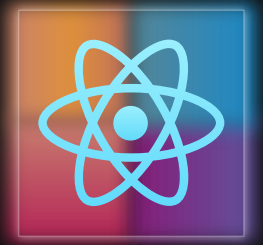
Enhancing React Performance: Mastering Re-render Optimization

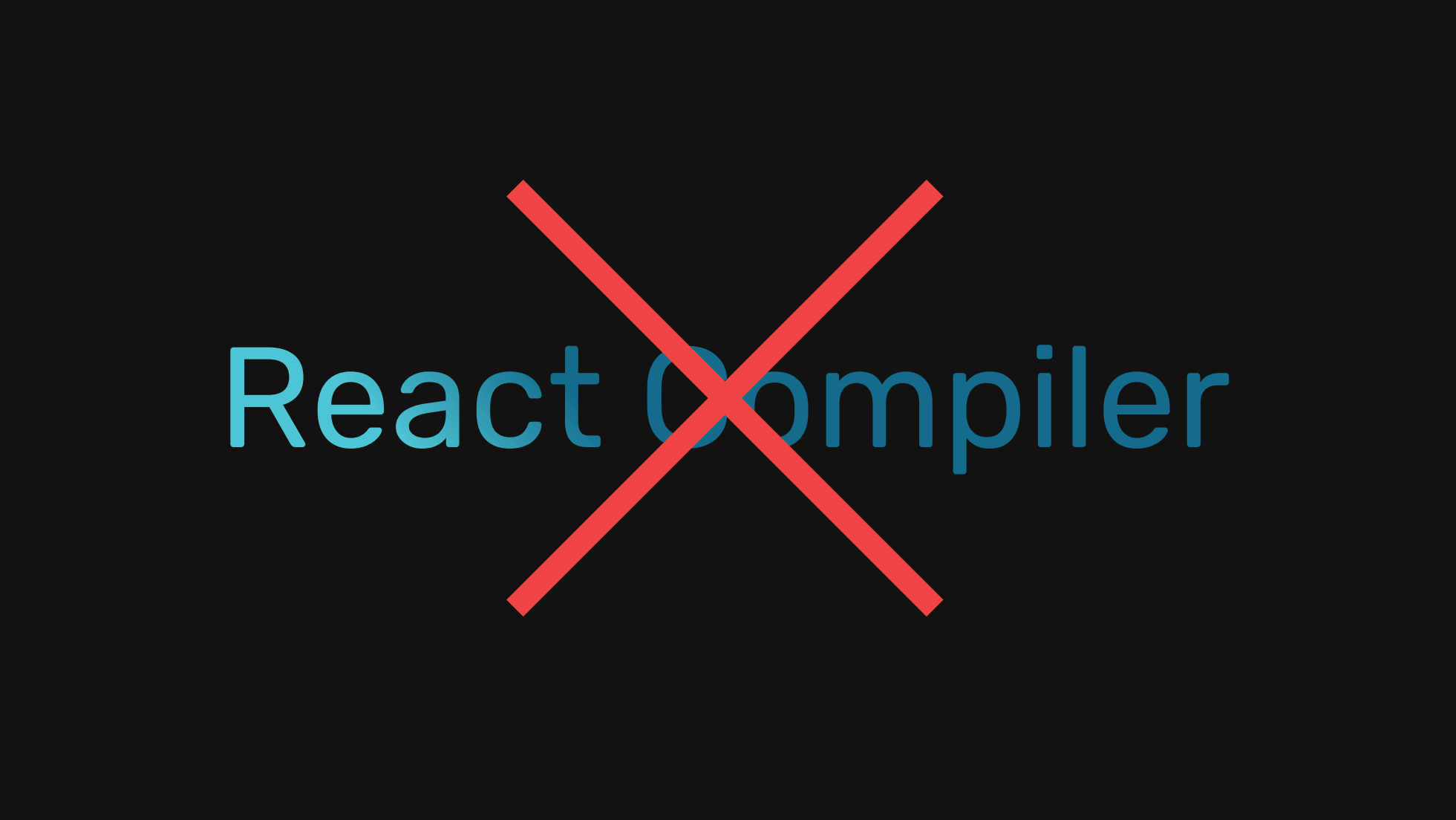
Tal Moskovich

What Is the Most Joyful Thing in React?



