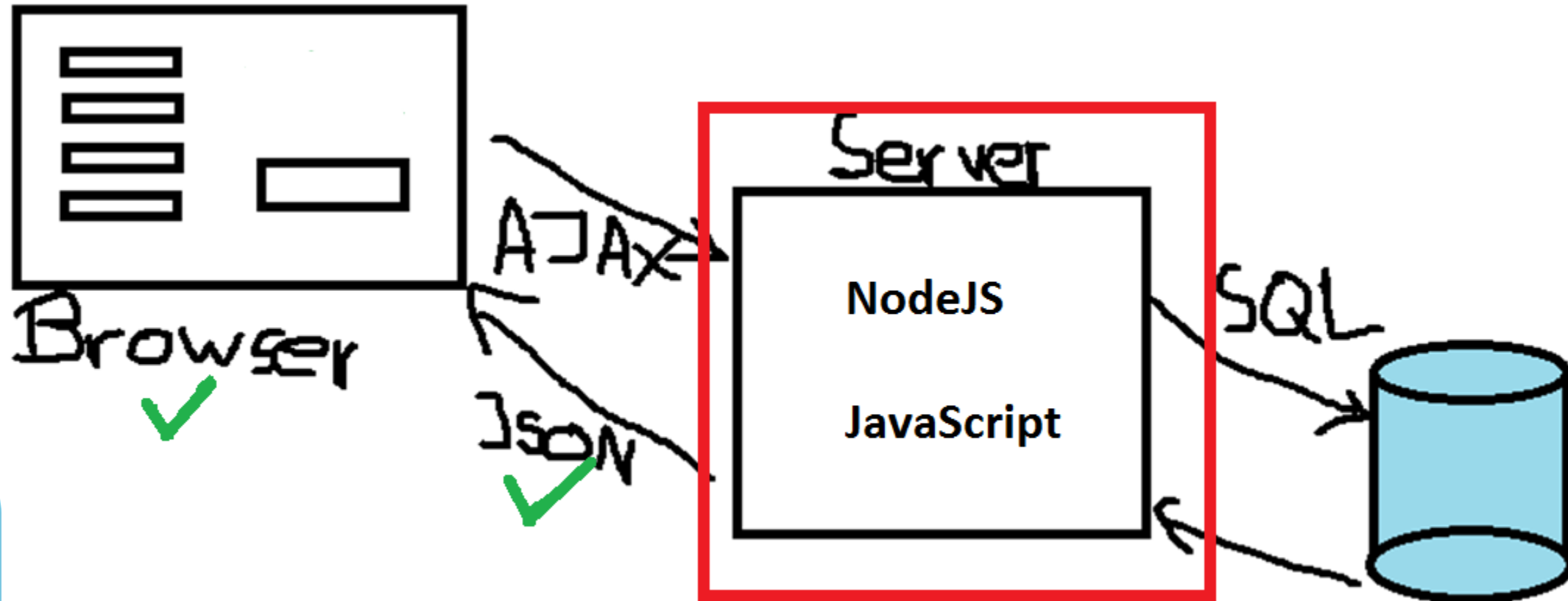


Webanwendungen

Vorlesung - Hochschule Mannheim

Webserver und HTTP

Überblick

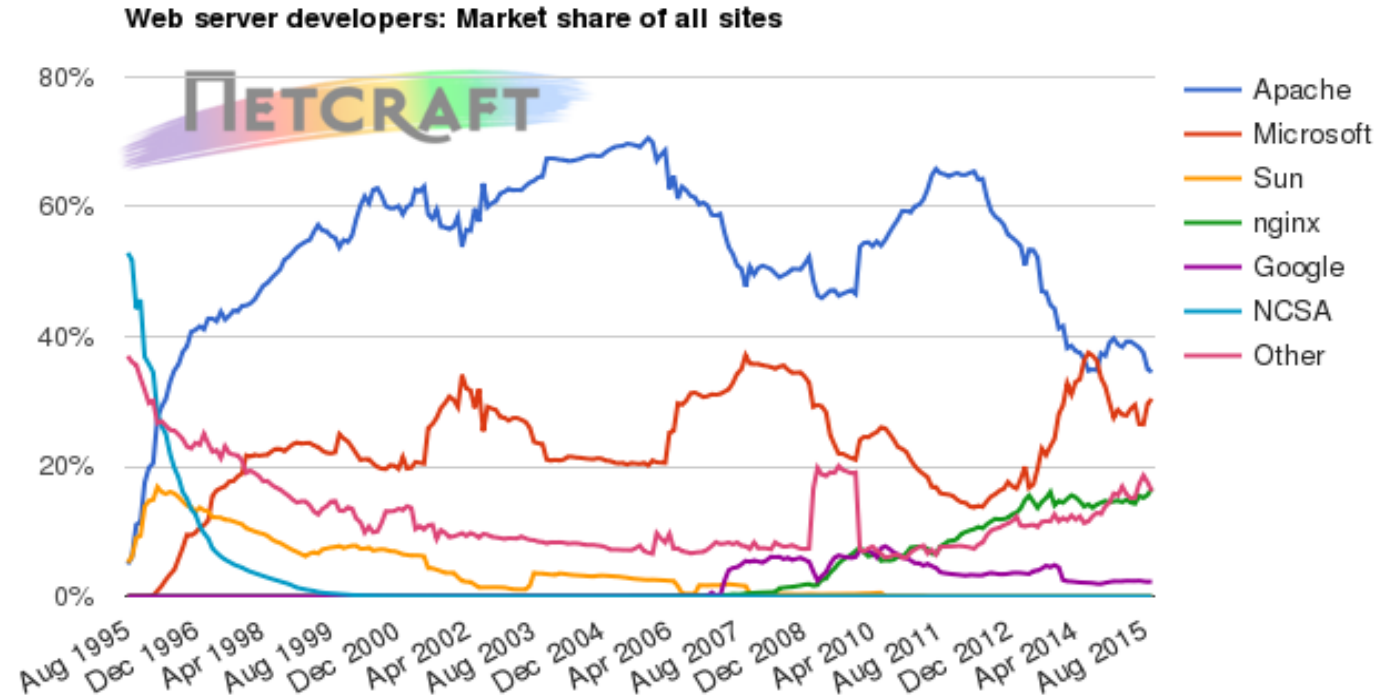


Inhaltsverzeichnis

- ▶ Webserver
- ▶ Sicherheit und Zugriffsbeschränkung
- ▶ HyperText Transfer Protocol (HTTP)
- ▶ Mehrschichten Architekturen
- ▶ Common Gateway Interface (CGI)

Webserver

Webserver (05/2014)



Developer	September 2015	Percent	October 2015	Percent	Change
Apache	312,106,638	34.96%	303,234,897	34.53%	-0.43
Microsoft	265,010,746	29.68%	267,012,322	30.40%	0.72
nginx	139,297,804	15.60%	146,229,307	16.65%	1.05
Google	19,683,087	2.20%	19,931,862	2.27%	0.06

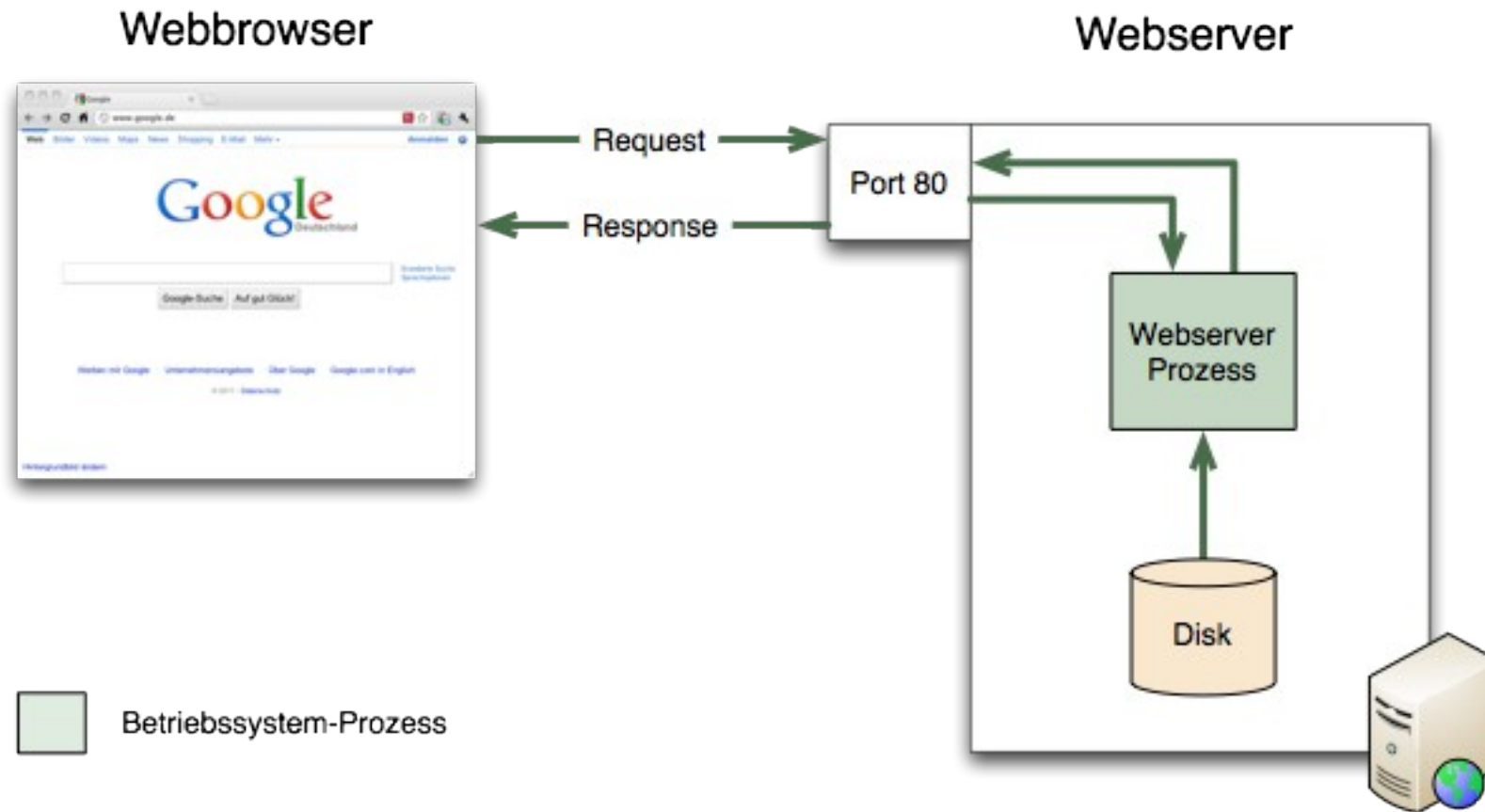
Webserver (Auswahl)

- ▶ Apache (1995) (<http://httpd.apache.org>)
 - ▶ defacto-Standard 50% Marktanteil)
 - ▶ OpenSource-Produkt und Freeware
 - ▶ für UNIX-Plattformen und für MS Windows/DOS verfügbar.
 - ▶ Microsoft's **Internet Information Server (1994)(IIS)**
 - ▶ Kommerzieller Webserver für Windows
 - ▶ Teil von Windows Server
 - ▶ **Google Web Server (GWS)**
- ▶ Google betreibt damit ca. 10 Millionen eigene Websites
- ▶ nicht allgemein verfügbar
- ▶ Auf Linux basierend

