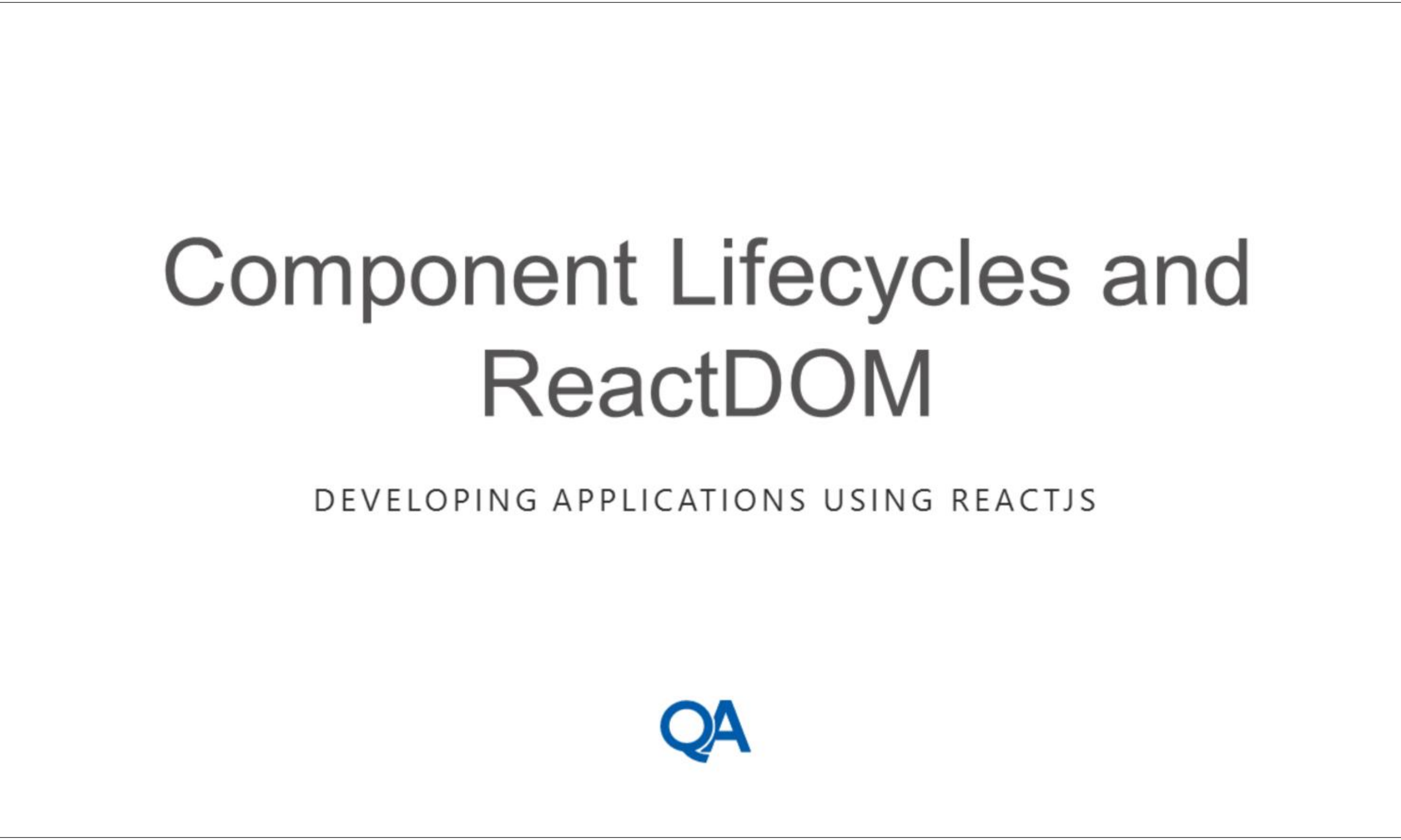
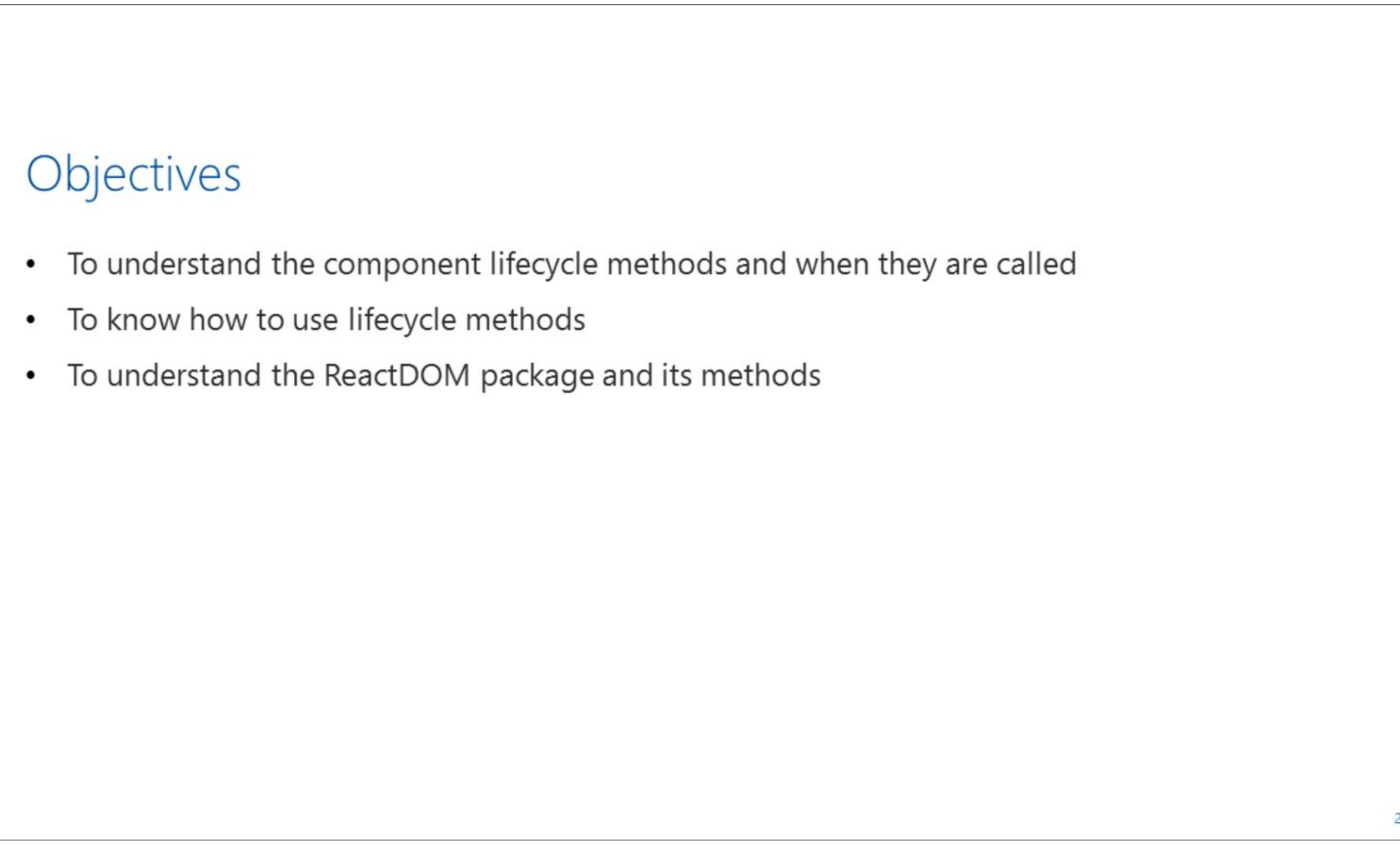


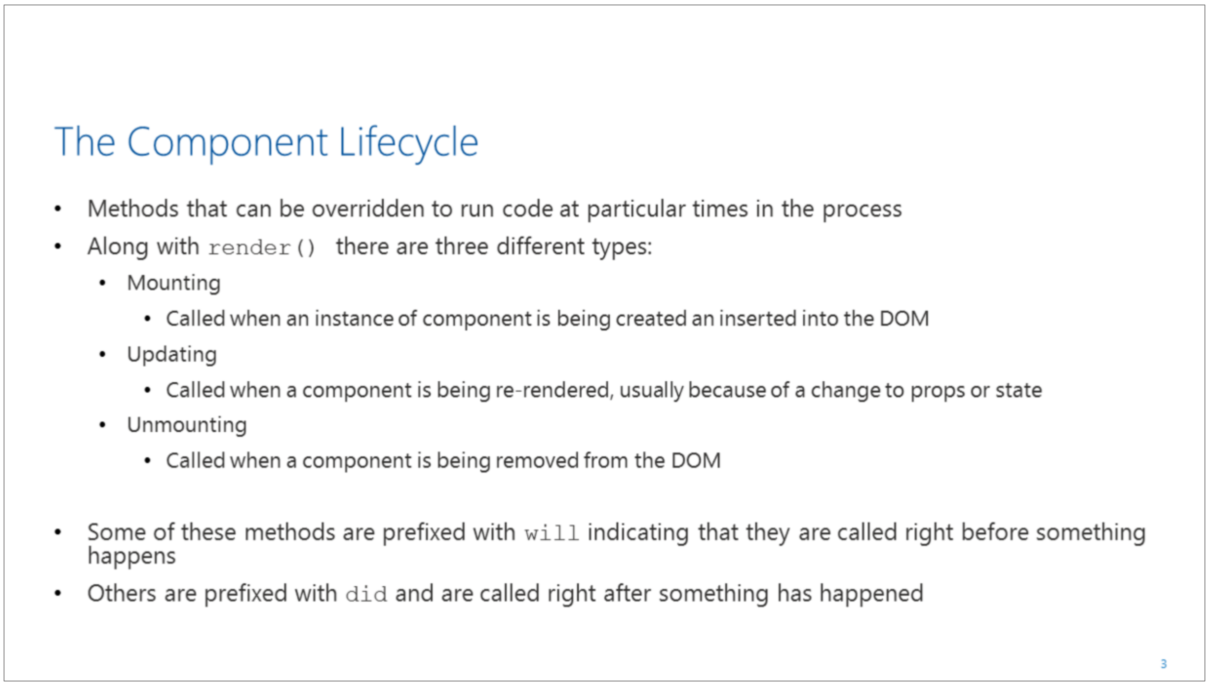
Component Lifecycles and ReactDOM

DEVELOPING APPLICATIONS USING REACTJS



Objectives

- To understand the component lifecycle methods and when they are called
- To know how to use lifecycle methods
- To understand the ReactDOM package and its methods

