

# JAVASCRIPT DEVELOPMENT

Sasha Vodnik, Instructor

# HELLO!

1. Pull changes from the `svodnik/JS-SF-8-resources` repo to your computer
2. Open the `starter-code` folder in your editor

**JAVASCRIPT DEVELOPMENT**

---

# **INTRODUCTION TO REACT**

# LEARNING OBJECTIVES

At the end of this class, you will be able to

- › Understand the roles of model, view, and controller
- › Describe the difference between frameworks and libraries
- › Recognize the primary uses of React
- › Create a component hierarchy
- › Build a React component

# AGENDA

- Model View Controller (MVC)
- Frameworks and libraries
- React overview
- Creating React components
- React lab

---

## INTRODUCTION TO REACT

---

# WEEKLY OVERVIEW

**WEEK 10**

React / Final project lab

**WEEK 11**

Final project presentations

# **EXIT TICKET QUESTIONS**

1. Can you connect firebase to a domain name that you already own?

# **Final Project Checkin**



# ACTIVITY

---



## EXERCISE

### KEY OBJECTIVE

---

- Check in on final project

### TYPE OF EXERCISE

---

- Groups of 3

### TIMING

---

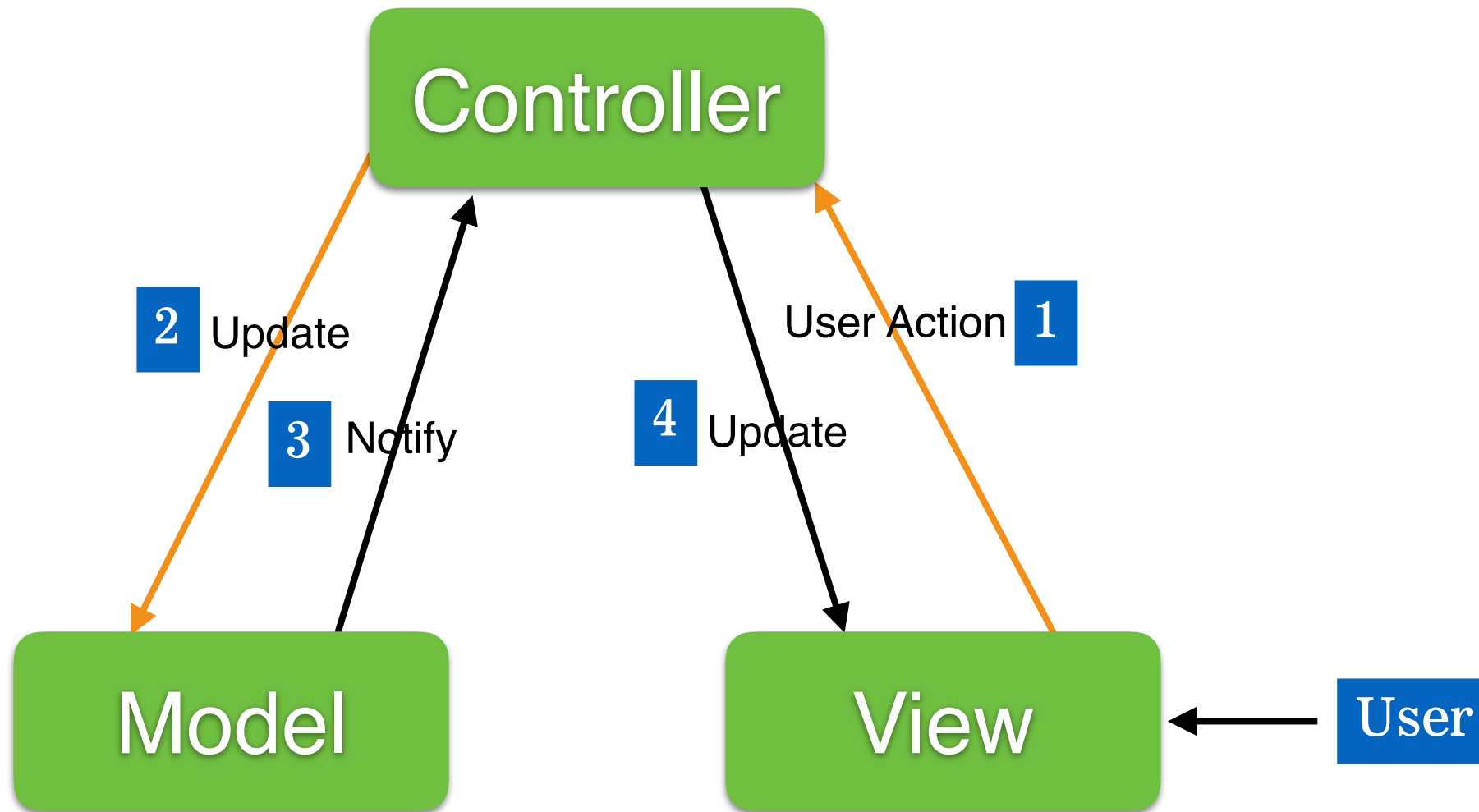
*10 min*

1. Take turns checking in about where you are with your final project. If you have a working prototype, display your app in your browser, demonstrate its functionality, and explain what you plan to add to your app.
