Ember.js In Depth

frontendmasters.com

@MichaelLNorth

github.com/mike-north

About.me

- Ember.js, Ember-cli, Ember-data contributor
- `Job.new` = CTO 

- `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

Actions

Talking to APIs &
using simple models

Exercise #2

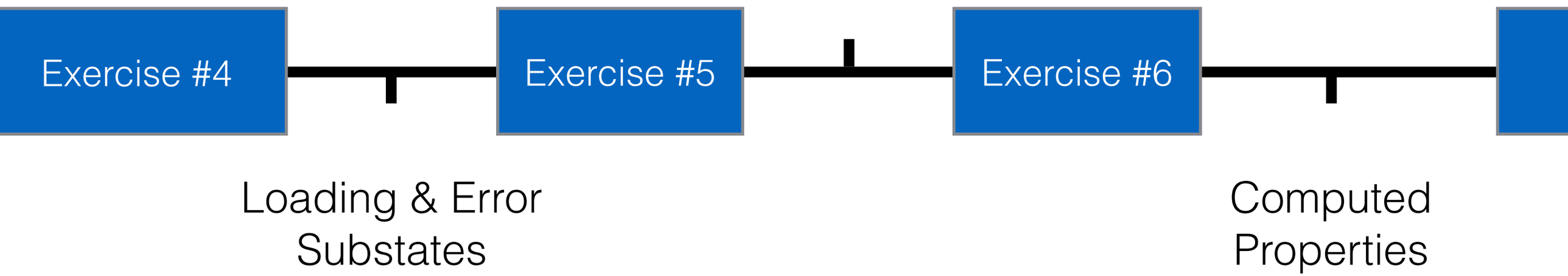
Exercise #3

Services

Object
Properties

Agenda

Components

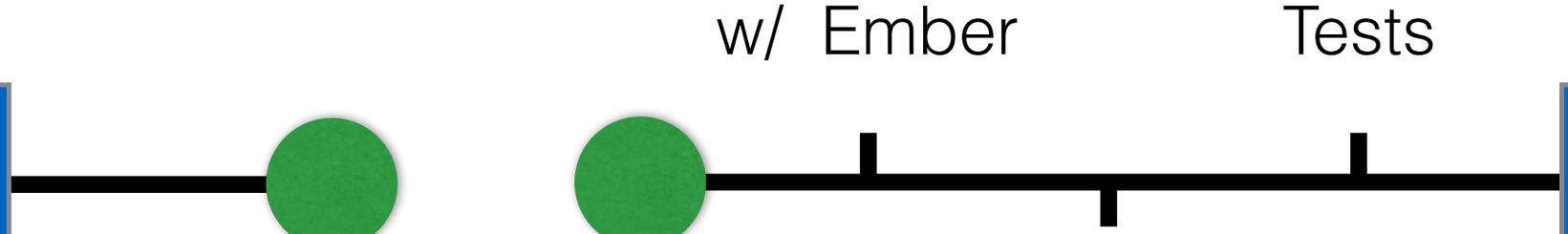


Agenda

Intro to Testing w/ Ember

Component Integration Tests

Exercise #7



Exercise #8

Unit Tests

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

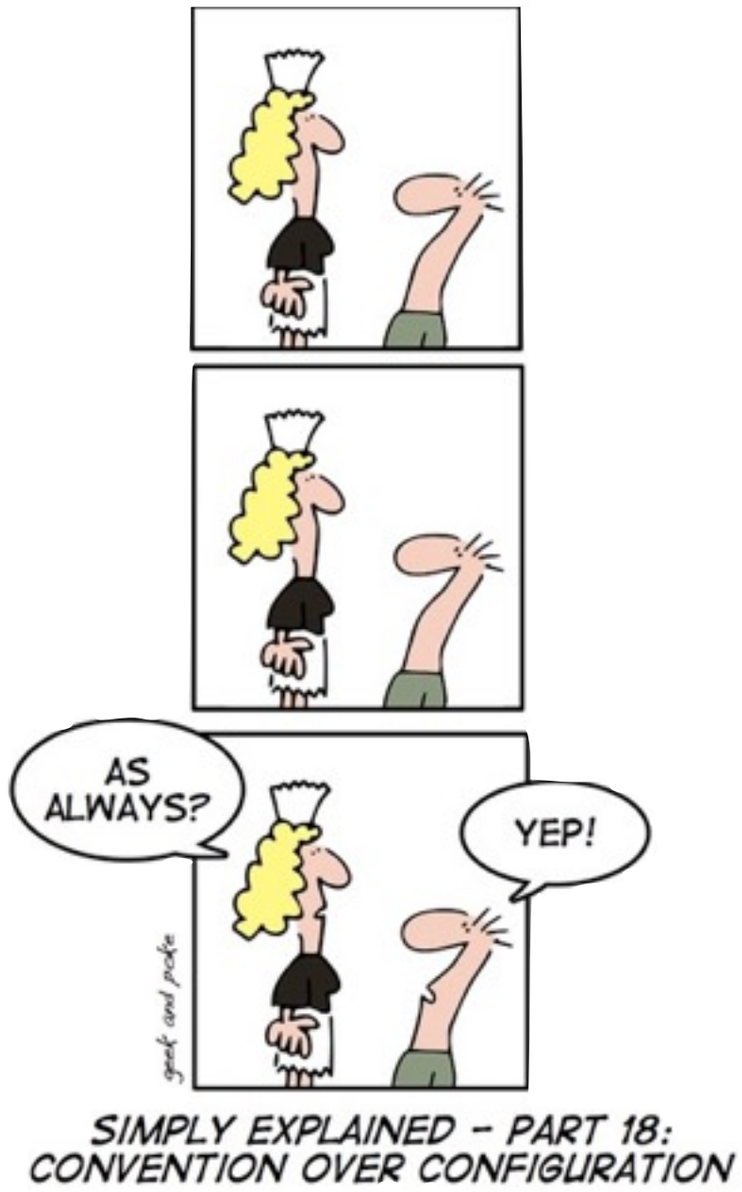
Exercise #11

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>
```

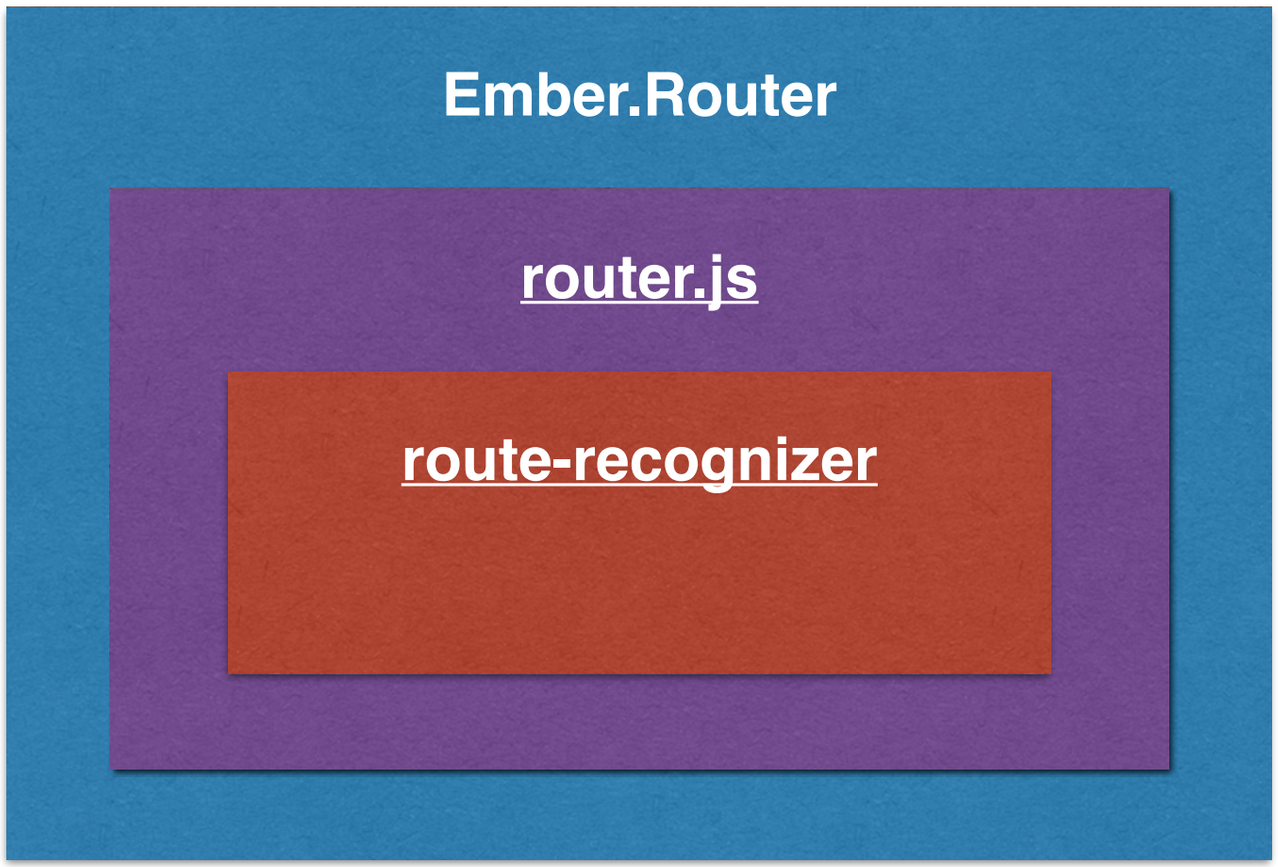
Router

Ember.Router

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

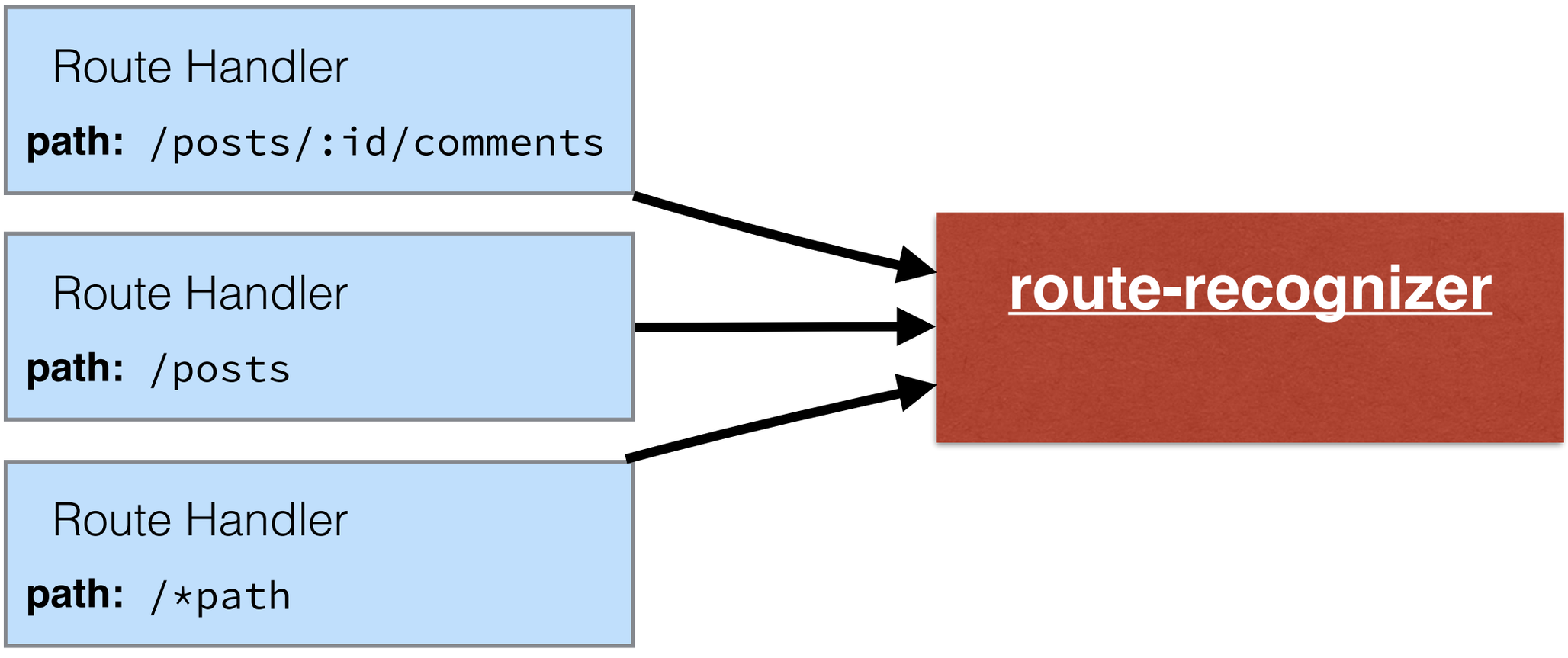
Ember.Router

