Quelle: Computernetzwerke: Der Top-Down-Ansatz, 6. Ausgabe
Jim Kurose, Keith Ross; Pearson, März 2014

Technische Grundlagen der Informatik 2 – Teil 2: Layer 7

Philipp Rettberg / Sebastian Harnau

Block 3/18

Anwendungsschicht (Layer 7)

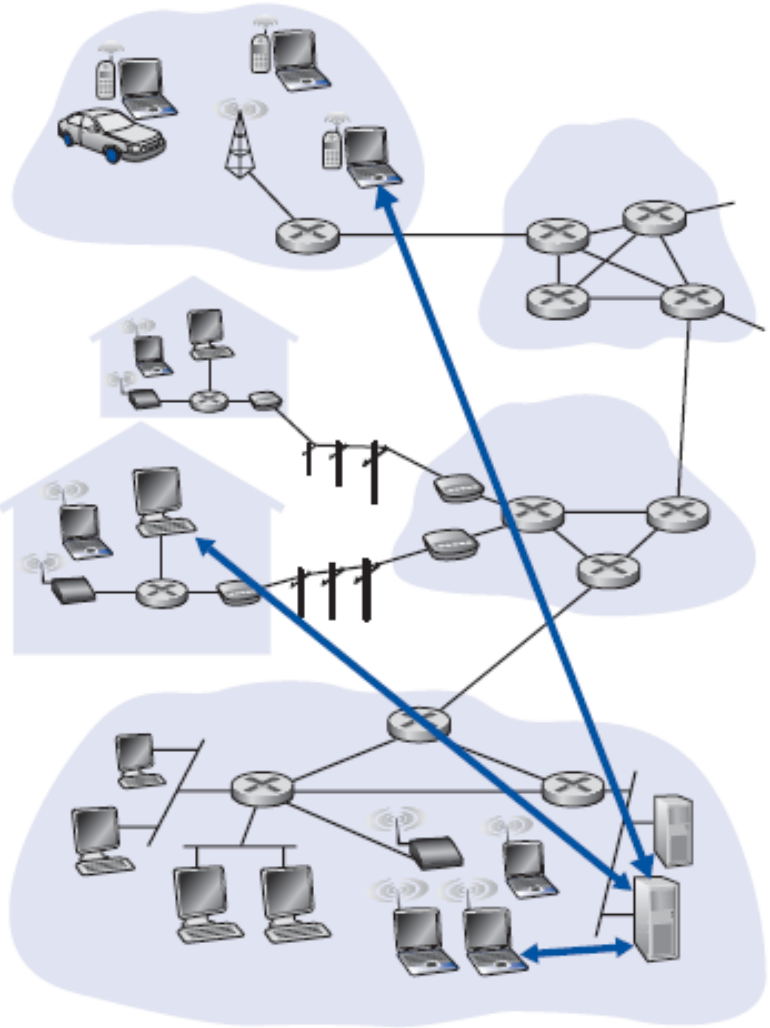
Client-Server-Architektur

Server (Anbieter):

- Hoch verfügbar (24/7)
- Definierter Name (Adresse)
- Eventuell Serverfarmen, um zu skalieren

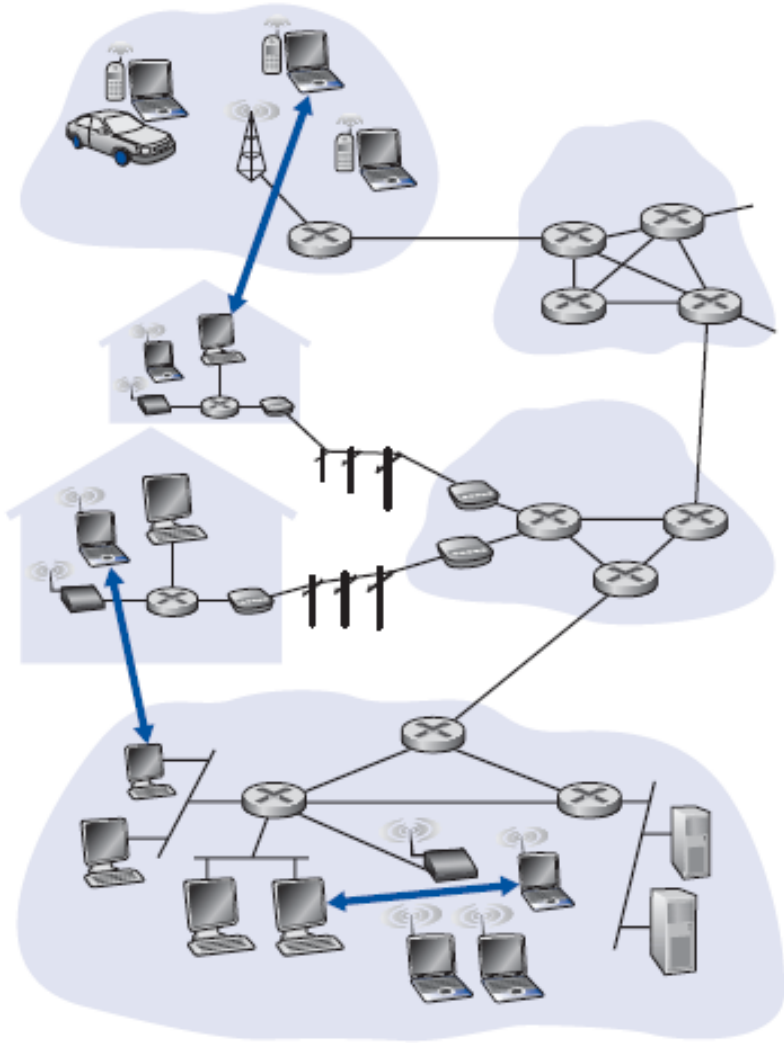
Clients (Konsumenten):

- Kommunizieren mit dem Server
- Sporadisch angeschlossen
- Keine festen Adressen
- Kommunizieren immer mit dem Server



Reine Peer-To-Peer (P2P)-Architektur

- Keine externen Server
- Beliebige Endsysteme kommunizieren direkt miteinander
- Peers sind sowohl Anbieter als auch Konsument
- Peers sind nur sporadisch angeschlossen und können Ihre Bezeichnung wechseln
- Zahl der Peers nicht durch Server-Restriktionen begrenzt.



