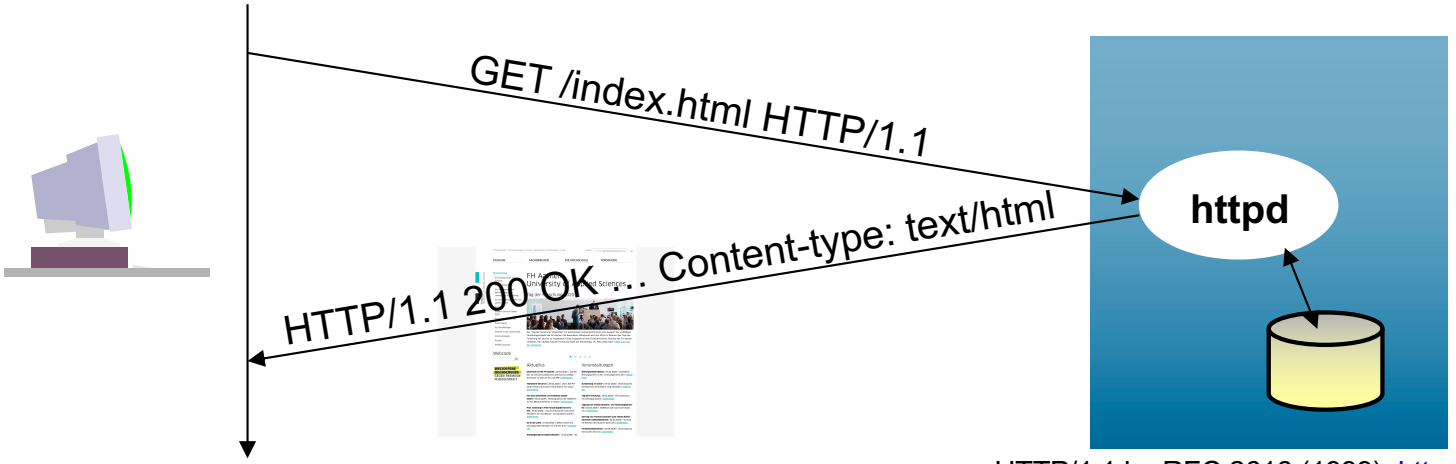
HTTP - Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP)

Das Protokoll des WWW: HTTP

- Zustandsloses Anfrage-/Antwort-Protokoll
- Textbasiert (pre 2.0)
- Dient der Übertragung von Ressourcen zwischen einem Server und einem Client
  - > Übertragung von Datenobjekten: Formatangaben, Inhalt, Server-Einstellungen
  - > Kodierungsregeln (analog zu E-Mail)
- Endpunkt wird über URL identifiziert:
  - > Protokoll://Host:Port/Ort?Parameter



HTTP/1.1 im RFC 2616 (1999): https://tools.ietf.org/html/rfc2616

URI: Uniform Resource Identifier

- Eine URI dient der eindeutigen Adressierung von abstrakten und physikalischen Ressourcen im Internet (Spezifikation in RFC2396)
- URI: URLs ∪ URNs

URI (**U**niform **R**esource **I**dentifier)

URL (**U**niform **R**esource **L**ocator)
Adressierung von Informationsobjekten mit Festlegung des Zugangs-Protokolls (Ort der Ressource). RFC2141

URN (**U**niform **R**esource **N**ame)
Adressierung von Objekten ohne ein Protokoll festzulegen (Eindeutige und gleichbleibende Referenz – Name der Ressource). RFC1738

Weitere Informationen: https://www.ietf.org/rfc/rfc3986.txt

HTTP
Adressierung

Unterschied URL/URI

- URI beschreibt allgemeinere Möglichkeit der Adressierung
- Jede URL ist auch eine URI
- Eine URL identifiziert Ressourcen eindeutig und gibt dabei gleichzeitig das **Zugriffsprotokoll** an

Beispiele:

- URI: example.com
- URI (oder eine relative URL): datei.txt
- URL: http://www.example.com/datei.txt
- URL: mailto:test@example.com
- URL: ftp://127.0.0.1/dump.sql
- URN: urn:isbn:0596517742

> URLs und URNs sind auch gleichzeitig gültige URIs

HTTP
Adressierung

URL-Syntax

- <Schemata>:<Schemata-spezifischer-Teil>
- Viele URL-Schemata besitzen einen hierarchischen Aufbau.

Beispiel: HTTP URL

```
http://<user>:<password>@<host>:<port>/<url-path>?<searchpart>
```

```
http://user:pw@example.com:80/datei.html?q=123
```

Relative URLs

- Auch die Nutzung relativer URLs wurde definiert
- Beispiel: `../datei.html`
- Nicht relative URLs werden als „absolut“ bezeichnet

Weitere Informationen: https://www.ietf.org/rfc/rfc1738.txt, https://tools.ietf.org/html/rfc1808

Basis-URL

- Für das Verständnis relativer URLs ist ein Analogon zur „Working Directory“ wichtig, die Basis-URL
- Die „Basis-URL“ einer mittels einer absoluten URL-spezifizierten Internet-Ressource beinhaltet alle Zeichen bis (einschließlich) zum letzten Schrägstrich (/) im Pfadnamen

Absolute URL	Basis URL
http://www.fh-aachen.de/abt_juelich.html	http://www.fh-aachen.de/
http://www.example.com/dir/inhalt.html	http://www.example.com/dir/

Relative URL

- Eine partielle (relative) URL ist immer dann vorhanden, wenn Schemataname und ":" fehlen.
- Aus der Kombination von Basis-URL (die, der aktuell angezeigten Seite im Browser) und relativer URL lässt sich immer eine absolute URL ableiten

Beispiel:

- Basis-URL: `http://www.example.com/html/`

Relative URL	Absolute URL
about.html	http://www.example.com/html/about.html
path/	http://www.example.com/html/path/
path/file.html	http://www.example.com/html/path/file.html
/	http://www.example.com/
//example.org/	http://example.org/
/config/	http://www.example.com/config/
../	http://www.example.com/
../dir/	http://www.example.com/dir/
../../../	http://www.example.com/
./	http://www.example.com/html/
./about.html	http://www.example.com/html/about.html

Dateinamen

- RFC3986 empfiehlt zwar die UTF-8 Kodierung, jedoch gibt es hier keinen expliziten Standard. Um etwaige Konflikte zu vermeiden sollten **besser nur ASCII-Zeichen** als Dateiname genutzt werden
- Andere Zeichen können zwar verwendet werden, sollten aber besser kodiert werden (auch Leerzeichen)
- Diverse ASCII-Zeichen wie u.a. %+&=: sind bereits für andere Aufgaben vorgesehen und können ebenfalls so nicht direkt auftauchen.

- Für all diese Zeichen gibt es deshalb eine andere Darstellungsform
  - > Leerzeichen werden zu einem „+“ kodiert („%20“ funktioniert alternativ)
  - > Nationale Sonderzeichen werden durch ihre hexadezimale Darstellung entsprechend ASCII-Zeichensatz dargestellt. Zur Erkennung dieser Hex-Zeichen wird ein "%" vorangestellt.
    - Beispiele: ß wird zu %DF,] zu %5D, } zu %7D
  - > **Browser zeigen heutzutage häufig die dekodierten Werte an**, was dazu führt, dass der Nutzer häufig gar nichts von der Umwandlung merkt

