RUBY ON RAILS

Markus Knofe

Gliederung

a) Was ist Rails
b) MVC in Rails
c) Rails praktisch
d) Fazit

Rails ist **innovativ !**

Rails ist **innovativ !**

Rails ist **elegant !**

Rails ist **innovativ !**

Rails ist **elegant !**

Rails ist **schnell !**

Rails ist **innovativ !**

Rails ist **elegant !**

Rails ist **schnell !**

Rails ist **Produktivität !**

Rails ist **innovativ !**

Rails ist **elegant !**

Rails ist **schnell !**

Rails ist **Produktivität !**

Rails ist **die Freude am Programmieren !**

Rails ist **innovativ !**

Rails ist **elegant !**

Rails ist **schnell !**

Rails ist **Produktivität !**

Rails ist **die Freude am Programmieren !**

Rails ist ein Hype!

Rails ist **innovativ !**

Rails ist **elegant !**

Rails ist **schnell !**

Rails ist **Produktivität !**

Rails ist **die Freude am Programmieren !**

Ist Rails ein Hype?

Rails?

Was ist Rails?

- Entwickelt von David Heinemeier Hansson (37signals)
- Framework für Datenbank gestützte Webanwendungen
- Eine Extraktion aus einer bestehenden Anwendung (basecamphq.com)
- Low-Level-Funktionalitäten bereits vorhanden
- Leicht wartbar
- Open-Source
- Eine wachsende Community
- Komplett in Ruby geschrieben

Ruby Entwicklung

- 1993 entwickelt von Yukihiro Matsoumoto
- Motivation: Frust durch inkonsistente Sprachen
- 1995 fertig gestellt und unter GPL verfügbar
- seit 2000 auch außerhalb von Japan bekannt

Ruby Eigenschaften

- Syntax u.a. an Eifel und C++ angelehnt
- Interpretersprache
- Komplett objektorientiert (wie z.B. Smalltalk)
- Dynamisch typisiert
- Folgt dem Prinzip der geringsten Überraschung
- Durch JRuby in Java integrierbar
- Java-Klassen durch JRuby in Ruby nutzbar

Ruby Beispiel 1: Klassen

```ruby
class Name
  attr_writer :name
  protected :name
  def initialize(name = „world“)
    @name = name
  end
end

class Greeter < Name
  def initialize(name)
    super(@name)
  end

  def say_hi
    puts “Hallo #{@name}!”
  end
end
```

```ruby
hw = Greeter.new()
hw.say_hi
==> “Hallo world!”
g = Greeter.new(“Zuhoerer”)
g.say_hi
==> “Hallo Zuhoerer!”
g.name = “Welt”
g.say_hi
==> “Hallo Welt!”
```

Ruby Beispiel 2: Dynamik

```ruby
def MYWHILE (cond)
  return if not cond
  yield
  retry
end
```

```ruby
i=0
MYWHILE(i<3) {print i, " "; i+=1}
==> 0 1 2
```

Ruby Beispiel 3: Lesbarkeit

```ruby
@names = 'Tim', 'Tom', 'Tina'

@names.each do |name|
  puts "Hallo #{name}!"
end
```

```
==> Hallo Tim
==> Hallo Tom
==> Hallo Tina
```

Auf http://tryruby.hobix.com/ steht eine interaktive Ruby-Shell zum ausprobieren zur Verfügung.

MVC-Architektur

- Model-View-Controller Framework
- Saubere Trennung von Daten / Ansichten / Verarbeitung
- Leicht wartbar

MVC-Architektur

- Model-View-Controller Framework
- Saubere Trennung von Daten / Ansichten / Verarbeitung
- Leicht wartbar

Object Relational Mapping

- Verhindert Programmierstieldifferenzen beim Arbeiten mit relationalen Datenbanken in OO-Umgebungen

MVC-Architektur

- Model-View-Controller Framework
- Saubere Trennung von Daten / Ansichten / Verarbeitung
- Leicht wartbar

Object Relational Mapping

- Verhindert Programmierstieldifferenzen beim Arbeiten mit relationalen Datenbanken in OO-Umgebungen

Create Read Update Delete - Funktionalität

- Bereitstellung der Low-Level-Datenoperationen
- Automatische Generierung mittels Scaffolding

„Covention over Configuration“

- Automatisches Konfiguration durch Namenskonventionen
- Sinnvolle Default-Werte
- Definierte Verzeichnisstruktur
- Konfiguration nur für die Datenbankanbindung notwendig, bzw. bei Abweichung von der Namenskonvention

„Covention over Configuration“

- Automatisches Konfiguration durch Namenskonventionen
- Sinnvolle Default-Werte
- Definierte Verzeichnisstruktur
- Konfiguration nur für die Datenbankanbindung notwendig, bzw. bei Abweichung von der Namenskonvention

DRY – „Don‘t Repeat Yourself“

- Verhinderung von Redundanz im Programm mittels Helpern
- Automatisches Anpassen der Datenmodelle bei Änderung der Datenbank

„Covention over Configuration“

- Automatisches Konfiguration durch Namenskonventionen
- Sinnvolle Default-Werte
- Definierte Verzeichnisstruktur
- Konfiguration nur für die Datenbankanbindung notwendig, bzw. bei Abweichung von der Namenskonvention

DRY – „Don‘t Repeat Yourself“

- Verhinderung von Redundanz im Programm mittels Helpern
- Automatisches Anpassen der Datenmodelle bei Änderung der Datenbank

Datenbankmigration

- „on-the-fly-Wechsel“ des DBMS möglich

Enviroments

- Möglichkeit zur Definition von Laufzeitumgebungen mit unterschiedlichen Parametern, z.B. die verwendete Datenbank oder die Art des Loggings

Enviroments

- Möglichkeit zur Definition von Laufzeitumgebungen mit unterschiedlichen Parametern, z.B. die verwendete Datenbank oder die Art des Loggings

WEBrick

- Out-of-the-Box Webserver (alternativ auch mit Apache und lighttpd verwendbar)
- Nur die Datenbank muss zusätzlich installiert werden

Enviroments

- Möglichkeit zur Definition von Laufzeitumgebungen mit unterschiedlichen Parametern, z.B. die verwendete Datenbank oder die Art des Loggings

WEBrick

- Out-of-the-Box Webserver (alternativ auch mit Apache und lighttpd verwendbar)
- Nur die Datenbank muss zusätzlich installiert werden

Generatoren

- Generatoren erzeugen automatisch die benötigte Datenstruktur, sowie Default-Code (für Model, Controller, Scaffolding)
- Integration von Generatoren aus der Community zur Generierung speziellerer Objekte (z.B. Login, Newsfeed, …)

Web 2.0

- Ajax und SOAP Unterstützung im Framework integriert

Web 2.0

- Ajax und SOAP Unterstützung im Framework integriert

Modularisierung

- Rails besteht aus fünf Modulen
- Weitere Module können integriert werden

  z.B. Bezahlsysteme für Shops

Web 2.0

- Ajax und SOAP Unterstützung im Framework integriert

Modularisierung

- Rails besteht aus fünf Modulen
- Weitere Module können integriert werden
  z.B. Bezahlsysteme für Shops

TestDrivenDevelopment

- Rails kommt mit einer kompletten Testsuite
- Alle Schichten der MVC-Architektur können getestet werden
- Profiling

