

Web-basierte Systeme

o8: Architektur moderner Browser

Wintersemester 2024

Rüdiger Kapitza



Lehrstuhl für Informatik 4
Systemsoftware



Friedrich-Alexander-Universität
Technische Fakultät

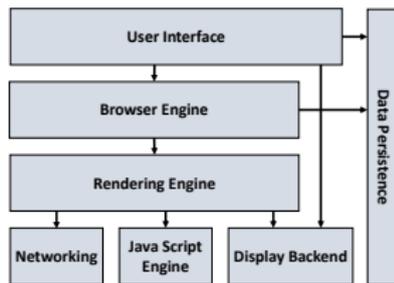
Architektur moderner Browser

Vorläufiger Vorlesungsplan

- 16. Oktober Einführung und Darstellung von Webseiten
- 23. Oktober HTML und CSS
- 30. Oktober Hypertext Transfer Protocol
- 6. November Browser Schnittstellen
- 13. November Kommunikationsschnittstellen im Browser
- 20. November WebAssembly
- 27. November **Architektur moderner Browser**
- 4. Dezember Clientseitige Architekturmuster und serverseitige Implementierung von Web-basierten Systemen
- 11. Dezember Vorbereitung Papieranalyse
- 8. Januar Papieranalyse
- 15. Januar Lastverteilung durch Zwischenspeicher
- 22. Januar Web3
- 29. Januar Aspekte von Web Sicherheit
- 5. Februar Zusammenfassung und Ausblick

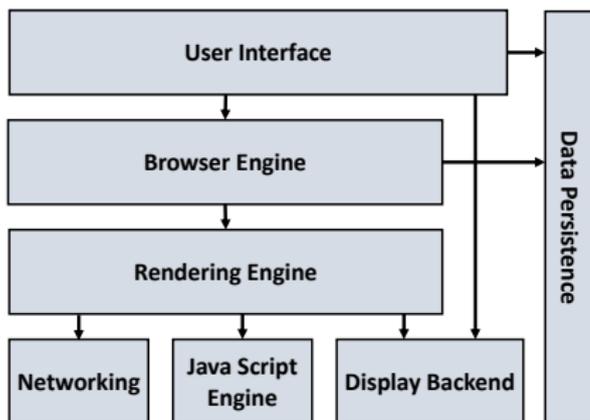
Zielsetzung der Lerneinheit

- Allgemeines Verständnis für die Architektur eines Browsers
- Verständnis für Fusion die von HTML, CSS & JS zu einem Layout
- Kennenlernen von Systemansätzen zur Verbesserung der Leistung und Sicherheit von Browsern
 - Beispiel: Chromium



Generische Softwarearchitektur eines Browsers

- Vgl. „A Reference Architecture for Web Browsers“ (von 2005!) [2]
- Die Arbeit stellt eine generische Softwarearchitektur vor, die durch die Analyse verschiedener Browser entwickelt wurde.



User Interface

- Schicht zwischen Nutzer und der Engine des Browsers
- Stellt die Oberfläche bereit: Werkzeugleiste, Rahmen mit Fortschrittsanzeige, Masken für Einstellungen, usw.
- Nicht selten Einbettung in den Desktopmanager des Systems

Browser Engine

- Implementiert die Grundfunktionen des Browsers und bietet eine Schnittstelle zur Rendering-Engine
- Funktionen sind z.B: Laden einer URI, Navigieren im Verlauf der bisher aufgesuchten Seiten oder erneutes Laden einer Seite
- Schnittstelle zur Abfrage von Sitzungsinformationen
 - Z.B.: Fortschrittsanzeige oder Warnungen der JavaScript-Engine
- Schnittstelle zur Statusabfrage der Rendering-Engine und zu deren Parametrierung

Rendering Engine

- Zuständig für die Darstellung einer HTML-Seite unter Verwendung von CSS und der Integration von Grafiken
- Es wird das genaue Layout festgelegt (Positionierung aller Elemente) und aktualisiert, sobald weitere Informationen verfügbar sind
- Zentraler Bestandteil des Rendering Engine sind HTML- & CSS-Parser