2. Share a challenge you've run into with your project. If you've overcome it, describe how. If not, brainstorm resources and next steps with your group members.

# **MODEL-VIEW-CONTROLLER (MVC)**

- **Model:** handles data and business logic
- **View:** presents data to user in any supported format and layout
- **Controller:** receives user inputs and calls appropriate resources to carry them out

# MODEL-VIEW-CONTROLLER (MVC)



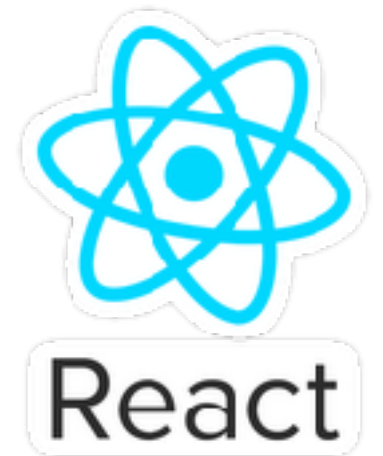
# A Library

- Set of predefined functions that your code calls
- Each call performs work and returns a result (and control) to your code
- Specific, well-defined operations
- Example: jQuery



# A Framework

- › Opinionated architecture for building software
- › Control-flow exists, you fill in with your code
- › Calls your code; is always in control
- › Examples: React, Angular, Vue, Ember





## GraphQL

Describe your data

```
type Project {  
  name: String  
  tagline: String  
  contributors: [User]  
}
```

Ask for what you want

```
{  
  project(name: "GraphQL") {  
    tagline  
  }  
}
```

Get predictable results

```
{  
  "project": {  
    "tagline": "A query language for APIs"  
  }  
}
```



