The Architecture of SaaS Applications

Chung-Kil Hur

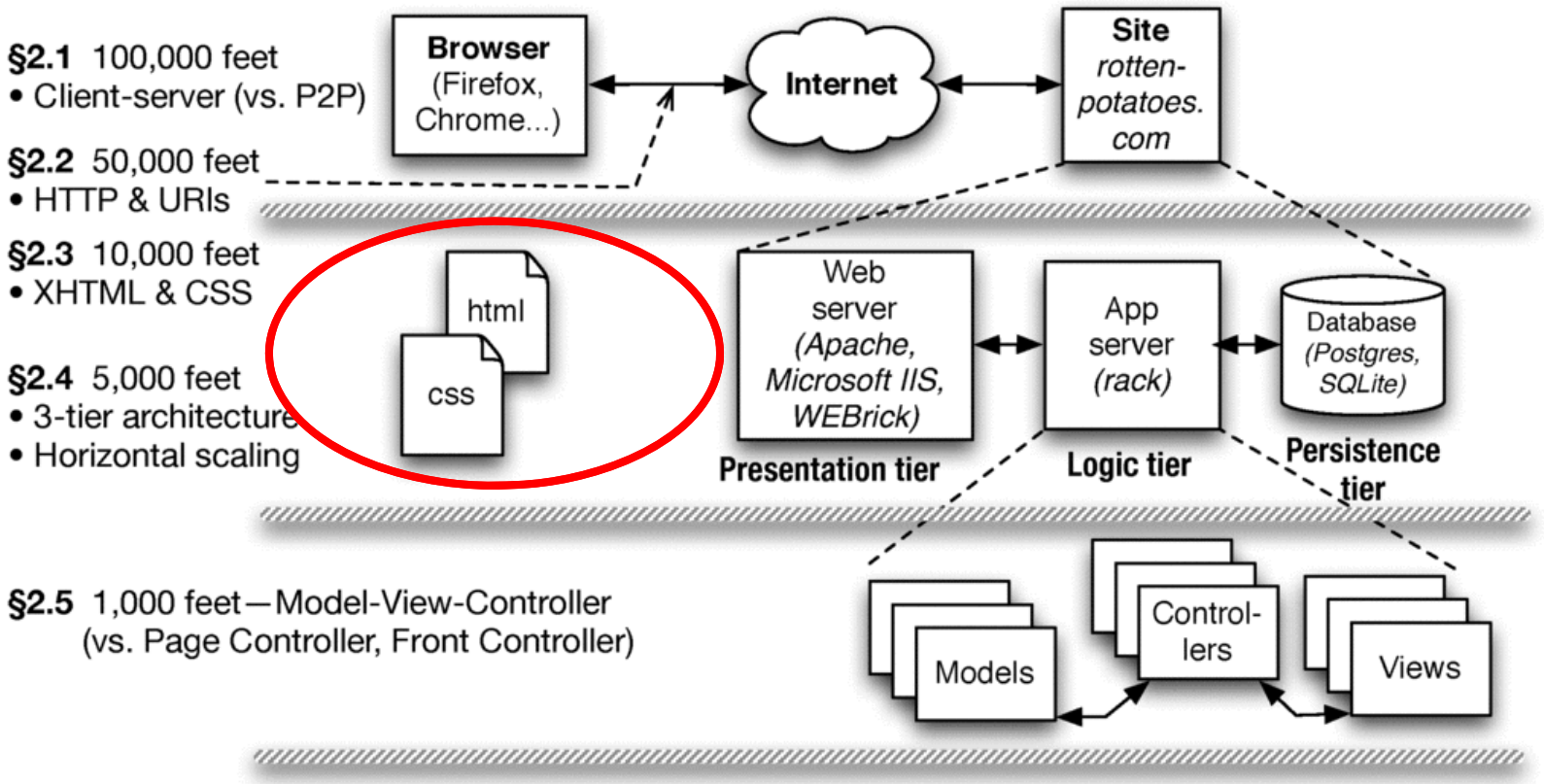
(Credit: Byung-Gon Chun & Many Slides from UCB CS169 taught by Armando Fox and David Patterson)

SWPP, CSE, SNU

Summary: Web 1.0 SaaS

- Browser *requests* web resource (URI) using HTTP
  - HTTP is a simple request-reply protocol that relies on TCP/IP
  - In SaaS, most URI's cause a program to be run, rather than a static file to be fetched
- *HTML* is used to encode content, *CSS* to style it visually
- *Cookies* allow server to track client
  - Browser automatically passes cookie to server on each request
  - Server may change cookie on each response
  - Typical usage: cookie includes a *handle* to server-side information
  - That's why some sites don't work if cookies are completely disabled
- *Frameworks* make all these abstractions convenient for programmers to use, without sweating the details
- ...and help map SaaS to 3-tier, shared-nothing architecture

HTML+CSS



§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)

- **Active Record**
- Data Mapper
- **REST**
- **Template View**
- Transform View

Introduction

This article is a review of the book Dietary Preferences of Penguins, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes three hard-to-swallow claims about penguins:

First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish

Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

```
<h1>Introduction</h1>
<p>
  This article is a review of the book
  <i>Dietary Preferences of Penguins</i>,
  by Alice Jones and Bill Smith. Jones and Smith's
  controversial work makes three hard-to-swallow claims
  about penguins:
</p>
<ul>
  <li>
   First, that penguins actually prefer tropical foods
   such as bananas and pineapple to their traditional diet
   of fish
  </li>
  <li>
   Second, that tropical foods give penguins an odor that
   makes them unattractive to their traditional predators
  </li>
</ul>
...
```

Introduction

This article is a review of the book *Dietary Preferences of Penguins*, by Alice Jones and Bill Smith. Jones and Smith's controversial work makes two hard-to-swallow claims about penguins:

- First, that penguins actually prefer tropical foods such as bananas and pineapple to their traditional diet of fish
- Second, that tropical foods give penguins an odor that makes them unattractive to their traditional predators

...

```
<h1>Introduction</h1>
<p>
This article is a review of the book
<i>Dietary Preferences of Penguins</i>,
by Alice Jones and Bill Smith. Jones
and Smith's controversial work makes
three hard-to-swallow claims about
penguins:
<ul>
<li>
First, ...
```

