key und authentifiziert sich gegenüber Client mit Master Key verschlüsselter Nachricht
 - ★ anschließend werden Keys benutzt, die aus Master Key abgeleitet sind
 - ▶ Optionale Client-Authentification
 - ★ Server schickt Challenge-Anfrage an Client
 - ★ Client antwortet mit signierter Anfrage und Client-Zertifikat
- Change Cipher Spec Protocol
- Alert Protocol: Fehler-Behandlung
- Application Data Protocol (für Anwendungsdaten)
- Record-Protokoll: Codierung und Transfer (untere Schicht, unmittelbar auf TCP, symm. Verschlüsselung u.a. DES, TripleDES, AES)

Verbreitete Implementierungen: OpenSSL, GnuTLS, LibreSSL, ...

TLS (3)

- TLS benötigt einen zuverlässigen Transportdienst (üblicher TCP)
- für UDP-Kommunikation steht analog Datagram Transport Layer Security (DTLS) zur Verfügung
besitzt hohe Relevanz für IoT-Anwendungen

● Beispiele

- ▶ HTTPS
 - ★ HTTP over TLS, https:// ...
 - ★ Standard in Browsern
 - ★ Aufbau einer SSL-gesicherten Transportverbindung
 - ★ HTTP nutzt diese Verbindung zur geschützten Übertragung von vertraulichen Daten
 - ★ Port 443 statt 80
- ▶ SMTPS
- ▶ IMAPS, POP3S
- ▶ FTPS

Firewalls

- Ziele

- ▶ Monitoring aller eingehenden (und ausgehenden) Nachrichten
- ▶ Eindringlinge verhindern
- ▶ autorisierten Zugriff erlauben
- ▶ möglichst geringe Performance-Einbußen

- Annahme

- ▶ Firewall ist selbst sicher und nicht angreifbar

- Klassifikation

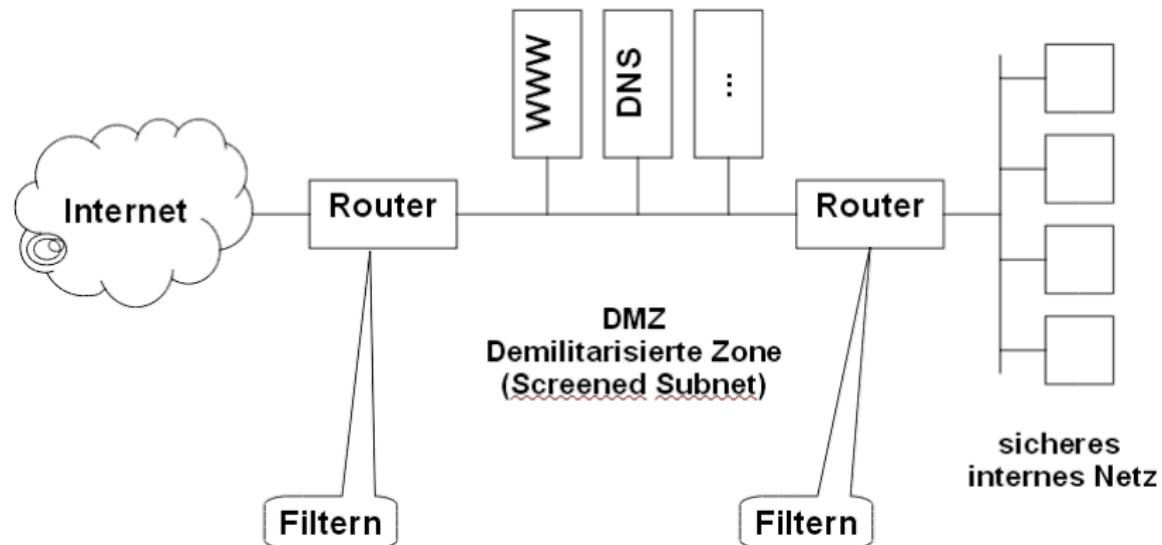
- ▶ nach Ebene, auf der Kontrollen stattfinden:
- ▶ Router-basiertes Filtern (Packet Filter, Screening)
- ▶ Application Level Gateways
- ▶ häufig kombiniert

Packet Filter

- Netzwerk-bezogen
 - ▶ Betrachtung auf Paketebene
- typischerweise in Routern realisiert
- Regeln für das Nichtweiterleiten
 - ▶ Sperren von Subnetzen
 - ▶ Sperren von Rechnern
 - ▶ Sperren von Diensten
 - ▶ basierend auf IP-Adressen u. Portnummern
- Vorteil
 - ▶ geringer Overhead (=hohe Performance)
- Nachteil
 - ▶ komplexe, nichtmodulare Regeln bei großen Netzen
 - ▶ i.d.R. kein Logging
- Beispiel: iptables

Packet Filter (2)

Architekturebene

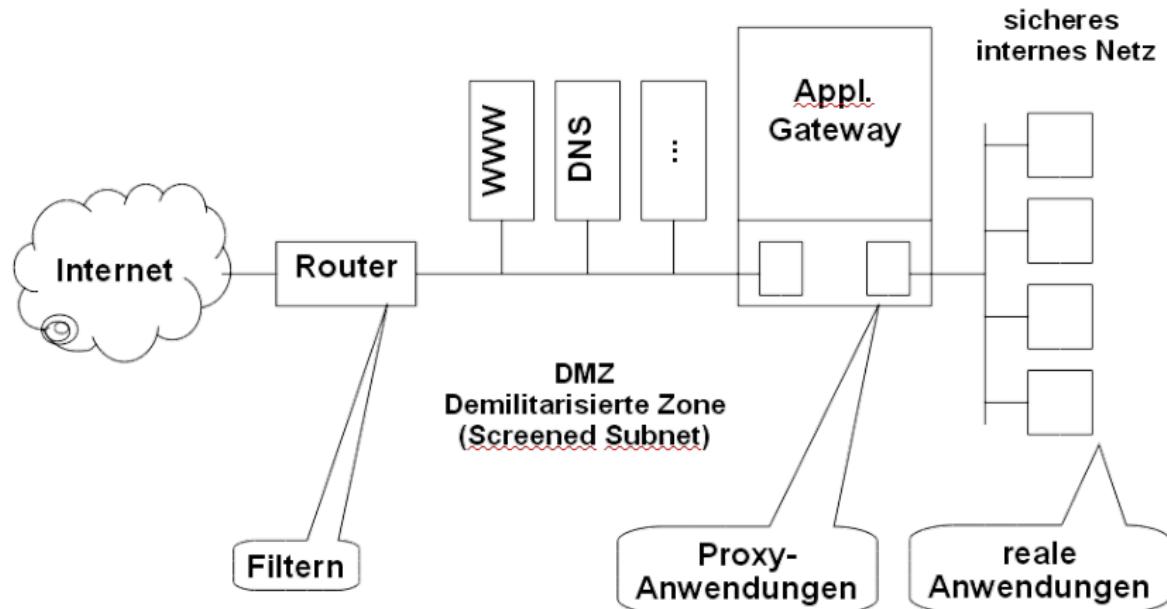


Application Level Gateway

- Sammlung spezieller Proxy-Programme als Ersatz der üblichen Anwendungen
- Üblich für HTTP, FTP, SMTP, X-Protokoll, ...
- Proxy-Anwendungen haben i.d.R.
 - ▶ Zugangsüberprüfung
 - ▶ Logging
- Vorteil: hoher Sicherheitsgrad
- Nachteile:
 - ▶ Notwendigkeit von Proxies
 - ▶ Nachziehen bei neuen Anwendungen
 - ▶ Hoher Performance-Overhead

Application Level Gateway (2)

Architekturebene



Zusammenfassung

- Vollständige Sicherheit (Security) gibt es nicht und ist immer ein Kompromiss.
- Sicherheitsmaßnahmen werden oft durch kryptographische Verfahren umgesetzt.
- Verschlüsselung und Authentifizierung sind zentrale Bestandteile jeden Sicherheitskonzeptes.

7. Globale Zeit / Uhrensynchronisation



http://www.bikersnews.de/motorrad/test+und+technik/im+sog_162.html

Inhalt

7. Globale Zeit / Uhrensynchronisation

7.1 Einführung

7.2 Zeitbegriff und Zeitsysteme

7.3 Rechneruhren

7.4 Synchronisationsprotokolle

7.5 Logische Zeitmarken

Einführung

Probleme mit nicht-synchronisierten Rechneruhren

- Zeitstempel von Dateien (make)
- zeitgesteuertes Ausführen von Aufträgen (cron)
- TDMA-basierte Medienzugriffsverfahren

Motivation

- Etablierung von systemweiter Zeit in verteilten Systemen
- Synchronität mit realer Außenzeit

Zielsetzung

- Synchronisation von Rechneruhren

Einführung (2)

Anwendungen

- korrekte Funktion zeitbezogener lokaler und verteilter Anwendungen
- korrektes Ordnen von Ereignissen in verteilten Systemen
- Leistungsmessung in verteilten Systemen
- verteilte Echtzeitsysteme einschl. Synchronität mit realer Außenzeit

Zeitbegriff und Zeitsysteme

Verschiedene Zeitbegriffe im Laufe der Geschichte Astronomische Zeit

- basiert auf der gleichförmigen Bewegung von Himmelskörpern und deren Beobachtung
- Sonnenzeit
 - ▶ mittlere Dauer einer Erdumdrehung
 - ▶ Sonnentag: Zenit-Zenit (bis 1956)
 - ▶ $1 \text{ Sek} = \frac{1}{24 \cdot 60 \cdot 60} \text{ Sonnentag}$
 - ▶ wenig stabil (Abbremsung der Erdrotation, Schwankungen durch Massenverlagerungen)
- Sternzeit
 - ▶ mittlere Dauer der Umlaufzeit der Erde um die Sonne
 - ▶ $1 \text{ Sek} = \frac{1}{31.556.925,9747} \text{ Teil des trop. Jahres 1900 (ab 1957)}$

Physikalische Zeit

basiert auf (periodischen) physikalischen Prozessen klassische Beispiele:

- Kerzenuhr (Verbrennen von Wachs)
- Pendeluhr, Genauigkeit best: 10^{-7}
- Quarzuhr, Genauigkeit best: 10^{-9} , typisch: $10^{-5} \dots 10^{-6}$

Atomuhr

- Definition im SI-Einheitensystem (ab 1967):
„Die Sekunde ist das 9.192.631.770fache der Periodendauer der dem Übergang zwischen den beiden Hyperfeinstrukturniveaus des Grundzustandes von Atomen des Nuklids ^{133}Cs entsprechenden Strahlung.“
- Cäsium-133-Uhr, Genauigkeit best: 10^{-14} , typisch: 10^{-13} ($< 1\mu\text{s}$ pro Jahr)
- Caesium-Fontäne, Genauigkeit $< 10^{-15}$

Physikalisch-Technische Bundesanstalt (PTB)



- in Braunschweig
- Betrieb mehrerer Atomuhren (CS1-CS4, CSF1)
- Verantwortung für die gesetzliche Zeit in D (ab 1978)
- Betrieb von Verteildiensten



<https://www.meinberg.de/images/xatomuhr.jpg.pagespeed.ic.3l8wJGqj54.jpg>

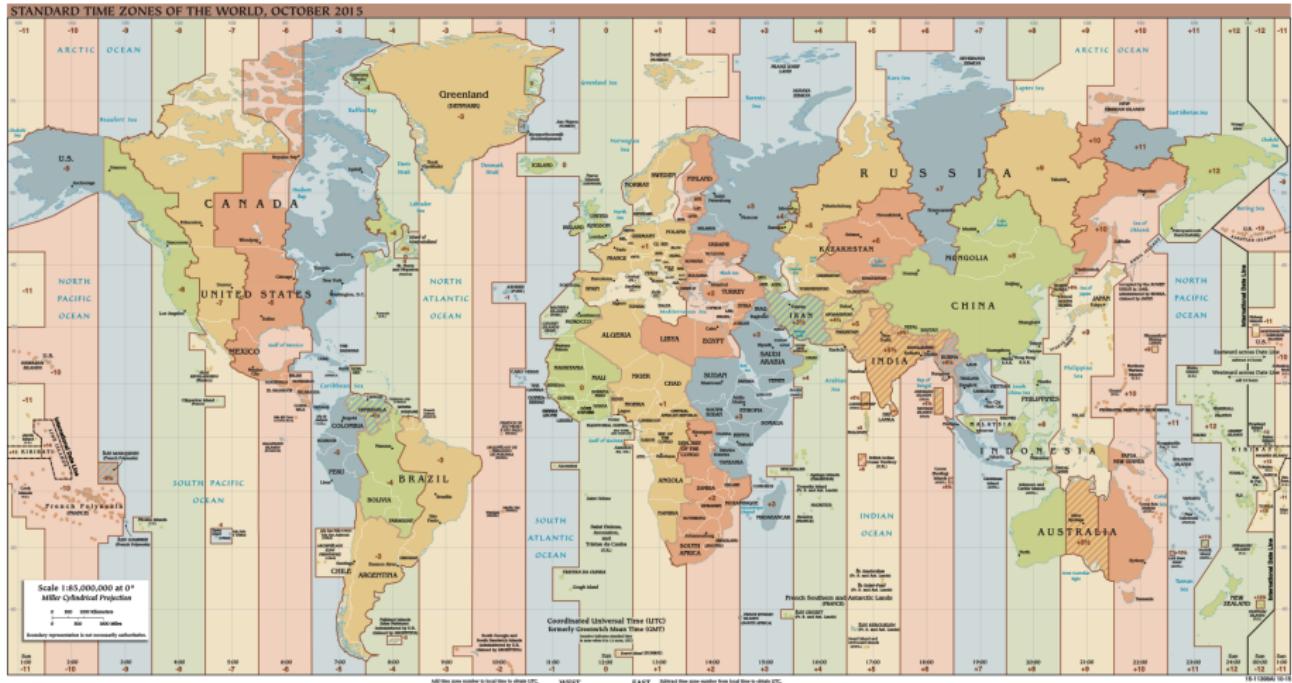
Foto: PTB

Zeitsysteme

GMT: Greenwich Mean Time

- Lokale Ortszeitangaben (wahre und mittlere) üblich bis ca. 1880
- Probleme für Eisenbahn-Fahrpläne
- „Greenwich Mean Time“ gesetzliche Standardzeit in England ab 1880
- Ab 1.06.1891: deutsche und österreichisch/ungarische Eisenbahnverwaltungen führen die Zeit des 15. Längengrads als *mitteleuropäische Eisenbahn-Zeit (M. E. Z.)* ein.
- Deutsches Reich: gesetzliche Uhrzeit ab 1.04.1893 ist „die mittlere Sonnenzeit des fünfzehnten Längengrades östlich von Greenwich“
- Meridiankonferenz Washington 1884 definiert Greenwich als Null-Meridian und führt Zeitzonen ein →GMT Sonnenzeit
- ab 1.1.1925 Beginn des Tags um Mitternacht (für Astronomen bis dahin mittags)

Zeitzonen



Zeitsysteme (2)

UT: Universal Time

- Weltzeit abgeleitet aus Sternzeit (ab 1957) am Null-Meridian
- UT1: Berücksichtigung der Polschwankungen

TAI: Temps Atomique International

- mittlere Atomzeit seit 1.1.1958
- Betrieb von ca. 250 Atomuhren weltweit
- weltweit koordiniert durch Bureau International de l'Heure (BIH)

UTC: Coordinated Universal Time

- heutiger Zeitstandard (ab 1972)
- basiert auf TAI, aber Anpassungen an UT1 durch „Schaltsekunde“ bei mehr als 900 ms Unterschied
- Abweichung: 1 Sek in 300.000 Jahren

Zeitverteildienste

Historisch

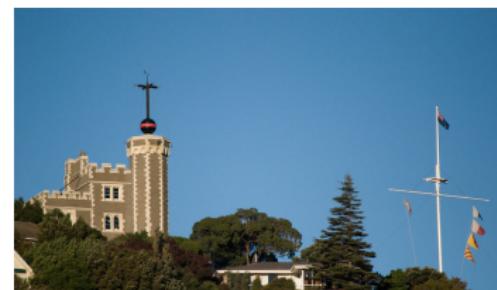
- One O'Clock Gun



https://upload.wikimedia.org/wikipedia/commons/d/d9/One_O'Clock_Gun.JPG

- ▶ **Problem:**
Schalllaufzeit

- Zeitball



<https://www.flickr.com/photos/fliessphil/2530153597/>

- ▶ **Problem:** Sichtverbindung
notwendig

Zeitverteildienste (2)

Langwellen-Radiosender

- z.B. in D: DCF77 (77.5 kHz, Frankfurt/Mainflingen)
- basierend auf Atomuhr CS-2 der PTB
- Sekudentakt
- aufmodulierter voller BCD-Zeitcode (58 Bit) in jeder Minute
- Genauigkeit
 - ▶ $2 \cdot 10^{-13}$ gemittelt über 100 Tage
 - ▶ 1-10 msec je Sek. (atmosphärische Störungen)

GOES Satellitensystem

- Geostationary Operational Environment Satellite
- Genauigkeit ca. 0.5 msec
- Dienst der NOAA von 1974 bis 2004

Zeitverteildienste (3)

GPS-Satellitensystem als Basis

- Global Positioning System, primär militärisch
- 24 Satelliten, Umlaufzeit 12 h, mind. 4 jederzeit „sichtbar“
- Cäsium-Uhren an Bord
- Synchronisation gegenüber Uhren anderer Satelliten durch Bodenstation auf ± 5 ns genau
- Standortbestimmung durch Unterschiede in Signallaufzeiten ($5\text{ns} \cong 1.5\text{m}$ mil.; $1\mu\text{s} \cong 300\text{m}$ zivil)
- künstliche Ungenauigkeiten in Krisenzeiten
- Differentielles GPS nutzt zusätzlich Bodenstationen mit bekannten Standorten (Geodäsie)

GPS-basierte Uhr

- GPS-Signal als Referenz einer PLL-Schaltung
- hochgenaue Sekundenimpulse (pps pulse-per-second)
- typ. Genauigkeit: ca. $1\mu\text{s}$

Zeitverteildienste (4)

Galileo-Satellitensystem der EU/ESA

- europäisches, zu GPS kompatibles System (GPS III)
- bis zu 30 Satelliten
 - ▶ mit je 2 Atomuhren
 - ▶ senden Zeitsignal und Positionsdaten
 - ▶ globale Abdeckung
- Dienste
 - ▶ kostenloser Dienst für Ortung, Navigation, Zeitsynchronisation (Genauigkeit ca. 4 m horizontal, 8 m vertikal)
 - ▶ kommerzieller Dienst (Genauigkeit 1 m, Bewegungen 0.2 m/sec) (Vermessungswesen, Netzsynchronisation, Flottenmanagement)
 - ▶ Safety-of-Life-Dienst, (sicherheitskritische Anwendungen in Luft- und Schifffahrt, Bahnverkehr)
 - ▶ Dienst „von öffentlichem Interesse“, (Signal mit sehr hoher Genauigkeit, Qualität, Zuverlässigkeit und Integrität für hoheitliche Anwendungen)



Andere Systeme: GLONASS (Russland) und Beidou (China)

Begriffe

Referenzzeit

- Approximation der wahren physikalischen Zeit

Abweichung, Genauigkeit (Accuracy)

- absolute oder relative Differenz zu einer Referenzzeit

Präzision, Auflösung (Precision)

- kleinste Zeitdauer zwischen zwei aufeinander folgenden anzeigbaren Zeitpunkten

Stabilität (Stability)

- Frequenzschwankung einer Uhr (Angabe oft in ppm)

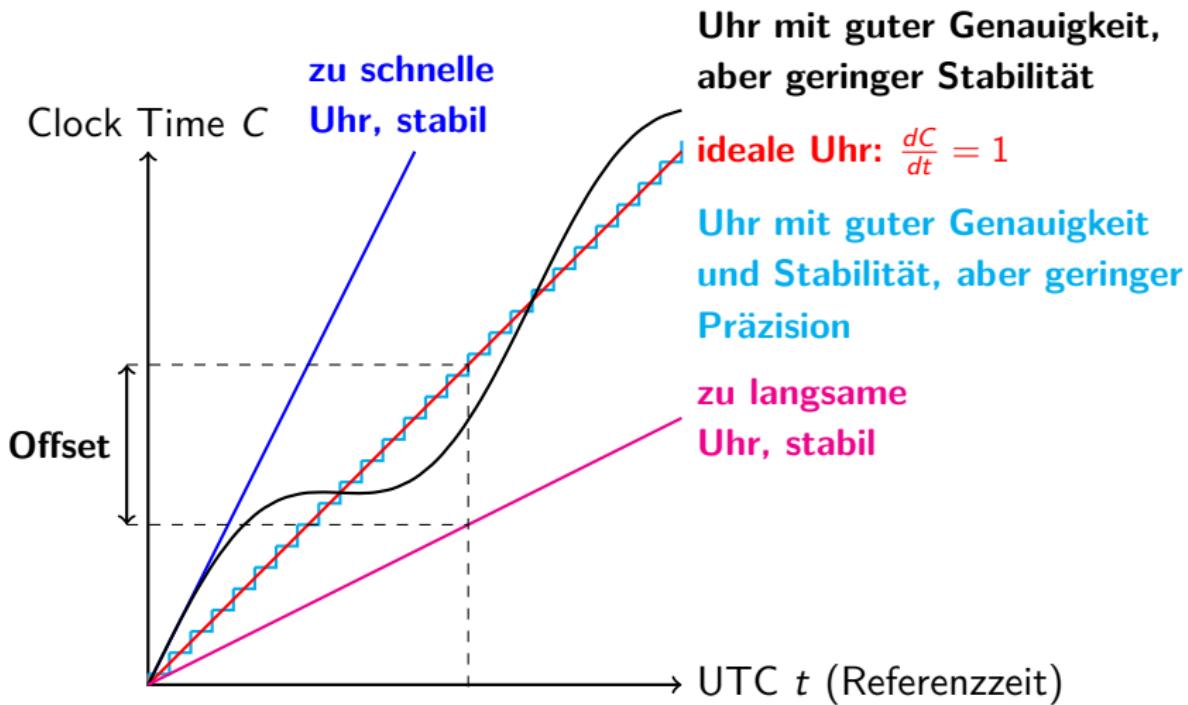
Offset

- Zeitdifferenz zwischen zwei Uhren bzw. zur Referenzzeit

Drift

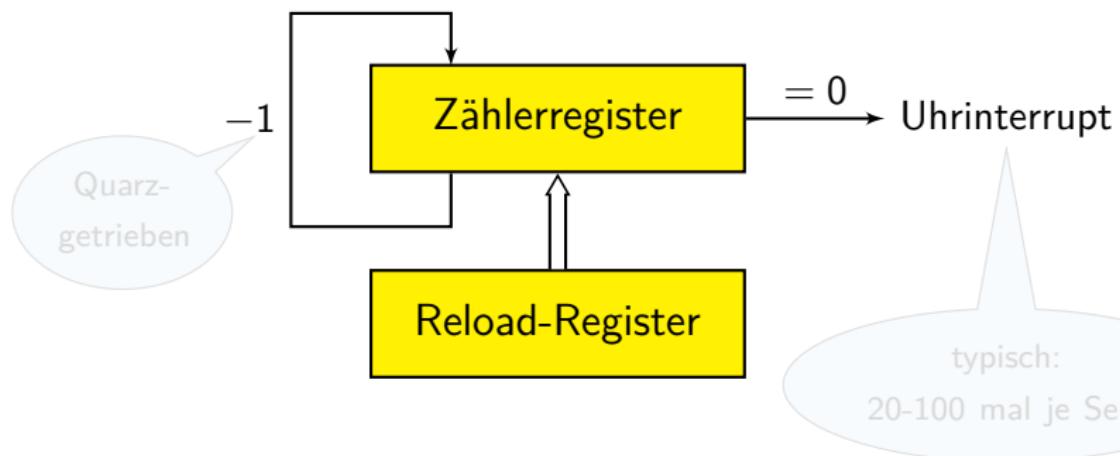
- Frequenzdifferenz zwischen zwei Uhren bzw. zur Referenzzeit

Veranschaulichung



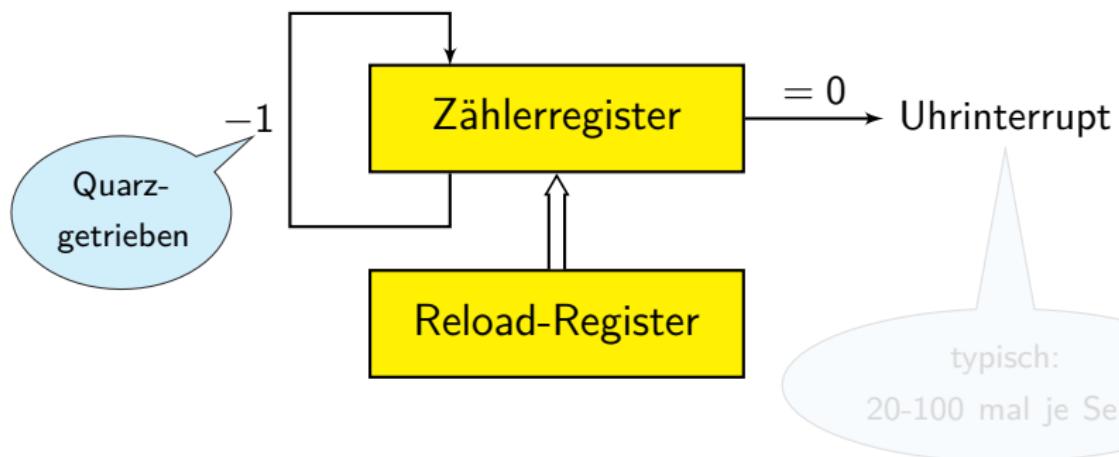
Rechneruhren

Hardware einer lokalen Rechensystemuhr



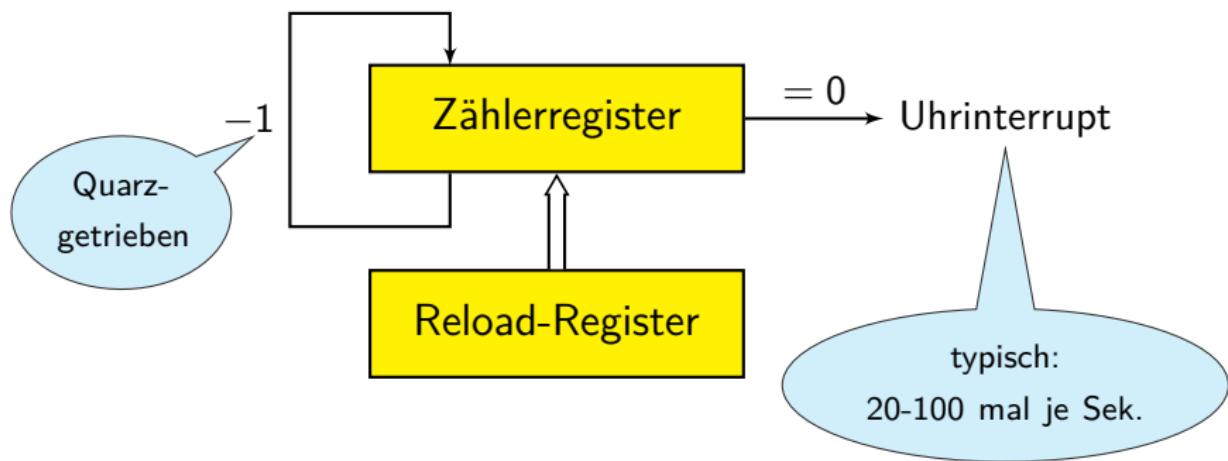
Rechneruhren

Hardware einer lokalen Rechensystemuhr



Rechneruhren

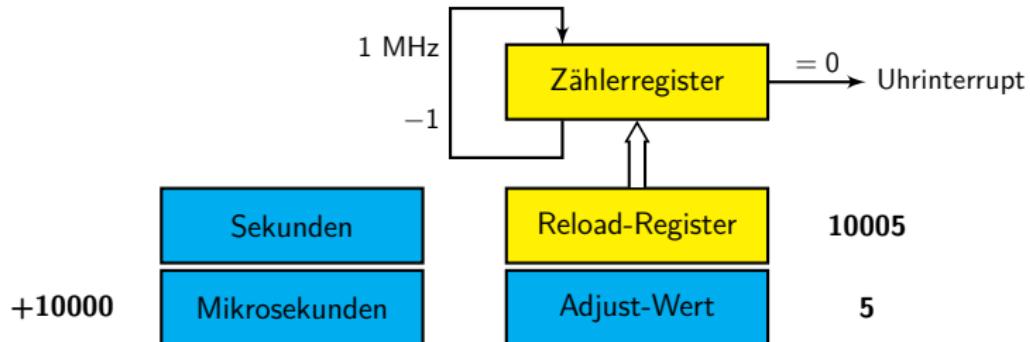
Hardware einer lokalen Rechensystemuhr



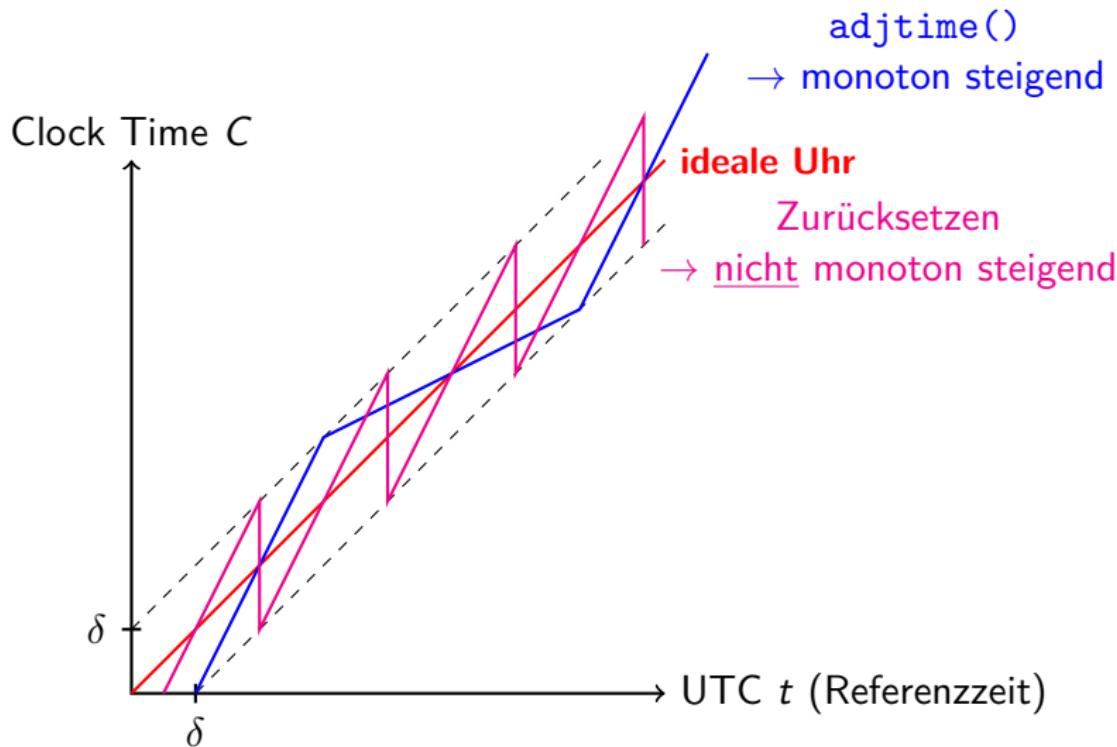
Betriebssystem-Uhren

Beispiel UNIX

- zwei 32-Bit (oder 64-Bit) Integer-Variablen
 - ▶ Anzahl Sekunden seit 1.1.1970
 - ▶ Anzahl μs (oder ns) in der aktuellen Sekunde
- typ. 100 Interrupts/s
- bei Interrupt werden Variablen um nominelle Anzahl μs erhöht
- Korrekturwert (Adjust-Wert) für Ausgleich der Drift des Quarzes
- Systemdienste `settimeofday`, `adjtime`