Webserver (Auswahl)

- ▶ Nginx (2004) (<http://nginx.org/>)
- ▶ freier Webserver unter BSD-Lizenz
- ▶ kleiner und schlanker Webserver
- ▶ Lighttpd (2007) (<http://www.lighttpd.net/>)
- ▶ freier Webserver unter BSD-Lizenz
- ▶ optimiert für Massendaten
- ▶ eingesetzt z. B. bei YouTube oder SourceForge
- ▶ Node.js (2009) (<http://nodejs.org/>)
- ▶ JavaScript Framework zur implementierung von Webservern
- ▶ basiert auf der JavaScript-Laufzeitumgebung „V8“ (Chrome)
- ▶ optimiert für große Anzahl gleichzeitig bestehender Netzwerkverbindungen

Webserver - Grobe Architektur



Allgemeines zur Konfiguration

- ▶ Konfiguration von Webservern
 - ▶ Konfiguration ist abhängig vom eingesetzten Server
 - ▶ Webserver läuft als
 - ▶ Anwendung - gut für Testzwecke
 - ▶ Daemon (Dienst) - gut für den Betrieb einer öffentlichen Seite
 - ▶ Für Testzwecke auch ohne Internetzugang zu verwenden
 - ▶ Viele Webserver unterstützen *Virtual Hosts*
 - ▶ mehrere Websites auf einem Server realisiert
 - ▶ Unterscheidung zwischen Sites über den Host-Header

Grundeinstellungen

- ▶ Bei allen Servern müssen eingestellt werden
 - ▶ **IP-Adresse** und **Hostnamen** des Servers
 - ▶ für lokalen Betrieb IP-Adresse 127.0.0.1 und Namen localhost
 - ▶ ansonsten eine öffentliche IP-Adresse (keine privaten Netze)
 - ▶ **Port** des Servers
 - ▶ normalerweise Port 80
 - ▶ für Testzwecke häufig 8080 günstiger wegen privilegierter Ports
 - ▶ **HTTP-Wurzelverzeichnis** für HTML-Dateien
 - ▶ Verzeichnis, unterhalb dessen sich die lokalen HTML-Dateien befinden
 - ▶ **Default-Datei** für Verzeichnisanfragen
 - ▶ z. B. index.html oder index.htm

Grundeinstellungen

- ▶ Log-Dateien
 - ▶ Protokollierung der Zugriffe, z.B. `access.log`
 - ▶ Protokollierung von Fehlern, z.B. `error.log`
 - ▶ **Timeouts** für das Senden und Empfangen
 - ▶ Angaben erfolgen in der Regel in Sekunden
- ▶ **MIME-Typen**
 - ▶ Dateiformate, die der Webserver kennt und an den aufrufenden Web-Browser überträgt
 - ▶ Andere Dateitypen sendet der Server nicht korrekt bzw. mit dem eingestellten Standard-MIME-Typ (`text/plain`)

Apache

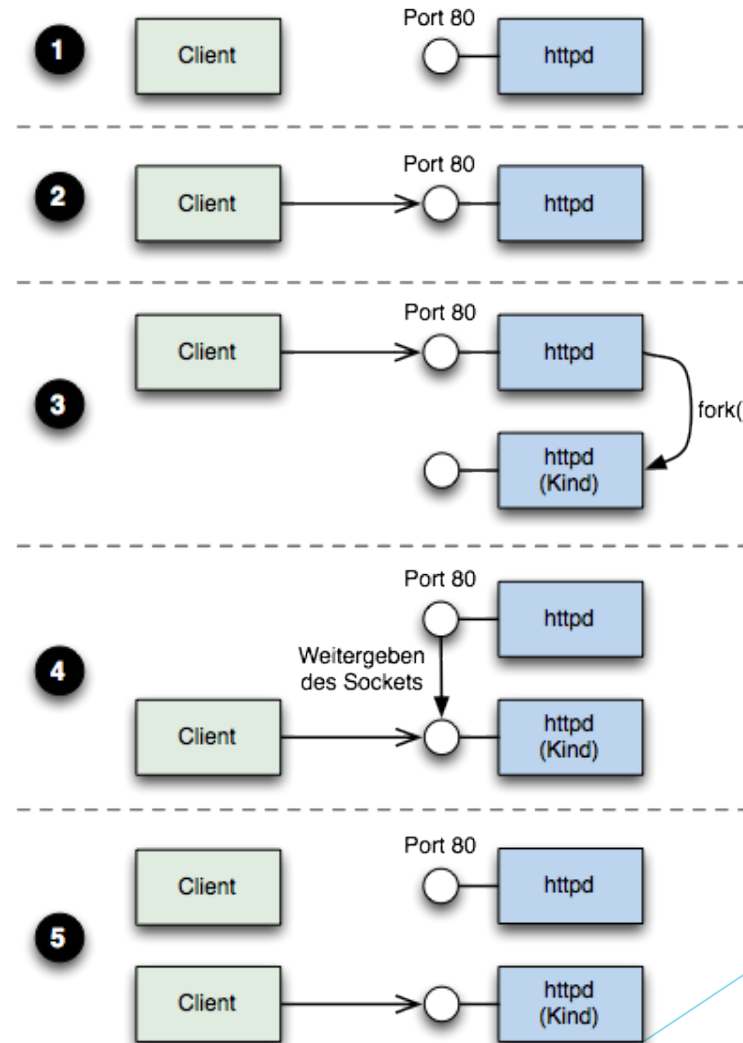
- ▶ Apache ist der am häufigsten eingesetzte Server
 - ▶ Name stammt von „a patchy NCSA Server“
 - ▶ Open-Source (<http://httpd.apache.org/>)
 - ▶ Seit 1995 verfügbar
 - ▶ Aktuelle Version 2.4.9
 - ▶ Teil vieler Linux-Distributionen und von Mac OS X
 - ▶ Kann auch selber kompiliert werden



Apache 1.x Architektur

► Multi-Prozess (pre-fork)

1. ein Prozess öffnet Port 80 und wartet auf Anfragen
2. er nimmt Anfragen entgegen
3. sofort danach wird ein Kindprozess mit `fork()` erzeugt
4. die Client-Verbindung wird an den Kindprozess übergeben
5. der ursprüngliche Prozess kann wieder Verbindungen entgegennehmen



Direktiven für pre-fork

- ▶ *ServerLimit* - Maximale Anzahl von Prozessen, die konfiguriert werden können
- ▶ *StartServer* - Anzahl der Server-Prozesse beim Start
- ▶ *MinSpareServer* - minimale Anzahl von unbeschäftigten Kind-Prozessen
- ▶ *MaxSpareServer* - maximale Anzahl von unbeschäftigten Kind-Prozessen
- ▶ *MaxClients* - maximale Anzahl von Clients, die parallel bedient werden können (== maximale Anzahl von Prozessen)
- ▶ *MaxRequestsPerChild* - maximale Anzahl von Requests, die ein Prozess beantworten darf, bevor er beendet wird

Apache 2.x Architektur

- ▶ Apache 2.x unterstützt neben pre-fork zusätzlich multi-threading (*worker*-Modell) innerhalb eines Servers
 - ▶ innerhalb eines Prozesses können mehrere Threads gestartet werden
 - ▶ jeder Thread kann einen Client bedienen
 - ▶ die Prozesse werden wie bei pre-fork behandelt, können jetzt aber parallel mehrere Anfragen bearbeiten
 - ▶ Modell reduziert drastisch den Speicherverbrauch bei vielen Clients

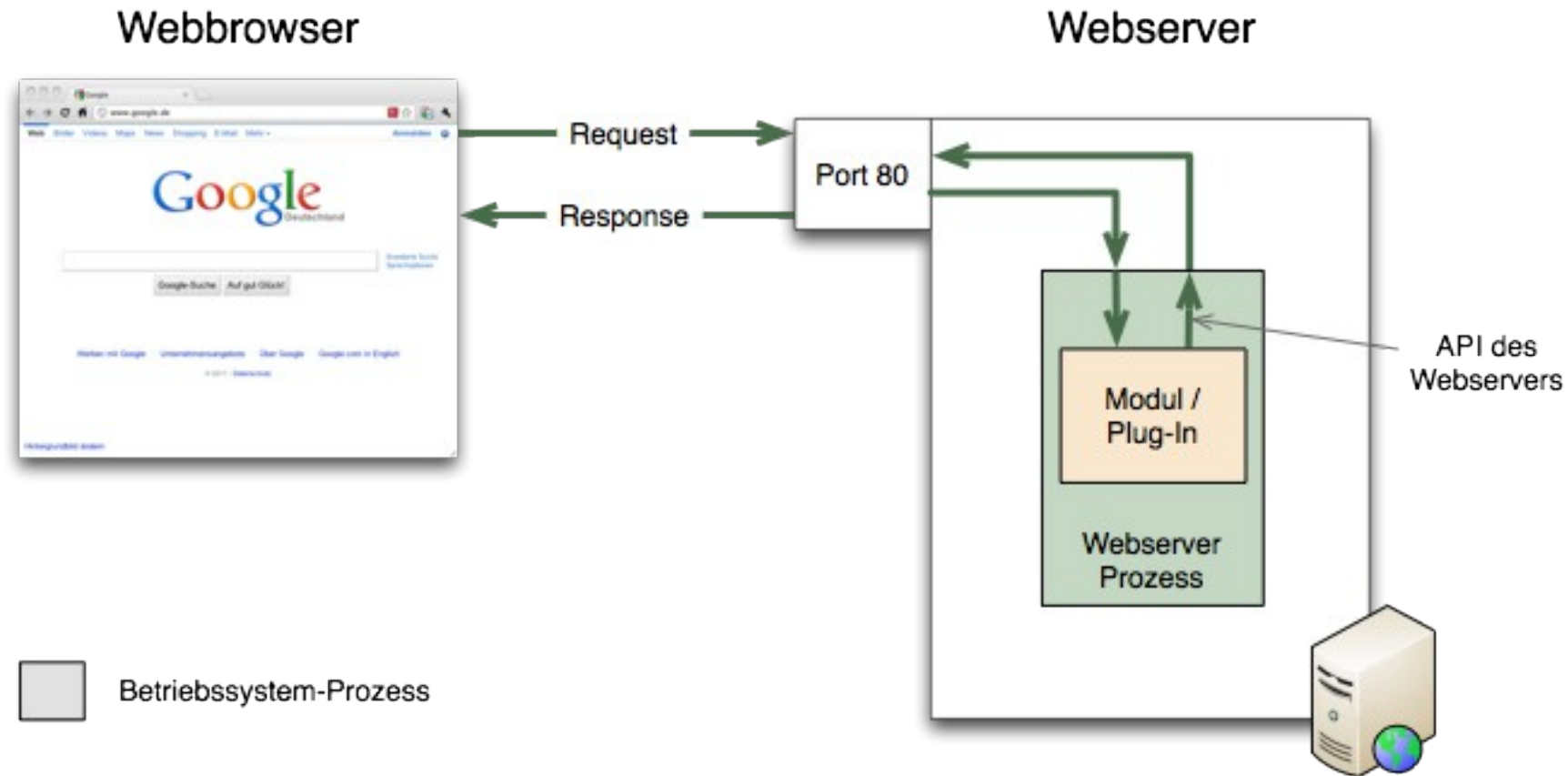
Direktiven für worker

- ▶ *ServerLimit* - Maximale Anzahl von Prozessen, die konfiguriert werden können (Begrenzt $MaxClients * ThreadsPerChild$)
- ▶ *MinSpareThreads* - minimale Anzahl von unbeschäftigten Threads (über alle Prozesse hinweg)
- ▶ *MaxSpareThreads* - maximale Anzahl von unbeschäftigten Threads (über alle Prozesse hinweg)
- ▶ *ThreadsPerChild* - Anzahl von Threads pro Prozess
- ▶ *MaxClients* - maximale Anzahl von Clients, die parallel bedient werden können (Prozesse = $MaxClients / ThreadsPerChild$)
- ▶ *MaxRequestsPerChild* - maximale Anzahl von Requests, die ein Prozess beantworten darf, bevor er beendet wird

