Web Engineering + Design 1

HTTP

Silvan Gehrig

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Lernziele

Die Vorlesung soll die Teilnehmer befähigen, HTTP Request und Response im Grundsatz und mit deren wesentlichen Eigenschaften zu verstehen und dessen Konzepte fachgerecht einzusetzen.

Die Teilnehmer...

- **verstehen, wie ein Web-Request zustande kommt und kennen dessen Ablauf.**
- **können HTTP Protokoll-Probleme zwischen Client und Server interpretieren und deren Fehlerquelle einschätzen.**
- **kennen einige der wichtigsten HTTP Header mit deren Funktionalitäten.**

Table of Contents

- **Hypertext Transfer Protocol – HTTP**
  - URI Schema
  - Request / Response
  - Headers
  - Session Management

HYPERTEXT TRANSFER PROTOCOL

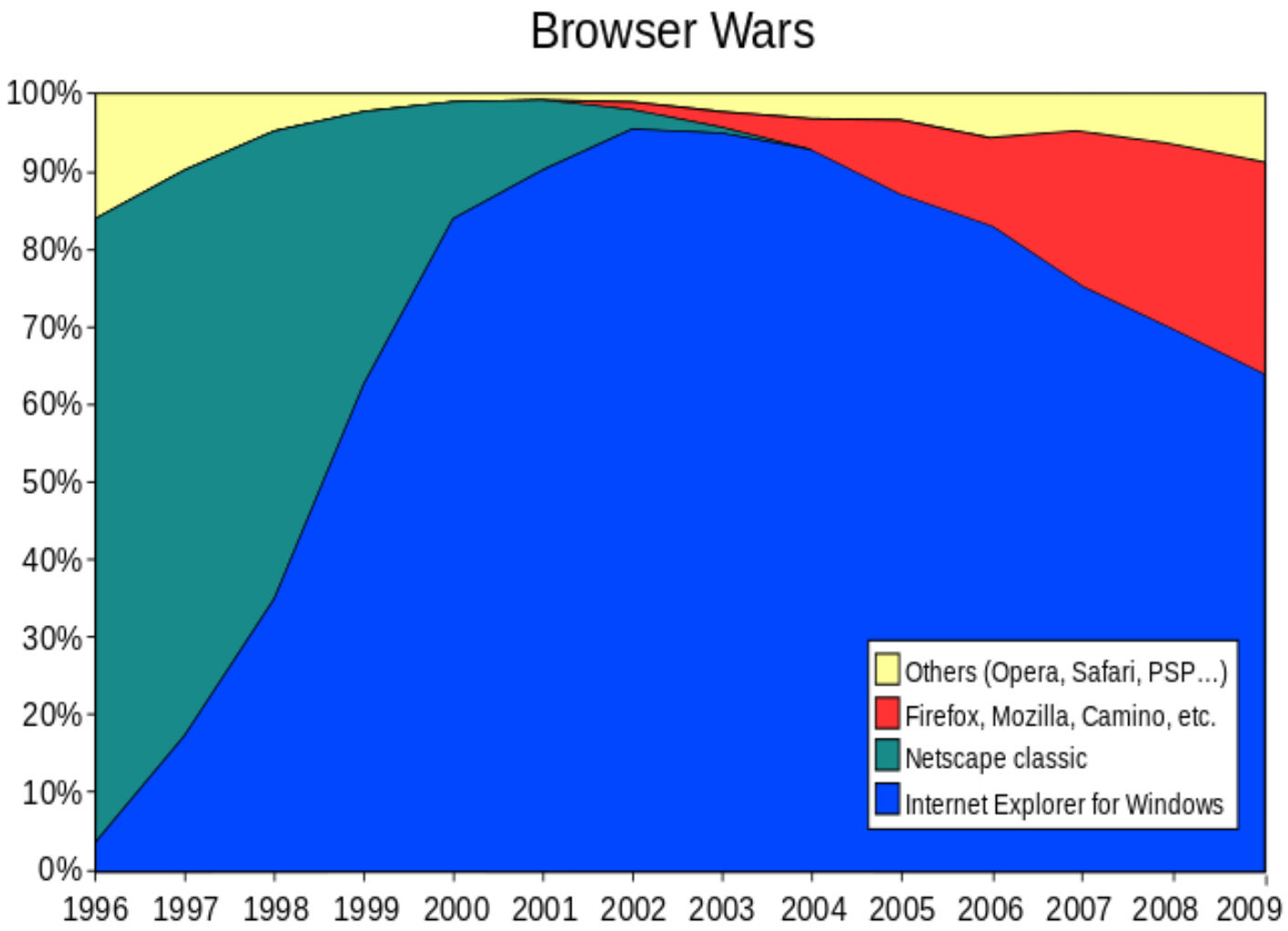
INTRODUCTION

HTTP

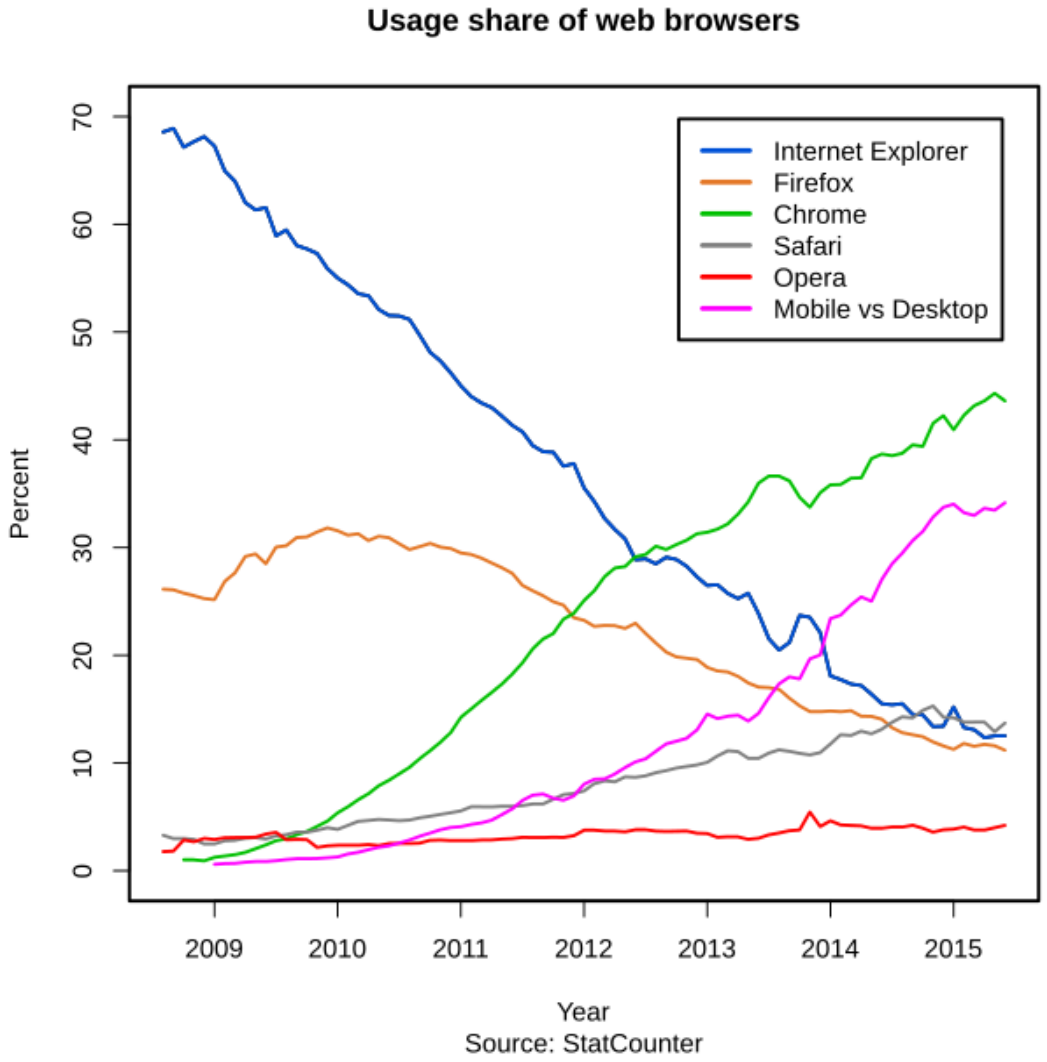
HTTP History

Year	Comment	
1962	ArpaNet	
as of 1981	TCP / IP as network protocol (→ Internet) Layer 7 Protocols: POP, SMTP, FTP, News etc.	
as of 1989	Development of HTTP at CERN Tim Berners-Lee	
1991	HTTP/0.9 Proprietary implementation Netscape, IE	
