Technische Universität Berlin

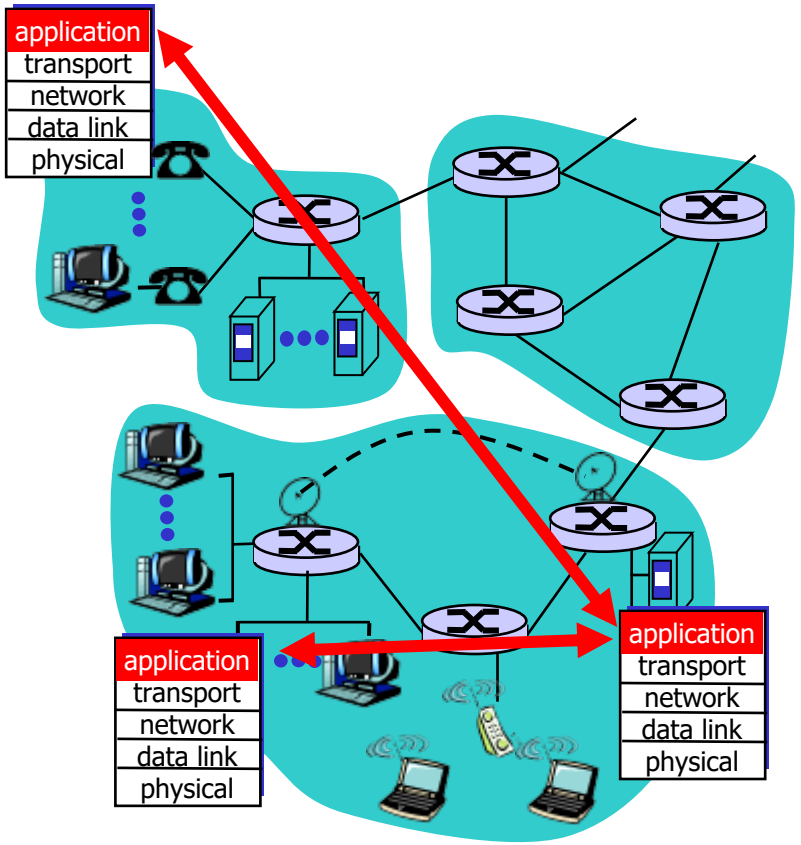
Rechnernetze

Anwendungsschicht

Anwendungsschicht

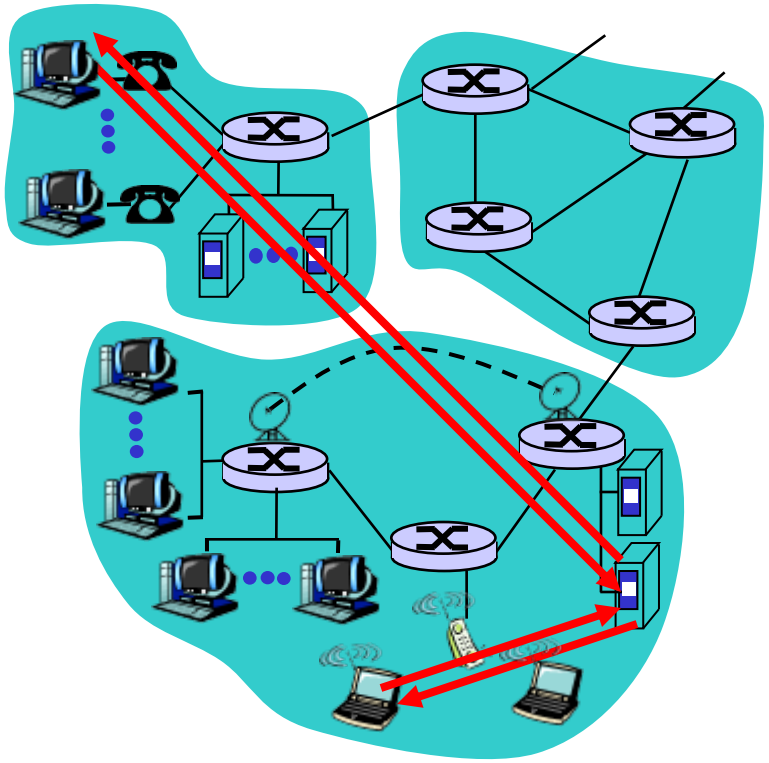
Einführung

- Netzwerkanwendung
  - **Anwendungsprozesse** auf verschiedenen Endsystemen (Hosts), die mittels Nachrichten über ein Netzwerk kommunizieren
  - kann direkt unter Verwendung der **Dienste der Transportschicht** implementiert werden
  - standardisierte Anwendungen benutzen ein **Anwendungsprotokoll**, das das Format der Nachrichten und das Verhalten beim Empfang von Nachrichten festlegt
  - z.B: Web-Browser und -Server
  - die unteren Schichten und der Netzwerkkern benötigen keine Kenntnis der Anwendung
  - einfache Verbreitung, große Dynamik



Einführung

- **Client-Server-Paradigma**
  - Server stellt Dienst zur Verfügung, der von Client angefordert wird
  - übliches Paradigma von vielen traditionellen Anwendungen, wie z.B. Web-Browser und –Server
  - typische Eigenschaften des Servers
    - leistungsfähig
    - immer verfügbar
  - typische Eigenschaften der Clients
    - nur manchmal verbunden
    - kommunizieren mit Server, nicht untereinander

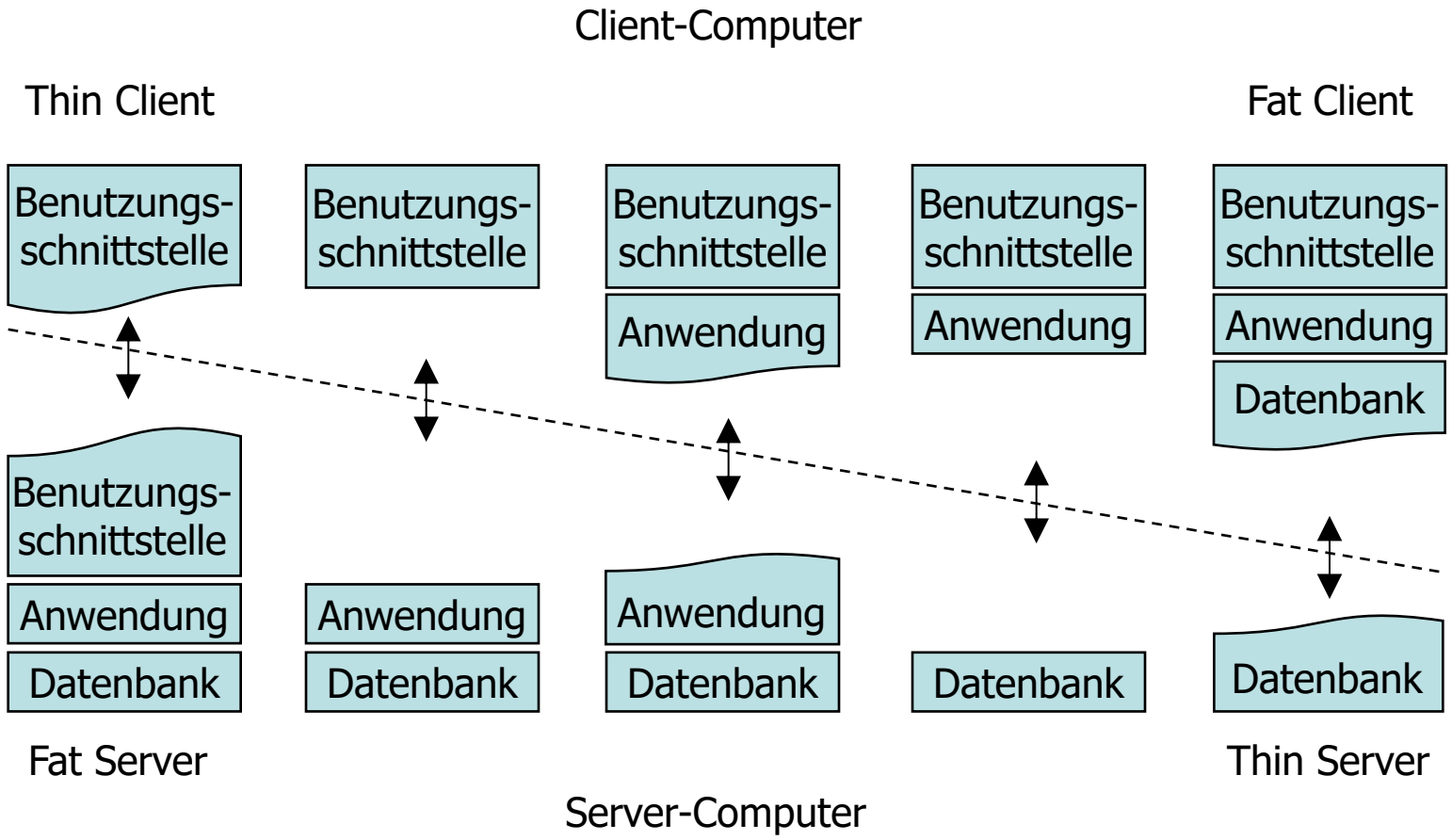


Einführung

- **Client-Server-Paradigma ist zentralisierte Architektur**
- Weitere Paradigmen
  - **wechselnde Rolle** von Client und Server: Hosts übernehmen mal die eine, mal die andere Rolle (z.B. bei Web-Caching oder SMTP)
  - **verteilte Anwendung**: besteht aus mehreren unabhängigen Anwendungen, die zusammen wie eine einzelne Anwendung erscheinen (z.B. Webshop mit Web-Server, Applikations-Server und Datenbank), Koordination ist zwar verteilt, findet aber für das Gesamtsystem statt
  - noch stärker **dezentrale Architektur**: autonome sich selbst organisierende Systeme ohne globale Steuerung (z.B. einige **Peer-to-Peer-Anwendungen** wie Gnutella, Chord)
  - **Hybridarchitektur**: zur Initialisierung ist eine zentrale Architektur nötig, die Anwendung findet dann dezentral direkt zwischen Hosts statt (z.B. bei Session Initiation Protocol, SIP oder bei manchen Peer-to-Peer-Anwendungen wie Bittorrent)

Einführung

- Varianten des Client-Server-Paradigmas



Einführung

- Dienste der Transportschicht
  - im Internet gibt es zwei dominierende Transportprotokolle
    - **TCP**: verbindungsorientiert (abstrakte Sicht des Versendens eines Bytestroms), zuverlässig
    - **UDP**: verbindungslos (Versenden einzelner Datagramme), unzuverlässig
  - werden meist im Betriebssystem realisiert
  - die meisten Betriebssysteme bieten **Socket-Schnittstelle**, die durch Programmiersprachen als API angeboten wird
  - mit Socket kann festgelegt werden
    - Transportprotokoll (TCP oder UDP)
    - IP-Adresse von Sende- und Zielhost
    - Portnummern (um Anwendungen auf Hosts zu unterscheiden)
  - und so können Anwendungen programmiert werden …

Einführung

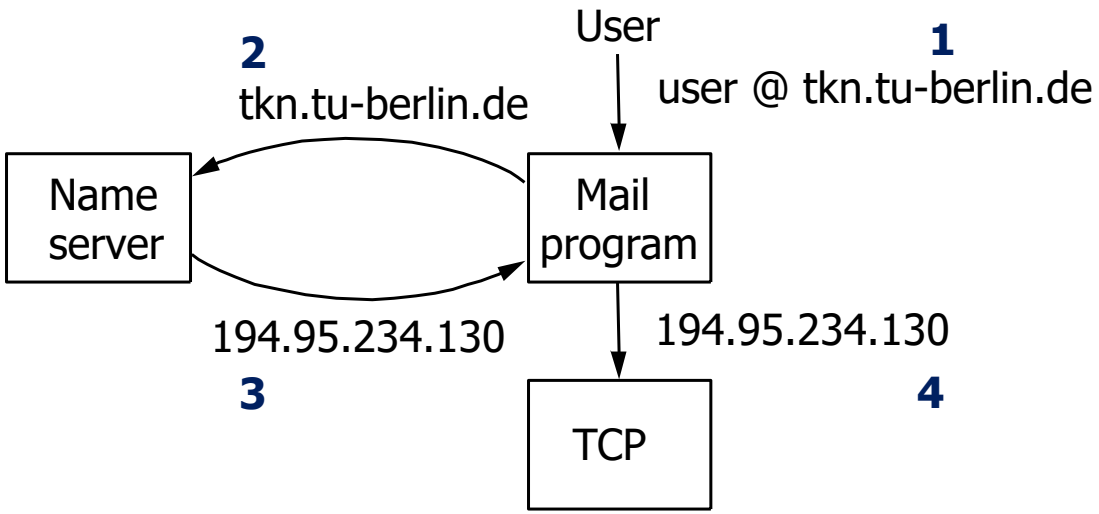
- Einige bekannte Anwendungsprotokolle und das darunterliegende Transportprotokoll

