Introduction to Ember.js

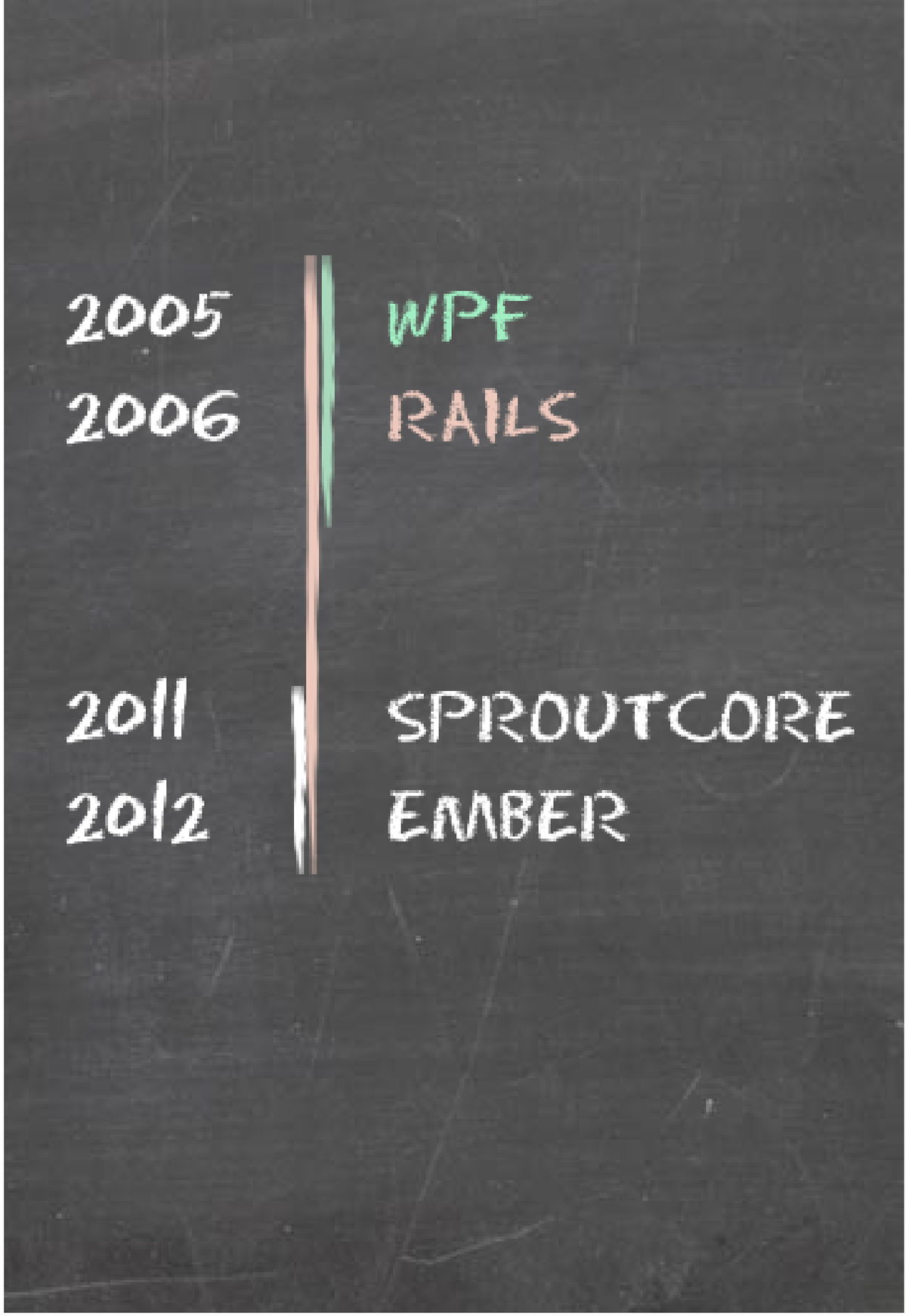
Luke Melia

Ember.js NYC Meetup

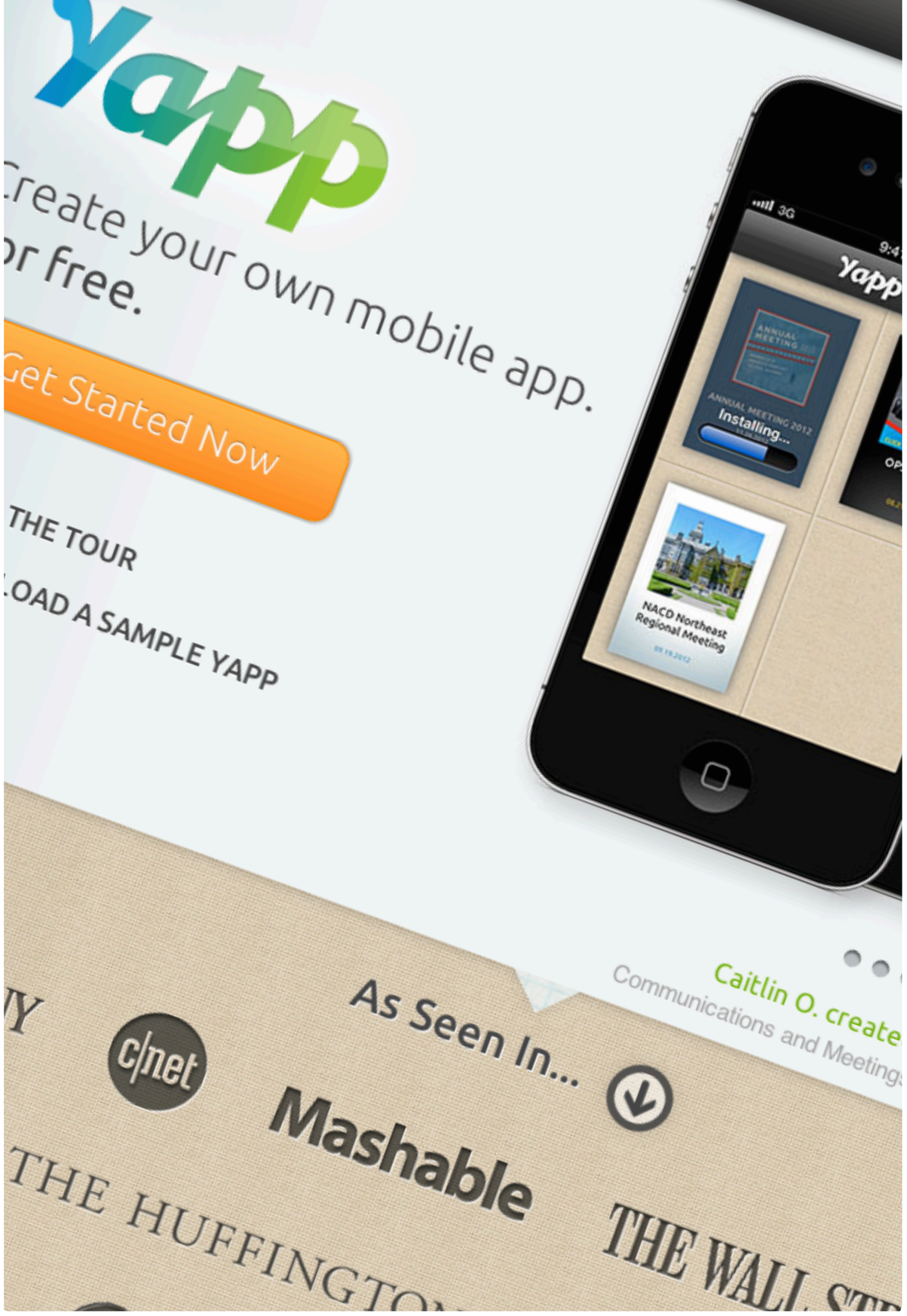
March 28th, 2013

About this Embereño

Relevant Experience



My Experience Building Ember Apps



Mobile

Cross-platform mobile, via PhoneGap

Editor

Rich desktop-style app for desktop/tablet

Dashboard

Simple one-pager for desktop/tablet

Account Settings

Form field-heavy one-pager

Goals of the Talk

- Develop an understanding of...
  - Application structure
  - Division of responsibilities
  - Some of the fundamental building blocks

The Idea Behind the Talk

- 98% of building good apps is understanding your layers and knowing what code should go where

Lay of the Land

Lay of the Land: MVC

- MVC is not created equal
- C is for Controller, but “controller” has many different meanings.
- +R for Router

Lay of the Land:
This is UI Development

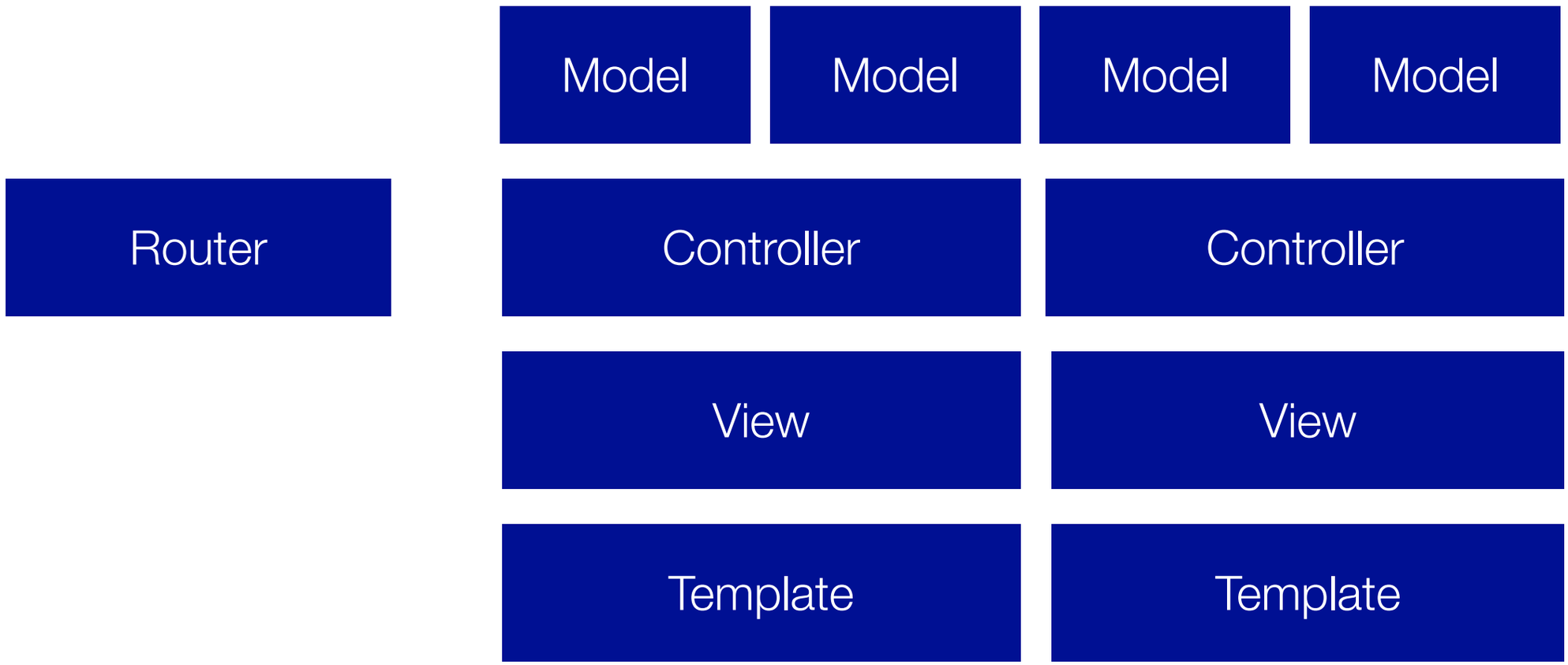
- If you've never considered yourself a professional UI Engineer, you're about to become one.
- Long-lived state
- UI responsiveness

