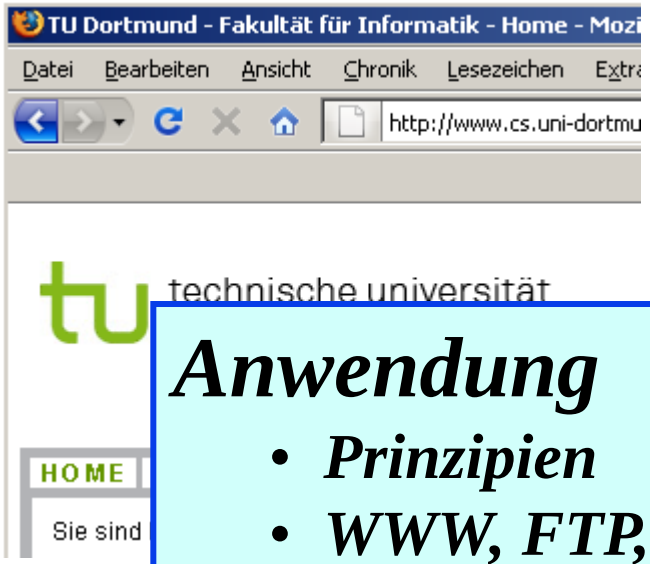
Rechnernetze und verteilte Systeme (BSRvS II)

Prof. Dr. Heiko Krumm

FB Informatik, LS IV, AG RvS
Universität Dortmund

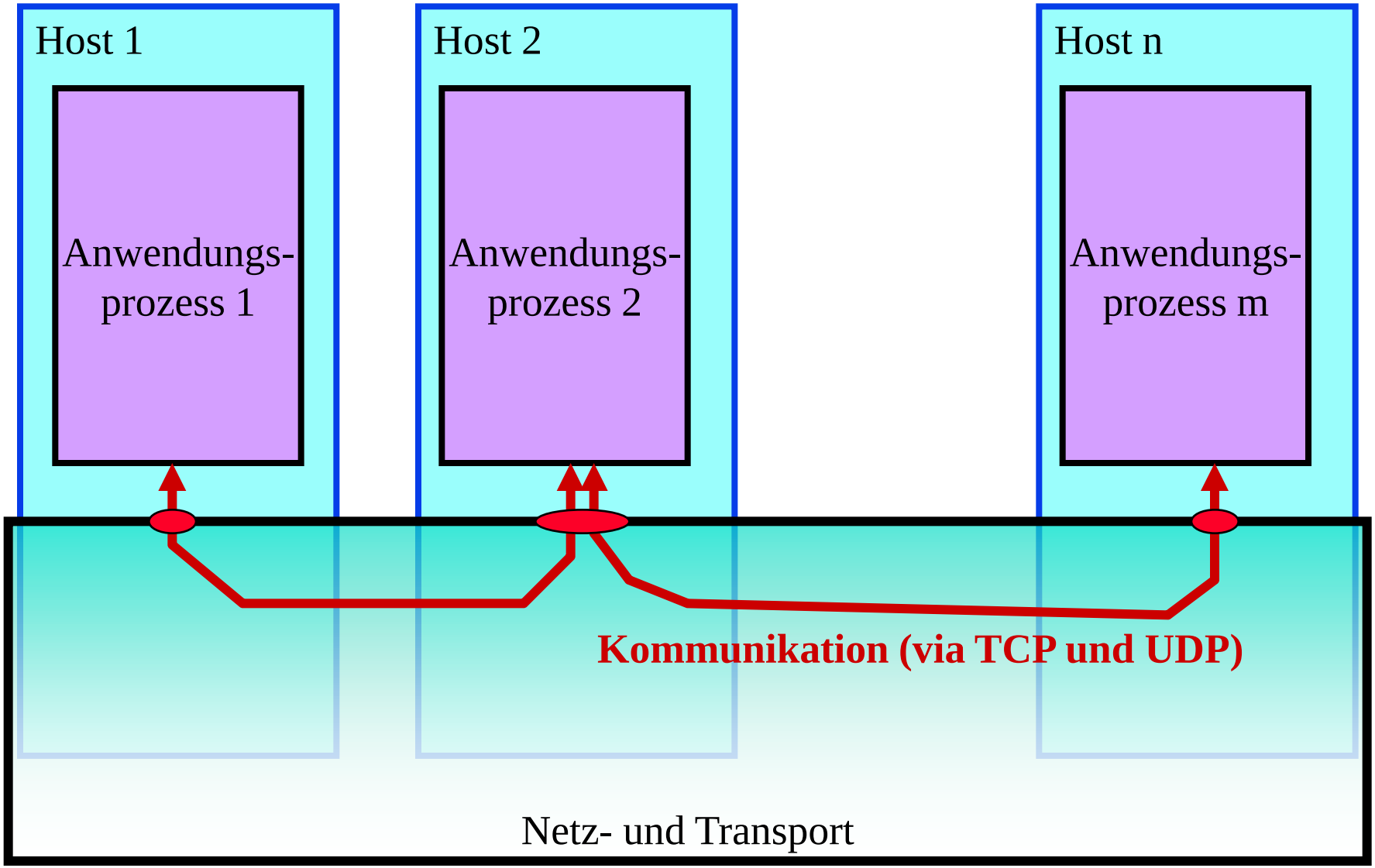


Anwendung

- ***Prinzipien***
- ***WWW, FTP, Mail***
- ***DNS***
- ***TCP-, UDP-Sockets im Client/Server-Programm***

- **Computernetze und das Internet**
- **Anwendung**
- **Transport**
- **Vermittlung**
- **Verbindung**
- **Multimedia**
- **Sicherheit**
- **Netzmanagement**
- **Middleware**
- **Verteilte Algorithmen**

Netzanwendungen: Struktur



Client/Server-Paradigma

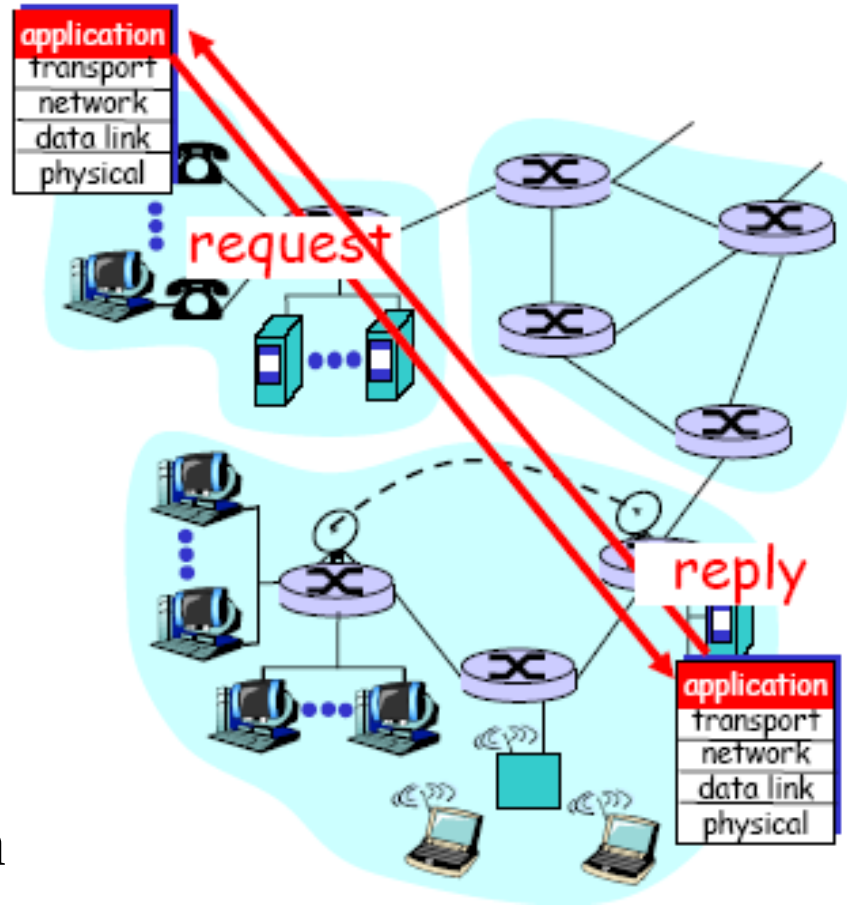
Viele Anwendungen bestehen aus zwei Teilen: **Client & Server**

Client:

- Initiiert Kontakt mit dem Server
- Fordert einen Dienst vom Server an
- Bsp. Web: Client im Browser, Email: Client im Mailsystem

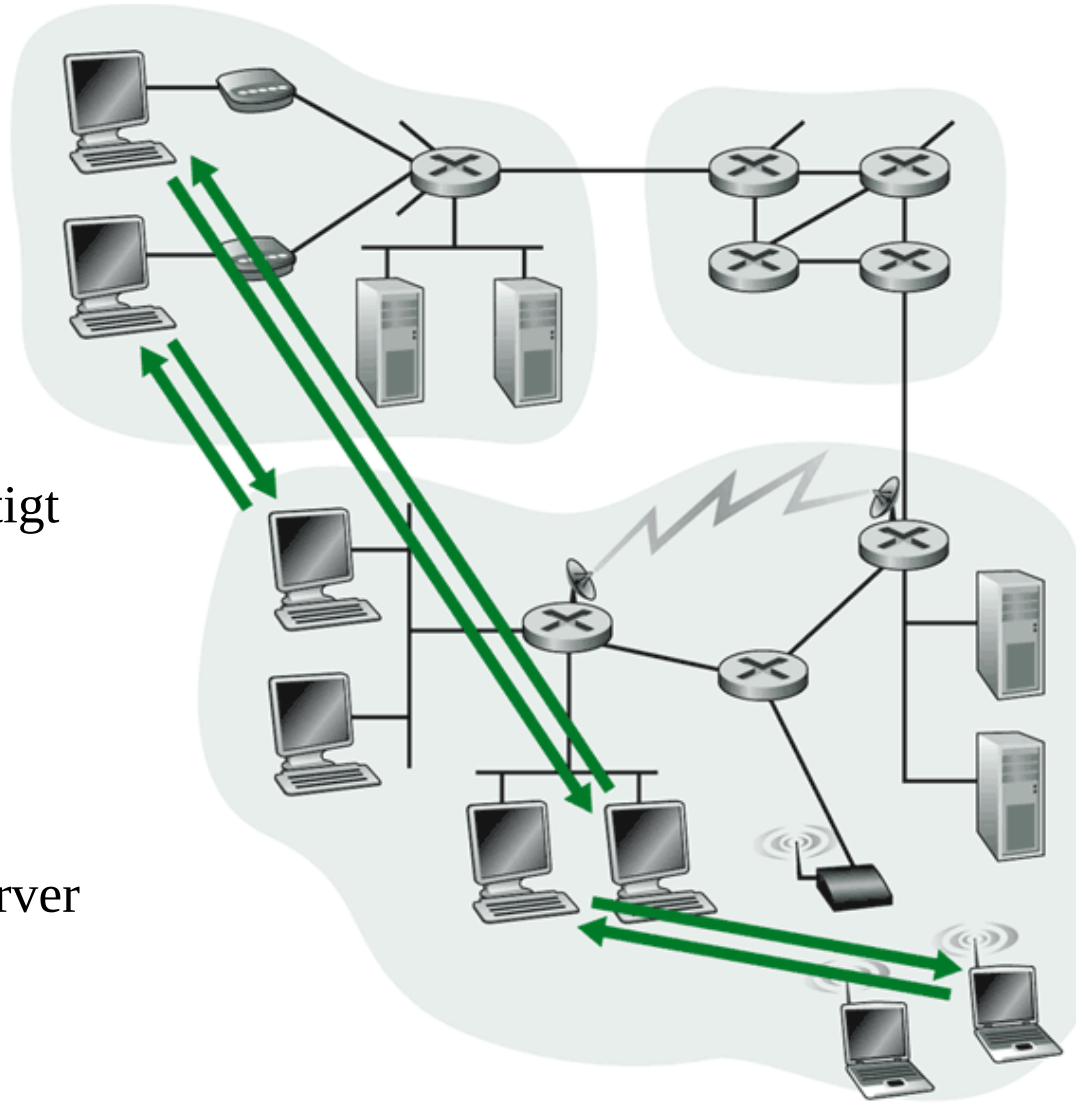
Server:

