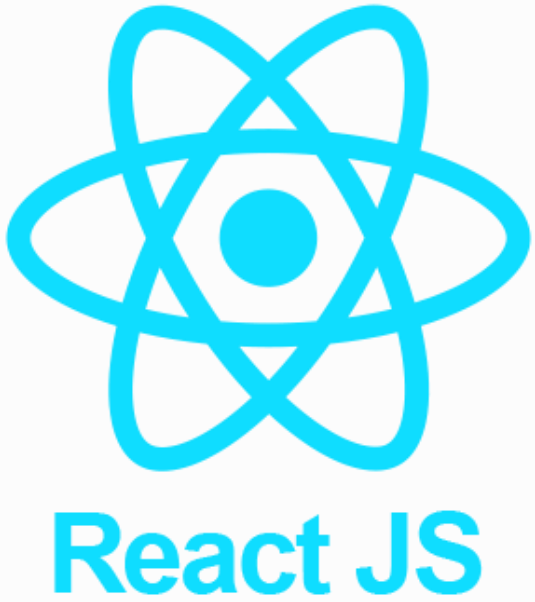
sourcemind

REACT JS

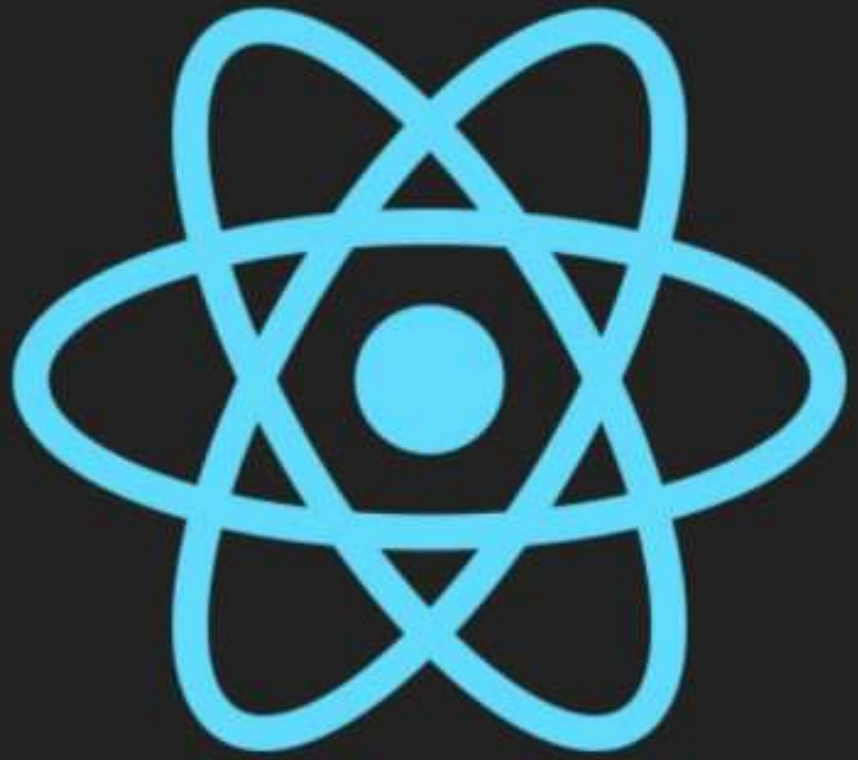
CONTEXT API



REACT HOOKS

GETTING STARTING WITH HOOKS

- useState
- useEffect
- useContext
- useCallback
- useReducer
- useMemo
- useRef
- useLayoutEffect



useReducer

The useReducer Hook is similar to the useState Hook.

It allows for custom state logic.
If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer may be useful.

```
function Todos() {
  const [todos, dispatch] = useReducer(reducer, initialTodos);

  const handleComplete = (todo) => {
    dispatch({ type: "COMPLETE", id: todo.id });
  };

  return (
    <>
      {todos.map((todo) => (
        <div key={todo.id}>
          <label>
            <input
              type="checkbox"
              checked={todo.complete}
              onChange={() => handleComplete(todo)}
            />
            {todo.title}
          </label>
        </div>
      ))}
    </>
  );
}
```

```
const reducer = (state, action) => {
  switch (action.type) {
    case "COMPLETE":
      return state.map((todo) => {
        if (todo.id === action.id) {
          return { ...todo, complete: !todo.complete };
        } else {
          return todo;
        }
      });
    default:
      return state;
  }
};
```

sourcemind

useCallBack

The React useCallback Hook returns a memoized callback function.

useMemo

The React useMemo Hook returns a memoized value.

Custom Hooks

Hooks are reusable functions.

