Single Page Apps

With: Knockout , Require and Durandal

Ynon Perek

ynon@tocode.co.il

Agenda

- Why Single Page Apps
- Require.JS
- Knockout.JS
- Durandal.JS

Web Applications

≠

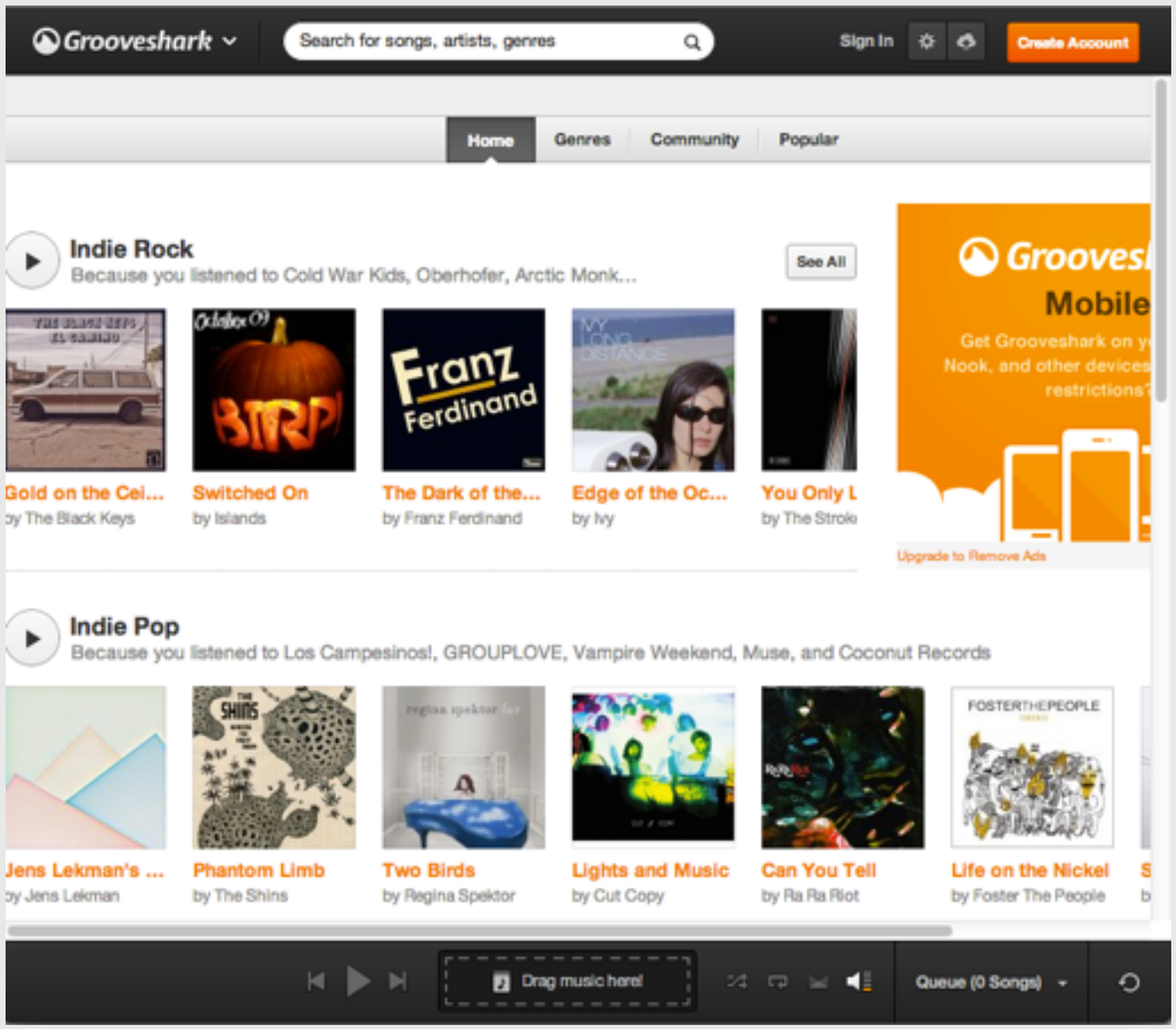
Web Sites

Web Applications

- Applications maintain state

- Applications are more device-specific (fixed sizes and images)

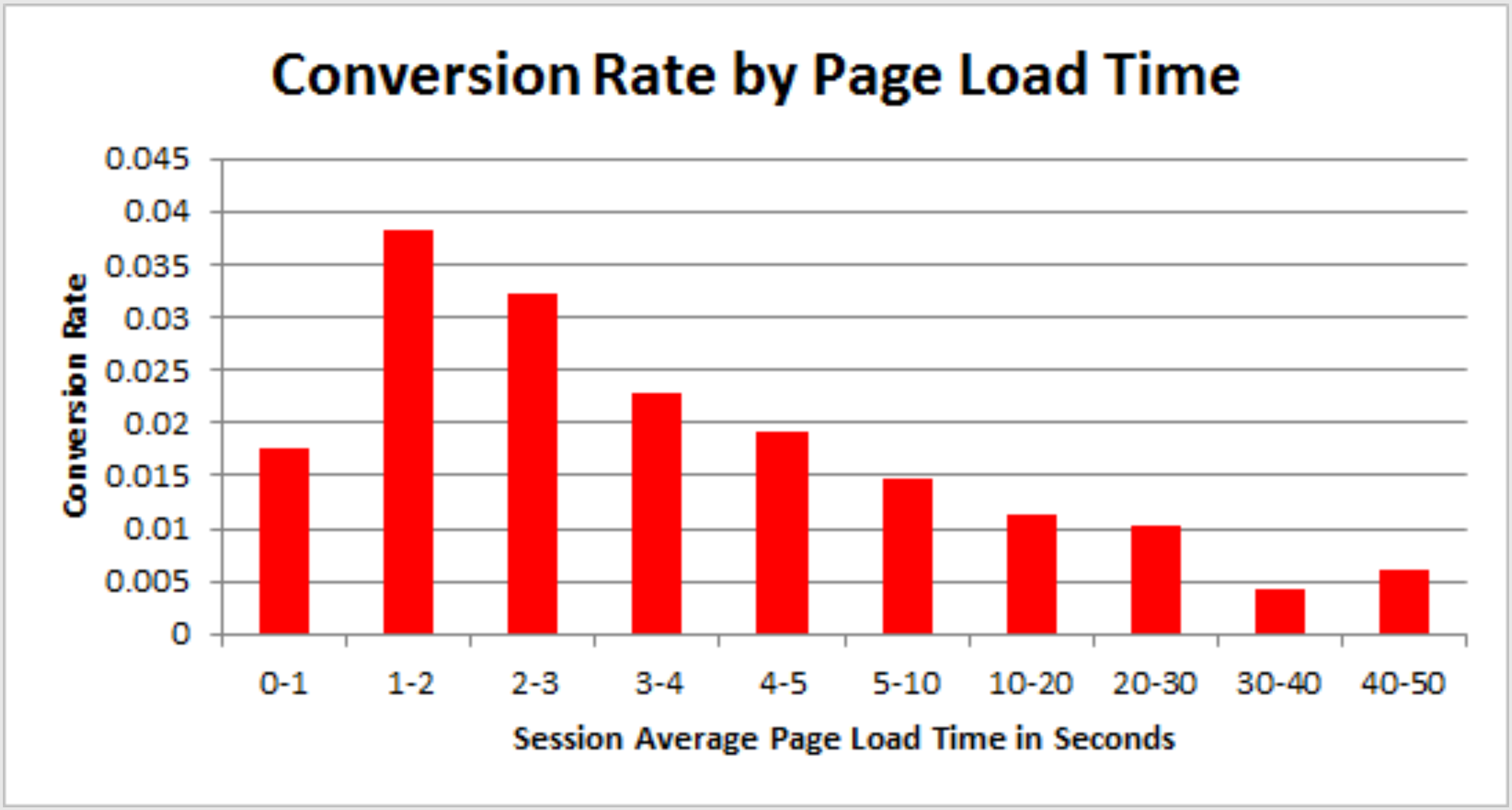
- Applications focus on UX



Why Would You Want One

> “Just One Second Delay In Page-Load Can Cause 7% Loss In Customer Conversions”