Module

- ▶ Apache kann durch Module erweitert werden, z. B.
 - ▶ `mod_cgi` - beliebige CGI-Skripte ausführen
 - ▶ `mod_perl` - Perl-Skripte ausführen
 - ▶ `mod_php` - PHP-Skripte ausführen
 - ▶ `mod_python` - Python-Skripte ausführen
 - ▶ `mod_rewrite` - URLs rewrites
 - ▶ `mod_jk` - Verbindung zu einem Java-Server herstellen
 - ▶ `mod_proxy` - Als Proxy fungieren

Module



MIME-Types

- ▶ Server ermittelt MIME-Type aus Datei-Endung
 - ▶ Zuordnung gängiger Typen in `mime.types`
`TypesConfig conf/mime.types`
 - ▶ Standardvorgabe, falls kein MIME-Type ermittelt werden kann
`DefaultType text/plain`

Anforderungen an gute Web-Server

- ▶ **Viele** gleichzeitig eintreffende Requests **schnell** und **zufriedenstellend** verarbeiten.
- ▶ Viel: Das C10k Problem
- ▶ Schnell: „Echtzeit“
- ▶ Zufriedenstellend: Nach Verarbeitung ein Ergebnis

Echtzeit

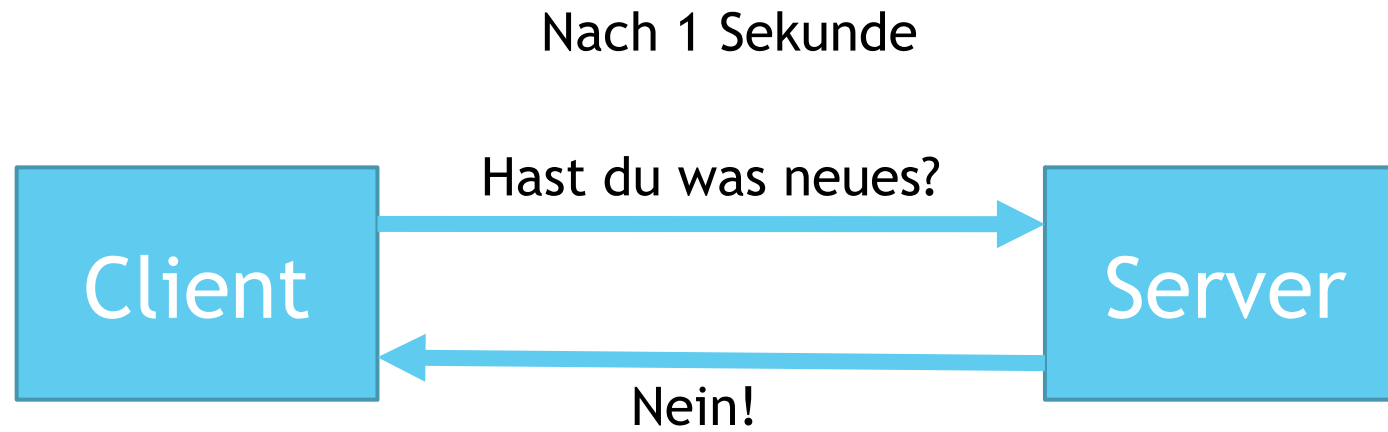
▶ Das sagt Wiki:

„Unter Echtzeit versteht man den Betrieb eines Rechensystems, bei dem Programme zur Verarbeitung anfallender Daten ständig betriebsbereit sind, derart, dass die Verarbeitungsergebnisse innerhalb einer vorgegebenen Zeitspanne verfügbar sind. Die Daten können je nach Anwendungsfall nach einer zeitlich zufälligen Verteilung oder zu vorherbestimmten Zeitpunkten anfallen“

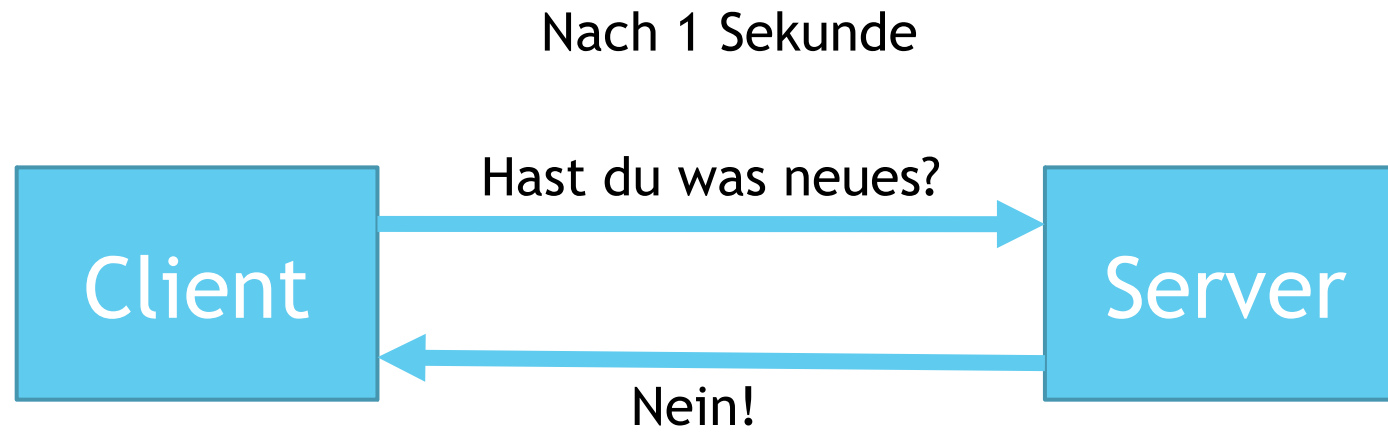
- ▶ Web-Seiten und Applikationen sollen sich wie Desktop-Anwendungen anfühlen
- ▶ Keine Verzögerung bei Client-Server Traffic

Umsetzung?

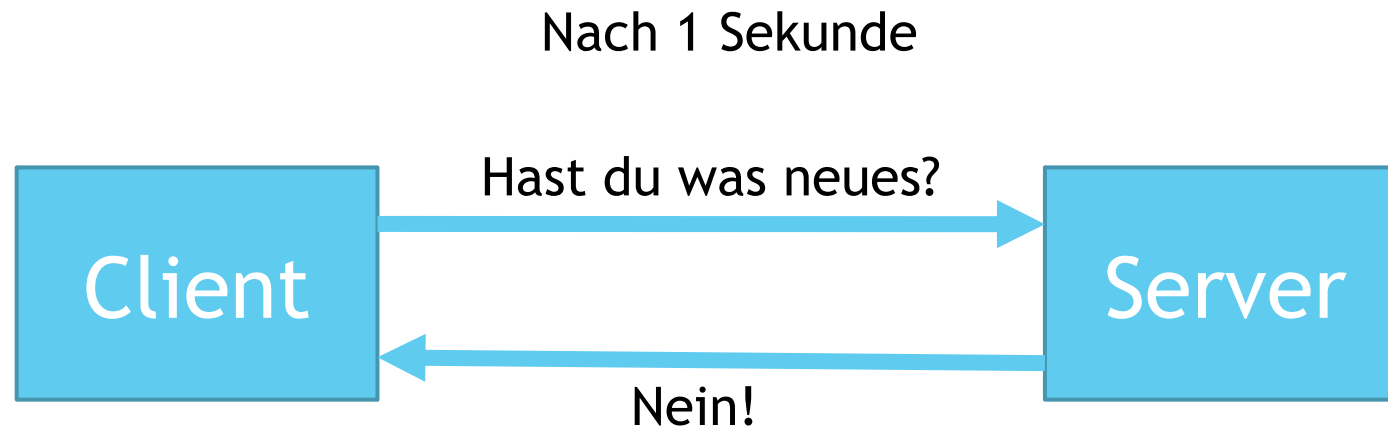
Polling(2000)



Polling(2000)



Polling(2000)



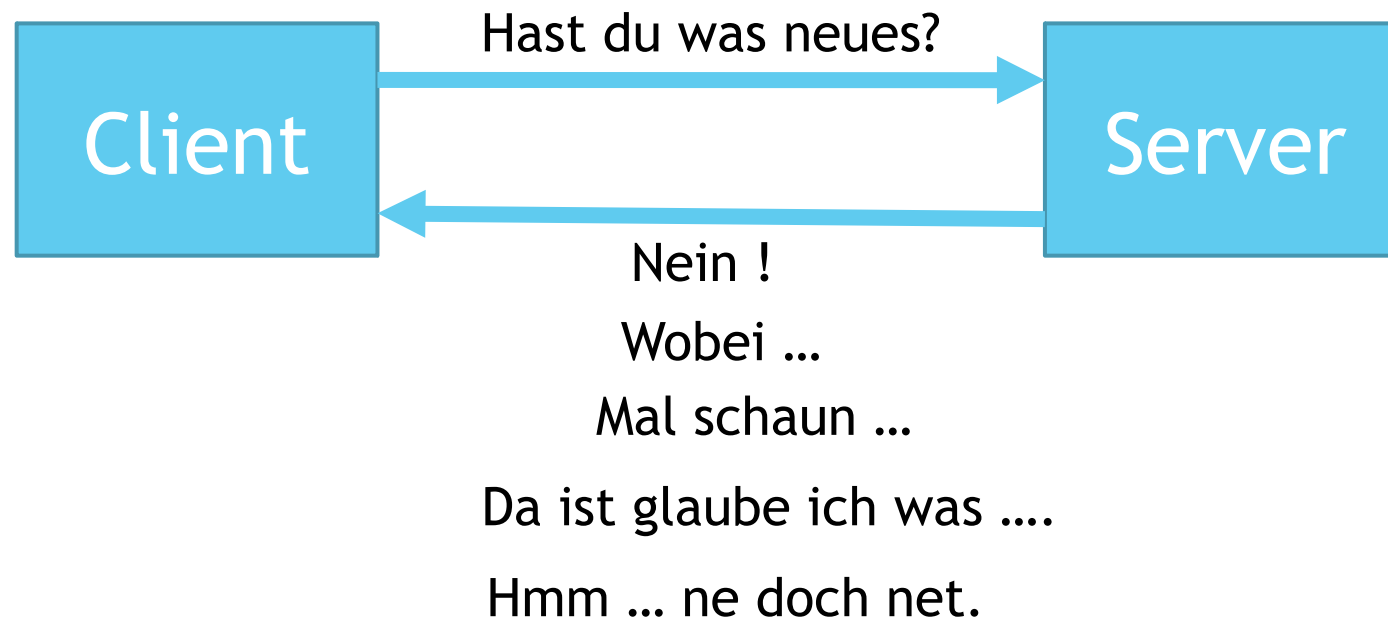
Polling(2000)

