



RICE[®]



COMP 431 / COMP 531

Model-View-Controller (and others)

Scott E Pollack, PhD

September 20, 2016

Recap

- HTML and HTML5, Storage, Canvas
- JavaScript and Scope
- Forms, CSS, Events
- jQuery, AJAX, and fetch
- Modern JS

COMP 531
Draft Front-End Review
Due Tuesday 9/27

~~Modern JS~~
MVC
React & Redux
Testing

Homework Assignment 3
(Draft Frontend)
Due **TONIGHT** by 2AM

Homework Assignment 4
(JavaScript Game)
Due Thursday 9/29

Separation of Concerns

In computer science, **separation of concerns** (SoC) is a design principle for **separating** a computer program into distinct sections, such that each section addresses a separate **concern**. A **concern** is a set of information that affects the code of a computer program. -Wikipedia

- Simplify development
- Increase maintainability
- Improve reusability
- Parallelize development
- Promotes Interface development and Encapsulation

SoC for a Web Application

- Content

HTML

- Presentation

CSS

- Behavior

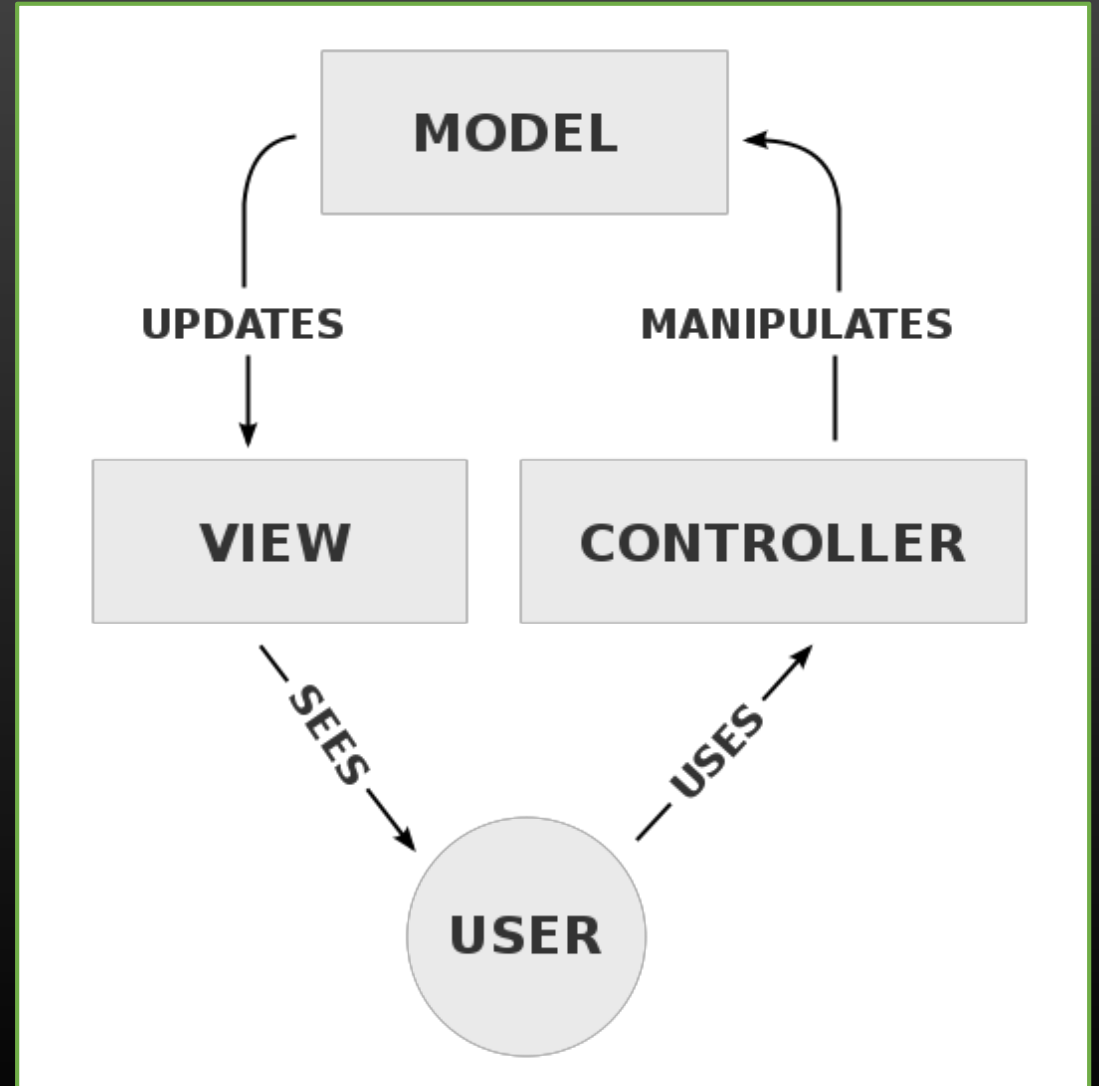
JAVASCRIPT

User interface design

- Dynamic layout
- Dynamic content
- Respond to user actions

Model-View-Controller

- **Model** contains the data
- **View** presents the data to the user in response to the **Model**
- **Controller** sends commands to update the **Model** (change data), or update the **View** (pagination)



