

# 7.1

Routing, prop drilling, Context API

# Routing

## Jargon

1. **Single page applications**
2. **Client side bundle**
3. **Client side routing**

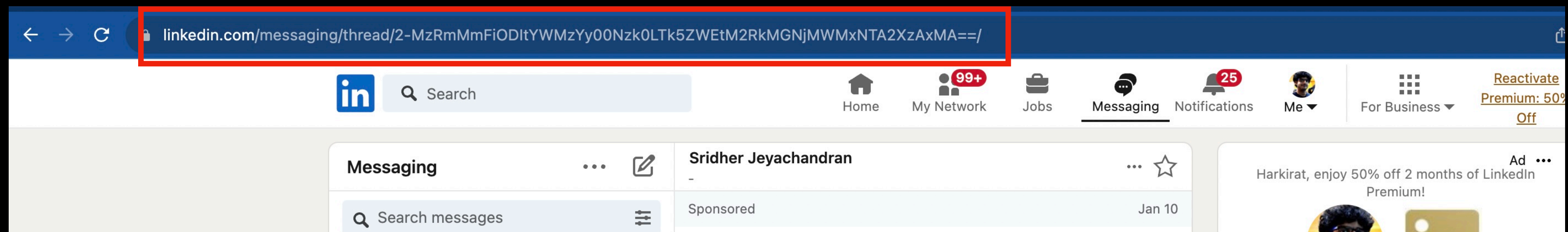
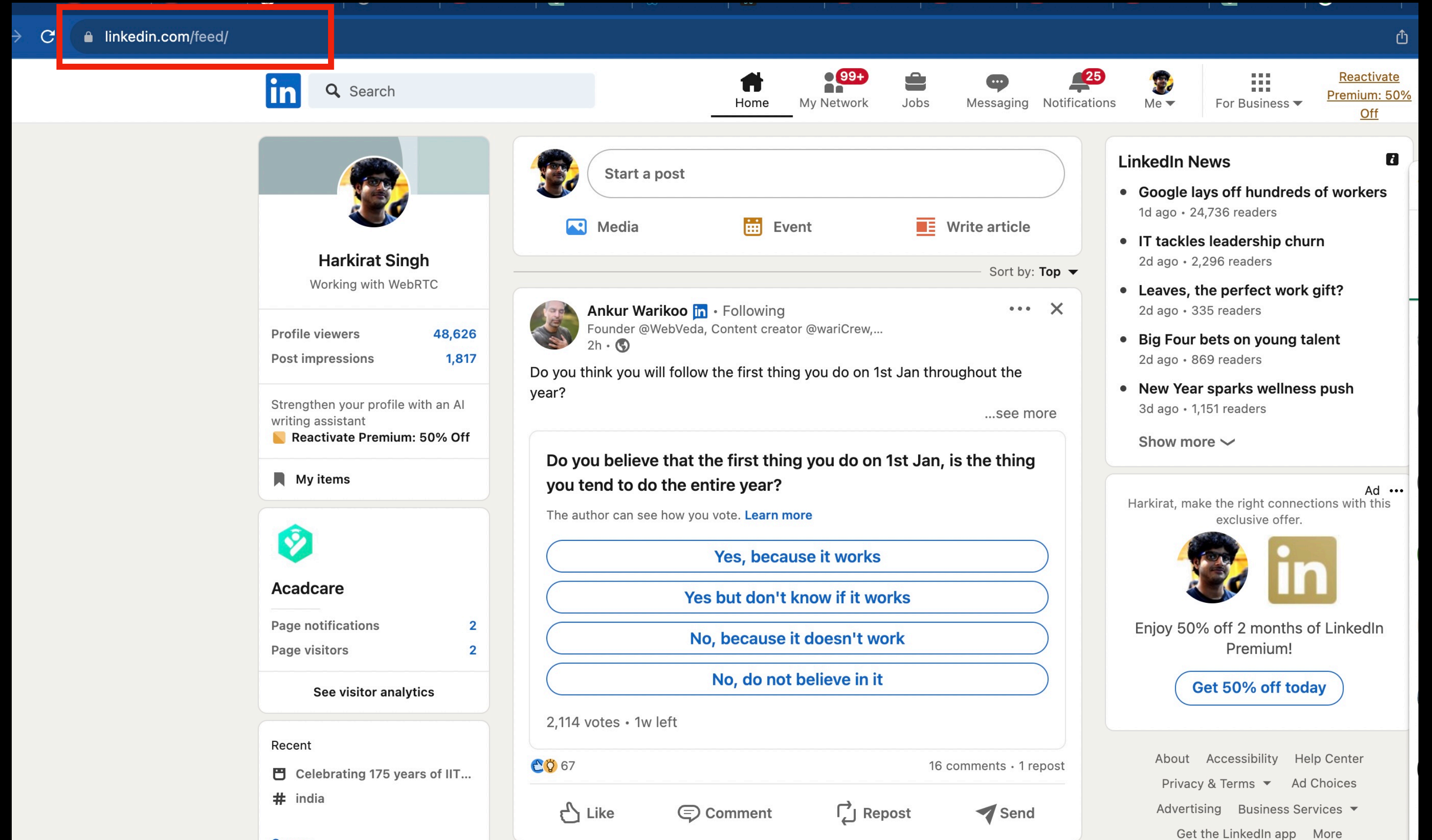
# Routing

