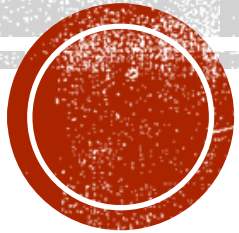
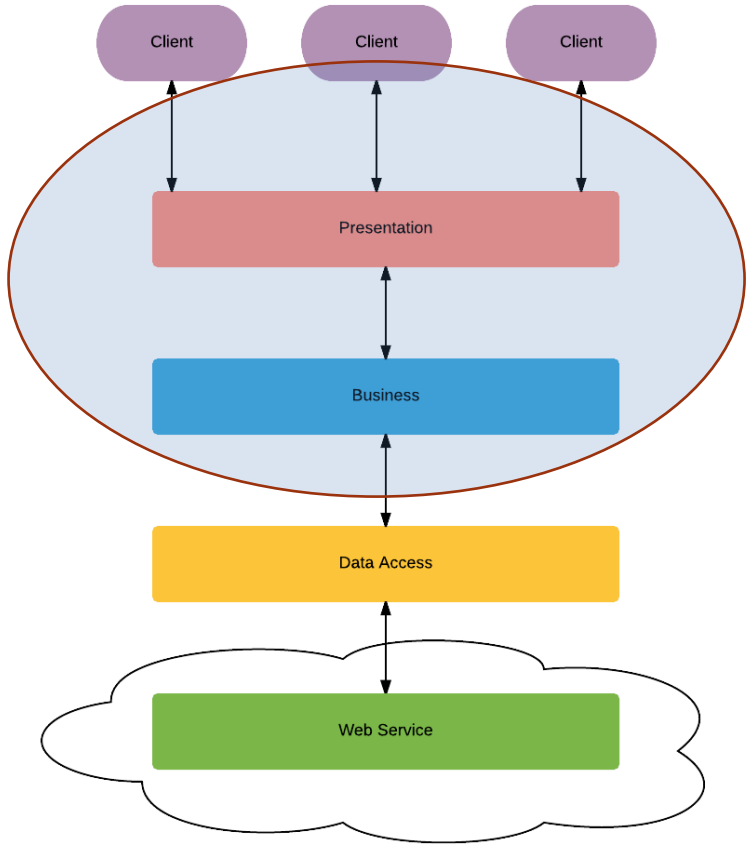
RAWDATA SECTION 3

Troels Andreasen & Henrik Bulskov



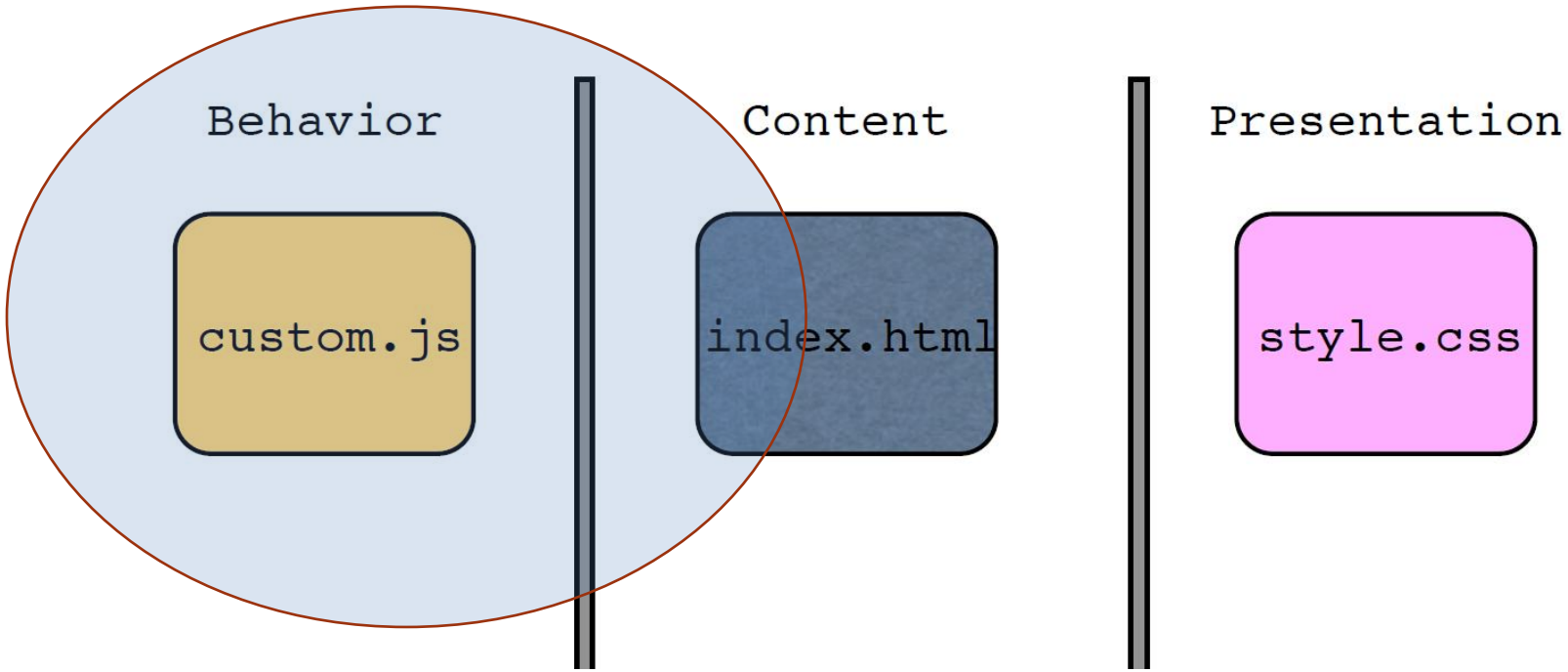
WHAT TO DO IN SECTION 3?