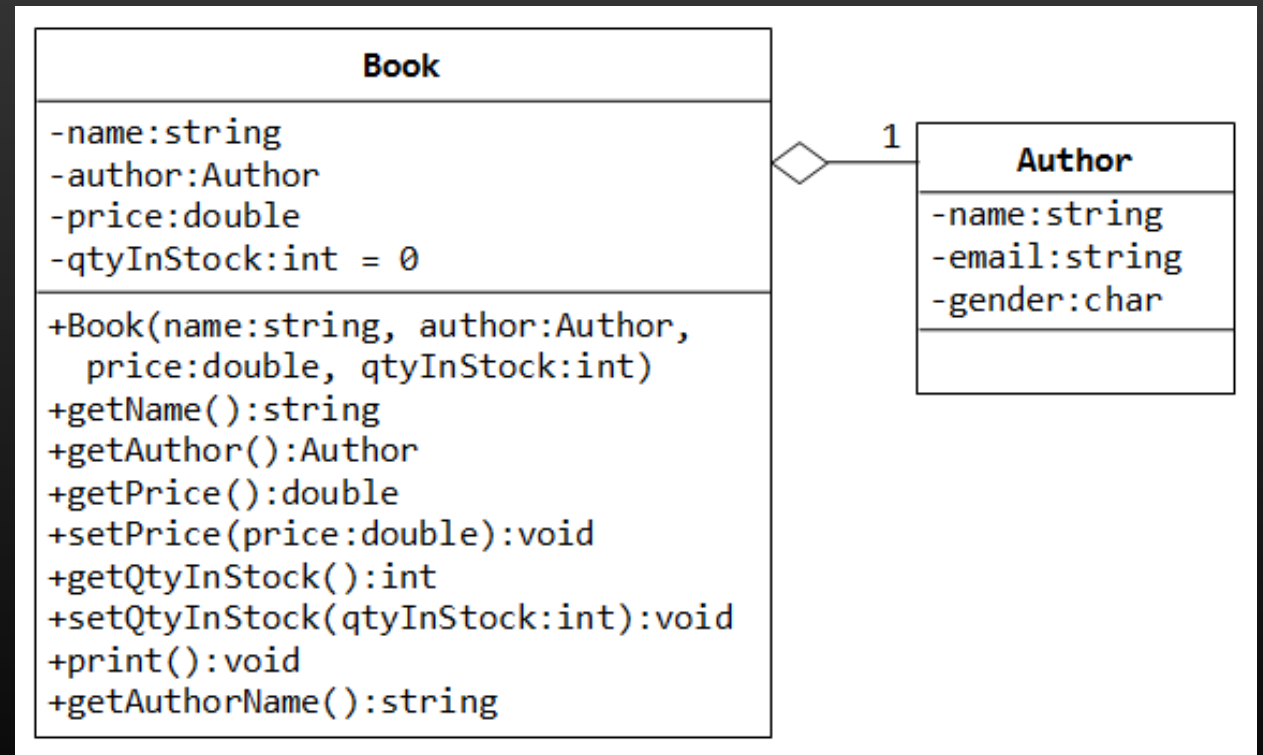
Example: Online Bookstore

- A database of books
 - Books have properties
 - Display all the books
 - Include selected properties
 - User can select books for purchase
- Model
 - A book
 - View
 - Presentation of books
 - Controller
 - Load model from database
 - Respond to user selection

Model of a Book

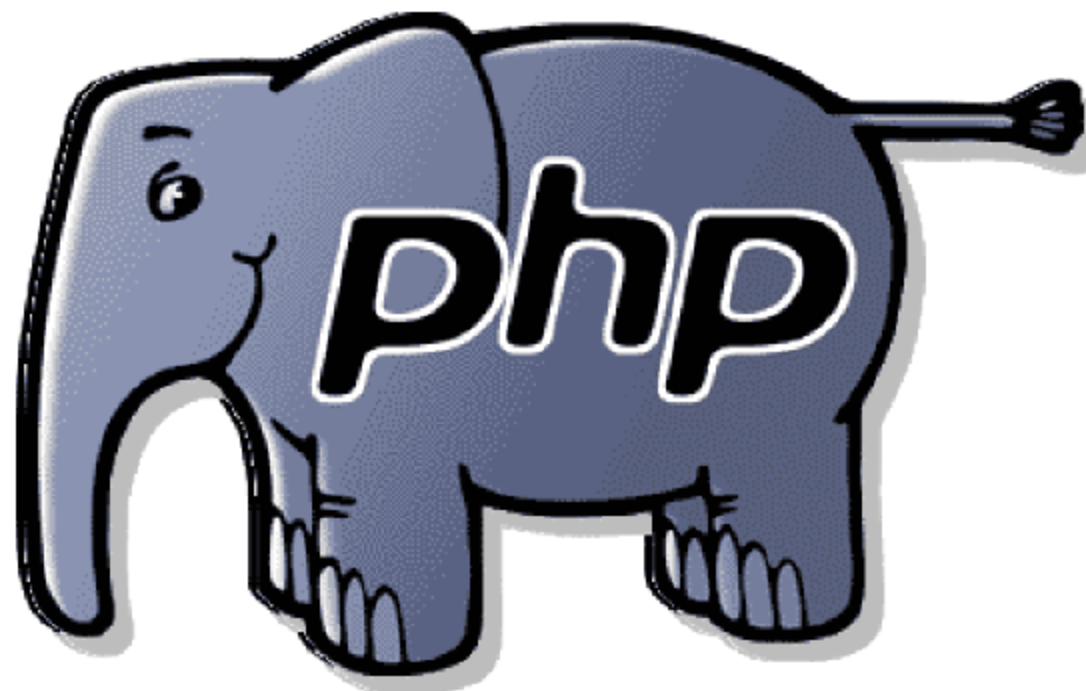
```
Book {  
    author: string  
    title: string  
    publicationDate: date  
    publisher: string  
    ISBN: string  
    price: float  
}
```

