

3.1 Netzanwendungen

3.2 Web und HTTP (HyperText Transfer Protocol)

3.2.1 HTTP Nachrichten

3.2.2 Web-Seiten Übertragung: Entwicklung von HTTP/1.0 bis HTTP/2.0

3.2.3 Proxies und Caches

3.3 DNS (Domain Name System)

3.4 Weitere Anwendungsprotokolle: Mail und FTP

Was spezifizieren Protokolle auf Anwendungsschicht?

- 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 z.B.
 - Cookie, Browserversion bei http
 - Play/Pause bei Videostreaming (RTSP)
 - (Licht) Ein-/Ausschalten bei ZigBee Light Link (IoT)
 - Übergang Protokoll der Anwendungsschicht und Anwendung teilweise fließend
- Regeln für das Senden von und Antworten auf Nachrichten

Öffentlich verfügbare (standardisierte) Protokolle:

- Definiert in RFCs
- Ermöglichen Interoperabilität
- z.B. HTTP, SMTP

Proprietäre Protokolle:

- z.B. Skype

Anmerkung:

Wir unterscheiden generell zwischen „standardisiert“ und „proprietär“. Standardisiert bedeutet durch einen Standard festgelegt, Produkte müssen zertifiziert werden. Proprietär heißt eigene Lösung eines Hardware- oder Softwareherstellers.

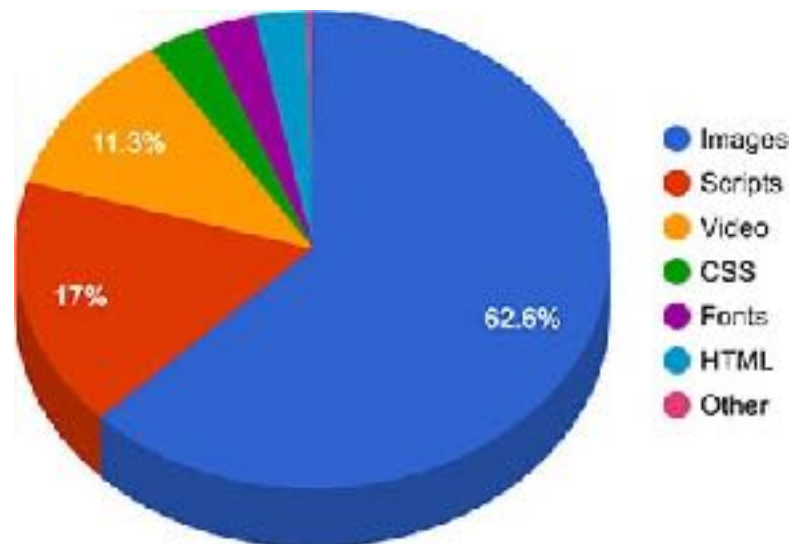
Was ist eigentlich eine Web-Seite?

- eine Web-Seite besteht aus Objekten
- Objekte sind HTML-Code, Grafiken,
- Objekte können sein:
 - HTML-Dateien, JPEG-Bilder, Java-Applets, Audiodateien, ...
- Webseite besteht aus
 - **Basis-HTML-Datei** (main object)
 - referenzierten Objekten (inline objects, assets)
- Objekte sind durch **URL** (Uniform Resource Locator) adressierbar
- Beispiel für eine URL:

www.kicker.de/news/fussball/chleague/startseite.html

Hostname

Pfad



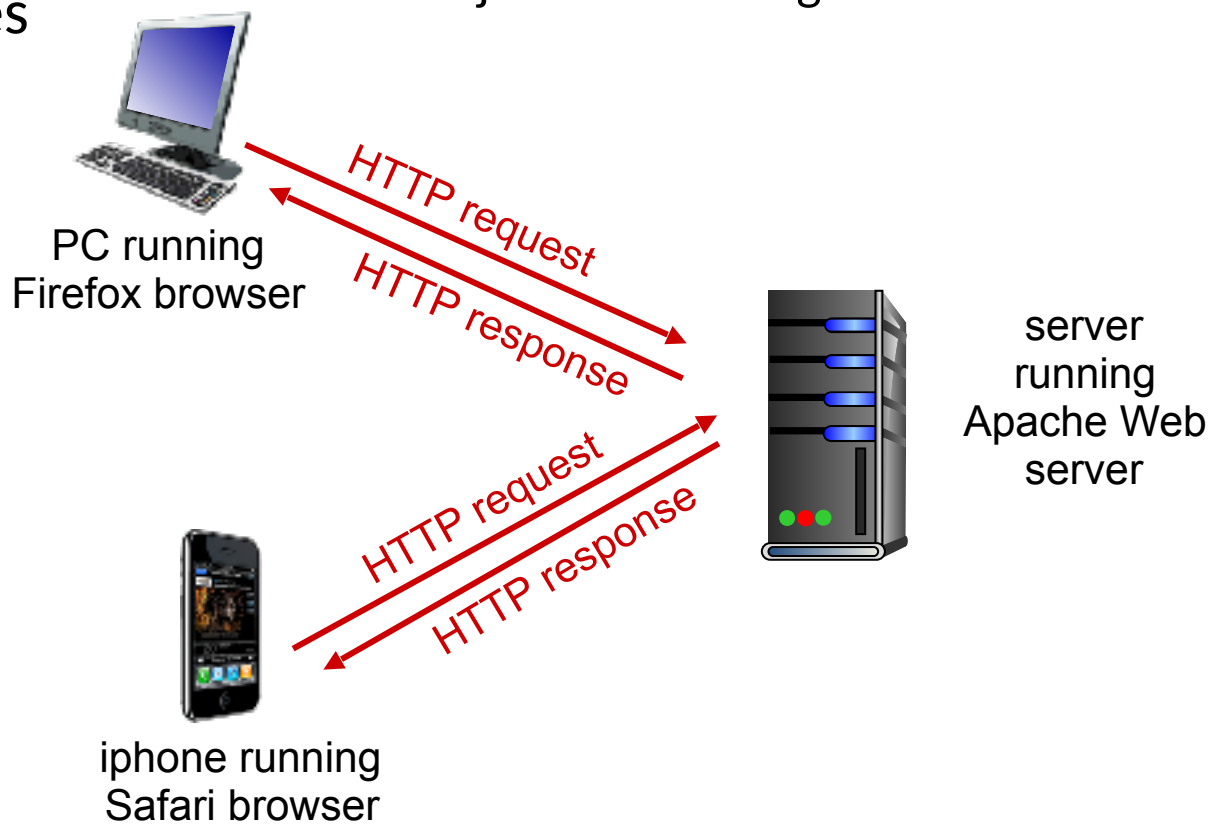
Quelle: developers.google.com

HTTP: HyperText Transfer Protocol

- Ursprünglich entwickelt als Anwendungsprotokoll des Word Wide Web
 - Transfer von Web-Seiten
- Inzwischen universelles Anwendungsprotokoll
 - REST Schnittstelle
 - Streaming
 - Datentransfer
 - IoT
 - etc.

Client/Server-Modell

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



- Standardisiert von der IETF HTTP Working-Group
 - 1996: RFC 1945 HTTP 1.0
 - 1999: RFC 2616 HTTP 1.1
 - 2014: RFC 7230-7235 verschiedene Ergänzungen zu HTTP/1.1: Message Syntax and Routing, Semantics and Content, Conditional Requests, Range Requests, Caching, Authentication
 - Standard wird nachträglich an Praxis im Netz angepasst
 - 2015: RFC 7540 – HTTP/2.0
 - 2021: Internet Draft – HTTP/3.0
- HTTP/2
 - von Browsern unterstützt, langsame Einführung auf Serverseite
 - höhere Komplexität als HTTP/1.1 könnte Einsatzgebiete (z.B. IoT) einschränken
 - Übertragung von Objekten in Streams, Header-Compression, Server-Push
- HTTP/3
 - Internet Draft, nicht finalisiert (Apr 2022)
 - von 72% der verwendeten Browsern und 25% der Top 10 Mio Web-Sites unterstützt
 - Features: QUIC (Übertragung über UDP), TLS 1.3 verpflichtend
- CoAP: Constrained Access Protocol
 - ressourcen-schonende Implementierung der REST Schnittstelle

HTTP/HTTPS verwenden TCP:

- Client **initiiert** eine TCP-Verbindung (**Socket**) zu **Port 80/443** des Servers auf
- Server akzeptiert die TCP-Verbindung des Clients auf **Port 80/443**
- HTTP/HTTPS-Nachrichten (Protokollnachrichten der Anwendungsschicht) werden zwischen Browser (HTTP-Client) und Webserver (HTTP-Server) ausgetauscht
- Die TCP-Verbindung wird geschlossen

Achtung: HTTP/3 über UDP Sockets

- HTTP/1 ist „**zustandslos**“ (stateless)
- ab HTTP/2 gibt es HTTP Session, d.h. HTTP wird „**zustandsbehaftet**“ (stateful)
- REST ist davon nicht betroffen und bleibt zustandslos

Bemerkung am Rande

Protokolle, die einen Zustand verwalten, sind komplex!

- Der Zustand muss gespeichert und verwaltet werden
 - Web-Server: Millionen Zustände
- Nach einem Absturz oft inkonsistenter Zustand, dann Synchronisierung erforderlich

3.1 Netzanwendungen

3.2 Web und HTTP (HyperText Transfer Protocol)

3.2.1 HTTP Nachrichten

3.2.1.1 Request und Response

3.2.1.2 HTTP/2 Header Compression

3.2.1.3 Cookies und Tracking

3.2.2 Web-Seiten Übertragung: Entwicklung von HTTP/1.0 bis HTTP/2.0

3.2.3 Proxies und Caches

3.3 DNS (Domain Name System)

3.4 Weitere Anwendungsprotokolle: Mail und FTP

- Arten von Nachrichten: Request, Response
- Request-Nachrichten (im RFC: Methoden, methods):
 - **Mandatory** (verpflichtend, Server muss diese unterstützen)
 - GET: Anfrage einer Ressource vom Server
 - HEAD: wie GET, zurückgeliefert wird aber nur Status und Header
 - **Optional** (Server kann diese unterstützen)
 - POST: Ressource-spezifische Verarbeitung der Payload
 - PUT: Ressource auf Server durch Payload ersetzen
 - weitere: DELETE, CONNECT, OPTIONS, TRACE
- **Ressource**: identifiziert durch URI (Uniform Resource Identifier)
http-URI = "http:" "://" authority path-abempty ["?" query] ["#" fragment]
 - authority: Hostname, optional Port
 - path-abempty: Pfad (kann leer sein oder fehlen)
 - query: Anfrage
 - fragment: Teil einer Ressource, verschiedene Bedeutungen abhängig von der Art der Ressource, z.B. Teil eines Videos, einer Tabelle etc.

Request Header

- Header-Zeilen: geben dem Server weitere Informationen
 - über den Sender der Nachricht
 - das erwartete Format der Antwort
 - Authentisierung
 - etc.
- Definitionen unter <http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html> oder in den RFCs

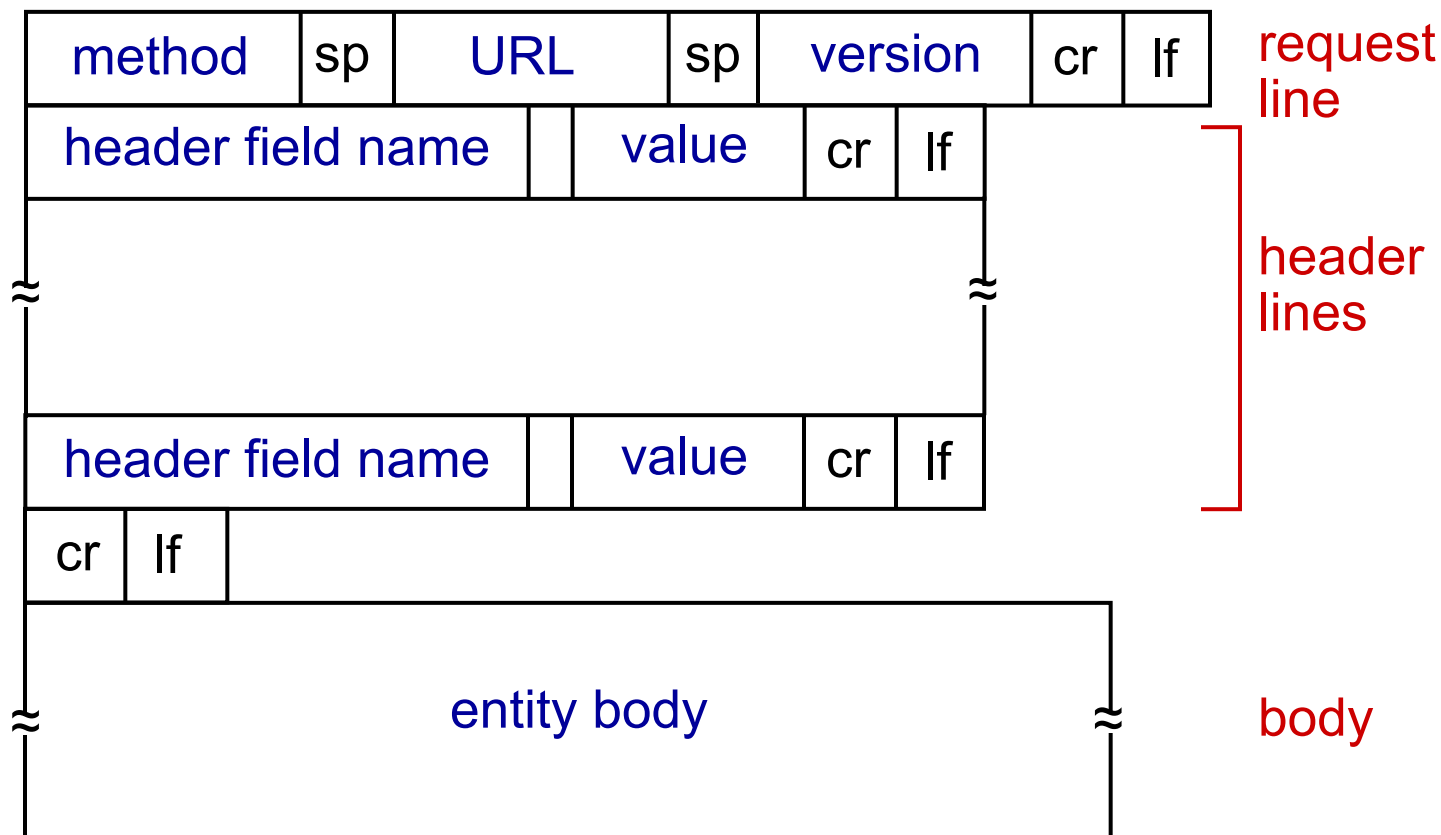
The diagram illustrates the structure of an HTTP request. It shows a request line and several header lines. Annotations with arrows point to specific parts of the request:

- Request-Zeile (GET, POST, HEAD commands)**: Points to the first line of the request.
- URI**: Points to the path part of the request line.
- Version**: Points to the HTTP version part of the request line.
- Header-Zeilen**: A bracket points to the entire block of header lines.
- Ende der Nachricht: Leerzeile (CR+LF)**: Points to the final empty line of the request.
- Qualifier**: Points to the 'q=' parameter in the 'Accept-Language' header.

