React Hooks 🪝

History

Hooks

The rules

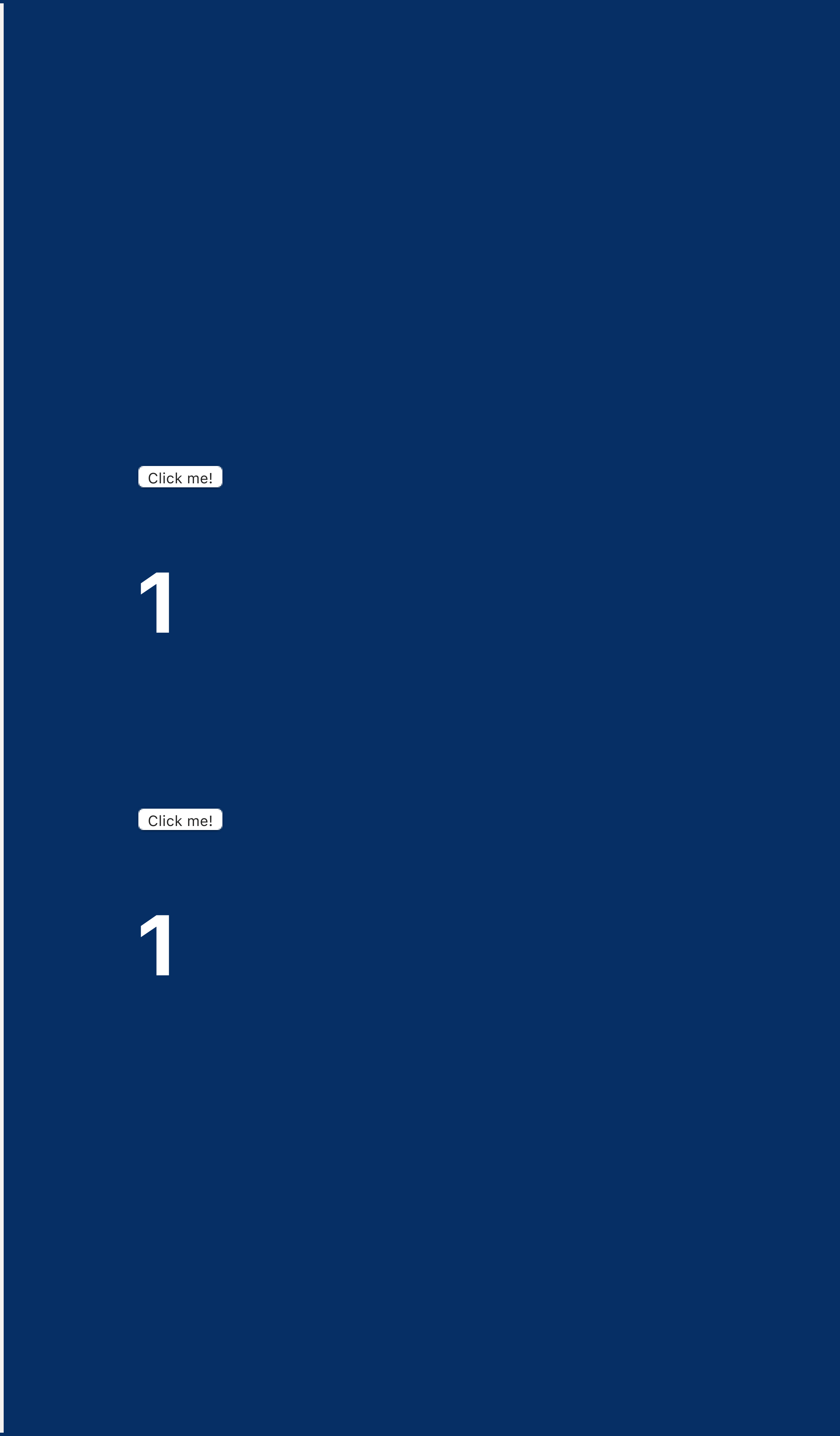
Gotchas/Performance

History

```
class DoubleApp extends React.Component {
  constructor() {
    super();
    this.state = {
      number: 1,
    };
  }

  double() {
    this.setState({ number: this.state.number * 2 });
  }

  render() {
    return (
      <div style={{ margin: "50px" }}>
        <button onClick={() => this.double()}>Click me!</button>
        <h1>{this.state.number}</h1>
      </div>
    );
  }
}
```



```
componentDidMount() {
  this.timer = setInterval(() => {
    this.setState({ number: this.state.number + 1 });
  }, 1000);
}

componentWillUnmount() {
  clearTimeout(this.timer);
}

render() {
  return <h1>{this.state.number}</h1>
}
```

2

No shared logic 🥵

Hard to follow state flow 🍝

Typescript types are complicated 😕

Recompose

github.com/acdlite/recompose

```
import { withState } from "recompose";

const enhance = withState("counter", "setCounter", 0);
const Counter = enhance(({ counter, setCounter }) => (
  <div>
    Count: {counter}
    <button onClick={() => setCounter((n) => n + 1)}>Increment</button>
    <button onClick={() => setCounter((n) => n - 1)}>Decrement</button>
  </div>
));
```

Hooks?

