

<https://github.com/mike-north/ember-pro>
[./EmberPro.pdf](#)

ember PRO



Sept 5, 2017
Mike North

© Mike.Works, Inc. 2017. All rights reserved



What's this course all about?

- ▶ Patterns, not just concepts
- ▶ Ecosystem, not just framework
- ▶ Tons of ES2018 & some Typescript ...
- ▶ Focus on:
 - ▶ Tools & Debugging
 - ▶ Managing Complexity
 - ▶ Easy-mode Architecture



**Building &
Booting**

**Professional
Patterns**

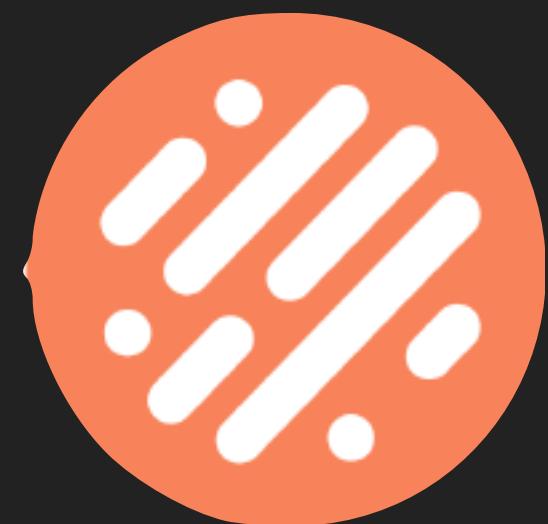
**Customizing
CLI**

**Amazing
Addons**

You'll walk away with an in-depth understanding of...

- ▶ Broccoli: the asset pipeline
- ▶ Ember-cli blueprints
- ▶ Build process
- ▶ Boot process
- ▶ Ember addons
- ▶ Debugging like a pro
- ▶ CRUD routes & actions
- ▶ Useful ES6 in Ember
- ▶ Server-side Rendering
- ▶ Modular app architecture
- ▶ Managing State
- ▶ Popular Addons

You'll walk away with a quick look at...

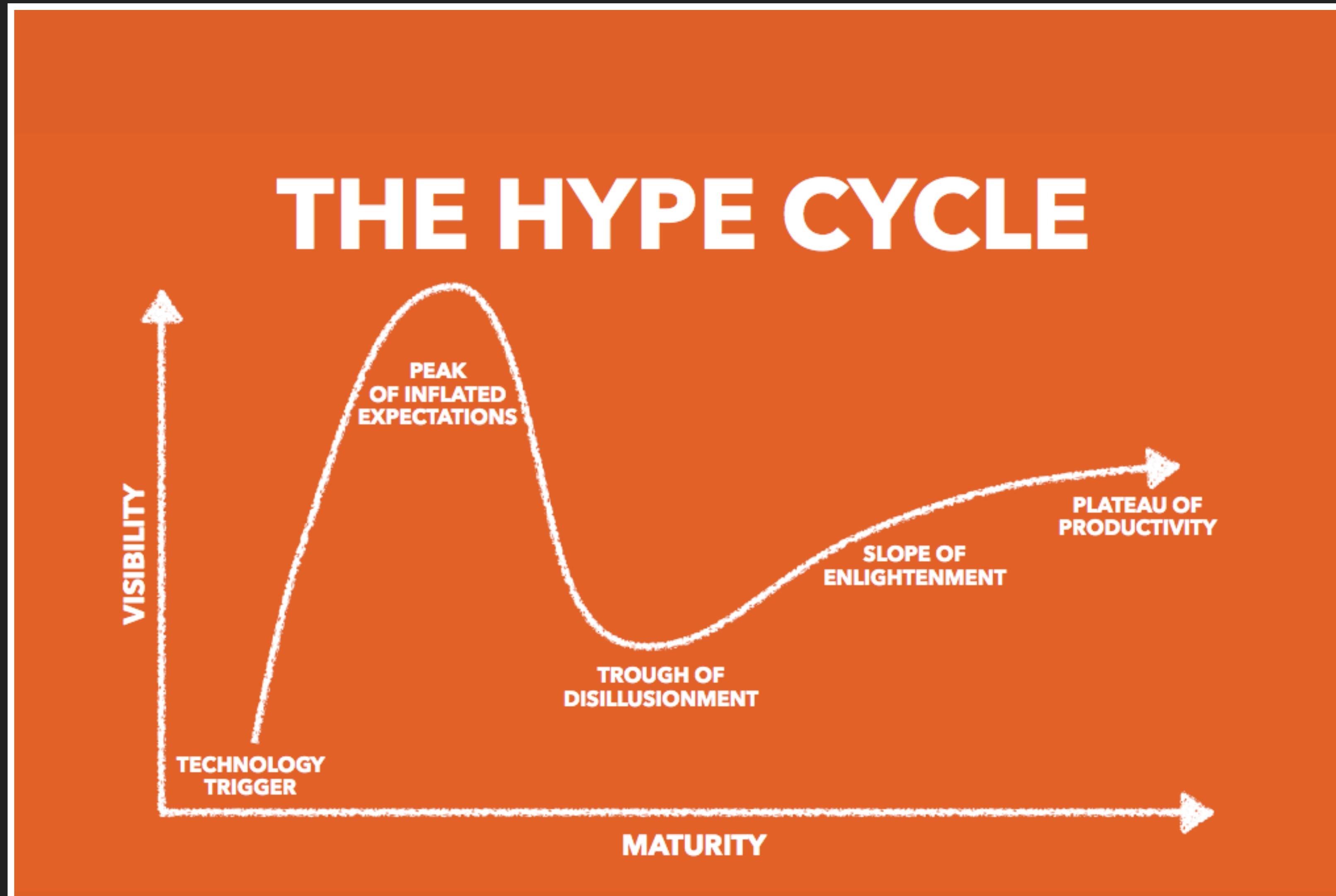


glimmer

You'll walk away with...

A start on leading a large,
complex ember project

Ember's Place among JS Frameworks





Ember ~~Magic~~ Autowatión

Container

A collection of an app's **important long-living objects**, organized by **key**

`"route:index": Class,`
`"router:main": Class,`

Core Concept

Why do we have a container?

- ▶ Keep singletons around
- ▶ Refer to important objects by name, keeping code loosely coupled
- ▶ Laziness (the awesome kind) & Automation
- ▶ App, Engine and Addon-level Encapsulation

Common Container Problems

- ▶ Empty or nonexistent component

► Uncaught Error: Attempting to inject an unknown injection: 'service:auth'(...)

- ▶ Custom module behavior is ignored
- ▶ Addon modules helpers aren't available to consuming app

► Uncaught Error: Compile Error: titlecase is not a helper(...)

Container Best Practices

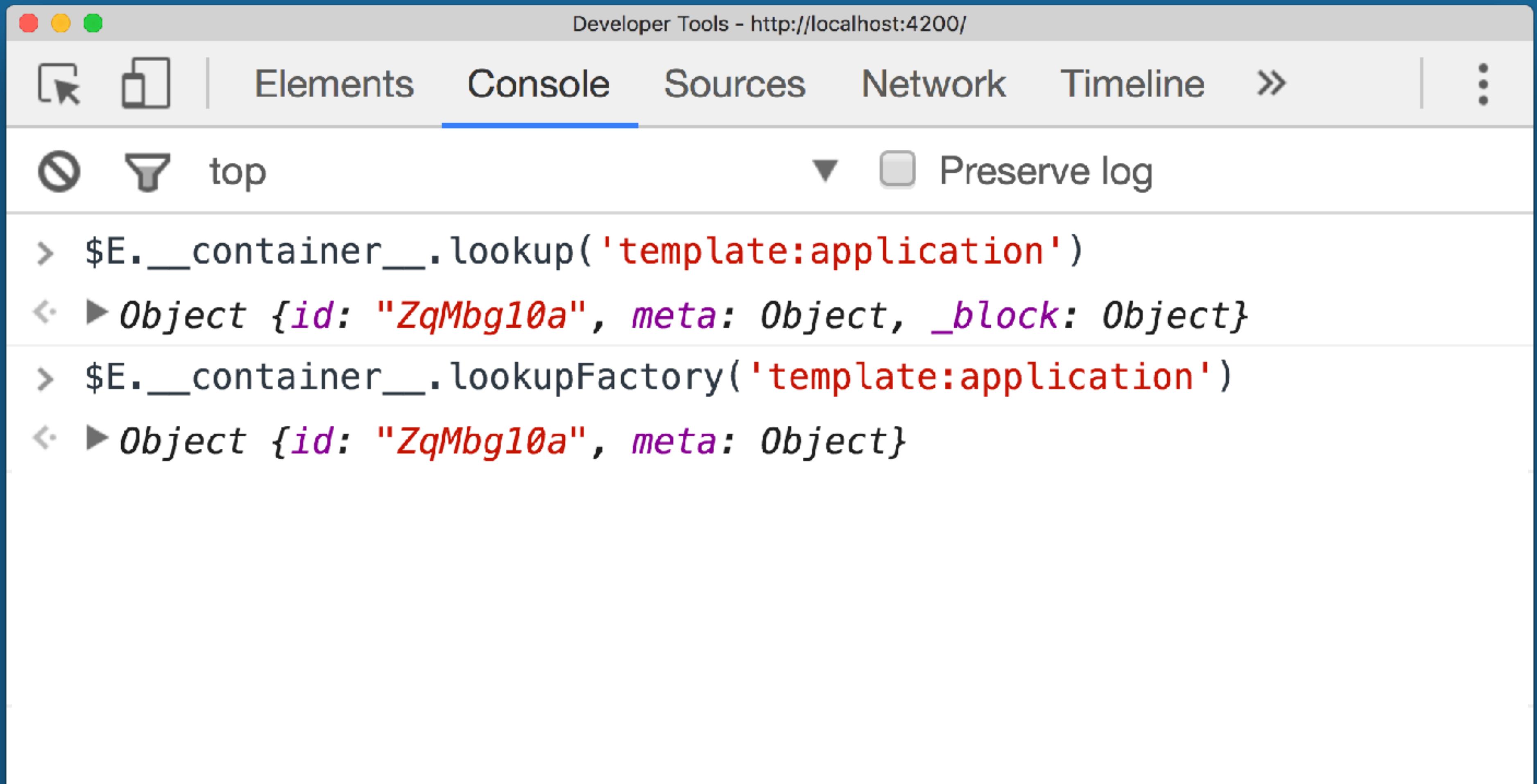
- ▶ Let Ember manage it. Don't touch it directly
- ▶ Little reason to modify it after boot completes
- ▶ Always use **Ember.getOwner(this)**
- ▶ Great place for "config" objects

Debugging: Container

The screenshot shows the Chrome Developer Tools interface with the 'Ember' tab selected. The URL is `http://localhost:4200/`. On the left, there's a sidebar with various developer tools like Elements, Console, Sources, Network, Timeline, Profiles, Application, Security, Audits, and Ember. The 'Ember' tab is active. The main area has a search bar at the top with the word 'glue'. Below it, there's a tree view under 'TYPES' with categories: View Tree, Routes, Data, Deprecations (0), Info, ADVANCED, Promises, Container, and Submit an Issue. The 'Container' category is currently selected. Under 'main', the 'application (1)' item is also selected. To the right, the details pane shows the properties of the selected object:

```
glue
Own Properties
name: glue
version: 0.0.0+756e00a2
autoboot: true
(subclass of Ember.Application)
Ember.Application
Ember.Engine
Ember._RegistryProxyMixin
function () { if (!wasApplied) {
  Class.prototype(); // prepare
  prototype... } if (arguments.length
> 0) { initProperties =
[arguments[0]]; }
this.__defineNonEnumerable__ember
var m = _emberMetal.meta(this);
var proto = m.proto; m.proto =
this; if (initProperties) { // capture
locally so we can clear the closed
over variable var props =
initProperties; initProperties = null;
var concatenatedProperties =
```

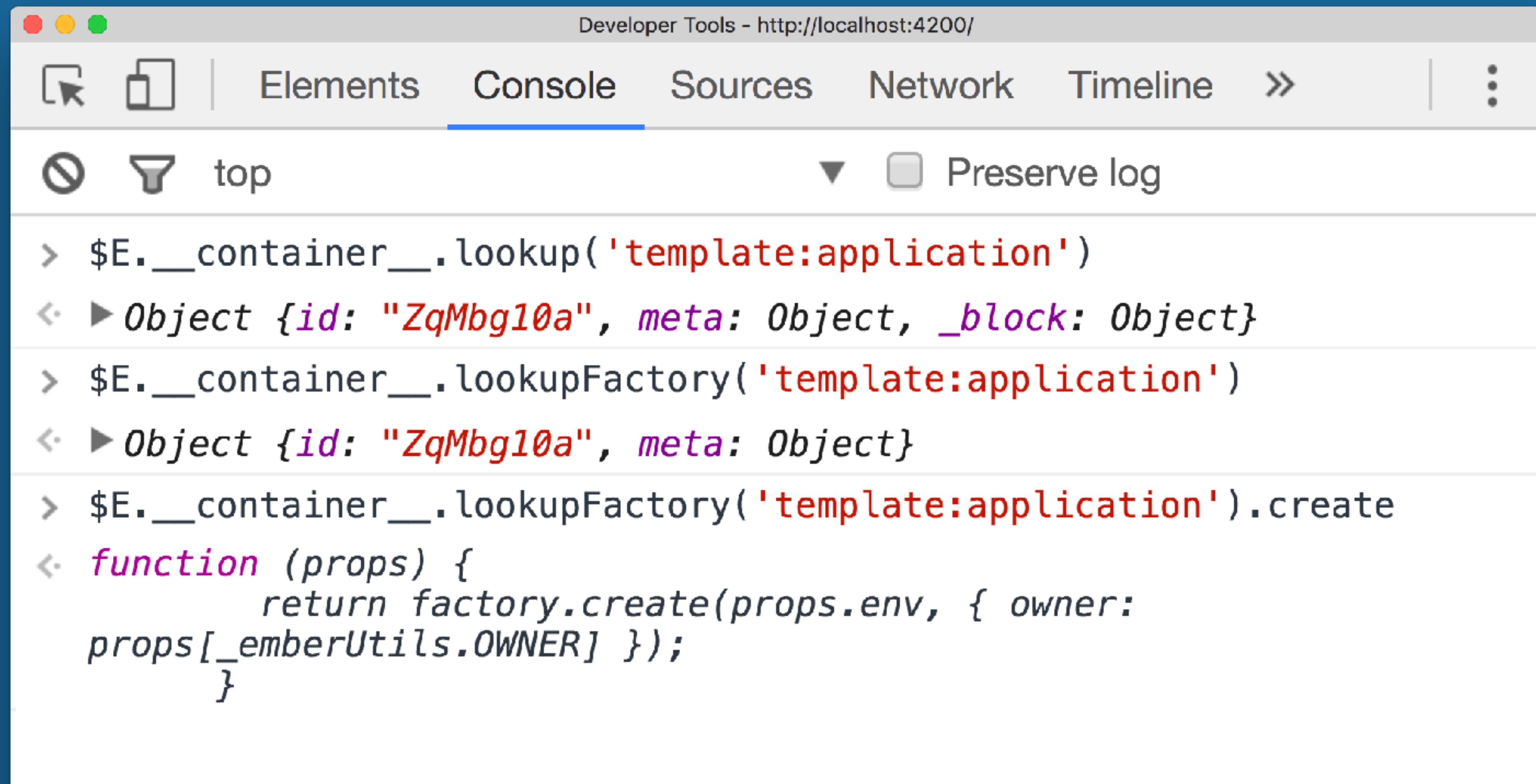
Debugging: Container



The screenshot shows the Chrome Developer Tools interface with the 'Console' tab selected. The title bar reads 'Developer Tools - http://localhost:4200/'. Below the tabs, there are filters: a 'no errors' icon, a 'filter' icon, and a dropdown set to 'top'. A checkbox for 'Preserve log' is also present. The main area displays the following log entries:

```
> $E.__container__.lookup('template:application')
< ► Object {id: "ZqMbg10a", meta: Object, _block: Object}
> $E.__container__.lookupFactory('template:application')
< ► Object {id: "ZqMbg10a", meta: Object}
```

Debugging: Container



The screenshot shows the Chrome Developer Tools interface with the 'Console' tab selected. The title bar reads 'Developer Tools - http://localhost:4200/'. Below the tabs, there are filters for 'top' and a checked 'Preserve log' option. The main area displays a stack trace:

```
> $E.__container__.lookup('template:application')
< ► Object {id: "ZqMbg10a", meta: Object, _block: Object}
> $E.__container__.lookupFactory('template:application')
< ► Object {id: "ZqMbg10a", meta: Object}
> $E.__container__.lookupFactory('template:application').create
< function (props) {
    return factory.create(props.env, { owner:
  props[_emberUtils.OWNER] });
}
```

Resolver

Given a **container key**, Attempts
to find **the appropriate JS module**
in your codebase



**Core
Concept**

Why do we have a Resolver?

- ▶ Many ways for us to organize our code

```
// Globals  
Ember.PostsController = ...
```

```
// AppKit, ember-cli  
controllers/posts.js
```

```
// (old) Pods  
posts/controller.js
```

How do we play with it?

- ▶ Ember.Application instance has `resolveRegistration()`

The screenshot shows the Ember Inspector interface for the URL `http://localhost:4200/`. The sidebar on the left lists components: Promises, Container (selected), Render Performance, and Submit an Issue. The main pane displays the application structure under the `main` tab, with `application (1)` selected. The right pane shows the properties of the selected component:

- `name: commently`
- `version: 0.0.0+62559fae`
- `autoboot: true`

Below this, it indicates `(subclass of Ember.Application)`, `Ember.Application`, `Ember.Engine`, and `Ember._RegistryProxyMixin`.

At the bottom, the console log shows:

```
Ember Inspector ($E):
  ► Class {__registry__: Registry, _readinessDeferrals: 0, _booted: true, __globalsMode: true,
    __deprecatedInstance__: Class...}
  > $E.resolveRegistration('route:index')
  < function (subclass of Ember.Route)
```

A red callout box with the text "NOTE: Constructor, not instance" points to the `$E.resolveRegistration` line in the console log.

One step deeper: Defining and loading modules

```
define('mike', ['ember'], function(em) {  
  const Ember = em.default;  
  console.log(Ember.VERSION);  
  return {  
    default: {  
      hello: "world"  
    }  
  };  
});  
  
const myModule = require('mike');
```

Bringing them into your asset pipeline

ember-cli-build.js

```
app.import('vendor/mike.js', {  
  exports: ['mike']  
})
```

Peeking Into Loaded Modules

```
> require._stats
< ► Object {define: 465, require: 15, reify: 139, findDeps: 140, modules: 465...}
> require.entries
< ▼ Object ⓘ
  ► commently/app: Module
  ► commently/components/-lf-get-outlet-state: Module
  ► commently/components/illiquid-model: Module
  ► commently/components/liquid-bind: Module
  ► commently/components/liquid-child: Module
  ► commently/components/liquid-container: Module
```

Add an ES6-ified module

- ▶ Your boss wants you to ES6-ify some `Math.*` utilities
- ▶ For now, you can write code in `app.js`
- ▶ In the end, I should be able to do this:

```
import { default as math, PI } from 'math';

math.sqrt(4); // 2
console.log(PI) // 3.145926 ...
```

Booting Up

Ember.Application

Booting Up

domReady



```
Ember.Application#_bootSync {  
}
```

Booting Up

domReady



```
Ember.Application#_bootSync {  
  this.runInitializers();  
}
```

Array[11]

0:	"App Version"
1:	"container-debug-adapter"
2:	"data-adapter"
3:	"ember-data"
4:	"export-application-global"
5:	"injectStore"
6:	"startup"
7:	"store"
8:	"transforms"
9:	"domTemplates"
10:	"deferReadiness in `testing` mode"

Initializer

A **function** that's invoked **while** your App boots up.

**Core
Concept**

Initializers

→  Modular boot process

Initializers

```
import Lemon from 'fruits/lemon';

export function initialize(app) {
    app.register('fruits:lemon', Lemon);
}

export default {
    name: 'my-initializer',
    after: 'other-initializer',
    initialize
};
```

Initializers

- ➡  Modular boot process
- ➡  Happens **WHILE** Application boots
- ➡  deferReadiness, advanceReadiness

Why do we have these things?

- ▶ Application-level, pre-boot setup/constructor logic
- ▶ Registration API (container stuff)
- ▶ Use cases:
 - ▶ Add a configuration object to the container
 - ▶ Setup for a/b testing
 - ▶ Conditionally load code

Booting Up

domReady



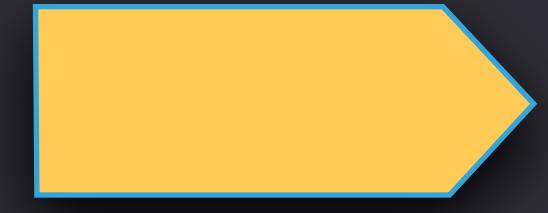
```
Ember.Application#_bootSync {  
  this.runInitializers();  
  this.advanceReadiness();  
}
```

Booting Up

domReady



```
Ember.Application#_bootSync {  
  this.runInitializers();  
  this.advanceReadiness() {  
    this.didBecomeReady();  
  }  
}
```



Booting Up

Ember.Application#bootSync

```
Ember.Application#didBecomeReady {  
}
```

Booting Up

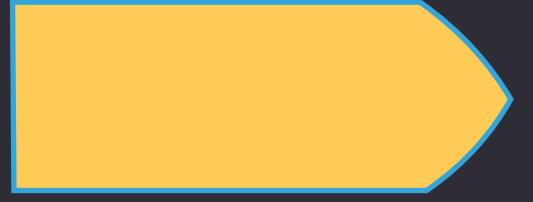
```
Ember.Application#_bootSync  
Ember.Application#didBecomeReady {  
  
  if (this.autoboot) {  
    // Instantiate a Ember.ApplicationInstance  
    let instance = this.buildInstance();  
    instance._bootSync();  
  
    instance.startRouting();  
  }  
  
}  
}
```

Booting Up

Ember.Application#_bootSync

Ember.Application#didBecomeReady

Ember.ApplicationInstance#_bootSync {



```
this.setupRegistry();
// define root element
// define location
this.runInstanceInitializers();
if (isInteractive) {
  this.setupEventDispatcher();
}
}
```

Instance Initializer

A function that's invoked immediately after your App boots up.

**Core
Concept**

Instance Initializers

- ➡  Modular boot process
- ➡  Happens **AFTER** ApplicationInstance boots
- ➡  Re-runs on Fastboot builds
- ➡  Think: Container, not Registry

Instance Initializers

- ▶ All the first application-instance initializer is run after the last application initializer
- ▶ Access to a fully materialized container, store, etc...
- ▶ Cannot defer/advance readiness

Registration API

- ▶ In an initializer, you're primarily dealing wth factories, and `app.register` and `app.resolveRegistration`
- ▶ In an instance-initializer, you're guaranteed a container. `app.lookup` is now available – this is for instances!

Registration API

```
import Lemon from 'fruits/lemon';

export function initialize(app) {
  app.register('fruits:lemon', Lemon);
}

export default {
  name: 'my-initializer',
  after: 'other-initializer',
  initialize
};
```

Registration API

- ▶ An app instance allows registration of anything in the container
- ▶ `.create()` is called on container items when looked up
- ▶ Same instance is reused over and over
- ▶ An optional third argument allows two options for deviating from that behavior

```
{   singleton: true, // re-use the same instance  
  instantiate: true } // call Ember.Object.create when looked up
```

Booting Up

Ember.Application#_bootSync

Ember.Application#didBecomeReady

Ember.ApplicationInstance#_bootSync {

```
this.setupRegistry();
// define root element
// define location

this.runInstanceInitializers();
if (isInteractive) {
  this.setupEventDispatcher();
}
}
```

Booting Up

```
Ember.Application#_bootSync
Ember.Application#didBecomeReady {

  if (this.autoboot) {
    // Instantiate a Ember.ApplicationInstance
    let instance = this.buildInstance();

    instance._bootSync();

    instance.startRouting();
  }

}
```

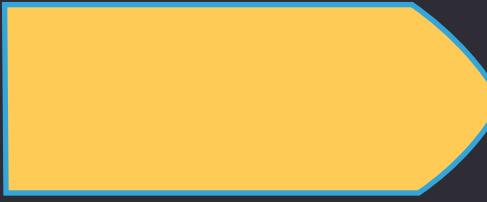
Booting Up

Ember.Application#`_bootSync`

Ember.Application#`didBecomeReady`

Ember.ApplicationInstance#`startRouting`

Ember.Router#`startRouting` {



```
let initialURL = location.getURL();
```

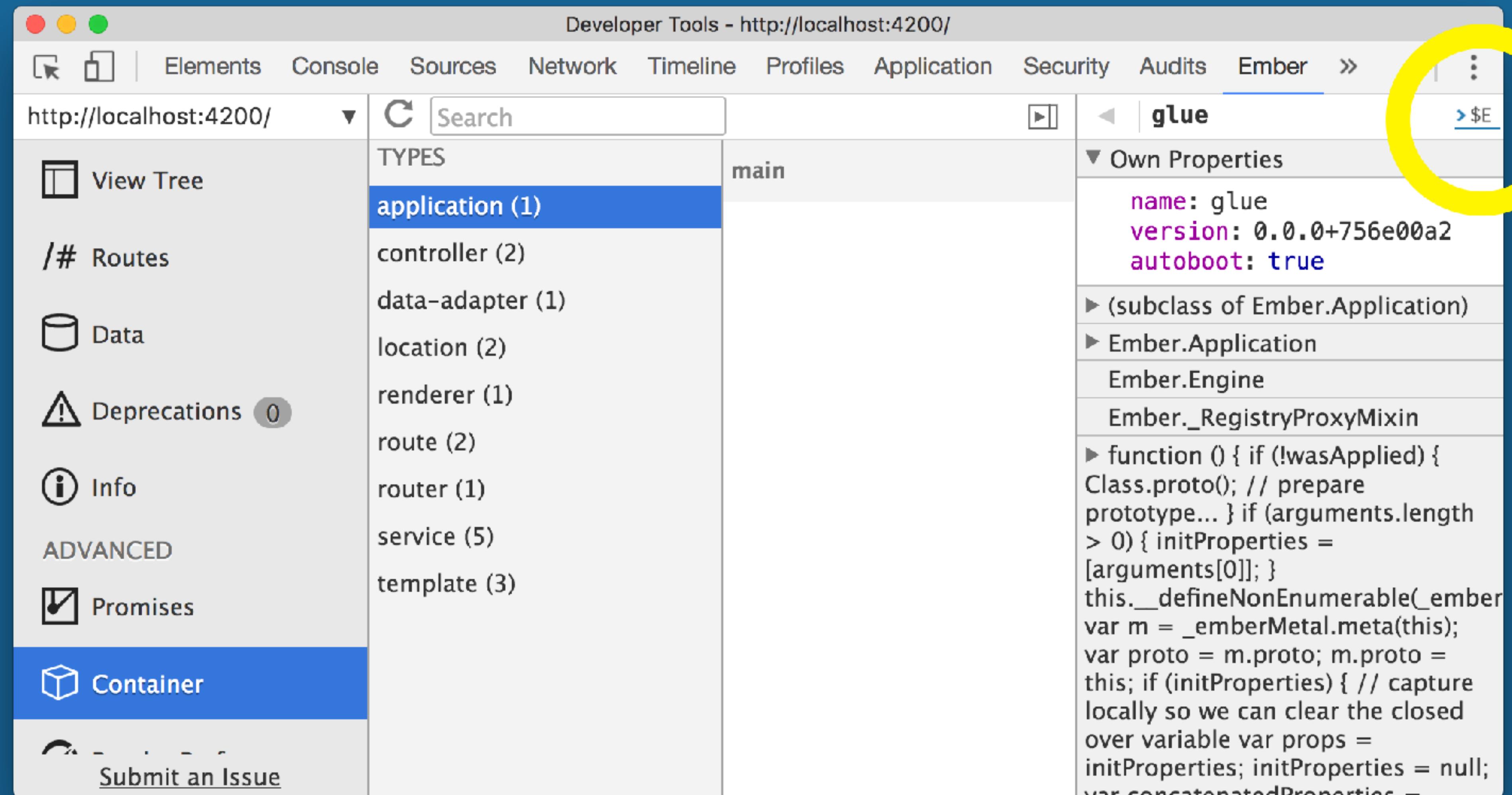
```
let initialTransition = this.handleURL(initialURL);
```

```
}
```

Application vs ApplicationInstance

Initializer vs InstanceInitializer

Debugging: Registry



Developer Tools - http://localhost:4200/

Elements Console Sources Network Timeline Profiles Application Security Audits Ember

http://localhost:4200/

C Search

TYPES

application (1)

controller (2)

data-adapter (1)

location (2)

renderer (1)

route (2)

router (1)

service (5)

template (3)

main

Own Properties

name: glue
version: 0.0.0+756e00a2
autoboot: true

(subclass of Ember.Application)

Ember.Application

Ember.Engine

Ember._RegistryProxyMixin

function () { if (!wasApplied) {
Class.prototype(); // prepare
prototype... } if (arguments.length
> 0) { initProperties =
[arguments[0]]; }
this.__defineNonEnumerable__ember
var m = _emberMetal.meta(this);
var proto = m.proto; m.proto =
this; if (initProperties) { // capture
locally so we can clear the closed
over variable var props =
initProperties; initProperties = null;
var concatenatedProperties =

View Tree

Routes

Data

Deprecations 0

Info

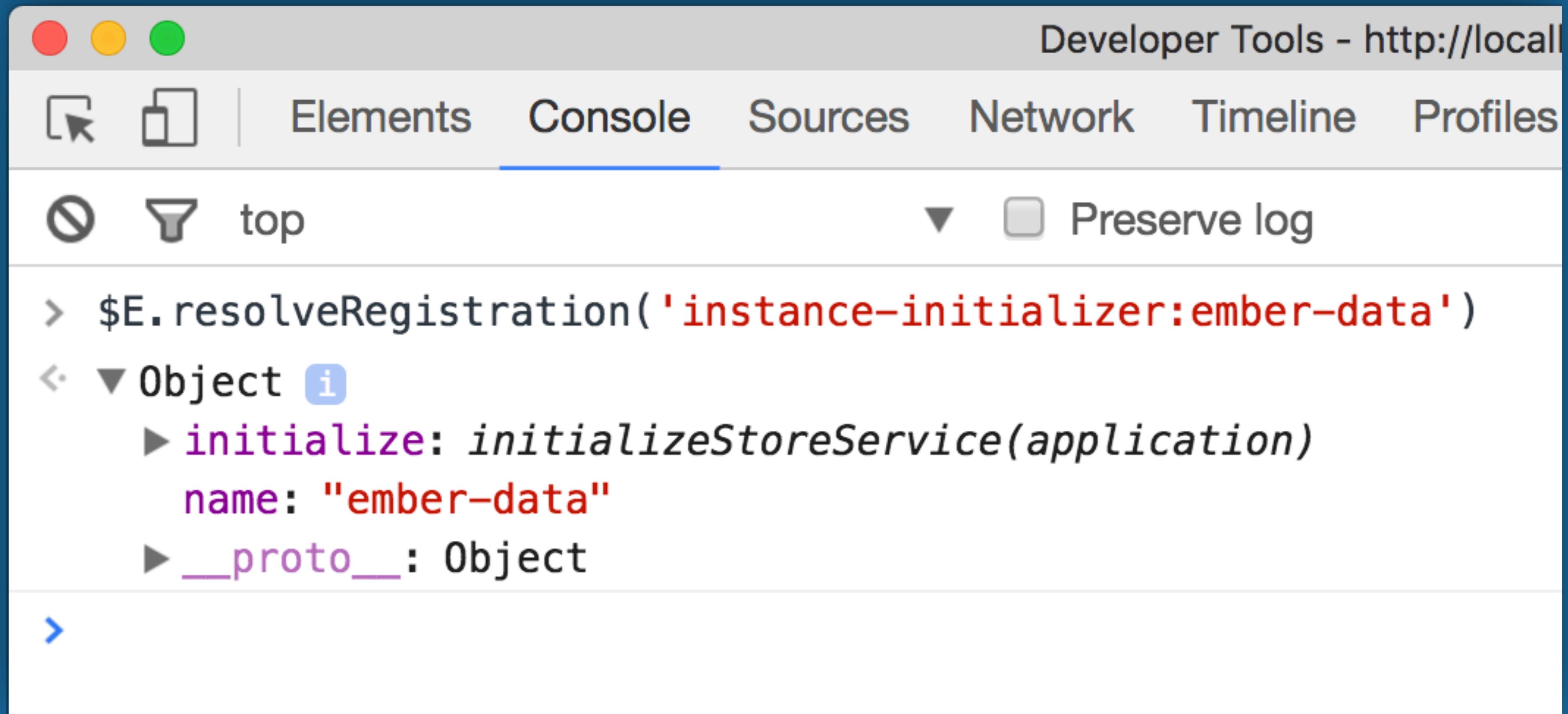
ADVANCED

Promises

Container

Submit an Issue

Debugging: Registry



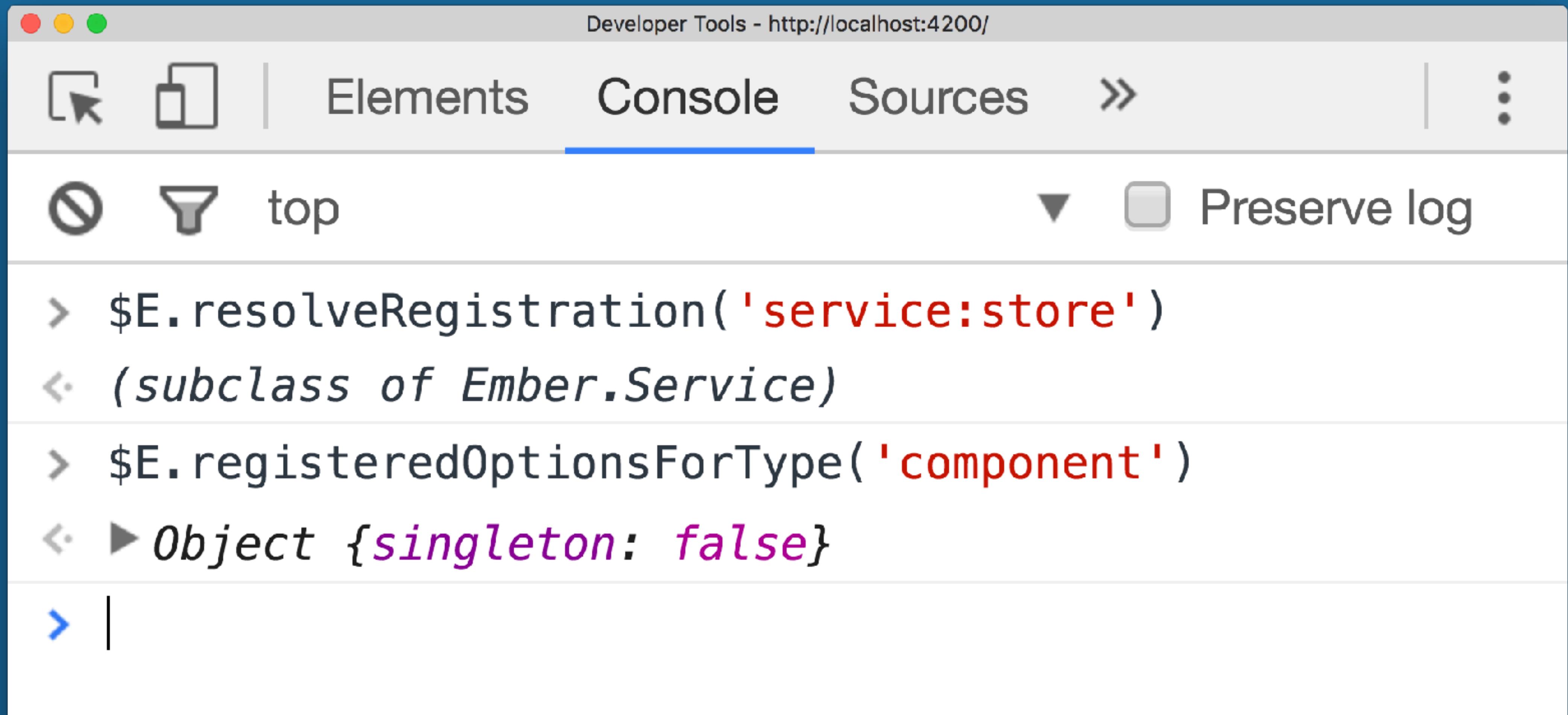
Developer Tools - http://localhost:4200

Elements Console Sources Network Timeline Profiles

top ▾ Preserve log

```
> $E.resolveRegistration('instance-initializer:ember-data')
<- ▼ Object ⓘ
  ► initialize: initializeStoreService(application)
  name: "ember-data"
  ► __proto__: Object
>
```

Debugging: Registry

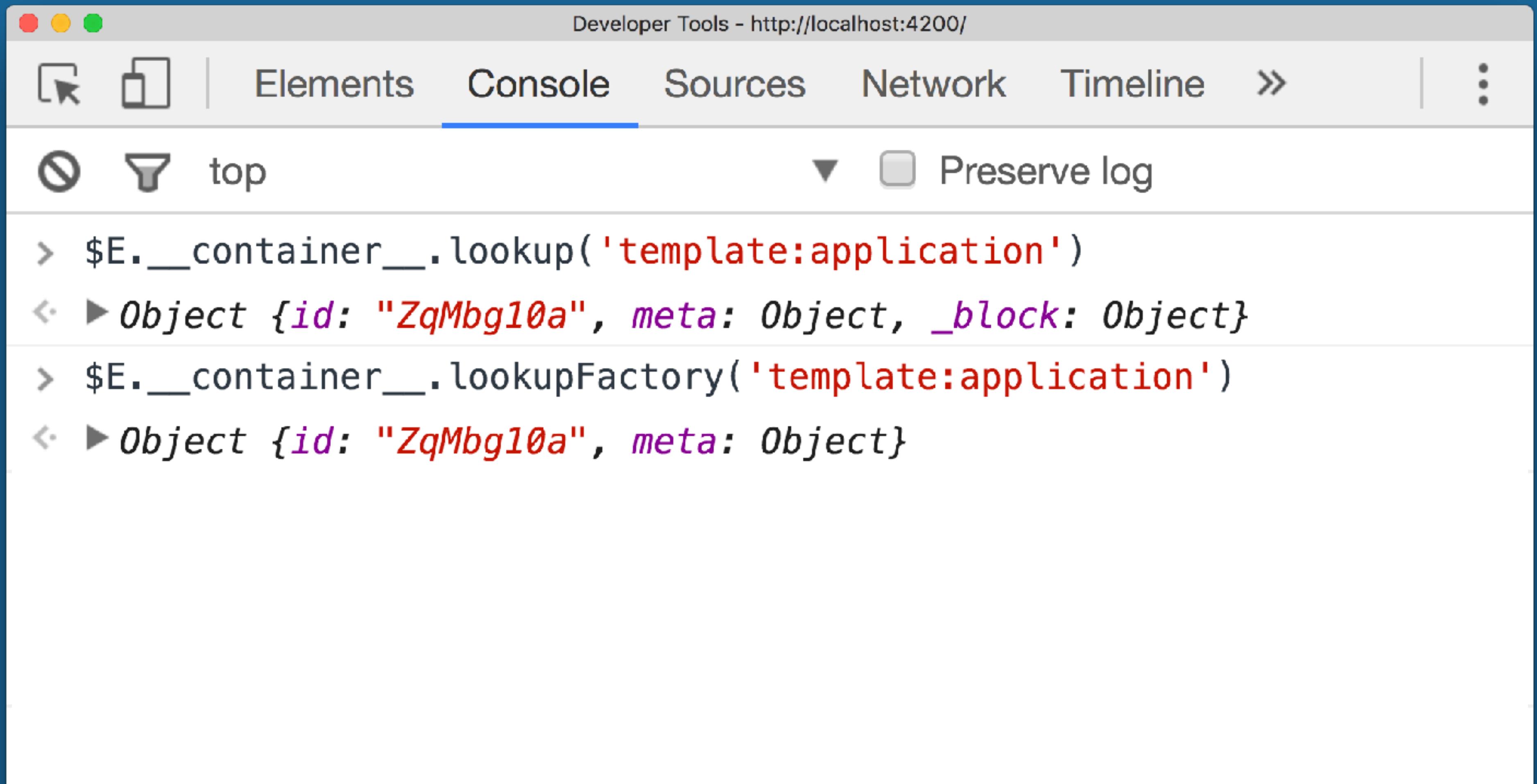


The screenshot shows the Chrome Developer Tools interface with the title "Developer Tools - http://localhost:4200/" at the top. The "Console" tab is active, indicated by a blue underline. Below the tabs, there are filter icons: a stop sign for "No Errors" and a funnel for "top". To the right of these filters is a dropdown arrow and a checkbox labeled "Preserve log". The main area displays the output of the Ember registry:

```
> $E.resolveRegistration('service:store')
< (subclass of Ember.Service)
> $E.registeredOptionsForType('component')
< ► Object {singleton: false}
> |
```

The output uses syntax highlighting where "service:store" and "component" are in red, and "singleton" is in purple.

