Modul: ARBK
Architektur von Rechnersystemen und Betriebssystemkonzepte
WS 2017/2018

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Netzwerk-dienste

Dr.-Ing. Arno Bücken
FB5 FH Aachen

Netzwerke

Kommunikation in Netzen

Dateisysteme im Netz

Arbeitsmodelle im Netz

Dienste im Netz

Netzwerk-Nutzen

- **electronic mail**

  Kommunikation: Terminabsprachen, Projektkoordination, Mitteilungen, ...

- **file sharing**

  keine multiplen Kopien: Dateikonsistenz, Speichererparnis

- **device sharing**

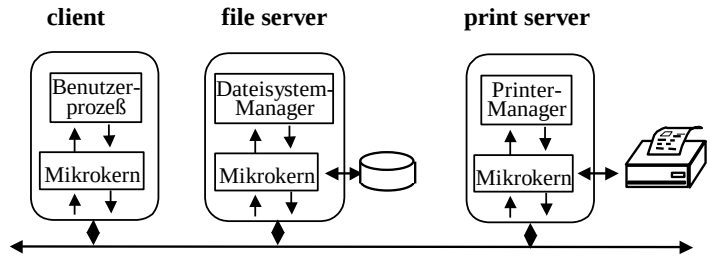
  bessere Druckerauslastung, lohnende Anschaffung von Spezialhardware
  (Farblaserdrucker, high-speed-scanner,...)

- **processor sharing**

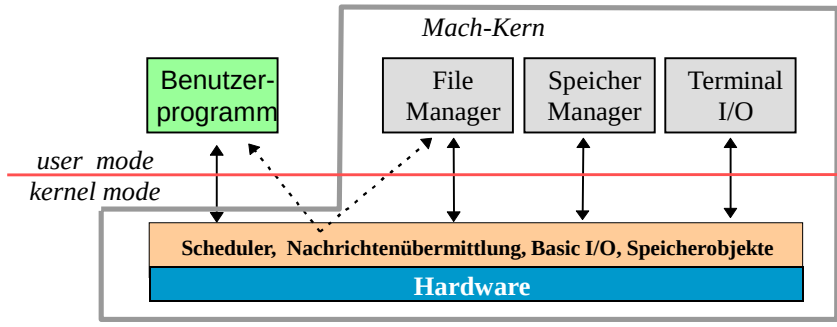
  Zeitersparnis durch bessere Prozessorauslastung bei Lastverteilung und /oder Kostenersparnis durch geringere Investitionen

Verteilte Betriebssysteme

- **Verteiltes System**: Aufteilung von Funktionen in einem Rechnernetz, wobei BS auf *jedem* Rechner ex.
- **Verteiltes Betriebssystem**:
  Jede BS-Funktion ex. nur einmal im Netz



MACH- Betriebssystemkern



Mikrokern

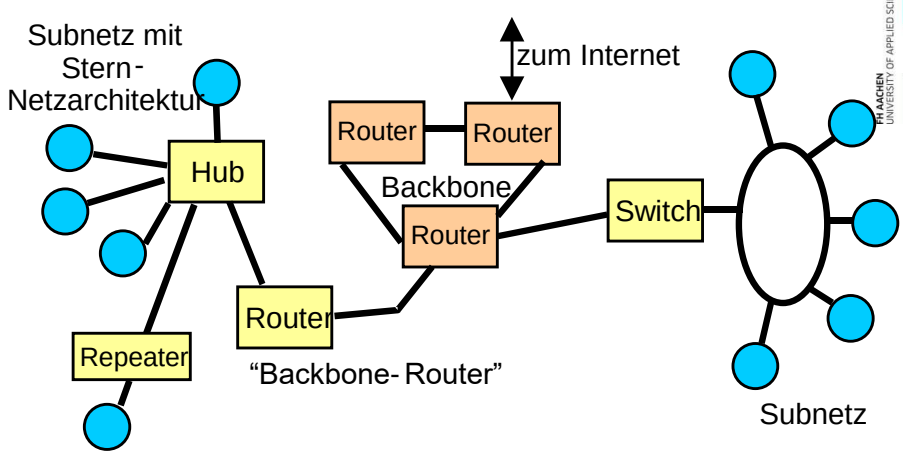
- **Vorteile**: minimaler Kern, alle Funktionen modularisiert austauschbar
- **Nachteile**: Kommunikationsdauer zwischen Managern

Verteilte Betriebssysteme

- **Vorteile**
  - **Flexibilität** inkrementelle Erweiterbarkeit um neue Dienste
  - **Transparenz** durch ortsunabhängige Dienste
  - **Leistungssteigerung** bei Lastverteilung
  - **Fehlertoleranz** bei multiplen, gleichen Diensten
- **Nachteile**
  - **Leistungseinbuße** durch Kommunikationsverzögerung
  - **Keine Fehlertoleranz** wenn Funktion nur einmal vorhanden
  - **Synchronisation** nötig bei Aktualisierung verteilter Daten
- **Fazit**

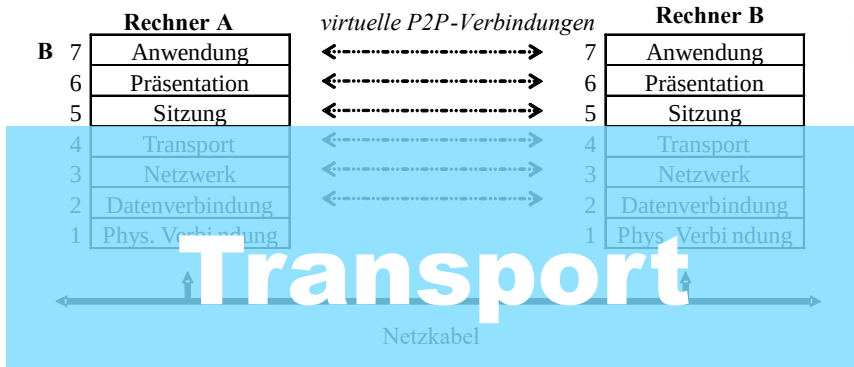
  Alle BS sind Mischsysteme aus netzbasierten & lokalen BS-Funktionen; es ex. kein „reines" System

Netzwerke: Grundbegriffe



Netzwerkschichten OSI-ISO

- Schichten virtueller Maschinen
- End-to-End Verbindung: portable Software



Vorteil Systematische, portable Einteilung

Nachteil zu starr und damit zu langsam

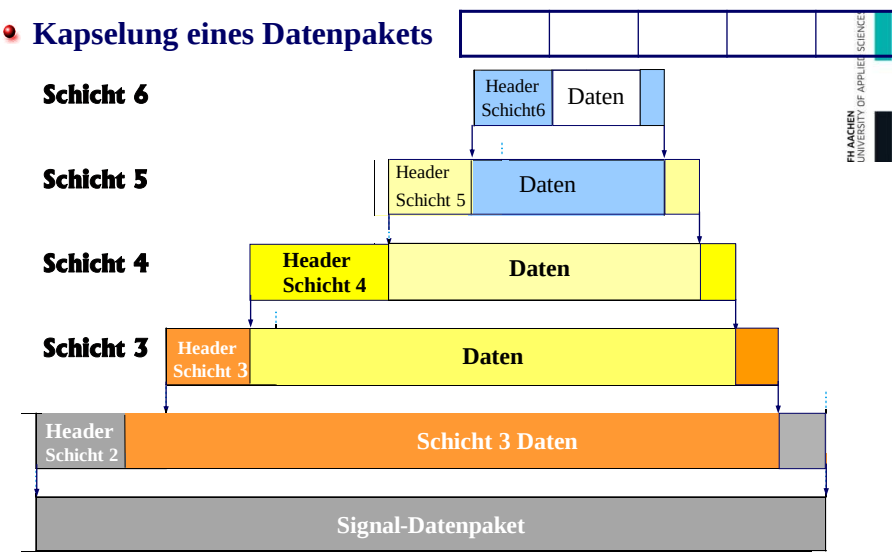
Lösung Zusammenfassung von Schichten

Netzwerkschichten: OSI-ISO

- **Layer 7 : Anwendungsschicht**
  High-level Programme: FTP, Grafik, electronic mail, ...
- **Layer 6 : Präsentationsebene**
  Datenformatierung, Kodierung, Gruppierung (Records, Verschlüsselung,)
- **Layer 5 : Sitzungsebene**
  open/close-Semantik: Sender, Empfänger, high-level-Fehlerbehandlung, logon-passwords, Daten/Kontrollunterscheidung,...
- **Layer 4 : Transportschicht**
  Umwandlung in Datenpakete, Reihenfolge der Pakete, usw. Bei **TCP** (*Transmission Control Protocol*): Fehlertoleranzgrad TP0-4 festlegen
- **Layer 3 : Netzwerkschicht *Router , Bridges***
  Fragen der Netztopologie: Übertragungsweg, Umleitung (routing), Netzstatus, Grenzen, Auslastung, usw. Typisch: *Internet Protocol* **IP**
- **Layer 2 : Datenverbindung *Layer2-Switch***
  Datenpakete → Unterteilung in log. Signalframes, Wiederholung bei NO-ACK. Aber: Frame-Reihenfolge ist unkontrolliert. Z.B.: **Ethernet**
- **Layer 1 : physikalische Signale** Bits→Impulse, Freq. z.B.100BaseT
  Repeater, Hub

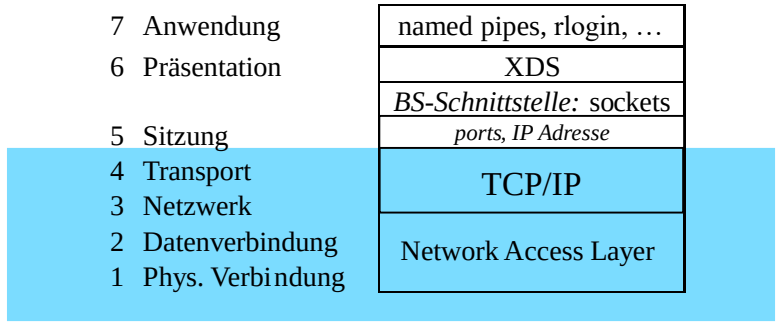
Netzwerkschichten: Datenpakete

- **Kapselung eines Datenpakets**



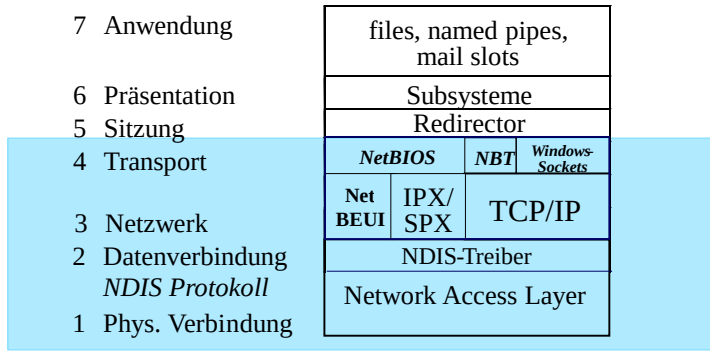
Kommunikationsschichten: Unix

- Stream-System für Protokollschichten
- Schicht = Treiber, leicht austauschbar



Kommunikationsschichten: Windows NT

- Kompatibilität zu bestehenden Protokollen
  SMB (*server message block*)
  NetBIOS (*network basic input output system*)