- JavaScript
  - Functions
- **Single Page Applications**
  - Databinding
  - Modularity
- Responsiveness
  - Adaptive applications
  - Bootstrap



SYSTEM DEVELOPMENT

UNOBTRUSIVE DESIGN



JS THE IMPORTANT PARTS

- Dynamic
- Untyped
- First-class Functions
  - Function scope
- Protype-based
- Fun, relaxing, and powerful
- To do it right – needs discipline!

AJAX

Asynchronous JavaScript and XML

A way to update a page without reloading

JQUERY AJAX

- Allows parts of a page to be updated
- Cross-Browser Support(Polyfill)
- Simple API
- GET and POST supported
- Load JSON, XML, HTML og even scripts

JQUERY AJAX FUNCTIONS

- $(selector).load()
  - Loads HTML data from the server
- $.get() and $.post()
  - get raw data from server
- $.getJSON()
  - Get/Post and return JSON
- $.ajax()
  - Provides core functionality

http://james.padolsey.com/jquery/

THE AJAX() FUNCTION

- **The ajax() function provides extra control over making Ajax calls to a server**
- **Configure using JSON properties:**
  - contentType
  - data
  - dataType
  - error
  - success
  - type (GET or POST)

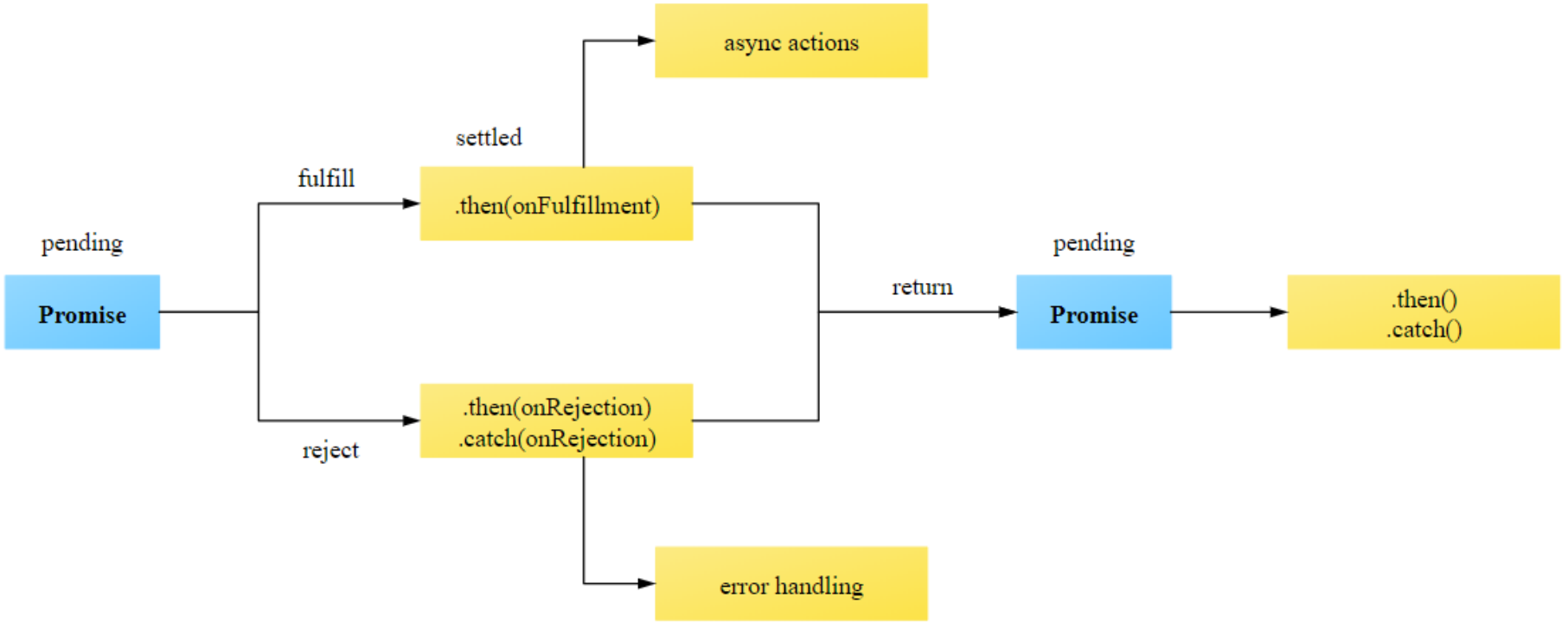
http://api.jquery.com/jQuery.ajax/

ES6 AJAX ALTERNATIVE

- fetch()
  - API that provides an interface for fetching resources (including across the network)
- Promises
  - A Promise is a proxy for a value not necessarily known when the promise is created