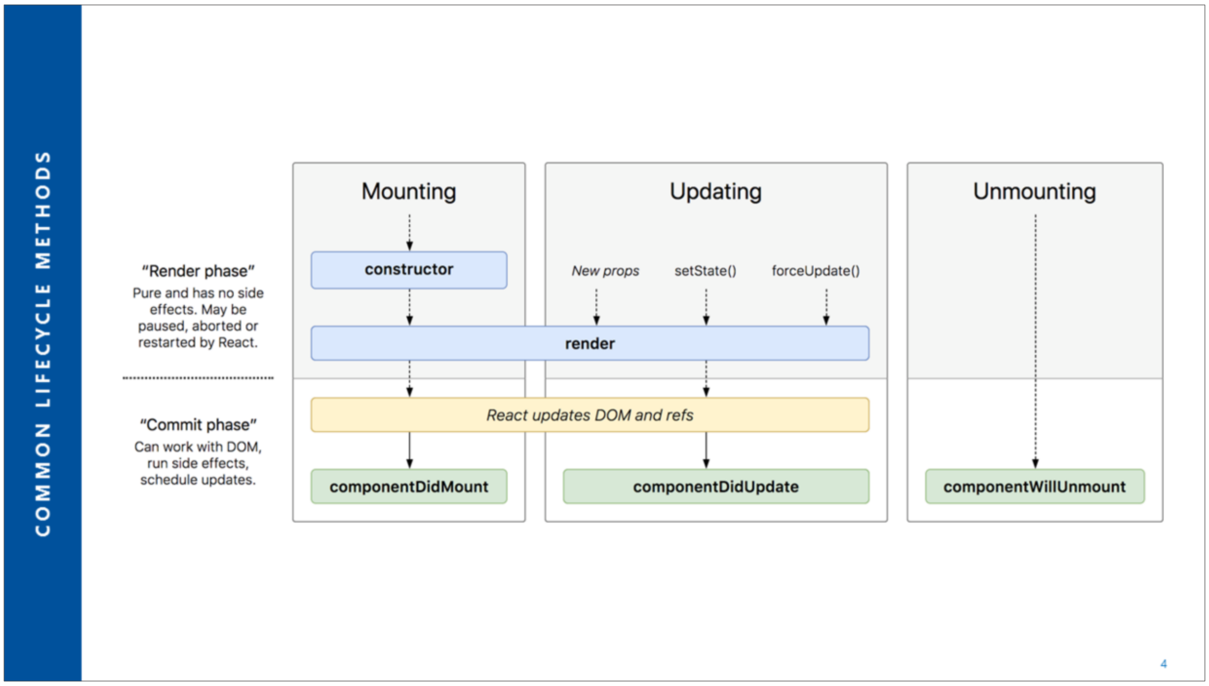


The Component Lifecycle

- Methods that can be overridden to run code at particular times in the process
- Along with `render()` there are three different types:
  - Mounting
    - Called when an instance of component is being created an inserted into the DOM
  - Updating
    - Called when a component is being re-rendered, usually because of a change to props or state
  - Unmounting
    - Called when a component is being removed from the DOM

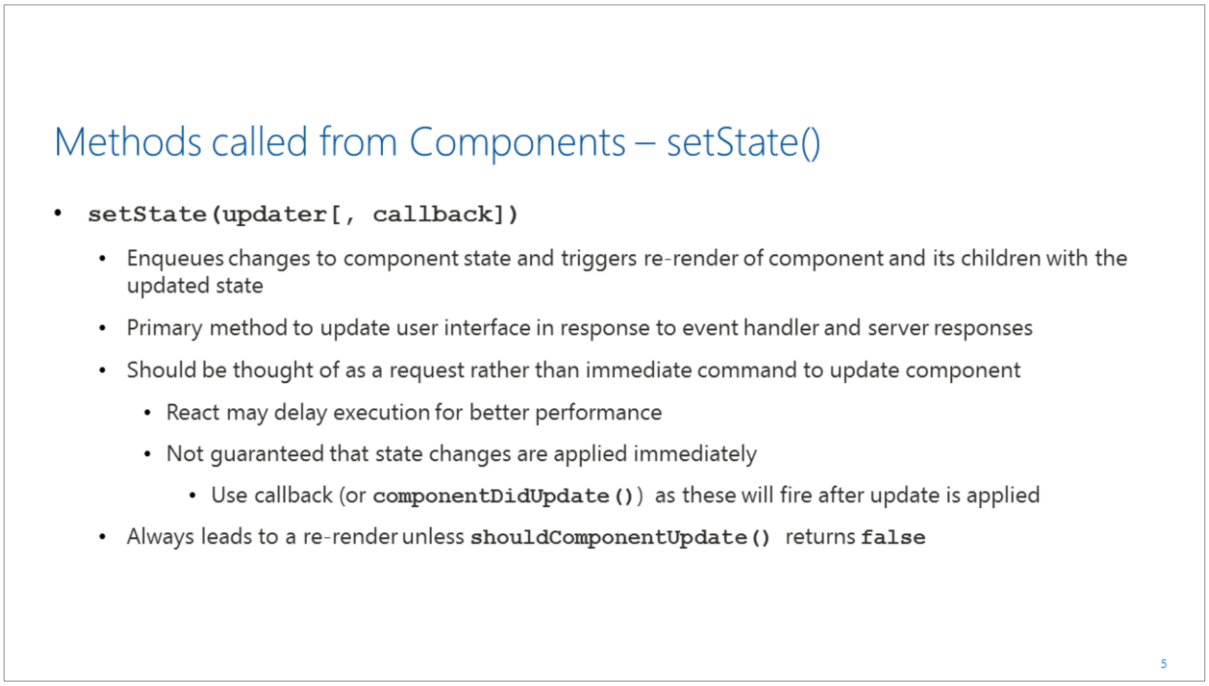
- Some of these methods are prefixed with `will` indicating that they are called right before something happens
- Others are prefixed with `did` and are called right after something has happened

3



COMMON LIFECYCLE METHODS

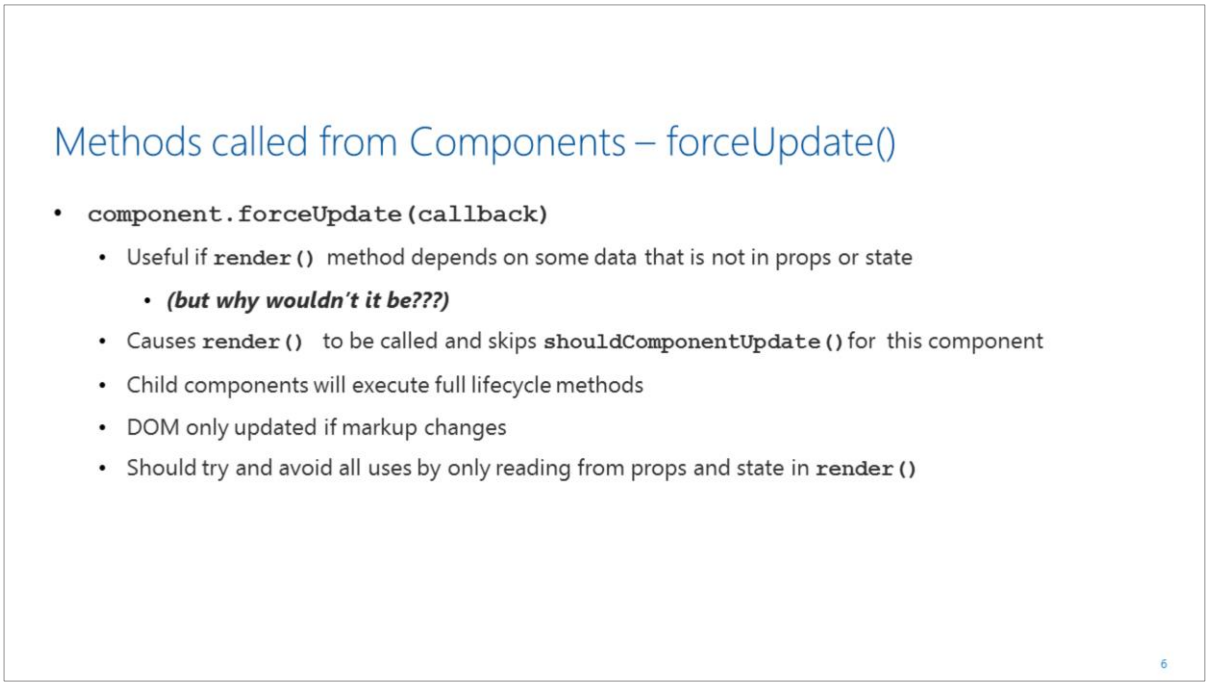
Image from: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/



Methods called from Components – setState()

- **`setState(updater[, callback])`**
  - Enqueues changes to component state and triggers re-render of component and its children with the updated state
  - Primary method to update user interface in response to event handler and server responses
  - Should be thought of as a request rather than immediate command to update component
    - React may delay execution for better performance
    - Not guaranteed that state changes are applied immediately
      - Use callback (or **`componentDidUpdate()`**) as these will fire after update is applied
  - Always leads to a re-render unless **`shouldComponentUpdate()`** returns **`false`**

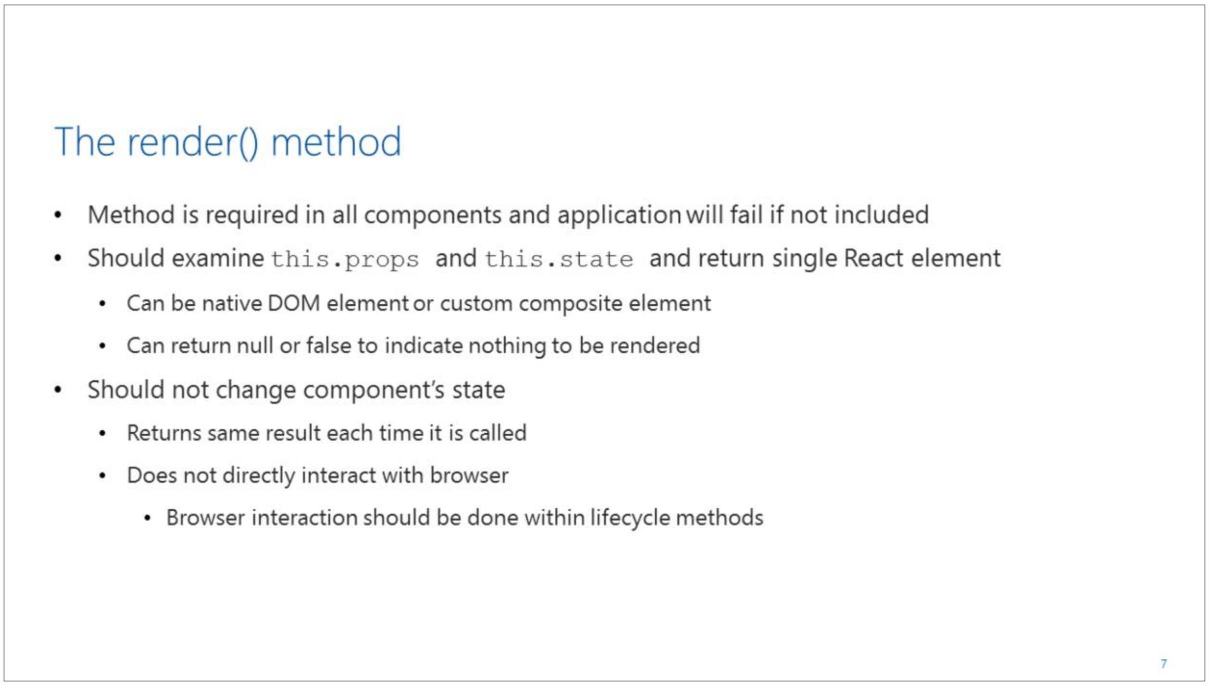
5



Methods called from Components – forceUpdate()

