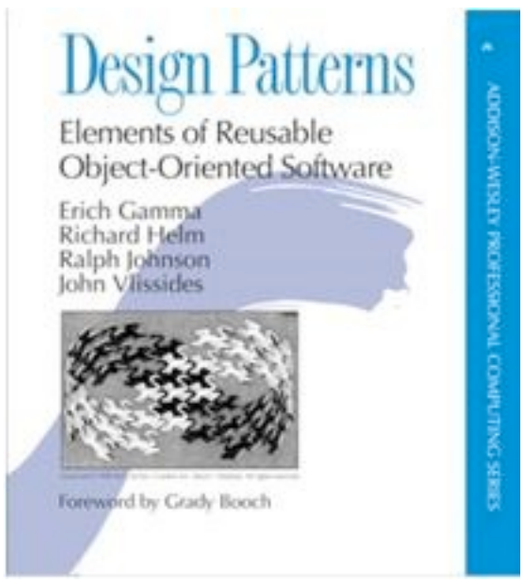
Model-View-Controller (MVC)

with Ruby on Rails

Software Languages Team
University of Koblenz-Landau
Ralf Lämmel and Andrei Varanovich

MVC - a classic definition

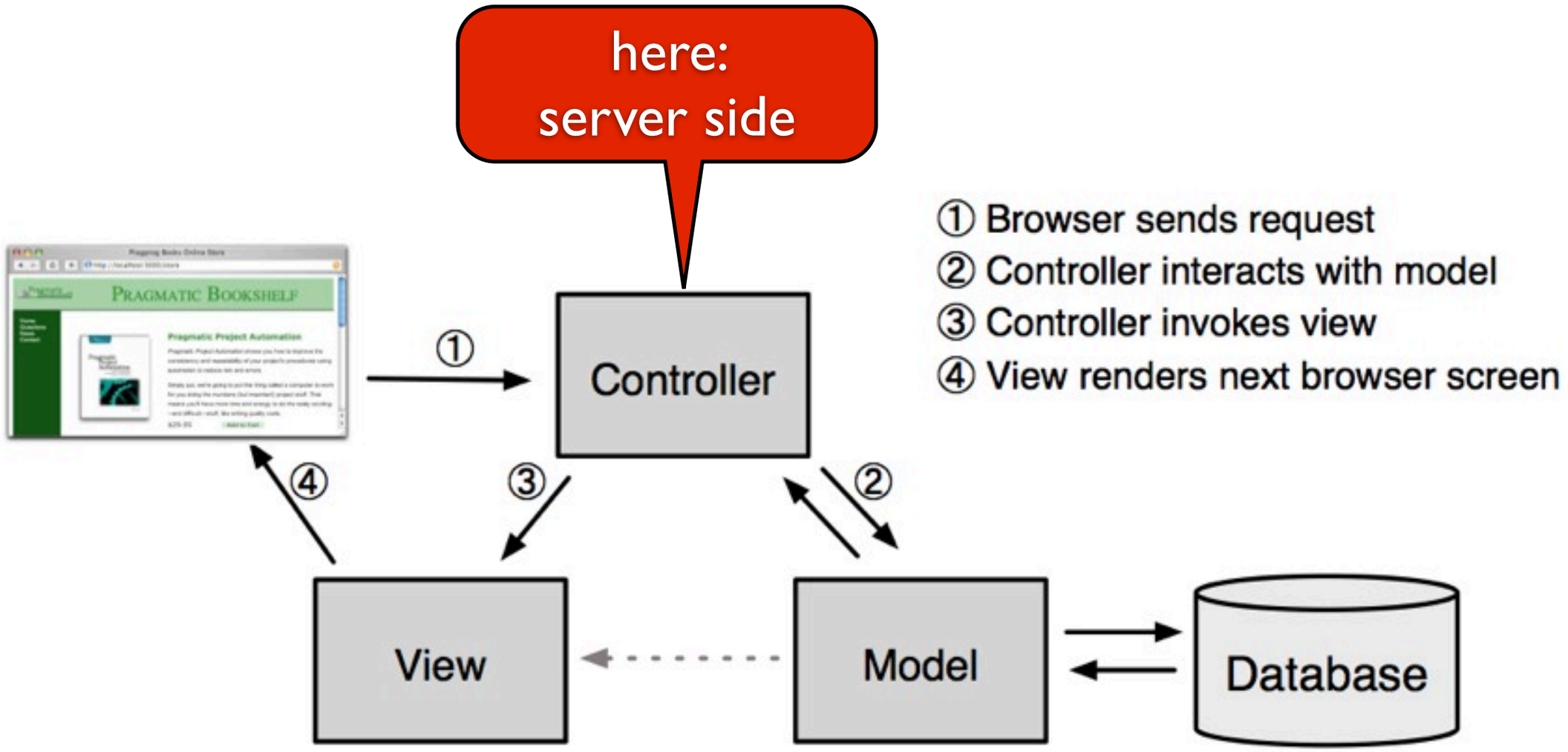
- The ***Model*** is the application object
- The ***View*** is its screen presentation
- The ***Controller*** defines the way the user interface reacts to user input



Model–View–Controller (MVC) is a computer software **design pattern** that separates the representation of information from the user's interaction with it. The model consists of application data and business rules, and the controller mediates input, converting it to commands for the model or view. A view can be any output representation of data, such as a chart or a diagram. Multiple views of the same data are possible

http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

The Model-View-Controller Architecture



What is Ruby on Rails?

