

Rechnernetze

Kapitel 7: Application Layer

Prof. Dr. Wolfgang Mühlbauer

Fakultät für Informatik

`wolfgang.muehlbauer@th-rosenheim.de`

Wintersemester 2019/2020

Slides are based on:

J. Kurose, K. Ross: Computer Networks – A Top-Down Approach

A. Tanenbaum, D. Wetherall: Computer Networks

Application Layer

□ Typische **Aufgaben**

- **Typen** der ausgetauschten Nachrichten
 - Beispiel: Anfrage („Request“) und Antwort („Response“)
- **Syntax** der Nachrichten
 - Welche Nachrichtfelder gibt es und wie sind diese in der Nachricht angeordnet?
- **Semantik** der Nachrichten
 - Bedeutung der Information in den einzelnen Feldern
- **Regeln** wann/ wie Prozesse Nachrichten senden bzw. auf diese antworten
 - Zustandsautomat

□ Offene Protokolle

- Definiert in "Request for Comments" (RFCs)
- Beispiele: HTTP, SMTP

□ Proprietäre Protokolle

- Beispiel: Skype

❑ **TCP und UDP**

- Keine Verschlüsselung
- Passwörter, die als Klartext über Socket gesendet werden, sind überall im Internet sichtbar.

❑ **TLS / SSL**

- Verschlüsselung für TCP Verbindungen, verfügbar für alle TCP Anwendungen
- Datenintegrität
- Authentifizierung der Endpunkte
- Prominentes Beispiel: HTTPS
- TLS/SSL ist aus Rechnernetze-Sicht ein Protokoll der Anwendungsschicht

- ❑ **Web und HTTP**
- ❑ DNS
- ❑ Ausblick: Sockets

Hypertext Transfer Protokoll (HTTP)

- Webseite / Webobjekt adressierbar durch **Uniform Resource Locator (URL)**

- <http://www.phdcomics.com/comics.php>

Protokoll Server Ressource auf Server

- **HTTP Client**

- Baut TCP Verbindung zu passendem Port auf, bei HTTP (fast immer) Port 80
- Senden des HTTP Requests und Auswerten der Antwort
- Ggfs. Nachladen weiterer URLs (z.B. Bilder), um die Seite anzuzeigen.
- Schließen der TCP Verbindung

- **HTTP Server**

- Akzeptieren von ankommenden TCP Verbindungen
- Abbilden von ankommenden Anfragen auf Ressourcen (z.B. Datei)
- Laden der Ressource und Senden an den Client.
- Server ist „**stateless**“: Er merkt sich nichts bezüglich früherer Anfragen des Clients

Webseiten

- ❑ Webseite besteht aus mehreren **Objekten**, z.B.
 - HTML Datei
 - JPEG Image
 - Audio Datei
 - Javascript
 - ...
- ❑ Meist zentraler Einstieg in Webseite, z.B.
 - `index.html` referenziert nachzuladende Objekte der Webseite.
 - `php.ini`: Falls Webseite PHP Code enthält.
- ❑ **MIME** gibt Information über den Typ des Inhalts
 - Beispiel: `text/html`, `image/jpeg`, `audio/mpeg`
 - Bei MIME-Type `text/html`: Direktes Anzeigen der Seite durch Browser
- ❑ Unterscheidung zwischen
 - **statischen** Webseiten, die für jeden Benutzer gleich aussehen.
 - **dynamischen** Webseiten, die bei jedem Aufruf entweder vom Client oder vom Server automatisch generiert werden.

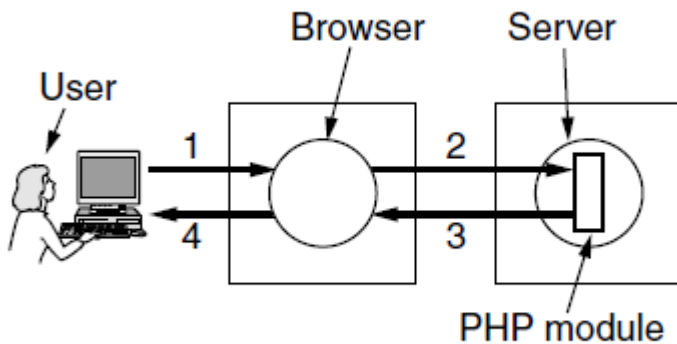
Statische Webseiten / HTML Versionen

Item	HTML 1.0	HTML 2.0	HTML 3.0	HTML 4.0	HTML 5.0
Hyperlinks	X	X	X	X	X
Images	X	X	X	X	X
Lists	X	X	X	X	X
Active maps & images		X	X	X	X
Forms		X	X	X	X
Equations			X	X	X
Toolbars			X	X	X
Tables			X	X	X
Accessibility features				X	X
Object embedding				X	X
Style sheets				X	X
Scripting				X	X
Video and audio					X
Inline vector graphics					X
XML representation					X
Background threads					X
Browser storage					X
Drawing canvas					X

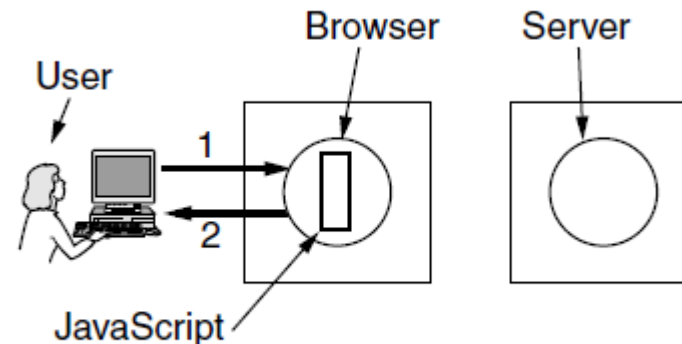
Quelle: Tanenbaum, Computer Networks

Dynamische Webseiten

- Sehen für jeden Benutzer anders aus.
- Können durch Client oder durch Server generiert werden.
 - **Server-Side:** PHP, CGI, usw. erlauben es z.B. durch "HTML Forms" übertragene Parameter auf dem Server auszuwerten und dann die HTML Seite zu erzeugen und zurückzuliefern.
 - **Client-Side:** JavaScript, VBScript. Innerhalb von <script> kann in eine HTML Seite Code eingebettet werden, der dann im Browser ausgeführt wird.



