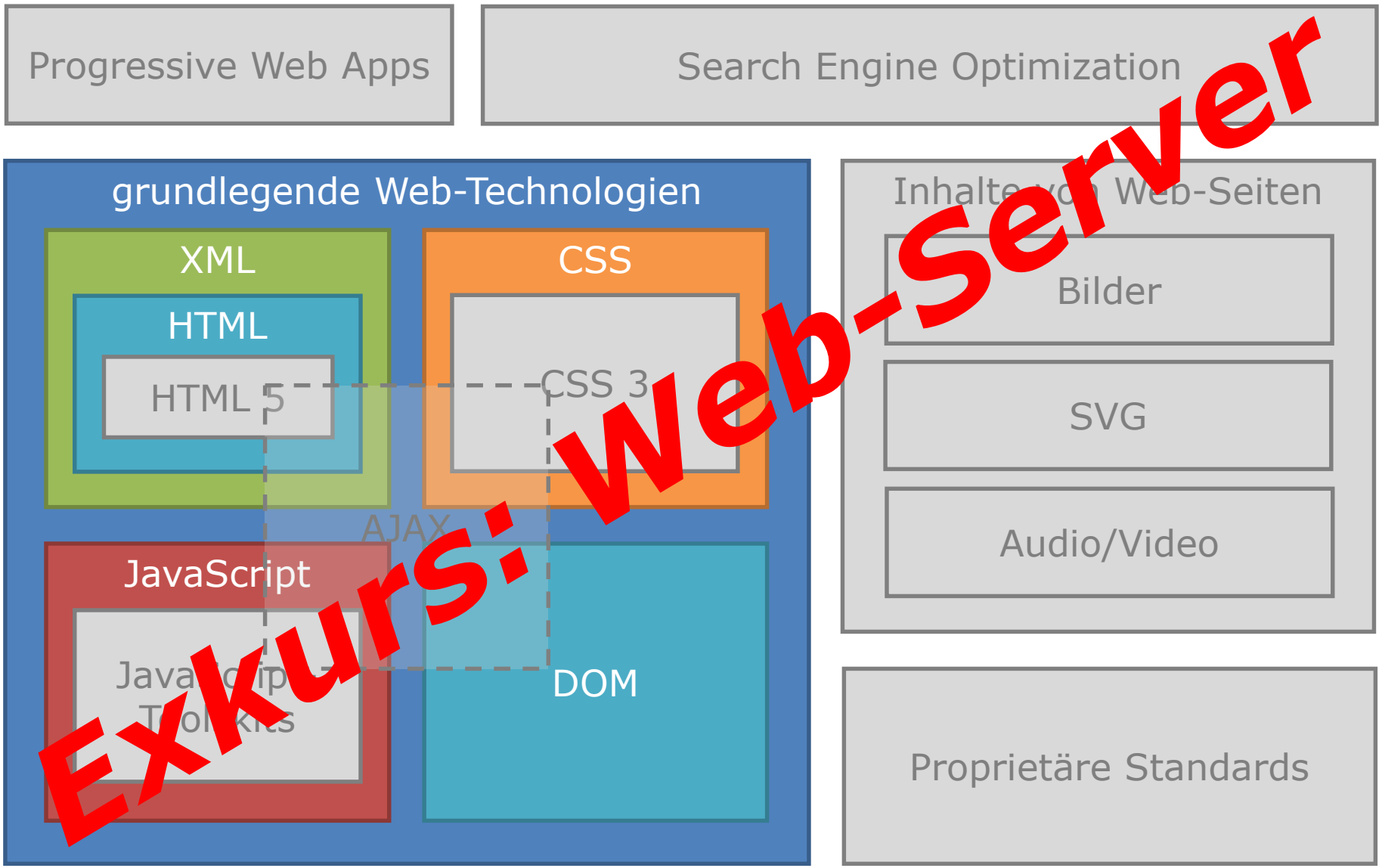
Version 2024b

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Grundlagen der Web-Programmierung
Exkurs: Web-Server

Prof. Dr.-Ing. Tenshi Hara
tenshi.hara@ba-sachsen.de

Aufbau der Lehrveranstaltung



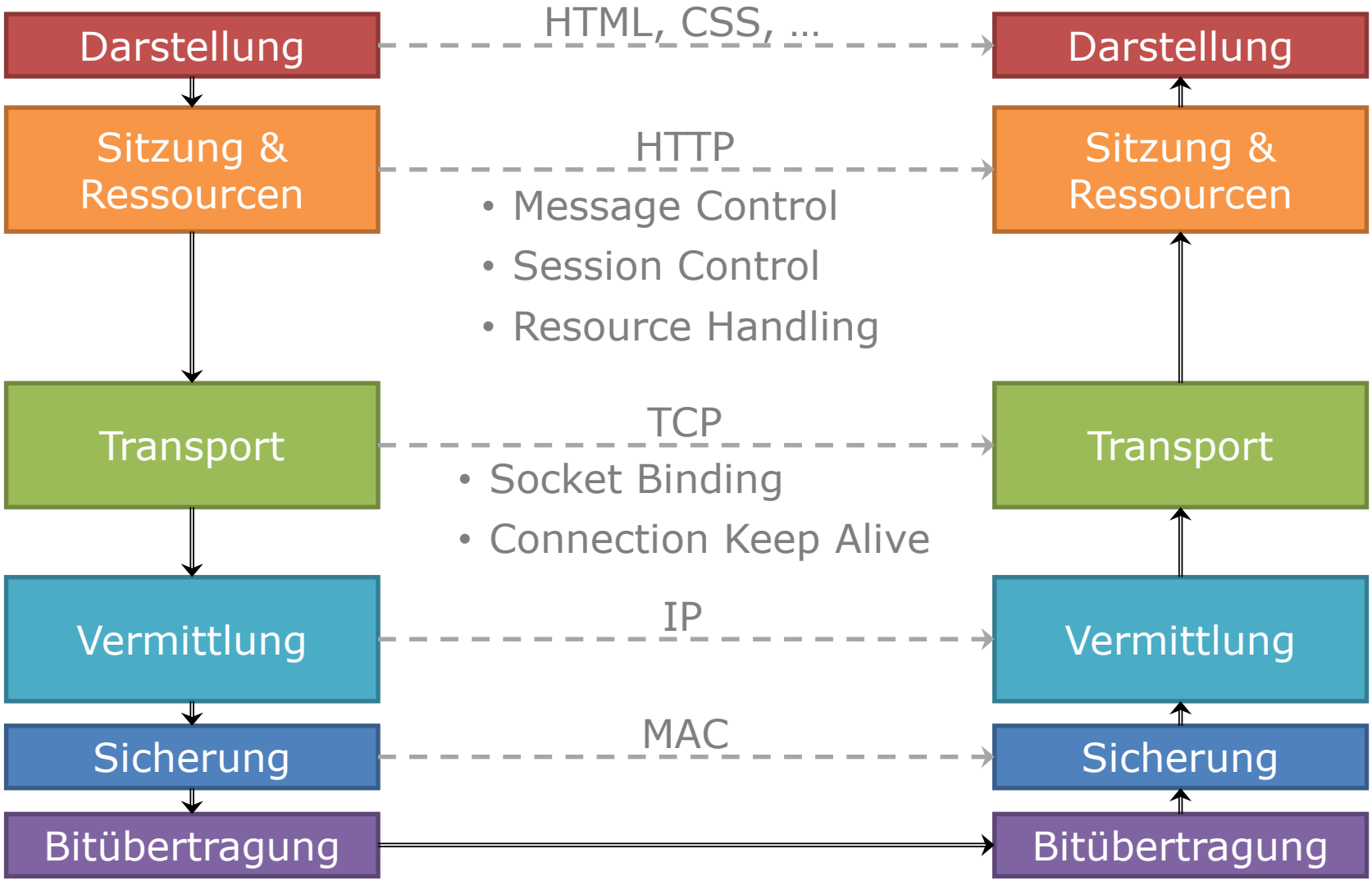
Web-Server

- liefern Web-Inhalte an Endgeräte aus
- sind i.d.R. dumm → Request-Response-Schema
  - liefern nur aus, was angefordert wird oder werden könnte
  - können nur mit Clients kommunizieren nachdem diese Verbindung etabliert haben
- können statische und dynamisch generierte Inhalte ausliefern
- kommunizieren socket-orientiert über wohldefinierte Ports
- liefern Inhalte i.d.R. über das Hypertext Transfer Protocol (HTTP) aus

