



Caching Strategien



Falko Höfte, Alexander Schultenkemper, Bastian Mildenerger, Simon Vallo

Agenda

- 1) Einleitung
- 2) Motivation für Caching
- 3) Caching-Strategien
 - 3.1) Allgemeine Strategien
 - 3.2) Caching in Ruby on Rails
- 4) Was muss beim Caching beachtet werden?
- 5) Fazit

1. Einleitung

- Anforderung und Erwartungen immer höher
- Kontinuierliche Verbesserung erforderlich
- Vorstellen von modernen ausgewählten Techniken



1. Allgemeines zu Caching

- › Mehrfach angeforderten Inhalt „auf Abruf zu halten“
- › Zwischenspeicherung auf schnellere Speichermedien
- › Einfaches und kostengünstiges Prinzip
- › In Verbindung mit weiteren Leistungsoptimierungen



2. Motivationsgründe

Wirtschaftlicher Faktor

- Angewiesen auf ständige Verfügbarkeit
- Zusammenbruch bei zu hoher Last
- Mögliche Umsatzeinbußen
- Einsparung von Ressourcen
- Reduzierte Hardwareanforderung



2. Motivationsgründe

Kundenzufriedenheit

- Signifikante Rolle für das weitere Nutzen
- Antwortzeiten tragen zur *User-Experience* bei
- Doppelte Zeit → Rückgang des Nutzen von 20%
- Benutzer reagieren sensibel
- Weniger Umsatz



2. Motivationsgründe

Suchmaschinen-Ranking

- Suchmaschinen präsentieren möglichst wertvolle Inhalte
- Einbeziehung der Antwortzeiten
- Je performanter desto „wertvoller“



Allgemeine Strategien



3. Caching Strategien – Allgemeine Strategien

› Unterteilung in ...

- › Clientseitiges Caching
- › Serverseitiges Caching
- › HTTP-Header Caching
 - › Caching auf dem Client
 - › Server jedoch maßgeblich beteiligt



Clientseitiges-Caching



3.1.1 Clientseitiges-Caching

- › Web-Storage (auch DOM-Storage / Supercookies)
 - › Lokale Speicherung auf dem Client
 - › Key-Value-Paare
 - › Session-Storage
 - › Speicherung für den Zeitraum des Besuches auf der Website
 - › Local-Storage
 - › Dauerhafte Speicherung der Daten



3.1.1 Clientseitiges-Caching

› Indexed DB

- › Ehemals Web-SQL-Datenbank (deprecated!)
- › Seit HTML5
- › NoSQL-Datenbank zur permanenten Speicherung auf dem Client
 - › Not-Only-SQL
 - › Schemalose Datenbank
 - › Daten werden als Key-Value-Paare in DB gespeichert
 - › Unterstützt Indizes zur Performanceoptimierung

3.1.1 Clientseitiges-Caching

› Application Cache

- › Seit HTML5
- › Ermöglicht Arbeiten im Offline-Modus
 - › Seite lokal gespeichert
 - › Schnelles Laden
 - › Entlastung des Servers durch Manifest-Datei
- › Einbindung im HTML-Tag:

```
<html manifest="example.mf"></html>
```

3.1.1 Clientseitiges-Caching

Application Cache – Beispiel einer Manifest-Datei

Cache Manifest

%Version: 1.2.3, Datum: 20.12.2012

CACHE:

/favicon.ico

index.html

haupt-css.css

bilder/logo.png

NETWORK:

login.php

/member-bereich

FALLBACK:

/main.php

bilder/vorschau

/fallback.html

bilder/offline.png



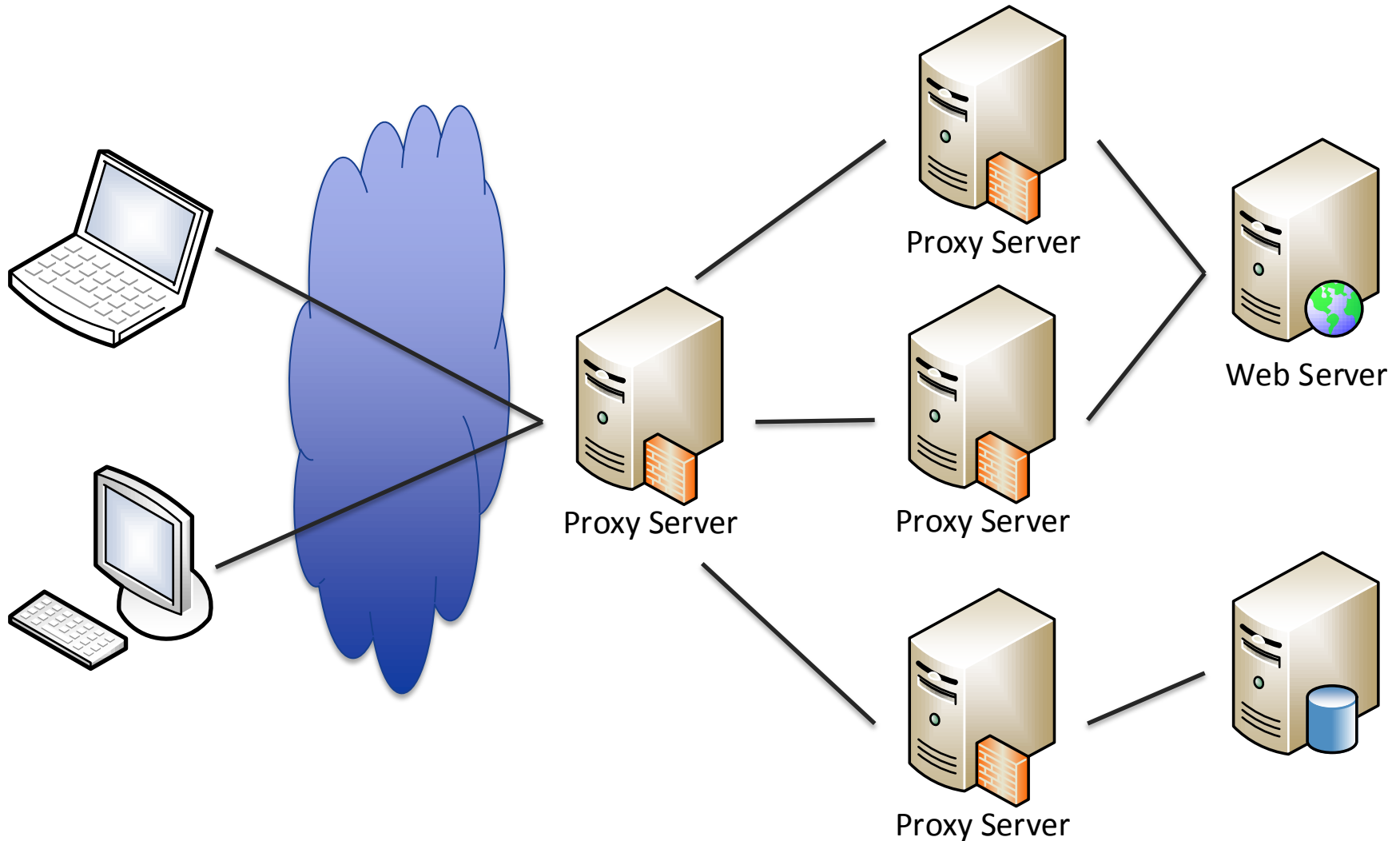
Serverseitiges-Caching

3.1.2 Serverseitiges-Caching

› Reverse-Proxy-Caching

- › Proxy vermittelt Anfragen an Webserver
- › Cached bereits Content vor und verteilt diesen an die Anfragenden Clients → Entlastung des/der Webserver (Loadbalancing)
- › In der Realität auch oft Hierarchie von Reverse-Proxy-Server
 - › Lastverteilung untereinander → Netzsicherheit

3.1.2 Serverseitiges-Caching



3.1.2 Serverseitiges-Caching

› Memcached-Caching

- › Vorhalten der gecachten Daten im Arbeitsspeicher
- › Verteiltes System → Daten werden auf Server (Memcached-Daemon) aufgeteilt
- › Speicherung der Daten mittels Consistent-Hashing-Verfahren
 - › Server werden auf Wertebereich des Hashwertes aufgeteilt
 - › Je nach Hashwert einer Datei wird diese auf den zutreffenden Server verteilt
 - › Server regelt dann intern mit neuer Hashfunktion, wo Daten abgelegt werden
- › Last-Recent-Use-Prinzip

HTTP-Header-Caching



3.1.3 http-Header-Caching

› Der „expires“-Header

- › Liefert Metainformationen über Verfallsdatum einer Datei
- › Client weiß, wann er die Datei neu anfragen muss
- › Funktion muss auf dem Webserver aktiviert werden

