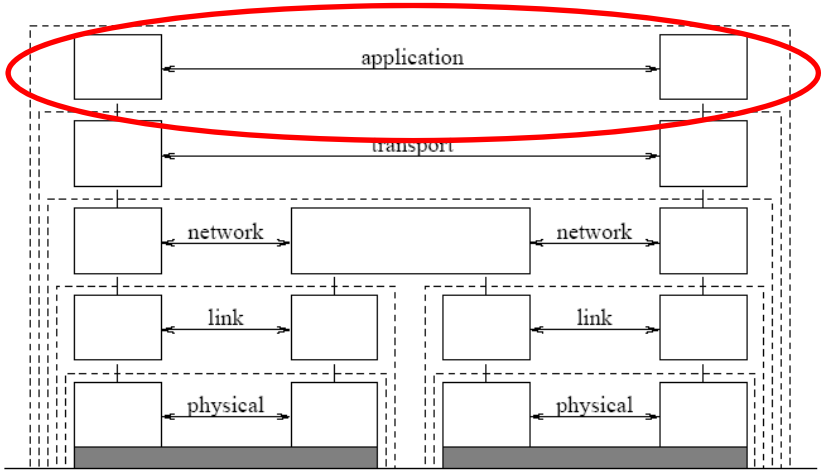
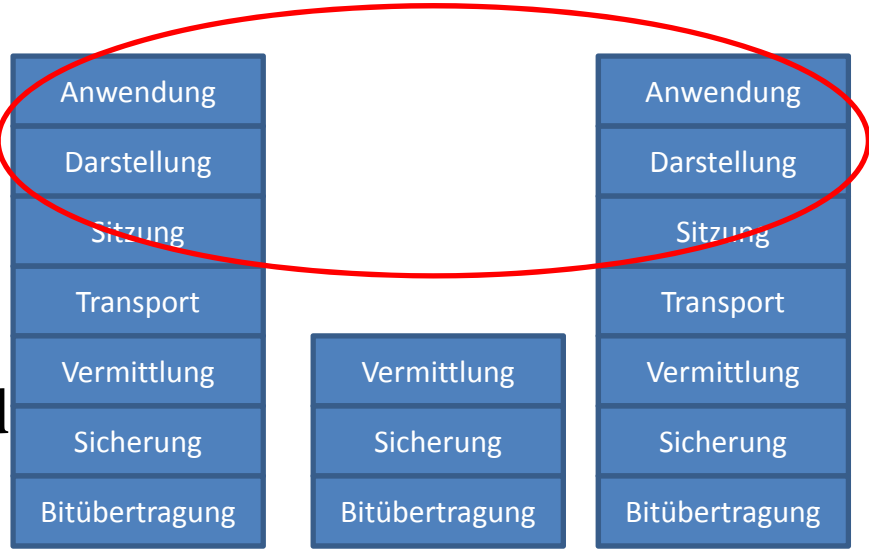
Die Anwendungsschicht

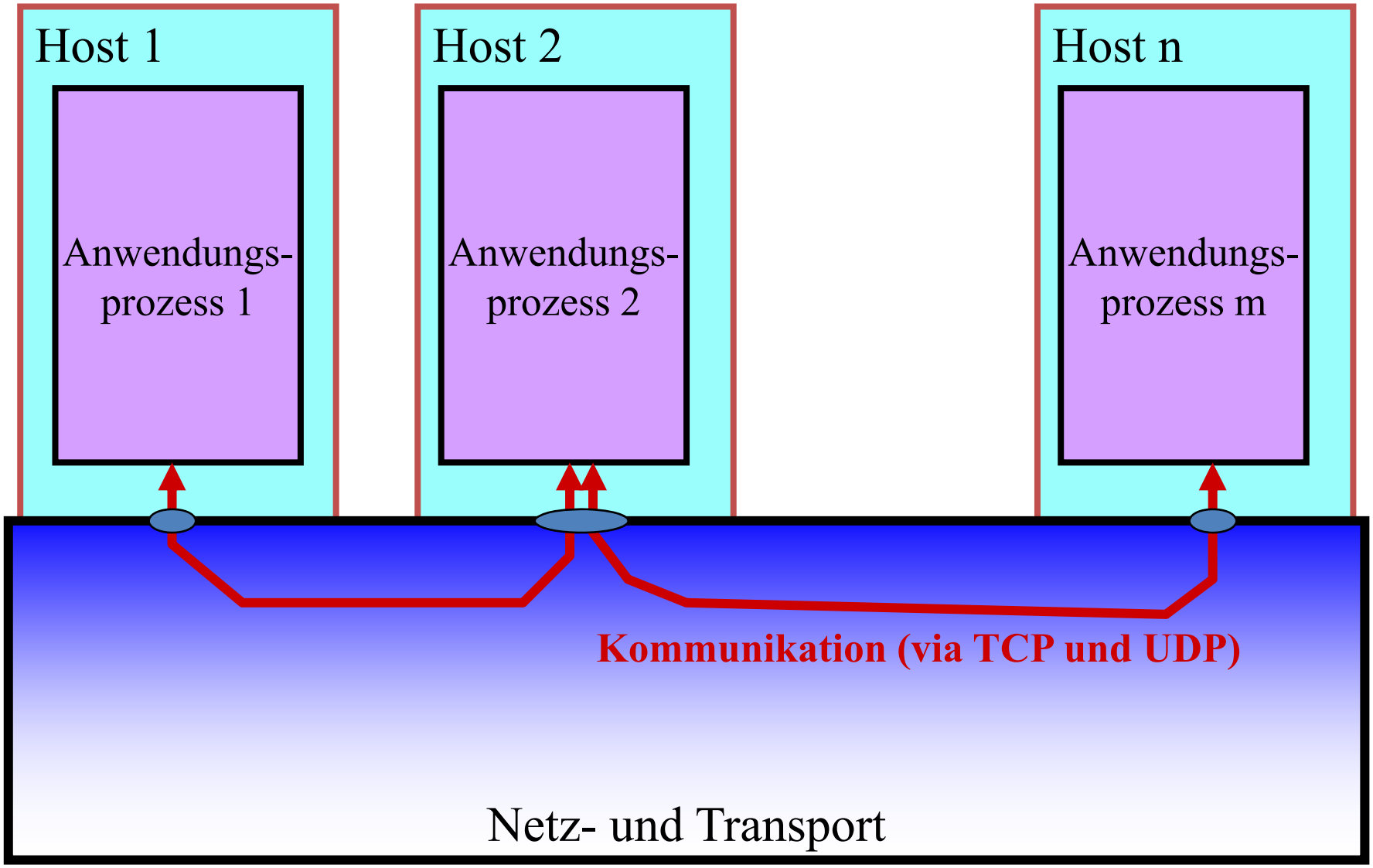
Gliederung

- Prinzipien
- WWW, FTP, Mail
- DNS
- P2P-Anwendungen
- TCP-, UDP-Sockets
- Client/Server-Programme und Socket Programmierung





Anwendung: Struktur



Netzwerkanwendungen

Anwendung läuft auf unterschiedlichen Endsystemen und besteht damit aus mehreren **Prozessen**
(Prozess := Programm, das auf einem Rechner läuft siehe BS)

Um Interaktionen durchzuführen, müssen Prozesse **kommunizieren**
(Kommunikation von Prozessen auf einem Rechner siehe BS, Kommunikation von Prozessen auf unterschiedlichen Rechnern durch Anwendungsprotokoll)

Netzwerkanwendung nutzt die Protokolle der Anwendungsschicht zur Realisierung von Diensten für einen Benutzer
(z.B. Browser nutzt HTTP)

Kommunikationsszenarien

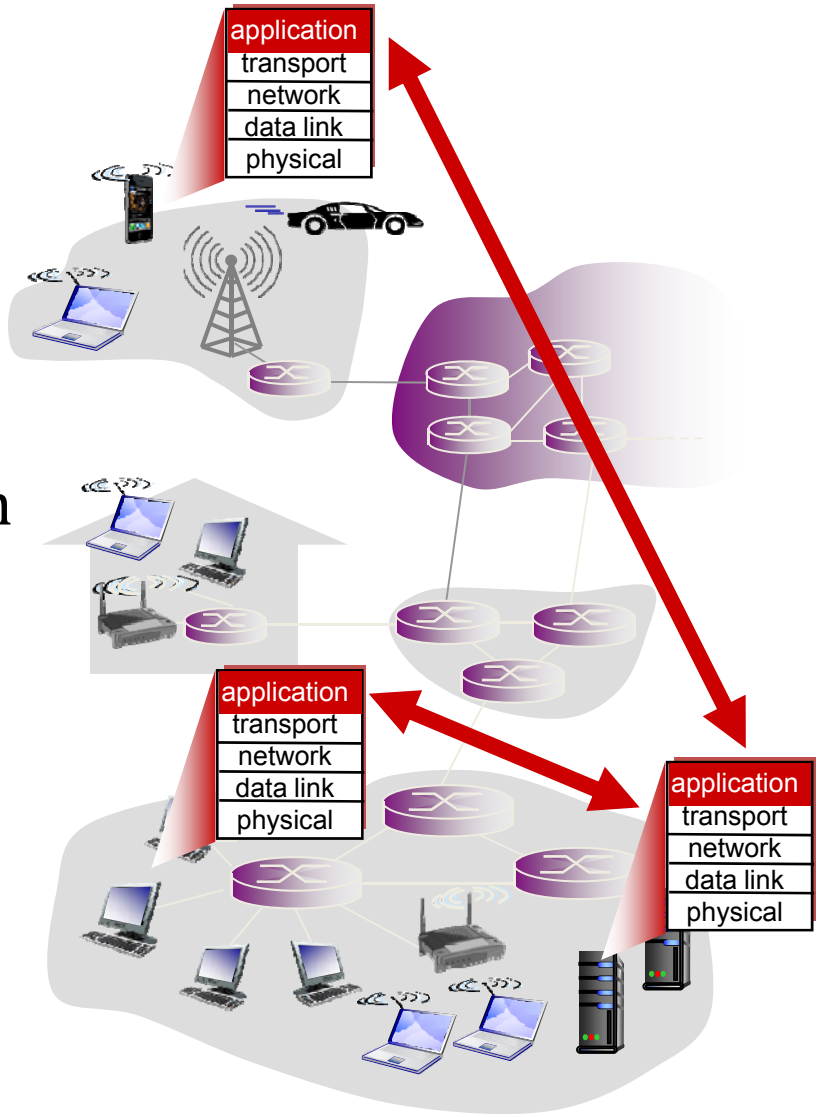
Viele Anwendungen bestehen aus zwei Teilen: **Client & Server**

Client:

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

Server:

- Bietet dem Client Dienste an
- Bsp. bietet Web-Seiten an,
  liefert E-Mails

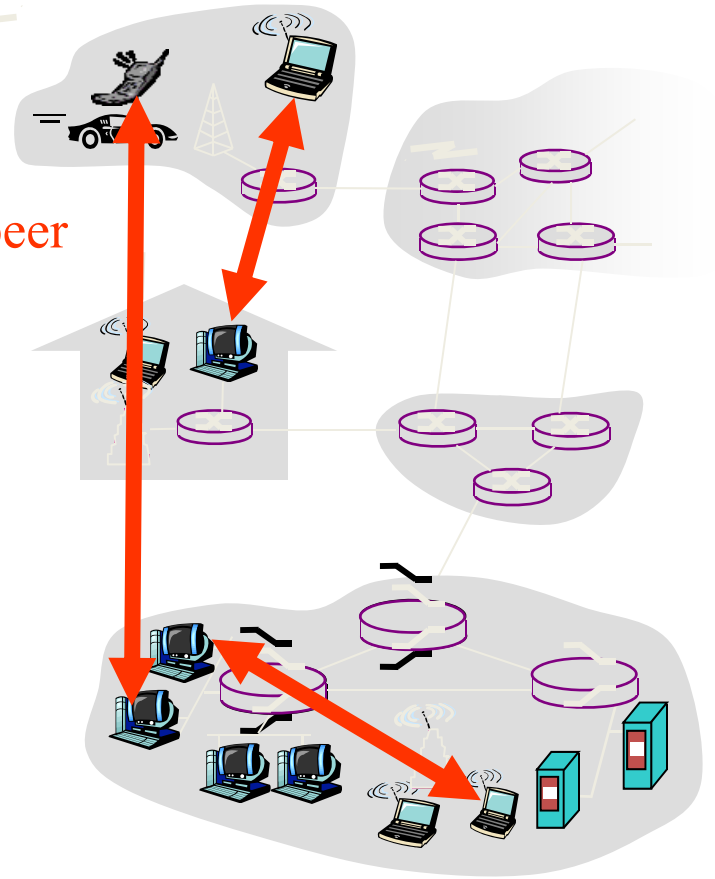


Kommunikationsszenarien

Alternative:

Peer to Peer (P2P) Systeme

- Benutzerprozesse bieten und fordern Dienste an
- Rechner kommunizieren direkt, ohne Einschaltung eines Servers
- Architektur skaliert, da keine zentralen Komponenten
- Management, Dynamik, Konsistenz, Sicherheit können zum Problem werden
- Bsp. BitTorrent, Skype,..



