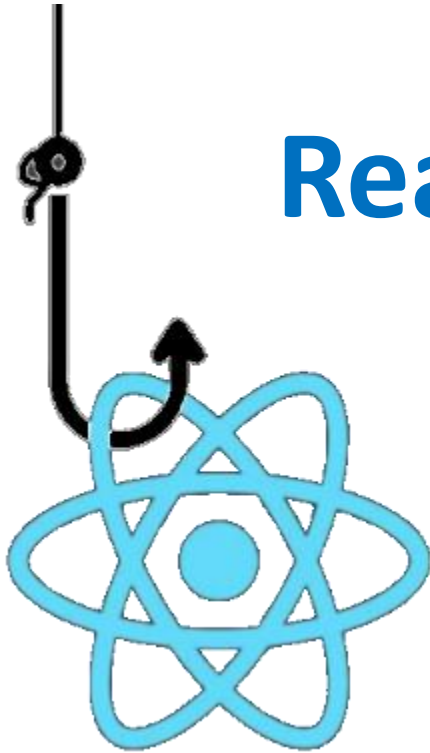


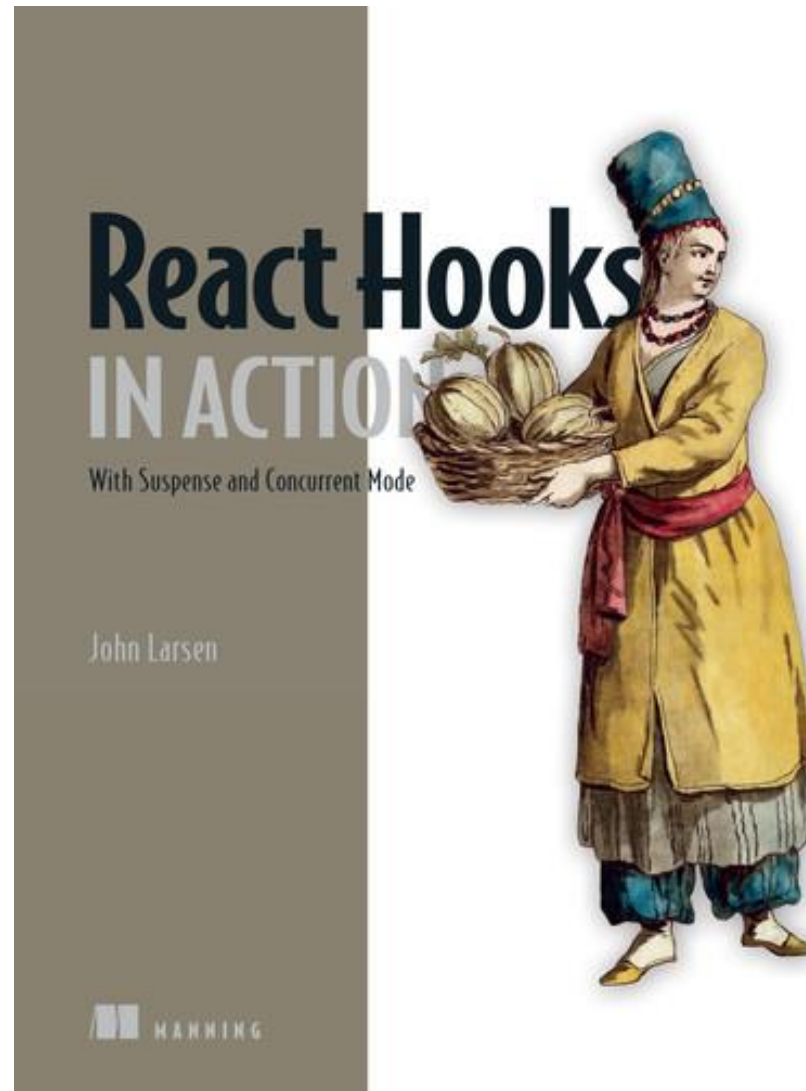
React Hooks



Outline

1. Introduction
2. useState
3. useEffect
4. useRef
5. useContext
6. useRouter
7. useActionState

Slides are based on



<https://learning.oreilly.com/library/view/react-hooks-in/9781617297632/>

What is Hook?

