Technical SEO Meetup

SEO FRIENDLY AJAX

München, 19. Juni 2017

AJAX: BASICS

Grundlagen & Funktionsweise von „Asynchronous JavaScript & XML“

AJAX: GRUNDLAGEN & FUNKTIONSWEISE

Klassisches HTTP-Request/Response-Modell verlagert sich vom Transfer ganzer Seiten hin auf eine granulare Ebene mit Modulen & Elementen

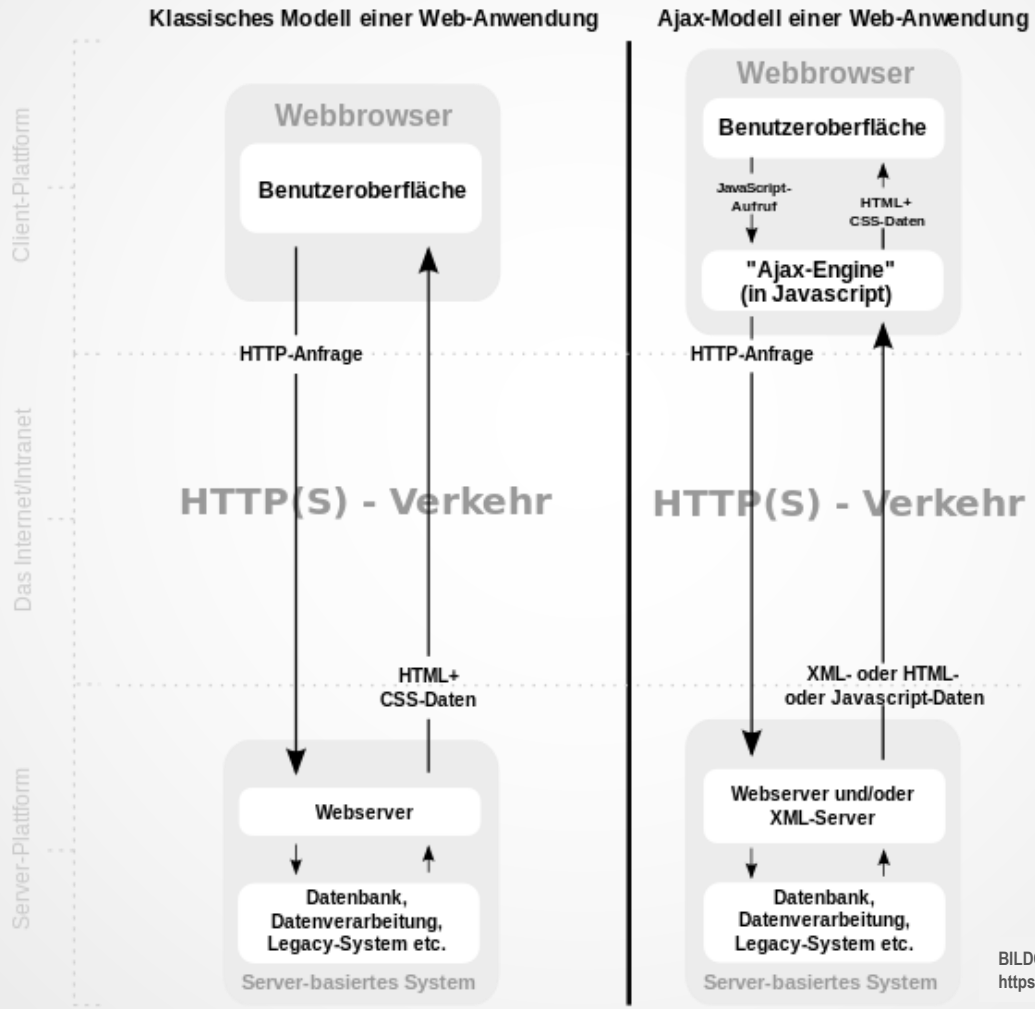
HTTP-TRANSPORT VOLLSTÄNDIGER RESSOURCEN (HTML)

A synchronous
→ Parallele Requests

J avascript
→ Clientseitig steuerbar

A nd

X ML
→ Datenformat für Transfer
→ JSON als schlankere Option
→ Auch HTML u.a. denkbar



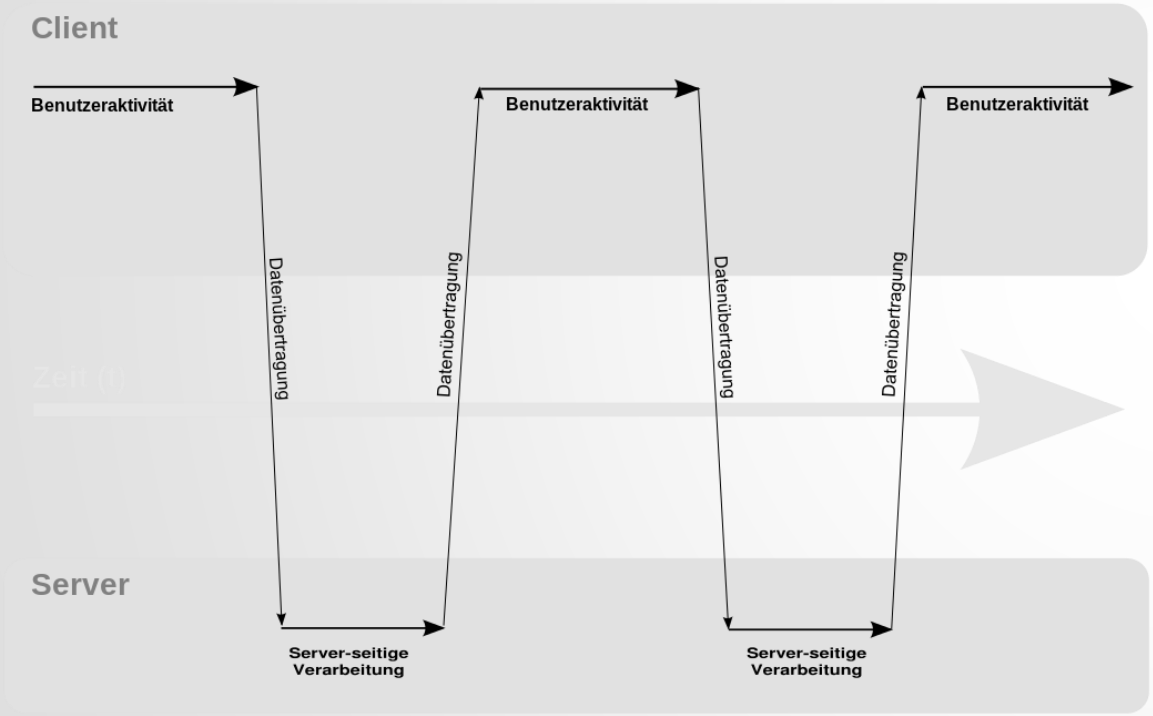
BILDQUELLE:
https://de.wikipedia.org/wiki/Ajax_(Programmierung)

