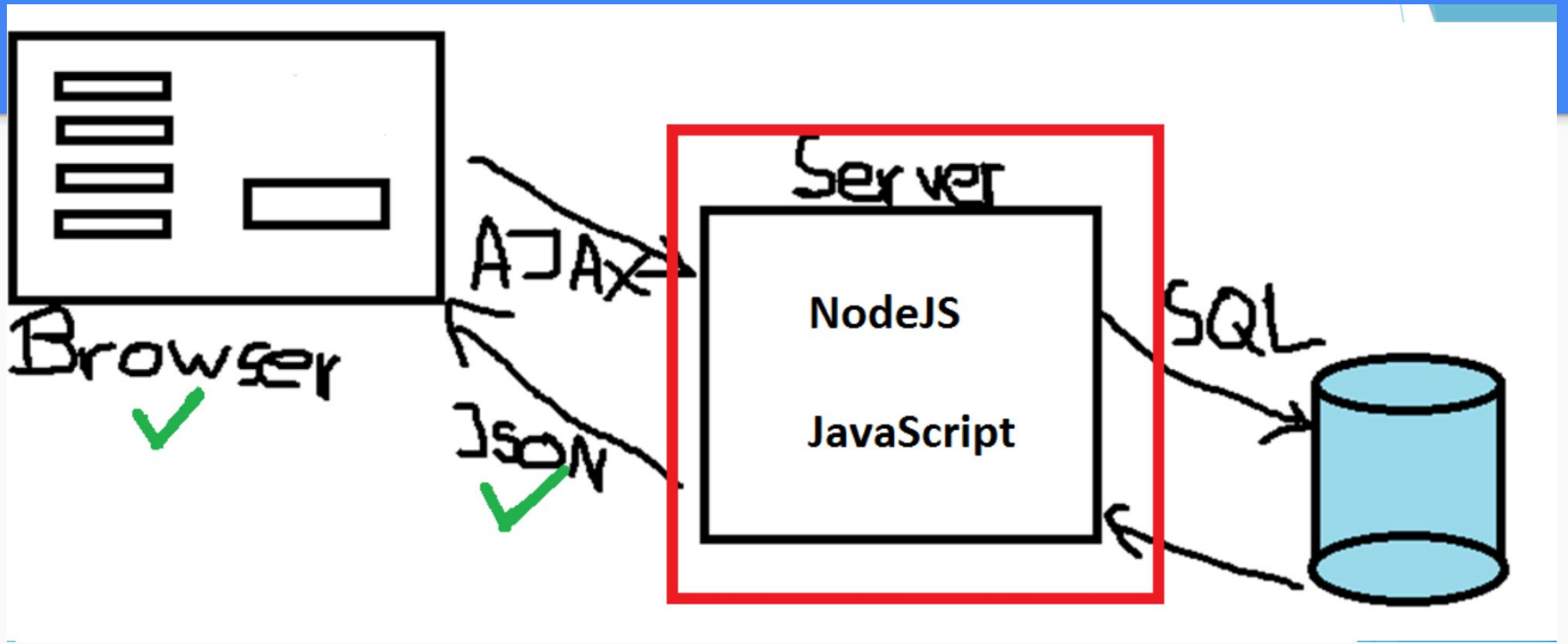
WebServer

Martina Kraus

Inhaltsverzeichnis

- WebServer
  - Konfiguration
  - Anforderungen
- Echtzeit
- Cache-Topologien
- Cache-Control
- Skalierbarkeit/ Verfügbarkeit