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

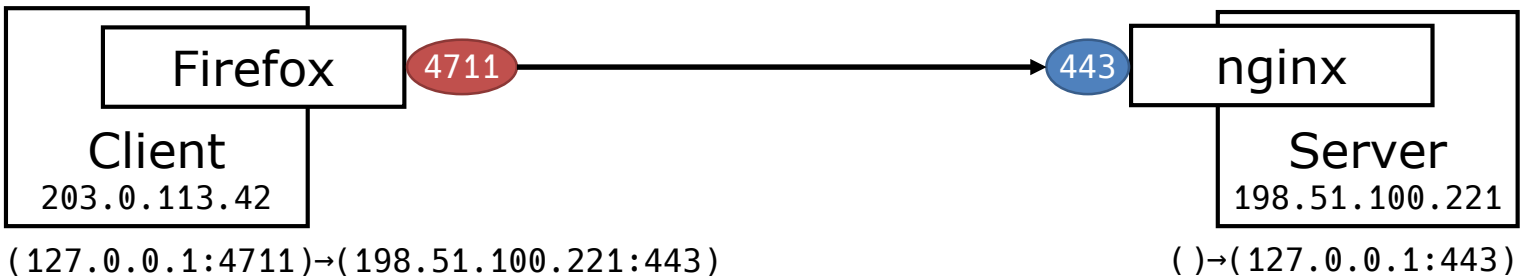
Aspekte der Kommunikation

Netzwerk-Stack des HTTP

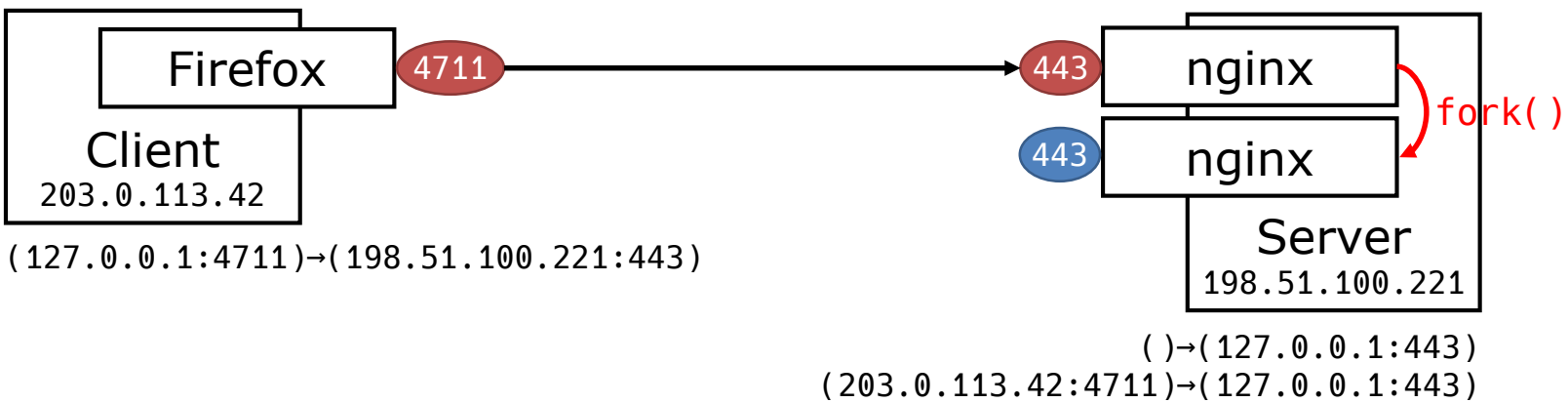


Ports und Socket-basierte Kommunikation

1. Client kontaktiert Server am Listener Socket



2. Server forkt Anwendung und bildet Connected Socket

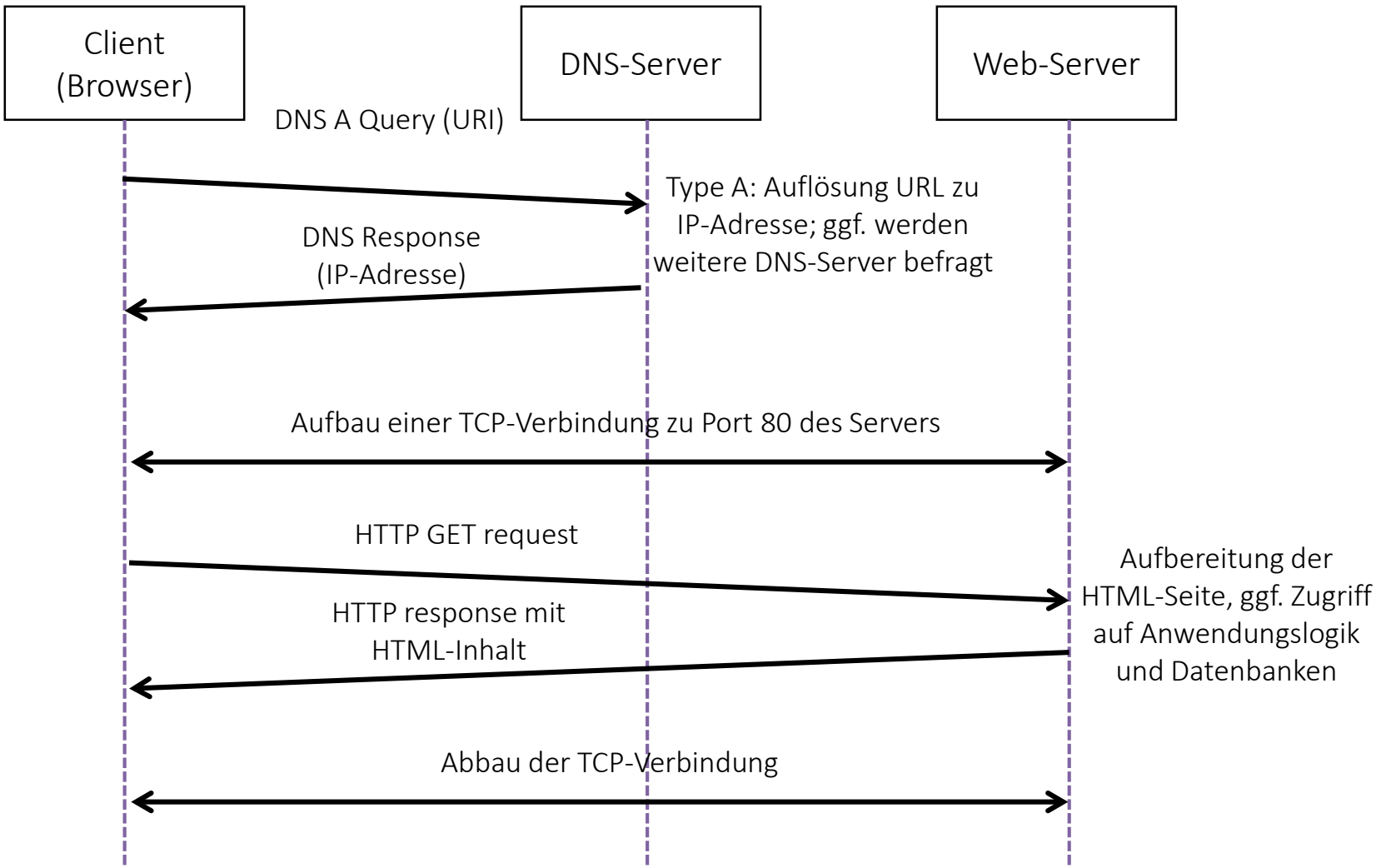


- i.d.R. hat Connected Socket gleichen Port wie Listener Socket
- Abweichungen sind aber möglich! (bspw. Active FTP)

