UseEffect()

miguel.garrido@ironhack.com

Components

Components re-render whenever:

1. Their state changes (via setState or hooks like useState)

2. Their props change (parent passes new values)

3. Their parent component re-renders (even if props haven't changed)"

Components

Some functions are outside the React rendering flow:

- Fetching

- Timer (SetInterval, setTimeout)

- Subscriptions (WebSockets)

- Event Listeners.

Components

These operations need special handling in React because they happen outside the component's render phase

Lifecycle of a Component

1. Mounting

When the component appears on the screen for the first time.

What happens:

- React creates the component
- The component function runs for the first time
- State is initialized (via useState)
- JSX is returned and rendered to the DOM

Lifecycle of a Component

2. Updating (Part 1)

When the component is already on screen and something changes.

What triggers an update:

- State changes
- Props change (parent passes new values)
- Parent component re-renders

Lifecycle of a Component

2. Updating (Part 2)

What happens:

- The component function runs again
- New JSX is returned
- React compares the new JSX with the previous one
- Only the changed parts of the DOM get updated

Lifecycle of a Component

2. Updating (Part 3)

```
function Counter() {
  const [count, setCount] = useState(0);

  return (
    <button onClick={() => setCount(count + 1)}>
      {count}
    </button>
  );
}
// The entire Counter component is updated on setCount
```

Lifecycle of a Component

2. Updating (Part 4)

React keeps track of the state in memory, outside the component.

When a component re-renders, React gives it back the current state value.

Lifecycle of a Component

3. Unmounting

When the component is removed from the screen.

What happens:

- Component gets destroyed
- Removed from the DOM
- All its state is lost

Lifecycle of a Component

3. Unmounting

```
function App() {
  const [show, setShow] = useState(true);

  return (
    <>
      {show && <Welcome />} {/* Unmounts when show becomes false */}
      <button onClick={() => setShow(!show)}>Toggle</button>
    </>
  );
}
```

UseEffects()

Short for side effects. They help us to setup on three stages of a component:

1. Mounting
2. On update phase
3. On unmounting phase

UseEffects()

Basic syntax:

```
useEffect(fn, dependencies);
```

`fn`

Side effect logic

`dependencies`

An **array of values** that trigger the effect to re-run when they change.

UseEffects() - [ ]

Basic syntax:

```
useEffect(fn, dependencies);
```

When:

```
dependencies = []
```

The fn logic will only be executed on **mounting**

UseEffects() - no dependencies

Basic syntax:

```
useEffect( fn );
```

When **dependencies** is not specified **fn** logic will render on every update ⚠️

UseEffects() - [val1, val2]

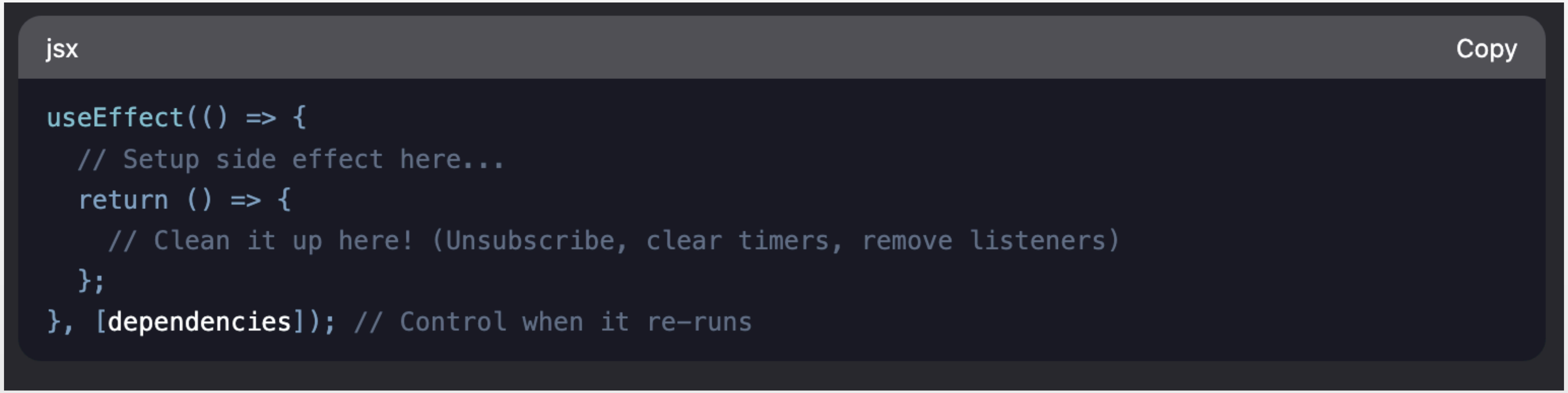
Basic syntax:

```
useEffect( fn, [val1, val2]);
```