Networking

- Unterstützung für den Transfer von Daten via HTTP(S) (TCP/QUIC)
- Konvertierung zwischen Zeichensätzen, sowie Unterstützung für MIME media Typen
- Unterstützung für die Zwischenlagerung aktueller Daten

JavaScript Interpreter

- Unterstützung für JavaScript (ECMAScript)
- Interaktion mit der Rendering Engine

XML Parser

- Ursprünglich separates Architekturelement, heute wahrscheinlich in andere Komponenten integriert

WebAssembly

- Mittlerweile neu hinzugekommen

Display Backend

- Stellt Grafikprimitive bereit
- Schnittstellen zum Desktopmanager und dem Betriebssystem
 - Bspw.: Look & Feel unter Linux vs. Windows und Schrifttypen

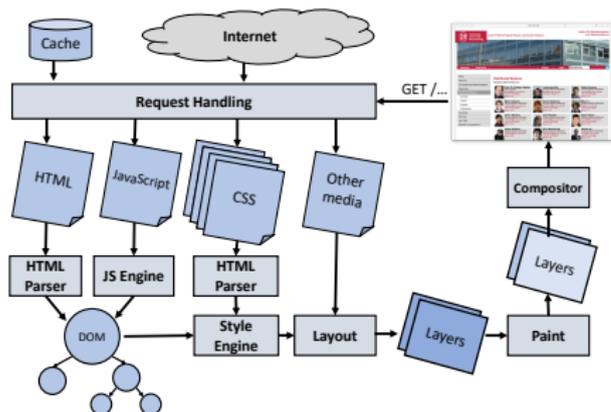
Data Persistence

- Verwaltung von Sitzungsdaten
- Verwaltungsdaten: Lesezeichen und Einstellung der Oberfläche
- Zwischenspeicher (Cache)
- Sicherheitsinformationen: Zertifikate etc.

Architekturüberblick (Firefox Quantum)

Architekturüberblick mit Fokus auf Firefox Quantum

- Quelle: Blog-Artikel zur Funktionsweise von Quantum¹
- *Achtung*: Artikel behandelt die *Browser Engine* – im Sinne von [2] entspricht dies aber *eher* der Rendering Engine



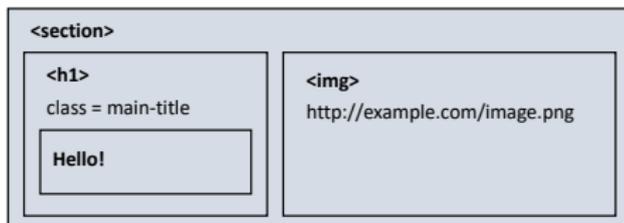
¹<https://hacks.mozilla.org/2017/05/quantum-up-close-what-is-a-browser-engine>

Vereinfachter Ablauf: Laden und Parsen

- Daten werden vom Modul Request Handling geladen
 - HTML, Grafiken, CSS und JavaScript
- Der HTML-Parser wandelt HTML-Dokumente in ein Document Object Model (DOM) um - einen so genannten Inhaltsbaum
- Andere Styledaten und Skripte, die im HTML enthalten oder referenziert sind, werden ebenfalls geladen und verarbeitet.
- Skripte können die Seitenstruktur und Styledaten verändern, bevor ein HTML-Dokument vollständig geparkt wurde

