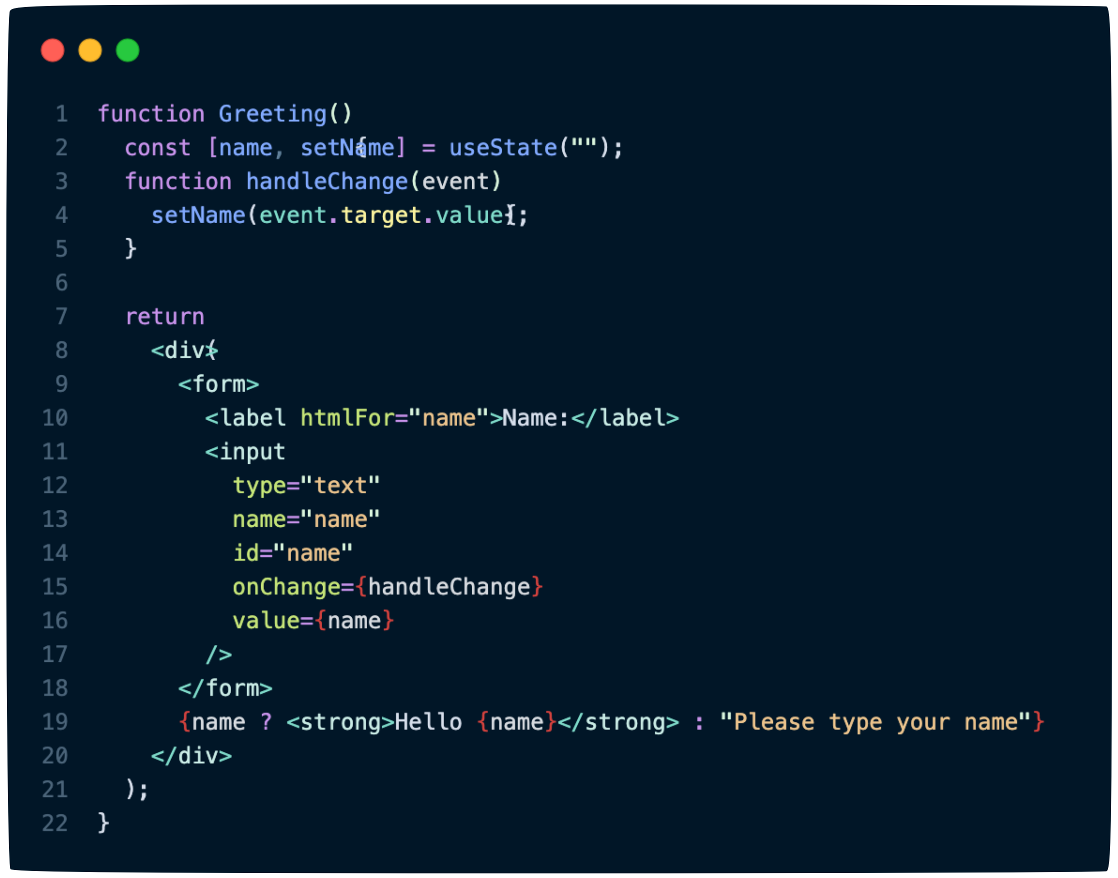
React Components and Introduction to React Hooks

Hristijan Veleski, Software Engineer @ GrabIT

React Components

- *Components* are one of the core concepts of React. They are the foundation upon which you build user interfaces (UI)
- React component is a JavaScript function that you can *sprinkle with markup*



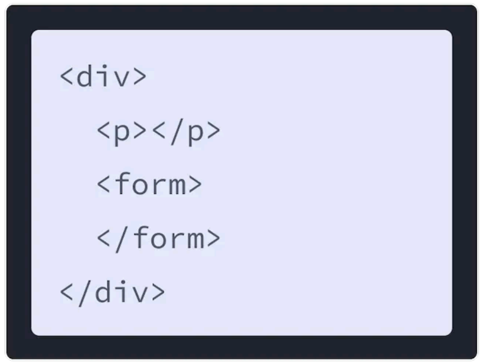
```
function Greeting()
  const [name, setName] = useState("");
  function handleChange(event)
    setName(event.target.value);
  }

  return
    <div>
      <form>
        <label htmlFor="name">Name:</label>
        <input
          type="text"
          name="name"
          id="name"
          onChange={handleChange}
          value={name}
        />
      </form>
      {name ? <strong>Hello {name}</strong> : "Please type your name"}
    </div>
  );
}
```

