Ember.js In Depth

frontendmasters.com

@MichaelLNorth

github.com/mike-north

About.me

- Ember.js, Ember-cli, Ember-data contributor
- `Job.new =` CTO LEVANTO FINANCIAL
- `Job.old =` UI Architect for Yahoo Ads & Data
- Organizer, Modern Web UI
- OSS Enthusiast

About.me

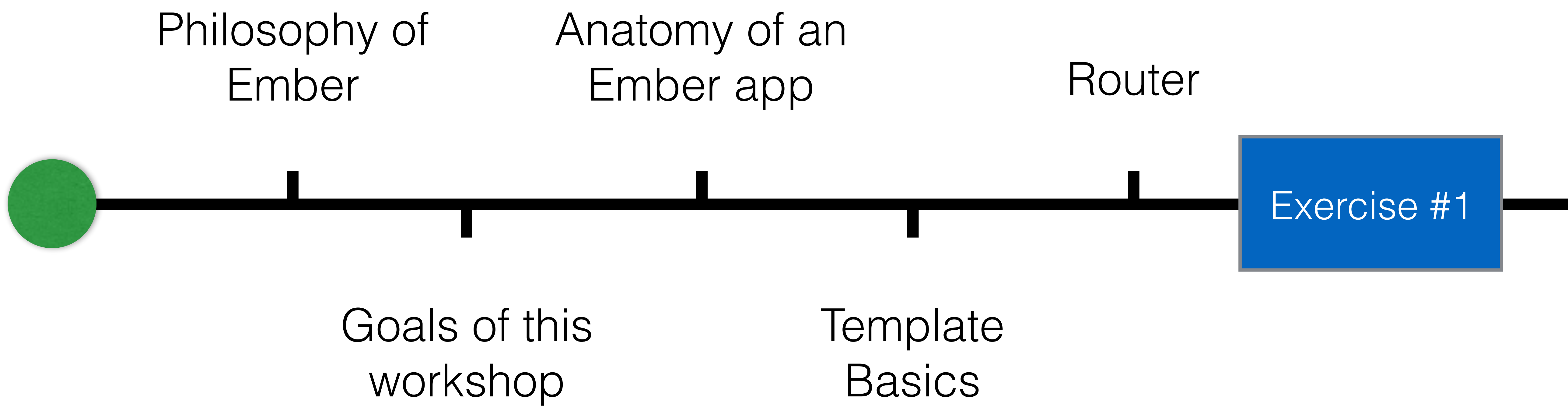
- I <3 Productivity and composability
- Product oriented
  - Building quickly and iteratively is better
  - Done is better than perfect
  - Hate reinventing the wheel

About.me

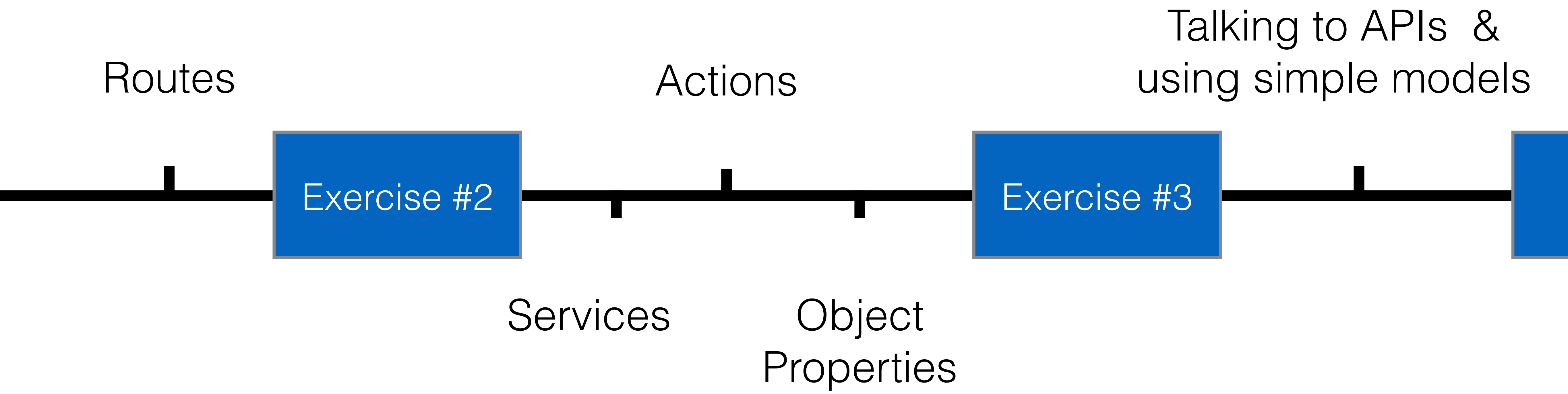
I’ve written a lot of ember addons

- ember-material-lite
- ember-cli-materialize
- ember-c3-shim
- ember-compostability
- ember-api-actions
- ember-orientation
- ember-perf
- ember-resize
- ember-literal
- ember-load
- ember-anchor
- ember-load

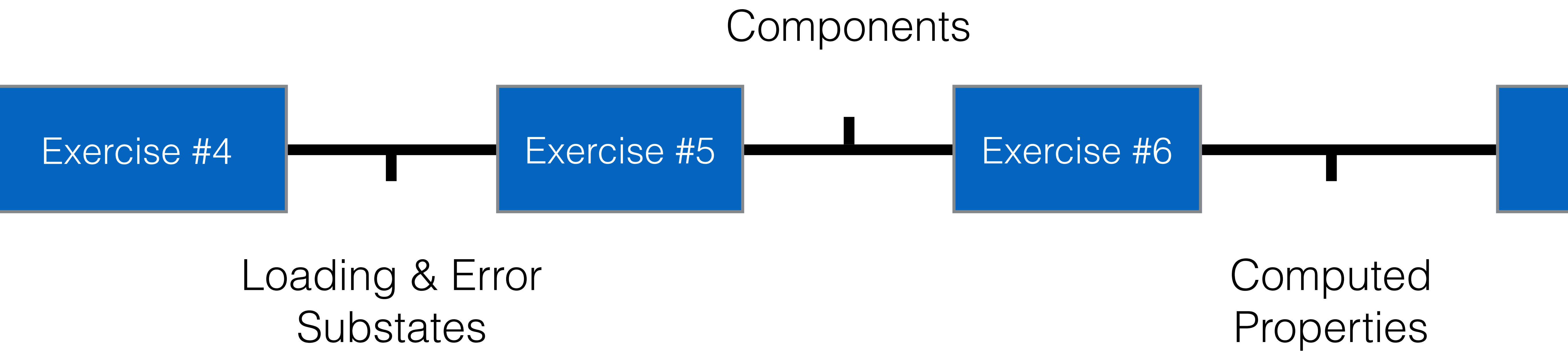
Agenda


Philosophy of Ember
Goals of this workshop
Anatomy of an Ember app
Template Basics
Router
Exercise #1

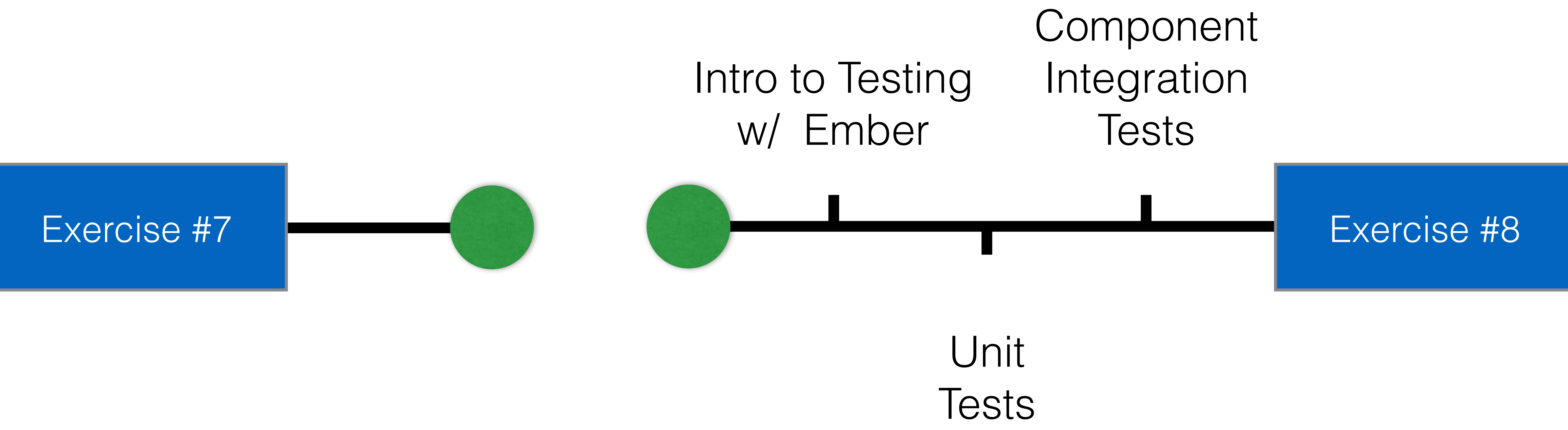
Agenda


Routes
Exercise #2
Services
Actions
Object Properties
Exercise #3
Talking to APIs & using simple models

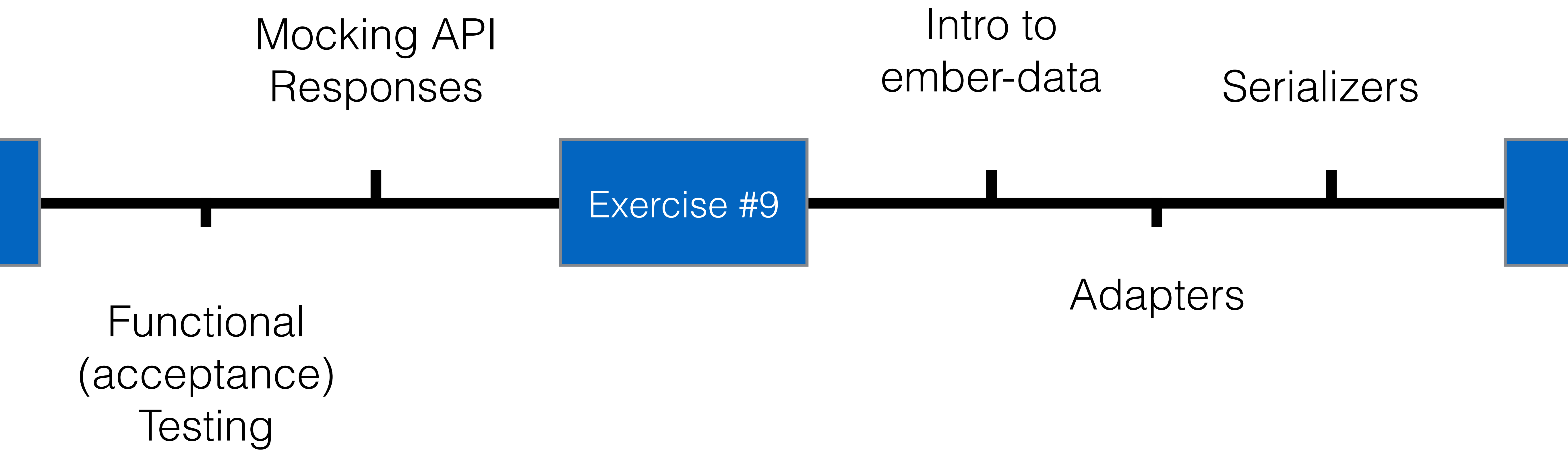
Agenda


Components
Exercise #4
Exercise #5
Exercise #6
Loading & Error Substates
Computed Properties

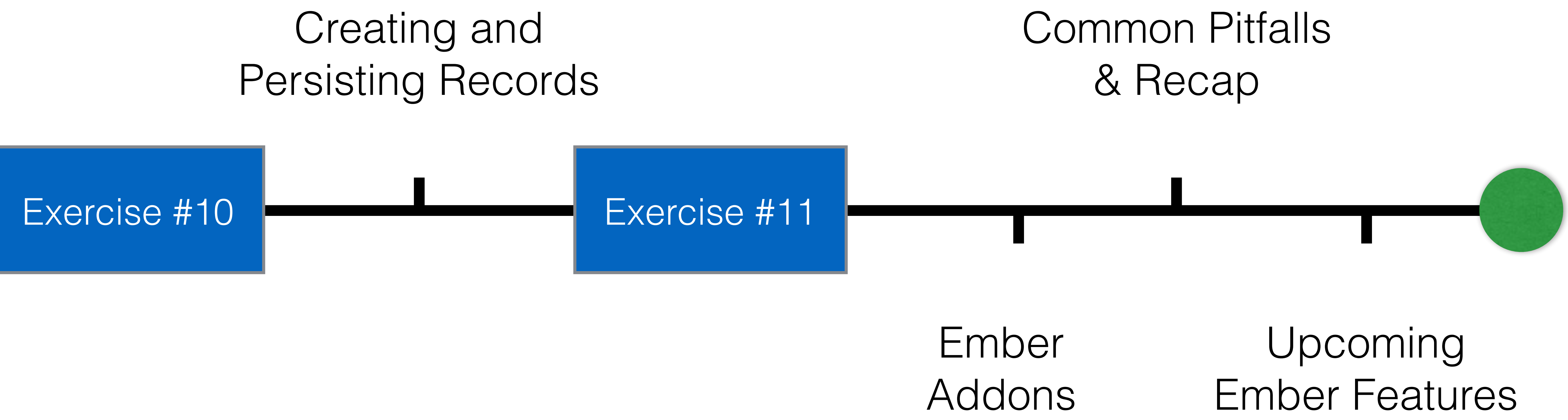
Agenda


Exercise #7
Intro to Testing w/ Ember
Unit Tests
Component Integration Tests
Exercise #8

Agenda

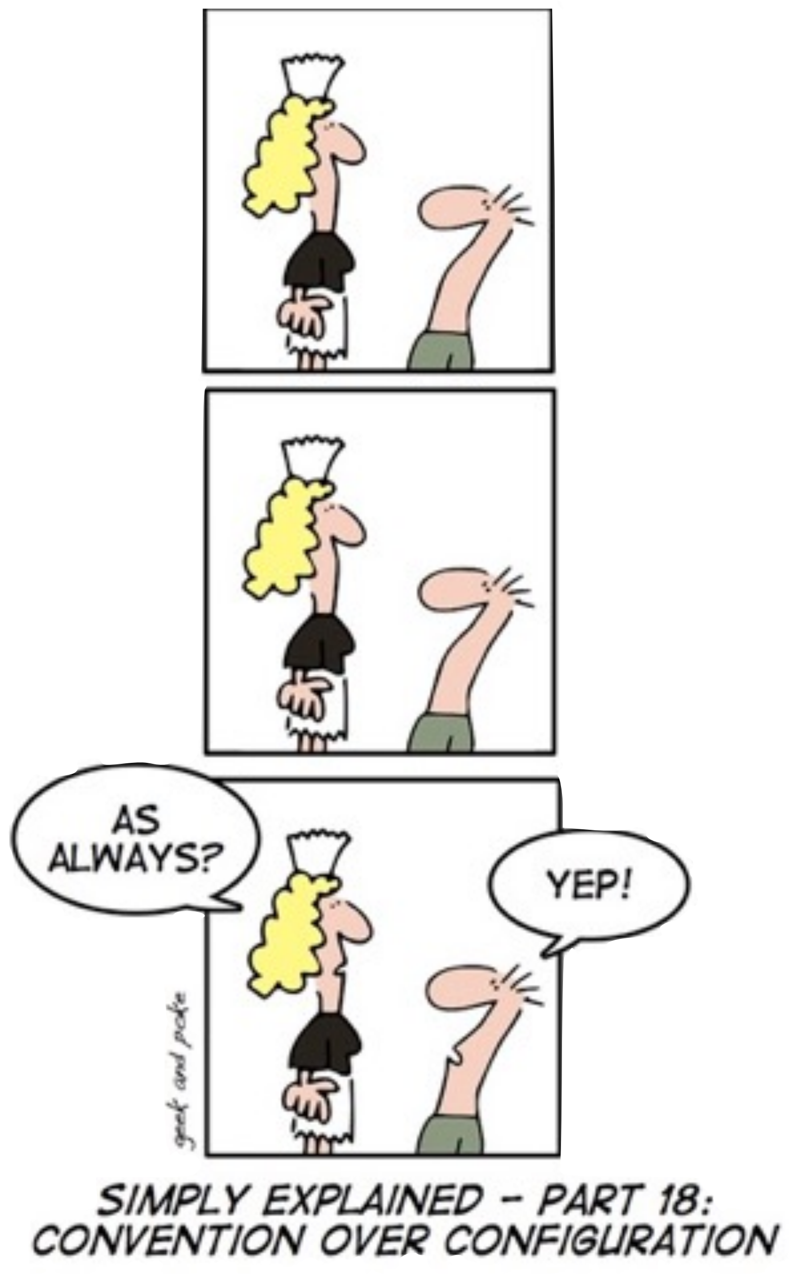

Mocking API Responses
Intro to ember-data
Serializers
Exercise #9
Functional (acceptance) Testing
Adapters

