Design Patterns

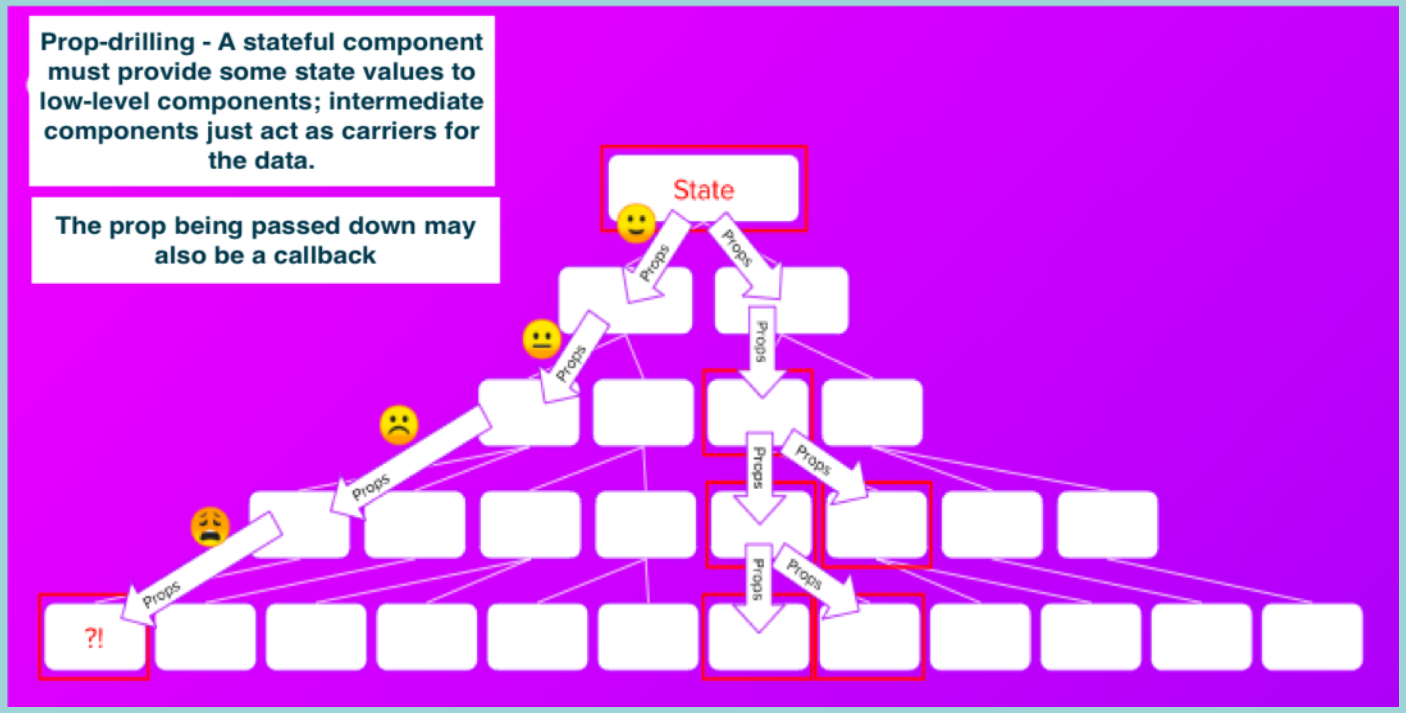
(Contd.)

The Provider pattern

React Contexts

The Provider pattern – When?