Microlibraries



Ember.Router

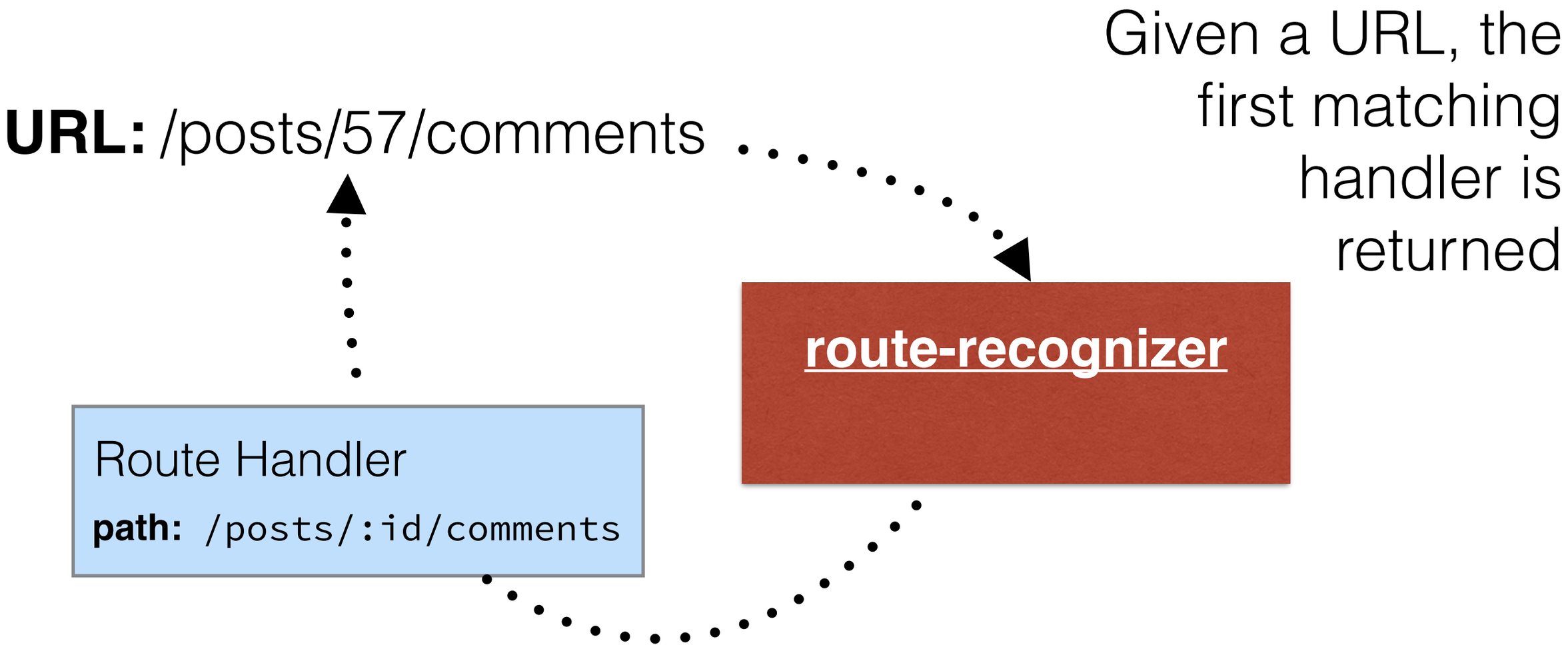
Microlibraries - route-recognizer



Route handlers are registered, each with a path

Ember.Router

Microlibraries - route-recognizer



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

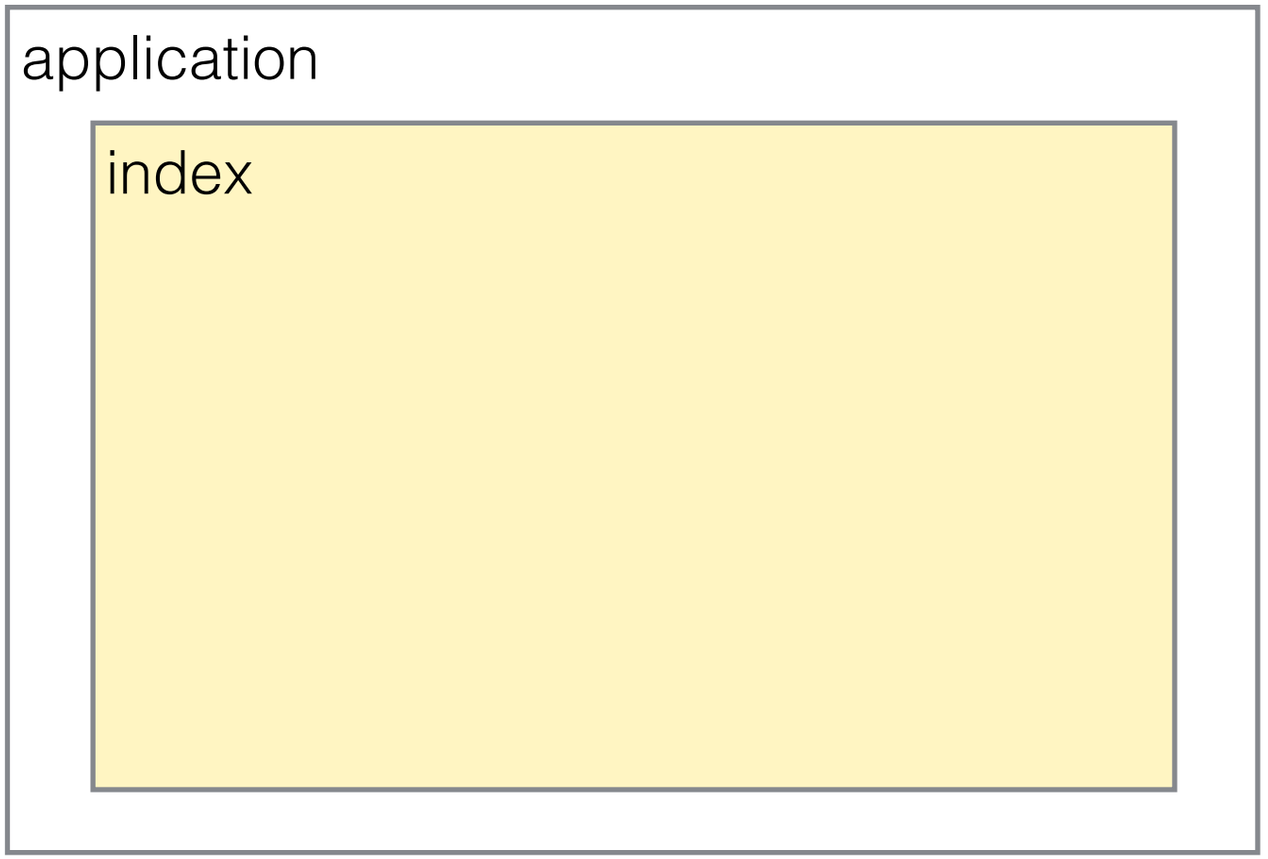
```
urlWatcher.onUpdate(funct
ion(url) {
  router.handleURL(url);
});
```

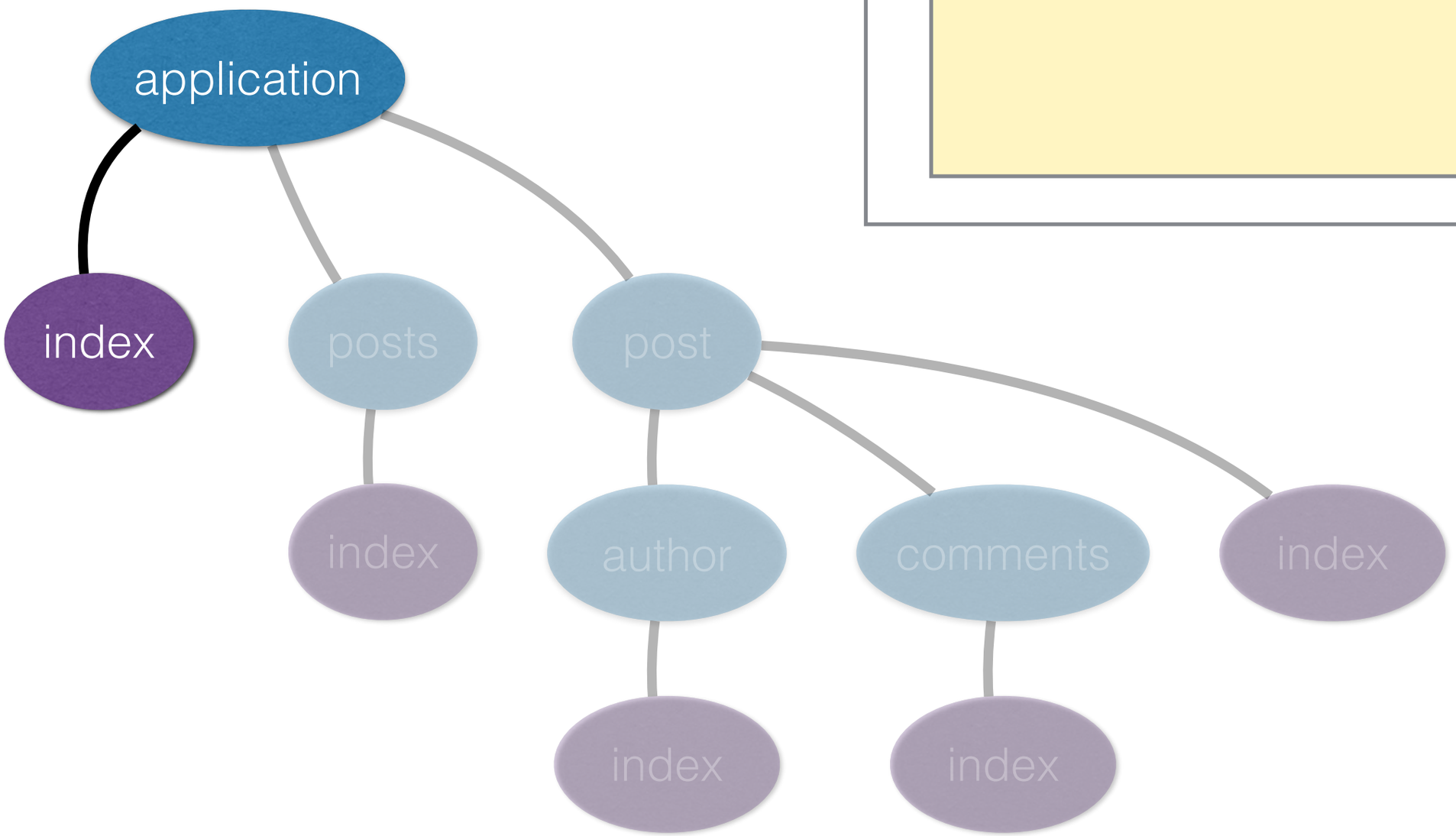
Ember.Router

Router - Routes as a Hierarchy

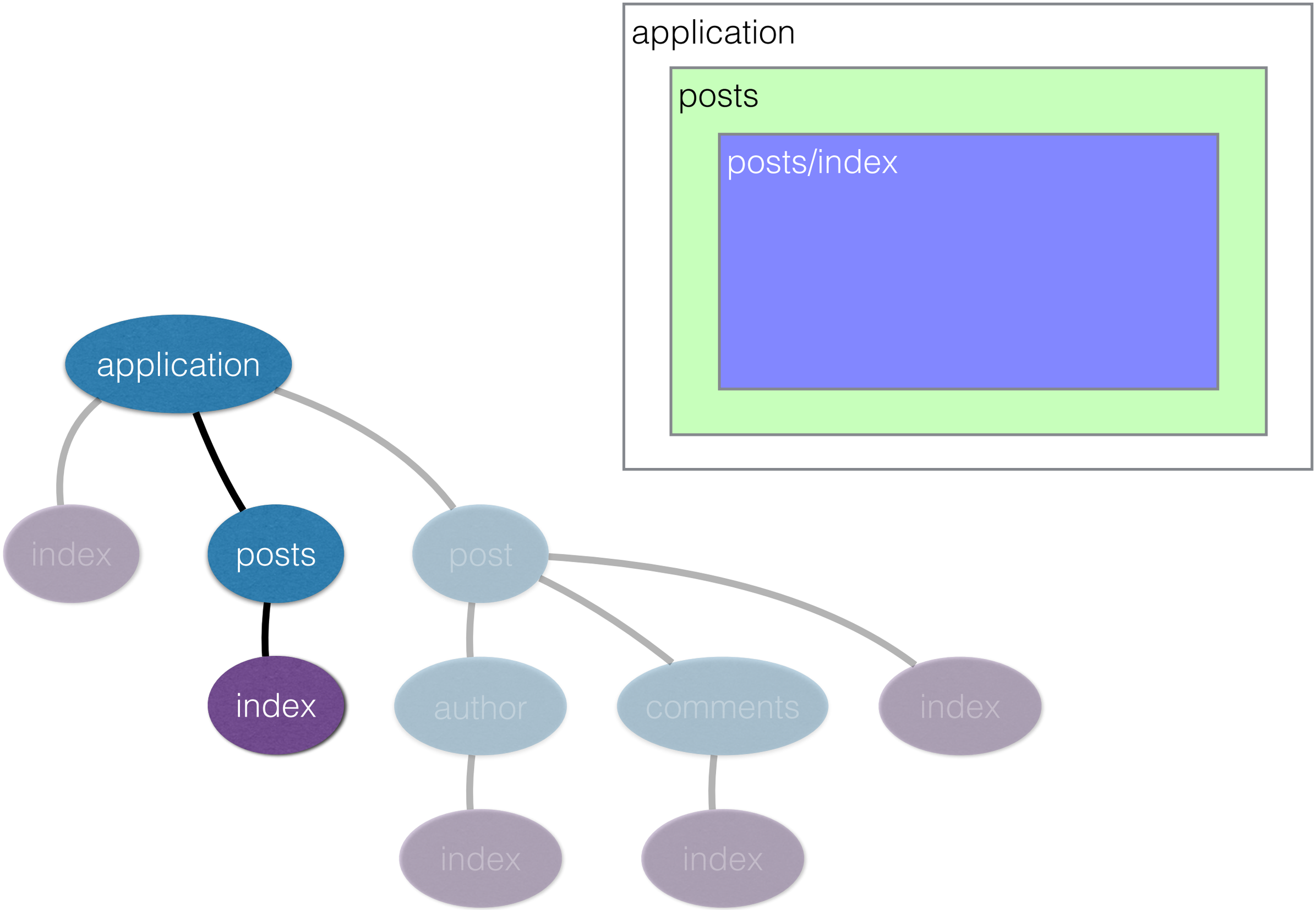
```
Router.map(function() {
   // IMPLIED index                 /
  this.route('posts'); //           /posts
   this.route('post', {path: 'post/:id'}, function() {
     // IMPLIED index                /post/123
     this.route('author'); //   /post/123/author
     this.route('comments');// /post/123/comments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