Glasses Direct Measured



Google Measured

Type Of Delay	Delay (ms)	Duration (weeks)	Impact
Pre Header	50	4	-
Pre Header	100	4	-0.2%
Post Header	200	6	-0.59%
Post Header	400	6	-0.59%

Why Would You Want One

- better performance
- animated transitions
- “feels” better to a user

We need a *Different* kind of JavaScript

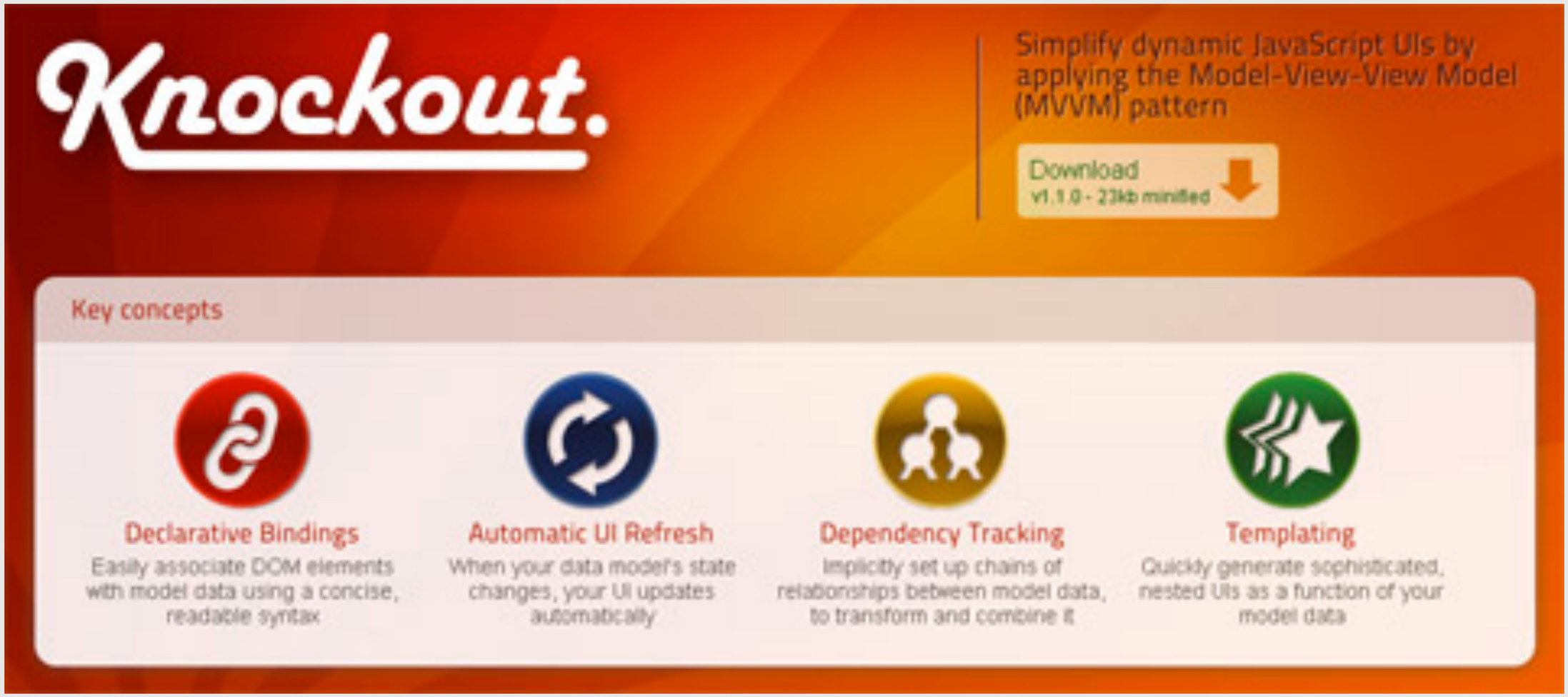
How Is This Different

- Larger code base
- No memory leaks
- Faster and more interactive pages
- Modular And Reusable Code
- data is kept in JS memory

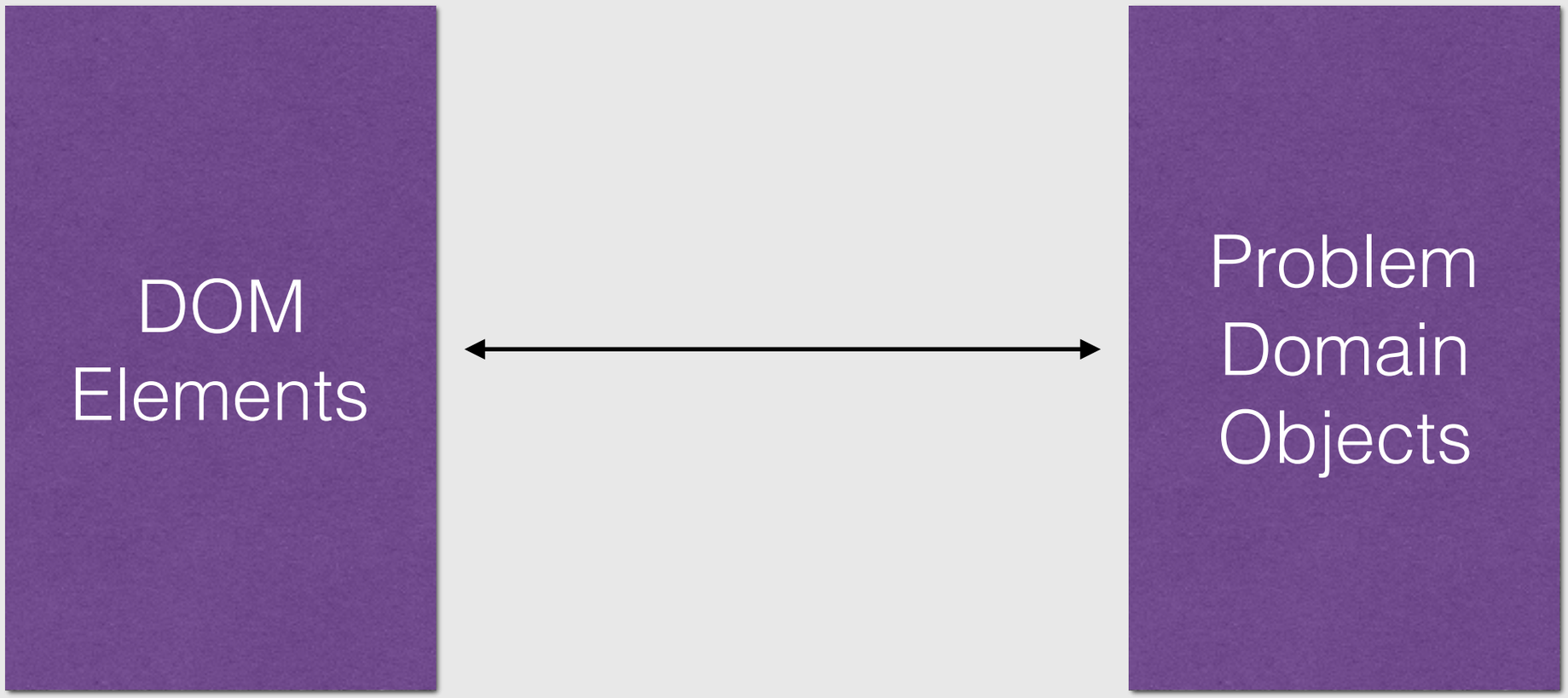
Durandal Ecosystem

- At first there was jQuery
- Then came SPA frameworks: Durandal, Angular, Backbone
- Then came modern ES6 frameworks: Aurelia, Angular2, React