Client

Server



Comet / Long-Polling (2006)

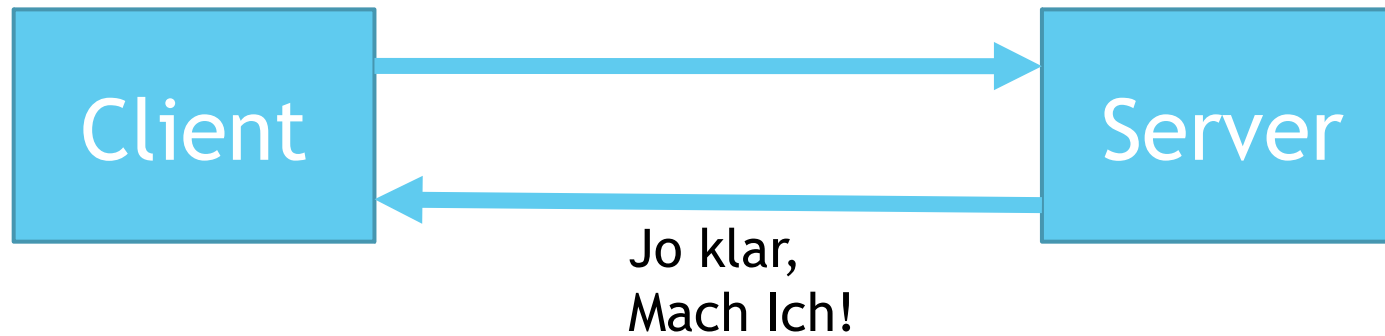


Comet / Long-Polling (2006)

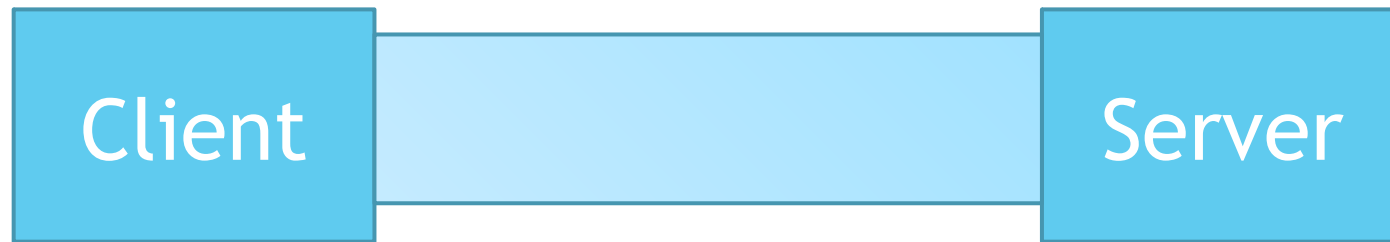


WebSockets(2011)

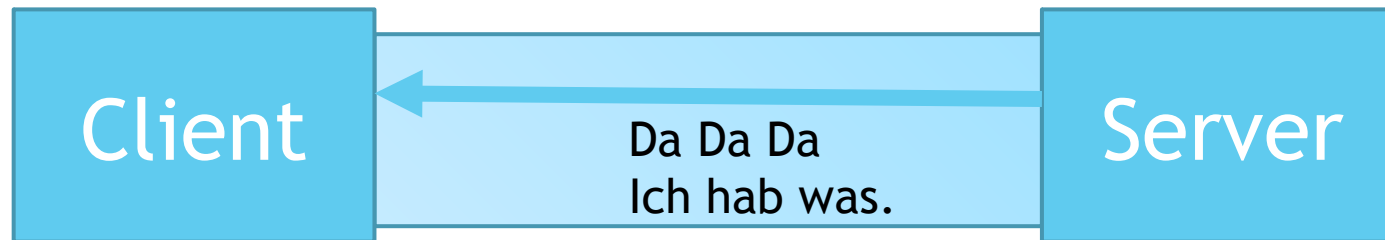
Ey Du gel, wenn du was neues
hast sagste bitte Bescheid.
Können wir dafür auch bitte ,nen anderes Protokoll benutzen?



WebSockets(2011)



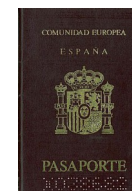
WebSockets(2011)



Sicherheit

Authentifizierung

- ▶ **Authentifizierung** (*authentication*) - Überprüfung, mit wem man es zu tun hat (des Benutzers)
- ▶ Basiert auf
 - ▶ Etwas was man *weiß*
(z. B. Parole, Passwort)
 - ▶ Etwas was man *hat*
(z. B. Schlüssel, Chipkarte, Private Key)
 - ▶ Etwas was man *ist*
(z. B. Foto, Fingerabdruck, Netzhaut-Scan)
 - ▶ Oder Kombinationen davon
(z. B. Reisepass, Secure-ID-Karte)



32



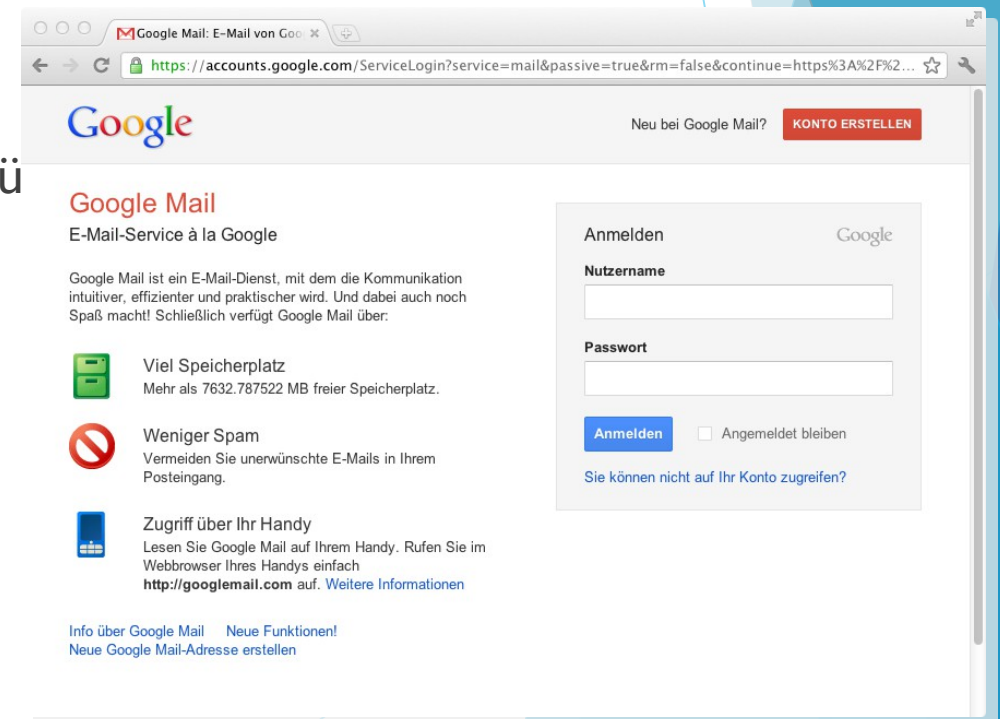
Basic Authentication

- ▶ Teil der HTTP-Spezifikation
- ▶ *Credentials* (Benutzername und Passwort) werden im HTTP-Header gesendet (Base64-codiert)
- ▶ Verwendet ein Browser-Popup (bei jedem Browser anders)
- ▶ SSL/TLS unbedingt erforderlich, um Daten zu schützen
- ▶ Credentials werden im Browser gespeichert
- ▶ Explizites Log-Out nicht möglich



Form-based Authentication

- ▶ Basiert auf einem HTML-Formular
- ▶ *Credentials* (Benutzername und Passwort) werden als HTTP POST-Parameter übertragen
- ▶ Credentials werden im Klartext übertragen
- ▶ SSL/TLS unbedingt erforderlich, um Daten zu schützen
- ▶ Browser speichert Credentials nicht automatisch
- ▶ Explizites Log-Out ist möglich

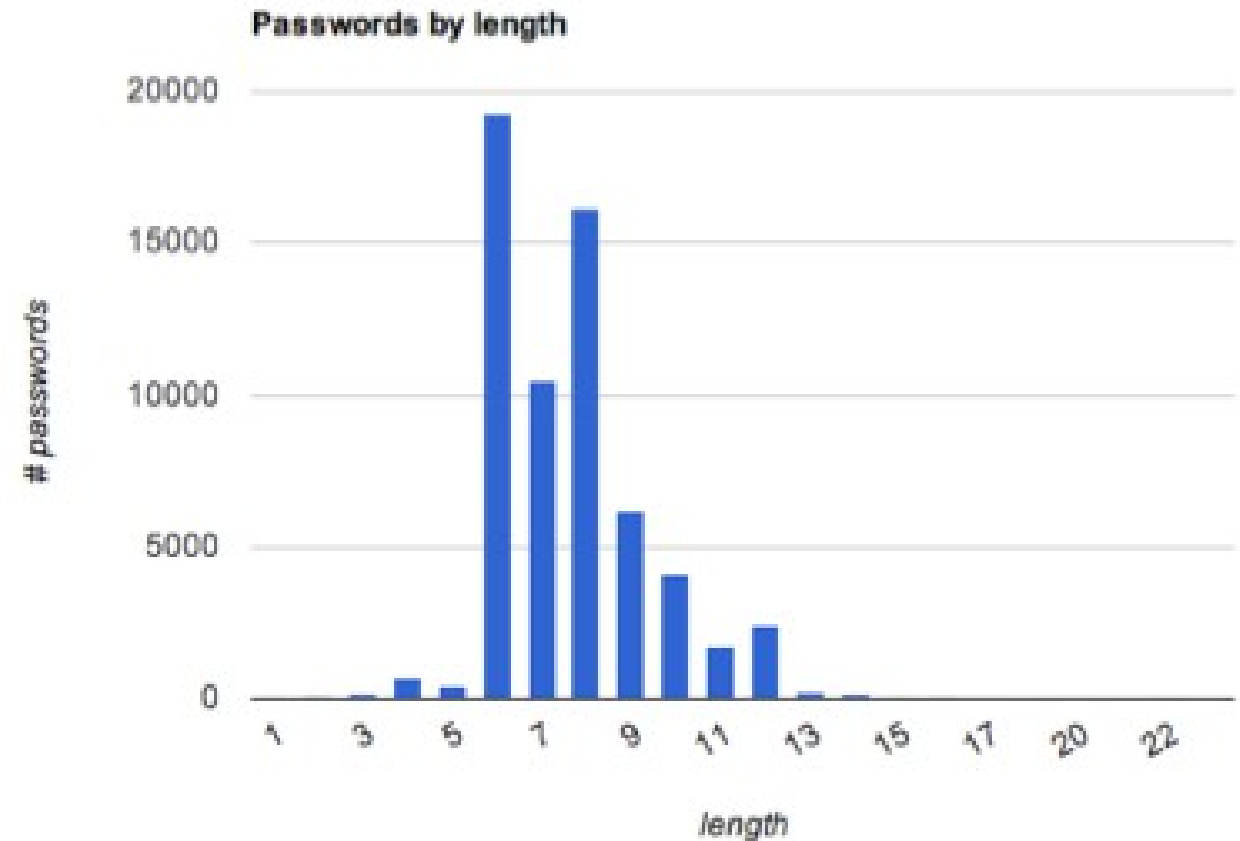


The screenshot shows a web browser window displaying the Google Mail login page. The address bar shows a URL from accounts.google.com. The page features the Google logo and a link to 'Konto erstellen' (Create account). The main content area is titled 'Google Mail' and 'E-Mail-Service à la Google'. It includes a description of Google Mail and three feature highlights: 'Viel Speicherplatz' (More storage space), 'Weniger Spam' (Less spam), and 'Zugriff über Ihr Handy' (Access via your phone). On the right side, there is a login form with fields for 'Anmelden' (Log in), 'Nutzername' (Username), and 'Passwort' (Password). Below the password field is an 'Anmelden' button and a checkbox for 'Angemeldet bleiben' (Stay signed in). A message at the bottom of the form states 'Sie können nicht auf Ihr Konto zugreifen?' (You cannot access your account?).

