



ASS Training

Ruby on Rails

“Web development that doesn’t hurt”

self.inspect

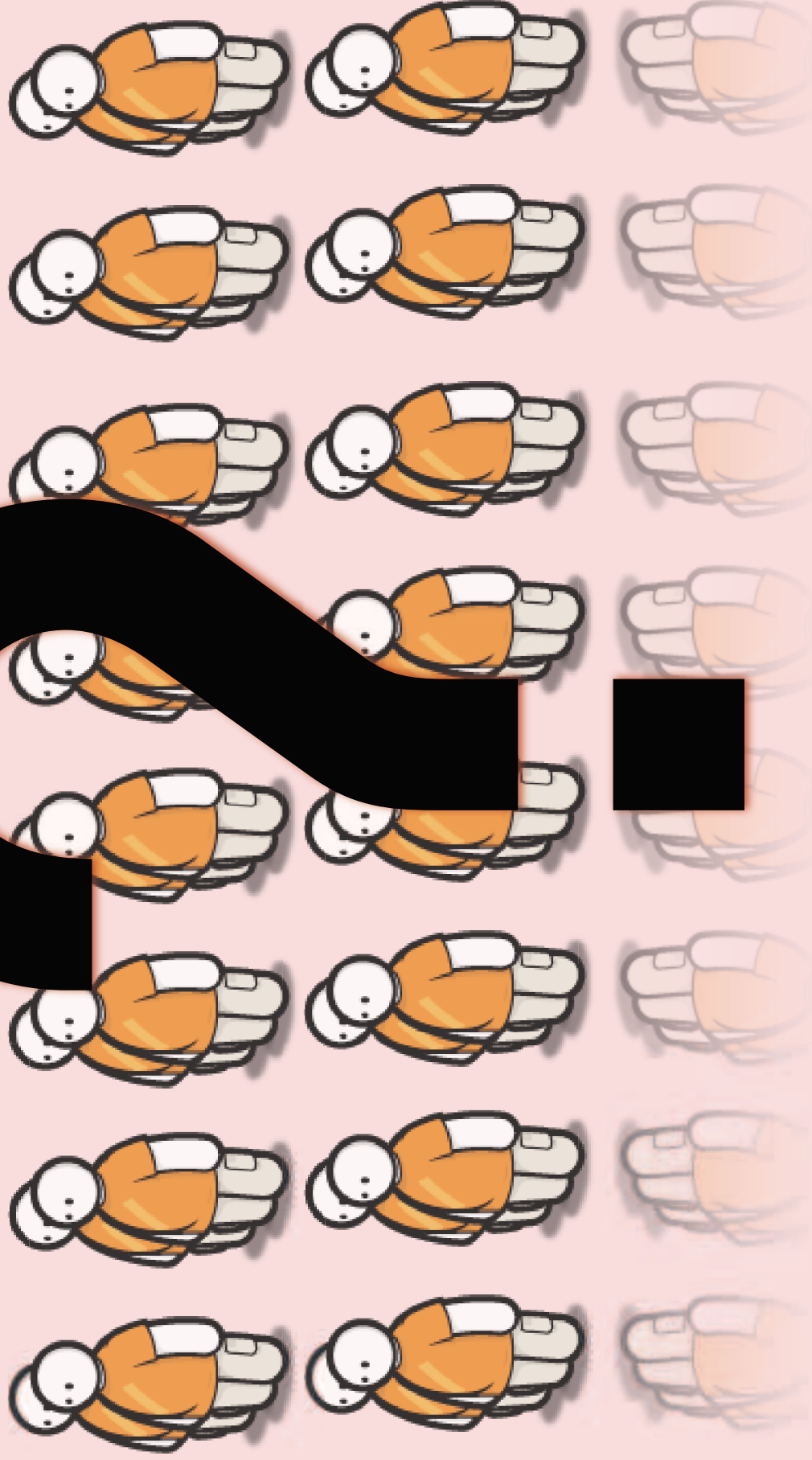


Andreas Bade



Dirk Breuer

self.inspect



`self.inspect`

- Sehr praxisbezogener Einstieg in Ruby, Ruby on Rails und Webentwicklung
- Wichtige theoretische Grundlagen
- Lernen durch Eigenleistung

def training; end

- Erstellen von Ruby Code
- Selbstständiges Erstellen von Rails Applikationen
- Konzepte und Praxis testgetriebener Entwicklung
- Mit der Stateless-Problematik von HTTP umgehen (Sessions/Cookies)

def training; end

- Beantwortung der Fragen:
- Was ist Ruby?
- Was ist Rails?
- Was sind sie Vorteile / Nachteile gegenüber anderen Technologien?

Organisation

- Im Wechsel Vortrag/Demos und Hands-On
- Mittagspause
- kleinere Pausen zwischendurch

Tools

Kommandozeile



iTerm

Text Editor



TextMate

Browser



Firefox

Datenbank



SQLite3

Setup

```
export PATH=/usr/local/bin:$PATH  
export http_proxy=wwwproxy-  
gm.fh-koeln.de:8080
```

3. days do

Tag 1

- Warm-Up & Grundlagen (Ruby)
- Beginn der Applikation (Rails)
- Das M in MVC (ActiveRecord)

Tag 2

- ActiveRecord::Associations
- View und Controller (ActionPack)

Tag 3

- Test-First Development
- User Login



Day One

Get on the Track

Ruby . new

- Dynamische OO-Programmiersprache
- Interpretersprache
- Fokus auf *Simplicity* und *Productivity*
- Ruby is fun!





Objektorientierung

- Alles sind Objekte!

```
42.class # Fixnum
```

- dynamische Typisierung

```
“Hello World”.class # String
```

- Keine “primitiven” Datentypen

```
[‘Andi’, ‘Dirk’].class # Array
```

```
{‘name’ => ‘Programming Ruby’,  
 ‘author’ => ‘Dave Thomas’}.class #
```



Objekte definieren

- class und module
Keywords

```
class Training
  def initialize
    # Constructor
  end
```

- initialize Methode

```
def my_method
  # ... do something
end
end
```

```
training = Training.new
```



Variablen

- **Klassenvariablen (@@class_var)**
- **Instanzvariablen (@instance_var)**
- **lokale Variablen (local_var)**
- **Konstanten (CONSTANT)**



Symbole

- Symbole werden einmal in Speicher abgelegt und bei Gebrauch instanziiert
- Andere Repräsentation einer Zeichenkette
- Spart Speicherplatz
- Schreibweise: `:andl`



Symbole

- Bei der `String` Klasse wird die selbe Zeichenkette zweimal im Speicher abgelegt.

```
>> "ruby".object_id  
=> 3256020  
>> "ruby".object_id  
=> 3253400
```

- Bei der `Symbol` Klasse die selbe Zeichenkette nur einmal.

```
>> :ruby.object_id  
=> 166018  
>> :ruby.object_id  
=> 166018
```



Vererbung

- Einfach Vererbung

```
class Training << ASS
end
```

- Aber: Mixins erlauben
dennoch weitere
Eigenschaften zu
inkludieren

```
class Training
  include Launch
end
```



Schleifen

- while
- until
- for

```
file = File.open('test.txt')
while line = file.gets
  # do something
end

until k < 42
  # do something
end

for i in 0...10 do
  puts i
end
```



Bedingungen

if und unless Ausdrücke

```
if name == "Bob"  
    puts "Hello Bob"  
else  
    puts "Who are you"  
end
```

Ternäre Ausdrücke

```
name == "Bob" ? puts "Hello Bob" : puts "Who are you"
```

case Anweisung

```
case name  
when "Bob"  
    puts "Hello Bob"  
else  
    puts "Who are you"  
end
```



Procs und Blöcke

- Ein Proc-Objekt verpackt einen *Block* Code und kann diesen zu einem späteren Zeitpunkt immer wieder ausführen.

```
my_proc = Proc.new  
  puts 'Ho'  
end
```

- Der Code im *Block* ist wie eine Methode nur ohne an ein Objekt gebunden zu sein.

```
my_proc.call  
my_proc.call  
my_proc.call
```

- An den Code im *Block* können auch Objekte übergeben werden und lokale Variablen innerhalb des *Blocks* gelten nicht außerhalb.

```
Ho!  
Ho!  
Ho!
```

Procs und Blöcke



An Proc-Objekte können auch auch Objekte übergeben werden.

```
today_weather = Proc.new do |temperature|  
  puts "Today we got #{temperature} degree!"  
end
```

```
today_weather.call 42  
today_weather.call 23
```

```
# => Today we got 42 degree!  
# => Today we got 23 degree!
```


Procs und Blöcke



Iteratoren werden in Ruby mit Proc-Objekten realisiert. Für jedes Element in einer Datenstruktur kann eine beliebige Funktion ausgeführt werden.

```
class Array
  def find(find_proc)
    for i in 0...size
      return self[i] if find_proc.call(self[i])
    end
    return nil
  end
end
```

```
[1,3,5,6,7].find Proc.new { |e| e.modulo(2) == 0 } # -> 6
```



Duck Typing

“If it walks like a duck and talks like a duck, it maybe is a duck.”

```
greeting = "Hallo"  
greeting << "Welt" # "HalloWelt"
```

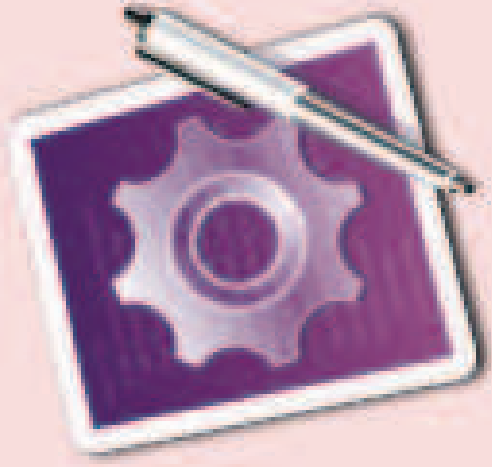
```
greeting = ["Hallo"]  
greeting << "Welt" # ["Hallo", "Welt"]
```


Syntactic Sugar

- Ruby Source Files enden auf `.rb`
- Ruby Programme mit `ruby Dateiname`
starten
- Interactive Ruby Shell (`irb`) für schnelles Experimentieren mit Code



Demo



Hands-On!