Agenda


Creating and Persisting Records
Common Pitfalls & Recap
Exercise #10
Exercise #11
Ember Addons
Upcoming Ember Features

Philosophy of Ember

- Designed for "ambitious apps"
- Focused on productivity & ergonomics
- Aligned with web standards
- A complete, wholistic solution
- Constantly growing & improving



The Ember Ecosystem

The Libraries

Framework	Ember.js
Data Persistence	Ember Data
Build Toolchain	Ember-cli
Animations	Liquid Fire
Virtualized List	List View

The Ember Ecosystem

The Microlibraries

Prioritized, Batched Queue	backburner.js
URL handling	route-recognizer.js
SPA routing	router.js
A+ Promises	rsvp.js
Template Engine	handlebars / HTMLbars
Asset Pipeline	broccoli.js

Goals of the workshop

You should leave feeling like you can…

- Plan out and implement a hierarchical routing layer
- Build dynamic templates using Handlebars, build your own Handlebars helpers
- Build ember components, and use them to expressively compose user interfaces
- Interact with a restful API using ember-data
- Write unit, functional and integration tests for your Ember app
- Build an ember addon, and use it in an ember app

Anatomy of an Ember App

Anatomy of an Ember App

Resource	Purpose
`/app`	Application source code
`/vendor`	Last resort for assets
`/public`	Pass-through in build pipeline (images, favicon, etc…)
`/tests`	Unit, functional, integration and acceptance tests
`/config`	App configuration
Managed by ember-cli	
`/tmp`	Intermediate build results
`/dist`	Destination for deployable assets (`ember build`)
`/node_modules`	NPM packages
`/bower_components`	Bower packages

Anatomy of an Ember App

Resource	Purpose
`/app`	Application source code
`/app/templates`	Handlebars templates
`/app/templates/`	Handlebars templates for components
`/app/routes`	Routes
`/app/components`	Components
`/app/helpers`	Handlebars helpers
`/app/models`	Models
`/app/styles`	CSS, SASS, LESS
`/app/initializers`	Initializers - for customizing app start-up
`/app/services`	Singletons
`/app/router.js`	Router
`/app/index.html`	Start-up HTML
`/app/app.js`	Ember Application object

Template Basics

Templates

- Handlebars - templating language
- Templates are compiled, and then populated with data via a context

```
Hello, <strong>{{firstName}} {{lastName}}</strong>!
```

```
define('examples/templates/index', ['exports'], function (exports) {

  'use strict';

  exports['default'] = Ember.HTMLBars.template((function() {
    return {
      ...
      buildFragment: function buildFragment(dom) {
        var el0 = dom.createDocumentFragment();
        var el1 = dom.createTextNode("Hello, ");
        dom.appendChild(el0, el1);
        var el1 = dom.createElement("strong");
        var el2 = dom.createComment("");
        dom.appendChild(el1, el2);
        var el2 = dom.createTextNode(" ");
        dom.appendChild(el1, el2);
        var el2 = dom.createComment("");
        dom.appendChild(el1, el2);
        dom.appendChild(el0, el1);
        var el1 = dom.createTextNode("!");
        dom.appendChild(el0, el1);
        return el0;
      },
      buildRenderNodes: function buildRenderNodes(dom, fragment,
contextualElement) {
        var element0 = dom.childAt(fragment, [1]);
        var morphs = new Array(2);
        morphs[0] = dom.createMorphAt(element0,0,0);
        morphs[1] = dom.createMorphAt(element0,2,2);
        return morphs;
      },
      statements: [
        ["content","firstName",["loc",[null,[1,15],[1,28]]]],
        ["content","lastName",["loc",[null,[1,29],[1,41]]]]
      ],
      ...
    };
  }()));
});
```

Templates

Conditionals

```
My pet goes {{if isDog "arf" "meow"}}
```

```
My pet goes
{{#if isDog}}
  arf
{{else}}
  meow
{{/if}}
```

```
My pet goes
{{#unless isDog}}
  meow
{{else}}
  arf
{{/unless}}
```

Block syntax begins with `{{#abc}}` and ends with `{{/abc}}`

Templates

Iteration

```
<ul>
{{#each myList as |myListItem|}}
   <li>{{myListItem}}</li>
{{/each}}
</ul>
```

```
<ul>
{{#each myList as |myListItem|}}
   <li>{{myListItem}}</li>
{{else}}
   <li>Sorry, list is empty!</li>
{{/each}}
</ul>
```

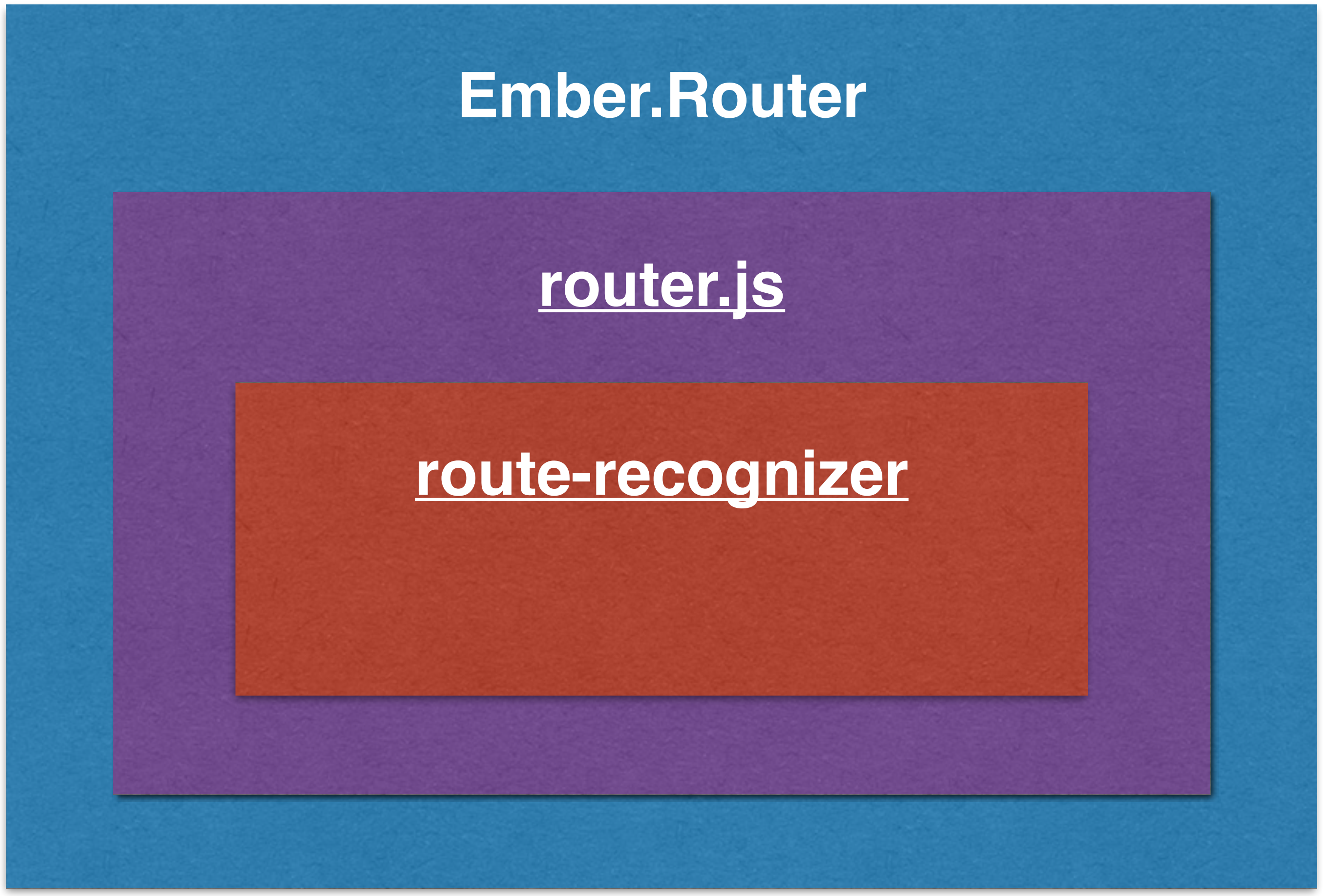
Router

Ember.Router

- One per app
- Manages transitions between URLs
- Usually leave it as-is, except for `Router.map`
- Core is a microlibrary: router.js

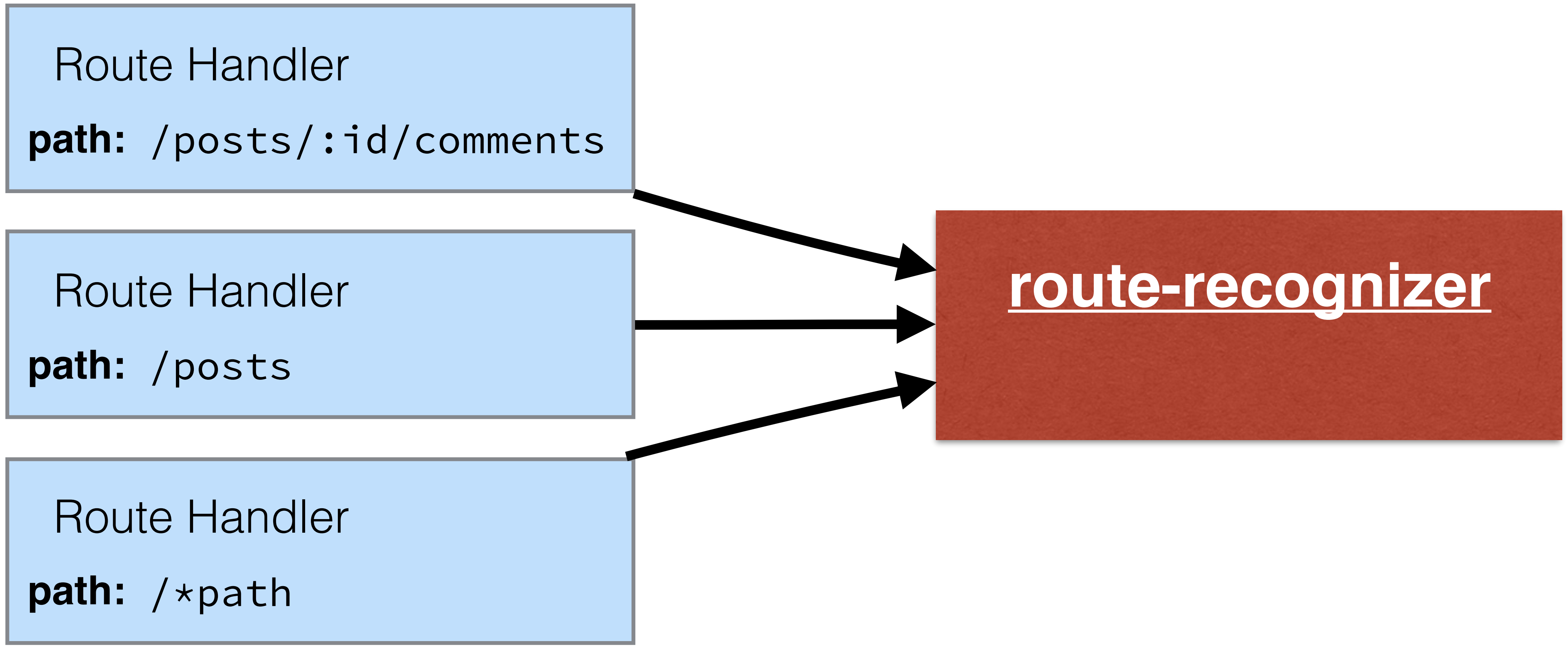
Ember.Router

Microlibraries



Ember.Router

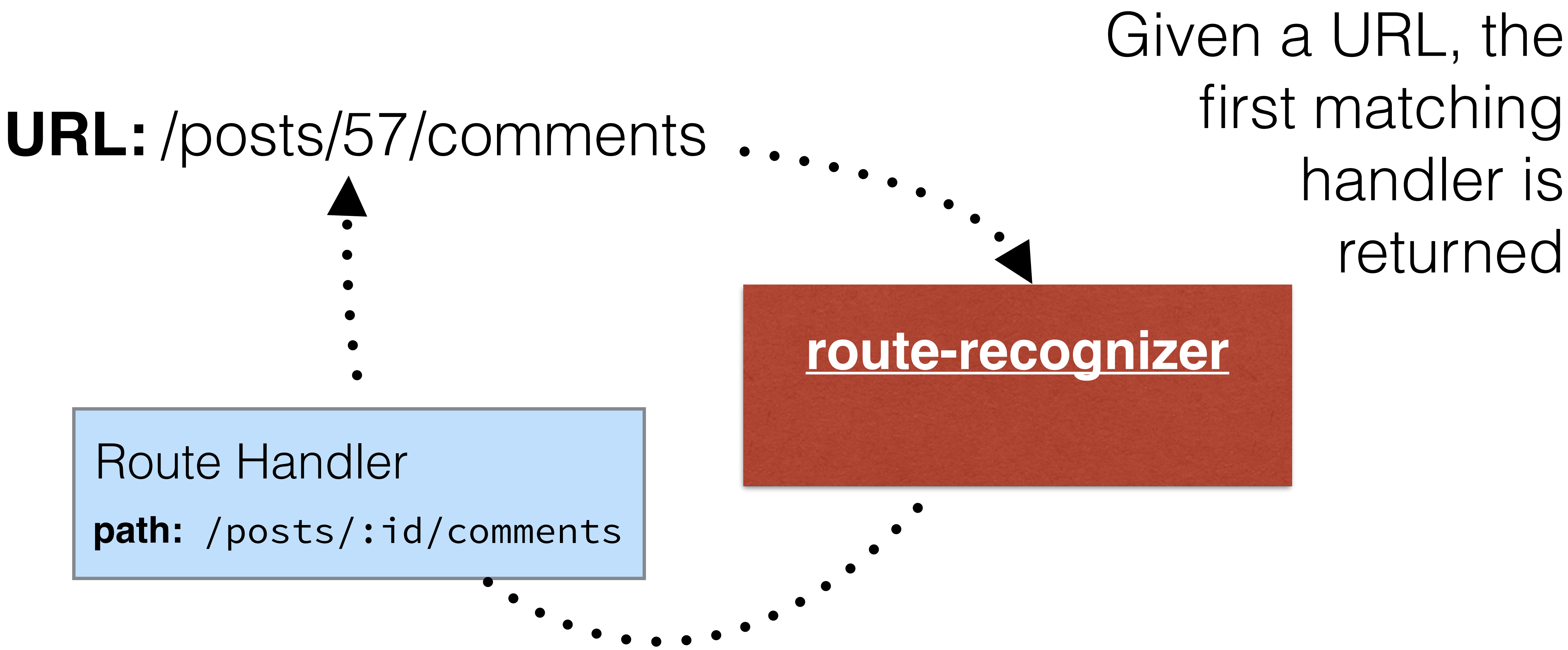
Microlibraries - route-recognizer



Route handlers are registered, each with a path

Ember.Router

Microlibraries - route-recognizer


Given a URL, the first matching handler is returned
URL: /posts/57/comments
route-recognizer
Route Handler
path: /posts/:id/comments

1. Define map of path to handler

```
router.map(function(match) {
  match(“/posts/:id")
    .to("showPost");
  match(“/posts")
    .to("postIndex");
  match("/posts/
new").to("newPost");
});
```

2. Define the handlers

```
var myHandlers = {
  showPost: {
    model: function(params) {
      // can be a promise!
      return $.get(
          "post/" + params.id
          );
    },
    setup: function(post) {
      // Render something
    }
  }
}
router.getHandler = function(name){
  return myHandlers[name];
};
```