1996	HTTP/1.0 as RFC1945 der IETF	*First Browser War*
1999	HTTP/1.1 as RFC 2616 / 2617 der IETF	
2008		*Second Browser War*
2014	HTTP/1.1 as RFC 7230-7235	
ab 2012	SPDY as predecessor of HTTP/2.0	
ab 2015	HTTP/2.0 as RFC 7540	

First Browser War 1996-2009



Second Browser War 2009-2015



DEMO INTRO

Client / Server Architecture

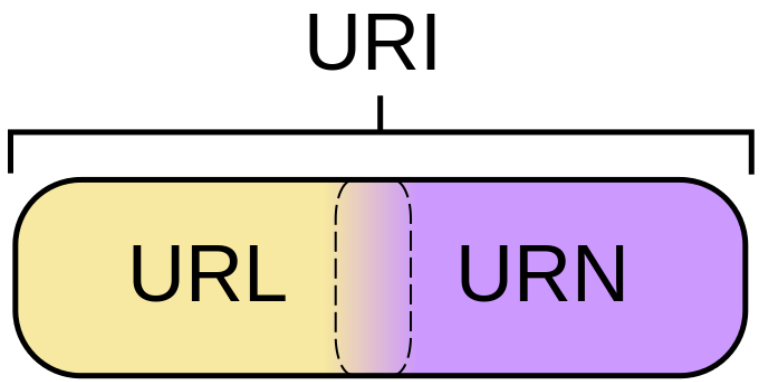
- **Client**
  - Workstation / PCs / …
  - Rely on server resources
- **Server**
  - (Powerful) Computers
  - Dedicated to mangage shared resources

URI SCHEMA

HTTP

HTTP URI

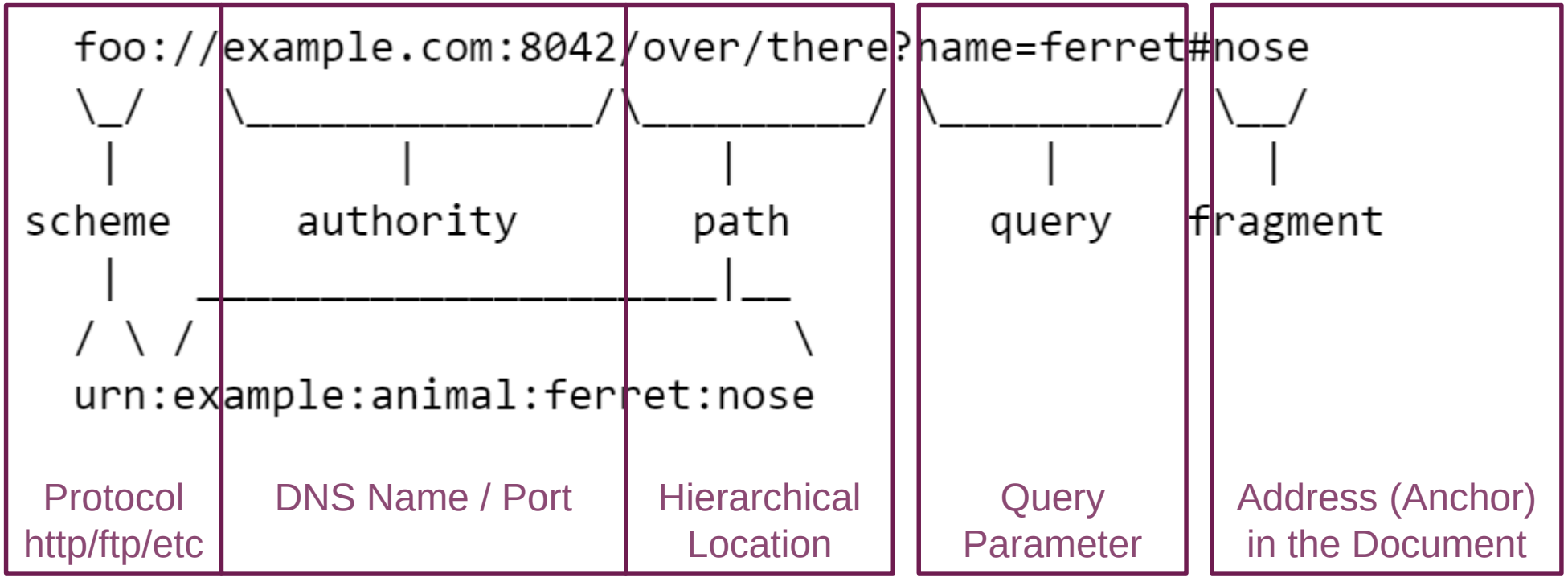
- **URI = Unified Resource Identifier (URI)**
  - is a string of characters used to identify a name of a resource
  - URL and URN are URIs
- **URL = Unified Resource Locator (URL)**
  - is a reference to a resource that specifies the location of the resource on a computer network and a mechanism for retrieving it
  - URLs may contain a URN
  - Limited schema range
- **URN = Unified Resource Name (URN)**
  - is a string of characters used to identify a name of a resource.
  - Independent from location / persistence / …
  - Limited to schema “URN”