Mittagspause

Ruby on Rails

- MVC-Framework für Webapplikationen
- “Convention over Configuration”
- DRY (“Don’t Repeat Yourself!”)
- DSL für Webapplikationen
- Open-Source (MIT-Lizenz)



Architektur

- ORM-Layer (**ActiveRecord**)
- HTTP-Request/Response Unit (**ActionController**)
- (X)HTML-Rendering (**ERb** + **ActionView**)
- Javascript Support (**ActiveSupport**)
- Webservice Integration (**RESTful** + **ActiveResource**)
- Test: :Unit Integration
- Code Generierung (**script/generate**)
- Werkzeuge zur Automatisierung (**rake**)



Bestandteil von Rails:

Build Tool

Tests

Generatoren

In Funktionalität

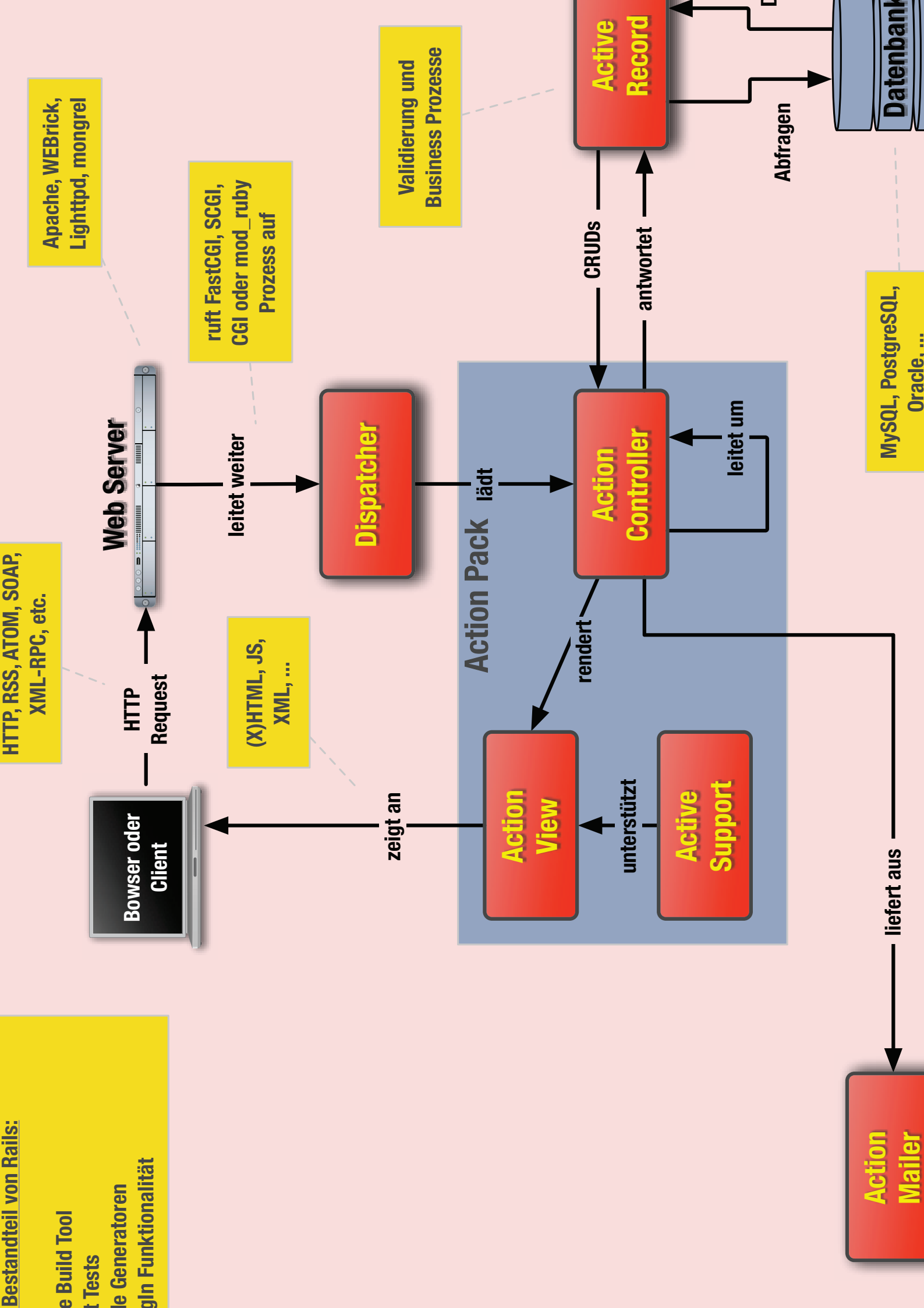
HTTP, RSS, ATOM, SOAP,
XML-RPC, etc.

Apache, WEBrick,
Lighttpd, mongrel

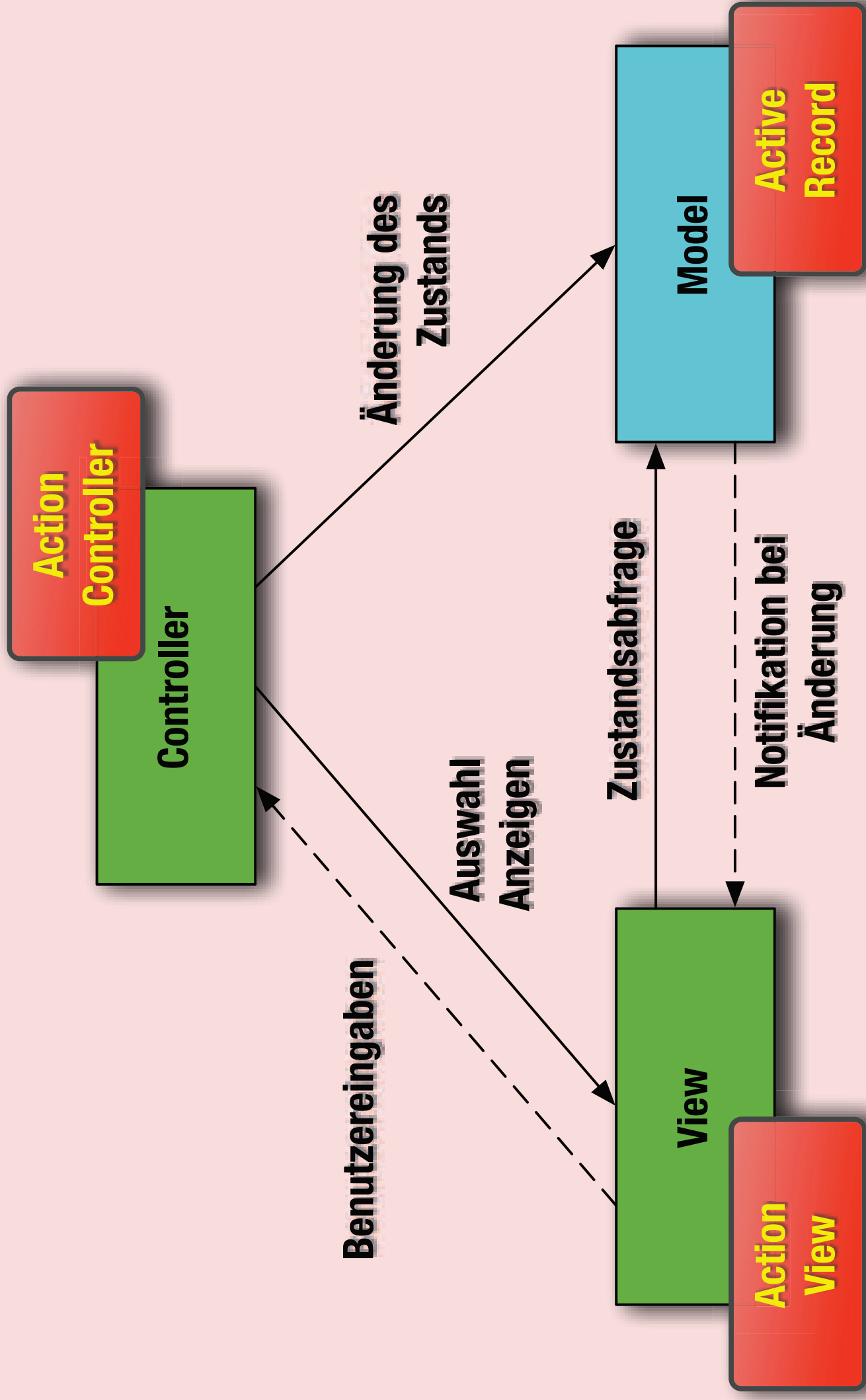
ruft FastCGI, SCGI,
CGI oder mod_ruby
Prozess auf

Validierung und
Business Prozesse

MySQL, PostgreSQL,
Oracle, ...