## Jargon

1. **Single page applications**
2. **Client side bundle**
3. **Client side routing**

# Routing

What are routes?



# Routing

How to do routing in react?

**react-router-dom**

<https://reactrouter.com>

# Routing

```
function App() {  
  return (  
    <BrowserRouter>  
      <Routes>  
        <Route path="/dashboard" element={<Dashboard />} />  
        <Route path="/" element={<Landing />} />  
      </Routes>  
    </BrowserRouter>  
  )  
}
```

# Routing

## How to navigate?

```
src > components > Dashboard.jsx > Dashboard
1  import { useNavigate } from "react-router-dom"
2
3
4  export default function Dashboard() {
5    const navigate = useNavigate();
6
7    function handleClick() {
8      navigate('/')
9    };
10
11    return <div>
12      Dashboard
13      <button onClick={handleClick}>Click to navigate</button>
14    </div>
15  }
```



# Routing

## Lazy loading

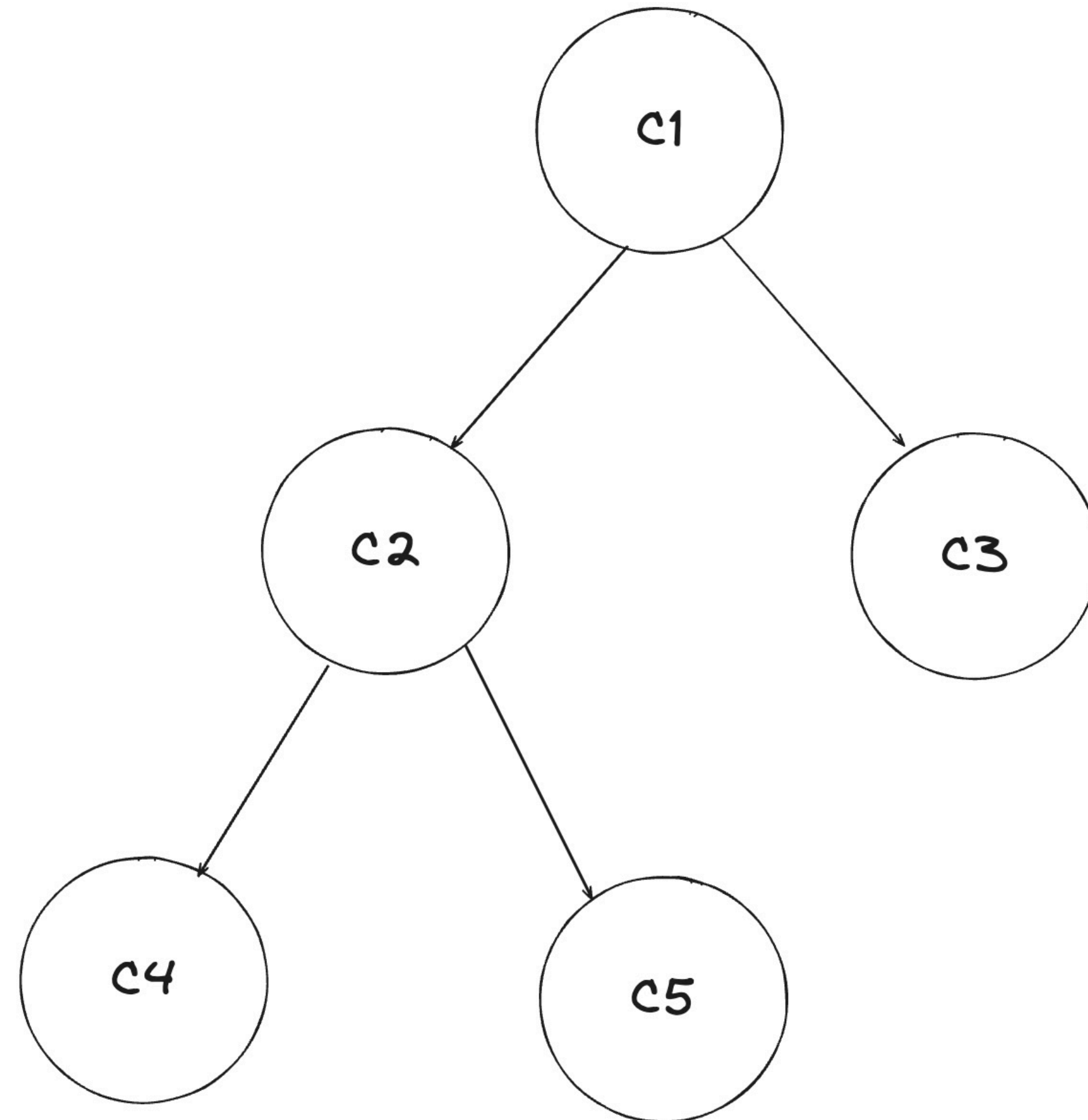
```
1
2 import React, { useCallback, useEffect, useRef, useState } from 'react'
3 import { BrowserRouter, Route, Routes } from "react-router-dom";
4 import { Landing } from './components/Landing';
5 const Dashboard = React.lazy(() => import("./components/Dashboard"));
6
7 function App() {
8
9   return (
10     <BrowserRouter>
11       <Routes>
12         <Route path="/dashboard" element={
13           <Dashboard />
14         } />
15         <Route path="/" element={<Landing />} />
16       </Routes>
17     </BrowserRouter>
18   )
19 }
20
21 export default App
```