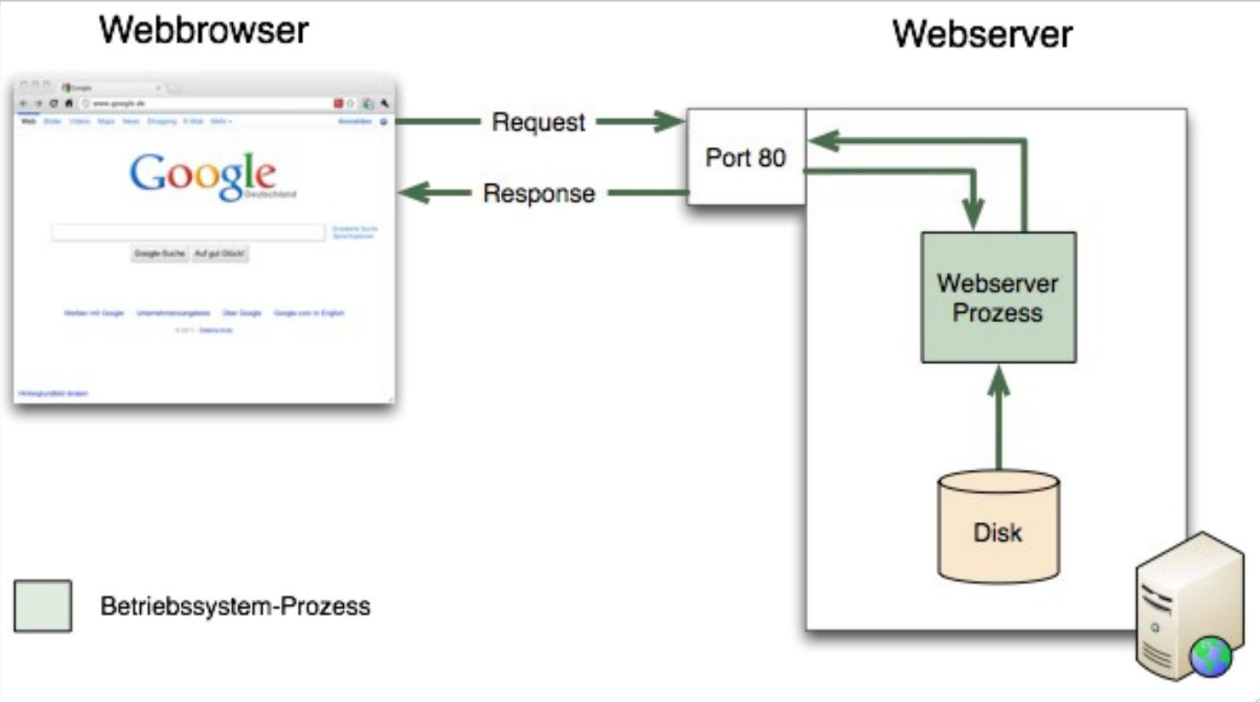
Übersicht



Definition

- Server:
  - Ein Programm welches Ressourcen und Dokumente bereitstellt
- WebServer
  - Ein Server welcher über HTTP HTML, JavaScript, CSS-Dateien oder Daten an einen Client übersendet

Architektur



Funktionalitäten

- Ausliefern statischer Dateien
- Zugriffsbeschränkung
- Sicherheit (HTTPS)
- Cookie Verwaltung
- Weiterleitung
- Fehlerbehandlung
- Protokollierung
- Caching

Auswahl

Apache (1995) (http://httpd.apache.org)

- 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

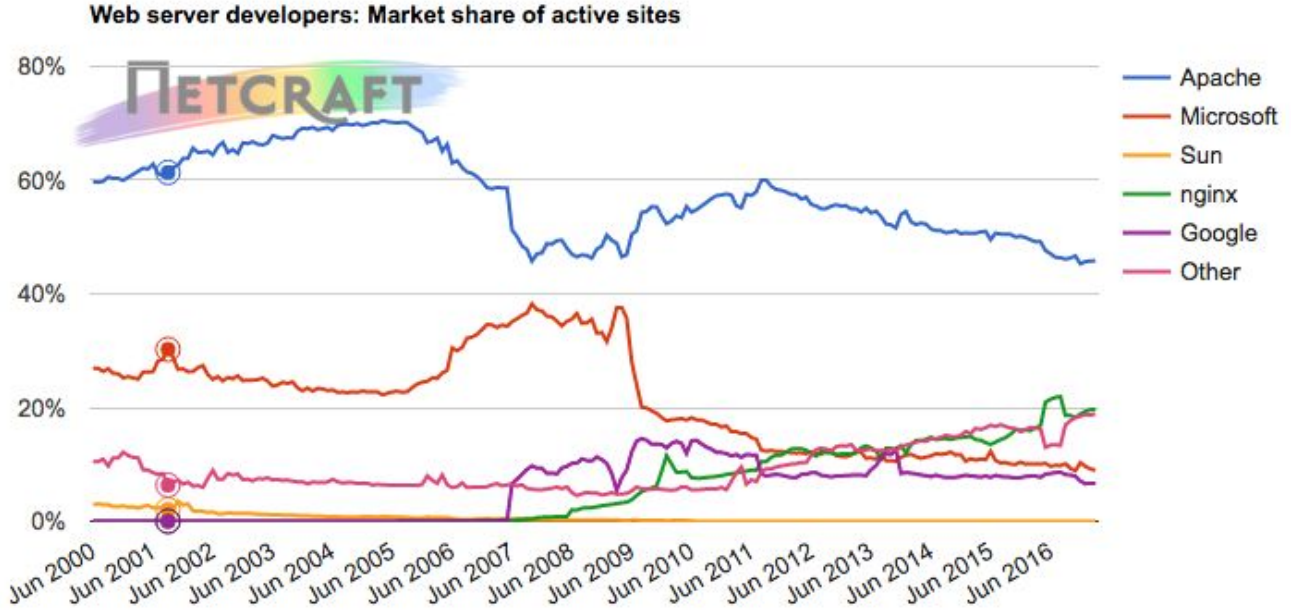
Auswahl

Google Web Server (GWS)