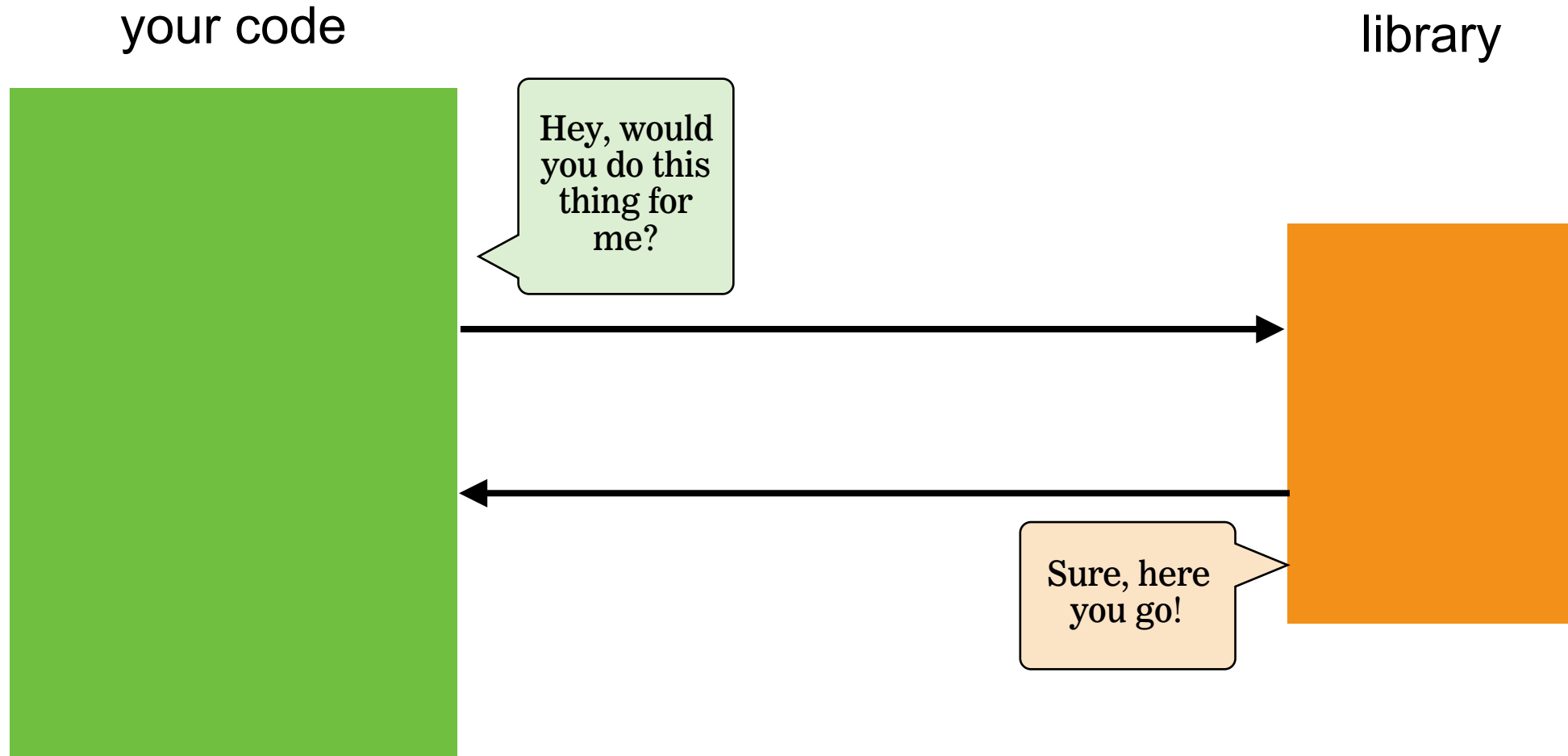
```
function init()
{
    // Initialize
    var gl = initWebGL("example");
    if (!gl) {
        return;
    }

    g.program = simpleSetup(
        gl, "vshader", "fshader",
        [ "vNormal", "vColor", "vPosition"], [ 0, 0, 0, 1 ], 10000);

    // Set some uniform variables for the shaders
    gl.uniform3f(gl.getUniformLocation(g.program, "lightDir"), 0, 0, 1);
    gl.uniform1i(gl.getUniformLocation(g.program, "sampler2d"), 0);

    // Create a box. with the BufferObjects containing the arrays
    // for vertices, normals, texture coords, and indices.
    g.box = makeBox(gl);
}
```

