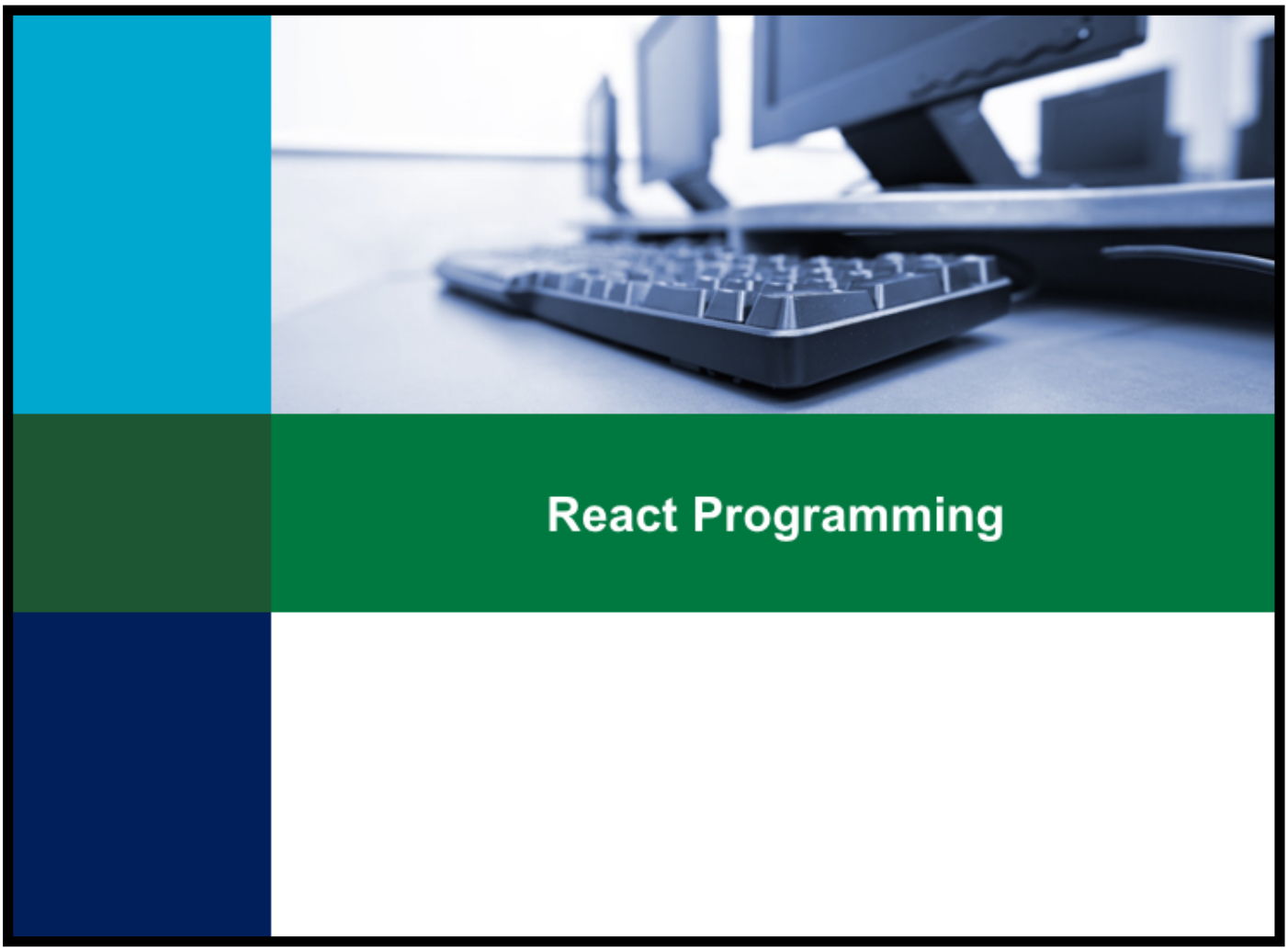


Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Overview

Course Overview

This course explores using the React framework to build client-side single-page applications. React stresses the programmatic creation of interface elements instead of static definitions. JSX is used to define interface components as tags inline in JavaScript code, and the code determines exactly what components will be built. The participant will learn about the component lifecycle, how to define components as classes in React, and work with props and state. Using react-router to manage views, and Redux to manage the flow of data are introduced. The course is fast-paced and relies on the participant having a firm grasp of HTML, CSS, and advanced JavaScript 2015 programming.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Target Audience and Prerequisites

- Target Audience
  - Client-side programmers who need to build robust applications
  - Anyone who wants to understand how React applications work
- Course prerequisites:
  - Understanding of HTML elements and CSS
  - Solid JavaScript and ES6 (ES2015) programming skills, including debugging scripts

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Objectives

- Introduce the React framework
- Build modular applications using React principles
- Add Redux to the application to control the flow of data

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Agenda

- Agenda
  - Module 1 React
  - Module 2 Component Development
  - Module 3 State
  - Module 4 DOM Abstraction
  - Module 5 Architecture with Components
  - Module 6 Forms
  - Module 7 react-router 4
  - Module 8 Redux
- Labs are named: the lab sequence is: Fundamentals, Components, State, Keys, Fetch, Forms, React-Router4, and Redux.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Focus

- Set your cell phones, smart phones, and pagers to vibrate
- Know where the restroom and break facilities are
- Adhere to the class hours and attendance requirements
- When is lunch?

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 0-6

Notes

Class Introductions

- Introduce yourself!
- What is your related experience?
- Why are you participating in this class?

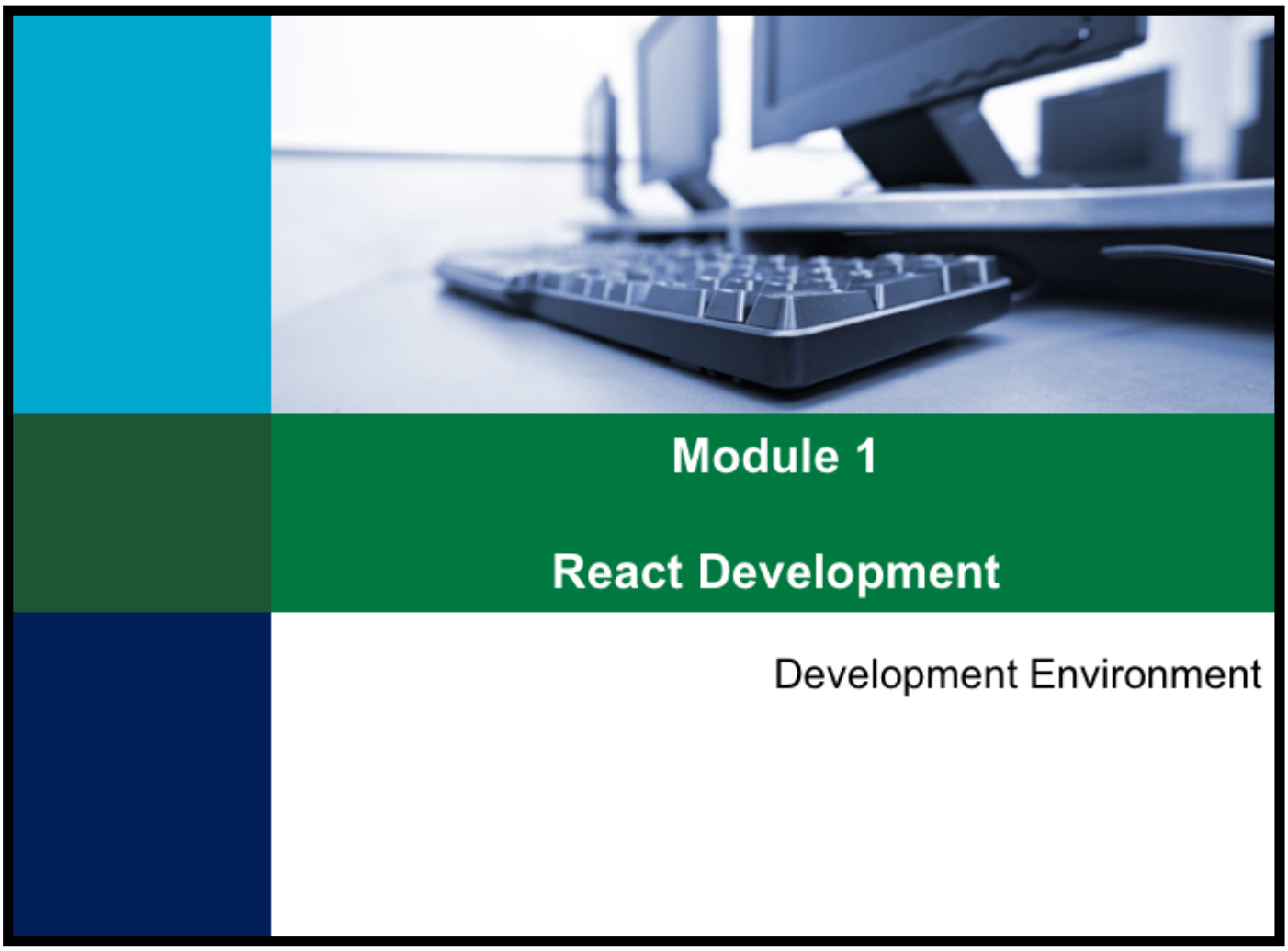
Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.


This page is intentionally blank
NT40801 20180421
Copyright © 2014-2018 nTier Training, LLC. All rights reserved.
0-8

Notes



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Objectives

- Introduce React
- Understand the React development life-cycle
- Build a basic React application

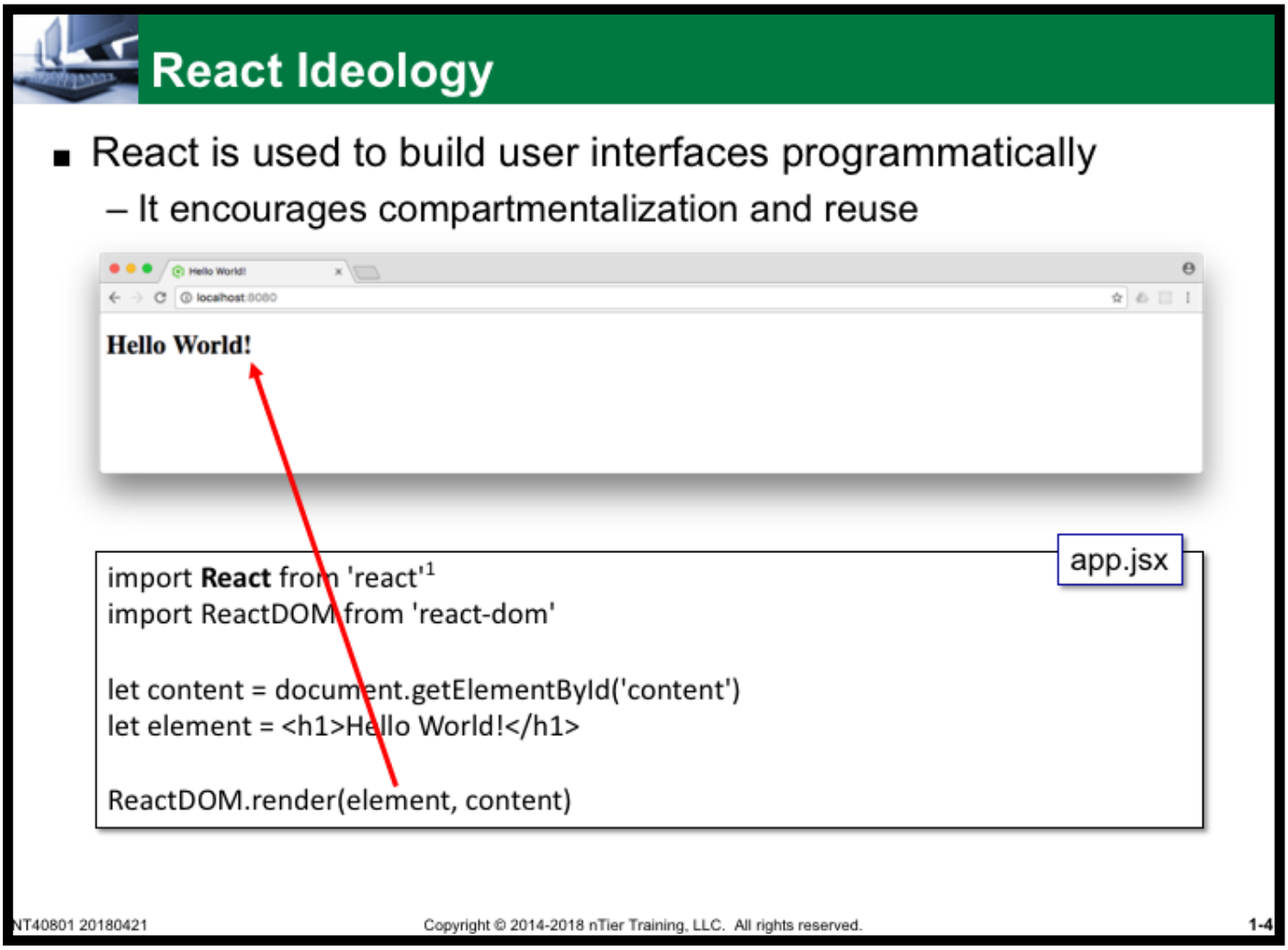
Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



[1] While most imports can rename the default import or any other import, the import of React cannot because JSX generates code with React as the module reference.

Notes

- React is intended to avoid the separation of templates and code that controls them.
- JSX allows the embedding of HTML-like XML tags in the JavaScript file and eliminates the need for quoted strings of HTML.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Project Structure

- There is no fixed structure for React projects
  - One choice is to organize the code by nature (module type)
  - A "domain" structure scales better (aka "facet" or "feature")

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Project Files

- As a class create and build a React project
  - Start in the folder **Student_Files/React/Labs/01_Fundamentals**[1]
- Work in the "src" folder for the project code
  - Add the app.jsx file:

app/app.jsx

```
import React from 'react'
import ReactDOM from 'react-dom'

let content = document.getElementById('content')
let element = <h1>Hello World!</h1>

ReactDOM.render(element, content)
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] This folder is part of the student files that need to be installed at the start of class. Your instructor will guide you through this.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Project Files

- The HTML that will be displayed must be injected somewhere
  - Use a "dist" folder to keep the deployment separate from src
  - Put an "index.html" file in the app folder to launch the application

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <div id="content"></div>
    <script type="text/javascript" src="assets/bundles/app-bundle.js"></script>
  </body>
</html>
```

- index.html will load and launch app-bundle.js

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

React Libraries

- React depends on two library packages: **react** & **react-dom**
- Packages are managed through **Node** and **npm**[1]
  - Create a basic **package.json** file in the project folder

```
{
   "name": "kaleidoscope",
   "version": "1.0.0",
   "description": "Kaleidoscope Cruise Lines",
   "private": true
}
```

- Add the core packages for react[2]

```
$ npm install react@^16.2.0 react-dom@^16.2.0 --save
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 1-8

Notes

[1] Node and npm will be used only to retrieve the packages that React requires, and to launch the webpack tool.

[2] The labs require specific versions of packages in a known configuration that works.

JavaScript 2015

- React depends on ECMAScript 6
  - Also referred to as ES6, ES2015, or JavaScript 2015

- ES6 Features: [1]
  - **Variable Scope**
  - **Arrow Functions**
  - **Extended Parameter Handling**
  - **Template Literals**
  - Extended Literals
  - Enhanced Regular Expressions
  - Enhanced Object Properties
  - **Destructuring Assignment**
  - **Modules**
  - **Classes**
  - Symbol Type
  - Iterators
  - Generators
  - Map/Set and WeakMap/WeakSet
  - Typed Arrays
  - New Built-in Methods
  - **Promises**
  - Meta-programming
  - Internationalization and Localization

- Browsers do not have full support for ES6…

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 1-9

[1] This React course definitely depends on the highlighted ES6 features.

Notes

Cross-Compilation

- To cross-compile ECMAScript 6 into ECMAScript 5
  - Add the **Babel** transpiler and tools to the project environment:

```
$ npm install babel-core@^6.26.0 babel-loader@^7.1.2
babel-preset-env@^1.6.1 babel-preset-react@^6.24.1 --save-dev
```

  - --save and --save-dev are only used to manage the packages in packages.json; they have no role for building the application

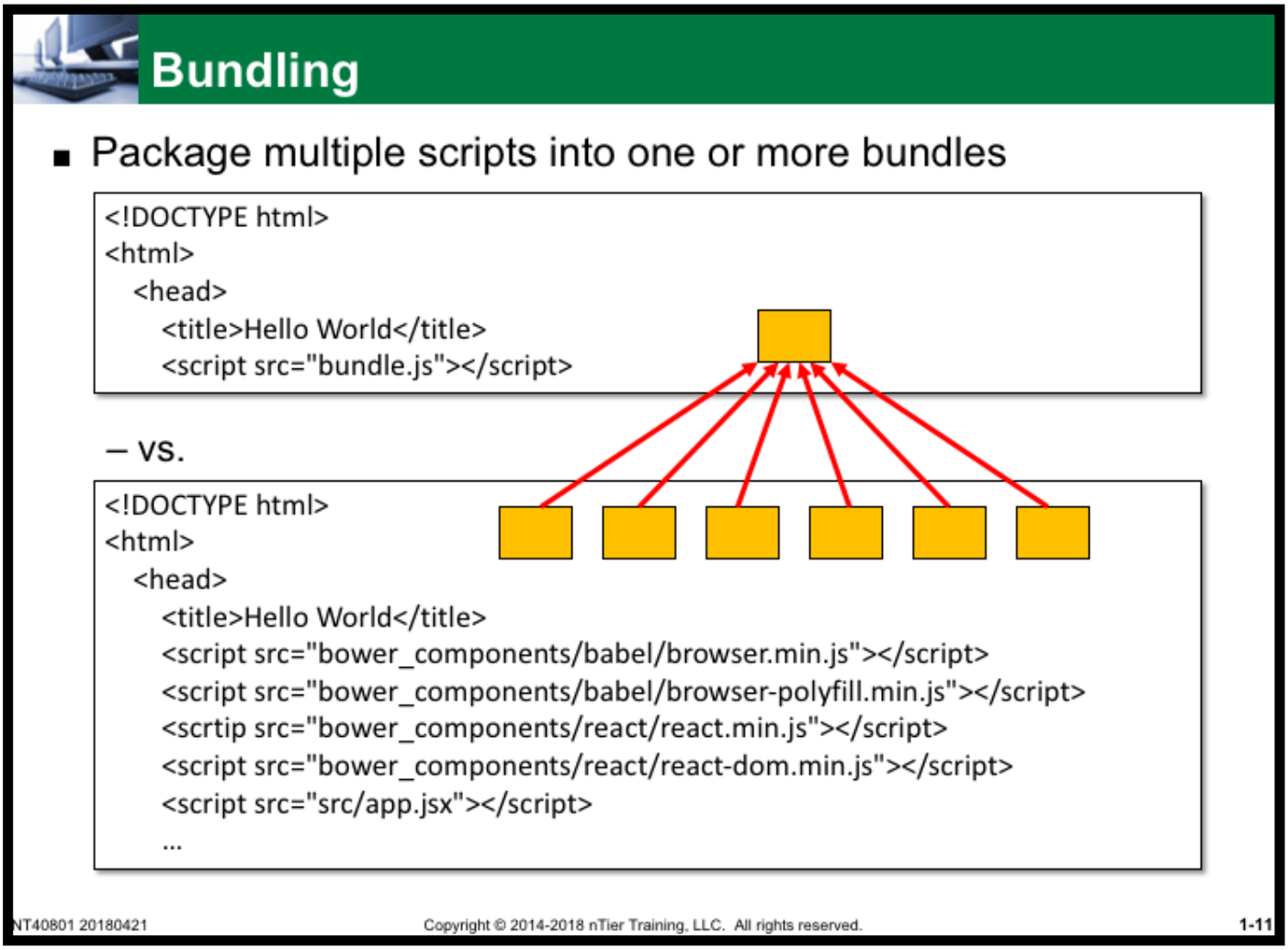
- More and more scripts to deliver for the application…
  - The react library scripts...
  - The scripts transpiled from the ES6 code...

- Also we need a way to manage the transpilation...

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

- Minifying scripts reduces the amount of data transferred in a script, bundling reduces the number of files. An alternative to bundling is dynamic script loading, where scripts are loaded when needed. The majority of the React community right now seems to prefer bundling, although that may change with HTTP 2 where bundling works against the protocol.

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

webpack

- **webpack** is a module bundler
  - JavaScript is bundled into a monolithic script
  - Supports **Babel** to cross-compile ES6 to ES5[1]
  - Includes all dependencies by following the ES6 module chain
  - Builds source maps for debugging[2]
- Add **webpack** to the project as a development tool:

```
$ npm install webpack@^3.10.0 --save-dev
```

- The **webpack** configuration is in **webpack.config.json**
  - Configure **webpack** to launch Babel for transpilation
  - Configure **webpack** to build the bundle

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 1-12

Notes

[1] Many browsers support ES6 directly now, but this course still uses cross-compilation to ES5 to make sure that everything works everywhere, and as a way to demonstrate it.

[2] Using a source map to support ES6 debugging is one of the big reasons that the React community prefers to use webpack to compile and bundle the source.

Configure webpack

- **webpack.config.js** goes in the project folder:
  - **webpack** will load and execute this script

webpack.config.js

```
var path = require('path')
var webpack = require('webpack')

var APP_DIR = path.resolve(__dirname, 'dist/')
var BUILD_DIR = path.resolve(__dirname, 'dist/assets/bundles/')
var BUNDLE_FILE = 'app-bundle.js'
var PUBLIC_PATH = 'assets/bundles/'
var SRC_DIR = path.resolve(__dirname, './')

process.noDeprecation = true[1,2]
module.exports = {
   context: SRC_DIR,
   entry: [
      './src/app.jsx'
   ],
```

The source is continued on the next page

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] This flag is set to true to remove deprecation warnings from the loader-utils arising from changes in Webpack 2 (https://github.com/webpack/loader-utils/issues/56).

[2] Do not put the footnote numbers into the file.

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Configure webpack

webpack.config.js

```
output: {
      path: BUILD_DIR,
      filename: BUNDLE_FILE,
      publicPath: PUBLIC_PATH
   },
   devtool: 'source-map',
   resolve: {
      extensions: ['.js', '.jsx'],
      modules: ['node_modules', 'src']
   },
```

The source is continued on the next page

- Compilation crawls the source code starting with src/app.jsx
  - **resolve** defines where to look for imported modules

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Configure webpack

webpack.config.js

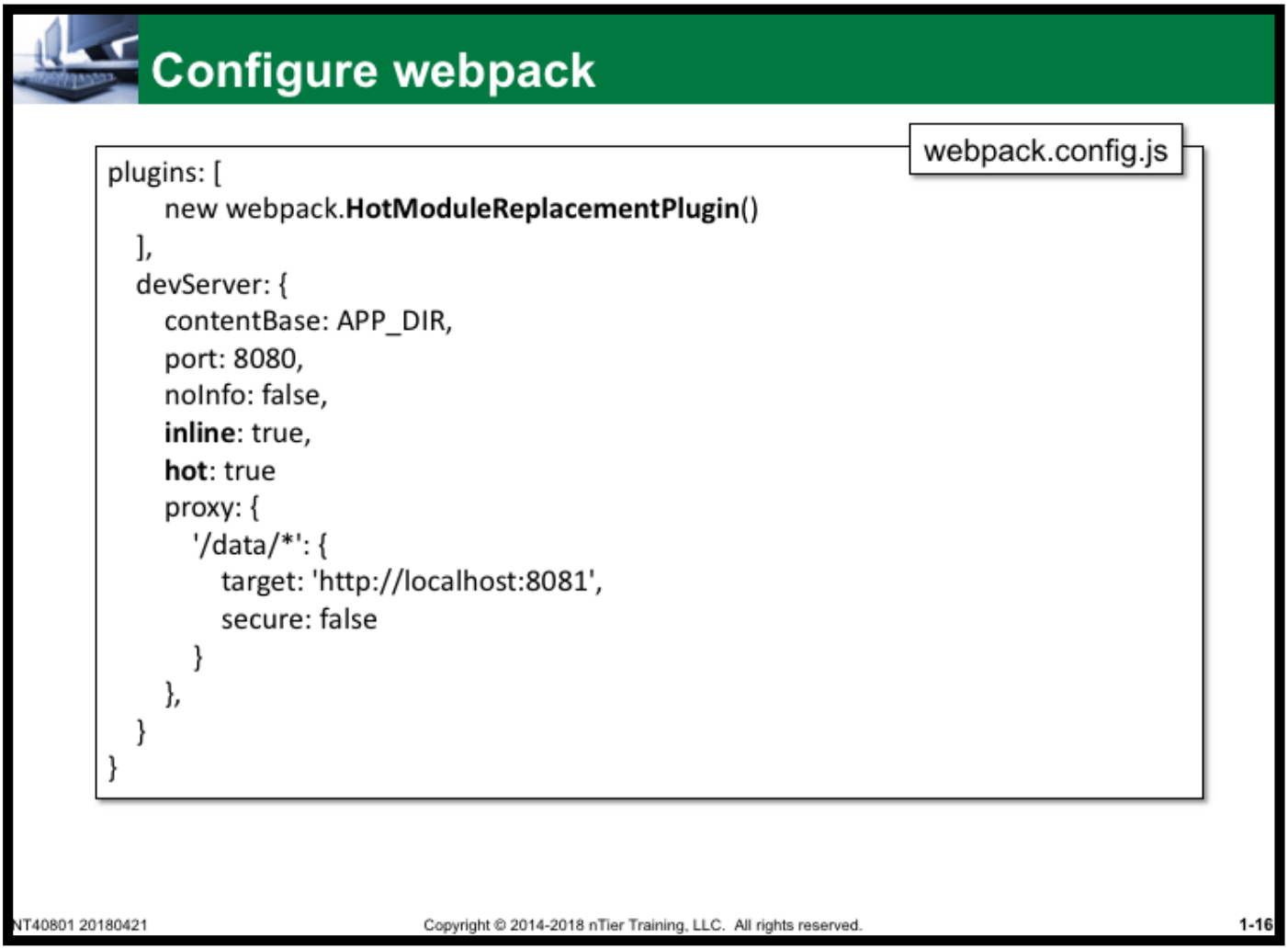
```
module: {
   loaders: [{
      test: /\.jsx?$/,
      loader: 'babel-loader',
      query: {
         presets: [ 'env', 'react' ],
         compact: false[1]
      }
   }]
}
}
```

- **loaders** defines how to handle different file types (use Babel)

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 1-15

[1] compact is set to false because the React core library is too big for the babel-loader to compact.

Notes



[1] compact is set to false because the React core library is too big for the babel-loader to compact.

Notes

Development Web Server

- webpack provides **webpack-dev-server** for testing[1]
  - Automatically re-bundles the JavaScript on changes
  - Supports module hot-replacement
  - Use **npm install webpack-dev-server@^2.9.5 --save-dev**
- The development server only serves the React program
  - The proxy is used to forward any URL beginning with /data to a web service running at port 8081 that serves the database
- The server is configured in webpack.config.js

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] Browsers can load a script bundle and other files directly from the file system but there can be complications. It is better to develop the application using a web server so the browser can go back to it to load additional resources.

[2] *noInfo* displays only errors and warning, *inline* adds the webpack-dev-server client entry point to the webpack bundle, and *hot* turns on hot-reloading of the bundle in the browser; the webpack-dev-server client polls the server and reloads the modules. The HotModuleReplacePlugin needs to be define for hot to work.

- The webpack-dev-server cannot run server-side programs, but it can be linked to Express. Express is the de facto standard for building web applications and services in the Node environment.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Launching webpack or webpack-dev-server

- webpack is a Node program
  - Add a script definition in package.json to launch it

```
"scripts": {
  "build": "./node_modules/.bin/webpack"
},
```

- Use **npm run [command]** with "build" to launch webpack

```
$ npm run build
```

- Wait for webpack to build the application
  - Check that the app-bundle.js file was built
  - Then open the index.html file in a browser

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Launching webpack or webpack-dev-server

- Add a script definition in package.json for the server

```
"scripts": {
  "dev": "./node_modules/.bin/webpack-dev-server",
  "build": "./node_modules/.bin/webpack"
},
```

- Use **npm run dev** to launch the web server