Hypertext Markup Language

- Document = Hierarchy of *elements*
  - inline (headings, tables, lists, paragraphs)
  - embedded (images, JavaScript)
  - forms—allow user to submit simple input (text, radio/check buttons, dropdown menus...)
- Elements delimited by `<tag>....</tag>`
  - Some have *content:* `<p>Hello world</p>`
  - Some have *attributes:* `<img src="http://...">`
  - `id` and `class` attributes useful for *styling*

Cascading Style Sheets (CSS) separate content from presentation

- `<link rel="stylesheet" href="http://..."/>` (inside `<head>` element): what stylesheet(s) go with this HTML page
- HTML `id` & `class` attributes important in CSS
  - *id* must be ***unique within this page***
  - same *class* can be attached to many elements

```
<div id="right" class="content">
  <p>
   I'm Gon. I teach SWPP and do
   research in Big Data, Cloud Computing
   and Mobile Systems and Security.
  </p>
</div>
```

CSS Selectors identify specific elements for styling

```
<div class="pageFrame" id="pageHead">
  <h1>
    Welcome,
    <span id="custName">Gon</span>
  </h1>
  <img src="welcome.jpg" id="welcome"/>
</div>
```

- tag name: `h1`
- class name: `.pageFrame`
- element ID: `#pageHead`

} both of these match the outer *div* above. Don't do this!

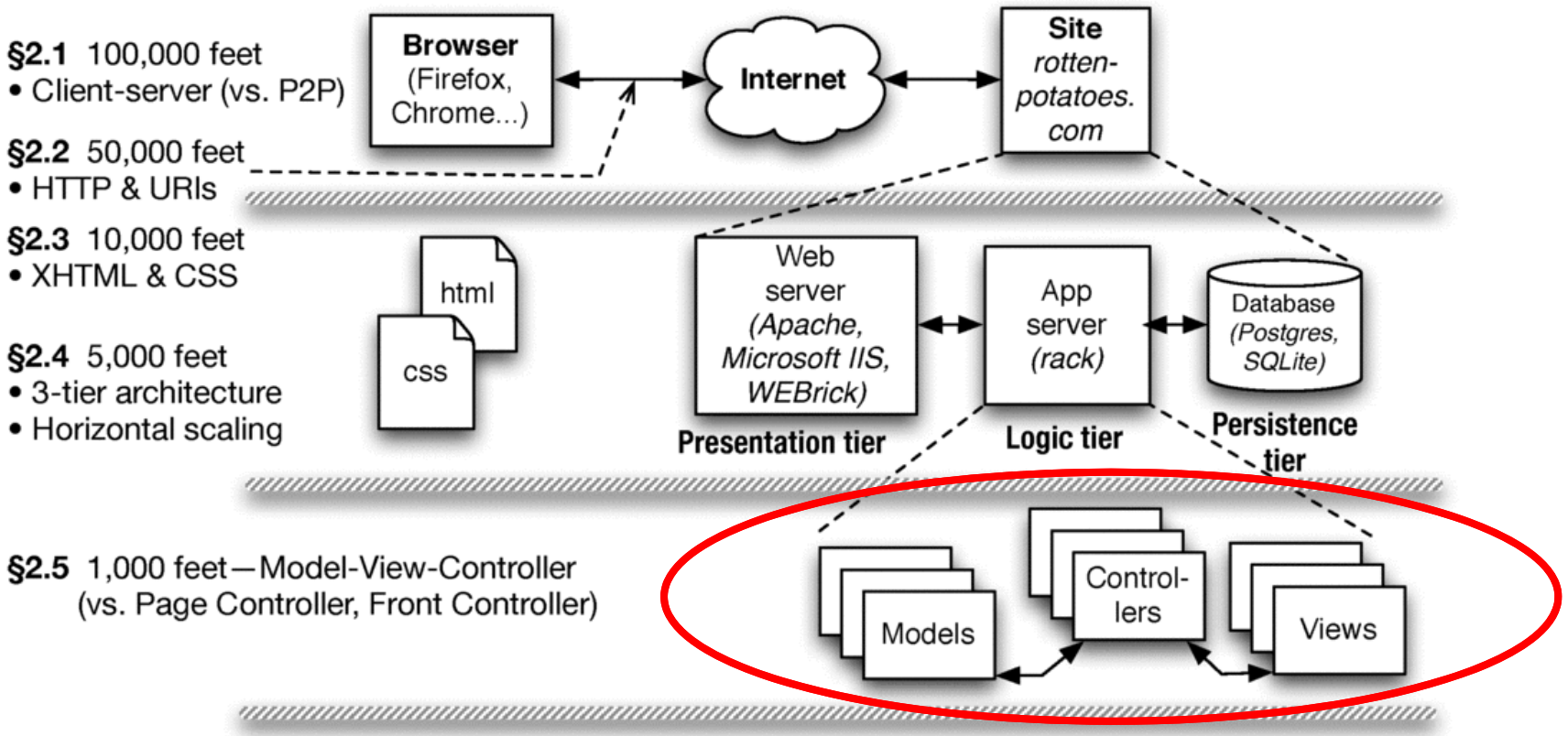
- tag name & class: `div.pageFrame`
- tag name & id: `img#welcome` (usually redundant)
- descendant relationship: `div .custName`
- Attributes *inherit* browser defaults unless overridden

Goal: HTML markup contains no visual styling information

Model-View-Controller

Whither frameworks?

- Is there common *application structure…*
- in *interactive user-facing* apps…
- …that could *simplify* app development if we captured them in a *framework?*



§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)