3. On URL change, tell router

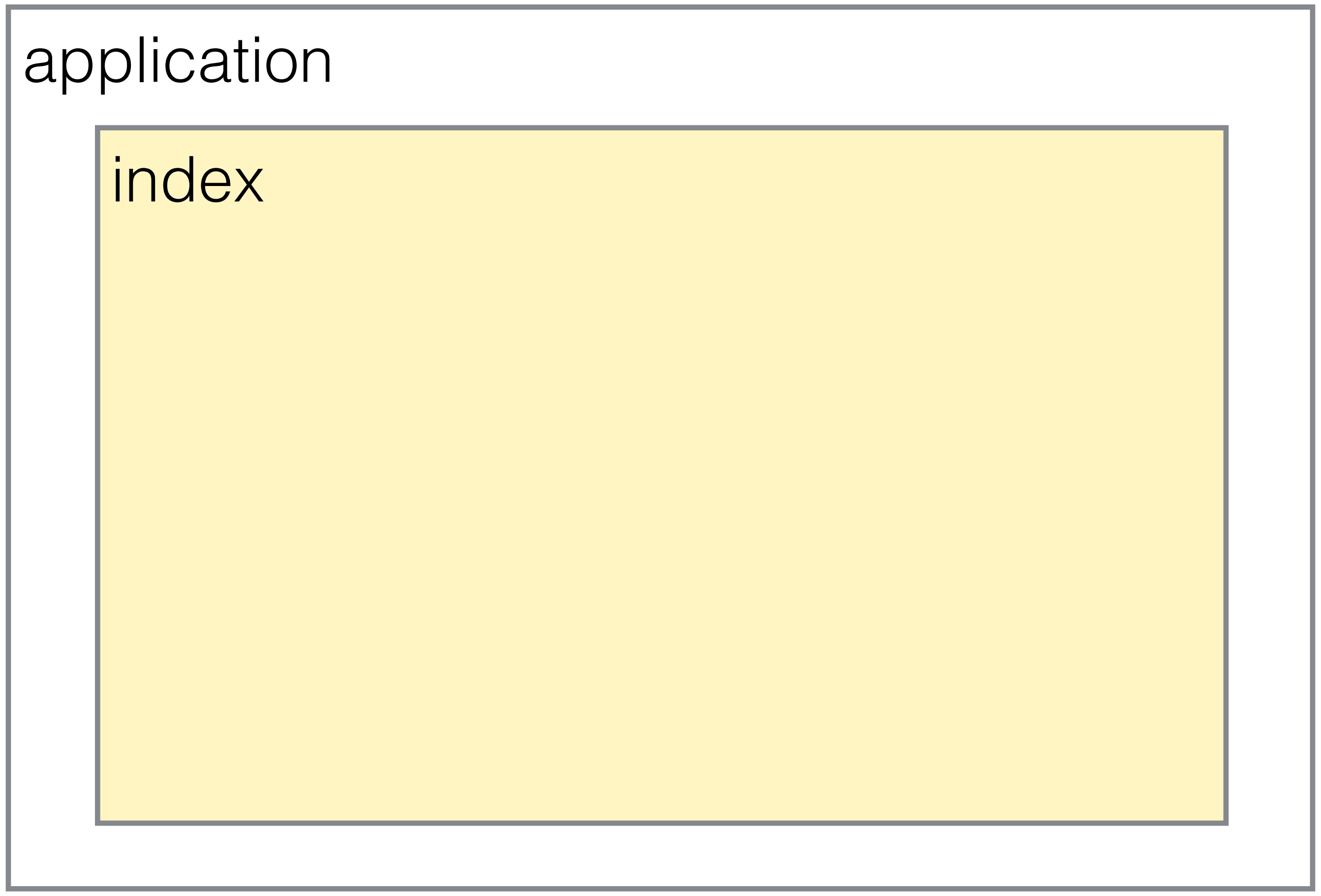
```
urlWatcher.onUpdate(funct
ion(url) {
  router.handleURL(url);
});
```

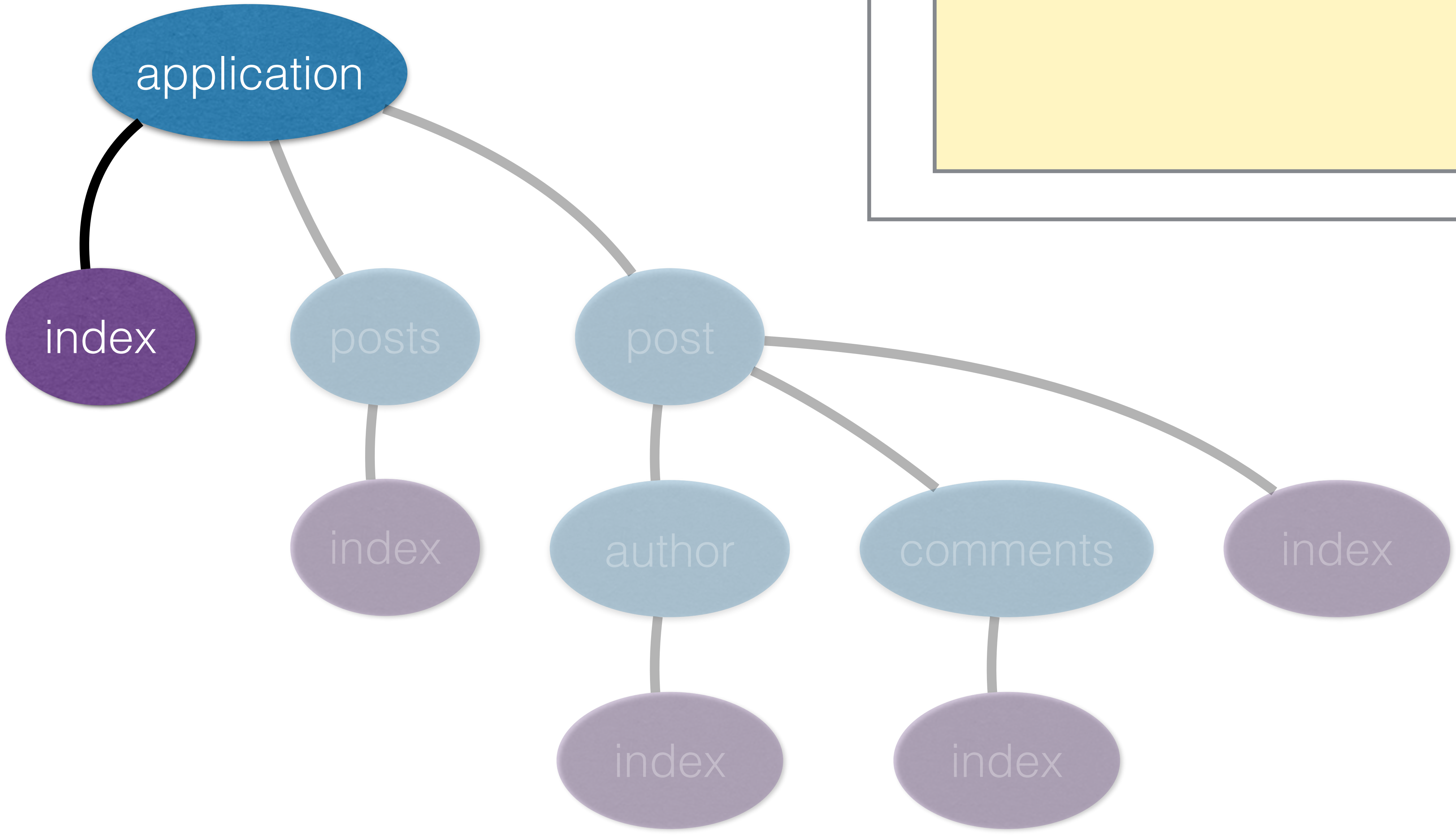
Ember.Router

Router - Routes as a Hierarchy

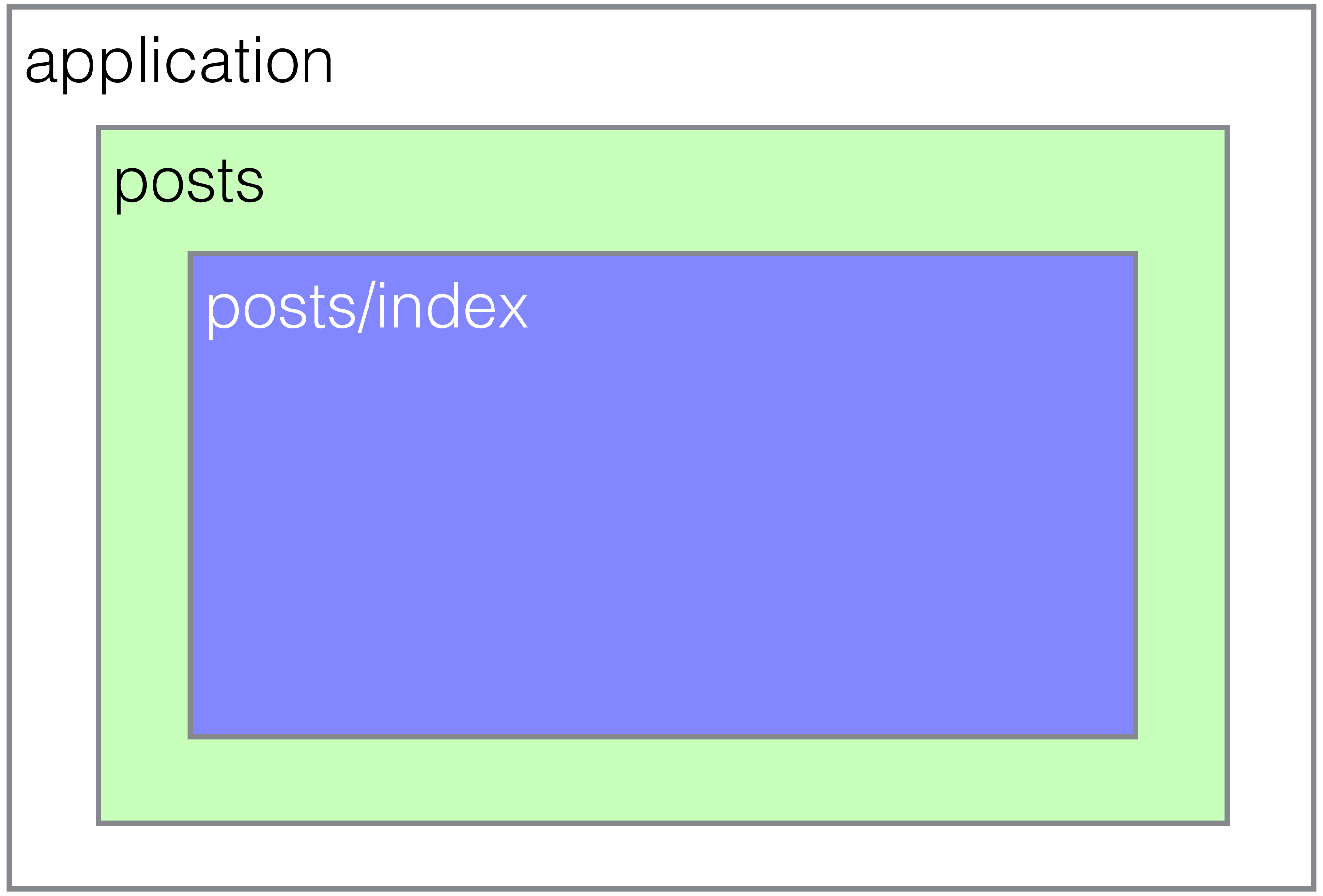
```
Router.map(function() {
  // IMPLIED index            /
 this.route(‘posts'); //      /posts
  this.route('post', {path: 'post/:id'}, function() {
    // IMPLIED index          /post/123
    this.route('author'); //  /post/123/author
    this.route('comments');// /post/123/comments
 });
});
```