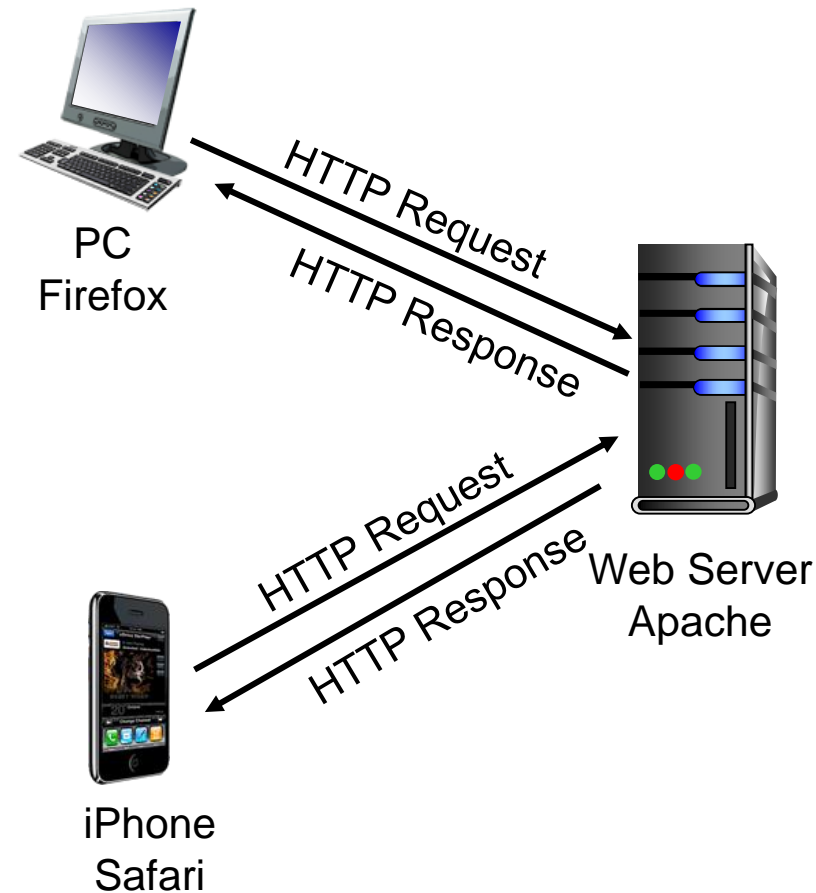
Server-side scripting mit PHP



Client-side Scripting mit JavaScript

Hypertext Transfer Protokoll (HTTP)

- HTTP: Application Layer Protokoll des Webs
 - Nicht verwechseln mit HTML!
 - Überträgt z.B. HTML-Dokumente
- **Request-Response-Protokoll** auf der Basis von TCP
- Client-Server Prinzip
 - **Client:** Browser, der Webseiten anfordert („Request), empfängt und die Web-Objekte darstellt
 - **Server:** Web Server sendet über HTTP die Web-Objekte, die angefragt wurden



HTTP Request Nachricht

- ❑ Verschiedene Request-Typen
- ❑ ASCII, Textformat

Request Zeile
(GET, POST,
HEAD commands)

Header-Zeilen

Carriage Return,
Line Feed am Zeilen-
anfang bedeutet Ende
des Headers

Method	Description
GET	Read a Web page
HEAD	Read a Web page's header
POST	Append to a Web page
PUT	Store a Web page
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Connect through a proxy
OPTIONS	Query options for a page