- A Hook is a special function that lets you **hook** into React features such as state and lifecycle methods
- There are 3 rules for hooks:
 - Hooks can only be called inside React function components.
 - Hooks can only be called at the top level of a component.
 - Hooks cannot be conditional

Common Hooks



useState: creates a state variable

- Used for basic state management inside a component

const [state, setState] = useState(initialState)



The name of
your state



The function you'll
eventually use to
change the value of this
state



The initial value
of your state

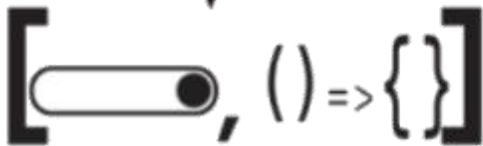
useEffect

- For doing stuff when a component is mounts/unmounts/updates
- Ideal for fetching data when the component is mounted

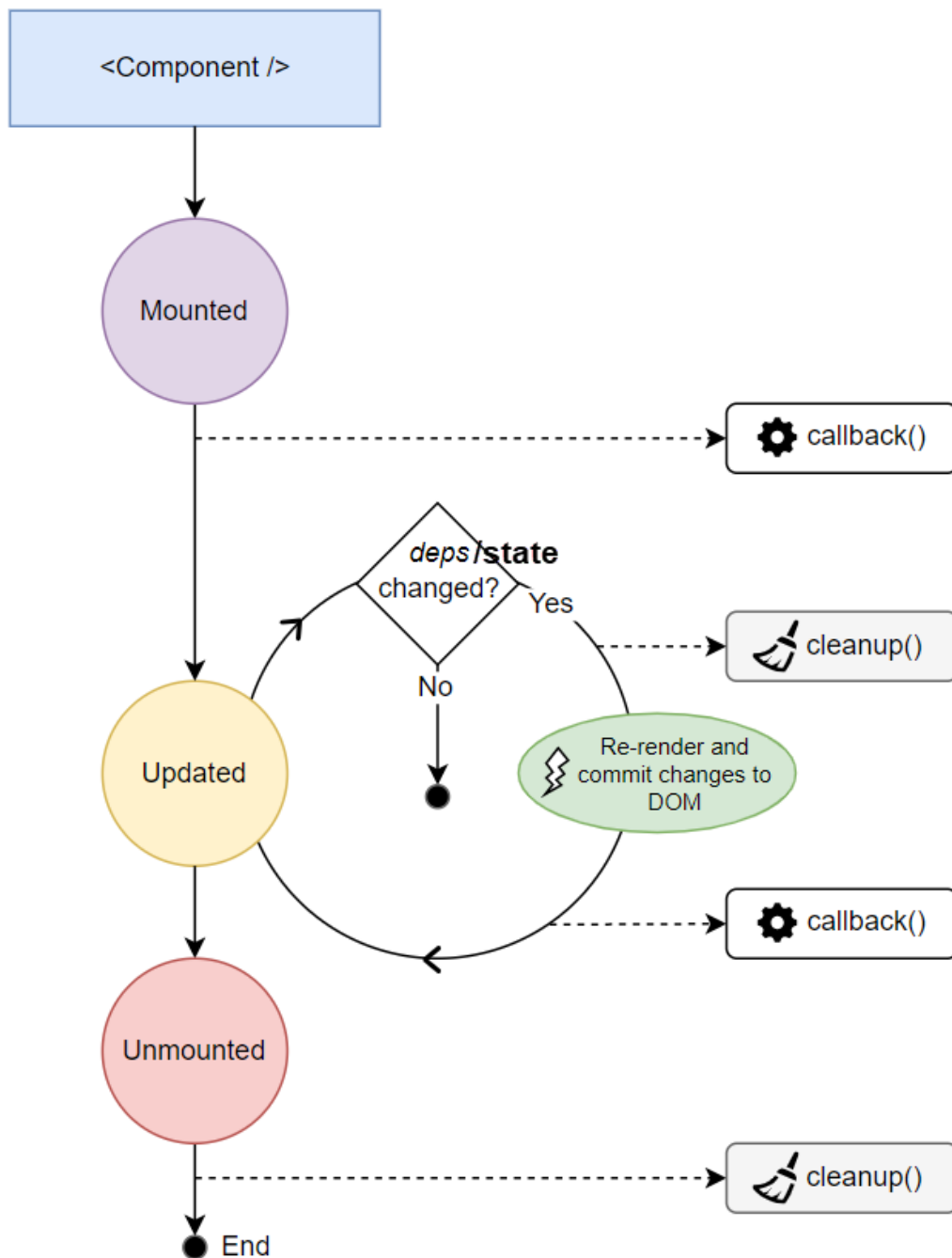
```
useEffect( () => {  
  // do something with dep1 and dep2  
  return () => { /* clean up */ };  
}, [dep1, dep2] );
```