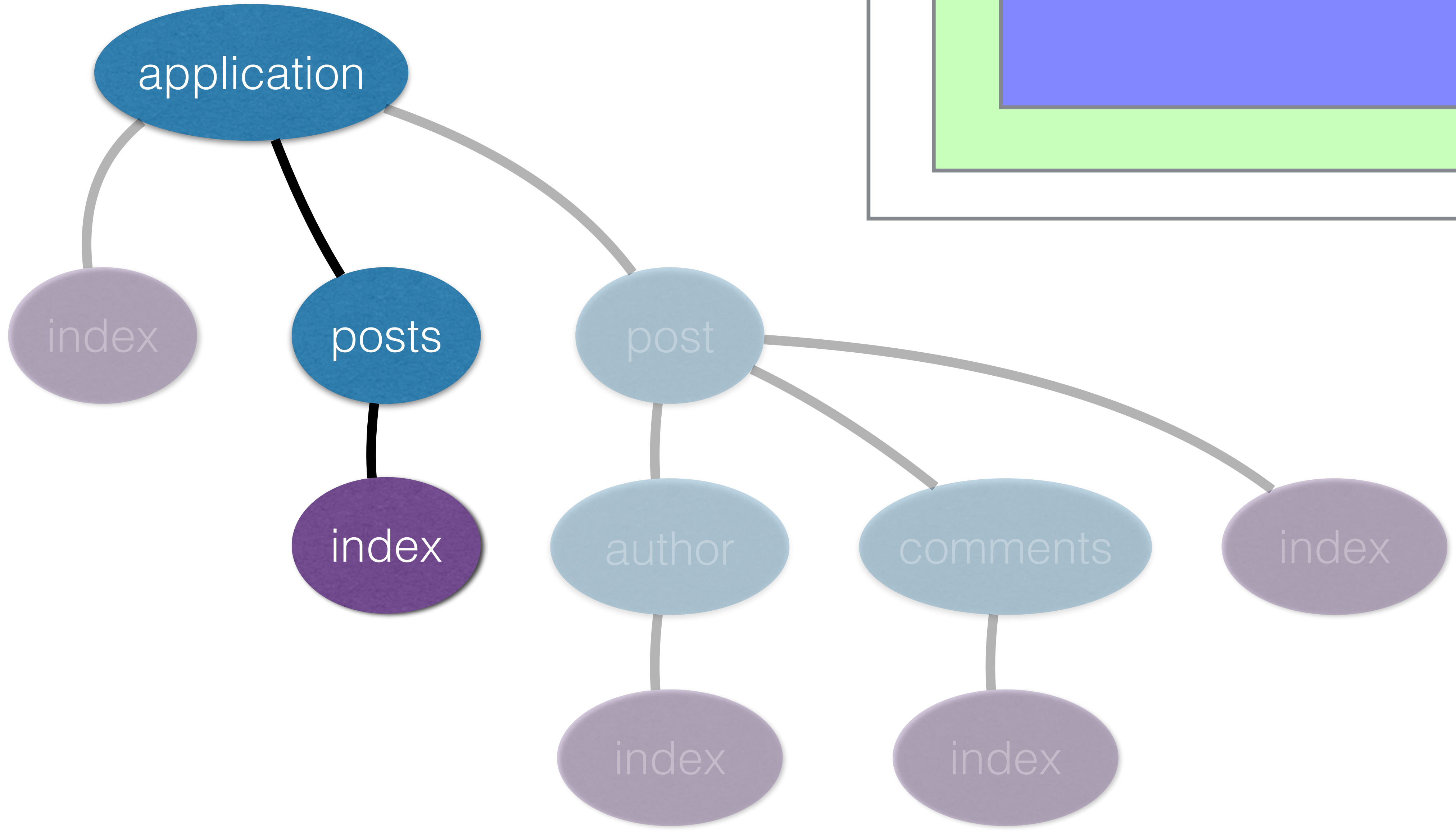
URL: /


application
index

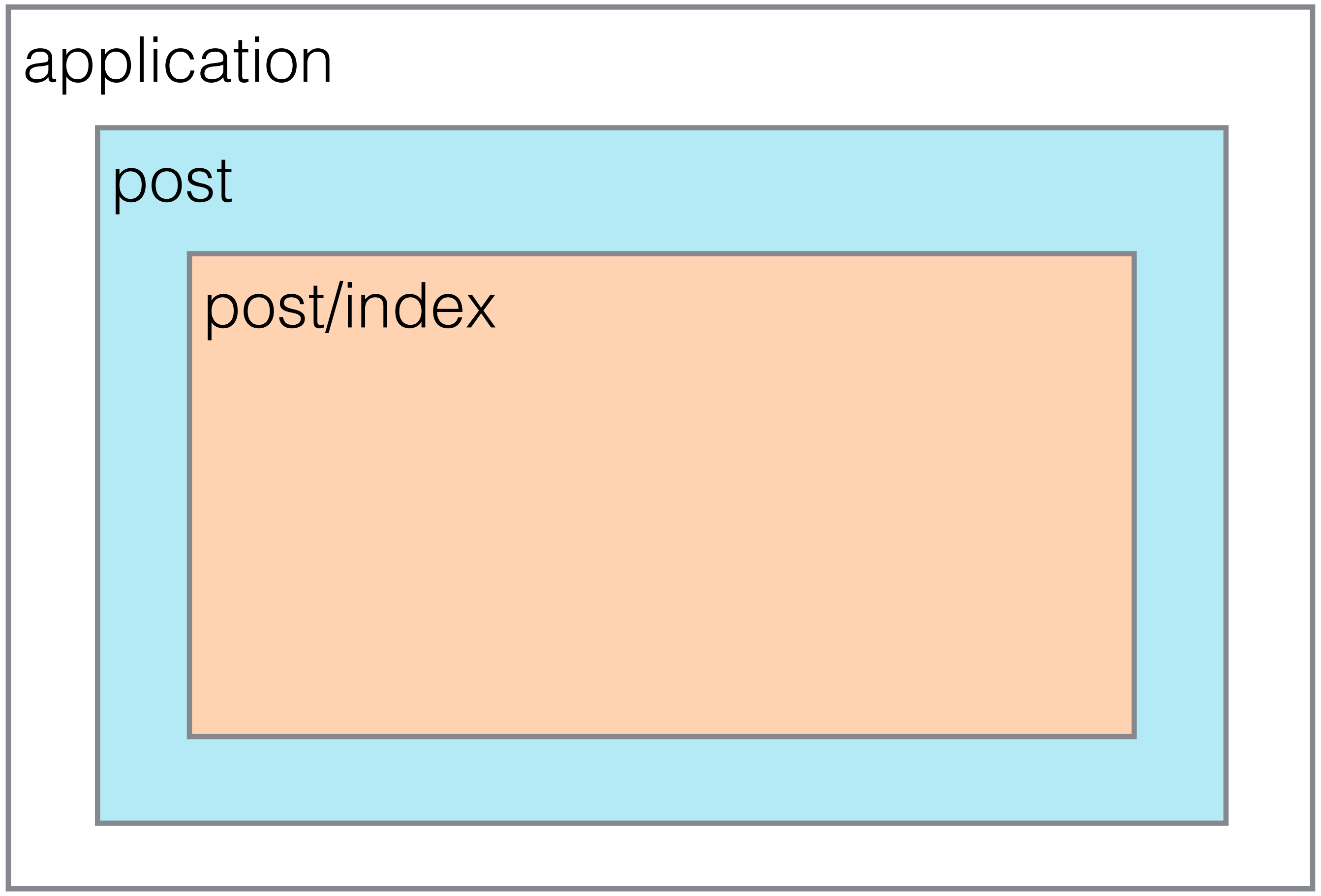

application
index
posts
post
index
author
comments
index
index
index

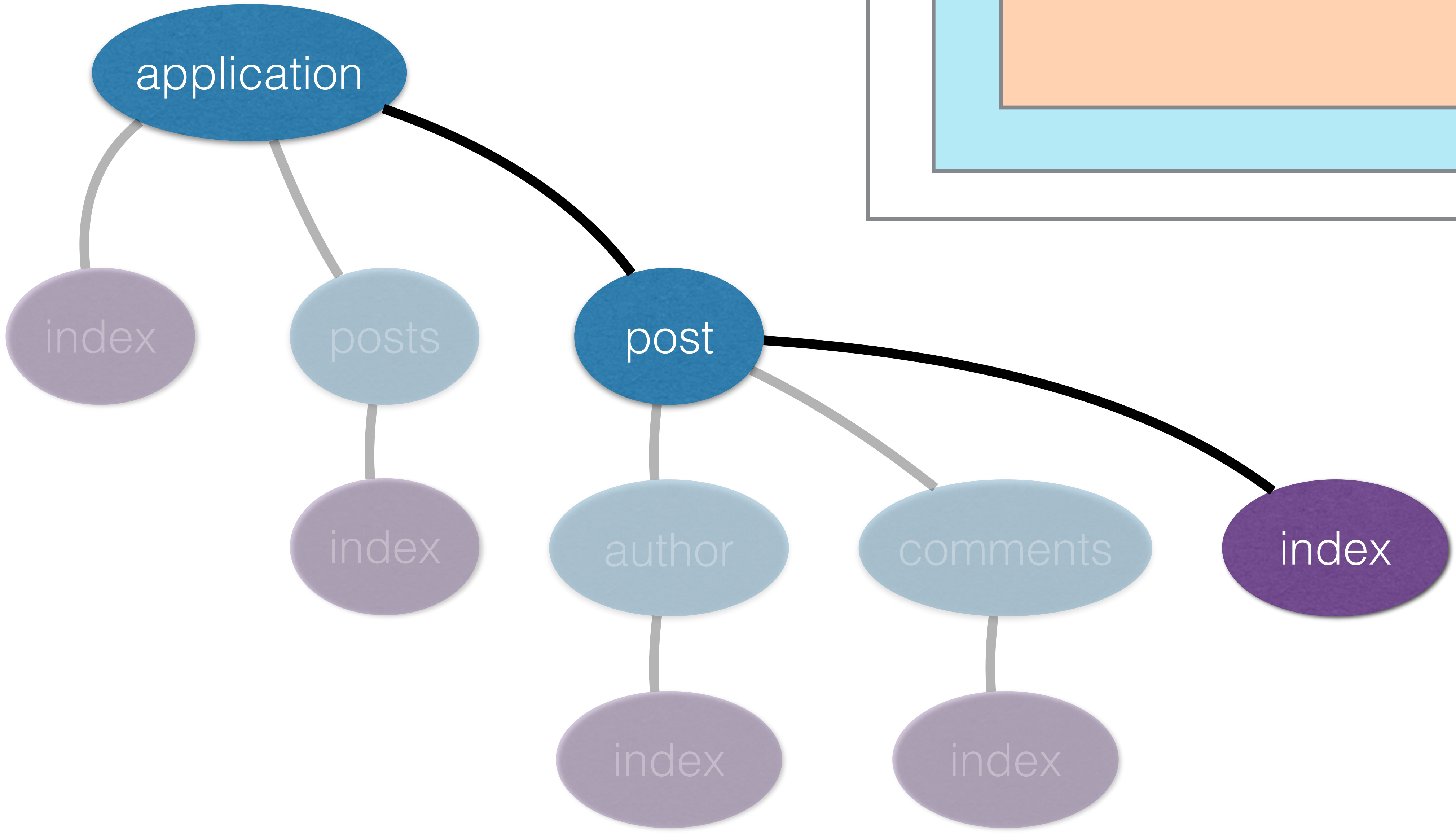
URL: `/posts/`


application
posts
posts/index

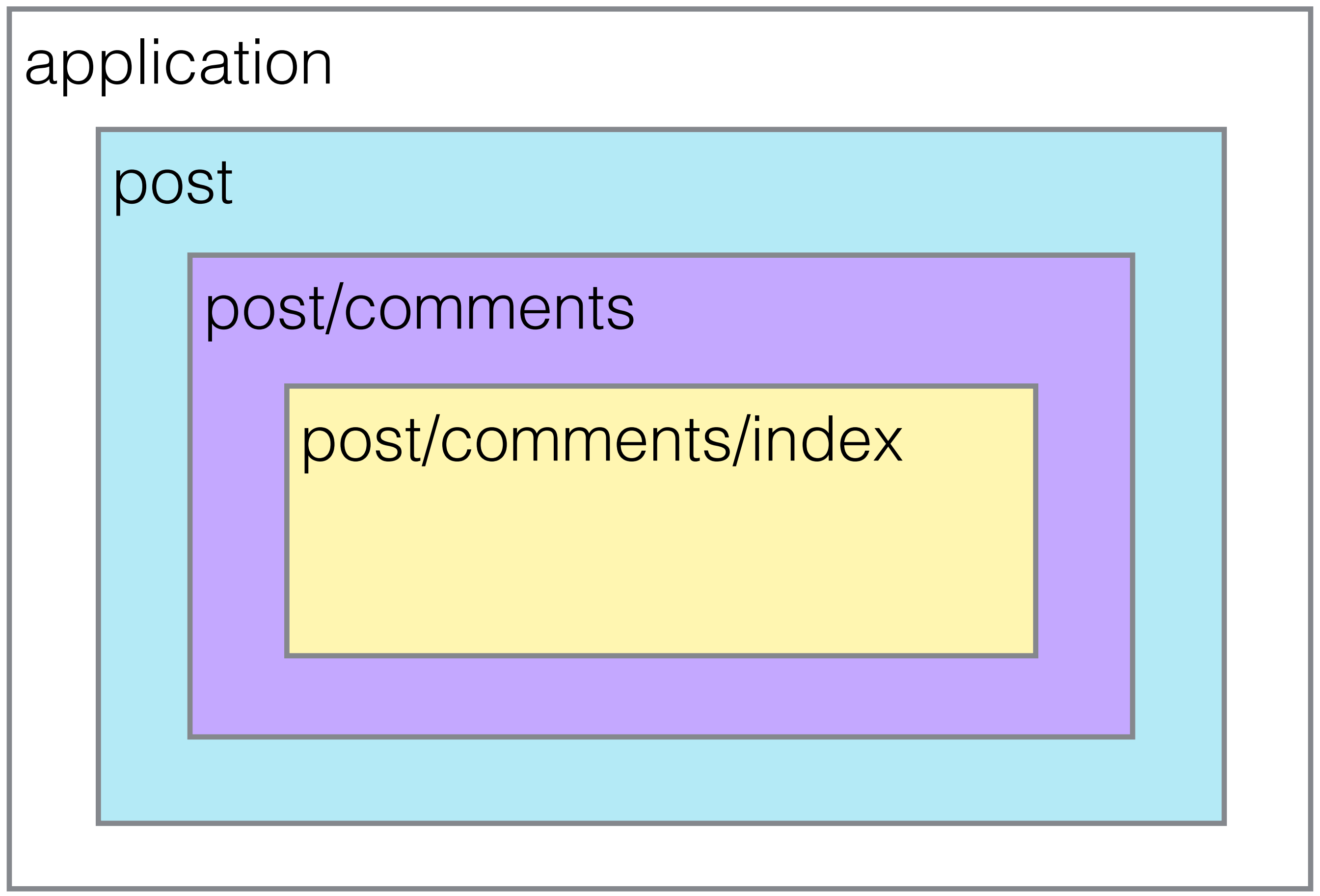

application
index
posts
post
index
author
comments
index
index
index

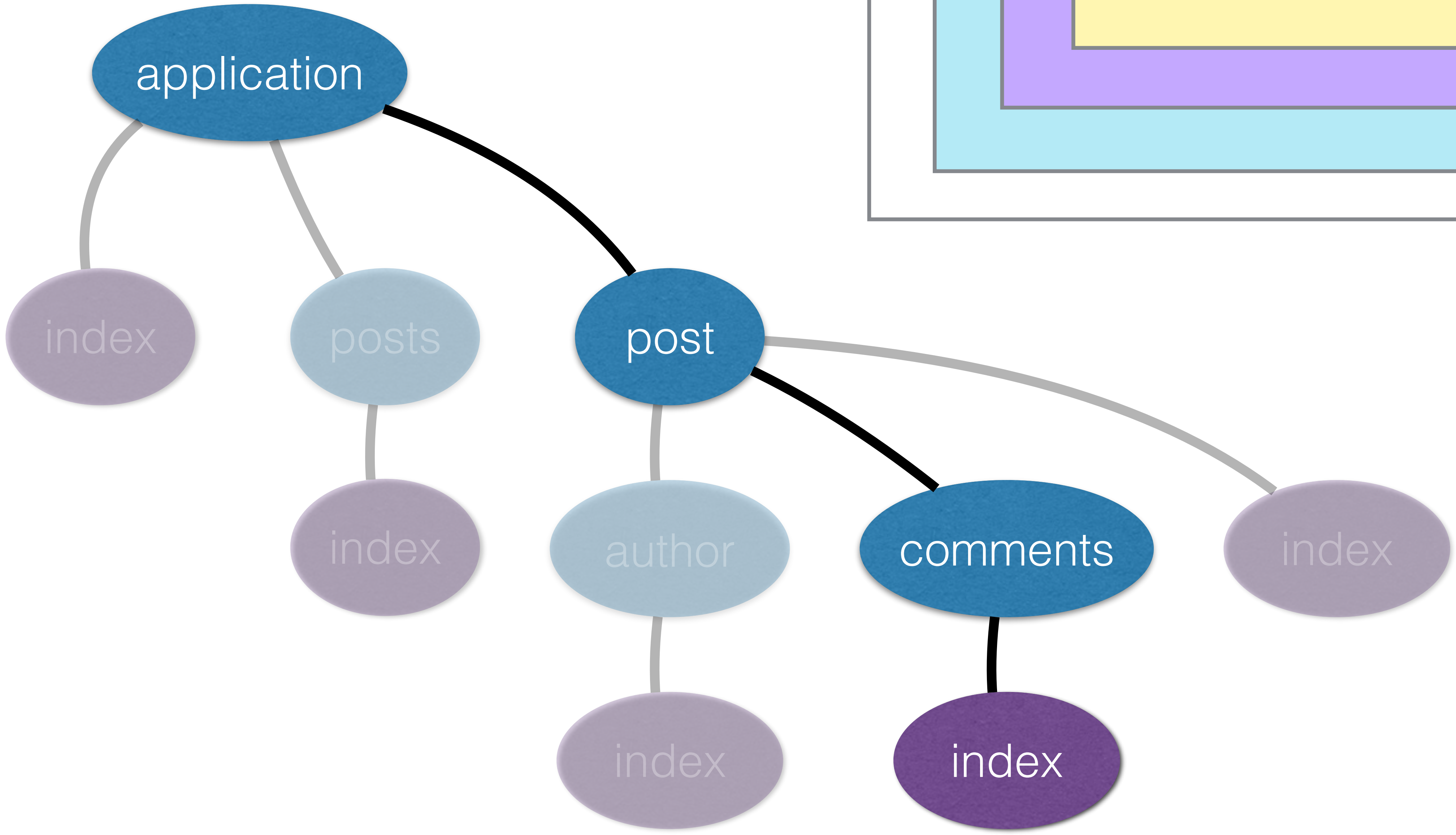
URL: `/post/31`


application
post
post/index

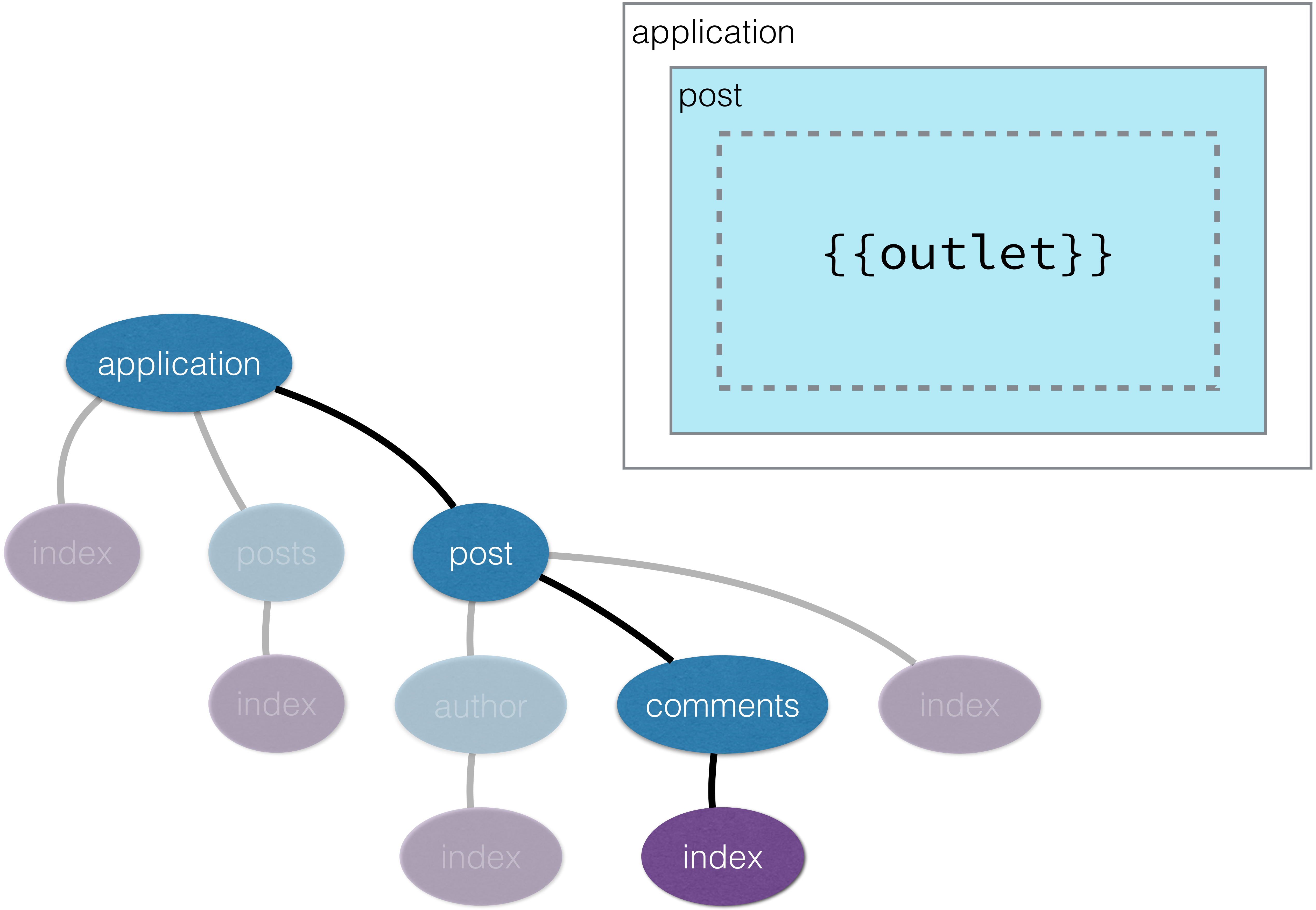

application
index
posts
post
index
author
comments
index
index
index

URL: `/post/31/comments/`



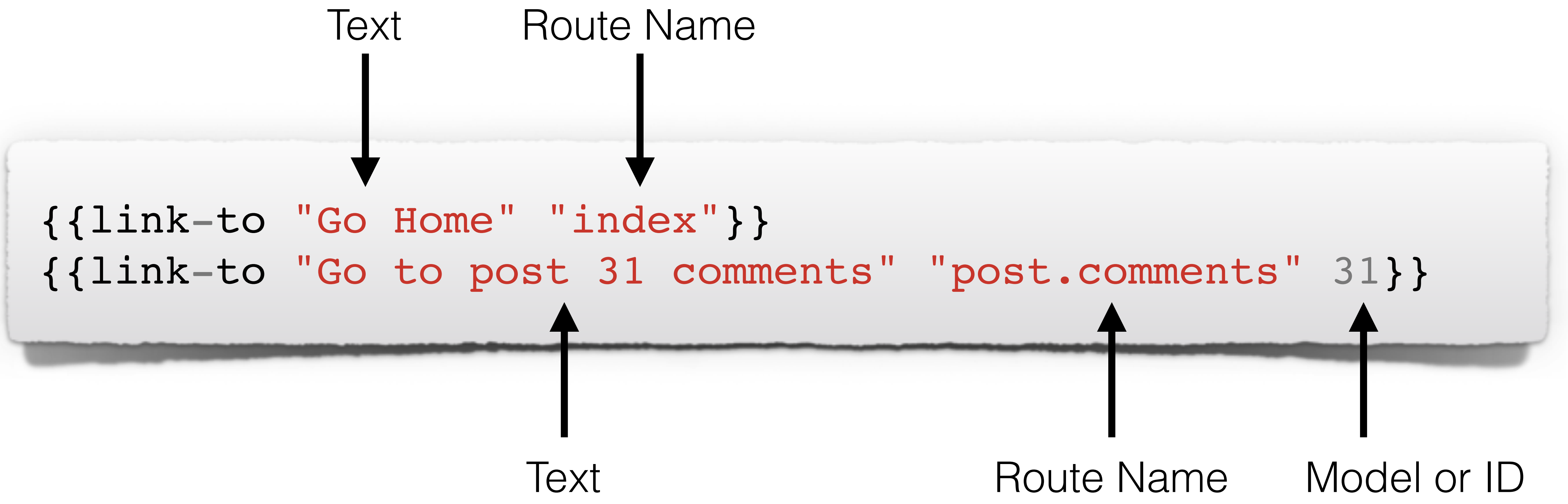


URL: `/post/31/comments/`



Ember.Router

link-to - for internal links to routes



```
{{link-to "Go Home" "index"}}
{{link-to "Go to post 31 comments" "post.comments" 31}}
```

Exercise #1

Setup your basic routes & placeholder templates

Project Setup

I’ve done some boring stuff for you!

Command	**Description**
`ember new github-ui`	Create a new ember.js project
`cd github-ui`	Change directory into newly created project
`ember install ember-cli-sass`	Install support for SASS
`mv app/styles/app.css app/styles/app.scss`	Make the “root” stylesheet a .scss file
`ember install ember-cli-bourbon`	Install bourbon SASS library
`bower install --save ember#beta`	Install latest beta version of ember.js
`bower install --save ember-data#beta`	Install latest beta version of ember-data

Exercise #1

- Set up some basic routes and templates for our app

Example URL	Description
/	Placeholder text “<h1>Homepage</h1>”
/orgs	List of some github orgs
/org/emberjs	Placeholder text “<h1>Org: Emberjs</h1>”
/org/emberjs/repos	List of repos in the ember's org
/org/emberjs/ember.js	Placeholder text “<h1>Repo: Ember.js</h1>”
/org/emberjs/ember.js/issues	List of issues in the ember.js repo, owned by the emberjs org
/org/emberjs/ember.js/contributors	List of contributors to the ember.js repo
ALL OTHER URLS	404 page

Exercise #1

/orgs

Orgs

- facebook
- netflix
- yahoo
- emberjs

/org/facebook/repos

back to orgs

Facebook

- react
- relay
- watchman
- react-native

/o/f/r/issues

back to orgs

Facebook / react

ISSUES CONTRIBUTORS

- #123 - Issue description goes here
- #456 - Issue description goes here

/o/f/r/contributors

back to orgs

Facebook / react

ISSUES **CONTRIBUTORS**

- User
- User

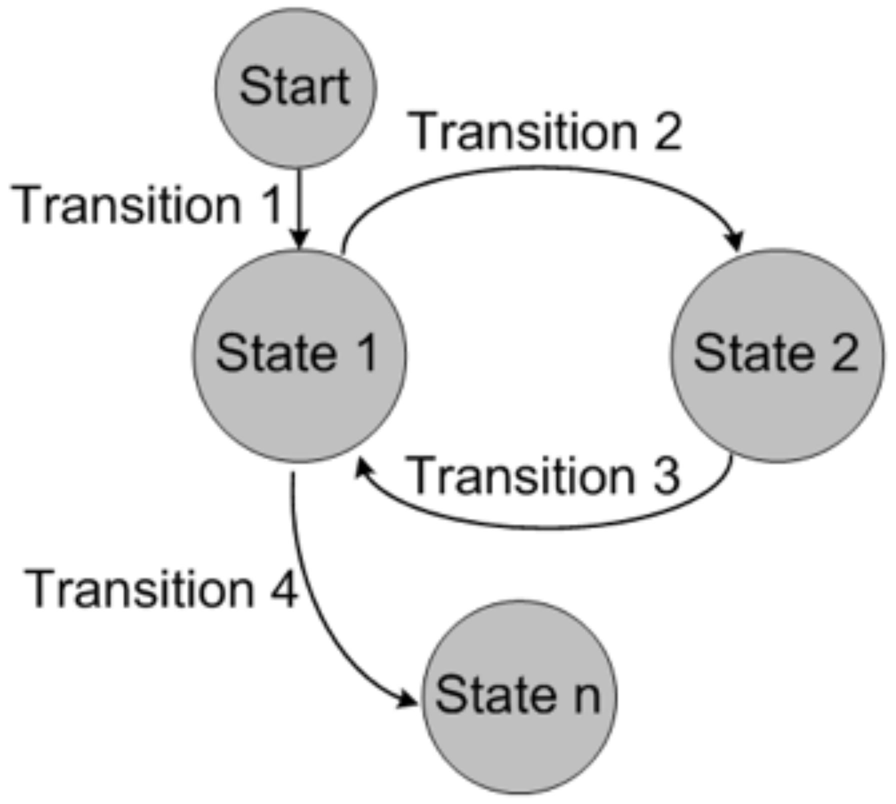
Exercise #1

- Hardcode content for now, we’ll make it dynamic later
- Don’t worry about CSS
- Orgs list should have working links to drill in to some org pages
- Org pages should have a working “back to orgs” link

Routes

Routes

- Router is a finite state machine
- Route manages transitions
  - Loads templates and models
  - Can handle user events



Routes

Request Lifecycle

▨ **Promise Aware**

`activate`	`()`	Called when router enters the route
▨ **`beforeModel`**	`(transition)`	First entry validation hook
▨ **`model`**	`(params, transition)`	Convert URL to a model
▨ **`afterModel`**	`(modelObj, transition)`	Called after model is resolved
`redirect`	`(modelObj, transition)`	Similar to afterModel
`setupController`	`(controller, model)`	Setup the controller for the current route
`renderTemplate`	`(controller, model)`	Render the template for the current route
`deactivate`	`()`	Called when router leaves route

Routes

`beforeModel (transition)`

This is the proper place for…

- A redirect that doesn’t require the resolved model
- Async entry validation

By default this does…

- Nothing

Routes

```
model(params, transition)
```

This is the proper place for…

- API call(s) to fetch data based on current URL

By default this does…

- If route has dynamic segment like `:post_id`, fetch a record `post` with id `post_id`

Routes

`model(params, transition)`

- Whatever this returns will be available
  - in template as `content`
  - in routes via `this.modelFor(‘post’)`
- If you need to make multiple API calls, use `Ember.RSVP.hash()`