- Google betreibt damit ca. 10 Millionen eigene Websites ▸ nicht allgemein verfügbar
- Auf Linux basierend

Nginx (2004) (http://nginx.org/)

- freier Webserver unter BSD-Lizenz
- kleiner und schlanker Webserver



Developer	February 2017	Percent	March 2017	Percent	Change
Apache	79,593,938	45.78%	79,942,445	45.82%	0.05
nginx	34,088,228	19.60%	34,317,972	19.67%	0.07
Microsoft	16,031,854	9.22%	15,611,256	8.95%	-0.27
Google	11,656,739	6.70%	11,684,677	6.70%	-0.01

Allgemeine Konfiguration

- Konfiguration ist abhängig vom eingesetzten Server
- Webserver läuft als
  - Anwendung – gut für Testzwecke (localhost bzw 127.0.0.1)
  - Daemon (Dienst) – gut für den Betrieb einer öffentlichen Seite

- Für Testzwecke auch ohne Internetzugang zu verwenden

Grundeinstellung

- 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

Grundeinstellung

- 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
- Log-Dateien
  - Protokollierung der Zugriffe/ Fehler *access.log/ error.log*

Grundeinstellung

- Timeouts
- MIME-Types
  - 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

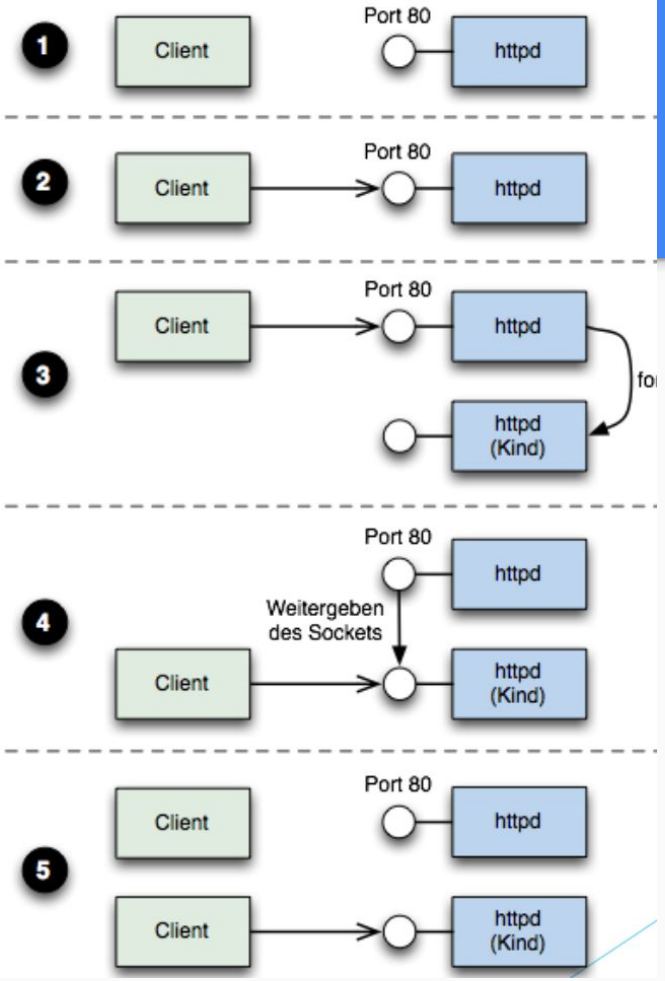
Apache ist der am häufigsten eingesetzte Server

- Open-Source (http://httpd.apache.org/)
- Seit 1995 verfügbar
- Aktuelle Version 2.4.25 (released 2016-12-20)
- 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 fuer 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 fuer 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

Anforderungen an gute Web-Server

Viele gleichzeitig eintreffende Requests schnell und zufriedenstellend verarbeiten.

- Viel: Das C10k Problem
- Schnell: „Echtzeit“
- Zufriedenstellend: Nach Verarbeitung ein Ergebnis

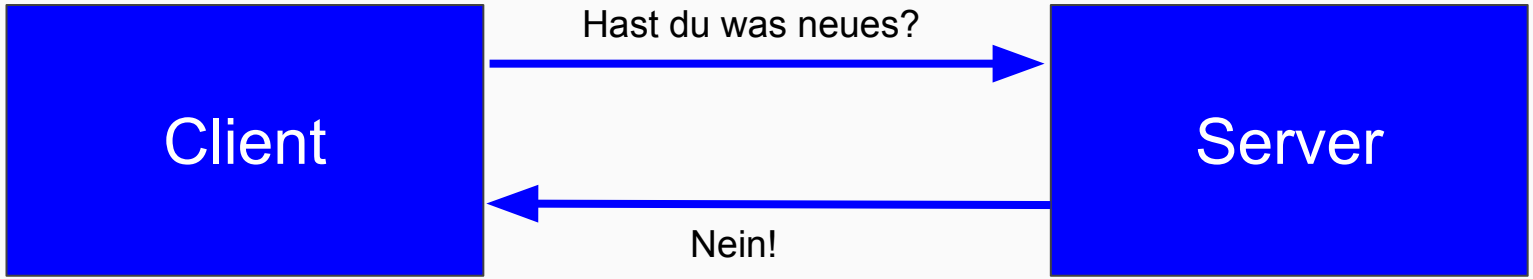
Echtzeit

„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“ (Wikipedia)

