Professionelles Webhosting

Basiswissen aus Sicht des **Kunden**

„Selbst die beste Webseite ist ohne einen funktionierenden Webserver nur eine Ansammlung nutzloser Files.“

Ausschlaggebend für diesen Vortrag

Sehr geehrter Herr Unglaub,
Ich habe bei meinem jetzigen Anbieter eine .at Adresse registriert und von der Firma nic.at eine Bestätigung im PDF Format bekommen. Das ist ein Besitzerzertifikat. Ich habe dieses wie es sich gehört unter C:/xampp/htdocs abgespeichert, allerdings funktioniert der E-Mail Versand nicht wie gewünscht und der Zugriff auf die Homepage funktioniert auch nicht wie gewünscht, genauer gesagt funktioniert er gar nicht. Sie wurden mir von einem Bekannten empfohlen, bitte können Sie mir helfen meine Webseite zum laufen zu bringen?

Inhalt / Roadmap

- Welche Server-Dienste braucht man für eine Webseite?
- Welche Server-Dienste (Daemons) gibt es?
- Protokolle und noch mal Protokolle ;)
- Das HTTP-Protokoll im Detail
- Beispielseitenaufruf
- Fragen und hoffentlich Antworten ;)

Eine kleine Überraschung

Welche Server-Dienste braucht man für eine Webseite?

- Server-OS (ist kein wirklicher Dienst)
- HTTP-Server
- Mailserver
  - SMTP-Server
  - IMAP / POP3 Server
- DNS-Server
- ggf. Datenbank-Server

Server OS

- Ein Server sollte möglichst „schlank“ sein.
- Je weniger drauf ist, desto weniger kann gehackt/gecrackt werden
- Professionelle Server haben keine GUI

- Mögliche Server OS-Systeme
  - Linux (Debian, Ubuntu, Red Hat, openSuse, Gentoo, Slackware, Mint, …)
  - Windows Web Edition
- Ich verwende Debian!

HTTP-Server

- Liefert angeforderte Daten an den Client aus.
  - Statische Dokumente
  - Dynamisch generierte Dokumente (Bsp: PHP)

- Mögliche Daemons
  - Apache2 (stable **2.2**, Debian: **2.2.9-10+lenny7**)
  - lighthttpd (stable: **1.4.26**, Debian: **1.4.19-5+lenny1**)
  - IIS vom Microsoft („stable“: **7**)

Mailserver

- Technisch gesehen gibt es keinen „Mailserver“
- Überbegriff für
  - SMTP (transport)
  - IMAP4 / POP3 (storage)

SMTP-Server

- SMTP-Server senden und empfangen E-Mails zwischen allen SMTP-Servern weltweit
- Empfangene E-Mails werden dem LDA übergeben
- ggf. Übergabe an content scanner (amavisd, clamav, spamassassin)
- Mögliche Daemons
  - Postfix (stable: **2.7**, Debian: **2.5.5-1.1**)
  - Sendmail (stable: **8.14.4**, Debian: **8.14.3-5+lenny1**)
  - Exim (stable: **4.71**, Debian: **4.69-9**)
  - Microsoft Exchange (transport und storage in einem)

IMAP4 / POP3 Server

- Verwalten die E-Mails in den Mailboxen
- Gewähren dem User Zugriff auf seine E-Mails
- Bekannteste Speicherformate:
  - Mbox
  - Maildir (besser, rockstable)
  - .pst (Datendatei)
- Mögliche Daemons
  - Cyrus (stable: **2.3.16**, Debian: **2.2.13-14+lenny3**)
  - Dovecot (stable: **1.2.11**, Debian: **1:1.2.11-1~bpo50+2**)
  - Courier (stable: **4.7.9**, Debian: **4.4.0-2**)
  - Microsoft Exchange

IMAP4 vs. POP3

- POP3
  - E-Mails werden vom Server in das Mail-Programm (MUA) heruntergeladen
  - Keine Speicherung nach dem Abrufen am Server (kann je nach MUA geändert werden)
  - **Vorteil:** sehr schnell und traffic sparend
  - **Vorteil:** Einfaches Protokoll welches schnell selbst implementiert werden kann
  - **Nachteil:** Wenn die Mailbox auf mehreren PC's verwendet wird herscht ein asynchroner Datenbestand
- IMAP4
  - MUA zeigt „Abbild“ des Servers an
  - E-Mails werden nur am Server gespeichert (kann im MUA zu caching zwecken auch gespeichert werden)
  - **Vorteil:** Die Mailbox hat auf mehreren Rechnern immer den gleichen Stand
  - **Nachteil:** IMAP4 ist ein komplexeres Protokoll als POP3, daher ist die Implementation für Entwickler schwieriger.
  - **Nachteil:** Erfordert ohne modernen MUA eine permanente Internetverbindung

