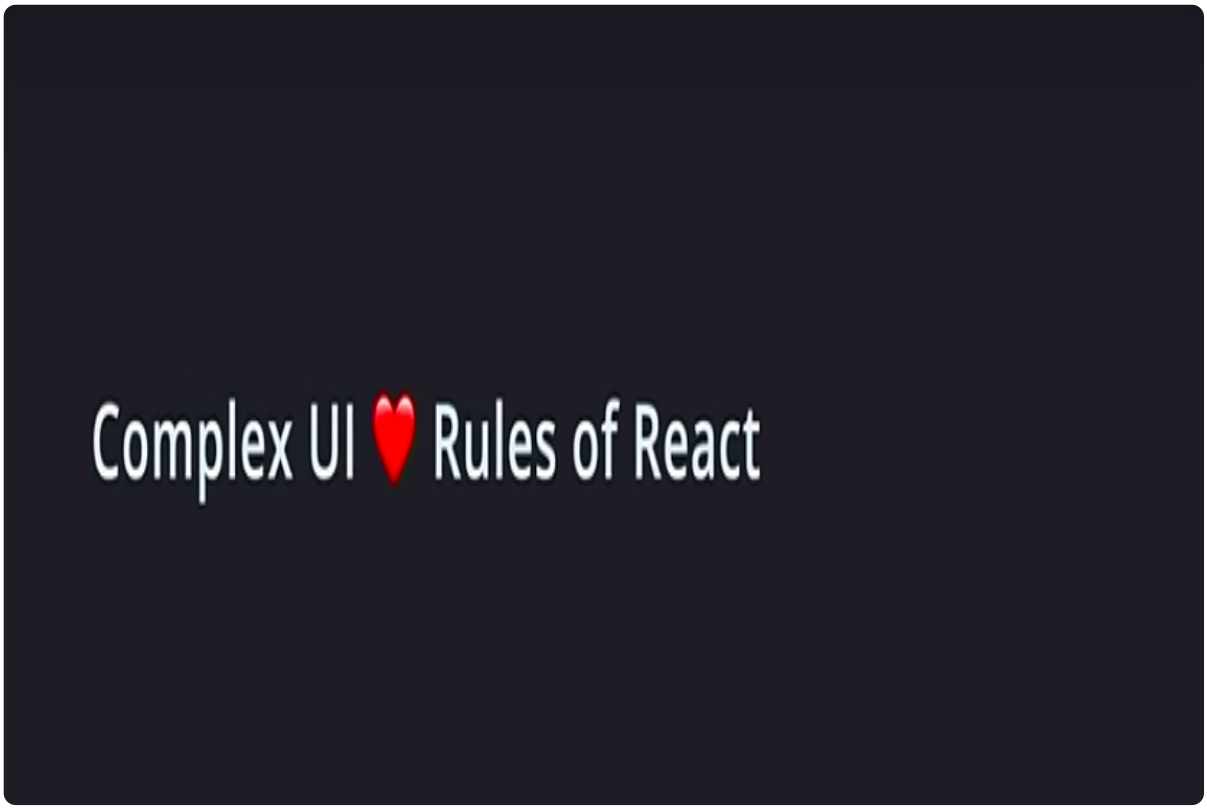
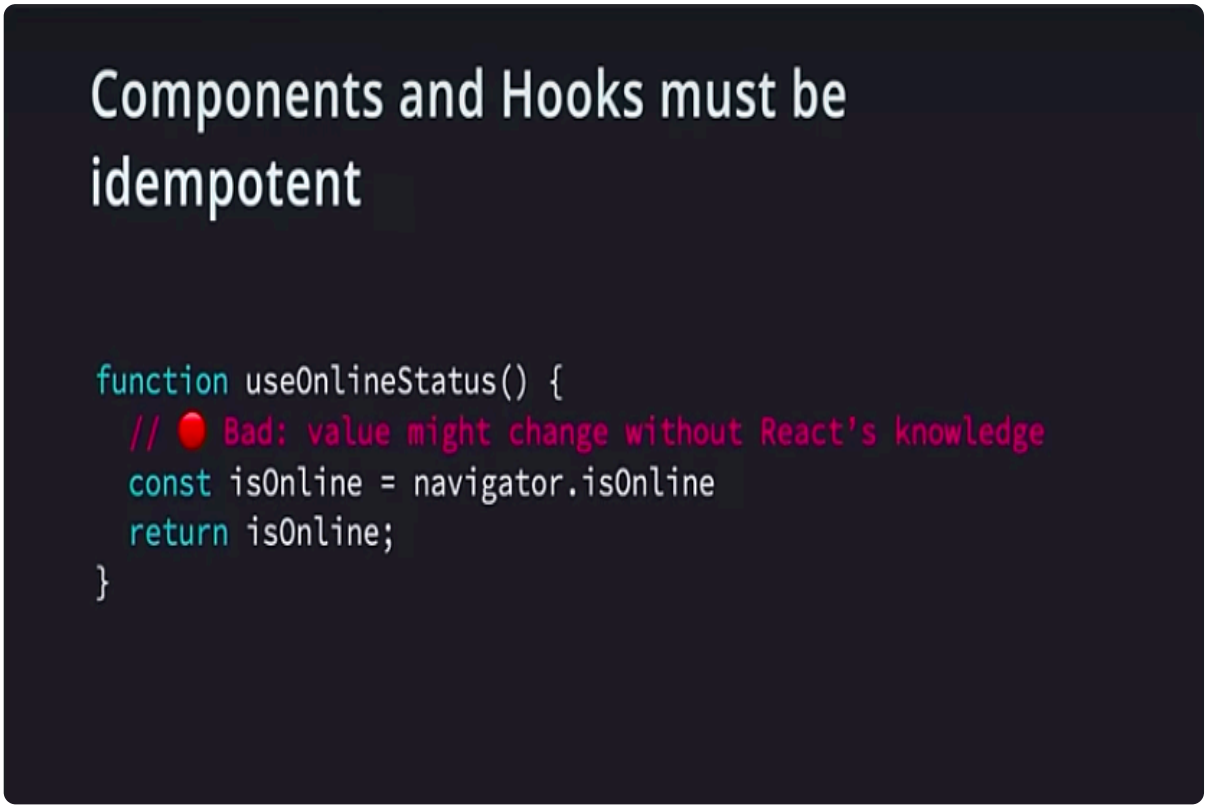
📑 React Compiler Deep Dive





