sourcemind

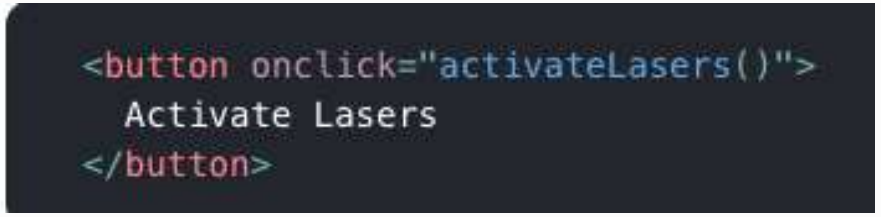
REACT JS

EVENTS, COMPONENT LIFECYCLE

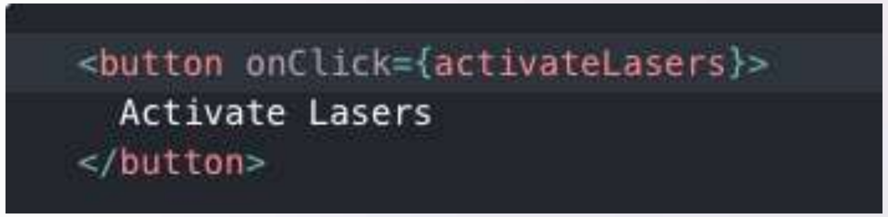
React JS

Handling Events

Handling events with React elements is very similar to handling events on DOM elements. There are some syntax differences:



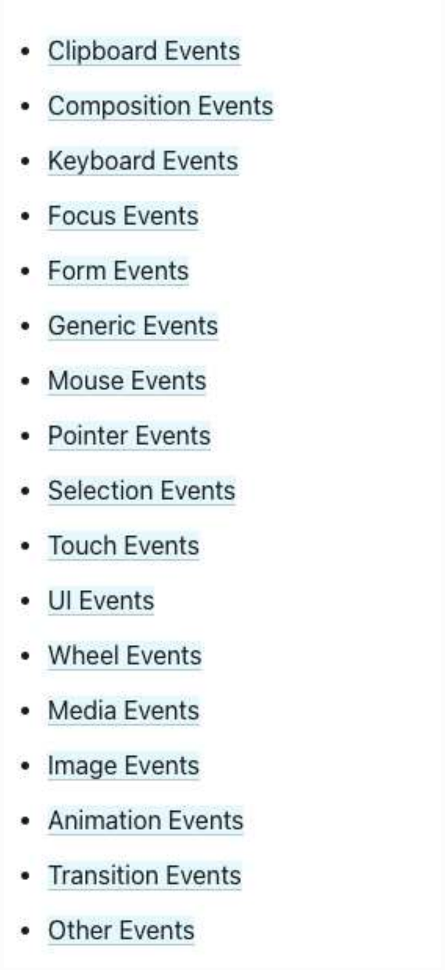
```html
<button onclick="activateLasers()">
  Activate Lasers
</button>
```



```jsx
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

SyntheticEvent

This reference guide documents the SyntheticEvent wrapper that forms part of React’s Event System.



- Clipboard Events
- Composition Events
- Keyboard Events
- Focus Events
- Form Events
- Generic Events
- Mouse Events
- Pointer Events
- Selection Events
- Touch Events
- UI Events
- Wheel Events
- Media Events
- Image Events
- Animation Events
- Transition Events
- Other Events

Clipboard Events