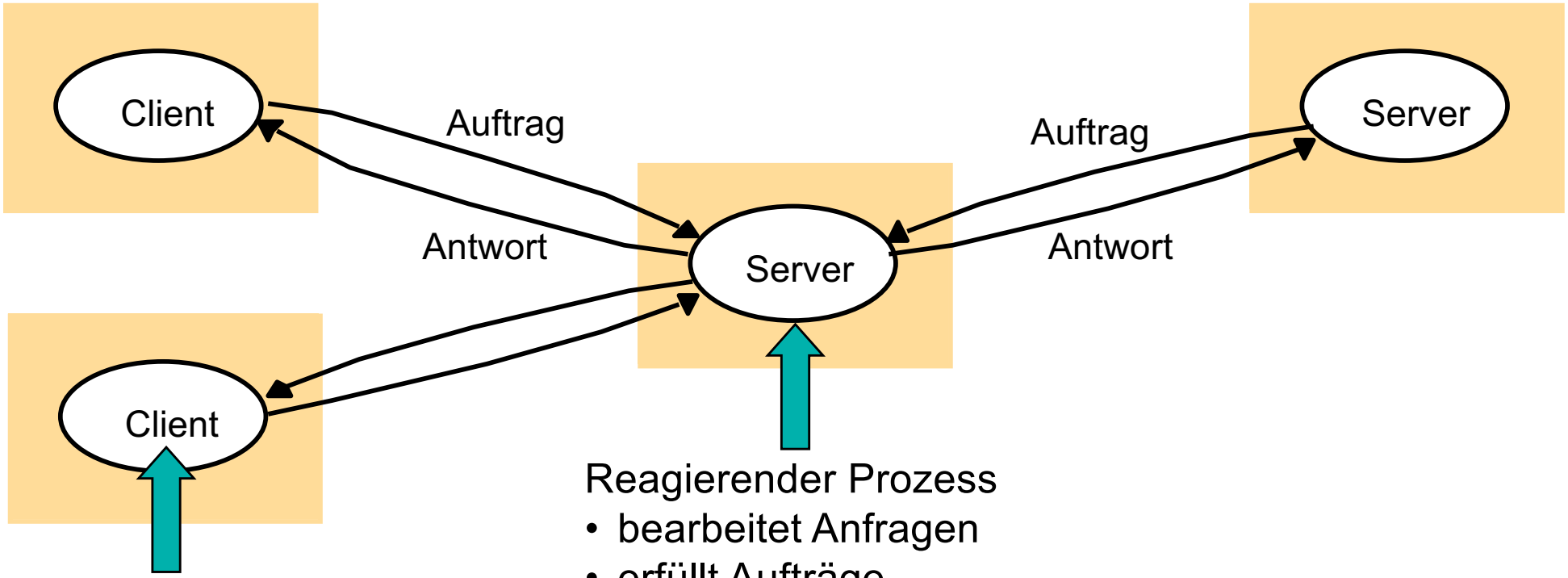
Weitere Informationen: https://tools.ietf.org/html/rfc3986

Im ISO/OSI-Modell (Siehe Kommunikationssysteme)

	Schicht	Einheit	Protokoll
7	Application	Anwendung	**HTTP**
6	Presentation		
5	Session		
4	Transport	Transport	TCP
3	Network	Internet	IP
2	Data Link	Netzzugriff	Ethernet
1	Physical		

HTTP
Client-Server-Modell

- Ein kurzlebiger Client-Prozess stellt Anfragen an einen langlebigen Server-Prozess
- Die Mehrzahl aller verteilten Anwendungen arbeiten nach dem Client / Server Modell



Initiierender Prozess
- stellt Anfragen
- erteilt Aufträge

Reagierender Prozess
- bearbeitet Anfragen
- erfüllt Aufträge
- Nutzt manchmal andere Dienste (agiert hier also als Client)

Client-Server-Modell

- Client sendet *Request*
- Server antwortet mit *Response*

Eigenschaften

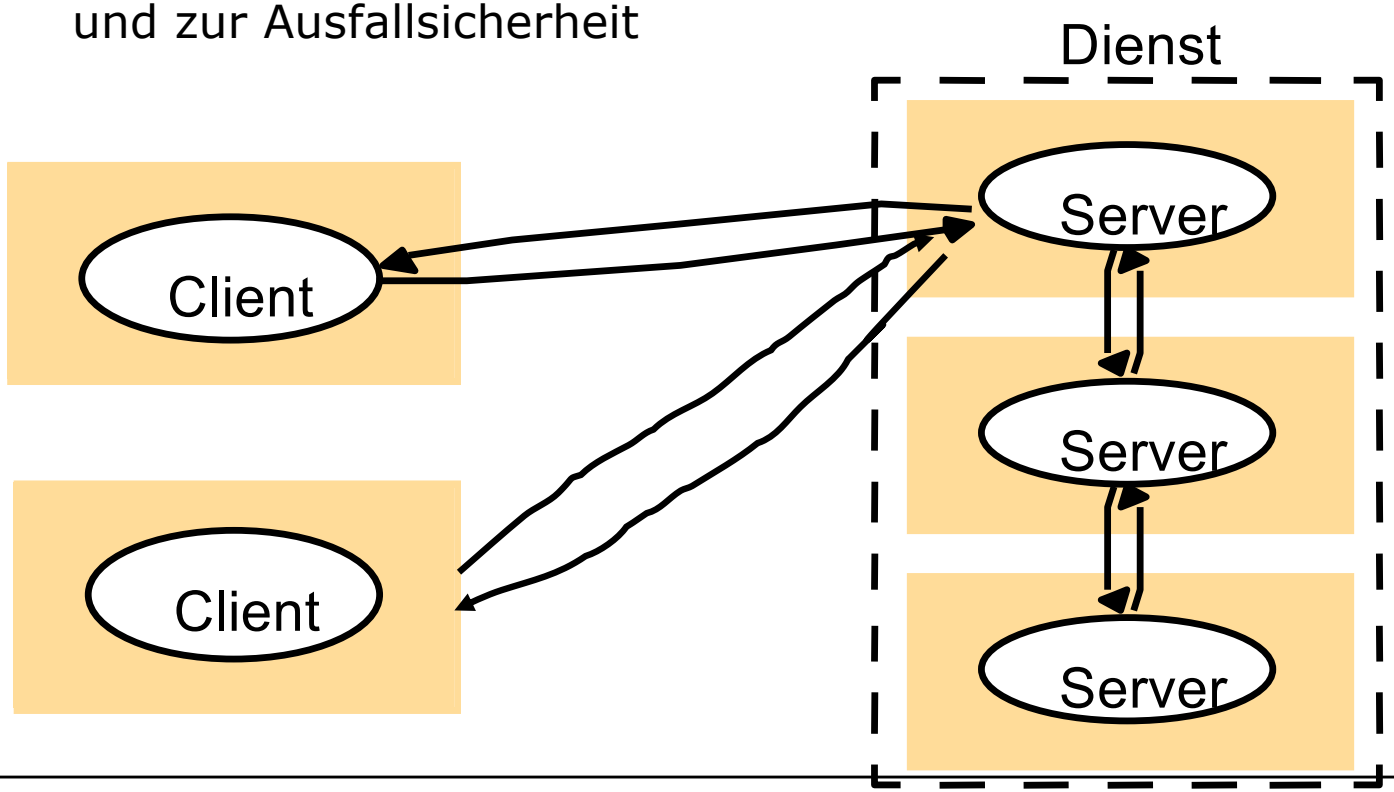
- Ein Server ist langlebig und stellt einen Dienst zur Verfügung
  - > Klassisch Anfrage-Antwort-Zyklus
  - > Webseite: Browser (Client) erfragt Ressource, Server antwortet (Response)
- Ein Server kann bei der Beantwortung der Anfrage zwischenzeitig als Client gegenüber einem anderen Server agieren
  - > Beispiel: Proxy-Server, Datenbank
- Begriff „Server" wird häufig sowohl für den Dienst (z.B. Apache) als auch für die Maschine auf der der Dienst angeboten wird genutzt
- Zustandslosigkeit vereinfacht den Server. Die Anfrage enthält alles an Information, was man zur Bearbeitung benötigt. Damit wird oftmals eine Verbindung nach Erhalt der Antwort geschlossen.

Grundsätzlicher Ablauf

1. Verbindungsaufbau
   > Der Client baut eine TCP/IP-Verbindung zum Server auf.
2. Request
   > Der Web-Client sendet einen HTTP-Anfrage (Kommando), die die Spezifikation des gewünschten Dokumentes in Form von URL und die Protokoll-Version enthält. Weitere Header-Elemente oder Anfrageparameter steuern das Verhalten.
3. Response
   > Der Server antwortet mit der Protokoll-Version und einem Statuscode enthält, der Erfolg bzw. Misserfolg der Anforderung mitteilt. Neben steuernden Header-Elementen folgt, sofern angefordert, das eigentliche Dokument in dem entsprechenden Datenformat (in der Regel HTML).
4. Schließen der Verbindung
   > **Bei HTTP 1.0 wird die Verbindung geschlossen**. Seit **HTTP/1.1** kann der Client mittels **„keep-alive"** alternativ mitteilen, dass die TCP/IP-Verbindung offen bleiben soll um zukünftige Anfragen zu beschleunigen. Die Schritte 4 und 1 entfallen dann bei zukünftigen Anfragen.