Anwendung	Anwendungsprotokoll	verwendetes Transportprotokoll
E-mail	SMTP [RFC 2821]	TCP
Remote Terminal Access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
Dateitransfer	FTP [RFC 959]	TCP
Remote File Server	NFS [McKusik 1996]	UDP or TCP
Streaming Multimedia	RTP [RFC 1889, 3550]	UDP or TCP
Internettelefonie	RTP [RFC 1889, 3550]	meistens UDP

Domain Name Service (DNS)

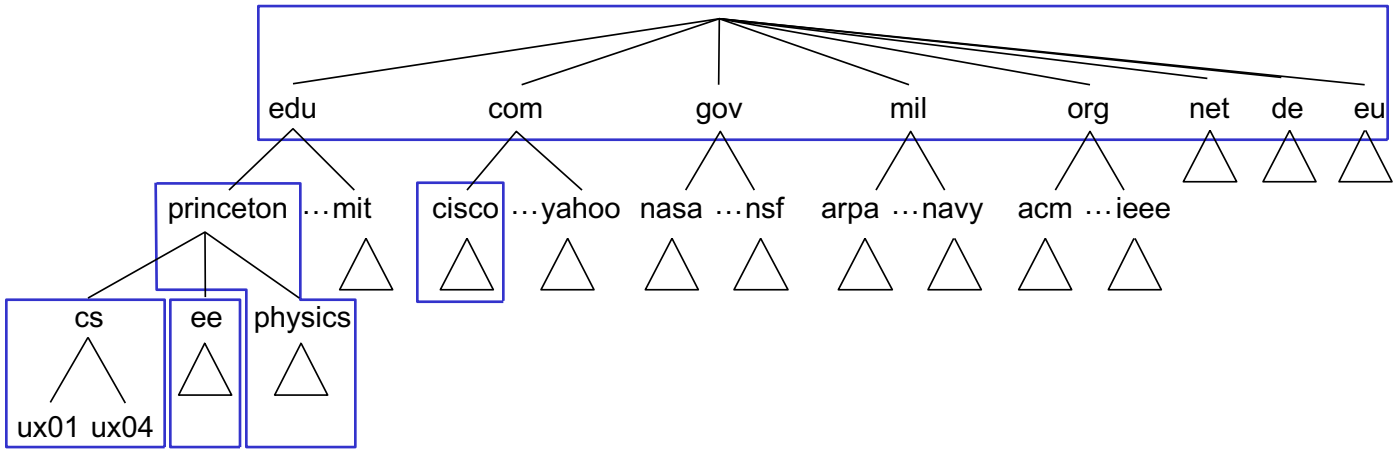
Domain Name Service (DNS)

- Host-Namen bzw. Domain-Namen lesbar
- DNS bildet Domain-Namen auf Werte ab
- diese Werte sind u.a. IP-Adressen
- DNS ist verteilte Datenbank, besteht aus vielen Namen-Servern, die über ein Anwendungsprotokoll kommunizieren
- eine wesentliche Aufgabe, um die Infrastruktur zu nutzen
- z.B. Namens-Auflösung beim Versenden einer Email:



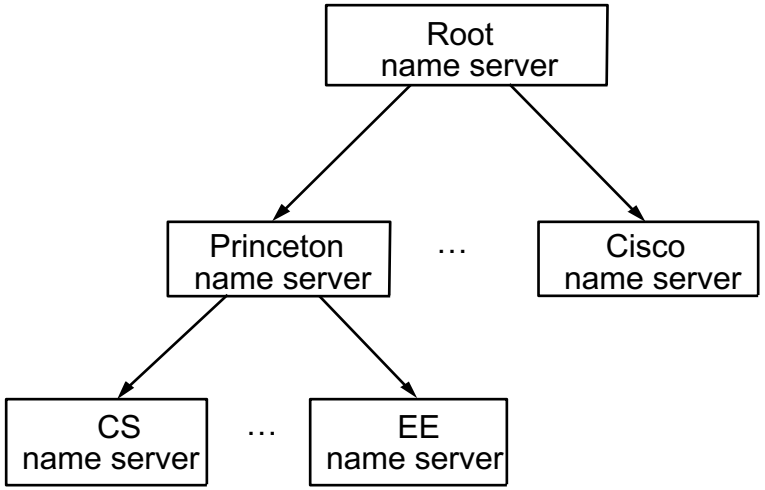
DNS

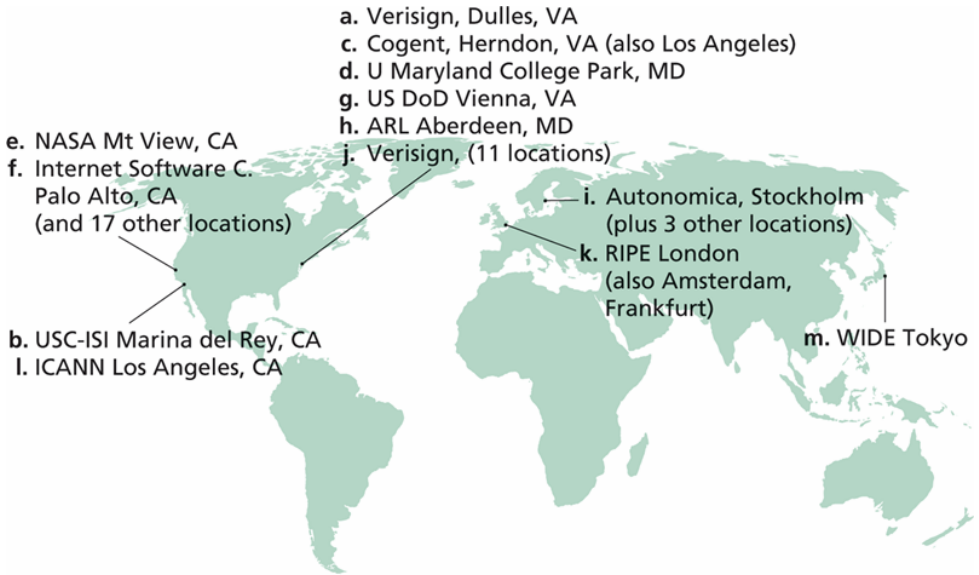
- Domain-Struktur
  - DNS implementiert hierarchischen Namensraum für Internet-Objekte
  - von links nach rechts lesen, von rechts nach links verarbeiten
  - eine **Zone** wird von einem Name-Server verwaltet
  - die Hierarchie wird durch die Namen-Server implementiert



DNS

- Hierarchie von Name-Servern
  - **Root Name Server**
    einige wenige
  - **Top-level Domain-Server**
    für com, org, net, edu, uk, de, eu, …
  - …
  - **autoritativer Name-Server**
    unterste Ebene, für
    einzelne Organisation





DNS: Resource Records

- Resource Records
  - Datensätze der Namenserver (Domainname, Wert, Typ, TTL)
  - TTL: Time to Live, Dauer der Gültigkeit
  - Typ = A
    - Wert = IP-Adresse
    - Bsp.: (www2.tkn.tu-berlin.de, 130.149.110.75, A, TTL)
  - Typ = NS
    - Wert = Domainname eines Hosts, auf dem ein Namen-Server läuft, der Namen in der Domain auflösen kann
    - Bsp.: (tkn.tu-berlin.de, ns.tu-berlin.de, NS, TTL)
  - Typ = CNAME (Canonical Name)
    - Wert = kanonischer Name eines Hosts, ermöglicht Aliasnamen
    - Bsp.: (cic.cs.princeton.edu, cicada.cs.princeton.edu, CNAME, TTL)
  - Typ = MX (Mail Exchange)
    - Wert = Domain-Name des Hosts, auf dem Mail-Server läuft
    - Bsp.: (tkn.tu-berlin.de, b1861.mx.srv.dfn.de, MX, TTL)

DNS: Resource Records

- Bsp: Resource Records
  - Root Name Server

    (tkn.tu-berlin.de, ns.tu-berlin.de, NS, TTL)
    (ns.tu-berlin.de, 130.149.7.7, A, TTL)
    (cisco.com, ns.cisco.com, NS, TTL)
    (ns.cisco.com, 128.96.32.20, A, TTL)
    …

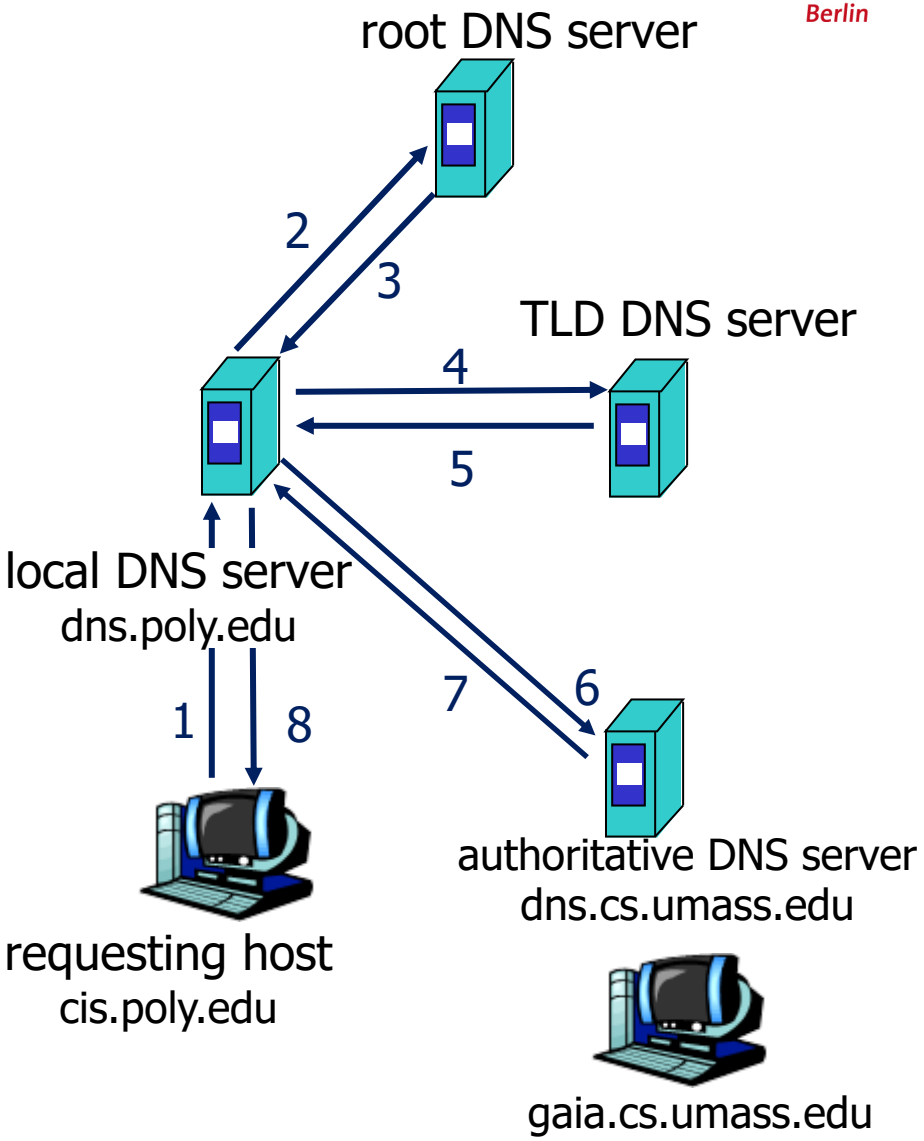
  - enthält einen NS-Datensatz für jeden Server der nächsten Ebene und einen A-Datensatz mit der IP-Adresse
  - diese bilden zusammen einen Verweis auf die Server der zweiten Ebene

DNS: Protokoll

- Anfragearten
  - **iterativ**
    - Antwort: anderer Server, der Namen evtl. auflösen kann (oder keine Antwort)
    - NS- und A-Datensatz
    - Antwort wird sofort geliefert, es muß keine Information gespeichert werden, gut für hochfrequentierte Server
  - **rekursiv**
    - Antwort: Auflösung des Namens, die u.U. von anderen Servern geholt wird
    - A-Datensatz
    - bei Anfrage an einen anderen Server muß die Information gespeichert werden