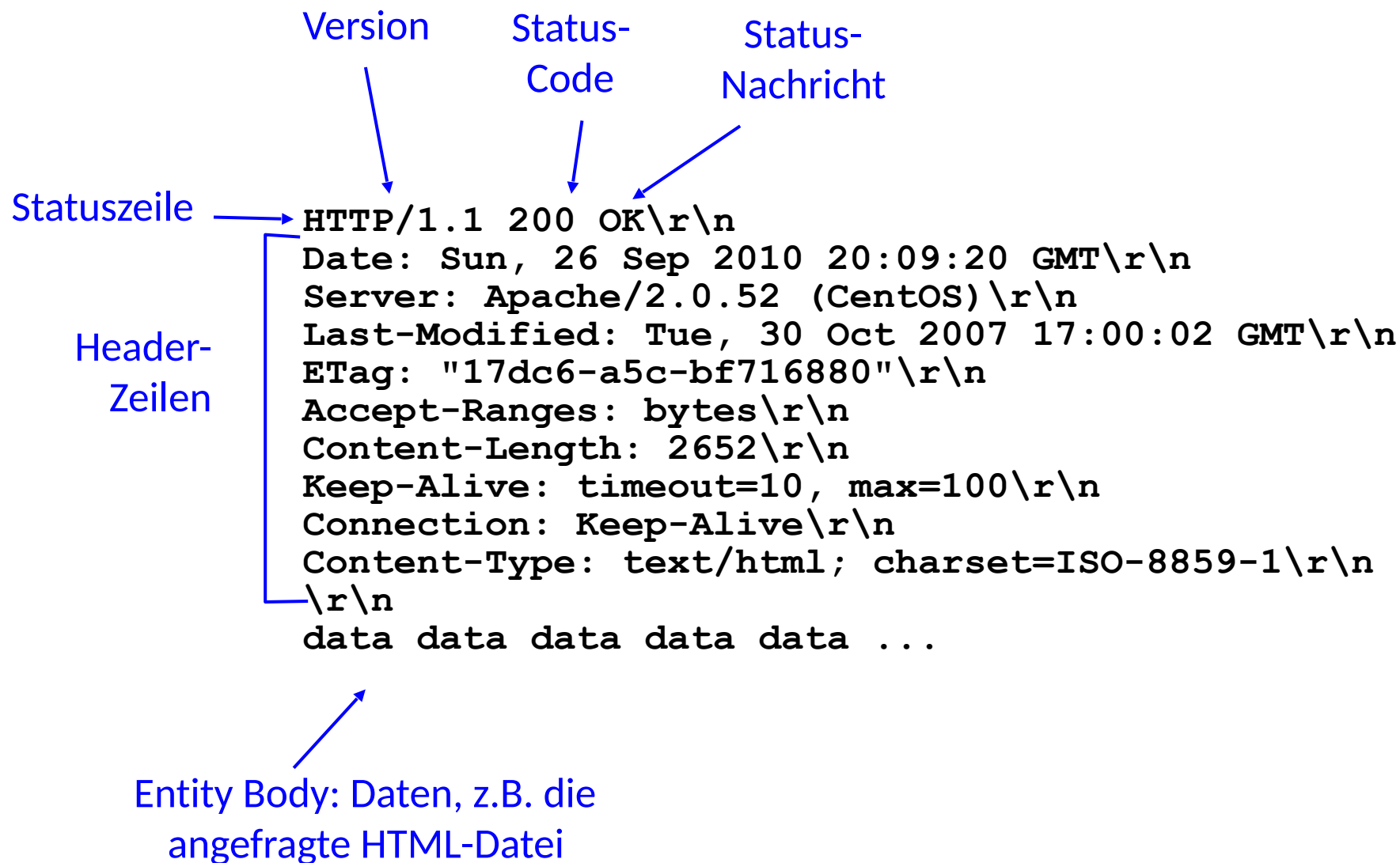
```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7\r\n
Keep-Alive: 115\r\n
Connection: keep-alive\r\n
\r\n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

Request Nachricht

Im Gegensatz zum TCP oder IP Header haben die Felder des HTTP Headers keine feste Länge sondern werden durch Leerzeichen oder Zeilenenumbrüche getrennt. Der HTTP Header (bis HTTP/2.0) wird als Text übertragen.



HTTP Response



Status-Codes erlauben es dem Server

- dem Client den Status der Anfrage mitzuteilen
- dem Client Informationen zu einer Ressource zu übermitteln
- dem Client Anweisungen zu geben, wie eine Ressource zu laden ist

Beispiele von Status-Code (RFC 7231):

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

3.1 Netzanwendungen

3.2 Web und HTTP (HyperText Transfer Protocol)

3.2.1 HTTP Nachrichten

3.2.1.1 Request und Response

3.2.1.2 HTTP/2 Header Compression

3.2.1.3 Cookies und Tracking

3.2.2 Web-Seiten Übertragung: Entwicklung von HTTP/1.0 bis HTTP/2.0

3.2.3 Proxies und Caches

3.3 DNS (Domain Name System)

3.4 Weitere Anwendungsprotokolle: Mail und FTP

What HTTP/2.0 will* do ~~for~~^{to} you

Mark Nottingham
<http://www.mnot.net/>
or, @mnot



Folien aus dem Vortrag "What HTTP/2.0 will do ~~for~~^{to} you"
von Mark Nottingham, httpbis working group chair

HTTP Header

```
GET / HTTP/1.1
Host: www.etsy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14
(KHTML, like Gecko) Version/6.0.1 Safari/536.26.14
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
DNT: 1
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: usaid=usaid%3D0Vdhk5WGsexG-_Y7ZBeQFa3eq7yMQ%26_now%3D1325204464%26_slrt
%3Ds_LCLVpU%26_kid%3D1%26_ver%3D1%26_mac
%3D1VnLM3hMdb3Cs3hqMVuk_dQEixsqQzU1NYCs9H_Kj8c.;
user_prefs=1&2596706699&q0tPzMlJLaoEAA==
Connection: keep-alive
```

525 bytes

```
GET /assets/dist/js/etsy.recent-searches.20121001205006.js HTTP/1.1
Host: www.etsy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14
(KHTML, like Gecko) Version/6.0.1 Safari/536.26.14
Accept: */*
DNT: 1
Referer: http://www.etsy.com/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: autosuggest_split=1;
etala=111461200.1476767743.1349274889.1349274889.1349274889.1.0;
etalb=111461200.1.10.1349274889; last_browse_page=%2F; uaid=uaid%3DVdhk5W6sexG-
_Y7ZBeQFa3cq7yMQ%26_now%3D1325204464%26_sl%3Ds_LCLVpU%26_kid%3D1%26_ver%3D1%26_mac
%3DLVnLm3hMdb3Cs3hqMVuk_dQEixsqQzUlnYCs9H_Kj8c.;
user_prefs=1&2596706699&q0tPzMlJLaoEAA==
Connection: keep-alive
```

226 new bytes; 690 total


```
GET /assets/dist/js/jquery.appear.20121001205006.js HTTP/1.1
Host: www.etsy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14
(KHTML, like Gecko) Version/6.0.1 Safari/536.26.14
Accept: */*
DNT: 1
Referer: http://www.etsy.com/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: autosuggest_split=1;
etala=111461200.1476767743.1349274889.1349274889.1349274889.1.0;
etalb=111461200.1.10.1349274889; last_browse_page=%2F; uaid=uaid%3Dvdhk5W6sexG-
_Y7ZBeQFa3cq7yMQ%26_now%3D1325204464%26_sl%3Ds_LCLVpU%26_kid%3D1%26_ver%3D1%26_mac
%3DLVnlM3hMdb3Cs3hqMVuk_dQEixsqQzULNYCs9H_Kj8c.;
user_prefs=1&2596706699&q&tPzMLJLaoEAA==
Connection: keep-alive
```

14 new bytes; 683 total