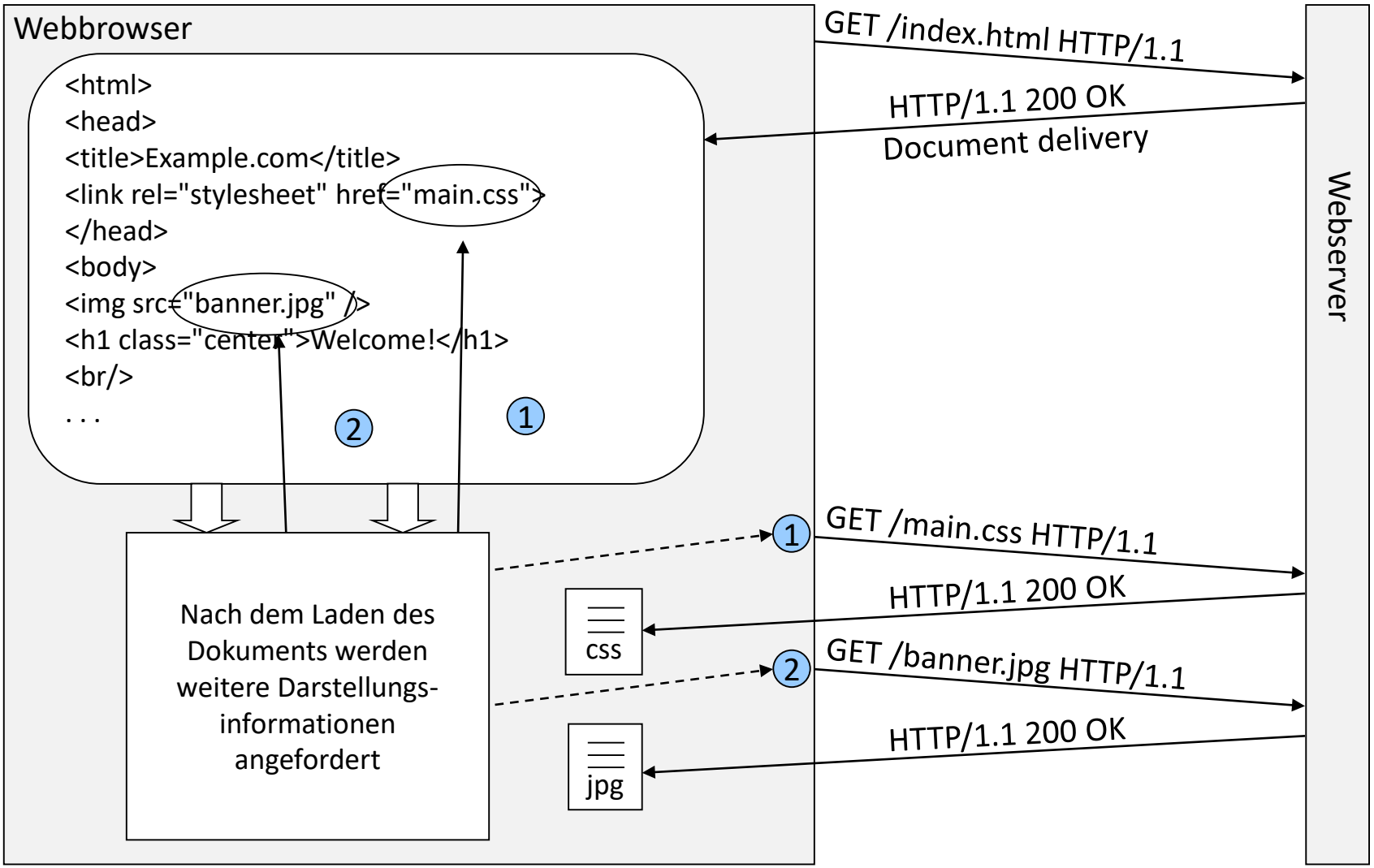
ZUSTANDSLOSE OPERATIONEN

- Web-Server merken sich ihre Clients nicht (wäre viel zu aufwendig)
- alles muss in der Kommunikation wiederholt werden
  - URL-Parameter
  - Cookies
- Problem: TCP-Verbindung

Abruf einer Webseite (1/2)

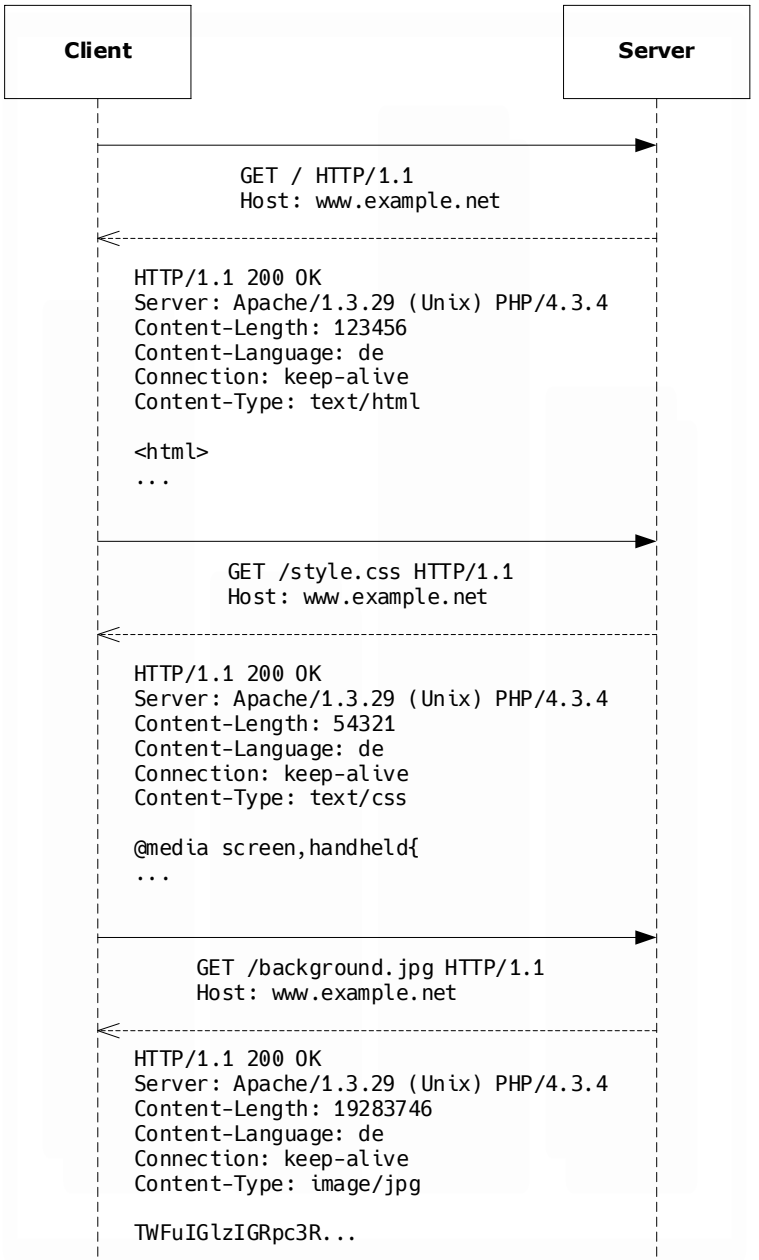


Abruf einer Webseite (2/2)



Hypertext Transfer Protocol (1/2)

- nachrichtenbasierte Kommunikation
  - Request fordert Ressourcen an
  - Response liefert Ressourcen aus
- Nachrichten bestehen aus
  - Header mit Meta-Informationen
    - angeforderter Host
    - Client-Kennung
    - Datum, Uhrzeit
    - Status-Code
    - …
  - ggf. Body mit Inhalten
    - HTML-Dokumente
    - Bilder und Videos
    - …



Hypertext Transfer Protocol (2/2)

Spezifikationen und Standards

- HTTP/1.0
  - RFC 1945 (1996)
- HTTP/1.1
  - RFC 2616 (1999)
  - RFC 7230 bis 7235 (2014)
- HTTP/2
  - RFC 7540 und 7541 (2015)
  - RFC 9113 (2022)
- HTTP/3 (HTTP over QUIC)
  - RFC 9114 (2022)
- TCP
  - RFC 793 (1981)
  - RFC 6298 (2011)
  - RFC 9293 (2022)
- UDP und QUIC
  - RFC 768 (1980)
  - RFC 8999 bis 9002 (2021)

HTTP-Methoden (1/3)

folgen klassischem CRUD-Prinzip

(**C**reate) Ressourcen erzeugen

- POST unbegrenzte Datenmenge an Server senden
- PUT Ressource auf Server hochladen (überschreibt existierende Ressourcen)

(**R**ead) Ressourcen lesen

- GET Ressource vom Server anfordern

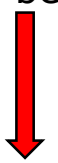
(**U**pdate) Ressourcen aktualisieren

- PATCH Ressource auf Server ergänzen (überschreibt nicht wie PUT)

(**D**elete) Ressourcen löschen

- DELETE löscht Ressource auf Server

bei Formularen (Kapitel 1) werden vornehmlich POST und GET benutzt.



GET ist in Server- und Proxy-Logs problematisch
→ URL wird nicht verschlüsselt

Besser: POST
→ Daten liegen im verschlüsselten HTTP-Paket.

HTTP-Methoden (2/3)

über CRUD hinaus: Kommunikation optimierende Methoden

- `HEAD` nur Metadaten der Ressource anfordern
  (entspricht dem Header der Server-Antwort auf ein GET)
- `TRACE` liefert Anfrage an Client zurück, so wie Server sie empfangen hat, um unterwegs veränderte Anfragen erkennen zu können
- `OPTIONS` liefert die vom Server unterstützten Methoden zurück
  (abfragen ob Server WebDAV, WebSockets, etc. unterstützt)
- `CONNECT` weist Proxy-Server an, TLS/SSL-Tunnel zu etablieren
- `UPGRADE` weist Server an, Verbindung auf WebSocket zu ändern
- `PROPFIND`, `PROPPATCH`, `MKCOL`, `COPY`, `MOVE`, `LOCK` und `UNLOCK` sind WebDAV-Methoden und werden hier nicht näher erläutert

HTTP-Methoden (3/3)

grundsätzlicher Aufbau von HTTP-Anfragen: Text-Nachrichten

```
<Method> <Resource>
         "HTTP/" ((("1"|"2"|"3")".0")|"1.1") "\CR\LF"
"Host: " <Host> "\CR\LF"
{<additionalHeader> "\CR\LF" }
[
"\CR\LF"
<Body>
]
```

Methoden-Aufrufe und Antworten darauf sind ähnlich aufgebaut

- Header enthält Metadaten
- doppelter Windows-Zeilenumbruch trennt Header vom Body
- Body enthält Ressourcen (kann leer sein)

HTTP-Antworten

grundsätzlicher Aufbau von HTTP-Antworten: Text-Nachrichten

```
"HTTP/" ((("1"|"2"|"3")".0")|"1.1")
        <StatusCode> <StatusMessage> "\CR\LF"
"Date: " <Date> "\CR\LF"
"Location: " <URI> "\CR\LF"
{<additionalHeader> "\CR\LF" }
[
"\CR\LF"
<Body>
]
```

- an HTTP-Versionsangabe erkennt Client Down-/Upgrade der Verbindung
- die Statusnachricht ist eigentlich redundant; Status-Code reicht
- Angabe des Ressourcenorts ist bei Umleitungen wichtig

HTTP-Status-Codes (1/2)