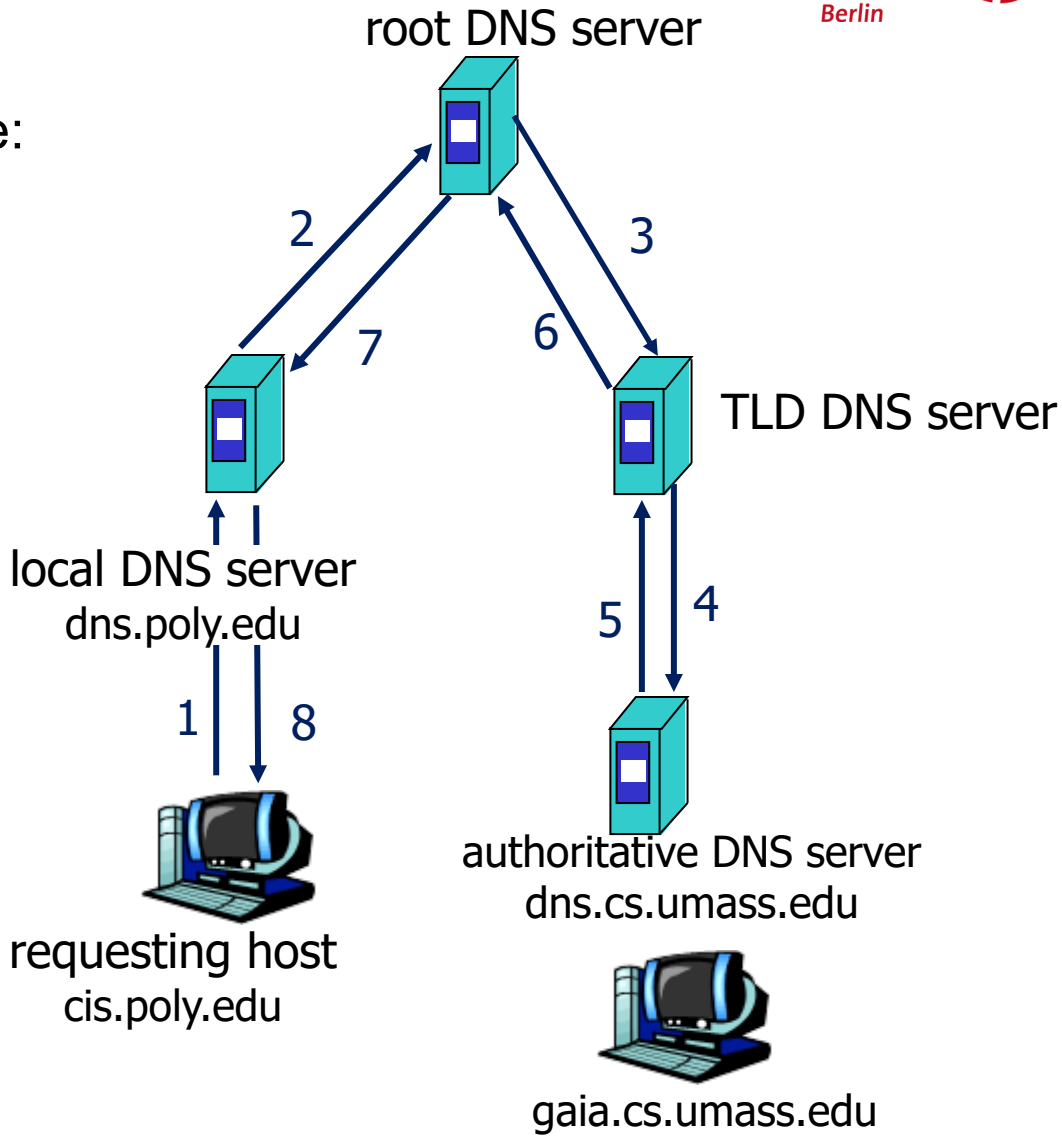
DNS: Protokoll

- Beispiel für iterative Anfrage:



DNS: Protokoll

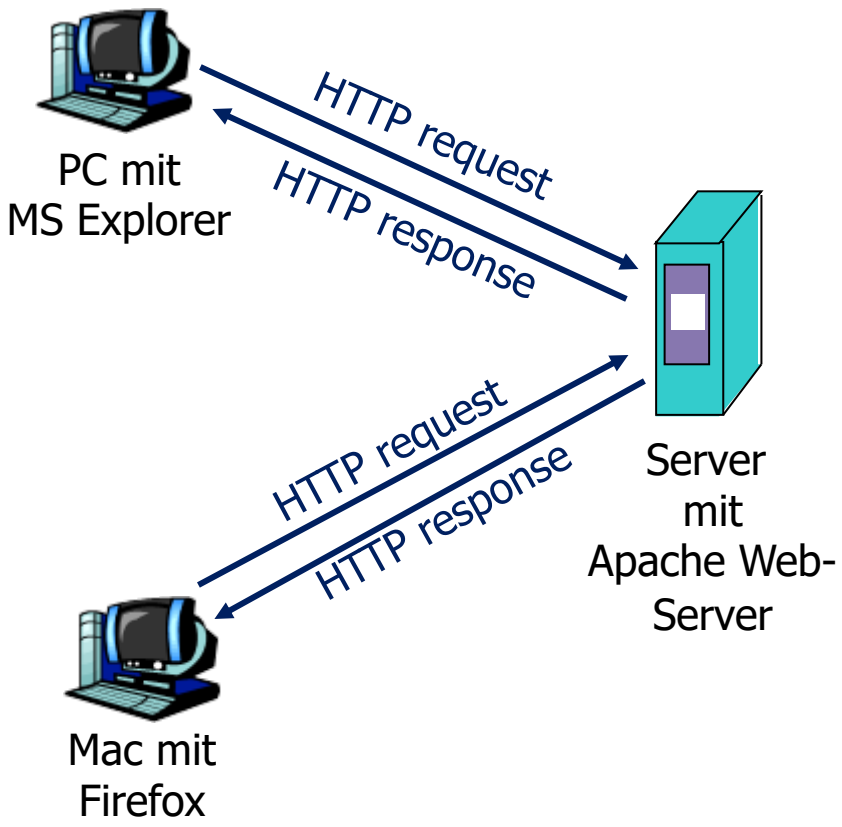
- Beispiel für rekursive Anfrage:



Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP)

- Ablauf
  - Benutzer gibt **Uniform Resource Locator (URL)** in Web-Browser ein
  - URL enthält Host-Namen eines Web-Servers und den Pfad zu einem **Objekt** (Datei) dort
  - Web-Browser stellt Anfrage an Web-Server für dieses Objekt
  - Web-Server liefert Objekt an Web-Browser zurück
  - Web-Browser stellt Objekt in für den Benutzer lesbarer Form dar



Uniform Resource Locator (URL)

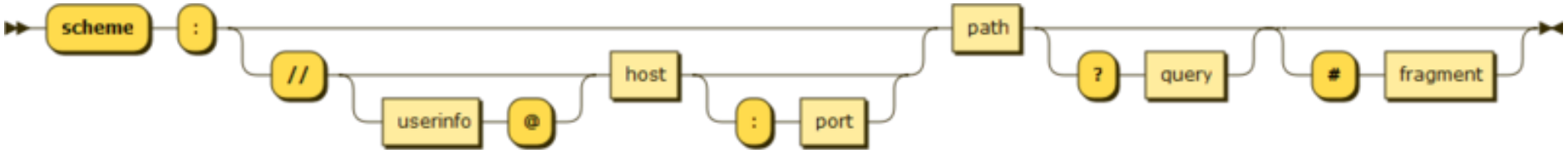
- einheitlicher Ressourcenzeiger
- Defines (roughly) the access method and the path
- protocol://host-name:port/directory-path/resource
- E.g.: http://www.example.com/index.html

http
protocol

hostname

file
name

- More precise: URL is a specific type of Uniform Resource Identifier (URI):

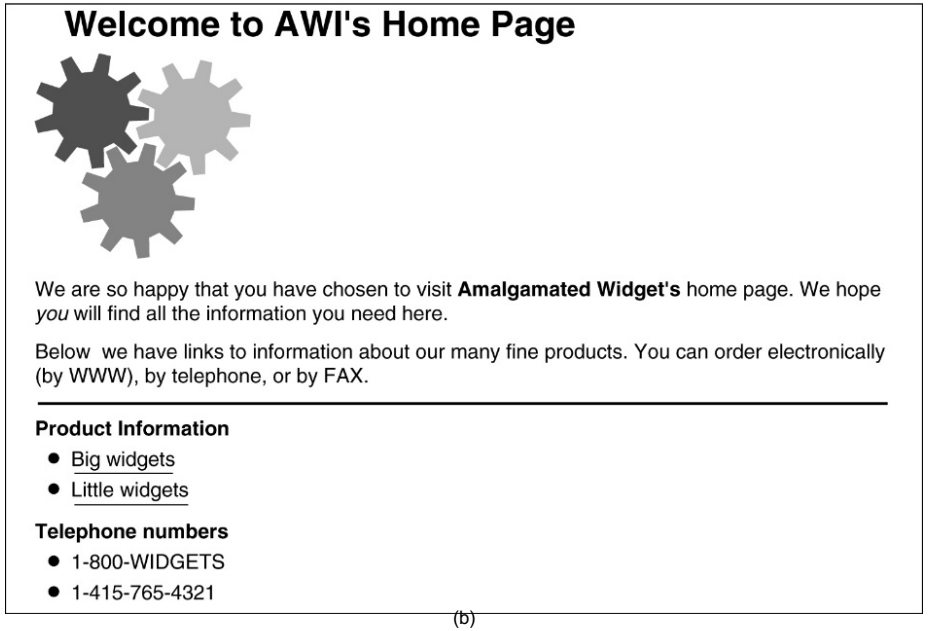


HTML – HyperText Markup Language

- (a) The HTML code for a sample Web page.
- (b) The formatted (rendered) page.

```
<html>
<head><title> AMALGAMATED  WIDGET,  INC. </title> </head>
<body> <h1> Welcome to AWI's Home Page</h1>
<img src="http://www.widget.com/images/logo.gif" ALT="AWI Logo"> <br>
We are so happy that you have chosen to visit <b> Amalgamated Widget's </b>
home page. We hope <i> you </i> will find all the information you need here.
<p>Below we have links to information about our many fine products.
You can order electronically (by WWW), by telephone, or by fax. </p>
<hr>
<h2> Product information  </h2>
<ul>
   <li> <a href="http://widget.com/products/big">  Big widgets </a>
   <li> <a href="http://widget.com/products/little">  Little widgets </a>
</ul>
<h2> Telephone numbers</h2>
<ul>
   <li> By telephone: 1-800-WIDGETS
   <li> By fax: 1-415-765-4321
</ul>
</body>
</html>
```

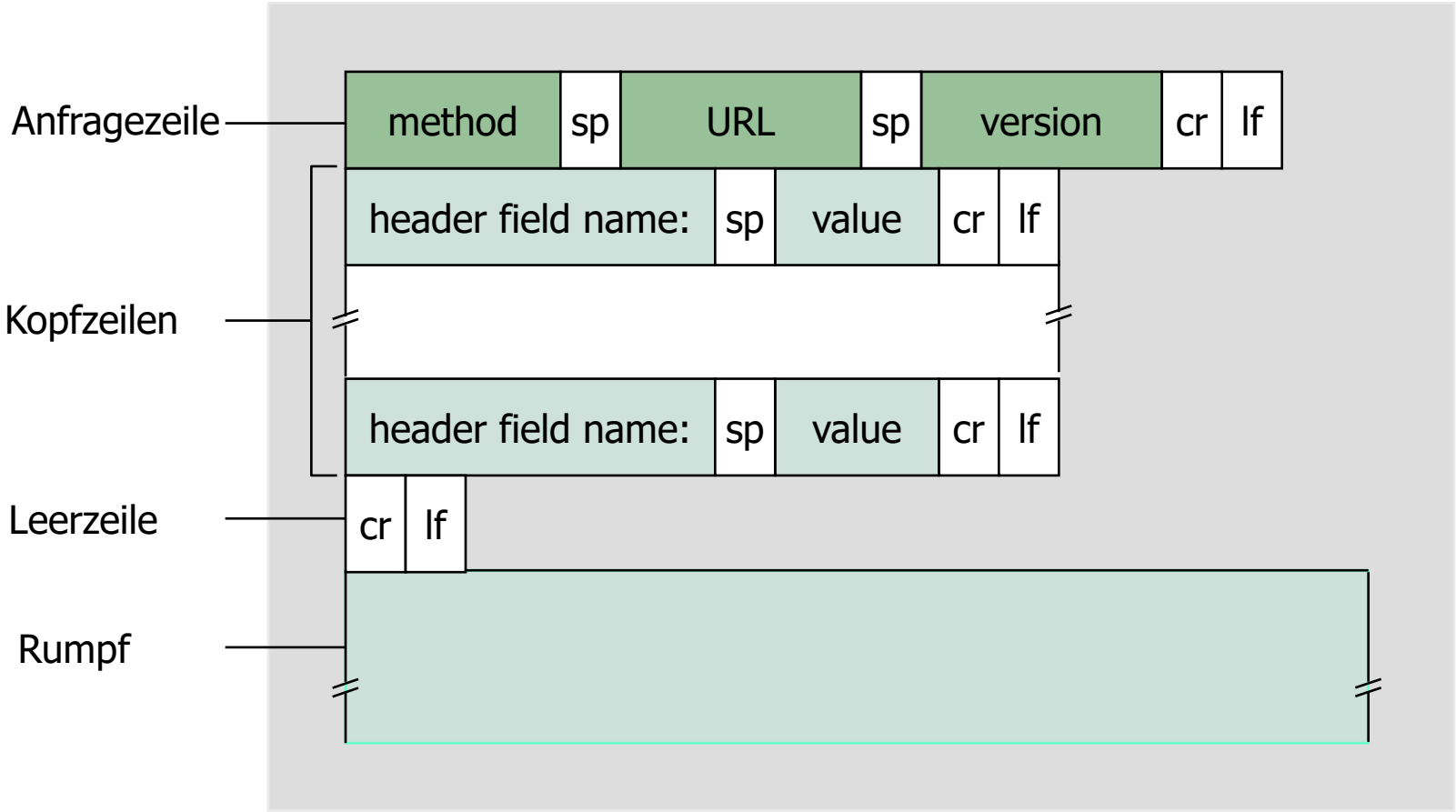
(a)