- Bietet dem Client Dienstleistungen an
- Bsp. Bietet Web-Seiten an, liefert Emails



Peer-to-Peer-Paradigma

Die Anwendungen bestehen aus gleichberechtigten Teilen: **die Peers** (teilweise auch **Agenten genannt)**



- Skalierbarkeit
  - \+ Keine zentralen Funktionen benötigt
  - \- Verwaltung von vielen Peer-Peer-Assoziationen
- Peers und Client/Server

  Peers wirken für andere Peers als Server und umgekehrt

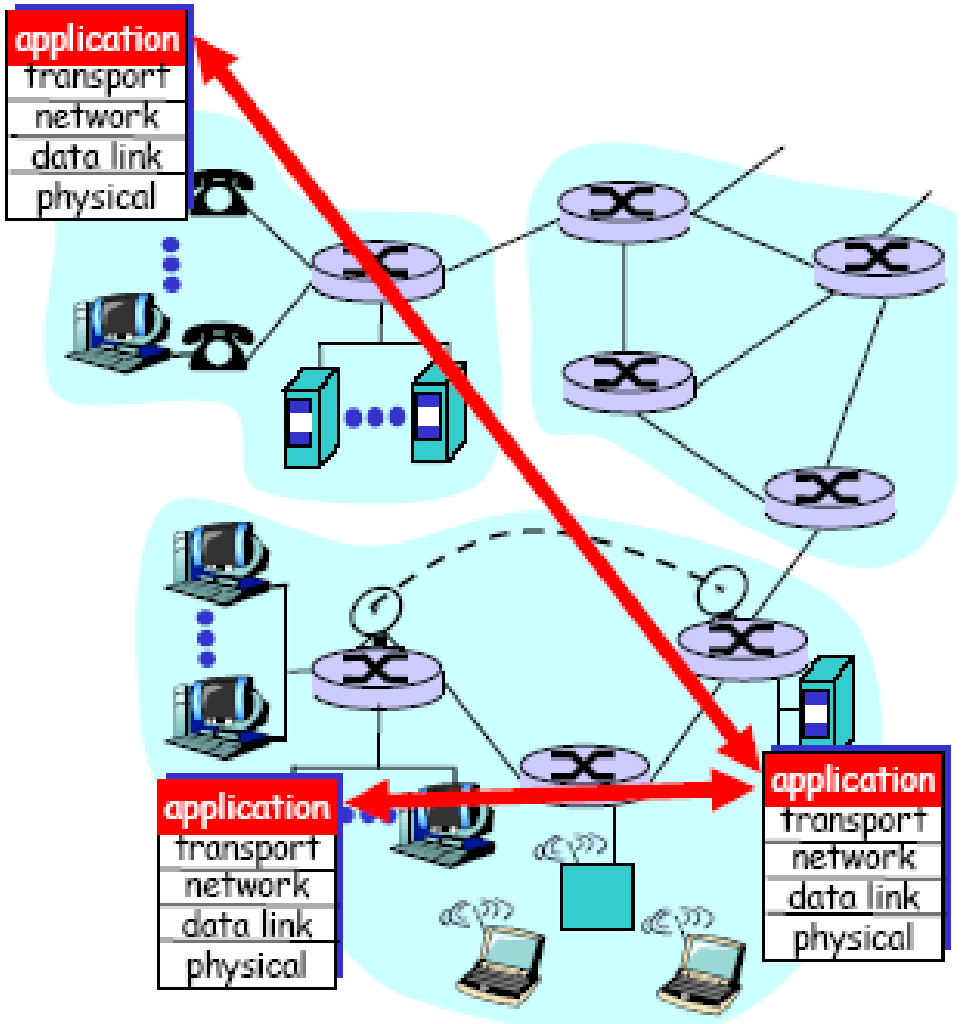
Anwendungen und Anwendungsprotokolle

Anwendung

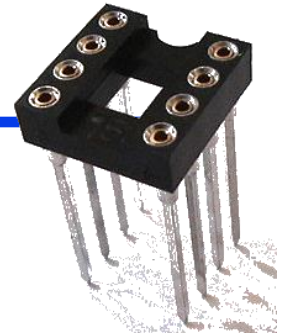
- Anwendungsprozesse werden in Endsystemen (Hosts) ausgeführt.
- Sie tauschen Anwendunsnachrichten aus.
- Beispiele:
  E-Mail, File Transfer, WWW

Anwendungsprotokolle

- Legen Syntax und Semantik der Anwendungsnachrichten fest
- Legen Ablauf des Nachrichtenaustauschs fest



Schnittstelle zum Transportsystem



- Ein Anwendungsprozess kommuniziert (sendet/empfängt) über das Netz durch eine Software-Schnittstelle ein so genanntes ***Socket***
- „Tür zum Netzwerk“
  Kommunikation durch Abgabe bzw. Abholung von Nachrichten an der Tür
- Programmierschnittstelle: ***API***
  (Application Programming Interface)
  u.U. Wahl-/Parametrierungsmöglichkeiten
  - Wahl des Transportprotokolls
  - Wahl einiger Parameter
    (z.B. Timeout, Puffergrößen)

Adressierung

- Um Nachrichten einem Prozess zuzuordnen wird eine eindeutige Kennung/Adresse benötigt
- Jeder Host hat eine eindeutige IP-Adresse (IP-V4: 32Bit)

Damit sind aber noch keine Prozesse adressierbar!!

Paar aus *IP-Adresse für den Rechner und*
Port-Nummer für die Anwendung

Echo-Server: TCP- oder UDP-Port 7

HTTP-Server: TCP-Port 80

SMTP-Mail-Server: TCP-Port 25

POP3-Mail-Server: TCP-Port 110

Anwendungsanforderungen an Transportdienst

Zuverlässigkeit beim Datentransfer

- Einige Anwendungen erfordern hohe Zuverlässigkeit (z.B. Dateitransfer)
- andere können einen gewissen Datenverlust verkraften (z.B. Audio-Übertragungen)

Zeitverhalten

- Einige Anwendungen verlangen die Einhaltung von Zeitschranken (z.B. Steuerungen, Spiele)
- andere verkraften auch längere Verzögerungen (z.B. Email, Dateitransfer)

Anwendungsanforderungen an Transportdienst

Bandbreite

- Einige Anwendungen benötigen eine gewisse minimale Bandbreite (z.B. Multimedia)
- andere (elastische) Anwendungen nutzen die Bandbreite, die verfügbar ist (z.B. Dateitransfer)

Sicherheit

- Einige Anwendungen haben hohe Sicherheitsanforderungen (z.B. E-Banking)
- andere nutzen nur allgemein einsehbare Daten
  (z.B. Fernsehen über Internet)
- Interaktion mit anderen Anwendungen
- Manche Anwendungen sollen andere nicht beeinflussen
  (z.B. Monitoring)

Internet – Transport: Dienste und Protokolle

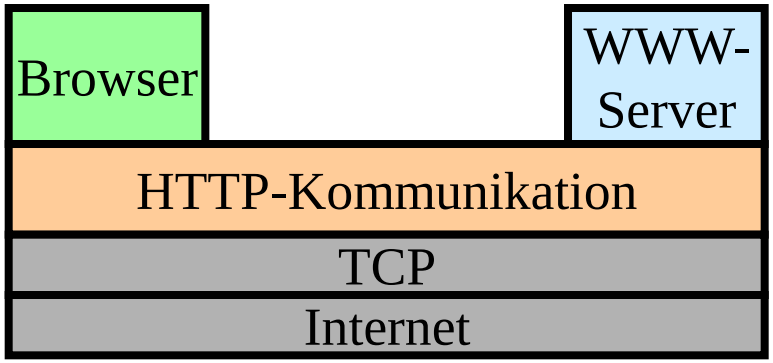
TCP

- verbindungsorientiert
- zuverlässiger Transport
- Flusskontrolle
- Überlastkontrolle
- kein Timing
- keine garantierte Bandbreite

UDP

- nicht verbindungsorientiert
- unzuverlässiger Transport
- keine Fluss- und Überlastkontrolle
- kein Timing
- keine garantierte Bandbreite

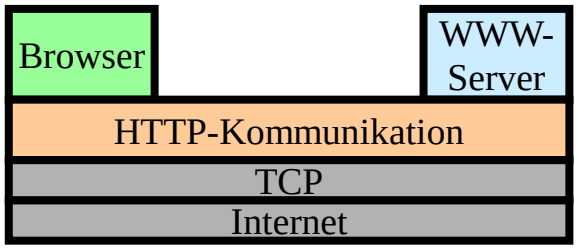
WWW: World Wide Web



HTTP: Hypertext-Transfer-Protocoll

- Client-Server
  - Client: Browser fordert WWW-Objekte an, empfängt sie und stellt sie dar
  - Server: WWW-Server sendet entsprechende Objekte
  - HTTP/1.0: RFC 1945
  - HTTP/1.1: RFC 2616
- benutzt TCP
  - Client iniziiert TCP-Verbindung zum Server
  - Server akzeptiert Verbindung
  - HTTP-Pakete werden ausgetauscht
  - TCP-Verbindung wird beendet
- zustandlos: Server speichert keine Informationen über vorangegangene Verbindungen.

WWW: HTTP



Nonpersistent-HTTP

- Maximal ein Objekt kann pro Verbindung übertragen werden.
- HTTP/1.0 benutzt Nonpersistent-HTTP.

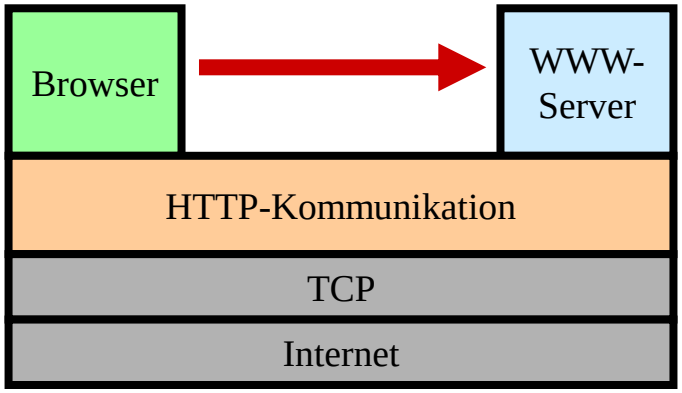
Persistent-HTTP

- Mehrere Objekte können pro Verbindung übertragen werden.
- Persistent-HTTP ohne Pipelining
  - Der Client fordert erst ein neues Objekt an, nachdem das vorangehende empfangen wurde.
- Persistent HTTP mit Pipelining
  - Der Client fordert ein neues Objekt an, sobald er auf eine Referenz stößt.
- HTTP/1.1 benutzt Persistent-HTTP mit Pipelining im Default-Modus

WWW: HTTP-PDUs

Nachrichten sind

- Request-Nachrichten oder
- Response-Nachrichten



Simple Request

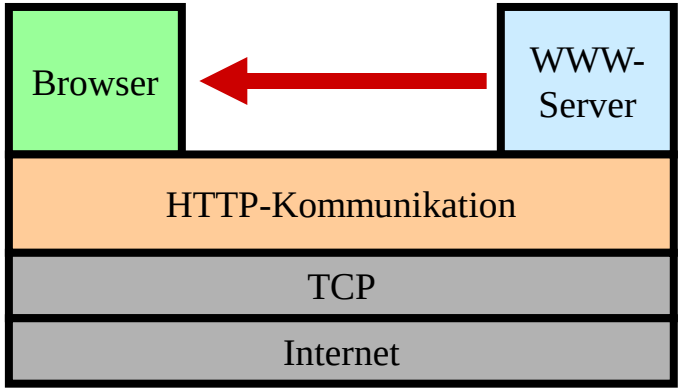
METHOD *URI* <crlf>

Full Request

METHOD *URI* HTTP/1.1 <crlf>
Header-Field-Name : value <crlf>
…
Header-F-ield-Name : value <crlf>
<crlf>
MIME-conform Body

METHOD
z.B. GET, POST, ..