# YOUR CODE CALLS A LIBRARY

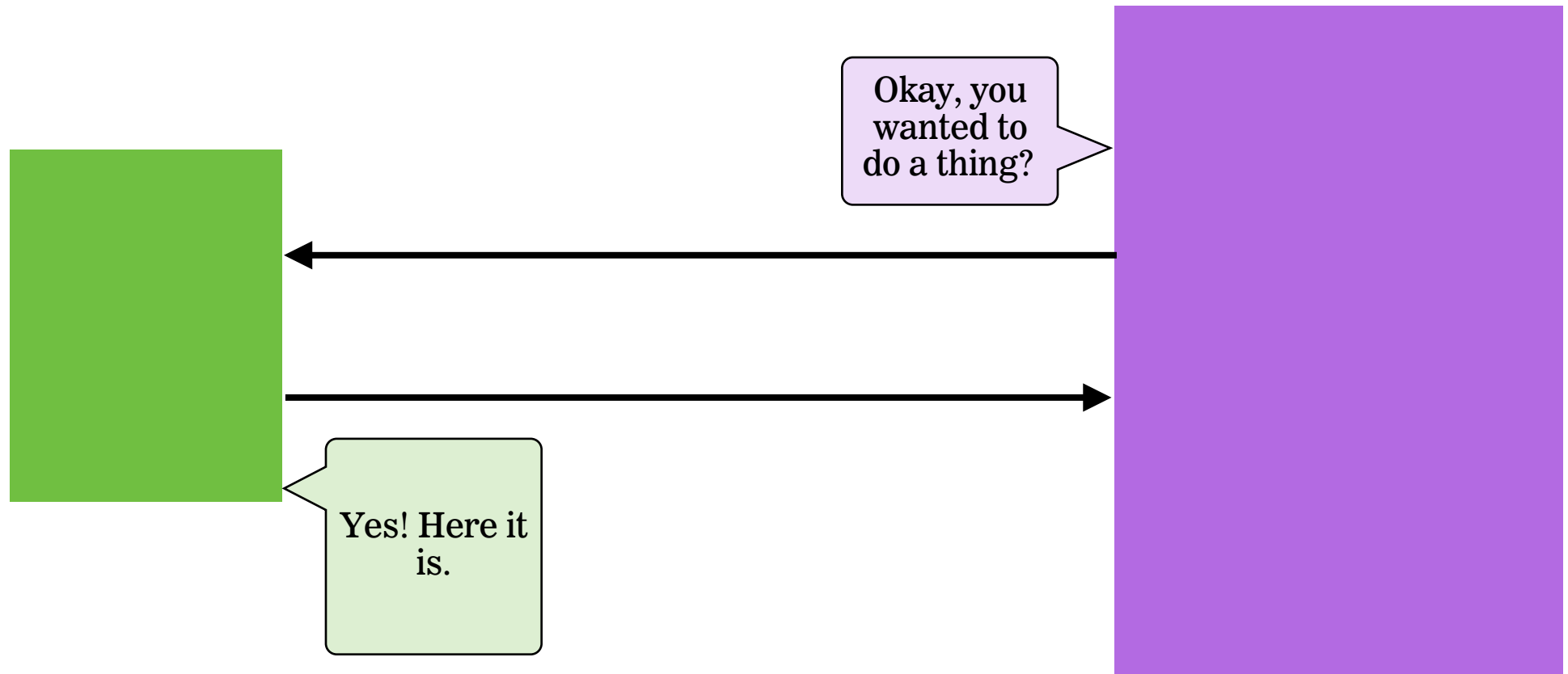




# A FRAMEWORK CALLS YOUR CODE

your code

framework



# **WHY USE FRAMEWORKS?**

- Standard / well known
  - Dictates a method that cannot be (easily) ignored
- Common problems already solved
  - Cross Browser
  - Accessibility
  - Complexity of state

# LIBRARIES

- Target a single problem
- Are usable in any project
- Often consist of a set of independent functions
- Are lightweight