URL: /

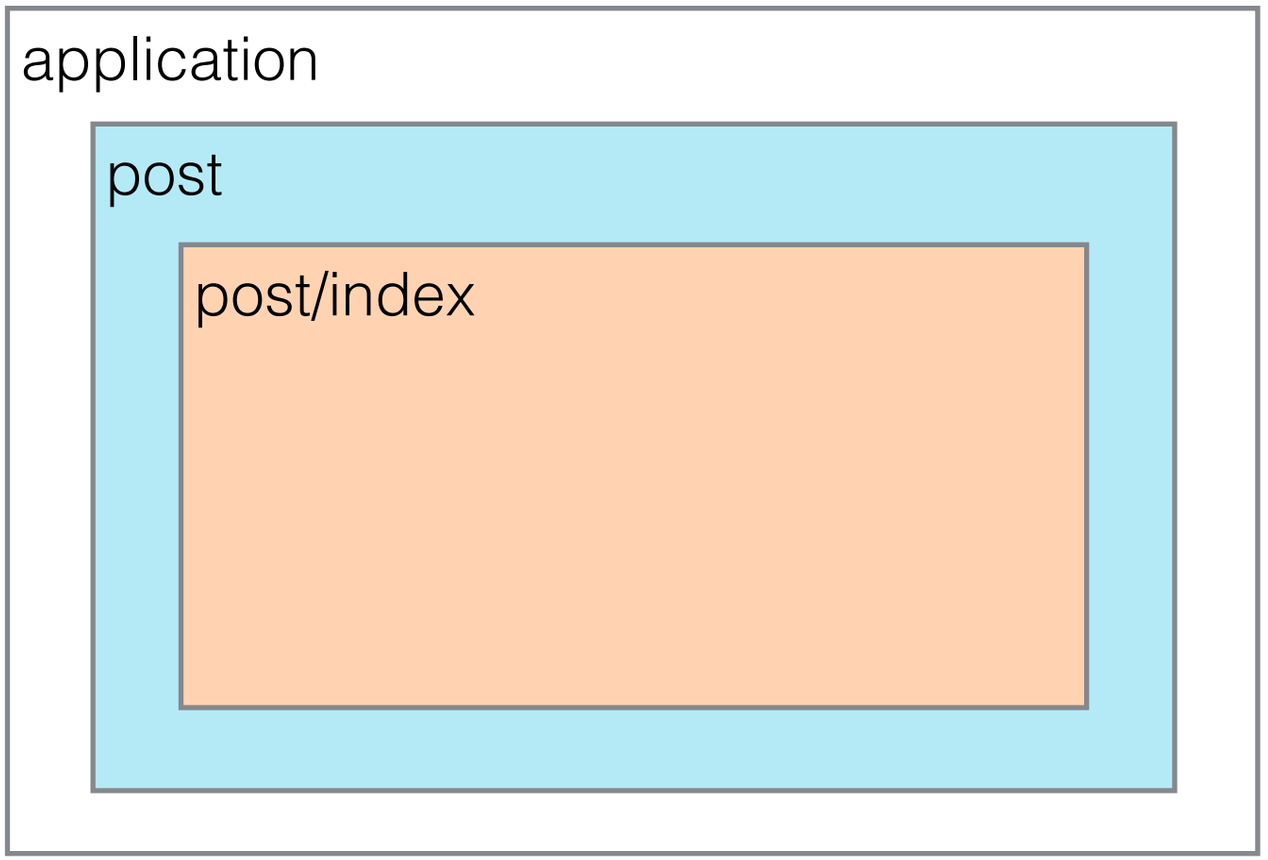

application
index

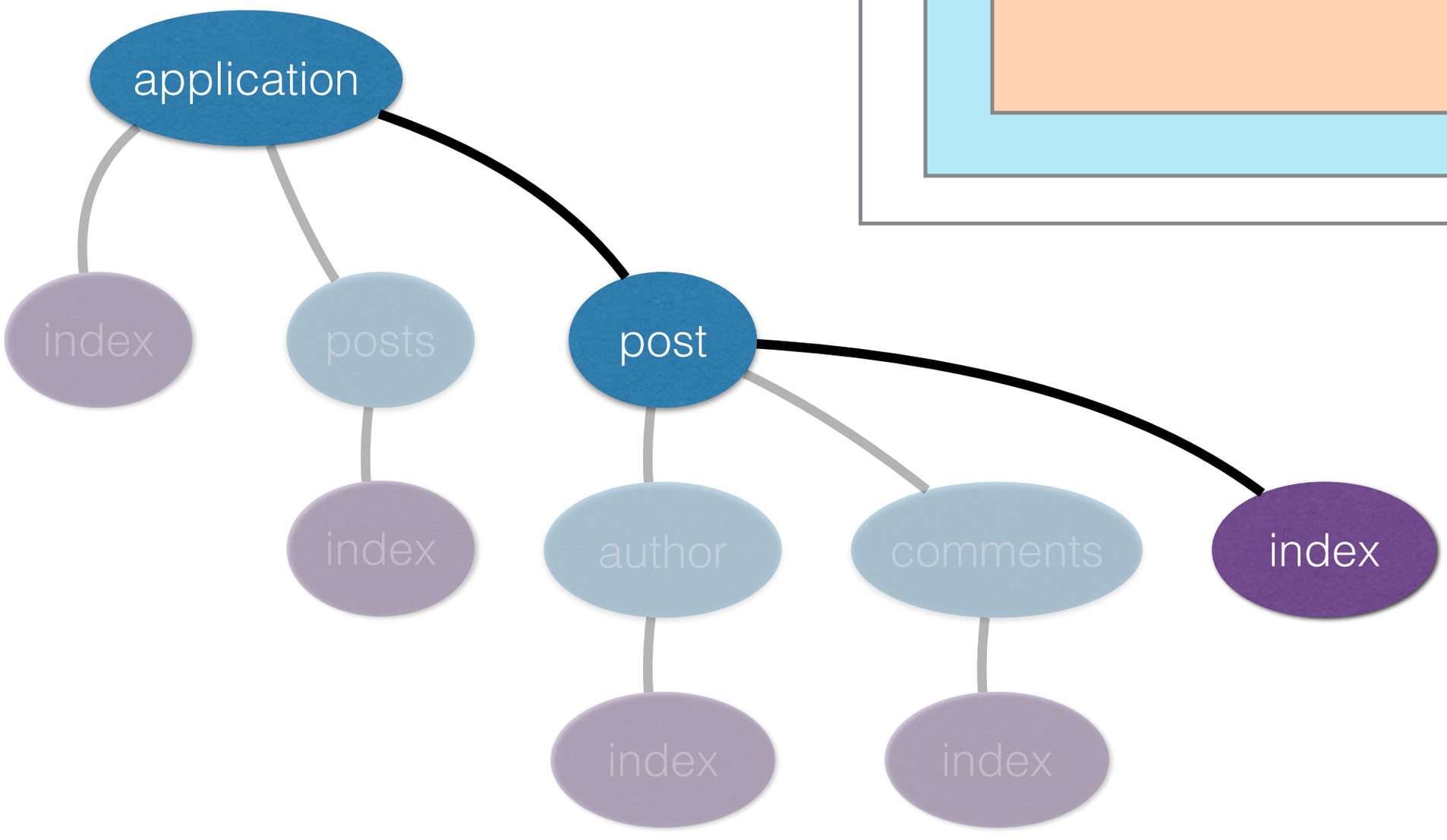

application
index
posts
post
index
author
comments
index
index
index

URL: `/posts/`

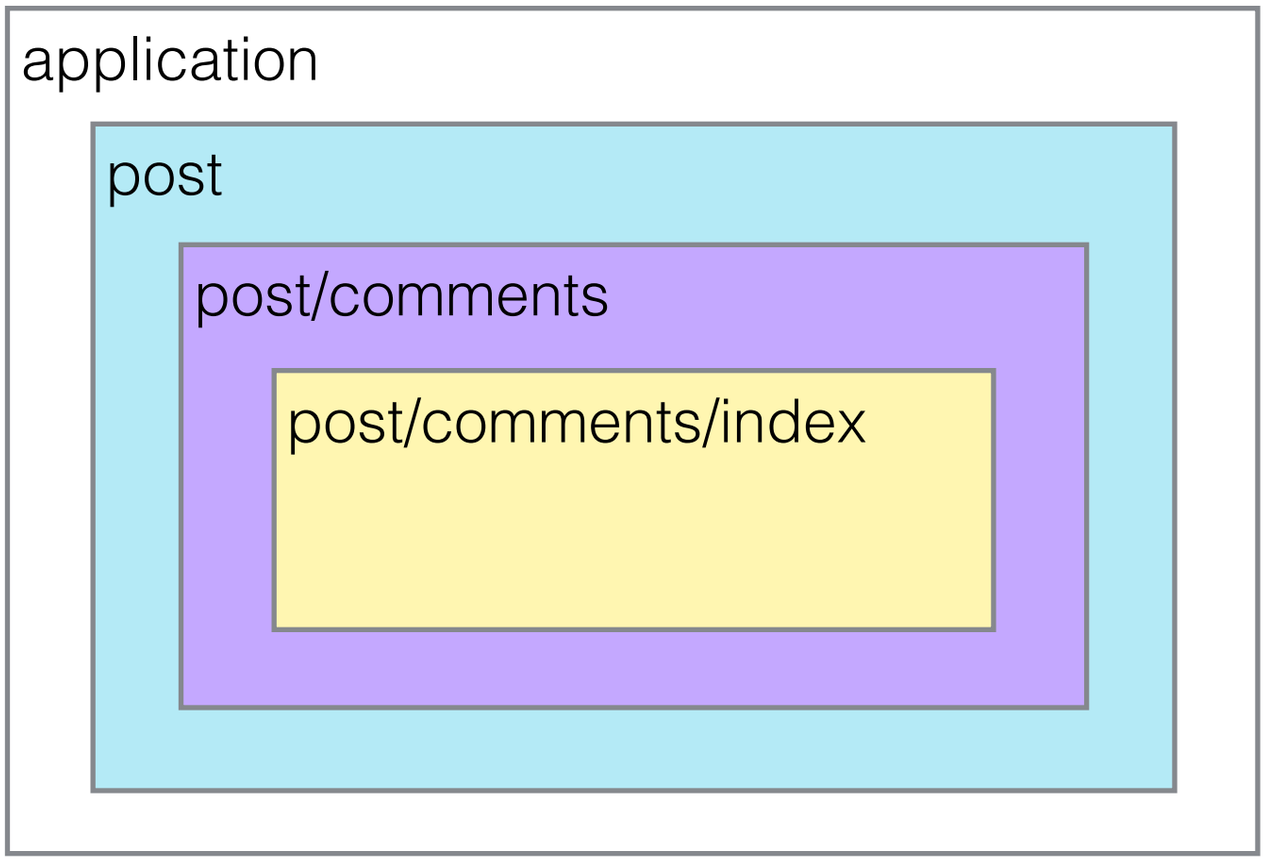

application
posts
posts/index
application
index
posts
index
post
author
index
comments
index
index

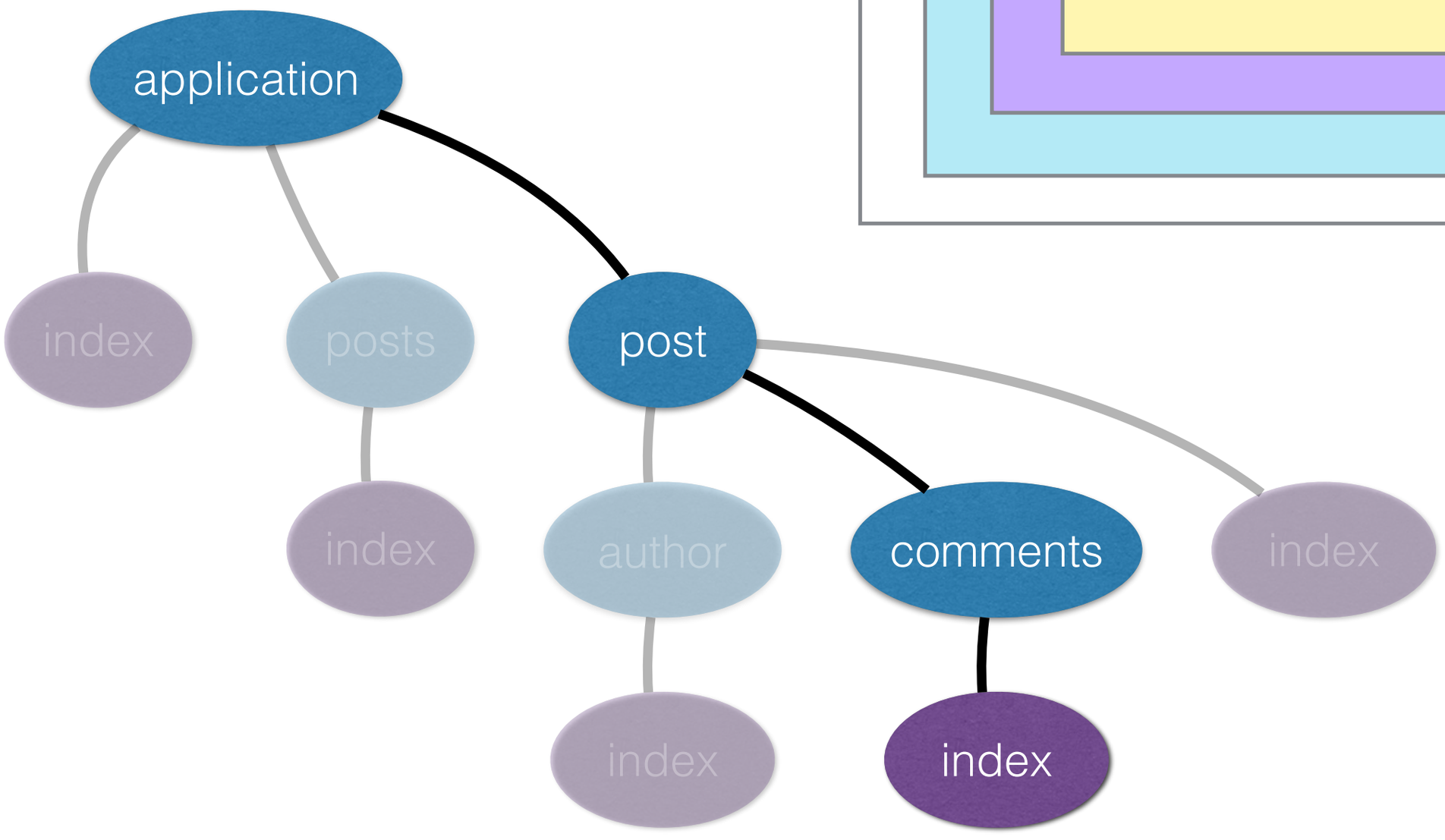
URL: `/post/31`


application
post
post/index

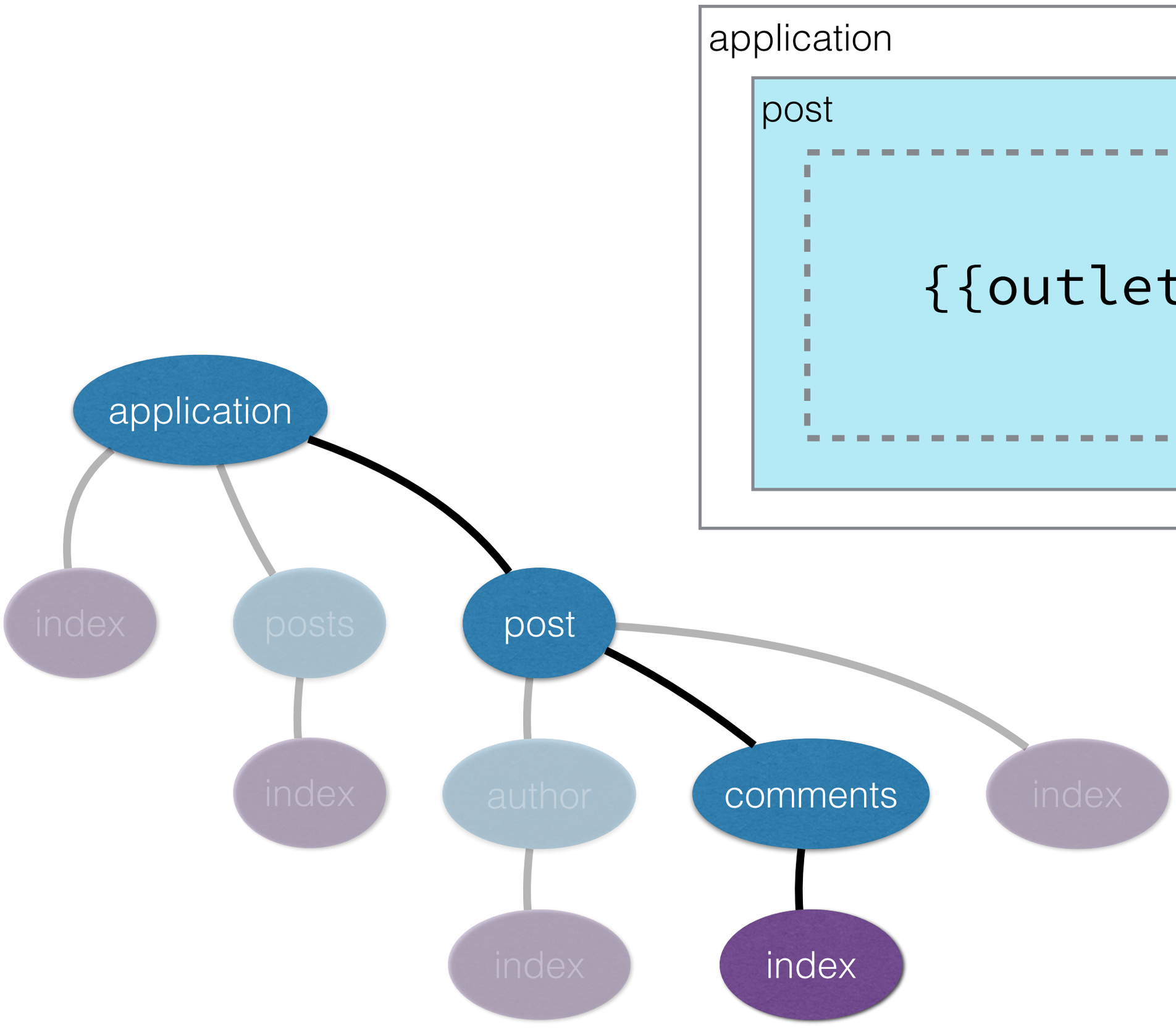

application
index
posts
post
index
author
comments
index
index
index

URL: `/post/31/comments/`





URL: `/post/31/comments/`



Ember.Router

link-to - for internal links to routes

Text

Route Name

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

Text

Route Name

Model or ID

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 “ Homepage ”
/orgs	List of some github orgs
/org/emberjs	Placeholder text “ Org: Emberjs ”
/org/emberjs/repos	List of repos in the ember's org
/org/emberjs/ember.js	Placeholder text “ Repo: Ember.js ”
/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