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

ES6 PROMISES

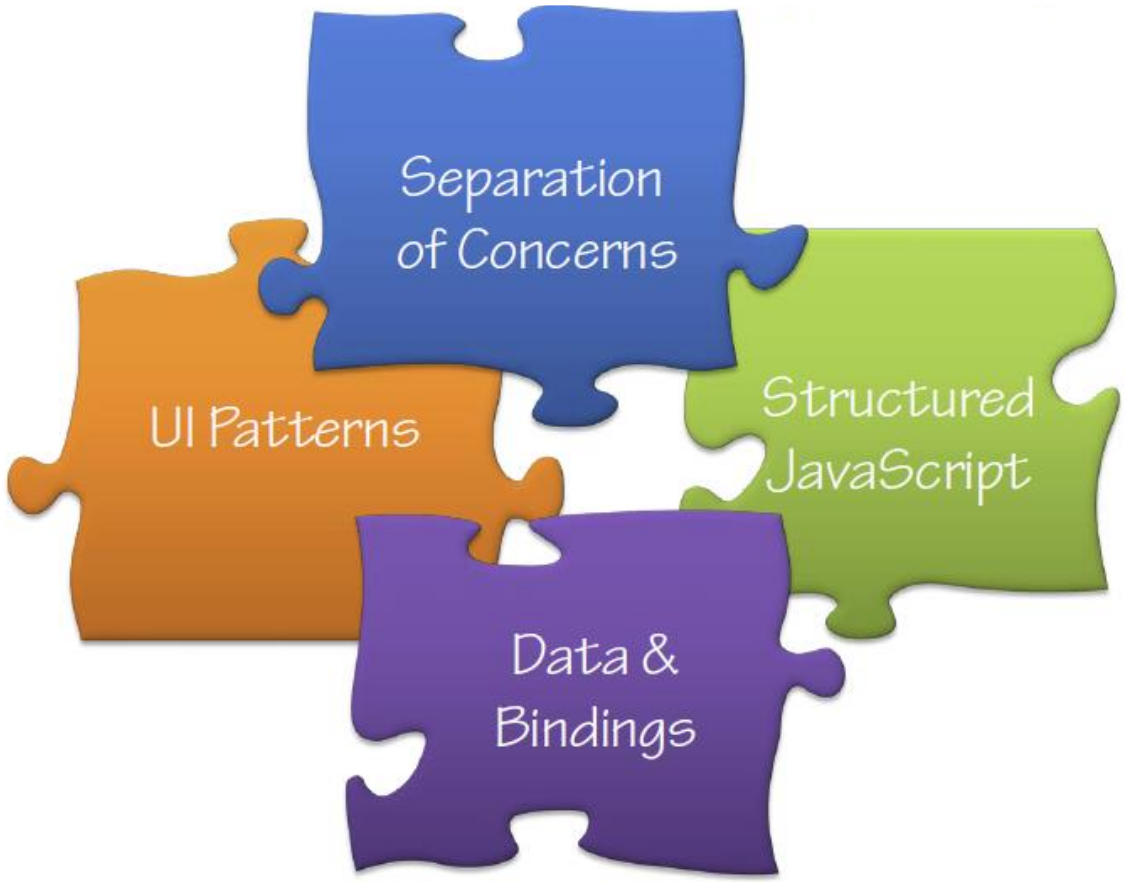


https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

ES6 ASYNC AJAX EXAMPLES

```
var getPosts = function (data) {
    fetch("api/posts", { method: 'GET' })
        .then(function (response) {
            return response.json();
        })
        .then(function (json) {
            data(json);
        });
};

var createPost = function (post, callback) {
    var headers = new Headers();
    headers.append("Content-Type", "application/json");
    fetch("api/posts", { method: 'POST', body: JSON.stringify(post), headers })
        .then(response => response.json())
        .then(json => callback(json));
}
```



JAVASCRIPT DEVELOPMENT

MODULAR PROGRAMMING

- Moving from spaghetti to ravioli coding will create several modules/files
- You need to add them in the right order in your HTML file
- One solution is to use asynchronous module definition (AMD), i.e. load the files when they are needed
- ES6 introduce a module system

REQUIRE.JS

- Is a file and module loader lib
  http://requirejs.org/

- You can require and define files and modules

3 STEPS TO USE REQUIRE.JS

1. Include require.js in your HTML file

```
<script data-main=”js/main.js" src=”js/lib/require.js"></script>
```

2. Configure require, normally in the data-main file

```
(function () {
    requirejs.config({
        baseUrl: 'Scripts',
        paths: {
            knockout: 'lib/knockout-3.4.0',
            jquery: 'lib/jquery-2.2.3.min',
            text: 'lib/text',
            bootstrap: 'lib/bootstrap.min'
        }
    });
})();
```

3 STEPS TO USE REQUIRE.JS

3. Use require to load your files

```
define(['knockout', 'app/config'], function (ko, config)
{
        ...
});

require(['knockout', 'app/viewmodel', 'app/config'],
    function (ko, vm, config) {
    ...
});
```

```
requirejs.config({
    baseUrl: 'Scripts',
    paths: {
        knockout: 'lib/knockout-3.4.0',
        jquery: 'lib/jquery-2.2.3.min',
        text: 'lib/text',
        bootstrap: 'lib/bootstrap.min'
    }
});
})();
```