Avoiding Unnecessary Re-rendering


Memoization


React Compiler


React Forget


React Unforget

Just Plain React!

Hello, I’m Tal

🔥 **Committing Code &**
Pushing Personal Boundaries

- 👷 **Front-end Developer** @ Enpitech
- 🎧 **Podcaster & Lecturer** @ lotechni.dev
- 💡 **Proactivity Advocate**

Enpitech
Tailored Frontend Solutions

Let’s Start with a Quick(?) Example

```
function App() {
  const [state, setState] = useState({});
}
```

Let’s Start with a Quick(?) Example

```
function App() {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
    </FormContext.Provider>
  );
}
```

Let’s Start with a Quick(?) Example

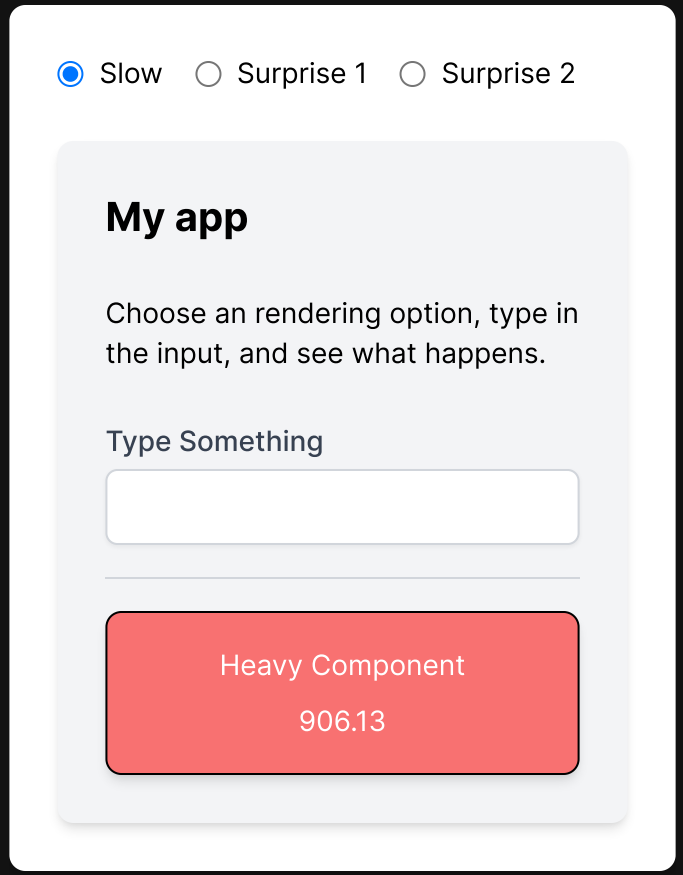
```
function App() {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
        <Form />
        <HeavyComponent /> /* A busy-awaited component */
    </FormContext.Provider>
  );
}
```

Let’s Start with a Quick(?) Example

```
function App() {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
        <Form />
        <HeavyComponent /> /* A busy-awaited component */
    </FormContext.Provider>
  );
}

function Form() {
  const [state, setState] = useContext(FormContext);
  // ...just a basic form
}
```

Let's Start with a Quick(?) Example

```
function App() {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
        <Form />
        <HeavyComponent /> /* A busy-awaited component */
    </FormContext.Provider>
  );
}

function Form() {
  const [state, setState] = useContext(FormContext);
  // ...just a basic form
}
```