Vereinfachter Ablauf: Laden und Parsen

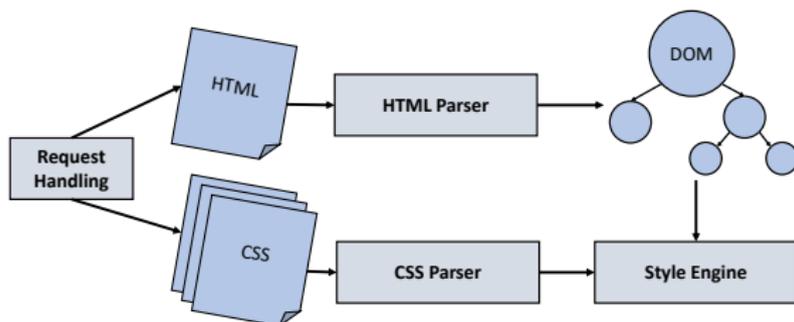
```
1 <section>
2   <h1 class="main-title">Hello!</h1>
3   
4 </section>
```



Architekturüberblick (Firefox Quantum)

Vereinfachter Ablauf: Verarbeitung von CSS

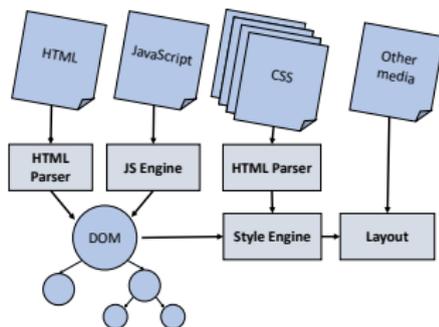
- Style-Anweisungen werden ebenfalls geparkt und auf das DOM angewendet, dies übernimmt die *Style Engine*
- Sind alle Style-Anweisungen ausgeführt worden, spricht man von einem *computed style*
 - Wirken mehrere Anweisungen auf ein Element, überschreibt die zuletzt angewandte Anweisung die vorhergehenden.



Architekturüberblick (Firefox Quantum)

Vereinfachter Ablauf: Erzeugen des Layouts

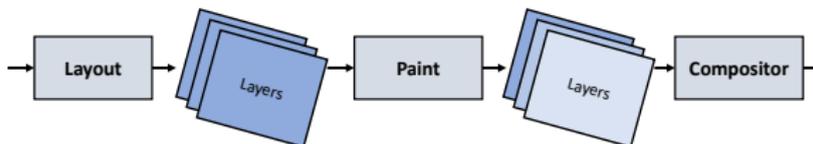
- DOM und die ermittelten Style-Anweisungen werden an die Layout Engine übergeben
- Die Layout Engine berechnet das Layout basierend auf der gegebenen Fenstergröße
- Dabei wird jedes Element einzeln betrachtet, entsprechend dimensioniert und alle Style-Anweisungen werden angewendet



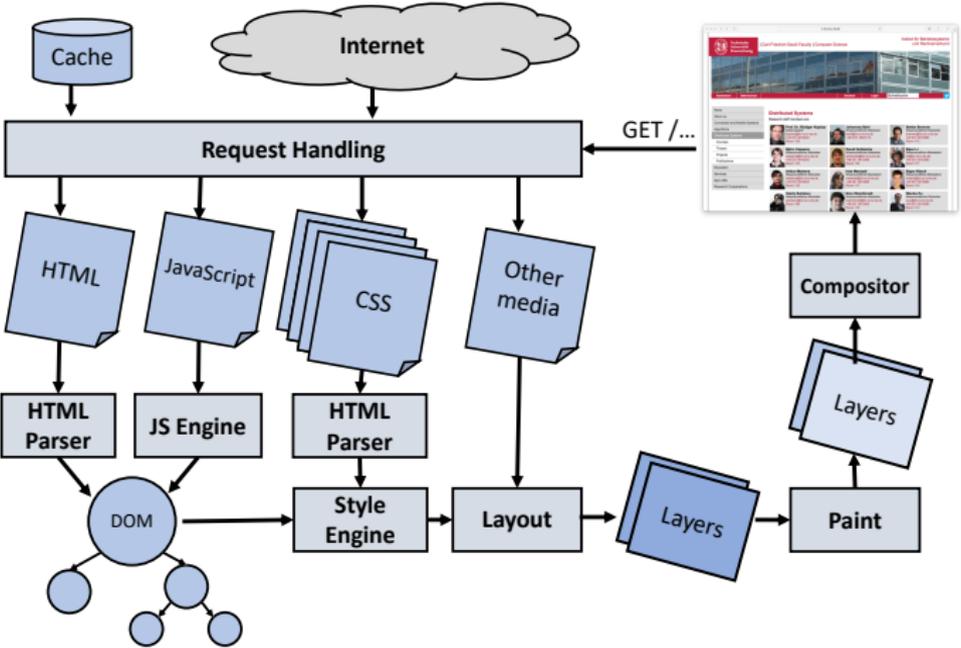
Architekturüberblick (Firefox Quantum)