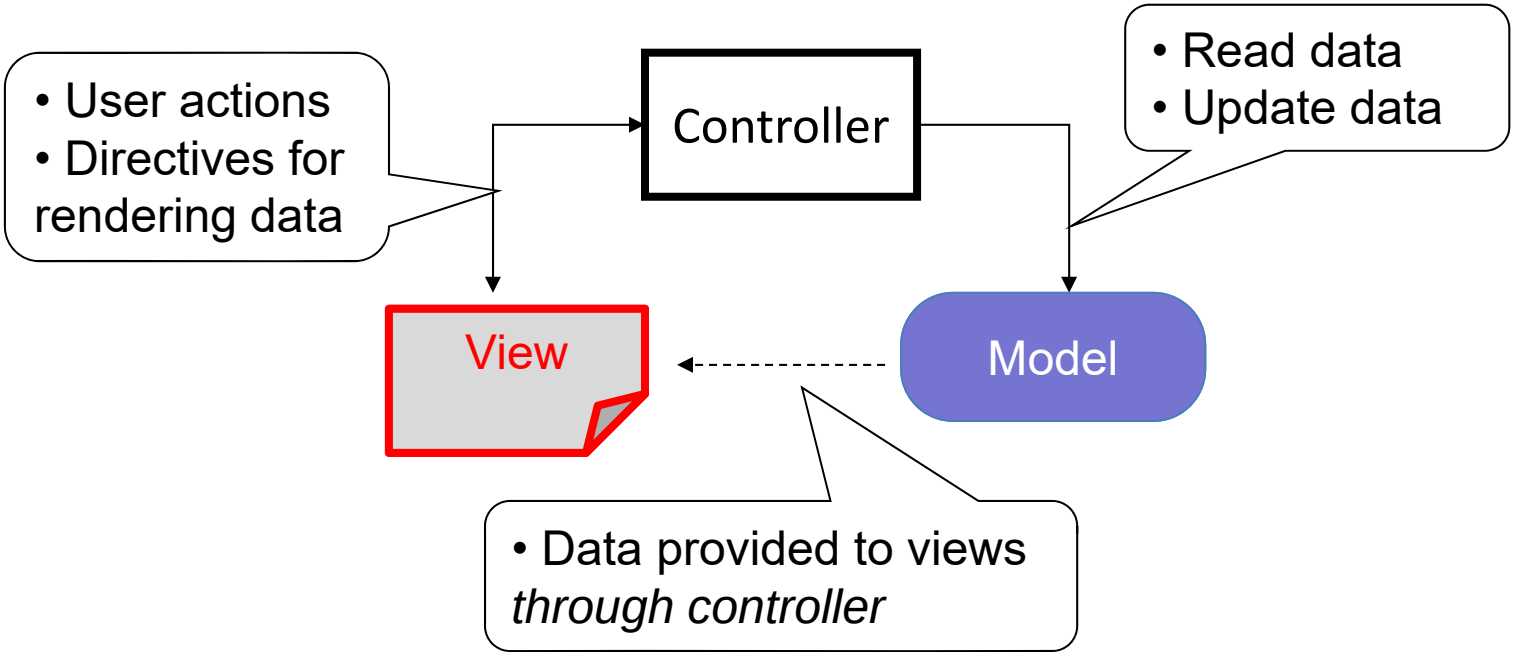
• **Active Record**
• Data Mapper

• **REST**

• **Template View**
• Transform View

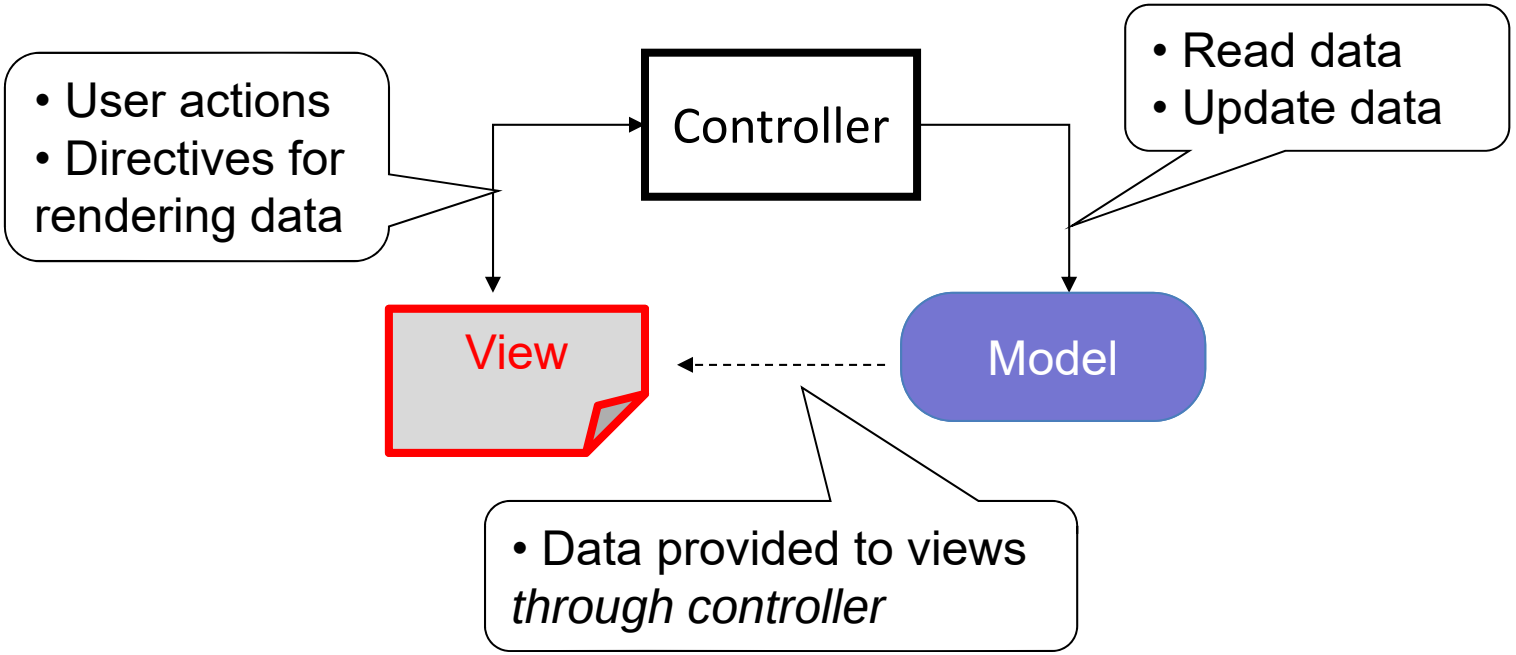
The MVC Design Pattern

- Goal: separate organization of data (model) from UI & presentation (view) by introducing *controller*
  - mediates user actions requesting access to data
  - presents data for *rendering* by the view