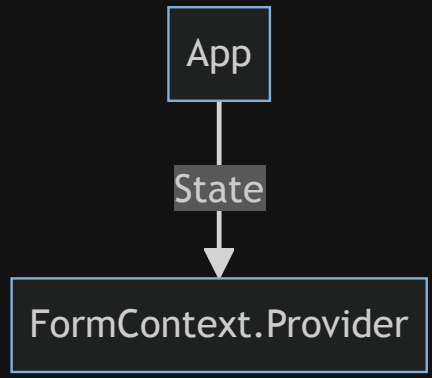


So, What Do We Have Here?

```
function App() {
  return (
    ...
  );
}
```

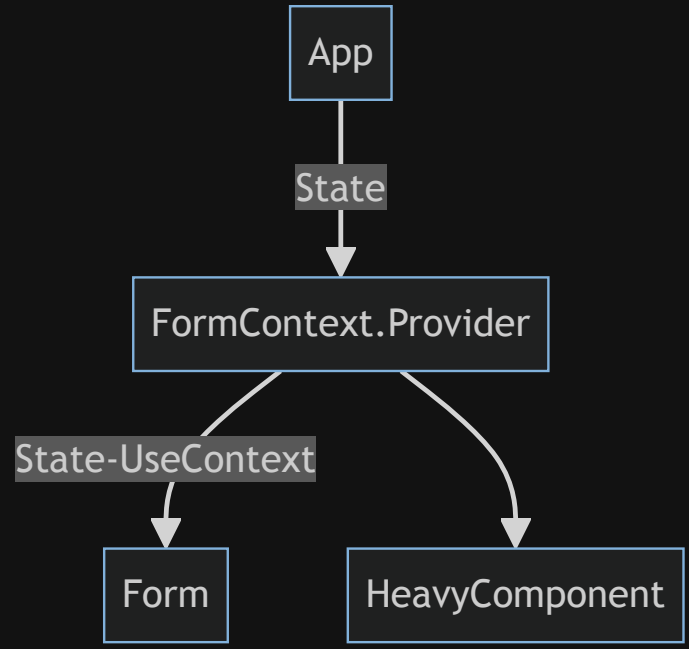
So, What Do We Have Here?

```
function App() {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
    </FormContext.Provider>
  );
}
```



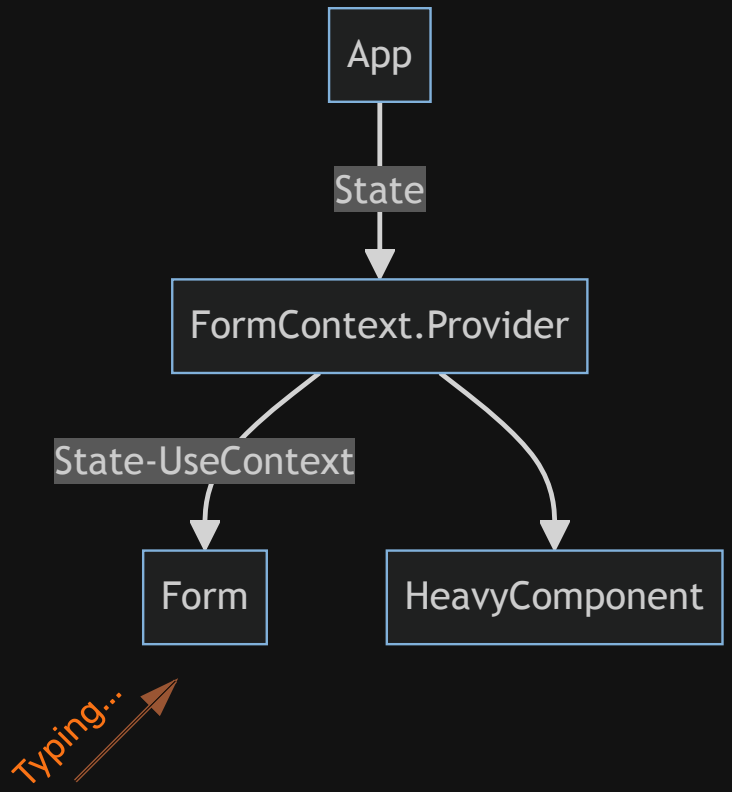
So, What Do We Have Here?

```
function App() {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      <Form />
      <HeavyComponent /> /* A busy-awaited component */
    </FormContext.Provider>
  );
}

function Form() {
  const [state, setState] = useContext(FormContext);
  // ...just a basic form
}
```