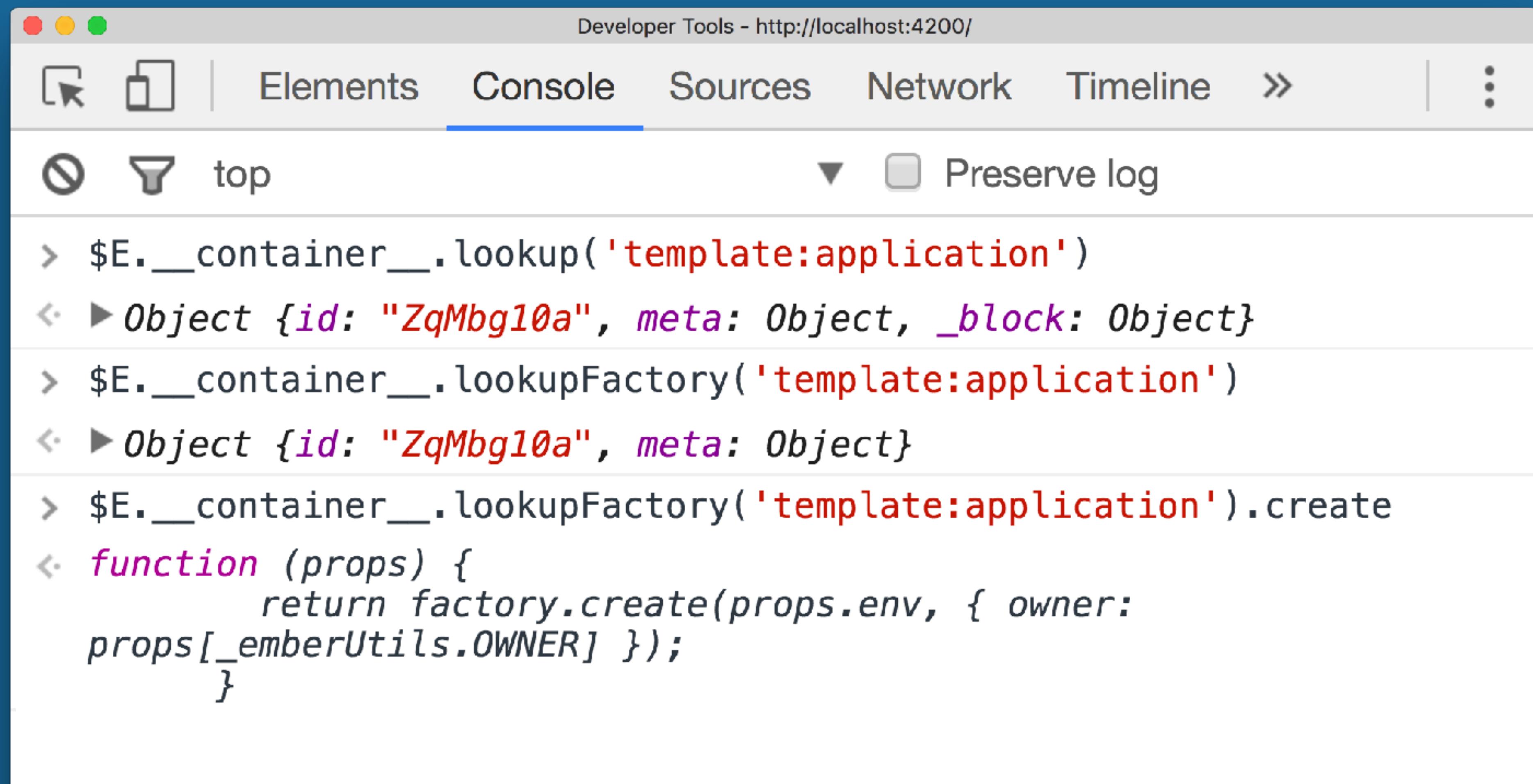
Debugging: Container



The screenshot shows the Chrome Developer Tools interface with the 'Console' tab selected. The title bar reads 'Developer Tools - http://localhost:4200/'. Below the tabs, there are filters: a 'no errors' icon, a 'filter' icon, and a dropdown set to 'top'. A checkbox for 'Preserve log' is also present. The main area displays the following log entries:

```
> $E.__container__.lookup('template:application')
< ► Object {id: "ZqMbg10a", meta: Object, _block: Object}
> $E.__container__.lookupFactory('template:application')
< ► Object {id: "ZqMbg10a", meta: Object}
```

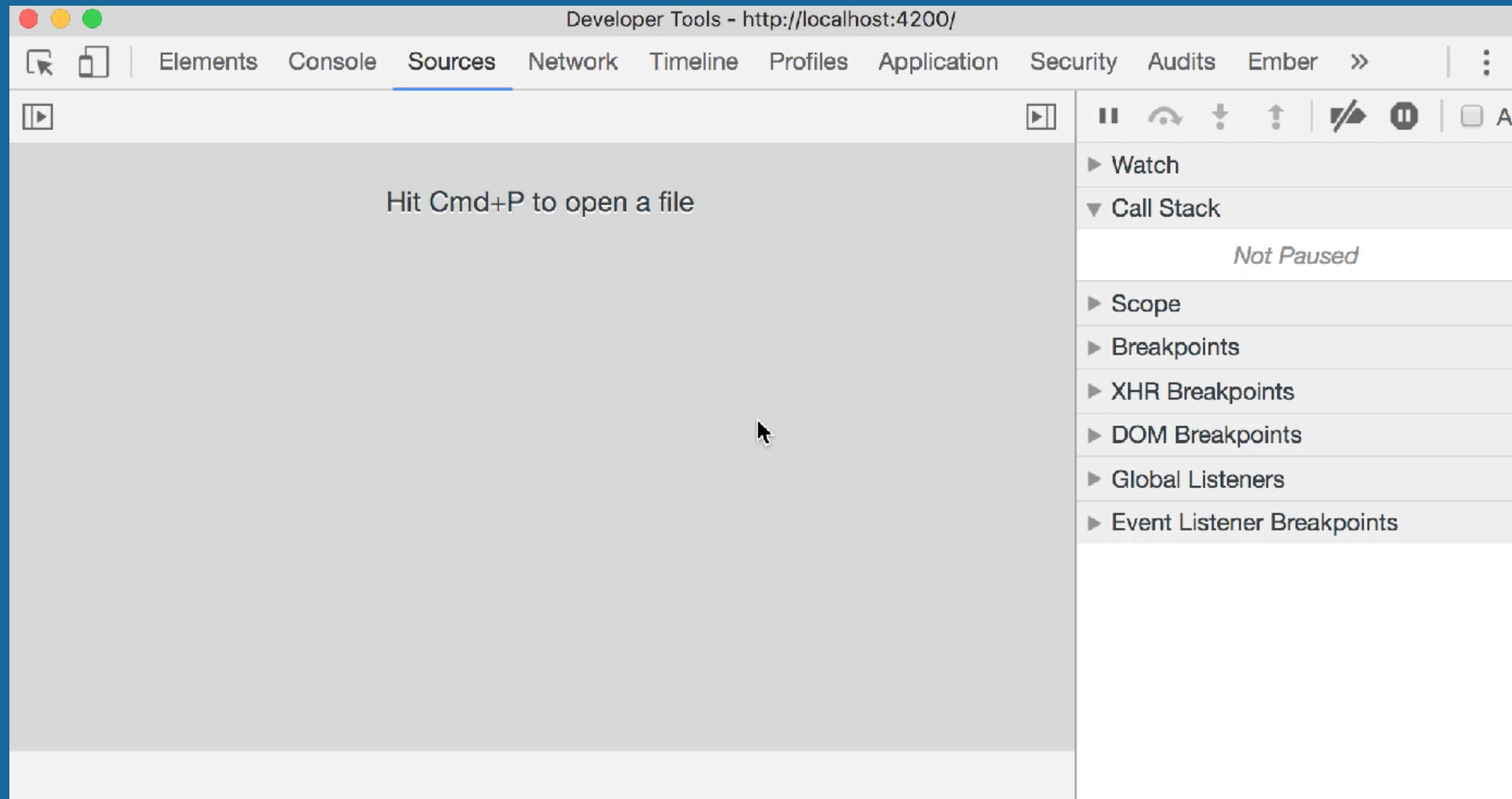
Debugging: Container



The screenshot shows the Chrome Developer Tools interface with the 'Console' tab selected. The title bar reads 'Developer Tools - http://localhost:4200/'. Below the tabs, there are filters for 'top' and a checked 'Preserve log' option. The main area displays a stack trace:

```
> $E.__container__.lookup('template:application')
< ► Object {id: "ZqMbg10a", meta: Object, _block: Object}
> $E.__container__.lookupFactory('template:application')
< ► Object {id: "ZqMbg10a", meta: Object}
> $E.__container__.lookupFactory('template:application').create
< function (props) {
    return factory.create(props.env, { owner:
  props[_emberUtils.OWNER] });
}
```

Debugging: Initializers



Debugging: Initializers

The screenshot shows the Chrome Developer Tools interface with the 'Sources' tab selected. The left pane displays the code for `startup.js`, which contains an initializer definition. The line `debugger;` is highlighted with a blue selection bar. The right pane shows the 'Call Stack' panel, which lists the stack trace of the current execution. The stack trace starts with the `initialize` function at `startup.js:5` and includes several layers of Ember framework code, such as `ember.debug.js` and `ember.debug.js:21619`.

```
1 define('glue/initializers/startup', ['exports'], function (exports) {
2   exports.initialize = initialize;
3
4   function initialize() /* application */{
5     debugger;
6     // application.inject('route', 'foo', 'service:foo');
7   }
8
9   exports['default'] = {
10     name: 'startup',
11     initialize: initialize
12   };
13});
```

{ } Line 5, Column 1 (source mapped from glue.js)

Developer Tools - http://localhost:4200/

Elements Console Sources Network Timeline Profiles Application Security Audits Ember > :

▶ Watch ▶ Call Stack → initialize startup.js:5
(anonymous) ember.debug.js:4564
Vertices.each ember.debug.js:2630
Vertices.topso ember.debug.js:2597
rt
DAG.topsort ember.debug.js:2542
_runInitializer ember.debug.js:4592
runInitializers ember.debug.js:4553
_bootSync ember.debug.js:3762
domReady ember.debug.js:3663
run ember.debug.js:720
join ember.debug.js:746
run.join ember.debug.js:21556
run.bind ember.debug.js:21619

Debugging: Instance Initializers

The screenshot shows the Chrome Developer Tools Sources tab for the file `ember.debug.js`. The code is part of the `_runInitializer` function, which takes a `bucketName` and a callback `cb`. The code initializes a `graph` and adds two initializers, "ember-data" and "startup", to it. The current line of code is `graph.topsort(cb);`. A tooltip on the right side of the screen provides a detailed view of the `initializers` variable, showing its value as an array with two elements: "ember-data" at index 0 and "startup" at index 1. The `__proto__` property is also shown as an empty array. The Call Stack panel on the right lists the stack frames from `graph.topsort` back up to the `_runInitializer` call.

```
Developer Tools - http://localhost:4200/
Elements Console Sources Network Timeline Profiles Application Security Audits Ember > ...
▶ startup.js ember.debug.js x
4580
4581     _runInitializer: function (bucketName, cb) { bucketName = "instanceIn
4582         var initializersByName = _emberMetal.get(this.constructor, bucketNam
4583         var initializers = props(initializersByName); initializers = ["embe
4584         var graph = new _dagMap.default(); graph = DAG {vertices: Vertices
4585         var initializer = undefined; initializer = Object {name: "startu
4586
4587             for (var i = 0; i < initializers.length; i++) { i = 2, initializers
4588                 initializer = initializersByName[initializers[i]]; initializ
4589                 graph.add(initializer)
4590             }
4591
4592             graph.topsort(cb);
4593         }
4594     });
4595
4596     Engine.reopenClass({
4597         initializers: new _e
4598         instanceInitializers
4599
4600         /**
4601             The goal of initia
4602             This phase runs on
4603             allowed to defer a
4604
4605     {} Line 4592, Column 1 (source)
```

▶ Watch
▼ Call Stack

- initialize startup.js:5
- (anonymous) ember.debug.js:4577
- Vertices.each ember.debug.js:2630
- Vertices.topso ember.debug.js:2597
- DAG.topsort ember.debug.js:2542
- ▶ _runInitializer ember.debug.js:4592
- runInstanceIni ember.debug.js:4575
- tializers
- _bootSync ember.debug.js:2848
- didBecomeRe ember.debug.js:3876
- ady
- invoke ember.debug.js:339
- flush ember.debug.js:407

Slow-lo-mo

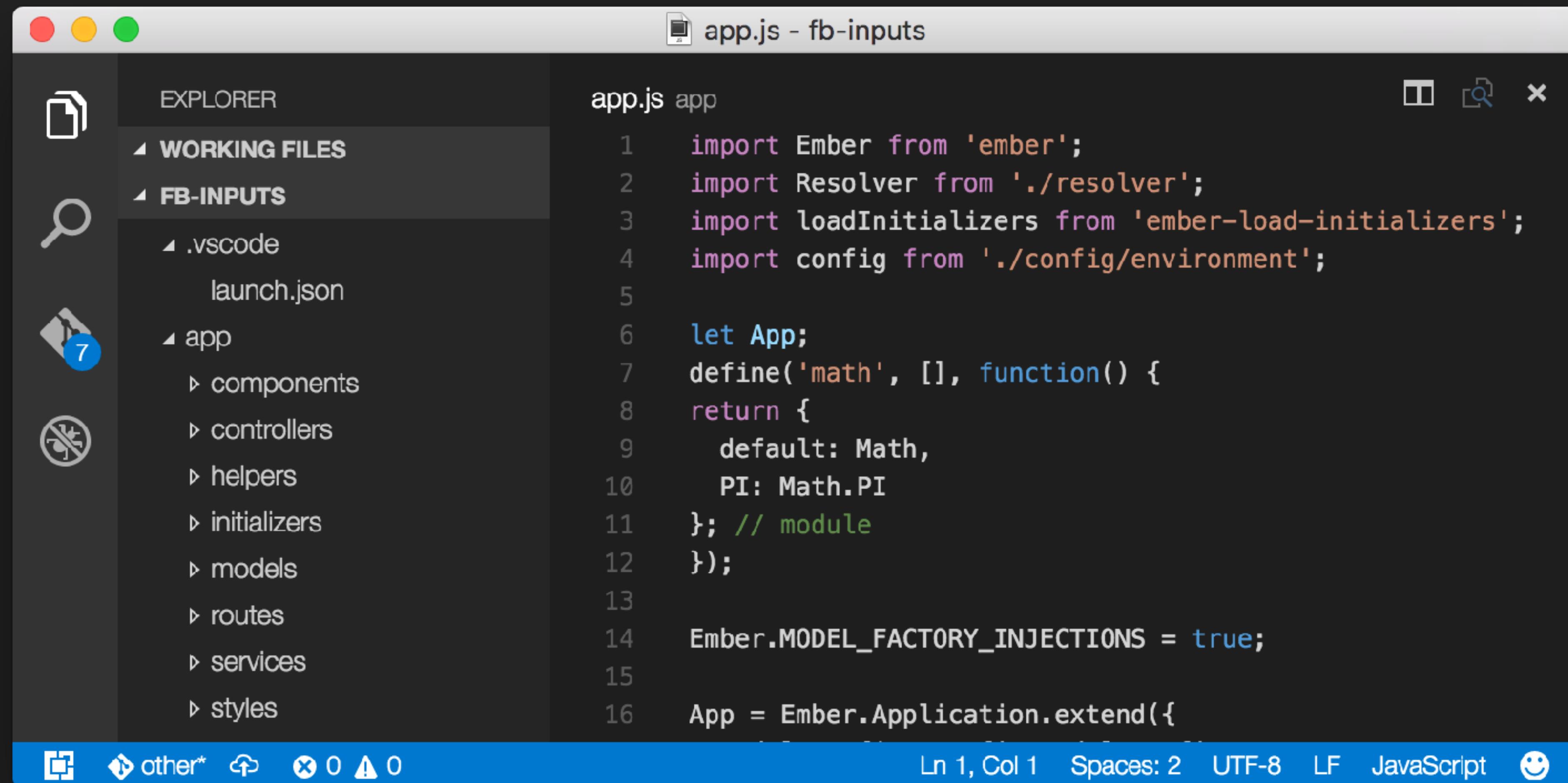
- ▶ Request access to geolocation data, and ensure your app doesn't start up until you have it
- ▶ Store the location data in your container, using the key "data:location" and value {lat: ..., lng: ...}
- ▶ In your application route, give the controller access to the location data, so that it can be rendered in the application.hbs template
- ▶ ensure that things are working, using ember inspector

```
const { geolocation } = navigator;
geolocation.getCurrentPosition((pos) => {
  let {
    coords: { latitude, longitude }
  } = pos;
  do_something(latitude, longitude);
});
```



Building

Debugging Node.js With VS Code



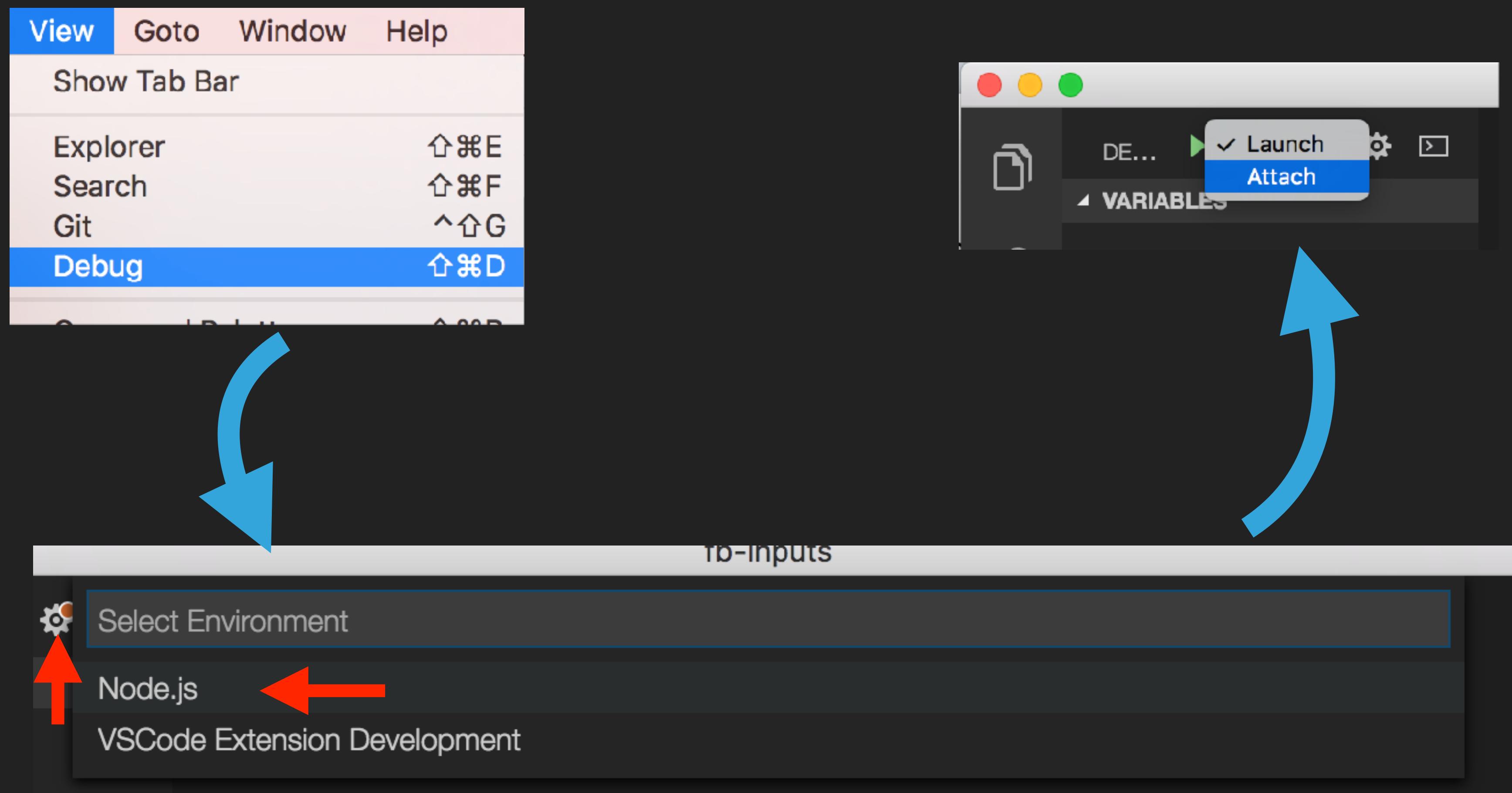
The screenshot shows the Visual Studio Code interface. The title bar says "app.js - fb-inputs". The left sidebar is the Explorer, showing a tree view of the project structure under "WORKING FILES". The "FB-INPUTS" folder is expanded, showing subfolders ".vscode", "app" (which contains "components", "controllers", "helpers", "initializers", "models", "routes", "services", and "styles"), and other files like "launch.json". There are 7 untracked files indicated by a blue circle with a '7' in the Explorer icon. The main editor area shows the content of "app.js". The code is as follows:

```
1 import Ember from 'ember';
2 import Resolver from './resolver';
3 import loadInitializers from 'ember-load-initializers';
4 import config from './config/environment';

5
6 let App;
7 define('math', [], function() {
8   return {
9     default: Math,
10    PI: Math.PI
11  }; // module
12 });
13
14 Ember.MODEL_FACTORY_INJECTIONS = true;
15
16 App = Ember.Application.extend({
```

The status bar at the bottom shows "Ln 1, Col 1" and "Spaces: 2" and "JavaScript".

Debugging Node.js With VS Code



Debugging Node.js With VS Code

The screenshot shows the VS Code interface during a Node.js debugging session. The title bar indicates the file is `index.js - fb-inputs`. The left sidebar contains the DEBUG, VARIABLES, WATCH, CALL STACK, and BREAKPOINTS sections. The VARIABLES section shows a UI object with properties like `length` and `name`, and a prototype object with methods like `constructor`, `prependLine`, and `prompt`. The CALL STACK section shows a stack trace with `resolve` at path.js:1138 and `module.exports` at index.js:90. The BREAKPOINTS section lists several breakpoints, with three checked: `All Exceptions`, `Uncaught Exceptions`, `ember` at node_modules/ember-cli/bin, and `index.js` at line 84. The main editor area displays code from node_modules/ember-cli/lib/cli/index.js, specifically lines 86 through 94. A green dot on line 90 indicates the current execution step. The DEBUG CONSOLE panel shows the global `Object` object with various static methods. The status bar at the bottom provides information about the current line (Ln 90, Col 1), character encoding (UTF-8), and file type (JavaScript).

```
node_modules/ember-cli/lib/cli/index.js - fb-inputs
```

UI:

```
function UI(options) { ... }
  length: 1
  name: "UI"
prototype:
```

constructor: function UI(options) { ... }
prependLine: function (prependData, data, prepend) { ... }
prompt: function (questions, callback) { ... }

VARIABLES

WATCH

CALL STACK

PAUSED ON STEP

resolve

path.js 1138

module.exports

index.js 90

BREAKPOINTS

- All Exceptions
- Uncaught Exceptions
- ember 26 node_modules/ember-cli/bin
- index.js 84 node_modules/ember-cli/lib/cli

DEBUG CONSOLE

```
Object:
  _makeLong: function _makeLong(path) { ... }
  basename: function basename(path, ext) { ... }
  delimiter: ":"
  dirname: function dirname(path) { ... }
  extname: function extname(path) { ... }
  format: function format(pathObject) { ... }
  isAbsolute: function isAbsolute(path) { ... }
  join: function join() { ... }
  name: "Path"
```

Ln 90, Col 1 Spaces: 2 UTF-8 LF JavaScript

Broccoli - A Fast Asset Pipeline

- ▶ Built on node's fs module
- ▶ Cache intermediate build results
- ▶ Think trees, not files
- ▶ Plugins are chainable



Trees & Plugins

- ▶ A string is a tree
- ▶ Plugins are either N:N or N:1
- ▶ TIP: Act like trees are immutable



```
var appTree = app.toTree();  
minify(appTree)  
return appTree;
```



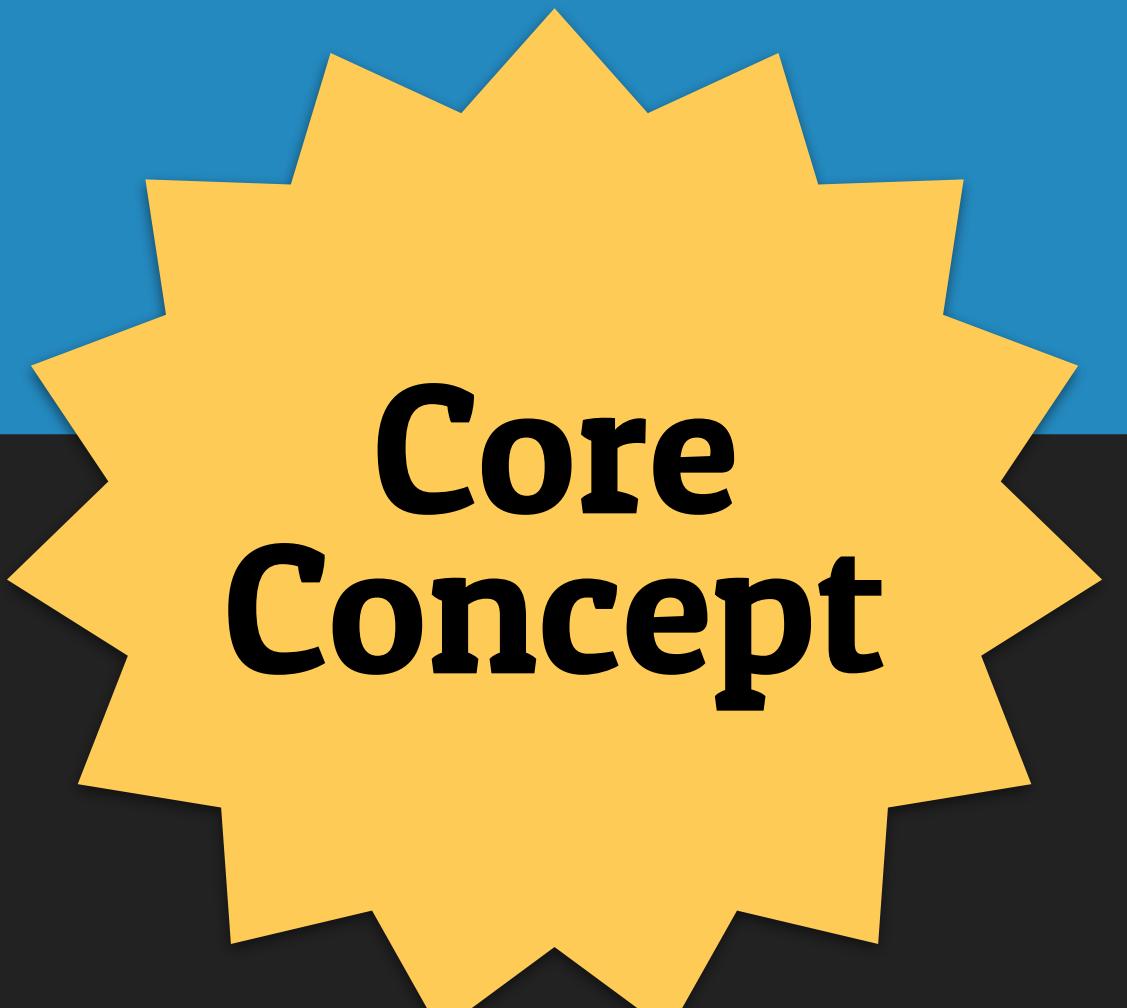
```
return new Minify(app.toTree());
```

Example Plugin:

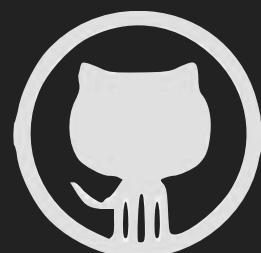
```
function MyFilter(inputNode) {  
  Filter.call(this, inputNode);  
}  
  
MyFilter.prototype = Object.create(Filter.prototype);  
  
MyFilter.prototype.processString = function(existingString) {  
  return 'newString';  
};  
  
MyFilter.prototype.extensions = ['coffee'];  
MyFilter.prototype.targetExtension = 'js';
```

Debugging Broccoli Chains

You're assembling a chain of functions, evaluated much later.



**Core
Concept**



STEFANPENNER/BROCCOLI-STEW

- `env` - Conditionally runs a callback based upon the current environment.
- `mv` - Moves an input tree to a different location.
- `rename` - Renames files in a tree.
- `find` - Match files in a tree.
- `map` - Maps files, allow for simple content mutation.
- `log` - Logs out files in the passed tree.
- `debug` - Writes the passed tree to disk at the root of the project.
- `rm` - Remove files from a tree.
- `beforeBuild` - Calls a callback function before the tree is read.
- `afterBuild` - Calls a callback function after the tree is read.
- `wrapBuild` - Calls callback functions before and after the tree is read.
- `npm.main` - Create a tree with Node module's main entry



Debugging Broccoli

- ▶ `npm install -save-dev broccoli-stew`

```
var debugTree = require('broccoli-stew').debug;

return new MyPlugin(
  debugTree(
    app.toTree(), {name: 'my-tree'} // Write to this folder
  )
);
```

Cooking Up Some



3

- ▶ I want a copyright notice at the top of each css/js file, listing 🦄🦄🦄🔫🌈🍺🍺 as the author and copyright holder.
- ▶ The comment should also include a timestamp so we know when these assets were built.
- ▶ Remove ember-cli-sri from package.json

```
/**  
 * vendor.js  
 *  
 * (c) 2016 🦄🦄🦄🔫🌈🍺🍺 All Rights Reserved  
 * generated at: 2016-12-06T19:20:23.171Z  
 */
```

Building & Deploying - Ember-App Options

- ▶ Many broccoli plugins act on options passed to the EmberApp constructor
- ▶ If you run ember build --prod, your files will be fingerprinted
- ▶ If we want a staging environment, we'll need to customize things a bit.

Ember-App Options

```
var app = new EmberApp({
  fingerprint: {
    exclude: ['fonts/169929'],
    prepend: 'https://sudomain.cloudfront.net/'
  },
  minifyJS: {
    enabled: false
  },
  minifyCSS: {
    enabled: false
  }
});
```

Per-Environment Configuration

- ▶ `config/environment.js` is where most per-env switching takes place
- ▶ baked into the build, via meta tag
- ▶ module is available for client and server-side stuff
- ▶ many non-build-related ember addons will be customized in this file

Automated Testing

- ▶ When building/testing, it's just like any node app
- ▶ Already set up for travis-ci w/ PhantomJS 2.1
- ▶ Addons are easily tested against multiple sets of npm/bower dependencies

Travis-CI

Test with Chrome instead of PhantomJS

```
ember g travis-chrome
```

Circle-CI

Test with Phantomjs 2.1

```
ember g circle-ci-phantom
```

Test with headless Chrome

```
ember g circle-ci-chrome
```

Heroku Deployment

```
# create a new app
heroku create
# setup the ember buildpack
heroku buildpacks:set \
  https://codon-buildpacks.s3.amazonaws.com/buildpacks/heroku/emberjs.tgz
# push to the heroku git remote
git push heroku master
```

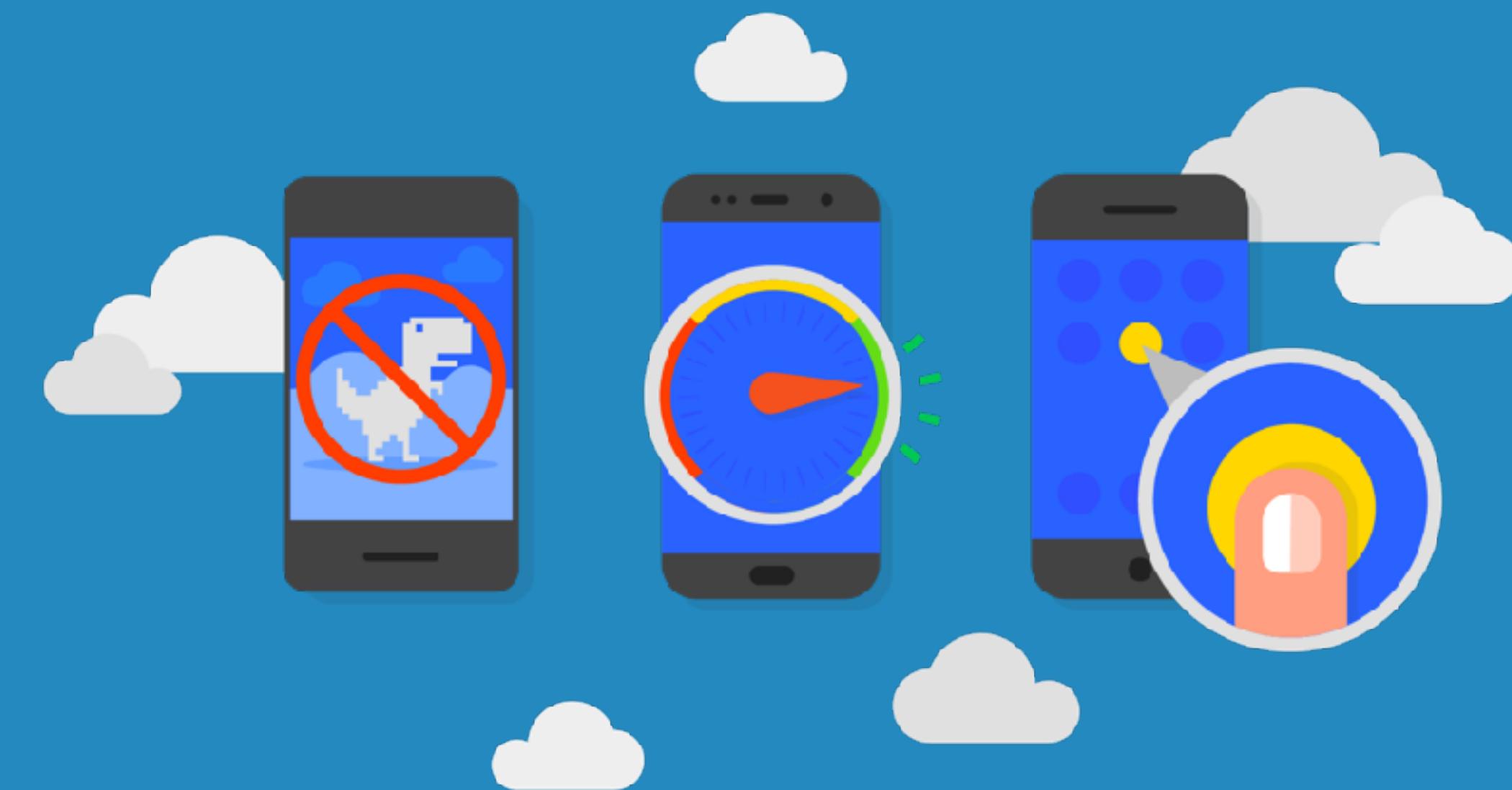
Deploy to Staging

- ▶ We know how to test with travis-ci and deploy to heroku. Let's put it all together! Push a prod-like build to staging

- ▶ HINT: you can install the travis gem and run

```
travis setup heroku
```

- ▶ Make sure anything secret (i.e., your Heroku API keys) are encrypted
- ▶ In the end, you should be able to push to GitHub, and test-passing code will deploy w/o human intervention

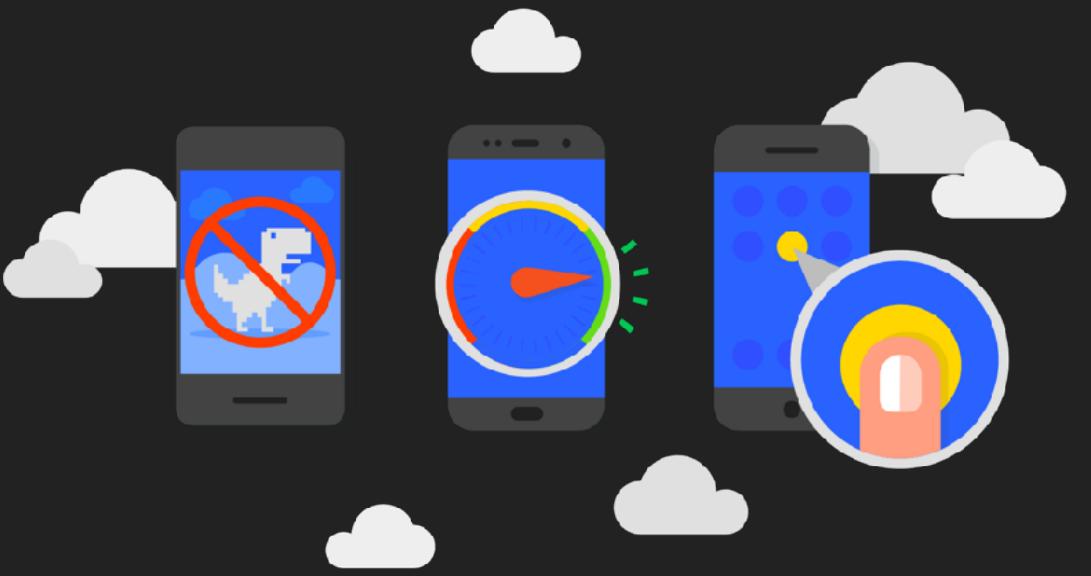


Fastboot & PWA

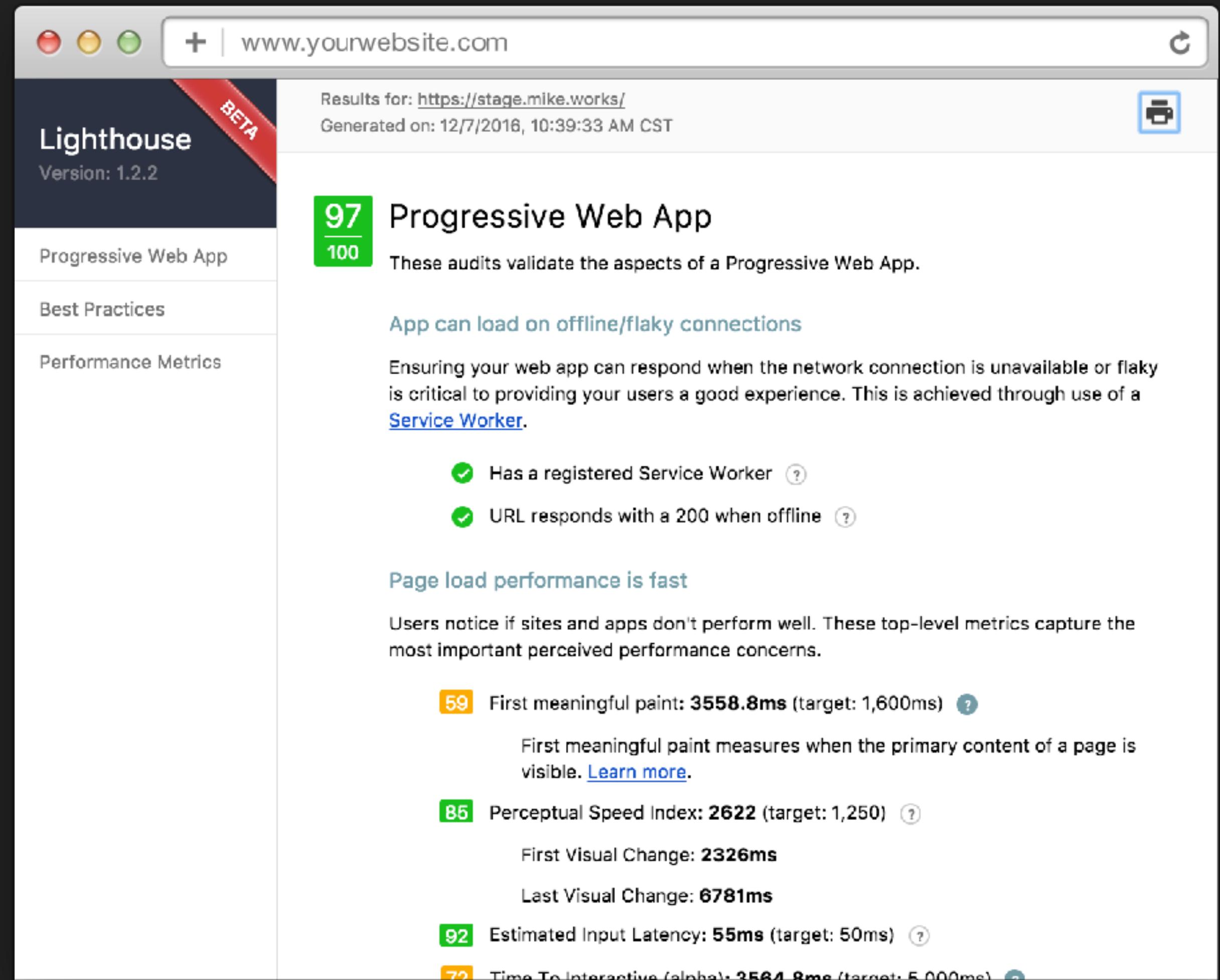
**Normal apps, that become
superheroes where possible**

Progressive Web App Technologies

- ▶ Time to first meaningful paint
- ▶ Time to first interaction
- ▶ Mobile web metadata (theme color, browser UI removal, add to home screen, etc...)
- ▶ Resiliency to network quality via service worker
- ▶ SEO + Structured Data



Progressive Web App



The screenshot shows the Lighthouse audit results for the URL <https://stage.mike.works/>. The audit was generated on December 7, 2016, at 10:39:33 AM CST. The overall score is 97 out of 100, categorized under "Progressive Web App".