Virtual Private Networks VPN

Probleme

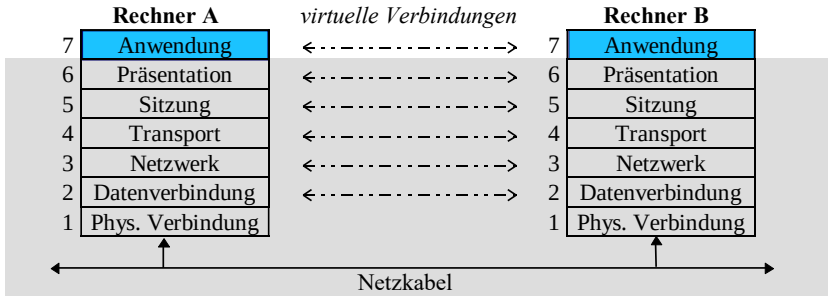
- **Geheimhaltung** von Daten (Sprache, Dokumente, email)
- Unterschiedl. **Grösse** der Datenpakete in gekoppelten Netzen
- Unterschiedl. **Art** von Transportprotokollen

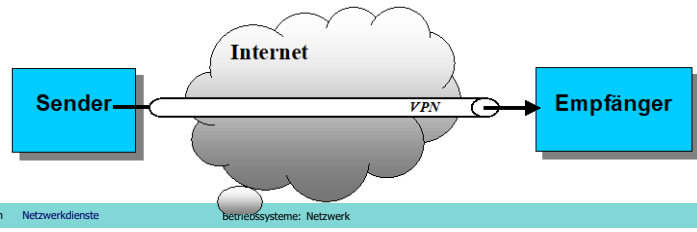
Lösung

- **Verschlüsselung** der Kommunikation der Anwenderebene

Virtual Private Networks VPN

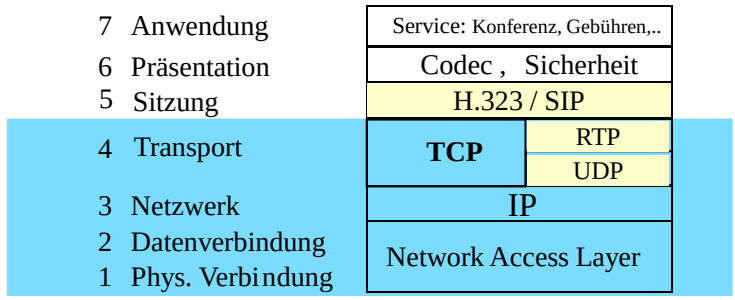
- End-to-End-Protokoll: VPN durch Verschlüsselung





Technik VoIP, Video

- **Anforderung**: Viele Sprach/Bildsamples
- **Lösung**: Neues Paketmanagement im Schichtenmodell



Overhead 40Byte/Paket: Header IPv4:20 Byte, UDP:12 Byte, RTP: 8 Byte

➢ Zusammenfassung mehrerer *samples* zu einem Paket!

VoIP Probleme

Anpassung der Parameter nötig gegen:

- **Echos** :Paketwiederholung auf mehreren Pfaden
- **Abhacken**, Aussetzer der Sprache : verlorene Pakete
- **Kompressionsverzerrung** : 1-Weg statt 2-Weg Kommunikation (Wechselkanal Sprecher-Zuhörer)

Netzwerke

Kommunikation in Netzen

Dateisysteme im Netz

Arbeitsmodelle im Netz

Dienste im Netz

IP-Adresse

Namensgebung im Internet

- **Eindeutige** IP-Adresse: z.B. „141.2.15.25“
  IPv4: 32 Bits, notiert in 4 Dezimalzahlen je 0..254 (1Byte),
  zu wenig Adressen (nur 65535) => **IPv6: 128Bit**
- **Name**: data.buecken.name *server.LocalNet.domain.country*
  Zuordnung IP-Nummer←→Name wird auf speziellen Rechner gehalten (*domain name service **DNS***)

Vergabe und Zuordnung der IP-Adresse durch **zentrale** Instanzen

Beispiele *CIDR = Classless Inter-Domain Routing*

127.0.0.0/8	lokaler Computer loopback
10.0.0.0/24	private Netzwerke (RFC 1918)
172.16.0.0/16 – 172.31.0.0/16	
192.168.0.0/16	

Automat. Konfiguration: Dynamic Host Configuration Protocol **DHCP**

169.254.0.0/16	privates, link-local Netz (APIPA)

IP-Adresse

Internetnamen: Subnetze

Problem: hoher zentraler Verwaltungsaufwand bei zu vielen Netzen

Lösung: Unterteilung der Rechneradresse in (Subnetz,Rechner), dezentrale Verwaltung

dynamische Aufteilung durch Bitmaske (Subnetzmaske)

Adressierung (Routingentscheidung) der Subnetze durch die Maske:

? (Adresse **AND** Maske) =? Subnetznummer

- JA : Zielrechner ist lokal im Subnetz
- NEIN : Routing-Rechner ansprechen

Beispiel	129.206.218.160 /24	*CIDR-Notation*
Rechner 160	129.206.218.160	1000.0001.1100.1110.1101.1010.1010.0000
Maske	255.255.255.0	1111.1111.1111.1111.1111.1111.0000.0000
im Subnetz	129.206.218.0	1000.0001.1100.1110.1101.1010.0000.0000

Also: Festlegung des Routing durch Angabe (Subnetznummer,Maske)

Netznamen

Namen im regionalen Netz *wide area network* WAN

Problem

Integration von Diensten mehrerer Domänen,
konsistente, zeitveränderliche Ressourcentabelle – WIE?

Lösung CCITT X.500 (1988)

DAP	*Directory Access Protocol*	Dateizugriff
DSP	*Directory Service Protocol*	Server-Server Kommunikation
DISP	*Directory Information Shadowing Protocol*	
LDAP	*Lightweight DAP*	vereinf. DAP-Version auf TCP/IP

Beispiel Windows NT

ADS *Active Directory Service* *nutzt LDAP*

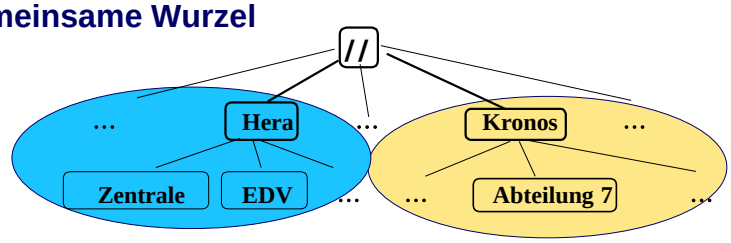
- Ressourcen sind Blätter im Pfadbaum *<DomänenId>://<Pfad>*
- „Aktive Objekte“: Jede Änderung im Verzeichnis wird dem Knoten darüber mitgeteilt (z.B. Druckerstatus)
- Nur die letzte Änderung an einem Objekt bleibt erhalten

Netznamen

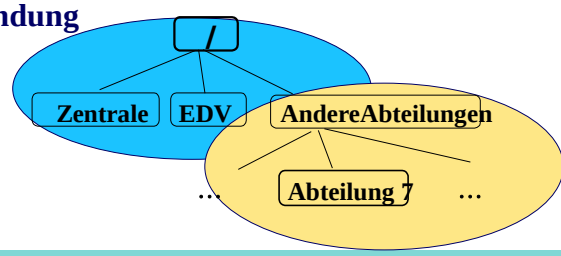
Namen im lokalen Netz *local area network LAN*

Zusammenschluß mehrer Rechner

- **gemeinsame Wurzel**



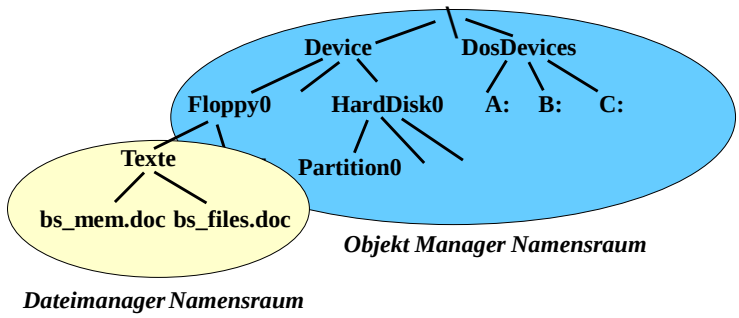
- **Einzelverbindung**



Dateinamen: Windows NT Namensraum

- **Wiederholung**: *Symbolic link parsing*-Methode

Beispiel *Lese Datei* A:\Texte\bs_files.doc



Objekt manager: **A:**Texte\bs_files.doc → \Device\Floppy0\Texte\bs_files.doc

Datei manager: *Lese* Texte\bs_files.doc

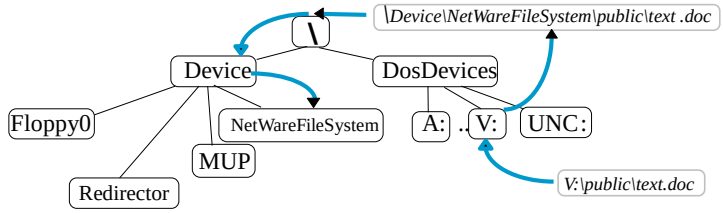
Netzkommunikation

Beispiel **Windows NT** Namensraum im lokalen Netz

- **Symbolic link**

  parse-Methode der Treiber (MS Redirector, Novell NetWare File System) führt zum Netzverbindungsaufbau.