```
onCopy onCut onPaste
```

Keyboard Events

```
onKeyDown onKeyPress onKeyUp
```

Focus Events

```
onFocus onBlur
```

sourcemind

Form Events

```
onChange onInput onInvalid onReset onSubmit
```

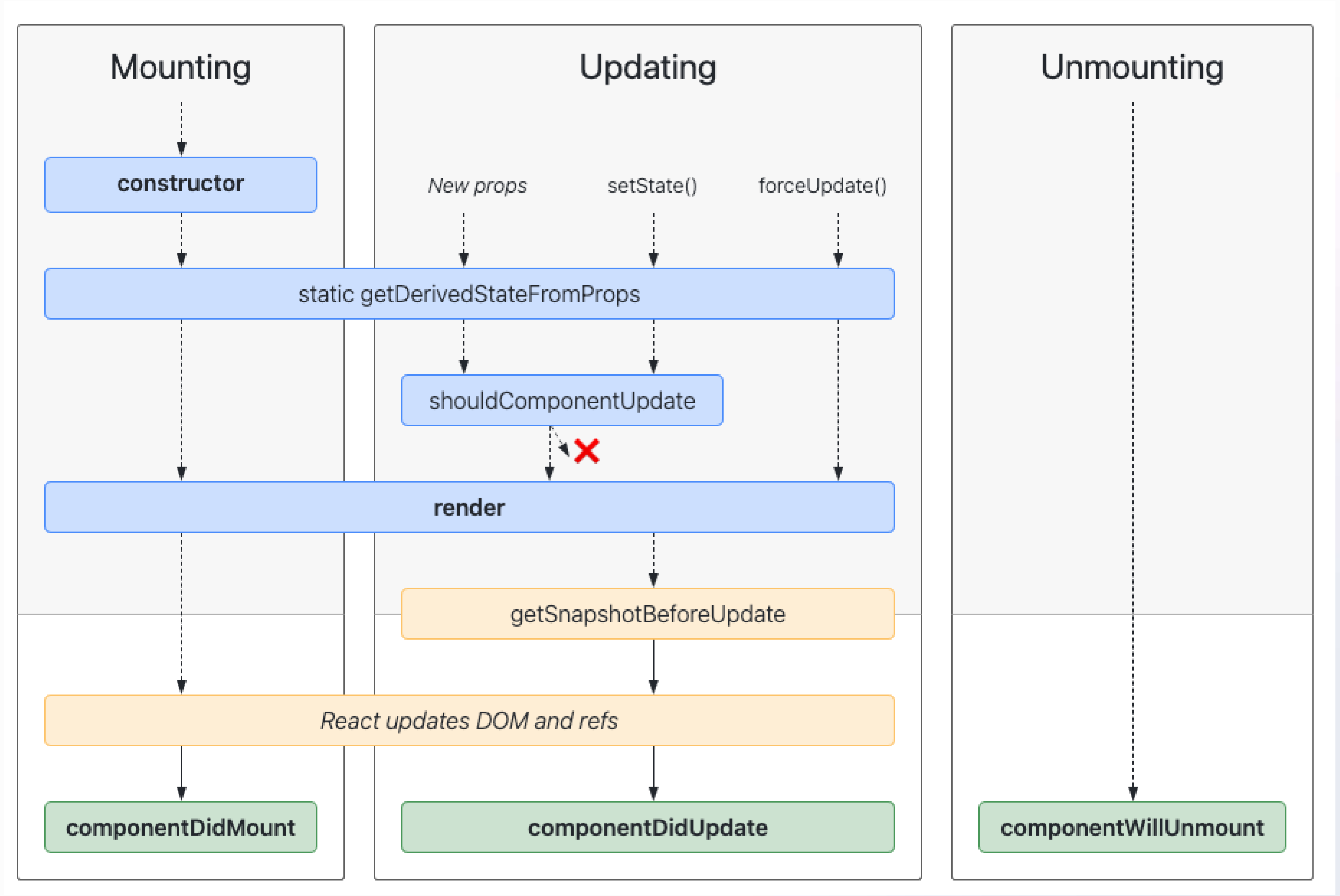
Generic Events