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

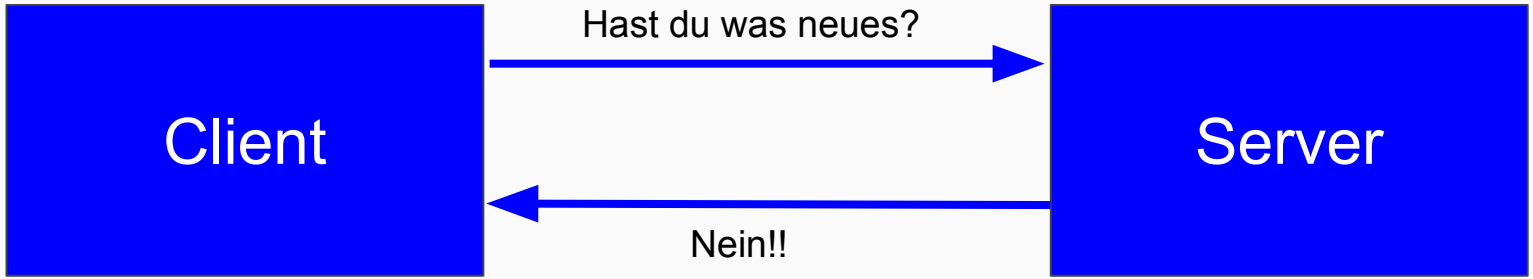
Polling (2000)

Nach einer Sekunde:



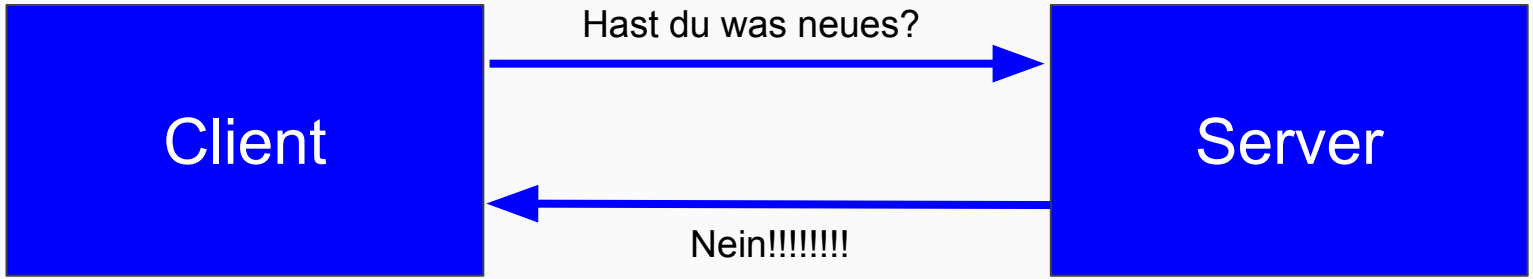
Polling (2000)

Nach einer Sekunde:



Polling (2000)

Nach einer Sekunde:



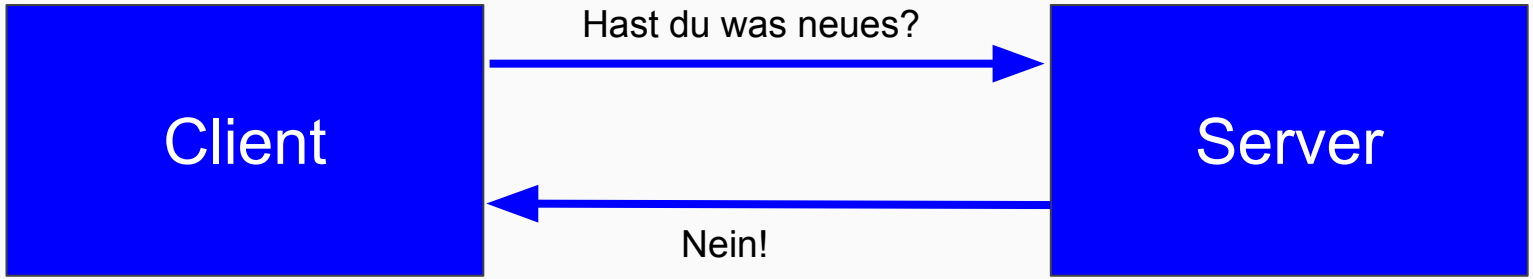
Polling (2000)

