Advanced Web Programming

Yuan Wang

2019 Spring

Lecture 9

Ruby - Writing web applications - RECAP

Different ways to implement:

- CGI script manually

- Using Ruby CGI standard library

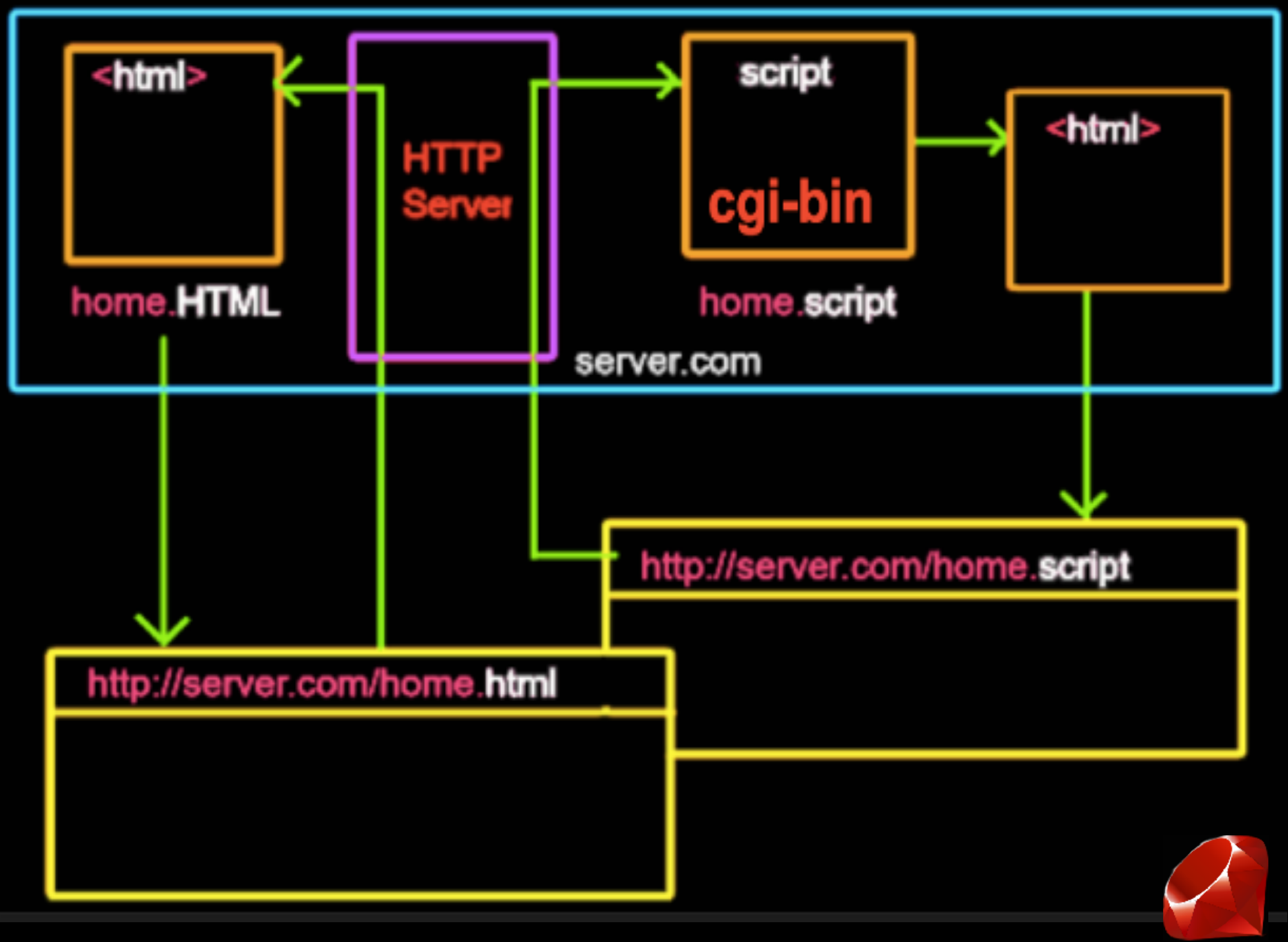
- Template systems (replace PHP)

- Frameworks

Ruby - Writing web applications - RECAP

- CGI script manually

web server calls standalone program to produce web page

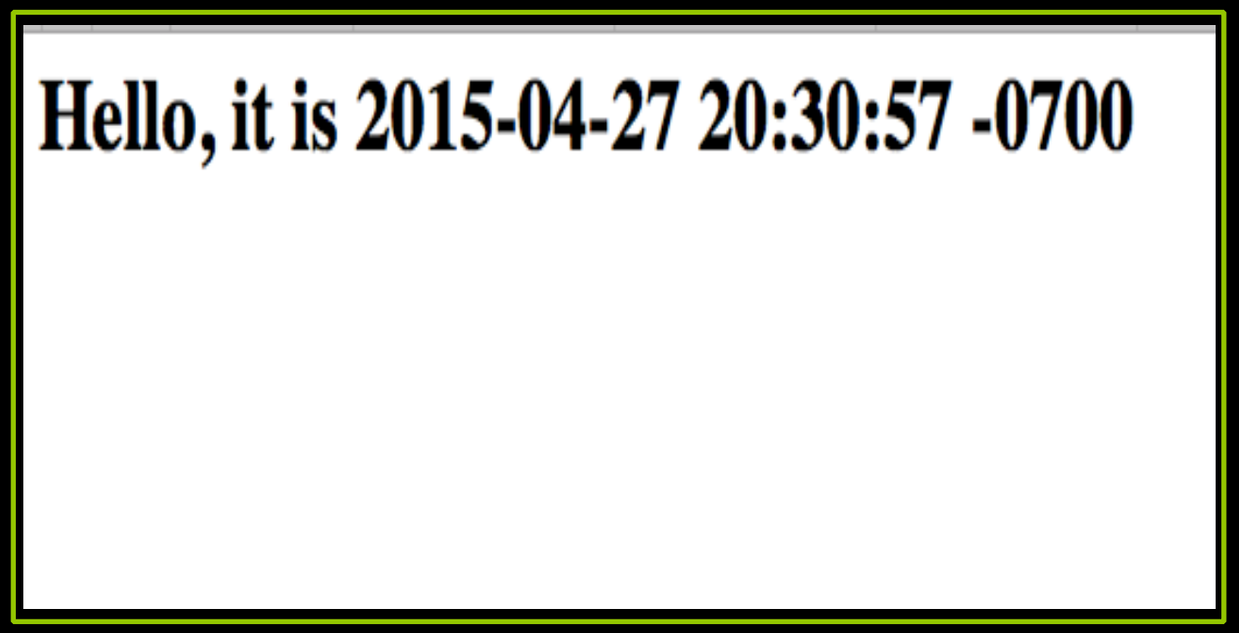


Ruby - Writing web applications -RECAP

- CGI script - to produce all response text manually

Example - a CGI program to send a simple html document to browser

```
#!/usr/bin/ruby
str = <<HTMLSTR
<html>
<head>
</head>
<body>
<h1>Hello, it is #{Time.now}</h1>
</body>
</html>
HTMLSTR

print "HTTP/1.1 200 OK"
print "Content-Type: text/html;charset=UTF-8\n"
print "Content-Length: #{str.size}\n"
print "Connection: close\n\n"

print str
```



Ruby - Writing web applications -RECAP

- Using Ruby CGI standard library

Example - to handle the same form submit

```
#!/usr/bin/ruby
require 'cgi'

cgi = CGI.new('html5')

if cgi.params['name'] != ""
  p1 = cgi.params['name']
else p1  = "name is empty"
end

if !cgi.params['reason'].empty?
  p2 = cgi.params['reason']
else p2 = "reason is empty"
end

#continue in green box
```

```
cgi.out do
  cgi.html do
    cgi.head {cgi.title {"this is a cgi program"}} +
      cgi.body do
      cgi.h1 {"your submit from the form are:"} +
    cgi.p {p1} +
    cgi.p {p2}
    end
  end
end
```

your submit from the form are:

["yuan"]

["popular", "fun"]

Ruby - Writing web applications -RECAP

- Template systems

ERB - Embedded Ruby

```
<% ruby code %>                 # executed
<%= ruby expression %>          # substitute with
                                 # value of expression

<% #comment %>                  # comment
% ruby code                     #  the whole line is ruby code
```

```
<ul>
  <%# This is just a comment %>
  <% for @item in @shopping_list %>
    <li><%= @item %></li>
     <% end %>
</ul>
```

Ruby - Writing web applications

- Template systems

HAML (HTML Abstract Markup Language)

ERB

```
<div id='content'>
  <div class='left column'>
    <h2>Welcome to our site!</h2>
    <p><%= print_information %></p>
  </div>
  <div class="right column">
    <%= render :partial => "sidebar" %>
  </div>
</div>
```

Haml

```
#content
  .left.column
    %h2 Welcome to our
site!
    %p= print_information
  .right.column
    = render :partial =>
“sidebar"
```

Ruby - Writing web applications

Note:

HAML is like a specially purpose language (as oppose to general purpose language) to make your writing web page easier.

It is defined on top of another language, sort of like a SYNTAX SUGAR.

This is called:

DSL - Domain Specific Language.

