Freie Universität Berlin

Web APIs (REST)

Markus Luczak-Rösch
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
markus.luczak-roesch@fu-berlin.de

Einige Folien basieren auf Vorlagen von R. Tolksdorf.

Motivation: „The **Web** as a platform“

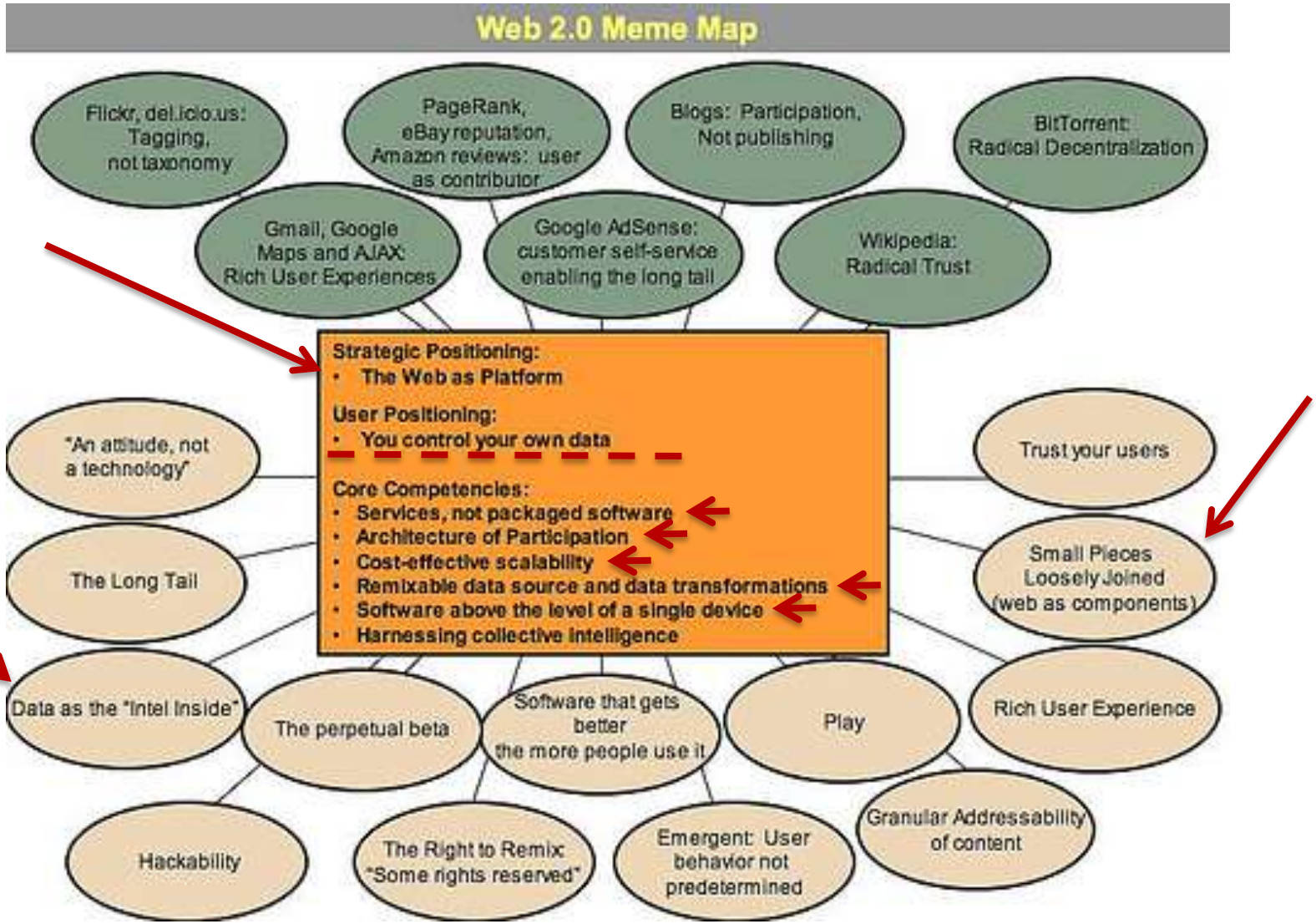
Das Web intuitiv – einfach und unendlich

Freie Universität Berlin

- es ist einfach dazu beizutragen (allerdings nicht erst seit „Web x.0“)
- es ist einfach zu konsumieren
- potenziell unendlich groß (w.r.t. #Benutzer, #Inhalte, …)
- unendlich heterogen

- Daten-Silos vs. people claiming „I want my data back“
- Always On – zu jeder Zeit an jedem Ort
- Technologie wider den Information Overload
- Daten aggregieren und integrieren

„Web 2.0 Meme“



Quelle: http://oreilly.com/web2/archive/what-is-web-20.html

- interdisziplinäre wissenschaftliche Auseinandersetzung mit dem Phänomen Web
  - Technologie
  - Nutzer und Nutzung
  - Emergenz

Inhalt der heutigen Vorlesung

- REST-Prinzipien
- HTTP und REST
- Ressourcen, Repräsentationen und Content Negotiation
- REST-Service Beispiel

- **Re**presentational **S**tate **T**ransfer
  - ist ein Architekturstil für **netzbasierte Systeme** („Network-based Application Architectures“)
  - Bekanntestes Beispiel: Das Web!
  - Literaturhinweis: Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-Based Software Architectures.* Ph.D. Dissertation. University of California, Irvine. AAI9980887.

REST Prinzip 1 – Client-Server

Freie Universität Berlin

- abgeleitet vom Prinzip zur Aufteilung von Aufgaben („separation of concerns“)
  - erlaubt unabhängige Evolution der Einzelkomponenten
  - Beispiel:
    - Rendering von HTML im Web-Browser (Client) – Serven von HTML-Seiten auf einem Web-Server (Server)

REST Prinzip 1 – Client-Server

- abgeleitet vom Prinzip zur Aufteilung von Aufgaben („separation of concerns“)

  \+ UI Portabilität
  \+ Skalierbarkeit durch Einfachheit
  \+ **unabhängige Evolution**

  – (Abhängigkeit vom Netz)

REST Prinzip 2 – Stateless

- jede Nachricht enthält alle notwendigen Informationen, die dem Empfänger die Verarbeitung erlauben
  - kein gespeicherter Kontext am Server
  - Sitzungszustand beim Client
    - Beispiele:
      - Cookies
      - Session IDs

REST Prinzip 2 – Stateless

- jede Nachricht enthält alle notwendigen Informationen, die dem Empfänger die Verarbeitung erlauben

  + Sichtbarkeit (alle Informationen in Anfrage)
  + Verlässlichkeit (bessere Erholung von Fehlern in Teilsystemen)
  + Skalierbarkeit (Server kann schnell Ressourcen wieder frei geben)

  – evtl. erhöhter Netzverkehr wegen Datenredundanz in Anfragesequenzen

REST Prinzip 3 – Cache

- Antworten auf Anfragen implizit oder explizit als cachable oder non-cachable klassifizieren
  - Client darf cachable-Antworten für spätere Anfragen wiederverwenden

REST Prinzip 3 – Cache

- Antworten auf Anfragen implizit oder explizit als cachable oder non-cachable klassifizieren

  \+ Vermeidung unnötiger Interaktionen erhöht Effizienz und Skalierbarkeit

  – Verlässlichkeitsprobleme, wenn Cache veraltete Daten hält

REST Prinzip 4 – Uniform Interface

- Vereinfachung der Systeminfrastruktur und hohe Sichtbarkeit von Interaktionen durch standardisierten Zugriff auf Komponenten
  - Ressourcenidentifikation
    - universelle Syntax für Identifier
    - Identifikation von „Dingen“ (Things)
  - Manipulation von Repräsentationen
    - wohldefinierte Aktionen auf einer Sequenz von Bytes + Metadaten (=Repräsentation), die den aktuellen oder gewünschten Zustand einer Ressource darstellt
  - selbstbeschreibende Nachrichten
  - Hypermedia