URL Schema

- **<scheme name> : <hierarchical part> [? <query>] [# <fragment>]**

- **Example:**



URI Encoding

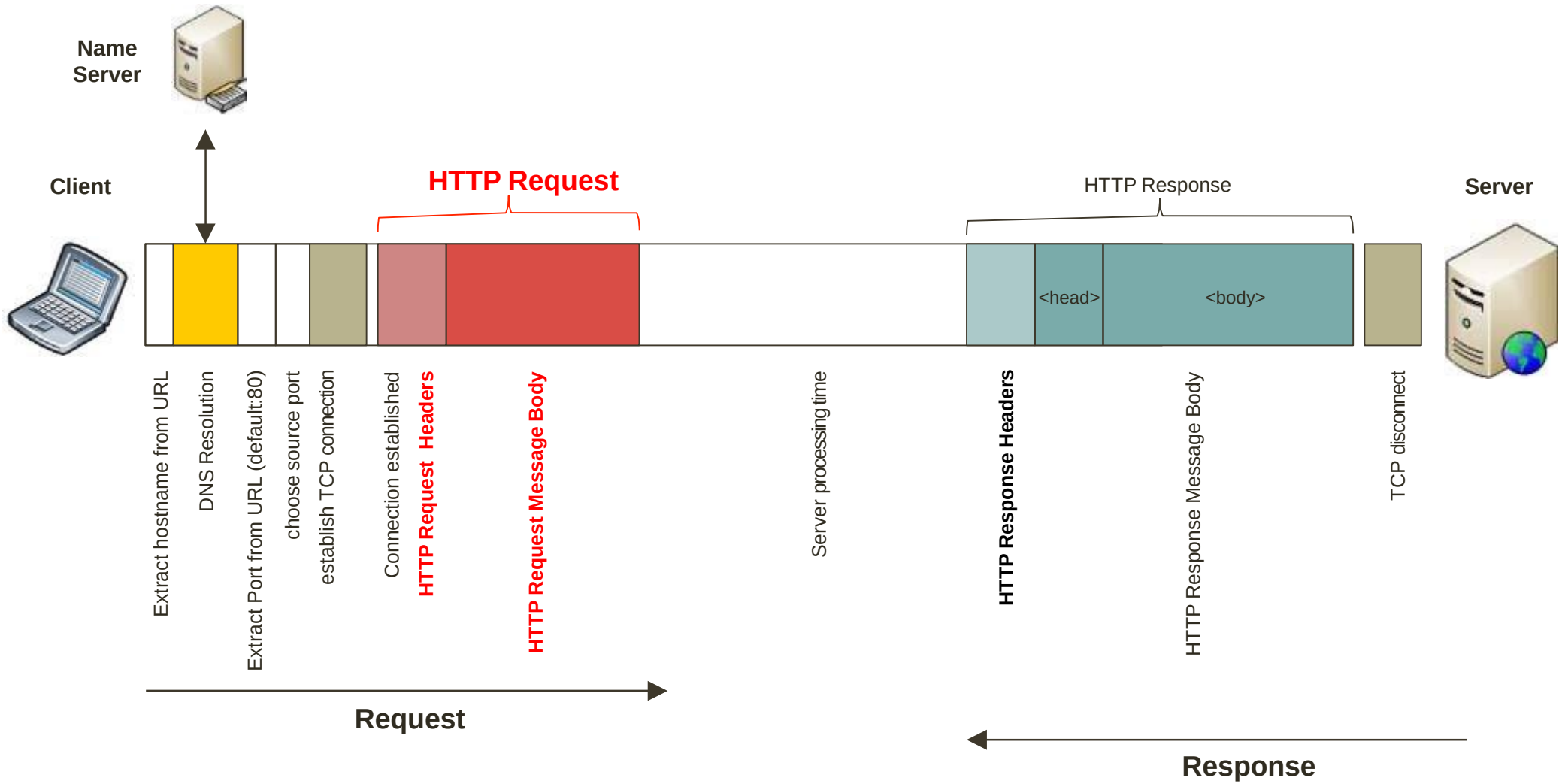
- **Special Characters must be encoded**
- **Important for control characters**

- **Example:**
  - ‘/’ must be converted %2F
  - ‘ ‘ (space) %20
  - ‘&’ %26
  - ‘#’ %23
  - ‘?’ %3F

