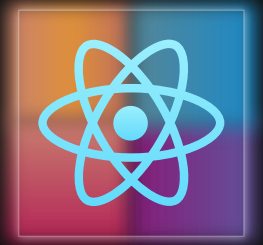
Enhancing React Performance: Mastering Re-render Optimization

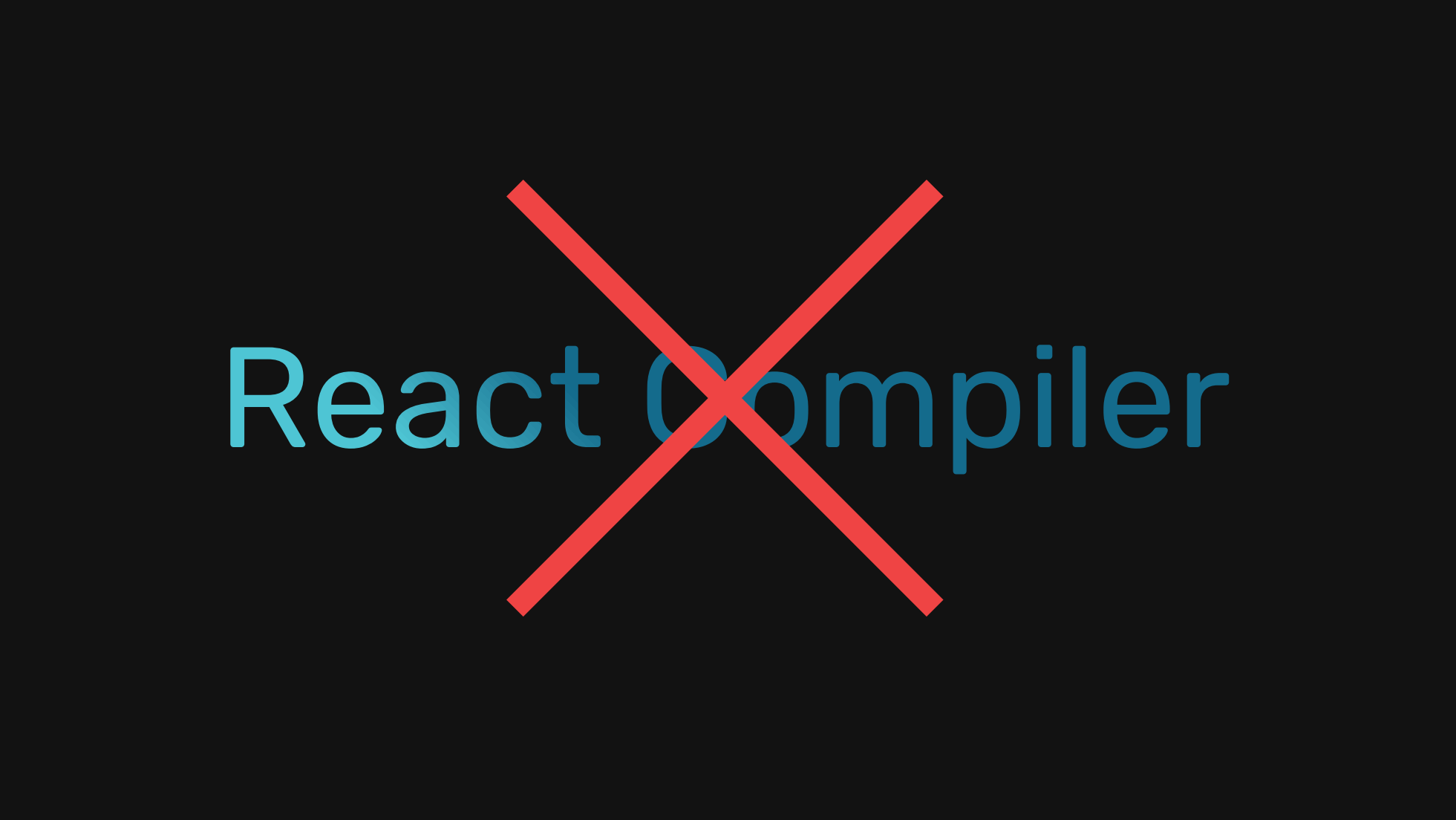
Tal Moskovich

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

לא טכני ולא במקרה
lotechni.dev

Let’s Start with a Quick(?) Example

Let's Start with a Quick(?) Example

```
1  function App() {
2    const [state, setState] = useState({});
3  }
```

