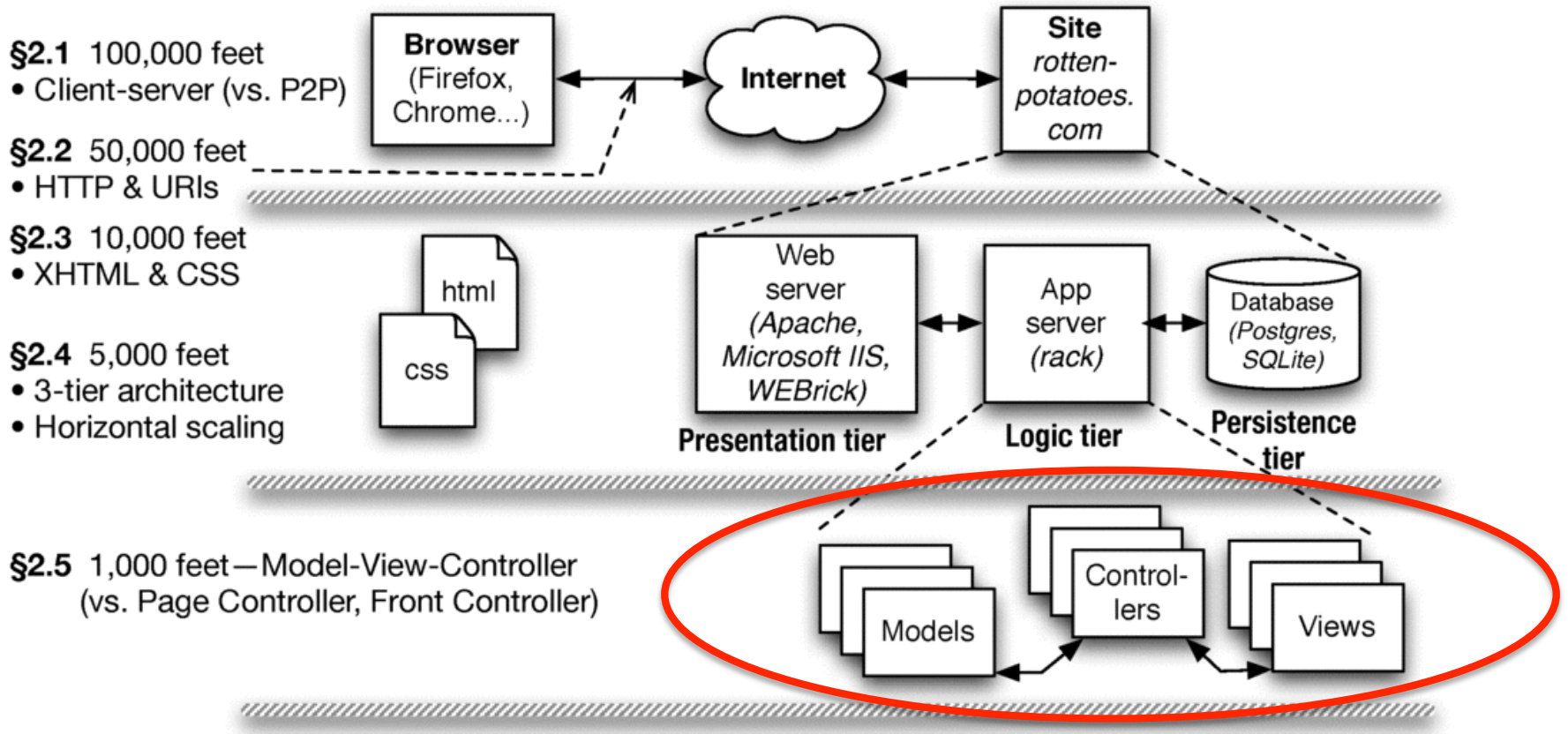
Model-View-Controller

Engineering Software as a Service §2.5

Armando Fox

Whither frameworks?