HTTP-TRANSPORT EINZELNER DATEN, MODULE & ELEMENTE

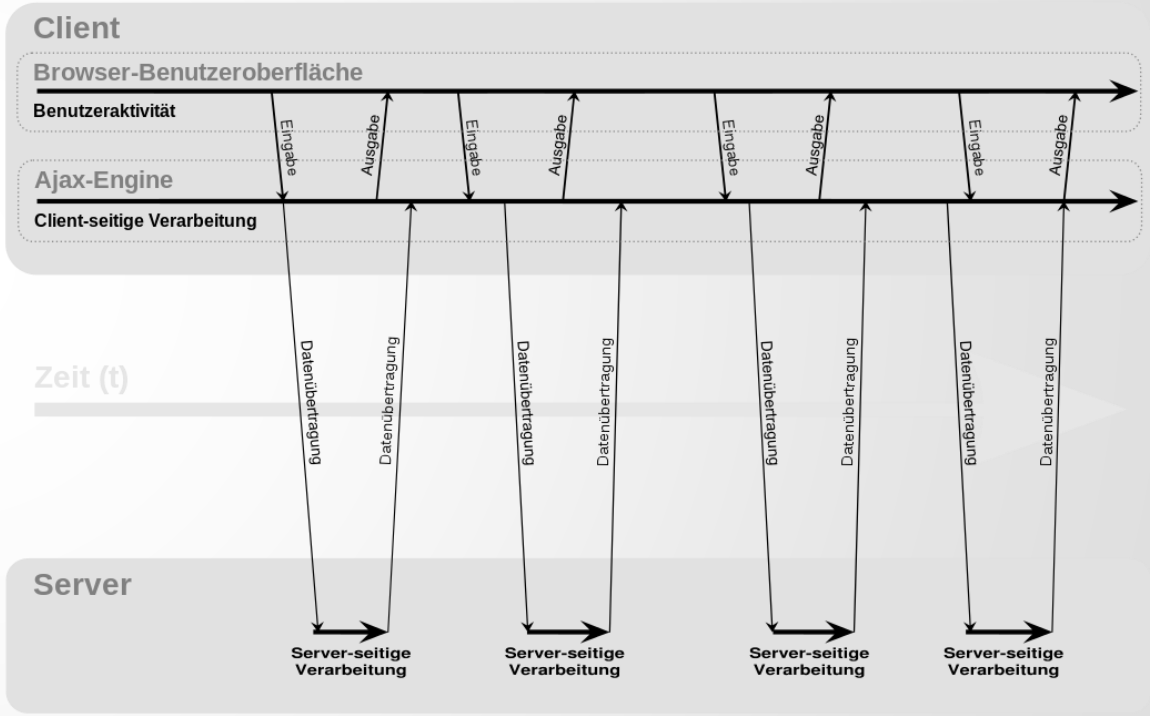
SYNCHRONE DATENÜBERTRAGUNG VERSUS ASYNCHRONE DATENÜBERTRAGUNG

AJAX ermöglicht verteilte, asynchrone und parallelisierte Datenübertragung als Basis für eine nahtlosere User Experience

Klassisches Modell einer Web-Anwendung (synchrone Datenübertragung)



Ajax Modell einer Web-Anwendung (asynchrone Datenübertragung)



BILDERQUELLE:
https://de.wikipedia.org/wiki/Ajax_(Programmierung)

GRUNDGERÜST EINER AJAX-ANWENDUNG IN JQUERY

Definition eines AJAX Request Handling bei Klick auf einen Button, inkl. Event Handler für Prozessszenarien

```
$("button").click(function(){ // Definiert Event Handling bei Klick auf ein Button Element

    $.ajax({
          url:  "/proxy/ajaxResponse",  // Request URL → default auf gleicher Domain wg. CORS
          method: "POST",               // GET vs. POST für offene vs. "versteckte" Übertragung
          dataType: "xml",              // Typ der Response-Daten - "xml" vs. "json" vs. "html"
          data: { dataArray },          // Request Parameter v.a. bei POST Request

          beforeSend: function(){
                              // wird vor dem Senden ausgeführt
                              // → Bsp. Anzeige von loading gif
                    }
          }),

          success: function(data){
                              // wird nach erfolgreichem Empfang einer Antwort ausgeführt
                              // → Weiterverarbeitung von data, Platzierung der Elemente im Page DOM
                    }
          }),
          error: function(data){
                              // wird im Fehlerfall ausgeführt
                              // → Fehlermeldung? Erneuter Request? Caching-Daten?
                    }
          }),
     })

});
```

RESPONSE-FORMATE VON AJAX-APPLIKATIONEN

XML vs. JSON (=„JavaScript-ready“) vs. Plain HTML: Vor- und Nachteile aus verschiedenen Perspektiven

EVENT	XML	JSON	HTML
SERVER-SIDE COMPLEXITY			
CLIENT-SIDE COMPLEXITY			
PAGE SPEED			
CROSS-BROWSER COMPATIBILITY			
FLEXIBILITY			

RESPONSE-FORMATE VON AJAX-APPLIKATIONEN

Beispiele für Implementierungen von Server-Side Data / Content Proxies in PHP

XML

```
<?php

    header("Content-type:
            text/xml");

    $dom = new DOMDocument();

    $d1 = $dom->createElement("data");
    $c1 = $doc->createElement("cont", "xyz");

    $d1->appendChild($c1);
    $dom->appendChild($d1);

    echo $dom->saveXML();

?>
```

Bau einer XML-Struktur mittels „DOMDocument“

JSON

```
<?php

    header("Content-type:
            application/json");

    $dataArray = array (
                        /* … data structure … */
                );

    $output = json_encode($dataArray);

    echo $output;

?>
```