So, What Do We Have Here?

```
1  function App() {
2    const [state, setState] = useState({});
3
4    return (
5      <FormContext.Provider value={[state, setState]}>
6        <Form />
7        <HeavyComponent /> /* A busy-awaited component */
8      </FormContext.Provider>
9    );
10 }
11
12 function Form() {
13   const [state, setState] = useContext(FormContext);
14   // ...just a basic form
15 }
```



So, What Do We Have Here?

```
1  //Re-rendering...
2  function App() {
3    return (
4      ...
5    );
6  }
```

So, What Do We Have Here?

```
//Re-rendering...
function App() {
  return (
    ...
  );
}
```

Maybe a Better One?

Maybe a Better One?

```
function App() {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
        <Form />
        <HeavyComponent />
    </FormContext.Provider>
  );
}
```

Maybe a Better One?

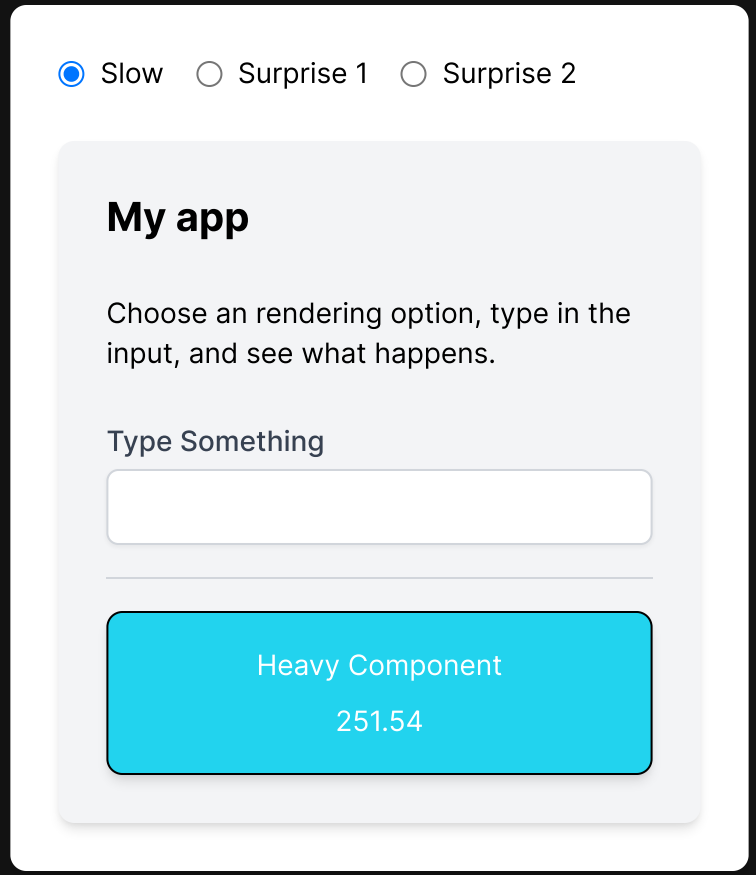
```
function App() {
  return (
    <ContextWrapper>
        <Form />
        <HeavyComponent />
    </ContextWrapper>
  );
}
```

Maybe a Better One?

```
function App() {
  return (
    <ContextWrapper>
        <Form />
        <HeavyComponent />
    </ContextWrapper>
  );
}

function ContextWrapper({ children }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {children}
    </FormContext.Provider>
  );
}
```

Maybe a Better One?

```jsx
function App() {
  return (
    <ContextWrapper>
        <Form />
        <HeavyComponent />
    </ContextWrapper>
  );
}

function ContextWrapper({ children }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {children}
    </FormContext.Provider>
  );
}
```