Let's Start with a Quick(?) Example

```
1  function App() {
2    const [state, setState] = useState({});
3
4    return (
5      <FormContext.Provider value={[state, setState]}>
6      </FormContext.Provider>
7    );
8  }
```

Let's Start with a Quick(?) Example

```
1  function App() {
2    const [state, setState] = useState({});
3
4    return (
5      <FormContext.Provider value={[state, setState]}>
6          <Form />
7          <HeavyComponent />
8      </FormContext.Provider>
9    );
10 }
```

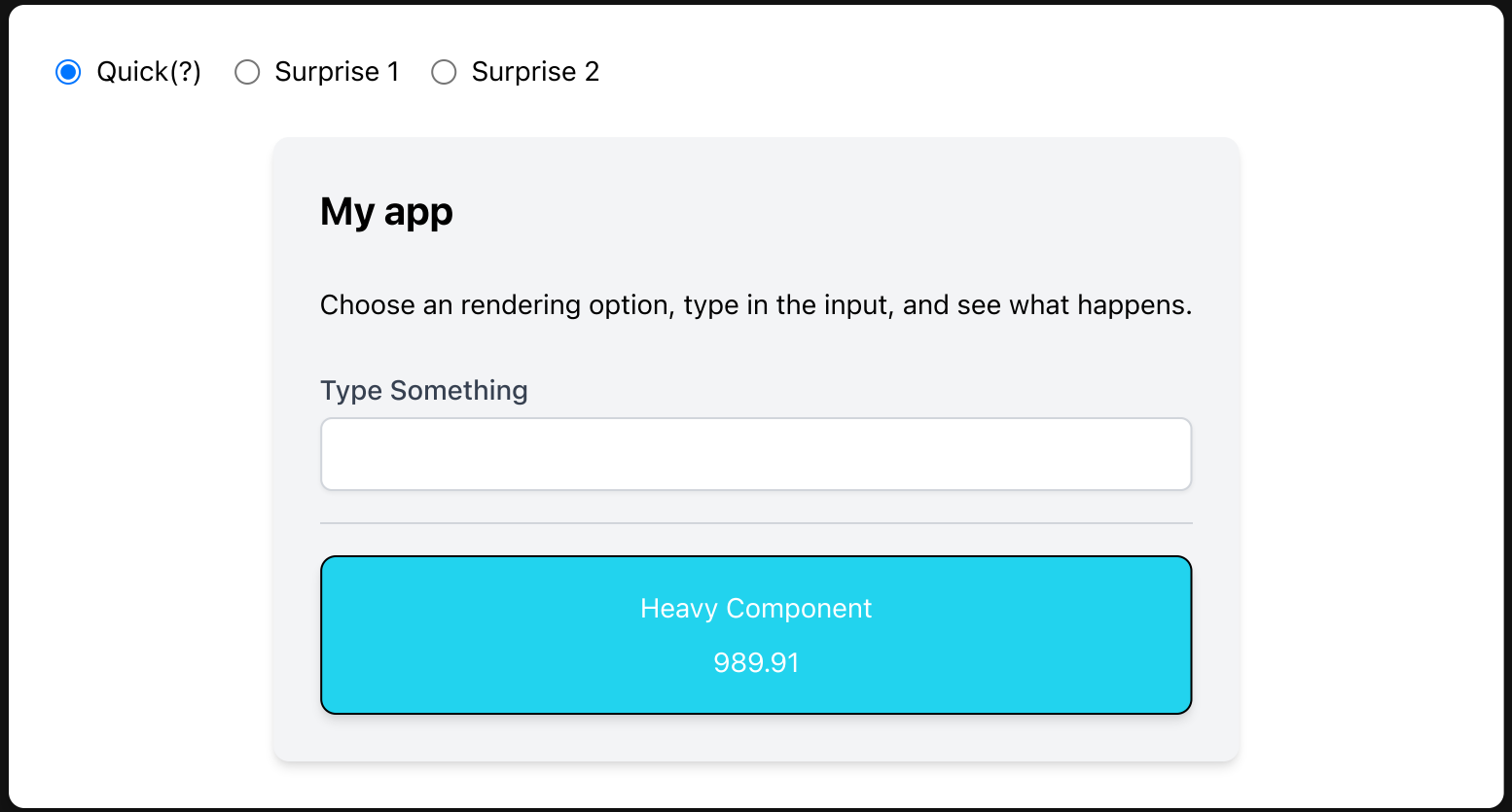
Don't share state