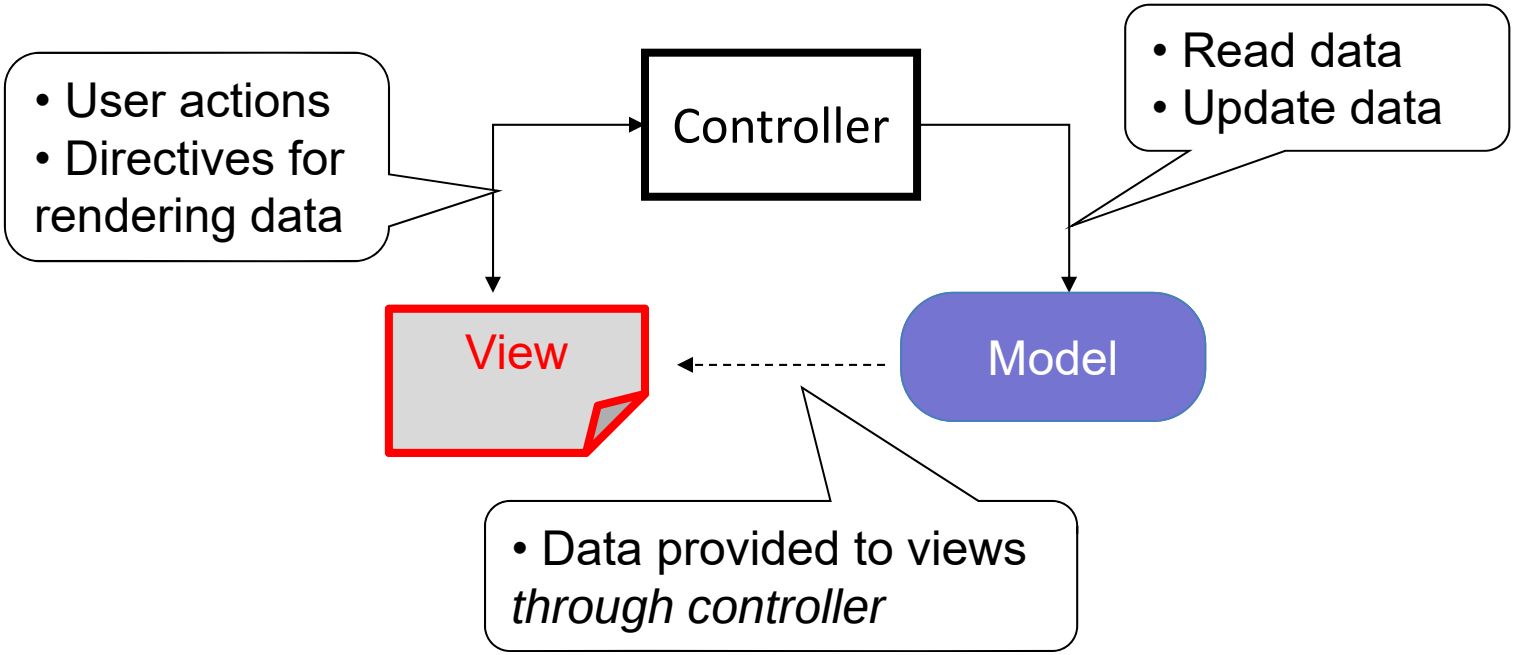
The MVC Design Pattern

- Model: data manipulated by the application
  - How to store it, how to operate on it, how to change it
  - Has a model for each type of entity manipulated by the app
  - Contain code that communicates with the storage tier



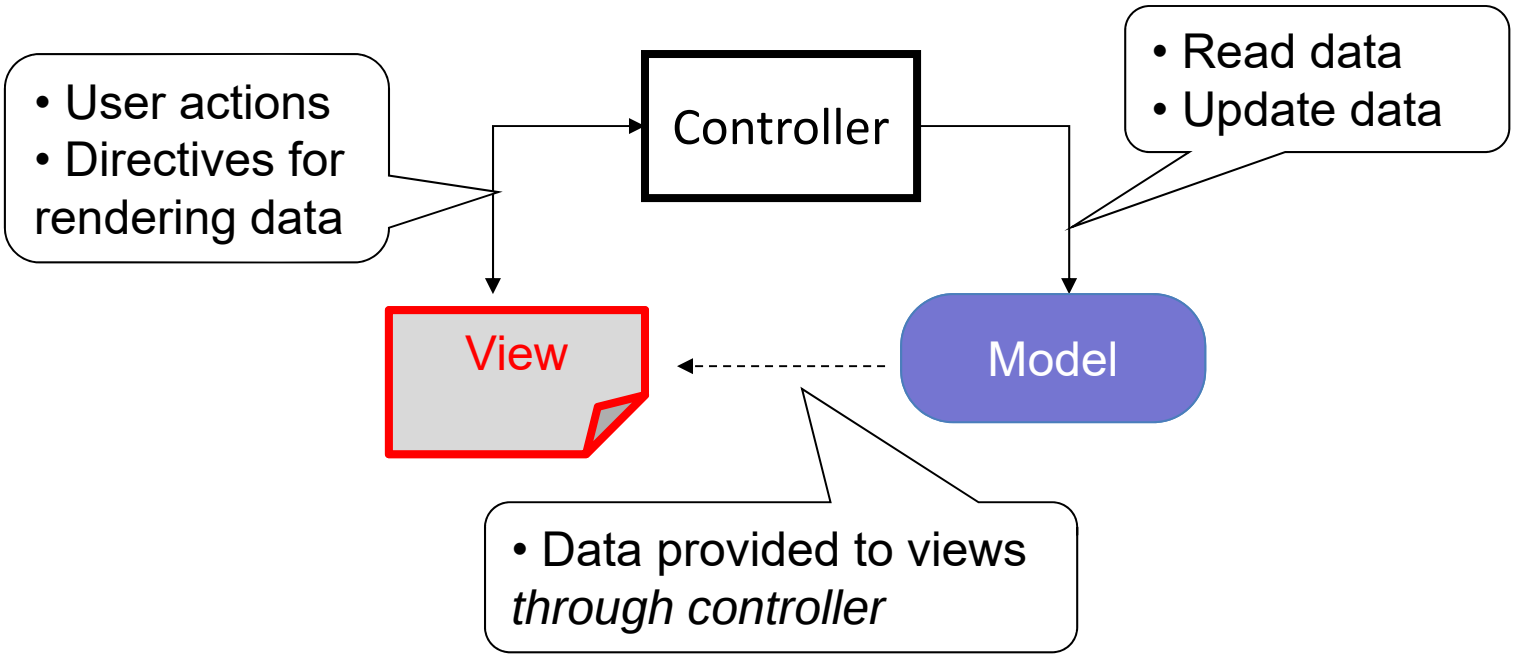
The MVC Design Pattern

- View: presented to the user and contain information about models with which the user can interact
  - The views serve as the interface between the system's users and its data
  - Model can be associated with a variety of views: one view lists all the movies, another view shows the details of a particular movie, etc.