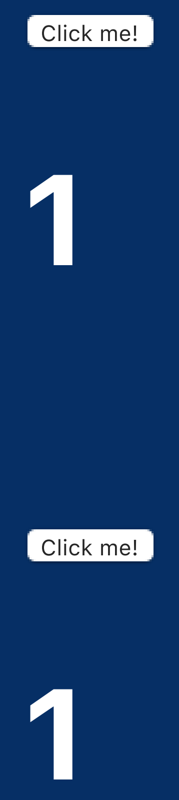
Introduced in React 16.8

NOT a breaking change and interact perfectly with older APIs

useState

```jsx
import React, { useState } from "react";

const DoubleApp = () => {
  const [number, setNumber] = useState(1);
  return (
    <div>
      <button
        onClick={() => {
          setNumber(number * 2);
        }}
      >
        Click me!
      </button>
      <h1>{number}</h1>
    </div>
  );
};
```



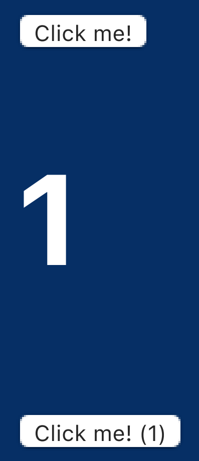
```
import React, { useState, useCallback } from "react";

const useDoubler = () => {
  const [number, setNumber] = useState(1);

  return [number, () => setNumber(number * 2)];
};
```

```
const Component1 = () => {
  const [number, double] = useDoubler();
  return (
    <div>
      <button onClick={double}>Click me!</button>
      <h1>{number}</h1>
    </div>
  );
};

const Component2 = () => {
  const [number, double] = useDoubler();
  return <button onClick={double}>Click me! ({number})</button>;
};
```



useEffect

```
const CountApp = () => {
  const [number, setNumber] = useState(1);

  useEffect(() => {
    // Component did mount
    const timeout = setInterval(() => {
      setNumber((n) => n + 1);
    }, 1000);

    // Component will unmount
    return () => clearTimeout(timeout);
  }, [setNumber]);

  return <h1>{number}</h1>;
};
```

2

```
const useCount = () => {
  const [number, setNumber] = useState(1);

  useEffect(() => {
    const timeout = setInterval(() => {
      setNumber((n) => n + 1);
    }, 1000);

    return () => clearTimeout(timeout);
  }, [setNumber]);

  return number;
};
```

```
const CountComponent1 = () => {
  const number = useCount();
  return <h1>{number}</h1>;
};

const CountComponent2 = () => {
  const number = useCount();
  return <button>Some button {number}</button>;
};
```

2

Some button 2

useContext

```
import React, { createContext, useContext } from "react";

const MyContext = createContext();

const Component = () => {
  const data = useContext(MyContext);
  return <h1>{data}</h1>;
};

const App = () => {
  return (
    <MyContext.Provider value="Bulb!">
      <Component />
    </MyContext.Provider>
  );
};
```

Bulb!

Hooks are composeable! 🏗️

The Primary Colours 🎨

`useState`

`useEffect`

`useContext`

`useRef`

`useMemo`

`useCallback`

`useReducer`

…

useRef

```
const Component = () => {
  const divRef = useRef();
  const [width, setWidth] = useState();

  useEffect(() => {
    setWidth(divRef.current.clientWidth);
  }, [divRef]);

  const time = new Date().toTimeString().split(" ")[0];
  return (
    <div>
      <h1 ref={divRef}>Hello lunch and learn! It's {time}</h1>
      {width && <h2>^ That's {width}px wide</h2>}
    </div>
  );
};
```

