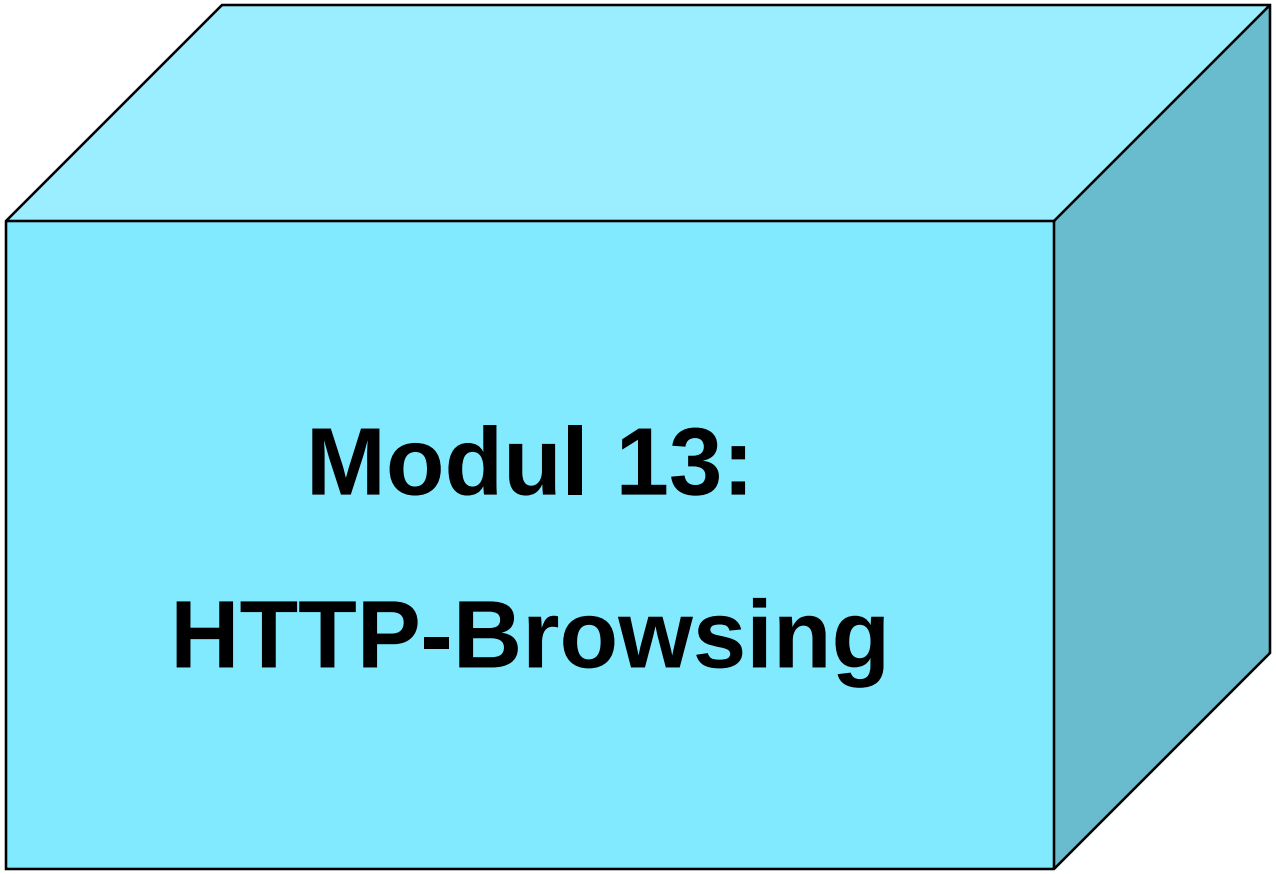
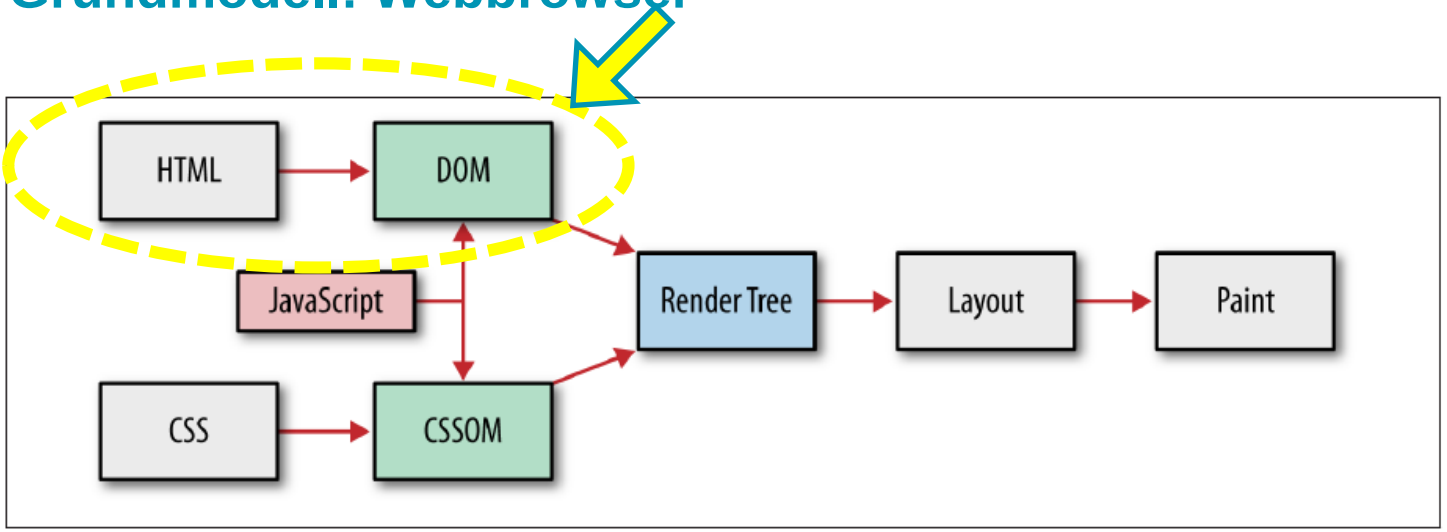
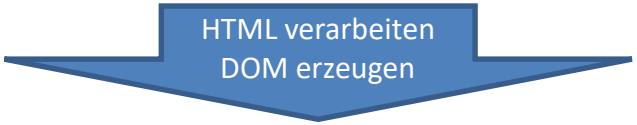