Today's topic:

Frameworks - Sinatra

Ruby - Frameworks

HTTP protocol revisit:

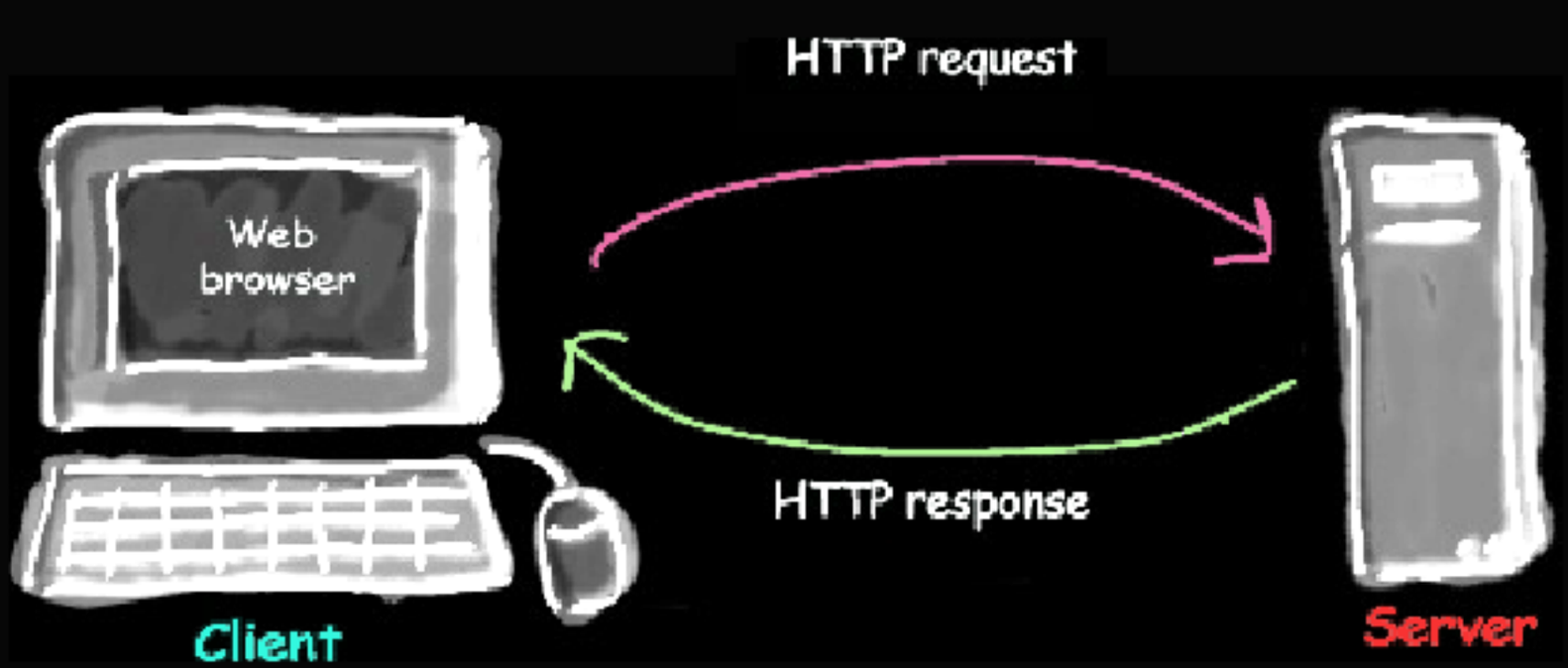
web application, is all about requesting an action to server through URL

different actions verbs

POST, PUT, DELETE...

GET http//mysite.com

```
 GET /hello.htm HTTP/1.1
User-Agent: Mozilla/4.0
(compatible; MSIE5.01; Windows NT)
Host: www.tutorialspoint.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
 Connection: Keep-Alive
```



HTTP/1.1 200 OK

```
   HTTP/1.1 200 OK
 Date: Mon, 27 Jul 2009 12:28:53
GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 22 Jul 2009
19:15:56 GMT
Content-Length: 88
Content-Type: text/html
Connection: Closed
<html>
<body>
<h1>Hello, World!</h1>
</body>
 </html>
```

and returning, by the server, a html page caused by the action in request

Ruby - Frameworks

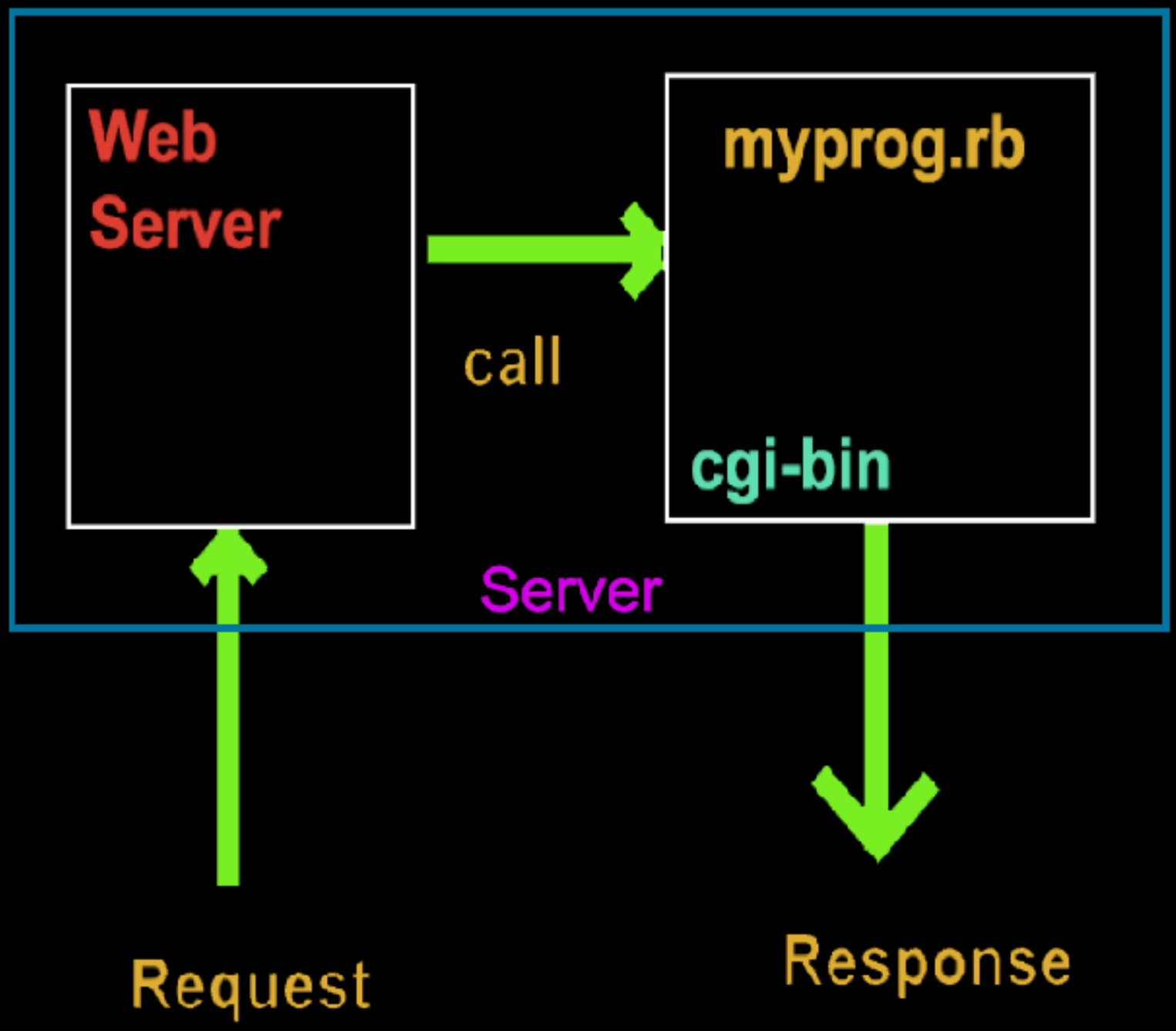
So far, we can write Ruby CGI script (either use just Ruby or use Ruby with CGI library) under Apache web server.

We can then call this script from Web Browser.

The reason we can ask Apache web server to execute our script, is because we configure Apache so that executing our Ruby CGI script is enabled.

Ruby - Frameworks

CGI diagram is like this:



Ruby - Frameworks

So a web application is like this structure (in concept):

```
case request
   when GET page1 then  page1

   when GET page2 then page2

   when POST page3 then page3
   …
   when DELETE page_n then delete page_n
end
```

it structures
"request"/"response"
in an organized way

This is called "routing"

Ruby - Frameworks

The problem with that is:

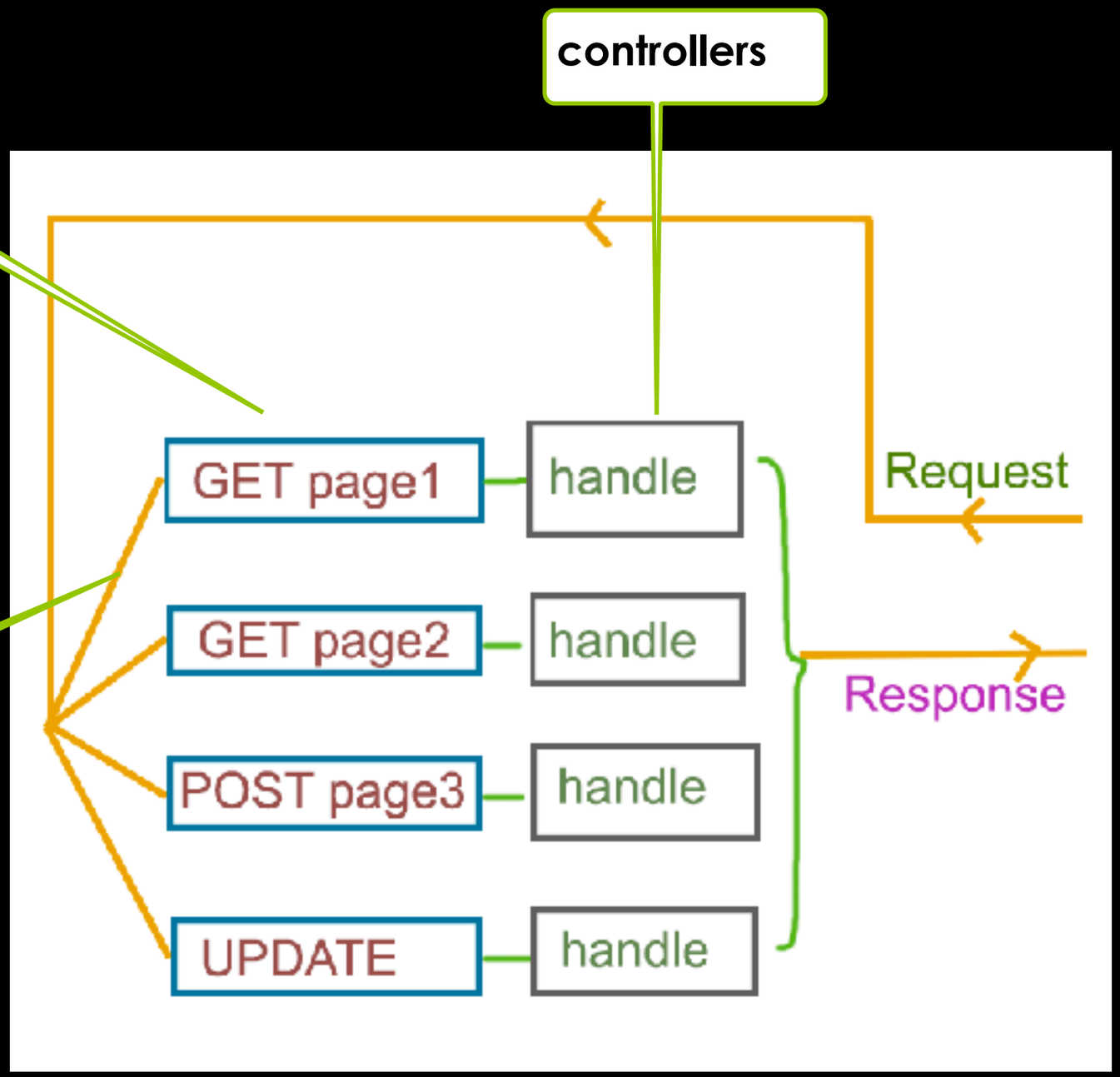
our CGI program sometimes need to handle HTTP details, or need to do some laborious coding.