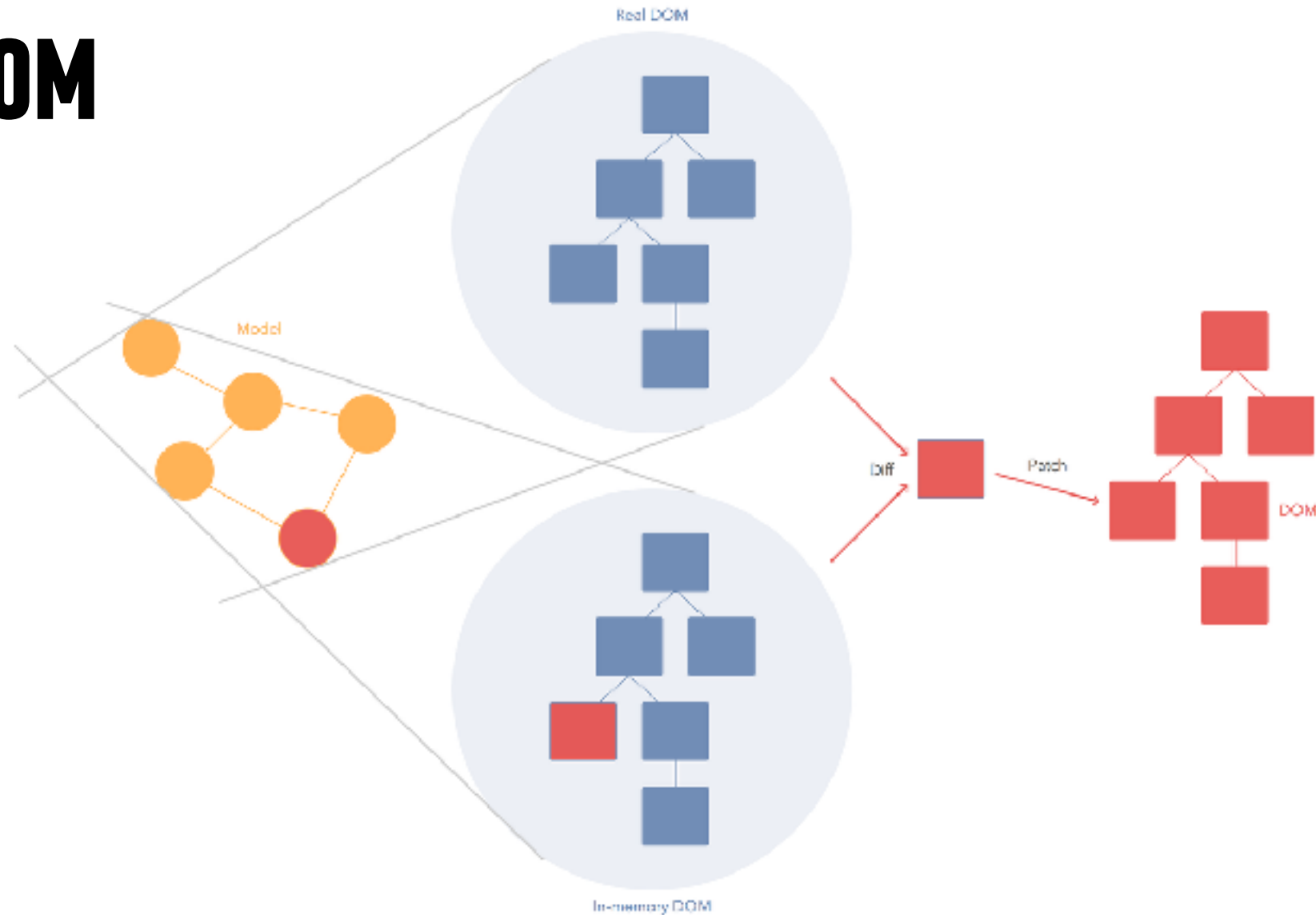
# REACT

- somewhere between a framework and a library
  - “a framework that feels like a library”
- It only cares about your views (V from MVC)
- BUT you must do your views the React way

# VIRTUAL DOM

- Tracks changes to DOM without making them immediately
- React changes DOM to match only when necessary
- This is quicker than doing direct DOM manipulation