- A web application development framework written in the Ruby language.
- It makes the assumption that there is a ‘best’ way to do things, and it’s designed to encourage that way – and in some cases to discourage alternatives.

Rails philosophy

- ***DRY*** – "Don't Repeat Yourself" – suggests that writing the same code over and over again is a bad thing.
- ***Convention Over Configuration*** – means that Rails makes assumptions about what you want to do and how you're going to do it, rather than requiring you to specify every little thing through endless configuration files.
- ***REST is the best pattern*** for web applications – organizing your application around resources and standard HTTP verbs is the fastest way to go.

Ruby

Ruby is a dynamic, reflective, general-purpose object-oriented programming language that combines syntax inspired by Perl with Smalltalk-like features. It was also influenced by Eiffel and Lisp.

http://en.wikipedia.org/wiki/Ruby_(programming_language

Why Rails is relevant in the Web MVC context?

Because its build to ***enforce*** using MVC as a pattern.

Model

- Maps to a table in a database. By convention, a model named Company will map to the database table companies, and the model will have a filename *company.rb* within *app/models* folder.

```ruby
class Company < ActiveRecord::Base
  validates :name,  :presence => true
  has_many :departments
end
```

```ruby
class Department < ActiveRecord::Base
  belongs_to :company
  belongs_to :department
  has_many :departments
  has_many :employees
end
```

```ruby
class Employee < ActiveRecord::Base
  belongs_to :department
end
```

Controller

- Responds to external requests from the web server to the application, and responds to the external request by determining which view file to render

```ruby
class CompaniesController < ApplicationController
  # GET /companies
  # GET /companies.json
  def index
    @companies = Company.all

    respond_to do |format|
      format.html # index.html.erb
      format.json { render :json => @companies }
    end
  end
```

how to respond to certain HTTP requests

Controller (II)

- Handles *people-friendly URLs* extremely well.
- Manages *caching*, which can give applications orders-of-magnitude performance boosts.
- Manages *sessions*, giving users the impression of ongoing interaction with our applications.