- Common location for some light data massaging

Routes

```
afterModel(modelObj, transition)
```

This is the proper place for…

- Entry validation that requires the resolved model

By default this does…

- Nothing

Routes

```
redirect(modelObj, transition)
```

Seems just like afterModel. Why do we need this?

- a redirect in `afterModel` will abort the current transition and start a new one
    - Think about the case where you're redirecting into a child route

Routes

`setupController (controller, model)`

This is the proper place for…

- Additional setup of the context for your template

By default this does…

- A lot of really important stuff. Make sure to call `super` if you extend!

Do not be tempted to…

- Use this as a place to “reset” your controller

Routes

Transitioning from within a route

```
this.transitionTo (routeName, models, options)
```

Example

```
    this.transitionTo (“post”, myPost)

    this.transitionTo (“post”, 31)

    this.transitionTo (“posts”)
```

Routes

Transitioning from within a route

```
this.replaceWith (routeName, models, options)
```

Example

```
this.replaceWith (“post”, myPost)

this.replaceWith (“post”, 31)

this.replaceWith (“posts”)
```

Exercise #2

Routes providing data, and redirecting

Exercise #2a

- Set up redirects for some of our less meaningful routes

Example URL	Description
/	Redirect to `/orgs`
/orgs	List of some github orgs
/org/emberjs	Redirect to `/org/emberjs/repos`
/org/emberjs/repos	List of repos in the ember's org
/org/emberjs/ember.js	Redirect to `/org/emberjs/ember.js/`
/org/emberjs/ember.js/	List of issues in the ember.js repo, owned by
/org/emberjs/ember.js/	List of contributors to the ember.js repo
ALL OTHER URLS	404 page

Exercise #2b

- `model()` hook of route corresponding to `/orgs` should return an array of github org names

- Template corresponding to `/orgs` should refer to the `content` property, iterating over it to build the list of github orgs

    - You'll want to use `{{#each}}`

- Move data to the route for this page too `/org/:id/repos`

```
[
  {id: "emberjs"},
  {id: "ember-cli"},
  {id: "microsoft"},
  {id: "yahoo"},
  {id: "netflix"},
  {id: "facebook"}
];
```

Exercise #2b

Routes to use in `{{link-to}}`

URL of page	Route name to use for each link
`/orgs`	`org`
`/org/emberjs/repos`	`org.repo`

You may need to create two new routes

```
ember g route org/index
ember g route org/repo/index
```

Templates

Iteration

```
<ul>
{{#each myList as |myListItem|}}
   <li>{{myListItem}}</li>
{{/each}}
</ul>
```

```
<ul>
{{#each myList as |myListItem|}}
   <li>{{myListItem}}</li>
{{else}}
   <li>Sorry, list is empty!</li>
{{/each}}
</ul>
```

Services

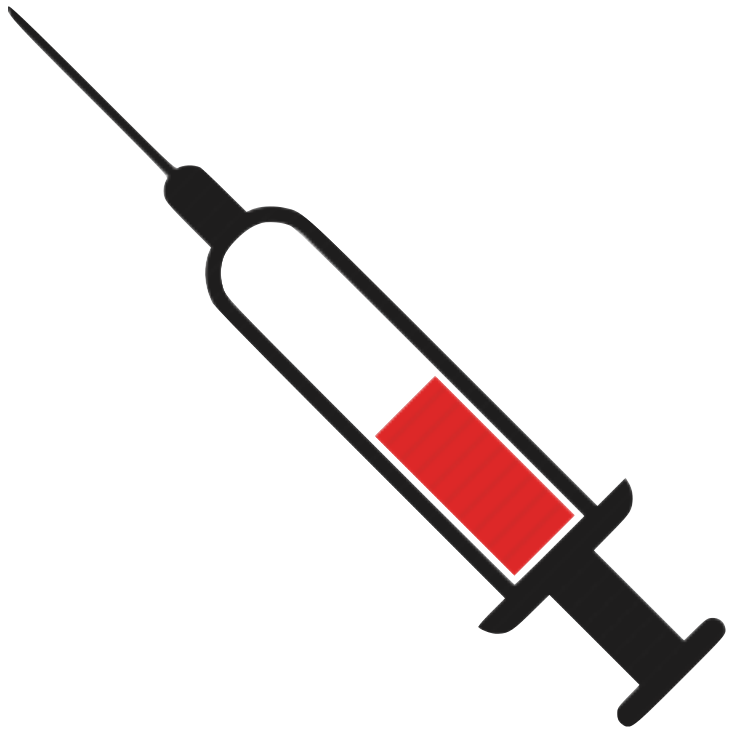
Services

- Singletons
- Very long lifecycle
- Means of managing shared app functionality/state
- live in the `/app/services` folder
- On need to know basis, `Ember.inject.service()`

Services

`Ember.inject.service(name)`

- `name` is optional
- Return value can be treated like a property
- Lazy
- No restrictions to what you can inject onto



Actions

Actions

- The primary means of handling user interaction
- action-binding is similar to data-binding
- Can be handled by Routes, Components, Views and Controllers

```
<span {{action 'thingWasClicked'}}>
  Click Here!
</span>
```

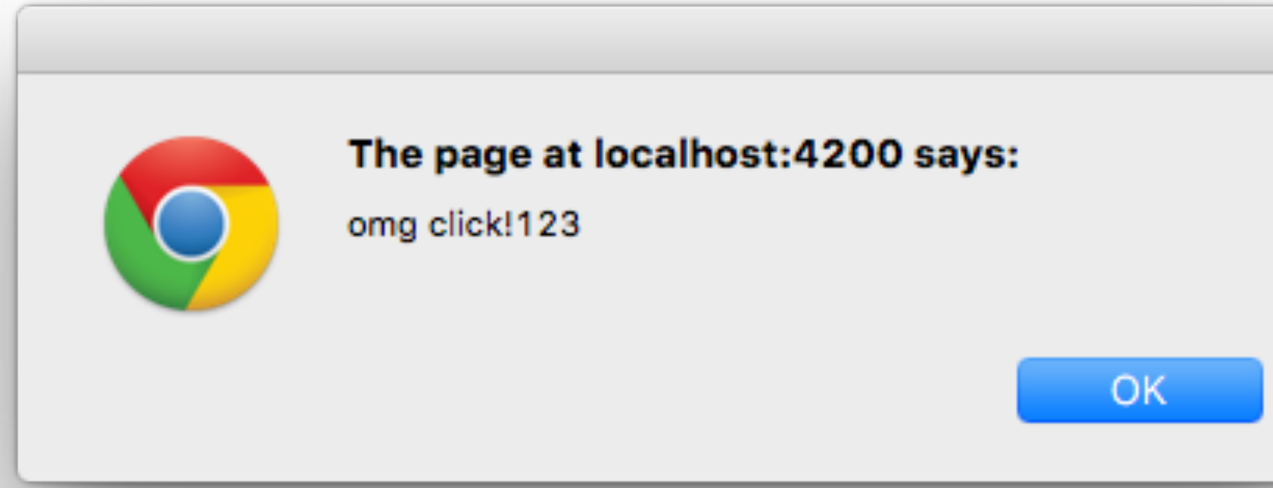
```
Ember.Route.extend({
  actions: {
    thingWasClicked() {
      alert('omg click!');
    }
  }
});
```

Actions

- You can send data along too

```
<span {{action ‘thingWasClicked' 123}}>Click Here!
</span>
```

```
Ember.Route.extend({
 actions: {
    thingWasClicked(num) {
      alert('omg click!' + num);
    }
  }
});
```


The page at localhost:4200 says:

omg click!123

`Ember.Object` and Simple Properties

Ember.Object

- Foundational Ember type. Base class of Ember.Route, Ember.Service, Ember.Component, etc…
- use `this.get` and `this.set` to access and mutate properties
- A lot more useful stuff, which we'll get to later!