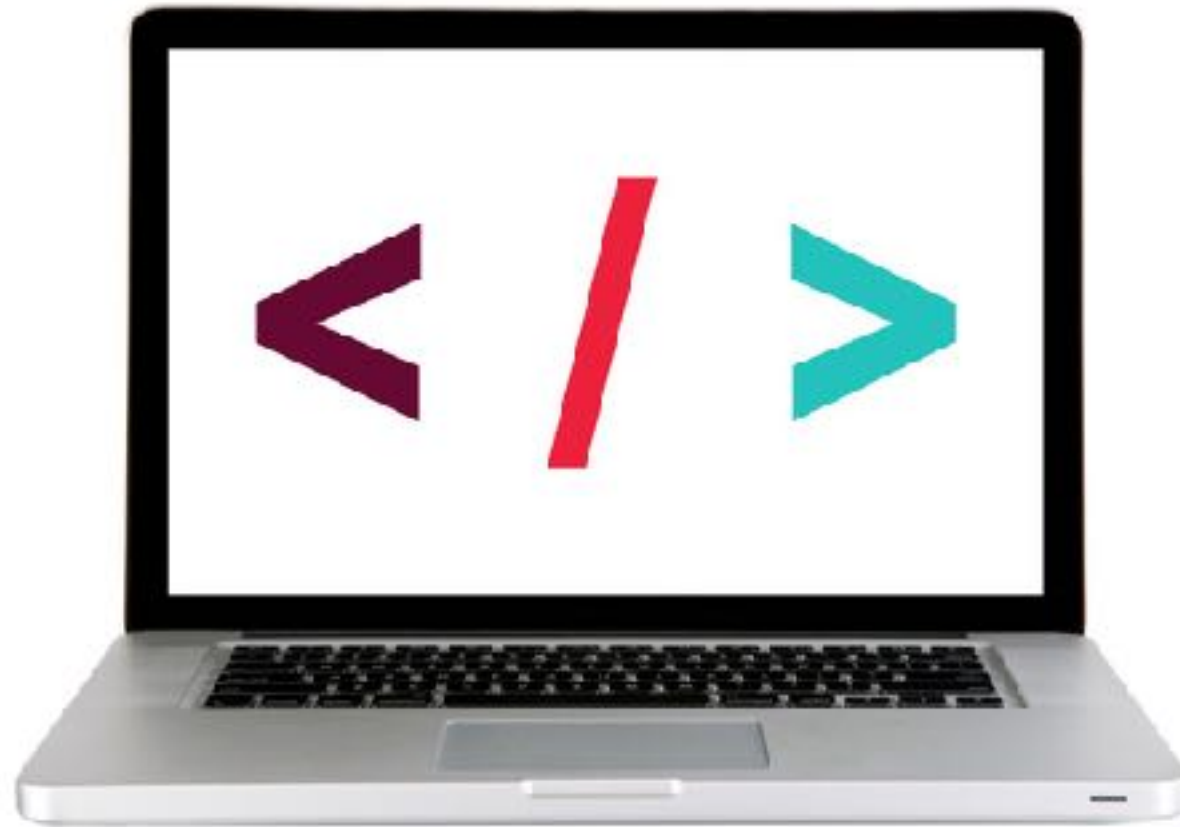
# VIRTUAL DOM



---

## LET'S TAKE A CLOSER LOOK

---



## COMPONENTS

The image shows a screenshot of the Facebook for Developers page. Red boxes highlight the following components:

- Page Header:** Includes 'Like', 'Share', 'Suggest Edits', and 'Send Message' buttons.
- Page Profile:** Features the Facebook for Developers profile picture, name, and handle (@FacebookforDevelopers).
- Left Navigation Menu:** A vertical list of links: Home, Posts, Videos, About, Photos, Events, Notes, and Community, with a 'Create a Page' button at the bottom.
- Main Content Area:** Contains the post text, a video player for the F8 2018 conference, and engagement metrics (35K Views, 179 Shares, 1,045 likes).
- Right Sidebar:** Includes sections for 'Community' (likes and followers), 'About' (typical reply time, contact info, and product/service), 'People Also Like' (recommendations for Facebook Analytics, Google AdSense, and Facebook Academics), and 'Pages liked by this Page'.



# **CREATING REACT COMPONENTS**

- Create a component function
  - initial cap in function name
  - props is parameter name
- Add a return statement to the component function
  - Contents should be JSX
  - Can include JavaScript expressions wrapped in `{ }`