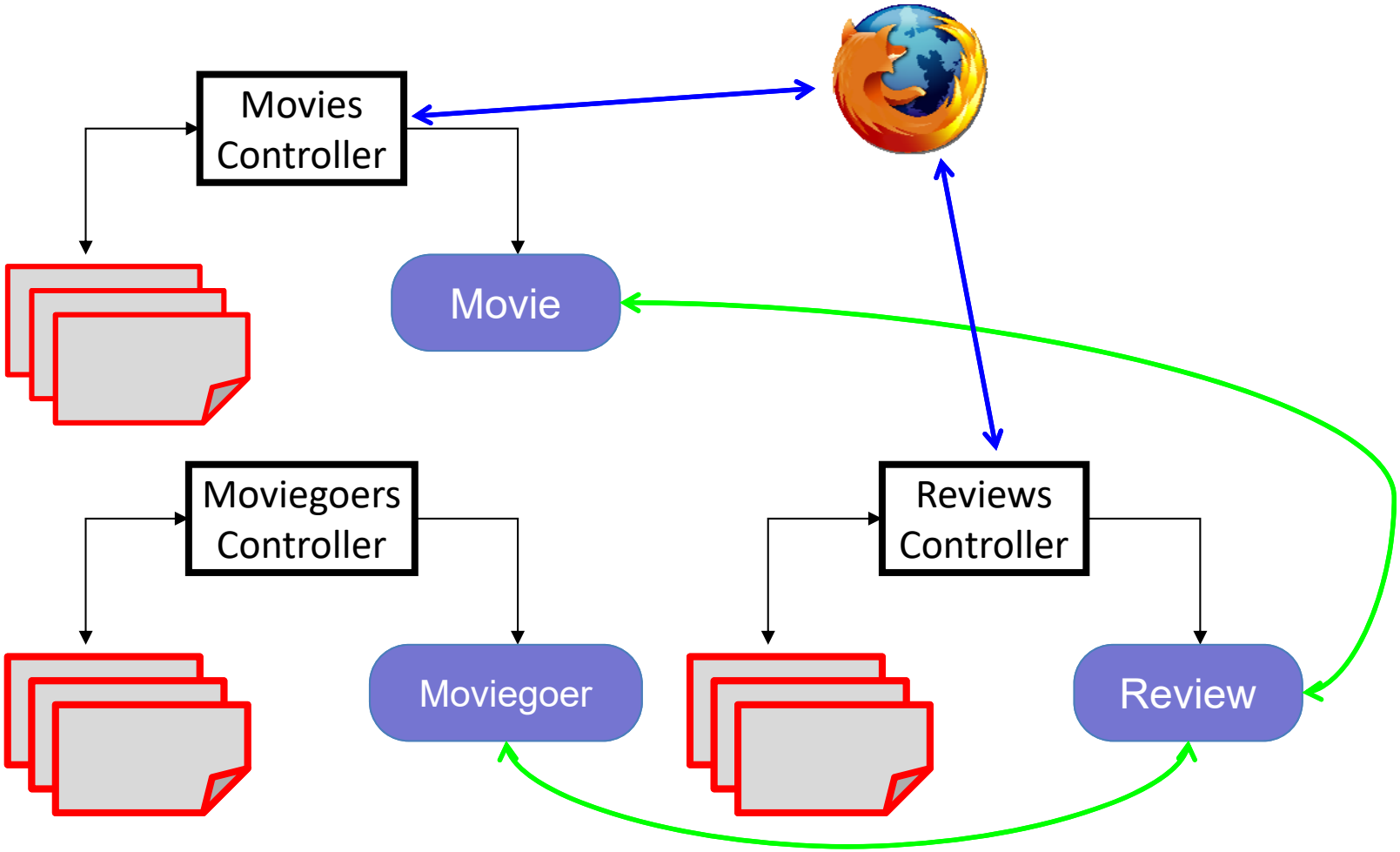


The MVC Design Pattern

- Controller: mediate the interaction in both ways
  - When a user interacts with a view (click something on a Web page), a specific controller action corresponding to that user activity is invoked
  - Each controller corresponds to one model.
  - The controller can ask the model to retrieve or modify information; depending on the results the controller decides what view will be presented next to the user and supplies that view with any necessary information.
  - The actions defined in the controller handle each type of user interaction with the view and contain the necessary logic to obtain Model data to render any of the views.