DNS-Server

- Übersetzung per DNS-Abfrage: **IP-Adresse** → **Hostname** (host command)
- Ermitteln von Ziel-Servern (A, Cname, MX, …)
- Mögliche Daemons
  - Bind (stable: **9.7.0**, Debian: **9.5.1.dfsg.P3-1+lenny1**)
  - Djbdns (stable: **1.05**, Debian: **1.05-4+lenny1**)
  - Dnsmasq (stable: **2.52**, Debian: **2.45-1+lenny1**)
  - NSCD (nur Lokal)

Datenbank-Server

- Speicherung von Daten in Datenbanken und Tabellen
- Abfragbar mit Queries

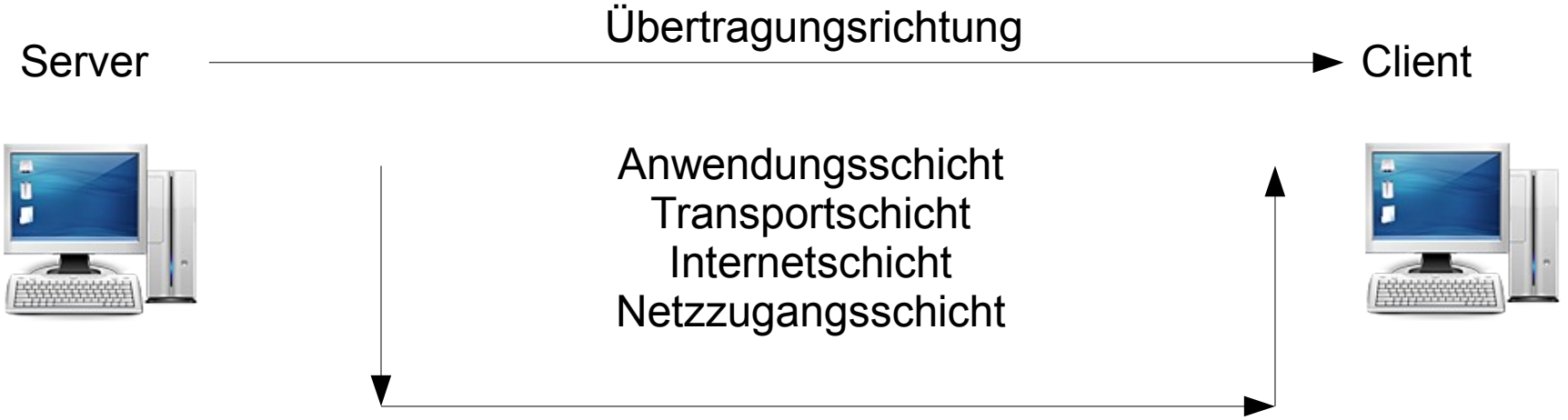
- Mögliche Daemons
  - MySQL (stable: **5.5**, Debian: **5.1.45-1~bpo50+1**)
  - PostgreSQL (stable: **8.4.3**, Debian: **8.3.9-0lenny1**)
  - MariaDB (stable: **5.1.44b**, Debian: kein .deb)
  - Weiters: MS SQL-Server, sqlite, Oracle, DB2 ...

Protokolle (TCP/IP)

- Protokolle dienen zur Kommunikation zwischen 2 Geräten
- Protokolle definieren Abläufe und Verhaltensmuster damit eine Kommunikation stattfinden kann.
- Protokolle sind in RFC's (request for comments) definiert

TCP/IP Referenzmodell (DoD)

TCP/IP-Schicht	≈ OSI-Schicht	Beispiel
Anwendungsschicht	5–7	HTTP, FTP, SMTP, POP, Telnet
Transportschicht	4	TCP, UDP, SCTP
Internetschicht	3	IP (IPv4,IPv6)
Netzzugangsschicht	1–2	Ethernet, Token Bus, Token Ring, FDDI