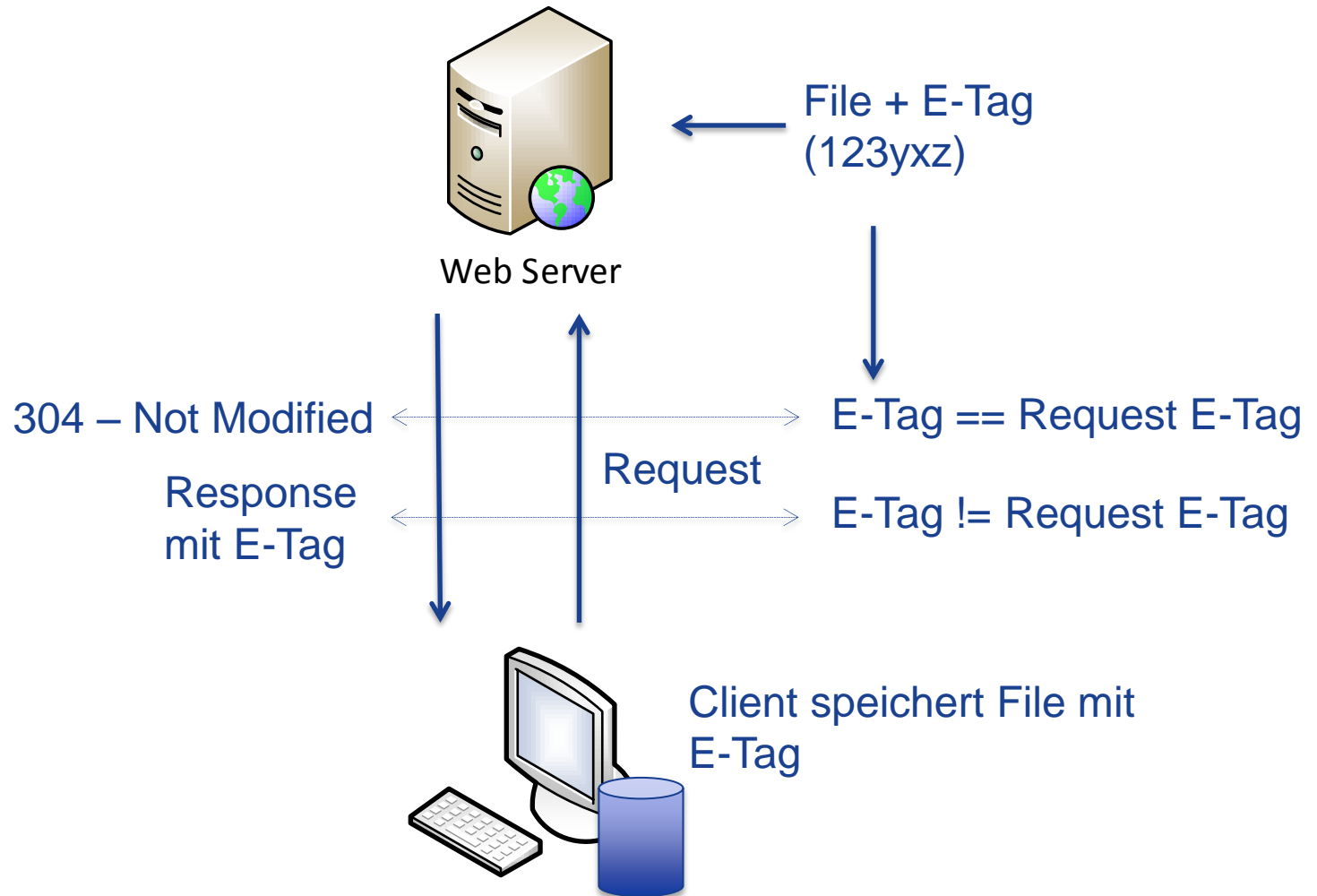
```

x Headers Preview Response
Remote Address: 173.194.39.4:80
Request URL: http://www.google-analytics.com/ga.js
Request Method: GET
Status Code: 200 OK (from cache)
▼ Request Headers CAUTION: Provisional headers are shown.
  User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.114 Safari/537.36
▼ Response Headers
  Age: 11349
  Alternate-Protocol: 80:quic
  Cache-Control: public, max-age=43200
  Content-Encoding: gzip
  Content-Length: 15836
  Content-Type: text/javascript
  Date: Wed, 11 Jun 2014 07:10:13 GMT
  Expires: Wed, 11 Jun 2014 19:10:13 GMT
  Last-Modified: Thu, 29 May 2014 22:33:33 GMT
  Server: Golfe2
  Vary: Accept-Encoding
  X-Content-Type-Options: nosniff