Progressive Web App (97/100)

These audits validate the aspects of a Progressive Web App.

App can load on offline/flaky connections

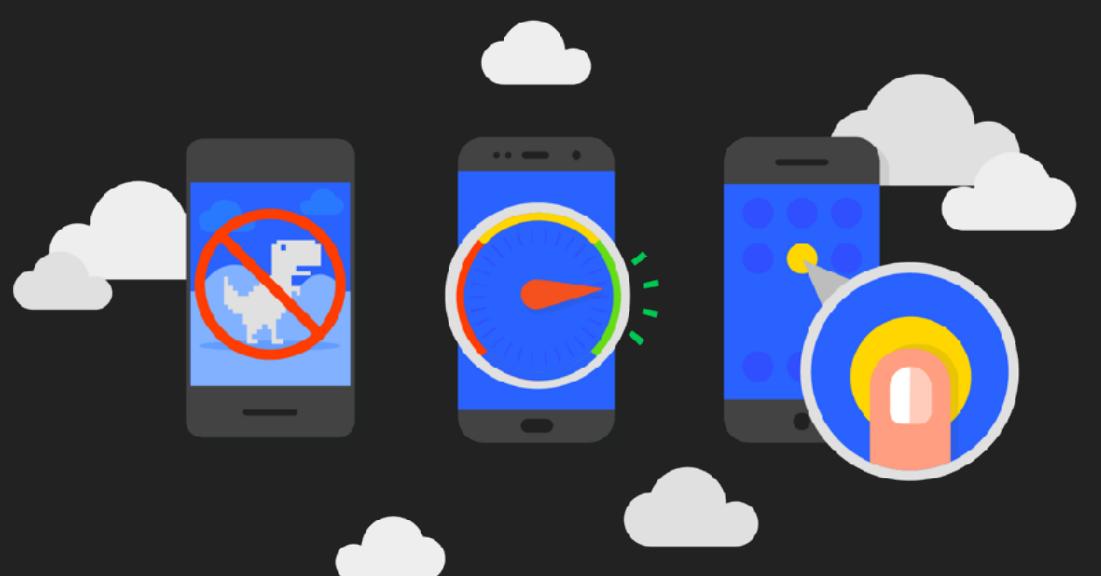
Ensuring your web app can respond when the network connection is unavailable or flaky is critical to providing your users a good experience. This is achieved through use of a [Service Worker](#).

- Has a registered Service Worker
- URL responds with a 200 when offline

Page load performance is fast

Users notice if sites and apps don't perform well. These top-level metrics capture the most important perceived performance concerns.

- First meaningful paint: **3558.8ms** (target: 1,600ms)
- First meaningful paint measures when the primary content of a page is visible. [Learn more](#).
- Perceptual Speed Index: **2622** (target: 1,250)
 - First Visual Change: **2326ms**
 - Last Visual Change: **6781ms**
- Estimated Input Latency: **55ms** (target: 50ms)
- Time To Interactive (alpha): **2561.8ms** (target: 5,000ms)



Developer Tools - chrome://settings/help

Elements Console Sources Network Performance Memory Application Security **Audits**

+ X | ⚡



Audits help you identify and fix common problems that affect your site's performance, accessibility, and user experience. [Learn more](#)

Perform an audit...

⋮ Console What's New X

Highlights from Chrome 59 update

CSS and JS code coverage

Find unused CSS and JS with the new Coverage drawer.

URL	Type	Total Bytes	Unused Bytes	% Unused
/script_foot_closure	JS	385 963	265 341	68.2 %
/query_u-bundle	JS	241 682	217 071	89.8 %
ht.../script_foot.js	JS	231 291	156 748	67.8 %
https://rawgit.com	JS	185 863	122 783	66.1 %

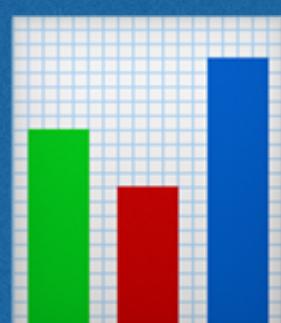
Client Rendered SPA



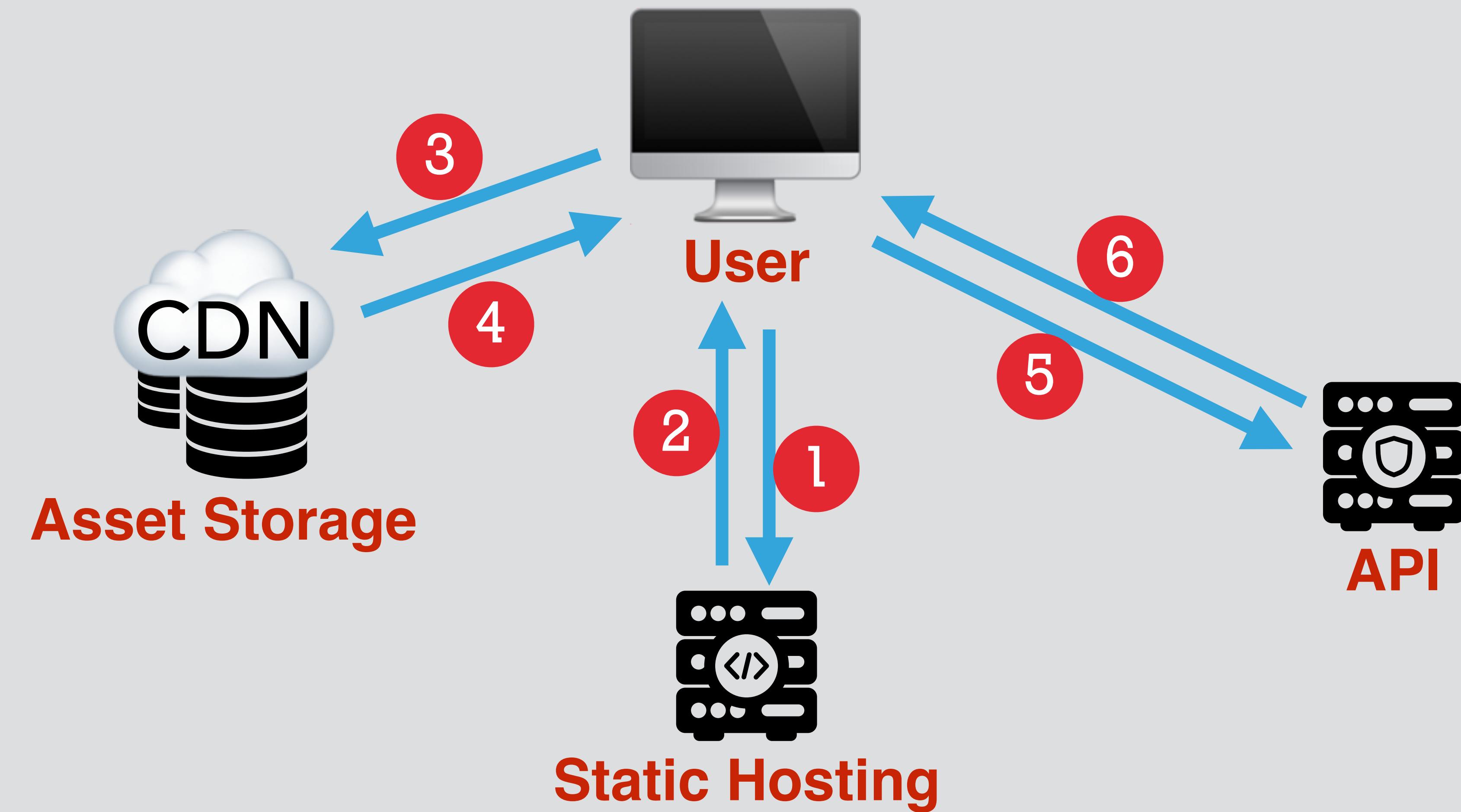
First Paint



Interactive



Data



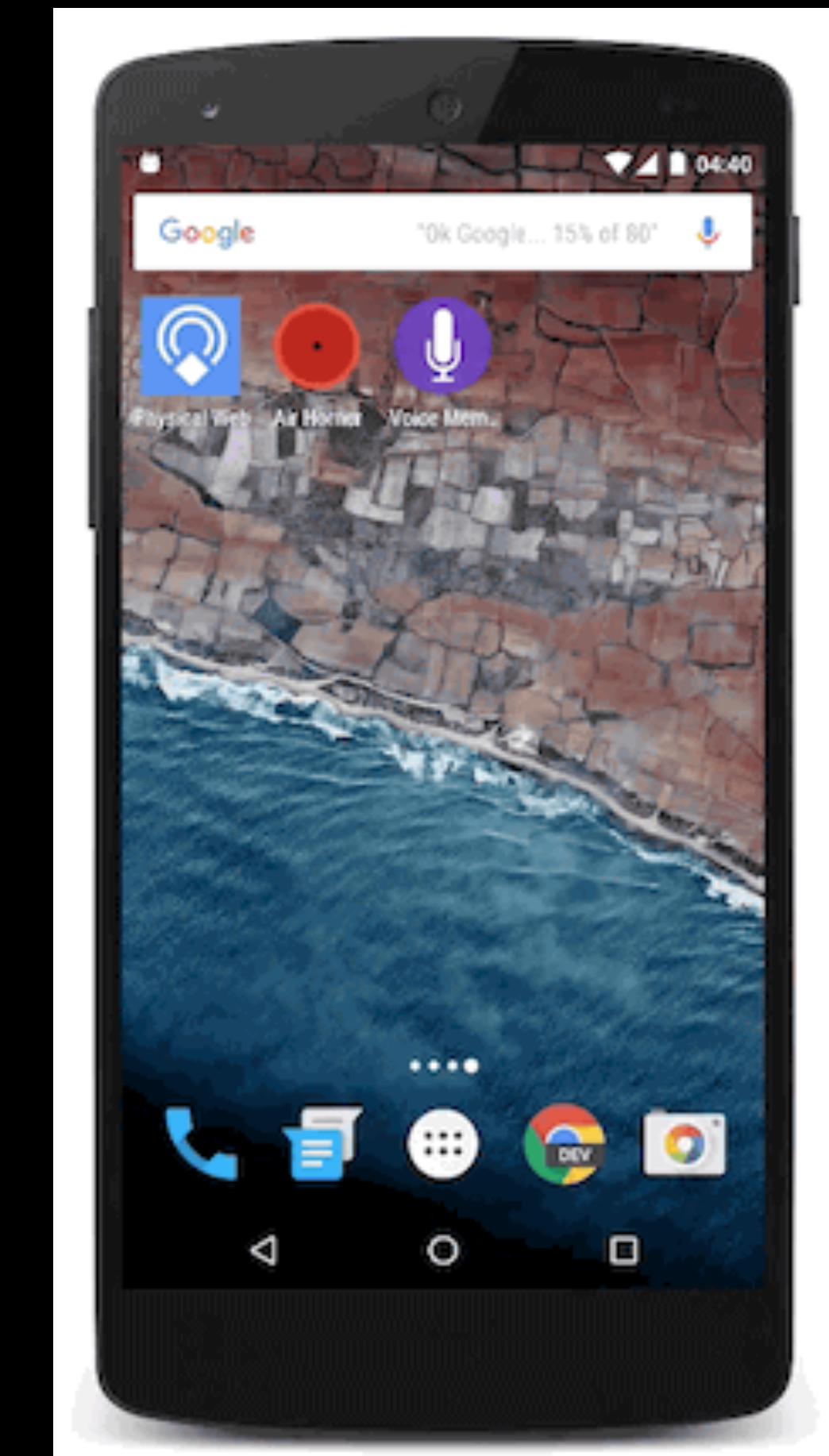
Major Pieces

App-Like Characteristics

Pruning the Critical Path

Tolerating Network Instability

App-Like Characteristics

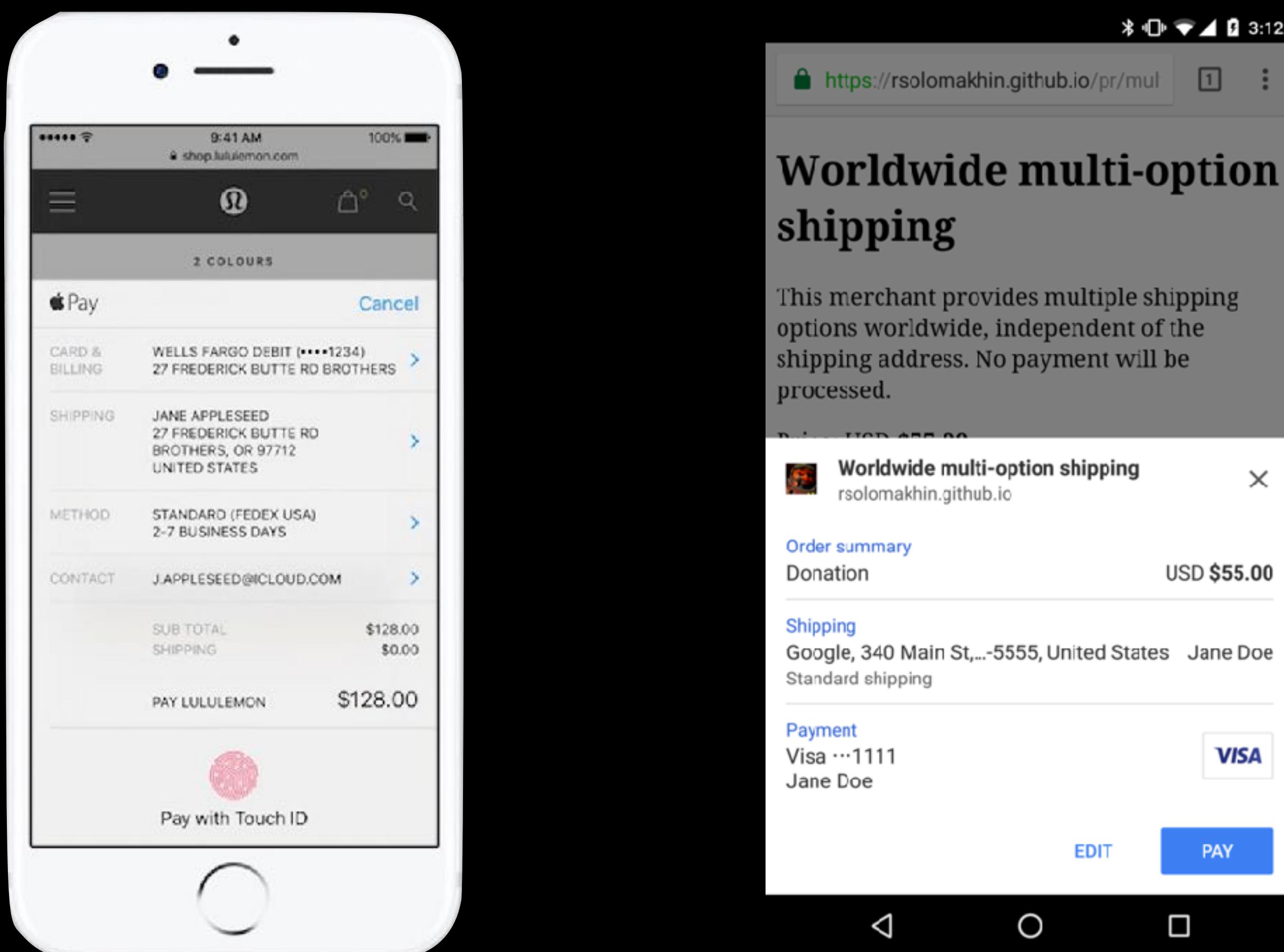


App-Like Characteristics

manifest.json

```
{  
  "name": "Mike.Works - World Class Developer Training",  
  "short_name": "Mike.Works",  
  "start_url": ".",  
  "display": "standalone",  
  "background_color": "#fff",  
  "theme_color": "#fff",  
  "description": "World Class Developer Training",  
  "orientation": "portrait-primary",  
  "lang": "en-US",  
  "icons": [ ]  
};
```

App-Like Characteristics



In The Future...

-  Push Notifications
-  Background Sync
-  First-Class Framework Support
-  IndexedDB 2.0
-  HTTP/2 Optimizations

App-Like Characteristics

Get rid of browser UI

```
<meta name="apple-mobile-web-app-capable" content="yes">
```

App Icons

```
<link sizes="180×180" rel="apple-touch-icon" href="icon.png">
```

Bookmark Name

```
<meta name="mobile-web-app-title" content="Mike.Works">
```

Splash Screen

```
<link href="splash.png" rel="apple-touch-startup-image">
```

TOOLS DEMO

Looks Like an App

```
ember install ember-cli-head  
ember install ember-cli-meta-tags  
ember install ember-add-to-homescreen
```

- ▶ Open image-assets.zip, and find a bunch of icons
- ▶ Ensure the add-to-homescreen behavior works as expected, and presents appropriate icon
- ▶ Strip away all browser UI you can for bookmarked launch
- ▶ Zooming behavior should match native apps
- ▶ Create a manifest.json, and follow all lighthouse advice

Major Pieces

App-Like Characteristics

Pruning the Critical Path

Tolerating Network Instability

Pruning the Critical Path

Important Metrics

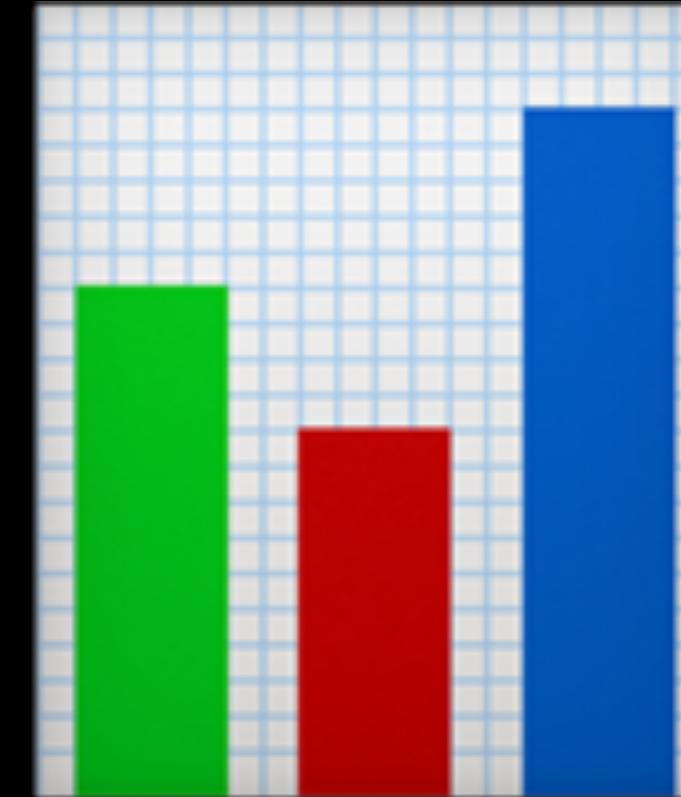
Time to...



First Paint



Interactive



Data

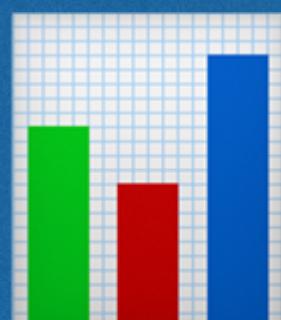
Client Rendered SPA



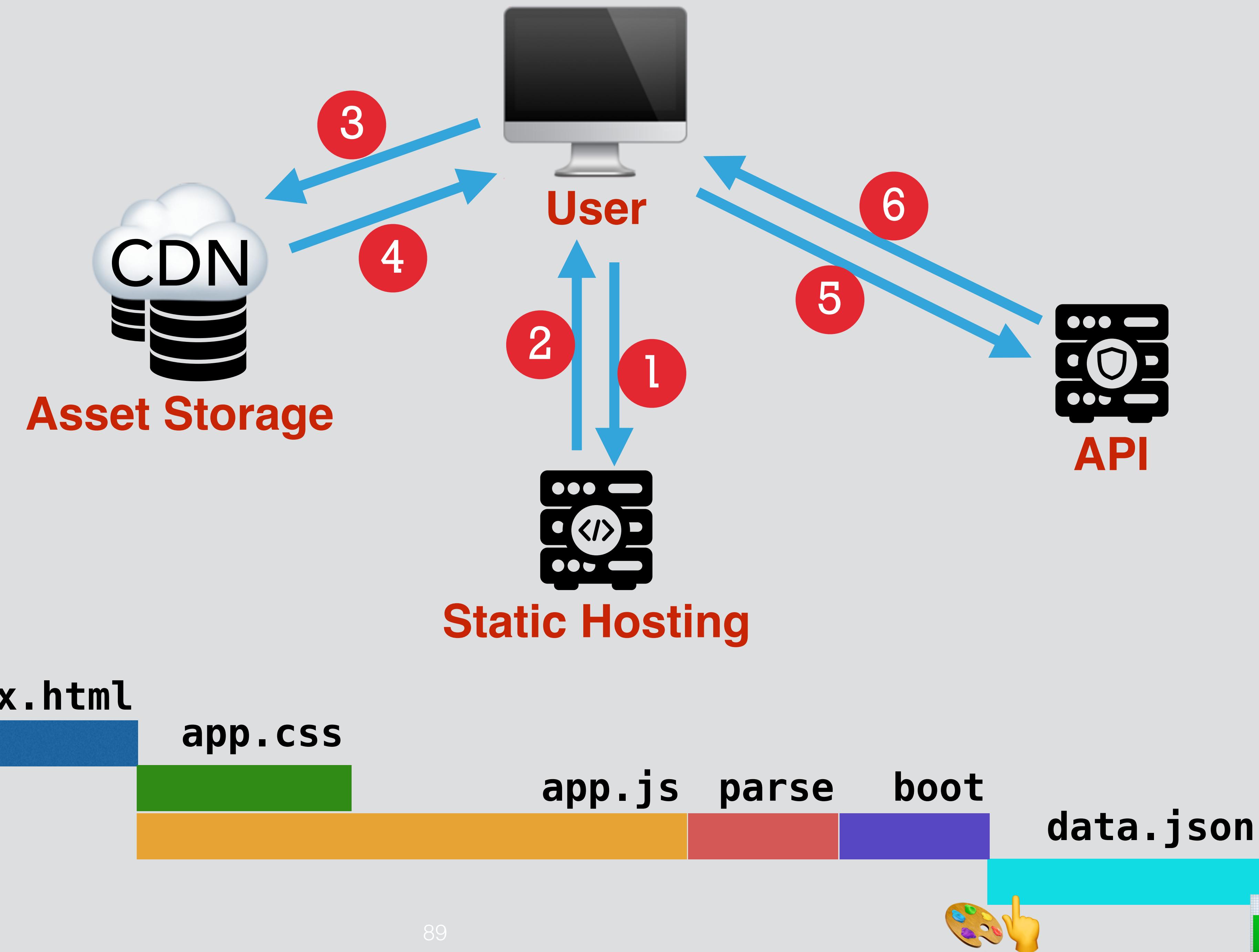
First Paint



Interactive



Data



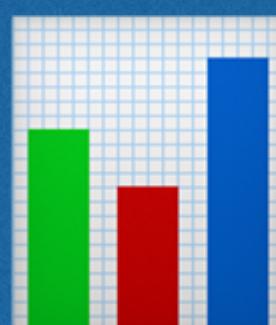
Server Rendered SPA



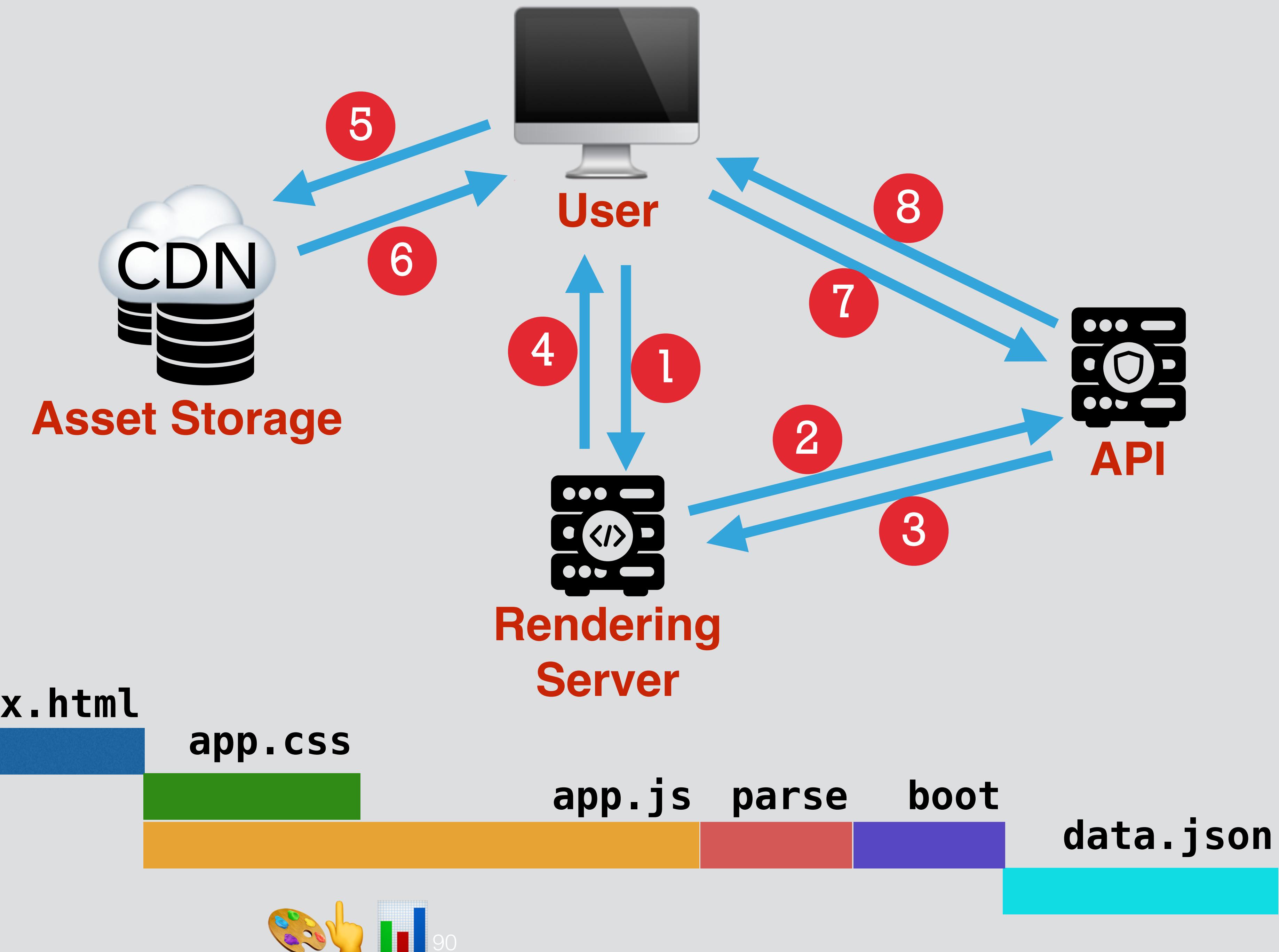
First Paint



Interactive



Data



Server-Side Rendering

- ⏳ Don't have to wait for client-side JS
- 🔗 Better for SEO & sharing links
- ⚠️ Node.js and the Browser are different
- 🤗 Try not to do the same work twice
- ⚡ HTML is assembled in your data center

Setting up Fastboot

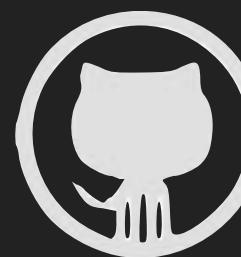
- ▶ Consumable as an ember addon

```
ember install ember-cli-fastboot
```

- ▶ Runnable on the cli

```
ember serve
```

- ▶ In production, using a node app



EMBER-FASTBOOT/FASTBOOT-APP-SERVER

Differences in Capabilities

- ▶ Simple-DOM vs browser-DOM
- ▶ FastBoot is the only global

Practical Consequences

- ▶ Libs that depend on full DOM implementation: 
- ▶ Common practice: replace a browser lib with a node one
- ▶ Use fetch instead of \$.ajax

Guards

Running {

```
if (typeof FastBoot === 'undefined') {  
    🎨 BROWSER CLIENT-SIDE STUFF  
} else {  
    🛠 FASTBOOT CLIENT-SIDE STUFF  
}
```

Building {

```
if (!process.env.EMBER_CLI_FASTBOOT) {  
    🎨 BUILD-FOR-BROWSER STUFF  
} else {  
    🛠 BUILD-FOR-FASTBOOT STUFF  
}
```

Guards

Running {

```
if (typeof FastBoot === 'undefined') {  
    🎨 BROWSER CLIENT-SIDE STUFF  
} else {  
    🛠 FASTBOOT CLIENT-SIDE STUFF  
}
```

Building {

```
treeForFastBoot: function(tree) {  
    🛠 BUILD-FOR-FASTBOOT STUFF  
}
```

Render with Fastboot

- ▶ Install ember-cli-fastboot
- ▶ Run fastboot locally, serving assets
- ▶ If you find anything that's unavoidably browser-specific, see if you can just skip it when rendering on the server side
- ▶ Push to heroku (bonus points for pushing to github, and letting travis deploy it for you)

Request Data in Fastboot

- ▶ Unlike in the static hosting setup, we have access to a request!

```
export default Ember.Route.extend({
  fastboot: Ember.inject.service(),
  model() {
    let headers = this.get('fastboot.request.headers');
    let xRequestHeader = headers.get('X-Request');
    // ...
  }
});
```

Handing off Data

- ▶ You may notice double requests to your API
- ▶ Rehydration - not 100% automated yet
- ▶ We have a useful tool: shoebox



Shoebox: Concept

- ▶ Get data from API using fetch
- ▶ Store it in the shoebox (node: memory)
- ▶ HTML is rendered, shoebox → meta tag
- ▶ Browser app starts up, and can get shoebox data

```
<script type="fastboot/shoebox" id="shoebox-main-store">
{"data": [
  {"attributes": {"name": "AEC Professionals"}, "id": 106, "type": "audience"},
  {"attributes": {"name": "Components"}, "id": 111, "type": "audience"},
]}
</script>
```

```
// Get the shoebox, and the shoebox store we want
let shoebox = this.get('fastboot.shoebox');
let shoeboxStore = shoebox.retrieve('my-store');

if (this.get('fastboot.isFastBoot')) {

}
```

```
// Get the shoebox, and the shoebox store we want
let shoebox = this.get('fastboot.shoebox');
let shoeboxStore = shoebox.retrieve('my-store');

if (this.get('fastboot.isFastBoot')) {
  // If rendering on fastboot server, make the request
  return this.store.findRecord('post', params.post_id)
    .then(post => {

  }) );
}
```

```
// Get the shoebox, and the shoebox store we want
let shoebox = this.get('fastboot.shoebox');
let shoeboxStore = shoebox.retrieve('my-store');

if (this.get('fastboot.isFastBoot')) {
  // If rendering on fastboot server, make the request
  return this.store.findRecord('post', params.post_id)
    .then(post => {
      if (!shoeboxStore) { // Lazily create the store
        shoeboxStore = {};
        shoebox.put('my-store', shoeboxStore);
      }
      // Put the data in the store
      shoeboxStore[post.id] = post.toJSON();
    });
}
```

Server Data, on Client

- ▶ Grab the user agent of the incoming request to index.html, and make it available in the container under the container key

"**data:request**"

- ▶ Prove that your solution works, by render this data in a template somewhere

```
$E.__container__.lookup('data:request')  
▶ Object {user-agent: "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/537.36 (KHTML"}
```

Shoebox → Store

- ▶ Server data retrieval is cheap
- ▶ Shoebox is nice, but very explicit
- ▶ ember-data-fastboot: automated serialization & population of the ember-data store

`ember install ember-data-fastboot`

Retrieving Data from the store

```
this.store.peekRecord(Post, 7); // Only cache
```

```
this.store.findRecord(Post, 7); // Cache first, then refresh
```

Retrieving Data from the store

- ▶ Usually, `findRecord` is what you want

```
this.store.peekRecord(Post, 7); // Only cache  
this.store.findRecord(Post, 7); // Cache first, then refresh  
  
this.store.findRecord(Post, 7, {  
  backgroundReload: false // Equivalent to "peek"  
});  
  
this.store.findRecord(Post, 7, {  
  reload: true // Equivalent to "fetch"  
});
```

Ember-Simple-Auth

- ▶ **Authenticator:** handles particulars of getting authenticated
- ▶ **Authorizer:** uses privileges to do things
- ▶ **Session-Store:** serializes/deserializes client-side session
- ▶ **Session-Service:** auth state + means of doing stuff w/ it



Shoeboxed Current User

`/api/users/current`

- ▶ Immediately after login, and as long as the session appears to be valid, we want a `currentUser` to be available for use
- ▶ We'll need this data in routes, components and elsewhere
- ▶ Show user email in the navbar about the current user
- ▶ Current authentication scheme: OAuth2 Password Grant
- ▶ The application adapter needs some attention to pass token to API

Major Pieces

App-Like Characteristics

Pruning the Critical Path

Tolerating Network Instability

Tolerating Network Instability

Resource Caching Strategies

Immutable Content

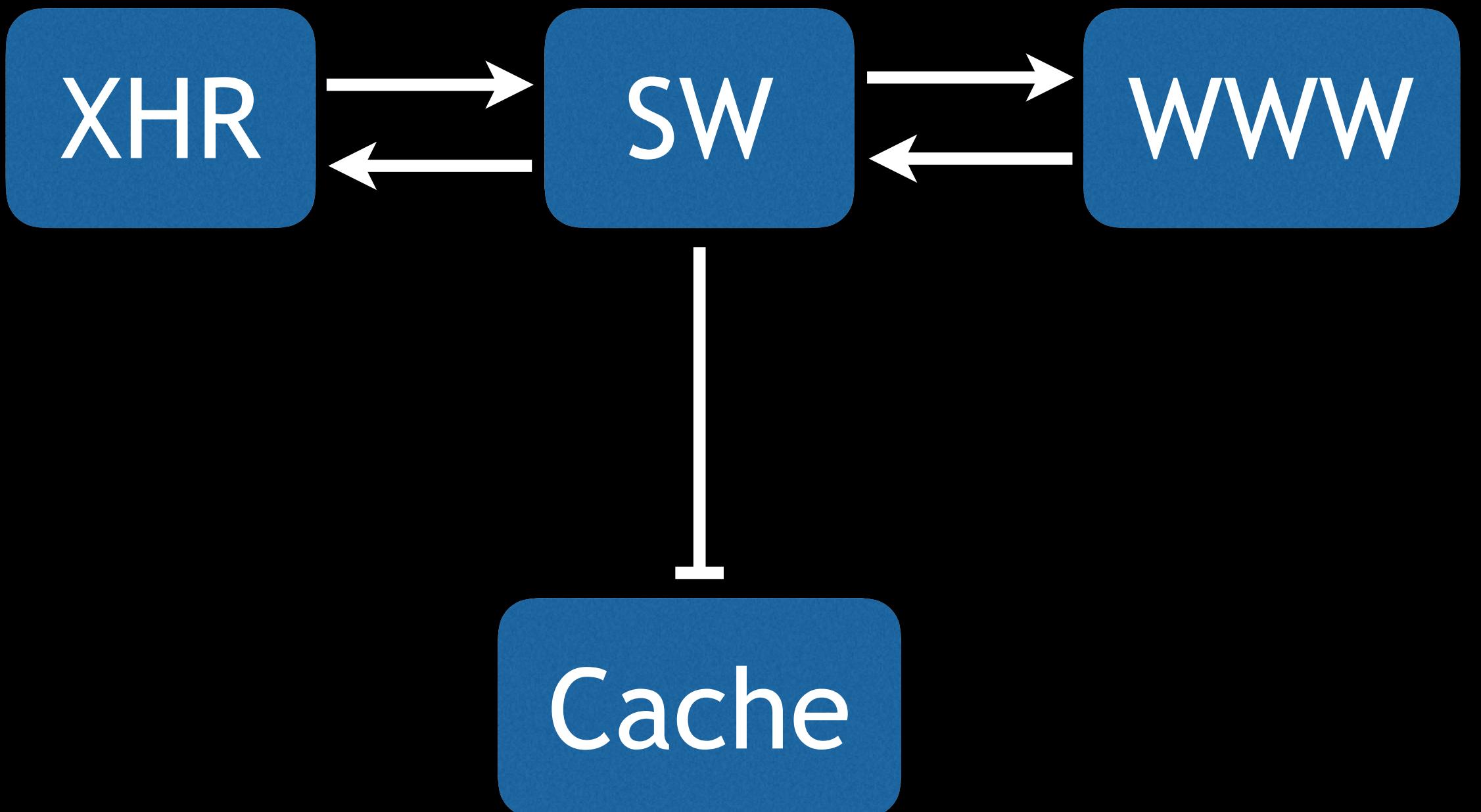
- Content at a URL is never changed
- max-age=year
- index.html is the key
- Offline-friendly