  Beispiel: Neuer „Laufwerks“buchstabe `V:` für Netzverbindung + Dateiname führt zu Umleitung „V:\public\text.doc“



- **Universal Naming Convention** UNC

  Beispiel \\ *textserv\public\text.doc*

  → *UNC*: *\textserv\public\text.doc* → *\Device\MUP* *\textserv\public\text.doc*

  → *\Device\NetWareFileSystem* *\textserv\public\text.doc*

Netzkommunikation: Ports

- **Konzept** Punkt-zu-Punkt Kommunikation („Kommunikationspunkte“)

  Beispiel TCP/IP: *well known port numbers*

Dienst	Portnummer	Protokoll
HTTP	80	TCP
FTP	21	TCP
SMTP	25	TCP
rlogin	513	TCP
rsh	514	TCP
portmap	111	TCP
rwhod	513	UDP
portmap	111	UDP

Unix: /etc/services

Windows NT:
\system32\drivers
\etc\services

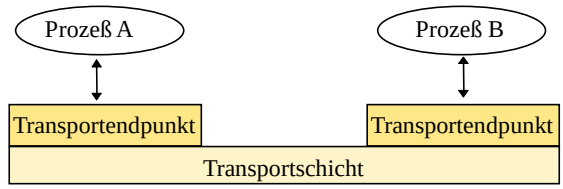
Netzkommunikation: Ports

- **Nachrichtenbasierte Punkt-zu-Punkt Kommunikation**

(**Protokoll**, RechnerAdresse von **A**, ProzeßId von **A**,
RechnerAdresse von **B**, ProzeßId von **B**)

Beispiel UNIX *Transport Layer Interface TLI*
X/Open: Extended Transport Interface XTI

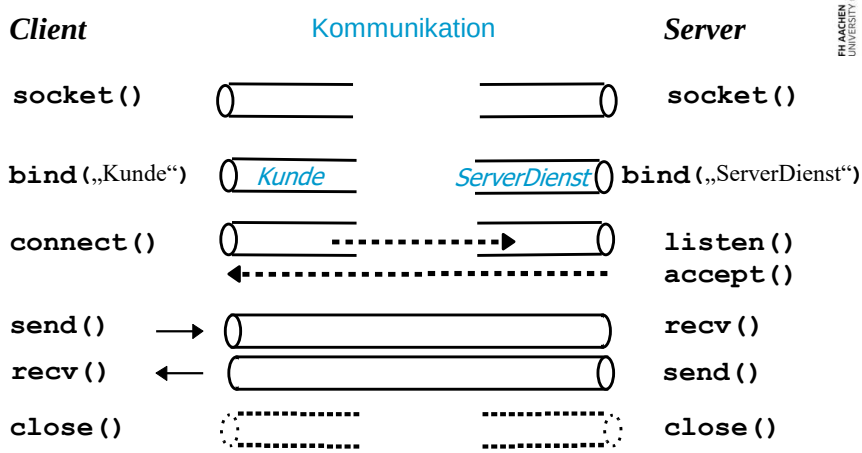
Transportendpunkte (Synchron/Asynchron)



Problem: Zwischenschicht transparent, ohne Beeinflussung

Netzkommunikation: Sockets

- **Verbindungsorientierte Punkt-zu-Punkt Kommunikation**



Netzkommunikation : Named Pipes

Globales Konzept: *Named pipe* („Netzwerk/Pfadname“)

=> LAN-Interprozeß-Kommunikation

- **Unix**

  Named pipe = *special device* ⇒ nur IPC auf **selbem** Rechner, **nicht** NFS

  Named pipe = SystemV: STREAM **socket pair() / bind()**

- **Windows NT**

  CreateNamedPipe() : Objekt im globalen Namensraum, auch NetzPfad

  IPC = ReadFile() / WriteFile()

  UNC-Name = „*ComputerName*\PIPE*PipeName*“

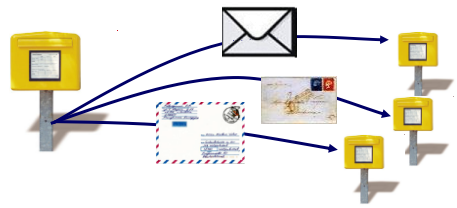
  Lokale pipe: „\\ .\PIPE*PipeName*“

Kommunikation zu Unix möglich, wenn LAN-Manager für Unix *LM/U* installiert.

Netzkommunikation: Mailbox

- **Konzept**: Briefkasten ex. für Sender und Empfänger

  Multicast & Broadcast möglich



- **Probleme**: keine garantierte Reihenfolge,

  kein garantierter Empfang

Netzkommunikation: Mailbox

Beispiel Windows NT *mail slots*

Briefkasten = mail slot, erzeugt mit `CreateMailslot(`*MailBoxName*`)`

Senden: `CreateFile(`*MailSlotName*`)-WriteFile()-CloseFile()`
mit *MailSlotName* = „*ComputerName*\mailslot*MailBoxName*“ *(UNC)*

bei *ComputerName*= „.“ ⇒ lokale IPC
bei *ComputerName*= „*“ ⇒ Broadcast an alle angeschlossenen Rechner
bei *ComputerName*= „*DomainName*“ ⇒ Broadcast an alle Rechner der Domäne

Empfänger sind jeweils alle Briefkästen mit dem angegebenen Namen, falls ex.

Einschränkungen:

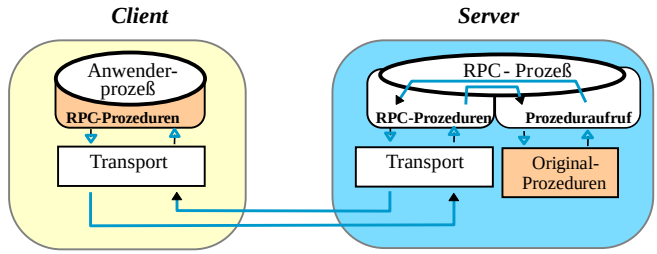
Nachrichtenlänge bei NetBEUI: 64kB bei Punkt-zu-Punkt, 400Byte bei broadcast
Höheres Protokoll erforderlich für Reihenfolge&Empfang etc., da UDP.

Netzkommunikation: RPC

- **Konzept: Prozedur-Fernaufruf**

  Remote Procedure Calls RPC
  Remote Method Invocation RMI *Java!*
  Remote Function Call RFC

  Form: wie normaler Prozedur/Methodenaufruf, Ausführung durch Netzwerkdienst & Transport bleiben verborgen (*Client-Server Standardmechanismus!)*

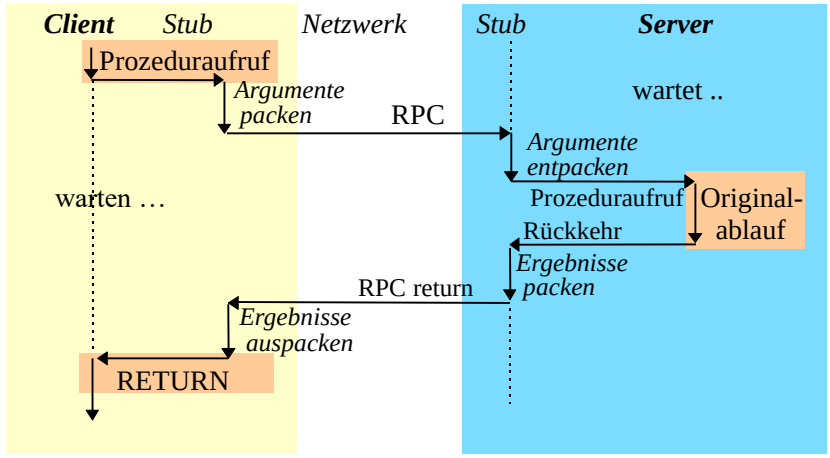


Syntaxformen *Wetter=7*

Stub-Procedure: ComputeWetter(heute) → RPC(*7*, „heute“)
StdProc+*Arg*. RPC(*7,“heute“*)

Netzkommunikation: RPC

RPC-Ablauf



Netzkommunikation: RPC

Transport der Daten

Problem: Hardwareformat von Zahlen
RPC-Argumente sollten maschinen**un**abhängig sein!

- **Big endian** Motorola 680X0, IBM 370

höherwertig			niederwertig
Byte**0**	Byte1	Byte2	Byte**3**

- **Little endian** Intel 80X86, VAX, NS32000

höherwertig			niederwertig
Byte**3**	Byte2	Byte1	Byte**0**

Transport: Umkehrung der Byte-Reihenfolge

Lösung: *data marshaling*, z.B. mit XML, Java Serialisierung, ...

auch für *compiler data alignment* (Adreßgrenzen bei *records*, Wortadressierung,...)

Netzkommunikation: RPC

Beispiel Unix

- Spezielle C-Bibliotheken */lib/libc.a;* SystemV: */usr/lib/librpc.a*
- RPC über NFS
- Schichtenmodell RPC/XDR *external data representation*

RPC-library

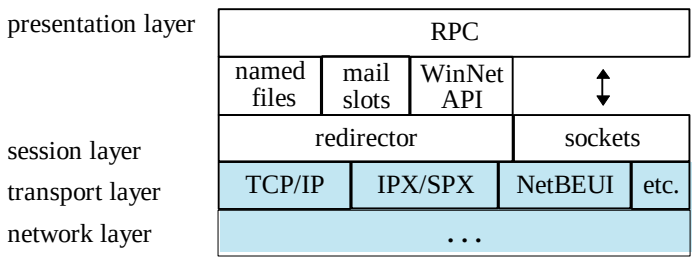
Client: anmelden mit **`registerrpc()`** Client: **`callrpc()`** Server: **`svr_run()`**
`clnt_`. ../ **`svc_`**... Parameter des Transportprotokoll TCP/IP setzen/lesen Berechtigungen setzen/lesen
`Pmap_`.., **`ath_`**.., **`xdr_`**.. Details des Protokolls: Vorsicht!

- RPC bei DCE: Compiler für spezielle Interface Definition Languge IDL.
  RPC durch stub-Aufrufe und Laufzeitbibliothek für Transport

Netzkommunikation: RPC

Beispiel Windows NT

- Verbindungslose RPC: anonymer Service (asynchron)
- Verbindungsorientierte RPC: bestimmte Prozeduren vom Server (synchr.)
- *Network Data Representation* (NDR)-Format
- Programmierung durch Microsoft IDL-Compiler MIDL
- Protokoll-Wahl durch Namensnotation:
  z.B. „ncacn_ip_tcp: MyServer[2004]“ = TCP/IP-Protokoll zu MyServer, port 2004



Netzwerke

Kommunikation in Netzen

Dateisysteme im Netz

Arbeitsmodelle im Netz

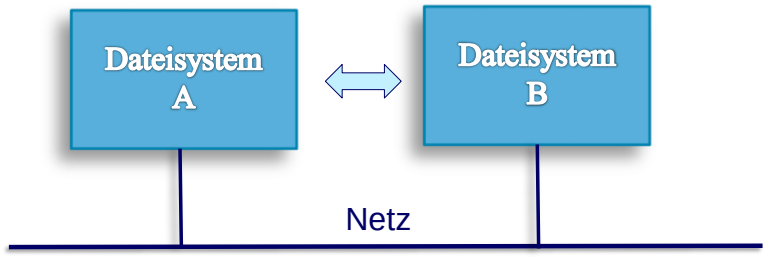
Dienste im Netz

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Dateisysteme im Netz

Synchronisationsprobleme

z.B. inkrement. Backup



Datei, Ordner
- neu
- gelöscht
- überschrieben
- umbenannt

⟺ ?

gegenüber früherem Synchronisationspunkt

Synchronisationsstrategien

Situation: **Datei** in *A* gegenüber **Datei** in *B*

▪ **Weil ….**	existiert in **A**, aber nicht in **B**	existiert in **B**, aber nicht in **A**
neuer umbenannt	B → A umbenennen	A → B umbenennen
älter umbenannt	A → B umbenennen	B → A umbenennen
neu erstellt	A → B kopieren	B → A kopieren
später gelöscht	auch in A löschen	auch in B löschen

Konfliktfall: Nach letztem Sync
Datei in **A** geändert und in **B**

- ist neuer in A
  A → B kopieren
- ist älter in A
  B → A kopieren

Synchronisationsstrategien

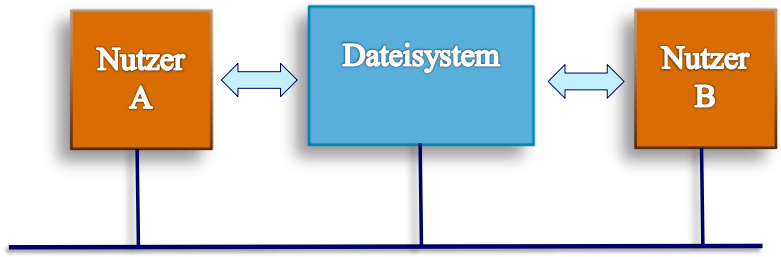
Situation: **Ordner** in *A* gegenüber **Ordner** in *B*

- existiert in beiden
  - Dateien darin synchronisieren
- existiert in A, aber nicht in B
  - neuer umbenannt: B → A umbenennen
  - älter umbenannt: A → B umbenennen
  - neu erstellt: A → B kopieren mit Inhalt
  - in B neu gelöscht: auch in A löschen mit Inhalt
- existiert nicht in A, aber in B
  - analog behandeln, s.o.