Einfache Konvertierung von PHP-Arrays in JSON-Struktur

HTML

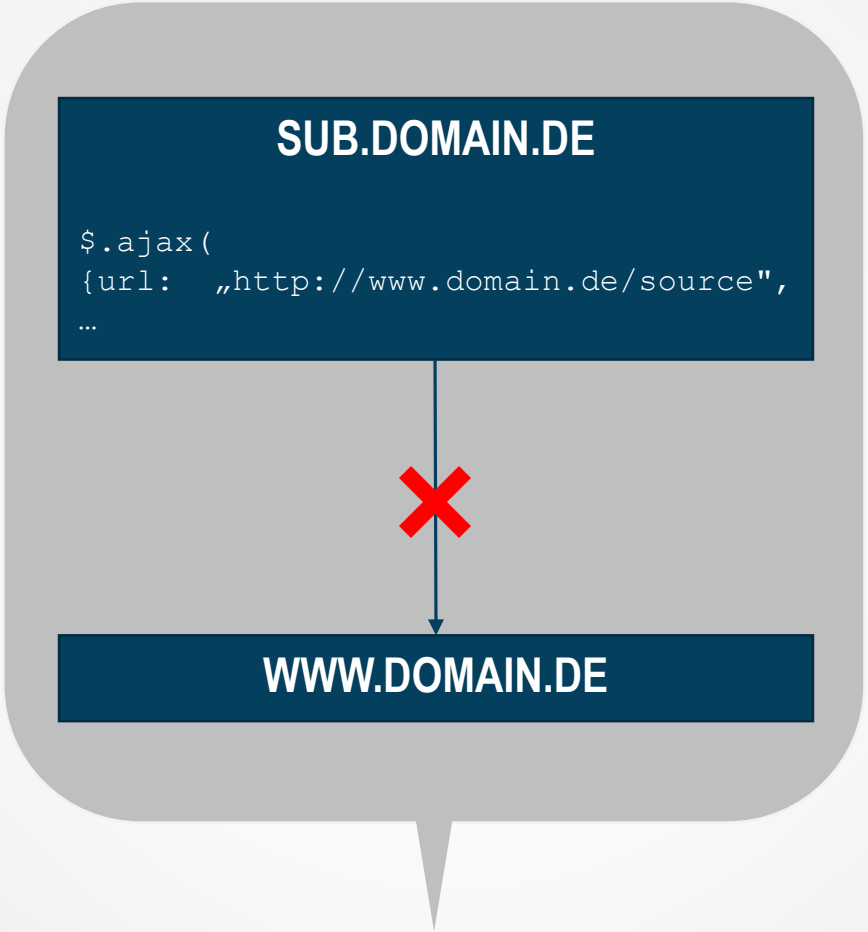
```
<?php

    header("Content-type:
            text/html");

    $output =  "<div>";
    $output .= "<span>xyz</span";
    $output .=  "</div>";

    echo $output;

?>
```

HTML Rendering erfolgt analog zu Initial Page

AJAX: TECHNISCHE HERAUSFORDERUNGEN

CROSS-ORIGIN RESSOURCE POLICY („CORS“)

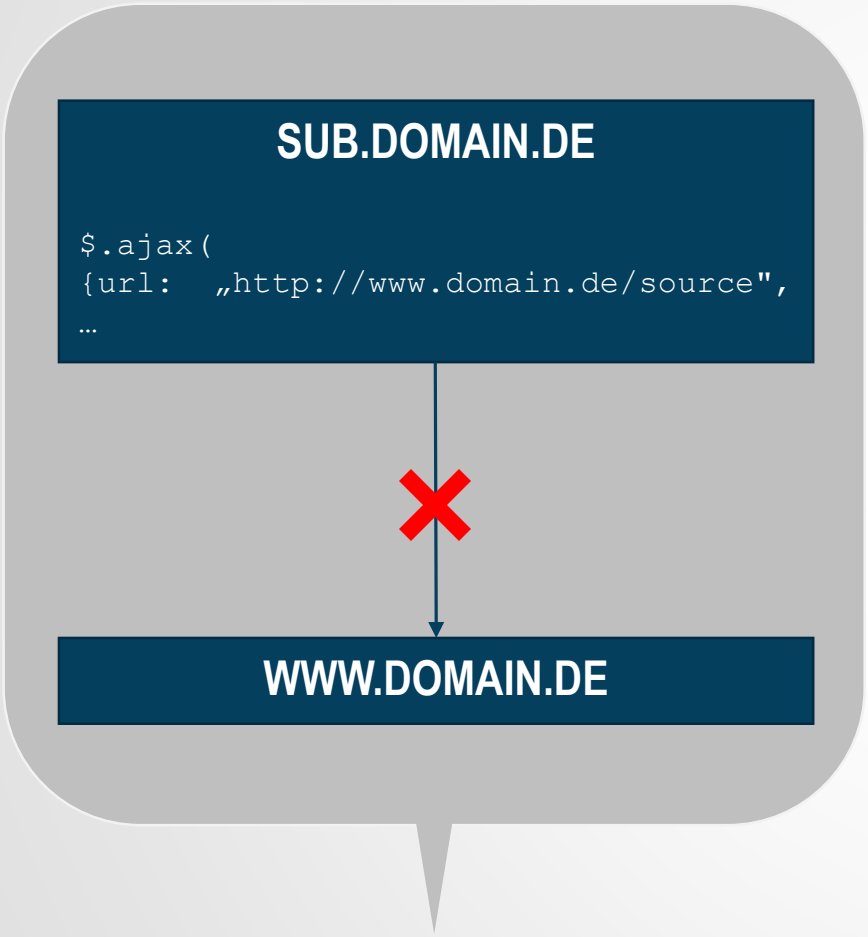
Datenaustausch zwischen verschiedenen Domains wird per Default aus Sicherheitsgründen blockiert