Kombination von Client-Server und P2P

- Skype
  - P2P-Anwendung für Voice-over-IP
  - Zentraler Server: Adresse des Kommunikationspartners finden
  - Verbindung zwischen Clients: direkt (nicht über einen Server)

- Instant Messaging
  - Chat zwischen zwei Benutzern: P2P
  - Zentralisierte Dienste: Erkennen von Anwesenheit, Zustand, Aufenthaltsort eines Anwenders
  - Benutzer registriert seine aktuelle Adresse beim Server, sobald er sich mit dem Netz verbindet
  - Benutzer fragt beim Server nach Informationen über seine Freunde und Bekannten

Kommunizierende Prozesse

- Prozess: Programm, welches auf einem Host läuft
- Innerhalb eines Hosts können zwei Prozesse mit Inter-Prozess-Kommunikation Daten austauschen (durch das Betriebssystem unterstützt)
- Prozesse auf verschiedenen Hosts kommunizieren, indem sie Nachrichten über ein Netzwerk austauschen

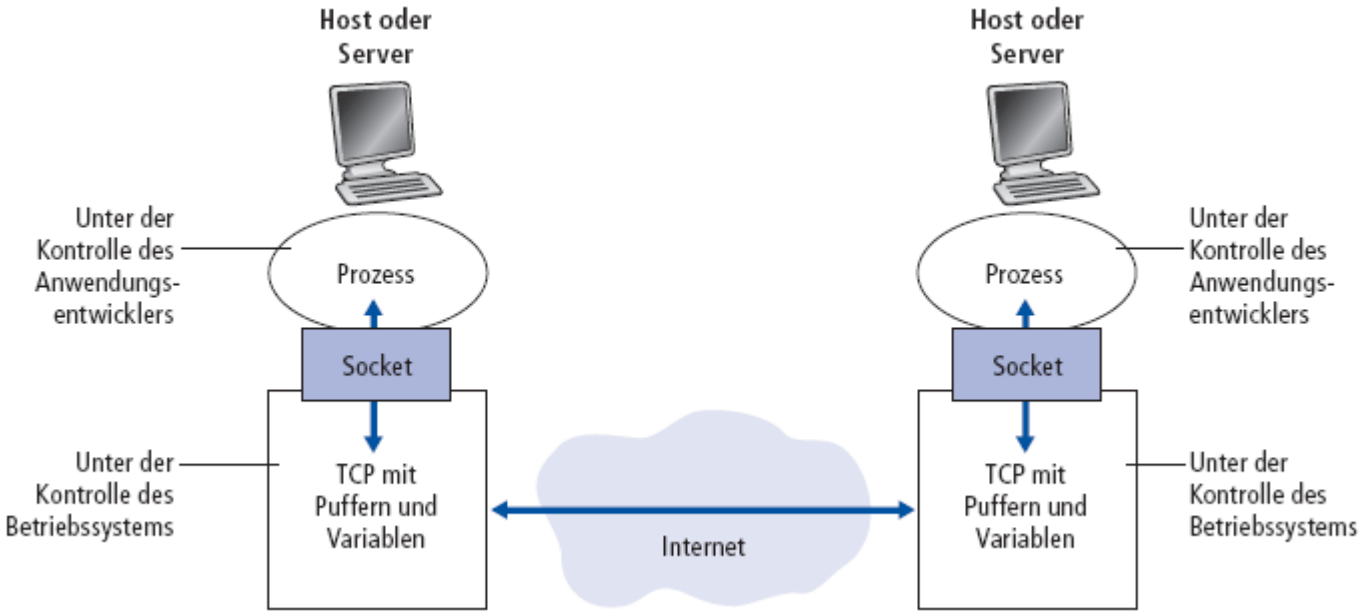
Client-Prozess: Prozess, der die Kommunikation beginnt

Server-Prozess: Prozess, der darauf wartet, kontaktiert zu werden

Anmerkung: Anwendungen mit einer P2P-Architektur haben Client- und Server-Prozesse

Sockets

- Prozesse senden/empfangen Nachrichten über einen Socket
- Ein Socket lässt sich mit einer Tür zwischen den Layern vergleichen :
  - Der sendende Prozess schiebt die Nachrichten durch die Tür
  - Der sendende Prozess verlässt sich auf die Transportinfrastruktur auf der anderen Seite der Tür, um die Nachricht zum Socket des empfangenden Prozesses zu bringen



API:

- Wahl des Transportprotokolls
- Einstellen einiger Parameter

Bekannte Protokolle je Layer

Öffentlich verfügbare Protokolle:
- Definiert in RFCs
- Ermöglichen Interoperabilität
- z.B. HTTP, FTP, SMTP, IMAP, DNS

Proprietäre Protokolle:
- z.B. Skype

- TCP
- UDP

- IP
- ICMP

- Ethernet

- IEEE 802.11

Anwendungsprotokolle bestimmen …

Arten von Nachrichten
- z.B. Request, Response

Syntax der Nachrichten
- Welche Felder sind vorhanden und wie werden diese voneinander getrennt?

Semantik der Nachrichten
- Bedeutung der Informationen in den Feldern

Regeln für das Senden von und Antworten auf Nachrichten

Adressierung von Prozessen

- Um eine Nachricht empfangen zu können, muss ein Prozess identifiziert werden können
- Ein Host wird durch eine IP-Adresse identifiziert:
  - IPv4 -> 32 Bit
  - IPv6 –> 128 Bit
- Ein Host kann mehrere IP-Adressen haben.
- Die IP-Adresse ist mindestens im lokalen Netzwerk eindeutig. Zum ISP hin kann sie maskiert sein (NAT=Network Address Translation).

- Prozesse werden durch eine IP-Adresse UND eine Portnummer identifiziert
- Beispiel-Portnummern:
  - HTTP-Server: 80
  - E-Mail-Server: 25
- Um an den Webserver gaia.cs.umass.edu eine HTTP-Nachricht zu schicken:
  - IP-Adresse: 128.119.245.12
  - Portnummer: 80

```
C:\Users\Fam>ping 128.119.245.12

Ping wird ausgeführt für 128.119.245.12 mit 32 Bytes Daten:
Antwort von 128.119.245.12: Bytes=32 Zeit=101ms TTL=46
Antwort von 128.119.245.12: Bytes=32 Zeit=98ms TTL=46
Antwort von 128.119.245.12: Bytes=32 Zeit=96ms TTL=46
Antwort von 128.119.245.12: Bytes=32 Zeit=96ms TTL=46

Ping-Statistik für 128.119.245.12:
    Pakete: Gesendet = 4, Empfangen = 4, Verloren = 0
    (0% Verlust),
Ca. Zeitangaben in Millisek.:
    Minimum = 96ms, Maximum = 101ms, Mittelwert = 97ms

C:\Users\Fam>nslookup 128.119.245.12
Server:  fritz.box
Address:  192.168.178.1

Name:    gaia.cs.umass.edu
Address:  128.119.245.12
```

Wahl des Transportdienstes

Bandbreite

- Einige Anwendungen (z.B. Multimedia-Streaming) brauchen eine Mindestbandbreite, um zu funktionieren.
- Andere Anwendungen verwenden einfach die verfügbare Bandbreite (bandbreitenelastische Anwendungen).

Datenverlust

- Einige Anwendungen können Datenverlust tolerieren (z.B. Audioübertragungen) .
- Andere Anwendungen benötigen einen absolut zuverlässigen Datentransfer (z.B. Dateitransfer).