Knockout Data Binding



What's A Data Binding



One way data bind

```
var person = {
  name:  'Batu',
  likes: 'cooking',
  image: 'profile.png'
};
```

```
<div>
  <span class="name">Batu</span>
  <img src="profile.png" />
</div>
```

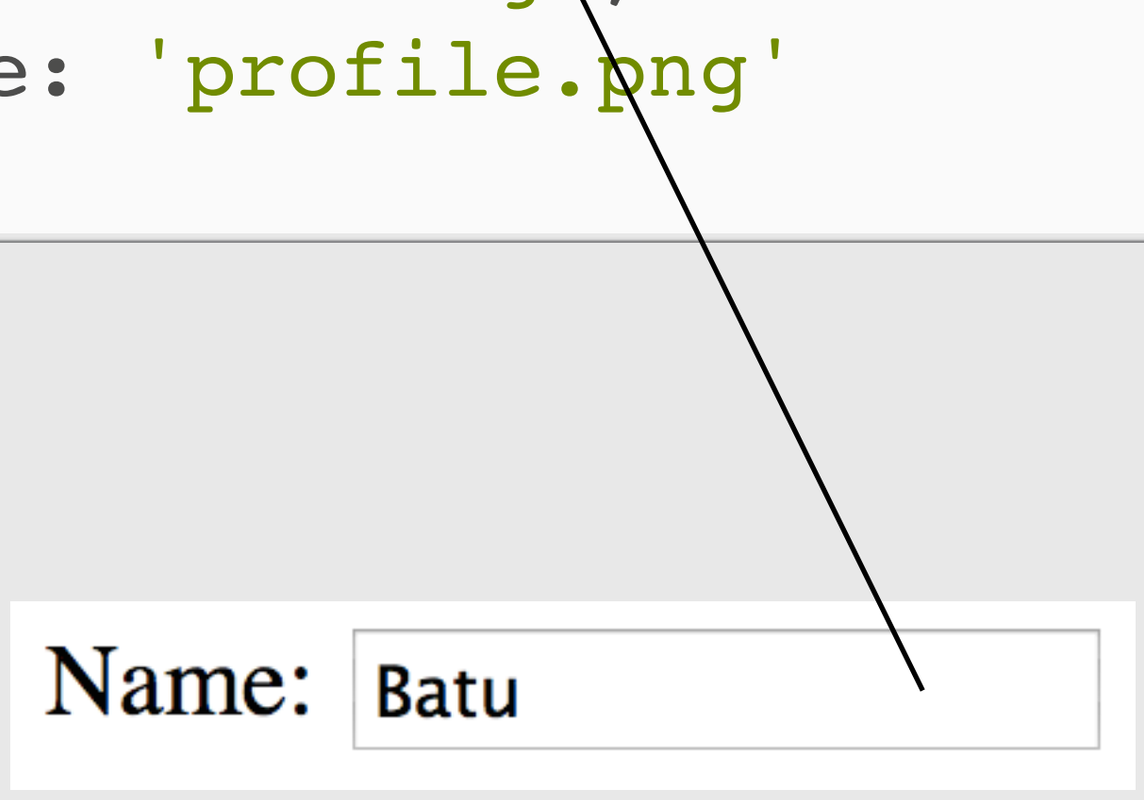
For Example…

```
var person = {
   name: 'Batu',
   likes: 'cooking',
   image: 'profile.png'
};
```

Name: Batu

Ok, so what's a two way data binding?

```
var person = {
   name:   'Batu',
   likes: 'cooking',
   image: 'profile.png'
};
```



Our Own Binder

```
var person = {
   name: 'Batu',
   likes: 'cooking'
};

$('input').on('change', function() {
    person.name = this.value;
    console.log('name is now: ' + person.name);
});
```

What Can Go Wrong

- Hard to track many event handlers
- Need to change DOM every time value is changed
- Bad display/logic separation

Data Binding Framework

- Write properties to bind in the HTML
- Write application logic in JS
- Framework will handle connections

Hello Knockout

home.js

```
define(function (require) {

  var self = {
    name: 'Batu',
    image: 'profile.png'
  };

  return self;
});
```

home.html

```
<section>
  <label>
    Name:
    <input class="text" data-bind="value: name" />
  </label>
  <img data-bind="attr: { src: image }" />

</section>
```

Hello Knockout

home.js

```
define(function (require) {

  var self = {
    name: 'Batu',
    image: 'profile.png'
  };

  return self;
});
```

home.html

```
<section>
  <label>
    Name:
    <input class="text" data-bind="value: name" />
  </label>
  <img data-bind="attr: { src: image }" />

</section>
```

Two Way Bindings

home.js

```
define(function (require) {
  var ko = require('knockout');

  var self = {
    name: ko.observable('Batu'),
    image: 'profile.png'
  };

  return self;
});
```

home.html

```
<section>
  <label>
    Name:
    <input class="text" data-bind="value: name" />
  </label>
  <img data-bind="attr: { src: image }" />

</section>
```

Demo: Click Counter

- Let’s build a simple click counter
- Contains:
  - A button
  - A text input showing total button clicks

Demo Takeaways

- Only observable attributes update the DOM
- Observable attributes are functions -> assignment is performed by calling
- `self.clicks( self.clicks() + 1 );`

Lab

- Display current date and time in 3 input fields:
    - First displays only date
    - Second display only time
    - Third displays both date and time
- Update display every second

Q & A

Knockout Bindings

Knockout Bindings

text	foreach	click, checked
html	if, ifnot	event
css, style	with	submit
attr		enable, disable
visible		value

Text

- Determines an element text
- Automatically escapes HTML tags

```
Today's message is: <span data-bind="text: myMessage"></span>

<script type="text/javascript">
    var viewModel = {
        myMessage: ko.observable() // Initially blank
    };
    viewModel.myMessage("Hello, world!"); // Text appears
</script>
```

html

- Just like text, but without escaping HTML chars

- Be careful sanitising your inputs

attr

- Sets values on attributes

```
<a data-bind="attr: { href: url, title: details }">
    Report
</a>

<script type="text/javascript">
    var viewModel = {
        url: ko.observable("year-end.html"),
        details: ko.observable("Report including final year-end
statistics")
    };
</script>
```

css

- Add / Remove css classes

```
<div data-bind="css: { profitWarning: currentProfit() < 0 }">
    Profit Information
</div>

<script type="text/javascript">
    var viewModel = {
        currentProfit: ko.observable(150000)
    };
    viewModel.currentProfit(-50);
</script>
```

foreach

- Iterates over an array
- creates inner scope - inside handlers “this” is the item

```
<h4>People</h4>
<ul data-bind="foreach: people">
    <li>
        Name at position <span data-bind="text: $index"> </span>:
        <span data-bind="text: name"> </span>
        <a href="#" data-bind="click: $parent.removePerson">Remove</a>
    </li>
</ul>
<button data-bind="click: addPerson">Add</button>
```