Prinzip der Korrektur



Referenzzeitquellen

DCF77-Uhr für einfache Anforderungen an Systemzeit

- Genauigkeit typisch: ± 2 msec

GPS-Uhr bei hohen Anforderungen (z.B. Messsystem)

- Genauigkeit typisch: ± 250 nsec

Atom-Uhr

- Rubidium / Caesium-Quellen
- Spezielle Zulassung erforderlich
- Montage in Rack
- z.T. ausschließlich für militärische Zwecke

Rechnerschnittstelle

- Erzeugung von Pulse-Per-Second (pps)-Signalen als Interrupts
- Kodierte Timecode-Signale,
z.B. IRIG-Standard (Inter Range Instrumentation Group)

Genaue lokale Betriebssystem-Uhren

Verwendung einer externen Referenzzeitquelle

Linux-Kern mit „Nano-Kernel-Patch“

- Erhöhung der Auflösung der Systemuhr auf 1 ns (statt μ s)
- Standard in neueren Linux-Kernen
- Nutzung der pps-Signale der Referenzzeitquelle als Interrupts
- Korrektur der Systemuhr entsprechend Referenzzeit der Hardware-Uhr
- Varianz der Interrupt-Latzenzeiten beeinflusst Genauigkeit
- mehrere externe Zeitquellen an einem Rechner möglich zur weiteren Erhöhung der Genauigkeit
- Genauigkeit: typisch $< 1 \mu$ s

Beispiel: David L. Mills' Uhren (Uni Delaware)



- Spectracom 8170 WWVB Receiver
- Spectracom 8183 GPS Receiver
- Hewlett Packard 105A Quartz Frequency Standard
- Hewlett Packard 5061A Cesium Beam Frequency Standard
- NTP primary time server *rackety* and *pogo* (elsewhere)

<http://doc.ntp.org/4.1.2/refclock.htm>

Kommerzielle Time Server

Time Server

- Dedizierter LAN-Netzwerkknoten zur Zeitsynchronisation
- Interne oder externe Referenzzeitquelle
- Unterstützung für Standard-Protokolle (s.u.) (NTP, SNTP, PTP/IEEE 1588)

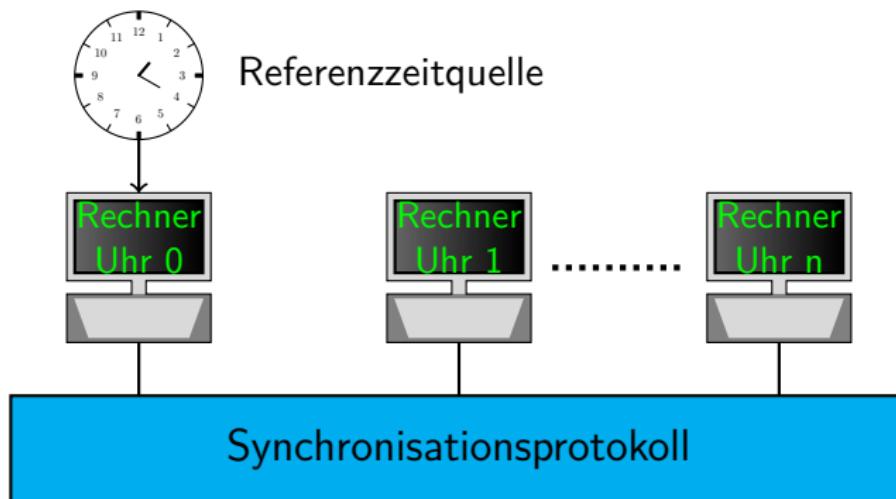
Produkte in vielen Varianten

- Meinberg (D)
- IPCAS (D)
- Galleon (UK)
- ELPROMA (NL)
- Time Tools (UK)

Synchronisationsprotokolle

Konstruktion einer verteilten Zeitbasis für Rechensysteme

- UTC-basierte externe Referenzzeitquelle
- lokale Uhren in den Rechensystemen
- Synchronisationsprotokoll



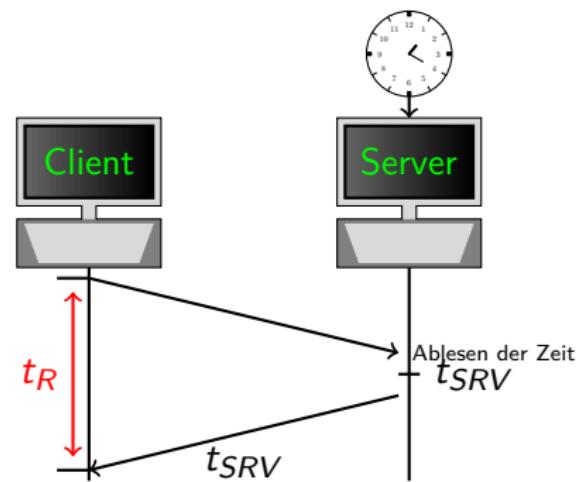
Probleme

- Nachrichtenverzögerung im Netzwerk nicht deterministisch
 - Bearbeitung der Protokollnachrichten zeitlich nicht deterministisch
- ⇒ **keine exakte Synchronisation möglich**

Algorithmus von Cristian (1989)

- passiver Zeitserver (als Referenzzeitquelle)
- periodisches Abfragen der Zeit durch Klienten
- mittlere Roundtriptlaufzeit (incl. Verarbeitungszeit auf dem Server) messen und berücksichtigen
- Schwächen:
 - ▶ „Rückwärtsgehen“ einer Uhr ist möglich
 - ▶ Schwankungen in Nachrichtenlaufzeiten

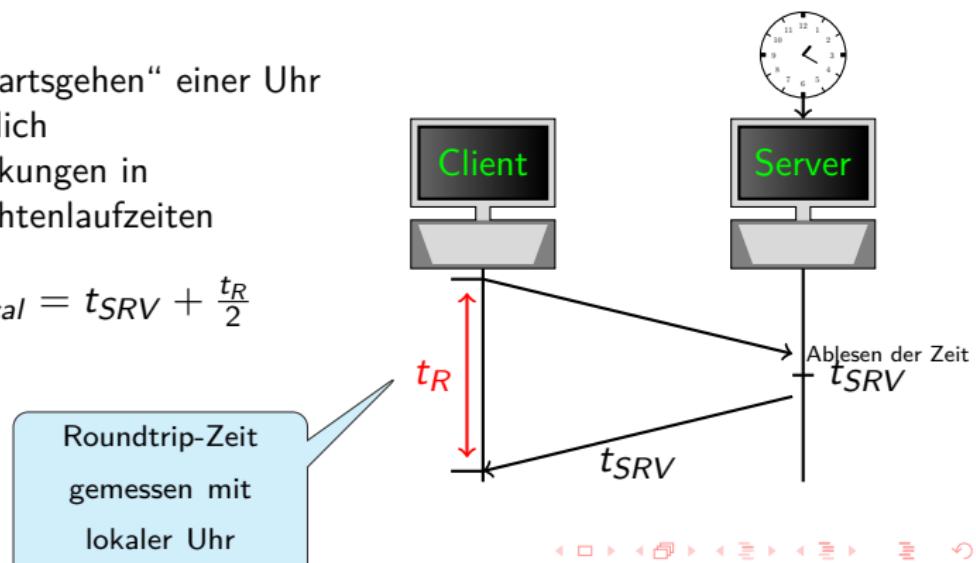
Setze: $t_{local} = t_{SRV} + \frac{t_R}{2}$



Algorithmus von Cristian (1989)

- passiver Zeitserver (als Referenzzeitquelle)
- periodisches Abfragen der Zeit durch Klienten
- mittlere Roundtriptlaufzeit (incl. Verarbeitungszeit auf dem Server) messen und berücksichtigen
- Schwächen:
 - ▶ „Rückwärtsgehen“ einer Uhr ist möglich
 - ▶ Schwankungen in Nachrichtenlaufzeiten

Setze: $t_{local} = t_{SRV} + \frac{t_R}{2}$



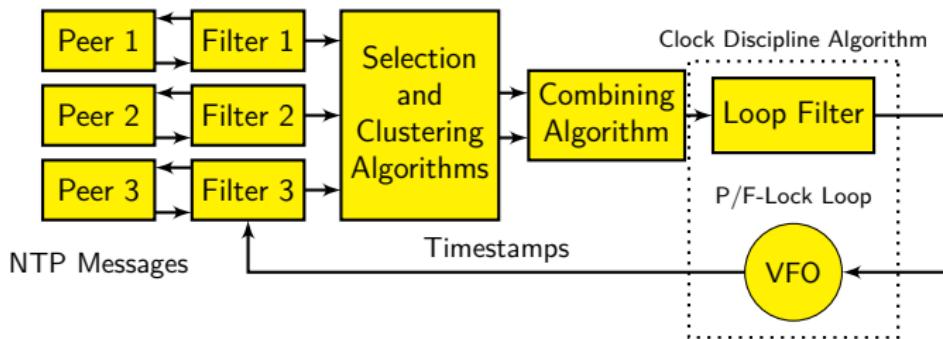
Time Synchronisation Protocol (TSP)

- Berkeley UNIX `timed`
- basiert auf ICMP/IP
- etabliert „mittlere Netzwerkzeit“ in allen Stellen
- Master/Slave-Algorithmus
 - ▶ aktiver Master: fragt aktuelle Zeiten aller Knoten ab, berechnet Mittel
 - ▶ verteilt Differenz (Offset) an jeden Client
- nutzt `settimeofday()` und `adjtime()` in den Knoten
- deutliche Schwächen
 - ▶ „Rückwärtsgehen“ einer Uhr ist möglich
 - ▶ keine Kompensation von Schwankungen in Nachrichtenlaufzeiten
 - ▶ keine Fehlerabschätzung
 - ▶ schlechte Skalierbarkeit
- Variante: Master mit ext. Referenzzeitquelle verteilt aktuelle Zeit statt berechnetem Mittelwert

Network Time Protocol (NTP)

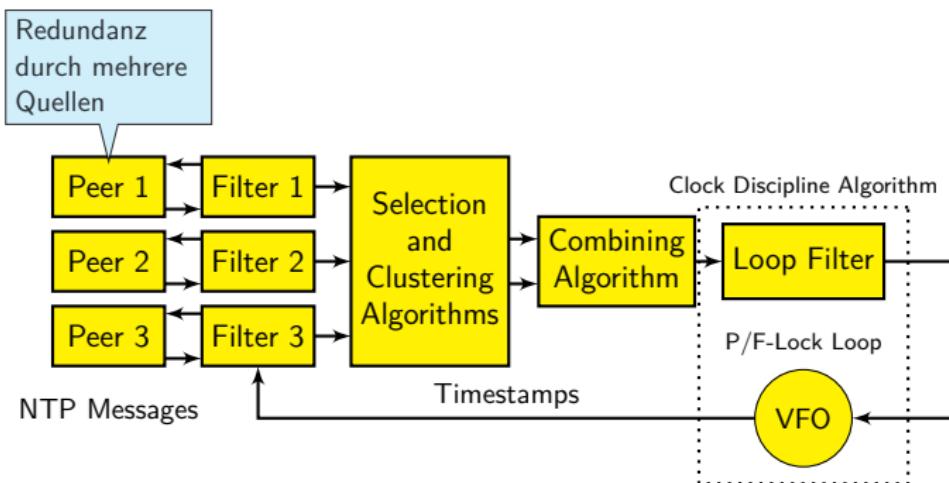
- Entwicklung primär durch D. Mills (Univ. of Delaware) getrieben
- <http://www.ntp.org>
- Ziele:
 - ▶ hohe Genauigkeit
 - ▶ Berücksichtigung schwankender Nachrichtenlaufzeiten
 - ▶ Berücksichtigung von Rechnerausfällen durch Bezug zu mehreren Zeitservern (Peers)
 - ▶ Aussortieren offensichtlich unbrauchbarer Zeitquellen (false ticker)
 - ▶ eingeschränkte Authentifizierung, Verschlüsselung
 - ▶ hohe Skalierbarkeit
- Heute Internet Standard
 - ▶ RFC 1305, 1992, frühere Version RFC 1129, RFC 958 (1985)
 - ▶ > 1.000.000 Rechner, Router, usw.
 - ▶ Nutzt UDP, Port 123
 - ▶ UNIX ntpd, xntpd (Clients aber auch für fast alle anderen Systeme)
 - ▶ Zeitserver der PTB: ptbtime1.ptb.de, ptbtime2.ptb.de
- Genauigkeit:
 - ▶ im LAN <1 ms, Internet < ca. 10 ms

NTP(2)-Arbeitsweise



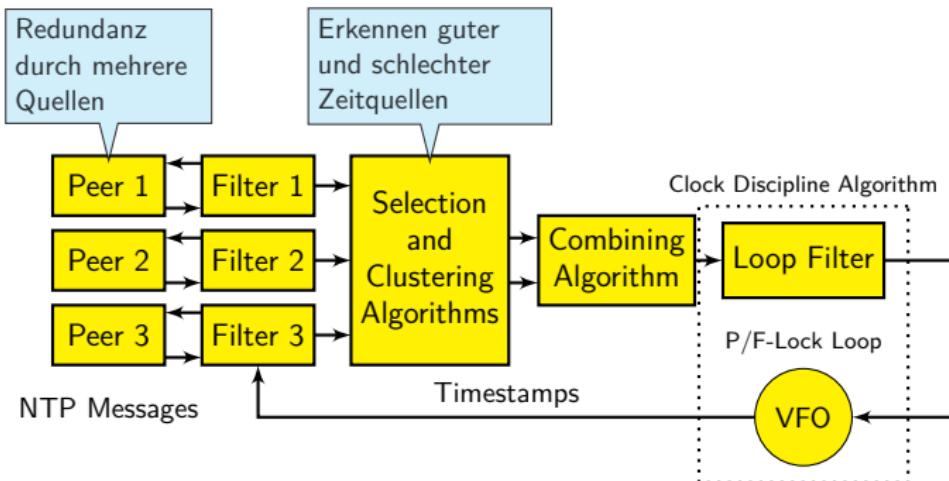
(Abbildung von Mills)

NTP(2)-Arbeitsweise



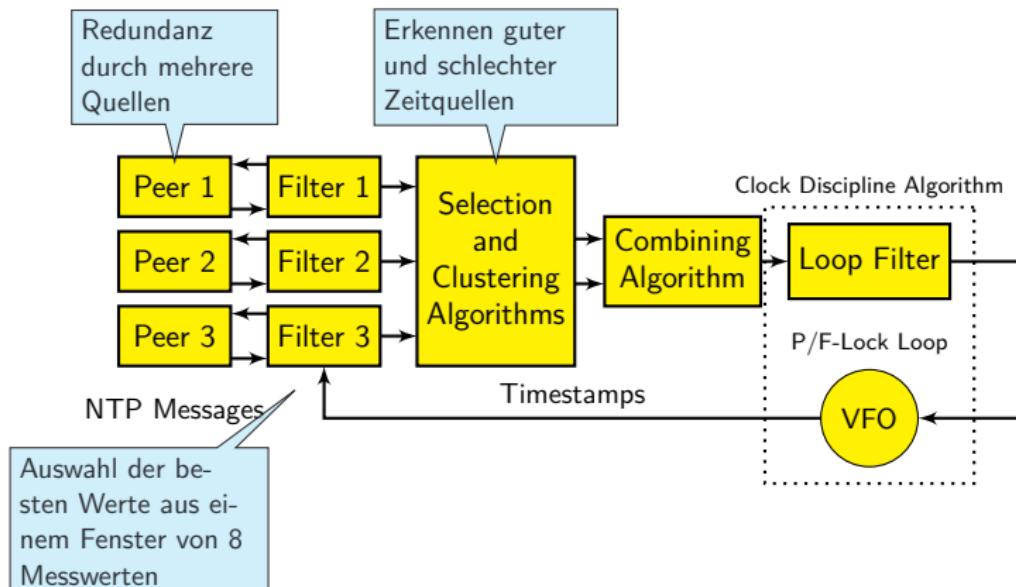
(Abbildung von Mills)

NTP(2)-Arbeitsweise



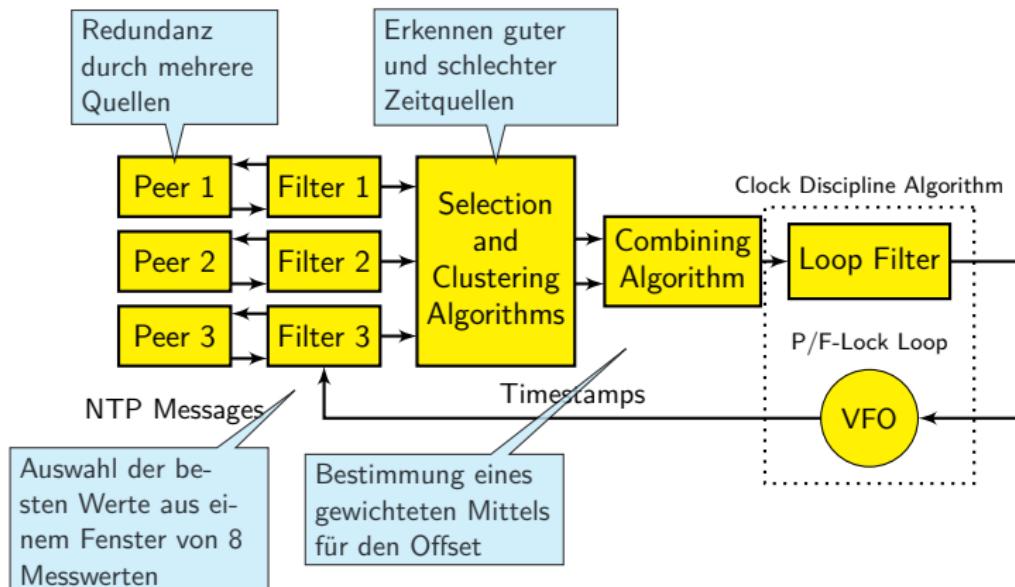
(Abbildung von Mills)

NTP(2)-Arbeitsweise



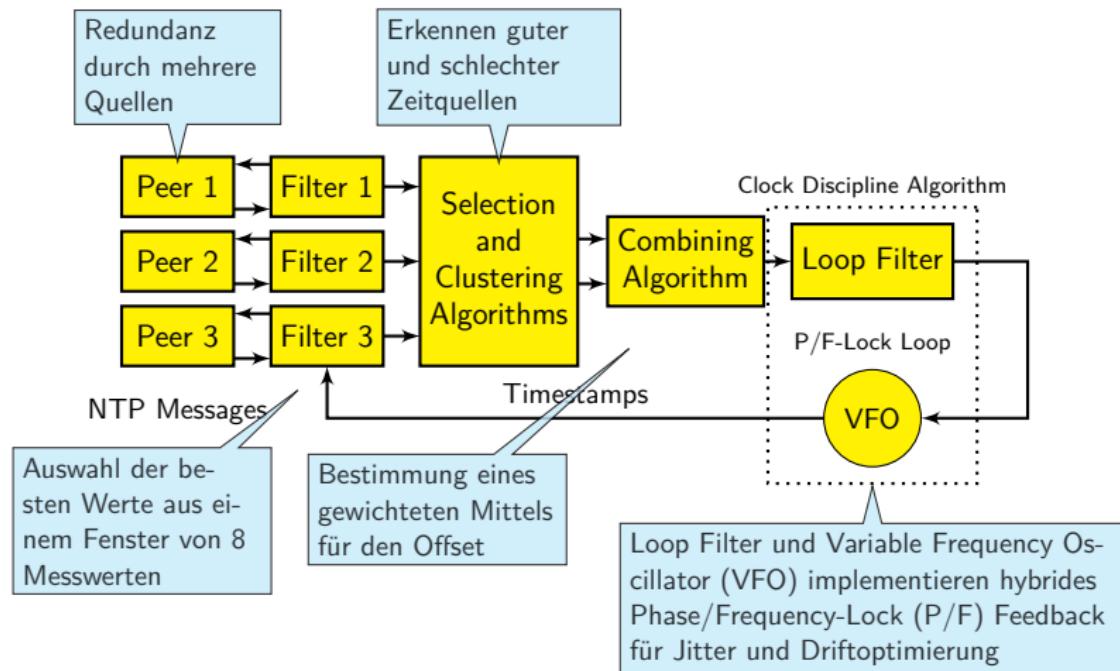
(Abbildung von Mills)

NTP(2)-Arbeitsweise



(Abbildung von Mills)

NTP(2)-Arbeitsweise

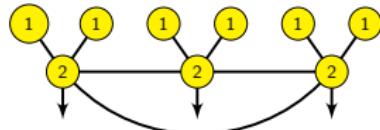


(Abbildung von Mills)

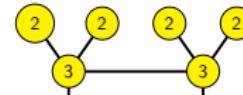
NTP(3)

Server legen Zeit fest, Clients beziehen Zeit
Hierarchiebildung der Server durch „Stratum“-Level

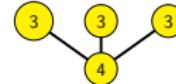
- Knoten mit externen Referenzzeitquellen bilden Stratum 1 -Server
 (Genauigkeit: $< 1 \mu\text{s}$ möglich)
- Stratum n - Server synchronisieren sich mit Stratum n-1 - Servern, usw.
- im Internet (2015)
 - ▶ Jeweils ca. 300 aktive Stratum-1 und Stratum-2 Server
<http://support.ntp.org/bin/view/Servers/StratumOneTimeServers>
 - ▶ Praktisch: 4-stufige Hierarchie, Lastausgleich durch regionale NTP Pool Server
- Typische Strukturen:



Unternehmens-Zeitserver
 (fehlertolerant)



Abteilungs-Zeitserver
 (fehlertolerant)



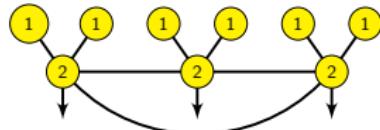
Workstation

NTP(3)

Server legen Zeit fest, Clients beziehen Zeit

Hierarchiebildung der Server durch „Stratum“-Level

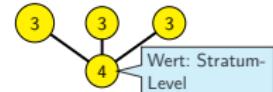
- Knoten mit externen Referenzzeitquellen bilden Stratum 1 -Server
(Genauigkeit: < 1 μ s möglich)
- Stratum n - Server synchronisieren sich mit Stratum n-1 - Servern, usw.
- im Internet (2015)
 - ▶ Jeweils ca. 300 aktive Stratum-1 und Stratum-2 Server
<http://support.ntp.org/bin/view/Servers/StratumOneTimeServers>
 - ▶ Praktisch: 4-stufige Hierarchie, Lastausgleich durch regionale NTP Pool Server
- Typische Strukturen:



Unternehmens-Zeitserver
(fehlertolerant)



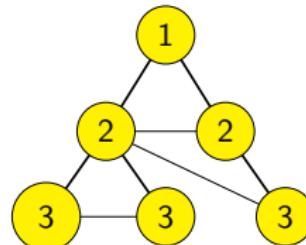
Abteilungs-Zeitserver
(fehlertolerant)



Workstation

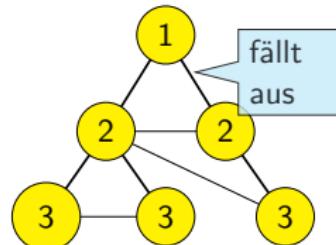
NTP(3)

- dynamisch festgelegte logische Verbindungsstruktur mit Backup-Verbindungen
 - ▶ spannende Bäume minimalen Gewichts basierend auf Server Level und Gesamtsynchronisationsverzögerung jedes Servers zu Primary Servern
- Beispiel Verbindungstopologie
- Nachrichtenaustausch zwischen Servern zwischen 64 sec und 1024 sec (17 min) je nach Qualität der Verbindung
- 64 Bit Zeitmarken
 - ▶ 32 Bit für Sekunden seit 1.1.1900 00:00:00
 - ▶ 32 Bit für Sekundenbruchteil
- Nutzung von `settimeofday()` und `adjtime()` zur Durchsetzung großer bzw. kleiner (<0.128 sec) Korrekturen.
- Kein Zurücksetzen der Uhr



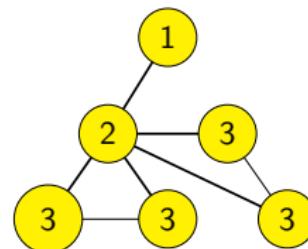
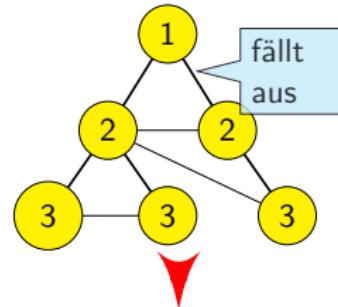
NTP(3)

- dynamisch festgelegte logische Verbindungsstruktur mit Backup-Verbindungen
 - ▶ spannende Bäume minimalen Gewichts basierend auf Server Level und Gesamtsynchronisationsverzögerung jedes Servers zu Primary Servern
- Beispiel Verbindungstopologie
- Nachrichtenaustausch zwischen Servern zwischen 64 sec und 1024 sec (17 min) je nach Qualität der Verbindung
- 64 Bit Zeitmarken
 - ▶ 32 Bit für Sekunden seit 1.1.1900 00:00:00
 - ▶ 32 Bit für Sekundenbruchteil
- Nutzung von `settimeofday()` und `adjtime()` zur Durchsetzung großer bzw. kleiner (<0.128 sec) Korrekturen.
- Kein Zurücksetzen der Uhr



NTP(3)

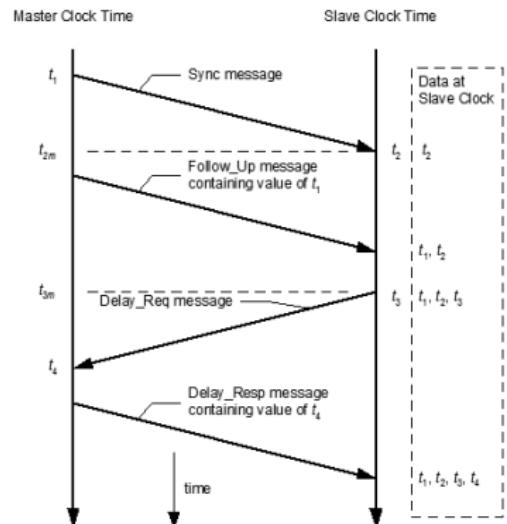
- dynamisch festgelegte logische Verbindungsstruktur mit Backup-Verbindungen
 - ▶ spannende Bäume minimalen Gewichts basierend auf Server Level und Gesamtsynchronisationsverzögerung jedes Servers zu Primary Servern
- Beispiel Verbindungstopologie
- Nachrichtenaustausch zwischen Servern zwischen 64 sec und 1024 sec (17 min) je nach Qualität der Verbindung
- 64 Bit Zeitmarken
 - ▶ 32 Bit für Sekunden seit 1.1.1900 00:00:00
 - ▶ 32 Bit für Sekundenbruchteil
- Nutzung von `settimeofday()` und `adjtime()` zur Durchsetzung großer bzw. kleiner (<0.128 sec) Korrekturen.
- Kein Zurücksetzen der Uhr



Precision Time Protocol (PTP, IEEE 1588)



- Hauptsächlich für mess- und regelungstechnische Anwendungen
 - Erreicht höhere Genauigkeit als NTP für Netze mit räumlich begrenzter Ausdehnung
 - Master-Slave-Verfahren
 - Automatische Wahl der besten Uhr als Grandmaster-Clock
 - Primär auf Ethernet-Netzen angewendet
 - Timestamping-Unit kann als Teil des Netzwerk-Controllers (in Hardware) implementiert sein
- ⇒ Genauigkeit im ns-Bereich, in Software im μs -Bereich
- Ptpd als freie Implementierung
 - Verbesserte Version IEEE 1588-2008



http://www.real-time-systems.com/ieee_1588/index.php

$$t_2 - t_1 = \text{offset} + d$$

$$t_4 - t_3 = -\text{offset} + d$$

$$\text{offset} = \frac{(t_2 - t_1 - t_4 + t_3)}{2}$$

Logische Zeitmarken

Realzeit ist nicht immer notwendig

Beispiele:

- Ordnen von Ereignissen (vor - nach)
- zeitmarkenbasiertes Concurrency Control in Datenbanken

Lamport Zeitstempel

Relation happens-before

- Notation: $a \rightarrow b$ (a passiert-vor b)
- Ereignisse im selben Prozess sind linear geordnet
- Nachrichtenversand:
 - ▶ a sei Ereignis des Versendens einer Nachricht m
 - ▶ b sei Empfang der Nachricht m in einem anderen Prozess
 - ▶ dann gilt: $a \rightarrow b$
- Relation ist transitiv:
 - ▶ $a \rightarrow b, b \rightarrow c \Rightarrow a \rightarrow c$
- Nebenläufigkeit:
 - ▶ falls weder $a \rightarrow b$ noch $b \rightarrow a$ gilt, heißen a und b nebenläufig

Uhrenbedingung

- $C(a)$ bezeichne die (logische) Zeit, zu der das Ereignis a stattfinde.
- $a \rightarrow b \Rightarrow C(a) < C(b)$

Lamport-Uhren

Algorithmus für logische Uhren nach Lamport (1978)

Annahmen:

- Prozesse kommunizieren über Nachrichten (und nur über Nachrichten) miteinander
- jeder Prozess P hat eine logische Uhr C_P
- jedes Ereignis e des Prozesses P erhält logischen Zeitstempel $C_P(e)$
- zwei aufeinander folgende Ereignisse e_i und e_{i+1} eines Prozesses haben nie den gleichen Zeitstempel: $C_P(e_i) < C_P(e_i + 1)$