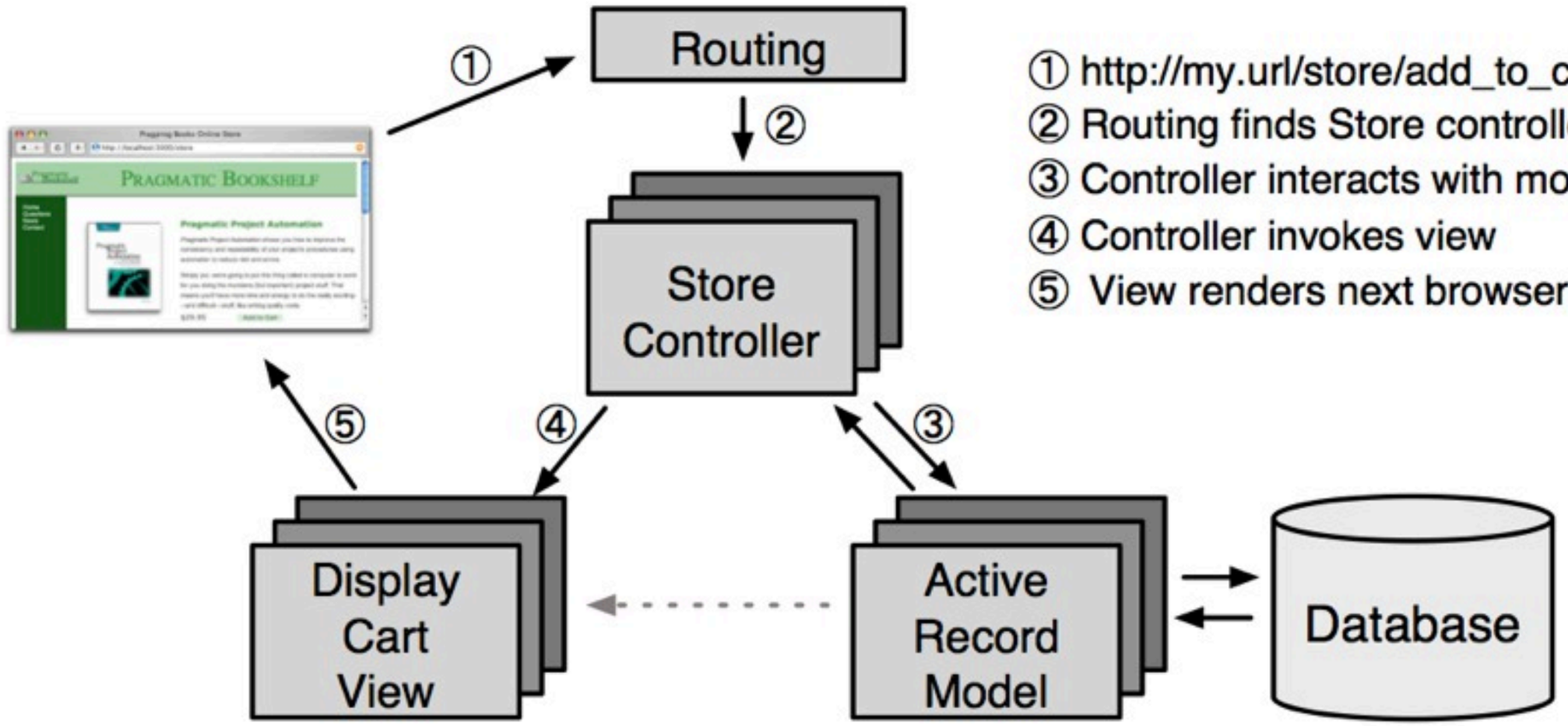
View

- In the default configuration of Rails is an *erb* file. It is typically converted to output html at run-time.

Ruby embedded instead of PHP

```
<div class="headline"><h2>101companies Ruby on Rails Web App</h2></
div>
<div class="content">
<h1>Listing companies</h1>
  <table>
    <tr>
      <th>Name</th>
      <th></th>
      <th></th>
      <th></th>
    </tr>
    <% @companies.each do |company| %>
      <tr>
        <td><%= company.name %></td>
        <td><%= link_to 'Show', company %></td>
        <td><%= link_to 'Edit', edit_company_path(company) %></td>
        <td><%= link_to 'Destroy', company, :confirm => 'Are you
          sure?', :method => :delete %></td>
      </tr>
    <% end %>
  </table>
  <br />
<%= link_to 'New Company', new_company_path %>
</div>
```

The Rails MVC



① http://my.url/store/add_to_cart/123
② Routing finds Store controller
③ Controller interacts with model
④ Controller invokes view
⑤ View renders next browser screen

DEMO

101implementation: rubyonrails

- Remember: previous demo was ‘black box’ (REST).
- Look into internals this time.
- Show details of MVC in this implementation.
- Explain some bits of Ruby on Rails philosophy.

config/routes.rb

```
Companies::Application.routes.draw do
  resources :employees

  resources :departments do
    resources :departments
    resources :employees
  end

resources :companies do
     resources :departments
end

get "home/index"
```

http://exampe.com/employees/

http://exampe.com/departments/

http://exampe.com/departments/:id

http://exampe.com/departments/:id/employees

http://exampe.com/companies/:id/departments

http://exampe.com/ (HTTP GET only)

rake routes

```
GET    /employees(.:format)                {:action=>"index", :controller=>"employees"}
POST    /employees(.:format)               {:action=>"create", :controller=>"employees"}
GET    /employees/new(.:format)            {:action=>"new", :controller=>"employees"}
GET    /employees/:id/edit(.:format)       {:action=>"edit", :controller=>"employees"}
GET    /employees/:id(.:format)             {:action=>"show", :controller=>"employees"}
PUT    /employees/:id(.:format)             {:action=>"update", :controller=>"employees"}
DELETE /employees/:id(.:format)            {:action=>"destroy", :controller=>"employees"}
```

departments_controller.rb

```
# GET /departments/1
# GET /departments/1.json
def show
  @department = Department.find(params[:id])

  respond_to do |format|
    format.html # show.html.erb
    format.json { render :json => @department }
  end
end
```

HTTP params

JSON/HTML

view file

view/departments/show.rb

```
<div class="headline"><h2>101companies Ruby on Rails Web App</h2></div>
<div class="content">
  <p id="notice"><%= notice %></p>
  <div class="attr">
        <p>
              <b>Name:</b>
              <%= @department.name %>
        </p>
  </div>
  <hr>
  <div class="attr">
        <p>
              <b>Manager:</b>
        </p>
        <% @department.employees.each do |employee| %>
              <p>
              <%=
                    if (employee.isManager?)
                          link_to employee.name, employee_path(employee)
                    end
              %>
              </p>
        <% end %>
.....
```

Summary

You learned ...

- how to use MVC design pattern to structure your web applications,
- how Ruby on Rails helps to build web applications using MVC and REST.

Resources

- http://guides.rubyonrails.org/index.html
- Agile Web Development with Rails (4th edition: http://pragprog.com/book/rails4/agile-web-development-with-rails)