Client

Server

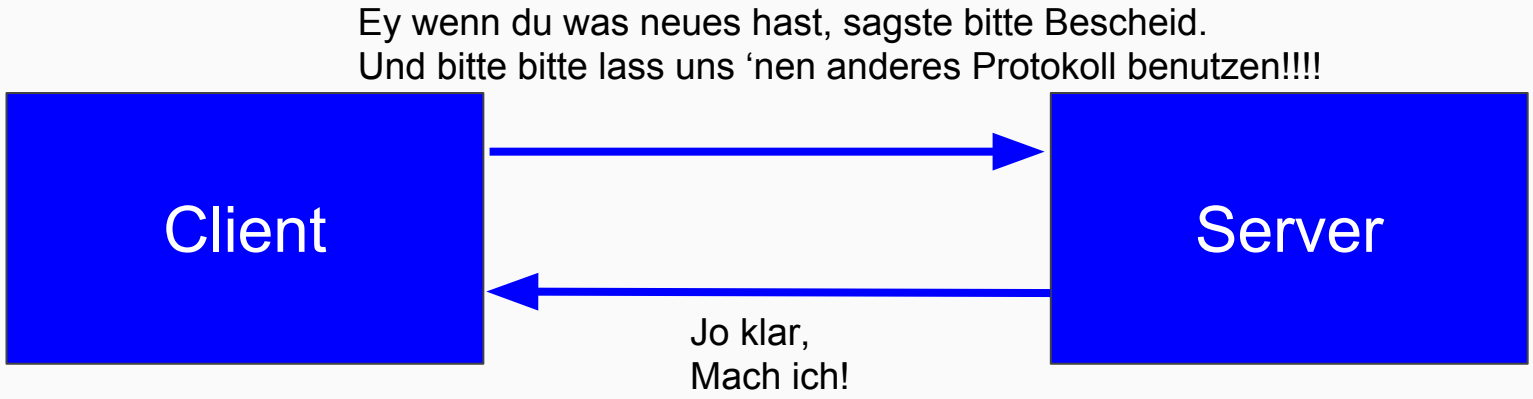
Comet / Long-Polling (2000)

Nach einer Sekunde:



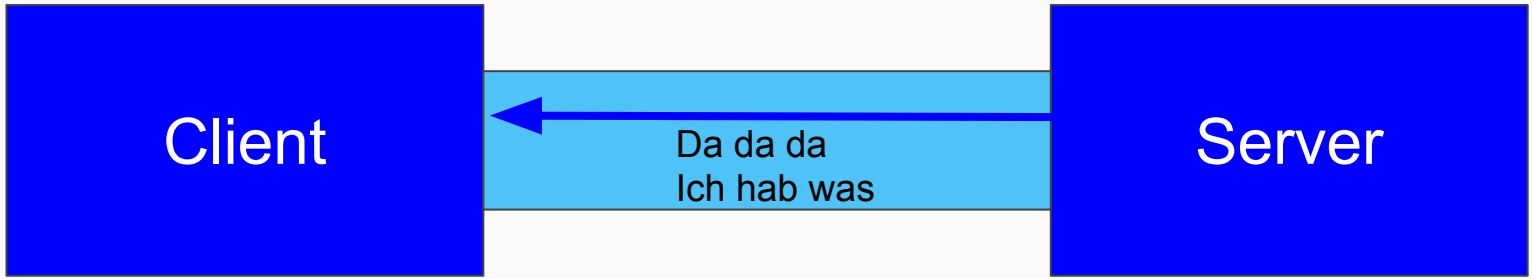
Comet / Long-Polling (2000)

WebSockets (2011)



WebSockets (2011)

WebSockets (2011)