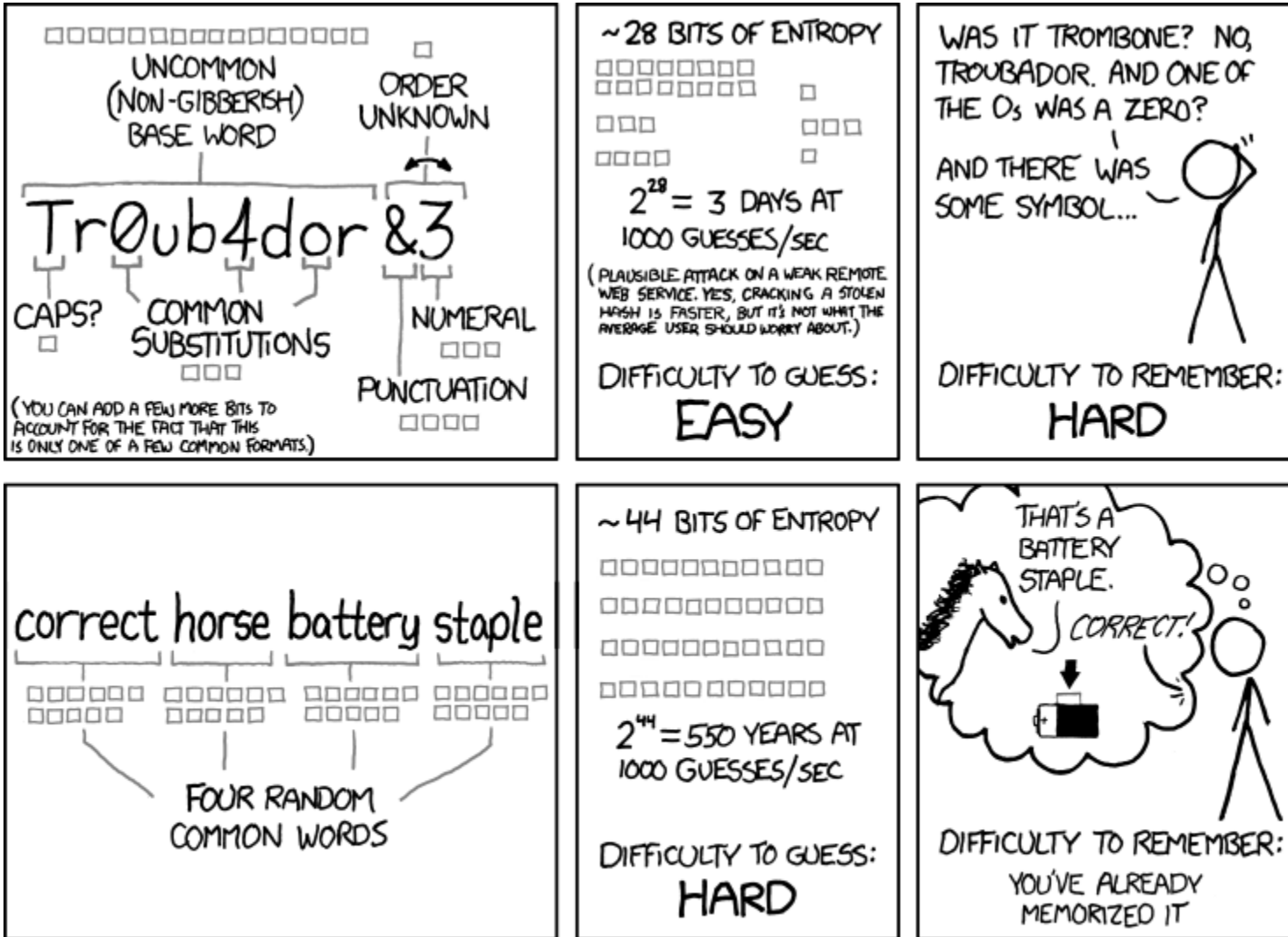
Passwortsicherheit

► 62.000 gestohlene Passwörter (LulSec List)

1. <username> - 780 Vorkommen
2. 123456 - 569 Vorkommen
3. 123456789 - 184 Vorkommen
4. password - 132 Vorkommen
5. romance - 95 Vorkommen
6. mystery - 70 Vorkommen
7. 102030 - 68 Vorkommen
8. tigger - 64 Vorkommen
9. shadow - 64 Vorkommen
10. ajcuivd289 - 62 Vorkommen



Passwortsicherheit



THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

Quelle: XKCD Comics, <http://xkcd.com/936/>

Adressierung per URI

Uniform Resource Identifier (URI)

- ▶ Identifiziert eine abstrakte oder physischer Ressource
- ▶ Obermenge von

Uniform Resource Name (URN) - ISBN

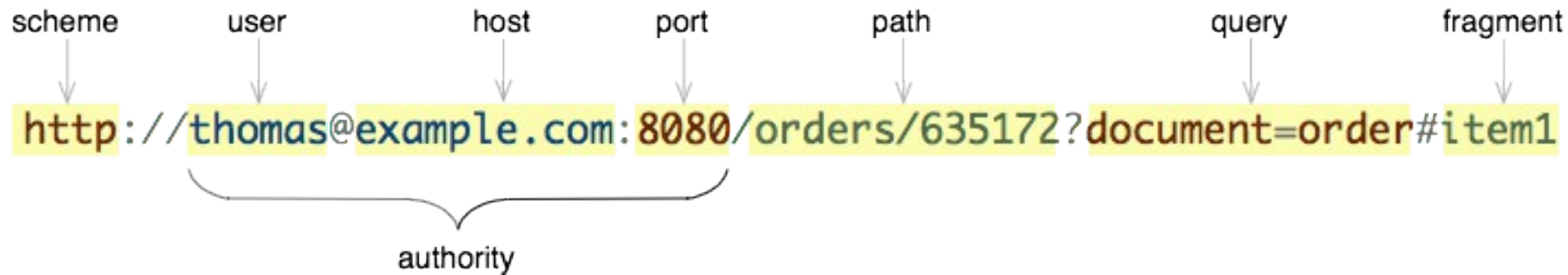
dauerhafter, ortsunabhängiger Bezeichner

Uniform Resource Locator (URL) -Links

identifiziert und lokalisiert eine Ressource

- ▶ Definiert in [RFC 1738](#) und [RFC 3986](#)

Uniform Resource Identifier (URI)



- ▶ **scheme**: gibt den Typ der URI an und definiert die Bedeutung der folgenden Teile (z. B. http, ftp, mailto, urn, doi)
- ▶ **authority**: kommt für viele URI-Schemata vor und bestimmt eine Instanz, die die Namen der URI zentral verwaltet
- ▶ **path**: hierarchisch organisierte Angabe, die auf die Ressource verweist
- ▶ **query**: nicht hierarchische Angaben zur Identifikation der Ressource
- ▶ **fragment**: verweist auf einen Teil der Ressource

Beispiele: URI

urn:isbn:3827370191

file://Users/martina/Sites/index.html

file://localhost/Users/martina/Sites/index.html

mailto:mkraus@hs-mannheim.de

http://bademeister.com

Scheme

Namespace Identifier

Path

Host

Mail-Adresse

Sonderzeichen in URI

Zeichen	Bedeutung	Codierung
<Space >	Erzeugt Probleme in Pfadangaben	+ oder %20
+	Ersetzt Leerzeichen in URLs	%43
@	trennt Benutzername und Passwort vom Servernamen	%40
:	leitet Portnummer ein	%3A
/	trennt Pfadkomponenten voneinander	%2F
?	leitet Parameter ein	%3F
&	trennt Parameter voneinander	%26
=	weist Parameter einen Wert zu	%3D
#	leitet Fragmentbezeichner ein	%23
%	hexadezimaler ASCII / UTF-8 Zeichencode folgt	%25
[]	umrahmt IP V6-Hostadressen	%5B %5D

HyperText Transfer Protocol (HTTP)

Begriffe

Server - bietet Dienstleistung an

- ▶ nimmt Anfragen (*Request*) entgegen und liefert Daten (*Response*)
- ▶ Dienste sind z. B. HTTP, FTP, Mail

Client - nimmt Dienstleistung in Anspruch

- ▶ initiiert eine Verbindung mit dem Server
- ▶ konsumiert Daten vom Server

Pull - Client fragt Daten vom Server ab

- ▶ normale Form der Kommunikation im Internet
- ▶ hierfür ist HTTP gedacht

Push - Server übermittelt von sich aus Daten an Client

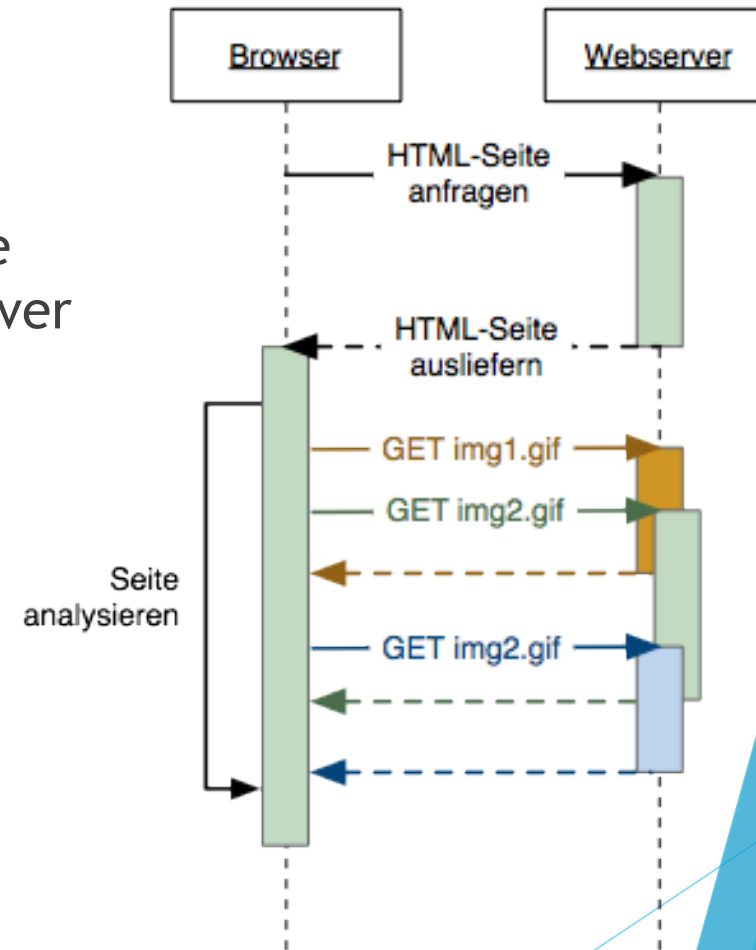
- ▶ hat sich im Internet nur wenig durchgesetzt
- ▶ Broadcast Systeme, Channels, Abonnements