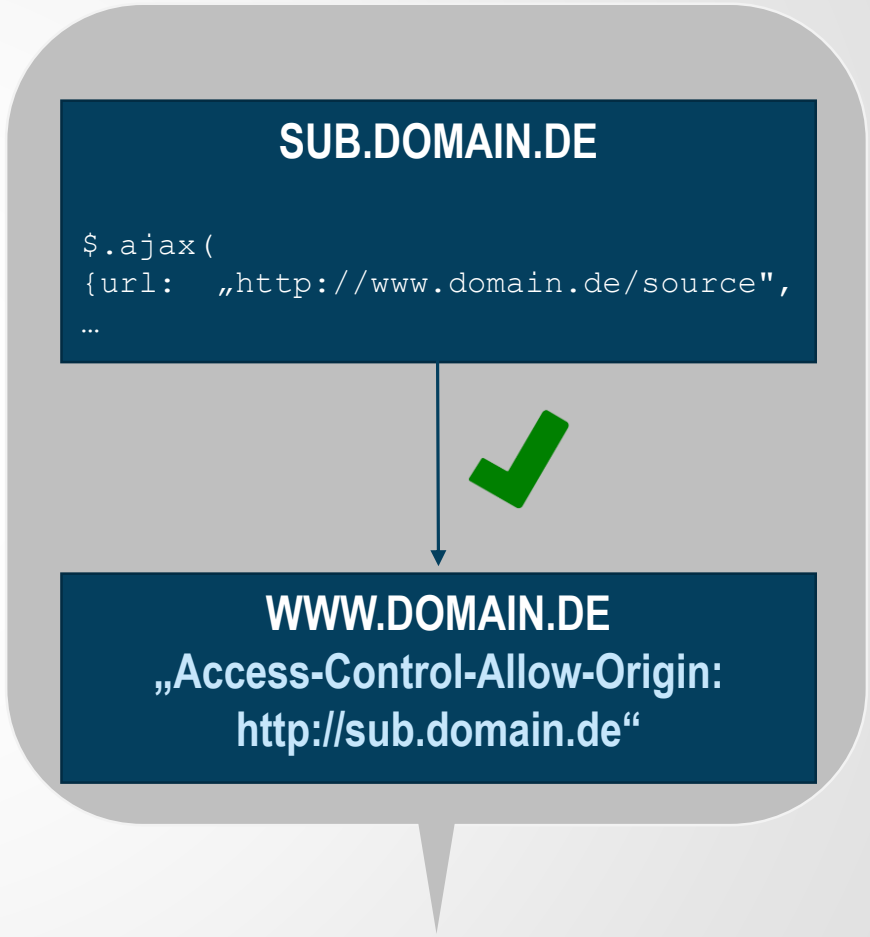
HERAUSFORDERUNG: BLOCKED REQUEST

CROSS-ORIGIN RESSOURCE POLICY („CORS“): LÖSUNG ÜBER ACCESS CONTROL VIA RESPONSE SERVER

Einfachster Lösungsweg: Anpassung der Sicherheitsmechanismen via HTTP Header



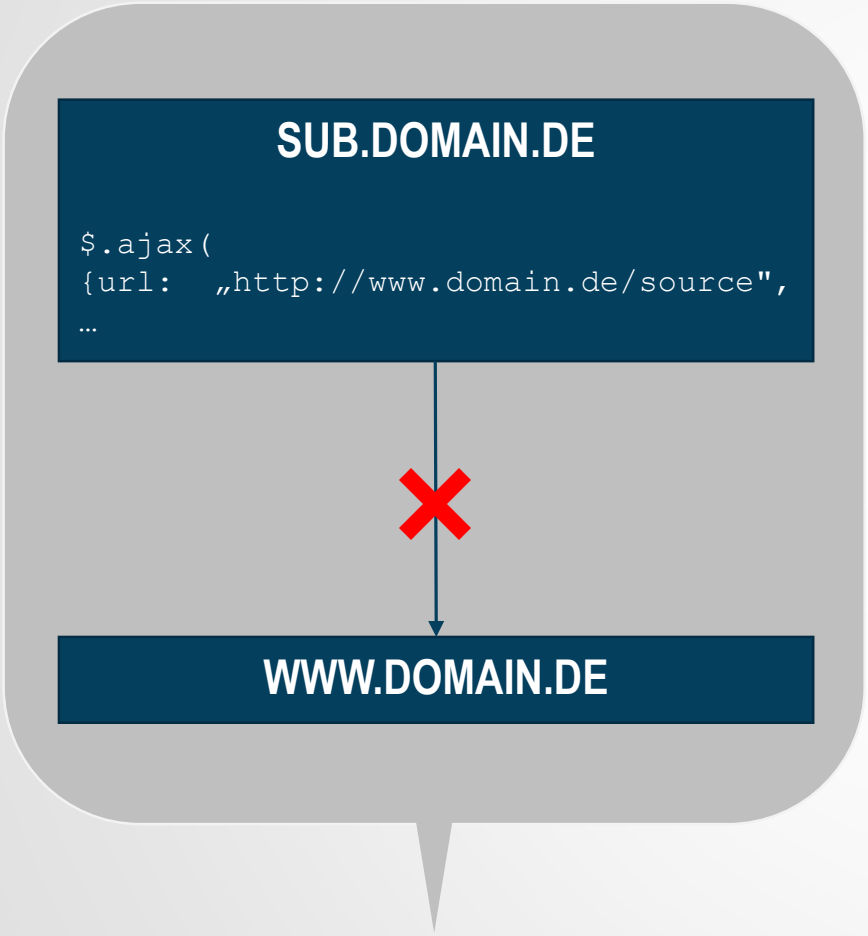
HERAUSFORDERUNG: BLOCKED REQUEST



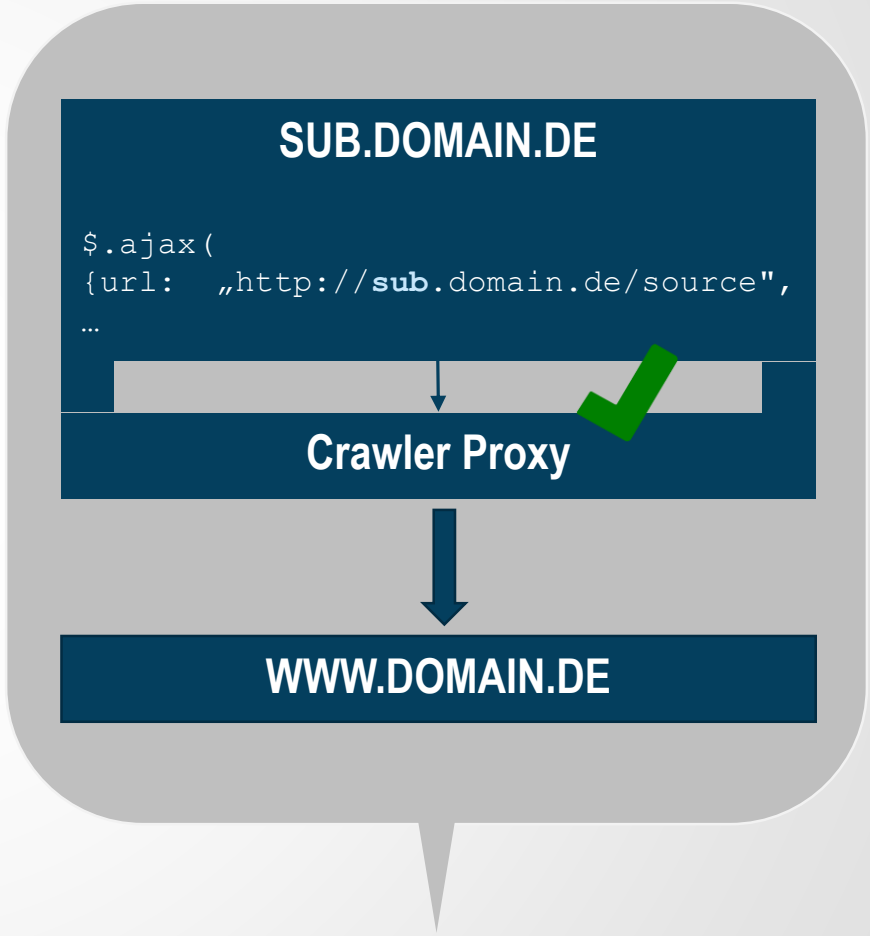
LÖSUNG A: HTTP HEADER ACCESS CONTROL

CROSS-ORIGIN RESSOURCE POLICY („CORS“): LÖSUNG ÜBER PROXY SKRIPT VIA REQUEST SERVER

Workaround: AJAX Request wird nicht direkt an den Zielserver gesendet, sondern an Request Server Proxy, der als Crawler den Response Server aufruft