WWW: HTTP-PDUs



Response

HTTP/1.1 *Status-Code Reason-Line* <crlf>
Header-Field-Name : value <crlf>
Header-Field-Name : value <crlf>
<crlf>
MIME-conform Body

Response-Status-Codes

200	OK	404	Not Found
301	Moved Permanently	505	HTTP Version Not Supported
400	Bad request		

WWW: HTTP - Operationen

GET-Methode: Fordert das Objekt mit gegebener URL an

ls4com2> telnet ls4-www.cs.uni-dortmund.de 80

Trying 129.217.16.36...

Connected to willi.

Escape character with´^]´.

GET http://www4.cs.uni-dortmund.de/Lehre/07-40303.html

<!DOCTYPE HTML PUBLIC „-//IETF//DTD HTML 2.0/EN“>

<html>

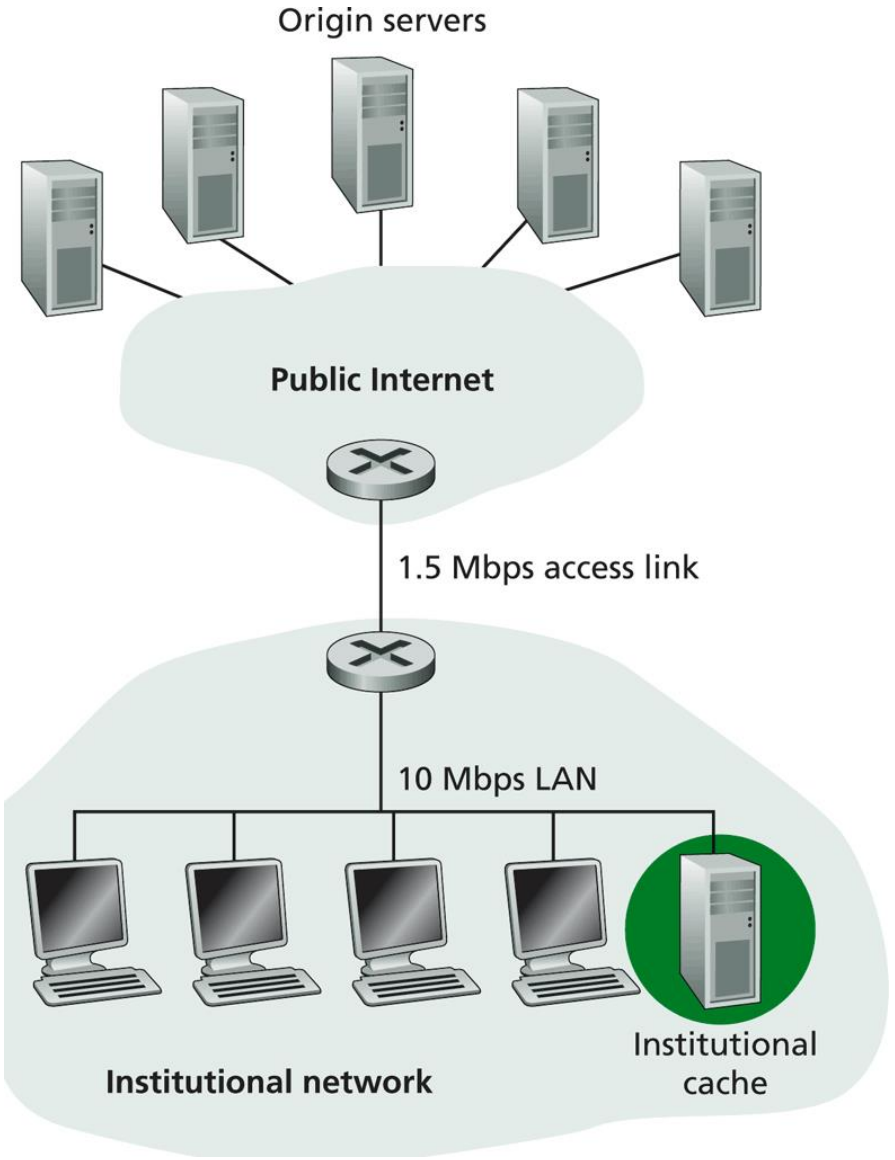
....

</html>

WWW: Caching

Laden von Seiten kann aufwändig/langwierig sein

- Caching im Browser
- Caching im Proxy-Server

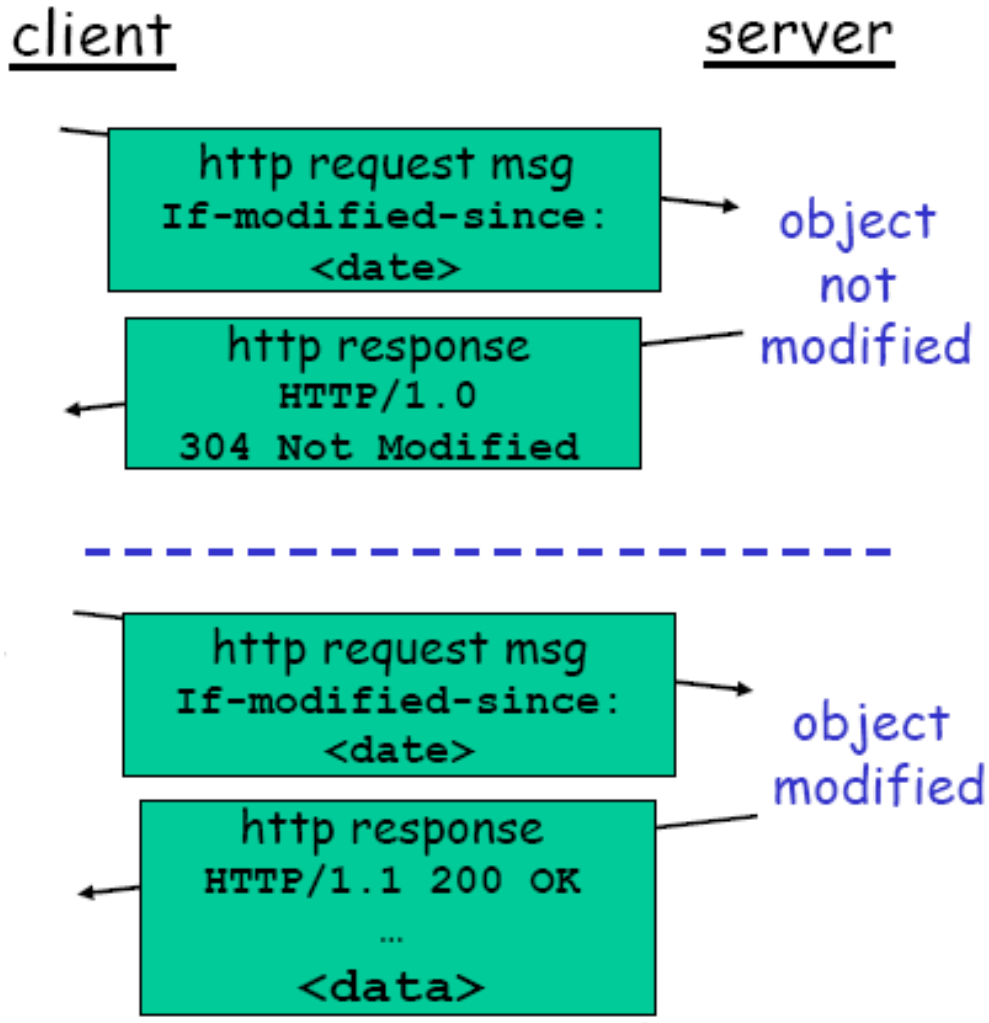


WWW: Conditional GET

Ziel:

Sende das angeforderte Objekt nicht, wenn der Client bereits eine aktuelle Version im Cache hat

- Client (u.U. Proxy) sendet das Datum seiner Kopie mit (if-modified-since)
- Server schickt das angeforderte Objekt nur, wenn eine neue Version vorhanden



WWW: Authentifikation und Autorisierung

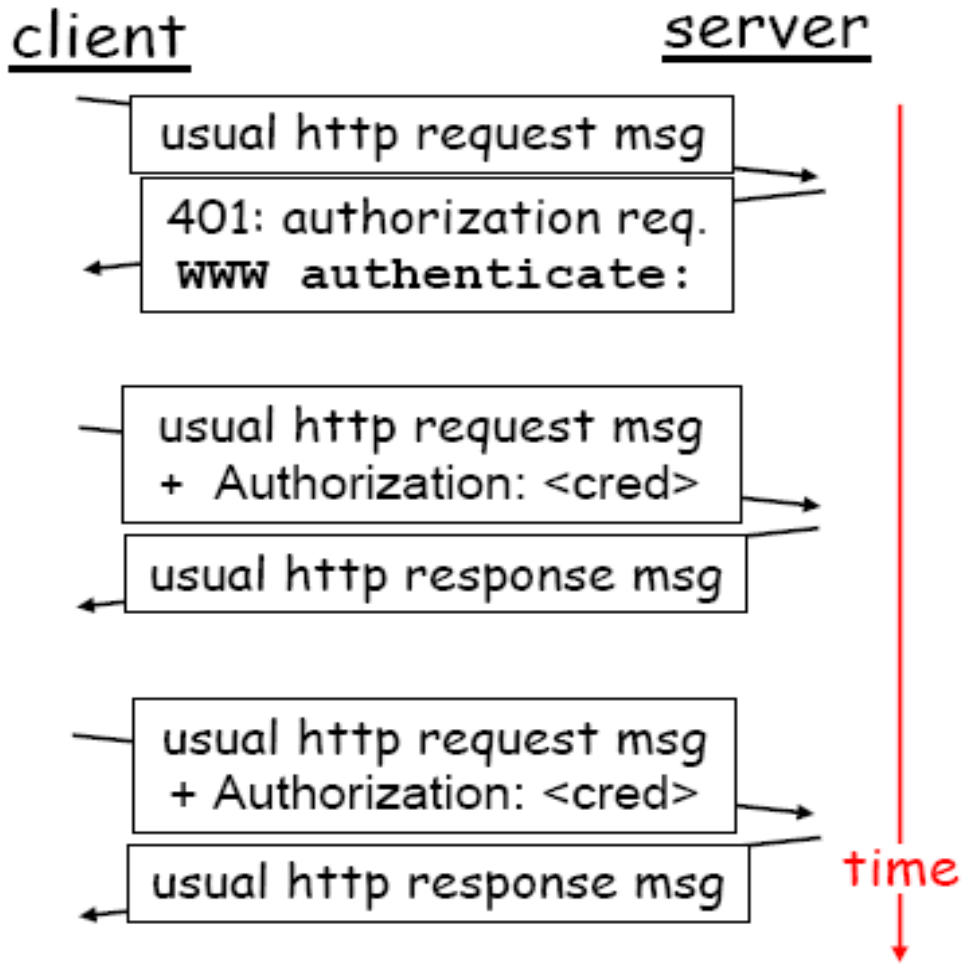
Authentifikation

Client gibt Id an und Beweis, dafür, dass er dies auch ist.