Problem: Versionsgeschichte (z.B. Löschinformation) ist nicht vorhanden
→ **Journaling Filesystem** ist nötig!

Dateisysteme im Netz

Zugriffssemantiken



Netz

z.B. gemeinsames Erstellen eines Reports

Wer darf wann schreiben ?

Dateisysteme im Netz

Zugriffssemantiken

- *Read Only File*
  Problemlos, da alle Kopien aktuell sind, unabhängig von der Pufferung
- *Operationssemantik* *race conditions*
  Alle Änderungen werden sofort umgesetzt; die zeitlich nächste Operation bemerkt die Folgen der vorigen
- *Sitzungssemantik* *race conditions*
  Alle Änderungen werden nur auf einer Kopie ausgeführt.
  Am Ende der Sitzung wird das Original überschrieben.
- *Transaktionssemantik*
  Atomare Transaktion: Während der Sitzung ist die Datei gesperrt.

Problem: Zugriffssemantik hängt von der *Implementierung* ab (Hardware, Existenz von Puffern, Netzprotokollen, ...)

Beispiel *Operationssemantik*: Reihenfolge der Operationen = Inhalt hängt von der Kommunikationsgeschwindigkeit (Leitungsgeschwindigkeit, Netzstruktur, CPU-Takt, BS-Version, Lastverteilung, ...) ab.

Dateisysteme im Netz

Zustandsbehaftete vs. zustandslose **Datei-Server**

= verbindungsorientierte Kommunikation
vs. verbindungslose Kommunikation

Server-Dienst/Verbindung eröffnen

- Datenstrukturen für Zugriff aufsetzen (Kennungen etc.)
- Zugriffsrechte prüfen
- Puffer einrichten

Server-Dienst/Verbindung nutzen

- Mit Dateikennung lesen/schreiben
- Auftragskopien werden über gleiche Sequenznummern erkannt

Server-Dienst/Verbindung schließen

- Puffer leeren + deallozieren
- Datenstrukturen abbauen

Dateisysteme im Netz

Zustandsbehaftete vs. zustandslose **Server**

Vorteile

- Schneller Zugriff: *keine Adreßinfo, keine Berechtigungsprüfung*
- Effizienter Cache: *Strategien möglich (read ahead etc.)*
- Vermeiden von Auftragskopien *Nummerierung der Aufträge*
- Dateisperrung möglich (Exklusiver, atomarer Zugriff) *Datenbanken!*

Nachteile

- Client crash: *kein Löschen der Strukturen+Puffer*
- Server crash: *kein Löschen der Strukturen+Puffer, Dateizustand ungewiß*
- Begrenzte, gleichzeitig benutzte Dateienzahl: *begrenzte Speicherbelegung*

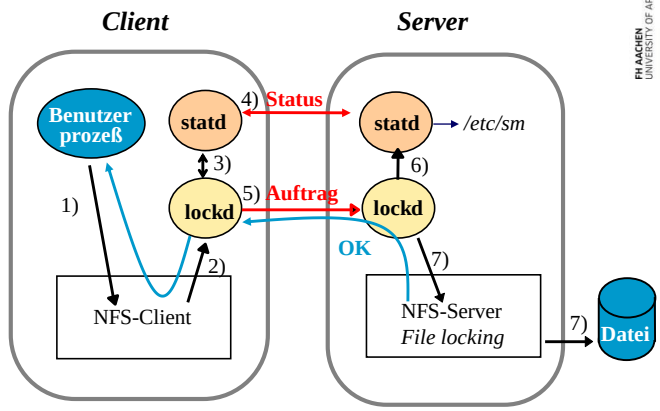
Fazit: Server(Verbindung) **mit** Zustand kann Dateien reservieren, Auftragskopien vermeiden.

Server(Verbindung) **ohne** Zustand ist fehlertoleranter, kann mehr Benutzer gleichzeitig verwalten.

Dateisysteme im Netz

Beispiel Unix *NFS-Server*
Auftrag: file locking *Network Lock Manager*

- Zustandslose Client & Server
- Zugriffsinfo auf Client +Server gespeichert
- File lock durch RPC



Frage: Sind Verklemmungen möglich ?

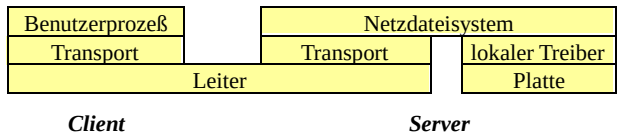
Dateisysteme im Netz: Cache

Cache und Puffer

Vorteil: Puffer auf Client beschleunigt Lesen/Schreiben

Nachteil: lokaler Puffer führt zu Inkonsistenz bei Zugriffen anderer Rechner

Mögliche Pufferorte:



- Benutzerprozeß *Heap/Stack*
- Transport Client *Ausgangspuffer*
- Leiter *1GHz auf 3 km=10kBit*
- Transport Server *Eingangspuffer*
- Netzdateisystem *Dateipuffer*
- Lokaler Treiber *Blockpuffer*
- Plattencontroller *Schreib-/Lesepuffer*

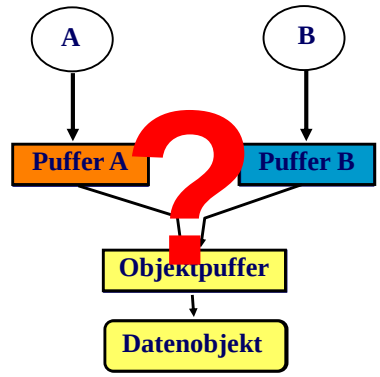
Dateisysteme im Netz : Cache

Problem: Konsistenz der lokalen Cache

A, B lesen

A schreibt

B schreibt



Inkonsistenz !

Dateisysteme im Netz: Cache

Cache und Puffer: Konsistenzstrategien für lokalen Cache

- *Zentrale Kontrolle*
  Vor dem Lesen Vergleich der Änderungsinfos (VersionsNr, Quersummen) zwischen Client und Server
  ***aber**: aufwändig!*
- *Delayed Write*
  Sammeln der Änderungen, dann erst schicken
  ***aber**: Zugriffssemantik verändert*
- *Write On Close*
  Sitzungssemantik: lokale Kopie geht an Server bei `close()`
  ***aber**: Inkonsistenzen durch Sitzungssemantik*
- *Write Through*
  Änderungen gehen am Puffer vorbei sofort zum Original
  ***aber**: langsam*

Fazit: Puffern auf Serverseite ist einfacher - auf Clientseite effizienter, aber komplexer (semant.Protokolle!)

Dateisysteme im Netz: Cache

Beispiel UNIX *NFS-Cachestrategien*

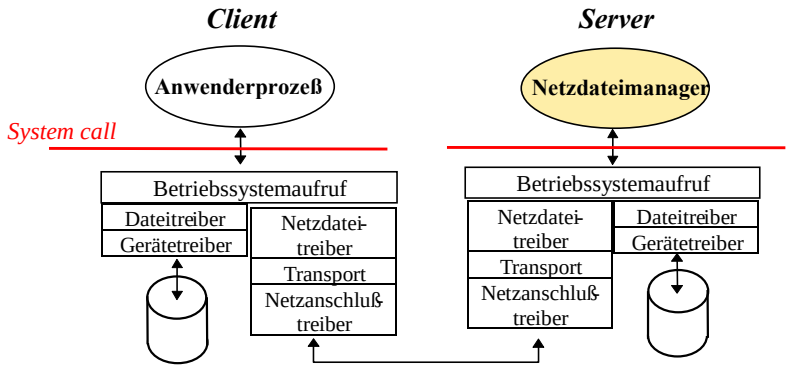
Asynchrone RPC durch *basic input output* **biod** – Dämonen

- *Read ahead*
  Vorauseilende Anforderung von Benutzerblöcken
- *Delayed write*
  Pufferung aller Schreibdaten,
  `flush()` alle 3 s (Daten) , 30 s (Verzeichnisse), bei `sync()`, Puffer belegt
- *Write through*
  bei exklusiv gesperrten Dateien