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

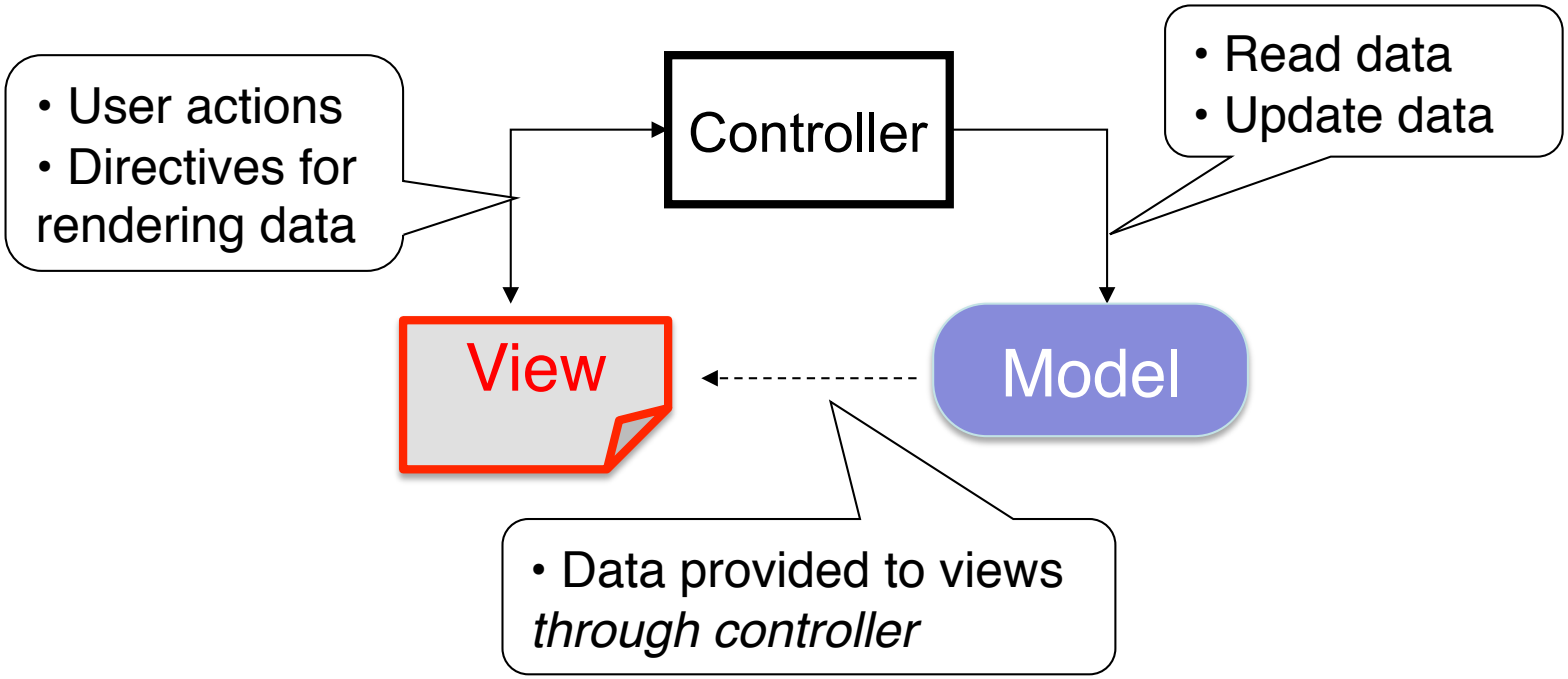


§2.6 500 feet: Active Record models (vs. Data Mapper)
§2.7 500 feet: RESTful controllers (Representational State Transfer for self-contained actions)
§2.8 500 feet: Template View (vs. Transform View)

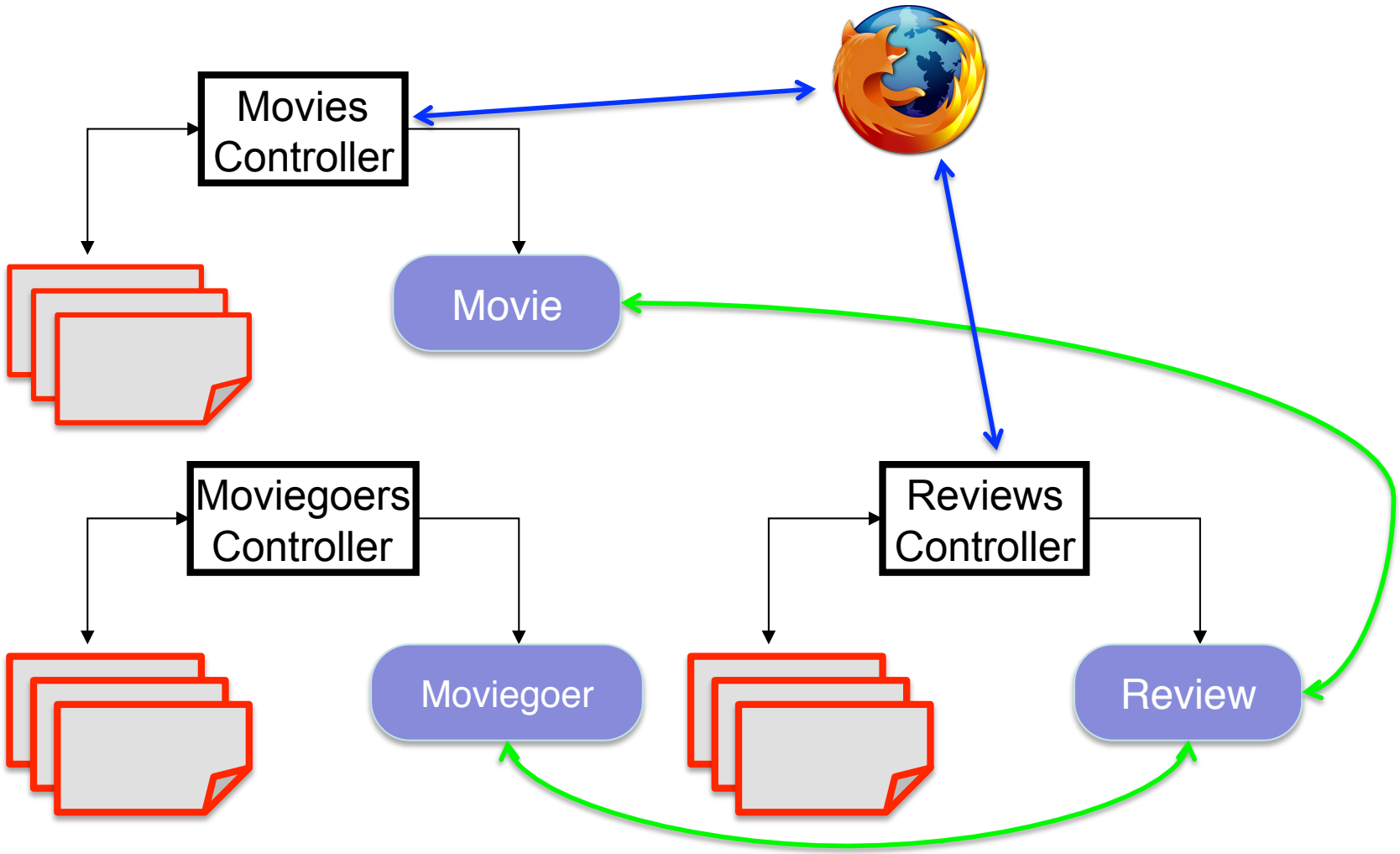
- **Active Record**
- Data Mapper
- **REST**
- **Template View**
- Transform View