Autorisierung

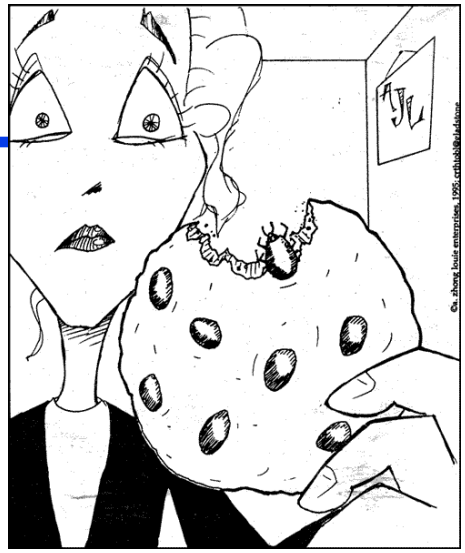
Zugriff für Client nur im Rahmen der ihm zugewiesenen Privilegien

- Üblicher Ansatz:
  Benutzername, Password
  oder IP-Adresse
- Zustandslos: Client muss Autorisierung in jeder Anfrage präsentieren
  - Autorisierung im Header
  - Ohne Autorisierung keine Zugriff



Vorsicht: Benutzername und Password im Browser-Cache!

WWW: Zustandsspeicherung mit Cookies



Viele Web-Seiten nutzen ***Cookies***

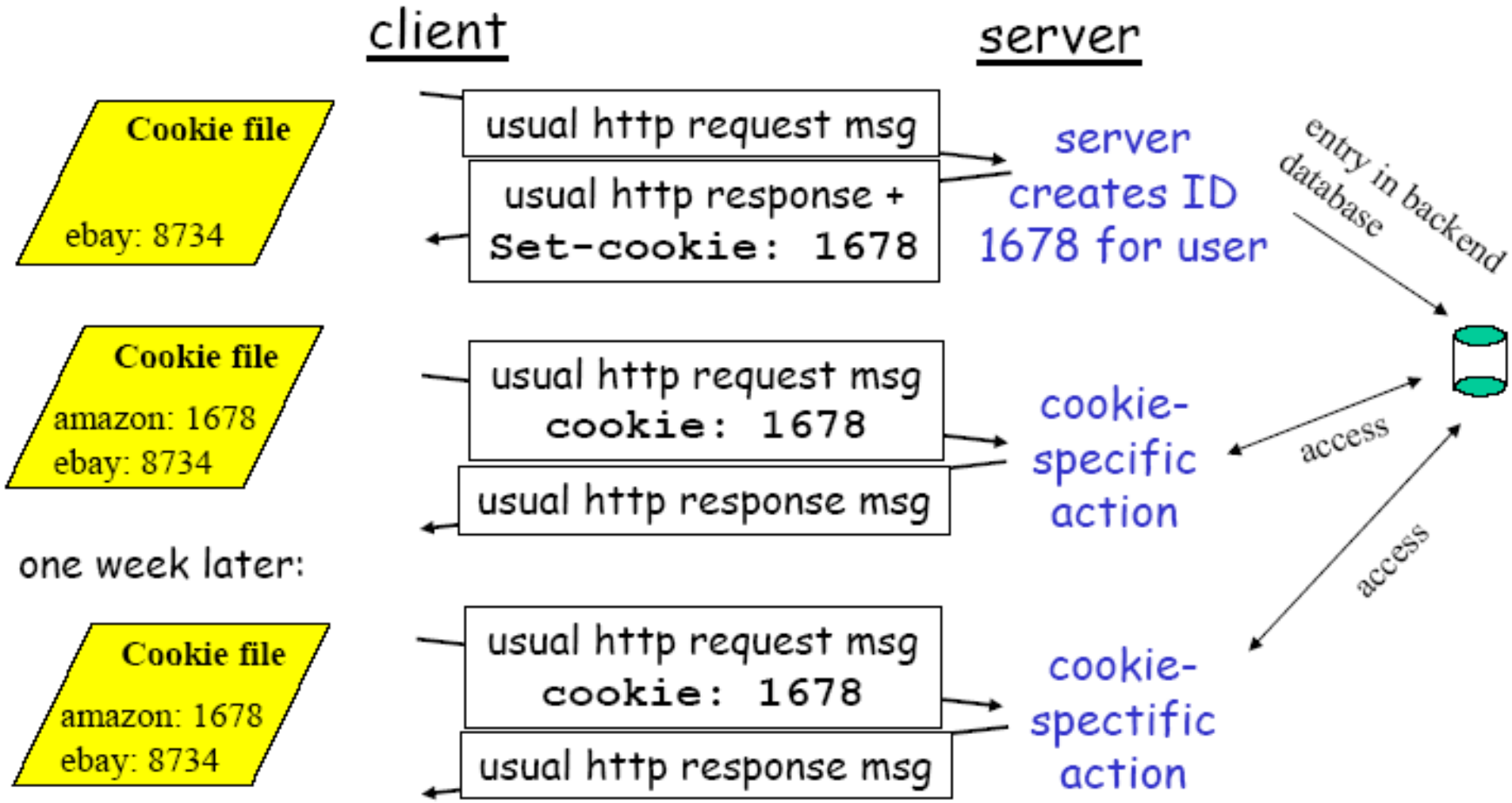
Basiskomponenten

1. Cookie-Zeile im Header der HTTP-Response Nachricht
2. Cookie-Zeile in der HTTP-Request Nachricht
3. Cookie-Datei auf dem Benutzerrechner, vom Browser verwaltet
4. Datenbank mit Cookies auf Server-Seite

Beispiel:

- Susan nutzt das Internet von ihrem PC
- Sie ruft eine e-Commerce Seite erstmals auf
- Es wird ein Cookie mit einer eindeutigen ID generiert und auf Susans Rechner und in der Datenbank gespeichert

WWW: Zustandsspeicherung mit Cookies



WWW: Zustandsspeicherung mit Cookies

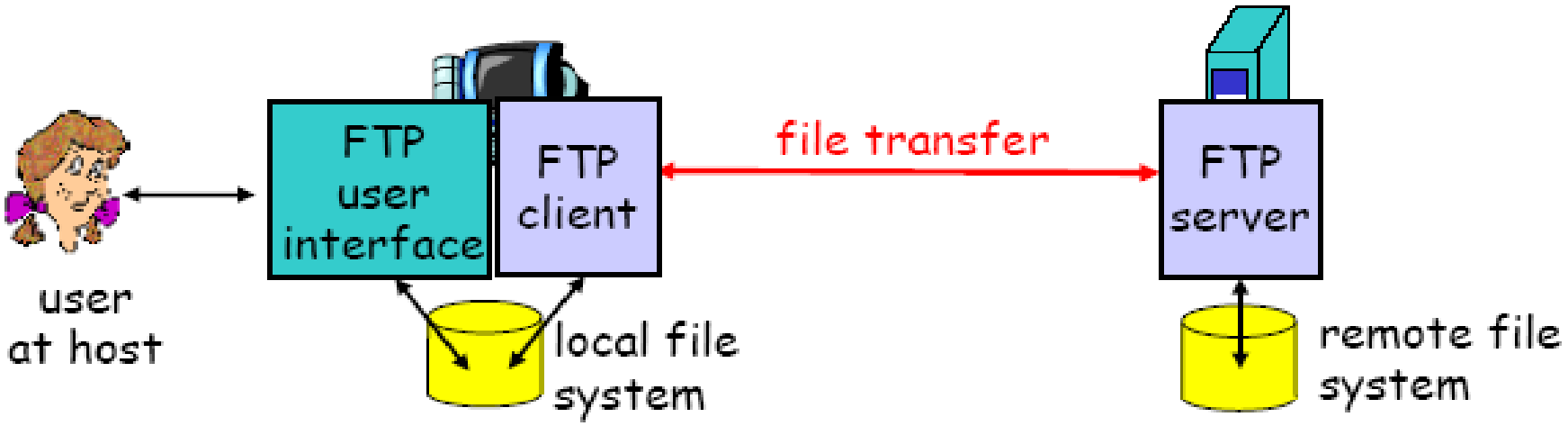
Vorteile der Cookies

- Autorisierung
- Einkaufskarten (Speichern von Käufen, Bezahlung, …)
- Empfehlungen
- Zustand einer Benutzersitzung

Die andere Seite der Cookies

- Cookies übermitteln Informationen
- Bei jedem neuen Zugriff kann die vorhandene Information ergänzt werden
- Suchmaschinen nutzen Cookies um über Nutzer zu lernen
- Informationen können an Dritte weitergegeben werden

FTP: Internet File Transfer Protocol

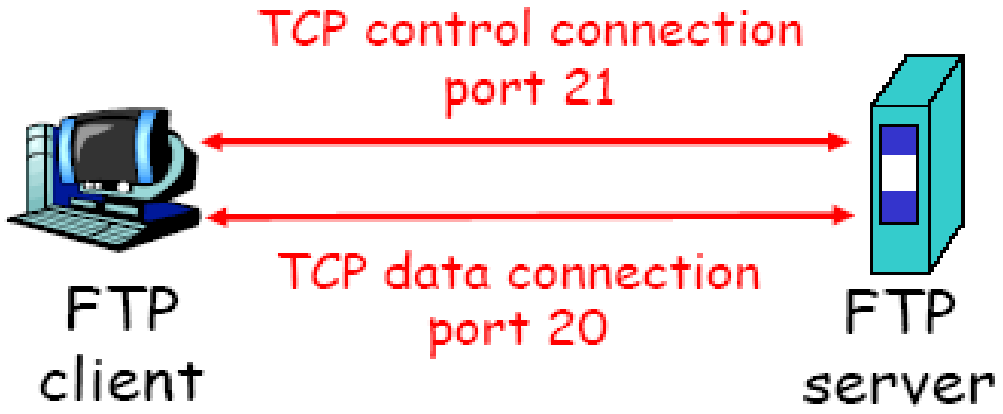


Aufgabe: Übertragung von Dateien von / zu einem entfernten Rechner

- Client/Server Modell
  - Client auf Nutzerseite initiiert die Übertragung
  - Server auf dem Host führt Übertragung aus
- RFC 959 und Port 21 auf dem Server

FTP: Separate Kontroll- und Datenverbindungen

- FTP Client ruft FTP Server über Port 21 auf und spezifiziert TCP als Transportprotokoll
- Client autorisiert sich über die Kontrollverbindung
- Client kann sich die Verzeichnisse des Servers mittels der Kontrollkommandos ansehen
- Dateitransfer führt zur Öffnung einer TCP-Datenverbindung
- Nach dem Transfer schließt der Server die Datenverbindung