Classic UML Object Design





WARNING



“Dynamic” View of Books

```
<table>
  <tr>
    <th>Author</th><th>Title</th><th>Price</th>
  </tr>

  <?php foreach ($books as $book): ?>
    <tr>
      <td> <?php echo $book['author']; ?> </td>
      <td> <?php echo $book['title']; ?> </td>
      <td> <?php echo $book['price']; ?> </td>
    </tr>
  <?php endforeach; ?>

</table>
```

Book Controller

```
class BookController {  
    public function init() {  
        $this->Model = new BookModel();  
  
        // $books is in "scope" for the view  
        $this->$books = $this->Model->getAll();  
    }  
}
```

```
new BookController().init();
```

```
class Book {  
    public $author;  
    public $title;  
    ...  
    // create a Book from a database row  
    public function __construct($row) {  
        $this->$author = ...  
    }  
}
```

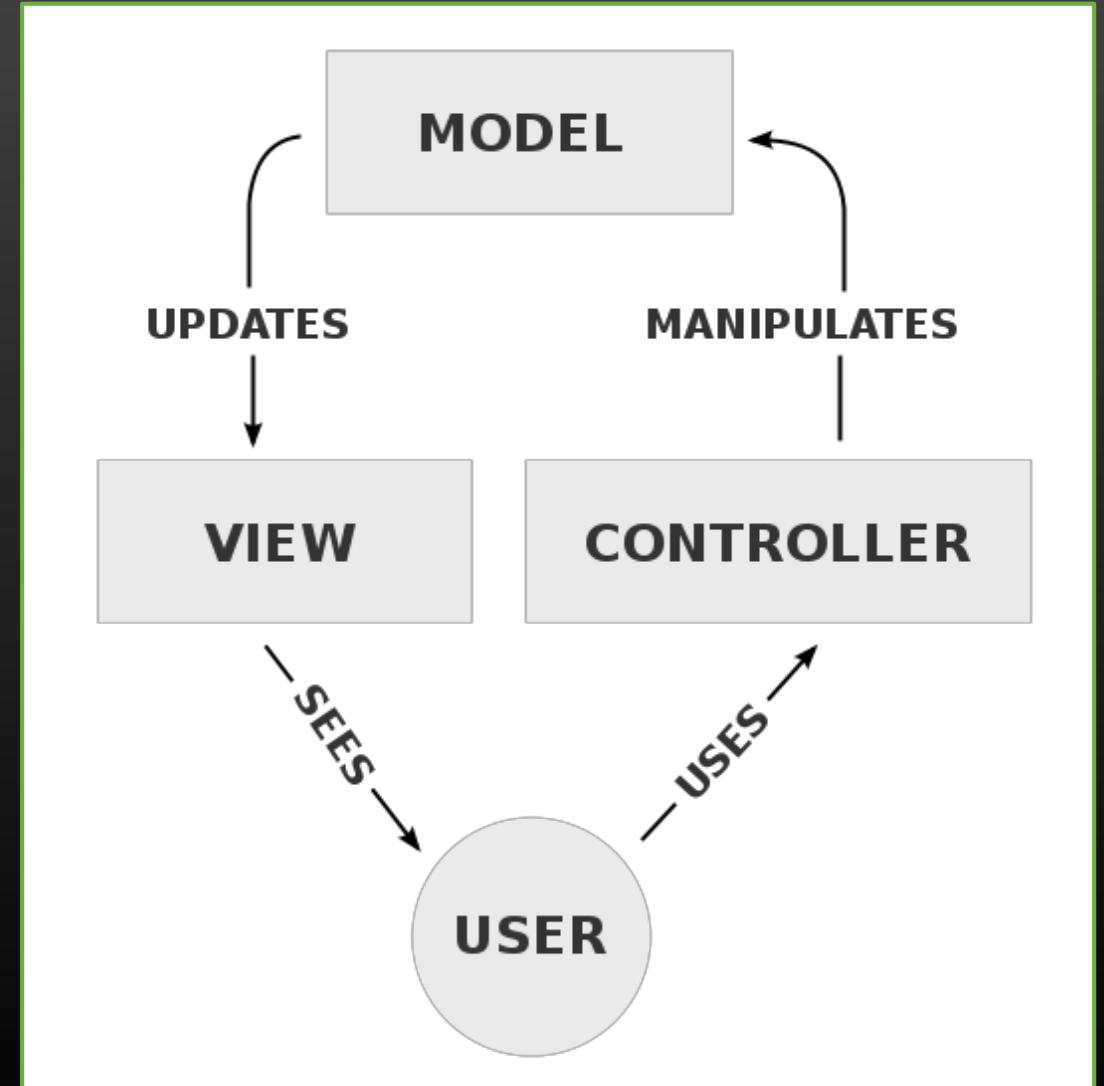
DTO

```
class BookModel {  
    // make database query to get all of the books.  
    // return the ResultSet as an array of Books  
    public function getAll() {  
        $results = array();  
        $resultSet = $this->db->getAllBooks();  
        foreach ($resultSet as $row) {  
            $results[] = new Book($row);  
        }  
        return $results;  
    }  
}
```