Our program need to be taking different requesting actions and returning corresponding pages in a more easy and organized way

Ruby - Frameworks

to draw in a diagram, it is like this:

it structures
"request"/"response"
in an organized way



This is called "routing"

routing is fundamental to how well a web application functions, it's the ability to map incoming requests to methods (for returning response) inside your code

Ruby - Frameworks

Frameworks come to the rescue.

They are systems that wrap low-level things up

for example:

- mapping url to your script
- interact with database
- provide an easier way generate html.

Ruby - Frameworks

Framework will provide a way for you to:
- organize your request-response mapping
- define views (templates) for dynamic page generation
- interact with database

Also it will create a predefined content for you to start.

Depend on the definition of Frameworks, different systems provide different level of predefined content.

We start with lightweight framework:

Ruby - Frameworks

Sinatra is a ruby library. It wraps things up to the extend that it provides a

DSL - Domain-Specific Language

Examples of other DSLs:

- **SQL**
- **Unix shell**
- **HAML**

Ruby - Frameworks

DSL

- In Ruby, this is closely related to topics of meta-programming.

- you can design your own DSL, then use your DSL to write program

Ruby - Frameworks

To install Sinatra library

> gem install sinatra

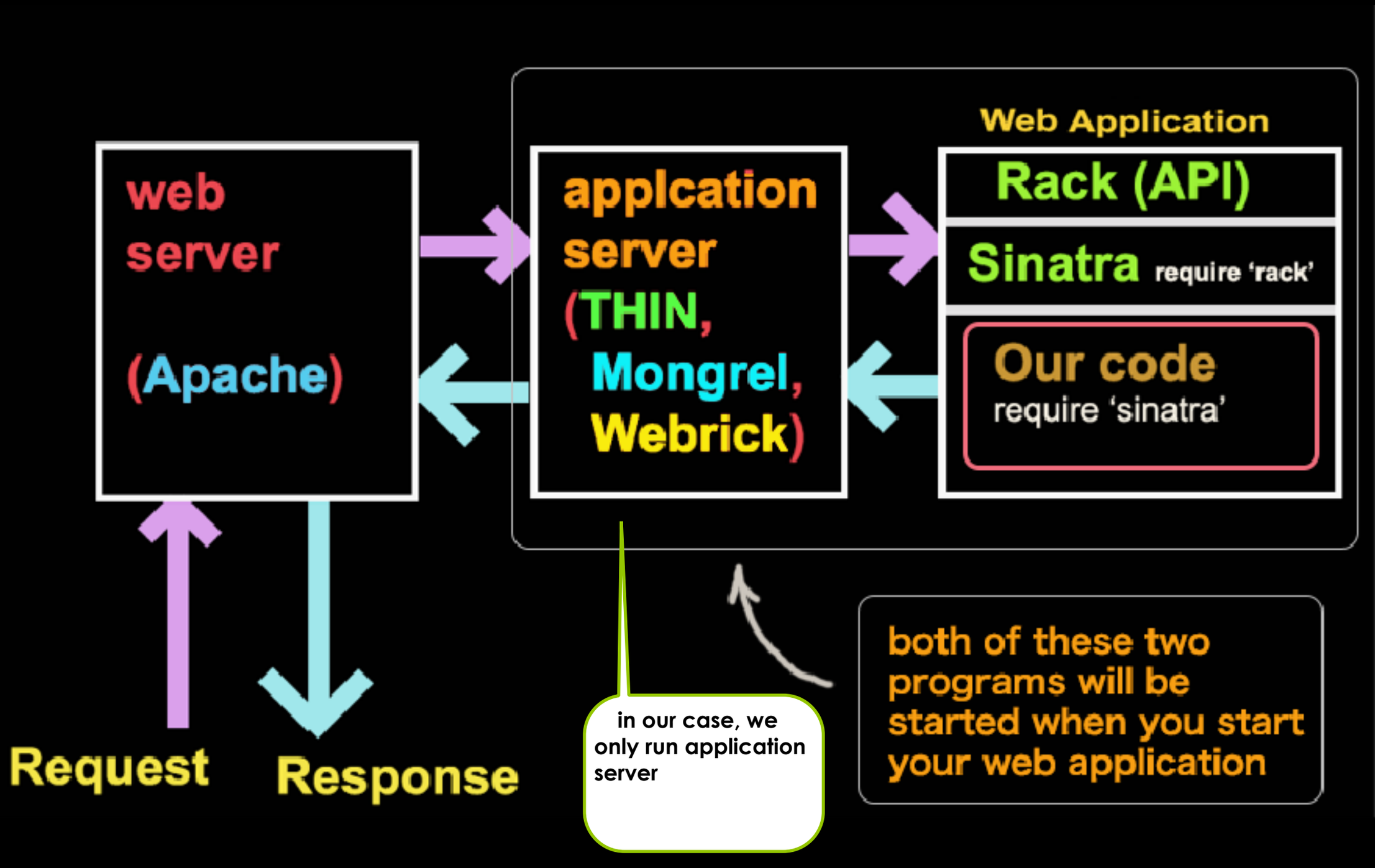
a gem is ruby library;
“gem” is a program of RubyGems;
RubyGem is a package manager (https://rubygems.org/)

“gem install” is a very common method of installing Ruby libraries, like this:

> gem install library_name

Ruby - Frameworks

Here is the top level structure of using Sinatra to build web applications



Ruby - Frameworks

Sinatra is not as "framework" as Ruby on Rails. It provides no file structure and so on, but still allow you to write simple and elegant code.

It is named after Frank Sinatra

Ruby - Frameworks

Example program and the idea of Sinatra:

```
# first.rb
require ‘sinatra’

get ‘/hello’ do
    “<h1>hello, this is my first sinatra web application<h1>”
end
```

This will be our web application. It will be running with web server to serve web document.

Ruby - Frameworks

To run this web application:

```
> ruby first.rb
== Sinatra (v1.4.6) has taken the stage on 4567 for
development with backup from Thin
Thin web server (v1.6.3 codename Protein Powder)
Maximum connections set to 1024
Listening on localhost:4567, CTRL+C to stop

THIN web server will be started
```

Ruby - Frameworks

Note.

THIN is not talking to Apache. you need to configure Apache for it to talk to THIN web server

Sometimes, Apache is called web server
THIN is called application server
(see in previous slide)

For development (as oppose to production), it is ok not to configure Apache to work with THIN.
THIN can take care of all web request.

Ruby - Frameworks

To install THIN

> gem install thin

Sinatra will first use THIN web server, if THIN is not installed, it will try Mongrel web server, otherwise, it will try Webrick web server.

THIN, Mongrel, Webrick, are some popular application servers dedicated to Ruby web applications.

Ruby - Frameworks

Now that the THIN server and our web application is running, we can connect to it from browser:

Ruby - Frameworks

Anatomy of this example program

DSL syntax:

pulls in all the code from sinatra library
you need that in all sinatra applications

Match the request from browser:
"get" request, with url /hello
this is called 'route'

"get" is a method
'/hello' is argument

```
require 'sinatra'

get '/hello' do

  "hello, this is my first sinatra web application"

end
```

Route handler (block),
handle that particular
request from the browser

whatever the return value, will be
the response back to the browser

note: the route here is not routes we used to use like:

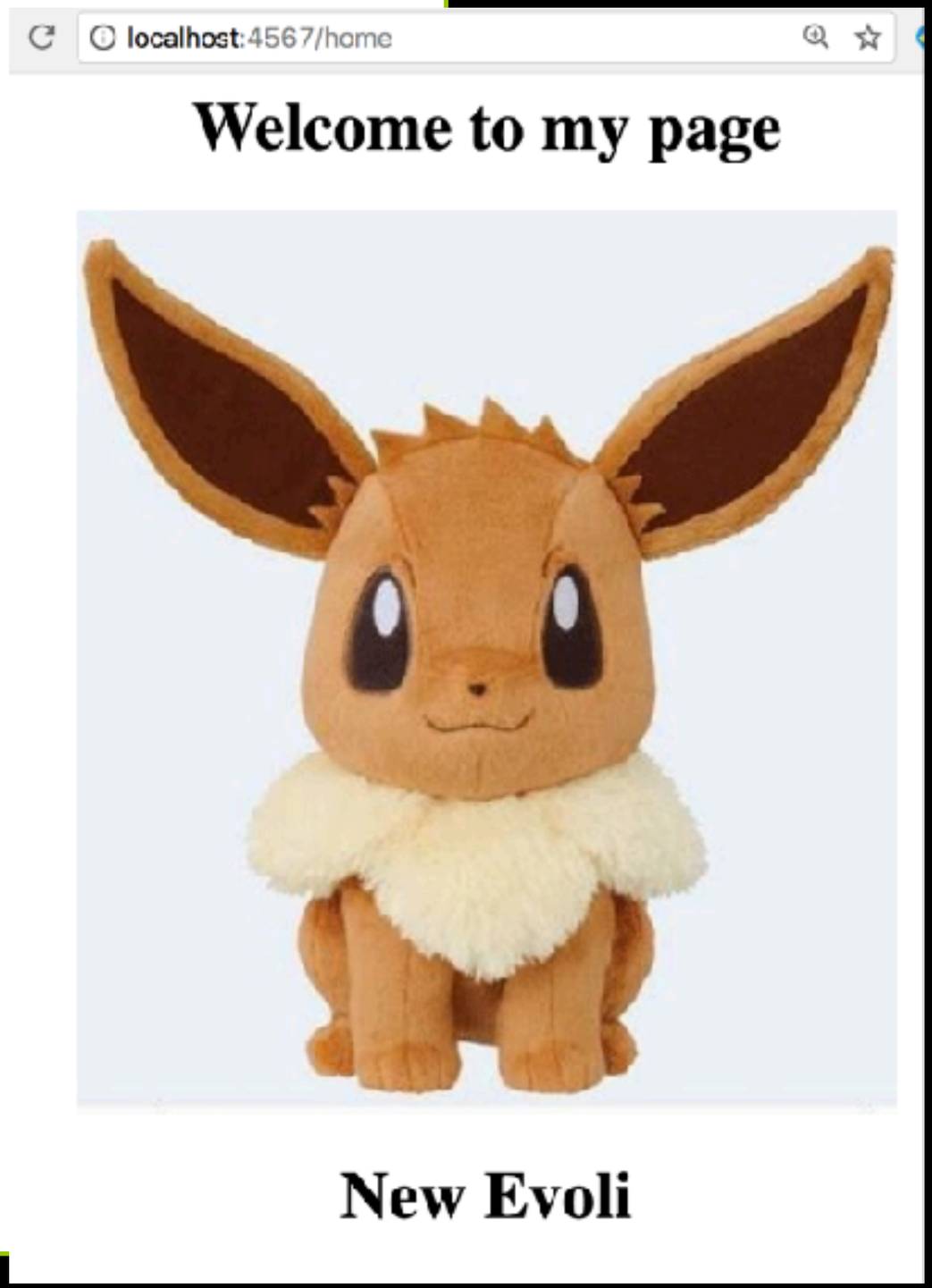
/page.html? x = 2 & y = 3

the route here is called "clean route"

Ruby - Frameworks

Try another one

```
# second.rb
require 'sinatra'

get ‘/home' do
%{<html>
  <body>
    <div style="text-align: center">
      <h1>Welcome to my page</h1>
      <img src="newevoli.jpg" />
      <h1>New Evoli</h1>
    </div>
  </body>
</html>}
end
```



Ruby - Frameworks

“route handler” is like event handler. it is defined to handle different possible URLs and return page

A route handler is in the format of the following method call with block:

```
verb ‘route’ do
    handle the request
    for example, return a page
end
```

Essentially, a sinatra web application is made up of：

list of “route handler”s.

Sinatra’s base class defines handful of methods matching the HTTP verbs, like post, put, delete

Ruby - Frameworks

Sinatra route methods (corresponding to HTTP verbs) are called with ‘route’ being the parameter, and block being the route handler, like these:

```
get ‘/route1‘ do
    #handle route1
end

post ‘/route2‘ do
   #handle route2
end

put ‘/route3‘ do
   #handle route3
end

delete ‘/‘route4 do
   #handle route4
end
```

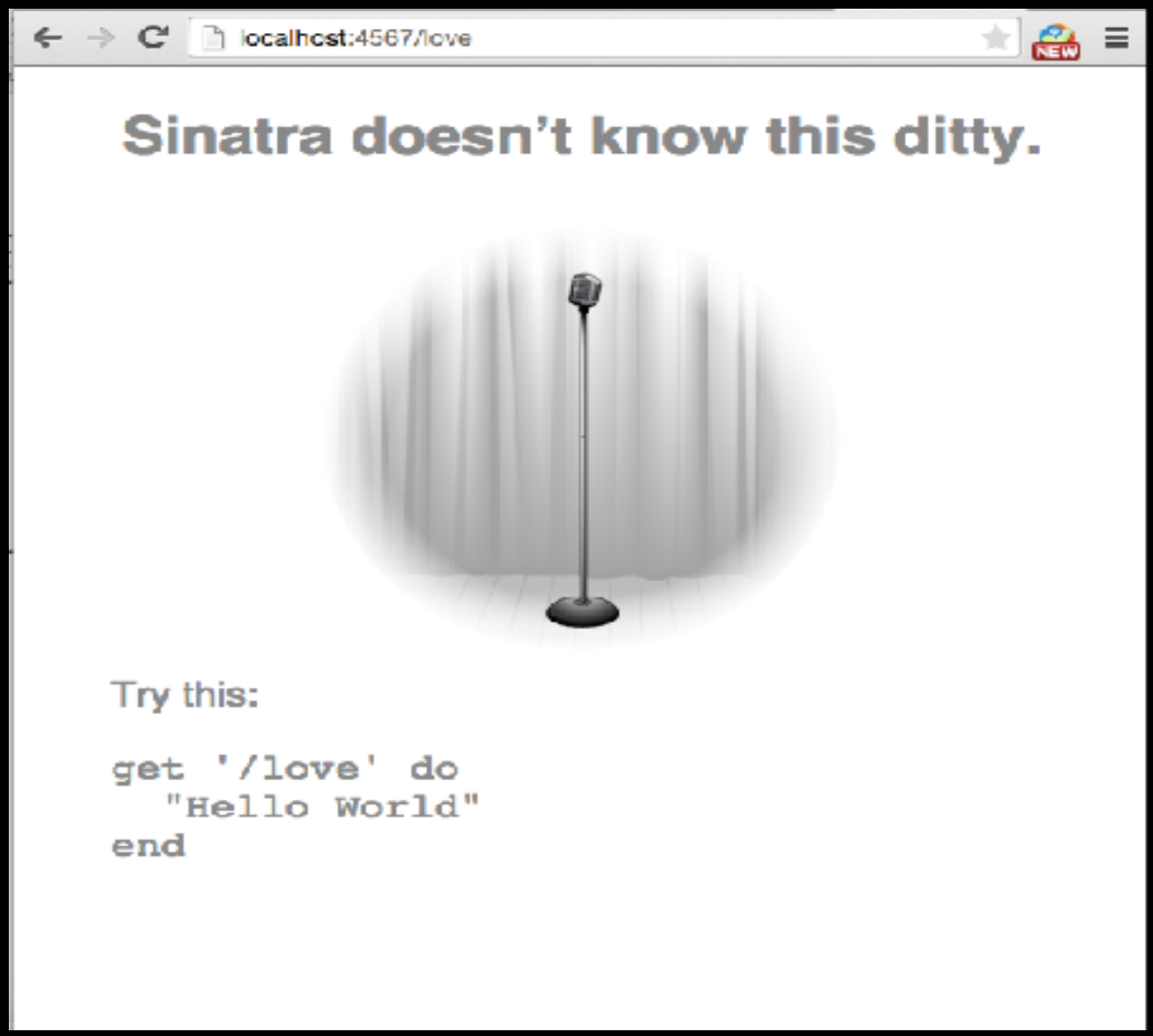
verb TRACE, CONNECT are not supported

Ruby - Frameworks

Sinatra will process the 'route handler' from top to bottom. As soon as it finds a match to the URL, it will execute the handler.

If it cannot find one, it will display some 'hint' info,

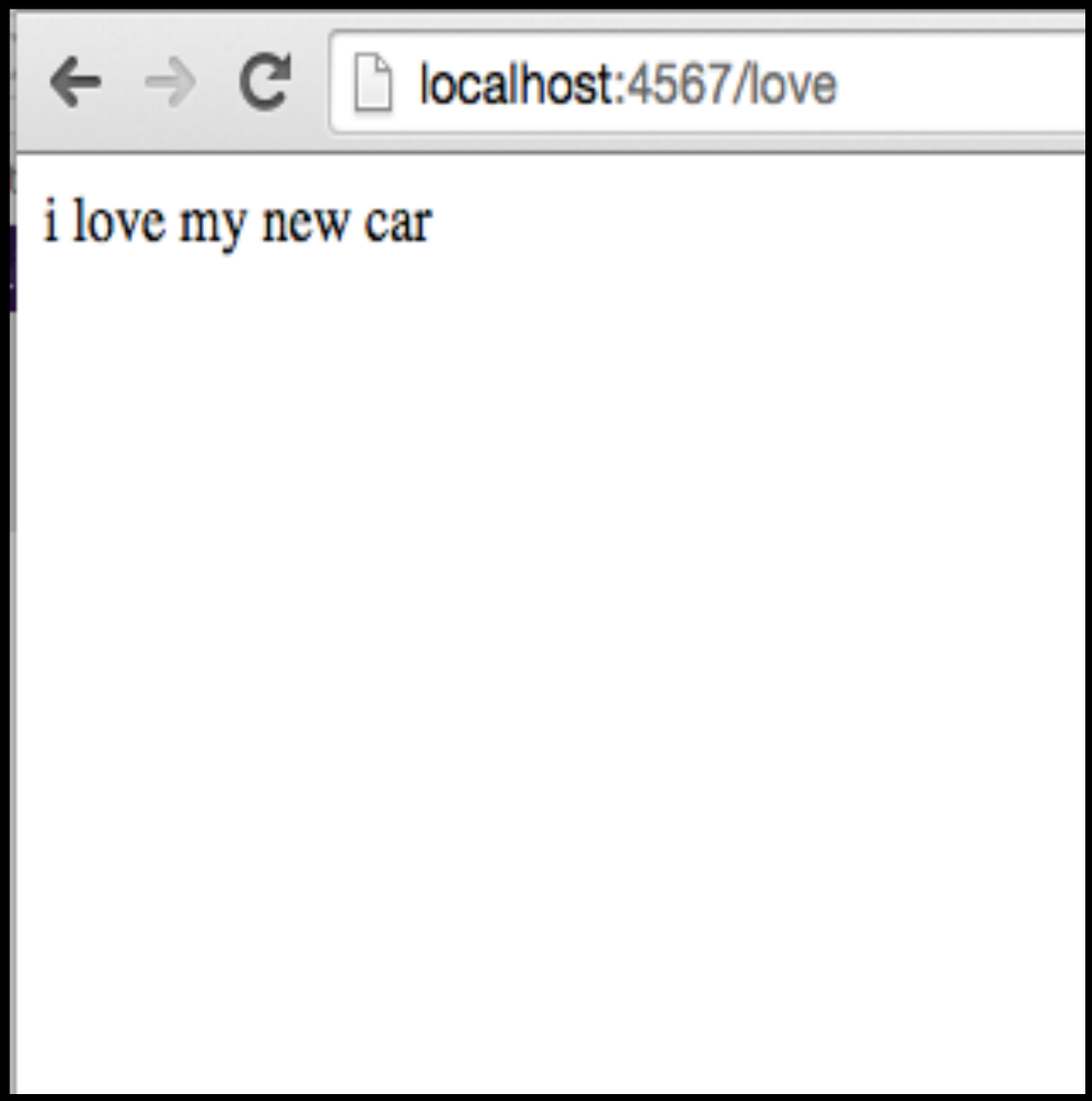
this info is asking you to define a route handler for the unfound route in URL



