Instructor Notes:

Add instructor notes here.

LEARNING FACEBOOK'S React.JS

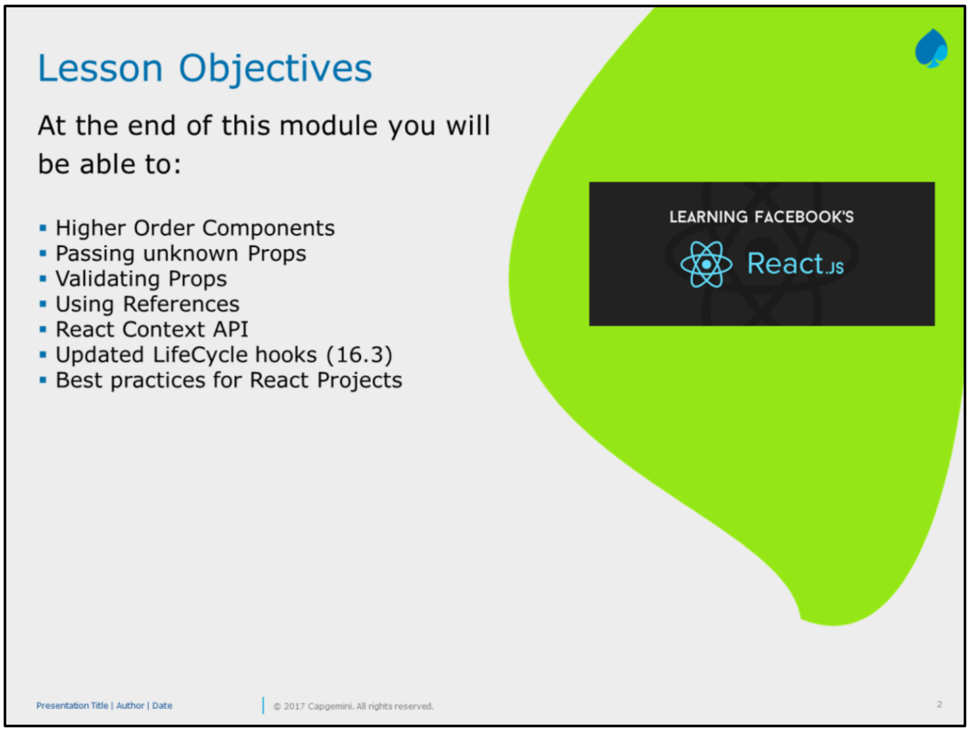
React Component in details

Lesson 07

Capgemini

Instructor Notes:

Add instructor notes here.



Lesson Objectives

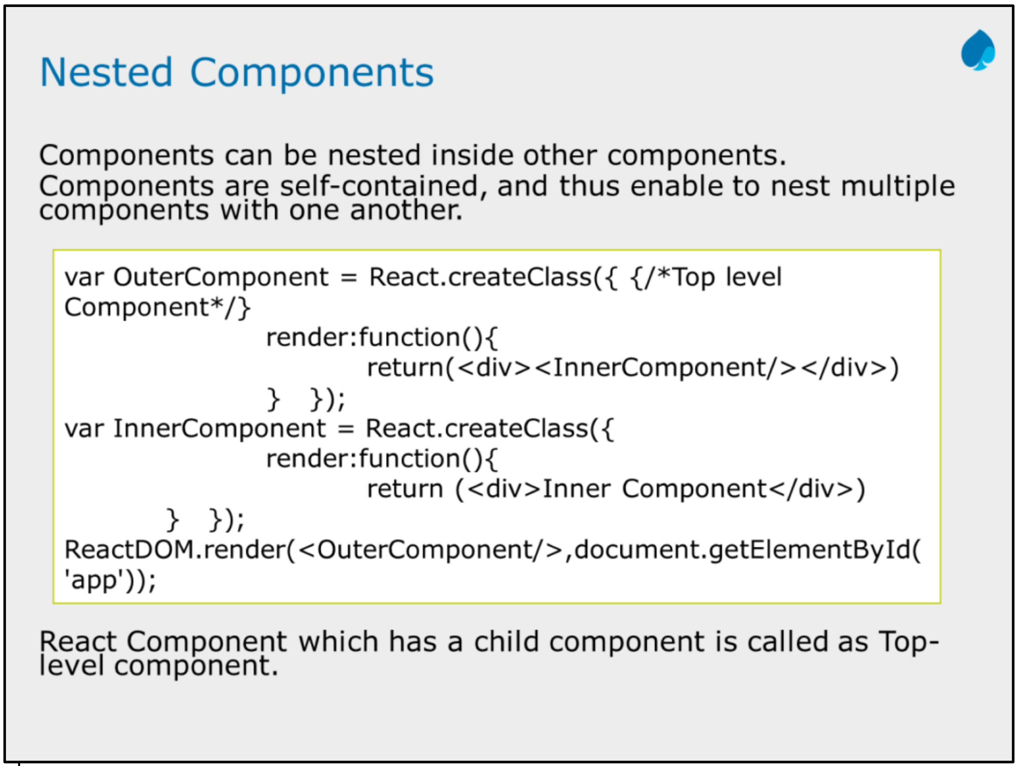
At the end of this module you will be able to:

- Higher Order Components
- Passing unknown Props
- Validating Props
- Using References
- React Context API
- Updated LifeCycle hooks (16.3)
- Best practices for React Projects

LEARNING FACEBOOK'S
React.JS

Instructor Notes:

Add instructor notes here.



Nested Components

Components can be nested inside other components.
Components are self-contained, and thus enable to nest multiple components with one another.

```
var OuterComponent = React.createClass({ {/*Top level
Component*/}
                    render:function(){
                              return(<div><InnerComponent/></div>)
                    }  });
var InnerComponent = React.createClass({
                    render:function(){
                              return (<div>Inner Component</div>)
          }  });
ReactDOM.render(<OuterComponent/>,document.getElementById(
'app'));
```

React Component which has a child component is called as Top-level component.

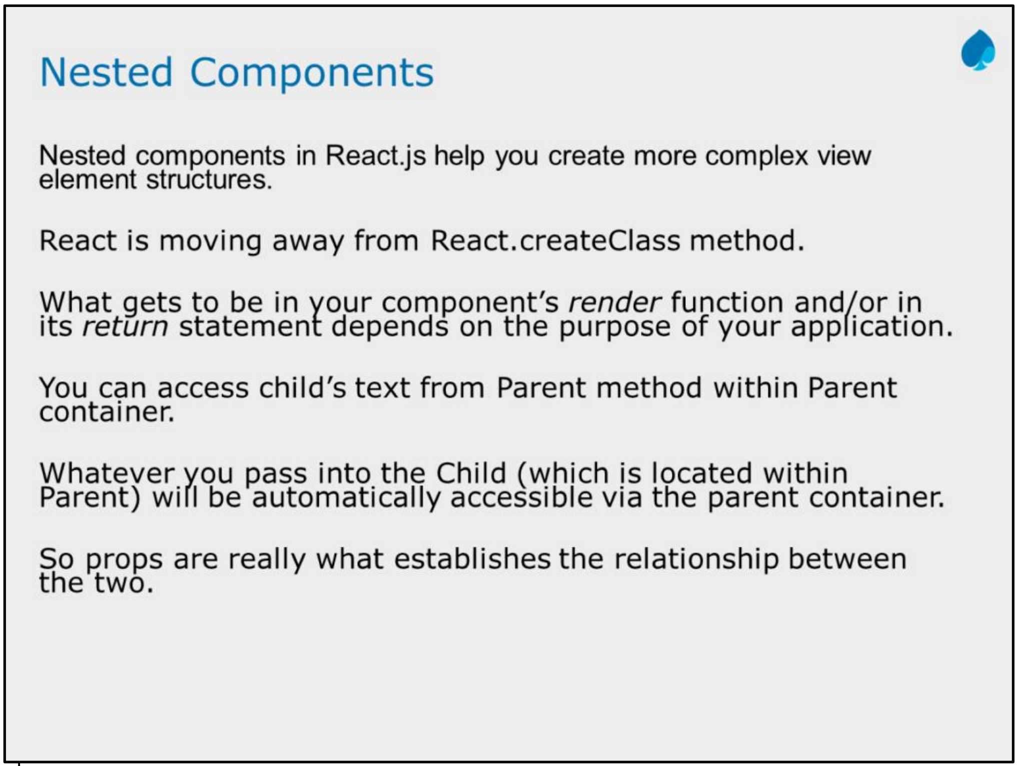
Nested components in React.js help you create more complex view element structures.

React is moving away from React.createClass method.

Top level components can also be called as Controller View, because it controls the data flow for all its child component by setting props on children.

Instructor Notes:

Add instructor notes here.



Nested Components

Nested components in React.js help you create more complex view element structures.

React is moving away from React.createClass method.

What gets to be in your component's *render* function and/or in its *return* statement depends on the purpose of your application.

You can access child's text from Parent method within Parent container.

Whatever you pass into the Child (which is located within Parent) will be automatically accessible via the parent container.

So props are really what establishes the relationship between the two.

Nested components in React.js help you create more complex view element structures.

React is moving away from React.createClass method.

Top level components can also be called as Controller View, because it controls the data flow for all its child component by setting props on children.

Instructor Notes:

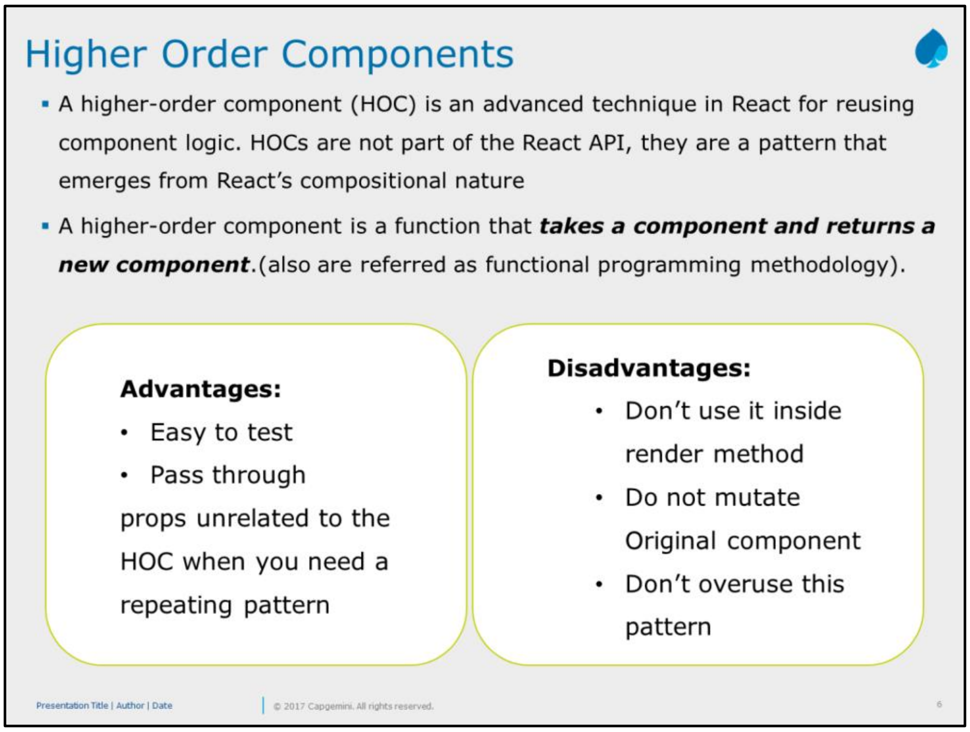
Add instructor notes here.