Cleanup function:
Return a function to clean up
after the effect (e.g., unsubscribe,
stop timers, remove listeners, etc.).



Dependency list:
Run the effect only if the
values in the array change.



A) After initial rendering, `useEffect()` invokes the callback having the side-effect. cleanup function is not invoked

B) On later renderings, before invoking the next side-effect callback, `useEffect()` invokes the cleanup function from the previous side-effect execution (to clean up everything after the previous side-effect), then runs the current side-effect

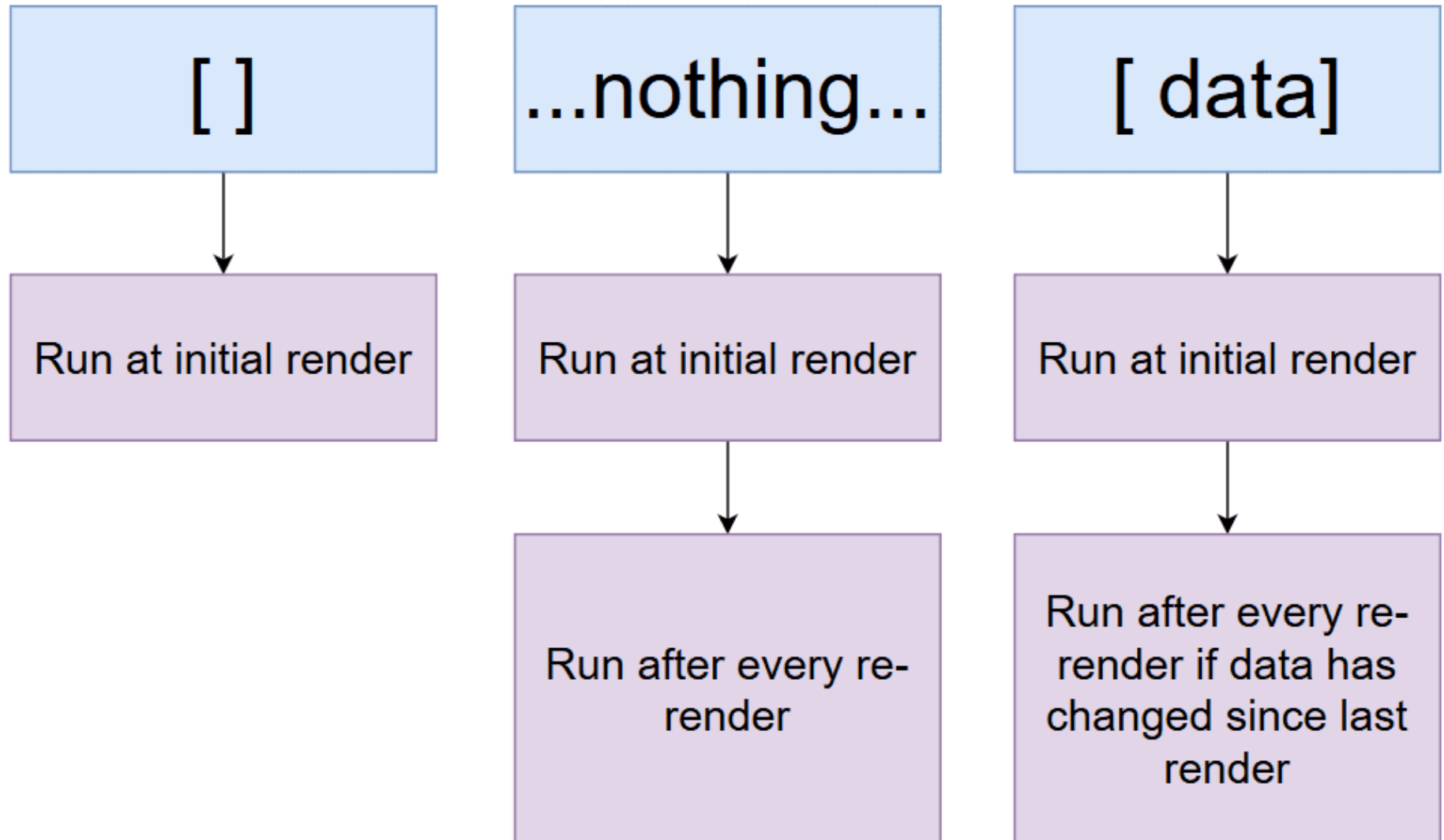
C) Finally, after unmounting the component, `useEffect()` invokes the cleanup function from the latest side-effect

