Web Engineering:
Hello Rails!

The University of Aizu
Quarter 2, AY 2018

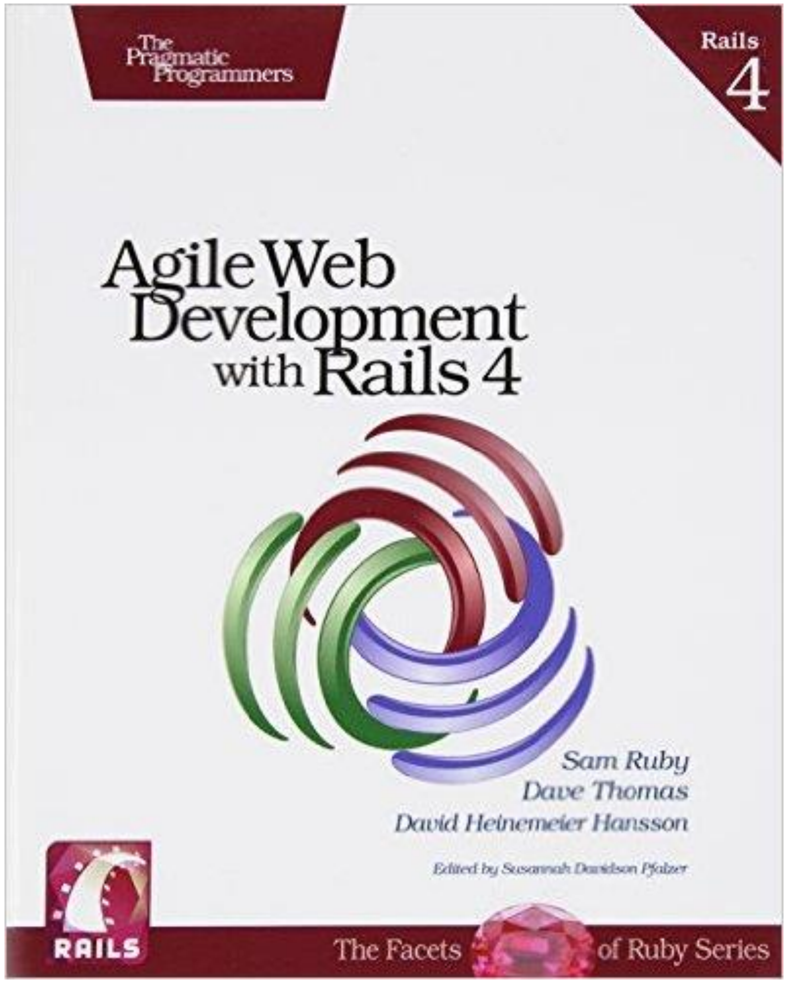
Outline

- Introduction to Ruby on Rails
- First Application: Hello Rails!
- Model-View-Controller Architecture
- Applications with Dynamic Content
- Applications with Several Pages
- Conclusion

Literature

- Agile Web Development with Rails 4 (1st edition) by Sam Ruby, Dave Thomas and Devid Hansson, The Pragmatic Bookshelf, 2013.
- Web resources:

http://www.buildingwebapps.com/


The Pragmatic Programmers
Rails 4
Agile Web Development with Rails 4
Sam Ruby
Dave Thomas
David Heinemeier Hansson
Edited by Susannah Davidson Pfalzer
RAILS
The Facets of Ruby Series

Introduction to Ruby on Rails

- Ruby on Rails is a framework that makes it easier to
  - develop,
  - deploy, and
  - maintain web applications.
- A large number of developers were frustrated with the technologies they were using to create web applications.
  - Technologies such as Java, PHP, and .NET.

Introduction to Ruby on Rails

- All Rails applications are implemented using the Model-View-Controller Architecture.
- In Rails, any developer starts with a working application!
- Rails automatically creates test stubs for new functionality
- Rails applications are written in Ruby, a modern object-oriented scripting language.

Introduction to Ruby on Rails

- Rails code is short and readable: It is DRY, that stands for *do not repeat yourself*. Every piece of knowledge in the system is expressed in one place.
- Convention feature of Rails: There are sensible details for just about every aspectof knitting together application.
- Rails was extracted from a real-word commercial application. It turns out that the best way to create a framework is to find the central themes in a specific application and then bottle them up in a generic foundation of code.