Code aus Effizienzgründen im Kernel

Dateisysteme im Netz: Dateiserver

Implementierung eines Dateiserver durch **Prozesse**

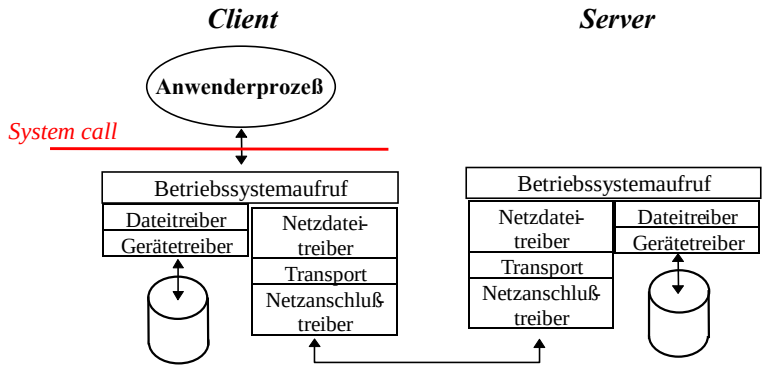


Vorteil symmetrisches System, jeder kann beides sein

Nachteil Kopieren der Systempuffer kernel space/user space

Dateisysteme im Netz: Dateiserver

Implementierung eines Dateiserver durch **Treiber**

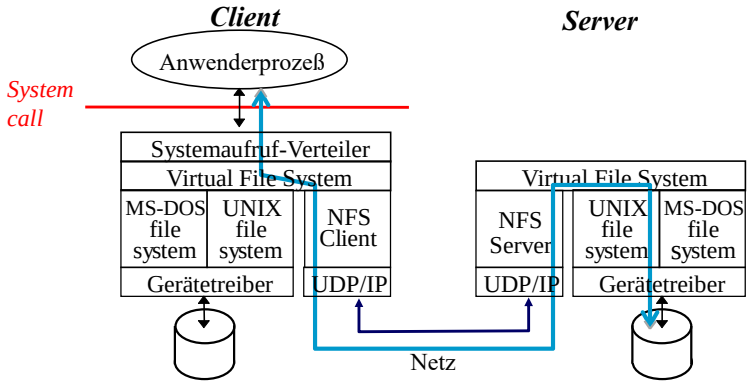


Vorteil schnelles System
Nachteil asymmetrische Kerne

Dateisysteme im Netz

Beispiel Unix Das NFS-System

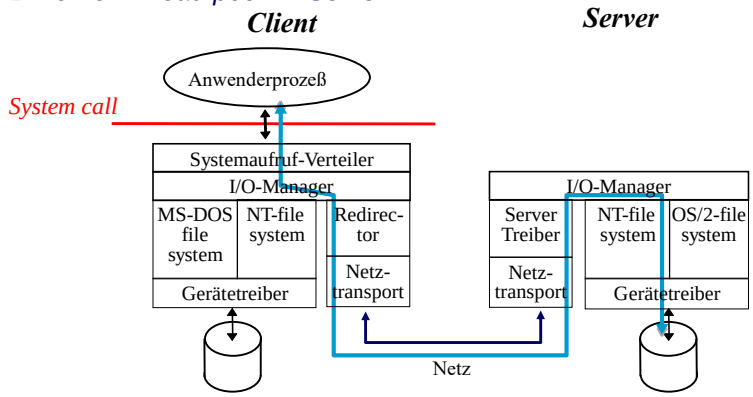
- **`Mount()`** zum Einhängen eines Server-Dateisystems
  Prozesskommunikation zum *mount-demon*
- **`Nfs_svc()`** im *kernel mode* auf dem Server
- Virtual i-nodes für virtuelles Dateisystem



Dateisysteme im Netz

Beispiel Windows NT — Netzdateisystem

- Verbindungsorientierter Netzaufbau durch Redirector mit *Transport Driver Interface* TDI über *virtual circuits* (Kanäle)
- Kein virt. Dateisystem, sondern „Durchsuchen“-Methode für Kanäle
- *Kernel Thread pool* im Server



Dateisysteme: Sicherheitsaspekte

Problem

Inkonsistente Netz-Kopplung von Systemen bei unterschiedlichen Sicherheitsmechanismen !

z.B. Authentifizierung bei unterschiedlich langen Namen und Groß/Kleinschreibung Unix/WinNT vs MS-DOS, fehlende ACLs, ...

- **Beispiel Unix** *NFS-Sicherheitssystem NIS*

  Benutzerliste *(yellow pages)* verwaltet von NIS
  RPC hat Zugriffsrechte user/group/other
  SuperUserID=0 auf Client ⇒ UserId=-2 auf Server („external Super User“)
  konsist. Behandlung von gleichen NutzerIds unterschiedl. Systeme

- **Beispiel Windows NT**

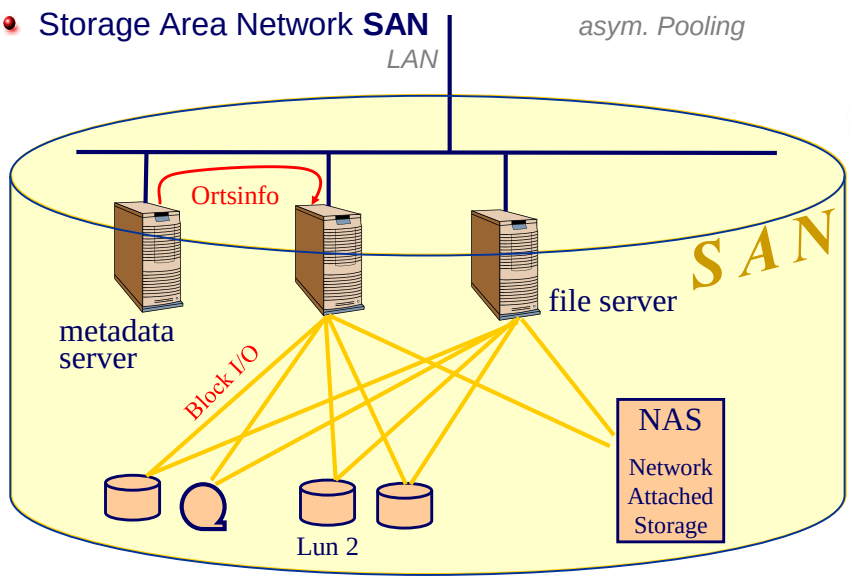
  NT 4.0: ACL, Netzbenutzer müssen beim SAM registriert sein mit gleichem Paßwort, sonst Nachfrage bzw. Ablehnung

  NT 5.0: Kerberos-System bei netzweiter Zugangskontrolle

Dateisysteme: Virtueller Massenspeicher

- Storage Area Network **SAN**

asym. Pooling



Dateisysteme: Speichernetze

Speicherkonfigurationen des Storage Area Network SAN

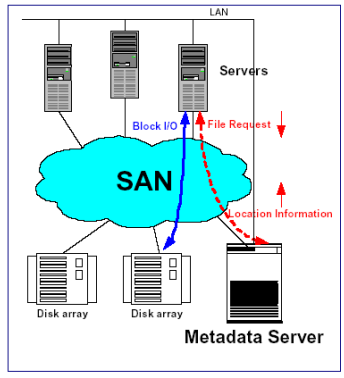


Figure 1:
Metadata Server (Asymmetrical Pooling)

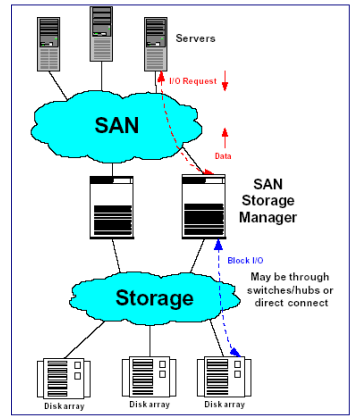


Figure 2:
SAN Storage Manager (Symmetrical Pooling)

Dateisysteme: Speichernetze

- **Info** SNIA-Schichtenkonzept

Netzwerke

Kommunikation in Netzen

Dateisysteme im Netz

Arbeitsmodelle im Netz

Dienste im Netz

Arbeitsmodelle im Netz

Anforderungen an *Load Sharing Facility*-Systeme

- Ausgewogene **Lastverteilung** verschiedener Jobarten&Leistungsklassen
- Zentrale **Warteschlangen** für Rechenzeit, Speicherbedarf, I/O-Geräte..
- **Jobdurchlaufzeiten** minimieren
- Transparente **Lizenzverwaltung** (unabh. von RechnerId)
- Normaler, **interaktiver Betrieb** soll möglich sein
- **Nutzung** der Rechner nachts, im Urlaub, an Wochenenden..
- Übersichtliche **Konfiguration**, leichte Wartbarkeit

Arbeitsmodelle im Netz

Probleme bei der Lastverteilung

- Jobübermittlung kostet Zeit *(nur größere Arbeitspakete!)*
- Datenübermittlung kostet Zeit *(nur kommunikationsarmer Code!)*
- Inhomogene Rechnerarchitekturen, inkompatibler Maschinencode *(nur portabler Code!)*

Arbeitsmodelle im Netz

Konzept Netzcomputer NC:

HW: CPU, Hauptspeicher, Bildschirm/Tastatur,Netzanschluß
SW: Nur Mikrokern, ***Keine*** Peripherie

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

Vorteile

- Billigere Hardware (Anschaffung)
- Aktuelle Daten und Programme (durch zentrale Wartung)
- Billigere Wartung (Konfiguration, SW-Pflege, HW,..)
- Höhere Datensicherheit (Ausfall, Datendiebstahl,Viren..)
- bessere Ressourcennutzung (Massenspeicher, Peripherie,...)
- **weniger Energie** **„green IT“**

Nachteile

- Erhöhter Netzaufwand (HW für Netz und Server)
- Erhöhter Pufferaufwand (RAM für Netz- und Datenpuffer, swap-Disks,...)
- **Benutzerbevormundung** (Zentrale entscheidet über Daten+Applikationen)

Arbeitsmodelle im Netz

Arbeitsverteilung durch JAVA-Applets

- **Lastverteilung** durch portablen Byte-Code
- **Sicherheit** durch Java Virtual Machine (*byte code interpreter*)

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

NC-Ablaufumgebung („Betriebssystem“) der Applets

- Interpretation des Java Byte Code
- Hauptspeicherverwaltung *(garbage collection)*
- Isolierung gleichzeitig ablaufender Programme (*sand box*)
- Standardfunktionen *Grafik-Ein/Ausgabe, Audiowiedergabe*
- Kein Platten/Peripheriezugriff unsignierter Applets

Browser-Ablaufumgebung

Java Plug-in: applikationsabhängige Funktionserweiterung

Arbeitsmodelle im Netz

Mobile Peripheriegeräte bei unzuverlässigen Verbindungen

Probleme

- *Roaming*
  Keine konstante Arbeitsumgebung für Außendienstmitarbeiter, Telearbeiter, ... bei wechselnden Arbeitsplätzen
- *Wartung*
  Keine Konfigurationspflege, Programmaktualisierung, Datensicherung, ...

mögliche Lösungen

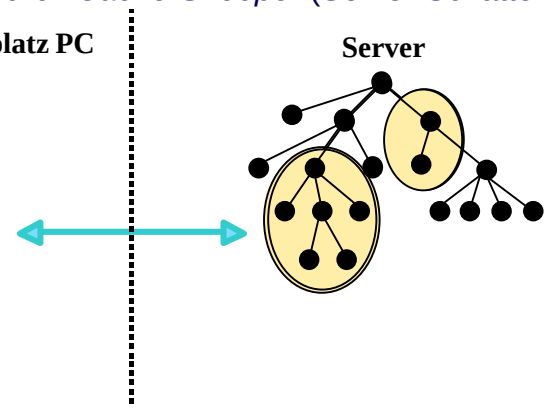
- Zentrale Aktualisierung
  Pro: konsistente Wartung. **Contra**: Nicht-einheitliche Systeme verboten
- Dezentrale Aktualisierung
  Pro: lokal angepasste Nicht-Standardfkt. **Contra**: arbeitsaufwändig

