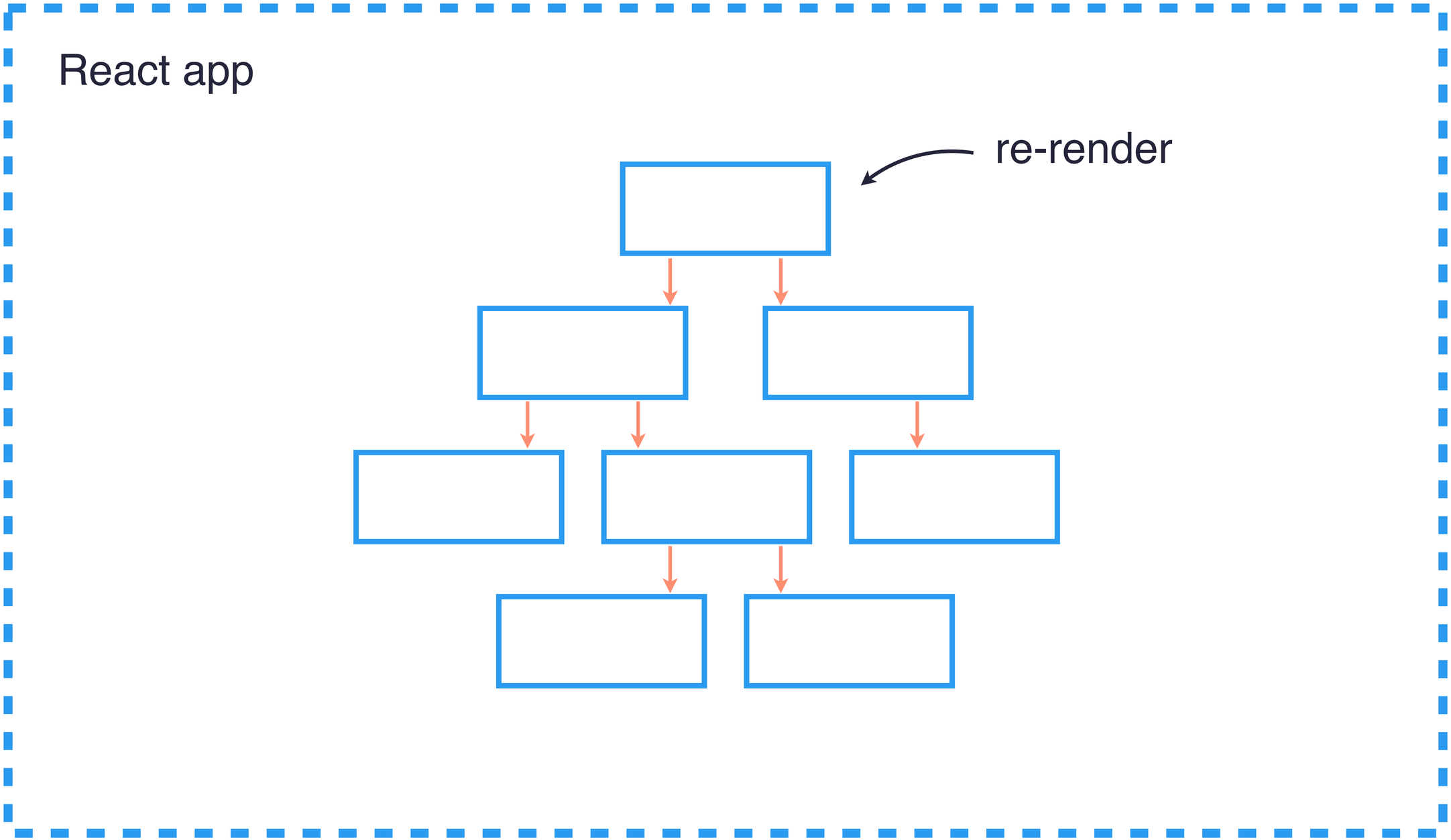
Let's talk about re-renders

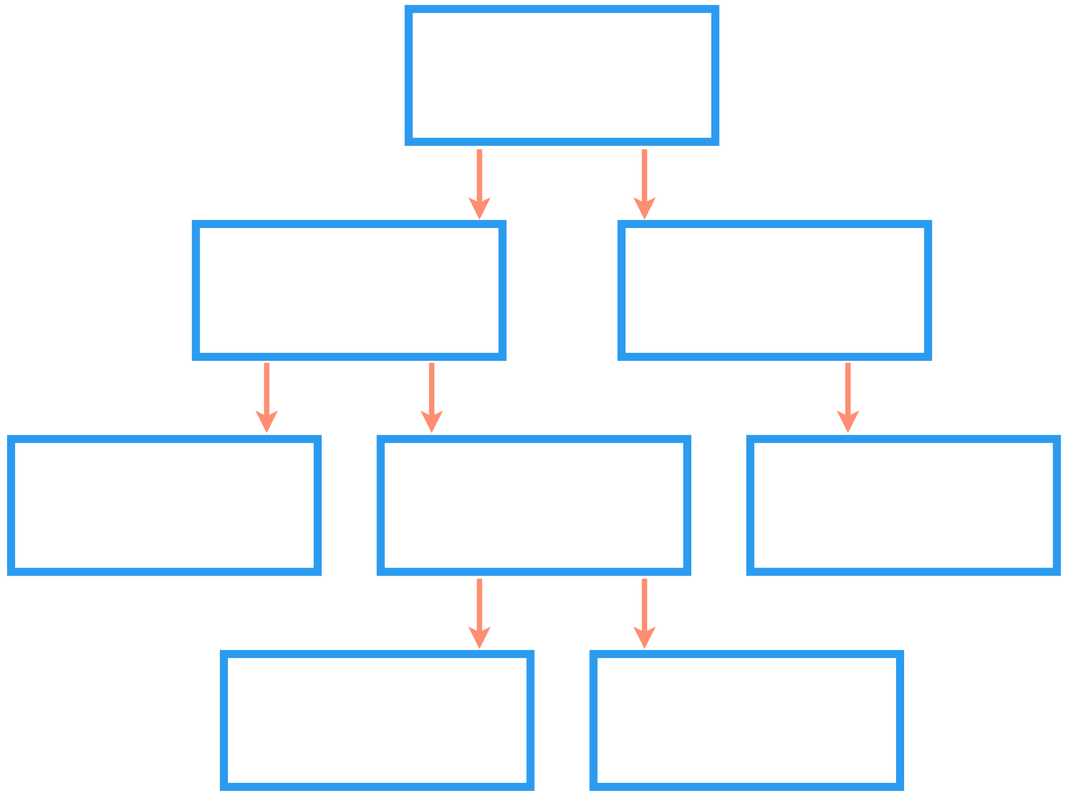
adevnadia


App

state or props change
Context value changed
parent re-renders


React app
re-render

React app



unnecessary!

type here

Same app

Country settings

Toggle theme

- Aruba
- Afghanistan
- Angola
- Anguilla
- Åland Islands
- Albania
- Andorra
- United Arab Emirates
- Argentina
- Armenia
- American Samoa
- Antarctica
- French Southern and Antarctic Lands

- Country: Albania
- Capital: Tirana
- Region: Europe
- Subregion: Southeast Europe

Save

Country settings

Toggle theme

- Aruba
- Afghanistan
- Angola
- Anguilla
- Åland Islands
- Albania
- Andorra
- United Arab Emirates
- Argentina
- Armenia
- American Samoa
- Antarctica
- French Southern and Antarctic Lands

- Country: United Arab Emirates
- Capital: Abu Dhabi
- Region: Asia
- Subregion: Western Asia

Save

common mistakes
performance tricks

Common mistakes

The myth of useMemo and useCallback

[1,2,3] === [1,2,3] → false

```
const Component = () => {
  return <Child value={[1,2,3]} />
}
```

Different!