```
$ npm run dev
```

- Wait for webpack to build the application
  - Then browse to http://localhost:8080 to load and run it
- The server caches the bundle in memory
  - It does not build or update the app-bundle.js file

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Hot Module Replacement

- webpack will watch the source files
  - If a file changes the bundle will be recompiled
  - If HMR is enabled, the browser receives new module versions
- React and HMR issues
  - Modules import other modules, a lot can get reloaded
  - Reloading a module may conflict with the current state (data)
  - HMR extension code pollutes the application code[1]
  - If it cannot replace a module, React *will* reload the application
- Change the text in app.jsx to "Hello World from React!"
  - Save the file
  - Verify webpack recompiled the application and it reloaded[2]

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 1-20

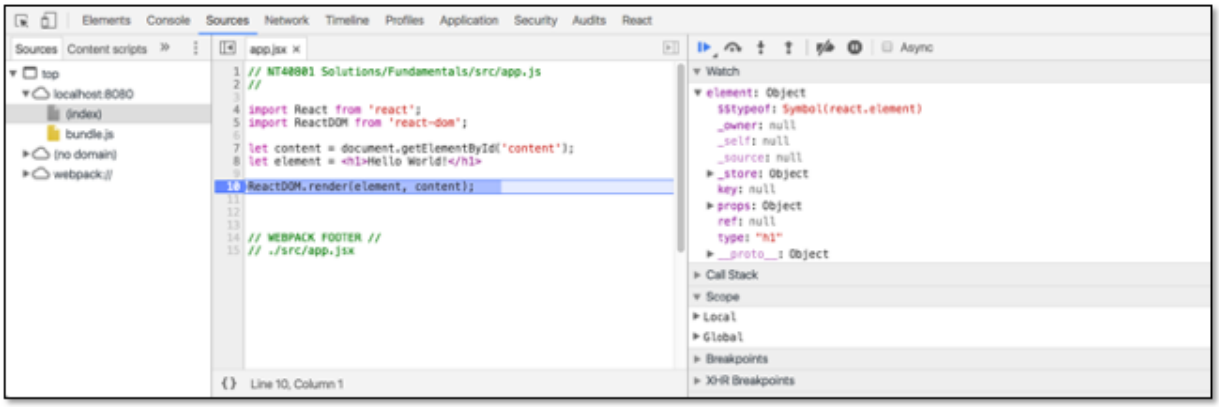
Notes

[1] Adding the HMR code into a React application is outside the scope of this course.

[2] Changing the app.jsx file pretty much guarantees that the application will be reloaded in the browser. To see if it did make the browser preserve the console across reloads ("preserve log" in the Chrome console display) and you will see the error messages if it needs to do a full reload of the page.

Debugging

- The bundle contains the ES5 transpiled code
  - **Source maps** map ES5 code to the original ES6 code
  - Debugging can take place in the ES6 code:



- Configure webpack.config.js to build **source maps**
  - **devtool** is a top-level field in the module.exports object:

```
devtool: 'source-map',[1]
```

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] The normal value for devtool would be "source-map", which builds and loads .map files. Warning: in webpack v1, the .map files for React are created too large for Chrome to handle so "inlining" the source maps with 'inline-source-map' fixes that.

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Application Launch

- index.html loaded the bundle at the bottom of the page
  - So the page is ready when the script loads and executes
  - webpack configured app-bundle.js to start executing in app.jsx

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
  </head>
  <body>
    <div id="content"></div>
    <script type="text/javascript" src="assets/bundle/app-bundle.js"></script>
  </body>
</html>
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

A Clean Application Launch

- Loading the script at the head of the page is cleaner

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World</title>
    <script type="text/javascript" src="app-bundle.js"></script>
  </head>
  <body>
    <div id="content"></div>
  </body>
</html>
```

- app.jsx must delay starting until the HTML page is loaded

app/app.jsx

```
window.addEventListener('load', () => {
  ReactDOM.render(<h1>Hello World!</h1>, document.getElementById('content'))
})
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 1-23

Notes

Checkpoint

- What is the basic premise of React?
- Why does the react community use webpack?
- Can the browser see changes to the application during development?
- Why is npm used during application development?
- What purpose does the Babel tool serve?
- How does webpack handle ES6? How can it be debugged?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Module 2

Component Development

React Elements
Custom Components
Props

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Objectives

- Create user interfaces using components
- Explore reusable components, inheritance, and composition
- Use JSX and expressions to embed component tags in JavaScript

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Component Development

React Elements

React Elements
Custom Components
Props

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

React DOM Elements and Props

- **React DOM Elements** look like embedded HTML tags
  - **Props** are properties on **React elements,** like attributes
  - Props are equivalent to the attributes of HTML elements

```
let element = <h1 id="title" className="title-bold">Hello World!</h1>
```

- React elements use DOM property names for **props**
  - The DOM property className is the HTML attribute class

- **Props** are strings or JavaScript values contained in braces
  - The **style** prop requires an object[1]
  - The fields of the object are the css properties

```
let element = <h1 id="title" style={{ color: 'Blue' }}>Hello World!</h1>
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] The outer braces define a JSX expression, the inner braces define a JavaScript literal.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Creating React DOM Elements

- JSX XML tags are a shorthand for calling **createElement**
  - Mostly we prefer to use JSX

```
// let element = React.createElement('h1', null, 'Hello World!')
let element = <h1>Hello World!</h1>

// let element = React.createElement('h1', { id: 'title' }, 'Hello World!')[1]
let element = <h1 id="title">Hello World!</h1>
```

- Elements may also be created from a **factory**
  - Using a factory is deprecated in favor of JSX

```
import React from 'react'

let h1 = React.createFactory('h1')

let element = h1({ id: 'title' }, 'Hello World!')
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] The second parameter to createElement are the *properties* or *props* of the element.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

JSX JavaScript Expressions

- JSX Expressions in braces may appear in child content

```
<span>
   Order total: { orderTotal }
</span>
```

- Put comments inside of expressions

```
<span>
   { // This expression will be ignored }
</span>
```

- Best-practice is to do calculations outside of expressions
  - Separate logic from presentation

```
let orderTotal = 0
this.props.orderItems.each( (item) => orderTotal += (item.count * item.amount) )
return (<span>Order total: { orderTotal }</span>)
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Element and Props Names

- Any name can be used with JSX or createElement
  - Interfaces other than the browser DOM may be supported
  - If the element used is not known the results will be unexpected
- Props of any name may be used
  - The name does have to follow the JavaScript identifier rules
  - Only names that are expected will be used
- The rules allow for **custom components** with **custom props**

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Component Development

Custom Components

React Elements
Custom Components
Props

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Custom Components

- React's strength is programmatically creating a user interface
  - The code to do this is managed in **custom components**
  - Programmatically creating the UI allows for precise control
  - Add or remove elements and content programmatically
- React favors designing simple components
  - They may be nested
  - They may be combined to create complex user interfaces
- React assumes a hierarchical interface built of components
  - It parallels the model the browser DOM has with HTML elements
  - It allows generic interface components to be reused

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Custom Components

- A **functional component** returns a tree of elements
  - There must be a root element
  - There may or may not be nested child elements

```
import React from 'react'

const HelloWorld = (props) => <h1>Hello World!</h1>

export default HelloWorld
```

- The function can be used with JSX, createElement, etc.
  - Or it may be called directly

```
import HelloWorld from 'HelloWorld'

ReactDOM.render(<HelloWorld />, document.getElementById('content'))
// ReactDOM.render(HelloWorld(), document.getElementById('content'))
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

createClass

- Traditional React passes **createClass** a literal definition
  - This builds a JavaScript 5 "class:" a constructor and a prototype
  - **render** is added as a method in the prototype
  - The render method returns the content
  - The prototype has all the core functionality for the component

```
import React from 'react'

const HelloWorld = React.createClass({

   render: function() {
      return <h1>Hello World!</h1>
   }
})

export default HelloWorld
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Extending React.Component

- Since ES6 React components are defined as classes
  - They inherit core functionality from the React.Component class
  - The **render** method is used to define the content
  - Rendering **null** adds nothing to the browser DOM

```
import React from 'react'

class HelloWorld extends React.Component {

   render() {
      return <h1>Hello World!</h1>
   }
}

export default HelloWorld
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Defining Component Classes

- A best-practice is to define one class per module
  - It makes it easier to find the class definition
  - Use the **default** export to make the class the module reference

components/HelloWorld.js

```
import React from 'react'

export default class HelloWorld extends React.Component {

   render() {
      return <h1>Hello World!</h1>
   }
}
```

- Modules simplify managing React components
  - Import only the required dependencies in each module

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-13

Notes

- Many sources will show the class definition with an anonymous class: export default class { … }. There is no harm in naming the class, the name is local to the module, and it makes it easy to remember what the class is when you are looking at the code.

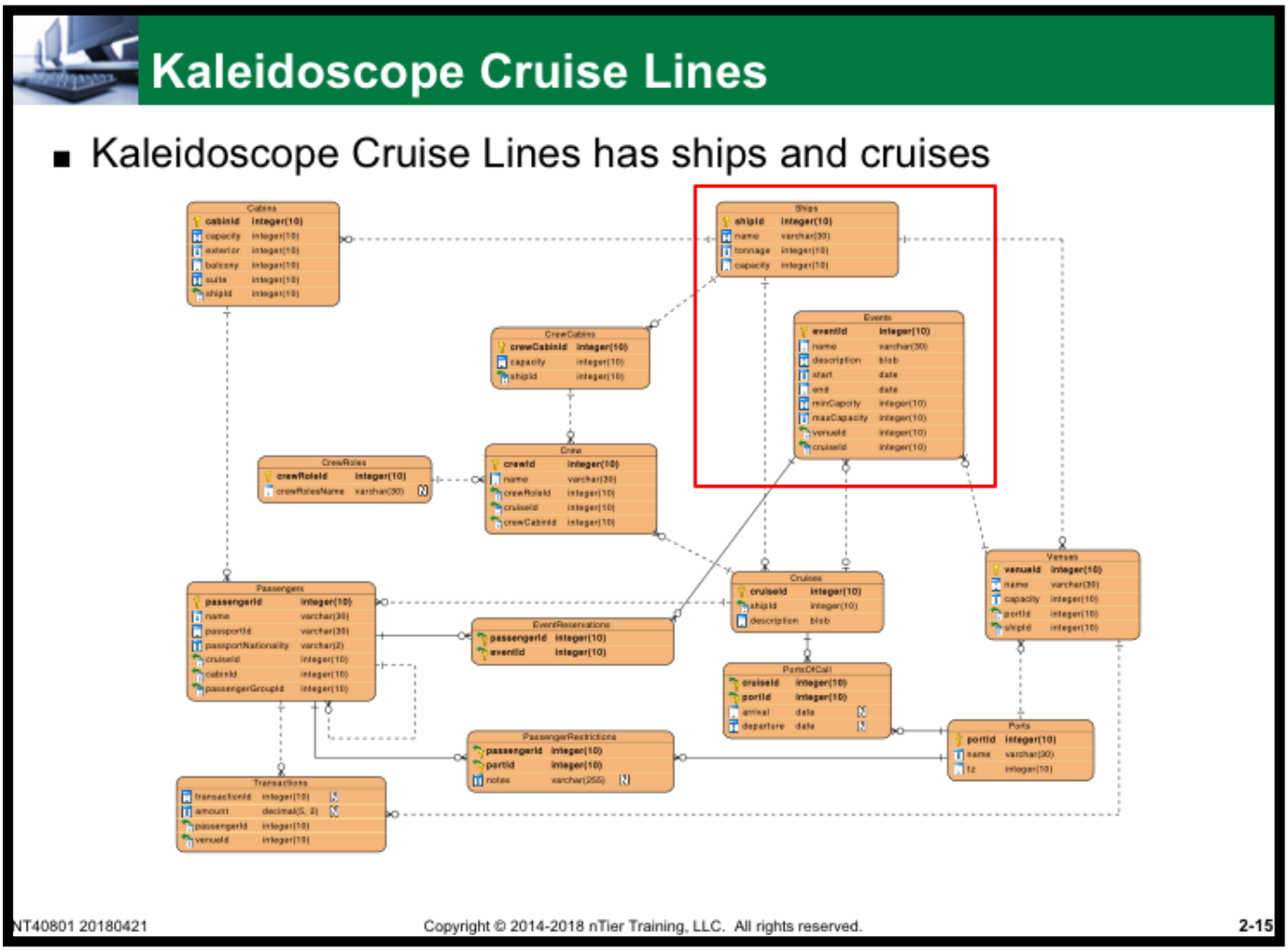
Component Development

Props

React Elements
Custom Components
Props

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



- Kaleidoscope Cruise Lines does just about what you expect, take passengers on cruise ships. This is a vision of the overall data model necessary to support the company operations. The course is only interested in Ships and Events, and will not model all the fields of the tables.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

KCL Event Management

- Our requirement is an interface to manage events
  - Events belong to a cruise, and a cruise has ships
  - To keep it simple all events belong to one cruise for now
  - Events take place at a venue
- Ultimately we need a way to manage the events for cruises, and the venues that the events will take place at
- Can we design components that lend themselves to reuse?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Components

- Our first goal is to provide a list of ships
  - Start with a simple, dedicated list of names

```
import React, { Component } from 'react'

export default class ShipList extends Component {
   render() {
      return (
         <div> <h2>Ships</h2>
         <ul>
            <li>Kaleidoscope Voyager, 138194 gt</li>
            <li>Kaleidoscope Freedom, 154407 gt</li>
         </ul> </div>
      )
   }
}
```

Ships

- Kaleidoscope Voyager, 138194 gt
- Kaleidoscope Freedom, 154407 gt

- No separation of concerns; a one-component application

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-17

Notes

- Note: watch out for a return with the value on the next line; the JSX translator will get confused even though it is technically correct. Fix that problem by surrounding the XML with parenthesis as in this example.

Nested Components

- Separate out the items and use them in a parent component
  - Pass data to the children with custom **props**

```
import ListItem from 'common/ListItem'
export default class ShipList extends Component {
   render() {
      return (
         <div>
            <h2>Ships</h2>
            <ul>
               <ListItem gt="138194">Kaleidoscope Voyager</ListItem>
               <ListItem gt="154407">Kaleidoscope Freedom</ListItem>
            </ul>
         </div>)
   }
}
```

- Information flows only downhill through the tree
  - The ListItem instances should not care about their parent

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-18

Notes

- The import for React, { Component } is *assumed* to save space in the example.
- Changing how the list item is managed and displayed is now separate from the list itself.

Flow of information

- **Props** flow from parent to child
  - In the ListItem component reference **props** from **this**
  - The special prop **children** contains the component content

```
export default class ShipList extends Component {
   render() {
      return (
         ...
            <ListItem gt="138194">Kaleidoscope Voyager</ListItem>
            <ListItem gt="154407">Kaleidoscope Freedom</ListItem>
         ... )
   }
}
```

```
export default class ListItem extends Component {
   render() {
      return  <li>{ this.props.children }, { this.props.gt } gt</li>
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Injecting Data through Props

- The data is statically defined in the ShipList component
  - Why not dynamically inject it through **props** as an array?
  - Add the title as an attribute too
  - Move the data to the app.jsx file and inject it

```
import ShipList from 'ship/ShipList'

let ships = [
   { name: "Kaleidoscope Voyager", gt: 138194 },
   { name: "Kaleidoscope Freedom", gt: 154407 }
]

let shipList = <ShipList title="ships" ships={ ships } />

ReactDOM.render(shipList, document.getElementById('content'))
```

- Of course *ShipList* must expect and use the data...

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Injecting Data through Props

- Process the array elements into a list of components
  - Now the list of ships may be changed without changing ShipList

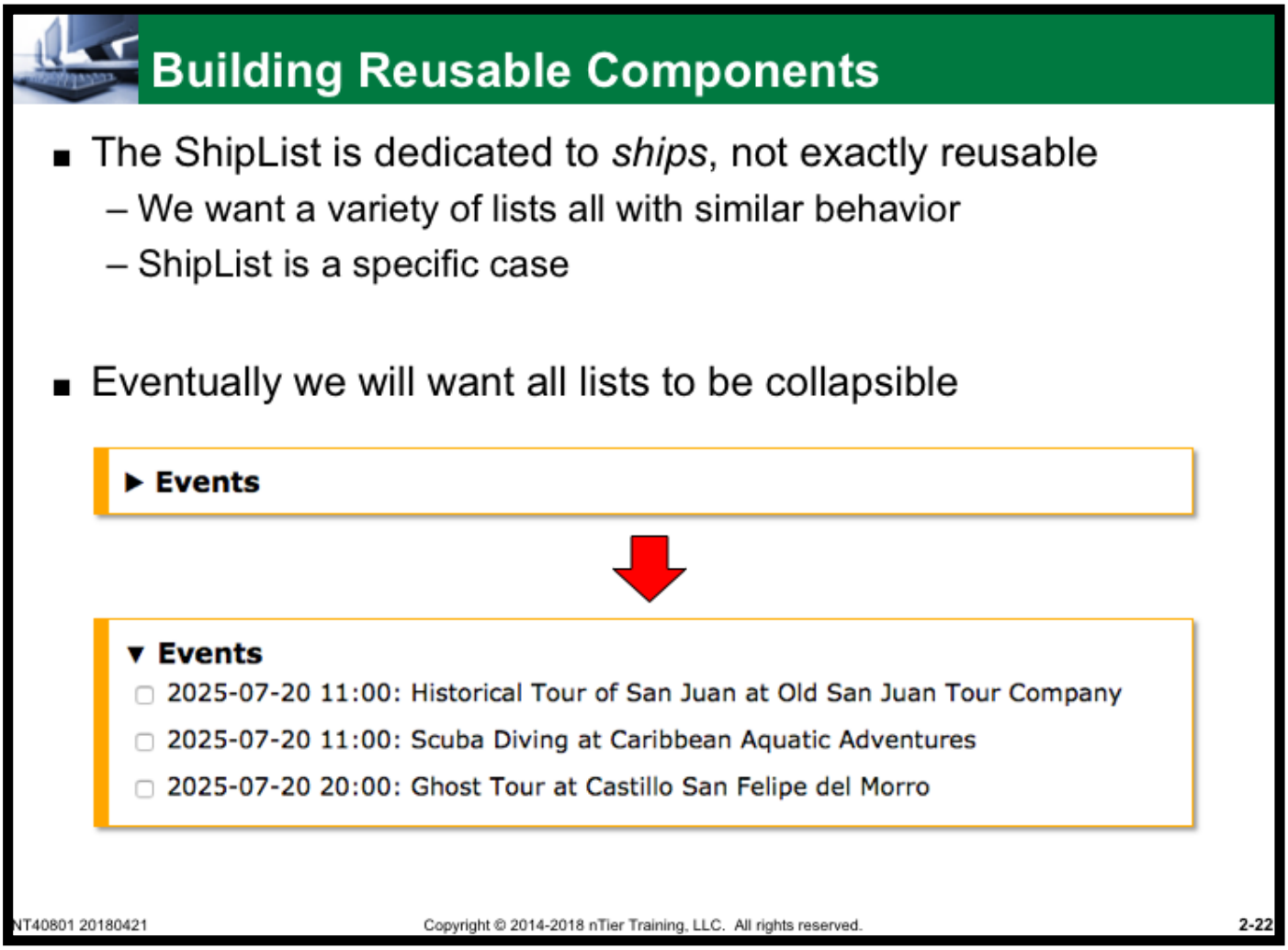
```
export default class ShipList extends Component {
   render() {
      let ships = this.props.ships.map( (ship) =>
         <ListItem gt={ ship.gt }>{ ship.name }</ListItem> )

      return <div><h2>{ this.props.title }</h2><ul>{ ships }</ul></div>
   }
}
```

- Officially props are immutable
  - They are owned by the "parent"
  - They cannot be changed by the component
  - *But realistically in JavaScript, that only works if we respect it*

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-21

Notes

- Using advanced JavaScript object properties this.props and this.props.property, where property was built when the component was created, are protected from change. This took place in the code behind the React.createClass method. So you cannot add new properties, and you cannot change property values. But… if the property is an object then the object itself can be changed, that is what we have to respect.



Notes

[1] There are some nasty surprises in store when components extend each other. The only benefit would be inheriting some functionality, such as a helper method. In OOP we should favor composition over inheritance anyways.

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Inheritance

- There are two important reasons ShipList cannot extend a List
  - It is extremely difficult to inject into what the super class renders!
  - State becomes a problem with inheritance: who's State is it?

```
import ListItem from 'common/ListItem'
export default class List extends Component {
   render() {
      let items = this.props.children.map( (item) =>
         <ListItem>{ item }</ListItem> )
      return ( <div><h2>{ this.props.title }</h2><ul>{ items }</ul></div> )
   }
}
```

```
export default class ShipList extends List {
   render() {
      // Even if you call the super-class render, how do you inject data into
      // each list item?
```

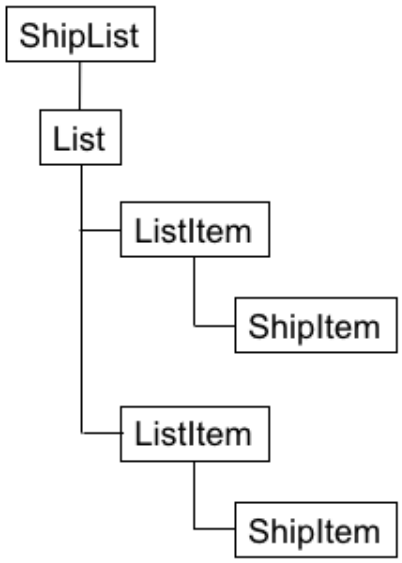
NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-23

Notes

[1] There are some nasty surprises in store when components extend each other. The only benefit would be inheriting some functionality, such as a helper method. In OOP we should favor composition over inheritance anyways.

Composition over Inheritance

- List and ListItem control the formatting of *lists* and *items*
  - ShipList and ShipItem format *lists of ships* and *ships*
  - *Lists* and *ListItems* provide generic functionality
  - So, ShipList *wraps* a List, and ListItem *wraps* a ShipItem



NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-24

Notes

- The ShipList wraps a List and controls how the list is presented.
- The ListItem wraps a ShipItem: the ListItem controls how the item is presented, and the ShipItem controls how the ship information is formatted.

ShipList

- A Ship wraps a list and feeds it a list of ShipItems
  - The ShipItem list is the content of the list, the *children*

```
import ShipItem from 'ship/ShipItem'

export default class ShipList extends Component {
   render() {
      let ships = this.props.ships.map( (ship) =>
         <ShipItem gt={ ship.gt }>{ ship.name }</ShipItem> )
      return <List title={ this.props.title }>{ ships }</List>
   }
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-25

Notes

[1] There are some nasty surprises in store when components extend each other. The only benefit would be inheriting some functionality, such as a helper method. In OOP we should favor composition over inheritance anyways.

A Reusable List

- A List provides generic list functionality
  - It renders a list of ListItems
  - The content of each ListItem is an instance of a child of the List

```
import ListItem from 'common/ListItem'
export default class List extends Component {
   render() {
      let items = this.props.children.map( (item) =>
         <ListItem>{ item }</ListItem> )
      return ( <div><h2>{ this.props.title }</h2><ul>{ items }</ul></div> )
   }
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-26

Notes

[1] There are some nasty surprises in store when components extend each other. The only benefit would be inheriting some functionality, such as a helper method. In OOP we should favor composition over inheritance anyways.

ListItem is Generic

- A ListItem may now be a generic, reusable component
  - All it has to do is render the components of an item and the contents

```
export default class ListItem extends Component {
   render() {
      return  ( <li>{ this.props.children }</li> )
   }
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 2-27

- The ListItem does not do much, but consider that the HTML elements selected, and any attributes they have, are now changeable in one place for any components that use this.

Notes

ShipItem Formats the Details

- ShipItem formats the details about a ship
  - Very similar to the original ListItem, but emits a *span*
  - This will be the contents of a ListItem

```
export default class ShipItem extends Component {
   render() {
      return  ( <span>{ this.props.children }, { this.props.gt } gt</span> )
   }
}
```

- ShipItem does not extend ListItem, ListItem wraps ShipItem
  - Inheritance does not work well: often render must be replaced
  - State will be an issue
  - Favor composition over inheritance![1]

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] There are some nasty surprises in store when components extend each other. The only benefit would be inheriting some functionality, such as a helper method. In OOP we should favor composition over inheritance anyways.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Rendering ShipList

- In app.jsx, fix the rendering of ShipList

```
let shipList = <ShipList title='Ships' ships={ ships} />

ReactDOM.render(shipList, document.getElementById('content'))
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

ShipList Hierarchy

- This is the hierarchy of the elements created
  - The hierarchy is rendered from the React browser plugin

```
▼ <ShipList title="Ships" ships=[{…}, {…}]>
  ▼ <List>
    ▼ <div>
        <h2/>
      ▼ <ul>
        ▼ <ListItem>
          ▼ <li>
            ▼ <ShipItem gt=138194>
              ▼ <span>
                  "Kaleidoscope Voyager"
                  ", "
                  "138194"
                  " gt"
                </span>
              </ShipItem>
            </li>
          </ListItem>
        ▶ <ListItem>…</ListItem>
        </ul>
      </div>
    </List>
  </ShipList>
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

HTML and Markup Content

- HTML and/or Markup content could be passed in the data
  - A description field in an object?
  - Markup is a shorthand that is translated into HTML

- React will block HTML data from being written
  - This is to prevent cross-site scripting attacks (XSS)
  - Defeat this with the prop **dangerouslySetInnerHTML**

```
render() {

   let questionableData = props.htmlToInject

   return <span dangerouslySetInnerHTML={ questionableContent } />
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Checkpoint

- What are React Components?
- What are props used for?
- How do you declare props?
- What is the purpose of JSX?
- How are JavaScript expressions represented in JSX?
- How can you create reusable React components?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Module 3

State

Interface Components and State
Closures vs Methods and Special Cases
Toggle Components

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Objectives

- Expand on the HTML interface components
- Manage the state of a component
- Respond to component events and update state

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

State

Interface Components and State

Interface Components and State
Closures vs Methods and Special Cases
Toggle Components

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Input Fields

- Add input fields to a control
  - As embedded input field in complex control

```
export default class ShipForm extends Component {
   render() {
      return  (
         <form>
            <label>name: <input type='text' /></label><br />
            <label>tonnage: <input type='text' /></label>
         </form>
      )
   }
}
```

- Embedded input fields
  - Can share data through the enclosing component
  - Presentation can be standardized through CSS

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 3-4

Notes

Uncontrolled Components

- An **uncontrolled component** does not have a bound **value**
  - The value attribute in a checkbox or radio button is **checked**
  - An uncontrolled component manages its own state
  - **defaultValue** (or **defaultChecked**) defines the initial value
  - An event handler is the only way to find out what changes

```
export default class EventItem extends Component {
   constructor(...args) {
      super(...args)
      this.isChecked = false
   }
   render() {
      return  (<span><input type='checkbox' defaultChecked={ this.isChecked; }
         onChange={ (event) => { this.isChecked = event.target.checked; }} />
         ... )
   }
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 3-5

Notes

- The constructor is used to define a property on the component with a value, and defaultChecked is used to set the initial value of the checkbox from that value.
- ES6 syntax is used to combine the list of arguments to the constructor into "args" with the rest operator, and then the spread operator is used to expand that list when the constructor for the super-class is called.

Controlled Components

- A **controlled component** has a bound value
  - Bound through the **value** or **checked** property

```
export default class EventItem extends Component {
   constructor() {
      super(...arguments)
      this.isChecked = false
   }
   render() {
      return  (<span><input type='checkbox' checked={ this.isChecked }
         onChange={ (event) => { this.isChecked = event.target.checked }} />
         ... )
   }
}
```

- Unfortunately React will not allow the checkbox to change
  - Check it, React resets it, and calls handler with the reset value!

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

State

- The answer is a **state** property for the component
  - Component provides the **setState** method to merge new values
  - **setState** causes the component to be rendered

```
export default class EventItem extends Component {
   constructor() {
      super(...arguments)
      this.state = {
         isChecked: false
      }
   }
   render() {
      return  <span><input type='checkbox' checked={ this.state.isChecked }
         onChange={ (event) =>
            { this.setState({ isChecked: event.target.checked }) } } />
            ... )
   }
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 3-7

Notes

- Initialize state as an object with properties in the constructor.
- Never mutate state directly, the component will not be rendered. Then your changes could be overridden by something else calling setState.