(b)

HTTP: Anfragenachrichten

- Format der Anfragenachrichten



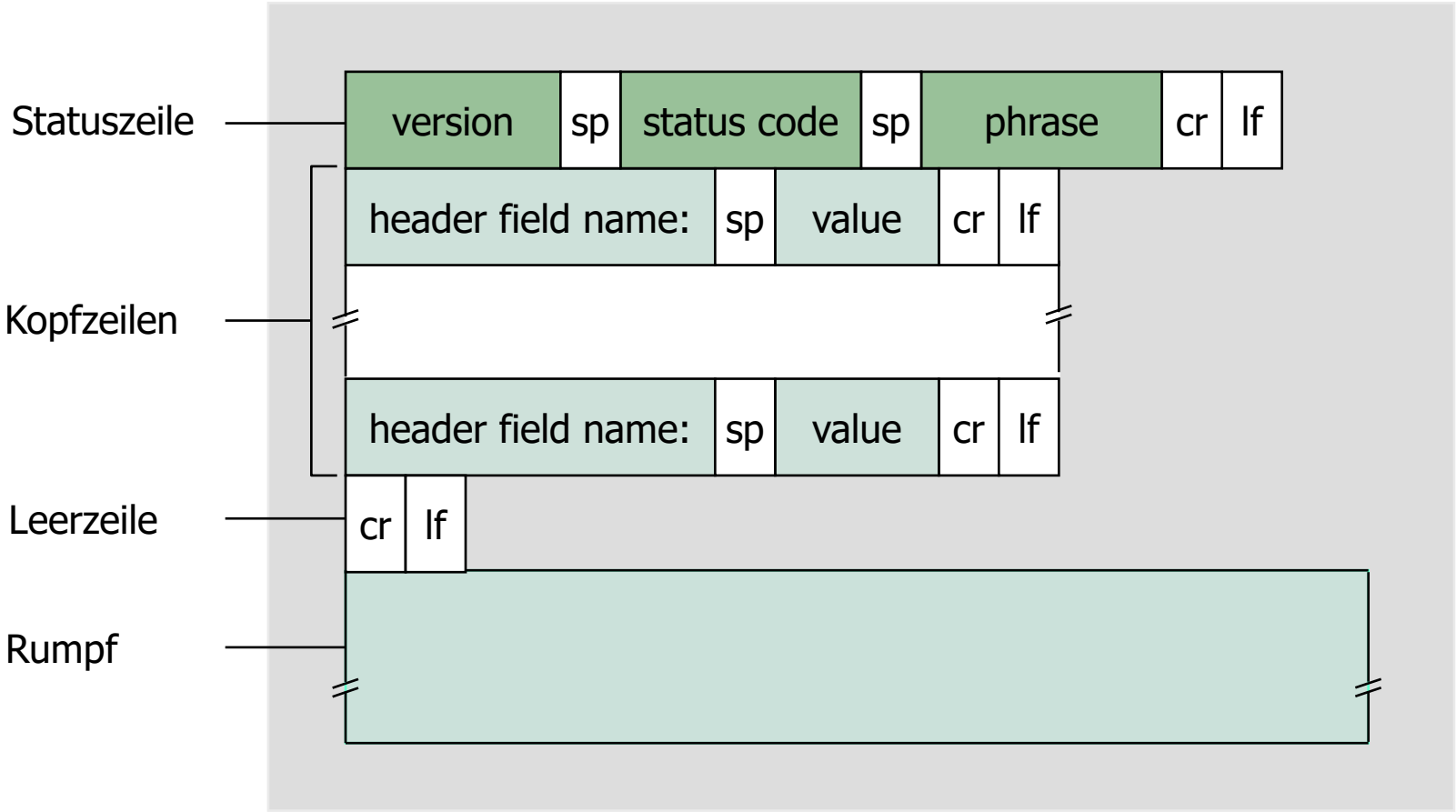
HTTP: Anfragenachrichten

- Methoden
  - **GET**: Abruf eines Dokuments, besteht aus Methode, URL, Version
  - **HEAD**: Abruf von Metainformationen eines Dokuments
  - **POST**: Ausgabe von Informationen an Server
  - Put, Delete, Trace, Options
- Kopfzeilen
  - Typ/Wert-Paare, Typen: Host, User-agent, …
- Rumpf
  - leer bei GET, kann bei POST Inhalt haben
- Beispiel Anfragenachricht:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language:fr

(extra carriage return, line feed)
```

HTTP: Antwortnachrichten

- Format der Antwortnachrichten



HTTP: Antwortnachrichten

- Mögliche Codes in der Statuszeile
  - 200 OK („alles klar“)
  - 301 Moved Permanently (Redirection: Objekt zu finden unter Location: …)
  - 400 Bad Request (Anfragenachricht nicht verstanden)
  - 404 Not Found (Objekt nicht gefunden)
  - 505 HTTP Version Not Supported

- Beispiel-Antwortnachricht:

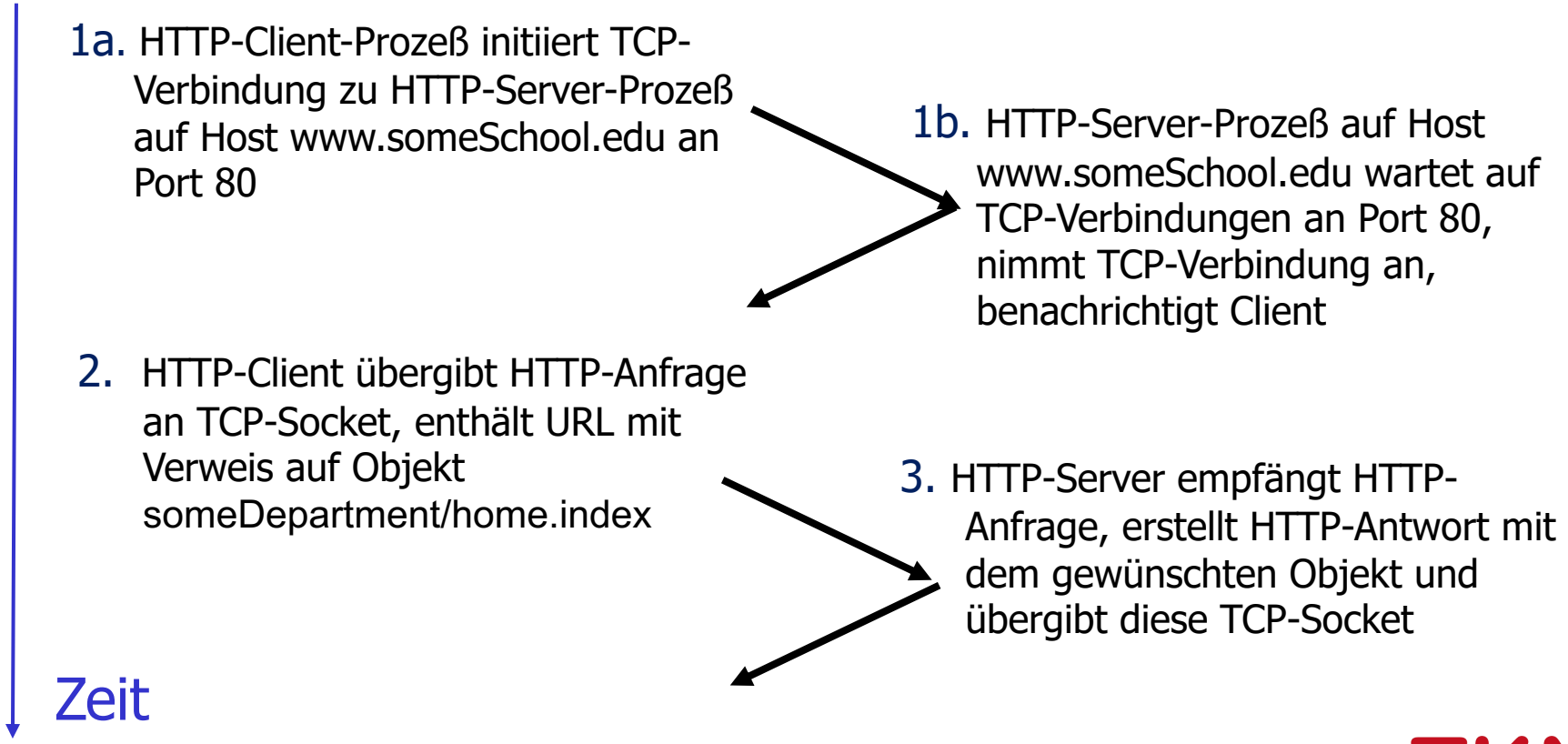
```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 …
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

HTTP: Ablauf

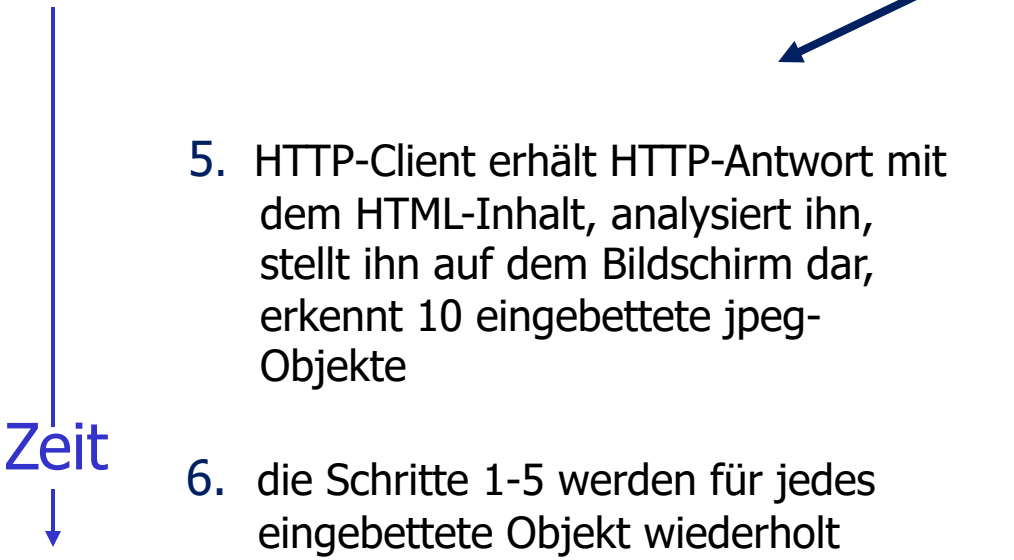
- HTTP-Ablauf
  - **nicht-persistentes HTTP**
    - für jedes Objekt wird einzelne TCP-Verbindung geöffnet, Server beendet sie sofort nach dem Senden eines Objekts
    - entweder Basis-Seite und eingebettete Objekte sequentiell
    - oder parallele einzelne Verbindungen für die eingebetteten Objekte
  - **persistentes HTTP**
    - Server läßt Verbindung bestehen
    - alle Objekte werden über eine TCP-Verbindung gesendet
    - **ohne Pipelining**: nach jedem Objekt Anfrage für nächstes Objekt
    - **mit Pipelining**: eine Anfrage für alle eingebetteten Objekte
  - Was sind die Vor- und Nachteile?
  - Standardport des Web-Servers: 80

HTTP: Ablauf

- Beispiel-Ablauf von nicht-persistentem HTTP
  - URL: www.someSchool.edu/someDepartment/home.index
  - Basis-Seite enthält 10 eingebettete Objekte (jpeg)



