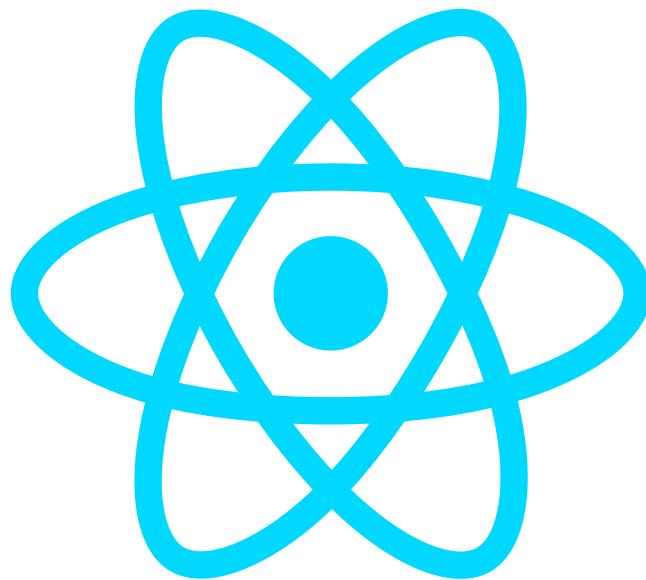


Day 17



Siva Prasad Thorlikonda
Follow me

useReducer() Master **State** Like a Pro 🚀

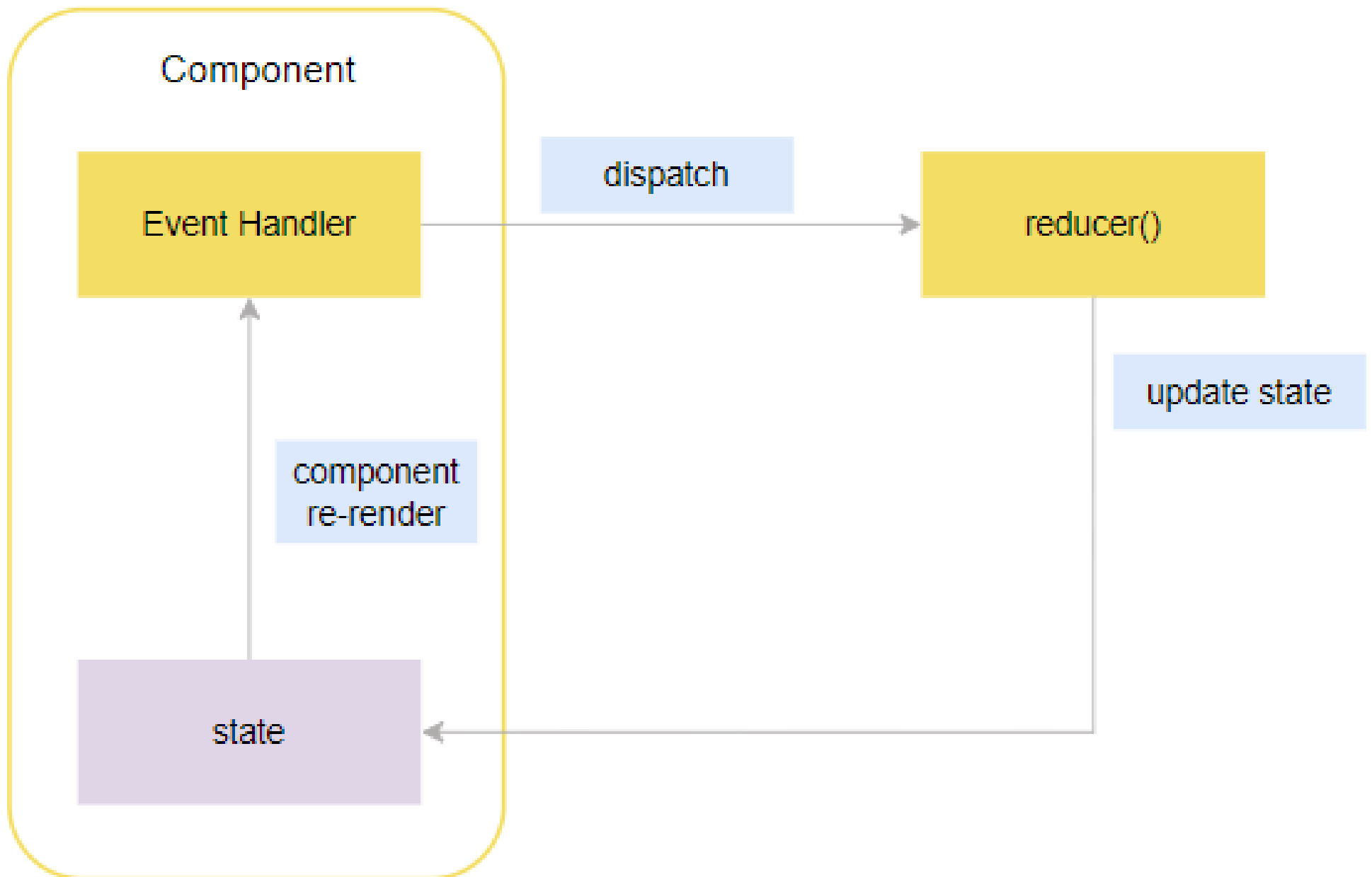


Save





useReducer() Hook Workflow



Save



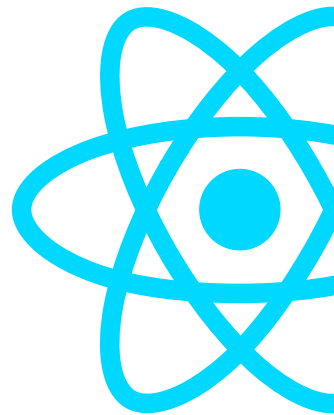


What is `useReducer()`?