“Check w/ server”

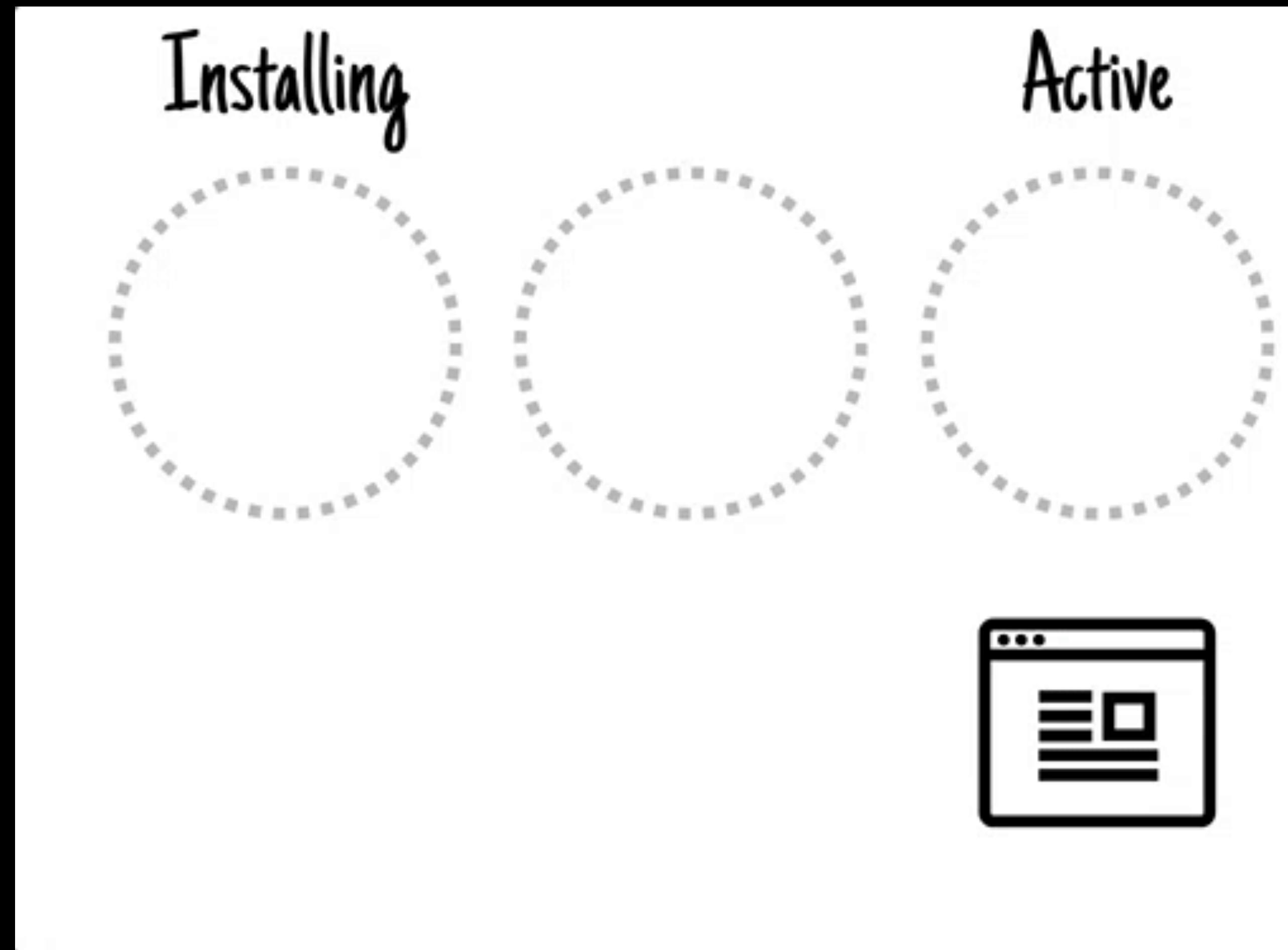
- Content at a URL can change
- Cache-Control: no-cache
- Use Last-Modified or ETag
- Involves extra network requests

Service Workers

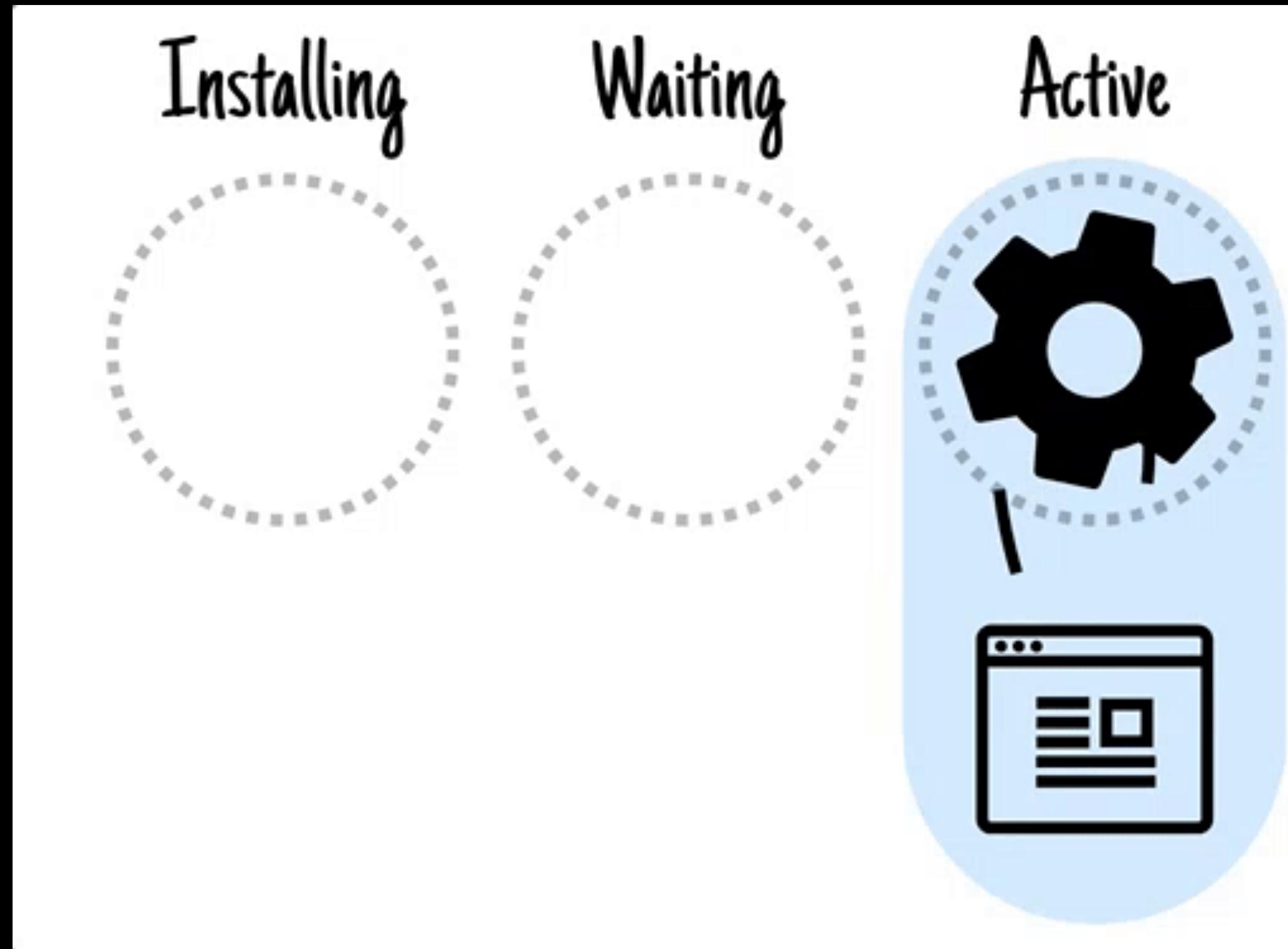
- A programmable network proxy
- A JavaScript worker
- Only works over HTTPS
- Has a predictable lifecycle



Service Workers



Service Workers



Service Workers

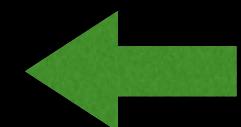
Precache

I have a list of important resources. I'll fetch them all and cache them.

Cache

App

I need item.png



Of course! Here it is.

Service Workers

Network or Cache

App

I need item.png

Ok, I'll get it

Cache



Just got it for you!

.....

Service Workers

Network or Cache

App

I need item.png

Ok, I'll get it

Cache ➔



Took too long. Here's a
cached version

⬅ ↻

Service Workers

Network or Cache

App

I need item.png

Ok, I'll get it

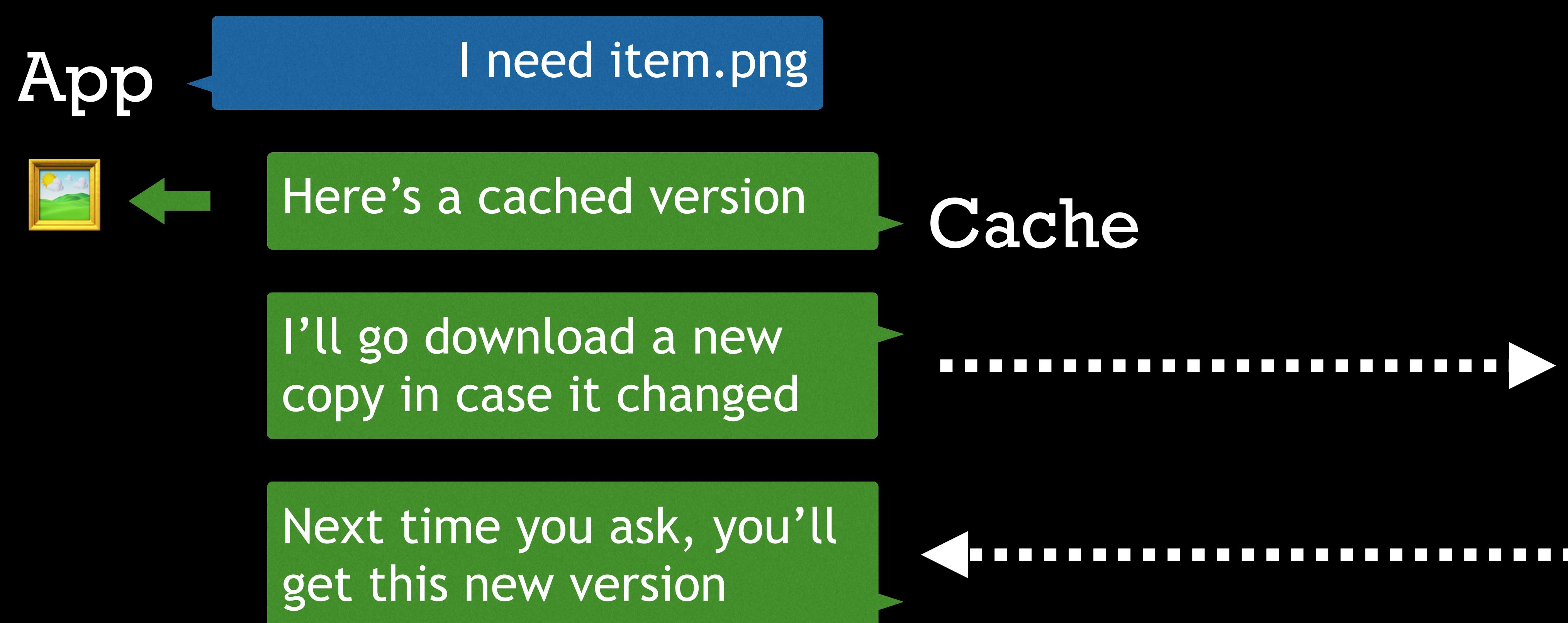
Cache



Something went wrong.
Here's a cached version

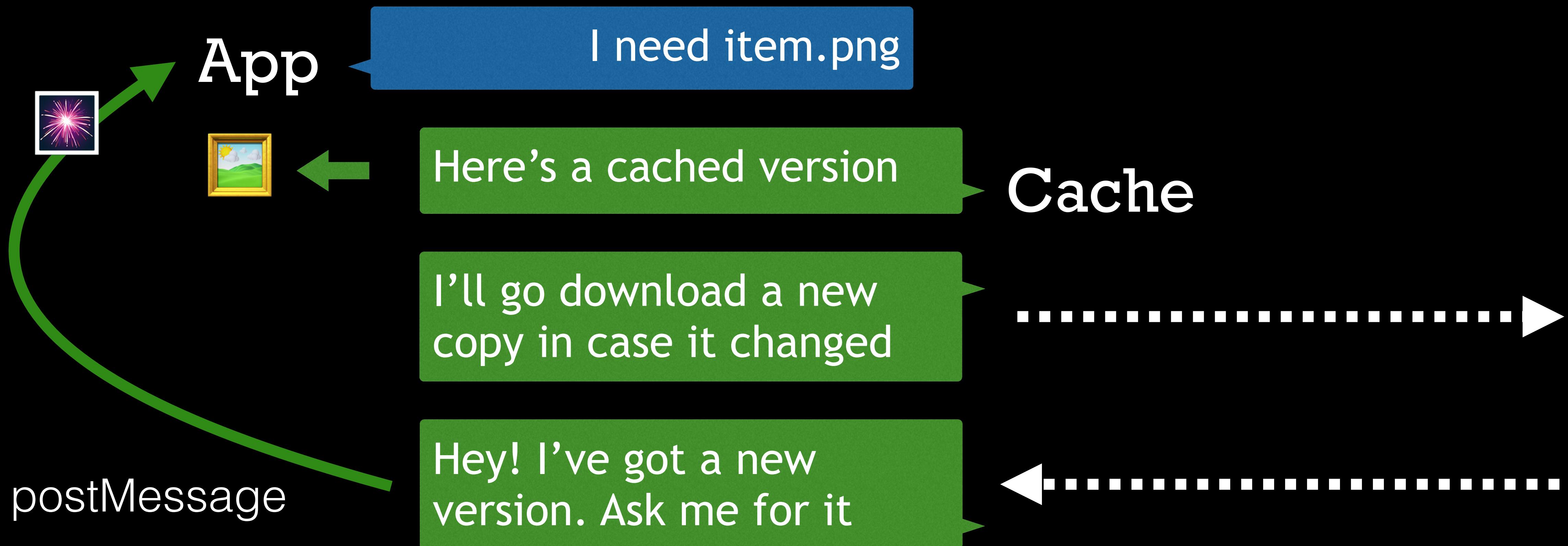
Service Workers

Cache and Update



Service Workers

Cache, Update and Refresh



Service Workers

ServiceWorker enthusiasm

The first thing any implementation needs.



Chrome: Shipped.

Firefox: Shipped.

Samsung Internet: Shipped. Based on Chromium 44.2403 with some [additions and changes](#). (See "Service Worker" section.)

Safari: [Under consideration](#), Brief positive signals in [five year plan](#).

Edge: [In development](#).

Support does not include iOS versions of third-party browsers on that platform (see [Safari support](#)).

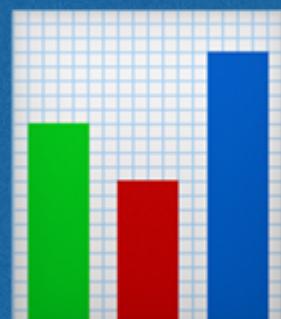
Server Rendered SPA



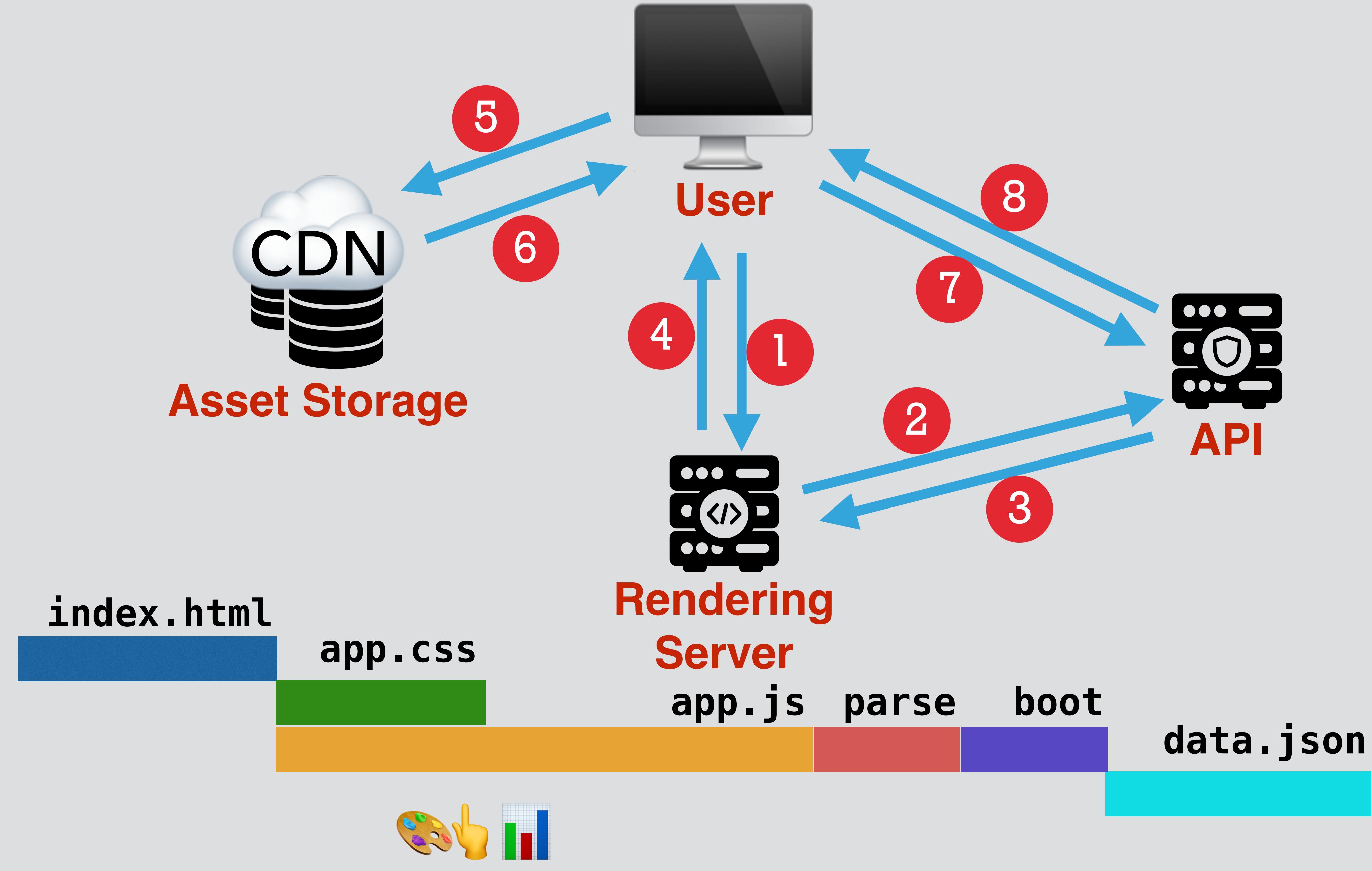
First Paint



Interactive



Data



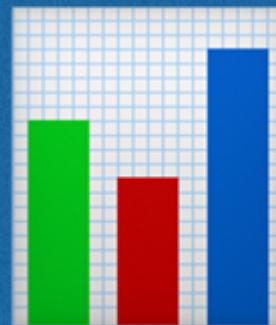
Server Rendered PWA v1



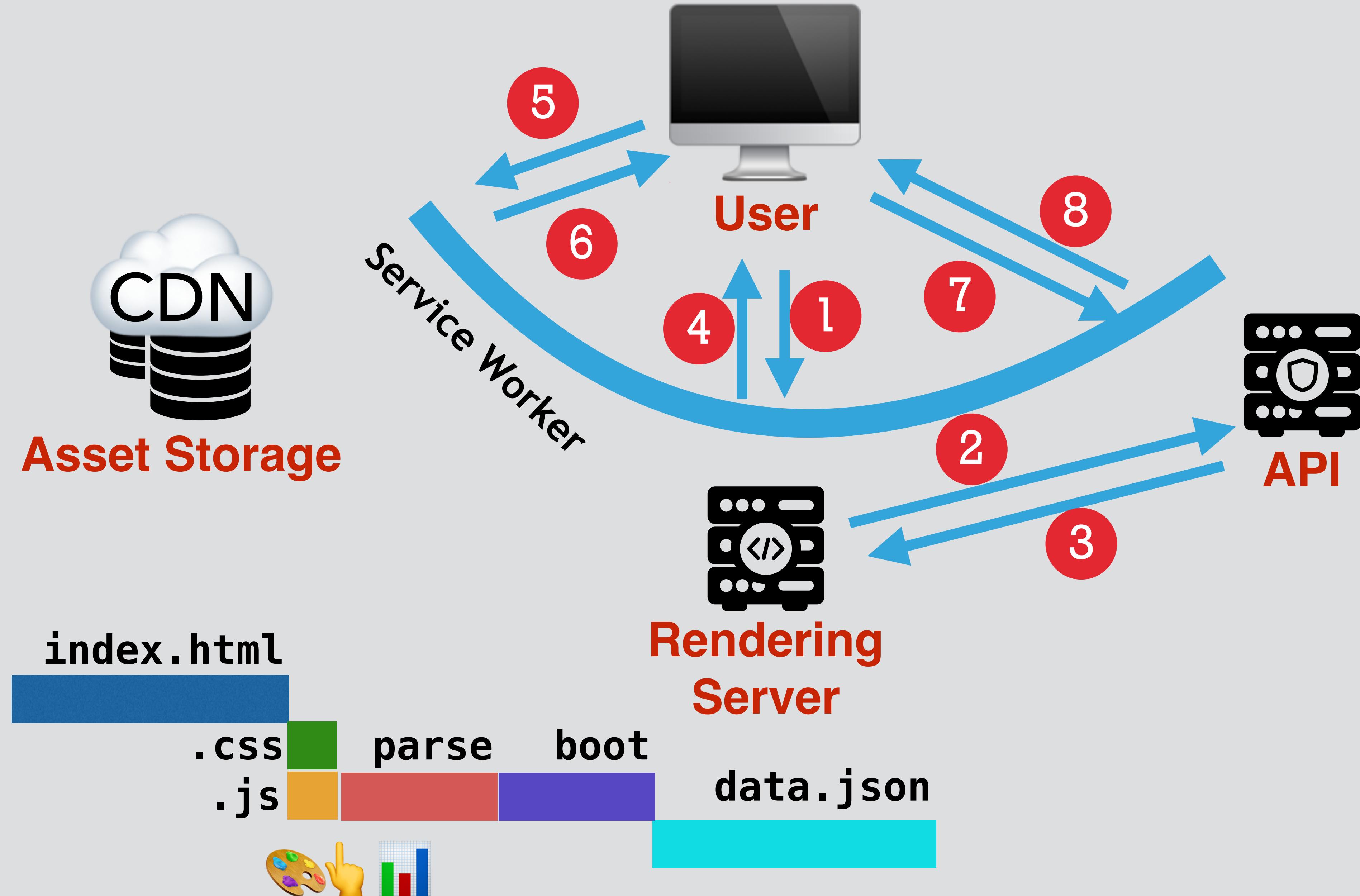
First Paint



Interactive



Data



Service Workers in Ember Apps

ember-service-worker

- Creates appropriate output files
- A pluggable, event-based approach
- Doesn't do anything on its own

package.json

```
"keywords": [  
  "ember-addon",  
  "ember-service-worker-plugin"  
],
```

Service Workers in Ember Apps

ember-service-worker-asset-cache

- Creates a list of your static assets at build time
- Instructs the service worker to download these on install
- Once SW activates, they're ready for instant response
- Versioned — new workers purge old stuff on activate

Service Workers in Ember Apps

ember-service-worker-index

- Caches HTML for initial load
- Currently appears to interfere with fastboot
- Supports app-shell architecture (we'll talk about this later)

Service Workers in Ember Apps

ember-service-worker-cache-first

- Specify regular expressions or patterns for files
- When your app asks for a resource, provide the cached version
- ... and then go and refresh the cache with new data

Service Workers in Ember Apps

ember-service-worker-cache-fallback

- **When you're online:** Passthrough (cache as requests are made)
- **When you're offline:** Serve responses from the cache

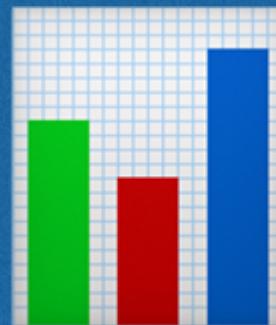
Server Rendered PWA v1



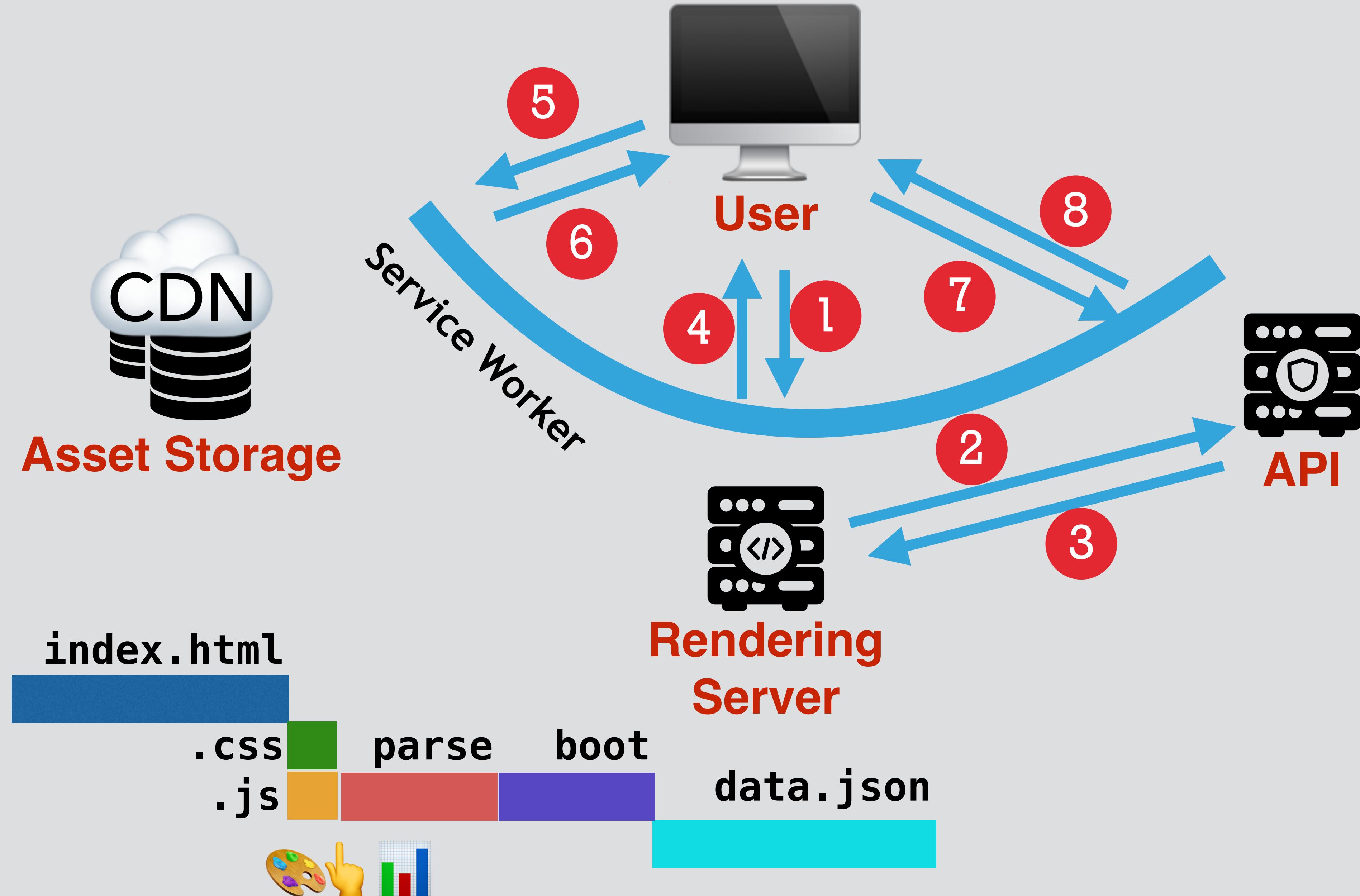
First Paint



Interactive



Data



<https://github.com/mike-north/ember-pro>
[./EmberPro.pdf](#)

ember PRO



Sept 5, 2017
Mike North

Works Offline

```
ember install ember-service-worker  
ember install ember-service-worker-asset-cache  
ember install ember-service-worker-cache-fallback  
ember install ember-service-worker-index
```

- ▶ Install these addons, and configure such that
 - ▶ JS, CSS, images in the public folder are handled by asset-cache
 - ▶ Speaker images and JSON are handled by the cache-fallback
 - ▶ HTML is handled by ember-service-worker-index
- ▶ Your app should now work even when you're offline

Storing Dynamic Content

- Cookies - Not worker-friendly, just a string
- LocalStorage - See 
- What's left???

Storing Dynamic Content

- Cookies - Not worker-friendly, just a string
- LocalStorage - See 

IndexedDB

IndexedDB

-  Versioned
-  Indexable
-  Worker-Friendly
-  Durable
-  Supports values of many types
-  Not a SQL, or a relational DB

IndexedDB

```
// Open (or create) the database
let open =
  indexedDB.open('MyDatabase', 1);

// Create and/or Migrate the Schema
open.onupgradeneeded = (evt) => { ... };

// Transactions
open.onsuccess = () => { ... };
```

IndexedDB

```
// Create and/or Migrate the Schema
open.onupgradeneeded = (evt) => {

    let db = open.result;
    let store = null;
    switch (evt.oldVersion) {
        case 0: // Upgrade from 0
            // ...
        case 1: // Upgrade from 1
            // ...
    }
};
```

IndexedDB

```
// Open (or create) the database
let open =
  indexedDB.open('MyDatabase', 1);
```

```
// Create and/or Migrate the Schema
open.onupgradeneeded = (evt) => { ... };
```

```
// Transactions
open.onsuccess = () => { ... };
```

IndexedDB

```
// Transactions
open.onsuccess = () => {

    // Start a new transaction
    let db = open.result;
    let tx = db.transaction('MyObjectStore', 'readwrite');
    let store = tx.objectStore('MyObjectStore');

    // ← Transaction Particulars → //

    // Close the db when the transaction is done
    tx.oncomplete = () => db.close();

};
```

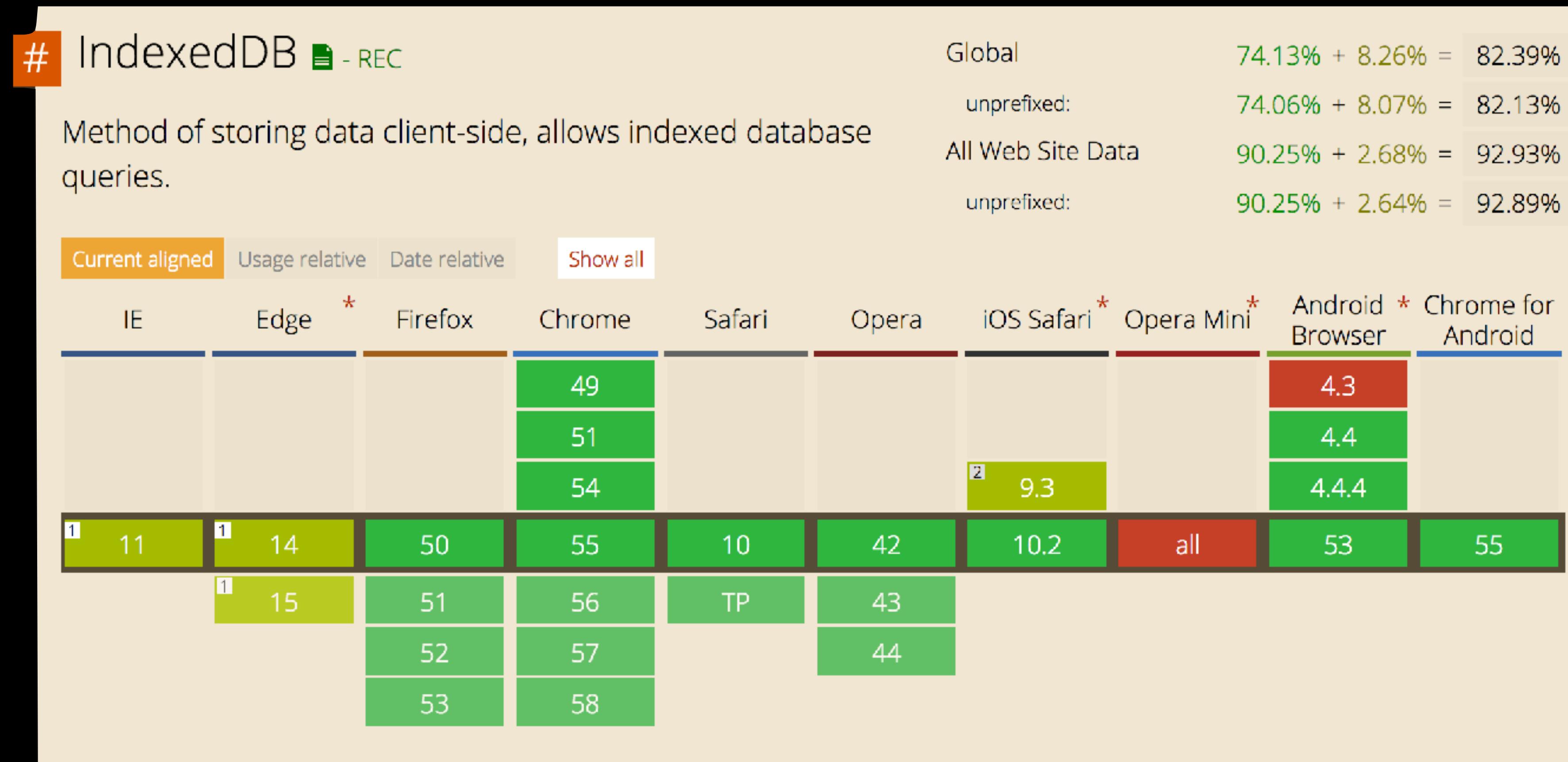
IndexedDB

```
let index = store.index('NameIndex');

// Add some data
store.put({
  id: 12345,
  name: { first: 'Mike', last: 'North' },
  age: 33
});

// Query the data
let getJohn = store.get(12345);
let getBob = index.get(['North', 'Mike']);
```

IndexedDB



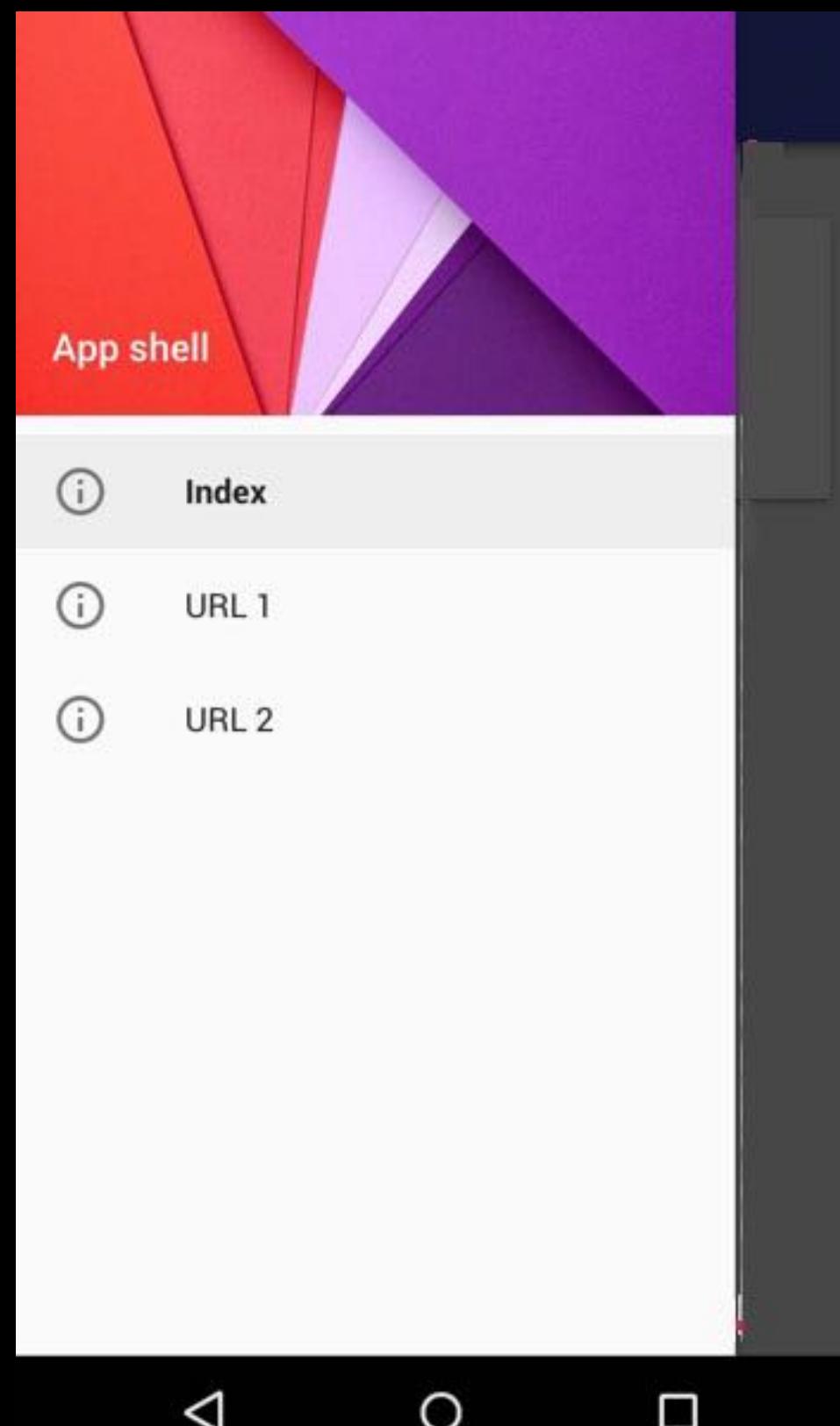
IndexedDB Storage

10

- ▶ Use either `localStorage` or `IndexDB` to allow “bingo pieces” to persist across reloads.
- ▶ If you use `IndexDB`, make sure to
 - ▶ `deferReadiness/advanceReadiness`
 - ▶ use a promise-aware hook in a route somewhere to ensure “moves” are restored before rendering happens