REST Prinzip 4 – Uniform Interface

- Vereinfachung der Systeminfrastruktur und hohe Sichtbarkeit von Interaktionen durch standardisierten Zugriff auf Komponenten
  - Ressourcenidentifikation
  - Manipulation von Repräsentationen
  - selbstbeschreibende Nachrichten
    - die Semantik von Nachrichten ist für alle verarbeitenden Komponenten (Mittler) sichtbar
    - Mittler können Inhalte verändern
  - Hypermedia
    - alle Inhalte UND Informationen zum Zustandsübergang (Hyperlinks) werden an den Client weitergegeben

REST Prinzip 4 – Uniform Interface

- Vereinfachung der Systeminfrastruktur und hohe Sichtbarkeit von Interaktionen durch standardisierten Zugriff auf Komponenten

  \+ Einfachheit

  \+ Sichtbarkeit von Interaktionen

  – standardisierte Informationsübertragung kann ineffizient sein im Vergleich mit anwendungsspezifischer Übertragung

REST Prinzip 5 – Layered System

- Unabhängigkeit der einzelnen Komponenten durch beschränkte Sicht auf das hierarchisch geschichtete Gesamtsystem
  - Komponenten „sehen“ nur bis zum Interaktionspartner

REST Prinzip 5 – Layered System

- Unabhängigkeit der einzelnen Komponenten durch beschränkte Sicht auf das hierarchisch geschichtete Gesamtsystem

  + Kapselung und Transparenz
  + Vereinfachung der einzelnen Komponenten
  + Load Balancing

  – Interaktionsoverhead
  – Latenz

REST Prinzip 6 – Code-on-Demand

- Erweiterbarkeit des Systems durch Download von Code nach dem Deployment
  - Beispiele:
    - Applets
    - Scripte
  - optionales Prinzip, weil es die Sichtbarkeit von Interaktionen reduziert

REST Prinzip 6 – Code-on-Demand

- Erweiterbarkeit des Systems durch Download von Code nach dem Deployment

  \+ Reduzierung der „Vorimplementierung“ am Client
  \+ Erweiterbarkeit

  – Reduzierung der Sichtbarkeit von Interaktionen

Das Web technologisch

- HTTP
  - Stateless Client-Server
  - Caching
  - Uniform Interface: standardisierte Manipulation von Repräsentationen via GET, PUT, POST, DELETE
  - Layered System
- HTTP-URIs
  - Uniform Interface: Ressourcenidentifikation
- Hypertext und Hyperlinks
  - Uniform Interface: standardisierte Repräsentation von Ressourcenzustand und Zustandsübergängen

Recap – TCP

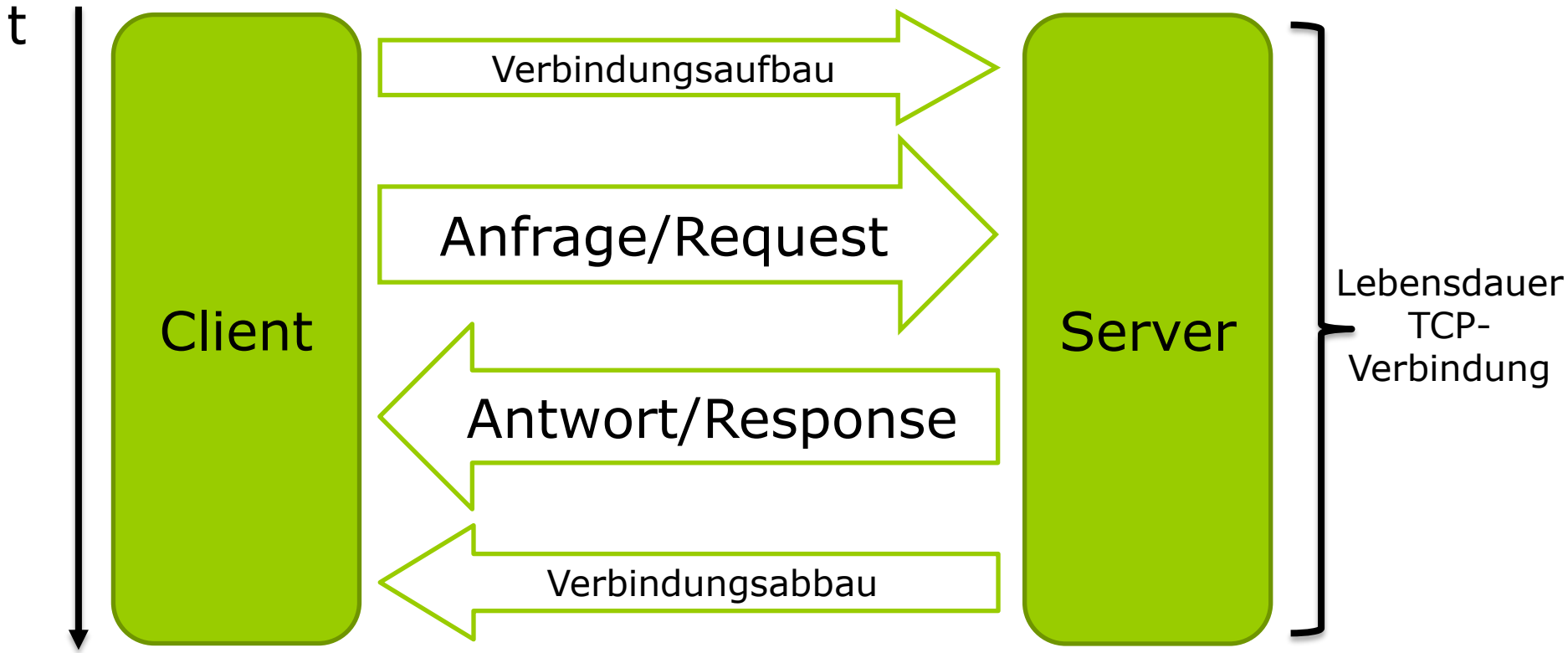
- verbindungsorientiert
- Kommunikation über Sockets (Port + IP-Adresse)
- Lauschen, Verbindungsaufbau, Datenaustausch (Kommunikations-Socket), Verbindungsabbau
- Flusssteuerung und Fehlerkorrektur
- …

Recap – DNS

- Zuordnung Rechnername zu IP-Adresse
  - google.de → 173.194.69.94
  - google.com → 173.194.69.139
  - markus-luczak.de → 83.169.29.65
  - loomp.org → 83.169.29.65

- Hypertext Transfer Protocol
  - Transfer von Informationen zwischen Web-Servern und Clients
  - Port 80 ist für HTTP reserviert
  - Transportprotokoll ist TCP
  - textbasiert
  - Literaturhinweis: R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach und T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. RFC 2616, http://www.ietf.org/rfc/rfc2616.txt