Very different from a stateless request/ response cycle that has been popular in web development recently

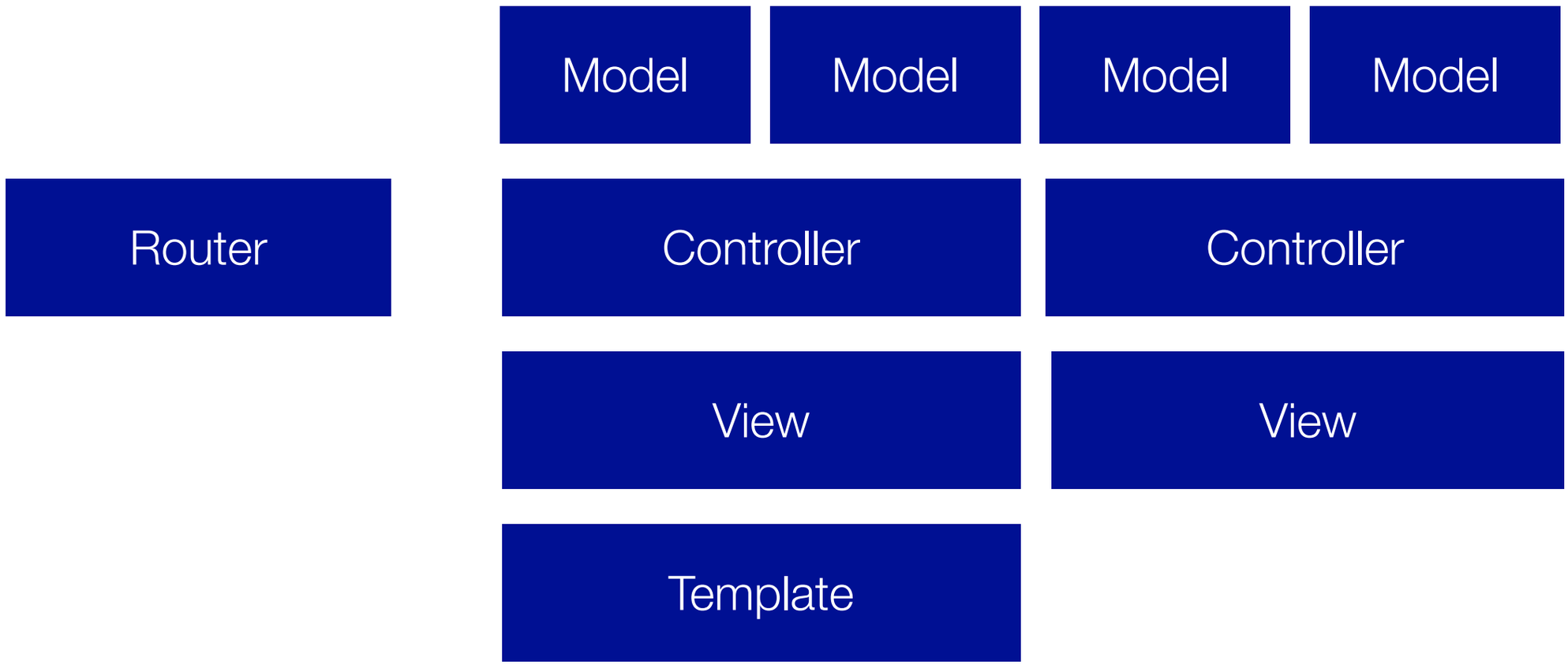
Lay of the Land: 1.0 Status

- Ember API has stabilized
- Will be at RC2 shortly
- Ember 1.0 final is coming Real Soon Now™