Lamport-Uhren (2)

Beispiel:

A	B	C
0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	56	70
48	64	80
54	72	90
60	80	100

m1

m2

m3

m4

Uhren mit unterschiedlichen
Geschwindigkeiten ohne Korrektur

A	B	C
0	0	0
6	8	10
12	16	20
18	24	30
24	32	40
30	40	50
36	48	60
42	56	70
48	64	80
70	77	85
76		

m1

m2

m3

m4

Uhren mit unterschiedlichen
Geschwindigkeiten **mit** Korrektur

Lamport-Uhren (3)

Algorithmus:

- Berücksichtigung von Kausalität im Nachrichtenversand!
- Sendeereignis s einer Nachricht m in Prozess A:
 - ▶ Zeitmarke $C_A(s)$
 - ▶ Versende Nachricht m zusammen mit aktuellem Zeitstempel des sendenden Prozesses $t = C_A(s)$
- Empfangsereignis e der Nachricht m in Prozess B :
 - ▶ sei $C_B(\text{alt})$ die Zeitmarke des letzten Ereignisses in B
 - ▶ Setze $C_B(e) := \max\{C_B(\text{alt}), t\} + 1$
- Falls zwei Ereignisse in verschiedenen Prozessen die gleiche Zeitmarke haben sollten, ordne sie anhand der Prozessordnung
- Algorithmus erfüllt Uhrenbedingung
- Umkehrung gilt **nicht**:
 $C(a) < C(b) \Rightarrow a \rightarrow b$ ist falsch !
- Lamport-Uhren lösen nicht das Kausalitätsproblem

Vector Clocks

Vektor-Uhren, Mattern (Uni Kaiserslautern, 1989)

Vektor-Uhren lösen o.a. Kausalitätsproblem

Algorithmus:

- nachrichtenbasierte Kommunikation
- jeder Prozess P_i besitzt Uhr VC_i als Vektor von Zeitmarken
- lokales Ereignis in P_i :
 - ▶ $VC_i[i] := VC_i[i] + 1$, sonst unverändert
- Sendeereignis in P_i :
 - ▶ $VC_i[i] := VC_i[i] + 1$ (Erhöhe eigenen Ereigniszähler)
 - ▶ Versende Nachricht mit eigener Vektorzeit $vt = VC_i$
- Empfangsereignis in P_k :
 - ▶ $VC_k[j] := \max\{VC_k[j], vt[j]\}$ für alle j
 - ▶ $VC_k[k] := VC_k[k] + 1$ (Erhöhe eigenen Ereigniszähler)

Vector Clocks (2)

Vergleich von Zeitmarkenvektoren

- $S \leq T \Rightarrow S[i] \leq T[i]$ für alle i
- $S < T \Rightarrow S \leq T$ und $S \neq T$
- $S || T \Rightarrow \neg(S < T) \text{ und } \neg(T < S)$

Nebenläufigkeit

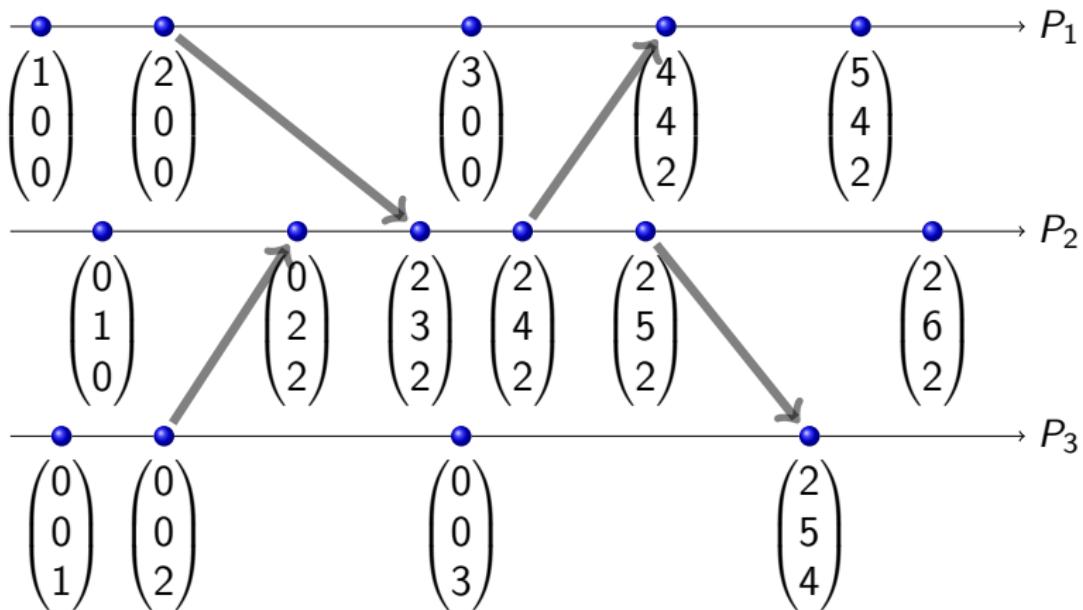
- Ereignisse a und b sind nebenläufig $\Leftrightarrow VC(a) || VC(b)$

Kausalität

- $a \rightarrow b \Leftrightarrow VC(a) < VC(b)$

Vector Clocks (3)

Beispiel



- kausal abhängige Ereignisse, z.B. $(0, 0, 1) \rightarrow (5, 4, 2)$, $(1, 0, 0) \rightarrow (2, 6, 2)$
- nebenläufige Ereignisse, z.B. $(0, 0, 3) \parallel (5, 4, 2)$

Zusammenfassung

- Zeiten lassen sich mit einer Referenzzeit hinsichtlich ihrer Stabilität, Genauigkeit und Auflösung vergleichen.
Dabei lassen sich Drift und Offset ermitteln.
- Um lokale Uhren in einem verteilten System mit einer Referenzzeitquelle zu synchronisieren bedarf es einen Zeit-Synchronisationsprotokolls.
- Lamport-Uhren und Vektor-Uhren können logische Zeitmarken realisieren.

8. Verteilte Transaktionen



<http://statuesquo.blogspot.fr/2017/07/kreislauf-des-geldes-circuit-de-largent.html>

Inhalt

8. Verteilte Transaktionen

8.1 Einführung

8.2 Transaktionskonzept

8.3 Stellenlokale Commit-Verfahren

8.4 Das 2-Phasen Commit-Protokoll

8.5 X/Open Distributed Transaction Processing

Einführung

Neues Problem bei der Entwicklung verteilter Anwendungen

- Ausfall von einzelnen Komponenten des verteilten Systems
(partial failure property)
⇒ komplexe Fehlersituationen in verteilten Anwendungsprogrammen

Motivation für Transaktionen

- Atomare Aktionen als Erweiterung des DB-Tansaktionskonzepts zu allgemeinem Programmierkonstrukt
- Reduktion der Komplexität der Anwendungsprogrammierung in Gegenwart von Fehlern und Nebenläufigkeit
- Bessere Einsicht in die Wirkungsweise eines Programms
- Automatisches Backward Error Recovery, Kombination mit Forward Error Recovery-Verfahren möglich

Transaktionskonzept

Eine **Transaktion** ist die Zusammenfassung einer Menge von Aktionen (Operationen auf log. Betriebsmitteln) mit folgenden Eigenschaften (**ACID**) (Härder/Reuter, 1983):

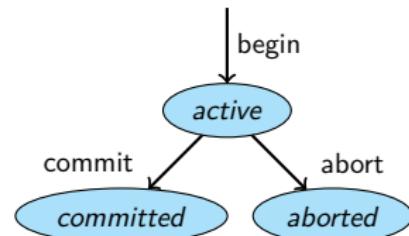
- **Atomicity** = Atomarität im Fehlerfall (Alles-oder-Nichts):
 - ▶ Transaktion entweder vollständig ausgeführt oder erscheint, als ob nie begonnen
 - ▶ Kein Zwischenzustand zwischen Anfangs- und Endzustand sichtbar
- **Consistency** = Konsistenz:
 - ▶ Eine Transaktion überführt das System von einem konsistenten Zustand in einen konsistenten Zustand
 - ▶ Allgemein: Erfüllung aller Anwendungs-Constraints
- **Isolation**:
 - ▶ Verhalten jeder Transaktion als ob „allein auf der Welt“
- **Durability** = Dauerhaftigkeit (Permanenz der Wirkung):
 - ▶ Wirkung abgeschlossener Transaktionen geht nicht verloren (selbst bei Auftreten aller erlaubten Fehler eines Fehlermodells)

Lokale/Verteilte Transaktionen

Lokale Transaktion

- Wirkung auf einzelnes Rechensystem beschränkt

Zustandsdiagramm



Verteilte Transaktion

- Wirkung auf mehreren Stellen eines verteilten Rechensystems

?

im Folgenden zu entwickeln

Fehlermodell

Das Fehlermodell beschreibt alle vorhergesehenen Fehler, auf die ein System geordnet reagiert, alle anderen heißen Katastrophen.

Transaktionsfehler (transaction abort):

- Zurücksetzen einzelner Transaktionen, z.B. durch
 - ▶ Expliziter Abbruch durch Benutzer
 - ▶ Fehler im Applikationsprogramm
 - ▶ als Folge einer Deadlockauflösung

Stellenfehler (site failure):

- Ausfall eines beteiligten Rechensystems, z.B. durch
 - ▶ Transiente oder permanente Hardware-Fehler (auch Stromausfall)
 - ▶ Betriebssystemabsturz mit reboot
- Alle laufenden Prozesse stürzen ab
- Alle Transaktionen im Zustand *active* gehen in den Zustand *aborted* über

Fehlermodell (2)

Medienfehler:

- Nicht-behebbarer Fehler auf nicht-flüchtigem Speichermedium, das von Transaktionsverarbeitung genutzt wird, z.B.
 - ▶ Magnetplatte (genutzt für Speicherung der log. Daten)
 - ▶ Magnetband (genutzt für Logging)
- Standardbehandlung durch stabilen Speicher (redundante Speicherung auf mehreren Medien, z.B. Spiegelplatten, RAID, ...)
- hier nicht weiter betrachtet

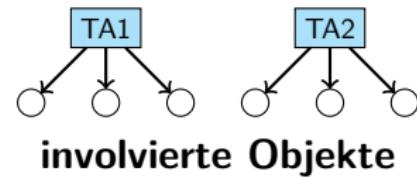
Kommunikationsfehler:

- Fehler im Nachrichtensystem führen zum Verlust von Nachrichten
- Partitionierung: Zerfall des Netzes in mehrere isolierte Teilnetze

Flache / geschachtelte Transaktionen

Flache Transaktionen

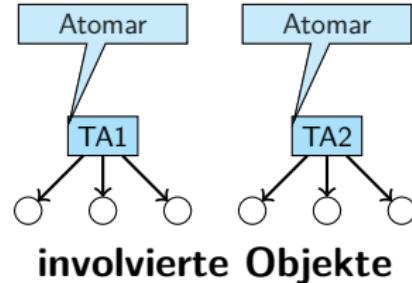
- klassisches Modell aus DB-Kontext
- Transaktion involviert Menge von Objekten (log. Betriebsmitteln)
- Transaktionen können Objekte gemeinsam nutzen (geregelt durch Concurrency Control- Mechanismen)
- Transaktionen können nicht geschachtelt werden



Flache / geschachtelte Transaktionen

Flache Transaktionen

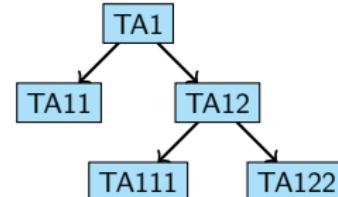
- klassisches Modell aus DB-Kontext
 - Transaktion involviert Menge von Objekten (log. Betriebsmitteln)
 - Transaktionen können Objekte gemeinsam nutzen (geregelt durch Concurrency Control- Mechanismen)
 - Transaktionen können nicht geschachtelt werden
- ⇒ Nachteil: Keine Möglichkeit, Teilergebnisse festzuschreiben



Flache / geschachtelte Transaktionen (2)

Geschachtelte Transaktionen (Nested Transactions)

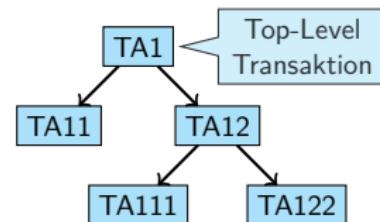
- Eine Transaktion kann (auch zu einem Zeitpunkt) innere Transaktionen (Subtransaktionen) besitzen
- isolierte Zurücksetzbarkeit innerer Transaktionen: Abort einer inneren TA führt zu Exception (nicht: Abbruch!) bei übergeordneter TA
- Abort einer TA führt zum Abort aller inneren TAs
- Commitment relativ zum Parent (endgültig erst mit Commitment der Top-Level TA)
- Nebenläufigkeitskontrolle auf jeder Schachtelungsebene



Flache / geschachtelte Transaktionen (2)

Geschachtelte Transaktionen (Nested Transactions)

- Eine Transaktion kann (auch zu einem Zeitpunkt) innere Transaktionen (Subtransaktionen) besitzen
- isolierte Zurücksetzbarkeit innerer Transaktionen: Abort einer inneren TA führt zu Exception (nicht: Abbruch!) bei übergeordneter TA
- Abort einer TA führt zum Abort aller inneren TAs
- Commitment relativ zum Parent (endgültig erst mit Commitment der Top-Level TA)
- Nebenläufigkeitskontrolle auf jeder Schachtelungsebene



Stellenlokale Commit-Verfahren

Verfahren zur lokalen Bereitstellung von Atomarität im Fehlerfall und Permanenz der Wirkung:

- Intention Lists (Lampson 1981)
- Shadowing (Gray 1981)
- Write-Ahead Logging

Intention lists

Vorgehensweise

- Beabsichtigte Veränderungen auf Datenobjekten werden in Liste gesammelt (d.h. Arbeiten auf Kopien der Originaldaten)
- Liste wird in stabilen Speicher geschrieben
- Entscheidung (committed-aborted) treffen
- Falls aborted: Liste verwerfen
- Falls committed: Originale der Objekte in nicht-flüchtigem Speicher aktualisieren

Im Verteilten System

- Jeder Knoten führt eine vorläufige Liste und kennt den Koordinator
- Ein Koordinator schreibt alle Knoten in eine Liste und benachrichtigt diese
- Die benachrichtigten Knoten aktualisieren die Objekte gemäß der Liste und löschen diese

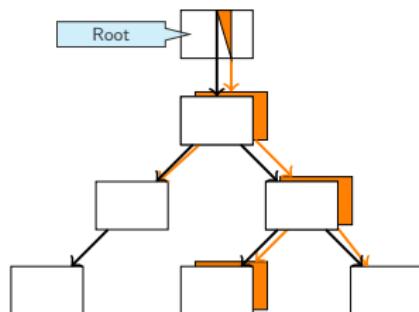
Shadowing

Vorgehensweise

- Nichtflüchtiger Speicher als Baumstruktur mit Verweisblöcken (entsprechend UNIX- Dateisystem) angenommen
- After Images aller Blöcke als Shadow-Version bis zur Wurzel anlegen
- Entscheidung (committed-aborted) treffen durch atomaren Pointer-Swap in Root-Block (Schreiben eines Blockes)
- Falls aborted: Shadow-Struktur verwerfen
- Falls committed: ehemalige Original-Blöcke freigeben, Shadow-Blöcke sind jetzt die Originale

nach Stellenfehler

- Entweder alter Zustand oder neuer Zustand ist etabliert



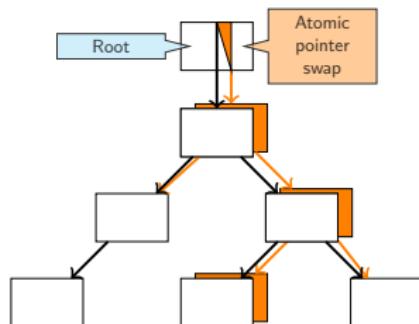
Shadowing

Vorgehensweise

- Nichtflüchtiger Speicher als Baumstruktur mit Verweisblöcken (entsprechend UNIX- Dateisystem) angenommen
- After Images aller Blöcke als Shadow-Version bis zur Wurzel anlegen
- Entscheidung (committed-aborted) treffen durch atomaren Pointer-Swap in Root-Block (Schreiben eines Blockes)
- Falls aborted: Shadow-Struktur verwerfen
- Falls committed: ehemalige Original-Blöcke freigeben, Shadow-Blöcke sind jetzt die Originale

nach Stellenfehler

- Entweder alter Zustand oder neuer Zustand ist etabliert



Shadowing (2)

Vorteile

- Atomarer Zustandswechsel durch Schreiben eines/des Root-Blockes

Nachteile

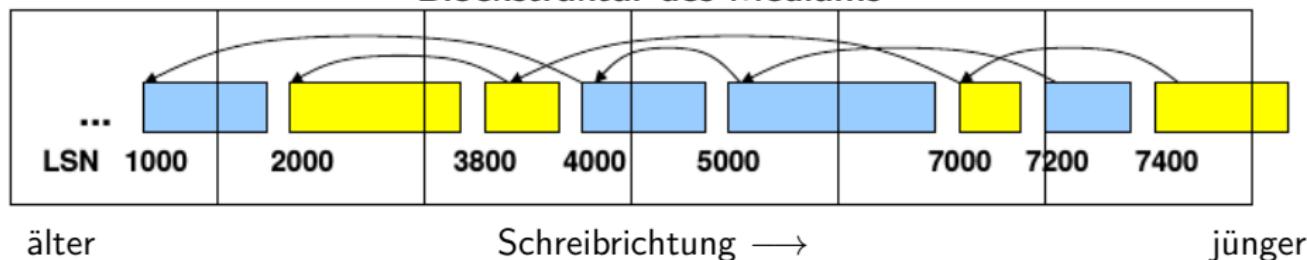
- keine Nebenläufigkeit von Commit-Vorgängen
- Physische Anordnung der Datenblöcke zueinander geht durch Commit-Vorgänge verloren, da Shadow-Blöcke aus der Freiliste genommen werden müssen.

Write-Ahead Logging

Log-Grundlagen

- Log besteht logisch aus Folge von sogenannten Log Records variabler Länge identifiziert durch Log Sequence Number (LSN)
→ Byte Offset im Log-Strom analog TCP Sequenznummer
- Log ist gemeinsam genutzt innerhalb eines Knotens
- Verkettung von zusammengehörigen Log-Einträgen eines Commit-Vorgangs
- Realisierung des Logs in replizierten Dateien (stabiler Speicher)
- Übergeordnete Tabelle mit Log-Einträgen und SQL-Zugriff in Datenbanken üblich

Blockstruktur des Mediums



Write-Ahead Logging(2)

Schreiben von Log-Einträgen

- Nur sequentielles Schreiben der Log Files (hohe Performance)
- Gepuffertes Schreiben von Log-Einträgen im Arbeitsspeicher
- *Forced Write* eines Log-Eintrags erzwingt Ablage aller vorangegangenen Log-Einträge (mit kleinerer LSN) und Rückkehr aus der Operation, nachdem Block physisch auf dem Medium steht

Lesen von Log-Einträgen

- Nur im Fall von Stellenfehlern und evtl. bei Transaktionsfehlern (s.u.)

Log-Verkürzung

- Log kann logisch beliebig lang werden, aber Restart-Dauer nach Stellenfehler muss begrenzt werden
- Nutzung von Checkpoints

Write-Ahead Logging(3)

Nutzung von Log-Einträgen im Rahmen des lokalen Commitments

- Log Records einer Transaktion sind untereinander verkettet
- Schreiben von Before Images (Undo Records) für alle Objekte, deren persistenter Originalzustand durch die Transaktion im Zustand active geändert werden (Update in Place).
- Schreiben von After Images (Redo Records) für alle erarbeiteten finalen Objektzustände der Transaktion
- Ablegen eines finalen Outcome Records mittels Forced Write:
 - ▶ erzwingt Ablage aller zugehörigen Log-Einträge
 - ▶ erscheint dieser im Log, ist das Commitment vollzogen
 - ▶ kann auch als Flag im letzten Block realisiert sein

Write-Ahead Logging Protocol

- Gesichertes Schreiben von Log Records **vor** Modifikation des originalen persistenten Zustands

Write-Ahead Logging(4)

Behandlung von Transaktionsfehlern

- (Eventuell) „Aborted“ Outcome Record erzeugen und ablegen
- Falls Before Records geschrieben wurden, deren Inhalte wieder als persistenten Objektzustand etablieren

Behandlung von Stellenfehlern

- Lesen des Logs
- Für alle nicht abgeschlossenen Transaktionen evtl. vorhandene Before Images etablieren
- Für alle Transaktionen mit vorhandenem „Committed“ Outcome Record das jeweils letzte After Image aller Objekte (irgendwann) als persistenten Objektzustand etablieren (Idempotenz)

Write-Ahead Logging(5)

Vorteile

- Mehrere ineinander verzahnte Commit-Vorgänge können gleichzeitig stattfinden
- Hohe I/O-Performance durch Buffering und sequentielles Schreiben des Logs
- Anordnung der Datenblöcke des persistenten Zustands bleibt erhalten (Update in Place)
- Parallelisierung des Logs möglich
- Nach Stellenfehler und anschließendem Neustart muss nur der Log analysiert werden
- Log kann auch von Commit-Protokollen mitgenutzt werden (s.u.)

Das 2-Phasen Commit-Protokoll

Commit-Protokolle

- dienen dazu, in einer verteilten Umgebung die Commit/Abort-Entscheidung einer Menge von Prozessen zu koordinieren
- z.B. für Durchsetzung von Atomarität im Fehlerfall und Dauerhaftigkeit für verteilte Transaktionsumgebung
- sind Spezialfälle sog. Konsensus-Protokolle (hier: ja/nein)

2-Phasen-Commit-Protokoll

- ist das am weitesten verbreitete Protokoll
- von hoher praktischer Bedeutung und auch in zahlreichen Produkten enthalten
- im weiteren besprochen

Allgemeinere Theorie

- Mehrphasen-Commit-Protokolle (z.B. Skeen, 1981)
- schwache / starke Terminierungsbedingungen führen zu blockierenden / nichtblockierenden Commit-Algorithmen

Eigenschaften eines Commit-Algorithmus

Ein Commit-Algorithmus für eine Menge von Prozessen besitzt folgende Eigenschaften:

- ① Alle Prozesse, die eine Entscheidung treffen, treffen die gleiche Entscheidung
- ② Eine einmal getroffene Entscheidung ist bindend für jeden Prozess und kann nicht widerrufen werden
- ③ Nur wenn alle Prozesse für Commit entscheiden, lautet die gemeinsame Entscheidung auf Commit, d.h. auch: sobald ein Prozess auf Abort entscheidet, muss die gemeinsame Entscheidung auf Abort lauten
- ④ Wenn zu einem Zeitpunkt alle aufgetretenen Fehler repariert sind und hinreichend lange keine neuen Fehler auftreten, erreichen die Prozesse eine gemeinsame Entscheidung.

Begriffe

Fenster der Verwundbarkeit (des Commit-Protokolls)

- Zeitspanne zwischen der lokalen Commit-Entscheidung eines Prozesses und Kenntnis der Gesamtentscheidung
- wird auch Unsicherheitsperiode des Prozesses (Uncertainty Period) genannt

Blockierendes Protokoll

- Protokoll sieht vor, dass ein Prozess die Reparatur eines Fehlers abwarten muss

Unabhängiges Recovery

- Prozess kann nach einem Fehler eine Entscheidung treffen, ohne mit anderen Prozessen zu kommunizieren

Teilhaber (Participants)

- Prozesse, die Commit-Protokoll abwickeln

Background

Sätze:

- Wenn Kommunikationsfehler oder Gesamtsystemfehler möglich sind, gibt es kein Commit-Protokoll, das keinen Prozess blockiert
Bemerkung: nicht ausgeschlossen ist die Existenz eines nicht-blockierenden Commit-Protokolls, wenn nur einzelne Stellen ausfallen
- Kein Commit-Protokoll kann unabhängiges Recovery ausgefallener Prozesse garantieren
Bemerkung: Es gibt kein Commit-Protokoll ohne Unsicherheitsperiode für Teilhaber

(Anmerkung: Nicht prüfungsrelevant)

Grundlagen des 2-P-C-Protokolls

J. Gray, 1978

Blockierender Commit-Algorithmus mit schwacher

Terminierungseigenschaft (\Rightarrow treten keine Fehler auf, treffen alle Prozesse irgendwann eine Entscheidung).

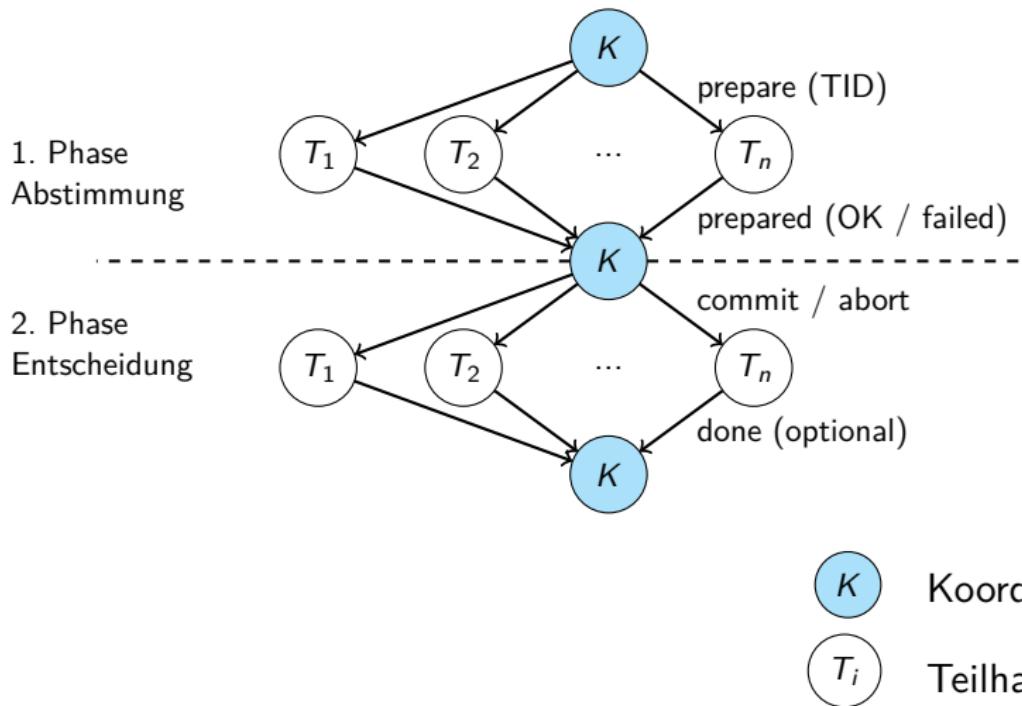
Rollen

- Teilhaber /Teilnehmer
- Koordinator als ausgezeichneter Teilhaber, der Abwicklung des Protokolls steuert Bemerkung: i.d.R. Teilhaber der Ursprungsstelle der Transaktion, der Commit-Vorgang initiiert
- Koordinator kennt alle Teilhaber
- Teilhaber kennen nur ihren Koordinator, aber sich nicht gegenseitig

Nutzung von jeweils lokalem Log jedes Teilhabers zur Fortschreibung des Zustands des Commit-Vorgangs

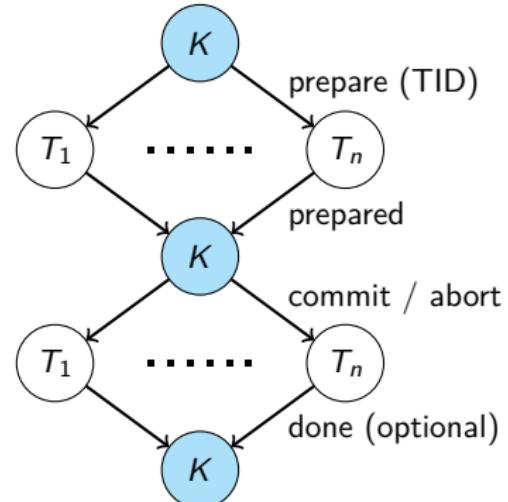
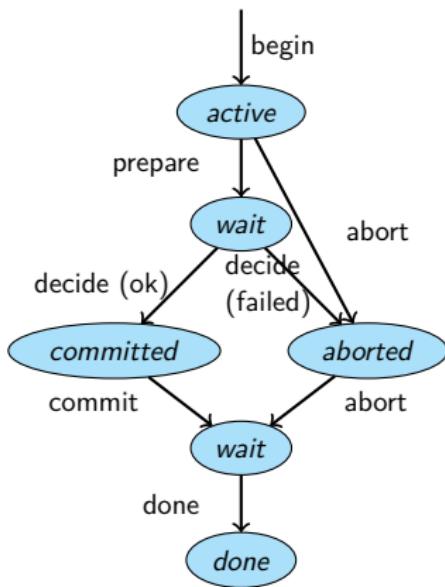
2-P-C-Protokoll

Nachrichtenfluss (Normalfall ohne Fehler)



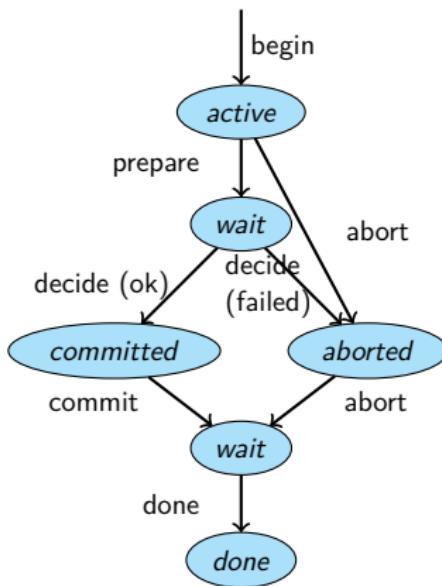
2-P-C-Protokoll (2)

Zustandsdiagramm des Koordinators

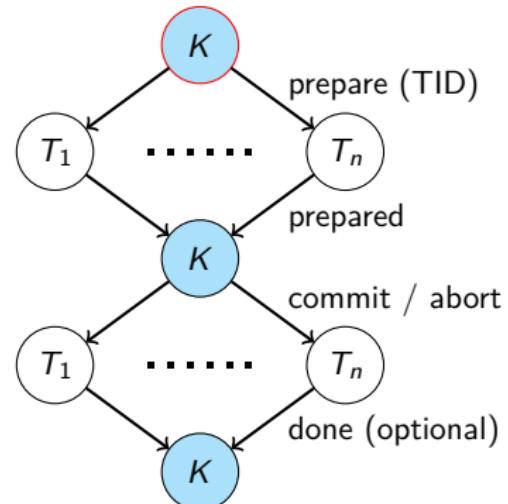


2-P-C-Protokoll (2)