```
GET /assets/dist/js/bootstrap/username-suggester.20121001205006.js HTTP/1.1
Host: www.etsy.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_8_2) AppleWebKit/536.26.14
(KHTML, like Gecko) Version/6.0.1 Safari/536.26.14
Accept: */*
DNT: 1
Referer: http://www.etsy.com/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Cookie: autosuggest_split=1;
etala=111461200.1476767743.1349274889.1349274889.1349274889.1.0;
etalb=111461200.1.10.1349274889; last_browse_page=%2F; uaid=uaid%3DVdhk5W6scxG-
_YTZBeQFa3cq7yMQ%26_now%3D1325204464%26_sl%3Ds_LCLVpU%26_kid%3D1%26_ver%3D1%26_mac
%3DlvnlM3hMdb3Cs3hqMVuk_dQEixsqZuLNYCs9H_Kj8c.;
user_prefs=1&2596706699&q0tPzMJLaoEAA==
Connection: keep-alive
```

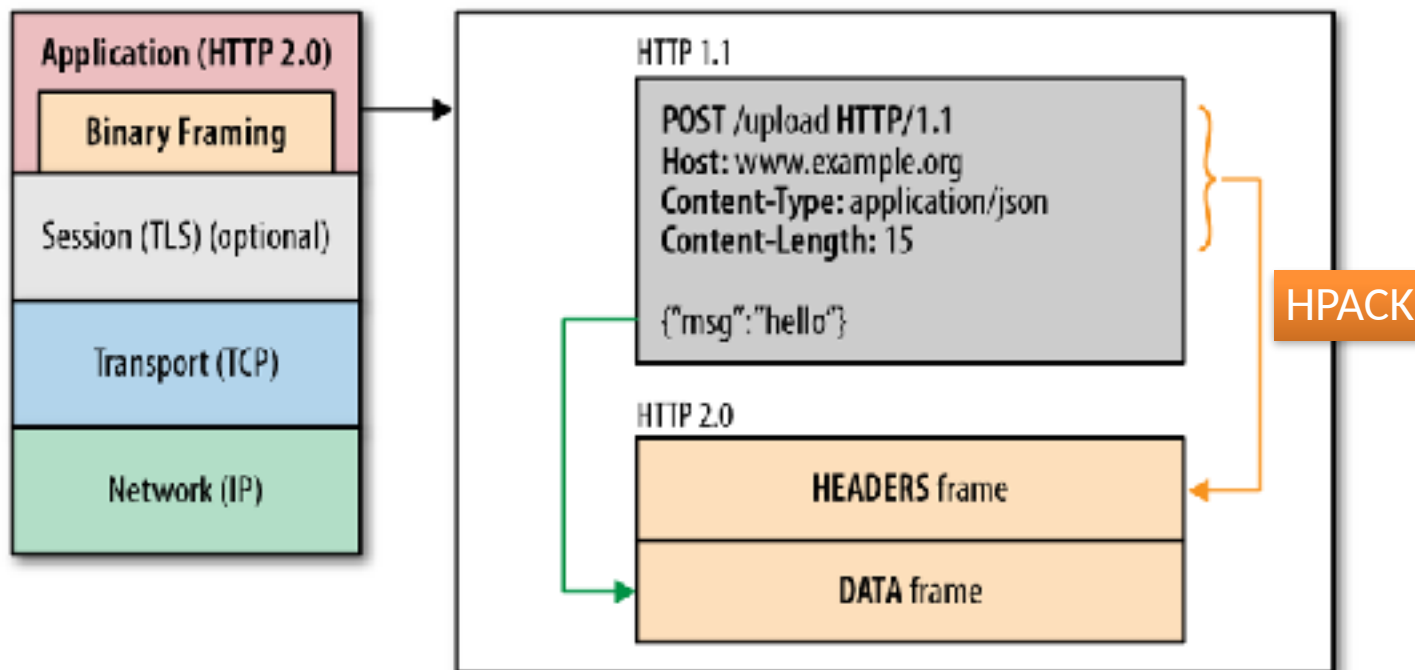
28 new bytes; 698 total

- **Four requests**
- **2,596 bytes total**
- **Minimum three packets in most places**
 - One for the HTML, two+ for assets
- **1,797 redundant bytes**

**HTTP headers on a
connection are
highly similar**

HTTP/2.0 Features – Header Compression

In HTTP/2.0 wird das HPACK Kompressionsverfahren spezifiziert, um Header zu übertragen. HPACK beruht auf statischen Tabellen mit Codes für häufig verwendete Header und zusätzlich dynamische Tabellen für Header die speziell in einer Verbindung vorkommen. HPACK verwendet „Delta-Coding“ d.h. nur der Unterschied zum letzten Header wird gesendet. Dynamische Tabellen und Delta-Coding machen HTTP/2.0 stateful.



3.1 Netzanwendungen

3.2 Web und HTTP (HyperText Transfer Protocol)

3.2.1 HTTP Nachrichten

3.2.1.1 Request und Response

3.2.1.2 HTTP/2 Header Compression

3.2.1.3 Cookies und Tracking

3.2.2 Web-Seiten Übertragung: Entwicklung von HTTP/1.0 bis HTTP/2.0

3.2.3 Proxies und Caches

3.3 DNS (Domain Name System)

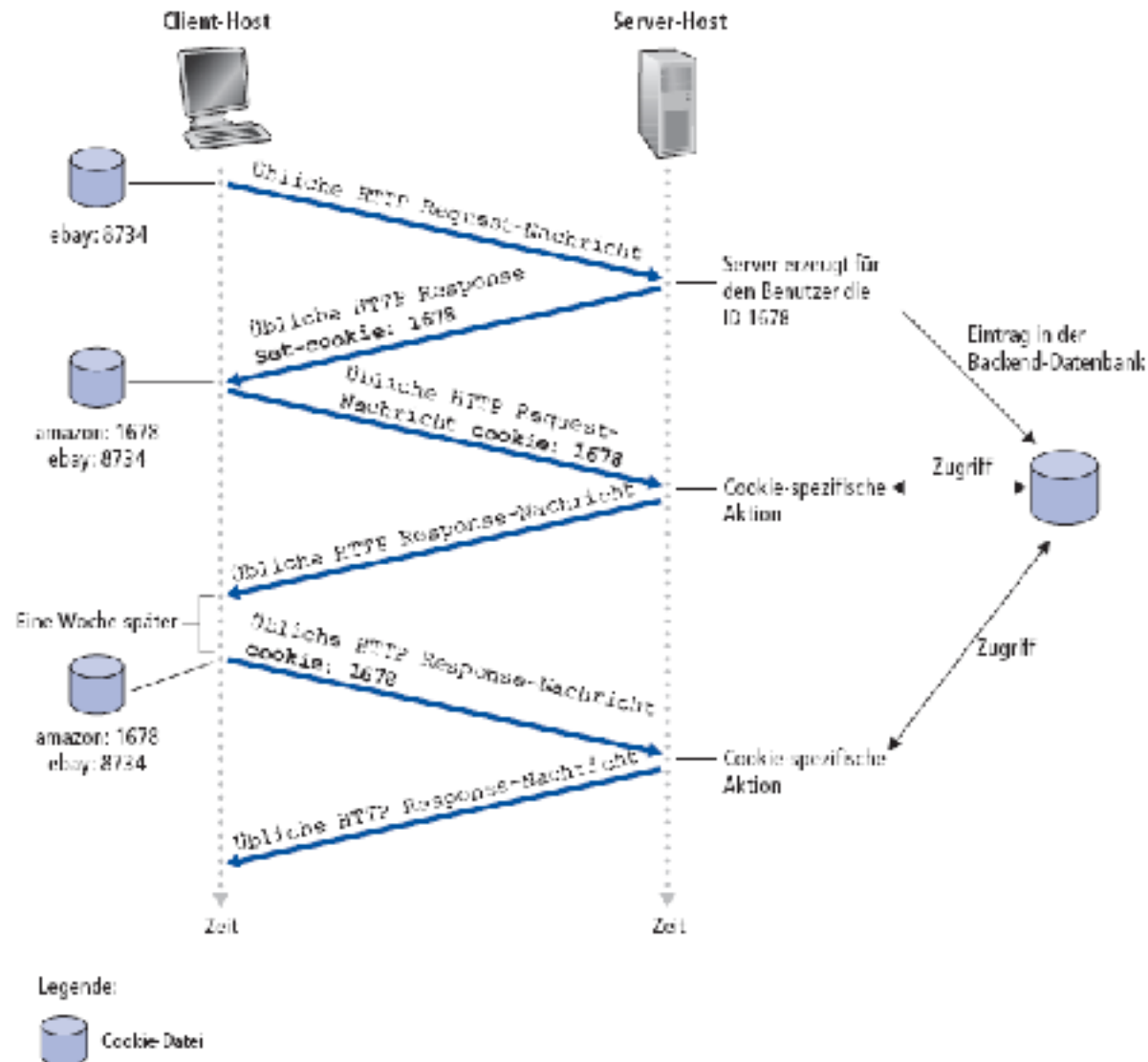
3.4 Weitere Anwendungsprotokolle: Mail und FTP

Cookies:

Ziel: Wiedererkennung eines Rechners bei wiederholter Verbindung zu einem Server

Vier 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



Einsatz von Cookies:

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

Am Rande

Cookies und Privatsphäre:

- Cookies ermöglichen es Websites, viel über den Anwender zu lernen:
 - Formulareingaben (Name, E-Mail-Adresse)
 - Besuchte Seiten

Wie der „Zustand“ gehalten werden kann:

- Protokollendpunkte: Zustand am Client oder Server speichern und für mehrere Transaktionen verwendet
- Cookies: HTTP-Nachrichten beinhalten den Zustand

- Browser Fingerprint:
 - In den Headerlines gibt eine Browser viele Informationen an den Server preis