App Shell Architecture

Shell loads instantly



Dynamic content in the shell



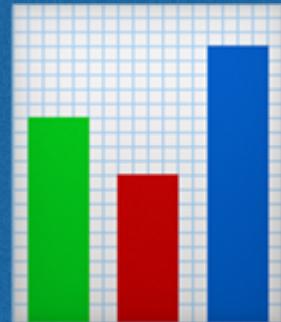
Server Rendered PWA v1



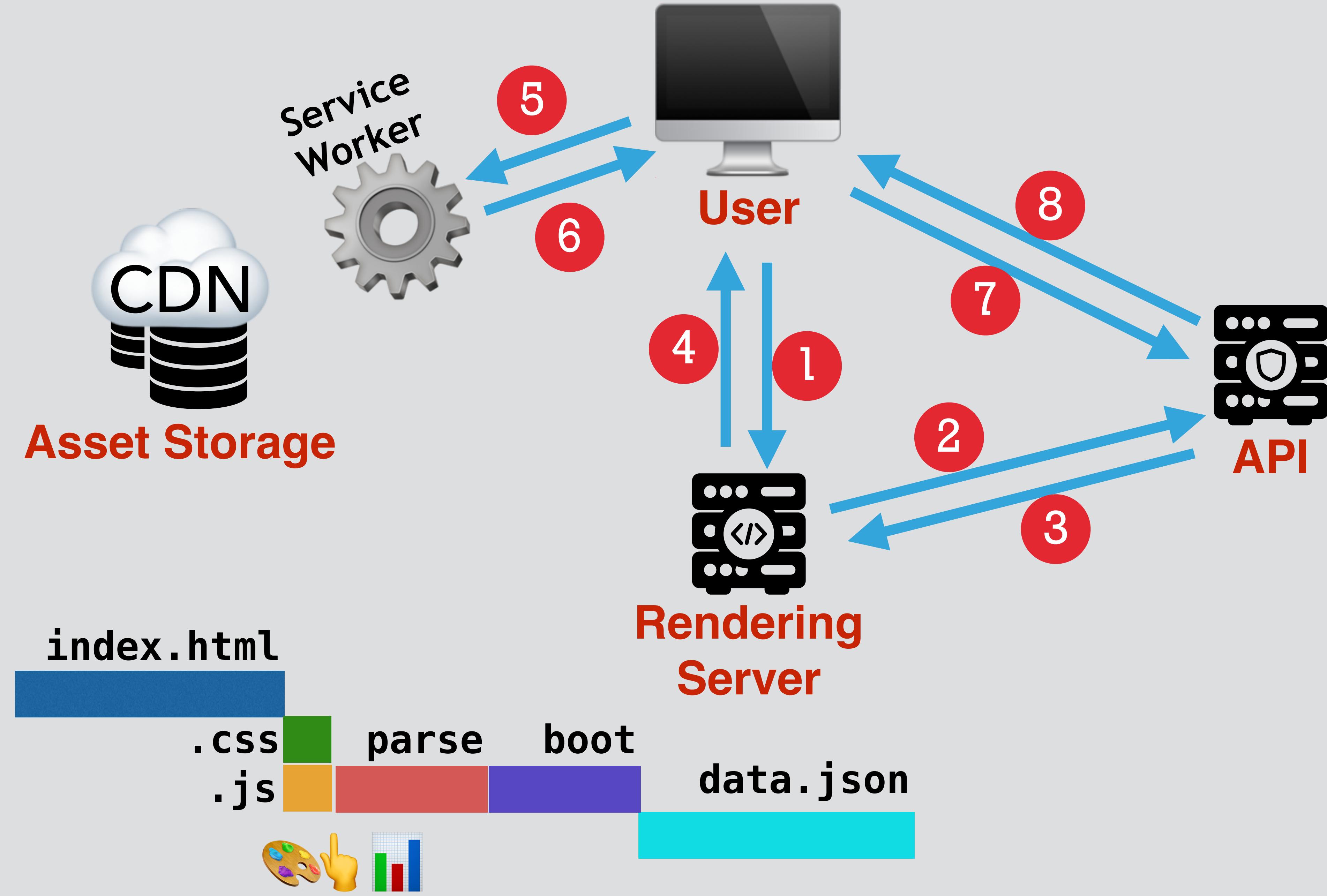
First Paint



Interactive



Data



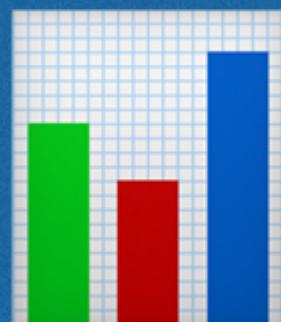
Server Rendered PWA v2



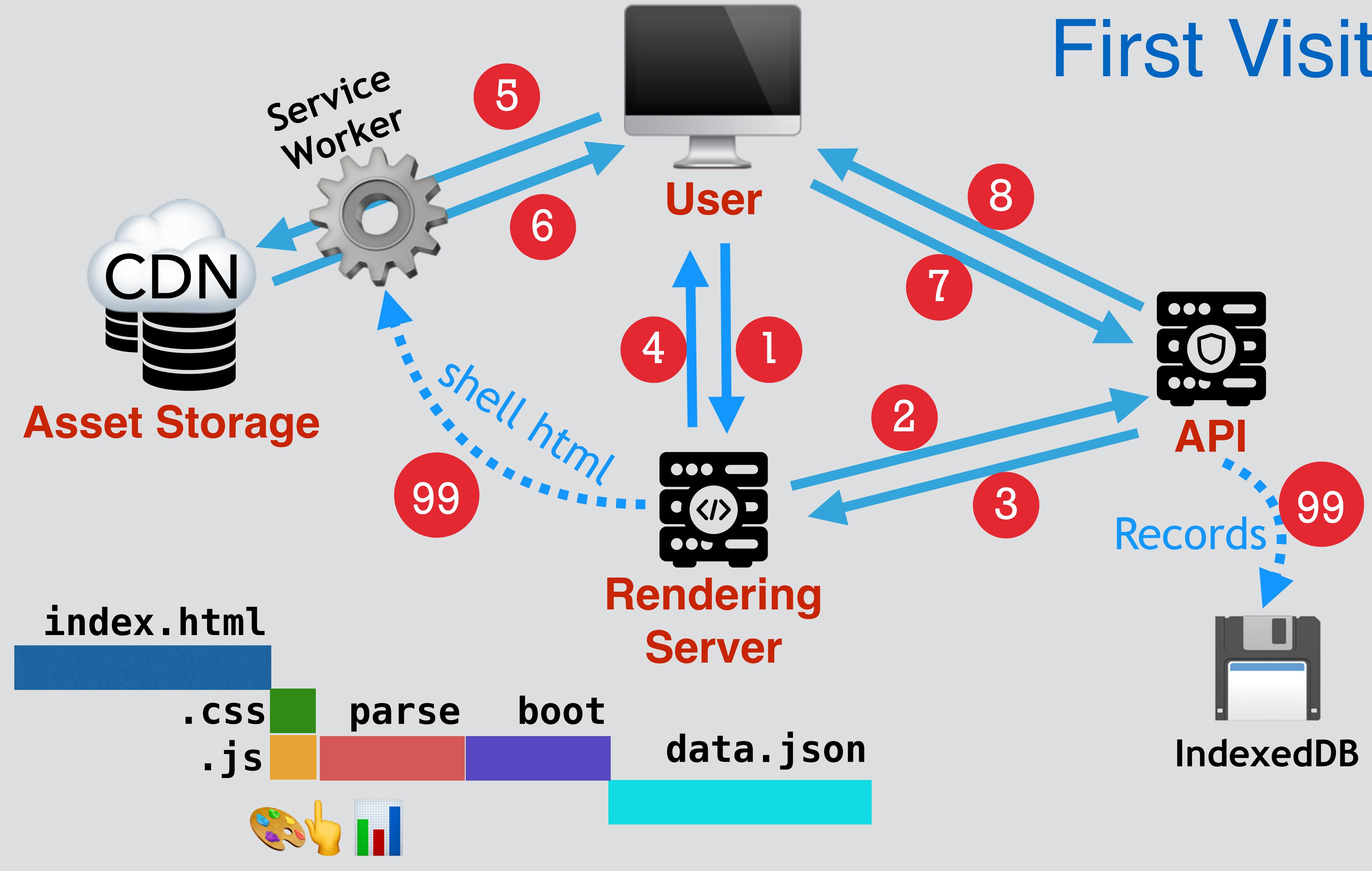
First Paint



Interactive



Data



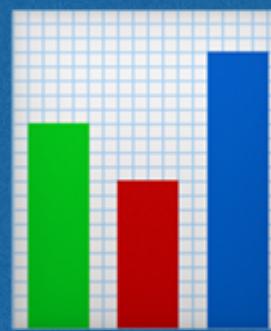
Server Rendered PWA v2



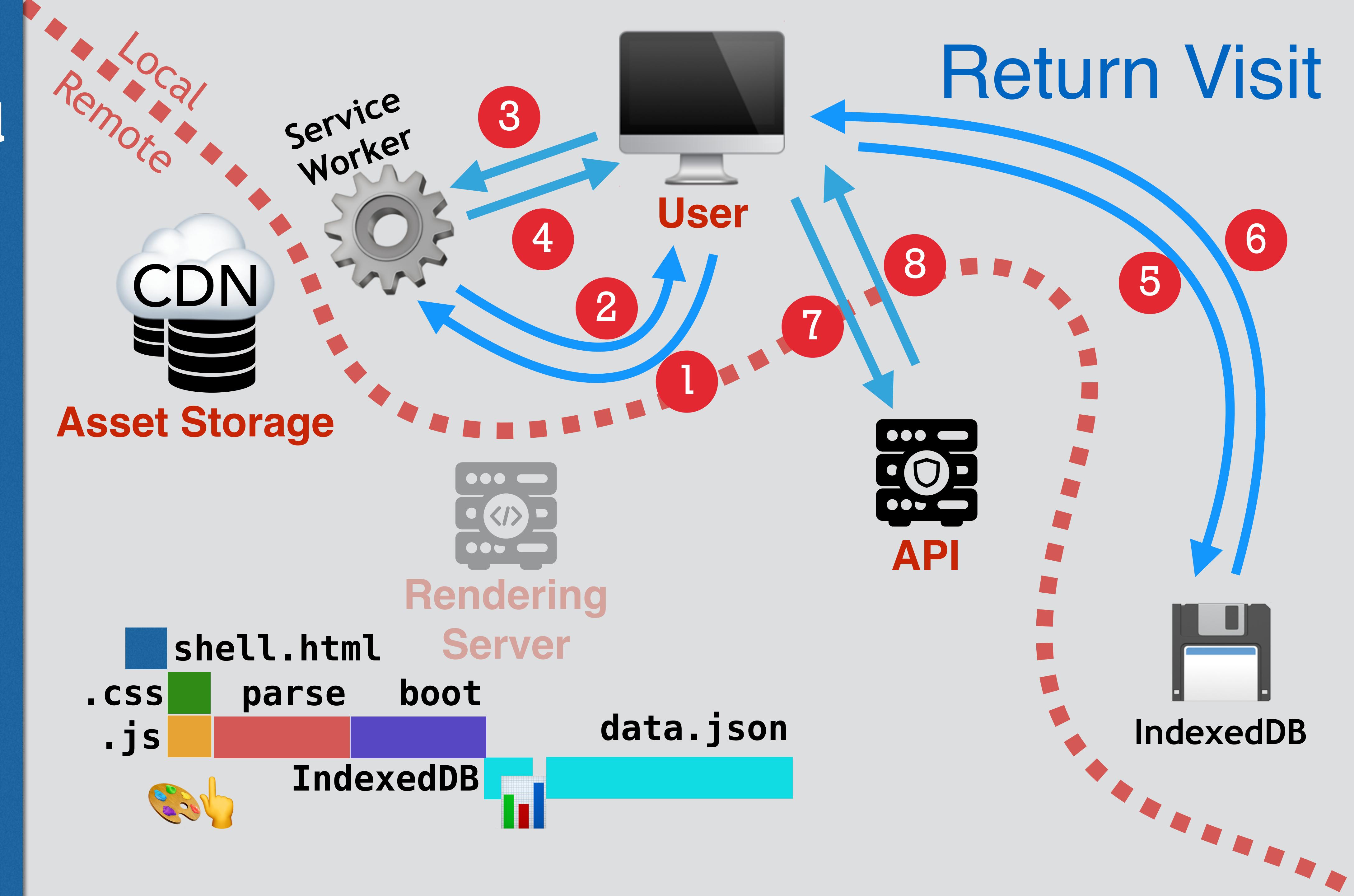
First Paint



Interactive



Data





State Management

4 Flavors of state

UI STATE

PERSISTED
STATE

ADDRESSABLE
STATE

DRAFT STATE

localhost:4200/posts/A96usFSIY4G0W4kwAqsw

148

The screenshot shows a web browser window with a list of posts on the left and a detailed view of a post on the right.

Posts List:

- Seven Tips From Ernest He...** by Mike Springer (October 2, 2016) - Preview: To get started, write one true sentence
- Where does your state belong** by Mike North (October 10, 2016) - Preview: For over a decade, most of us built web UIs operating under the assumption that ...
- Down the Rabbit Hole** by Lewis Carroll (October 2, 2016) - Preview: Alice was beginning to get very tired of sitting by her sister on the bank, and ...

Detailed View (Right Side):

Seven Tips From Ernest Hemingway on How to Write Fiction



[details...](#)

To get started, write one true sentence

Sometimes when I was starting a new story and I could not get it going, I would sit in front of the fire and squeeze the peel of the little oranges into the edge of the flame and watch the sputter of blue that they made. I would stand and look out over the roofs of Paris and think, “Do not worry. You have always written before and you will write now. All you have to do is write one true sentence. Write the truest sentence that you know.”

So finally I would write one true sentence, and then go on from there. It was easy then because there was always one true sentence that I knew or had seen or had heard someone say. If I started to write elaborately, or like someone introducing or presenting something, I found that I could cut that scrollwork or ornament out and throw it away and start with the first true simple declarative sentence I had written.

Always stop for the day while you still know what will happen next

The best way is always to stop when you are doing good and

localhost:4200/posts/1fm1gRRxhS0GAkC2k4G8UG?q=When

Where

 Where does your state belong 1
Mike North October 10, 2016
For over a decade, most of us built web UIs operating under the assumption that ...

our users make about how apps should work, and we must work a bit harder in order to keep them intact. Take the 'Back' and 'Refresh' buttons, for example: in order for this to work as our users expect, we must keep certain elements of state serialized in the URL in order to avoid breaking this as we simulate a multi-page experience in a SPA.

Add in the concept of server-side rendering, where our asset serving layer sometimes needs browser details (i.e. viewport dimensions) in order to render the correct content, and state decisions become even more consequential and complex.

In this talk, I'll outline four types of state

- Navigation state
- Persisted state
- UI state
- "Will be persisted" state

and provide examples for each. Along the way, we'll start to assemble a framework of questions that you can ask yourself when encountering new pieces of state, to lead you down the right path(s).

This is another comment X
21 minutes ago

© 2016 GitHub, Inc. All rights reserved.

localhost:4200/posts/1fm1gRRxhS0GAkC2k4G8

'Refresh' buttons, for example: in order for this to work as our users expect, we must keep certain elements of state serialized in the URL in order to avoid breaking this as we simulate a multi-page experience in a SPA.

Add in the concept of server-side rendering, where our asset serving layer sometimes needs browser details (i.e. viewport dimensions) in order to render the correct content, and state decisions become even more consequential and complex.

In this talk, I'll outline four types of state

- Navigation state
- Persisted state
- UI state
- "Will be persisted" state

and provide examples for each. Along the way, we'll start to assemble a framework of questions that you can ask yourself when encountering new pieces of state, to lead you down the right path(s).

This is another comment

21 minutes ago

This talk is going pretty well!

Reply Cancel

localhost:4200/posts/1fm1gRRxhS0GAkC2k4G81

'Refresh' buttons, for example: in order for this to work as our users expect, we must keep certain elements of state serialized in the URL in order to avoid breaking this as we simulate a multi-page experience in a SPA.

Add in the concept of server-side rendering, where our asset serving layer sometimes needs browser details (i.e. viewport dimensions) in order to render the correct content, and state decisions become even more consequential and complex.

In this talk, I'll outline four types of state

- Navigation state
- Persisted state
- UI state
- "Will be persisted" state

and provide examples for each. Along the way, we'll start to assemble a framework of questions that you can ask yourself when encountering new pieces of state, to lead you down the right path(s).

This is another comment

21 minutes ago

This talk is going pretty well!

a few seconds ago

© 2014 GitHub, Inc. All rights reserved.

POSTS

{{POST-TILE}}

{{POST-TILE}}

{{POST-TILE}}

POSTS/SHOW

{{POST-FULL}}

{{POST-COMMENT}}

{{POST-COMMENT}}

Addressable State

Addressable State

Sharable, Historically archived
and URL-Driven

**Core
Concept**

Examples

- ▶ Sorted/filtered search results
- ▶ Tabs/Navigation
- ▶ A section of a Github Readme

How to identify Addressable state

- ▶ Affects GET API calls
 - ▶ A perspective on a record or collection of records
- ▶ Useful when shared
- ▶ Mismanaged when - **back button breaks**

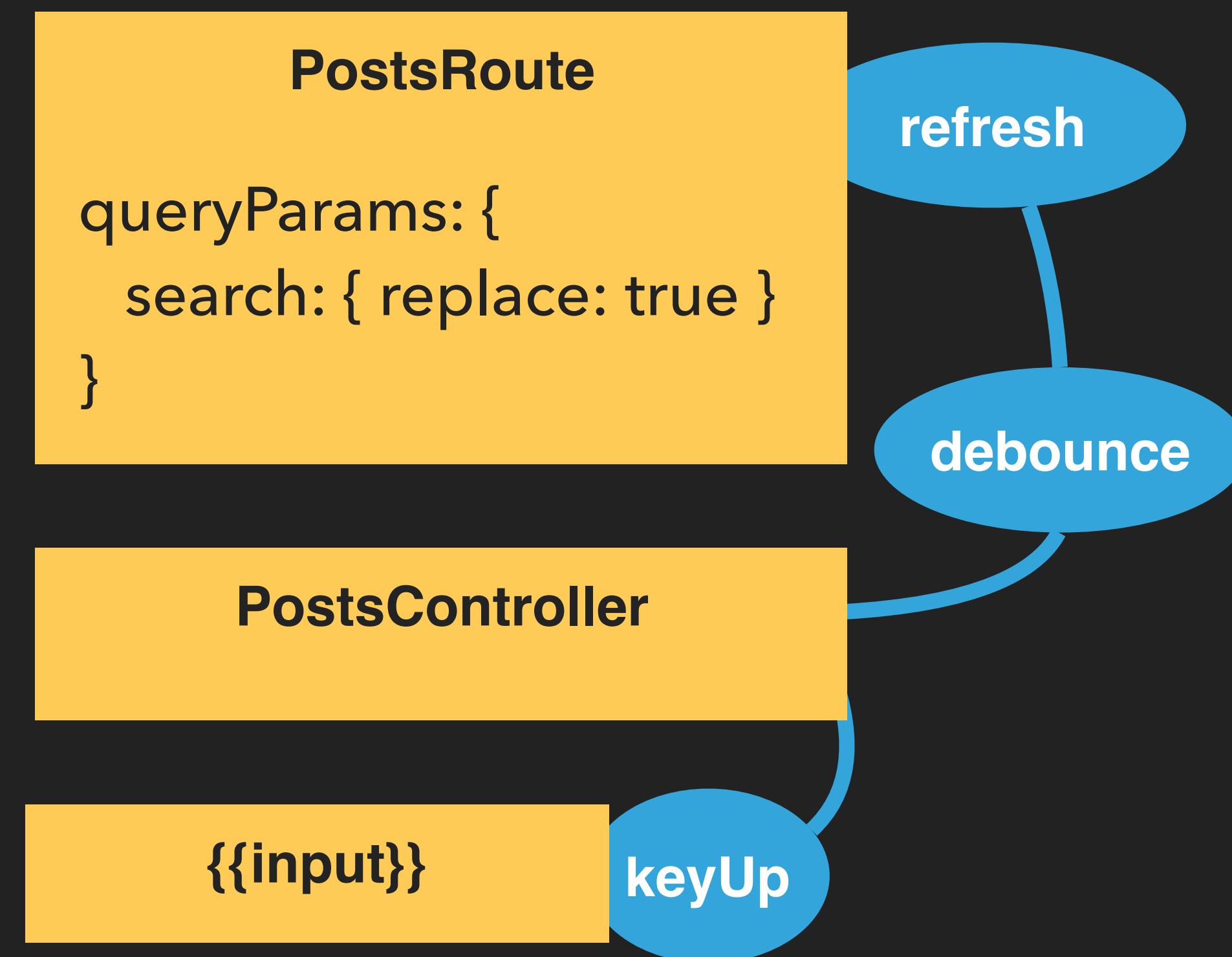
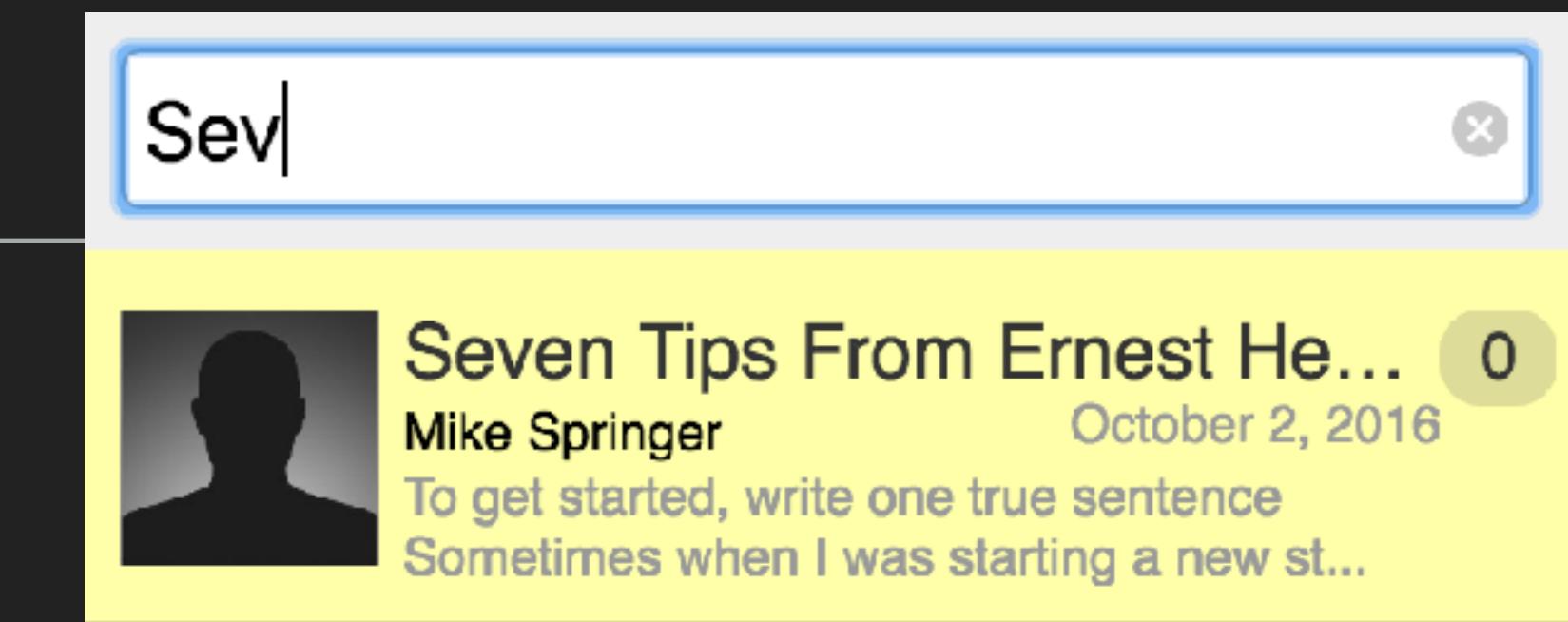
contract: users and the web

- ▶ **Refresh:** never destructive by surprise
- ▶ **Bookmarks:** get back to where I was
- ▶ **Back:** previous major thing
- ▶ **Back a lot:** leave the app



Enhancing Our Basic App

- ▶ Filter by a name fragment
- ▶ Bookmark & Share
- ▶ Don't abuse our API
- ▶ Data Down, Actions Up



Gotchas and Common Issues

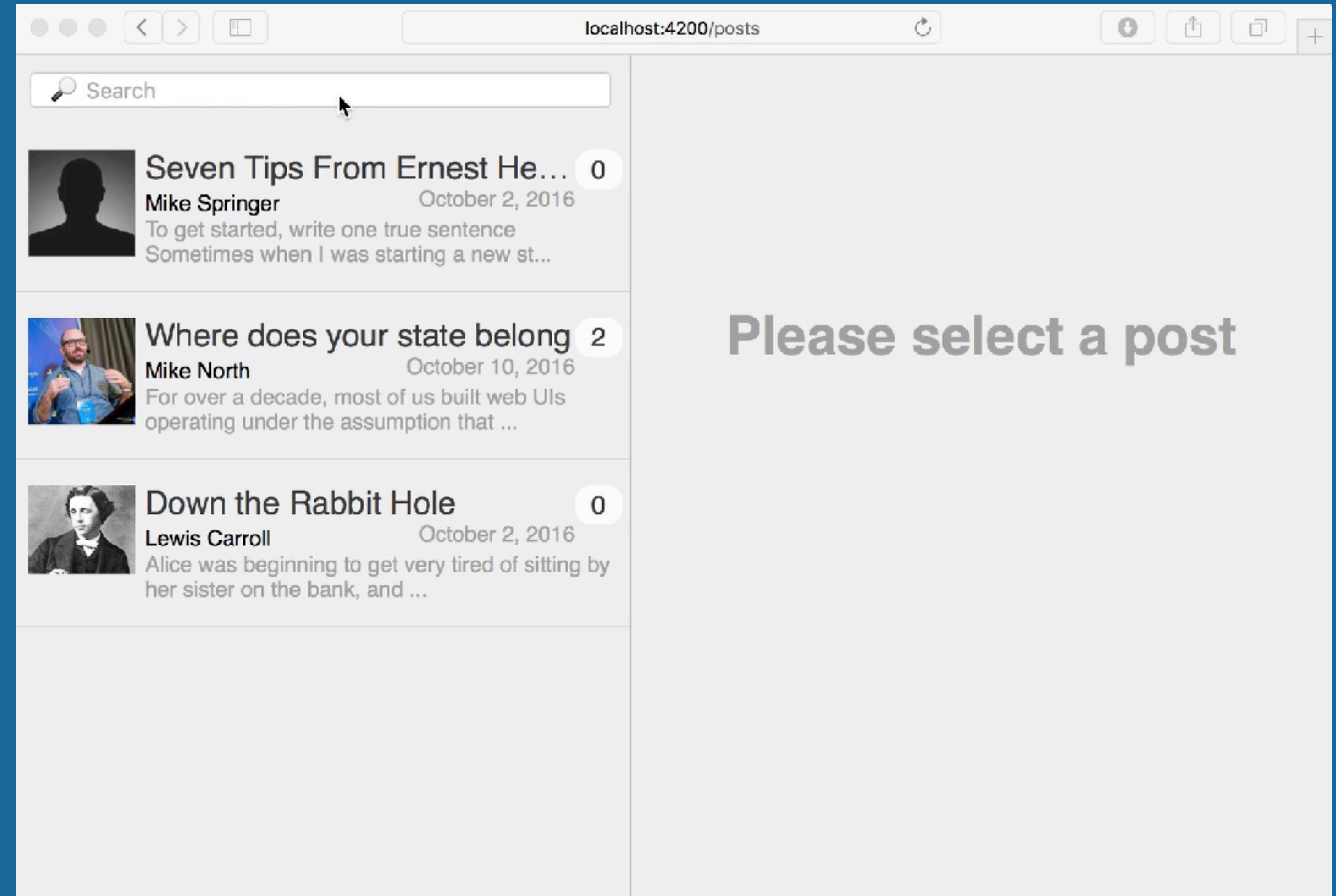
- ▶ Trapping users
- ▶ Deleting useful history
- ▶ Unimportant history states
- ▶ Some Route queryParam options are mutually exclusive (right now) (replace vs refreshModel)
- ▶ Tricky: Add to Home Screen

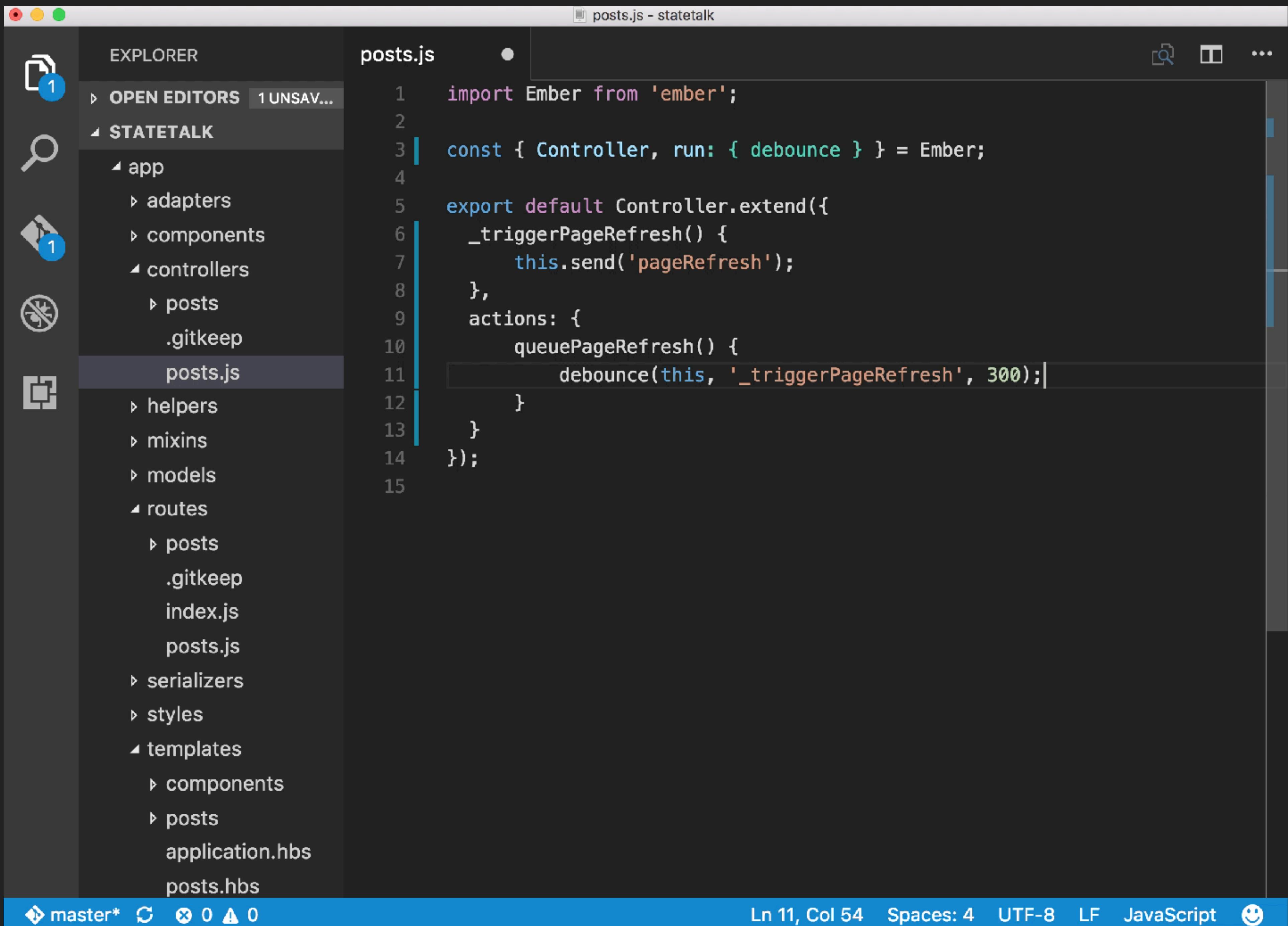


Filter byQueryParam

10

- ▶ Set up a queryparam on the posts route
- ▶ ▶ Typing 3+ letters in the search field should result in the list
- ▶ of posts being filtered
- ▶ ▶ Adhere to data-down, actions up
- ▶ ▶ Bonus: Debounce if you know what that means





The screenshot shows the VS Code interface with the following details:

- Title Bar:** posts.js - statetalk
- File Explorer (Left):** Shows the project structure under the STATETALK folder:
 - controllers
 - posts (highlighted)
 - .gitkeep
 - posts.js
- helpers
- mixins
- models
- routes (highlighted)
 - posts (highlighted)
 - .gitkeep
 - index.js
 - posts.js
- serializers
- styles
- templates (highlighted)
 - components
 - posts
 - application.hbs
 - posts.hbs

- Code Editor (Right):** The posts.js file content is displayed:

```
1 import Ember from 'ember';
2
3 const { Route } = Ember;
4
5 export default Route.extend({
6   queryParams: {
7     search: {
8       as: 's',
9       replace: true
10    }
11  },
12  model() {
13    return this.store.findAll('post');
14  }
15});
16
```
- Bottom Status Bar:** master* 0 0 0 | Ln 11, Col 5 | Spaces: 2 | UTF-8 | LF | JavaScript | ☺

The screenshot shows the VS Code interface with the following details:

- Title Bar:** posts.js - statetalk
- Explorer:** Shows the project structure under the STATETALK folder:
 - controllers
 - posts (highlighted)
 - .gitkeep
 - posts.js

Below this, other sections like helpers, mixins, models, routes, serializers, styles, templates, and components are listed.
- Editor:** The posts.js file is open, showing the following code:

```
1 import Ember from 'ember';
2
3 const { Route } = Ember;
4
5 export default Route.extend({
6   queryParams: {
7     search: {
8       as: 's',
9       replace: true
10    }
11  },
12  actions: {
13    pageRefresh() {
14      this.refresh();
15    }
16  },
17  model() {
18    return this.store.findAll('post');
19  }
20});
21
```
- Status Bar:** master* 3 3 3 | Ln 16, Col 5 Spaces: 2 UTF-8 LF JavaScript

The screenshot shows the VS Code interface with the following details:

- Title Bar:** posts.js - statetalk
- Explorer:** Shows the project structure under the STATETALK folder:
 - controllers
 - posts
 - .gitkeep
 - posts.js
 - helpers
 - mixins
 - models
 - routes
 - posts
 - .gitkeep
 - index.js
 - posts.js
 - serializers
 - styles
 - templates
 - components
 - posts
 - application.hbs
 - posts.hbs
- Active Editor:** posts.js routes (tab bar)
- Code Content:**

```
1 import Ember from 'ember';
2
3 const { Route } = Ember;
4
5 export default Route.extend({
6   queryParams: {
7     search: {
8       as: 's',
9       replace: true
10    }
11  },
12  actions: {
13    pageRefresh() {
14      this.refresh();
15    }
16  },
17  model({ search }) {
18    if (!search) {
19      return this.store.findAll('post');
20    } else {
21      return this.store.query('post', { search });
22    }
23  }
24});
25
```
- Status Bar:** master* 0 0 0 0 Ln 21, Col 39 Spaces: 2 UTF-8 LF JavaScript

Draft State

Draft State

Locally-stored Temporarily, will eventually be persisted



**Core
Concept**

Examples

- ▶ A reply to an email message
- ▶ The body of a huge GitHub comment
- ▶ In-flight records
- ▶ Will turn into persisted state
- ▶ High user effort
- ▶ Survives **transitions**
- ▶ Detrimental to UX if lost