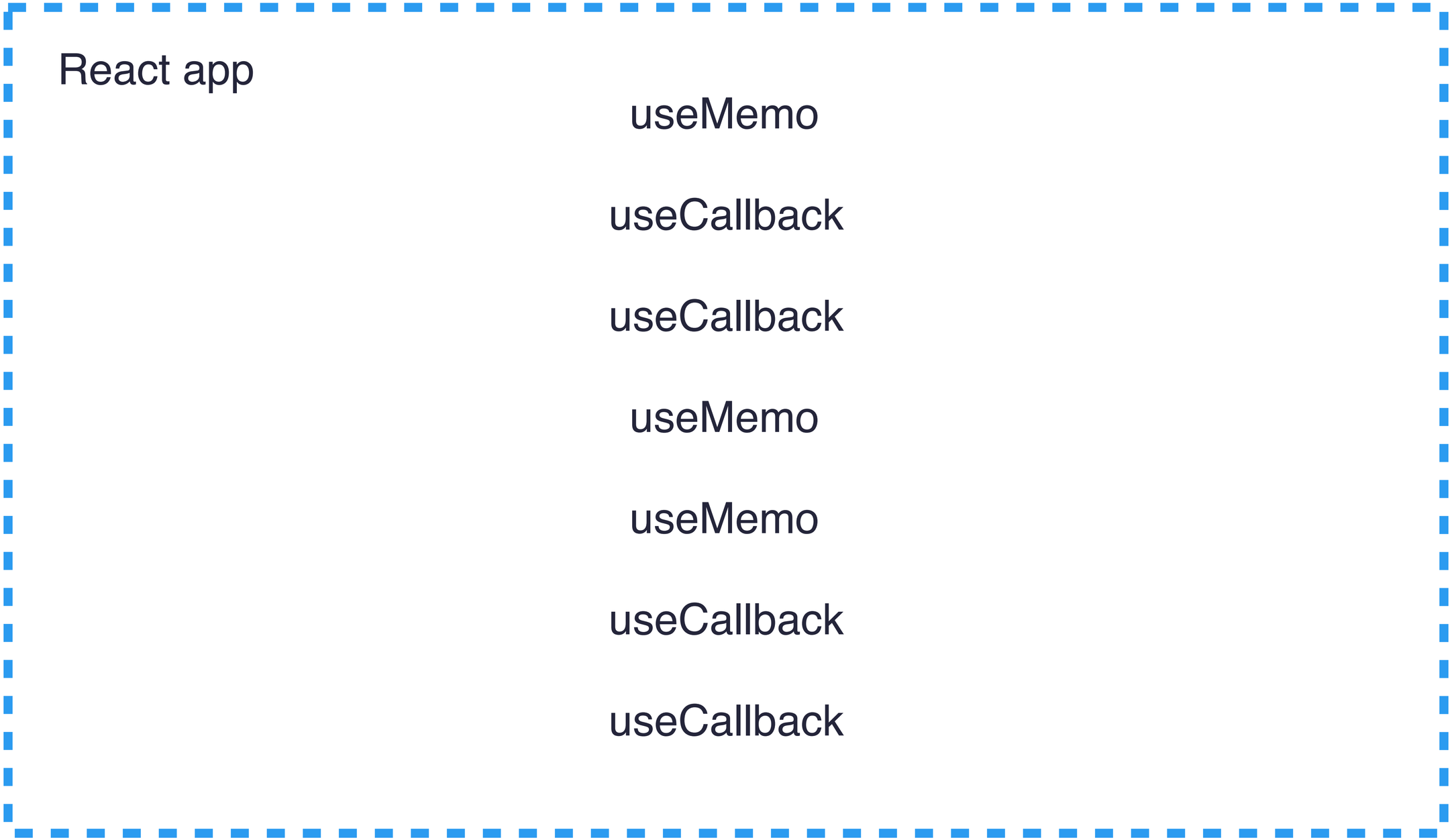
```
const Component = () => {
  return <Child value={[1,2,3]} />
}
```