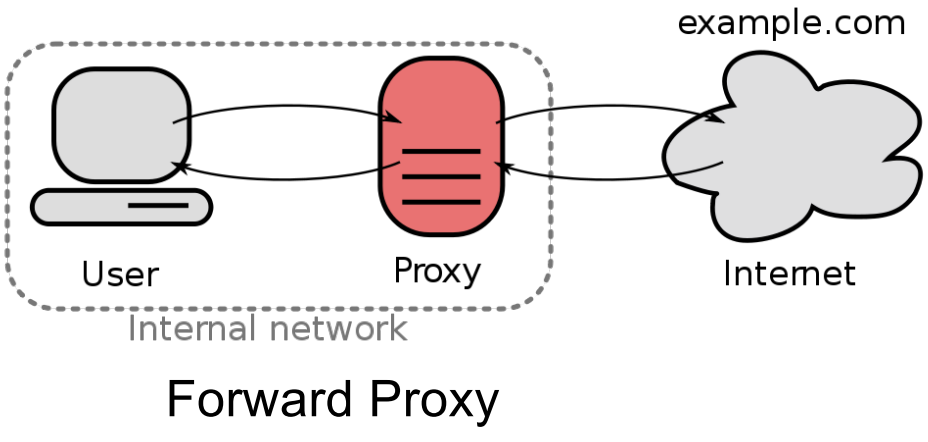
Replizierte Server

- Es werden Replikate von Serverprozesse zur Verfügung gestellt
- Jeder Server hat hier quasi die gleiche Sicht!
- Beispiele:
  - > Google, eBay, ...
  - > Transparente Replikate in Clustern zur Verbesserung der Performance und zur Ausfallsicherheit



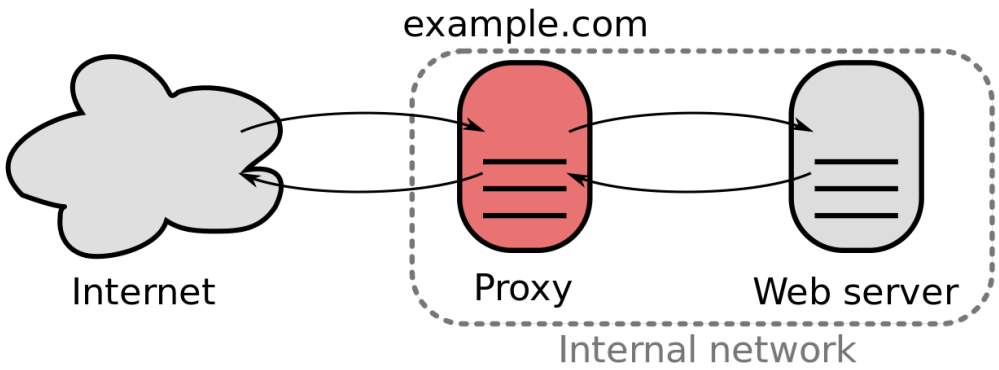
(Forward) Proxy und Reverse-Proxy Modell

- Ziele
  - > Verbesserung der Performance
  - > Filter und Kontrollpunkt
- Proxy zum Zwischenspeichern oder Lastbalanzieren von Webseiten



Forward Proxy

https://upload.wikimedia.org/wikipedia/commons/thumb/1/19/Forward_proxy_h2g2bob.svg/500px-Forward_proxy_h2g2bob.svg.png



Reverse Proxy

https://upload.wikimedia.org/wikipedia/commons/thumb/6/67/Reverse_proxy_h2g2bob.svg/1280px-Reverse_proxy_h2g2bob.svg.png

Content Delivery Networks
CDN

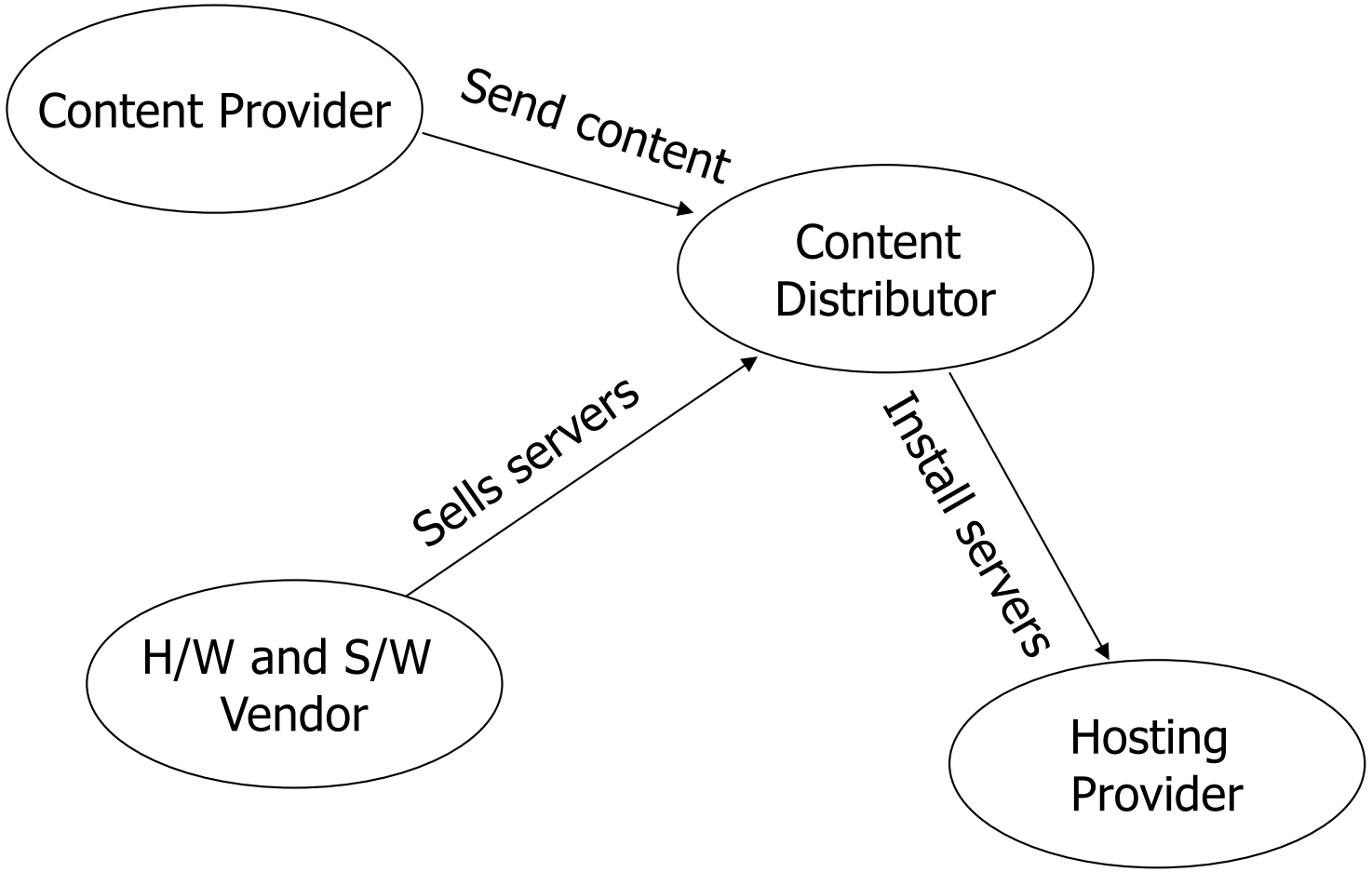
Große Anbieter wollen nahe beim Nutzer sein

Einbinden großer Frameworks soll skalierbar sein und nicht immer nur von einem Server geladen werden

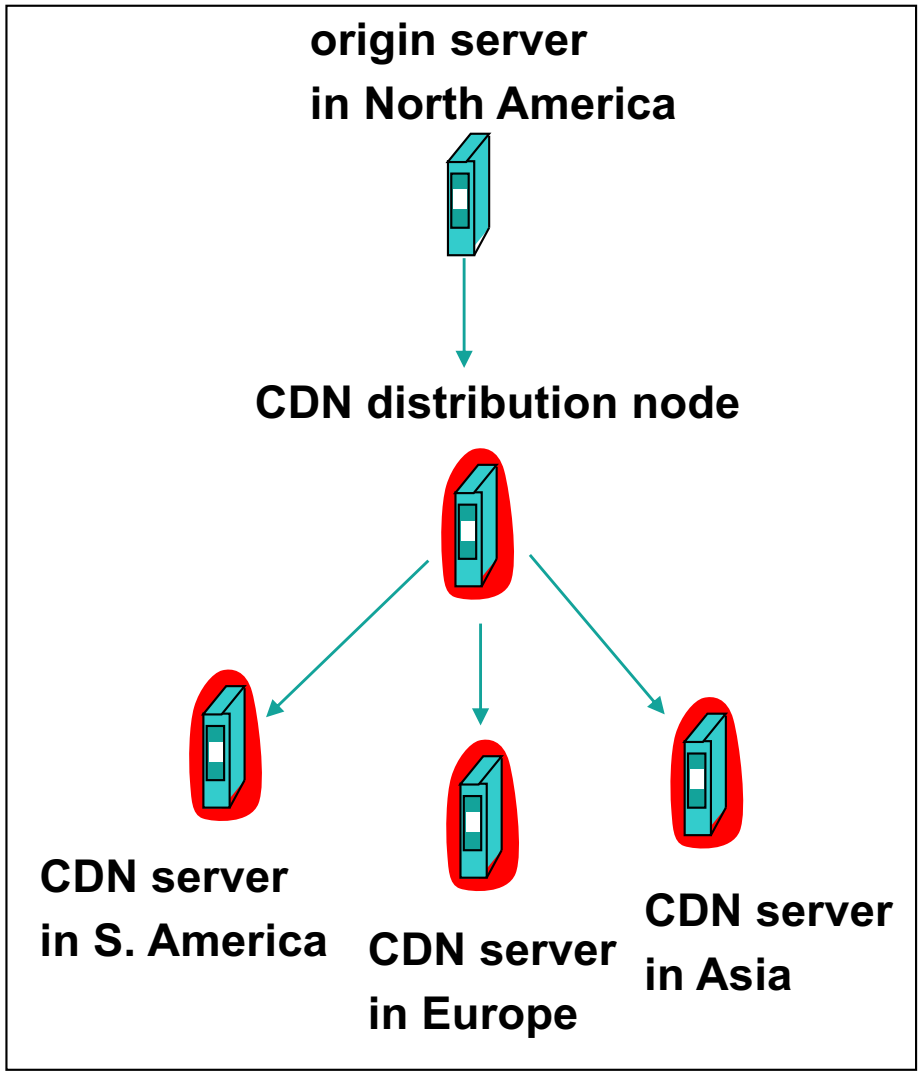
Ein CDN bietet einen Dienst an, um den Inhalt auf multiple Server zu replizieren