Zustandsdiagramm des Koordinators

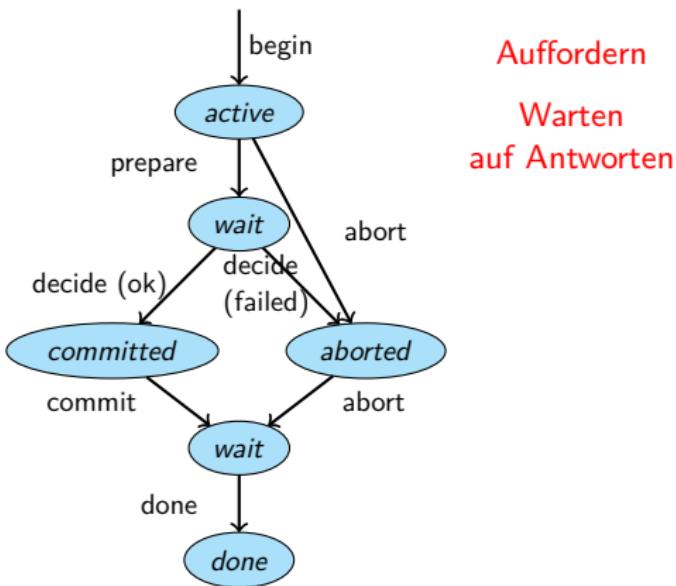


Auffordern

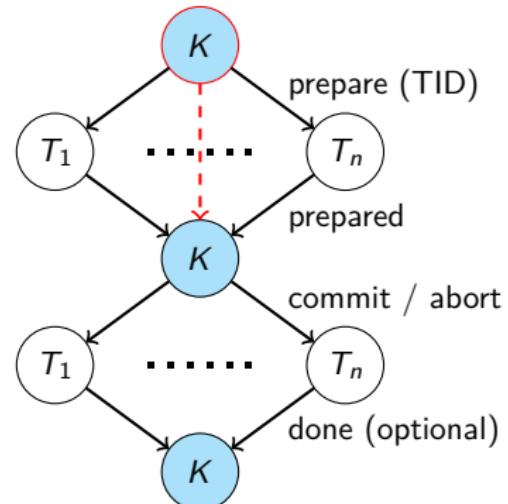


2-P-C-Protokoll (2)

Zustandsdiagramm des Koordinators

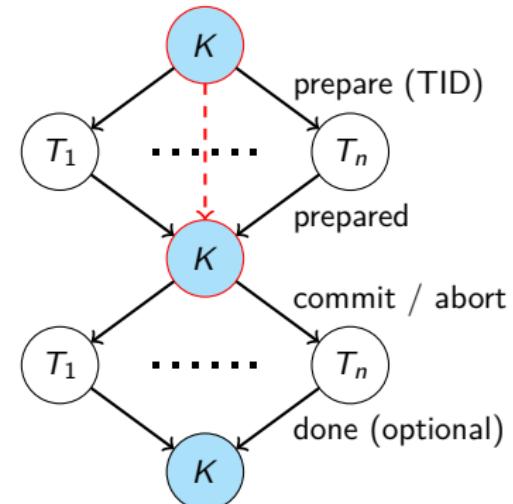
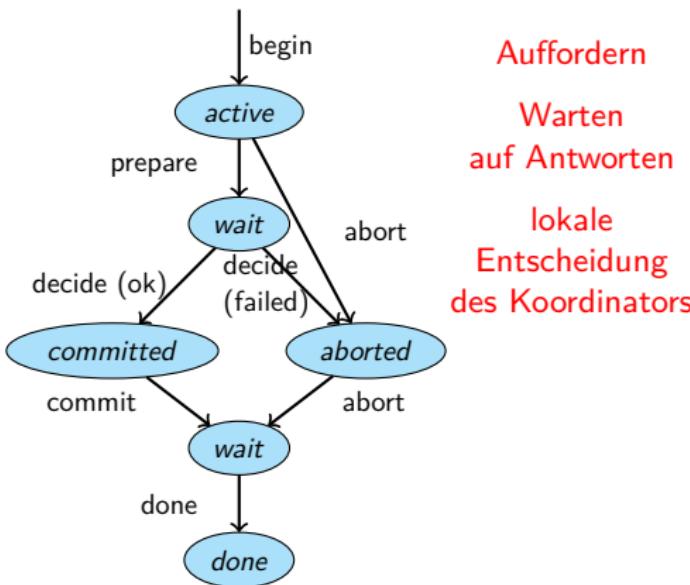


Auffordern
Warten
auf Antworten



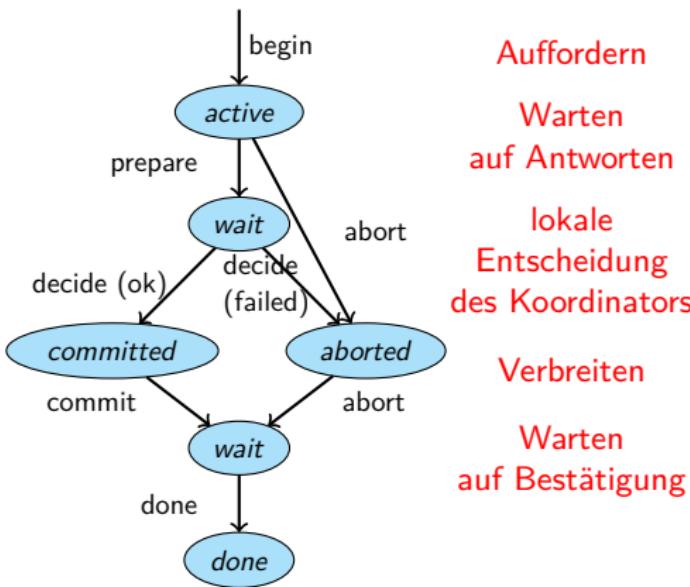
2-P-C-Protokoll (2)

Zustandsdiagramm des Koordinators

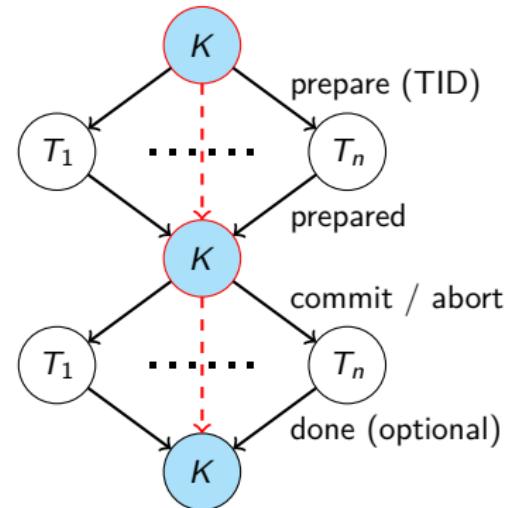


2-P-C-Protokoll (2)

Zustandsdiagramm des Koordinators

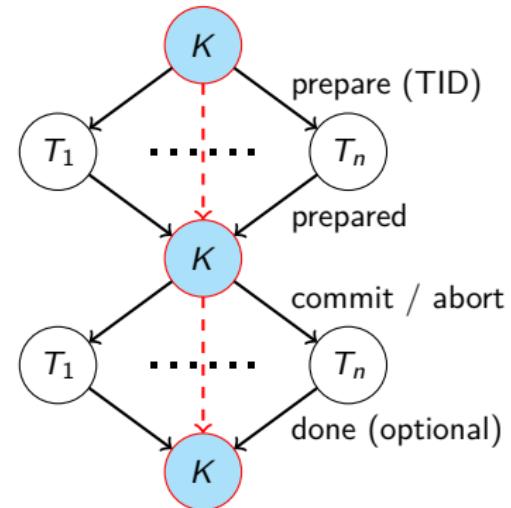
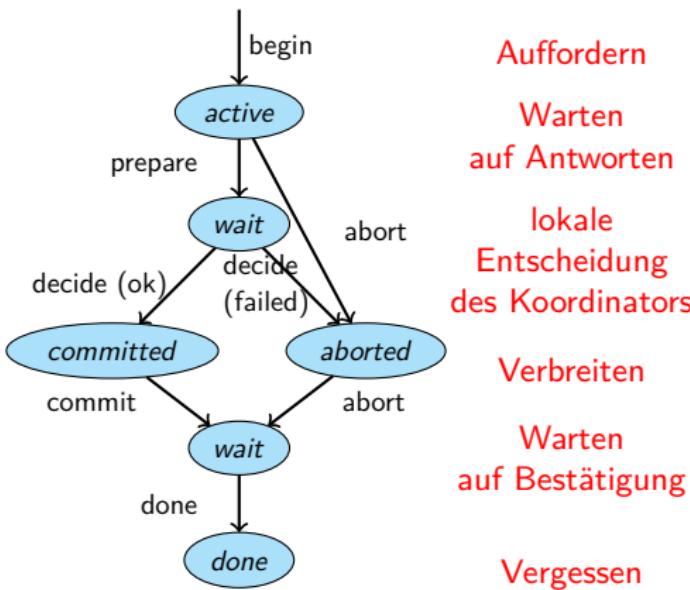


Auffordern
Warten auf Antworten
lokale Entscheidung des Koordinators
Verbreiten
Warten auf Bestätigung



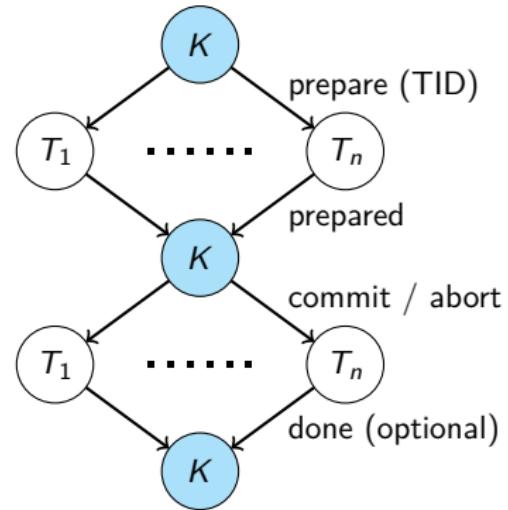
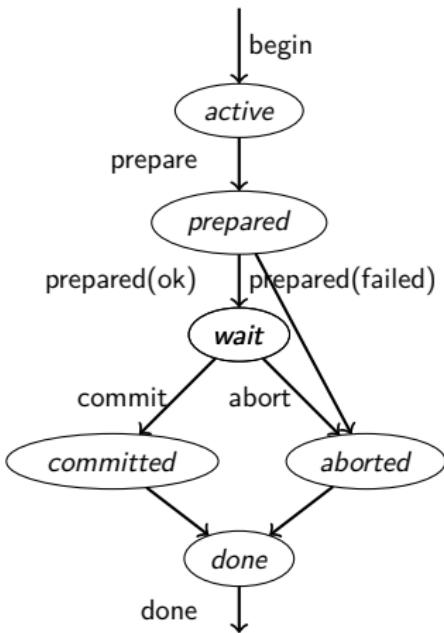
2-P-C-Protokoll (2)

Zustandsdiagramm des Koordinators



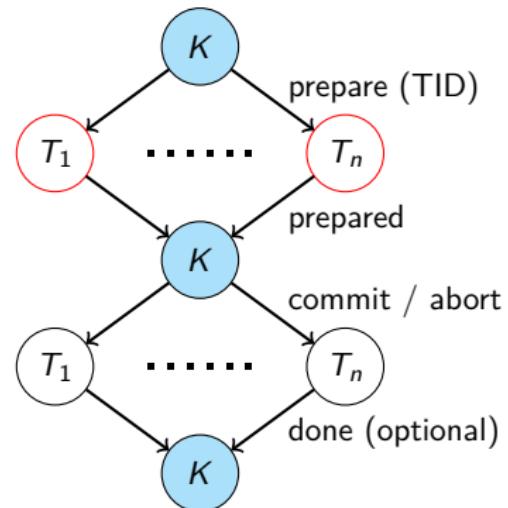
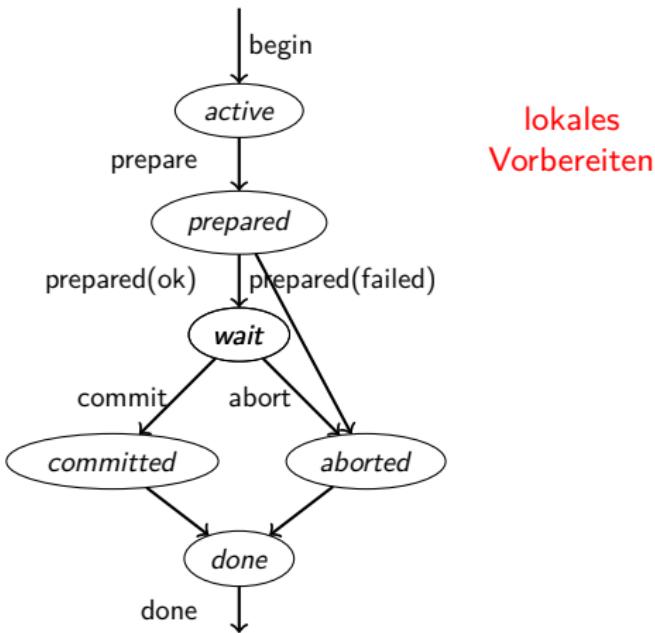
2-P-C-Protokoll (3)

Zustandsdiagramm der Teilhaber



2-P-C-Protokoll (3)

Zustandsdiagramm der Teilhaber

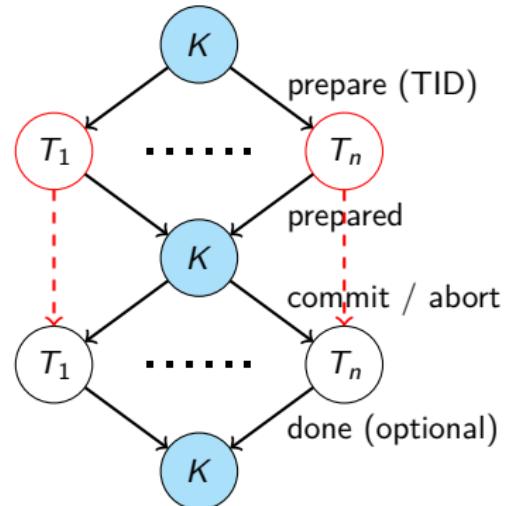


2-P-C-Protokoll (3)

Zustandsdiagramm der Teilhaber

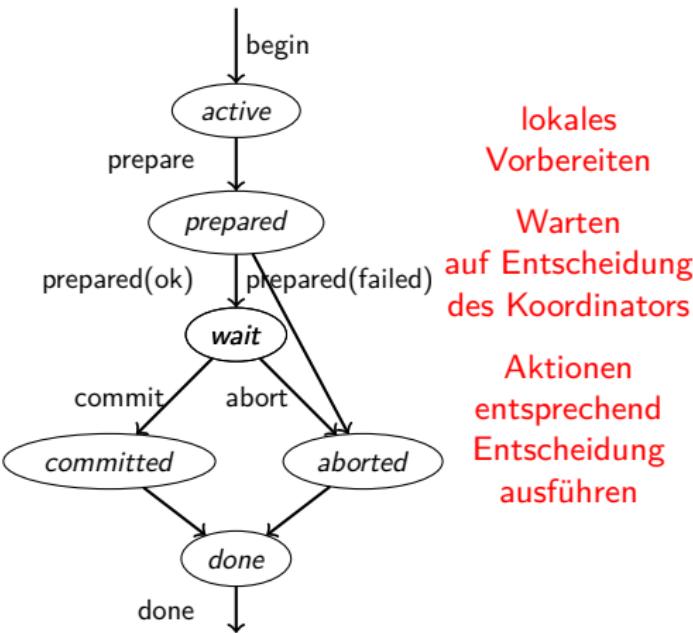


lokales
Vorbereiten
Warten
auf Entscheidung
des Koordinators

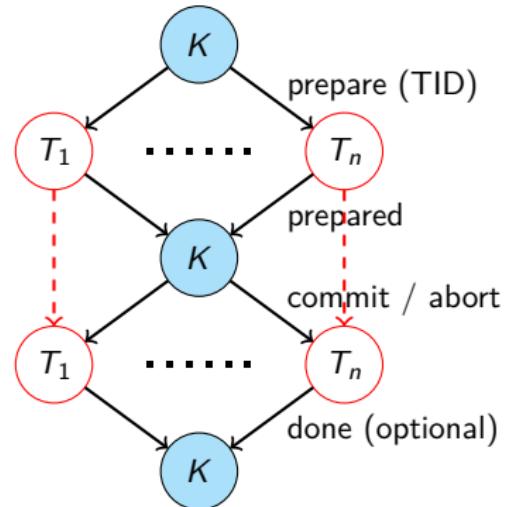


2-P-C-Protokoll (3)

Zustandsdiagramm der Teilhaber

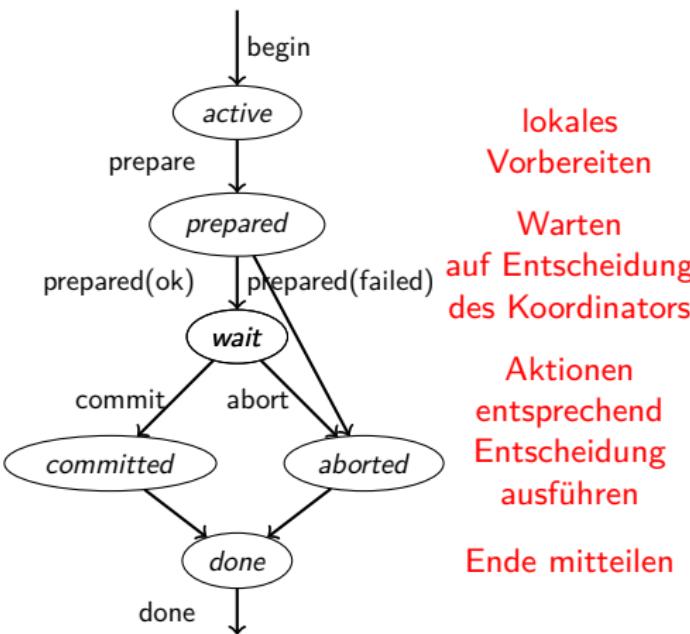


lokales Vorbereiten
 Warten auf Entscheidung des Koordinators
 Aktionen entsprechend Entscheidung ausführen

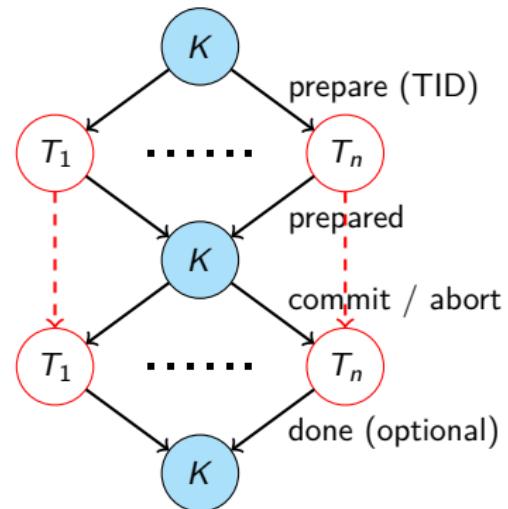


2-P-C-Protokoll (3)

Zustandsdiagramm der Teilhaber

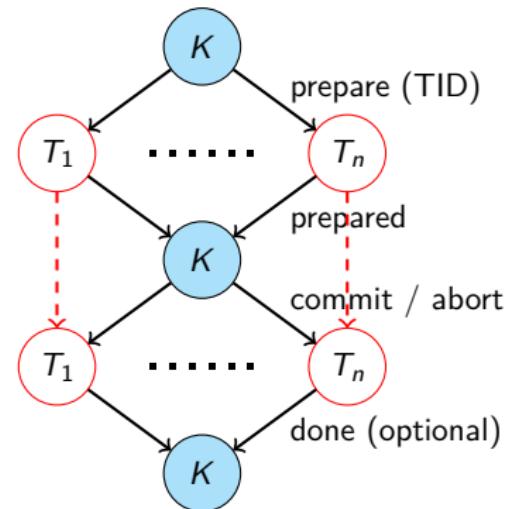
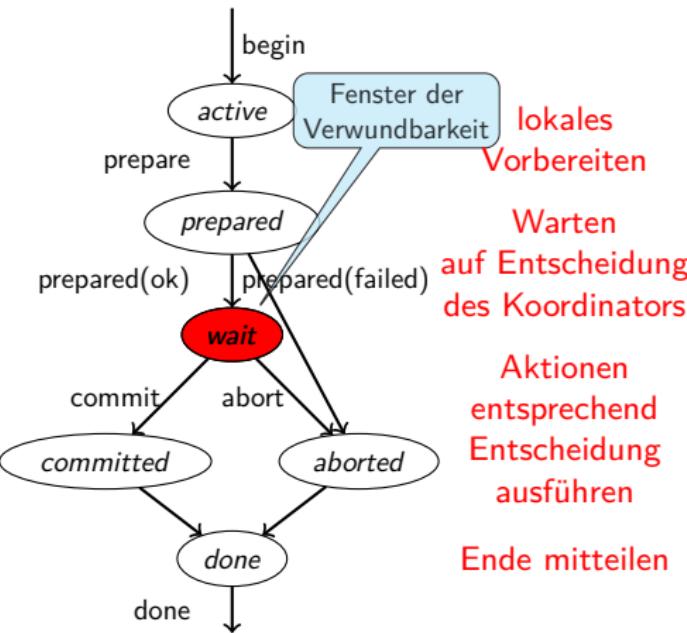


lokales Vorbereiten
 Warten auf Entscheidung des Koordinators
 Aktionen entsprechend Entscheidung ausführen
 Ende mitteilen



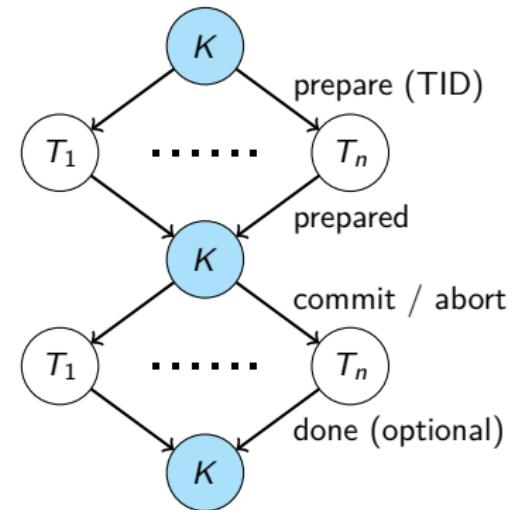
2-P-C-Protokoll (3)

Zustandsdiagramm der Teilhaber



2-P-C-Protokoll (4)

Logging des Koordinators

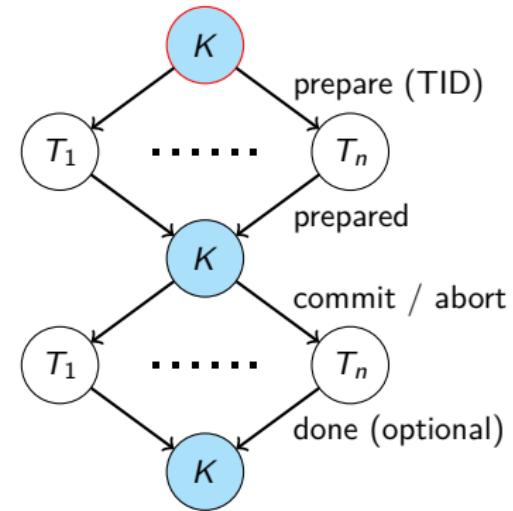


2-P-C-Protokoll (4)

Logging des Koordinators

$TID : begin(T_1, \dots, T_n)$

Auffordern

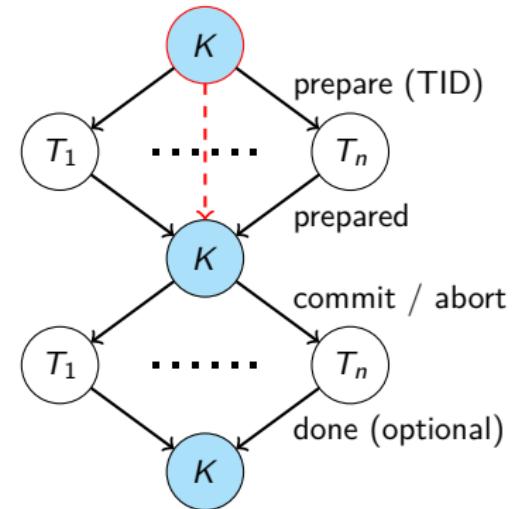


2-P-C-Protokoll (4)

Logging des Koordinators

$TID : begin(T_1, \dots, T_n)$

Auffordern
Warten
auf Antworten



2-P-C-Protokoll (4)

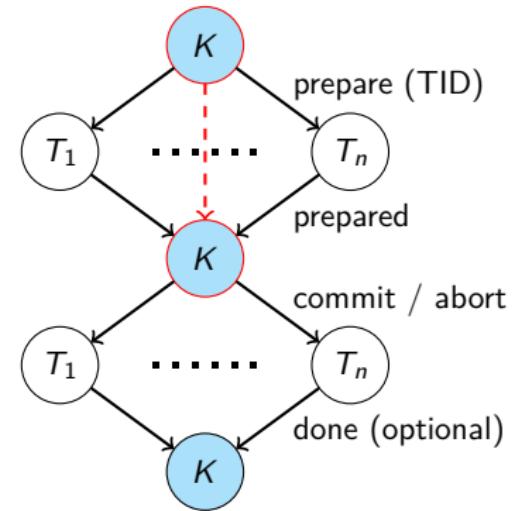
Logging des Koordinators

TID : begin(T_1, \dots, T_n)

TID : committed

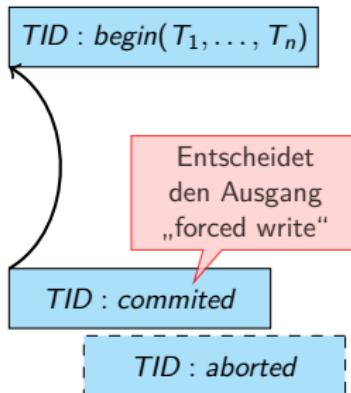
TID : aborted

Auffordern
Warten
auf Antworten
lokale
Entscheidung
des Koordinators

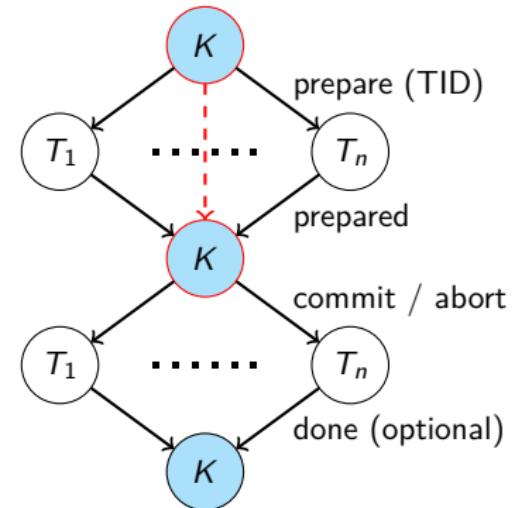


2-P-C-Protokoll (4)

Logging des Koordinators

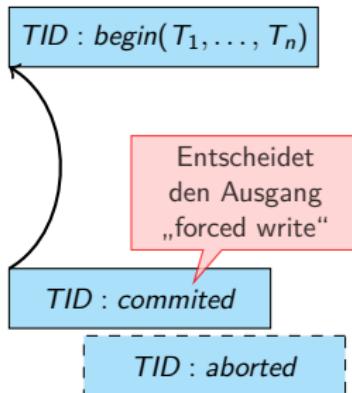


Auffordern
Warten
auf Antworten
lokale
Entscheidung
des Koordinators

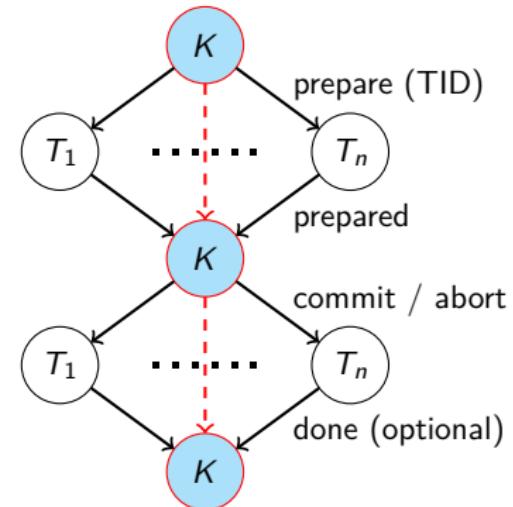


2-P-C-Protokoll (4)

Logging des Koordinators

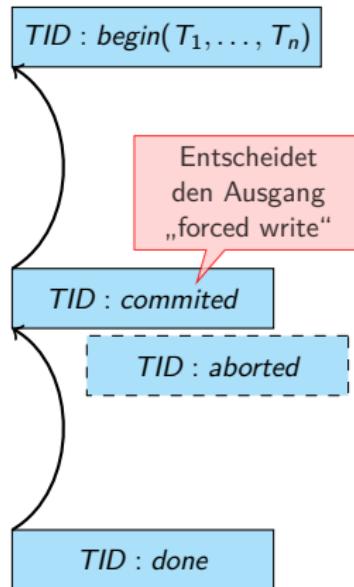


Auffordern
Warten auf Antworten
lokale Entscheidung des Koordinators
Verbreiten
Warten auf Bestätigung

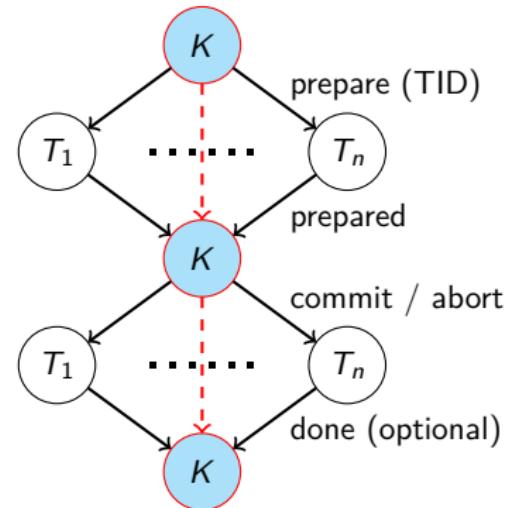


2-P-C-Protokoll (4)