Model View Controller

in Rails

Model View Controller

- Architekturmuster
- 1970 veröffentlicht und erste Anwendung in Smalltalk
- Vorwiegender Einsatz bei GUI-basierten Systemen

Model View Controller

- Architekturmuster
- 1970 veröffentlicht und erste Anwendung in Smalltalk
- Vorwiegender Einsatz bei GUI-basierten Systemen

Model

- Status des Systems
- Manipuliert den Status

Model View Controller

- Architekturmuster
- 1970 veröffentlicht und erste Anwendung in Smalltalk
- Vorwiegender Einsatz bei GUI-basierten Systemen

Model

- Status des Systems
- Manipuliert den Status

View

- Visualisierung
- Interaktion mit Benutzer
- Mehrere Views möglich
- Ein View ist immer an ein konkretes Model gebunden

Controller

- Verbindet View und Model
- Steuert die Verarbeitung der Benutzerinteraktion

Controller

- Verbindet View und Model
- Steuert die Verarbeitung der Benutzerinteraktion

Vorteile

- Keine Vermengung zwischen Programmcode und Ausgabeelementen
- Austauschbarkeit der Präsentationsschicht
- Direkte Reaktion auf Benutzerinteraktion

Nachteil

- Enge Bindungen zwischen den einzelnen Objekten

MVC - Ablauf

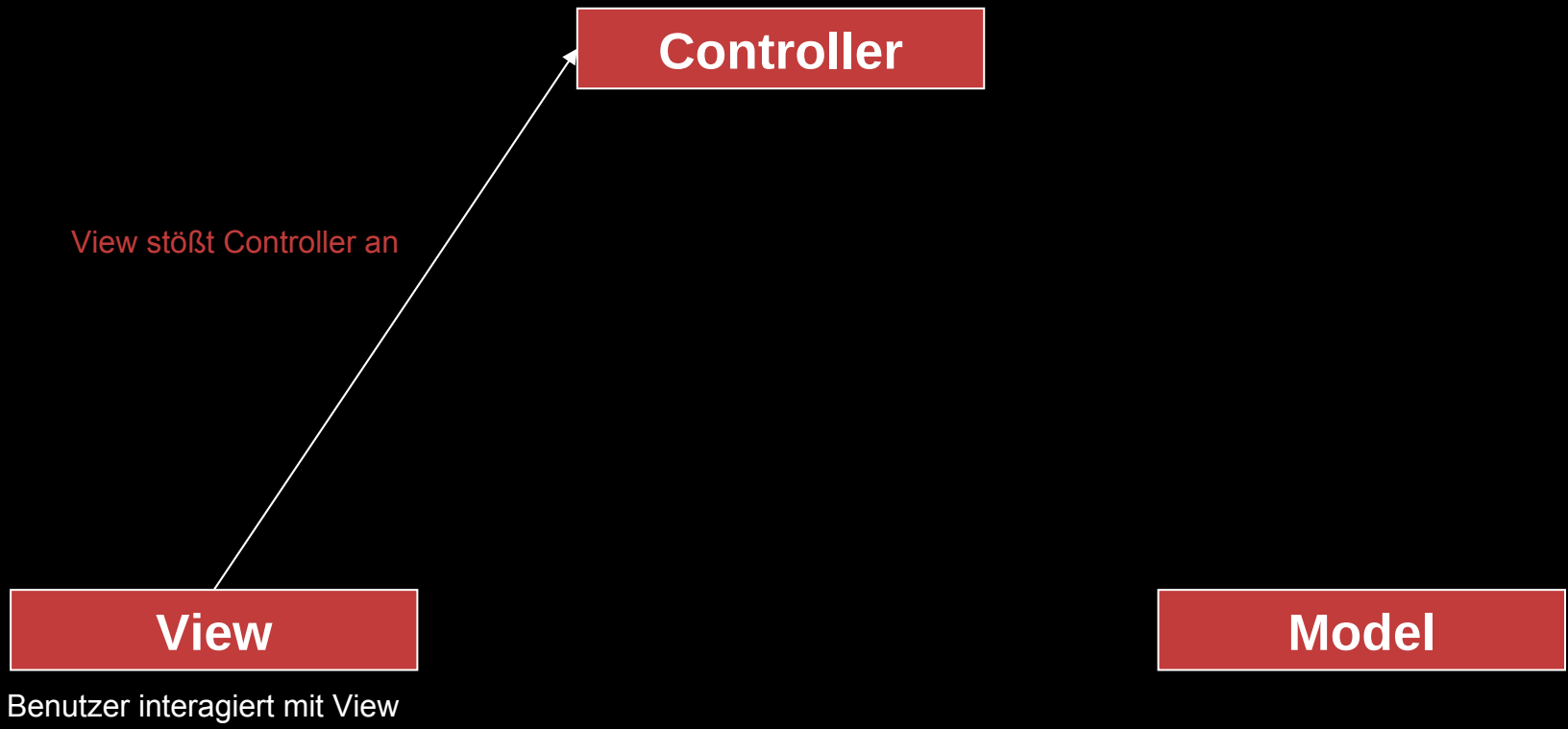
Controller

View

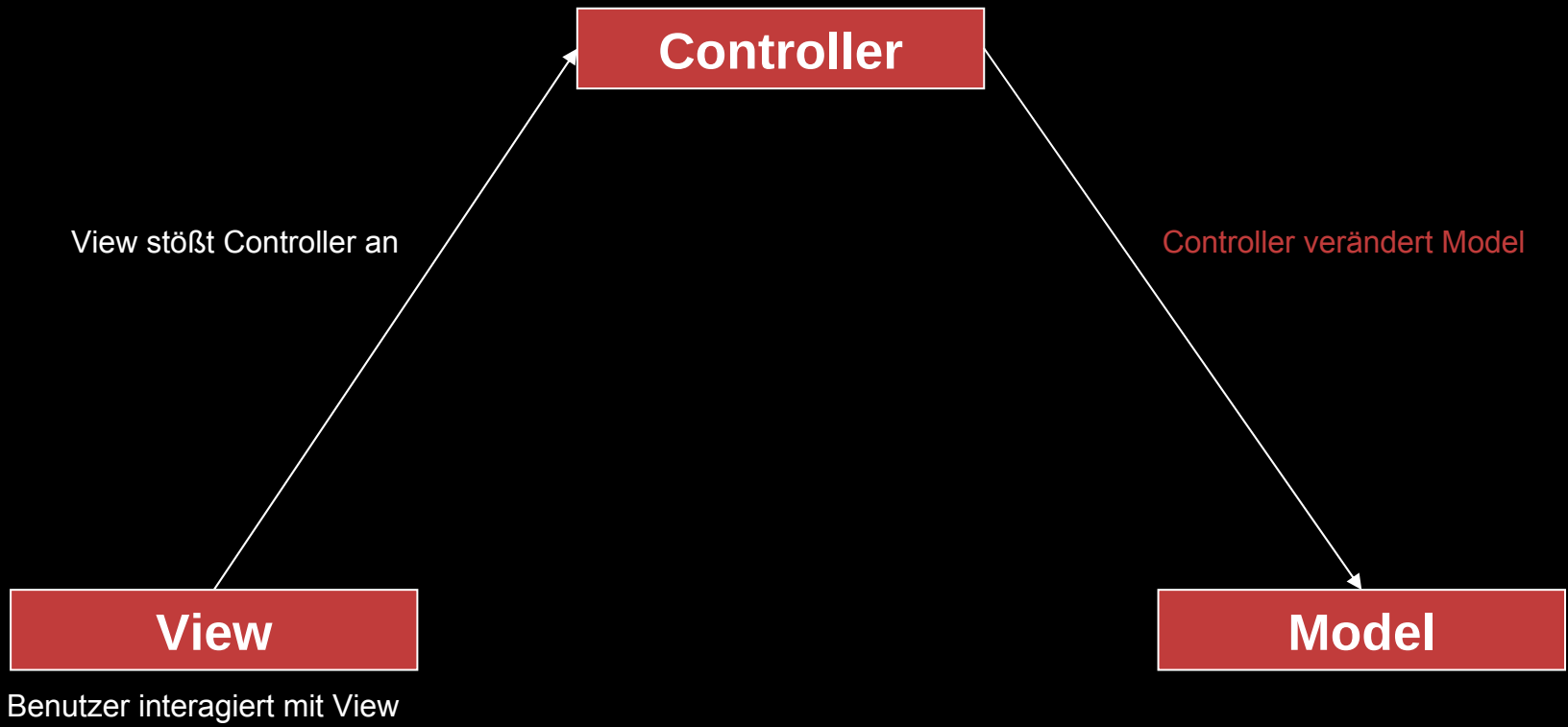
Benutzer interagiert mit View

Model

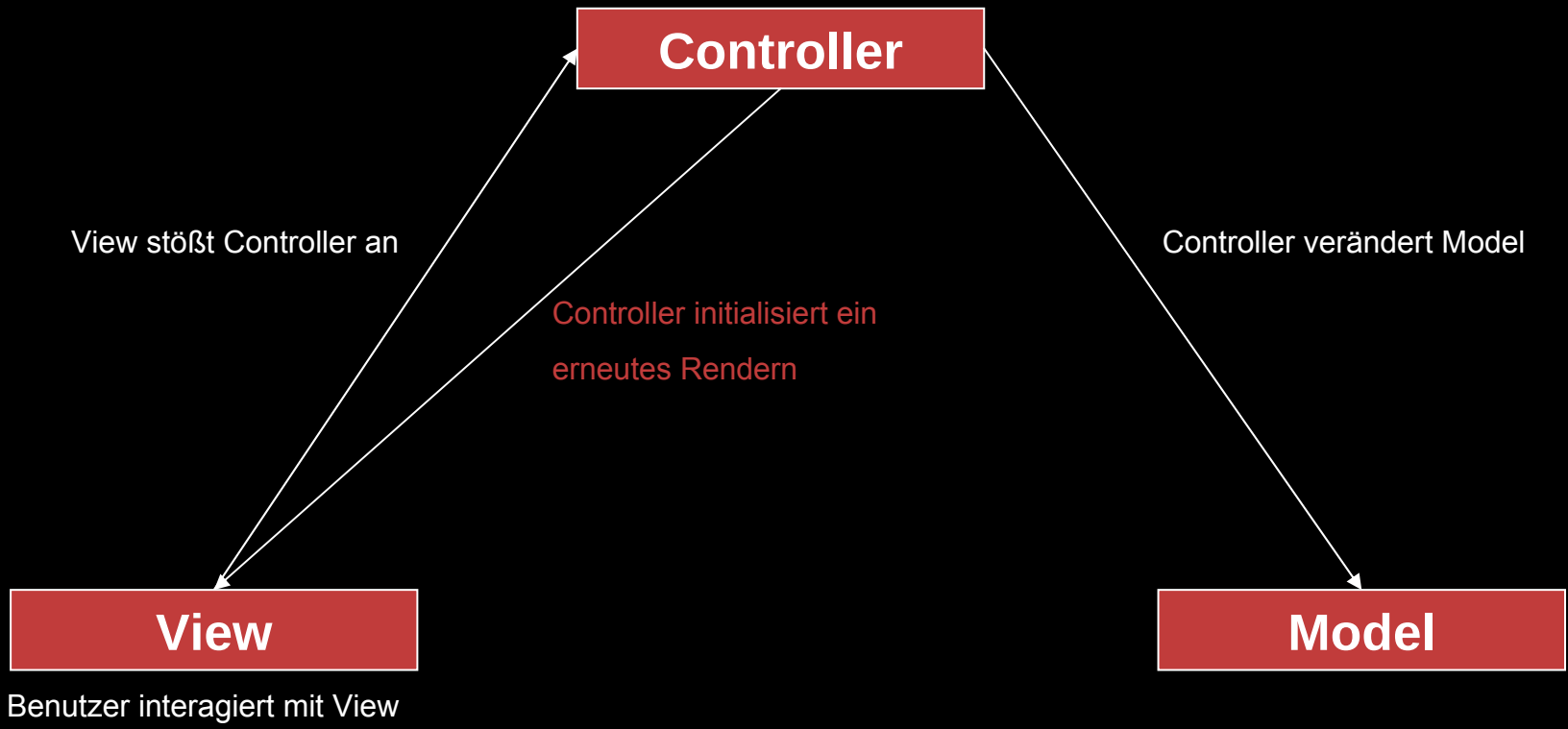
MVC - Ablauf


Controller
View stößt Controller an
View
Model
Benutzer interagiert mit View

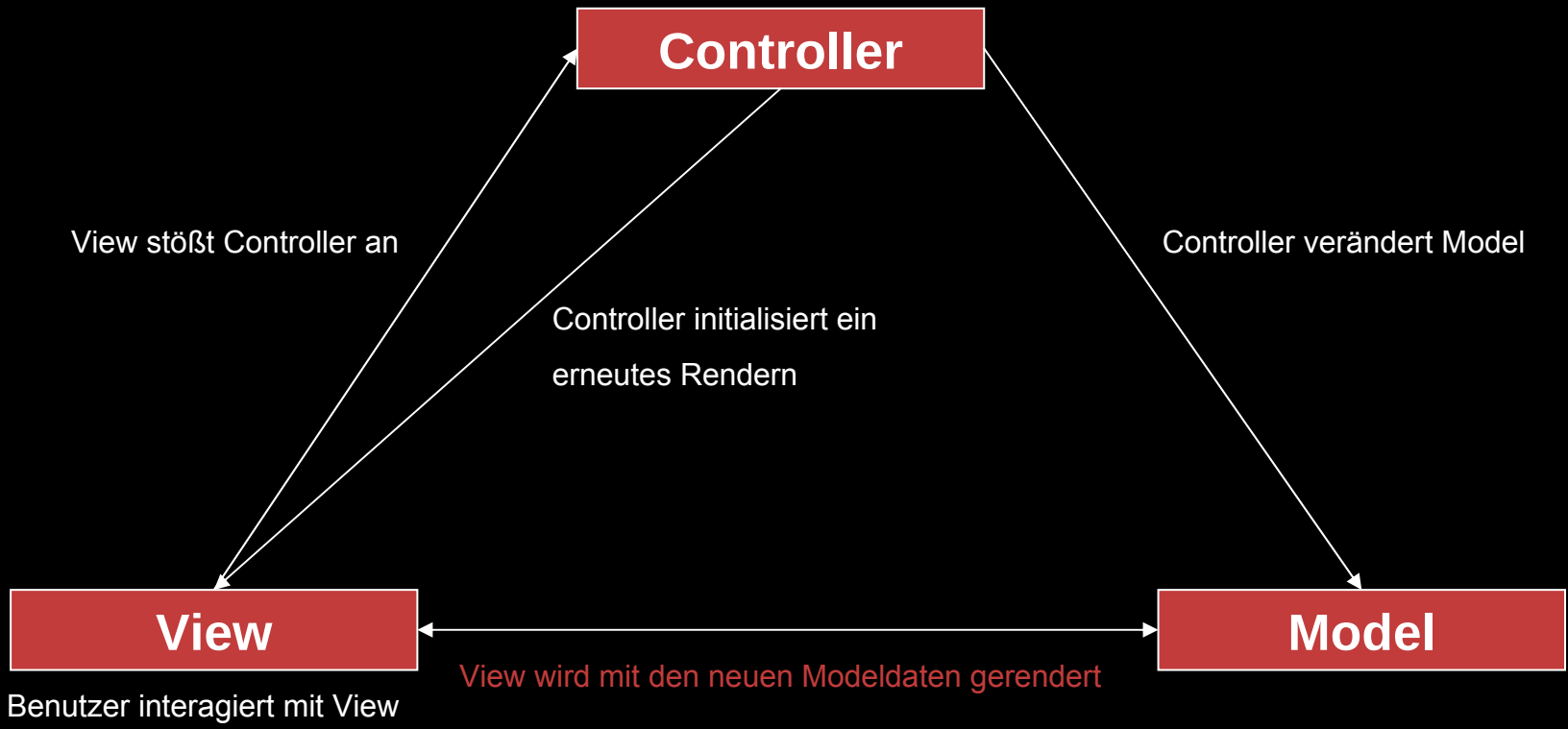
MVC - Ablauf


Controller
View stößt Controller an
Controller verändert Model
View
Model
Benutzer interagiert mit View