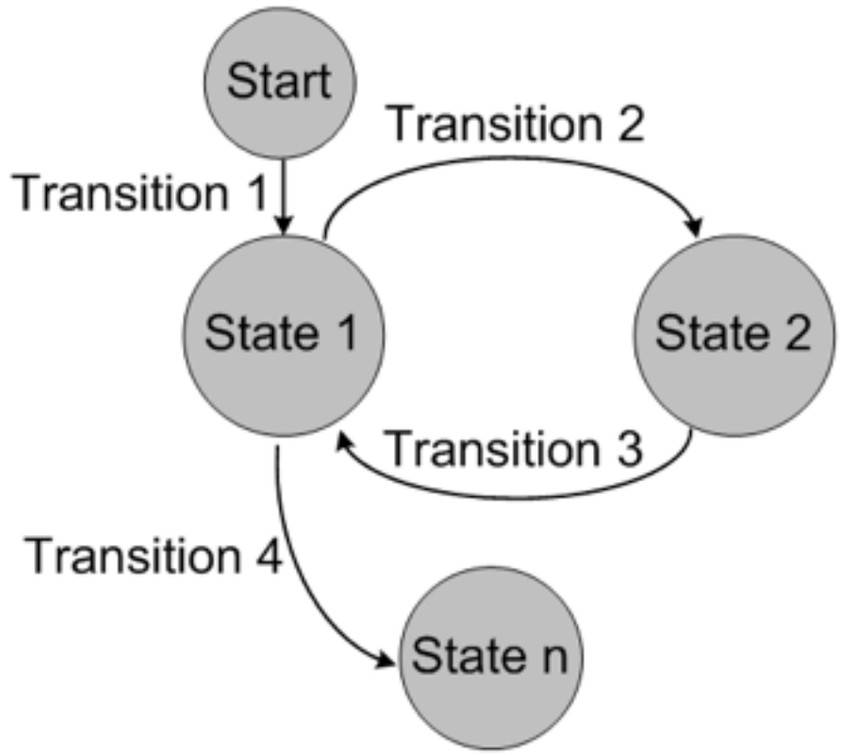
Exercise #1

- 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/issues`
/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 #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>
```

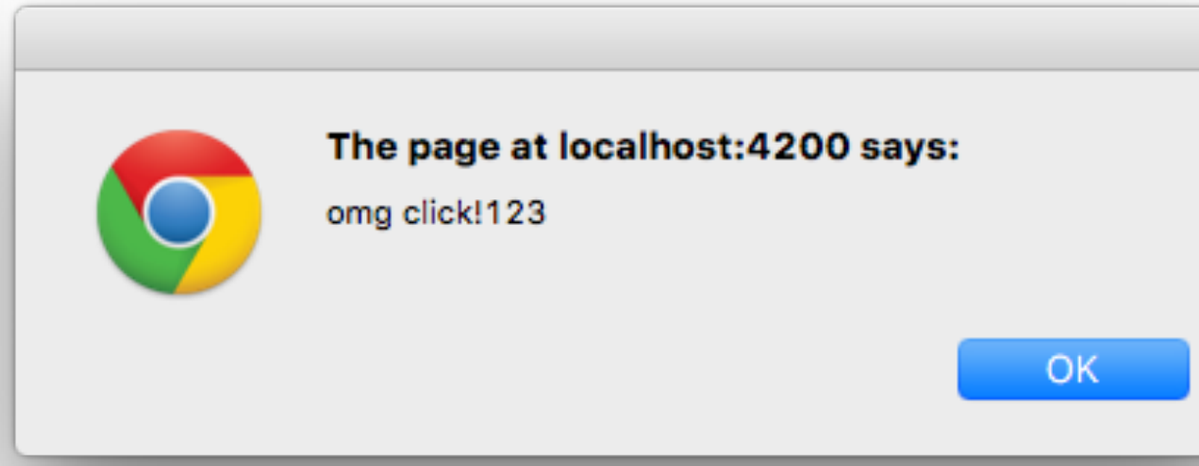
```
Ember.Route.extend({
  actions: {
    thingWasClicked() {
      alert('omg click!');
    }
  }
});
```

Actions

- You can send data along too

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

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



`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

Welcome Back!

Agenda

Philosophy of Ember

Goals of this workshop

Anatomy of an Ember app

Template Basics

Router

Exercise #1

Agenda

Routes

Actions

Talking to APIs &
using simple models

Exercise #2

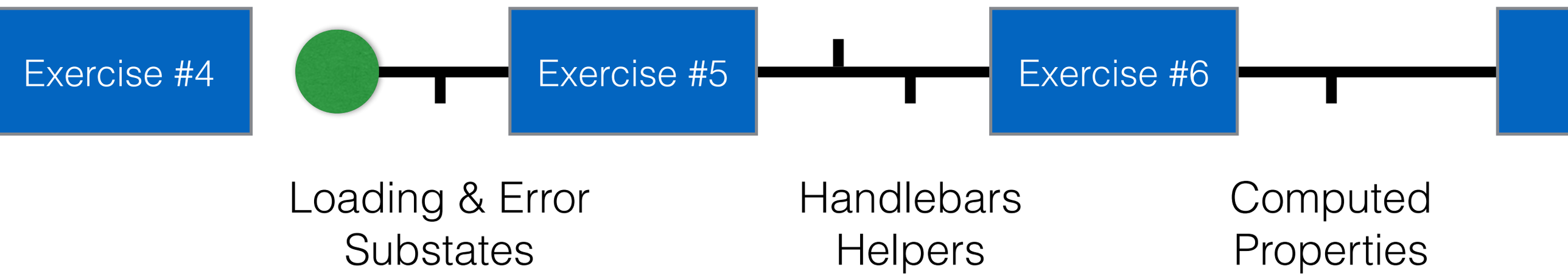
Exercise #3

Services

Object
Properties

Agenda

Components



Agenda

Intro to Testing
w/ Ember

Component
Integration
Tests

Exercise #7

Exercise #8

Unit
Tests

Agenda

Mocking API Responses

Intro to ember-data

Serializers

Exercise #9

Functional (acceptance) Testing

Adapters

Agenda

Creating and
Persisting Records

Exercise #10

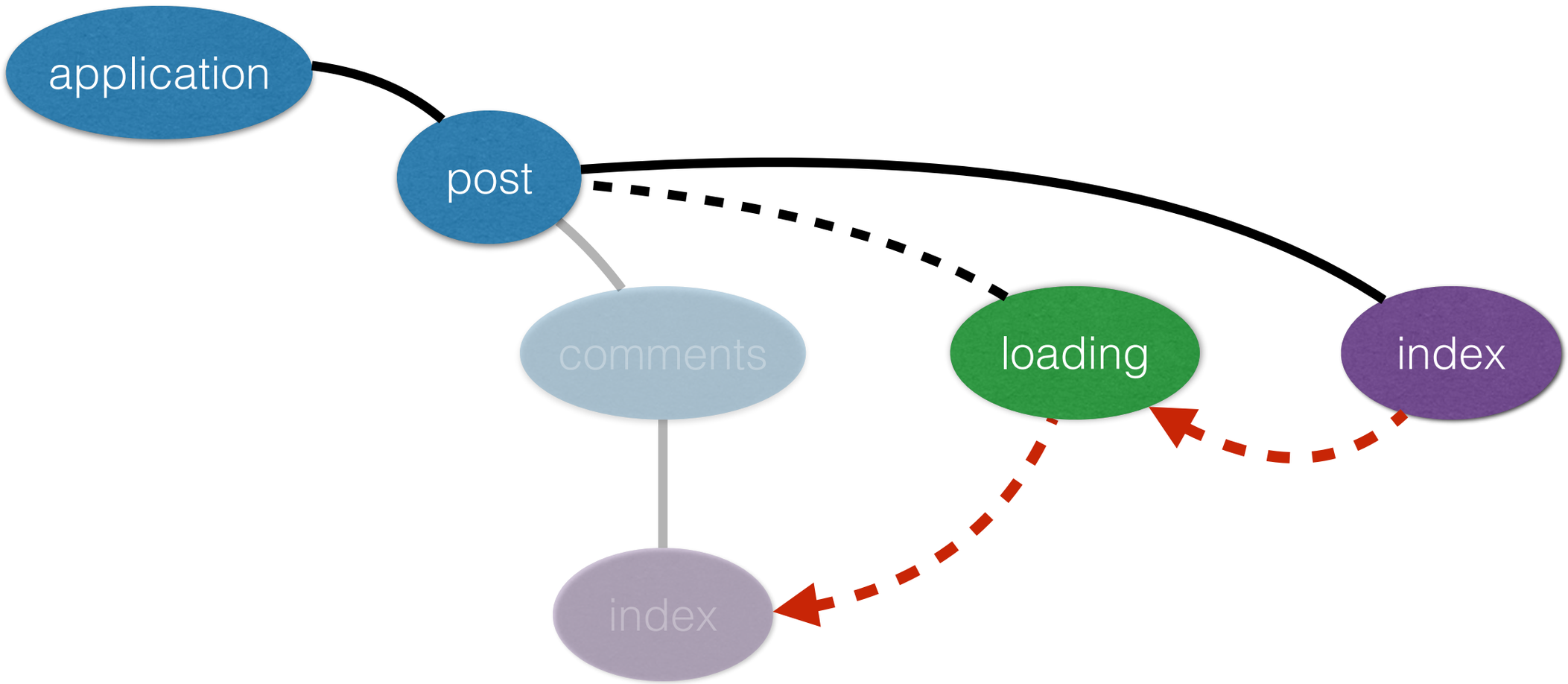
Exercise #11

Upcoming
Ember Features

Loading & Error Substates

Loading Substate

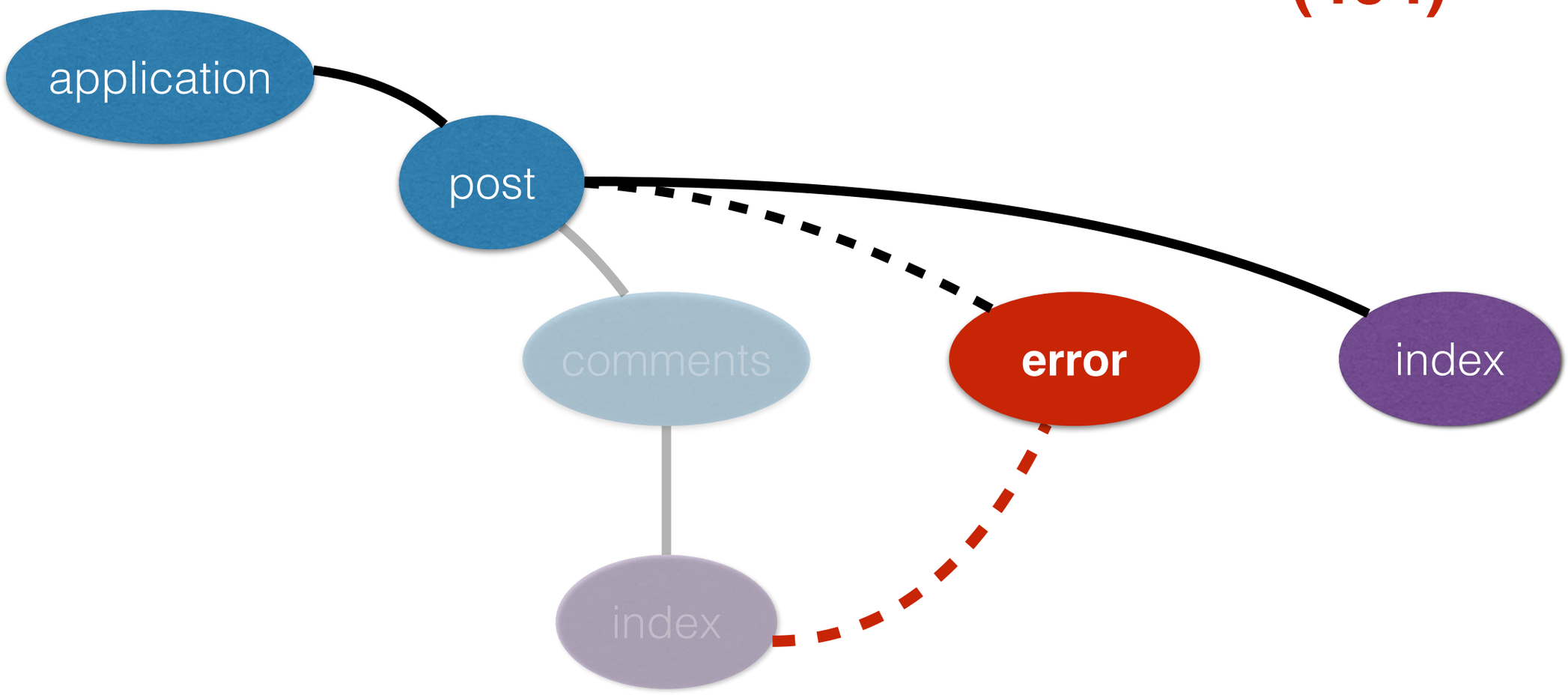
`/post/37` → `/post/37/comments`



Error Substate

`/post/37/comments` → `/post/9000001231`

(404)



Substates

