React Lifecycle Triggers and Events

Component Lifecycle

- From when a component is invoked to when it is destroyed, it goes through a series of lifecycle events
- These functions give you the opportunity to make decisions and take appropriate actions.
- There are four triggers that kick off these lifecycle events. From these triggers, we will examine the most commonly used lifecycle methods.

Lifecycle Event Triggers

- Initialization
- Updating State
- Updating Props
- Unmounting

Trigger: Initialization

- The most commonly used lifecycle events triggered on initialization are:
    - `render()`
    - `constructor()` (if used)
    - `componentDidMount()` (one of the most used methods)

Trigger: Initialization

- `render()` returns the component markup, which can be a single child component, a set of components, or null or false (in case you don’t want anything rendering)

Trigger: Initialization

- `constructor(props)` is not necessary if you do not initialize state and/or you do not bind methods to a component.
- Called before a component is mounted.
- Should call `super(props)` if using `constructor` before any other statement, otherwise `this.props` will be undefined in the `constructor` which can lead to bugs)
- Typically, `constructors` are used for two purposes:
    - Initialize local state by assigning an object to this.state
    - Binding event handler methods to an instance

Trigger: Initialization

- `componentDidMount()` is called once immediately after initial rendering has occurred
- The DOM is now available at this point
- This is where you'll want to use things like `setInterval()`, `setTimeout()`, and some AJAX requests

Trigger: Updating State or Props

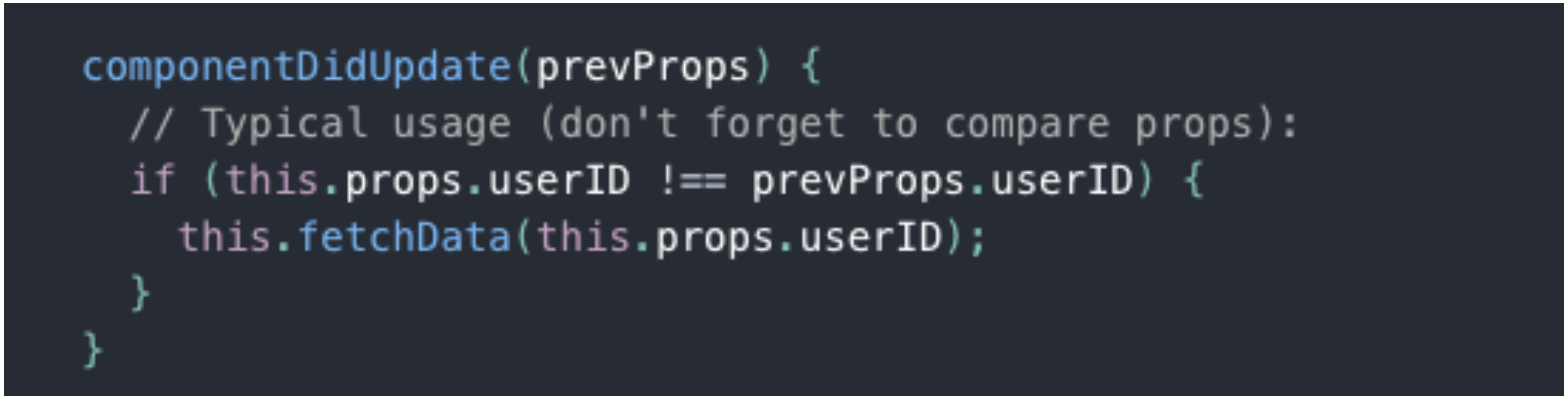
- The most commonly used lifecycle events triggered on Updating State or Props are:
  - `render()`
  - `componentDidUpdate()`

Trigger: Updating State or Props

- `componentDidUpdate()` has access to three properties, two of which are leveraged more than the third:
  - `prevProps`
  - `prevState`
  - `snapshot` (rarely used, typically undefined)
- `componentDidUpdate()` is invoked immediately after updating occurs, and is not called for the initial render.
- Use this as an opportunity to operate on the DOM when the component has been updated.

Trigger: Updating State or Props

- Can also do network requests as long as you compare current props to previous props, as a network request may not be necessary if props haven't changed



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

Trigger: Updating State or Props

- You can call `setState()` immediately in a componentDidUpdate, however make sure to wrap it in a conditional statement like in the previous example or you can cause an infinite loop.

- Updating state also causes a re-render, which may not be visible to the user but could affect the component performance.

Trigger: Unmounting

- `componentWillUnmount()` will be invoked immediately before a component is unmounted/removed from the DOM
- You can perform any necessary cleanup in this method, such as clearing timers, cancelling network requests, or cleaning up any subscriptions created in `componentDidMount()`.

Resources

- React.Component API: https://reactjs.org/docs/react-component.html
- Diagram of when lifecycle methods are used, along with a toggle to show where less used methods would be used: http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/