Advanced Web Programming

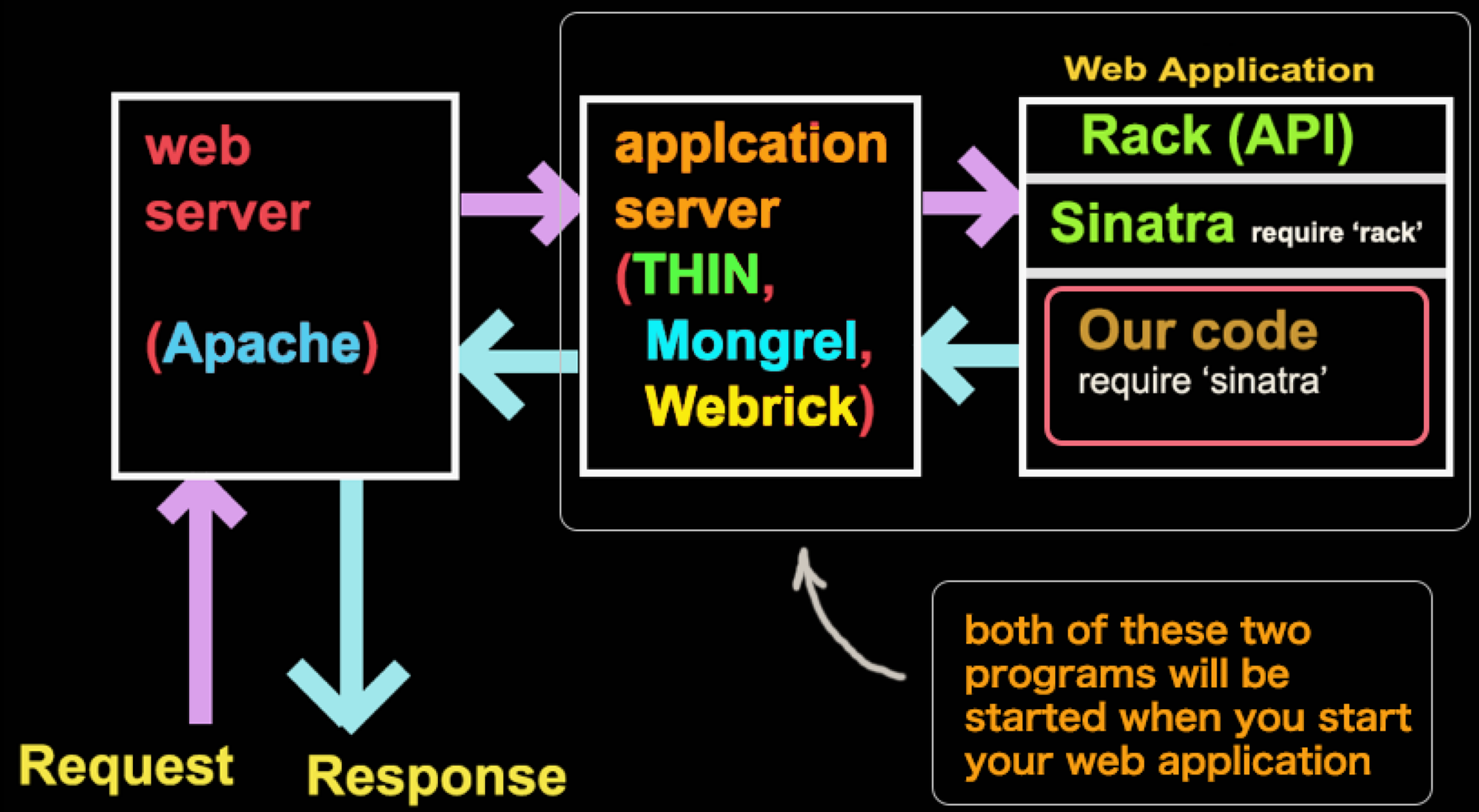
Yuan Wang

2019 Spring

Lecture 10

Ruby - Sinatra RECAP

The structure of web applications built using Sinatra



Ruby - Sinatra RECAP

Anatomy of Sinatra application

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

```
require 'sinatra'

get '/hello' do

  "hello, this is my first sinatra web application"

end
```

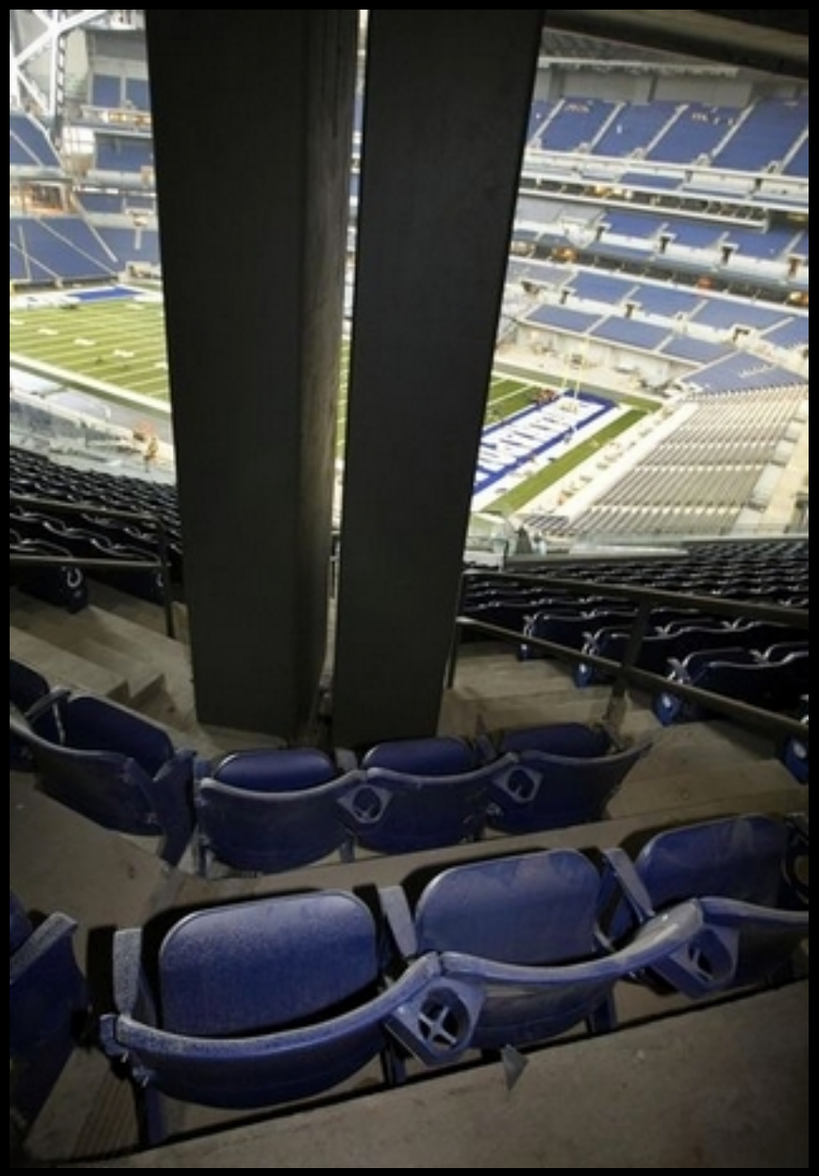
Match the request from browser:
"get" request, with url /hello
this is called 'route'
"get" is a method call

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

Ruby - Sinatra

Today's topic:

View



Ruby - Sinatra

So far, we've been producing document and return it back to the client in plain text format (except returning static html file).

```
get '/' do
  "hello dear user"
end
```

plain text

returned back to browser by Sinatra

Ruby - Sinatra

How are we going to return HTML dynamically?