Vereinfachter Ablauf: Darstellung im Browser

- Im letzten Schritt werden die berechneten Elemente mit allen Informationen dargestellt und die Seite für den Benutzer sichtbar gemacht
- Wenn der Benutzer den sichtbaren Bereich der Seite ändert, muss die Seite neu gezeichnet werden
 - Dies kommt häufig vor, und Browser berechnen entsprechend mehr als den Viewport, um schnell reagieren zu können



Architekturüberblick (Firefox Quantum)



Mehrprozessbrowserarchitektur & Isolation von Webanwendungen

Motivation

- Browser werden immer komplexer
- Webanwendungen nehmen ebenfalls in der Komplexität zu!
 - Beispiel: Single Page Apps (SPAs) & Progressive Web Apps (PWAs)
- Fehler in Browsern und Angriffe sind an der Tagesordnung
- Bedarf nach einer widerstandsfähigen Browser-Architektur

Ausgangspunkt

- Frühere Browser wurden als *Monolithen* konzipiert!
- Alle Architekturkomponenten haben vollen Zugriff
 - Zum Beispiel auf das Dateisystem
- Fehler können zum Absturz des gesamten Browsers führen
- Schlecht Isolation auch bezgl. Performance

Mehrprozessbrowserarchitektur & Isolation von Webanwendungen

Zielsetzung für die Mehrprozessarchitektur von Chromium

- Etablierung von Abstraktionen zur besseren Isolierung der einzelnen Komponenten eines Browsers
- Isolationskonzepte müssen so weit wie möglich mit den bestehenden Systemannahmen kompatibel sein
- Eine Mehrprozessarchitektur bietet bessere Sicherheit, Fehlertoleranz, Ressourcenverwaltung und Ausführungsgeschwindigkeit

Literatur

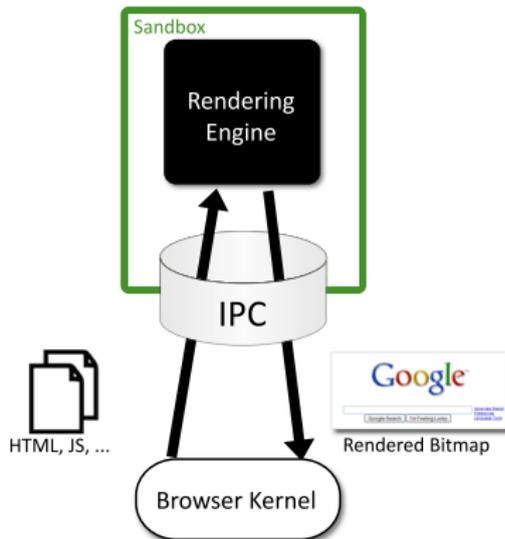
- *The Security Architecture of the Chromium Browser* [1]
- <https://www.chromium.org/developers/design-documents/multi-process-architecture>
- *Isolating Web Programs in Modern Browser Architectures* [3]
- <https://www.chromium.org/developers/design-documents/site-isolation>