HERAUSFORDERUNG: BLOCKED REQUEST



LÖSUNG B: AUFRUF VON CRAWLER PROXY

AJAX IM SEO-KONTEXT

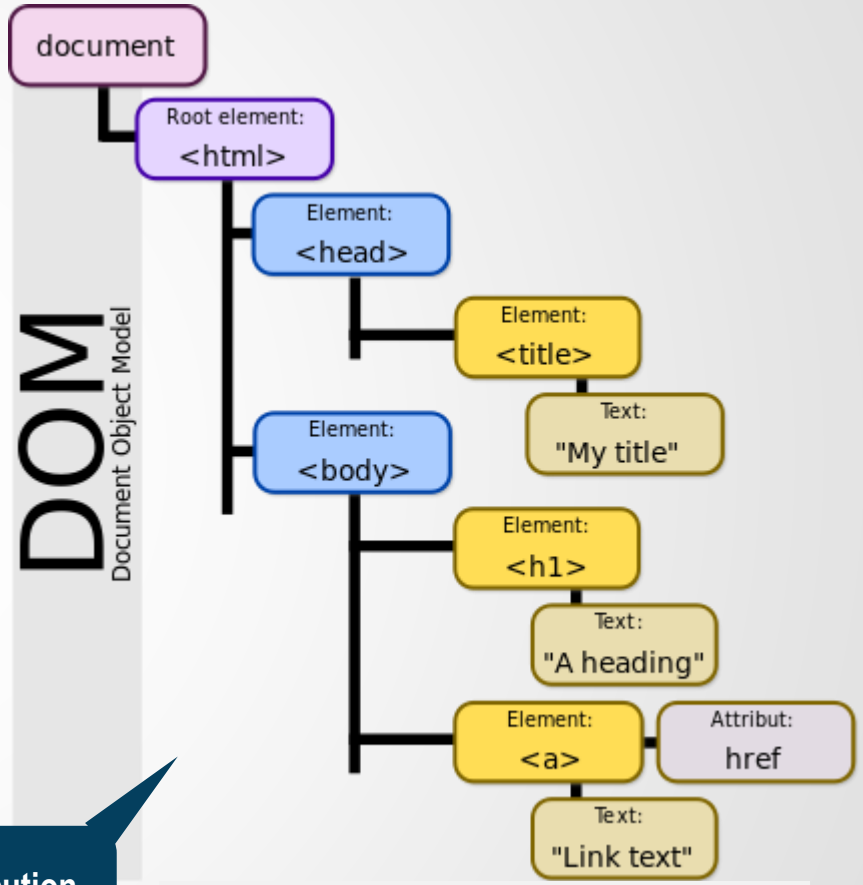
blueSummit

AJAX-BASED CONTENT DELIVERY ERKENNEN: SOURCE CODE VS. DOM

HTML-Content, der in DOM zu sehen ist, aber nicht im Quellcode, wurde in der Regel via AJAX geladen bzw. via JavaScript platziert

```
223                 <form action="/de/de/Hilfe-und-Kontakt" method="post" class="fo
224                     >
225                     <input type="hidden" name="action" value="searchAction"/>
226                     <input type="hidden" name="command" value="search"/>
227                     <input type="hidden" name="contactChannelIndex" value="0"/>
228                     <input type="hidden" name="searchType" value="1"/>
229                     <input type="hidden" name="sourceTaxonomy" value="Homepage2
230
231
232                     <div class="form-group">
233     <label for="search-classic-field" class="control-label sr-only">Suche</labe
234     <div class="input-group">
235     <input class="form-control" type="text" name="searchTerm" id="search-classi
236                         <button type="submit" class="btn btn-link"><span class=
237                             <i class="li li-form-search hidden-xs hidden-sm hid
238                             <span class="hidden-lg">Was suchen Sie?</span></but
239                     </span></div></div>
240                     <div id="lh-search-classic-store" class="hidden"></div>
241                 </form>
242             </div>
243         </div>
244
245
246
247             <div id="header-profile" class="lh-layerhandler" data-lh-layerhandl
248                 <a href="#" id="header-profile-toggle" class="lh-layerhandler-t
249                 Login<span class="sr-only">Open menu entry using the Enter, Spa
250
251                 <div id="header-profile-menu" class="lh-layerhandler-menu" aria
252
253                             <div id="login-layer" class="container-fluid">
254                                 <div class="row noSocial">
```

Plain HTML Code vor JavaScript Execution
→ Lesbar per Source Code Viewer



„Document Tree“ nach JavaScript Execution
→ Siehe Konsole, Firebug, Chrome Dev Tools

BILDQUELLE:
https://en.wikipedia.org/wiki/Document_Object_Model#/media/File:DOM-model.svg

AJAX & SEO: WIE GEHT GOOGLE IM JAHR 2017 MIT JAVASCRIPT UM?

Definition der Trigger-Events in JavaScript ist essenziell für die Transparenz der ausgespielten Inhalte gegenüber dem Google-Bot

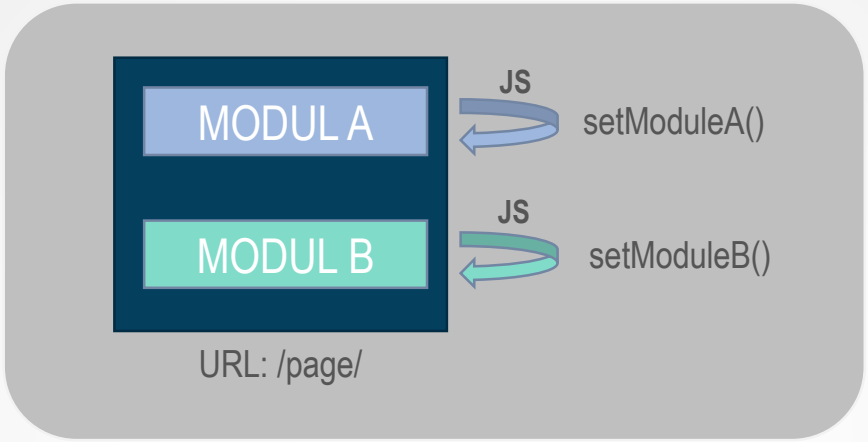
EVENT	OLD GOOGLEBOT*	NEW GOOGLEBOT**
GET DYNAMIC CONTENT SET „ON LOAD“		
GET DYNAMIC CONTENT SET „ON DELAY“		
GET DYNAMIC CONTENT SET „ON SCROLL“		
GET DYNAMIC CONTENT SET „ON CLICK“		
INDEX PAGES LINKED VIA JAVASCRIPT		

→ **HERAUSFORDERUNG: RISIKOFREIE, ABER ZUKUNFTSFÄHIGE LÖSUNG**

* Googlebot/2.1 (http://www.googlebot.com/bot.html)
** Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)

CRAWLABILITY VON JAVASCRIPT & AJAX BASED WEB APPS

Wo liegt die grundlegende Problematik?