© Kurose/Ross 2009

Kommunikationsszenarien

Viele P2P-Anwendungen enthalten Client-Server Komponenten:

- Skype:
  Voice-over-IP-Anwendung im Prinzip nach dem P2P-Prinzip aber
  Finden des Partners über einen zentralen Server, danach direkte Kommunikation zwischen Clients

- Instant Messaging:
  Chat zwischen Benutzern nach dem P2P-Prinzip aber
  Verwalten der aktiven Benutzer über einen zentralen Server

Anwendungsprotokollspezifikation

- Nachrichtentypen (z.B. request-response, …)
- Syntax der Nachrichten Felder in der Nachricht, Datentypen in den Feldern
- Semantik der einzelnen Nachrichtenfelder
- Regeln, wann & wie Nachrichten generiert werden und wann & wie auf Nachrichten reagiert wird

Public Domain Protokolle

- werden in Standardisierungs-dokumenten (RFCs request for comments, es gibt über 5000) festgelegt
- sind oft interoperabel
- Bsp. HTTP, SMTP

Proprietäre Protokolle

- für spezielle Netze oder Anwendungen

Schnittstelle zum Transportsystem

Prozess 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 (32Bit für IP-V4, 128 Bit für IP-V6)

Damit sind aber noch keine Prozesse adressierbar!!

Zusatz 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

Anforderungen an Transportdienste

Anwendungen nutzten einen Transportdienst und haben unterschiedliche Anforderungen

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. E-Mail, Dateitransfer)

Anforderungen an Transportdienste

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)

Anforderungen an Transportdienste

Anwendung	Datenverlust	Durchsatz	Zeitkritisch
Dateitransfer	verlustfrei	elastisch	nein
E-Mail	verlustfrei	elastisch	nein
WWW	verlustfrei	elastisch	nein
Realzeit Audio/Video	tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	ja, 100's msec
Audio/Video	tolerant	wie vorher	ja, wenige Sek.
Interaktive Spiele	Tolerant	wenige kbps bis …	ja, 100's msec
Instant Messaging	verlustfrei	elastisch	ja / nein
Netz-Management	teils/teils	keine	ja / nein

Internet – Transport: Dienste & Protokolle

TCP

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

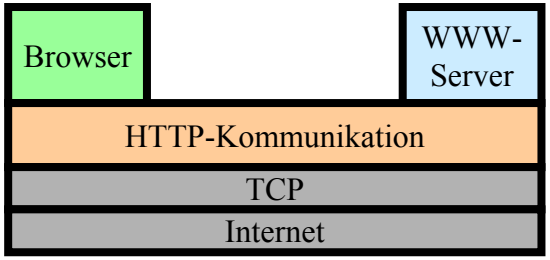
UDP

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

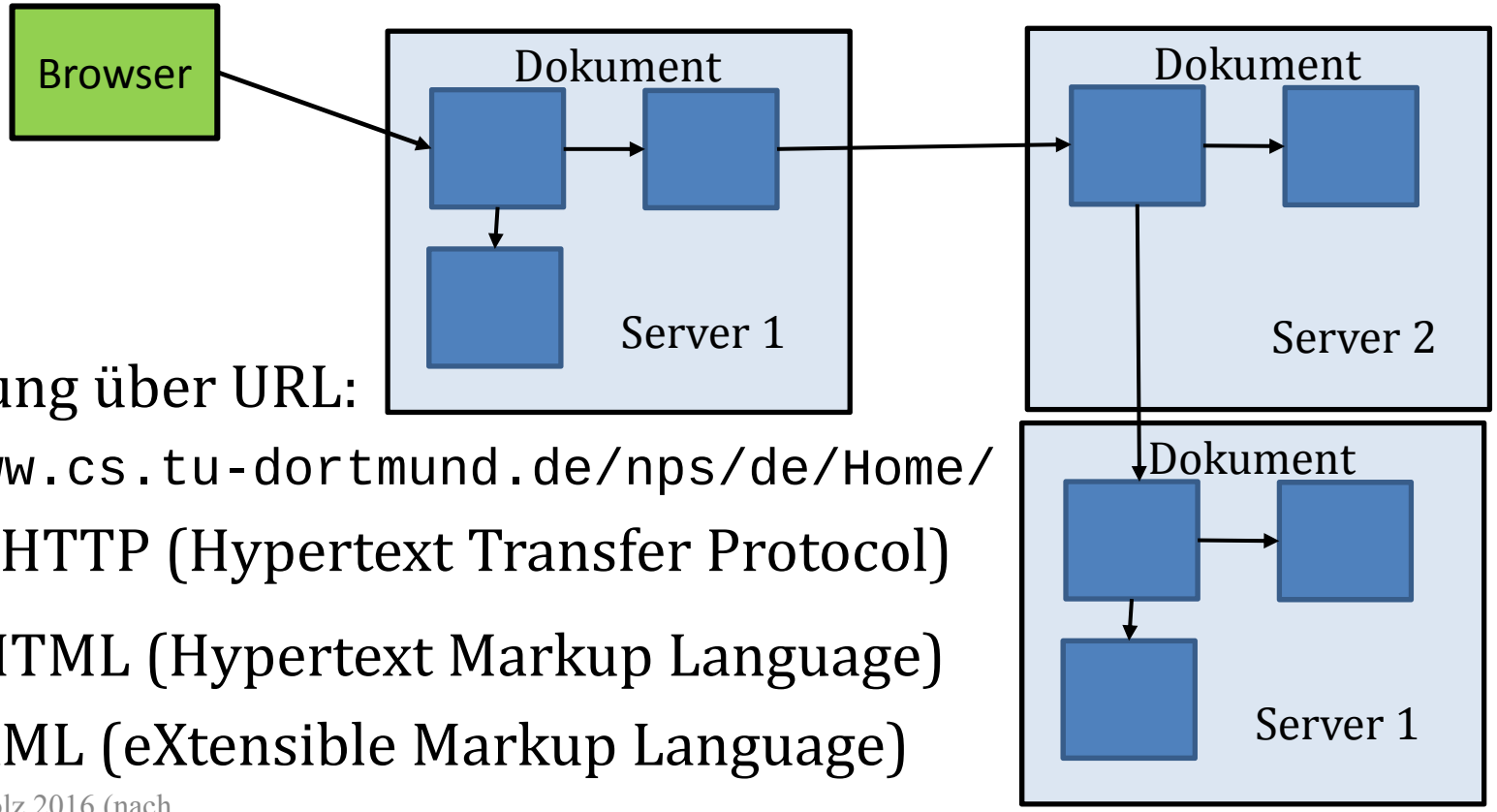
Internet-Anwedungen und benutzter Transport

Anwendung	Anwendungsprotokoll	Transportprotokoll
E-Mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
WWW	HTTP [RFC 2616]	TCP
Dateitransfer	FTP [RFC 959]	TCP
streaming multimedia	HTTP (z.B. Youtube), RTP [RFC 1889]	TCP oder UDP TCP oder UDP
Internettelefonie	SIP, RTP, proprietär (e.g., Skype)	typischerweise UDP

WWW: World Wide Web



WWW: Weltweites Hypertext-System
im Internet
Dokument mit logischer Verknüpfung



Adressierung über URL:

`http://www.cs.tu-dortmund.de/nps/de/Home/`

Protokoll: HTTP (Hypertext Transfer Protocol)

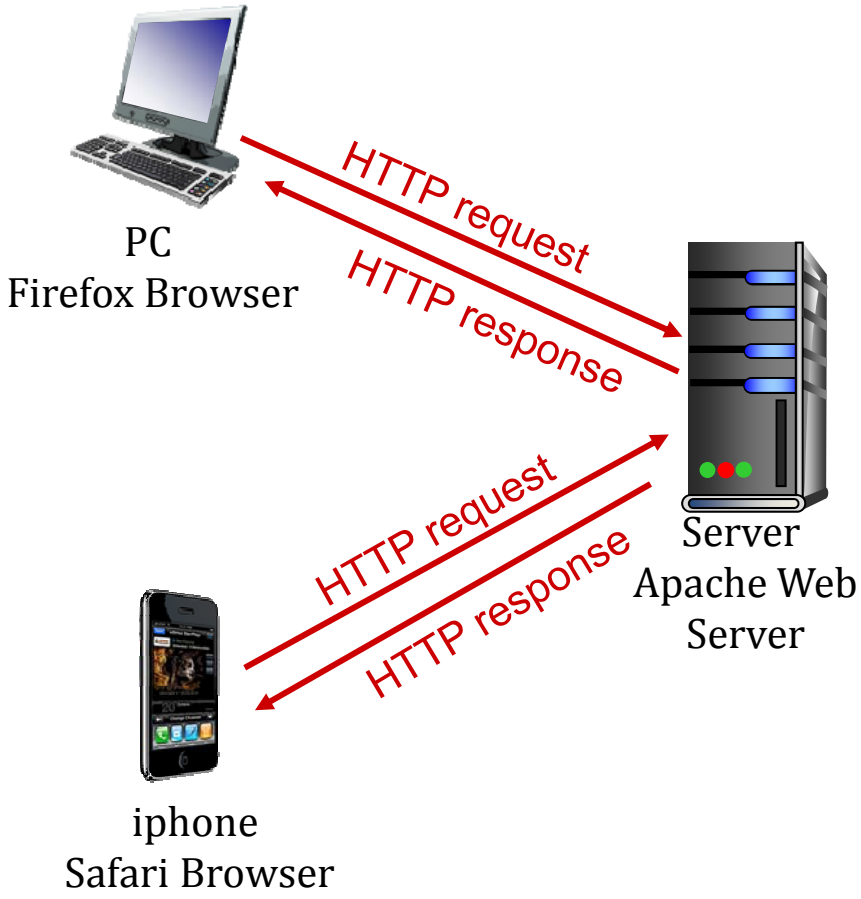
Sprache: HTML (Hypertext Markup Language)

XML (eXtensible Markup Language)

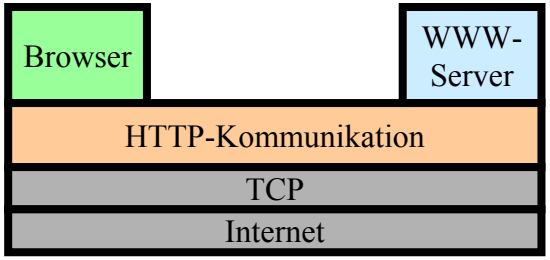
Ablauf der Kommunikation im WWW mit HTTP

HTTP: Hypertext- Transfer-Protocol

- Anwendungsprotokoll des Web's
- client/server Ansatz
  - *client:* Browser der Web-Seiten/Objekte anfordert, empfängt und anzeigt (mit Hilfe des HTTP-Protokolls)
  - *server:* Web -Server sendet Objekte als Antwort auf Anforderungen (mit Hilfe des HTTP-Protokolls, request-response)



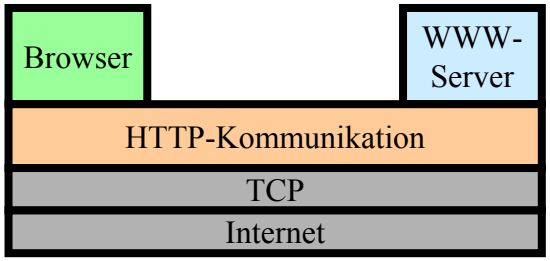
WWW: World Wide Web



HTTP: Hypertext-Transfer-Protocol

- 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
- zustandslos: 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.