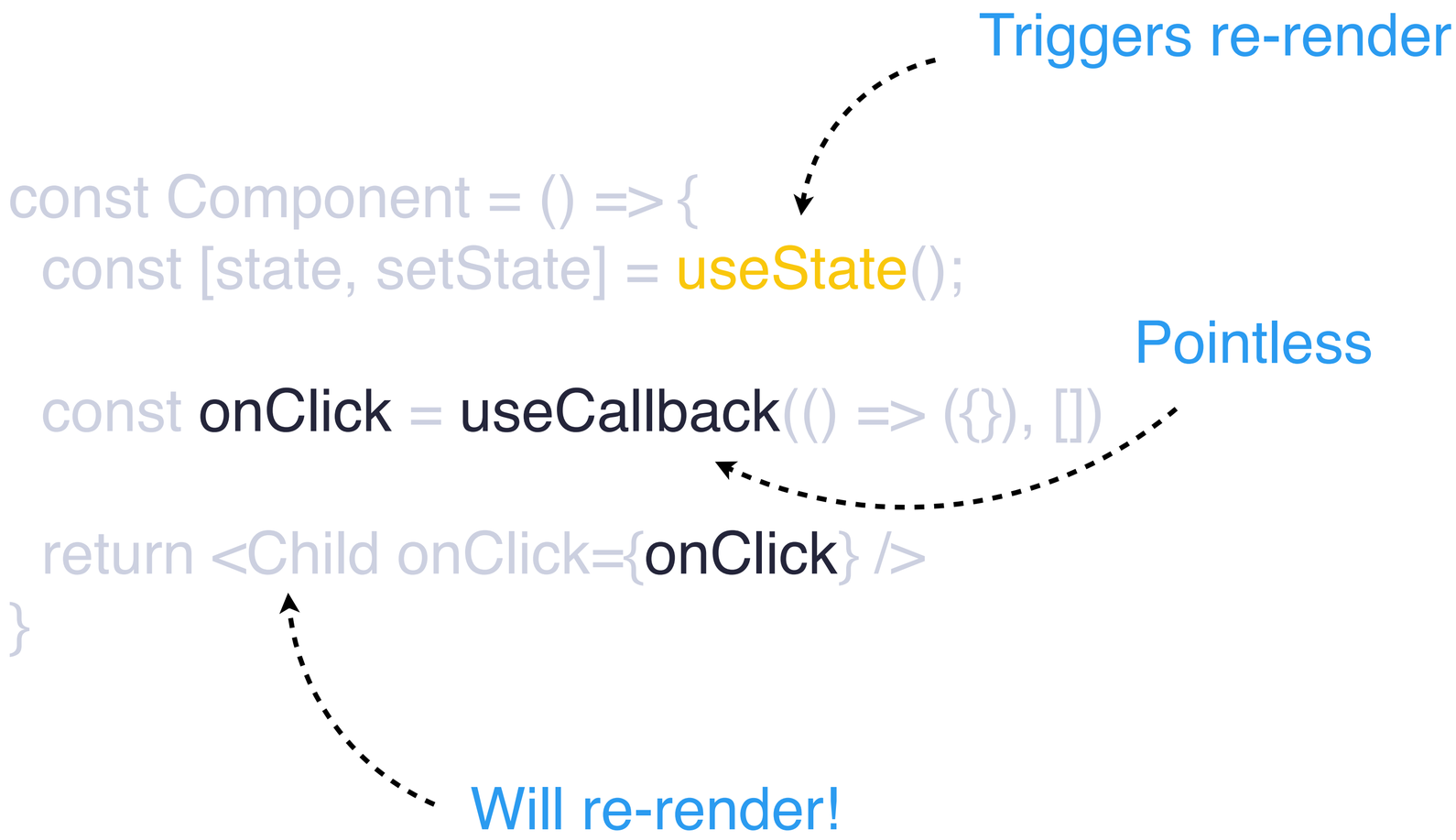
```
const Component = () => {
  const value = useMemo(() => ([1,2,3]), [])
  return <Child value={value} />
}
```