MVC - Ablauf


Controller
View stößt Controller an
Controller initialisiert ein
erneutes Rendern
Controller verändert Model
View
Model
Benutzer interagiert mit View

MVC - Ablauf


Controller
View stößt Controller an
Controller initialisiert ein
erneutes Rendern
Controller verändert Model
View
Model
View wird mit den neuen Modeldaten gerendert
Benutzer interagiert mit View

Problem der Benutzerinteraktion bei Webanwendungen

- Webseiten ≠ direkte Interaktion
- Interaktion nur indirekt über URLs

Problem der Benutzerinteraktion bei Webanwendungen

- Webseiten ≠ direkte Interaktion
- Interaktion nur indirekt über URLs

Problemlösung: Erweiterung um eine vorschaltete Instanz

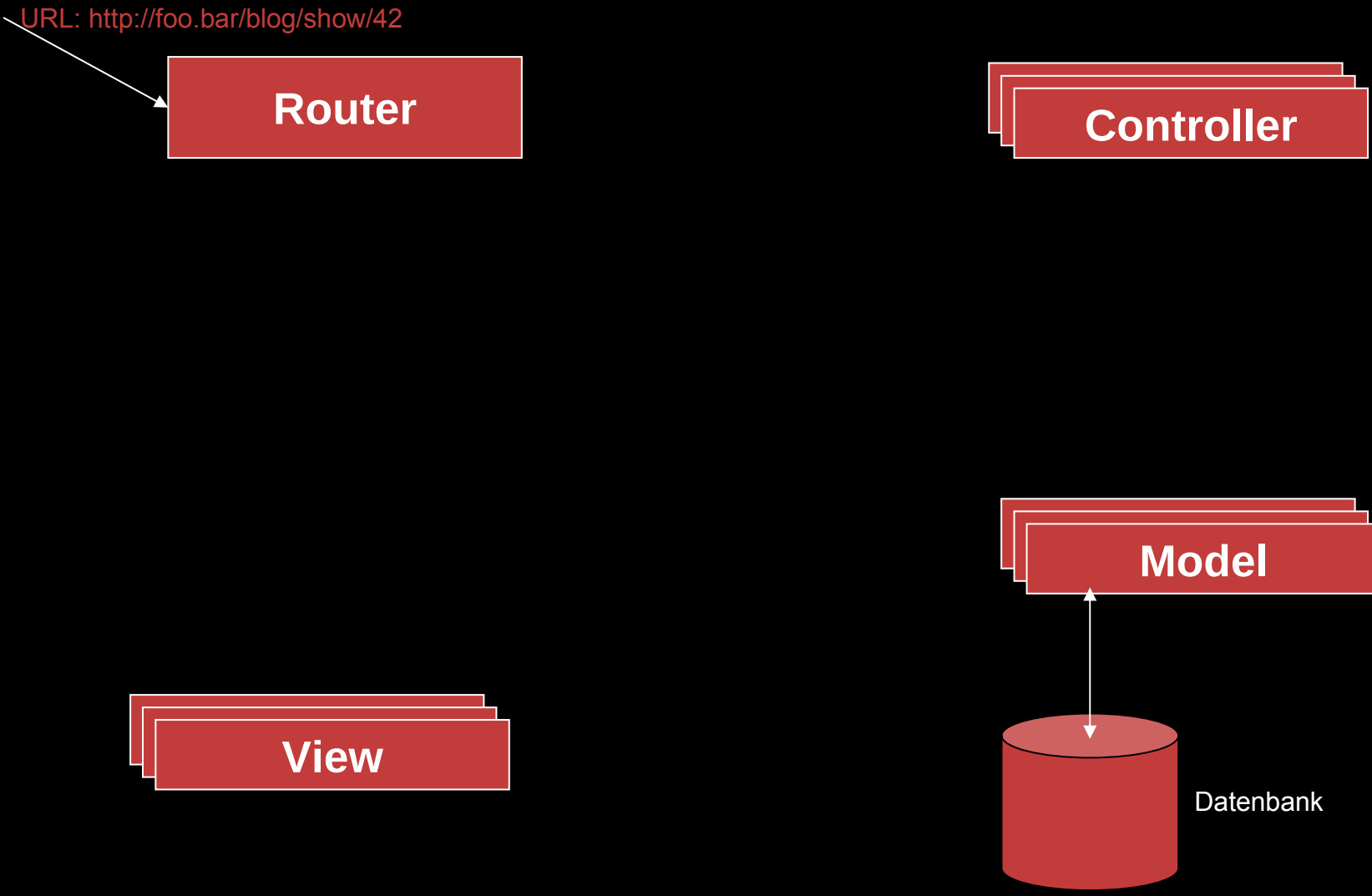
- Analyse der URLs
- Auslösen von Events