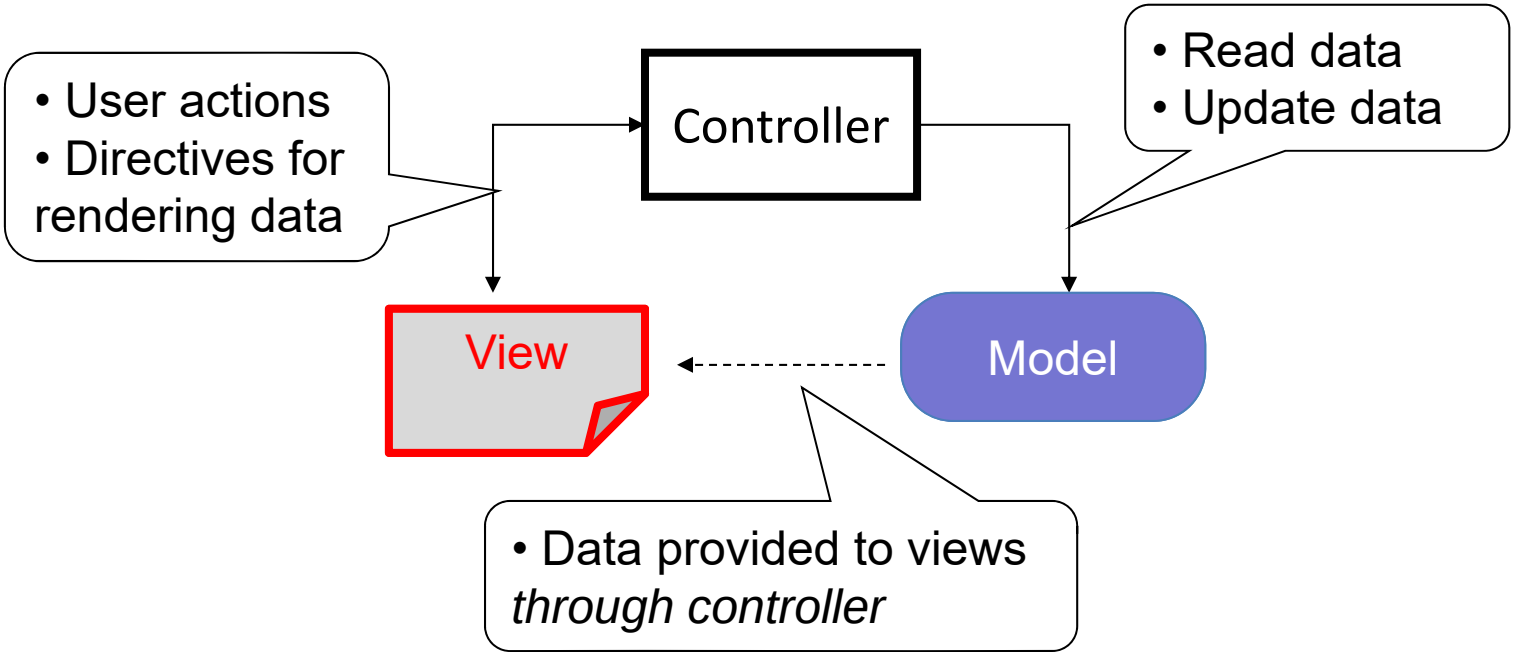


Each entity has a model, controller, & set of views



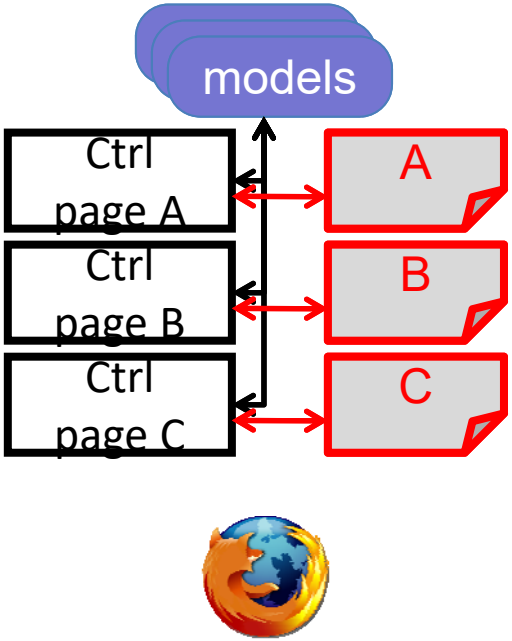
The MVC Design Pattern

- Goal: separate organization of data (model) from UI & presentation (view) by introducing *controller*
- Web apps may seem "obviously" MVC by design, but other alternatives are possible...

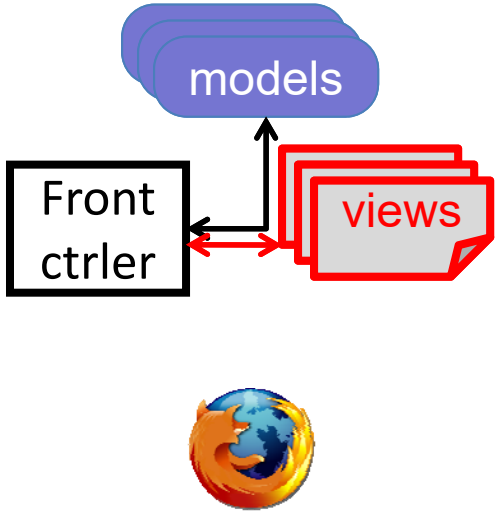


Alternatives to MVC

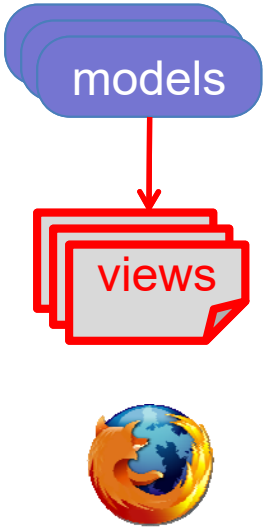
Page Controller
(Ruby Sinatra)



Front Controller
(J2EE servlet)



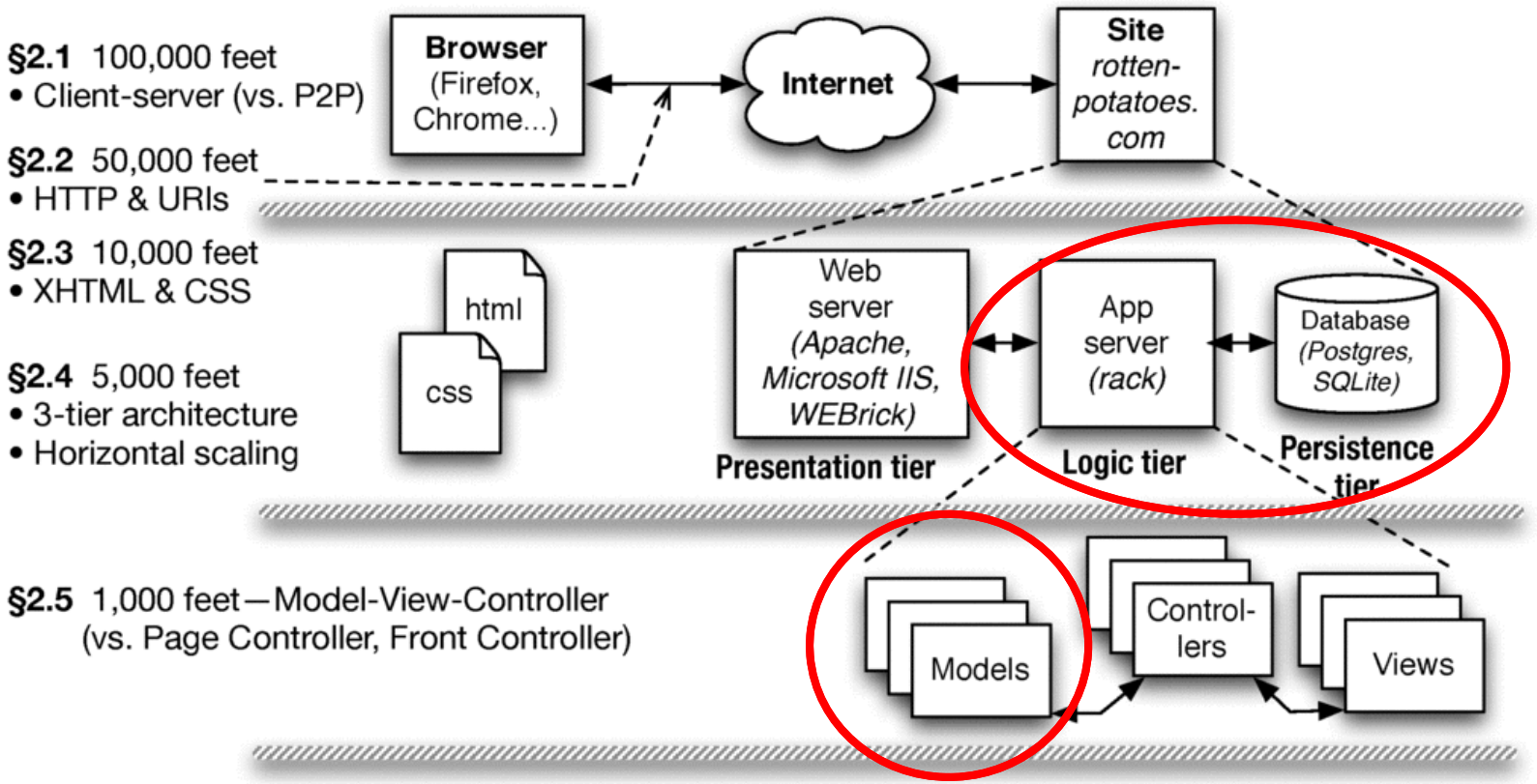
Template View
(PHP)