```

3.1.3 http-Header-Caching

› E-Tags



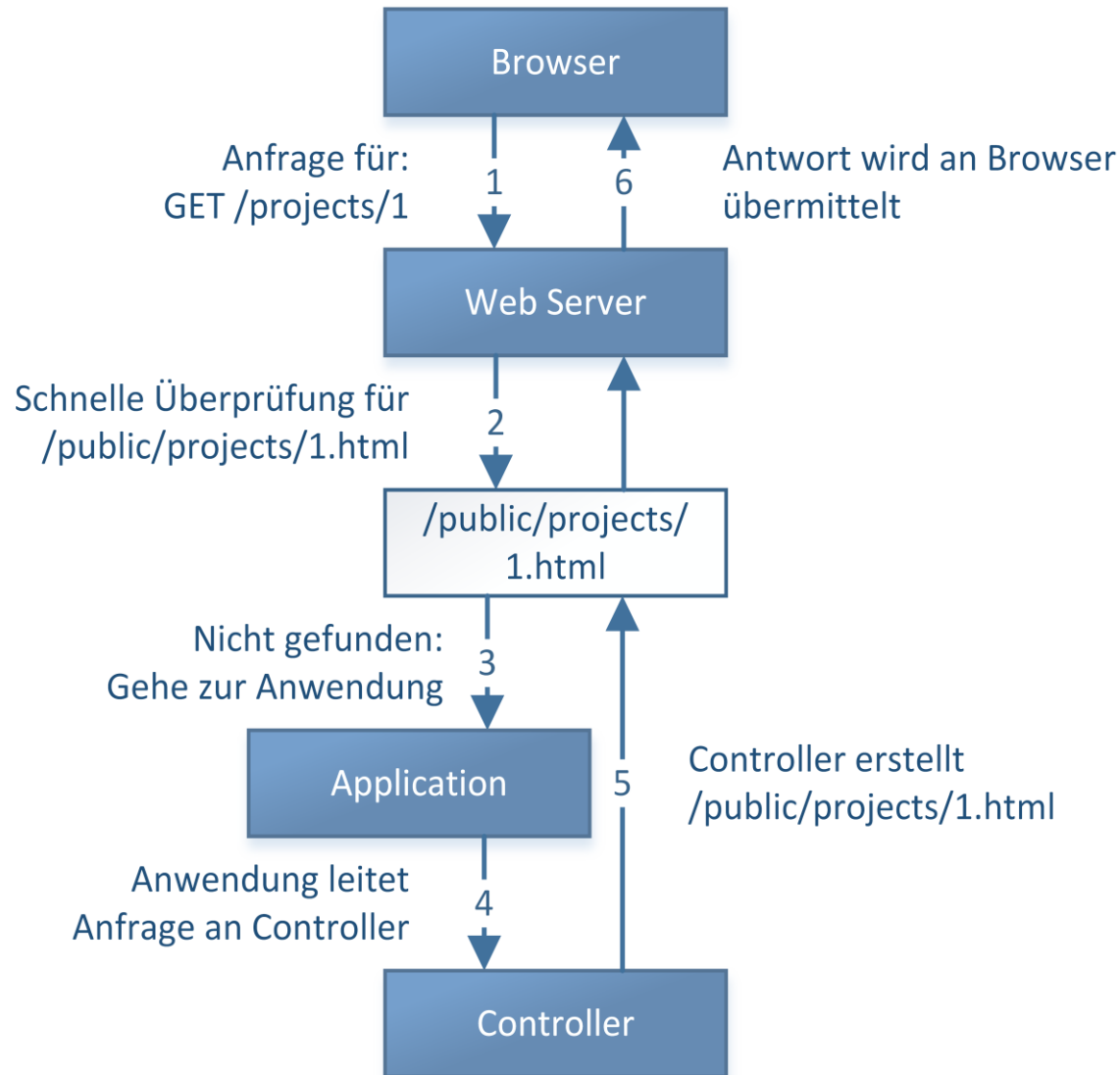
Caching in Rails

3.2.1 Page-Caching

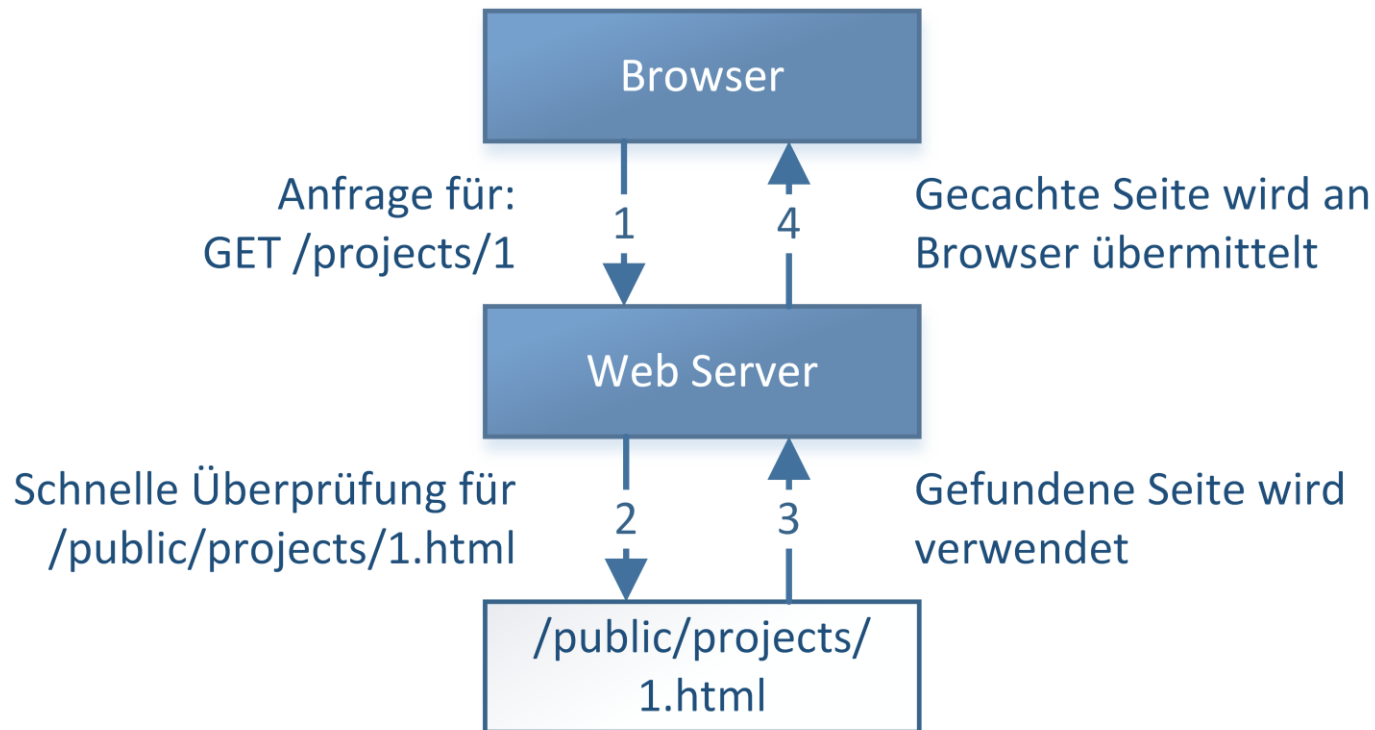
- Simpleste und dennoch effizienteste Form des Rails-Cachings
 - Gecachte Dateien werden vom Webserver mit der gleichen Geschwindigkeit ausgeliefert, in der auch andere statische Ressourcen (z.B. Grafiken oder CSS-Dateien) ausgeliefert werden
- Nach dem ersten Aufruf einer Ressource wird der Output als statische Datei im public/ Ordner der Webanwendung abgelegt
- Bei erneutem Zugriff wird vom Webserver geprüft, ob die angefragte Ressource vorhanden ist
 - Sollte sie vorhanden sein, wird sie direkt vom Webserver ausgeliefert
 - Keine Rails-Interaktion!