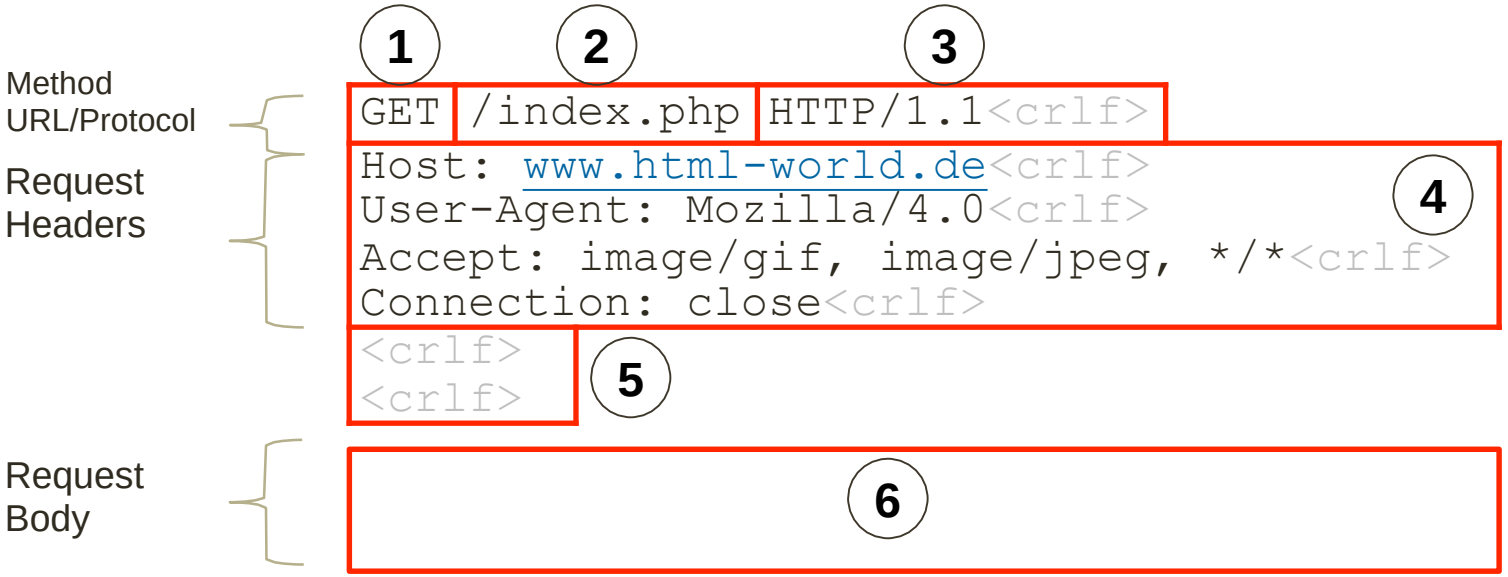
REQUEST

HTTP

Establish an HTTP Connection



HTTP Request



1. Method
2. Address (URL Path)
3. Protocol Version (HTTP/0.9 | HTTP/1.0 | HTTP/1.1. | HTTP/2.0)
4. Request Headers (optional, depends on the application)
5. Header/Body Separator (2 x crlf)
6. Request Body (used for example in <form> Data Transmitions)

HTTP Methods

Method	Description
GET	The GET method is used to **retrieve information** from the given server using a given URI. Requests using GET should only retrieve data and should have **no other effect on the data**.
HEAD	Same as GET, but transfers the **status line and header section** only.
POST	A POST request is used to **send data to the server**, for example, **new** customer information, *file upload*, etc. using *HTML forms*.
PUT	**Replaces all current representations** of the target resource with the uploaded content.
DELETE	**Removes all current representations** of the target resource given by a URI.
CONNECT	Establishes a tunnel to the server identified by a given URI.
OPTION	Describes the communication options for the target resource.
TRACE	Performs a message loop-back test along the path to the target resource.

HTTP GET vs POST Methods

- **GET data is placed into the URL (as query string)**

```
GET /search?query=abc HTTP/1.1<crlf>
Host: www.html-world.de<crlf>
User-Agent: Mozilla/4.0<crlf>
Accept: image/gif, image/jpeg, */*<crlf>
Connection: close<crlf>
<crlf>
<crlf>
```

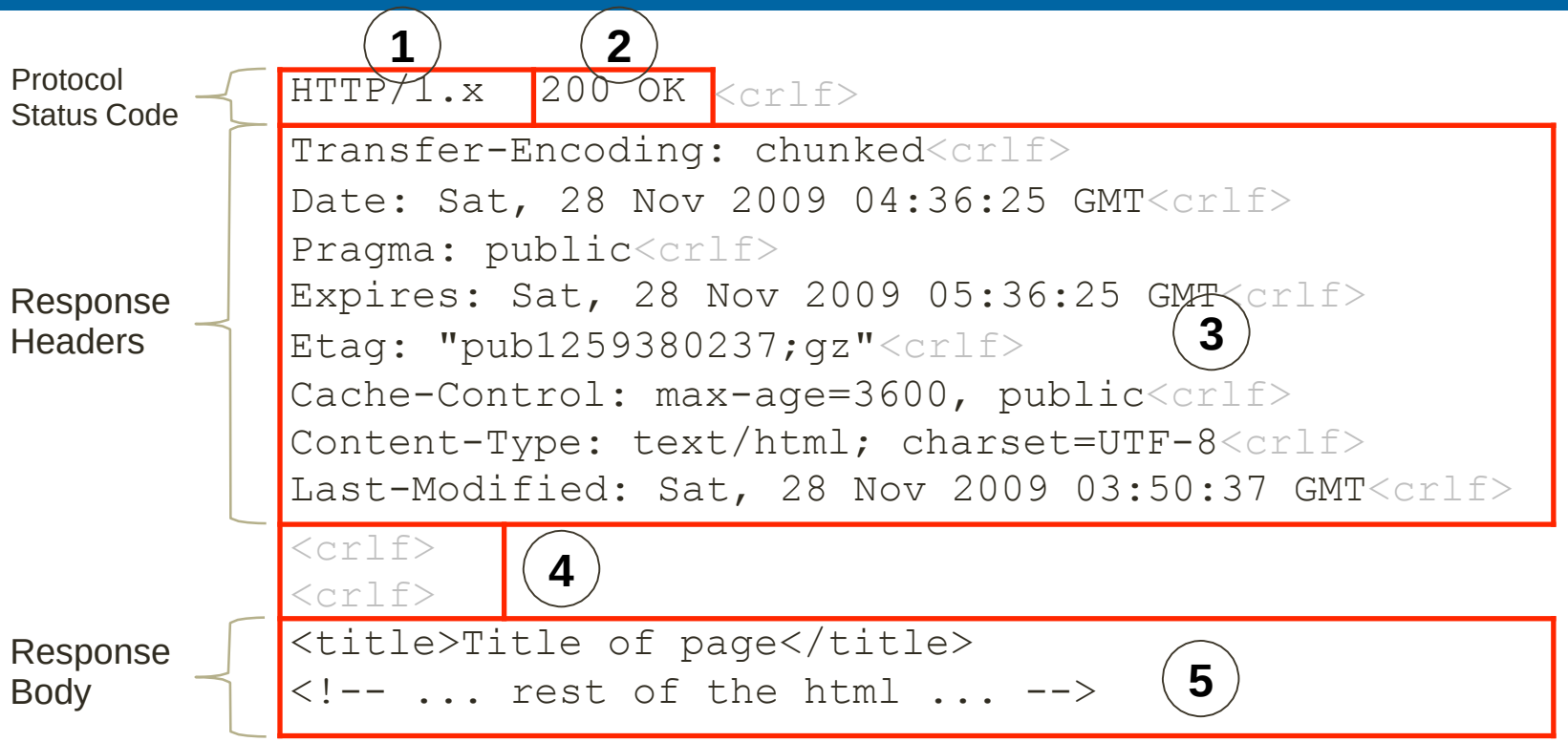
- **POST data is placed into the request body**

```
POST /search HTTP/1.1<crlf>
Host: www.html-world.de<crlf>
User-Agent: Mozilla/4.0<crlf>
Accept: image/gif, image/jpeg, */*<crlf>
Connection: close<crlf>
<crlf>
<crlf>
query=abc
```