Rails supports SaaS apps structured as MVC, but other architectures may be better fit for some apps.

Models, Databases, and Active Record

- How should we store and retrieve *record-oriented structured* data?
- What is the relationship between data *as stored* and data *as manipulated in a programming language?*

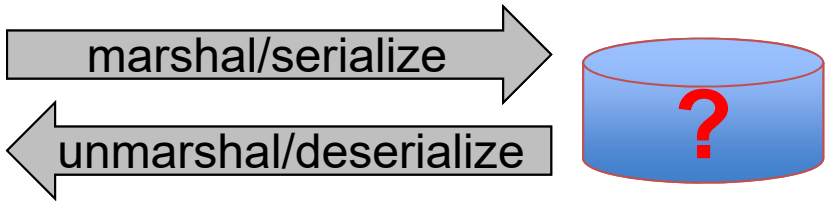


§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)

- **Active Record**
- Data Mapper
- **REST**
- **Template View**
- Transform View

In-Memory vs. In-Storage objects

```
#<Movie:0x1295580>
m.name, m.rating, ...
 #<Movie:0x32ffe416>
m.name, m.rating, ...
```



- How to represent persisted object in storage
  - Example: `Movie` with `name` & `rating` attributes
- Basic operations on object: **CRUD** (Create, Read, Update, Delete)
- ActiveRecord: every model knows how to CRUD itself, using common mechanisms

Rails Models Store Data in Relational Databases (RDBMS)

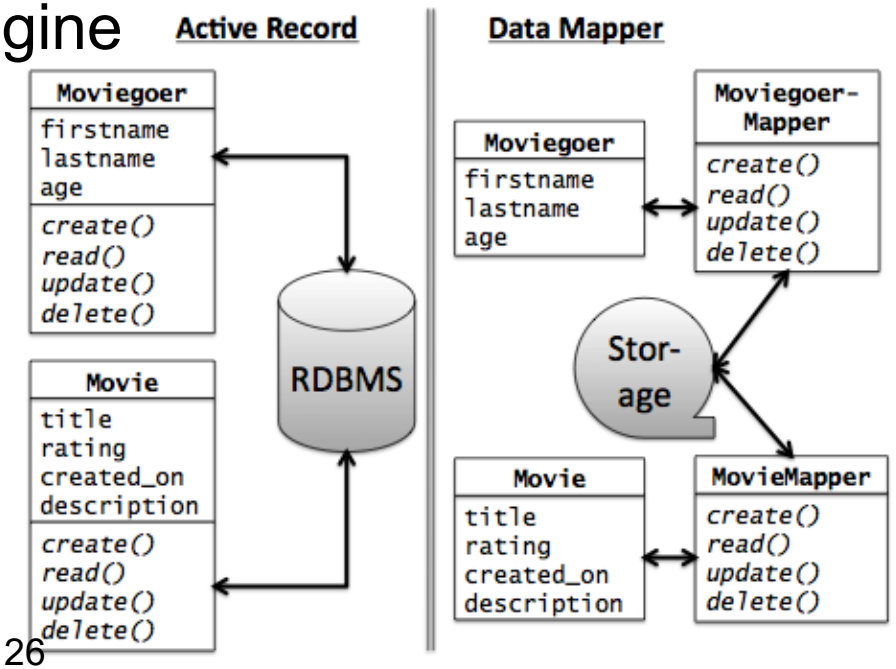
- Each type of model gets its own database *table*
  - All rows in table have identical structure
  - one row in table == one instance of model's class
  - Each column stores value of an *attribute* of the model
  - Each row has unique value for *primary key* (by convention, in Rails this is an integer and is called *id*)

id	rating	title	release_date
2	G	Gone With the Wind	1939-12-15
11	PG	Casablanca	1942-11-26
...	...	...	...
35	PG	Star Wars	1977-05-25