DAO

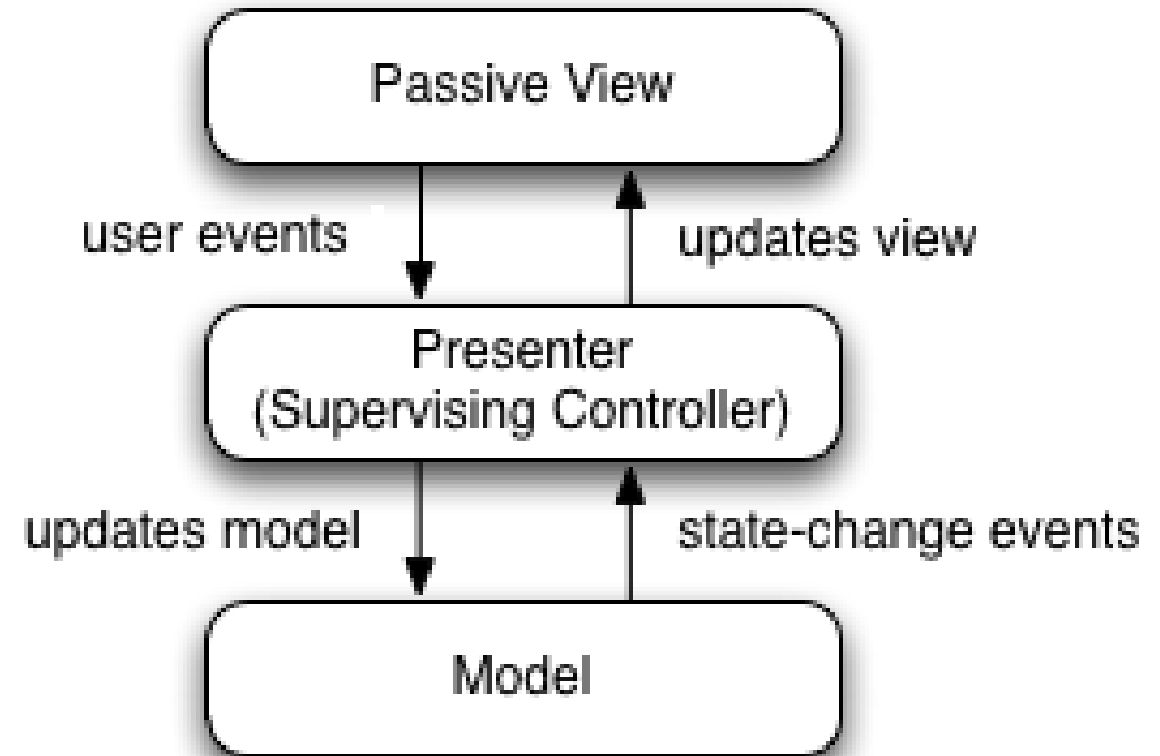
Model-View-Controller

- **Model** contains the data
- **View** presents the data to the user in response to the **Model**
- **Controller** sends commands to update the **Model** (change data), or update the **View** (pagination)