Zeitanforderungen

- Einige Anwendungen (z.B. Internettelefonie oder Netzwerkspiele) tolerieren nur eine sehr geringe Verzögerung

Beispiele für Anforderungen von Anwendungen

Anwendung	Datenverlust	Bandbreite	Echtzeit
Dateitransfer	Kein Verlust	Elastisch	Nein
E-Mail	Kein Verlust	Elastisch	Nein
Web	Kein Verlust	Elastisch (wenige Kbps)	Nein
Internettelefonie/ Bildkonferenz	Toleriert Verluste	Audio: wenige Kbps bis 1 Mbps Video: 10 Kbps bis 5 Mbps	Ja: einige Hundert ms
Gespeichertes Audio/Video	Toleriert Verluste	Wie oben	Ja: wenige Sekunden
Interaktive Spiele	Toleriert Verluste	Wenige Kbps bis 10 Kbps	Ja: einige Hundert ms
Instant Messaging	Kein Verlust	Elastisch	Ja und nein

Dienste der Transportprotokolle

TCP-Dienste:

- Verbindungsorientierung: Herstellen einer Verbindung zwischen Client und Server
- Zuverlässiger Transport zwischen sendendem und empfangendem Prozess
- Flusskontrolle: Sender überlastet den Empfänger nicht
- Überlastkontrolle: Bremsen des Senders, wenn das Netzwerk überlastet ist
- Nicht: Zeit- und Bandbreitengarantien

UDP-Dienste:

- Unzuverlässiger Transport von Daten zwischen Sender und Empfänger
- Nicht: Verbindungsorientierung, Zuverlässigkeit, Flusskontrolle, Überlastkontrolle, Zeit- oder Bandbreitengarantien

	Datenverlust	**Bandbreite**	**Echtzeit**
UDP	Unzuverlässiger Transport von Daten zwischen Sender und Empfänger	Keine Bandbreiten-garantien	Kein Zeitgarantien, aber schneller als TCP
TCP	Zuverlässiger Transport zwischen sendendem und empfangendem Prozess		Kein Zeitgarantien

Beispiele aus dem Internet

Anwendung	Anwendungsschichtprotokoll	Zugrunde liegendes Transportprotokoll
E-Mail-Dienst	SMTP [RFC 2821]	TCP
Remote-Terminalzugang	Telnet [RFC 854]	TCP
World Wide Web	HTTP [RFC 2616]	TCP
Dateitransfer	FTP [RFC 959]	TCP
Multimedia-Streaming	HTTP (z. B. YouTube), RTP	TCP oder UDP
Internettelefonie	SIP, RTP oder proprietär (z. B. Skype)	Normalerweise UDP

Rekapitulation: HTML / Webseitenaufbau

- Was ist HTML?
- Wie ist eine Webseite aufgebaut?
- Woraus bestehen heutige Webseiten?

Funktionsweise des Internets

https://www.youtube.com/watch?v=8PNRrOGJqUI

Web und HTTP

Einige Definitionen

- Eine Webseite besteht aus Objekten
- Objekte können sein: HTML-Dateien, JPEG-Bilder, Java-Applets, Audiodateien, …
- Eine Webseite hat eine Basis-HTML-Datei, die mehrere referenzierte Objekte beinhalten kann.
- Jedes Objekt kann durch eine URI, hier URL (Uniform Ressource Locator), adressiert werden.

Beispiel für eine URL:

`www.someschool.edu/someDept/pic.gif`

Hostname | Pfad

HTTP: Überblick und Beispiele

HTTP: HyperText Transfer Protocol

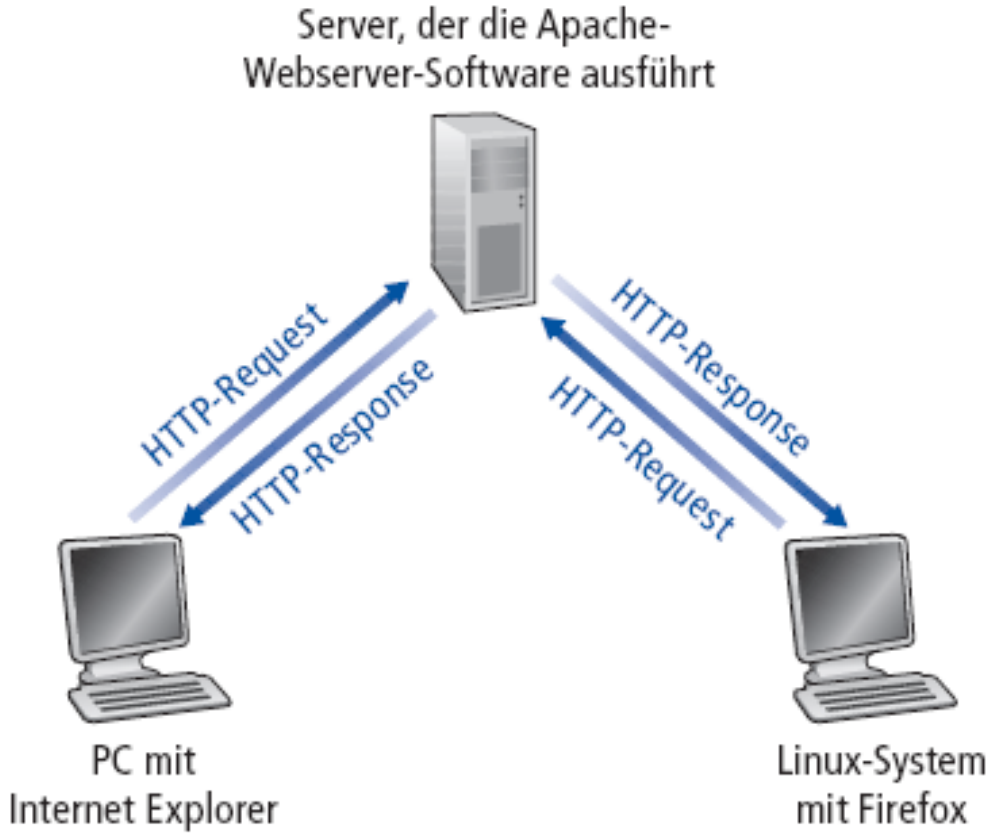
- Anwendungsprotokoll des Web

Client/Server-Modell

- Client: Browser, der Objekte anfragt, erhält und anzeigt
- Server: Webserver verschickt Objekte auf Anfrage

HTTP 1.0: RFC 1945

HTTP 1.1: RFC 2068



HTTP: Überblick (Fortsetzung)

Verwendet TCP:

- Client baut mit der Socket-API eine TCP-Verbindung zum Server auf
- Server wartet auf Port 80
- Server nimmt die TCP-Verbindung des Clients an
- HTTP-Nachrichten (Protokollnachrichten der Anwendungsschicht) werden zwischen Browser (HTTP-Client) und Webserver (HTTP-Server) ausgetauscht
- Die TCP-Verbindung wird geschlossen

HTTP ist “zustandslos”

- Server merkt sich keine Informationen über frühere Anfragen von Clients

Protokolle, die einen Zustand verwalten, sind komplex!