Demo

Nested-Components

Instructor Notes:

Slide explains that communication between service provider and consumer happen via SOAP messages



Higher Order Components

- A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API, they are a pattern that emerges from React's compositional nature
- A higher-order component is a function that ***takes a component and returns a new component***.(also are referred as functional programming methodology).

Advantages:

- Easy to test
- Pass through

props unrelated to the HOC when you need a repeating pattern

Disadvantages:

- Don't use it inside render method
- Do not mutate Original component
- Don't overuse this pattern

Presentation Title | Author | Date

6

HIGH ORDER COMPONENTS:

Helps to write out business logic

The goal of this HOC pattern is to decompose the logic into simpler and smaller functions that can be reused.

A rule of thumb is a function does just one task and does it well.

HOC makes debugging and maintenance a whole lot easier as they are modular.

Instructor Notes:

Add instructor notes here.

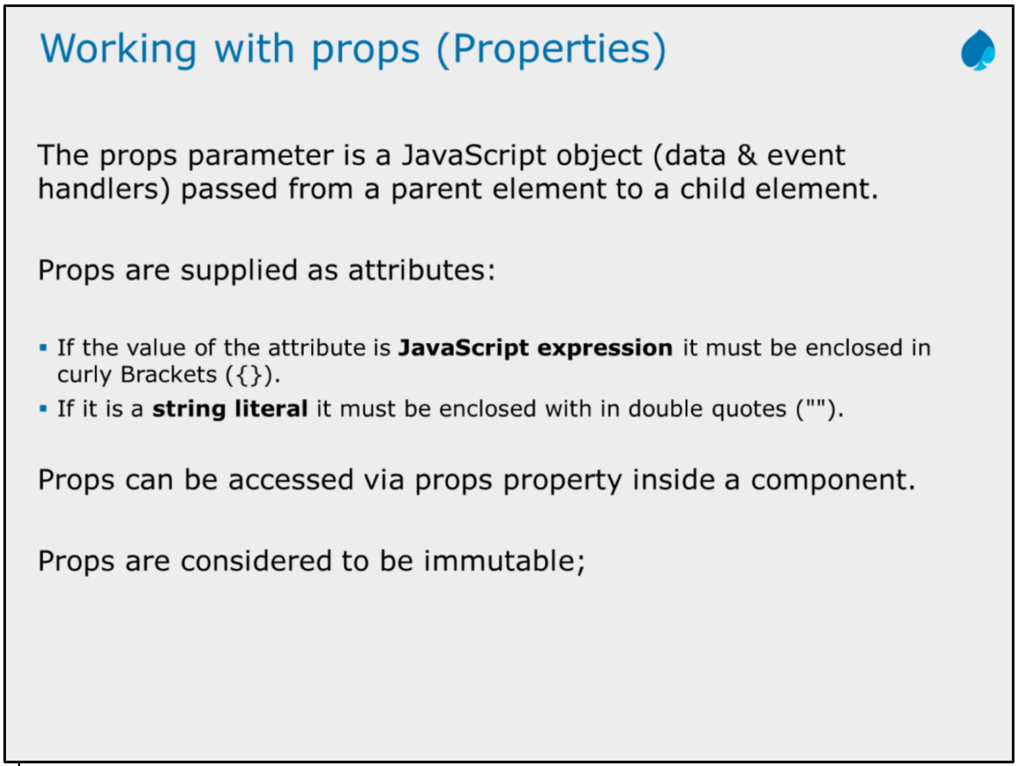
Demo

Higher order components

React-higherorder-2019

Instructor Notes:

Add instructor notes here.



Working with props (Properties)

The props parameter is a JavaScript object (data & event handlers) passed from a parent element to a child element.

Props are supplied as attributes:

- If the value of the attribute is **JavaScript expression** it must be enclosed in curly Brackets ({}).
- If it is a **string literal** it must be enclosed with in double quotes ("").

Props can be accessed via props property inside a component.

Props are considered to be immutable;

```
// Props supplied as attributes
<Sample math={Math.pow(2,3)} name="Veena Deshpande" />

// Access Props via the props property
this.props.math, this.props.name

//SampleComponent has text "Karthik" has its children which can be  accessed via //this.props.children in the component
<SampleComponent>Karthik</SampleComponent>
```

The props parameter is a JavaScript object (data & event handlers) passed from a parent element to a child element.
Props are supplied as attributes.

If the value of the attribute is **JavaScript expression** it must be enclosed in curly Brackets ({}).
If it is a **string literal** it must be enclosed with in double quotes ("").

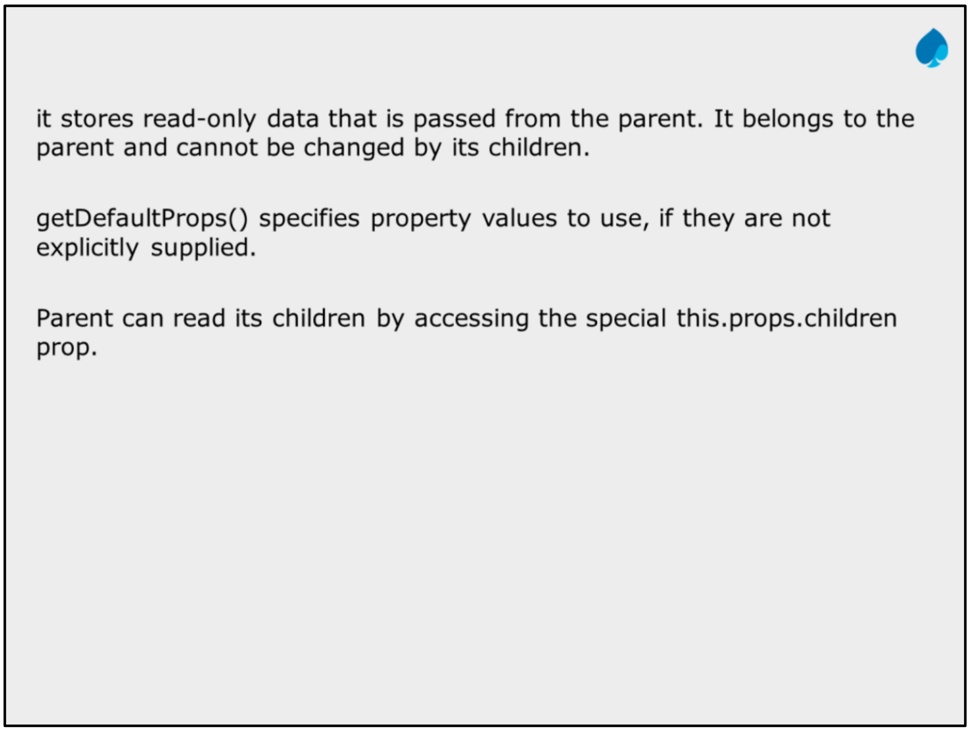
Props can be accessed via props property inside a component.
Props are considered to be immutable; it stores read-only data that is passed from the parent. It belongs to the parent and cannot be changed by its children.

getDefaultProps() specifies property values to use, if they are not explicitly supplied.

Parent can read its children by accessing the special this.props.children prop.

Note : ***this.props.children*** is an opaque data structure, use the React.Children utilities to manipulate them.

Instructor Notes:



it stores read-only data that is passed from the parent. It belongs to the parent and cannot be changed by its children.

getDefaultProps() specifies property values to use, if they are not explicitly supplied.

Parent can read its children by accessing the special this.props.children prop.

Instructor Notes:

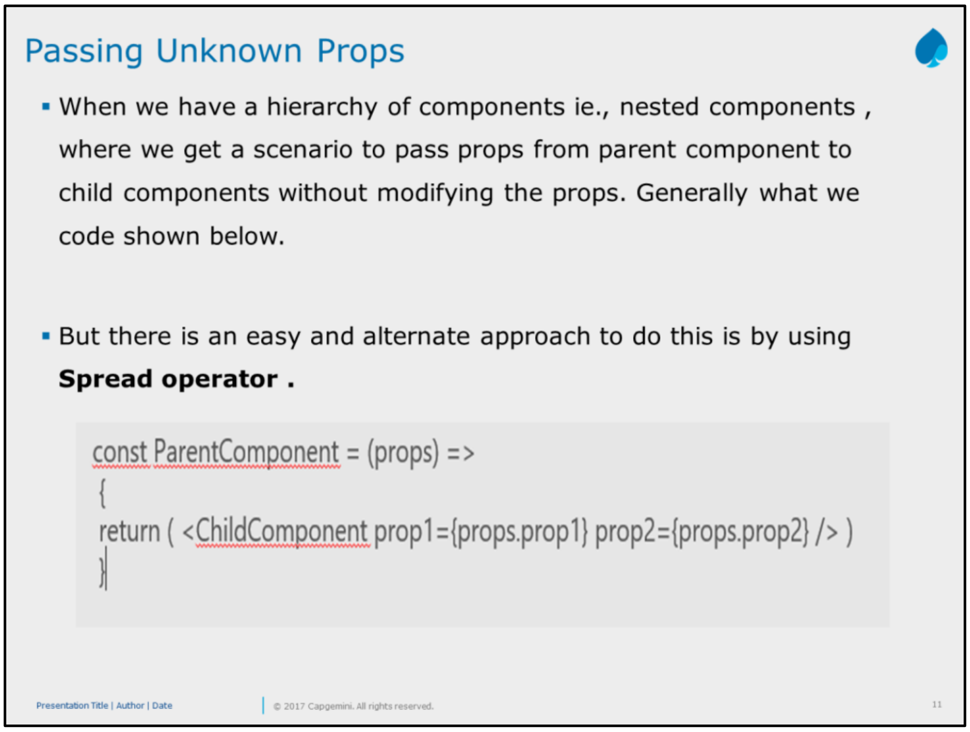
Add instructor notes here.

Demo

Using-Props

Create-React-props

Instructor Notes:



Passing Unknown Props

- When we have a hierarchy of components ie., nested components , where we get a scenario to pass props from parent component to child components without modifying the props. Generally what we code shown below.
- But there is an easy and alternate approach to do this is by using **Spread operator .**

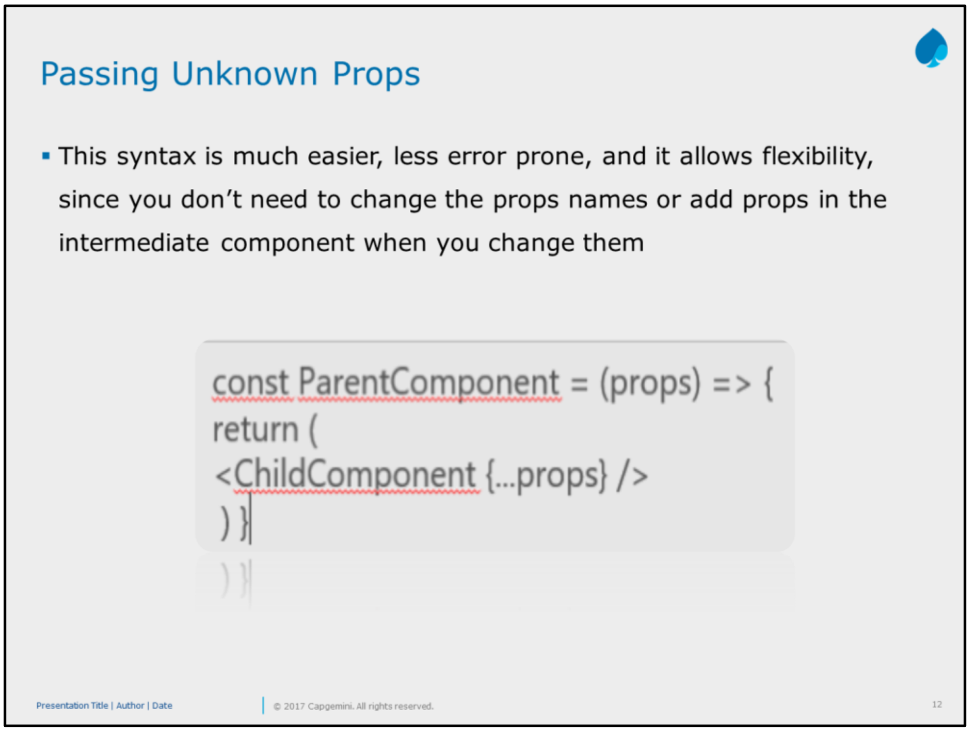
```
const ParentComponent = (props) =>
{
return ( <ChildComponent prop1={props.prop1} prop2={props.prop2} /> )
}
```

```
const ParentComponent = (props) => {
return (
<ChildComponent {...props} />
 ) }
```

Advanced usage of transferring props

```
=========================
const withCounter = Component =>
class Hoc extends React.Component {
constructor(props) {
super(props);
this.state = { count: 0 };
}
update = type => {
if (type === "Inc") {
this.setState(({ count }) => ({ count: count + 1 }));
} else if (type === "Dec") {
this.setState(({ count }) => ({ count: count - 1 }));
}
};
render() {
return <Component
{...this.state}
{...this.props}
update={this.update}
/>;
}
};
```

Instructor Notes:



Passing Unknown Props

- This syntax is much easier, less error prone, and it allows flexibility, since you don't need to change the props names or add props in the intermediate component when you change them

```
const ParentComponent = (props) => {
return (
<ChildComponent {...props} />
) }
```

Presentation Title | Author | Date

© 2017 Capgemini. All rights reserved.