- **`component.forceUpdate(callback)`**
  - Useful if **`render()`** method depends on some data that is not in props or state
    - ***(but why wouldn't it be???)***
  - Causes **`render()`** to be called and skips **`shouldComponentUpdate()`** for this component
  - Child components will execute full lifecycle methods
  - DOM only updated if markup changes
  - Should try and avoid all uses by only reading from props and state in **`render()`**

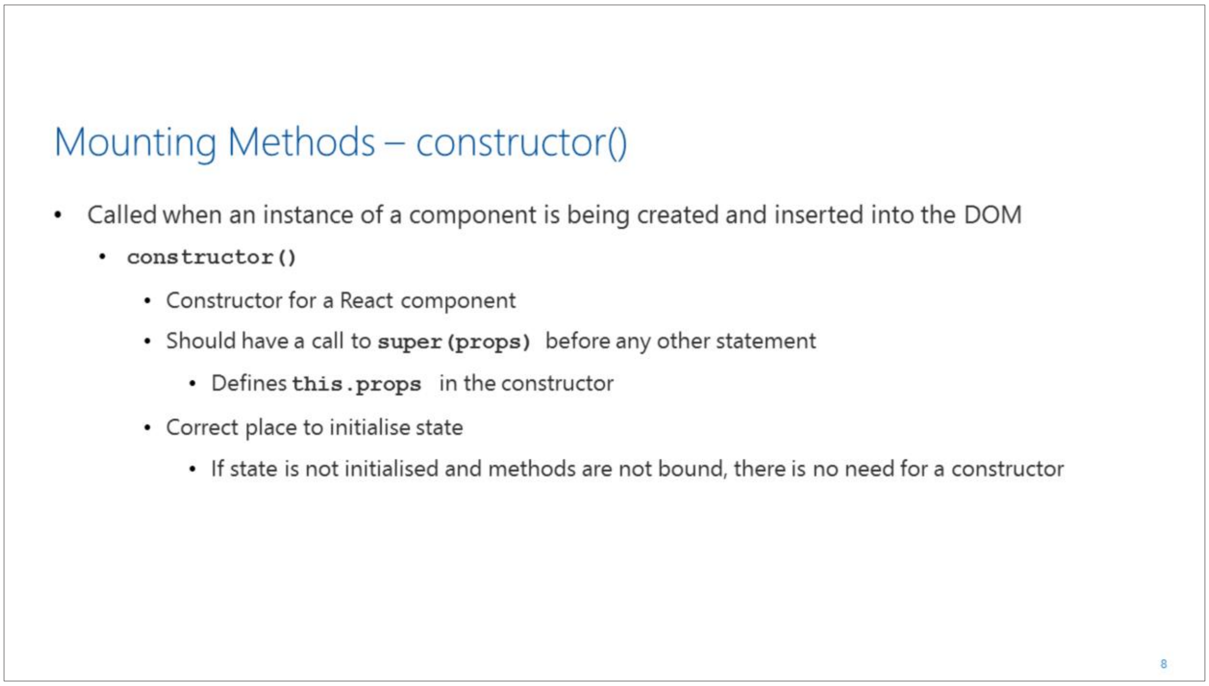
6



The render() method

- Method is required in all components and application will fail if not included
- Should examine `this.props` and `this.state` and return single React element
  - Can be native DOM element or custom composite element
  - Can return null or false to indicate nothing to be rendered
- Should not change component's state
  - Returns same result each time it is called
  - Does not directly interact with browser
    - Browser interaction should be done within lifecycle methods

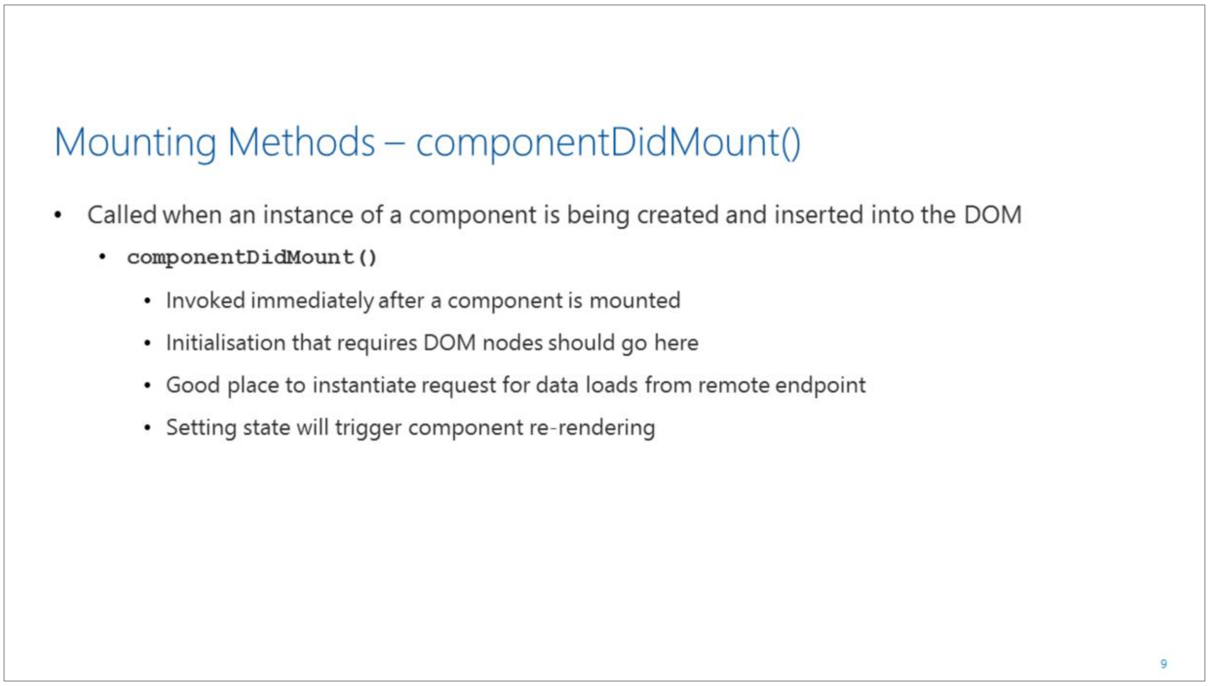
7



Mounting Methods – constructor()

- Called when an instance of a component is being created and inserted into the DOM
  - **`constructor()`**
    - Constructor for a React component
    - Should have a call to **`super(props)`** before any other statement
      - Defines **`this.props`** in the constructor
    - Correct place to initialise state
      - If state is not initialised and methods are not bound, there is no need for a constructor