Reference:https://react.dev/learn/your-first-component

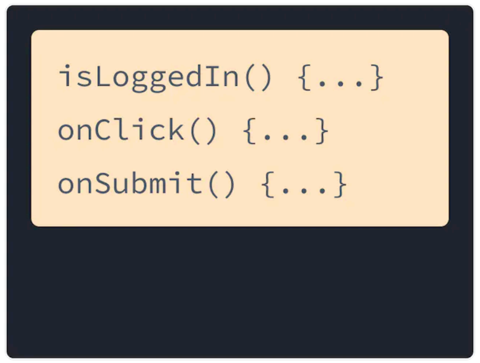
Writing JSX

- *JSX* is a syntax extension for JavaScript that lets you write HTML-like markup inside a JavaScript file
- Sometimes you will want to add a little JavaScript logic or reference a dynamic property inside that markup. In this situation, you can use curly braces in your JSX to open a window to JavaScript



```
<div>
  <p></p>
  <form>
  </form>
</div>
```

HTML

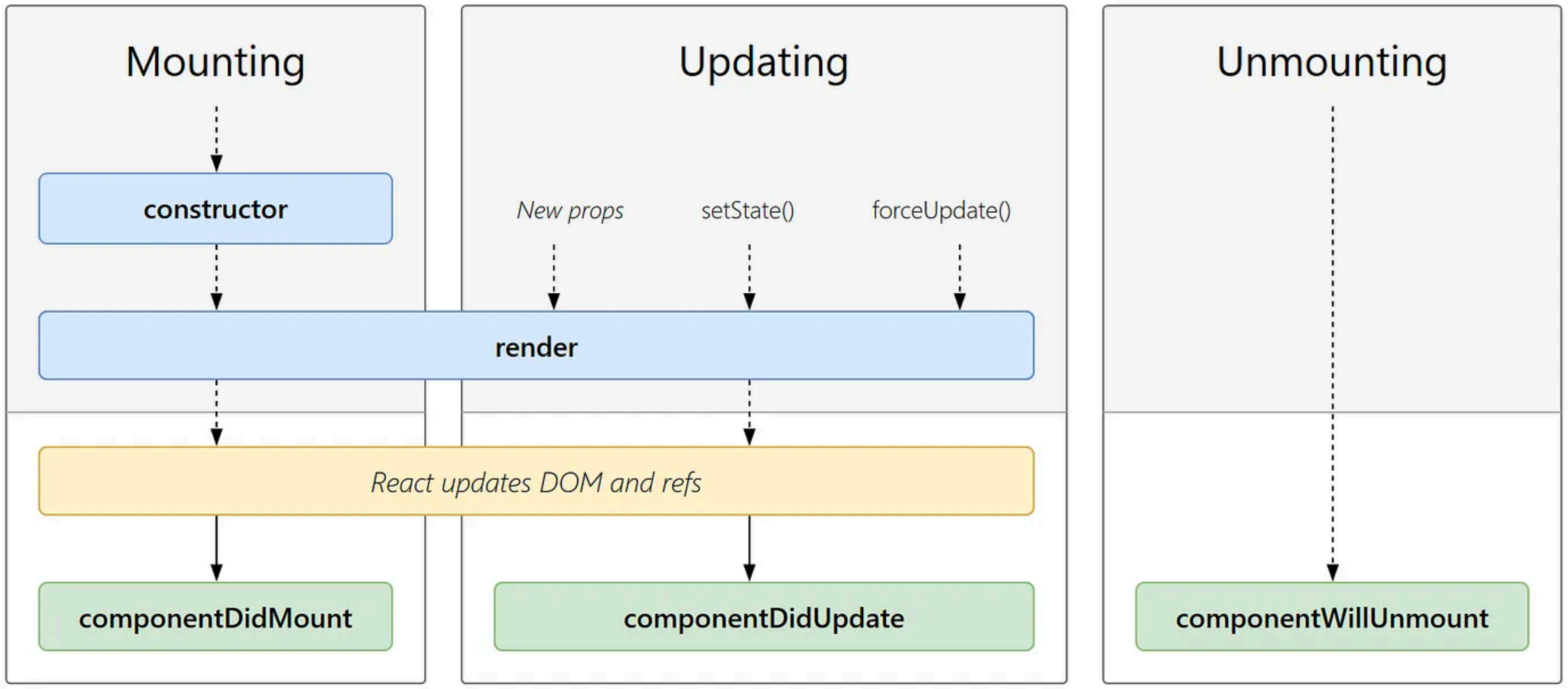


```
isLoggedIn() {...}
onClick() {...}
onSubmit() {...}
```