12

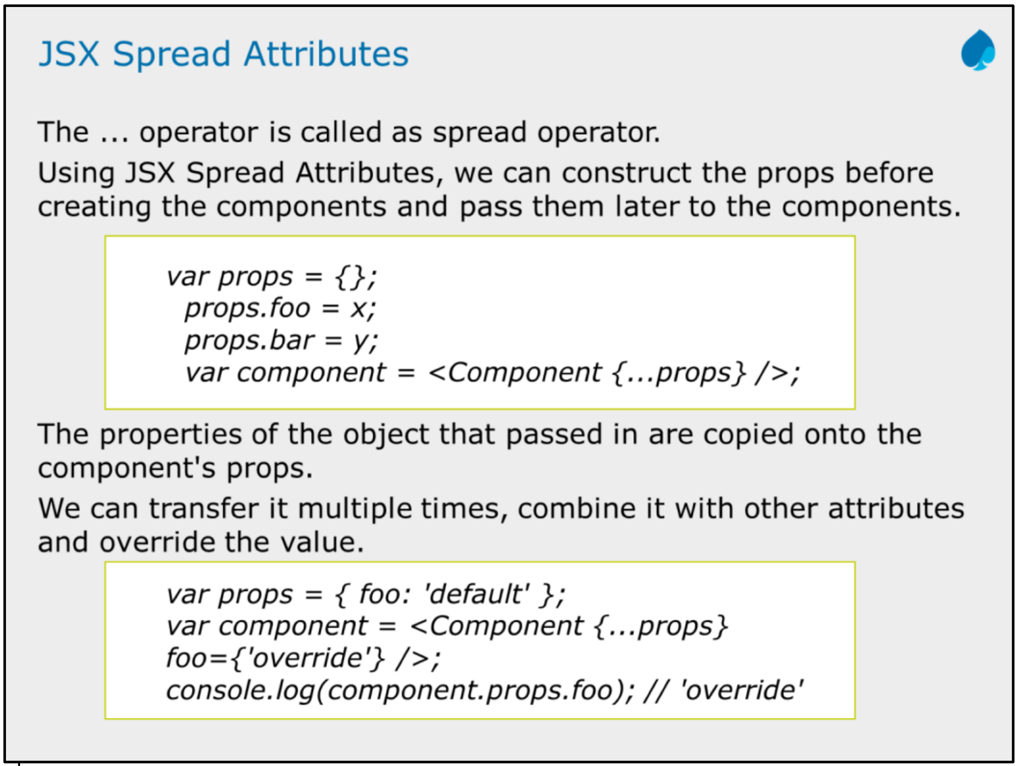
```
const ParentComponent = (props) => {
return (
<ChildComponent {...props} />
 ) }
```

Advanced usage of transferring props

```
==========================
const withCounter = Component =>
class Hoc extends React.Component {
constructor(props) {
super(props);
this.state = { count: 0 };
}
update = type => {
if (type === "Inc") {
this.setState(({ count }) => ({ count: count + 1 }));
} else if (type === "Dec") {
this.setState(({ count }) => ({ count: count - 1 }));
}
};
render() {
return <Component
{...this.state}
{...this.props}
update={this.update}
/>;
}
};
```

Instructor Notes:

Add instructor notes here.



JSX Spread Attributes

The … operator is called as spread operator.

Using JSX Spread Attributes, we can construct the props before creating the components and pass them later to the components.

```
var props = {};
  props.foo = x;
  props.bar = y;
  var component = <Component {...props} />;
```

The properties of the object that passed in are copied onto the component's props.

We can transfer it multiple times, combine it with other attributes and override the value.

```
var props = { foo: 'default' };
var component = <Component {...props}
foo={'override'} />;
console.log(component.props.foo); // 'override'
```

Instructor Notes:

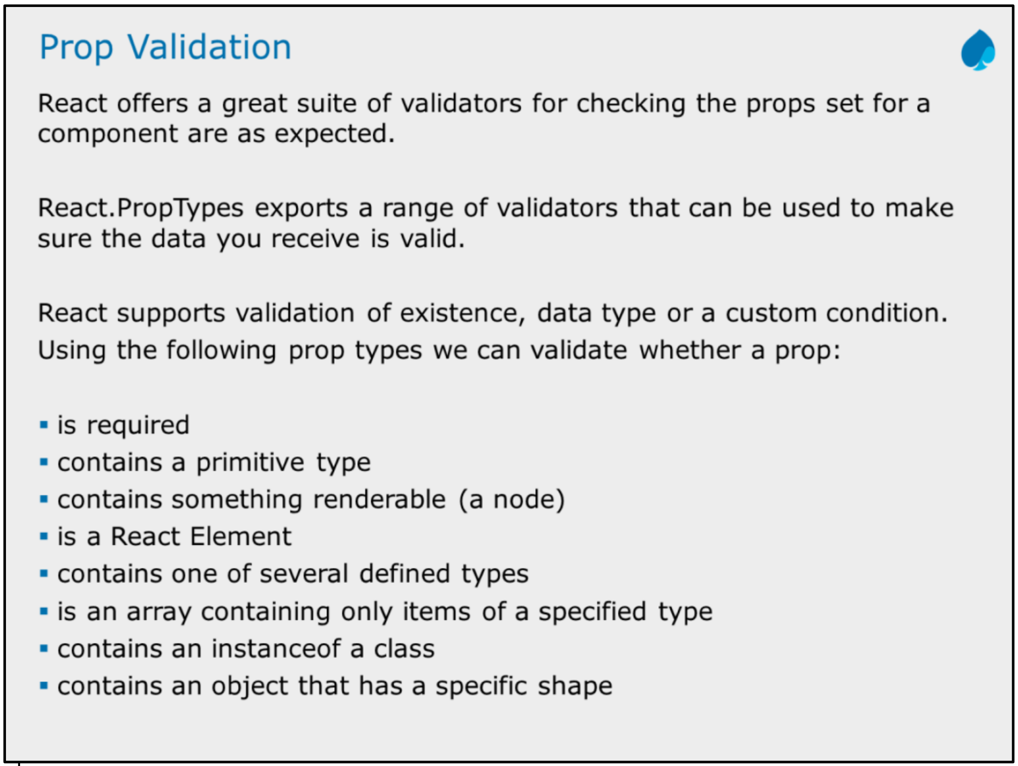
Add instructor notes here.

Demo

React-Spread-attr-2019

Instructor Notes:

Add instructor notes here.



Prop Validation

React offers a great suite of validators for checking the props set for a component are as expected.

React.PropTypes exports a range of validators that can be used to make sure the data you receive is valid.

React supports validation of existence, data type or a custom condition. Using the following prop types we can validate whether a prop:

- is required
- contains a primitive type
- contains something renderable (a node)
- is a React Element
- contains one of several defined types
- is an array containing only items of a specified type
- contains an instanceof a class
- contains an object that has a specific shape

Prop validations helps us to :

1. **Immediately see what data a component can process**
   propTypes can serve as a sort of mini-reference to your back-end's API by just looking at the code of the component. This eliminates needing to switch between looking at the API documentation and your component code.

2. **Get console warnings if a component receives an incorrect or missing data type**
   If a prop is missing, or has an incorrect data type, you'll see a warning in the JavaScript console. React will only check the propTypes in development mode.

3. **Check whether API Data is Changed**
   It is often the case as a project grows, that the structure of a back-end API response could change, and therefore break an element in the UI if that piece of data is missing, or if a new property is added. Having propTypes can eliminate a whole swath of these kinds of errors. If a new property is added which is not defined in proptype, the console would warn to re-examine the data we're getting from this.props, and update our prop checks accordingly.

4. **Ensure strong type checking**
   Enforcing types in JS is tricky business, but with the proper use of propTypes can really minimize this. prop checks can drastically improve

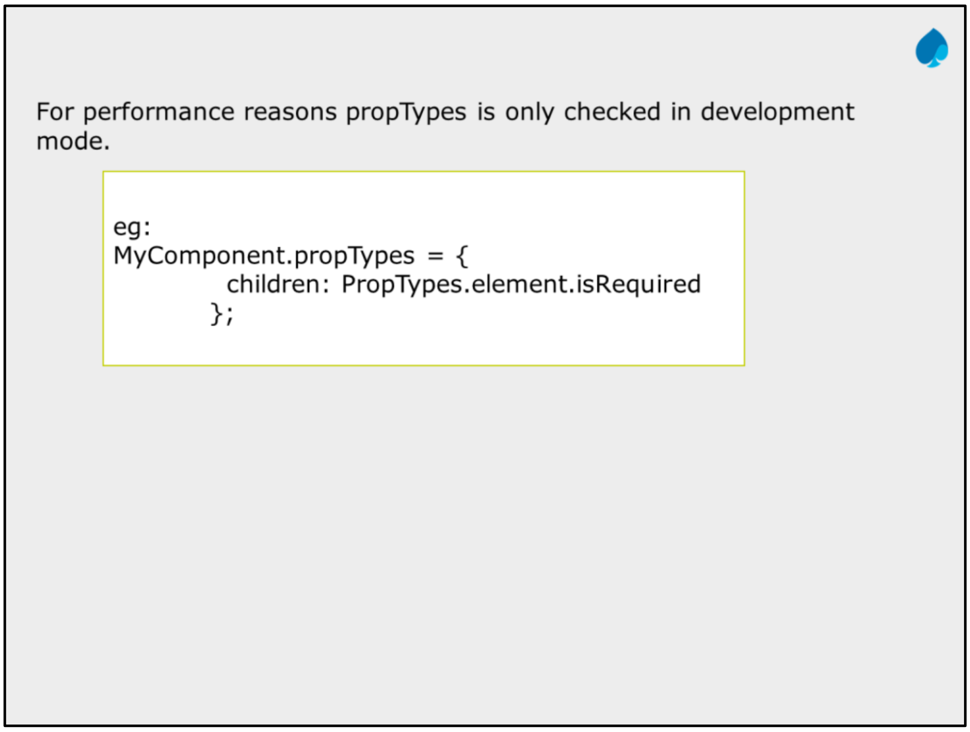
Instru

long-term productivity and coerce the code to seem more strongly typed.

