

Design Patterns

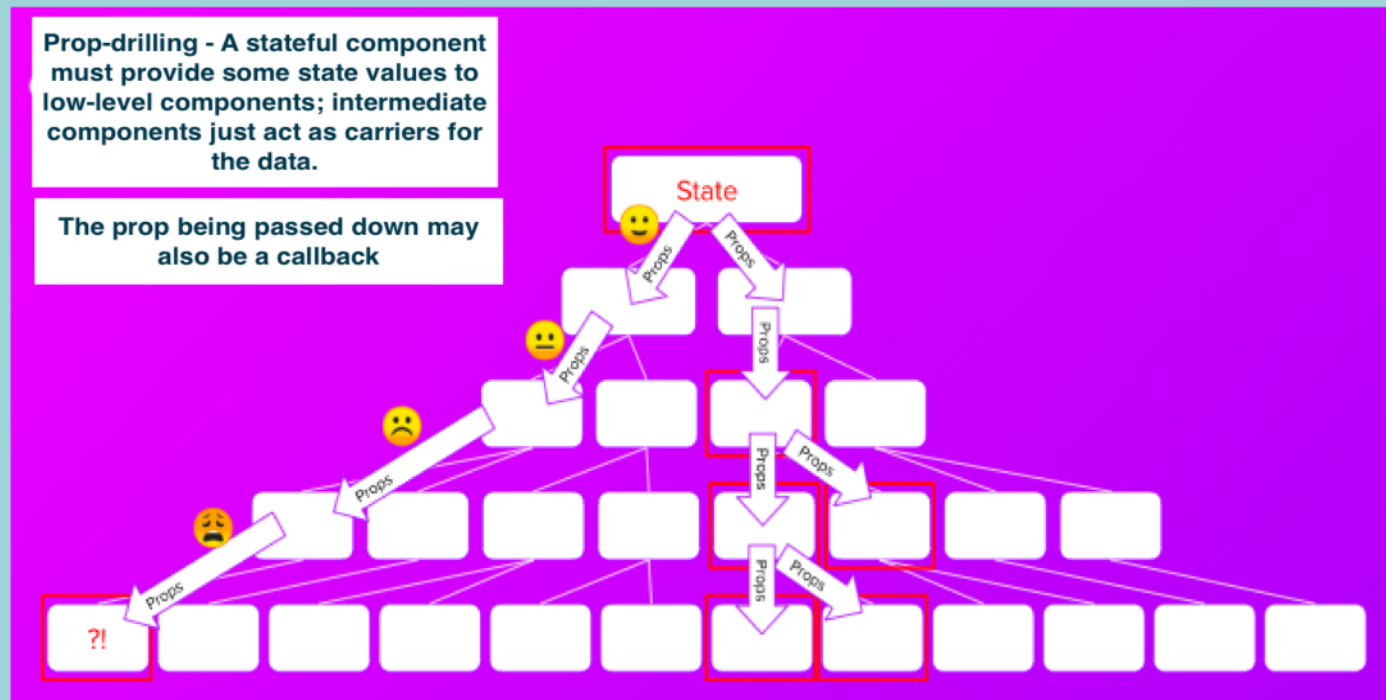
(Contd.)

The Provider pattern

React Contexts

The Provider pattern – When?