```
onError onLoad
```

Mouse Events

```
onClick onContextMenu onDoubleClick onDrag onDragEnd onDragEnter onDragExit
onDragLeave onDragOver onDragStart onDrop onMouseDown onMouseEnter onMouseLeave
onMouseMove onMouseOut onMouseOver onMouseUp
```

React Lifecycle Methods



sourcemind

render()

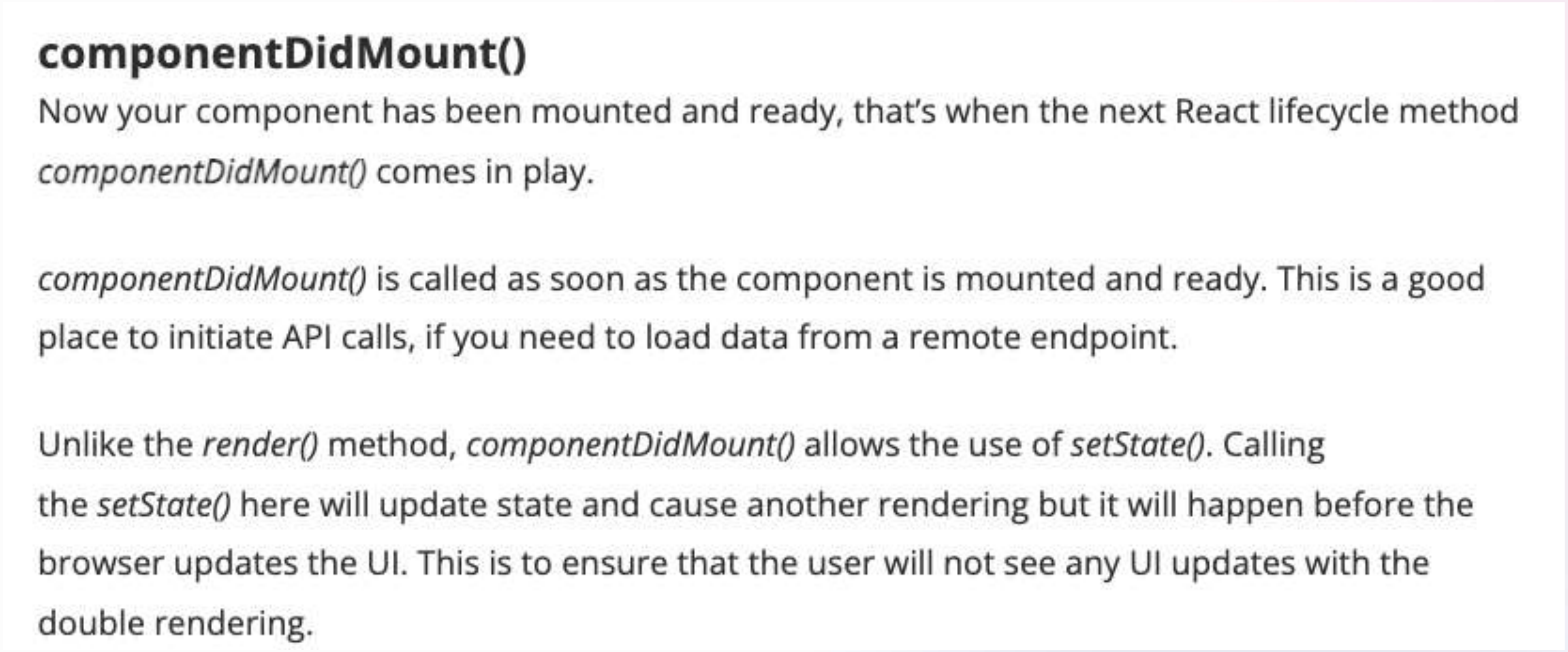
The *render()* method is the most used lifecycle method. You will see it in all React classes. This is because *render()* is the only required method within a class component in React.

As the name suggests it handles the rendering of your component to the UI. It happens during the **mounting** and **updating** of your component.

Below is an example of a simple *render()* in React.