RESPONSE

HTTP

HTTP Response



1. Protocol Version (HTTP/0.9 | HTTP/1.0 | HTTP/1.1. | HTTP/2.0)
2. Status Code
3. Response Headers (optional, depends on application)
4. Header/Body Separator (2 x crlf)
5. Response Body (The returning HTML Code)

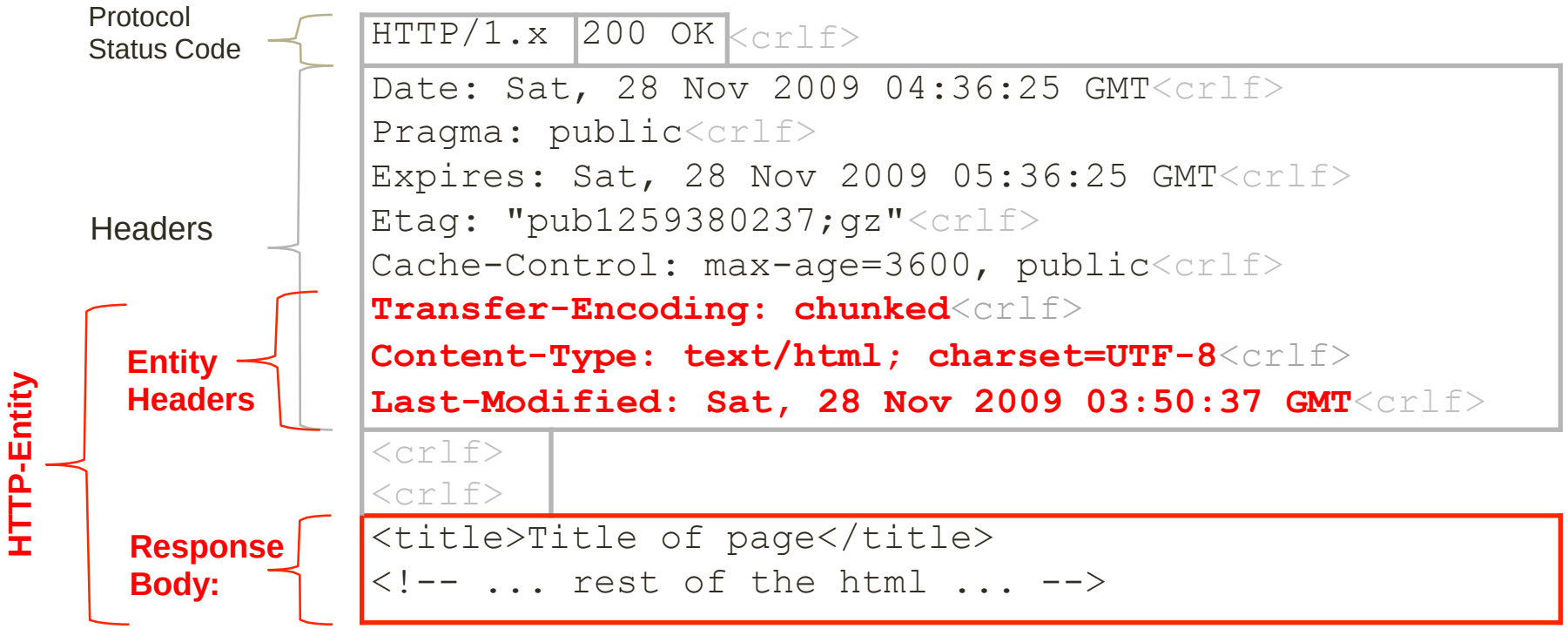
HTTP Response Status Codes

CODE	
1xx	Informational
2xx	Successful 200 OK 201 Created 204 No Content
3xx	Redirection 301 Moved Permanently
4xx	Client Error 400 Bad Request 401 Unauthorized 403 Forbidden 404 Not Found
5xx	Server Error 500 Internal Server Error 505 HTTP Version Not Supported
9xx	Non-Standard Codes