Let's Start with a Quick(?) Example

```
1  function Form() {
2    const [state, setState] = useContext(FormContext);
3
4    function handleChange(e) { ... }
5
6    return (
7        <input... />
8    );
9  }
```

Demo Time!

Demo Time!

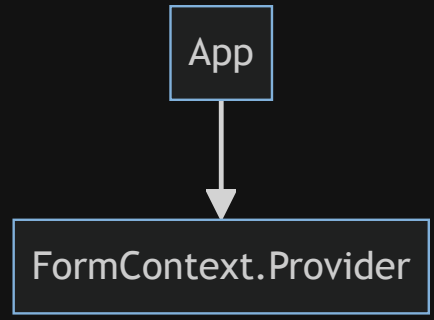

Quick(?)
Surprise 1
Surprise 2
My app
Choose an rendering option, type in the input, and see what happens.
Type Something
Heavy Component
989.91

So, What Do We Have Here?

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

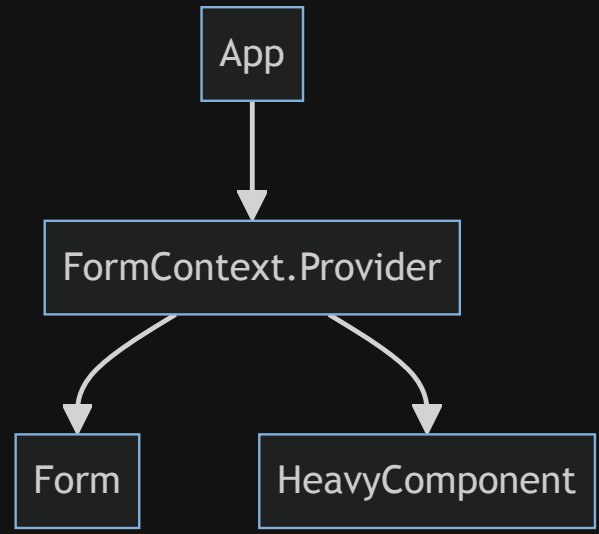
So, What Do We Have Here?