HOW TO IDENTIFY

Conceptual Model

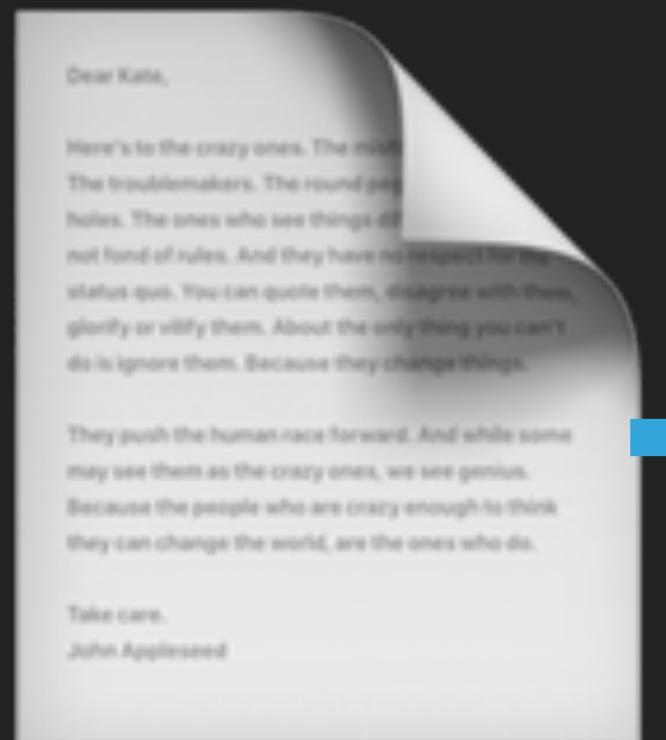


A Post



A Comment

Conceptual Model

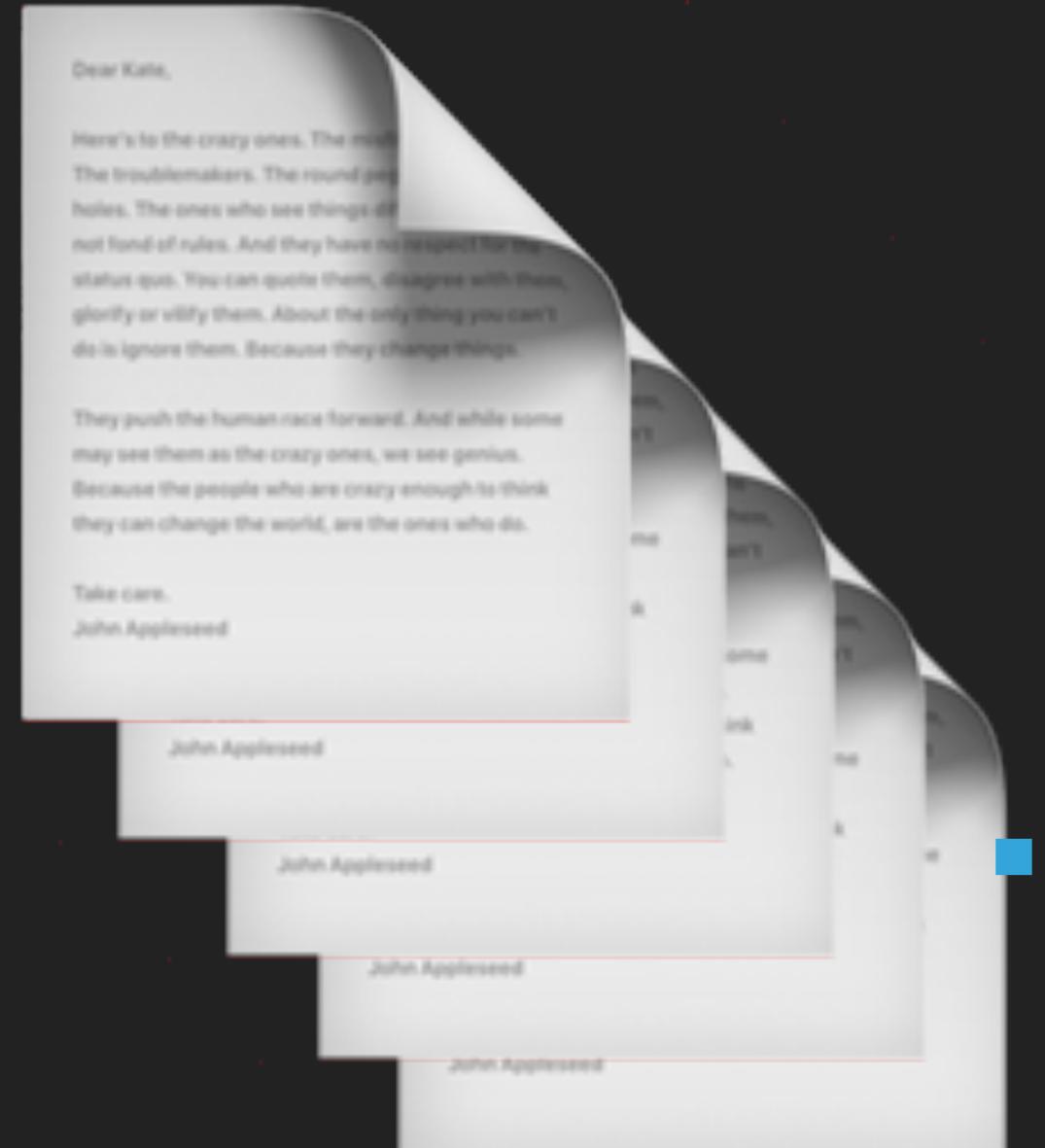


A Post



A Comment

Conceptual Model

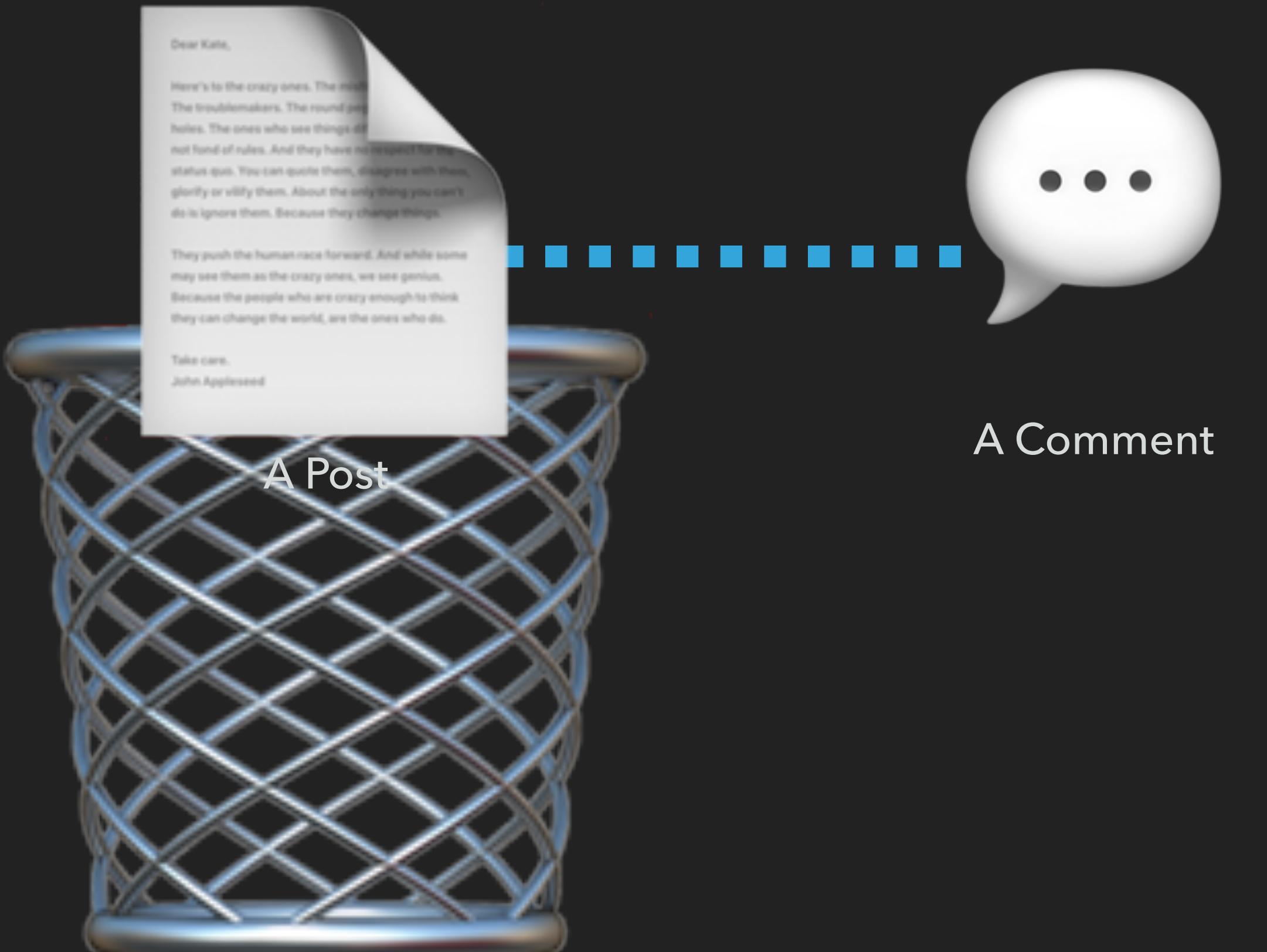


A Post

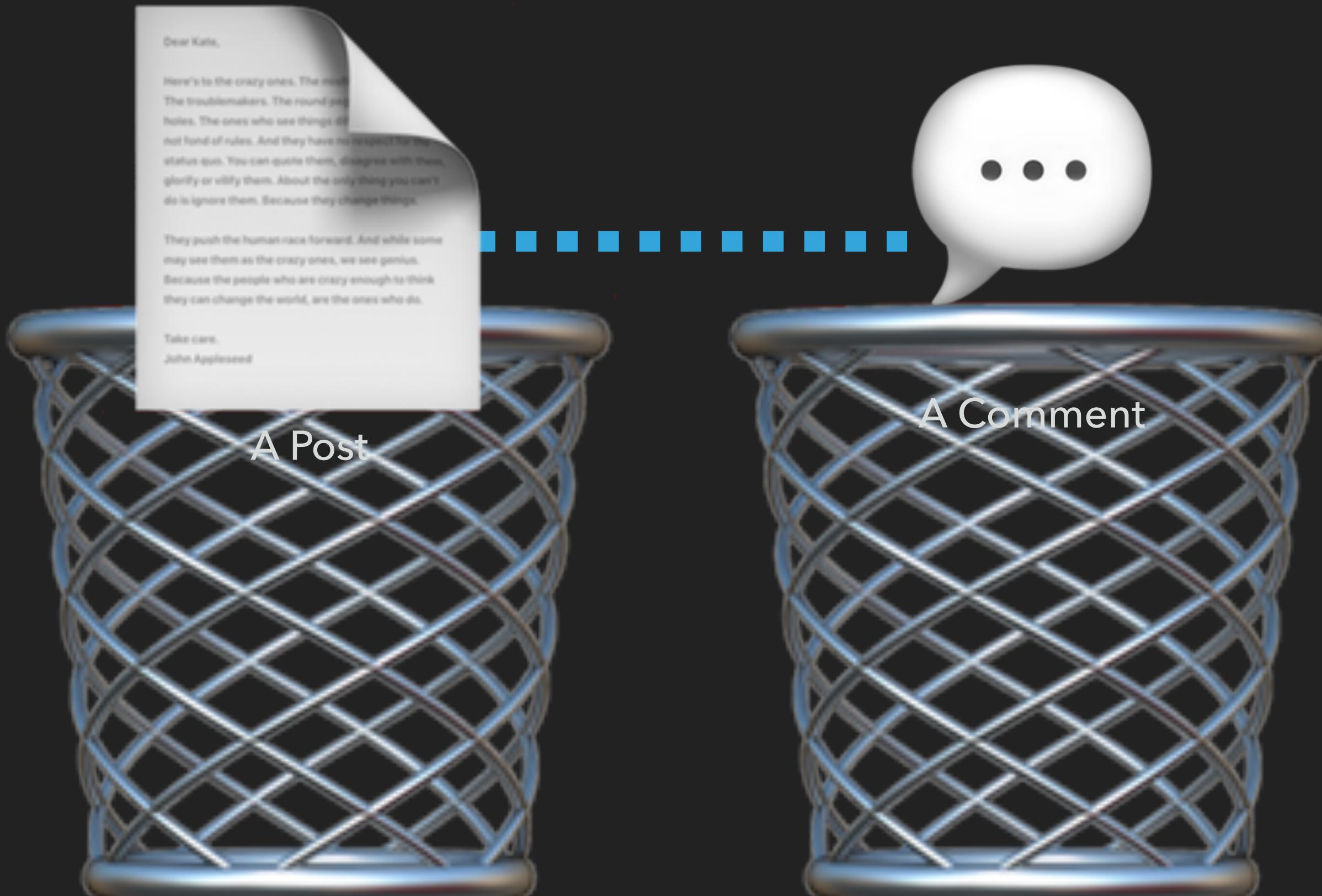


A Comment

Conceptual Model



Conceptual Model



Conceptual Model - KEY CONCEPT

WEAK MAP

Conceptual Model - KEY CONCEPT

WEAK MAP

If an object that is being used as the key of a WeakMap key/value pair is only reachable by following a chain of references that start within that WeakMap, that key/value pair is inaccessible and is automatically removed from the WeakMap.

Conceptual Model - KEY CONCEPT

WEAK MAP

If an object that is being used as the key of a WeakMap key/value pair is only reachable by following a chain of references that start within that WeakMap, that key/value pair is inaccessible and is automatically removed from the WeakMap.

Conceptual Model - KEY CONCEPT

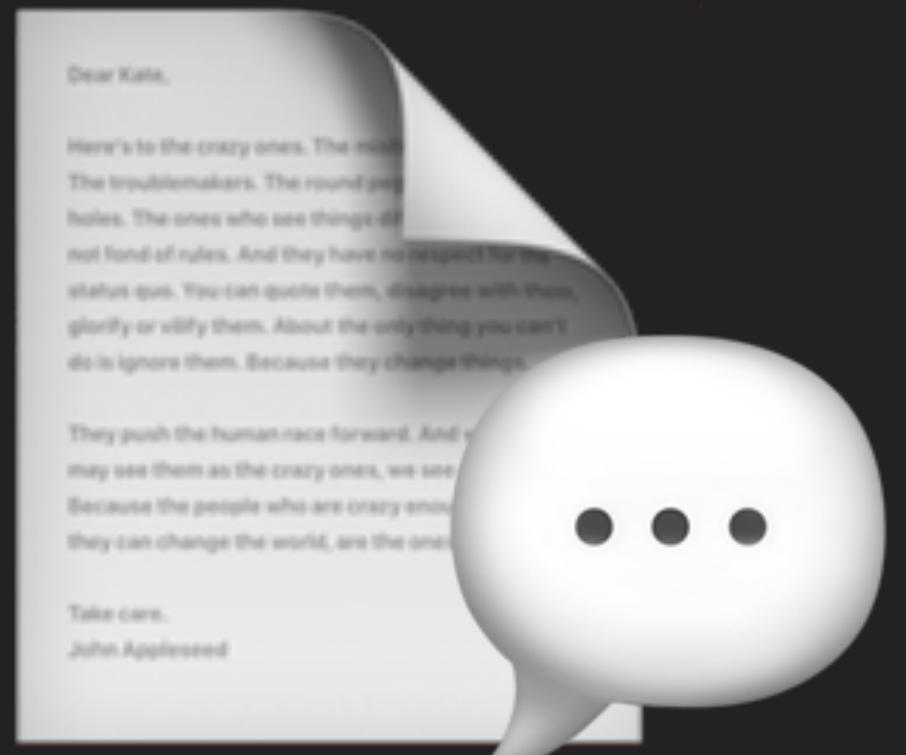
WEAK MAP

reachable by following a chain of
references that start within that WeakMap,
that key/value pair is inaccessible and is
automatically removed from the
WeakMap.

```
// Create
let comm
// Post is
comment
```

Conceptual Model - KEY CONCEPT

WEAK MAP*



A Post

A Comment

```
// Store the value on the key  
post._some_random_property_name = comment;
```

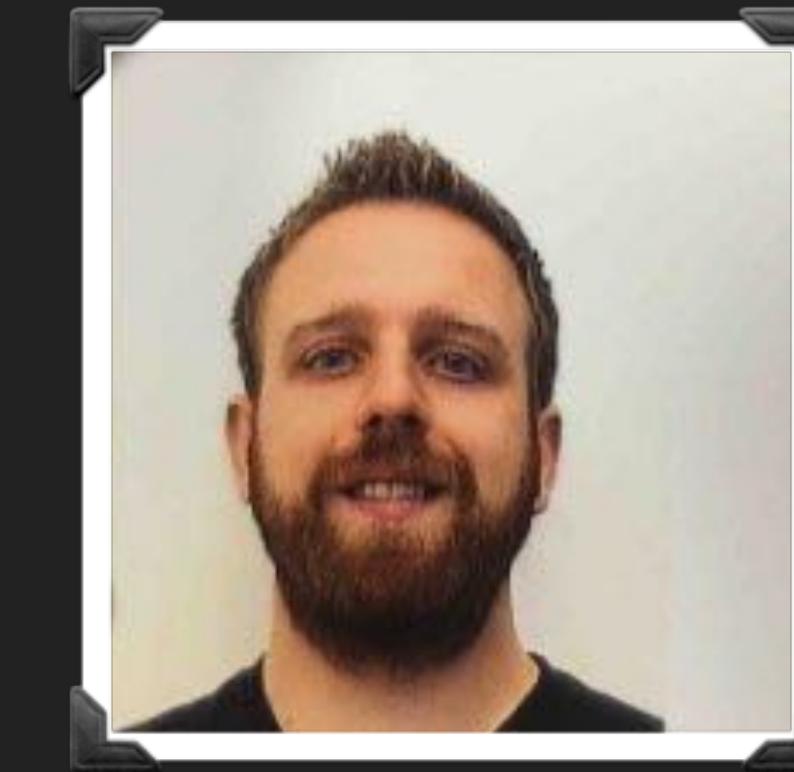
```
// Access the value, given the key  
WeakMap.prototype.get = function(key) {  
    return key._some_random_property_name;  
};
```

The solution for Ember Apps

- ▶ **ember-state-services**
- ▶ Built on top of WeakMap
- ▶ **StateFactory** - creates draft state objects
- ▶ **stateFor** - computed property that returns a particular state



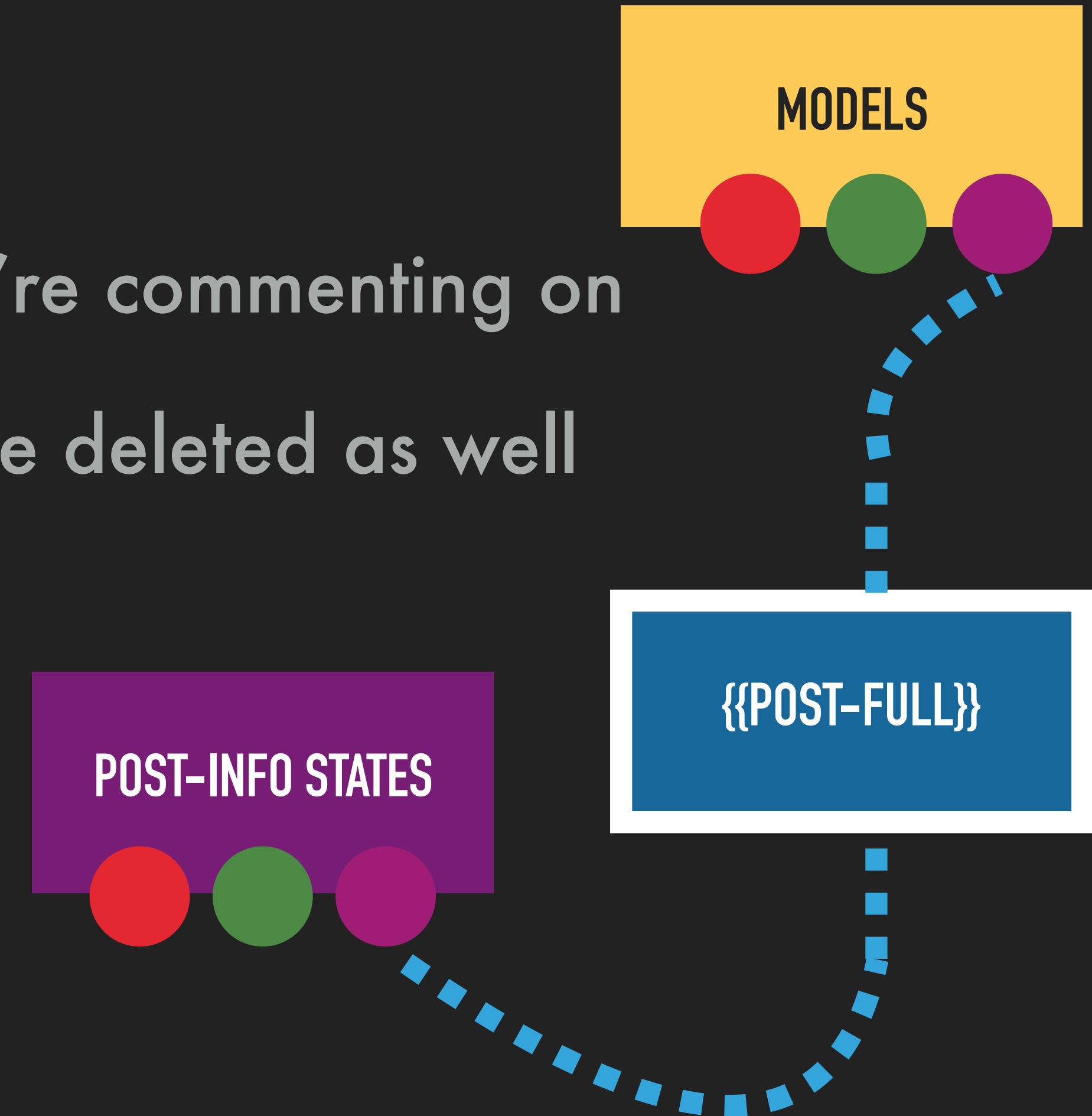
@thoov



@stefanpenner

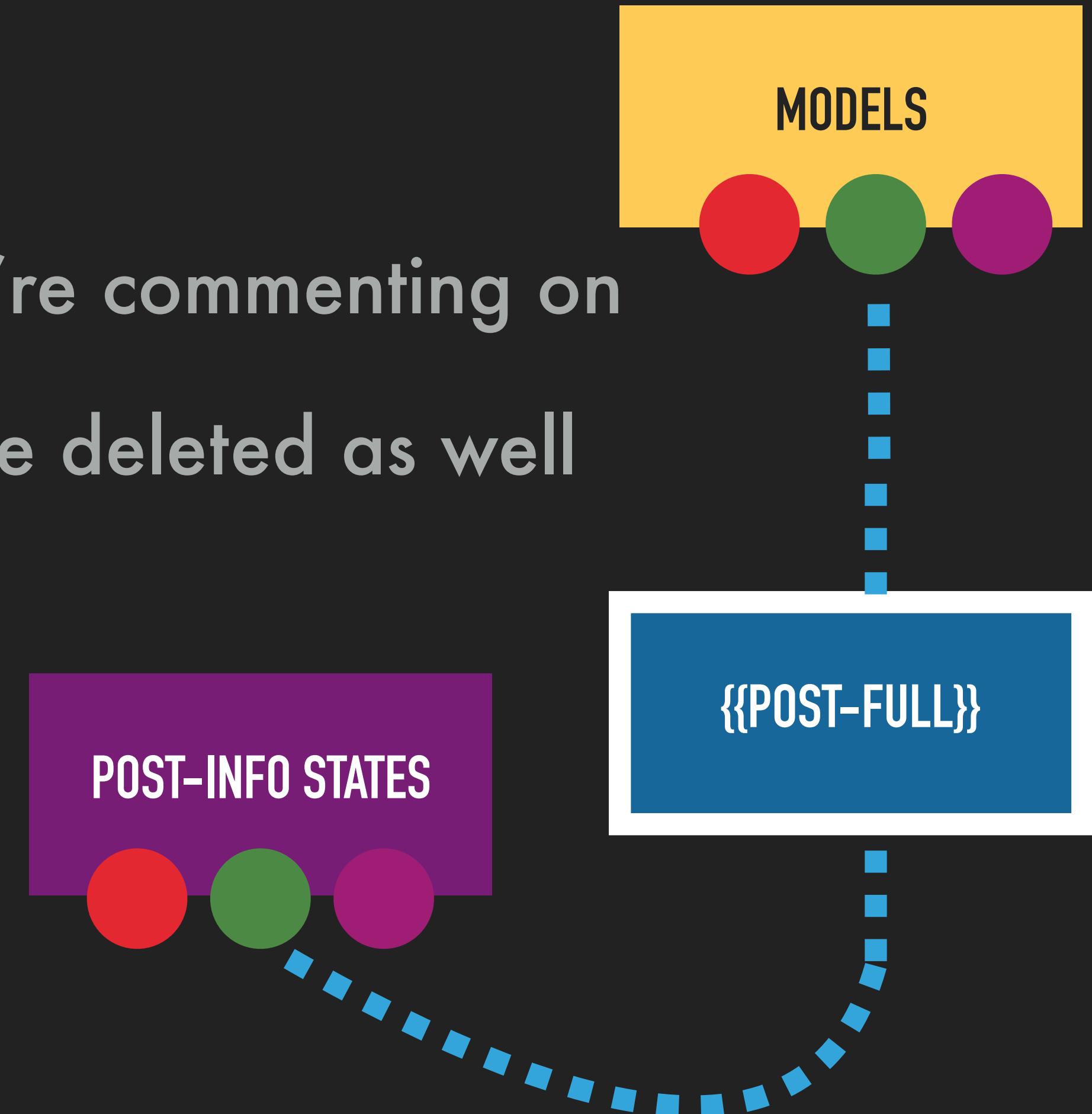
Enhancing our App

- ▶ They must “stick” with the post they’re commenting on
- ▶ If we delete a post, the draft must be deleted as well
- ▶ No leaks



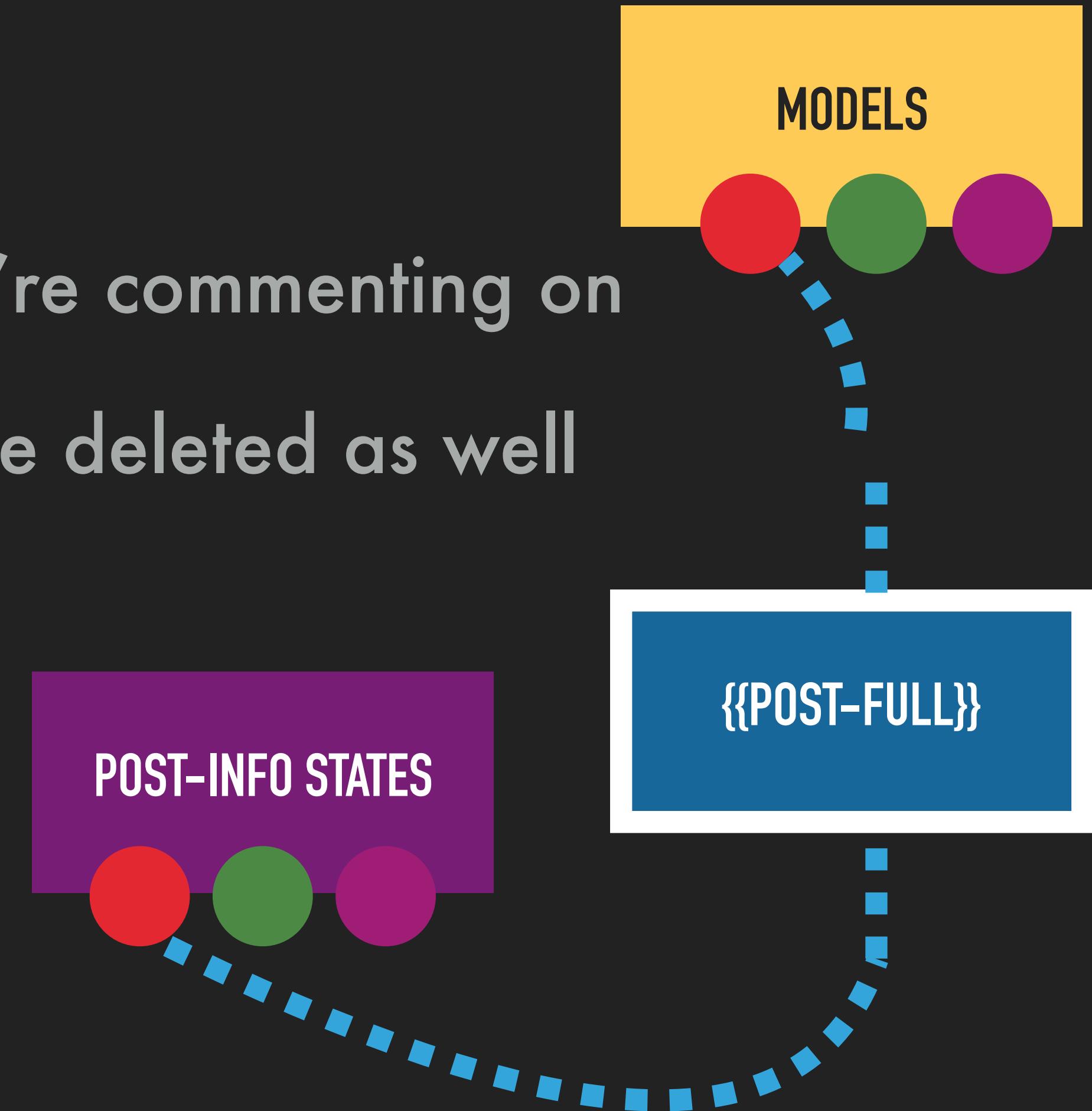
Enhancing our App

- ▶ They must “stick” with the post they’re commenting on
- ▶ If we delete a post, the draft must be deleted as well
- ▶ No leaks



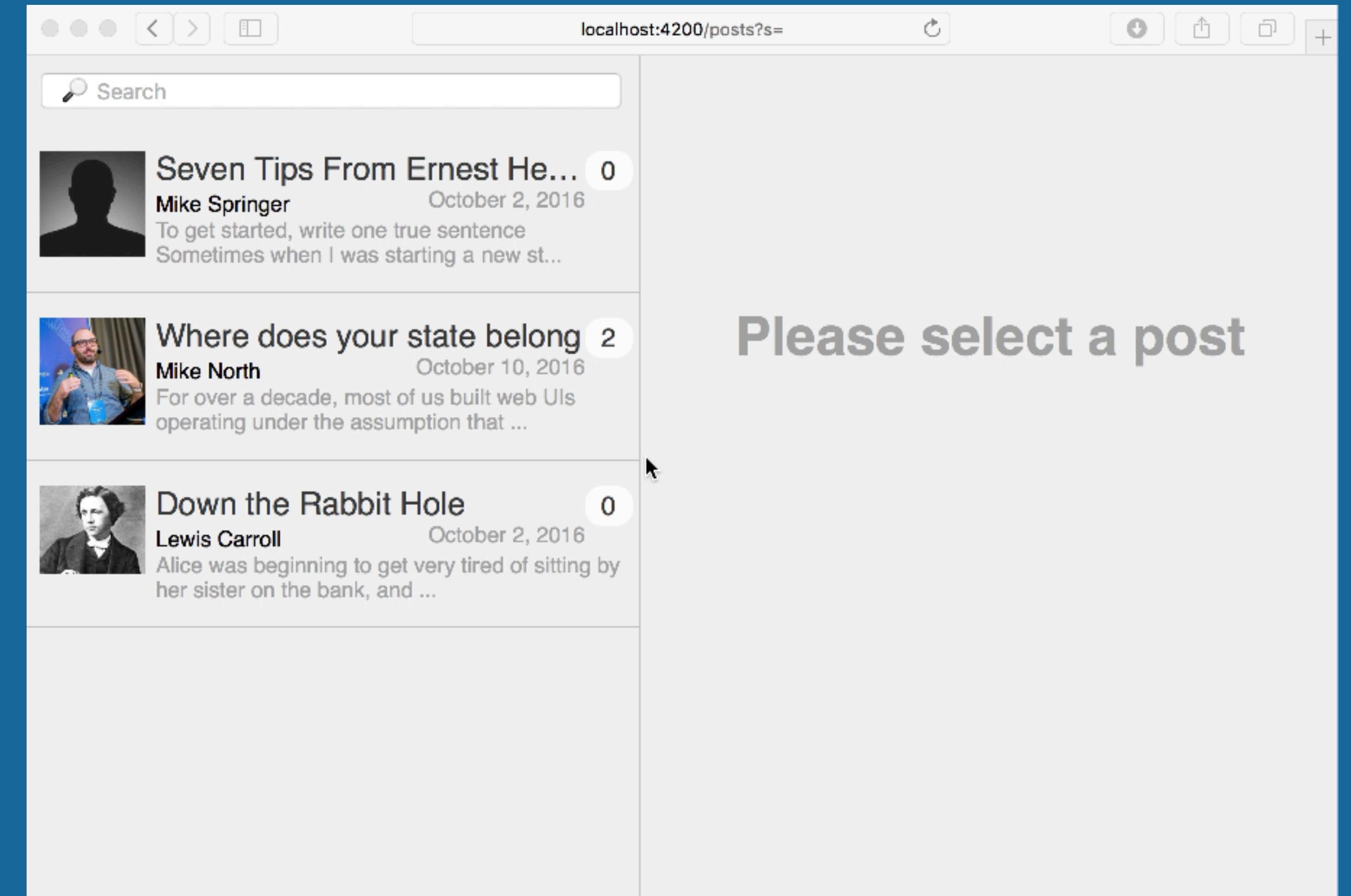
Enhancing our App

- ▶ They must “stick” with the post they’re commenting on
- ▶ If we delete a post, the draft must be deleted as well
- ▶ No leaks



Draft Comments

- ▶ Using ember-state-services, allow comments to be “saved” even if we navigate away from (and back to) a post
- ▶ Show an indication in the list of posts whether there’s draft data present
- ▶ Avoid memory leaks



The screenshot shows a dark-themed code editor interface with the following details:

- Top Bar:** Includes standard OS X window controls (red, yellow, green) and a tab labeled "post-info.js - statetalk".
- Left Sidebar (Explorer):** A tree view of the project structure:
 - OPEN EDITORS:** Shows "post-info.js" (highlighted).
 - STATES:** Shows "app" (highlighted), "adapters", "components", "controllers", "helpers", "mixins", "models", "routes", "serializers", "states" (highlighted), "post-info.js" (highlighted), "styles", and "templates".
 - Other Directories:** "bower_components", "config", "dist", "node_modules", and "public".
- Right Panel:** The main code editor area displays the contents of "post-info.js":

```
1 import Ember from 'ember';
2
3 const PostInfoState = Ember.Object.extend({});
4
5 PostInfoState.reopenClass({
6     initialState/*instance*/ {
7         return {
8             body: ''
9         }
10    }
11 });
12
13 export default PostInfoState;
```
- Bottom Status Bar:** Shows "master*" (branch), "0 1↑ 0 0 ▲ 1" (git status), "Ln 13, Col 30" (cursor position), "Spaces: 4" (text width), "UTF-8" (encoding), "LF" (line ending), "JavaScript" (language), and a smiley face icon.

post-full.hbs - statetalk

post-full.js

```
1 import Ember from 'ember';
2 import stateFor from 'ember-state-services/stat...
3
4 const { Component } = Ember;
5
6 export default Component.extend({
7   classNames: ['post-full'],
8   postInfo: stateFor('post-info', 'model')
9 });
10
```

post-full.hbs

```
1 + <div class="title-row">...
26   </div>
27
28
29   {{#if model.featuredImageUrl}}
30   + <div class="feature-image-row">...
32     </div>
33   {{/if}}
34
35   + <p class="body">...
37     </p>
38
39   <hr>
40
41   + {{#each model.comments as |comment|}}...
43   + {{else}}...
47   {{/each}}
48   + {{textarea value=postInfo.body class='repl...
49
50   + <div class="post-actions">...
52     </div>
53
```

master* 0↓ 1↑ 0 0 ▲ 0

Ln 48, Col 32 Spaces: 2 UTF-8 LF Handlebars

post-tile.js - statetalk

post-tile.js x post-tile.hbs

```
1 import Ember from 'ember';
2 import stateFor from 'ember-state-services/state-for';
3
4 const { Component } = Ember;
5
6 export default Component.extend({
7   classNames: ['post-tile'],
8   postInfo: stateFor('post-info', 'model')
9 });
10
```

4

master* 0↓ 1↑ 0 0 ▲ 0

Ln 8, Col 41 Spaces: 2 UTF-8 LF JavaScript

post-tile.hbs - statetalk

post-tile.js post-tile.hbs x

1 {{#link-to "posts.show" model.id}}

2 <div class="author-image">

3

4 </div>

5 <div class="post-tile-content">

6

7 {{model.comments.length}}

8

9 <h3>{{model.title}}</h3>

10

11 {{model.authorName}}

12

13

14 {{moment-format model.createdAt "LL"}}

15

16 <div class="post-tease">

17 {{tease model.body chars=80}}...

18 </div>

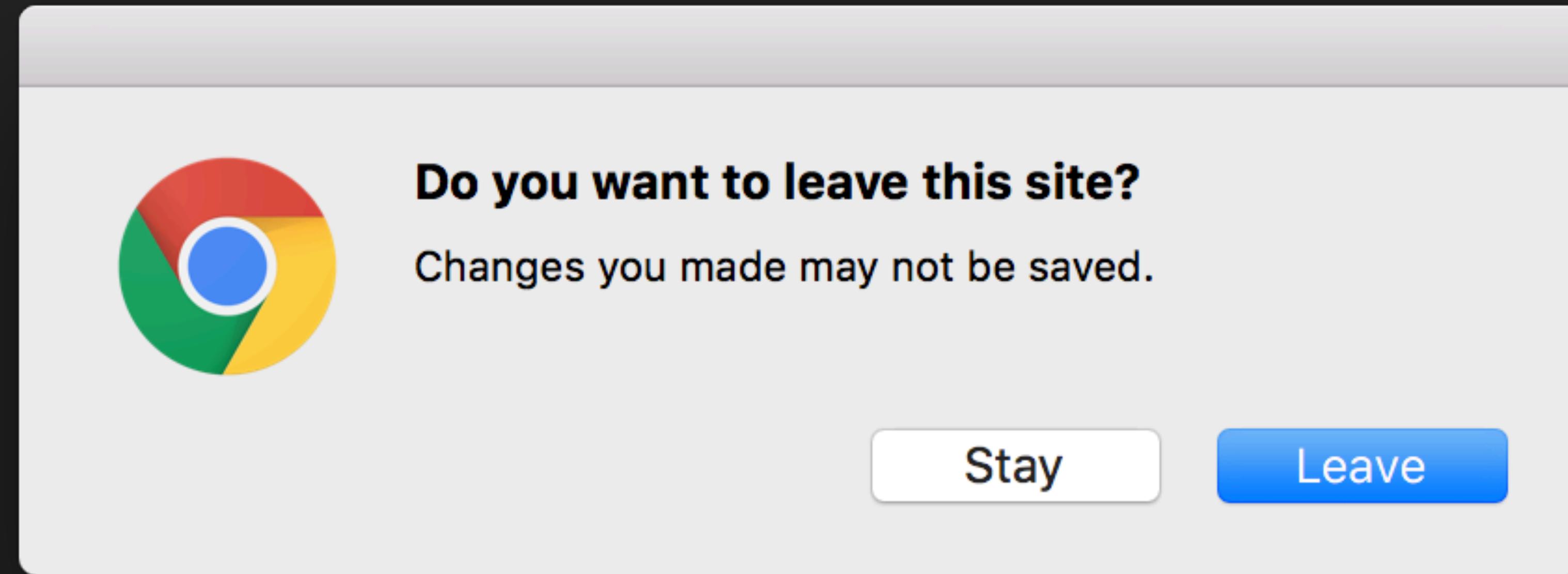
19 </div>

20 {{/link-to}}

4

Ln 1, Col 1 Spaces: 2 UTF-8 LF Handlebars

Making drafts even better



[ember-state-services/issues/64](#)

Persisted State

Persisted State

Stored in a long-living medium
for a long time

**Core
Concept**

Examples

- ▶ Anything stored in/on a...
 - ▶ Database/Disk
 - ▶ Local storage
 - ▶ Third party API
-
- ▶ Lives longer than a session
 - ▶ Tracked over time
 - ▶ Seen by many users

HOW TO IDENTIFY

Managing Comments in our App

- ▶ **POST** /api/posts/:post_id/comments for create
 - ▶ Draft cleared if successful
- ▶ **DELETE** /api/posts/:post_id/comments/:comment_id for delete
- ▶ Must work in high-latency environments

Customizing API URLs with Ember-Data

- ▶ URLs - an adapter concern
- ▶ New hooks for per-operation customization

```
urlForCreateRecord(    modelName, snapshot){ } // One  
urlForDeleteRecord(id, modelName, snapshot){ } // One  
urlForFindRecord  (id, modelName, snapshot){ } // One  
urlForUpdateRecord(id, modelName, snapshot){ } // One  
urlForDeleteRecord(id, modelName, snapshot){ } // One  
urlForfindAll      (modelName, snapshot){ } // List  
urlForQueryRecord (modelName, snapshot){ } // One  
urlForQuery        (modelName, snapshot){ } // List
```

Working with Snapshots

```
snapshot.attr('title'); // attribute  
snapshot.belongsTo('author'); // relationship  
snapshothasMany('comment');
```

Enhancing our Basic App

- ▶ POST to API for create
 - ▶ Draft cleared if successful
- ▶ DELETE to API for delete
- ▶ Must work in high-latency