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n
```

Publikums-Joker: HTTP

Welche der folgenden Aussagen ist **falsch**?

- A. HTTP benötigt zwingend eine TCP Verbindung.
- B. Der HTTP-Header ist "human-readable", kein Bytecode
- C. HTTP überträgt stets HTML-Dateien.
- D. HTTP arbeitet nach dem GET-RESPONSE Prinzip.



HTTP Response – Status Codes

- ❑ Status Code erscheint in der 1. Zeile der Nachricht vom Server an den Client

- ❑ **200 OK**

- Request war erfolgreich

- ❑ **301 Moved Permanently**

- Das angeforderte Objekt wurde verschoben; der neue Ort wird in der Nutzlast spezifiziert

- ❑ **400 Bad Request**

- Die Anfrage wurde vom Server nicht verstanden

- ❑ **404 Not Found**

- Das angeforderte Dokument wurde auf dem Server nicht gefunden

- ❑ **505 HTTP Version not supported**

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

Nutzlast



HTTP Upload

❑ POST Methode

- Webseiten enthalten oft HTML Formulare und Eingabefelder
- Die Eingabe wird in der HTML Nutzlast einer POST Nachricht übertragen.

❑ URL Methode

- Verwendet eine GET Nachricht
- Die Eingabe wird im URL Feld der Nachricht hochgeladen
- Beispiel: **`www.somesite.com/animalsearch?monkeys&banana`**

HTTP Verbesserungen

❑ Non-persistent versus persistent HTTP

○ *Non-persistent*

- Höchstens 1 Webobjekt wird über 1 TCP Verbindung gesendet, Verbindung wird dann geschlossen
- Folge: Unter Umständen mehrere TCP Verbindungen für 1 Webseite notwendig
- 2RTTs pro Objekt, großer Overhead

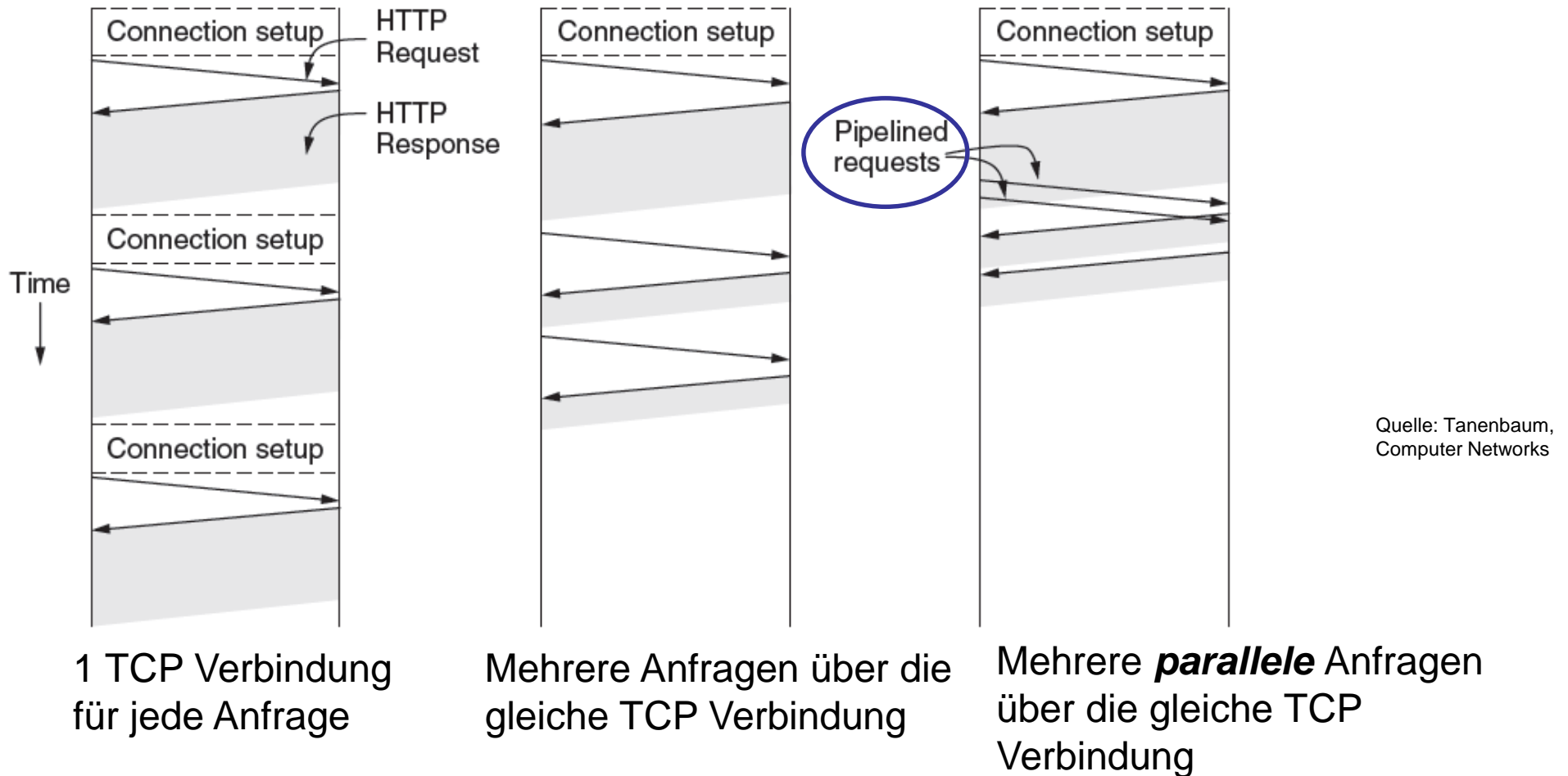
○ *Persistent*

- Mehrere Webobjekte können über gleiche TCP Verbindung gesendet werden
- Default-Einstellung in den meisten Webbrowsern

❑ Pipelining

- Webbrowser stellen neue Anfrage, ohne Antwort auf vorherige Anfrage abzuwarten.
- Häufig sogar über mehrere parallele TCP Verbindungen

