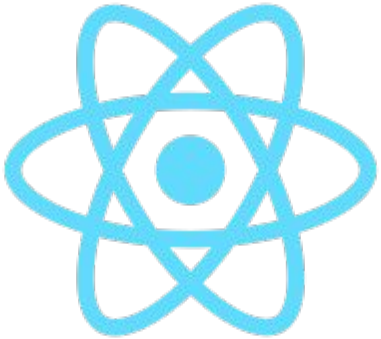
React Lifecycle



Triggers and Lifecycle Events

Component Lifecycle

- **Lifecycle events:** Components go through a series of events from creation to destruction. These events allow you to perform actions and make decisions at specific moments.

- **Action opportunities:** Each event provides a chance to take actions or make decisions based on the component's current state. This can include updating state, fetching data, or interacting with the UI.

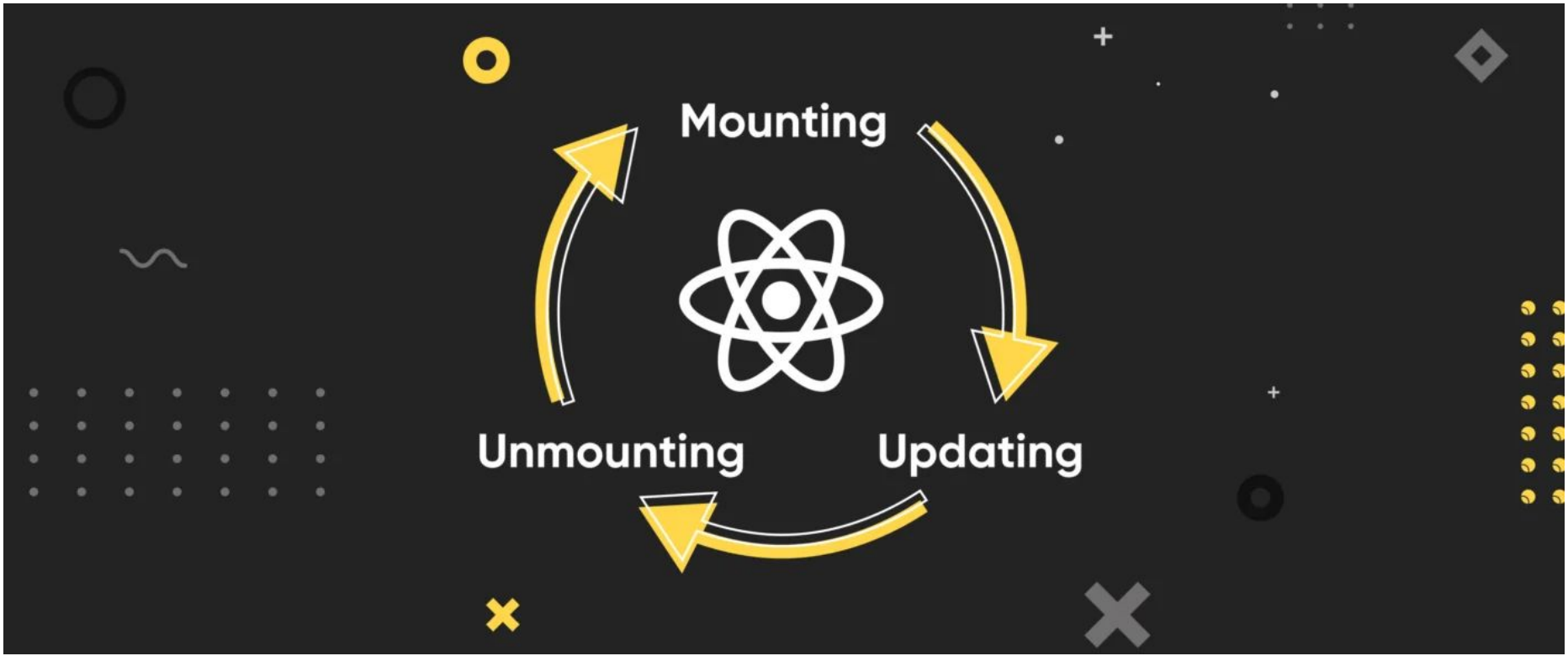
- **Triggers and common methods:** There are four triggers: component creation, updates, unmounting, and error handling. We focus on commonly used methods that hook into these events and let you perform actions accordingly.

Lifecycle Event Triggers

React Lifecycle

- Initialization
- Updating State
- Updating Props
- Unmounting

React Lifecycle



Mounting

- **constructor():** This is the first method called when a component is created.
  - Uses: Initializing state and binding event handlers.
- **render():** This method is responsible for rendering the component’s JSX and returns the component's UI.
- **componentDidMount():** It is invoked immediately after the component is mounted (i.e., inserted into the DOM).
  - Uses: For performing side effects, such as making API requests or setting up event listeners.

Updating

- **render():** The render method is called again to update the component's UI when there are changes to its props or state.
- **componentDidUpdate(prevProps, prevState):** This method is called after the component's update is reflected in the DOM.
  - Uses: To perform additional actions based on the updated props or state.
- **shouldComponentUpdate(nextProps, nextState):** It determines whether the component should re-render or not.
  - Uses: Custom logic here to optimize performance by preventing unnecessary re-renders.

Updating State and Props - componentDidUpdate()

- 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.
- 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);
  }
}
```

Updating State and Props - componentDidUpdate() Continued

- State can be updated using the setState() method.
- In the componentDidUpdate lifecycle method, you can call setState() to update the state.
- It is important to wrap the setState() call in a conditional statement to avoid potential infinite loops.
- It's worth noting that updating the state triggers a re-render of the component. Although the re-render may not be visible to the user, it can impact the performance of the component.

Unmounting

- **componentWillUnmount():** This method is called right before the component is unmounted from the DOM.
  - Uses: For cleaning up resources, such as cancelling network requests or removing event listeners.

Quick Note

In addition to these methods, there are other lifecycle methods like **getDerivedStateFromProps()** and **getSnapshotBeforeUpdate()** that provide additional control during the component lifecycle. However, it's worth noting that some of the lifecycle methods have been deprecated in recent versions of React in favor of newer alternatives.

It's important to mention that with the introduction of React Hooks, functional components can also manage their lifecycle using the **useEffect()** hook, which provides similar functionality to the lifecycle methods mentioned prior.