Cache-Topologien

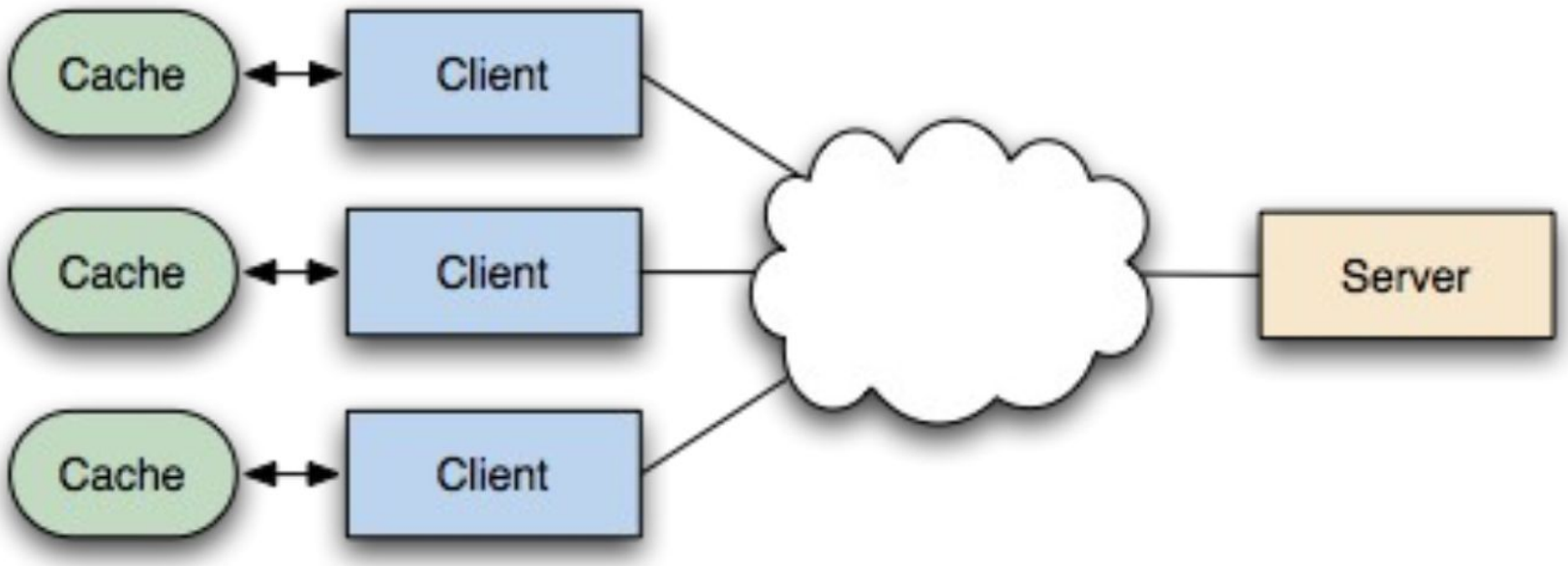
Cache

- Ein schneller Puffer-Speicher
- Wiederholte gleiche Zugriffe auf langsames Hintergrundmedium
- Daten / Dateien die bereits einmal geladen wurden verbleiben im Cache
- Cache muss immer wieder mit aktuellen Daten geupdated werden