Controlled vs. Uncontrolled Components

- React is all about the **state**
  - Components *observe* the state and change when it changes
  - Multiple components can share state in a complex component
- Controlled components are preferred
  - Multiple controlled components can be bound to one state
- Uncontrolled components will not see changes to state
  - The logic to link these components will be brittle
- State is the other reason components are never extended
  - The super-class and subclass share the same state object
  - The state in the subclass may conflict with the super-class

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Composition over Inheritance

- State is another reason component inheritance does not work
  - The sub-class must share the state with super-class
  - It can collide with state properties the super-class created

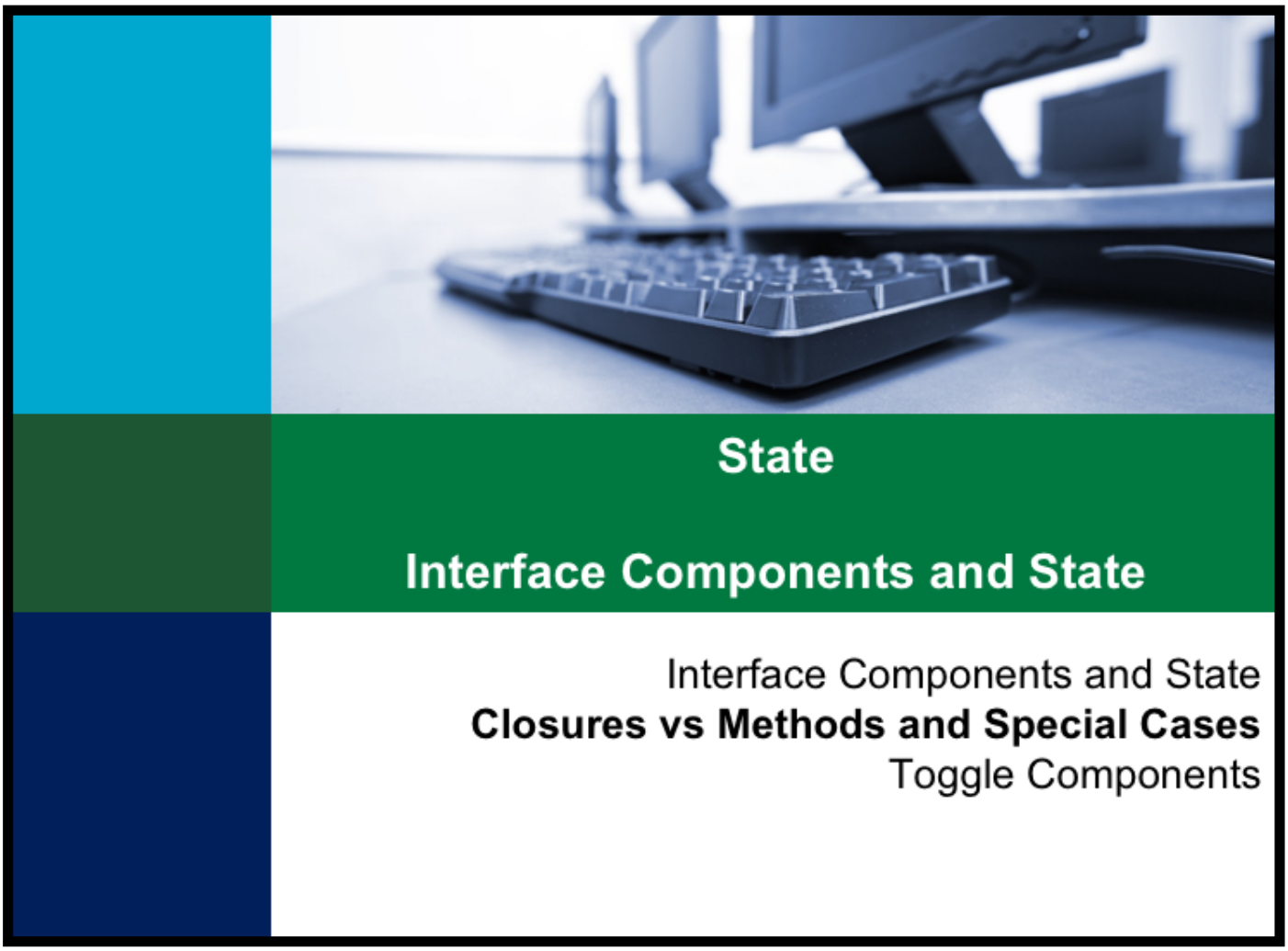
```
import ListItem from 'common/ListItem'
export default class List extends Component {
   constructor() {
      this.state = { myproperty: 'something' }
```

```
export default class ShipList extends List {
   constructor(...args) {
      super(...args)
      // this.state = { ... } // this is bad: the super-class state is destroyed!
      state.myproperty = 'something else' // also a problem, collides with super-class
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Callbacks and Arrow Functions

- Class methods were used through React version 0.12
  - *Autobinding* existed from versions 0.4 through 0.12[1]
  - Referencing a method automatically bound it to the instance

```
onChange={ this.handleChange } }
```

- Autobinding was removed in 0.13 (enter ES6)
  - Because ES6 arrow functions are automatically bound to *this*

```
onChange={ (event) => { this.isChecked = event.target.checked } }
```

- Arrow functions are closures
  - They are recreated each time the component is rendered!
  - While more popular, constantly creating closures is inefficient

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] Just in case you stumble across old code.

- Methods vs. arrow functions are a point of conflict. On one hand it is a really good best-practice to move the logic out of the JSX and just call the methods. On the other, this is exactly why arrow functions were put into ES6 and are bound to *this*.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Methods as Callbacks

- Class methods can still be used but must be bound

```
onChange={ handleChange.bind(this) } }
```

  - Binding creates a new function, and…
  - The function is created every time the component is rendered

- So, bind all the callback methods in the constructor
  - Each bound function is created once, the same methods are used every time the component is rendered
  - Different bound functions will be used in each class instance

```
constructor() {
   ....
   this.handleChange = this.handleChange.bind(this)
}

... onChange={ this.handleChange() } }
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] Just in case you stumble across old code.

- Methods vs. arrow functions are a point of conflict. On one hand it is a really good best-practice to move the logic out of the JSX and just call the methods. On the other, this is exactly why arrow functions were put into ES6 and are bound to *this*.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Input Field Special Cases

- <textarea value="" />
  - In HTML the content of the textarea is the data
  - In React, the **value** prop bind the component to the data

- In select lists each option has a value
  - The value of the select component identifies the selected option

```
this.state.selectedColor = 'red'
...
<select value={ this.state.selectedColor } >
   <option value='red'>The color red</option>
   <option value='green'>The color green</option>
   <option value='blue'>The color blue</option>
</select>
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Component Development

Toggle Components

Interface Components and State
Closures vs Methods and Special Cases
Toggle Components

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Improve the Presentation

- The <ul> bullet points are still in front of the checkboxes
  - Fix that in a site.css file linked into index.html
  - Add <ul **className**='classlist'> to apply it

```
.classlist { list-style-type: none; }
```

- What if the list was removed entirely?
  - How about a wrapping a series of for the elements
  - And change the title from an <h2> to a also

```
export default class List extends Component {
   render() {
      let items = this.props.children.map( (item) => <ListItem>{ item }</ListItem> )
      return ( <div className='list'><div className='list-title'>
         { this.props.title }</div>{ items }</div> )
   }
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 3-15

Notes

- The only change in List was to remove the <ul> tags and add a class to the enclosing div.

Improve the Presentation

- The ListItems were
  - Change them to

```
export default class ListItem extends Component {
   render() {
      return  ( <div className='list-item'>{ this.props.children }</div> )
   }
}
```

- Add some better CSS to create a slick look
  - Set the font for the page

```
html, body {
   font-family: verdana, arial, helvetica, sans-serif;
   font-size: 12pt;
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Improve the Presentation

- The list div with a fat border on the left[1]

```
div.list {
   width: 700px;
   margin: 5px;
   border: 1px solid orange;
   border-left: 10px solid orange;
}
```

- The title uses three styles
  - The second two add an arrow in front

```
div.list-title {
   font-size: 14pt;
   font-weight: bold;
}
```

The source is continued on the next page

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] We could get even more creative and override the border color defined in CSS through a color prop on the List component that ends up as a style on the HTML.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Improve the Presentation

```
div.list-title:before {
   font-size: 12pt;
   width: 1em;
   content: '\25bc';
}

div.list-title-closed:before {
   font-size: 12pt;
   width: 1em;
   content: '\25b6';
}
```

- And finally some space around the items

```
div.list-item {
   padding: 5px;
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Improve the Presentation

- This all provides a slick look to the list of events:

▼ **Events**

- ☐ 2025-07-20 11:00: Historical Tour of San Juan at Old San Juan Tour Company
- ☐ 2025-07-20 11:00: Scuba Diving at Caribbean Aquatic Adventures
- ☐ 2025-07-20 20:00: Ghost Tour at Castillo San Felipe del Morro

- But a new requirement was just made:
  - Clicking on the title of the list should make the items appear and disappear; initially they should be hidden.
  - This can also be handled using state…

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Toggle

- Toggling the visibility of the list is handled in the rendering
  - The only state needed is a Boolean flag to control it
  - The flag will be set by an event handler on the title div

```
export default class List extends Component {
   constructor(...args) {
      super(...args)
      this.state = { displayList: false }
   }
   render() {
      let items = null
      if (this.state.displayList) { items = this.props.children.map( (item) =>
         <ListItem>{ item }</ListItem> ) }
      return ( <div className='list'}><div className='list-title'
         onClick={ (event) =>  this.setState({ displayList: !this.state.displayList }) } >
         { this.props.title }</div>{ items }</div> )
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Toggle

- The CSS contains a style for the closed list arrow
  - Toggle the div by changing the class name

```
export default class List extends Component {
   constructor(...args) {
      super(...args)
      this.state = { displayList: false }
   }
   render() {
      let items = null
      if (this.state.displayList) { items = this.props.children.map( (item) =>
         <ListItem>{ item }</ListItem> ) }
      return ( <div className='list'}>
         <div className={ this.state.displayList ? 'list-title' : 'list-title-closed' }
         onClick={ (event) =>  this.setState({ displayList: !this.state.displayList }) } >
         { this.props.title }</div>{ items }</div> )    }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Toggle

- Clicking on the title will show or hide the list

▶ **Events**

▼ **Events**

- ☐ 2025-07-20 11:00: Historical Tour of San Juan at Old San Juan Tour Company
- ☐ 2025-07-20 11:00: Scuba Diving at Caribbean Aquatic Adventures
- ☐ 2025-07-20 20:00: Ghost Tour at Castillo San Felipe del Morro

- One problem for the future: the state of checkboxes is not persistent across the toggling of the list!
  - It happens because the event items are re-instantiated

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Checkpoint

- Why are there uncontrolled and controlled components?
- Why do callback functions need to be bound to *this?*
- Why was the navigation state saved in Main and changed with a callback?
- How was state used to toggle the list view?
- What is the philosophy of DOM management in React?
- If they are kept in the state of an EventItem, why are the checkboxes not preserved when the list collapses?

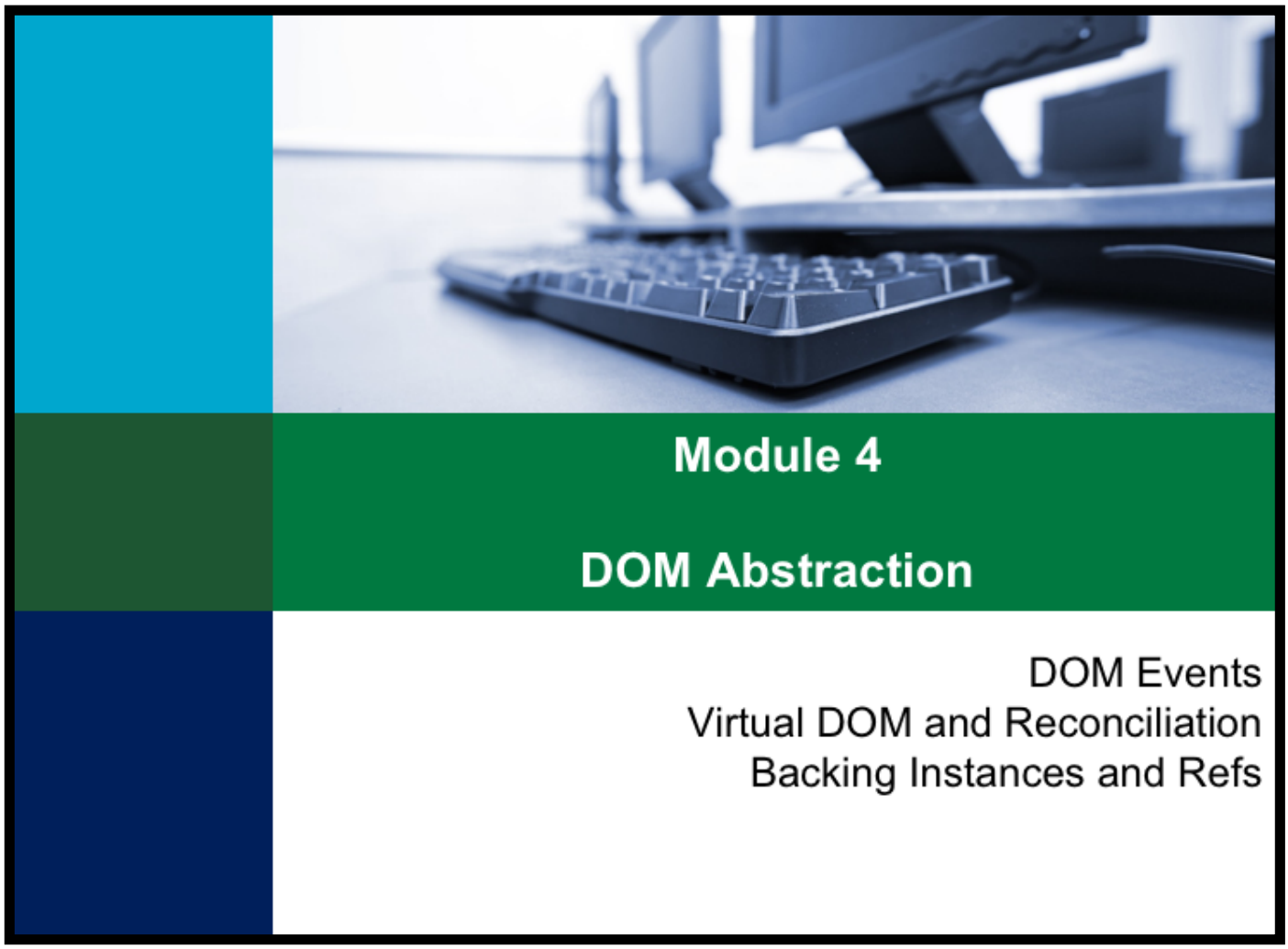
Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.


This page is intentionally blank
NT40801 20180421
Copyright © 2014-2018 nTier Training, LLC. All rights reserved.
3-24

Notes



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

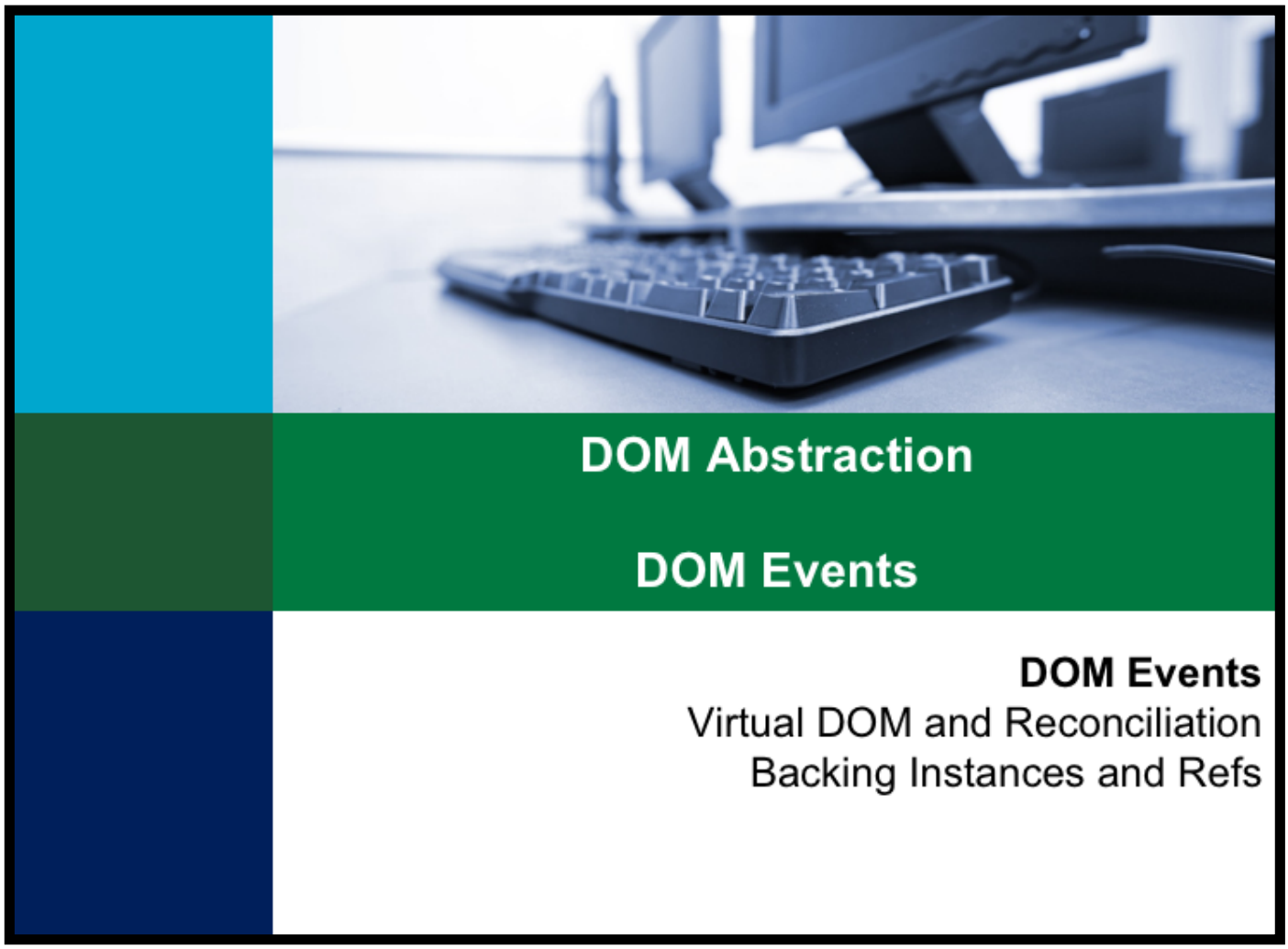
Objectives

- Understand the virtual DOM and reconciliation
- Help reconciliation with key values
- Step outside of the virtual DOM with refs

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Normalized DOM Events

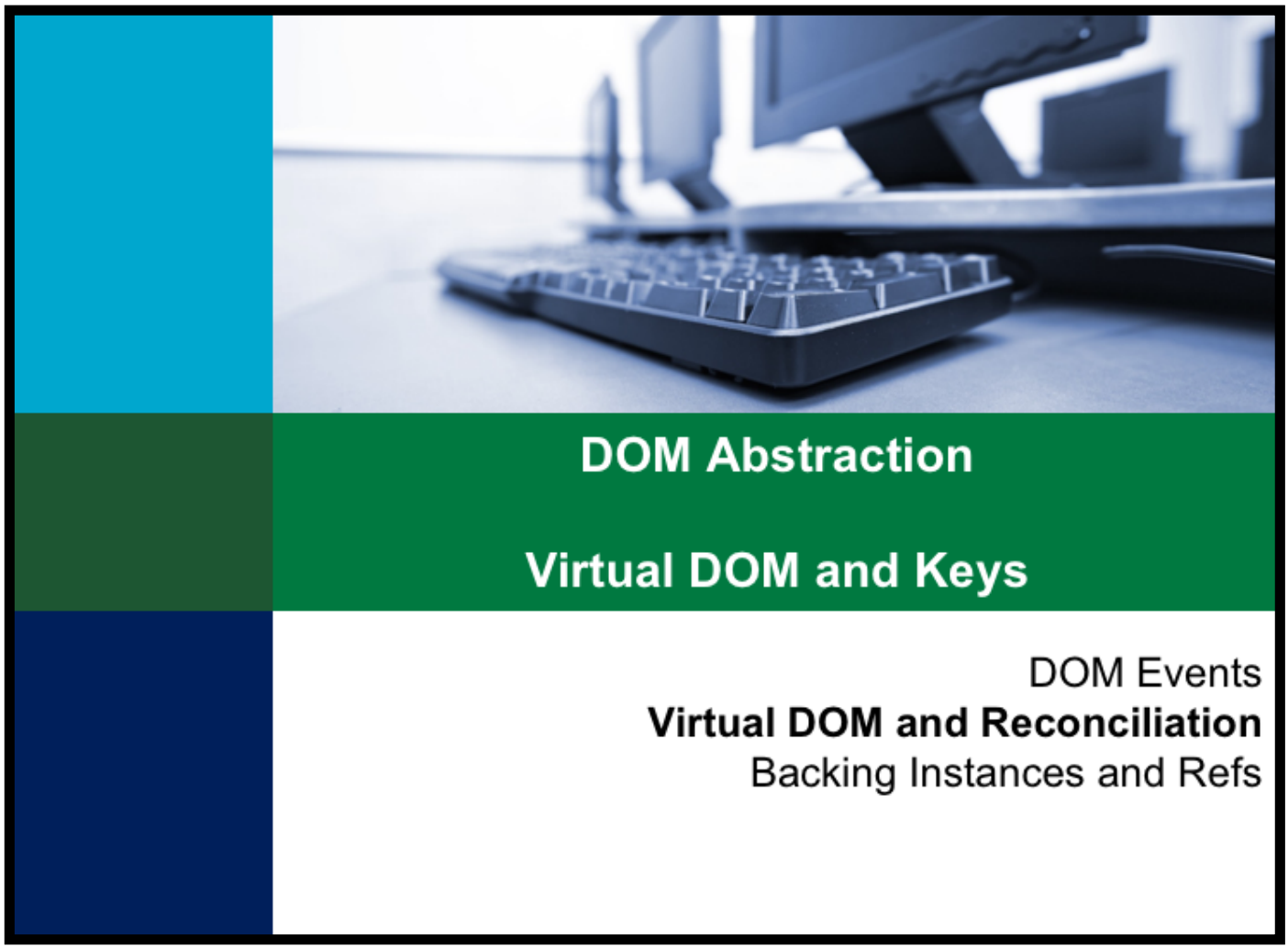
- Touch and Mouse Events
  - onClick, onContextMenu, onDoubleClick, onDrag, onDragEnd, onDragEnter, onDragExit, onDragLeave, onDragOver, onDragStart, onDrop, onMouseDOwn, onMouseEnter, onMouseLeave, onMouseOut, onMouseOver, onMouseUp, onTouchCancel, onTouchEnd, onTouchMove, onTouchStart
- Keyboard Events
  - onKeyDown, onKeyPress, onKeyUp
- Focus and Form Events
  - onBlur, onChange, onFocus, onInput, onSubmit
- Other Events
  - onCopy, onCut, onPaste, onScroll, onWheel

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

- "Normalized" means that these are the events React elements see, regardless of what the browser actually generates.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

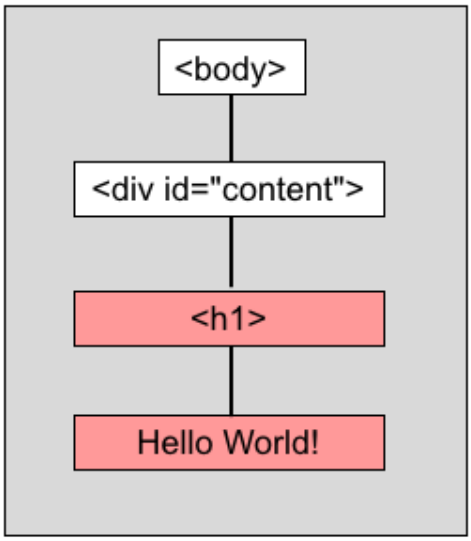


Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

HTML Elements and React Elements

- HTML tags are made into objects
  - Added to a document-object-model
- React separates itself from the DOM
  - DOM objects are heavyweight
  - DOM objects are slow to manipulate
- React.createElement
  - Produces lightweight React Elements
  - Plain old JavaScript objects
  - Describe a component and data
  - JSX translates to calling createElement
  - React DOM Elements are leaf nodes



NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 4-6

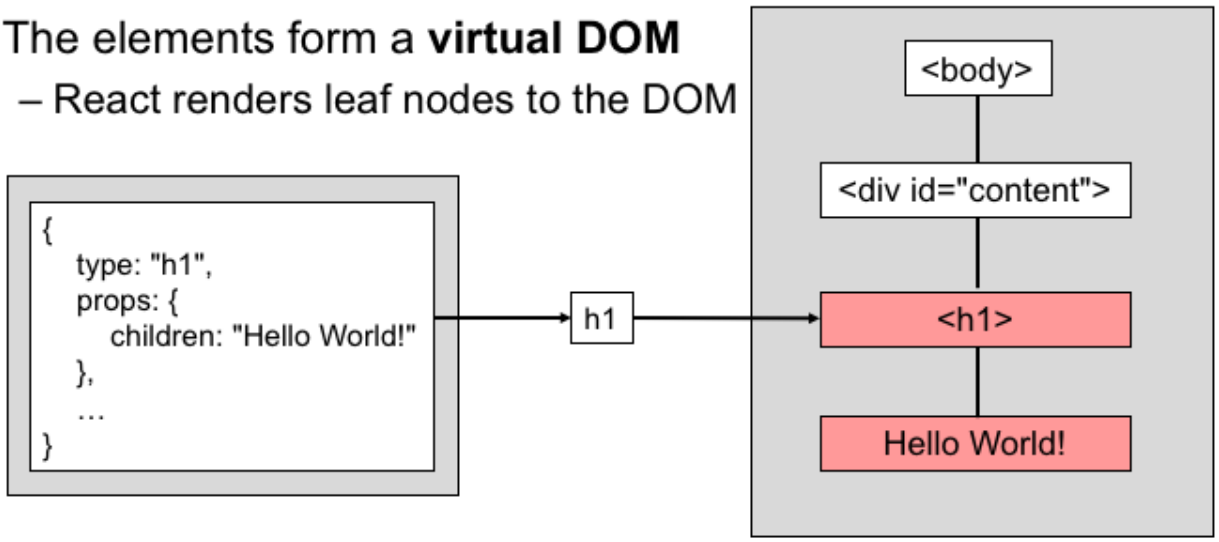
Notes

React Elements

- **createElement** builds a **React Element** *for* a component
  - The *h1* component represents a browser DOM *h1* element

```
// let element = <h1>Hello World!</h1> // JSX tags are calls to createElement
let element = React.createElement('h1', null, 'Hello World!')
```

- The elements form a **virtual DOM**
  - React renders leaf nodes to the DOM



Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

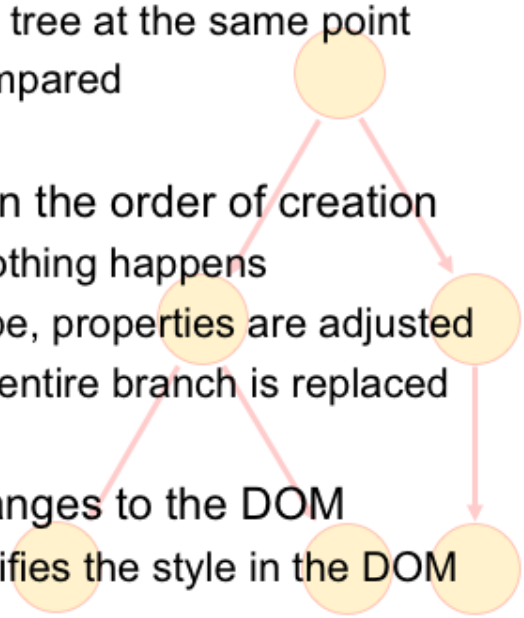
Notes

- The second parameter to createElement are the *properties* or *props* of the element.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Reconciliation

- React keeps a record of the entire virtual DOM after rendering
- Rendering a component returns a tree of React Elements
  - The new tree is compared to the old tree at the same point
  - Only the changed portion will be compared
- Elements are by default compared in the order of creation
  - If the two elements are the same, nothing happens
  - If the two elements are the same type, properties are adjusted
  - If an element is a different type, the entire branch is replaced
- React is efficient and minimizes changes to the DOM
  - E.g. changing a CSS style only modifies the style in the DOM



Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

- Reconciliation is the key point in React. React programs are stable an uncoupled because they are always rendered forward. Props flow downhill, and two-way bindings are minimized. To make that fast, reconciliation of plain-old-javascript-objects is avoids a slow comparison of browser DOM objects and properties, and React quickly finds the minimal changes to make to bring the browser DOM into line with the virtual DOM.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Changing Elements