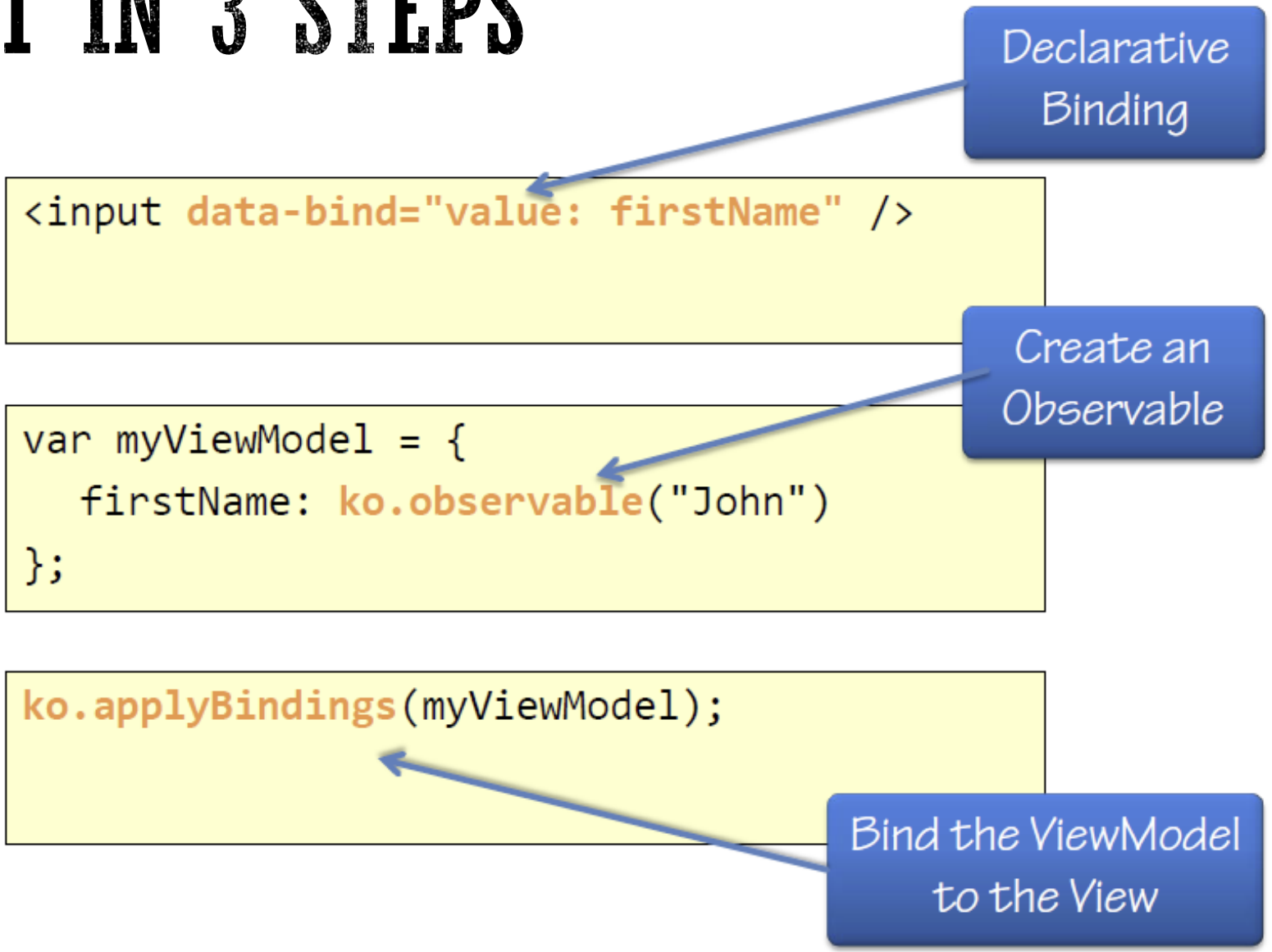
KNOCKOUT

WHAT IS KNOCKOUT?

- JavaScript MVVM Framework
- MVVM – Model-View-View Model
  - Model – objects in your business domain
  - View – user interface that is visible to user
  - View Model – code representing data/operations on a UI
- Complementary to other JavaScript frameworks
  - e.g., jQuery, CoffeeScript, Prototype, etc.