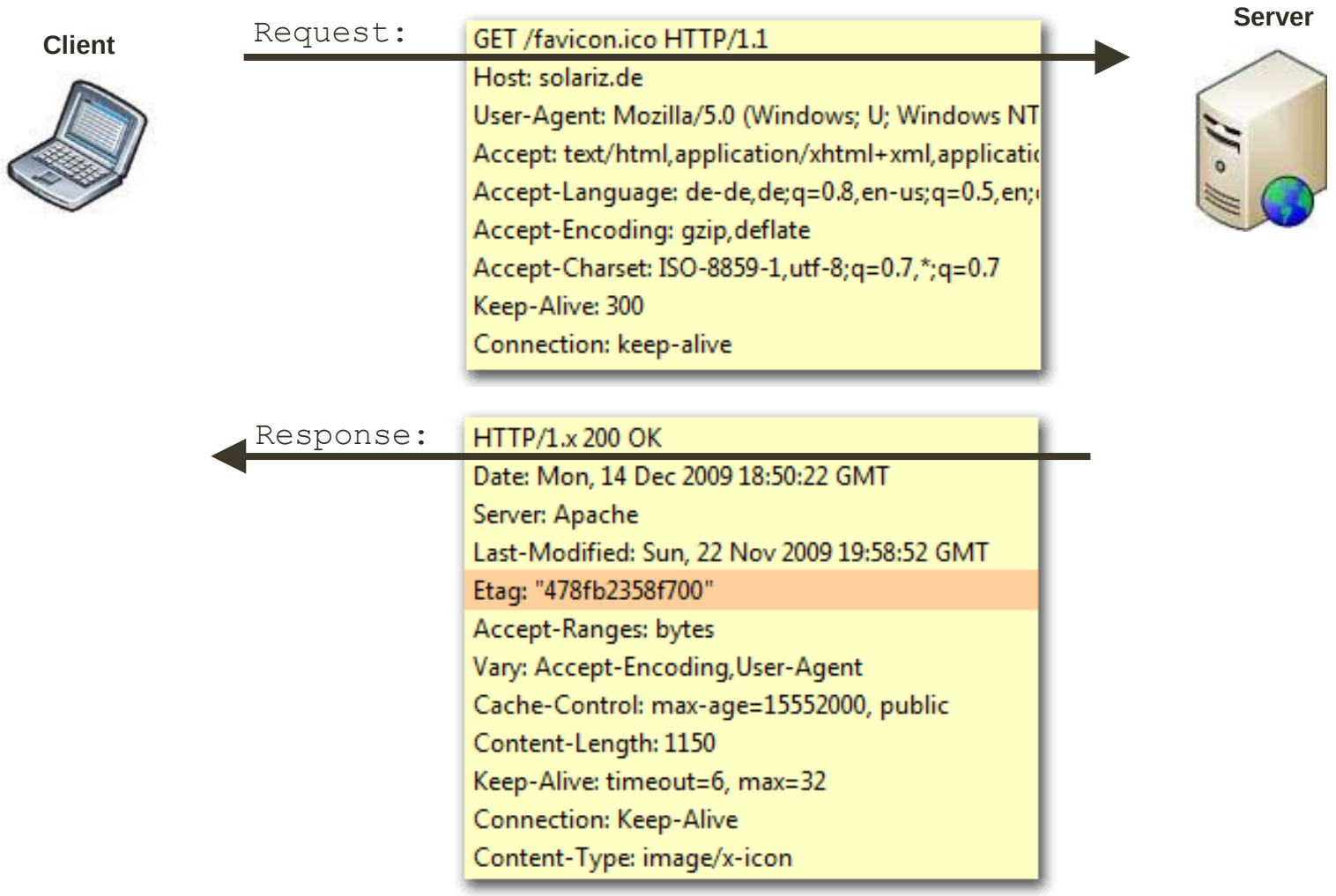
HEADERS

HTTP

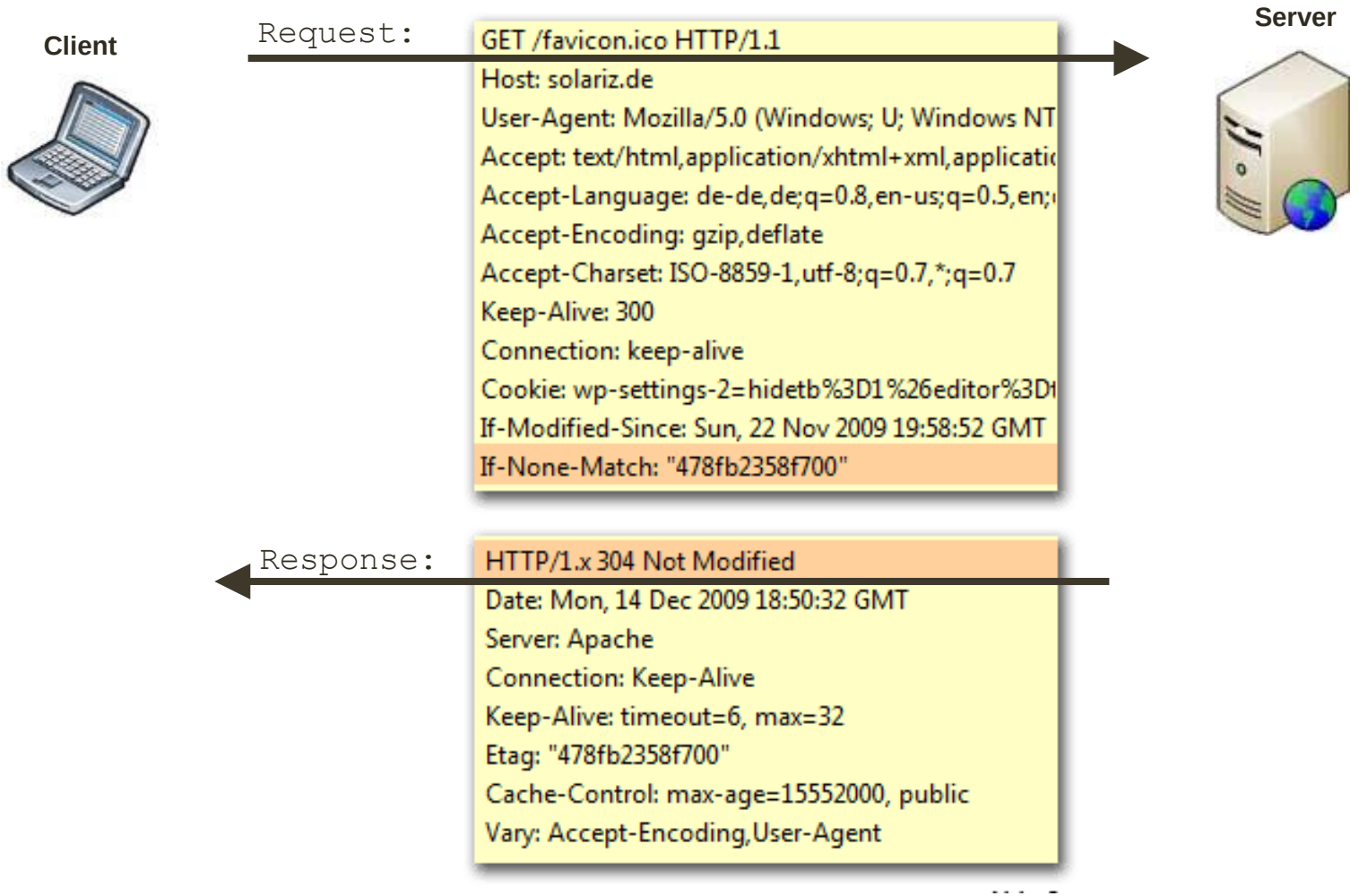
Entity (Response)



Response handling with Etag 1/2



Response handling with Etag 2/2



Headers Overview (Request & Response)

