bocacode
your future re/imagined

Redux is dead! Long live HOC! Context is King!

Dr. Todd H. Albert
Founder and Lead Instructor
todd@bocacode.com

Starter project

https://github.com/toddalbert/react-context

create-react-app
+ react-boostrap
+ a few child components

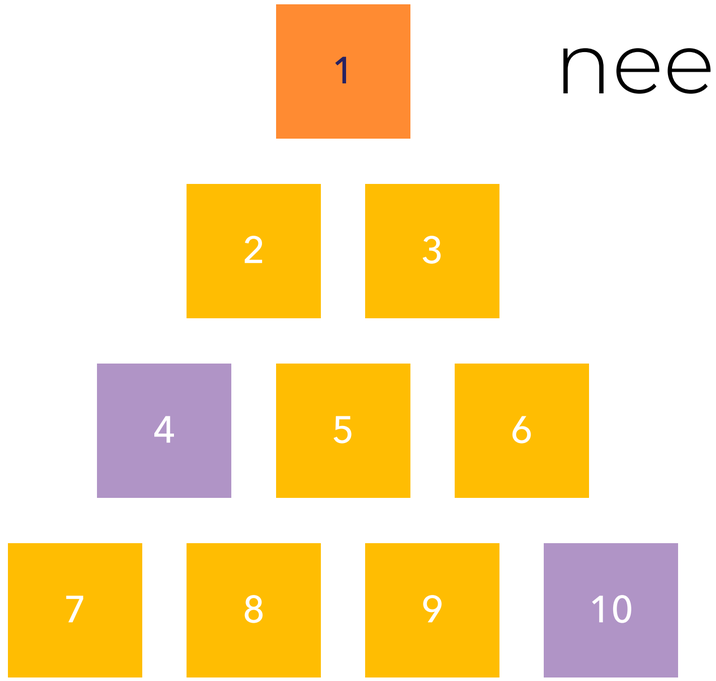
Context

Situation

We have data (state) in component 1, but need it in 4 and 10