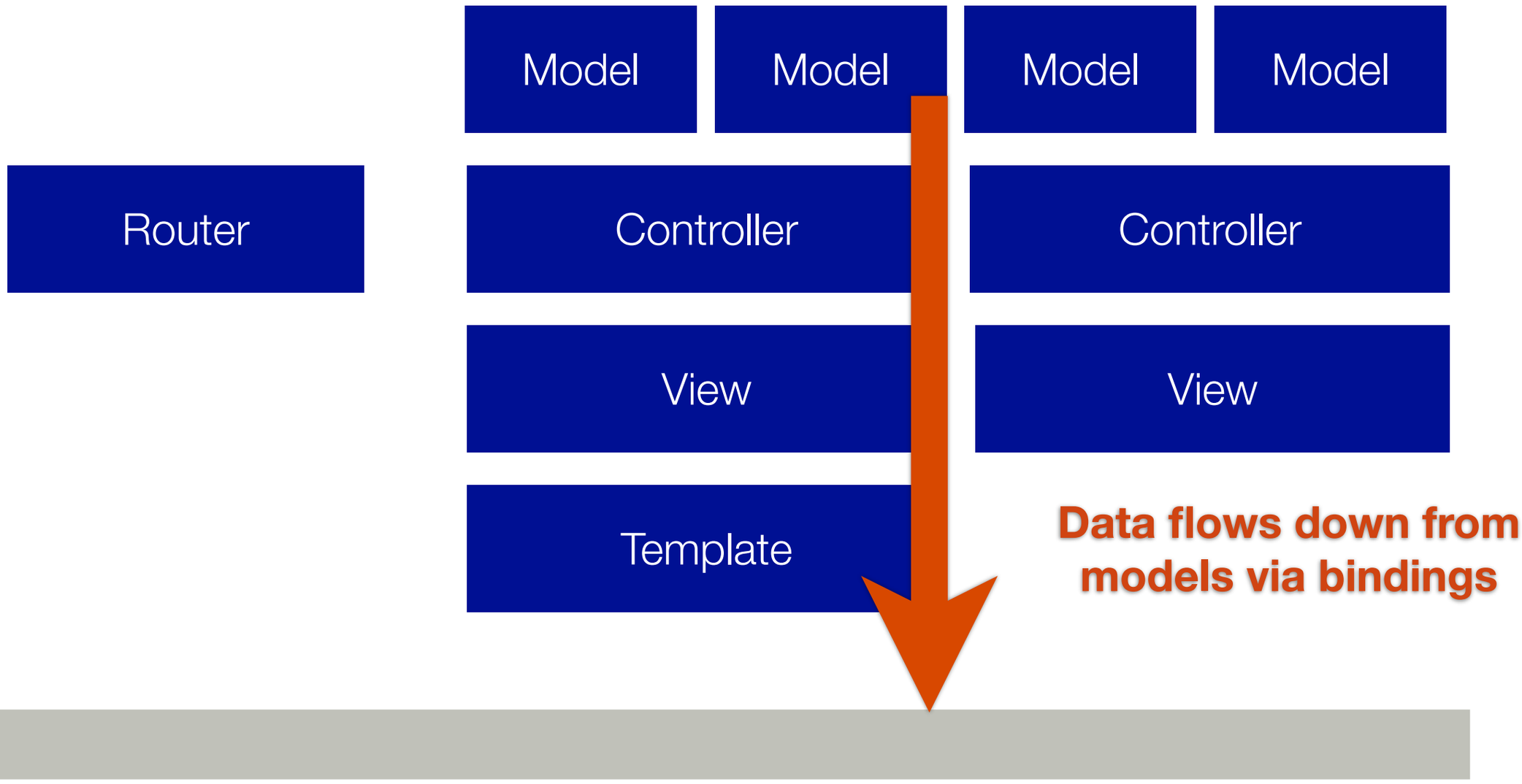
How the layers relate



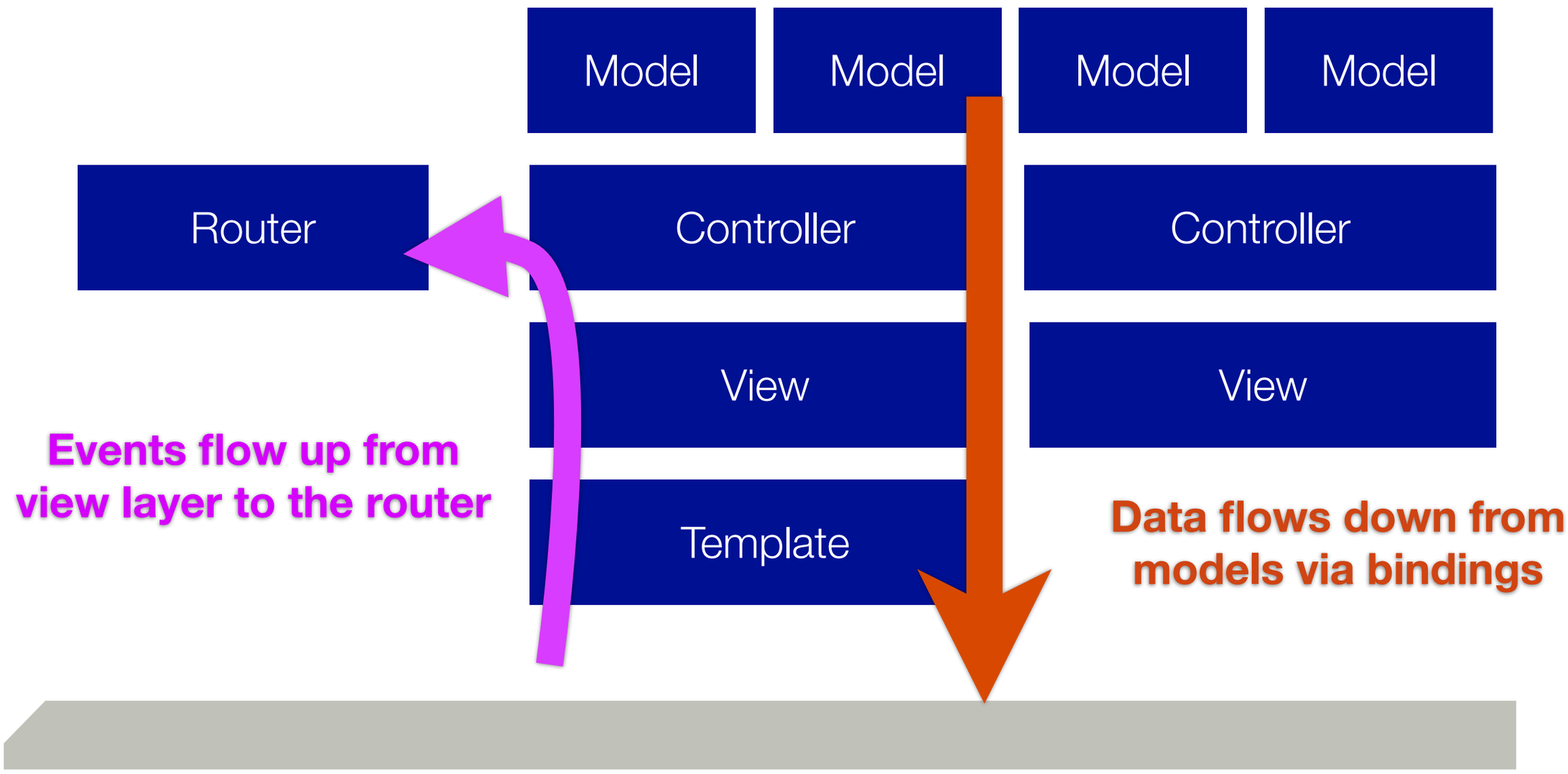
Overall app data flow



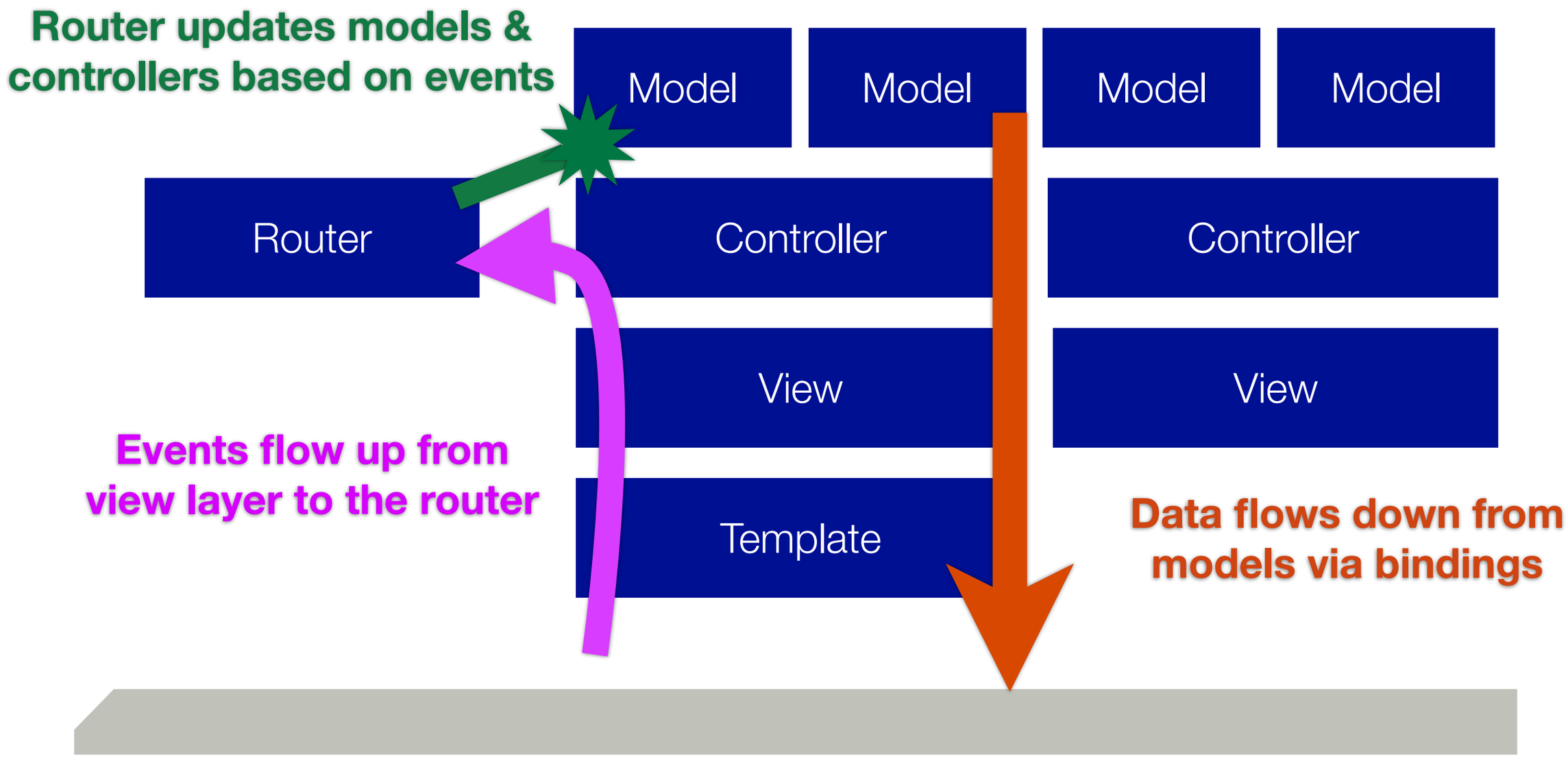
Overall app data flow



Overall app data flow

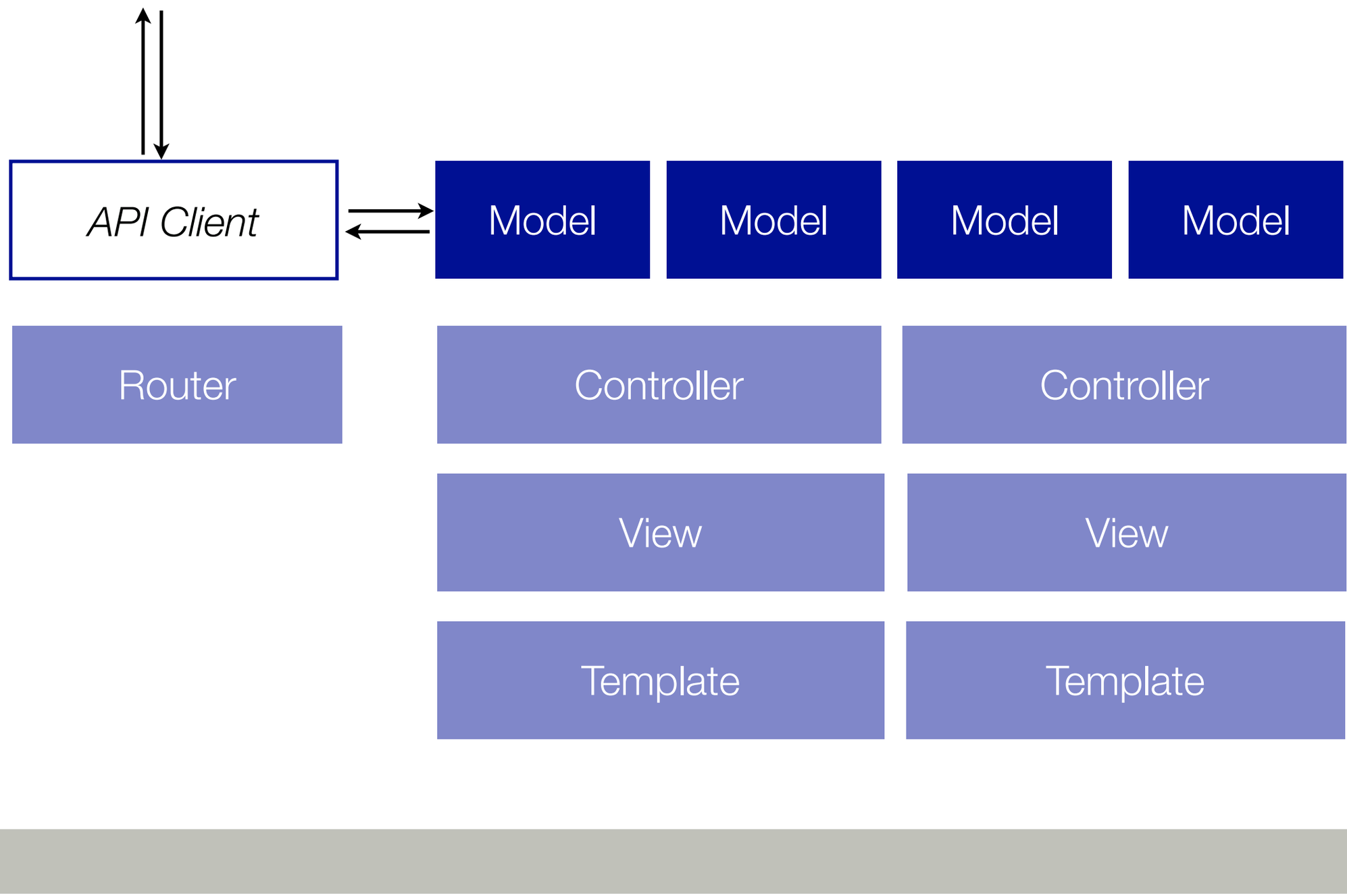


Overall app data flow

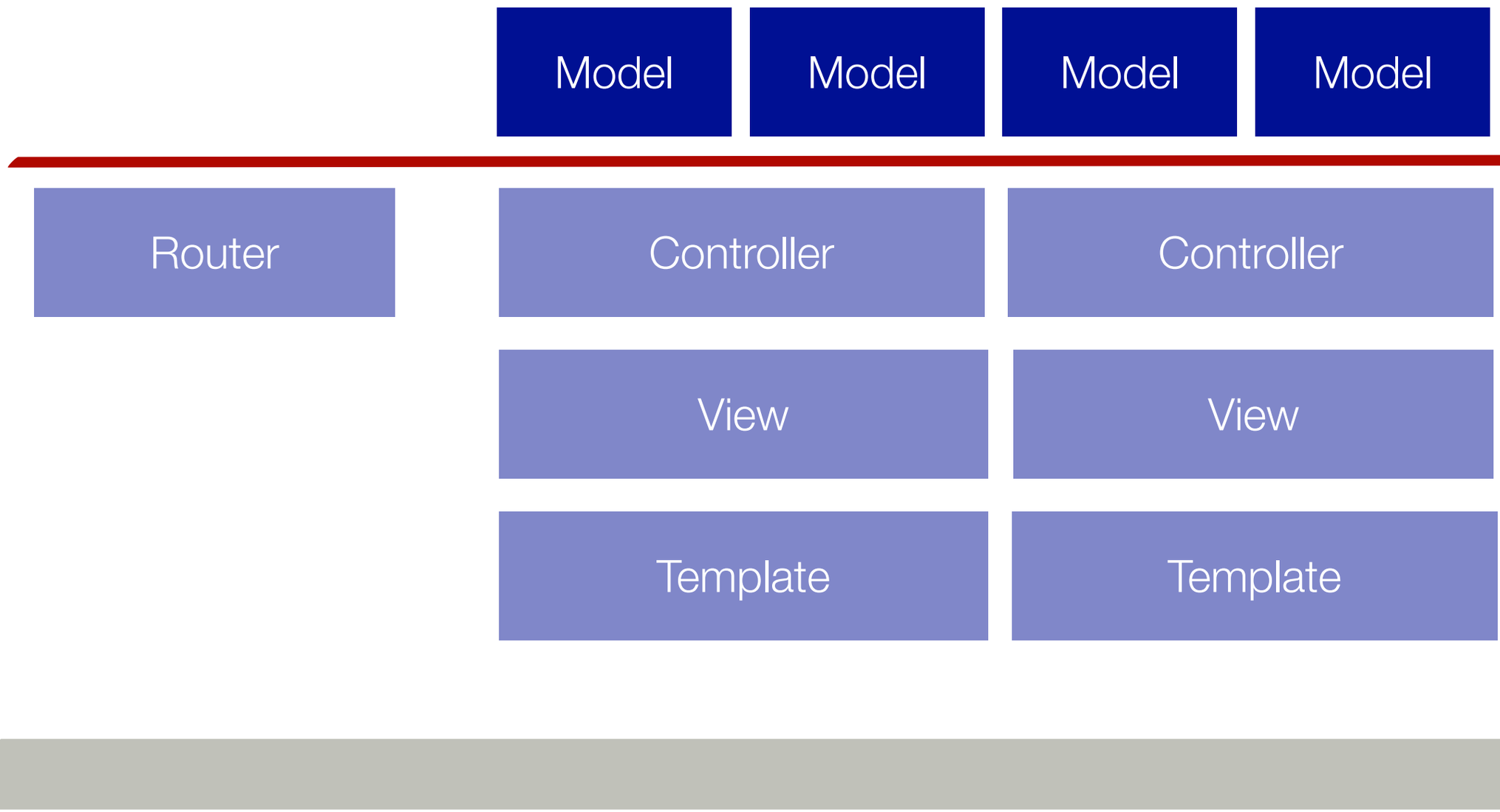


Models

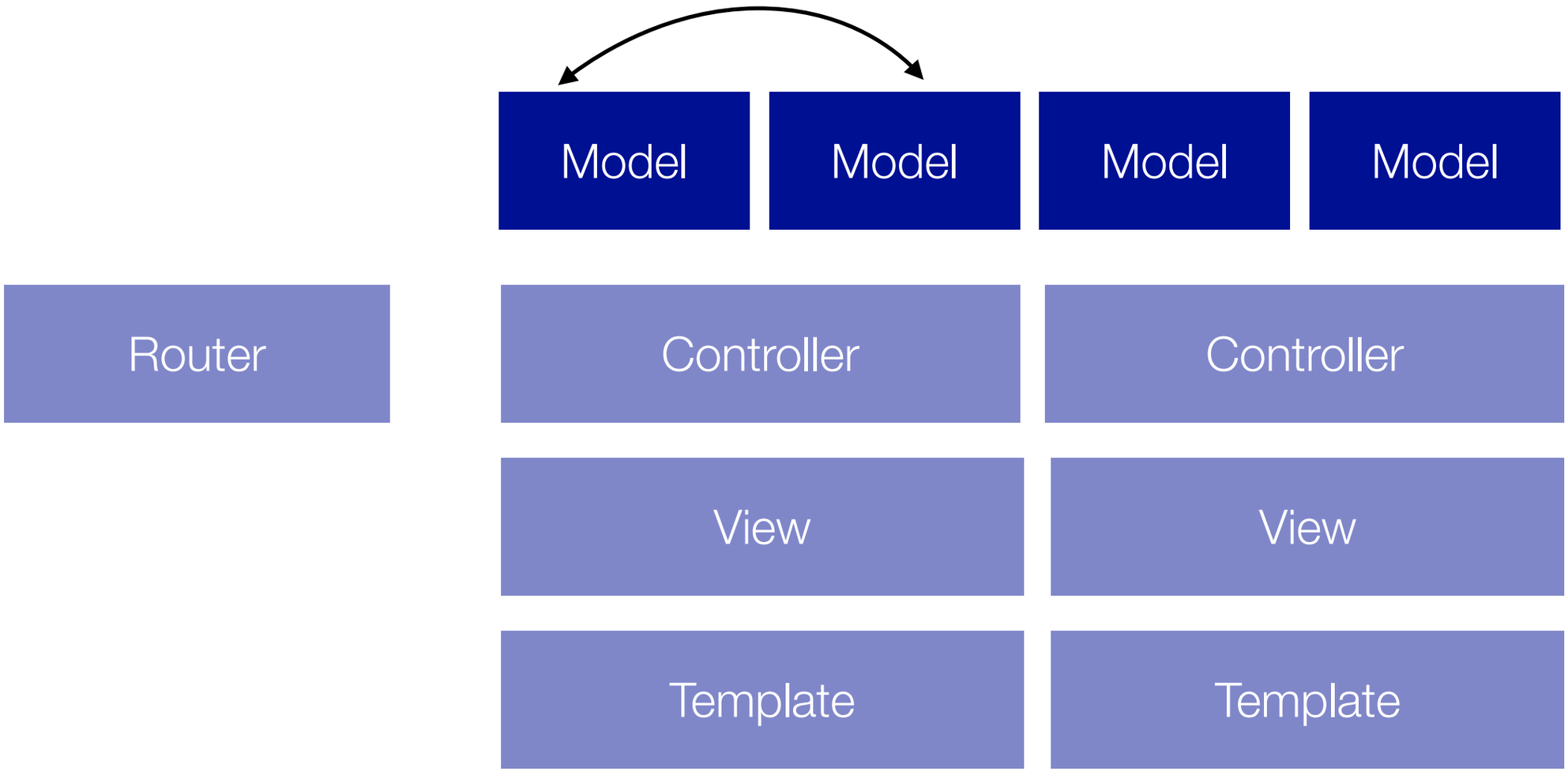
Models:
Usually serialized/deserialized to/from API



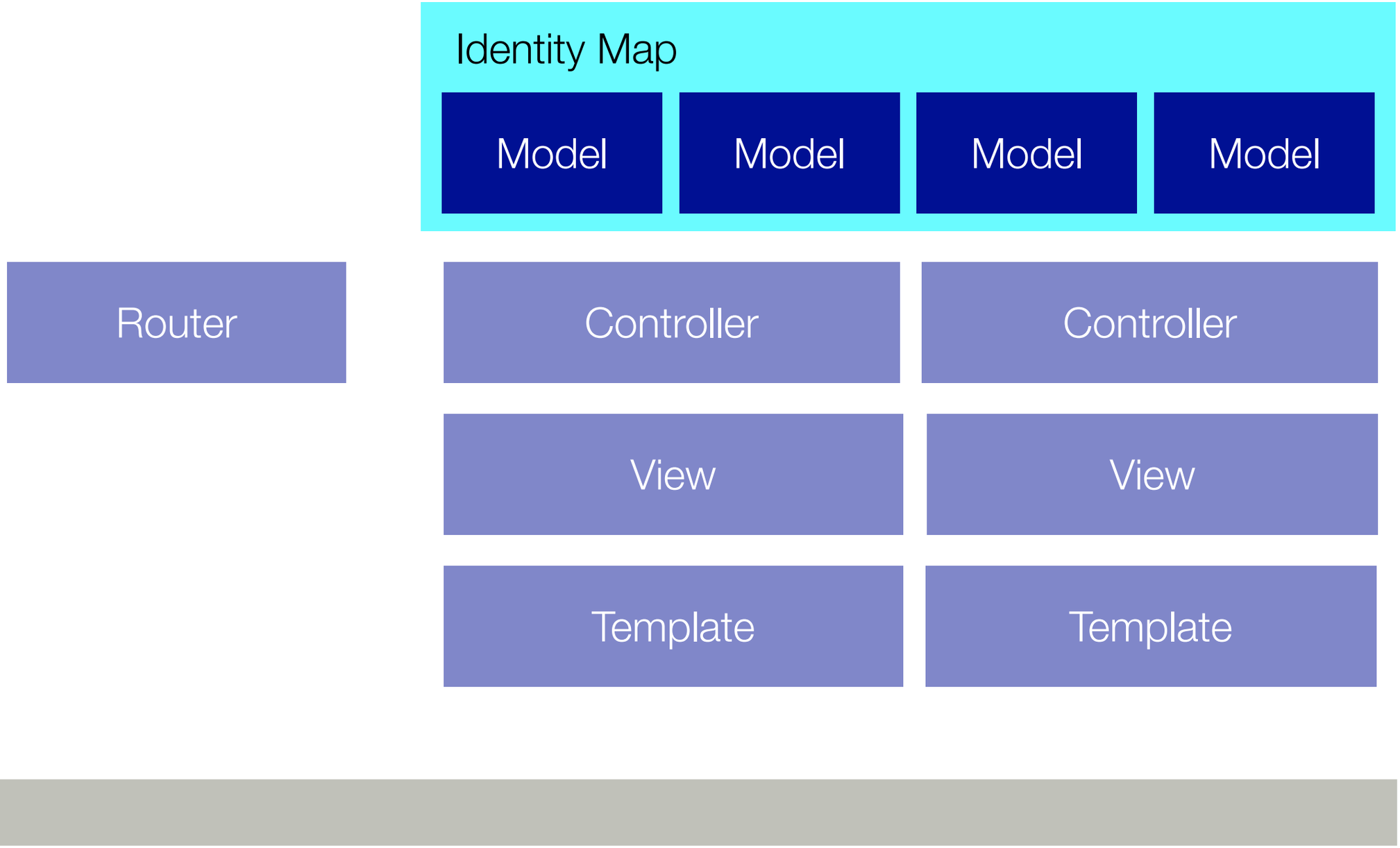
Models:
Should not depend on controllers, views or other app state