Common side effects

Common side effects include:

- Setting the page title imperatively
- Working with timers like `setInterval` or `setTimeout`
- Logging messages to the console or other service
- Fetching data or subscribing and unsubscribing to services
- Setting or getting values in local storage

useEffect - 2nd argument



Use cases for the useEffect hook

| Call pattern | Code pattern | Execution pattern |
|-------------------------------------|---|--|
| No second argument | <pre>useEffect(() => { // perform effect });</pre> | Run after every render. |
| Empty array as second argument | <pre>useEffect(() => { // perform effect }, []);</pre> | Run once, when the component mounts. |
| Dependency array as second argument | <pre>useEffect(() => { // perform effect // that uses dep1 and dep2 }, [dep1, dep2]);</pre> | Run whenever a value in the dependency array changes. |
| Return a function | <pre>useEffect(() => { // perform effect return () => {/* clean-up */}; }, [dep1, dep2]);</pre> | React will run the cleanup function when the component unmounts and before rerunning the effect. |

useEffect – Executes code during Component Life Cycle

- **Initialize state data** when the component loads

```
useEffect(() => {  
  async function fetchData() {  
    const url = "https://api.github.com/users";  
    const response = await fetch(url);  
    setUsers( await response.json() ); } // set users in state  
    fetchData();  
  }, []); // pass empty array to run this effect once when the component is first mounted to the DOM.
```

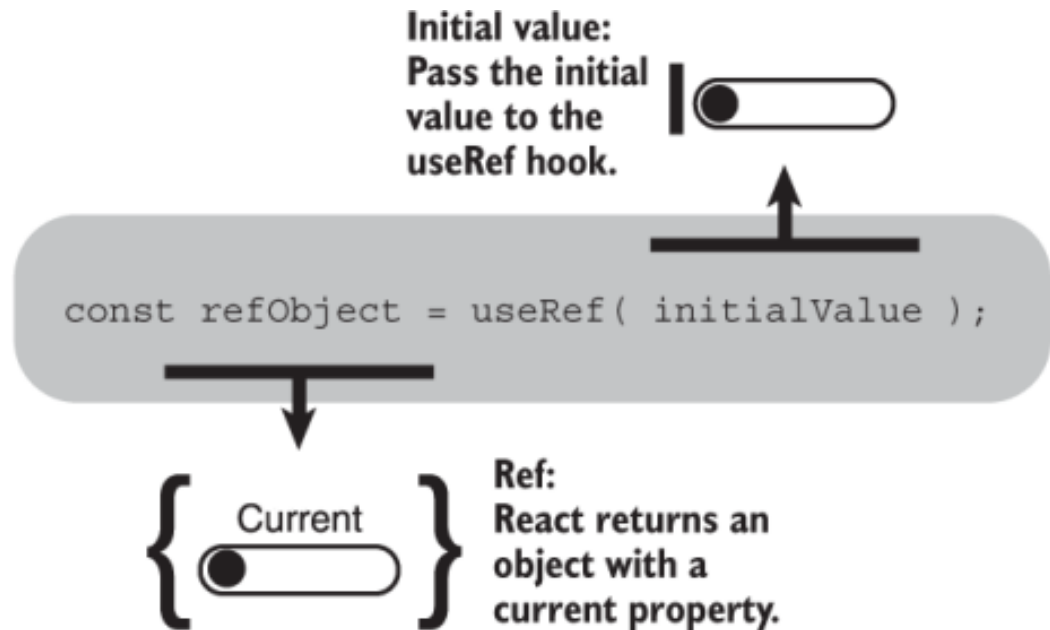
- **Executing a function every time a state variable changes**

```
useEffect(() => {  
  async function fetchData() {  
    const url = `https://hn.algolia.com/api/v1/search?query=${query}`;  
    const response = await fetch(url);  
    const data = await response.json();  
    setNews(data.hits);  
  }  
  fetchData();  
}, [query]);
```