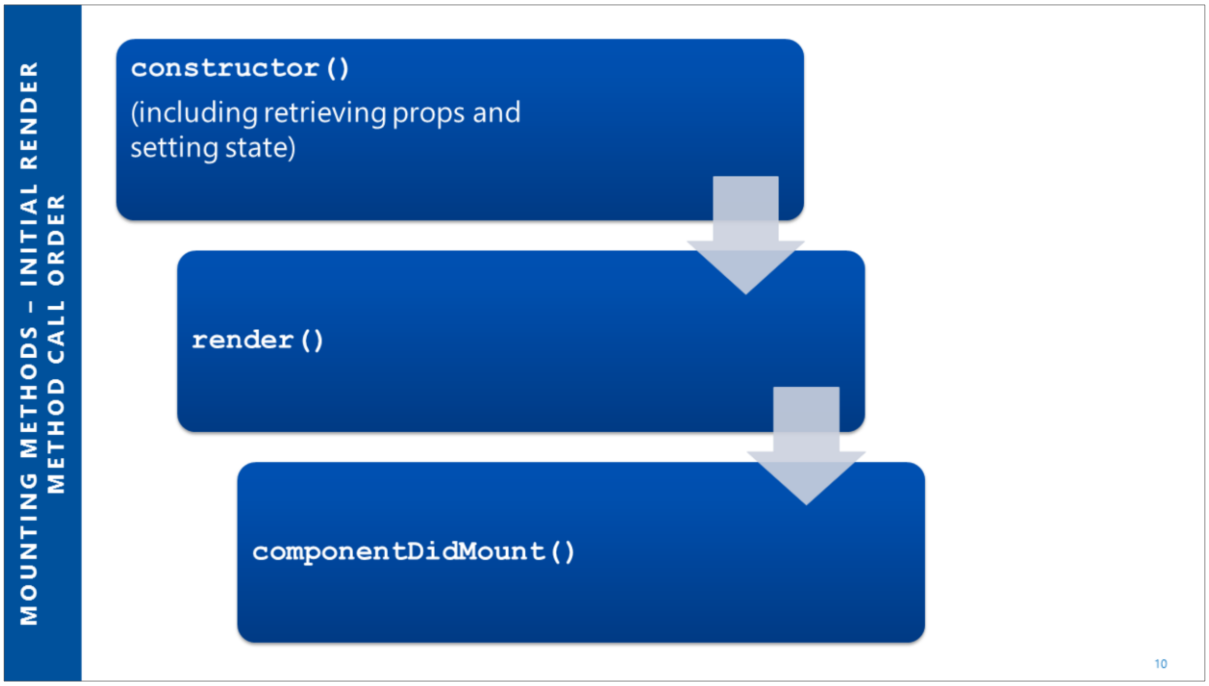
8



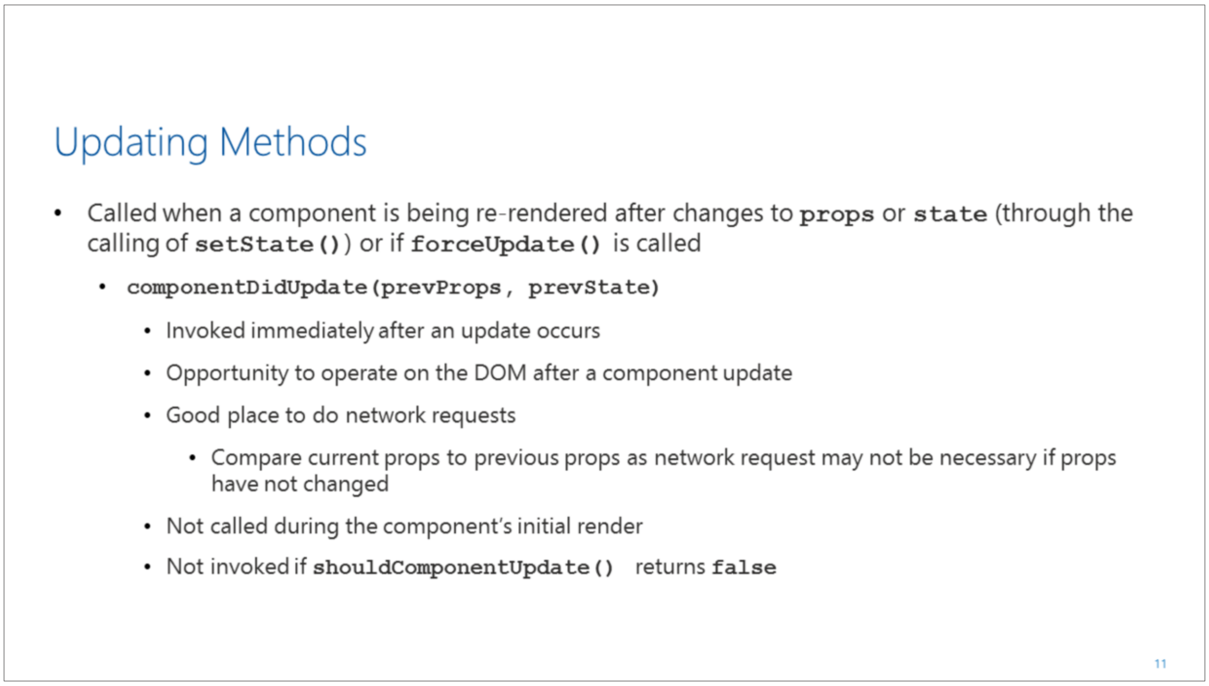
Mounting Methods – componentDidMount()

- Called when an instance of a component is being created and inserted into the DOM
  - **`componentDidMount()`**
    - Invoked immediately after a component is mounted
    - Initialisation that requires DOM nodes should go here
    - Good place to instantiate request for data loads from remote endpoint
    - Setting state will trigger component re-rendering

9



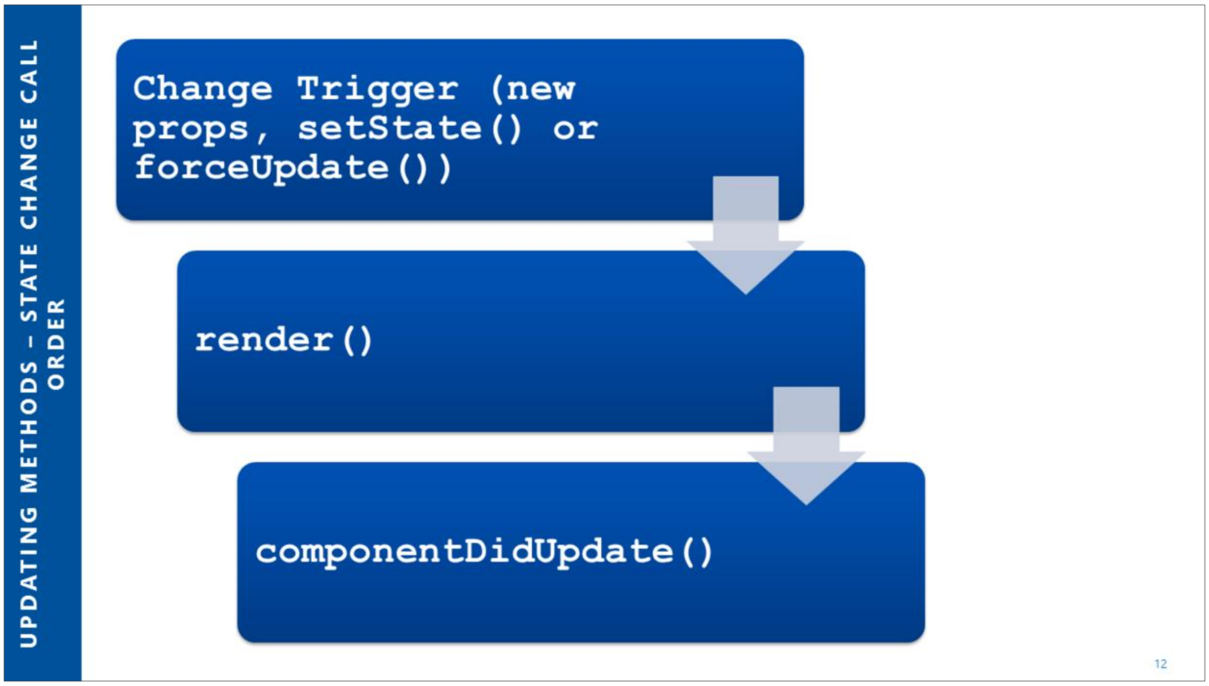
MOUNTING METHODS – INITIAL RENDER METHOD CALL ORDER



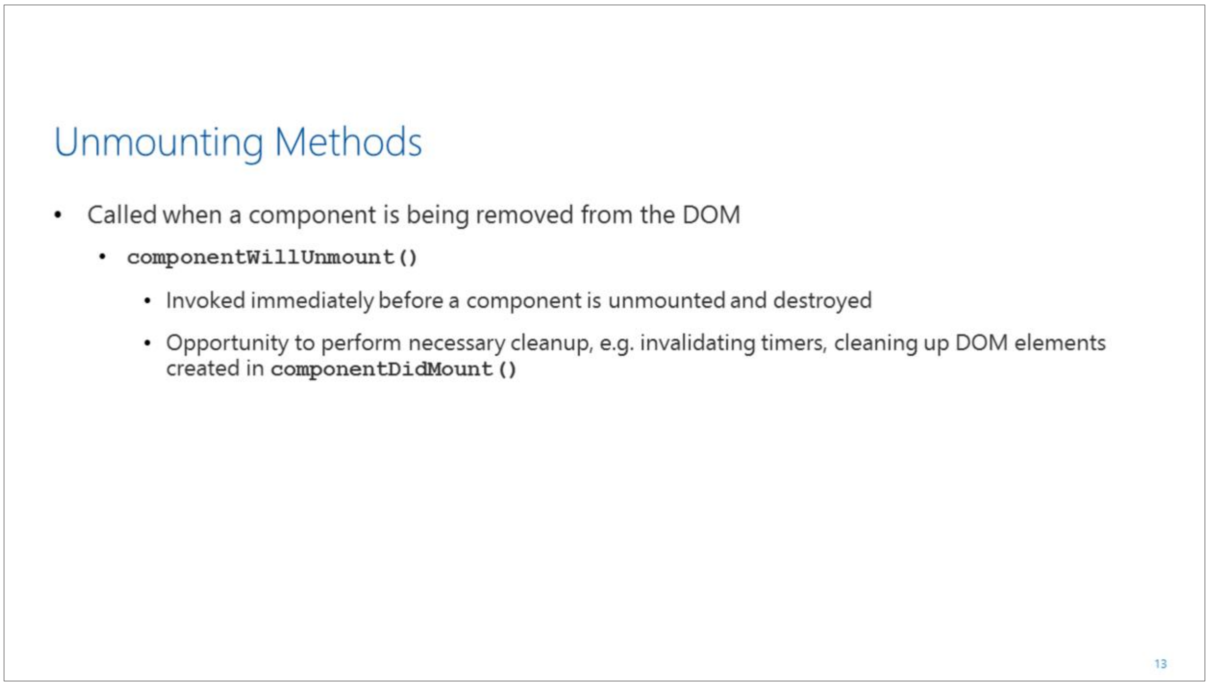
Updating Methods

- Called when a component is being re-rendered after changes to **`props`** or **`state`** (through the calling of **`setState()`**) or if **`forceUpdate()`** is called
  - **`componentDidUpdate(prevProps, prevState)`**
    - Invoked immediately after an update occurs
    - Opportunity to operate on the DOM after a component update
    - Good place to do network requests
      - Compare current props to previous props as network request may not be necessary if props have not changed
    - Not called during the component's initial render
    - Not invoked if **`shouldComponentUpdate()`** returns **`false`**

11



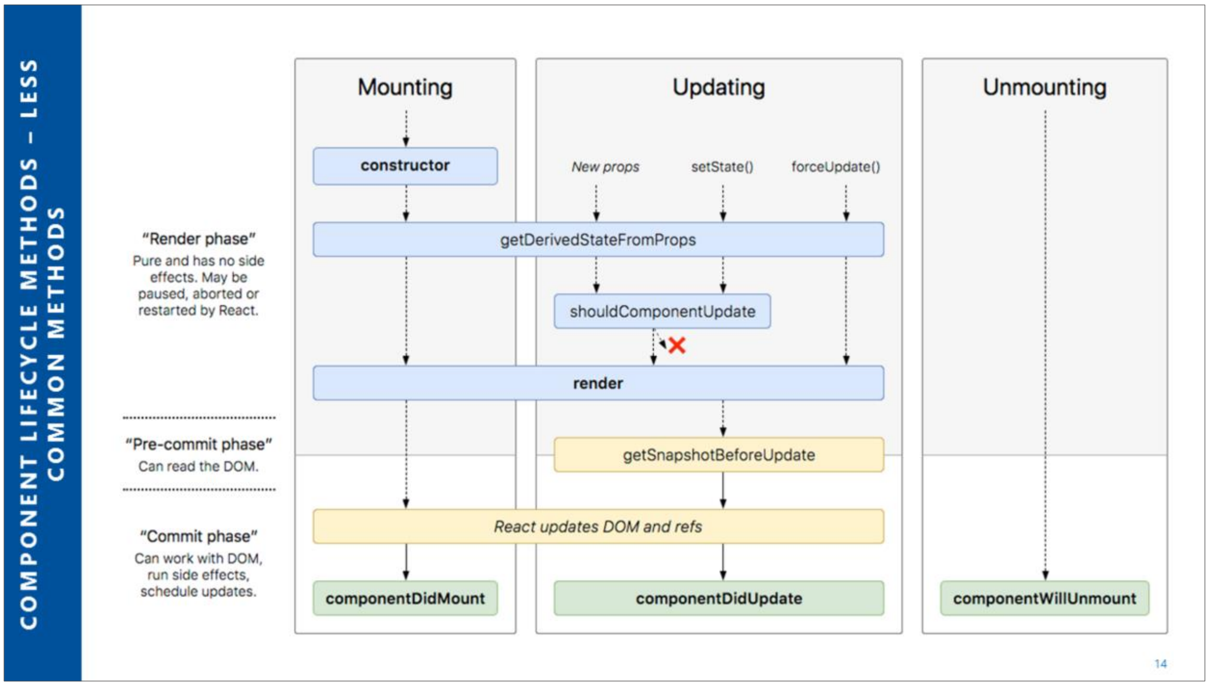
UPDATING METHODS – STATE CHANGE CALL ORDER