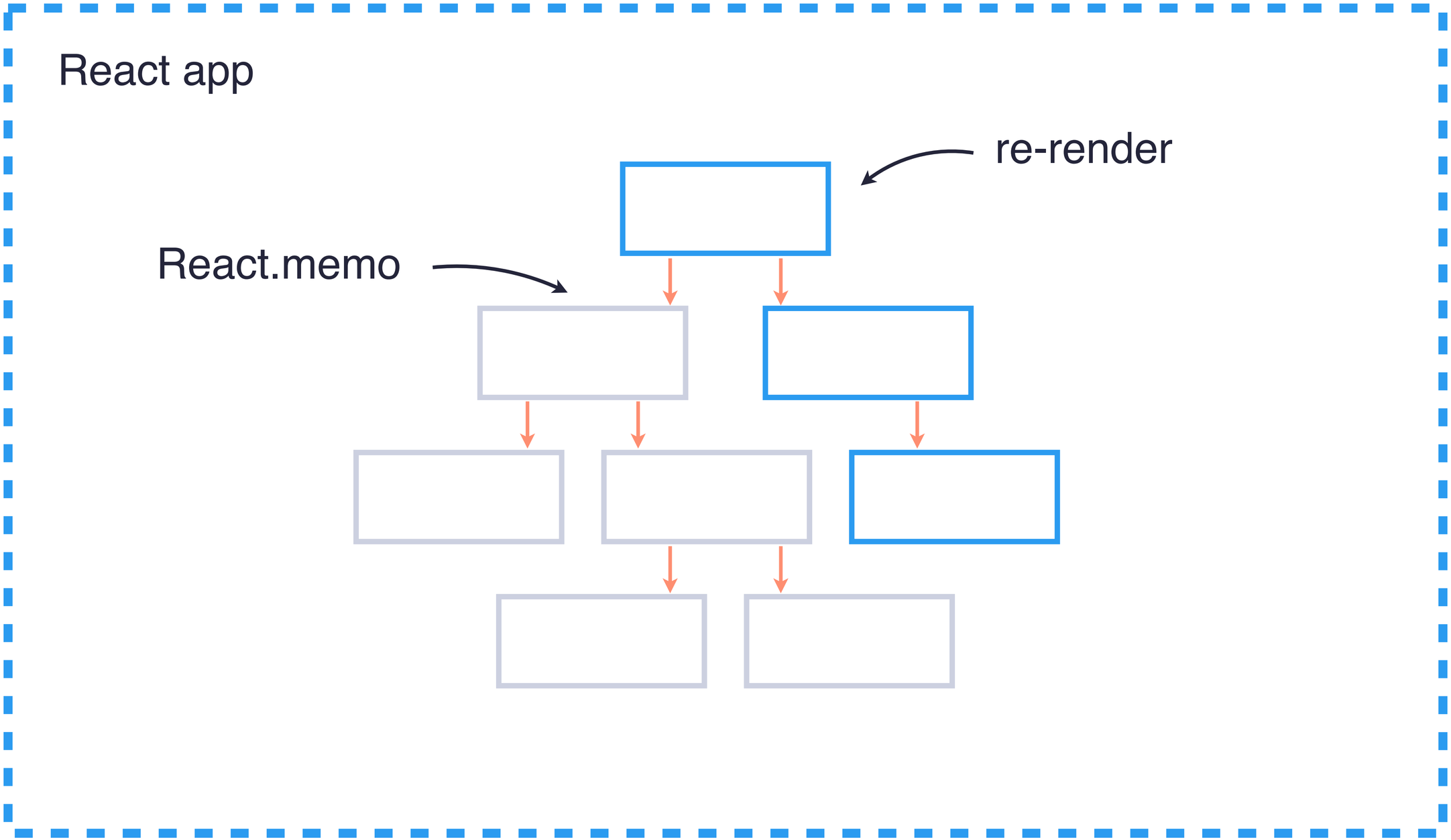
Same!

```
const Component = () => {
  const value = useMemo(() => ([1,2,3]), [])
  return <Child value={value} />
}
```

state or props change


React app
useMemo
useCallback
useCallback
useMemo
useMemo
useCallback
useCallback


Triggers re-render
const Component = () => {
const [state, setState] = useState();
Pointless
const onClick = useCallback(() => ({}), [])
return <Child onClick={onClick} />
}
Will re-render!