- zustandslos
- Request-Response-Verfahren



abgeleitet von: R. Tolksdorf

Client

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 … Firefox/10.0.3
Host: markus-luczak.de:80
Accept: */*
```

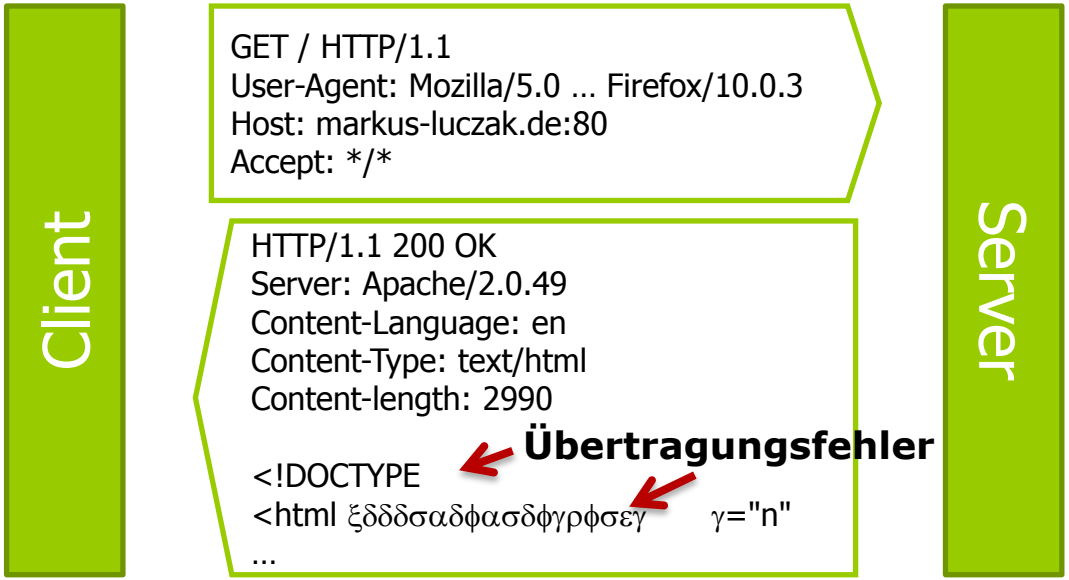
```
HTTP/1.1 200 OK
Server: Apache/2.0.49
Content-Language: en
Content-Type: text/html
Content-length: 2990

<!DOCTYPE html>
<html xml:lang="en"
...
```

Server

abgeleitet von: R. Tolksdorf

Zwischenfragen

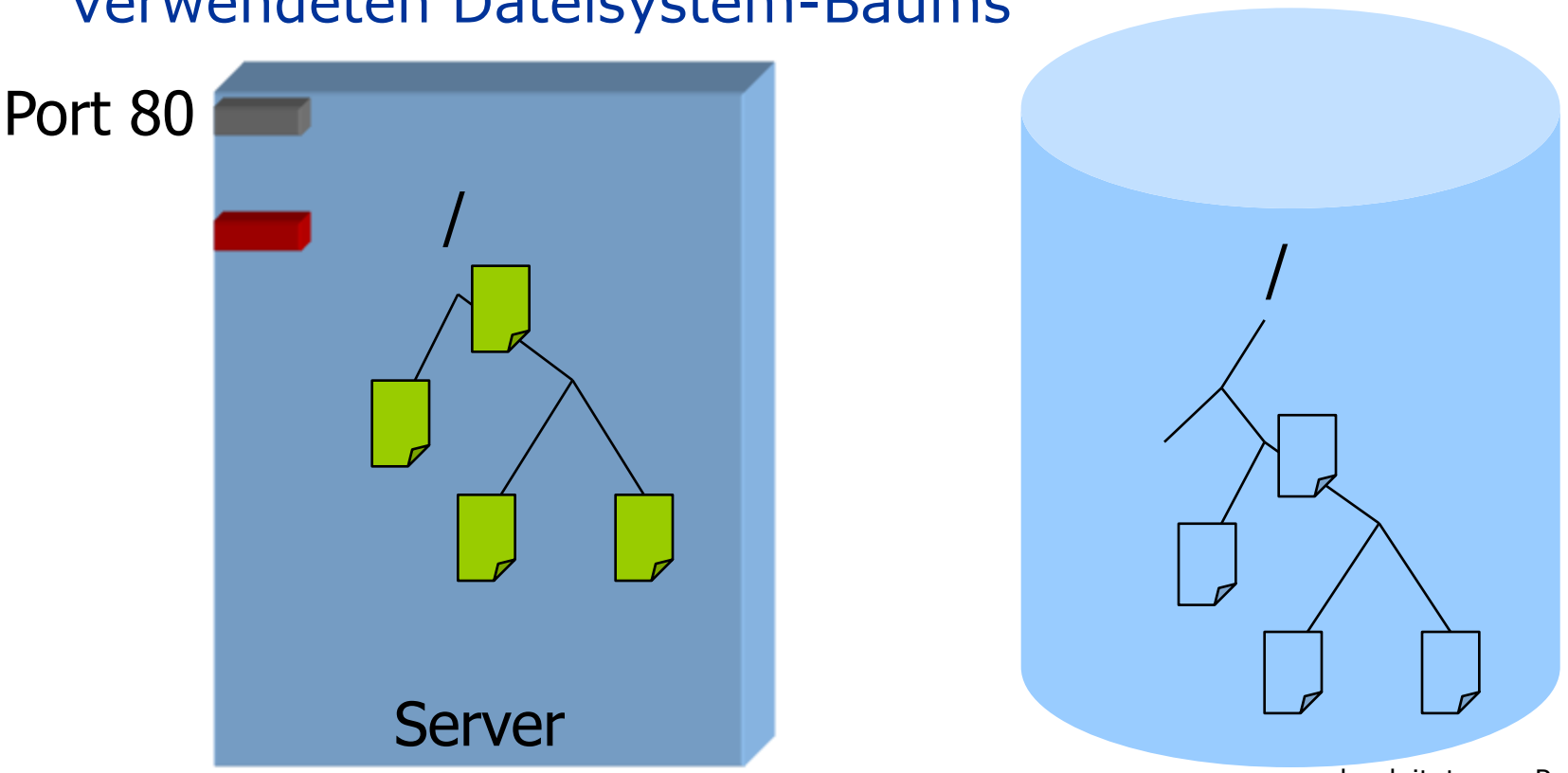
- Warum TCP als Transportprotokoll?
  - Verlässlichkeit



- Warum DNS?
  - Virtual Hosting

HTTP Web-Server

- Web-Server wartet auf Verbindungen
- beantwortet Nachfragen nach Ressourcen bzgl. des Web-Server-Verzeichnisbaums mit Dateien des verwendeten Dateisystem-Baums



abgeleitet von: R. Tolksdorf

HTTP-Anfrage

- Anfrage besteht aus
  - Anfragemethode
  - Anfragebeschreibung durch Kopfzeilen
  - Leerzeile
  - eventuell Inhalt

- Format:
  - *Methode Ressource* HTTP/*x.y*
    Host: *Domain-Name* ← Pflicht, wenn x=1 && y=1

- Beispiel:
  - GET /test.html HTTP/1.1
    Host: markus-luczak.de

abgeleitet von: R. Tolksdorf

Weitere Anfragemethoden in HTTP

- PUT
  - Abspeichern einer Informationseinheit auf einem Server
  - PUT /index.html HTTP/1.1
  - Beantwortet mit Code, Kopfzeilen
- POST
  - Hinzufügen von Informationen zu einer Informationseinheit
  - POST /speichere.cgi HTTP/1.1
    Daten daten daten
  - Beantwortet mit Code, Kopfzeilen, eventuell Inhalt
- DELETE
  - Löschen einer Informationseinheit auf einem Server
  - DELETE /index.html HTTP/1.1
  - Beantwortet mit Code, Kopfzeilen

abgeleitet von: R. Tolksdorf

Weitere Anfragemethoden in HTTP

- TRACE
  - Server schickt erhaltenen Inhalt zurück
- CONNECT
  - Sagt Proxy, dass er Tunnel aufbauen soll
  - Tunnel: Verpacken eines Protokolls A in ein anderes Protokoll B, so dass die Anwendung A spricht, aber B benutzt

Weitere Anfragemethoden in HTTP

Freie Universität Berlin

- OPTIONS
  - Informationen über Fähigkeiten des Servers
  - Überträgt alle Allow-Kopfzeilen
  - Anfrage:
    OPTIONS * HTTP/1.1
    Host: www.inf.fu-berlin.de

  - Antwort:
    HTTP/1.1 200 OK
    Date: Tue, 25 Nov 2003 11:29:16 GMT
    Server: Apache/1.3.26 Ben-SSL/1.48 (Unix) Debian GNU/Linux mod_perl/1.26 PHP/4.1.2
    Content-Length: 0
    Allow: GET, HEAD, OPTIONS, TRACE

abgeleitet von: R. Tolksdorf

Anfrage Kopfzeilen

- Host: *Name*
  Aus der URL ermittelter Name des Rechners von dem angefordert wird. *Einzige Pflichtkopfzeile in HTTP 1.1*
- If-Modified-Since: *Datum*
  Änderung der Informationseinheit seit *Datum*
  - Ja: 200 und Inhalt schicken
  - Nein: 304 und Inhalt nicht schicken
- If-Unmodified-Since: *Datum*
  Änderung der Informationseinheit seit Datum
  - Ja: 412 und nicht verarbeiten
  - Nein: Normal verarbeiten (als sei If-Unmodified-Since: nicht vorhanden)

abgeleitet von: R. Tolksdorf

Anfrage Kopfzeilen

- Max-Forwards: *Anzahl*
  Wie oft ein OPTIONS oder TRACE weitergeleitet werden darf
- Range: *Bytebereich*
  Nur Teile der Information anfordern, Antwort ist dann 216
  Range: bytes=500-999
- Expect: *Token*
  Client erwartet bestimmte Eigenschaften von Server/Proxy
  (Falls nicht: 417)

abgeleitet von: R. Tolksdorf

Anfrage Kopfzeilen

- From: *Mailadresse*
  Nutzer
- User-Agent: *Produkt/Version*
  Browser z.B.
  (Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
- Referer: *URL*
  Seite auf der ein Link auf die angeforderte Seite stand
- Authorization: *Nachweis*
  Autorisierungsnachweis falls mit 401 angefordert
- Authorization: username="Mufasa",
  realm="testrealm@host.com",
  response="6629fae49393a05397450978507c4ef1"
- Proxy-Authorization: *Nachweis*
  Autorisierungsnachweis für Proxy, falls mit 407 angefordert

abgeleitet von: R. Tolksdorf

Angeforderte Medienart

- Accept: *Medienart/Variante;* q=*Qualität;* mxb=*Maximale Größe*
  - Accept: text/postscript; mxb=200000