Grundmodell: Webbrowser



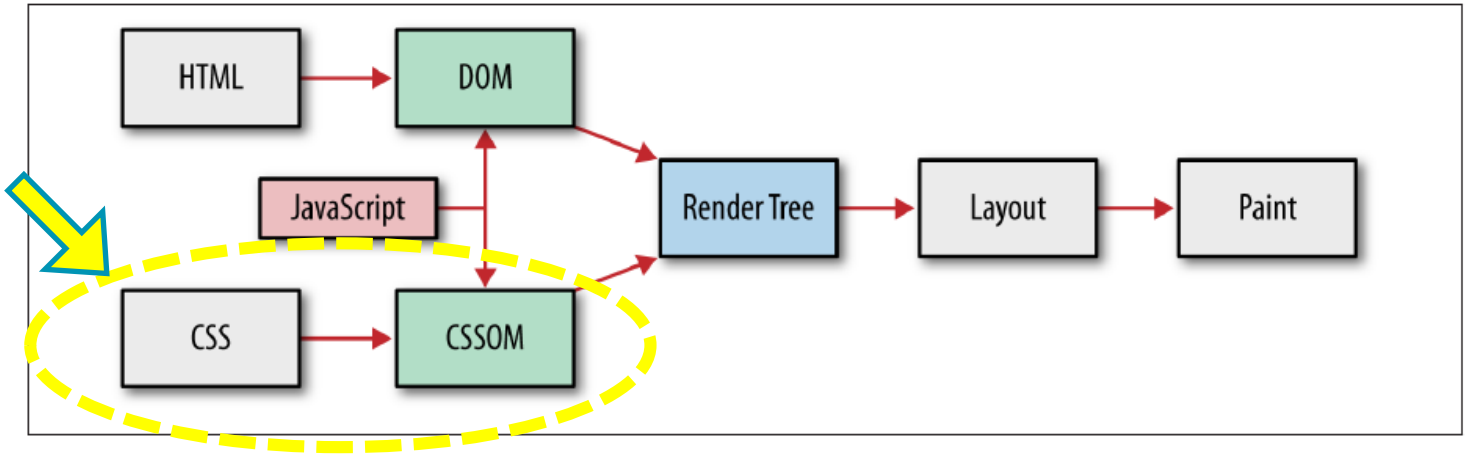
Komplexes Zusammenspiel verschiedenartiger Elemente:

- **HTML (Hypertext Markup Language): Struktureller Aufbau der Internetseite und Angabe von weiteren Objekten.**



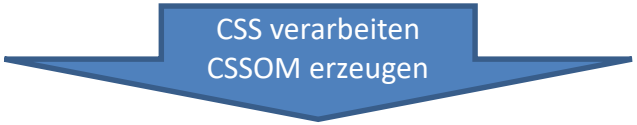
- **DOM (Document Object Model): Darstellung der HTML-Seite als Baum sowie Definition einer API für Operationen auf diesem Baum.**

Grundmodell: Webbrowser



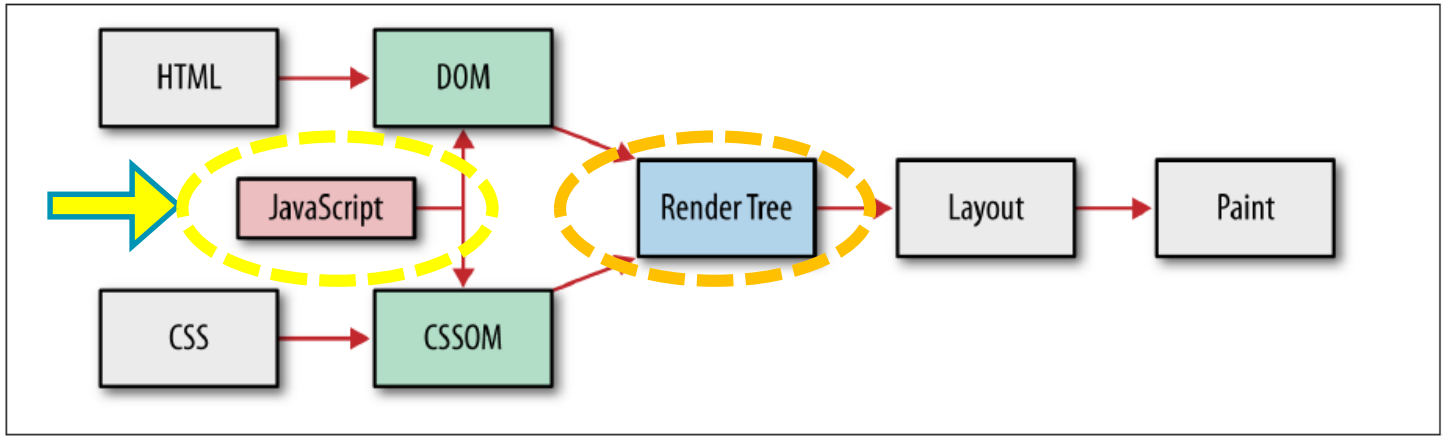
Komplexes Zusammenspiel verschiedenartiger Elemente:

- **CSS (Cascading Style Sheets): Formatierungssprache, um das Layout einer HTML-Seite festzulegen.**



- **CSSOM (CSS Object Model): Objektmodell, um die CSS-Formatierungen auf Knoten und Objekte im DOM-Baum zu verteilen.**

Grundmodell: Webbrowser

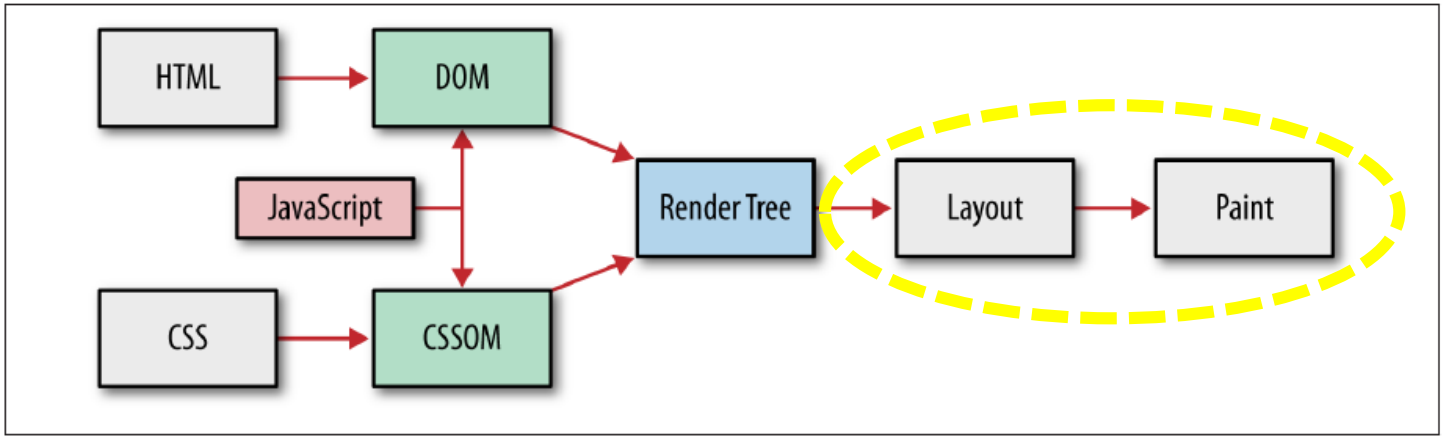


Komplexes Zusammenspiel verschiedenartiger Elemente:

- **JavaScript: Skriptsprache für dynamisches HTML in Webbrowsern. JavaScript verwendet die DOM-API.**
  Mit JavaScript kann DOM-Baum und CSSOM umgebaut bzw. modifiziert.