Rails & MVC



REST

- Representational State Transfer (REST)
- Repräsentationen von Ressourcen werden über URI adressiert
- Keine Methodeninformation im URI, sondern in der HTTP Schnittstelle (HTTP-Verbs)

RESTful

- CRUD wird auf HTTP Verbs abgebildet

Action	HTTP Verbs
Create	POST
Read	GET
Update	PUT
Delete	DELETE



Demo

Min MVC



- Modelle halten die Business Logik der Applikation
- Kapseln den Zugriff auf die Daten
- Entsprechen meist “realen” Entitäten
- Webapplikationen: Meist Datenbanken als Speicher
- Modellierung in Rails durch ActiveRecord

ActiveRecord?



- Martin Fowler (Chief Scientist ThoughtWorks.com)
- Objekte beinhaltet Daten und Verhalten
- Daten sind meist persistent (Datenbank)
- Datenzugriffs- und Domänenlogik vereinen sich in einem Objekt

An object that wraps a row in a database table or view, encapsulates database access, and adds domain logic on that data.“(Martin Fowler)

Fähigkeiten von AR



- Abbilden von Relationen
- Validierung von Werten
- Abbilden der Objekte auf Datenbank (ORM)
- “Single Table Inheritance”
(wird nicht behandelt)

Fähigkeiten von AR

- Dynamische Extraktion der Attribute eines Objekts aus der DDL der entsprechenden Tabelle
- Spaltennamen werden zur Laufzeit als Methode bereit gestellt

Rails und DBMS

- (Fast) jedes relationale Datenbanksystem
- Abstraktion der Datenbankabfragen
 - ➔ Transparent für die Applikation
- Abstraktion der Schemadefinition durch ActiveRecord::Migration

ActiveRecord::Migration

- **Erstellung der Migration**

(script/generate migration)

```
class CreatePeople < ActiveRecord::Migration
```

```
  def self.up
```

```
    create_table :people do |t|
```

```
      t.string :firstname, :lastname
```

```
      t.date :birthdate
```

```
      t.timestamps
```

```
    end
```

- **Zwei Methoden**

(self.up, self.down)

```
    execute „ALTER TABLE ...“
```

```
  end
```

- **Migration durchführen**

(rake db:migrate)

```
  def self.down
```

```
    drop_table :people
```

```
  end
```

ActiveRecord::Base

```
class CreateArticle < ActiveRecord::Migration
```

```
  def self.up
    create_table :people do |t|
      t.string :firstname, :lastname
      t.date :birthdate
```

```
      t.timestamps
```

```
    end
```

```
  def self.down
```

```
    drop_table :people
```

```
CREATE TABLE `people` (
  `id` int(11) NOT NULL auto_increment,
  `firstname` varchar(255) default NULL,
  `lastname` varchar(255) default NULL,
  `birthdate` datetime default NULL,
  PRIMARY KEY (`id`)
ENGINE=InnoDB DEFAULT CHARSET=latin1
```

```
class Person < ActiveRecord::Base
```

- Migration definiert in Ruby die DDL
- rake db:migrate wandelt diese in spezifischen SQL Code um
- Über die Klasse Customer lässt dich die Eigenschaften zugreifen. Dabei repräsentiert:
 - Die Klasse die Tabelle
 - Eine Instanz eine Zeile
 - Die Attribute die jeweiligen Spalte



Demo

MyBlog



- Beispielaufgabe für die Dauer des Workshops



- Einfaches Blogsystem mit Kommentarfunktion und Loginbereich
- Live Suche in den Einträgen

MyBlog



- Was ist Kernstück eines (unseres) Blogs?



Artikel



User



Kommentare

Flowchart

DETAILSEITE ARTIKEL

BILD

BLOG
EINTRAG

KOMMENTAR

...

ZURÜCK

EINEN ANTRAG
ANSEHEN

NEUEN EINTRAG
ANSEHEN

NEUEN ARTIKEL ANLEGEN

TITEL

TEXT

OK

LOGIN

STARTSEITE

NAVI

BLOG
EINTRAG

BLOG
EINTRAG

BLOG
EINTRAG

ZURÜCK

KOMMENTIEREN

NEUES KOMMENTAR ANLEGEN

AUTOR

TITEL

TEXT

OK

anzeigen

suchen

Artikel anzeigen

hinzufügen (nach

kommentieren



Vorgehensweise

1. Rails Projekt anlegen (`rails mybLog`)
2. Anlegen der nötigen Modelle
3. Anlegen der nötigen Controller
4. Anlegen der nötigen Views



Vorgehensweise

- Entweder “Alles auf einmal”
- oder “Iteratives Vorgehen”
- Rails begünstigt iteratives Vorgehen!
- Warum?

Iteratives Vorgehen!?

- Oft Änderungen der Anforderungen während der Entwicklung
- Dadurch:
 - Gefahr der Wiederholung
 - Nicht Berücksichtigung der Änderungen



Lösung

- Ausnutzung der Möglichkeiten der OO-Programmierung
- Rails unterstützt dabei durch
 - MVC-Integration
 - DSL
 - Test-Integration (später)



MyBlog

- Modell "Article" erzeugen
- Controller und View dafür erzeugen mit Scaffold Generator

- Anfangen mit "Artikel"
- Scaffolding um Artikel einzutragen
- weitere Funktionen anschließend



Hands-On!



Validierung

- Validierung von Werten werden im Model festgelegt
- Validierung vor Speicherung des Objekts
- Jedes (Model-)Objekt hat eine 'validate' Methode
- Gescheiterte Validierung = Fehlermeldung (errors Hash wird gefüllt)



Validierung

Mehr Informationen zu
Validatoren im Handout

- `validates_presence_of :name`
- `validates_uniqueness_of :key`
- `validates_format_of :email`
- `validates_length_of :password,
:within => 5..20`



Validierung

Mehr Informationen zu Validatoren im Handout

- Bei eigener `validate` Logik muss das `errors Hash` gefüllt werden, da es die Schnittstelle darstellt um mit den anderen Funktionen von `ActiveRecord` weiterhin arbeiten zu können (Bsp.: `valid?`, `error_messages_for`).
- `errors` beinhaltet immer alle Fehler die beim Versuch der Speicherung aufgetreten sind.

```
class Customer < ActiveRecord::Base
  validate_presence_of :firstname

  def validate
    errors.add(„attribute_name“, „message“
      unless „attribute_name“ ==
        „other_attribute“
    end
  end
end
```


Back to MyBlog



- Validieren der Benutzereingaben
- Sinnvolle Auswahl der Attribute
- Ausgabe bei Fehlermeldungen



Hands-On!



Tag 1 - Fazit

- Ruby als dynamische OO-Programmiersprache
- Rails als agiles MVC-Framework
- Meinungen bisher?



See You Tomorrow

Thursday, 16 Feb. 2008,

09:00 Uhr



Day Two

Feel the Speed

3. days do

Tag 1

- Warm-Up & Grundlagen
- Beginn der Applikation
- Das M in MVC (ActiveRecord)

Tag 2

- ActiveRecord::Associations
- View und Controller (ActionPack)

Tag 3

- Test-First Development
- User Login

Fragen bis hierher?



MyBlog

- Bisher: Artikel anlegen und editieren
- Was fehlt: Artikel sollen kommentiert werden können

||||▶ Weiteres Model/Tabelle in der Datenbank!

||||▶ Problem: Abbilden der Relation in Rails

Working with AR



- Bisher nur Verwendung des generierten Codes (Scaffolding)
- Stärkere Individualisierung der Applikation
- Daher: Auseinandersetzen mit ActiveRecord



Demo

ActiveRecord::Associations



- Arbeit mit mehreren Entitäten
- Abbilden von Relationen zwischen Entitäten
- Relationen auf Objektebene modellieren
- Die Relation muss persistiert werden können



Beziehungstypen

- has_one
- belongs_to
- has_many / has_many :through
- has_and_belongs_to_many

has_one

en sind nach
ntion im Plural
n Keys sind
Konvention der
enname im
lar plus '_id'

- Bildet eine 1:1 Beziehung auf Objektebene ab

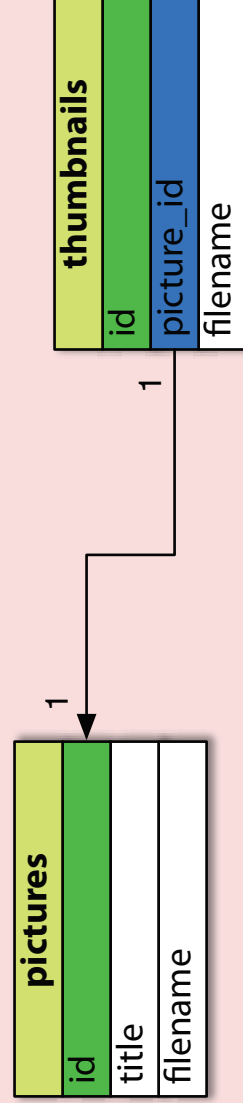
- Erweitert 'Picture' um Methode 'thumbnail'