When you have component logic that needs to be used by multiple components, we can extract that logic to a custom Hook.
Custom Hooks start with "use". Example: useFetch

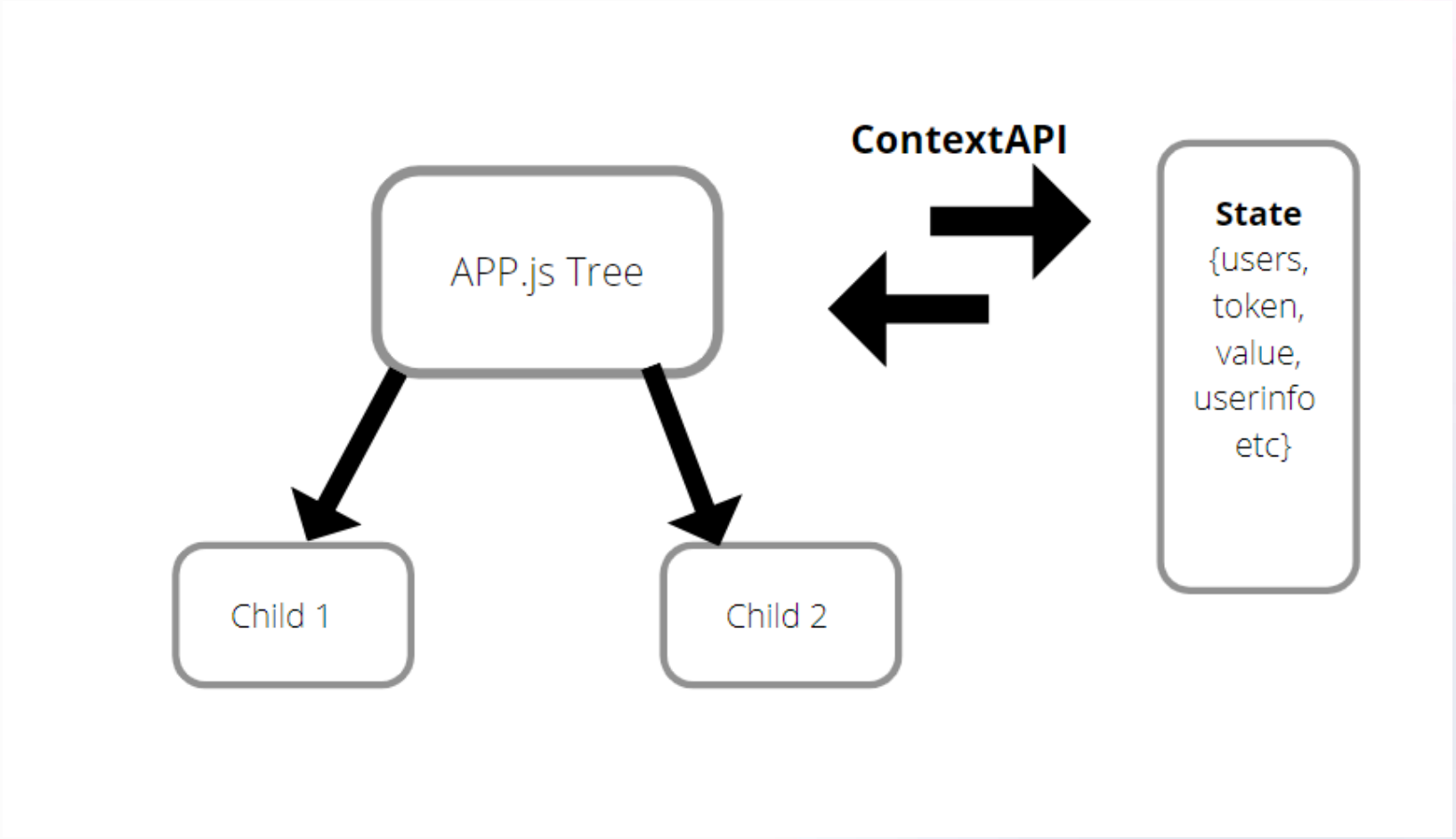
```
import { useState, useEffect } from "react";

const useFetch = (url) => {
  const [data, setData] = useState(null);

  useEffect(() => {
    fetch(url)
      .then((res) => res.json())
      .then((data) => setData(data));
  }, [url]);

  return [data];
};

export default useFetch;
```

```
import ReactDOM from "react-dom/client";
import useFetch from "./useFetch";

const Home = () => {
  const [data] = useFetch("https://jsonplaceholder.typicode.com/todos");

  return (
    <>
      {data &&
        data.map((item) => {
          return <p key={item.id}>{item.title}</p>;
        })}
    </>
  );
};
```