Ember.Object

```
const me = Ember.Object.create({
  firstName: 'Mike'
});

console.log(me.get(‘firstName')); // "Mike"
```

Create objects with initial property values

Ember.Object

```
const me = Ember.Object.create({
  firstName: 'Mike'
});

console.log(me.get('firstName')); // "Mike"

me.set('firstName', 'Marc');

console.log(me.get('firstName')); // "Marc"
```

Use `set` to change property values

Ember.Object

```
const me = Ember.Object.create({
  name: {
    first: 'Mike'
  }
});

console.log(me.get('name.first')); // "Mike"
```

You can get/set on a property path

Ember.Object

```
const me = Ember.Object.create({
  nameParts: ['Mike', 'North']
});

console.log(me.get('nameParts').join(' ')); // "Mike North"

// Remove last object, using KVO-compliant API
me.get('nameParts').popObject();

console.log(me.get('nameParts').join(' ')); // "Mike"
```

When mutating arrays, first `get` the array and then manipulate it

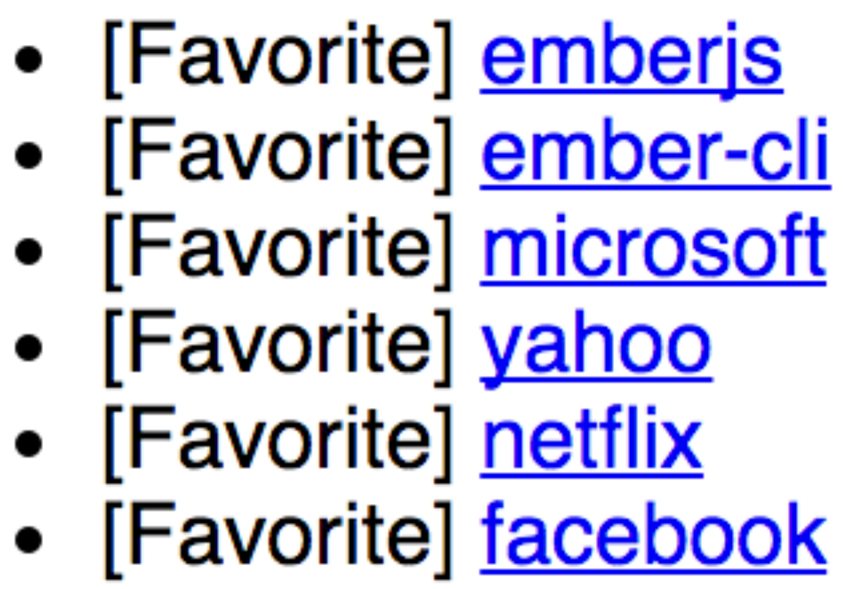
Exercise #3

Favorite service and actions

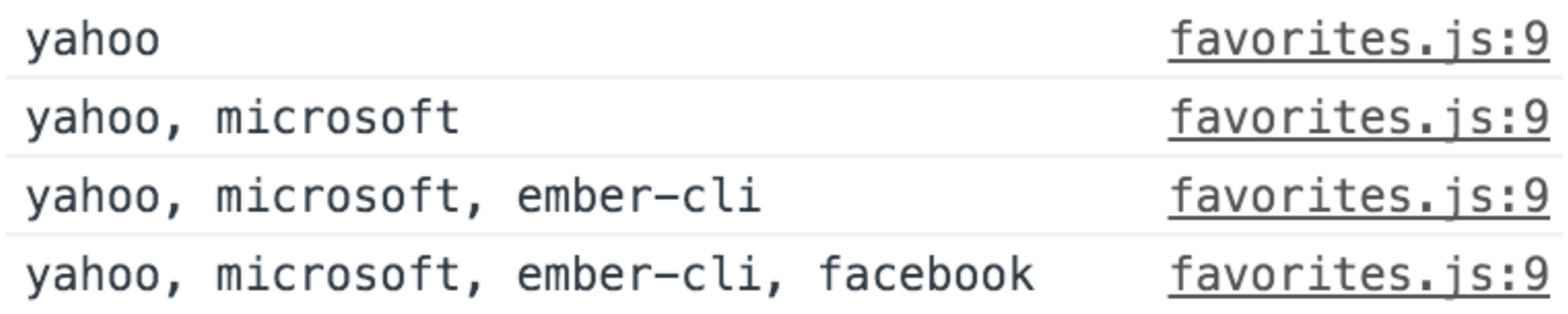
Exercise #3

- Create a `favorites` service
  - The service should have an `items` array property
- Inject the service onto the route for the `/orgs` page
- On the template for the `/orgs` page, create a span inside your `{{#each}}` loop, with an action bound to it
  - The action should add the respective item to the `items` property on the `favorites` service, if it's not already present
  - When you add an item to the array, `console.log` the entire array. This code should be in the service.
- PROTIP: Ember shims [`Array.prototype.addObject`], which checks for existing presence

Exercise #3



- [Favorite] emberjs
- [Favorite] ember-cli
- [Favorite] microsoft
- [Favorite] yahoo
- [Favorite] netflix
- [Favorite] facebook



```
yahoo                                              favorites.js:9
yahoo, microsoft                                   favorites.js:9
yahoo, microsoft, ember-cli                        favorites.js:9
yahoo, microsoft, ember-cli, facebook              favorites.js:9
```

Talking to APIs & using simple models

Retrieving Data

- If you can use `$.get`, you can talk to APIs
- If you need to set additional context up (i.e., if a child route/template needs to know about a model from another route's model), `setupController` is a decent place to do it
- the Controller is the context for top-level templates (for now).

```
import Ember from 'ember';

export default Ember.Route.extend({
  model(params) {
    // Get the "id" property from the model resolved in the "org"
route
    let orgName = this.modelFor('org').id;
    // Fetch API data
    return $.get(`https://api.github.com/orgs/${orgName}/repos`);
  },

  setupController(controller) {
    this._super(...arguments);
    // Make the model resolved in the "org" route available to
    //   this route's template, via a property called "org"
    controller.set('org', this.modelFor('org'));
  }
});
```

```
<h1>{{org.id}} repos</h1>

<ul>
  {{#each content as |repo|}}
  <li>{{link-to repo.name 'org.repo' repo.name}}</li>
  {{/each}}
</ul>
{{outlet}}
```

Simple Models

- Not ember-data yet
- Plain JS objects are fine, as long as they have a property called `id` on them
  - If you're going to pass these objects into `{{link-to}}`, the `id` will be used for dynamic segments

Exercise #4

Talking to the Github API!

Exercise #4

- Except for the `/orgs` page, all hard-coded data from the template and routes should be replaced with real data
- Remember that child routes can access the models from parent routes via `this.modelFor(‘routeName’)`
- Pay attention to the JSON structure that the API returns. you will have to massage some data in the model hook

Exercise #4

```
// API Returns
{
   "id": 11635549,
   "name": "filter-expl
   "full_name": "Micros
}
```

```
// We want something li
{
   "id": "filter-explor
   "oldId": 11635549,
   "name": "filter-explorer",
   "full_name": "Microsoft/filter-explorer"
}
```

```
return $.get("<api url>").then(raw => {
  raw.oldId = raw.id;
  raw.id = raw.name;
  return raw;
});
```

Exercise #4

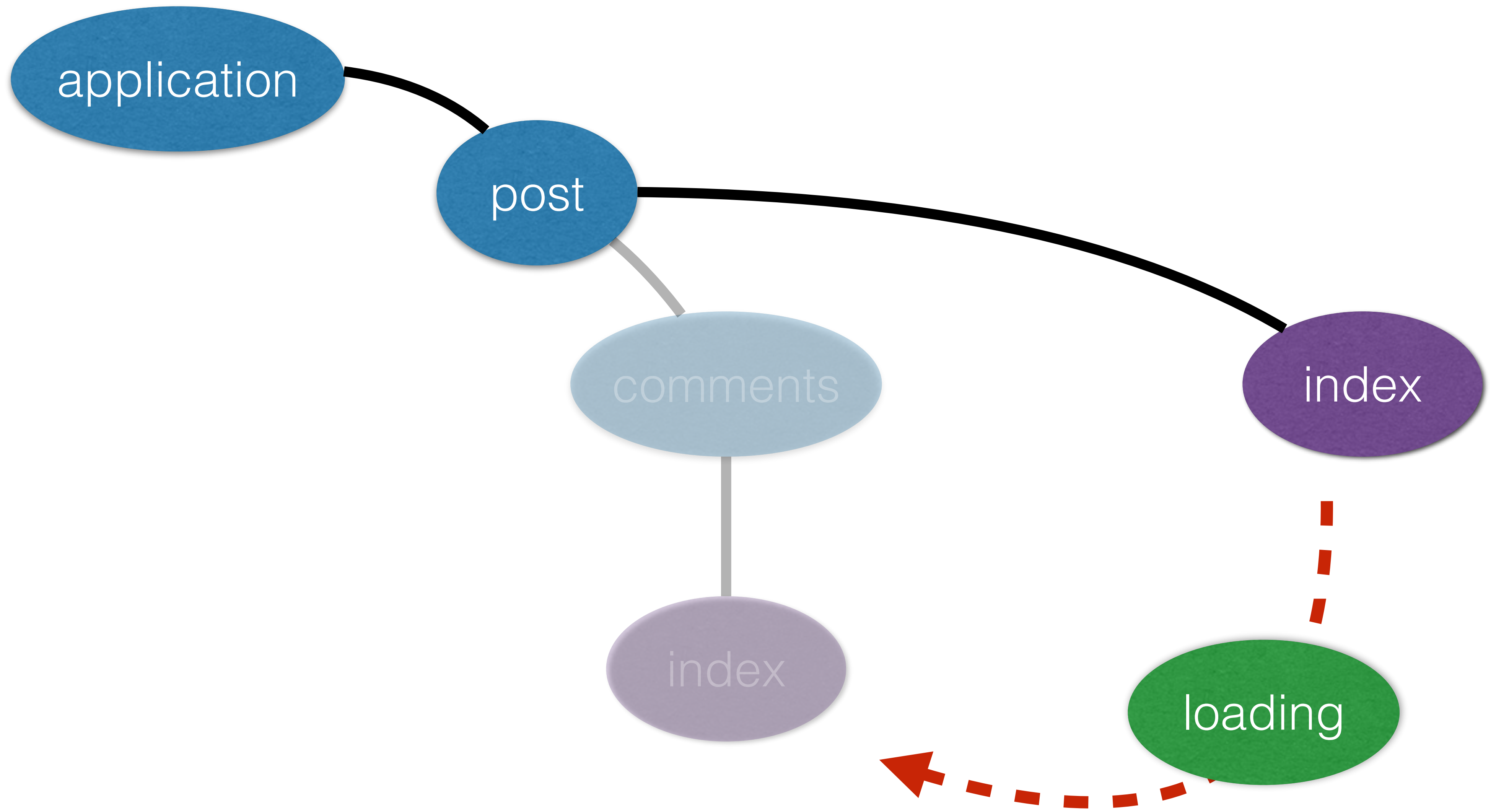
Useful Github API endpoints

https://api.github.com/….	Data
/orgs/:org_id	get information about an org
/orgs/:org_id/repos	get list of repos owned by an org
/repos/:org_id/:repo_id	get information about a particular repo
/repos/:org_id/:repo_id/issues	get list of issues associated with a repo
/repos/:org_id/:repo_id/contributors	get list of contributors associated with a repo

Loading & Error Substates

Loading Substate

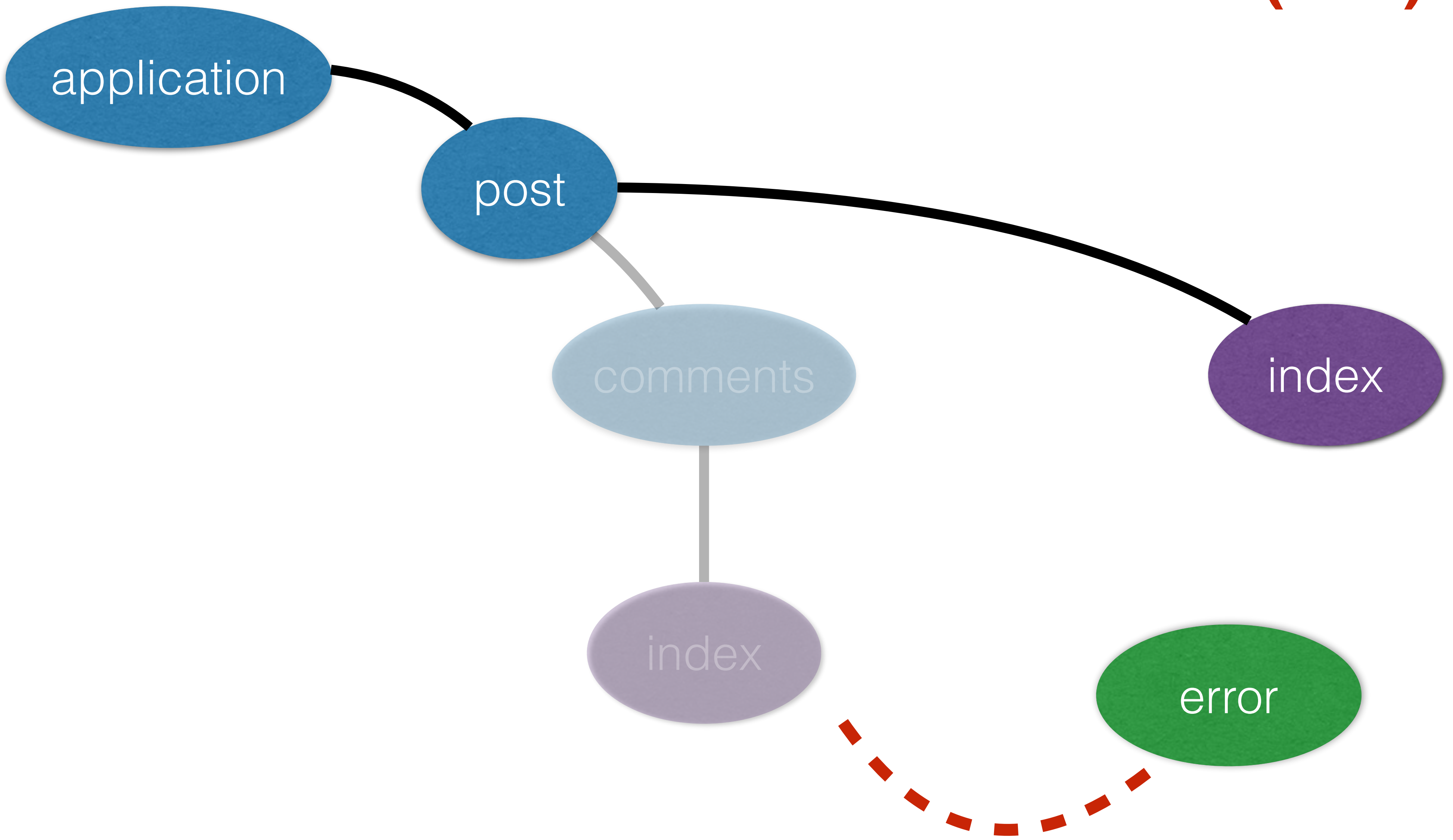
`/post/37` $\longrightarrow$ `/post/37/comments`



Error Substate

`/post/37/comments` ⟶ `/post/9000001231`

(404)



Error Substate

- Also fires the `error` action on the route

```
actions: {
  error(/*jqXhr, transition, route*/) {
    return true;
  }
}
```

```
actions: {
  error(/*jqXhr, transition, route*/) {
    return false; // Don't enter error
substate
  }
}
```

Exercise #5

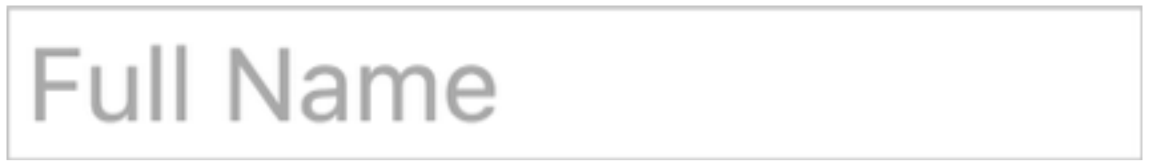
Add loading and error substates to org pages

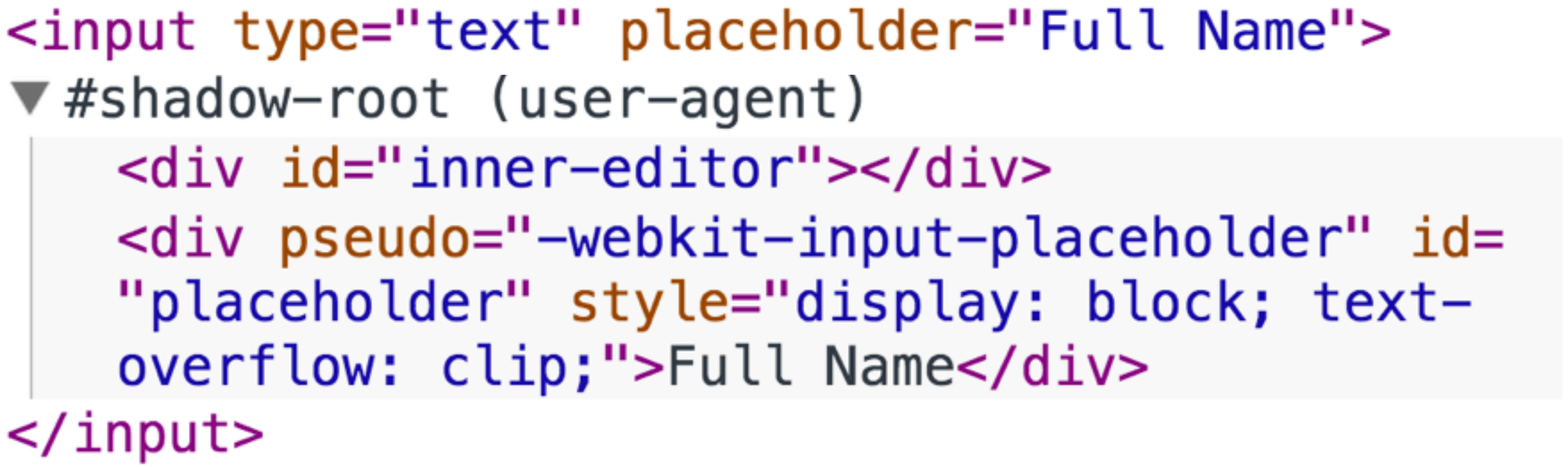
Exercise #5

- When transitioning from the `/orgs` page to `/org/emberjs`, I want a loading spinner while we wait for the API
    - I suggest checking out tobiasahlin.com/spinkit/
- If I attempt to enter the org page for an invalid org, the user should end up on an error page telling the user that something has gone wrong
    - If it's a 404, I want a dedicated 404 page saying "this organization doesn't exist"
    - You'll need to use `this.intermediateTransitionTo(<route name>)` to transition to a custom error page for 404 cases