# 7.1

Routing, prop drilling, Context API

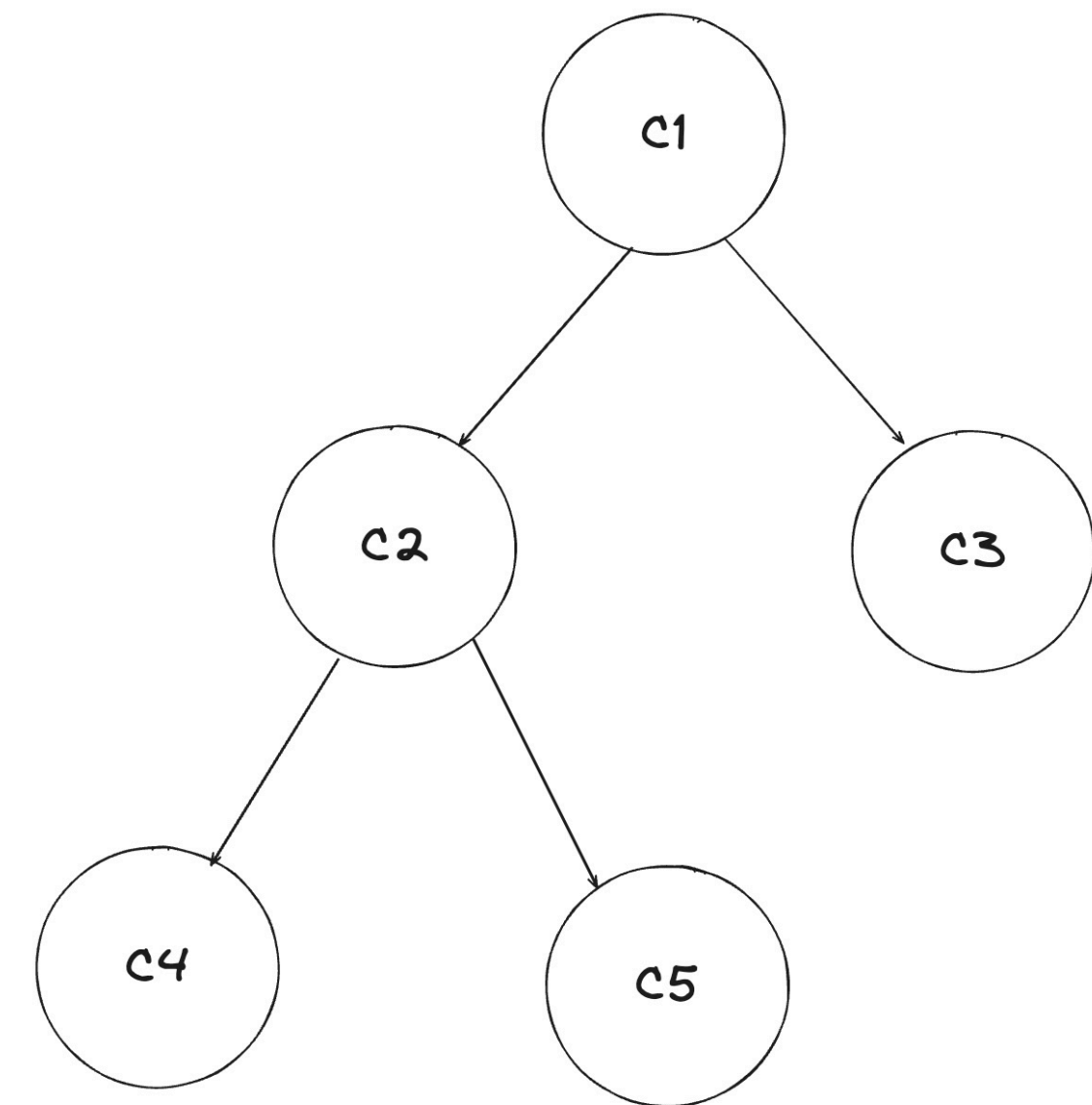
# Prop drilling



# Prop