Logging des Koordinators

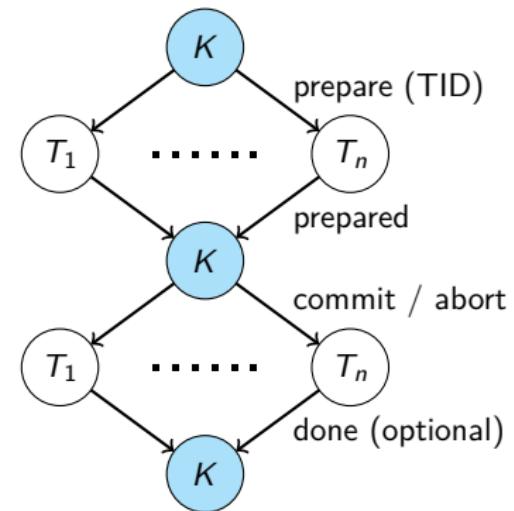


Auffordern
Warten auf Antworten
lokale Entscheidung des Koordinators
Verbreiten
Warten auf Bestätigung
Vergessen



2-P-C-Protokoll (5)

Logging der Teilhaber



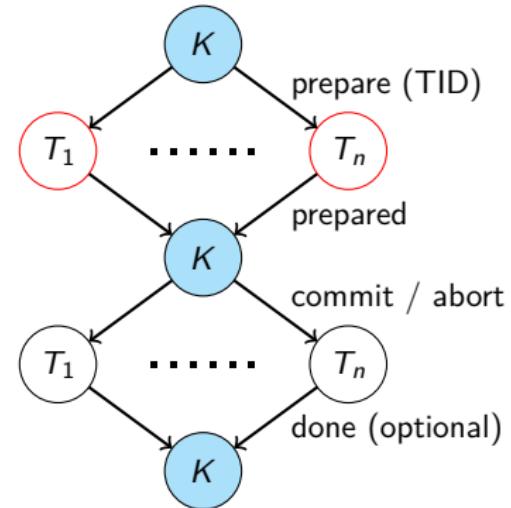
entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

Logging der Teilhaber

$TID : begin(coordK)$

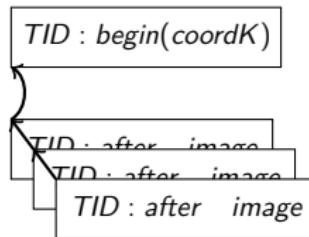
lokales
Vorbereiten



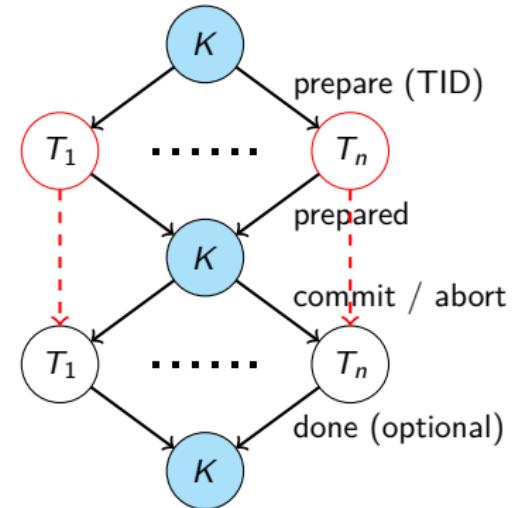
entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

Logging der Teilhaber



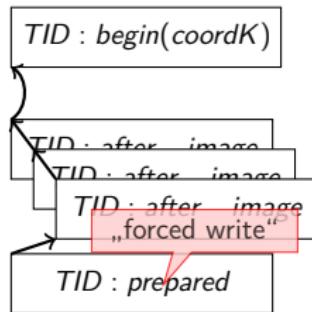
lokales
Vorbereiten
Warten
auf Entscheidung
des Koordinators



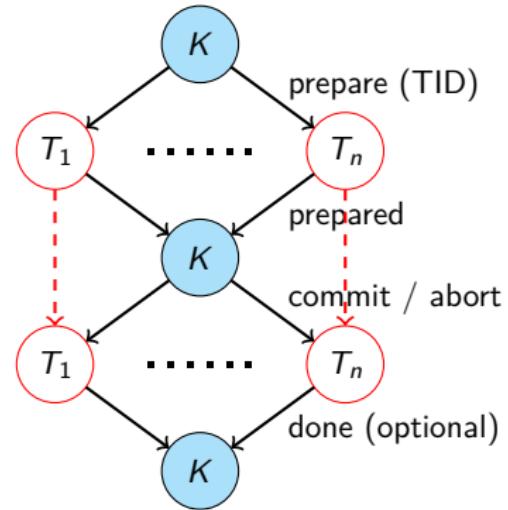
entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

Logging der Teilhaber



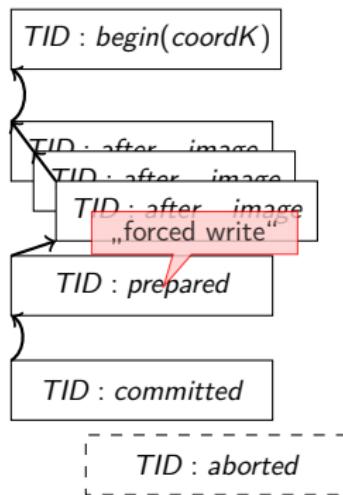
lokales
Vorbereiten
Warten
auf Entscheidung
des Koordinators



entsprechend bei einseitigem Abort

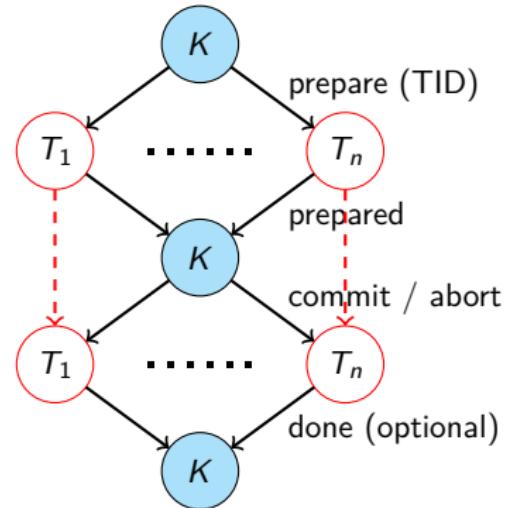
2-P-C-Protokoll (5)

Logging der Teilhaber



lokales
Vorbereiten
Warten
auf Entscheidung
des Koordinators

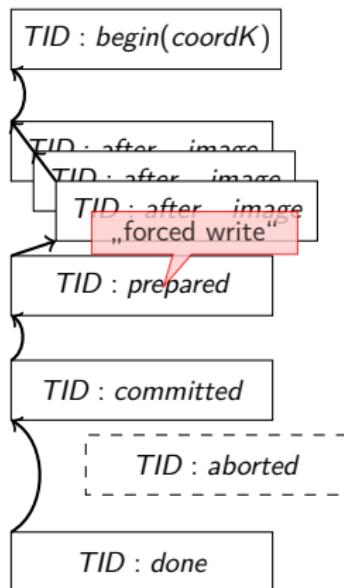
Aktionen
entsprechend
Entscheidung
ausführen



entsprechend bei einseitigem Abort

2-P-C-Protokoll (5)

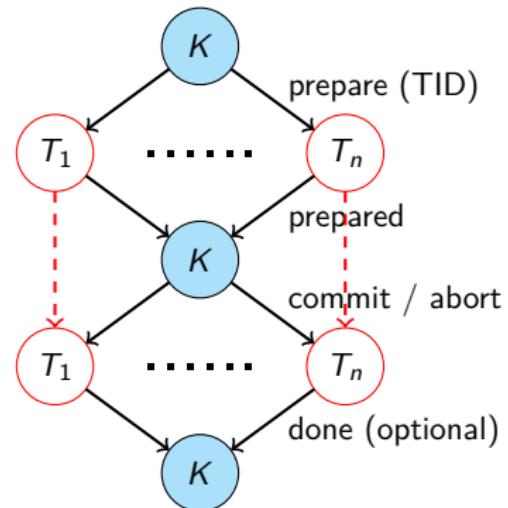
Logging der Teilhaber



lokales
Vorbereiten
Warten
auf Entscheidung
des Koordinators

Aktionen
entsprechend
Entscheidung
ausführen

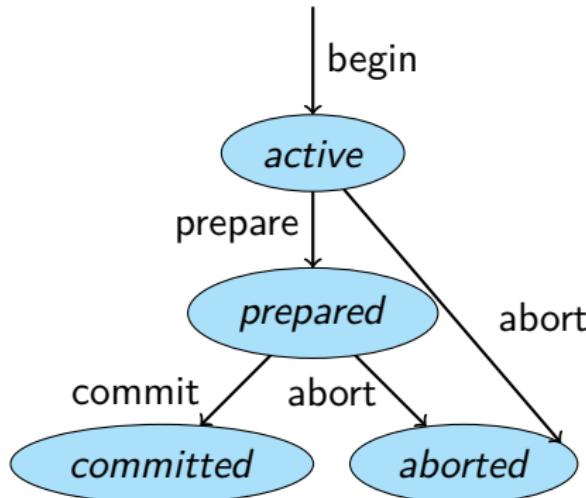
Ende mitteilen



entsprechend bei einseitigem Abort

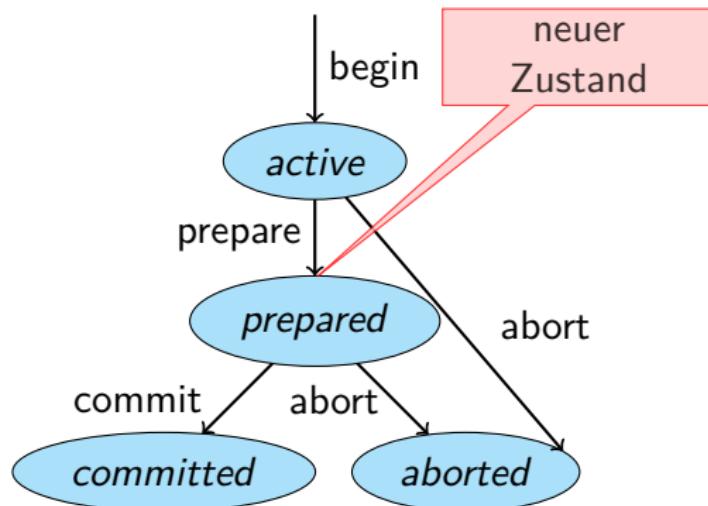
2-P-C-Protokoll (6)

zusammenfassendes Zustandsdiagramm für verteilte Transaktionen



2-P-C-Protokoll (6)

zusammenfassendes Zustandsdiagramm für verteilte Transaktionen



2-P-C-Protokoll (7)

Behandlung von Transaktionsfehlern

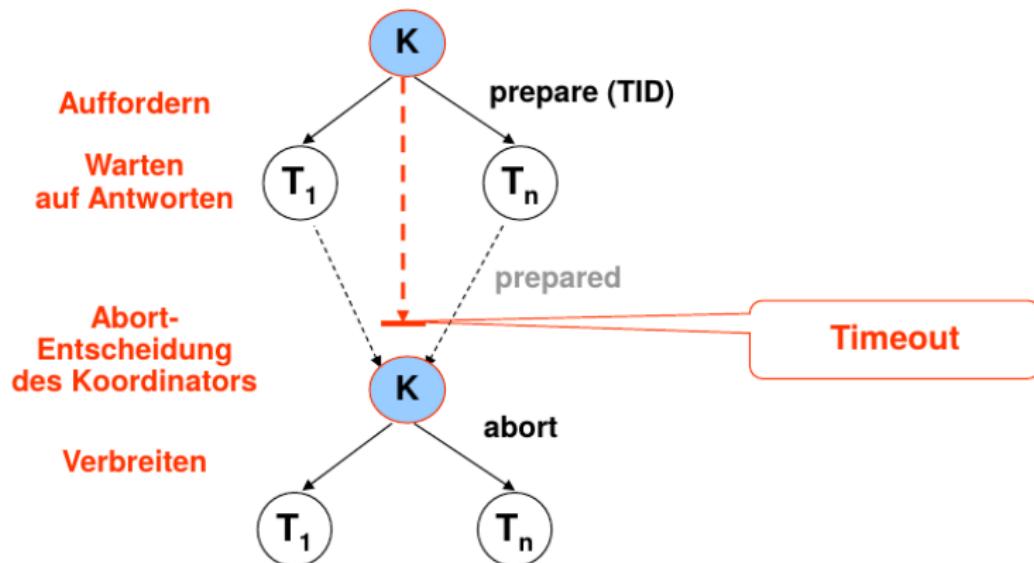
- Übergang von aktiven Transaktionen in den Zustand aborted
- unilaterale Abort-Entscheidung von Koordinator und jedem Teilhaber möglich
 - ▶ Koordinator sendet später abort, auch wenn alle Teilhaber mit prepared geantwortet haben
 - ▶ Teilhaber antworten auf prepare-Request des Koordinators mit prepared (failed).

Behandlung von Kommunikationsfehlern bei aktiven Transaktionen

- Erkennung (wie auch anderer Vorkommnisse) durch Timeouts
- Überführung in Transaktionsfehler mit Behandlung wie oben

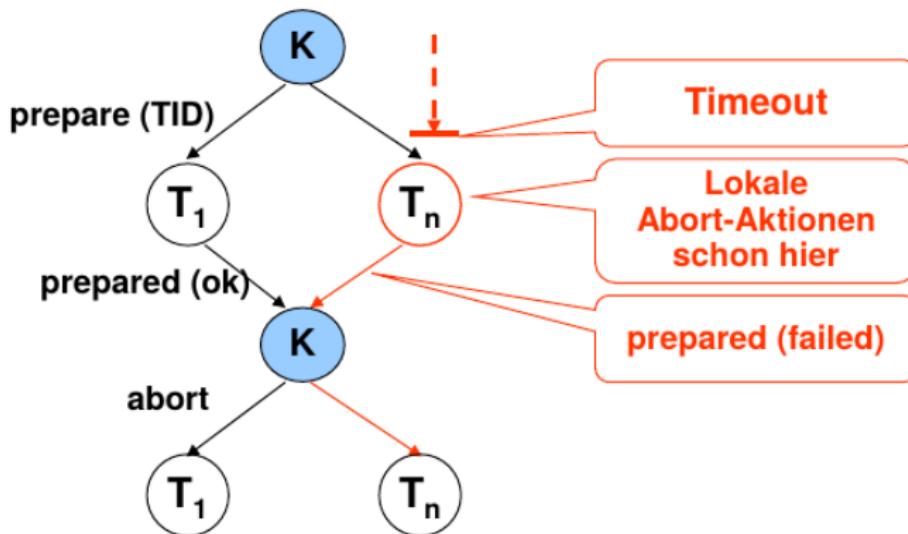
2-P-C-Protokoll (8)

Beispiel für Abort-Entscheidung des Koordinators



2-P-C-Protokoll (9)

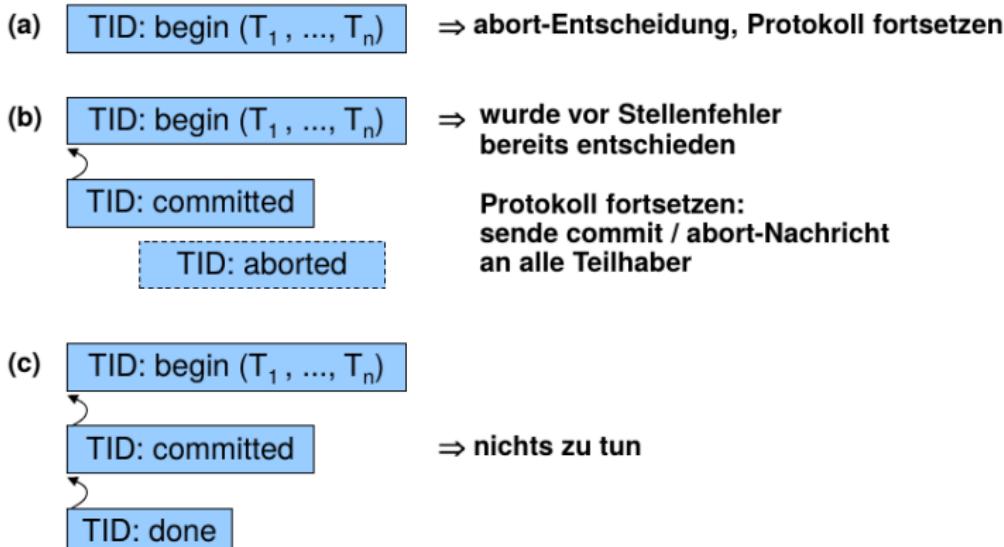
Beispiel für Abort-Entscheidung eines Teilhabers



2-P-C-Protokoll (10)

Behandlung eines Stellenfehlers des Koordinators

- Vorgehensweise abhängig von Information im Log

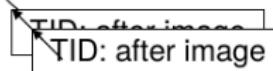


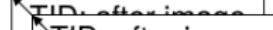
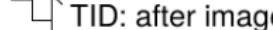
2-P-C-Protokoll (11)

Behandlung eines Stellenfehlers eines Teilhabers

- Vorgehensweise ebenfalls abhängig von Information im Log

(a)  TID: begin (coord K) \Rightarrow abort

(b)  TID: begin (coord K)
 TID: after image
 \Rightarrow abort

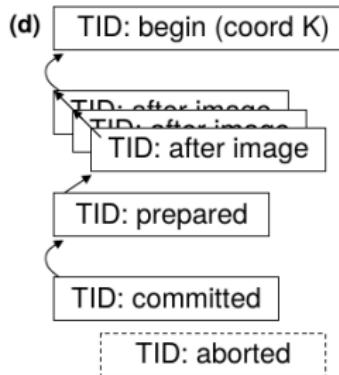
(c)  TID: begin (coord K)
 TID: after image
 TID: after image
 \Rightarrow keine unilaterale Entscheidung möglich

Nachfrage bei Koordinator K
`getDecision(TID)`

Protokoll fortsetzen

2-P-C-Protokoll (12)

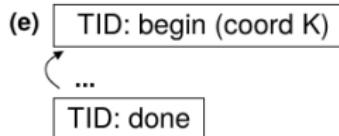
Behandlung eines Stellenfehlers eines Teilhabers (2)



⇒ unabhängiges Recovery möglich

lokale commit / abort-Aktionen wiederholen

Protokoll fortsetzen



⇒ nichts zu tun

Erweiterungen

Coordinator Migration

- Die Koordinator-Rolle wird einem hochzuverlässigen Rechner übertragen
- Damit wird Blockierung der Teilhaber bei Ausfall des Koordinators unwahrscheinlicher

Group Commitment

- Gemeinsames Commitment mehrerer Transaktionen
- Weniger forced write-Operationen
- Durchsatzsteigerung bei geringfügig erhöhter Ausführungszeit des Protokolls

Kooperative Terminierung

- Teilhaber kennen sich gegenseitig
- Nach Stellenfehler kann Nachfrage bei anderen Teilhabern die Blockierung vermindern, falls jetzt Koordinator ausgefallen ist

Erweiterungen (2)

Presumed Abort / Presumed Commit

- Default-Wert für Ausgang der Transaktion angenommen, wenn kein spezifischer Outcome Record im Log vorhanden ist
- Vereinfachungen bei Log-Verkürzung möglich

Dezentrales 2-Phasen-Commit-Protokoll

- Kein zentraler Koordinator
- Kommunikation der Teilhaber untereinander, z.B. vorteilhaft über Broadcast-Protokoll
- Verringert Zeitkomplexität

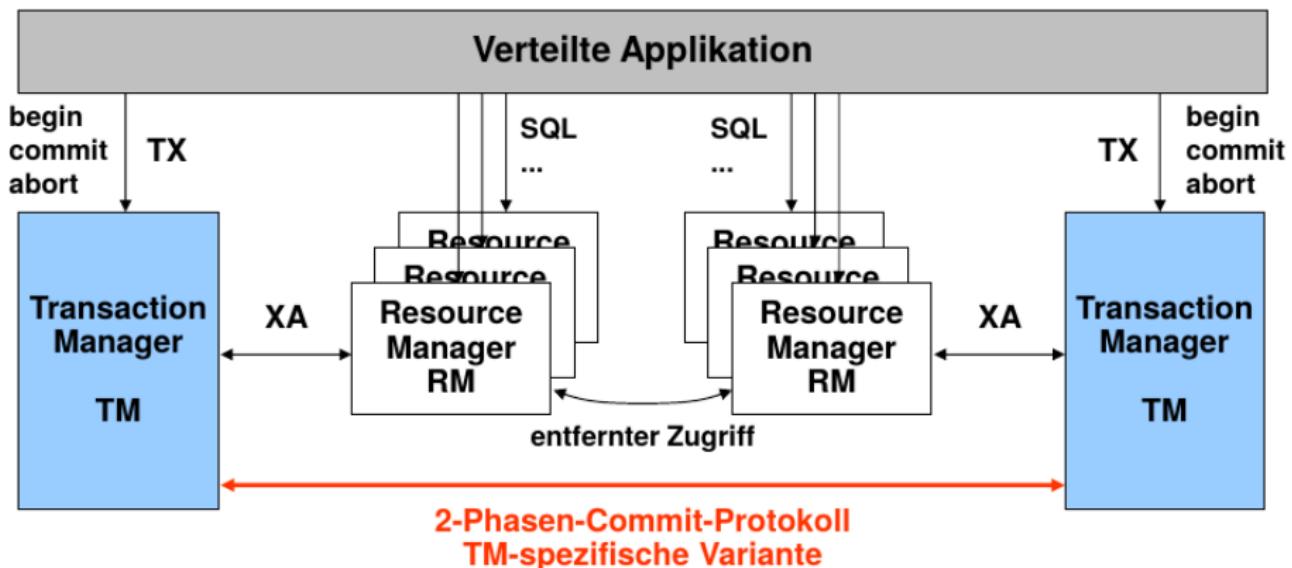
Anmerkung: von Erweiterungen ist nur Coordinator Migration prüfungsrelevant

X/Open XA

- X/Open
 - ▶ Historisches Industriekonsortium zur Förderung offener Systeme
 - ▶ Bildet 1996 zusammen mit Open Software Foundation (OSF) die Open Group
- Modell für Transaktionsverarbeitung in offenen verteilten Umgebungen
- Überwindung von Hersteller-spezifischen Lösungen der Transaktionssteuerung
- Anwendung des 2-Phasen-Commit-Protokolls
- Kommerziell ausgerichtet auf RDBMS-e
- Problem klassischer lokaler Datenbanksysteme
 - ▶ enge innere Verquickung der verschiedenen Funktionsbereiche
 - ▶ Unterstützung für verteilte Transaktionen erfordert Auftrennung zwischen Prepare-Aktivitäten und Commit-Entscheidung
 - ▶ Commit-Entscheidung muss auch extern getroffen werden können

Architekturüberblick

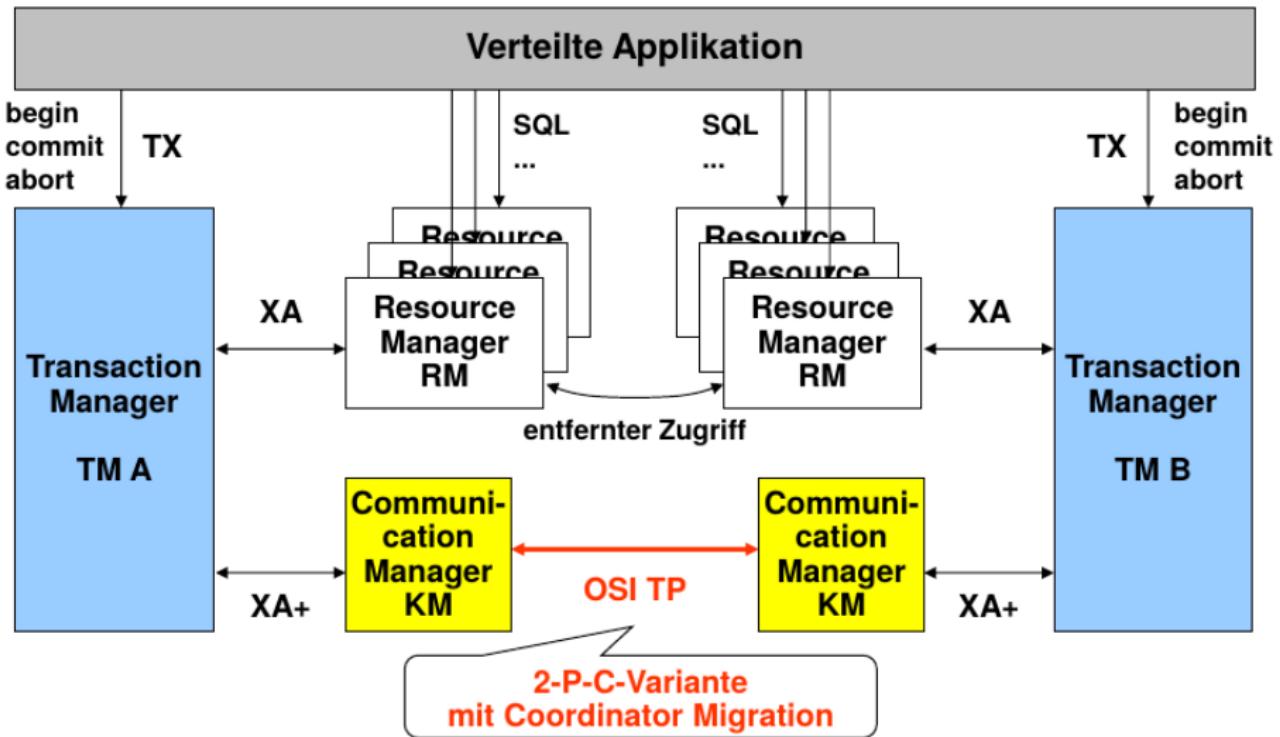
Homogener Fall



- RM kapseln log. Betriebsmittel (DB-Tupel, Dateien, Messages, Objekte, ...)
- TM kapselt Transaktionssteuerung
- für identische TMs, unterschiedliche RMs

Architekturüberblick (2)

Heterogener Fall



Beispiele

X/Open DTP-konforme Transaction Manager

- Tuxedo
 - ▶ historischer De-Facto-Standard
 - ▶ von USL, jetzt BEA, jetzt Oracle
 - ▶ nur flache Transaktionen
- Encina
 - ▶ unterstützt auch genestete Transaktionen
 - ▶ nutzt OSF DCE
 - ▶ von Transarc (Ausgründung CMU), jetzt eingegliedert in IBM
- CICS/6000
 - ▶ CICS: IBM Host-Standard
 - ▶ Encina für AIX (RS 6000)
- NCR Top End

Beispiele

Transaction Manager in CORBA- und J2EE-Produkten

- BEA Weblogic: Tuxedo
- IONA Orbix: Encina
- Heute i.d.R. in Applikationsservern integriert

X/Open DTP-konforme Resource Manager (XA-Schnittstelle)

- Relationale DBMS-e
 - ▶ historisch erstes System mit XA-Schnittstelle: Informix
 - ▶ heute von allen großen DB-Systemen unterstützt
 - ▶ selbst bei Microsoft (MS DTC)
- Message Queueing Systeme
 - ▶ MQ Series
 - ▶ Swift MQ
- Dateisysteme
 - ▶ ISAM XA (Gresham Computing)

Zusammenfassung

- Transaktionen dienen der reduzierten Komplexität verteilter Anwendungen.
- Das zugehörige Fehlermodell muss zwischen Transaktions-, Stellen-, Medien- und Kommunikationsfehlern unterscheiden.
- Das 2-Phasen-Commit-Protokoll ist das am meisten eingesetzte Commit-Protokoll.

9. Verteilte Dateisysteme



<https://tagebucheinesdeutschen.wordpress.com/2016/08/06/neues-von-der-scheinbehoerden-front-strategiewechsel-der-firma-finanzamt/comment-page-1/>

Inhalt

9. Verteilte Dateisysteme

9.1 Einführung

9.2 Grundlagen

9.3 NFS

9.4 AFS und Coda

9.5 Speichernetze

Einführung

Datenhaltungssysteme

	Dateisysteme	Datenbank-systeme	Objekt-management-systeme
Inhalt	universell	Massendaten weniger struktureller Typen	Beziehungen stehen im Vordergrund
Gespeicherte Informationen	passiv	passiv	aktiv
Semantik auf- prägender Code	extern	extern	intern (Typen)
Zugriff	über Namen, einfache Navigation	komplexe assoziative Suchfunktionen	komplexe Such- und Navigations- funktionen

Modelle von Dateisystemen

Dateien als klassische Abstraktion in Betriebssystemen
Historische Entwicklung im folgenden betrachtet

1 Rechner, 1 Benutzer, 1 Prozess

- zu lösende Probleme
 - ▶ Struktur des Dateisystems
 - ▶ Benennung (Naming)
 - ▶ Programmierschnittstelle
 - ▶ Abbildung auf physikalischen Speicher
 - ▶ Integrität
- Beispiele
 - ▶ PC-DOS
 - ▶ klassisches MacOS

Modelle von Dateisystemen (2)

1 Rechner, 1 Benutzer, mehrere Prozesse

- zusätzlich zu lösende Probleme
 - ▶ Nebenläufigkeitskontrolle
- Beispiel
 - ▶ OS/2

1 Rechner, mehrere Benutzer, mehrere Prozesse

- zusätzlich zu lösende Probleme
 - ▶ Sicherheit und Rechteverwaltung
- Beispiel
 - ▶ UNIX

mehrere Rechner, mehrere Benutzer, mehrere Prozesse

- im weiteren betrachtet
- zusätzlich zu lösende Probleme
 - ▶ Verteiltheit, d.h.
 - ★ sichtbare Gesamtstruktur
 - ★ Zugriffsmodell
 - ★ Aufenthaltsort
 - ★ Replikation
 - ★ Verfügbarkeit
 - ★ ...
 - ▶ Kein Zugriff auf gemeinsamen Block-Speicher der Knoten
(shared nothing)
 - ▶ Sharing gemeinsamer Platten zwischen den Knoten hier nicht weiter betrachtet
(vgl. 9.5: SAN Storage Area Networks) → Cluster File Systems
- Client/Server-Modell
 - ▶ ausgezeichneter File-Server
- Peer-to-Peer-Modell
 - ▶ jeder kann Dateien bereitstellen

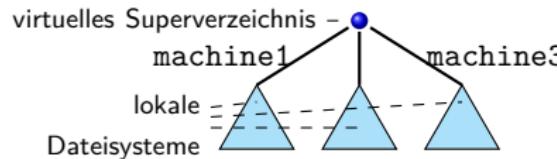
Historische Vorläufer

Völlige Trennung

- Ausschließlich lokale Zugriffe
- Dateitransfer zwischen isolierten Dateisystemen
(Download/Upload-Modell)
- Beispiel: UNIX uucp, ftp, rcp, scp

Adjungierte Dateisysteme

- Zugriff auf entfernte Dateien
- Explizite Angabe des Aufenthaltsorts im Dateinamen
- Beispiel: Newcastle Connection



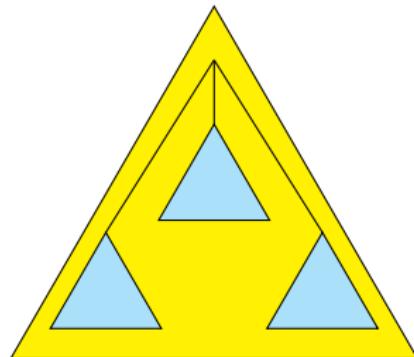
/machine1/<localpath>
machine2!<localpath>
/.../machine3/<localpath>

Verteiltes Dateisystem

Ein **verteiltes Dateisystem** ist ein Dateisystem, das seinen Nutzern auf allen Maschinen im Netz ein einheitliches Dateisystem zur Verfügung stellt

Mögliche Transparenzarten:

- Ortstransparenz
 - ▶ Dateiname enthält keine Ortsangabe
- Zugriffstransparenz
 - ▶ API unabhängig, ob Datei lokal oder entfernt
- Namenstransparenz
 - ▶ Dateiname an allen Stellen identisch
- Replikationstransparenz
- ...