Components

Components





```
<input type="text" placeholder="Full Name">
▼ #shadow-root (user-agent)
    <div id="inner-editor"></div>
    <div pseudo="-webkit-input-placeholder" id=
    "placeholder" style="display: block; text-
    overflow: clip;">Full Name</div>
</input>
```

Components

- Live in the `app/components folder`

- Associated with a top-level DOM element

- Strong encapsulation, and explicit contract with the outside world

- Well-orchestrated life cycle

Components

Life Cycle Hooks

`init`	on instantiation
`willInsertElement`	before the component's element is inserted into the DOM
`didInsertElement`	after the component's element has been inserted into the DOM
`willDestroyElement`	before the component's element is removed from the DOM

Clean up before tear down

`$(document).ready()` for components

Components

Consuming

- Components register their own handlebars helper automatically `{{my-component}}`
- They can be used with `{{inline-syntax}}` or `{{#block-syntax}}  {{/block-syntax}}`
- All components in the application namespace are available in any template, throughout the application
  - This means any ES6 module named `github-ui/components/*.js`

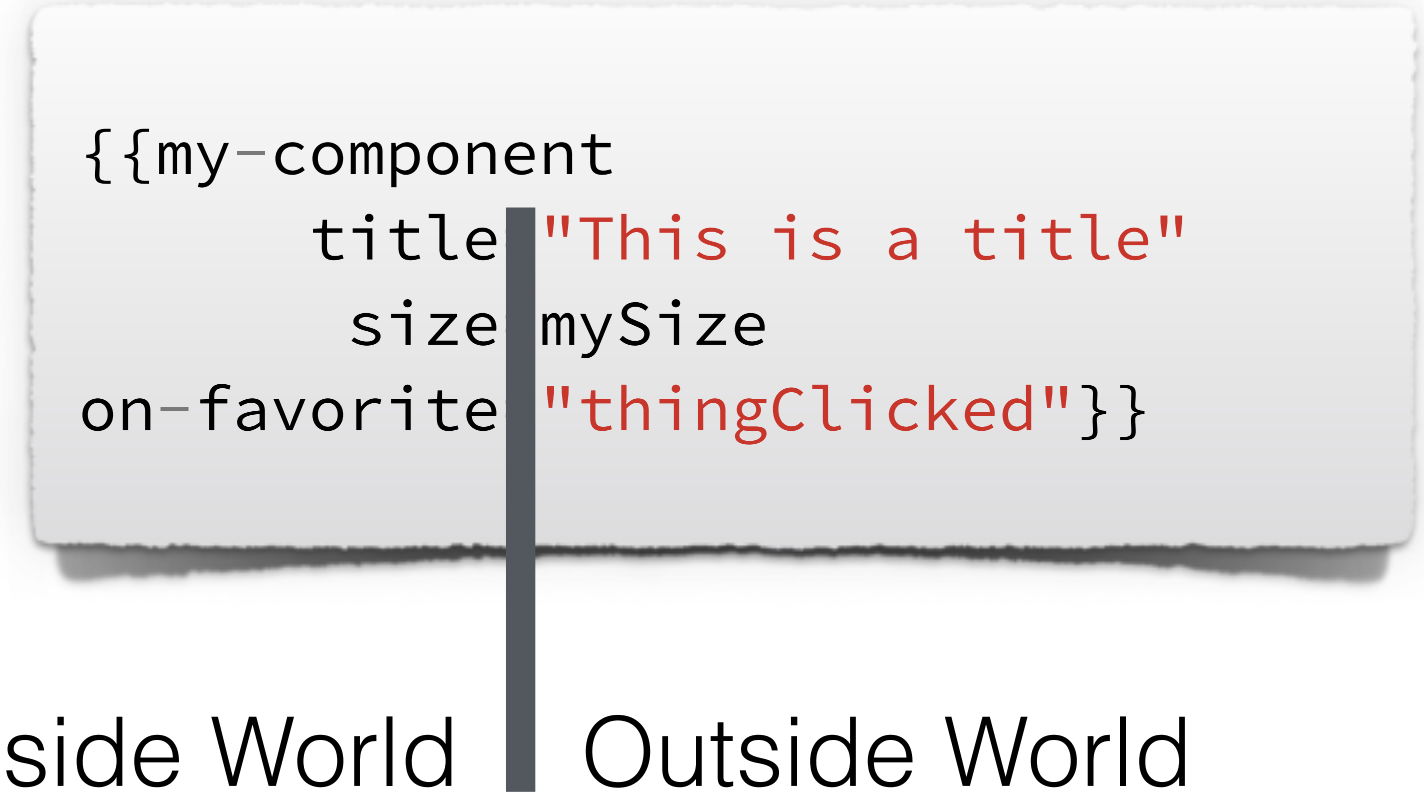
Components

An Example

```
{{my-component
      title="This is a title"
       size=mySize
on-favorite="thingClicked"}}
```

Components

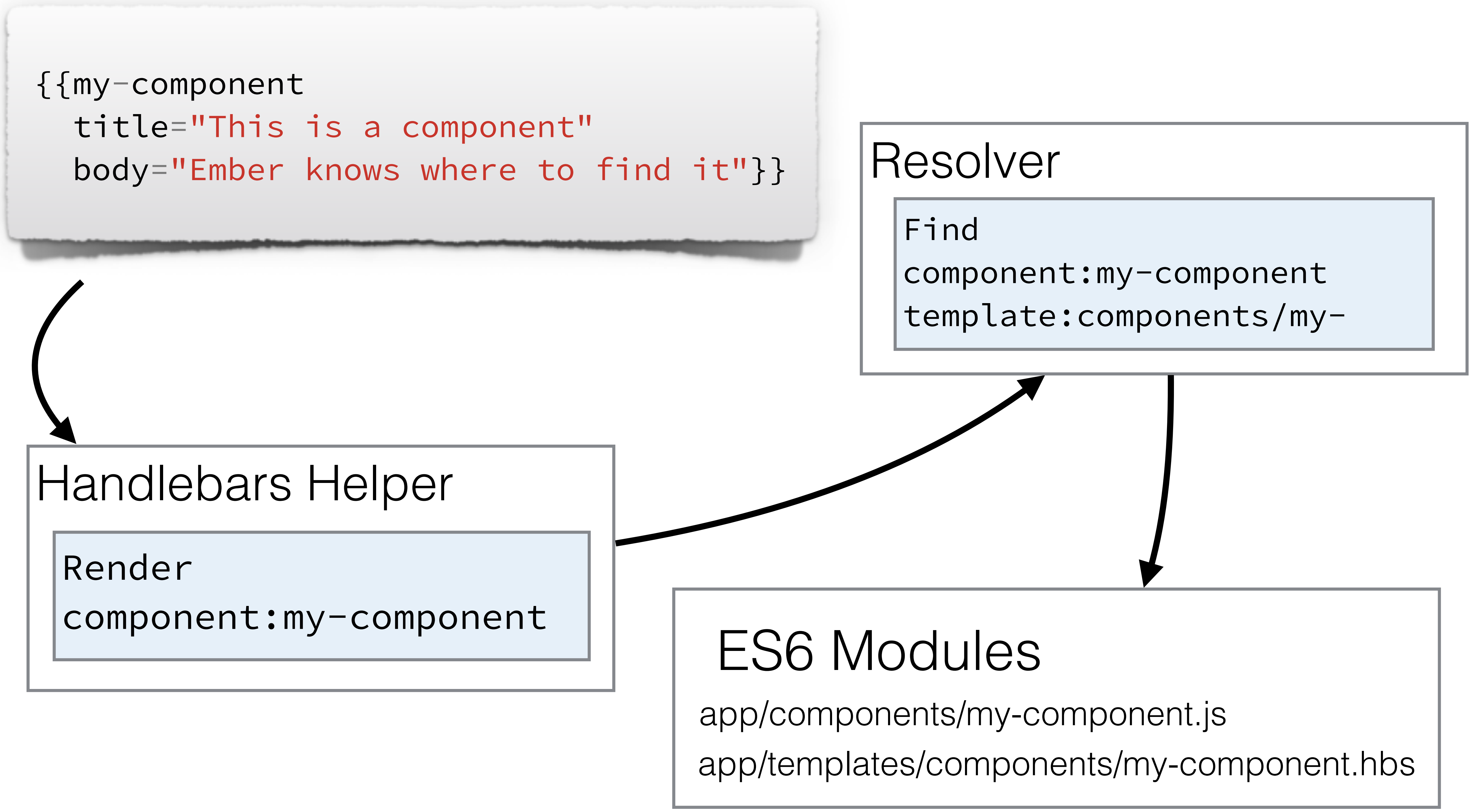
Consuming



```
{{my-component
      title "This is a title"
       size mySize
on-favorite "thingClicked"}}
```

Components

Naming Convention



```
{{my-component
  title="This is a component"
  body="Ember knows where to find it"}}
```

Component

```
import Ember from 'ember';

export default Ember.Component.extend({
  // Change root element from <div> to <li>
  tagName: 'li',

  // Add classes to root element
  classNames: ['pull-right'],

  // Bind two of this components properties
  //    to DOM element attributes on the root
  //    element
  attributeBindings: ['age:data-age', 'align'],
  age: 32,
  align: 'left'
});
```

Consumption in Template

```
{{my-component age=39}}
{{my-component}}
```

Resulting HTML

```
<li id="ember404"
    data-age="39" align="left"
    class="ember-view pull-right"></li>

<li id="ember405"
    data-age="32" align="left"
    class="ember-view pull-right"></li>
```

Component Examples

Life Cycle Hooks

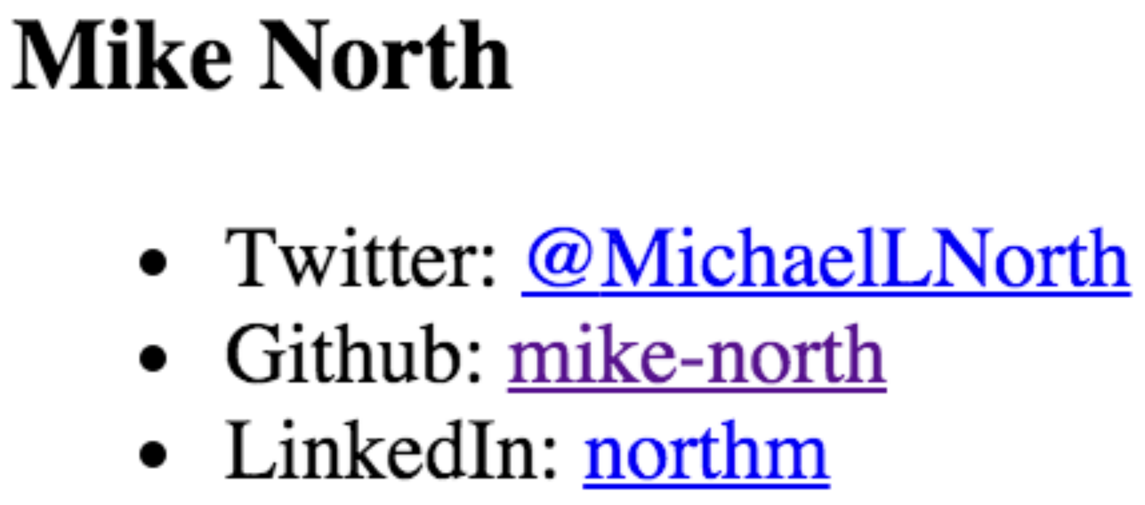
- Time for some live coding
  - Social media info
  - Collapsible card (title/content)

Using pure HTML

```
<div class="social-info">
  <h3>Mike North</h3>
  <ul>
    <li>
      Twitter:
      <a href="https://twitter.com/
michaellnorth">
        @MichaelLNorth
      </a>
    </li>
    <li>
      Github:
      <a href="https://github.com/
mike-north">
        mike-north
      </a>
    </li>
    <li>
      LinkedIn:
      <a href="https://
www.linkedin.com/in/northm">
        northm
      </a>
    </li>
  </ul>
</div>
```

Using a component

```
{{social-info
  name="Mike North"
  twitter="MichaelLNorth"
  linkedin="northm"
  github="mike-north"}}
```



Mike North

- Twitter: @MichaelLNorth
- Github: mike-north
- LinkedIn: northm

Exercise #6

Componentizing orgs and repos

Exercise #6

- Replace the body of the `{{#each}}` on the `/orgs` page with a `{{github-org}}` component
  - Make sure the “favorite” behavior continues to work
- Replace the body of the `{{#each}}` on the `/org/:id/`repos page with a `{{github-repo}}` component
  - Add fork count and watcher count to each item in the repo listing

Computed Properties

Computed Properties

- Properties that are calculated, based on other properties
- Evaluated lazily
- Cached
- Still can get/set

Computed Properties

read only

```
const { computed } = Ember;

export default Ember.Component.extend({
  fullName: computed(‘firstName', 'lastName',function() {
    return [this.get('firstName'),
            this.get('lastName')].join(' ');
  })
});
```

Computed Properties

read only (alternate syntax)

```
const { computed } = Ember;

export default Ember.Component.extend({
  fullName: computed('firstName', 'lastName', {
    get() {
      return [this.get('firstName'),
              this.get('lastName')].join(' ');
    }
  })
});
```

Computed Properties

settable

```
const { computed } = Ember;

export default Ember.Component.extend({
  fullName: computed('firstName', 'lastName', {
    get() {
      return [this.get('firstName'),
              this.get('lastName')].join(' ');
    },
    set(key, newVal) {
      const nameParts = newVal.split(' ');
      this.set('firstName', nameParts[0]);
      this.set('lastName', nameParts[1]);
      return newVal;
    }
  })
});
```

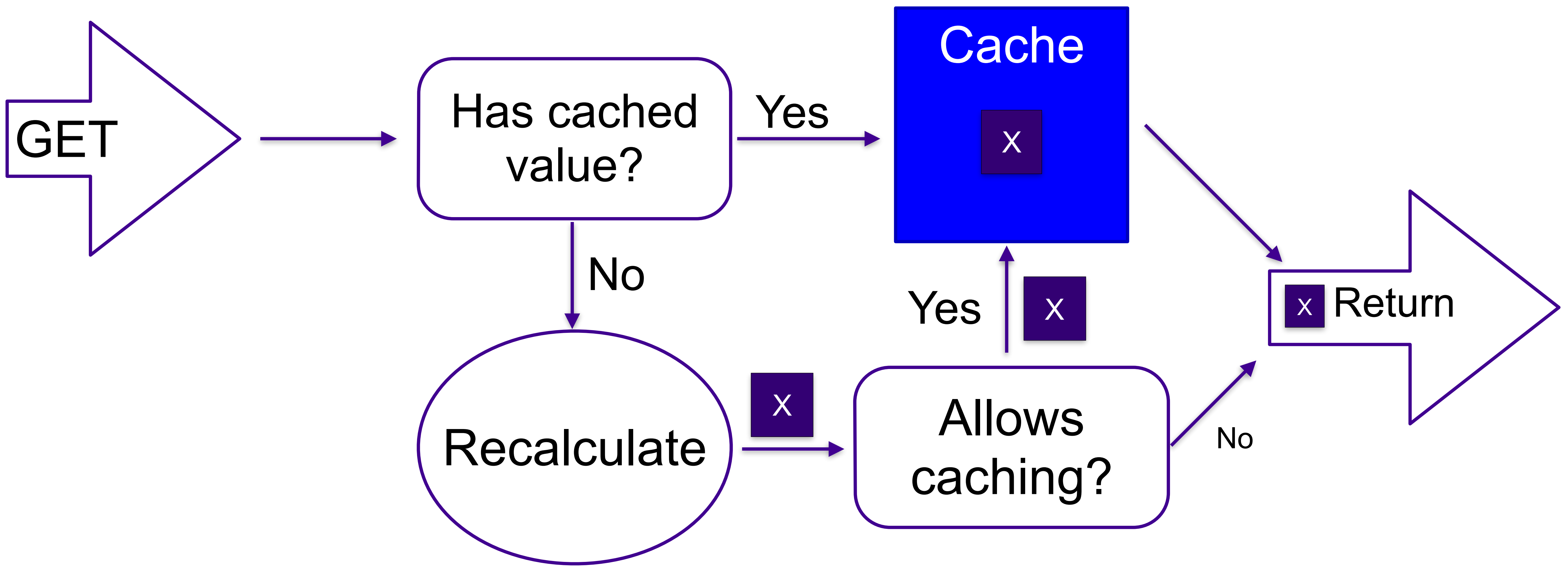
Computed Properties

Dependencies on Arrays

- Syntax for CP dependencies on array properties can be done two `ways`
    - `"myList.[]"` defines a dependency on the length of the array
    - `"myList.@each.id"` defines a dependency the `id` property of each item in the array