```
class Hello extends Component{
   render(){
      return <div>Hello {this.props.name}</div>
   }
}
```

sourcemind

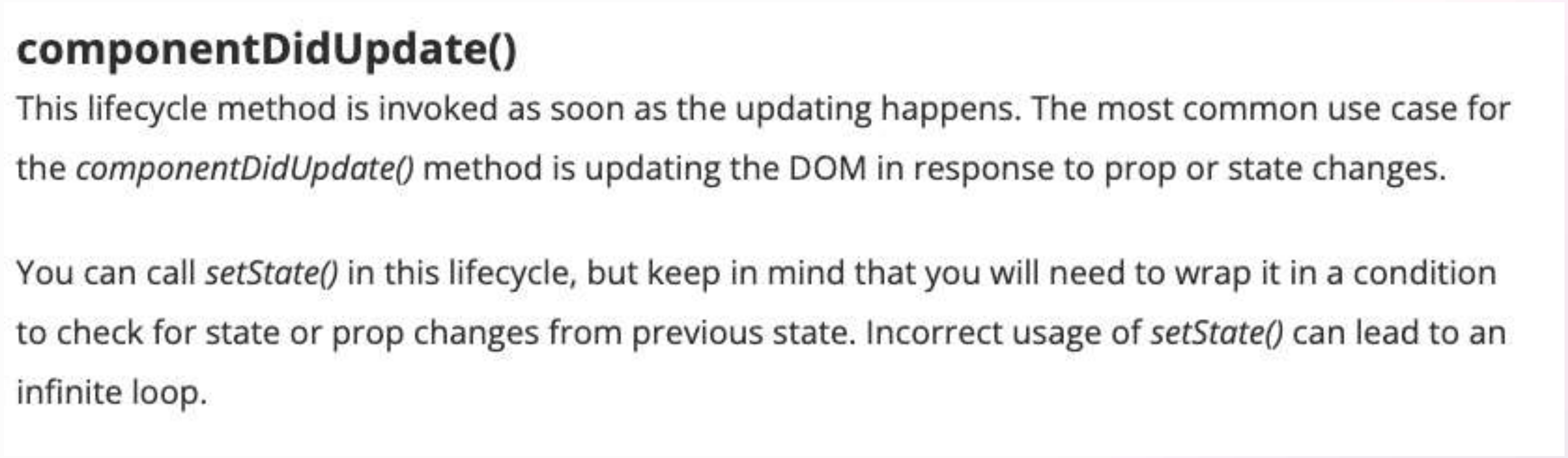


componentDidMount()

Now your component has been mounted and ready, that's when the next React lifecycle method *componentDidMount()* comes in play.

componentDidMount() is called as soon as the component is mounted and ready. This is a good place to initiate API calls, if you need to load data from a remote endpoint.

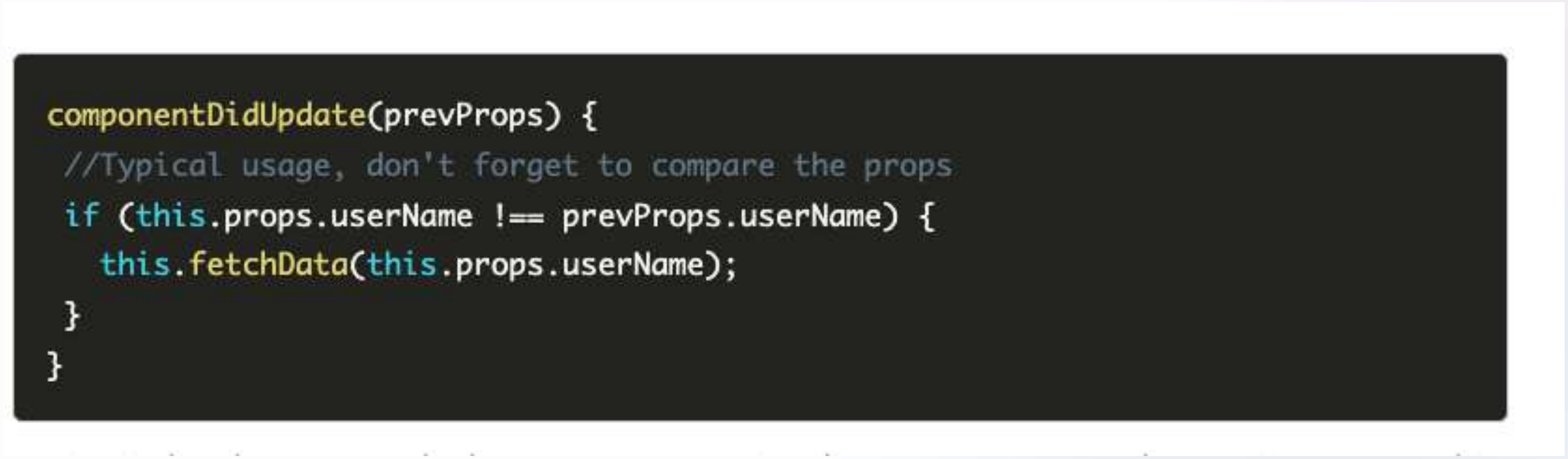
Unlike the *render()* method, *componentDidMount()* allows the use of *setState()*. Calling the *setState()* here will update state and cause another rendering but it will happen before the browser updates the UI. This is to ensure that the user will not see any UI updates with the double rendering.



componentDidUpdate()

This lifecycle method is invoked as soon as the updating happens. The most common use case for the *componentDidUpdate()* method is updating the DOM in response to prop or state changes.

You can call *setState()* in this lifecycle, but keep in mind that you will need to wrap it in a condition to check for state or prop changes from previous state. Incorrect usage of *setState()* can lead to an infinite loop.