React app
re-render
React.memo

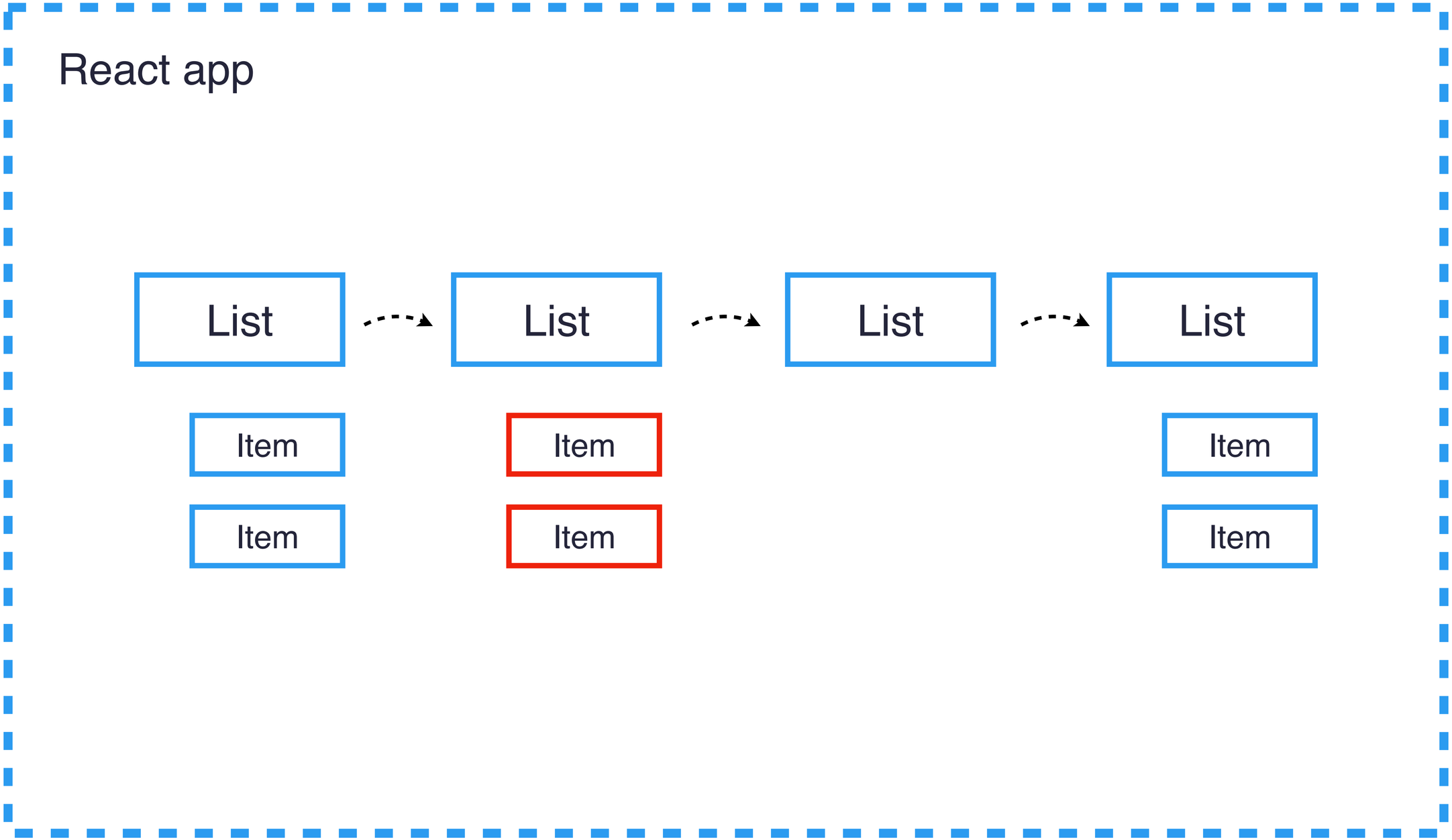
```
const Child = () => ...

const Component = () => {
  const onClick = useCallback(() => ({}), [])
  return <Child onClick={onClick} />
}
```

```
const Child = () => ...

const ChildMemo = React.memo(child);

const Component = () => {
  const onClick = useCallback(() => ({}), [])
  return <ChildMemo onClick={onClick} />
}
```

Creating components in render function

```
const List = ({ countries }) => {
  return (
    <>
      {countries.map((country) => (
        <button>{country.name}</button>
      ))}
    </>
  );
};
```