```
GET / HTTP/1.1
User-Agent: Mozilla/5.0 … Firefox/10.0.3
Host: markus-luczak.de:80
Accept: */*
```

abgeleitet von: R. Tolksdorf

Exkurs: Inhaltstypen

- Per HTTP können beliebige Inhalte transportiert werden, nicht nur HTML
- Multipurpose Internet Mail Extensions MIME (RFC 2045, RFC 2046) definiert ein Schema zur eindeutigen Benennung durch einen Inhaltstypen
- In HTTP in Kopfzeile Content-Type
- Format: *Typ/Untertyp*
  - text/html
  - image/jpeg
  - vnd.motorola.video

```
HTTP/1.1 200 OK
Server: Apache/2.0.49
Content-Language: en
Content-Type: text/html
Content-length: 2990

<!DOCTYPE html>
<html xml:lang="en"
...
```

abgeleitet von: R. Tolksdorf

MIME Typen

- Acht Typen:
  - text: Text
    - text/plain, text/html, text/rtf, text/vnd.latex-z
  - image: Grafiken
    - image/png, vnd.microsoft.icon
  - video: Bewegtbilder
    - video/mpeg, video/quicktime, video/vnd.vivo
  - audio: Audiodaten
    - audio/G726-16 , audio/vnd.nokia.mobile-xmf
  - application: binäre und/oder anwendungsspezifische Daten
    - application/EDIFACT, application/vnd.ms-powerpoint
  - multipart: mehrteilige Daten
    - multipart/mixed
  - message: Nachrichten
    - message/rfc822
  - model: Daten
    - model/vrml

abgeleitet von: R. Tolksdorf

MIME Typen

- MIME-Typen werden von der Internet Corporation for Assigned Names and Numbers IANA verwaltet
- http://www.iana.org/assignments/media-types/
- Verarbeiten eines bestimmten Medientyps nach Erhalt:
  - Teil der Anwendung (siehe auch: javax.mail.internet.MimeMessage)
  - eventuell Unterstützung durch Betriebssystem
- Ermittlung des MIME-Typs für eine Datei:
  - Ableitung aus Endung (javax.activation.MimetypesFileTypeMap)
  - Ableitung aus Inhalt der Datei

abgeleitet von: R. Tolksdorf

Content Negotiation – Anfrage

- Auswahl passender Information bezüglich der Dimensionen
  - Medienart (`Accept: text/html, text/plain`)
  - Sprache (`AcceptLanguage: en-us;q=0.75,en;q=0.5;*;q=0.25`)
  - Encoding(`Accept-Encoding: compress;q=0.5, gzip;q=1.0`)
  - Charset (`AcceptCharset: iso-8859-1,utf-8;q=0.75,*;q=0.5`)
  - angegebene Qualitätsmaße
- Server-abhängige Implementierungen
  - z.B. Schema über Dateinamen:
    - foo.en.html
    - foo.html.en
    - foo.en.html.gz

abgeleitet von: R. Tolksdorf

HTTP-Antwort

- Antwort besteht aus
  - Antwortcode
  - Antwortbeschreibung durch Kopfzeilen
    - Allgemeine Beschreibungen
    - Antwortspezifische Beschreibungen
    - Beschreibung eventuell beiliegenden Inhalts
  - Leerzeile
  - eventuell Inhalt
- Beispiel:

```
HTTP/1.1 200 OK
Server: Apache/2.0.49
Content-Language: en
Content-Type: text/html
Content-length: 2990