➔MVC Version 2.0

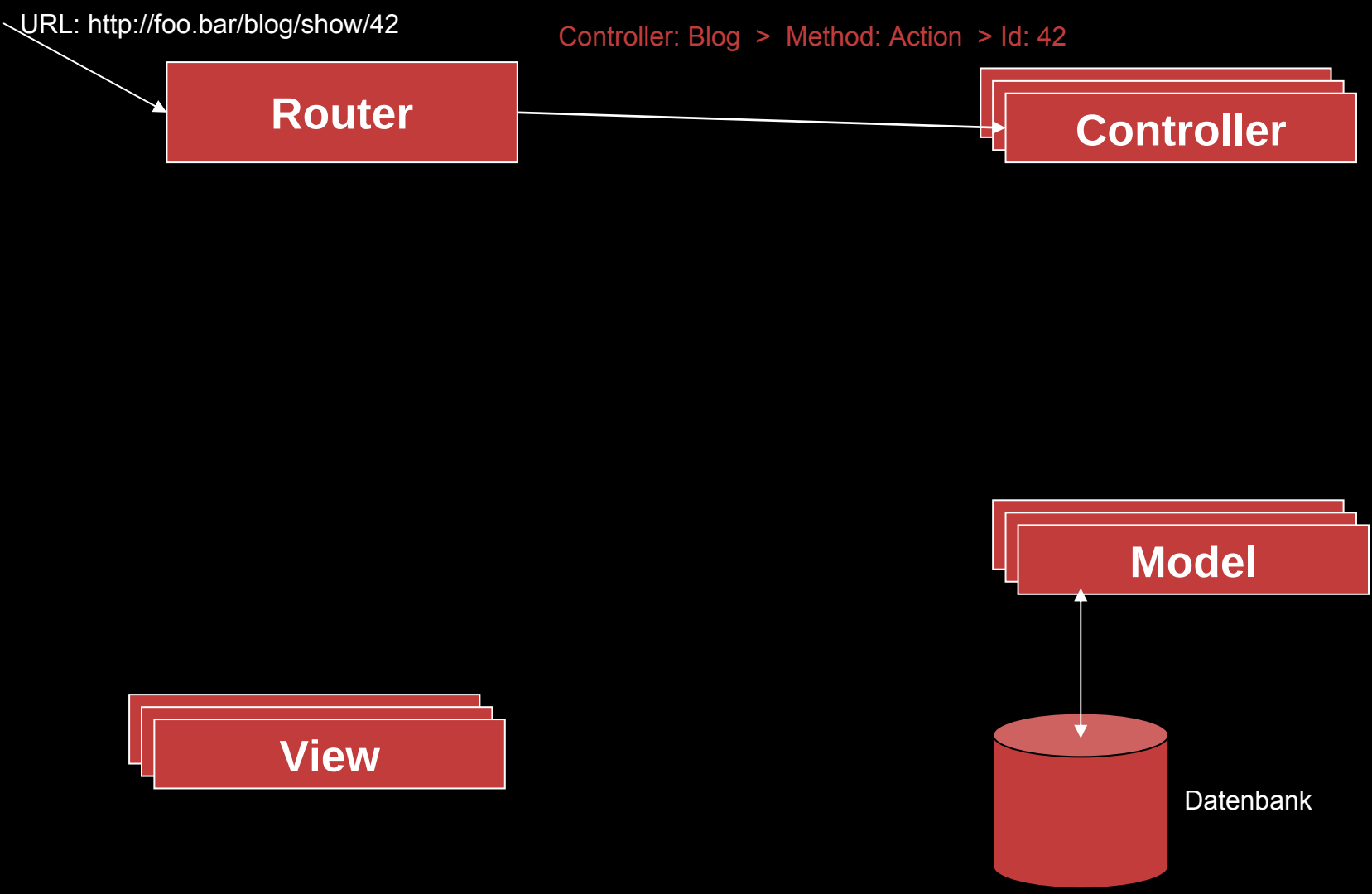
Rails MVC Architektur

- Jede URL wird vom Router der Anwendung auf die gewünschte Methode des angegeben Controllers gemappt
- Datenbanktabelle == Model
- Funktionsgruppe == Controller
- Funktion == Methode und View