fn run when **val1** or **val2** changes

val1, **val2** can be state variables, props or any other variables

UseEffects()



jsx Copy

```jsx
useEffect(() => {
  // Setup side effect here...
  return () => {
    // Clean it up here! (Unsubscribe, clear timers, remove listeners)
  };
}, [dependencies]); // Control when it re-runs
```

UseEffects() on Mounting

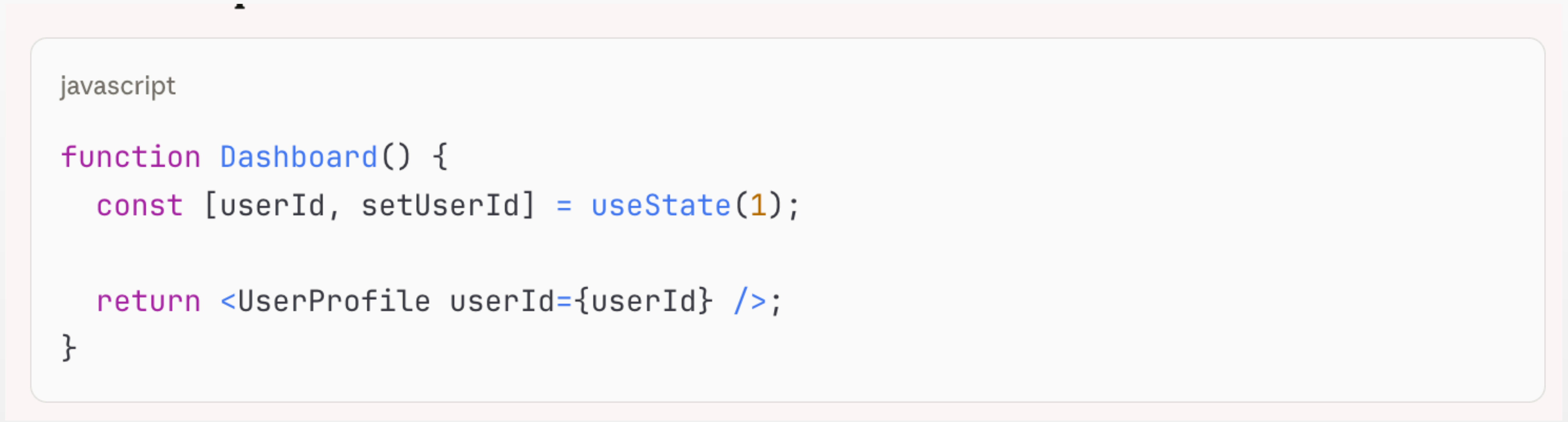
Useful for:

1. Fetching data

2. Analytic tools (Eg: Google) and other third-party libraries

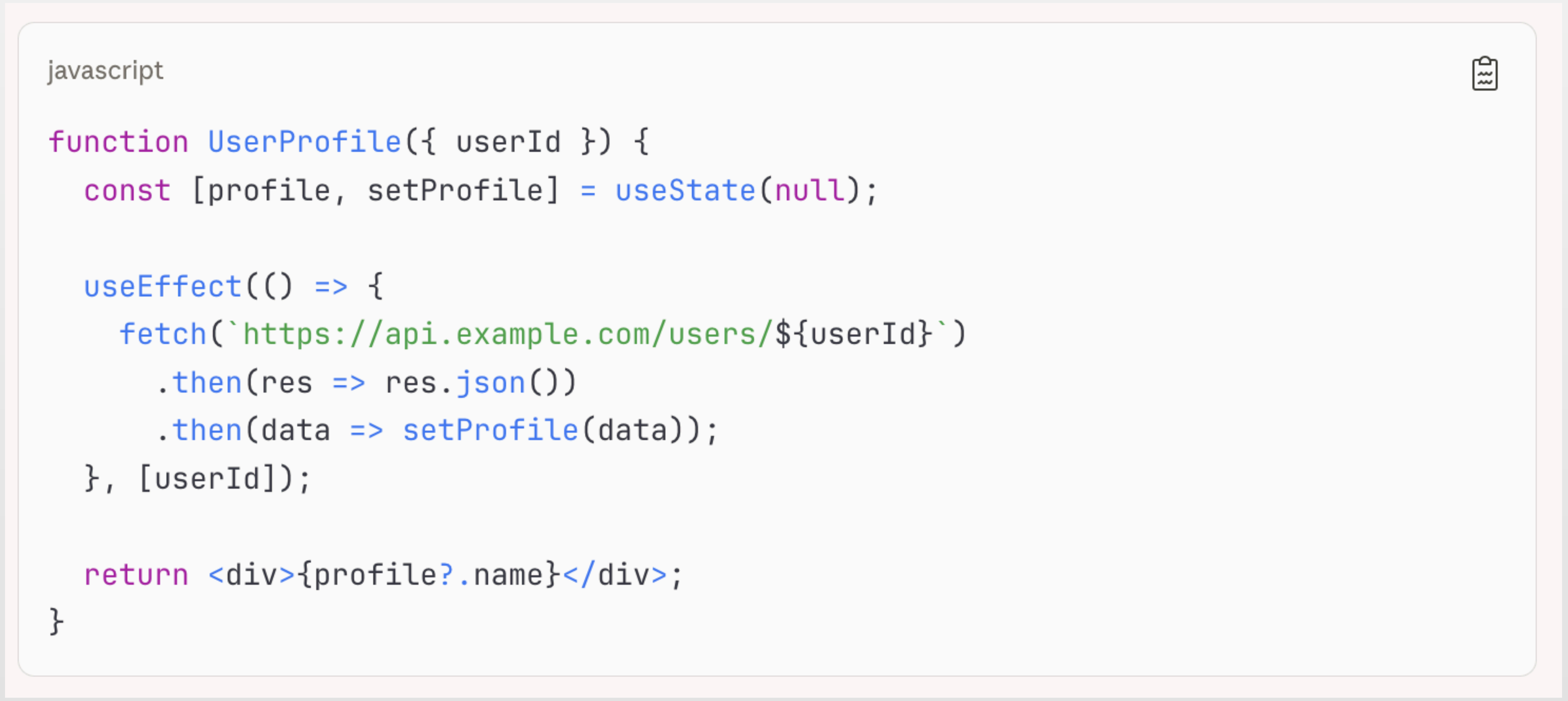
Avoids unnecessary re-runs, or re-fetch

UseEffects() - Dashboard & User



javascript

```javascript
function Dashboard() {
  const [userId, setUserId] = useState(1);

  return <UserProfile userId={userId} />;
}
```