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

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



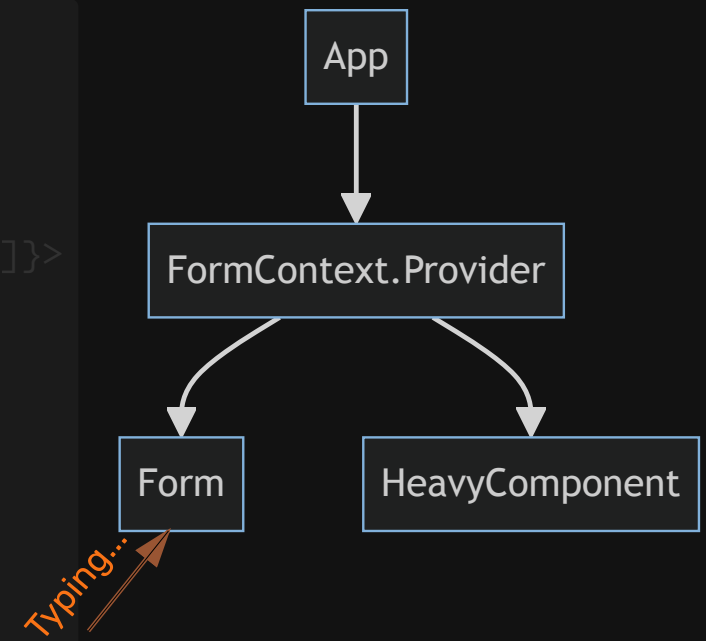
So, What Do We Have Here?

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



So, What Do We Have Here?

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

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

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



So, What Do We Have Here?

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

Maybe a Better Shot?

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

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

Maybe a Better Shot?

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

Maybe a Better Shot?