- Loading and error substates are part of the route hierarchy

- REMEMBER - **the substate that will be entered is the immediate child of whatever you're pivoting on while waiting**

- You can enter a route as a substate (no URL change) via `this.intermediateTransitionTo('myroute')`

Error Substate

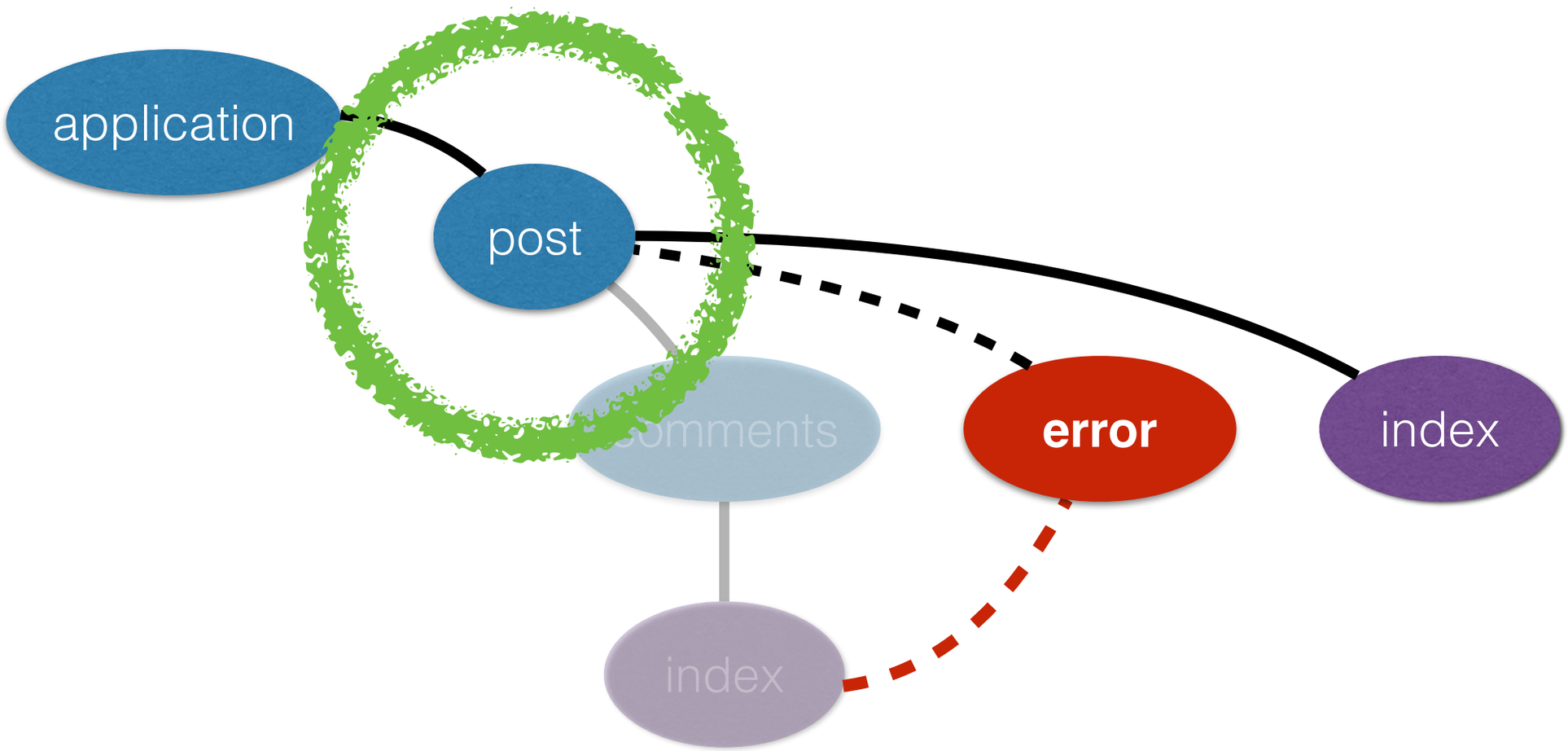
- Also fires the `error` action on the route

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

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

Error Substate

- Remember that actions are handled via the **chain of responsibility** design pattern
  - returning `true` from the action continues bubbling to parent handlers



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 a specific error page telling the user that the org was not found.
    - If it's a 404, I want a dedicated 404 page saying "this organization doesn't exist". I don't want the URL to change as a result of being taken to this error screen.
- Any other kind of error should be handled by a top-level error route (direct child of application route), which should render some information about the error its self to the screen. The `{{debugger}}` handlebars helper will be useful here.

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

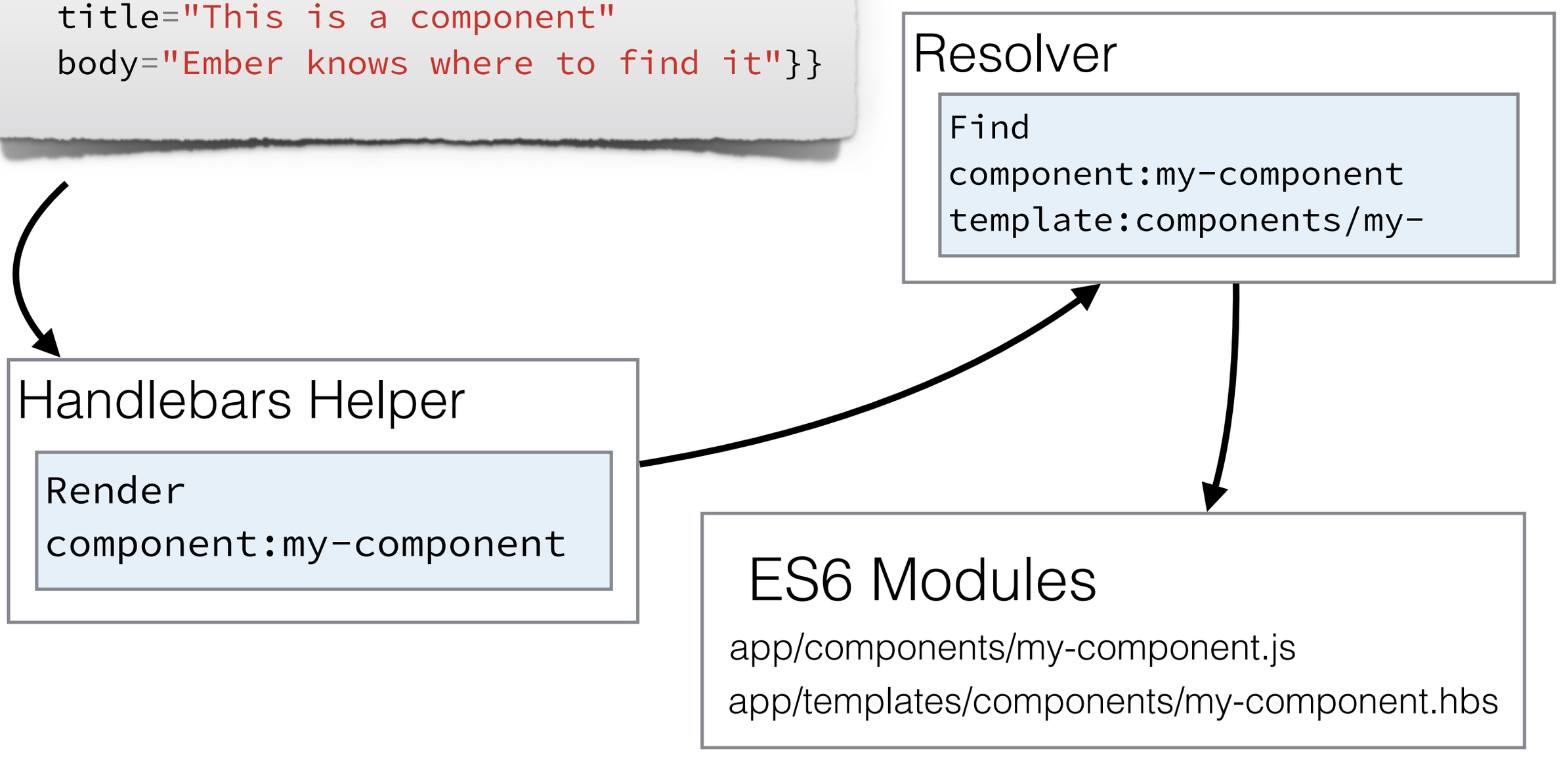
```
{{my-component
      title "This is a title"
       size mySize
on-favorite "thingClicked"}}
```

Inside World | Outside World

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>
```