Clients können dann den Server nutzen, der den Inhalt am schnellsten liefert

Content Delivery Networks
CDN



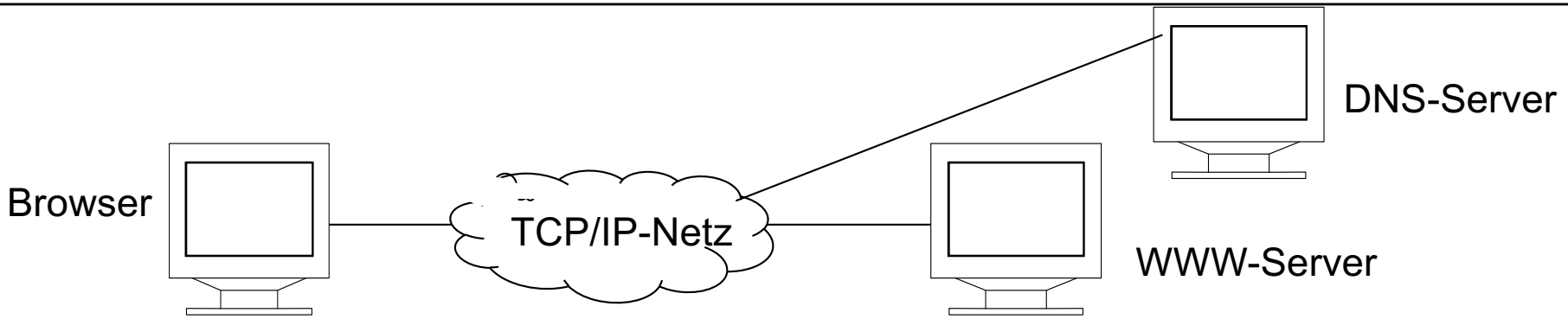
Content Delivery Networks
CDN



Clients nutzen einen in HTTP verankerten Umleitungsmechanismus (Redirection) um auf den passenden Server zuzugreifen

Aus Kurose-Ross: Computer Networking: A Top-Down Approach

HTTP
Ablauf einer Anfrage



Browser fragt DNS nach der IP-Adresse des Servers

DNS antwortet

Browser öffnet eine TCP-Verbindung zu Port 80 des Rechners

Browser sendet das Kommando GET bzgl. der entsprechenden Datei

WWW-Server schickt die Datei Datei zurück

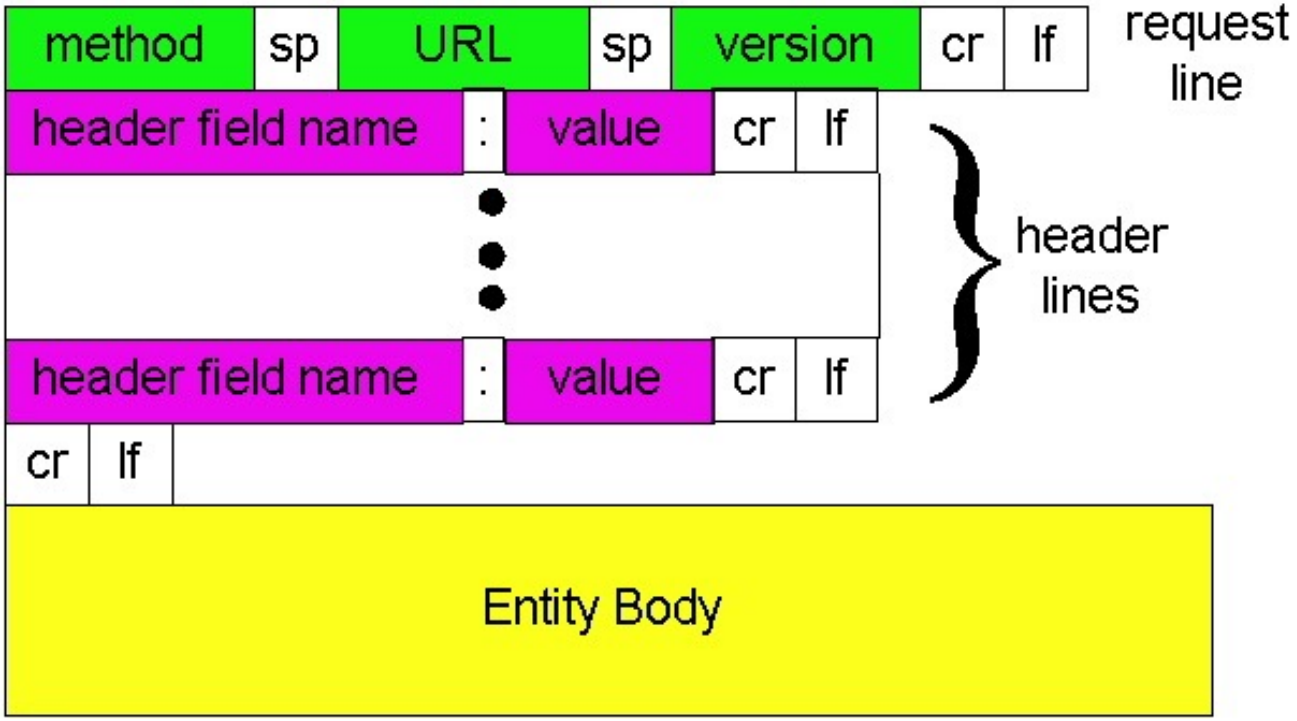
Verbindung wird wieder abgebaut.

Formaler Aufbau von Request und Response

```
Request          = Request-Line
                   *((general-header | request-header | entity-header)) <CRLF>
                   [message-body]

Response         =  Status-Line
                   *((general-header | response-header | entity-header)) <CRLF>
                   [message-body]

Request-Line     = Method blank Request-URI blank HTTP-Version <CRLF>

Status-Line      =  HTTP-Version blank Status-Code blank Reason-Phrase <CRLF>
```