Example:

```
import PropTypes from 'prop-types'
import React from 'react'
class Blogs extends Component {
render(){
return (
<div>
<h1>{this.props.title}</h1>
<p>{this.props.description}</p>
</div>
)
}
}
Blogs.propTypes = {
title: PropTypes.string,
description: PropTypes.string
}
export default Blogs;
```

Instructor Notes:

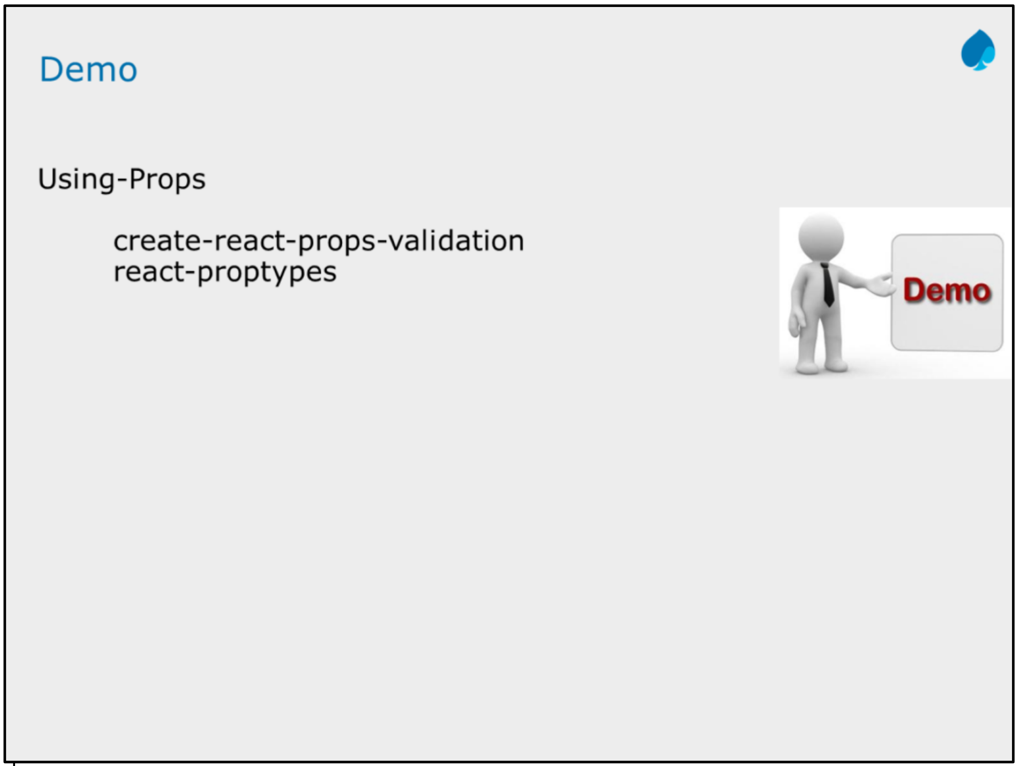


For performance reasons propTypes is only checked in development mode.

```
eg:
MyComponent.propTypes = {
          children: PropTypes.element.isRequired
        };
```

Instructor Notes:

Add instructor notes here.

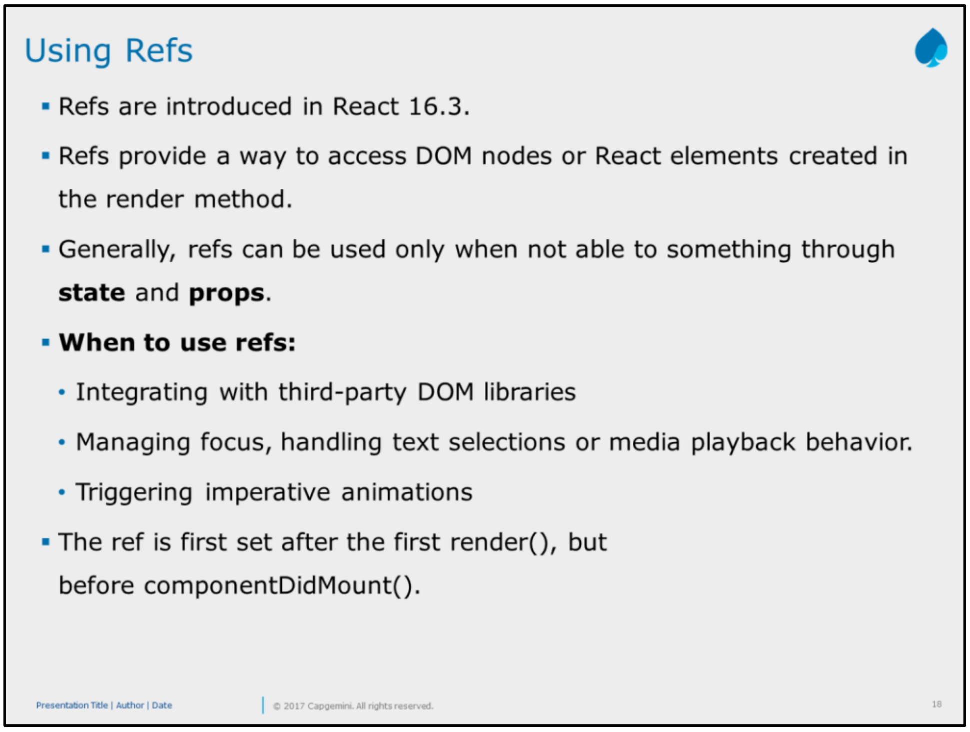


Demo

Using-Props

create-react-props-validation
react-proptypes

Instructor Notes:



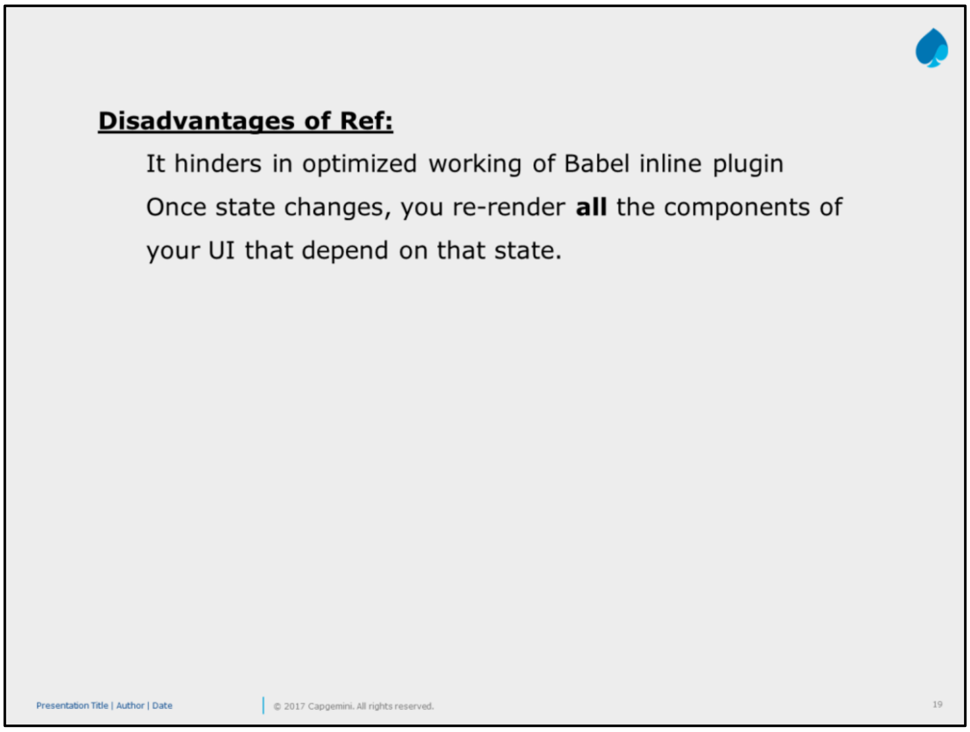
Using Refs

- Refs are introduced in React 16.3.
- Refs provide a way to access DOM nodes or React elements created in the render method.
- Generally, refs can be used only when not able to something through **state** and **props**.
- **When to use refs:**
  - Integrating with third-party DOM libraries
  - Managing focus, handling text selections or media playback behavior.
  - Triggering imperative animations
- The ref is first set after the first render(), but before componentDidMount().

React dataflow, props are the only way that parent components interact with their children.
React provides three major ways of creating refs. Here is a list of the different methods starting from the oldest of them:
String Refs *(legacy method)*
Callback Refs
React.createRef *(from React 16.3)*

Prior to React v16.3 the **callback ref** were the preferred way to create and use refs.

Instructor Notes:



Disadvantages of Ref:

It hinders in optimized working of Babel inline plugin

Once state changes, you re-render **all** the components of your UI that depend on that state.

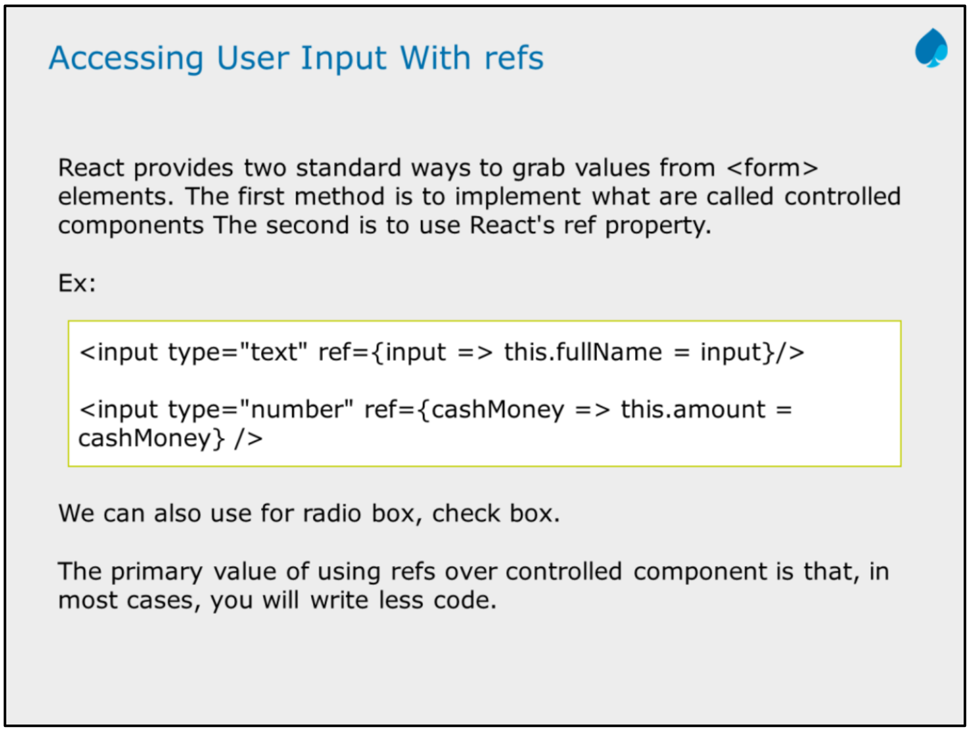
Instructor Notes:

Add instructor notes here.

Demo

react-refs-2019

Instructor Notes:



Accessing User Input With refs

React provides two standard ways to grab values from <form> elements. The first method is to implement what are called controlled components The second is to use React's ref property.

Ex:

```
<input type="text" ref={input => this.fullName = input}/>

<input type="number" ref={cashMoney => this.amount =
cashMoney} />
```

We can also use for radio box, check box.

The primary value of using refs over controlled component is that, in most cases, you will write less code.

Controlled components are heavy duty. The defining characteristic of a controlled component is the displayed value is bound to component state. To update the value, you execute a function attached to the onChange event handler on the form element and updates the state property, which in turn updates the form element's value.