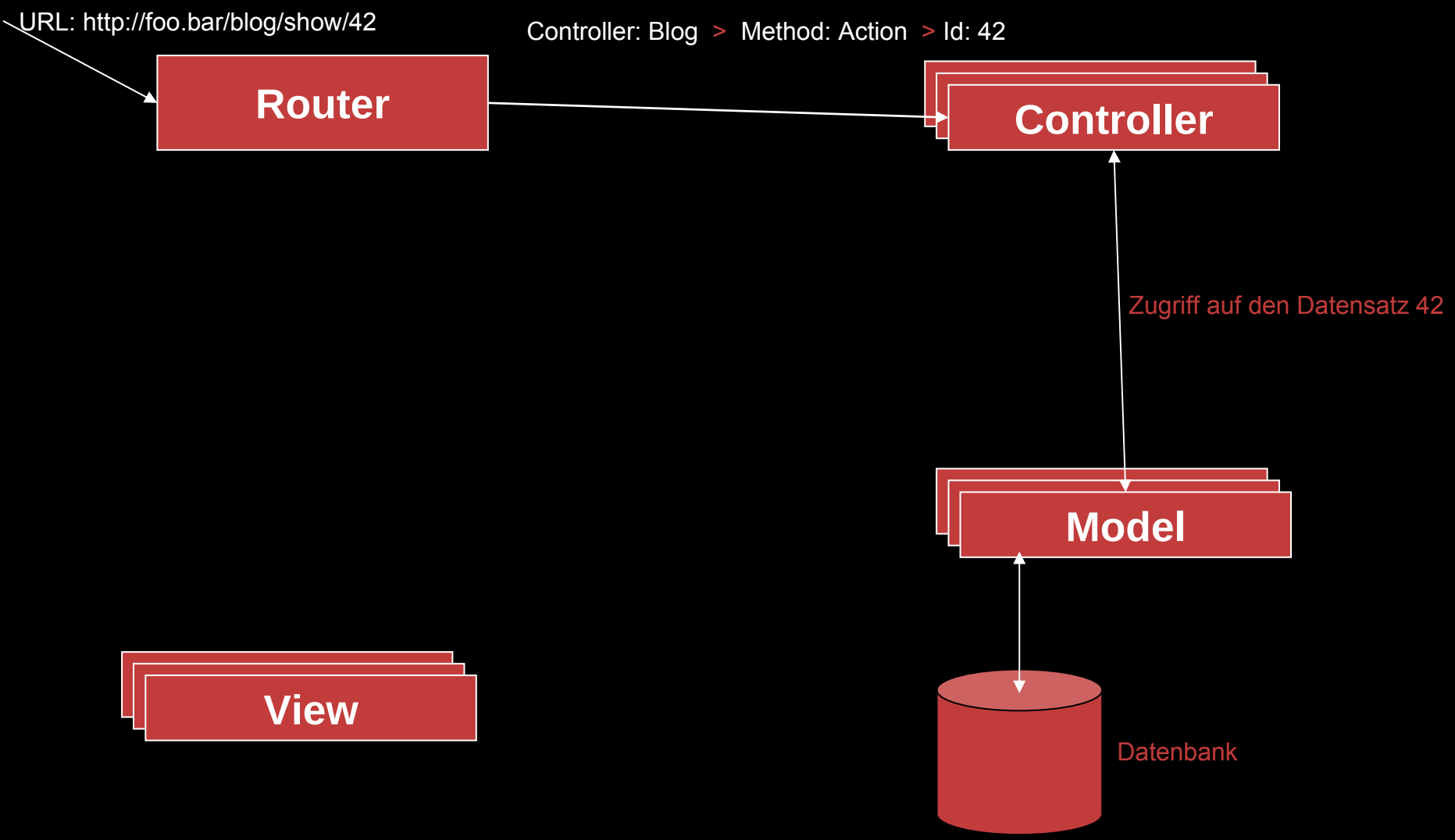
MVC – Ablauf in Rails



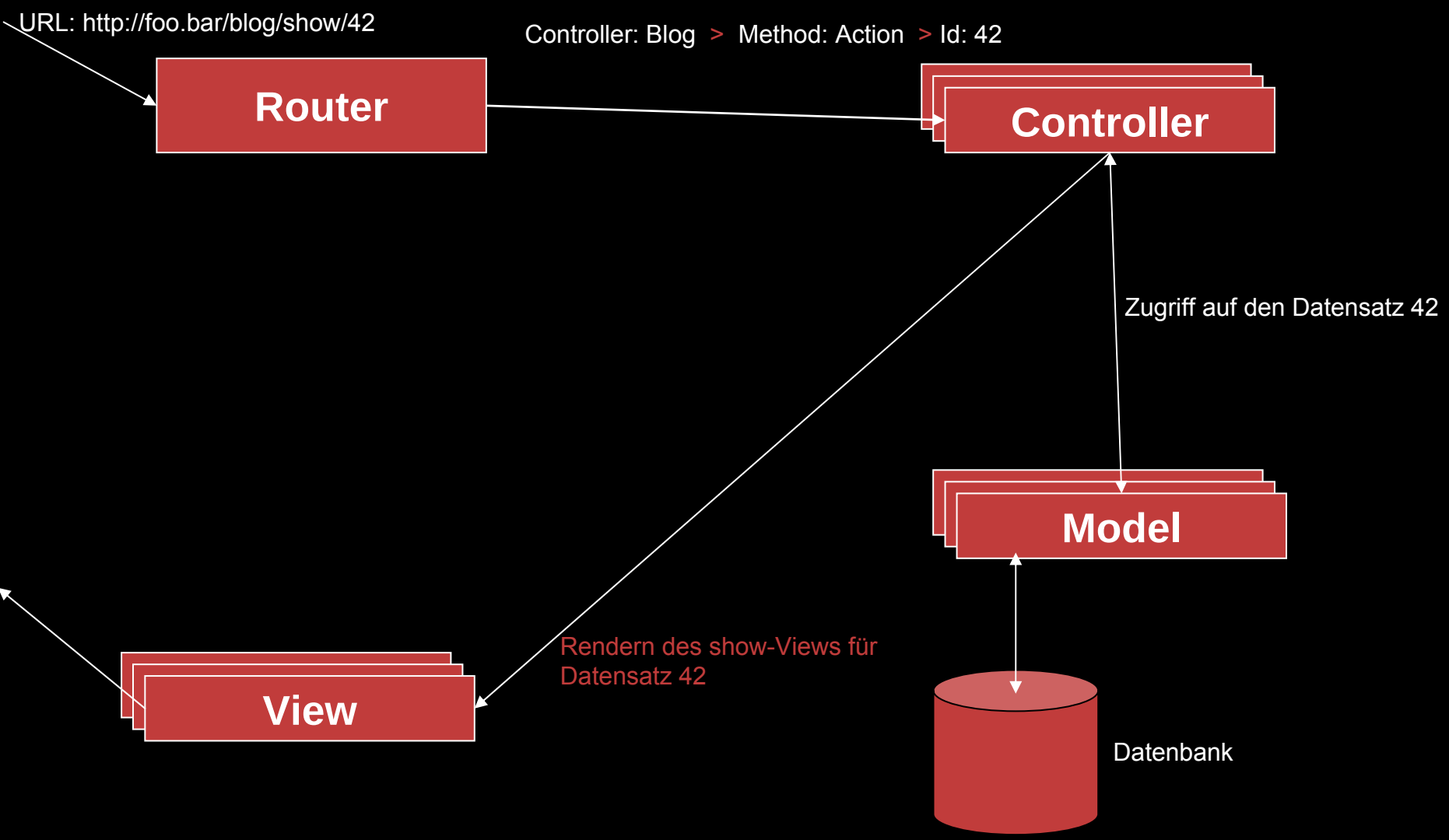
MVC – Ablauf in Rails



MVC – Ablauf in Rails



MVC – Ablauf in Rails



Routing mit Rails

- Routingregeln in config/routes.rb festgelegt
- Priorität festgelegt durch Reinfolge
- Nicht regelkonforme URL werden auf Fehlerseiten umgeleitet oder können abgefangen werden (catchall)

Beispiel:

```
map.connect 'show_by_date/:year/:month/:day',
              :controller   => 'Newslist',
              :action       => 'show_by_date',
              :requirements => { :year  => /(19|20)\d\d/,
                                 :month => /[01]?\d/,
                                 :day   => /[0123]?\d/},
              :month => nil,
              :day => nil
```

Reverse Routing mit Rails

- Verlinkung durch Reverse Routing (url_for())
- Kontext sensitiv

Beispiel:

```
#Kontext:
#controller    : Newslist
#action        : show_by_date
#year          : 2006
#month         : 11
#day           : 15

url_for(:day => '14')
==> /show_by_date/2006/11/14
```

Model – Active Record (AR)

- Eine Model-Klasse == Eine Tabelle der Datenbank
- Ein Model-Klassen-Objekt == Ein Datensatz
- Ein Objekt-Attribut == Eine Spalte der Tabelle
- Es werden KEINE SQL-Kenntisse benötigt

Model Konventionen

Tabellenname:	news_items
Klasse:	NewsItem
Pfad:	app/models/news_item.rb

Abbildung von relationalen Datenbanken auf Klassen

- ORM => Object Relation Mapping
- Rails generiert automatisch die notwendige Klasse

→ keine Konfiguration notwendig!

Code einer Modelklasse

```
class NewsItem < ActiveRecord::Base
end
```

Kein SQL notwendig?

- CRUD = Create, Read, Update, Delete
- Active Record stellt CRUD bereit
- Die Operationen sind Bestandteil jeder von AR erbenden Klasse
- Bereitstellung dynamischer Suchfunktionen

Beispiele

```
a_news = NewsItem.new
a_news.schlagzeile = 'ROR ist da'
a_news.text ='...'
a_news.save

get_news = NewsItem.find(:first, :conditions ["schlagzeile = ?", "ROR ist da"])
puts "#{get_news.id}"
==> 1

a_news.update(1, :text => "Kein Text")

a_news.delete(1)
```

Abhängigkeiten zwischen Tabellen modellieren

- Erkennung nicht durch AR automatisiert
- Spezielle Spalte in DB-Tabelle erforderlich: parent_id
- AR stellt jedoch passende Funktionalität
- 1 : 1, 1 : n, m : n

Definition der Abhängigkeiten

```ruby
class NewsItem < ActiveRecord::Base
  belongs_to  :category
  has_many    :comments
  has_one     :author
#  [...]
end

class Category < ActiveRecord::Base
  has_and_belongs_to_many  :categories
#  [...]
end
```

```ruby
news = NewsItem.find_by_id(1)
puts news.author
puts news.catergory
news.comments.each do |com| %>
  puts com.text
end

cat = Category.find_by_name("web")
cat.comments.each do |subcat| %>
  puts subcat.name
end
```

