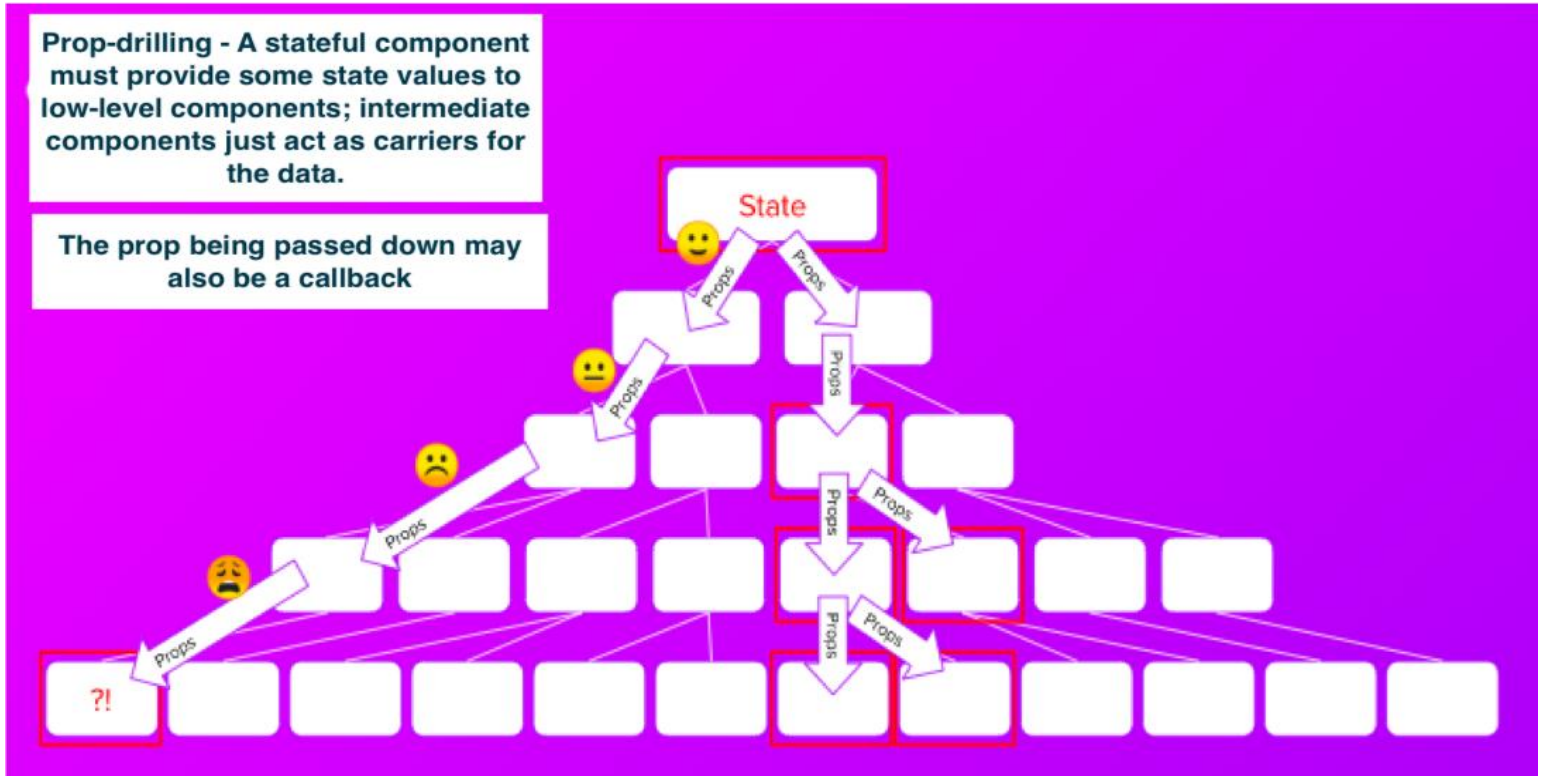
Design Patterns

(Contd.).