- Löst die Foreign Key Beziehung in der Datenbank auf

```
class Picture < ActiveRecord::Base
  has_one :thumbnail
```

```
class Thumbnail < ActiveRecord::Base
  belongs_to :picture
```

belongs_to

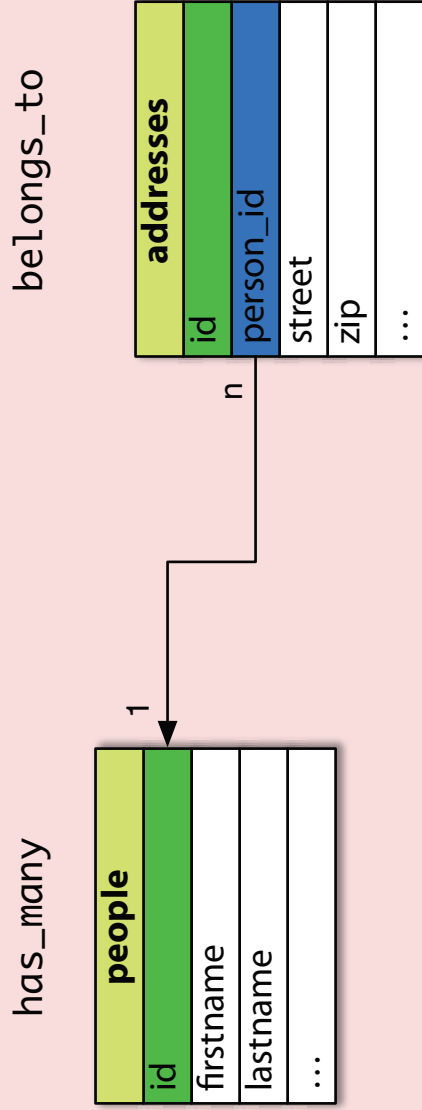


belongs_to

```
class Person < ActiveRecord::Base
  has_many :addresses
```

```
class Address < ActiveRecord::Base
  belongs_to :person
```

- Beschreibt die Tabelle, den Foreign Key in einer oder 1:1 Beziehung hält
- Erweitert 'Address' um Methode 'person'



has_many

- Bildet eine 1:n Beziehung auf Objektebene ab

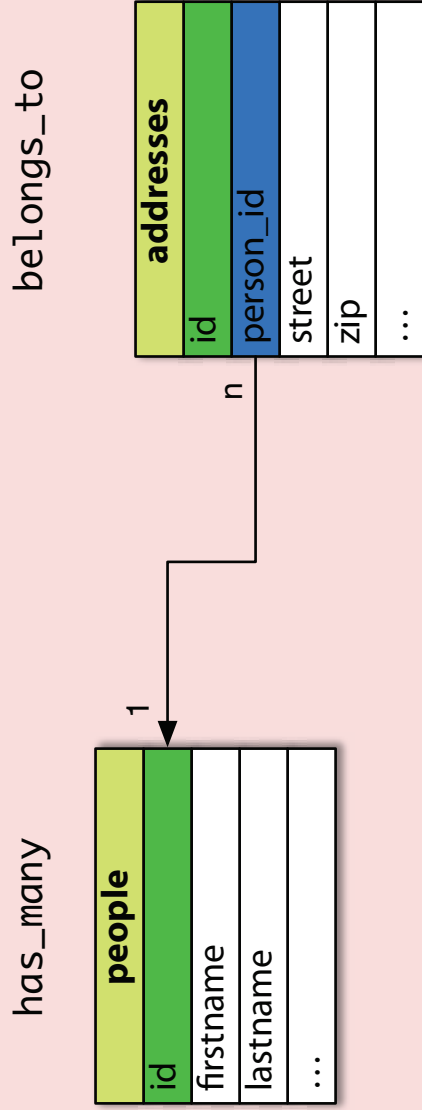
- Erweitert 'Person' um eine Methode 'addresses'

- 'addresses' gibt ein Array aller Adressen zurück.

```
'SELECT * FROM addresses  
WHERE people.id =  
addresses.person_id'
```

```
class Person < ActiveRecord::Base  
  has_many :addresses
```

```
class Address < ActiveRecord::Base  
  belongs_to :person
```



habtm

lisch geordnet
amen der
üpfen Tabellen

```
class Project < ActiveRecord::Base
```

```
  has_and_belongs_to_many :developers
```

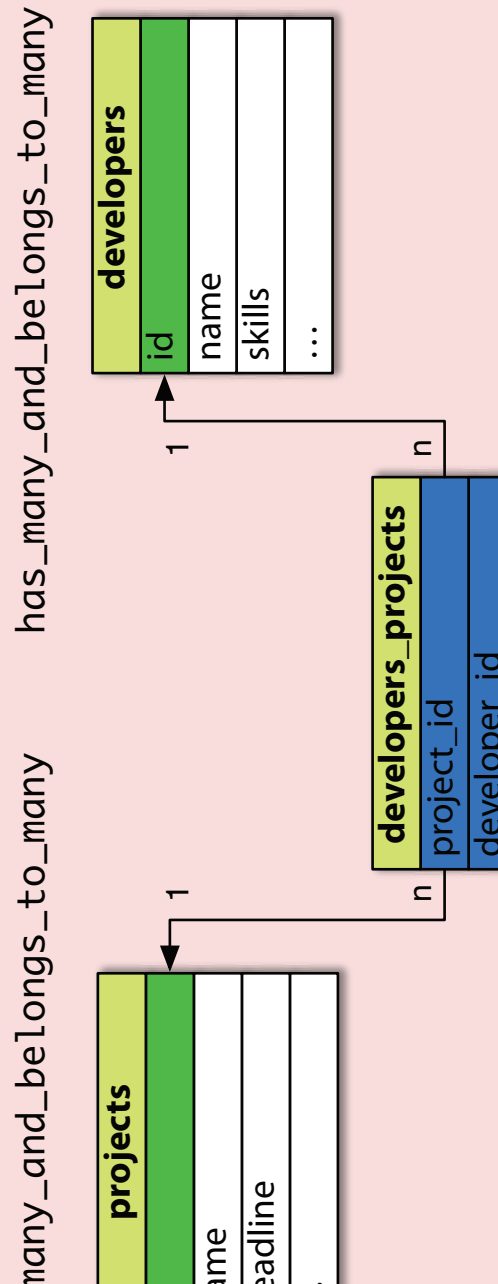
- has_and_belongs_to_r

```
class Developer < ActiveRecord::Base
```

```
  has_and_belongs_to_many :projects
```

- Bildet eine n:m Beziehung auf Objektebene ab

- Zwischentabelle nur! in Datenbank notwendig, KEINE ActiveRecord::Base Klasse



has_many :through

```
class Project < ActiveRecord::Base
```

```
  has_many :developers, :through => :assignments
```

```
  class Assignment < ActiveRecord::Base
```

```
    belongs_to :developer
```

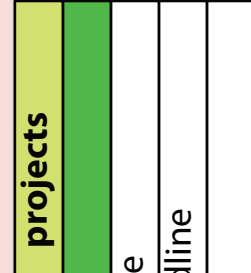
```
    belongs_to :project
```

```
  class Developer < ActiveRecord::Base
```

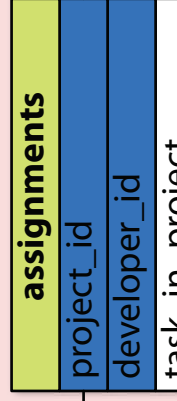
```
    has_many :projects, :through => :assignments
```

```
  end
```

```
  has_many :through
```



belongs_to



- Bildet eine n:m Beziehung auf Objektebene ab
- Zwischentabelle wird durch ActiveRecord::Base Klasse 'assignments' abgebildet
- Zwischentabelle enthält Attribute und bietet die Möglichkeit Validierung durchzuführen



Demo



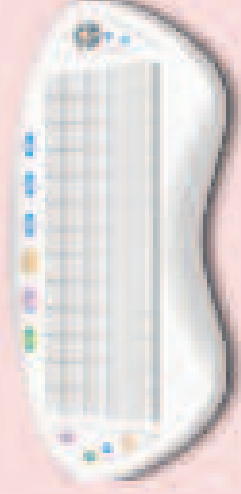
Back to MyBlog

- Modelle haben noch keine Relationen
- Relationen erstellen mit Hilfe von `ActiveRecord::Associations`
- Datenbanktabellen ggf. überarbeiten
(mit Migration)
- Validierung beachten!
(Ein Kommentar darf niemals alleine stehen)

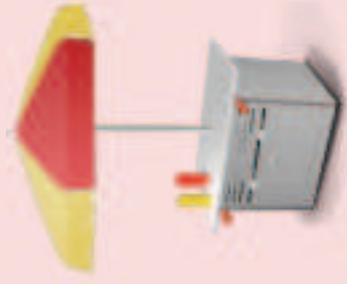
Assoziationen

URL: `arrubyonrails`

Kommentarfunktion
implementieren!