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
- Web apps may seem "obviously" MVC by design, but other alternatives are possible...

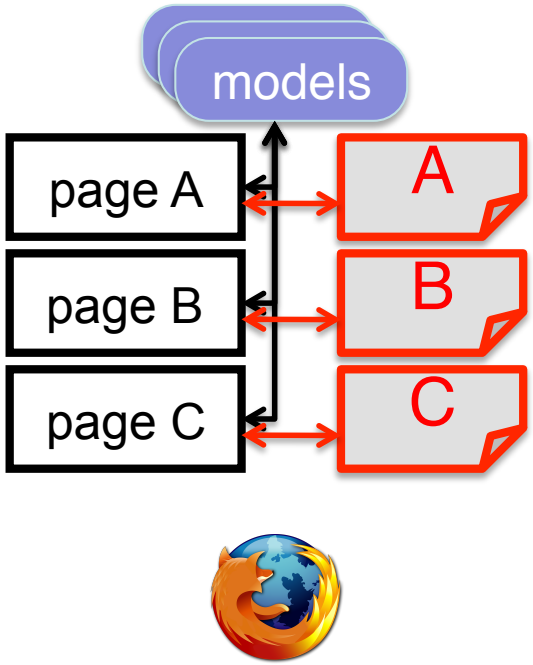


Each entity has a model, controller, & set of views

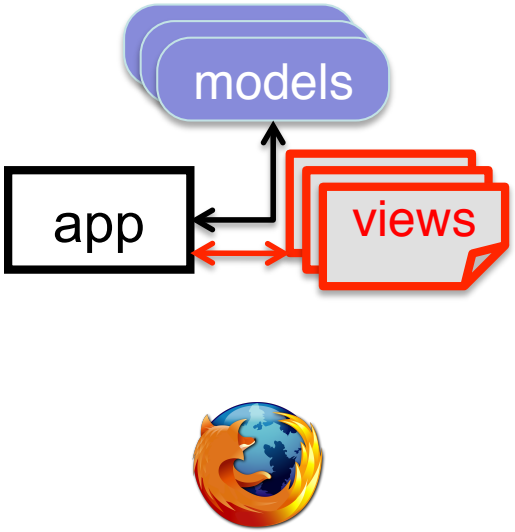


Alternatives to MVC

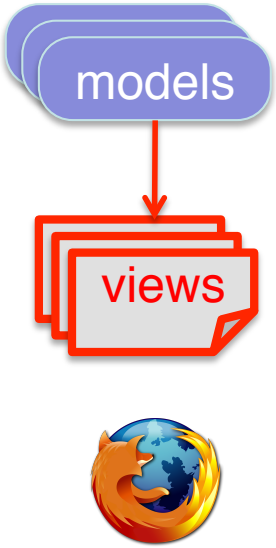
Page Controller (Ruby Sinatra)



Front Controller (J2EE servlet)



Template View (PHP)



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

Which statement is **NOT** true about the Model-View-Controller (MVC) architectural pattern:

