Hochschule
Albstadt-Sigmaringen

Albstadt-Sigmaringen University

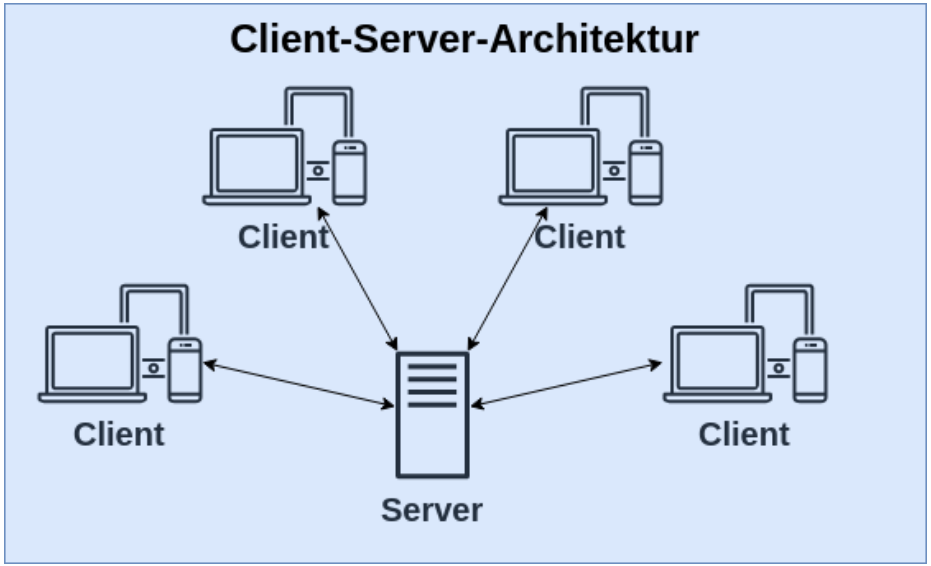
HTTP 1 / 1.1 / 2

Dipl. Ing. Sven Eppler (FH)
sodge IT GmbH

Client/Server Request/Response

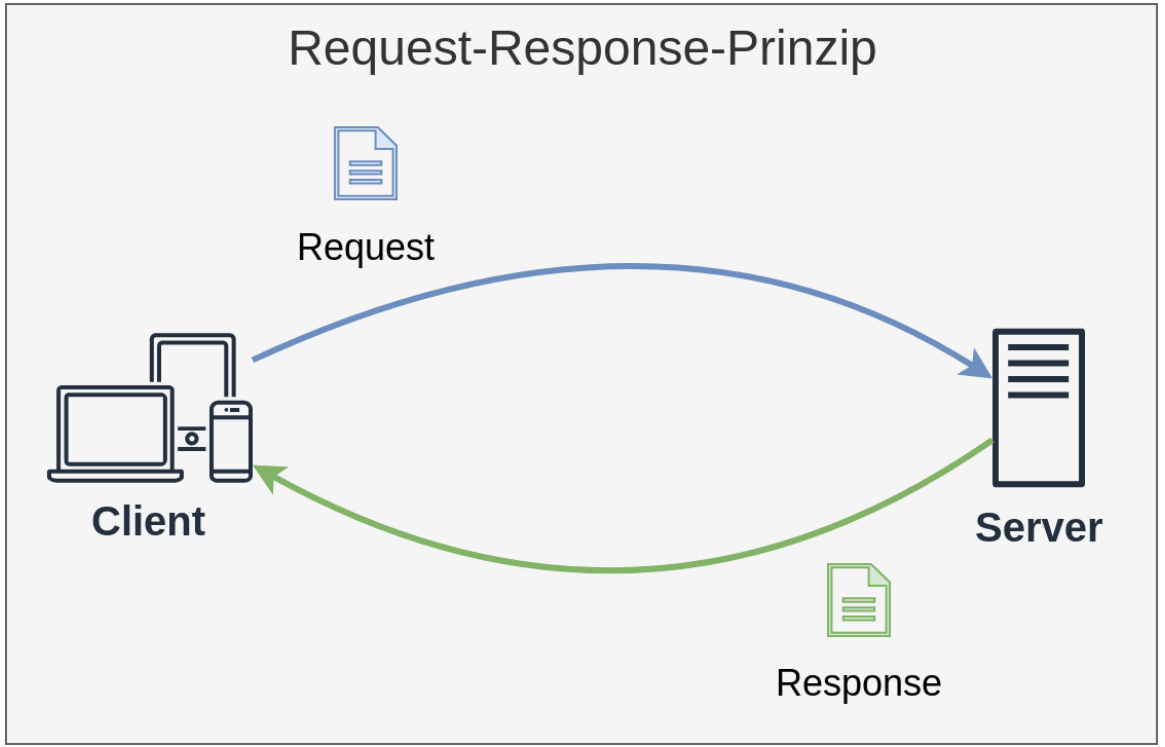
Client-Server-Architektur

- Server: Stellt Dienst bereit
- Client: Konsumiert Dienst
- Client baut Verbindung auf
- Prinzipbedingt Zentralisiert
- Server ist „passiv“
- Client ist „aktiv“
- Beispiele: HTTP, FTP, IRC, SMTP, SSH



Request-Response-Prinzip

- **Ein Client stellt eine Anfrage (Request)**
- **Der Server Antwortet auf die Anfrage (Response)**



HTTP Basics

HTTP Basics

- HyperText Transport Protocol
- 1989, Tim Berners-Lee am CERN
- Basiert auf TCP/IP
- OSI-Schichtenmodell Schicht 7 Anwendungsschicht
- Plain-Text Protokoll
- HTTP ist stateless (zustandslos)
- Client-Server / Request-Response Konzept

HTTP Basics

- **HTTP Message besteht aus**
  - Request-Line | Status-Line
    (Unterscheidet zwischen Request und Response)
  - HTTP Headers (Kopfzeilen, Meta-Daten)
    Zeilenende: <CRLF> (0x0D 0x0A; \r\n)
  - HTTP Body (Körper, Inhalt)

HTTP Request Line & Status Line

- **Request Line (rfc2616)**
  - `Request-Line = Method SP RequestURI SP HTTPVersion CRLF`

    Beispiel:
    `GET /index.html HTTP/1.1<CRLF>`
- **Status Line (rfc2616)**