HTTP/1.1 im RFC 2616 (1999): https://tools.ietf.org/html/rfc2616

HTTP
Request

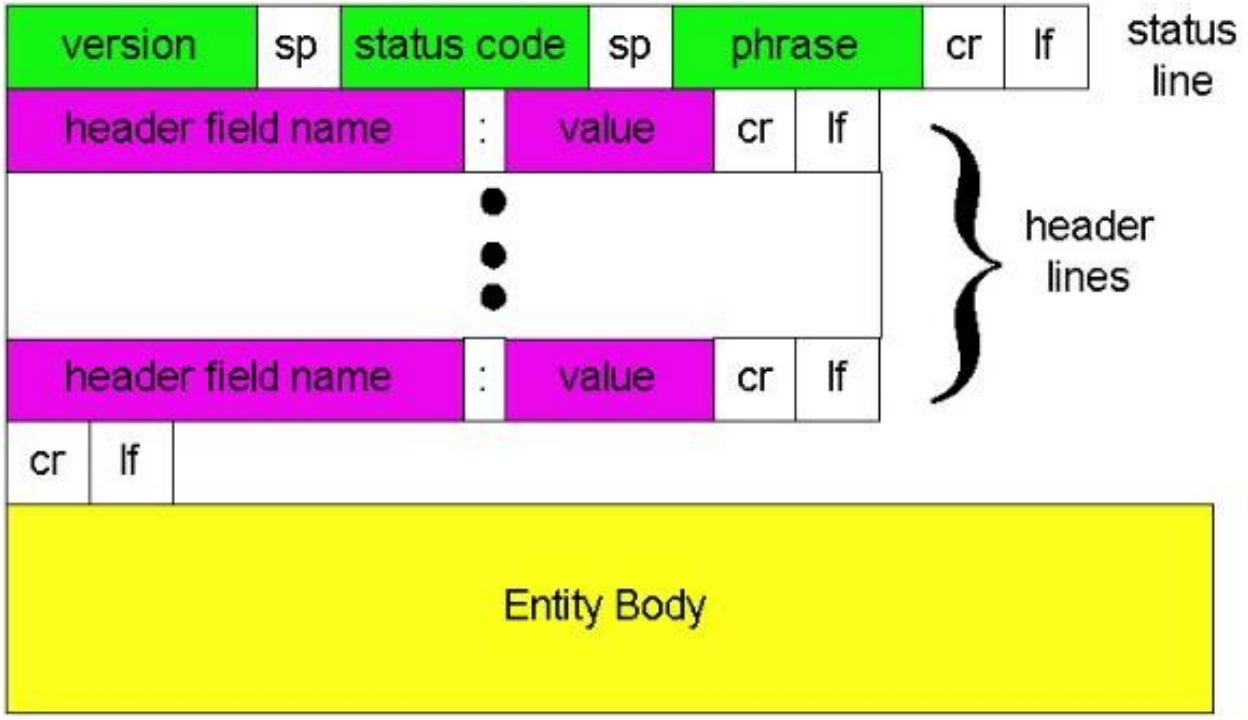


Quelle: http://userpages.umbc.edu/~dgorin1/451/OSI7/dcomm/http.htm

Request-Beispiel (GET):

```
GET /index.html HTTP/1.1
Host: www.example.com
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36
            (KHTML, like Gecko) Chrome/27.0.1453.116 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: de-DE, de;q=0.8, en-US;q=0.6, en;q=0.4
Accept-Charset: iso-8859-1, utf-8, utf-16, *;q=0.1
Accept-Encoding: gzip, deflate, sdch
Connection: keep-alive
```

HTTP
Response



Quelle: http://userpages.umbc.edu/~dgorin1/451/OSI7/dcomm/http.htm

Response-Beispiel:

```
HTTP/1.1 200 OK
Date: Wed, 26 Jun 2013 16:36:27 GMT
Server: Apache
Content-Type: text/html; charset=UTF-8
Content-Length: 12313
Last-Modified: Mon, 16 Apr 2013 20:27:06 UTC
Connection: keep-alive

<!DOCTYPE html>
<html>
…
```

Statuscodes

Code	Bedeutung
1xx	**Informational** – Anforderung empfangen, Aktion folgt
2xx	**Success** - Erfolgreiche Abarbeitung der Anfrage
3xx	**Redirection** - Angefragtes Objekt befindet sich an anderer Stelle
4xx	**Client Error** - Falsche / Unerlaubte Anfrage
5xx	**Server Error** - Server kann Anfrage nicht ausführen

- Beispiele:
  - > 200 OK
  - > 301 Moved Permanently
  - > 302 Moved Temporarily
  - > 404 Not Found
  - > 500 Internal Server Error

Details im HTTP/1.1 RFC: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

HTTP
Request-Methoden

Methode	Bedeutung
GET	Anforderung, der im Request-URI angeforderten Ressource
POST	Wird zur Übertragung von Daten an den Server benutzt
HEAD	Anforderung des Headers einer Ressource (wie GET, lässt aber den Seiteninhalt weg)
PUT	Wird zur Übertragung einer neuen Ressource an den Server benutzt.
DELETE	Löschen von Dokumenten auf dem Server. Der Server muss entsprechende Rechte auf dem Server besitzen (Sicherheitsprobleme)
OPTIONS	Diese Methode erlaub, den Zugriffspfad zu einer Ressource zu ermitteln und die Methoden, die darüber möglich sind.
TRACE	Für Testzwecke: Die Anfrage wird vom Server zurück geschickt

- Zunächst werden nur GET und POST genutzt, teilweise HEAD
- Andere Methoden erst durch die REST-Anwendungen populär (neue patch-Methode)

Details im HTTP/1.1 RFC: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

HTTP
GET-Request

Eigenschaften

- Übertragung der Daten über die Adresszeile
- Parameter werden durch das Zeichen ? in der URL eingeleitet
- Kann als Lesezeichen gespeichert werden
- Nicht geeignet zur Übertragung großer Datenmengen
- Daten, die per GET übertragen werden, werden häufig geloggt (z.B. vom Server oder einem Proxy) und möglicherweise auch vom Browser gespeichert (im Cache)
  - > Keine sensiblen Daten per GET übermitteln
  - > Sichtbar in der URL

Beispiel

```
GET /index.php?page=welcome&id=42 HTTP/1.1
Host: www.example.com
...
Connection: keep-alive
```

HTTP
POST-Request

Eigenschaften

- Übertragung der Daten im HTTP-Body (≠ HTML-Body!)
  - > Daten sind nicht in der URL sichtbar
- Geeignet für große Datenmengen
- Zusätzliche Übermittlung von Daten in der URL analog zur GET-Methode möglich
- Daten, die per POST übertragen werden, werden i.d.R. nicht mitgeloggt und auch nicht in einem Cache vorgehalten
  - > Zusätzliche Verschlüsselung ist trotzdem sinnvoll
- Leerzeile trennt HTTP-Header von body (wie bei der Response)

Beispiel

```
POST /edit.php HTTP/1.1
Host: www.example.com
...
Connection: keep-alive

username=thomas&pw=supersecret123
```

Beispiel POST

```
<form action="do.php?q=login" method="post">
  <input type="text" name="username" />
  <input type="password" name="pw" />
  <input type="submit" value="Login" />
</form>
```

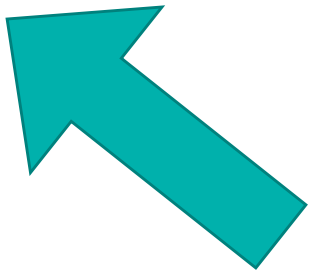
- Erzeugter HTTP-Request (bei entsprechender Eingabe):

```
POST /do.php?q=login HTTP/1.1
Host: www.example.com
...
Content-Type: application/x-www-form-urlencoded
Connection: keep-alive

username=thomas&pw=supersecret123
```

Weitere Informationen: https://developer.mozilla.org/docs/Web/HTML/Element/form

Beispiel GET

```
<form action="do.php?q=login" method="get">
  <input type="text" name="username" />
  <input type="password" name="pw" />
  <input type="submit" value="Login" />
</form>
```



- Erzeugter HTTP-Request:

```
GET /do.php?username=thomas&pw=supersecret123 HTTP/1.1
Host: www.example.com
...
Connection: keep-alive
```

Weitere Informationen: https://developer.mozilla.org/docs/Web/HTML/Element/form

HTML Elemente
Formulare

```
<form action="do.php?q=login" method="post">
    <input type="text" name="username" />
    <input type="password" name="pw" />
    <input type="submit" value="Login" />
</form>
```

- Problem: Wozu würde oben `method="get"` führen?
  - > `do.php?username=…&pw=…`
  - > `q=login` würde wegfallen, da URL überschrieben wird

HTML Elemente
Formulare

- Auswahl an Eingabemöglichkeiten via <input>

<input type="...">	Bedeutung
text	Einfaches (einzeiliges) Eingabfeld
password	Eingabefeld, das die Eingabe verschleiert [*******]
submit	Button zum Übermitteln des Formulars, gute Alternative: <button>
radio	„Schaltknöpfe“ von denen genau ein Feld ausgewählt werden kann
checkbox	Mehrfachauswahl über Kästchen
file	Auswahl einer Datei zum Upload
hidden	Verstecktes Feld, das dem Nutzer nicht angezeigt wird
number	Eingabefeld für eine Zahl
date	Eingabefeld für Datum
datetime	Eingabefeld für Datum und Uhrzeiet
email	Eingabefeld für eine Emailadresse
range	Slider zur Auswahl in einem festgelegten Bereich
url	Eingabefeld für eine URL

Vollständige Liste mit weiteren Live-Beispielen: https://wiki.selfhtml.org/wiki/HTML/Formulare

HTTP 1.0

Ursprüngliches HTTP/1.0 Protokoll von 1996

- RFC 1945: https://tools.ietf.org/html/rfc1945
- Pragmatisch einfaches Protokoll
  - > Einfach zu implementieren
- ASCII-basiert (statt wie z.B. TCP binär)
- Jede einzelne Operation wird über eine separate TCP-Verbindung realisiert
  - > TCP CONNECT => GET index.html => TCP CLOSE
  - > TCP CONNECT => GET image.html => TCP CLOSE
  - > ...
  - > Je eingebundener Resource ist ein weiterer TCP Handshake notwendig
  - > Großen Aufwand beim sukzessiven Laden mehrerer Resourcen, insbesondere auch die den damit verbundenen TCP-SlowStart