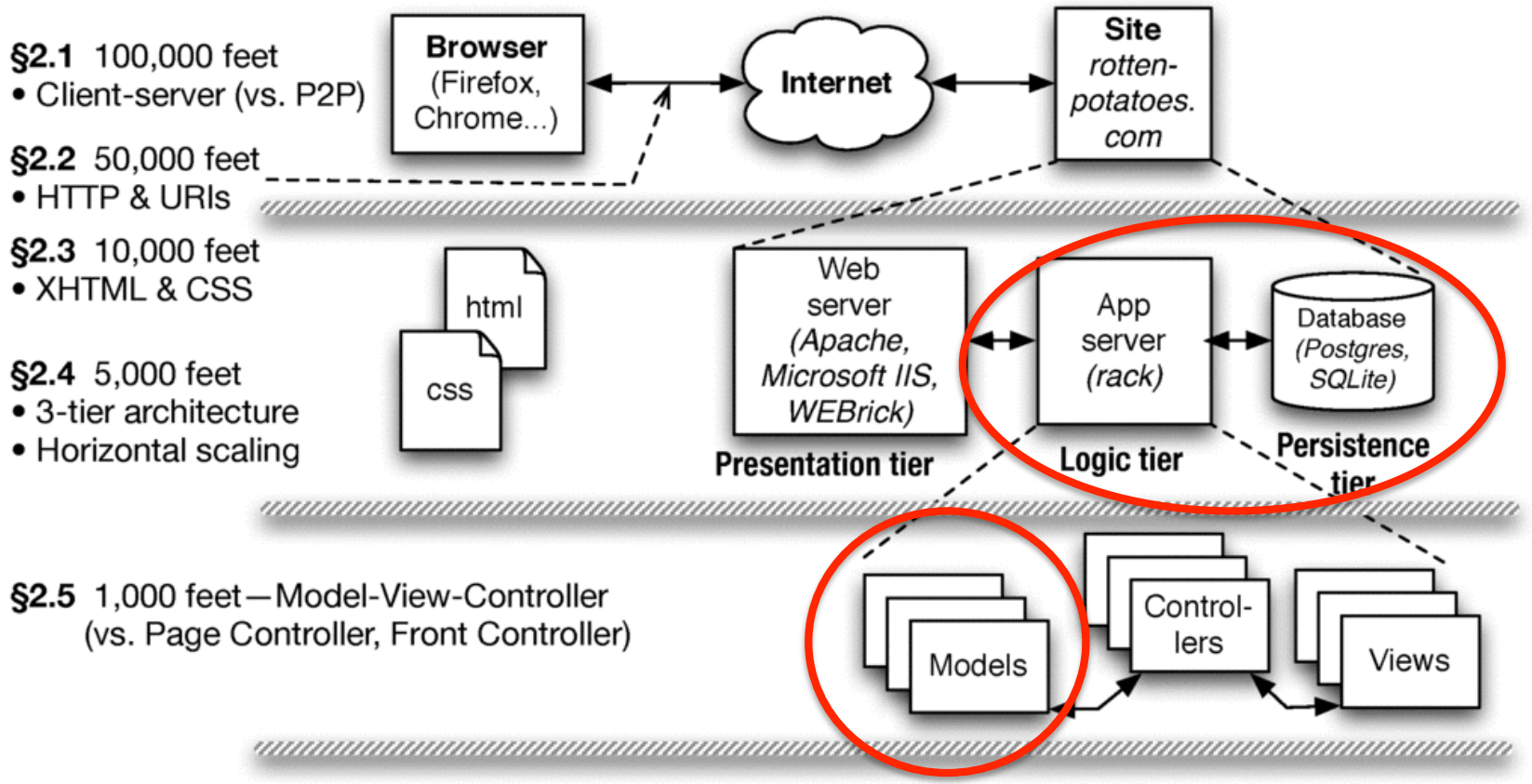
- ☐ In SaaS apps on the Web, controller actions and view contents are transmitted using HTTP.
- ☐ All MVC apps have both a "client" part (e.g. Web browser) and a "cloud" part (e.g. Rails app on cloud).
- ☐ Model-View-Controller is just one of several possible ways to structure a SaaS app.
- ☐ Peer-to-peer apps (vs. client-server apps) can be structured as Model-View-Controller.

Models, Databases, and Active Record

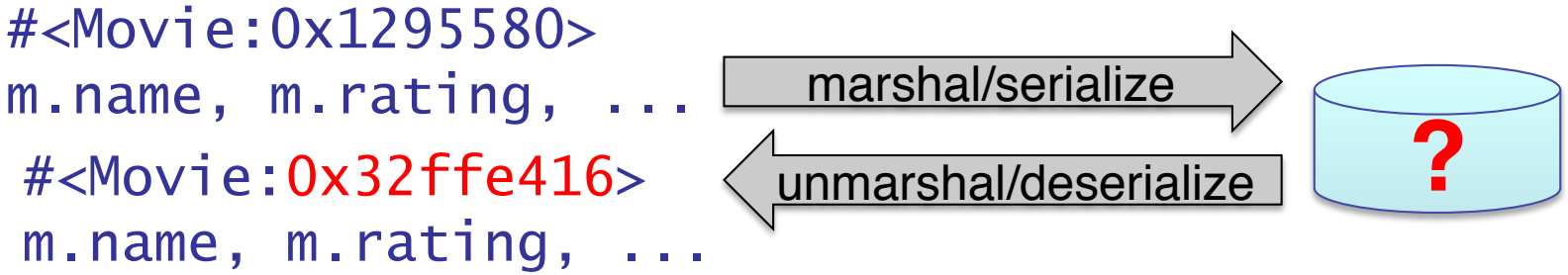
*Engineering Software as a Service §*2.6

Armando Fox

- 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?*


§2.1 100,000 feet
• Client-server (vs. P2P)
§2.2 50,000 feet
• HTTP & URIs
§2.3 10,000 feet
• XHTML & CSS
§2.4 5,000 feet
• 3-tier architecture
• Horizontal scaling
§2.5 1,000 feet—Model-View-Controller
(vs. Page Controller, Front Controller)
Browser
(Firefox, Chrome...)
Internet
Site
rotten-potatoes.com
html
css
Web server
(Apache, Microsoft IIS, WEBrick)
App server
(rack)
Database
(Postgres, SQLite)
Presentation tier
Logic tier
Persistence tier
Models
Control-lers
Views
§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



- 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)