That is, create HTML document on the fly and return to the browser

For dynamically returning html file to the browser, we will be using:

views

Ruby - Sinatra - view

What is a view?

a view is a html template with ruby code embedded.

we have already seen it. We know how to create ERB template, or HAML template. those are views.

they are documents that are ready to become HTML files to return to browser.

Ruby - Sinatra - view

Back in ERB section, we were using ERB library (filter) to change the template into html:

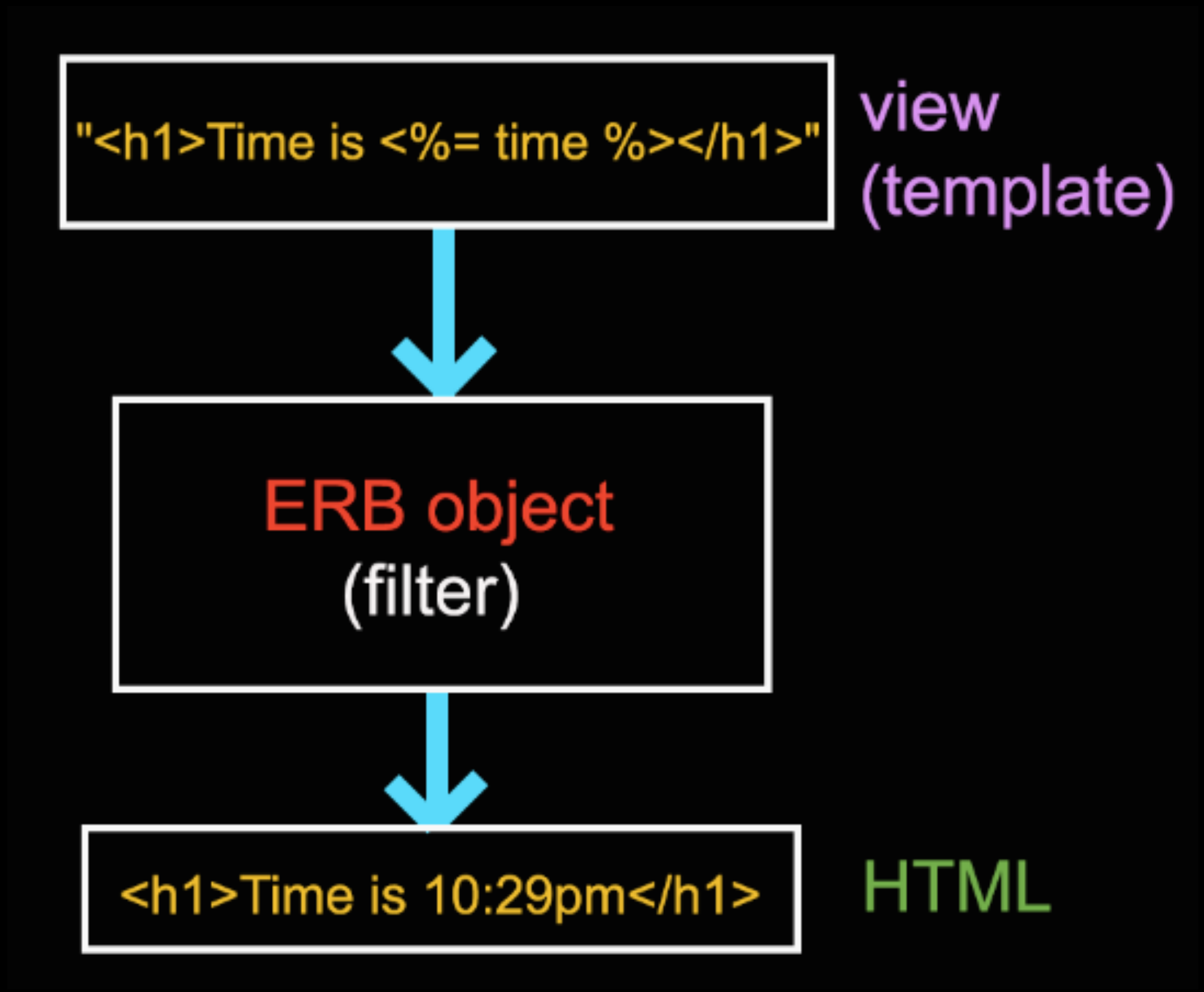
```
template = "<h1>Time is <%= time %></h1>”

renderer = ERB.new(template)
# process template
puts output = renderer.result()
```

filter object

Ruby - Sinatra - view

using ERB library is like this:



Ruby - Sinatra - view

using Sinatra:
is the same idea, but is simpler:

```
get '/' do
    erb :home
end
```

the name of the view

filter the view

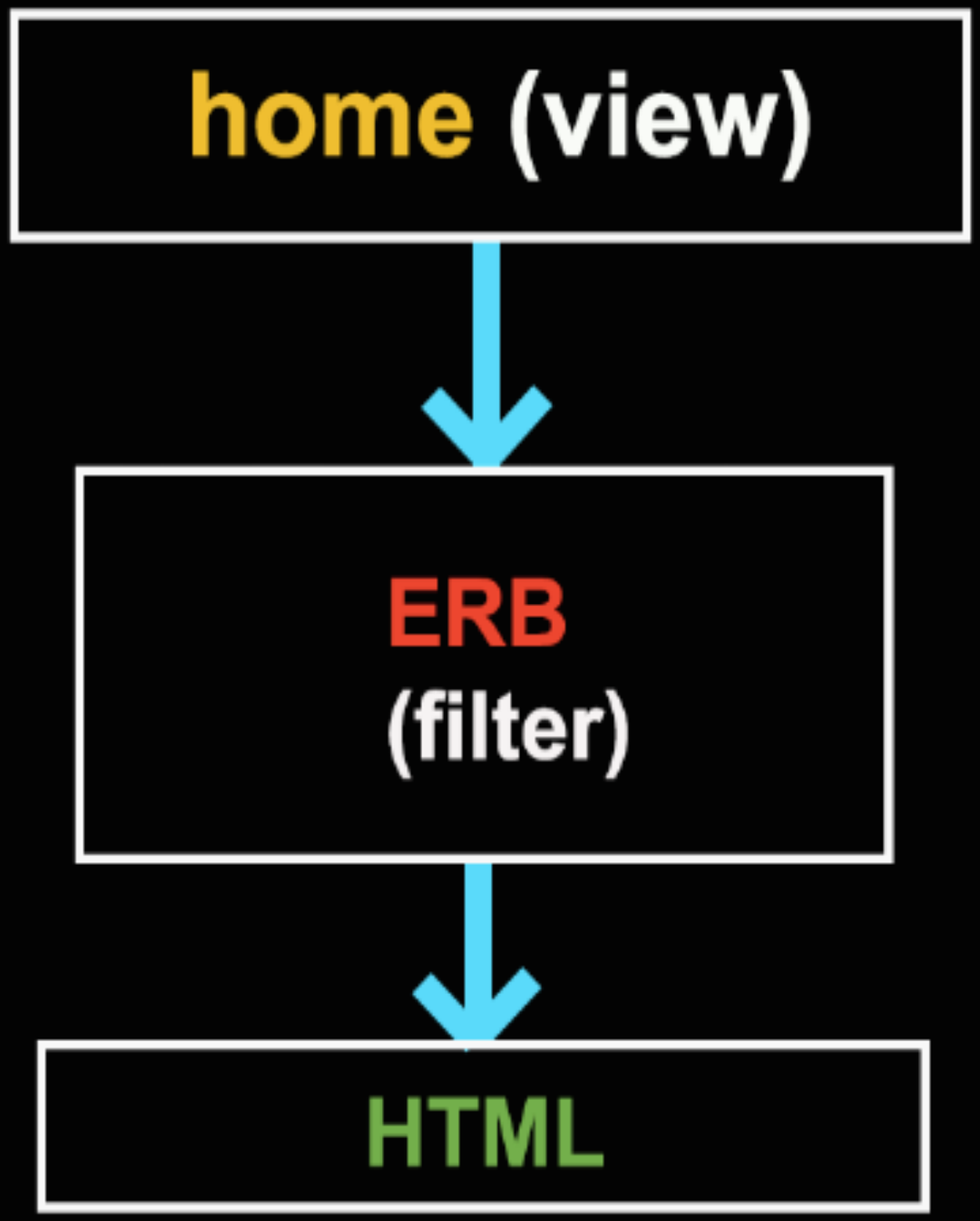
Instead of returning a text string back to browser, we call the "erb" method, to filter a view (template) called "home", then return the resulting html back to browser