# CREATING REACT COMPONENTS

function name has an initial cap

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

# CREATING REACT COMPONENTS

standard parameter name is props

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

# CREATING REACT COMPONENTS

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

function always includes a return statement

# CREATING REACT COMPONENTS

```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

content of the return statement is JSX

# CREATING REACT COMPONENTS

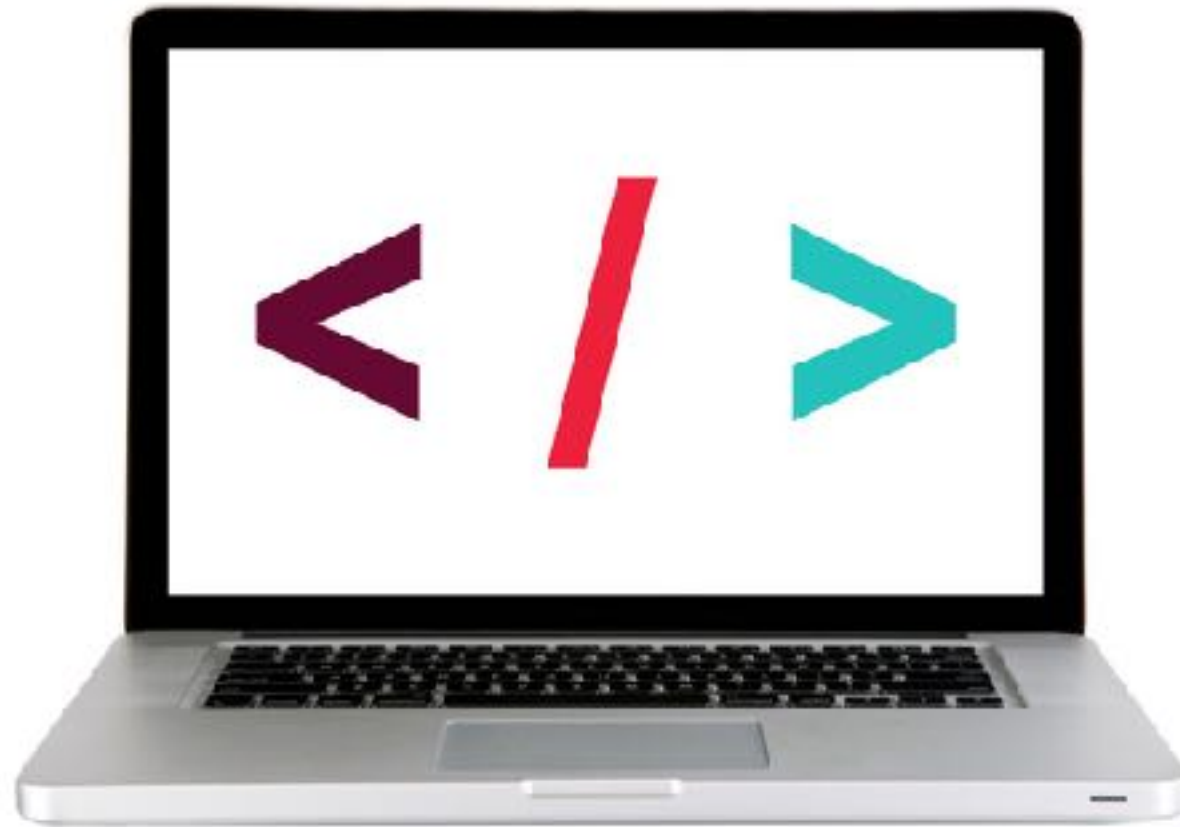
```
function Welcome(props) {  
  return <h1>Hello, {props.name}</h1>;  
}
```

JSX can include JavaScript expressions wrapped in {}

---

## LET'S TAKE A CLOSER LOOK

---



# JSX

- Extension to JavaScript
- Lets you write JavaScript code that looks like HTML (actually XML)
- Compiles to a JavaScript object
- Supports JavaScript expressions in curly braces



# ES6 SPREAD OPERATOR

- ... characters
- lets you specify an object as the parameter of a function, but transforms that argument into key-value pairs at runtime
- essentially setting key-value pairs as HTML attributes in the React code
- only evaluated at runtime, so it's based on the current value of the state at runtime

# ES6 SPREAD OPERATOR