Angreifermodell

- Entfernter Angreifer, der eine Lücke im Code des Browsers kennt
- Um die Schwachstelle auszunutzen, muss der Angreifer den Benutzer dazu bringen, Schadcode zu laden
- Ziele des Angreifers
 - Installation von Schadsoftware wie z.B. Keylogger
 - Entwenden von wichtigen Daten/Dateien
- Angriffe, die nicht berücksichtigt werden, weil sie nicht durch die Multiprozessarchitektur adressiert werden:
 - Phishing und Schwachstellen in Webanwendungen (z.B. für cross-site scripting (XSS))

Architekturüberblick

- Architektur wie 2008 etabliert ([1])



Funktionsaufteilung

- Rendering Engine
 - HTML/XML/XSLT/CSS parsing, Bilder dekodieren, SVG, Document Object Model (DOM), JavaScript Interpreter
 - Sonstiges: regulärer Ausdrücke
- Browser Engine
 - Verwaltung von persistenten Information
 - Zugriff auf native Grafikelemente bzw. den Fenstermanager
 - Oberfläche des Browsers (z.B. Location Bar)
 - Netzwerkfunktionen
 - *Achtung*: Die Browser Engine ist wesentlich weiter gefasst als in [2]
- Gemeinsame Funktionen
 - URL und Unicode parsing

Ziel der Aufteilung

- Rendering Engine enthält komplexen Code, der Webanwendungen ausführt und eher fehleranfällig ist
- Browser Engine enthält hauptsächlich Code, der direkt auf das System zugreifen kann

Sandbox

- Entzieht der Rendering Engine alle Rechte und verhindert den direkten Zugriff auf das System.
- Stattdessen wird die Schnittstelle der Browser Engine für alle wichtigen externen Aufrufe genutzt

IPC (Inter-Process Communication)

- Schneller und effizienter Austausch zwischen Browser Engine und Rendering Engine
- Low-level IPC – ursprünglich über *async. named pipes* realisiert
- Inzwischen eigenes System (Mojo²), das auf verschiedene Subsysteme (z.B. Unix Domain Sockets) abgebildet werden kann

²<https://chromium.googlesource.com/chromium/src/+master/mojo/README.md>

Granularität der Aufteilung

- Vereinfacht kann (vorerst) angenommen werden, dass jedes *Fenster/Tab* einer eigenen Rendering-Engine-Instanz entspricht.
- Wichtige Informationen, wie z.B. Fehlermeldungen zu HTTPS-Zertifikaten oder Warnungen vor Phishing-Versuchen, werden über eine separate Instanz kommuniziert
- Ausnahmen gibt es bei Warnungen zu dedizierten Seiteninhalten, diese werden in der verarbeitenden Rendering Engine erzeugt

Externe Hilfsprogramme (Plug-ins)

- Hilfsprogramme werden als eigene Prozesse betrieben (evtl. auch in einer Sandbox)
- Es gibt in der Regel nur eine Instanz pro Plug-in

Schnittstelle der Browser Engine

- Rendering
 - Alles wird gezeichnet und an die Browser Engine übergeben
 - Browser Engine übergibt die Bilddaten an die entsprechenden Grafikschnittstellen zur Darstellung
 - Im Rahmen der IPC bedeutet dies einen zusätzlichen Kopiervorgang (!)
- Benutzereingaben
 - Alle Benutzereingaben werden vom Betriebssystem an die Browser Engine übergeben
 - Browser Engine wertet Eingaben entweder direkt aus oder gibt sie je nach Fokus an die Rendering Engine
 - Rendering Engine erhält nur Ereignisse, die sich direkt an die ausgeführte Webanwendung gerichtet sind

Schnittstelle des Browser Engine (Fortsetzung)

- Persistenter Speicher
 - Aufgabe der Sandbox ist es, zu erzwingen, dass die Rendering Engine (bspw. bei Übernahme) nicht auf Dateien zugreifen kann
 - Vom Browser initiierte Dialogfenster informieren den Benutzer über Up- und Downloads.
 - Bei Downloads ist der Speicherort der Dateien voreingestellt, z.B. ein Standardverzeichnis
 - Ziel ist es, das Überschreiben von Dateien oder das Ablegen an kritischen Stellen zu verhindern