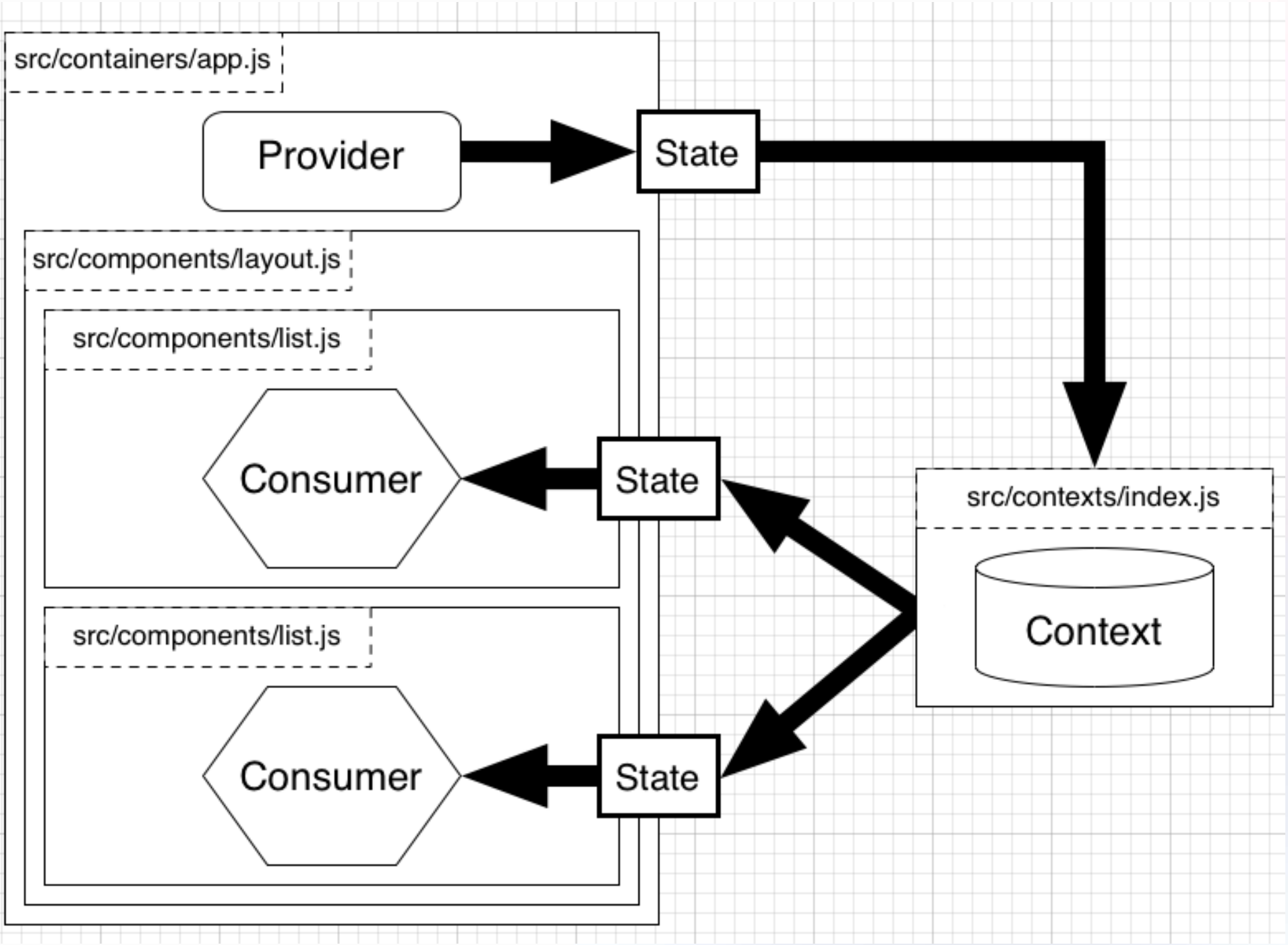
sourcemind

Context API

Context provides a way to pass data through the component tree without having to pass props down manually at every level.



sourcemind


src/containers/app.js
Provider
State
src/components/layout.js
src/components/list.js
Consumer
State
src/contexts/index.js
Context
src/components/list.js
Consumer
State

sourcemind

```
const colors = {
  blue: "#03619c",
  yellow: "#8c8f03",
  red: "#9c0312"
};

export const ColorContext = React.createContext(colors.blue);
```

Copy

```
[label index.js]
import React from 'react';
import { ColorContext } from "./ColorContext";

function App() {
  return (
    <ColorContext.Provider value={colors}>
      <Home />
    </ColorContext.Provider>
  );
}
```

Copy

```
[index.js]
return (
  <ColorContext.Consumer>
    {colors => <div style={colors.blue}>Hello World</div>}
  </ColorContext.Consumer>
);
```

```jsx
import React, { useContext } from "react";
import ColorContext from './ColorContext';

const MyComponent = () => {
  const colors = useContext(ColorContext);

  return <div style={{ backgroundColor: colors.blue }}>Hello World</div>;
};
```