- Server öffnet eine neue Datenverbindung zur Übertragung der nächsten Datei
- Server speichert den Zustand des Clients (Verzeichnis, Authentifizierung)
- FTP sendet Kontrollinformation separat (*out of band*) im Gegensatz zu HTTP (*in band*)

FTP: Kommandos und Antworten

Einige Kommandos

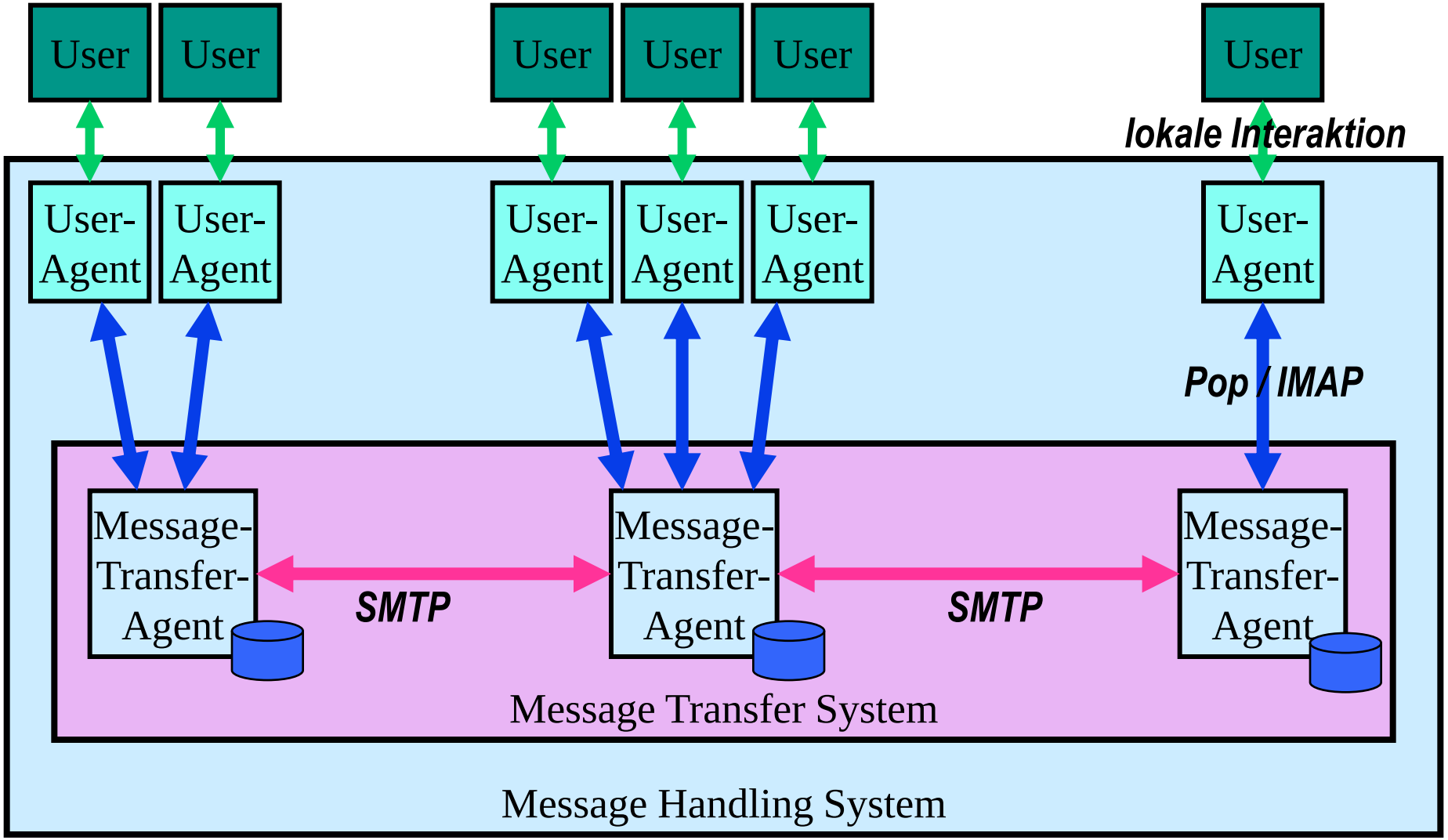
- Nach dem Aufru
  ftp <Hostname>
  muss das Password eingegeben werden
- **binary** setzt den Übertragungsmodus auf binär (statt ASCII)
- **ls** listet das entfernte Verzeichnis
- **put** überträgt eine Datei zum entfernten Rechner
- **get** überträgt eine Datei vom entfernten Rechner

Einige Rückgabewerte

- Jedes Kommando wird vom Server per Reply beantwortet
- Verwendete Statuscodes:
  - **331** Username ok, password required
  - **125** Data connection already open; transfer starting
  - **425** can‘t open data connection
  - **452** error writing file

FTP ist „altes“ Protokoll und hat schwerwiegende Sicherheitsmängel
(z.B. unverschlüsselte Passwortübertragung, unverschlüsselte Dateiübertragung)

E-MAIL


User
User
User
User
User
User
lokale Interaktion
User-
Agent
User-
Agent
User-
Agent
User-
Agent
User-
Agent
User-
Agent
Pop / IMAP
Message-
Transfer-
Agent
Message-
Transfer-
Agent
Message-
Transfer-
Agent
SMTP
SMTP
Message Transfer System
Message Handling System

E-MAIL: Protokolle

SMTP: Simple Mail Transfer Protocol [RFC 2821]

- Benutzt TCP, um Emails vom Sender (Mail Server des Senders) zum Empfänger (Mail Server des Empfängers) zu transferieren
- 3 Phasen
  - Handshake
  - Nachrichtentransfer
  - Beenden der Verbindung
- 7-bit ASCII-Format zeilenweise organisiert

Empfang von Emails

Mail-Server speichert empfangene Emails

User-Agenten greifen auf die Emails zu, durch

- POP: Post Office Protocol
  Autorisierung und Download
- IMAP: Internet Mail Access Protocol
  komplexer und variantenreicher

E-MAIL: Mail-Format

„Plain“: ASCII-Zeilen

Umschlag (Envelope)
FROM:
TO:
SUBJECT:
Leerzeile

Inhalt (Body)
Textzeile
Textzeile
•

```
Subject: is this your picture?
From: "Adele Krause"
    <Buckquillwort00259245958@yahoo.com>
Date: Thu, 09 Sep 2004 11:18:39 -0100
To: bause@ls4.cs.uni-dortmund.de,
    herrmann@ls4.cs.uni-dortmund.de,
    kemper@ls4.cs.uni-dortmund.de,
    krumm@ls4.cs.uni-dortmund.de,
    lengewitz@ls4.cs.uni-dortmund.de,
    lindemann@ls4.cs.uni-dortmund.de,
    luebeck@ls4.cs.uni-dortmund.de

I think this is your picture..
.
```

E-MAIL: MIME – Multimedia Mail Extensions

Multimedia-Daten oder auch kodierte Dateien können nicht direkt per Email verschickt werden, da sie nicht ASCII-kodiert sind

Lösung MIME-Format

- Beim Sender
  - Daten in ASCII kodieren
  - MIME Header hinzufügen, in welchem das Dateiformat angegeben ist (z.B. jpeg, word, gif, mpeg, plain)
- Beim Empfänger
  - Interpretation des MIME-headers
  - Dekodierung des ASCII-Textes
  - Aufruf des zugehörigen Programms zum Anzeigen der Daten

E-MAIL: *MIME – Multimedia Mail Extensions*

```
From: Frank Thorsten Breuer <breuer@ls3.cs.uni-dortmund.de>
MIME-Version: 1.0
To: breuer@ls3.cs.uni-dortmund.de
Subject: MIME-Demo
Content-Type: multipart/mixed;
boundary="------------E59DCA0902797826211CDB5F"
Dies ist eine mehrteilige Nachricht im MIME-Format.
--------------E59DCA0902797826211CDB5F
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Dies ist eine Multipart-Email. Der Text-Teil hat eine Zeile.
--------------E59DCA0902797826211CDB5F
Content-Type: image/jpeg;
name="inpud.jpg"
Content-Transfer-Encoding: base64
Content-Disposition: inline;
filename="inpud.jpg"
/9j/4AAQSkZJRgABAQEAXwBfAAD//gAcU29mdHdhcmU6IE1pY3Jvc29mdCBPZmZpY2X/2wBD
6zX4R/UvpID0Ai8/+s1+Ef1L6Ses1+Ef1L6SA9AIvP8A6zX4R/UvpJ6zX4R/UvpID//Z
--------------E59DCA0902797826211CDB5F--
```

DNS: *Domain Name System*

„ls4-www.cs.uni-dortmund.de“ → [129,217,16.36]

Menschen werden durch mehrere Identifikatoren identifizierbar:

- SSN, Name, Pass, …

Internet Hosts oder Router:

- IP-Adresse (32 Bit) – Teil jedes Datagramms
- Name gaia.cs.umass.edu wird von Menschen genutzt

⇒ Abbildung Name ↔ IP-Adresse nötig

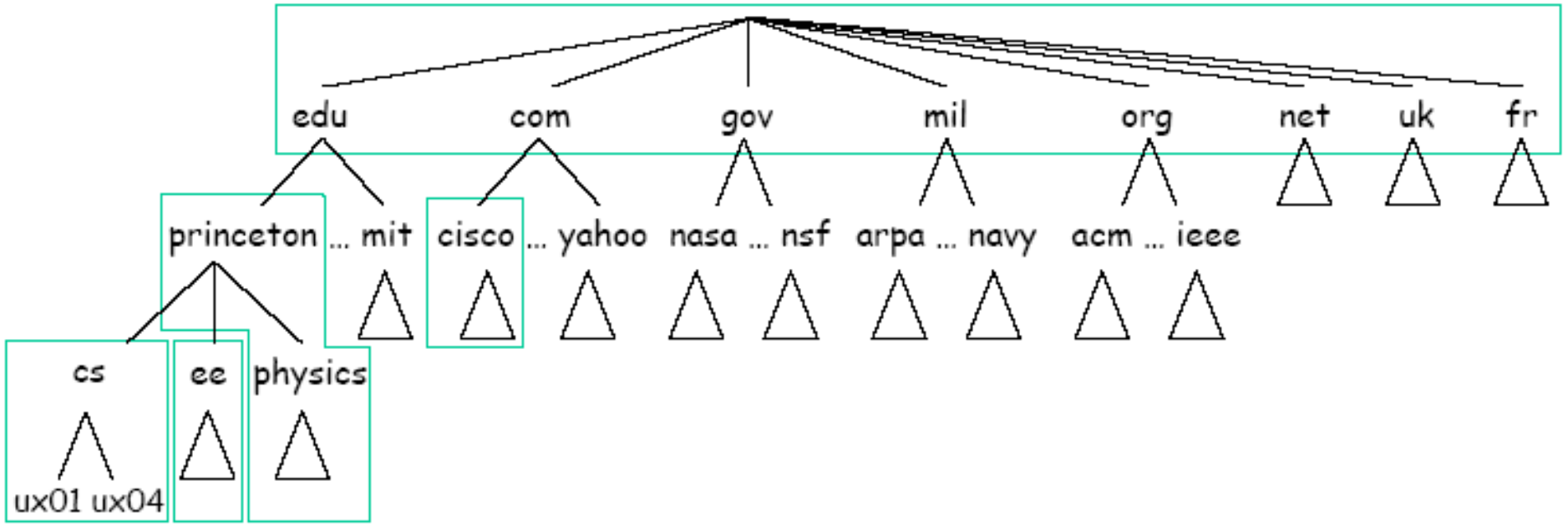
Domain Name System

- Verteilte Datenbank implementiert eine Hierarchie von Name-Servern
- Protokoll der Anwendungsschicht zur Übersetzung Name ↔ IP Adresse
  - Zentraler Dienst des Internets auf der Anwendungsebene realisiert