KEY KNOCKOUT CONCEPTS

KNOCKOUT IN 3 STEPS



YOUR FIRST KNOCKOUT

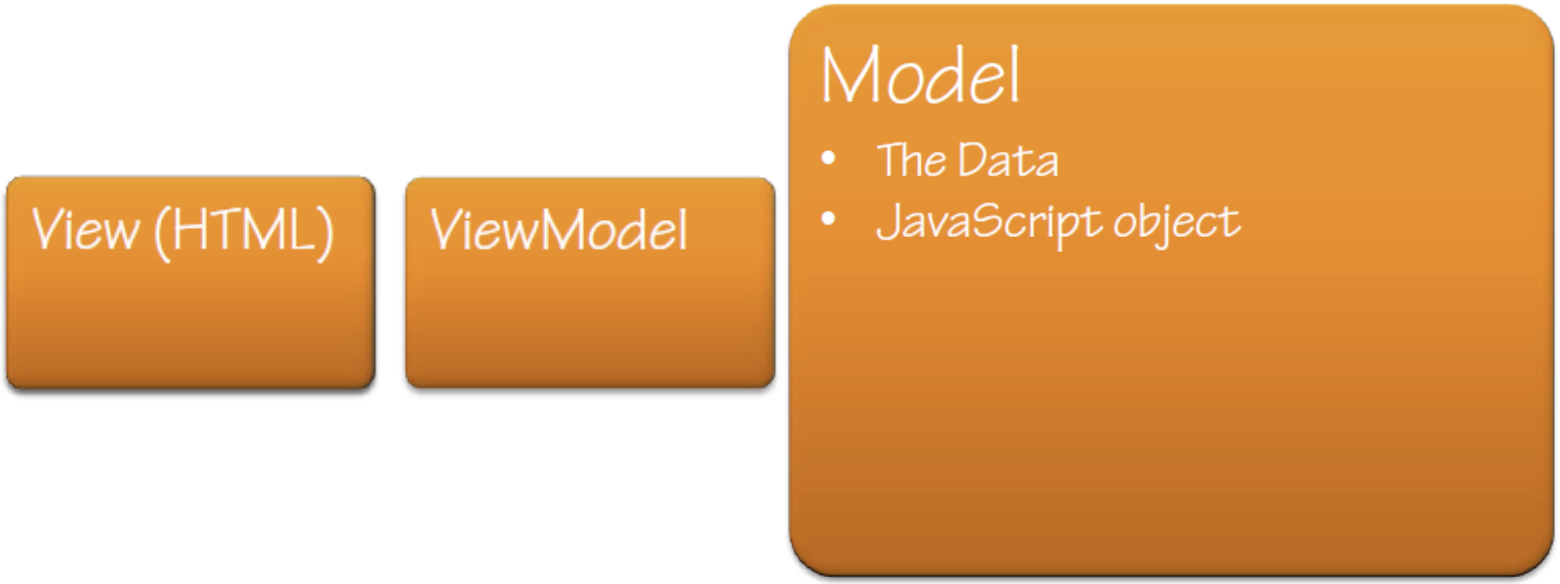
- Resources:
  - http://knockoutjs.com/
  - http://blog.stevensanderson.com/
  - http://www.knockmeout.net/
  - http://stackoverflow.com/questions/tagged/knockout.js
  - https://groups.google.com/forum/#!forum/knockoutjs