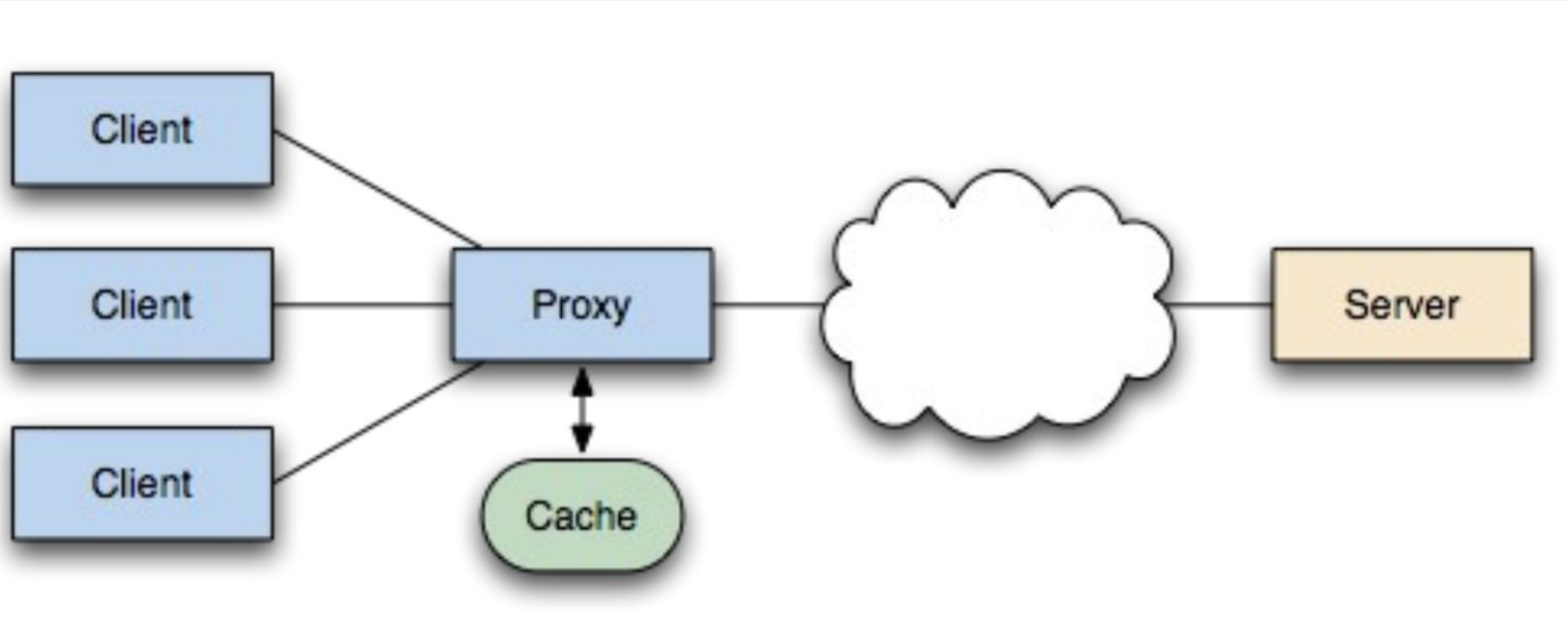
Cache-Topologien

- Unterschiedliche Cache-Topologien sind möglich:
  - Clientseitiger Cache
  - Clientseitiger shared Cache
  - Server-Cache
  - Mischformen