 - über JavaScript lernt der Server noch weitere Informationen
 - dadurch ist ein Browser meist eindeutig identifizierbar
 - Test: <https://amiunique.org/fp>
- Canvas Fingerprinting:
 - Identifikation des Browsers über die Darstellung eines nicht-sichtbaren Canvas-Elements
 - <https://browserleaks.com/canvas#how-does-it-work>
- Mausbewegungen und Tippgeschwindigkeit
 - auch die Art und Weise, wie wir tippen und die Maus bewegen kann aufgezeichnet werden
 - Banken sollen diese Technik nutzen, um Kunden wiedererkennen zu können

3.1 Netzanwendungen

3.2 Web und HTTP (HyperText Transfer Protocol)

3.2.1 HTTP Nachrichten

3.2.2 Web-Seiten Übertragung: Entwicklung von HTTP/1.0 bis HTTP/2.0

3.2.3 Proxies und Caches

3.3 DNS (Domain Name System)

3.4 Weitere Anwendungsprotokolle: Mail und FTP

Ein Web-Page besteht aus VIELEN Objekten. Einer der Hauptunterschiede der verschiedenen HTTP Versionen liegt im Download-Verfahren einer Web-Seite mit vielen Objekten in einer oder mehreren TCP Verbindungen.

Non-persistent HTTP (HTTP/1.0)

- eigene TCP Verbindung für jedes Objekt

Persistent HTTP (HTTP/1.1)

- mehrere Objekte nacheinander in einer TCP Verbindung
- Keep-alive Feature: TCP Verbindung zum Server bleibt offen
 - ermöglicht das Laden mehrerer Seiten vom gleichen Server in einer Verbindung

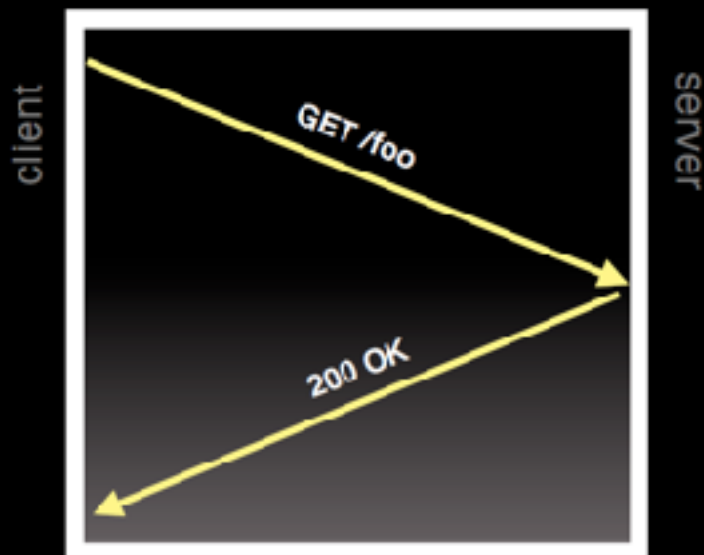
Pipelining (HTTP/1.1)

- Parallelisierung von Requests und Responses
 - vorher: nächster HTTP Request erst nach Erhalt der ausstehenden HTTP Response
 - jetzt: mehrere HTTP Requests, dann mehrere HTTP Responses in gleicher Reihenfolge

Parallele TCP Verbindungen:

- Inline-Objekte werden in mehreren TCP-Verbindungen parallel übertragen

connection setup

**vanilla
HTTP/1.0**One request
per TCP connection

connection close

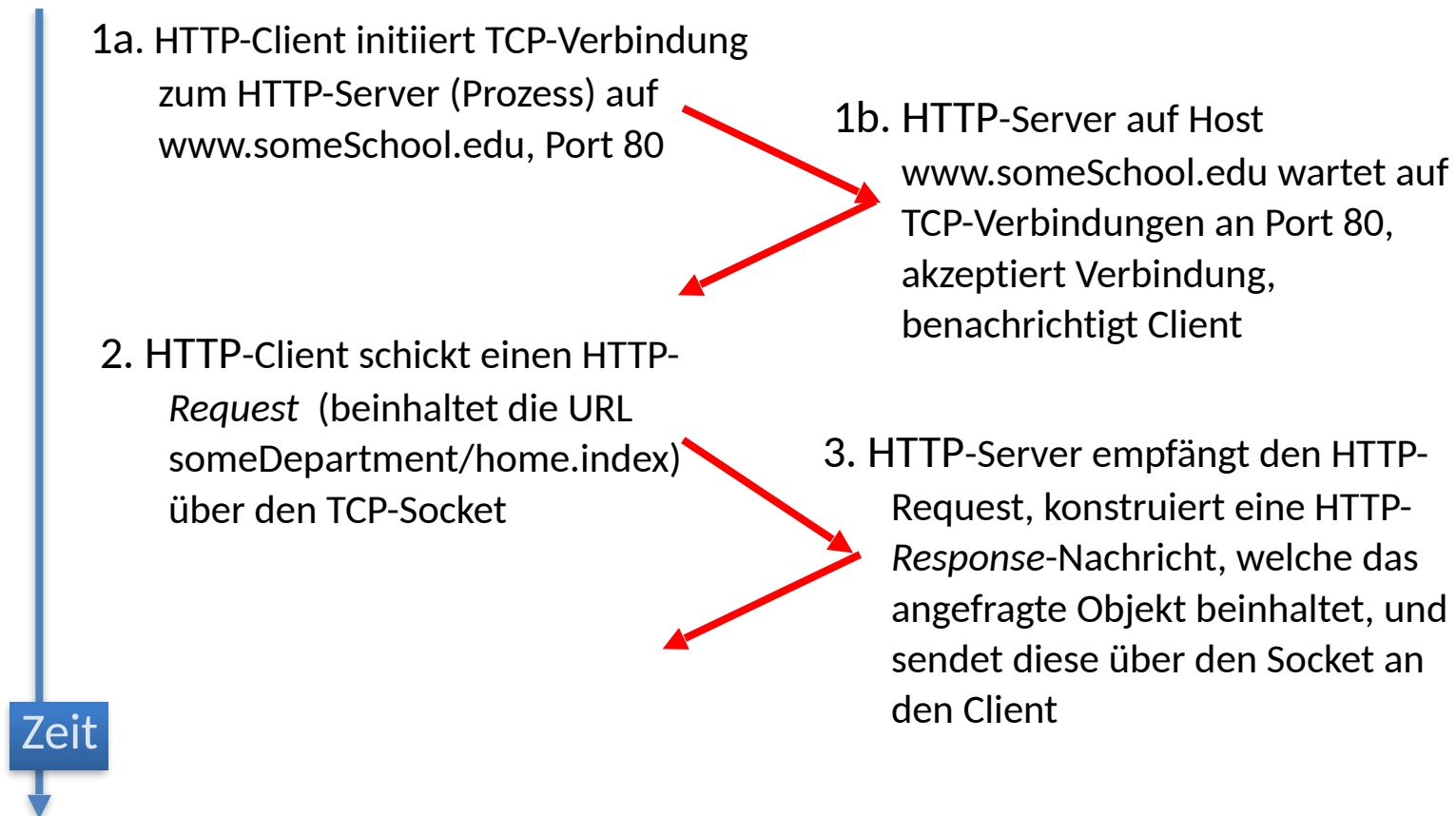
OMG ITS SO SLOW

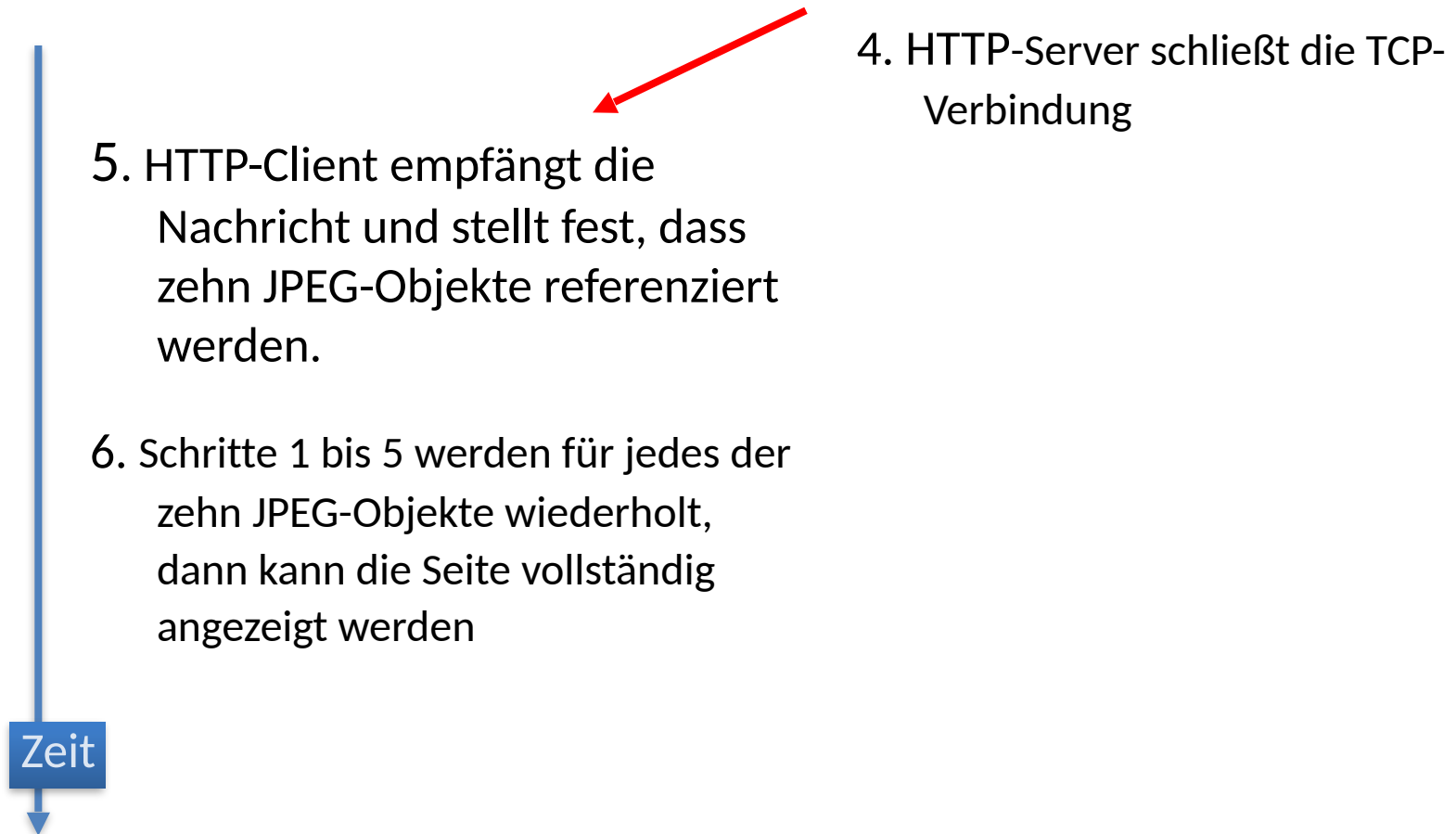
Folien aus dem Vortrag "What HTTP/2.0 will do ^{to} for you"
von Mark Nottingham, httpbis working group chair

Non-persistent HTTP

Beispiel:

- wir „laden“ `www.someSchool.edu/someDepartment/home.index`
- Struktur: HTML-Code mit 10 Bildern



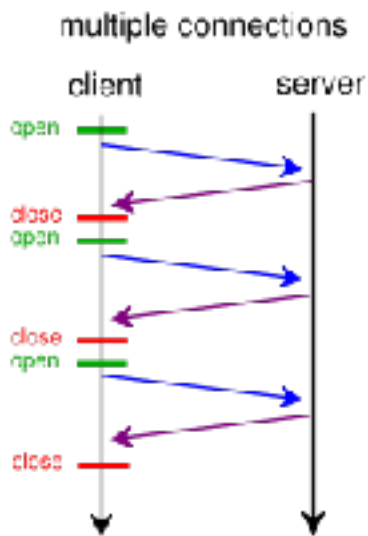
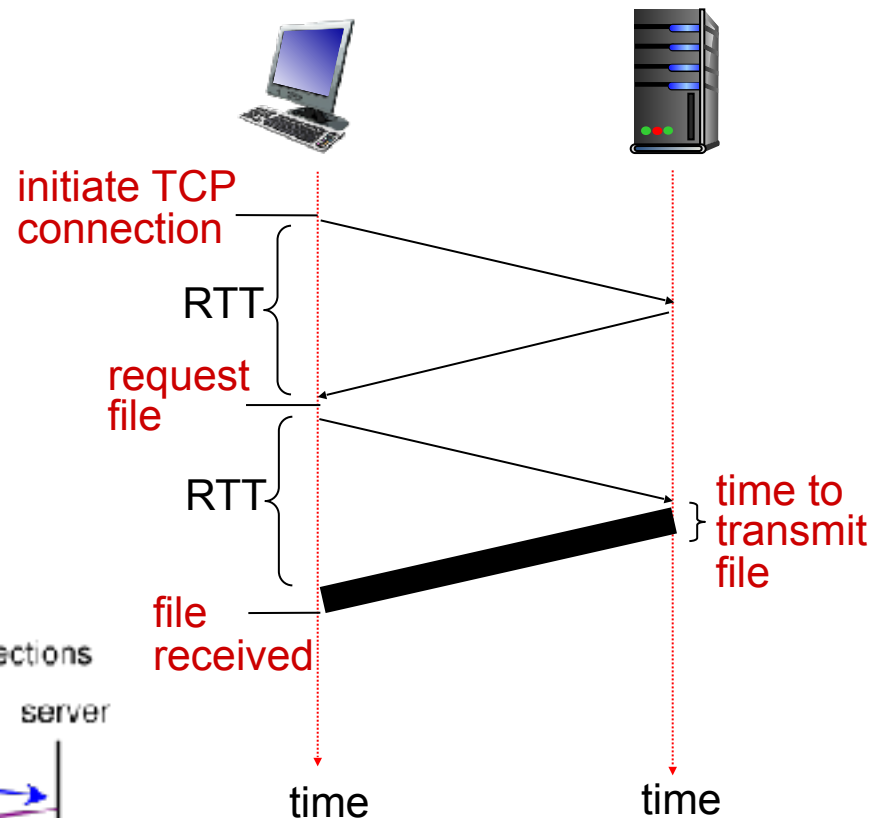


Non-persistent HTTP: PLT (Page Load Time)

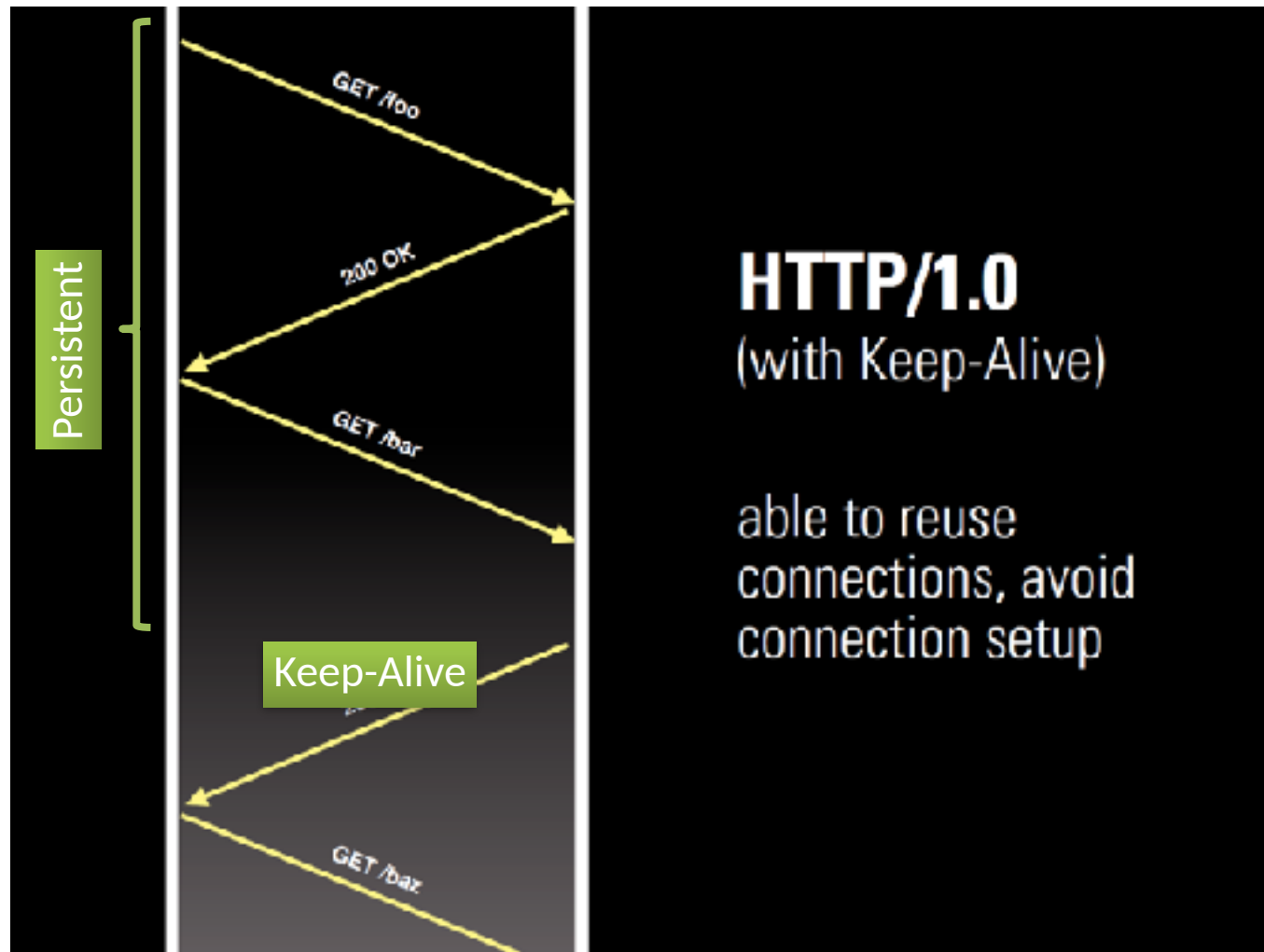
Round Trip Time (RTT): Zeit vom Sender zum Empfänger und zurück

Analyse **Page Load Time (PLT):**

- pro Objekt:
 - TCP-Verbindungsaufbau: eine RTT
 - HTTP-Request bis Empfang des erstes Bytes der HTTP-Response: eine RTT