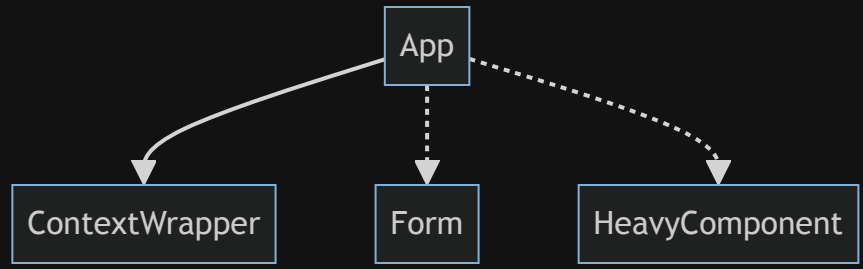


How Did We Get Here?

```
function App() {
  return (
    ...
  );
}
```

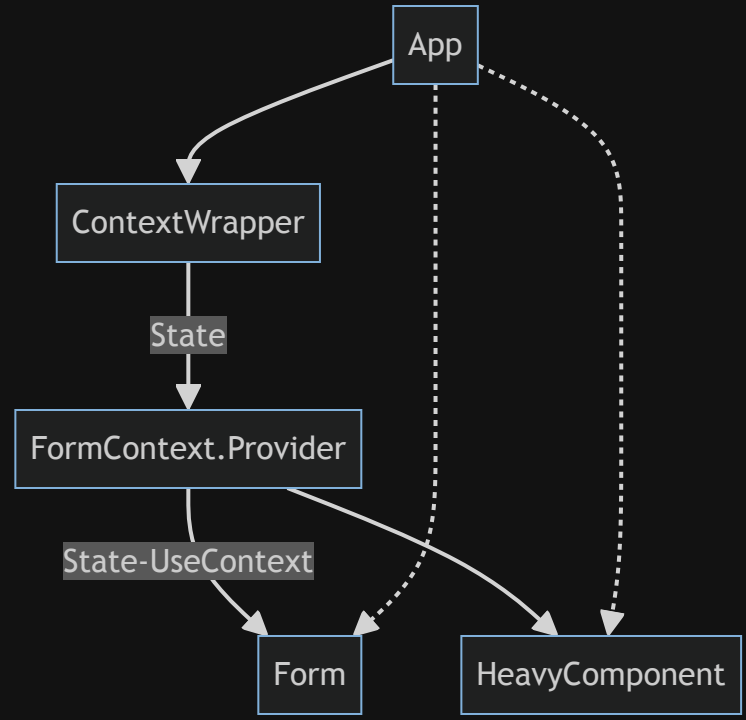
How Did We Get Here?

```
function App() {
  return (
    <ContextWrapper>
        <Form />
        <HeavyComponent />
    </ContextWrapper>
  );
}
```



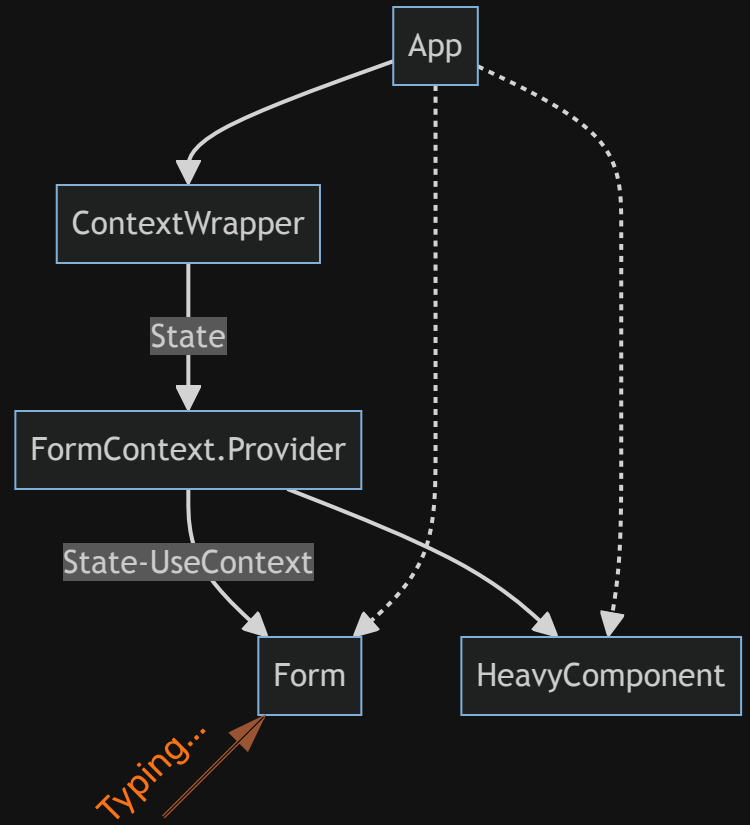
How Did We Get Here?

```jsx
function App() {
  return (
    <ContextWrapper>
        <Form />
        <HeavyComponent />
    </ContextWrapper>
  );
}

function ContextWrapper({ children }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {children}
    </FormContext.Provider>
  );
}
```