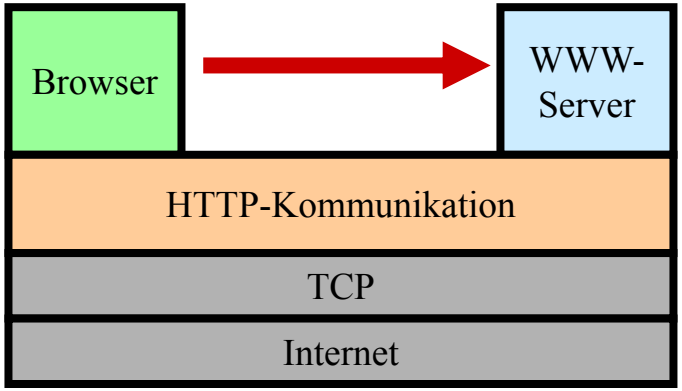
Persistent-HTTP

- 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

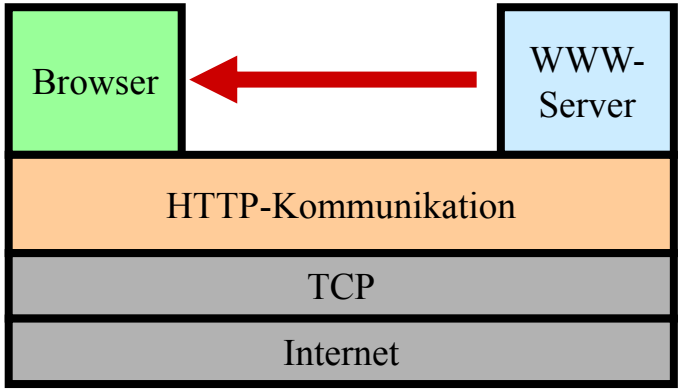
METHOD *URI* HTTP/1.1 <crlf>
Header-Field-Name : value <crlf>
...
Header-Field-Name : value <crlf>
<crlf>
MIME-conform Body

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

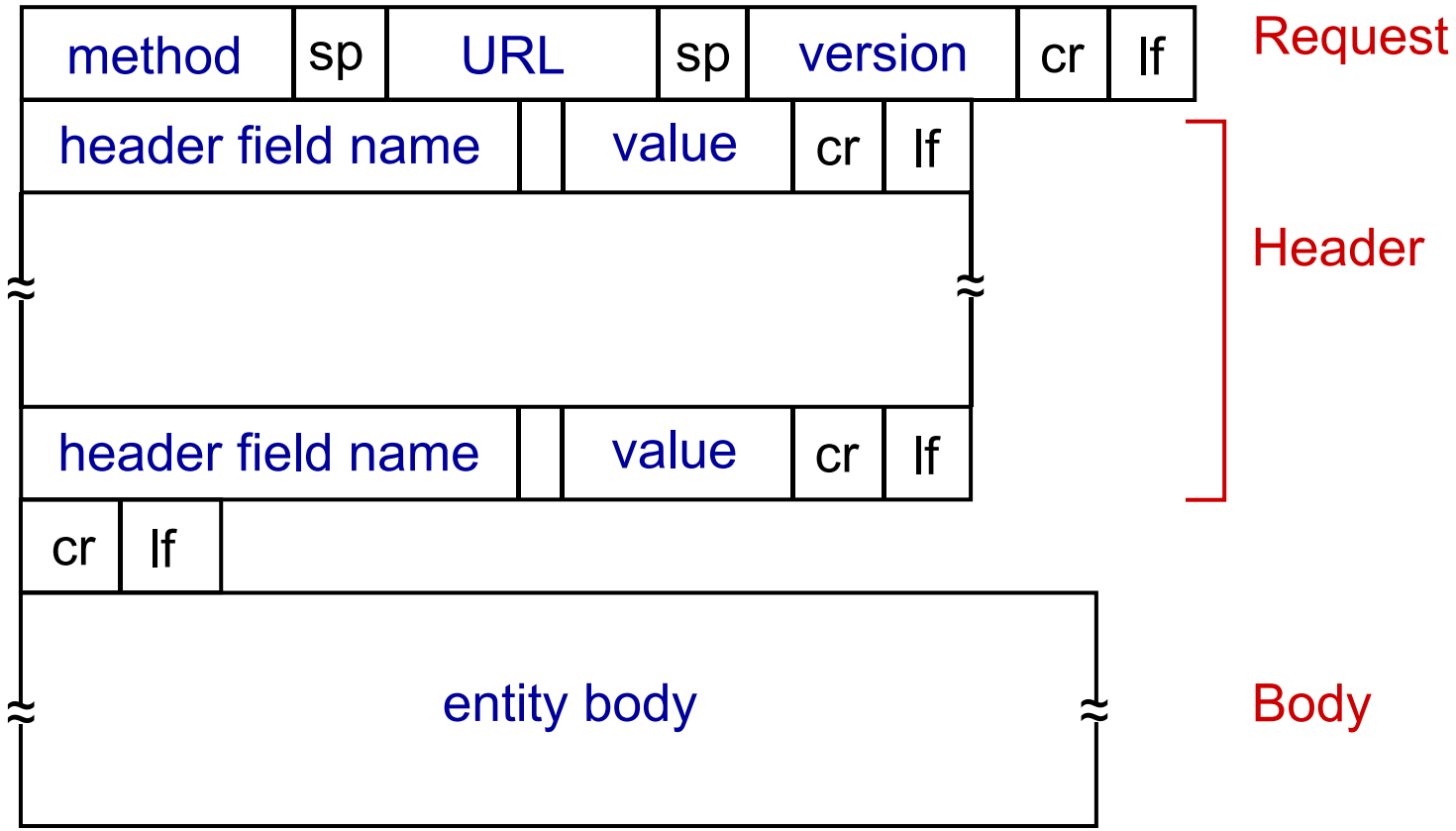


Response-Status-Codes

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

WWW: HTTP-PDUs

Format der HTTP-Nachrichten



WWW-HTTP Operationen

GET-Methode: Fordert das Objekt mit gegebener URL an

```
ls4com2> Telnet ls4-www.cs.tu-dortmund.de 80
Trying 129.217.16.36...
Connected to willi.
Escape character with´^]´.
GET http://ls4-www.cs.tu-dortmund.de/cms/de/lehre/2016_ws/rvs
<!DOCTYPE HTML PUBLIC „-//IETF//DTD HTML 2.0/EN“>
<html>
....
</html>
```

Öffnet TCP-Verbindung auf Port 80

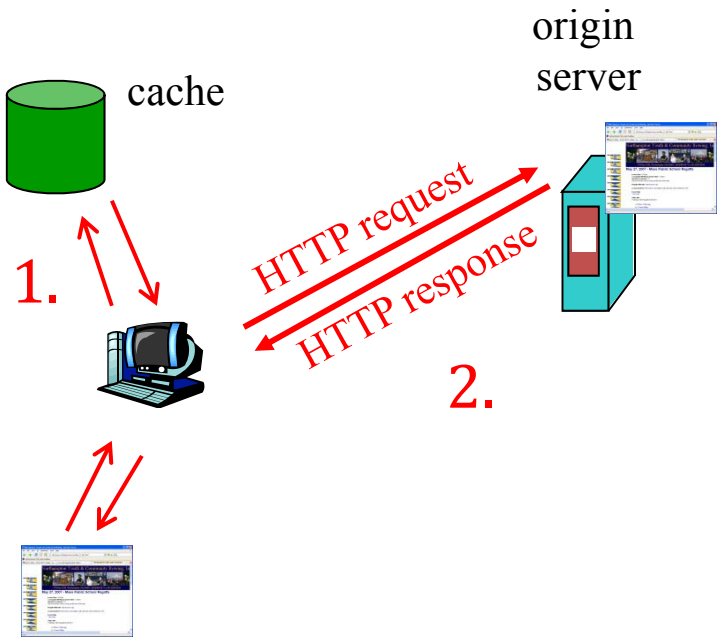
Liefert die ange-forderte Web-Seite

WWW. Web-Caching

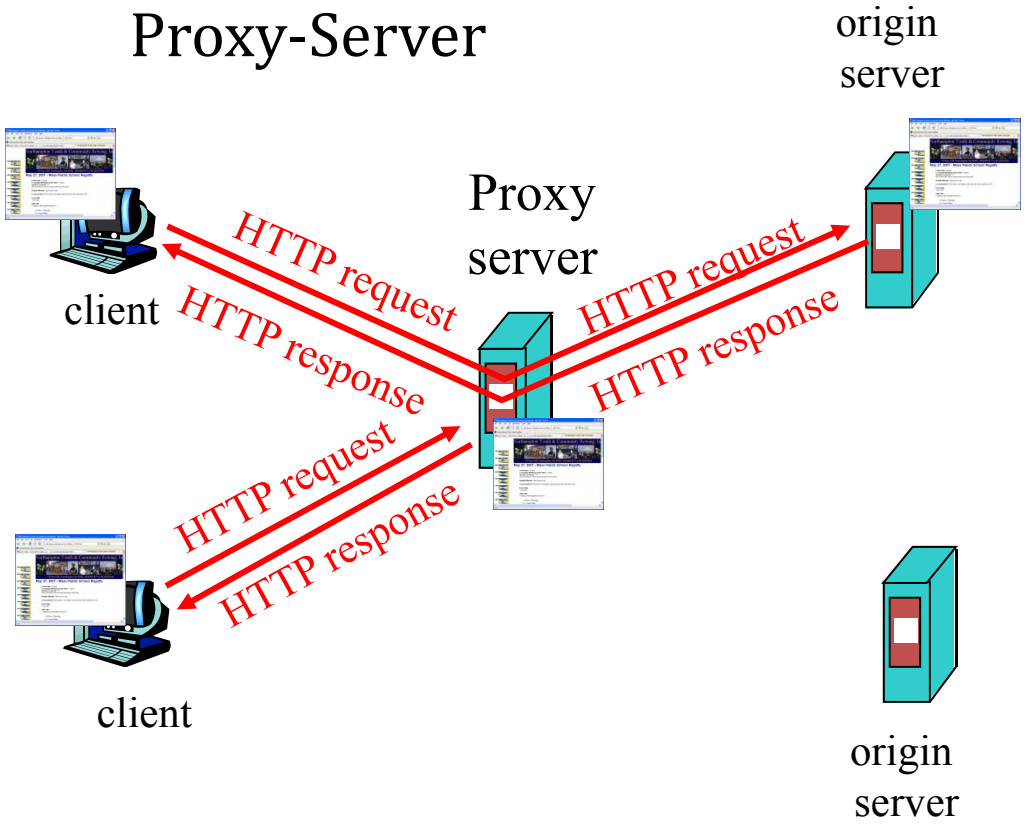
Laden von Seiten kann aufwändig/langwierig sein

⇒ Verringerung des Aufwandes durch Nutzung vorhandener Daten oder gemeinsame Laden von Daten