- Der Zustand muss gespeichert und verwaltet werden
- Wenn Server oder Client abstürzen, dann muss der Zustand wieder synchronisiert werden

HTTP-Verbindungen

Nichtpersistentes HTTP

Maximal ein Objekt wird über eine TCP-Verbindung übertragen

HTTP/1.0 verwendet nichtpersistentes HTTP

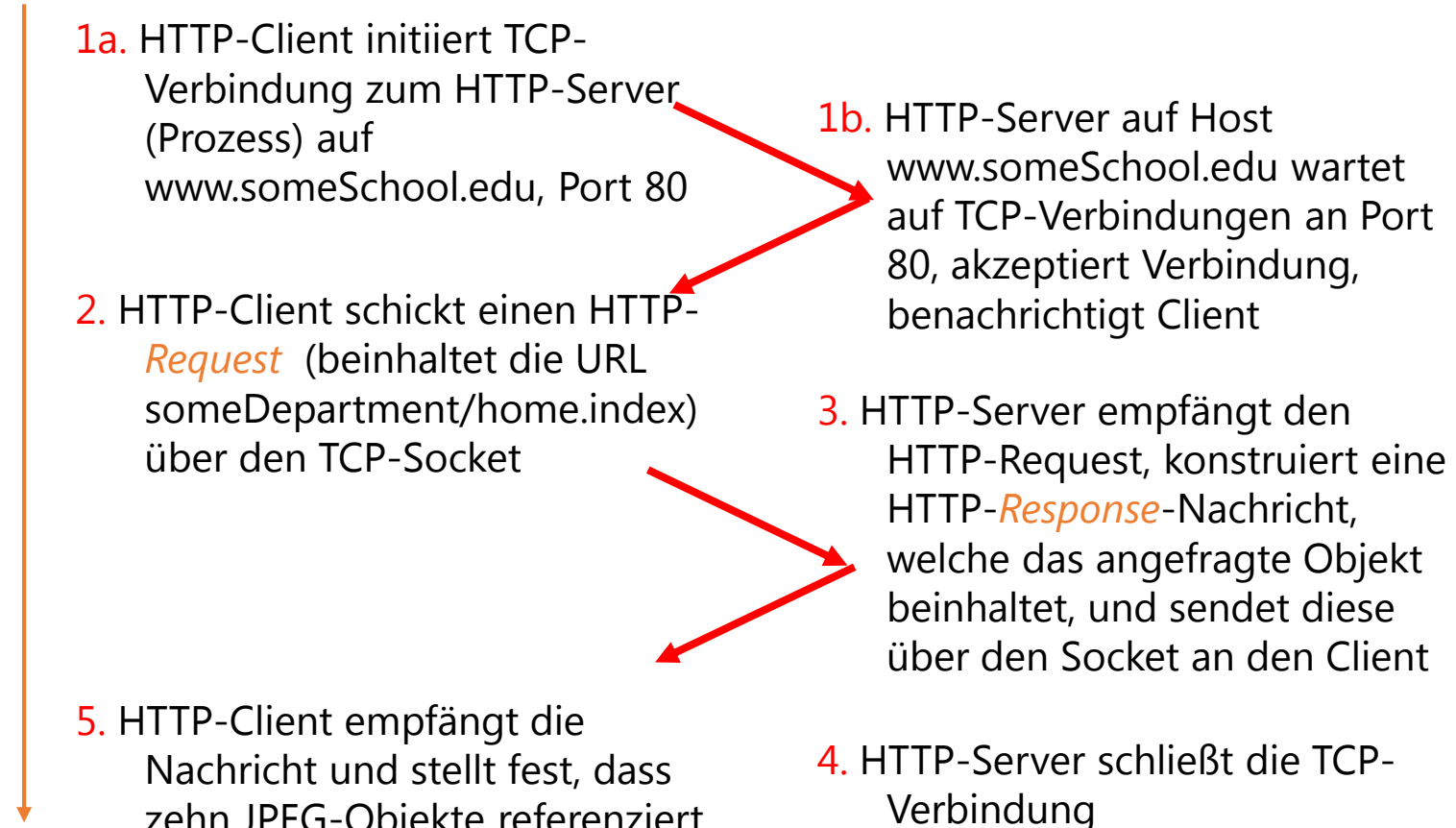
Persistentes HTTP

Mehrere Objekte können über eine TCP-Verbindung übertragen werden

HTTP/1.1 verwendet standardmäßig persistentes HTTP

Nichtpersistentes HTTP

Es soll folgende URL geladen werden:
`www.someSchool.edu/someDepartment/home.index`



6. Schritte 1 bis 5 werden für jedes der zehn JPEG-Objekte wiederholt, dann kann die Seite vollständig angezeigt werden

Nichtpersistentes HTTP: Verzögerung

Definition von RTT (Round Trip Time):
Zeit, um ein kleines Paket vom Client zum Server und zurück zu schicken

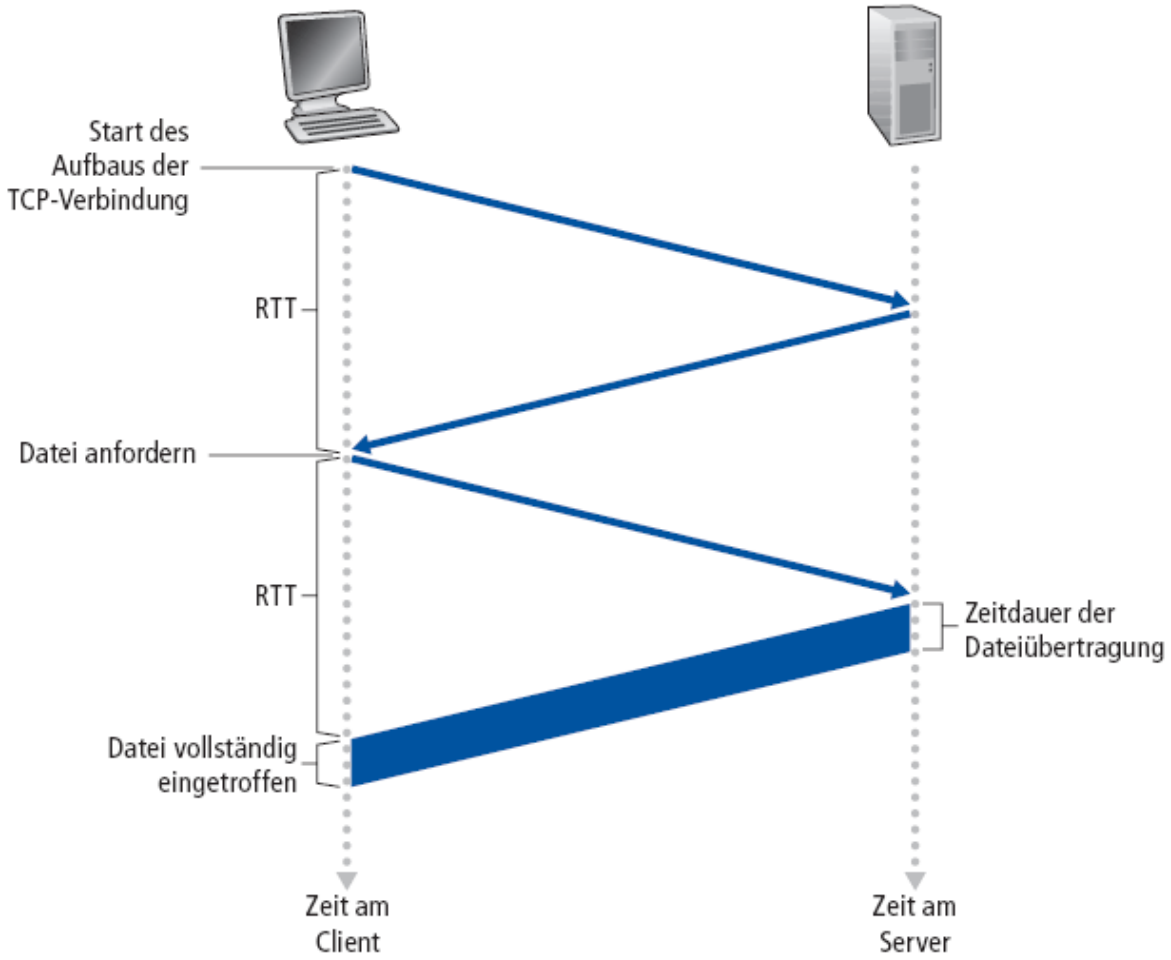
Verzögerung:

Eine RTT für den TCP-Verbindungsaufbau

Eine RTT für den HTTP-Request, bis das erste Byte der HTTP-Response beim Client ist

Zeit für das Übertragen der Daten auf der Leitung

Zusammen = 2 RTT + Übertragungsverzögerung



Persistentes HTTP

Probleme mit nichtpersistentem HTTP:

2 RTTs pro Objekt

Aufwand im Betriebssystem für jede TCP-Verbindung

Browser öffnen oft mehrere parallele TCP-Verbindungen, um die referenzierten Objekte zu laden

Persistentes HTTP

Server lässt die Verbindung nach dem Senden der Antwort offen

Nachfolgende HTTP-Nachrichten können über dieselbe Verbindung übertragen werden

Persistent ***ohne*** Pipelining:

- Client schickt neuen Request erst, nachdem die Antwort auf den vorangegangenen Request empfangen wurde
- Eine RTT für jedes referenzierte Objekt

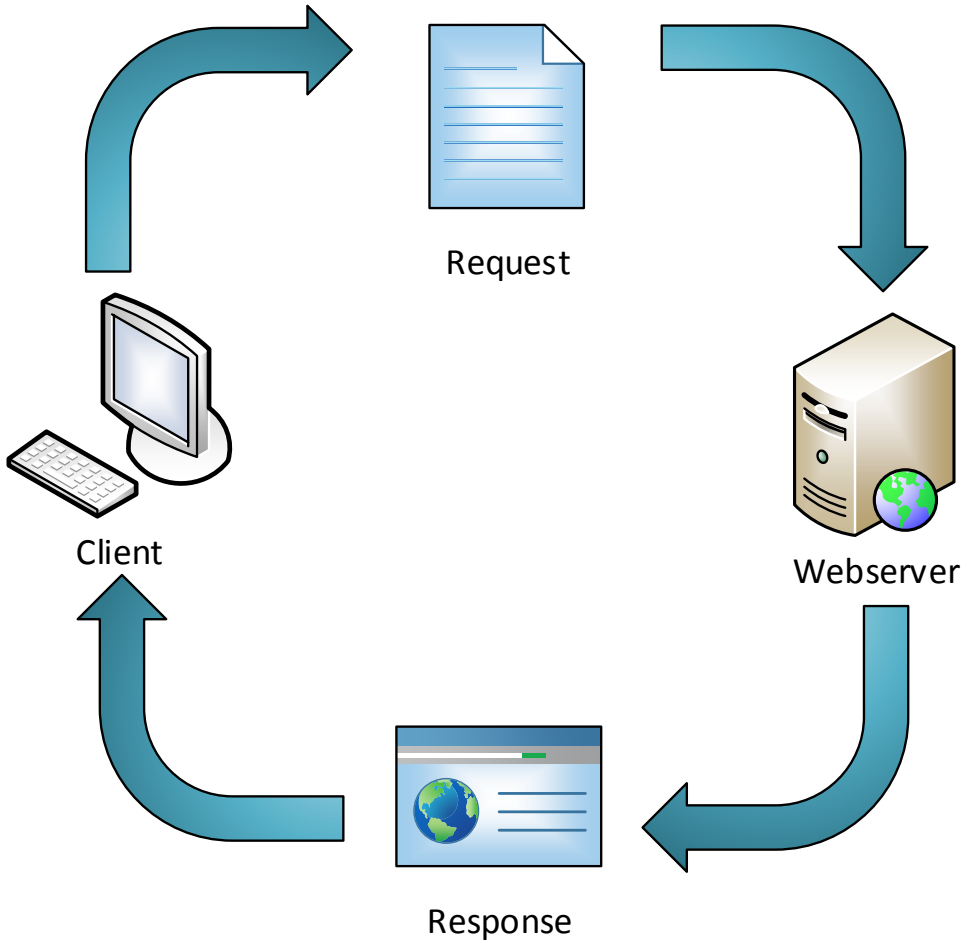
Persistent mit Pipelining:

- Standard in HTTP/1.1
- Client schickt Requests, sobald er die Referenz zu einem Objekt findet
- Idealerweise wird nur wenig mehr als eine RTT für das Laden aller referenzierten Objekte benötigt

HTTP/2

- Wie geht es weiter nach HTTP 1.1?
- HTTP/2 beschleunigt die Übertragungen durch:
    - Multiplexing von Anfragen
    - Kompressionsmöglichkeiten für Header-Informationen
    - Möglichkeit der Binärkodierung von Inhalten
    - Server-Push