Unmounting Methods

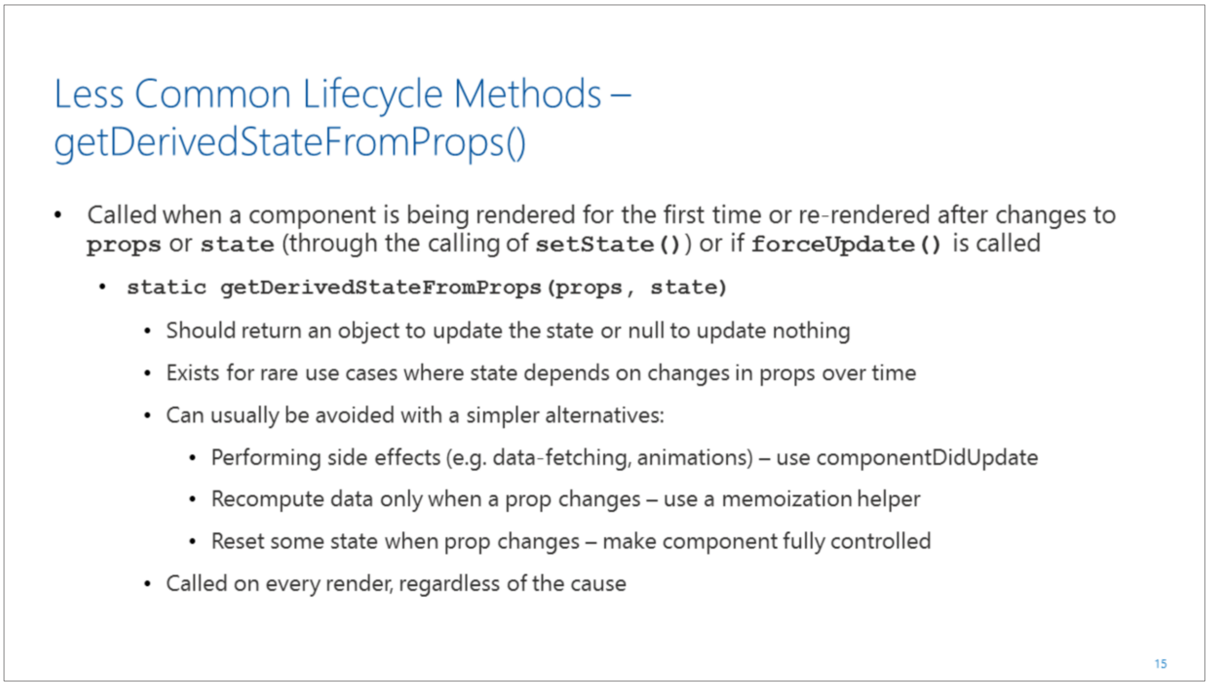
- Called when a component is being removed from the DOM
  - **`componentWillUnmount()`**
    - Invoked immediately before a component is unmounted and destroyed
    - Opportunity to perform necessary cleanup, e.g. invalidating timers, cleaning up DOM elements created in **`componentDidMount()`**

13



COMPONENT LIFECYCLE METHODS – LESS COMMON METHODS

Image from: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

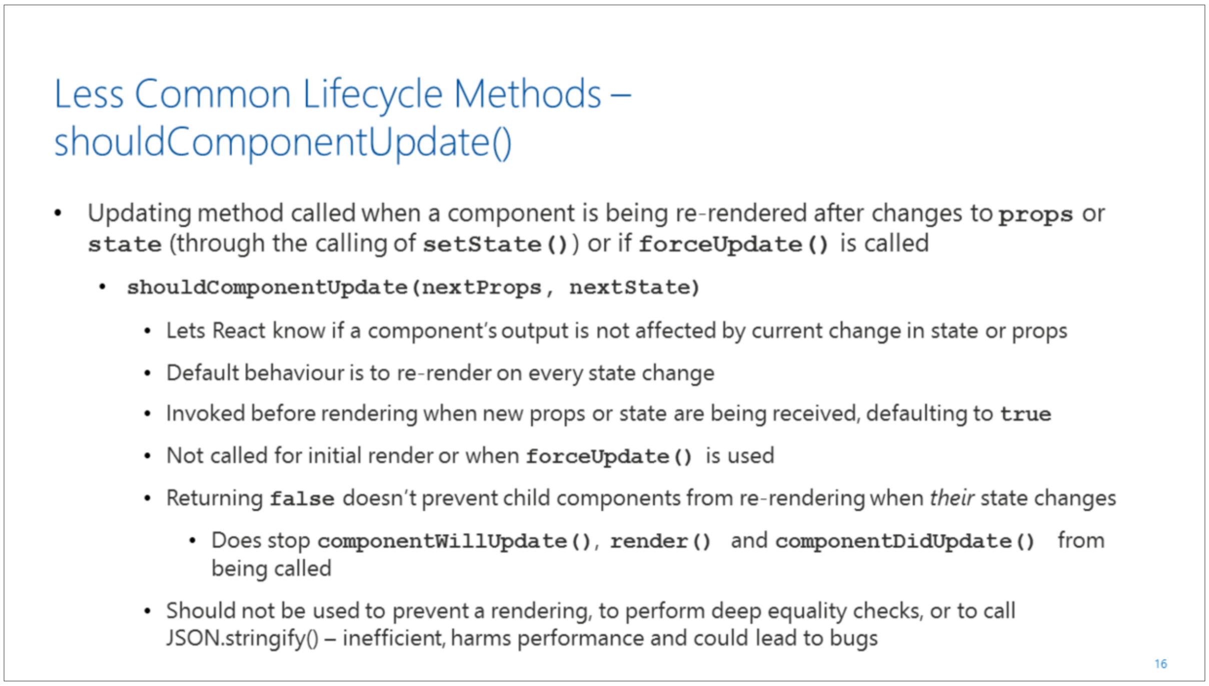


Less Common Lifecycle Methods – getDerivedStateFromProps()

- Called when a component is being rendered for the first time or re-rendered after changes to **`props`** or **`state`** (through the calling of **`setState()`**) or if **`forceUpdate()`** is called
  - **`static getDerivedStateFromProps(props, state)`**
    - Should return an object to update the state or null to update nothing
    - Exists for rare use cases where state depends on changes in props over time
    - Can usually be avoided with a simpler alternatives:
      - Performing side effects (e.g. data-fetching, animations) – use componentDidUpdate
      - Recompute data only when a prop changes – use a memoization helper
      - Reset some state when prop changes – make component fully controlled
    - Called on every render, regardless of the cause

15

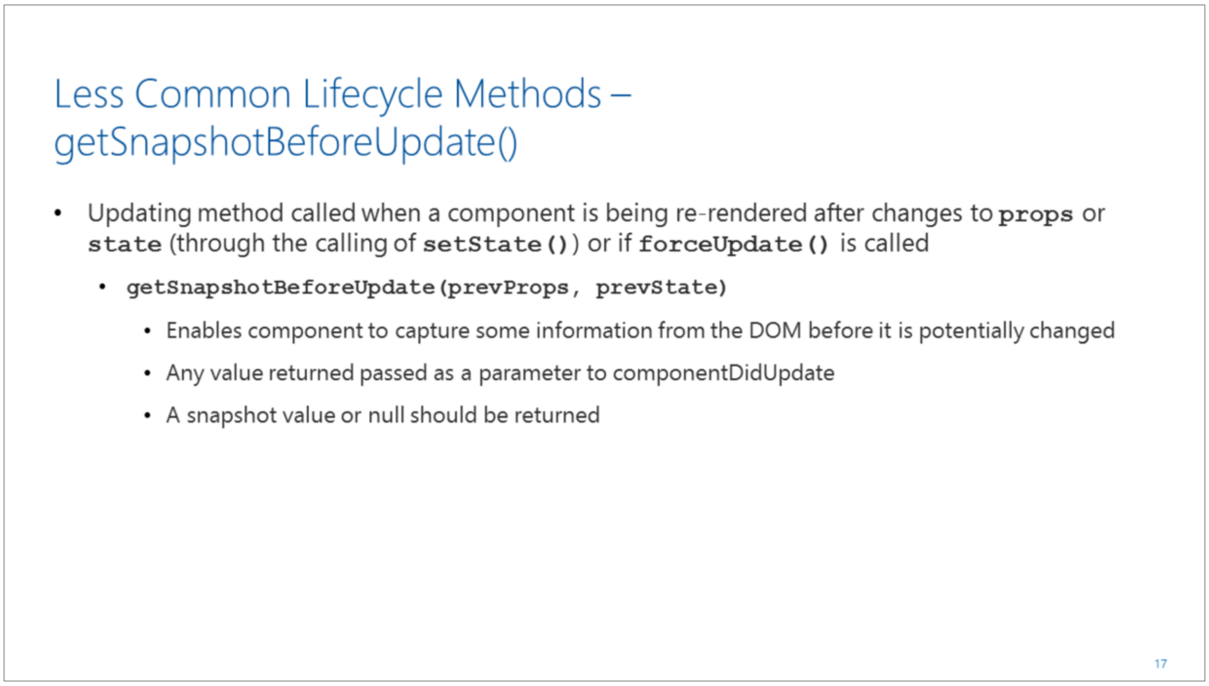
More information on memoization can be found at: https://reactjs.org/blog/2018/06/07/you-probably-dont-need-derived-state.html#what-about-memoization



Less Common Lifecycle Methods – shouldComponentUpdate()

- Updating method called when a component is being re-rendered after changes to **`props`** or **`state`** (through the calling of **`setState()`**) or if **`forceUpdate()`** is called
  - **`shouldComponentUpdate(nextProps, nextState)`**
    - Lets React know if a component's output is not affected by current change in state or props
    - Default behaviour is to re-render on every state change
    - Invoked before rendering when new props or state are being received, defaulting to **`true`**
    - Not called for initial render or when **`forceUpdate()`** is used
    - Returning **`false`** doesn't prevent child components from re-rendering when *their* state changes
      - Does stop **`componentWillUpdate()`**, **`render()`** and **`componentDidUpdate()`** from being called
    - Should not be used to prevent a rendering, to perform deep equality checks, or to call JSON.stringify() – inefficient, harms performance and could lead to bugs