 - Zeit für das Übertragen der Daten auf der Leitung
 - Zeit pro Objekt: 2 RTT + Übertragungsdauer der Daten des Objekts
- PLT: 22 RTT + Übertragungsdauern



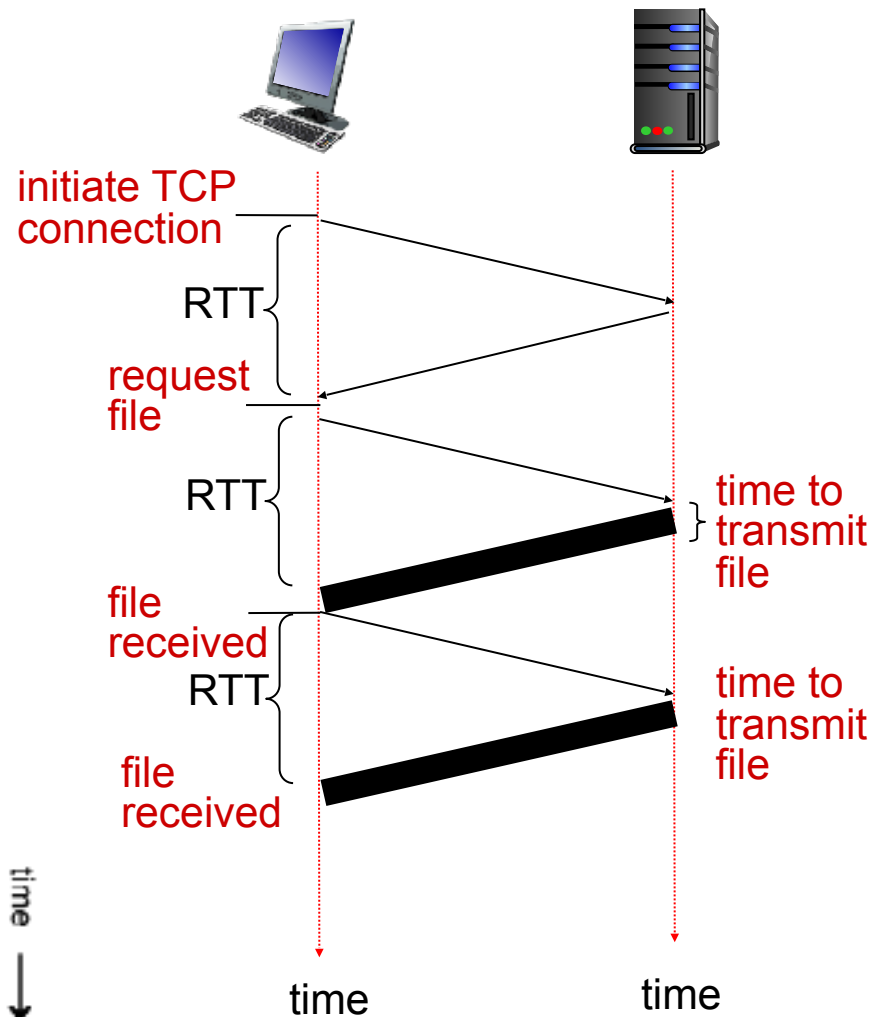
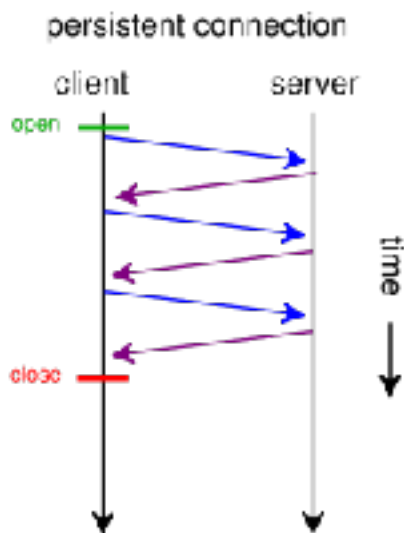
Persistent + Keep-Alive



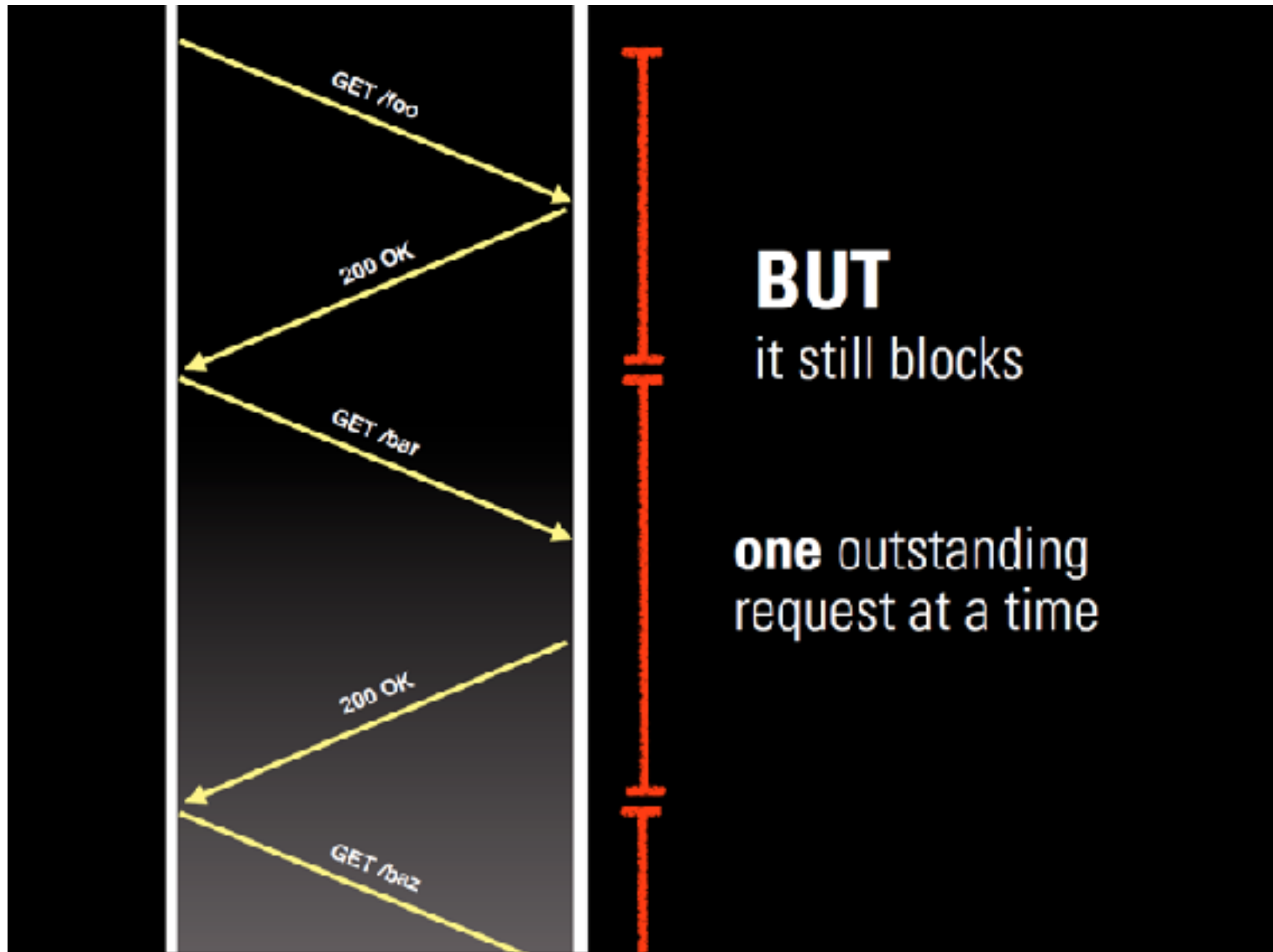
Persistent HTTP: PLT (Page Load Time)

Analyse **Page Load Time** (PLT):

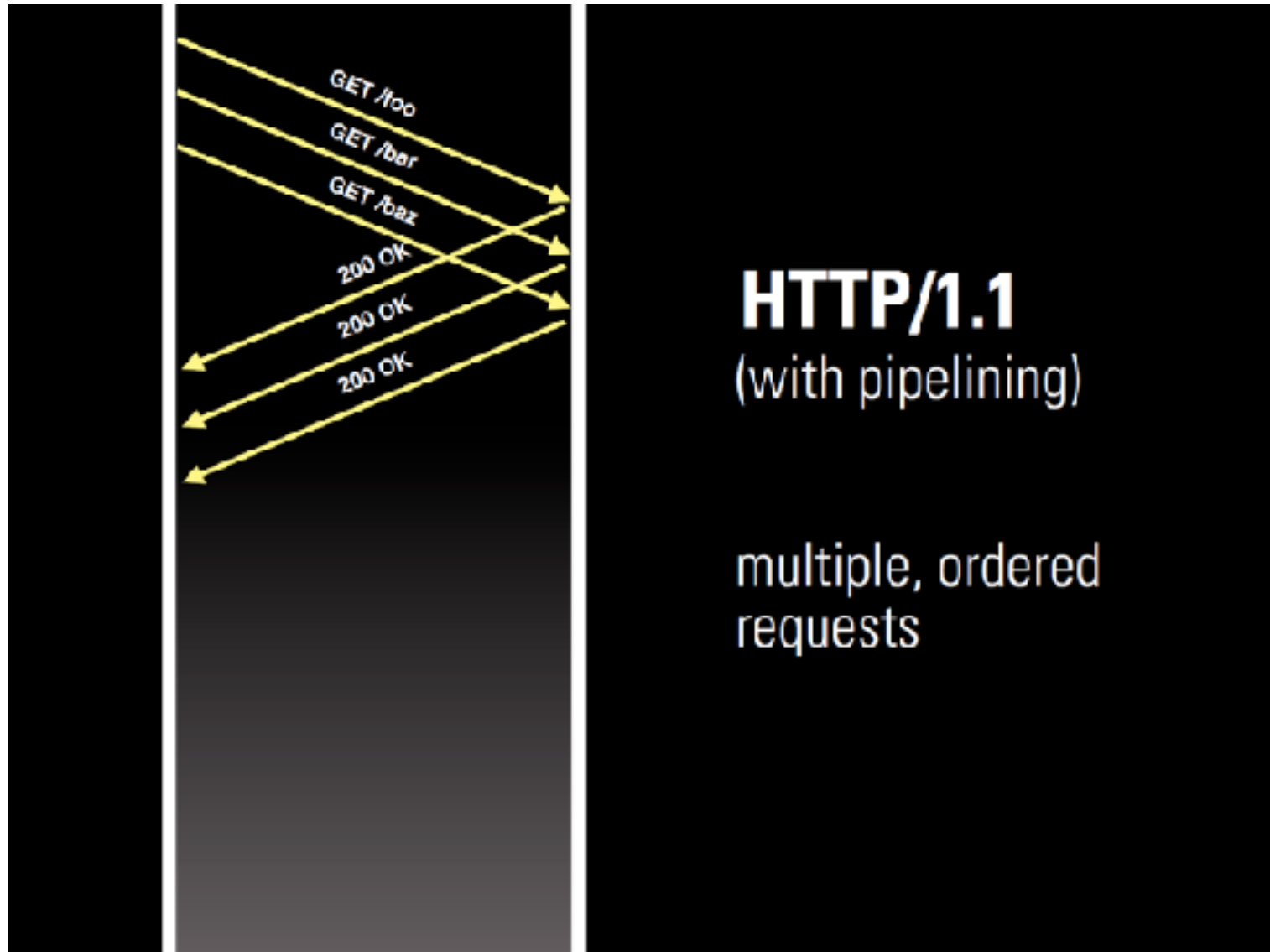
- TCP-Verbindungsaufbau: eine RTT
- pro Objekt
 - HTTP-Request bis Empfang des erstes Bytes der HTTP-Response: eine RTT
 - Zeit für das Übertragen der Daten auf der Leitung
 - pro Objekt: 1 RTT + Übertragungsdauer
- PLT: 12 RTT + Übertragungsdauer



Persistent

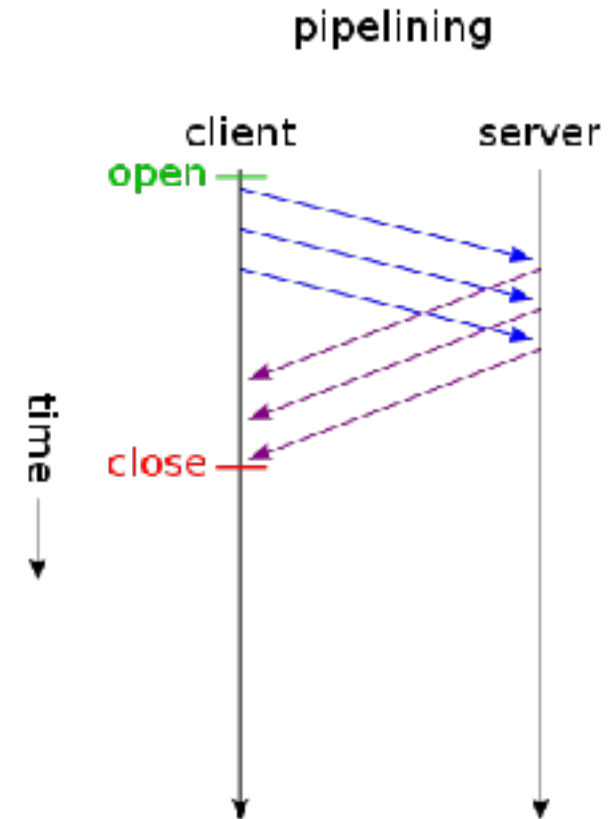


Pipelining

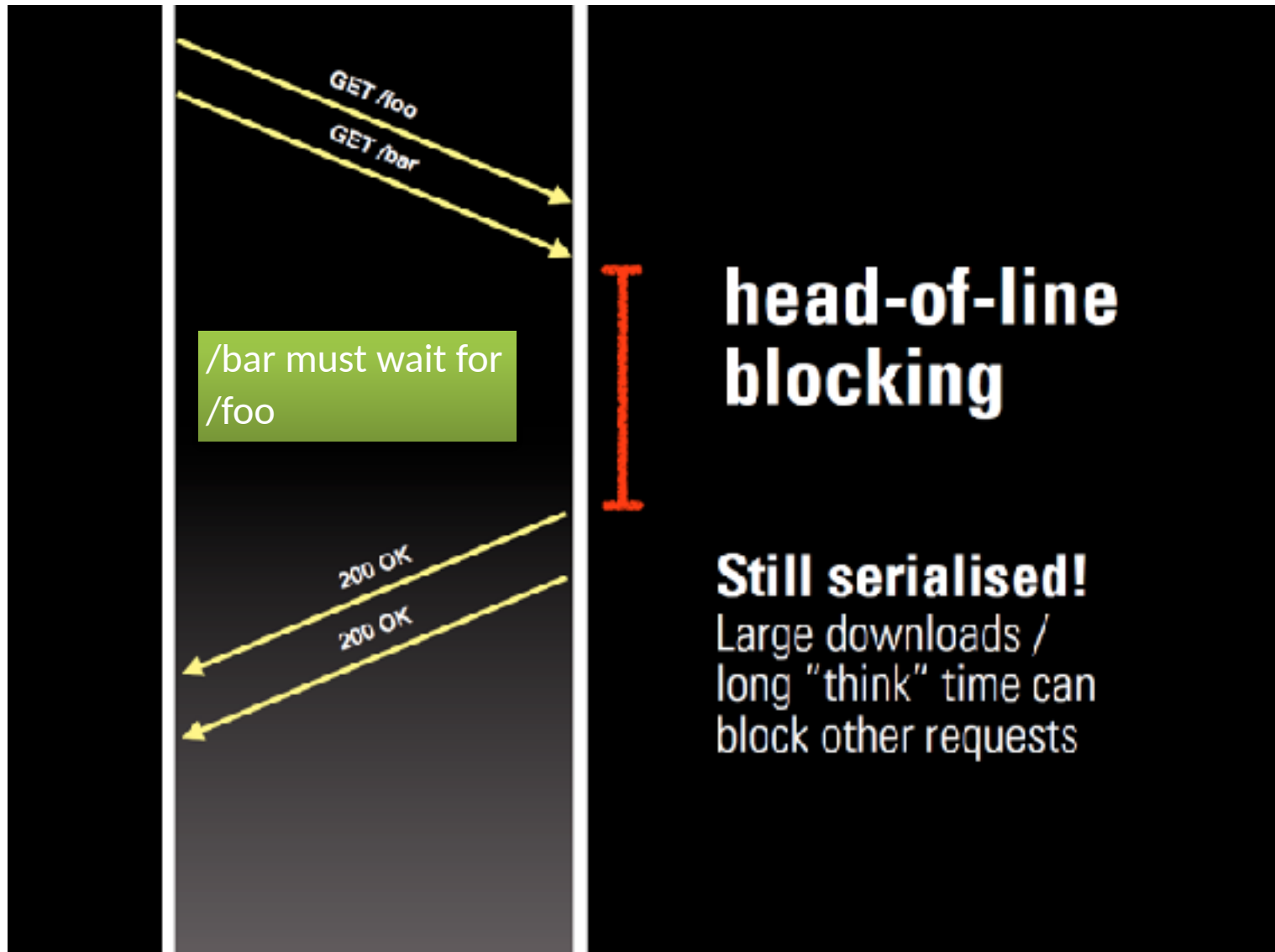


Analyse **Page Load Time** (PLT):

- TCP-Verbindungsaufbau: eine RTT
- HTTP-Request bis Empfang des erstes Bytes der HTTP-Response: eine RTT
- pro Objekt:
 - Zeit für das Übertragen der Daten auf der Leitung
- PLT:
 - HTML-Code: 2 RTT + Übertragungsdauer
 - alle Inline-Objects: RTT + Übertragungsdauer
 - gesamt: 3 RTT + Übertragungsdauer



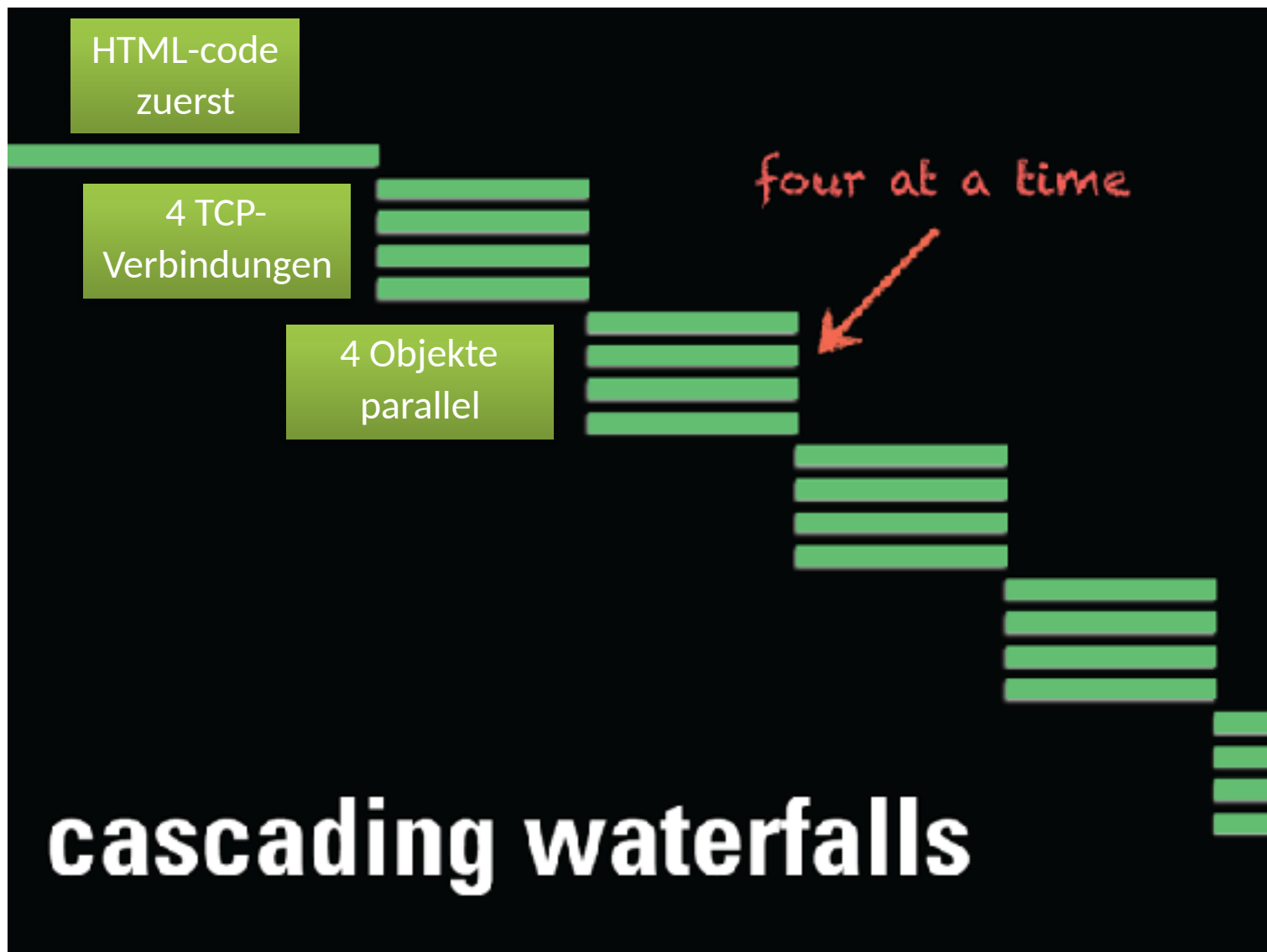
Head-of-Line-Blocking



What's so bad about that?

- **TCP is built for long-lived flows**

Web-Page Lade-Zeit als Waterfall

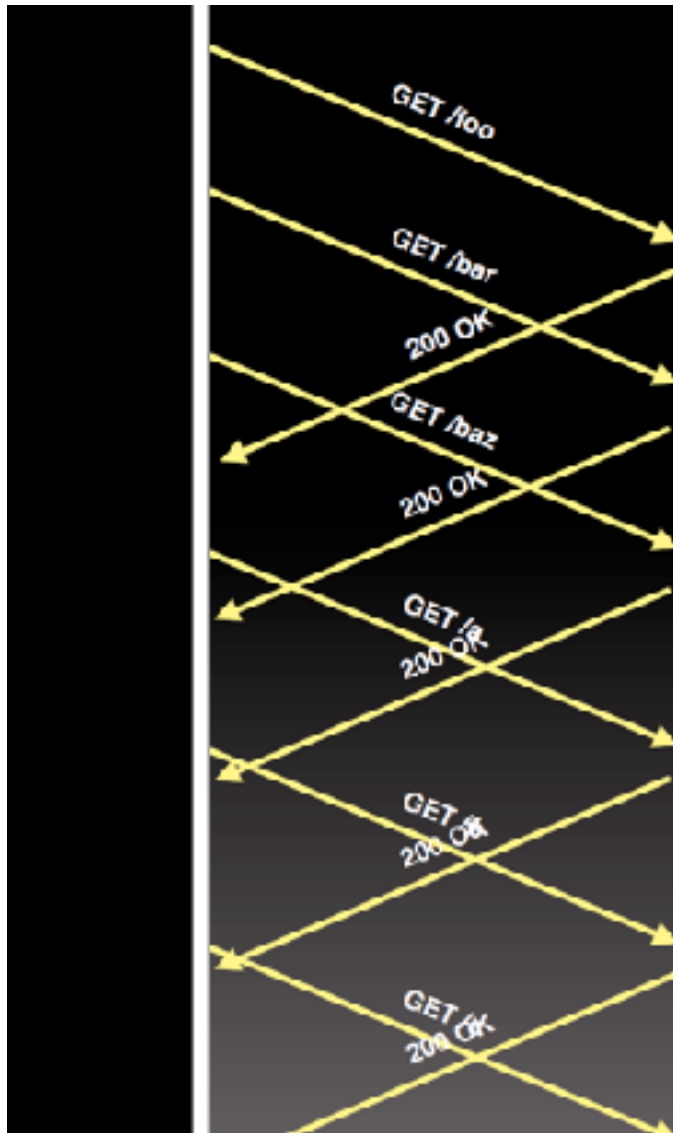


Analyse **Page Load Time** (PLT):

- TCP-Verbindungsaufbau: eine RTT
- HTTP-Request bis Empfang des erstes Bytes der HTTP-Response: eine RTT
- pro Objekt:
 - HTTP-Request bis Empfang des erstes Bytes der HTTP-Response: eine RTT
 - Zeit für das Übertragen der Daten auf der Leitung
- PLT für zwei TCP Verbindungen:
 - HTML-Code: 2 RTT + Übertragungsdauer
 - Parallel: 5 Inline-Objects pro TCP Verbindung
 - 5 RTT + Übertragungsdauer
 - Gesamt: 7 RTT + Übertragungsdauer

• Übersicht:

- Non-persistent 1.0: 22 RTT
- Persistent: 12 RTT
- Pipelining: 3 RTT
- Parallel TCP:
 - 2 TCP: 7 RTT
 - 4 TCP: 5 RTT
 - 5 TCP: 4 RTT
 - 10 TCP: 3 RTT
- jeweils plus Übertragungsdauer für Daten



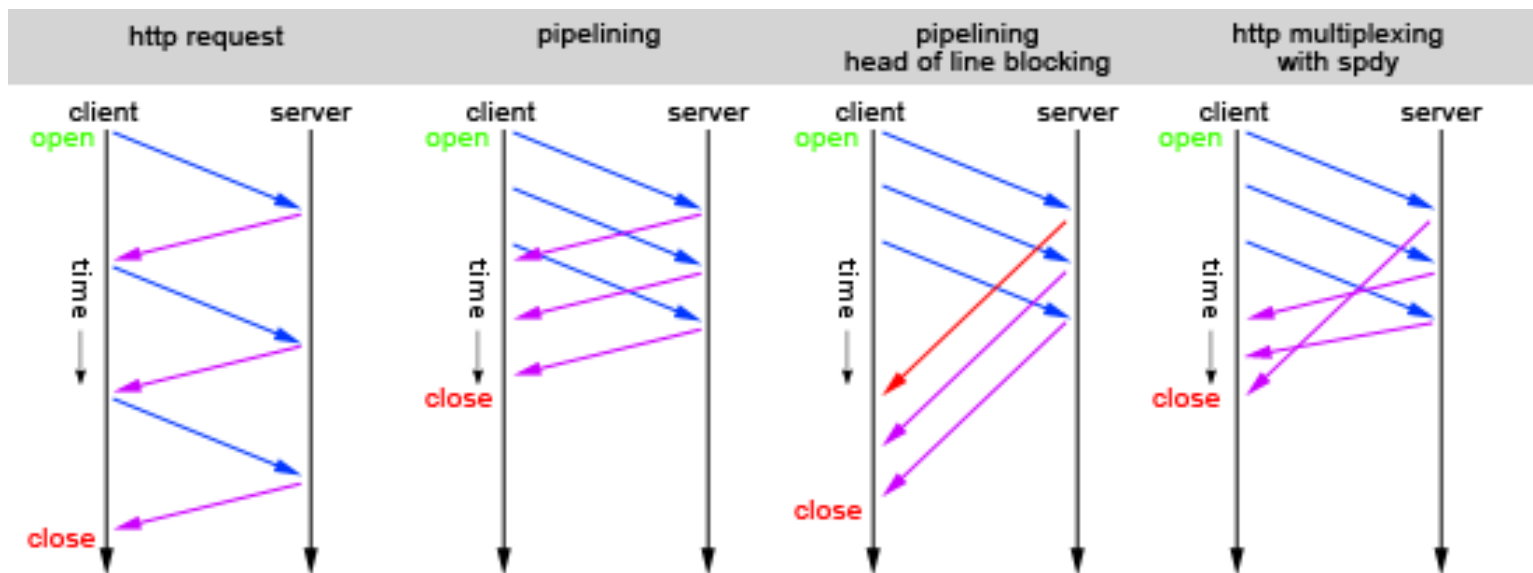
Mehrere **Streams** in einer TCP Verbindung

SPDY multiplexing

- one connection
- many requests
- prioritisation
- out of order
- interleaved

NO* QUEUING

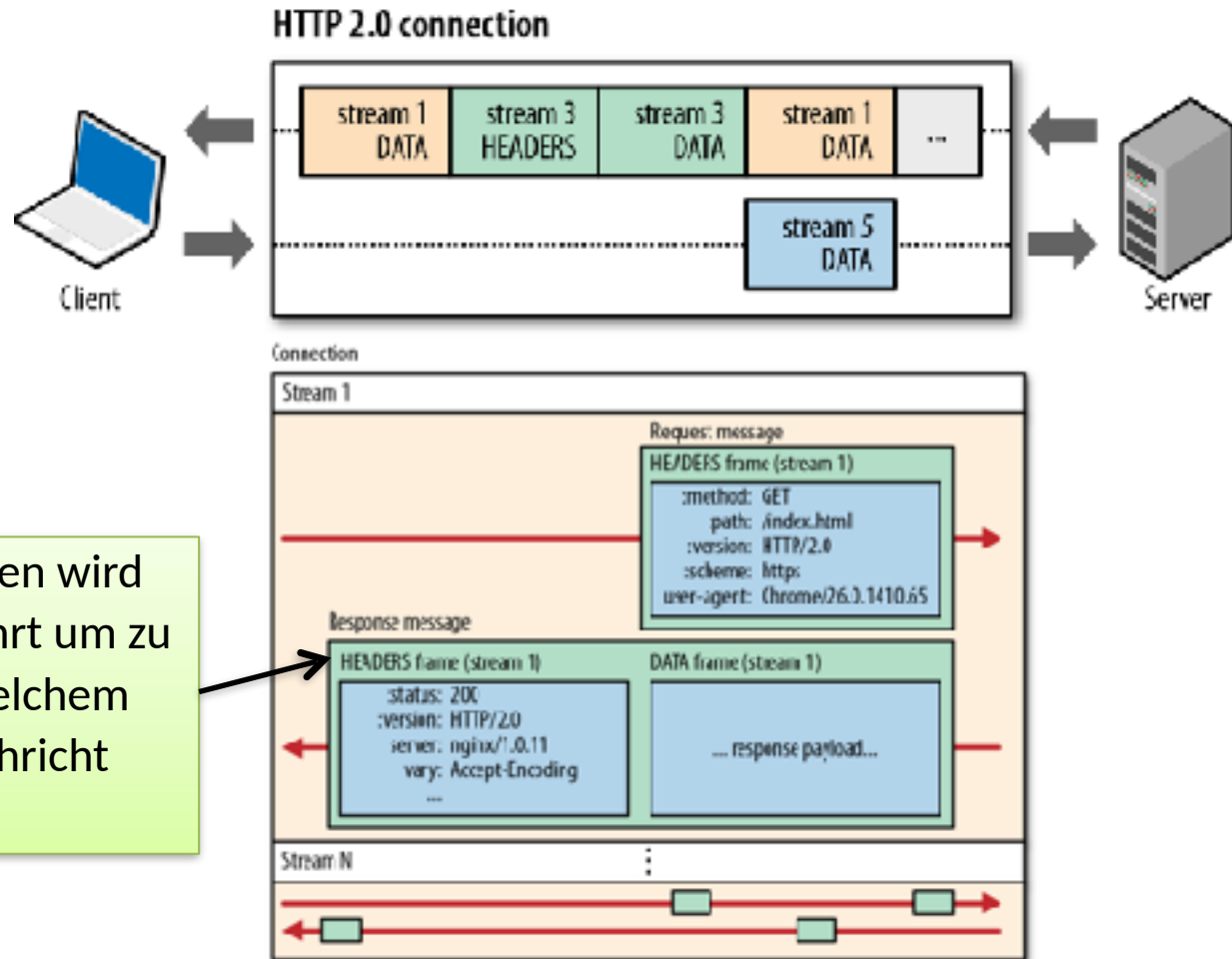
HTTP Multiplexing: Übersicht



SPDY: keine sequentielle
Übertragung der
Requests notwendig

HTTP/2.0 Features – Streams

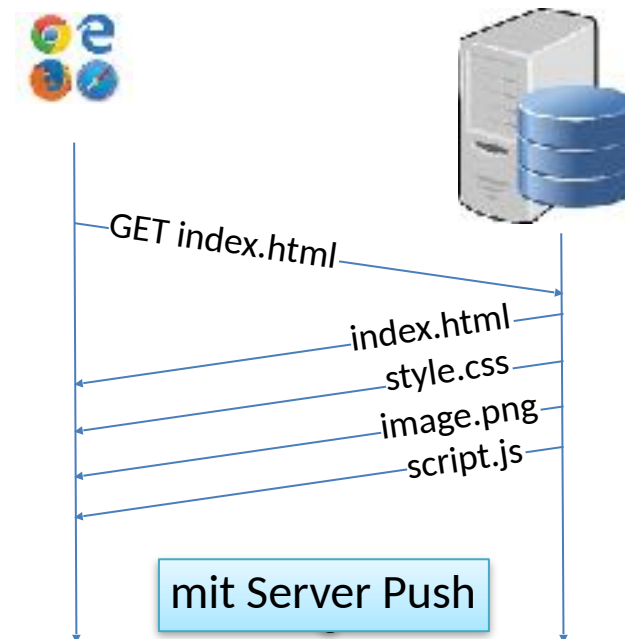
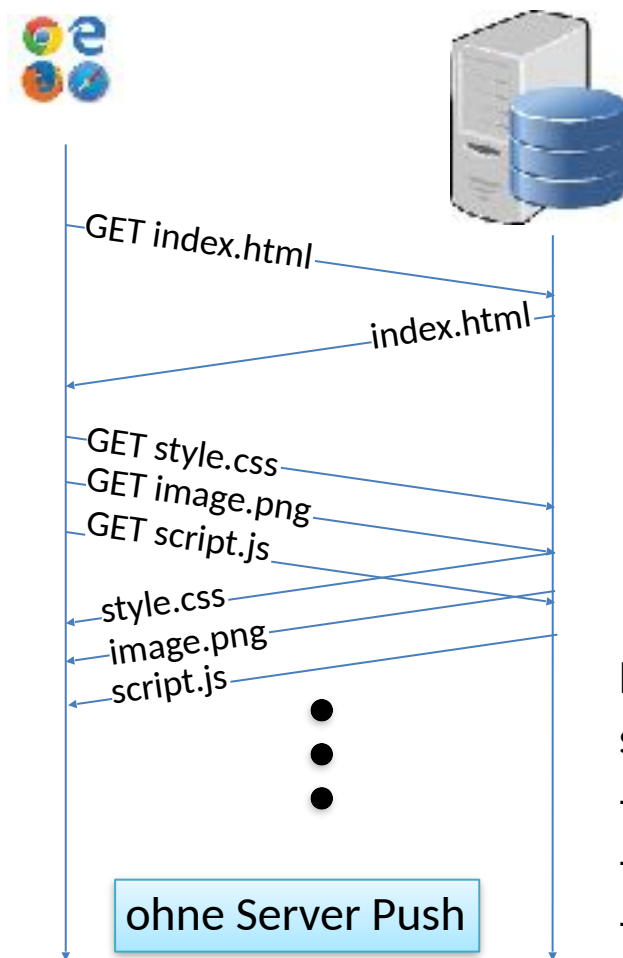