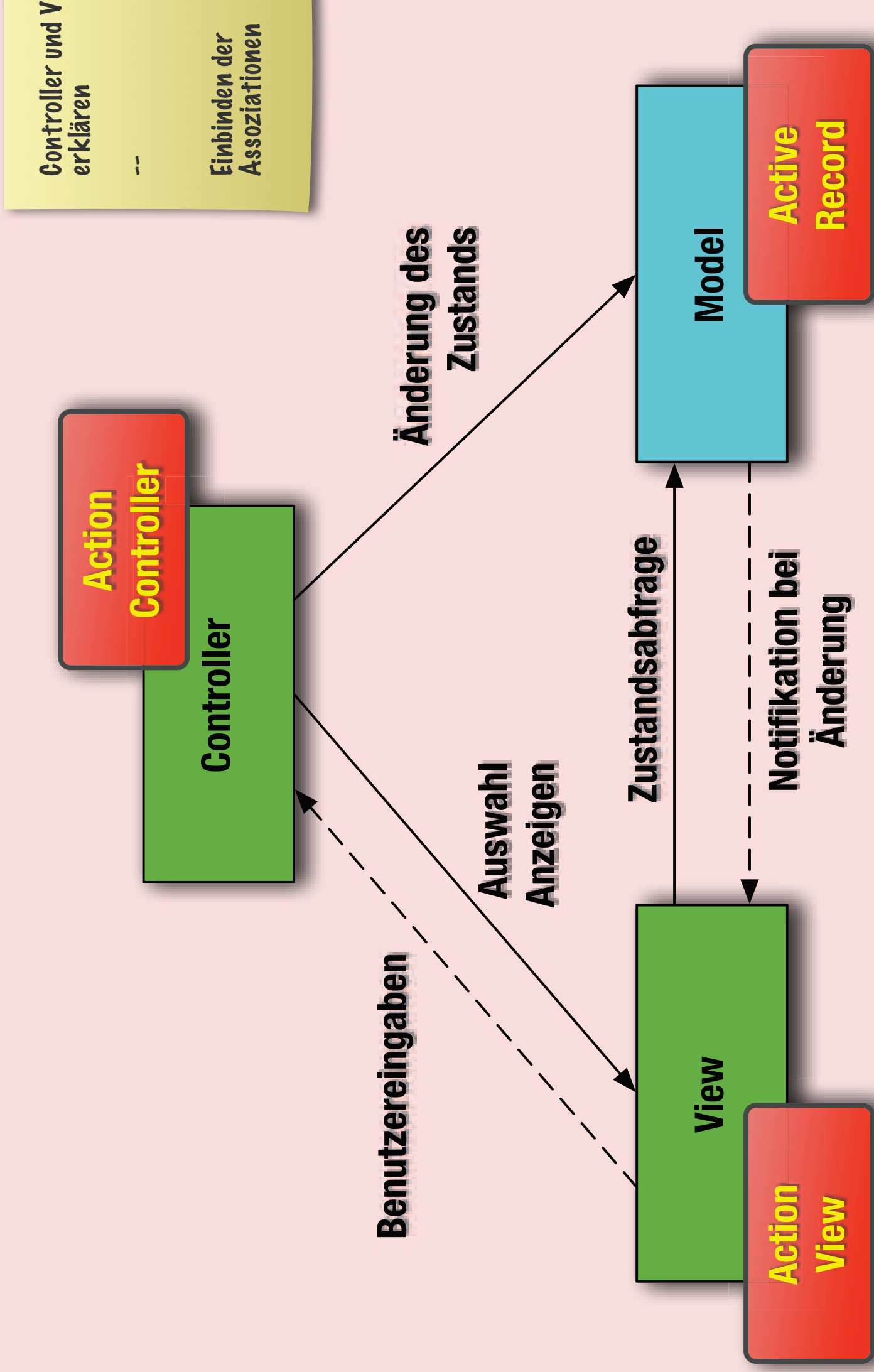


Hands-On!



Mittagspause

Views und Controller





View

- View generiert das User-Interface
- Repräsentation der Daten aus DB
- View-Elemente in Ordner `app/views`
- Layouts für ständigen Inhalt (Bsp: Header, Footer, etc)
`app/views/layouts`
- Jeder Controller hat eigenen View-Template Ordner in
`app/views/controller_name`
- Jede Action hat eigenes View-Template
`app/views/controller_name/action_name.rhtml`





View in Detail

- ERB - Embedded Ruby
- Templatesprache für Plain-Text Dokumente (XHTML, XML...)
- Einbettung durch Inline-Element `<% %>`

`<%- Ruby code -- inline with output %>`


`<%= Ruby expression -- replace with result %>`

Ausgabe in der View

```
<table>
<tbody>
<% @articles.each do |article| %>
<tr>
<% Articles.column_names.each do |column| %>
  <td><%= article.send(column) %></td>
<% end %>
</tr>
<% end %>
</tbody>
</table>
```

ERB Parser

```
<table>
<tbody>
...
<tr>
  <td>ASS Training ist toll</td>
  <td>Dirk Breuer</td>
  ...
</tr>
...
</tbody>
</table>
```

ActionView::Helper

- Entlastung der View von Ruby-Code
- Problem: Ruby-Code in der Präsentationsschicht
- Lösung: Wrapper-Methoden für komplexe HTML-Fragmente

ActionView::Helper

```
<%= link_to "Übersicht", :controller => 'articles', :action => 'list' %>

<%= start_form_tag :action => 'create' %>
  <%= text_field "article", "title" %>
  <%= submit_tag "Create" %>
<%= end_form_tag %>
```

ERB Parser

```
a href="/articles/list">Übersicht</a>

form action="/articles/create" method="post">
  <input type="text" size="30" name="article[title]" id="article_title" />
  <input type="submit" name="Create" value="commit" />
```



Controller

- Alle Controller erben vom ApplicationController (application.rb)

➡ Teilen von übergreifenden Funktionen

- 'Moderator' der Applikation (app/controllers)
- HTTP-Request/Response Handler
- Jeder Controller besitzt ein Request/Response-Objekt



Controller



- Eigentliche Operationen werden von den Actions durchgeführt
- Actions sind öffentliche Methoden des Controllers
- Actions können rendern oder an andere Actions weiterleiten
- Verfügbarkeit der Actions durch Routing

Controller im Detail



- Action ruft automatisch gleichnamiges View-Template auf
- **Render** (render :template => 'show')
- **Redirect** (redirect_to :action => 'index')
- **Zugriff auf Request Parameter und Session**
params[:key], session[:key]

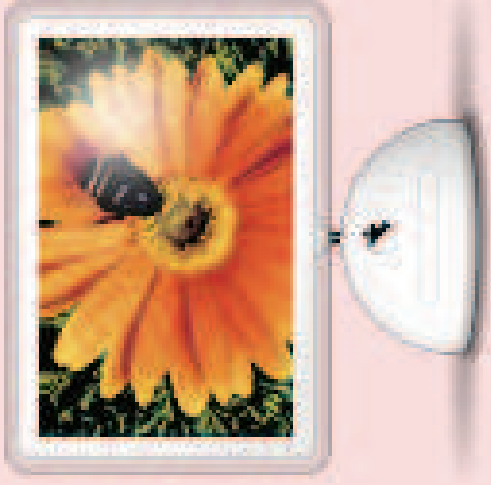
URL und Parameter

- <http://localhost:3000/articles/1/edit>
- rake routes
- articles -> controller (params[:controller])
- 1 -> params[:id]
- edit -> action (params[:action])

Back to MyBlog



- Überarbeiten von Controller und Views
- Create Read Update Delete - Methoden
- Eingaben und Ausgaben berücksichtigen
- Artikel sollen kommentiert werden können



Hands-On!



Partials

- Partials sind Teilstücke der View
- Seperate Dateien (`_form.rhtml`)
- Wiederverwendbarkeit von Code
- Modularisierung der View



Partial Aufruf

- **Verschiedene Möglichkeiten**

- ▶ `render :partial=>'person',
 :locals => {:name => "andi"}`
- Rendert Partial 'person' mit lokaler Variable
- ▶ `render :partial=>'comment',
 :collection => @articles.comments`
- Rendert Partial 'comment' mit gleichnamiger lokaler Variable
 'comment' für alle Objekte in der Collection

Back to MyBlog



- Liste der Kommentare als Partial implementieren
- Sinnvolle Partial-Elemente erzeugen
- DRY erleben!



Hands-On!



Tag 2 - Fazit

- Integration des MVC-Konzepts in Rails
- Model (ActiveRecord)
- View (ActionView)
- Controller (ActionController)
- Meinungen bisher?

RURUG Köln

- Jeden zweiten Donnerstag im Monat von 19:30 - 21:30 Uhr
- Treffpunkt: Chaos Computer Club Cologne
- <http://rurug.de>



See You Tomorrow

Friday, 15 Feb. 2008,

09:00 Uhr



Day Three

Final round

3. days do

Tag 1

- Warm-Up & Grundlagen
- Beginn der Applikation
- Das M in MVC (ActiveRecord)

Tag 2

- ActiveRecord::Associations
- View und Controller (ActionPack)

Tag 3


- Test-First Development
- User Login

Fragen bis hierher?



MyBlog

- Bisher: Artikel anlegen und editieren, Kommentare hinzufügen
- Was fehlt: Testing
- Was fehlt: Login
- ||||▶ Rails Test Environment kennen lernen
- ||||▶ Login Implementieren (test-driven)



Testen?!