The Provider pattern - React Context

The Provider pattern – When?

- **Use cases:**
  1. Sharing data/state **with multiple components**, i.e. global data**, e.g. favourite movies.**
  2. **To avoid** prop-drilling**.**



The Provider pattern – How?

- **React Implementation steps:**
  1. **Create a Context object using the createContext function from React library**

```
export const SomeContext = createContext<ContextInterface | null>(null);
```

  2. **Implement a Provider Component that uses the Provider component from the context object.**

```
const ContextProvider: React.FC<React.PropsWithChildren> = (props) => {
  ...
  return (
    <SomeContext.Provider value={{
      key: value
      // Pass any methods you need as well
    }}>
      {props.children}
    </SomeContext.Provider>
  );
}
```

4.Consume the Context Data: Any component wrapped by MyProvider can now access the context data. You can use the useContext hook in functional components or the Context.Consumer component in class components.

The Provider pattern – How?

3. **Wrap Your Components with the Provider. Use the provider component to wrap any components that need access to the context data.**

```
const App = () => {
  return (
    <BrowserRouter>
      <SomeProvider>
        <Header />
        <Routes>
          ...
        </Routes>
      </SomeProvider>
    </BrowserRouter>
  );
};
```

4. **Use the Context Data: Any component wrapped by <SomeProvider> can now access the context data using the useContext hook.**

```
const context = useContext(SomeContext) || {};

console.log(context.key);
```

The Provider pattern – Implementation

- **Create the context object and Declare the Provider component:**

```
export const SomeContext = createContext<SomeInterface | null>(null);

const ContextProvider: React.FC<React.PropsWithChildren> = (props) => {
 ... Use useState and useEffect hooks to initialise global state variables
return (
  <SomeContext.Provider
    value={{
      key: value1,
      ...
      }}
  >
    {props.children}
  </SomeContext.Provider>
);
};
export default ContextProvider;
```

- **We link the** Context Object **to the** Provider component **using** <contextName.Provider>.
- **The** values object **declares the context's content. .**
  - **Can be functions (behaviour) as well as data (state).**

The Provider pattern – Implementation.