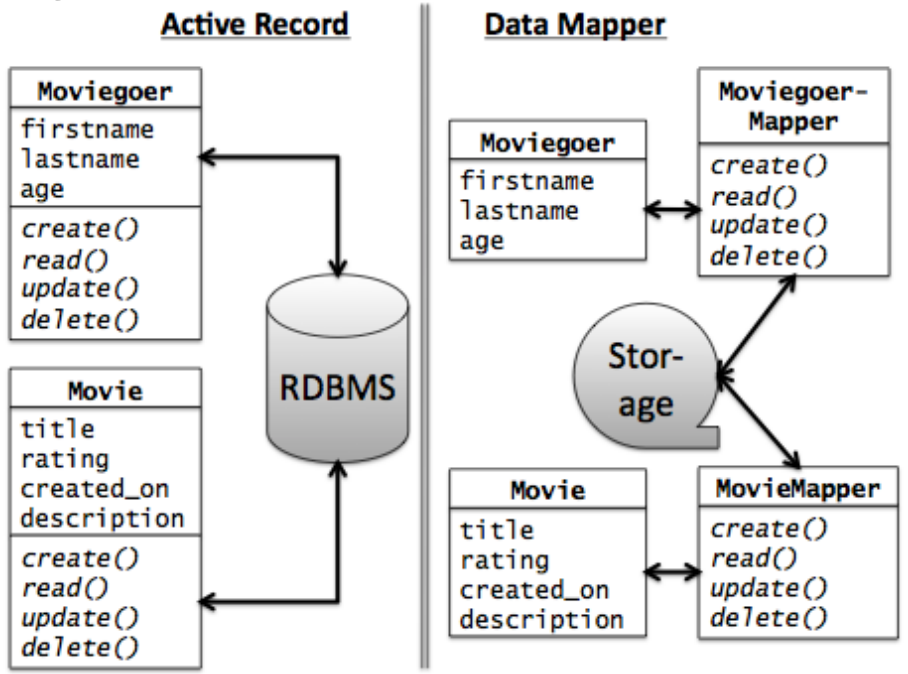
- Each type of model gets its own database *table*
  - All rows in table have identical structure
  - one row in table == one instance of model's class
  - Each column stores value of an *attribute* of the model
  - Each row has unique value for *primary key* (by convention, in Rails this is an integer 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
- We'll revisit when talking about *associations*



Which statement is ***not*** true about the Model in Model-View-Controller:

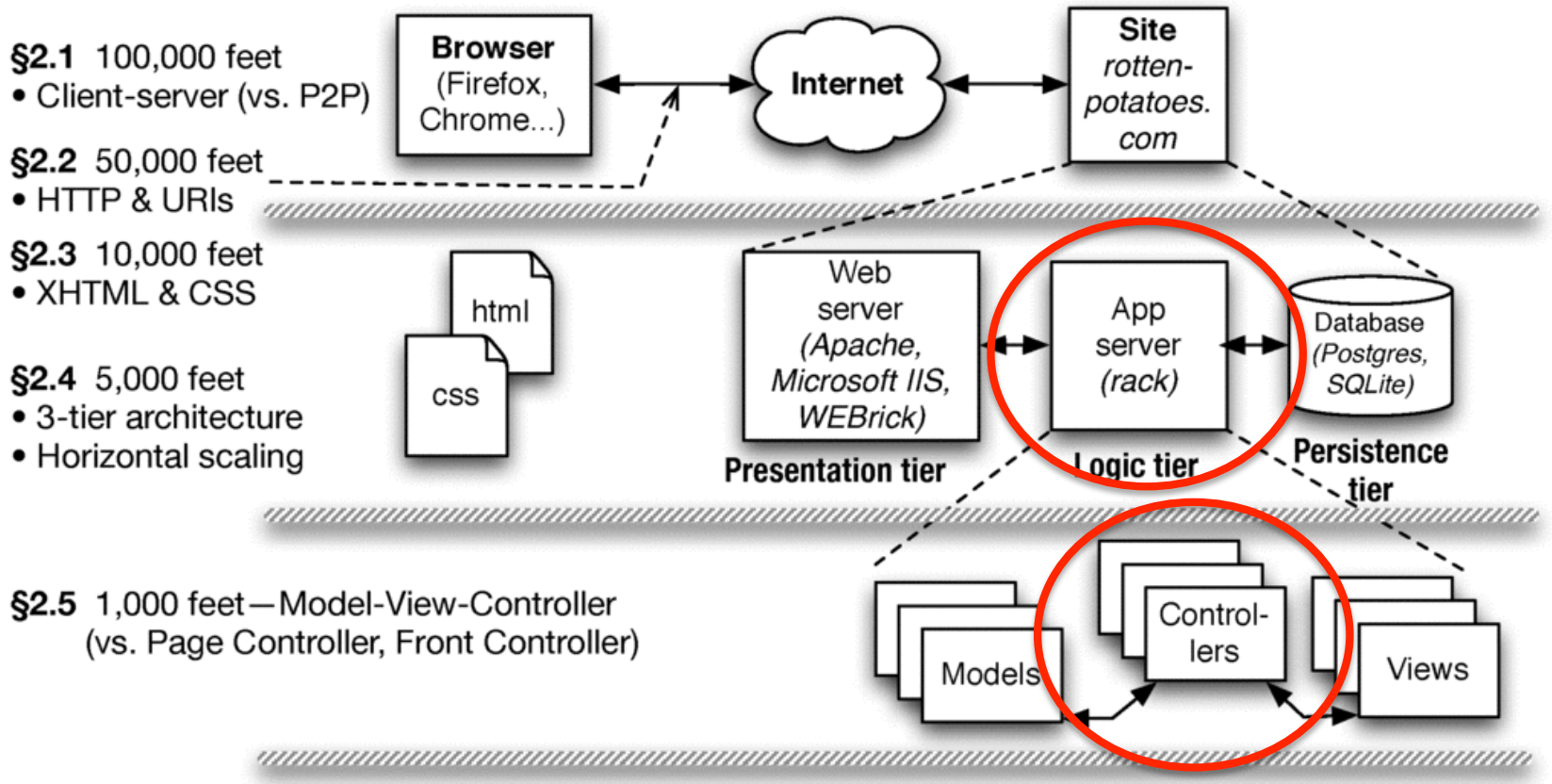
- ☐ The CRUD actions only apply to models backed by a database that supports ActiveRecord.
- ☐ Part of the Model's job is to convert between in-memory and stored representations of objects.
- ☐ Although Model data is displayed by the View, a Models' direct interaction is with Controllers.
- ☐ Although DataMapper doesn't use relational databases, it's a valid way to implement a Model.