drilling

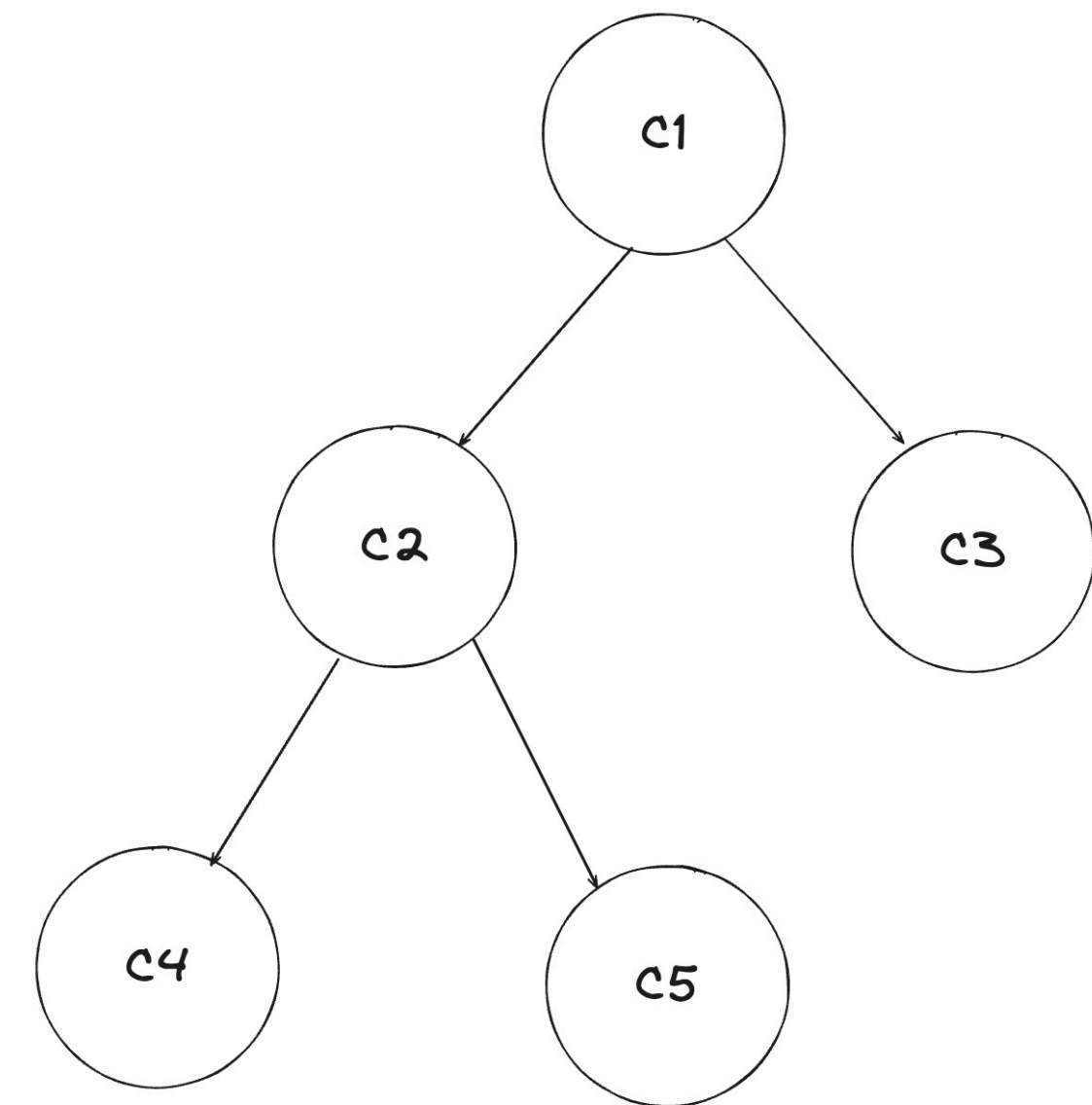
**Before we begin, how do you think one should one manage state?**