Request

Request Headers

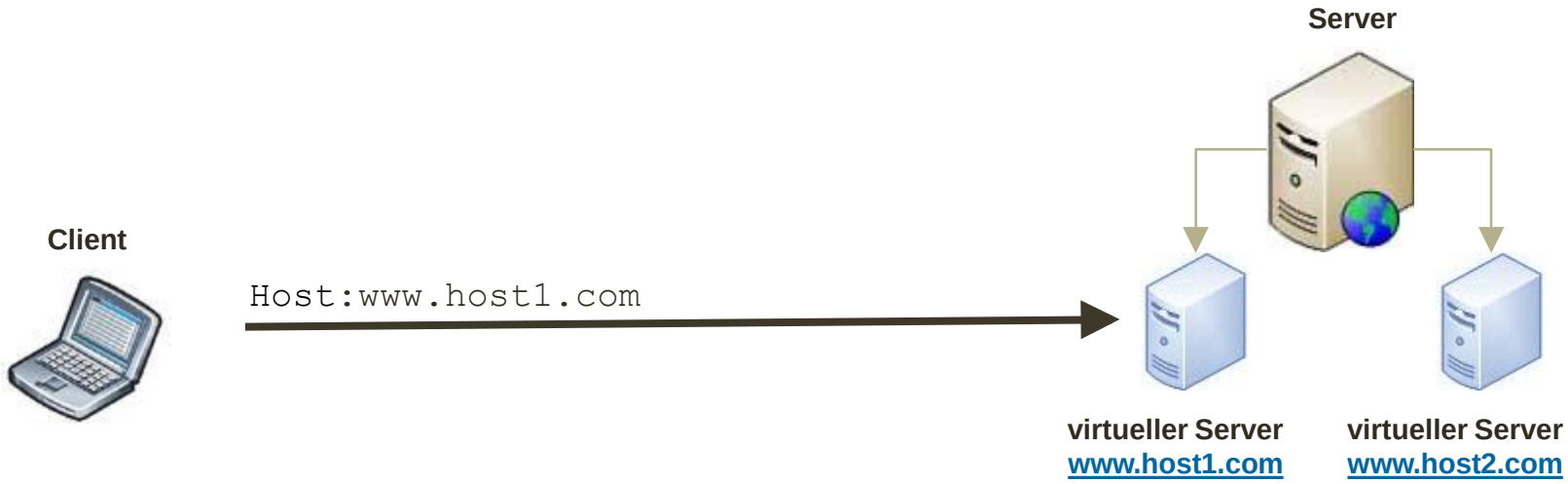
```
GET /index.php HTTP/1.1<crlf>
Host: www.html-world.de<crlf>
User-Agent: Mozilla/4.0<crlf>
Accept: image/gif, image/jpeg, */*<crlf>
Connection: close<crlf>
<crlf>
<crlf>
```

Response

Response Headers

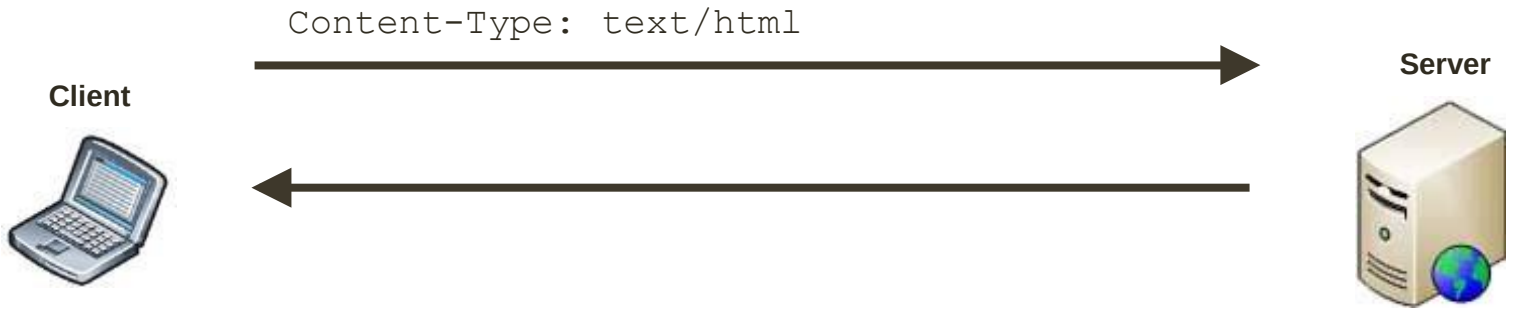
```
HTTP/1.x 200 OK<crlf>
Transfer-Encoding: chunked<crlf>
Date: Sat, 28 Nov 2009 04:36:25 GMT<crlf>
Pragma: public<crlf>
Expires: Sat, 28 Nov 2009 05:36:25 GMT<crlf>
Etag: "pub1259380237;gz"<crlf>
Cache-Control: max-age=3600, public<crlf>
Content-Type: text/html; charset=UTF-8<crlf>
Last-Modified: Sat, 28 Nov 2009 03:50:37 GMT<crlf>
<crlf>
<crlf>
<title>Title of page</title>
<!-- ... rest of the html ... --
>
```

Host Request Header



- **The same IP address can be used for multiple hosts**
- **To determine the virtual server the host name is by-passed**

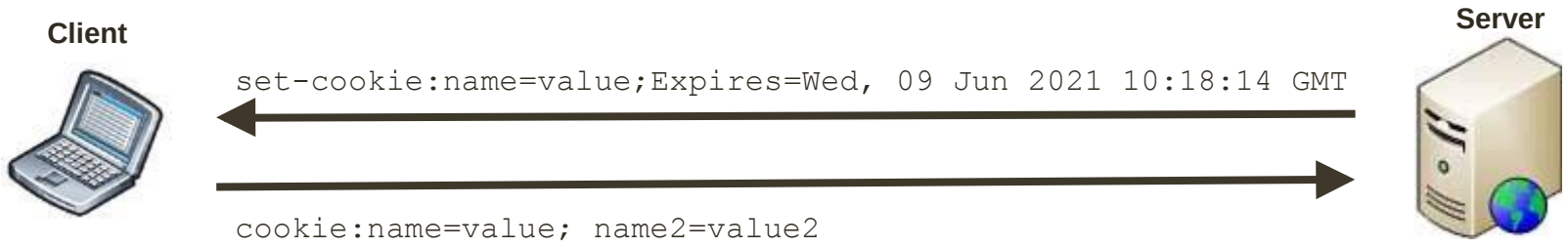
Content-Type Entity Header



- **Declares the parsing / interpreting format for the Client/Server**
- **The Content-Type is declared in MIME-Type format**
  - Specifies the media type (text, video, audio, application, …) and a subtype (html)