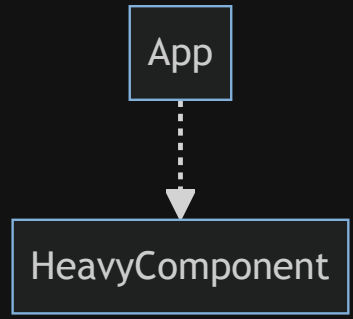
How Did We Get Here?

```jsx
function App() {
  return (
    <ContextWrapper>
        <Form />
        <HeavyComponent />
    </ContextWrapper>
  );
}

function ContextWrapper({ children }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {children}
    </FormContext.Provider>
  );
}
```



How Did We Get Here?

```jsx
function App() {
  return (
    <ContextWrapper>
        <Form /> //Re-rendered because context changing
        <HeavyComponent /> // 🤓 NOT RE-RENDERED
    </ContextWrapper>
  );
}

  /*
  Re-rendered because state changing
  */
function ContextWrapper({ children }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {children}
    </FormContext.Provider>
  );
}
```



How Did We Get Here?

```jsx
function App() {
  return (
    <ContextWrapper>
        <Form /> //Re-rendered because context changing
        <HeavyComponent /> // 🤓 NOT RE-RENDERED
    </ContextWrapper>
  );
}

  /*
  Re-rendered because state changing
  */
function ContextWrapper({ children }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {children}
    </FormContext.Provider>
  );
}
```

How Isn’t HeavyComponent Re-Rendered?

It’s all about how React works!

```
function Component(props) {
  return (...)
}
```

How Isn’t HeavyComponent Re-Rendered?

It’s all about how React works!

- 📦 **Props are Objects**
- 📂 **Props are saved in the Fiber nodes** (the "Virtual DOM")

A reference to an object in the Fiber node

```
function Component(props) {
  return (...)
}
```

How Isn’t HeavyComponent Re-Rendered?

It’s all about how React works!

- 📦 **Props are Objects**
- 📂 **Props are saved in the Fiber nodes** (the "Virtual DOM")

Therefore:

A reference to an object in the Fiber node

```
function Component({...}) {
  return (...)
}
```

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

- 📦 **Props are Objects**
- 📂 **Props are saved in the Fiber nodes** (the "Virtual DOM")

Therefore:

- 🔄 **Props Objects are re-created on each render**

A reference to an object in the Fiber node

```
function Component({...}) {
  return (...)
}
```

How Isn’t HeavyComponent Re-Rendered?

It’s all about how React works!

- 📦 **Props are Objects**
- 📂 **Props are saved in the Fiber nodes** (the "Virtual DOM")

Therefore:

- 🔄 **Props Objects are re-created on each render**

But what about `children` ?

```
function Component
({children}:{children: ReactNode})
{
  return (...)
}
```

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

- 👜 **Props are Objects**
- 📂 **Props are saved in the Fiber nodes** (the "Virtual DOM")

Therefore:

- 🔄 **Props Objects are re-created on each render**

ReactNode is actually a Fiber node

```
function Component
({children}:{children: ReactNode})
{
  return (...)
}
```

But what about `children` ?