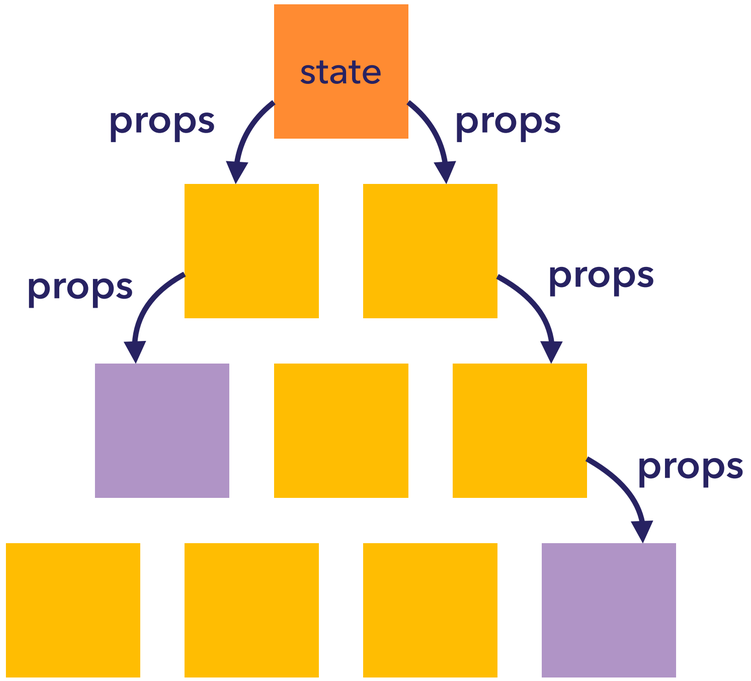
Login/logout in Auth.js

Display user in Welcome.js



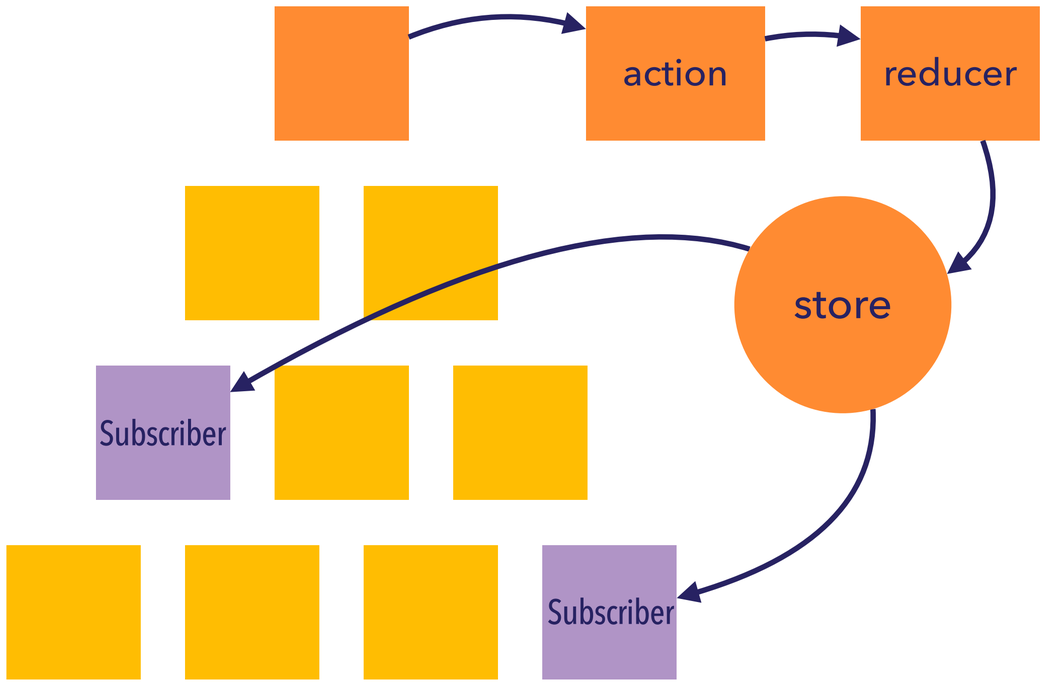
Props

Cascade Using Props



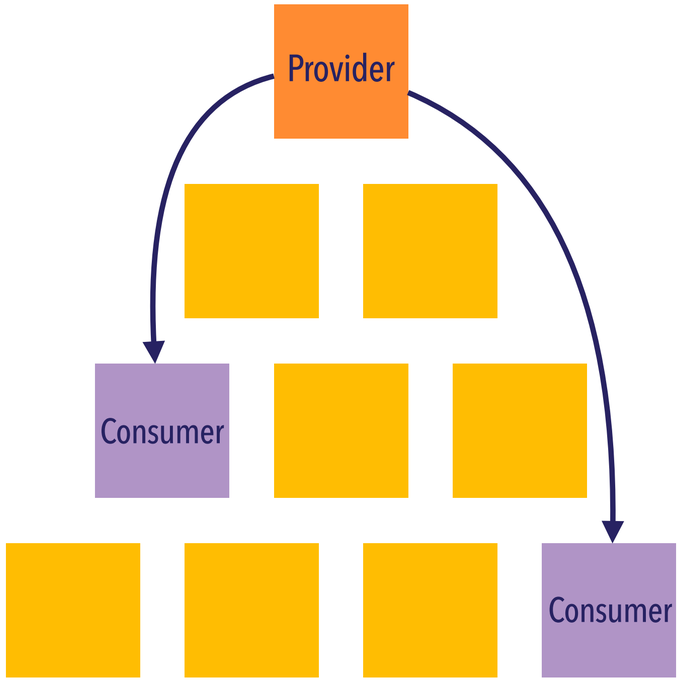
Redux

Using Redux



Context

Using Context



Context

Used to pass data through the component tree without having to pass props down manually at every level

