**MODULE: 4 (List and Hooks)**

**• Explain Life cycle in Class Component and functional component with Hooks**

* **Class Components**

- **Lifecycle Methods:** These are special methods within a class component that React calls at different stages of the component's lifecycle. They allow you to perform actions at specific moments, such as:

* **Mounting**

- constructor(): Initializes state and calls super().

- getDerivedStateFromProps(): Updates state based on props changes (rarely used).

- render(): Returns the component's JSX.

- componentDidMount(): Executed after the component is mounted and rendered in the DOM.

* **Updating:**

- getDerivedStateFromProps(): Can be used for state updates based on props changes.

- shouldComponentUpdate(): Determines if a re-render is necessary (rarely used).

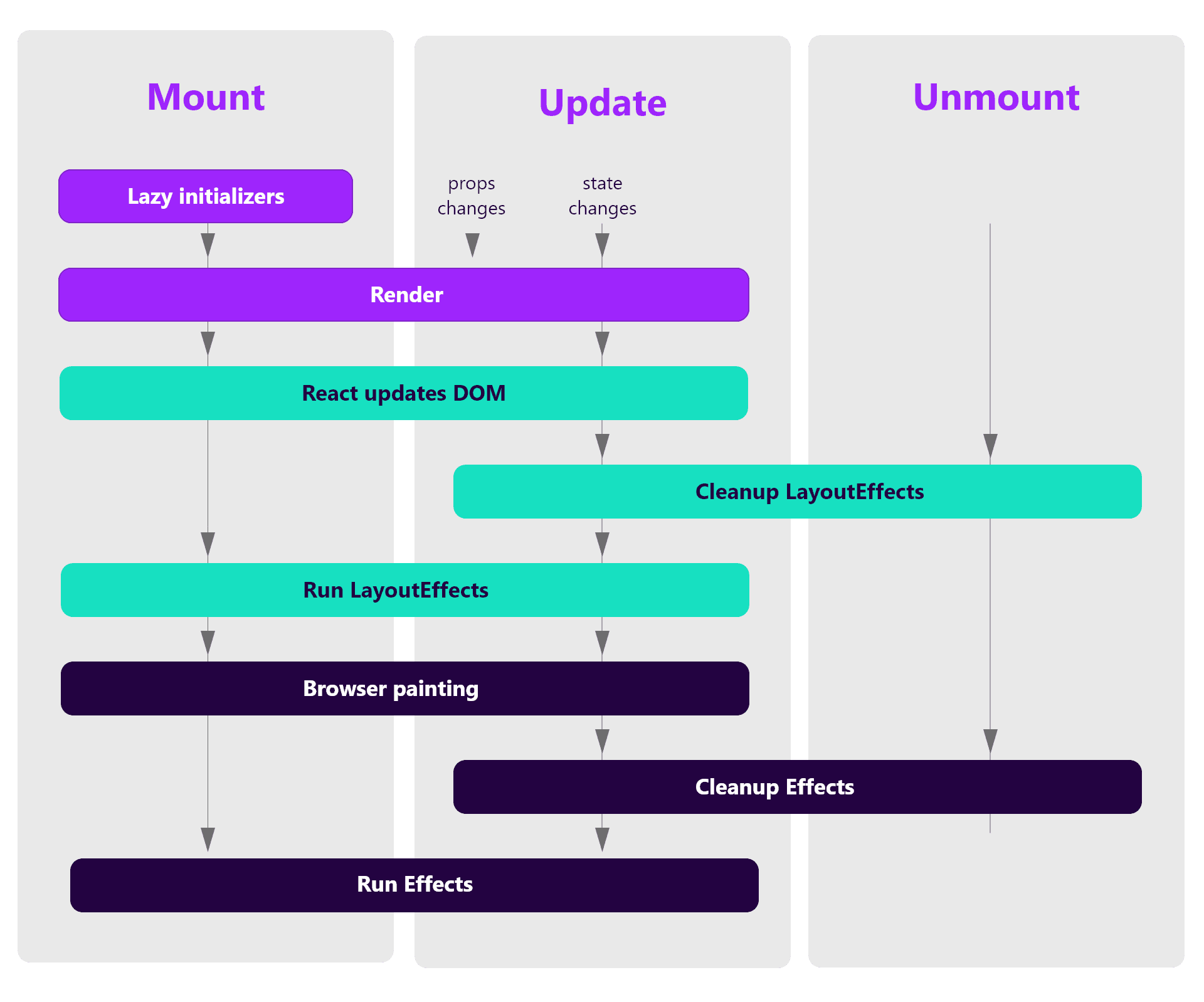
- render(): Called again for a re-render.

- getSnapshotBeforeUpdate(): Captures information before DOM updates (rarely used).

- componentDidUpdate(): Executed after the update is complete.

* **Unmounting:**

- componentWillUnmount(): Executed before the component is removed from the DOM.



* **Functional Components with Hooks**

- **Hooks:** These are functions introduced in React 16.8 that allow functional components to access state and lifecycle features without needing to convert them to class components. Hooks provide a more concise and less error-prone way to manage state and side effects.

- **useState():** Manages state within functional components.

- **useEffect():** Handles side effects, such as data fetching, subscriptions, or DOM manipulation. It can be used to simulate various lifecycle methods depending on its dependency array:

- **Mounting**: If the dependency array is empty, it runs only once after the first render, similar to `componentDidMount()`.

- **Updating:** If the dependency array includes values, it runs after every render where those values change, similar to `componentDidUpdate()`.

- **Cleanup**: It returns a cleanup function that runs before the component unmounts, similar to `componentWillUnmount()`.

* **Key Differences**

- Class components have explicit lifecycle methods, while functional components with hooks use hooks to manage lifecycle actions.

- Hooks often make code more concise and easier to read, especially for complex lifecycle logic.

- Hooks encourage better code organization and reusability.

* **Modern React Development**

- Hooks are generally favored in modern React development due to their benefits in terms of readability, maintainability, and reusability.

- Class components are still supported, but their use is becoming less common.

* **Choosing the Right Approach**

- If you're starting a new project, it's recommended to use functional components with hooks.

- For existing class components, you can consider migrating them to functional components with hooks if it makes sense for your project.

- In some cases, class components might still be appropriate, such as when dealing with third-party libraries that require them or for complex state management scenarios.