- **Render Tree: Ergebnis des gesamten Parsingprozesses, also der fertige DOM-Baum angereichert mit den Formatinformationen für die sichtbaren Elemente.**

Grundmodell: Webbrowser



Komplexes Zusammenspiel verschiedenartiger Elemente:

- **Layout: Größe und Positionierung der Element im Anzeigebereich festlegen. Element „fließen“ hierbei an bestimmte Stellen (Reflow).**
- **Paint: Umsetzung in reale Pixel.**

Grundstruktur HTML Homepage Leischner

```
<!DOCTYPE html>
<html lang="de" xmlns="http://www.w3.org/1999/xhtml">

<head>
<meta charset="utf-8" />
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-alpha.6/css/bootstrap.min.css" rel="stylesheet" />
<link href="styles.css" rel="stylesheet" />
<link href="print.css" media="print" rel="stylesheet" title="CSS" type="text/css" />
</head>

<body>

<div class="container-fluid NavContainer hidden-print">
        <nav class="navbar fixed-top navbar-inverse rounded navbar-toggleable-md hidden-print">
        .
        .
        </nav>
</div>
<div class="container-fluid ContentContainer">
        <div class="row hidden-print" style="height: 0.5em">
                <div class="col-md-2 dunkelblau2 hidden-md-down">
                </div>
                .
                .
                        <div class="card">
                        <h5 class="card-header">Prof. Dr. Martin Leischner </h5>
                        <div class="card-block">
                                                .
                                                .
</div>
<script src="https://code.jquery.com/jquery-3.1.1.slim.min.js"></script>

<script>
  (function(i,s,o,g,r,a,m){i['GoogleAnalyticsObject']=r;i[r]=i[r]||function(){
.
</script>

</body>
</html>
```

Definition des XML namespaces

Stylesheet bootstrap 4 aus CDN

Tag-Klassen

wird nicht gedruckt

responsive

selbstdefiniertes Format

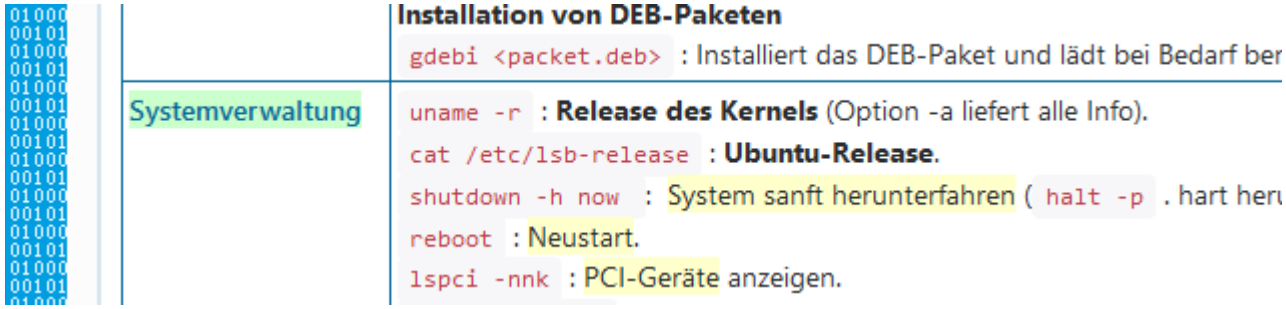
Skript für bootstrap

Skript für Google Analytics

wurde wegen DSGVO entfernt

Grundstruktur CSS Homepage Leischner

```
/* Ubuntu-Modifizierungen der blauen Tabelle*/
.befehlstabelle {
        font-size: 0.85em;
        margin: 10px;
        float: left;
}

.befehlstabelle th {
        font-size: 1.2em;
        font-weight: 600;
}

.befehlstabelle td {
        padding: 3px;
        margin: 1px;
        vertical-align: top;
        background-color: #FFFFFF;
}

.befehlstabelle tr td:nth-child(1) strong {
        background-color: hsl(120, 100%, 90%);
        color: #06689A;
        font-weight: 600;
}

.befehlstabelle tr td:nth-child(2) strong {
        font-weight: 700;
}

.befehlstabelle  .gelb {
        background-color: #FFFFCC;
```



XML Pfadausdrücke gemäß XPath

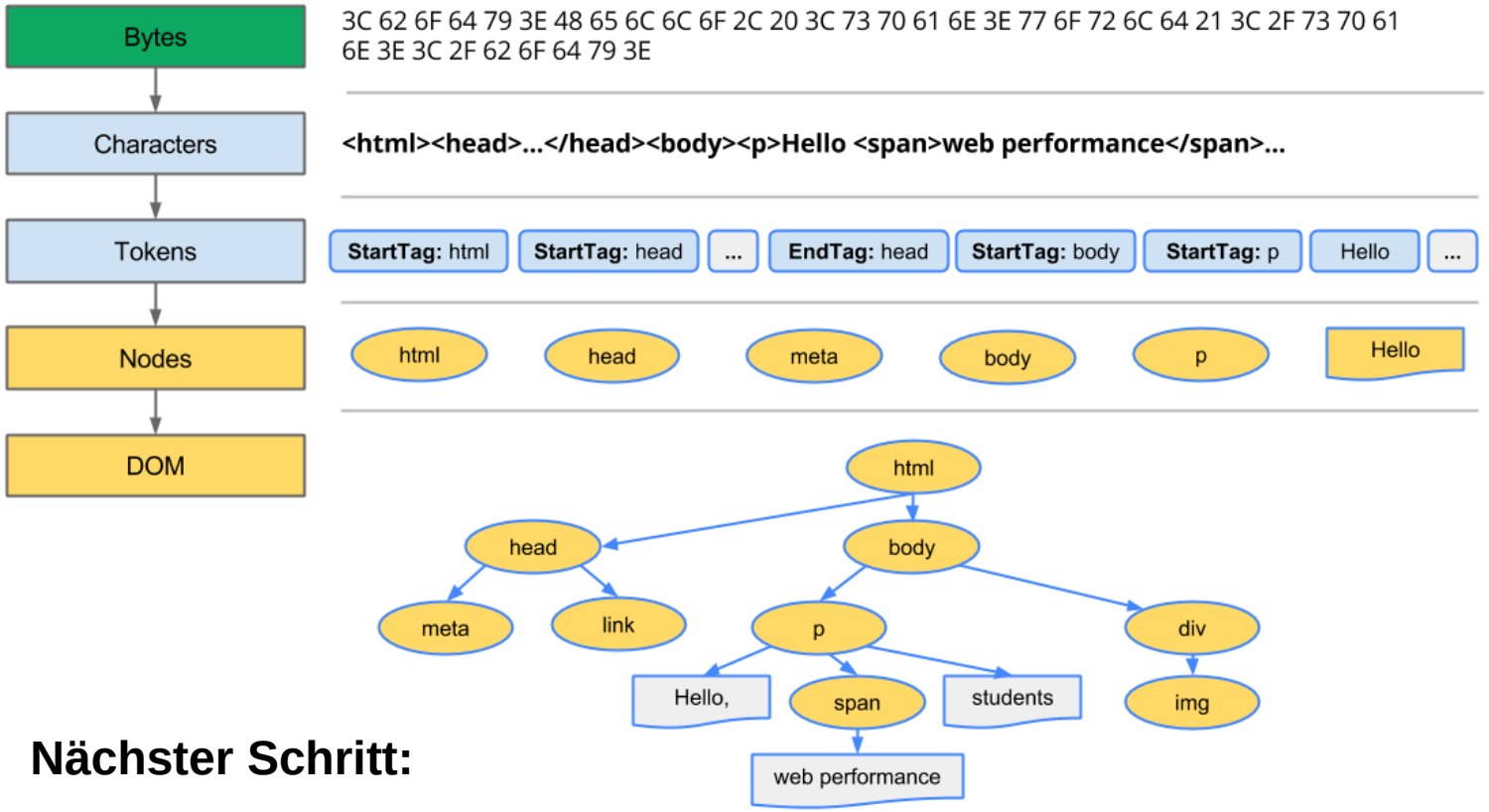
*Hier wird im CSSOM-Baum zunächst das Tag strong in der **ersten** Spalte der befehlstabelle selektiert.*

Danach wird u.a. als Hintergrund grün hinterlegt

*Hier wird im CSSOM-Baum zunächst das Tag strong in der **zweiten** Spalte der befehlstabelle selektiert.*

Als Format wird die Schrift angefettet.