HyperText Transfer Protocol (HTTP)

- ▶ einfaches Protokoll für die Übertragung von Hypertext-Dokumenten über das Internet
- ▶ Request / Response Verfahren über eine TCP-Verbindung
 - mehrere Nachrichten über dieselbe Verbindung
 - Verbindung kann auch nach jedem Request abgebaut werden
- ▶ Klartext-Protokoll
 - reines ASCII
 - unverschlüsselt
 - zeilenorientiert

Interaktion bei HTTP

- ▶ Browser stellt Anfrage nach HTML-Seite
- ▶ Server liefert Seite
- ▶ Browser analysiert Seite und fordert alle abhängigen Ressourcen parallel vom Server an



Zustandslosigkeit

- ▶ HTTP ist grundsätzlich zustandslos
 - ▶ keine Zustand zwischen zwei Aufrufen eines Clients
 - ▶ Verbindung kann zwischen Aufrufen abgebaut werden
 - ▶ Browser-Sitzungen brauchen nicht geschlossen zu werden



Definition idempotent / sicher

def

Einen HTTP-Request, bei dem mehrfache (erfolgreiche) Zugriffe, die gleiche Wirkung haben wie ein einmaliger (erfolgreicher) Zugriff, nennt man *idempotent*.

def

Einen HTTP-Request, bei dem ein (erfolgreicher) Zugriffe keine unerwünschten Seiteneffekte auf dem Server erzeugt nennt man *sicher*.

Request-Line

- ▶ *Request-Line*: erste Zeile des Requests
 - ▶ HTTP-Methode der Anfrage (*HTTP-Verb*)
 - ▶ URI der Ressource die angefordert wird
 - ▶ HTTP-Version (heute immer HTTP/1.1)

GET / HTTP/1.1

Host: www.hs-mannheim.de

User-Agent: Mozilla/5.0

Accept: text/html,application/xhtml+xml;q=0.9,*/*;q=0.8

Accept-Language: en-us,en;q=0.5

Accept-Encoding: gzip, deflate

Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7

Connection: keep-alive

Cache-Control: max-age=0

GET, POST, HEAD und OPTIONS

- ▶ **GET** - Fordert eine Ressource an
 - sollte keine Daten auf dem Server verändern (sicher)
 - Formulardaten können in der URI übergeben werden
- ▶ **POST** - Sendet Daten zum Server
 - darf Daten auf dem Server modifizieren (nicht sicher)
 - Formulardaten werden im Body des Request übertragen
- ▶ **HEAD** - Fordert die Metadaten einer Ressource an
 - wie GET, nur ohne Body
 - wird vom Browser für das Caching verwendet
- ▶ **OPTIONS** - Liefert die Fähigkeiten des Servers

DELETE und PUT

- ▶ **DELETE** - Löscht bestehende Ressource
 - Für Web-Anwendungen nicht relevant (wichtig für REST)
- ▶ **PUT** - Aktualisiert eine Ressource oder legt sie neu an
 - inverse Operation zu GET
 - Für Web-Anwendungen nicht relevant (wichtig für REST)

Wichtige Unterschiede GET/ POST

GET übermittelt alle Formulardaten in der URL

- ▶ Variablen werden mit ? an die URL angehängt
- ▶ für jede Variable name=wert, durch & getrennt (Query)

Vor- und Nachteile

- ▶ Parameter werden in Bookmarks gespeichert
- ▶ Parameter können auch manuell angehängt werden
- ▶ URLs werden sehr lang
- ▶ Viele Web-Server beschränken die URL-Länge (Apache 8190 Bytes)
- ▶ Parameter sind sofort in der URL sichtbar
- ▶ GET sollte keine Daten auf dem Server verändern

Wichtige Unterschiede GET/ POST

POST übermittelt Formulardaten im Request-Body

- ▶ kurze URLs
- ▶ Beliebig große Daten können übertragen werden
- ▶ Parameter nicht in Bookmarks speicherbar
- ▶ Parameter tauchen nicht in URL auf
- ▶ POST ist für Datenänderungen geeignet

Überblick: HTTP-Verben

Verb	sicher	idempotent	URI zeigt auf Ressource	cachebar	Semantik definiert
GET			✓	✓	✓
HEAD			✓	✓	✓
PUT			✓		✓
POST					
OPTIONS					✓
DELETE			✓		✓

53

Überblick: HTTP-Verben

Verb	sicher	idempotent	URI zeigt auf Ressource	cachebar	Semantik definiert
GET	✓	✓	✓	✓	✓
HEAD	✓	✓	✓	✓	✓
PUT		✓	✓		✓
POST					
OPTIONS	✓	✓			✓
DELETE		✓	✓		✓

Response-Line

- ▶ *Response-Line*: erste Zeile des Responses
 - ▶ HTTP-Version (heute immer HTTP/1.1)
 - ▶ Status-Code

HTTP/1.1 200 OK

Date: Fri, 30 Sep 2011 21:10:50 GMT

Server: Apache/2.2.10 (Linux/SUSE)

Expires: Thu, 19 Nov 1981 08:52:00 GMT

Content-Encoding: gzip

Content-Length: 7010

Content-Type: text/html; charset=utf-8

HTTP-Status-Codes

Code	Name	Beschreibung
1xx	Informational	Response wurde ausgelöst, der Request ist aber noch nicht vollständig verarbeitet, z.B. 101 Protocol Switch
2xx	Success	Keine Probleme aufgetreten und der Request konnte verarbeitet werden, z.B. 200 OK
3xx	Redirect	Client muss weitere Schritte durchführen, damit der Request bearbeitet werden kann, z.B. 303 See Other
4xx	Client Error	Ursache des Fehlers liegt im Verantwortungsbereich des Clients, z.B. 404 Not Found
5xx	Server Error	Ursache des Fehlers liegt im Verantwortungsbereich des Servers, z.B. 500 Internal Server Error

100 - Information

#	Name	Erklärung
100	Continue	Anfrage noch in Bearbeitung
101	Switching Protocols	Server stimmt Protokollwechsel zu (z.B. auf HTTPS)

200 - Erfolg

#	Name	Erklärung
200	OK	Anfrage erfolgreich bearbeitet
201	Created	Angeforderte Ressource erzeugt, URL im Location-Header
202	Accepted	Akzeptiert, wird später ausgeführt
203	Non-Authorative	Anfrage bearbeitet, Ergebnis aber mglw. veraltet
204	No Content	Anfrage akzeptiert, keine Ergebnisdaten
205	Reset Content	Client soll Dokument neu aufbauen
206	Partial Content	Angeforderter Teil erfolgreich übertragen (Range-Request)

300 - Umleitung

#	Name	Erklärung
300	Multiple Choice	Client soll aus Ressourcen auswählen
301	Moved Permanently	Ressource ist dauerhaft umgezogen, Adresse im Location-Header
302	Found	Ressource temporär umgez., Adresse im Location-Header (GET)
303	See Other	Antwort unter Adresse, die im Location-Header steht (GET)
304	Not Modified	Antwort gegenüber voriger Anfrage unverändert
305	Use Proxy	Ressource nur über Proxy (Adresse in Location-Header) erreichbar
306	-	(reserviert)
307	Temporary Redirect	Ressource temporär umgez., Adresse im Location-Header

400 - Client-Fehler

#	Name	Erklärung
400	Bad Request	Ungültige Anfrage-Nachricht
401	Unauthorized	Authentifizierung notwendig
402	Payment Required	(reserviert)
403	Forbidden	Zugriff verboten, unabhängig von Authentifizierung
404	Not Found	Ressource nicht gefunden
405	Method Not Allowed	Falsche Zugriffsmethode (GET / POST)
406	Not Acceptable	Ressource nicht im gewünschten Format verfügbar
407	Proxy Authentication	Authentifizierung (Login / Password) am Proxy notwendig
408	Request Time-out	Server hat Anfrage nicht innerhalb Server-Timeout empfangen

400 - Client-Fehler

#	Name	Erklärung
409	Conflict	Anfrage unter falschen Voraussetzungen (PUT-Konflikt)
410	Gone	Ressource dauerhaft entfernt, steht nicht mehr bereit
411	Length Required	Anfrage muss mit Content-Length gestellt werden
412	Precondition Failed	Voraussetzung (z.B. If-Match) trifft nicht zu
413	Request Entity Too Large	Anfrage ist zu groß
414	Request-URI Too Long	URI der Anfrage zu lang
415	Unsupported Media Type	Medientyp wird vom Server nicht unterstützt
416	Request Range Not Satisfiable	Angeforderter Range nicht vorhanden
417	Expectation Failed	Server kann gefordertes Verhalten (Expect-Header) nicht zeigen

500 - Server-Fehler

#	Name	Erklärung
500	Internal Server Error	Unerwarteter Serverfehler
501	Not Implemented	Funktionalität vom Server nicht implementiert
502	Bad Gateway	Proxy-Server hat ungültige Antwort vom benutzen Server erhalten
503	Service Unavailable	Server steht nicht zur Verfügung
504	Gateway Timed Out	Proxy-Server hat seinerseits Timeout vom benutzen Server erhalten
505	HTTP Version Not Supported	HTTP-Version vom Server nicht unterstützt

Content Negotiation

- ▶ Der Client teilt dem Server mit, welche Medientypen und Formate er empfangen möchte (*content negotiation*)

```
GET / HTTP/1.1
```

```
Host: www.hs-mannheim.de
```

```
User-Agent: Mozilla/5.0
```

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

```
Accept-Language: en-us,en;q=0.5
```

```
Accept-Encoding: gzip, deflate
```

```
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
```

```
Connection: keep-alive
```

```
Cache-Control: max-age=0
```

```
HTTP/1.1 200 OK
```

```
Date: Fri, 30 Sep 2011 21:10:50 GMT
```

```
Server: Apache/2.2.10 (Linux/SUSE)
```

```
Expires: Thu, 19 Nov 1981 08:52:00 GMT
```

```
Content-Encoding: gzip
```

```
Content-Length: 7010
```

```
Content-Type: text/html; charset=utf-8
```