Kernfunktionen zentral gewartet: **Schattenserver!**

Arbeitsmodelle im Netz

Schattenserver zur Datenhaltung

- Initiale Festlegung gespiegelter Pfadteile
- Arbeit wie ein Netzcomputer (Daten+Programm-Cache)
- Aktualisierung durch *Cache-Snooper* (Server-Schatten)



Arbeitsmodelle im Netz

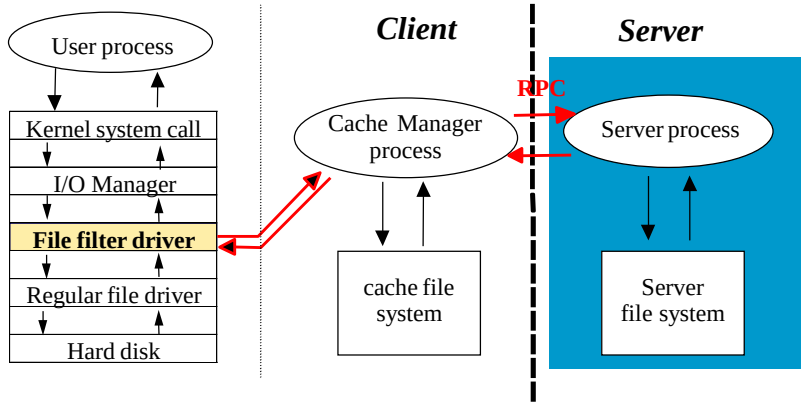
Schattenserver Vorteile

- Automatische Datei- und Programmaktualisierung
  ohne Administratoraufwand!
- Atomatische Datensicherung
- Netzunabhängiger *stand-alone* Betrieb möglich
- Benutzerangepaßte Konfiguration
- Rechnerunabhänge Konfiguration
- Geringere Wartungskosten

Arbeitsmodelle im Netz

Beispiel Linux *Coda-Dateisystem*

Abfangen von Systemaufrufen *CreateFile, OpenFile, CloseFile, RenameFile*



Arbeitsmodelle im Netz

Windows NT Schattenserver

- Deklaration von Netz-bekannten Ordnern („Freigabe“)
- Einrichtung als „Offline-Dateien“
- Resynchronisierung („Offline-Aktualisierung“) bei login/logout.
- Semantik (Überschreiben, Speichern mit neuem Namen, usw.) individuell pro Datei oder einmalig für alle Dateien festgelegt.

FRAGE

- Was ist der Unterschied zwischen einem NC-System und einem Schattenserver-System?

- Ein NC hat keine Software, jede Applikation wird immer zentral geladen. Funktioniert nur mit Netz.
- Ein Schattenserver-Client hat alle Applikationen, die benötigt werden. Funktioniert auch ohne Netz.

Netzwerke

Kommunikation in Netzen

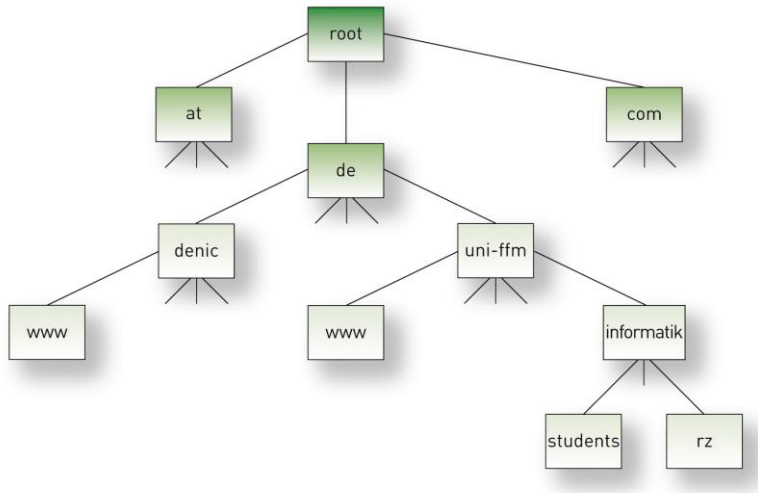
Dateisysteme im Netz

Arbeitsmodelle im Netz

Dienste im Netz

Dienste im Netz

- **DNS - Domain Name Service:** Hierarchie der Domänen



Dienste im Netz

- **DNS - Domain Name Service**

  Zuordnung *IP-Nummer* ↔ *Internetname*

  `192.138.228.1` `siemens.com`

hierarchisches System: Jeder Domäne ihr DNS (top level **.de** ca. 8 Stück)

Beispiel Konsoleneingabe an DNS

```
nslookup siemens.com
Server:styx.rbi.informatik.uni-frankfurt.de
Address: 141.2.15.5
Name:     siemens.com
Address: 192.138.228.1
```

Dienste: **ping** <URL,IP-Adr> Echo eines Rechners

traceroute (tracert) <URL,IP-Adr> Alle auf der Route

Dienste im Netz

- **FTP - File Transfer Protocol** *verbindungsorientiert*
  - Anzeige des Inhaltsverzeichnisses des Ordners im Dateisystem
  - Senden einer Datei
  - Empfangen einer Datei
  - Umbenennen/Löschen von Dateien

- **WWW - World Wide Web** *verbindungslos*

  die „Killerapplikation“ fürs Internet, weil
  - einheitliche Namen im Netz (URL)
  - einfache Navigation
    Hyperlinks: „auf Knopfdruck“ zwischen den Seiten
  - genormte Seitenbeschreibungssprache HTML
    Bilder, Audio und Video in einem gemeinsames Dokument
  - verbindliches Protokoll (z.B. HyperText Transport Protocol HTTP)
    zum Transport von Dokumenten zwischen Server und Client

Dienste im Netz

- **EMAIL - Electronic Mail**

  Typische Merkmale:

  - Asynchrones, zeitlich entkoppeltes Senden und Empfangen
    Der Sender kann weiterarbeiten, ohne auf den Erhalt der Nachricht durch den Empfänger warten zu müssen.
  - Die Nachrichten werden zwischengespeichert
    Keine direkte Verbindung zwischen Sender und Empfänger muss ex.

  SMTP für synchrones Senden (und Empfangen),
  POP für asynchr. Empfangen, IMAP für Mailbox-Dienst (virt. Ordner)

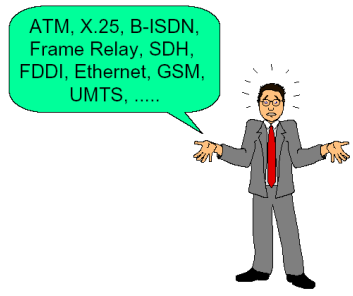
- **NEWS - Usenet**
  - Dezentrale Diskussionsforen
  - Name ist hierarchisch organisiert: z.B. comp.lang.c, rec.games.chess
  - Top level:

sci	science-Wissenschaft
soc	society-Gesellschaft, Politik
rec	recreation-Hobbys, Essen Trinken
comp	computer
alt	alternative-Klatsch, Tratsch, Unkonventionelles

Middleware

- **Problem**: heterogenes **Chaos** aus Produkten und Normen

Netze

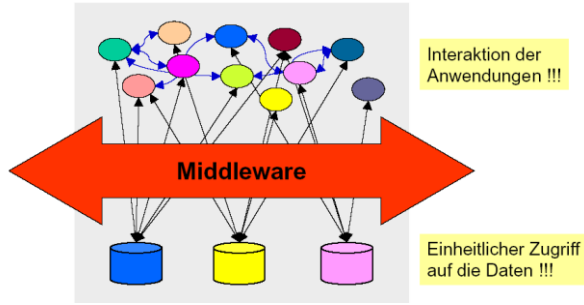


IT-Konsolidierung ?

Middleware

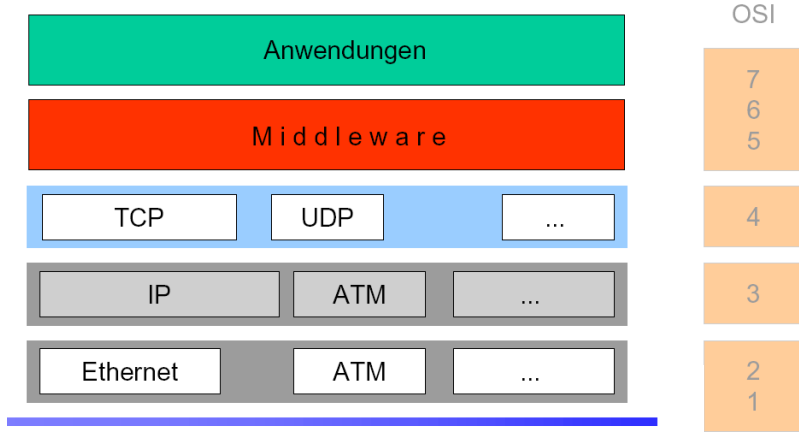
- Mögliche Lösung: **Monokultur**
  Probleme: - teuer
  - dauert lange, um alles zu implementieren
  - Firmenabhängigkeit
- **Bessere** Lösung: VermittlungsSW **„Middleware“**

Enterprise Application Integration



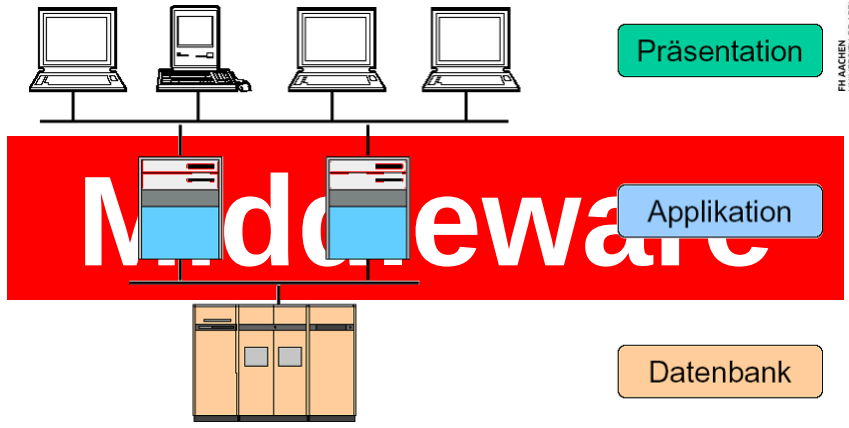
Middleware Kommunikation

- Middleware setzt direkt auf Transportprotokollen auf:
  Applikationen werden **unabhängig** vom Transportprotokoll



Middleware und 3-tiers System

- **3-Schichten-System** (SAP/R3)



mittlere Schicht = Middleware

Middleware-Arten

- **Spezielle Anwendungen**
  - Dateitransfer — Fernzugriff auf gemeinsame Dateien
  - Datenbankzugriff — Datenzugriff auf entfernte DB
  - Transaktionsverarbeitung — Koordination verteilter Transaktionen
  - Groupware — Zusammenarbeit in Gruppen
  - Workflow — Organisation arbeitsteiliger Prozesse
- **Allgemeine Dienste**
  - Virtual Shared Memory: — Zugriff auf virtuell gemeinsamen Speicher
  - Remote Procedure Call (RPC) — Aufruf einer entfernten Prozedur
  - Message Passing — Send/Receive–Kommunikation
  - **Object Request Broker** — Dienstvermittlung im Netz

Dienstvermittlung im Netz

- **CORBA** = **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
  Dienstvermittler

Aufgaben

- Initiale Registrierung aller Dienste im Netz
- Registrierung einer Anfrage
- Ermitteln des passenden Servers
- Übermitteln des Auftrags + Parameter
- Übermitteln des Ergebnisses
- Auftragsabschluß

Referenzimplementierung durch Object Management Group OMG 1989

Problem: langsam!