Models:
Often depend on other models



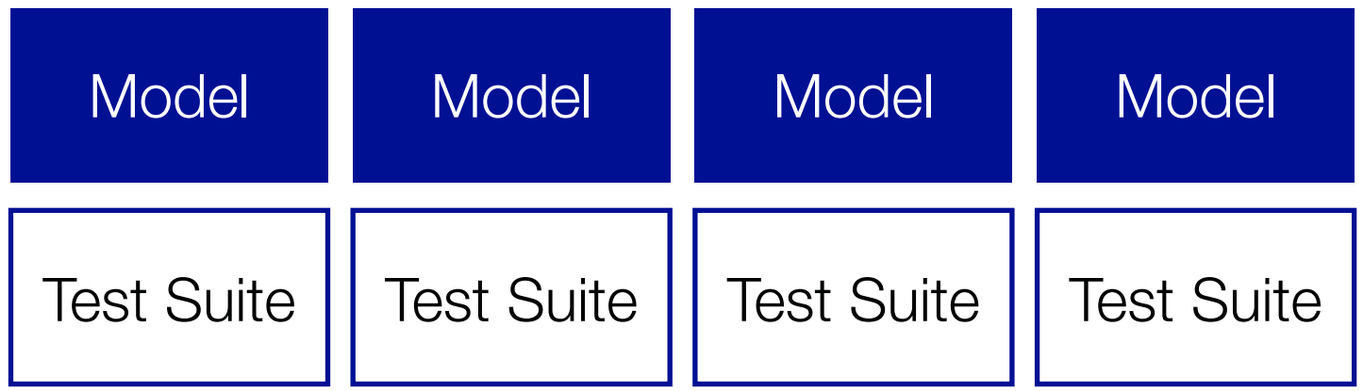
Models:
Should use an identity map



Models:
Working with router

- To work with the router, a model class should:
  - implement **find(id)**
  - implement **then(success, failure)** (promise pattern)

Models:
Testable in isolation



Models: ember-data

- ember-data is an ORM for Ember
- Store + Adapter + Serializer
- DS.Store implements an Identity Map
- RESTAdapter; BasicAdapter
- Work in progress; API is much less stable than Ember core