MIME-Typ

Content Negotiation basiert auf *MIME-Typ* (*MIME type*)

- ▶ *MIME* = *Multi Purpose Internet Mail Extensions*
- ▶ Definiert den Typ einer Nachricht im Internet
- ▶ Besteht aus
 - *Medientyp* (*content type*)
 - *Subtyp* (*subtype*)
- ▶ Syntax: Medientyp/Subtyp
- ▶ Ursprünglich für in [RFC1049](#) für E-Mail spezifiziert, aktuelle Spezifikation ist [RFC2047](#)
- ▶ [Liste der MIME-Typen](#) wird von der IANA verwaltet

Medientyp

Medientyp	Bedeutung	Beispiel(e)
text	Textuelle Daten	text/plain, text/html, text/xml
audio	Audio	audio/ogg, audio/mp4, audio/mpeg
image	Bilder	image/jpeg, image/tiff, image/gif
video	Video	video/mpeg, video/ogg
application	Programmspezifische Daten	application/zip, application/pdf; application/json
message	Nachrichten	message/rfc822, message/partial
multipart	Mehrteilige Daten	multipart/signed
example	Typ für Beispiele und Dokumentation	-
model	Strukturierte Daten	model/mesh

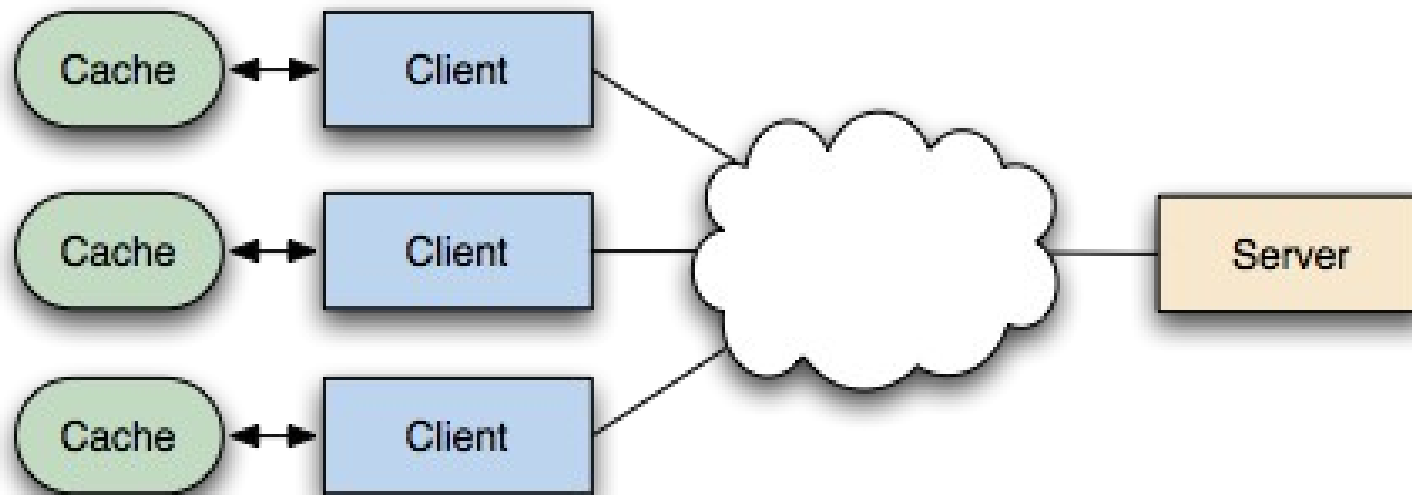
Cache-Topologien

Unterschiedliche Cache-Topologien sind möglich

- ▶ Ausschließlich clientseitiger Cache
- ▶ Clientseitig shared Cache (Proxy)
- ▶ Server-Cache
- ▶ Beliebige Mischformen dieser drei Formen

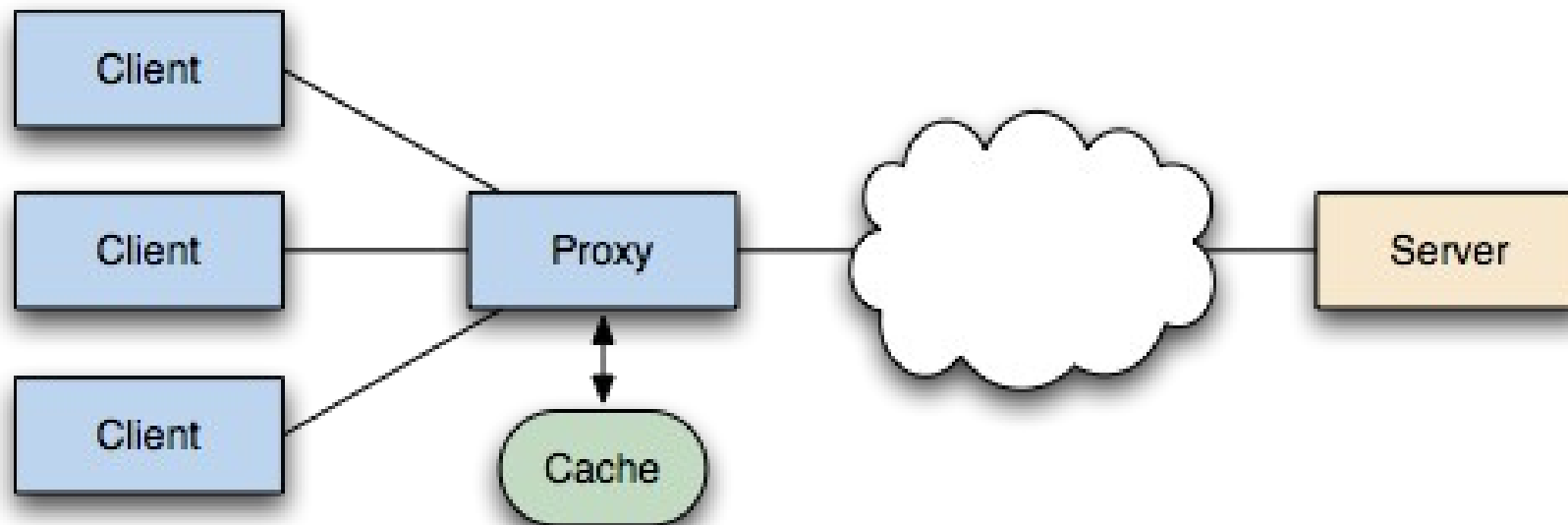
Cache-Topologien

- ▶ Ausschließlich clientseitiger Cache



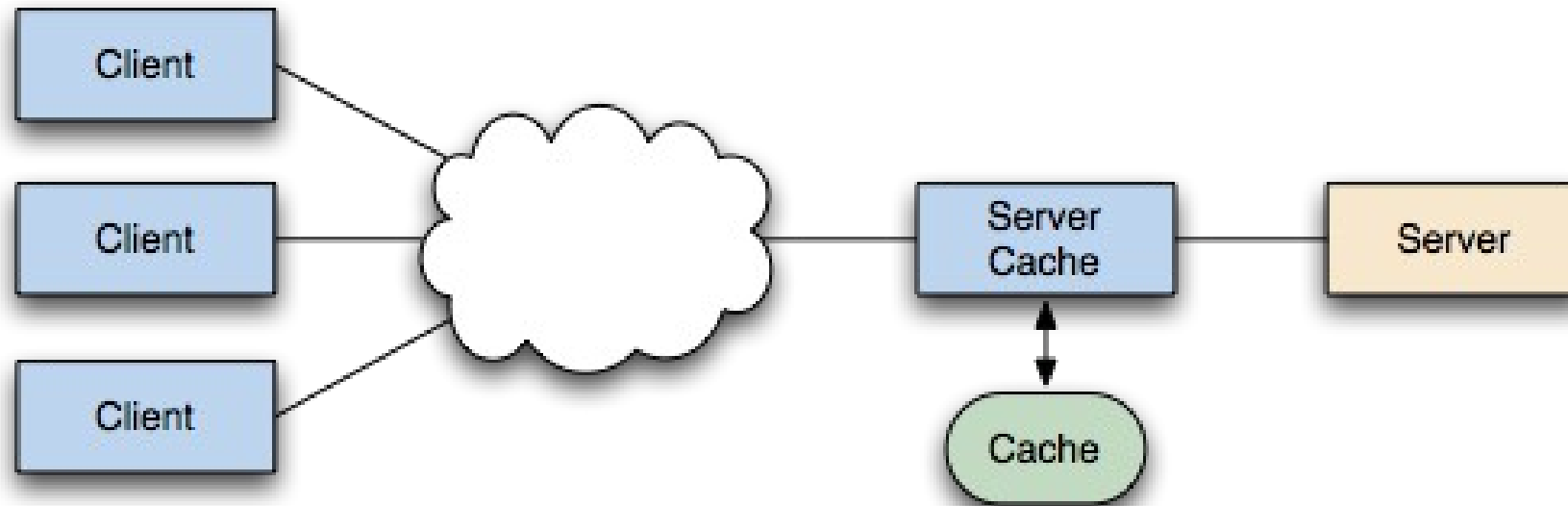
Cache-Topologien

- ▶ Clientseitiger shared Cache (Proxy)



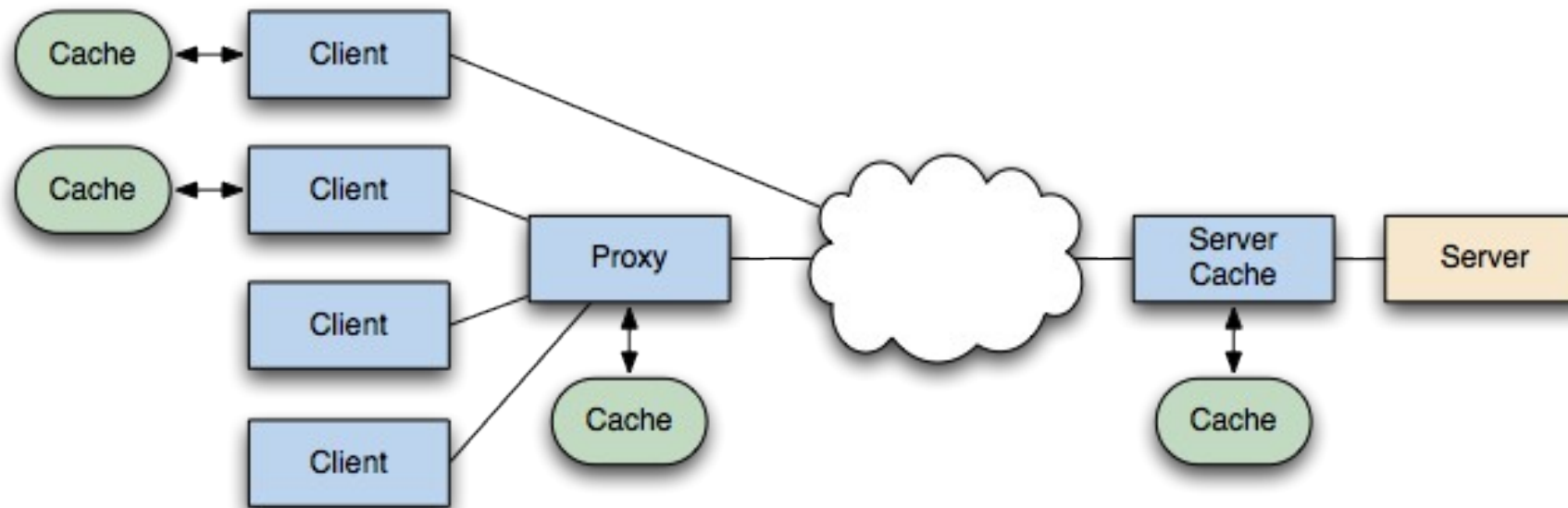
Cache-Topologien

► Server-Cache



Cache-Topologien

► Mischform



Konsistenz des Caches

- ▶ Wie kann man den Cache konsistent halten?
 - ▶ Server notifiziert die Caches über Änderungen (*Notifikationsmodell*)
 - ▶ Server sagt dem Cache, wie lange Daten gültig sind (*Expirationsmodell*)
 - ▶ Cache fragt beim Server nach, ob Daten noch gültig sind (*Validierungsmodell*)



Welche dieser Methoden ist für HTTP ungeeignet und warum?