SEPARATION, ORGANIZATION, DATA BINDING

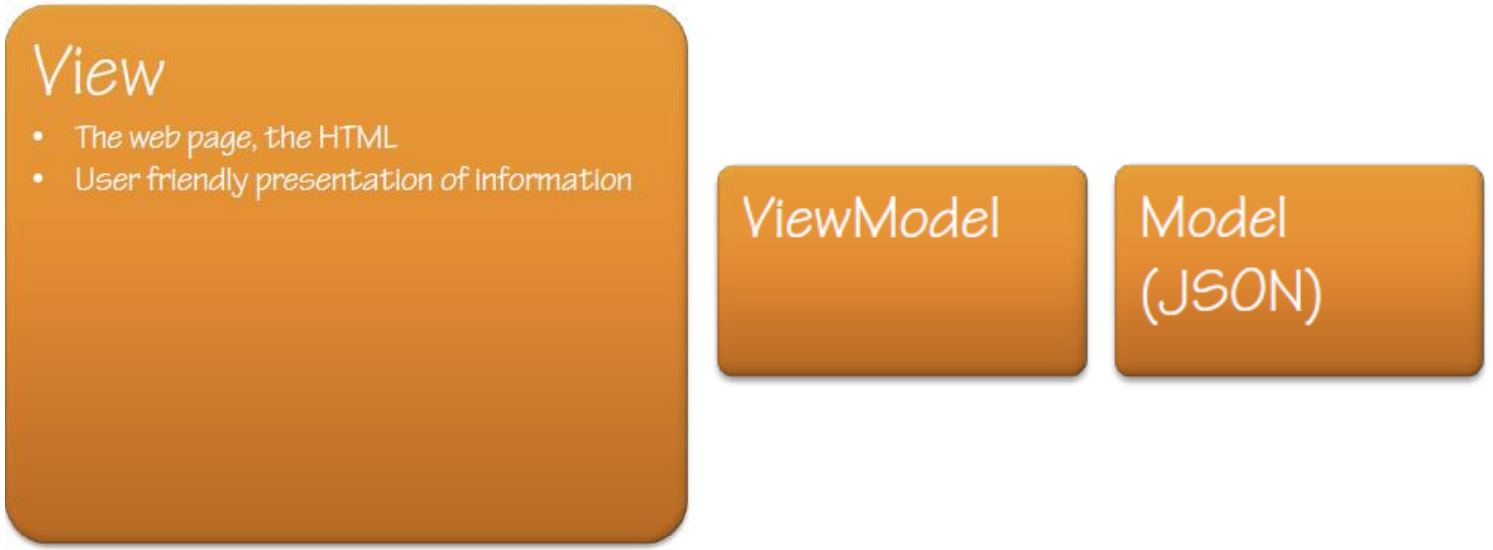
- MVVM
  - Foremost, a separation pattern
  - Model – View – ViewModel
  - Not technology specific
  - Works well with data binding