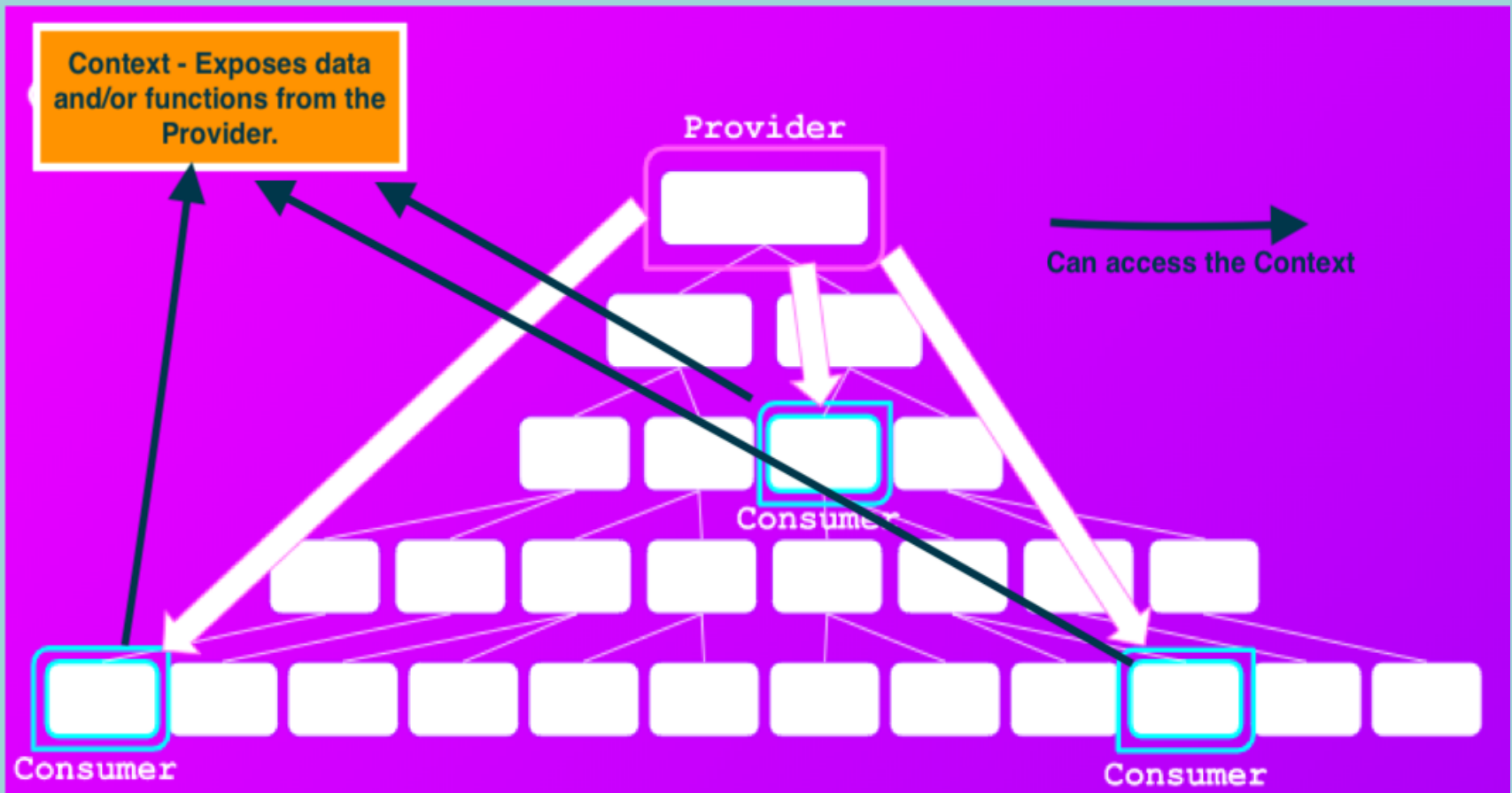
- **Use cases:**
 1. Sharing data/state **with multiple components**, i.e. global data, **e.g. favourite movies.**
 2. **To avoid prop-drilling.**



The Provider pattern – How?

- **React Implementation steps:**
 1. **Declare a component for managing the data to be shared – the Provider component.**
 2. **Create a context construct and associate it with the Provider.**
 3. **Place the data to be shared inside the context.**
 4. **Use composition to integrate Provider and consumer(s).**
 5. **Use a hook to enable consumers access the context's data.**
- **Contexts – the glue for the provider pattern in React.**
 - **Enables a component to share its data (and behaviour) with subordinates, without the need for prop drilling.**
 - **Provider component manages the context.**
 - **Consumer accesses the context with useContext hook**

The Provider pattern – React Contexts.



The Provider pattern – Implementation

- **Declare the Provider component:**

```
0 export const SomeContext = React.createContext(null)
1
2 const ContextProvider = props => {
3   . . . Use useState and useEffect hooks to
4   . . . initialize global state variables
5   return (
6     <SomeContext.Provider
7       value={{ key1: value1, . . . }} >
8     {props.children}
9   </SomeContext.Provider>
10 );
11 };
12 export default ContextProvider
```

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

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"));
```

- The Provider's children (and their children) will now be able to access the context.

The Provider pattern – Implementation.

- **The context consumer uses the useContext hook.**
contextRef = useContext(ContextName)
// contextRef points at context's values object.
 - Use contextRef.key **to access an element inside the context.**

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

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

  . . . access context values with 'context.keyX'
};
```

The Provider pattern – Implementation.

- **For better separation of concerns, have multiple context instead of a 'catch all' context.**

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


The Provider pattern.

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

The End