Konsistenzerhaltung durch „validate…“

- 18 pre definierte Validatoren
- Durch überschreiben der Methoden kann die eigene Datenstruktur konsistent gehalten werden

```
class NewsItem < ActiveRecord::Base

  #Stellt immer sicher das die Schlagzeile eindeutig ist
  def validate
    if NewsItem.find_by_schlagzeile(schlagzeile)
      errors.add(:schlagzeile, 'Schlagzeile schon vorhanden!')
    end
  end

end
```

Konsistenzerhaltung durch Model-Callbacks

- Werden vor oder nach create, update oder delete aufgerufen

```
class NewsItem < ActiveRecord::Base
  before_destroy :delete_all_comments
  def delete_all_comments
      Comment.delete_all(["news_id =?", self.id])
  end
#  [...]
end
```

View – Action View

- Rails Template-System
- Für HTML und strukturierte Dokumente
- Templates, Layouts, Partials
- Helper

View Konventionen

URL:	http://.../blog/list
Template:	app/views/blog/show[.rhtml | .rxml]
Helper:	BlogHelper
Helper-Pfad:	app/helpers/blog_helper.rb

Action View Umgebung

- Zugriff auf alle Insantzvariablen (@varname) des Controllers:
  session, params, response, request, headers, sowie eigene
- Zusätzliche Variable controller mit Referenz auf das Controller-Objekt

Action View Umgebung

- Zugriff auf alle Insantzvariablen (@varname) des Controllers:
  session, params, response, request, headers, sowie eigene
- Zusätzliche Variable controller mit Referenz auf das Controller-Objekt

Buildertemplates

- Erzeugen strukturierte Text, z.B. XML
- Dateinendung .rxml
- Verwendet die Ruby Bibliothek Builder
- Support von: namespace, entities, processing instructions, usw.

RXML-Beispiel

```
xml.div(:class => "authorlist") do
  @authors.each do |author|
    xml.div(:class =>"author") do
      xml.p(author.firstname, :class => "name")
      xml.br
      xml.p(author.lastname, :class => "name")
    end
  end
end
```

```
<div class="authorlist">
  <div class="author">
      <p class="name">Tim</p>
      <br/>
      <p class="name">Tailor</p>
  </div>
  <div class="author">
      <p class="name">Al</p>
      <br/>
      <p class="name">Bundy</p>
  </div>
</div>
```

RHTML - Templates

- Buildertemplates nicht für HTML geeignet
- Besser: RHTML = HTML + inline Ruby
- RHTML wird in ein Rubyscript umgewandelt und ausgeführt
- Syntax an JSP angelehnt

RHTML - Templates

- Buildertemplates nicht für HTML geeignet
- Besser: RHTML = HTML + inline Ruby
- RHTML wird in ein Rubyscript umgewandelt und ausgeführt
- Syntax an JSP angelehnt

Inline-Ruby: <%= ... %>

- Eingeschlossener Code wird ausgewertet
- Zu einem String konvertiert
- Ausgegeben

RHTML - Templates

- Buildertemplates nicht für HTML geeignet
- Besser: RHTML = HTML + inline Ruby
- RHTML wird in ein Rubyscript umgewandelt und ausgeführt
- Syntax an JSP angelehnt

Inline-Ruby: <%= … %>

- Eingeschlossener Code wird ausgewertet
- Zu einem String konvertiert
- Ausgegeben

Inline-Ruby: <% … %>

- Eingeschlossener Code wird nur ausgewertet
- Auch über mehrere Tags

RHTML - Beispiel

```
<h1>News</h1>
<table>
<% for newsmessage in @newsmessages %>
  <tr>
  <% for column in Newsmessage.content_columns %>
    <td><%=h newsmessage.send(column.name) %></td>
  <% end %>
  </tr>
<% end %>
</table>
```

```
<h1>News</h1>
<table>
  <tr>
    <td>ROR ist da!</td>
    <td>Kein Text</td>
  </tr>
  <tr>
    <td>Poke High Schueler verpasst Rekord!</td>
    <td>Al Bundy weiterhin Rekordhalter</td>
  </tr>
</table>
```

Layouts

- Ein Layout per Controller
- Auch manuell setzbar

```
#Layout
<html><head>
 <title>
  <%= controller.action_name %>
 </title>
 <%= stylesheet_link_tag 'scaffold'%>
</head><body>
<p style="color: green">
 <%= flash[:notice] %>
</p>
<%= yield  %>
</body></html>
```

Layouts

- Ein Layout per Controller
- Auch manuell setzbar

```
#Layout
<html><head>
 <title>
  <%= controller.action_name %>
 </title>
 <%= stylesheet_link_tag 'scaffold'%>
</head><body>
<p style="color: green">
 <%= flash[:notice] %>
</p>
<%= yield  %>
</body></html>
```

Partial

- Kapselung der Präsentation von Objekten
- DRY-Ansatz
- Partials beginnen mit einem _

```
#Aufruf
<%= render( :partial => "blogentry",
            :object  => Blog.find_by_id(@params[:bolg_id] ) %>

#Partial
<div id="blogentry">
  <div = "headline"><%= blogentry.headline %></div>
  <div = "text"><%= blogentry.text %></div>
</div>
```

Template Helper

- Helper == logische Konsequenz von DRY
- Format- und Form-Helper
- eigene problemspezifische Helper definierbar
- Wichtigster Helper: h
- Können auch in beliebige Controller eingebunden werden

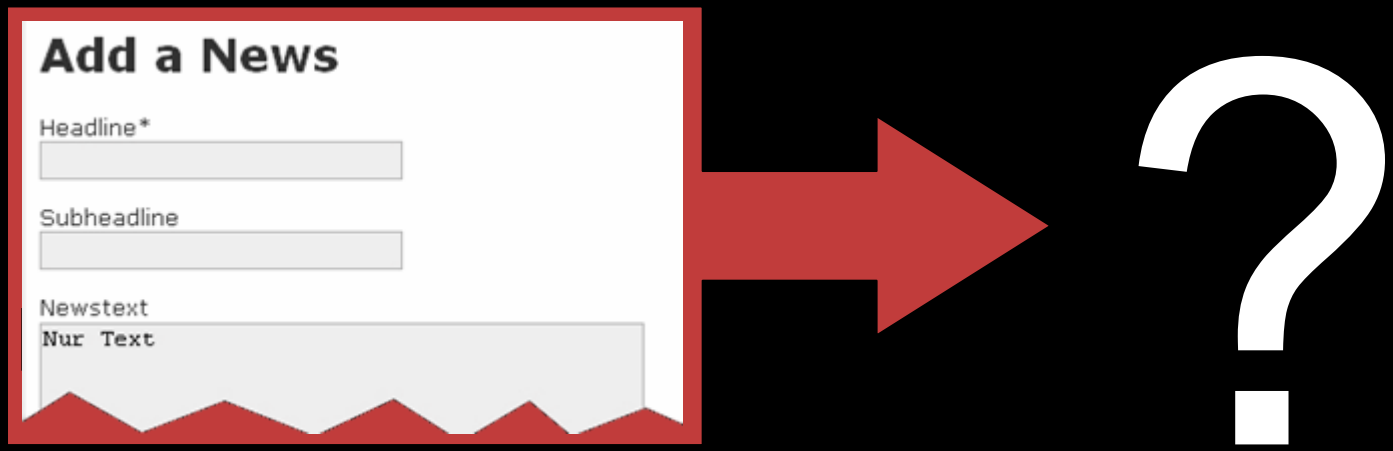
Template Helper

- Helper == logische Konsequenz von DRY
- Format- und Form-Helper
- eigene problemspezifische Helper definierbar
- Wichtigster Helper: h
- Können auch in beliebige Controller eingebunden werden