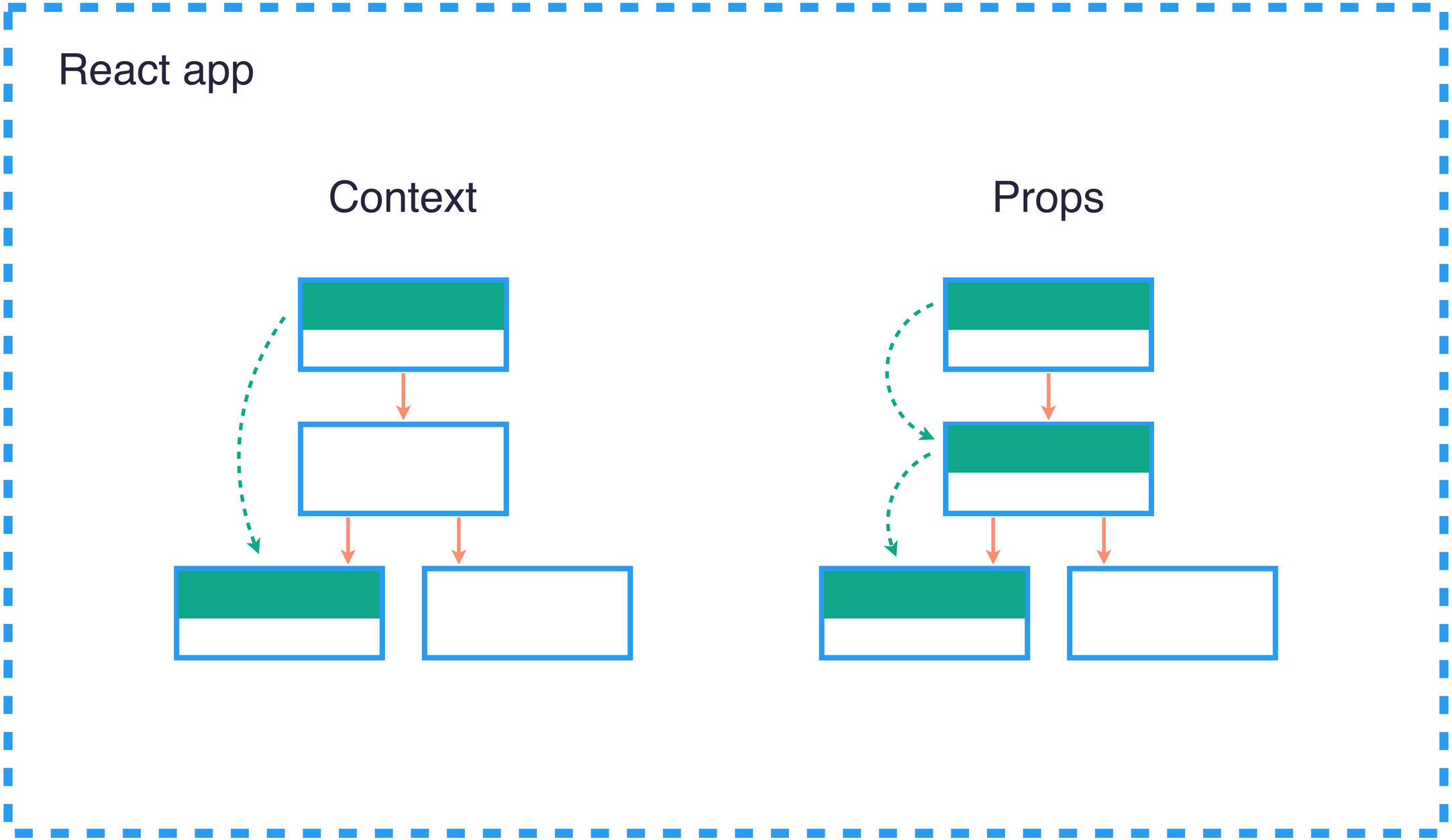
```
const List = ({ countries }) => {

  const Item = ({ country }) => <button>{country.name}</button>;

   return (
    <>
      {countries.map((country) => (
        <Item country={country} />
      ))}
    </>
  );
};
```


React app
List
List
List
List
Item
Item
Item
Item
Item
Item

```
const List = ({ countries }) => {

  const Item = ({ country }) => <button>{country.name}</button>;

  return (
    <>
      {countries.map((country) => (
        <Item country={country} />
      ))}
    </>
  );
};
```

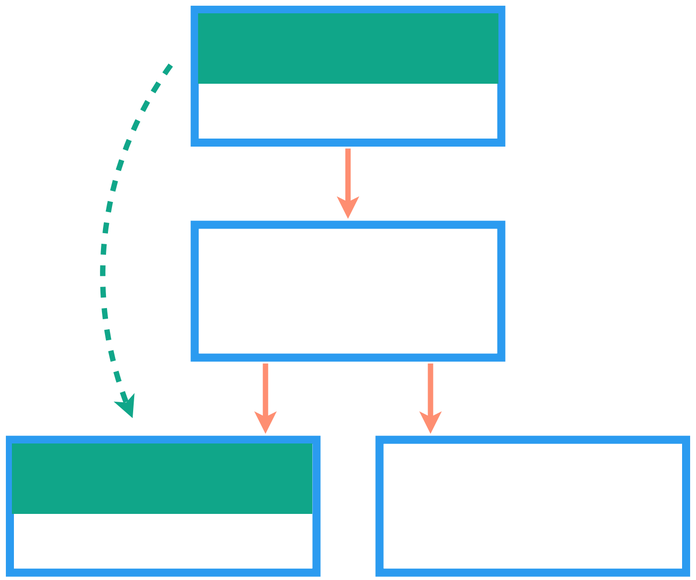
```
const List = ({ countries }) => {

  const Item = ({ country }) => <button>{country.name}</button>;

   return (
    <>
      {countries.map((country) => (
        <Item country={country} />
      ))}
    </>
  );
};
```

Context Provider


React app
Context
Props

React app

Context



```
const Component = () => {

  // something

  return (
    <Context.Provider value={{ theme: 'dark' }}>
      // stuff
    </>
  );
};
```

```
const Component = () => {

  const theme = useMemo(() => ({ theme: 'dark' }))

  return (
    <Context.Provider value={theme}>
      // stuff
    </>
  );
};
```