<!DOCTYPE html>
<html xml:lang="en"
...
```

abgeleitet von: R. Tolksdorf

Antwort Codes

- 200-er Codes: Erfolgreiche Ausführung
  - 200 – OK
    GET, HEAD, POST, TRACE erfolgreich, Antwort anbei
  - 201 – Created
    Erfolgreiches PUT oder POST
  - 202 – Accepted
    Für spätere Ausführung vermerkt
  - 203 – Non-Authoritative Information
    Metainformationen in Kopfzeilen stammen von Dritten
  - 204 – No Content
    Anfrage verarbeitet, kein Antwortinhalt notwendig
  - 205 – Reset Content
    Anfrage verarbeitet, Ansicht erneuern
  - 206 – Partial Content
    GET mit Teilanforderung erfolgreich, Teilantwort anbei

abgeleitet von: R. Tolksdorf

Antwort Codes

- 300-er Codes: Weitere Aktion des Client zur erfolgreichen Ausführung notwendig
  - 300 - Multiple Choices
    Verschiedene Versionen erhältlich, Accept-Kopfzeile nicht eindeutig
  - 301 - Moved Permanently
    Verschoben (Location und URI Kopfzeilen geben Auskunft)
  - 302 - Found Moved Temporarily
    Verschoben (Location und URI Kopfzeilen geben Auskunft)
  - 303 See Other
    Andere Resource laden (Location und URI Kopfzeilen geben Auskunft)
  - 304 - Not Modied
    Bei GET mit If-Modified-Since Kopfzeile
  - 305 Use Proxy
    Muss durch Proxy angesprochen werden (Adresse in Location)
  - 307 Temporary Redirect
    Umleitung bei GET, HEAD

abgeleitet von: R. Tolksdorf

Antwort Codes

- 400-er Codes: Nicht erfolgreich, Fehler bei Client
  - 400 - Bad Request
    Falsche Anfragesyntax
  - 401 - Unauthorized
    Passwort notwendig
  - 403 – Forbidden
    Ohne Angabe von Gründen verweigert
  - 404 - Not Found
    Nicht auffindbar
  - 405 - Method Not Allowed
    Methode für die Resource nicht zugelassen
  - 406 - Not Acceptable
    Information vorhanden aber nicht passend zu Accept-Kopfzeilen
  - 407 Proxy Authentication Required
    Zuerst Authentifizierung bei Proxy nötig, der Proxy-Authenticate Kopfzeilen mit schicken muss
  - 408 - Request Timeout
    Timeout bei Übermittlung der Anfrage

abgeleitet von: R. Tolksdorf

Antwort Codes

- 409 Conflict
  Methode steht in Konflikt mit Zustand des Servers, Client kann Konflikt aufheben
- 410 Gone
  Permanent und absichtlich nicht auffindbar
- 411 Length Required
  Content- Length Kopfzeile ist notwendig
- 412 Precondition Failed
  Bedingungen der Anfrage (in Kopfzeilen) unerfüllbar
- 413 Request Entity Too Large
  Anfrage zu groß
- 414 Request-URI Too Long
  URI zu lang
- 415 Unsupported Media Type
  Unbekanntes Inhaltsformat
- 416 Requested Range Not Satisfiable
  Teilanforderung falsch beschrieben
- 417 Expectation Failed
  Expect Kopfzeile unerfüllbar

abgeleitet von: R. Tolksdorf

Antwort Codes

- 500-er Codes: Nicht erfolgreich, Fehler bei Server
  - 500 - Internal Server Error
  - 501 - Not Implemented
    Angeforderte Methode nicht unterstützt
  - 502 - Bad Gateway
    Weiterer benutzer Server nicht erreichbar
  - 503 - Service Unavailable
    Server kann Dienst gerade nicht erbringen (Retry-After Kopfzeile)
  - 504 - Gateway Timeout
    Weiterer benutzter Server antwortet nicht rechtzeitig
  - 505 HTTP Version Not Supported
    Unbekannte HTTP Version

abgeleitet von: R. Tolksdorf

Antwort Kopfzeilen

- Server: *Produkt*
  Server-Produkt
  Server: CERNb-HTTPD/3.0 libwww/2.17
- Accept-Ranges: *Token*
  Inwiefern der Server Teilübertragungen unterstützt
  Accept-Ranges: bytes
  Accept-Ranges: none
- Retry-After: *Datum*
  Bei 503: Zeitpunkt zur Wiederholung der Anfrage
  Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
  Retry-After: 120
- Age: *Sekunden*
  Geschätztes Alter der Resource

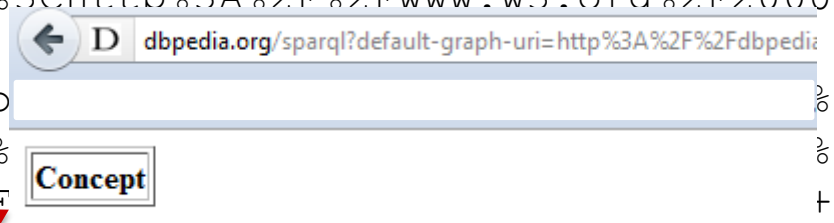
abgeleitet von: R. Tolksdorf

Antwort Kopfzeilen

- Location: *URI*
  Adresse unter der Resource aufzufinden ist
  Bei 201: Adresse der neu geschaffenen Resource
  Bei 3xx: URI für Umlenkung
- WWW-Authenticate: *Aufgabe*
  Bei 401: Client muss sich gegenüber Server ausweisen
- Proxy-Authenticate: *Aufgabe*
  Bei 407: Client muss sich gegenüber Proxy ausweisen

Wählerisch mit Antwortcodes?

```
xxx.xxx.xxx.xxx - - [21/Sep/2009:00:00:00 -0600]
        "GET /page/Jeroen_Simaeys HTTP/1.1"
        200 26777 "" "msnbot/2.0b (+http://search.msn.com/msnbot.htm)"
xxx.xxx.xxx.xxx - - [21/Sep/2009:00:00:00 -0600]
        "GET /resource/Guano_Apes HTTP/1.1"
        303 0 "" "Mozilla/5.0 (compatible; Googlebot/2.1;
        +http://www.google.com/bot.html)"
xxx.xxx.xxx.xxx - - [21/Sep/2009:00:00:01 -0600]
        "GET
        /sparql?query=PREFIX+rdfs%3A+%3Chttp%3A%2F%2Fwww.w3.org%2F2000
        %2F01%2Frdf-
        schema%23%3E%0APREFIX+wgs84_po
        2F2003%2F01%2Fgeo%2Fwgs84_pos%
        3Flong%0AFROM+%3Chttp%3A%2F%2F
        ++++++%3Chttp%3A%2F%2Fdbpedia.
        3Alabel+%3Flabel+.%0A++++++++O
        p%3A%2F%2Fdbpedia.org%2Fresour
        3Flat+.%0A++++++++++%3Chttp%3A%2F%2Fdbpedia.org%2Fresource%2
        FQueens%3E+wgs84_pos%3Along+%3Flong%0A++++++++%7D%0A%7D
        HTTP/1.1" 200 1844 "" ""
```



Bsp.: Status Code 204 „No Content“, falls Anfrage ohne Ergebnis, erleichtert die Auswertung.

Inhalts-Kopfzeilen

- Content-Encoding: *Kodierung*
  Kodierung des Inhalts
  - deflate, gzip, …
- Content-Type: *Medienart*
  Medientyp des Inhalts
  - text/html, image/gif, ..
- Content-Language: *Sprachkürzel*
  Sprache des Inhalts
  - de, en, en-US
- Content-Length: *Länge*
  Länge des Inhalts in Byte
- Content-Range: *Range*
  Beschreibung des Ausschnitts bei Teilanforderung

abgeleitet von: R. Tolksdorf

Inhalts-Kopfzeilen

- Content-Location: *URI*
  Inhalt ist in Antwort, der Inhalt steht aber auch an einer anderen URI
- Content-MD5: *MD5Checksum*
  Message Digest für Inhalt zur Integritätsprüfung

- Expires: *Datum*
  Kann nach *Datum* aus Caches gelöscht werden
- Last-Modified: *Datum*
  Letzte Änderung

abgeleitet von: R. Tolksdorf

Allgemeine Kopfzeilen (in Anfrage und Antwort)

- Date: Tue, 15 Nov 1994 08:12:31 GMT
  Datum des Abschickens der Anfrage im RFC 1123 Format
- Connection: close
  Verbindung nach Ergebnisübermittlung abbauen
- Cache-Control: *Direktive*
  Steuert das Caching von Anfragen und Antworten
  - no-cache: Antwort darf nicht zur Beantwortung anderer Anfragen genutzt werden
  - no-store: Antwort- oder Anfragemitteilungen dürfen nicht gespeichert werden
  - weitere: max-age, max-stale, min-fresh, no-transform, only-if-cached, public, private, must-revalidate, proxy-revalidate, s-maxage
- Pragma: no-cache
  Entspricht Cache-Control: no-cache

abgeleitet von: R. Tolksdorf

Allgemeine Kopfzeilen (in Anfrage und Antwort)

Freie Universität Berlin

- Transfer-Encoding: *Encoding*
  Wie die Mitteilung für den Transfer kodiert wurde
  - chunked: Mitteilung in Teilen geschickt, Zeichenanzahl in initialer Hexzahl

```
>java HttpGetClient11 focus.msn.de
java HttpGetClient11 focus.msn.de
HTTP/1.1 200 OK
Date: Fri, 25 Nov 2005 13:20:01 GMT
Server: Apache
set-cookie: NGUserID=11329248012594; path=/; domain=.msn.de;
   expires=fri, 10-aug-2012 16:48:59 gmt
Transfer-Encoding: chunked
Content-Type: text/html