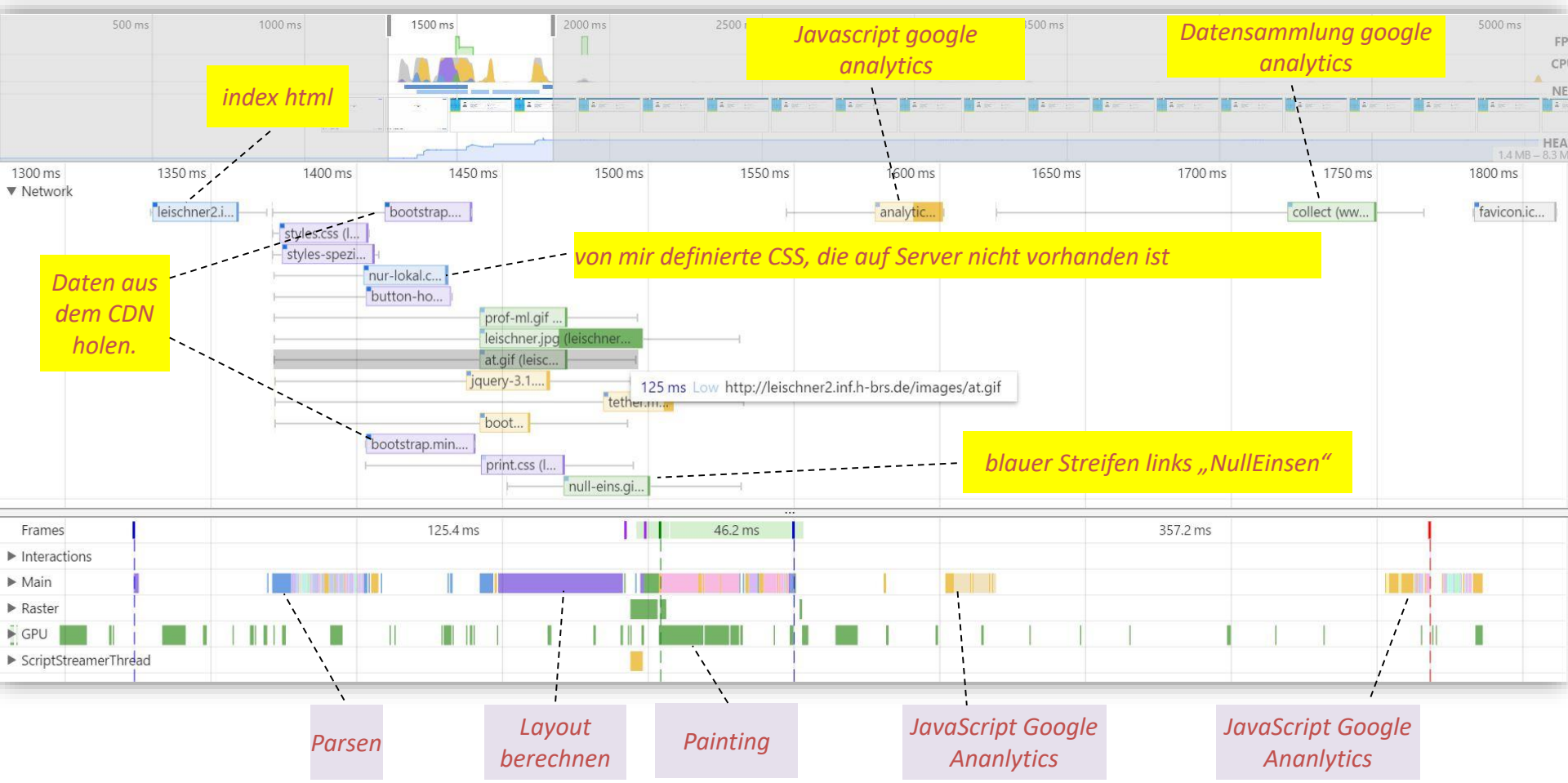
Parsing-Prozess bis Aufbau DOM



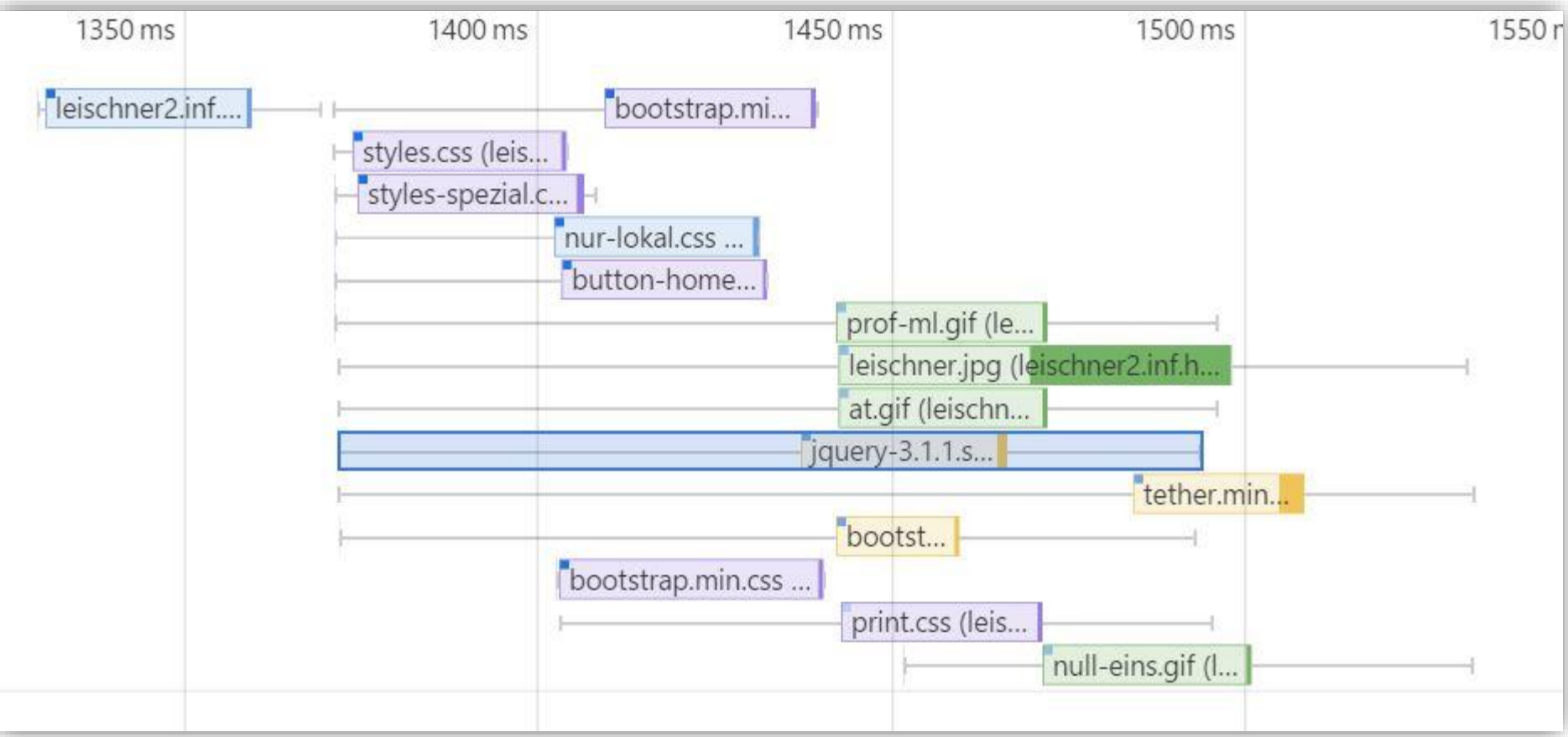
Nächster Schritt:

- **CSSOM (CSS Object Model) aufbauen**

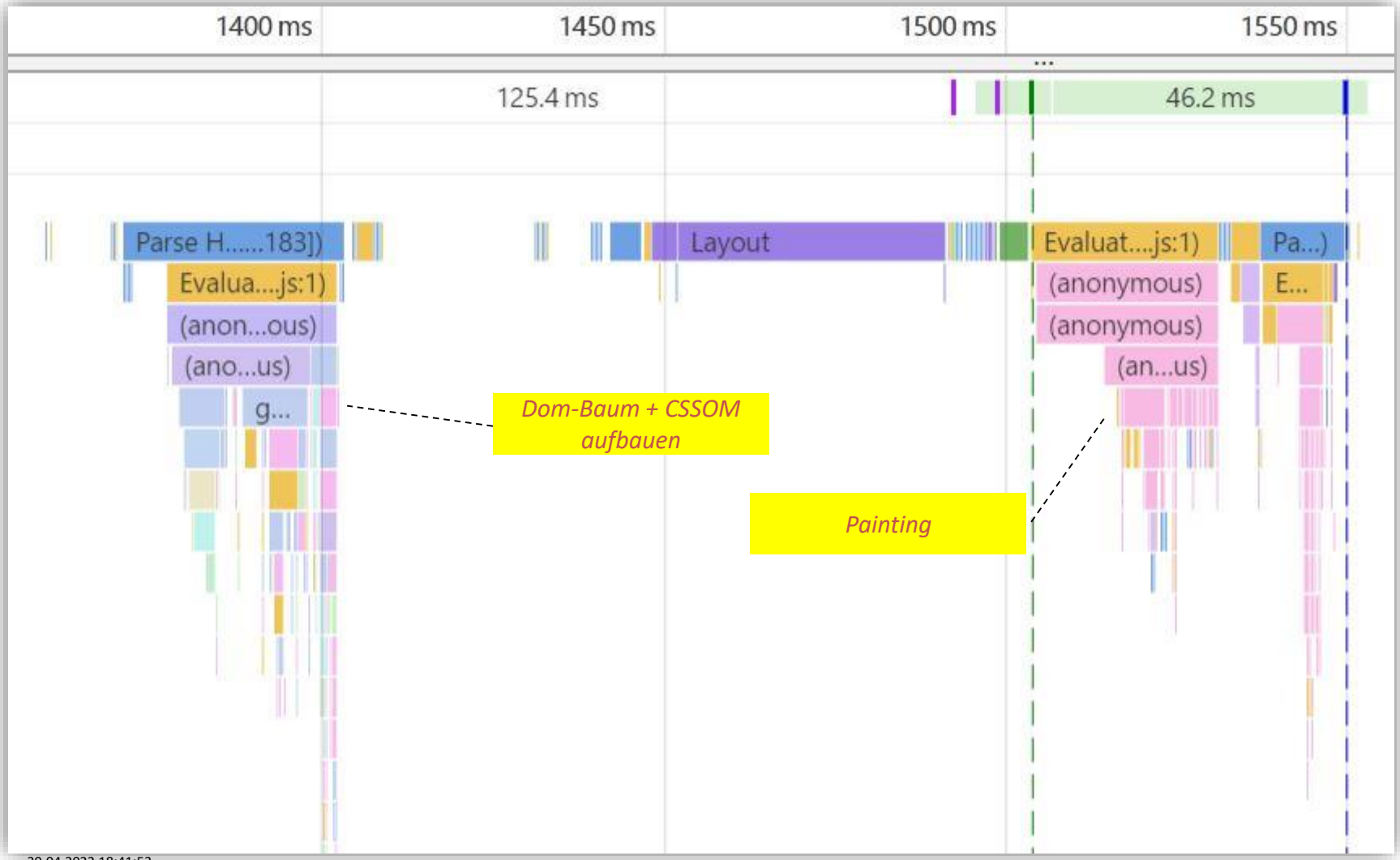
Ablauf des Seitenaufrufs Homepage Leischner mit http 1.1



Seitenaufruf leischner2.inf.h-brs.de mit http 1.1 – im Detail



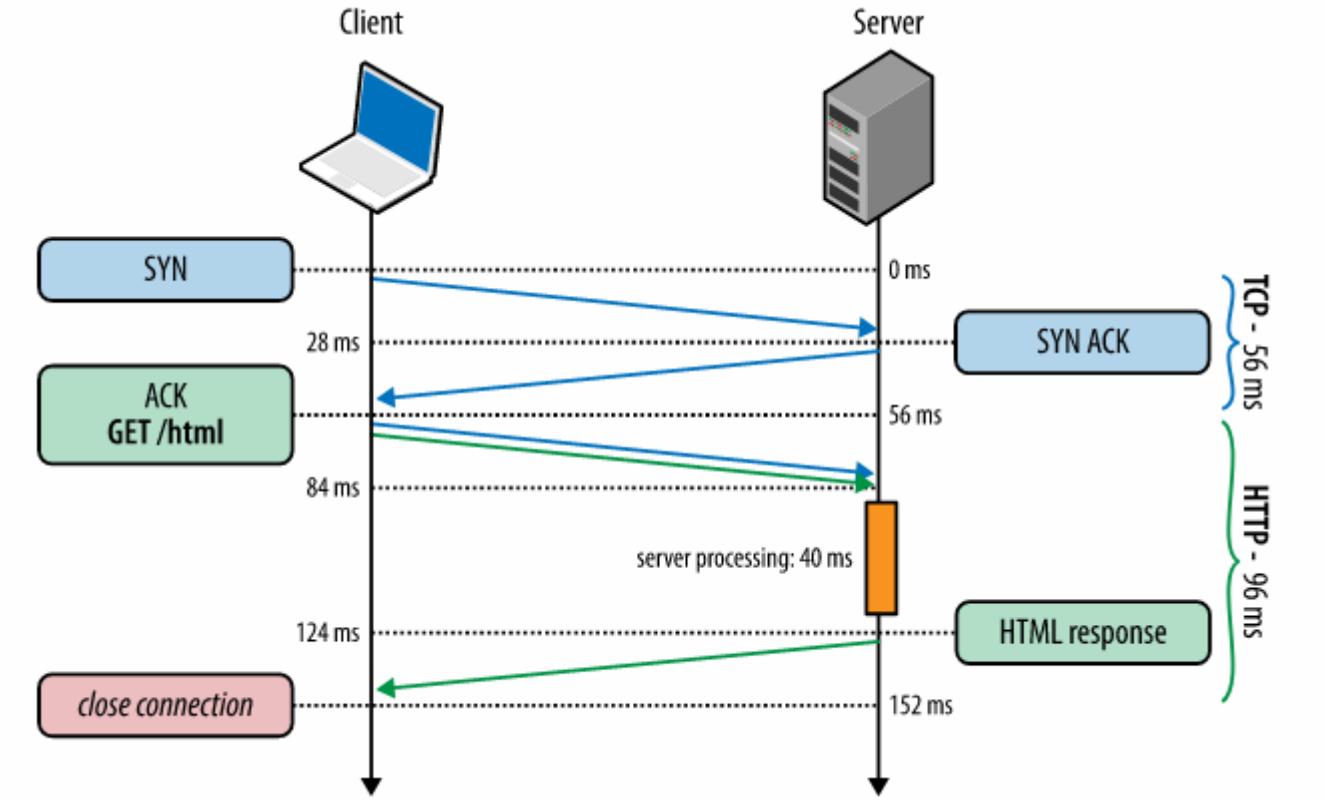
Rekursives Parsing entlang der Baumstrukturen (DOM, …)