Introduction to Ruby on Rails

- Agility is a part of the fabric of Rails. Rails support 4 preferences of the Agile Manifesto:
  - Individuals and interactionsover processes and tools.
  - Working software over comprehensive documentation.
  - Customer collaboration over contract negotiation.
  - Responding to change over following a paln.

Installing Rails

- Available for MacOSX, Linux, Windows
- Easy to install
  - uses *RubyGems packaging system*

Creating a New Application

- Rails does the groundwork for you
  - sets up default directory structure
  - everything where it expects to see it ("convention over configuration")
  - templates or skeletons for standard files
  - *nothing magic - just Ruby code (you could do the same from scratch... given lots of time...)*

```
rubys> cd work
work> rails new demo --skip-bundle
create
create app/controllers
create app/helpers create app/models . . .
create log/test.log work>
```

Creating a New Application

- File structure...

```
work> cd demo
demo> ls -p
```

- Start with *script* and *app*

Rails Servers

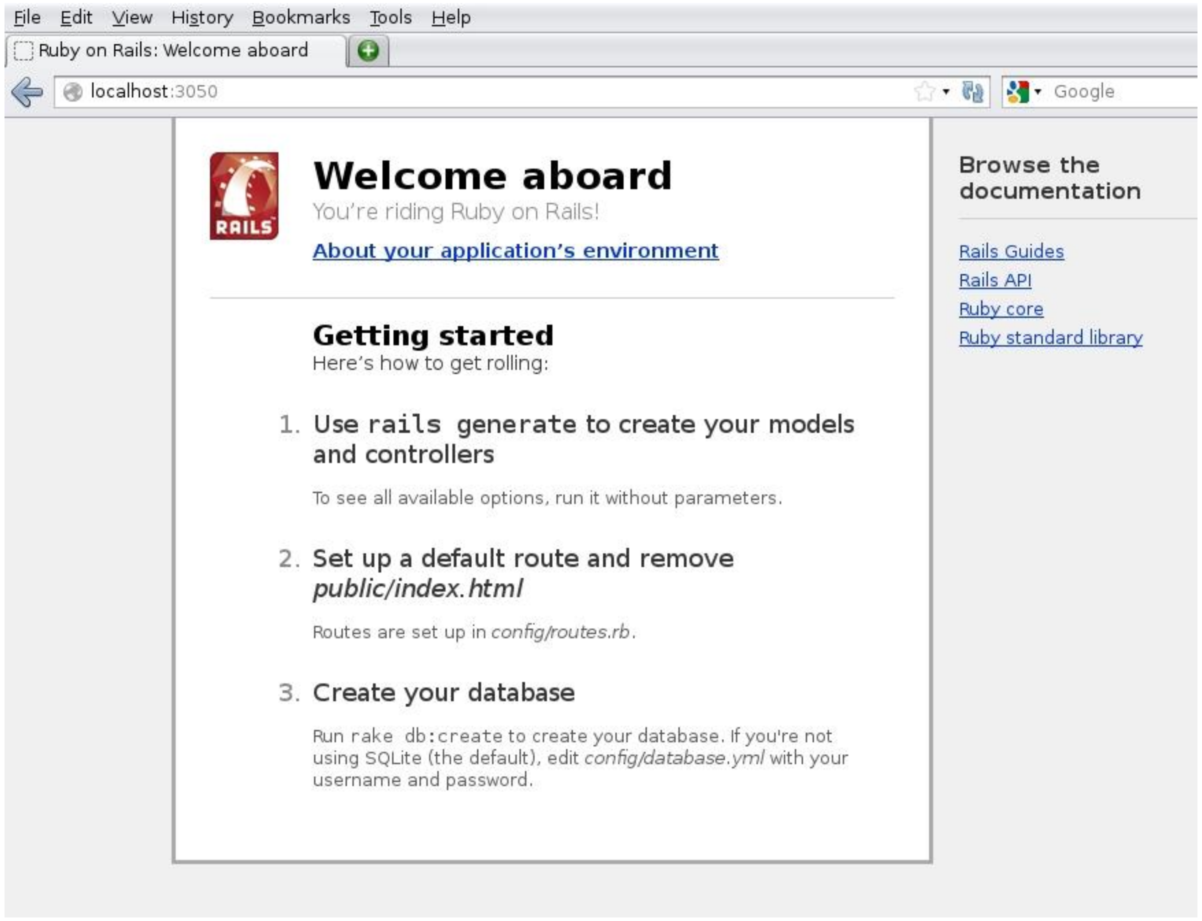
- Rails packaged with two built-in web servers
  - Mongrel
  - WEBrick
- Started by the *server* script

```
/demo>rails server -p3050
=> Booting WEBrick
=> Rails 4.2.4 application starting in development on http://localhost:3050
=> Run 'rails server –h' for more startup options
=> Ctrl-C to shutdown server
[2016-10-17 11:19:22] INFO WEBrick 1.3.1
[2016-10-17 11:19:22] INFO ruby 2.2.3 (2015-08-18) [i686-solaris2.11]
[2016-10-77 11:19:22] INFO WEBrick::HTTPServer#start: pid=11062
    port=3050
```