Javascript

Reference:https://react.dev/learn/your-first-component

Lifecycle diagram (Class based components)



Introduction to React Hooks

- useEffect
- useLayoutEffect
- useRef
- useState
- useReducer
- useMemo
- useCallback
- useContext

useState

- useState is a React hook that enables functional components to manage state. It allows you to declare state variables and update them, triggering component re-renders when the state changes.
- It takes an initial state value and returns an array containing the current state and a function to update it.
- You can pass a function to the initialState in order to initialize the state only once on the first render

```
const [state, setState] = useState(initialState);
```

Reference: https://react.dev/reference/react/useState

useRef

- The useRef hook allows you to persist a value between renders.
- It can be used to store a mutable value that does not cause a re-render when updated.
- It can be use to access a DOM element directly.

```
const ref = useRef(initialValue)
```

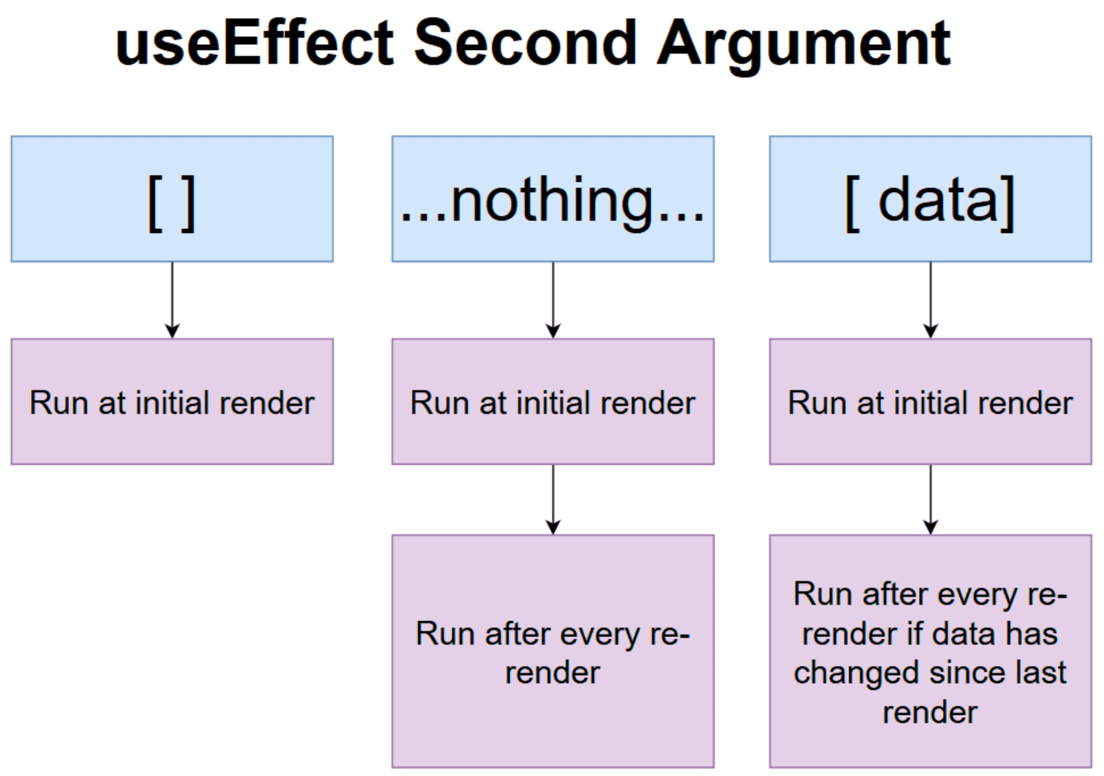
Reference:https://react.dev/reference/react/useRef