HTTP Performance –"Schneller ist besser"

- **Geschwindigkeit ("Page Speed") ist eines der wichtigsten Merkmale einer Website.**
- **Die Absprungrate einer Landing Page hängt stark von ihrer Ladezeit ab. Hierzu hat Google Studien durchgeführt.**
  (Beispiel: Anstieg der Ladezeit von 1 auf 3 Sekunden erhöht Absprungrate um 32%)
- **Insgesamt sinkt die Conversion Rate einer Website bei geringer Seitengeschwindigkeit deutlich.**
  (Conversion Rate = Conversions (Zielerreichung) / Impressions)
- **Maßnahme zur Erhöhung der Page Speed:**
  - **Bilder komprimieren, Auflösung herabsetzen**
  - **Caching via CDN (z.B. Cloudflare)**
  - **Effizientes Zugriffsprotokoll (HTTP/1.1 → HTTP/2)**

HTTP/1.0



- supereinfach
- TCP-Slowstart-Problematik
- häufiger TCP-Verbindungs-aufbau
- insgesamt ineffektive Nutzung von TCP

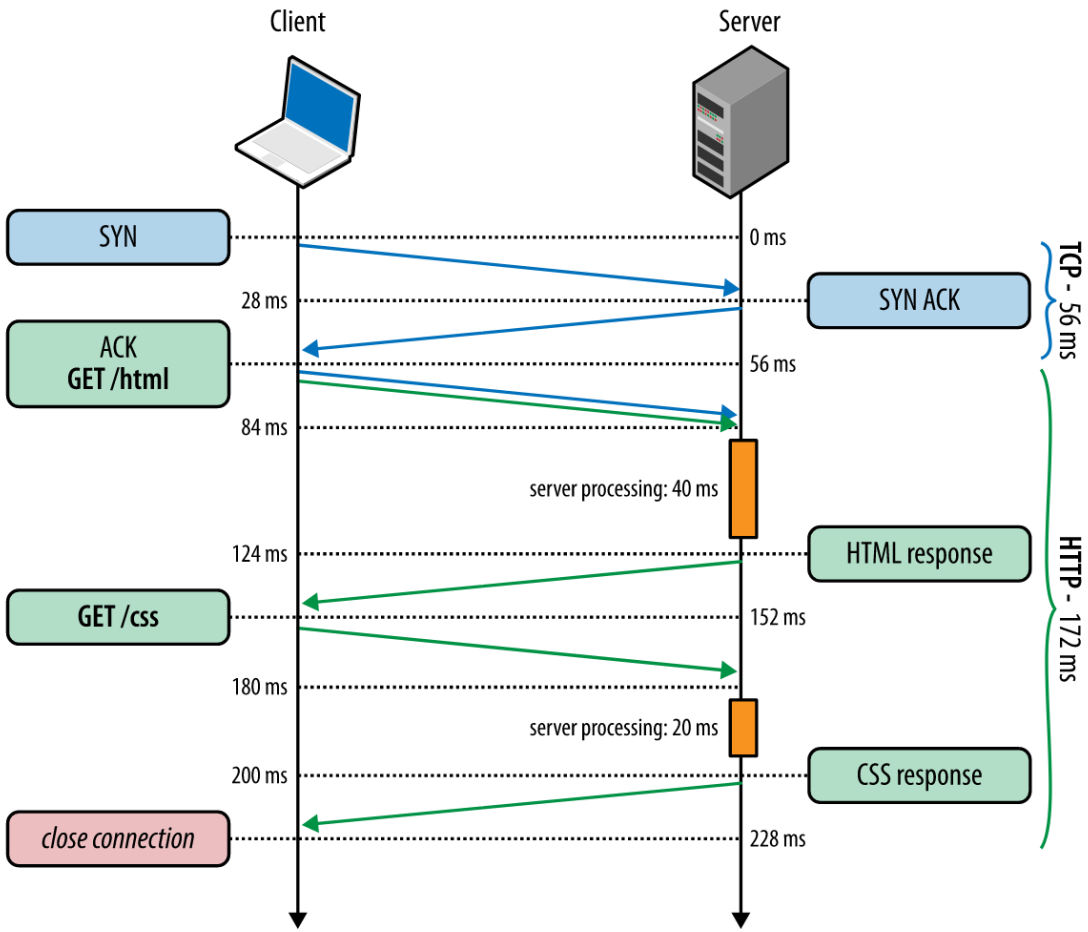
RFC 1945, Hypertext Transfer Protocol -- HTTP/1.0, May 1996

HTTP/1.1

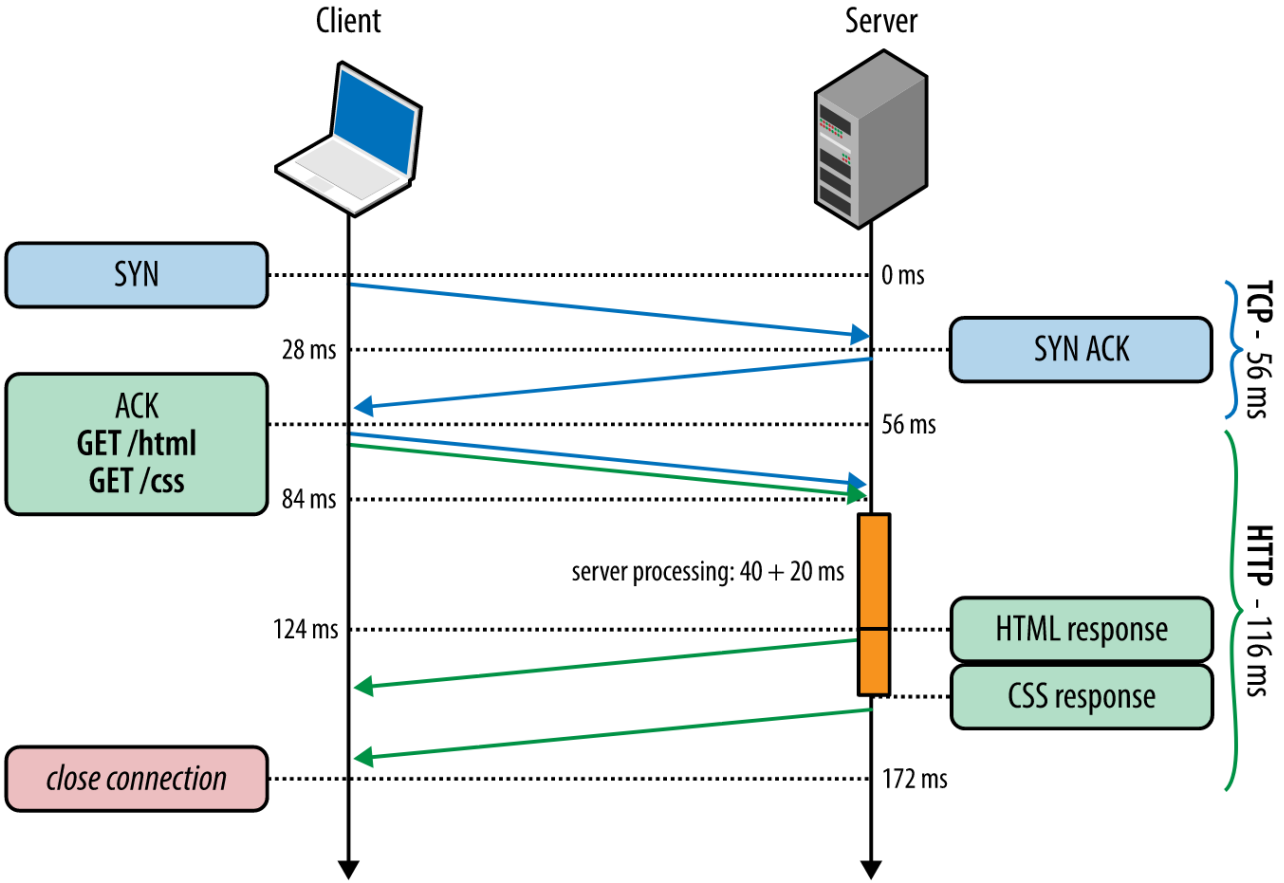
- RFC 2068, Hypertext Transfer Protocol -- HTTP/1.1, Jan 97
- Aktuell:
  - RFC 7230, Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, June 14
  - RFC 7231, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, June 14