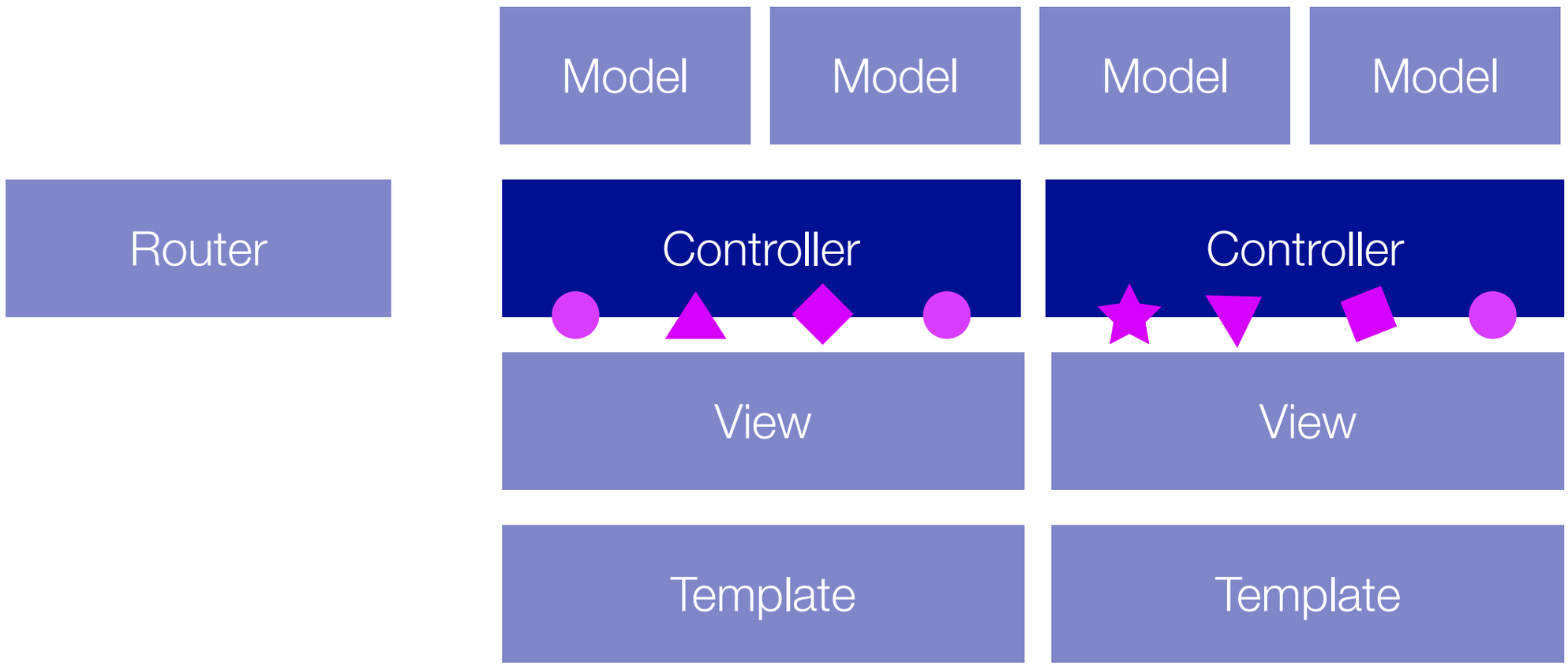
Models:
$.ajax plus Ember.Object

- Subclass Ember.Object for your model classes
- Retrieve and persist data using jQuery

For a great example of this in an open source project, check out Discourse

Controllers

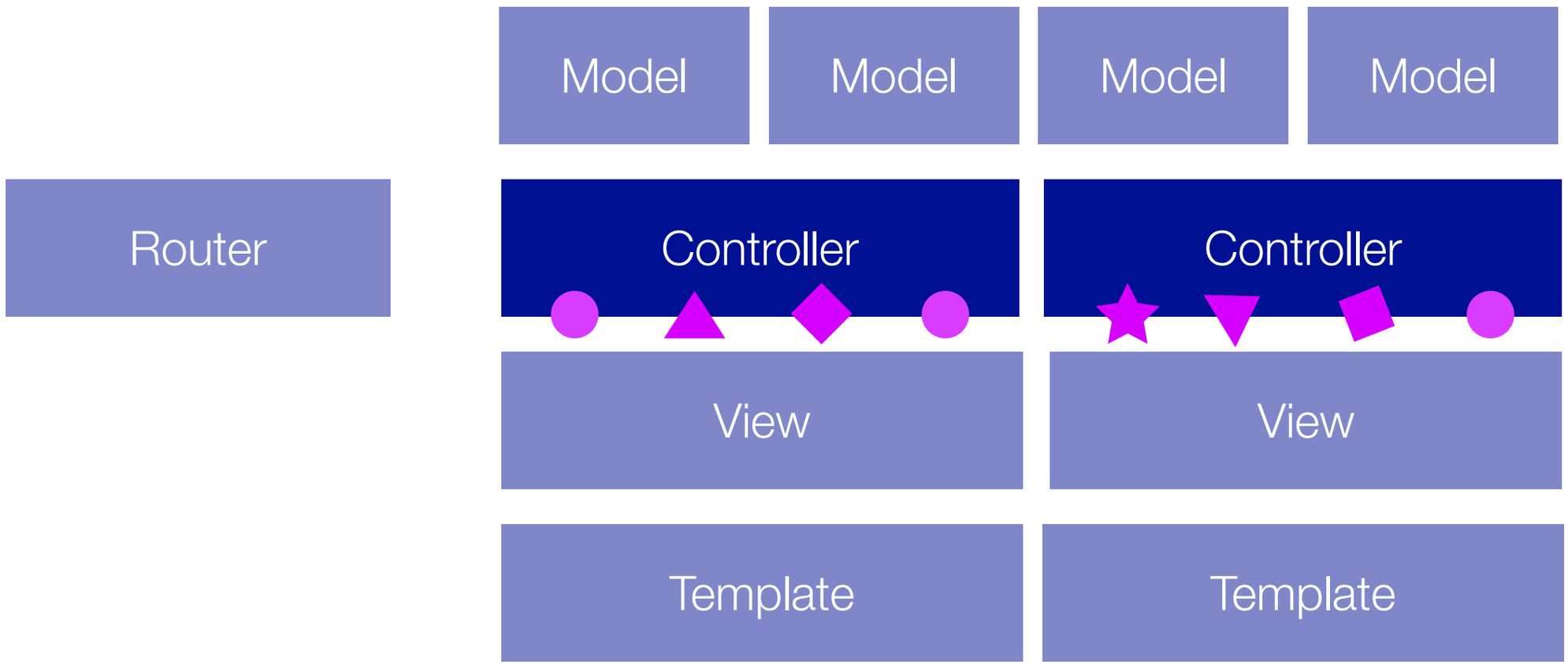
Controllers:
Present data for the view layer to render



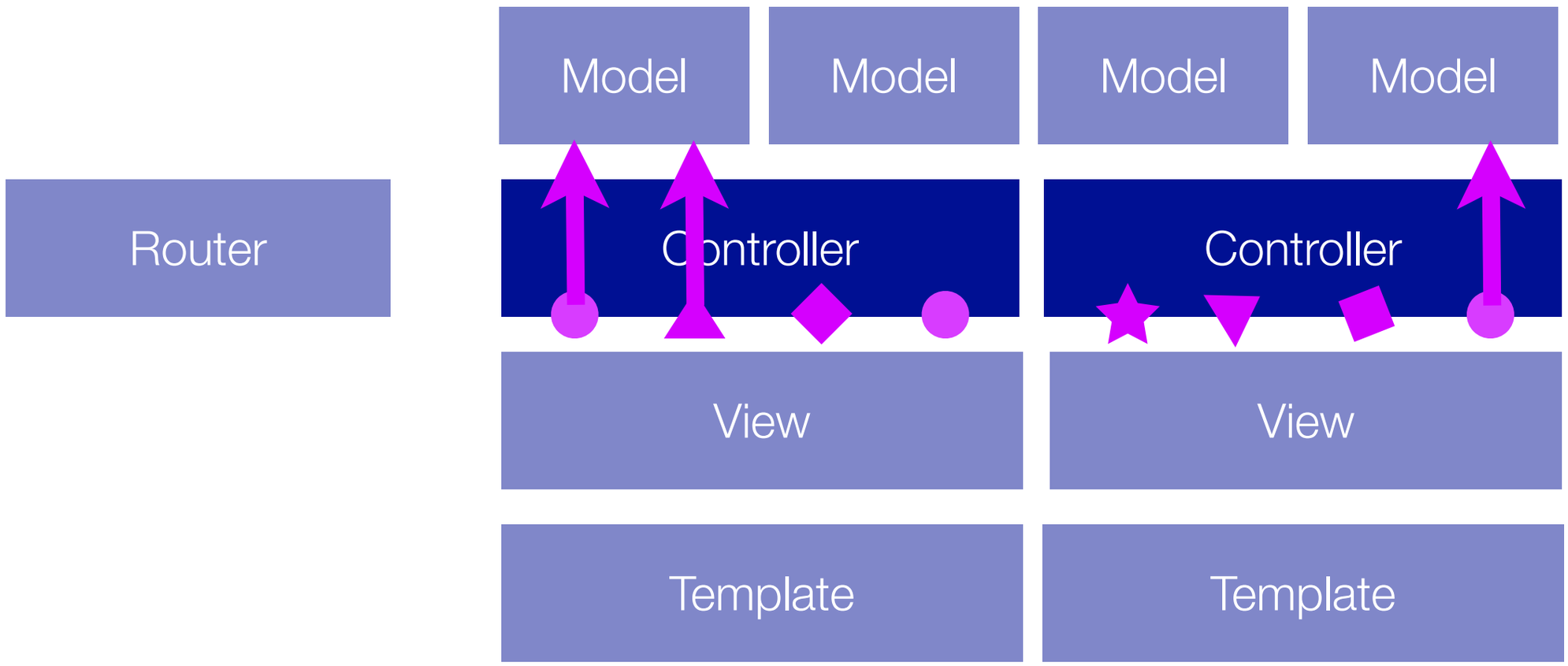
Controllers:
Expose bindable properties


```js
App.TodosController = Ember.Object.extend({
  todos: [],
  remaining: function() {
    var todos = this.get('todos');
    return todos.filterProperty('isDone', false).get('length');
  }.property('todos.@each.isDone')
});
```

Controllers:
Often proxy models



Controllers:
Often proxy models



Controllers: Ember.ObjectController

- Transparently proxies missing properties and methods to the object set as its content property
- Destroyer of boilerplate

Controllers: Ember.ObjectController


```js
App.TodoController = Em.ObjectController.extend({
  content: null
});
var controller = App.TodoController.create(),
    todo = Em.Object.create({title: "Floss"});
controller.set('content', todo);
controller.get('title'); //=> "Floss"
```

Controllers: Ember.ArrayController


```
App.TodosController = Em.ArrayController.extend({
  content: null
});
var controller = App.TodosController.create(),
    todos = [App.Todo.create(), App.Todo.create()];
controller.set('content', todos);
controller.get('length'); //=> 2
```