HTTP: Ablauf



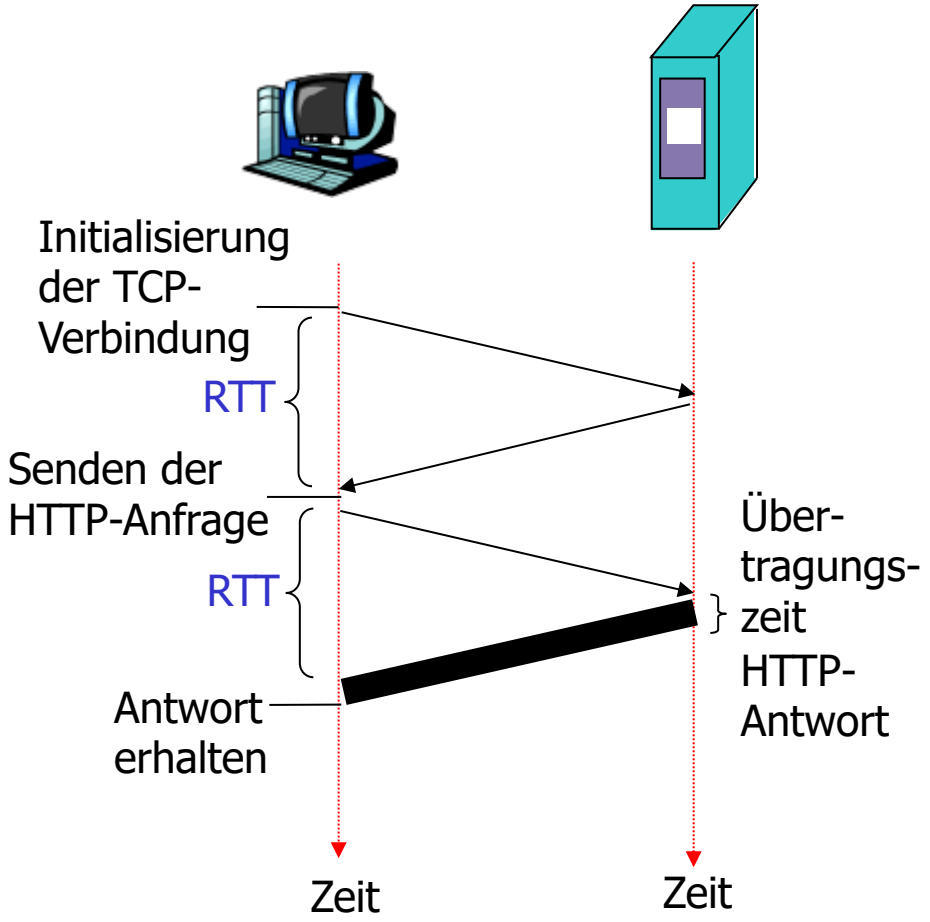
4. HTTP-Server schließt TCP-Verbindung

5. HTTP-Client erhält HTTP-Antwort mit dem HTML-Inhalt, analysiert ihn, stellt ihn auf dem Bildschirm dar, erkennt 10 eingebettete jpeg-Objekte

6. die Schritte 1-5 werden für jedes eingebettete Objekt wiederholt

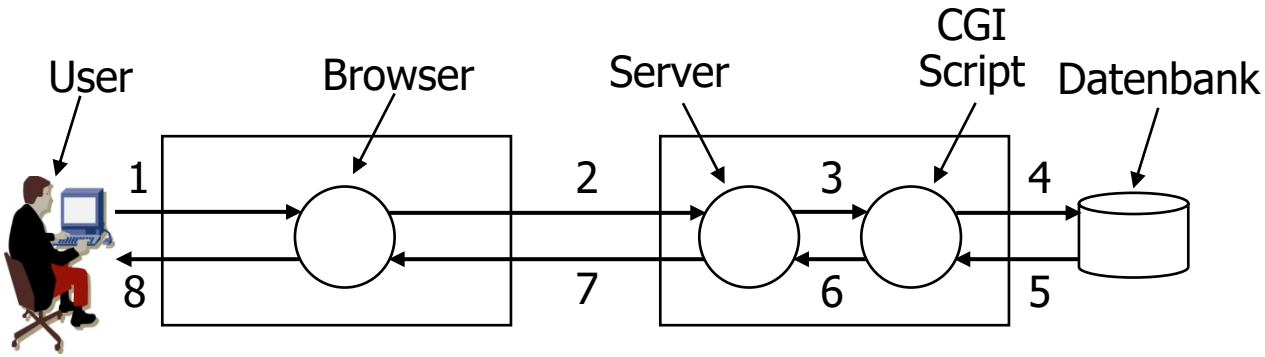
HTTP: Ablauf

- Antwortzeit
  - Basis-Seite
    - Aufbau der TCP-Verbindung erfordert eine **Round Trip Time (RTT)**
    - Anfragenachricht hin, Antwortnachricht zurück, erfordert noch eine RTT
    - insgesamt:
      2 RTT
      + Zeit zum Senden
      + weitere Wartezeiten durch TCP
  - wie ist es bei den anderen HTTP-Varianten?



HTTP: Dynamische Inhalte

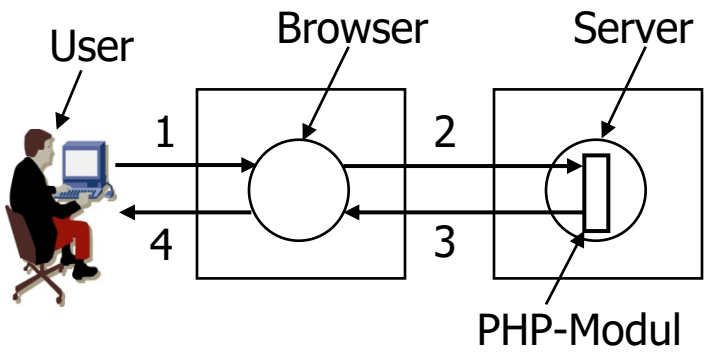
- Senden von Information vom Browser zum Server
  - in Rumpf von Anfragenachricht mit POST
  - häufig: als Typ/Wert-Paare angehängt an die URL in einer Anfragenachricht mit GET
- Dynamische Inhalte mit CGI-Skripten
  - Common Gateway Interface (CGI) verarbeitet als externer Prozeß die Information und liefert neue HTML-Seite an Server

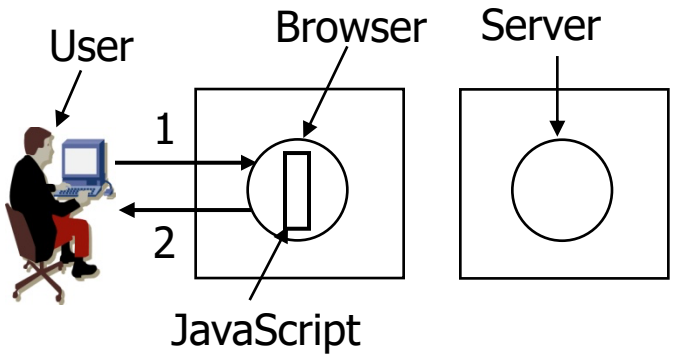


1. User füllt Formular aus
2. mit HTTP an Server
3. wird CGI übergeben
4. CGI fragt DB
5. DB-Eintrag gefunden
6. CGI erstellt HTML
7. mit HTTP an User
8. HTML darstellen

HTTP: Dynamische Inhalte

- Dynamische Inhalte durch Scripting
  - durch Interpretation von eingebetteten Skripten können dynamische Inhalte erzeugt werden
  - Server-seitiges Scripting: im HTML ist Code eingebettet, der vom Server interpretiert wird und dabei HTML erzeugt, z.B. PHP
  - Client-seitiges Scripting: im HTML ist Code eingebettet, der vom Client interpretiert wird, z.B. JavaScript



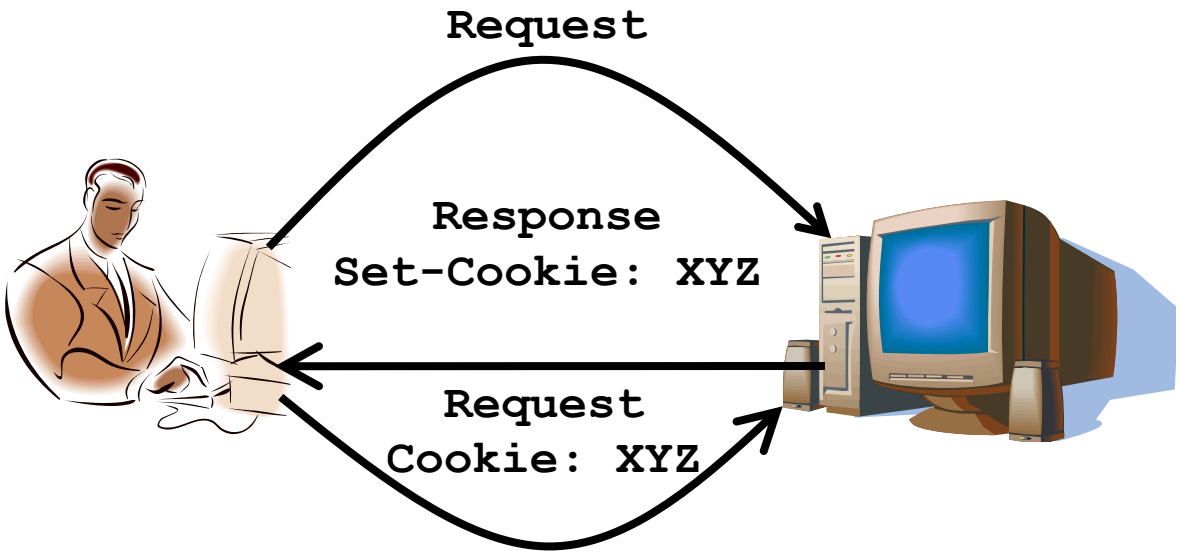


HTTP is Stateless

- **Stateless** protocol (e.g. GET, PUT, DELETE)
  - Each request-response exchange treated independently
  - Servers not required to retain state
- This is **good** as it improves scalability on the server-side
  - Don’t have to retain info across requests
  - Can handle higher rate of requests
  - Order of requests doesn’t matter
- This is also **bad** as some applications need persistent state
  - Need to uniquely identify user or store temporary info
  - e.g., shopping cart, user preferences/profiles, usage tracking, …

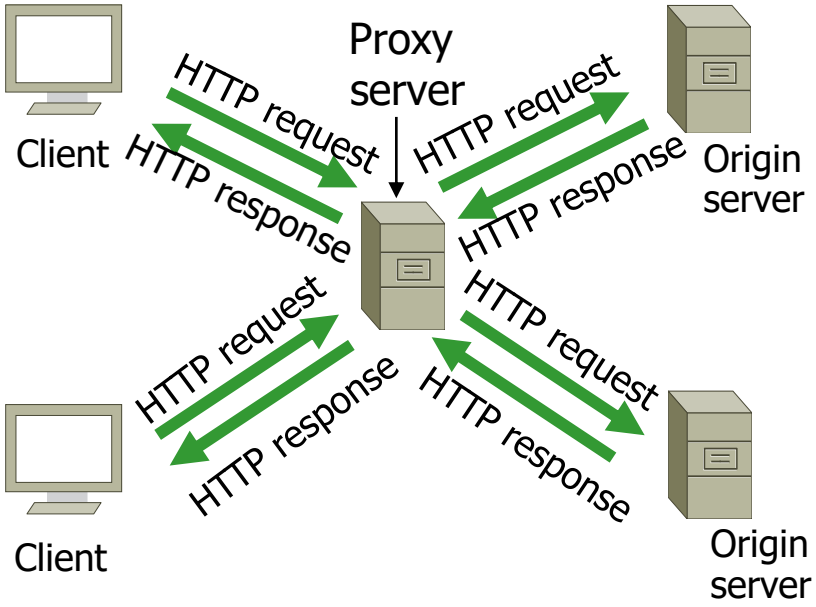
State in a Stateless Protocol: Cookies

- Client-side state maintenance
  - Client stores small (?) state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication