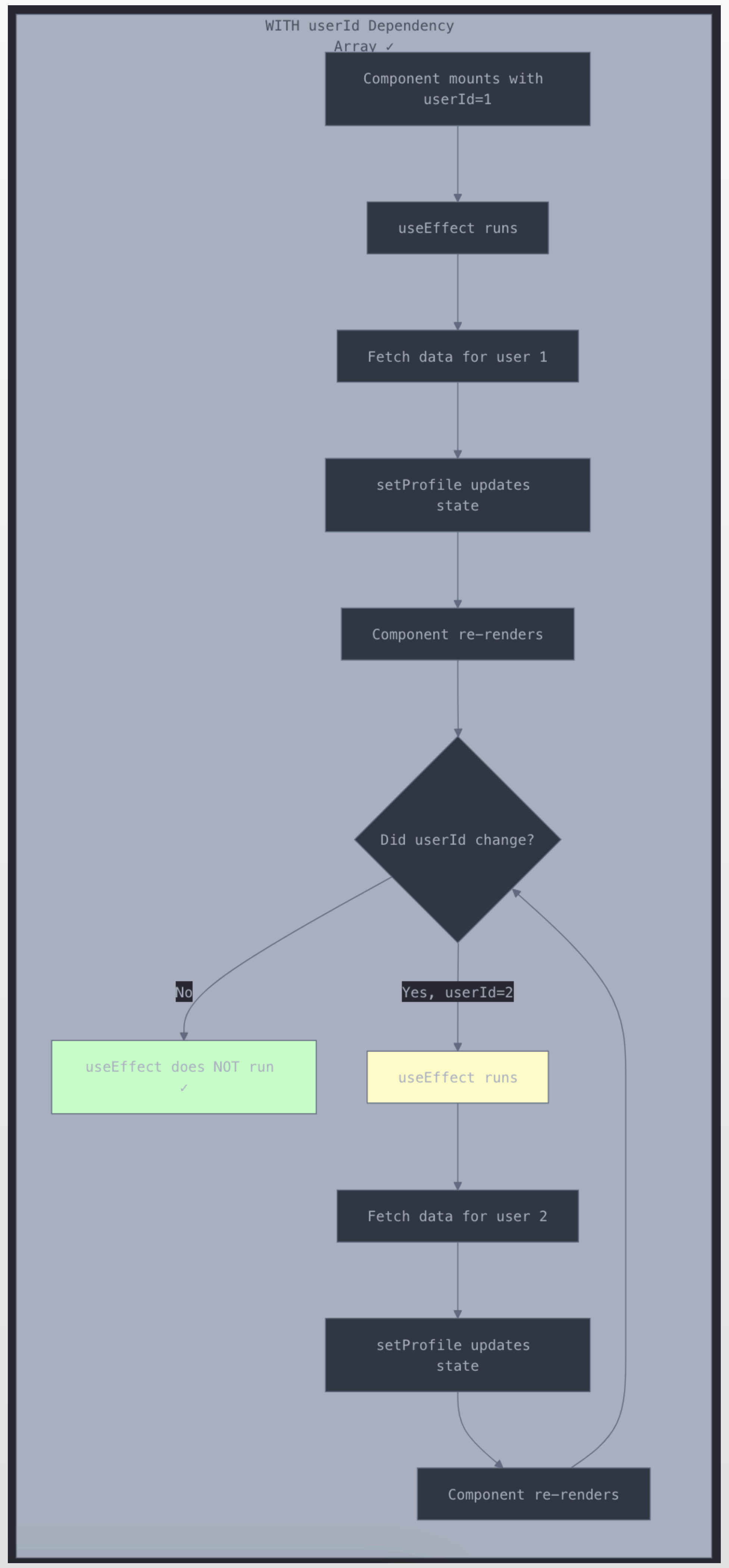
javascript

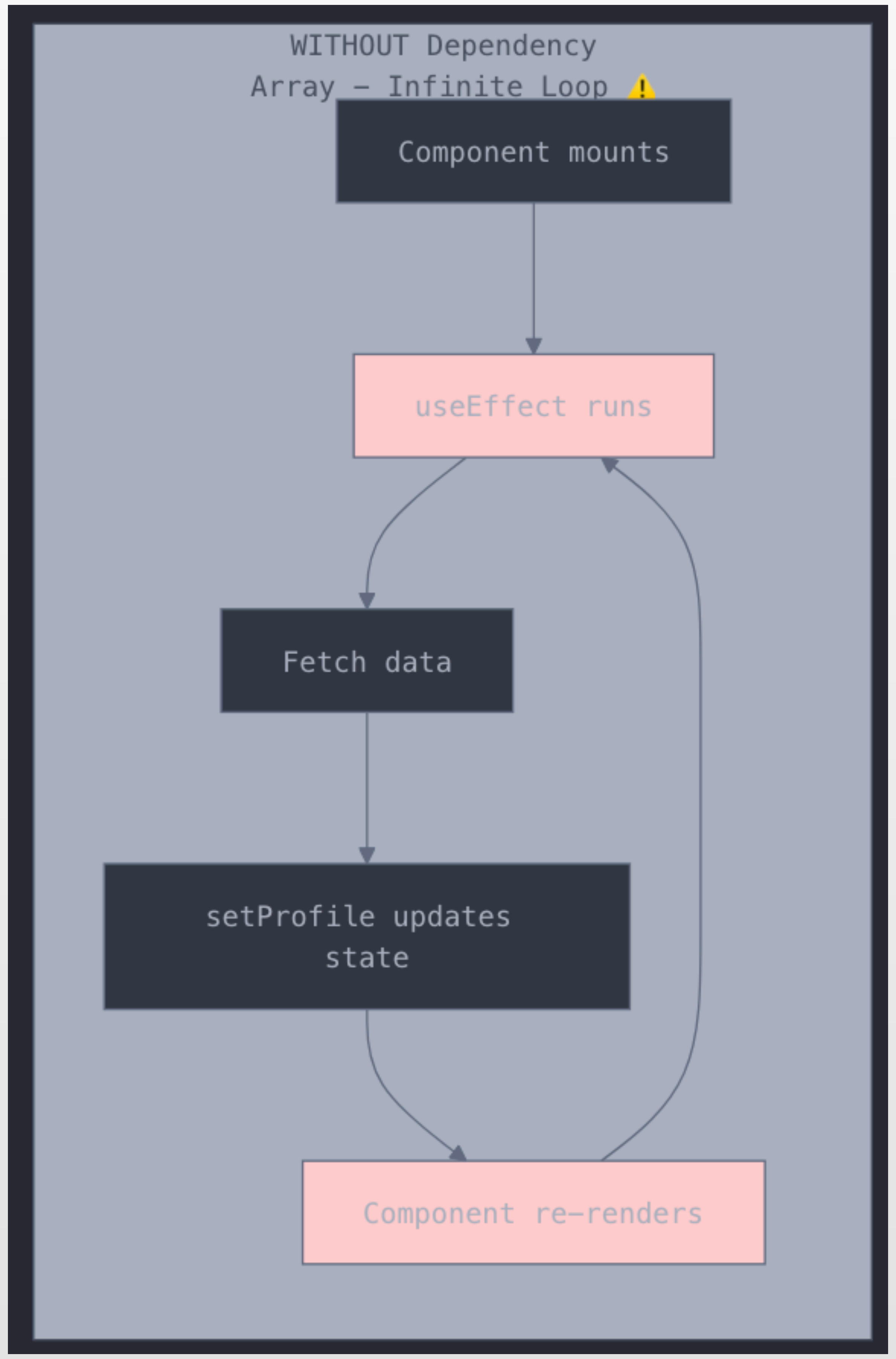
```javascript
function UserProfile({ userId }) {
  const [profile, setProfile] = useState(null);

  useEffect(() => {
    fetch(`https://api.example.com/users/${userId}`)
      .then(res => res.json())
      .then(data => setProfile(data));
  }, [userId]);

  return <div>{profile?.name}</div>;
}
```

When userId changes in Dashboard:

1. Dashboard's state changes (userId from 1 to 2)
2. Dashboard re-renders
3. UserProfile re-renders (because parent re-rendered)
4. UserProfile receives new prop userId={2}
5. After re-render completes, useEffect checks its dependencies
6. useEffect sees userId changed (1 → 2)
7. useEffect callback runs → fetch executes with new userId

UseEffects() - With & without dependency


WITH userId Dependency Array ✓
Component mounts with userId=1
useEffect runs
Fetch data for user 1
setProfile updates state
Component re-renders
Did userId change?
No
Yes, userId=2
useEffect does NOT run ✓
useEffect runs
Fetch data for user 2
setProfile updates state
Component re-renders


WITHOUT Dependency Array – Infinite Loop ⚠️
Component mounts
useEffect runs
Fetch data
setProfile updates state
Component re-renders