Hello lunch and learn! It's 12:45:00

^ That's 269px wide

```
const useRef = (initialValue) => {
  const ref = useState(() => {
    const refFunc = (newValue) => {
      ref.current = newValue;
    };
    refFunc.current = initialValue;

    return refFunc;
  });

  return ref;
};
```

useMemo

```
const Component = () => {
  const data = useMemo(() => {
    // Long running calc
    // or same reference
  }, [depsForCalc])
  ...
}
```

```
import React, { useRef } from "react";

const useMemo = (factory, deps) => {
  const depsRef = useRef(null);
  const memoriedRef = useRef(null);

  if (!depsRef.current || !isSame(depsRef.current, deps)) {
    depsRef.current = deps;
    memoriedRef.current = factory();
  }

  return memoriedRef.current;
};
```

useCallback

```
import React, { useRef } from "react";

const useCallback = (callback, deps) => useMemo(() => callback, deps);
```

`useState`

⬇️

`useRef`

⬇️

`useMemo`

⬇️

`useCallback`

Disclaimer

This is not how react composes these in production

But how do they work?

```
const DoubleApp = () => {
  const [number, setNumber] = useState(1);

  ...
};
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver"); // <---
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Oliver", // <--
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586"); // <---

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Oliver",
  "07584838586", // <--
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  useEffect(() => {
    document.title = name;
  }, [name]);  // <---

  ...
};
```

"Memory"

```
[
  "Oliver",
  "07584838586",
  ["Oliver"], // <---
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  useEffect(() => {
    document.title = name; // <---
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Oliver",
  "07584838586",
  ["Oliver"], // <---
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
  // event -> setName("Ollie")
};
```

"Memory"

```
[
  -"Oliver",
  +"Ollie", // Replace
  "07584838586",
  ["Oliver"],
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver"); // <---
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Ollie", // <---
  "07584838586",
  ["Oliver"],
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586"); // <---

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Ollie",
  "07584838586", // <---
  ["Oliver"],
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  useEffect(() => {
    document.title = name;
  }, [name]); // <---

  ...
};
```

"Memory"

```
[
  "Ollie",
  "07584838586",
  ["Oliver"], // <---
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  useEffect(() => {
    document.title = name; // <---
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Ollie",
  "07584838586",
  ["Ollie"], // <---
];
```

Rules of hooks 101

🚨 Don’t call Hooks inside loops, conditions, or nested functions 🚨

```
const Component = () => {
  const [name, setName] = useState("Oliver"); // <---
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  if (name === "Ollie") {
    const details = useData(name);

    return <div>{details}</div>
  }

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Ollie", // <---
  "07584838586",
  ["Oliver"],
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586"); // <---

  if (name === "Ollie") {
    const details = useData(name);

    return <div>{details}</div>
  }

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Ollie",
  "07584838586", // <---
  ["Oliver"],
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  if (name === "Ollie") {
    const details = useData(name); // <---

    return <div>{details}</div>
  }

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Ollie",
  "07584838586",
  ["Oliver"], // <--- 🥴 borked
];
```

```
const Component = () => {
  const [name, setName] = useState("Oliver");
  const [
    phoneNumber,
    setPhoneNumber
  ] = useState("07584838586");

  const details = useData(name, { skip: name !== "Ollie" });

  useEffect(() => {
    document.title = name;
  }, [name]);

  ...
};
```

"Memory"

```
[
  "Ollie",
  "07584838586",
  ...
];
```

Further reading

reactjs.org/docs/hooks-rules.html

Believe the linter!

Gotchas

`useEffect` deps are confusing

```
const useMyHook = () => {
  return {
    someData: ...,
    someOtherData: ...
  }
}

const Component = () => {
  const data = useMyHook(...)

  useEffect(() => {
    ...
    // Gonna be a bad time
  }, [data]);
}
```

github.com/kentcdodds/use-deep-compare-effect

WARNING: Please only use this if you really can't find a way to use `React.useEffect`.

There's often a better way to do what you're trying to do than a deep comparison.

```
const Component = () => {
  const { someData } = useMyHook(...)

  useEffect(() => {
    ...
  }, [someData]);
}
```

```
useMemo(
  () => ({
    someData,
    someOtherData,
  }),
  [someData, someOtherData]
);
```

```
useCallback(() => setNumber((n) => n + 1), [setNumber]);
```

Performance 🏎️

Hooks = more lines executing per render

`useState`

⬇️

`useRef`

⬇️

`useMemo`

⬇️

`useCallback`

🙏 JS engine gods auto optimise repetative code

Your bottleneck is very likely to be DOM rendering

Don't optimise unless you really need to

```
const Component = () => {
  const data = useMyHook(...)

  useEffect(() => {
    ...
    // Gonna be a bad time
  }, [data]);
}
```

Source

reactjs.org/docs/hooks-intro.html

I ❤️ Hooks

You should too