DNS: *Domänen-Hierarchie*

„ls4-www.cs.uni-dortmund.de“ → [129.217.16.36]

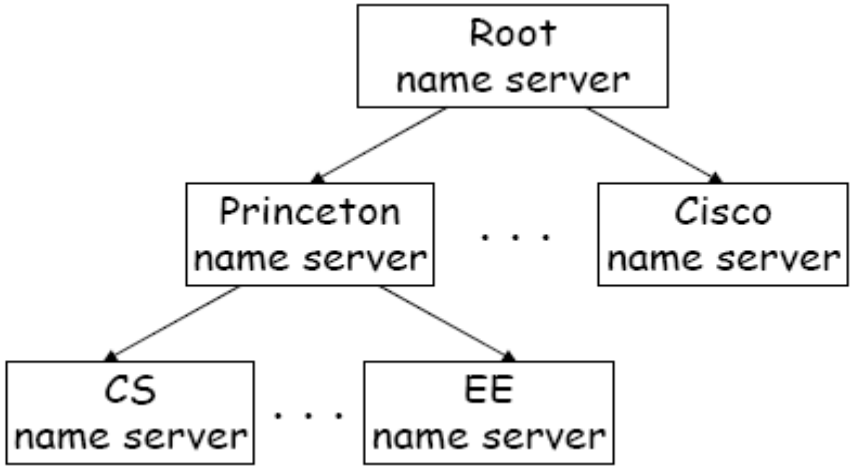
- DNS Namen werden von rechts nach links abgearbeitet
- Namenshierarchie kann als Baum visualisiert werden
  - Blätter sind Rechnernamen
  - Innere Knoten gehören zu Domains



DNS: *Nameserver-Hierarchie*

Warum kein zentrales DNS?

- Zentraler Ausfallpunkt
- Großes Verkehrsaufkommen
- Weit entfernte Datenbank
- Schlechte Wartbarkeit



Im DNS hat kein Server alle Abbildungen von Namen auf IP-Adressen

Lokale Name-Server

- Gehören zu einem ISP
  (local (default) name server)
- Jede Anfrage wird erst zum lokalen Server geleitet

Root Name-Server

- Einige wenige meist in den USA, empfangen Anfragen von lokalen Name-Servern

Autoritative Name-Server

- Jeder Host gehört zu einem autoritativen Name-Server, der seine Adresse verbindlich speichert
- Antwortet auf Anfragen bzgl. zugehöriger Hosts

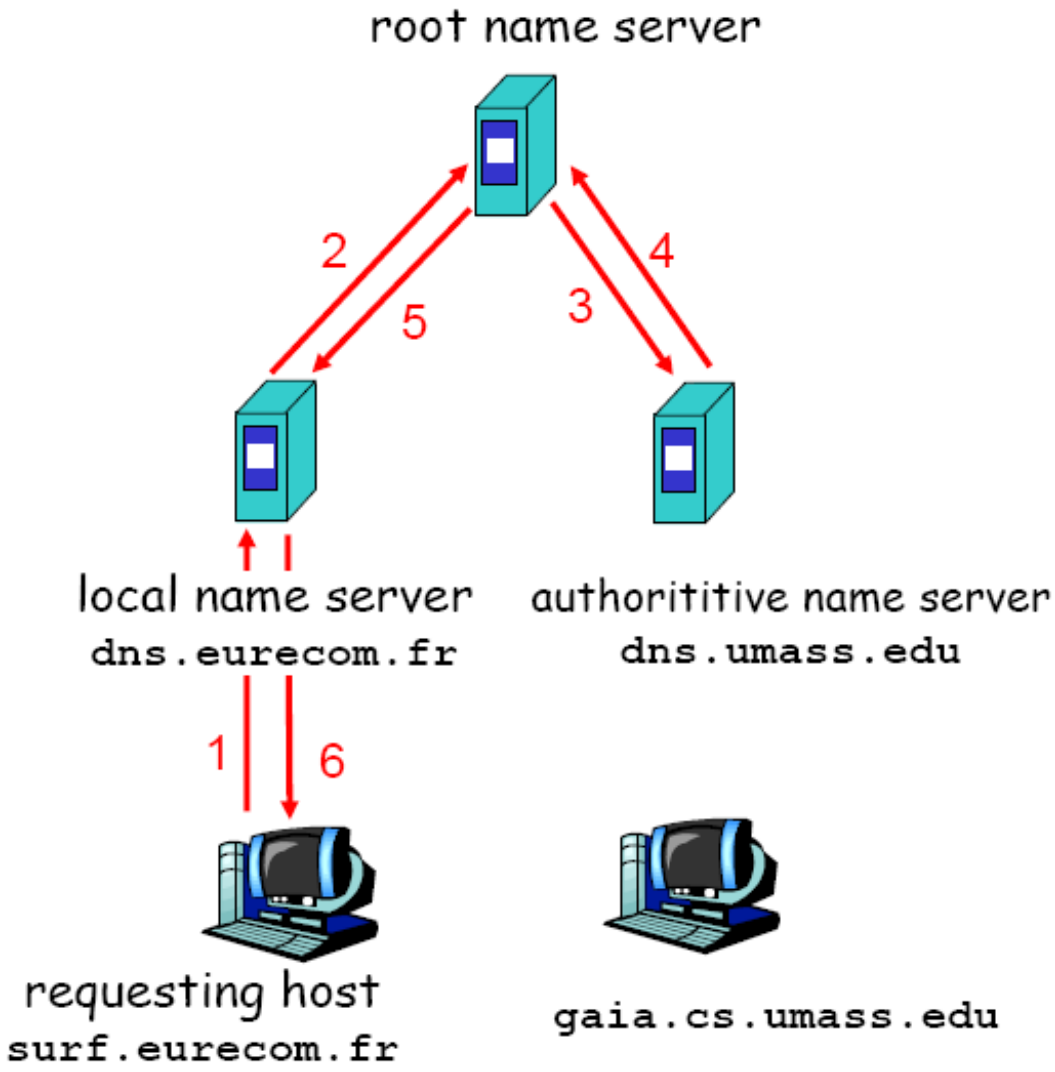
DNS: *Root Name Server*

- Werden von lokalen Name-Servern kontaktiert,
  wenn diese eine Adresse/einen Namen nicht auflösen können
- Root Name-Server (13 weltweit)
  - fragen den zugehörigen autorativen Name-Server, falls sie die Adresse nicht kennen
  - erhalten von diesem die Abbildung und leiten sie an den lokalen Name-Server weiter

DNS: Einfaches Beispiel

Ablauf einer Anfrage von **surf.eurecom.fr** nach der Adresse von **gaia.cs.umass.edu**

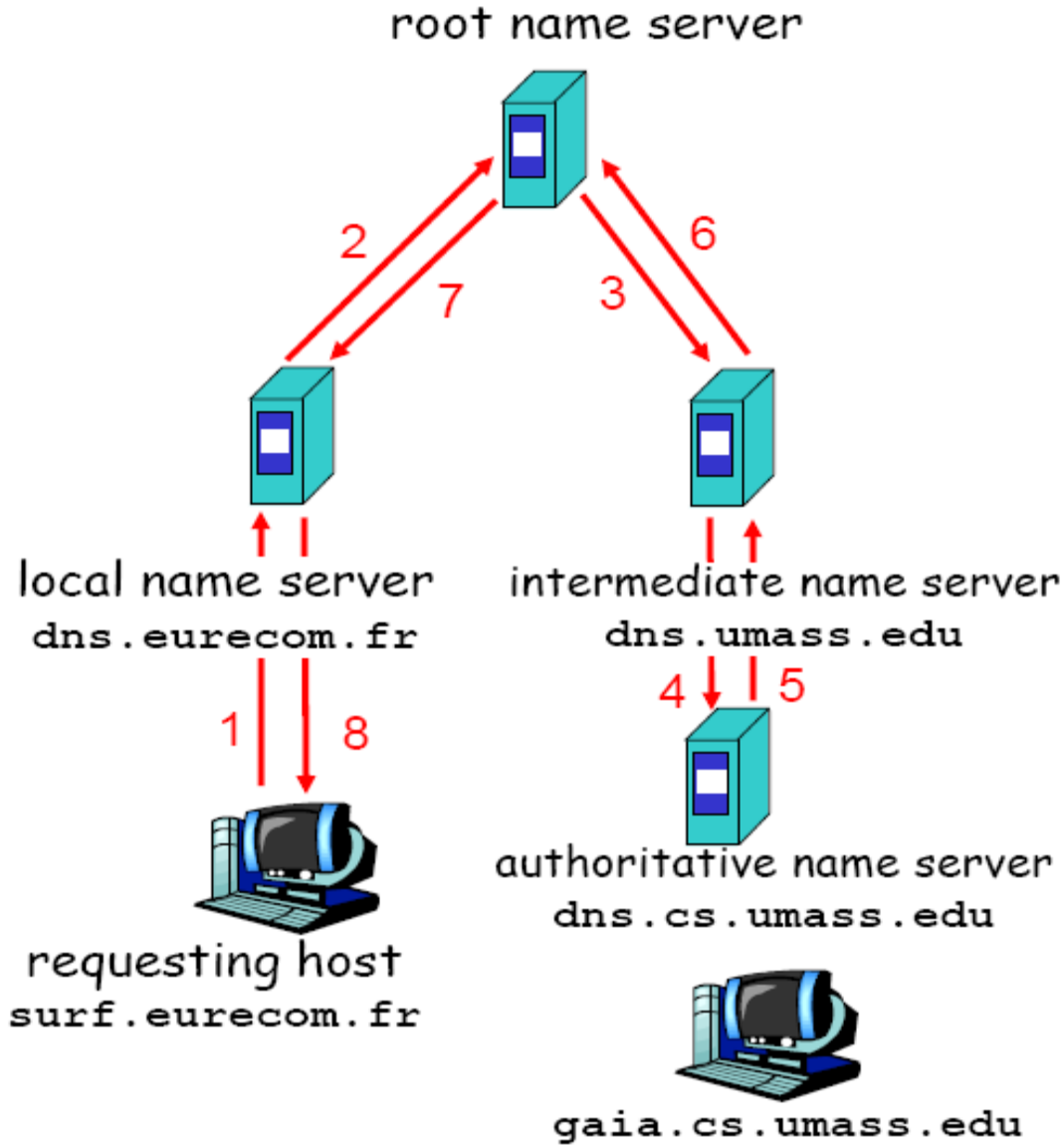
1. Nachfrage beim lokalen Name-Server dns.eurecom.fr
2. dns.eurecom.fr kontaktiert den Root-Name-Server (falls notwendig)
3. Root-Name-Server kontaktiert den autoritativen Name-Server (falls notwendig)
4. Antwort an Root
5. Antwort an eurecom
6. Antwort an surf



DNS: Einfaches Beispiel - Variante

Root Name-Server

- kennt u.U. den richtigen autoritativen Name-Server nicht
- fragt dann bei einem intermediate Name-Server, um den autorativen Name-Server zu finden



