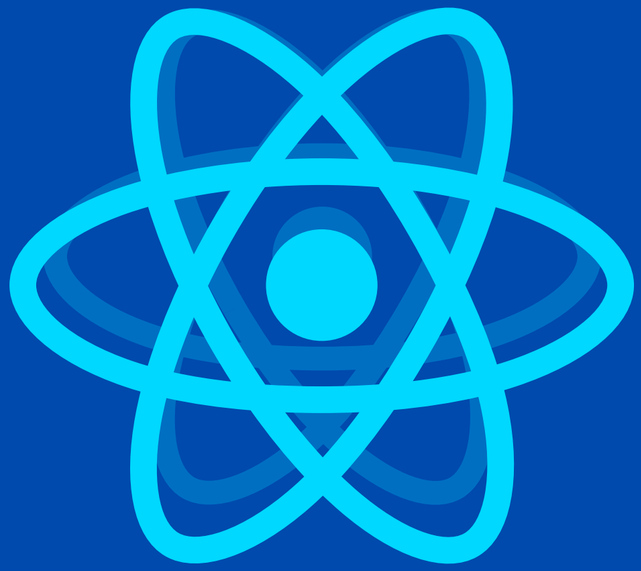


React JS

WHAT IS REACT ?

React is a popular JavaScript library used for building dynamic user interfaces. Developed by Facebook, React has become one of the most widely used libraries for front-end development due to its flexibility, efficiency, and scalability.

What is a Library ?

In programming, a library refers to a collection of pre-written code

Using a library can save time and effort as it eliminates the need to write code from scratch to perform a specific task.

Module: A module is like a single file or a group of files that contain related code. It helps organize and separate different functionalities in your application, making it easier to manage and reuse code.

Package: A package is a collection of modules or files that work together. It's like a bundle or container that holds related code. Packages can be shared and distributed, so other developers can use them in their projects.

In summary, a module is a single file or a group of files, a package is a collection of related modules or files, and a library is a package that provides reusable code for specific purposes. Modules are the building blocks, packages are the containers, and libraries are the ready-to-use tools for JavaScript development.

PACKAGES

Packages are files of code. We can install packages written by other developers using a tool called NPM [Node Package Manager].

NPM

- **Package installation**: npm allows developers to easily install packages from the npm registry using the npm install command.

- **Package management**: npm allows developers to manage their project's dependencies, including updating and removing packages.

- **Version control**: npm provides version control for packages, allowing developers to specify which version of a package they want to use in their project.

- **Publishing packages**: npm allows developers to publish their own packages to the npm registry, making them available for others to use.

Lodash

Lodash is a utility library that provides helpful functions for common programming tasks, such as array manipulation, object iteration, and more.

```
const numbers = [1, 2, 3, 4, 5];
const sum = _.sum(numbers);
console.log("Sum of numbers:", sum);
```

package.json: The package.json file is like an information card for your Node.js project. It holds details such as the project's name, version, and dependencies. It helps you keep track of what your project needs to work correctly and allows you to manage its configuration and scripts.

node_modules: The node_modules directory is where all the required code packages (dependencies) for your project are stored. When you install packages using npm, they are placed in this directory. It's like a storage place for the code that your project depends on. You don't need to worry about it too much; npm handles it for you.

npm install: Installs dependencies listed in the package.json file. It reads the dependencies section and fetches the required packages from the npm registry. If no package.json is present, it installs packages based on the command-line arguments

npm install <package-name>: Installs a specific package. The package is downloaded from the npm registry and added to the node_modules directory. The package name can be followed by a version number or a tag to specify a specific version

npm uninstall <package-name>: Removes a specific package from the node_modules directory and updates the package.json file to remove the package from the list of dependencies.

npm update: Updates the packages listed in the package.json file to their latest versions, respecting version constraints specified in the file. It fetches the latest versions of the packages from the npm registry.

Moment

Moment.js is a library for parsing, manipulating, and formatting dates and times.

```
const moment = require('moment');


const date = moment().format('YYYY-MM-DD');
console.log("Current date:", date);
```

REACT

React is a library for building user interfaces

VITE

Vite is a build tool that promotes a faster development experience for developers.

```
$ npm create vite@latest
```

FOLDER STRUCTURE

There is importance in creating a proper file structure in a React application. A well-organized file structure can help to improve the code maintainability and readability of the application. It makes it easier for developers to find and modify the code, as well as to collaborate with other team members. A good file structure can also make the code easier to test, debug, and deploy. Overall, having a clear and logical file structure can make the development process smoother and more efficient.

src directory

This is the primary directory where you write your React application's source code.

public directory

When you place files in the public directory, they can be accessed directly by their URLs.

main.jsx

The main.jsx file typically serves as the main entry point for a React application

The main.jsx file is responsible for initializing and rendering your React application. It typically includes the necessary imports, sets up any required configurations, and mounts the root React component to the DOM.

