

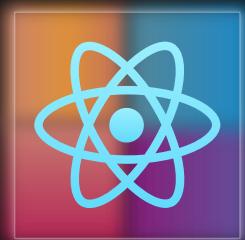
Enhancing React Performance: Mastering Re-render Optimization

Tal Moskovich

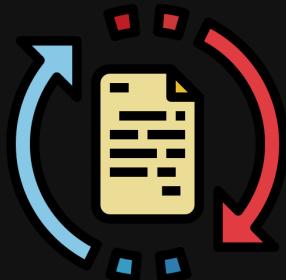


ReactNext

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



Tailored Frontend Solutions



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 />           ← Aren't share anything  
7             <HeavyComponent />  
8         </FormContext.Provider>  
9     );  
10 }
```

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

548.08

So, What Do We Have Here?

```
1 function App() {  
2   return (  
3     ...  
4   );  
5 }
```

App

So, What Do We Have Here?

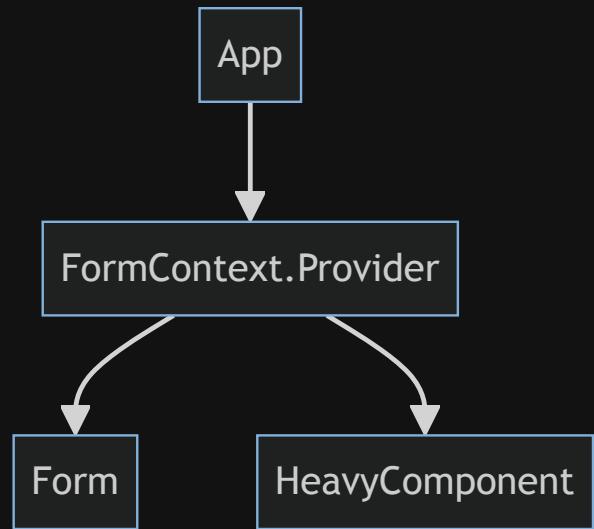
```
1 function App() {  
2   const [state, setState] = useState({});  
3  
4   return (  
5     <FormContext.Provider value={[state, setState]}>  
6       </FormContext.Provider>  
7     );  
8 }
```



FormContext.Provider

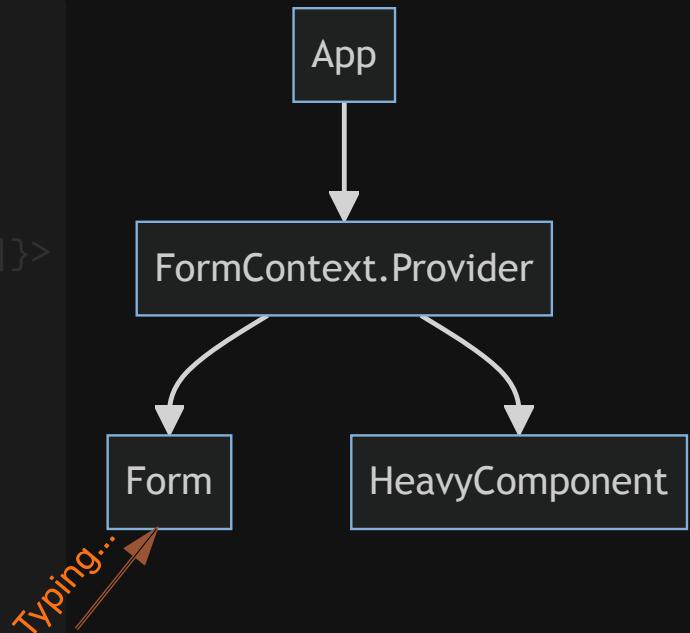
So, What Do We Have Here?

```
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?

```
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?

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

App

Maybe a Better One?

```
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 }
```

Maybe a Better One?

```
1 function App() {  
2   return (  
3     <ContextWrapper>  
4       <Form />  
5       <HeavyComponent />  
6     </ContextWrapper>  
7   );  
8 }
```

Maybe a Better One?

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

Did I Change Something?