- Server beantworten Anfragen immer mit einem Ressourcen-Status
- Status-Codes sind in Kategorien eingeteilt
  - 1xx – Informationen
    - Bearbeitung der Anfrage dauert an
    - diese Zwischenantworten verhindern, dass Client nach einer Zeitüberschreitung die Anfrage abbricht
  - 2xx – Erfolgreiche Operation
    - Anfrage wurde erfolgreich bearbeitet
    - Antwort-Body kann leer sein oder Ressourcen enthalten
  - 3xx – Umleitung
    - weitere Schritte seitens des Clients sind erforderlich
    - Client muss Ressource an einem anderen Ort erneut anfordern
    - Server-Antwort enthält im `Location`-Header, wo sich die angeforderte Ressource befindet

HTTP-Status-Codes (2/2)

- Server beantworten Anfragen immer mit einem Ressourcen-Status
- Status-Codes sind in Kategorien eingeteilt
  - 4xx – Client-Fehler
    - bei der Bearbeitung der Anfrage ist ein Fehler aufgetreten
    - Fehler begründet sich in der vom Client gestellten Anfrage (angeforderte Ressource existiert nicht, benötigt Authentifizierungsinformationen, …)
  - 5xx – Server-Fehler
    - bei der Bearbeitung der Anfrage ist ein Fehler aufgetreten
    - Fehler bei der Server-seitigen Bearbeitung (auszuführende Programme nicht gefunden (PHP, Java, …), Speicher- oder Zugriffsverletzungen, …)

→ https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

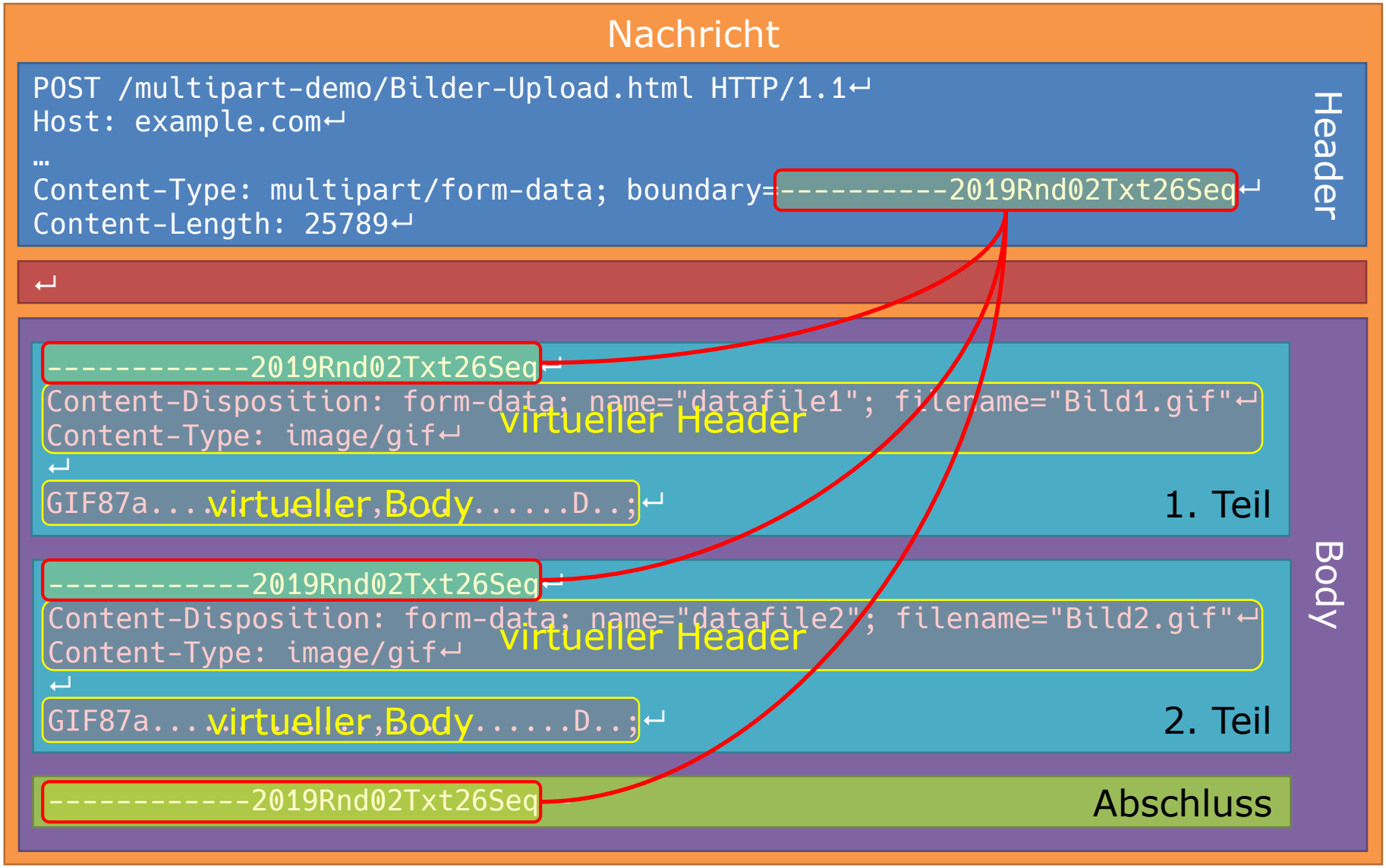
Multipart-Nachrichten (1/2)

- HTTP-Anfragen und -Antworten können auch multiple Ressourcen für den Upload/Download enthalten
- es bleibt beim doppelten Zeilenumbruch zwischen Header und Body
- neu: es gibt besondere doppelte Zeilenumbrüche
  - trennen multiple Header-Body-Kombinationen voneinander
  - müssen sich von „normalen" Doppel-Zeilenumbrüchen unterscheiden

  → Voranstellen einer eineindeutigen Identifikationssequenz

  → Identifikationssequenz
    - wird zufällig gewählt aber so, dass sie sonst nicht in der Anfrage bzw. Antwort vorkommen kann
    - hat eine Mindest- und Maximallänge

Multipart-Nachrichten (2/2)



BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Ressourcen-auslieferung

Statische und dynamische Web-Inhalte

- Web-Server liefern Ressourcen nach Bedarf aus
- Ressourcen können sein
  - statisch – zum direkten Ausliefern auf Server abgelegt
    - Mediendateien (Bild, Video, Audio)
    - CSS-Dateien
    - …
  - dynamisch – in Folge des Aufrufs durch den Server zu berechnen
    - HTML-Inhalte
    - Bildausschnitte
    - interaktive Elemente
    - …

DYNAMISCHE INHALTE

- werden durch Server-seitige Skripte generiert
  - synchron – sofort als Antwort auf eine Anfrage
  - asynchron – zeitversetzt als Reaktion auf eine Anfrage (ggf. zweite Anfrage für Ergebnis notwendig)
- können durch unterschiedliche Mechanismen generiert werden
  - Kommandozeilenskript (Bash Scripting)
  - allgemeine Programmiersprachen (insbesondere Python)
  - angepasste Programmiersprachen (bspw. Java)
  - spezialisierte Programmiersprachen (bspw. PHP, Node)

→ mehr dazu im Wahlpflichtmodul 3MI-WEBPR-50

Forwards

- Ressourcen müssen oft aggregiert oder andernorts angefordert werden (Anfragedelegation; Request Delegation)
- Server teilt dies dem Client mit mittels
  - agnostischer Weiterleitung (Method-agnostic Forward)
    - `301 Moved Permanently`
    - `302 Found`
    - `303 See Other`
  - Angabe eines (zuständigen) Proxy-Servers (Proxy Forward)
    - `305 Use Proxy`
  - treuer Weiterleitung (Method-true Forward)
    - `307 Temporary Redirect`
    - `308 Permanent Redirect`

Rewrites

- Ressourcen werden oft unter verschiedenen URI oder prokuriert angeboten, oder dynamisch erzeugt
- Nutzung von Platzhalter-URI, die Server-intern umgeschrieben werden
  - Ortsumschreibung (Location Rewrite)
    Beispiel in httpd:
    ```
    RewriteCond ^Bier.jpg$ /images/beer.jpg [L]
    ```
  - Prozessdelegation (Process Delegation)
    Beispiel in httpd:
    ```
    RewriteRule ^images/(\d{1,6})\.png$ /badge.php?id=$1 [L]
    ```
  - Delegation an Prokura (Proxy Delegation)
    Beispiel in nginx:
    ```
    location / {
       proxy_set_header Host            $host;
       proxy_set_header X-Real-IP       $remote_addr;
       proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
       proxy_pass http://[::1]:4711;
    }
    ```

Kompression

- Text-Inhalte eignen sich besonders gut für verlustfreie Kompression
- im WWW sind zwei Standards weit verbreitet
  - DEFLATE
    - kombiniert Lempel-Ziv-Storer-Szymanski-Algorithmus und Huffman-Kodierung
    - Basis von ZIP
    - arbeitet mit variabler Fenstergröße
      → dadurch effizienter als GZIP
  - GZIP (GnuZIP)
    - basiert auf DEFLATE, aber feste Fenstergröße (32KiB)
    - weniger effizient als DEFLATE; Fenster ohne Wiederholungen werden unkomprimiert übertragen
      → dadurch verzögerungsfreier als DEFLATE

BROTLI

- Google-Alternative zu DEFLATE und GZIP
  (Zopfli ist Googles DEFLATE-Algorithmus)
- basiert auf LZ77; nutzt aber auch Huffman-Kodierung
- statt dynamischem Wörterbuch:
  vordefiniertes Wörterbuch (120 KiB) mit 13.000 Einträge der am meisten genutzten Ausdrücke in Text und HTML-Dokumenten
  → bis zu 20% bessere Kompression für solche Daten als bei DEFLATE