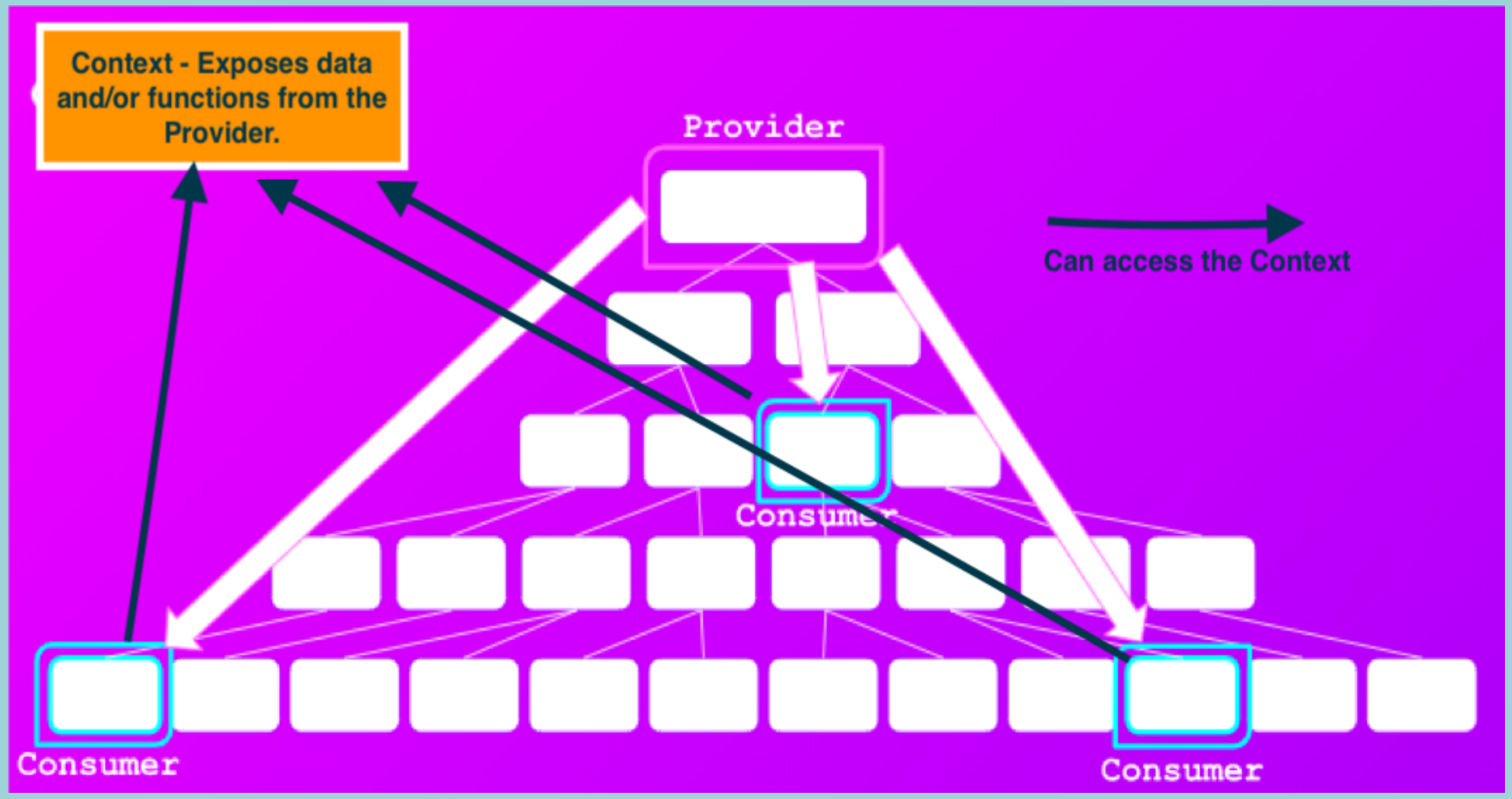
- **Use cases:**
  1. Sharing data/state **with multiple components**, i.e. global data**, e.g. favourite movies.**
  2. **To avoid** prop-drilling**.**



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:**

```
export const SomeContext = React.createContext(null)

const ContextProvider = props => {
  . . . Use useState and useEffect hooks to
  . . . initialize global state variables
  return (
    <SomeContext.Provider
         value={{ key1: value1,. . . . }} >
      {props.children}
    </SomeContext.Provider>
  );
};
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'

};
```

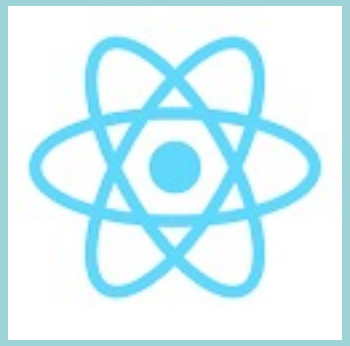
The Provider pattern – Implementation.

- **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.**



Custom Hooks

Custom Hooks.

- **Custom Hooks let you extract component logic into** reusable **functions.**
- **Improves code readability and modularity.**

Example:

```
const BookPage = props => {
  const isbm = props.isbn;
  const [book, setBook] = useState(null);
  useEffect(() => {
    fetch(
    `https://api.for.books?isbn=${isbn}`)
    .then(res => res.json())
    .then(book => {
      setBook(book);
    });
  }, [isbn]);
  . . . .rest of component code . . . .
}
```

Objective – Extract the book-related state code into a custom hook.

Custom Hook Example.

Solution:

```
const useBook = isbn => {
  const [book, setBook] = useState(null);
  useEffect(() => {
     fetch(
    `https://api.for.books?isbn=${isbn}`)
    .then(res => res.json())
    .then(book => {
      setBook(book);
    });
  }, [isbn]);
   return [book, setBook];
};
```

```
const BookPage = props => {
  const isbm = props.isbn;
  const [book, setBook] = useBook(isbn);

  . . . .rest of component code . . . .
}
```

- **Custom Hook is an ordinary function BUT should only be called from a React component function.**
- **Prefix hook function name with** use **to leverage linting support.**
- **Function can return any collection type (array, object), with any number of entries.**

The End