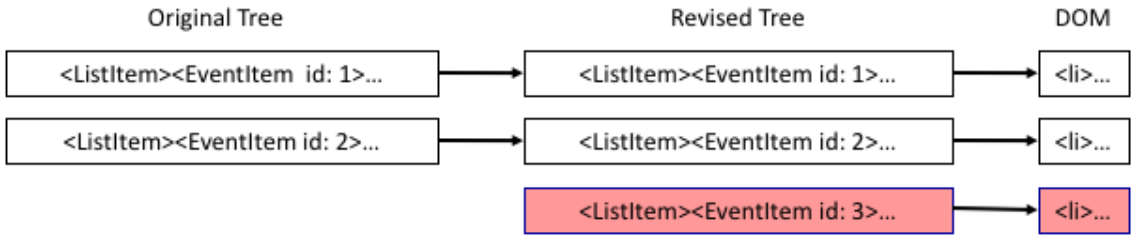
- Consider adding an Event to the list with id 3

```
let events = [   { id: 1, name: 'Historical Tour of San Juan', … },
                 { id: 2, name: 'Scuba Diving', … },
                 { id: 3, name: 'Ghost Tour', … } ]
```

- The ListItems contain EventItems

```
this.children.map( (item) => <ListItem>{ item }</ListItem> )
```

- React matches two existing elements to existing components
  - The new element is added to the end with a new component



Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

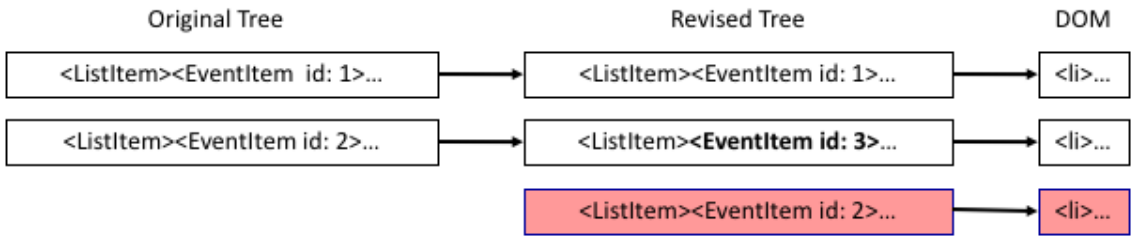
Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Changing Elements

- But if the new Event is added in the middle

```
let events = [   { id: 1, name: 'Historical Tour of San Juan', ... },
                 { id: 3, name: 'Ghost Tour', ... },
                 { id: 2, name: 'Scuba Diving', ... } ]
```



- React matches the new element 3 to the old element 2
  - And updates the properties of the element and component
  - Then element 2 and a new component are instantiated (again)
  - *EventItem 3 has the original EventItem 2's state!*
  - *EventItem 2 gets a new, initial state!*

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

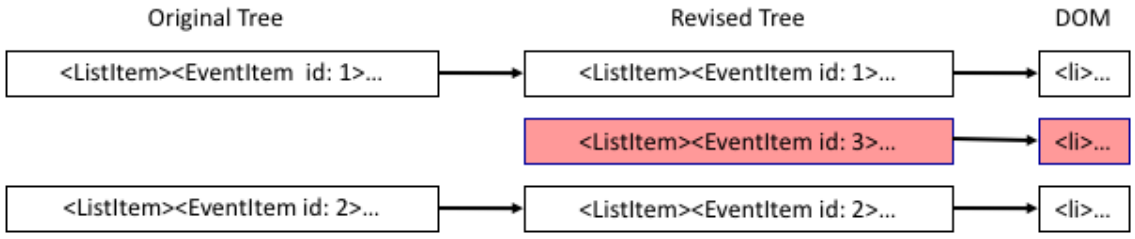
List Keys

- React fixes the problem with keys added to the elements
  - The keys can be anything, but need to be unique in the context
  - React uses the key to match existing elements

```
this.children.map( (item) => <ListItem key={ item.id }>{ item }</ListItem> )
```

- Never re-use keys, or elements will be matched incorrectly

```
let events = [   { id: 1, name: 'Historical Tour of San Juan', ... },
                 { id: 3, name: 'Ghost Tour', ... },
                 { id: 2, name: 'Scuba Diving', ... } ]
```



Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

List Keys

- The Event example has two lists to render
  - Eventually the EventItems are separate, but they start as a list
  - Adding a key to the EventItems is easy, use the record id!

```
export default class EventList extends Component {
   render() {
      let events = this.props.events.map( (event) => {
          return <EventItem key={ event.id }  event={ event } />
      }
      return <List keyProp='key'>{ events }</List>
   }
}
```

- Keep ListItem uncoupled from the EventItem:
  - Do not assume the contained element field is named "key"
  - Pass the List the name of the field to use with the ListItems

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Reconciling Lists

- Adjust the ListItem to use the appropriate field as its key

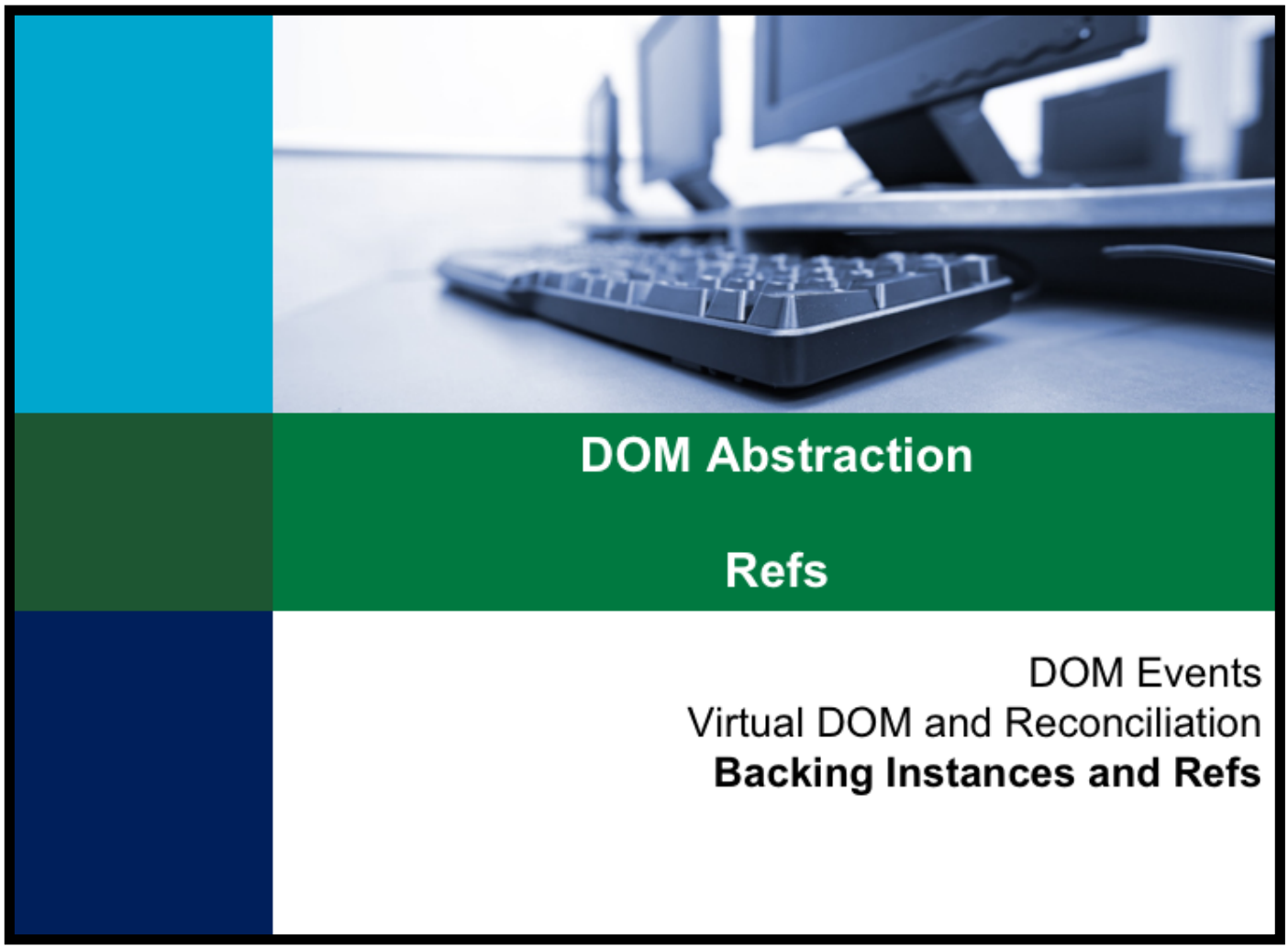
common/List.jsx

```
export default class List extends Component {
   constructor(...args) {
      super(...args)
      this.state = { displayList: false }
   }
   render() {
      let items = null
      if (this.state.displayList) { items = this.props.children.map( (item) => { return (
         <ListItem key={ item[this.props.keyProp] } >{ item }</ListItem> ) }) }
      return ( <div className='list' onClick={ (event) =>
         this.setState({ displayList: !this.state.displayList }) } }>
         <div><div className={ this.state.displayList ? 'list-title' : 'list-title-closed' }>
            { this.props.title }</div></div>{ items }</div> )
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Browser DOM

- Manipulating the browser DOM is a bad idea
  - The react philosophy is to render a whole DOM tree
  - The flow is from complex components down to simpler ones
- But there are some cases where it is not easily avoided
  - e.g. setting the focus to a particular control after an event
  - Perhaps an initial form load or after a validation failure
- To handle these cases React provides **refs**

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Refs and Ref by ReactDOM.render

- **Refs** are
  - References to the *backing instance*, which is the DOM node
  - Or references to component instances
- What you get depends on how you get *it* and whatever *it* is
  - ReactDOM.findDOMNode(element) locates a *backing instance*
- ReactDOM.render returns a reference to the *backing instance*
  - Or null if there is no backing instance
  - This really only works if render produces an HTML node, and not if render produces another component

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

String Refs

- A component instance can be labeled with ref='name'
  - The label appears as a field of **this.refs** in the parent
- What you get depends on the component
  - The *backing instance* if there is one,
  - or else the component instance
- ***ref strings*** *are deprecated!*

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Ref Callback

- The **ref** prop may be a callback function
  - Callbacks are made after the tree is rendered in reverse order

```
export default class AddressForm extends Component {
  constructor(...args) {
    super(...args)
    this.state = {
      address: '',
      city: '',
      stateOrProvince: '',
      postalCode: ''
    }
  }
  render() {
    return (
      <div>
        <label>Address <input type='text' value={ this.state.address }
          onChange={ (event) => { this.setState({ address: event.target.value }) }}
          ref={ (ref) => { ref.focus() }} /></label><br/>
        ...
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

- Because the reference is made to an input field which translates directly to a DOM node, the backing instance is passed as the function argument.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Ref Callback

- The **ref** callback gets
  - The backing instance for HTML components
  - The Component instance for custom components
  - May be null under certain circumstances
- A null reference indicates one of two things
  - The component has been unmounted
  - The ref callback is an inline function (not a method)
- An inline ref callback function-object is created on each render
  - So React passes the previous version null to say it is "done."

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

- Because the reference is made to an input field which translates directly to a DOM node, the backing instance is passed as the function argument.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Checkpoint

- What is the purpose of the virtual DOM?
- Why does React want keys for components in a list?
- When might you need a reference to a browser DOM node?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Module 5

Architecture with Components

Prop Validation
Data Flow
Component Lifecycle

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

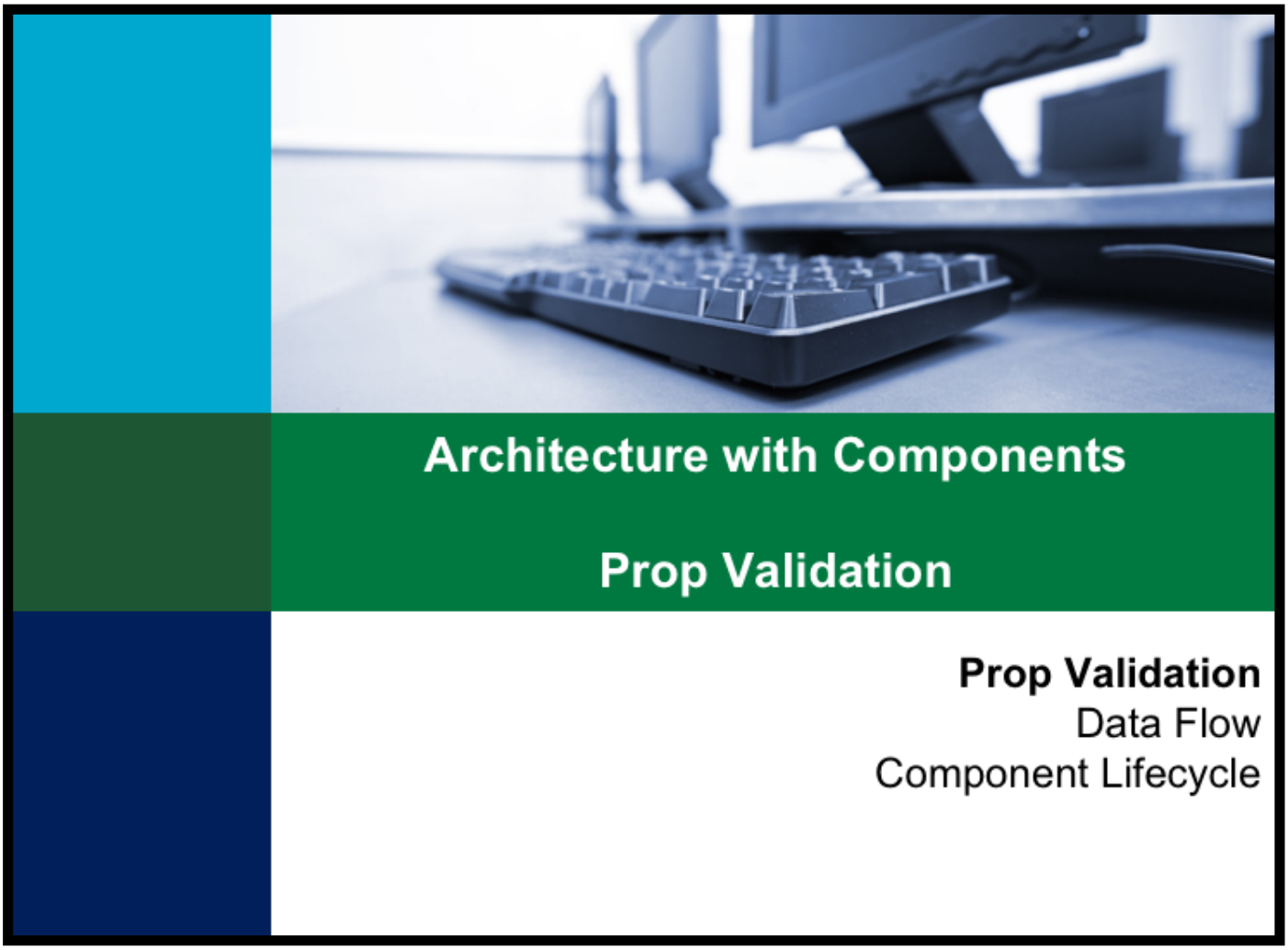
Objectives

- Validate props in a component
- Understand the data flow and immutability
- Learn how to design an application using components

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Prop Validation

- Prop validation
  - Checks the existence and type of props during component instantiation
  - Allows React to issue console warnings about problems

- Prop definitions are attached to the class object as **propTypes**[1]
  - Each field matches a property and defines its characteristics

```
import React, { Component} from 'react'
import PropTypes from 'prop-types'

export default class List extends Component {
   constructor(...args) {
   ...
}

List.propTypes = {
   title: PropTypes.string.isRequired
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] ES6 class definitions define a behavioral contract with the client code. It is intentional that class data members cannot be defined in the class declaration.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Keeping PropTypes in the Class

- **PropTypes** can be moved to the class as a static getter:

```
import React, { Component} from 'react'
import PropTypes from 'prop-types'

export default class List extends Component {
  constructor(...args) {
  ...
  static get propTypes() = {
    return {
      title: PropTypes.string.isRequired
    }
  }
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 5-5

Notes

Validators

Validator	Description
array	The prop must be an array
arrayOf	An array of a particular type: PropTypes.arrayOf(PropTypes.bool)
bool	A Boolean
element	A React element (a component instance)
func	A function object
instanceOf	*instanceOf* a JavaScript class: PropTypes.instanceOf(Error)
number	A number or may be parsed into a number
object	An object reference
objectOf	The object must have property values of a certain type: PropTypes.objectOf([PropTypes.number, …])
oneOf	On of a specific value: PropTypes.oneOf(['one', 2])
oneOfType	An array with a list of acceptable types: PropTypes.oneOfType([PropTypes.bool, …])
node	The prop must be a value that may be rendered
string	A string

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Default Prop Value

- **defaultProps** supports default prop value declarations
  - The value will be used if the prop is not passed to the component
  - Props with defaults are never required
  - defaultProps coexists with propTypes

```
export default class List extends Component {
  constructor(...args) {
  ...
  static get defaultProps() {
    return {
      title: 'Untitled List'
    }
  }
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Custom Validators

- A validator may be function object reference
  - Receives **props**, the prop to validate, and the component type
  - If the function returns an *Error* it is logged as a warning

```
static get propTypes() {
   return {
      title: (props, propName, componentName) => {
         result = null
         if (!props[propName]) {
            result = `\`${propName}\` is required in \`${componentName}\``
         } else {
            let value = props[propName]
            if (typeof value !== 'string' || value.length > 10) {
               result = `\`${propName}\` may not exceed ten characters in
\`${componentName}\``
            }
         }
         if (result) { return new Error(result) }
      }
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 5-8

Notes

- Note the ES6 string interpolation when the Error object is created. The extra grave accents are to be consistent with the React built-in error messages, which use grave accents.

Custom Validators

- Babel breaks instanceof during transpilation
  - Prototypes.instanceOf is broken too
  - Build a duck-type checker to see if an object looks right

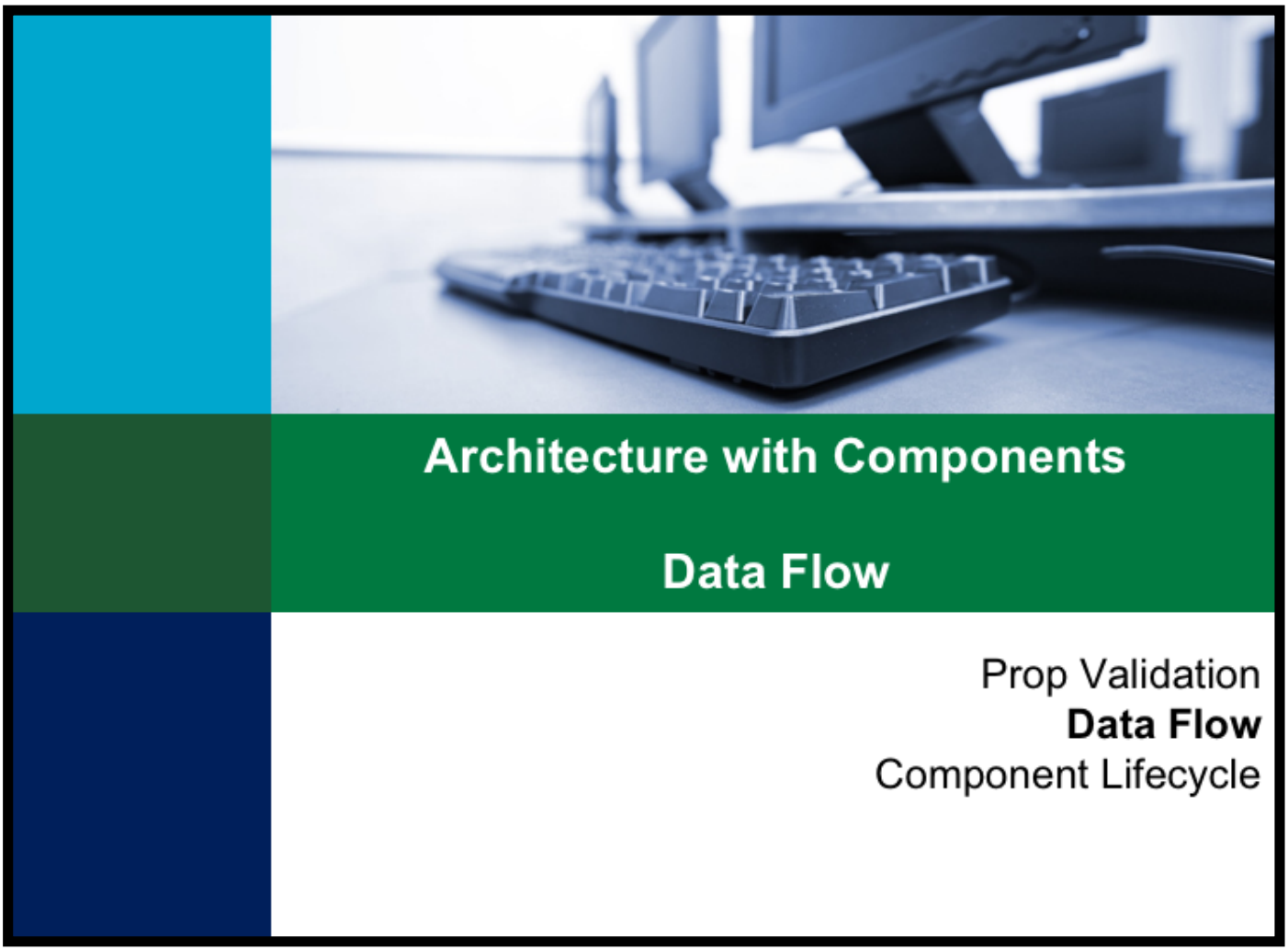
events/Event.jsx

```
…
static looksLikeEvent(props, propName, componentName) {
   let result = null
   let event = props[propName]
   if (typeof(event.id) === "undefined" || typeof(event.name) === "undefined" ||
      typeof(event.date) === "undefined" || typeof(event.tz) === "undefined" ||
      typeof(event.venue) === "undefined") {
         result = new Error(`\`${propName}\` must be an instance of \`Event\` in
\`${componentName}\``)
   }
   return result
}

export const looksLikeEvent = Event.looksLikeEvent
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 5-9

Notes

- Of course checking for particular types violates the principle of low-coupling and that JavaScript is a dynamic language: the party passing the data should have gotten it right to begin with.

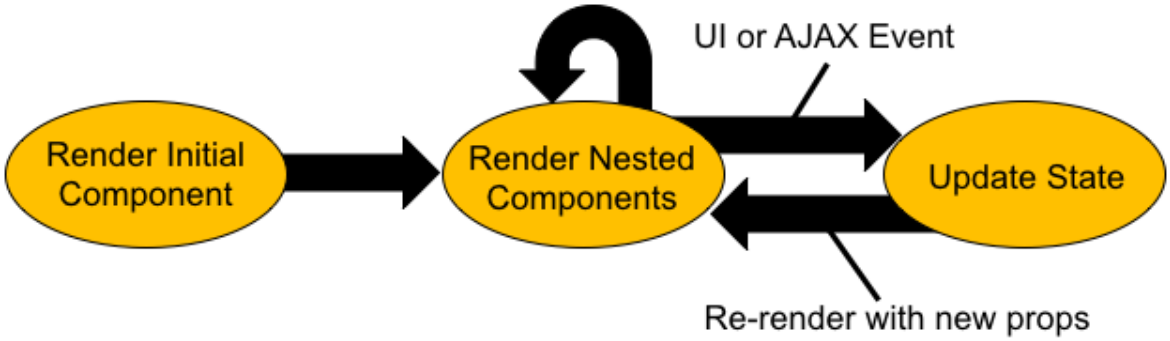


Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Application Flow

- The application flow is the key to data flow



  - A virtual DOM branch is rendered when an event triggers a callback that changes state – from the component on down
  - Push data down into nested components as prop changes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Data Flow

- In a React application data flows downhill
  - Parents pass data to children via props
  - The props can be changed by a parent, which may cause the child component to be re-rendered
- State is changed with callback functions
  - The UI or an asynchronous event triggers the callback
  - The setState method updates data and triggers rendering of the branch from the component down
  - Props and state data can be rendered to the page
  - Data can be passed down to children as props

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

State vs. Props

- Separate state from props[1]
  - props come from the parent, state comes from user input or asynchronous data requests
- State should stay within a component
  - Changing state causes the component to be rendered
  - Data rendered comes from props or state
  - Data passed to nested components comes from props or state, but is always passed as a prop
- Props should not affect state
  - Render prop data (or calculations) or pass it down
  - Prop data should not affect state – they can change from above

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] When a parent changes a prop, we keep that new value, there is no point in saving it as state data. Changing state while inside of a render caused by changing state is not supported and will cause an error.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Pure and Stateful Components

- A **pure** component has no state
  - It only renders prop data or passes it on to children
  - Presentation components are often pure components[1]
- A **stateful** component keeps its own state
  - The state is updated by the callbacks, causing rendering
  - Avoid changing the state in one component from another component
- Stateful components are usually parents
  - Pure components are usually children, farther down the tree
  - React DOM elements are the leaves on the tree

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

1 It is a little complicated; presentation components may use state to control the presentation, as opposed to state which saves data, and may copy data from another source to state in order to support bound controls.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Container Components

- **Container components** are used to manage data
  - They usually only render complex presentation components
  - They push data to presentation components as props
  - Containers usually manage state

```
export default class HelloWorld extends Component {
   render() { return <h1>{ this.props.message }</h1> }
}
```

```
export default class AppContainer extends Component {
   let message = 'Hello World'
   render() { return <HelloWorld message={ message } /> }
})
```

```
window.addEventListener('load', () => {
   ReactDOM.render(<AppContainer />, document.getElementById('content'))
})
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 5-15

Notes

- So the *only* difference between this and our projects so far is that the data is moved out of the app.jsx launching the application and placed in its own component. See the next page…

Immutability

- Immutability is an idea, not a restriction
  - Never change data, replace it and render again!
- Props should be immutable in components that receive them
  - The props and the fields in props are immutable
  - But objects referenced by prop values are not immutable but should be treated as if they were
  - e.g. an array cannot be replaced, but the elements can be
- To replace data use the Kolodny immutability helper
  - Replacing data means copying everything and making changes[1]