Demo

- Let’s write a simple TODO list
- Input field for adding new tasks
- Unordered list for showing the tasks

Conclusions

- KO event binding automatically binds your functions and passes params:

  (data, event) => handleEvent

Conclusions

- Can rebind KO handlers to pass a different set of parameters:

```
<button data-bind="click:
$parent.removeItem.bind(null, $index())">Remove</
button>
```

Conclusions

- Careful with “this”

if

- Add / Remove node contents based on a condition
- Needs a container

```
<label>
  <input type="checkbox" data-bind="checked: displayMessage" />
  Display message
</label>

<div data-bind="if: displayMessage">
  Here is a message. Astonishing.
</div>
```

Virtual Elements

- Sometimes a container is too much to ask for (consider <li>)
- Knockout provides virtual elements

```
<!-- ko if: displayMessage -->
  Here is a message. Astonishing.
<!-- /ko -->
```

value

- Binds an observable to an input value

```
<p>Login name: <input data-bind="value: userName" /></p>
<p>Password: <input type="password" data-bind="value: userPassword" /></p>

<script type="text/javascript">
    var viewModel = {
        userName: ko.observable(""),
        userPassword: ko.observable("abc"),
    };
</script>
```

value

- pass `valueUpdate` to control when value is updated
- Possible values:
  - input, keyup, keypress, afterkeydown

Binding Warning

- Keep in mind: You don’t always need a binding

- Bindings create event handlers,
  too many of them will slow down your page

Q & A

Computed Properties

Red

RGB(255,0,0)

HSB(9,100%,100%)

What's a Computed Property

- Computed properties display the same value as a real property, but in a different way

- The value is derived from other properties

Simply Computed

```
function AppViewModel() {
    // ... leave firstName and lastName unchanged ...

    this.fullName = ko.computed(function() {
        return this.firstName() + " " + this.lastName();
    }, this);
}
```

Writable Computed Attributes

```
1 var tags = ko.computed({
2     read: function() {
3
4     },
5     write: function(value) {
6
7     }
8 });
```

Demo #1

- Write a screen with a text field and a list of tags
  - Text field should get a string of tags separated by commas
  - Tag list should show one entry per tag

Demo #2

- Let's allow deleting a tag by clicking it
- One approach:
  - split tag line
  - remove tag
  - join back tag list
- A better solution: save tags as an array

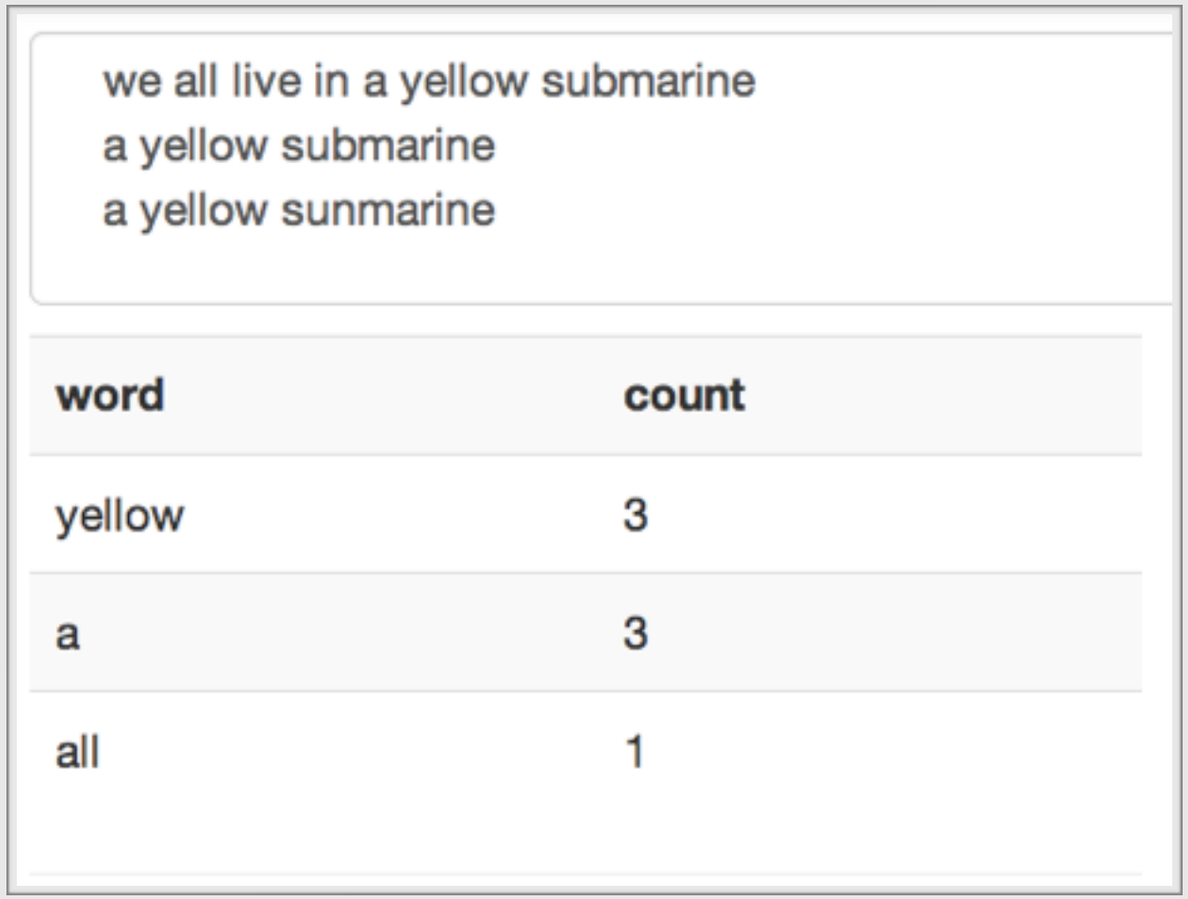
Lab 1

- Write a currency converter that converts between 3 currencies: NIS, Dollar and Euro

- App has 3 inputs (one for each currency), changing one should affect all

Lab 2

- Implement a text area and a word frequency table
- Table has 2 columns: word, count
- After a user types in some text, table should be updated with the correct word counts for each word
- Hint: Use class="table" on the table for styling



Computable Properties Takeaways

- Careful when using “this”

- But if you do decide to use it, remember to bind its value to the computed property