```
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

934.33

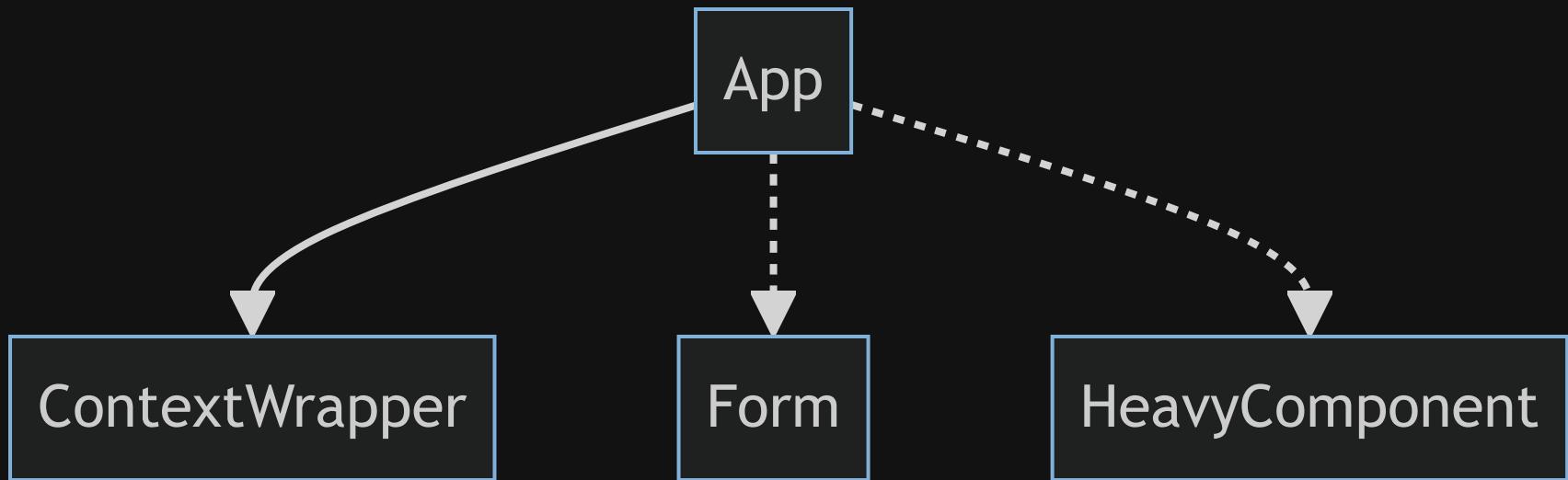
Did I Change Something?

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

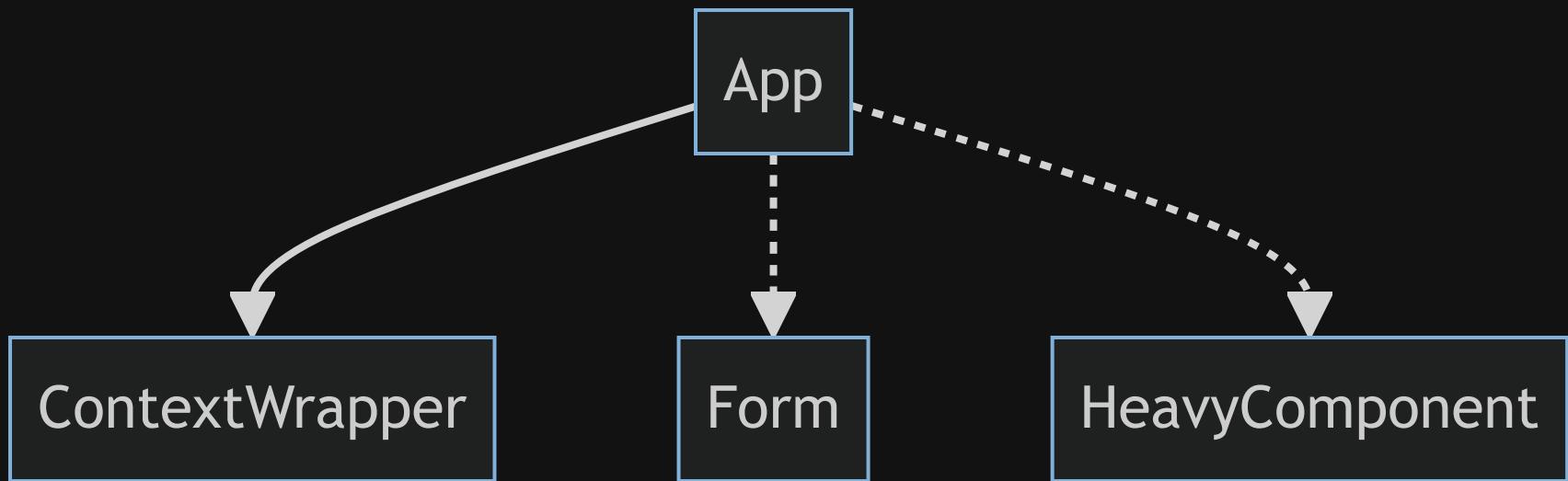
Did I Change Something?