HTTP Nachrichten werden unabhängig in mehreren Streams übertragen. Eine lange Antwort in Stream 1 kann von einer schnellen Antwort in Stream 3 überholt werden. Es wird eine Flow Control (mehr dazu in Kapitel 4) und eine Priorisierung der Streams eingeführt.



In den Nachrichten wird ein Feld eingeführt um zu markieren, zu welchem Stream eine Nachricht gehört.

HTTP 2.0 Features/Server PUSH

- Mit Server PUSH kann ein Web-Server einem Browser Ressourcen (Bilder, js, etc.) schicken, bevor diese angefragt werden



Der Server kennt die Inline-Objekte einer Seite und sendet diese ohne Anfrage des Browsers

- Seite lädt schneller
- Server erhält mehr Kontrolle über Reihenfolge
- Add-Blocker kann Laden von Werbe-Inhalten vom gleichen Server nicht verhindern (evtl. das Anzeigen)

3.1 Netzanwendungen

3.2 Web und HTTP (HyperText Transfer Protocol)

3.2.1 HTTP Nachrichten

3.2.2 Web-Seiten Übertragung: Entwicklung von HTTP/1.0 bis HTTP/2.0

3.2.3 Proxies und Caches

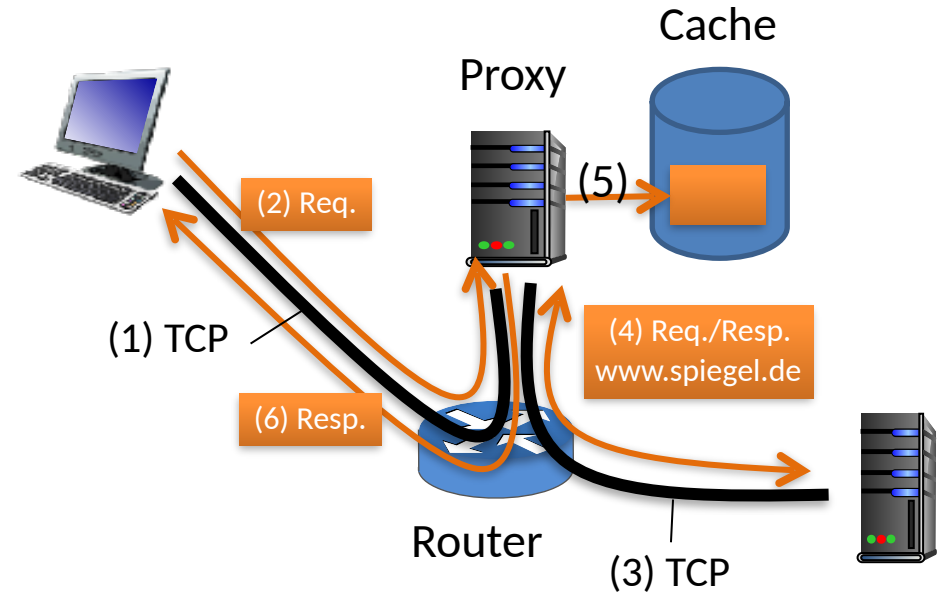
3.3 DNS (Domain Name System)

3.4 Weitere Anwendungsprotokolle: Mail und FTP

- Ein **Cache** ist ein Speicher, in dem aktuell angefragte Objekte abgelegt werden, damit sie bei wiederholten Anfragen schnell verfügbar sind
- Caches werden an verschiedenen Stellen auf dem Weg zum Server installiert
 - Browser
 - Firmennetzwerk
 - ISP
- Caches in einem Netz werden über einen **Proxy** realisiert
 - Proxy Cache
- Der Betrieb eines Caches führt im allgemeinen zu einer Win-Win-Win-Situation für Nutzer, Netzbetreiber und Serverbetreiber
- Ein **Proxy** ist ein System, das die Ende-zu-Ende Kommunikation unterbricht
- Ein Proxy tritt gegenüber dem Server als Stellvertreter für den Client auf. Im Falle eines Web-Servers stellt der Proxy einen Request für einen Client
- HTTP Proxies werden neben dem Support für Caching auch für andere Zwecke eingesetzt
 - Weiterleitung auf eine Login-Seite bei erstem Zugriff auf WLANs in Hotels, Flughäfen, etc.
- Neben HTTP Proxies gibt es noch Proxys auch noch für eine ganze Reihe weiterer Protokolle.
- bei HTTPS ist der Betrieb eines **transparent oder intercepting Proxies** nicht möglich

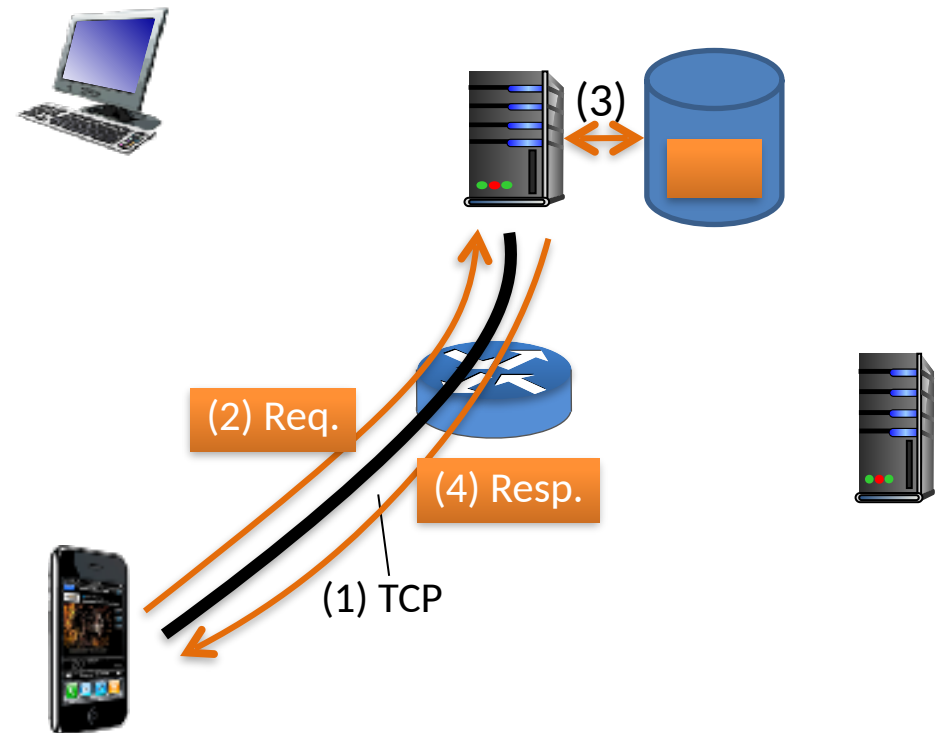
Grundprinzip: Caching Proxy

- Browser sendet alle HTTP-Requests an den Proxy
- Proxy schaut im Cache nach
 - Objekt im Cache: Proxy gibt Objekt zurück
 - Sonst:
 - Proxy fragt das Objekt vom ursprünglichen Server
 - Proxy erhält das Objekt vom Server
 - Proxy kann das Objekt im Cache speichern
 - Proxy leitet das Objekt an den Client weiter