2e96
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
   <html> <head>
<title>FOCUS Online in Kooperation mit MSN  Homepage</title>
   <meta
```

  - identity: Mitteilung unkodiert geschickt
  - gzip, compress, deflate: Komprimierte Übertragung

abgeleitet von: R. Tolksdorf

Allgemeine Kopfzeilen (in Anfrage und Antwort)

- Via: *Protokollversion Host …*
  Weg der Nachricht, z.B. Via: 1.0 fred, 1.1 nowhere.com (Apache/1.1)
- Upgrade: *Protokoll*
  Wunsch nach Verwendung eines neueren Protokolls
  z.B.: Upgrade: HTTP/2.0
- Trailer: *Trailer-Header*
  Nach dem Inhalt folgen weitere Kopfzeilen geschickt
- Warning: *Freitext*
  Zusätzlicher Hinweis

HTTP-Authentication

- Ressourcen im Web können Zugriffsschutz tragen
- Interaktion zum Abruf
  - Normales GET
  - Antwort 401 und WWW-Authenticate: Header, der Nachweis in unterschiedlichen Schemata anfordert
  - Weiteres GET mit Authorization: Header, der je nach Schema Parameter trägt
  - Antwort 200

abgeleitet von: R. Tolksdorf

HTTP-Authentication

- Beispiel am Apache-Web-Server
  - Basic Authentication mit .htaccess:
    AuthUserFile /vhome/www/htdocs/meineseite/.htpasswd
    AuthGroupFile /dev/null
    AuthName "Zugang zur geschützten Seite"
    AuthType Basic
    require user [Nutzername]
  - .htpasswd:
    - [Nutzername]:[Passworthash]
      - erzeugt mit
        /usr/apache/bin/htpasswd -c .htpasswd [Nutzername]

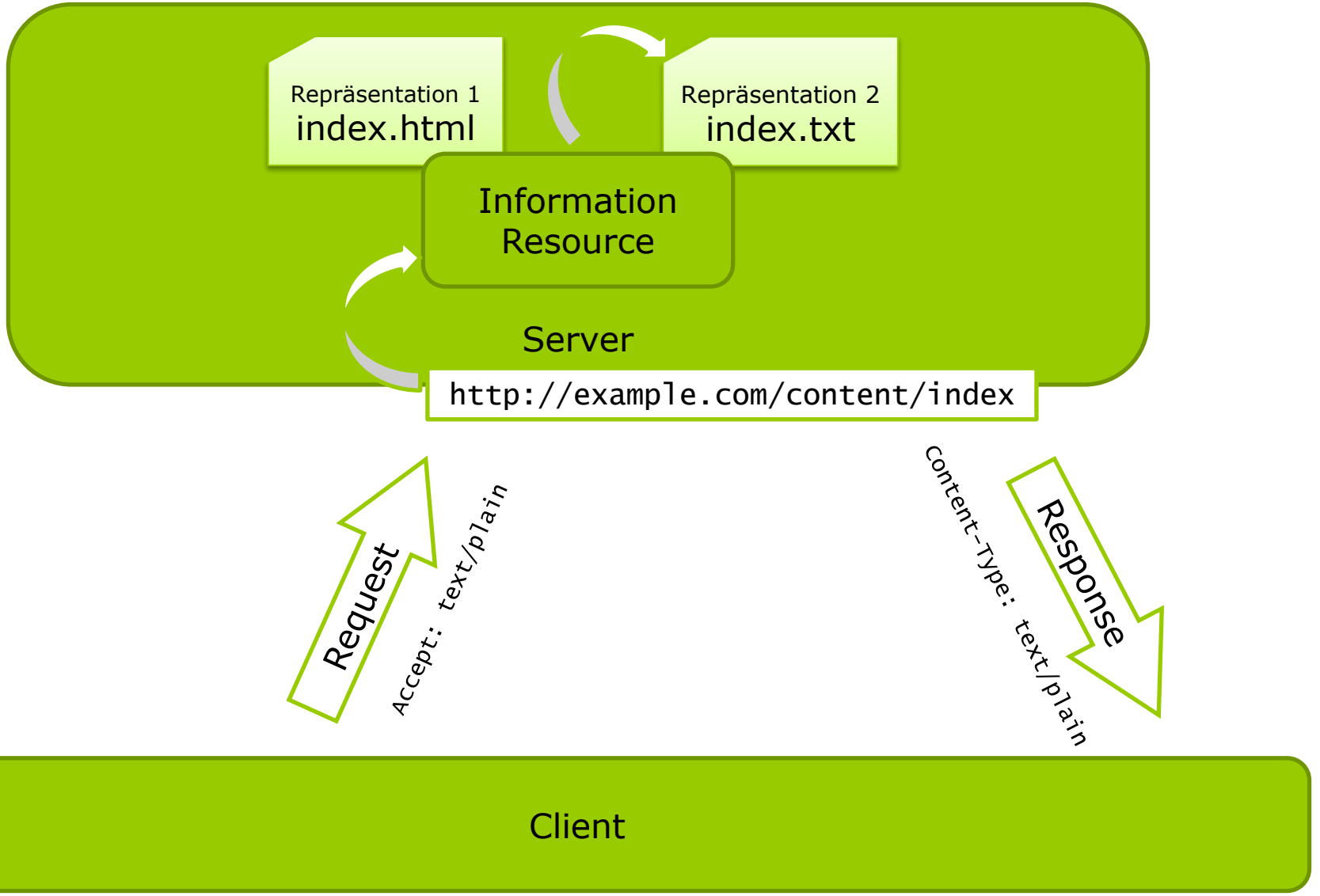
abgeleitet von: R. Tolksdorf

Ressourcen und Repräsentationen

- Information Resources
  - URI-identifizierte Ressourcen, deren Repräsentation in einer Nachricht elektronisch übertragen werden kann
  - eine bestimmte Repräsentation anzufordern und zu erhalten nennt man **Derefrenzierung**

- Non-Information Resources
  - abstrakte Konzepte wie z.B. in Ontologien modelliert
    - Beispiel: Meter
      Ich kann lediglich andere Information Resources ausliefern, die „Meter" beschreiben aber eben keine Repräsentation von Meter sind.
  - URI-identifizierbar aber nicht dereferenzierbar

Content Negotiation – Req.-Resp.

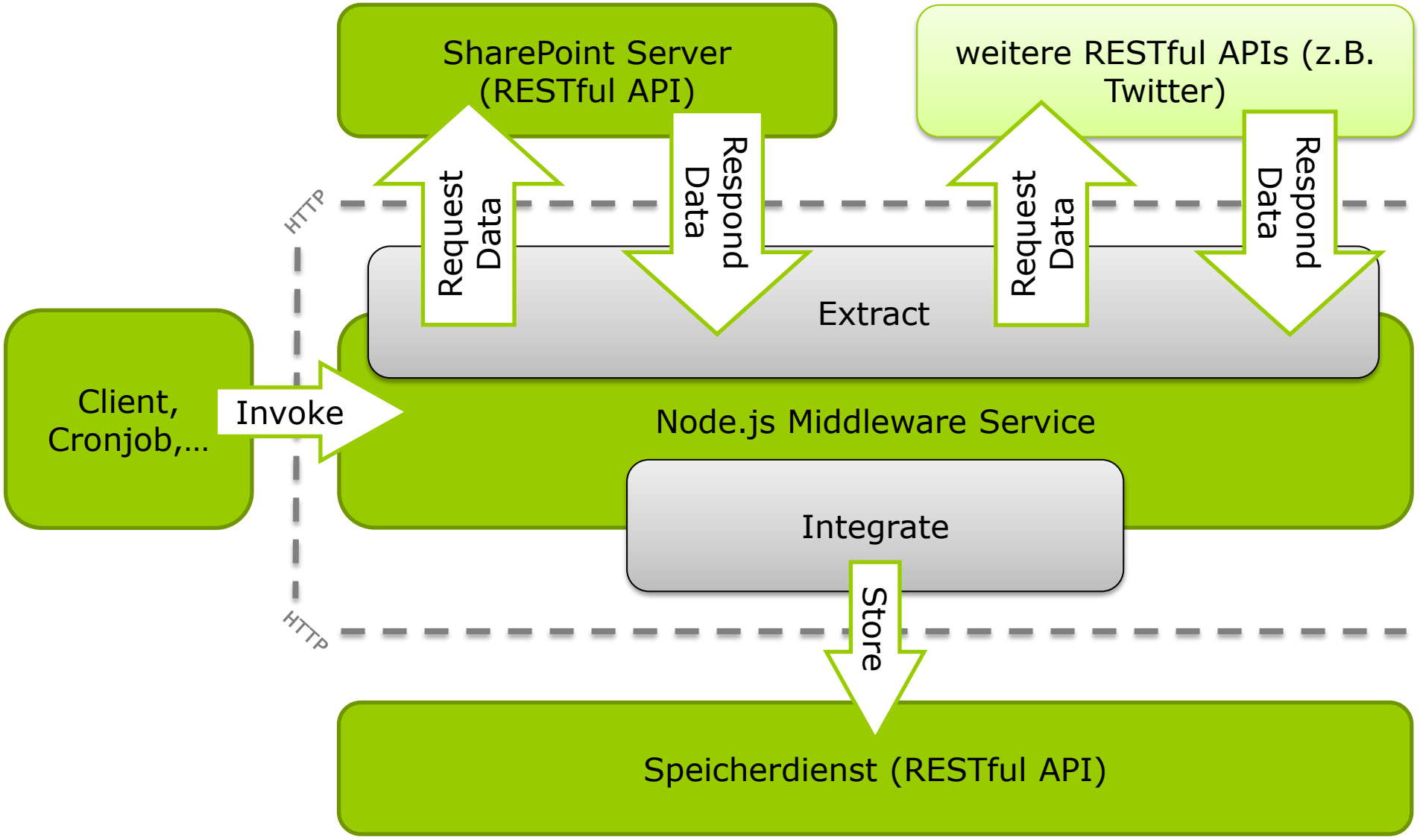


- **J**ava**S**cript **O**bject **N**otation

  - Datenstrukturen als Zeichenkette
    - Objekt: Liste von Eigenschaften zwischen { und }
    - Eigenschaft: [Schlüssel]:[Wert]
    - Array: \[[.]*\]
    - Zeichenkette: ["[.]*"]
    - Boolesche Werte: [true|false]