text/css	*.css
text/html	*.htm *.html *.shtml
text/javascript	*.js
text/plain	*.txt
image/gif	*.gif
image/jpeg	*.jpeg *.jpg *.jpe
image/png	*.png
message/http	
video/mpeg	*.mpeg *.mpg *.mpe
audio/basic	*.au *.snd
application/gzip	*.gz
application/msexcel	*.xls *.xla
application/mspowerpoint	*.ppt *.ppz *.pps *.pot
application/msword	*.doc *.dot
application/octet-stream	*.bin *.exe *.com *.dll *.class
application/pdf	*.pdf
application/postscript	*.ai *.eps *.ps
application/rtc	*.rtc
application/xml	*.xml
application/x-javascript	*.js
application/zip	*.zip

Cookie Request/Response Headers



- **A cookie represents a small piece of data**

- **Server writes the Cookie to the Client (`set-cookie:…`)**
- **Client transmits all Cookies for the current site back to the Server (`cookie:…`)**

SESSION MANAGEMENT

HTTP

Introduction

- **Invented by Netscape as HTTP Header**
  - Originally a non-standard header
  - Today standardized in RFC 6265
- **Cookie is a small data unit stored on the Client and transmitted to Server on every Request**
  - Max 4096 Bytes
  - Max 50 Cookies per domain (varies by browser)
  - Max 3000 Cookies overall (varies by browser)
- **Cookie Expiration Time can be declared**
  - If a Cookie has no expiration, the Cookie is valid until the browser gets closed
  - Memory only Cookies are treaded as **Session**
- **Cookie can be declared as HTTPOnly to use the cookie only for HTTP / HTTPS requests**

Scope

- **Domain and Path declare the Scope of the Cookie**

- **Domain attribute**
  - **Cookie/Session is also present for subdomains**
  - **If no domain specified, the current domain is set**
- **Path attribute**
  - **Defines the scope inside a domain**
  - **The cookie is valid for the given path (and paths below)**

- **Example**
  - sessionId=AYQEVnDKrdst; Domain=.foo.com; Path=/access; HttpOnly

QUESTIONS?

Sources

- **Slides**
  - Markus Wirrer, Namics
  - http://en.wikipedia.org/

Repetitionsfragen 1/2

- **HTTP Basics**
  1. Ihr Browser verwendet HTTP/1.1, der von Ihnen angesprochene Server HTTP/2. Welche HTTP-Version wird für die Kommunikation verwendet?
  2. Beschreiben Sie den Unterschied zwischen URN und URL sowie den Zusammenhang mit URI.
  3. Erläutern Sie das Verhalten des Browsers bezüglich des Fragments einer URL bei einem Request auf eine Server-Resource.
  4. Ein Server gibt beim Request auf die Resource /index2.html den Status 404 zurück. Beschreiben Sie die wahrscheinlichste Ursache des Problems.
  5. Welche HTTP Methode wählen Sie für einen Upload eines neuen Files?

Repetitionsfragen 2/2

- **Headers / Cookies**
  1. Sie beobachten in der Chrome Netzwerkkonsole den Status 304 (Not Modified). Welcher Teil der Response wurde vom Server weggelassen?
     - Erläutern Sie den Mechanismus (zwischen Request & Repsonse), welcher dieses Verhalten beschreibt.
  2. Mehrere Domains werden auf demselben Web-Server mit derselben IP gehostet. Wie werden die einzelnen Virtual Hosts adressiert?
  3. Beschreiben Sie den Einsatzzweck von Session Cookies.
  4. Wozu dient das HttpOnly- sowie das Secure-Flag bei Cookies in Bezug auf die Browser API?

Repetitionsfragen Lösungen 1/2

- **HTTP Basics**
  1. HTTP ist abwärtskompatibel, es wird immer den höchsten gemeinsamen Nenner (d.h. HTTP/1.1) verwendet.
  2. URI ist der Überbegriff und beschreibt URNs sowie URLs. Eine URN beschreibt eine Resource eindeutig, beschreibt aber nicht, wo sich diese befindet. Die URL beschreibt die Lokation einer Resource und kann eine URN enthalten.
  3. Das Fragment wird Client-seitig abgeschnitten. Der Server erhält dieses nicht, es handelt sich beim Fragment um Markers innerhalb des HTMLs.
  4. Das File liegt auf dem Server nicht vor – File Not Found.
  5. Für das Kreieren von neuen Resourcen auf einem Server ist aus Sicht des Standards die POST Methode vorgesehen.

Repetitionsfragen Lösungen 2/2

- **Headers / Cookies**
  1. Die HTTP Entity (d.h. spezifische Headers sowie der Content) wurden nicht mitgesendet.
     - Siehe Folien 25/26 (spezifisch ETag / If-None-Match Headers).
  2. Der Client beschreibt im Request die verlangte Domain mittels `Host` Header. Dieser wird vom Web-Server ausgewertet und der Request der zugewiesenen Web-Applikation übergeben.
  3. Session Cookies werden vor allem für Logins verwendet. Diese Cookies verfallen beim Beenden des Browsers, was ein weiteres Login beim nächsten Verwenden derselben Page nach sich zieht.
     - Sessions bestehen häufig aus einer zufälligen, Server-seitig generierten ID. Diese ID wird durch set-cookie an den Client übertragen. Der Browser sendet das Cookie bei jedem weiteren Request zurück an den Server.
       Der Server an sich assoziert im diese ID mit einem Server-Speicher, wo beispielsweise den Login-Status o.ä. abgelegt werden kann.
       Ein Logout wird häufig durch das Zerstören der Server-seitigen Session (d.h. entfernen der ID aus dem Server-Speicher) bewerkstelligt.
  4. Das HttpOnly Flag dient dazu, den Zugriff auf das Cookie ausschliesslich für HTTP-gebundene Requests/Responses zu ermöglichen. Anderen Protokollen sowie Browser API calls (z.B. mit JavaScript / VBScript / …) wird der Zugriff auf das Cookie untersagt.
     Mit Secure wird das Cookie ausschliesslich über sichere Channels gesendet. Die Klar-Text Übertragung wird somit ausgeschlossen und Cookie Hijacking erschwert.