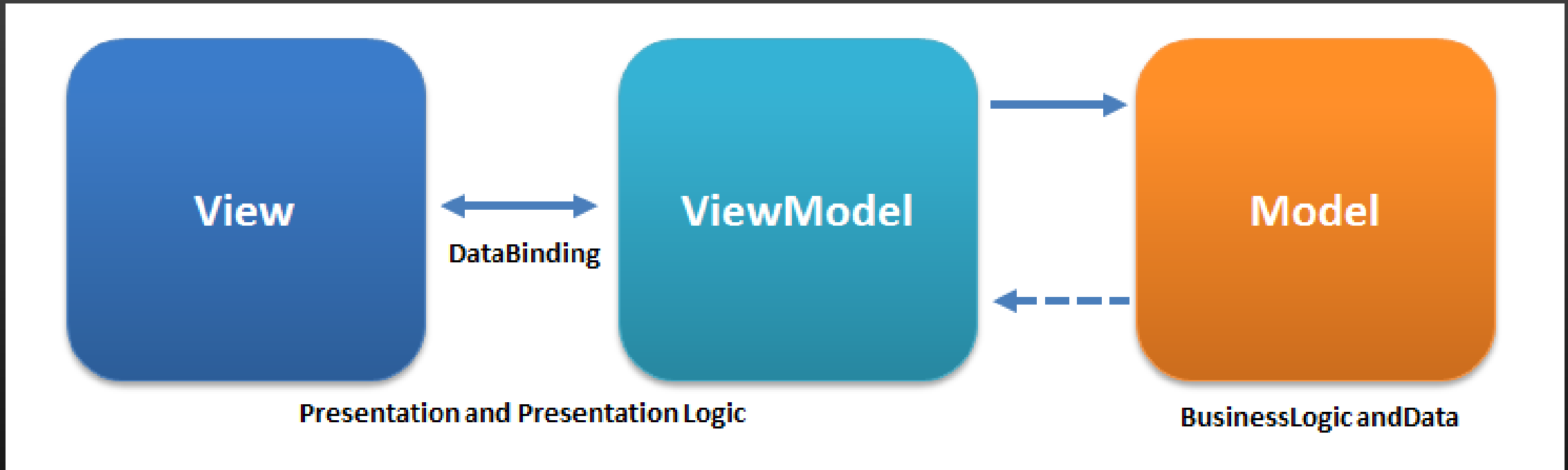


Model-View-Presenter

- **Model** contains the data
- **View** presents the data to the user in response to the **Model**
- ~~Controller~~ **Presenter** sends commands to update the **Model** (change data), or update the **View** (pagination)



Model-View-ViewModel



- **ViewModel** is a state representation of the data from the **Model**
- **Binder** utilizes **declarative data and command binding** in the markup for data synchronization

MVVM vs MVC

MVVM

1. ViewModel encapsulates presentation logic and state.
2. ViewModel is an optional Pattern.
3. User hits the View first
4. View can't see the Model
5. View has an instance of the ViewModel

MVC

1. The Controller determine the Application Flow.
2. Controller is a must.
3. User hits the Controller first
4. The View knows about the Model
5. View gets an instance of the Model

Front-End Frameworks

- Template engines for Views
- Utilize data-binding to populate View from ViewModel / Controller / Presenter
- Lots to choose from – different styles and techniques
- Some modular, others opinionated

View Template

Preprocessed on Server

```
<table>
  <tr>
    <th>Author</th><th>Title</th><th>Price</th>
  </tr>

  <?php foreach ($books as $book): ?>
    <tr>
      <td> <?php echo $book['author']; ?> </td>
      <td> <?php echo $book['title']; ?> </td>
      <td> <?php echo $book['price']; ?> </td>
    </tr>
  <?php endforeach; ?>
</table>
```

Front-End Controlled by JS

```
<tr data-repeat="book in books">
  <td>{{ book.author }}</td>
  <td>{{ book.title }}</td>
  <td>{{ book.price }}</td>
</tr>
```


The Mustache Template System

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <script src="http://cdnjs.cloudflare.com/ajax/libs/mustache.js/0.7.0/mustache.min.js"></script>
5     <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
6 </head>
7 <body>
8
9 <h1>Posts</h1>
10
11 <div id="posts"></div>
12
13 <script id="postTpl" type="text/template">
14 {{#posts}}
15 <h2>{{title}}</h2>
16 <h3>Posted {{date}} by {{author}}</h3>
17 <p>{{body}}</p>
18 <hr>
19 {{/posts}}
20 </script>
```

The Mustache Template

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="http://cdnjs.cloudflare.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
5   <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js"></script>
6 </head>
7 <body>
8
9 <h1>Posts</h1>
10
11 <div id="posts"></div>
12
13 <script id="postTpl" type="text/template">
14   {{#posts}}
15   <h2>{{title}}</h2>
16   <h3>Posted {{date}} by {{author}}</h3>
17   <p>{{body}}</p>
18   <hr>
19   {{/posts}}
20 </script>
```

Posts

Church-key irony meggings ...

Posted Wed Jul 01 2015 06:40:47 GMT-0500 (Central Daylight Time) by Hipster

Church-key irony meggings biodiesel chillwave bespoke. Excepteur adipisicing polaroid, ex ullamco delectus tilde do nostrud salvia Portland mlkshk sunt culpa. Meditation Tumblr Etsy, delectus sint aute meh chambray ex selfies. Consequat ad disrupt sed. Godard aliqua sed DIY eu, freegan squid art party. Craft beer flannel bitters before they sold out butcher, vegan tousled Godard. Nulla culpa flexitarian, butcher irony pickled polaroid forage 3 wolf moon mixtape hella anim placeat odio paleo.

8-bit aesthetic American ...

Posted Wed May 06 2015 12:07:44 GMT-0500 (Central Daylight Time) by Hipster

8-bit aesthetic American Apparel do pork belly. Fap Banksy roof party XOXO reprehenderit, officia cred organic Odd Future 8-bit biodiesel Brooklyn. Marfa distillery placeat Shoreditch et, bicycle rights eiusmod banjo retro. Master cleanse semiotics craft beer occaecat Banksy. Fugiat vegan cray, authentic kitsch banjo Banksy hashtag narwhal bespoke Marfa tilde iPhone. Sustainable plaid meditation, listicle fugiat selvage aesthetic ugh keffiyeh tilde health goth Godard artisan 8-bit. Mumblecore tofu cupidatat, deserunt skateboard sed health goth non farm-to-table Banksy sartorial Godard Etsy distillery.