- Hauptidee: persistent HTTP - TCP-Verbindung nach Request/Response offen lassen.
- Zusatzidee: HTTP Pipelining - auf einer offenen TCP-Verbindung mehrere Requests parallel anstoßen.
  Problem hier: " not mandatory", nicht durchgehend implementiert.
- Fragestellung: Welche Parallelität ist besser?
  - Parallele TCP-Verbindungen vs. HTTP-Pipelining

Persistent HTTP 1.1



HTTP/1.1 Pipelining



HTTP/2

- RFC 7540, Hypertext Transfer Protocol Version 2 (HTTP/2), May 15
- Aktuell: RFC 8740, Using TLS 1.3 with HTTP/2, Feb 20

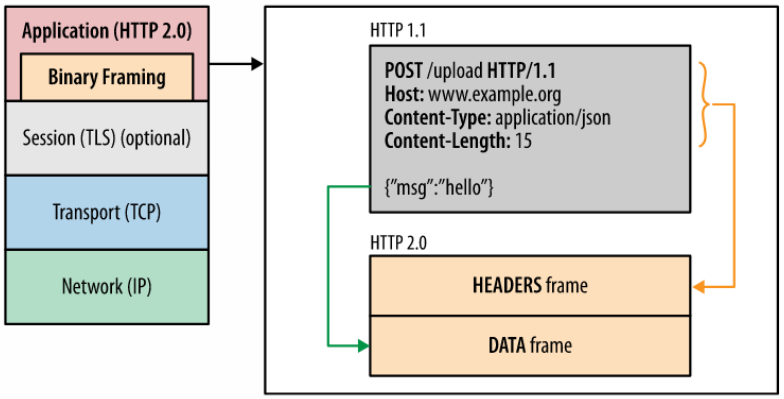
Hauptziele von HTTP/2:

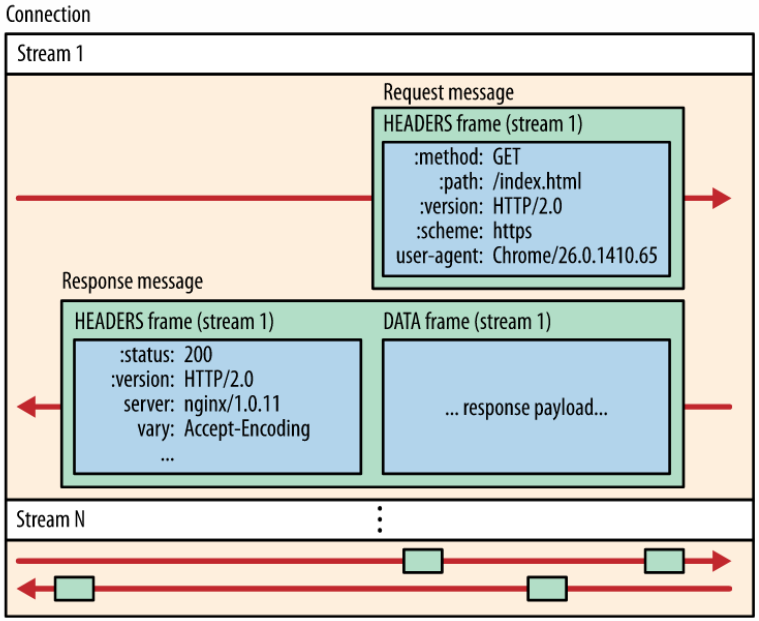
- Latenzzeiten (Verzögerungen) minimieren.
- Parallelität durch volles Multiplexen von Request/Response.
- Protokolloverhead durch effiziente Kompression der Header-Felder minimieren.
- Einführen von Daten-Streams, die priorisiert werden können. Wichtige Ressourcen für eine Webseite (z.B. Stylesheets) können vorrangig ausgeliefert werden.
- Applikationskompatibilität mit alten Versionen von HTTP.
  Alle Kernkonzepte von HTTP wie Methoden, URI, Status-Codes bleiben an der Programmierschnittstelle vollständig erhalten. Es sind keine Änderungen in den Applikationen notwendig. Die ganze neue Komplexität ist im HTTP/2-Protokoll gekapselt.

HTTP/2 - Binary Framing Layer

Prinzip:

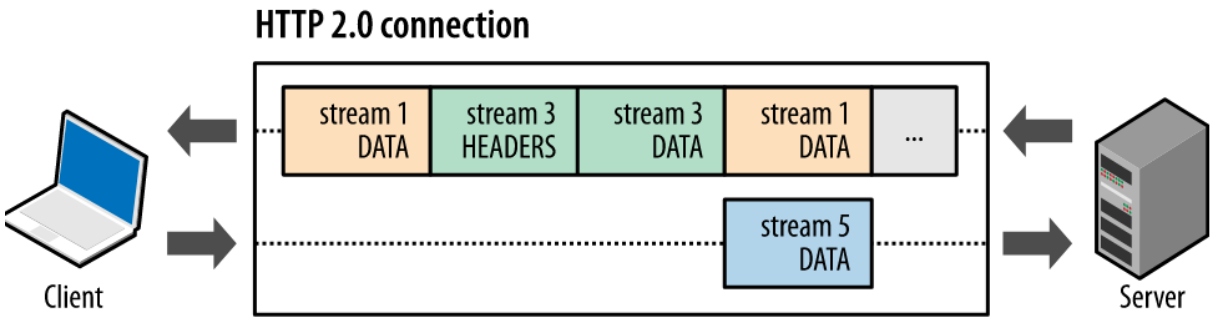
- Alle Daten werden in Frames verpackt.
- Eine Message kann sich aus mehreren Frames zusammensetzen
- Messages werden einem Datenstrom zugeordnet. Ein Datenstrom wird durch einen Identifieren geknnzechnet