To stop the server, press Ctrl/C in the window you used to start it.

Welcome Aboard

- Listens on port 3050...


File Edit View History Bookmarks Tools Help
Ruby on Rails: Welcome aboard
localhost:3050
Google

Welcome aboard

You're riding Ruby on Rails!

About your application's environment

Getting started

Here's how to get rolling:

1. Use `rails generate` to create your models and controllers

   To see all available options, run it without parameters.

2. Set up a default route and remove *public/index.html*

   Routes are set up in *config/routes.rb*.

3. Create your database

   Run `rake db:create` to create your database. If you're not using SQLite (the default), edit *config/database.yml* with your username and password.

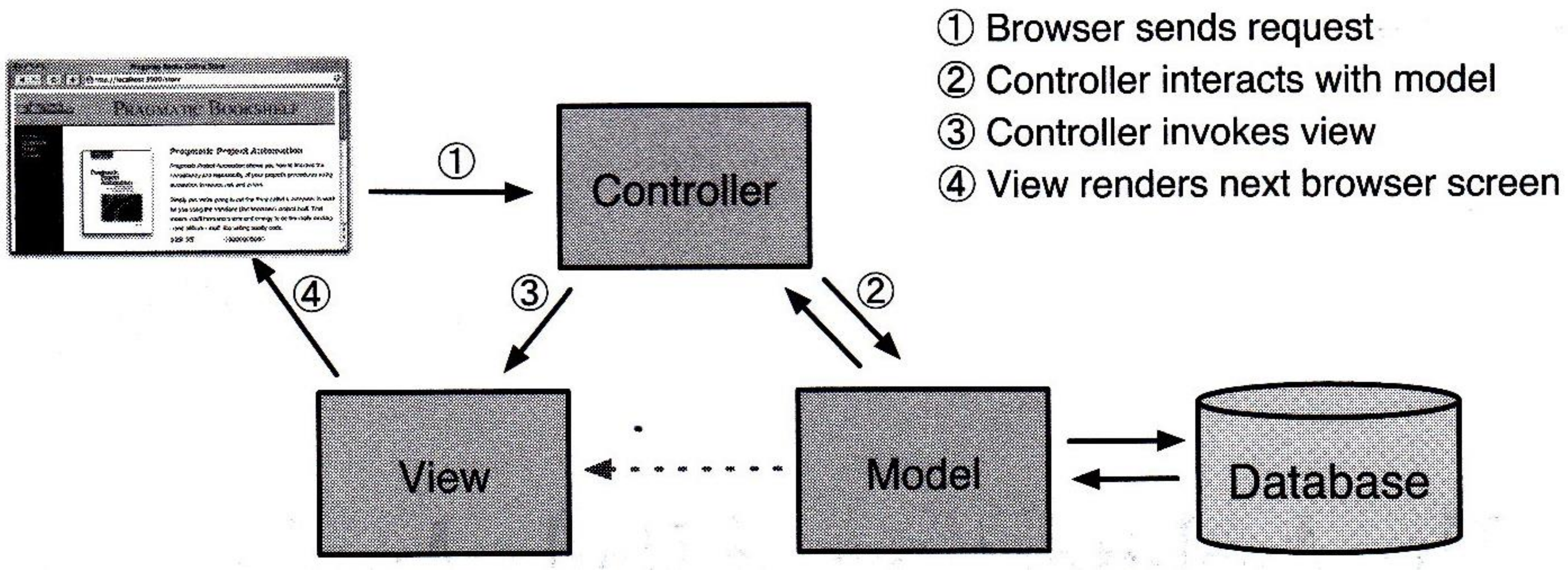
Browse the documentation

Rails Guides
Rails API
Ruby core
Ruby standard library

Hello, Rails! Our First Rails App

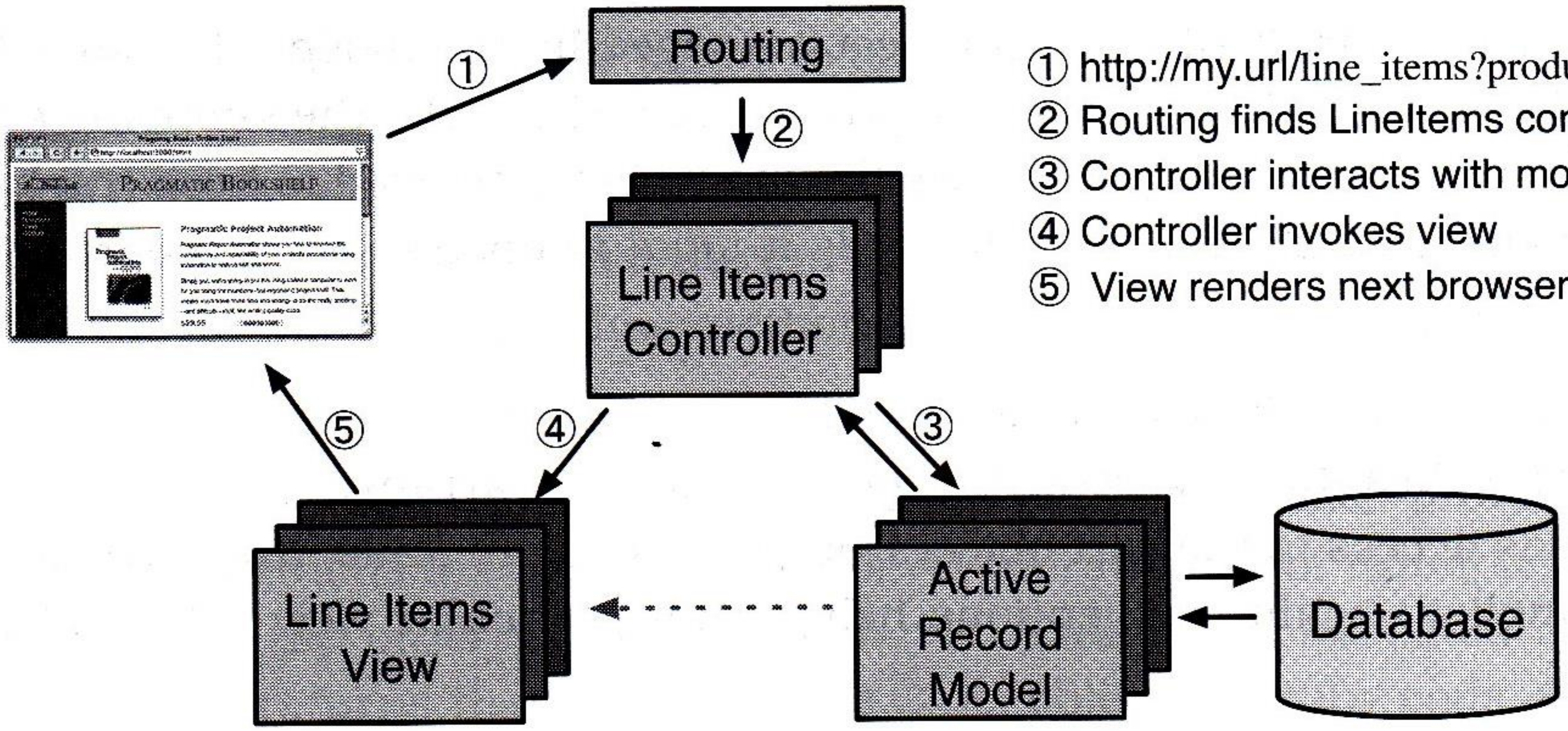
- Rails manifestation…
  - *requests-response cycle*
  - *model-view-controller architecture*
    - *A* model is more than just data; it acts as both a gatekeeper and a data store
    - The *view is responsible for generating a user interface, normally* based on data in the model.
    - *Controllers orchestrate the application. Controllers receive events* from the outside world (normally user input), interact with the
    - model, and display an appropriate view to the user.