AJAX BASED WEB APP

HERAUSFORDERUNGEN

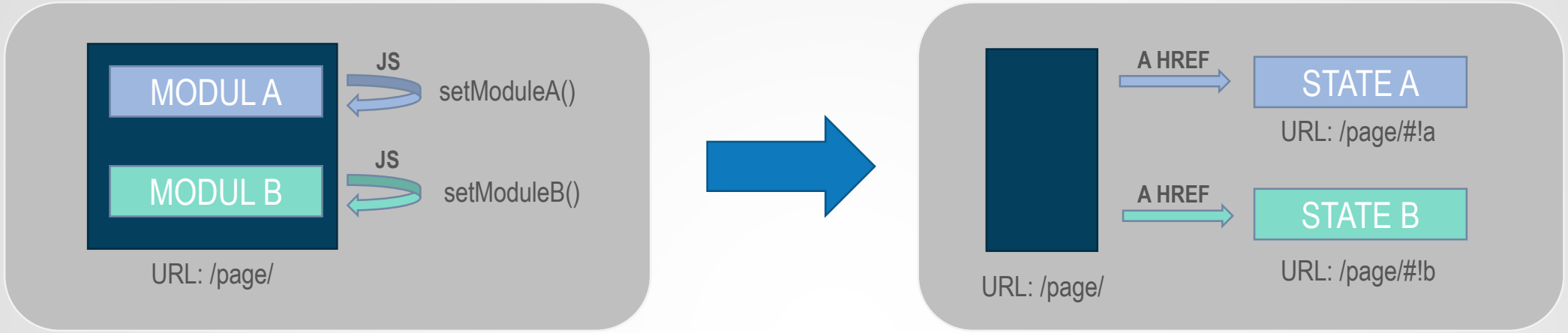
» Page- & Content-Updates nach User-Interactions (Klicks) liegen nicht „klassisch“ crawlbar in Form von Seiten & Verlinkungen vor

» Änderungen der URL-Location sind zwar via JavaScript simulierbar, beinhalten jedoch technische Hürden (Bookmarks, Back Button…)

→ Struktur eines AJAX-basierten Portals ist für den Crawler somit grundsätzlich erst einmal schwerer nachzuvollziehen

LÖSUNGSANSATZ (A): EINSATZ VON HTML SNAPSHOTS

Workaround mit statischen HTML-Abbildern einer JavaScript-basierten Web App



AJAX BASED WEB APP

HTML SNAPSHOTS

KOMBINIERTER LÖSUNGSANSATZ

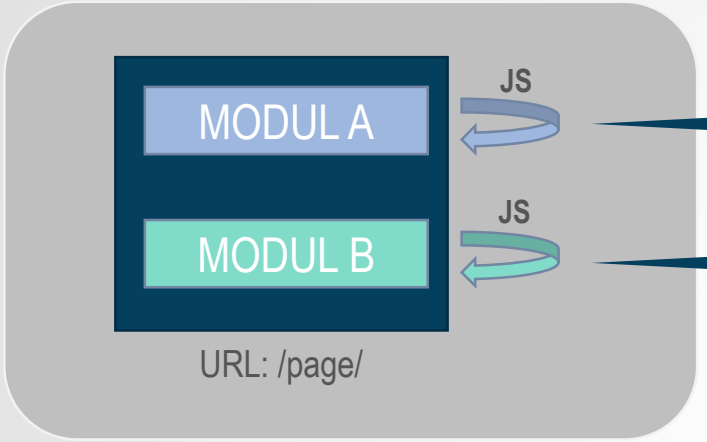
1. Bereitstellung der dynamischen Zustände der Web App via „Hashbang URLs“, wobei „#!“ statischen & dynamischen Part trennt
2. Für jeden Zustand ein HTML-Snapshot unter Pendant-URL mit „_escaped_fragment_“ statt „#!“ bereitstellen

» „#!“ signalisiert dem Bot die Existenz eines Snapshots (Alternative im HTML-Head-Bereich: <meta name="fragment" content="!">)

» Anwendungsserver muss sicherstellen, dass Hashbang- und Snapshot-Version einer URL den gleichen Inhalt liefern

LÖSUNGSANSATZ (B): URL HANDLING VIA HISTORY / PUSH STATE & POP STATE

Die JavaScript-Funktion history.pushState() erlaubt es, URLs zu ändern, mit einem Status zu verbinden & in die Chronik des Clients zu schreiben




```
$(".js-link").click (function(event)  {
     state = { /* alle Status-Daten */ };
     history.pushState(state,
          event.target.textContent,
          event.target.href);
     return event.preventDefault();
});

window.addEventListener('popstate',
          function(event) {
            eventHandling(event.state);
          }
);
```

Funktion history.pushState(status, title, url)

- Speichert den Status der Seite
- Verknüpft Status mit URL (+Title) und setzt URL
- Legt den Status in der Historie der Client-Chronik ab
- Ausführung sollte an jeden JavaScript-Link gekoppelt werden

Event „popstate“

- Wird automatisch abgefeuert, sobald der Client aus der Historie liest
- Wird somit bei jedem Besuch der Seite initial gefeuert
- Event Handling sollte Status der Seiten aus Historie herstellen

  …und wenn nicht verfügbar, Status auf Basis der URL “(re)konstruieren“

LÖSUNGSANSATZ (C): PROGRESSIVE ENHANCEMENT

Integriere JavaScript- & AJAX-Funktionalität in bestehende „klassische“ Links, um breite Funktionalität für Bot & User zu bieten

Suchmaschinenoptimierte Verlinkung von statischem Snapshot des Applikationsstatus „XY“

```
<a href="/path/snapshot-stateXY" onClick="createStateXY(); return false()">
        State XY
</a>
```