JSX

JSX (JavaScript XML) is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. It is commonly used with React to define the structure and content of React components.

In simple terms, JSX allows you to write HTML-like code directly in your JavaScript files. It combines the power of JavaScript and HTML, making it easier to describe the structure and behavior of user interfaces.

JavaScript expressions can be used inside JSX with curly brackets {}

```
<h1>{10+1}</h1>
```

Javascript transpiler transforms JSX into regular js objects first before passing it to the browser. JSX is one of the many key and valuable features of React that make it such a powerful and popular tool for front-end developers.

Dynamic Variable

```
function Greeting() {
  const name = 'Alice';

  return <h1>Hello, {name}!</h1>;
}
```

Arithematic expression

```
function MathOperation() {
  const num1 = 10;
  const num2 = 5;

  return <p>The result is {num1 + num2}.</p>;
}
```

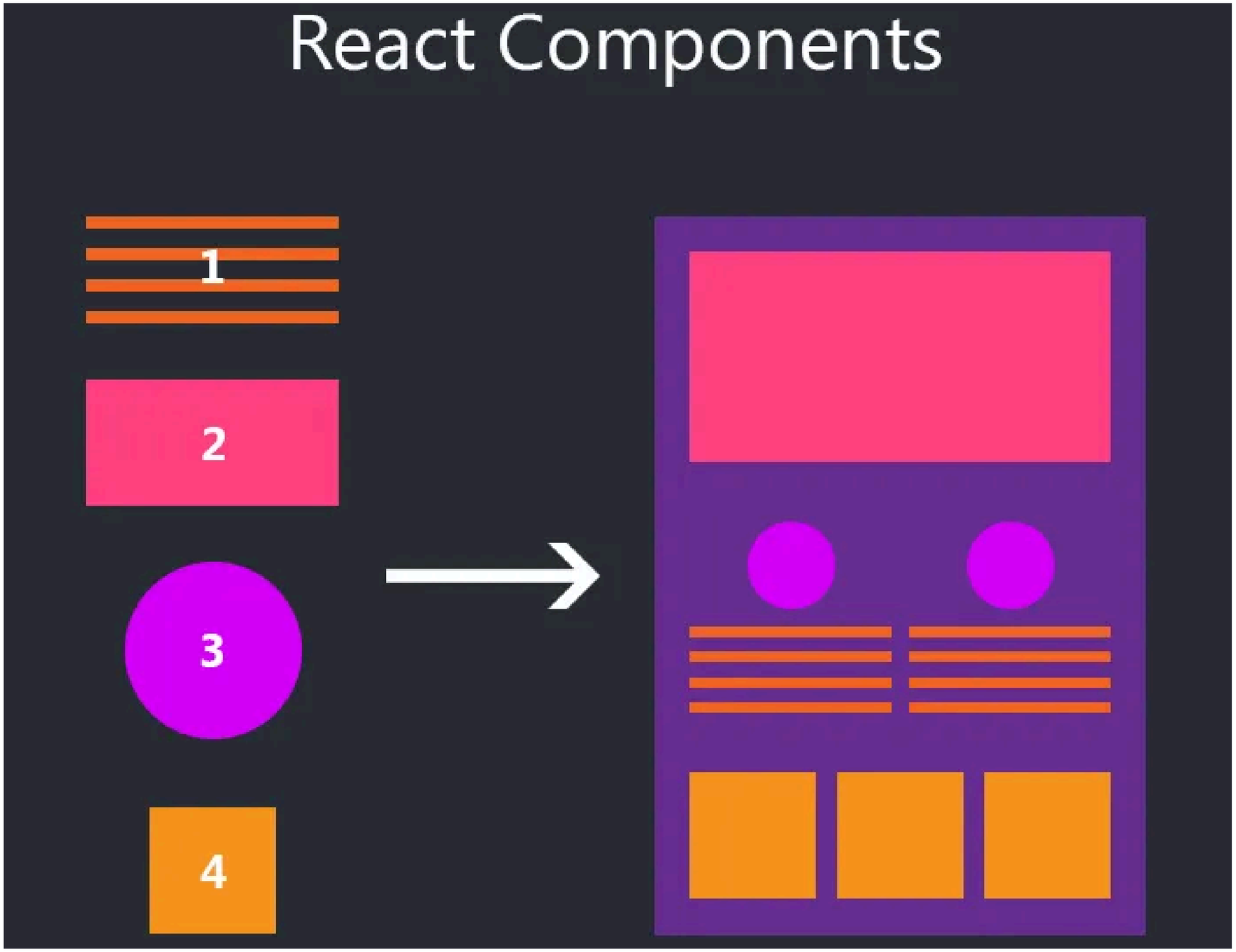
Condition Logic

```
function ConditionalRendering() {
  const isLoggedIn = true;

  return (
    <div>
      {isLoggedIn ? <p>Welcome, user!</p> : <p>Please log in.</p>}
    </div>
  );
}
```