Aushandlung der Kompression

Anfrage enthält Liste akzeptierter Kodierungen:

```
GET /resource.html HTTP/1.1
Host: www.example.com
Accept-Encoding: brotli, deflate, gzip
```

Antwort des Servers führt ausgewählte Kodierung explizit auf:

```
HTTP/1.1 200 OK
Date: Tue, 28 Jun 2022 07:49:34 GMT
Server: Apache
Last-Modified: Tue, 28 Jun 2022 07:45:00 GMT
Accept-Ranges: bytes
Content-Length: 438
Connection: close
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
```

Kompression von Medien

- DEFLATE und GZIP sind für Medien-Kompression eher ungeeignet
- Beispiel: ein JPEG-Bild ist bereits komprimiert; erneute Kompression würde keine weitere Einsparung bringen; stattdessen würde das doppelt komprimierte Bild sogar noch größer werden, da weitere Header hinzukommen
- weiterhin sind zu kleine Datenmengen auch schlecht komprimierbar
  → nicht für Streams geeignet

⇒ mehr zur Medien-Kompression im Kapitel „Inhalte von Web-Seiten"

Beispiel-Server

Apache httpd

- Open-Source-Projekt der Apache Foundation
- häufig im LAMP-Stack (Linux, Apache, MySQL und PHP)
- eingeteilt in (derzeit über 60 offiziell unterstützte) Module
- mit modularer Unterstützung von HTTP/2
- läuft als Dienst im Betriebssystem
  - in Linux-Umgebungen i.d.R. als unprivilegierter Nutzer `www-data`
  - in Windows-Umgebungen oft ungünstig mit Admin-Rechten
- liefert nur unbestimmte Inhalte aus
  - dynamische Inhalte werden an Verarbeitungsmodule (z.B. PHP) als Typressourcen ausgelagert
  - statische Inhalte werden vom Kernmodul als Dateiressourcen bedient

Apache httpd – Multi-Prozess-Module (MPM)

- `Process` - Pre-Fork-Methode
  - zuverlässig für kleine Seiten, skaliert aber schlecht, insbesondere wegen hohem Speicherbedarf
  - weist ggf. Verbindungen wegen Überlastung ab
- `Worker` - ein Kontrollprozess mit mehreren Arbeitsprozessen
  - jeder Arbeitsprozess startet festgelegte Anzahl Worker-Threads und je einen Listener-Thread
  - Listener nimmt Anfragen entgegen; verteilt auf die Worker
  - Probleme mit hochfrequenten Zugriffen auf Grund von Skalierungsproblemen wegen des einzelnen Kontrollprozesses
- `Event` - ähnlich Worker, aber ein Listener-Thread arbeitet auf einem Worker-Thread im Prozesskontrollfluss
  - skaliert mit lang-anhaltenden Verbindung besser (TCP Connection Keep-Alive)

Apache httpd – Konfigurationseinstellungen

- Konfigurationseinstellungen gelten global und sollten spärlich gesetzt werden
  - Debug-Meldungen an/aus
  - Nutzerkontext im Betriebssystem
  - verfügbare Module (bei LAMP i.d.R. in `/etc/apache2/mods-enabled/*`)
  - globale Konfiguration bei LAMP i.d.R. in `/etc/apache2/apache2.conf`
- seitenspezifische Einstellungen zur Verfeinerung der Einstellungen
  - SSL/TLS-Einstellungen (Pfad zum Zertifikat, etc.)
  - verfügbare Module einbinden und aktivieren (Rewrite, MIME, …)
  - Log-Verzeichnis und –Tiefe
  - Regeln für Forward und Rewrite
  - bei LAMP i.d.R.
    - allgemeiner Seitenkonfiguration: `/etc/apache2/sites-enabled/*`
    - spezifischen Ordnern: `/var/www/example.com/httpdocs/.htaccess`

Apache httpd – Beispiel globaler Konfiguration

```
ServerAdmin webmaster@example.com
User  ${APACHE_RUN_USER}
Group ${APACHE_RUN_GROUP}

TimeOut 180
KeepAlive on
MaxKeepAliveRequests 100
HostnameLookups Off

ErrorLog ${APACHE_LOG_DIR}/error.log
LogLevel error

IncludeOptional mods-enabled/*.load
IncludeOptional mods-enabled/*.conf

<FilesMatch "^\.ht">
    Require all denied
</FilesMatch>

SSLCipherSuite EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH
SSLProtocol All -SSLv2 -SSLv3 -TLSv1 -TLSv1.1
SSLHonorCipherOrder On
Header always set Strict-Transport-Security "max-age=63072000; includeSubDomains; preload"
Header always set X-Frame-Options DENY
Header always set X-Content-Type-Options nosniff
SSLCompression off
SSLUseStapling on
SSLStaplingCache "shmcb:/tmp/stapling-cache(128000)"
SSLSessionTickets Off

ServerTokens Prod
ServerSignature Off

IncludeOptional sites-enabled/*.conf
```

Apache httpd – Beispiel seitenspezifischer Konfiguration (1/2)

```
<VirtualHost *:443>
    SuexecUserGroup "#1001" "#1001"
    ServerName example.com
    ServerAlias www.example.com

    Protocols h2 http/1.1
    H2Push on
    H2PushPriority * after
    H2PushPriority text/css before
    H2PushPriority image/jpg after 32
    H2PushPriority image/jpeg after 32
    H2PushPriority image/png after 32
    H2PushPriority application/javascript interleaved

    DocumentRoot /var/www/example.com/htdocs
    LogLevel warn
    ErrorLog /var/www/example.com/errors/httpd_ssl_error
    CustomLog /var/www/example.com/errors/httpd_ssl_access combined

    AddType application/x-httpd-php .php
    DirectoryIndex index.php index.html

    <Directory /var/www/example.com/htdocs>
        Options -Indexes +IncludesNOEXEC +SymLinksIfOwnerMatch +ExecCGI
        allow from all
        AllowOverride All
        Options=ExecCGI,Includes,IncludesNOEXEC,Indexes,MultiViews,SymLinksIfOwnerMatch
        Require all granted
        SSLRequireSSL
        Options -Includes -ExecCGI
    </Directory>
```

Apache httpd – Beispiel seitenspezifischer Konfiguration (2/2)

```
    RewriteEngine on
    RewriteCond %{HTTP_HOST} =example.com
    RewriteRule ^(.*) https://www.example.com$1 [R]

    SSLEngine on
    SSLCertificateKeyFile     /etc/letsencrypt/live/example.com/privkey.pem
    SSLCertificateFile        /etc/letsencrypt/live/example.com/cert.pem
    SSLCertificateChainFile   /etc/letsencrypt/live/example.com/chain.pem

    Alias /error_docs /var/www/example.com/error_docs
    ErrorDocument 400 /error_docs/bad_request.html
    ErrorDocument 401 /error_docs/unauthorized.html
    ErrorDocument 403 /error_docs/forbidden.html
    ErrorDocument 404 /error_docs/not_found.html
    ErrorDocument 500 /error_docs/internal_server_error.html
    ErrorDocument 405 /error_docs/method_not_allowed.html
    ErrorDocument 406 /error_docs/not_acceptable.html
    ErrorDocument 407 /error_docs/proxy_authentication_required.html
    ErrorDocument 412 /error_docs/precondition_failed.html
    ErrorDocument 414 /error_docs/request_uri_too_long.html
    ErrorDocument 415 /error_docs/unsupported_media_type.html
    ErrorDocument 501 /error_docs/not_implemented.html
    ErrorDocument 502 /error_docs/bad_gateway.html
    ErrorDocument 503 /error_docs/maintenance.html
</VirtualHost>
```