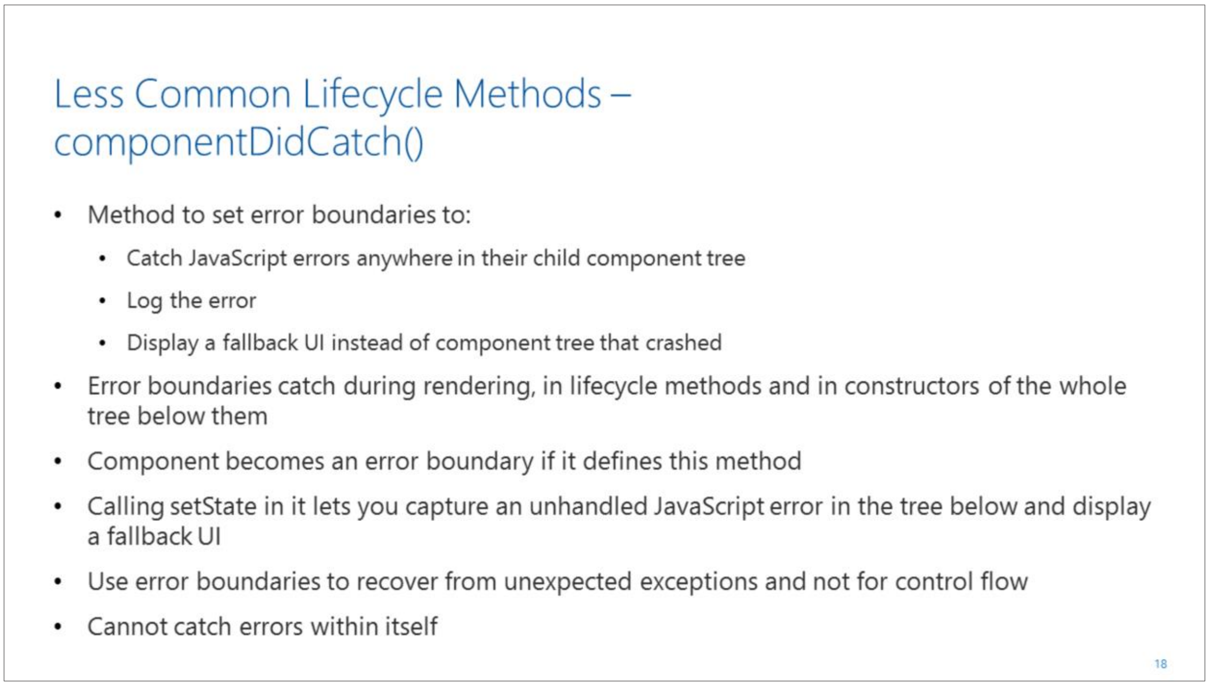
16



Less Common Lifecycle Methods – getSnapshotBeforeUpdate()

- Updating method called when a component is being re-rendered after changes to **`props`** or **`state`** (through the calling of **`setState()`**) or if **`forceUpdate()`** is called
  - **`getSnapshotBeforeUpdate(prevProps, prevState)`**
    - Enables component to capture some information from the DOM before it is potentially changed
    - Any value returned passed as a parameter to componentDidUpdate
    - A snapshot value or null should be returned

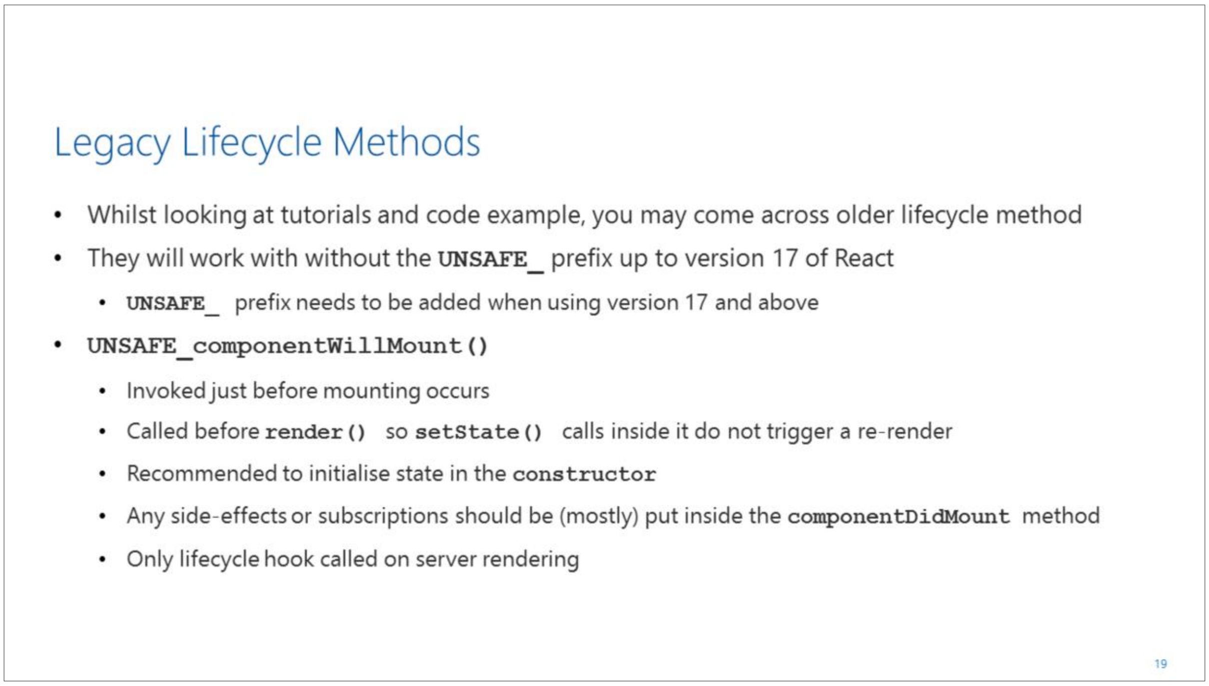
17



Less Common Lifecycle Methods – componentDidCatch()

- Method to set error boundaries to:
  - Catch JavaScript errors anywhere in their child component tree
  - Log the error
  - Display a fallback UI instead of component tree that crashed
- Error boundaries catch during rendering, in lifecycle methods and in constructors of the whole tree below them
- Component becomes an error boundary if it defines this method
- Calling setState in it lets you capture an unhandled JavaScript error in the tree below and display a fallback UI
- Use error boundaries to recover from unexpected exceptions and not for control flow
- Cannot catch errors within itself

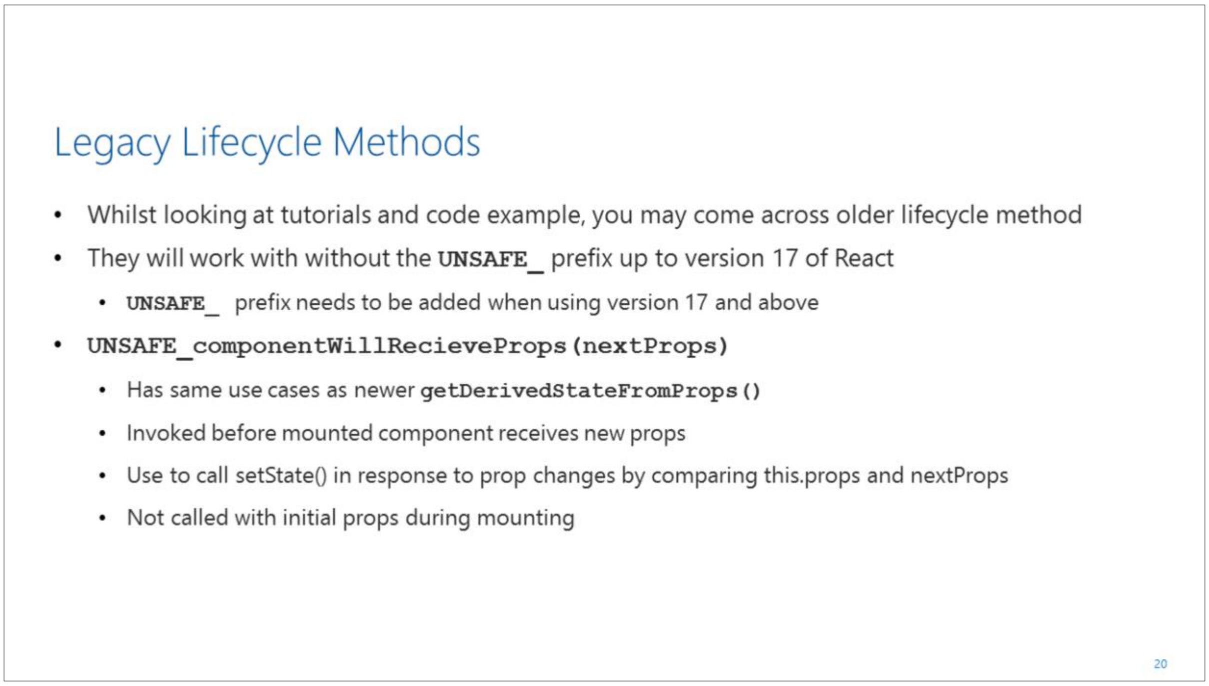
18



Legacy Lifecycle Methods

- Whilst looking at tutorials and code example, you may come across older lifecycle method
- They will work with without the **`UNSAFE_`** prefix up to version 17 of React
  - **`UNSAFE_`** prefix needs to be added when using version 17 and above
- **`UNSAFE_componentWillMount()`**
  - Invoked just before mounting occurs
  - Called before **`render()`** so **`setState()`** calls inside it do not trigger a re-render
  - Recommended to initialise state in the **`constructor`**
  - Any side-effects or subscriptions should be (mostly) put inside the **`componentDidMount`** method
  - Only lifecycle hook called on server rendering

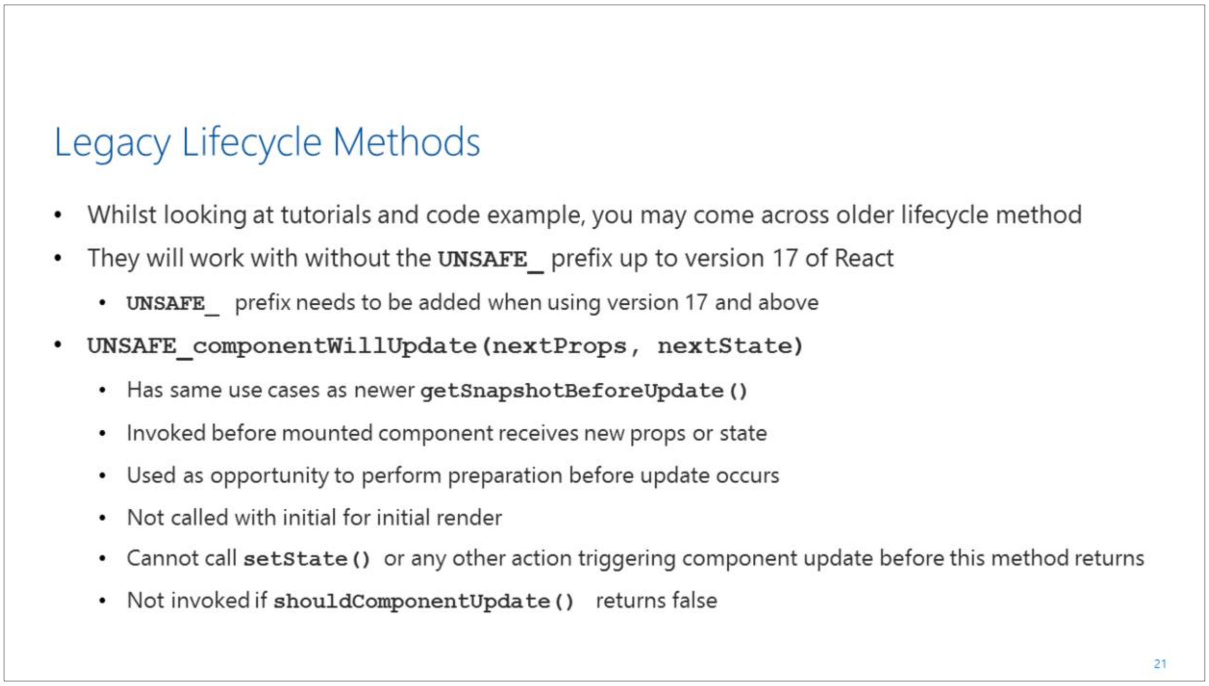
19



Legacy Lifecycle Methods

- Whilst looking at tutorials and code example, you may come across older lifecycle method
- They will work with without the **`UNSAFE_`** prefix up to version 17 of React
  - **`UNSAFE_`** prefix needs to be added when using version 17 and above
- **`UNSAFE_componentWillRecieveProps(nextProps)`**
  - Has same use cases as newer **`getDerivedStateFromProps()`**
  - Invoked before mounted component receives new props
  - Use to call setState() in response to prop changes by comparing this.props and nextProps
  - Not called with initial props during mounting