JS

Controllers:
Lazily instantiated once by the container

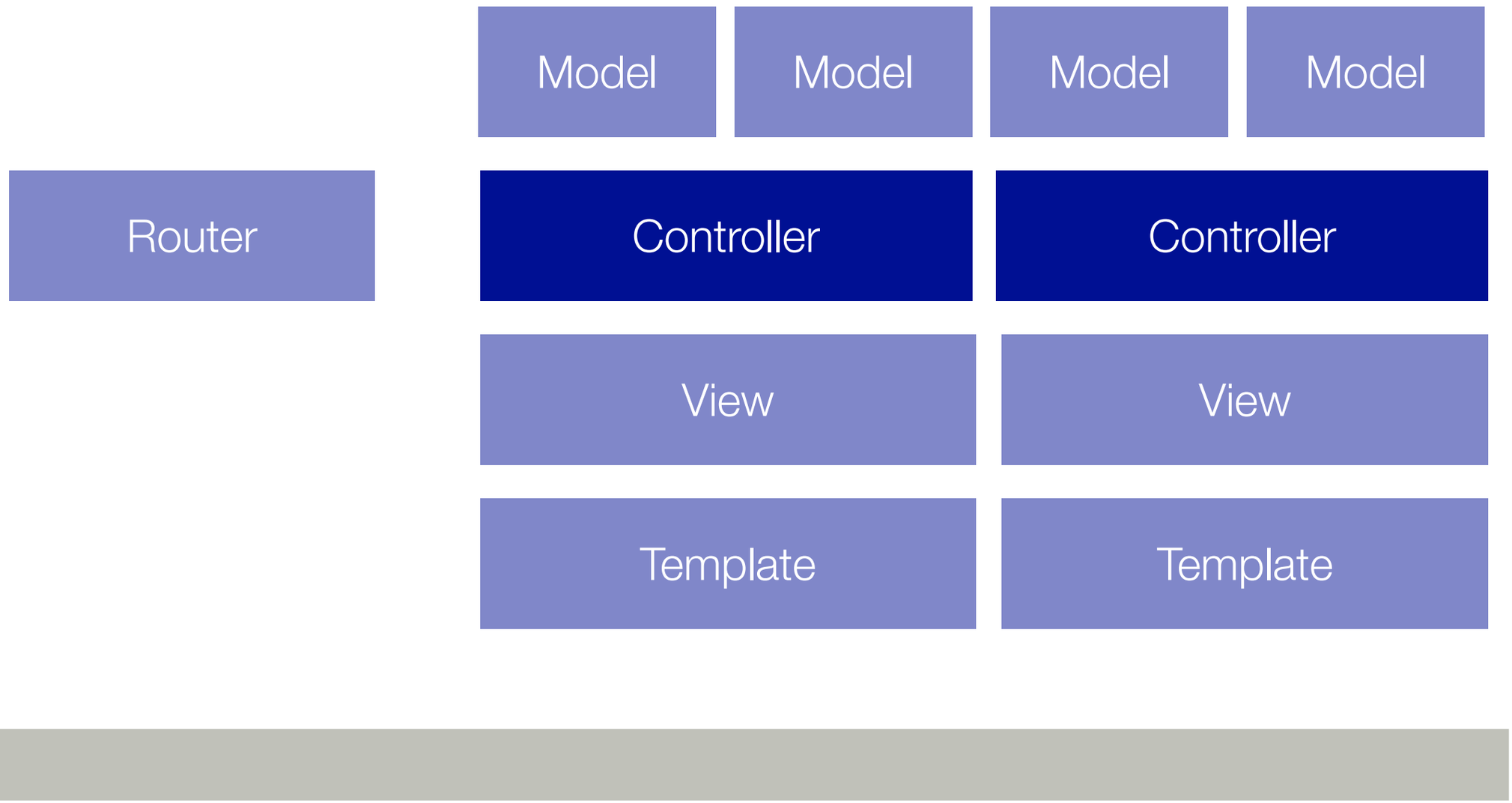

```
1  Ember.Route = Ember.Object.extend({
2    controllerFor: function(name, model) {
3      var container = this.router.container,
4          controller = container.lookup('controller:' + name);
5
6      if (!controller) {
7        model = model || this.modelFor(name);
8        controller = Em.generateController(container, name, model);
9      }
10
11     return controller;
12   }
13 });
14
```

container.lookup is where the lazy instantiation will occur

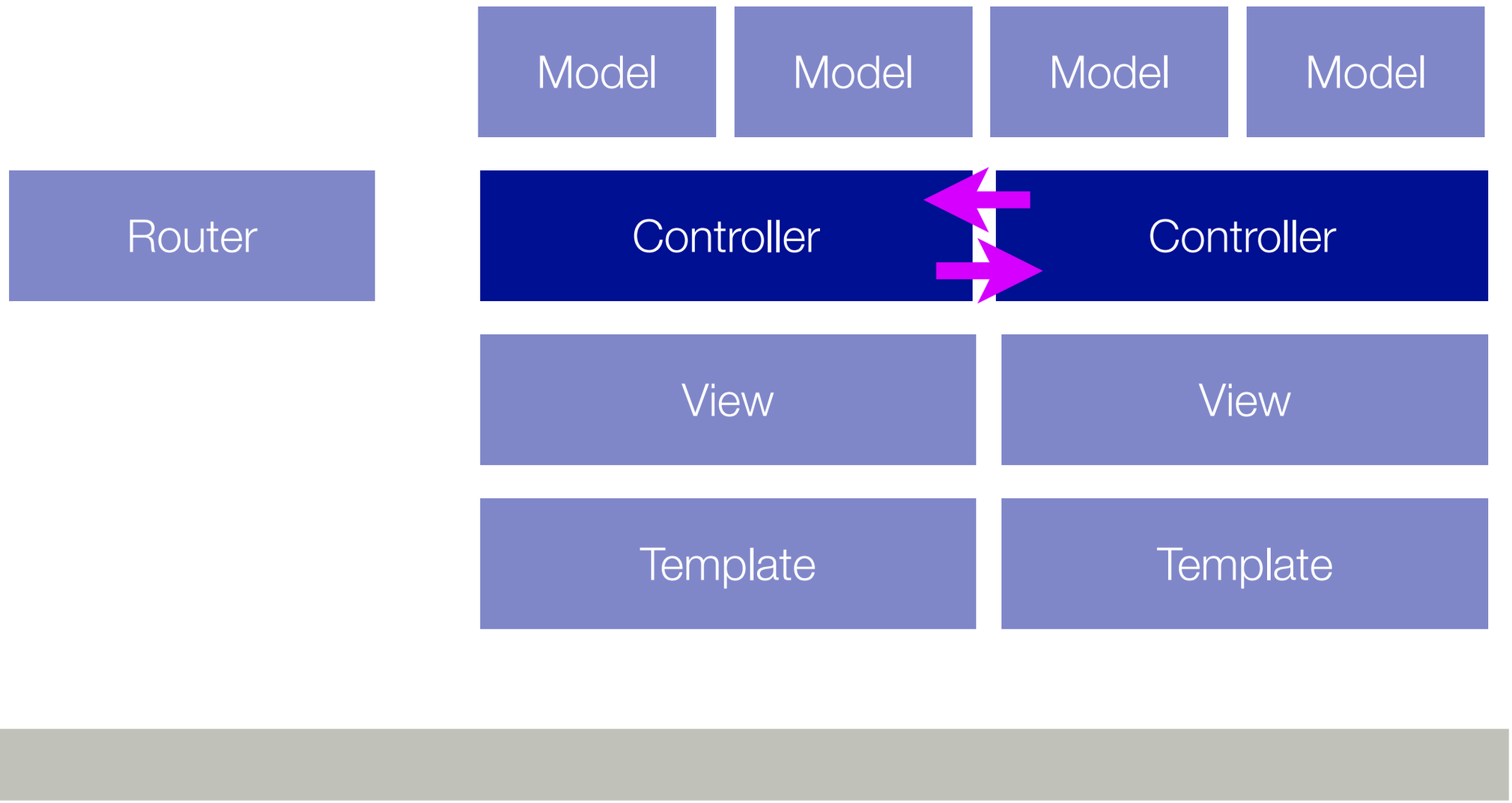
Also, notice how a controller is generated for you if none is found

JS

Controllers:
May collaborate with other controllers



Controllers:
May collaborate with other controllers



Controllers: “needs”


```
1  App.ProfileController = Ember.Controller.extend({
2    needs: ['user'],
3    username: Em.computed.alias('controllers.user.username')
4  });
```

JS

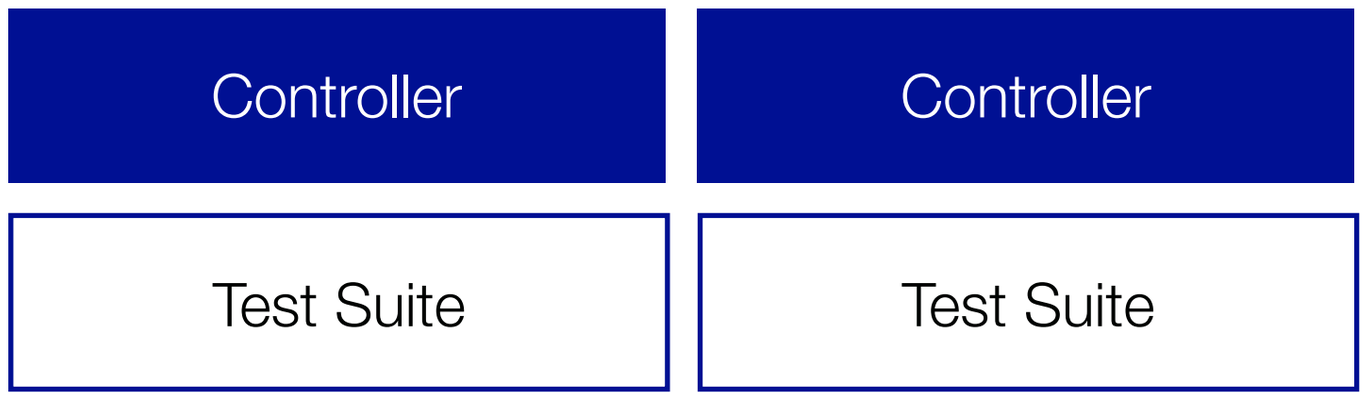
Explanation of the shortcuts used here:

Em is short for Ember

Ember.computed.alias usage here expands to:

```
function(){
  return this.get('controllers.user.username');
}.property('controllers.user.username')
```

Controllers:
Testable in isolation



The View Layer

View Layer:
Contain all DOM interaction

Model | Model | Model | Model

Router | Controller | Controller

View | View

Template | Template

View Classes: Responsibilities

- Optional (will be generated)
- Translate primitive events (DOM events) into semantic events that affect app state

Views:
Let Handlebars do the heavy lifting


```js
App.PeopleListView = Ember.View.extend({
  templateName: 'people_list'
});
```

```html
<h1>People</h1>
<ul>
{{#each people}}
  <li>Hello, <b>{{fullName}}</b>!</li>
{{/each}}
</ul>
```

Templates:
Bound Expressions


```hbs
{{firstName}} {{lastName}}
<img {{bindAttr src="imageUrl"}}
```

HBS

Templates: Conditionals


```hbs
{{#if person}}
  Welcome back, <b>{{person.firstName}} {{person.lastName}}</b>!
{{else}}
  Please log in.
{{/if}}
```

Templates:
Outlets are placeholders


```hbs
{{outlet}} <!-- default outlet -->
{{outlet sidebar}} <!-- named outlet -->
```

Templates: Nesting I


```hbs
{{view App.PersonView}}
{{template 'person'}}
{{partial 'person'}}
```

HBS

Templates:
Changing Scope


```hbs
Welcome back, <b>{{person.firstName}} {{person.lastName}}</b>!
```

```hbs
{{#with person}}
  Welcome back, <b>{{firstName}} {{lastName}}</b>!
{{/with}}
```

Templates:
Nesting and Changing Scope


```hbs
<script type="text/x-handlebars" data-template-name='author'>
  Written by {{firstName}} {{lastName}}.
  double click to toggle l Posts: {{postCount}}
</script>
```