When to use Context

To share **data** considered “**global**” such as the current user, theme, or other preferences

When to use Context

To avoid passing props from component to component to component

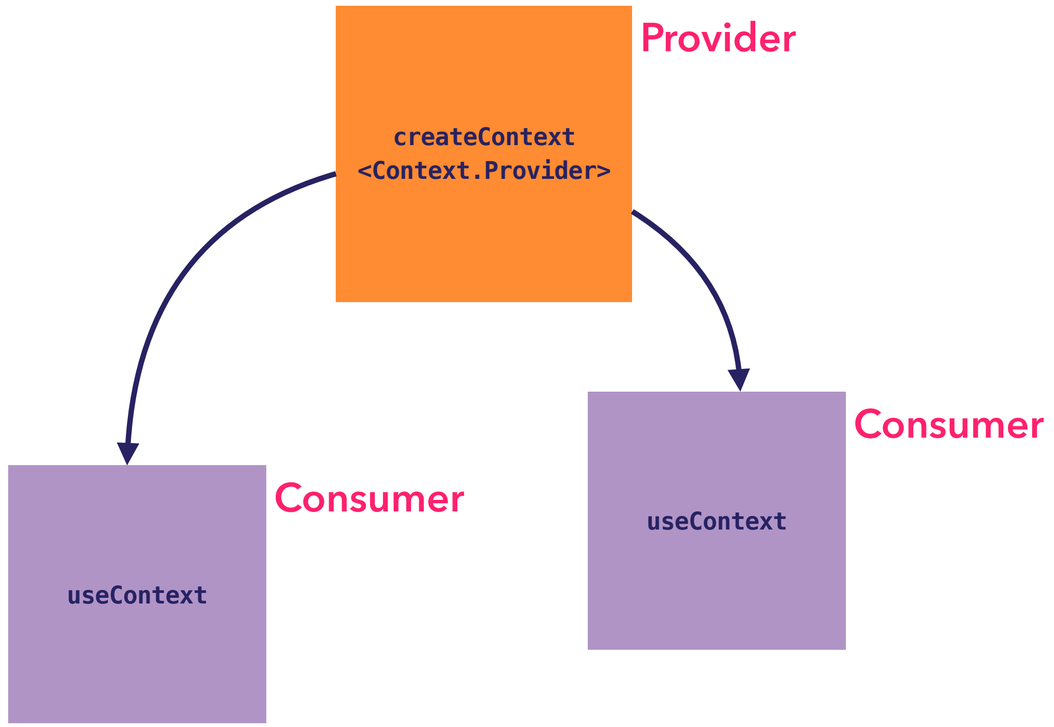
Why not use everywhere?

Only use when needed
Makes component re-use difficult

Setting up Context

createContext
<Context.Provider>

useContext



Higher-Order Components

Higher-Order Component (HOC)

Advanced technique in React for reusing component logic

Higher-Order Component (HOC)

Technically, it is a function that takes a component and returns a new component

Higher-Order Component (HOC)

Transforms a component into another component (with more power)

Higher-Order Component (HOC)

Doesn't modify original component, but wraps it in a container component

createContext

createContext

Creates a context object
If no Provider is found, defaultValue is used (usually null)

```
const MyContext = createContext(defaultValue)
```

Naming convention

Upper-case name of context (e.g. 'User') + 'Context' = 'UserContext'

```
const UserContext = createContext(null)
```

createContext

Automatically defines a Provider component

For our project

Let's create an AuthContext in App.js

```
export const AuthContext = createContext(null)
```

Add some state

```
const [user , setUser] = useState(null)
const [isLoggedIn , setisLoggedIn] = useState(false)
```

<Context.Provider>

Context.Provider

Allows consuming components to subscribe to changes

```
<MyContext.Provider value={/* some value */}>
```