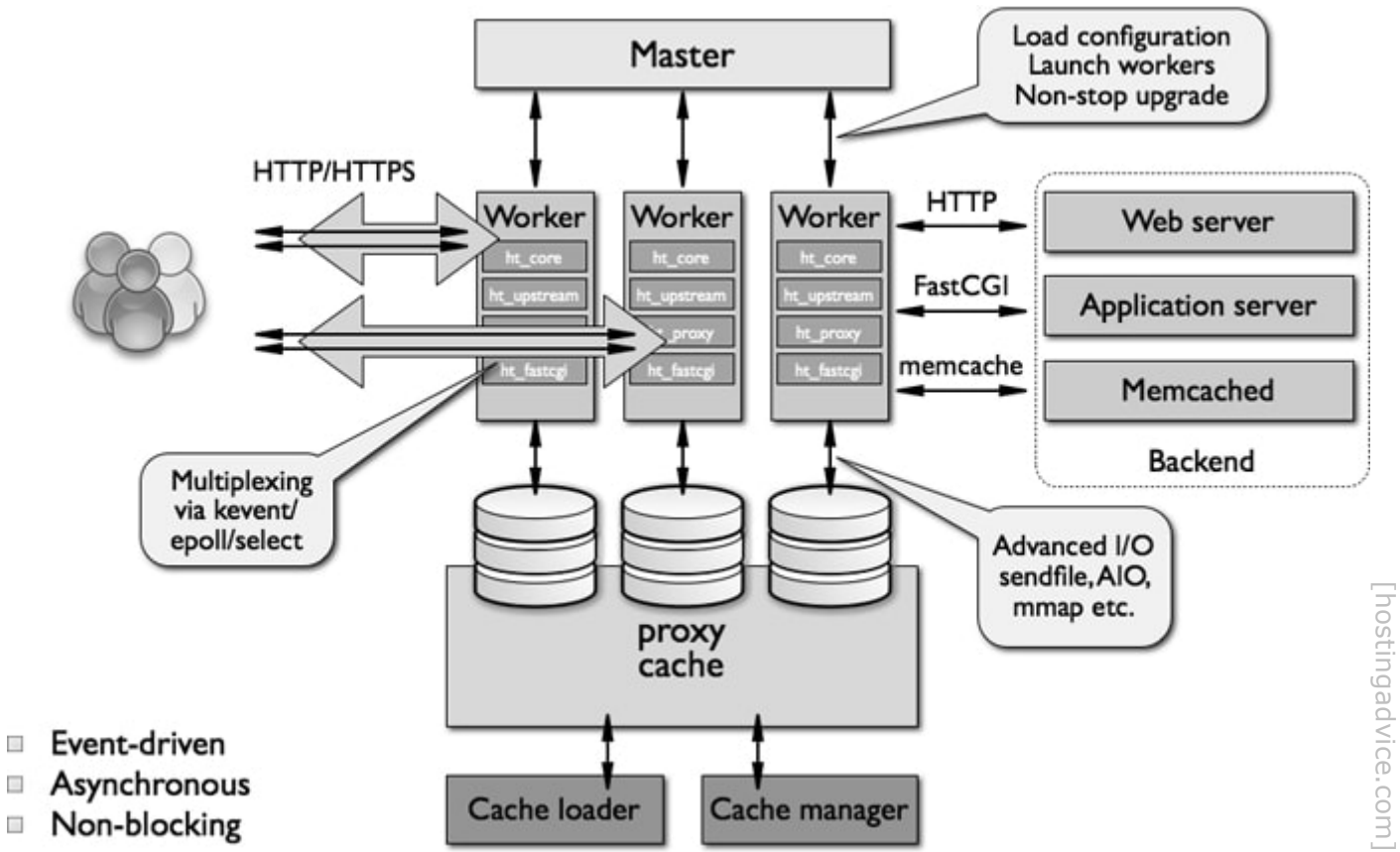
Apache nginx (1/2)

- entwickelt, um mindestens 10.000 parallele Anfragen effizient und ohne Drops bedienen zu können
- asynchrone, ereignisgetriebene Architektur
  (ähnlich Event-MPM bei httpd)
  → Handhabung hoher und fluktuierender Zugriffe
  → geeignete Vorhersagen zu CPU, RAM und Latenz
- ein Worker pro physischer CPU (Maximierung der Hardware-Effizienz)
  (nicht wie bei httpd ein Worker pro Verbindung)
- muss mit gewünschten Funktionen kompiliert werden
  (keine Modul-Unterstützung)
- läuft als Dienst im Betriebssystem
  - in Linux-Umgebungen i.d.R. als unprivilegierter Nutzer (`www-data`)
  - in Windows-Umgebungen oft ungünstig mit Admin-Rechten

Apache nginx (2/2)

- integrierte Unterstützung von HTTP/2
- gängigstes Einsatzszenario: Reverse-Proxy
  - nicht beschränkt auf HTTP
  - Upgrade-Server
    - internes HTTP auf externes HTTPs
    - internes HTTP/1 oder HTTP/1.1 auf externes HTTP/2 (und umgekehrt)
  - SMTP-, POP3- und IMAP-Proxy
- häufig auch als Load-Balancer
- Kombinationsbetrieb
  - dynamische Inhalte durch httpd
  - statische Inhalte direkt durch nginx

Apache nginx – Architektur



Apache nginx – Beispiel globaler Konfiguration

```
user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;

events {
    worker_connections 768;
    multi_accept on;
}

http {
    sendfile on;
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 2048;
    server_tokens off;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    ssl_protocols TLSv1.1 TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;

    access_log /var/log/nginx/access.log;
    error_log /var/log/nginx/error.log;

    gzip on;
    gzip_disable "msie6";

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;
}
```

Apache nginx – Beispiel seitenspezifischer Konfiguration

```
server {
    listen 443 ssl http2;

    server_name example.com;
    server_name www.example.com;

    ssl_certificate     /etc/letsencrypt/live/example.com/cert.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

    client_max_body_size 64m;

    root "/var/www/example.com/public_html";
    access_log "/var/log/example.com_nginx_ssl_access";
    error_log "/var/log/example.com_nginx_ssl_error";

    if ($host ~* ^example.com$) {
        rewrite ^(.*)$ https://www.example.com$1 permanent;
    }

    location / {
        proxy_set_header Host              $host;
        proxy_set_header X-Real-IP         $remote_addr;
        proxy_set_header X-Forwarded-For   $proxy_add_x_forwarded_for;
        proxy_pass http://127.0.0.1:8080;
    }

    location @fallback {
        proxy_set_header Host $host;
        proxy_pass http://127.0.0.1:8888;
        access_log off;
    }

    location ~ ^/(.*\.(ac3|avi|bmp|bz2|css|cue|doc|docx|dts|flv|gif|ico|img|iso|jpeg|jpg|js|mkv)$ {
        try_files $uri @fallback;
    }
}
```

Apache nginx – Beispiel Umleitung

```
server {
   listen 80 default_server;
   server_name _;
   access_log "/var/log/nginx_access";
   error_log "/var/log/nginx_error";
   return 301 https://$host$request_uri;
}
```

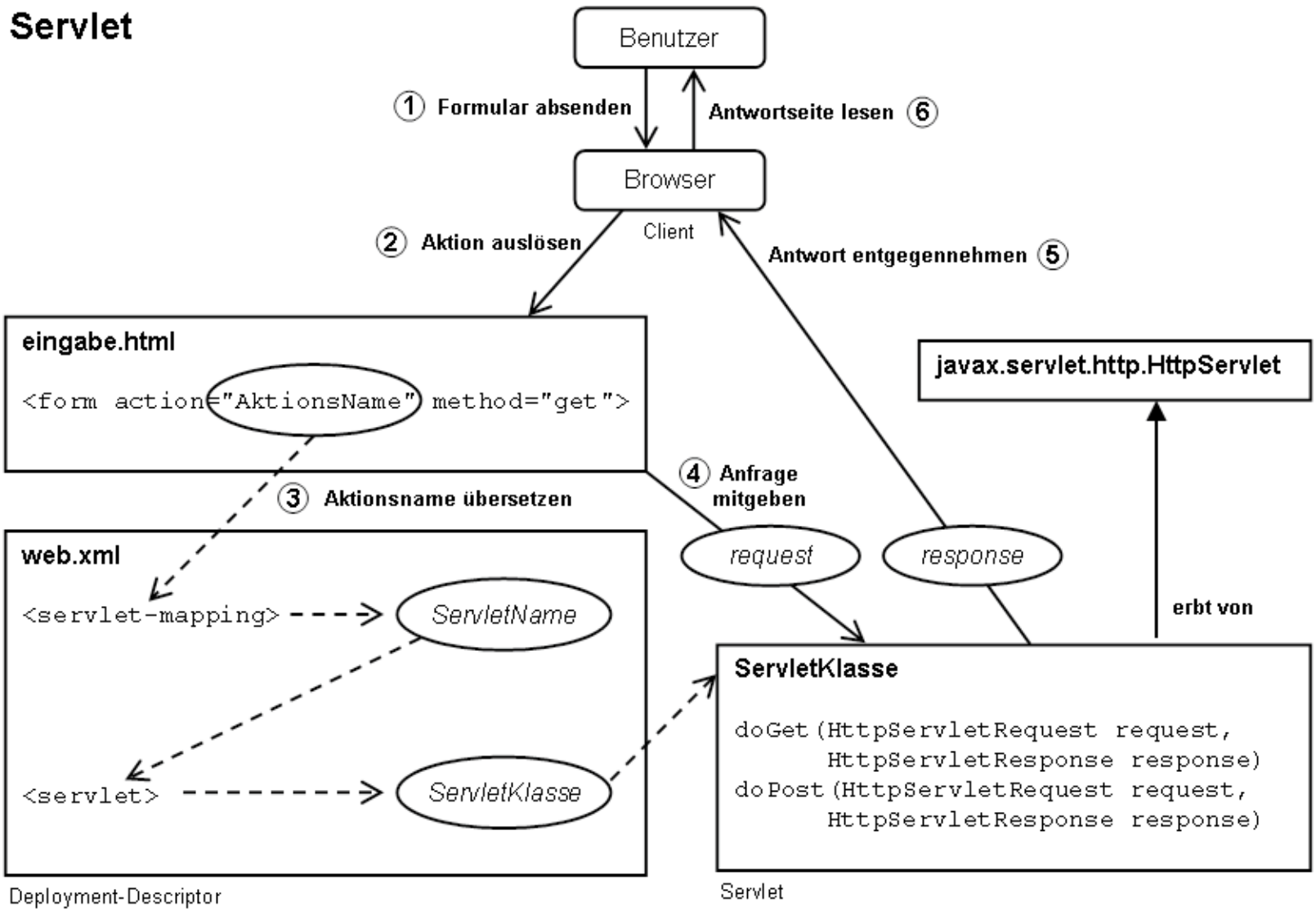
Performanzvergleich: httpd vs nginx

- statische Inhalte
  - nginx ist schneller als httpd (MPM Event)
    - 1024 konkurrierende Verbindungen: ca. 2,5mal schneller
    - 512 konkurrierende Verbindungen: ca. 2mal schneller und 4% weniger Speicherauslastung
- dynamische Inhalte
  - httpd (MPM Event, PHP-FPM, FastCGI) ähnlich zu nginx (PHP-Modul)
  - fast gesamte Anfrageverarbeitungszeit (Request Processing Time; RPT) fällt in PHP an → PHP-FPM, PHP-CGI & PHP-CLI ähnlich schnell
  - echte Zugewinne für dynamische Inhalte nur mittels
    - Varnish oder memcached als Caching-Layer
    - HHVM als PHP-Runtime
    - Load-Balancing