- 1. Keep everything in the top level component (C1)**
- 2. Keep everything as low as possible  
(at the LCA of children that need a state)**



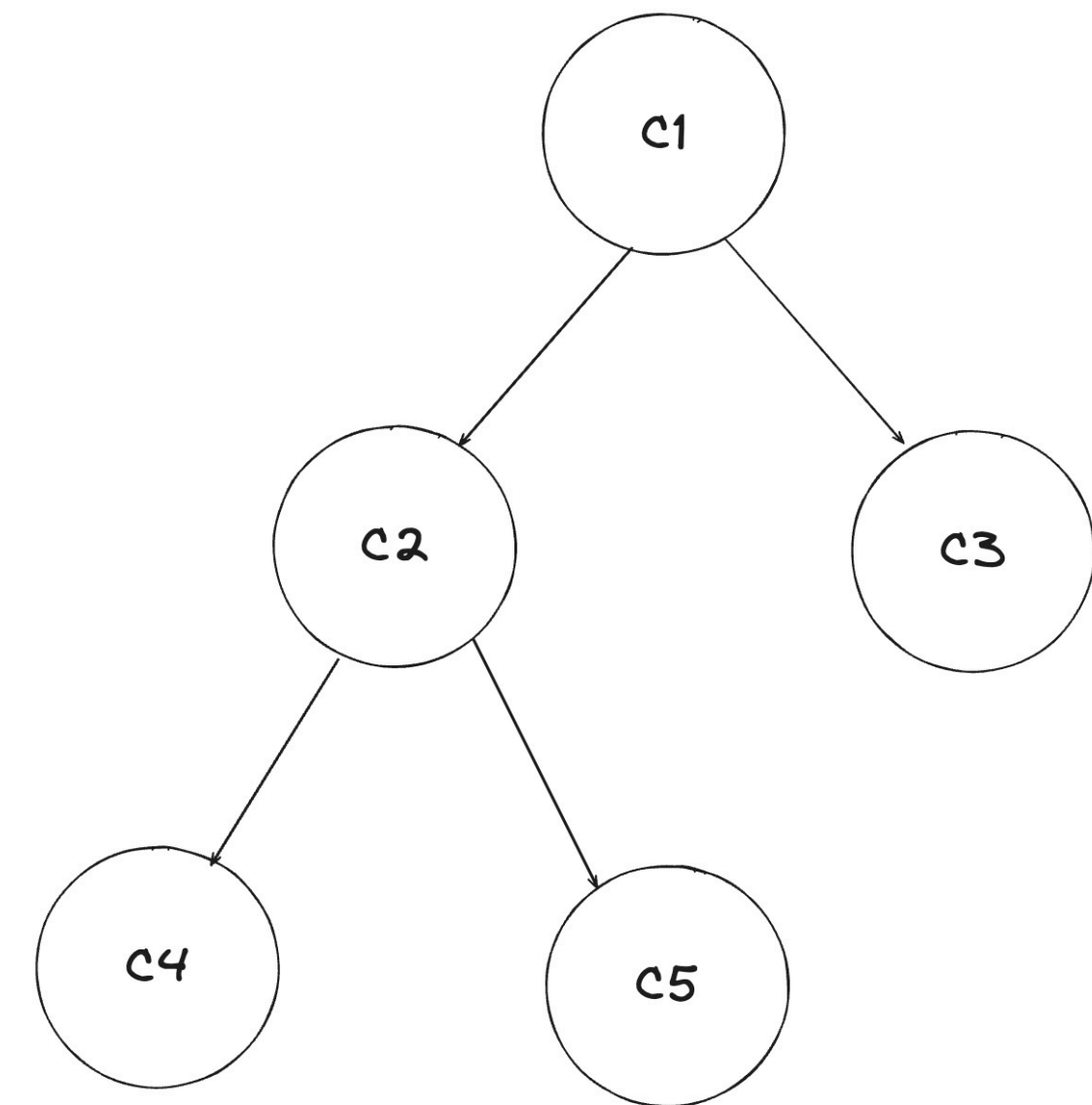
# Prop drilling

Either way, you will need to **drill** props down through the Component tree.  
This gets very hard to maintain and highly verbose  
Makes code highly unreadable