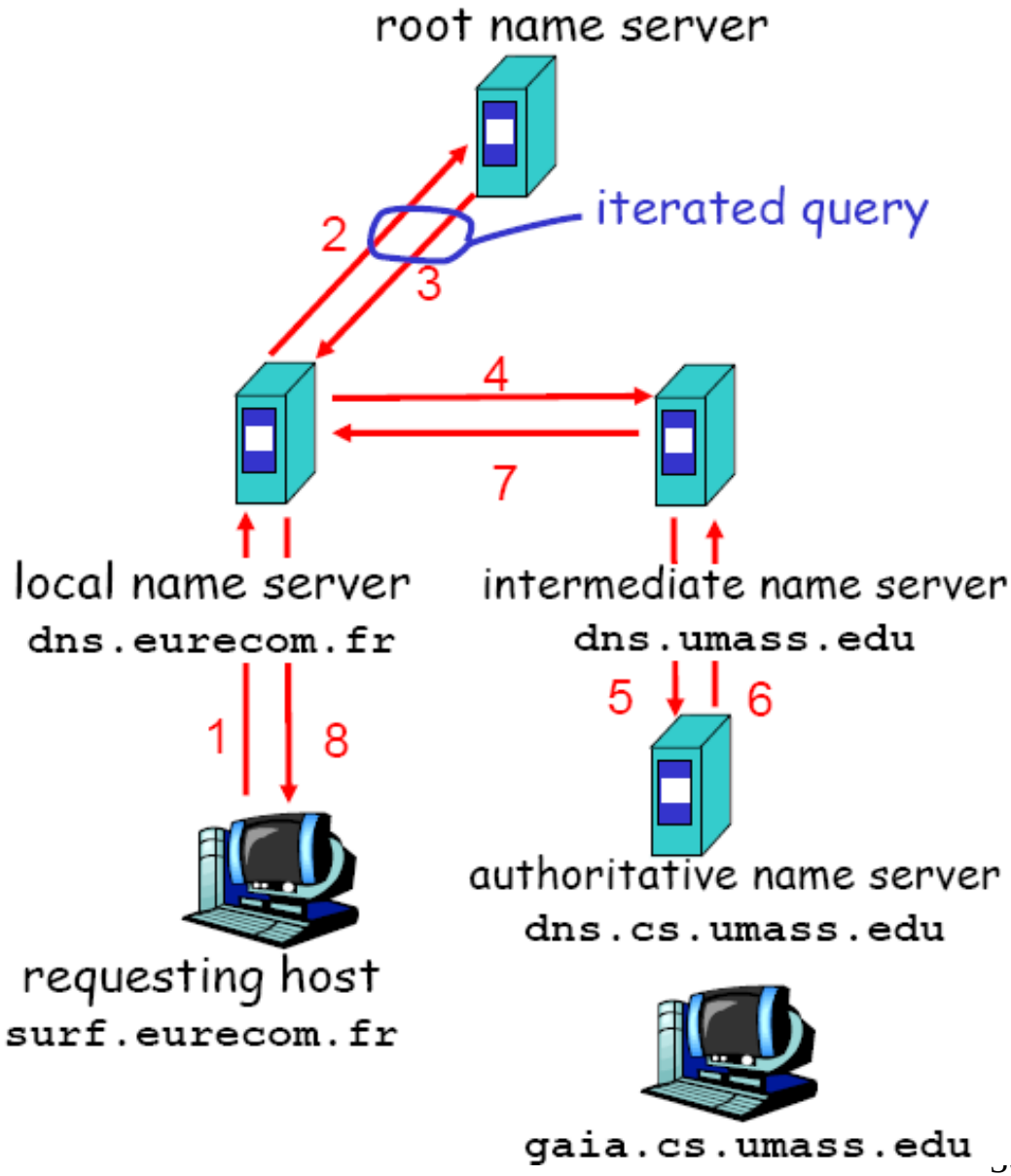
DNS: Anfrage-Ketten

Rekursive Anfrage

- Angefragte Name-Server ist für die Auflösung der Adresse verantwortlich
- Dies kann zu zahlreichen weiteren Anfragen und hoher Last führen

Iterative Anfrage

- Angefragter Server antwortet mit der Adresse des nächsten Name-Servers
- Fragender Server kontaktiert direkt diesen Server



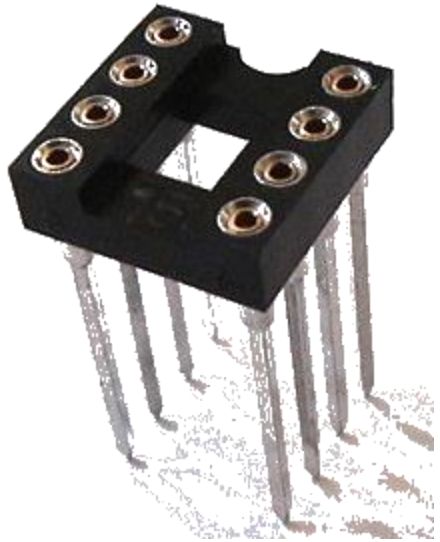
DNS: Caching und Aktualisierung

- Nachdem ein Name-Server eine Abbildung
  Name ↔ IP-Adresse erhalten hat, speichert er diese im Cache
- Einträge im Cache, die nicht nachgefragt wurden, werden nach einer Zeitspanne wieder gelöscht
- Struktur der Einträge (Name, Wert, Typ, TTL)
  - Typ A Name IP-Adresse beschreiben die Abbildung,
    Typ NS Name beschreibt eine Domain und IP-Adresse die Adresse des autorativen Servers
    Typ CNAME Name ist ein Alias-Name und Wert der vollständige Name
    Typ MX Name ist ein Alias-Name für einen Mail-Server und Wert ist der vollständige Name
  - TTL (time to live)

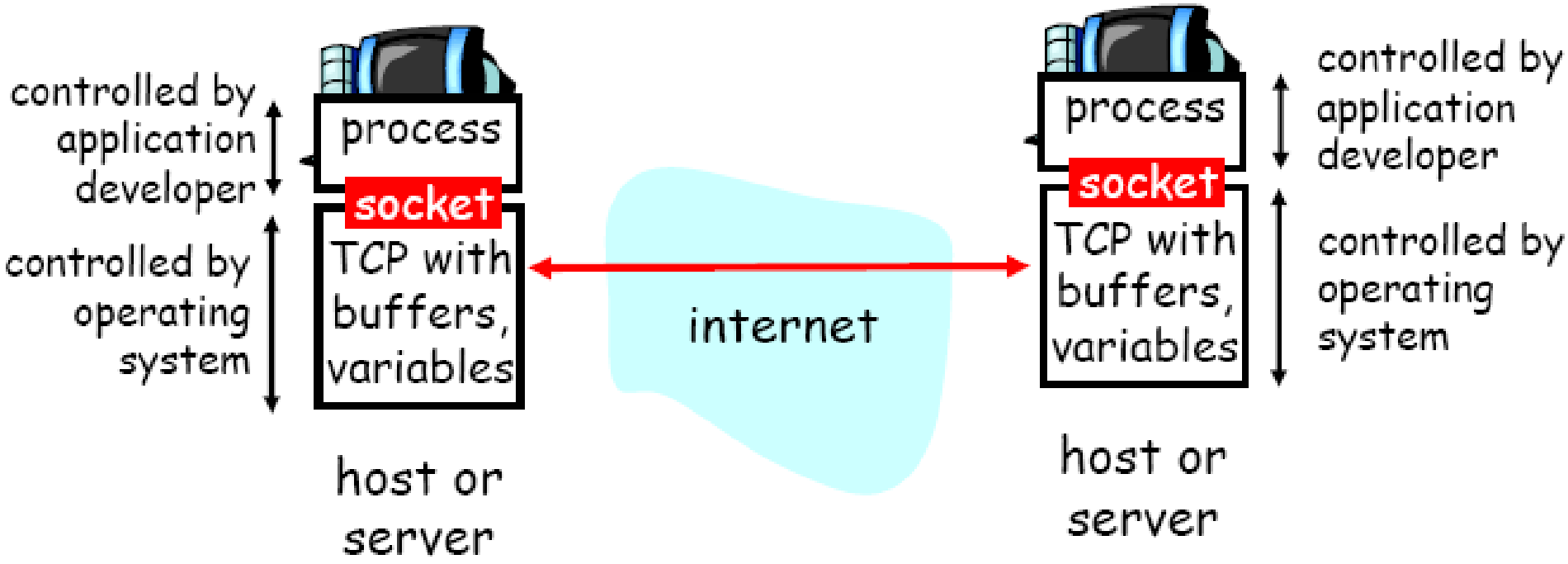
Socket-Programmierung: Kommunikations-API

Socket API

- Eingeführt für BSD4.1 UNIX, 1981
- Sockets werden von Anwendungsprogrammen erzeugt, genutzt und abschließend freigegeben
- API: erlaubt die Wahl eines Transportprotokolls und das Setzen einiger Parameter, i.e.
  - UDP: Unzuverlässige Datagramme
  - TCP: Zuverlässige Byteströme

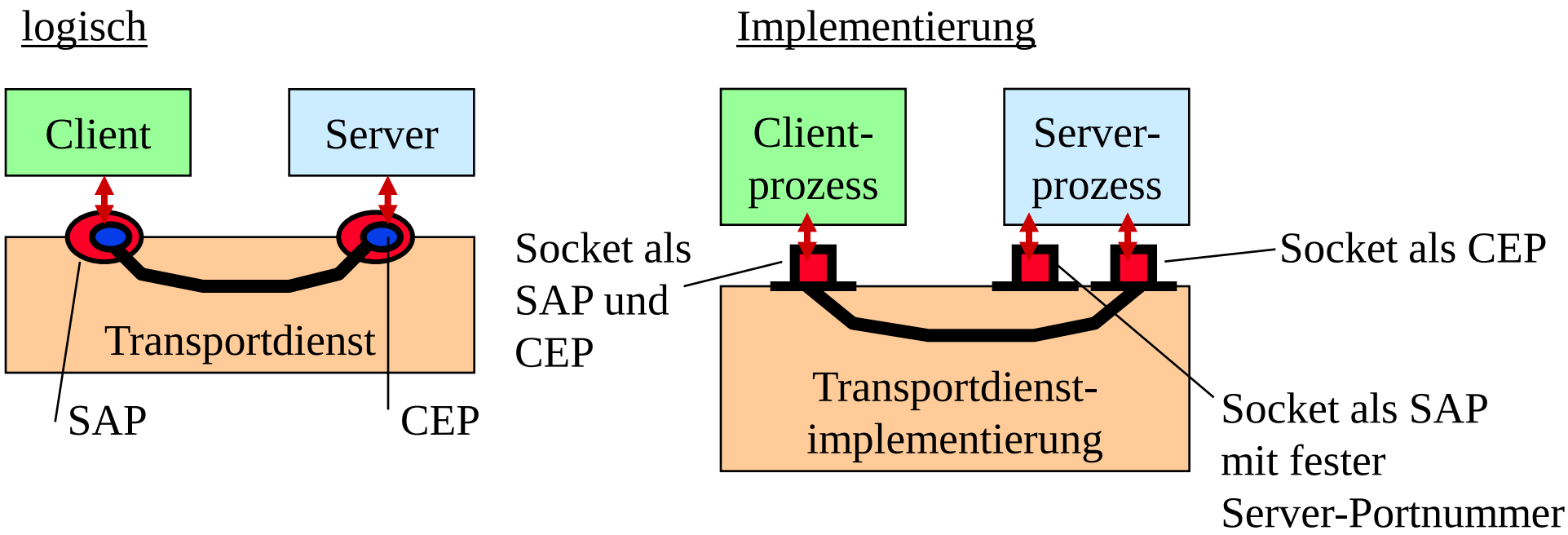


Sockets



Socket:	Implementierungsobjekt
Socket-Operationen:	Verwaltung, Ein- und Ausgabe zum Netz