Typische Designziele

- Hohes Maß an Transparenz
 - ▶ s.o.
- Performance
 - ▶ ähnlich lokalem Zugriff
- Hohe Verfügbarkeit (Availability) / Fehlertoleranz
- Sicherheit
- Skalierbarkeit
- Unterstützung für mobile Knoten mit zeitweiser Diskonnektivität
- Unterstützung für shared disk und shared nothing
(lose gekoppelte) Knoten
- Cloud-Anbindung

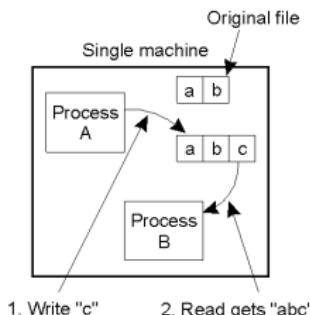
Lösungsüberblick

- Network File System (NFS) — ab 1985
- Andrew File System (AFS) + Coda — ab etwa 1985
- Common Internet File System (CIFS) + Server Message Block (SMB)
- GlusterFS (Gluster Inc. → Red Hat 2011)
- IBM General Parallel File System (GPFS) (ursprünglich Cluster File System, weiterentwickelt)
- Google File System (GFS)
- Apache Hadoop
- ...

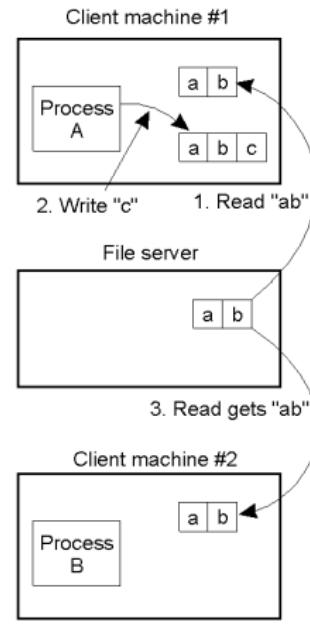
Grundlagen

Zugriffskonsistenzproblem

- (a): Änderungen für jeden unmittelbar sichtbar und damit aktuell
- (b): sichtbare Werte können veraltet sein



(a)



(b)

Abb. aus Tanenbaum/Steen

Semantiken

Strenge Konsistenz

- Änderungen unmittelbar für alle anderen sichtbar
- Beispiel: lokales UNIX

Sitzungssemantik

- Aktualisierung der Datei beim Schließen
- erlaubt lokalen Cache, solange Datei geöffnet ist
- Beispiel: Andrew File System

Read-Only Dateien

- Änderungen sind nicht möglich
- Gemeinsame Nutzung und Replikation werden deutlich vereinfacht

Transaktionssemantik

- Änderungen auf einer Menge von Dateien finden atomar statt
(vgl. Kap. 8)

Zustandslose / zustandsbehaftete Server

Zustandlosigkeit

- Server hat kein Gedächtnis

Vorteile zustandsloser Server

- Recovery leicht realisierbar
- Keine Probleme mit Client-Abstürzen
- Öffnen / Schließen von Dateien unnötig
- Anzahl offener Dateien unbegrenzt

Vorteile zustandsbehafteter Server

- kürzere Nachrichten
- höhere Performance
- Read-ahead möglich
- Idempotenz von Operationen leichter realisierbar
- Dateisperren möglich

Replikation

Ziele

- Verfügbarkeit der Daten
- Lastausgleich
- Transparenz aus Benutzersicht

Übliche Algorithmen

- hier nicht im Detail behandelt
- Primary Copy Update
 - ▶ Primary = ausgezeichnete Kopie
 - ▶ Veränderungen nur auf Primary, dieser kümmert sich um Kopien
 - ▶ Primary sieht damit alle Updates
 - ▶ Lesen von beliebiger Kopie (Performance)
- Voting (Gifford)
 - ▶ Lese-Quorum und Schreib-Quorum haben einen nicht-leeren Schnitt
- Multiple Copy Update
 - ▶ vgl. Problematik in Datenbanken (z.B. Bernstein, Goodman, 1984)

Network File System (NFS)

Designziele (1985)

- Sharing in einem Netzwerk heterogener Systeme
 - ▶ Ausgangspunkt: Diskless Workstations
- Zugriffstransparenz
 - ▶ keine speziellen Pfadnamen, Bibliotheken oder Rekomplilation
- Portabilität
 - ▶ Festlegung von NFS als Schnittstelle
 - ▶ Implementierung von Client- und Server-Seite kann unterschiedlich sein
- Einfache Behandlung von Stellenausfällen
 - ▶ Zustandslosigkeit des Servers
- Performance
 - ▶ äquivalent zu lokalem Plattenzugriff
- Industriestandard
 - ▶ durch Offenlegung von Schnittstelle und Referenz-Implementierung

Gesamtarchitektur

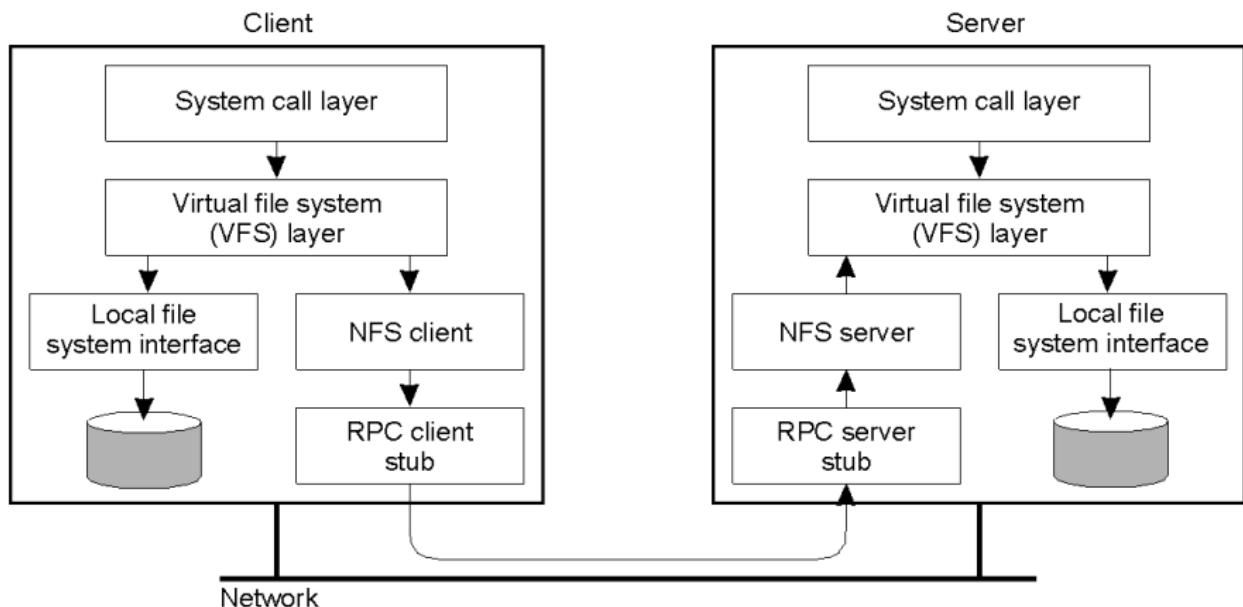


Abb. aus Tanenbaum/Steen

Gesamtarchitektur

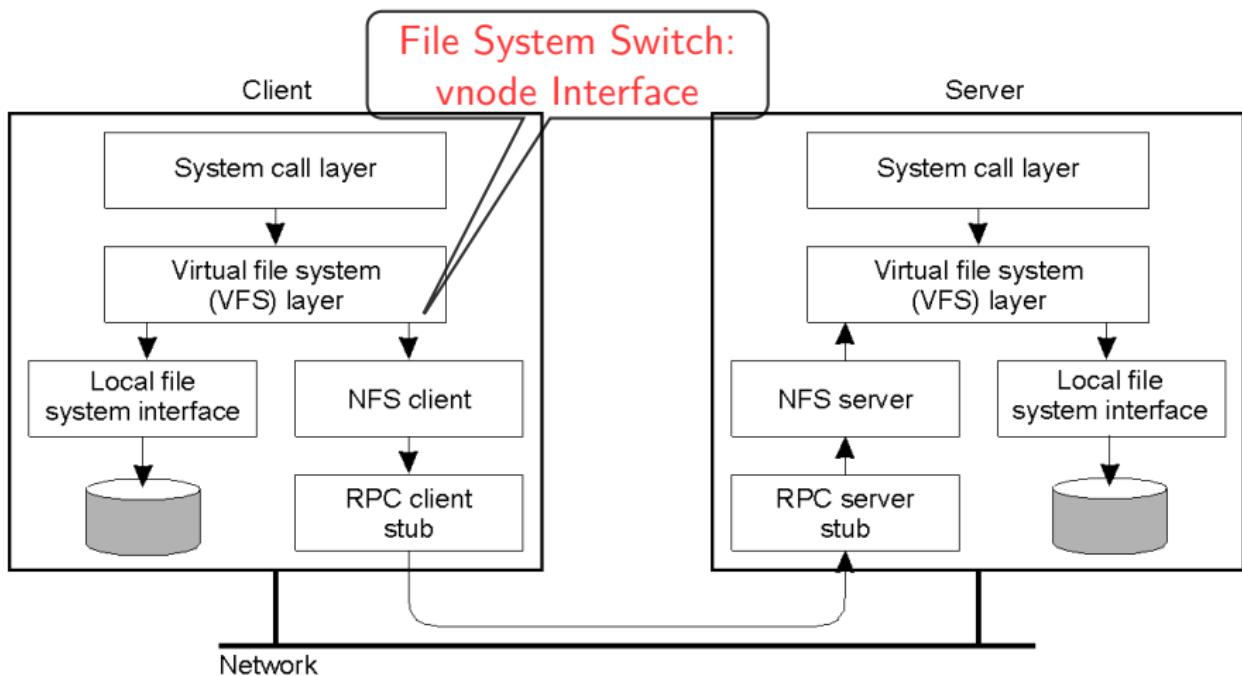


Abb. aus Tanenbaum/Steen

Funktionsweise

Rollen

- Jeder Knoten kann gleichzeitig Client und Server sein
- Jeder NFS-Server exportiert ein oder mehrere Verzeichnisse (mit dem gesamten Unterbaum)
- Gemeinsame Nutzung durch mehrere Clients möglich
- Client-Zugriff erfordert Mounting

Naming

- hierarchischer UNIX-Pfadnamensraum
- Ortstransparenz entsteht nur durch Konvention
 - ▶ nicht erzwungen
 - ▶ Mount-Punkte können prinzipiell beliebig benannt werden

Locating

- lokale Mount-Tabelle im Betriebssystem
- daher kein Protokoll zur Aufenthaltsortsbestimmung notwendig

Verzeichnisstruktur

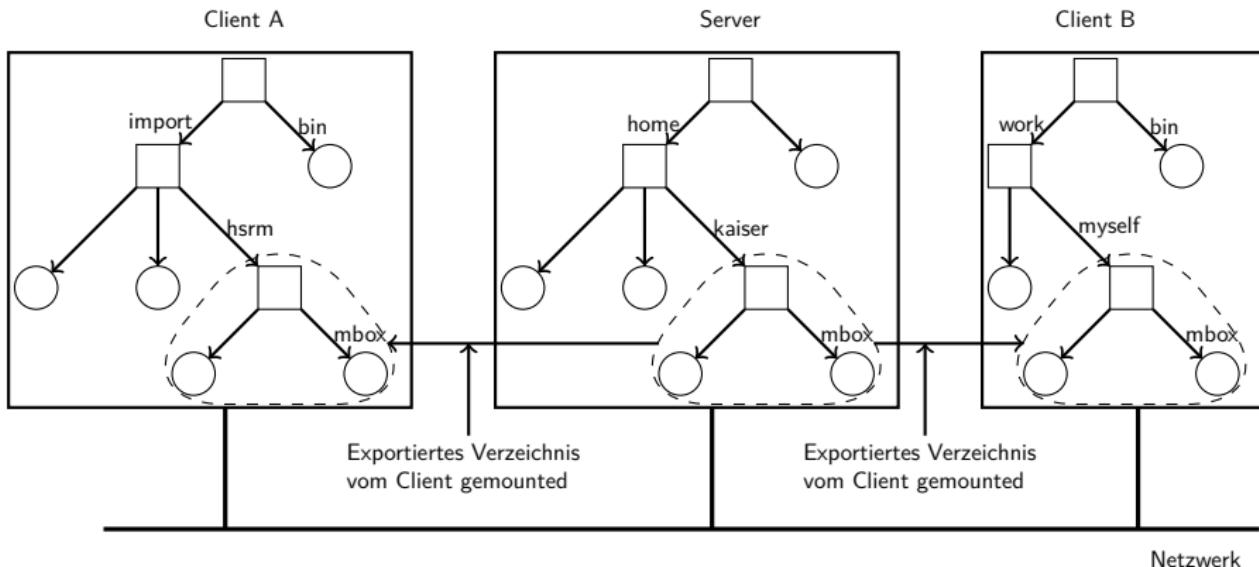


Abb. nach Tanenbaum/Steen

Verzeichnisstruktur

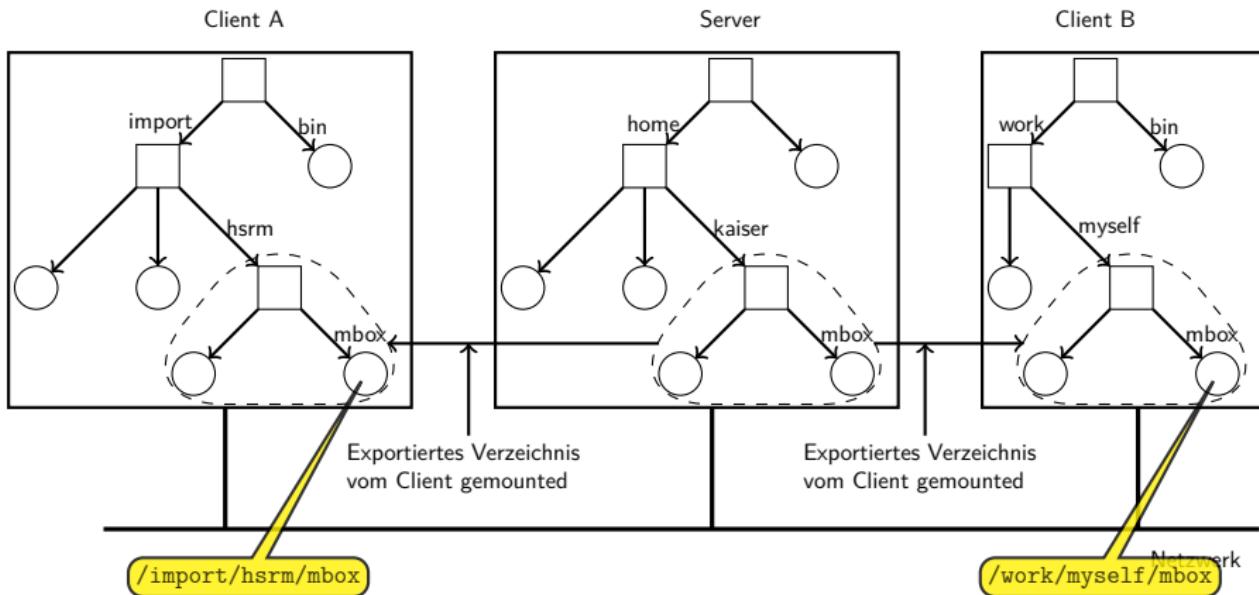
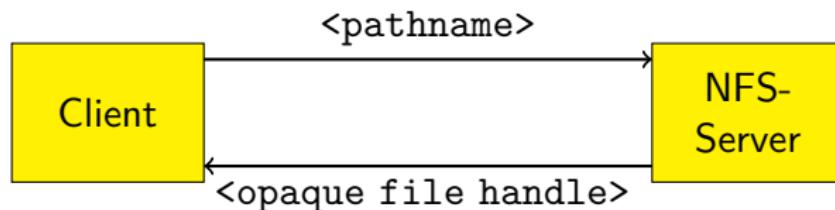


Abb. nach Tanenbaum/Steen

Mount-Protokoll

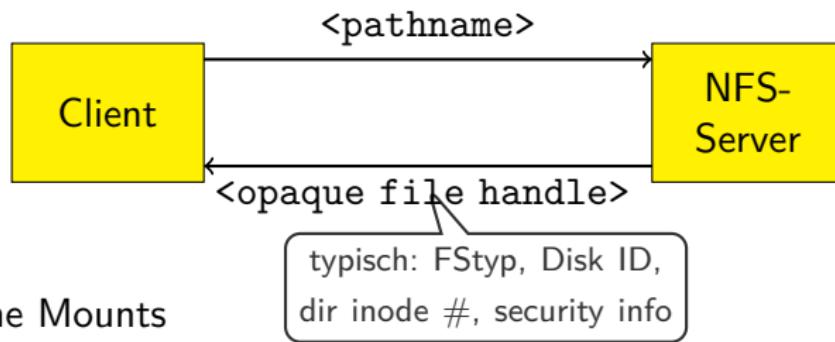
- existiert bis zur aktuellen Version 3 als Teilprotokoll
- in Version 4 in allgemeines Zugriffsprotokoll integriert



- Statische Mounts
 - ▶ zum Boot-Zeitpunkt ausgeführt
- Problem:
 - ▶ u.U. nicht-verfügbarer Server zum Mount-Zeitpunkt
 - Client kann nicht ohne Probleme booten

Mount-Protokoll

- existiert bis zur aktuellen Version 3 als Teilprotokoll
- in Version 4 in allgemeines Zugriffsprotokoll integriert



- Statische Mounts
 - ▶ zum Boot-Zeitpunkt ausgeführt
- Problem:
 - ▶ u.U. nicht-verfügbarer Server zum Mount-Zeitpunkt
 - Client kann nicht ohne Probleme booten

Automounter

Zur Lösung der Probleme statischer Mounts eingeführt Funktionsweise

- Zuordnung:
lokaler Mount-Punkt ↔ Menge von exportierten Verzeichnissen
- Keine Aktion zum Bootzeitpunkt
- Erster Zugriff unterhalb des Mount-Punkts bewirkt Nachricht an alle Server der Menge
- Wer zuerst antwortet, wird gemountet
 - ▶ Ausgefallene Server antworten nicht und können toleriert werden
 - ▶ Lastausgleich möglich
- Keine Unterstützung für allgemeine Replikation
 - ▶ daher oft nur für read-only-Dateisysteme genutzt (z.B. /usr)

Zugriffsprotokoll

Für Zugriffe auf Verzeichnisse und Dateien analog UNIX system calls

Unterschiede zwischen Version 3 und neuer Version 4

- Version 3 ist zustandslos
 - ▶ keine Unterstützung für open und close
 - ▶ read/write müssen notwendige Umgebung mit anliefern (file handle, offset, nbytes)
 - ▶ keine Sperren auf Dateien, sondern in separatem Lock-Server
- Version 4 ist **nicht** zustandslos !
 - ▶ Ziel: Effiziente Nutzung von NFS in Weitverkehrsnetzen ermöglichen
 - ▶ erfordert effizientes korrektes Caching auf Client-Seite
 - ▶ geht nicht zustandslos
 - ▶ auch Dateisperren werden erlaubt

Zugriffsprotokoll (2)

Unterlagertes Protokoll

- SunRPC (ONC RPC) mit XDR-Datenkodierung
- at-least-once-Semantik
- nutzt selbst UDP/IP

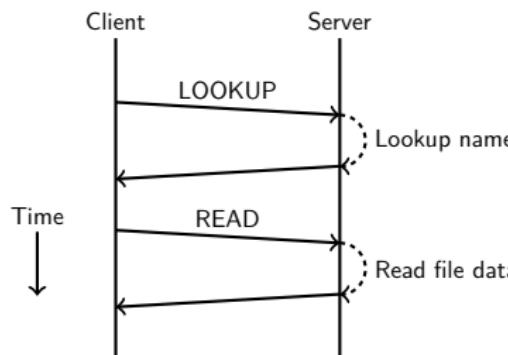
Dienstschnittstelle

Operation	v3	v4	Beschreibung
Create	Yes	No	Create a regular file
Create	No	Yes	Create a nonregular file
Link	Yes	Yes	Create a hard link to a file
Symlink	Yes	No	Create a symbolic link to a file
Mkdir	Yes	No	Create a subdirectory in a given directory
Mknod	Yes	No	Create a special file
Rename	Yes	Yes	Change the name of a file
Rmdir	Yes	No	Remove an empty subdirectory from a directory
Open	No	Yes	Open a file
Close	No	Yes	Close a file
Lookup	Yes	Yes	Look up a file by means of a file name
Readdir	Yes	Yes	Read the entries in a directory
Readlink	Yes	Yes	Read the path name stored in a symbolic link
Getattr	Yes	Yes	Read the attribute values for a file
Setattr	Yes	Yes	Set one or more attribute values for a file
Read	Yes	Yes	Read the data contained in a file
Write	Yes	Yes	Write data to a file

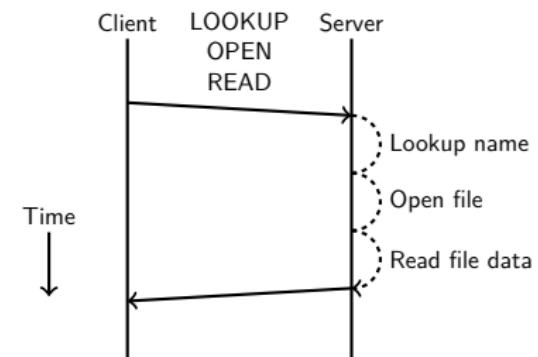
Zugriffsprotokoll (3)

Zusammengesetzte Prozeduren

- Performance-Optimierung in Version 4
- insbesondere in Weitverkehrsnetzen relevant
- keine Nebenläufigkeitskontrolle oder Atomarität



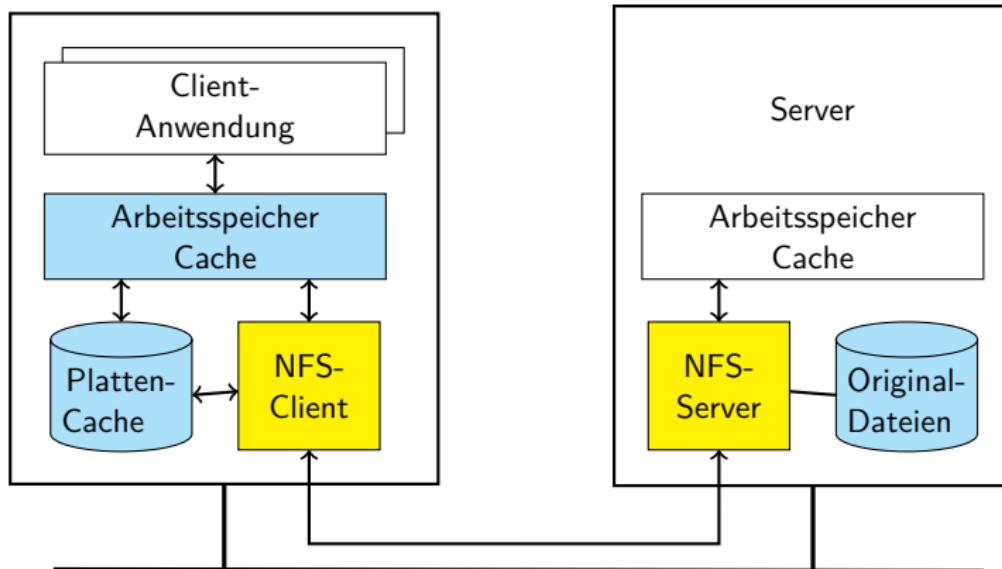
bisher in Version 3



Version 4

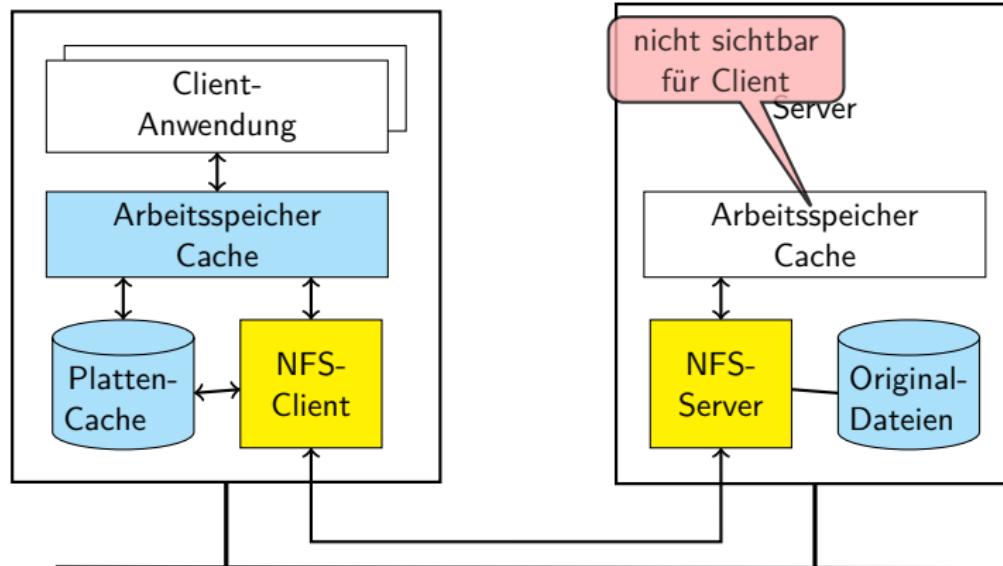
Caching

Client-seitiges Caching



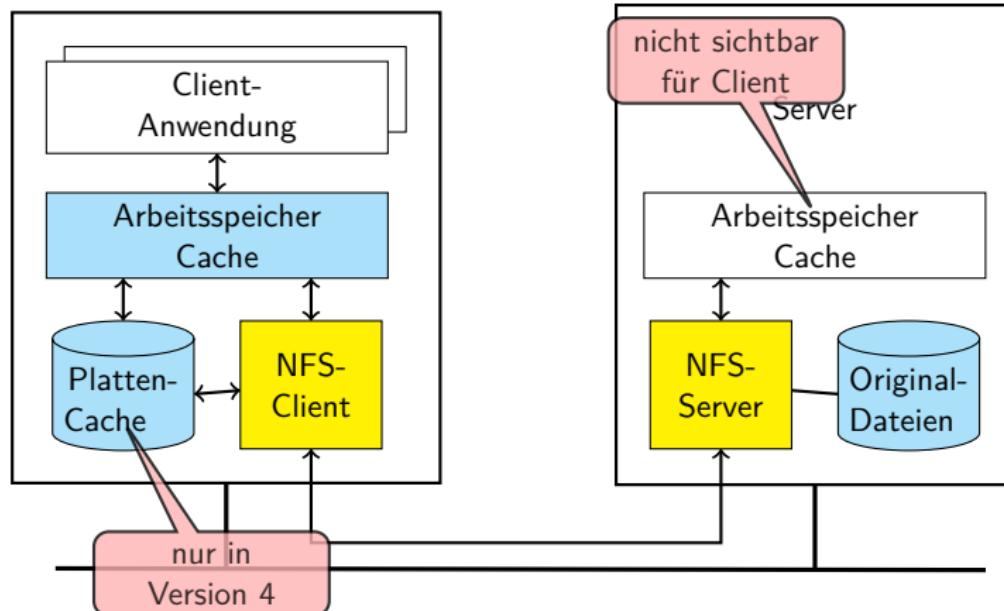
Caching

Client-seitiges Caching



Caching

Client-seitiges Caching



Caching (2)

Arbeitsspeicher-Cache

- Caching einzelner Blöcke entfernter Dateien
- große Blockgröße für effizienten Transfer, typisch 8 KB
- Read-ahead des nächsten Blocks
- Zugriffe auf ausführbare Dateien mit Größe < Schwellwert führt zu vollständiger Übertragung

Caching (3)

Cache-Kohärenz

- in Version 3 **nicht** gegeben
 - ▶ Problem: Mehrere Clients können Blöcke derselben Datei / desselben Verzeichnisses cachen und auch modifizieren
 - ▶ zeitmarkenbasiertes unsicheres Validierungsschema
 - ★ Validierung bei `open()`, Cache Miss und Timeout
(typisch: Dateien 3 Sek, Verzeichnisse 30 Sek)
 - ★ Nach Validierung wird Gültigkeit für eine Zeitspanne angenommen
 - ★ Write-Through für Blöcke von Verzeichnissen
 - ★ Alle modifizierten Blöcke werden spätestens bei `close()` an Server übertragen
 - ▶ Cache kann damit insgesamt veraltete Dateien und Verzeichnisse enthalten
 - ⇒ Kooperation von Prozessen über Dateisystem unter NFS 3 nicht immer korrekt
- in Version 4 gegeben
 - ▶ Cache-Invalidierung veralteter Daten, Überprüfung bei `open()`
 - ▶ Sitzungssemantik

Caching (4)