Expirationsmodell

- ▶ Server sendet die Gültigkeitsdauer der Daten im Response mit

```
HTTP/1.1 200 OK
Date: Mon, 03 Oct 2011 14:14:27 GMT
Server: Apache
Cache-Control: max-age=60
Content-Language: de-DE
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
```

- ▶ Alternativ kann er auch ein absolutes Datum angeben
`Expires: Tue, 04 Oct 2011 14:14:27 GMT`
- ▶ Client (oder Proxy) fragt die Ressource erst nach Ablauf der Gültigkeitsdauer erneut an
- ▶ Alle Anfragen vor Ablauf werden aus dem Cache beantwortet

Cache-Control-Header: Response

- ▶ Response-Header-Feld `Cache-Control` steuert Caching

Attribut	Bedeutung
<code>public</code>	Darf auch von einem shared Cache gecached werden
<code>private</code>	Darf von einem shared Cache <u>nicht</u> gecached werden
<code>no-cache</code>	Darf nicht gecached werden
<code>no-store</code>	Darf nicht persistent (z.B. auf Platte) abgelegt werden
<code>must-revalidate</code>	Ressource noch einmal validiert werden bevor sie ausgeliefert wird
<code>proxy-revalidate</code>	Wie <i>must-revalidate</i> aber nur für shared Caches
<code>max-age=...</code>	Zeit in Sekunden, für die die Antwort gültig ist

Validierungsmodell

- ▶ Client (oder Proxy) fragt beim Server an, ob seine Kopie der Ressource noch aktuell ist
- ▶ Client kann bedingten Request stellen

```
GET /orders/ HTTP/1.1  
Host: hs-mannheim.de  
If-Modified-Since: Mon, 03 Oct 2011 14:14:27 GMT
```

- ▶ Client kann eine Prüfsumme (*Entity-Tag*, *Etag*) über die Ressource berechnen und entsprechende Anfrage stellen

```
GET /orders/ HTTP/1.1  
Host: hs-mannheim.de  
If-None-Match: "b1b88d1e56778fe933fd4de66342f59b"
```

- ▶ Wenn sich die Ressource nicht verändert hat, liefert der Server ein *304 Not Modified*, andernfalls die Ressource

Prüfsummen mit ETag

Zwei Arten von ETags:

- ▶ **Starke ETags** (*strong etags*) - Response muss Byte für Byte identisch sein, damit der Server dasselbe ETag verwenden darf
 - ▶ **Schwache ETags** (*weak etags*) - Repräsentieren Ressource auf logischer Ebene, Darstellung darf sich bei identischem ETag leicht unterscheiden
-
- ▶ Zu erkennen an einem vorangestellten W/
Etag: W/"845250bc4bb5783e0d618fcc97204e81"

Cache-Control-Header: Request

- ▶ Request-Header-Feld `Cache-Control` steuert Caching

Attribut	Bedeutung
<code>no-cache</code>	Antwort muss vom original Server kommen, nicht aus dem Cache
<code>no-store</code>	Darf nicht persistent (z.B. auf Platte) abgelegt werden
<code>max-age=...</code>	Zeit in Sekunden, die die Antwort alt sein darf. 0 entspricht <i>no-cache</i>
<code>max-stale=...</code>	Client nimmt auch nicht aktuellen Antworten, solange sie nicht älter als die angegebenen Sekunden sind
<code>min-fresh=...</code>	Wie lange muss die Antwort mindestens noch gültig sein
<code>only-if-cached</code>	Client will die Daten nur, wenn sie aus einem Cache stammen

Skalierbarkeit

def

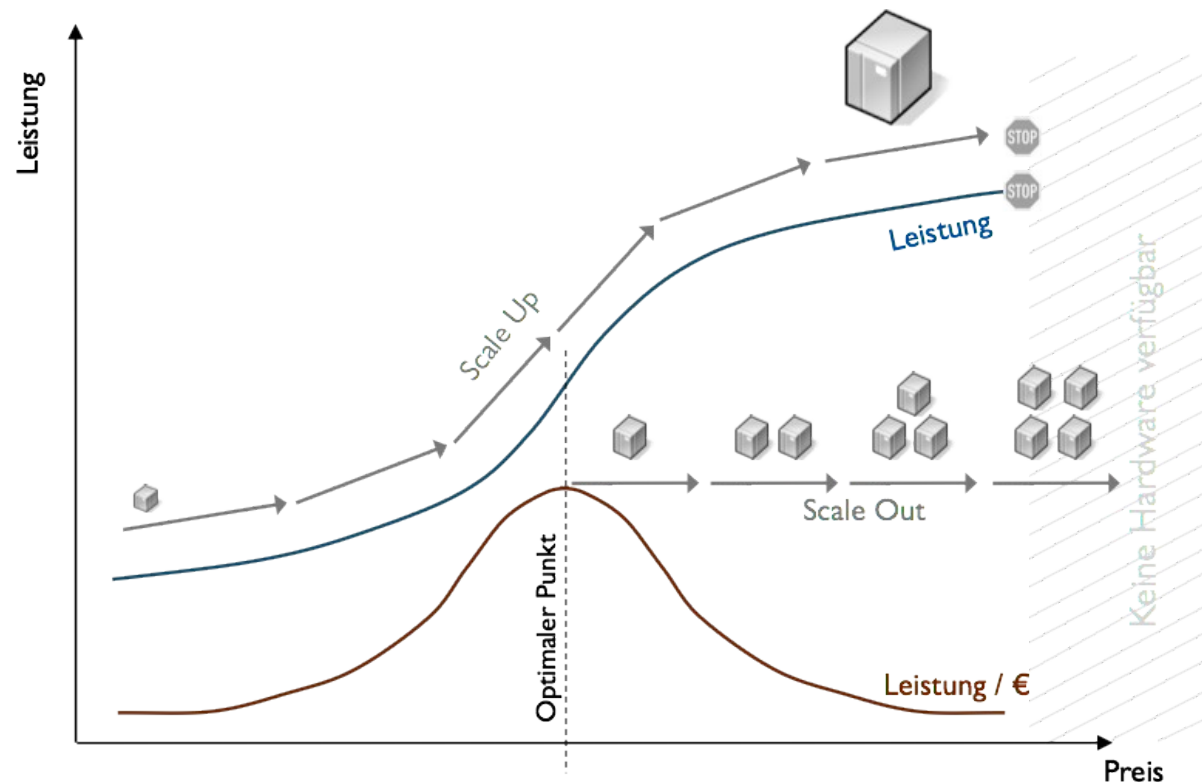
Skalierbarkeit ist die Fähigkeit eines Softwaresystems sich an eine wachsende (oder sinkende) Nutzungsrate flexibel anzupassen.

Eine Architektur

- ▶ *Skaliert gut*, wenn der Ressourcenbedarf der Anwendung linear mit der Nutzung wächst
- ▶ *Skaliert schlecht*, wenn der Ressourcenbedarf über-linear mit der Nutzung wächst

Grundlegende Ansätze zur Skalierung

- ▶ **Vertikale Skalierung** (*scale up*) - Hardware wird vergrößert (mehr CPU, mehr Speicher, etc.)
- ▶ **Horizontale Skalierung** (*scale out*) - Hardware wird dupliziert und Services werden mehrfach angeboten



Verfügbarkeit

def

Ein System ist *verfügbar*, wenn es den Nutzern im erwarteten und zugesicherten Umfang Dienstleistungen zur Verfügung stellt. *Verfügbarkeit* beschreibt das Verhältnis aus der Zeit, die ein System in einem gegebenen Zeitraum verfügbar ist zur Gesamtzeit des Zeitraums.

$$\text{Verfügbarkeit} = \frac{\text{verfügbare Zeit}}{\text{Gesamtzeit}} = \frac{\text{verfügbare Zeit}}{\text{verfügbare Zeit} + \text{Ausfallzeit}}$$

Verfügbarkeitsanforderungen

- ▶ Zeiten der Nichtverfügbarkeit (Ausfallzeiten) können
 - ▶ geplant sein: z. B. Wartungsarbeiten
 - ▶ ungeplant sein: z. B. Auftreten eines Fehlers.
- ▶ Typische Verfügbarkeitsanforderungen
 - ▶ 24 * 7-Tage Betrieb - Keine Wartungsfenster
 - ▶ 90% Verfügbarkeit - Ausfall maximal für 36 Tage pro Jahr
 - ▶ 99% Verfügbarkeit - Ausfall maximal für 3,7 Tagen pro Jahr
 - ▶ 99,9% Verfügbarkeit - Ausfall maximal für 9 Stunden pro Jahr
 - ▶ 99,99% Verfügbarkeit - Ausfall maximal für 53 Minuten pro Jahr
 - ▶ 99,999% Verfügbarkeit - Ausfall maximal für 5 Minuten pro Jahr
 - ▶ 99,9999% Verfügbarkeit - Ausfall maximal für 30 Sekunden pro Jahr

Thin-Client- vs. Fat-Client

- ▶ Thin- und Fat-Client-Architekturen verfeinern die Zuordnung der Anwendungslogik
 - ▶ *Ultra-Thin-Client* - Client dient nur der reinen Anzeige (typisch für 4-Tier), z. B. Web-Browser
 - ▶ *Thin-Client* - Client beschränkt sich auf die Anzeige und die Aufbereitung der Daten zur Anzeige, z. B. Web-Browser mit AJAX-Framework
 - ▶ *Fat-Client* - Teile der Anwendungslogik liegen zusammen mit der Präsentation auf der Client-Tier (typisch für 2-Tier)

Numbers everyone should know

L1 cache reference	0,5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10.000 ns
Send 2K bytes over 1 Gbps network	20.000 ns
Read 1 MB sequentially from memory	250.000 ns
Round trip within same datacenter	500.000 ns
Disk seek	10.000.000 ns
Read 1 MB sequentially from network	10.000.000 ns
Read 1 MB sequentially from disk	30.000.000 ns
Send packet CA->Netherlands->CA	150.000.000 ns