  - `Status-Line = HTTPVersion SP StatusCode SP ReasonPhrase CRLF`

    Beispiel:
    `HTTP/1.1 200 OK`

- **RFC: https://www.ietf.org/rfc/rfc2616.txt**

Request HTTP Methods

- **GET**
  - Abrufen einer Resource
- **POST**
  - Lädt Daten zum Server
- **PUT**
  - Erzeugt eine Resource
- **DELETE**
  - Löscht eine Resource
- **PATCH**
  - Ändert eine Resource

Request HTTP Methods

- **HEAD**
  - Ruft Metadaten für eine Resource ab
- **TRACE**
  - Spiegelt die Anfrage
- **OPTIONS**
  - Unterstützte Methoden des Server (wichtig für CORS)
- **CONNECT**
  - Der Proxy-Server soll zum Ziel durchstellen (SSL-PassThrough)

Response HTTP Status Codes

- **1XX – Informationen**
  - 101 Switching Protocols
- **2XX – Erfolg**
  - 200 OK
- **3XX – Umleitung**
  - 301 Moved Permanently
- **4XX – Client Fehler**
  - 400 Bad Request
- **5xx – Server Fehler**
  - 500 Internal Server Error
- **9xx – Proprietäre Fehler**

HTTP Header

- **Schlüssel-Wert Paare die die HTTP-Message beschreiben**
  - `Header-Name: Header-Value <CRLF>`
- **Header dürfen wiederholt werden**
- **Reihenfolge irrelevant**
- **Lange Header müssen kompliziert umgebrochen und eingerückt werden**
- **Es gibt Request- und Response-Header aber auch Header die in beiden Fällen zum Einsatz kommen**
- **Der Header endet mit einem zusätzlichen `<CRLF>` (`0x0D 0x0A`)**