POST/SHOW

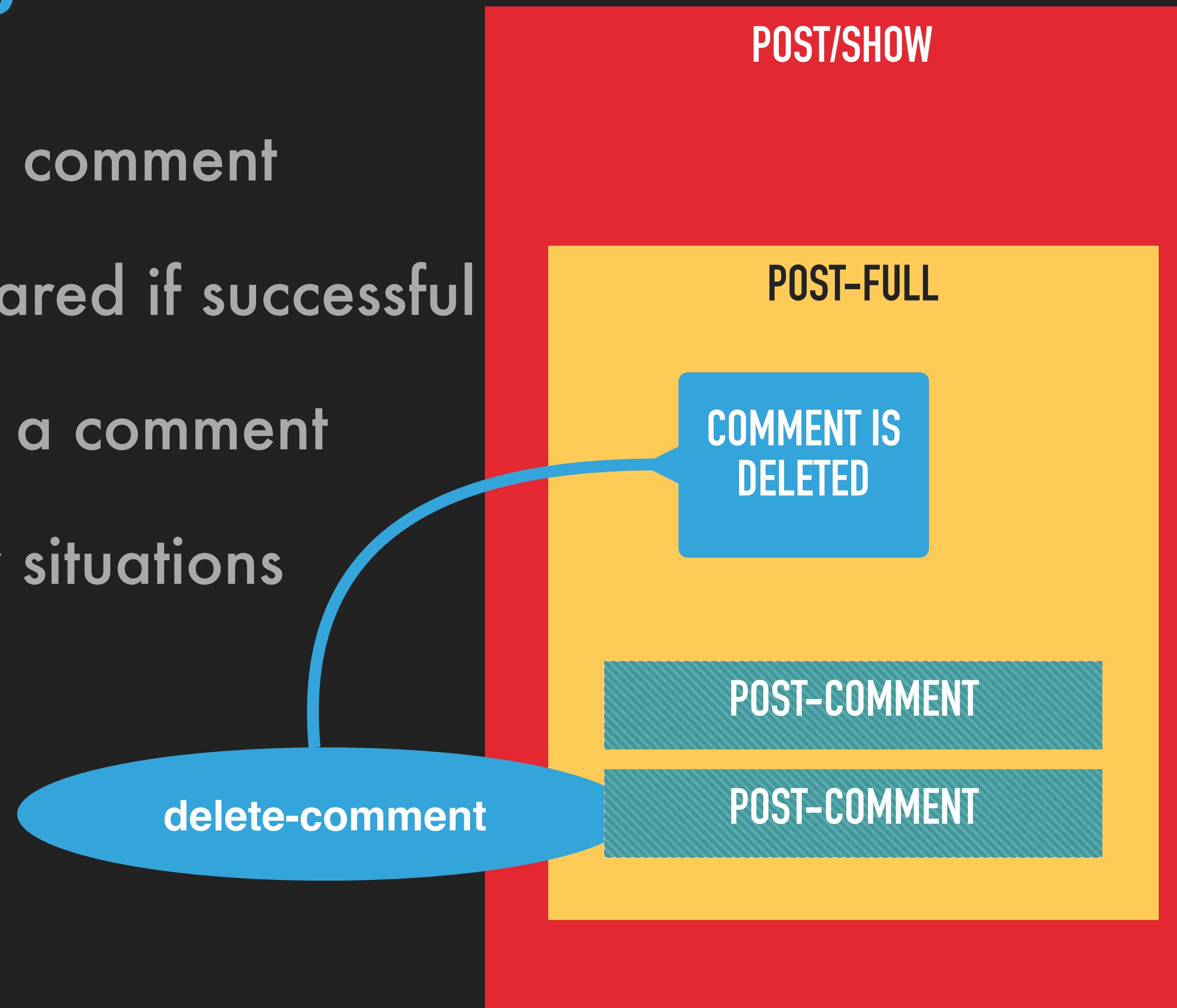
{{ POST-FULL }}

{{ POST-COMMENT }}

{{ POST-COMMENT }}

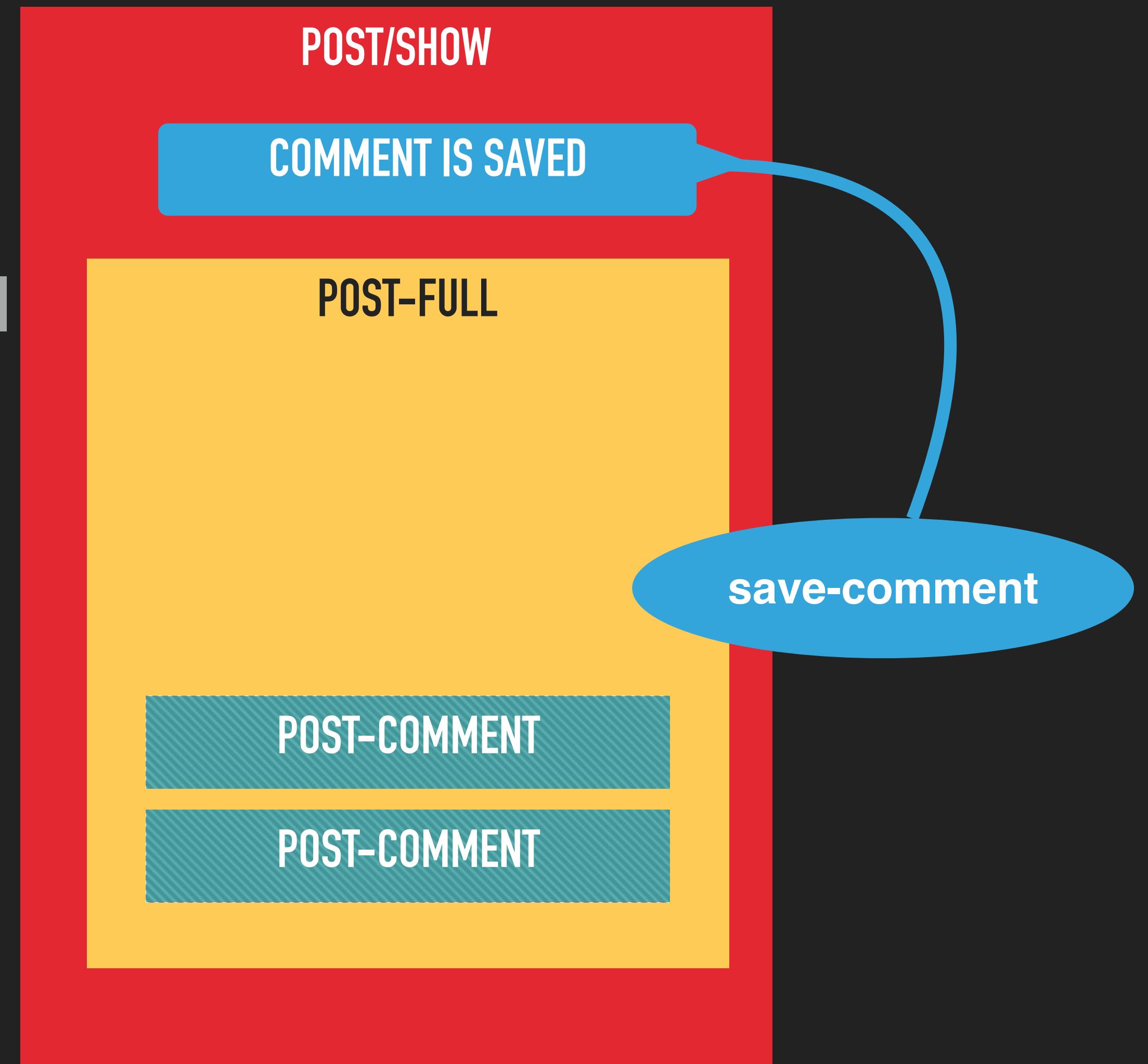
Enhancing our Basic App

- ▶ POST to API for creating a comment
 - ▶ Draft should only be cleared if successful
- ▶ DELETE to API for deleting a comment
- ▶ Must work for high-latency situations



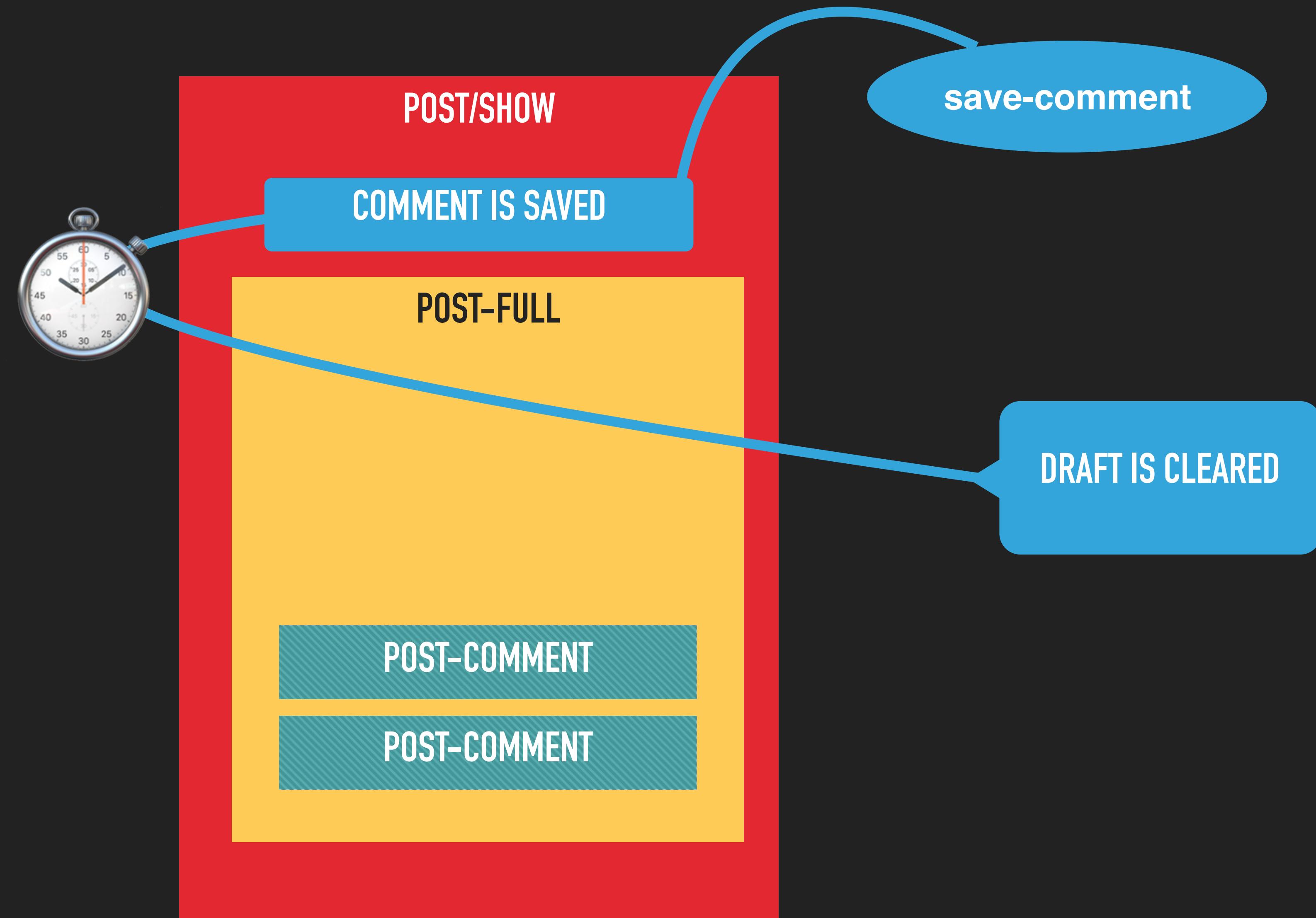
Enhancing our Basic App

- ▶ POST to API for creating a comment
 - ▶ Draft should only be cleared if successful
- ▶ DELETE to API for deleting a comment
- ▶ Must work for high-latency situations



Enhancing our Basic App

- ▶ POST to API for creating a comment
 - ▶ Draft should only be cleared if successful
- ▶ DELETE to API for deleting a comment
- ▶ Must work for high-latency situations



Persisted State - Best Practices

- ▶ Consider that things may change during async
- ▶ Build simple objects and **then** createRecord
- ▶ Consider in-flight records

Persisted State - Best Practices

- ▶ Double clicking “delete”
- ▶ Double clicking “reply”

Uncaught [internal-model.js:502](#)
EmberError {description: undefined, fileName:
undefined, lineNumber: undefined, message: "Attempted
to handle event `deleteRecord` on <state...ber689:35>
while in state root.deleted.inFlight. ", name:
"Error"...}

Be brief

It wasn't by accident that the Gettysburg address was so short. The laws of prose writing are as immutable as those of flight, of mathematics, of physics.

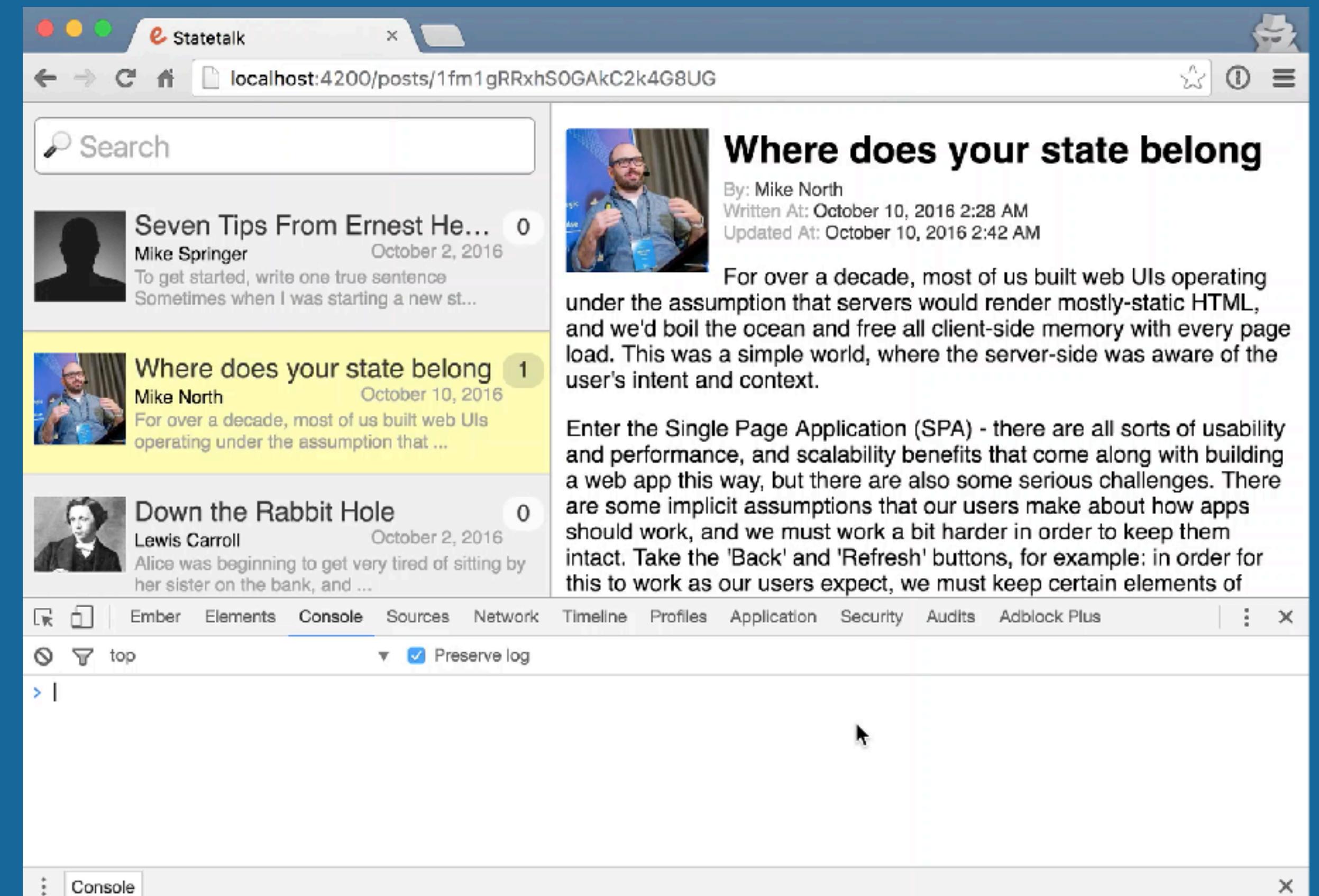
The original, unabridged version of the blog post can be found on the [Open Culture](#) website.

No comments yet

Reply

Persisting Comments

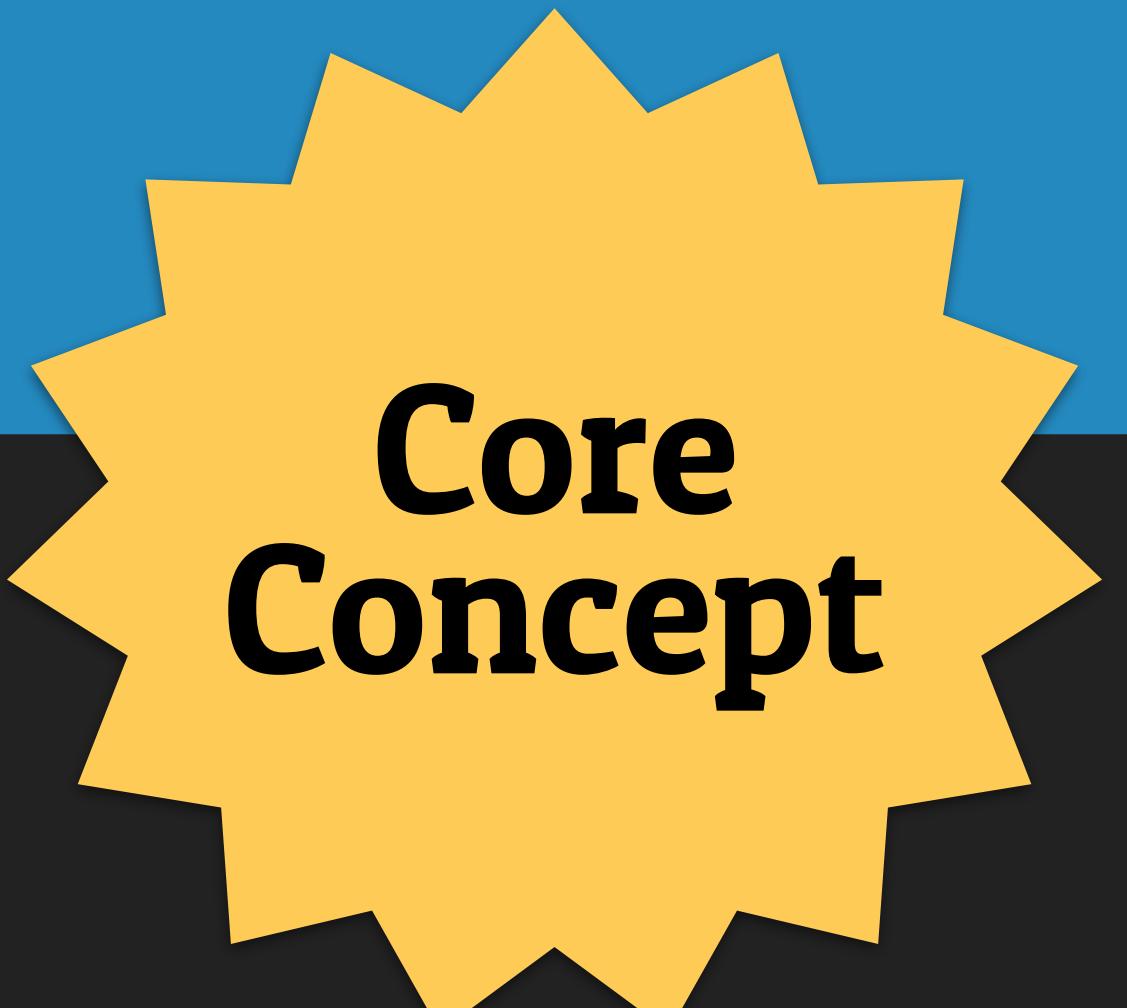
- ▶ Persist comments to the API, associating with the current user, and appropriate post
- ▶ Make sure the draft gets cleared as appropriate (set to blank is fine)
- ▶ This is an async operation. Consider what might happen with very high latency
- ▶ Add a new attribute (gravatarUrl) to the comment model & component



UI State

UI State

Short-lived, useful only “in the moment”, And relevant only to the UI



**Core
Concept**

Examples

- ▶ Accordion expansion
- ▶ Scroll position
- ▶ Text selection & Focus
- ▶ Matthew Beale is typing...

How To Identify

- ▶ Short lived, only meaningful in the moment
- ▶ Easiest to re-create
- ▶ Potentially harmful if persisted/restored

post-full.hbs - statetalk

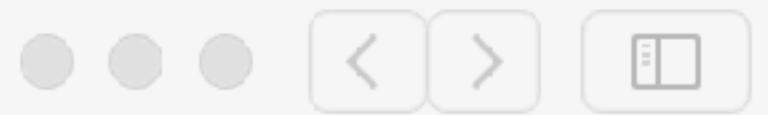
post-full.js post-full.hbs 205

1 <div class="title-row">
2 {{titlecase model.title}}</h1>
4
5 {{#if detailsOpen}}
6 + <div onClick={{action (mut detailsOpen) false}} class="metadata-row">
7 </div>
8 {{else}}
9
10 details...
11
12 {{/if}}
13
14 </div>
15
16
17 {{#if model.featuredImageUrl}}
18 <div class="feature-image-row">
19
20 </div>
21 {{/if}}
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42

master ⌘ 0↓1↑ ✖ 0 ▲ 0 Ln 6, Col 48 Spaces: 2 UTF-8 LF Handlebars 😊

post-full.js × post-full.hbs

```
1 import Ember from 'ember';
2 import stateFor from 'ember-state-services/state-for';
3
4 const { Component } = Ember;
5
6 export default Component.extend({
7   classNames: ['post-full'],
8   postInfo: stateFor('post-info', 'model'),
9   detailsOpen: false,
10  actions: {
11    deleteComment(commentRecord) {
12      commentRecord.destroyRecord();
13    },
14    persistCommentDraft() {
15      let commentBody = this.get('postInfo.body');
16      let post = this.get('model');
17      this.attrs['save-comment'](post, commentBody);
18    }
19  }
20 });
21
```



Search



Seven Tips From Ernest He... 3

Mike Springer

October 2, 2016

To get started, write one true sentence

Sometimes when I was starting a new st...



Where does your state belong 1

Mike North

October 10, 2016

For over a decade, most of us built web UIs

operating under the assumption that ...



Down the Rabbit Hole 0

Lewis Carroll

October 2, 2016

Alice was beginning to get very tired of sitting by

her sister on the bank, and ...

Please select a post

Keep component
life-cycle in mind

post-full.js x post-full.hbs

```
1 import Ember from 'ember';
2 import stateFor from 'ember-state-services/state-for';
3
4 const { Component } = Ember;
5
6 export default Component.extend({
7   classNames: ['post-full'],
8   postInfo: stateFor('post-info', 'model'),
9   detailsOpen: false,
10  actions: {
11    deleteComment(commentRecord) {
12      commentRecord.destroyRecord();
13    },
14    persistCommentDraft() {
15      let commentBody = this.get('postInfo.body');
16      let post = this.get('model');
17      this.attrs['save-comment'](post, commentBody);
18    }
19  }
20});
21
```

post-info.js x

1 import Ember from 'ember';
2
3 // jscs:disable disallowDirectPropertyAccess
4 const PostInfoState = Ember.Object.extend({});
5 // jscs:enable disallowDirectPropertyAccess
6
7 PostInfoState.reopenClass({
8 initializeState(*instance*) {
9 return {
10 body: '',
11 detailsOpen: false
12 };
13 }
14 });
15
16 export default PostInfoState;

post-info.js - statetalk

210

master ⌂ 0↓1↑ ✘ 0 ▲ 0

Ln 11, Col 25 Spaces: 2 UTF-8 LF JavaScript 😊

```
1 <div class="title-row">
2   {{titlecase model.title}}</h1>
4
5   {{#if detailsOpen}}
6   + <div onClick={{action (mut detailsOpen) false}} class="metadata-row">
7     </div>
8   {{else}}
9     <span onClick={{action (mut detailsOpen) true}}>
10       details...
11     </span>
12   {{/if}}
13
14 </div>
15
16
17 {{#if model.featuredImageUrl}}
18 <div class="feature-image-row">
19   
20 </div>
21 {{/if}}
22
23
24
25
26
```

Search

Seven Tips From Ernest He... 3

Mike Springer

October 2, 2016

To get started, write one true sentence

Sometimes when I was starting a new st...



Where does your state belong 1

Mike North

October 10, 2016

For over a decade, most of us built web UIs

operating under the assumption that ...



Down the Rabbit Hole 0

Lewis Carroll

October 2, 2016

Alice was beginning to get very tired of sitting by

her sister on the bank, and ...



Where does your state belong

details...

For over a decade, most of us built web UIs

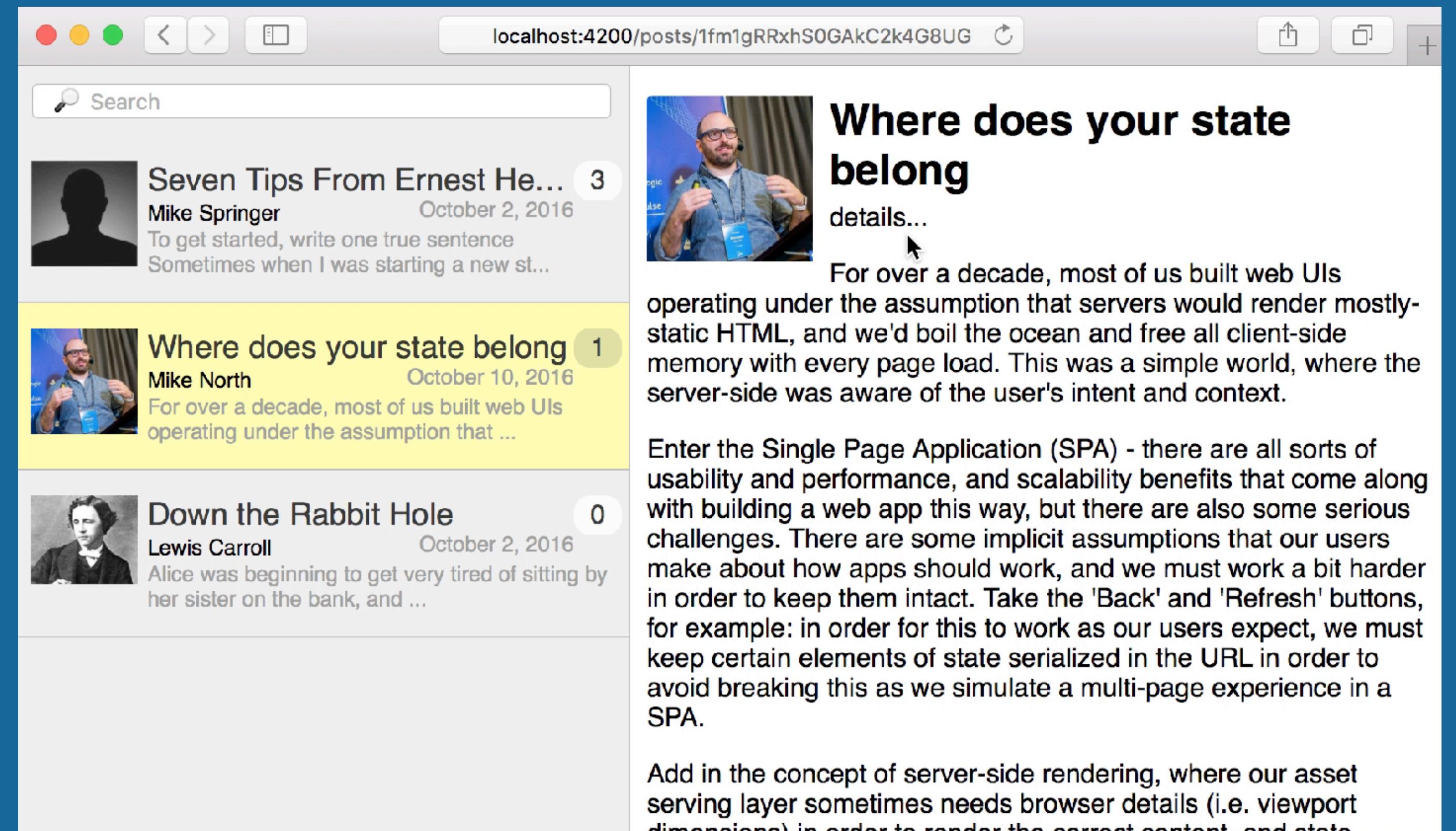
operating under the assumption that servers would render mostly-static HTML, and we'd boil the ocean and free all client-side memory with every page load. This was a simple world, where the server-side was aware of the user's intent and context.

Enter the Single Page Application (SPA) - there are all sorts of usability and performance, and scalability benefits that come along with building a web app this way, but there are also some serious challenges. There are some implicit assumptions that our users make about how apps should work, and we must work a bit harder in order to keep them intact. Take the 'Back' and 'Refresh' buttons, for example: in order for this to work as our users expect, we must keep certain elements of state serialized in the URL in order to avoid breaking this as we simulate a multi-page experience in a SPA.

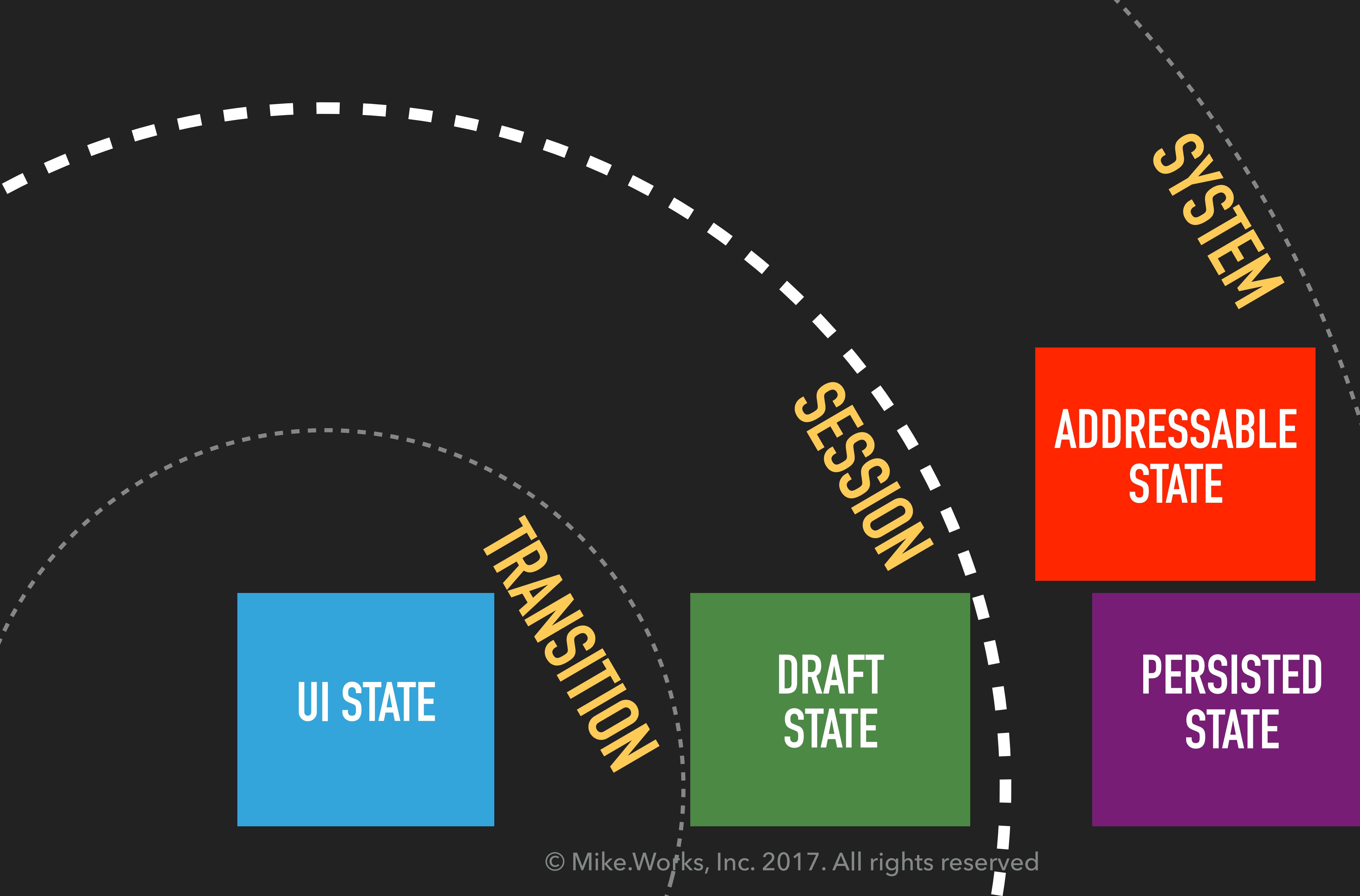
Add in the concept of server-side rendering, where our asset serving layer sometimes needs browser details (i.e. viewport dimensions) in order to render the correct content and state

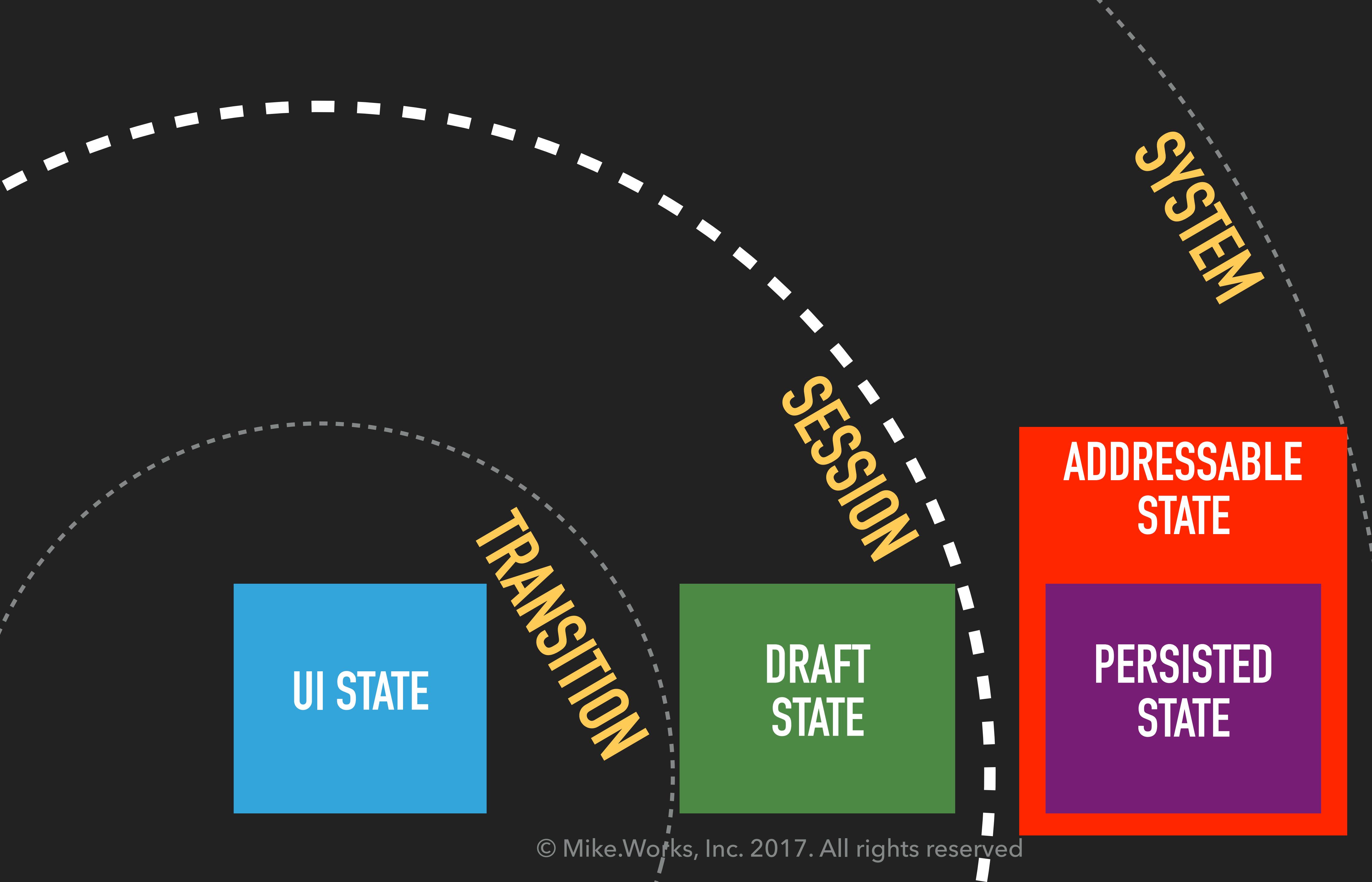
Collapsible Metadata

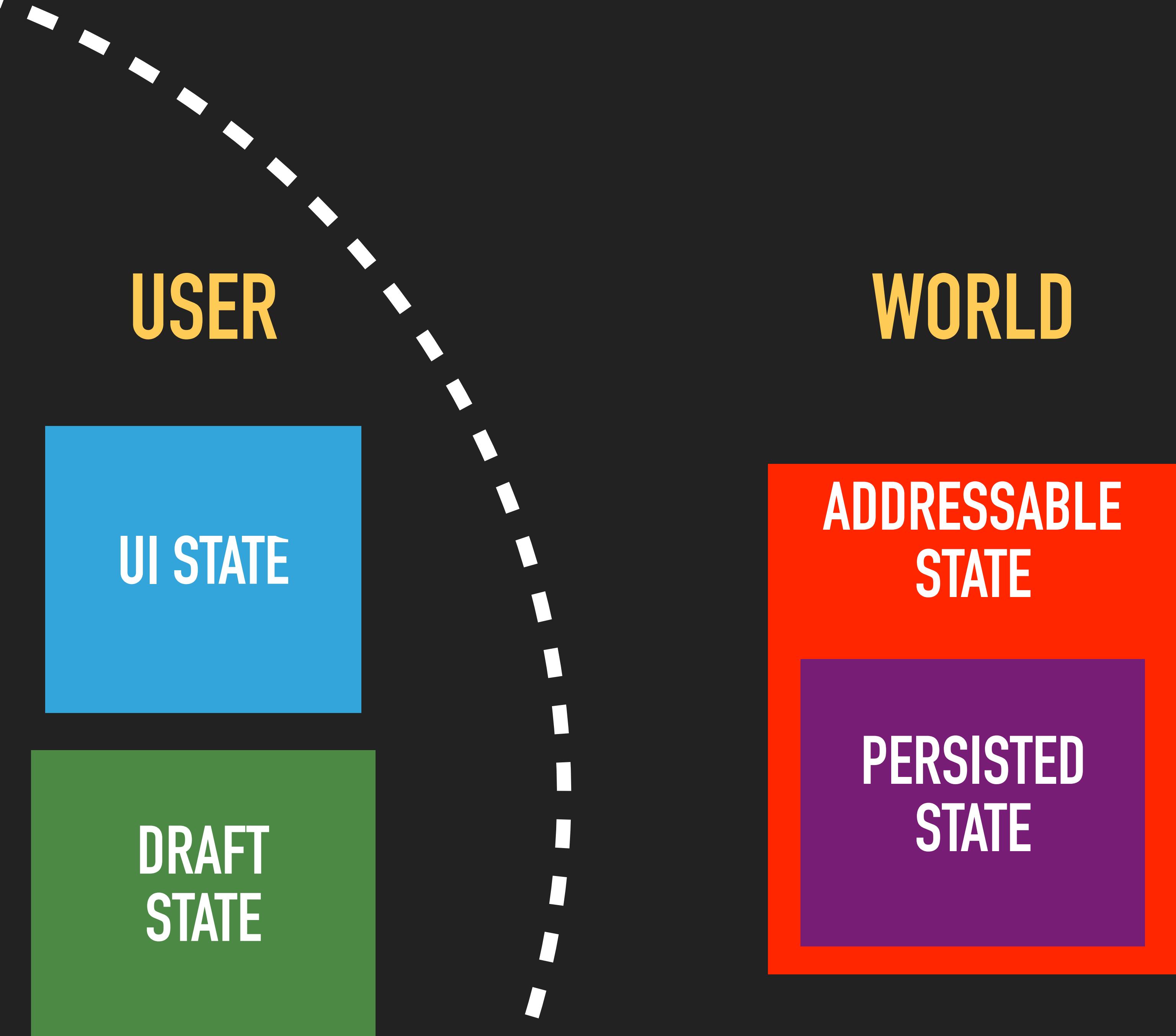
- ▶ Make the metadata area at the top of the post-full component hideable/showable
- ▶ expanded/collapsed state should “stick” to each record for the duration of our UI session
- ▶ Initial state should be: hidden