Ruby - Frameworks

Let's add the missing route /love

```
get '/love' do
    "i love my new car"
end
```



Ruby - Frameworks

Note. You need to restart the server after every change.

if you don't like to restart the server every time you make a change, you can use Sinatra::Reloader (part of Sinatra::Contrib library)

To install sinatra-contrib gem:

> gem install sinatra-contrib

Ruby - Frameworks

Then, add the following line to your file after “require ‘sinatra’”

```
require ‘sinatra’
require ‘sinatra/reloader’ if development?
```

“development?” is a method defined by Sinatra, it will check environment variable and decide if the environment is “development” or “production”, there is a “production?” method too.

the environment variable can be set on command line when you start your Sinatra application or set in the program

Ruby - Frameworks

A crucial question:

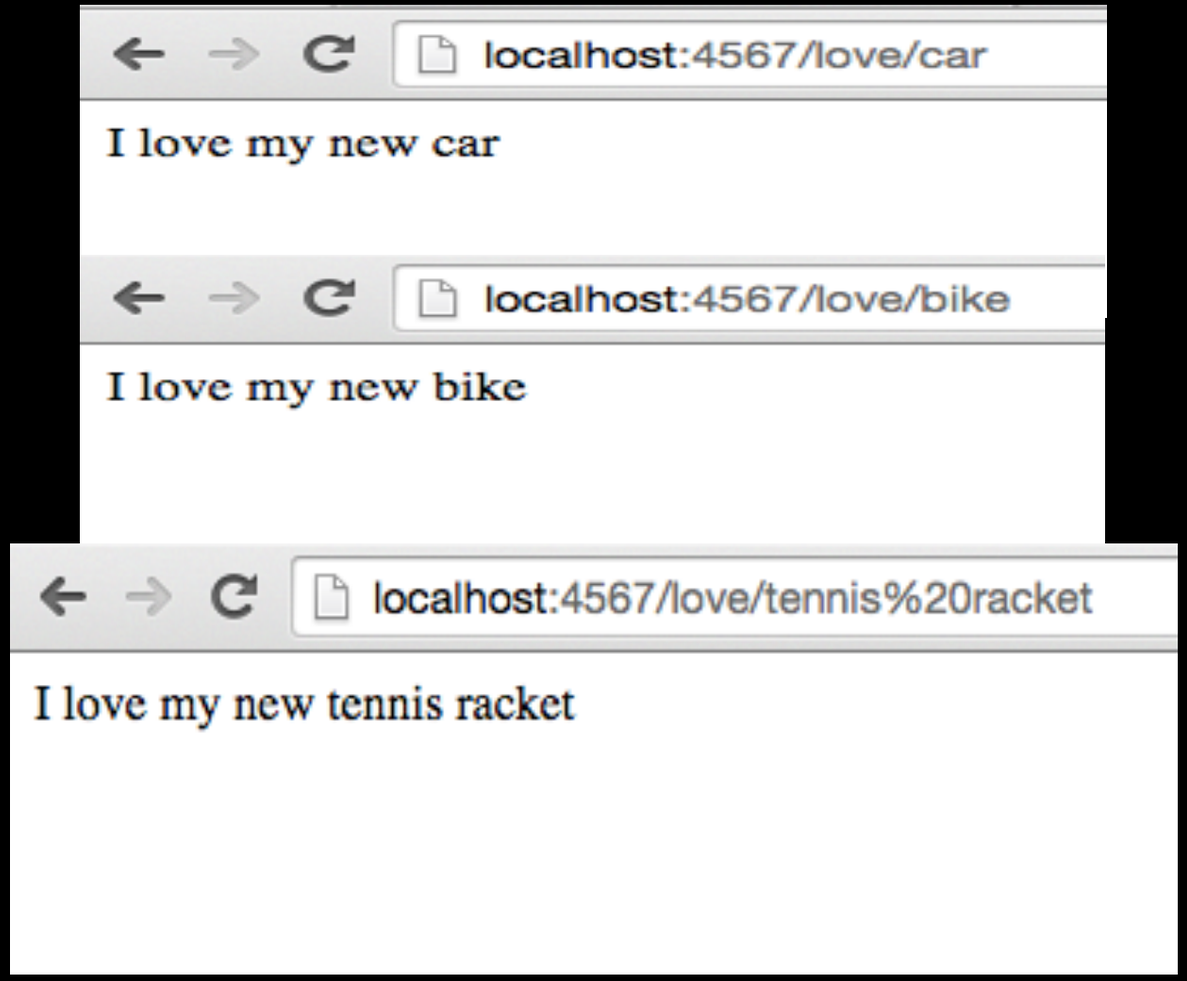
how to pass named parameters (“name=value” data) from browser?

Your route will be using a symbol to specify that it is a key waiting for a value passed from browser:

the value will be available as part of params hash.

example

```
get ‘/love/:name’ do
   name = params[:name]
   “I love my new #{name}”
end
```



Ruby - Frameworks

Pass named parameters from URL.

or via block

```
get '/hello/:name' do |n|
   "Hello #{n}!"
end
```

when match with "GET /hello/april" or "GET /hello/may”, params['name'] will be 'april' or ‘may’, then they are passed to variable ’n’

Ruby - Frameworks

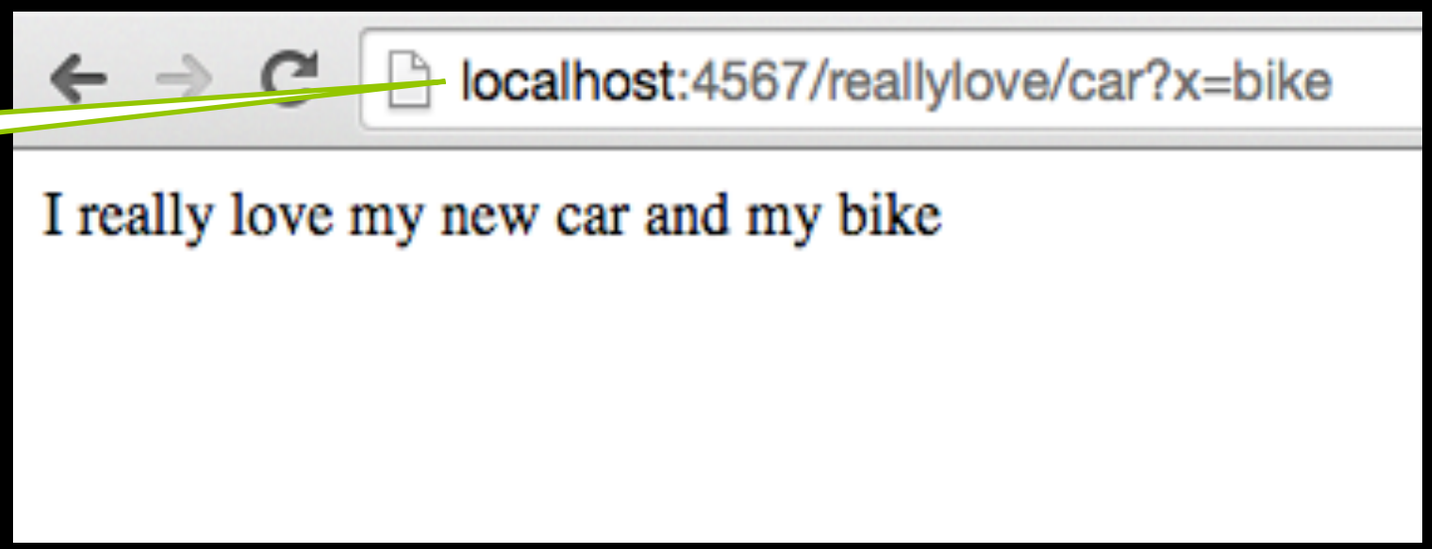
Pass named parameters from URL.

You can also use query string to pass parameters.
the values will also be in params hash.

example

```
get ‘/reallylove/:name’ do
  “I really love my new #{params[:name]} and
    my #{params[:x]}”
end
```

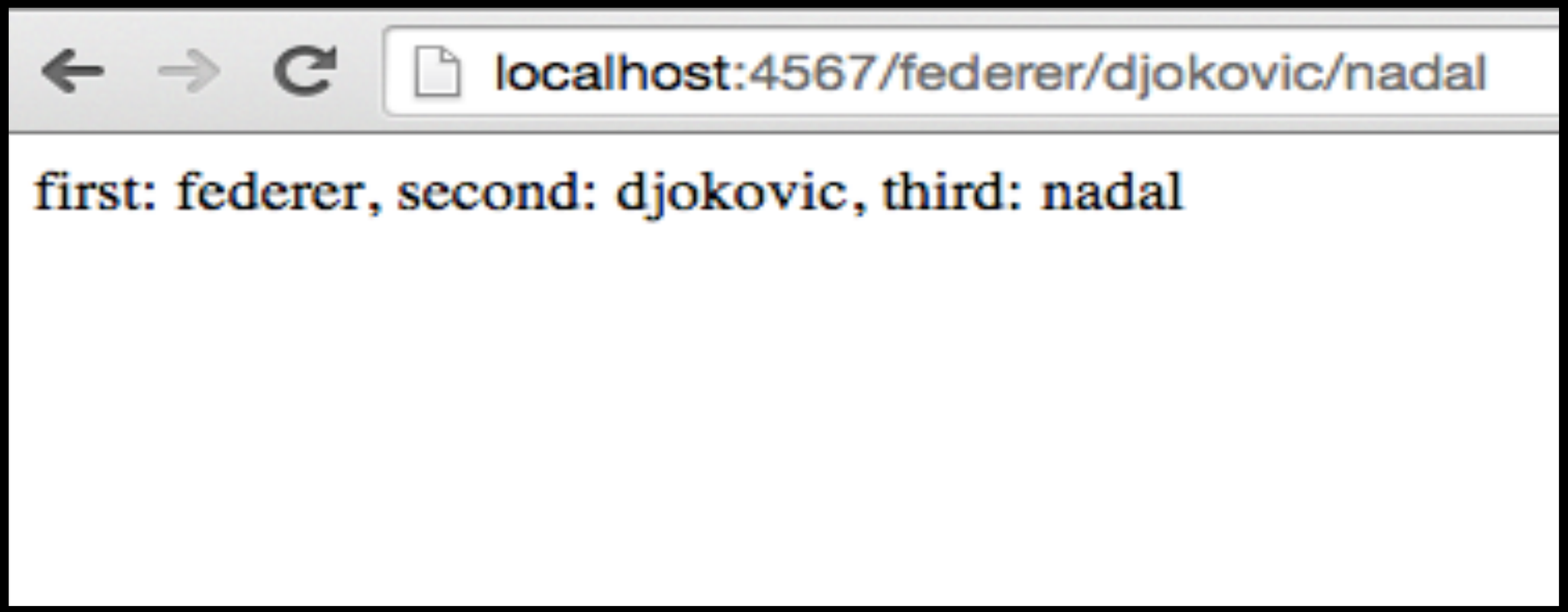
note: it’s not like your application is accepting arbitrary route request, these route requests are mostly defined by yourself in your web page. that is, you know what are the possible requests.