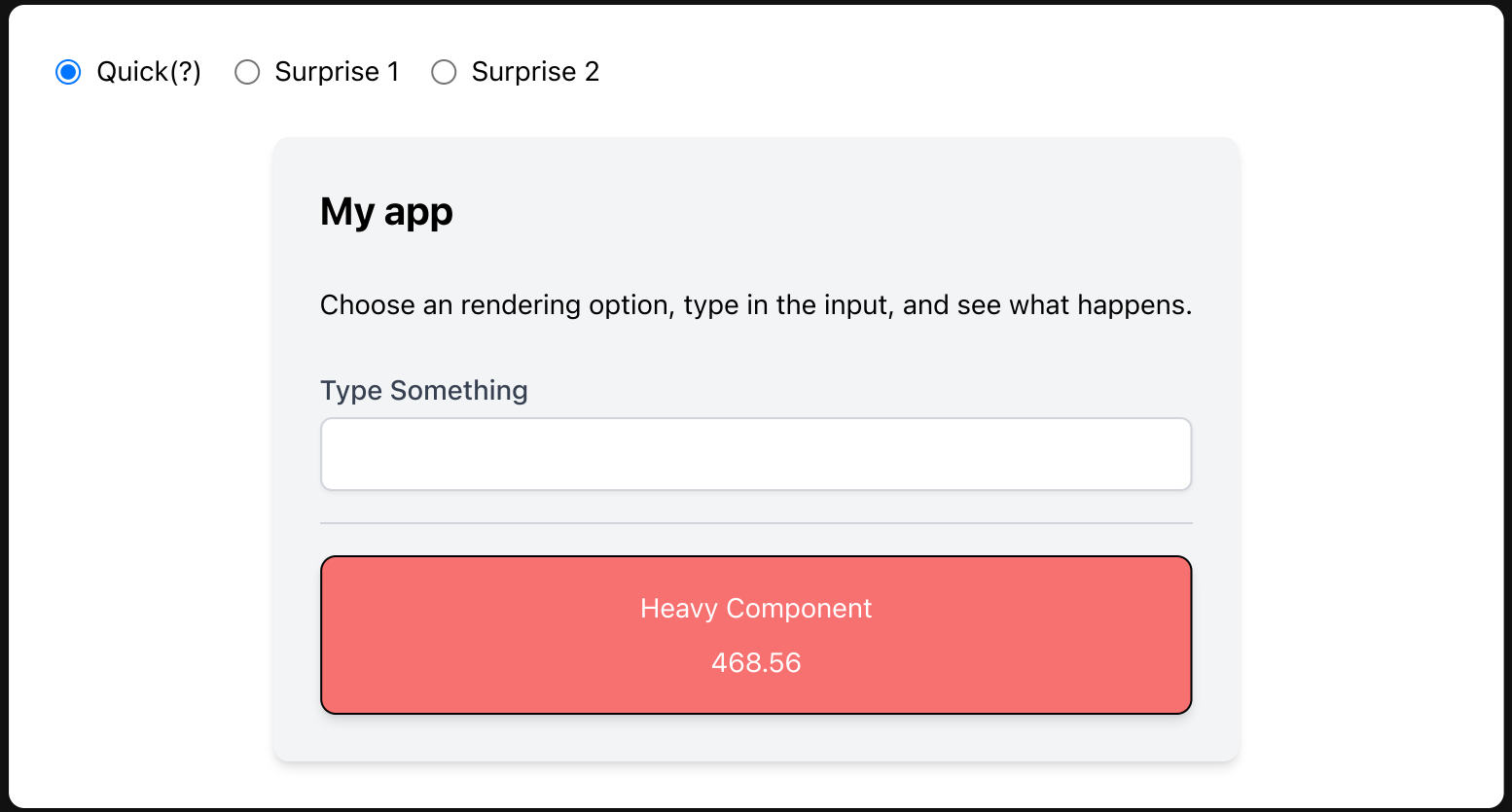
```
function ContextWrapper({ children }) {
  const [state, setState] = useState({});

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

Did I Change Anything?

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

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

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

Demo Time!

Demo Time!


Quick(?)
Surprise 1
Surprise 2
My app
Choose an rendering option, type in the input, and see what happens.
Type Something
Heavy Component
468.56

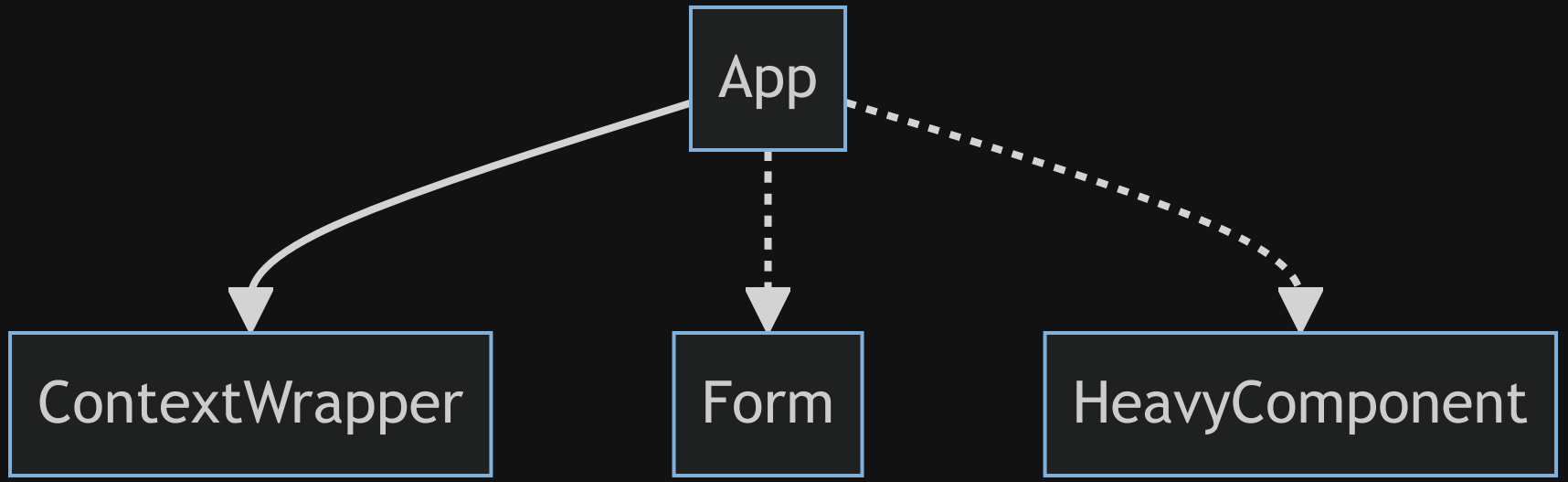