Persistent HTTP and HTTP Pipelining



□ Ziel

- HTTP ist "stateless", d. h. es vergisst sofort die letzte Anfrage.
- Wie erkennt Webserver Benutzer beim nächsten Besuch wieder?
- Speichern von Zustand über HTTP Sessions hinweg.

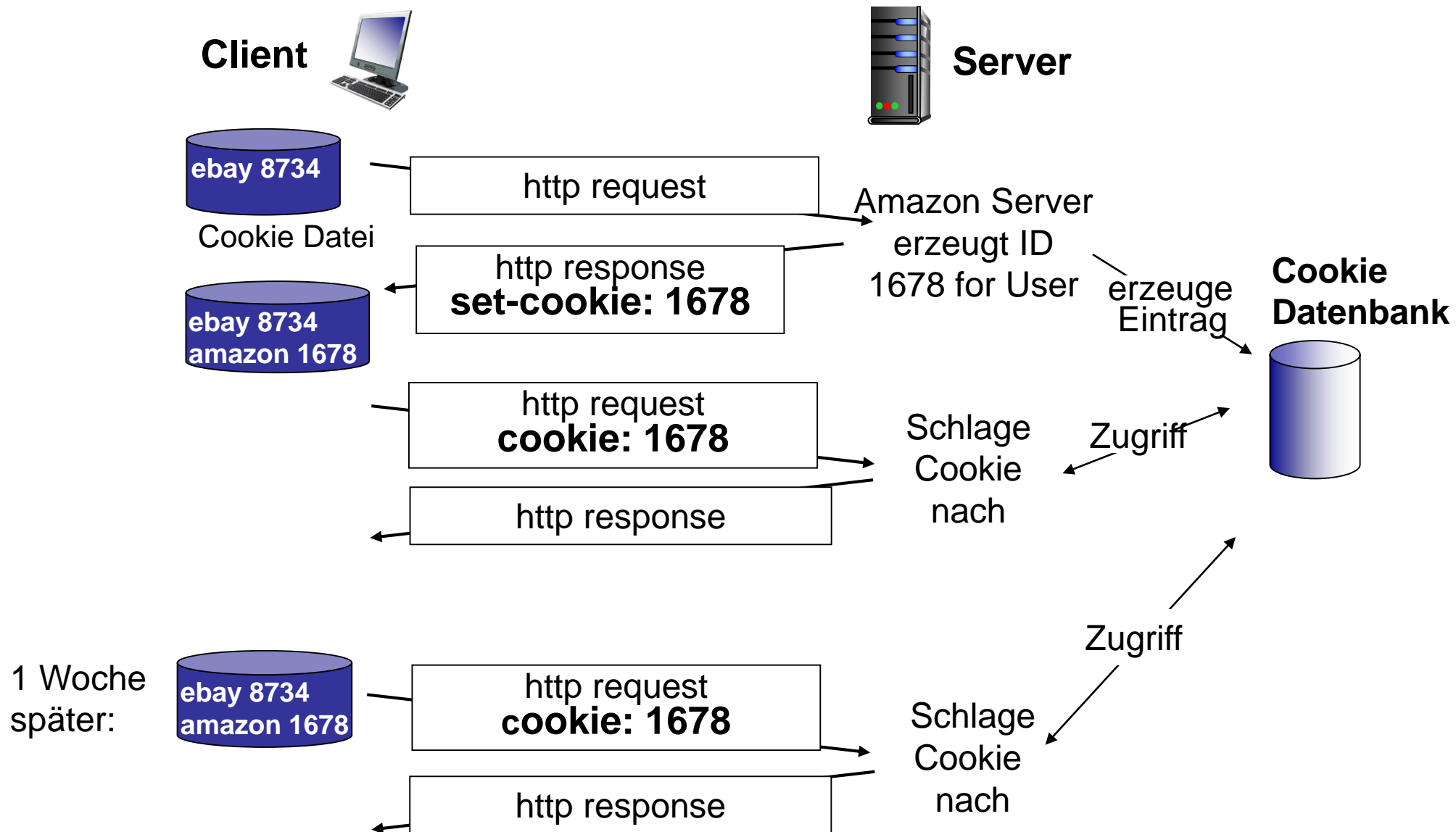
□ Einsatz von Cookies

- Identifikation von Web-Benutzern
- Einkaufswagen in e-Shops, Kaufempfehlungen
- Speichern von Session State z.B. bei Webmail

□ Cookies: 4 Komponenten

- Headerzeile in der HTTP Response Nachricht
- Headerzeile in der HTTP Request Nachricht
- Datei auf dem Computer des Nutzers, verwaltet durch Browser
- Cookie Datenbank auf Webserver

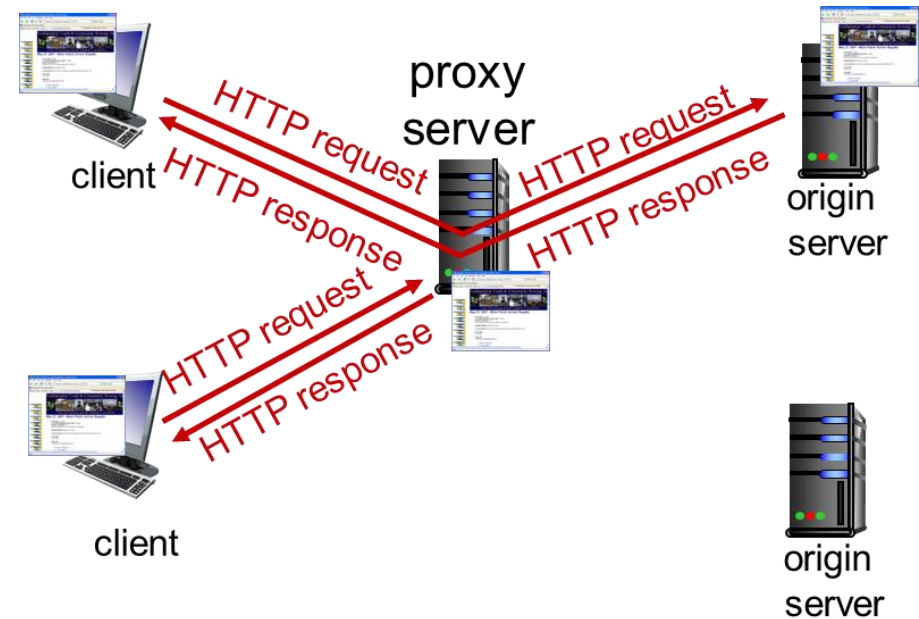
Cookies



Web Caching

□ Ziele

- Kürzere Antwortzeiten für einen Web Request
- Einsparen von Traffic auf Link zu Provider



- Konfiguration eines Proxy Server im Web Browser
- Browser sendet alle Anfragen an einen Proxy/Cache
 - Falls Objekt in Cache: Cache liefert Objekt zurück
 - Andernfalls: Proxy stellt Request an Original-Server, speichert Objekt in Cache und beantwortet Anfrage

Web Caching: Bedingtes GET

❑ Ziel

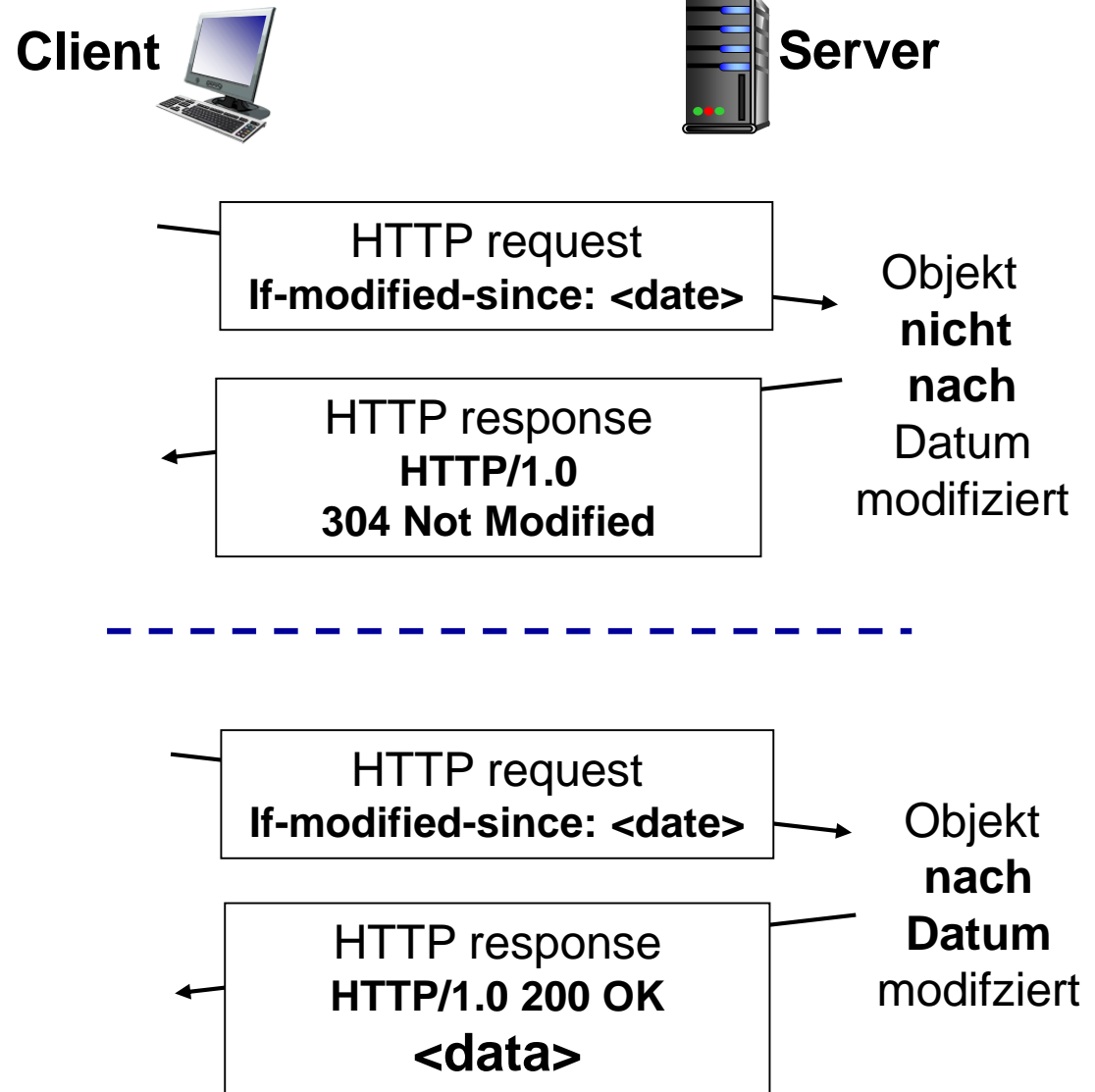
- Server liefert Objekt nur aus, falls Cache eine ältere Version hat.

❑ Client

- Spezifiziert Datum der zwischengespeicherten Kopie im HTTP Request

❑ Server

- Antwort enthält kein Objekt falls zwischengespeicherte Kopie aktuell ist.



Inhalt

- ❑ Web und HTTP
- ❑ **DNS**
- ❑ Ausblick: Sockets

Domain Name System: Aufgaben

❑ Übersetzung Hostname → IP Adresse

- Name: `th-rosenheim.de` (einfach zu merken)
- IP Adresse: `141.60.160.196` (lesbar für Maschinen)

❑ Weitere Aufgaben

- **Host Aliasing:** Host kann mehrere Namen haben, Übersetzung
 - *Canonical Name:* `relay1.west-coast.enterprise.com`
 - *Alias Name:* www.enterprise.com