Format-Helper

```
<%= image_tag("imagefile", :size => "20x20") %>

<%= link_to "Bloghome", :action => "Home" %>

<%= debug(hash) %>

<%= highlight(@text, "word") %>

<%= truncate(@text, 42) %>

<%= distance_of_time_in_words(Time.now, time.now + 33, true) %>
==> half a minute

usw...
```

Form/Field-Helper



Form/Field-Helper

Form/Field-Helper

```
#Model
class Newsmessage < ActiveRecord::Base
  def validate_on_create
    if headline.blank?
      errors.add_to_base('headline is missing')
    end
  end

end
```

```
#View
<h1>Add a News</h1>
<%= error_messages_for 'newsmessage' %>
<%= start_form_tag :action => 'create' %>
    <p><label for="newsmessage_headline">Headline*</label><br/>
    <%= text_field 'newsmessage', 'headline'  %></p>

    <p><label for="newsmessage_subheadline">Subheadline</label><br/>
    <%= text_field 'newsmessage', 'subheadline'  %></p>

    <p><label for="newsmessage_newstext">Newstext*</label><br/>
    <%= text_area 'newsmessage', 'newstext'  %></p>

    <%= submit_tag "Create" %>
<%= end_form_tag %>
```

Form/Field-Helper

```ruby
#Controller
class NewsadminController < ApplicationController
  def new
    @newsmessage = Newsmessage.new
  end

  def create
    @newsmessage = Newsmessage.new(params[:newsmessage])
    if @newsmessage.save
      flash[:notice] = 'Newsmessage was successfully created.'
      redirect_to :action => 'list'
    else
      render :action => 'new'
    end
  end

end
```

Controller – Action Controller

- Koordinator der Anwendung
- Methoden = Anwendungsfunktionen
- Enge Verknüpfung mit den Views
- Verwaltet Models und Umgebungsvarbiablen

Controller Konventionen

URL:	http://.../blog/list
Klasse:	BlogController
Pfad:	app/controller/blog_controller.rb
Methode:	show()
Layout:	app/views/layouts/blog.rhtml

Umgebungsvariablen

- request : Aufrufsdaten
- params : Parameter des Aufrufs
- sessions : Aktuelle Sitzungsdaten
- cookies : Daten des Cookies

Umgebungsvariablen

- request : Aufrufsdaten
- params : Parameter des Aufrufs
- sessions : Aktuelle Sitzungsdaten
- cookies : Daten des Cookies

render()

- Dient dem Rendern eines bestimmten Outputs

```
render( :inline => %{<pre><%= debug(params) %></pre>} %>
```

- :inline
- :partial
- :template
- :text

Methodendelegierung

- Delegiert Arbeit auf bestehende Funktionalität (DRY)
- Sicherer als ein render(:template => “bar“)

```
redirect_to :action => 'show_list'
```

Methodendelegierung

- Delegiert Arbeit auf bestehende Funktionalität (DRY)
- Sicherer als ein render(:template => “bar“)

```
redirect_to :action => 'show_list'
```

Flash

- Kommunikation über den Controller hinaus
- Request gebunden
- Session

```
class SomeController < ApplicationController
  def index
    flash[:welcome] = 'Willkommen'
    redirect_to :action => 'show_home'
  end
end
```

```
<div id="flash"><%= flash[:welcome] %></div>
```

Filter

- Umschließen Methoden
- Zugriffsrechte, Logging, …

```
class someClass < ApplicationController
  before_filter :checkLogin, :only   => [:new, :del]
  after_filter  :logIt,      :except => :list
  [...]
end
```

Filter

- Umschließen Methoden
- Zugriffsrechte, Logging, …

```
class someClass < ApplicationController
  before_filter :checkLogin, :only   => [:new, :del]
  after_filter  :logIt,      :except => :list
  [...]
end
```

Verify bei Contollern

- Filter mit automatisierter Methodendeklatarion

```
class someClass < ApplicationController
  verify :only => [:new, :del],
         :methode => :post,
         :session => :user_loggedin,
         :add_flash => {:info => "please login first"},
         :redirect_to => :index
  [...]
end
```

Rails + Generatoren

Die erste Railsanwendung

1. Datenbank erzeugen
2. Eine Railsanwendung anlegen
3. Datenbankverbindung konfigurieren
4. Scaffold erzeugen
5. WEBrick starten

items	
PK	**id**
	itemtype content date

Testing

1. Unit-Tests
2. Functional Tests

Rails = Hype ?

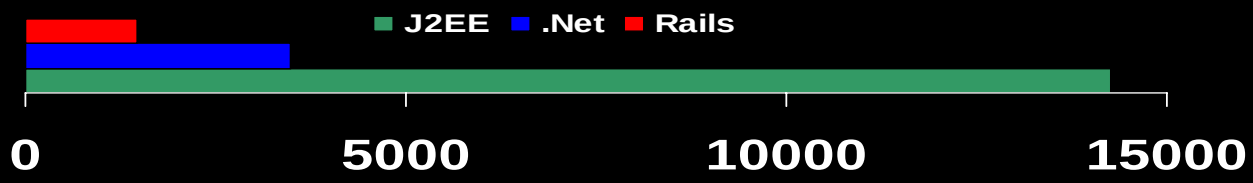
Vorteile

- Wenig LOC als vergleichbare Javaanwendungen
- Echte OO im Gegensatz zu PHP
- Einfachheit
- Nicht im Labor entstanden
- Schnell sichtbare Erfolge
- Fast Prototyping
- Guter Code wird häufig kopiert: Grails, cakePHP, Biscuit, Monorail, …
- Open Source und eine schnell wachsende Community

Vorteile

- Wenig LOC als vergleichbare Javaanwendungen
- Echte OO im Gegensatz zu PHP
- Einfachheit
- Nicht im Labor entstanden
- Schnell sichtbare Erfolge
- Fast Prototyping
- Guter Code wird häufig kopiert: Grails, cakePHP, Biscuit, Monorail, …
- Open Source und eine schnell wachsende Community

LOC-Beispiel - Petstore



Nachteile

- Interpretersprache ➔ langsamer als J2EE / asp.Net
- Wenige Webhosts
- Wenig Kontrolle
- Überladene API-Dokumentation (http://api.rubyonrails.org/)

Nachteile

- Interpretersprache ➔ langsamer als J2EE / asp.Net
- Wenige Webhosts
- Wenig Kontrolle
- Überladene API-Dokumentation (http://api.rubyonrails.org/)

Ausblick – Rails 1.2 / Ruby 2.0

- REST-Support (Web Service)
- Ein Controller für unterschiedliche Clients (z.B. Mobile, Browser, …)
- Active Ressource (Verwenden von Webservices in Railsmanier)
- Ruby-VM (Yarv-Projekt)
- Ruby-Bytecode-Format

Fragen ?

Vielen Dank !