```
componentDidUpdate(prevProps) {
 //Typical usage, don't forget to compare the props
 if (this.props.userName !== prevProps.userName) {
   this.fetchData(this.props.userName);
 }
}
```

componentWillUnmount()

As the name suggests this lifecycle method is called just before the component is unmounted and destroyed. If there are any cleanup actions that you would need to do, this would be the right spot.

```
componentWillUnmount() {
 window.removeEventListener('resize', this.resizeListener)
}
```

sourcemind

shouldComponentUpdate()

This lifecycle can be handy sometimes when you don’t want React to render your state or prop changes.

Anytime *setState()* is called, the component re-renders by default. The *shouldComponentUpdate()* method is used to let React know if a component is not affected by the state and prop changes.

```
shouldComponentUpdate(nextProps, nextState) {
 return this.props.title !== nextProps.title ||
  this.state.input !== nextState.input }
```

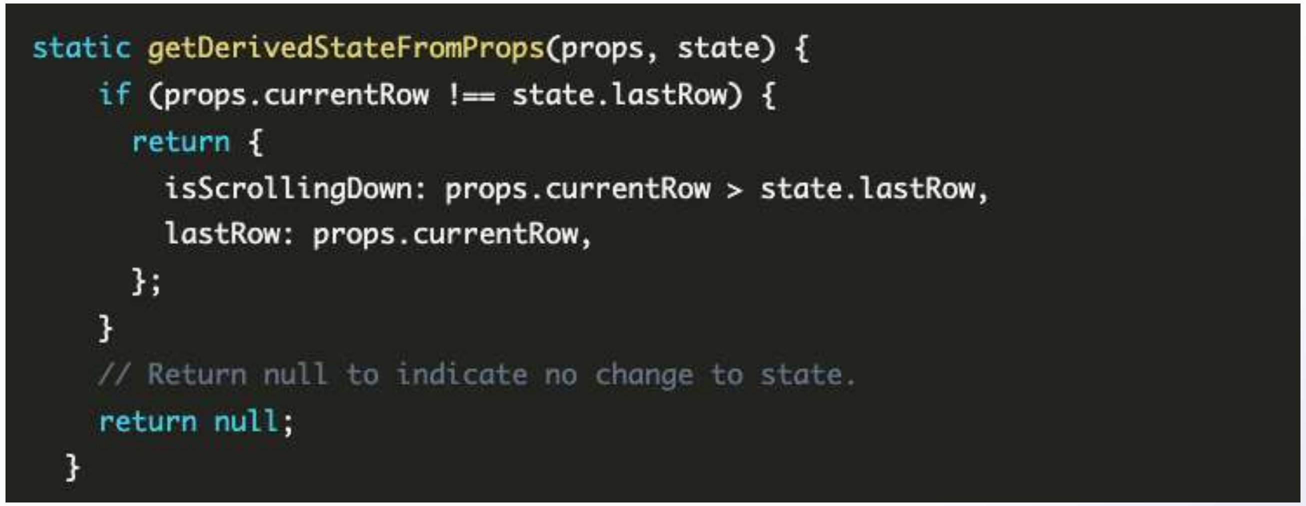
static getDerivedStateFromProps()

This is one of the newer lifecycle methods introduced very recently by the React team.

This will be a safer alternative to the previous lifecycle method *componentWillReceiveProps().*

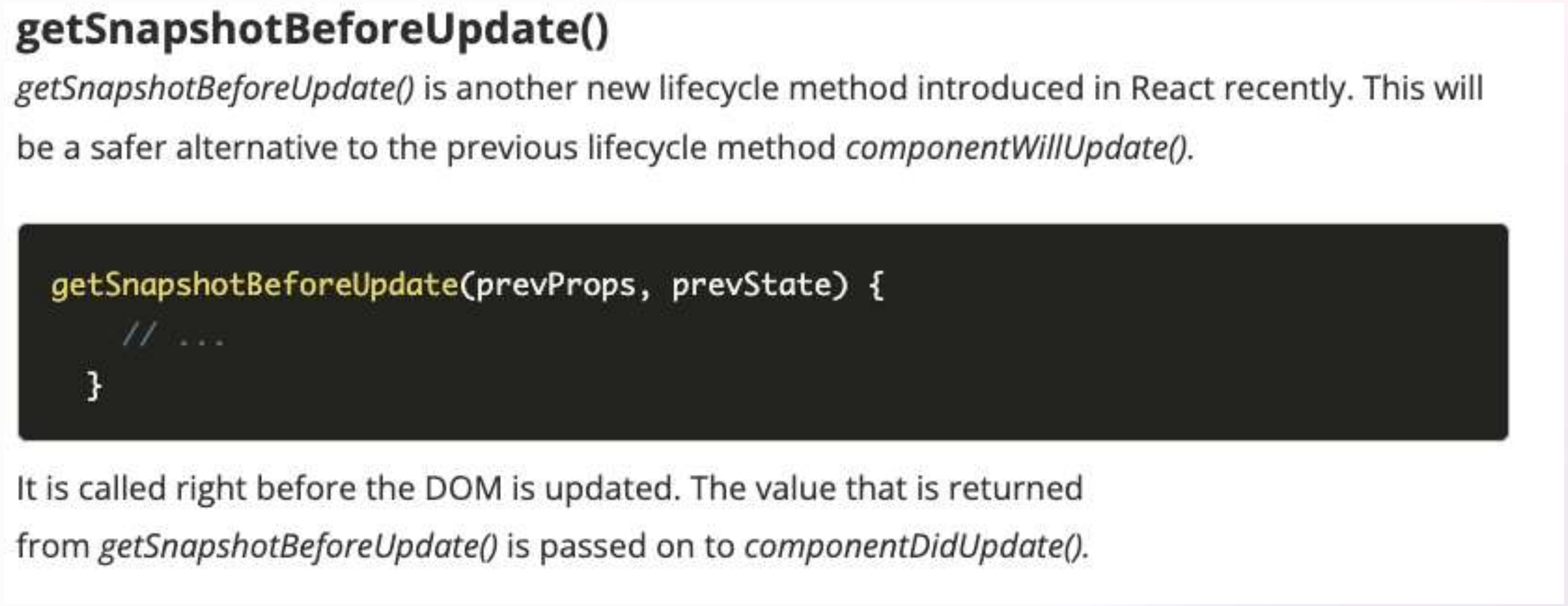
It is called just before calling the *render*() method.

This is a *static* function that does not have access to "*this*". *getDerivedStateFromProps()* returns an object to update *state* in response to *prop* changes. It can return a *null* if there is no change to state.