Single-origin coffee post-ironic ...

Posted Sat Jul 18 2015 23:32:56 GMT-0500 (Central Daylight Time) by Hipster

Single-origin coffee post-ironic fap messenger bag. Organic High Life nulla viral, elit gentrify Kickstarter Blue Bottle kogi Carles. Post-ironic craft beer aesthetic, Williamsburg gastropub Godard street art occaecat letterpress placeat raw denim. Exercitation Kickstarter quinoa semiotics, listicle trust fund vero readymade kitsch cornhole taxidermy single-origin coffee salvia fap. Trust fund readymade blog pariatur, sriracha wayfarers minim street art slow-carb Carles occupy quis. Neutra in meditation, sint officia irony gastropub. Velit kitsch skateboard tousled, butcher chambray ea +1 Blue Bottle asymmetrical Tumblr cliché.

The Template System

```
<script id="postTpl" type="text/template">
{{#posts}}
<h2>{{title}}</h2>
<h3>Posted {{date}} by {{author}}</h3>
<p>{{body}}</p>
<hr>
{{/posts}}
</script>
```

```
<script>
window.onload = function() {
  var url = 'http://hipsterjesus.com/api/'

  $.get(url, function(result) {
    var posts = []
    result.text.split('<p>')
      .filter(function(t) { return t.length > 0 })
      .forEach(function(text) {
        var body = text.replace('</p>', '')
        var s = text.split(' ')
        var title = [s[0], s[1], s[2], '...'].join(' ')
        var entry = {
          "author": "Hipster",
          "title": title,
          "body": body,
          "date": new Date(1430012345678 + Math.random()*130e8)
        }
        posts.push(entry)
      })
    var template = $('#postTpl').html()
    var html = Mustache.to_html(template, {"posts":posts});
    $('#posts').html(html)
  })
}
</script>
```

Knockout JS

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.3.0/knockout-min.js"></script>
5   <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.3/jquery.min.js"></script>
6 </head>
7 <body>
8
9 <h1>Posts</h1>
10
11 <div id="posts" data-bind="foreach: posts">
12   <h2 data-bind="text: title"></h2>
13   <h3>Posted <span data-bind="text: date"></span>
14     by <span data-bind="text: author"></span></h3>
15   <p data-bind="text: body"></p>
16   <hr>
17 </div>
```

```
<script id="postTpl" type="text/template">
  {{#posts}}
  <h2>{{title}}</h2>
  <h3>Posted {{date}} by {{author}}</h3>
  <p>{{body}}</p>
  <hr>
  {{/posts}}
</script>
```

Mustache Template

Knockout JS

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <script src="https://cdnjs.cloudflare.com/ajax/
5   <script src="https://ajax.googleapis.com/ajax/1
6 </head>
7 <body>
8
9 <h1>Posts</h1>
10
11 <div id="posts" data-bind="foreach: posts">
12   <h2 data-bind="text: title"></h2>
13   <h3>Posted <span data-bind="text: date"></span>
14     by <span data-bind="text: author"></span></h3>
15   <p data-bind="text: body"></p>
16   <hr>
17 </div>
```

```
var template = $('#postTpl').html()
var html = Mustache.to_html(template, {"posts":posts});
$('#posts').html(html)
```

Mustache

```
function PostModel(text) {
  var body = text.replace('</p>', '')
  var s = text.split(' ')
  var title = [s[0], s[1], s[2], '...'].join(' ')

  this.author = "Hipster"
  this.title = title
  this.body = body
  this.date = new Date(1430012345678 + Math.random()*130e8)
}

function PostViewModel() {
  var vm = this;
  vm.posts = ko.observableArray([])

  var url = 'http://hipsterjesus.com/api/'
  $.get(url, function(result) {
    result.text.split('<p>')
      .filter(function(t) { return t.length > 0 })
      .forEach(function(text) {
        vm.posts.push(new PostModel(text))
      })
  })
}

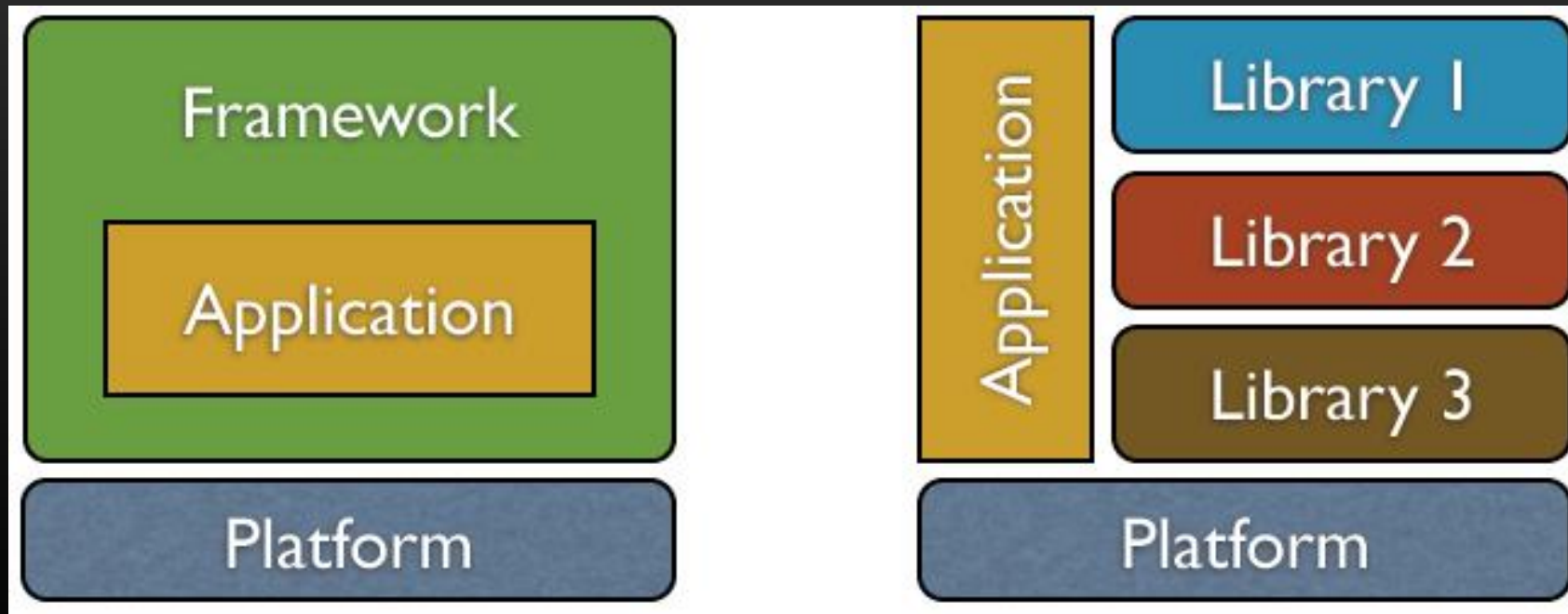
window.onload = function() {
  ko.applyBindings(new PostViewModel());
}
```