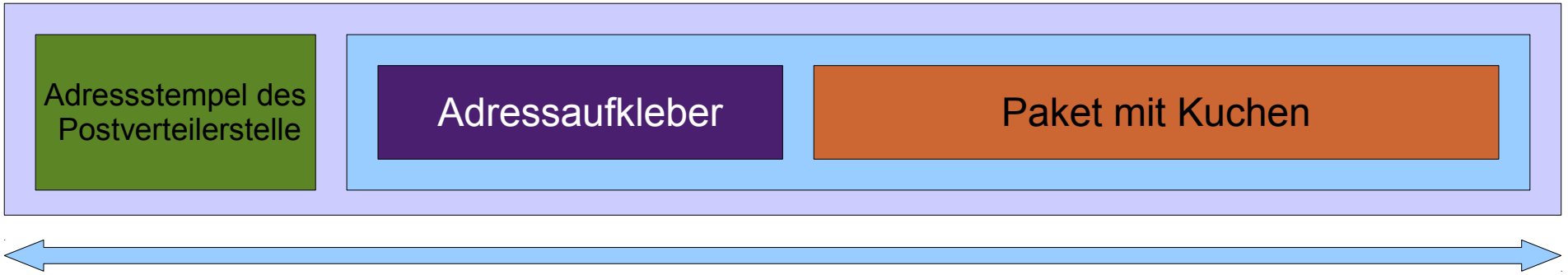


Header / Kapselung

- Bei der Übergabe zwischen den Schichten werden die Datenpakete immer neu verpackt und neue Header-Infos hinzugefügt.

Abstraktes BSP einer Kapselung

Ein Paket vom Kunden



Das Paket im Versand

So weit, so gut

- Gibt es Fragen zu den Serverdiensten oder Tools?
  - Wenn ja, bitte jetzt stellen
- Gibt es Fragen zu den Protokollen?
  - Wenn ja, bald wird es klarer, was das ganze bringt..

Auf zu Teil 2 oder wie man im TYPOlight-Framework sagen würde:

- $this->redirect('part2.html');

Hypertext Transfer Protocol / HTTP

- Spezifiziert in RFC 1945 (1.0) und RFC 2616 (1.1)
- Befindet sich auf der Anwendungsschicht (OSI 5-7)
- HTTP ist ein zustandsloses Protokoll
- Anfragen werden immer im Request/Response Verfahren abgewickelt
- HTTP basiert auf TCP, nicht UDP
- Der Standardport ist 80/TCP
- Entwickelt 1989 von Tim Berners-Lee am CERN

Aufbau des HTTP

- HTTP verwendet immer Anfrage/Antwort Schemas
  - Request / Response
- Die Blöcke werden Messages genannt
  - Message Header
    - Zeichenatz
    - MIME
  - Message Body
    - Der Inhalt der Seite

body != body

- Ganz wichtig ist folgende Differenzierung:
- Der Message-Body im HTTP Protokoll hat nichts mit dem <body> aus HTML zu tun. Der HTML <body> weiß NICHTS vom HTTP Body. Er ist ein Teil davon.


```
<html>
<head><title>Foobar</title></head>
<body>
<h1>Gib SEO kein Chance!!!</h1>
....
```

So funktioniert HTTP (Request)

- Ein Browser will die Webseite http://www.foobar.com/explain-it.html
- Der Client schickt folgende Anfrage an den Server
  - **GET /explain-it.html HTTP/1.1**
  - **HOST: www.foobar.com**
- Optional noch weitere Header-Informationen wie User Agent, Sprache, …
- Eine Leerzeile schließt den Request ab und der Server liefert eine Response zurück.

So funktioniert HTTP (Response)

- Nach der Leerzeile des Requests wird die Antwort gesendet

```
HTTP/1.1 200 OK
Server: Apache/2.2 (Unix, Debian) PHP/5.3.1
Content-Length: 12345
Content-Language: de
Content-Type: text/html
Connection: close

(Inhalt von explain-it.html)
```

1.0 vs. 1.1

- HTTP 1.0 baut nach jeden Response die TCP Verbindung wieder ab.
- HTTP 1.1 kann durch den **Keep-Alive** Header dazu veranlasst werden die Verbindung persistent aufrecht zu halten. Dadurch können mehrere Anfragen über eine TCP/IP Session übertragen werden

Request Methoden

- GET
  - Daten werden in der URL Übertragen: ?foo=bar
- POST
  - Benötigt BSP:
    - **Content-Type: application/x-www-form-urlencoded**
- Weitere: HEAD, PUT, DELETE, ...