Ruby - Sinatra - view

The name of the view need to be a symbol (:home)

The server then try to find the view

and sends the resulting HTML back to the browser.



Ruby - Sinatra - view

Where to put view (template)?

We can put it inside the same .rb file (inline view)

```
require 'sinatra'

get '/' do
    erb :home
end
__END__    #  put view after __END__ declaration at the bottom
@@home              # start the view with @@, follow by its name
<!doctype html>
<html>
<head></head>
<body>
<h1>this is my view in plain HTML</h1>
</body>
<html>
```

note: this comments should not be here.
it is here just for explaining things

this is my view in plain HTML

Ruby - Sinatra - view

We can easily modify the view to make the return like this:

Songs by Sinatra

- Home
- About
- Contact

Welcome to the website about songs by Frank Sinatra

Ruby - Sinatra - view

Right now, these hyper links (Home, About and Contact) are not returning any different pages but the same home page.

```
<a href='/'>Home</a>
<a href='/'>About</a>
<a href='/'>Contact</a>
```

If we want Home, About and Contact to return different page, what should we do?

Ruby - Sinatra - view

Just add more routes:

```
get '/' do
  erb :home
end

get '/about' do
  erb :about
end

get '/contact' do
  erb :contact
end
```

Ruby - Sinatra - view

and in the view, we need to specify the href:

```
<a href="/">Home</a>
<a href="/about">About</a>
<a href="/contact">Contact</a>
```

Ruby - Sinatra - view

we also need to add two more views at the bottom of the file:

```
@@home
<html>
<head></head>
<body>
<h1>this is my view in plain HTML</h1>
</body>
<html>
```

```
@@about
<!doctype html>
  <html>
  <head></head>
  <body>
  <h1>Songs by Sinatra</h1>
  <p>This is a practice site for web programming in Sinatra</p>
  </body>
</html>
```

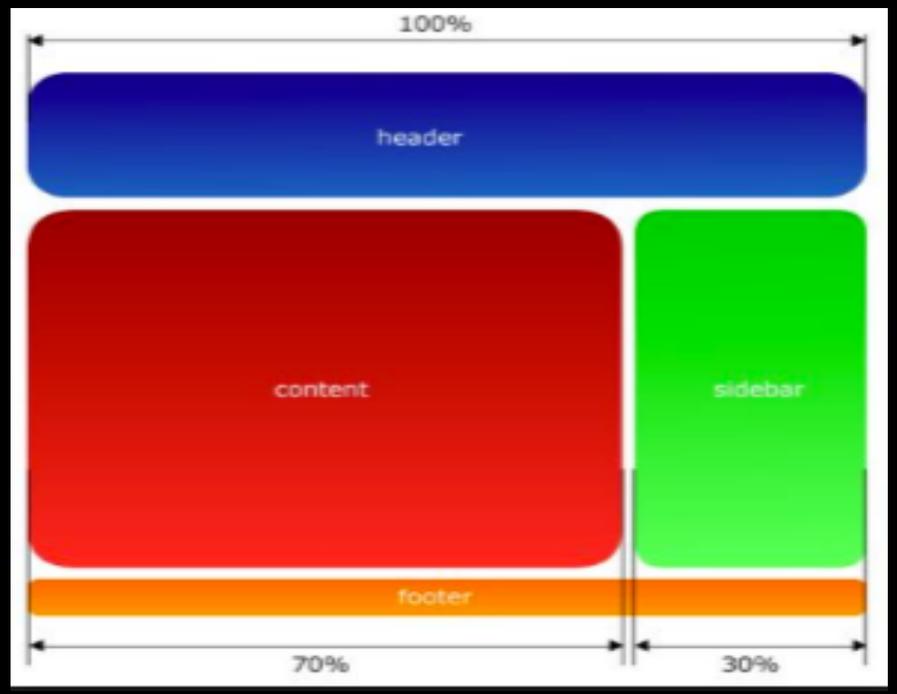
```
@@contact
<!doctype html>
  <html>
  <head></head>
  <body>
  <h1>Songs by Sinatra</h1>
  <p>Contact me at ywang6@scu.edu</p>
  </body>
</html>
```

You might have noticed, that the 3 views ‘home’, ‘about’, and ‘contact’, are kind of sharing some duplicated html.

Ruby - Sinatra - view

To get rid of duplication, we use a special view: layout

layout is usually basic structure and common components of all the pages. each different page then add extra components to the layout page.