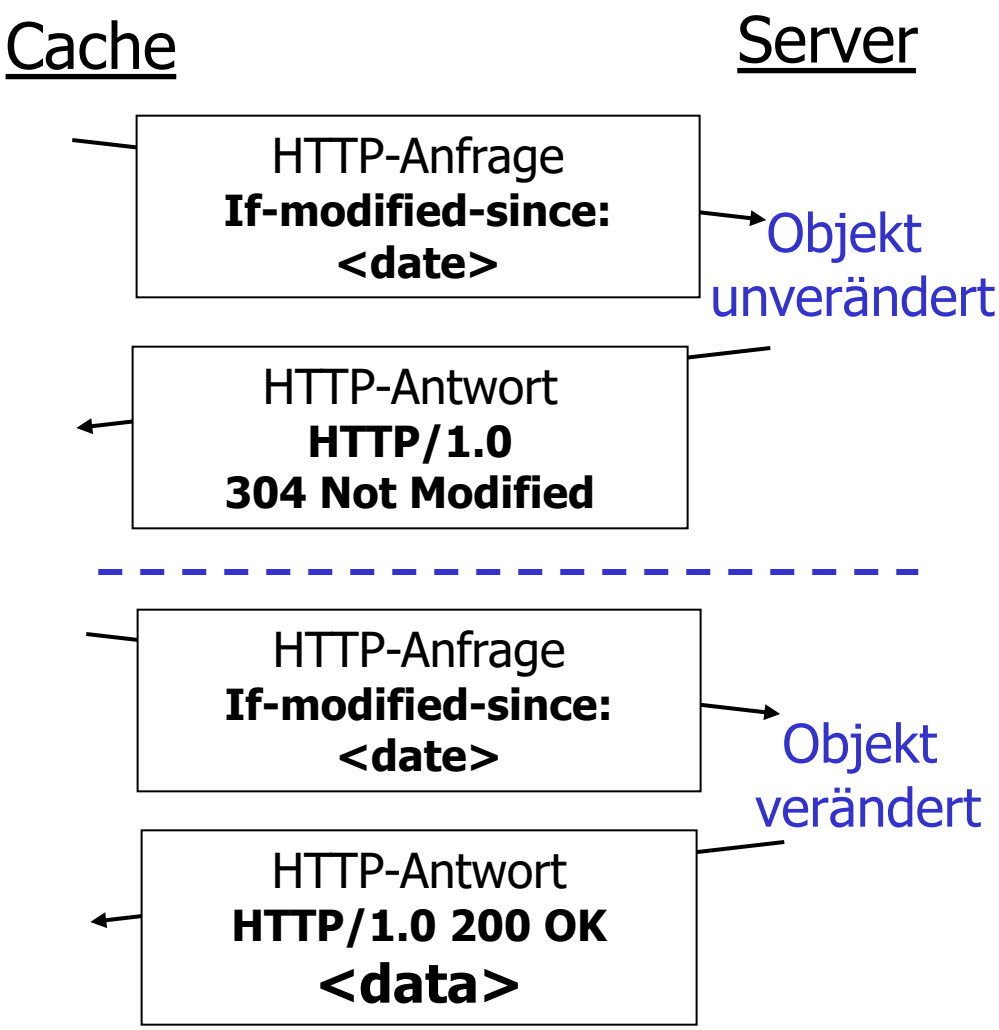
HTTP: Caching

- Web-Caching
  - Verringerung der Wartezeit des Benutzers und des Netzwerkverkehrs durch Zwischenspeicher
  - Cache ist Server für Web-Browser und Client für Web-Server
  - möglich an vielen Stellen: Browser, angeschlossenes LAN, ISP, …



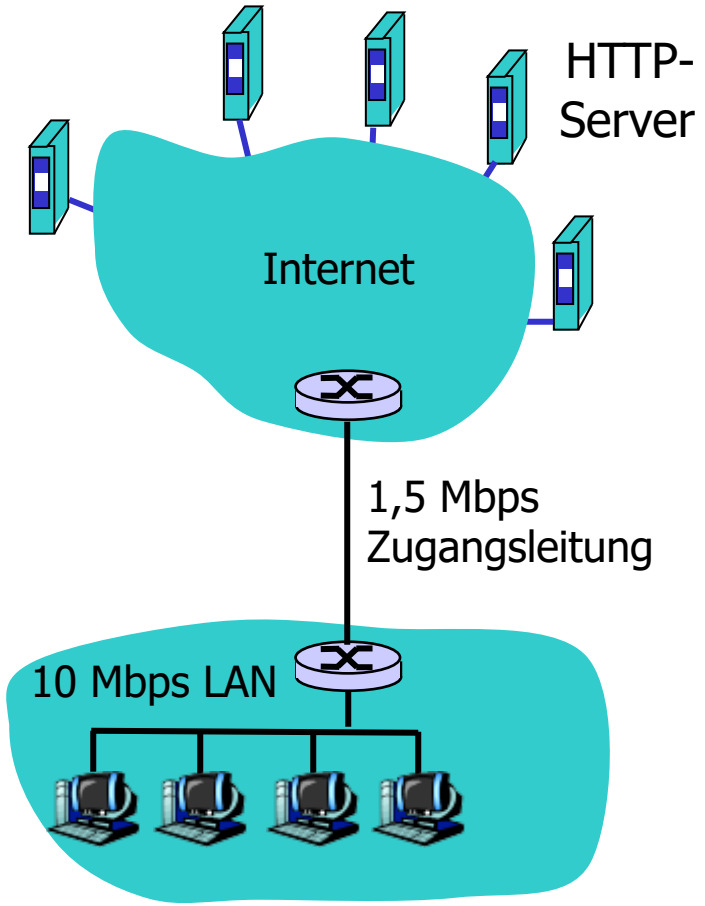
HTTP: Caching

- Cache kann bei Server erfragen, ob sein Objekt noch aktuell ist:



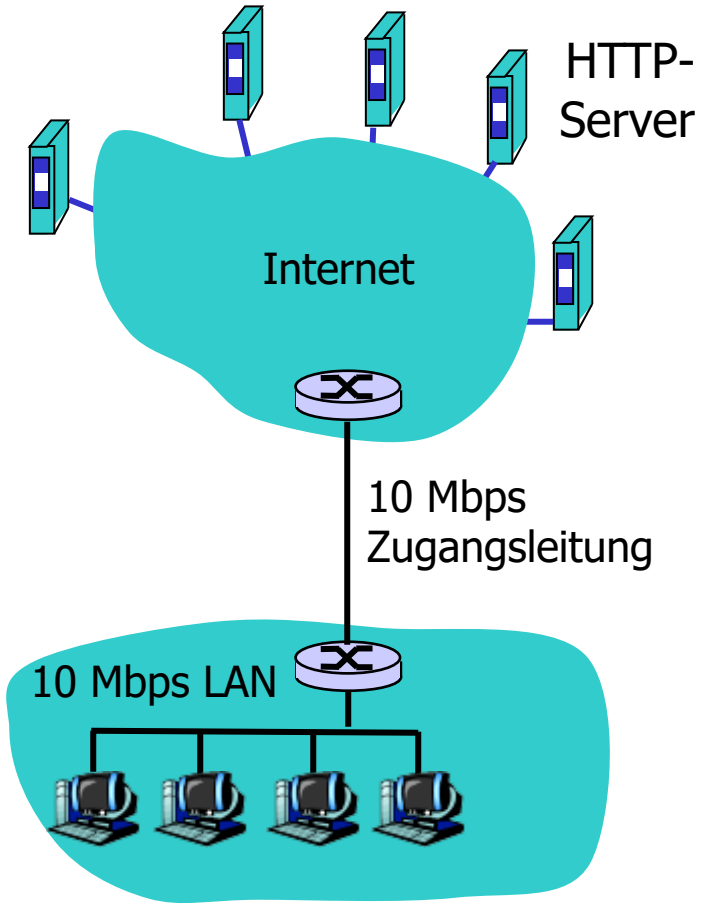
HTTP: Caching

- Beispiel für Nutzen eines Caches
  - Annahmen
    - mittlere Objektgröße = 100.000 Bits
    - mittlere Rate von HTTP-Anfragen der Clients im LAN = 15/s
    - Internetverzögerung zwischen LAN und HTTP-Server = 2 s
  - Folgen
    - Auslastung des LANs
      15/s · 100.000 Bits / 10 · 10^6 Bits/s
      = 0,15 = 15 %
    - Auslastung der Zugangsleitung
      15/s · 100.000 Bits / 1,5 · 10^6 Bits/s
      = 1 = 100 %
    - Gesamtverzögerung = Verzögerung im LAN + beim Zugang + im Internet = ms + Minuten + 2 s = Minuten



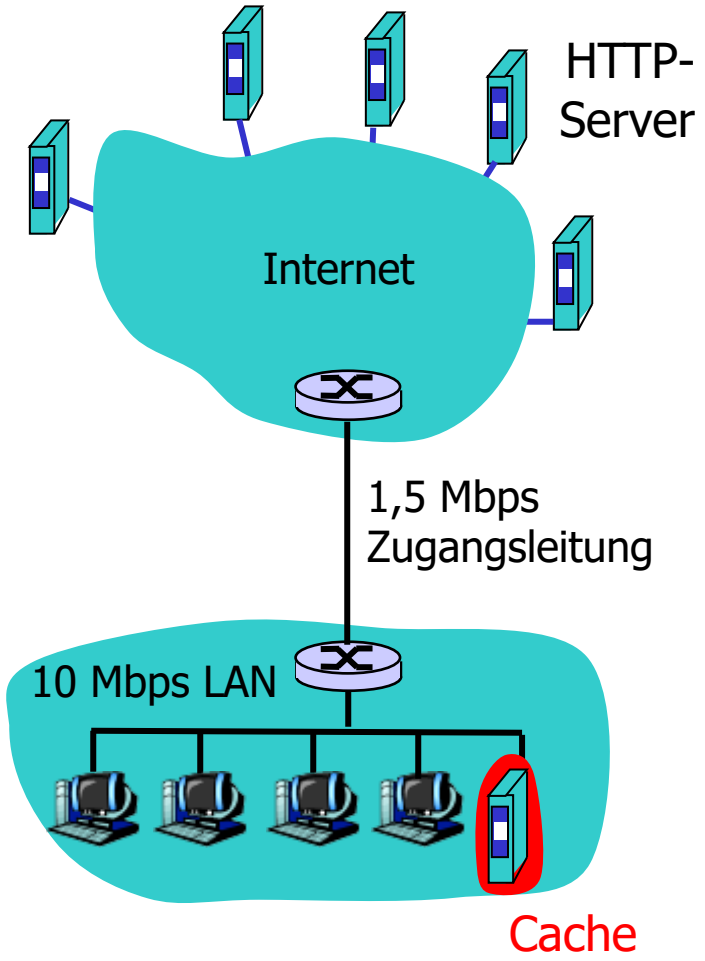
HTTP: Caching

- 1. Lösung: Upgrade des Zugangs
  - Zugangsleitung mit 10 Mbps
  - möglich, aber mit Kosten verbunden
  - Folgen
    - Auslastung des LANs = 15 %
    - Auslastung der Zugangsleitung
      15/s · 100.000 Bits / 10 · 10^6 Bits/s
      = 0,15 = 15 %
    - Gesamtverzögerung = Verzögerung im LAN + beim Zugang + im Internet = ms + ms + 2 s = Sekunden



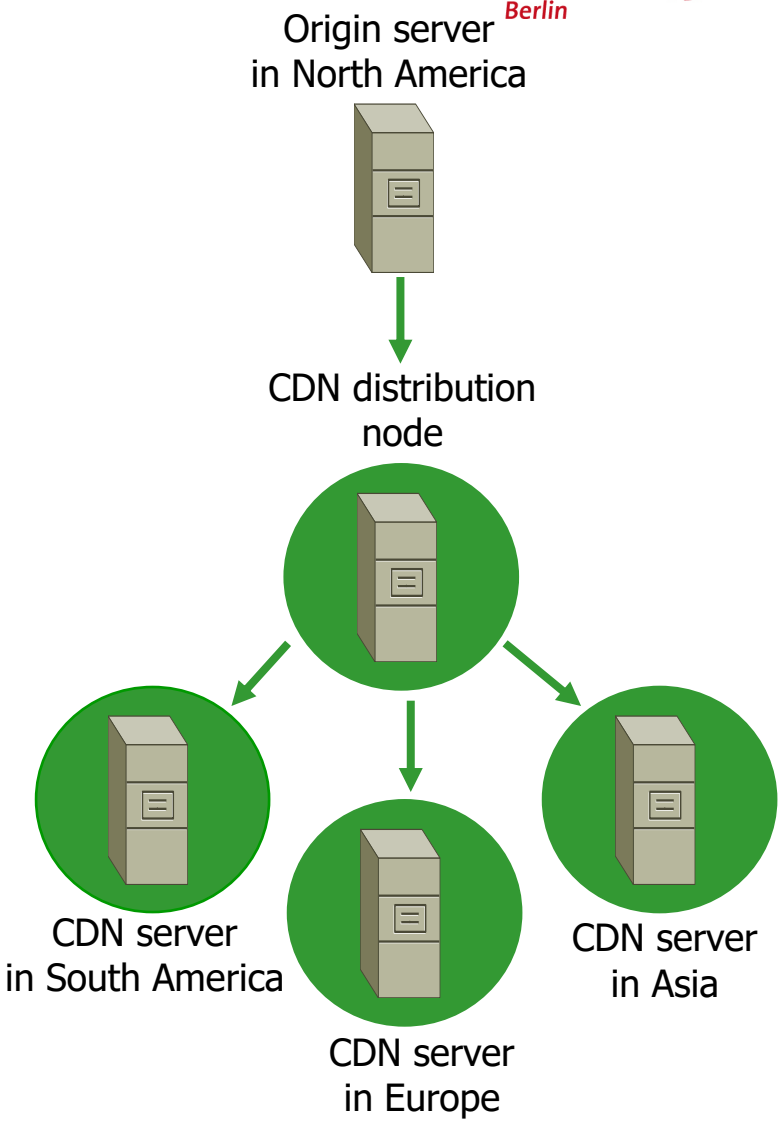
HTTP: Caching

- 2. Lösung: Verwendung eines Caches
  - Annahme: Cache-Hitrate ist 0,4
  - realistisch: 40 % der abgefragten Seiten befinden sich langfristig im Cache, 60% müssen bei HTTP-Servern angefordert werden
  - Folgen
    - Auslastung des LANs = 15 %
    - Auslastung der Zugangsleitung
      $0{,}6 \cdot 15/s \cdot 100.000$ Bits /
      $1{,}5 \cdot 10^6$ Bits/s = 0,6 = 60 %
    - Gesamtverzögerung = Verzögerung im LAN + beim Zugang + im Internet = ms + ms + $0{,}6 \cdot 2$ s < 2 s