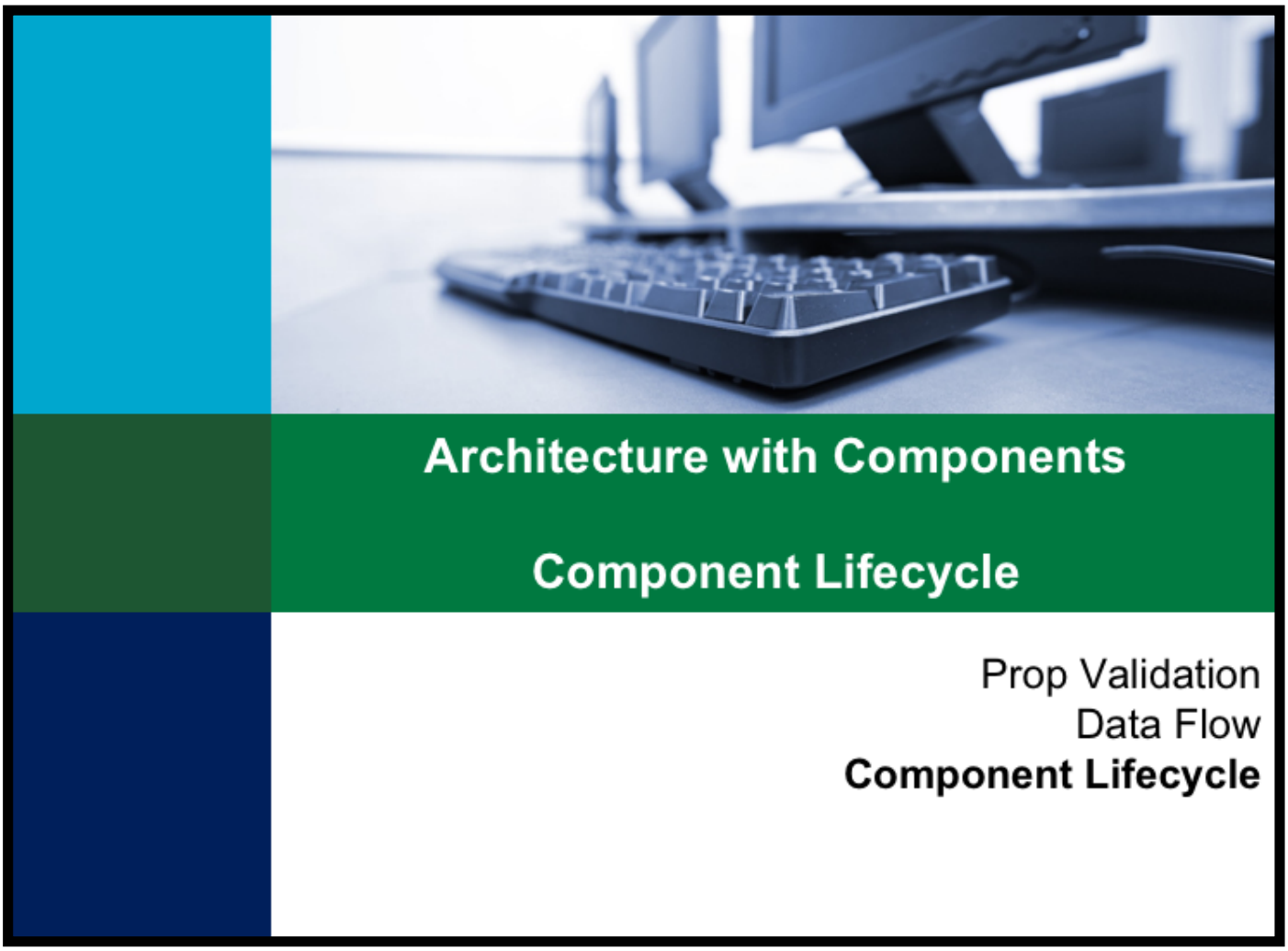
```
import update from 'immutability-helper'

let newData = update(oldData, { x: { y: { z: { $push: [ 99 ] }}}})
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

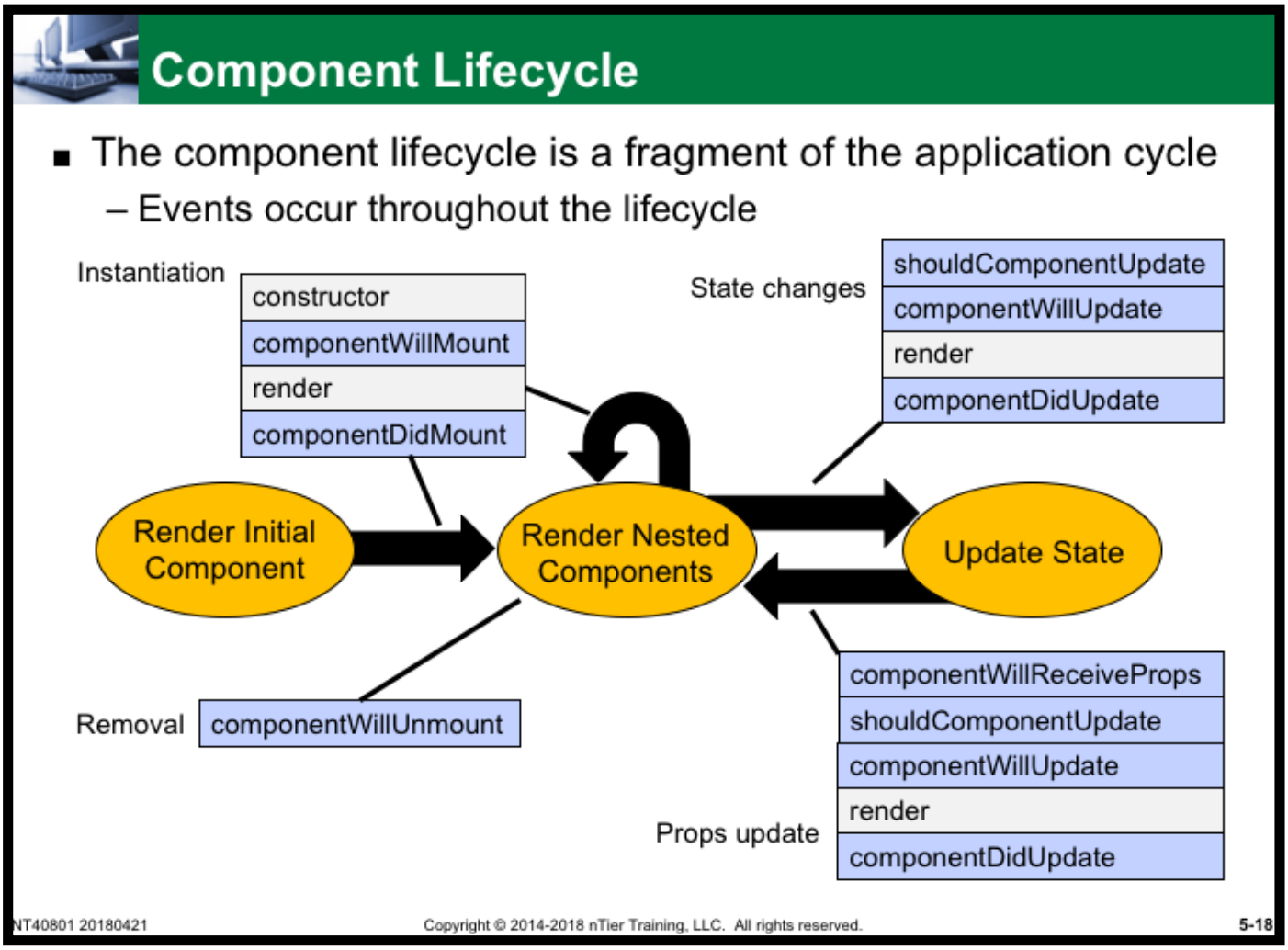
[1] The change verbs are similar to MongoDB operations on data. This example pushes 99 onto the target array z, which is a property of y, which is a property of x, which is a property of the old data.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



- These are all methods of the React.Component class, inherited by all components, and appear in the order that they fire.
- Any change to the props is a superset of of the methods called to change state: componentWillReceiveProps is called too.

Notes

Event Handlers

- The key handlers get the next (or previous) props and state
  - This allows comparisons before any changes are made

Handler Signatures
constructor(props)
componentDidMount()
componentDidUpdate(prevProps, prevState)
componentWillMount()
componentWillReceiveProps(nextProps)
componentWillUnmount()
componentWillUpdate(nextProps, nextState)
shouldComponentUpdate(nextProps, nextState)

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Component Lifecycle

- The state change and props update cycles are similar
  - Both share the same event methods
  - And both have an opportunity to reject the changes
- The really important methods in stateful components:
  - The constructor
  - componentDidMount
  - componentWillUpdate
- These are the opportunity for AJAX requests to be made
  - The callback for the request should set the state
  - Which causes the rendering in the component, and potentially props changes in the children

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Fetch

- The promise-based fetch
  - **window.fetch** is a cleaner way to handle AJAX requests

```
export default class EventsContainer extends Component {
   constructor(...args) {
      super(...args)
      this.state = { events: [] }

      let req = fetch('/data/events')

      req.then( (response) => response.json() )
         .then( (data) => this.setState( { events: data } ) )[1]
         .catch( (error) => { ... } )
   }
   render () {
      return <EventList items={ events } />
   }
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 5-21

Notes

[1] data is an array of event items, or more correctly an array of objects that look like event items.

- An alternative is to combine the JSON conversion and setState, but then there is no separation of concerns: req.then((response) => this.setState({ events: response.json() }).

Type

- In React it helps with validation if data in props has type
  - Data really has a "schema," we expect a certain structure
  - Use Object.assign() to merge data with correct type
  - This can be done in the constructor, passing the raw data

```
export default class Event {
   constructor(obj) { Object.assign(this, obj) }
   get id() { return this._id }
   ....
}
```

```
      ...
      let events = null
      if (data && Array.isArray(data)) {
         events = data.map( (rawEvent) => new Event(rawEvent) ) }
      this.setState( { events: events } ) })
      ...
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Pushing Data

- Fetch supports all the HTTP methods
  - Add the *init* object that defines what to do
  - Add a *Headers* object

```
import 'whatwg-fetch'[1]

let headers = new Headers()
let events = [ { id: 999, name: 'Old Kingston Tour', ... }, ... ]

headers.set('content-type', 'application/json')
headers.set('accept', 'application/json')

let req = fetch('/data/events', {
   method: POST,
   headers: headers
   body: JSON.stringify(events)
})

req.then( (response) => { ... } ).catch( (error) => { ... })
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] The import of whatwg-fetch is a "poly-fill," where if fetch is not defined on the window object the poly-fill will add the functionality.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

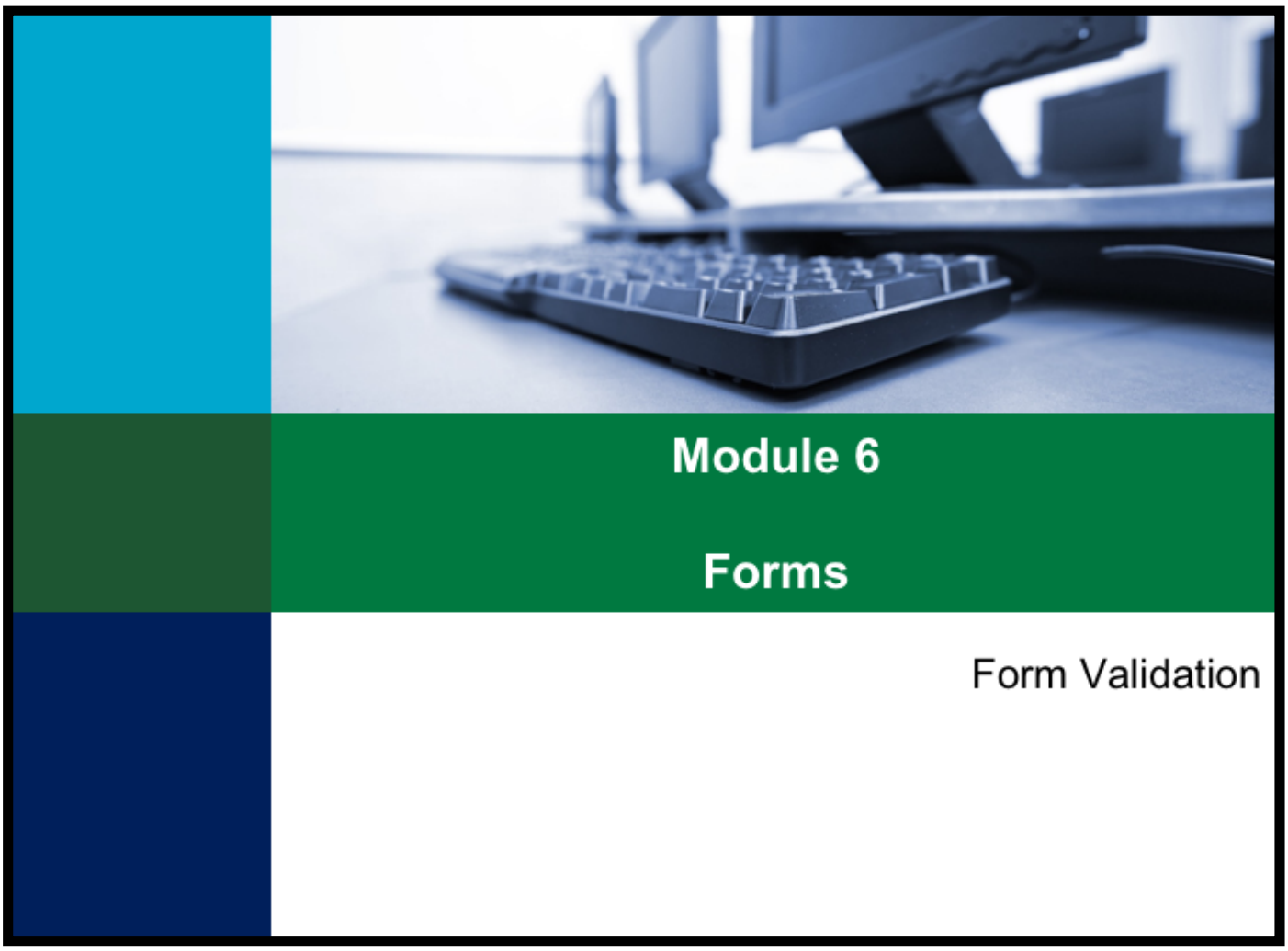
Checkpoint

- What does React prop validation do?
- What do custom validators add?
- How should data flow in a React application?
- What is point of "container" components?
- How are component lifecycle event handlers written?
- Where will you use lifecycle events?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

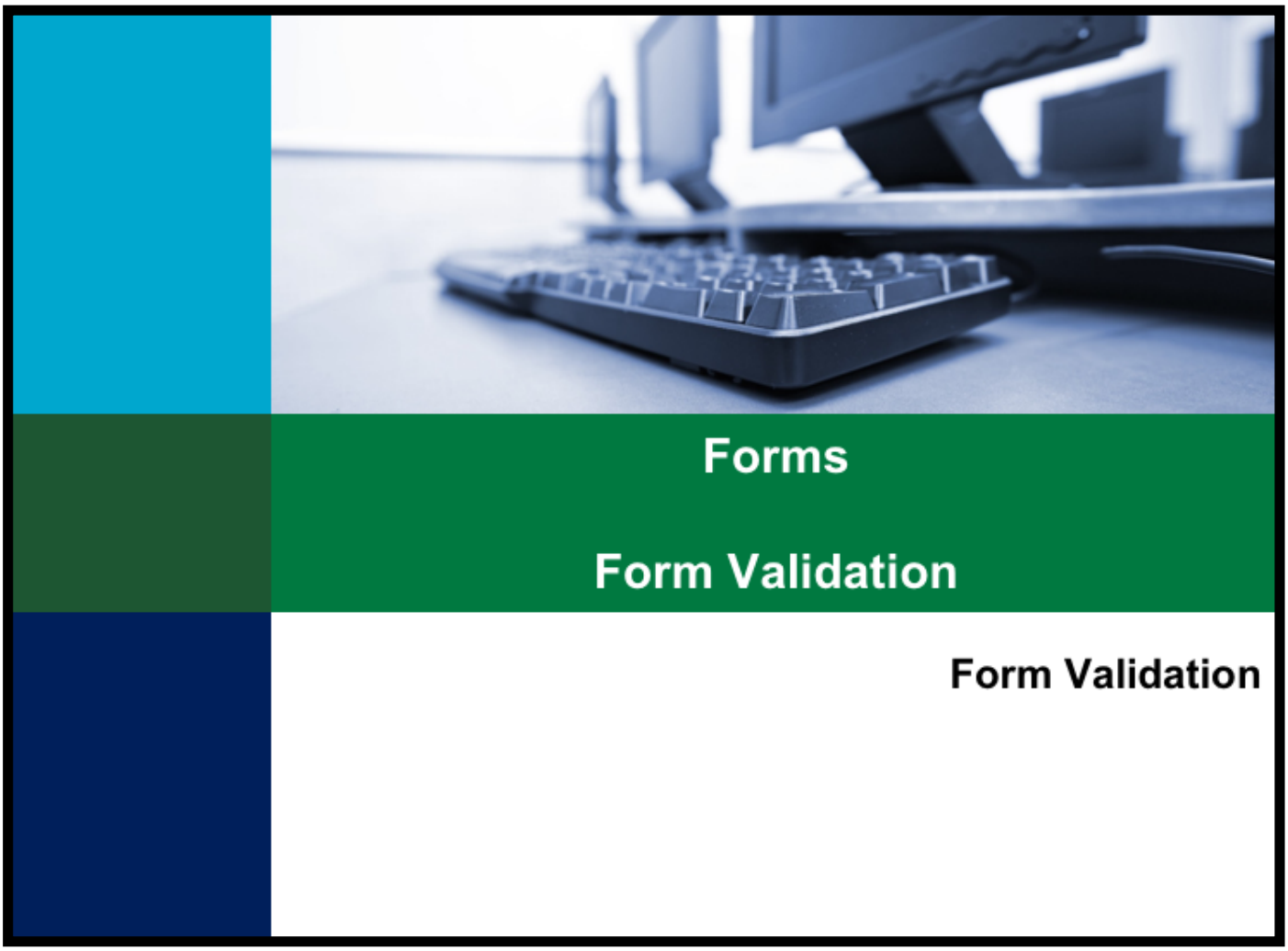
Objectives

- Expand on forms
- Address field validation
- Create a validation component

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Form Validation

- An important benefit of client-side applications is validation
  - The earliest JavaScript applications focused on field validation
- React's downhill flow of data interferes with this
  - Data is a child component and does not flow uphill to the parent
  - It is tedious to get any information from the child component
- Validation could take place completely in the parent
  - Tedious to show and hide messages during validation
  - Need to trigger methods that both manage state and validation
  - Encourages duplication of validation algorithms
- There is no standard solution for React applications!

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Form Validation

- One solution is to put *standardized validation* in a component
  - The component sees the same data in the form field
  - The component can handle the messages
  - The component can be placed anywhere in the interface
  - Multiple components can be instantiated for multiple fields
- Information must flow uphill
  - So information can flow uphill through a callback
  - If the *validator* is rendered because of a state change, the callback *cannot* initiate another state change
- Validate with static values, lists of values, numeric ranges, regular expressions, and callback functions

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Validator Component Code

- An example of a validator component

```
import React, { Component, PropTypes } from 'react'

export default class Validator extends Component {

   constructor(...args) {
      super(...args)
   }

   render() {

      let valid = this.validate(this.props.value, this.props.constraint)

      if (!valid) {
         this.props.notify()
      }
```

The source is continued on the next page

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Validator Component Code

```
    let rendered = null

    if (this.props.children && !valid &&
       (this.props.value || this.props.renderOnEmpty)) {

       rendered = ( <span className={ this.props.className }>
          { this.props.children }</span> )

    return rendered
}

validate(value, constraint) {

    let result = true
```

The source is continued on the next page

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Validator Component Code

```
if (this.props.isRequired && !value && value !== 0) {

   result = false
}

if (constraint instanceof RegExp) {

   result = this.validateRegex(value, constraint)
}

if (typeof constraint === 'object') {

   if (constraint.min && !isNaN(constraint.min) && constraint.max &&
      !isNaN(constraint.max)) {

      result = this.validateRange(value, constraint.min, constraint.max)
   }
}
```

The source is continued on the next page

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Validator Component Code

```
    if (Array.isArray(constraint)) {

       result = this.validateList(value, constraint)
    }

    if (typeof constraint === 'string') {

       result = this.validateExact(value, constraint)
    }

    if (typeof constraint === 'function') {

       result = this.validateWithCallback(value, constraint)
    }

    return result
}
```

The source is continued on the next page

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Validator Component Code

```
validateRegex(value, expression) {

   return expression.test(value)
}

validateRange(value, min, max) {

   value = parseFloat(value)

   return (value >= min && value <= max)
}

validateList(value, list) {

   result = false
```

The source is continued on the next page

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Validator Component Code

```
    for (let i = 0; i < list.length; i++) {

      if (list[i] === value) {

        result = true
        break
      }
    }

    return result
}

validateExact(value, exact) {

    return value === exact
}
```

The source is continued on the next page

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Validator Component Code

```
   validateWithCallback(value, callback) {

      return callback(value)
   }
}

Validator.propTypes = {
   PropTypes.string.isRequired,
   isRequired: PropTypes.bool
}

Validator.defaultProps = {
   isRequired: false,
   renderOnEmpty: false
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Checkpoint

- What does React prop validation do?
- What do custom validators add?
- How did a validator component help?

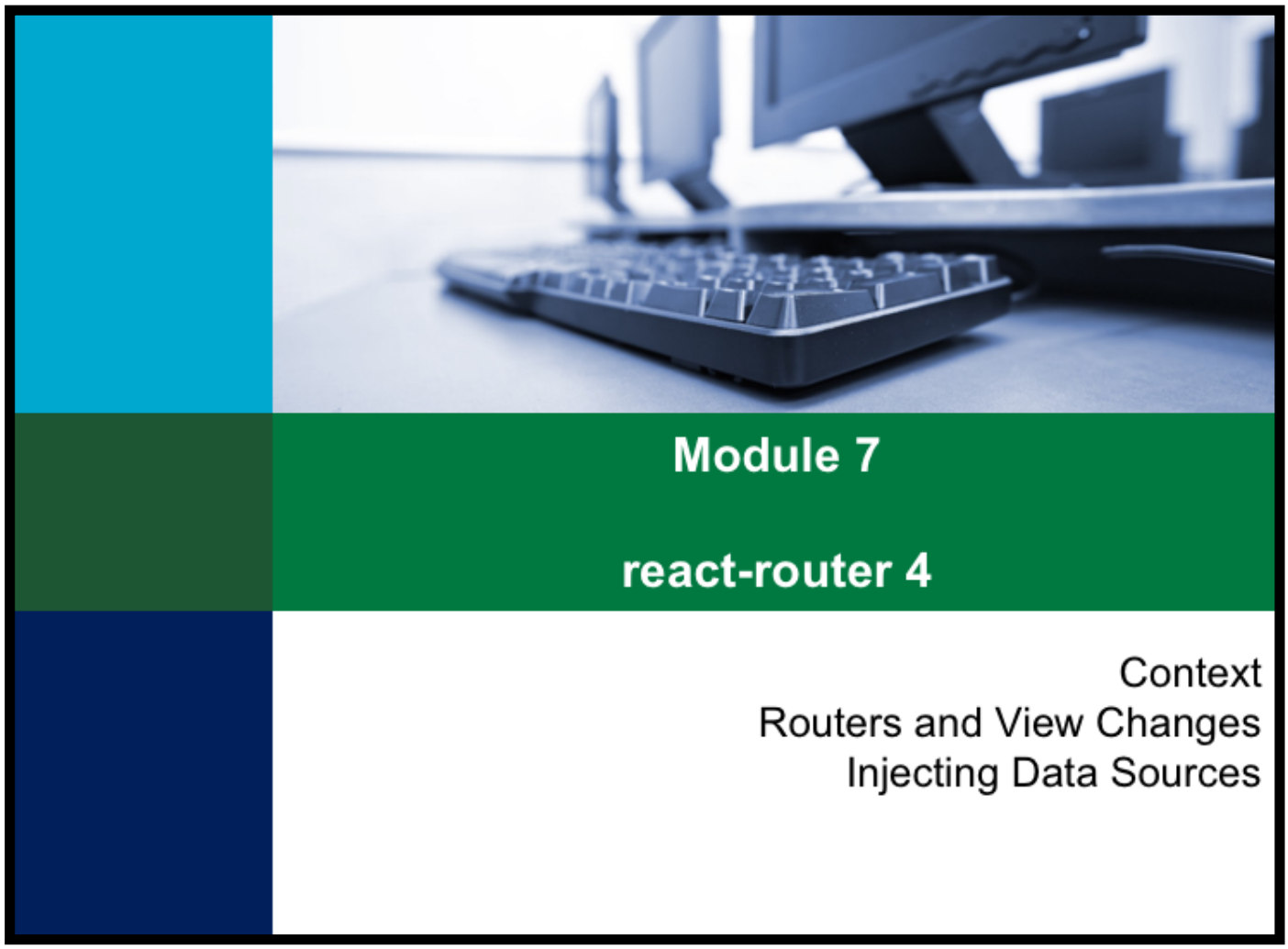
 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.


This page is intentionally blank
NT40801 20180421
Copyright © 2014-2018 nTier Training, LLC. All rights reserved.
6-14

Notes



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

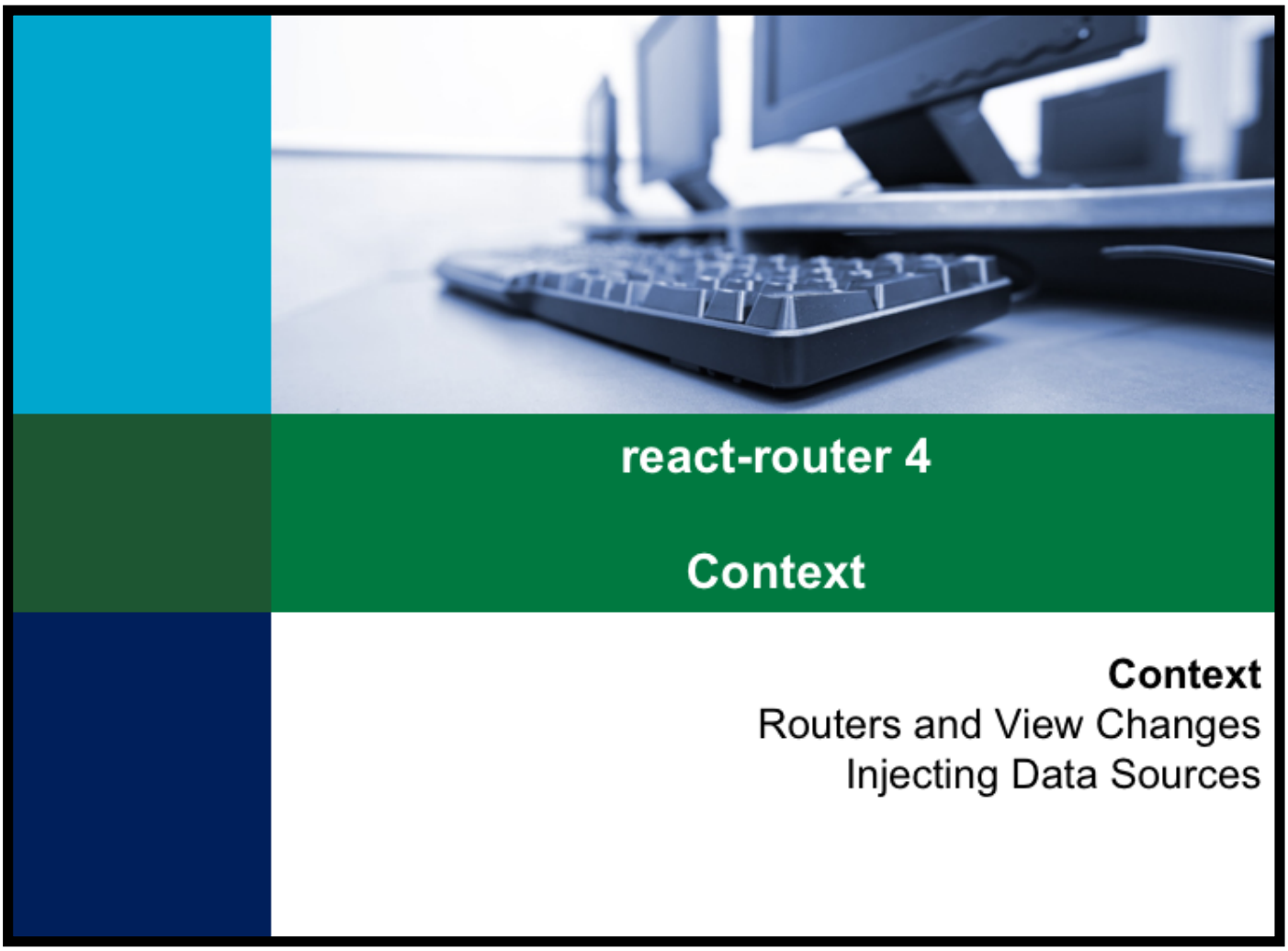
Objectives

- Address changing application views
- Use the routing framework to control view changes
- Inject data using a context through the router

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

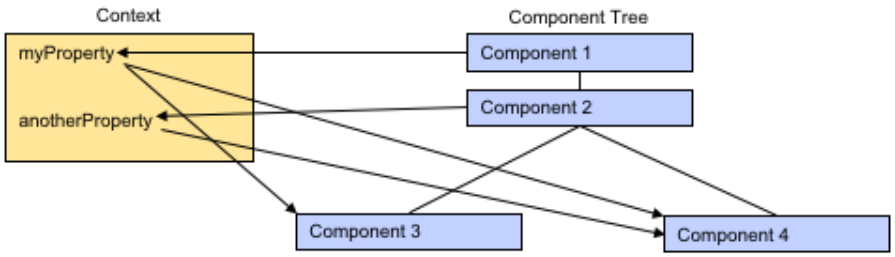


Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Props vs Context

- A child component is rendered by a parent component
  - This forms the tree of components
  - Data can be passed through props
  - Intermediates must pass a prop, but may not be responsible for it
- Context data is available to the tree of child components
  - Context allows multiple values to be passed as a single unit
  - Any component can add to the context
  - Components choose what they want to use



NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 7-4

Notes

New Context API

- React 16.3 introduces a new Context API, using components
  - Creating a context produces a provider and consumer