MVVM COMPONENTS

MVVM COMPONENTS



MVVM COMPONENTS



MVVM COMPONENTS

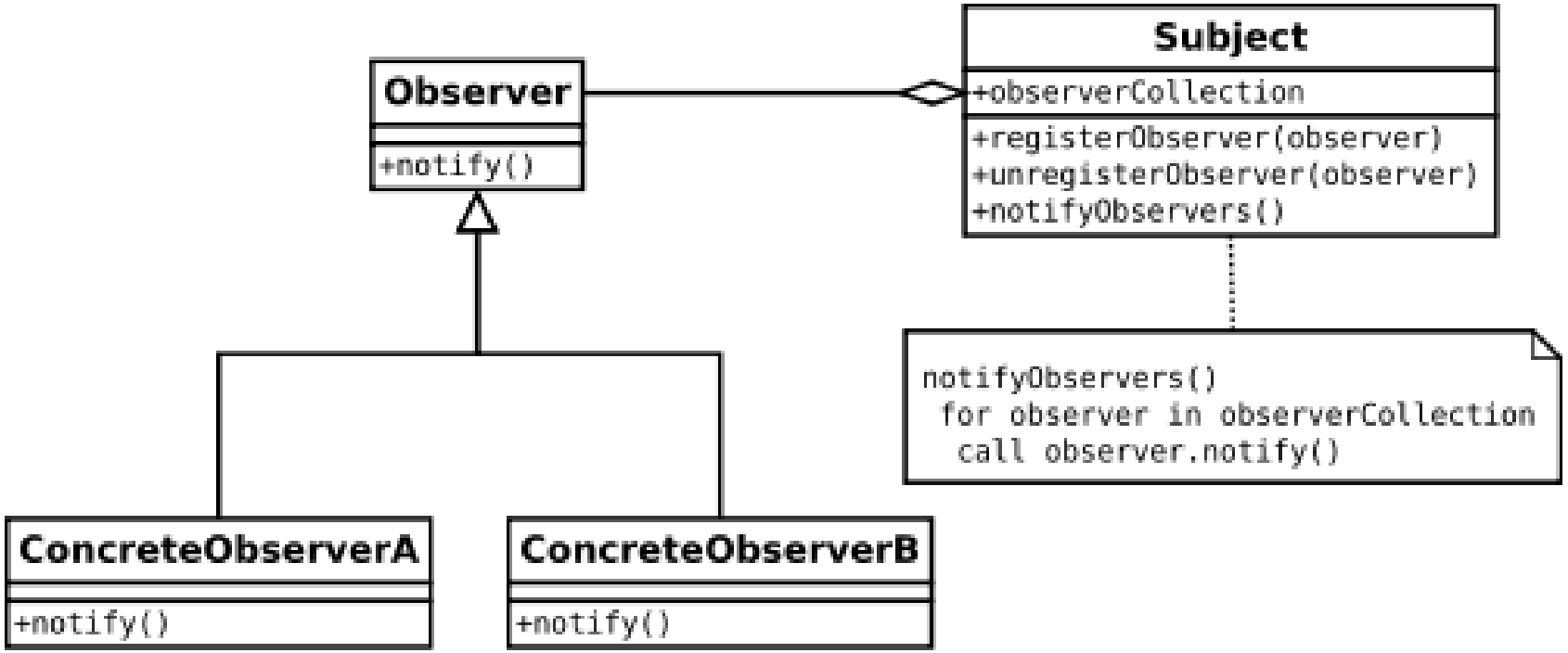
View

ViewModel

- Behavior and Data for the View
- Contains Properties, Methods & the Model

```
viewmodel = {
    id: ko.observable("123"),
    salePrice: ko.observable(1995),
    rating: ko.observable(4),
    isInStock: ko.observable(true),
    guitarModel: {
        code: ko.observable("314ce"),
        name: ko.observable("Taylor 314 ce")
    },
    showDetails: function () {
        /* method goes here */
    }
};
```

Model (JSON)



OBSERVER PATTERN

https://en.wikipedia.org/wiki/Observer_pattern

OBSERVABLES

- JavaScript functions
  - Not all browsers support JavaScript getters/setters
- Internally KO's bindings observe the observables

```
// read a value
var name = viewModel.name();

// write a value
viewModel.name("Peter");
```

3 TYPES OF OBSERVABLES

- Observable
  - Used for view model properties
- Observable array
  - Used for collections