- 🧩 **Children can be considered as a reference to other (rendered) Fiber nodes**
- 🧬 **Children save their referential identity from the previous render**

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

- 👜 **Props are Objects**
- 📂 **Props are saved in the Fiber nodes** (the "Virtual DOM")

Therefore:

- 🔄 **Props Objects are re-created on each render**

ReactNode is actually a Fiber node

```
function Component
({children}:{children: ReactNode})
{
  return (...)
}
```

But what about `children` ?

- 🧩 **Children can be considered as a reference to other (rendered) Fiber nodes**
- 🧬 **Children save their referential identity from the previous render**

Therefore:

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

- 👜 **Props are Objects**
- 📂 **Props are saved in the Fiber nodes** (the "Virtual DOM")

Therefore:

- 🔄 **Props Objects are re-created on each render**

ReactNode is actually a Fiber node

```
function Component
({children}:{children: ReactNode})
{
  return (...)
}
```

But what about `children` ?

- 🧩 **Children can be considered as a reference to other (rendered) Fiber nodes**
- 🧬 **Children save their referential identity from the previous render**

Therefore:

- 🛡️ **Our HeavyComponent isn't re-rendered!**

Will It Also Work?

Will It Also Work?

```
function App() {
  return (
    <ContextWrapper>
        <Form />
        <HeavyComponent />
    </ContextWrapper>
  );
}
```

Will It Also Work?

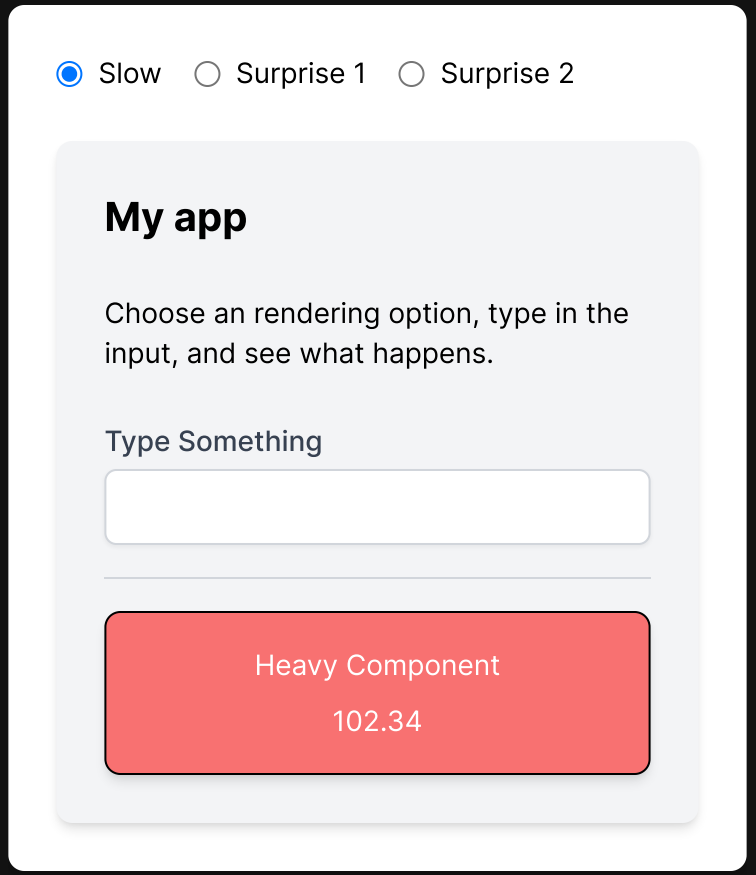
```
function App() {
  return (
    <ContextWrapper
        form={<Form />}
        heavyComponent={<HeavyComponent />}
    />
  );
}
```

Will It Also Work?

```
function App() {
  return (
    <ContextWrapper
        form={<Form />}
        heavyComponent={<HeavyComponent />}
    />
  );
}

function ContextWrapper({ form, heavyComponent }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {form}
      {heavyComponent}
    </FormContext.Provider>
  );
}
```

Will It Also Work?

```jsx
function App() {
  return (
    <ContextWrapper
        form={<Form />}
        heavyComponent={<HeavyComponent />}
    />
  );
}

function ContextWrapper({ form, heavyComponent }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {form}
      {heavyComponent}
    </FormContext.Provider>
  );
}
```



OK, But How?