Dienstvermittlung im Netz

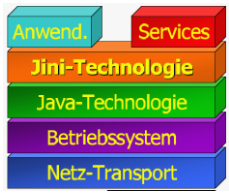
- **Die sieben Trugschlüsse verteilter Anwendungen**
  - Das Netzwerk ist immer verfügbar
  - Die Wartezeit (engl. latency) ist Null
  - Die Übertragungsrate (Bandbreite) ist unendlich groß
  - Das Netzwerk ist sicher
  - Der Aufbau des Netzwerks ändert sich nicht
  - Es gibt nur einen Administrator
  - Es fallen keine Transportkosten an

Gegenkonzept: Jini *Java Intelligent Network Infrastructure*

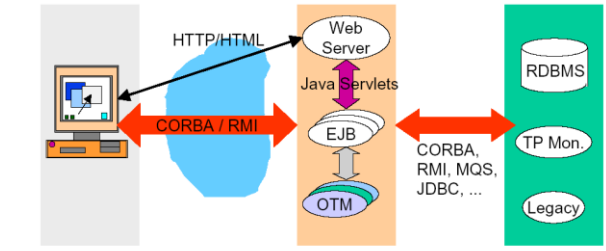
Dienstvermittlung im Netz

- **Java-Technologie** **Jini**
  - Code-Mobilität
  - Protokoll-unabh. Programme
  - Flexibilität & Integration der Netzknoten mit RMI
  - Leasing: Automat. Rekonfiguration des Netzwerks

Jini



Anwendung: Thin client/Thick Server

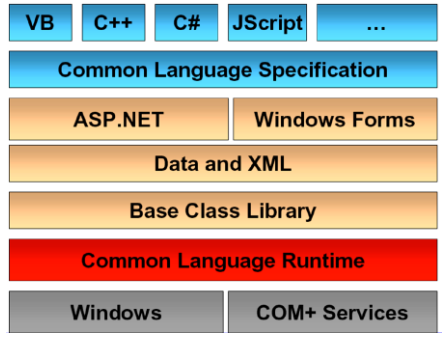


EJB *Enterprise Java Beans*
OTM *Object Transaction Monitor*
Application Server

Dienstvermittlung im Netz

- **Microsoft Middleware**

COM	binäre Schnittstellendef. für IPC auf selbem Rechner
DCOM	verteiltes COM im Netz (RPC und MS-IDL)
COM+	DB-Anwendung mit Transaction Server MTS incl. Lastverteilung, DB-Cache, async. Aufrufe
.NET	Programmier- + Laufzeitumgebung für ORB & uPnP



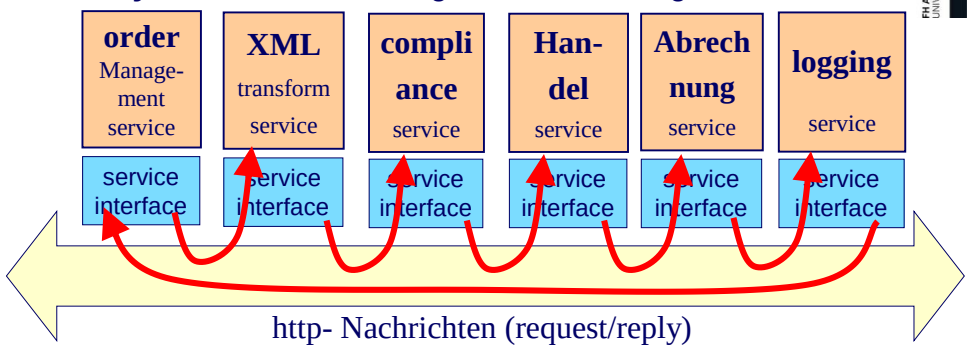
Service-Orientierte Architektur SOA

- **Problem**: Viele alte, teuer zu wartende Dienste
- **Lösung**:
  - **Kapselung** durch Middelware-Ansatz
    Die Dienste implementieren bei fest definierten Funktionszusicherungen bestimmte Leistungen, ohne dabei anzugeben, auf welche Art dies geschieht.
  - **Gemeinsames Kommunikationsprotokoll**
    Ihr Aufruf wird durch einen für alle Module einheitlichen, losen (d.h. nachrichtenbasierten) Kommunikationsmechanismus (SOA-Protokoll SOAP) sichergestellt.
- **Voraussetzung**: Modellierung der Geschäftsprozesse (OASIS 06)
  - Grafik, Regeln ("Business Process Management" BPM)
  - Spezifikationen ("Business Process Execution Language" BPEL)

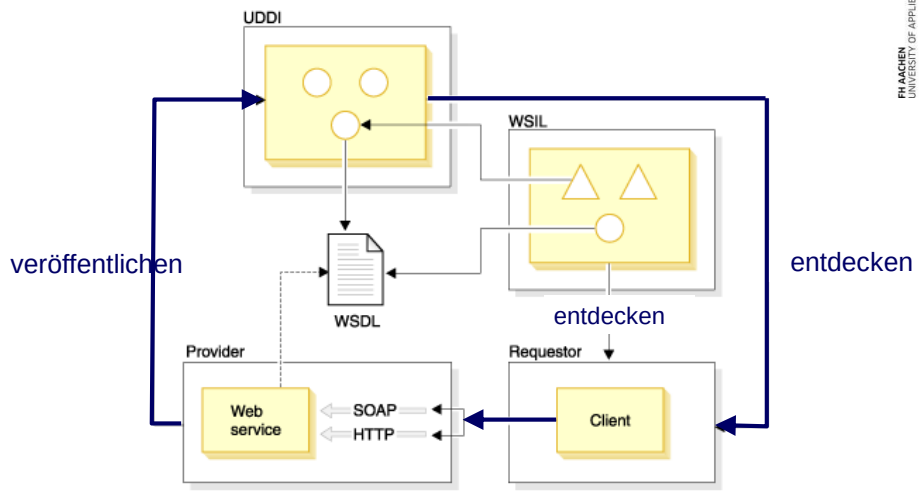
Software-Busstruktur der SOA

- **Beispiel**: Handelsprozess durch **Enterprise Service Bus** (IBM)

Asynchrone Abwicklung einer Bestellung



Beispiel: Web Services WOA



SOA charakt. Anforderungen

- **robuste, skalierbare** Datenübertragung
- **Kapselung** der Services: Keine inkompatiblen Protokolle, Datenformate, Interaktionsmuster der Legacy-, Java-, .NET-Applikationen
- **Serviceorchestrierung**: Modellierung der Geschäftsprozesse und Abbildung auf services
- **Verteilte Services**: unabhängige Installation, Skalierung, Konfigurierung
- **Zentrale Installation**, Administration, Überwachung, Wartung der Services
- **Gemeinsame Datenformate** (XML): einheitliche Weiterverarbeitung, Dokumentation, Auditing möglich

Vorteile SOA

- Neue oder geänderte Geschäftsprozesse können **schneller** und damit **preisgünstiger** durch Kombination bestehender Dienste realisiert werden.
- Bewährte, ältere Systeme können **weiter genutzt** werden, ohne Neuentwicklungen zu blockieren oder sie mit Kompatibilitäts-forderungen zu belasten (Investitionsschutz).
- Die älteren Einzeldienste können dann Stück für Stück durch moderne Versionen (z.B. Hardware-Software-Kombinationen) **ersetzt** werden.
- Durch die Modularisierung, klare Aufgabentrennung und Funktionskapselung werden die Systeme **beherrschbarer** und **leichter wartbar**.
- Damit ist auch eine **Auslagerung** unwirtschaftlicher Teile an Fremdanbieter (*outsourcing*) wird durch die Modularisierung leichter möglich.

ROBOTER-BETRIEBSSYSTEME

Roboter-Betriebssysteme

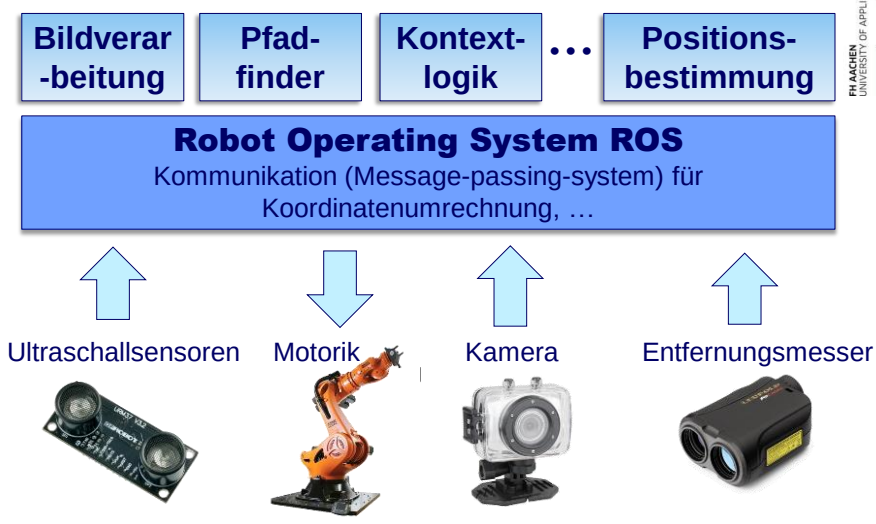
- **Idee**
  Roboter programmieren ist mühsam – warum nicht bewährte Module anderer Gruppen wiederverwenden?

- **Probleme**
  - Sehr viele Module müssen Daten austauschen
  - Sie sind sehr unterschiedlich (LaserScanner, Ultraschall-Sensor, Kameras…)
  - Unterschiedliche Entwicklungsgruppen haben unterschiedliche Schwerpunkte
  - Software wird zeitlich getrennt und an unterschiedlichen Orten für unterschiedliche Roboter entwickelt.