- **Mail Server Aliasing**
 - Finde Mailserver für eine Domain.
 - *MX Record:* Speichert *Canonical* Name des Mail Server für eine Domain.
- **Load Balancing**
 - Replizierte Web Server: Viele IP Adressen haben gleichen Namen
 - Antwort des DNS Servers bestimmt, welcher physikalische Server / IP verwendet wird.

Komponenten von DNS

❑ **Verteiltes Verzeichnis**

- Hierarchie von Name Servern

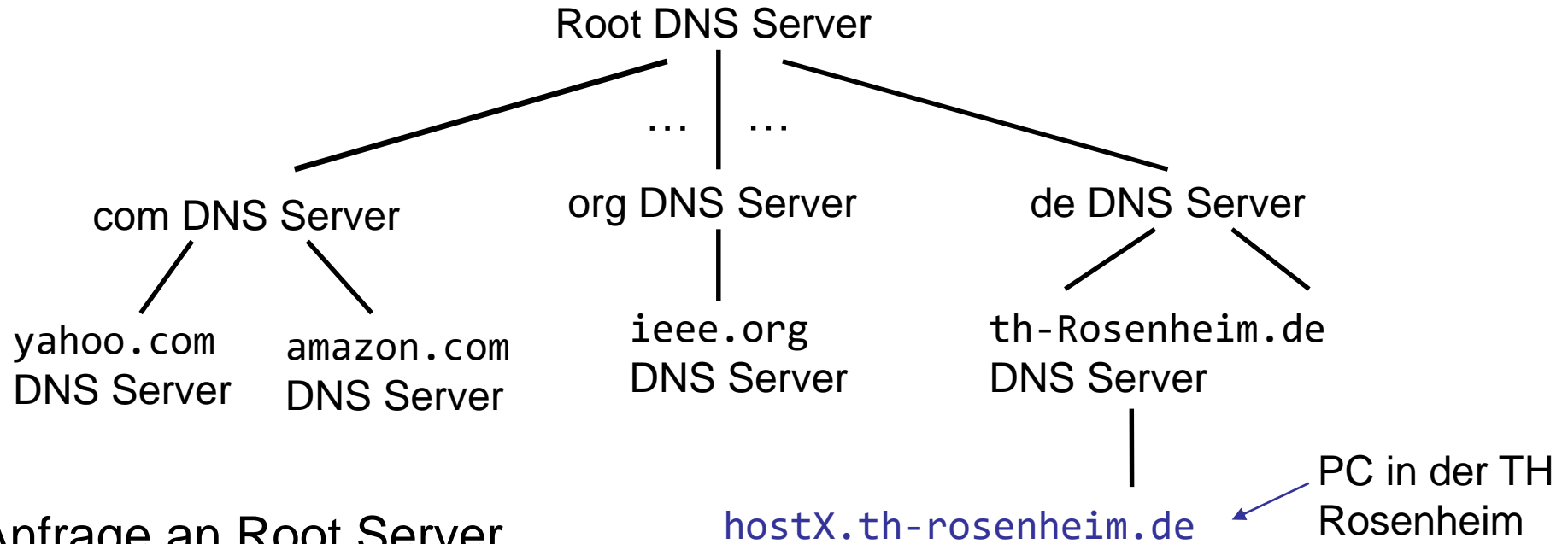
❑ **Protokoll der Application Layer**

- Hosts und DNS Server kommunizieren miteinander.
- Wichtige Internetfunktion wurde in der Application Layer implementiert.
- Prinzip: „Komplexität am Rande des Internets“

❑ **Warum kein zentralisiertes DNS?**

- Single Point of Failure
- Zu hohes Verkehrsvolumen.
- Name Server eventuell sehr weit von anfragendem Host entfernt → hohe Round Trip Time für DNS Anfragen

DNS: Hierarchischer Namensraum



- ❑ Anfrage an Root Server
 - DNS Server von .com?
- ❑ Anfrage an .de DNS Server
 - DNS Server der FH Rosenheim?
- ❑ Anfrage an DNS Server der FH Rosenheim
 - IP Adresse von www.th-rosenheim.de?

Klassifizierung der Name Server

□ Root

- Kennt IPs aller TLD-Nameserver
- de, .org, .net, .com, .edu

□ Top-Level Domain (TLD)

- Kennt IP der **Nameserver**, die zuständig für Domains der rechten Tabelle sind.
- Bsp: .de TLD Server kennt Nameserver für Domain `th-rosenheim.de`

□ Authoritative

- DNS Server einer Domain.
- Kennt IP Adressen von Hosts, seine Auskunft ist verbindlich.
- Beispiel: `eagle.th-rosenheim.de`

□ Resolver

- Stellt DNS Anfragen an Root/TLD/Authoritative
- Stellt Anfragen im Namen von Hosts („Proxy“)
- Speichert keine verbindliche Info, nur Caching!
- Häufig vom ISP bereitgestellt.

Domain	Intended use	Start date	Restricted?
com	Commercial	1985	No
edu	Educational institutions	1985	Yes
gov	Government	1985	Yes