Ruby - Frameworks

Pass named parameters from URL.

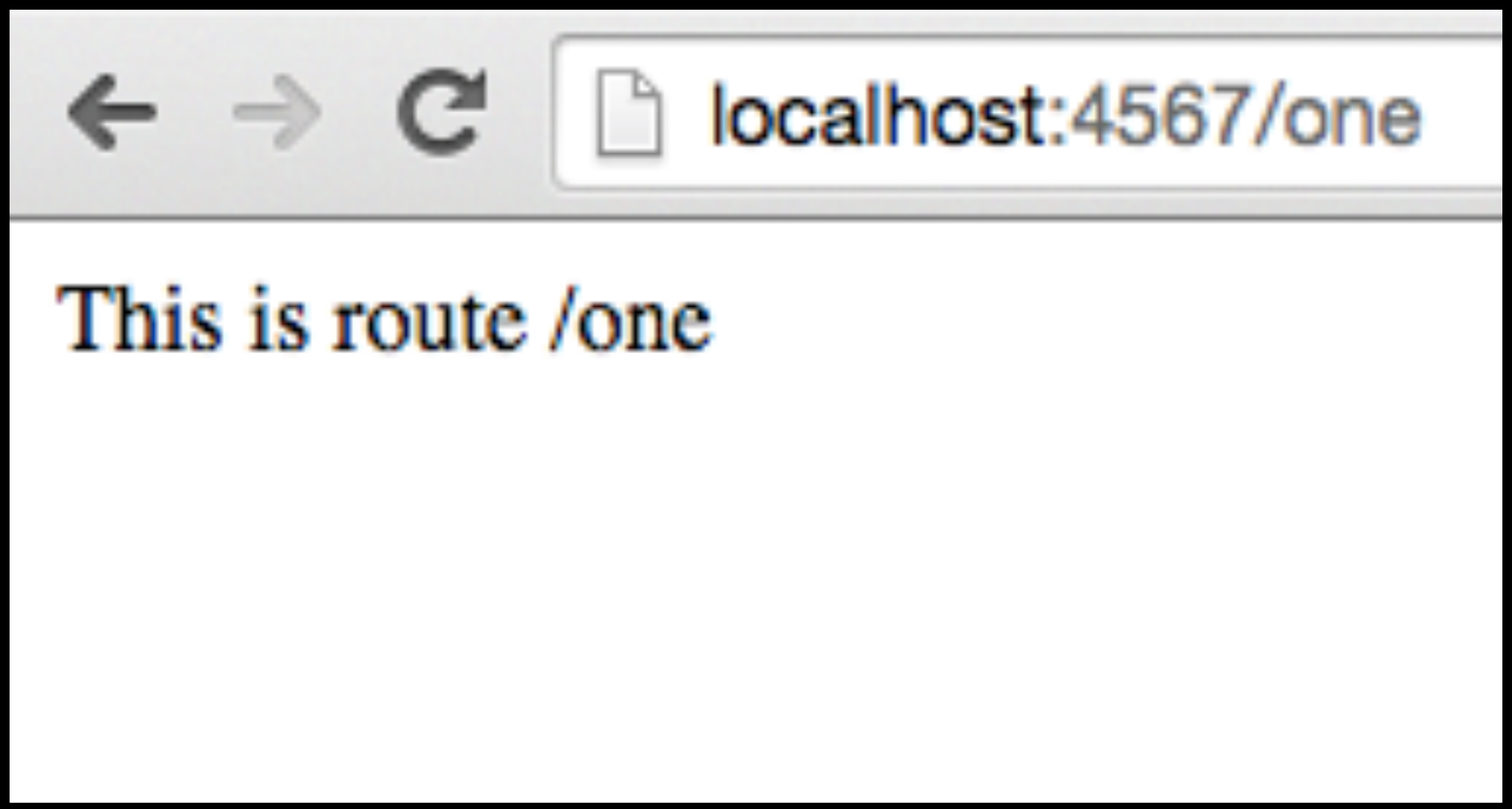
```
get '/:one/:two/:three' do

    "first:  #{params[:one]},
     second: #{params[:two]},
     third:  #{params[:three]}"
end
```



Ruby - Frameworks

Multiple routes respond the same way
- array of route

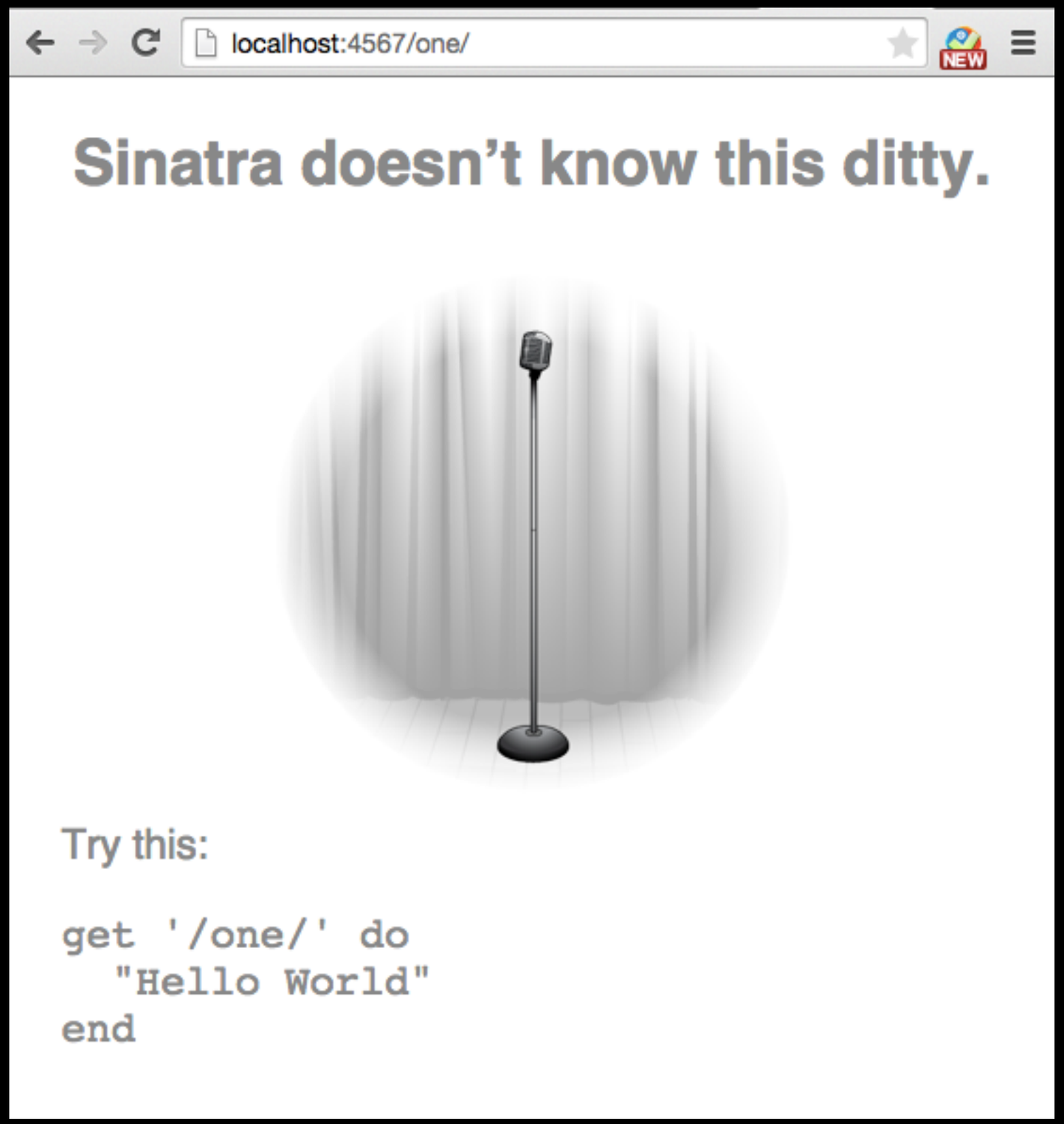
```
['/one', '/two', '/three'].each do |route|
    get route do
        "This is route #{route}"
    end
end
```



Ruby - Frameworks

Multiple routes respond the same way
- array of route

Note: if you type /one/ in your URL you will not be able to find the route.