# Prop drilling

**Prop drilling doesn't mean that parent re-renders children  
It just means the syntactic uneasiness when writing code**

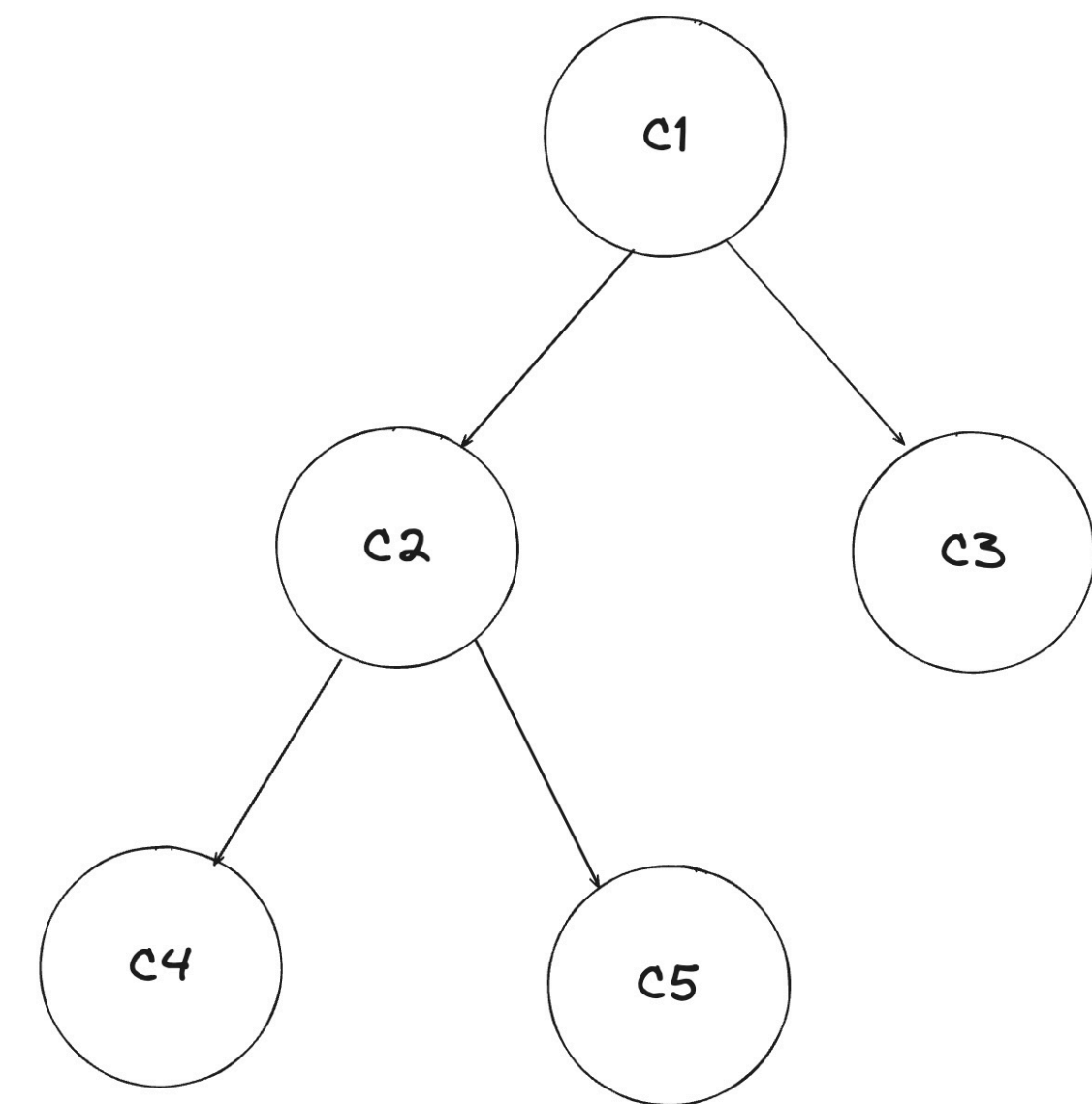


# Prop drilling

## The problem with passing props

Passing props is a great way to explicitly pipe data through your UI tree to the components that use it.

But passing props can become verbose and inconvenient when you need to pass some prop deeply through the tree, or if many components need the same prop. The nearest common ancestor could be far removed from the components that need data, and lifting state up that high can lead to a situation called “prop drilling”.