The Model-View-Controller Architecture



① Browser sends request
② Controller interacts with model
③ Controller invokes view
④ View renders next browser screen

Rails and MVC



① http://my.url/line_items?product_id=2
② Routing finds LineItems controller
③ Controller interacts with model
④ Controller invokes view
⑤ View renders next browser screen

Hello, Rails! Our First Rails App

- Hello, World! app has no data → no model
  - only need controller and view
- Rails works as follows:
  - Accept request from browser
  - Rails decodes request to find appropriate controller
  - calls an action method in controller, and invokes a view
  - passed back to browser to display to user
  - ➞ Rails takes care of "internal plumbing"

Controller

- *generate* script creates controllers
  - to generate a controller called "say" with the name of the action *hello*...

```
demo> rails generate controller say hello
```

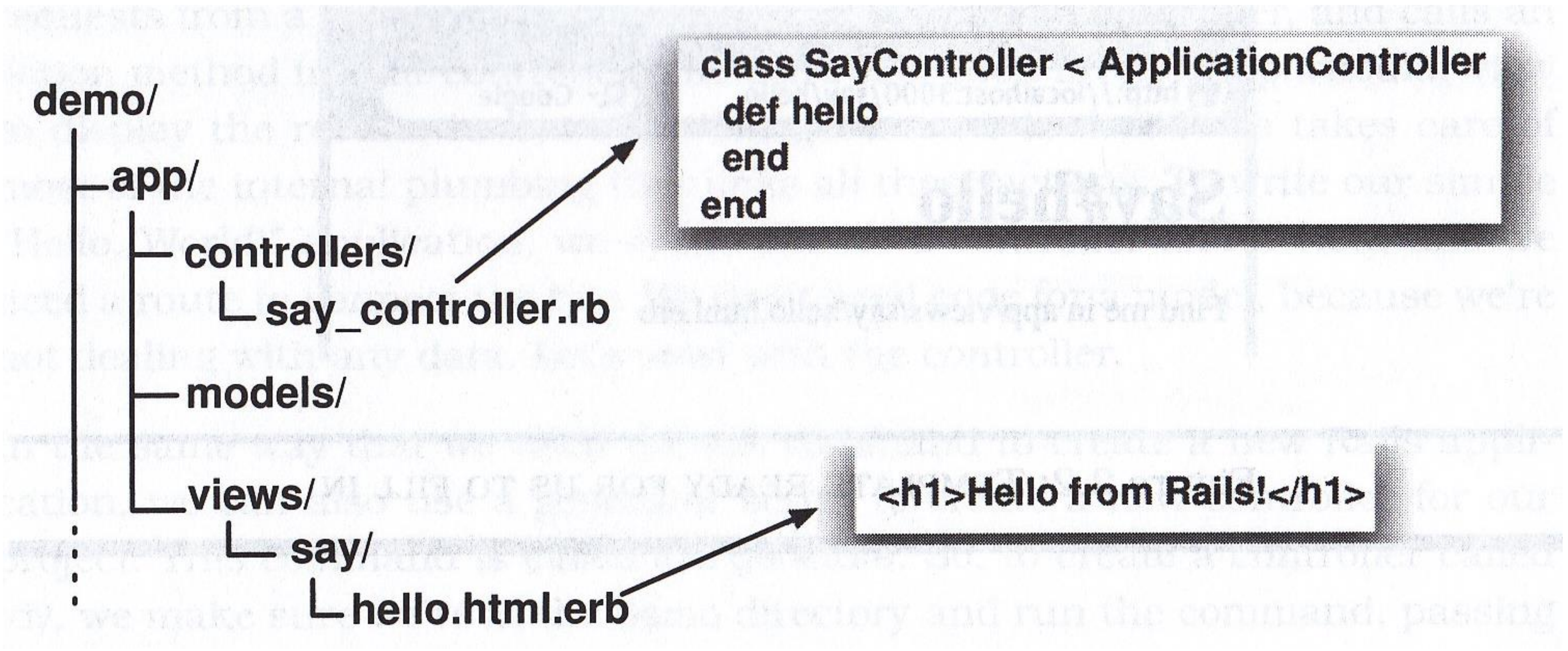
  - You will see the protocol

Controller

- Controller's job to set things up so view knows what information (data, calculations, etc) to display
  - in this case, nothing to set up

```
class SayController < ApplicationController
  def hello
  end
end
```

Standard Location for Controllers and Views



```
class SayController < ApplicationController
  def hello
  end
end
```

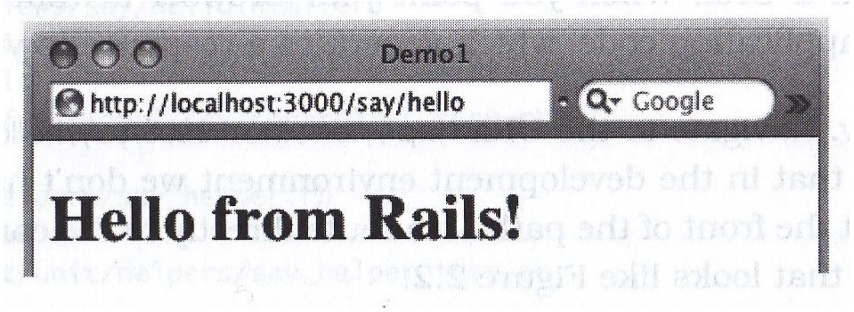
```
<h1>Hello from Rails!</h1>
```

View

- Recall *generate* also created a directory *app/views/say*
  - will contain views for controller *Say*