int	International organizations	1988	Yes
mil	Military	1985	Yes
net	Network providers	1985	No
org	Non-profit organizations	1985	No
aero	Air transport	2001	Yes
biz	Businesses	2001	No
coop	Cooperatives	2001	Yes
info	Informational	2002	No
museum	Museums	2002	Yes
name	People	2002	No
pro	Professionals	2002	Yes
cat	Catalan	2005	Yes
jobs	Employment	2005	Yes
mobi	Mobile devices	2005	Yes
tel	Contact details	2005	Yes
travel	Travel industry	2005	Yes
xxx	Sex industry	2010	No

Quelle: Tanenbaum,
Computer Networks

DNS Einträge

- ❑ Verteilte Datenbank, die verschiedene Einträge speichern kann.
- ❑ Einträge == Resource Records (RR)

RR Format: (name, value, type, ttl)

type=A

- **Name:** Hostname
- **Value:** IPv4 address

type=NS

- **Name:** Name, z.B., foo.com
- **Value:** Name des authoritative Nameservers für diese Domain

type=CNAME

- **Name:** Alias-Name
- **Value:** Canonical Name
- **Beispiel:** www.ibm.com ist eigentlich servereast.backup2.ibm.com

type=MX

- **Name:** z.B. foo.com
- **Value:** Name des Mail Servers für diese Domain, z.B. mail.foo.com

Domain Resource Records: Typen

Type	Meaning	Value
SOA	Start of authority	Parameters for this zone
A	IPv4 address of a host	32-Bit integer
AAAA	IPv6 address of a host	128-Bit integer
MX	Mail exchange	Priority, domain willing to accept email
NS	Name server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
SPF	Sender policy framework	Text encoding of mail sending policy
SRV	Service	Host that provides it
TXT	Text	Descriptive ASCII text

Quelle: Tanenbaum,
Computer Networks

Zonendatei: Konfiguration von DNS

; Authoritative data for cs.vu.nl

cs.vu.nl.	86400	IN	SOA	star boss (9527,7200,7200,241920,86400)	Auf diesen Hosts läuft ein Mailserver
cs.vu.nl.	86400	IN	MX	1 zephyr	
cs.vu.nl.	86400	IN	MX	2 top	Nameserver
cs.vu.nl.	86400	IN	NS	star	