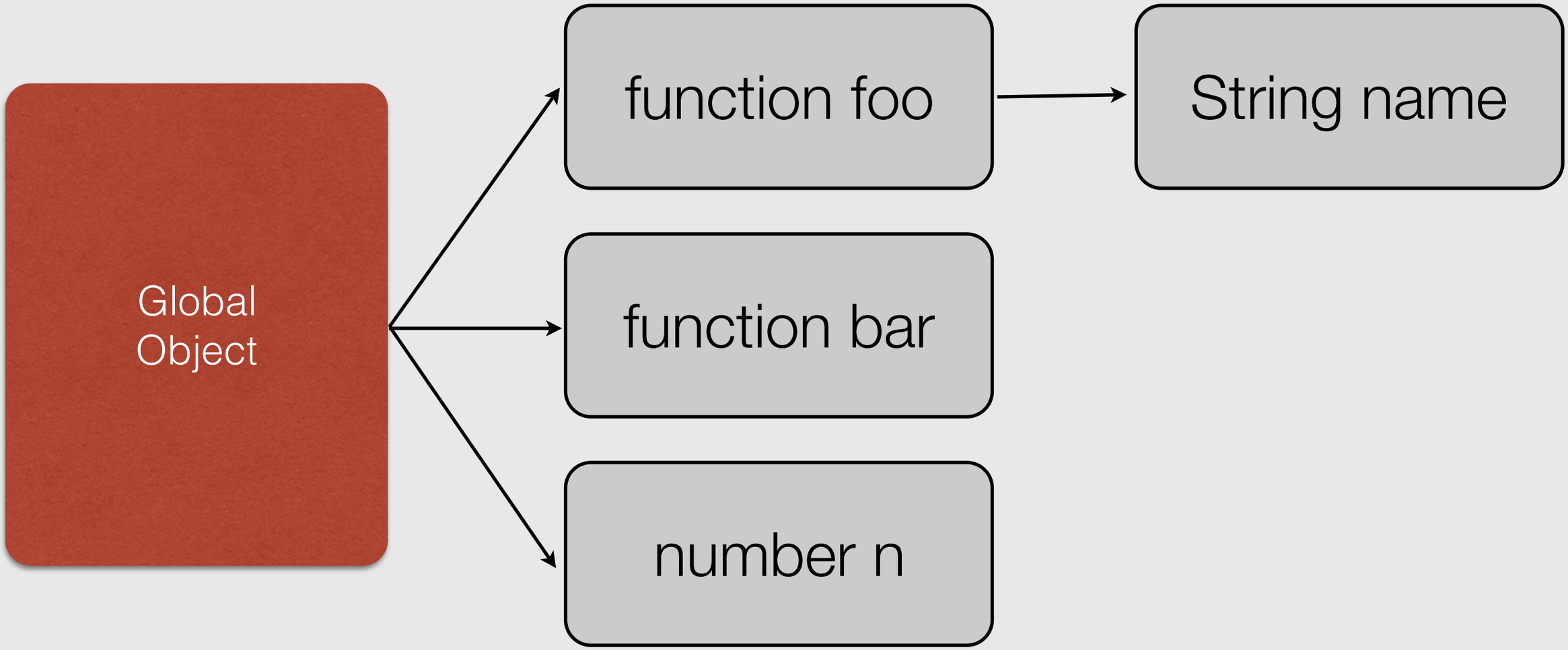
Computable Properties Takeaways

- Dispose computed property when it's no longer needed with `prop.dispose()`

Q & A

Require.JS

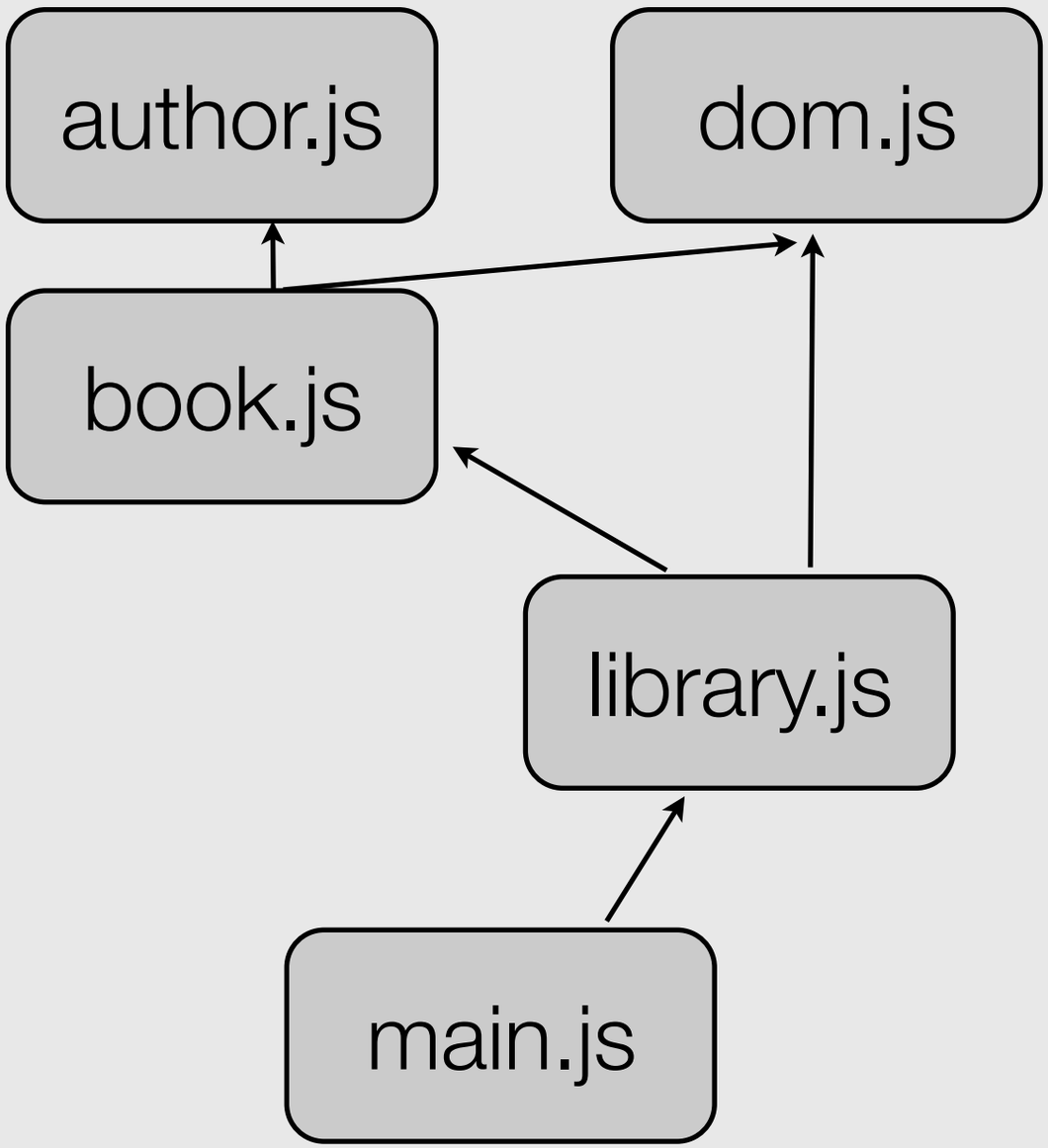
JavaScript Bad Call



JavaScript Bad Call

- By default, all variables are global
- No easy way to load dependencies
- In practice - developers use too many globals

How We Use Dependencies




```
<script src="dom.js"></script>
<script src="author.js"></script>
<script src="book.js"></script>
<script src="library.js"></script>
<script src="main.js"></script>
```

Disadvantages Of <script>

- Scripts != modules
- No Hierarchy in scripts
- No optional scripts
- No lazy loading
- No “private” scripts

What We Want

- Treat our code as “modules”
- Specify dependencies within code
- Lexical names
- => Require.JS

Classic “Class”

```
1 function Book(title, author) {
2   var self = this;
3
4   self.title = title;
5   self.author = author;
6
7   self.year = new Date().getUTCFullYear();
8 }
```