```
const mc = React.createContext('myvalue')

class Parent extends React.Component {
   render() {
      return (
         <mc.Provider value = { 'green' }>{this.props.children}</mc.Provider>)
   }
}

class App extends React.Component {[1]
   render() {
      return (<Parent>
         <mc.Consumer>{val => <div>{val}</div>}</mc.Consumer></Parent>)
   }
}
```

  - Unfortunately, react-router-dom 4.2 still uses the old API…

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

1 This is example is a little twisted, to be short. App renders a Parent, which provides a context. App passes to the parent content, and the content in the parent consumes the context. Clearly in most cases the consumer would probably not be rendered by the parent, but by another component rendered farther down the tree from the provider. And, all three things would be in separate modules to be imported.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Old API: Defining Context

- A parent defines the context that any child may receive
  - The **childContextTypes** defines the properties
  - **getChildContext** serves the context properties
  - **getChildContext** is used by React, not called directly

```
class MyParentComponent extends Component {

   getChildContext() {
      return { myProperty: "Hello, World" }
   }

   static get childContextTypes() { [1]
      return {
         myProperty: PropTypes.string
      }
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] Note that the property types are the same values used for property validation.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Old API: Using Context

- The child uses **contextTypes** to list required properties

```
class MyChildComponent extends Component {

  render() {
    return <div>{ this.context.myProperty }</div>
  }

  static get contextTypes = {
    myProperty: PropTypes.object.isRequired
  }
```

- The context is a mix of all parent contexts[1]
  - Filtered to just include the required properties

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] This is a big problem: context properties are essentially "global," and components can step on each other naming the properties.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Context and Life Cycle Events

- Certain life cycle events receive context

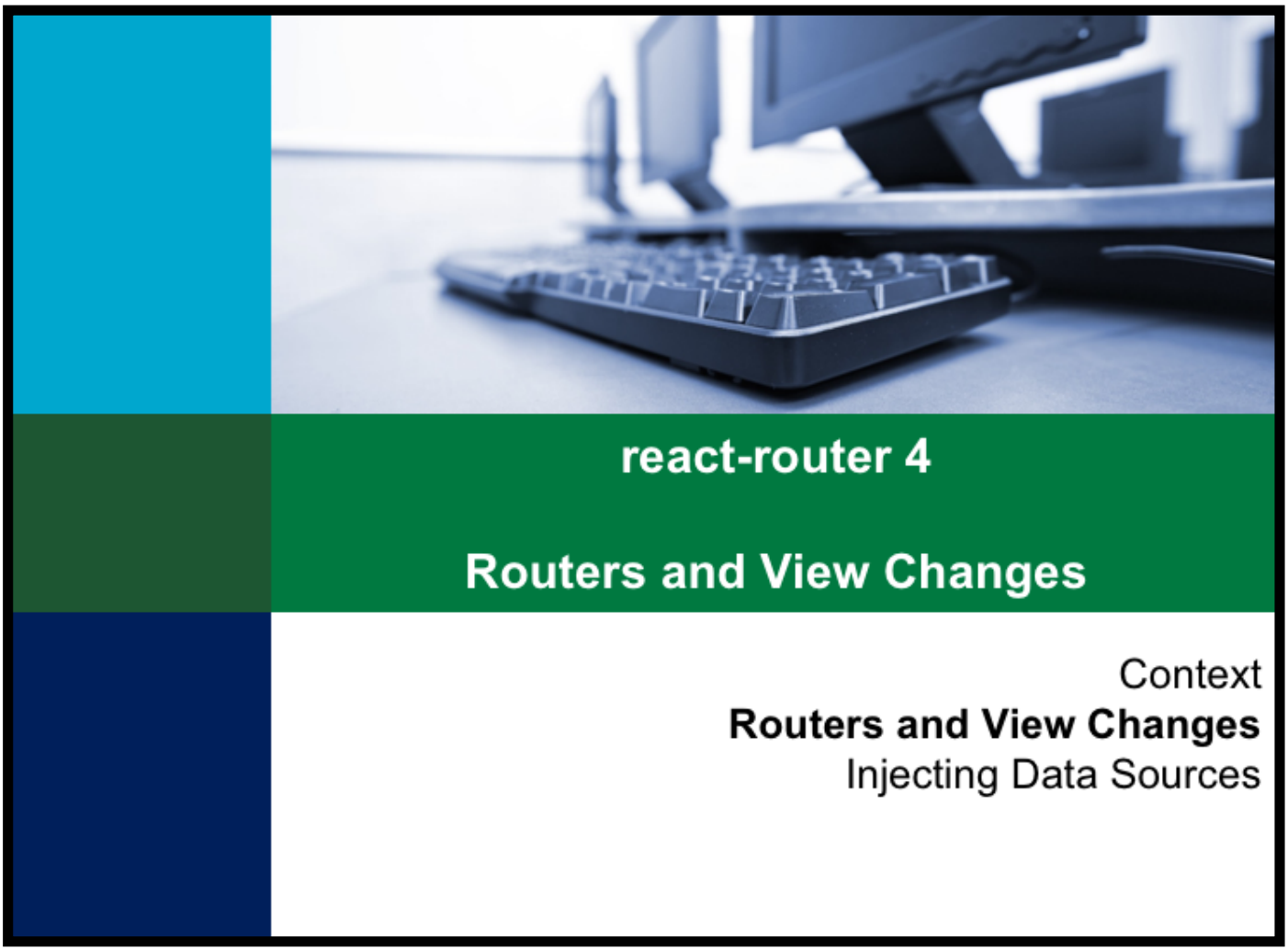
Handler Signatures
constructor(props, context)
componentWillReceiveProps(nextProps, nextContext)
componentWillUpdate(nextProps, nextState, nextContext)
shouldComponentUpdate(nextProps, nextState, nextContext)

- *Avoid context:* creates tight coupling between components
  - Parents need to provide what the children require
- *Use context:* when multiple children need to share a value
  - react-router-dom and Redux both use context

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Application Routing

- Routing controls how the application view changes
  - E.g. moving from a list of events to the data for one event
- React renders from the top down
  - A container needs to decide what will be displayed
  - An action in a child could trigger a callback to change the view[1]
- Routing frameworks simplify this
  - react-router configures the routes for the application
  - Groups of routes can be configured at multiple levels
  - Components change the view by changing the path

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] This violates the premise of React, because we are moving information upstream. We have to create the callback, the child needs to know to use the callback, etc.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

react-router

- **react-router** 4 *routers* and *routes* are components
  - A *router* defines that routes will be used below in the tree
  - A *router* must wrap a single component
- The routers use a history object to manage the view
  - *HashHistory*
  - *BrowserHistory*
  - *MemoryHistory*

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

History

- Defines how URLs look and provides for changing the URL
- hashHistory – uses the hash-mark, works with older browsers
  - http://localhost:8080/#/events
  - http://localhost:8080/events
- browserHistory – HTML5 URL format without #
- memoryHistory – working URL not in the address bar
- hashHistory and browserHistory allow the user to manually alter the address or bookmark it; that could be good or bad

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] The choice of the history component still applies even if the context object is used to change the URL.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Configuring History

- Import **react-router-dom**
  - react-router-dom depends on react-router and imports it
  - react-router-native is an alternate for non-browser environments

- The router imported sets the history model

```
import { MemoryRouter as Router } from 'react-router-dom'

ReactDOM.render(
   <Router> … </Router>, ...
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

withRouter

- Presentation components will need router values
  - The router is a component, it wraps a child component
  - At the top of the tree there are no props to forward

- withRouter is a factory that wraps an existing component
  - The component made will get the props for the router
  - If Main was the top-level component to render:

```
import { MemoryRouter as Router, withRouter } from 'react-router-dom'

let MainWithRouter = withRouter(Main)

ReactDOM.render(
  <Router>
    <MainWithRouter />
  </Router>, ...
```

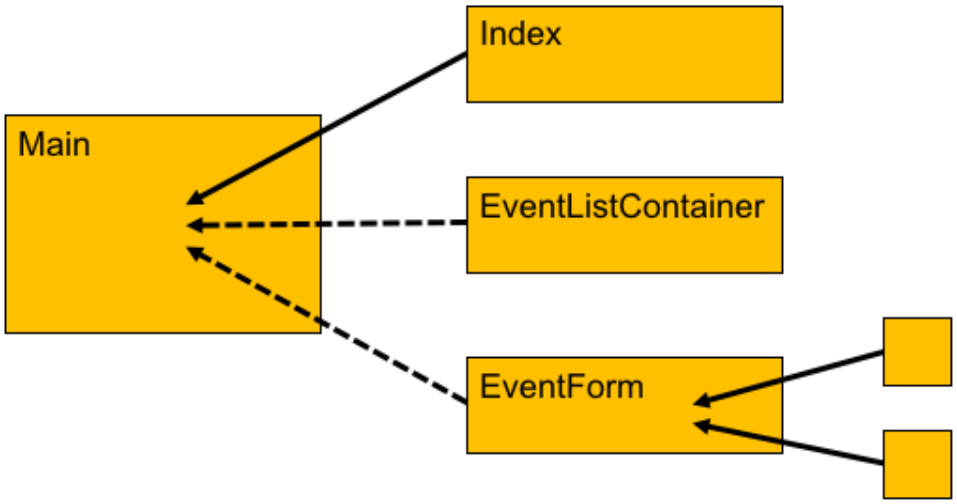

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Routes

- A group of routes define what to present for a particular path
  - The matching route(s) point to components to render



NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 7-15

Notes

Main

- The Main component is the boilerplate
  - The selected view component(s) will be rendered in place

app/Main.jsx

```
import React, { Component } from 'react'
import { Route, Switch } from 'react-router-dom'

import EventForm from 'events/EventForm'
import EventListContainer from 'events/EventListContainer'
import Index from 'app/Index'import Navigation from 'app/Navigation'

export default class Main extends Component {
   render() {
      return (
         <div>
            <div className='header'>
               <div className='logo'>
                  <img src='assets/images/kaleidoscope.png' />
               </div>
```

The source is continued on the next page

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Routes

app/Main.jsx

```
          </div>
          <Navigation location={ this.props.location } />
          <div className="page">
            <Switch>
              <Route exact path="/" component={ Index } />
              <Route path="/events/:id" component={ EventForm } />
              <Route path='/events' component ={ EventListContainer } />
            </Switch>
          </div>
        </div>
      )
    }
}
```

- The components(s) are rendered in the div "page"
- The Switch ensures that only the first matching route is selected
- **exact** forces an exact match for the path /

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Main

- Route does not allow props to be declared for the component
  - Route does provide for a **render** method that receives props
  - The JSX that render produces can have embedded props

app/Main.jsx

```
<Switch>
  <Route exact path="/" component={ Index } />
  <Route path="/events/:id" render={
(props) => <EventForm { ...props } dataContext={ this.props.eventContext } /> } />
  <Route path='/events' render={
() => <EventListContainer dataContext={ this.props.eventContext } /> } />
</Switch>
```

- The **props** passed to render contain **matches**
  - **matches.params** contains parameters from the path
  - **matches.params.id** contains the id from /events/:id

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Changing the URL

- Use context to get the router
  - The router **push** method changes the history
- The child component declares the context property

events/EventItem.jsx

```
EventItem.contextTypes = {
      router: PropTypes.object.isRequired
}
```

- The property can be used to push the new URL

```
... onChange={ (event) =>
   this.context.router.history.push(`/events/${this.props.event.id}`) } ...
```

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

- This is a good use of context: the router expects any component could use the context, and while the components that use the context are dependent on the router it is not unreasonable to depend on that one application-wide facility.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

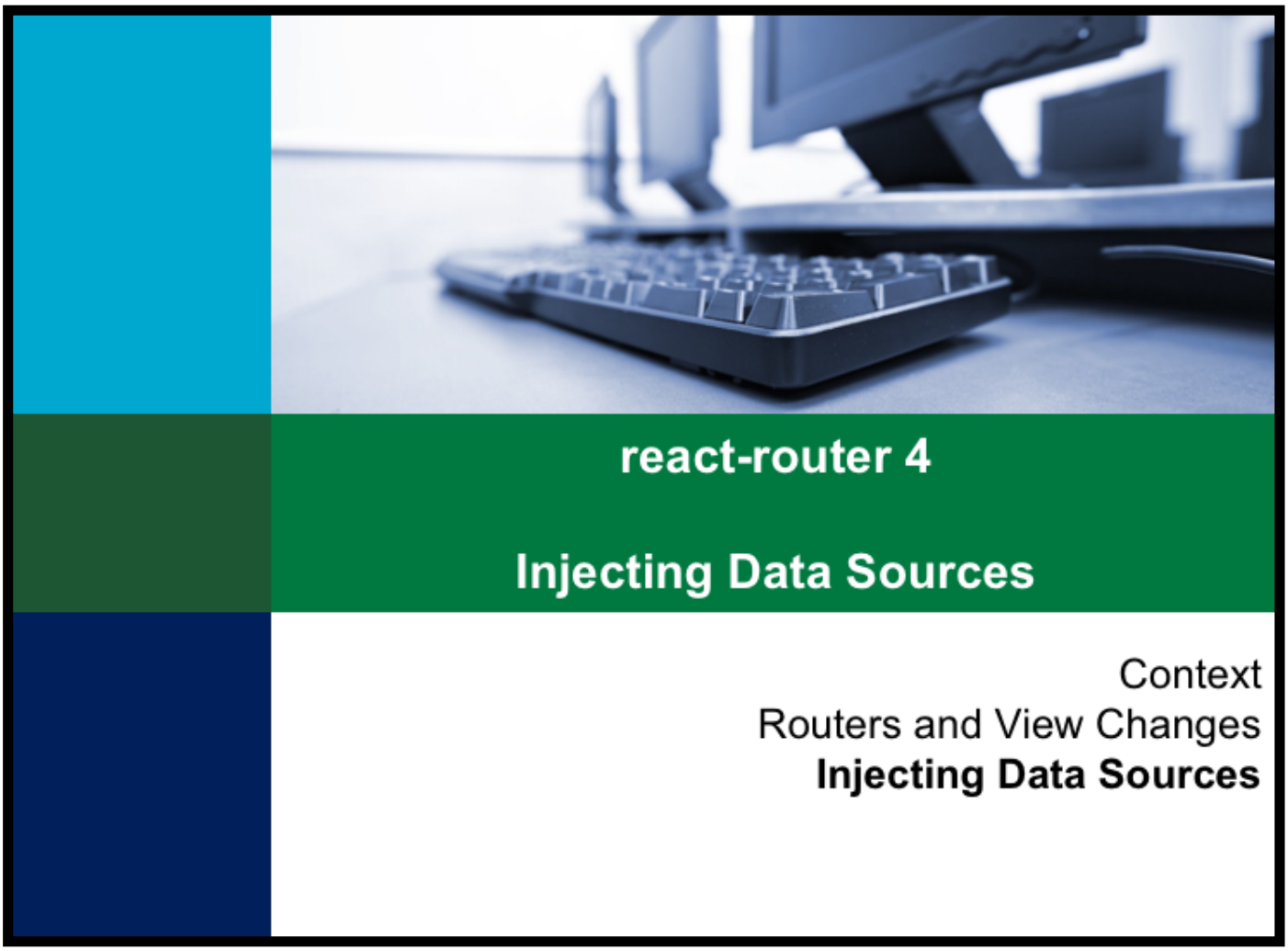
Link

- A **Link** is the equivalent of a static anchor tag
  - But it changes the URL triggering the router
  - <Link to='/events/1' activeClassName='highlight'>Click!</Link>
- The content is displayed as the link
- activeClassName is the CSS class used when active
  - The className prop still defines the CSS class for the normal state of the link
- Make a button a link by wrapping it
  - <Link to="/events/new"><button>New Event</button></Link>

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Separation of Concerns

- The EventListContainer and EventForm both work with Events
  - Dates need to be converted from/to ISO date strings
  - Separate the connection with the database into another class

data-access/EventContext.js

```
export default class EventContext {
  getEvents() {
    return fetch('/data/events')
      .then( (result) => { return result.json() } )
      .then( (rows) => {
        let result = []
        if (Array.isArray(rows)) {
          result = rows.map( (row) => this.buildEvent(row) )
        }
        return result
      })
  }
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 7-22

Notes

- Note that the method getEvents returns a promise, which is a promise that will complete after the results have been converted from JSON and converted into Events. The buildEvent method will handle converting row data into Event objects.

Dependency Injection

- The components should be given a EventContext
  - If it can be injected by the router then it can be changed[1]
  - Props given to the route become properties of the context

app/app.jsx

```
import { MemoryRouter as Router, withRouter } from 'react-router-dom'

let eventContext = new EventContext()
let MainWithRouter = withRouter(Main)

ReactDOM.render(
  <Router>
    <MainWithRouter eventContext={ eventContext } />
  </Router>, ...
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

- A different source that behaves the same could be substituted later.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

JSON Conversion

- To write Event objects they need to become JSON strings
  - Not the private fields, only what the ES6 getters return1
  - Accomplish this with a toJSON method in the Event class

```
toJSON() {
   return {
      id: this.id,
      name: this.name,
      date: this.date,
      tz: this.tz,
      venue: this.venue
   }
}
```

  - When an Event is stringified the date will become an ISO string

```
let data = JSON.stringify(event) // { "id": 99, ... }
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] This.id, etc. are properties of the class. Without a toJSON method, JSON.stringify will get all the fields and properties.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Getting and Setting Event Data

- The EventForm needs to get and save data
  - The dates need to be translated from and to ISO date strings
  - Methods go from Event objects to state fields and back again
  - The key is in the manipulation of the date using moment

buildStateFromEvent

```
buildStateFromEvent() {
   let date = moment(event.date.getTime()).format('L')
   let time = moment(event.date.getTime()).format('HH:mm')
   return { id: event.id, name: event.name, date: date, time: time, … }[1]
}
```

buildEventFromState

```
buildEventFromState() {
   let formattedDate = moment(Date.parse(this.state.date)).format('YYYY-MM-DD')
   let date = new Date(`${formattedDate}T${this.state.time}${this.state.tz}`)
   return new Event({  id: this.state.id, name: this.state.name, date: date, … }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] This method builds an object that can be merged with the state using setState.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Getting Event Data

- The EventForm needs to load data
  - Do this in the componentDidMount method
  - Do not load anything for new events (the URL has a 'new' id)!

```
componentDidMount() {
   if (this.props.matches.params.id !== 'new') {
      this.props.dataContext.getEvent(this.props.matches.params.id)
         .then( (event) => this.setState(this.buildStateFromEvent(event)))
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Saving Event Data

- We need a method the commit button can use to save data
  - The context method is different for inserts and updates

```
saveEvent() {
   if (this.state.valid) {
      let event = this.buildEventFromState()
      if (event.id) {
         this.props.route.dataContext.updateEvent(this.props.params.id, event)
            .then( (event) => this.context.router.history.push('/events') )
            .catch( (err) => { /* do something with this */ })
      } else {
         this.props.route.dataContext.insertEvent(event)
            .then( (event) => this.context.router.history.push('/events') )
            .catch( (err) => { /* do something with this */ })
      }
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Checkpoint

- How should data flow in a React application?
- What is the point of "container" components?
- Where will you use lifecycle events?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Module 08

Redux

Redux Architecture
Store and Reducers
Components and Action Creators

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Objectives

- What does the Flux architecture offer?
- Why do applications need to enforce Flux?
- How does Flux work with react?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Redux

Redux Architecture

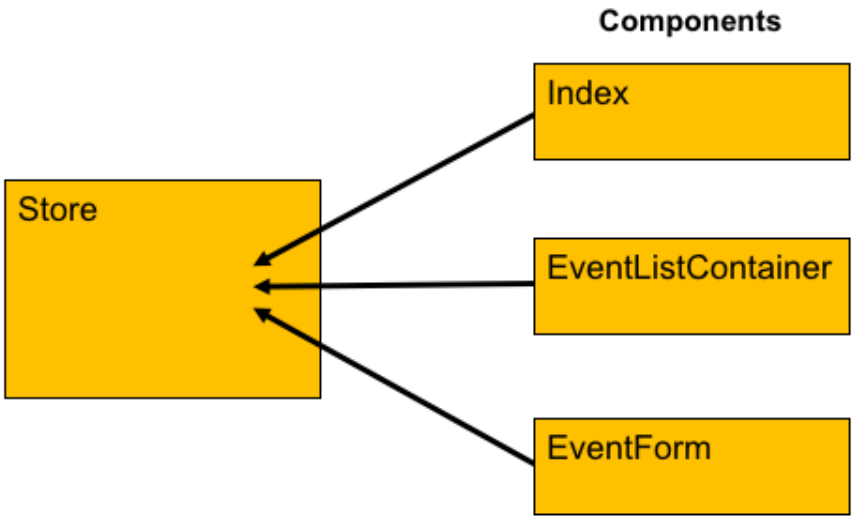
Redux Architecture
Store and Reducers
Components and Action Creators

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Redux Architecture

- The intention of Redux is to organize state
  - All application state is kept in a **store**
  - The store is **immutable**, read-only
  - New states are created through **reducers**



Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

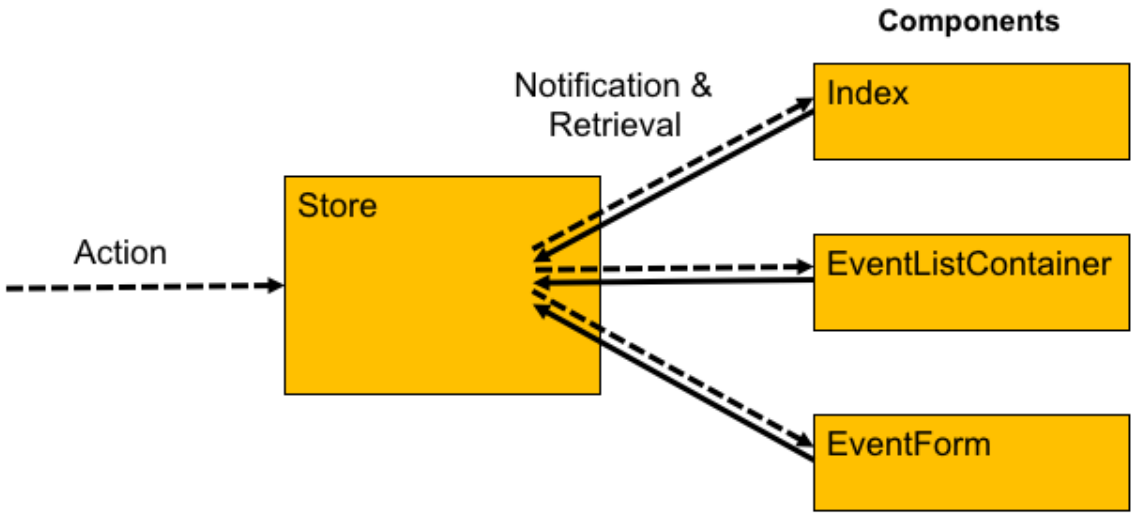
Notes

- The direction of the arrows show that the components observe the store.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Redux Architecture

- The components *observe* stores
  - **Notifications** occur when the store changes
  - The store only changes in response to an **action**

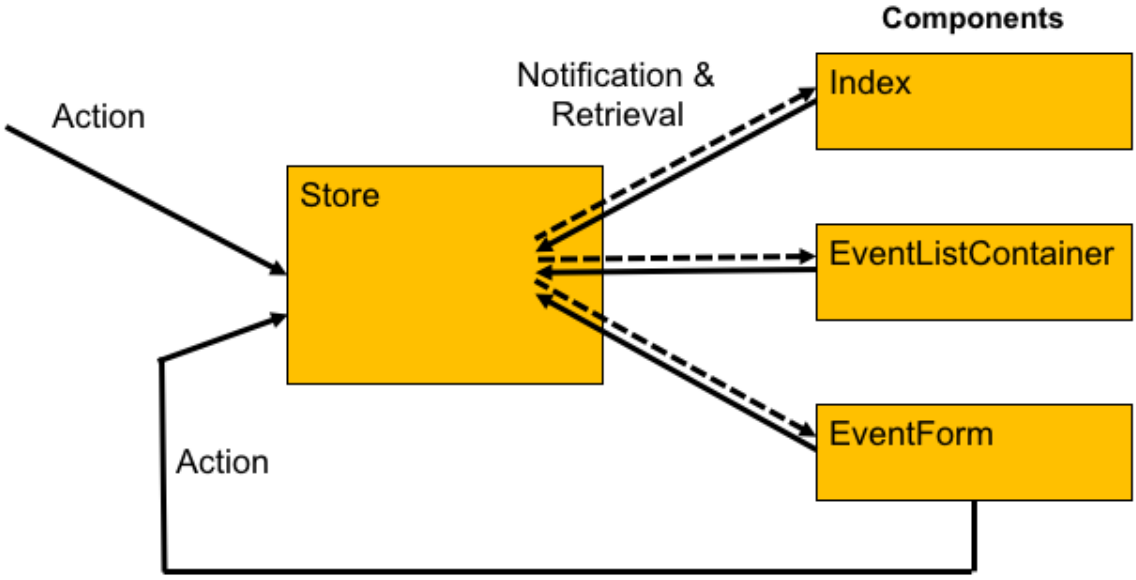


NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 8-5

Notes

Redux Architecture

- **Actions** are sent by clients using the store's **dispatch** method
  - An action is an object with the action and data



Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Reducers

- A dispatched action triggers the **reducer**
  - The reducer builds a new state from an existing state and action
  - A reducer always take a state and return a new state
  - The reducer NEVER modifies the received state
  - The returned state becomes the new state of the store[1]
  - Multiple reducers may be combined into the reducer for the store

```
function flipRegistration(originalState) {

   let newState = { ...originalState }[2]

   newState.registered = newState.registered ? false : true
   return newState
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 8-7

Notes

[1] This is simply a performance optimization. All registered observers are notified when the store is replaced. That simplifies trying to notice that a property buried in the store has changed.

[2] The spread operator is used in a literal to create a shallow copy. Deep copies should be avoided, but any property which is changed, or has a property which is changed, must be a new property. More on patterns for doing this later in the chapter.