Did I Change Anything?

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

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

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

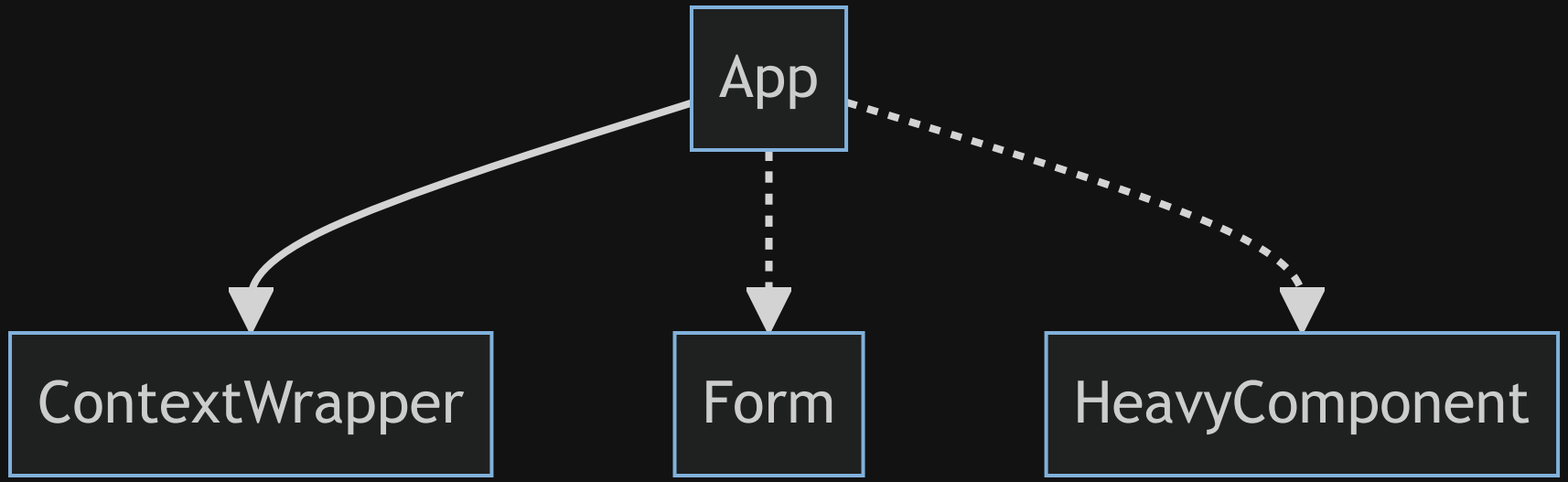
Did I Change Anything?

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

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

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

Did I Change Anything?



Did I Change Anything?



ContextWrapper's `children`,
are actually its SIBLINGS

How Isn’t HeavyComponent Re-Rendered?

It’s all about how React works!