Hello Require.JS

Module book: No new globals

```
1 // book.js
2 define(function() {
3   return function(title, author) {
4     this.title  = title;
5     this.author = author;
6     this.year   = new Date().getUTCFullYear();
7   };
8 });
```

```
var Book = function(title, author) {
  var self = this;

  self.title = title;
  self.author = author;

  self.year = new Date().getUTCFullYear();
}
```

```
// book.js
define(function() {
  return function(title, author) {
    this.title  = title;
    this.author = author;
    this.year   = new Date().getUTCFullYear();
  };
});
```

Using The Book

Importing the Book module

```
1 // main.js
2 define(['book'], function(Book) {
3     var foo = new Book('JS Rocks', 'Jim');
4     console.dir( foo );
5 });
```

Using The Book

and lexically naming it

```
1 // main.js
2 define(['book'], function(Book) {
3   var foo = new Book('JS Rocks', 'Jim');
4   console.dir( foo );
5 });
```

Require Modules

- `define(...)` creates a new module. Must return a value

- When client requests a module, it gets the return value of define

- Module can be anything: function, object or even a simple value

Require Starting Point

- Now we just need one <script> tag: require

```
<!DOCTYPE html>
<html>
<head>
   <title></title>
</head>
<body>
    <script data-main="scripts/main"  src="http://
cdnjs.cloudflare.com/ajax/libs/require.js/2.1.1/
require.min.js">
</script>
</body>
</html>
```

Require Modules

- Return an object to define a singleton

```
1 define(function() {
2    return {
3       url: '/api/contacts',
4       list: function() {
5       },
6       save: function() {
7       }
8    };
9 });
```

Require Modules

- Return a function to define a class

```
1 define(function() {
2    return function(name) {
3       var self = this;
4       self.hello = function() {
5       };
6    };
7 });
```

Alternative Syntax

```
1 define(function(require) {
2     var Book = require('book');
3     var Author = require('author');
4
5 });
```

Require Configuration

```
<script>
    requirejs.config({
        option: value
    });

</script>
```

Useful Config Options

- paths defines search paths for modules
  - jquery -> ../lib/jquery/jquery-1.9.1.js
  - vm/home -> view_models/home.js

```
1 requirejs.config({
2   paths: {
3     'jquery': '../lib/jquery/jquery-1.9.1',
4     'vm': 'view_models'
5   }
6 });
```

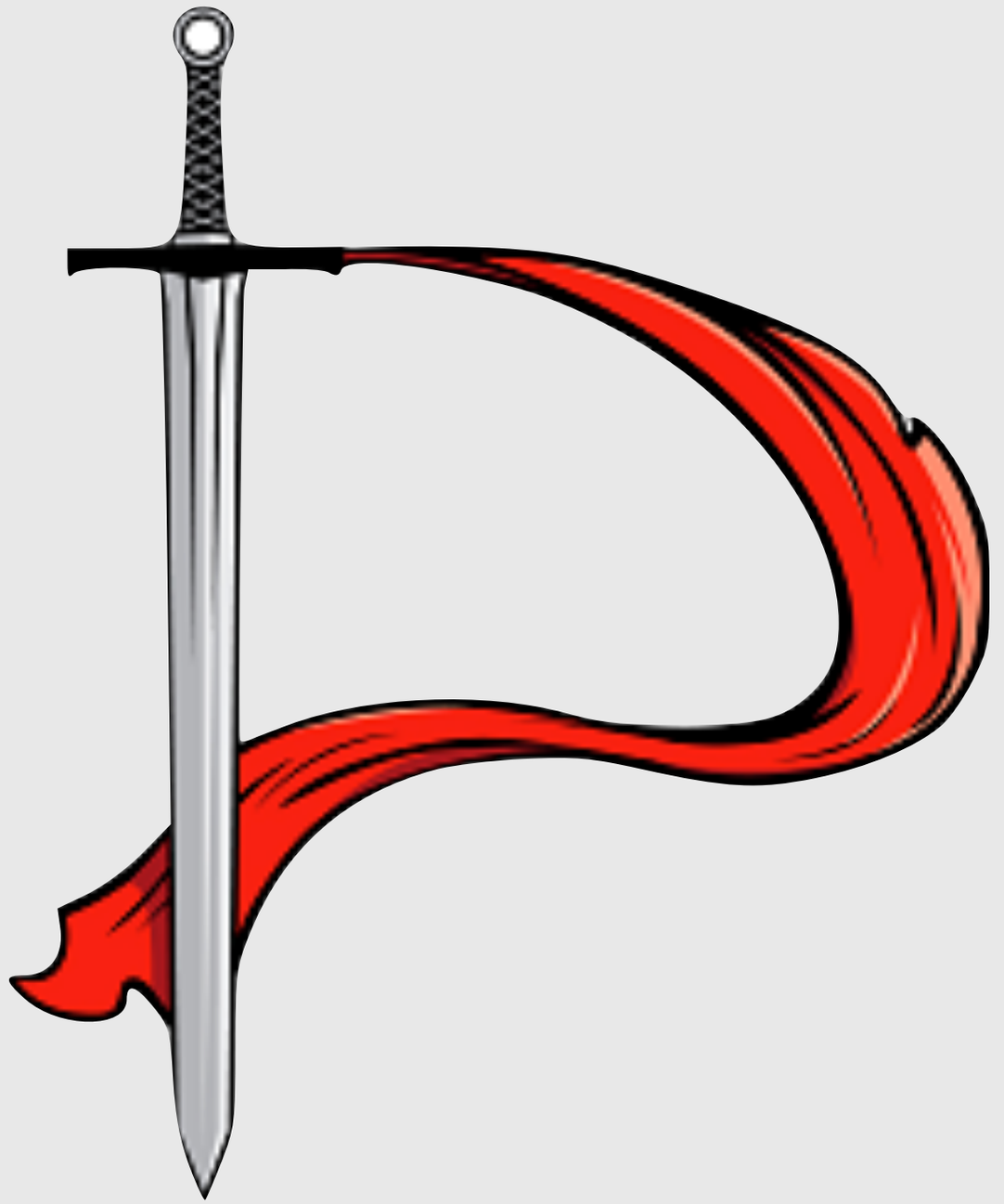
Require: Keep In Mind

- Every file is a module

- Root module is specified when loading require (in html)