3.2.1 Page-Caching



3.2.1 Page-Caching



3.2.1 Page-Caching

› Vorteile

- › Bereits gecachte Daten werden komplett ohne Interaktion mit dem Rails Stack ausgeliefert
- › Performance-Gewinn immer spürbar

› Nachteile

- › Keine Authentifizierung möglich
- › dynamische Webseiten nicht realisierbar
- › GET-Parameter werden nicht beachtet

3.2.1 Page-Caching

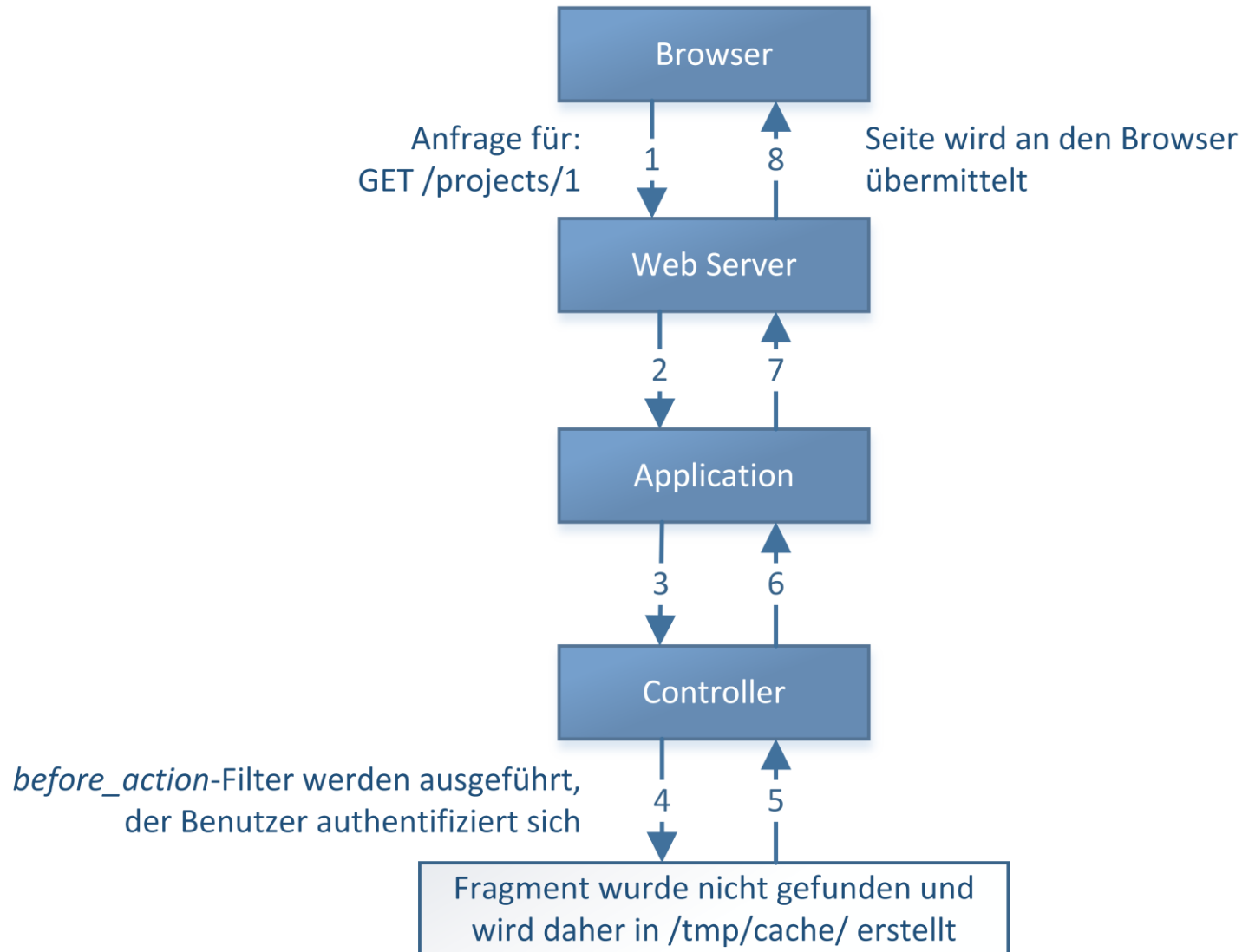
- › Beispielhafte Implementierung im Controller
 - › `cache_page :show, :index`
 - › Die beiden Actions *show* und *index* werden mit der Page-Caching-Methode gecacht
- › Hinweise
 - › Seit Rails 4.0 muss das Gem *actionpack-page_caching* im Gemfile eingebunden werden
 - › Rails' Standard-Webserver WEBrick unterstützt Page-Caching nicht