  - default: same name as controller, with extension *.html.erb*
- app/views/say/hello.html.erb (the code below is the replacement of the original code)

```
<h1>Hello from Rails!</h1>
```

After reloading page, you will see…



Hello from Rails!

Dynamic Content

- The power of server-side programming comes from being able to add *dynamically generated content*
- Two ways in RoR
  - *builder templates*
  - *embedded Ruby code* (we will focus on this way)
- .erb files preprocessed for embedded Ruby
- *ERb (embedded Ruby) filter*
  - content between <%= .... %> interpreted as Ruby code and executed
    - result converted to string and substituted
  - content between <% .... %> interpreted but not substituted

Embedded Ruby

- Example:

```
<ul>
<li>Addition: <%= 1+2 %> </li>
<li>Concatenation: <%= "cow" + "boy" %> </li>
<li>Time in one hour: <%= 1.hour.from_now %> </li>
</ul>
```

- Result

```
Addition: 3
Concatenation: cowboy
Time in one hour: Tue Oct 23 11:30:32 +0900 2012
```

Embedded Ruby

- Can be intermixed with non-Ruby code

```
<% 3.times do %> Ho!<br /> <% end %> Merry Christmas!
        Ho!<br />
        Ho!<br />
        Ho!<br />
        Merry Christmas!
```

  - Note: there are newline characters in the loop that can be removed by using <% .... -%>

Embedded Ruby

- Substitution

```
<% 3.downto(1) do |count| -%> <%= count %>...<br />
<% end -%>
Lift off!
3...<br /> 2...<br /> 1...<br /> Lift off!
```

- XHTML character substitution

```
Email: <%= h("Ann & Bill <frazers@isp.email>") %>
Email: Ann &amp; Bill &lt;frazers@isp.email&gt;
```