Q & A

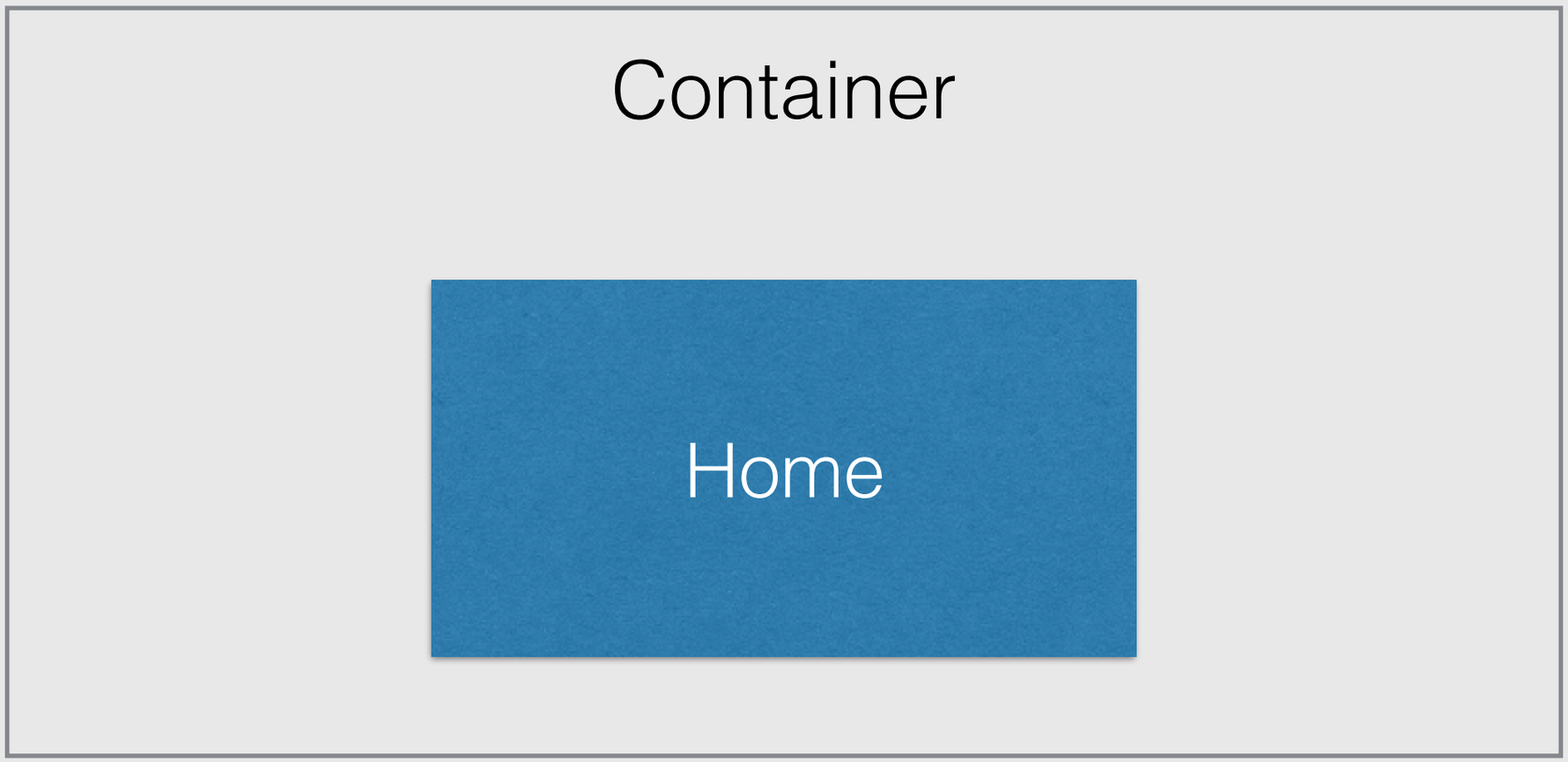
Going Multi Page with Durandal's Router



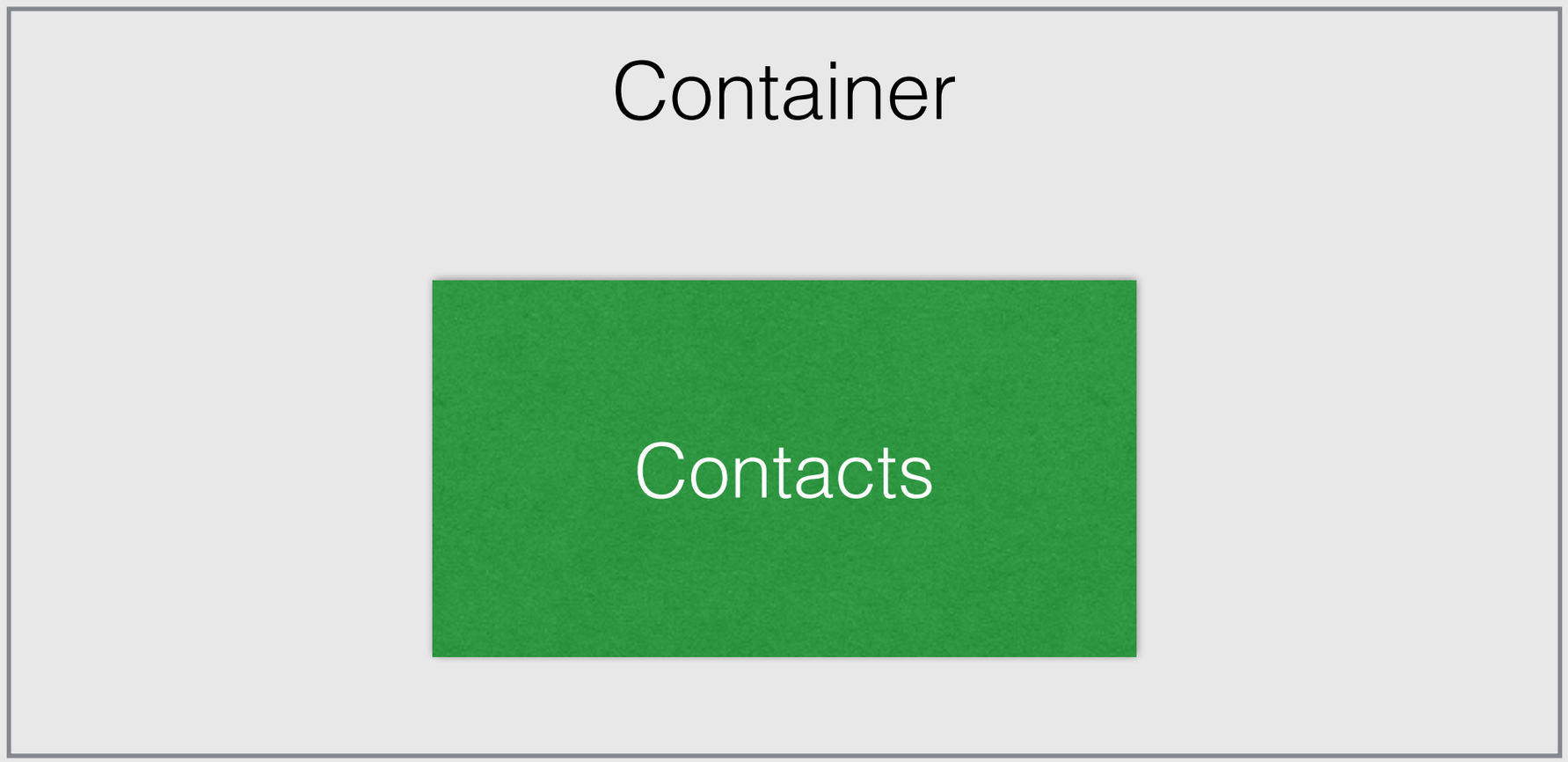
Multi Page Single App

- One container, inner html changes
- Each page is a div injected to the container

Multi Page Single App



Multi Page Single App



Swapping Pages

- Durandal provides a router class that does most of the swapping work for us
- A Router:
  - Defines routes - a mapping from url pattern to a view model
  - Swap container’s inner html when route changes

Hello Router

```
define(function (require) {
  var router = require('plugins/router');

  return {
    router: router,
    activate: function () {
      router.map([

      { route: '',        title:'Home',   moduleId: 'vm/home',         nav: true },
      { route: 'clicks', title:'Clicks', moduleId: 'vm/clickcounter', nav: true },

      ]).buildNavigationModel();

      return router.activate();
    }
  };
});
```

Router Provides

- `router.map()` defines routes
- `router.activate()` starts to listen on hash change events, and will swap views
- `router.buildNavigationModel()` creates an array of route items, that are used to build a menu bar