3.2.2 Action-Caching

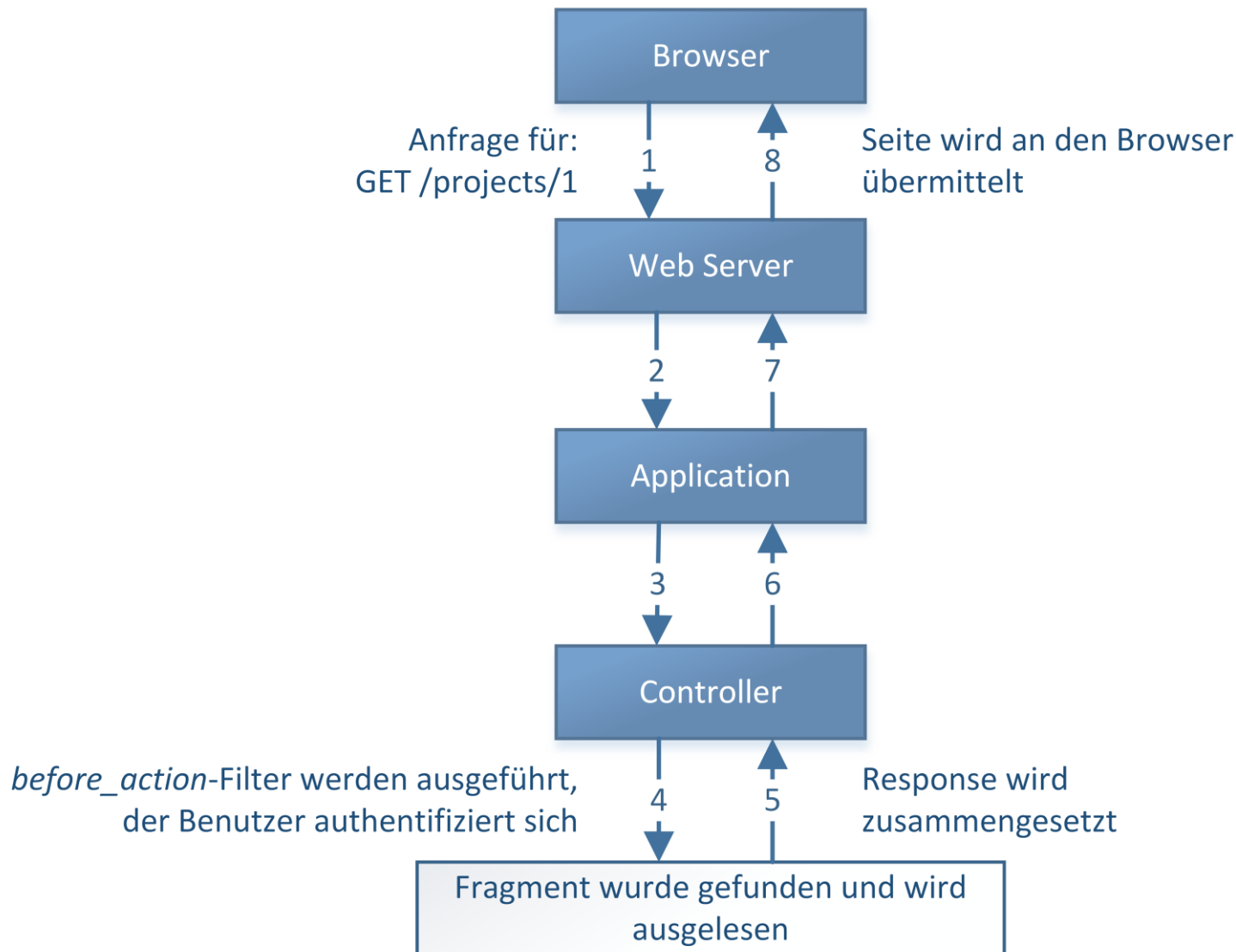
- › Action-Caching ist eine Erweiterung des Page-Cachings
 - › Es werden auch komplette Seiten gecacht
 - › Wird vor allem bei zeitintensiven Actions verwendet
- › Vor dem Ausliefern einer bereits gecachten Ressource
 - › Geht die Anfrage durch den kompletten Rails Stack
 - › *before_action*-Filter werden immer ausgeführt
 - › Entspricht *before_filter* in Rails Versionen < 4.0.0
 - › Rails-Anwendung gibt den Output der angeforderten Action an den Webserver weiter