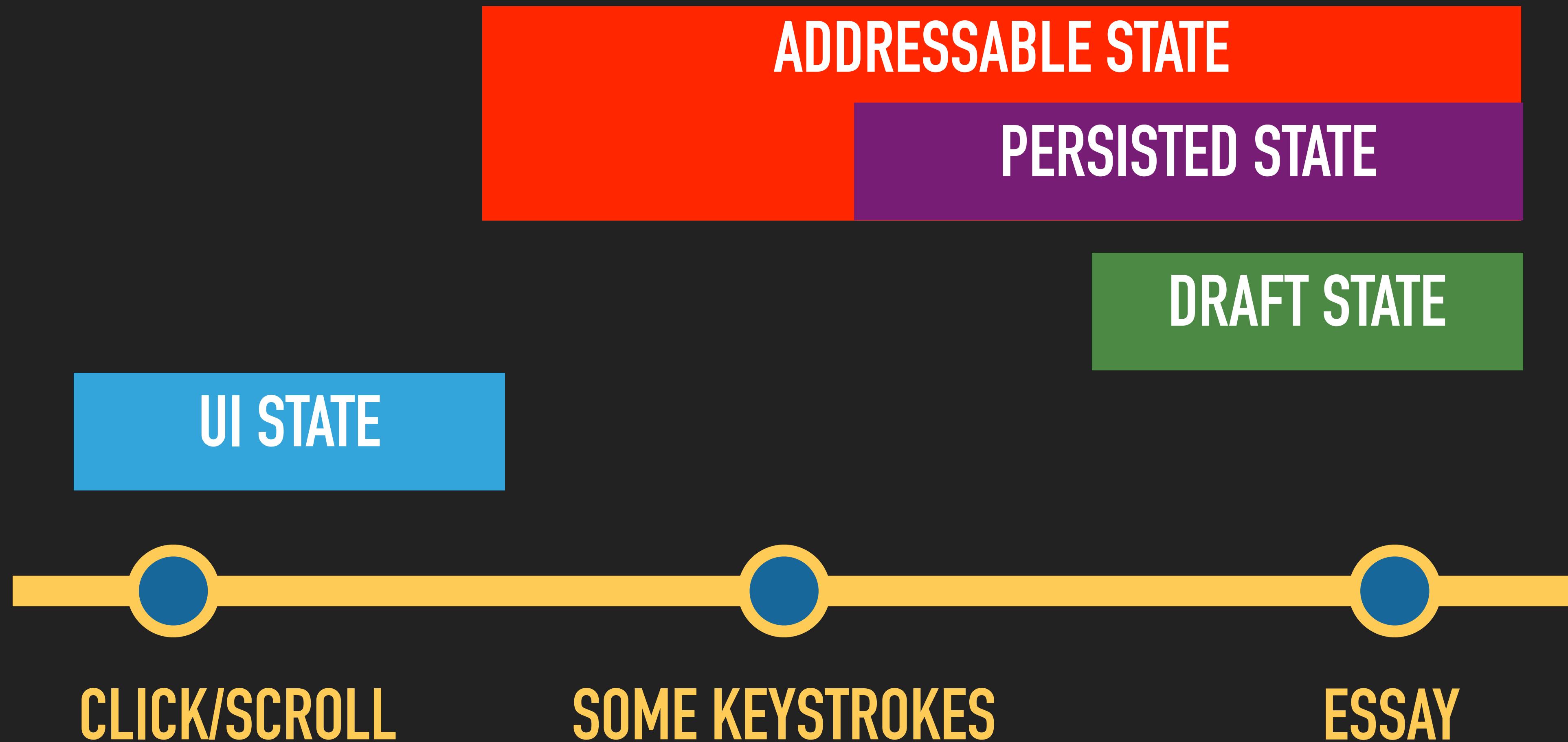


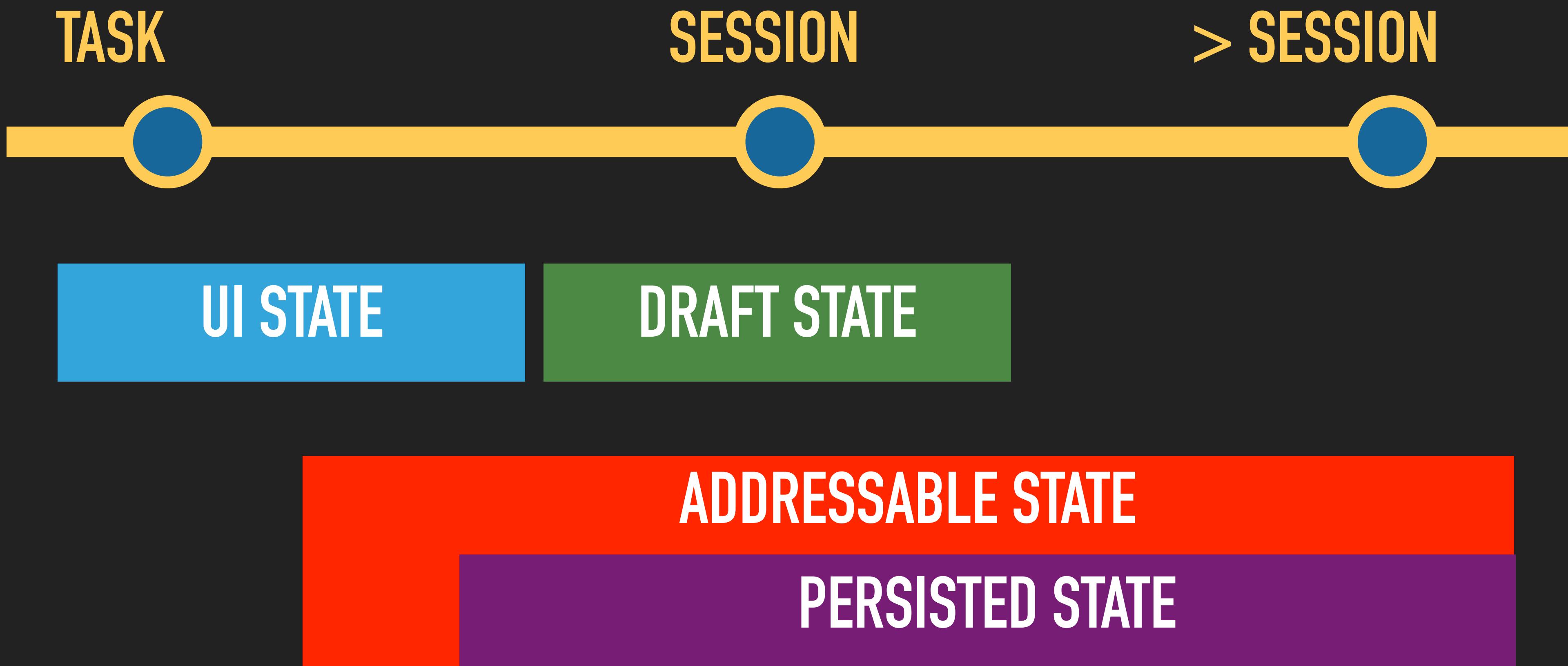














Concurrency

Parallelism vs Concurrency

- ▶ **Parallelism** - A lot of things happening at the same time
- ▶ **Concurrency** - Keeping organized about many long-running tasks
- ▶ Lately, a lot of excitement around Observables & Rx.js

```
Promise.then(() => {  
});
```

Parallelism vs Concurrency

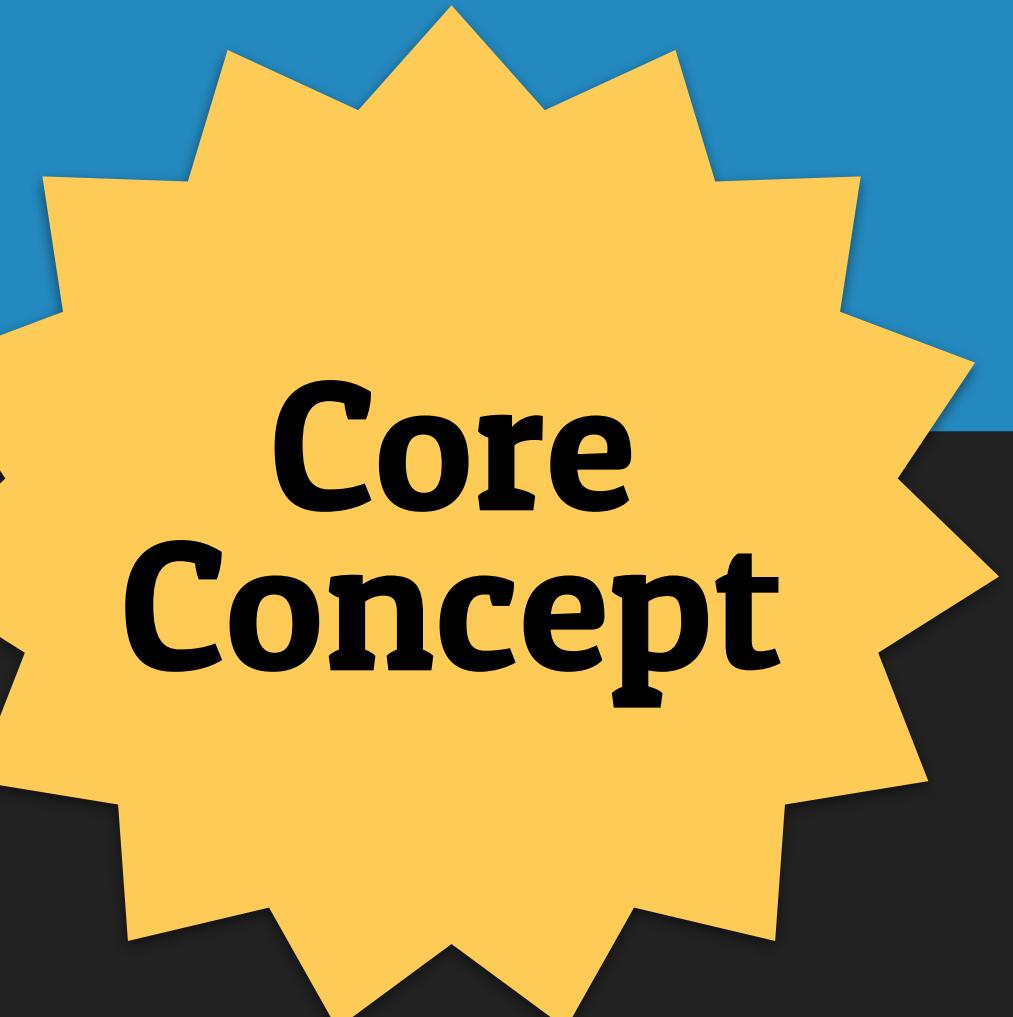
- ▶ Cancellation
- ▶ Restart
- ▶ Limiting concurrency
- ▶ Black-holing async stuff



```
Uncaught                                internal-model.js:502
EmberError {description: undefined, fileName:
undefined, lineNumber: undefined, message: "Attempted
to handle event `deleteRecord` on <state...ber689:35>
while in state root.deleted.inFlight. ", name:
"Error"...}
```

Ember-Concurrency Task

asynchronous, cancelable operations, bound to the lifetime of the object they live on



Defining a Task

```
import Ember from 'ember';
import { task } from 'ember-concurrency';

export default Ember.Component.extend({
  myTask: task(function * () {
    alert("hello!");
  })
});
```

Defining a Task

```
import Ember from 'ember';
import { task } from 'ember-concurrency';

export default Ember.Component.extend({
  myTask: task(function * () {
    alert("hello!");
  })
});
```

ES2015

JS

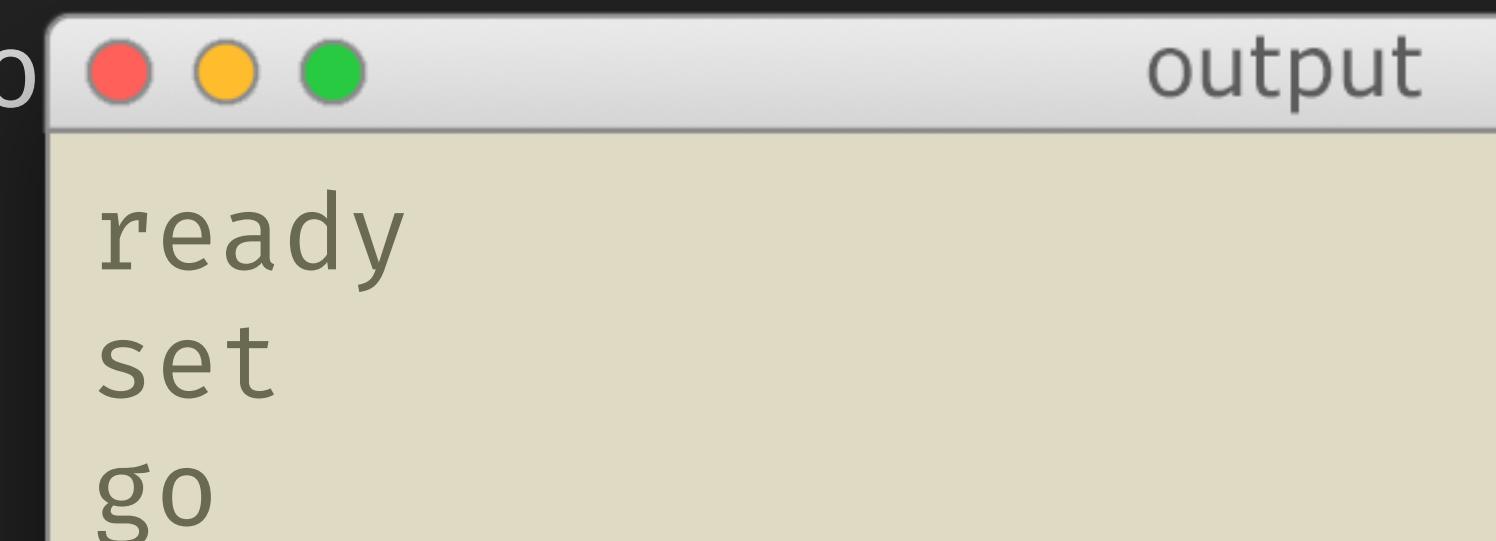
Iterators & Generators

Iterators

- ▶ Iterators allow access one item from a collection at a time, keeping track of current position
- ▶ the `next()` method is what's used to get the next item in the sequence.

```
function makeIterator(arr) {  
  var i = 0;  
  return {  
    next: () => i < arr.length ?  
      { value: arr[i++], done: false } :  
      { done: true }  
  };  
}
```

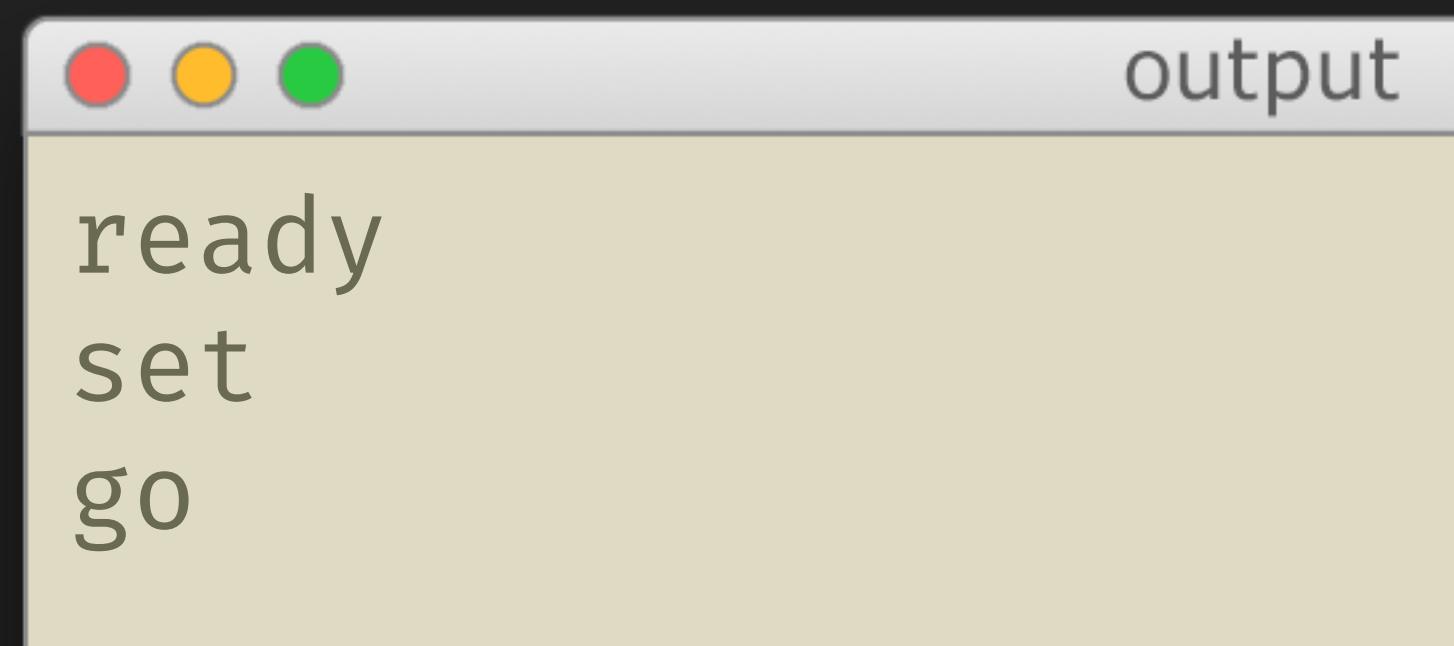
```
let l = ['ready', 'set', 'go'];  
let it = makeIterator(l)  
for (let p = it.next();  
     !p.done;  
     p = it.next()) {  
  console.log(p.value);  
}
```



Generators

- ▶ Define their own iterative algorithm, yielding each item in the sequence
- ▶ Use the `function*()` syntax
- ▶ Return an iterator
- ▶ State of the closure is preserved between `.next()` calls.

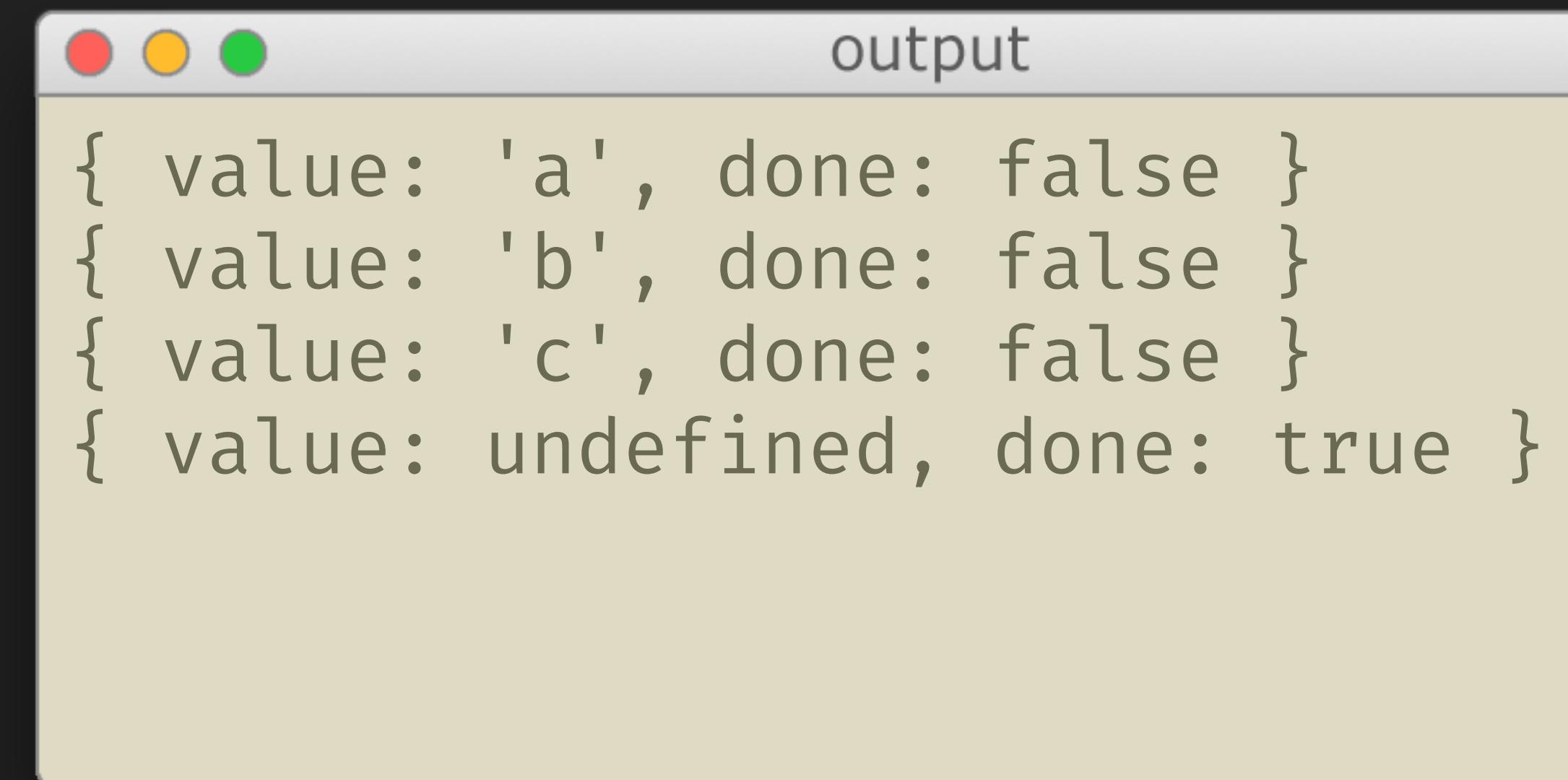
```
function* makeIterator(arr) {  
  var i = 0;  
  while(i < arr.length) {  
    yield arr[i++];  
  }  
}  
  
let l = ['ready', 'set', 'go'];  
let it = makeIterator(l)  
for (let p = it.next();  
     !p.done;  
     p = it.next()) {  
  console.log(p.value);  
}
```



Iterables

- ▶ Support iteration within a `for .. of` loop
- ▶ Requires implementation of the `Symbol.iterator` method
- ▶ Array and Map already support this!

```
let arr = ['a', 'b', 'c'];
let it = arr[Symbol.iterator]();
console.log(it.next());
console.log(it.next());
console.log(it.next());
console.log(it.next());
```



Iterables - Defining our own iterable

- ▶ Generator function makes this very simple

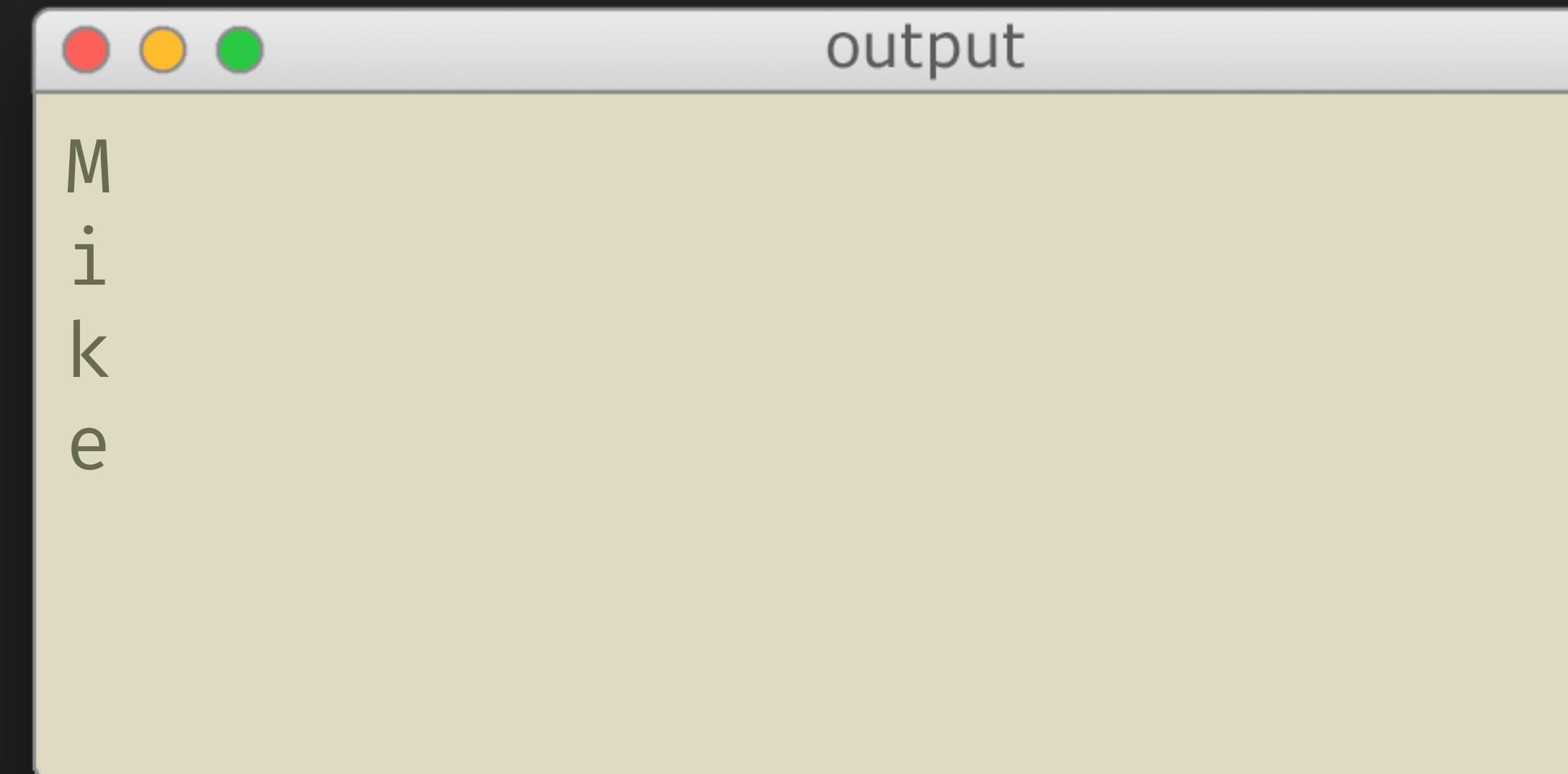
```
let mike = {  
  [Symbol.iterator]: function*() {  
    yield 'M';  
    yield 'i';  
    yield 'k';  
    yield 'e';  
  }  
}  
  
for (let m of mike) {  
  console.log(m);  
}
```



Using Iterables - `yield*`

- ▶ In generator functions, the `yield*` keyword will yield each value of an iterable, one by one.

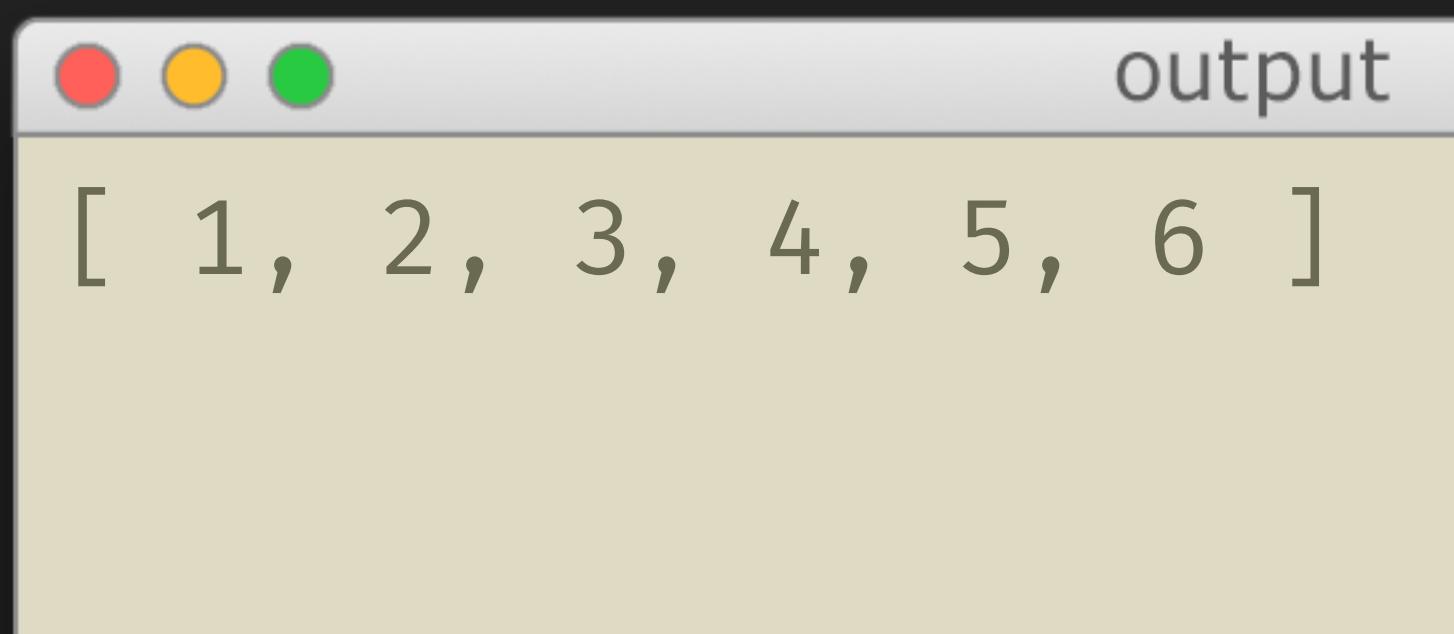
```
let mike = {
  [Symbol.iterator]: function*() {
    yield* 'Mike';
  }
}
for (let m of mike) {
  console.log(m);
}
```



Using Iterables - Destructuring Assignment

- ▶ Restructuring assignment works with any iterable, not just arrays!

```
let nums = {  
  [Symbol.iterator]: function*() {  
    yield* [1, 2, 3];  
    yield 4;  
    yield* [5, 6];  
  }  
}  
  
console.log([...nums]);
```



Defining a Task

- ▶ Yielding a promise will pause task execution until it's resolved

```
waitAFewSeconds: task(function * () {  
    this.set('status', "Gimme one second ...");  
    yield timeout(1000);  
    this.set('status', "Gimme one more second ...");  
    yield timeout(1000);  
    this.set('status', "OK, I'm done.");  
})
```

Editing Comments

- ▶ The “update” API call for comments takes a long time

PUT /api/posts/:post_id/comments/:id

- ▶ We want to avoid double-posting

Be brief

It wasn't by accident that the Gettysburg address was so short. The laws of prose writing are as immutable as those of flight, of mathematics, of physics.

The original, unabridged version of the blog post can be found on the [Open Culture](#) website.

No comments yet

|

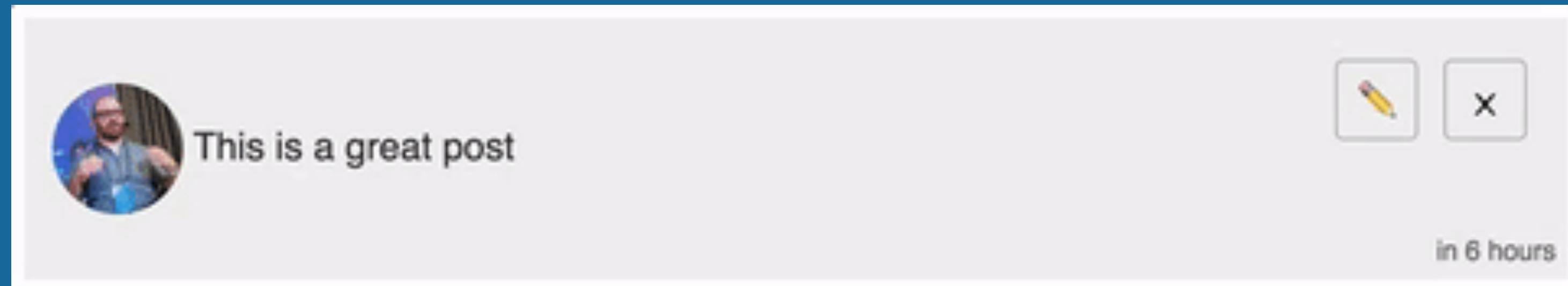
Reply

Editing Comments

- ▶ The “update” API call for comments takes a long time

```
PUT /api/posts/:post_id/comments/:id
```

- ▶ We want to avoid double-posting
- ▶ Here's the desired UX. Extra credit for “cancel” edit button



- ▶ Update app.css from GitHub latest master commit



Animation

Liquid-Fire

- ▶ Ember's official animation library
- ▶ Modifies transition logic between views
- ▶ Built on `velocity.js`
- ▶ Extensible - you can write your own animations

Liquid-Fire: route transitions

app/transitions.js

```
export default function() {
  this.transition(
    this.fromRoute('index'),
    this.toRoute('posts'),
    this.use('toLeft'),
    this.reverse('toRight')
  );
};
```

app/templates/*

```
{{liquid-outlet}}
```

Liquid-Fire: Bind Transitions

app/transitions.js

```
export default function() {
  this.transition(
    this.childOf('.my-thing'),
    this.use('toUp')
  );
};
```

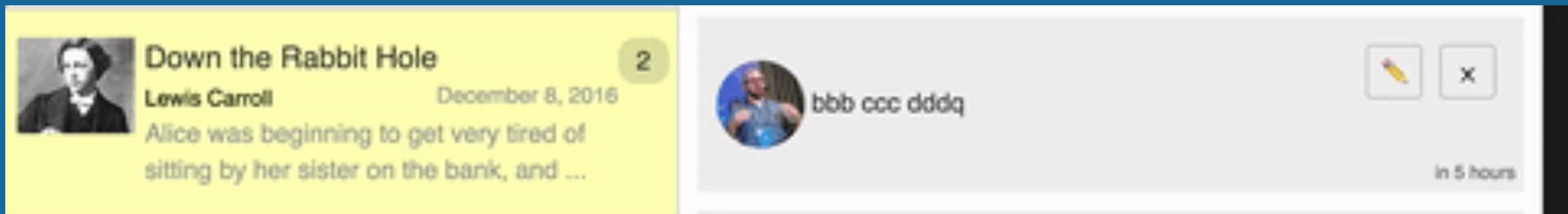
app/templates/*

```
{{liquid-bind x}}
```

Animated Comment Count

15

- ▶ in the {{post-tile}} component, animate comment count changes with {{liquid-bind}}





Blueprints

Code Generation in Ember-cli

- ▶ Generating code with ember-cli leaves you with stubs for test coverage
- ▶ Generators are generally reversible
- ▶ They can also touch other files and do other things
- ▶ Arguments!!!

`ember g <thing> <name>`

`ember g route home --path=“/homepage”`

Code Generation in Ember-CLI

- ▶ A blueprint is a folder
- ▶ Inside is `index.js` – this is the entry point of the blueprint
- ▶ You can customize lots of things here

```
.  
  └── files  
    |   └── _root_  
    |       └── _path_  
    |           └── _name_.js  
    └── index.js
```

```
module.exports = {
  locals: function (options) {
    // Return custom template variables here.
    return {};
  },
  normalizeEntityName: function (entityName) {
    // Normalize and validate entity name here.
    return entityName;
  },
  fileMapTokens: function (options) {
    // Return custom tokens to be replaced in your files
    return {
      __token__: function (options) {
        // logic to determine value goes here
        return 'value';
      }
    }
  },
  beforeInstall: function (options) { },
  afterInstall: function (options) { },
  beforeUninstall: function (options) { },
  afterUninstall: function (options) { }
};
```

```
locals: function(options) {
  // Return custom template variables here.
  return {};
},
```

--path

```
normalizeEntityName: function(entityName) {
  // Normalize and validate entity name here.
  return entityName;
},
```

Posts/Index → posts/index

```
fileMapTokens: function(options) {
  // Return custom tokens to be replaced in your files
  return {
    __token__: function(options) {
      // logic to determine value goes here
      return 'value';
    }
  }
}
```

--root__ = "app"

--path__ = "posts"

--name__ = "index"

Placeholder-img blueprint

- ▶ Build a blueprint that makes a helper, for a placeholder image url
- ▶ Many of these services accept urls like <base>/w/h

<http://placekitten.com/400/420>

<http://placehold.it/400/400>

```

```



Modular Architecture

Ember addons

- ▶ A uniquely awesome benefit of using ember
- ▶ NPM modules with extras
- ▶ Can either extend ember, extend ember-cli or both!
- ▶ Similar conventions to ember apps
- ▶ emberaddons.com
- ▶ in-repo addons, in your lib folder!

```
ember g in-repo-addon <addon name>
```



A bundle of
client-side modules

Two Module Namespaces: App & Addon

- ▶ JS Modules in the addon folder will be in the addon's namespace
`ember-dropdown/utils/calculate-width.js`
- ▶ Certain types of modules need to be in the app namespace
 - ▶ Components
 - ▶ Adapters
 - ▶ Helpers
 - ▶ Serializers
 - ▶ Models
 - ▶ Templates

Components via addon

- ▶ create a new in-repo addon called core-ui
- ▶ move the x-input and x-textarea components and templates into the appropriate folders of the addon
- ▶ keep the interesting stuff in the addon folder, and re-export in to the app tree.

```
import Ember from 'ember';
import layout from 'my-addon/templates/components/x-foo';

export default Ember.Component.extend({
  layout
});
```



Modify asset pipeline

Broccoli Plugins for ember apps

- ▶ Most of the work is done in the addon's index.js file
- ▶ included hook lets us register broccoli plugins for filetypes

```
module.exports = {
  name: 'addon-name',
  included: function(app, parentAddon) {
    var app = (parentAddon || app);
    app.registry.add('hbs', new MyPlugin());
  }
};
```

Pipeline modification

18

- ▶ Move your code commenting broccoli plugin into an in-repo addon
- ▶ Host app should still end up with code comments at the top of JS files



Add a command to ember-cli

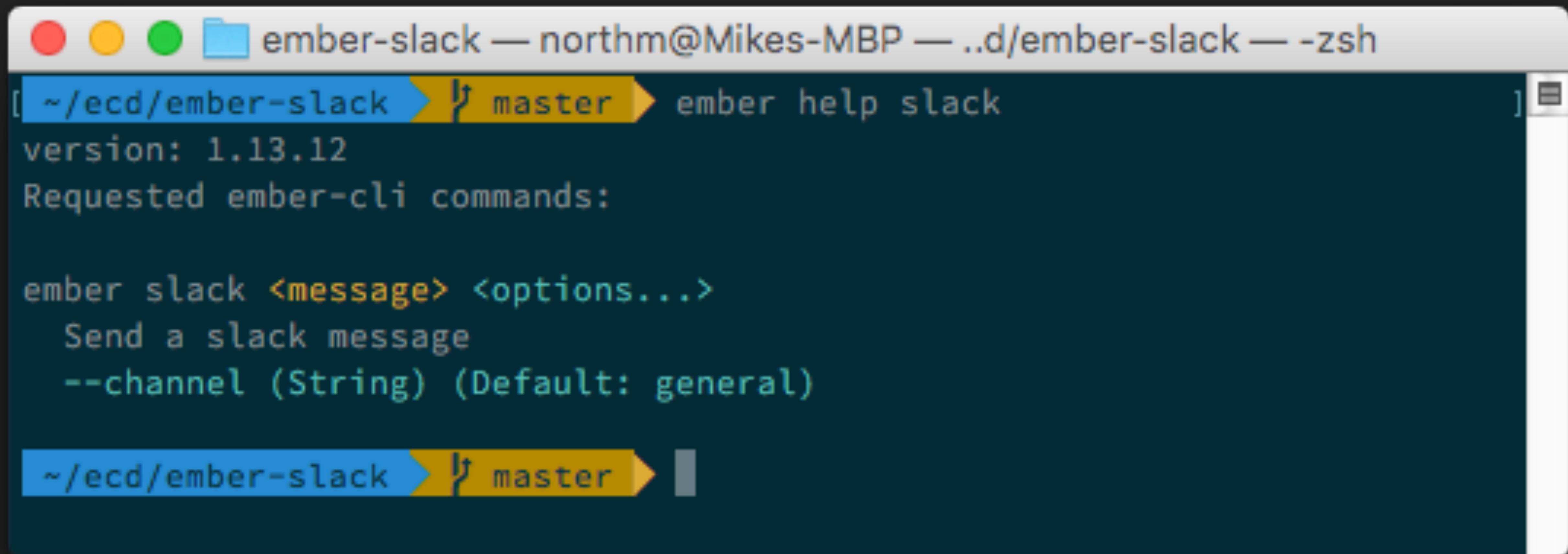
```
module.exports = {
  name: 'my-addon',
  includedCommands() {
    return {
      fem: {
        name: 'fem',
        description: 'Prints something to the screen',
        works: 'insideProject',
        anonymousOptions: [],
        availableOptions: [],
        run(commandOptions, rawArgs) {
          console.log('Welcome to Front End Masters')
        }
      }
    }
  }
};
```

named and anonymous options

```
// Un-named options
anonymousOptions: [
  '<message>'
],  

// Named options
availableOptions: [
  {
    name: 'channel',
    type: String,
    default:
      'general'
  }
],
```

Named and Anonymous Options



The screenshot shows a terminal window with the following details:

- Top bar: Shows the terminal icon, the current directory (~/ecd/ember-slack), the user (northm@Mikes-MBP), the path (..d/ember-slack), and the shell (-zsh).
- Command: The user has run the command `ember help slack`.
- Output:
 - Version information: `version: 1.13.12`
 - Requested commands: `Requested ember-cli commands:`
 - Help for `ember slack`:
 - Description: `Send a slack message`
 - Option: `--channel (String) (Default: general)`
- Bottom bar: Shows the current directory (~/ecd/ember-slack), the branch (master), and a cursor icon.

Slack Notifications

- ▶ Let's build the first part of a slack notification ember-cli-command. I want to be able to type

ember slack “🚀 Deployed” --channel announcements

- ▶ and we should see the following printed to the console

#Announcements: 🚀 Deployed