Front-End JavaScript Frameworks

- Mustache, Handlebars, Knockout
- Dojo, MooTools, Backbone
- Ember, Angular, React*

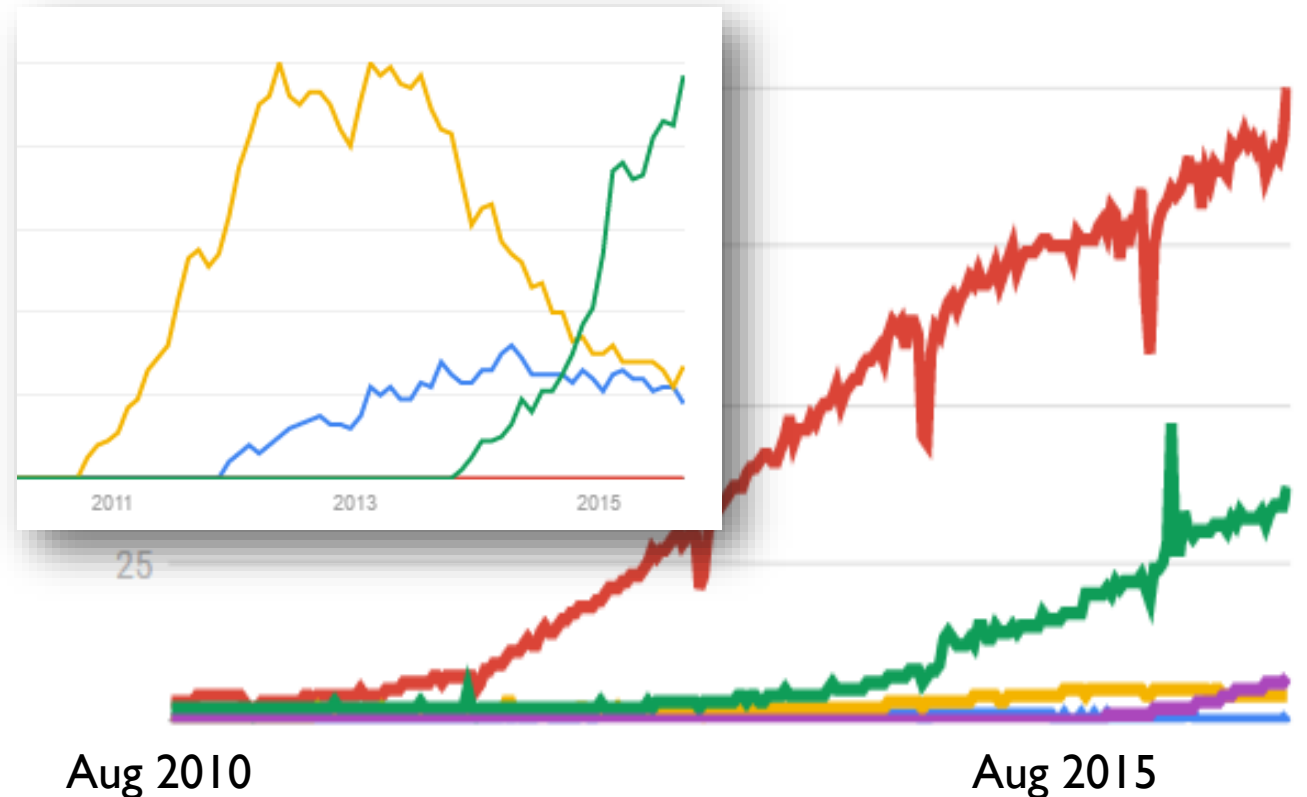
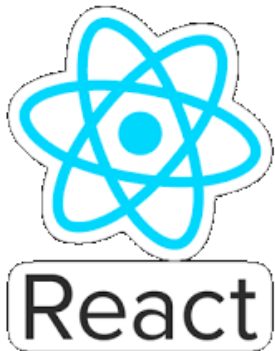
~ Library