useEffect

- useEffect is a built-in hook that allows you to run some code after React renders (and re-renders) your component to the DOM.
- It is used for side effects in functional components. It addresses the need to perform tasks like data fetching, subscriptions, or manually interacting with the DOM after a component renders.

```
useEffect(setup, dependencies?)
```

Reference:https://react.dev/reference/react/useEffect


useEffect Second Argument
[ ]
...nothing...
[data]
Run at initial render
Run at initial render
Run at initial render
Run after every re-render
Run after every re-render if data has changed since last render

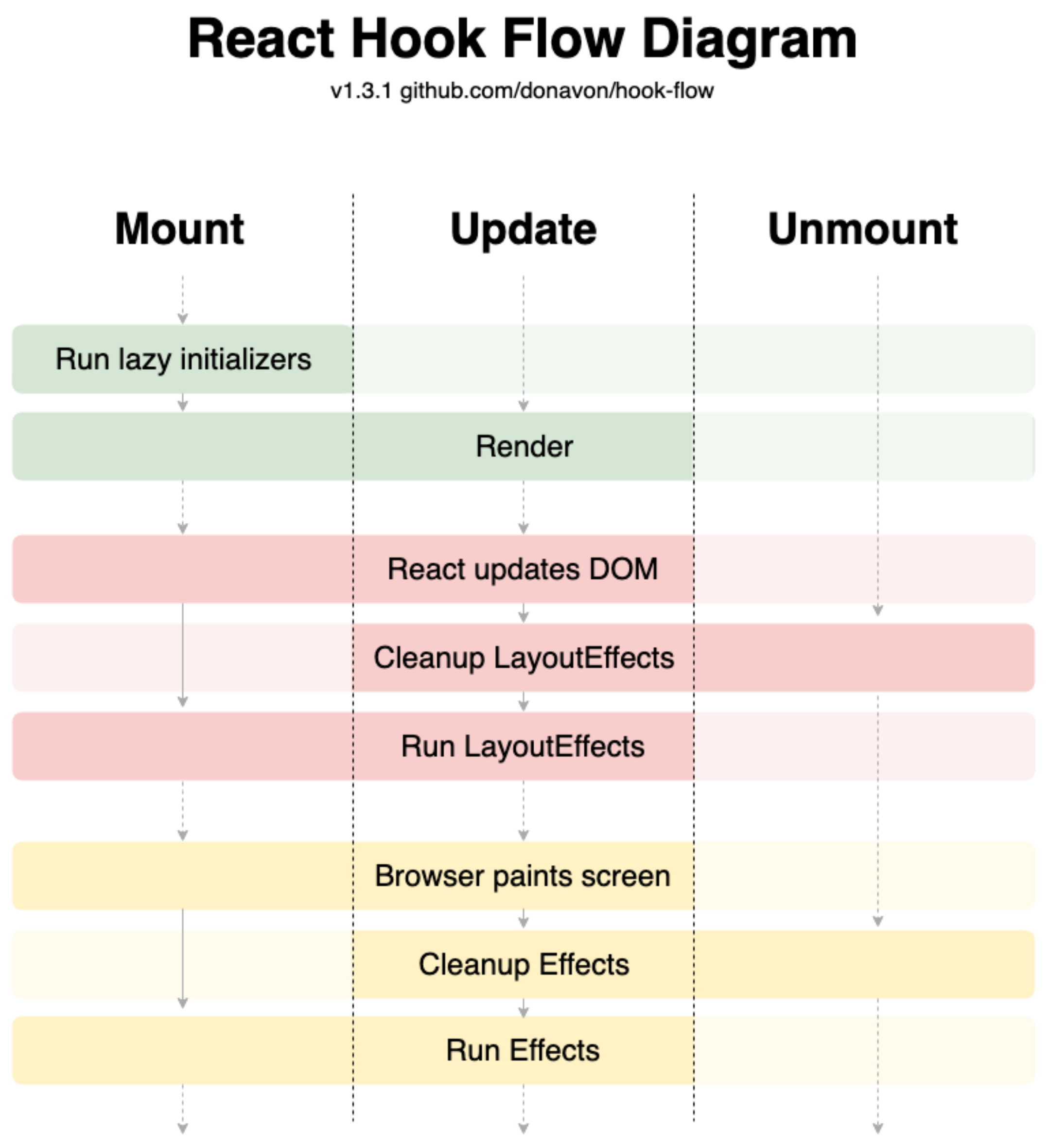
useLayoutEffect

- The `useLayoutEffect` Hook is a variation of the `useEffect` Hook that runs synchronously before the browser repaints the screen. It was designed to handle side effects that require immediate DOM layout updates.
- useLayoutEffect ensures that any changes made within the hook are applied synchronously before the browser repaints the screen. While this might not seem ideal, it is highly encouraged in specific use cases, such as when measuring DOM elements, or animating or transitioning elements

```
useLayoutEffect(setup, dependencies?)
```

Reference:https://react.dev/reference/react/useLayoutEffect

Lifecycle diagram (Function based components)



Notes:

1. Updates are caused by a parent re-render, state change, or context change.
2. Lazy initializers are functions passed to useState and useReducer.

useReducer

- The useReducer Hook is used to store and update states, just like the useState Hook. It accepts a reducer function as its first parameter and the initial state as the second.
- useReducer returns an array that holds the current state value and a dispatch function to which you can pass an action and later invoke it.

```
const [state, dispatch] = useReducer(reducer, initialArg, init?)
```

Reference:https://react.dev/reference/react/useReducer

useContext

- useContext is a React hook that allows components to access values from the React context without prop drilling. It simplifies state management by providing a way to share values like themes, authentication status, or user preferences across components, avoiding the need to pass them explicitly through every level of the component tree.