An easier and less labor-intensive way to grab values from a form element is to use the ref property.

Use Refs on any one of the scenario's

Managing focus, text selection, or media playback.
Triggering imperative animations.
Integrating with third-party DOM libraries.

Your first inclination may be to use refs to "make things happen" in your app. If this is the case, take a moment and think more critically about where state should be owned in the component hierarchy. Often, it becomes clear that the proper place to "own" that state is at a higher level in the hierarchy

So avoid mostly using refs, because it may overhead the process
They're bad for maintainability, and lose a lot of the simplicity

For drop down:

<select ref={select => this.petType = select} name="petType"> <option value="cat">Cat</option> <option value="dog">Dog</option> <option value="ferret">Ferret</option> </select>

Note
The examples below have been updated to use the React.createRef() API introduced in React 16.3. If you are using an earlier release of React, we recommend using callback refsinstead.

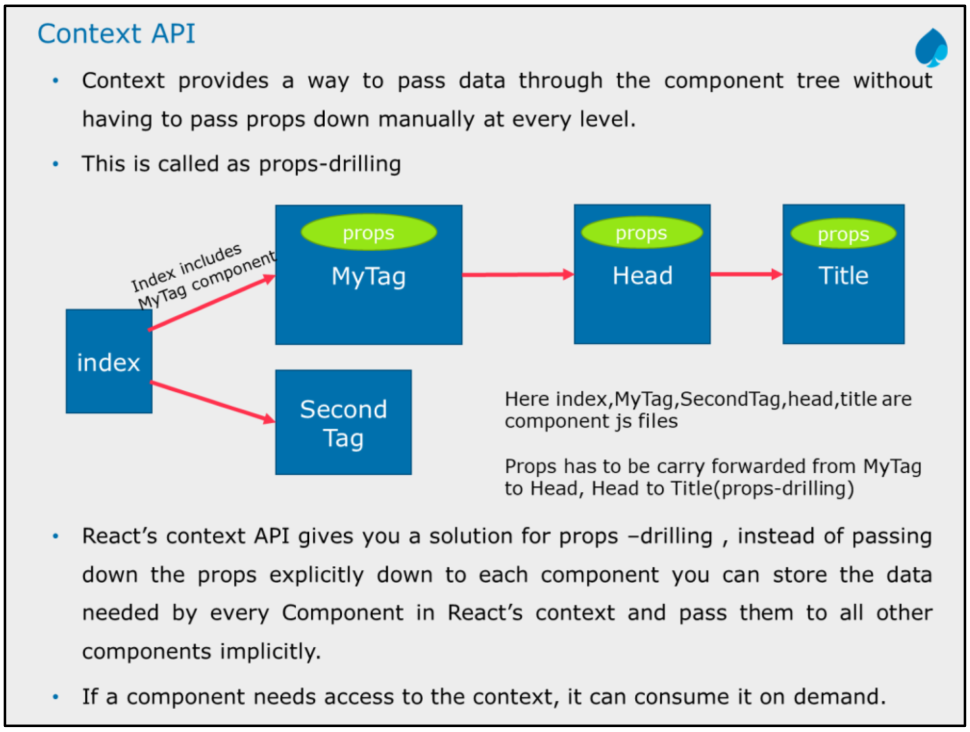
Instructor Notes:

Add instructor notes here.

Demo

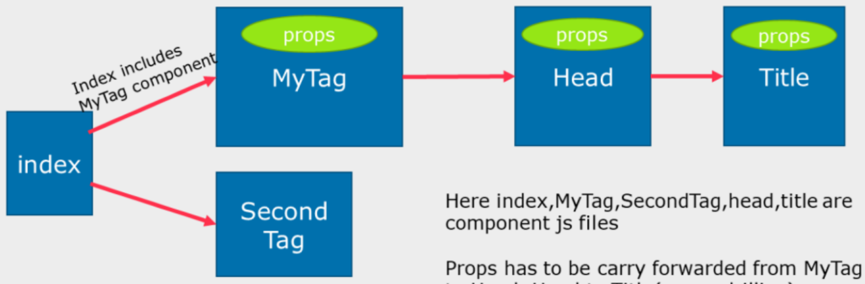
Create-react-forms-inputrefs

Instructor Notes:



Context API

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- This is called as props-drilling



Here index,MyTag,SecondTag,head,title are component js files

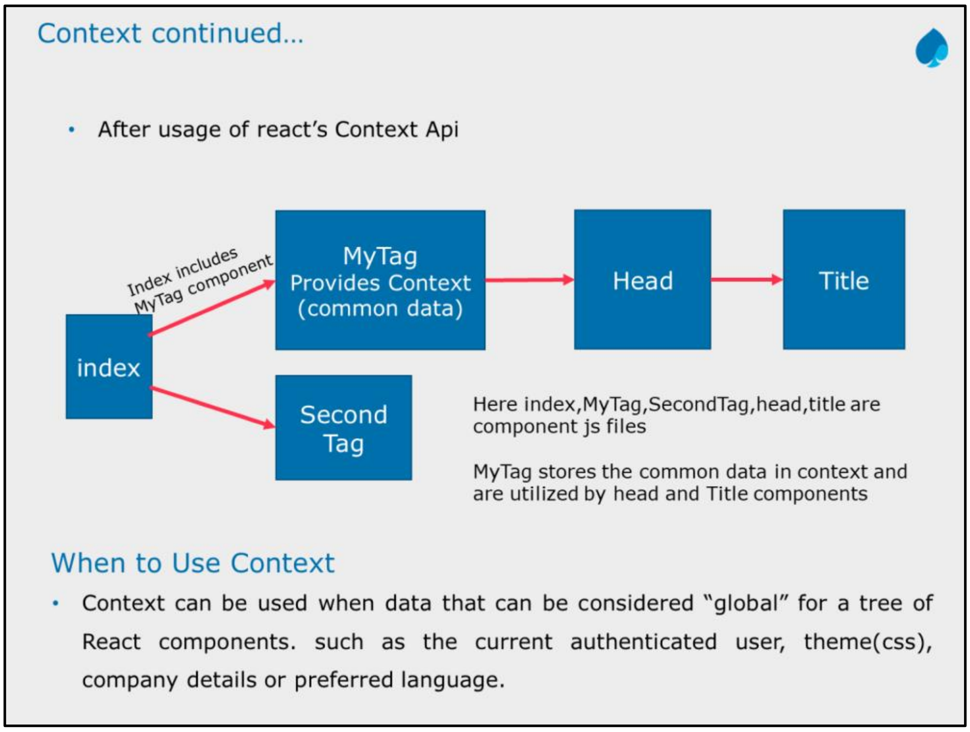
Props has to be carry forwarded from MyTag to Head, Head to Title(props-drilling)

- React's context API gives you a solution for props –drilling , instead of passing down the props explicitly down to each component you can store the data needed by every Component in React's context and pass them to all other components implicitly.
- If a component needs access to the context, it can consume it on demand.

Passing props down the component tree - Props drilling

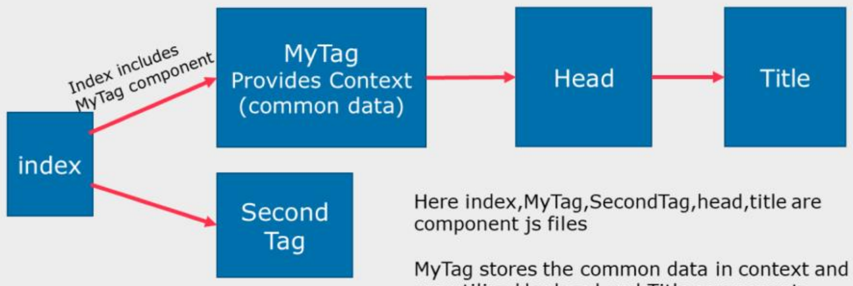
Context provides a way to share values like this between components without having to explicitly pass a prop through every level of the tree.

Instructor Notes:



Context continued...

- After usage of react's Context Api


Index includes MyTag component
index
MyTag Provides Context (common data)
Head
Title
Second Tag

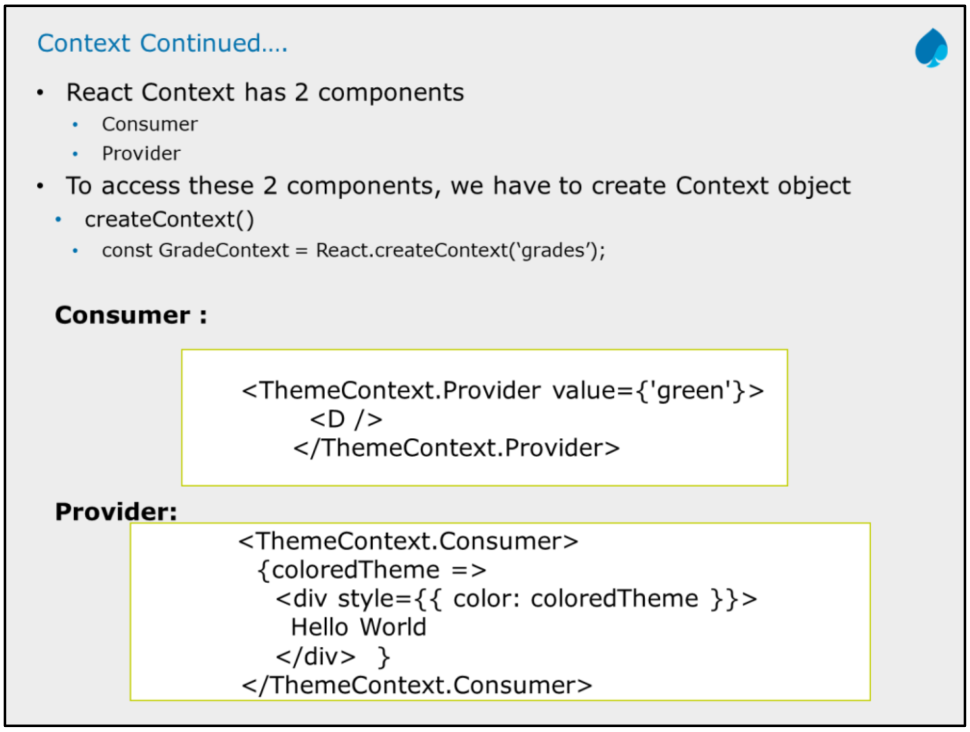
Here index,MyTag,SecondTag,head,title are component js files

MyTag stores the common data in context and are utilized by head and Title components

When to Use Context

- Context can be used when data that can be considered "global" for a tree of React components. such as the current authenticated user, theme(css), company details or preferred language.

Instructor Notes:



Context Continued….

- React Context has 2 components
  - Consumer
  - Provider
- To access these 2 components, we have to create Context object
  - createContext()
  - const GradeContext = React.createContext('grades');

Consumer :

```
<ThemeContext.Provider value={'green'}>
      <D />
    </ThemeContext.Provider>
```

Provider:

```
<ThemeContext.Consumer>
  {coloredTheme =>
    <div style={{ color: coloredTheme }}>
      Hello World
    </div>  }
</ThemeContext.Consumer>
```

For create Context() method , there is single argument :

the initial value is can be null, or if you need a default value provide it as argument.