FRAMEWORK



** React is more like a library than a framework*

Front-End JavaScript Frameworks



emberjs
Search term

angularjs
Search term

backbone.js ×
Search term

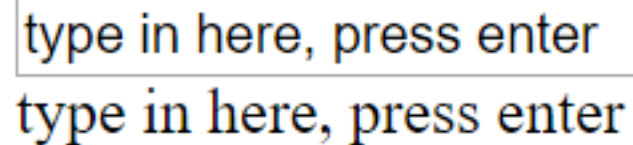
reactjs
Search term

● Meteor
● Angular2

Write our own Framework

- To have a better understanding of how template engines work
- Consider a simple page:

```
<body>  
  <input data-model="message" type="text" />  
  <div data-model="message"></div>  
  <script src="app.js"></script>  
</body>
```



type in here, press enter
type in here, press enter

- The idea:
 1. We have a map from model source to target(s)
 2. On page load, register elements as sources or targets
 3. Sources have event handlers that update the targets

<https://www.clear.rice.edu/comp431/sample/framework>

<https://www.clear.rice.edu/comp431/sample/framework/app.js>

What about dynamic elements?

```
<tr data-repeat="book in books">  
  <td>{{ book.author }}</td>  
  <td>{{ book.title }}</td>  
  <td>{{ book.price }}</td>  
</tr>
```

What about dynamic elements?

- Problem:

- Updating the DOM is slow

- Solution:

- DOM fragments
 - see John Resig's post <http://ejohn.org/blog/dom-documentfragments/>
 - See David Walsh's post <https://davidwalsh.name/documentfragment>

- Problem:

- Multiple edits to the DOM?

- Solution:

- Virtual DOM

Virtual DOM

- An in-memory representation of the DOM
 - it's an abstraction of an abstraction!

```
<div>
  <p class="title">A list of stuff</p>
  <ul class="stuff">
    <li><span class="fancy">thing1</span></li>
    <li>thing2</li>
    <li></li>
  </ul>
</div>
```

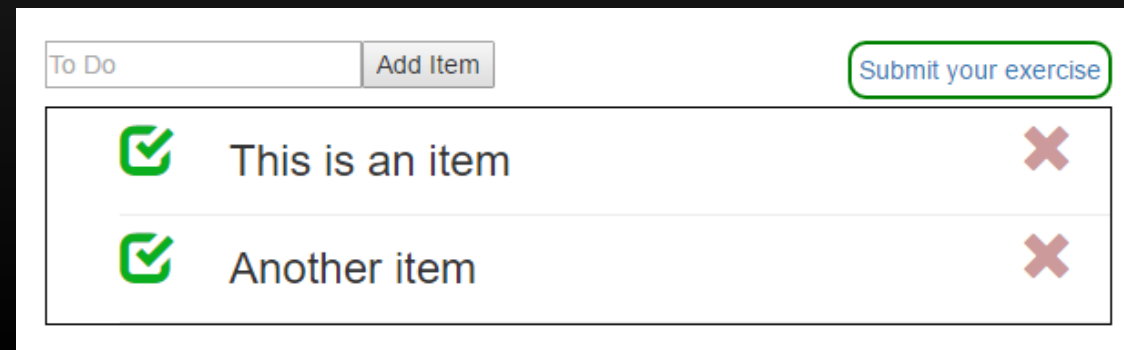
```
h('div', null, [
  h('p', { className: 'title' }, 'A list of stuff'),
  h('ul', { className: 'stuff' }, [
    h('li', null, h('span', { className: 'fancy' }, 'thing1')),
    h('li', null, 'thing2'),
    h('li', null, h('img', { src: 'seuss.png', alt: 'cat' }))
  ])
])
```

In-Class Exercise: Virtual DOM

/inclass-9/...

```
inclass-9/  
|-- index.html  
|-- src  
|   |-- index.js  
|   `-- vdom.js  
`-- styles.css  
  
1 directory, 4 files
```

- Download and unzip <https://www.clear.rice.edu/comp431/sample/vdom.zip>
- We are going to write our own virtual DOM
- Your task is to implement `createElement()` and `updateElement()` in `src/vdom.js`
- When completed the page should load like the image below
 - The check box should be functional (strike through)
 - The X should remove the task
 - “Add Item” adds new items.



The screenshot shows a web application titled "To Do". It features a text input field, an "Add Item" button, and a "Submit your exercise" button. Below the input field is a list of two items: "This is an item" and "Another item". Each item has a green checkmark icon on the left and a red 'X' icon on the right, indicating that the checkmarks are functional and the 'X' icons are used for removal.