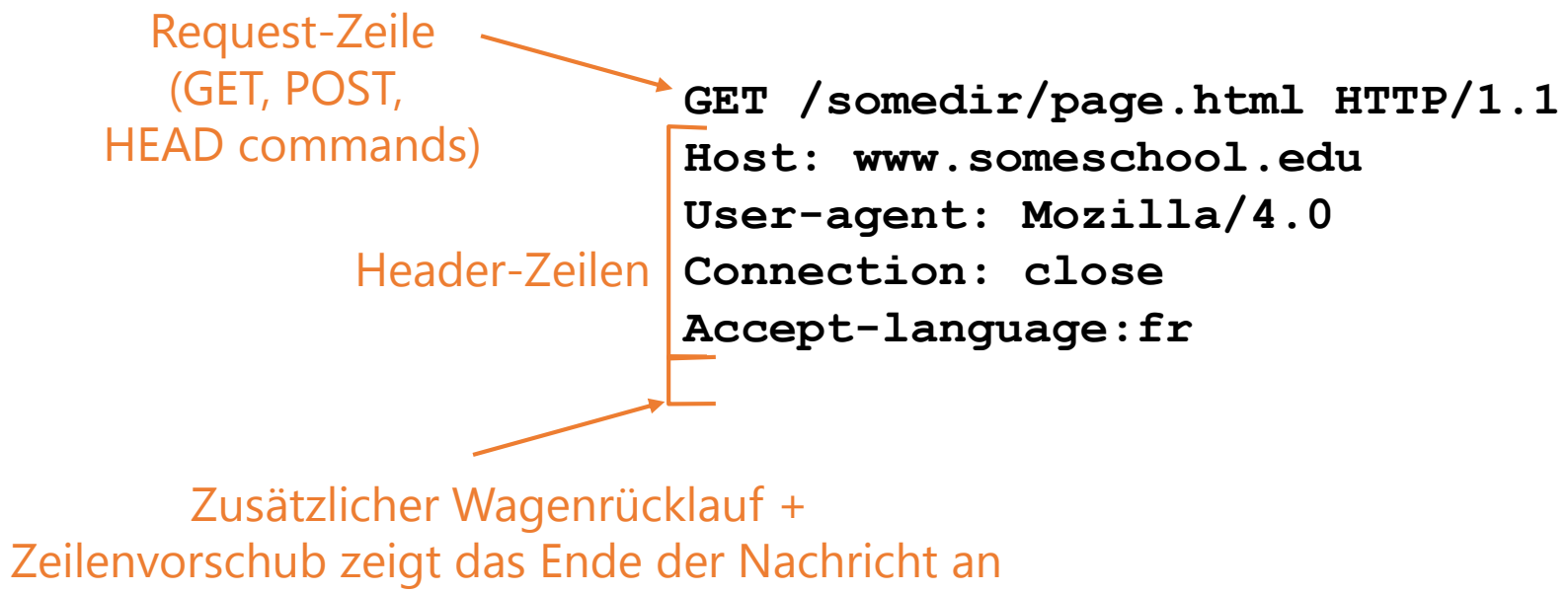
HTTP-Nachrichten



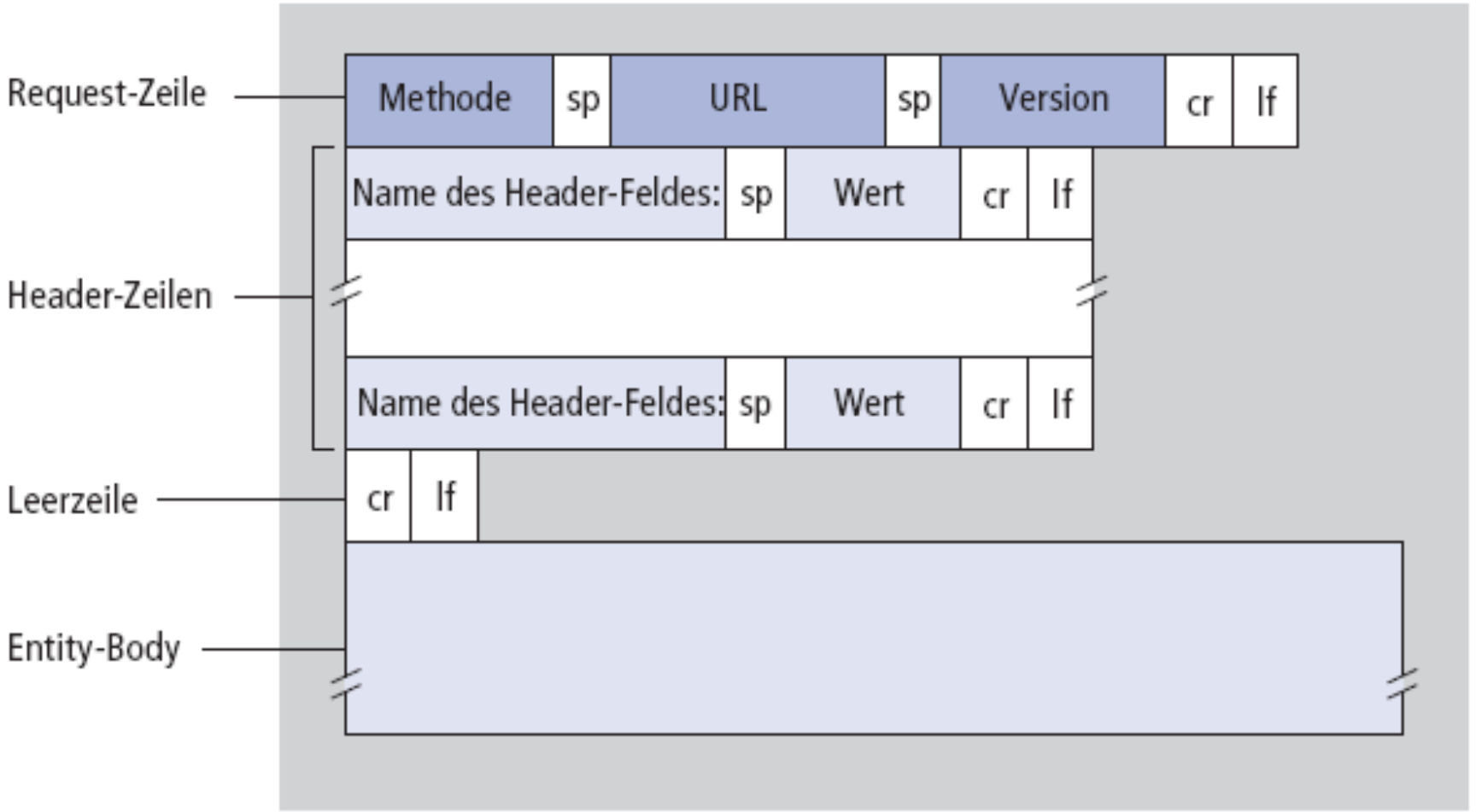
HTTP-Request-Nachricht

HTTP-Request-Nachricht:

- ASCII (vom Menschen leicht zu lesen)



HTTP-Request-Nachricht: Format



„Hochladen“ von Informationen

Post-Anweisung:

Webseiten beinhalten häufig Formulare, in denen Eingaben erfolgen sollen
Eingaben werden zum Server im Datenteil (entity body) der Post-Anweisung übertragen.