If 2nd parameter is not set, then the useEffect function will run on every re-render

useRef

- useRef() hook to create **persisted mutable values** as well as directly **access DOM elements** (e.g., focusing an input)
 - The value of the reference is persisted (stays the same) between component re-renderings;
 - Updating a reference doesn't trigger a component re-rendering.



useRef for Mutable values

- `useRef(initialValue)` accepts one argument as the initial value and returns a reference. A reference is an object having a special property `current`

```
import { useRef } from 'react';

function LogButtonClicks() {
  const countRef = useRef(0);

  const handle = () => {
    countRef.current++;
    console.log(`Clicked ${countRef.current} times`);
  };

  console.log('I rendered!');

  return <button onClick={handle}>Click me</button>;
}
```

- `reference.current` accesses the reference value, and `reference.current = newValue` updates the reference value
- The value of the reference is persisted (stays the same) between component re-renderings
- Updating a reference doesn't trigger a component re-rendering

useRef for accessing DOM elements

- useRef() hook can be used to access DOM elements

```
import { useRef, useEffect } from 'react';

function InputFocus() {
  const inputRef = useRef();

  useEffect(() => {
    inputRef.current.focus();
  }, []);

  return (
    <input
      ref={inputRef}
      type="text"
    />
  );
}
```

- Define the reference to access the element

```
const inputRef = useRef();
```

- Assign the reference to **ref** attribute of the element:

```
<input ref={inputRef} />
```

- After mounting, `inputRef.current` points to the DOM element

=> In this example, we access the input to focus on it when the component mounts. After mounting we call

```
inputRef.current.focus()
```