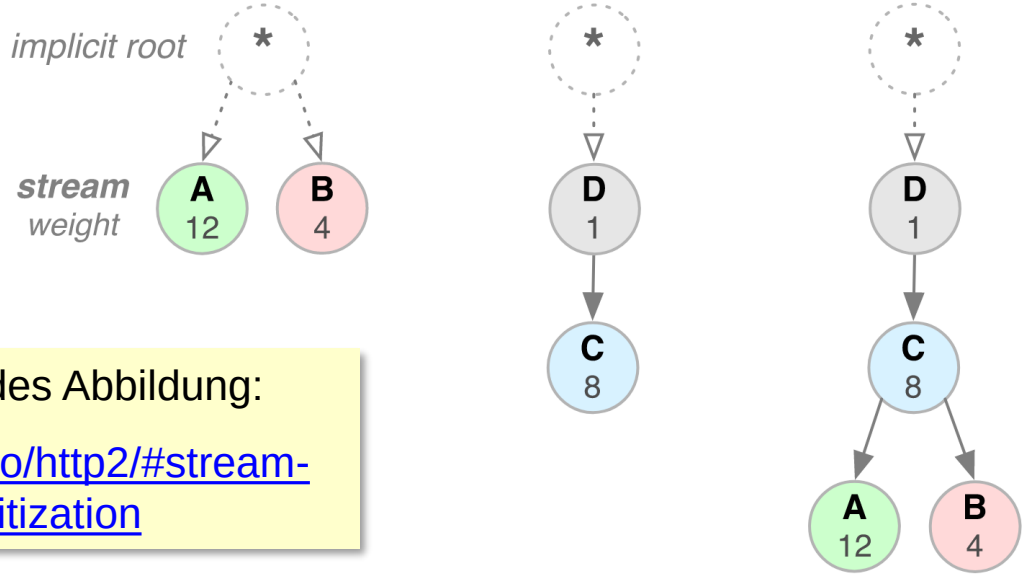
HTTP/2 - Request and response multiplexing

- Die gesamte Kommunikation läuft über eine einzige TCP-Verbindung
- Die Parallelität wird durch das Framing erreicht



HTTP/2 – Stream Priorisierung

- **Jedem HTTP/2-Strom kann ein Gewicht zugeordnet werden. Höher gewichtete Stream erhalten vom Browser mehr Resourcen.**
- **Streams können voneinander abhängig sein.**

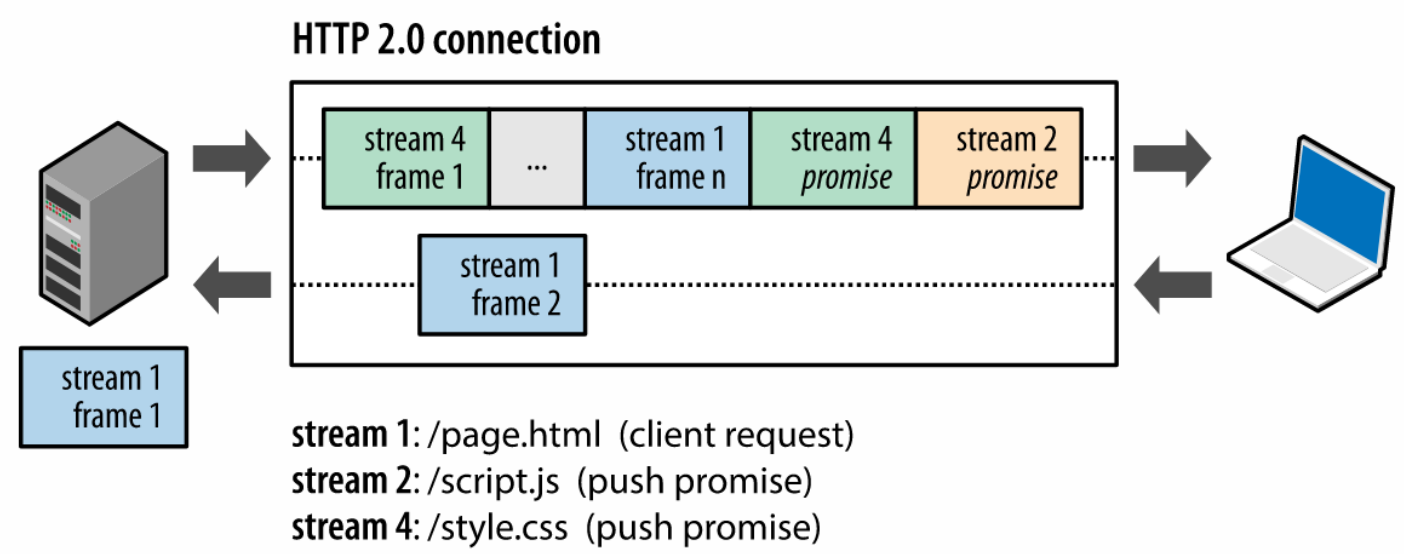


Erklärung des Abbildung:

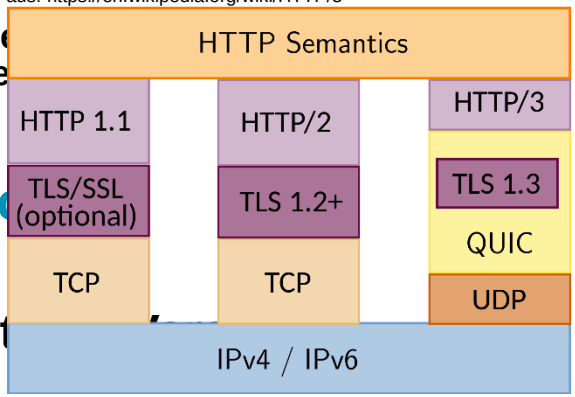
https://hpbn.co/http2/#stream-prioritization

HTTP/2 – Server Push

- **Der Server kann Daten an den Client pushen, ohne eine Request erhalten zu haben.**
- **Server-Push eröffnet neue Anwendungsfelder.**



aus: https://en.wikipedia.org/wiki/HTTP/3



HTTP/3 - Das neue Hypertext Transfer Protoc

- **draft-ietf-quic-http-34: Hypertext Transfer Prot , (HTTP/3), Feb 21**
- **Hauptidee: Transportprotokoll TCP durch UDP ersetzen.**
- **Hierzu hat Google das Protokoll QUIC (normiert in RFC 9000, May 2021) entwickelt, das auf UDP statt TCP aufsetzt.**
  Diese Entwicklung ist die Basis von HTTP/3.
- **Die Verwendung von UDP als Transportprotokoll hat eine Menge von Vorteilen, aber führt auch zu neuen Herausforderungen. (Diese könnten ausführlich diskutiert werden).**
  Beispiele:
  - TCP-Head-of-line-Blocking wird durch UDP vermieden.
  - Sicherheitsprobleme durch eingeschränkte TLS-Authentifikation.
- **Integrierte TLS-1.3-Verschlüsselung als obligatorischer Bestandteil.**

Literatur

- Ilya Grigorik: *High Performance Browser Networking*, O´Reilly, 2014.
- Ilya Grigorik: *High Performance Browser Networking*, https://hpbn.co, (letzter Zugriff 30.04.22).
- Ilya Grigorik: *Critical Rendering Path, https://developers.google.com/web/fundamentals/performance/critical-rendering-path*, (letzter Zugriff 30.04.22).
- DeavidSedice's blog: *HTTP Pipelining is useless*, https://deavid.wordpress.com/2019/10/06/http-pipelining-is-useless/ (letzter Zugriff 30.04.22).
- Ionos: HTTP/3: *Das nächste Hypertext Transfer Protocol einfach erklärt*, https://www.ionos.de/digitalguide/hosting/hosting-technik/http3-erklaert/ (letzter Zugriff 30.04.22).