Schnittstelle des Browser Engine (Fortsetzung)

■ Netzwerk

- Netzwerkverkehr wird durch die Browser Engine abgewickelt
- Rendering Engine stellt Anfragen an den Browser Engine
 - Bei Protokollen wie `http(s)` (und `ftp`) lädt die Browser Engine die Daten und übergibt sie an die Rendering Engine
 - Typischerweise wird `file`-URIs blockiert, da die Rendering-Engine sonst Zugriff auf das Dateisystem hätte
 - Ausnahme, wenn der Benutzer z.B. über die Location Bar explizit eine solche URL eingibt. In diesem Fall wird die URL über eine separate Rendering Engine Instanz verarbeitet

Evaluation

- Wirksamkeit wird überprüft mit Hilfe der Betrachtung von *Common Vulnerabilities and Exposures (CVEs)*³
- Betrachtung der CVEs zwischen Juli 2007 und Juli 2008 für Internet Explorer, Firefox und Safari
- Frage: Welche der veröffentlichten Schwachstellen können durch die vorgeschlagene Architektur vermieden werden?
 - Antwort: 38 von 87 der Rendering Engine zugeordneten Schwachstellen können verhindert werden
 - Dies entspricht 70,4% (38 von 54) aller Schwachstellen, die die Ausführung von beliebigem Code erlauben
- Mehr Details im technischen Bericht [1]

³<https://cve.mitre.org>

Isolation von Webanwendungen

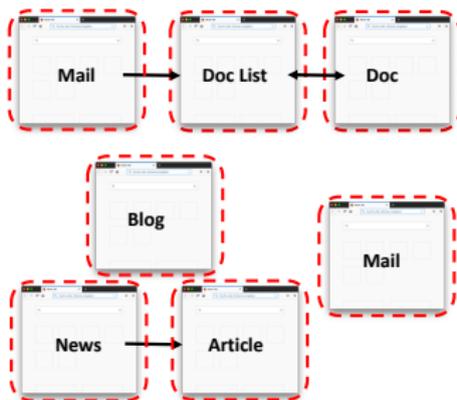
- Verschiedene unabhängige Web-Anwendungen werden im Kontext eines Browsers ausgeführt
- Vergangenheit: **Alle** Anwendungen einem Prozess zugeordnet!
- Wie kann man komplexe Webanwendungen auf eine Mehrprozessbrowserarchitektur abbilden?



Isolation von Webanwendungen

Naiver Vorschlag

- Jedes Fenster/Tab erhält seinen eigenen Prozess
- Bricht mit den Annahmen typischer Webanwendungen, die Daten *miteinander* austauschen!
 - Gegenseitiger Zugriff auf den DOM nicht mehr möglich



Isolation von Webanwendungen

Zielsetzung

- Entwicklung einer Abstraktion, die Webanwendungen identifizierbar macht und ihre Isolation ermöglicht
- Gruppierung von zusammengehörigen Fenstern
 - Sollte den intuitiven Annahmen entsprechen und kompatible zu existierenden Webanwendungen sein
- Idee: Ausnützen von existierenden Informationen
- Zusammengehörige Fenster sollen durch einen Prozess ausgeführt werden



Annäherung durch eine *ideale* Abstraktion

- **Webanwendung** (Web Program)
 - Menge von verknüpften Seiten und zugeordneten Ressourcen, die einen Dienst implementieren
- **Instanz einer Webanwendung** (Web Program Instance)
 - Eine laufende Instanz einer Webanwendung in einem Browser
 - Isolation erfolgt mit Unterstützung der Browserarchitektur

- Wie kann das konkret umgesetzt werden?