useRef vs. useState

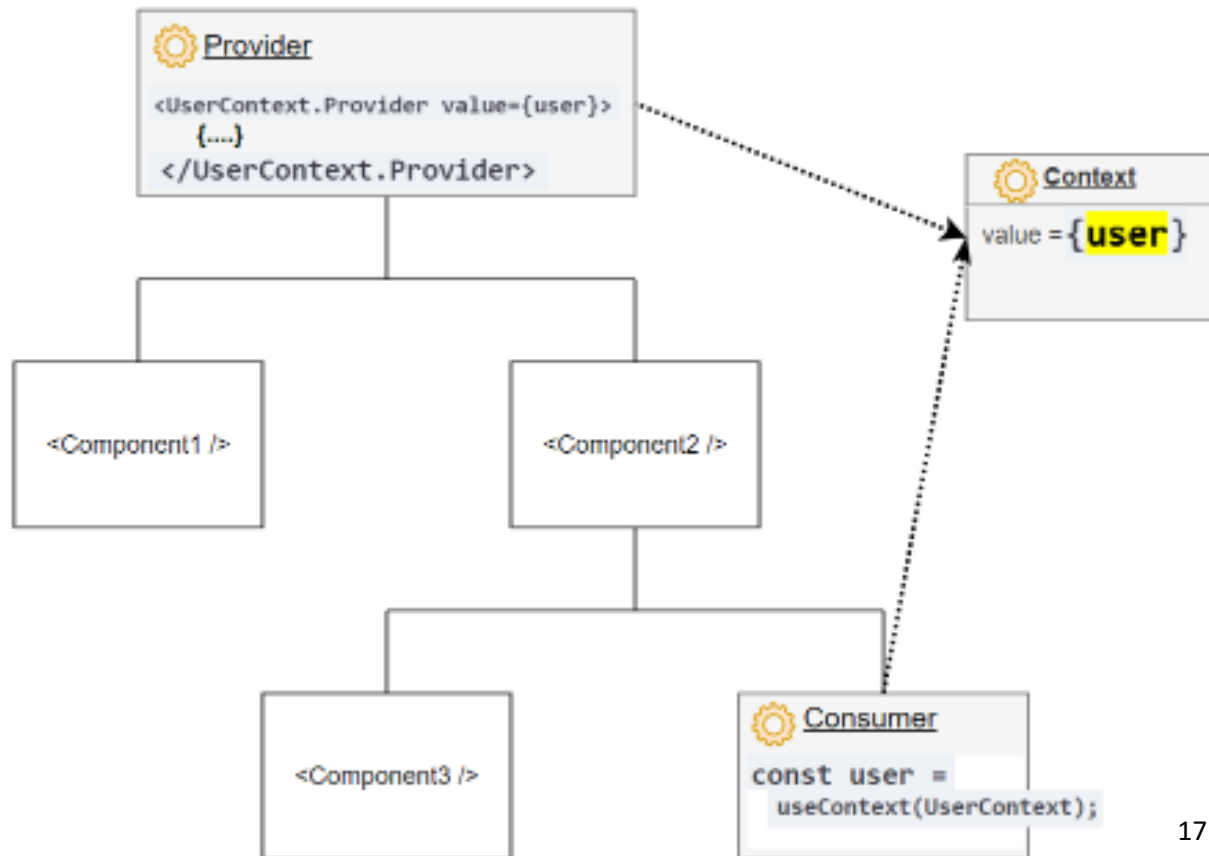
- useState, useReducer, and useContext hooks triggering re-renders when a state variable changes
- useRef remembers the state value but change of value does not trigger rerender
 - The values of refs persist (specifically the **current** property) throughout render cycles

useContext

- Share state (e.g., current user, user settings) between deeply nested components more easily than prop drilling (i.e., without pass the state as props through each nested component)

- Using the context requires 3 steps: creating, providing, and consuming the context

- If the context variables change then all consumers are notified and re-rendered



useContext – provides shared variables and functions

1. **Create a context** instance (i.e., a container to hold shared variables and functions)

```
import React from 'react';  
const UserContext = React.createContext();  
export default UserContext;
```