Ruby - Sinatra - view

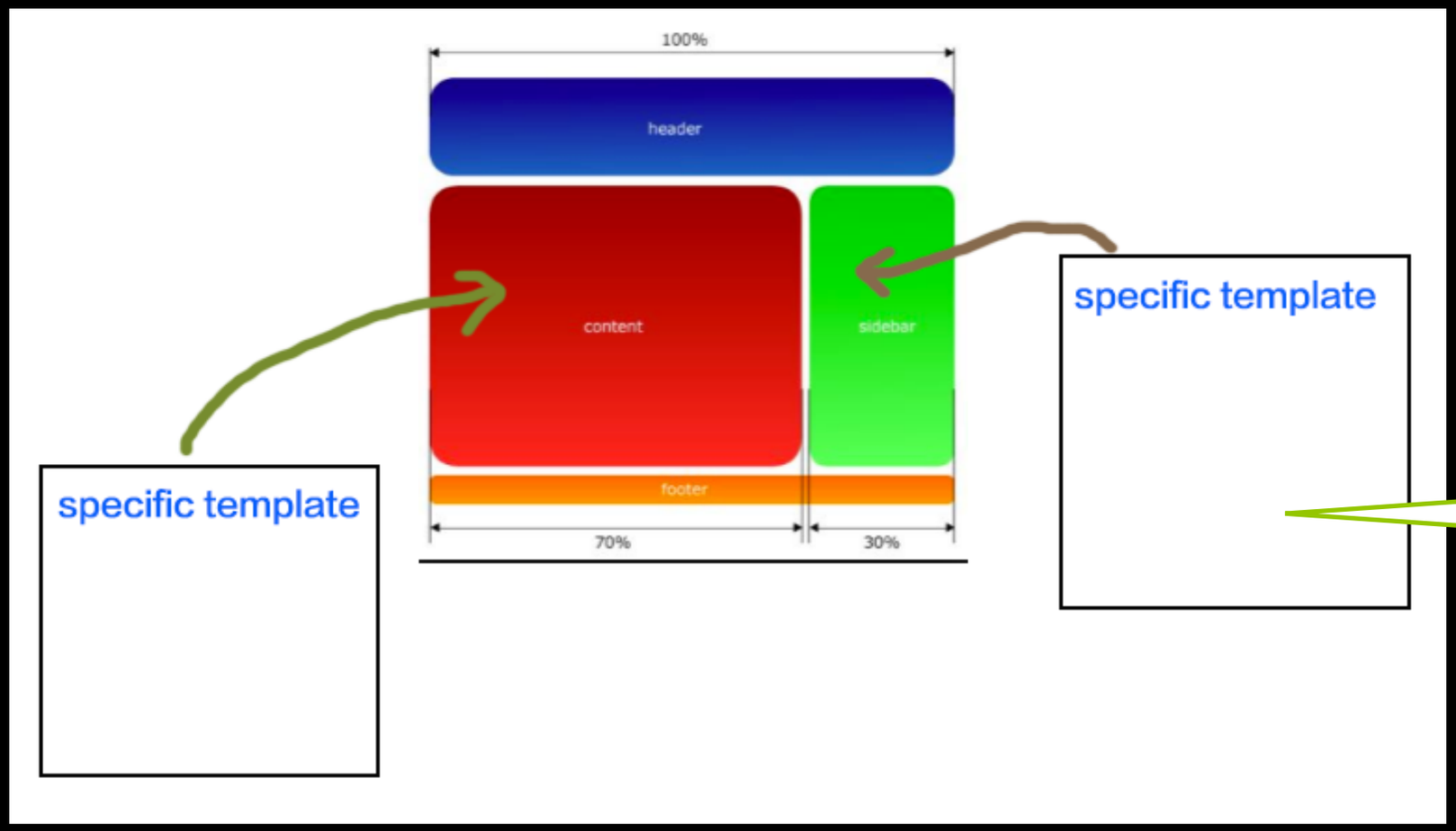
layout is a special general view (template).

it will be returned by all routes handler as a common view.

then each specific view will be inserted into this common view

Ruby - Sinatra - view

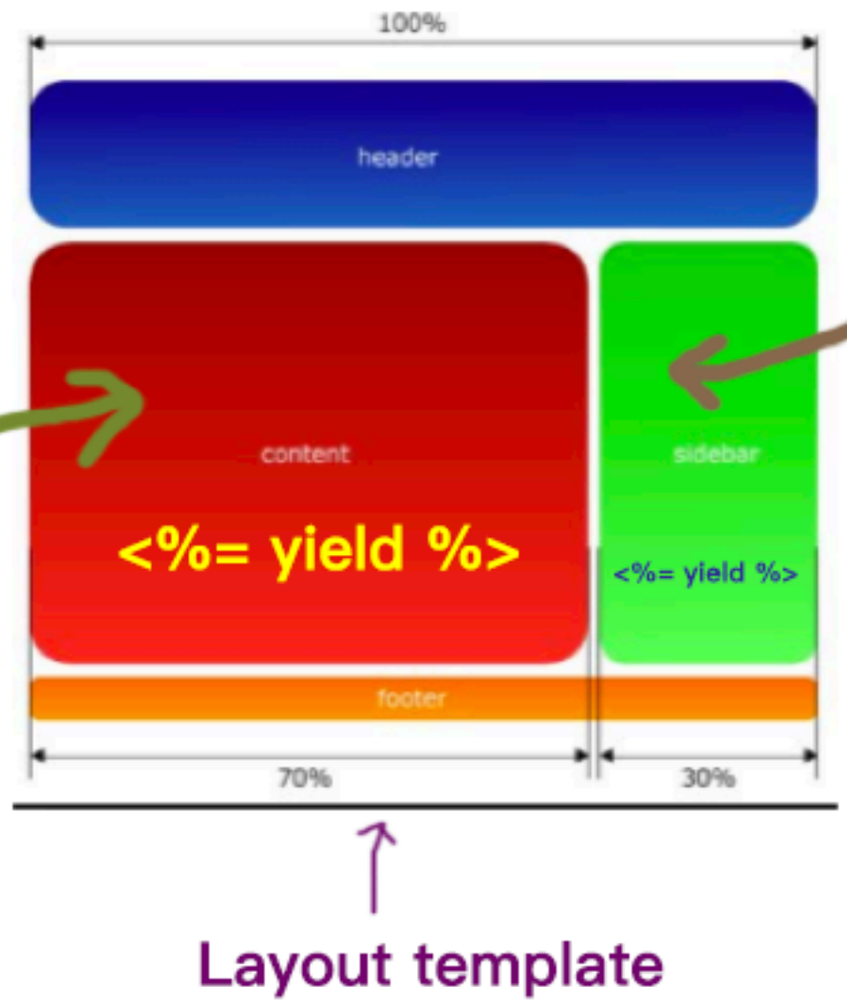
it is like a template of template: there is place holder inside layout view that need to be replaced with specific view.



the specific view, will be inserted into layout to form the real return page

Ruby - Sinatra - view

the layout just need to say <%= yield %> at the place that the specific view need to be inserted



specific template

specific template

note, the flow is this: route-> route handler -> route handler fetch fetch layout view, embed the specific view at the place of "yield"

Ruby - Sinatra - view

In our case, it will be like this:

```
__END__
@@layout
<!doctype html>
  <html>
  <head></head>
  <body>
  <h1>Songs by Sinatra</h1>
  <%= yield %>
  </body>
</html>
@@home
<p>This is a practice site for web programming in Sinatra</p>
@@about
<p>This is a practice site for web programming in Sinatra</p>
@@contact
<p>Contact me at ywang6@scu.edu</p>
```

general view that are shared by all pages

this view will always be filtered first by default before the specific views is filtered

specific views for home page, about page and contact page

Ruby - Sinatra - view

External views:

As inline views (views that are in the same file as your route handler) get more complex, your main program will get crowded.

So it makes more sense to put each view in separate file (.erb)

Ruby - Sinatra - view

Using external views:

main program becomes:

```
require 'sinatra'

get '/' do
  erb :home
end

get '/about' do
  erb :about
end

get '/contact' do
  erb :contact
end
```

'home' view

'about' view

'contact' view

Ruby - Sinatra - view

create these
4 files:

layout.erb

```
<!doctype html>
  <html>
  <head></head>
  <body>
  <h1>Songs by Sinatra</h1>
  <ul>
  <li><a href="/">Home</a></li>
  <li><a href="/about">About</a></li>
  <li><a href="/contact">Contact</a></li>
  </ul>
  <%= yield %>
  </body>
</html>
```

home.erb

```
<p>Welcome to the website about songs by Frank Sinatra</p>
```

about.erb

```
<p>This is a practice site for web programming in Sinatra</p>
```

contact.erb

```
<p>Contact me at ywang6@scu.edu</p>
```

Create a folder called ‘**views**’ and put these 4 .erb files inside this folder

Ruby - Sinatra - view

Create a folder ‘public’

this is the folder for images, CSS files, static HTML files, and JavaScript files.