Context.Provider

Takes a value prop and passes it to all consuming components

```
<MyContext.Provider value={/* some value */}>
```

Context.Provider

All consumers will re-render when the Provider value props change

```
<MyContext.Provider value={/* some value */}>
```

For our project

Let's create an AuthContext.Provider in App.js

```
<AuthContext.Provider value={{ user, setUser, isLoggedIn, setIsLoggedIn }}>

...

<AuthContext.Provider/>
```

useContext

useContext

The next most-basic / common hook after useState and useEffect

useContext

Subscribes to a Context.Provider and provides access to Provider prop values

```
const MyValue = useContext(MyContext)
```

useContext

Gets current context value from the prop value of the <MyContext.Provider>

For our project

Let's use AuthContext in Auth.js and Welcome.js

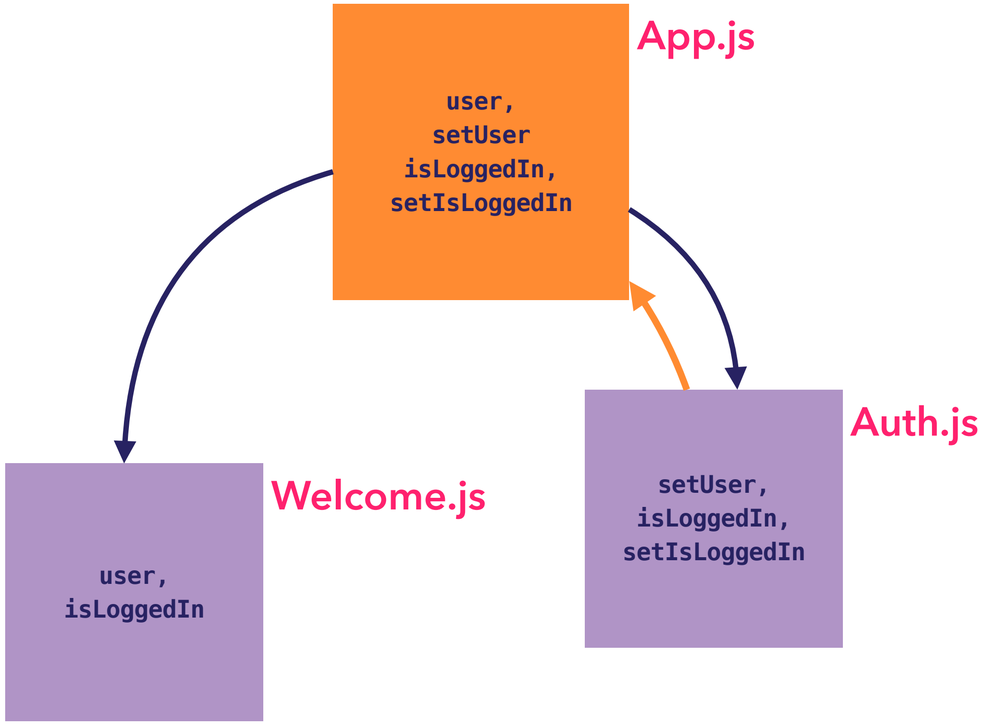
```
import React, { useContext } from 'react'
import AuthContext from '../../App'


let { user, setUser, isLoggedIn, setIsLoggedIn } = useContext(AuthContext)
```

Putting it together

createContext

<Context.Provider>

useContext



Putting it together

Provider Component

```
import React, { createContext } from 'react'

export const AuthContext = createContext(null)

<AuthContext.Provider value={{ user }}>
...
</AuthContext.Provider>
```

Consumer Component

```
import React, { useContext } from 'react'
import { AuthContext } from './App'


const Auth = useContext(AuthContext)

// now we have access to Auth.user
```

Thank you.

bocacode

your future re/imagined

bocacode

your future re/imagined

www.bocacode.com