```
Content-Type: application/json
```

Beispiel: RESTful-Service-Infrastruktur (Service Mashup)



Beispiel: RESTful-Service-Infrastruktur

- Authentication
- Retrieval
- JSON-Serialisierung

```
http_get_all_lists_opt = {
        host: sp_host,
        port: sp_port,
        path: '/_vti_bin/ListData.svc/'+spAllLists,
        headers: {
                'Accept': 'application/json; charset=utf-8',
                'Authorization': 'Basic ' + new Buffer(uname + ':' +
                        pword).toString('base64')}
};
```

Beispiel: RESTful-Service-Infrastruktur

Freie Universität Berlin

```
http.get(http_get_all_lists_opt, function(res){
        var allListsArr = new Array();
        var strJson = '';
        res.on('data', function(data) {
                strJson += data;
        });

        res.on('end', function() {
                var allLists = JSON.parse(strJson).d.results;
                for(var list in allLists){
                        …
                }

                next(err, allListsArr);
        });

        res.on('error', function(e) {
                …
        });
});
```

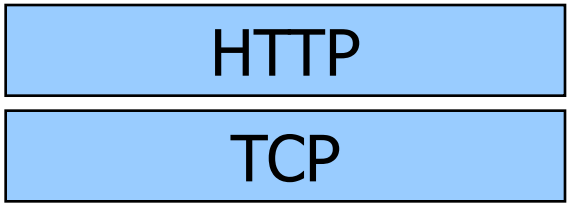
Beispiel: RESTful-Service-Infrastruktur

Freie Universität Berlin

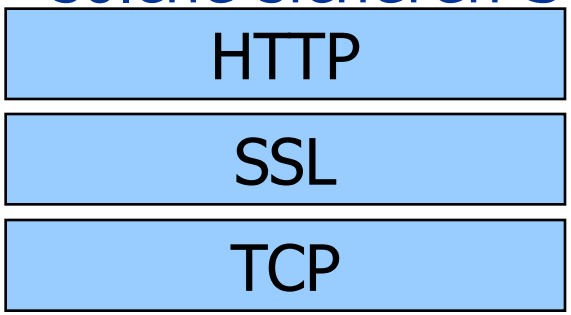
```
{"d" : { "results": [
        {
                "__metadata": {
                        "uri": "http://12.34.567.89/…/KalenderItemX",
                        "etag": "W/\"1\"",
                        "type": "Microsoft.SharePoint.DataService..."
                },
                "InhaltstypID": "..."
                "Titel": "Kopie: Termin an der FU",
                ...
        },
        ...
]}}
```

HTTP über SSL Sockets

- HTTP benutzt TCP Sockets zur Kommunikation



- Secure Sockets Layer SSL erweitert Sockets um Sicherheitsmerkmale
- HTTPS bezeichnet eine HTTP Kommunikation über solche sicheren Sockets

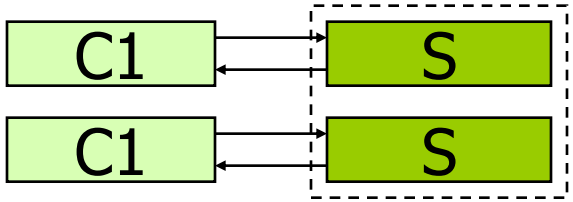


- Port 443 als Default-Port festgelegt

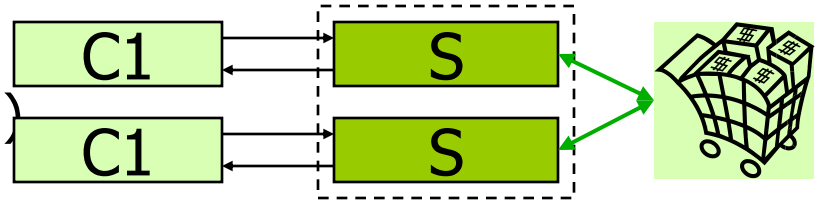
abgeleitet von: R. Tolksdorf

Zustand in HTTP

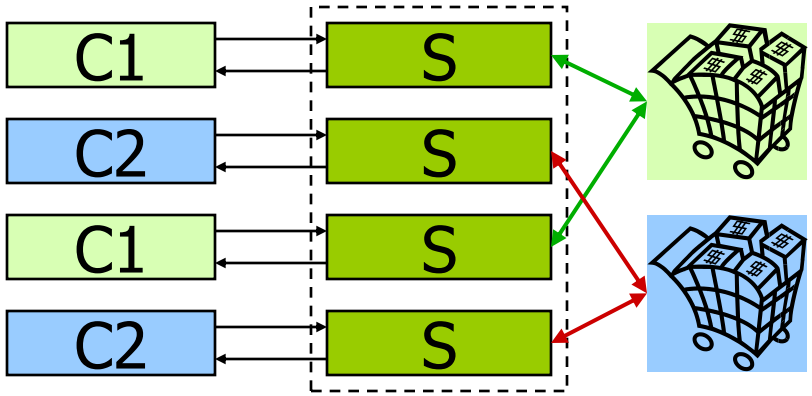
- HTTP ist zustandslos
  - Zwei Interaktionen sind unabhängig voneinander



- Zustand aber oft benötigt
  - Transaktionen auf Datensatz beim Server (z.B. Warenkorb)



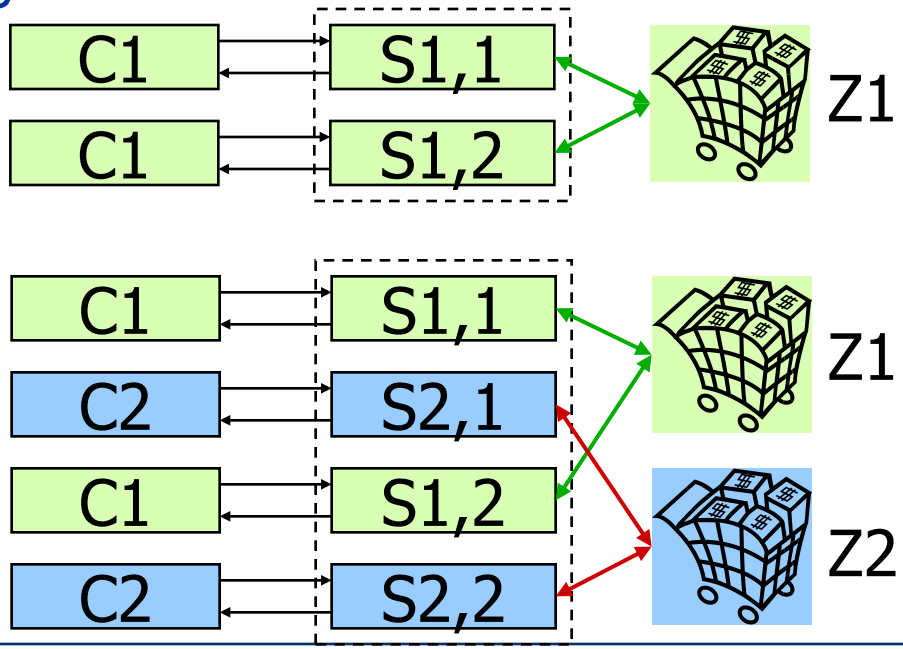
  - Unterscheidung von Klienten zur
    - Personalisierung
    - Authentifizierung
    - ...



abgeleitet von: R. Tolksdorf

Sessions

- Einführung von Sitzungen (Sessions)