HTTP 1.1

HTTP/1.1 wurde 1999 standardisiert

- Ursprünglicher RFC 2616: https://tools.ietf.org/html/rfc2616
  - > Formulierungen in RFCs 7230-7235 verbessert

- Persistente Verbindungen
  - > Der Client kann nun mittels „Connection: keep-alive“ angeben, dass er gerne die TCP/IP-Verbindung offen halten möchte
  - > Kompliziertere Implementierung, aber deutliche Verbesserung beim Laden

- Virtual Hosts
  - > Auf einem Rechner sollen verschiedene Domains und Dienste zur Verfügung stehen
  - > Vorherige Lösungen (HTTP/0.9) sahen verschiedene Serverports pro Dienst oder auch mehrere IP-Adressen für einen Rechner vor
  - > Non-IP-based Virtual Hosts lösten dieses Problem entgültig
    - – Im HTTP Request muss der Header *Host* enthalten sein
    - – Beliebig viele Domains können sich dann die gleiche IP-Adresse teilen

HTTP
Binärdaten und das textbasiert

HTTP ist textbasiert (eMail auch!)

Wie können binäre Daten übertragen werden?

- Die Antworten des Servers auf eine vollständige GET-Request beinhaltet MIME-Informationen
- MIME = Multipurpose Internet Mail Extensions
- Definiert die Kodierungsregeln für Nicht-ASCII-Nachrichten
- MIME ermöglicht die Nutzung verschiedener Kodierungen (media types) in einer Nachricht

Die “Content-Type:”-Zeile im MIME-Header legt den Datentyp (type/subtype) einer Nachricht fest

Content-Transfer-Encoding: definiert die Transfersyntax, in der die Daten des Hauptteils übertragen werden, wird aber bei HTTP nicht benutzt

- Content-Encoding und Transfer-Encoding Felder

Beispiele:

- Content-Type: text/html
- Content-Type: image/GIF

HTTP
Binärdaten und das textbasiert

```
MIME-Version: 1.0
Content-Type: MULTIPART/MIXED;
  BOUNDARY= "8323328-2120168431-824156555=:325"
--8323328-2120168431-824156555=:325
Content-Type: TEXT/PLAIN; charset=US-ASCII
A picture is in the appendix
--8323328-2120168431-824156555=:325
Content-Type: IMAGE/JPEG; name="picture.jpg"
Content-Transfer-Encoding: BASE64
Content-ID: <PINE.LNX.3.91.960212212235.325B@localhost>
Content-Description:
/9j/4AAQSkZJRgABAQEAlgCWAAD/2wBDAAEBAQEBAQEBAQEBAQEBAQIBAQEBA
  QIBAQECAgICAgICAgIDAwQDAwMDAwICAwQDAwQEBAQEAgMFBQQEBQQEBAT/
  2wBDAQEBAQEBAQIBAQIEAwIDBAQEBA […]
KKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAoooo
  AKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiig
  AooooAD//Z
---8323328-2120168431-824156555=:325 –
```

HTTP 2.0

HTTP/2 wurde 2015 veröffentlicht

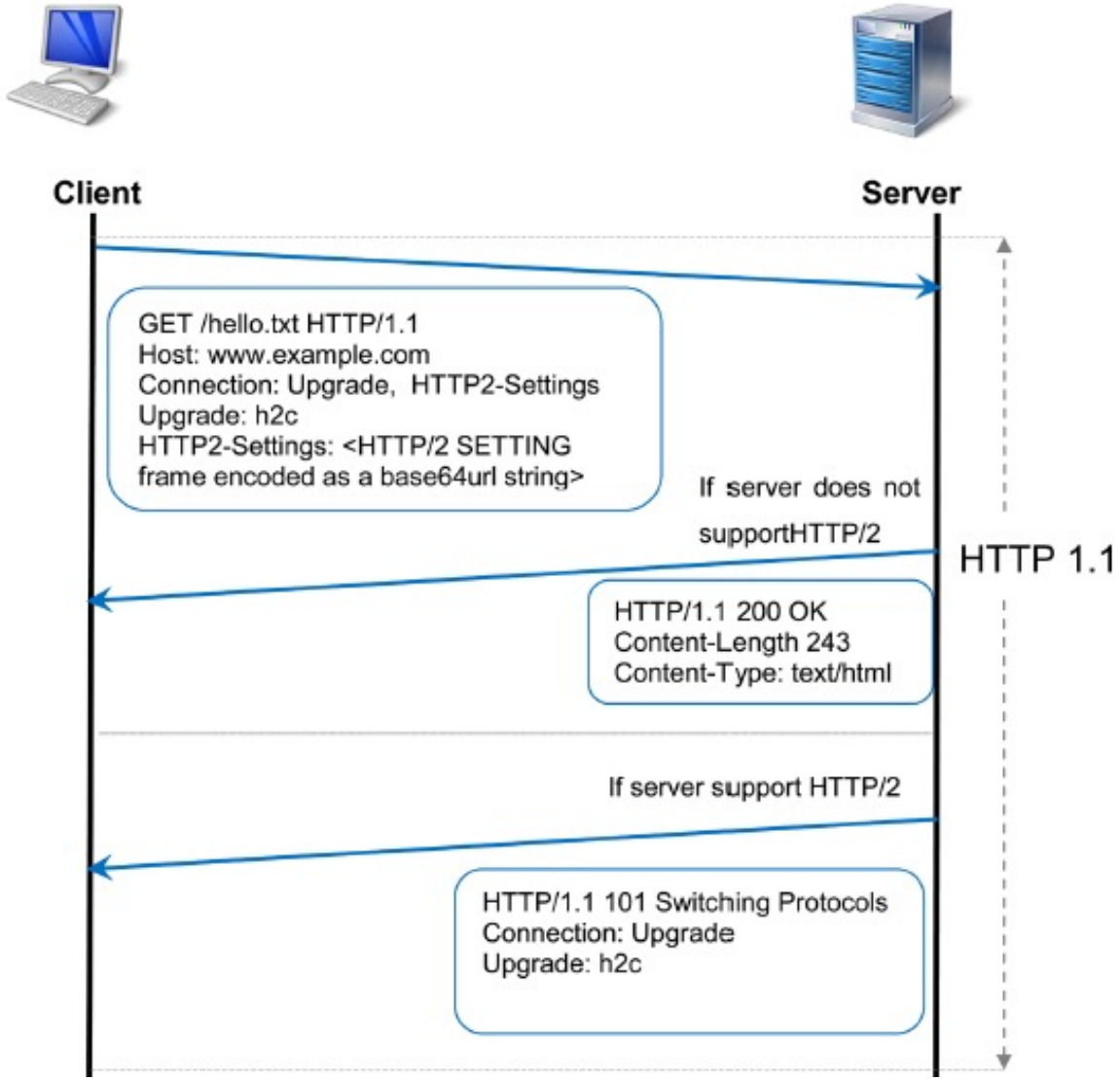
- RFC 7540 (und 7541): https://tools.ietf.org/html/rfc7540

- Weitere Beschleunigung der Datenübertragung (Streaming/Framing zur Erhöhung der Parallelität durch Multiplexing: Kein Head-of-Line-Blocking mehr)
- Abwärtskompatibel zu HTTP/1.1 (das übliche URL-Schema bleibt erhalten); Upgrade der Verbindung
- Trotzdem ist die Übertragung der Inhalte binär

- Wichtigste Neuerungen
  - > Mehrere Anfragen in einer Übertragung zusammenfassen
  - > Header-Compression um die zunehmend langen geschwätzigen Header effektiver zu übertragen
  - > PUSH-Verfahren, also Datenübertragungen, die vom Server initiiert werden können; Einsparen weiterer Ladevorgänge

HTTP 2.0

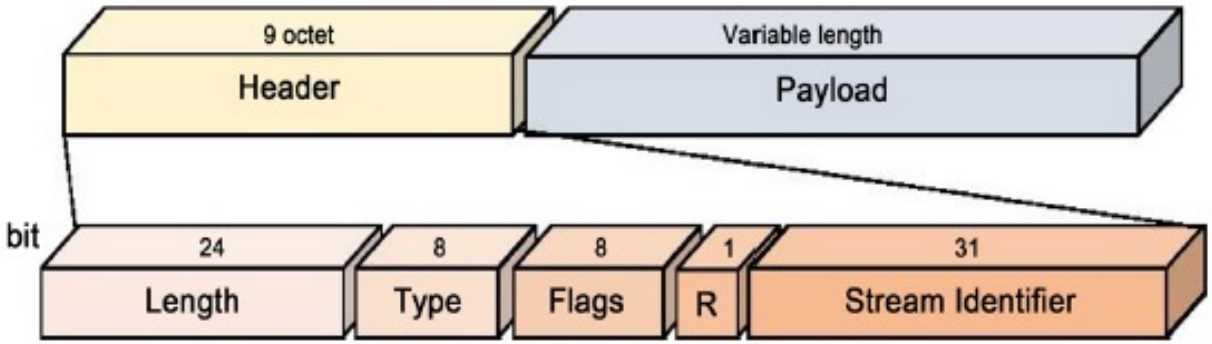
Biljana Dimitrova, Aleksandra Mileva
Journal of Computer and Communications, 2017, 5, 98-111