Grundprinzip: Caching Proxy

- Browser sendet alle HTTP-Requests an den Proxy
- Proxy schaut im Cache nach
 - Objekt im Cache: Proxy gibt Objekt zurück
 - Sonst:
 - Proxy fragt das Objekt vom ursprünglichen Server
 - Proxy erhält das Objekt vom Server
 - Proxy kann das Objekt im Cache speichern
 - Proxy leitet das Objekt an den Client weiter



Arten von Proxys und Caches

Expliziter Proxy

- im Browser **konfiguriert**
- manuell oder automatisch

Impliziter (**transparenter**) Proxy

- von ISP, Firma, etc. „auf dem Pfad“ ins Internet installiert
- **fängt HTTP Requests ab** und „bedient“ sie
- auch: **intercepting** proxy, inline proxy
- Typische Umsetzung: Router leitet allen Verkehr mit TCP Port 80 an Proxy-Server

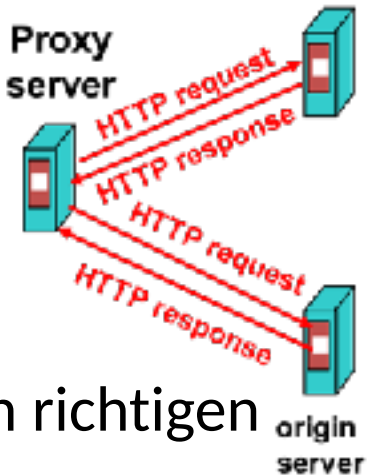
Forward Proxy (Cache)

- auf der **Client-Seite**

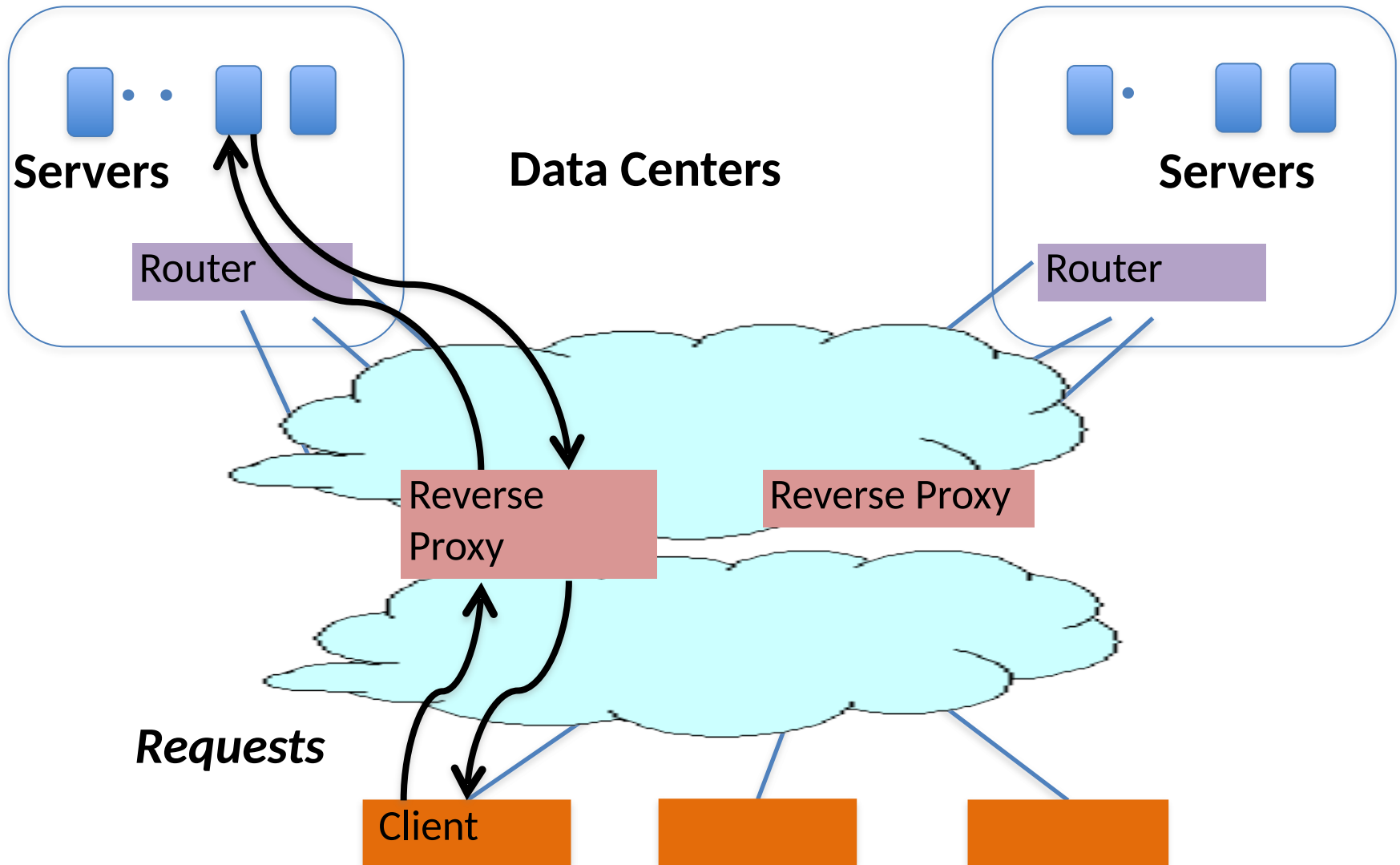


Reverse Proxy

- auf der **Server-Seite**
- leitet Anfrage an richtigen Server weiter



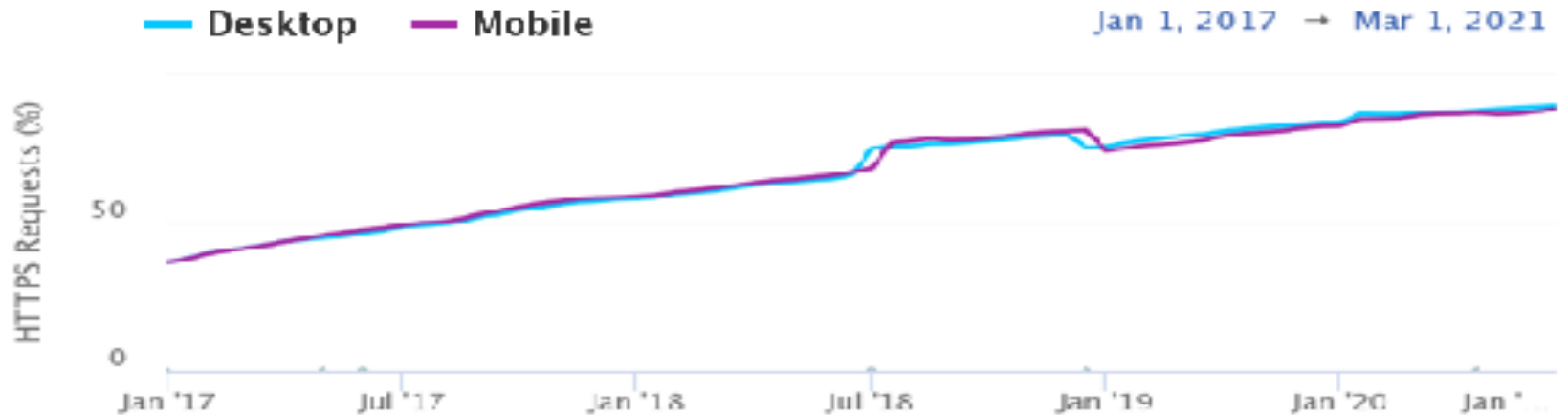
Beispiel: Google



- Gibt es Inhalte, die nicht gecacht werden können?
- Ja, z.B.
 - Dynamische Daten: Aktienpreise, Fußballergebnisse, WebCams, ...
 - CGI Skripten: Inhalt wird von übergebenen Parametern bestimmt
 - Cookies: Ergebnisse können von Daten im Cookie abhängen
 - Web-Seiten Analyse: Web-Hoster möchte Statistiken machen
 - **TLS/HTTPS**: verschlüsselte Daten

TLS/SSH Konsequenzen

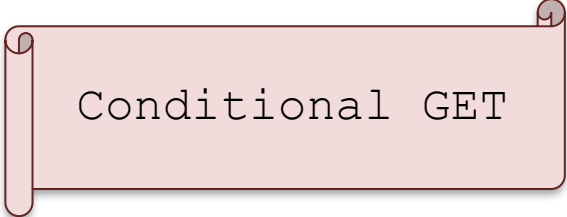
- Verschlüsselter Internetverkehr mit TLS (Transport Layer Security)/SSL (Secure Sockets Layer) nimmt zu
 - Dezember 2020: knapp 90% der Requests nutzen https (http-archive)



- Konsequenz:
 - 2017: Neues Internet-Ecosystem ohne Caches???
 - 2019: Neues Internet-Ecosystem !!!
 - ISPs betreiben kaum Caches für http
 - (erzwungene) Kooperation von ISPs mit CDNs (zu den Bedingungen der CDNs)
 - Google Global Cache: ISPs "dürfen" in ihrem Netz Google-Caches betreiben
 - Ähnliche Lösungen für Netflix, Akamai, Amazon
 - Firmen können Caching mit automatisch vorkonfigurierten Proxies/Caches betreiben

Grenzen der Cachebarkeit

- Gibt es Inhalte, die nicht gecacht werden können?
- Ja, z.B.
 - Dynamische Daten: Aktienpreise, Fußballergebnisse, WebCams, ...
 - CGI Skripten: Inhalt wird von übergebenen Parametern bestimmt
 - Cookies: Ergebnisse können von Daten im Cookie abhängen
 - Web-Seiten Analyse: Web-Hoster möchte Statistiken machen
 - **TLS/HTTPS**: verschlüsselte Daten
- Zusätzliche Problems:
 - ungültige nicht mehr aktuelle Inhalte
 - Cache-Proxys müssen aktuellen Inhalt liefern