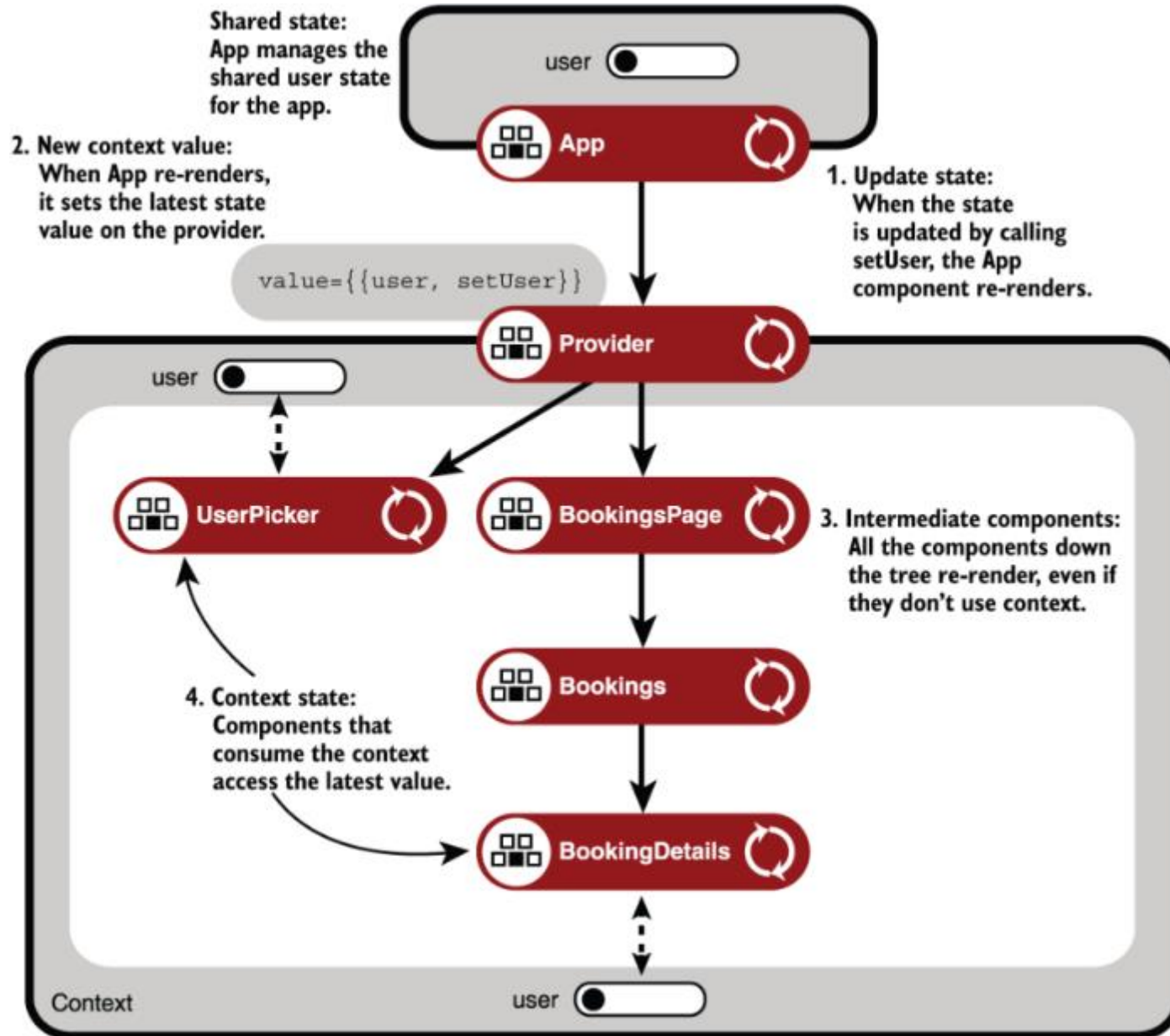
2. **Provider places shared variables / functions in the context** to make them available to child components

```
import UserContext from './components/UserContext';  
function App() { return (  
  <UserContext.Provider value={ user }>  
    <Welcome /> ...  
  </UserContext.Provider>  
); }
```

3. **Consumer access the shared variables / functions in the context**

```
import React, {useContext} from "react"; import UserContext from './UserContext';  
export default function Welcome() {  
  const user = useContext(UserContext);  
  return <div>You are login as: {user.username}</div>;  
}
```

Shared State Example



useRouter

- **useRouter** hook allows you to programmatically change routes inside Client Components
 - Use the <Link> component for navigation unless you have a specific requirement for using useRouter

```
'use client'

import { useRouter } from 'next/navigation'

export default function Page() {
  const router = useRouter()

  return (
    <button type="button" onClick={() => router.push('/dashboard')}>
      Dashboard
    </button>
  )
}
```

useActionState

- useActionState is a Hook that allows you to update state based on the result of a form action
- <https://react.dev/reference/react/useActionState>

Summary

- Hooks are functions which "hook into" React state and lifecycle features from components
- **useState** : manage state
- **useEffect**: perform side effects and hook into moments in the component's life cycle
- **useRef**: access DOM elements directly
- **useContext**: share data and functions with child components without prop drilling using

Resources

- **Hooks at a Glance**

<https://reactjs.org/docs/hooks-overview.html>

- React Hooks in Action textbook

<https://learning.oreilly.com/library/view/react-hooks-in/9781617297632/>

- Useful list of resources

<https://github.com/rehooks/awesome-react-hooks>