Mapping an Array

```
function ListItems() {
  const items = ['Apple', 'Banana', 'Orange'];

  return (
    <ul>
      {items.map((item, index) => (
        <li key={index}>{item}</li>
      ))}
    </ul>
  );
}
```

Components


React Components
1
2
3
4

A component is basically JavaScript **function**.

Components are the heart of any React Application. You can think of components as building blocks. You can make a beautiful structure using these building blocks.

You can make a whole UI merging these components. You can think of them as custom HTML elements. They are independent and isolated from each other and because of that, they can be maintained and managed very easily.

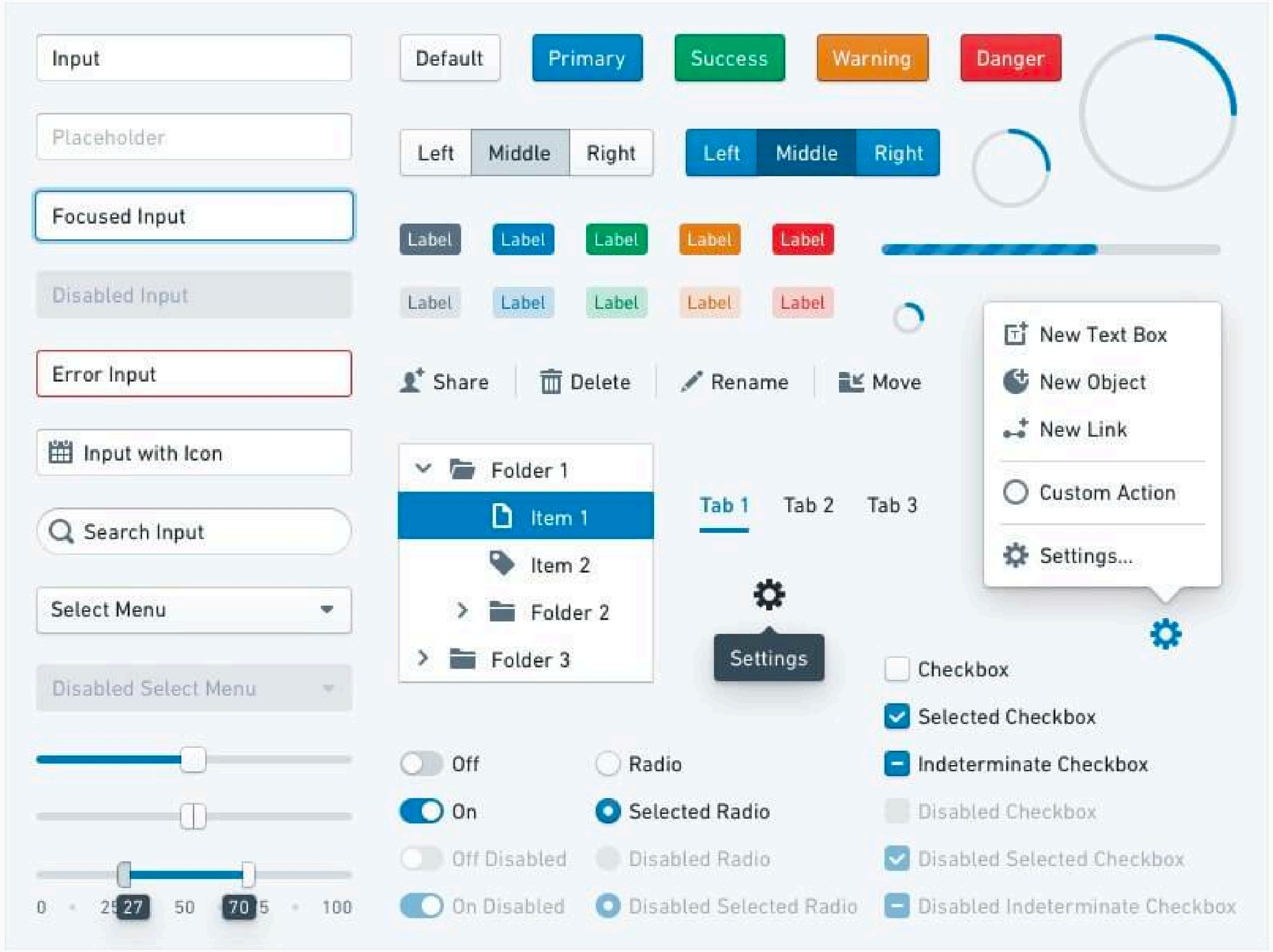
But the most important thing is, they are reusable. So, you can create a React component once and use it as many times as you want.