```
function Component(props) {
  return <Child />;
}

// Somewhere else in the code
return <Component prop1={prop} />;
```

How Isn’t HeavyComponent Re-Rendered?

It’s all about how React works!

```
function Component(props) {
  return React.createElement(Child, null);
}

// Somewhere else in the code
return React.createElement(Component, {prop1: prop});
```

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

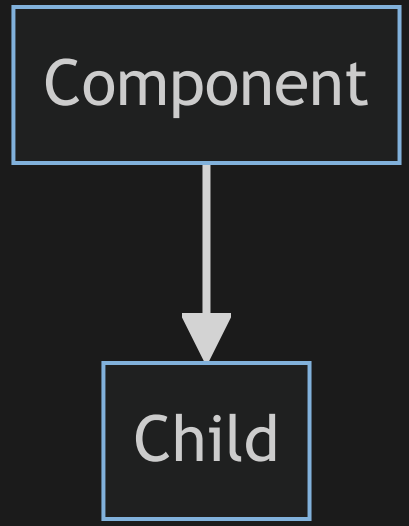
```
const ChildFiber = {
  type: Child,
  child: null,
  props: {}
};

const ComponentFiber = {
  type: Component,
  child: ChildFiber,
  props: { prop1: prop }
};
```

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

```
const ChildFiber = {
  type: Child,
  child: null,
  props: {}
};

const ComponentFiber = {
  type: Component,
  child: ChildFiber,
  props: { prop1: prop }
};
```

Hello, "VDOM"!



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!

A reference to the `props` object in the Fiber node

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

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

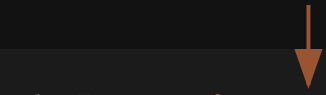
```
function Component({...}) {
  return <Child />;
}
```

Will be rendered
when the parent re-renders

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

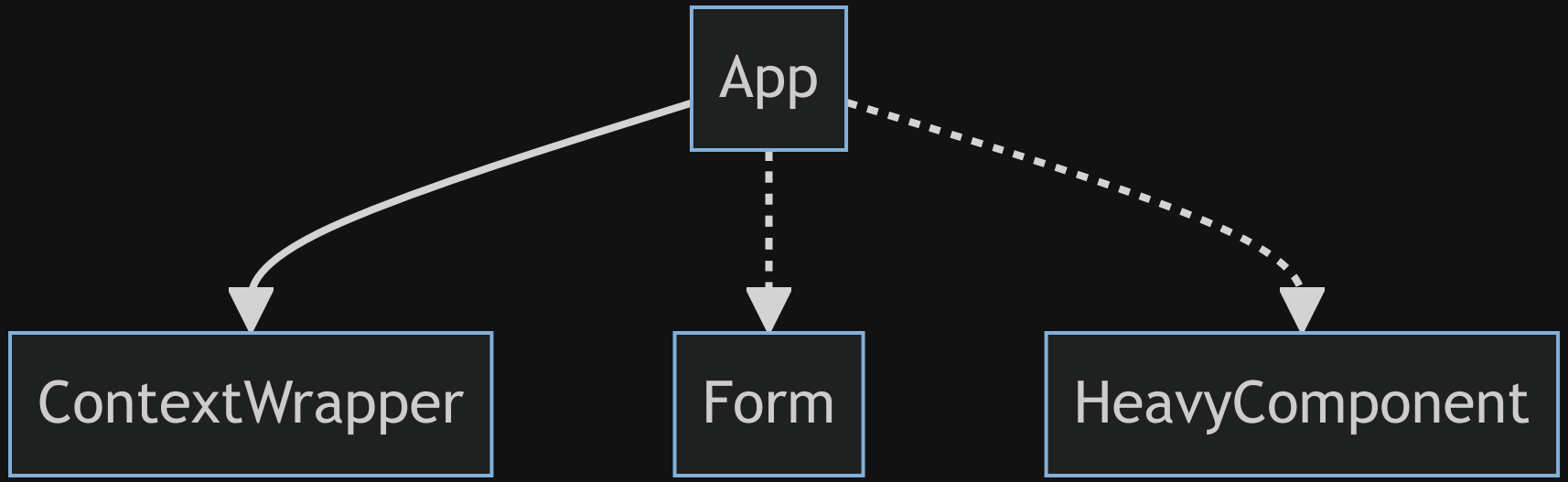
`children` is a rendered Fiber node