- „Aber ich teste doch! Ich mach das dann immer im Browser.“
- Nur automatisierte Tests gelten als Test
- Bei TDD: Code der nicht getestet wird, existiert nicht!

TDD



- Test Case formulieren vor der eigentlichen Implementierung
- Funktionalität implementieren bis Test durchläuft
- Die Tests sind Spezifikation der Applikation zu verstehen

rake test

- Führt alle vorhandenen Test-Fälle im Ordner `test` aus
- 3 verschiedene Arten von Tests
 - Unit (Testen der Modelle)
 - Functional (Testen der Controller)
 - Integration (Übergreifende Testfälle)



Unit Tests

- Allgemein: Testen einzelner Module in einer Software
- Bei Rails: Testen der ActiveRecord Klassen
- Testen von Funktionen wie
 - Validatoren (vor allem Selbstgeschriebene)
 - eigene Methoden
 - Abhängigkeiten

Fixtures

- Testdaten
- Werden für jeden Testfall neu in die Datenbank gespielt
- Fixturedaten werden über `users(:dirk)` im `TestCase` bereitgestellt

Fixtures

- Definition von Objekten in YAML (Serialisierung).
Hierarchien werden durch Einrückungen abgebildet (je zwei Leerzeichen KEINE Tabs)
- Assoziationen zwischen Modellklassen lassen sich ebenfalls abbilden
- Eine Datei pro Tabelle

```
# articles.yml
first:
  title: Mein Artikel
  content: Wichtiger Inhalt zum Artikel
  author: dirk

# users.yml
dirk:
  email: dirk.breuer@myblog.com
  first_name: Dirk
  last_name: Breuer
  password: 5eb942810a75ebc850972a89285d570d48
  created_at: <%= Time.parse("2006-04-12
16:00:12").to_s(:db) %>
```

Aufbau Test Case

Erbt von `Test::Unit::TestCase`

Fixtures (Testdaten)

Setup Methode (wird vor jeder Testmethode ausgeführt)

Test Methoden (beginnen immer mit `test`)

Assertions (Aussagen bestätigen)

```
require File.dirname(__FILE__) + '/../test_helper'

class ArticleTest < Test::Unit::TestCase
  fixtures :articles

  def setup
    # Do some initial stuff
  end

  def test_has_valid_title
    article = Article.new(:title => ' ')
    assert !article.valid?

    article.title = "<script>javascript:alert('Hallo');</script>"
    assert !article.valid?

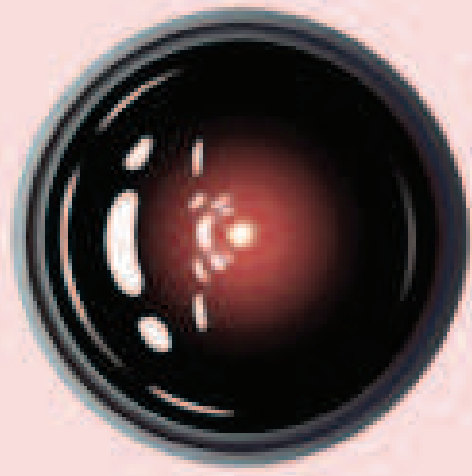
    article.title = "Valid Title"
    article.author = users(:bob)
    assert_valid article
    assert article.save
  end
end
```



Demo

Back to MyBlog

- Gehört ein Kommentar immer zu einem Artikel?
- Kann ein Artikel mehrere Kommentare haben?
- rake test:units



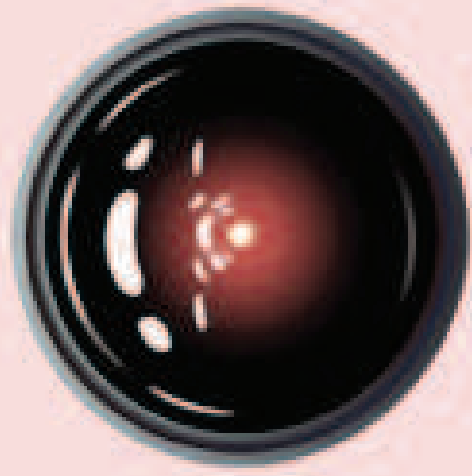
Hands-On!

Functional Tests

- Testen der Funktionen die einem User (oder anderem System) angeboten werden
- Testen einzelner Controller
- Testen von Funktionen wie
 - Werden die richtigen Variablen gesetzt?
 - Wird das richtige Suchergebnis geliefert?
 - Stimmt die Sortierung?
 - Ist der Zugang tatsächlich beschränkt?



Demo



Hands-On!

Integration Tests

- Testen von Workflows innerhalb der Applikation
- Testen mehrerer Controller im Zusammenspiel
- Testen von Funktionen wie:
 - Klappt der Request von A nach B?
 - Funktioniert der Login, anlegen eines Artikels und anschließend das anlegen von Kommentaren in Reihe?



Mittagspause



User Login

- Nicht jeder soll neue Artikel anlegen oder bestehende editieren können
- Loginbereich
- Bestimmte Aktionen nur nach Login zulassen



User Login

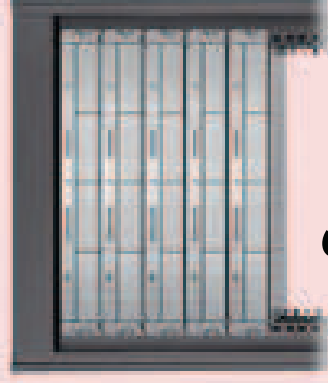
- Anwendung des bisher Gelernten
- HTTP-Stateless-Problematik



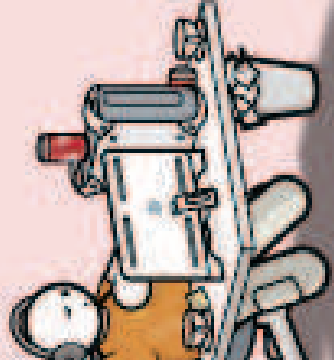
Sessions

- Problem: HTTP ist Stateless Protokoll
- Lösung: Pseudostate einführen in Form von Session
- Session ist (temporär) persistent auf dem Server hinterlegt
- Session-Key wird i.d.R. im Cookie hinterlegt