‘views’ and ‘public’ folders are default folder folders. you can change their names by:

```
# set the name of the public folder to be ‘assets’
set :public_folder, ‘assets’

# set the name of the view folder to be ‘templates’
set :views, ‘templates’
```

“set” is a method defined in Sinatra library, you can call it in your code as a configuration

Ruby - Sinatra - view

Add image to our site

create a ‘images’ folder inside ‘public’ folder

put an image inside it.

change ‘home.erb’ to:

```
<p>Welcome to the website about songs by Frank Sinatra</p>
<img src="/images/fsinatra.jpg" alt="Frank Sinatra">
```

Ruby - Sinatra

Now we have

Songs by Sinatra

- Home
- About
- Contact

Welcome to the website about songs by Frank Sinatra

Ruby - Sinatra - view

We can then add CSS to make the page prettier.

create ‘style.css” and put inside ‘public’ folder

```
html {background-color: #ccc;}
body {
    width:700px;
    height:900px;
    background-color:#dddcd9;
    margin:2em auto 2em;
    padding:25px;
    position:relative;
}
h1 {
    color:#a09b89
}
```

Ruby - Sinatra - view

CSS

add the following line

```
<link href="style.css" rel="stylesheet" type="text/css">
```

into your view file, which is layout.erb file (of course you need to add it into your view file, because view file is going to be your final html)

Ruby - Sinatra - view

It's time to make our HTML file dynamic

what do we mean by that?

That means we need some ruby code inside template, and execute ruby code and put evaluated values in the template to make it the final html.

where do these values in the template come from?

some of them will be created by the route handler.

Ruby - Sinatra - view

dynamic HTML

how can we pass values from route handler to the view?

instance variable

Ruby - Sinatra - view

Using instance variables to pass values from route handlers to views:

example:

```
get '/home' do
  @name = 'yuan'
  erb :home
end
__END__
@@home
<h1>Hello <%= @name %></h1>
```

Templates are evaluated within the same context as route handlers.

Instance variables set in route handlers are directly accessible by templates

Ruby - Sinatra - view

For example: set different titles for different pages.

```
get '/about' do
  @title = "this is the about page"
  erb :about
end
```

when 'about' view is called, @title in layout.erb will be the value from 'about' route handler

in layout.erb view:

```
<head>
<title><%=@title || "Songs By Sinatra" %> </title>
```

"Songs by Sinatra" will be the default

Ruby - Sinatra - view

Using non-default layout.

```
get '/' do
  erb :home, :layout => :homelayout
end

get '/about' do
  erb :about, :layout => :aboutlayout
end

get '/contact' do
  erb :contact, :layout => :contactlayout
end
```

pass this as second parameter to erb()
then create your own layout view, like homelayout.erb

Ruby - Sinatra - view

Another way to specify layout

```
get '/' do
  erb :homelayout do
    erb :home
  end
end
```

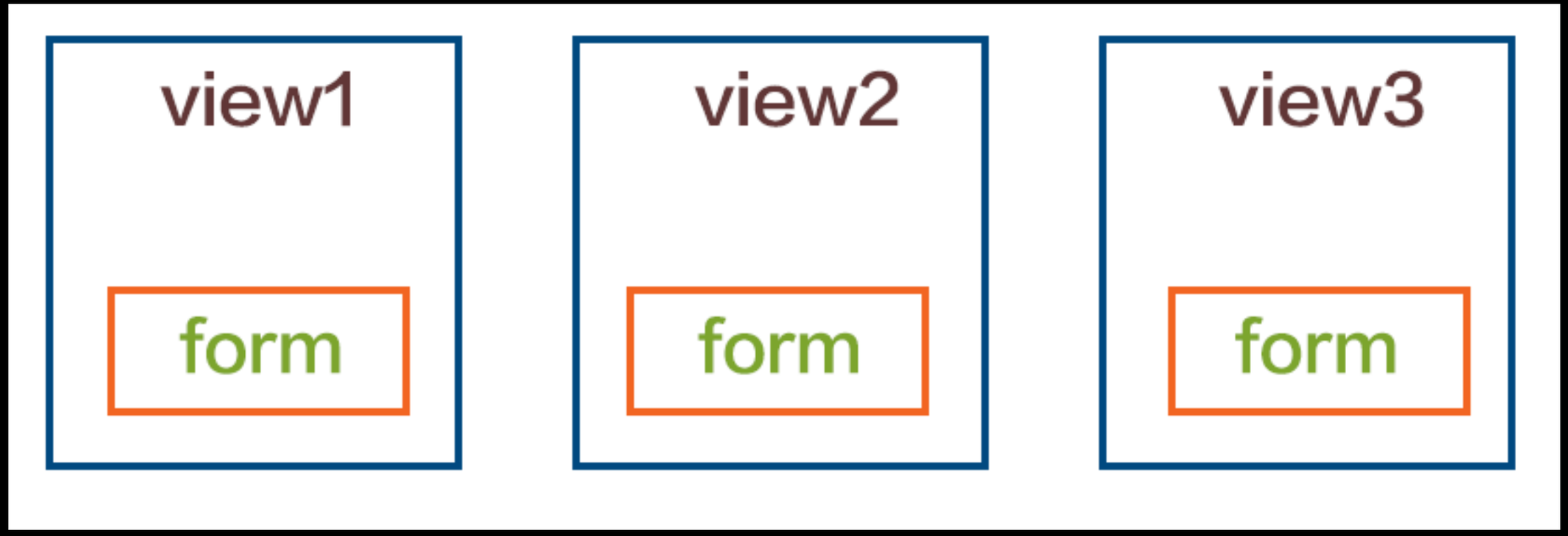
same as

```
get '/' do
  erb :home, :layout => :homelayout do
end
```