Caching im Browser



Proxy-Server

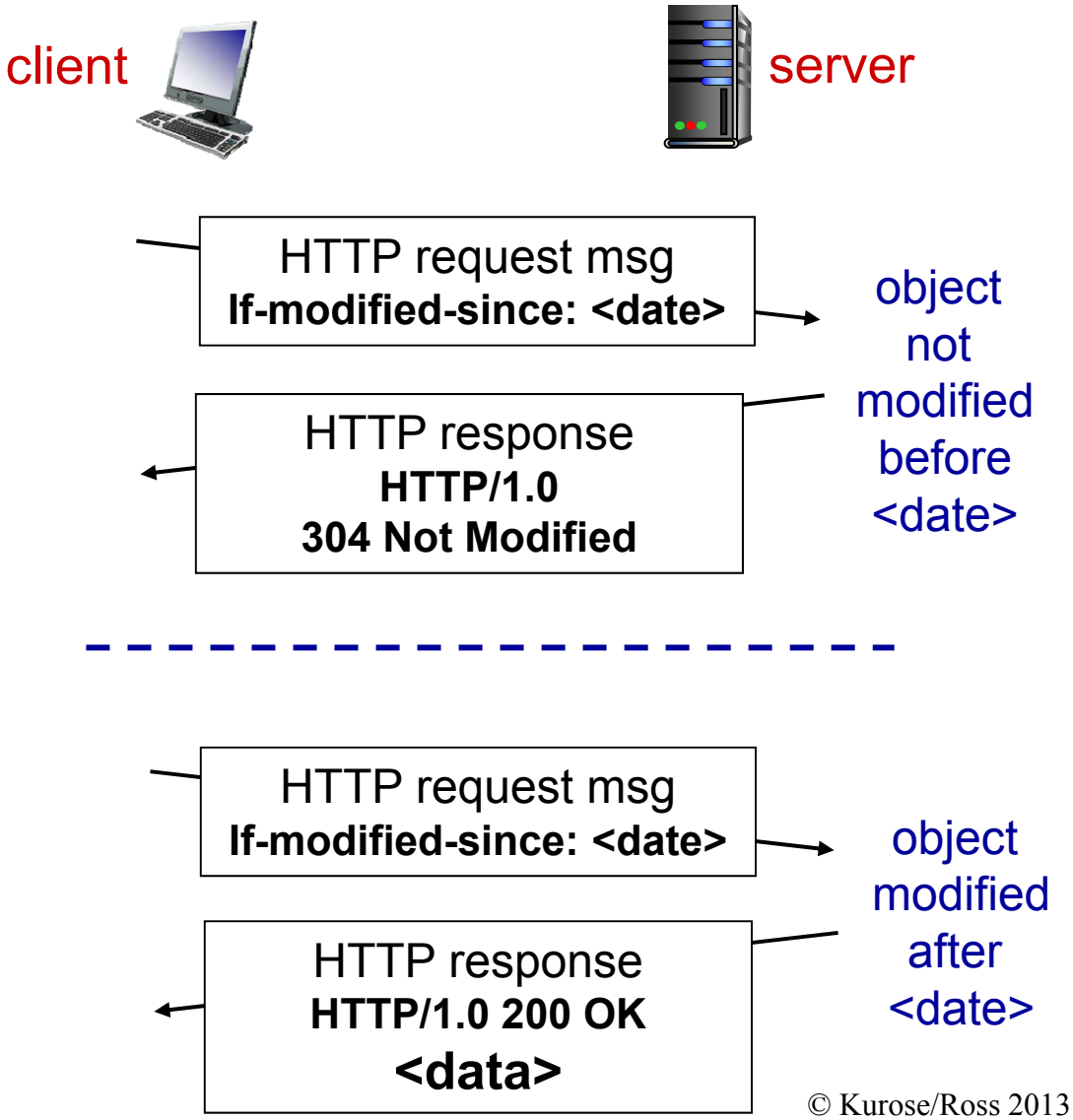


© Kurose/Ross 2009

WWW: Bedingtes GET

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

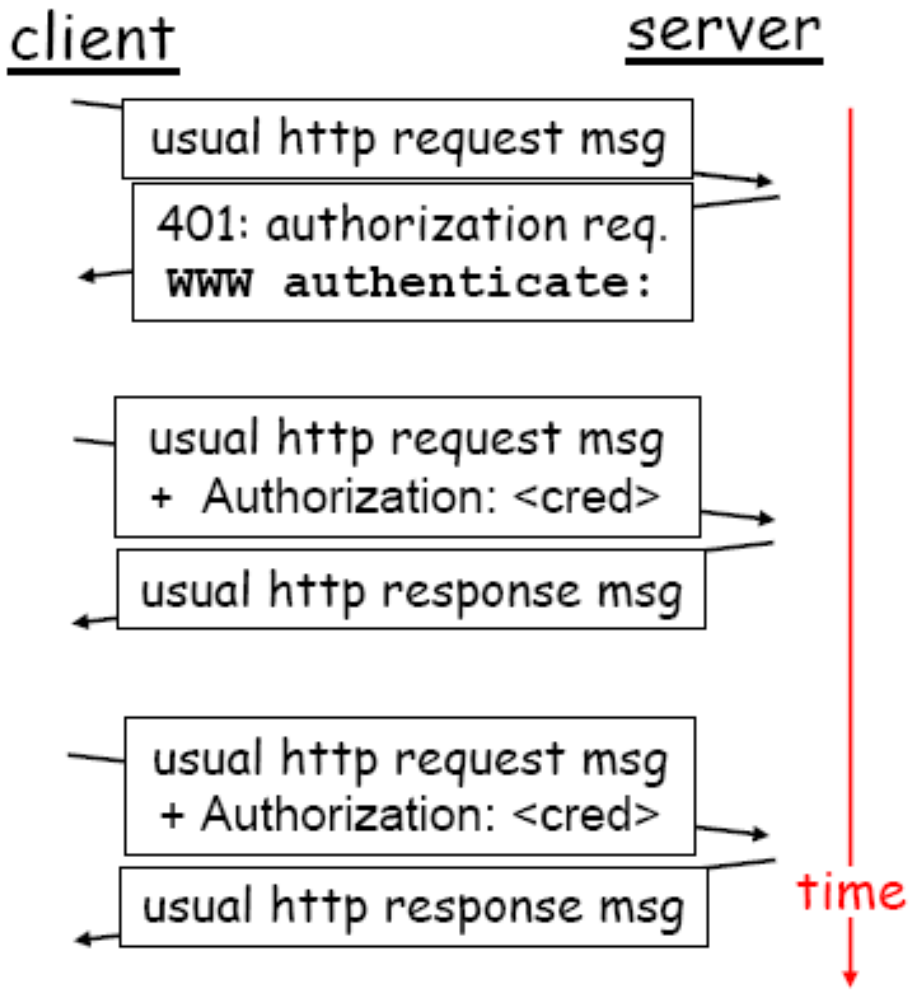
- Client (u.U. Proxy) sendet das Datum seiner Kopie mit (if-modified-since)
- Server schickt das angeforderte Objekt nur, wenn eine neue Version vorhanden



WWW: Authentifikation und Autorisierung

Autorisierung:
Kontrollierter Zugriff auf Inhalte eines Servers

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

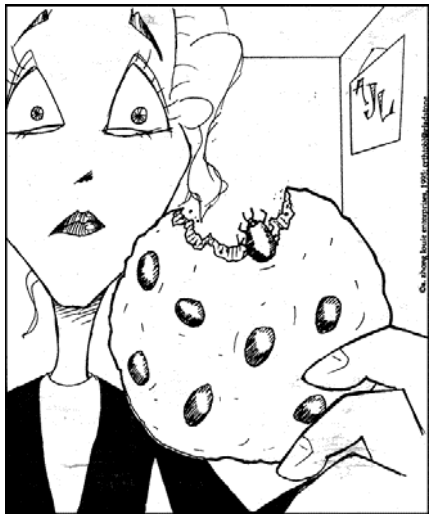


Vorsicht: Benutzername und Password im Browser-Cache!

WWW: Zustandsspeicherung durch Cookies

Viele Web-Seiten nutzen
Cookies
Basiskomponenten

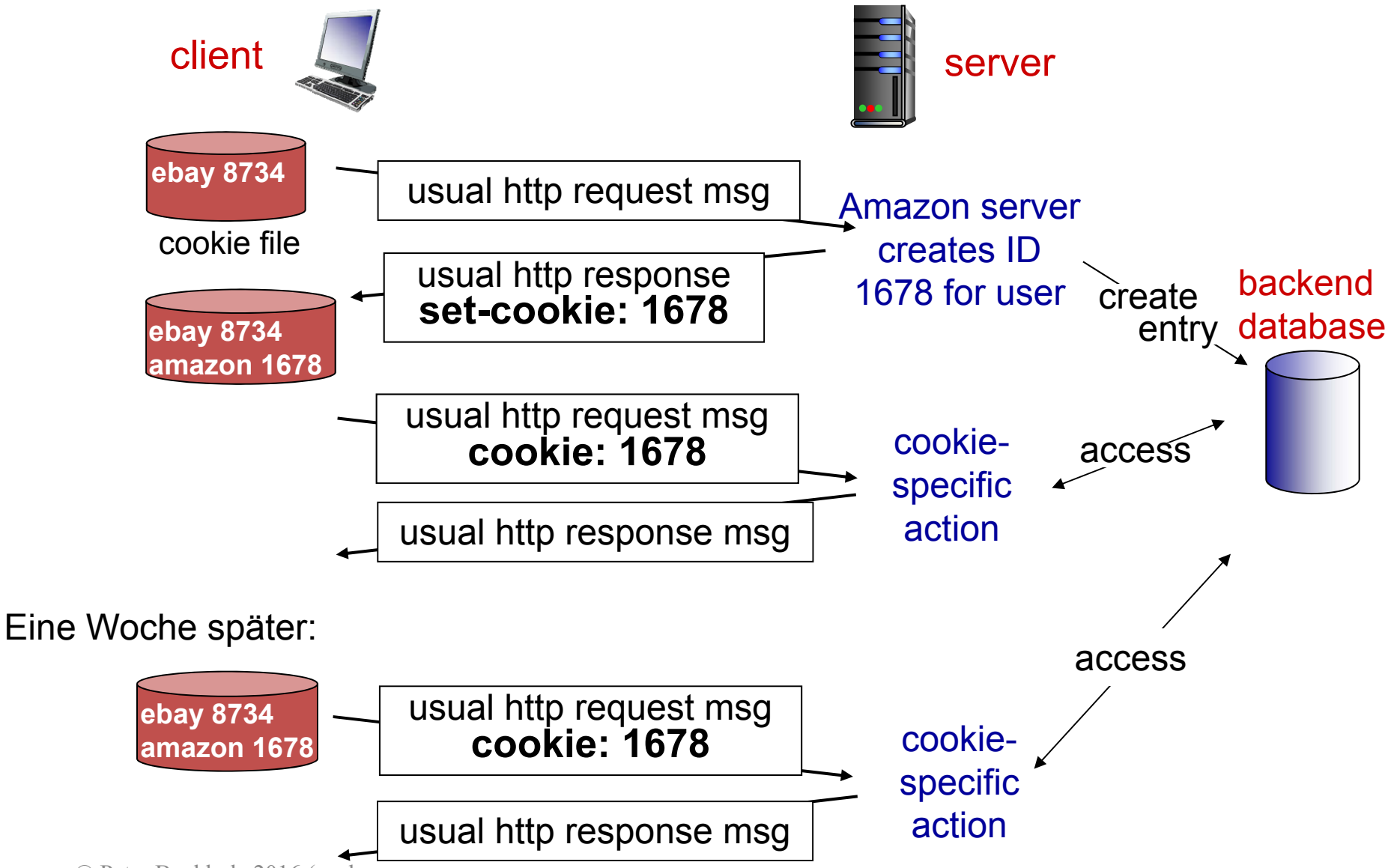
1. Cookie-Zeile im Header der HTTP-Response Nachricht
2. Cookie-Zeile in der HTTP-Request Nachricht
3. Cookie-Datei auf dem Benutzerrechner, vom Browser verwaltet
4. Datenbank mit Cookies auf Server-Seite



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 durch Cookies



WWW: Zustandsspeicherung durch Cookies

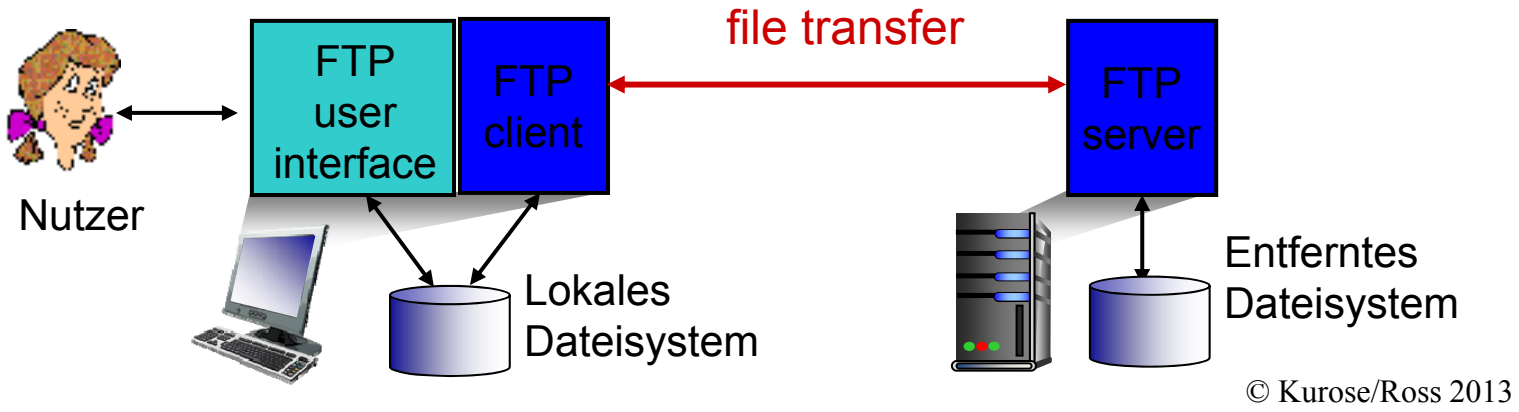
Vorteile der Cookies:

- 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

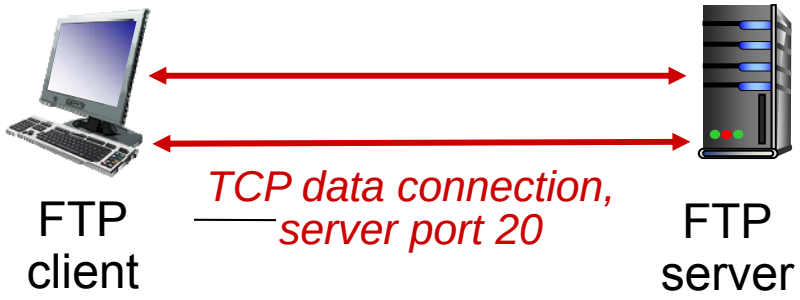


Aufgabe: Übertragung einer Datei von/zu einem entfernten Rechner

- 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 Aufruf

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: Eigenschaften

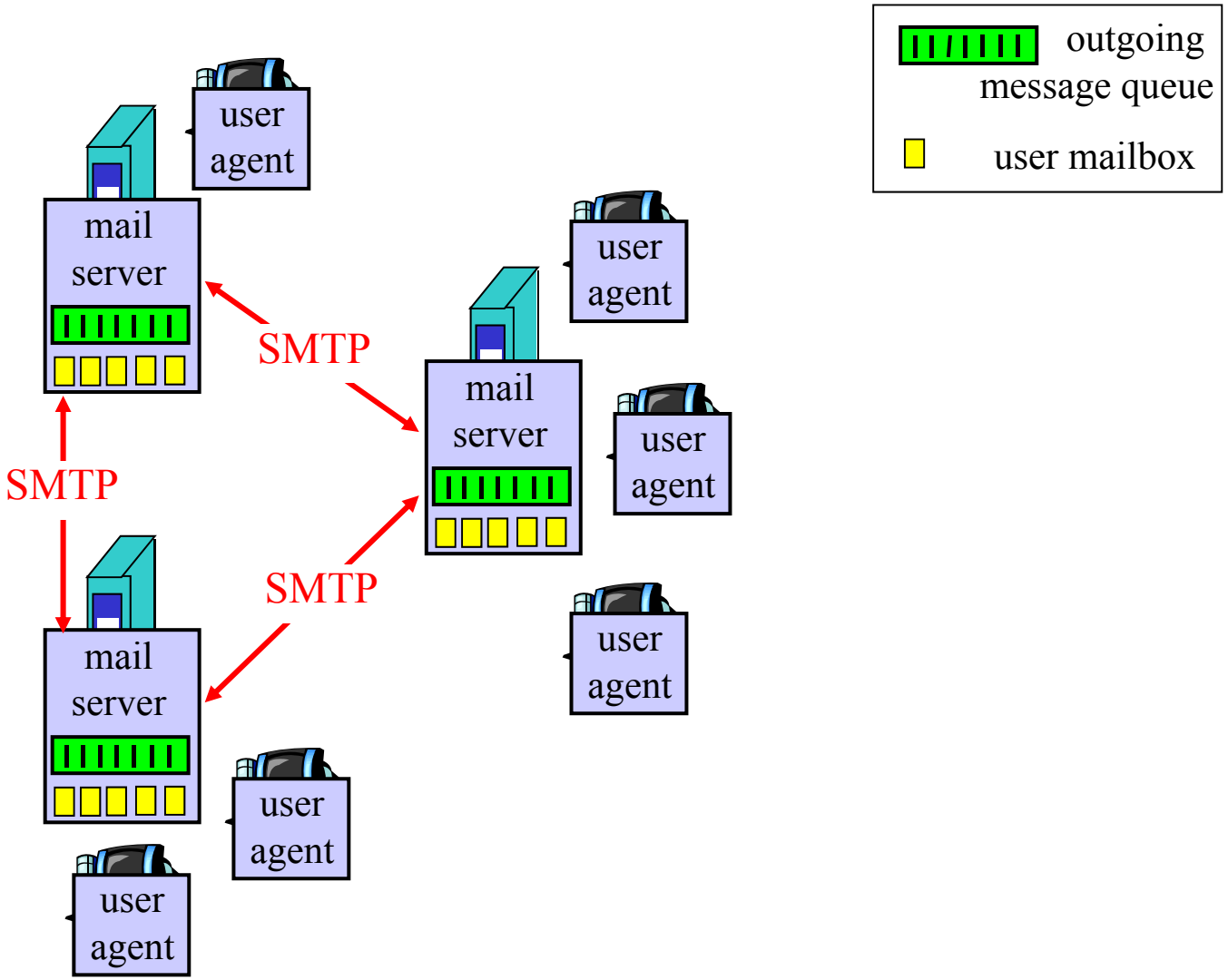
Größter Nachteil von FTP: Sicherheitsprobleme
(altes Protokoll)

- Bei jeder Anmeldung werden Benutzername und Password als Klartext übertragen
- Auch die Daten werden unverschlüsselt übertragen
- ✓ Ein fremder Rechner könnte mithören und dabei Zugangsdaten und übertragene Daten ausspähen

FTP, wenn überhaupt, nur im Intranet einsetzen!!

Besser ist die Verwendung von SFTP mit praktisch identischer Funktionalität aber verschlüsselter Anmeldung und Übertragung (siehe auch Kap. 8)

E-MAIL



E-MAIL: Protokolle

SMTP: Simple Mail Transfer Protocol [RFC 2821]

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

Empfang von E-Mails:

Mail-Server speichert empfangene E-Mails

User-Agenten greifen auf die E-Mails 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

•

From: buchholz@ls4.cs.tu-dortmund.de
To: falko.bause@udo.edu, krumm@ls4
Subject: Is this your picture
MIME-Version: 1.0
Content-Tranfer-Encoding: base64
Content-Type: image/jpeg

Base64 encoded data ……..
……
…..base64 encoded data

E-MAIL: MIME – Multipurpose Internat Mail Extension

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

Lösung MIME-Format

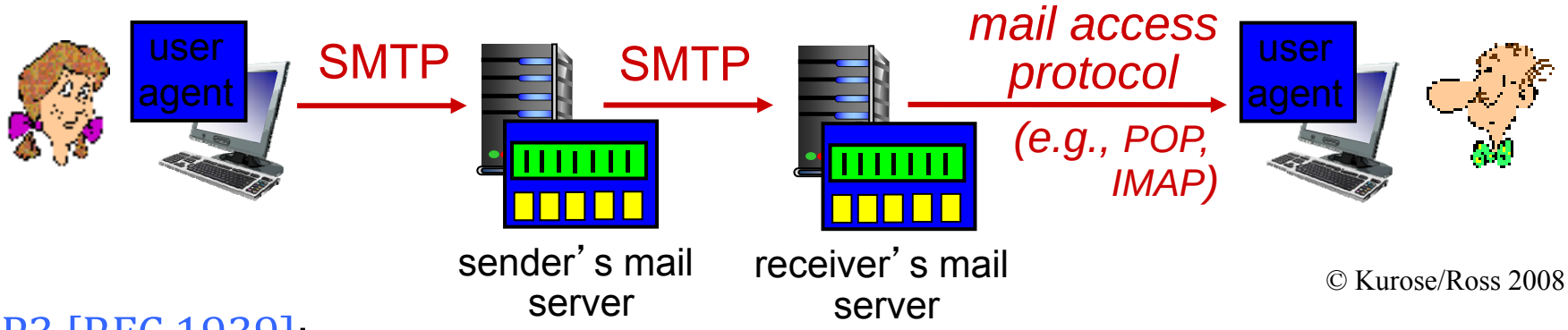
Beim Sender

- Daten in ASCII kodieren
- MIME Header hinzufügen, in dem 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: Zugriff auf den Server



POP3 [RFC 1939]:

- Sehr einfaches Protokoll
- 3 Phasen
  - Authentifizierung
  - Transaktion (inkl. Löschen von Nachrichten)
  - Update

HTTP (Web-basiert)

- Nachrichten werden per HTTP verschickt
- Funktionalität im Browser

IMAP [RFC 3501]:

- Deutlich komplexer als POP3
- Erlaubt u.a.
  - Verwalten von Ordnern auf dem Server
  - Auslesen von Mail-Headern
  - Suchen von Nachrichten

DNS: *Domain Name System*

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

Menschen werden durch mehrere Identifikatoren identifizierbar:

- Name, Pass, SSN, …

Internet Hosts oder Router:

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

⇒ Abbildung Name ↔ IP-Adresse

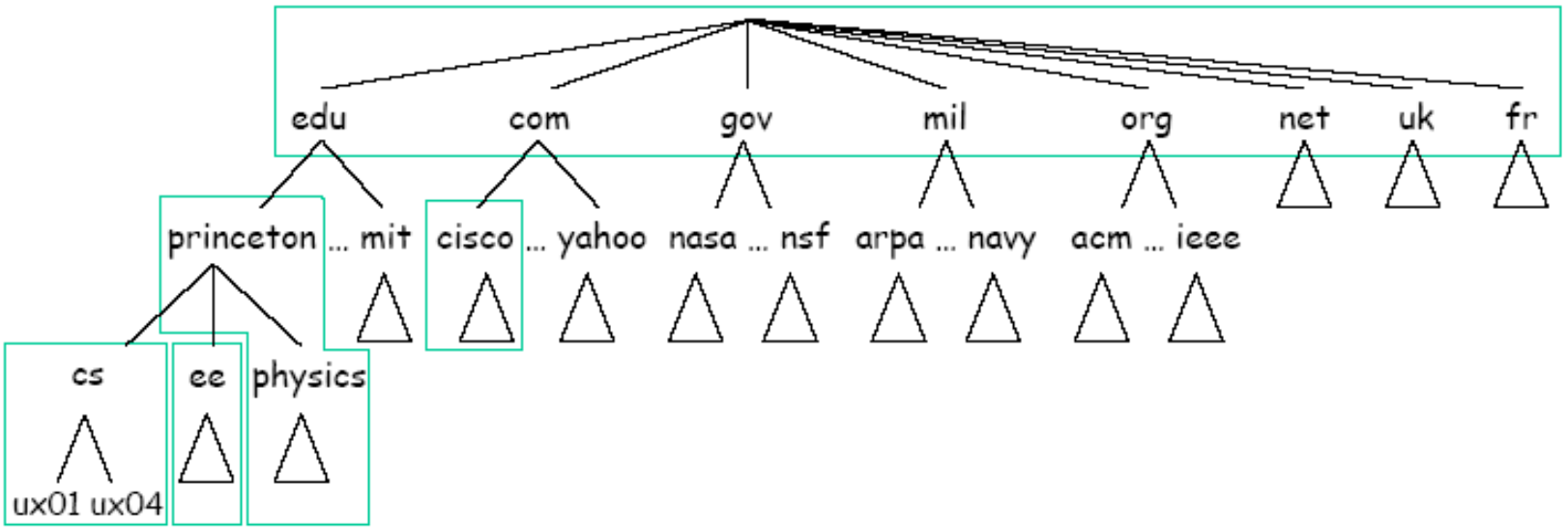
Domain Name System

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

DNS: *Domänen-Hierarchie*

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

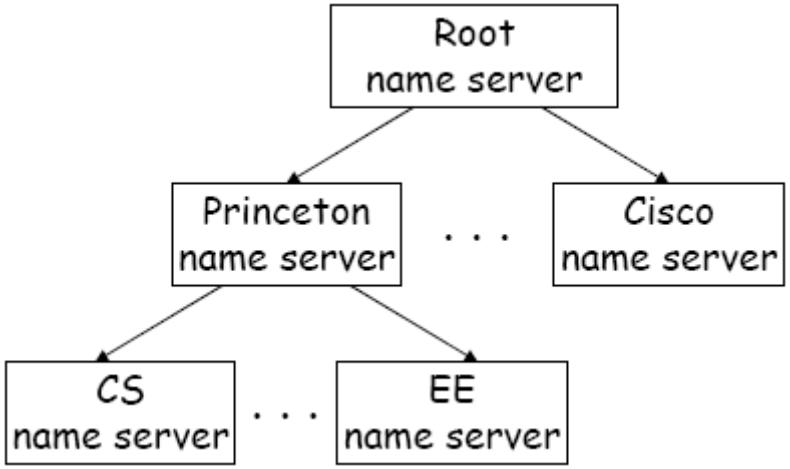
- DNS Namen werden von rechts nach links abgearbeitet
- Namenshierarchie kann als Baum visualisiert werden
  - Blätter sind Rechnernamen
  - Innere Knoten gehören zu Domains



DNS: *Name Server-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

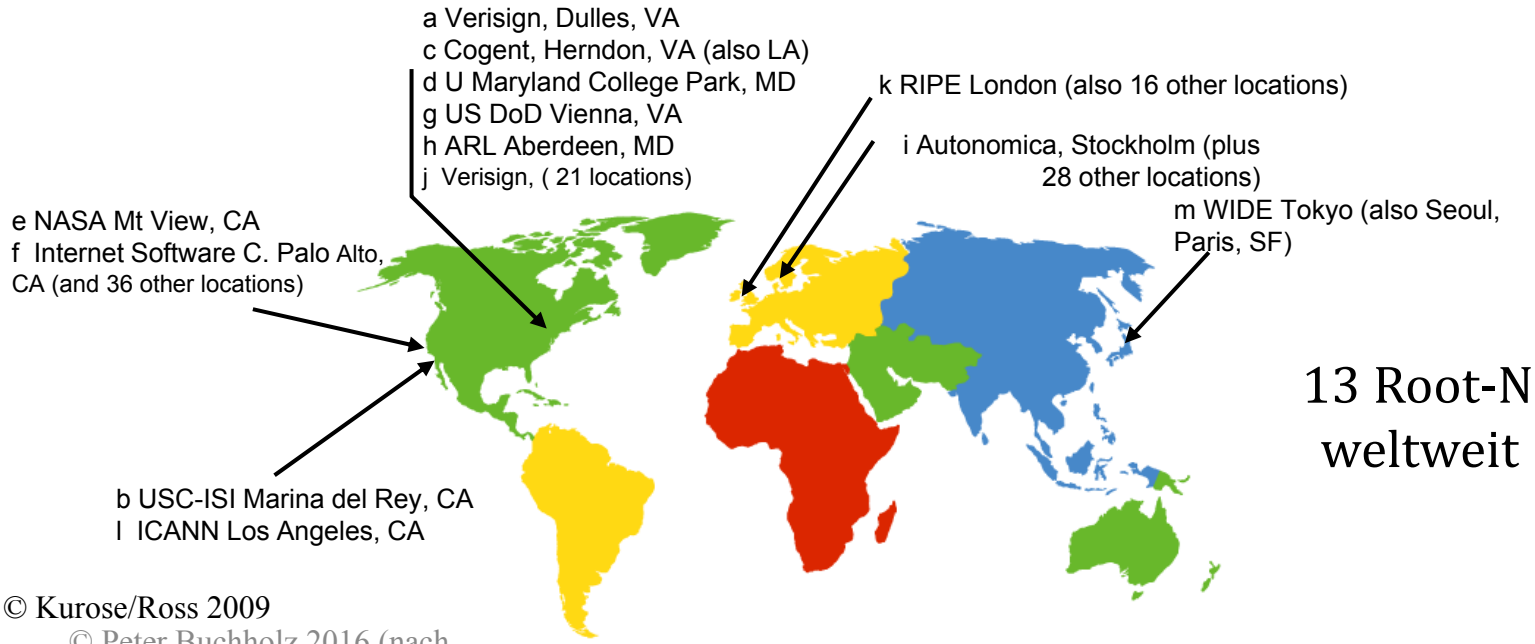
- Einige wenige meist in den USA, empfangen Anfragen von lokalen Name-Servern