Statuscodes

- In HTTP definieren **Status Codes** den Status der Aktion
- Sind in RFC 1945 (1.0) und RFC 2616 (1.1) definiert
- Status Codes bestehen immer aus 3 Nummern
- Der bekannteste: **HTTP/1.1 404 Not Found**

Liste aller wichtigen Statuscodes

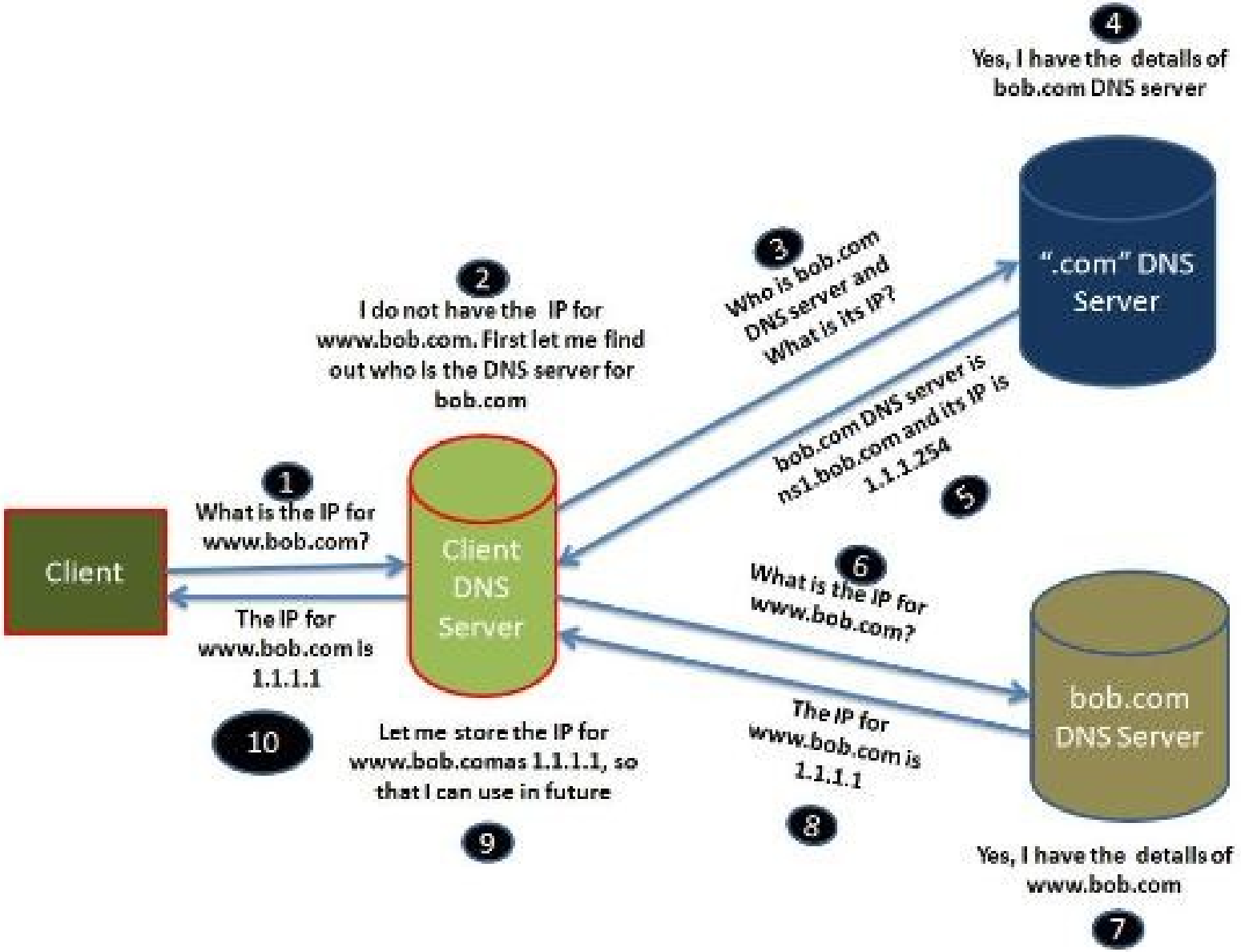
- **1XX:** Bearbeitung der Anfrage dauert noch etwas. Dient dazu, nicht in einen Timeout zu rennen
- **2XX:** Anfrage wurde bearbeitet, Antwort wurde an den Client gesendet
- **3XX:** Hat theoretisch funktioniert, aber der Client muss noch etwas machen damit die Aktion positiv beendet wird. (BSP: File has moved)
- **4XX:** Ein Client-Fehler ist aufgetreten
  - **404:** File not found
  - **403:** Permission denied
- **5XX:** Der Server hat ein Problem, muss vom Admin behoben werden

