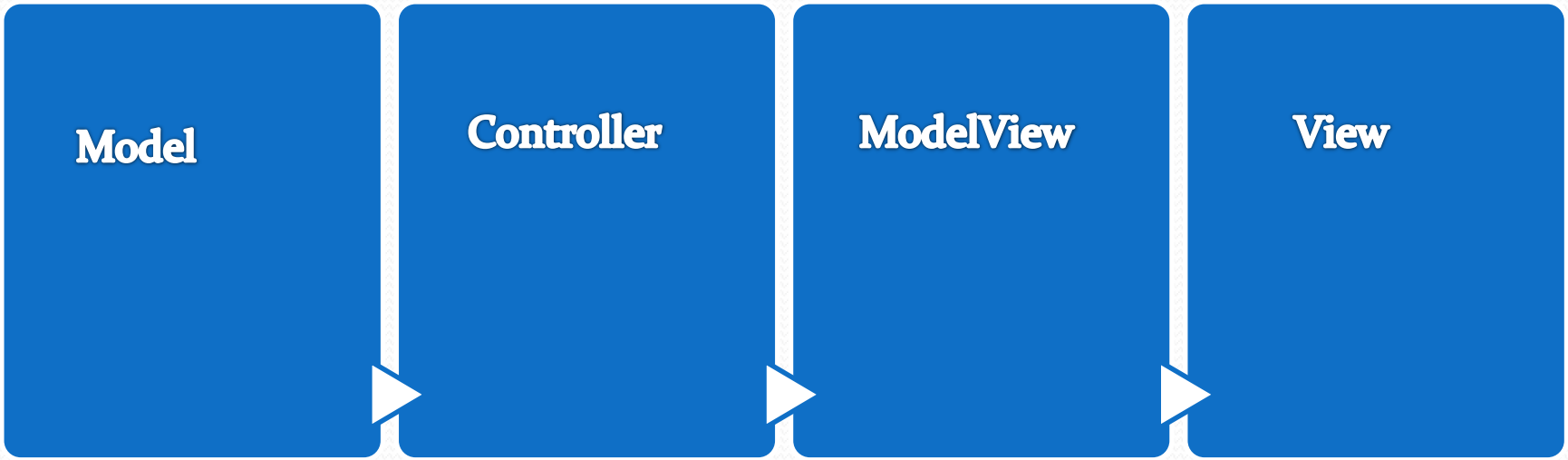
# Introduction to React

# What is React?

- It is a library for creating user interfaces
  - Most people think it is a framework  but it is library

# Traditional MV* Frameworks Separation of Concerns



# Building Components not templates

- Components are the building blocks of React.
- Very similar to Directives in Angular JS
- You can think of a component as a collection of HTML, CSS, JS
- Components can be written in pure Javascript or JSX

# Pure javascript

## Creating the UI Description

```
var child1 = React.createElement('li', null, 'First Text Content');
var child2 = React.createElement('li', null, 'Second Text Content');
var root = React.createElement('ul', { className: 'my-list' }, child1,
child2);
React.render(root, document.getElementById('example'));
```

# JSX (Javascript XML)

```
var Header = React.createClass({
        render: function () {
            return (
                <h1>Employee Directory</h1>
            );
        }
});
```

# Benefits from React

- You avoid expensive DOM operations
- Minimize access to the DOM
- Update elements offline before inserting to the DOM

# React Virtual Dom

- Re-render everything on every update?
- It sounds expensive but it is not

- React creates an in-memory data structure cache, computes the resulting differences, and then updates the browser's displayed DOM efficiently
    1. Create lightweight description of component UI
    2. Diff it with the old one
    3. Compute minimal set of changes to apply to the DOM

# Benefist of Virtual DOM

- Clean
  - Clear  and descriptive programming model
- Fast
  - Optimized DOM updates

# Creating components

```
var Header = React.createClass({
    render: function () {
        return (
            <h1>Employee Directory</h1>
        );
    }
});
```

# Component rendering

- Value returned from render() method is not a DOM node
- It is a lightweight description of the UI
- It is diffed with the current description to perform the minimum set of changes

# Data Flow
## Parent-to-child

- Properties passed to children by parent are accessible in **this.props**

- Properties are immutable
  - Once you instantiate the object, you can't change its properties

# Parent

```
var HomePage = React.createClass({
    render: function () {
        return (
            <div>
                <Header title="Employee Directory"/>
                <SearchBar/>
                <EmployeeList/>
            </div>
        );
    }
```

# Child

```
var Header = React.createClass({
    render: function () {
        return (
            <h1>{this.props.title}</h1>
        );
    }
});
```

# Inverse data flow Parent

```
var HomePage = React.createClass({
    searchHandler:function(key) {
        alert('Search key: ' + key);
    },
    render: function () {
        return (
            <div>
                <Header text="Employee Directory"/>
                <SearchBar searchHandler={this.searchHandler}/>
                <EmployeeList/>
            </div>
        );
    }
});
```

# Child

```
var SearchBar = React.createClass({
    keyChange: function(event) {
        var searchKey = event.target.value;
        this.props.searchHandler(searchKey);
    },
    render: function () {
        return (
            <input type="search" onChange={this.keyChange}/>
        );
    }
});
```

# State object

- Like properties, state affects how a component behaves and renders. Unlike properties, there's no way to define what state should be applied to components via JSX

- Properties are defined when components are created, whether by JSX or by pure JavaScript. State, on the other hand, is only seen on the inside of component definitions. This is the first, and most important difference between the two.

# Setting initial state

# Changing state

# this.setState function

- Whenever this function is called the UI is automatically re-rendered.
  - Just the part of app that has changed.

# Thank you for listening