Sessions



Server



User



Session Key/Value Paar in
Cookie (der Domain) schreiben

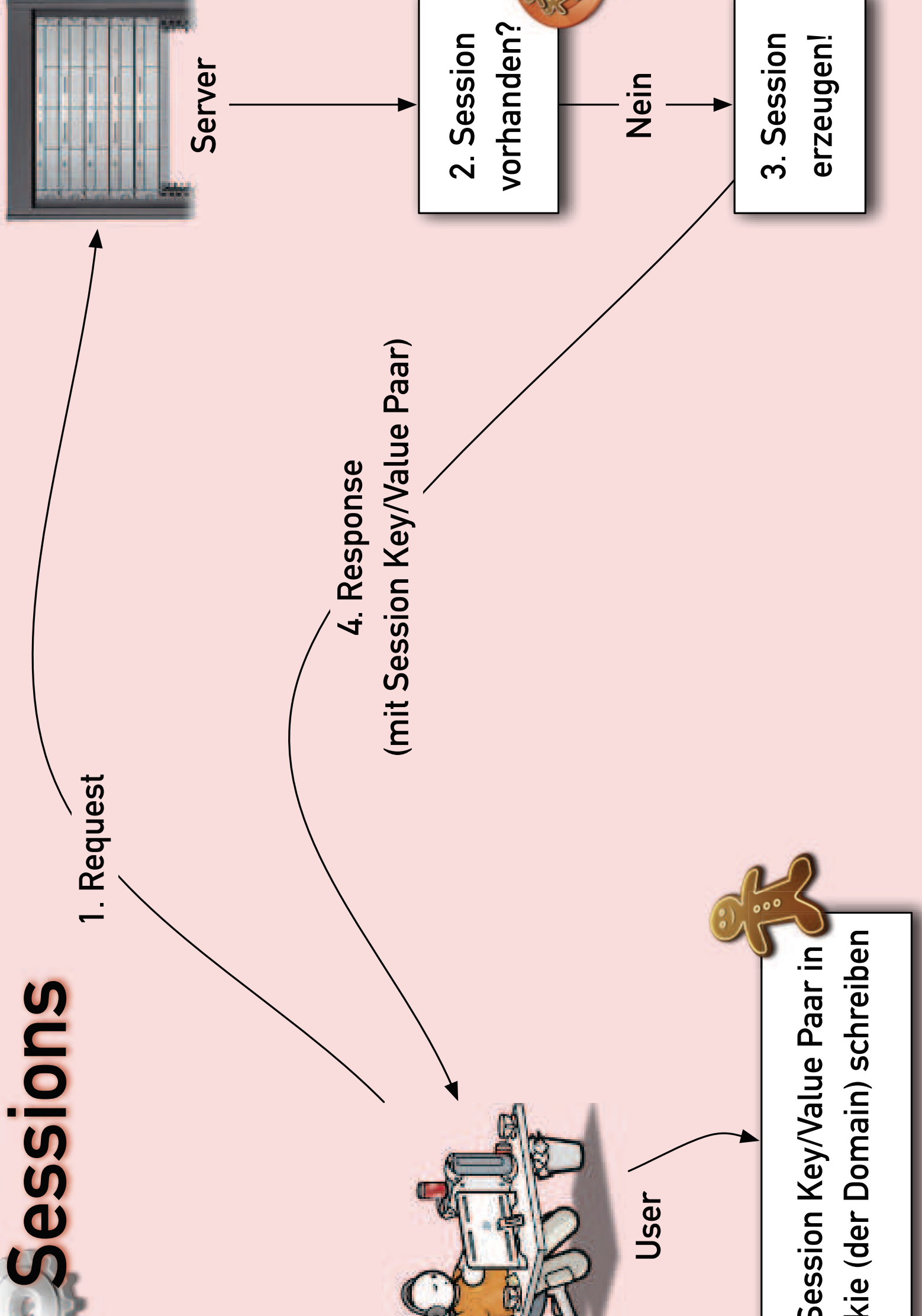
1. Request

4. Response
(mit Session Key/Value Paar)

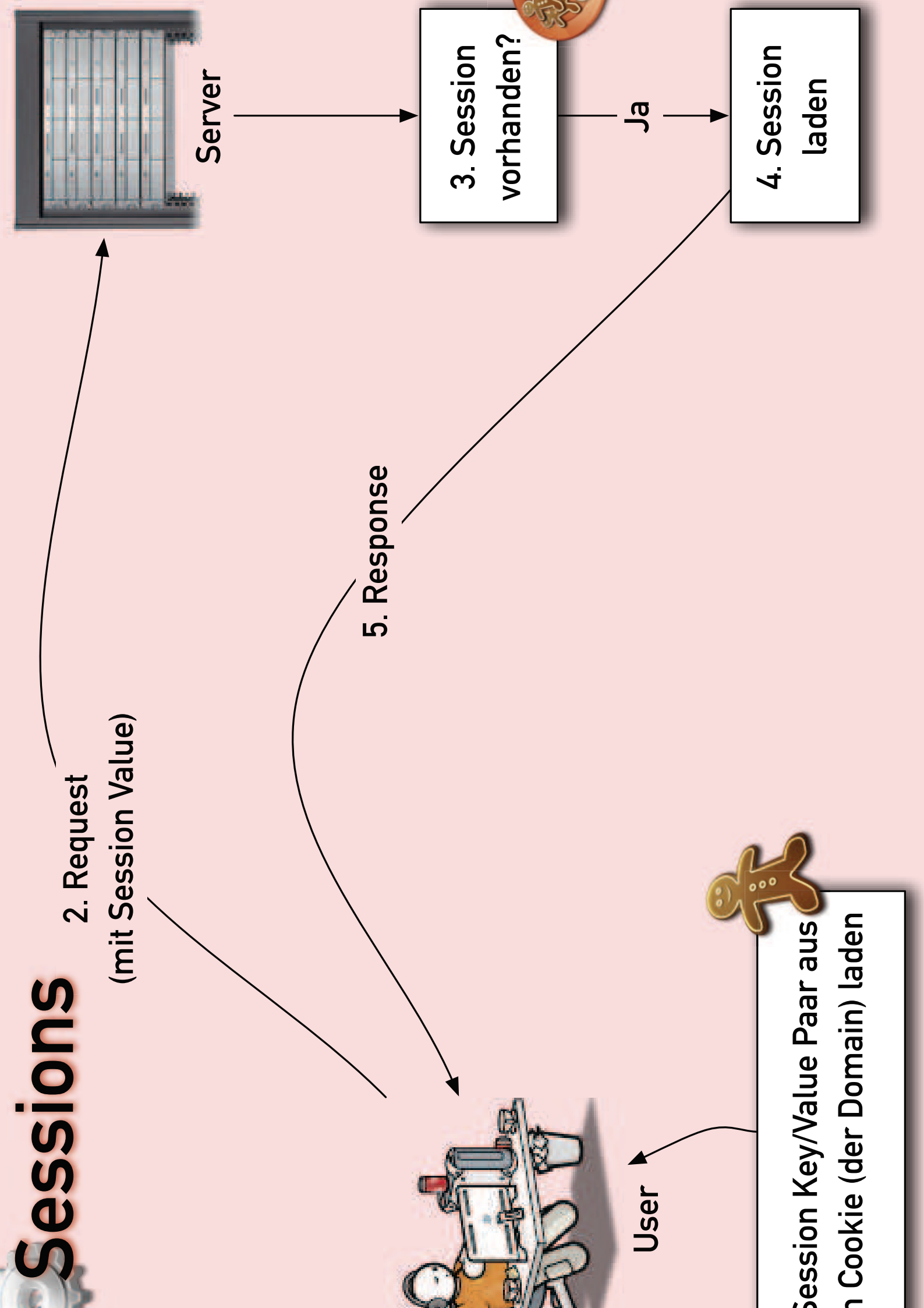
2. Session
vorhanden?

Nein

3. Session
erzeugen!



Sessions





Demo



Sessions

- In Rails ist die Session als Hash realisiert, das serialisiert auf einem persistenten Speicher hinterlegt wird, z.B.:
 - Filesystem
 - Datenbank
 - Speicher
 - Memcached
 - Cookie (HÄ??)



Sessions

- Hier verwenden wir nicht die Standardlösung (Cookie-Based), sondern Filesystem
- Zugriff auf das Session Hash über die Methode `session`
- Prinzipiell kann alles in der Session gespeichert werden, aber abwägen was tatsächlich gespeichert werden soll



Filter Chain

- Filter fügen Querschnittsfunktionalität in die Applikation ein (Bsp: Authentisierung)
- Filter können von verschiedenen Controllern gemeinsam benutzt werden
- Hier: `before_filter` (andere werden nicht behandelt)



Filter Chain

- Definition der Filter im Controller
- Definition der Methode die beim Filter ausgeführt, typischerweise im ApplicationController

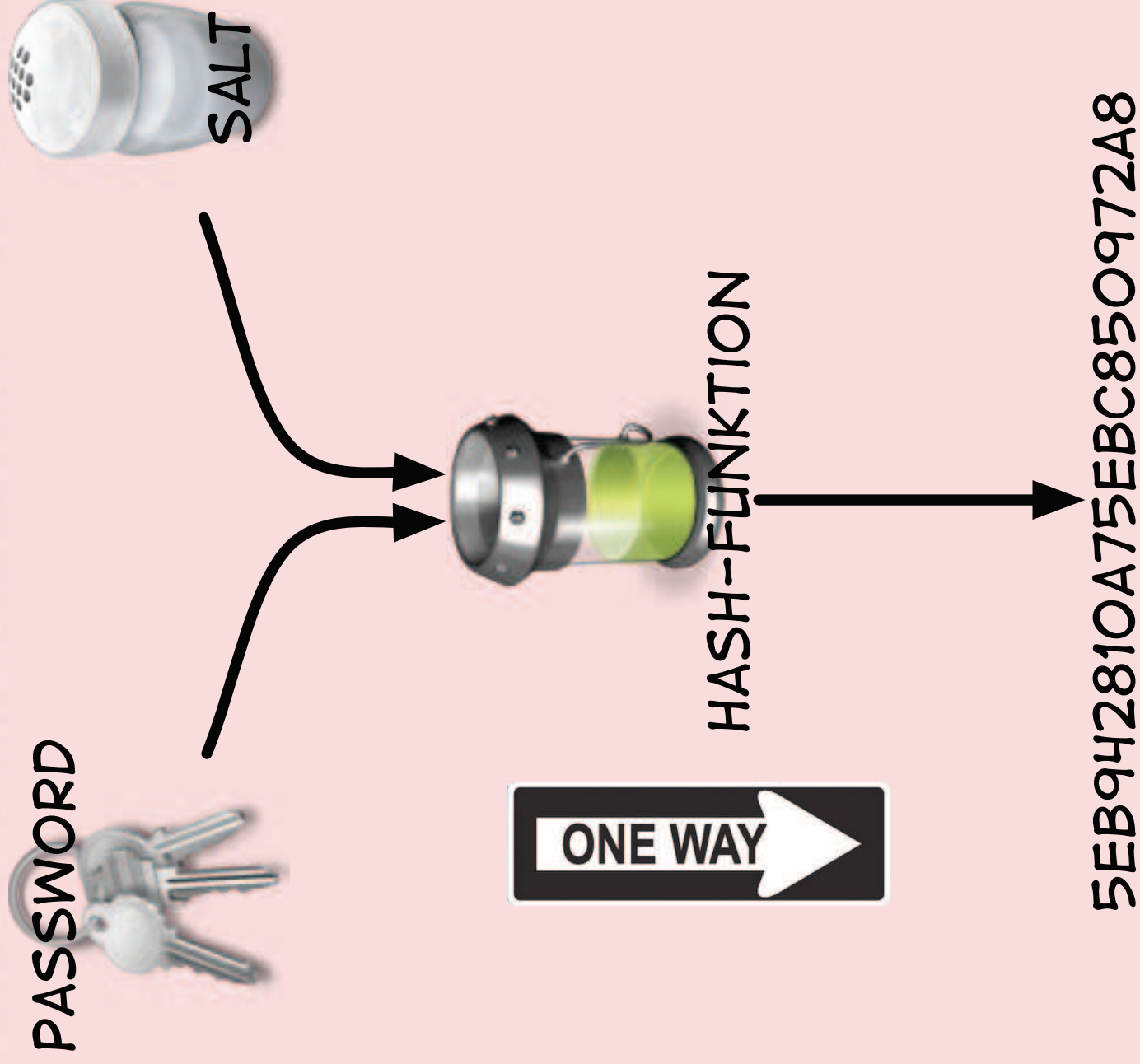
```
articlesController < ApplicationController  
  def filter :authenticated,  
    :only => ['new', 'create',  
              'update', 'edit']
```

```
class ApplicationController < ActionController::Base  
  def authenticated  
    if session[:user]  
      true  
    else  
      redirect_to :controller => 'users', :action => '  
    end  
  end  
end
```