Beispielabfrage

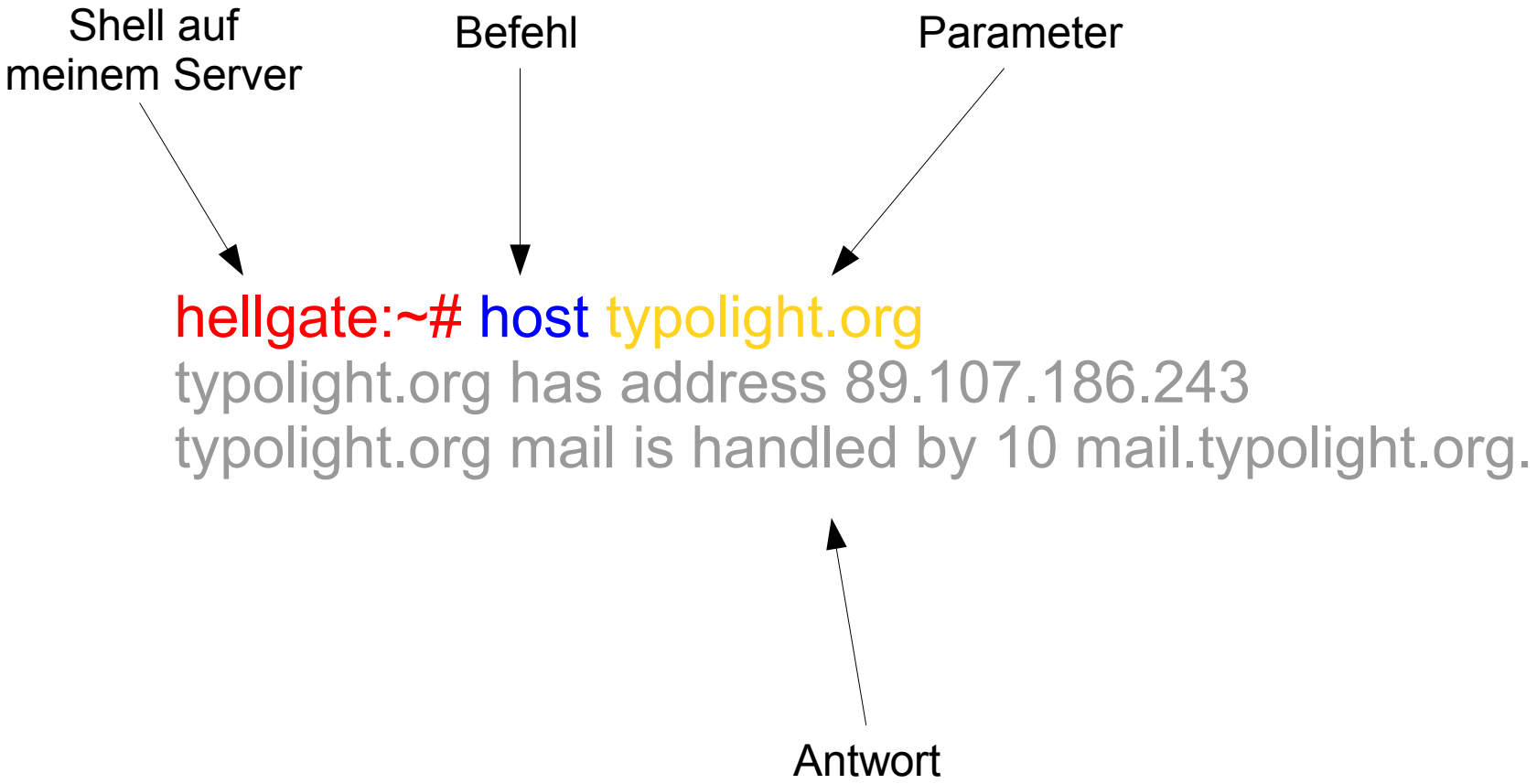
Der User Jack will die Webseite http://www.foo.bar besuchen.

- Er tippt die Domain http://www.foo.bar in den Browser ein.
- Der Browser löst die Domain per DNS Abfrage auf.