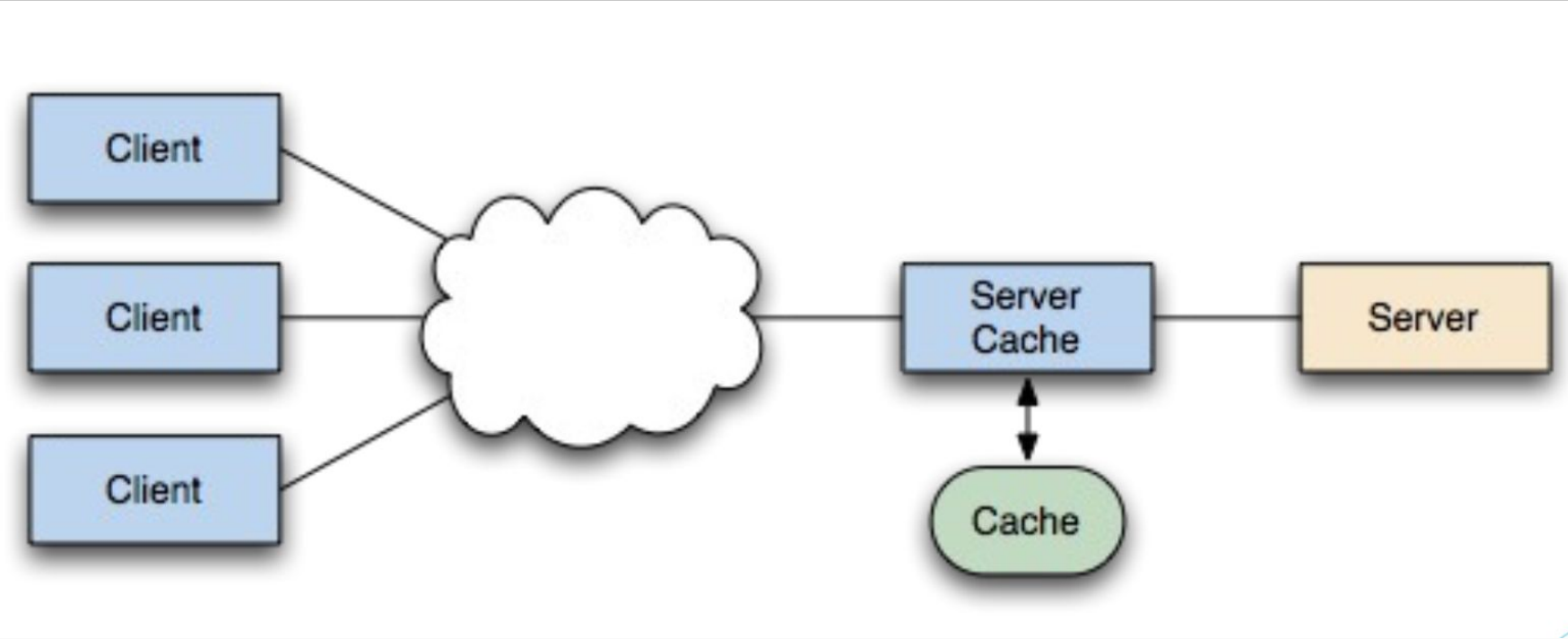
Clientseitiger Cache



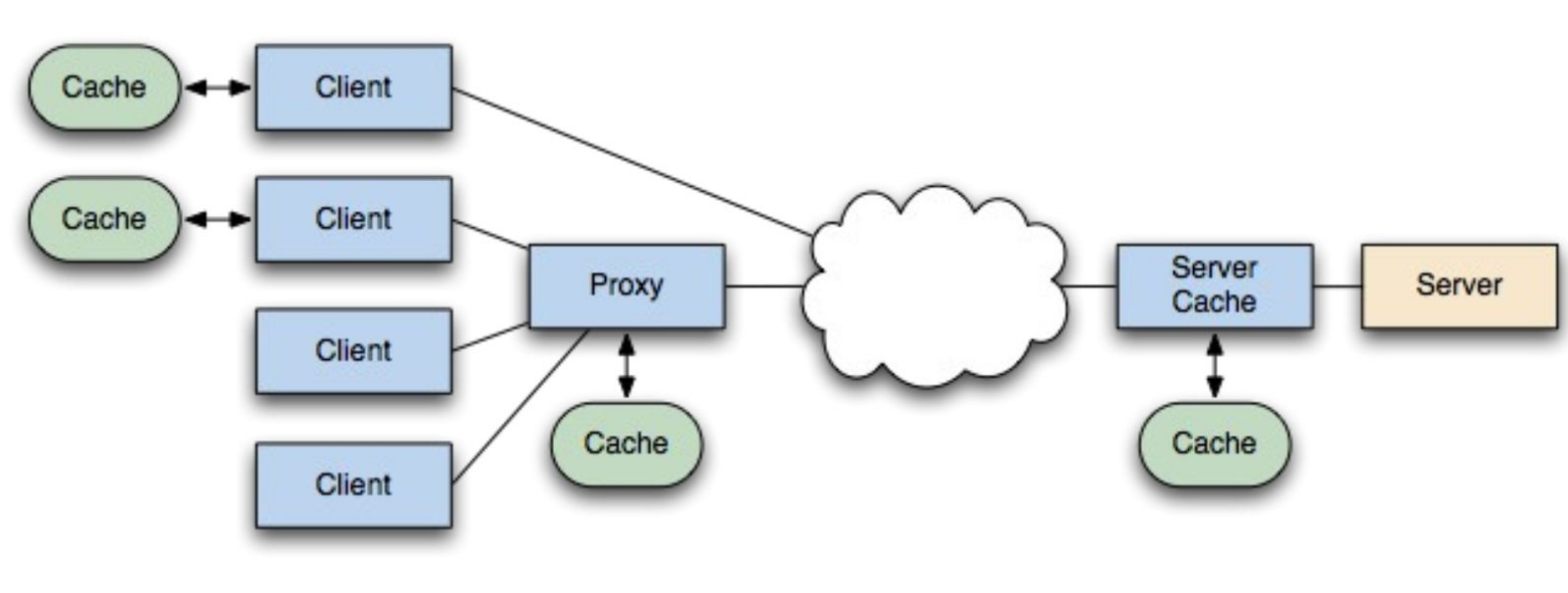
Clientseitiger shared Cache



Server Cache



Mischform


Cache
Client
Cache
Client
Client
Client
Proxy
Cache
Server
Cache
Server
Cache

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?

Exprationsmodell

- 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
```

Exprationsmodell

- 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
`public`	Darf auch von einem shared Cache gecached werden
`private`	Darf von einem shared Cache nicht gecached werden
`no-cache`	Darf nicht gecached werden
`no-store`	Darf nicht persistent (z.B. auf Platte) abgelegt werden
`must-revalidate`	Ressource noch einmal validiert werden bevor sie ausgeliefert wird
`proxy-revalidate`	Wie *must-revalidate* aber nur für shared Caches
`max-age=…`	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"
```

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

Skalierbarkeit/ Verfügbarkeit

Skalierbarkeit

def

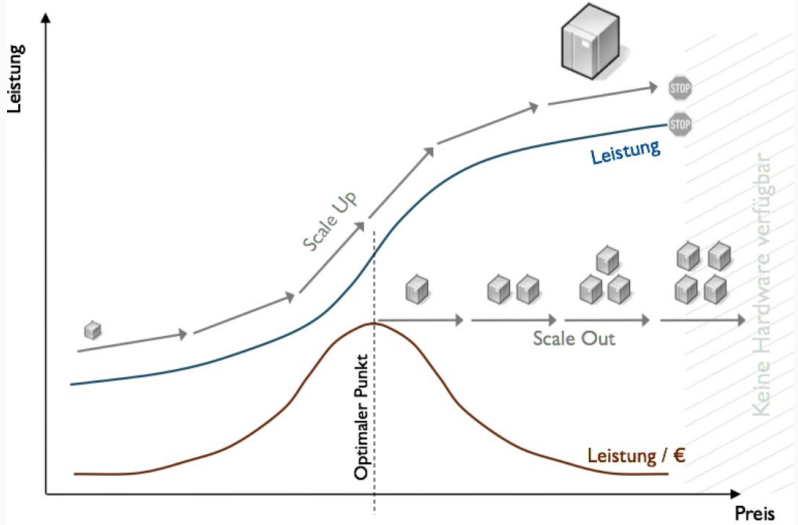
Skalierbarkeit ist die Fähigkeit eines Softwaresystems sich an eine wachsende (oder sinkende) Nutzungsrate flexibel anzupassen.

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 wächst

Skalierbarkeit

- *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

Verfügbarkeitsanforderungen

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

Fragen???