View Model Life Cycle

- A router notifies its view models when switching views, giving them a chance to handle the event

- This is done by calling agreed methods on the view model, if they exist

View Life Cycle

Method Name	Purpose
canDeactivate(), canActivate()	Allow vm to cancel activation/ deactivation
activate()	Allow running custom activation logic
deactivate()	Allow previous view to run custom deactivation logic
attached()	Notifies new object its view is attached
detached()	Notifies a view model its view was removed from the DOM

Using Activation Methods

- `activate()` is used to load data from server

- It's so common - that if you return a promise, routing won't be performed until it is completed

- Remember: `$.ajax()` is a promise

Demo

- Creating two pages: Home + About

- Each with its own view and view model

Demo

- omdbapi provides information about movies and TV shows

- Write a new page that shows a list of matches for “Fargo” by:

  - GET http://omdbapi.com/?s=fargo

  - Show results in page

- Use “activate” to fetch data before changing pages

Router Takeaways

- Define a new route in `router.map()`
- Implement view model's `activate()` for fetching remote data
- Use `router.isNavigating()` to show/hide a loading spinner

Q & A

Routing Parameters

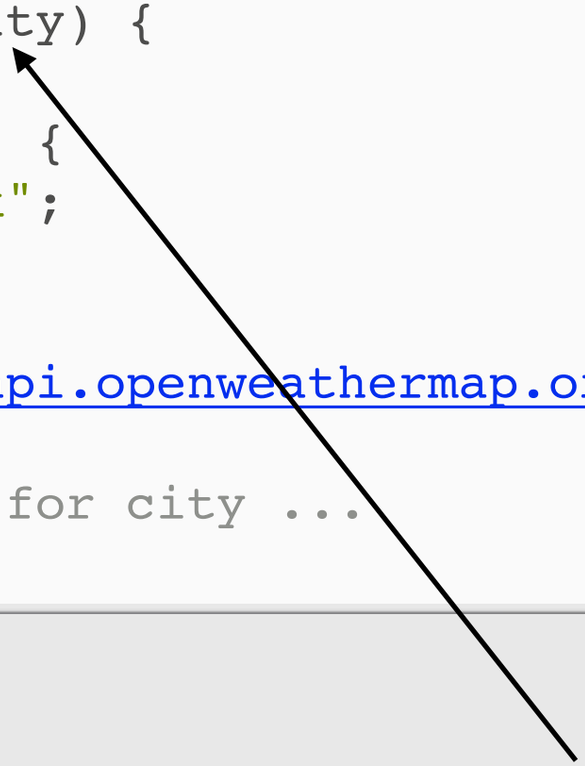
- This time let's assume our app wants to display weather in one of several large cities
- We don't want to write a view model and view for each city
- Meet parameterised routes

Routing Parameters

```
1  activate: function(city) {
2
3    if ( city == null ) {
4      city = "London,uk";
5    }
6
7    var url = 'http://api.openweathermap.org/data/2.5/weather?q=' + city;
8
9    // now get weather for city ...
10 }
```

Routing Parameters

```
1  activate: function(city) {
2
3    if ( city == null ) {
4      city = "London,uk";
5    }
6
7    var url = 'http://api.openweathermap.org/data/2.5/weather?q=' + city;
8
9    // now get weather for city ...
10 }
```



This is called a routing parameter

Parameterised Route

```
router.map([
  { route: '', title:'Home', moduleId: 'vm/home'},
  { route: 'weather/:city', title:'Weather', moduleId: 'vm/weather'},
]).buildNavigationModel();
```

Route assigns value to the parameter

Parameterised Route

weather/London,uk ⟶ Calls activate with: London,uk

```
router.map([
  { route: '', title:'Home', moduleId: 'vm/home'},
  { route: 'weather/:city', title:'Weather', moduleId: 'vm/weather'},
]).buildNavigationModel();
```

Route assigns value to the parameter

Parameterised Route

- Now we have two options to proceed:

- Navigate from code using `router.navigate(...)`

- Navigate using `<a>`

Demo

- Let’s modify our app to take a search keyword from the user

Lab

- Site haveibeenpwned provides an API to show data breaches:
  https://haveibeenpwned.com/api/v2/breachedaccount/{account}

- Add a page that checks if a user's email has recently been pwned

Durandal Widgets

Multiple Weather Selectors

- Let's try another modification:
  - Weather page should display 3 weather selectors
  - Each selector has a city select box and weather description for that city

Multiple Weather Selectors

- Solution 1 — Use nested routes
- #/weather/:city1/:city2/:city3
  - +) internal bookmarking
  - -) doesn’t scale well

Multiple Weather Selectors

- Alternative: Widgets
- A widget is a reusable component that can be embedded in pages
- Includes: View + View Model

Hello Widget

- Add view and view model in:
  - app/widgets/widget-name
    - view.html
    - viewmodel.js

Hello Widget

- Add your widget type to the widget plugin configuration (in main.js)

```
1 app.configurePlugins({
2     router:true,
4     widget: {
5         kinds: ['weather']
6     }
7 });
```

Hello Widget

- Use the widget by data binding to it from an existing page

```
<section>
 <div data-bind="weather: { name: 'bob'}"></div>
</section>
```

Hello Widget

- Use the widget by data binding to it from an existing page

```
<section>
 <div data-bind="weather: { name: 'bob'}"></div>
</section>
```

Can pass arguments to the widget

Using Widget Params

```
1 // widget/viewmodel.js
2
3 self.activate = function(settings) {
4   self.name = settings.name;
5 };
6
```

Widget View Model

- View models for a page return an object - all you need is one model per page

- View models for a widget should return a class (ctor function). Each instance needs its own data.

Demo: Name Widget

- Let's write a simple text widget

- Takes "name" as argument, and prints "Hello, name"

Q & A

Lab

- Implement a widget for a “Catch Red” game
- Show 3 game boards on screen
- All 3 should share the same “score”

Widgets Without The View

- For some widgets, it may be preferable to create the view from JS
- This is usually the case when integrating existing KendoUI or jQuery Plugins
- For this we'll use custom KO bindings

Custom KO Bindings

anywhere in the app, preferable file under bindings/ folder

```
ko.bindingHandlers.yourBindingName = {
    init: function(element, valueAccessor, allBindings, viewModel, bindingContext) {
        // This will be called when the binding is first applied to an element
        // Set up any initial state, event handlers, etc. here
    },
    update: function(element, valueAccessor, allBindings, viewModel, bindingContext) {
        // This will be called once when the binding is first applied to an element,
        // and again whenever the associated observable changes value.
        // Update the DOM element based on the supplied values here.
    }
};
```

Custom KO Binding

- `init()` is called once when binding is created
- `update()` is called every time an observable property changes
- Use `init()` to create DOM event handlers

Demo

- Let's integrate jQuery UI accordion as a custom binding

- Need to:
  - Load jQuery UI
  - Create a custom binding
  - Delete custom binding when done using: `ko.utils.domNodeDisposal.addDisposeCallback(…)`

Q & A

Lab

- Goal Progress is a jQuery plugin that shows an animated goal progress bar

- Write a custom binding to turn a into a goal progress

Thanks For Listening

- Ynon Perek
- ynon@ynonperek.com
- http://ynonperek.com

Photos From

- (On a wire) https://flic.kr/p/54TGfM
- (Ski Festival in Turkey) https://flic.kr/p/7BugLa
- (Antelope in wild) https://flic.kr/p/79C88k