Components and Hooks must be idempotent

```
function useOnlineStatus() {
  // ✅ Good: subscribe using useSyncExternalStore
  const isOnline = useSyncExternalStore(subscribe, getSnapshot);
  return isOnline;
}

function getSnapshot() { ... }

function subscribe(callback) { ... }
```

Side effects must run outside of render

```
let guest = 0;

function Cup() {
  // 🔴 Bad: changing a preexisting variable!
  guest = guest + 1;
  return <h2>Tea cup for guest #{guest}</h2>;
}
```

Side effects must run outside of render

```
// ✅ Good: pass a prop
function Cup({ guest }) {
  return <h2>Tea cup for guest #{guest}</h2>;
}
```

Props and State are immutable

```
function Post({ item }) {
  // 🔴 Bad: never mutate props directly
  item.url = new Url(item.url, base);
  return <Link url={item.url}>{item.title}</Link>;
}
```

Props and State are immutable

```
function Post({ item }) {
  // ✅ Good: make a copy instead
  const url = new Url(item.url, base);
  return <Link url={url}>{item.title}</Link>;
}
```

Return values and arguments to hooks are immutable

```
function useIconStyle(icon) {
  const theme = useContext(ThemeContext);
  if (icon.enabled) {
    // 🔴 Bad: never mutate hook arguments directly
    icon.className = computeStyle(icon, theme);
  }
  return icon;
}
```

Values are immutable after being passed to JSX

```
function Page({ colour }) {
  const styles = { colour, size: "large" };
  const header = <Header styles={styles} />;

  // 🔴 Bad: styles was already used in the JSX above
  styles.size = "small";
  return <Content header={header} styles={styles}/>;
}
```

Values are immutable after being passed to JSX

```
function Page({ colour }) {
  const styles = { colour, size: "large" };
  const header = <Header styles={styles} />;

  // ✅ Good: we created a new value
  const footerStyles = { colour, size: "small" };
  return <Content header={header} styles={footerStyles}/>;
}
```

React Compiler ❣️ Rules of React

We wrote react compiler?!

analyze memoization

analyze mutability

analyze aliasing

analyze (re)assignment

understand control flow

codegen JS

optimize memoization

...