Let's look at the code again:

```
function App() {
  return (
    <ContextWrapper
       form={<Form />}
       heavyComponent={<HeavyComponent />}
    />
  );
}

function ContextWrapper({ form, heavyComponent }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {form}
      {heavyComponent}
    </FormContext.Provider>
  );
}
```

OK, But How?

Let's look at the code again:

```
1  function App() {
2    return (
3      <ContextWrapper
4          form={<Form />}
5          heavyComponent={<HeavyComponent />}
6      />
7    );
8  }
9
10 function ContextWrapper({ form, heavyComponent }) {
11   const [state, setState] = useState({});
12
13   return (
14     <FormContext.Provider value={[state, setState]}>
15       {form}
16       {heavyComponent}
17     </FormContext.Provider>
18   );
19 }
```

OK, But How?

Let's look at the code again:

```
1  function App() {
2    return (
3      <ContextWrapper
4          form={<Form />}
5          heavyComponent={<HeavyComponent />}
6      />
7    );
8  }
9
10 function ContextWrapper({ form, heavyComponent }) {
11   const [state, setState] = useState({});
12
13   return (
14     <FormContext.Provider value={[state, setState]}>
15       {form}
16       {heavyComponent}
17     </FormContext.Provider>
18   );
19 }
```

`heavyComponent` gets a rendered (ReactNode) value

OK, But How?

Let's look at the code again:

```
function App() {
  return (
    <ContextWrapper
        form={<Form />}
        heavyComponent={<HeavyComponent />}
    />
  );
}

function ContextWrapper({ form, heavyComponent }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {form}
      {heavyComponent}
    </FormContext.Provider>
  );
}
```

'form' and 'heavyComponent' serves as slots

OK, But How?

Let's look at the code again:

```
1  function App() {
2    return (
3      <ContextWrapper
4          form={<Form />}
5          heavyComponent={<HeavyComponent />}
6      />
7    );
8  }
9
10 function ContextWrapper({ form, heavyComponent }) {
11   const [state, setState] = useState({});
12
13   return (
14     <FormContext.Provider value={[state, setState]}>
15       {form}
16       {heavyComponent}
17     </FormContext.Provider>
18   );
19 }
```

Typing in 'form' initiates state change

OK, But How?

Let's look at the code again:

```
function App() {
  return (
    <ContextWrapper
        form={<Form />}
        heavyComponent={<HeavyComponent />}
    />
  );
}

function ContextWrapper({ form, heavyComponent }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {form}
      {heavyComponent}
    </FormContext.Provider>
  );
}
```

Only 'Form' and 'ContextWrapper' are re-rendered!
And 'HeavyComponent' is "memoized"!

OK, But How?

Let's look at the code again:

```
function App() {
  return (
    <ContextWrapper
        form={<Form />}
        heavyComponent={<HeavyComponent />}
    />
  );
}

function ContextWrapper({ form, heavyComponent }) {
  const [state, setState] = useState({});

  return (
    <FormContext.Provider value={[state, setState]}>
      {form}
      {heavyComponent}
    </FormContext.Provider>
  );
}
```

From Props Drilling to Component Drilling

From Props Drilling to Component Drilling

- If you give React the same element you gave it on the last render, **it won't bother re-rendering** that element.

From Props Drilling to Component Drilling

- If you give React the same element you gave it on the last render, **it won't bother re-rendering** that element.
- "Lift" the expensive component to a parent where it will be rendered **less often**.

From Props Drilling to Component Drilling

- If you give React the same element you gave it on the last render, **it won't bother re-rendering** that element.
- "Lift" the expensive component to a parent where it will be rendered **less often**.
- Then pass the expensive component **down as a prop**.

Thanks For Listening!

Tal Moskovich

Code & Slides repo

Contact Me