DNS-Lookup (schema)

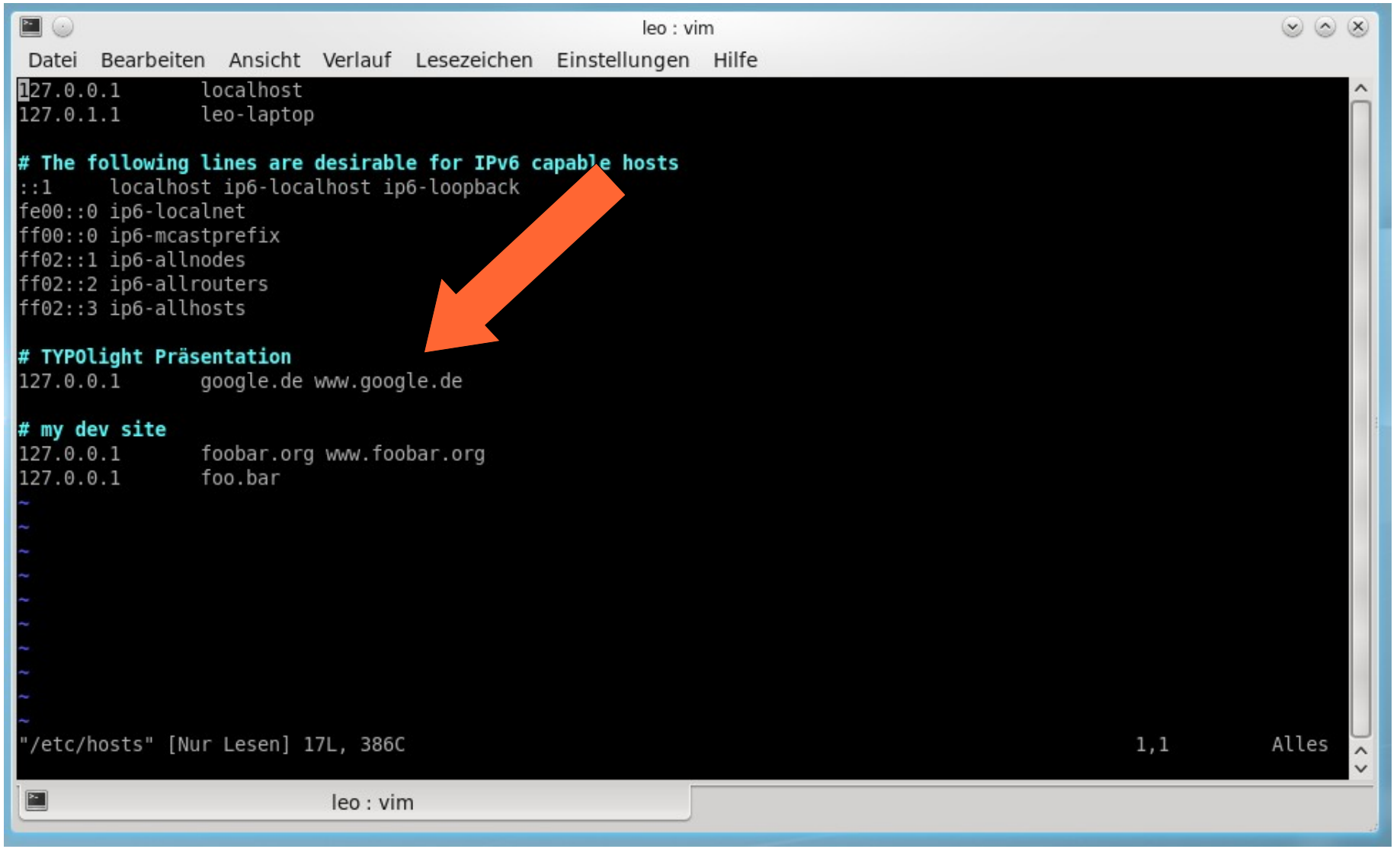


Manueller DNS Lookup auf der Bash



- Nachdem der Browser nun die IP des Webservers hat wird ein HTTP Request an die IP des Webservers auf Port 80 geschickt.
- Mitgeschickt wird auch die Header-Zeile HOST **foo.bar**
- Einzig anhand des Header Parameters HOST entscheidet der Webserver welche Webseite ausgeliefert werden soll.
- Nach abschließender Leerzeile im HTTP Request wird der HTTP Response zurückgeliefert.
- Dieser beinhaltet nun die Webseite welche Jack so dringend anschauen wollte.

Erklärung: der Google Trick



Done – das wars ...

Fragen?

Wenn nicht, geht’s mit einem HTTP 100 in die Pause.

Danke für Eure Aufmerksamkeit ....