Components

Handling actions

- Components can handle actions internally, just like routes
- Components can fire actions to the outside world (through an action binding) via `this.sendAction()`
  - If nothing is bound to the “sent” action in the outside world, there is no error
  - the default action’s name is `action`

Component Examples

Pre-Example

Mike North [Expand]

Mike North

- Twitter: @MichaelLNorth
- Github: mike-north

[Collapse]

- Non-trivial complexity encapsulated in the component
- An internal action w/in the component
- An action being sent out of the component

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 repo in the repos list

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

two-way computed properties

```
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

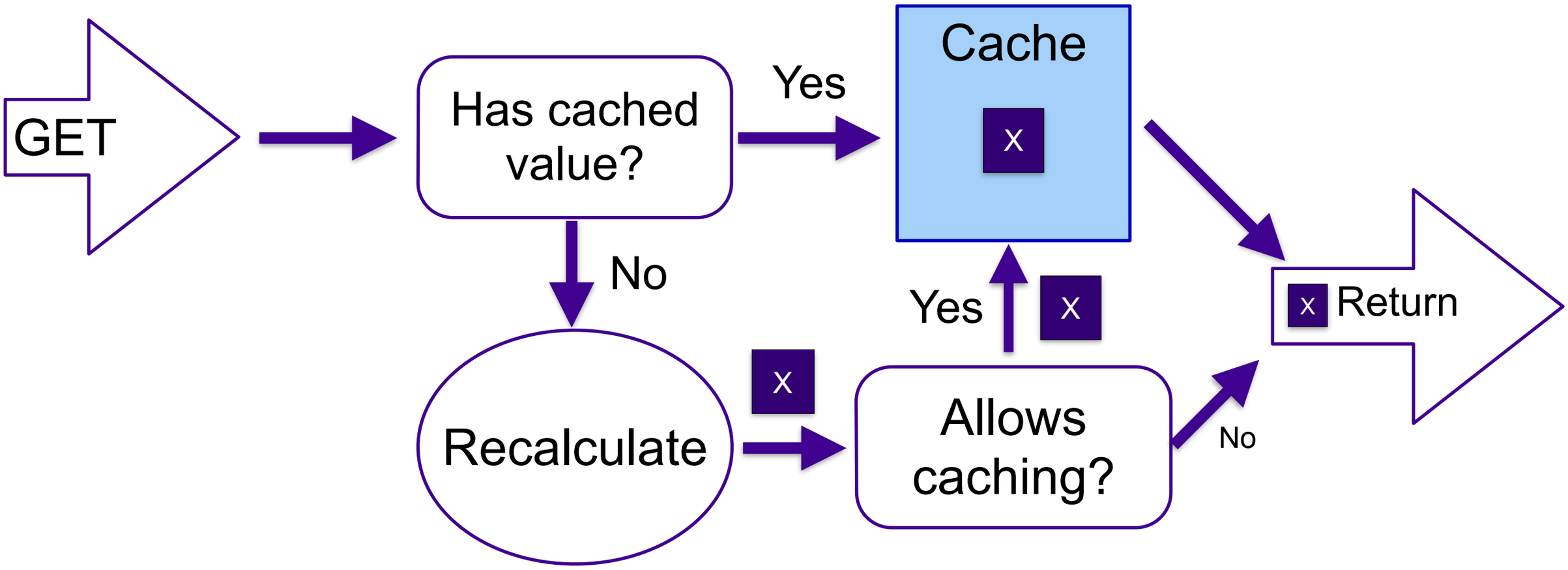
- Syntax for CP dependencies on array properties can be done two ways

- `"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



Computed Property Macros

```
function sum(first, second) {
  return Ember.computed(first, second, function () {
    return this.get(first) + this.get(second);
  });
}

Ember.Component.extend({
  a: 6,
  b: 12,
  c: sum('a', 'b') // 18
});
```

Exercise #7

Using computed properties for “favorite” state

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

- Do this by making a generally useful computed property macro `isFavorited: isInArray('item', 'listOfItems')`

- Make sure that the text of the “Favorite” button alternates between “Favorite” and “Unfavorite” as appropriate, based on the value of your new computed property

Testing

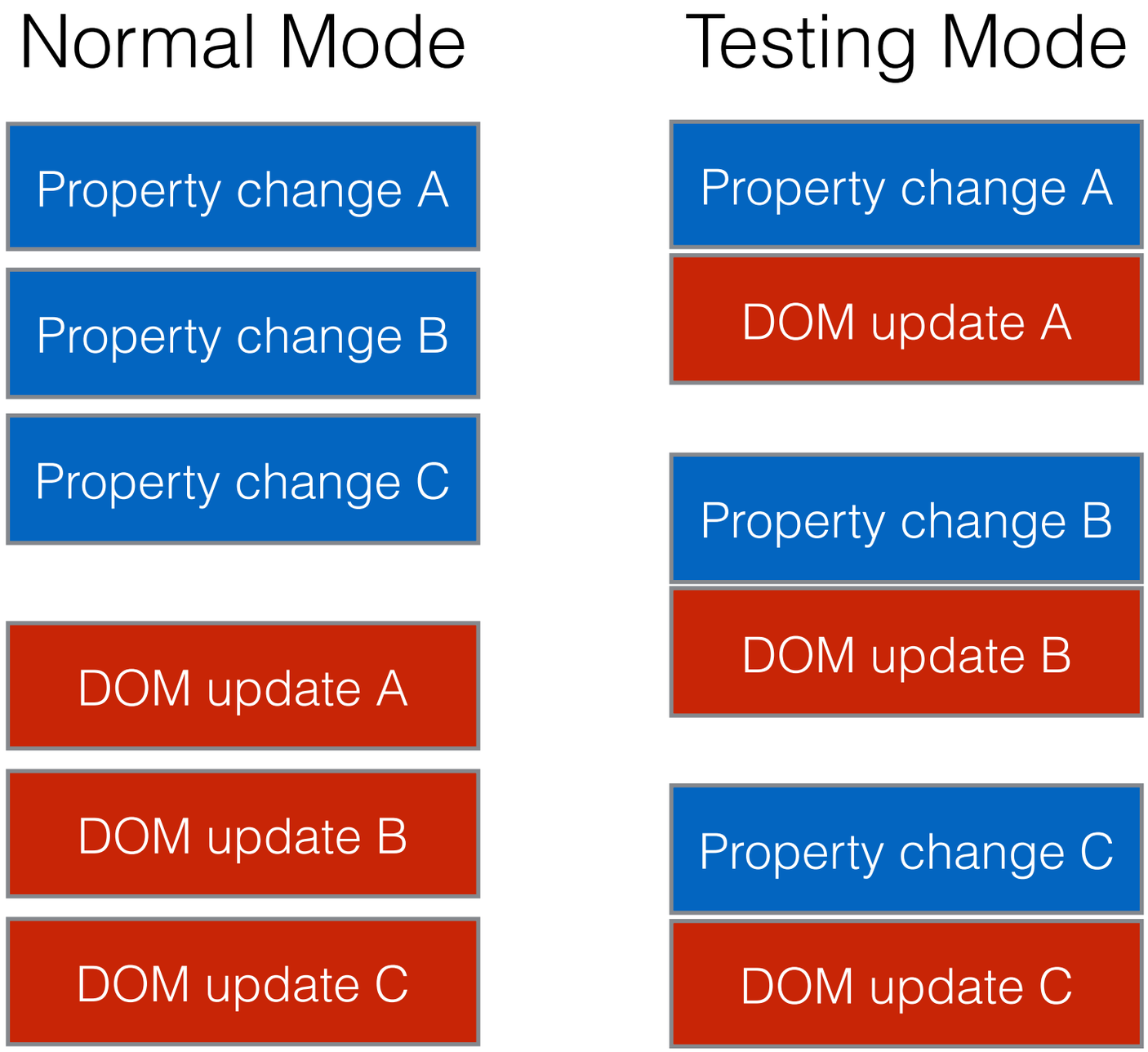
A few flavors of tests

- **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

Type of object	Type of test
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`

Unit tests

- Automatically generated by ember-cli
- Testing in isolation
  - Bring other objects into the test via `needs`
- Very fast

Component Integration Tests

- Test a component in isolation
- Setup test case via
  - setting component state
  - provide the scenario template
- Inline HBS!

Acceptance Tests

- Testing the app as a whole
- Slower
- May involve attempts to make API requests
- Best bet for testing real user workflows

Acceptance Tests

Test Helpers

Synchronous

- currentPath()
- currentRouteName()
- currentURL()
- find(selector, context)

Asynchronous

- click(selector)
- fillIn(selector, value)
- keyEvent(selector, type, keyCode)
- triggerEvent(selector, type, options)
- visit(url)

Acceptance Tests

Dealing with Asynchrony

```
// Visit a URL
visit('/org/emberjs/repos');

// Wait for everything to settle
andThen(function() {
  // Make sure we're at the right URL
  assert.equal(currentURL(), '/org/emberjs/repos');
});

// Click a link (which starts a transition)
click('a[href*="org/emberjs/data"]');

// Wait for everything to settle
andThen(function() {
  // Make sure we're at the right URL
  assert.equal(currentURL(), '/org/emberjs/data/issues');
});
```

Mocking API Responses

- Pretender - github.com/pretenderjs/pretender
- Mirage - www.ember-cli-mirage.com

Exercise #8

Let’s get some test coverage!

Exercise #8

Unit Test

- Unit test for the `is-in-array` module in the `app/utils` folder

Integration Test

- Test the `{{github-org}}` component, ensuring that…
    - Changing the `org` property will result in an appropriate change to the DOM
    - Clicking the “Favorite” button will result in the external action being fired
      see: http://guides.emberjs.com/v2.1.0/testing/testing-components/

Exercise #8

Acceptance Test

1. Visit the / page
2. Verify that the current route name is correct
3. Click on the link for the “emberjs” github org
4. Verify that the current URL is correct
5. Click on the link for the “data” repo
6. Verify that the current URL is correct
7. Verify that there are a nonzero number of issues in the issues list
8. Go to the contributors page
9. Verify that the current URL is correct
10. Verify that there are a nonzero number of contributors in the contributors list

BONUS: Use pretender instead of hitting the API during tests

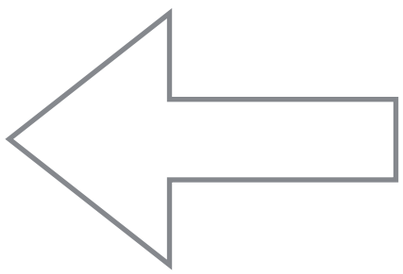
Exercise #8

Using Pretender to Mock APIs