🧠 `useReducer()` is a React Hook used for managing complex state logic.

- It's like an **advanced version** of `useState` — ideal when:

- ✓ Multiple states are related
- ✓ You want clear state transitions
- ✓ You prefer an action-based approach



Save

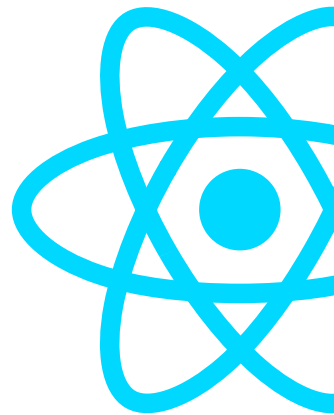




When to Use `useReducer()`?

Use it when:

- ✓ State logic depends on previous state
 - ✓ Multiple states need to be updated together
 - ✓ You want a Redux-like structure in React (without Redux)
- 👤 Perfect for forms, counters, toggles, etc.



Save



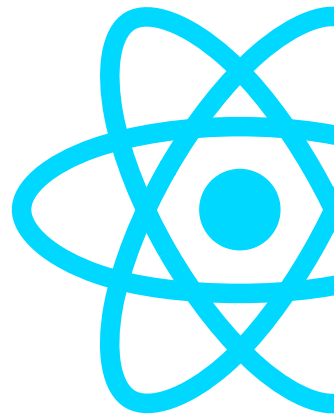


Core Concepts of `useReducer()`

`useReducer` takes two arguments:

```
const [state, dispatch] = useReducer(reducer,  
                                     initialState);
```

- ◆ `initialState` → The starting state
- ◆ `reducer` → A function that handles state changes
- ◆ `dispatch` → Triggers the state update using an action
- ◆ `action` → An object that tells what to do





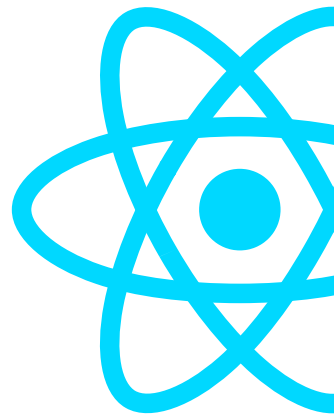
Define **initialState**

This is your default or starting data.

```
const initialState = { count: 0 };
```



You can store any structure — object, array, number, etc.



Save






Define **reducer** function

The heart of useReducer - it controls how state changes!

```
function reducer(state, action) {  
  switch (action.type) {  
    case 'increment':  
      return { count: state.count + 1 };  
    case 'decrement':  
      return { count: state.count - 1 };  
    default:  
      return state;  
  }  
}
```



 It receives current state and action, then returns the new state.



Save





Dispatch actions from component

Now we connect everything in a component using dispatch.

```
import { useReducer } from 'react';  
function Counter() {  
  const initialState = { count: 0 };  
  function reducer(state, action) {  
    switch (action.type) {  
      case 'increment':  
        return { count: state.count + 1 };  
      case 'decrement':  
        return { count: state.count - 1 };  
      default:  
        return state;  
    }  
  }  
}
```



Continues.....





```
const [state, dispatch] = useReducer(reducer,
initialState);
return (
<div>
<p>Count: {state.count}</p>
<button onClick={() => dispatch({ type: 'decrement'
})}>-</button>
<button onClick={() => dispatch({ type: 'increment'
})}>+</button>
</div>
);
}
```

📌 `dispatch({ type: 'increment' })` tells reducer what to do



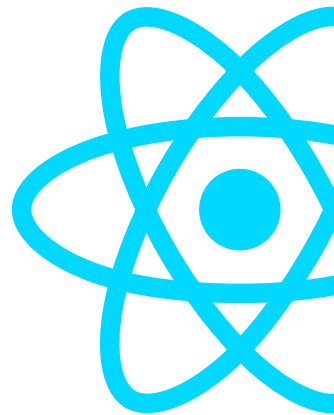
Save





useReducer vs useState

Feature	useState	useReducer
Simplicity	✅ Easy for basic cases	❌ More setup needed
Complex logic	❌ Can get messy	✅ Clean with reducer function
Redux-like behavior	❌ No	✅ Yes



Save

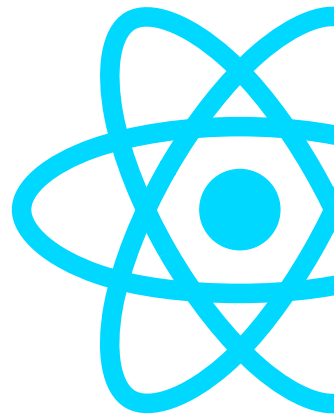




Summary of useReducer

- ✓ Best for structured state management
- ✓ Clean and readable code using action type
- ✓ Works well for forms, toggle, multiple states
- ✓ Makes your logic more predictable and scalable

Thank You
FOR
SUPPORTING ME



Save