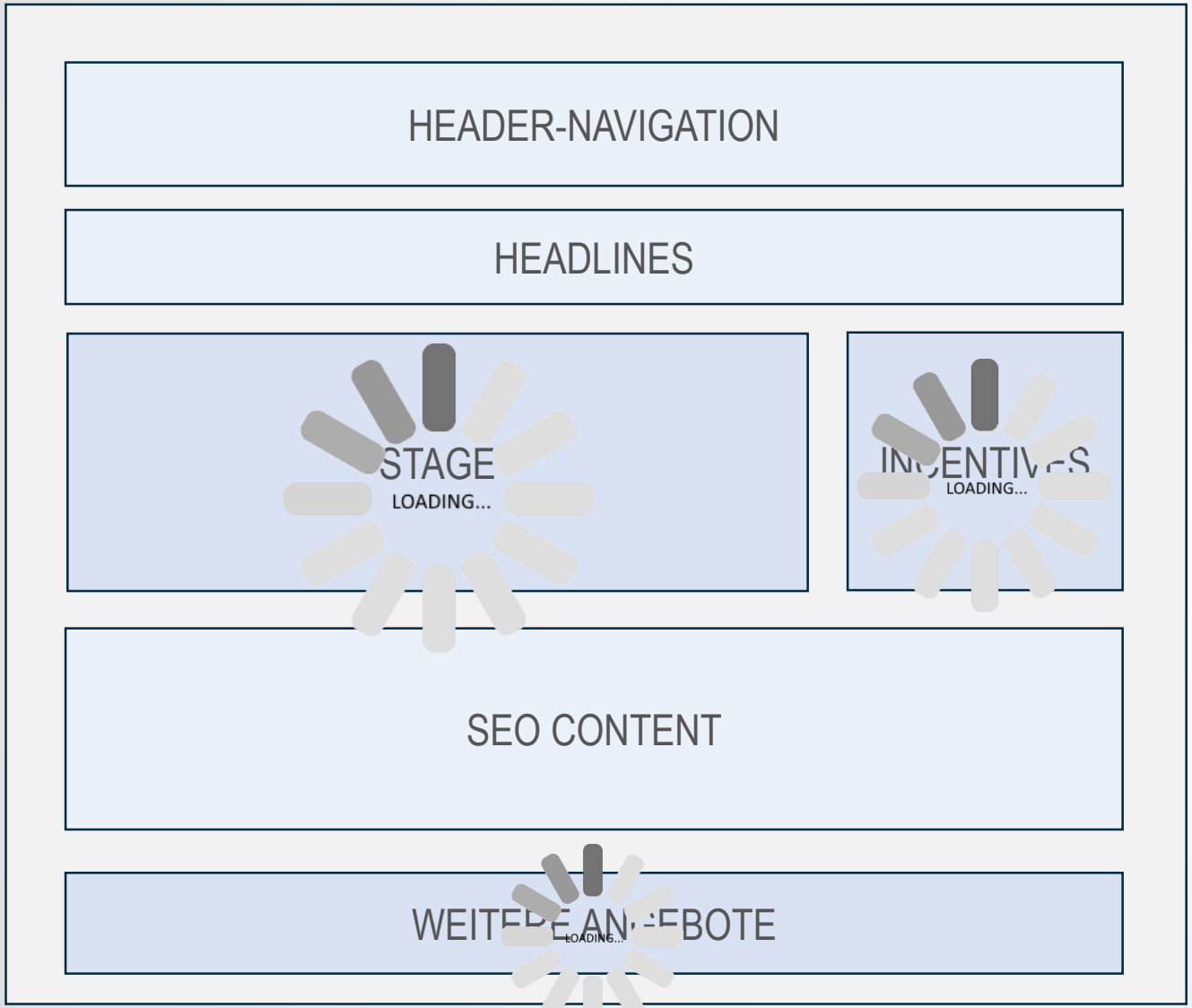
- Klickevent triggert Status „XY“
- „OnClick“ hat Prio vor „Href“
- „return false“ verhindert folgenden Visit des Href-Links

AJAX IM KONTEXT VON USABILITY

blueSummit

AJAX & PROGRESSIVES PAGE RENDERING

Dynamisierung der Seiteninhalte bei gleichzeitiger Optimierung des Pagespeeds



REDUZIERUNG DES INITIAL PAGE LOADS

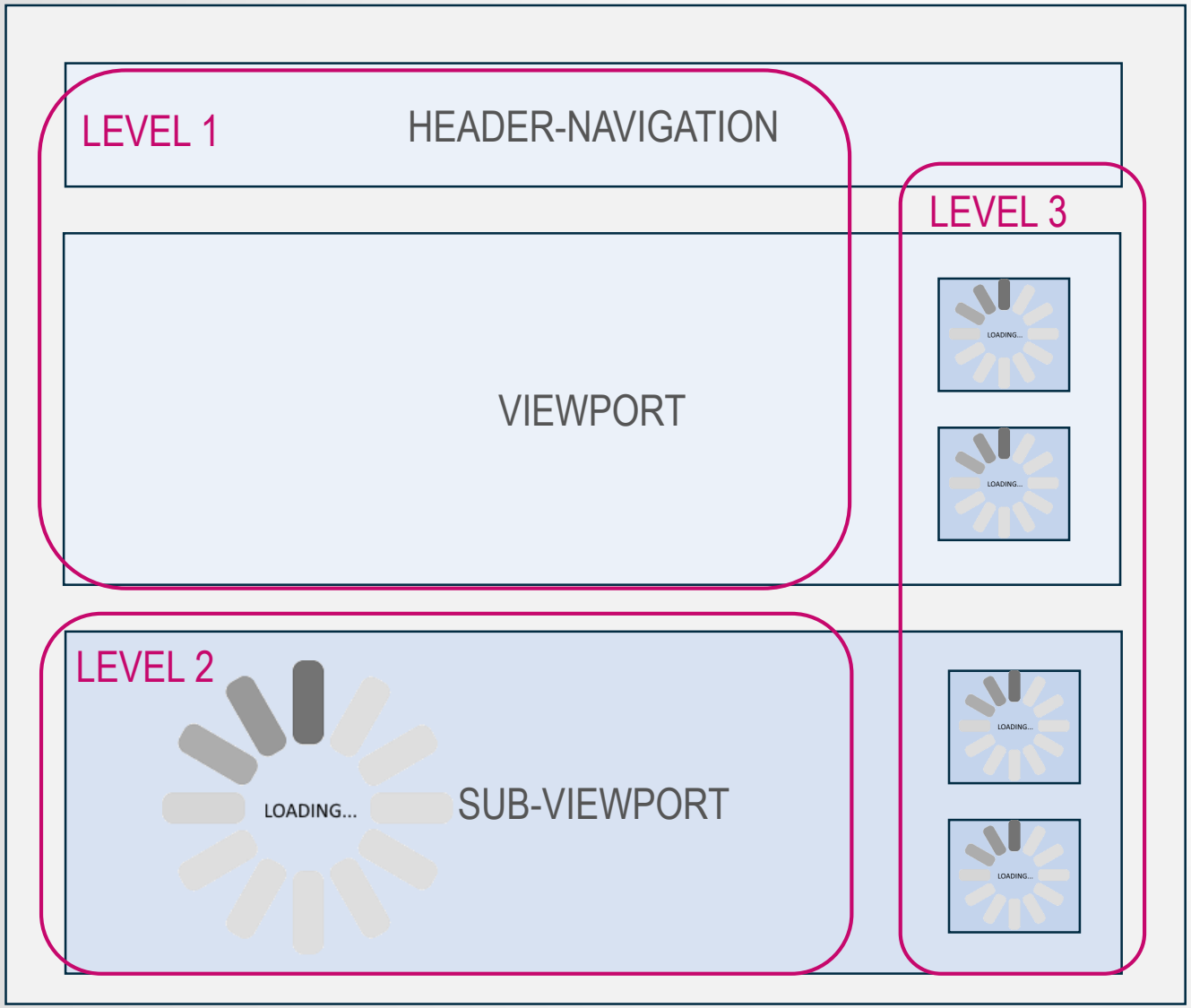
- Rahmen & Layout der Seite initial ausliefern
- Globale & statische Elemente (Header, Footer) initial ausliefern
- SEO-Content und SEO-sensitive Elemente initial ausliefern