React.createContext(null);

The value is set to null, so later we can set the value

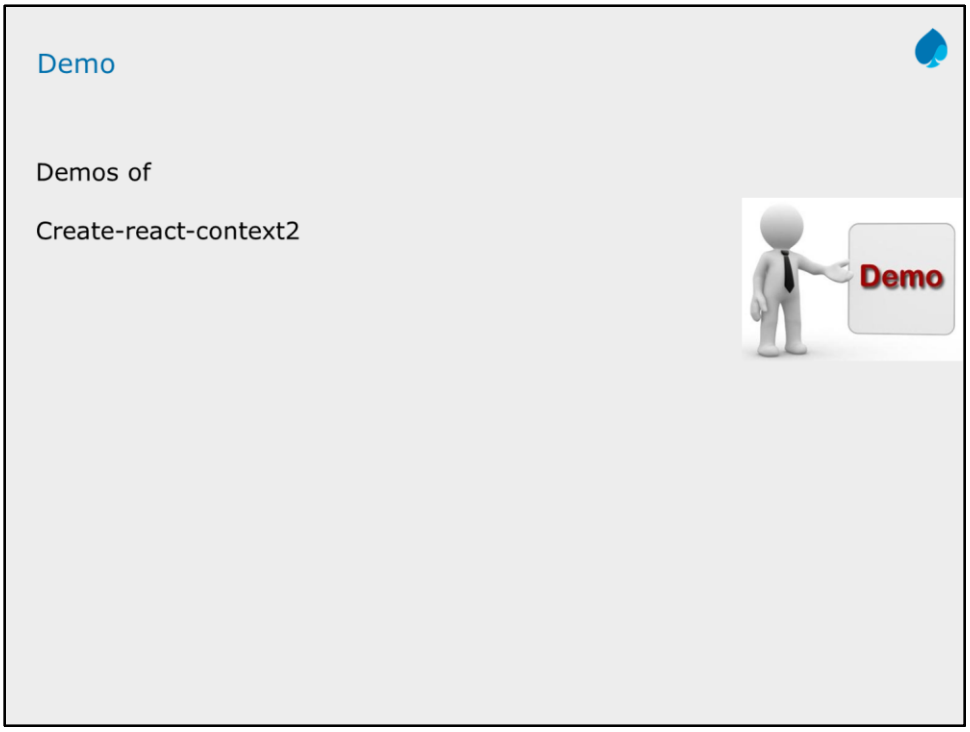
React.createContext('grades'); -→ here grades is considered as initial value

Context is similar to props except that a change in context doesn't actually trigger a render. Usually context takes its value from a state or a store so that's usually not a problem. Another downside is unlike props, React doesn't provide a way to set a default value for it.

Validasting child context Types:
// Define types of elements in context // We define it the same way as `propTypes` childContextTypes: { eventBus: React.PropTypes.object.isRequired },

Instructor Notes:

Add instructor notes here.



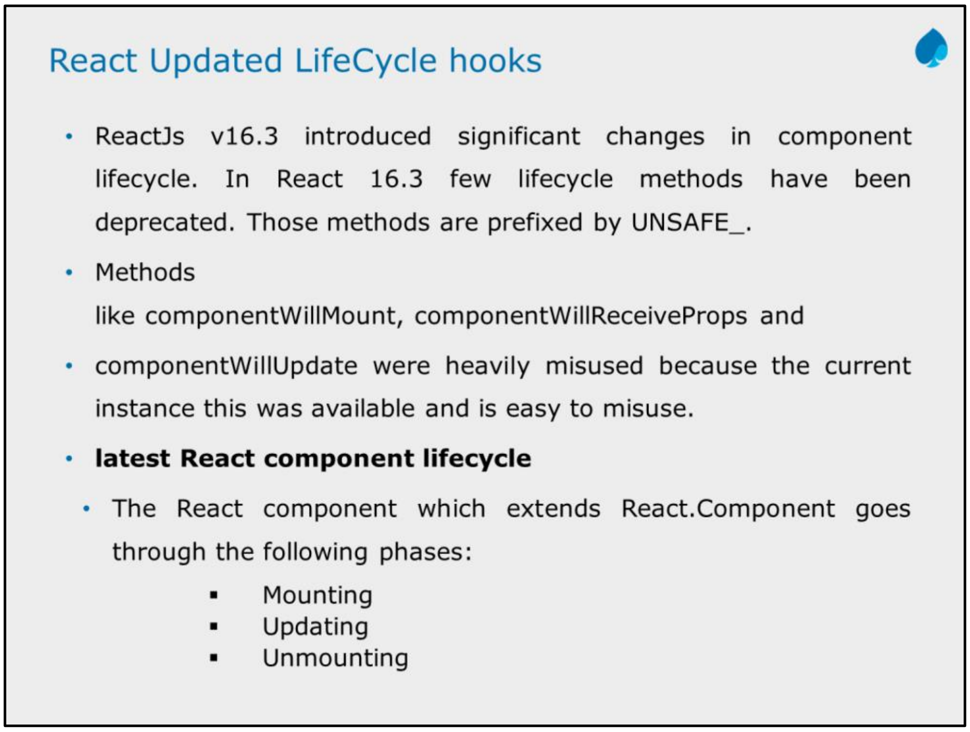
Demo

Demos of

Create-react-context2

Add the notes here.

Instructor Notes:



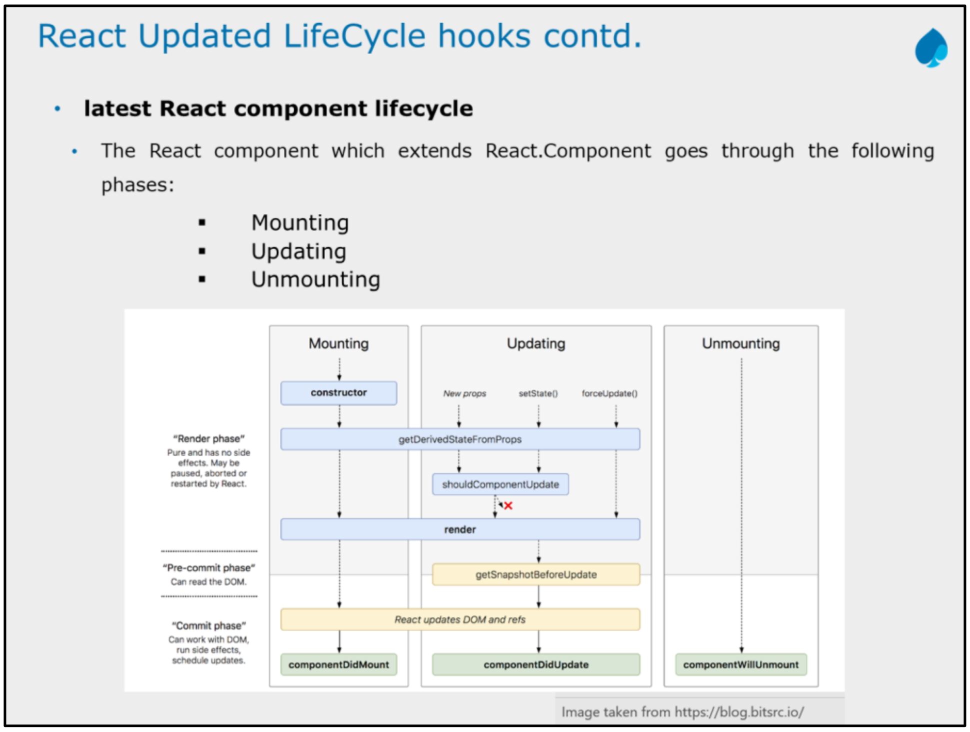
React Updated LifeCycle hooks

- ReactJs v16.3 introduced significant changes in component lifecycle. In React 16.3 few lifecycle methods have been deprecated. Those methods are prefixed by UNSAFE_.
- Methods like componentWillMount, componentWillReceiveProps and
- componentWillUpdate were heavily misused because the current instance this was available and is easy to misuse.
- **latest React component lifecycle**
  - The React component which extends React.Component goes through the following phases:
    - Mounting
    - Updating
    - Unmounting

React Team decided to move onto static lifecycle methods and started seeking to improve the UX and performance of React, they moved towards async rendering.

componentWillMount
All the legacy use cases are covered in the constructor. This is renamed as UNSAFE_componentWillMount.
componentWillReceiveProps
The new static method getDerivedStateFromProps is safe rewrite for this method and covers all the use cases of componentWillReceiveProps. The new name for this method is UNSAFE_componentWillReceiveProps.
componentWillUpdate
The new method getSnapshotBeforeUpdate is safe rewrite for this method and covers all the use cases of componentWillUpdate.
The new name for this method is UNSAFE_componentWillUpdate.

Instructor Notes:



React Updated LifeCycle hooks contd.

- **latest React component lifecycle**
  - The React component which extends React.Component goes through the following phases:
    - Mounting
    - Updating
    - Unmounting

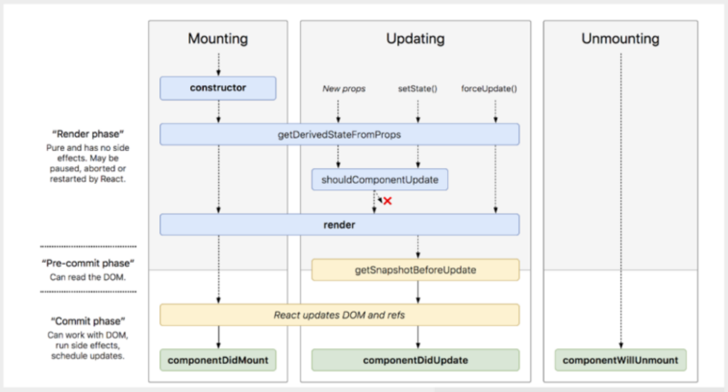
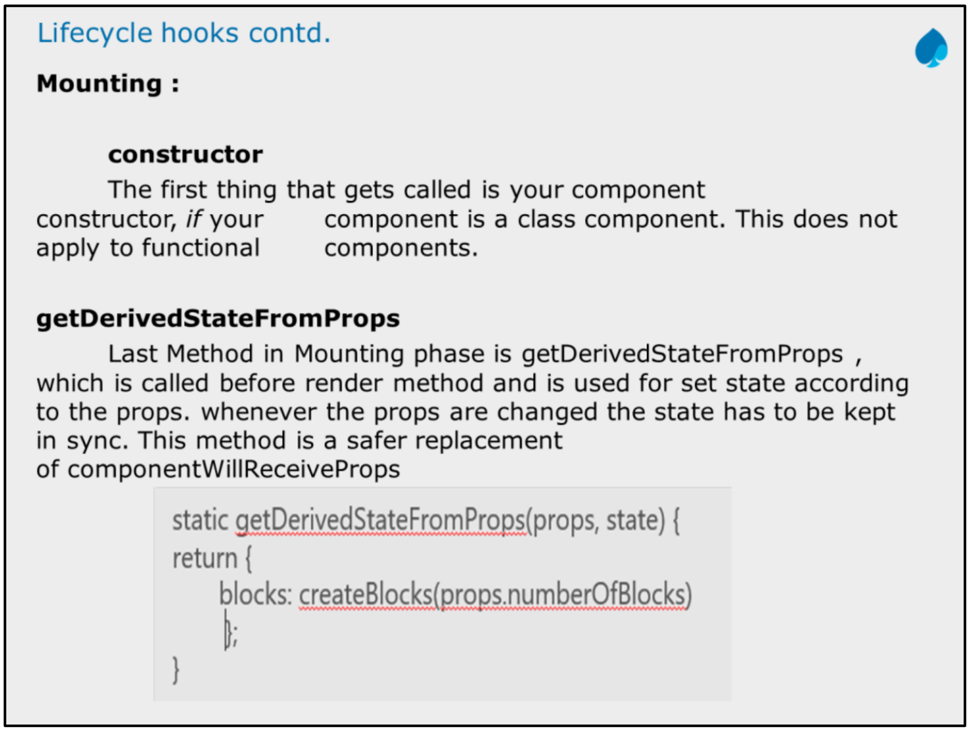


Image taken from https://blog.bitsrc.io/

Image taken from https://blog.bitsrc.io/

Instructor Notes:



Lifecycle hooks contd.