3.2.2 Action-Caching



3.2.2 Action-Caching



3.2.2 Action-Caching

- › Gecachte Ressourcen werden in tmp/cache abgelegt
 - › Webserver findet die Dateien nicht in public/ und gibt die Anfrage somit an die Rails-Anwendung weiter
- › Beispielhafte Implementierung im Controller

```
cache_action :show
```

 - › Die Action *show* wird mit der Action-Caching-Methode gecacht

3.2.2 Action-Caching

- › Cachen einer Ressource für verschiedene Benutzer
 - › Ähnlich wie bei dem Page-Caching wird standardmäßig nur der erste Output einer Action gecacht

- › Für ein benutzerbezogenes Caching einer Action wird der Parameter *cache_path* verwendet:

```

caches_action :show, :cache_path => (proc do
  project_path(params[:id]) + "/#{current_user.id}"
end)
    
```

Der Pfad setzt sich aus dem Pfad des **URL-Helpers** und der **User-ID** zusammen:

/tmp/cache/views/localhost:3000/projects/1/1

- › Die gecachte Ressource wird für jeden Benutzer einzeln gecacht

3.2.2 Action-Caching

› GET-Parameter ohne weitere Anpassung ebenfalls ignoriert

- › Da wir in Rails Zugriff auf die Parameter haben, können wir sie ebenfalls in den *cache_path* miteinbeziehen

- › Ermöglicht z.B. Seitennummerierung

```
cache_action :show, :cache_path => (proc do
  project_path(params[:id]) +
  "/#{current_user.id}/#{params[:page] || 1}"
end)
```

› Hinweis

- › Seit Rails 4.0 muss das Gem *actionpack-action_caching* im Gemfile eingebunden werden

3.2.3 Fragment-Caching

- › Fragmente einer View werden gecacht, wenn diese lange dauern, um sie zu rendern
- › Flexibelste Caching-Strategie
 - › Die zu cachenden Teile einer View können genau definiert werden
- › Innerhalb der *Views* werden die zu cachenden Bereiche mit der Methode *cache* definiert
 - <% cache do %> ... lange Operation(en) <% end %>*

3.2.3 Fragment-Caching

- Wenn Rails beim Rendern auf einen *cache*-Block trifft, wird in tmp/cache nach dem Fragment gesucht und falls vorhanden, an die passende Stelle eingefügt
- Weitere Aspekte und Konfigurationsmöglichkeiten (z.B. Seitennummerierung, mehrere Versionen einer Seite) können unserer Ausarbeitung entnommen werden

3.2.4 Cache Sweepers

- › Cache Sweeper (zu Deutsch Cache-Kehrmaschine) halten die gecachten Daten auf den aktuellsten Stand
 - › Wenden das Observer-Pattern auf Controller-Instanzen an
 - › Die Logik des Aktualisierens muss vom Entwickler selbst implementiert werden
 - › Ähneln einem *after_action*-Filter
- › Beispielhafte Implementierung im zu observierenden Controller

```
cache_sweeper :tickets_sweeper, :only => [ :create, :update,  
:destroy]
```