Controllers, Routes, and RESTfulness

Engineering Software as a Service §2.7

Armando Fox

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


§2.1 100,000 feet
• Client-server (vs. P2P)
§2.2 50,000 feet
• HTTP & URIs
§2.3 10,000 feet
• XHTML & CSS
§2.4 5,000 feet
• 3-tier architecture
• Horizontal scaling
Browser (Firefox, Chrome...)
Internet
Site rotten-potatoes.com
html
css
Web server (Apache, Microsoft IIS, WEBrick)
App server (rack)
Database (Postgres, SQLite)
Presentation tier
Logic tier
Persistence tier
§2.5 1,000 feet—Model-View-Controller (vs. Page Controller, Front Controller)
Models
Control-lers
Views
§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* or *action*
  - *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

Routes

- In MVC, each interaction the user can do is handled by a *controller action*
  - Ruby method that handles that interaction
- 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
- provides *helper methods* that generate a <method,URI> pair given a 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"}
```

Which statement is **NOT** true regarding Rails RESTful routes and the resources to which they refer:

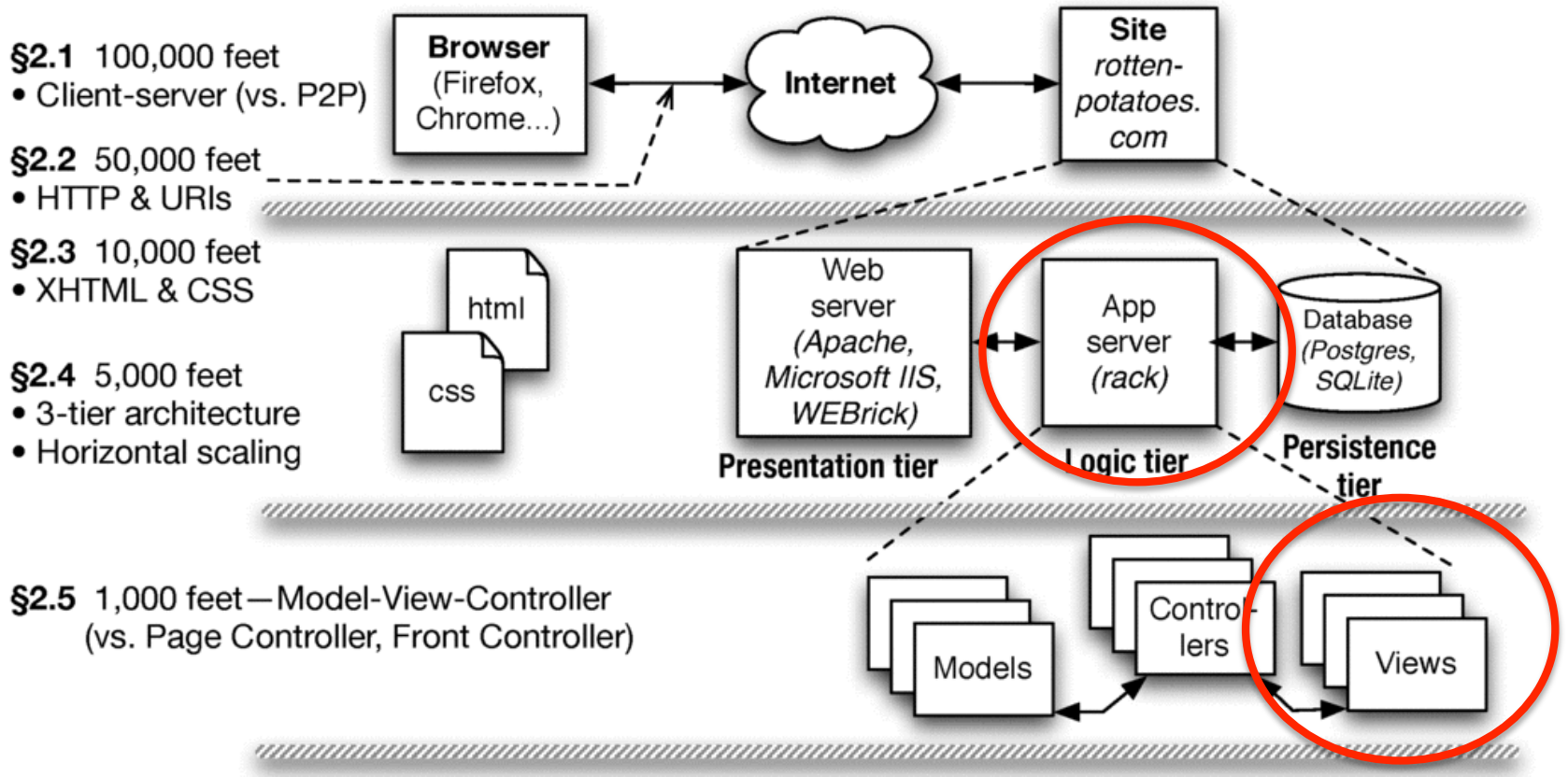
- ☐ A *resource* may be existing content or a request to modify something.