Mounting :

constructor

The first thing that gets called is your component constructor, *if* your component is a class component. This does not apply to functional components.

getDerivedStateFromProps

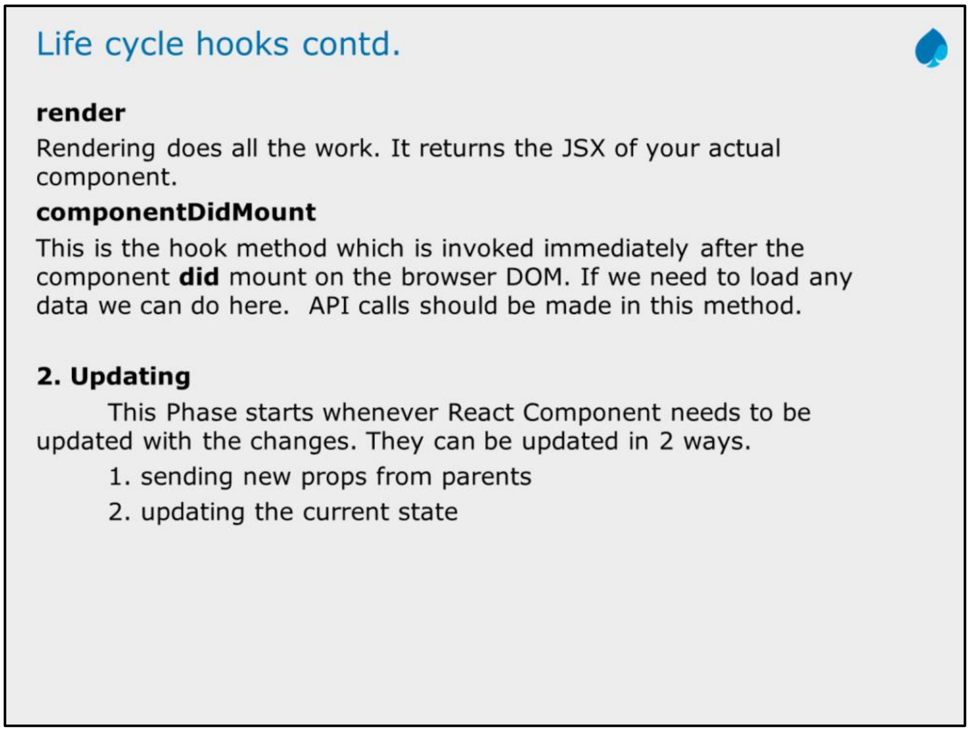
Last Method in Mounting phase is getDerivedStateFromProps , which is called before render method and is used for set state according to the props. whenever the props are changed the state has to be kept in sync. This method is a safer replacement of componentWillReceiveProps

```
static getDerivedStateFromProps(props, state) {
return {
      blocks: createBlocks(props.numberOfBlocks)
       };
}
```

getDerivedStateFromProps:

The method getDerivedStateFromProps is static, hence it has no access to this. This method has access to the current state and props.

Instructor Notes:



Life cycle hooks contd.

render

Rendering does all the work. It returns the JSX of your actual component.

componentDidMount

This is the hook method which is invoked immediately after the component **did** mount on the browser DOM. If we need to load any data we can do here. API calls should be made in this method.

2. Updating

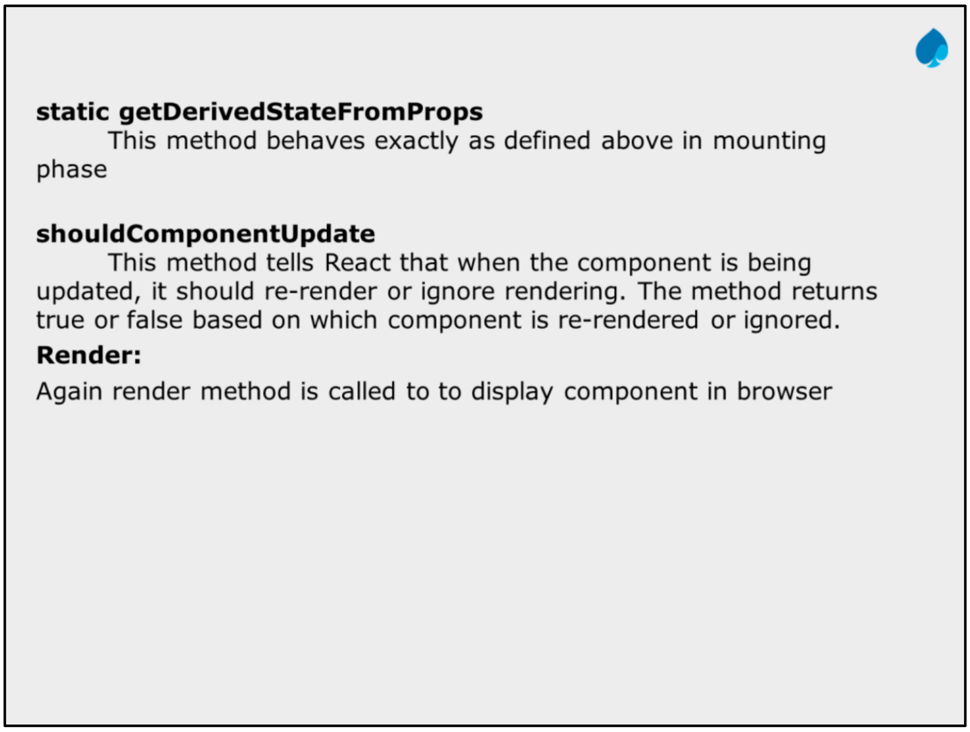
This Phase starts whenever React Component needs to be updated with the changes. They can be updated in 2 ways.

1. sending new props from parents
2. updating the current state

Updating:

Components must be re-rendered only if any changes happens in props changes

Instructor Notes:



static getDerivedStateFromProps

This method behaves exactly as defined above in mounting phase

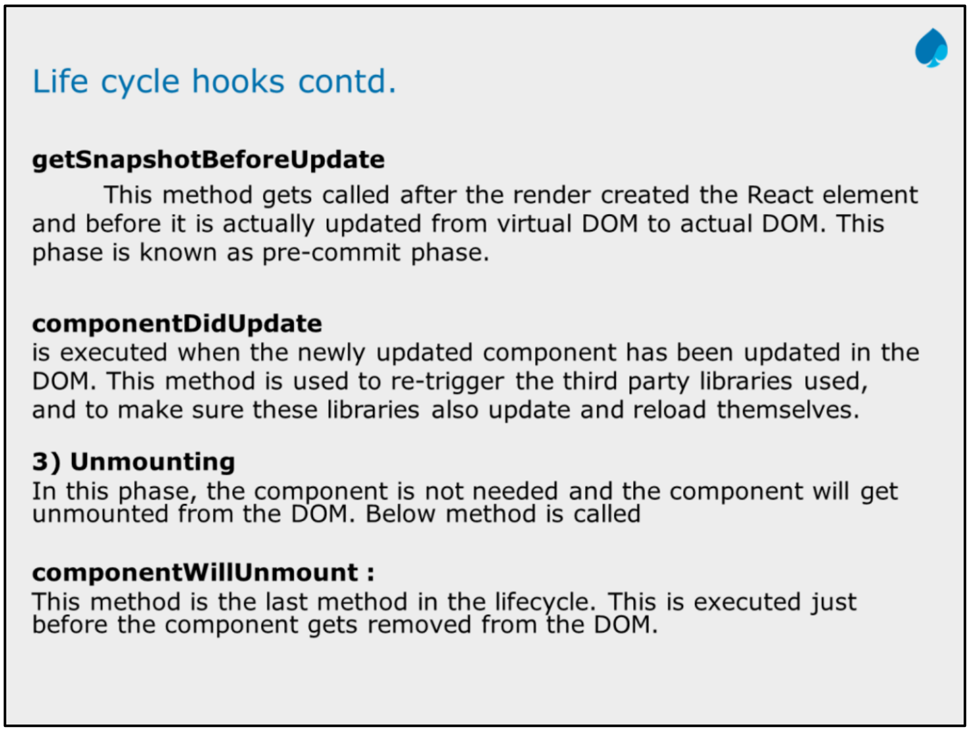
shouldComponentUpdate

This method tells React that when the component is being updated, it should re-render or ignore rendering. The method returns true or false based on which component is re-rendered or ignored.

Render:

Again render method is called to to display component in browser

Instructor Notes:



Life cycle hooks contd.

getSnapshotBeforeUpdate

This method gets called after the render created the React element and before it is actually updated from virtual DOM to actual DOM. This phase is known as pre-commit phase.

componentDidUpdate
is executed when the newly updated component has been updated in the DOM. This method is used to re-trigger the third party libraries used, and to make sure these libraries also update and reload themselves.

3) Unmounting
In this phase, the component is not needed and the component will get unmounted from the DOM. Below method is called

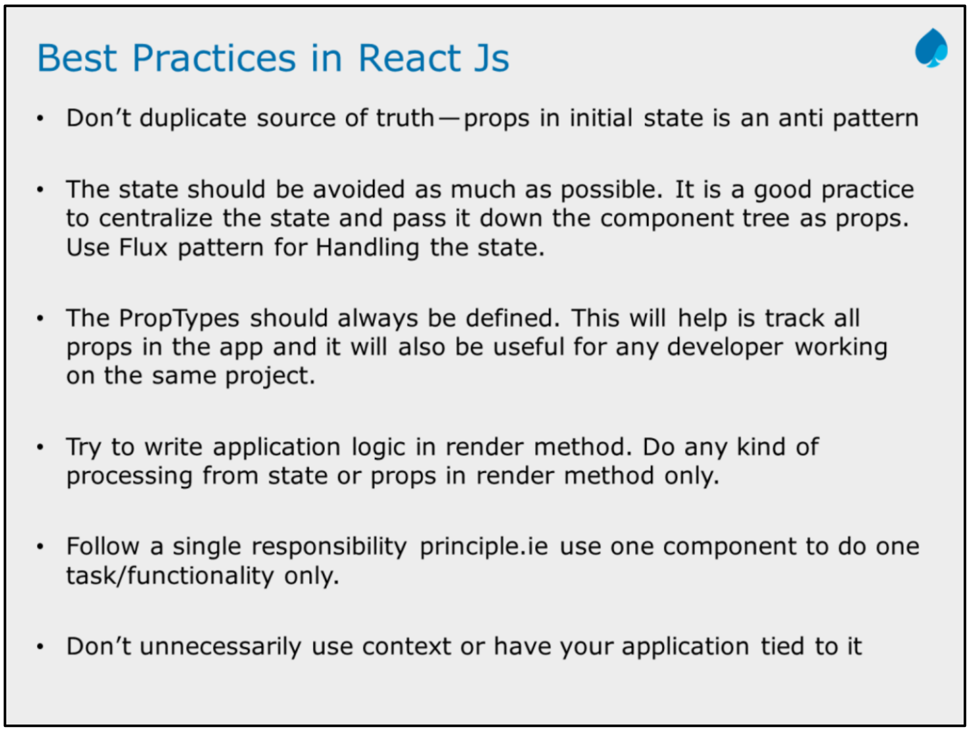
componentWillUnmount :
This method is the last method in the lifecycle. This is executed just before the component gets removed from the DOM.

getSnapshotBeforeUpdate

Usage: This method is useful if you want to keep sync in-between state of current DOM with the updated DOM. E.g. scroll position, audio/video, text-selection, cursor position, tool-tip position, etc.

ComponentWillUnmount:
Usage: In this method, we do all the cleanups related to the component.