File Delegation

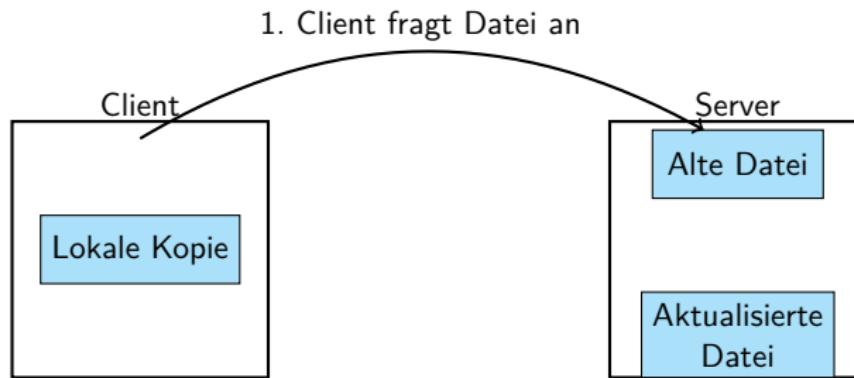
- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft open()- und close()-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig



Caching (4)

File Delegation

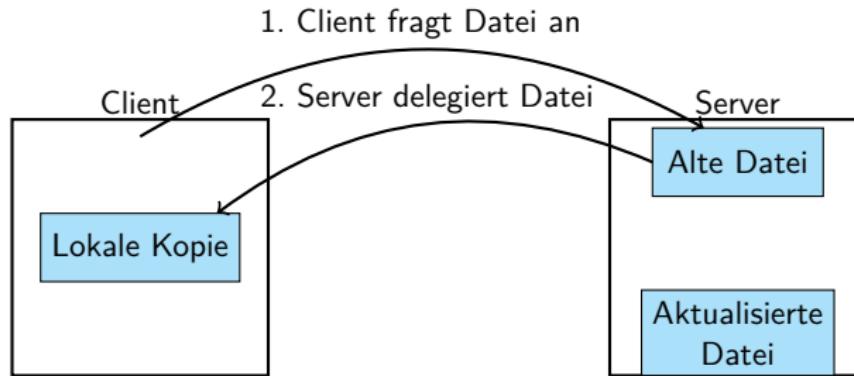
- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft open()- und close()-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig



Caching (4)

File Delegation

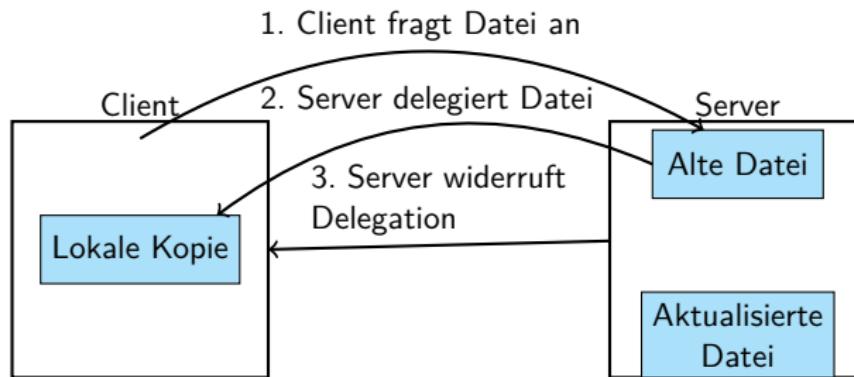
- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft open()- und close()-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig



Caching (4)

File Delegation

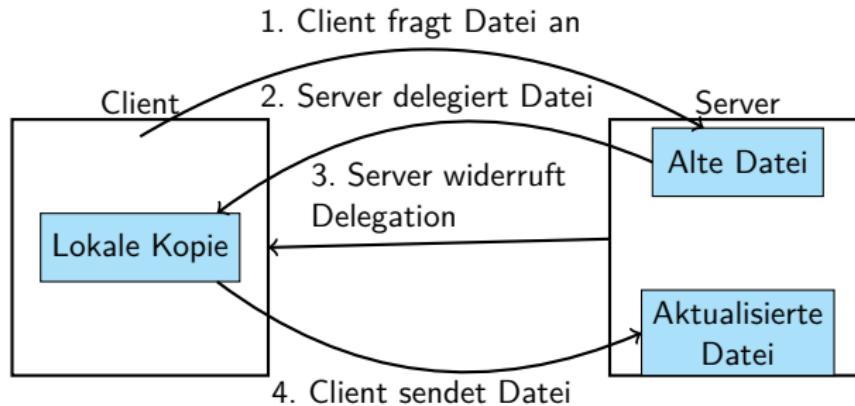
- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft open()- und close()-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig



Caching (4)

File Delegation

- nur in Version 4
- Delegation von Aufgaben des Servers an Client. Dieser überprüft open()- und close()-Operationen anderer Clients
- Möglichkeit zur Rücknahme der Delegation notwendig



Sicherheit

Prinzipielle Architektur

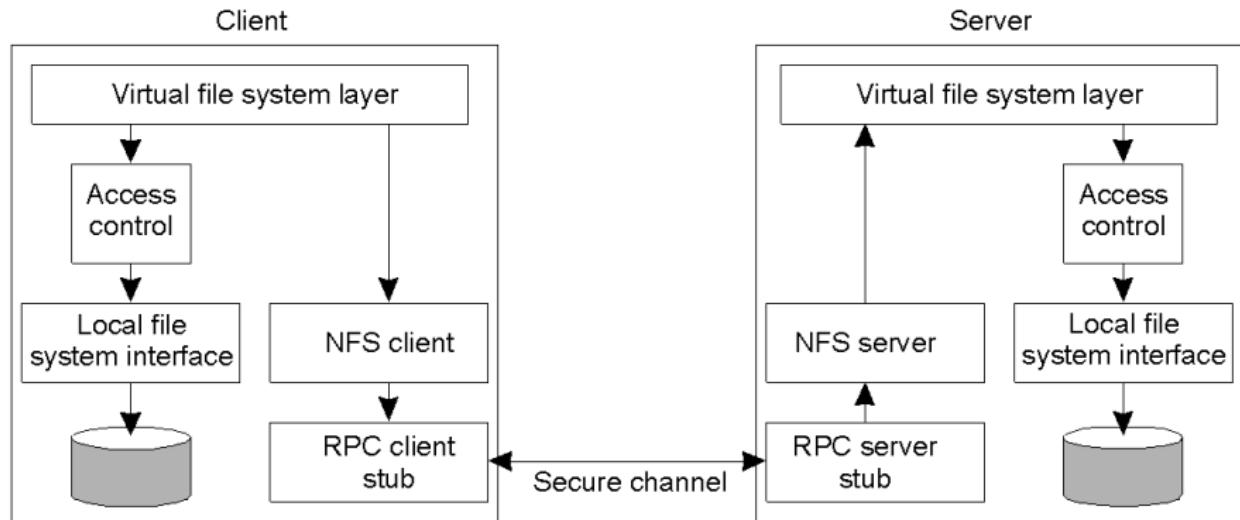


Abb. aus Tanenbaum/Steen

Sicherheit

Prinzipielle Architektur

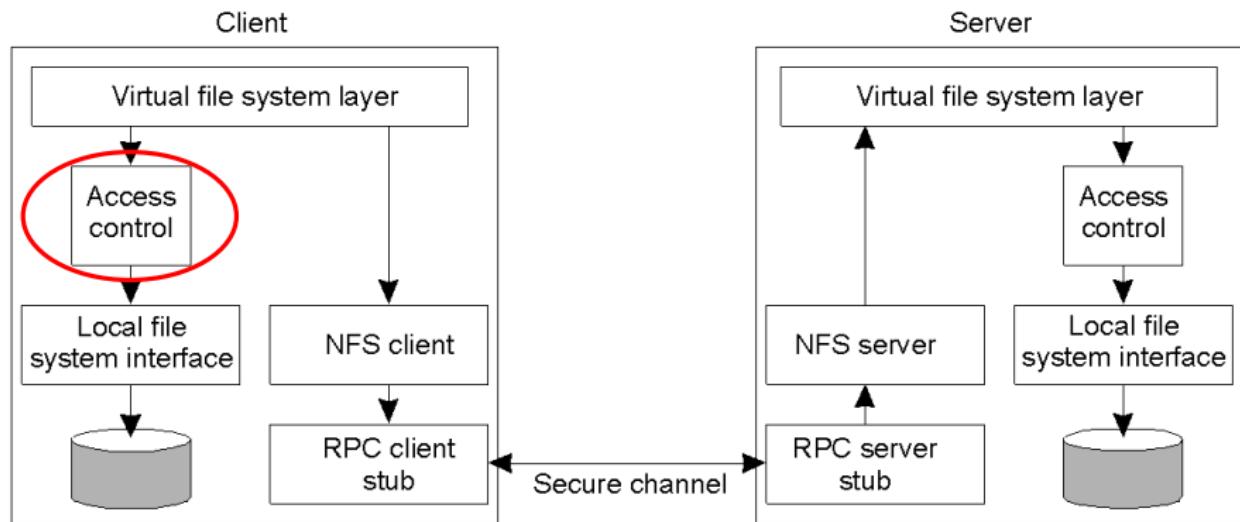


Abb. aus Tanenbaum/Steen

Sicherheit

Prinzipielle Architektur

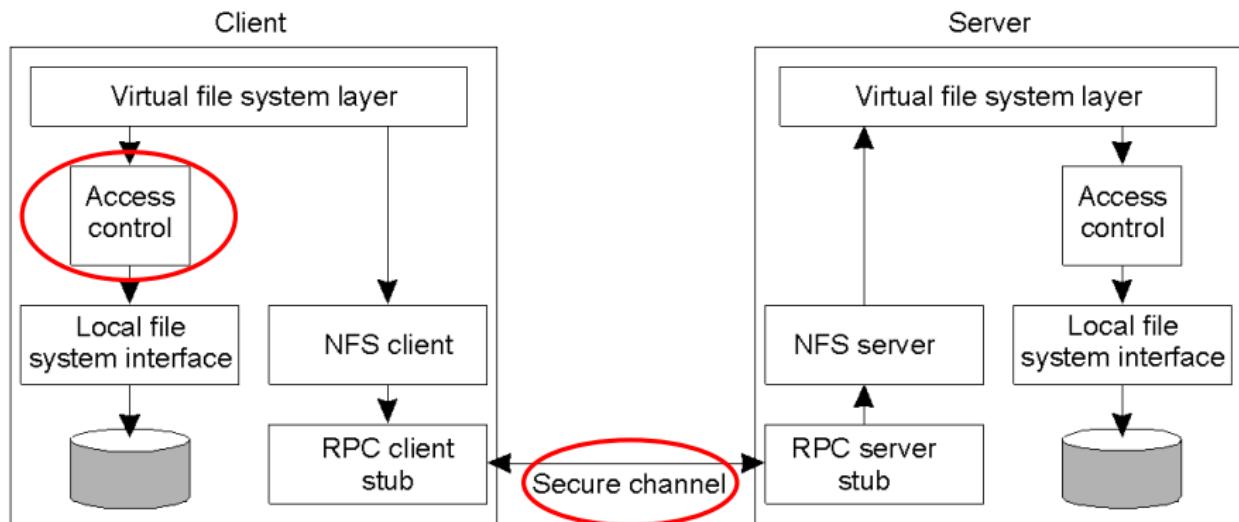


Abb. aus Tanenbaum/Steen

Sicherheit

Prinzipielle Architektur

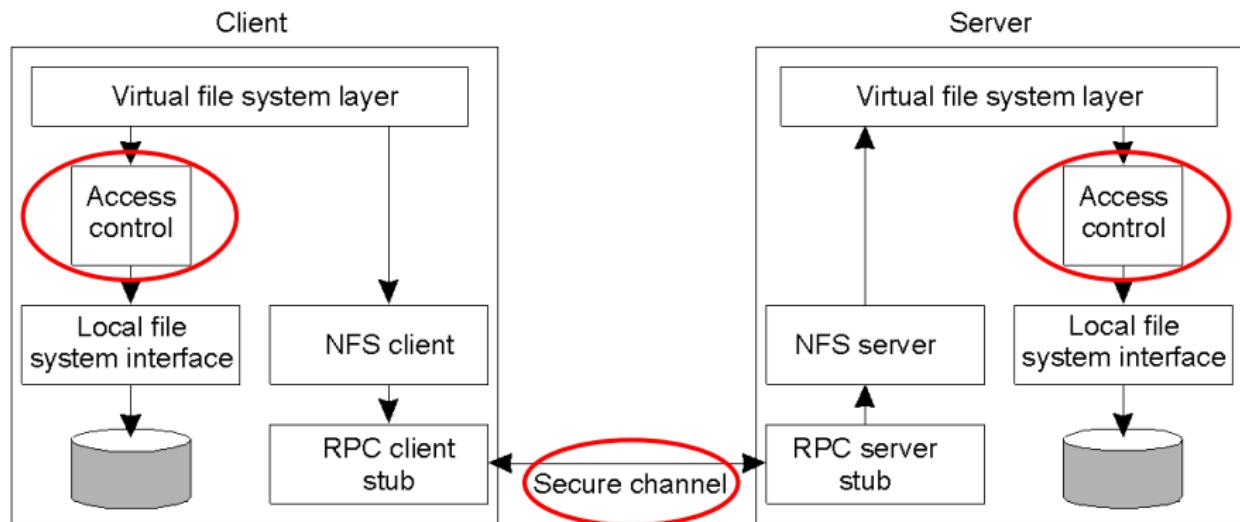


Abb. aus Tanenbaum/Steen

Sichere RPCs

in Version 3:

- Nur Authentifizierung
 - ▶ „System“
 - ★ basierend auf UNIX (uid, gid)
 - ★ übertragen im Klartext ohne Signatur (Server traut Client)
 - ▶ Diffie-Hellman
 - ★ selten genutzt
 - ★ wegen zu kurzer Schlüssel heute als definitiv unsicher eingestuft
 - ▶ Kerberos

in Version 4:

- Keine eigenen, fest eingebauten Mechanismen
- Unterstützung von RPCSEC_GSS
 - ▶ Sicherheitsumgebung für verschiedenste einklinkbare Mechanismen
 - ▶ Neben Authentifizierung auch Integrität und Vertraulichkeit

Sichere RPCs (2)

Architektur sicherer RPCs in Version 4:

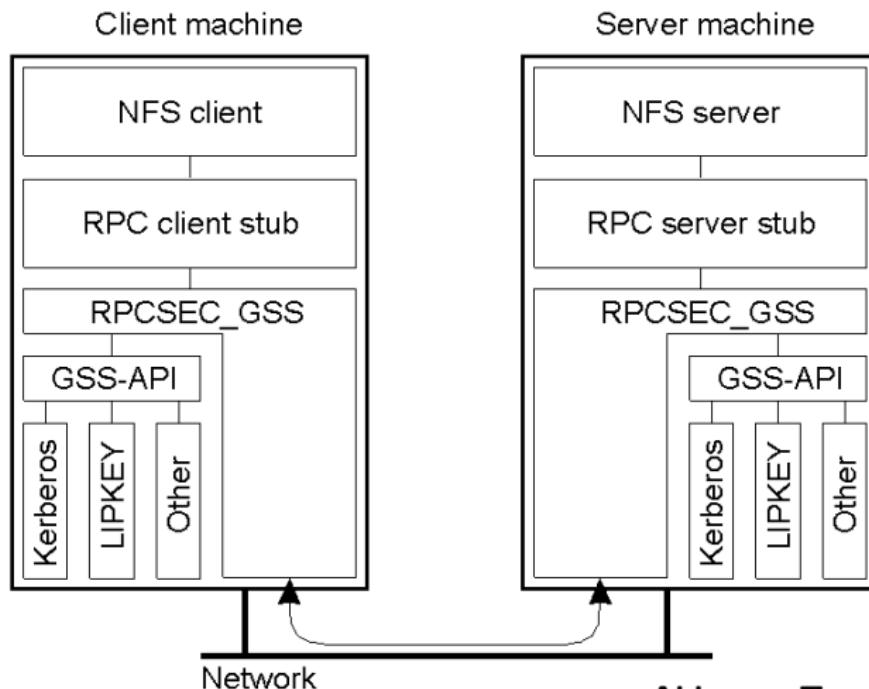


Abb. aus Tanenbaum/Steen

Zugriffskontrolle

in Version 3:

- UNIX-Rechteüberprüfung (uid, gid) auf Server-Seite

in Version 4:

- ACL-basiert
- Subjekte stark differenziert

Zugriffskontrolle (Operationen)

Operation	Beschreibung
Read_data	Permission to read the data contained in a file
Write_data	Permission to modify a file's data
Append_data	Permission to append data to a file
Execute	Permission to execute a file
List_directory	Permission to list the contents of a directory
Add_file	Permission to add a new file to a directory
Add_subdirectory	Permission to create a subdirectory to a directory
Delete	Permission to delete a file
Delete_child	Permission to delete a file or directory within a directory
Read_acl	Permission to read the ACL
Write_acl	Permission to write the ACL
Read_attributes	The ability to read the other basic attributes of a file
Write_attributes	Permission to change the other basic attributes of a file
Read_named_attrs	Permission to read the named attributes of a file
Write_named_attrs	Permission to write the named attributes of a file
Write_owner	Permission to change the owner
Synchronize	Permission to access a file locally at the server with synchronous reads and writes

Zugriffskontrolle (Subjekte)

Benutzertyp	Beschreibung
Owner	The owner of a file
Group	The group of users associated with a file
Everyone	Any user of a process
Interactive	Any process accessing the file from an interactive terminal
Network	Any process accessing the file via the network
Dialup	Any process accessing the file through a dialup connection to the server
Batch	Any process accessing the file as part of a batch job
Anonymous	Anyone accessing the file without authentication
Authenticated	Any authenticated user of a process
Service	Any system-defined service process

Nach Tanenbaum/Steen

AFS und Coda

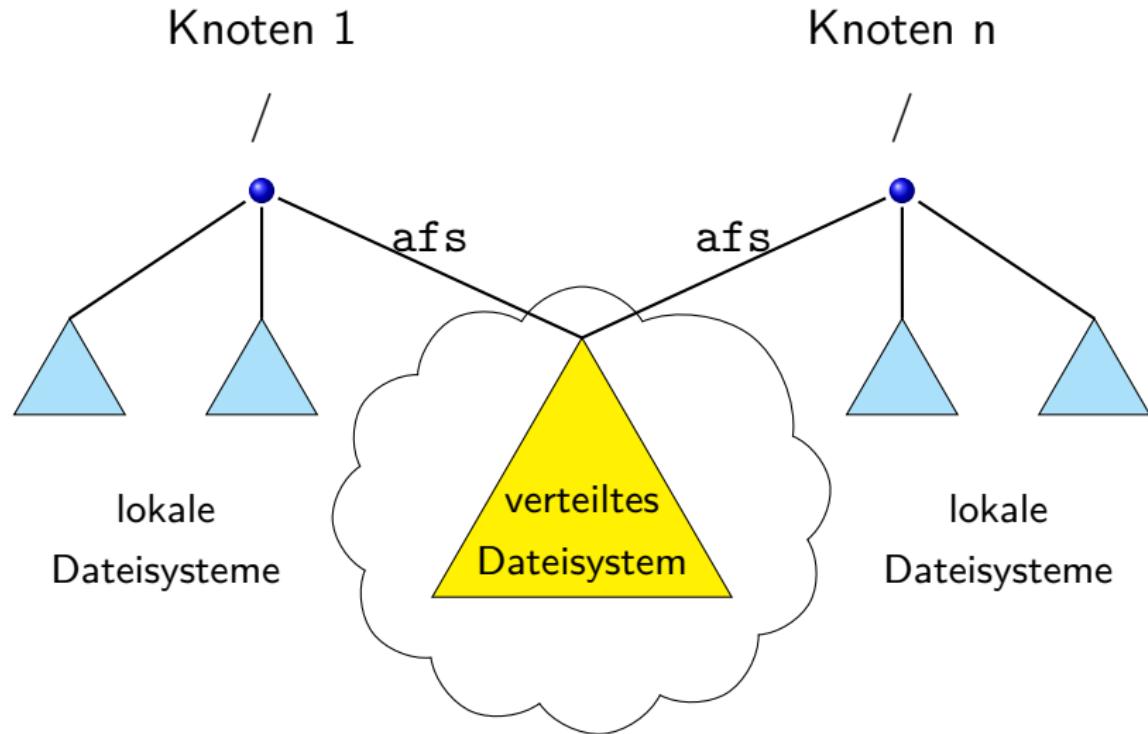
Andrew File System (AFS)

- CMU zusammen mit IBM, 1983-1989, danach Transarc
- Dateisystem für den Campus mit > 5.000 aktiven Studenten
- Ziele
 - ▶ Ortstransparenter, globaler, gemeinsam genutzter Dateinamensraum, erreichbar über den lokalen Namen /afs
 - ▶ Hohe Performance
 - ▶ Hohe Verfügbarkeit
 - ★ Replikation
 - ▶ Hohe Sicherheit
 - ★ Gesicherte Authentifikation
 - ★ Verschlüsselte Übertragung
 - ★ ACLs zur Zugriffskontrolle
 - ▶ Automatische Migration von Home-Verzeichnissen von Benutzern

Coda

- Weiterentwicklung von AFS-2

Dateinamensraum



Gesamtarchitektur

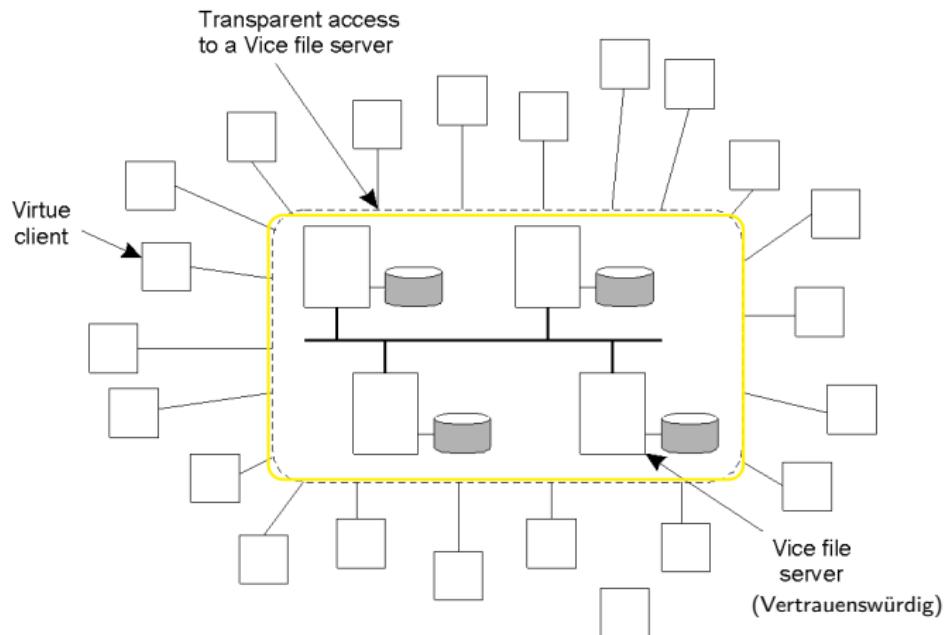
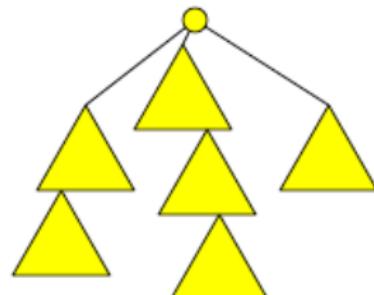


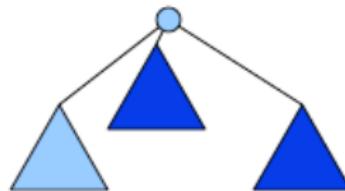
Abb. aus Tanenbaum/Steen

Volumes

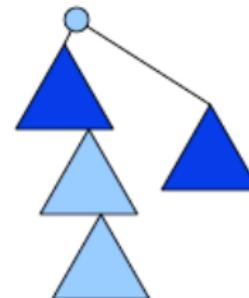
Mountbare
Volumes



Struktur
des globalen Dateiraums
in Vice



Knoten 1



Knoten n

reale Struktur
in den Knoten
(Teilgraph)



Architektur eines Virtue-Clients

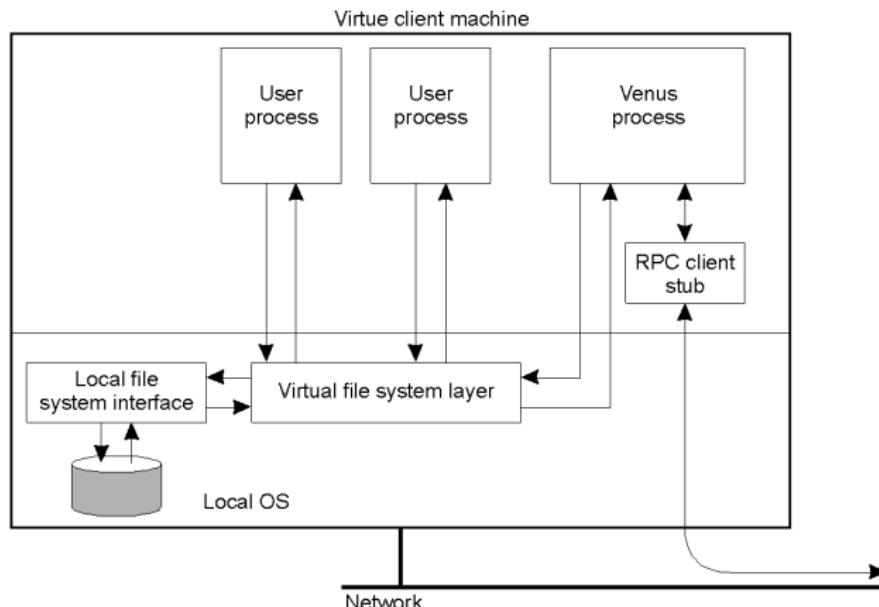


Abb. aus Tanenbaum/Steen

Architektur eines Virtue-Clients

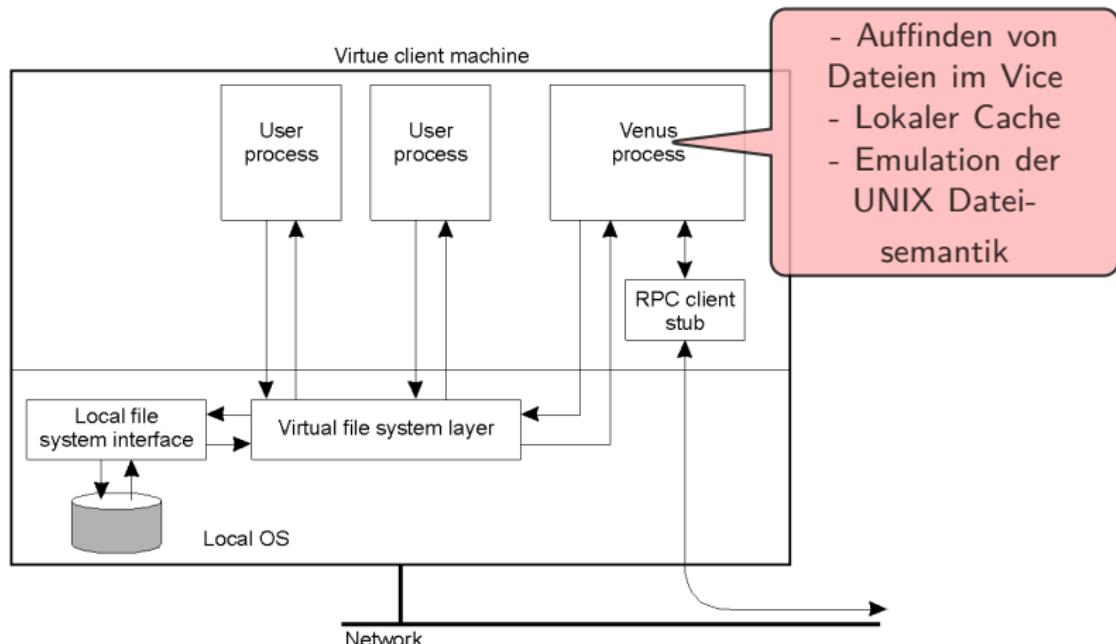


Abb. aus Tanenbaum/Steen

Architektur eines Virtue-Clients

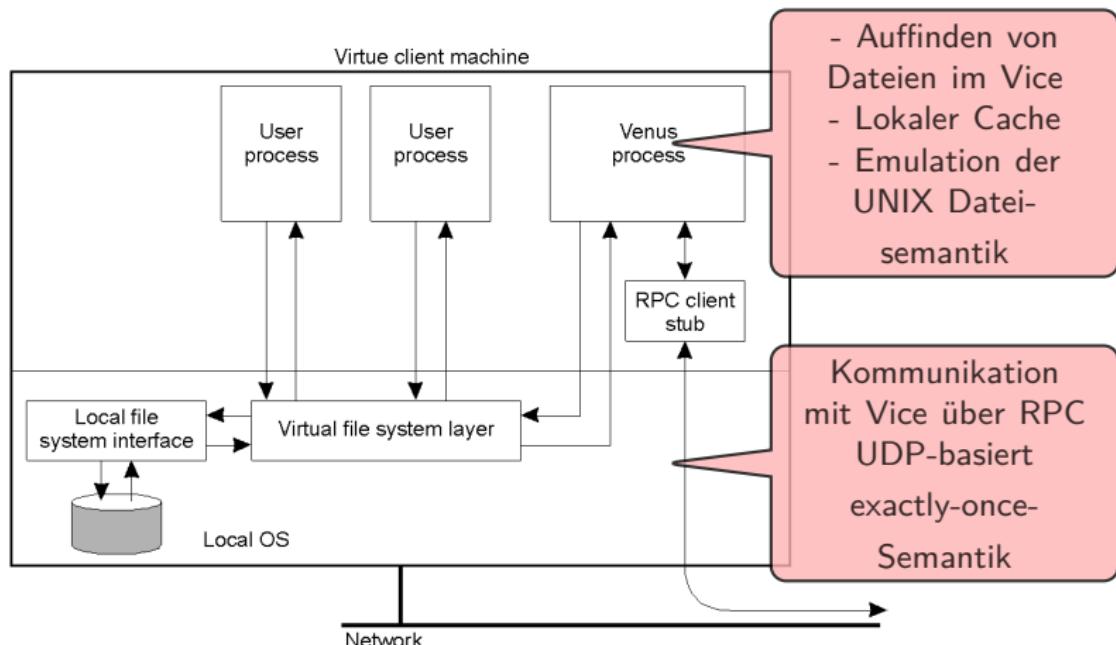


Abb. aus Tanenbaum/Steen

Eigenschaften

- gemeinsam genutzte Dateien mit Sitzungssemantik
- Lokales Caching von ganzen Dateien bis AFS-2, von großen Dateiblöcken (64 kB) ab AFS-3
- Cache-Kohärenz
 - ▶ Überprüfen der Gültigkeit des Caches nicht bei jedem open() notwendig
 - ▶ Callback-Verfahren, d.h. explizites Invalidieren durch den Server, bevor ein anderer Client Schreibrecht bekommt.