star	86400	IN	A	130.37.56.205	IP Adressen einzelner Hosts
zephyr	86400	IN	A	130.37.20.10	
top	86400	IN	A	130.37.20.11	
www	86400	IN	CNAME	star.cs.vu.nl	
ftp	86400	IN	CNAME	zephyr.cs.vu.nl	

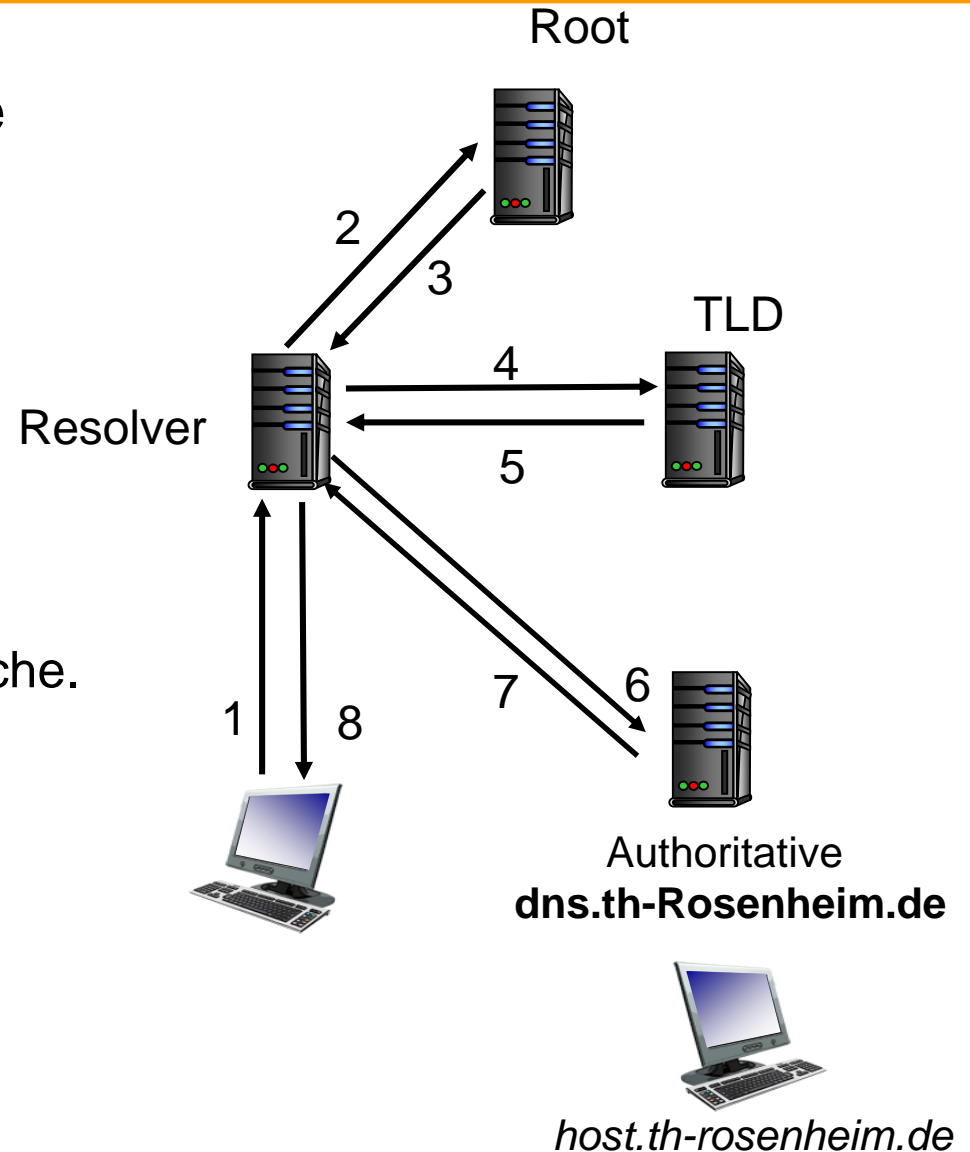
Iterative Namensauflösung

- Ein Host benötigt die IP Adresse von:

- `host.th-rosenheim.de`

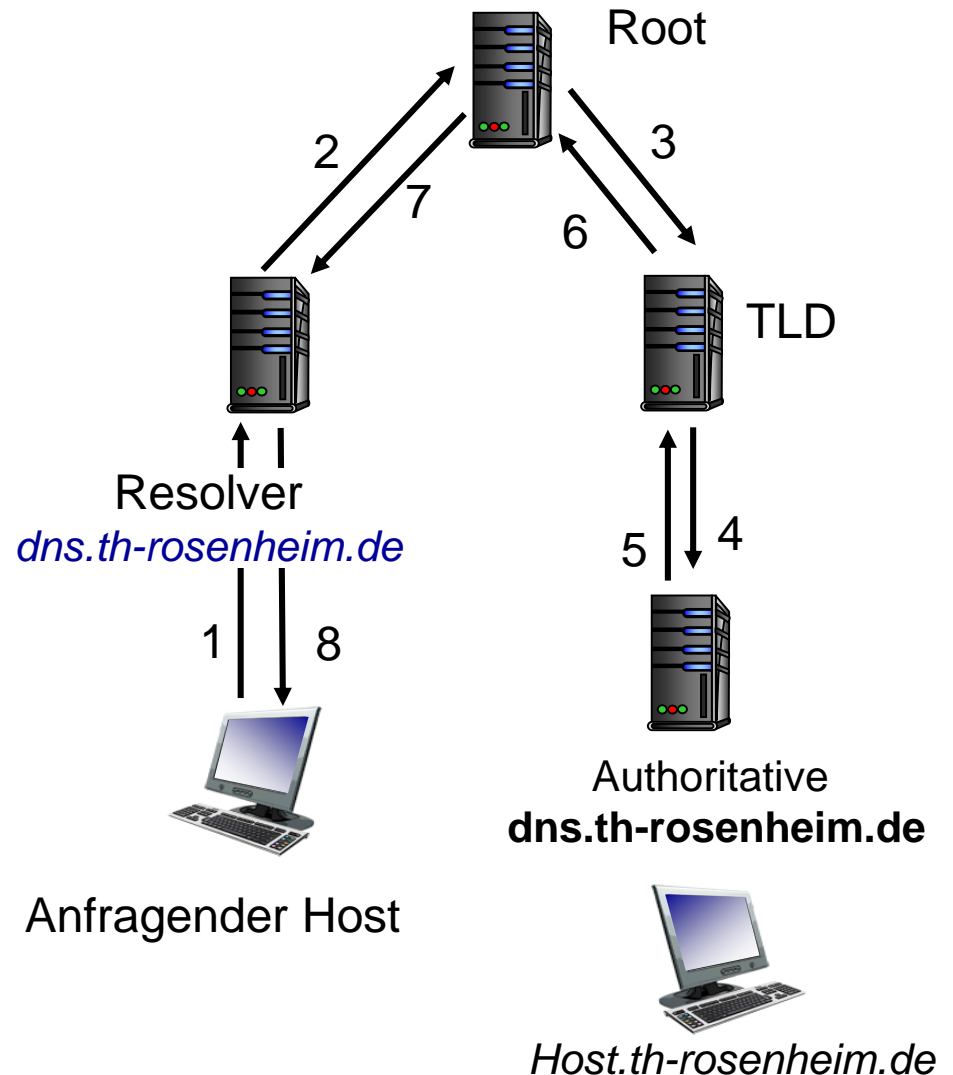
- Iterative Anfrage:

- Resolver hat Antwort nicht im Cache.
- Kontaktierter DNS Server kann Namen nicht auflösen
- Er informiert aber über den „nächsten“ Name Server, der kontaktiert werden muss.
- Nachteil?



Rekursive Namensauflösung

- ❑ Ein Host benötigt die IP Adresse von
 - `host.th-rosenheim.de`
- ❑ Rekursive Anfrage:
 - Jeder Server holt rekursiv die Daten vom nächsten Name Server
 - Hohe Last auf den höheren Hierarchieebenen!



DNS Caching

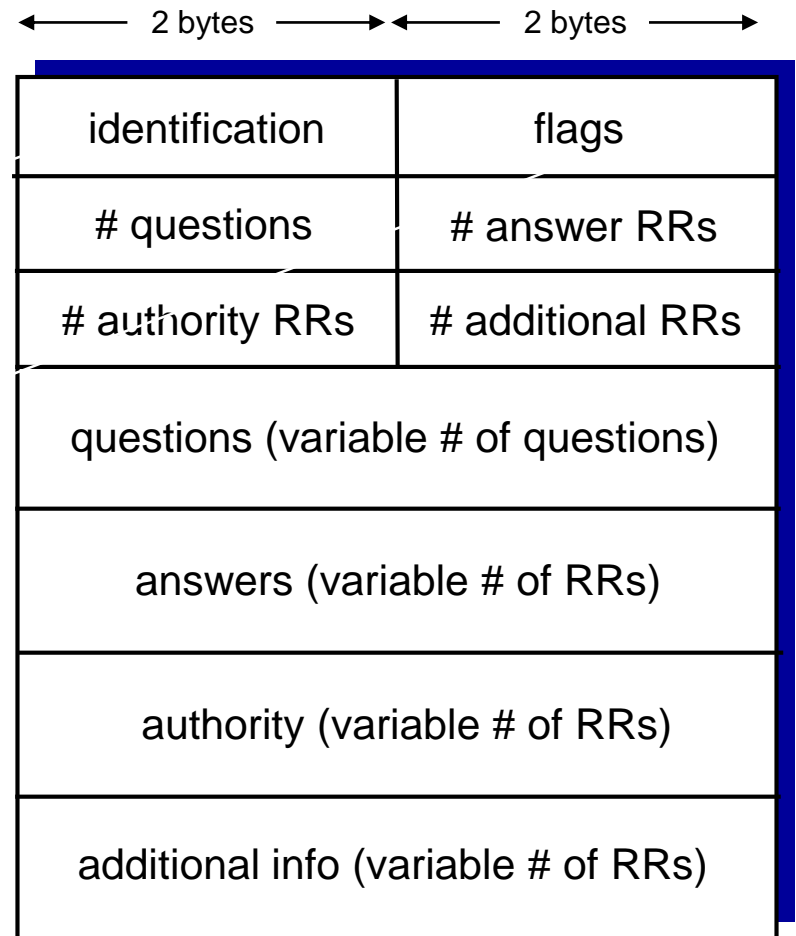
- ❑ Lernt Nameserver eine Zuordnung, wird der Inhalt zwischengespeichert.
 - Timeout für Cache-Einträge: TTL
 - IP Adressen der TLD Server sind so gut wie immer im Cache des Resolvers.

- ❑ Cache-Einträge können **veraltet sein**
 - Es dauert etwas bis sich die Änderungen im DNS verbreiten.

- ❑ Änderung von DNS Einträgen (DNS Update)
 - RFC 2136

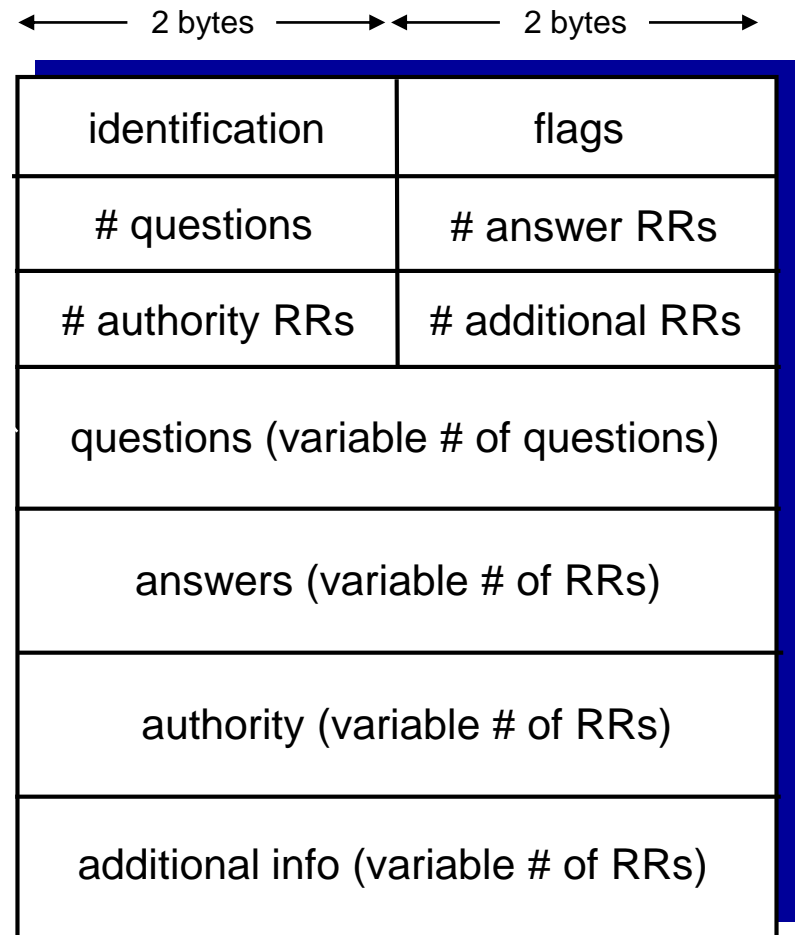
DNS Protokoll: Aufbau der Nachrichten

- ❑ *Query* und *Reply* haben das gleiche Format
- ❑ *Identification:*
 - 16 Bit Zahl
 - Identisch für Query und zugehörige Reply
- ❑ *Flags*
 - Query oder Reply?
 - Recursive Query erwünscht
 - Recursion verfügbar
 - Antwort ist „authoritative“



DNS Protokoll: Aufbau der Nachrichten.

- ❑ Query: Name und Typ
- ❑ Antworten auf eine Query
 - Ggfs. mehrere IP Adressen für einen Namen
- ❑ Hier wird Verweis auf Authoritative Server eingetragen
- ❑ Zusätzliche „hilfreiche“ Information



Betrieb eines eigenen Nameservers / Domain