Instructor Notes:



Best Practices in React Js

- Don’t duplicate source of truth—props in initial state is an anti pattern
- The state should be avoided as much as possible. It is a good practice to centralize the state and pass it down the component tree as props. Use Flux pattern for Handling the state.
- The PropTypes should always be defined. This will help is track all props in the app and it will also be useful for any developer working on the same project.
- Try to write application logic in render method. Do any kind of processing from state or props in render method only.
- Follow a single responsibility principle.ie use one component to do one task/functionality only.
- Don’t unnecessarily use context or have your application tied to it

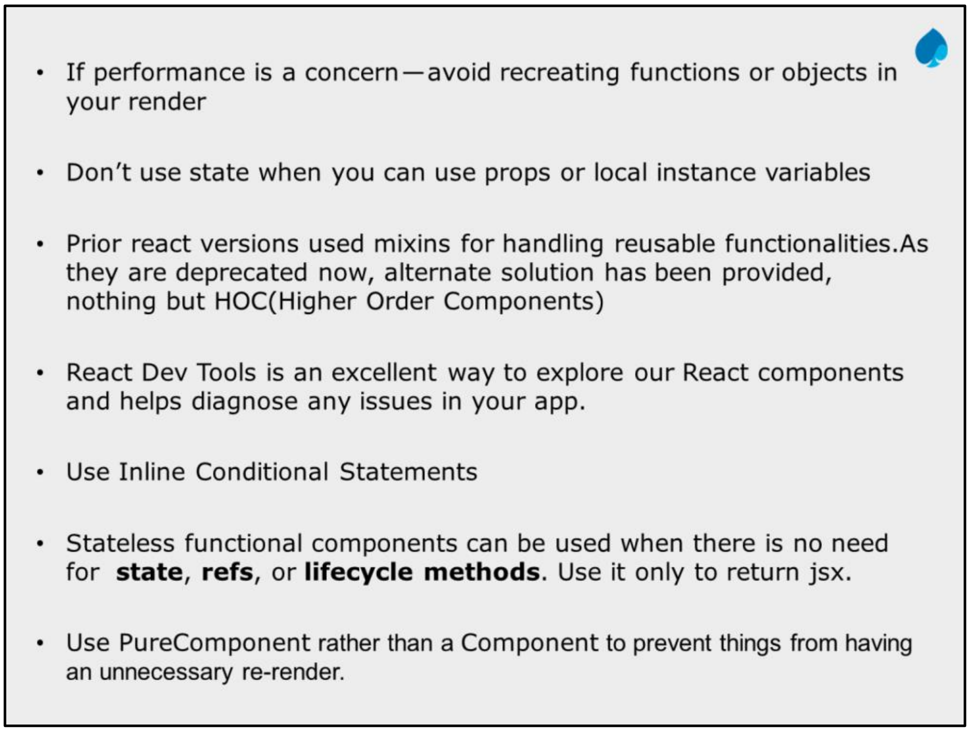
Don’t duplicate source of truth:

```
constructor(props){
super(props);
this.state = {
myname: props.name,
};
}
```

If you pass in new props.name it wont be used because its only invoked when the component is first created—**DON’T** do this.

Donot use Context Api much or put most of application data in context api as its not supported with shouldComponentUpdatelife cycle hook

Instructor Notes:



- If performance is a concern—avoid recreating functions or objects in your render
- Don't use state when you can use props or local instance variables
- Prior react versions used mixins for handling reusable functionalities.As they are deprecated now, alternate solution has been provided, nothing but HOC(Higher Order Components)
- React Dev Tools is an excellent way to explore our React components and helps diagnose any issues in your app.
- Use Inline Conditional Statements
- Stateless functional components can be used when there is no need for **state**, **refs**, or **lifecycle methods**. Use it only to return jsx.
- Use PureComponent rather than a Component to prevent things from having an unnecessary re-render.

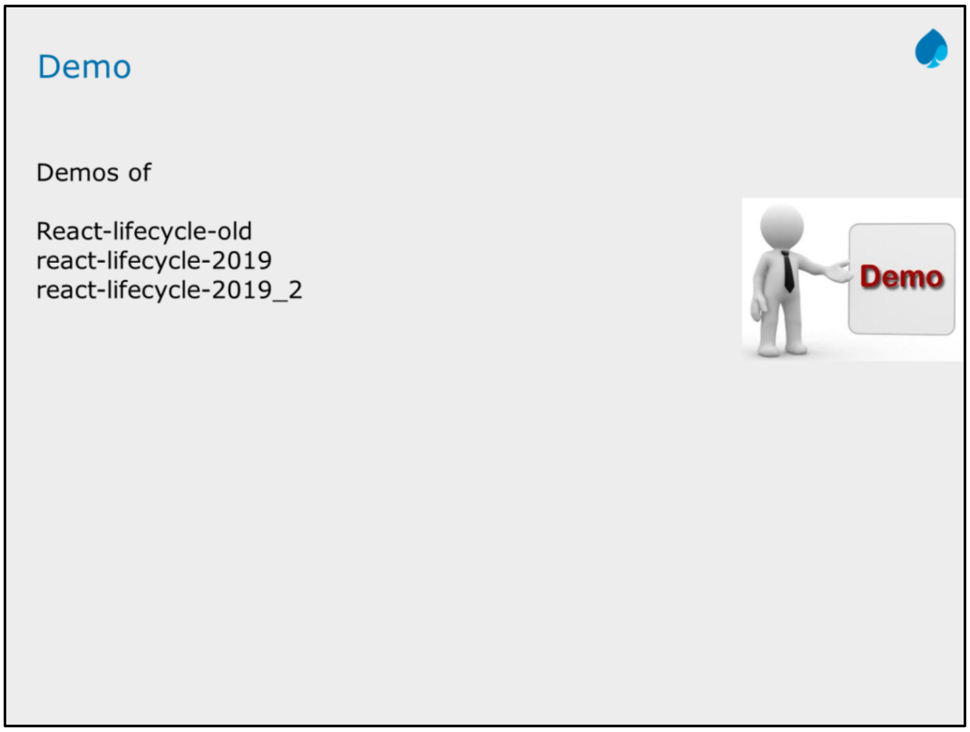
Inconditional statements example:
<div> {isRole('admin', user.id) && <UserProfile userId={user.id} /> }</div>

The above code avoids writing a function for checking role

React Dev tools:
React Dev Tools are available for all major browsers such as Chrome and Firefox.

Instructor Notes:

Add instructor notes here.



Demo

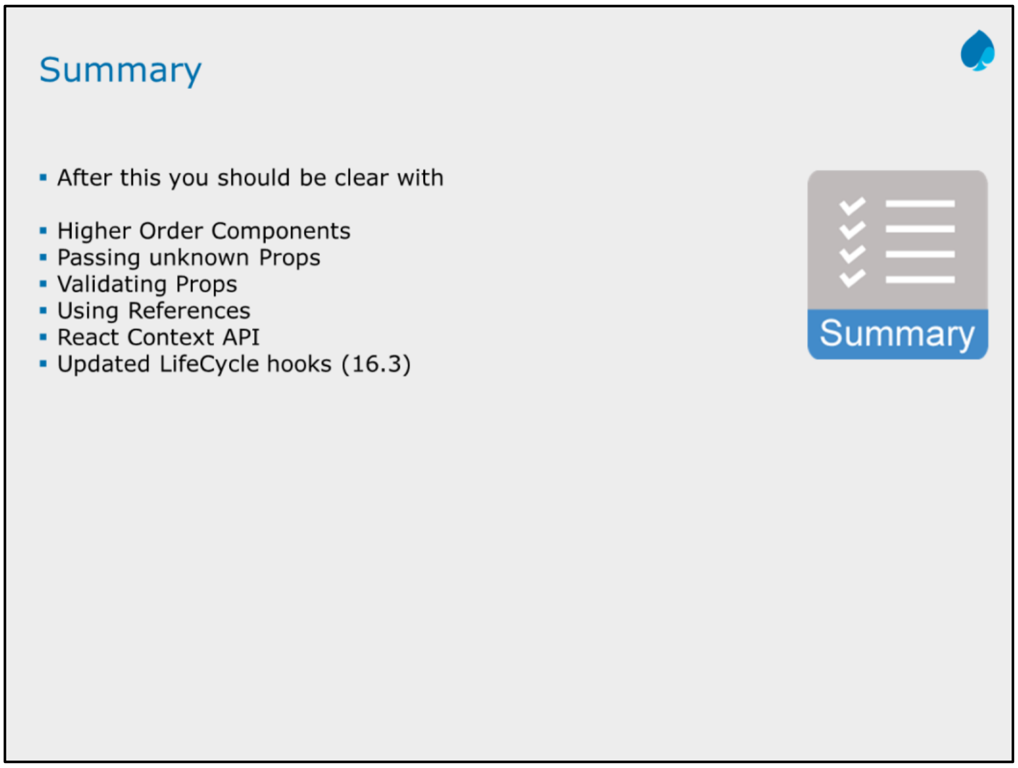
Demos of

React-lifecycle-old
react-lifecycle-2019
react-lifecycle-2019_2

Add the notes here.

Instructor Notes:

Add instructor notes here.

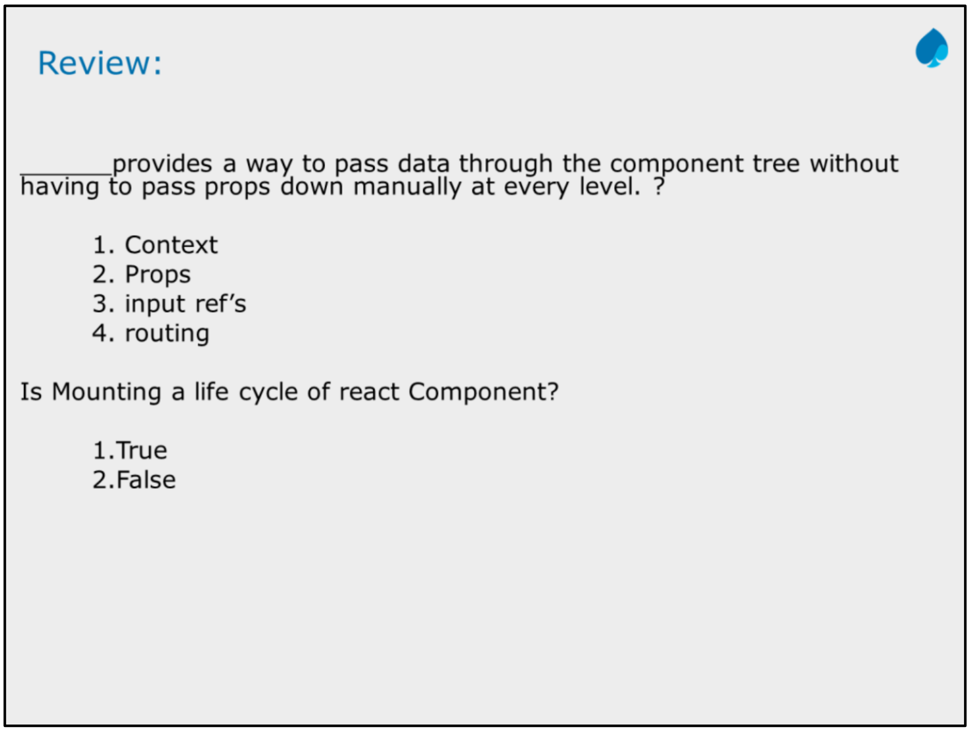


Summary

- After this you should be clear with
- Higher Order Components
- Passing unknown Props
- Validating Props
- Using References
- React Context API
- Updated LifeCycle hooks (16.3)

Add the notes here.

Instructor Notes:



Review:

_______provides a way to pass data through the component tree without having to pass props down manually at every level. ?

1. Context
2. Props
3. input ref's
4. routing

Is Mounting a life cycle of react Component?

1.True
2.False