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

Did I Change Something?



Did I Change Something?



ContextWrapper

Form

HeavyComponent

ContextWrapper children,
are actually 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 }  
};
```

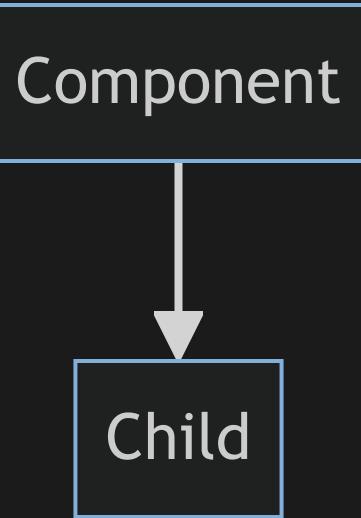
How Isn't HeavyComponent Re-Rendered?

It's all about how React works!

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

Hello, "VDOM"!

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



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!

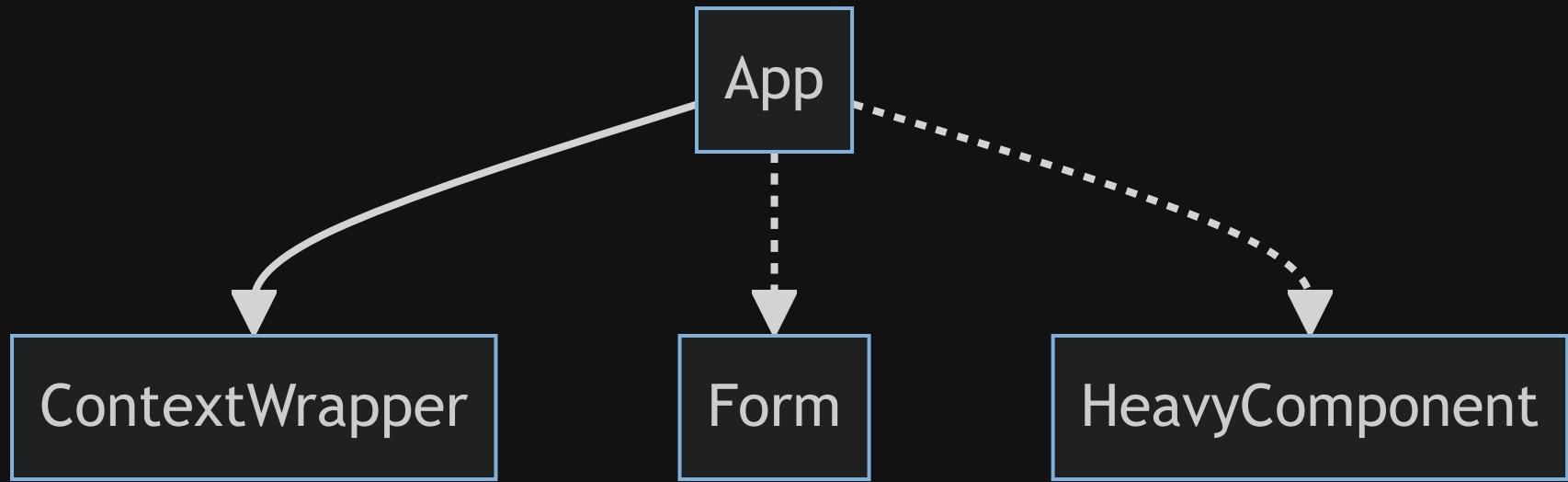
```
function Component({children}) {  
  return <>{children}</>;  
}
```

children is a
rendered Fiber node



How Isn't HeavyComponent Re-Rendered?

It's all about how React works!



ContextWrapper children,
are actually SIBLINGS

Will It Also Work?

```
1  function App() {  
2    return (  
3      <ContextWrapper>  
4        <Form />  
5        <HeavyComponent />  
6      </ContextWrapper>  
7    );  
8  }
```

Will It Also Work?

```
1  function App() {  
2    return (  
3      <ContextWrapper  
4        form={<Form />}  
5        heavyComponent={<HeavyComponent />}  
6      />  
7    );  
8  }
```

Will It Also Work?

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

Will It Also Work?

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

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

956.71

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](#)

I will be speaking at

ReactNext2024



Tal Moskovich

Frontend & People Developer @ Enpitech

Enhancing React Performance: Mastering Re-render Optimization