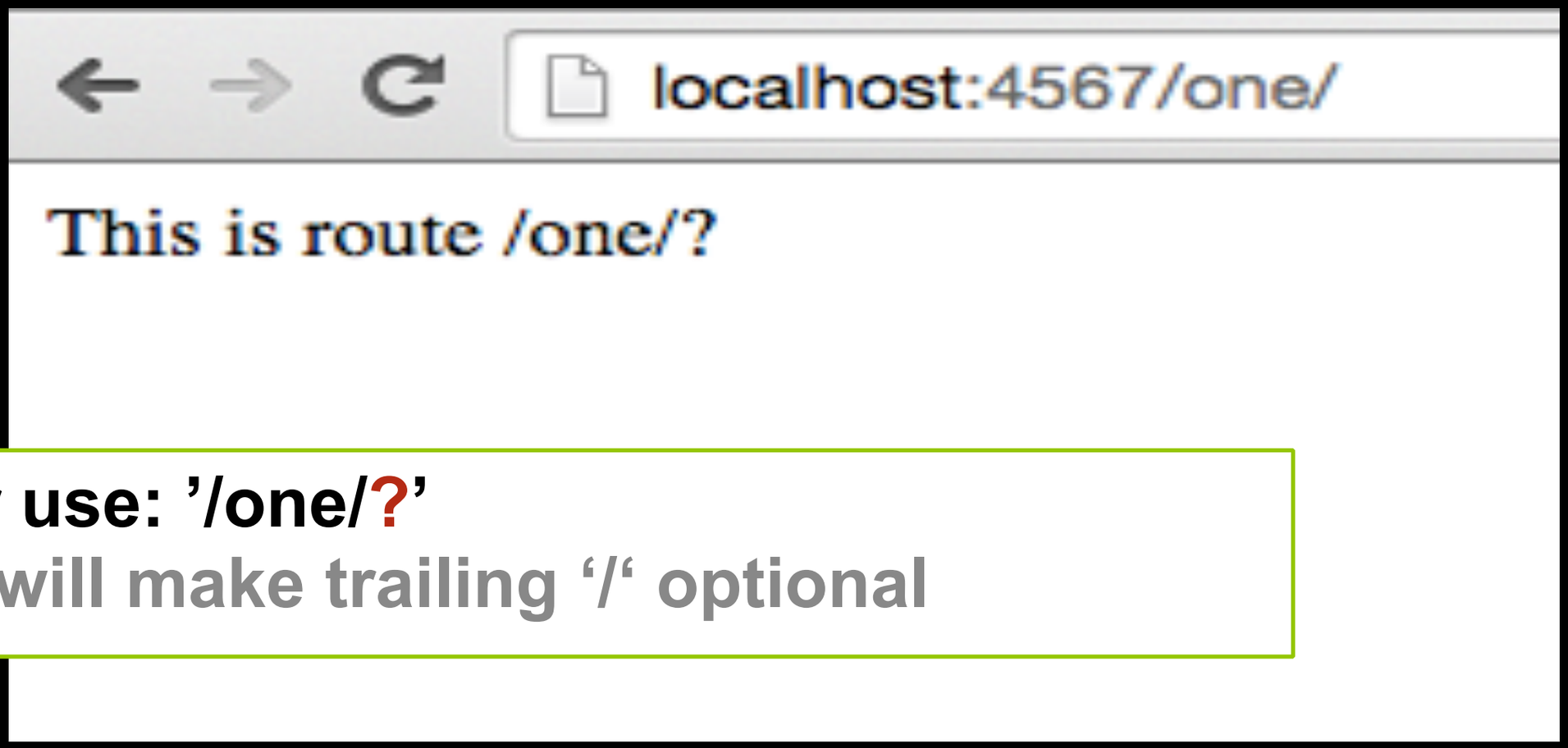


Ruby - Frameworks

Multiple routes respond the same way
- array of route

```
['/one/', '/two/', '/three/'].each do |route|
    get route do
        "This is route #{route}"
    end
end
```

now it is ok



or use: '/one/?'
it will make trailing '/' optional

Ruby - Frameworks

Post route:

come to here as form submission
(a separate GET route will receive the initial request and return a blank form html file for user to fill up, then form html file submit request will match this route)

```
post ‘/login’ do
    username = params[:username]
    password = params[:password]
end
```

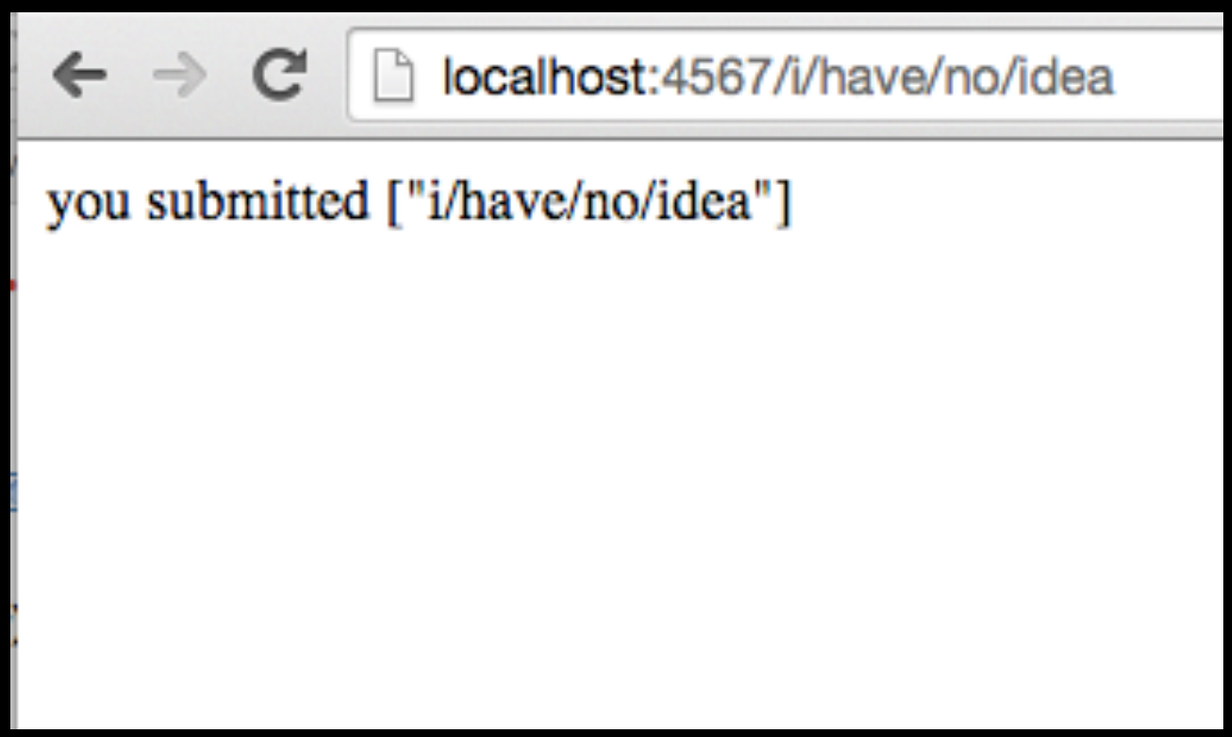
Ruby - Frameworks

Routes with wildcards:

accessed through params[:splat]

it will get all the wildcards values

```
get '/*' do
    "you submitted #{params[:splat]}"
end
```



Ruby - Frameworks

Routes with wildcards:

```
get '/say/*/to/*' do
    params['splat']     # => ["hello", "world"]
end
```

```
# matches /say/hello/to/world
```

Ruby - Frameworks

Routes with wildcards:

```
get '/download/*.*' do
  params[‘splat"]       # => [“path/to/file/myfile", "xml"]
end
```

matches /download/path/to/file/myfile.xml

or

```
get '/download/*.*' do |path, ext|
  [path, ext]           # => [“path/to/file/myfile", "xml"]
end
```

Ruby - Frameworks

Routes with wildcards:

Be careful:

```
get '/*' do
    "Look at me"
end

get '/specific' do
    "Are you ever going to see me?"
end
```

this route will never be reached

Ruby - Frameworks

Routes with regular expressions:

Specify regular expressions to match incoming request to particular route handler.

```
get %r{/coen(164|278)} do
    “Advanced Web Programming”
end

get ‘/coen164’ do
    “never reach here”
end
```

match:
/course/coen164
/coen164

Ruby - Frameworks

Redirect

many times, we redirect to another route

```
get ‘/redirect’ do
    redirect ‘http://www.google.com'
end
```

redirect is a method defined by Sinatra

default status code is 302, to add code:

```
get ‘/redirect2’ do
    redirect ‘http://www.google.com', 301
end
```

Ruby - Frameworks

static file -

```
get '/page.html' do
  "This sentence is not delivered,
   the static file will be delivered
  instead"
end
```

```
<html>
 <head>
 </head>
 <body>
  <h1>This is a static file served
        by sinatra application
  </h1>
 </body>
</html>
```

note: the static file will be in "public" folder by default
note: if route is the same as the name of a static file, static resource will be considered.

actually, you don't need to have a route for static file, if you request static file, it will be returned.

Ruby - Frameworks

Filters:

use before block and after block to modify request and response.