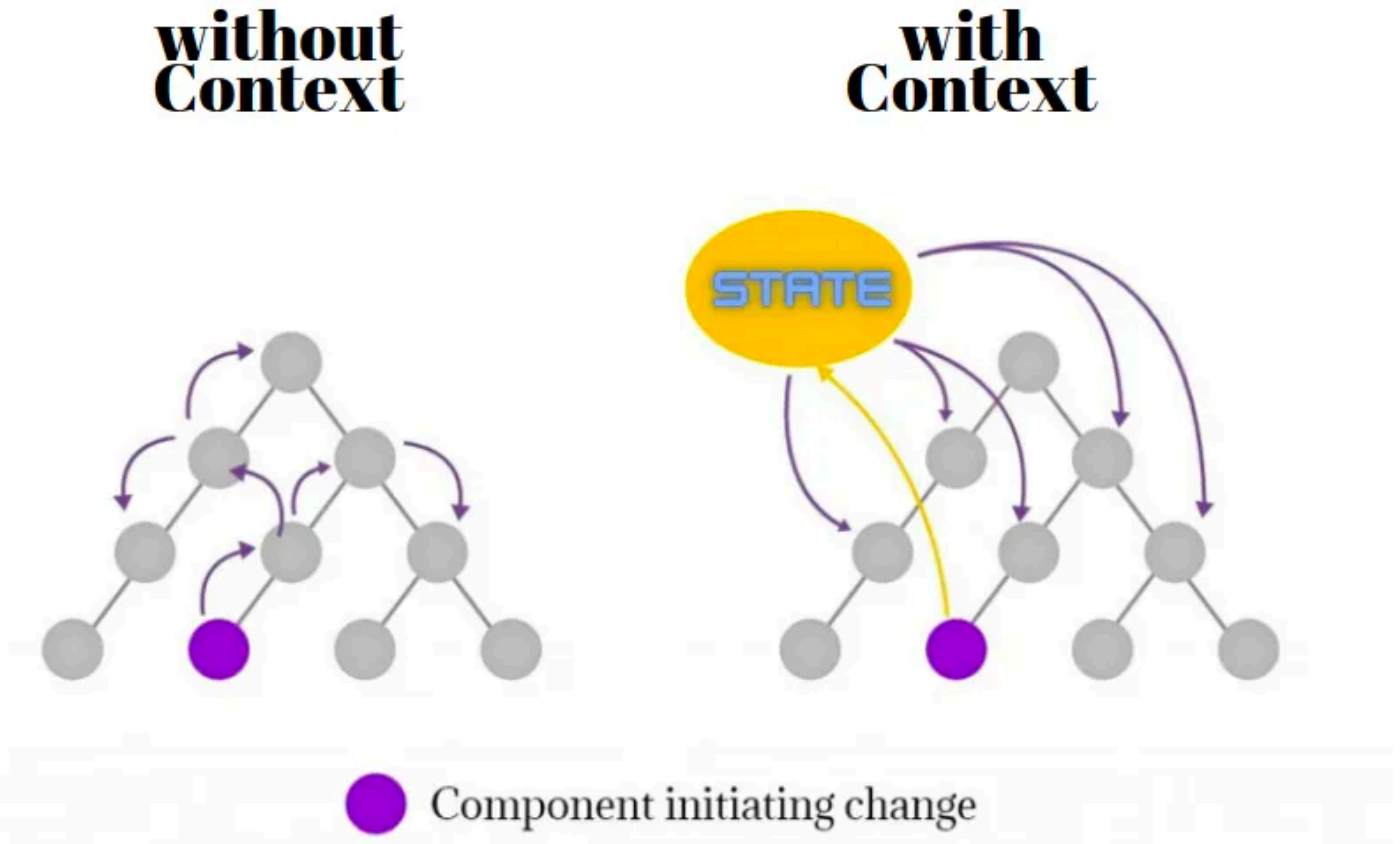
- This helps improve code readability and maintainability by reducing the complexity of passing data down through multiple layers of components.

```
const SomeContext = createContext(defaultValue)
```

```
const value = useContext(SomeContext)
```

Reference:https://react.dev/reference/react/createContext
Reference:https://react.dev/reference/react/useContext

useContext Illustration



useCallback

- useCallback is a React hook that memoizes a callback function, preventing it from being recreated on every render. It's particularly useful in optimizing performance by ensuring that functions passed down to child components do not trigger unnecessary re-renders.

- This is valuable when dealing with callbacks in scenarios like event handlers or passing functions as props, helping to avoid unnecessary component updates and optimizing the overall application performance.

- useCallback is a tool for enhancing the efficiency of React components by memoizing callback functions and managing their dependencies.

```
const cachedFn = useCallback(fn, dependencies)
```

Reference:https://react.dev/reference/react/useCallback

useMemo

- useMemo is a React hook that memoizes the result of a computation, preventing unnecessary recalculations. It's useful for optimizing performance by caching the result of expensive operations and only recomputing when the dependencies change.
- This helps avoid redundant calculations in scenarios like rendering, especially when dealing with complex data transformations or filtering.
- useMemo is a tool for enhancing the efficiency of React components by memoizing values and managing their dependencies.

```
const cachedValue = useMemo(calculateValue, dependencies)
```

Reference:https://react.dev/reference/react/useMemo

Appendices

[1]. Official documentation for React
https://react.dev/learn

[2]. Epic React course
https://epicreact.dev/

REACH OUT

Feel free to drop by
or if you are shy, contact our HR department at
careers@grabit.io

The droid you are looking for is

Ms. Elena Mircheska
People and Talent Acquisition Specialist
E: elena.mircheska@grabit.io
M: +38975932179

www.grabit.io