Content Distribution Networks

- Content Distribution Networks (CDNs)
  - Ziel: Vermeiden längerer Wartezeiten beim Laden von Web-Seiten, z.B. bei Flash-Crowds (Millionen Benutzer greifen auf eine Seite zu)
  - 3 Engpässe: erste Meile, letzte Meile, Peering-Punkte (Übergänge zwischen ISPs)
  - Idee: sehr viele (Hunderte) Spiegel-Server geografisch verteilen (diese sind wie Web-Caches, der Inhalt wird aber proaktiv auf sie repliziert)
  - bekannte CDNs: Akamai, Digital Island

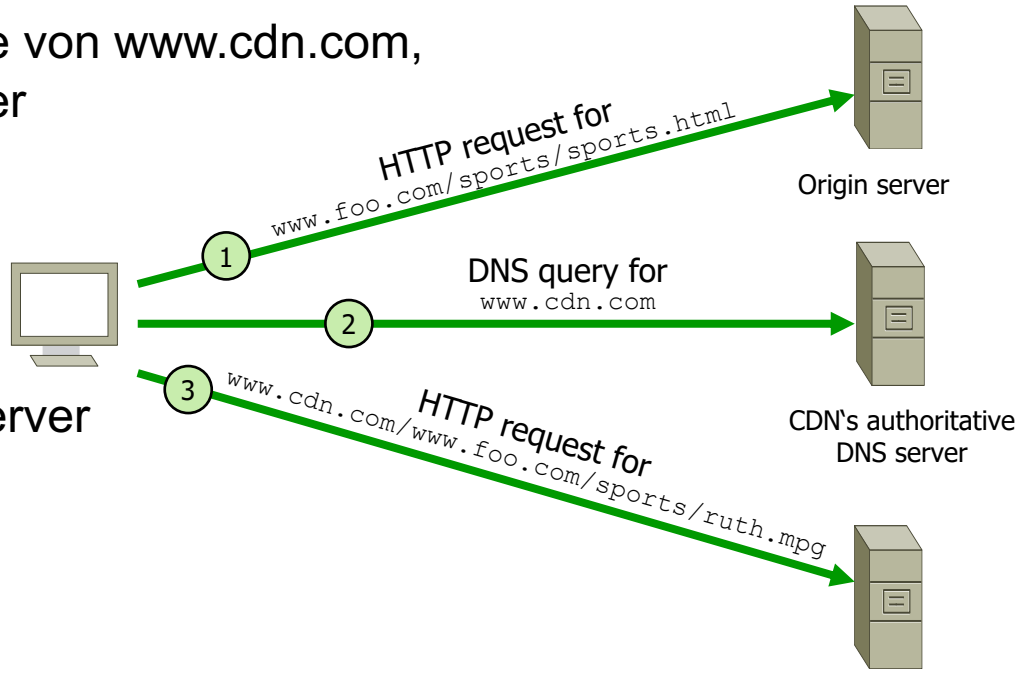


Content Distribution Networks

- Verteilung der Anfragen
  - **Server-basierte HTTP Redirection**: Server liefert aufgrund der IP-Adresse des Clients einen geeigneten anderen Server, erfordert zusätzliche RTT, Gefahr der Überlast für Server
  - **Client-nahe HTTP-Redirection**: z.B. durch Web-Proxy, schwieriger zu verwirklichen
  - **DNS-basierte Redirection**: DNS-Server bildet den Domain-Namen des Servers auf die IP-Adresse eines geeigneten Servers ab
  - **URL-Rewriting**: Server liefert Basisseite, die URLs der eingebetteten Objekte werden umgeschrieben, mit dem Domain-Namen eines geeigneten anderen Servers
  - kommerzielle CDNs verwenden meist Kombination aus DNS-basierter Redirection und URL-Rewriting

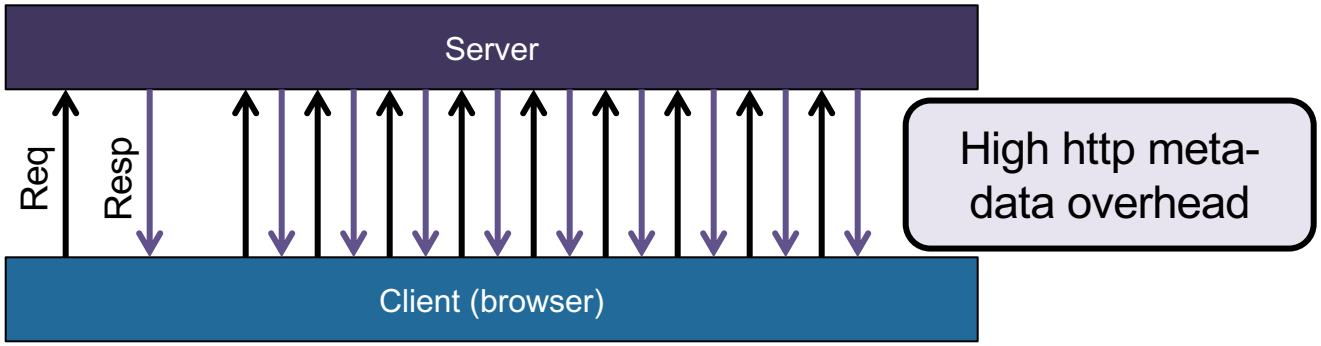
Content Distribution Networks

- Bsp. für URL-Rewriting
  - www.foo.com ist Content-Provider, Video-Dateien werden auf den CDN-Servern von cdn.com verteilt
  - 1. HTTP-Anfrage für HTML-Basisseite, Antwort enthält eingebettete Video-Datei www.cdn.com/www.foo.com/sports/ruth.mpg
  - 2. DNS-Anfrage für IP-Adresse von www.cdn.com, DNS-Server liefert aufgrund der IP-Adresse des Clients und einer internen Tabelle die P-Adresse eines geeigneten Servers
  - 3. HTTP-Anfrage an diesen Server



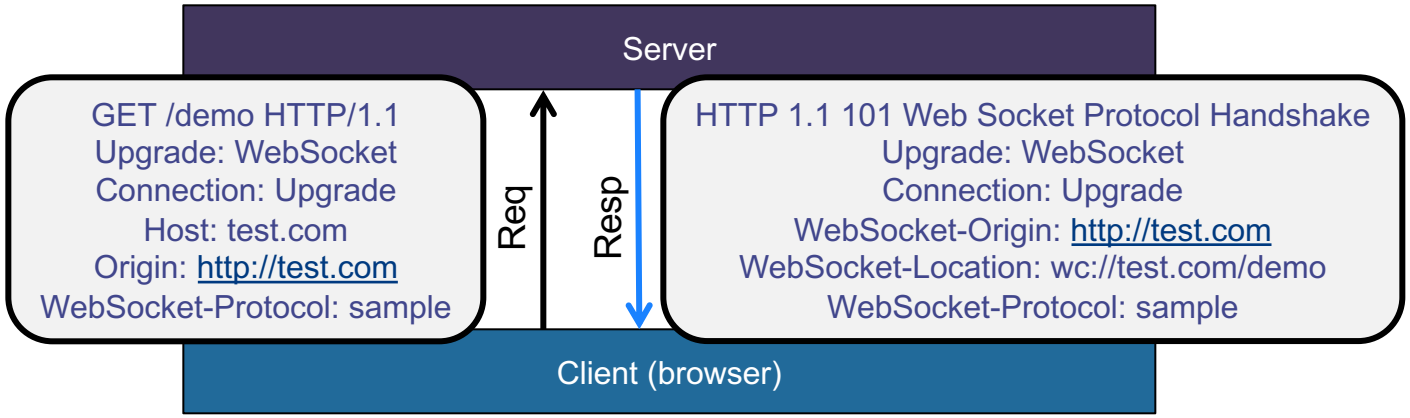
HTML5 WebSocket API

- Why do we need WebSockets? (or something similar!)
  - Some web apps demand real-time, event-driven communication with minimal latency
    - E.g. financial applications, online games, …
  - Problems with HTTP
    - Request-reply pattern with half duplex data flow (traffic flow in only one direction at a time)
    - Adds latency
- Typical use case: polling
  - Poll server for updates, wait at client



What is a WebSocket (WS)?

- W3C/IETF standard
- Uses Websocket protocol instead of HTTP: ws:// and wss://
- Establishes a true full-duplex communication channel
  - Strings + binary frames sent in any direction at the same time
- Uses port 80 (http) or 443 (https) to traverse firewalls (-> proxy/firewall)
- Connection established by „upgrading“ from HTTP to Websocket protocol



WebSockets Advantages

- … as compared to „bare“ sockets
- Traversing firewalls (avoids blocking of the port)
- “The name resolution” and agreement on port implicit
- After establishing the connection server can send info to client anytime:
  - Update latency reduced
  - No polling = reduced data traffic! (but might require client heartbeat)
  - Each message frame has only 2 Bytes of overhead
- Server handles fewer transport level connection requests
  - Might reset the connection if no new info's for the „client“

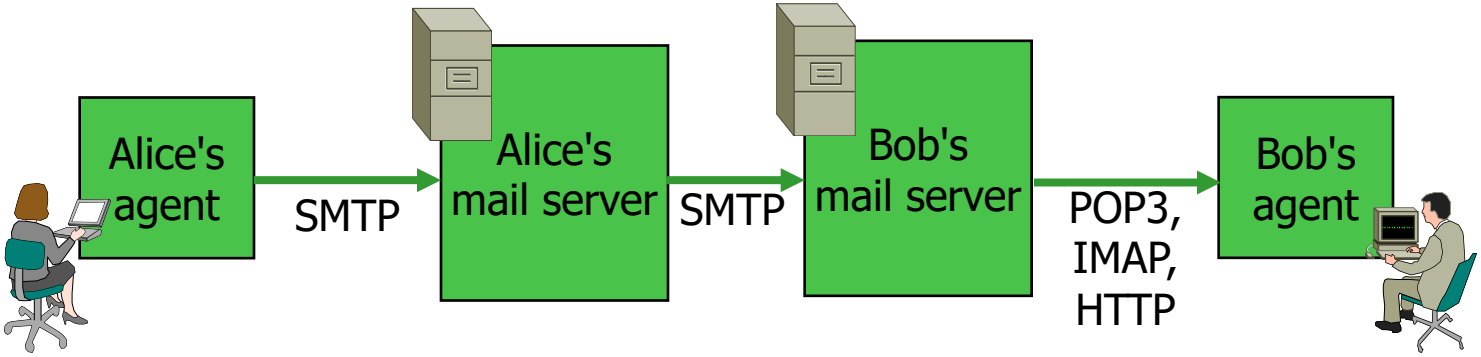
Add-on: HTTP/2

- Derived from the earlier experimental SPDY protocol (→ Google),
- Published as RFC 7540 in 2015,
- Data compression of HTTP headers
- HTTP/2 Server Push
- Pipelining of requests
- Fixing the head-of-line blocking problem in HTTP 1.x
- Multiplexing multiple requests over a single TCP connection

E-Mail

E-Mail

- Simple Mail Transfer Protocol (SMTP)
  - Nachrichten im ASCII-Format, Kopf, Rumpf
  - andere Daten (Word-Dateien u.ä.) werden in ASCII umgewandelt angehängt: multimedia mail extension (MIME)
  - Versenden mit SMTP über TCP (lesbar)
  - Abholen mit POP3, IMAP, HTTP (lesbar)
  - mehr Einzelheiten in der Übung



E-Mail: SMTP [RFC 821]

- nutzt TCP zur zuverlässigen Übertragung der Nachrichten vom Client zum Server, dazu wird Port 25 verwendet.
- direkte Übertragung: vom sendenden Server zu empfangendem Server
- drei Phasen der Übertragung
  - Handshaking (Begrüßung)
  - Nachrichtenübertragung
  - Abschlussphase
- Interaktion mittels Befehlen und Antworten
  - Befehle: ASCII-Text
  - Antworten: Statuscode und Text
- Nachrichten müssen **7-bit ASCII-Text** sein

Beispiel für einen SMTP-Dialog

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C:   How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