Apache Tomcat

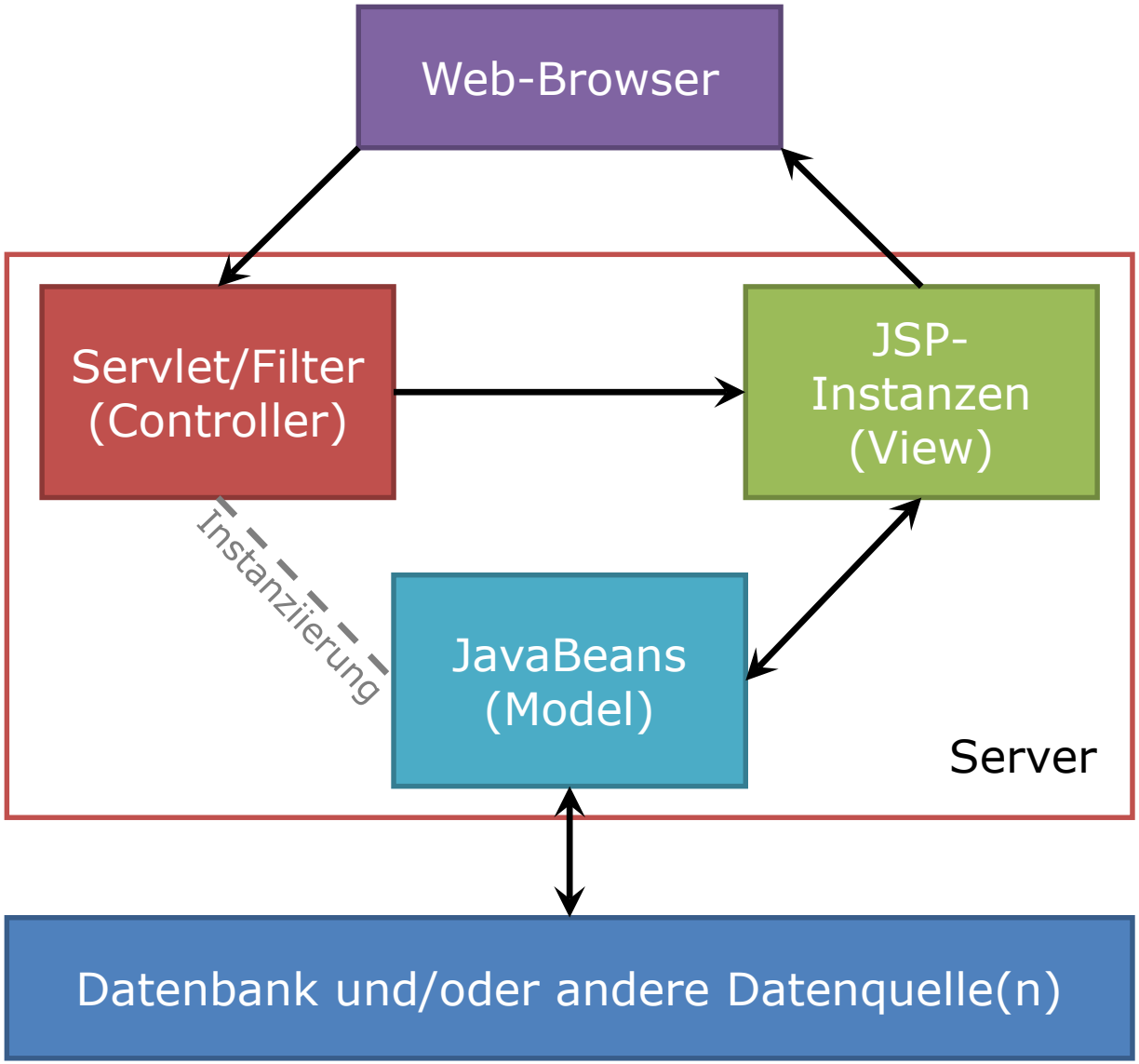
- Server für dynamisch generierte Inhalte
- implementiert die Spezifikationen von
  - Java Servlets
  - Java Server Pages (JSP)
- besteht aus
  - Servlet-Container (Catalina)
  - JSP-Engine (Jasper)
  - Connector-Framework (Coyote)
- zwei Betriebsmodi
  - eigenständiger Web-Server (mittels HTTP-Connector)
  - Applikations-Server (mittels AJP-Connector)
    (i.d.R. mit Load-Balancer und angebunden an httpd, nginx oder IIS)

Java Servlet

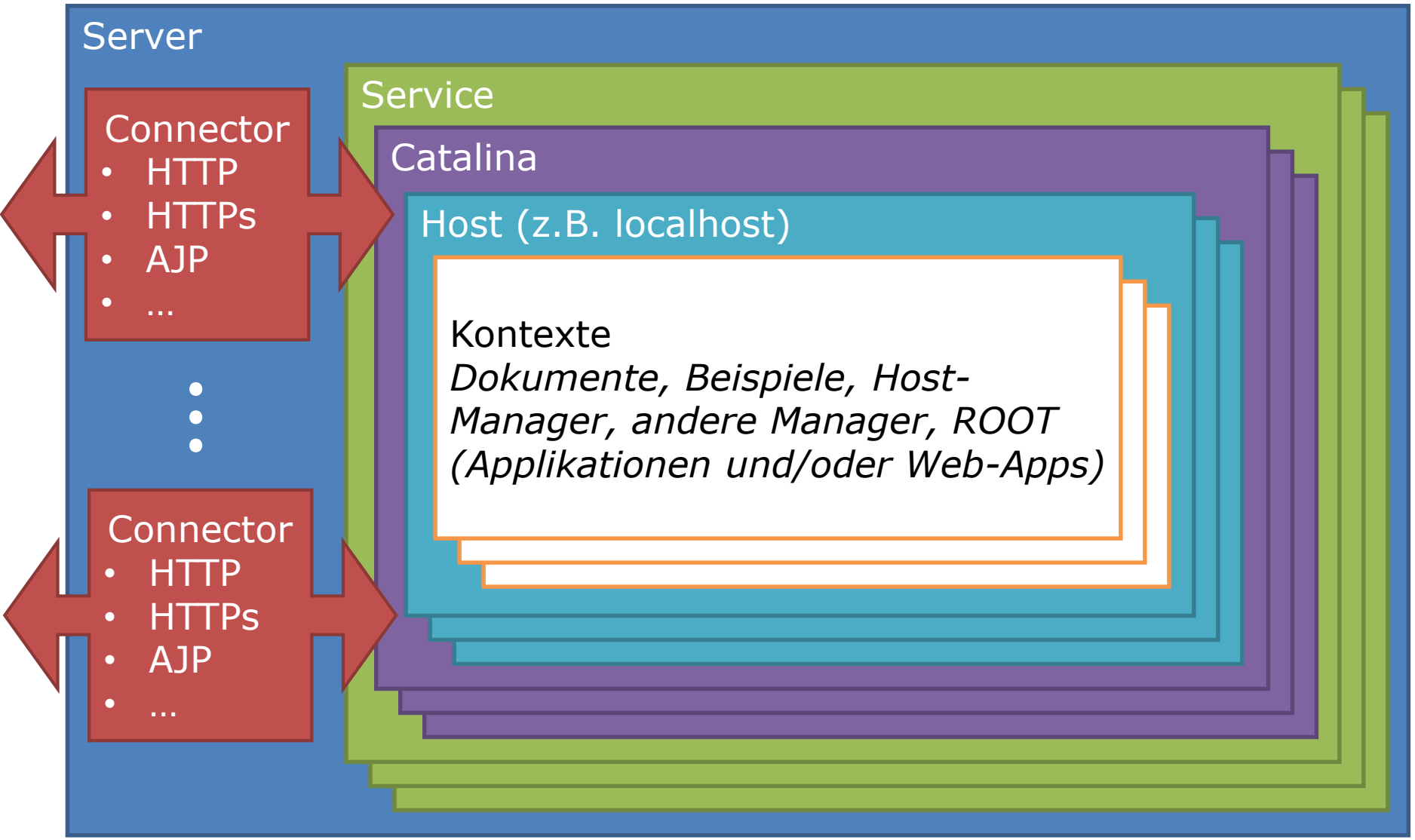


[Frederik Wahl, Public Domain, 2006]

Java Server Page – MVC im Web



Apache Tomcat – Architektur



Apache Tomcat – Beispiel globaler Konfiguration

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
    <Listener className="org.apache.catalina.core.JasperListener" />
    <Listener className="org.apache.catalina.core.AprLifecycleListener" SSLEngine="on" />
    <Listener className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
    <Listener className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
    <Listener className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