Performance tricks

Moving state down

```
const Component = () => {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <SomeComponent />
      <Button onClick={() => setIsOpen(true)} />
      {isOpen && <ModalDialog />}
      <MoreComponents />
      <AnotherBigComponent>
        <ThingsHere />
        <MoreThingsHere />
        <AnotherThingsHere />
      </AnotherBigComponent>
    </div>
  );
};
```

```
const Component = () => {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <SomeComponent />
      <Button onClick={() => setIsOpen(true)} />
      {isOpen && <ModalDialog />}
      <MoreComponents />
      <AnotherBigComponent>
        <ThingsHere />
        <MoreThingsHere />
        <AnotherThingsHere />
      </AnotherBigComponent>
    </div>
  );
};
```

updated

re-render

click

```jsx
const Component = () => {
  const [isOpen, setIsOpen] = useState(false);

  return (
    <div>
      <SomeComponent />
      <Button onClick={() => setIsOpen(true)} />
      {isOpen && <ModalDialog />}
      <MoreComponents />
      <AnotherBigComponent>
        <ThingsHere />
        <MoreThingsHere />
        <AnotherThingsHere />
      </AnotherBigComponent>
    </div>
  );
};
```

<OpenModalButton />

```jsx
const OpenModalButton = () => {
  const [isOpen, setIsOpen] = useState(false);
  return (
    <>
      <Button onClick={() => setIsOpen(true)} />
      {isOpen && <ModalDialog />}
    </>
  );
};
```

Wrapping state around children


updated
const Component = () => {
const [scroll, setScroll] = useState(0);
return (
<div onScroll={(e) => setScroll(e.target.scrollTop)}>
<SomeComponent />
<MoreComponents />
<AnotherBigComponent>
<ThingsHere />
<MoreThingsHere />
<AnotherThingsHere />
</AnotherBigComponent>

);
};
scroll
re-render

```
const Component = () => {
  const [scroll, setScroll] = useState(0);

  return (
    <div onScroll={(e) => setScroll(e.target.scrollTop)}>
      <SomeComponent />
      <MoreComponents />
      <AnotherBigComponent>
        <ThingsHere />
        <MoreThingsHere />
        <AnotherThingsHere />
      </AnotherBigComponent>
    </div>
  );
};
```

```
const Component = () => {
  const [scroll, setScroll] = useState(0);

  return (
    <ScrollComponent onScroll={(e) => setScroll(e.target.scrollTop)}>
      <SomeComponent />
      <MoreComponents />
      <AnotherBigComponent>
        <ThingsHere />
        <MoreThingsHere />
        <AnotherThingsHere />
      </AnotherBigComponent>
    </ScrollComponent>
  );
};
```

not affected!

updated

```
const ScrollComponent = ({ children }) => {
  const [scroll, setScroll] = useState(0);

  return (
    <div onScroll={(e) => setScroll(e.target.scrollTop)}>
      {children}
    </div>
  );
};
```

scroll

memoizing part of the render tree

```
const Component = () => {
  const [theme, setTheme] = useState('light');

  return (
    <ScrollComponent theme={theme}>
      <SomeComponent />
      <MoreComponents theme={theme} />
      <AnotherBigComponent>
        <ThingsHere />
        <MoreThingsHere />
        <AnotherThingsHere />
      </AnotherBigComponent>
    </ScrollComponent>
  );
};
```

```
const Component = () => {
  const [theme, setTheme] = useState('light');

  return (
    <ScrollComponent theme={theme}>
      <SomeComponent />
      <MoreComponents theme={theme} />
      <AnotherBigComponent>
        <ThingsHere />
        <MoreThingsHere />
        <AnotherThingsHere />
      </AnotherBigComponent>
    </ScrollComponent>
  );
};
```

```
const Component = () => {
  const [theme, setTheme] = useState('light');

  const memoTree = useMemo(() => (
    <AnotherBigComponent>
      <ThingsHere />
      <MoreThingsHere />
      <AnotherThingsHere />
    </AnotherBigComponent>
  ), [])

  return (
    <ScrollComponent theme={theme}>
      <SomeComponent />
      <MoreComponents theme={theme} />
      {memoTree}
    </ScrollComponent>
  );
};
```

Takeaways

1) remove 80% of useCallback and useMemo

2) don't create components inside render function

3) useMemo for Context value

4) move state down

5) wrap state around children

6) useMemo for part of render tree

https://bit.ly/re-renders-article

https://bit.ly/re-renders-good-code

https://bit.ly/re-renders-bad-code

https://developerway.com

adevnadia

Thank you!