<https://react.dev/learn/passing-data-deeply-with-context>



# 7.1

Routing, prop drilling, Context API

# Context API

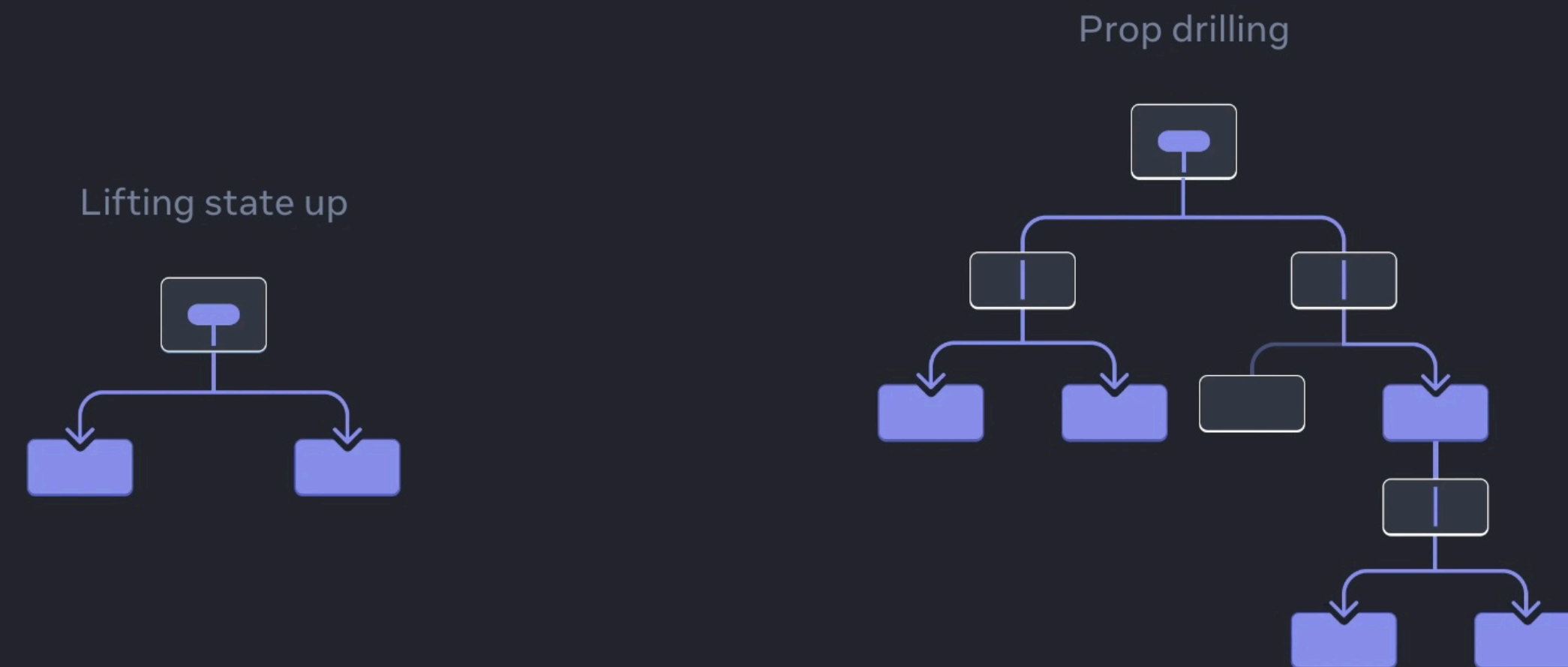
<https://react.dev/learn/passing-data-deeply-with-context>

# Context API

## The problem with passing props

Passing props is a great way to explicitly pipe data through your UI tree to the components that use it.

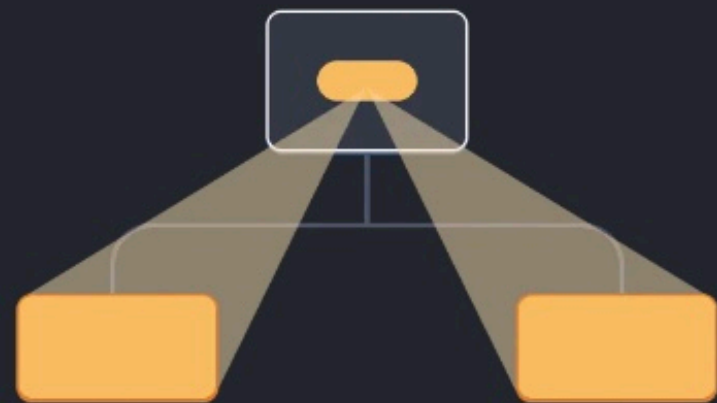
But passing props can become verbose and inconvenient when you need to pass some prop deeply through the tree, or if many components need the same prop. The nearest common ancestor could be far removed from the components that need data, and lifting state up that high can lead to a situation called “prop drilling”.