 - zusätzlicher Aufwand



Conditional GET

Conditional GET (Bedingter GET-Request)

Conditional GET

Ziel: Client/Cache fragt Server nur nach neuerer Version seiner Kopie

Cache:

- liefert Datum mit, wann Objekt gecacht wurde

`If-modified-since: <date>`

Server:

- wenn gecachte Version aktuell ist:
Response Code 304

`HTTP/1.0 304 Not Modified`

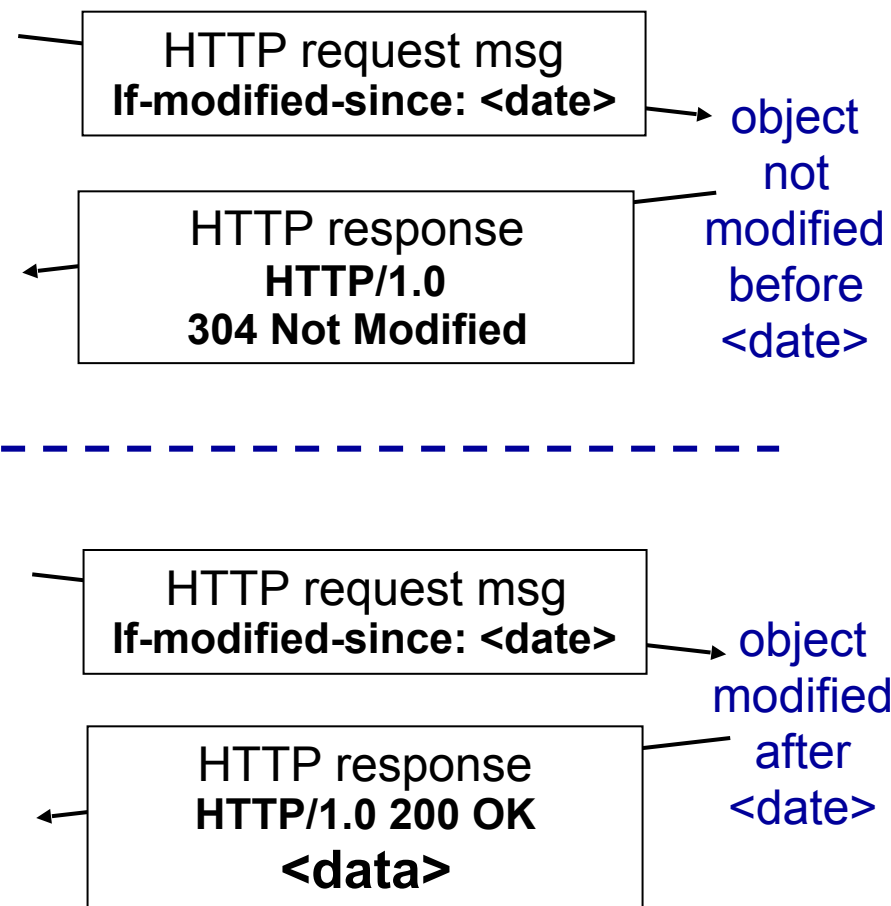
- sonst: Objekt übertragen



client



server

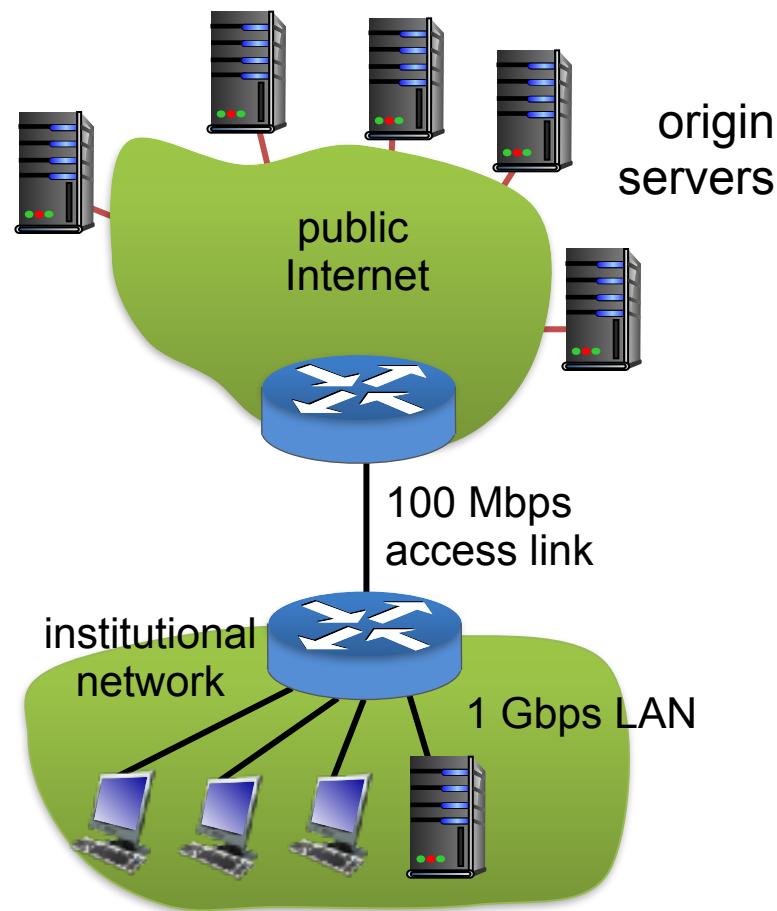


Annahmen

- Durchschnittliche Größe eines Objektes = 1 Mbyte
- Durch. Rate von Anfragen aller Webbrowser der Firma = 15/s
- Verzögerung v. Router d. Firma zum Server und zurück = 20 ms

Resultat

- Auslastung des LAN = 12%
 - Auslastung der Zugangsleitung = 120%
 - Verzögerung = Internet + Zugangsleitung + LAN
- = 20ms + Sekunden+ Millisekunden



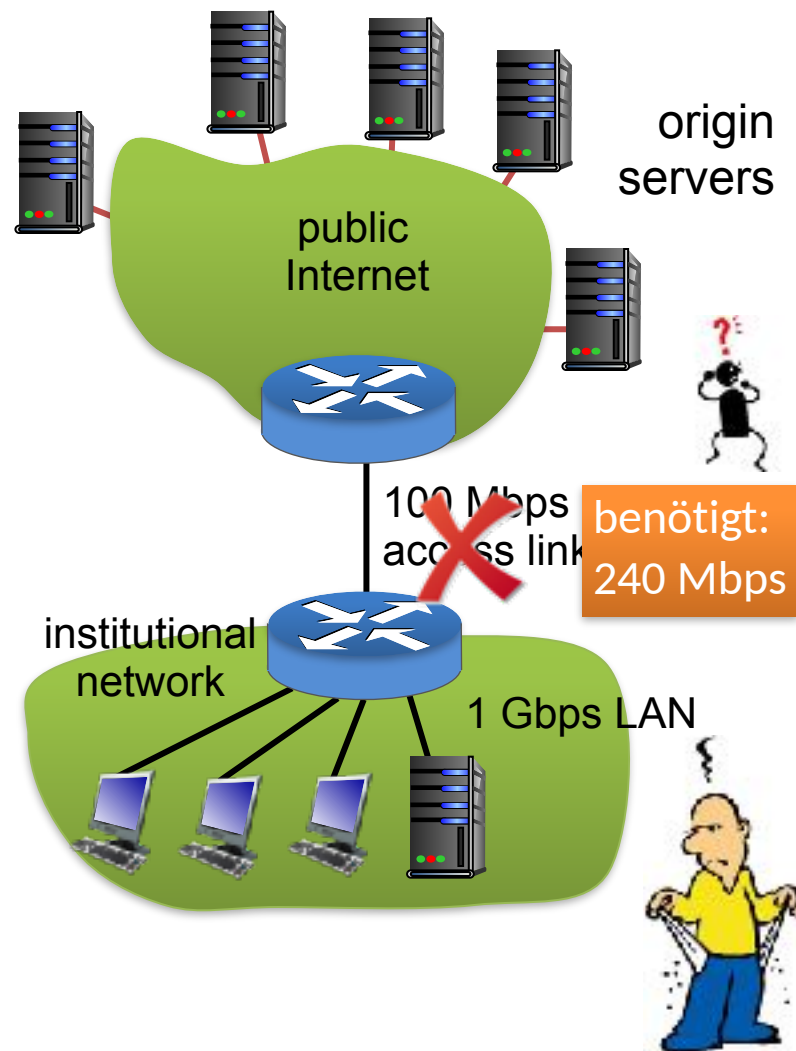
Annahmen

- Durchschnittliche Größe eines Objektes = 1 Mbyte
- Durch. Rate von Anfragen aller Webbrowser der Firma = 15/s
- Verzögerung v. Router d. Firma zum Server und zurück = 20 ms

Resultat

- Auslastung des LAN = 12%
 - Auslastung der Zugangsleitung =
 - Verzögerung = Internet + Zugangsleitung + LAN
- = 20ms + Sekunden + Millisekunden

Ziel:
max: 50%



Beispiel für Caching-Effizienz

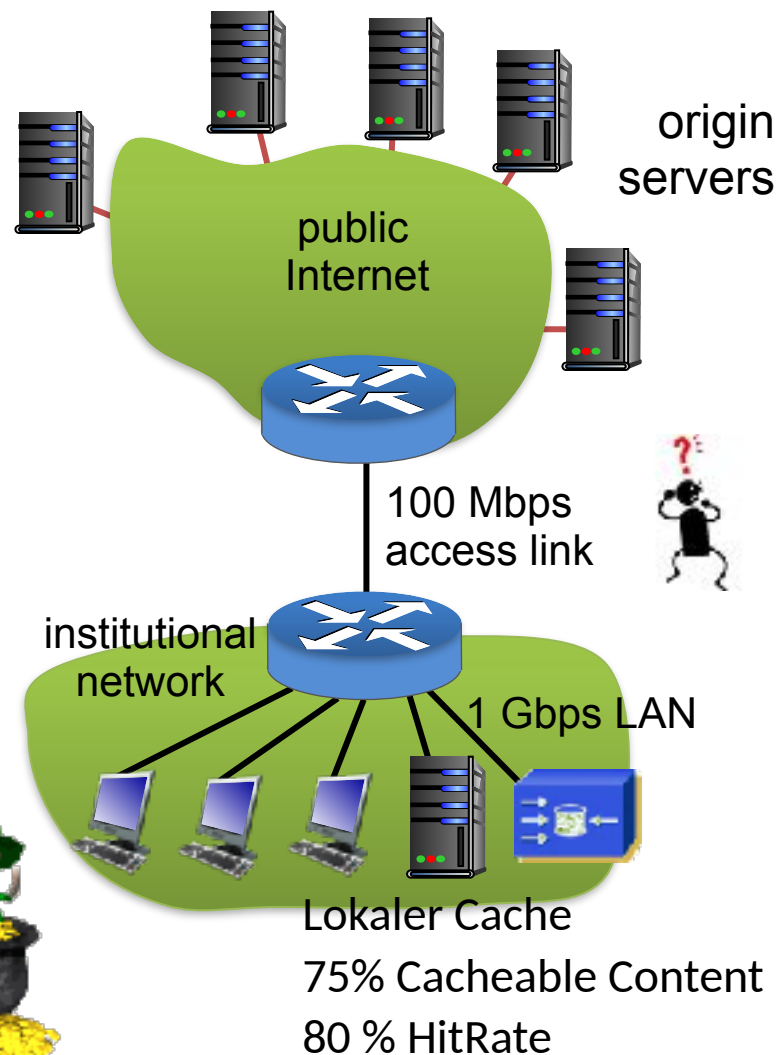
Annahmen

- Durchschnittliche Größe eines Objektes = 1 Mbyte
- Durch. Rate von Anfragen aller Webbrowser der Firma = 15/s
- Verzögerung v. Router d. Firma zum Server und zurück = 20 ms

Resultat

- Auslastung des LAN = 12 %
- Auslastung der Zugangsleitung =
- Verzögerung
 - Cache: Millisekunden
 - Server: 20ms + Millisekunden

Auslastung:
48%



Beispiel: Berechnung

- Gesamtverkehr=Volumengröße*Anfragerate
 - $1 \text{ MB} * 15/\text{s} = 120 \text{ Mbps}$
- LAN-Auslastung=Gesamtverkehr/LAN-Kapazität
 - $120\text{Mbps}/1 \text{ Gbps} = 12 \%$
- Access-Auslastung=Gesamtverkehr/Access-Kapazität
 - $120 \text{ Mbps}/100 \text{ Mbps} = 120\%$ (überlastet)
- Benötigte Access-Kapazität=Gesamtverkehr/Ziel-Access-Auslastung
 - $120 \text{ Mbps}/0.5=240 \text{ Mbps}$
- Gecachter Verkehr=Gesamtverkehr*Cachebarer Anteil*Hit-Rate
 - $120 \text{ Mbps} * 0.75 * 0.8 = 72 \text{ Mbps}$
 - Hit-Rate: Anteil der Anfragen für cachebaren Inhalt, die aus dem Cache bedient werden
- Nicht gecachter Verkehr=Gesamtverkehr-gecachter Verkehr
 - $120 \text{ Mbps}-72 \text{ Mbps}= 48 \text{ Mbps}$
- Auslastung Access-Link: Nicht gecachter Verkehr/Link-Kapazität:
 - $48 \text{ Mbps} / 100 \text{ Mbps}= 48 \%$