HBS

```hbs
<script type="text/x-handlebars" data-template-name='post'>
  <h1>{{title}}</h1>
  <div>{{body}}</div>
  {{render "author" author}}
</script>
```

HBS

```js
App.AuthorController = Ember.ObjectController.extend({
  postCount: function() {
    return App.Post.countForAuthor(this.get("model"));
  }.property("model","App.Post.@each.author")
})
```

JS

Templates: Firing events


```hbs
<p>
  <button {{action "selectPost" post}}>✓</button>
  {{post.title}}
</p>
```

HBS

That will try to call a selectPost method on the controller, or on the current route

View Layer: Understanding keywords

- controller
- view
- context

Templates: Understanding keywords


```
1  firstName from controller: {{controller.firstName}}
2  firstName from view: {{view.firstName}}
3  firstName from context: {{firstName}}
```

HTML

Views:
One controller per view

- Should bind to properties of one controller
- If you need data from elsewhere:
  - bring it into this view's controller using `needs`
  - or, {{render}} a new template

Views:
Ways to handle DOM events

- Ember-dispatched events
- Handlebars action helper
- jQuery

Views:
Ember-dispatched events


```js
App.ThumbnailView = Ember.View.extend({
  templateName: 'thumbnail',
  mouseDown: function(event){
    window.alert('I pressed a thumbnail!');
  }
});
```

Views:
Ember-dispatched events

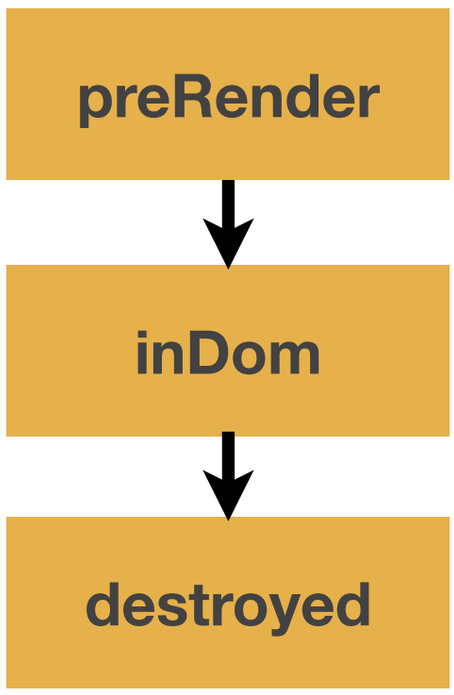
touchStart, touchMove, touchEnd, touchCancel, keyDown, keyUp, keyPress, mouseDown, mouseUp, contextMenu, click, doubleClick, mouseMove, focusIn, focusOut, mouseEnter, mouseLeave, submit, input, change, dragStart, drag, dragEnter, dragLeave, dragOver, drop, dragEnd

Views:
Ember-dispatched events

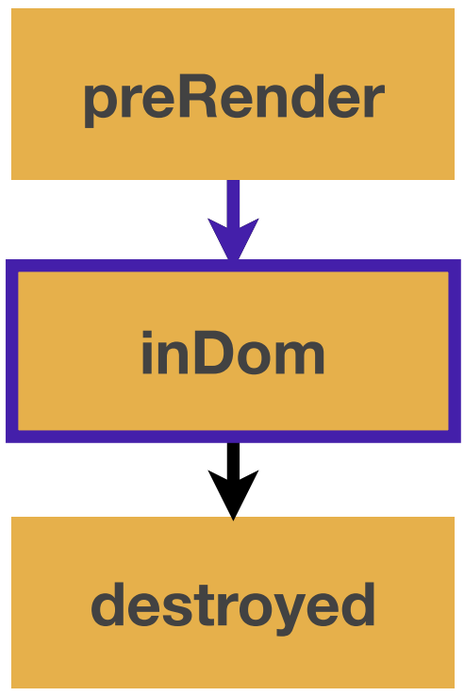

```
1  window.App = Ember.Application.create({
2    customEvents: {
3      mousewheel: 'mouseWheel'
4    }
5  });
```

JS

Views:
Using jQuery to handle events

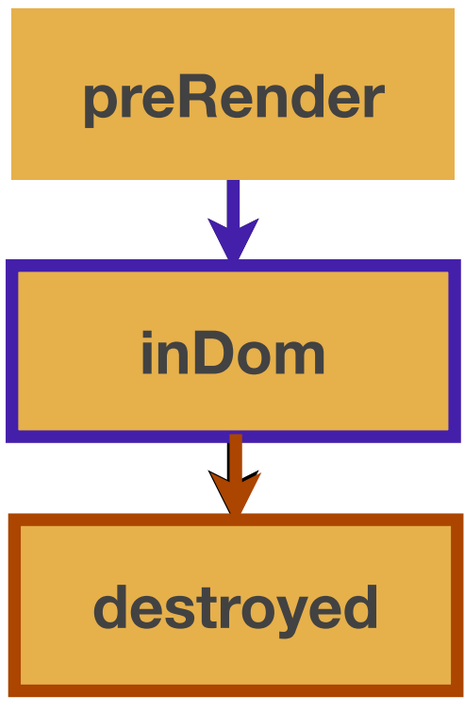


Views:
Using jQuery to handle events



didInsertElement *(after)*

Views:
Using jQuery to handle events



didInsertElement *(after)*

willDestroyElement *(before)*

Views:
Using jQuery to handle events


```
App.ThumbnailView = Em.View.extend({
  didInsertElement: function(){
    this.$().lightbox();
  },
  willDestroyElement: function(){
    this.$().unbindLightbox();
  }
});
```

JS

View tips I

- Remember that views are created and destroyed over lifetime of app and at render time

```
1  {{#if}}
2    {{view Foo}}
3  {{else}}
4    {{view Bar}}
5  {{/if}}
```

HTML

Be sure you are not storing state you want to keep around on view instances!

View tips II

- It’s a good practice for views to know as little as possible about their parent view hierarchy

Views: ContainerView

- When {{#each}} isn't enough...
- To manually manage views, use Ember.ContainerView
- http://emberjs.com/guides/views/manually-managing-view-hierarchy/

Data-binding caveat

- Bound properties, using computed properties or bindings, are very powerful
- Two-way bindings can lead to unexpected behavior and bugs and are often unnecessary

Router

Router: Responsibilities

- Manages application state
- Keeps the URL up to date as you transition between routes

Router: DSL


```js
App.Router.map(function() {

  // users template will be displayed when navigating to /users
  this.resource("users");

  // posts.index template will be displayed when navigating to /posts
  this.resource("posts", function() {

    // posts.show template will be displayed when navigating to /posts/1
    this.route('show', {path: ':post_id'});

    // posts.edit template will be displayed when navigating to /posts/1/edit
    this.route('edit', {path: ':post_id/edit'});
  });
});
```

Router:
Route classes I


```js
App.PostsIndexRoute = Ember.Route.extend({
  events: {
    myCoolAction: function() {
      // do your business
    }
  }
});
```

JS

Actions look for a handler at the current route and then check up the route hierarchy until they find a handler.

Router:
Route classes II

- Lots of hooks to override conventional behavior when needed
- http://emberjs.com/api/classes/Ember.Route.html

Router: Tips

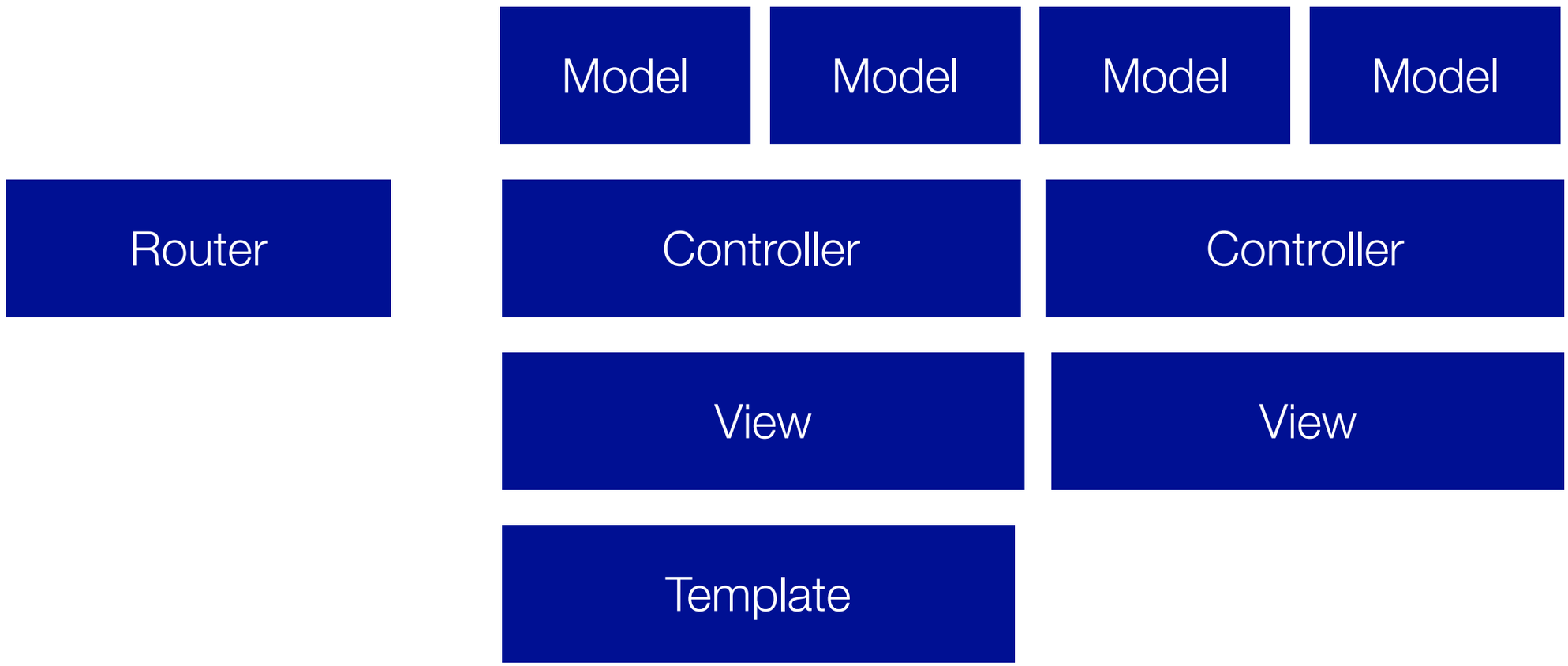
- Be careful with asynchronicity in the router event handlers.
- Delegate to separate state managers where helpful.