STARK DYNAMISCHE ELEMENTE NACHLADEN

- Zeitsensitive Preise und Angebotsinformationen
- Dynamisierungen für weitere Kanäle
- Segmentierte Useransprache in SEO-unkritischen Modulen
- Personalisierte Useransprache in SEO-unkritischen Modulen

FRAMEWORK FÜR STRUKTURELLE PAGESPEED-OPTIMIERUNG

AJAX-basiertes Nachladen mit Fokus auf Viewport und nach Interaktionsgrad ausrichten



LEVEL 1: VIEWPORT

- Viewport = für den User ohne Scrollen sichtbarer Bereich (stark Device-abhängig)
- Viewport-Elemente sollten initial mit ausgeliefert werden und beim ersten Rendern sofort sichtbar sein
- Nach Möglichkeit sollte hier daher auf Nachlade-Prozesse verzichtet werden

LEVEL 2: SUB VIEWPORT

- Für den User nicht sofort sichtbarer Seitenbereich wird evtl. nie vom User gesehen werden
- Dieser Seitenbereich kann somit entweder asynchron oder erst als Reaktion auf ein „Scroll-Event“ nachgeladen werden
- Einschränkung: SEO-Relevanz des Contents

LEVEL 3: USER INTERACTIONS (CLICKS)

- Falls Content erst auf eine dedizierte Interaktion hin sichtbar wird (und keine hochgradige SEO-Relevanz besitzt), kann er nachgeladen werden

blueSummit

DISKUSSION

SEO-FRIENDLY AJAX - BULLSHIT-BINGO

Richtig oder falsch?

„JavaScript vererbt keinen Linkjuice!“

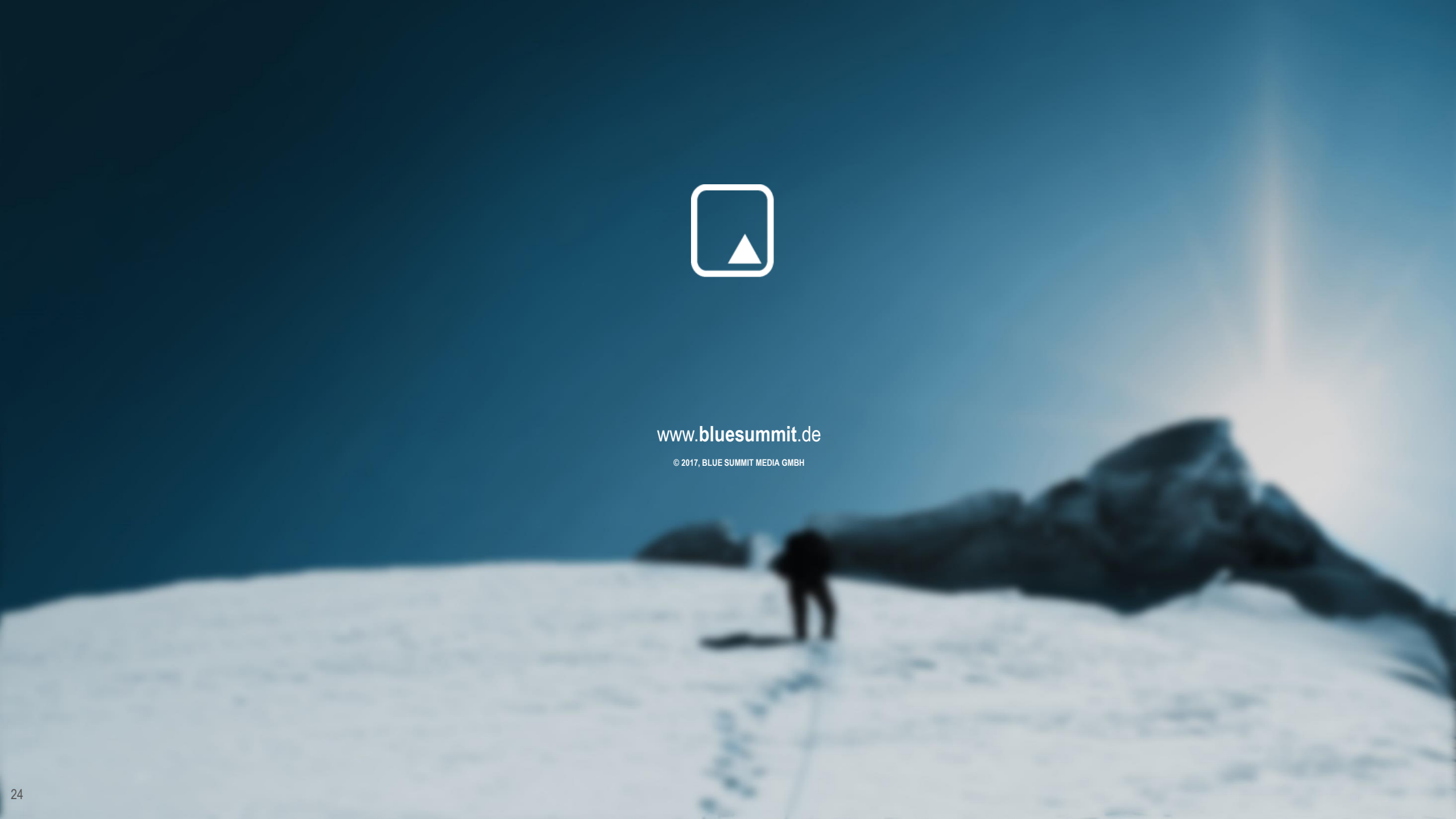
„Google sieht doch eh alles!“

„Wenn die Seite schnell lädt, ist alles kein Problem!“

„Google kann alles crawlen,
muss aber seine Ressourcen managen!“

„Alles muss im HTML-Code stehen!“

„Google kann JavaScript,
Bing, Baidu & Yandex nicht!“



www.**bluesummit**.de