Integration des Logins

- Wichtige Funktion! Niemand ohne Berechtigung darf geschützte Funktionen ausführen.
- Sicherstellung durch Testen

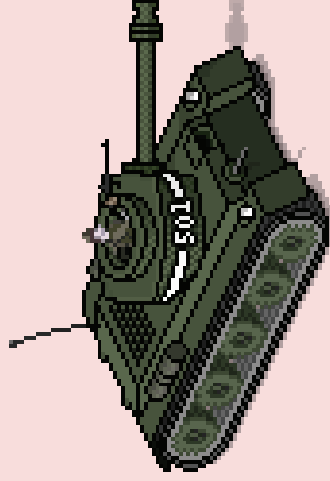
Exkurs: Hash-Funktion





Back to MyBlog

- User mit Login erstellen
- Test-Case für Login im Blog erstellen
- Fixtures erstellen
- Testfall schreiben
- Test laufen lassen



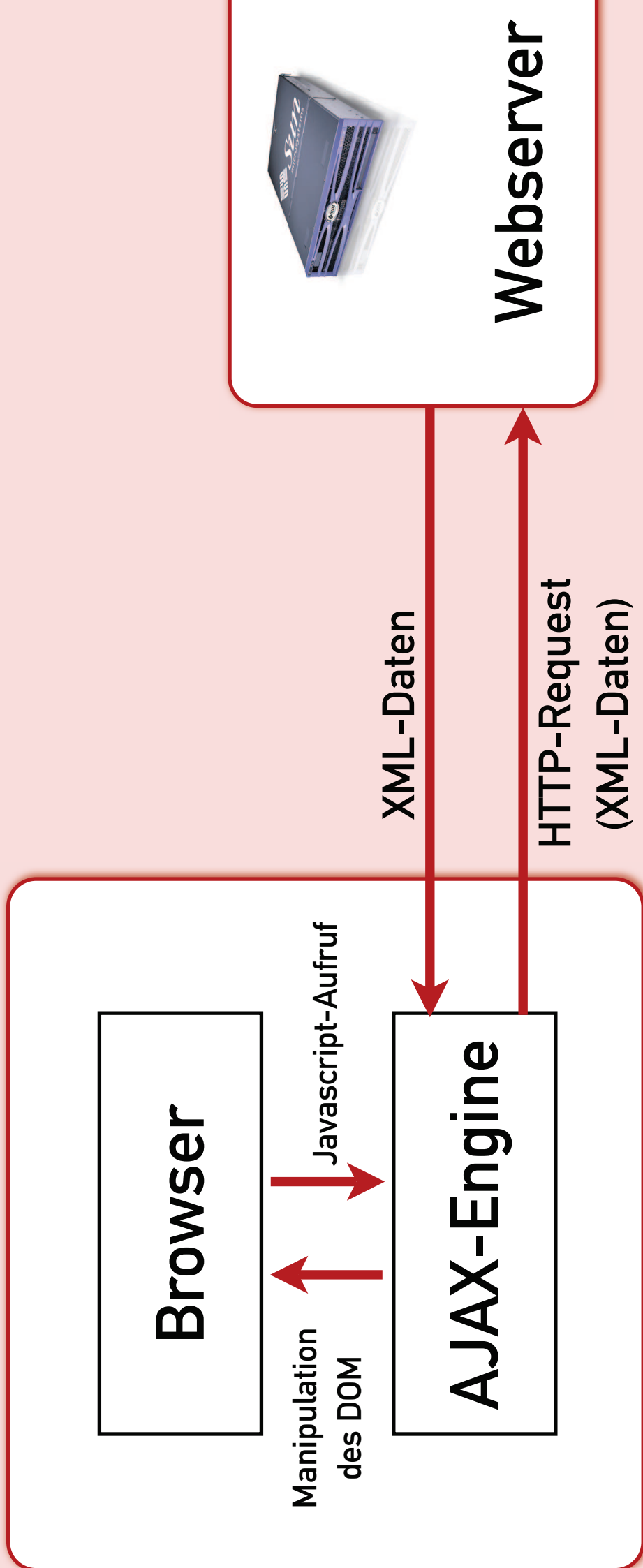
Hands-On!



AJAX

- Akronym für Asynchronous JavaScript and XML
- Der Begriff ist neu und voll im Trend!
- Das Konzept ist alt! (XmlHttpRequest)
- Asynchrone Datenübertragung zwischen Browser und Server
- Kein Reload der Seite bei Aktualisierung
- Look&Feel von Desktop-Applikationen

AJAX-Modell





Vorteile

- Kein Plugin erforderlich
- Völlig neue Interaktionsmöglichkeiten
- Live-Suche, Auto-Completion, etc.
- Viele kostenlose API's und Frameworks
([Script.aculo.us](https://script.aculo.us), Prototype...)



Nachteile

- Polling-Problematik
- Zurück-Button
- Lesezeichen
- Feedback



AJAX in Rails

- **Vollständige Integration von Prototype und Script.aculo.us**
- **Wrapper-Methoden ermöglichen einfachen Zugriff auf AJAX-Funktionen**
- `link_to_remote, form_remote_for, observe_field...`



AJAX in Action

- Live-Suche
- Suchergebnisse werden bei der Eingabe des Suchworts angezeigt

JavaScript Generator Templates (rjs)

- Veränderung mehrerer DOM-Elemente
- Nutzt XMLHttpRequest
- Bestimmt WIE Templates modifiziert werden

render :update

```
render :update do |page|
  page.insert_html :before, 'element-id', :partial => 'cron'
  page.replace_html 'element-id', :partial => 'search',
    :locals => { :articles => @articles }
  page.remove 'element-id'
  page.show 'element-id'
  page.hide 'element-id'
end
```




Back to MyBlog

- „Ajaxified“ Blog mit Live-Suche
- Suche für Artikel
- Welche Suchkriterien sind relevant?



Hands-On!



End of Days

Projektabschluss

- Hauptanforderungen erfüllt!
- Nach der Pflicht kommt die Kür
- RSS-Feed für den Blog
- Bilder-Upload für Artikel
- Live-Suche mit AJAX
- ...

Zusammenfassung

- Einstieg in das Rails Framework
- Agile Methode zur Entwicklung von Web-Applikationen
- Rails unterstützt das MVC-Model
- Entlastung des Entwicklers bei Konfiguration
- Konzentration auf die wesentlichen Dinge
- Knowledge-Pack für alle!

Knowledge-Pack

- <http://www.medieninformatik.fh-koeln.de>
- Folien
- Hand-Outs
- Kompletter Source Code

Knowledge-Pack

- Literatur
- The Pragmatic Programmer - Dave Thomas, Andrew Hunt (Addison-Wesley ISBN: 0-201-61622-X)
- Programming Ruby - Dave Thomas, Andrew Hunt (The Pragmatic Programmers ISBN: 0-974-51405-5)
- Ruby for Rails - Dave A. Black (Manning, ISBN: 1-932394-69-9)
- Agile Web Development with Rails (2nd Ed.) - Dave Thomas, David Heinerm
Hansson (The Pragmatic Programmers, ISBN: 0-9776166-3-0)
- Rails Recipes - Chad Fowler (The Pragmatic Programmers, ISBN:
0-9776166-0-6)

Web Links



- <http://www.ruby-lang.org>
- <http://rubyonrails.com>
- <http://script.aculo.us>
- <http://www.prototypejs.org>
- <http://de.selfhtml.org/>
- <http://groups.google.com/group/rubyonrails-talk>
- <http://subversion.tigris.org/>
- <http://git.or.cz>
- <http://rubyquiz.com/>
- <http://rubyforge.org>
- <http://errtheblog.com>
- <http://weblog.jamisbuck.org>
- <http://drnicwilliams.com>
- <http://railscasts.com/>
- <http://nubyonrails.com>
- <http://peepcode.com/>
- <http://therailsway.com/>
- <http://code.google.com/p/blueprintcss/>
- <http://csszengarden.com/>
- <http://studios.thoughtworks.com>



Tools

Firefox + Firebug

- Windows + e + cygwin

Mac OS X + TextMate

- Linux + JEdit

Commandline

- IDEs

RubyGems

- RadRails

MySQL / SQLite3

- NetBeans 6

Subversion / Git

- 3rdRails

Trac / Mingle / Basecamp

- Aptana



About Us

- <http://railsbros.de>
- andi.bade@gmail.com
- dirk.breuer@gmail.com
- <http://mediaventures.de>
- <http://pkw.de>



One Question...

- Eure Meinung ist uns wichtig!
- Was hat euch gefallen?
- Was hat euch nicht gefallen?
- Was kann man besser machen?



**Macht's gut und Danke
für den Fisch**