```
function Component({children}) {
  return <>{children}</>;
}
```

How Isn't HeavyComponent Re-Rendered?

It's all about how React works!



ContextWrapper's `children`,
are actually its SIBLINGS

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?

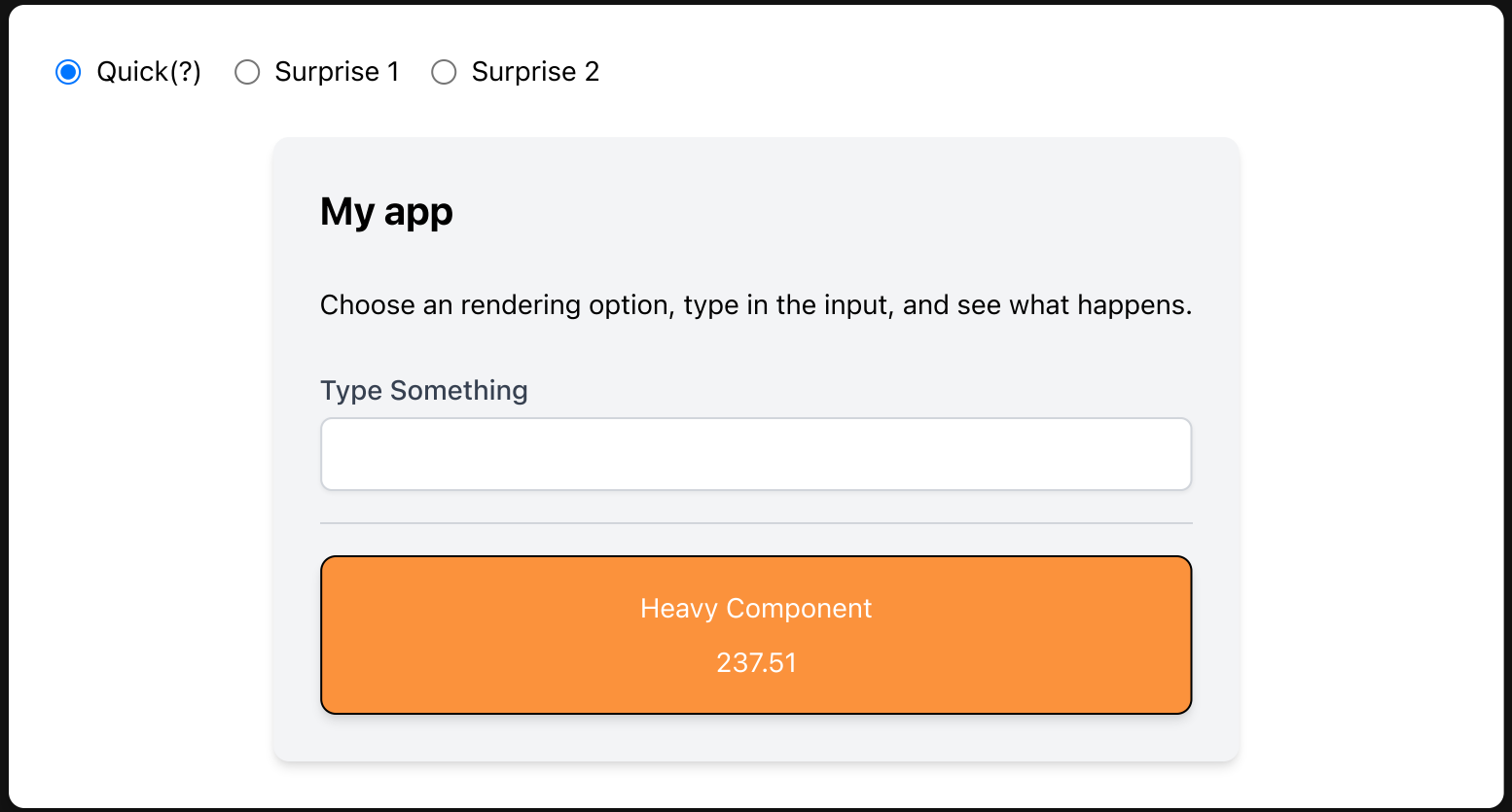
```
function ContextWrapper({ children }) {
  const [state, setState] = useState({});

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

Will It Also Work?

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

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

Demo Time!

Demo Time!


Quick(?)
Surprise 1
Surprise 2
My app
Choose an rendering option, type in the input, and see what happens.
Type Something
Heavy Component
237.51

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