Sockets



SAP (Service Access Point) und CEP (Connection Endpoint):
Elemente der logischen Architektur

Sockets: Elemente der Implementierung, Programmschnittstellen-Elemente zum Kommunikationssystem, implementieren SAPs und CEPs;
Netz-E/A = Socket-E/A ähnlich Datei-E/A
Socket hat Filedeskriptornummer (Socket-ID ist FD)

TCP-Sockets: Client und Server

Client initiiert den Verbindungsaufbau

- Server-Prozess muss dazu laufen
- Server muss einen Socket generiert haben

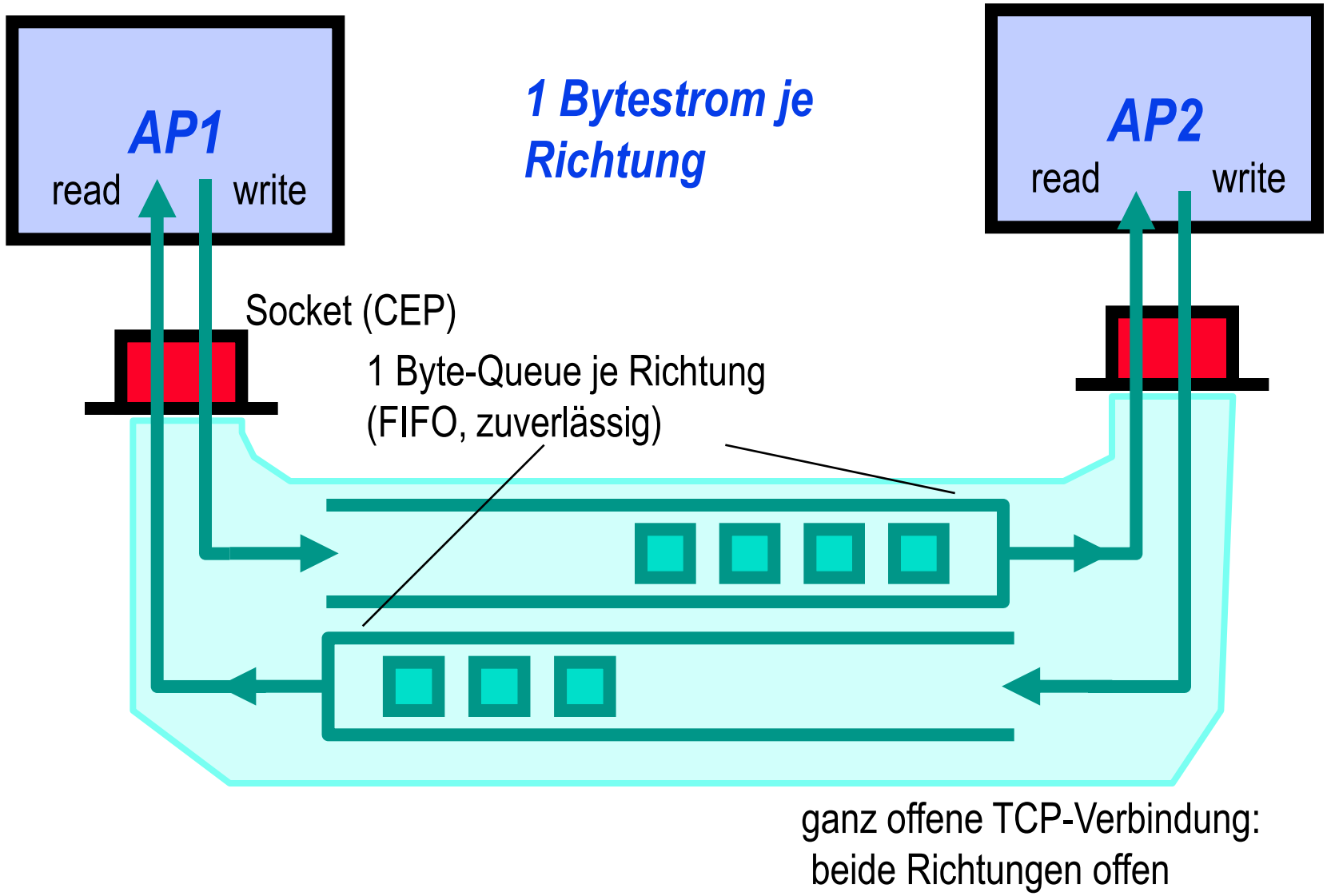
Client muss Server-Adresse kennen (IP-Adresse, Portnr.)

- Generiert einen TCP-Socket
- Spezifiziert die IP-Adresse und die Port-Nummer des Server-Sockets zur Kontaktaufnahme

- Client fordert den Aufbau einer TCP-Verbindung an
- Mit dem Verbindungsaufbau generiert der Server einen neuen Socket zur Kommunikation mit dem Client (so kann Server nebenläufig mit mehreren Clients kommunizieren)

Anwendungssicht:
TCP bietet eine zuverlässige, reihenfolgentreue Verbindung

TCP-Sockets: Anwendungssicht auf TCP-Verbindung



TCP-Socket - API

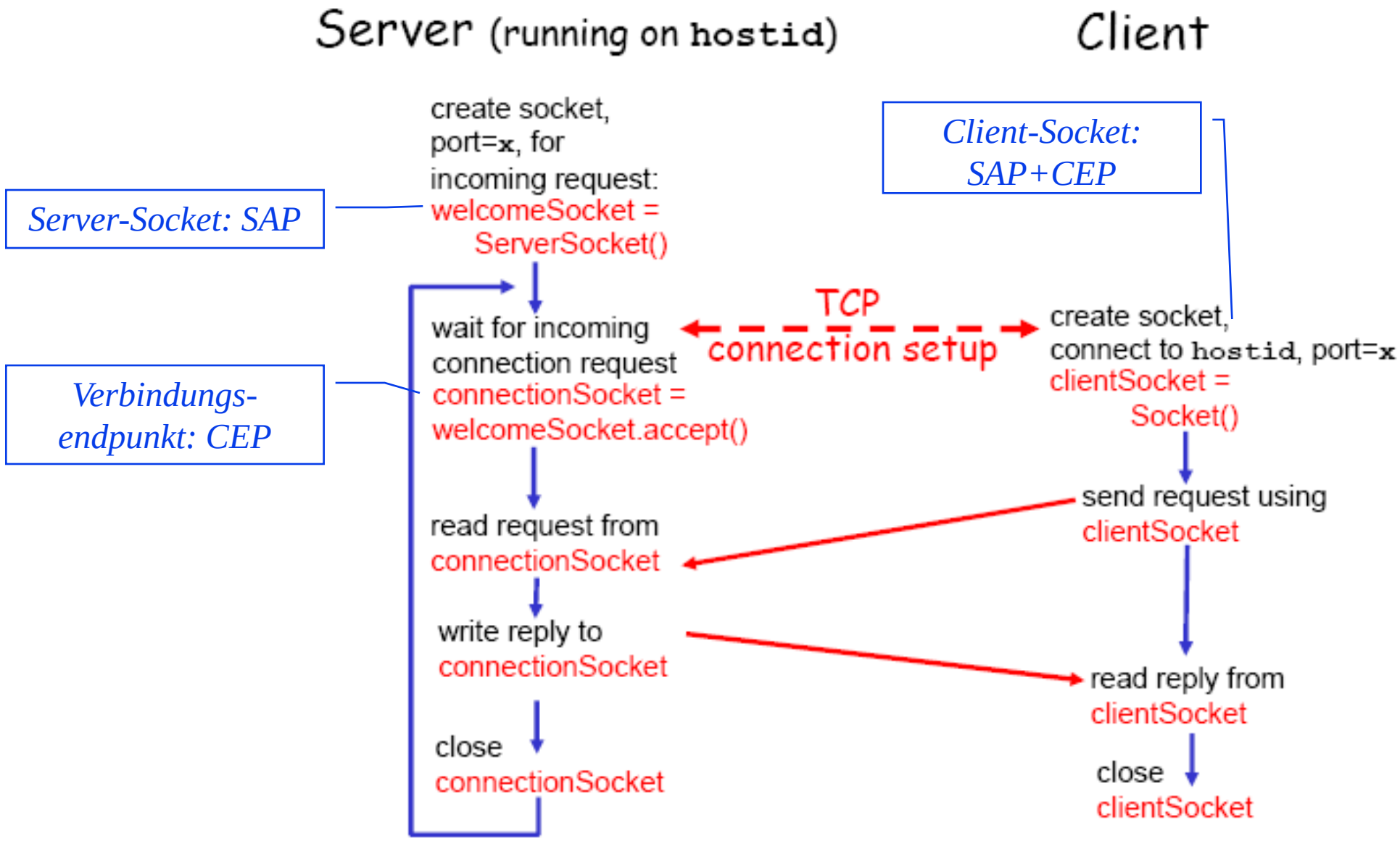
Auf Server-Seite:

- socket(), erzeugt einen Socket und liefert die Socket-ID für den Server
- bind(), bindet den Server-Socket an die IP-Adresse und Port-Nummer des Servers
- listen(), Server-Socket wird als Responder-Socket beim Betriebssystem angemeldet
- accept(), warten auf neue Verbindung, liefert CEP-Socket
- send(), schreiben in den CEP-Socket
- revc(), lesen aus dem CEP-Socket
- close(), schließt die Richtung Server→Client der Verbindung

Auf Client-Seite:

- socket(), erzeugt einen Socket und liefert die Socket-ID für den Client
- connect(), sendet einen Connect-Request an die angegebene IP-Adr. (existiert nicht in Java!)
- send(), schreibt Daten in den Client-Socket (Senden zum Server)
- revc(), liest Daten aus dem Client-Socket (Empfangen vom Server)
- close(), schließt die Richtung Client→Server der Verbindung

Client/Server: TCP - Interaktion



UDP-Sockets

UDP nutzt keine Verbindung zwischen Client und Server

- Kein Verbindungsauf- und Abbau (Connect/Accept, Close)

- Sender muss mit jedem Senden die IP-Adresse und Port-Nummer des Empfängers angeben
- Empfänger erfährt die IP-Adresse und Port-Nummer des Senders aus dem empfangenen Datagramm

Daten via UDP können in falscher Reihenfolge ankommen oder ohne Anzeige verloren gehen

Anwendungssicht:
UDP bietet nur eine unzuverlässige Nachrichtenübertragung

UDP-Socket - API

Auf Server-Seite:

- socket(), erzeugt einen Socket und liefert die Socket-ID für den Server
- bind(), bindet den Server-Socket an die IP-Adresse und Port-Nummer des Servers
- sendto(), schreibt in den Server-Socket unter Angabe der IP-Adresse und Port-Nummer des gewünschten Empfängers
- revcfrom(), liest aus dem Server-Socket inkl. IP-Adresse und Port-Nummer des Senders

Auf Client-Seite:

- socket(), erzeugt einen Socket und liefert die Socket-ID für den Client
- sendto(), schreibt in den Client-Socket unter Angabe der IP-Adresse und Port-Nummer des gewünschten Empfängers
- revcfrom(), liest aus dem Client-Socket inkl. IP-Adresse und Port-Nummer des Senders
- bind(), bindet den Socket an eine gewünschte Port-Nummer (optional)

Client/Server: UDP - Interaktion