```
function json(obj, status=200) {
  return [status, { 'Content-Type' : 'text/json'}, JSON.stringify(obj)];
}

server = new Pretender(function(){

  this.get('https://api.github.com/orgs/:id',
      () => json({
        id: 99,
        login: 'emberjs',
        name: 'Ember.js'
      }));

  this.get('https://api.github.com/orgs/:id/:repoid',
      () => json([{
        id: 123,
        name: 'data'
      }]));

  this.get('https://api.github.com/repos/:id/:repoid',
      () => json({
        id: 123,
        name: 'data'
      }));

  this.get('https://api.github.com/repos/:id/:repoid/issues',
      () => json([
        {id: 1, title: 'Issue 1'},
        {id: 2, title: 'Issue 2'}
      ]));

  this.get('https://api.github.com/repos/:id/:repoid/contributors',
      () => json([
        {id: 1, login: 'contributor1'},
        {id: 2, login: 'contributor2'}
      ]));
});
```

```
ember install ember-cli-pretender
```

```
import Pretender from 'pretender';
```



To save time, I've mocked out some API responses for you

Handlebars Helpers

Handlebars Helpers

- Another way of extending what you can do in templates
- Particularly useful for when you DON'T want a DOM element
- Can be bound to data, or not
- `Ember.Handlebars.SafeString` for XSS defense

Handlebars Helpers

```
{{number-with-units 1 'framework'}}
{{number-with-units 2 'apple'}}
{{number-with-units 3 'course'}}
```

1 framework 2 apples 3 courses

```
//app/helpers/number-with-units.js
import Ember from 'ember';

const { Helper: { helper } } = Ember;

export function numberWithUnits(params/*, hash*/) {
  const [qty, unit] = params;
  if (qty === 1) {
    return `${qty}
    ${Ember.Inflector.inflector.singularize(unit)}`;
  } else {
    return `${qty}
    ${Ember.Inflector.inflector.pluralize(unit)}`;
  }
}
export default helper(numberWithUnits);
```

Exercise #9

Build your own Handlebars helper

Exercise #9

nice-number

- If a number is < 1000, let it pass through unaltered
- If a number is > 1000, express it as something like “2.6K”
- Unit test coverage!

Ember-Data

Ember-Data

- A persistence library
- Key objects: Store, Model, Adapter, Serializer
- Internal data structures: JSON-API (http://jsonapi.org/)
- Define how to interact with your APIs, and then no longer need to explicitly interact with them explicitly

Ember-Data

Model

- A representation of any data you may present to the user (usually persistent)

- Mostly defined by attributes, and relationships to other models

- “model” is the factory that defines the structure of “records”

Ember-Data

Model

```
// app/models/book.js
import DS from 'ember-data';

const { attr, hasMany } = DS;

export default DS.Model.extend({
  title:          attr('string'),
  publishedAt:    attr('date'),
  chapters:       hasMany('chapter')
});
```

Ember-Data

Store

- Where you get/create/destroy records
- A single source of truth
  - This means that all changes are kept in sync!
- Similar concept in Facebook/flux, angular-data

Ember-Data

Store - Retrieving Data

```
// API request for all records of type "author"
this.store.findAll('author');
// API request for record of type "post" with id 37
this.store.findRecord('post', 37);

// look in cache for all records of type "author"
this.store.peekAll('author');
// look in cache for record of type "post" with id
37
this.store.peekRecord('post', 37);
```

Ember-Data

Adapters

- Handle the particulars of making requests
- You'll typically never touch them directly
- Adapter used:
  - First attempt to find a type-specific adapter
  - Fall back to application adapter

Ember-Data

Adapter

```
export default DS.RESTAdapter.extend({
  host: 'https://api.github.com',

  urlForQuery (query, modelName) {
    switch(modelName) {
      case 'repo':
        return `${this.get('host')}/orgs/${query.orgId}/repos`;
      default:
        return this._super(...arguments);
    }
  }
});
```

Ember-Data

Serializers

- Handle the particulars of converting JSON between API and client-side representations

- normalizeResponse - lowest level normalization

- many other potentially more convenient normalization hooks

Ember-Data

Store - Retrieving Data

```
export default DS.RESTSerializer.extend({

  normalizeResponse(store, modelClass, payload, id, requestType){
    const newPayload = { org: payload };
    return this._super(store, modelClass, newPayload, id, requestType);
  }

});
```

Exercise #10

Move from $.get to ember-data

Exercise #10

Use this.store.*

- On the `org` route and the `org.repos` route, use ember-data to retrieve data instead of jQuery
- For building complex paths that need parent entity IDs, I suggest using `this.store.query` or `this.store.queryRecord` — some customization of the URL building in the adapter will be required
- Extend from `DS.RESTSerializer` and `DS.RESTAdapter`

Exercise #10

Application Adapter

- You'll probably need to override `urlForQueryRecord`, `urlForQuery`
- Set `https://api.github.com` as the adapter's `host` property

Application Serializer

- Highest level place to customize JSON is normalizeResponse
- The `modelName` property on the `primaryModelClass` argument will give you something like "org" or "repo"
- `Ember.Inflector.inflector.pluralize` is a handy function for pluralizing a string (i.e,. "org" to "orgs")
- Remember that ember-data cares about top-level key names!

Ember-Addons

Ember Addons

- Ember's plugin architecture
- Structure is very similar to an app
- Built-in demo app for acceptance testing and development
- Opportunity to run "post-install" code

New In 2

New in 2

New in 2

- Late 1.x releases…
  - introduced some awesome stuff
  - deprecated some things
- 2.0 release removed the deprecated code
  - future 2.x releases will introduce more brand new functionality

New in 2

Recent Releases

Ember Version	Enhancement
1.10	HTMLbars
1.11	Attribute bindings
1.12	Computed property improvements
1.13	Glimmer (rendering engine)
2.0	Remove deprecated code

…coming soon…

…coming soon…

Simplified mental model

- No more controllers
- Angle bracket components

```
<mike-card title="Mike North">
   This is the body of my component
</mike-card>
```

- Re-worked queryParam support

…coming soon…

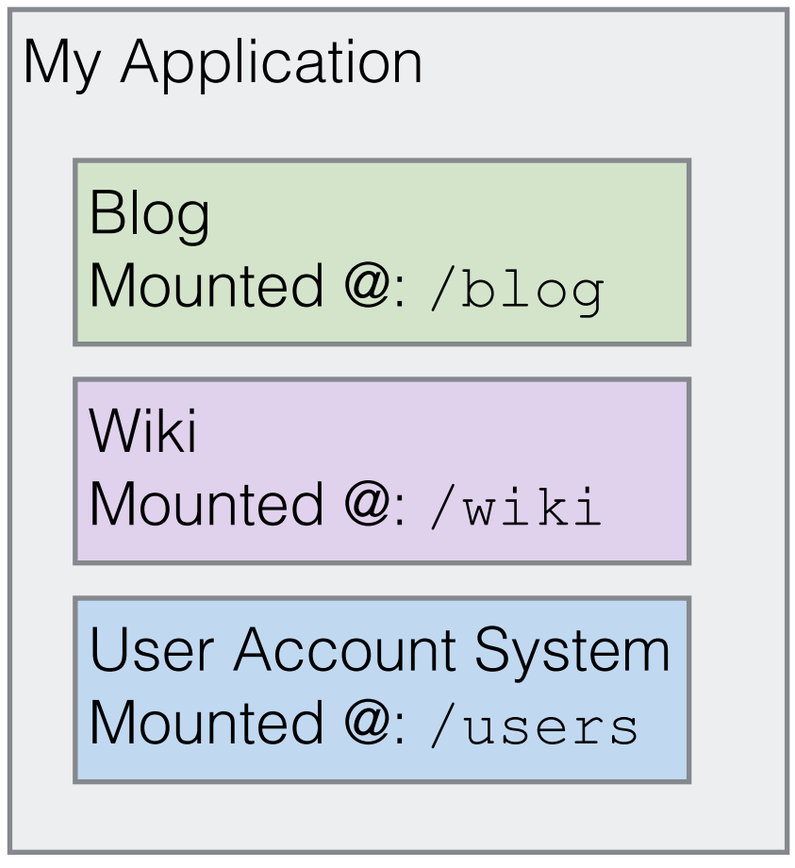
Fastboot - Server-side Rendering

- Already exists as add-on, but really just for SEO
    - Very limited
- Fast initial page load
- Corporate sponsorship

…coming soon…

Engines

- Similar to Rails engines
- Modular app construction
- Load engines only when needed



Why consider ember?

Ember, Standing out

Developers are expensive

- Focus decisions unique to your app
- Consistency leads to interoperability
- Huge complex apps are a target use case
- Robust and active community

Adoption

NETFLIX YAHOO! Travis CI Square

Microsoft zendesk twitch livingsocial

heroku urbanspoon TED KICKSTARTER

LinkedIn GROUPON McGraw Hill MHElabs Vine


ember

Agenda

Routes

Actions

Talking to APIs &
using simple models

Exercise #2

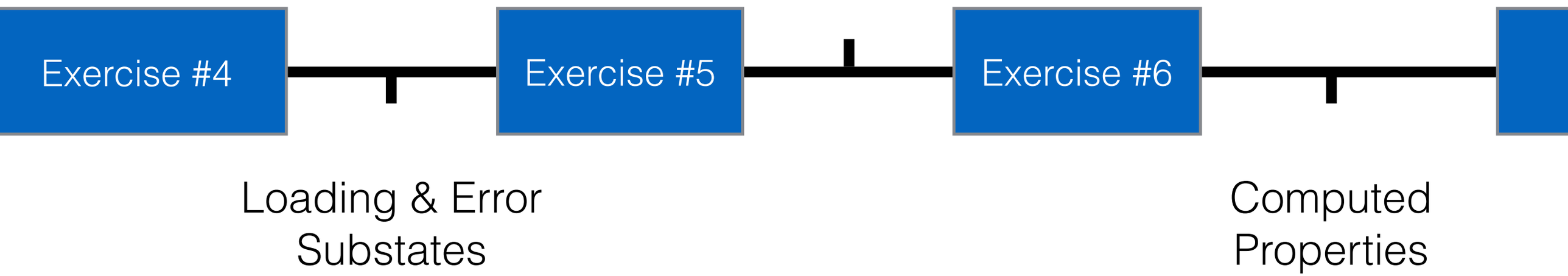
Exercise #3

Services

Object
Properties

Agenda

Components

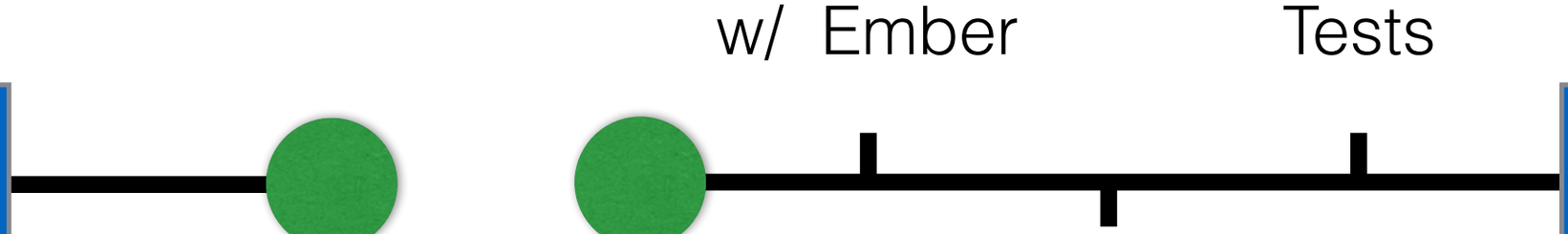


Agenda

Intro to Testing w/ Ember

Component Integration Tests

Exercise #7



Exercise #8

Unit Tests

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