HTTP Header

- **Wichtige HTTP-Header:**
  - content-type: Beschreibt die Art des HTTP-Body (z.B. text/plain, text/html, application/json, video/mp4, image/jpeg)
  - host: Beschreibt welcher Hostname ursprünglich angefragt wurde
  - content-length: Wieviele Bytes ist der Content lang
  - last-modified: Wann wurde die angefragte Resource zuletzt geändert
  - set-cookie: Zum setzen von Cookies beim Client
  - cookie: Zum zurück senden eines Cookies zum Server

HTTP-Body

- **Der Body folgt auf den Header, <CRLF><CRLF>**
- **Nicht jeder Request/Response hat einen Body**
  - GET Request
  - 404 Not Found Response
- **Transportiert die Nutzdaten (Payload)**
  - Request: z.B. Formulardaten, JPEG-Upload
  - Response: HTML-Seite, JPEG-Bild, JSON
- **Wichtig: Content-Length Header und tatsächliche länge des Body *sollten* stimmen!**

HTTP-Request

- **Einfache HTTP-Request Beispiele**
  Header = Grün, Body = Blau

```
GET / HTTP/1.1<CRLF>
Host: heise.de<CRLF>
<CRLF>
```

```
POST / HTTP/1.1<CRLF>
Host: localhost:3000<CRLF>
Content-Type: application/json<CRLF>
Content-Length: 18<CRLF>
<CRLF>
{ "some": "JSON" }
```