Action Creators

- Actions are an object
  - With a code defining what action to perform
  - And the data associated with the action
- Action creators build action objects
  - Creators enforce DRY, actions are not built everywhere
  - Creators support Single Responsibility
- Actions often represent asynchronous operations
  - The action should be dispatched when the promise is resolved
  - The operation and the action are tightly bound
  - Consider combining these in the action creator

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Redux Cycle in React

- It is difficult in React for data to move uphill
  - But not impossible – consider callback functions
- Move data in one direction through a cycle
  - Components render the current state in the store
  - User actions or AJAX requests trigger dispatch calls
  - Reducers are called to update the immutable store
  - Components are notified of changes, and render the state
- **react-redux** has methods to optimize the performance

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Redux

Stores and Reducers

Redux Architecture
Stores and Reducers
Components and Action Creators

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Architect with Stores

- Not everything belongs in the store
  - Component state is still important to manage input fields[1]
  - Component state needs to be translated into business objects
- The store is for shared data
  - The store is analogous to the model in MVC
  - The model data is the business data, the state of the application
  - Application state can also define presentation
- The stores expose all the data to all the application
  - Components are at risk to of dependency on extraneous data
- Actions belong to the store, components use actions

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] Consider that the state for the component is mutated when the user types something, and we probably do not want to commit that change into the store until the user clicks a save button.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Redux Store

- The redux store in React is created with **createStore**
  - **createStore** expects the store reducer function
  - **createStore** may receive initialization data

store/Store.js

```
import { createStore } from 'react-redux'

export default createStore(reducer, { events: [] })
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Redux Store

- Using a class for organization

store/Store.js

```
import { createStore } from 'react-redux'

class Store {
   constructor() {
      this._store = createStore(reducer, { events: [] })
   }

   get store() {
      return this._store
   }
}

export default (new Store()).store
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Redux Store

- Expand on the class with initialization data

store/Store.js

```
import { createStore } from 'react-redux'

class Store {
   constructor() {
      this._store = createStore(reducer, this.initialState)
   }
...

   get initialState() {
      return { events: [] }
   }
}

export default (new Store()).store
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Defining Actions

- Actions belong to the store
  - Define the actions in the Store module

store/Store.js

```
import { createStore } from 'react-redux'

export const ES_DELETE_EVENT_ACTION = 'ES_DELETE_ACTION'
export const ES_SET_EVENT_ACTION = 'ES_SET_EVENT_ACTION'
export const ES_SET_EVENTS_ACTION = 'ES_SET_EVENTS_ACTION'

class Store {
...
```

- Actions can be any unique value
  - Using a string helps with debugging

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Reducers

- Separate the reducer into its own class
  - A class so the actions can be separated into methods
  - The reducer receives the current state and an action object

store/EventActionReducer.js

```
import store, { ES_DELETE_EVENT_ACTION,
   ES_SET_EVENT_ACTION, ES_SET_EVENTS_ACTION } from 'store/Store'

class EventActionReducer {
   reduce(state, action) {
      let result = state ? state : store.initialState
      if (action && action.type) {
         switch (action.type) {
            case ES_DELETE_EVENT_ACTION:
               result = this._delete(state, action)
...
      return result
...
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Reducer Methods

- A contract exists between the client and the reducer
  - The reducer expects certain data, the client must provide it

store/EventActionReducer.js

```
class EventActionReducer {
...
    _delete(state, id) {
        let result = { ...state }
        result.events = { ...state.events }
        delete result.events[id]
        return result
    }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Initial State

- The reducer must return some state
  - If no state is provided, it is best to start with the initial state
  - This removes the necessity of setting the state in **createStore**

store/EventActionReducer.js

```
import store, { ES_DELETE_EVENT_ACTION,
   ES_SET_EVENT_ACTION, ES_SET_EVENTS_ACTION } from 'store/Store'

class EventActionReducer {
   reducer(state, action) {
      let result = state ? state : store.initialState
...
      return result
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Shallow and Deep Copies

- Never use a deep copy for new state
  - Never change a value in an old reference in a shallow copy
- The rules: always start with a shallow copy
  - If a property must be changed, replace it
  - If a child property must be changed, replace the parent too
  - If an array element is changed, it must be replaced
  - And the array must be replaced too, but the copy may have references to elements that are not changed
- Add babel-preset-stage2 to the project for destructuring:

```
let result = { ...state }
result.events = { ...state.events }
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

The Reducer and the Store

- Bind the reducer to an instance in the module

store/EventActionReducer.js

```
class EventActionReducer {
   constructor() {
      this.reduce = this.reduce.bind(this)
   }
...
export default (new EventActionReducer()).reduce // export bound reducer
```

- Import the reducer and use it

store/Store.js

```
import { createStore } from 'react-redux'
import eventActionReducer from 'store/EventActionReducer'

class Store {
   constructor() {
      this._store = createStore(eventActionReducer, this.initialState)
   }
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Combining Reducers

- **createStore** requires one reducer
  - Separating reducers promotes separation of concerns
  - Combine the reducers for **createStore**

```
import { createStore, combineReducers } from 'react-redux'
import eventActionReducer from 'store/EventActionReducer'
…
let combinedReducer = combineReducers( {
    events: eventActionReducer,
    …
} )
```

- The logic of the reducer needs to be changed
  - EventActionReducer was producing a state with an events property referencing an object of events
  - Now it needs to produce just the object of events

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 8-21

Notes

Redux

Components and Action Creators

Redux Architecture
Stores and Reducers
Components and Action Creators

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Binding Components

- A component can register with the store
  - Its subscription function will be called for every change

```
let unsubscribe = store.subscribe( () => {

   console.log(store.getState())
} )

// Do things that cause the store to change

unsubscribe() // removes the subscription to the store
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Binding Components

- **connect** is much more efficient
  - **connect** uses mapStateToProps and mapDispatchToProps[1]
- **mapStateToProps**
  - Is called before the component is rendered
  - Receives the state and any other props
  - Turns the state into props to use during rendering
- **connect** returns a wrapped component
  - The wrapped component should be the component used
  - Export the wrapped component, not the component

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] mapDispatchToProps is beyond the scope of this book

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

connect

- **connect** needs the store, but there is no parameter to pass it
- One way is to use **Provider** at the top of the tree
  - This passes the store as a prop to everybody

app/app.jsx

```
import { Provider } from 'react-redux'
import store from 'store/Store'

ReactDOM.render(
   <Provider store={ store }>
      <Router>
         <MainWithRouter dataContext={ eventContext } />
      </Router>
   </Provider>, document.getElementById('content'))
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

connectWith

- For single components
  - Create a static **connectWith** method in the store that injects the store before calling connect to wrap the component class[1]

store/Store.js

```
class Store {
…
   static connectWithStore(store, WrappedComponent, ...args) {
      let ConnectedWrappedComponent =
         connect(...args)(WrappedComponent)
      return function (props) {
         return <ConnectedWrappedComponent {...props} store={store} />
      }
   }
…
}
let connectWithStore = Store.connectWithStore
export { connectWithStore }
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] This pattern will not conflict with Provide even if they both are used in the same application. If you use Provide you don't need this pattern, but just in case.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

mapStateToProps

- mapStateToProps must be a static method
  - It may be called before the wrapped component is even used
  - It translates the state into props for rendering the component
  - The result is an object of props

events/EventListContainer

```
class EventListContainer {
...
    static mapStateToProps(state, ownProps) {
        let evenLists = []
        if (state && state.events) {
            for (let property in state.events) {
                eventList.push(state.events[property])
            }
        }
        return { events: eventList }
    }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Wrap the Component

- Tie it together by exporting the wrapped component

events/EventListContainer

```
class EventListContainer {
...
}

export default
   connect(EventListContainer.mapStateToProps)(EventListContainer)
```

- Or using the **connectWith** pattern

events/EventListContainer

```
import store, { connectWithStore } from 'store/Store'
…
export default
   connectWithStore(store, EventListContainer,
      EventListContainer.mapStateToProps)
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. 8-28

Notes

Action Creators

- When a component needs to change the store
  - Enforce DRY and use an action creator

events/EventListContainer

```
saveEvent() {
    if (this.state.valid) {
        let event = this.buildEventFromState()
        if (event.id) {
            this.eventActionCreator.updateEvent(this.state.id, event)
        } else {
            this.eventActionCreator.insertEvent(event)
        }
        this.context.router.push('/events')
    }
}
```

- Of course the **eventActionCreator** was imported to do this

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Action Creator

- A typical action creator is a function to build an action object

```
function deleteEvent(event) {
  return {
     type: ES_DELETE_ACTION_EVENT,
     id: event.id
  }
}
```

- Remember a contract exists with the reducer
  - The properties in the action object must match the reducer

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Action Creator

- Organize the functions as methods of a class

store/EventActionCreator.js

```
export default class EventActionCreator {
   deleteEvent(id) {
      return {
            type: ES_DELETE_EVENT_ACTION,
            id: id
      }
   }
…
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Action Creator

- So, the client is
  - Responsible for interface with the user
  - Responsible for performing an asynchronous operation
  - Responsible for using an action creator
  - And responsible for dispatching the action
- The last three are related
  - Why not just make the action creator method do them?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Combined Action Creator

- Business logic and store update in one
  - Use the constructor to inject the data context

store/EventActionCreator.js

```
import store, { … } from 'store/Store'

export default class EventActionCreator {
   constructor(dataContext) {
      this.dataContext = dataContext
   }
   deleteEvent(id) {
      this._dataContext.deleteEvent(id)
         .then( () => store.dispatch({
            type: ES_DELETE_EVENT_ACTION,
            id: id
         }))
         .catch( (err) => console.log(err) )
   }
…
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Checkpoint

- Why should we avoid bi-directional data flow?
- What is the problem with cascading state changes?
- How does Redux fix the data flow problems?
- What are the actions? Who uses them?
- What does the action creator module really do?
- What big advantage does the store provide?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Appendix A

react-router 3

Context
Routers and View Changes
Injecting Data Sources

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Objectives

- Address changing application views
- Use the routing framework to control view changes
- Inject data using a context through the router

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

react-router 3

Context

Context

Routers and View Changes

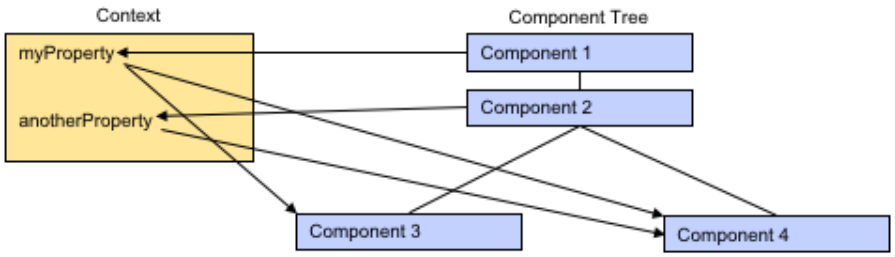
Injecting Data Sources

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Props vs Context

- A child component is rendered by a parent component
  - This forms the tree of components
  - Data can be passed through props
  - Intermediates must pass a prop, but may not be responsible for it
- Context data is available to the tree of child components
  - Context allows multiple values to be passed as a single unit
  - Any component can add to the context
  - Components choose what they want to use



Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Defining Context

- A parent defines the context that any child may receive
  - The **childContextTypes** defines the properties
  - **getChildContext** serves the context properties
  - **getChildContext** is used by React, not called directly

```
class MyParentComponent extends Context {

   getChildContext() {
      return { myProperty: "Hello, World" }
   }
}

MyParentComponent.childContextTypes = { [1]
      myProperty: PropTypes.string
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. A-5

[1] Note that the property types are the same values used for property validation.

Notes

Using Context

- The child uses **contextTypes** to list required properties

```
class MyChildComponent extends Component {

   render() {
      return <div>{ this.context.myProperty }</div>


MyChildComponent.contextTypes = {
      myProperty: PropTypes.object.isRequired
}
```

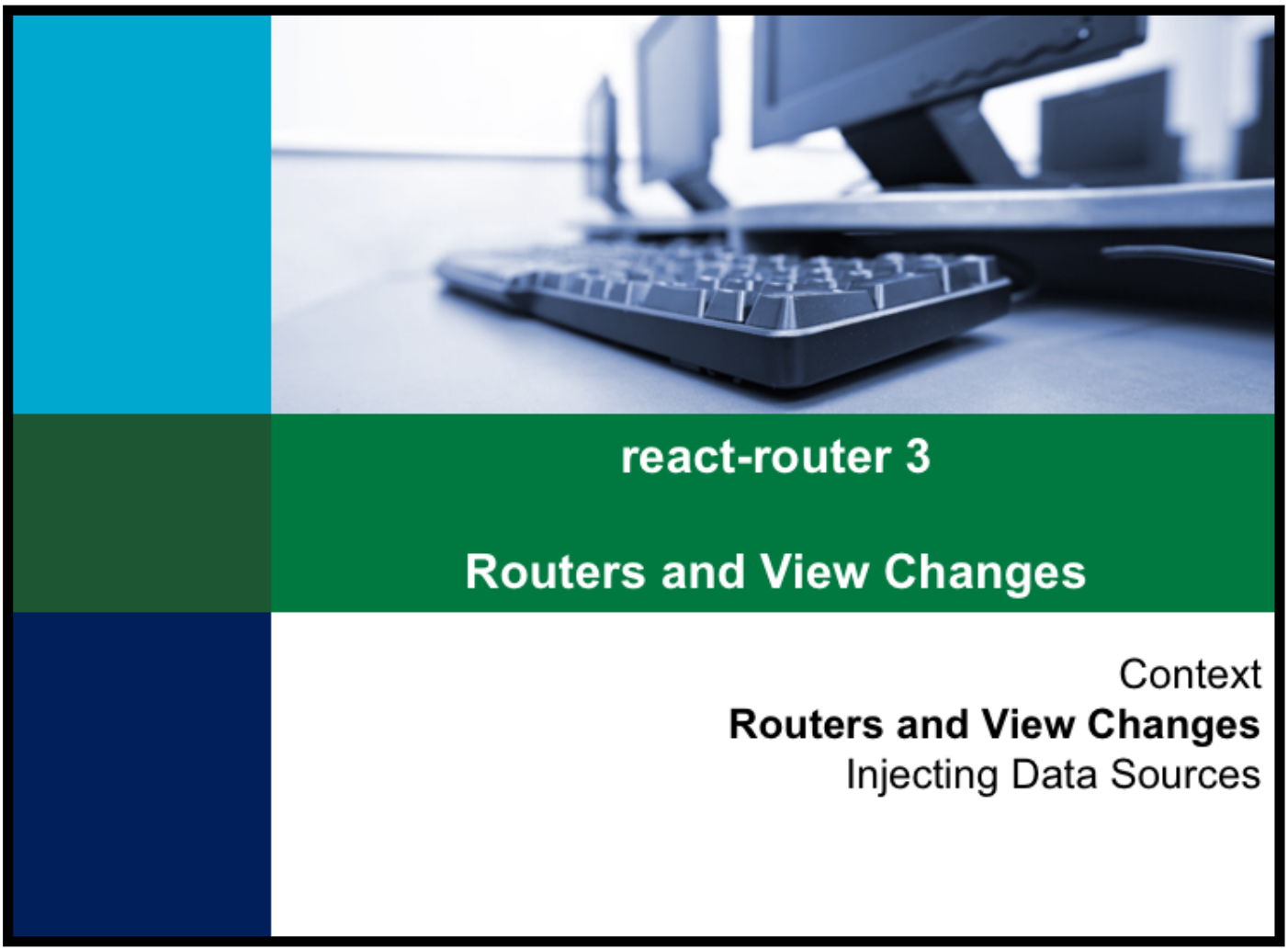
- The context is a mix of all parent contexts[1]
  - Filtered to just include the required properties

- *Avoid context: creates tight coupling between components*
  - *Parents need to provide what the children require*

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. A-6

[1] This is a big problem: context properties are essentially "global," and components can step on each other naming the properties.

Notes



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Application Routing

- Routing controls how the application view changes
  - E.g. moving from a list of events to the data for one event
- React renders from the top down
  - A container needs to decide what will be displayed
  - An action in a child could trigger a callback to change the view[1]
- Routing frameworks simplify this
  - The boilerplate of rendering the view is fixed in the framework
  - react-router configures the routes for the application

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

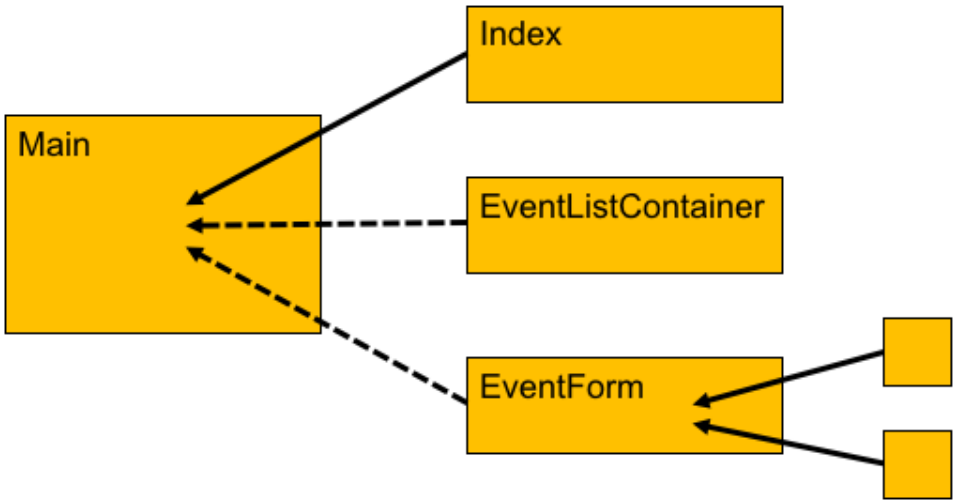
Notes

[1] This violates the premise of React, because we are moving information upstream. We have to create the callback, the child needs to know to use the callback, etc.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

react-router

- **react-router** is built around **groups**
  - A **Route** element wraps a group of view definitions
  - One component is a container for the views in the Route
  - The router will set one of the other components into the container



Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Main

- The Main component is boilerplate
  - The selected view component will be injected as **children**

Main.jsx

```
import Navigation from 'App/Navigation'

export default class Main extends Component {
   render() {
      return (
         <div>
            <div className='header'>
               <div className='logo'>
                  <img src='assets/images/kaleidoscope.png' />
               </div>
            </div>
            { this.props.children }
         </div>
      )
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Routing

- Routing brings scattered view changes into one place
  - Components change the URL, the router changes the view
  - React-router is a popular implementation of this pattern
  - React-router is configured at the top of the tree that uses it

app.jsx

```
import { Router, Route, IndexRoute, browserHistory } from 'react-router'
…

ReactDOM.render(
  <Router history={ browserHistory }>
    <Route path='/' component={ Main }>
      <IndexRoute component={ Index } />
      <Route path='/events' component={ EventListContainer } />
      <Route path='/events/:id' component={ EventForm } />
    </Route>
  </Router>,
  document.getElementById('content'))
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. A-11

- The Router is the top-level container, is rendered as the entry point for the application, and it decides exactly what will be rendered.

Notes

Route Definitions

- A Route describes a view with a URL relative to the group
  - Routes may be nested to create groups
  - The IndexRoute is special, it is the default view for the group

```
<Router history={ browserHistory } >
  <Route path='/' component={ Main }>
    <IndexRoute component={ Index } />
    <Route path='/events' component={ EventListContainer } />
    <Route path='/events/:id' component={ EventForm } />
  </Route>
</Router>
```

- A route may require one or more parameters[1]
  - The parameter id appears as this.props.params.id in EventForm
  - Query parameters appear as fields in this.props.location.query

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] Follow the REST pattern and use a single parameter

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

History

- Defines how URLs look and provides for changing the URL
- hashHistory – uses the hash-mark, works with older browsers
  - http://localhost:8080/#/events
  - http://localhost:8080/events
- browserHistory – HTML5 URL format without #
- memoryHistory – working URL not in the address bar
- hashHistory and browserHistory allow the user to manually alter the address or bookmark it; that could be good or bad

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] The choice of the history component still applies even if the context object is used to change the URL.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Configuring History

- The history object is imported and put in the router tag

```
import { Router, Route, IndexRoute, browserHistory } from 'react-router'

ReactDOM.render(
  <Router history={ browserHistory }> ...
```

- memoryHistory is different
  - needs to be instantiated first

```
import { Router, Route, IndexRoute, createMemoryHistory} from 'react-router'

const history = createMemoryHistory(location)

ReactDOM.render(
  <Router history={ history }> ...
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Changing the URL

- Changing the URL
  - The Router props includes a 'history'
  - The view can be changed by pushing a URL onto the history
  - The Router props need to be propagated down the tree
  - …this.props expands to all the props in the current component

events/EventListContainer.jsx

```
render() {
   return (<EventList title='Events' keyProp='id' events={ this.state.events }
      { ...this.props } />)
}
```

events/EventList.jsx

```
render() {
   let events = this.props.events.map( (event) => <EventItem
key={ event[this.props.keyProp] } { ...this.props } event={ event } /> )
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. A-15

Notes

- …this.props is an ES6 feature, the spread operator expands an array into a list. There is an issue though: router props cannot be expanded into a React DOM element, because they probably do not exist as properties of the DOM element. In React v15.2+ trying to propagate props into a React DOM element will throw an error.

Changing the URL

- In the component that is going to change the URL
  - The history is now available because of …this.props
  - Use it in an event handler to change the URL

events/EventItem.jsx

```
render() {
   return ( <span><input type='checkbox' checked={ this.state.isChecked }
      onChange={ (event) => { this.props.history.pushState(null,
         `/events/${this.props.event.id}`) } } /> 
   ...
}
```

- The change will be picked up immediately by the router

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. A-16

Notes

- This example uses an ES6 template string to fill in the event id and build the URL.

Changing the URL

- Using the history from the props generates several warnings
  - The component is expected to use **context**
  - The router attaches itself to the context for all the children
- The child component declares the context property

events/EventItem.jsx

```
EventItem.contextTypes = {
      router: PropTypes.object.isRequired
}
```

- The property can be used to push the new URL

```
... onChange={ (event) => this.context.router.push(
`/events/${this.props.event.id}`) } ...
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. A-17

Notes

- This is a good use of context: the router expects any component could use the context, and while the components that use the context are dependent on the router it is not unreasonable to depend on that one application-wide facility.

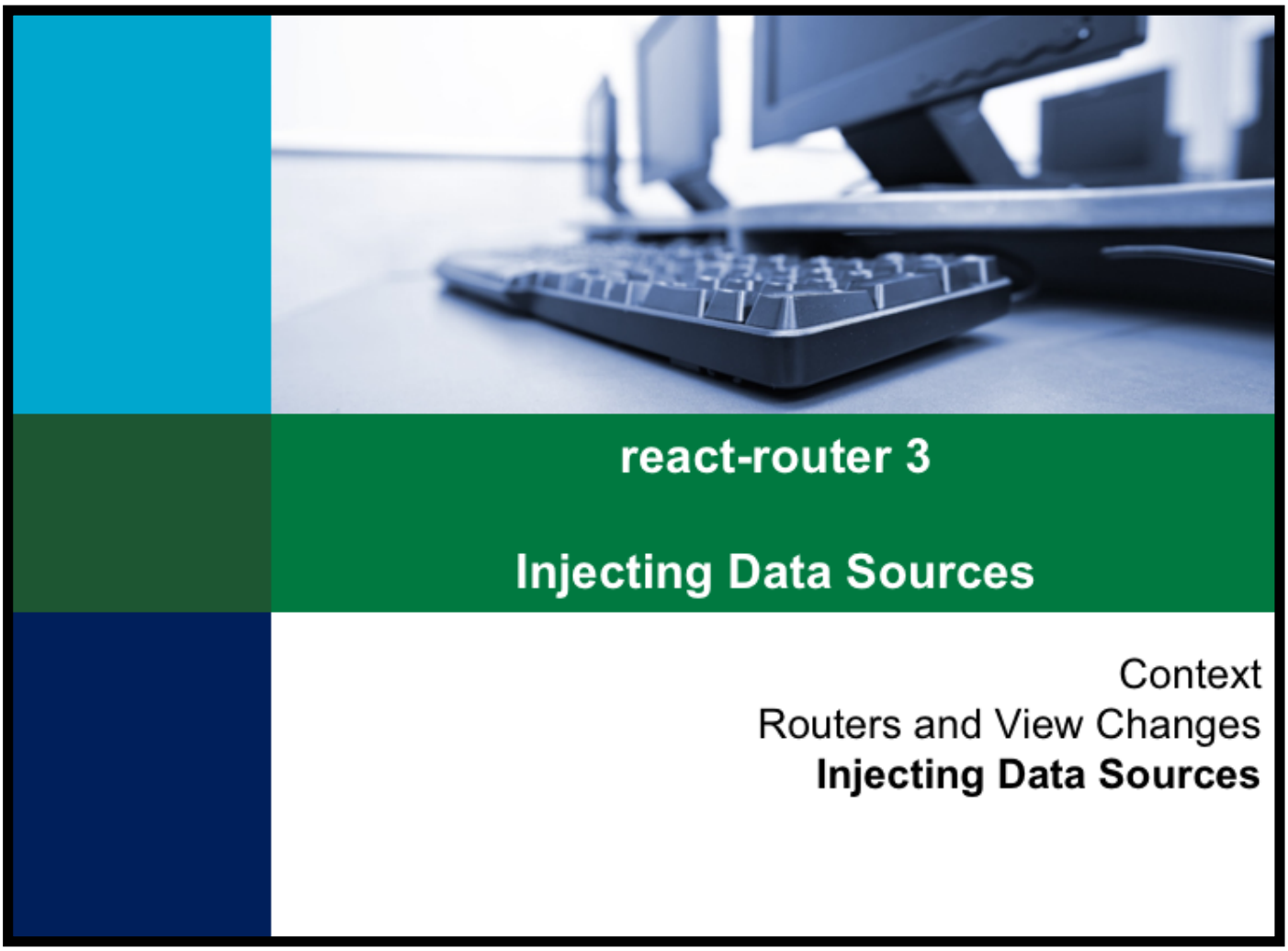
Link

- A **Link** is the equivalent of a static anchor tag
  - But it changes the URL triggering the router
  - <Link to='/events/1' activeClassName='highlight'>Text</Link>
- The text is displayed as the link
- activeClassName is the CSS class used when active
  - The className prop still defines the CSS class for the normal state of the link

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Separation of Concerns

- The EventListContainer and EventForm both work with Events
  - Dates need to be converted from/to ISO date strings
  - Separate the connection with the database into another class

data-access/EventContext.jsx

```
export default class EventContext {
  getEvents() {
    return fetch('/data/events')
      .then( (result) => result.json() )
      .then( (rows) => {
        let result = []
        if (Array.isArray(rows)) {
          result = rows.map( (row) => this.buildEvent(row) )
        }
        return result
      })
  }
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. A-20

Notes

- Note that the method getEvents returns a promise, which is a promise that will complete after the results have been converted from JSON and converted into Events. The buildEvent method will handle converting row data into Event objects.

Dependency Injection

- The components should be given a EventContext
  - If it can be injected by the router then it can be changed[1]
  - Props given to the route become properties of the context

```
let eventContext = new EventContext()

ReactDOM.render(
  <Router history={ browserHistory }>
    <Route path='/' component={ Main }>
      <IndexRoute component={ Index } />
      <Route path='/events' component={ EventListContainer }
        dataContext={ eventContext } />
      <Route path='/events/:id' component={ EventForm }
        dataContext={ eventContext } />
    </Route>
  </Router>,
  document.getElementById('content'))
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

- A different source that behaves the same could be substituted later.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

JSON Conversion

- To write Event objects they need to become JSON strings
  - Not the private fields, only what the ES6 getters return1
  - Accomplish this with a toJSON method in the Event class

```
toJSON() {
   return {
      id: this.id,
      name: this.name,
      date: this.date,
      tz: this.tz,
      venue: this.venue
   }
}
```

  - When an Event is stringified the date will become an ISO string

```
let data = JSON.stringify(event) // { "id": 99, ... }
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. A-22

[1] This.id, etc. are properties of the class. Without a toJSON method, JSON.stringify will get all the fields and properties.

Notes

Getting and Setting Event Data

- The EventForm needs to get and save data
  - The dates need to be translated from and to ISO date strings
  - Methods go from Event objects to state fields and back again
  - The key is in the manipulation of the date using moment

buildStateFromEvent

```
buildStateFromEvent() {
   let date = moment(event.date.getTime()).format('L')
   let time = moment(event.date.getTime()).format('HH:mm')
   return { id: event.id, name: event.name, date: date, time: time, … }[1]
}
```

buildEventFromState

```
buildEventFromState() {
   let formattedDate = moment(Date.parse(this.state.date)).format('YYYY-MM-DD')
   let date = new Date(`${formattedDate}T${this.state.time}${this.state.tz}`)
   return new Event({  id: this.state.id, name: this.state.name, date: date, … }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

[1] This method builds an object that can be merged with the state using setState.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Getting Event Data

- The EventForm needs to load data
  - Do this in the componentDidMount method
  - Do not load anything for new events (the URL has a 'new' id)!