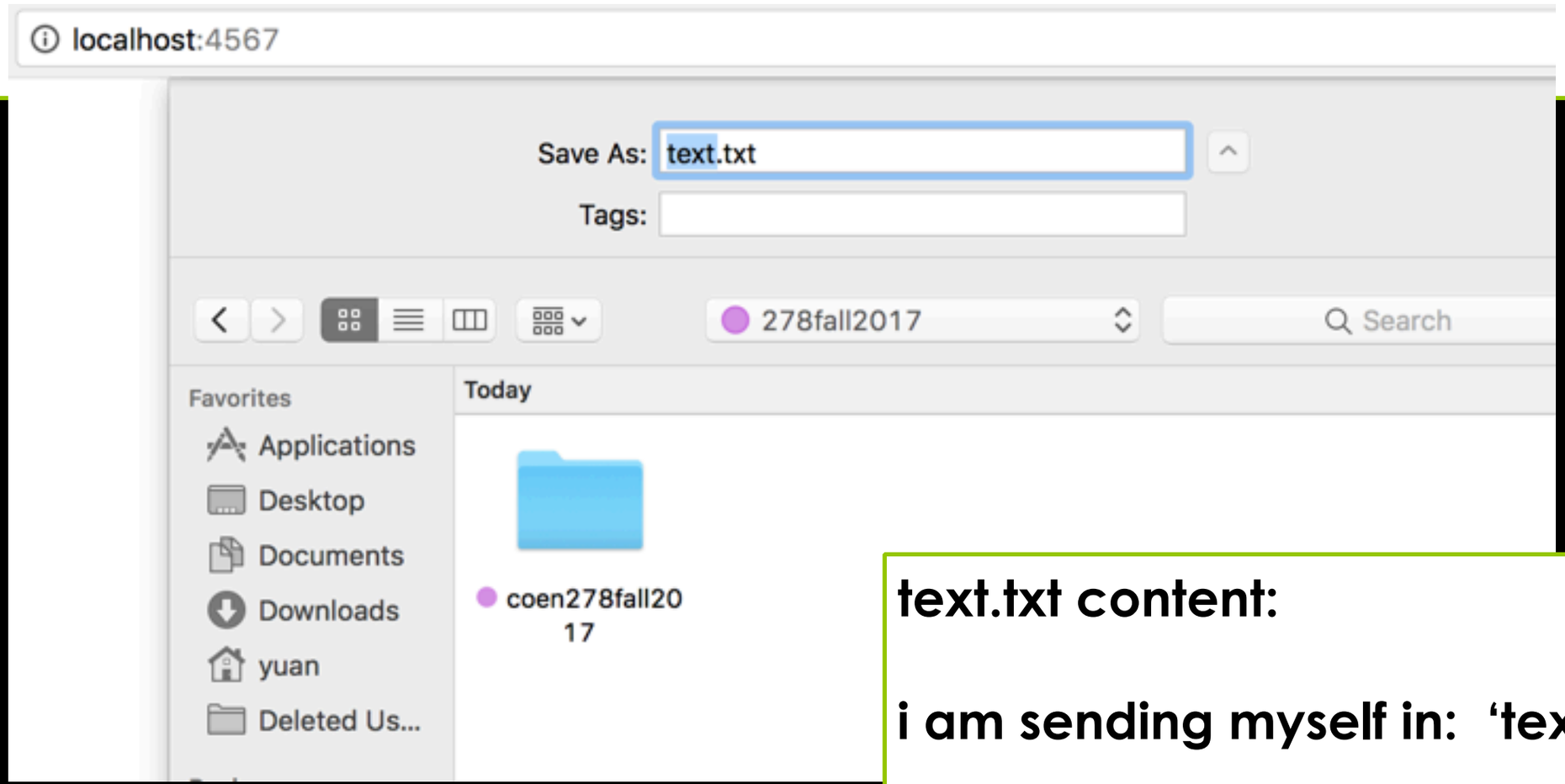
```
before do
   @before_value = ‘new value’
end

get ‘/‘ do
   “before_value has been set to #{@before_value}”
end

after do
   “after block”
end
```

Ruby - Frameworks

attachement

"attachment" is a method defined by sinatra, prompt the browser to save the file (download)

```
get "/download/:filename" do
    attachment params[:filename]
    "i am sending myself in: #{params[:filename]}"
end
```



text.txt content:

i am sending myself in: 'text.txt'

Ruby - Frameworks

configure
use this method to set some name/value

```
configure do
    # setting one option
    set :option, 'value'

    # setting multiple options
    set :a => 1, :b => 2

    # same as `set :option, true`
    enable :option

    # same as `set :option, false`
    disable :option
end
```

configure()
set()
enable()
disable()

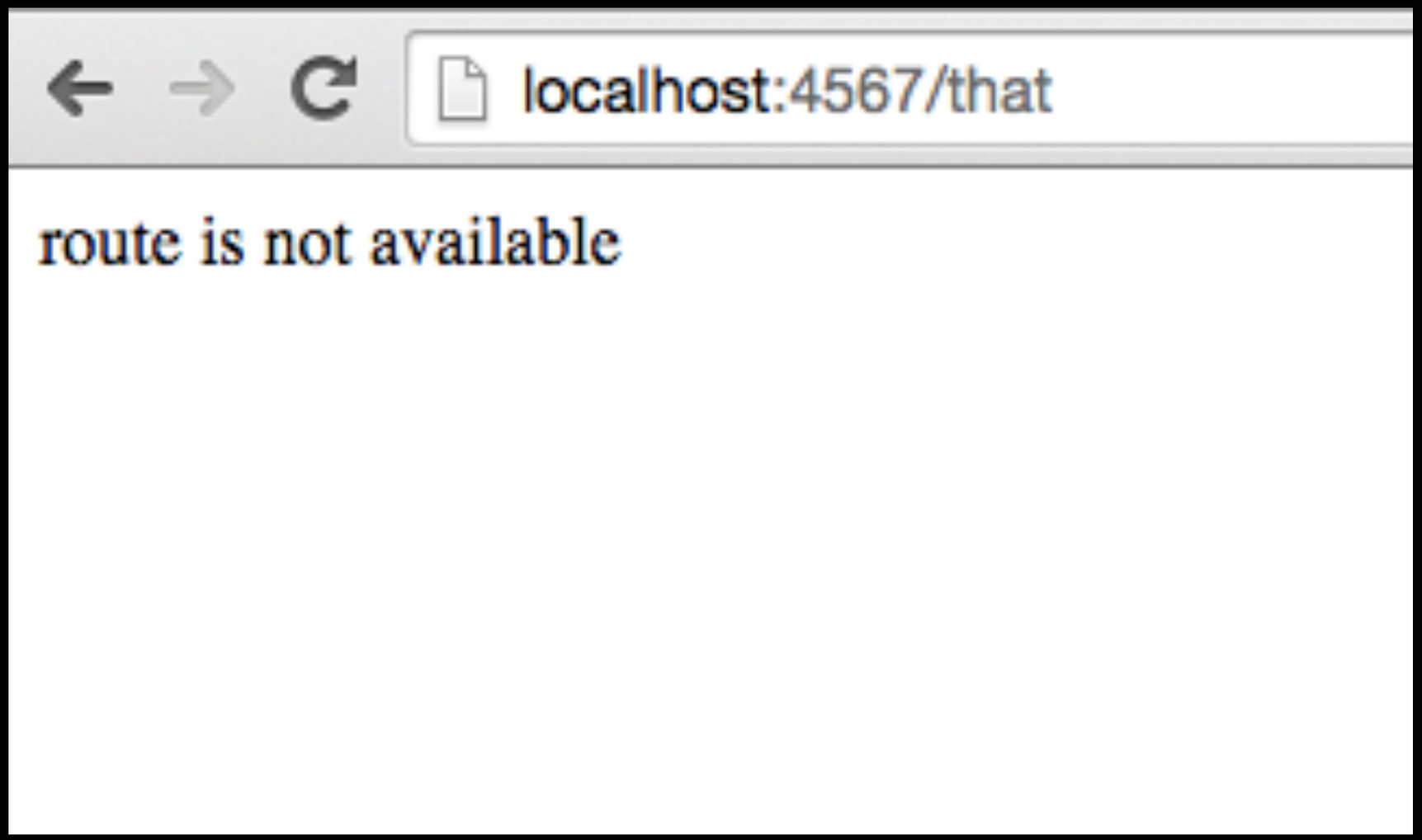
are all methods
defined by sinatra

Ruby - Frameworks

error

“not_found“ is a method defined by sinatra

```
not_found do
    “route is not available”
end
```

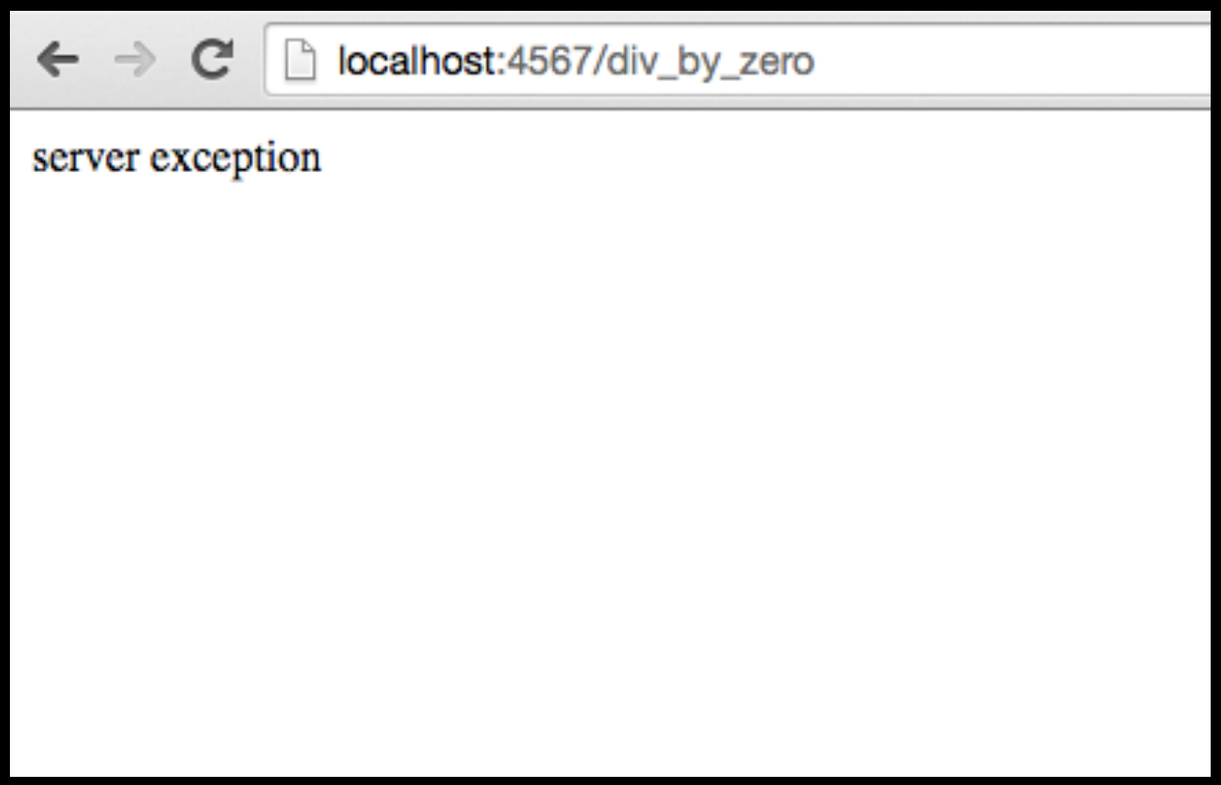


Ruby - Frameworks

error

for some server exception, use “error”

```
configure do
    set :show_exceptions, false
end

get ‘/div_by_zero’ do
    0/0
end

error do
    “server exception”
end
```



Ruby - Frameworks

pass method

pass to the next matching route

```
get '/guess/:who' do
  pass unless params['who'] == 'Frank'
  'You got me!'
end

get '/guess/*' do
  'You missed!'
end
```

Ruby - Frameworks

To be continued…