- ❑ Beispiel: Neues Startup „*NetworkTec*“
- ❑ Registriere *networktec.de*
 - Registrar für .de Domain: *Denic*
 - Informiere Registrar über IP Adresse des eigenen *Nameservers*
 - Hier: `dns1.networktec.de`
 - Registrar fügt seinem (TLD) Nameserver 2 Records hinzu
(`networktec.de`, `dns1.networktec.de`, NS)
(`dns1.networktec.de`, `212.212.212.1`, A)
- ❑ Auf eigenem Nameserver
 - Lege Type A Record für www.networktec.de an
 - Lege Type MX Record für networktec an
 - ...

Publikums-Joker: DNS

Welche der folgenden Aussagen ist **falsch**?

- A. DNS ist ein Protokoll der Anwendungsschicht.
- B. Die Root DNS Server im Internet arbeiten rekursiv.
- C. Im Standard DNS werden DNS Anfragen unverschlüsselt übertragen.
- D. DNS basiert auf UDP.



- ❑ Web und HTTP
- ❑ DNS
- ❑ **Ausblick: Sockets**

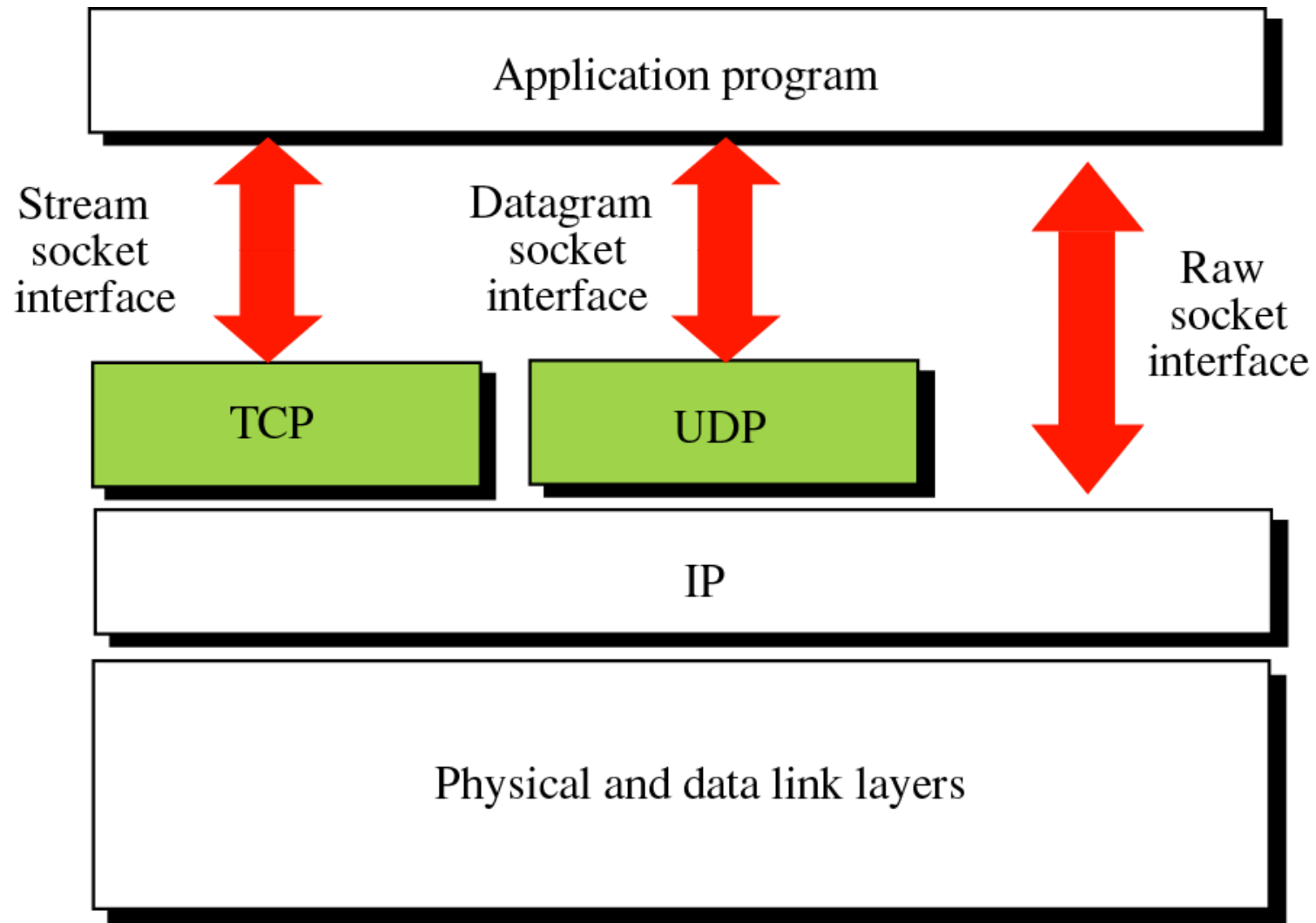
□ 2 Typen

- UDP: unzuverlässige Kommunikation, Datagramme
- TCP: zuverlässig, Datenströme

□ Anwendungsbeispiel

- Client liest Zeile von Tastatur und sendet sie an den Server.
- Server empfängt die Daten und wandelt die Zeichen in Großbuchstaben um.
- Server sendet die modifizierten Zeichen zurück an Client.
- Client empfängt die modifizierten Daten und zeigt sie an.

Socket Types



Exkurs: TCP in Python

server (running on `hostid`)

create socket,
port=`x`, for incoming
request:
`serverSocket = socket()`

wait for incoming
connection request
`connectionSocket =`
`serverSocket.accept()`

read request from
`connectionSocket`

write reply to
`connectionSocket`

close
`connectionSocket`

client

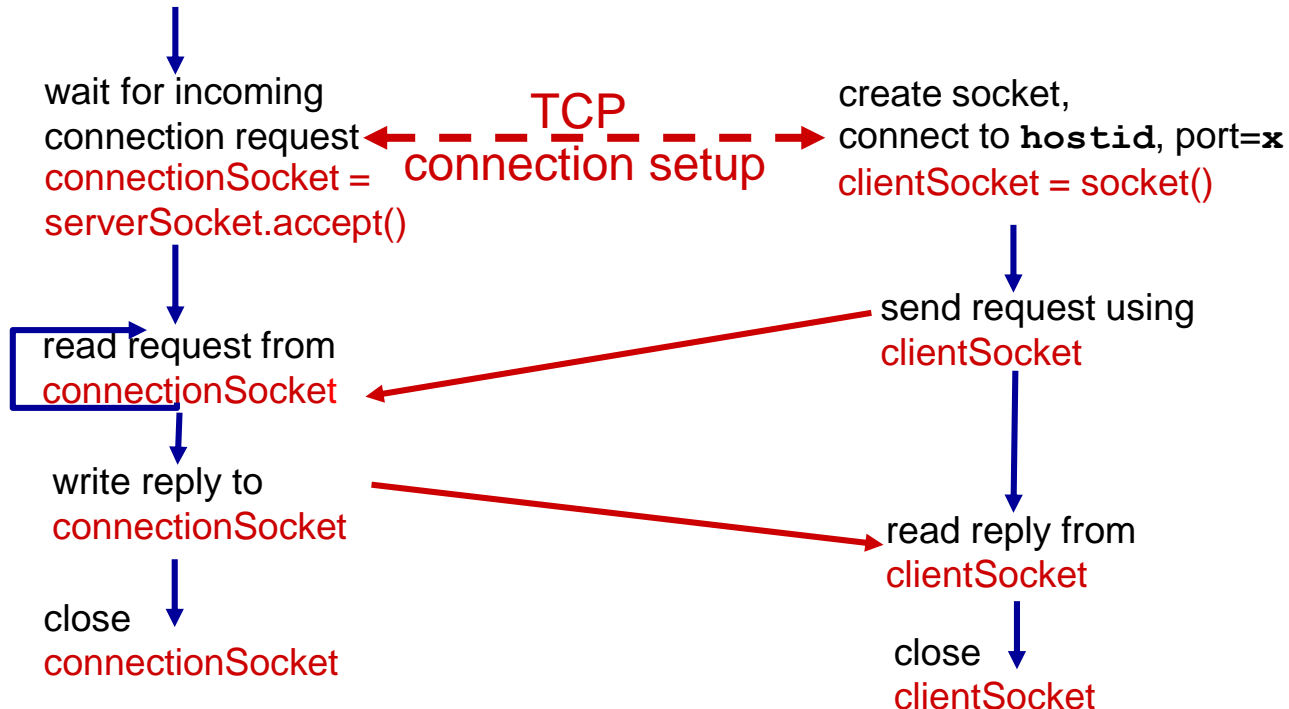
create socket,
connect to `hostid`, port=`x`
`clientSocket = socket()`

send request using
`clientSocket`

read reply from
`clientSocket`

close
`clientSocket`

TCP
connection setup



Exkurs: TCP in C