```
{  
  firstName: 'Ben',  
  lastName: 'Hector'  
}  
  
return <Greeting {...props} />;
```

is parsed as

```
return <Greeting firstName="Ben" lastName="Hector" />;
```

# **LOOPING IN REACT COMPONENTS**

- Commonly used for an array of values
- `array.map()` function built into JavaScript
  - accepts a function as an argument
  - loops through the array, executing the specified function with each element as the argument
  - can return a JSX expression to build out an HTML structure based on a set of values

# EXERCISE — CREATE REACT COMPONENTS

---



## EXERCISE

### KEY OBJECTIVE

---

- Build a React component

### TYPE OF EXERCISE

---

- Solo or in pairs

### LOCATION

---

- `starter-code > 1-component-exercise`

### TIMING

---

*10 min*

1. The start file contains the components we've already been working with, along with additional data in the state variable.
2. Create variables storing references to the two new elements in the DOM.
3. Create components to render the contents of the new state properties.
4. Call the render method for each of your two new components.

# THINKING IN REACT

Data returned from a JSON API

```
[
  {category: "Sporting Goods", price: "$49.99", stocked: true, name: "Football"},
  {category: "Sporting Goods", price: "$9.99", stocked: true, name: "Baseball"},
  {category: "Sporting Goods", price: "$29.99", stocked: false, name: "Basketball"},
  {category: "Electronics", price: "$99.99", stocked: true, name: "iPod Touch"},
  {category: "Electronics", price: "$399.99", stocked: false, name: "iPhone 5"},
  {category: "Electronics", price: "$199.99", stocked: true, name: "Nexus 7"}
];
```

Mock from designer

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
Electronics	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99

# THINKING IN REACT

## DRAW SOME BOXES

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
Electronics	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99

# THINKING IN REACT

## NAME THE BOXES (SEMANTICALLY!)

- FilterableProductTable
- SearchBar
- ProductTable
- ProductCategoryRow
- ProductRow

The diagram illustrates a web application interface with several semantic components highlighted by colored boxes:

- SearchBar** (blue box): Contains a search input field with the placeholder text "Search..." and a checkbox labeled "Only show products in stock".
- ProductTable** (green box): A table with two columns, "Name" and "Price". It contains two sections: "Sporting Goods" and "Electronics".
- ProductCategoryRow** (cyan box): A row within the table that serves as a header for a category of products.
- ProductRow** (red box): Individual rows within the table, each representing a product with its name and price.

Name	Price
<b>Sporting Goods</b>	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
<b>Electronics</b>	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99

# THINKING IN REACT

## MAKE A HIERARCHY

components!

- FilterableProductTable
  - SearchBar
  - ProductTable
    - » ProductCategoryRow
    - » ProductRow

☐ Only show products in stock

Name	Price
Sporting Goods	
Football	\$49.99
Baseball	\$9.99
Basketball	\$29.99
Electronics	
iPod Touch	\$99.99
iPhone 5	\$399.99
Nexus 7	\$199.99



# EXERCISE

---



## EXERCISE

### KEY OBJECTIVE

---

- Create a component hierarchy

### TYPE OF EXERCISE

---

- Individual/pair

### TIMING

---

*7 min*

1. Choose a section of your favorite website
2. Write down the component hierarchy (remember the steps: 1. Mock, 2. Boxes, 3. Name, 4. Hierarchy)
3. Don't forget to use semantic names!

# BUILDING A PROJECT WITH NODE/NPM

package.json

```
{
  "name": "2-react-workshop",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "start": "http-server lib"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "http-server": "^0.11.1"
  }
}
```

# BUILDING A PROJECT WITH NODE/NPM

lib folder



# **REACT LAB**

- Created by Jess Telford, a GA JSD instructor in Australia

<https://github.com/svodnik/react-workshop>

# **Exit Tickets!**

**(Class #17)**

## **LEARNING OBJECTIVES – REVIEW**

- Understand the roles of model, view, and controller
- Describe the difference between frameworks and libraries
- Recognize the primary uses of React
- Create a component hierarchy
- Build a React component

# **NEXT CLASS PREVIEW**

## **Final project lab**

- › All of next class will be lab time for you to work on your final project.
- › Larissa and I will be available during class if you want to think through challenges together. (Your classmates will, too!)

# **Q&A**