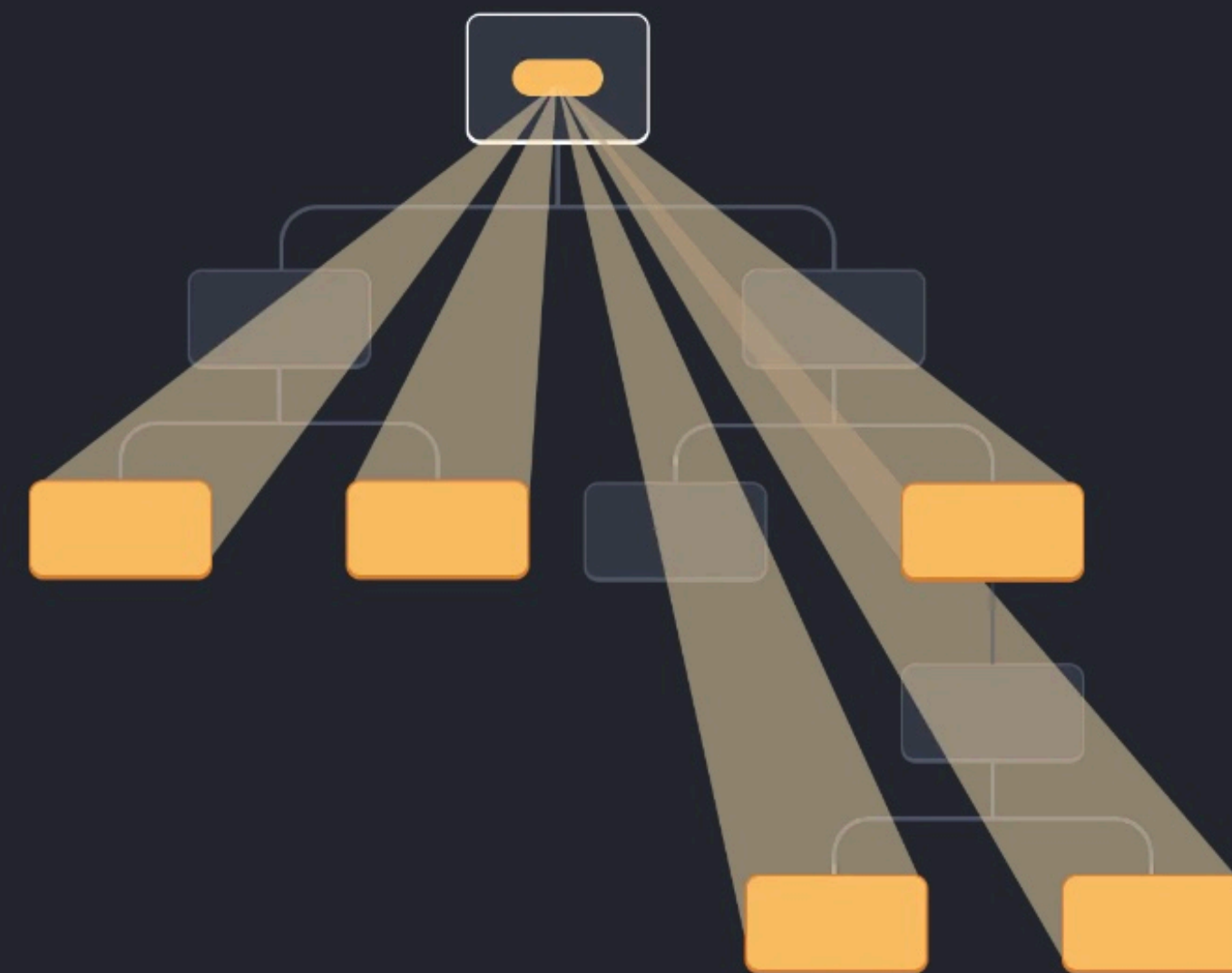
Wouldn't it be great if there were a way to “teleport” data to the components in the tree that need it without passing props? With React's context feature, there is!

# Context API

Using context in close children



Using context in distant children



# Context API

**If you use the context api, you're pushing your state management outside the code react components**

# Context API

**Let's create a simple Counter application, first without the context API and then with it**

**Things to learn -  
createContext  
Provider  
useContext hook**