Autoritative Name-Server

- Jeder Host gehört zu einem autorativen Name-Server, der seine Adresse 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
  - fragen den zugehörigen autorativen Name-Server, falls sie die Adresse nicht kennen
  - erhalten von diesem die Abbildung
  - leiten diese an den lokalen Name-Server weiter

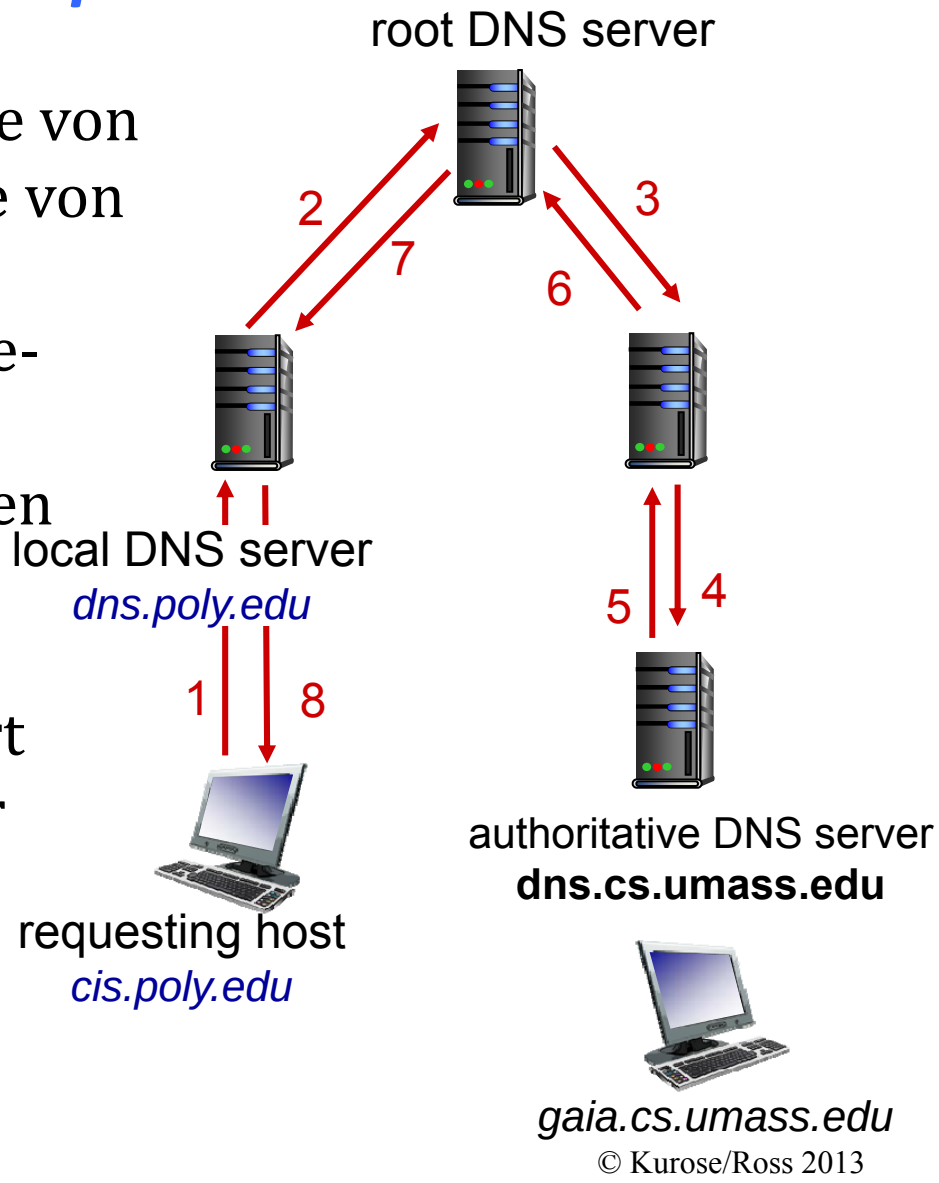


13 Root-Name-Server weltweit

DNS: *Einfaches Beispiel*

Ablauf einer (rekursiven) Anfrage von surf.eurecom.fr nach der Adresse von gaia.cs.umass.edu

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



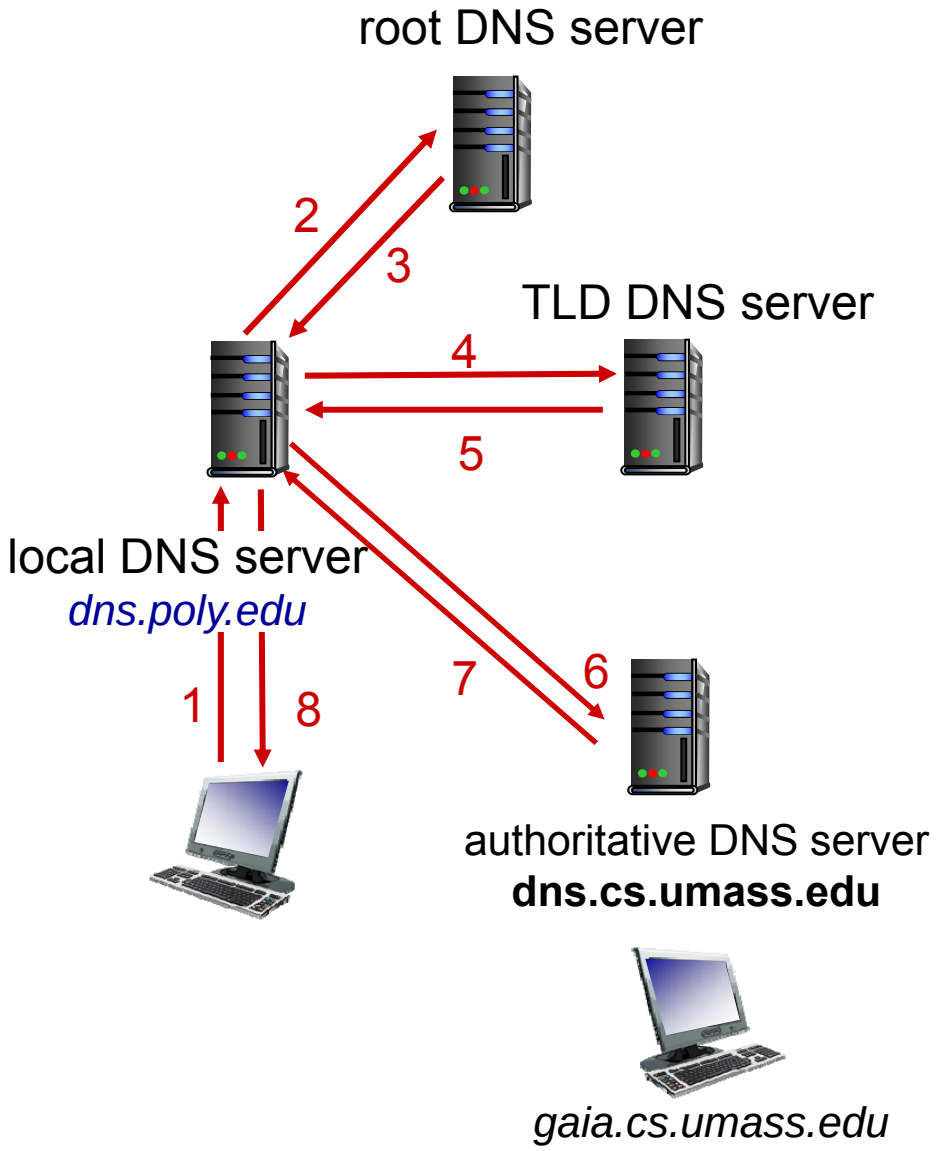
DNS: Wiederholte Anfragen

Rekursive Anfrage:

- Angefragter 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 = Hostname, Wert = IP-Adresse,
    Typ NS Name = Domain und Wert = IP-Adresse des zugehörigen autorativen Servers
    Typ CNAME Name = Alias-Name und Wert = der vollständige Name
    Typ MX Name = Alias-Name für einen Mail-Server und Wert = der vollständige Name
  - TTL (time to live)

DNS: Protokolldateneinheiten

DNS-Protokoll: Anfrage- und Antwort-Nachrichten mit identischem Format

Header:

- Identifikation einer Anfrage durch 16 Bit Nummer
- Antwort auf die Anfrage nutzt die selben 16 Bit
- Flags
  - Anfrage oder Antwort
  - Antwort oder autorativer Name-Server
  - Rekursion gewünscht
  - Rekursion wird unterstützt

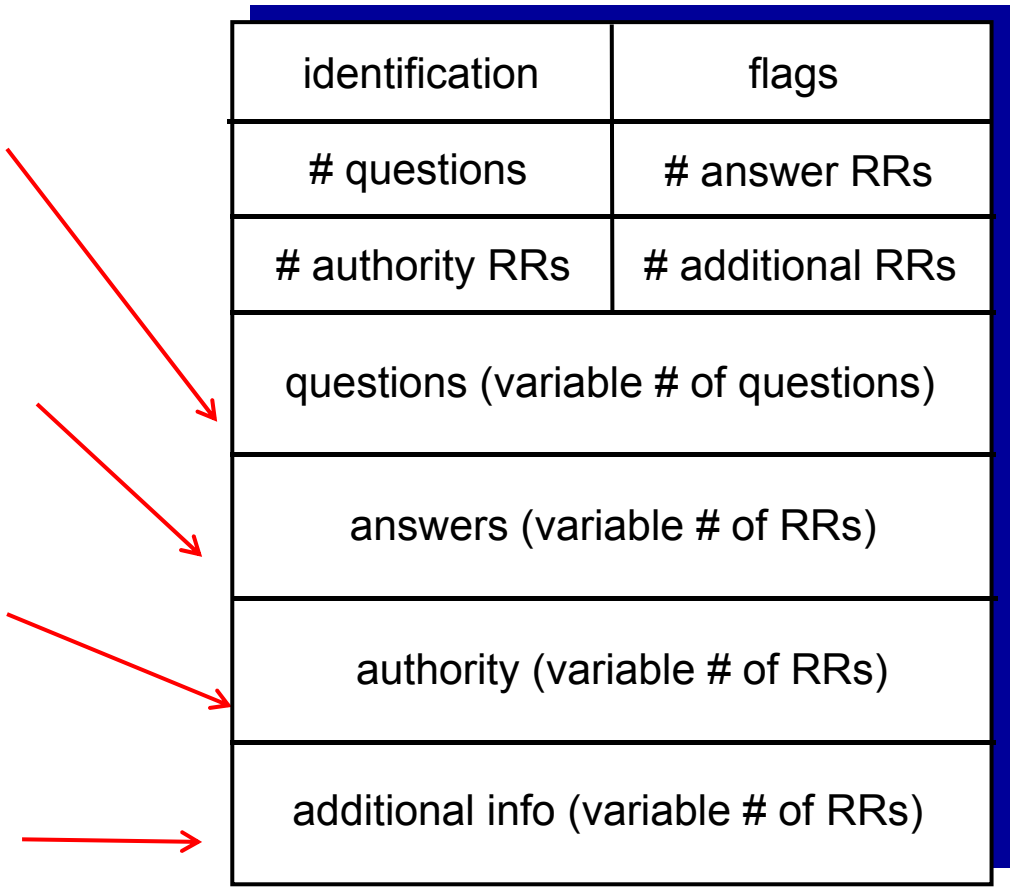
DNS: Protokolldateneinheiten

Name, Typ-Feld für eine Anfrage

Ressource-Records als Antwort der Anfrage

Records anderer autorativer Server

Zusatzinformationen



Peer-to-Peer Anwendungen

Alternative zu Client/Server Anwendungen
Peer-to-Peer Anwendungen

Peers

- werden von einzelnen Rechnern betrieben
- sind alle gleichwertig (und damit Client und Server)
- unterstehen der Kontrolle des Nutzers und sind damit autonom
- kommunizieren direkt miteinander