Get-Anweisung:

Eingabe wird als Bestandteil der URL übertragen:

```
www.somesite.com/animalsearch?var1=monkeys&var2=banana
```

Typen von Anweisungen

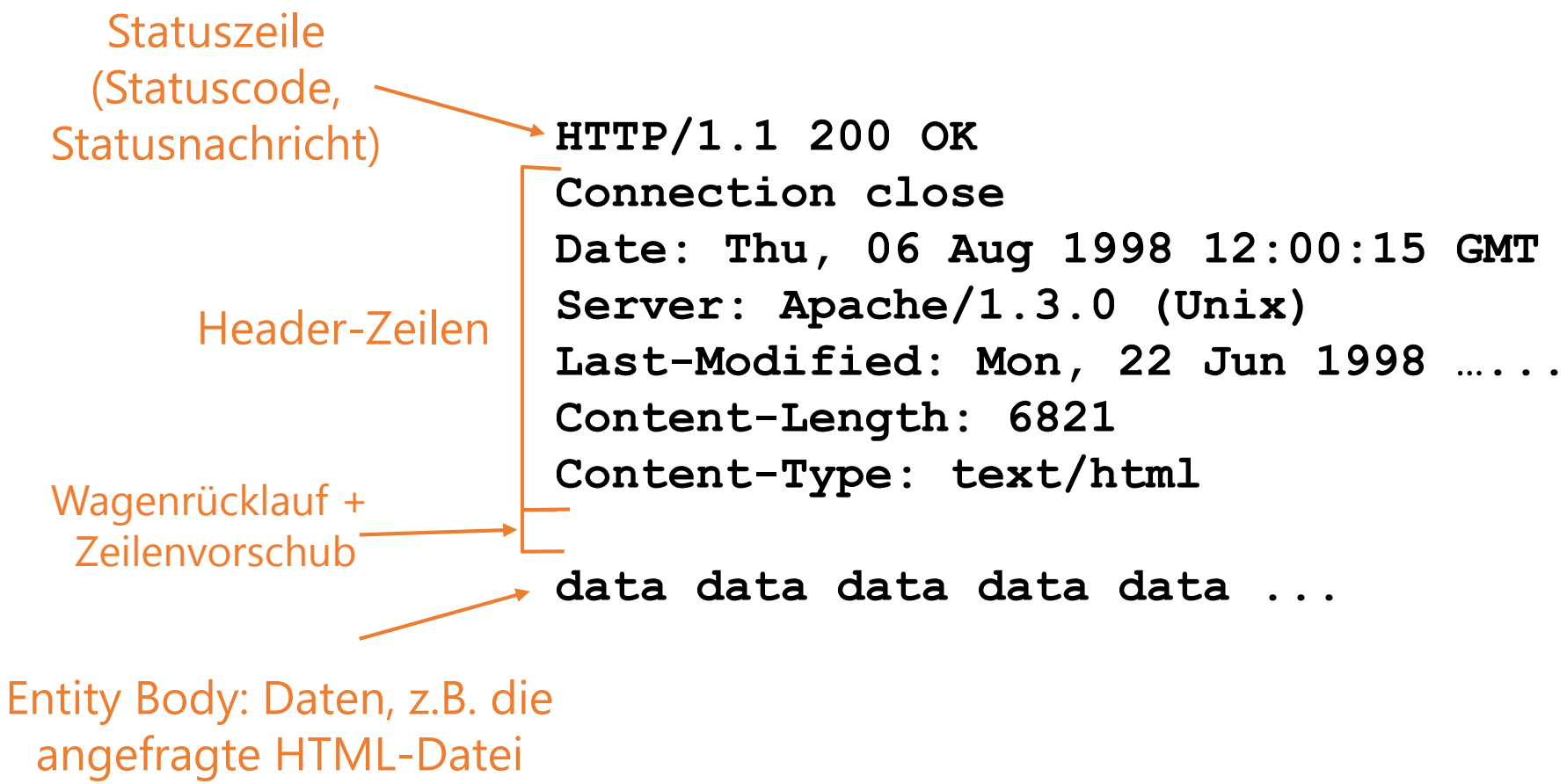
HTTP/1.0

- GET
- POST
- HEAD
    - Bittet den Server, nur die Kopfzeilen der Antwort (und nicht das Objekt) zu übertragen

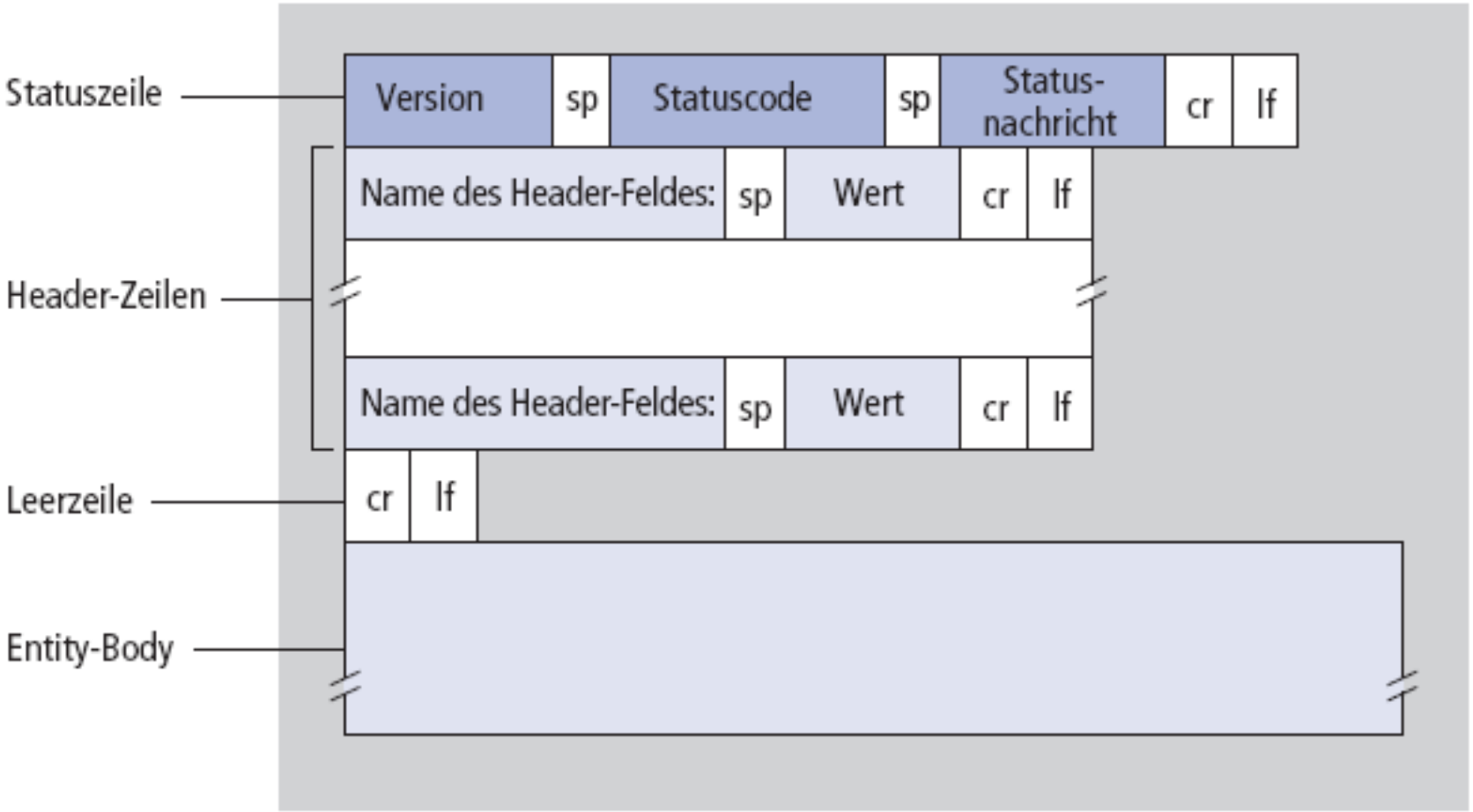
HTTP/1.1

- GET, POST, HEAD
- PUT
    - Lädt die im Datenteil enthaltene Datei an die durch eine URL bezeichnete Position hoch
- DELETE
    - Löscht die durch eine URL angegebene Datei auf dem Server