Router:
Naming Conventions

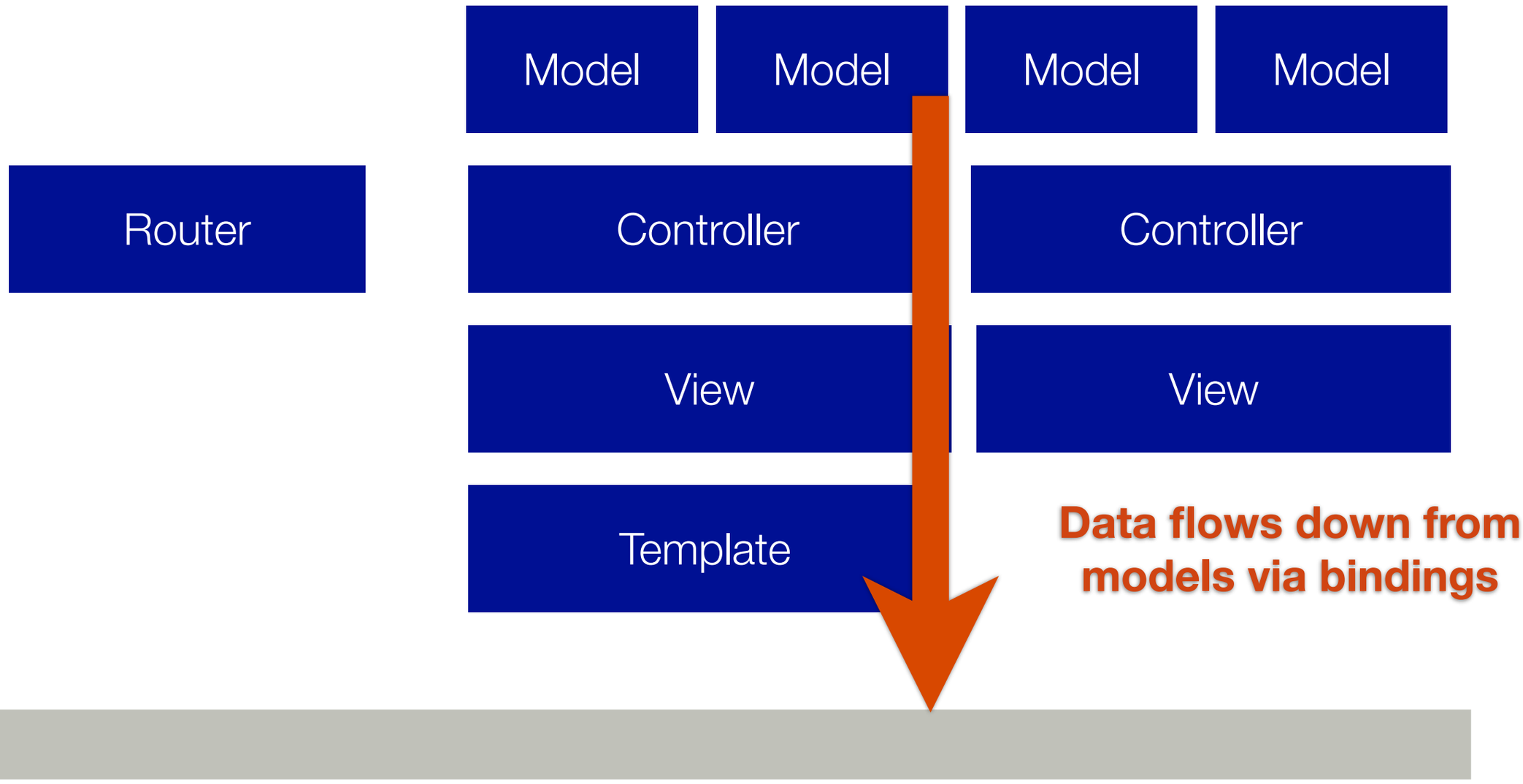

```js
App.Router.map(function() {
  this.resource('posts', function() { // the `posts` route
    this.route('favorites');          // the `posts.favorites` route
    this.resource('post');            // the `post` route
  });
});
```

Route Name	Controller	Route	Template
`posts`	`PostsController`	`PostsRoute`	`posts`
`posts.favorites`	`PostsFavoritesController`	`PostsFavoritesRoute`	`posts/favorites`
`post`	`PostController`	`PostRoute`	`post`

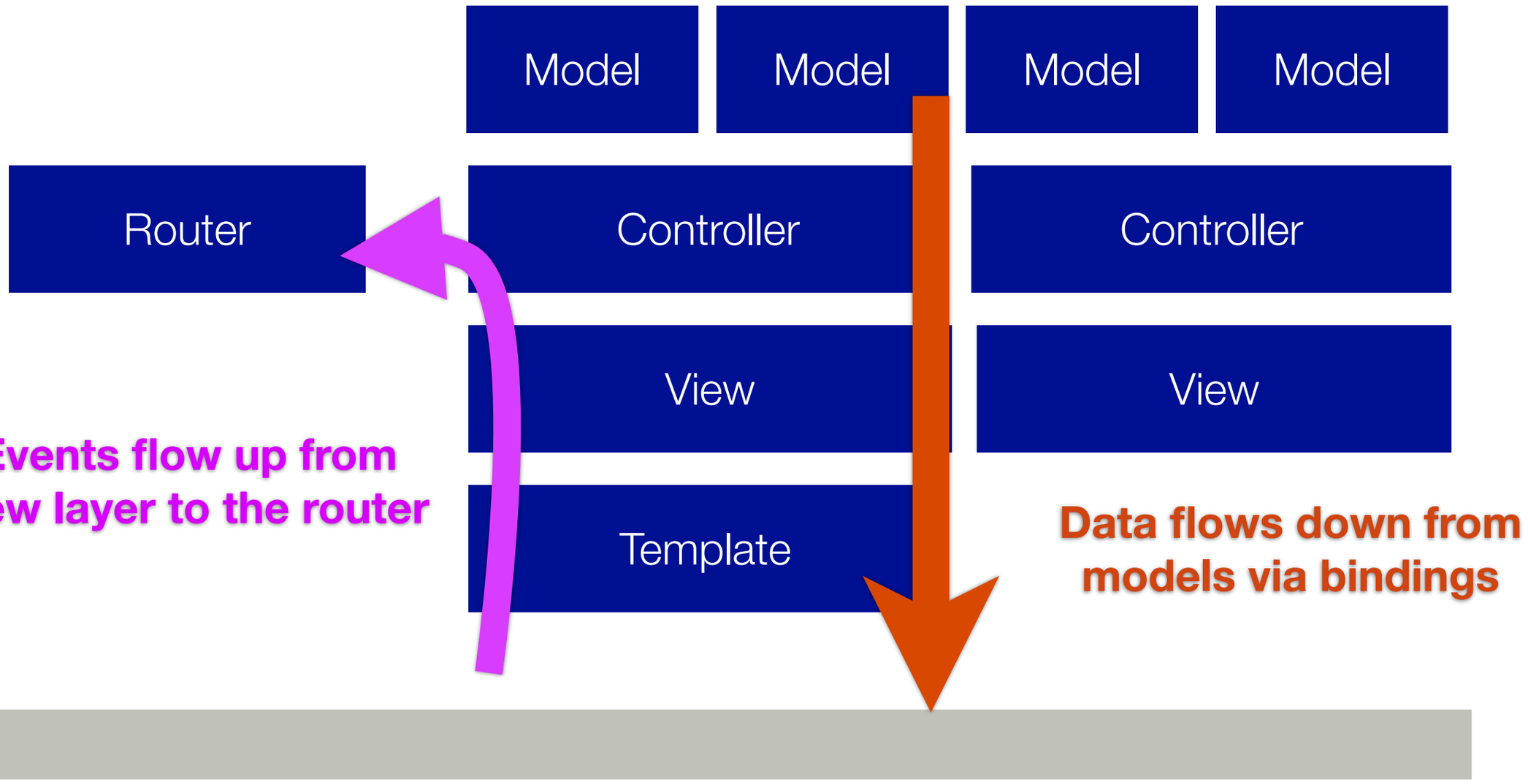
Overall app data flow, reprise



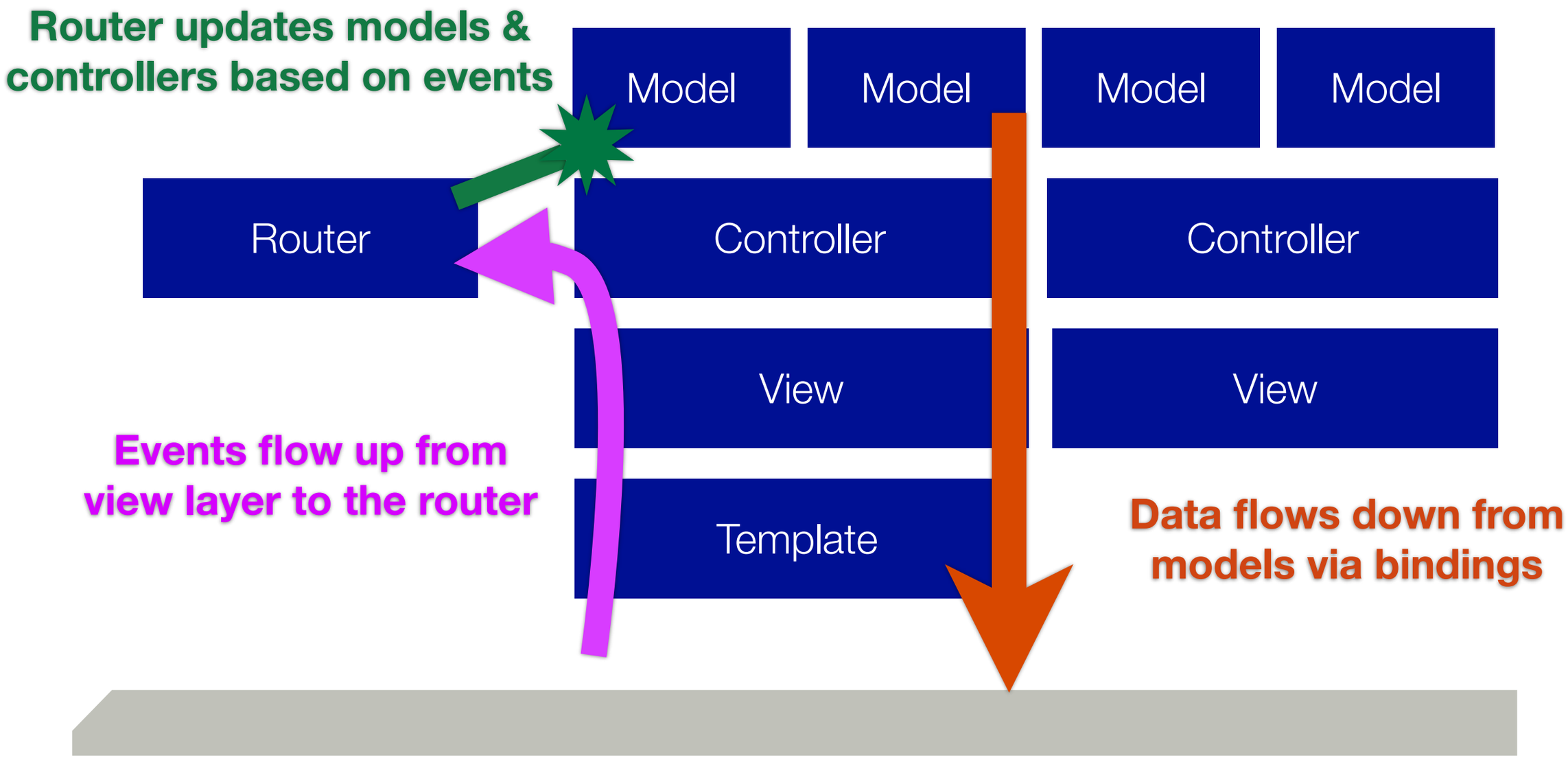
Overall app data flow, reprise



Overall app data flow, reprise



Overall app data flow, reprise



Q & A

Follow me @lukemelia

Some examples appear courtesy of my company, Yapp (which is hiring now or soon).

Creative Commons photo credits: flickr.com/photos/fictures/4596895