Ab hier HTTP/2

HTTP 2.0

Biljana Dimitrova, Aleksandra Mileva
Journal of Computer and Communications, 2017, 5, 98-111



HTTP 2.0

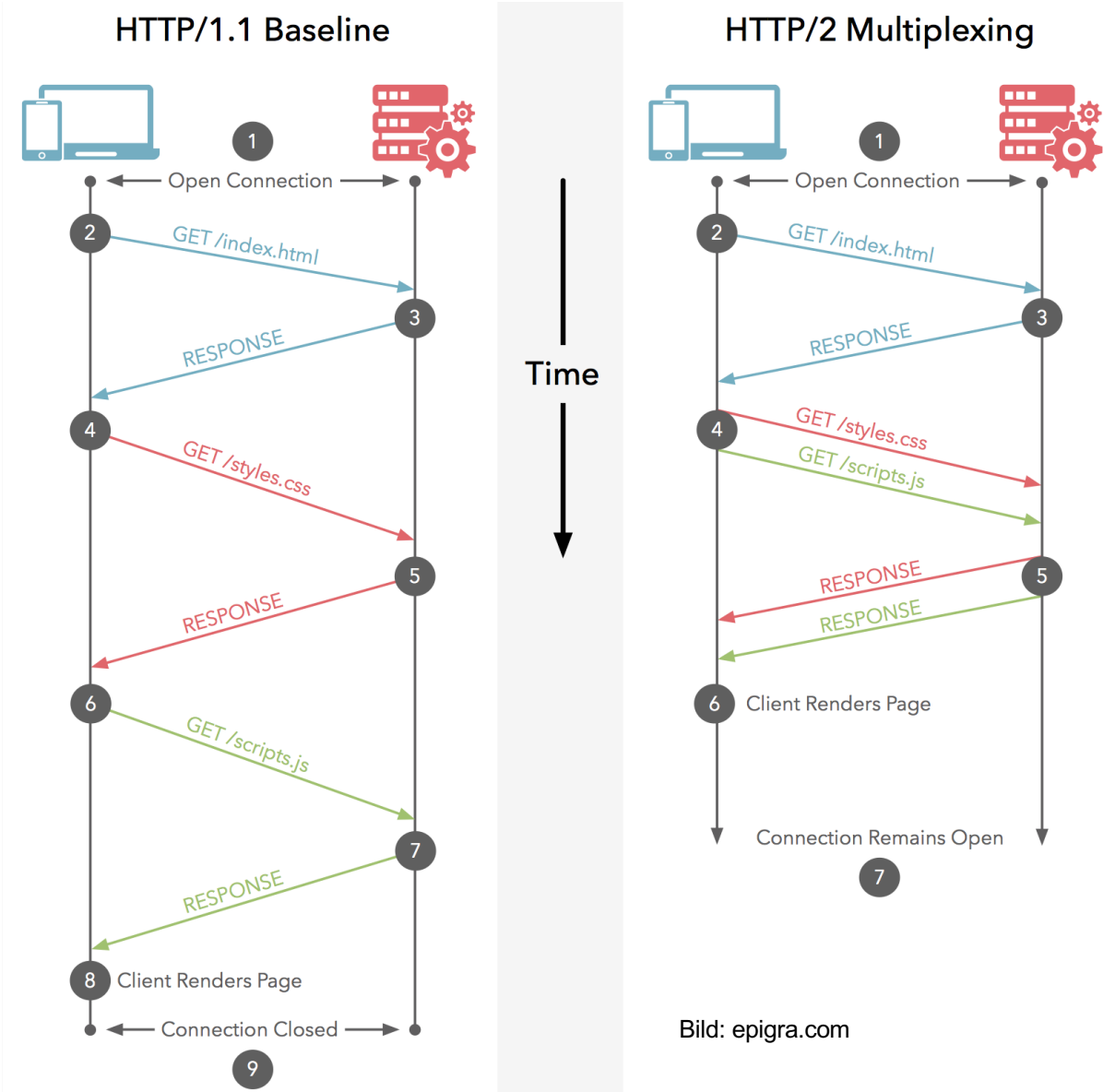
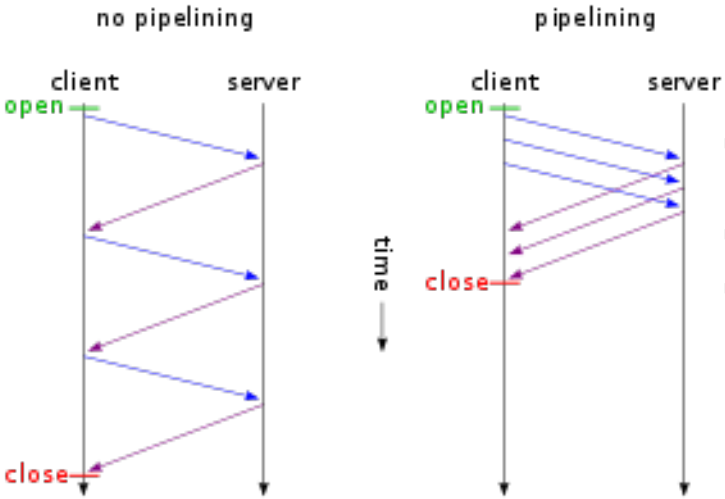


Bild: epigra.com

HTTP
Pipelining vs. Multiplexing
https://en.m.wikipedia.org/wiki/HTTP_pipelining



HTTP1.1 erlaubt Pipelining, aber…

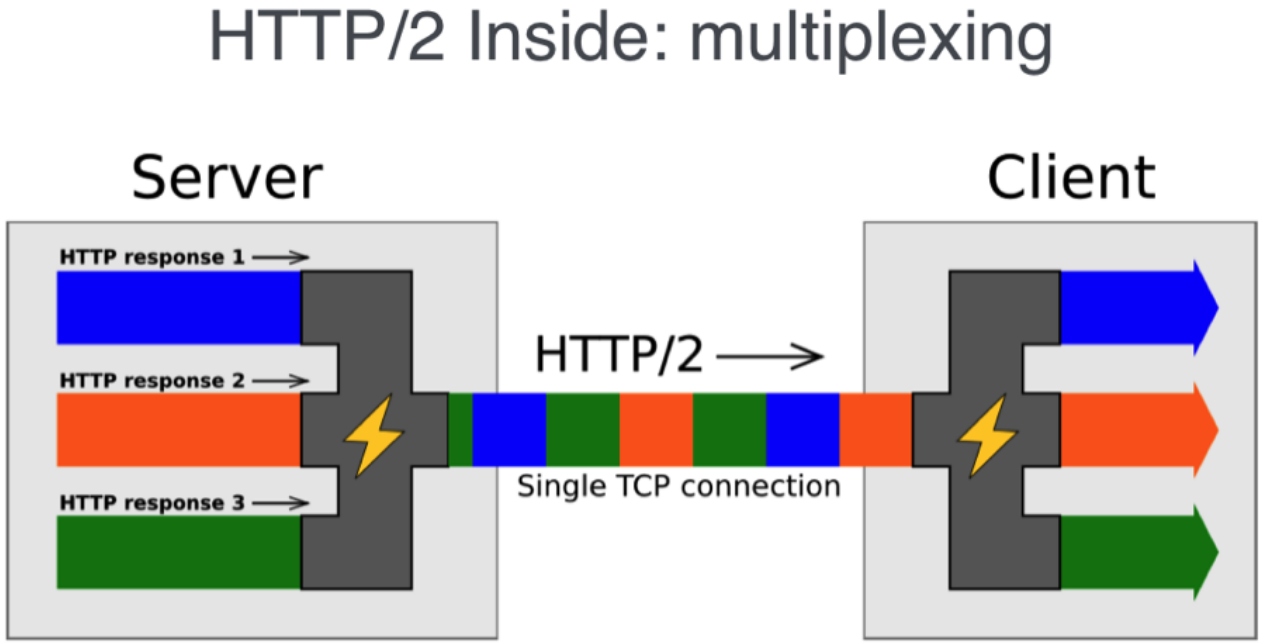
- Implementation in web browsers
- Of all the major browsers, only Opera based on Presto layout engine had a fully working implementation that was enabled by default. In all other browsers HTTP pipelining is disabled or not implemented.[5]
- Internet Explorer 8 does not pipeline requests, due to concerns regarding buggy proxies and head-of-line blocking.[9]
- Internet Explorer 11 does not support pipelining. [10]
- Mozilla browsers (such as Mozilla Firefox, SeaMonkey and Camino) support pipelining; however, it is disabled by default.[11][12] Pipelining is disabled by default to avoid issues with misbehaving servers.[13] When pipelining is enabled, Mozilla browsers use some heuristics, especially to turn pipelining off for older IIS servers.[14] Support for H1 Pipeline was removed from Mozilla Firefox in Version 54.[15]
- Konqueror 2.0 supports pipelining, but it's disabled by default.[citation needed]
- Google Chrome previously supported pipelining, but it has been disabled due to bugs and problems with poorly behaving servers.[16]
- Pale Moon (web browser) supports pipelining, and is enabled by default[17]