HTTP-Response-Nachricht



HTTP-Response-Nachricht: Format



Statuscodes für HTTP-Response

In der ersten Zeile der Response-Nachricht:

1xx – Informationen

2xx – Erfolgreiche Operation

3xx – Umleitung

4xx – Client-Fehler

5xx – Server-Fehler

9xx – Proprietäre Statuscodes

Einige Beispiele für Statuscodes:

200 OK
- Request war erfolgreich, gewünschtes Objekt ist in der Antwort enthalten

301 Moved Permanently
- Gewünschtes Objekt wurde verschoben, neue URL ist in der Antwort enthalten

400 Bad Request
- Request-Nachricht wurde vom Server nicht verstanden

404 Not Found
- Gewünschtes Objekt wurde nicht gefunden

505 HTTP Version Not Supported

http://de.wikipedia.org/wiki/HTTP-Statuscode

Lab: HTTP-Anfrage

1. Telnet zu einem Webserver:

```
telnet cis.poly.edu 80
```

Öffnet eine TCP-Verbindung zu Port 80 (Standard-HTTP-Server-Port) auf cis.poly.edu. Alles, was jetzt eingegeben wird, wird an Port 80 auf cis.poly.edu geschickt

2. Eingeben eines HTTP-GET-Requests:

```
GET /~ross/ HTTP/1.1
Host: cis.poly.edu
```

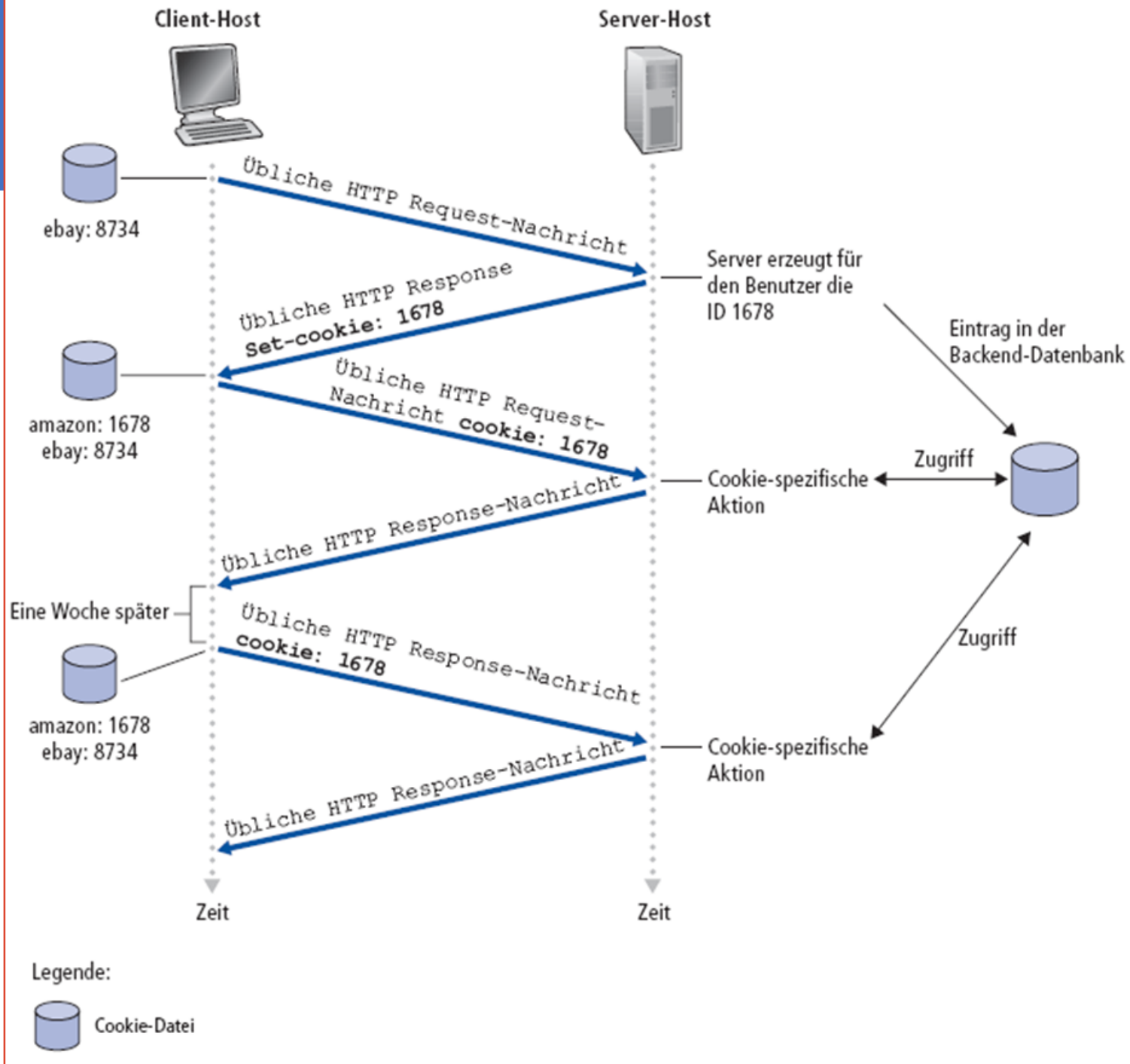
Durch diese Eingabe (am Ende zweimal Return tippen!) wird ein minimaler (aber vollständiger) GET-Request an den HTTP-Server geschickt

Testbar z.B. unter: http://telnet.browseas.com/

Zustand halten: Cookies

Realisiert durch vier wesentliche Bestandteile:

1. Cookie-Kopfzeile in der HTTP-Response-Nachricht
2. Cookie-Kopfzeile in der HTTP-Request-Nachricht
3. Cookie-Datei, die auf dem Rechner des Anwenders angelegt und vom Browser verwaltet wird
4. Backend-Datenbank auf dem Webserver



Cookies

Einsatz von Cookies:

- Autorisierung
- Einkaufswagen
- Empfehlungen
- Sitzungszustand (z.B. bei Web-E-Mail)

Privatsphäre:
Cookies ermöglichen es Websites, viel über den Anwender zu lernen:

- Formulareingaben (Name, E-Mail-Adresse)
- Besuchte Seiten

<u>Alternativen, um Zustand zu halten:</u>

- In den Endsystemen: Zustand wird im Protokoll auf dem Client oder Server gespeichert und für mehrere Transaktionen verwendet
- Cookies: HTTP-Nachrichten beinhalten den Zustand

Proxyserver

Gateway
- Integration von Antivirus-Lösungen als Content-Scanner

Cache
- Auslieferung von oft genutzten Inhalten ohne Zugriff auf den eigentlichen Quell-Webserver