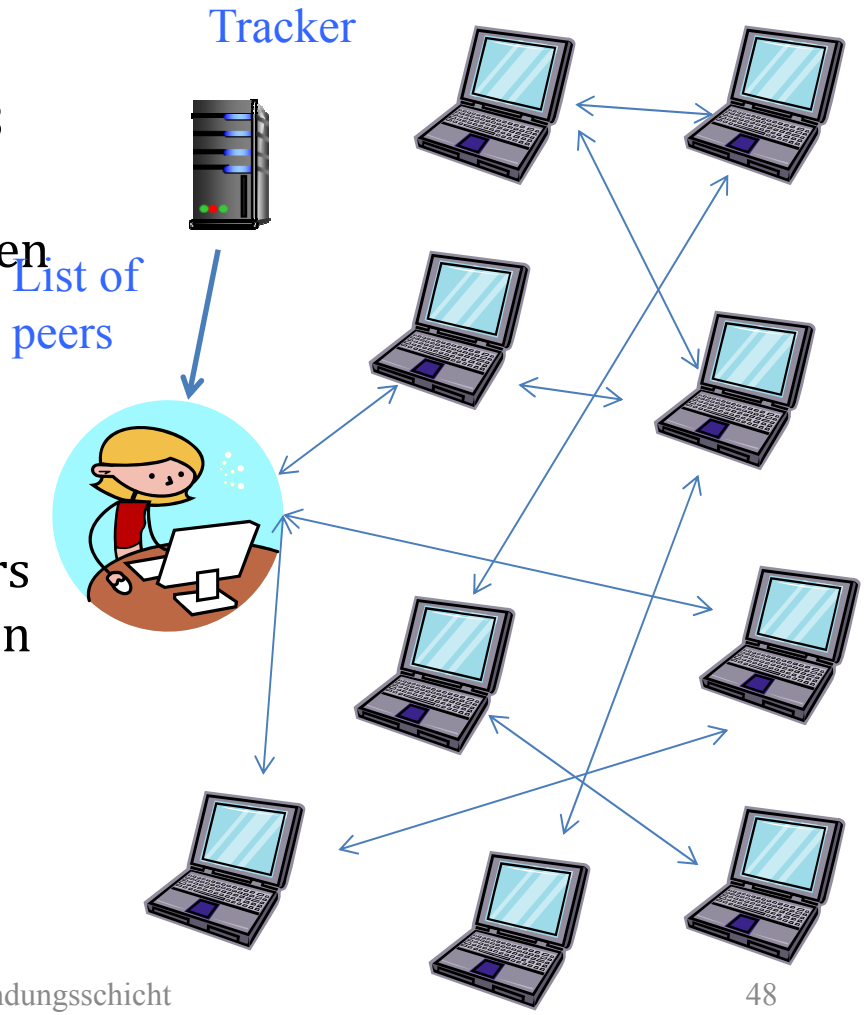
P2P-Systeme

- haben keine zentrale Kontrollinstanz
- sind dynamisch (Peers kommen und gehen)
- beinhalten keine Übersicht über verfügbare Inhalte oder Dienste

Peer-To-Peer: Verteilte Dateisysteme

Verteilung großer Dateien in einem P2P-System Basisprotokoll BitTorrent

- Menge der Teilnehmer, die eine Datei speichern nennt man *torrent*
- Datei wird in *chunks* der Größe 256 KB zerlegt
- Torrent enthält einen Infrastruturknoten den *tracker*
- Tracker versorgt Peers mit Informationen über andere Peers
- Peers tauschen chunks aus und kommunizieren dazu mit anderen Peers
- laden möglichst chunks, die bei anderen peers in der Nachbarschaft selten sind
- senden chunks zu anderen peers proportional zu deren Senderate zum peer



Peer-To-Peer: Informationssuche

Index mit Informationen muss dynamisch erstellt werden

- Teilnehmer und damit Inhalte wechseln

Zentraler Index:

- Zentraler Server speichert Inhalte
- Peers informieren Server über Inhalte
- Peers fragen beim Server nach Inhalten und bekommen die IP-Adresse eines Anbieters
- Peers treten dann direkt in Verbindung mit dem Anbieter
- Bsp. Napster

Peer-To-Peer: Informationssuche

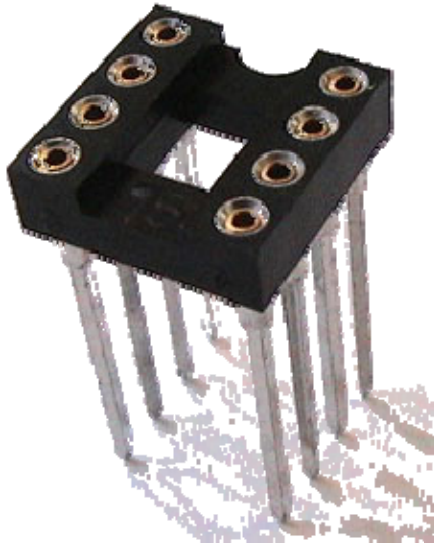
Flutung von Anfragen

- Peers bilden ein virtuelles Overlay-Netz
- Anfragen werden an Nachbarn verschickt und von dort weiter-gereicht (i.d.R. nur über eine maximale Anzahl von Knoten)
- Hat ein Peer die geforderte Information, so antwortet er auf dem selben Pfad
- Initiator kann dann direkt die Information bei einem Peer anfordern
- Bsp. erste Versionen von Gnutella

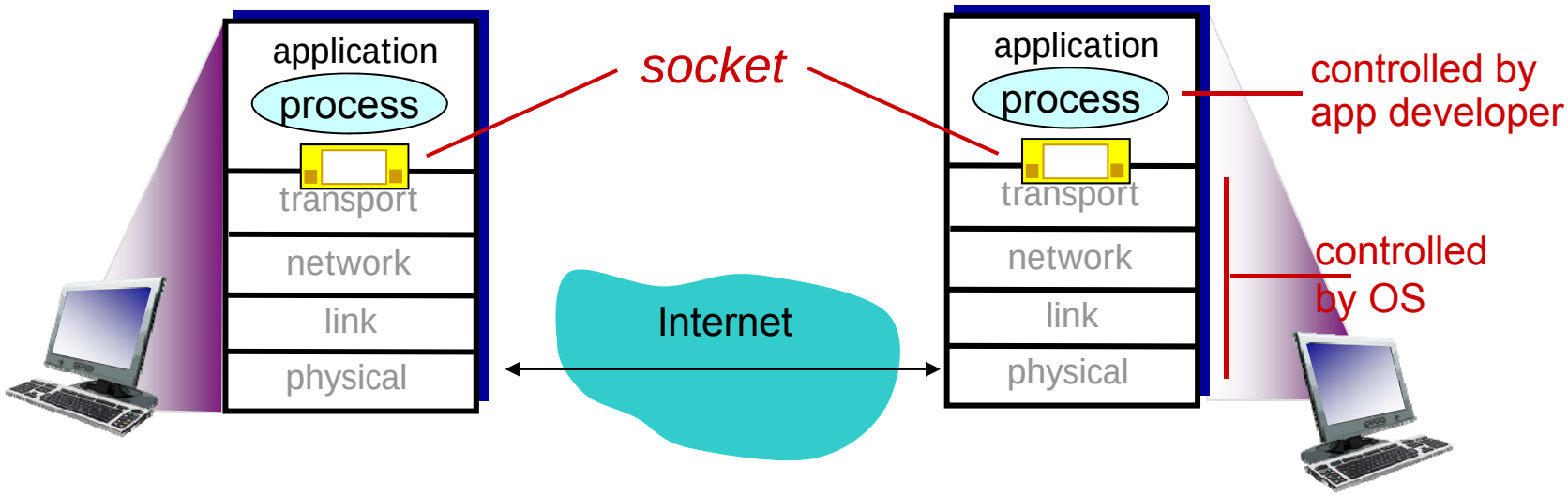
Socket-Programmierung: Kommunikations-API

Socket API

- Eingeführt für BSD4.1 UNIX, 1981
- Sockets werden von Anwendungsprogrammen erzeugt, genutzt und abschließend freigegeben
- Client/Server Ansatz
- API: erlaubt die Wahl eines Transportprotokolls und das Setzen einiger Parameter, z.B.
  - Unzuverlässige Datagramme
  - Zuverlässige Bitströme