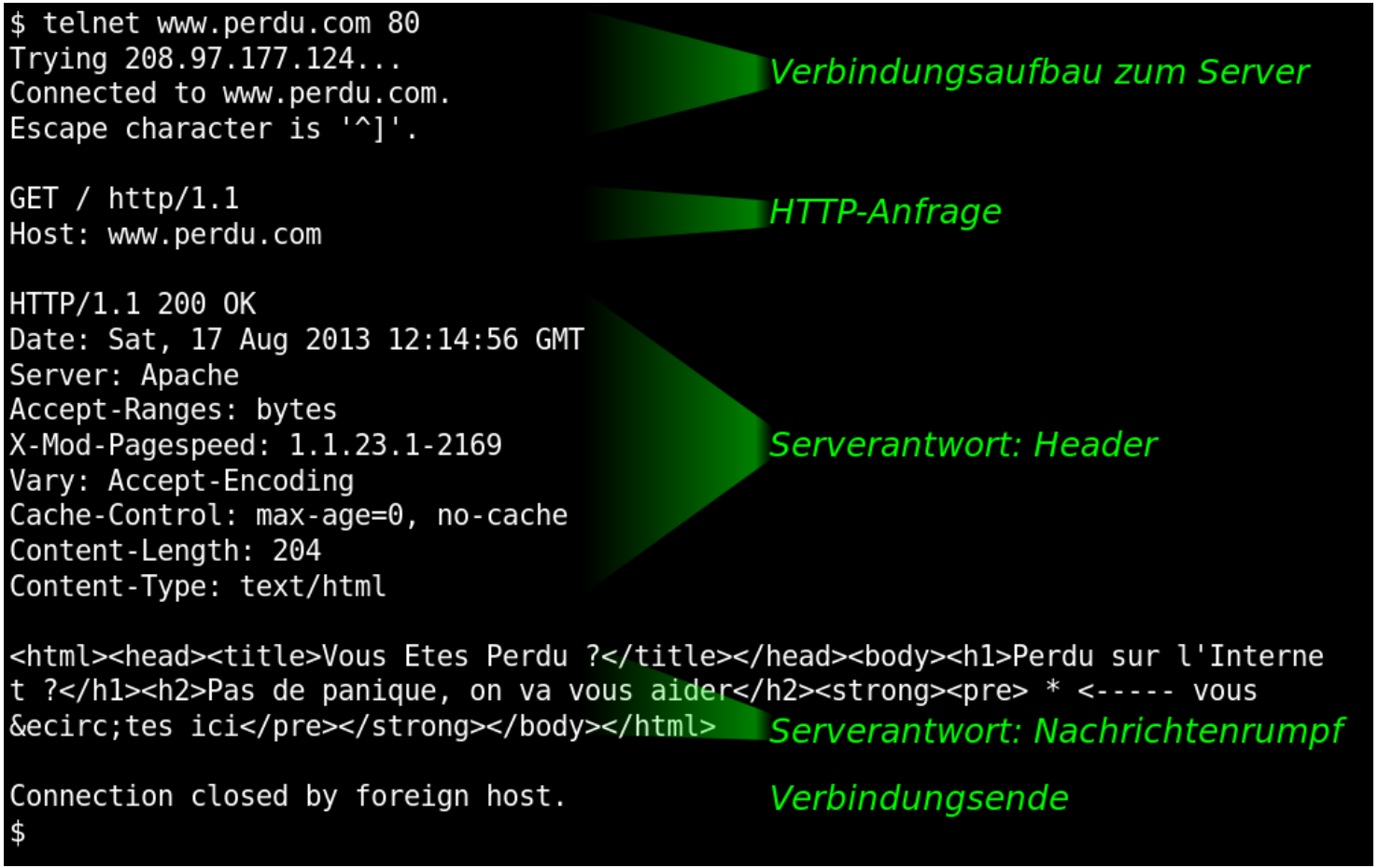
HTTP-Response

- **Einfache HTTP-Response Beispiele**
  Header = Grün, Body = Blau

```
HTTP/1.1 200 OK<CRLF>
Content-Length: 11<CRLF>
Content-Type: text/plain<CRLF>
<CRLF>
Hallo Welt!
```

```
HTTP/1.1 404 Not Found<CRLF>
Date: Mon, 25 Mar 2019 19:41:24 GMT<CRLF>
Server: Apache/2.4.29 (Ubuntu)<CRLF>
Content-Length: 303<CRLF>
Content-Type: text/html; charset=iso-8859-1<CRLF>
<CRLF>
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
Antwort Gekürzt
</body></html>
```

Sprechen Sie HTTP?