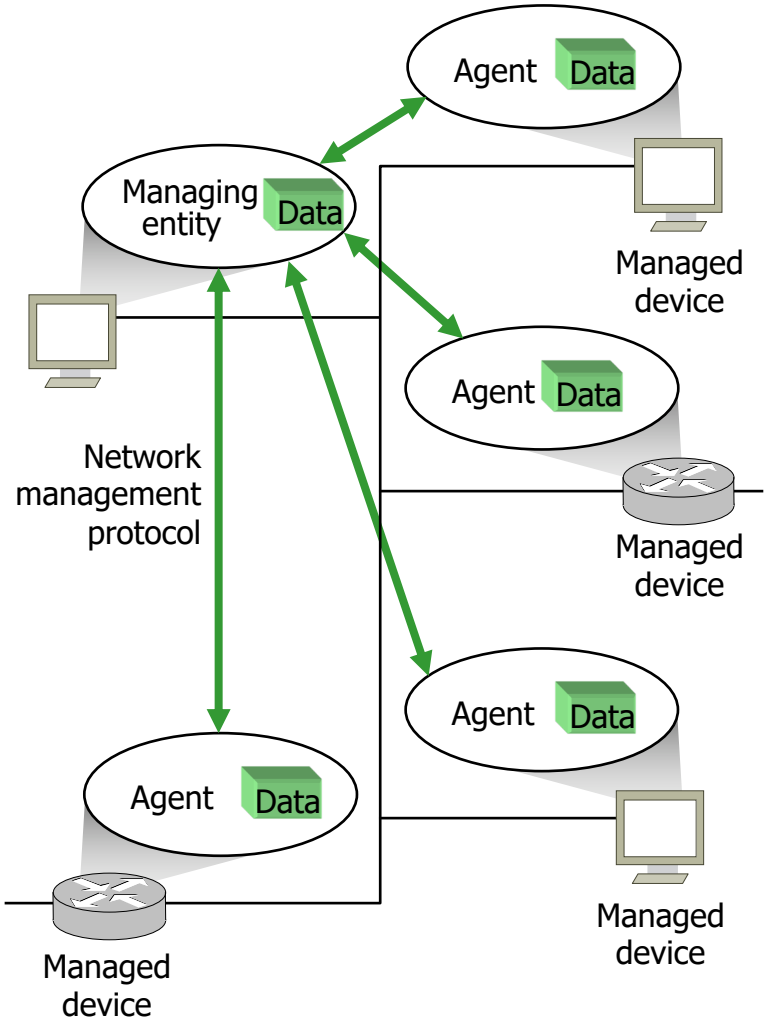
Netzwerkmanagement

Netzwerkmanagement

- Aufgaben des Netzwerkmanagements
  - Überwachung und Verwaltung eines Netzwerks = komplexes HW/SW-Gebilde (zahlreiche Geräte, Leitungen, Datenstrukturen, …)
  - nach ISO 5 Einsatzbereiche
    - **Leistung**: Monitoring von Auslastung, Durchsatz, Antwortzeiten, Dokumentation (z.B. für die Überwachung von **Service Level Agreements**), Reaktionsmaßnahmen
    - **Fehler**: Monitoring, Dokumentation, Reaktionsmaßnahmen
    - **Konfiguration**: Übersicht über Geräte und deren HW/SW-Konfigurationen
    - **Zugang**: Festlegung, Kontrolle, Dokumentation des Zugangs von Benutzern und Geräten
    - **Sicherheit**: Monitoring und Kontrolle des Zugangs, Schlüsselverwaltung, z.B. Filterregeln für Firewalls,
      Intrusion Detection
  - diverse komplexe Standards, z.B. TMN, TINA

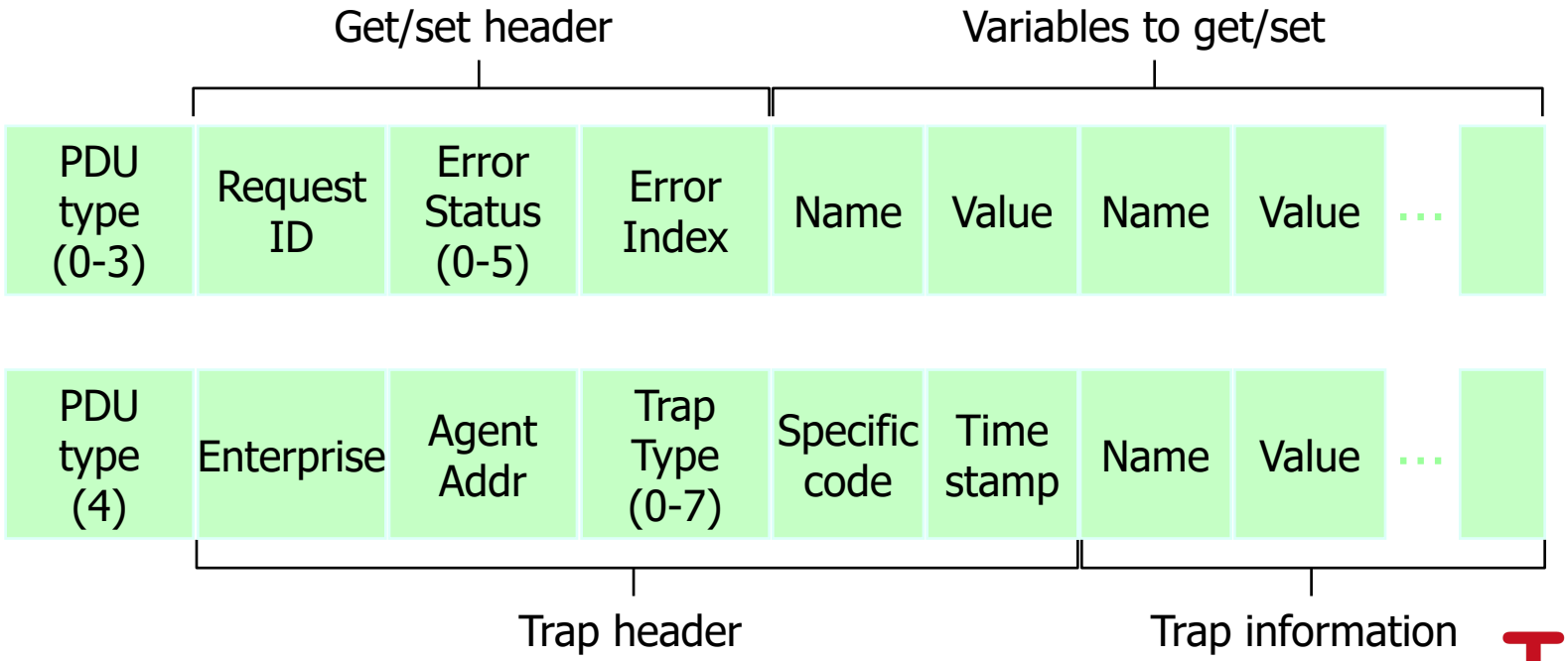
Netzwerkmanagement

- Simple Network Management Protocol (SNMP)
  - einfach und verbreitet
  - **Managing Entity**, Prozeß auf zentraler Management Station, ≈ Client
  - **Managed Device**, Gerät im Netz
  - **Managed Object**, HW oder SW im Managed Device, z.B. Routing-Tabelle
  - **Management Agent**, Prozeß auf Managed Device, kann lokale Aktionen ausführen, ≈ Server
  - Anfrage/Antwort-Protokoll zwischen Management Entity und Management Agent über UDP



Netzwerkmanagement

- SNMP Nachrichten (Version 2)
  - **GET**: Anfrage der Managing Entity einer Variable des Managed Objects
  - **SET**: Setzen der Variable eines Managed Objects durch Managing Entity
  - auch **GET-NEXT** and **GET-BULK** für Datenstrukturen
  - **TRAP**: Nachricht des Managing Agents über Fehlersituation

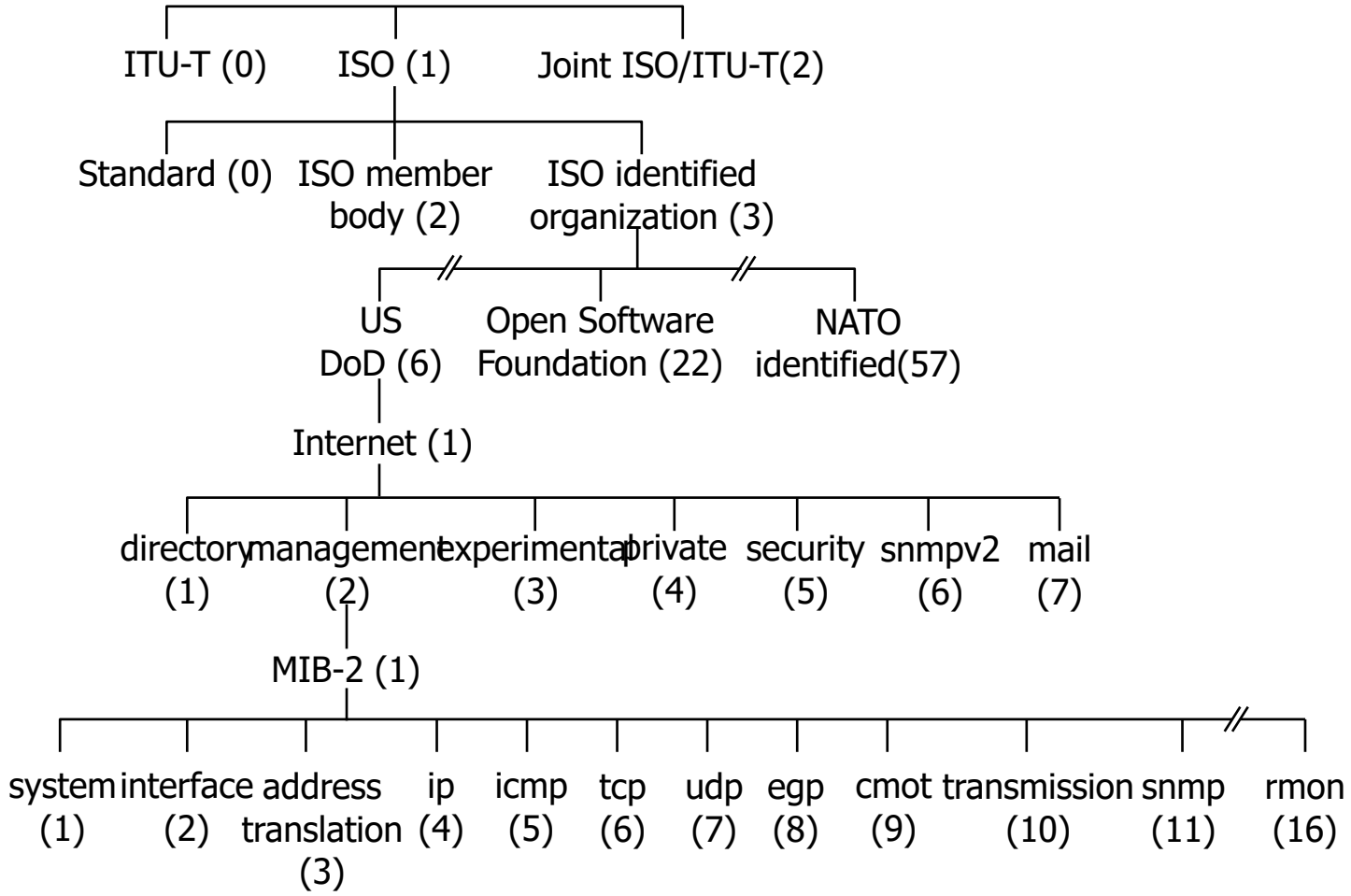


Netzwerkmanagement

- Management Information Base (MIB)
  - **MIB-Module** enthalten Datenstrukturen für die Managed Objects, von der IETF genormt
  - Syntax wird in Structure of Management Information (SMI) der IETF festgelegt, die wiederum die **Abstract Syntax Notation One (ASN.1)** der ISO benutzt (im wesentlichen wie C ohne Referenzen)
  - ASN.1 besitzt auch ein Nummerierungsschema zur eindeutigen Objekt-Identifizierung, damit wird jedes MIB-Modul eindeutig bezeichnet
  - mit den Basic Encoding Rules (BER) wird noch das genaue binäre Format für die Übertragung festgelegt

Netzwerkmanagement

- Nummerierung von ASN.1



- MIB-Modul für UDP

Object Identifier	Name	Type	Description (from RFC 2013)
1.3.6.1.2.1.7.1	`udpInDatagrams`	Counter32	"total number of UDP datagrams delivered to UDP users"
1.3.6.1.2.1.7.2	`udpNoPorts`	Counter32	"total number of received UDP datagrams for which there was no application at the destination port"
1.3.6.1.2.1.7.3	`udpInErrors`	Counter32	"number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port"
1.3.6.1.2.1.7.4	`udpOutDatagrams`	Counter32	"total number of UDP datagrams sent from this entity"
1.3.6.1.2.1.7.5	`udpTable`	SEQUENCE of UdpEntry	"a sequence of UdpEntry objects, one for each port that is currently open by an application, giving the IP address and the port number used by application"

Netzwerkmanagement

- Basic Encoding Rules (BER)
  - Repräsentation zur Übertragung
  - **Tag, Length, Value (TLV)**
    - Tag = Nummer für Typ
    - Length = Länge in Bytes
  - Übertragung von „smith“
    - Tag 4 für OCTET STRING
    - Length 5
    - ASCII-Werte der Zeichen
  - Übertragung von 282
    - Tag 2 für INTEGER
    - Length 2
    - 0x011a (hexadezimal), höherwertiges Byte zuerst („Big Endian“)