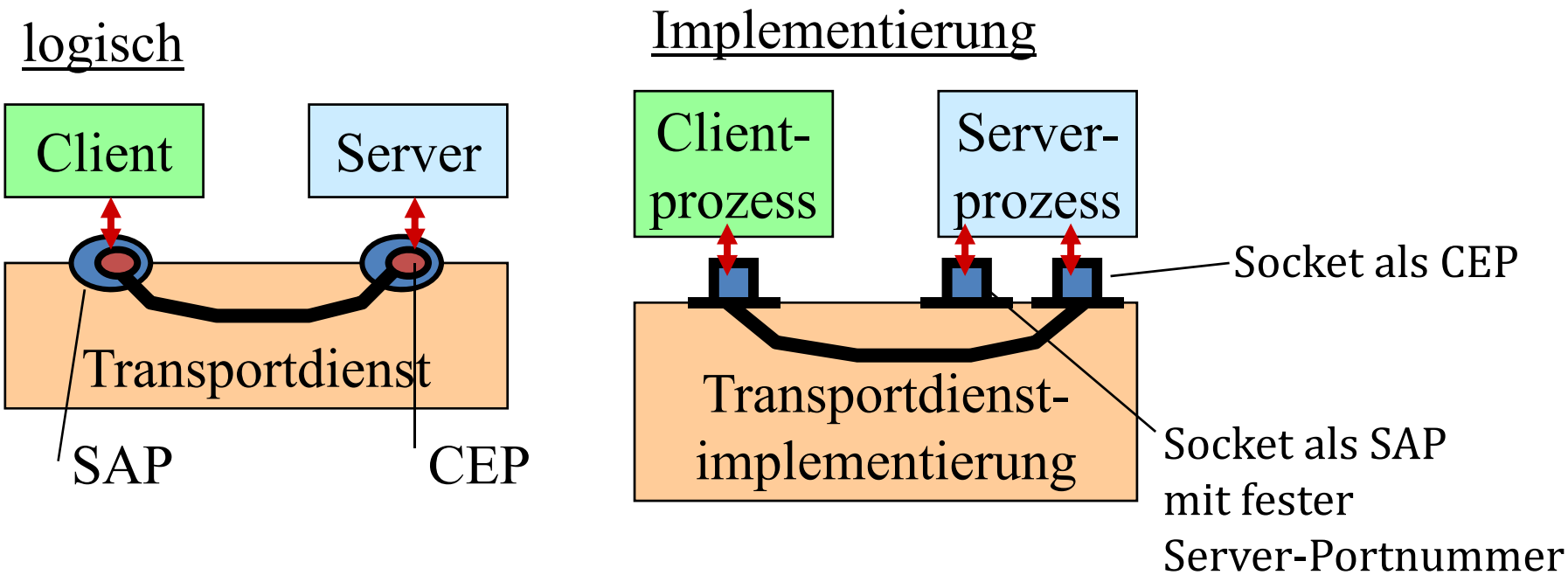
Sockets



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 eine Dateideskriptornummer

TCP-Sockets: Client und Server

Client nimmt mit dem Server Kontakt auf

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

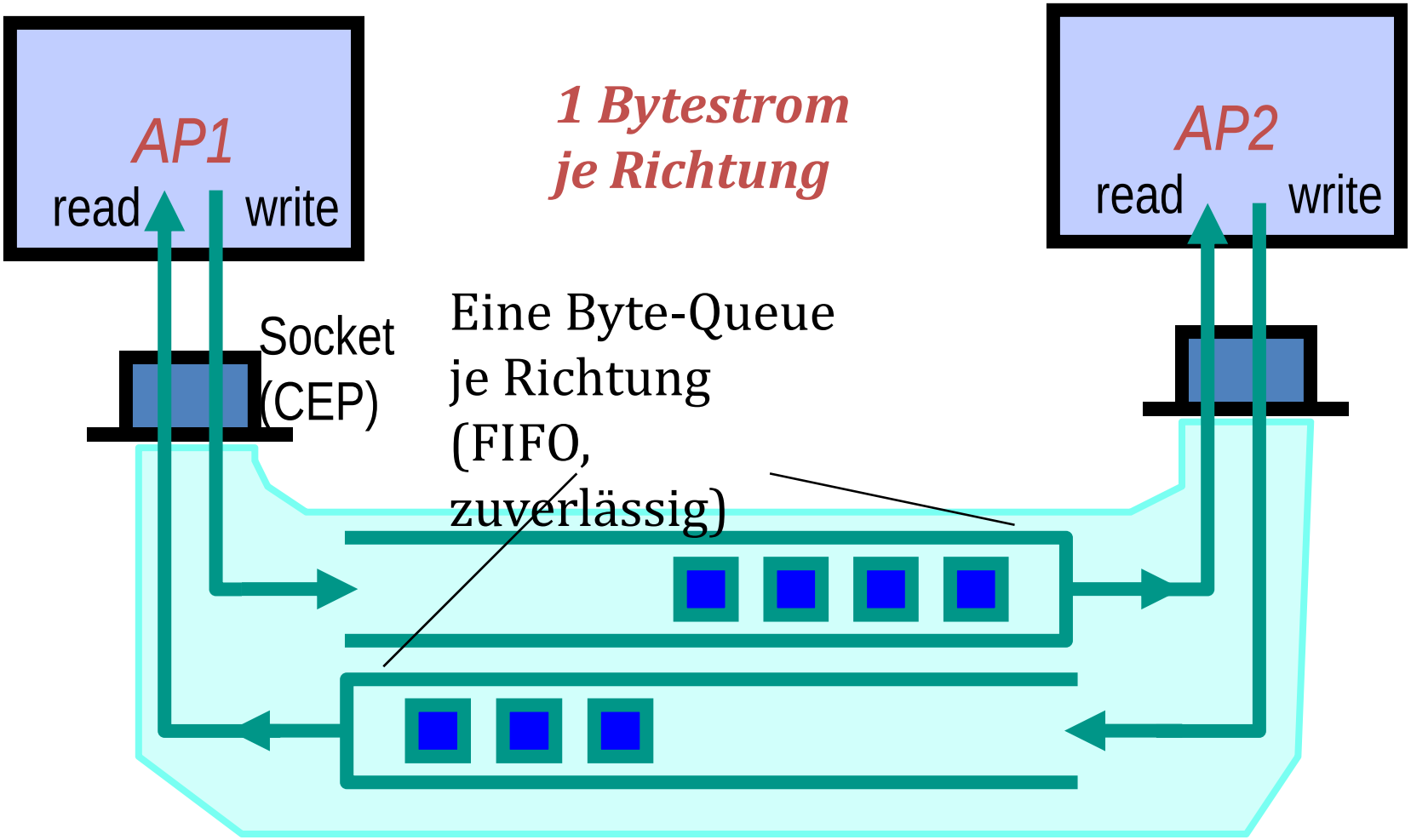
Client

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

- Client etabliert durch den Aufruf eine TCP-Verbindung
- Nach der Kontaktaufnahme generiert der Server einen neuen Socket zur Kommunikation mit dem Client (d.h. Server kann mit mehreren Clients kommunizieren)

Anwendungssicht:
TCP bietet eine zuverlässige, reihenfolgeerhaltende Verbindung

TCP-Sockets: Anwendungssicht auf TCP-Verbindung



ganz offene TCP-Verbindung:
beide Richtungen offen

TCP-Socket - API

Auf Client-Seite:

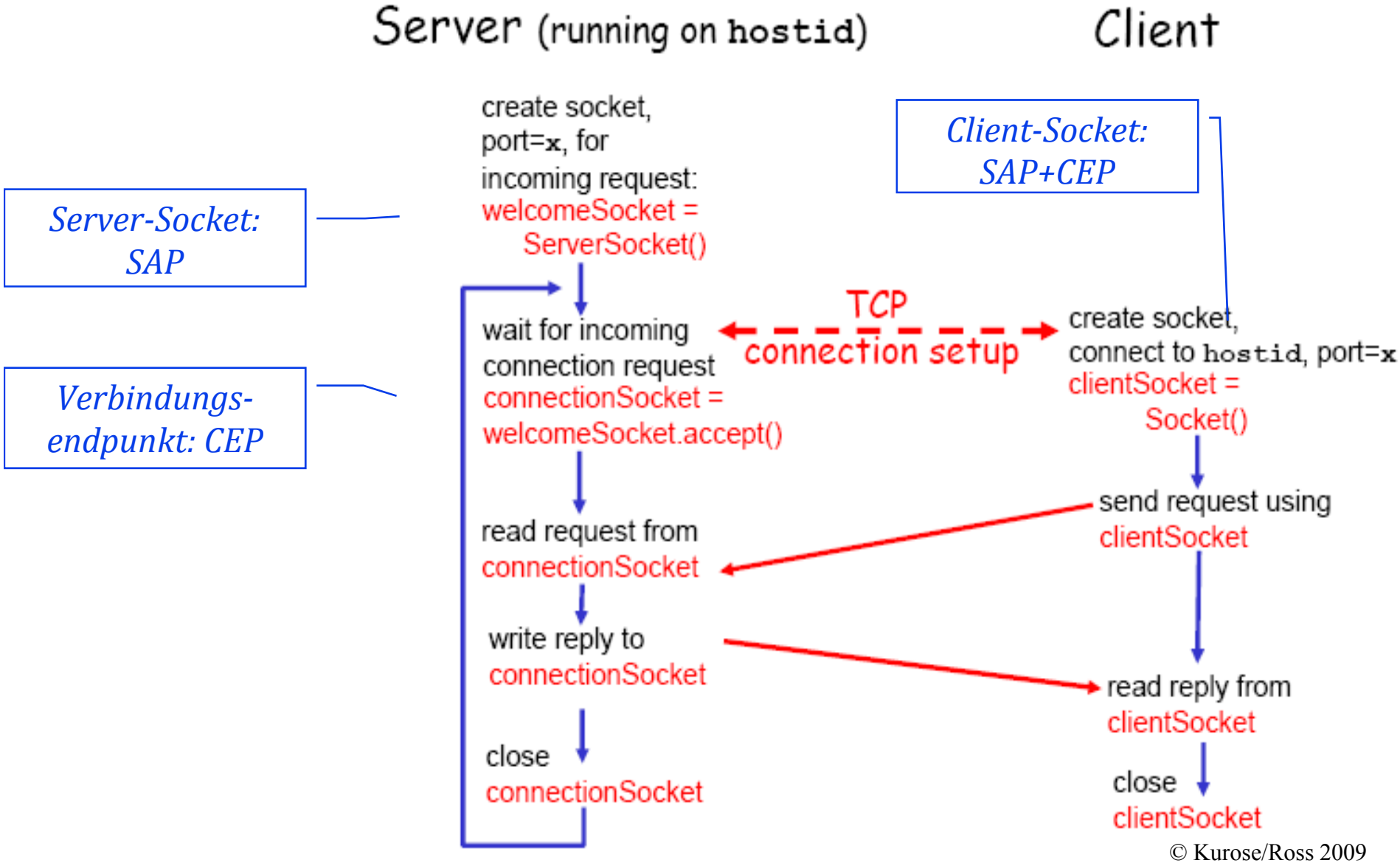
- socket(), liefert die Socket-ID für den Client
- connect(), sendet eine conn.-req. an die angegebene IP-Adresse
- send(), sendet an den Client-Socket
- recv(), empfängt vom Client-Socket
- close(), schließt die Verbindung

BS liefert lokale IP-Adresse und Port für einen Client

Auf Server-Seite:

- socket(), liefert die Socket-ID für den Server
- bind(), bindet den Server-Socket an die IP-Adresse und Port-Nummer des Servers
- listen(), warten auf eine Anfrage am Server-Socket
- accept(), akzeptiert eine neue Verbindung
- send(), sendet an den Server-Socket
- recv(), empfängt vom Server-Socket
- close(), schließt die Verbindung

Client/Server: TCP - Interaktion



© Kurose/Ross 2009

UDP-Sockets

UDP realisiert keine Verbindung zwischen Client und Server

- Kein handshake
- Sender muss IP-Adresse und Port-Nummer des Empfängers bereithalten
- Server erfährt die IP-Adresse und Port-Nummer des Clients auf dem empfangenen Datagramm

Daten via UDP können in falscher Reihenfolge oder gar nicht empfangen werden

Anwendungssicht:
UDP bietet eine unzuverlässige Verbindung

UDP-Socket - API

Auf Client-Seite:

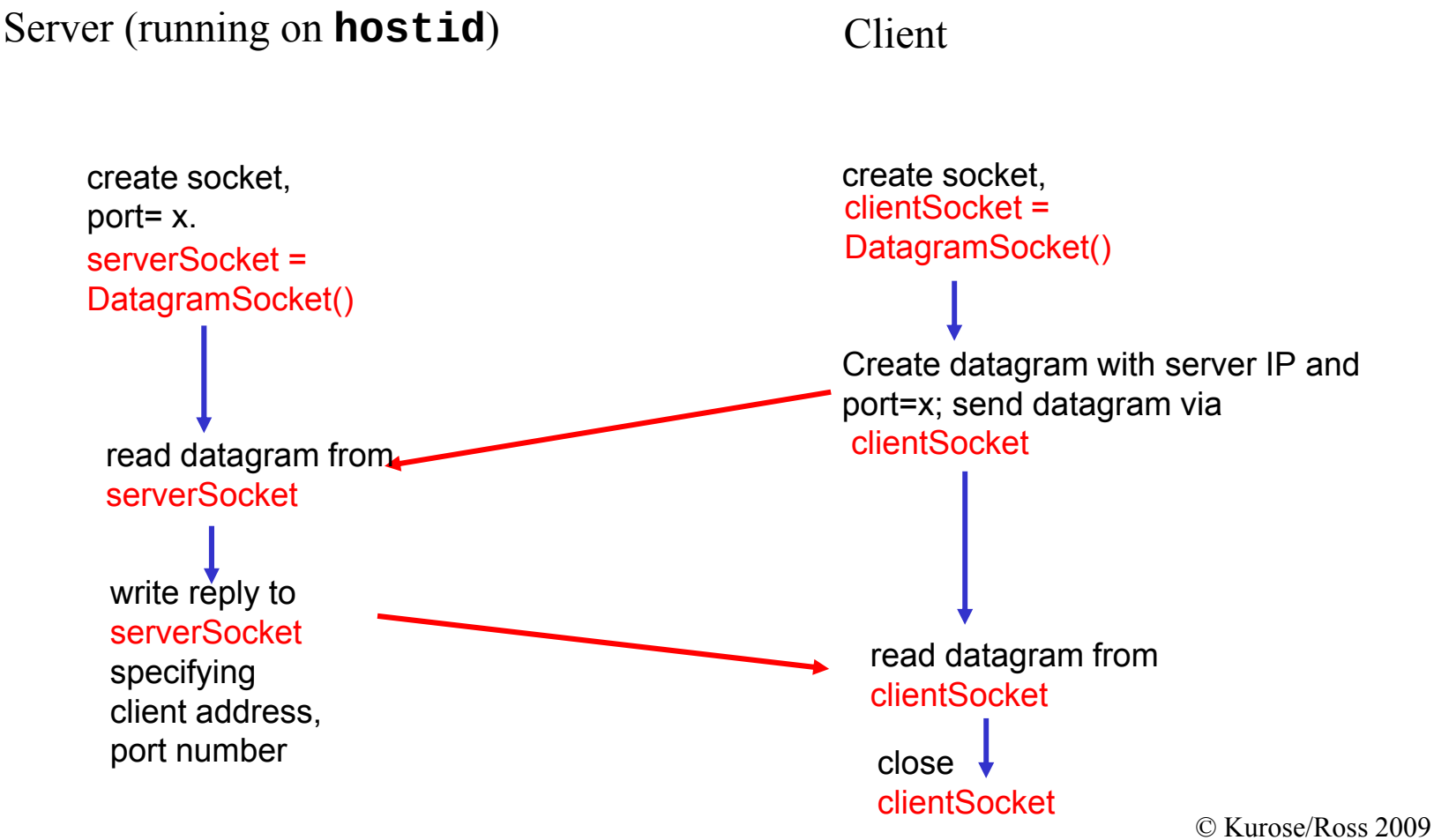
- socket(), liefert die Socket-ID für den Client
- sendto(), sendet an den Client-Socket, IP-Adresse und Port-Nummer sind nötig
- recvfrom(), empfängt vom Client-Socket inkl. IP-Adresse und Port-Nummer des Senders
- bind(), bindet den Socket an die IP-Adresse und Port-Nummer (optional)

Auf Server-Seite:

- socket(), liefert die Socket-ID für den Server
- bind(), bindet den Server-Socket an die IP-Adresse und Port-Nummer des Servers
- sendto(), sendet an den Server-Socket, IP-Adresse und Port-Nummer sind nötig
- recvfrom(), empfängt vom Server-Socket inkl. IP-Adresse und Port-Nummer des Senders

Timeout-Mechanismus ist notwendig

Client/Server: UDP - Interaktion



© Kurose/Ross 2009