Ruby - Sinatra - view

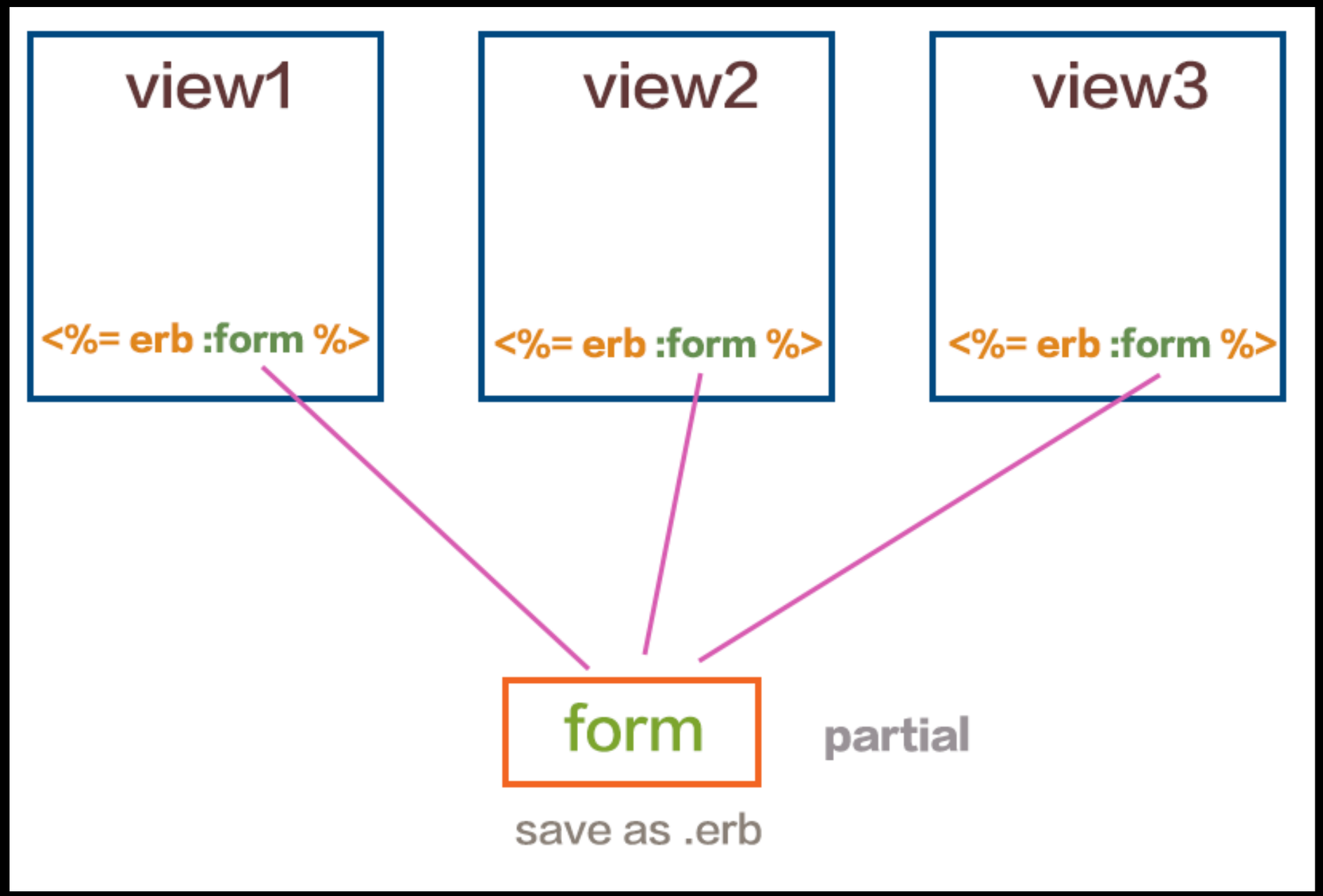
Partial: is a piece of view that is included in multiple views so that you don't need to duplicate this piece in all the views.

For example, if multiple views have the same form:



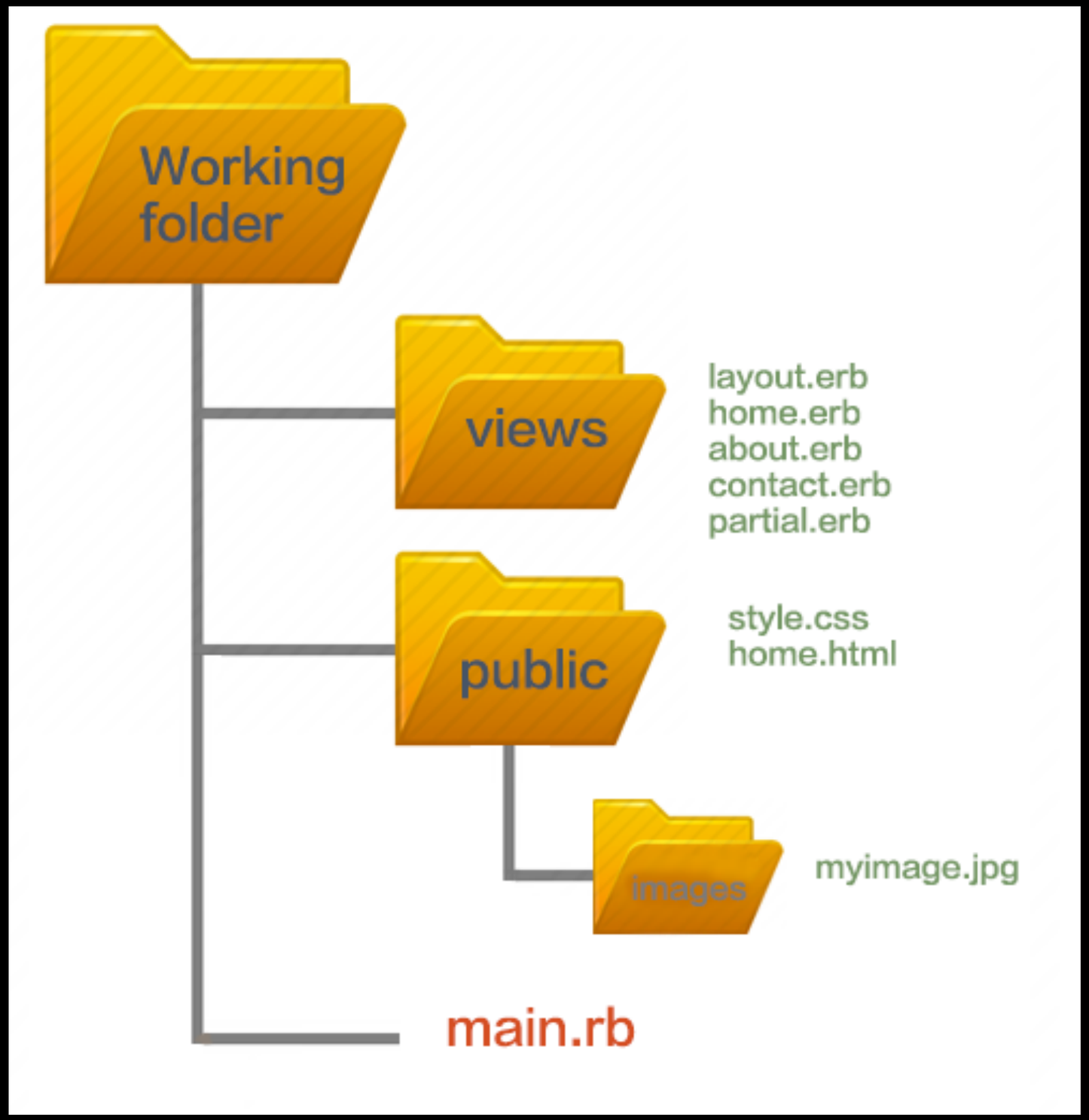
Ruby - Sinatra - view

…then, you can take the form out and save it in a separate view (partial), then render this form in all views (view1, view2, view3), like this



Ruby - Sinatra - view

Directory structure summary:



Ruby - Sinatra - view

Demo code:

```ruby
require 'sinatra'
require 'sinatra/reloader'

get '/' do
  @title = "home page"
  erb :home
end

get '/about' do
  @title = "about page"
  erb :about
end

get '/contact' do
  @title = "contact page"
  erb :contact
end


not_found do
  @title = "not found page"
  erb :notfound, :layout => false
end
```

```
<h1>i don't know what to do</h1>
```

```
<html>
  <head>
  <link href="style.css" rel="stylesheet" type="text/css">
<title><%= @title || "Songs by Sinatra" %></title>
</head>
<body>
<h1> Songs by Frank Sinatra (layout)</h1>
<ul>
  <li><a href='/'>Home</a> </li>
  <li><a href='/about'>About</a> </li>
  <li><a href='/contact'>Contact</a> </li>
  </ul>
<%= yield %>
</body>
  </html>
```

```
  <p> Welcome to website of songs by sinatra</p>
<img src="images/fsinatra.jpg" alt="frank sinatra" width=100%>
<%= erb :form %>
```

```
  <p> information about my site</p>
<%= erb :form %>
```

```
  <p> contact me at 911</p>
<%= erb :form %>
```

form.erb

```
<h1>Please logon</h1>
<form action="/login" method='post'>
      username: <input type='text' name='username'></input><br>
      password: <input type='text' name='password'></input>
      <br>
      <input type='submit' name='submit'></input>
</form>
```

Ruby - Sinatra - view

About other template systems.