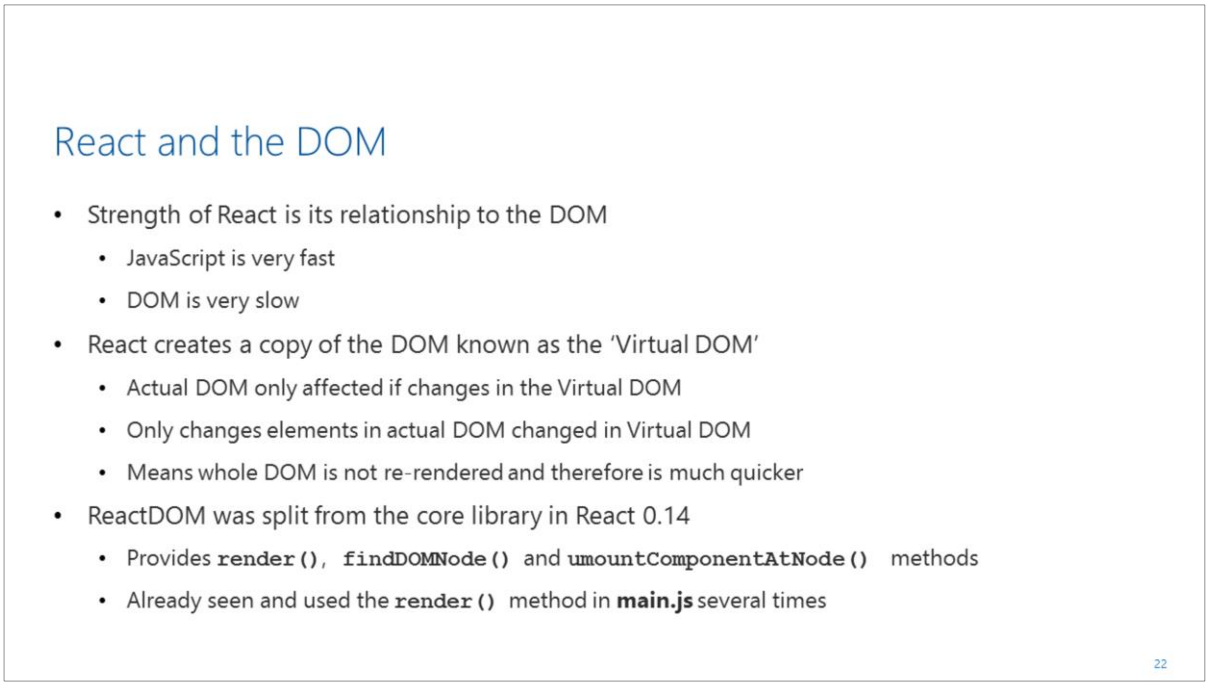
20



Legacy Lifecycle Methods

- Whilst looking at tutorials and code example, you may come across older lifecycle method
- They will work with without the **`UNSAFE_`** prefix up to version 17 of React
  - **`UNSAFE_`** prefix needs to be added when using version 17 and above
- **`UNSAFE_componentWillUpdate(nextProps, nextState)`**
  - Has same use cases as newer **`getSnapshotBeforeUpdate()`**
  - Invoked before mounted component receives new props or state
  - Used as opportunity to perform preparation before update occurs
  - Not called with initial for initial render
  - Cannot call **`setState()`** or any other action triggering component update before this method returns
  - Not invoked if **`shouldComponentUpdate()`** returns false

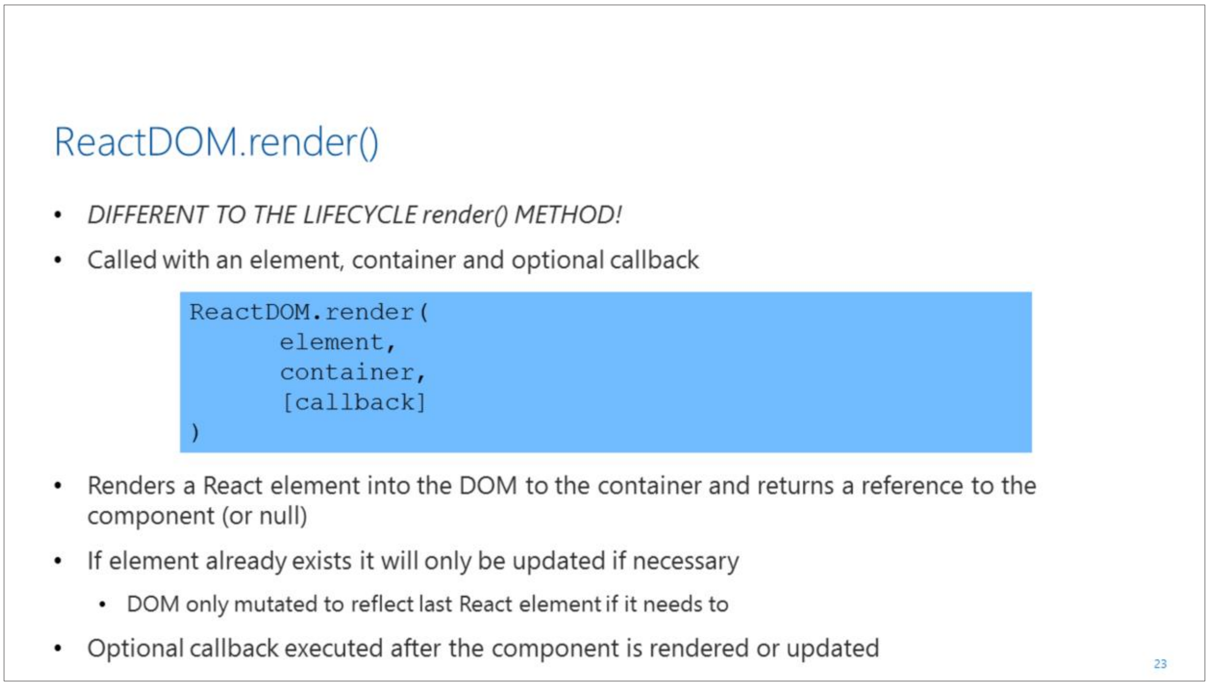
21



React and the DOM

- Strength of React is its relationship to the DOM
  - JavaScript is very fast
  - DOM is very slow
- React creates a copy of the DOM known as the 'Virtual DOM'
  - Actual DOM only affected if changes in the Virtual DOM
  - Only changes elements in actual DOM changed in Virtual DOM
  - Means whole DOM is not re-rendered and therefore is much quicker
- ReactDOM was split from the core library in React 0.14
  - Provides **`render()`**, **`findDOMNode()`** and **`umountComponentAtNode()`** methods
  - Already seen and used the **`render()`** method in **main.js** several times

22



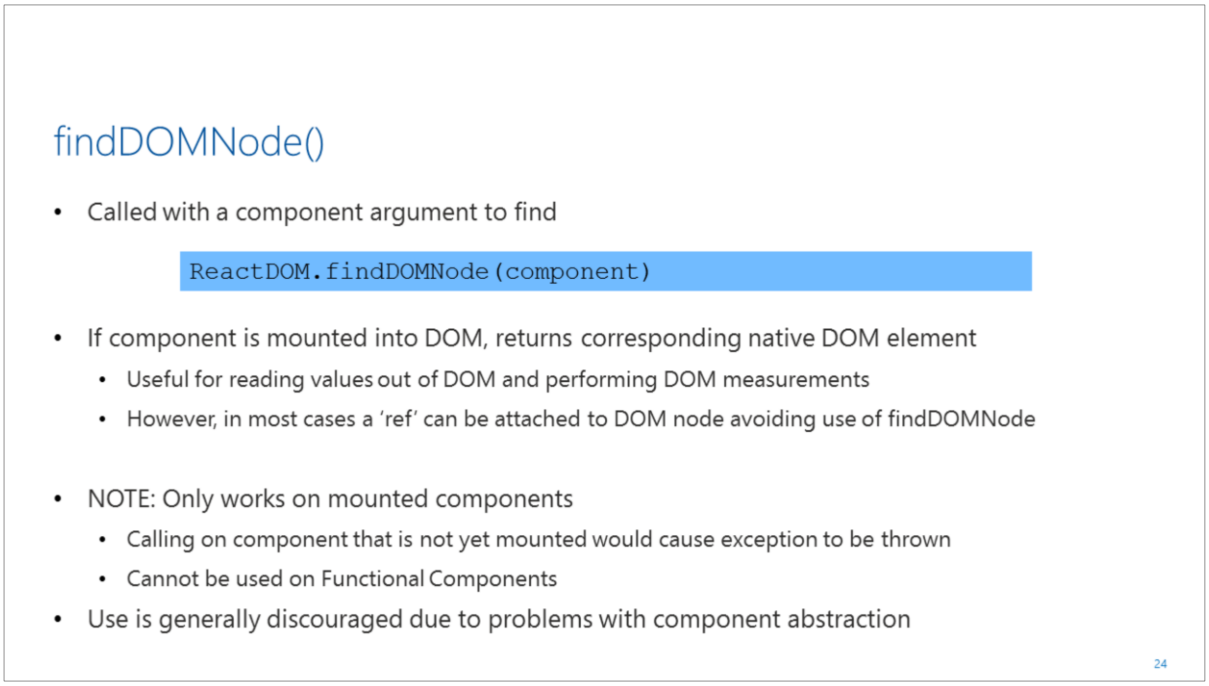
ReactDOM.render()