Pipelining vs. Multiplexing

https://en.m.wikipedia.org/wiki/HTTP_pipelining

Der Unterschied

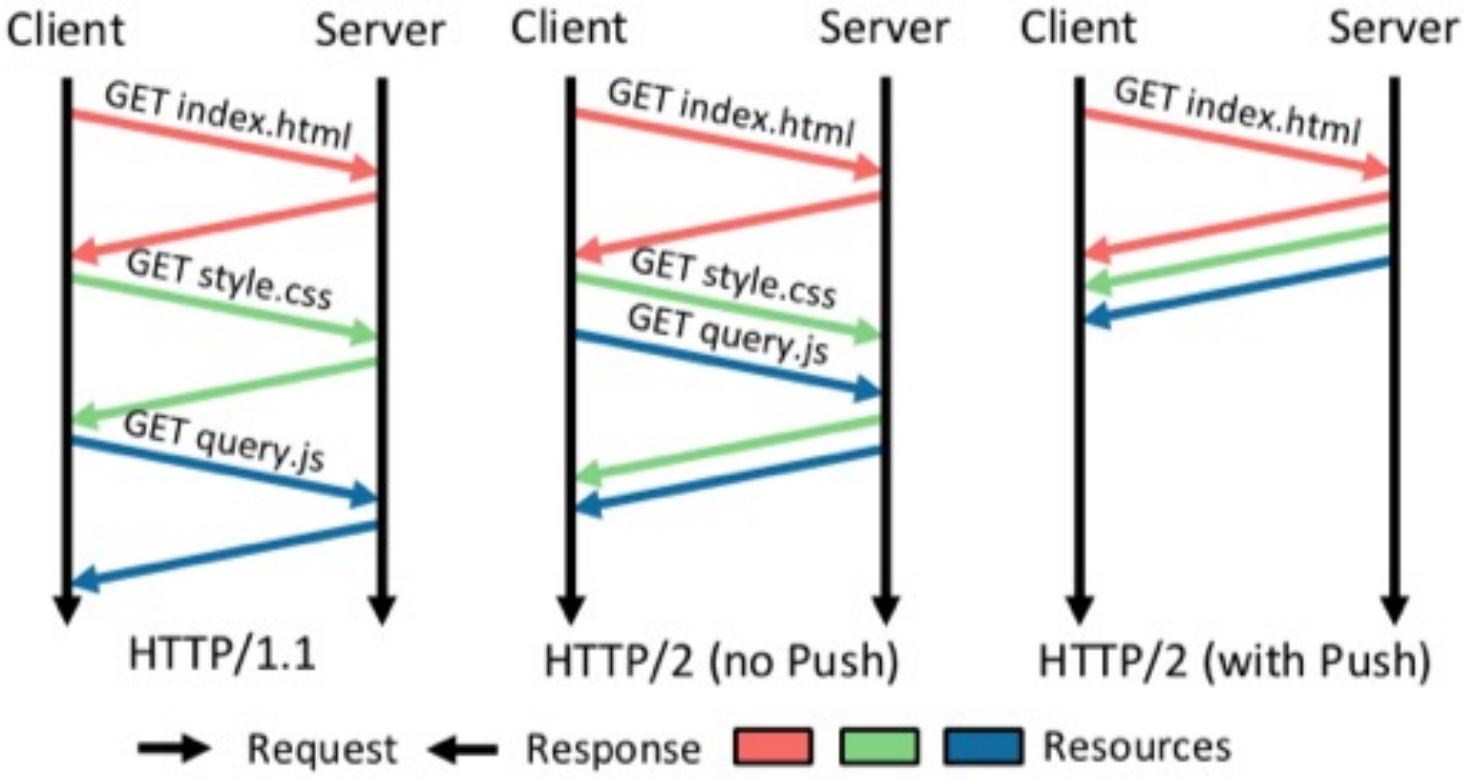
- Beim Pipelining können die Folgeanfragen gestartet werden, bevor eine Antwort erhalten wurde
- Die Antworten kommen aber in der gleichen Reihenfolge, wie die Anfragen gestartet wurden
- Beim Multiplexing können jedoch die Antworten in beliebiger Reihenfolge kommen. Letztlich werden Streams in Chunks aufgeteilt und via Multiplexing verteilt



Quelle: NGINX

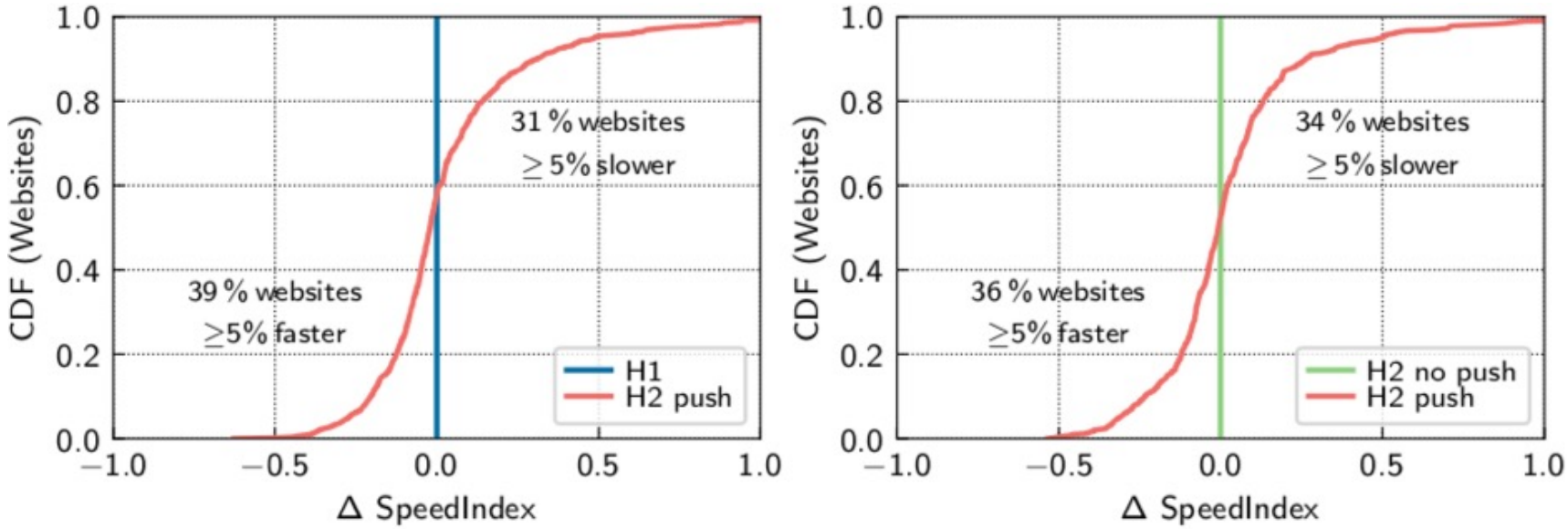
HTTP 2.0

https://blog.apnic.net/2018/04/26/adoption-performance-and-human-perception-of-http-2-server-push/



HTTP 2.0

https://blog.apnic.net/2018/04/26/adoption-performance-and-human-perception-of-http-2-server-push/



HTTP 3.0

- Noch nicht offiziell standardisiert
- Einige Spezifikation bereits in einem Internet Draft(vorläufiges Dokument der IETF)
- Hauptänderung: Umstieg von TCP auf Quic

Quic:

- basiert auf UDP
- beseitigt ineffiziente Elemente von TCP
- beinhaltet eigene Fehlerkorrektur
- übernimmt und verbessert das Multiplexing aus HTTP/2
- integriert beim Verbindungsaufbau effizient TLS-Konzepte