- **Ansatz**: Bibliotheksmodule, z.B. Yet Another Robot Platform YARP

Roboterbetriebssystem ROS

- **Konzept** *Middleware für Roboter-Module* **ROS.org**

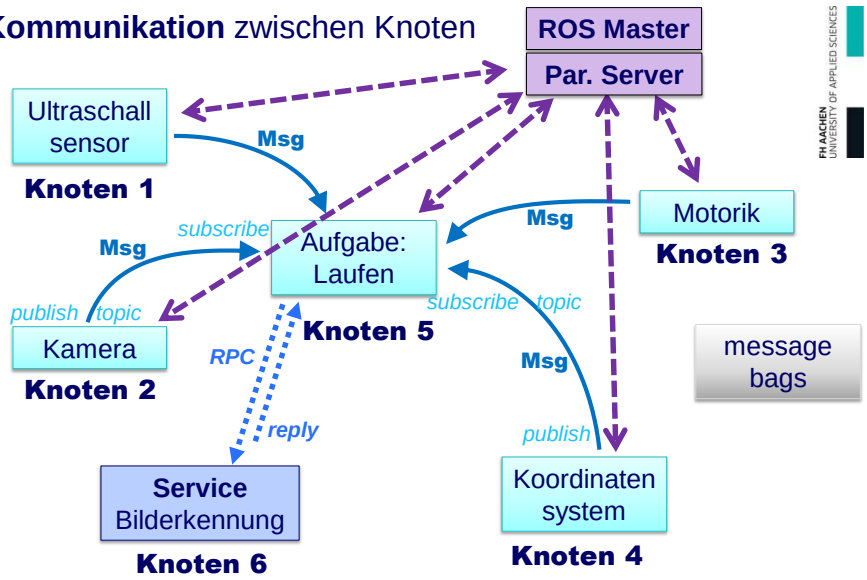


ROS-Kommunikation

- **Message passing**
  - Schnittstellendefinition der Knoten (*message IDL*)
  - Anonymes, asynchrones *Publish/subscribe*-System
- **Aufnahme und Wiedergabe** von Nachrichtenströmen
  - Umleiten, Abspeichern und Einspielen von Sensordaten und Motorsteuerungsbefehlen für Testzwecke
- **Remote Procedure Calls** RPC
  - Bereitstellung von ***services*** für synchrone Kommunikation mittels messages
- **Verteiltes Parametersystem**
  - Globale Datenbasis für die Konfiguration (start-up Zeit, …)

Konzept *graph layer*

- **Kommunikation** zwischen Knoten

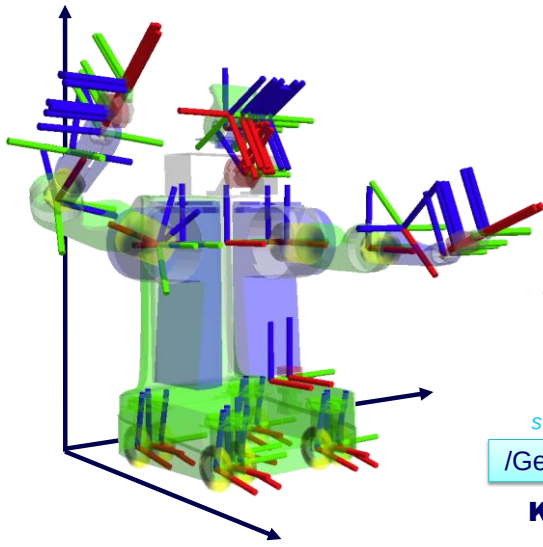


Höhere Konzepte

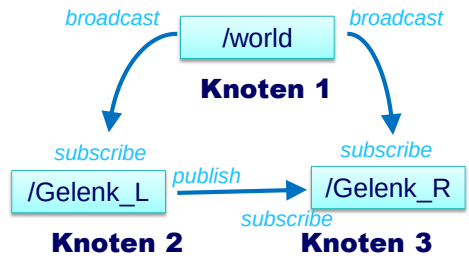
- **Koordinatentransformationen**
  Zeitliche Beschreibung von mehreren Koordinatenrahmen
- **Robotermodelle**
  XML-Formulierung
- **Aufgaben**
  Interface für zeitbegrenzte Bündelung von Kommunikationskanälen gleichen Nachrichtentyps.
- **Nachrichten-Ontologie**
  Nachrichtenarten für verschiedene Zwecke
- **Datenfilter**
  C++ Bibliothek für Filtersequenzen
- **Dynamische Erweiterungen**
  Plugins für C++

Koordinaten-Transformationen

- **Transformations-Bibliothek tf**

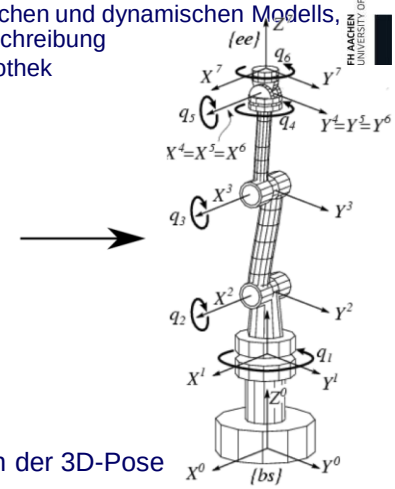


Errechnen einer Position relativ zu den Weltkoordinaten oder zu anderen Positionen



Robotermodelle `robot_model`

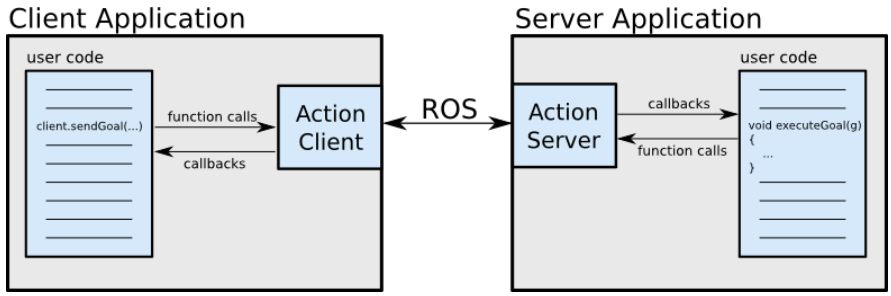
- **urdf** — Unified Robot Description Format: Beschreibung durch XML + Parser
- **srdf** — XML Semant. Beschreibung + Parser
- **kdl_parser** — Erzeugung eines kinematischen und dynamischen Modells, basierend auf der *urdf*- Beschreibung
  → Nutzung der *kdl*-Bibliothek



- **robot_state-publisher** — Publizieren der 3D-Pose

Aufgaben

- **actionlib**
  Ausführen von *user*-Befehlen, die vom *user* während der Ausführung beeinflusst oder abgebrochen werden können.
  Nutzung eines speziellen *ROS-Action-Protokolls*, das auf dem msg-passing-Protokoll aufsetzt mit *goal*, *feedback* und *result*.



Nachrichten .msg

- **Verwendung**
  Kommunikation zwischen Knoten mittels TCP/IP und UDP (*TCPROS, UDPROS*)

- **Format** *Datenstruktur*
  - Felder:

```
uint32 seq      # header, z.B. Sequenznummer
time  stamp     # und Zeit
int32 x         # Datenfelder
bool b=true     # Konstanten
```

- **Einsatz**
  Kompilierung zur verwendeten Modulsprache C++, Python, … mittels *client library (z.B. roscpp, rospy, roslisp...)*

Nachrichten-Ontologie

- actionlib_msgs: *GoalID, GoalStatus, GoalStatusArray*

```
time stamp                  # for preemption of goals at deadline
string id                   # unique ID for feedback
```

- diagnostic_msgs: *DiagnosticArray, DiagnosticStatus, KeyValue,..*

```
# This message is used to send diagnostic information about the state of the robot
Header header                   #for timestamp
DiagnosticStatus[] status       # an array of components being reported on
```

- geometry_msgs: *Polygon, Point, Pose, Transform, Vector3, ….*

```
#A specification of a polygon where the first and last points are assumed to be connected
Point32[] points
```

- nav_msgs: *Path, GridCells, MapMetaData, OccupancyGrid, …*

```
#An array of poses that represents a Path for a robot to follow
Header header
geometry_msgs/PoseStamped[] poses
```

- sensor_msgs *Joy, CameraInfo, FluidPressure, LaserScan, …*

```
# Reports the state of a joysticks axes and buttons.
Header header    # timestamp in the header: time the data is received from the joystick
float32[] axes   # the axes measurements from a joystick
int32[] buttons  # the buttons measurements from a joystick
```

MultiChannelIncrementFilterDouble

Datenfilter

- Filteroperationen, basierend auf C++ templates *FilterBase*
- Konfigurierbar über den Parameter Server
- Leichtes Einrichten einer Filterkette (*FilterChain*).

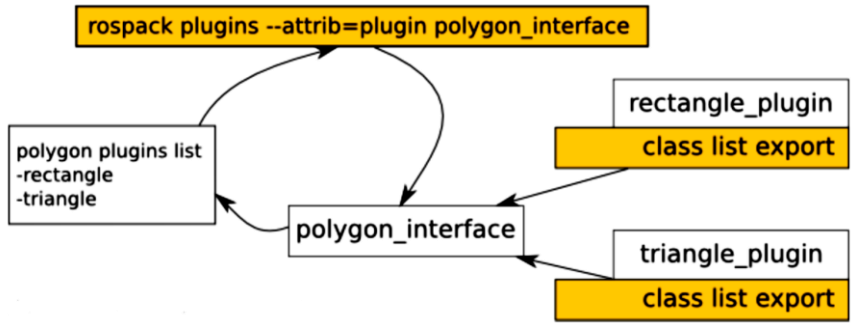
```
param_namespace_passed_to_configure:
 filter_chain:
   -
     name: median_test
     type: MultiChannelMedianFilterDouble
     params: {number_of_observations: 5}
   -
     name: increment
     type: MultiChannelIncrementFilterDouble
     params: {string: my_name}
```

Knoten 1 **Knoten 2**

Plugins für C++

pluginlib
Dynamisches Laden und Entladen von selbst geschriebenen plugin-Bibliotheken bei Bedarf durch vorherige Deklaration in *package.xml*

- **Beispiel** Rechteck und Dreieck deklarieren und abfragen



ROS Module

- Definition des Nachrichtenformats **common_msg**
- Abbrechbare RPC‘s (Aufgaben) **actionlib**
- Roboter Beschreibungssprache **urdf, sdf, kdl**
- Roboter Geometrie-Bibliothek **tf**
- Haltungsschätzung (*pose estimation)* **robot_pose-ekf**
- Lokalisierung 2D **amcl**
- Mapping: laserbasiertes SLAM **gmapping**
- Navigation, auch mobil **navigation**
- Diagnosemodule **diagnostics**
- **+ weitere >2000 Bibliotheken**

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

ROS Werkzeuge

- **rqt** Entwicklungswerkzeug für graf. **Roboter-Bedienungs-oberfläche**: Zusammenfassung unterschiedlicher Fenster und GUI unter einer Oberfläche.

FH AACHEN UNIVERSITY OF APPLIED SCIENCES

ROS Werkzeuge

- **rviz** 3D Visualisierung der Roboter und ihrer Umgebung