- *DIFFERENT TO THE LIFECYCLE render() METHOD!*
- Called with an element, container and optional callback

```
ReactDOM.render(
      element,
      container,
      [callback]
)
```

- Renders a React element into the DOM to the container and returns a reference to the component (or null)
- If element already exists it will only be updated if necessary
  - DOM only mutated to reflect last React element if it needs to
- Optional callback executed after the component is rendered or updated

23



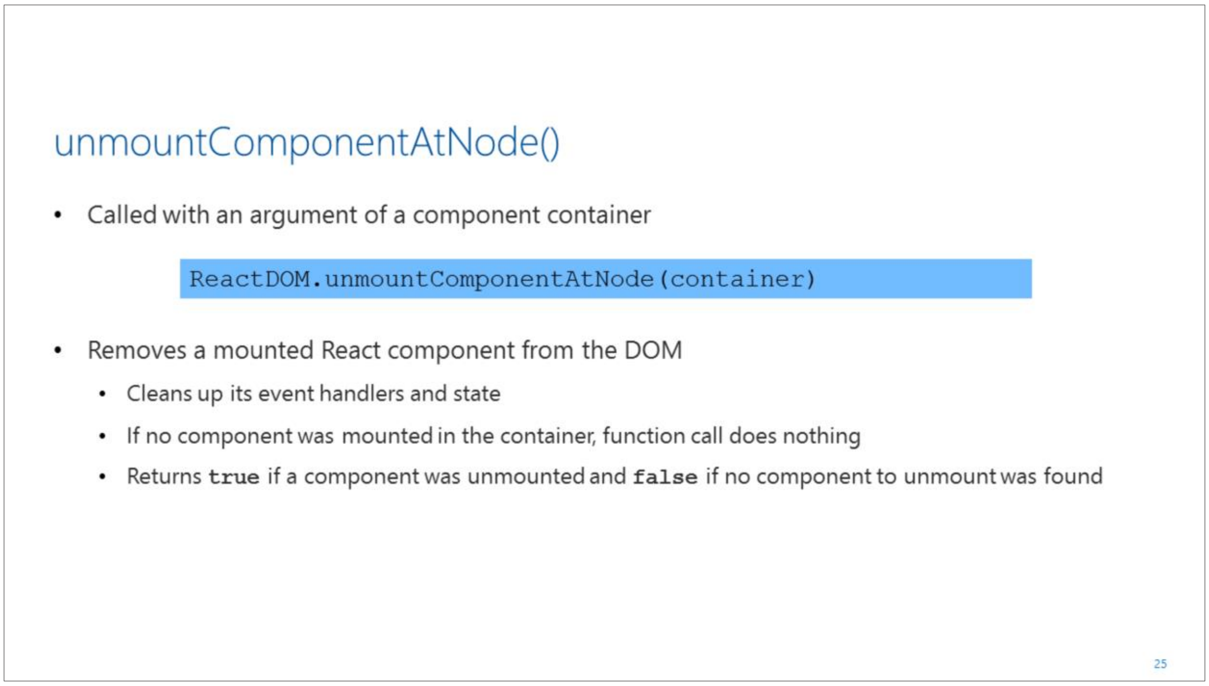
findDOMNode()

- Called with a component argument to find

```
ReactDOM.findDOMNode(component)
```

- If component is mounted into DOM, returns corresponding native DOM element
  - Useful for reading values out of DOM and performing DOM measurements
  - However, in most cases a 'ref' can be attached to DOM node avoiding use of findDOMNode
- NOTE: Only works on mounted components
  - Calling on component that is not yet mounted would cause exception to be thrown
  - Cannot be used on Functional Components
- Use is generally discouraged due to problems with component abstraction

24



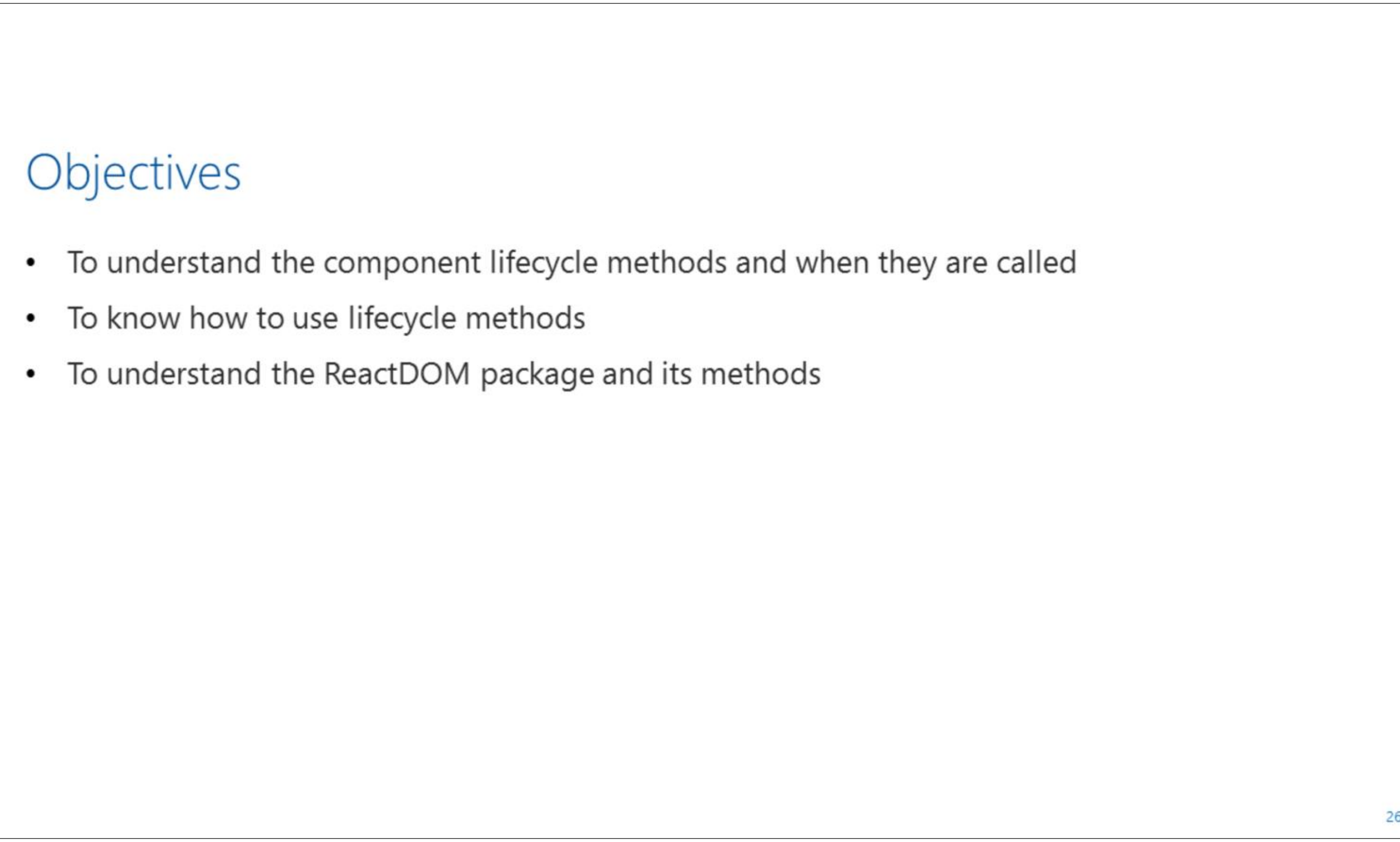
unmountComponentAtNode()

- Called with an argument of a component container

```
ReactDOM.unmountComponentAtNode(container)
```

- Removes a mounted React component from the DOM
  - Cleans up its event handlers and state
  - If no component was mounted in the container, function call does nothing
  - Returns **`true`** if a component was unmounted and **`false`** if no component to unmount was found

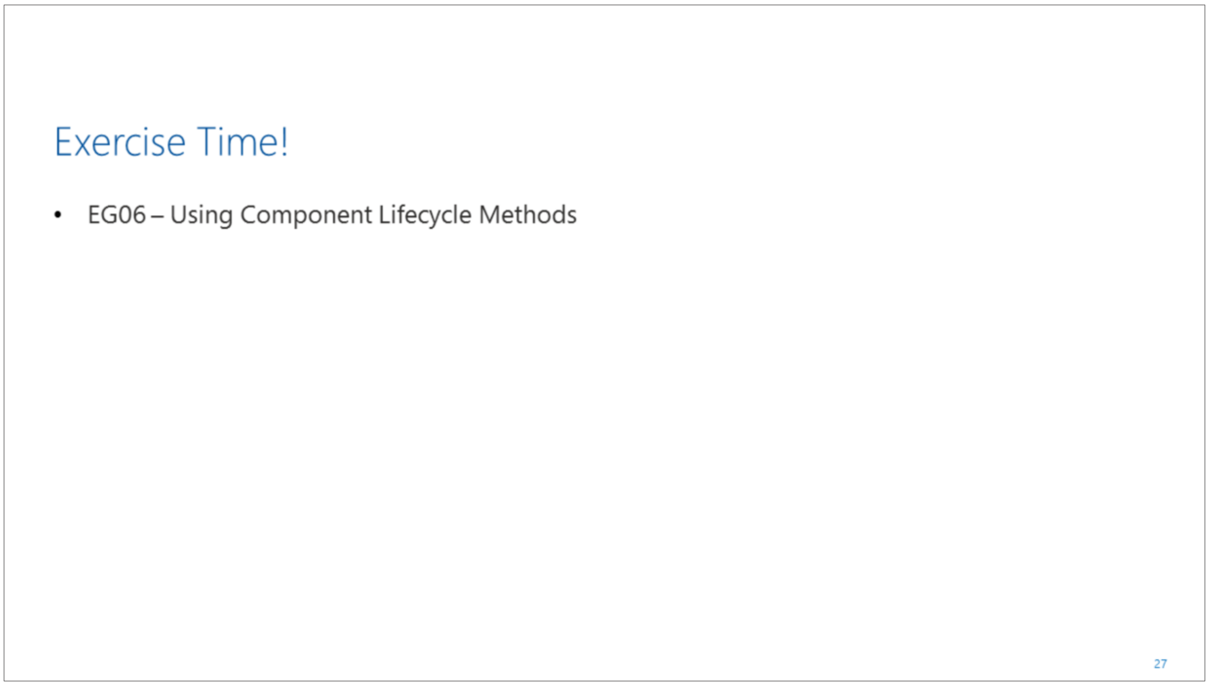
25



Objectives

- To understand the component lifecycle methods and when they are called
- To know how to use lifecycle methods
- To understand the ReactDOM package and its methods

26



Exercise Time!

- EG06 – Using Component Lifecycle Methods

27