- Sitzung: Folge von Interaktionen, die einen gemeinsamen Zustand haben
- Identifikation in der Interaktion durch eindeutige Sitzungsnummer (Session-ID)
- Ermittlung des Zustand auf Basis der Session-ID



abgeleitet von: R. Tolksdorf

Zustand in HTTP

- Client aus HTTP- und Socket-Informationen eindeutig identifizieren?
- Session-ID=
  (Browsername x User x Betriebssystem x IP-Adresse)
- *Nicht* eindeutig, weil:
  - Informationen bis auf IP-Adresse nicht immer vorhanden
  - IP-Adresse nicht eindeutig
    - Mehrere Nutzer auf einem Rechner
    - Proxy/Firewall/NAT Problematik: Keine individuellen IP-Adressen nach aussen
    - Mehrere Browser-Sessions des gleichen Nutzers
- => Session-ID muss in der Interaktion immer zwischen Client und Server ausgetauscht werden

abgeleitet von: R. Tolksdorf

1. Versteckte Formularfelder enthalten Session-ID

- Formularfelder in HTML:

  <input type=typ name="name" …> z.B.:

  <input type="text" name="PLZ"> Texteingabe

  <input type="password" name="pw">Passwordeingabe

  Weitere Auswahlen und Textfelder

  <input type="hidden" name="SessionID"
  value="977e5d8ae8500c456ab1fca6cbaa12af">

- Bei Submit wird ein Query-String

  PLZ=14195&pw=geheim&SessionID=977e5d8ae8500c456ab1fca6cbaa12af

  erzeugt und an den Server übermittelt

- Server wertet Session-ID aus und baut sie in Formulare auf Ergebnisseite ein

abgeleitet von: R. Tolksdorf

2. Session-ID in URLs in Verweisen (URL Rewriting)

- Session-ID kann unterschiedlich kodiert werden
- Abbildung von Pfad auf Informationen ist Server-Sache
  - Im Pfad:
    http://www.amazon.de/exec/obidos/tg/browse/-/301128/ref=cs_nav_tab_1/028-1096689-7395702
  - Im Query-String:
    http://www.cyberport.de/webshop/cyberportShop.omeco?ORDER=&PHPSESSID=8823f90c85597aedc87100cd91a4c7fd&FINANZING=

\+ Zustand kann außerhalb von Formulareingaben gehalten werden

\+ portabel

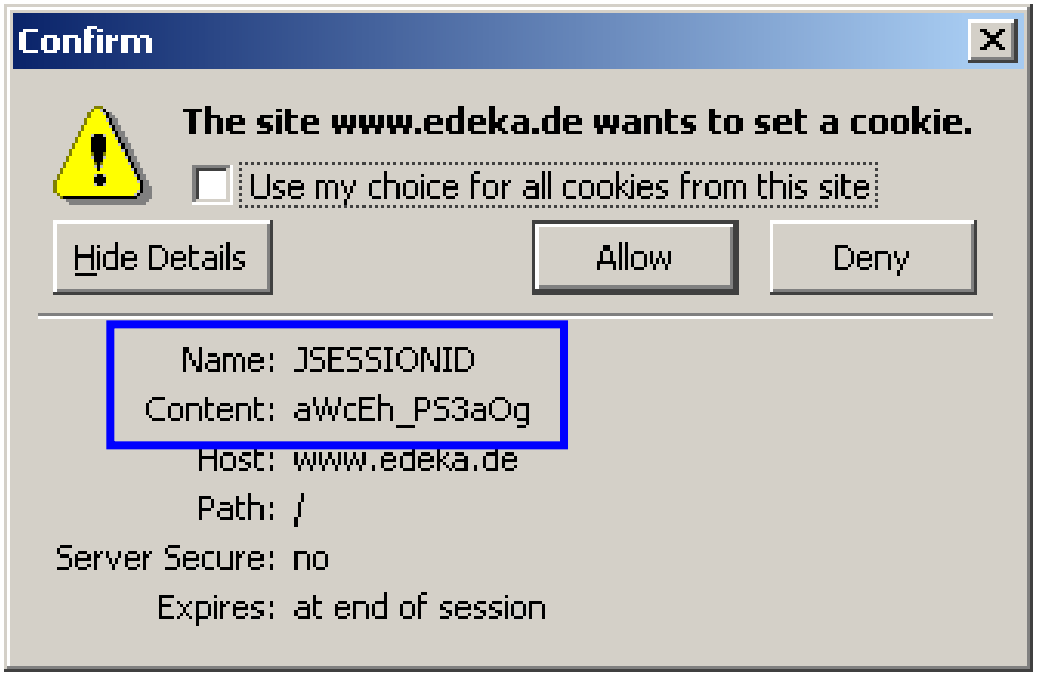
– alle Verweise müssen entsprechend markiert werden

– alte Session-ID kann in Bookmark sein

– gültige Session-ID kann einfach an andere Nutzer gelangen

abgeleitet von: R. Tolksdorf

3. Cookie Mechanismus zum Speicher der Session-ID



- Cookie ist kleiner Datensatz, der bei Client gespeichert ist
- Server kann ihn setzen
- Client schickt ihn bei jeder weiteren Interaktion mit
- Implementiert mit zusätzlichen HTTP-Headern

abgeleitet von: R. Tolksdorf

Zusammenfassung Web APIs

- REST-Prinzipien als Grundlage für Web APIs
- HTTP-Anfragen und -Antworten
- Resourcen, Repräsentationen und Content Negotiation