- *Schema:* Collection of all tables and their structure

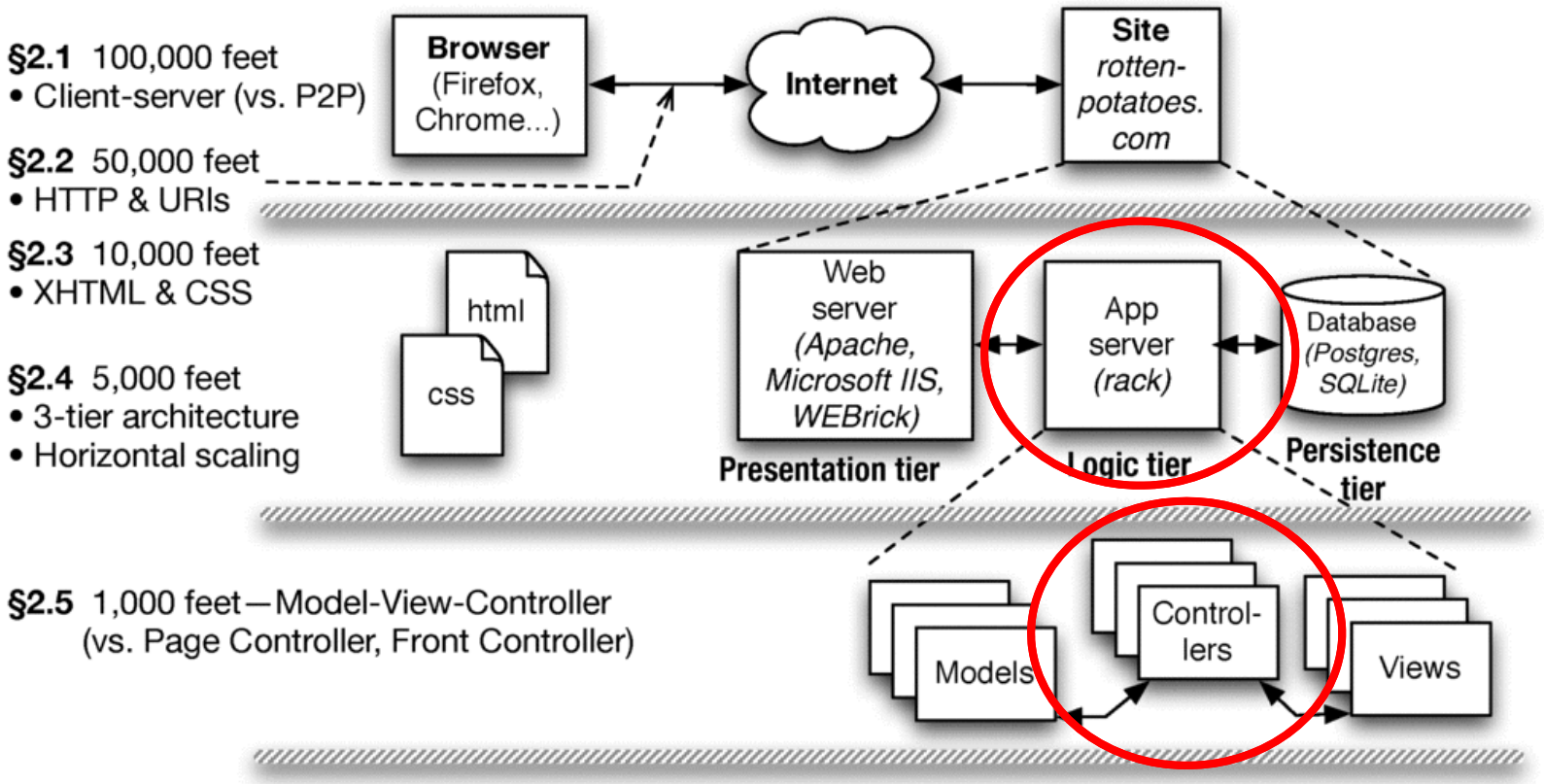
Alternative: DataMapper

- Data Mapper associates separate *mapper* with each model
  - Idea: keep mapping *independent* of particular data store used => works with more types of databases
  - Used by Google AppEngine
  - Con: can't exploit RDBMS features to simplify complex queries & relationships



Controllers, Routes, and RESTfulness

- What design decisions would allow our app to support Service-Oriented Architecture?



§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)

• **Active Record**
• Data Mapper

• **REST**

• **Template View**
• Transform View

REST (Representational State Transfer)—R. Fielding, 2000

- Idea: URI names *resource*, not *page*
  - ***Self-contained***: which *resource,* and what to do to it
  - Responses include hyperlinks to discover additional RESTful resources
  - "a *post hoc* [after the fact] *description of the features that made the Web successful"*
- A service (in the SOA sense) whose operations are like this is a RESTful service
- Ideally, RESTful URIs name the operations

Non-RESTful site URI vs. RESTful site URI

	Non-RESTful site URI
Login to site	POST /login/dave
Welcome page	GET /welcome
Add item ID 427 to cart	POST /add/427
View cart	GET /cart
Checkout	POST /checkout

REST example

With Google Translate, you can dynamically translate text between thousands of language pairs. The Google Translate API lets websites and programs integrate with Google Translate programmatically.

Google Translate API

Feedback on this document

Pricing

Getting Started

Using REST

Using REST

Google Translate API is a paid service. For website translations, we encourage you to use the Google Website Translator gadget.

```
GET https://www.googleapis.com/language/translate/v2?key=INSERT-YOUR-KEY&target=de&q=Hello%20world
```

If the request succeeds, the server responds with a `200 OK` HTTP status code and the data properties:

JSON

```
200 OK

{
    "data": {
        "translations": [
            {
                "translatedText": "Hallo Welt",
                "detectedSourceLanguage": "en"
            }
        ]
    }
}
```

Routes

- In MVC, each interaction the user can do is handled by a *controller action*
- A *route* maps `<HTTP method, URI>` to controller action

Route	Action
`GET /movies/3`	Show info about movie whose ID=3
`POST /movies`	Create new movie from attached form data
`PUT /movies/5`	Update movie ID 5 from attached form data
`DELETE /movies/5`	Delete movie whose ID=5

Brief Intro to Rails' Routing Subsystem

- dispatch <method,URI> to correct controller action
- parses query *parameters* from both URI and form submission into a convenient hash
- Built-in shortcuts to generate all CRUD routes (though most apps will also have other routes)

```
rake routes
I     GET /movies           {:action=>"index", :controller=>"movies"}
C    POST /movies           {:action=>"create", :controller=>"movies"}
     GET /movies/new        {:action=>"new", :controller=>"movies"}
     GET /movies/:id/edit   {:action=>"edit", :controller=>"movies"}
R    GET /movies/:id        {:action=>"show", :controller=>"movies"}
U    PUT /movies/:id        {:action=>"update", :controller=>"movies"}
D DELETE /movies/:id        {:action=>"destroy", :controller=>"movies"}
```

GET /movies/3/edit HTTP/1.0

- Matches route:

```
GET /movies/:id/edit {:action=>"edit", :controller=>"movies"}
```

- Parse wildcard parameters: `params[:id] = "3"`
- Dispatch to `edit` method in `movies_controller.rb`
- To include a URI in generated view that will submit the form to the update controller action with `params[:id]==3`, call helper:

```
update_movie_path(3) # => PUT /movies/3
```

rake routes

```
I     GET /movies           {:action=>"index", :controller=>"movies"}
C    POST /movies           {:action=>"create", :controller=>"movies"}
      GET /movies/new       {:action=>"new", :controller=>"movies"}
      GET /movies/:id/edit  {:action=>"edit", :controller=>"movies"}
R     GET /movies/:id       {:action=>"show", :controller=>"movies"}
U     PUT /movies/:id       {:action=>"update", :controller=>"movies"}
D DELETE /movies/:id        {:action=>"destroy", :controller=>"movies"}
```