Isolation von Webanwendungen

Isolation auf Grund des Website

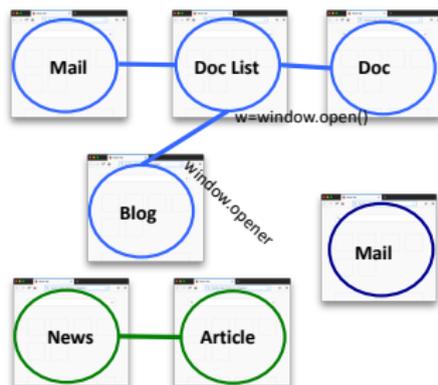
- *Same Origin Policy* erzwingt bereits eine gewisse Isolation auf der Basis von Host, Protokoll und Portnummer
- Beschränkung nur im Rahmen des Domännennamen
- Website ist durch den Domännennamen, Protokoll und Port definiert



Isolation von Webanwendungen

Browsing Instanzen

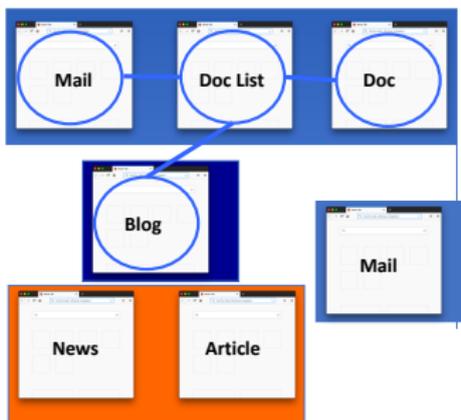
- Referenzen zwischen *verwandten* Fenstern
 - Eltern/Kind Relation durch `window.open()`
 - Zugriff auf den DOM der geöffneten Seite
 - Lebenszyklus eines Fensters
- Verbindung von Fenstern unabhängig vom Website



Isolation von Webanwendungen

Website Instanzen

- Schnittmenge von Website und Browsing Instanz
- Ermöglicht sichere Isolation von anderen Seiten
- Bildet eine kompatible Variante einer Webanwendung



Evaluierung

- Performance Isolation
 - Im Vergleich zu einer monolithischen Konfiguration reagiert der Browser unter Last wesentlich schneller auf externe Ereignisse
 - Verbesserte Geschwindigkeit durch Ausnutzung von Mehrkernsystemen
- Verzögerung durch die Erzeugung eines Prozesses ist bzgl. der Laufzeit nicht relevant
- Speicherbedarf nimmt spürbar zu, ist aber zu erwarten und ein akzeptabler Kompromiss

- Browser bestehen aus einer Vielzahl von Komponenten
 - Speziell die Rendering Engine ist kritisch, da sie entscheidet ob Webseiten einheitlich dargestellt werden
 - Aktuell gibt es wenige Kernsysteme die von verschiedenen Browsern genutzt werden (z.B. Webkit, Blink und Gecko)
- Ähnlich zu Betriebssystemen haben sich Mehrprozessarchitekturen auch bei Browsern durchgesetzt
 - Gründe sind Sicherheit, Fehlertoleranz aber auch Leistung
 - Weitere Informationen: *Site Isolation: Process Separation for Web Sites within the Browser*
<https://www.usenix.org/conference/usenixsecurity19/presentation/reis>
- Betrachtete Architektur Aspekte kratzen nur an der Oberfläche
 - Bspw. sind die internen Abläufe komplexer als dargestellt

Literatur

- [1] Adam Barth et al. ***The Security Architecture of the Chromium Browser***. Jan. 2008.
- [2] Alan Grosskurth und Michael W. Godfrey. **“A Reference Architecture for Web Browsers”**. In: *Proceedings of the 21st IEEE International Conference on Software Maintenance*. ICSM '05. IEEE Computer Society, 2005, S. 661–664.
- [3] Charles Reis und Steven D. Gribble. **“Isolating Web Programs in Modern Browser Architectures”**. In: *Proceedings of the 4th ACM European Conference on Computer Systems*. EuroSys '09. Nuremberg, Germany: ACM, 2009, S. 219–232.