Quelle: https://commons.wikimedia.org/wiki/File:HTTP-Anfrage.svg

HTTP 1 / 1.1 / 2

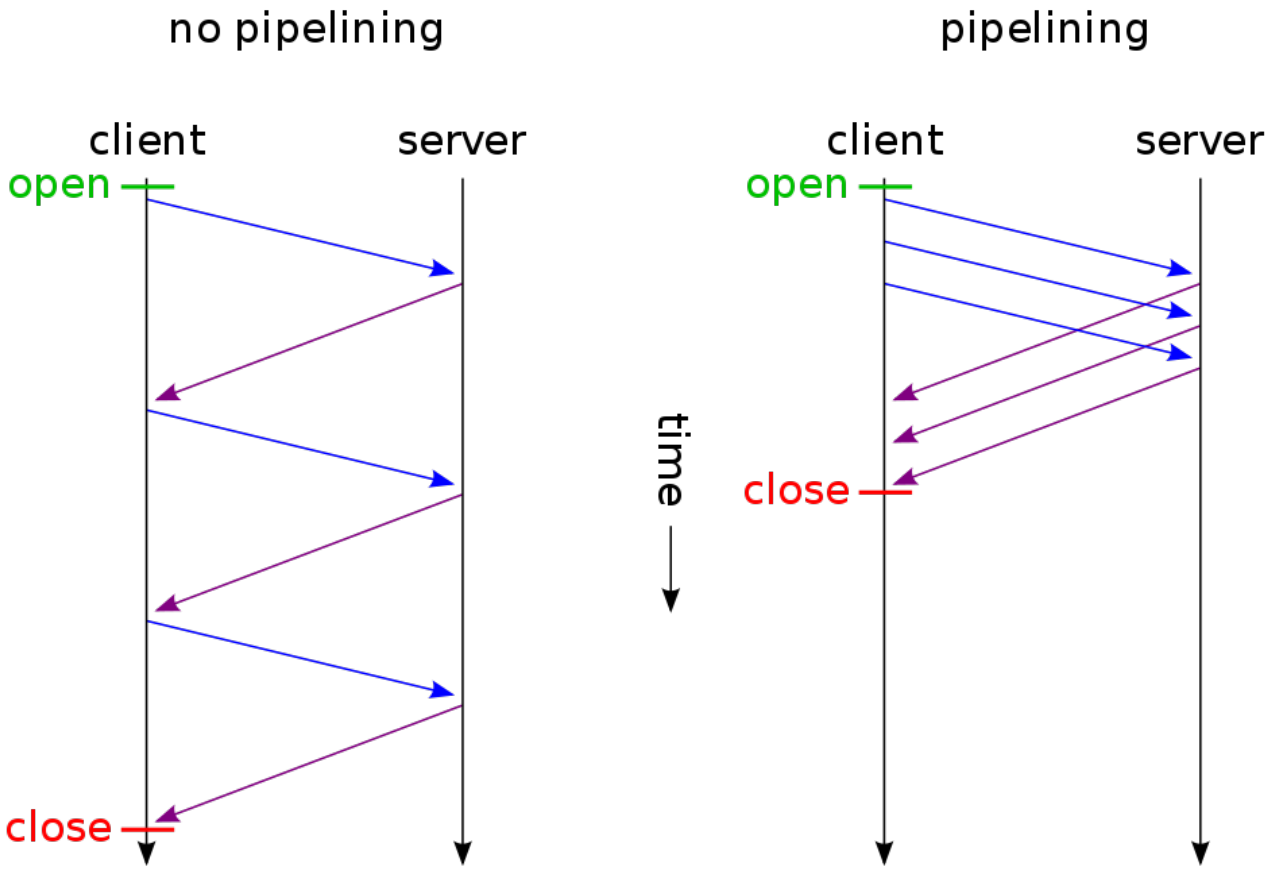
Besonderheiten HTTP1.0

- **Jeder Request eine TCP/IP-Connection**
  - Hoher TCP/IP Overhead wegen Slow-Start-Algorithmus (Congestion Window)
- **Server schließt Verbindung nach jeder Response**

Besonderheiten 1.1

- **Keep Alive**
  - TCP Verbindung kann nach Response weiter verwendet werden
    `Connection: keep-alive`
    `Connection: close` (HTTP1.0 Verhalten)
- **HTTP-Pipelining**
  - Der Client kann mehrere Requests gleichzeitig schicken, der Server antwortet dann in der selben Reihenfolge mit allen Responses (first-in-first-out)

Besonderheiten 1.1 / HTTP Piplining



Quelle: https://commons.wikimedia.org/wiki/File:HTTP_pipelining2.svg

Besonderheiten HTTP2

- Connection-Multiplexing (Streams)
- Komprimierbarkeit
- Binäre Datenübertragung (bye bye MIME-Bodies)
- Push-Messages (Server to Client)
- Prioritisierung von Streams
- TLS-Only (leider optional, aber von Google und Mozilla forciert)

HTTP Debugging

HTTP Debugging

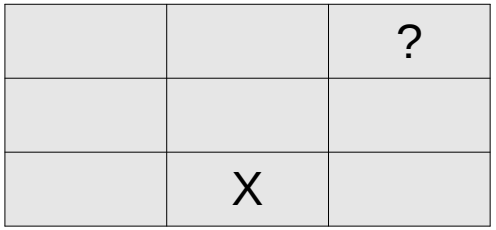
- Browser Cache ausschalten!
- Developer Console
- Server Logging
- Curl
- Wget
- Postman (getpostman.com)
- Insomnia (insomnia.rest)
- Telnet
- Wireshark

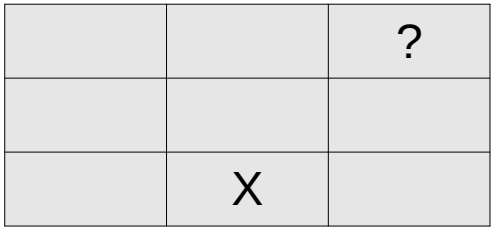
Stateless / Statefull

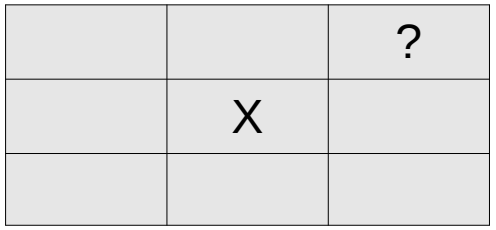
Stateless vs. Statefull

- **Stateless**
  - Jede Anfrage wird behandelt als wäre es die erste und einzige
  - Die selbe Anfrage liefert auch immer das selbe Ergebnis
- **Statefull**
  - Jede Anfrage verändert einen Zustand (State)
  - In Abhängigkeit vom Zustand liefert die selbe Anfrage, ein anderes Ergebnis