Computed Properties

Internal Mechanism



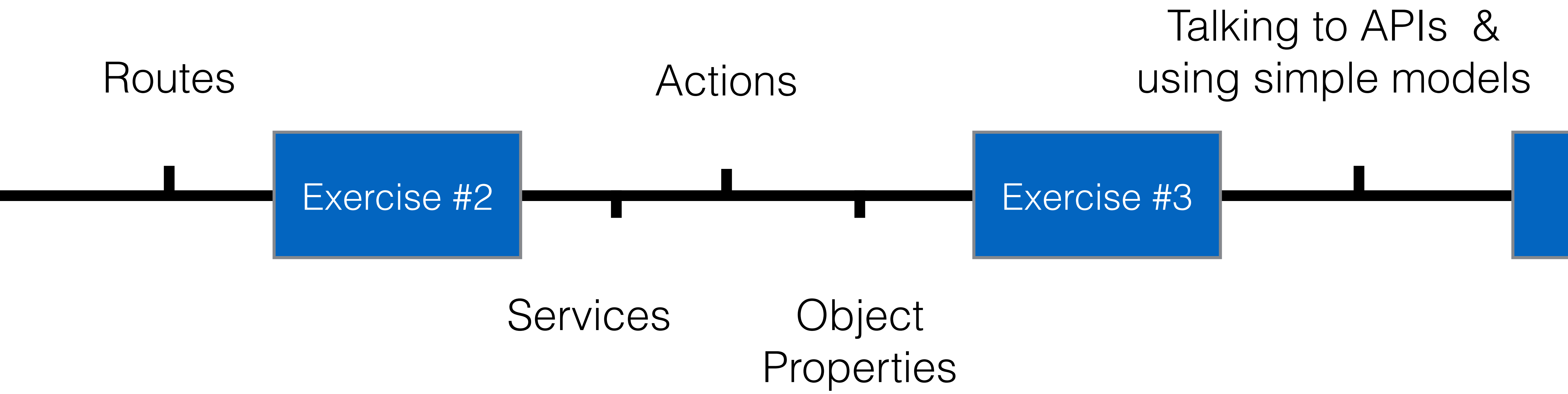
Exercise #7

Componentizing orgs and repos

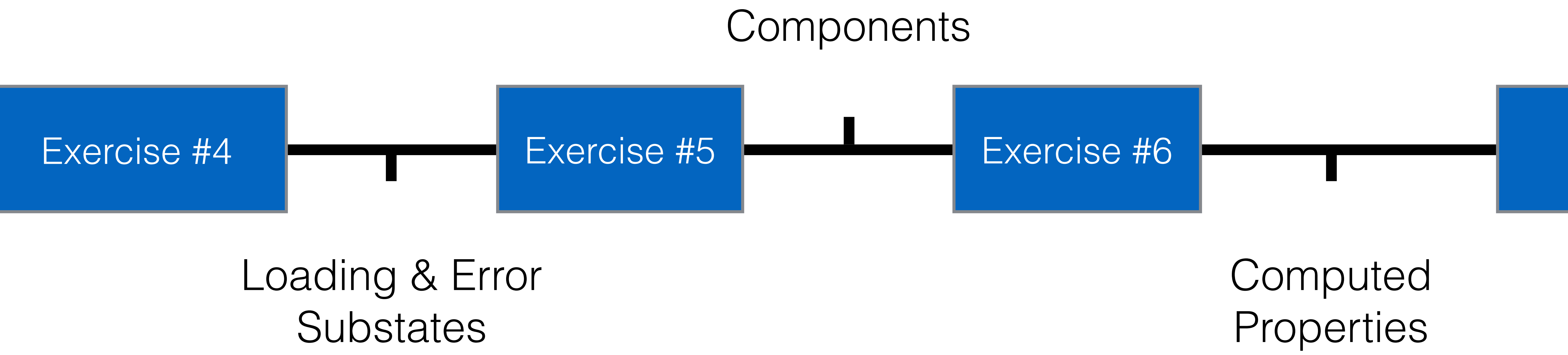
Exercise #7

- Enhance the `{{github-org}}` by adding a computed property for the "favorite" state
  - You will have to inject the `favorites` service onto the component
  - Declaring a dependency on items in an array `'favorites.items.[]'` or `'favorites.items.@each.id'`

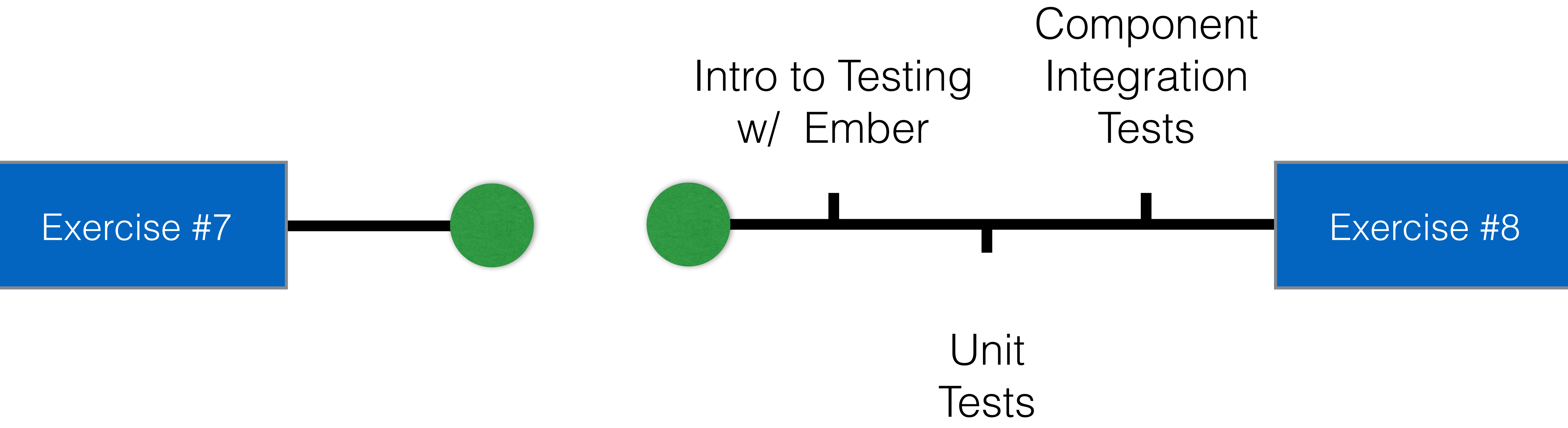
Agenda


Routes
Exercise #2
Services
Actions
Object Properties
Exercise #3
Talking to APIs & using simple models

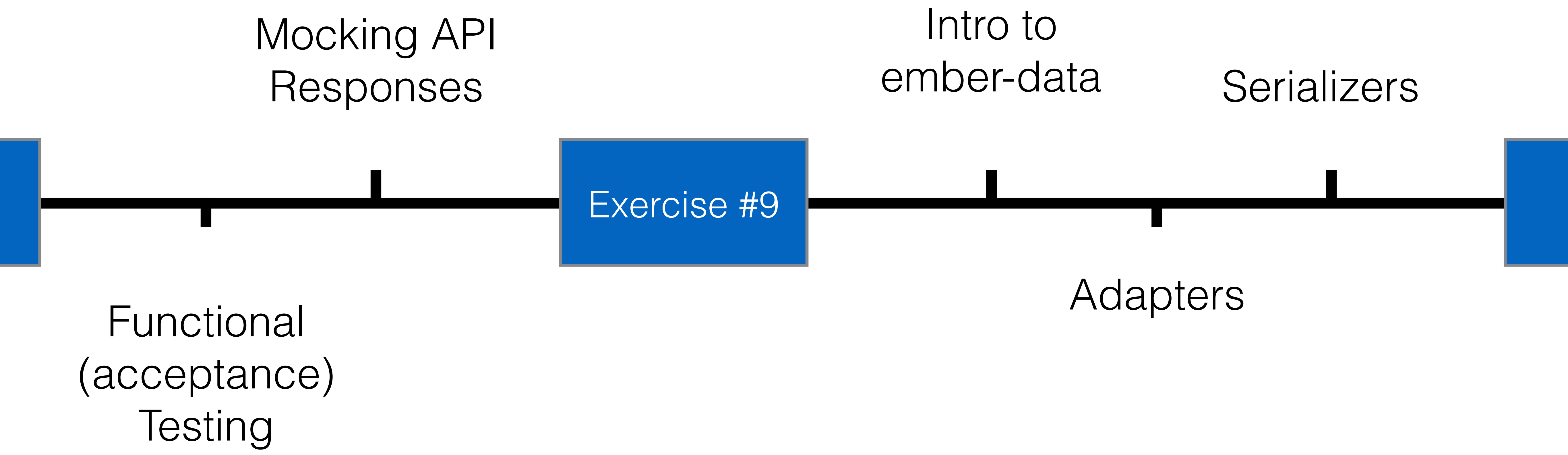
Agenda


Components
Exercise #4
Exercise #5
Exercise #6
Loading & Error Substates
Computed Properties

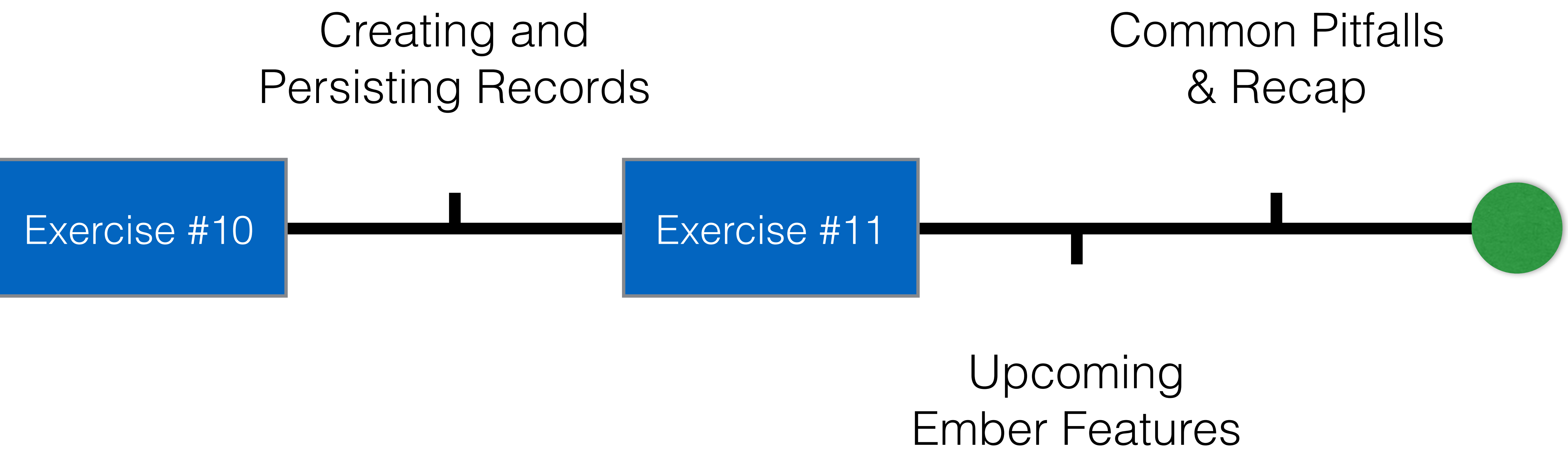
Agenda


Exercise #7
Intro to Testing w/ Ember
Component Integration Tests
Unit Tests
Exercise #8

Agenda


Mocking API Responses
Intro to ember-data
Serializers
Exercise #9
Functional (acceptance) Testing
Adapters

Agenda


Creating and Persisting Records
Common Pitfalls & Recap
Exercise #10
Exercise #11
Upcoming Ember Features

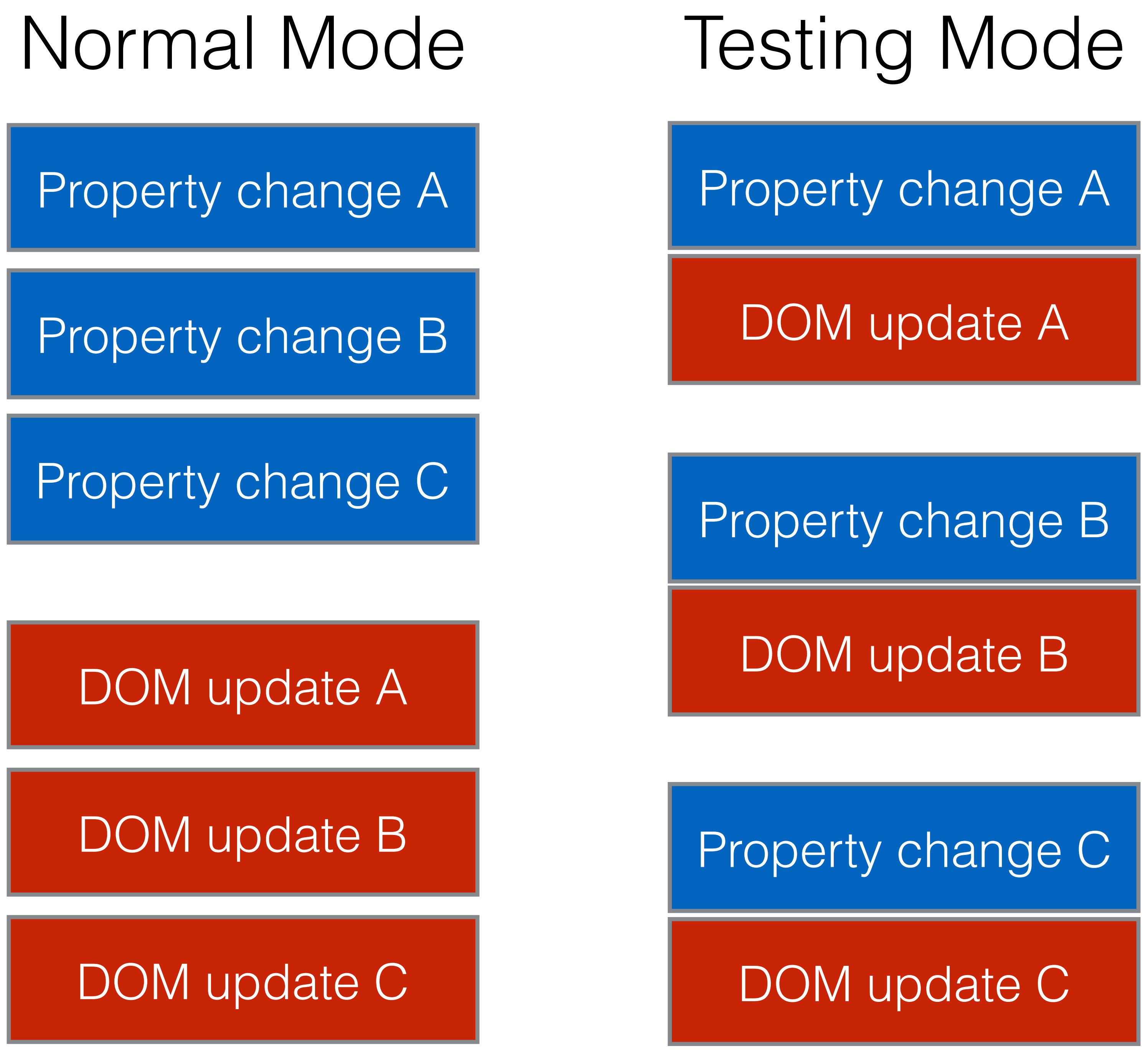
A few flavors of tests

- Unit - for testing algorithmic complexity
- Integration - for testing contracts between pieces of code, and how things work together
- Acceptance (functional) - testing user workflows in the context of your entire application

The flavor I reach for

utils	Unit
models	Unit
services	Unit
components	Integration
routes	Acceptance
ember-data stuff	Acceptance

Running in testing mode



Running tests

- http://localhost:4200/tests
- `ember test ci`
- `ember test —server`

ember