- Computed observable
  - Encapsulate one or more other observables

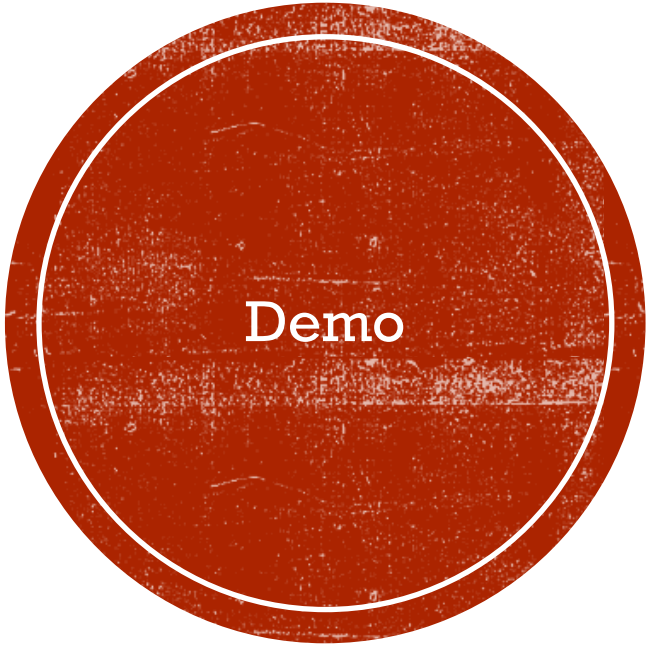
COMPUTED OBSERVABLE

- Encapsulate one or more observables
- Need to manage this pointer

```
var viewModel = {
    firstName: ko.observable("Peter"),
    lastName: ko.observable(”Smith")
};

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

COMPUTED OBSERVABLE



OBSERVABLE ARRAYS

- Use with collections
- Detect changes to collection – add/remove
- Use Knockout array methods
  - Cross browser
  - Dependency Tracking
  - Clean Syntax

OBSERVABLE ARRAY METHODS

list.indexOf("value")	Returns zero-based index of item
list.slice(2, 4)	Returns items between start/end index
list.push("value")	Adds new item to end
list.pop()	Removes last item
list.unshift("value")	Inserts item at beginning
list.shift()	Removes first item
list.reverse()	Reverses order
list.sort()	Sorts the items
list.remove(item)	Removes specified item
list.removeAll()	Removes all items

OBSERVABLE ARRAYS