- ☐ In an MVC app, every route must eventually trigger a controller action.
- ☐ One common set of RESTful actions is the CRUD actions on models.
- ☐ The route always contains one or more "wildcard" parameters, such as `:id`, to identify the particular resource instance in the operation

Template Views and Haml

*Engineering Software as a Service §*2.8

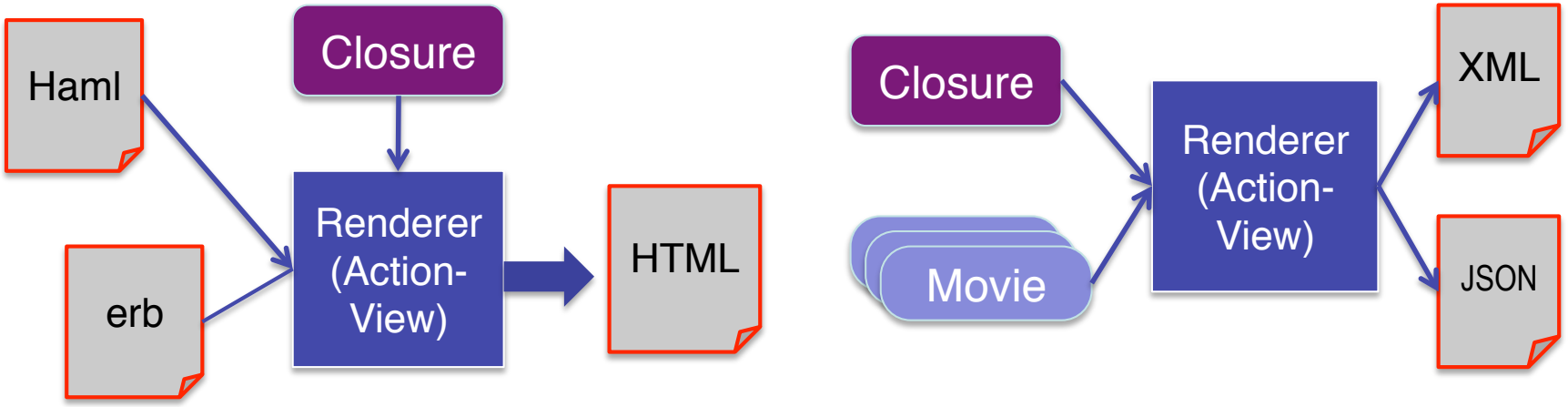
Armando Fox

- HTML is how we must present content to browsers…
- …but what's the process by which our app's output becomes HTML?


§2.1 100,000 feet
• Client-server (vs. P2P)
§2.2 50,000 feet
• HTTP & URIs
§2.3 10,000 feet
• XHTML & CSS
§2.4 5,000 feet
• 3-tier architecture
• Horizontal scaling
Browser (Firefox, Chrome...)
Internet
Site rotten-potatoes.com
html
css
Web server (Apache, Microsoft IIS, WEBrick)
App server (rack)
Database (Postgres, SQLite)
Presentation tier
Logic tier
Persistence tier
§2.5 1,000 feet—Model-View-Controller (vs. Page Controller, Front Controller)
Models
Controllers
Views
§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

Template View pattern

- View consists of markup with selected *interpolation* to happen at runtime
  - Usually, values of variables or result of evaluating short bits of code
- In Elder Days, this *was* the app (e.g. PHP)
- *Alternative:* Transform View



Haml is HTML on a diet