    <GlobalNamingResources>
        <Resource name="MyDB" auth="Container" type="org.apache.catalina.UserDatabase"
            description="User database" factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
            pathname="conf/tomcat-users.xml" />
    </GlobalNamingResources>

    <Service name="Catalina">
        <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" />
        <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

        <Engine name="Catalina" defaultHost="localhost">
            <Realm className="org.apache.catalina.realm.LockOutRealm">
                <Realm className="org.apache.catalina.realm.UserDatabaseRealm" resourceName="MyDB"/>
            </Realm>

            <Host name="localhost" appBase="webapps" unpackWARs="true" autoDeploy="true">
                <Valve className="org.apache.catalina.valves.AccessLogValve" directory="logs"
                    prefix="localhost_access_log." suffix=".txt" pattern="%h %l %u %t '%r' %s %b" />
            </Host>
        </Engine>
    </Service>
</Server>
```

Microsoft Internet Information Services (IIS)

- Server für diverse Zwecke
  - Web-Server (dynamische Inhalte; ASP, ASP.NET)
  - Mail-Server
  - Datei- und DAV-Server
  - Applikations-Server (mittels ISAPI-Filter; bindet insb. PHP & JSP ein)
  - Load-Balancer
  - ...
- Kommunikationsprotokolle
  - HTTP, HTTPS, H2
  - POP3, IMAP, SMTP, Exchange
  - FTP, SFTP, FTPs
  - WebDAV
  - NNTP

Achtung!
IIS gibt es nur für ™.

Microsoft IIS – IIS Extensions

Module als Applikationspakete

- FTP Publishing Service (inkl. SSL/TLS-Unterstützung)
- Administration Pack (Web-basierte Administrationsoberfläche) für
  - Authentifikation und Autorisierung für ASP.NET
  - Individualisierung von Fehlerseiten
  - FastCGI-Konfiguration
  - Request-Filtering-Konfiguration
- Application Request Routing
  - Proxy-basiertes Routing-Modul
  - leitet HTTP-Anfragen an Inhalte-Server weiter
- Database Manager (für lokale und externe Datenbanken)
- Media Services (reine Medien- und Rich-Media-Inhalte)
- URL Rewrite Module
- WebDAV

Microsoft IIS Express

- Freeware-Version des IIS
- abgespeckt auf HTTP und HTTPs
- keine Extensions, insb. kein ARR
- sehr leichtgewichtig ggü. httpd (6,5 MB statt 10,3 MB)
  (zum Vergleich nginx: 1,6 MB)
- breite Windows-Unterstützung (lauffähig ab Windows XP)
  (zum Vergleich httpd/nginx: ab Windows 7/8.1)

Microsoft IIS – Beispiel globaler Konfiguration

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <directoryBrowse enabled="true" showFlags="Date, Time, Size, Extension, LongDate" />
        <staticContent>
            <mimeMap fileExtension=".*" mimeType="application/octet-stream" />
        </staticContent>
        <rewrite>
            <rules>
                <rule name="HTTP to HTTPs Redirect" enabled="false" stopProcessing="true">
                    <match url="(.*)" />
                    <conditions logicalGrouping="MatchAny">
                        <add input="{SERVER_PORT_SECURE}" pattern="^0$" />
                    </conditions>
                    <action type="Redirect" url="https://{HTTP_HOST}/" redirectType="Permanent" />
                </rule>
            </rules>
        </rewrite>
    </system.webServer>
</configuration>
```

Microsoft IIS – Beispiel Proxy-Konfiguration

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <rewrite>
            <rules>
                <clear />
                <rule name="ReverseProxyInboundRule1" stopProcessing="true">
                    <match url="(.*)" />
                    <conditions logicalGrouping="MatchAll" trackAllCaptures="false" />
                    <action type="Rewrite" url="http://localhost:4713/{R:1}" />
                    <serverVariables>
                        <set name="REMOTE_ADDR" value="::2" />
                    </serverVariables>
                </rule>
            </rules>
        </rewrite>
    </system.webServer>
</configuration>
```

BERUFSAKADEMIE SACHSEN
STAATLICHE STUDIENAKADEMIE
DRESDEN
UNIVERSITY OF COOPERATIVE EDUCATION

Don't get pawn'd
(Kenn' Deinen Server!)

Beispiel: HPP – multiple URL-Parameter

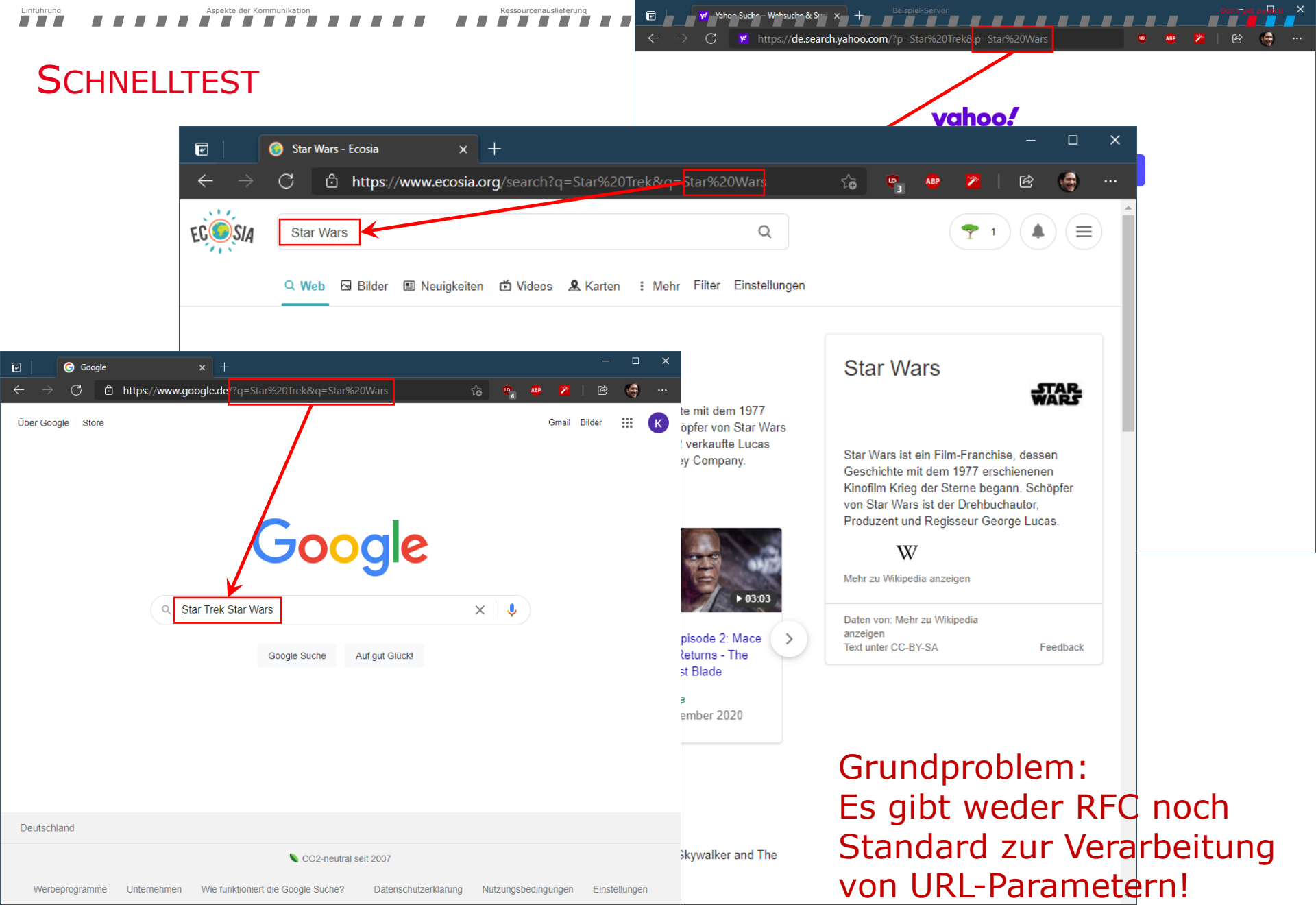
HPP – HTTP Parameter Pollution

```
https://example.com/?p1=Star%20Trek&p1=Star%20Wars
```

Frage: *Welchen Wert hat der Parameter p1?*

- Server verarbeiten URL-Parameter unterschiedlich
  - Tomcat: nur erstes Auftreten eines Parmeters
  - httpd: nur letztes Auftreten eines Parameters
  - IIS: Konkatenation aller Auftreten eines Parameters
- kann für Angriffe verwendet werden

SCHNELLTEST



Grundproblem:
Es gibt weder RFC noch Standard zur Verarbeitung von URL-Parametern!

Ausnutzung als Vektor (1/2)

Problem:

Klartext &

Teilen bei Facebook:

`https://facebook.com/sharer.php?u=cataas.com/cat&quote=Süß&Katze`

URL des geteilten Inhalts

Titel des geteilten Inhalts

Ausnutzung als Vektor (2/2)

Teilen bei Facebook:

`https://facebook.com/…`
`…sharer.php?u=cataas.com/cat&quote=Süß%26Katze&u=hacker.com`

Wurzel des Bösen!