```
componentDidMount() {
   if (this.props.params.id !== 'new') {
      this.props.route.dataContext.getEvent(this.props.params.id)
         .then( (event) => this.setState(this.buildStateFromEvent(event)))
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Saving Event Data

- We need a method the commit button can use to save data
  - The context method is different for inserts and updates

```
saveEvent() {
   if (this.state.valid) {
      let event = this.buildEventFromState()
      if (event.id) {
         this.props.route.dataContext.updateEvent(this.props.params.id, event)
            .then( (event) => this.context.router.push('/events'))
            .catch( (err) => { /* do something with this */ })
      } else {
         this.props.route.dataContext.insertEvent(event)
            .then( (event) => this.context.router.push('/events'))
            .catch( (err) => { /* do something with this */ })
      }
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Checkpoint

- How should data flow in a React application?
- What is the point of "container" components?
- Where will you use lifecycle events?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Appendix A

Flux

Flux Architecture
Flux in React

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

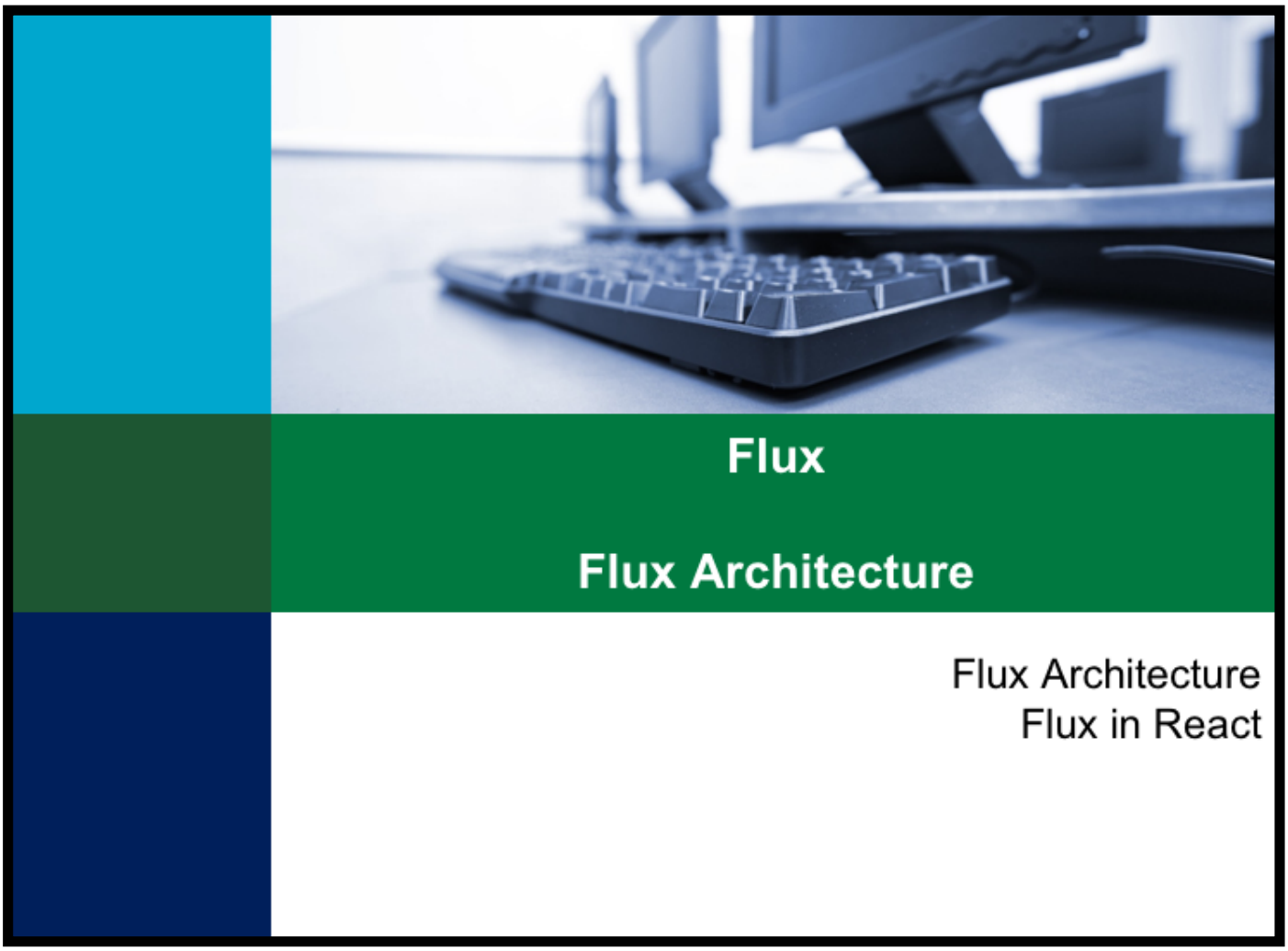
Objectives

- What does the Flux architecture offer?
- Why do applications need to enforce Flux?
- How does Flux work with react?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

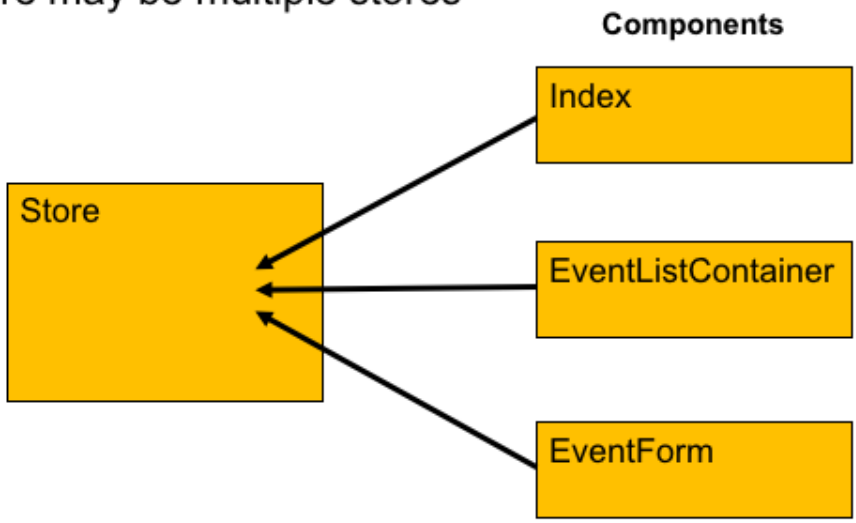


Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Flux Architecture

- The intention of Flux is to move data in one direction
  - The focal point is a **store**
  - Stores hold shared data
  - Components can read a store, but not change it
  - There may be multiple stores



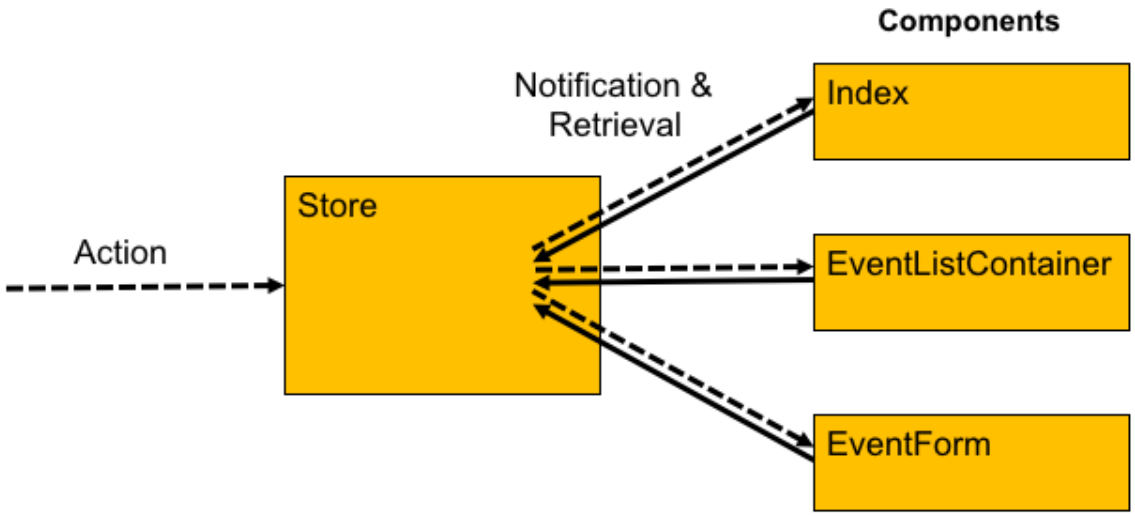
NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. B-4

Notes

- The direction of the arrows show that the components observe the store.

Flux Architecture

- The components *observe* stores
  - **Notifications** occur when the store changes
  - The store changes in response to an **action**

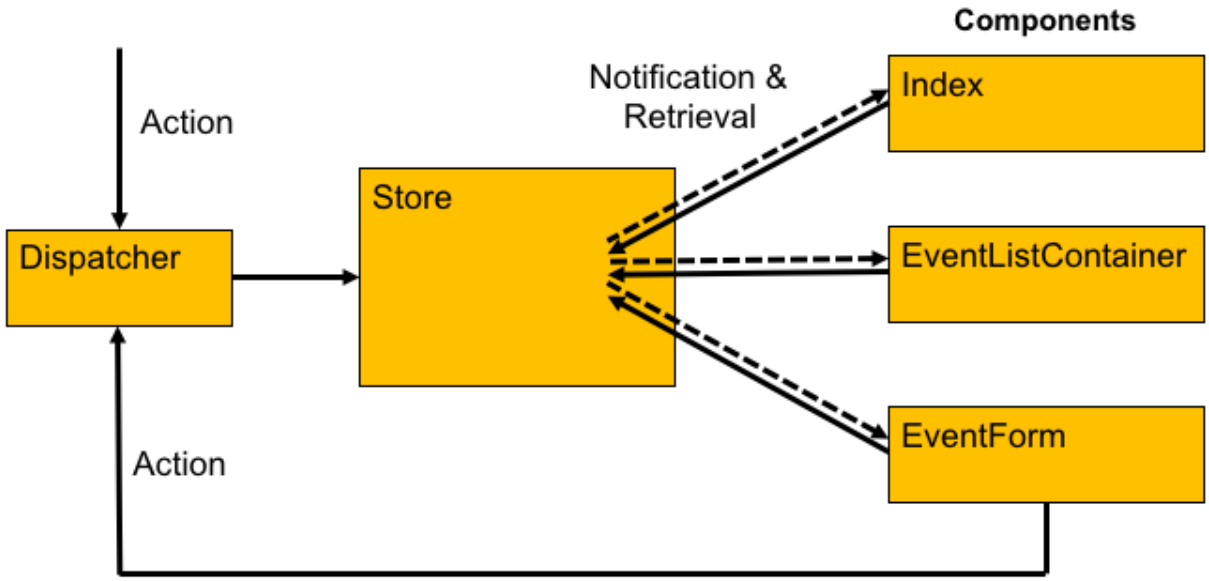


NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. B-5

Notes

Flux Architecture

- **Actions** are invoked by clients and reach the dispatcher
  - The **dispatcher** controls the actions to avoid cascades



NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. B-6

Notes

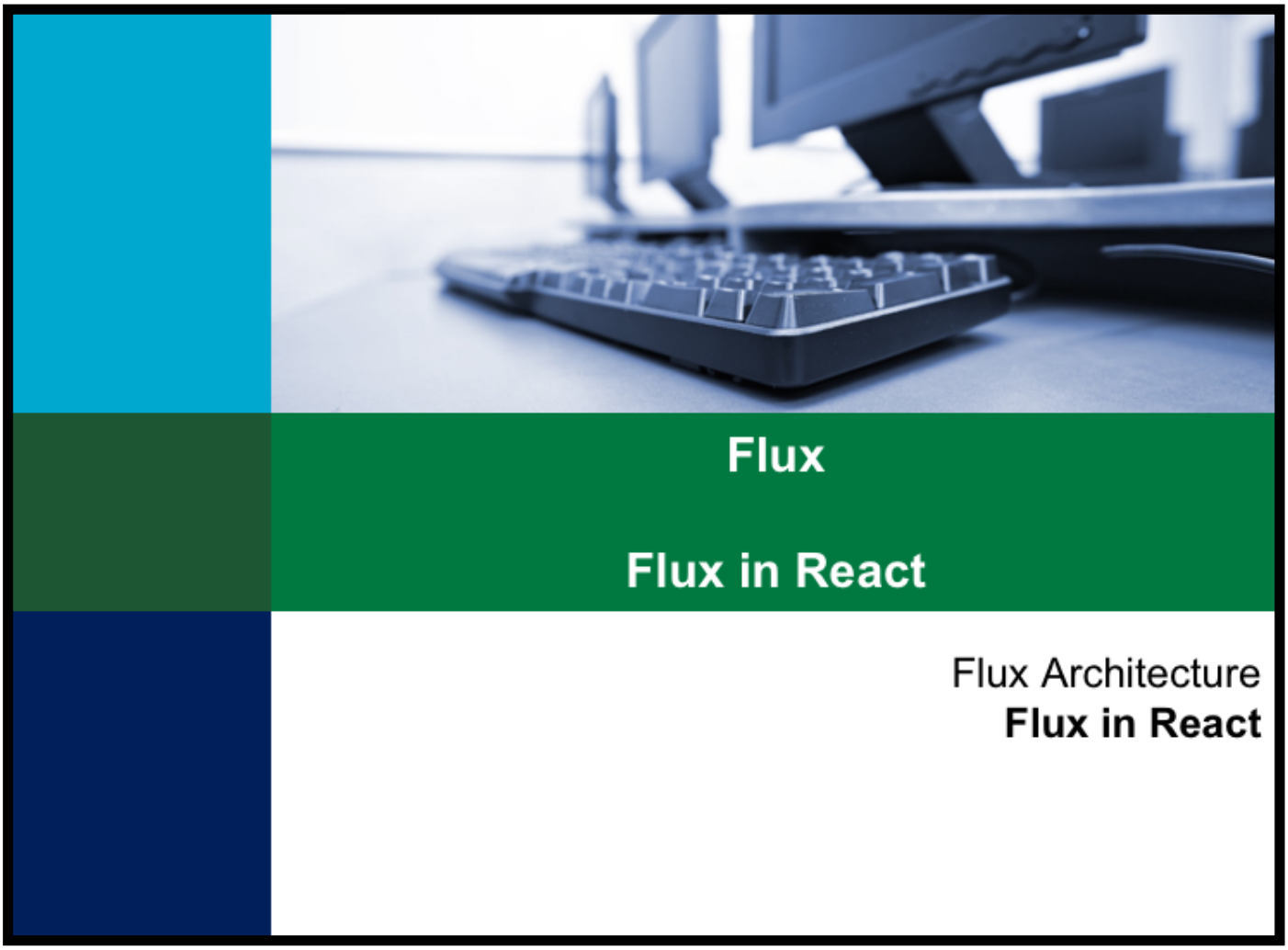
Flux Architecture

- It is difficult in React for data to move uphill
  - But not impossible – consider callback functions
- To support Flux in React
  - Avoid the callbacks – data is moved through actions
- The Flux cycle:
  - The dispatcher coordinates the stores receiving actions
  - The stores see every action – and respond to those of interest
  - The stores notify observers of changes
  - The components update themselves from the stores
  - The components trigger actions, or code that triggers actions

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.



Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Architect with Stores

- Not everything belongs in a store
  - State is still important to manage input fields
  - State still needs to be translated into business objects
- Stores are for shared data
  - Stores are analogous to the model in MVC
  - Stores expose all the data to all the application
  - To preserve immutability stores often return clones
  - Components are at risk to of dependency on extraneous data
- Stores and dependencies must be carefully considered
  - Actions belong to the stores

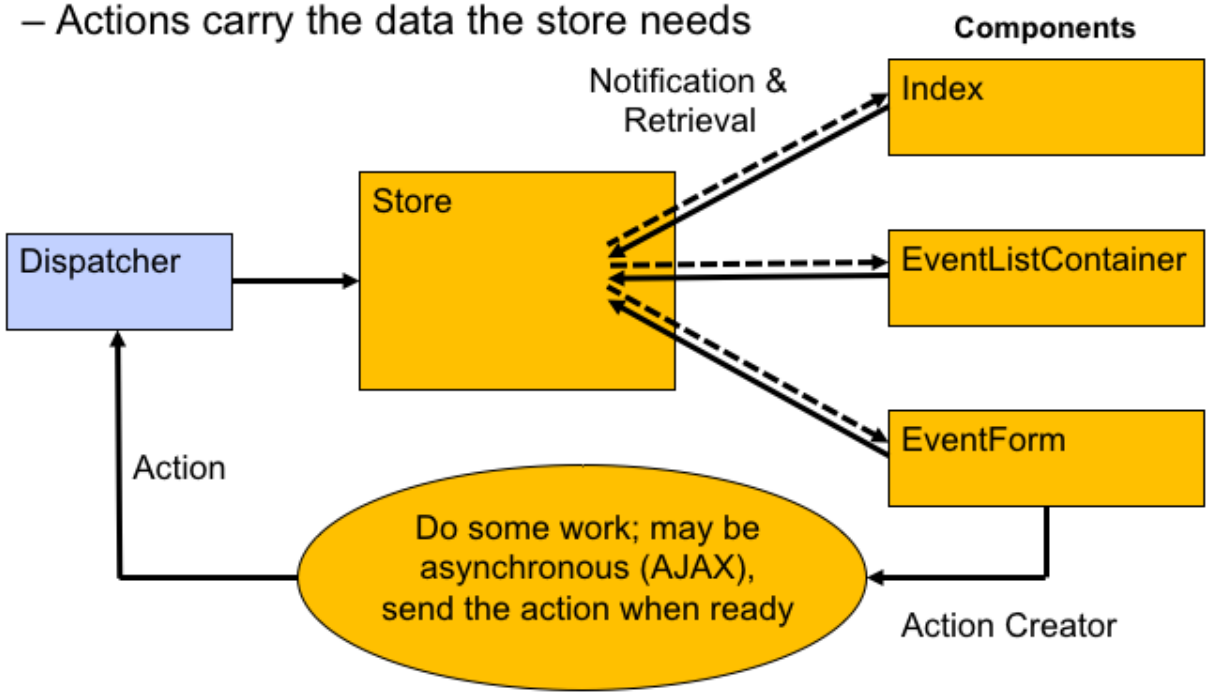
Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Flux-React Architecture

- **Actions** are built by functions invoked by clients
  - When the work is done the action reaches the dispatcher
  - Actions carry the data the store needs



NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. B-10

Notes

Defining Actions

- Define **action** constants between the creator and store
  - These allowed action types belong to the store

events/EventStore.js

```
export const ES_DELETE_EVENT_ACTION = 'ES_DELETE_EVENT_ACTION'
export const ES_SET_EVENT_ACTION = 'ES_SET_EVENT_ACTION'
export const ES_SET_EVENTS_ACTION = 'ES_SET_EVENTS_ACTION'
```

- The events emitted by the store also belong to the store

```
export const ES_CHANGE_EVENT = 'ES_CHANGE_EVENT'
```

- Use descriptive identifiers that will not conflict across stores

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

- Some examples will show the event types defined in a global context external to the store and the action creators. But, the store will only work with actions that it expects, and clients can only expect to be notified about events that it will send. So the actions and events really belong to the store, and that is why they are defined here in the store module.
- Some examples use descriptive strings to name the event, but the only place these strings show up is in the debugger and it is more obvious to simply use the event name as the value.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Actions

- Actions are a protocol between the **action creator** and **store**
  - The creator calls the dispatcher with the action when ready
  - The action reaches the store with data
  - The store notifies all observers of the change
- The action is passed is an object with properties

stores/EventActionCreator.js

```
getEvents() {
   this.dataContext.getEvents()
      .then( (events) => appDispatcher.dispatch( {
            type: ES_SET_EVENTS_ACTION,
            events: events
         }
      })
      .catch( (err) => console.log(err) )
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. B-12

- All of the properties, including type, define the protocol between the creator and the store. There is no formal definition in code, the creator must emit what the store accepts.

Notes

Store

- The store is a context-object that holds data
  - The observers will get the latest copy of the data they want
- Multiple observers could be interested in the same data!
  - One component causes data to be updated
  - Another component gets notified and renders the new data
- In the Kaleidoscope example it needs to hold all the Events[1]
  - It offers methods to get the list or and individual Event
  - It registers with the dispatcher, responds to actions, notifies clients
  - It must be a singleton

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

[1] Yes, it is confusing to identify if what being discussed is an "event" that the store emits or and Event that the application is managing. This is a Kaleidoscope Event.

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Store

- The store needs to bind to the dispatcher
  - And be an emitter itself

```
import { EventEmitter } from 'fbemitter'
import AppDispatcher from 'stores/AppDispatcher'

class EventStore {
   constructor() {
      this.emitter = new EventEmitter()
      this.dispatchAction = this.dispatchAction.bind(this)
      this.dispatchToken = AppDispatcher.register(this.dispatchAction)
      this.store = { }
   }
```

- As an emitter proxy the addListener method

```
addListener(event, callback) {
   return this.emitter.addListener(event, callback)
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Store

- Override dispatchAction to handle the actions

```
dispatchAction(action) {
   switch (action.type) {
      case ES_DELETE_EVENT_ACTION:
         this._deleteEvent(action.id)
         break
      case ES_SET_EVENT_ACTION:
         this._setEvent(action.id, action.event)
         break
      case ES_SET_EVENTS_ACTION:
         this._setEvents(action.events)
         break
      default:
         break
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Store

- In the methods modify the store and notify listeners

```
_deleteEvent(id) {
   delete this.store[id]
   this.emitter.emit(ES_CHANGE_EVENT)
}

_setEvent(id, event) {
  this.store[id] = event
  this.emitter.emit(ES_CHANGE_EVENT)
}

_setEvents(events) {
   this.store = { }
   events.forEach( (event) => this.store[event.id] = event )
   this.emitter.emit(ES_CHANGE_EVENT)
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Store

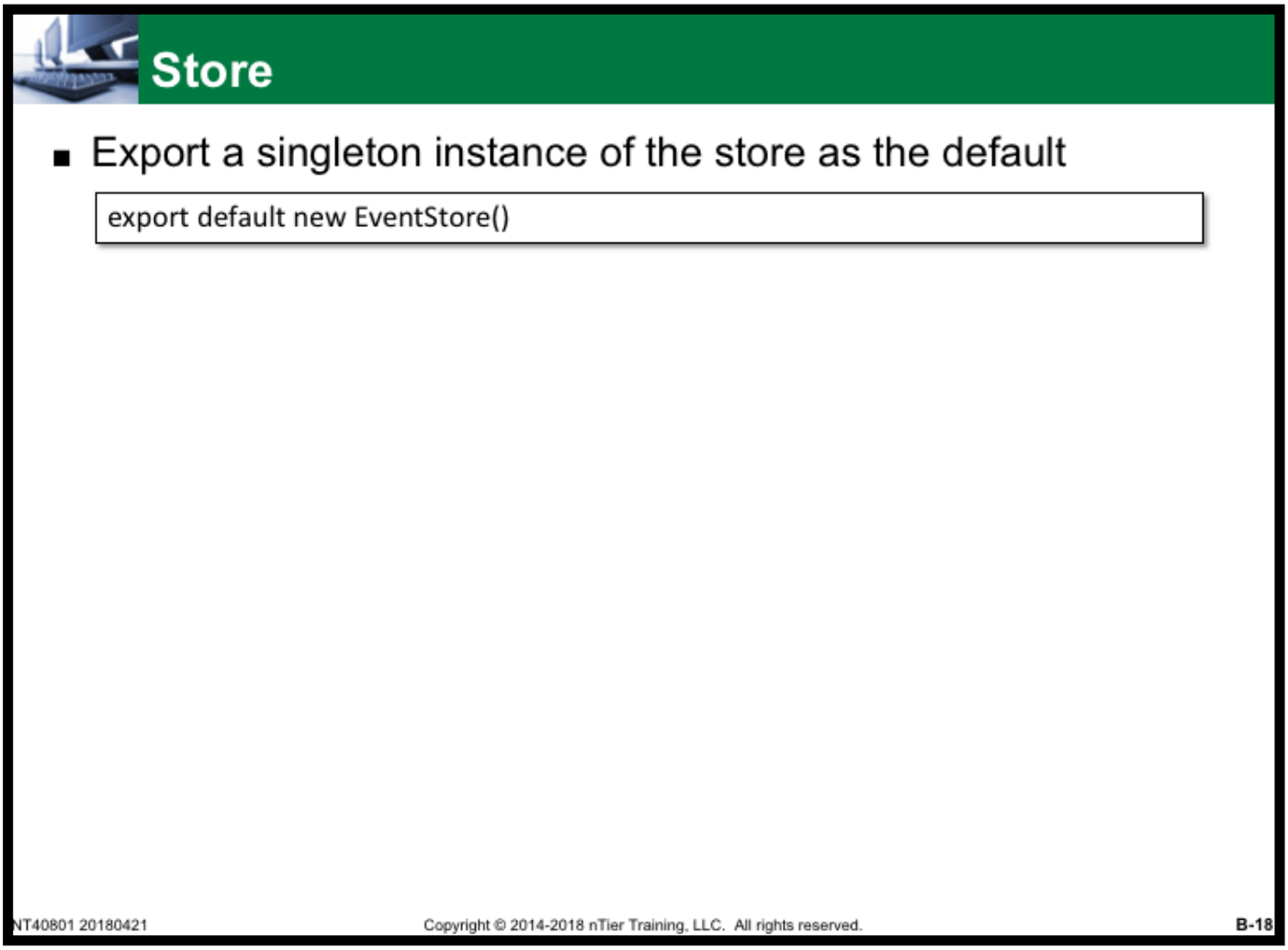
- Add methods for clients to get an event or the list of events

```
getEvent(id) {
   let result = null
   if (this.store[id]) {
      result = new Event(this.store[id])
   }
   return result
}

getEvents() {
   let result = []
   for (let id in this.store) {
      result.push(new Event(this.store[id]))
   }
   return result
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.


Store
Export a singleton instance of the store as the default
export default new EventStore()
NT40801 20180421
Copyright © 2014-2018 nTier Training, LLC. All rights reserved.
B-18

Notes

Action Creator

- Build a singleton EventActionCreator that creates and dispatches actions
  - The methods do something, then pass an action
  - In Kaleidoscope the "something" is an AJAX request, so the action creator is the consumer of the EventContext
- The components will inject the dataContext

```
import appDispatcher from 'stores/AppDispatcher'
import eventStore, { ES_DELETE_EVENT_ACTION, ES_SET_EVENT_ACTION,
ES_SET_EVENTS_ACTION } from 'stores/EventStore'

export default class EventActionCreator {
   constructor(dataContext) {
      this.dataContext = dataContext
   }
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. B-19

Notes

Action Creator

- The rest of EventActionCreator accepts client requests
  - After doing the work sends an action to the dispatcher

```
deleteEvent(id) {
   this.dataContext.deleteEvent(id)
      .then( () => appDispatcher.dispatch({
            type: ES_DELETE_EVENT_ACTION,
            id: id
         }) )
      .catch( (err) => console.log(err) )
}
```

The source is continued on the next page

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Action Creator

```
getEvent(id) {
   this.dataContext.getEvent(id)
      .then( (event) => appDispatcher.dispatch({
            type: ES_SET_EVENT_ACTION,
            event: event
         }) )
      .catch( (err) => console.log(err) )
}

getEvents() {
   this.dataContext.getEvents()
      .then( (events) => appDispatcher.dispatch({
            type: ES_SET_EVENTS_ACTION,
            events: events
         }) )
      .catch( (err) => console.log(err) )
}
```

The source is continued on the next page

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Action Creator

```
insertEvent(event) {
   this.dataContext.insertEvent(event)
      .then( (event) => appDispatcher.dispatch({
            type: ES_SET_EVENT_ACTION,
            event: event
         }) )
      .catch( (err) => console.log(err) )
}

updateEvent(id, event) {
   this.dataContext.updateEvent(id, event)
      .then( (event) => appDispatcher.dispatch({
            type: ES_SET_EVENT_ACTION,
            event: event
         }) )
      .catch( (err) => console.log(err) )
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Dispatcher

- Technically the Flux dispatcher is sufficient
  - But extend it AppDispatcher to log all the actions dispatched
  - The *dispatcher* must be a singleton

```
import { Dispatcher } from 'flux'

class AppDispatcher extends Dispatcher {
   dispatch( action = {} ) {
      console.log('Dispatched:', action)
      super.dispatch( action)
   }
}

export default new AppDispatcher()
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Link the Components

- Make the components observers
  - The invoke an action through the action creator
  - The action creator needs the data context, pass it through
  - Get notified when the store changes and render changes
  - Use **componentDidMount** and **componentWillUnmount**

events/EventListContainer.jsx

```
constructor(...args) {
   super(...args)
   this.state = { events: [ ] }
   this.eventActionCreator = new EventActionCreator(this.props.route.dataContext)
}
```

The source is continued on the next page

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

- This replaces the componentDidMount method that used the dataContext to make an AJAX request. Now the EventActionCreator is passed the dataContext to use, and it delegates the AJAX requests.

 Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Link the Components

```
componentDidMount() {
   this.storeSubscription = eventStore.addListener(ES_CHANGE_EVENT, () =>
      this.setState({ events: eventStore.getEvents() }) )

   this.eventActionCreator.getEvents()
}

componentWillUnmount() {
   if (this.storeSubscription) {
         this.storeSubscription.remove()
   }
}
```

NT40801 20180421 Copyright © 2014-2018 nTier Training, LLC. All rights reserved. B-25

- This replaces the componentDidMount method that called the dataContext.

Notes

Link the Components

- In EventForm the request needs to be moved from render()
  - Into **componentDidMount**

events/EventForm.jsx

```
componentDidMount() {
   // Launch a request for the record, it will populate the state when we get it.
   if (this.props.params.id !== 'new') {
      this.storeSubscription = eventStore.addListener(ES_CHANGE_EVENT, () =>
         this.setState(this.buildStateFromEvent(
            eventStore.getEvent(this.props.params.id))) )
      this.eventActionCreator.getEvent(this.props.params.id)
   }
}

componentWillUnmount() {
   if (this.storeSubscription) {
         this.storeSubscription.remove()
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Link the Components

- In EventForm **saveEvent** needs to use EventActionCreator
  - The notification from the store is simply ignored

```
saveEvent() {
   if (this.state.valid) {
      let event = this.buildEventFromState()
      if (event.id) {
         this.eventActionCreator.updateEvent(this.props.params.id, event)
      } else {
         this.eventActionCreator.insertEvent(event)
      }
      this.context.router.push('/events')
   }
}
```


Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Checkpoint

- Why should we avoid bi-directional data flow?
- What is the problem with cascading state changes?
- How does Flux fix the data flow problems?
- What are the actions? Who uses them?
- What does the action creator module really do?
- What big advantage does the store provide?

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.

Notes

Copyright © 2014-2018 nTier Training, LLC. All rights reserved.