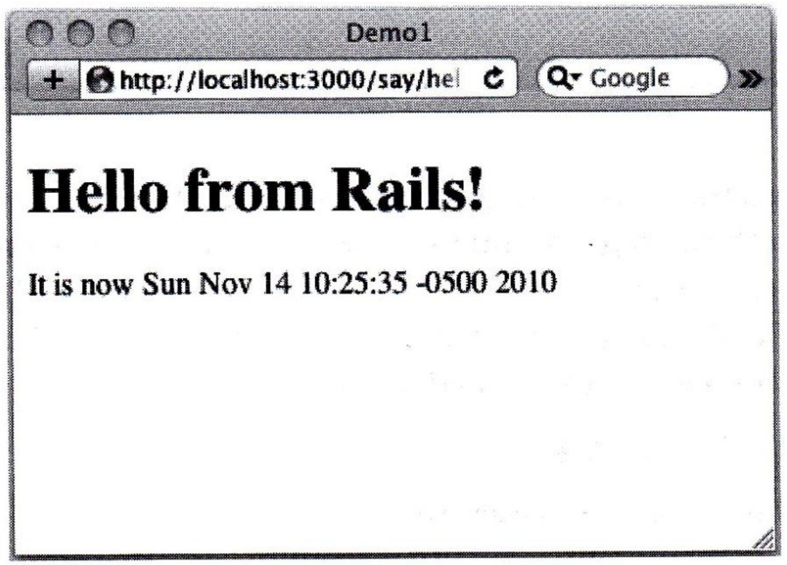
Embedding Results from Controller

- *say_controller.rb* (in the directory *app/controllers*)

```
class SayController < ApplicationController
  def hello
        @time = Time.now
  end
end
```

- *hello.html.end* (in the directory *app/views/say*)

```
<h1>Hello from Rails!</h1>
<p>It is now <%= @time %></p>
```



Hello from Rails!

It is now Sun Nov 14 10:25:35 -0500 2010

Linking Pages Together

- Assume we had a second page, *goodbye.html.erb* in the *app/view/say* directory, its content is below

```
<h1>Goodbye!</h1>
<p>It was nice having you here.</p>
```

- See the next slide

Linking Pages Together

- and our controller...

```
class SayController < ApplicationController
  def hello
        @time = Time.now
  end
  def goodbye
  end
end
```

Linking Pages Together

- We could link the pages using relative addresses:

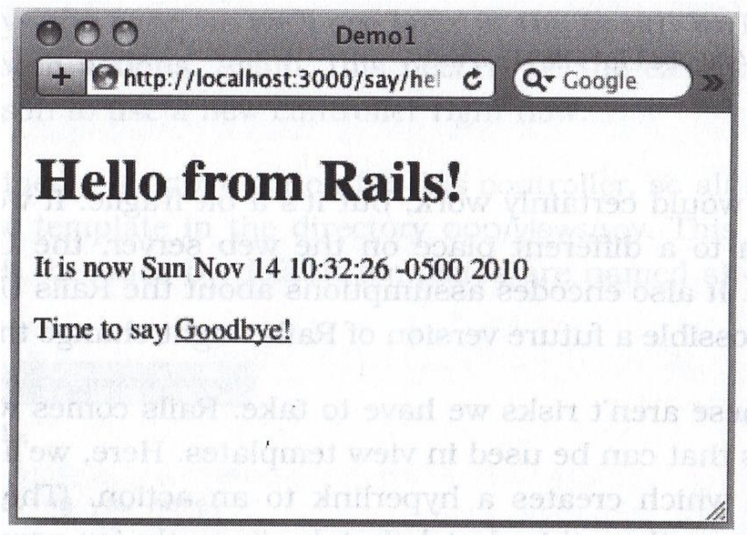
```
<p>
Say <a href="/say/goodbye">Goodbye</a>!
</p>
```

- but its brittle
  - move application to different location in deployment on web server
  - assumptions about Rails URLs which might change
    - “hard wired”
- → should always link to the *action that generates the output, not* the output

Linking Pages Together

- *helper method: link_to*
  - creates hyperlink via the *action that generates the page*
  - Code for hello.html.erb in the directory *app/views/say*

```
<h1>Hello from Rails!</h1>
<p>It is now <%= @time %></p>
<p>Time to say
<%= link_to "Goodbye", say_goodbye_path %>!
</p>
```



Hello from Rails!

It is now Sun Nov 14 10:32:26 -0500 2010

Time to say Goodbye!

Conclusion

- We have seen:
  - how to create a Rails application
    - skeleton structure, controller, views
  - how Rails maps incoming requests to methods in code
  - how to create dynamic content in controller and display it via view template
  - how to link pages together