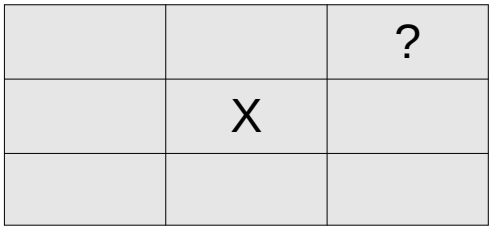
Stateless vs. Statefull Beispiel

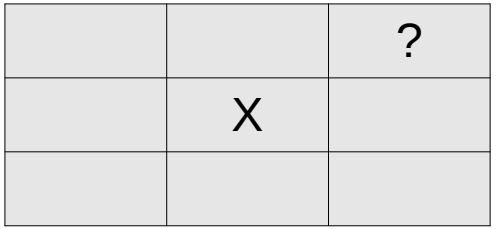




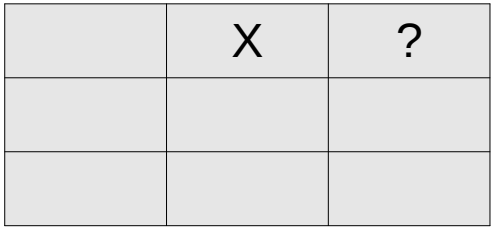


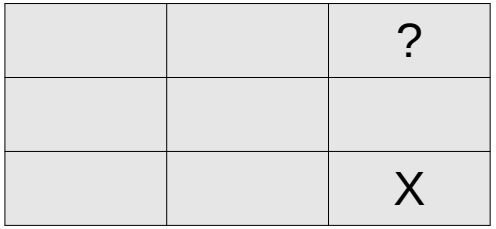
Go Up



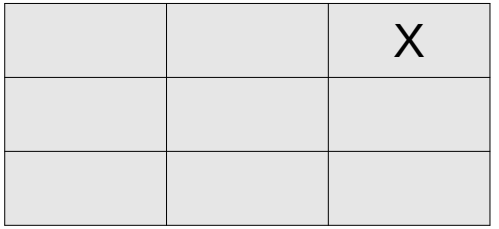


Go Up





Go Right



Stateless Implikationen

- **Wird in einem zustandlosen System ein Zustand benötigt**
  - Muss der gesamt Zustand transportiert werden
    `(Go Up + Go Up + Go Right)`
  - Oder der Zustand auf dem Server gespeichert werden
    - `GameId=987 Go Up`
    - `GameId=987 Go Up`
    - `GameId=987 Go Right`
- **HTTP Bietet für diesen Zweck Cookies**

HTTP Cookies

- **Geringe Datenmenge die vom Server via HTTP-Header gesetzt wird**
- **Cookies werden automatisch vom Browser beim nächsten Request zurück geschickt**
- **Cookies können auch mit JavaScript auf dem Client angelegt werden**
- **Ermöglichen das „tracken“ eines Users**
  - Login
  - Werbung
  - Überwachung

HTTP Cookies

- **Server setzt Cookie mit GameID**
  - set-cookie: gameID=987
- **Client schickt Go-Befehl**
  Browser setzt Cookie in den Request-Headern
  - `cookie: gameID=987`
    `Go Up`
  - `cookie: gameID=987`
    `Go Up`
  - `cookie: gameID=987`
    `Go Right`

HTTP Cookies

- **Cookies sollten keine Vertrauenswürdigen Daten enthalten**
- **Es gibt spezielle `HttpOnly` Cookies die nicht via JavaScript auslesbar/änderbar sind**
- **Cookies können als `Secure` markiert werden, dürfen dann nur per HTTPS zurück zum Server**
- **Cookies können einfach manipuliert werden**
  - Developer Console der Browser
  - HMAC Signed Session Cookies