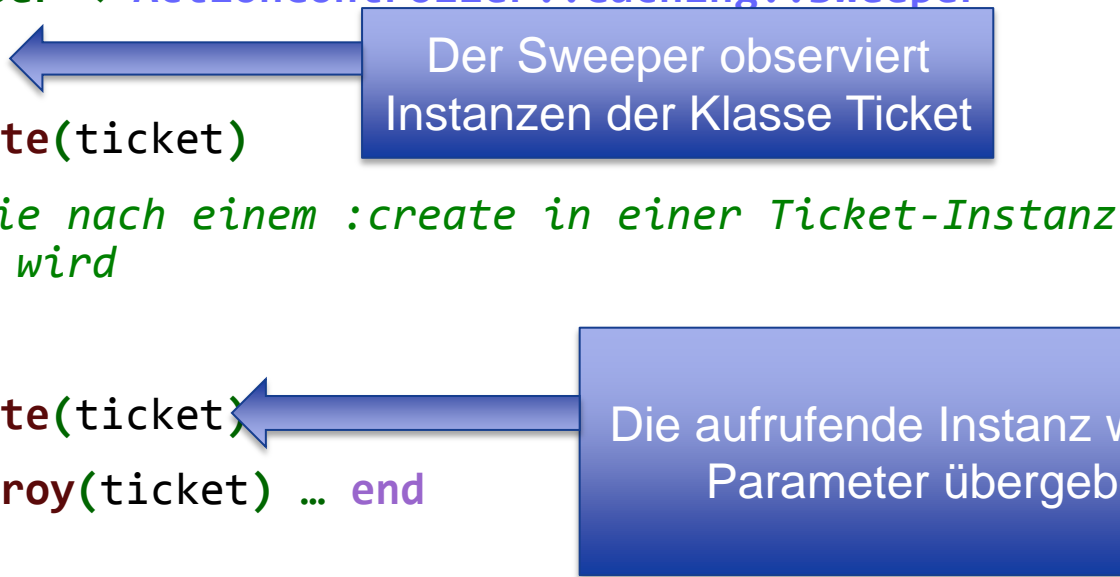


Die zu observierenden Actions

3.2.4 Cache Sweepers

- › Implementierungen eines Sweepers werden in `app/sweepers` abgelegt
 - › Ein Sweeper kann mehrere Controller observieren
 - › Es können mehrere Sweeper angelegt werden
- › Beispielhafte Implementierung eines Sweepers `tickets_sweeper.rb`

```
class TicketsSweeper < ActionController::Caching::Sweeper
  observe Ticket
  def after_create(ticket)
    # Logik, die nach einem :create in einer Ticket-Instanz
    # ausgeführt wird
  end
  def after_update(ticket)
  def after_destroy(ticket) ... end
end
```



Der Sweeper observiert Instanzen der Klasse Ticket

Die aufrufende Instanz wird als Parameter übergeben

3.2.4 Cache Sweepers

› *expire_fragment*-Methode

- › Für Action- und Fragment-Caching

› *expire_page*-Methode

- › Für Page-Caching

› Übergabeparameter der Methoden ist ein konkreter Pfad oder ein regulärer Ausdruck

- › `expire_page(project_path(1))`
- › `expire_fragment(/projects/#{project.id}/.*?/)`

3.2.4 Cache Stores

- Rails bietet die Möglichkeit, den Cache an verschiedenen „Speicherorten“ zu hinterlegen
 - z.B. im Dateisystem, im Arbeitsspeicher, auf einem verteilten Server, ..
 - Nur bei Action- und Fragment-Caching
 - Beim Page-Caching müssen die Daten im Dateisystem liegen, damit der Webserver ohne Interaktion mit Rails darauf zugreifen kann

3.2.4 Cache Stores

› MemoryStore

- › Speichert im Arbeitsspeicher; nicht für Großprojekte geeignet

› FileStore

- › Speichert im Dateisystem; Standardmäßige aktiviert

› MemCacheStore

- › Verwendet memcached

› NullStore

- › Nur für Entwicklungs- und Testzwecke geeignet

Was muss beim Caching beachtet werden?

4.1 Probleme und Risiken?

- Verhältnis zwischen Cache-Hits und –Misses
 - Muss überprüft werden
 - Gibt Aufschluss über Effektivität

- Caching kann Fehler verbergen
 - Architektur bedingte Schwächen
 - Logische Fehler
 - Mangelhaft implementierte Algorithmen

- Korrekte Funktionsweise muss vorher überprüft werden!

4.2 Was spricht gegen den Einsatz von Caching

- › Fehlerhaftes Ablaufdatum
 - › Alte Informationen
 - › Alternative: Fingerprint
- › Analyse wird erschwert
- › Dynamische Ressourcen & persönliche Daten
 - › nicht zwischenspeichern





4.3 Caching Best Practice

› Browser-Caching

› Statische Ressourcen

› Ablaufdatum

› Dynamische Ressourcen

› Fingerprint (Etag)

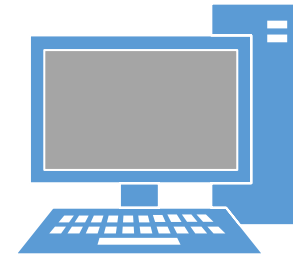
› Proxy-Caching

› Statische Ressourcen

› Kein „query String“ da „?“ teilweise blockiert werden

› Keine Cookies

› Kompression aktivieren



Browser-Caching



Proxy-Caching

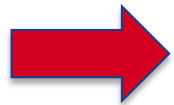


4.4 Anwendungen zur Analyse der Webseite

- › Website oder Plug-In
- › Analyse der Web-Performance
 - › Generiert Verbesserungsvorschläge
- › 60% Verlust durch falsches Bildformat
 - › Hohes Komprimierungspotential
- › Best-Practice Beispiele



YSlow



[Pagespeed Beispiel](#)

5. Fazit

- › Grundsätzlich sollte Caching verwendet werden, aber man sollte sich fragen ...
- › ... was möchte ich konkret cachen/optimieren?
- › ... welche Verfahren benötige ich?
- › ... kann ich diese Verfahren in meiner Anwendung realisieren?
- › ... möchte ich Client- und/oder Serverseitig cachen?

Vielen Dank für ihre Aufmerksamkeit,
noch Fragen?