```
%h1.pagename All Movies
%table#movies
  %thead
    %tr
      %th Movie Title
      %th Release Date
      %th More Info
  %tbody
    - @movies.each do |movie|
      %tr
        %td= movie.title
        %td= movie.release_date
        %td= link_to "More on #{movie.title}",  |
          movie_path(movie) |
= link_to 'Add new movie', new_movie_path
```

Don't put code in your views

- Syntactically, you can put any code in view
- But MVC advocates thin views & controllers
  - Haml makes deliberately awkward to put in lots of code
- *Helpers* (methods that "prettify" objects for including in views) have their own place in Rails app
- Alternative to Haml: html.erb (Embedded Ruby) templates, look more like PHP

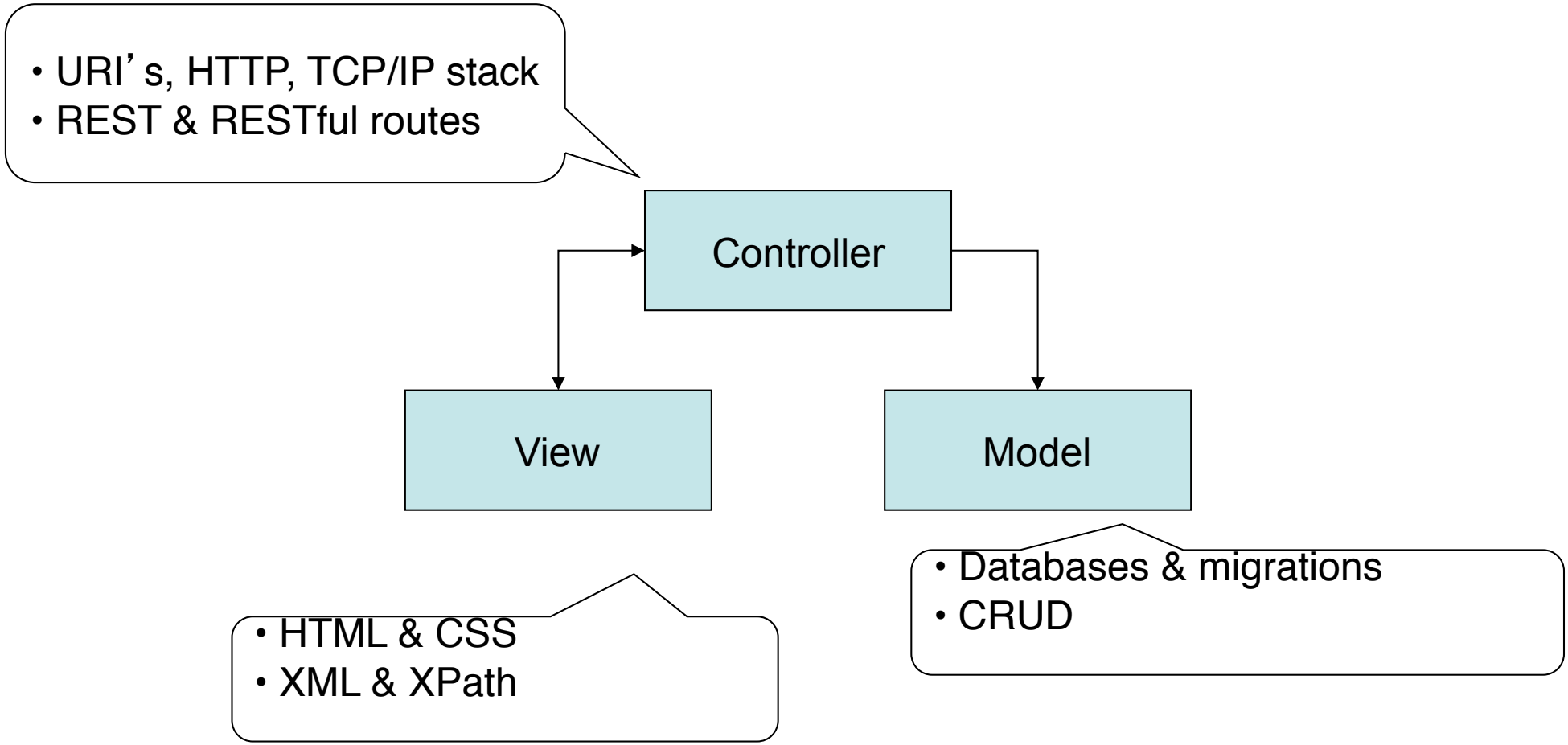
What happens if you embed code in your Rails views that tries to *directly access the model* (in the database)?

- ☐ It will work, but it's bad form and violates the MVC guidelines
- ☐ It will work when developing against a "toy" database, but not in production
- ☐ It won't work, because Views can't communicate directly with Models
- ☐ Behavior varies depending on the app

Summary & Reflections: SaaS Architecture (*Engineering Software as a Service* §2.9-2.10)

Armando Fox

The big picture (technologies)



c. 2008: "Rails doesn't scale"

- Scalability is an *architectural* concern—not confined to language or framework
- The stateless tiers of 3-tier arch *do scale*
  - With cloud computing, just worry about constants
- Traditional relational databases *do not scale*
- Various solutions combining relational and non-relational storage ("NoSQL") *scale much better*
  - DataMapper works well with some of them
- Intelligent use of *caching* (later in course) can greatly improve the constant factors

Frameworks, Apps, Design patterns

- Many design patterns so far, more to come
- *In 1995, it was the wild west:* biggest Web sites were minicomputers, *not* 3-tier/cloud
- Best practices (patterns) "extracted" from experience and captured in frameworks
- But API's transcended it: 1969 protocols + 1960s markup language + 1990 browser + 1992 Web server works in 2011

Architecture is about Alternatives

Pattern we're using	Alternatives
Client-Server	Peer-to-Peer
Shared-nothing (cloud computing)	Symmetric multiprocessor, shared global address space
Model-View-Controller	Page controller, Front controller, Template view
Active Record	Data Mapper
RESTful URIs (all state affecting request is explicit)	Same URI does different things depending on internal state

As you work on other SaaS apps beyond this course, you should find yourself considering different architectural choices and questioning the choices being made.

Summary: Architecture & Rails

- Model-view-controller is a well known *architectural pattern* for structuring apps
- Rails codifies SaaS app structure as MVC
- *Views* are Haml w/embedded Ruby code, transformed to HTML when sent to browser
- *Models* are stored in tables of a relational database, accessed using ActiveRecord
- *Controllers* tie views and models together via *routes* and code in controller methods

Other factors being equal, which statement is **NOT** true regarding SaaS scalability?

- ☐ Shared-nothing clusters scale better than systems built from mainframes
- ☐ Relational databases scale better than "NoSQL" databases
- ☐ The programming language used (Ruby, Java, etc.) isn't a main factor in scalability
- ☐ Scalability can be impeded by *any* part of the app that becomes a bottleneck