We already know:

- erb
- haml

There is one more:

- slim

Ruby - Sinatra - view

Slim

to use slim, install it

```
> gem install slim
```

Ruby - Sinatra - view

Slim

in your handler, instead using erb, use slim method

```
require 'sinatra'
require 'slim'

get '/' do
  slim :myhome
end
```

use ‘slim’ method

Ruby - Sinatra - view

Slim

slim view file, is in saved in .slim (instead of .erb)

like HAML, it is not using html syntax directly, but using syntax sugar.

its syntax is closer to html than HAML is, it is simply without <> and closing tag. for example:

```
doctype html
html
  head
  body
    h1 this is my view in plain HTML
```

Ruby - Sinatra

Official site:

http://www.sinatrarb.com

Code:

https://github.com/sinatra/sinatra/

Ruby - Sinatra - More about CSS

Not only can you set a template for your HTML,

you can also set a “template” for your CSS, and a CSS preprocessor will process it, and turn it into pure CSS

this is called: CSS preprocessor: SASS
(Syntactically Awesome Stylesheets)

Ruby - Sinatra - More about CSS

CSS preprocessor: SASS

there are two types of syntax:

1. SCSS (Sassy CSS): new syntax
   file extension: .scss

2. Indent syntax: old syntax.
   file extension: .sass

both are supported.

Ruby - Sinatra - More about CSS

There are mainly two things you can define using SASS:

- Variables: to save some values and reuse them later

- Mixin: a fragments of CSS and can be reused by including it in other declarations.

Ruby - Sinatra - More about CSS

To use SASS, down load ruby library

install sass gem

```
> gem install sass
```

in your .rb main program:

```
require 'sass'
```

Ruby - Sinatra - More about CSS

in .rb main program, create a route to call scss method to process scss style file

```
get '/styles.css' do
  scss :styles
end
```

put before any other route handlers

note, the route is still '/styles.css'

This is similar to the format:

```
erb :about   # calling erb method to process
             #the "about" view
```

Ruby - Sinatra - More about CSS

Create CSS “template”: -example

styles.scss

variables

```
$red: #903;
$black: #444;
$white: #fff;
$main-font: Helvetica, Arial, sans-serif;

body {
  font-family: $main-font;
}

h1 {
  color: $red;
  font: 32px/1 $main-font;
}
```

Ruby - Sinatra - More about CSS

```
header h1 {
  font-size: 40px;
  line-height: 80px;
  background: transparent url(/images/logo.png) 0 0 no-repeat;
  padding-left: 84px;
}

nav {
  @include tabs ($background: $black, $color: $white);
  font-weight: bold;
}

p {
  font: 13px/1.4 $main-font;
}

label {
  display: block;
}
```

include the mixin (for <nav>)

parameters

Ruby - Sinatra - More about CSS

```
@mixin tabs ($background: blue, $color: yellow) {
  ul {
    list-style: none;
    margin: 0;
    padding: 0;
    background: $background;
    overflow: hidden;
  }
  li {
    float: left;
  }
  a {
    text-decoration: none;
    display: block;
    padding: 8px;
    background: $background;
    color: $color;
    &:hover {
      background: darken($background, 20%);
    }
  }
}
}
```

define a style mixin called "tabs", with parameters $background and $color, default values are blue and yellow

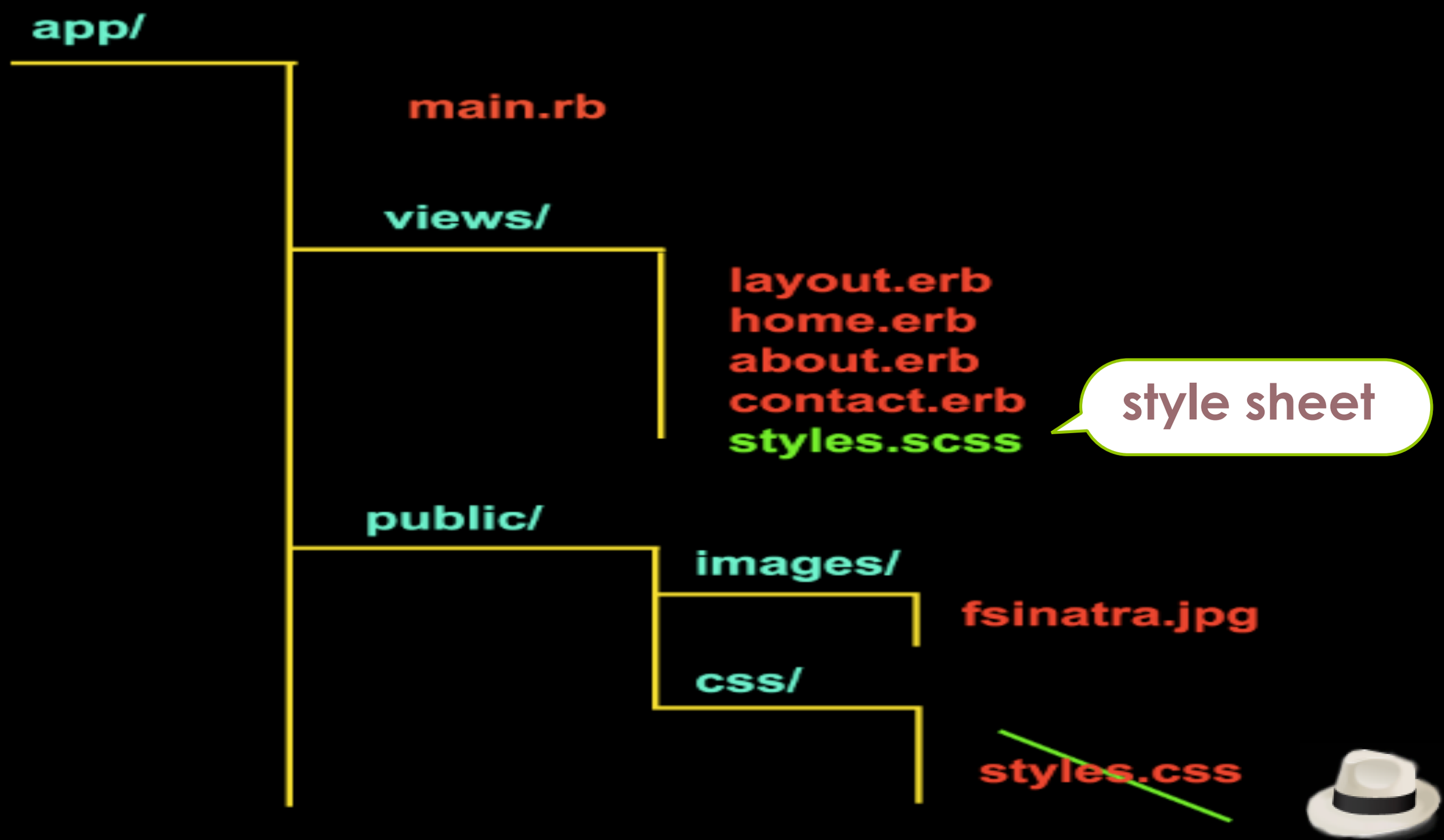
Ruby - Sinatra - More about CSS

Delete the original styles.css from public folder

put new styles.scss in the views folder

Ruby - Sinatra

The application folder structure becomes like this:



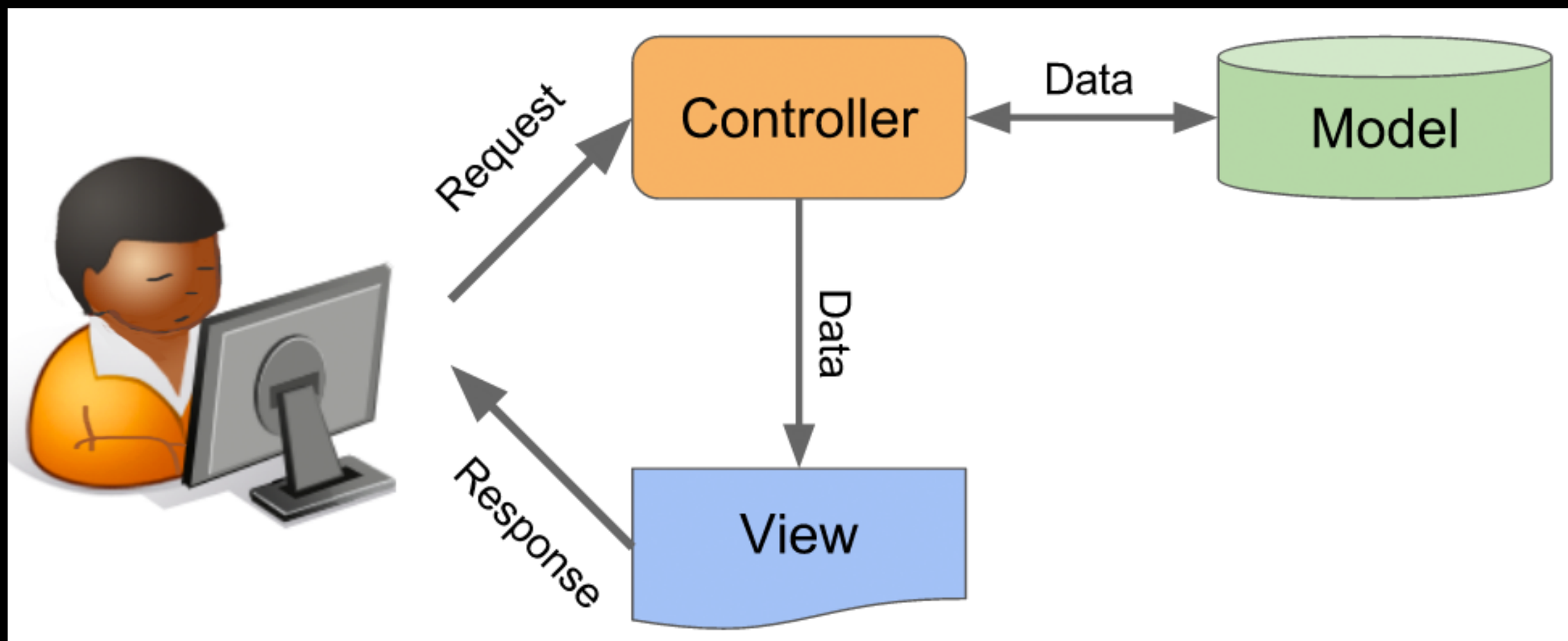
Ruby - Sinatra

Using database

This is the datasource for our views

Ruby - Sinatra

MVC design pattern



Ruby - Sinatra

Adding database

There are different database systems, they are either relational databases:

MySQL, PostgreSQL, SQLite, Oracle

or non-relational ones:

MongoDB, Redis, CouchDB, DynamoDB

Ruby - Sinatra - SQLite

We will be using SQLite

http://sqlite.org

Ruby - Sinatra - SQLite

SQLite:

- Open source

- Server-less relational database

  meaning: no separate database process (server) is running such that your application need to talk to the server as a client, like in the case of MySQL, Oracle

Ruby - Sinatra - SQLite

SQLite:

- All the data information is stored in a single file (cross platform)

- It is an embedded database engine: other programming language can embed SQLite library calls

Ruby - Sinatra - SQLite

SQLite:

- Think of SQLite not as equivalent to MySQL or Oracle, but as a fopen() , read(), write() functions, to read and write directly to ordinary disk files.

Ruby - Sinatra - SQLite

SQLite:

- No “installation”:

The SQLite library is linked in and becomes an integral part of the application program.

The application program uses SQLite’s functionalities by function calls.

Ruby - Sinatra - SQLite

SQLite:

- Zero-configuration:

  There is no "setup" procedure.
  There is no server process that needs to be started, stopped, or configured.
  There is no need for an administrator assign access permissions to users.
  SQLite uses no configuration files.

Ruby - Sinatra - SQLite

SQLite: command line utility: sqlite3

```
> sqlite3 database-file
```

Use the database-file, created if does not exist

use any file extension you like
.db, .database, .sqlite, etc…

example:

```
> sqlite3 test1.db
SQLite version 3.8.5 2014-08-15 22:37:57
Enter ".help" for usage hints.
sqlite>
```

Note. if database-file, does not exist, then you are creating a database file.

SQLite is one file per database.

Ruby - Sinatra - SQLite

```
> sqlite3
SQLite version 3.8.5 2014-08-15 22:37:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent
database.
sqlite>
```

If you do not specify a file name, the operations are saving in memory and will be lost after you exit the shell

Ruby - Sinatra - SQLite

```
> sqlite3
SQLite version 3.8.5 2014-08-15 22:37:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent
database.
sqlite> .open test1.db
```

open a database file
from within if you did not give a
database file when starting sqlite3

Ruby - Sinatra - SQLite

```
> sqlite3
SQLite version 3.8.5 2014-08-15 22:37:57
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a
persistent database.
sqlite> .save test1.db
```

You can also save your in-memory work into a database file

Ruby - Sinatra - SQLite

```
> sqlite3 database-file
sqlite> .exit
```

this will exit sqlite3
or use Control-d

```
> sqlite3 database-file
sqlite> .help
```

for help

Ruby - Sinatra - SQLite

list the names and files of the database

```
> sqlite3 database-file
sqlite> .databases
```

most commands in sqlite3 start with '.'

Ruby - Sinatra - SQLite

create table

```
> sqlite3 database-file
sqlite> CREATE TABLE table-name (
   . . . >      col1 varchar(10) primary key,
   . . . >      col2 text,
   . . . >      col3 real );
sqlite>
```

this of course is SQL command
not sqlite3 command

Ruby - Sinatra - SQLite

```
> sqlite3 database-file
sqlite> .schema table-name
```

show the CREATE TABLE command used to create the table “table-name”, which gives us the definition (schema) of the table

Ruby - Sinatra - SQLite

insert data into table

```
> sqlite3 database-file
sqlite> INSERT INTO table-name
   . . .> (col1, col2, col3) values
   . . .> (“10”, “yuan”, “wang”);
```

another SQL command

Ruby - Sinatra - SQLite

delete from table

```
> sqlite3 database-file
sqlite> DELETE FROM table-name
   . . .> where col1=="10";
```

Ruby - Sinatra - SQLite

send query results into an output file

```
> sqlite3 database-file
sqlite> .output outputfile.txt
sqlite>
```

this will send all result to the output file. if you only want to send result of one query, use .once

Ruby - Sinatra

```
end
```