```
static getDerivedStateFromProps(props, state) {
    if (props.currentRow !== state.lastRow) {
      return {
        isScrollingDown: props.currentRow > state.lastRow,
        lastRow: props.currentRow,
      };
    }
    // Return null to indicate no change to state.
    return null;
  }
```

sourcemind



getSnapshotBeforeUpdate()

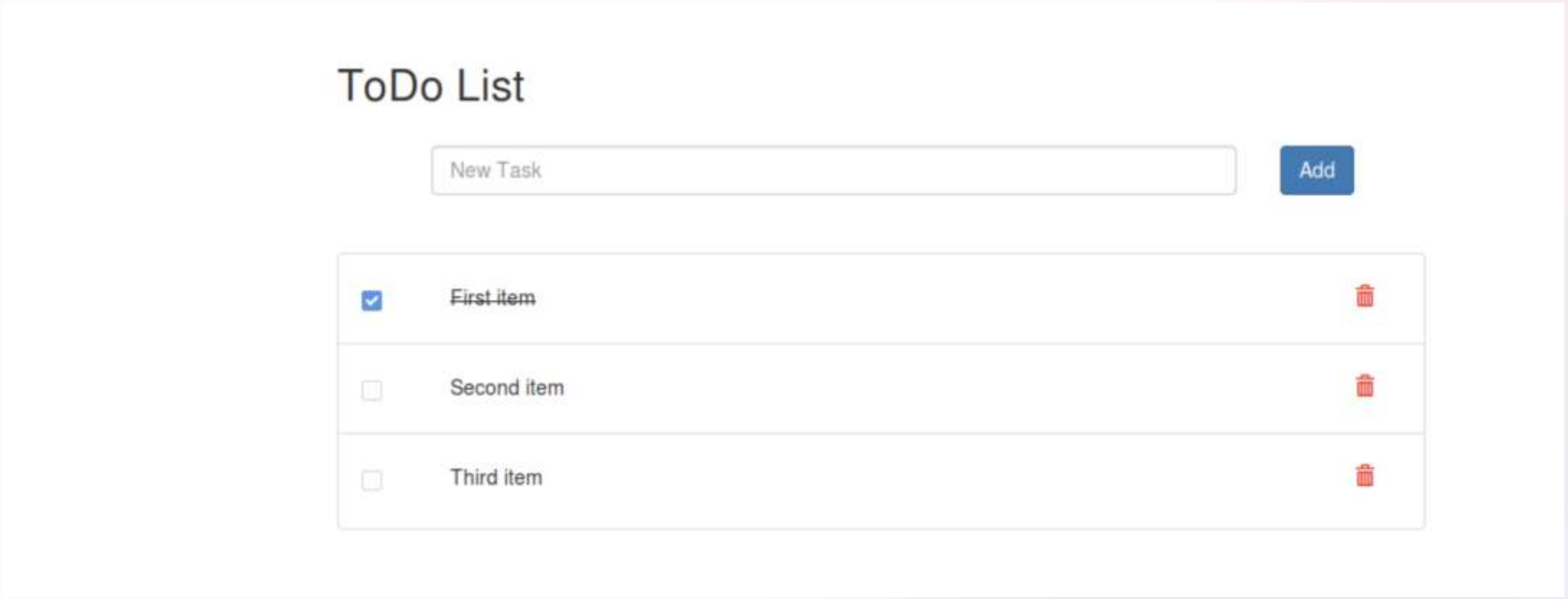
getSnapshotBeforeUpdate() is another new lifecycle method introduced in React recently. This will be a safer alternative to the previous lifecycle method *componentWillUpdate()*.

```
getSnapshotBeforeUpdate(prevProps, prevState) {
    // ...
  }
```

It is called right before the DOM is updated. The value that is returned from *getSnapshotBeforeUpdate()* is passed on to *componentDidUpdate()*.

sourcemind

TASKS


ToDo List

- [x] ~~First item~~
- [ ] Second item
- [ ] Third item

sourcemind