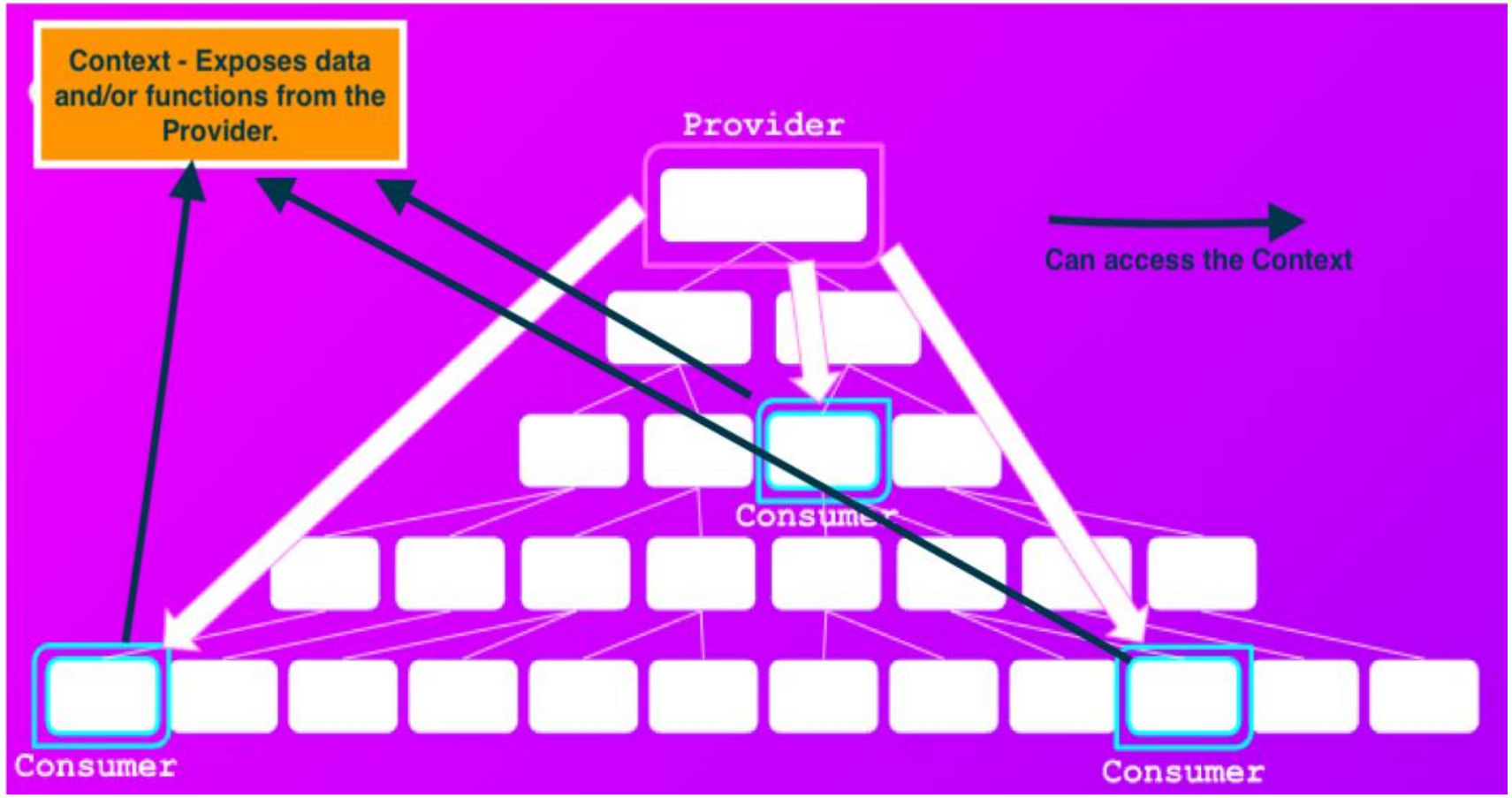
- **Integrate (Compose) the Provider with the rest of the app, using the Container pattern.**

```
const App = () => {
    return (
        <BrowserRouter>
          <ContextProvider>
              . . . . . .
          </ContextProvider>
        </BrowserRouter>
    );
  };

  ReactDOM.render(<App />, document.getElementById("root"));
```

- **All the app's pages can now access the context.**

The Provider pattern – React Context.



The Provider pattern – Implementation.

- useContext **hook – gives a component access to a context:**

  const contextRef = useContext(ContextName)

  // contextRef points at context's values object.

```
import React, { useContext } from "react";
import {SomeContext} from '.....'

const ConsumerComponent = props => {
  const context = useContext(SomeContext);

  . . . access context values with 'context.keyX'

};
```

```
1   export const SomeContext = createContext<SomeInterface | null>(null);
2
3   const ContextProvider: React.FC<React.PropsWithChildren> = (props) => {
4    ... Use useState and useEffect hooks to initialise global state variab
5   return (
6     <SomeContext.Provider
7       value={{
8         key: value1,
9         ...
10        }}
11    >
12      {props.children}
13    </SomeContext.Provider>
14  );
15  };
16  export default ContextProvider;
```

The Provider pattern – Implementation.

- **For improved** separation of concerns**, use multiple context instead of a 'catch all' context.**

```
const App = () => {
    return (
        <BrowserRouter>
          <ContextProviderA>
            <ContextProviderB>
              . . . . . .
            </ContextProviderB>
          </ContextProviderA>
        </BrowserRouter>
    );
  };
```

The Provider pattern.

- **When NOT to use a Context:**
  1. **To avoid ‘shallow’ prop drilling.**
     - **Prop drilling is faster for ‘shallow’ cases.**
  2. **For state that should be kept local to a component, e.g. web form inputs.**
  3. **For large object - monitor performance and refactor as necessary.**