Sicherheit

- Organisation
 - ▶ Vice-Server sind vertrauenswürdig
 - ▶ keine Benutzer-Anwendungen auf Servern
 - ▶ Einführung administrativer Zellen zur Erhöhung der Skalierbarkeit
- Subjekte
 - ▶ Benutzer
 - ▶ Gruppen
- Authentifizierung
 - ▶ spezielle Authentication Server, Kerberos (ab AFS-3)
 - ▶ sicherer RPC
- Zugriffskontrolle
 - ▶ ACLs für Verzeichnisse definiert, gelten für alle Dateien des Verzeichnisses

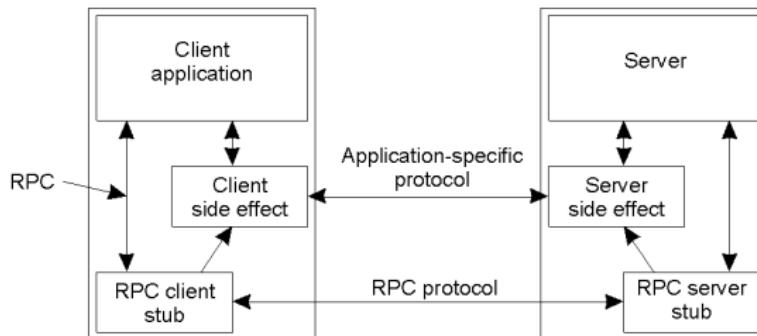
Coda

- Weiterentwicklung von AFS-2 ab 1987
- Ziele:
 - ▶ Hohe Verfügbarkeit der Daten
 - ▶ Client soll weiterarbeiten können, auch wenn Server vorübergehend nicht erreichbar ist (Netzpartitionierung)
 - ▶ Einbeziehung von mobilen Rechnern (gewollte Netzpartitionierung)

Kommunikation

RPC2-System

- Nutzung von internem Multithreading für Venus und Vice
- Unterstützung von sog. Nebeneffekten



- MultiRPCs (transparenter Aufruf mehrerer Server) genutzt für Cache-Invalidierung
(Parallele Einzelaufrufe oder Nutzung von IP-Multicast)

File IDs

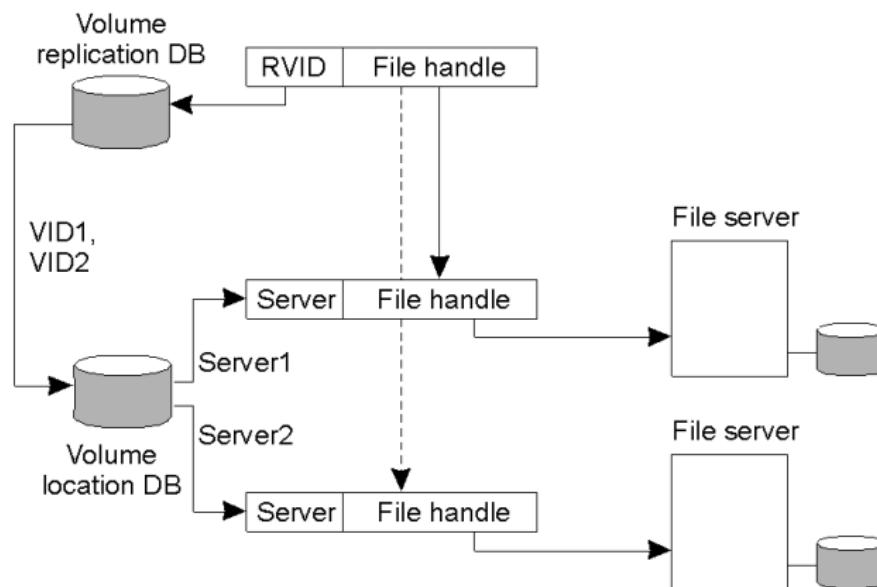


Abb. aus Tanenbaum/Steen

File IDs

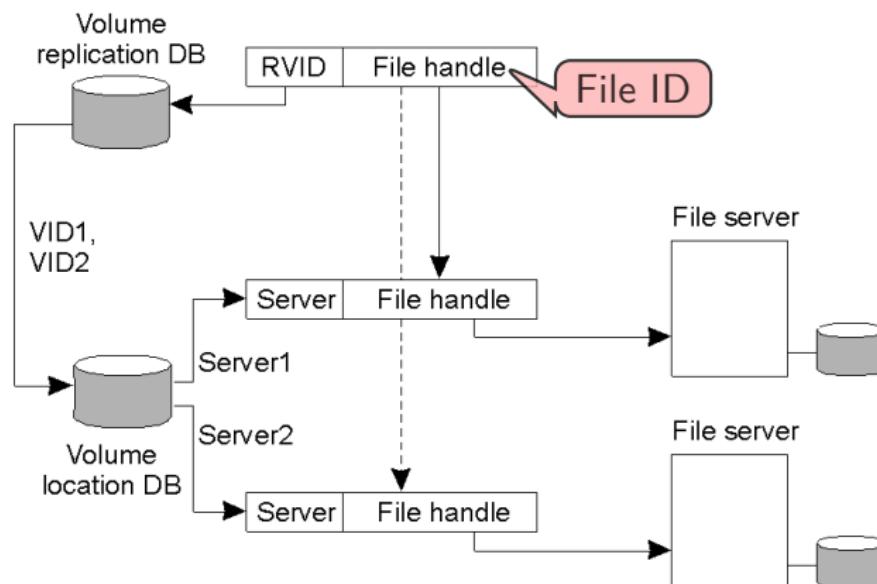


Abb. aus Tanenbaum/Steen

File IDs

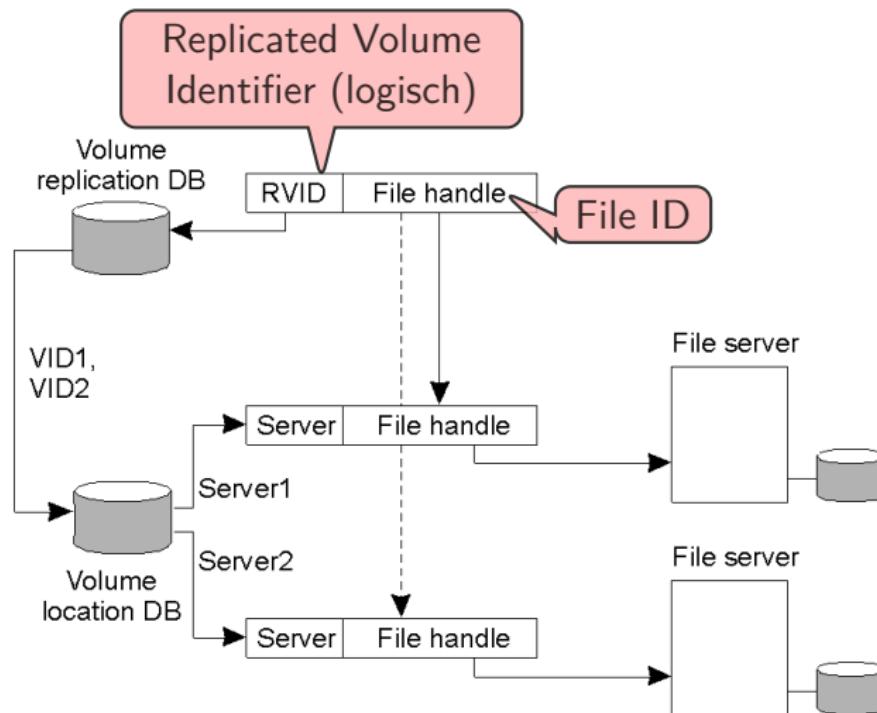


Abb. aus Tanenbaum/Steen

File IDs

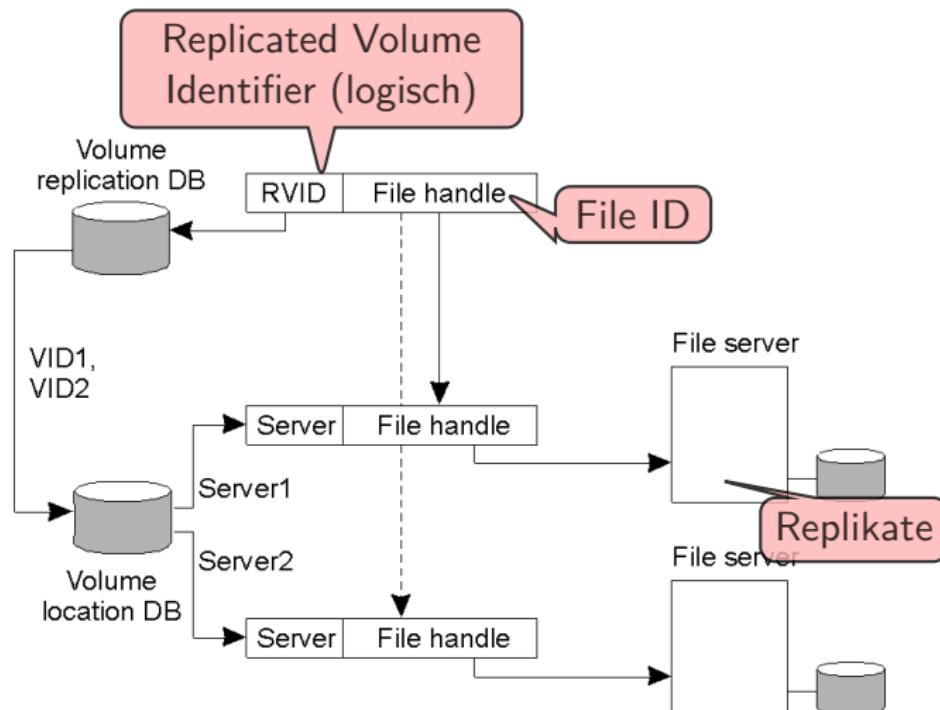


Abb. aus Tanenbaum/Steen

Eigenschaften im Normalbetrieb

- Sitzungssemantik
- ein Schreiber oder mehrere Leser
- „Transaktions“-Verhalten

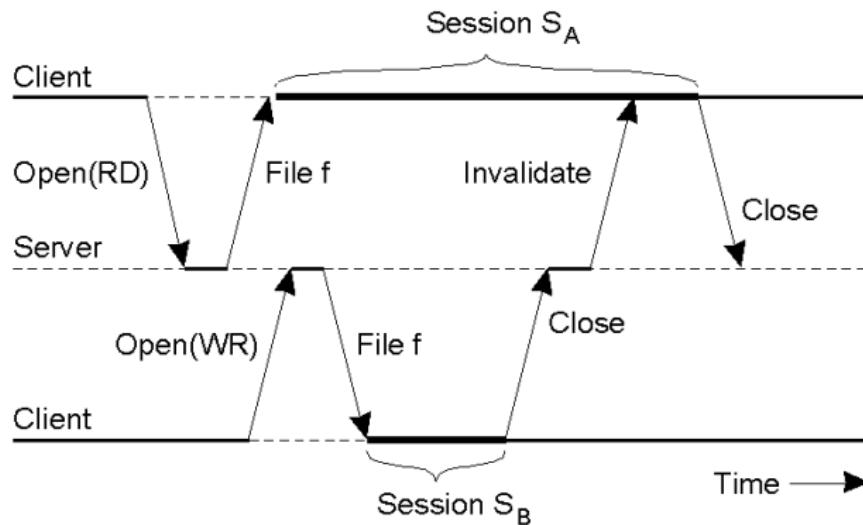


Abb. aus Tanenbaum/Steen

Eigenschaften im Normalbetrieb

- Sitzungssemantik
- ein Schreiber oder mehrere Leser
- „Transaktions“-Verhalten

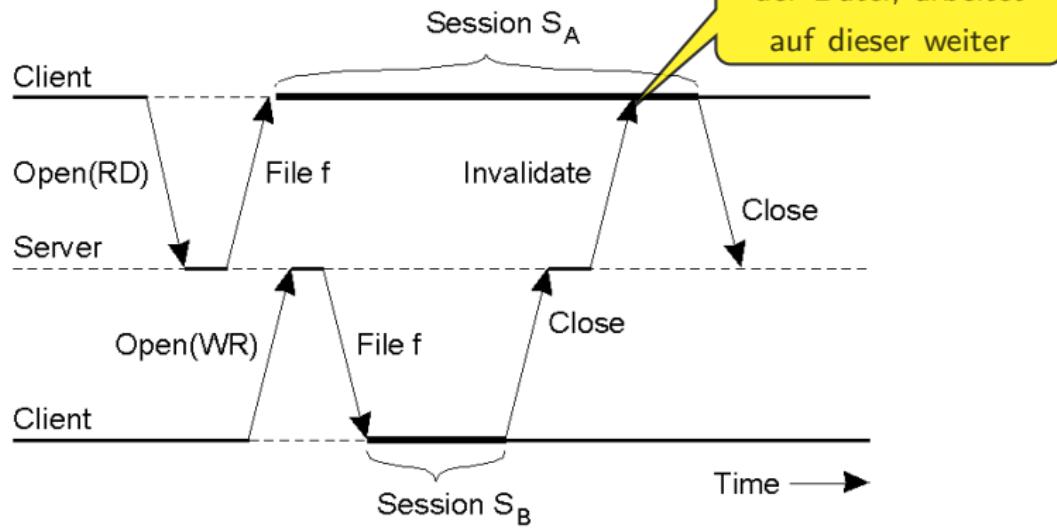


Abb. aus Tanenbaum/Steen

Caching

- Bei `open()` wird Datei in Client-Cache geladen
- Server gibt Callback-Versprechen
- Zur Invalidierung schickt Server Callback Break

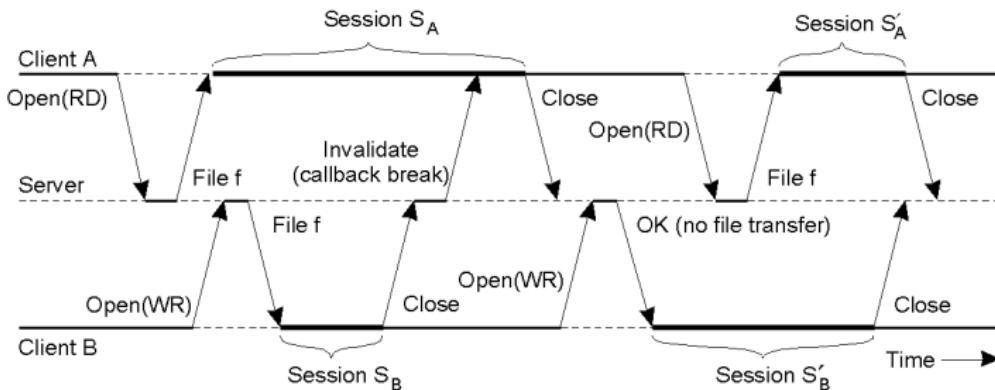


Abb. aus Tanenbaum/Steen

Caching

- Bei `open()` wird Datei in Client-Cache geladen
- Server gibt Callback-Versprechen
- Zur Invalidierung schickt Server Callback Break

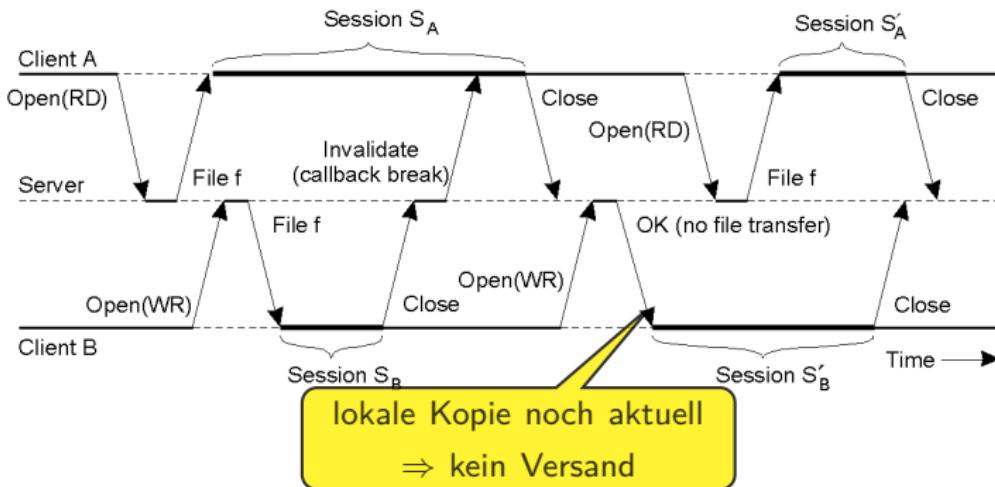


Abb. aus Tanenbaum/Steen

Caching

- Bei `open()` wird Datei in Client-Cache geladen
- Server gibt Callback-Versprechen
- Zur Invalidierung schickt Server Callback Break

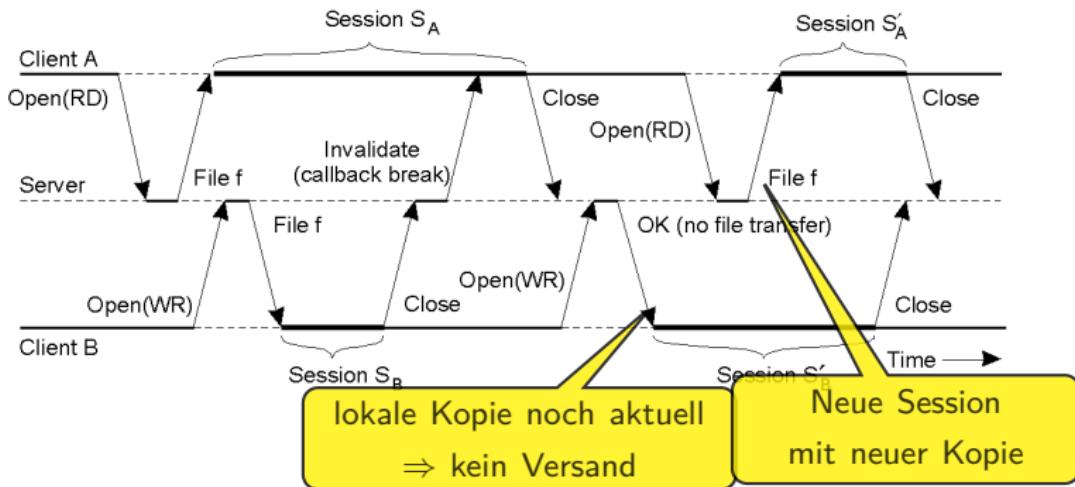


Abb. aus Tanenbaum/Steen

Server-Replikation und Netz-Partitionierung

- Volume ist Einheit der Replikation
- Volume Storage Group (VSG)
 - ▶ Menge der Server mit Kopie eines Volumes
- Accessible Volume Storage Group (AVSG)
 - ▶ Teilmenge von VSG, die Client kontaktieren kann
- Lesen von einem Replikat, Schreiben an alle mittels MultiRPC
- Optimistische Strategie für Dateireplikation
 - ▶ Bei Partitionierung dürfen mehrere Schreiber existieren und auf ihre jeweilige AVSG zurückschreiben
- Führen von Versionsvektoren entsprechend Vektorzeitstempel (vgl. Kap. 7) und Überprüfung bei Aktualisierung
- Spätere Zusammenführung verschiedener Versionen erfordert z.T. manuelle Hilfe

Verbindungsloser Betrieb

- Verbindungslos: AVSG = \emptyset , dann Nutzung der lokalen Kopie
- Konflikterkennung bei Übertragung auf den Server (s.o.)
- Beobachtung
 - ▶ Konflikte selten, da selten eine Datei von mehreren Prozessen gleichzeitig modifiziert wird
- Problem
 - ▶ Relevante Dateien im lokalen Cache haben, wenn Verbindungslosigkeit eintritt
- Ansatz: Hoarding (Horten von Dateien)
 - ▶ Heuristisches Verfahren
 - ▶ Explizite Angabe von Dateien und Verzeichnissen durch Benutzer
 - ▶ Priorisierung durch Abgleich mit aktueller Zugriffsinformation
 - ▶ Cache-Ausgleich (Hoard-Walk) alle 10 min
 - ▶ Gute Erfahrungen, aber es kommt natürlich vor, dass gelegentlich relevante Dateien fehlen

Zusammenfassung

File-associated data	Read-lock	Write-Lock
Issue	NFS	Coda
Design goals	Access transparency	High availability
Access model	Remote	Up/Download
Communication	RPC	RPC
Client process	Thin/Fat	Fat
Server groups	No	Yes
Mount granularity	Directory	File system
Name space	Per client	Global
File ID scope	File server	Global
Sharing sem.	Session	Transactional
Cache consist.	write-back	write-back
Replication	Minimal	ROWA
Fault tolerance	Reliable comm.	Replication and caching
Recovery	Client-based	Reintegration
Secure channels	Existing mechanisms	Needham-Schroeder
Access control	Many operations	Directory operations

Nach Tanenbaum/Steen

Nutzung von Netzwerktechnologie und verteilten Systemen innerhalb von Speichersystemen

Motivation

- Kostenreduktion
 - ▶ Geringere bereitgestellte Speicherkapazitäten
 - ▶ Zentrale Administration
- Höhere Flexibilität in der Zuordnung
 - ▶ Schnelle Anpassbarkeit an neue Bedürfnisse
- Skalierbarkeit
 - ▶ Kleine bis sehr große Speicherkapazitäten
- Möglichkeit für Desaster Recovery
 - ▶ Datenspiegelung an entfernte Stellen

Architekturansätze

Wesentliche heutige Architekturansätze

- Direct Attached Storage (DAS) (klassischer lokaler Speicher)
- Storage Area Networks (SAN)
- Network-Attached Storage (NAS)
- Content Adressed Storage (CAS)

Im Folgenden vorgestellt

Speichersysteme als geschichtete Systeme

Grundlegende Aufteilung der Funktionalität von Informationsspeicherung

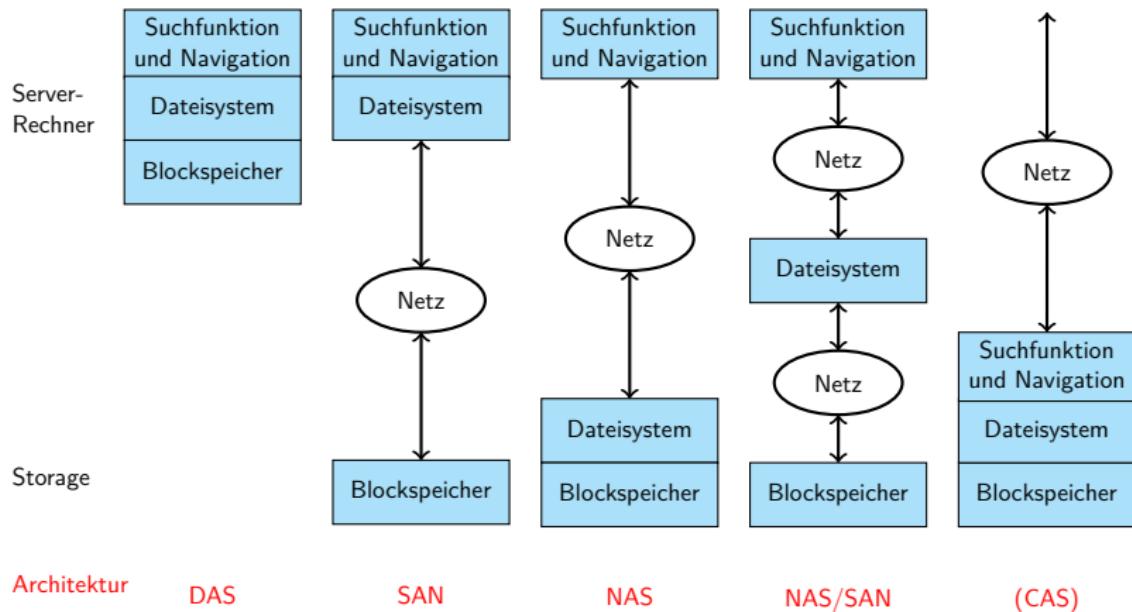


Pfadnamen, Query, Index, Metadaten, ...

Abbildung auf logische Blockmenge,
z.B. NFS, AFS, Microsoft SMB/CIFS

Abbildung auf phys. Speichergerät

Überblick zur Integration von Netzen



Architektur

DAS

SAN

NAS

NAS/SAN

(CAS)

DAS: Direct Attached Storage

- Traditionelle, lokal angeschlossene Speichergeräte
- Lokales Betriebssystem enthält Dateisystem und Gerätetreiber
- Typische Geräteschnittstellen
 - ▶ IDE/ATA, SCSI, Serial ATA (SATA), ...
- Beschränkungen
 - ▶ Anzahl vorhandener Kanäle
 - ▶ Anzahl anschließbarer Devices je Kanal
 - ▶ maximale Entfernung ca. 1-25 m
 - ⇒ kein Disaster-Recovery möglich
 - ▶ Performance
 - ⇒ schlechte Skalierbarkeit

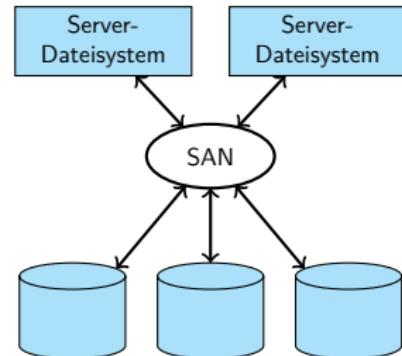
SAN: Storage Area Network

Funktionsweise

- SAN offeriert Blockspeicher (z.B. Festplatten als Logical Devices)
- Server-Betriebssysteme enthalten ein oder mehrere Dateisysteme
- Blockspeicher wird über das SAN von Servern zugegriffen

Vorteile

- Sehr leichte Erweiterbarkeit
- Sehr flexible Zuordnbarkeit
- Hohe Skalierbarkeit
- Basis für Replikation, auch für Disaster Recovery
- Bootbare Netzwerkpartitionen
- Besonders geeignet für Anwendungen, die mit Volumes (ohne Dateisystem) arbeiten, z.B. DBMS



SAN: Storage Area Network(2)

Verbreitete Netzwerke

- Fibre Channel (FC)

- ▶ Dediziertes Netzwerk mit 1/2/4/8/16 GBit/s über Glasfaser
- ▶ geschichtetes Protokoll FC-0 bis FC-4
- ▶ verschiedene Übertragungsprotokolle in FC-4 einbettbar, typisch: serielles SCSI-3
- ▶ Skalierbarkeit über Switches
- ▶ Ausdehnung bis 10-100 km

- Internet SCSI (iSCSI)

- ▶ neuerer Standard (RFC 3720, 2004)
- ▶ Nutzung von preiswerter Internet-Technologie zur Blockübertragung
- ▶ TCP/IP als unterlagertes Protokoll
- ▶ Ausdehnung beliebig
- ▶ Security- und QoS-Standards nutzbar
- ▶ hohes Wachstum mit Gbit/s-Ethernet erwartet

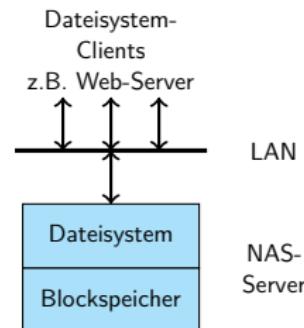
NAS: Network Attached Storage

Funktionsweise

- NAS offeriert Netzwerkdateisysteme (z.B. NFS, SMB/CIFS (Server Message Block bzw. Nachfolger Common Internet File System))
- Clients können Dateisysteme nutzen

Vorteile

- Speicher-Konsolidierung
- Erweiterbarkeit
- Skalierbarkeit
- Managebarkeit
- Besonders geeignet für Anwendungen, die auf Dateizugriffen basieren, z.B. Web-Anwendungen, Home-Verzeichnisse



NAS: Network Attached Storage (2)

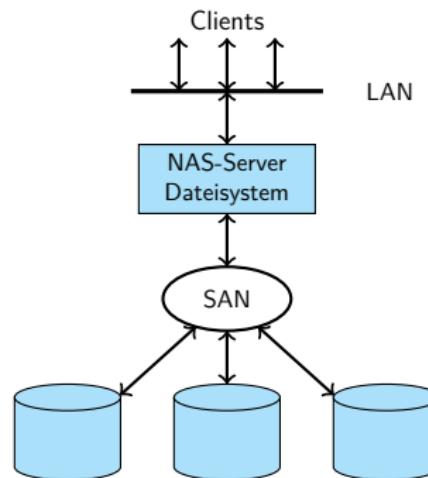
Beispiele:

- Home Server
 - ▶ Synology
 - ▶ Zyxel
- IBM Storvize V7000

NAS / SAN Verbund

Funktionsweise

- NAS und SAN können kombiniert eingesetzt werden
- Vorteile lassen sich so kombinieren



CAS: Content Adressable Storage

Grundlage des Ansatzes: Unveränderliche Information

- daher auch Fixed Content Storage (FCS) genannt
- Ziel: Archiv-Speicher

Teilproblem des Information Lifecycle Management (ILM)

- Massendaten (Hunderte von Terabytes bis Petabytes)
- Langlebigkeit
- Integrität
- Unveränderbarkeit von Dokumenten z.T. gesetzlich gefordert

Anwendung im Bereich Content Management

- Digitale Medien (Ton-, Bild-Dokumente)
- email-Archivierung
- Gesundheitswesen (Röntgenbilder etc.)
- ...

CAS: Content Adressable Storage (2)

Funktionsweise

- Content Identifier als Referenz
- Bestimmung eines Content-Identifiers
 - ▶ ausschließlich aus Inhalt (analog Hash-Wert) ⇒ ortsunabhängig
 - ▶ oder aus Speicherort (invertiert)
- System ermittelt aus Content Identifier Location für Zugriff

Beispiele

- Erstes System: EMC Centera 2002
- iTernity iCAS (Software-Lösung)
- Versch. Open Source Lösungen, z.B. Keep Content Adressable Storage

Zusammenfassung

- Verteilte Dateisysteme ermöglichen den nebenläufigen Zugriff unterschiedlicher Nutzer auf Dateien unabhängig von ihrem Speicherort.
- Der Zugriff auf Dateien kann je nach Konsistenz-Semantik zu unterschiedlichen Ergebnissen führen.
- Durch Aufteilung der Funktionalität zur Informationsspeicherung auf unterschiedliche Systeme im Netzwerk lassen sich u.a. Skalierbarkeit, Fehlertoleranz und Erweiterbarkeit verbessern.