Each button can be a React component

'Display number' component

'Display screen' component

Entire calculator can be a component that wraps around all other sub-components


Input
Placeholder
Focused Input
Disabled Input
Error Input
Input with Icon
Search Input
Select Menu
Disabled Select Menu
0
25
27
50
70
5
100
Default
Primary
Success
Warning
Danger
Left
Middle
Right
Left
Middle
Right
Label
Label
Label
Label
Label
Label
Label
Label
Label
Label
Share
Delete
Rename
Move
Folder 1
Item 1
Item 2
Folder 2
Folder 3
Tab 1
Tab 2
Tab 3
Settings
New Text Box
New Object
New Link
Custom Action
Settings...
Off
On
Off Disabled
On Disabled
Radio
Selected Radio
Disabled Radio
Disabled Selected Radio
Checkbox
Selected Checkbox
Indeterminate Checkbox
Disabled Checkbox
Disabled Selected Checkbox
Disabled Indeterminate Checkbox

React apps are made out of *components*. A component is a piece of the UI (user interface) that has its own logic and appearance. A component can be as small as a button, or as large as an entire page.

```
function MyButton() {
  return (
    <button>I'm a button</button>
  );
}
```

```jsx
export default function MyApp() {
  return (
    <div>
      <h1>Welcome to my app</h1>
      <MyButton />
    </div>
  );
}
```

Notice that `<MyButton />` starts with a capital letter. That's how you know it's a React component. React component names must always start with a capital letter, while HTML tags must be lowercase.

Export and Import

Importing and **exporting** in React JS will help us write modular code, i.e., splitting code into multiple files. *Importing* allows using contents from another file, whereas *exporting* makes the file contents eligible for importing. The basic idea behind imports and exports is to exchange contents between several JavaScript files.

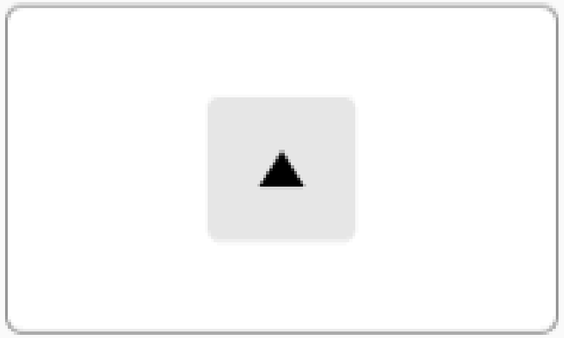
React core concepts

1. **Components**
2. **Props**
3. **State**

Components

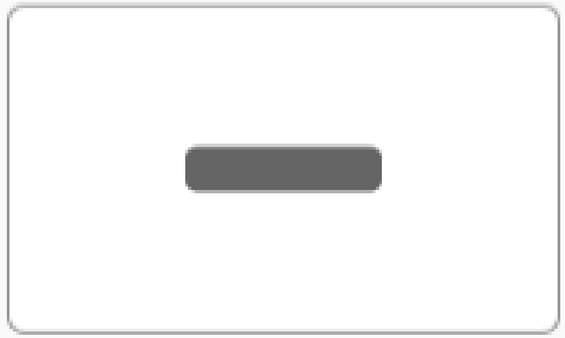
Media Component

Image

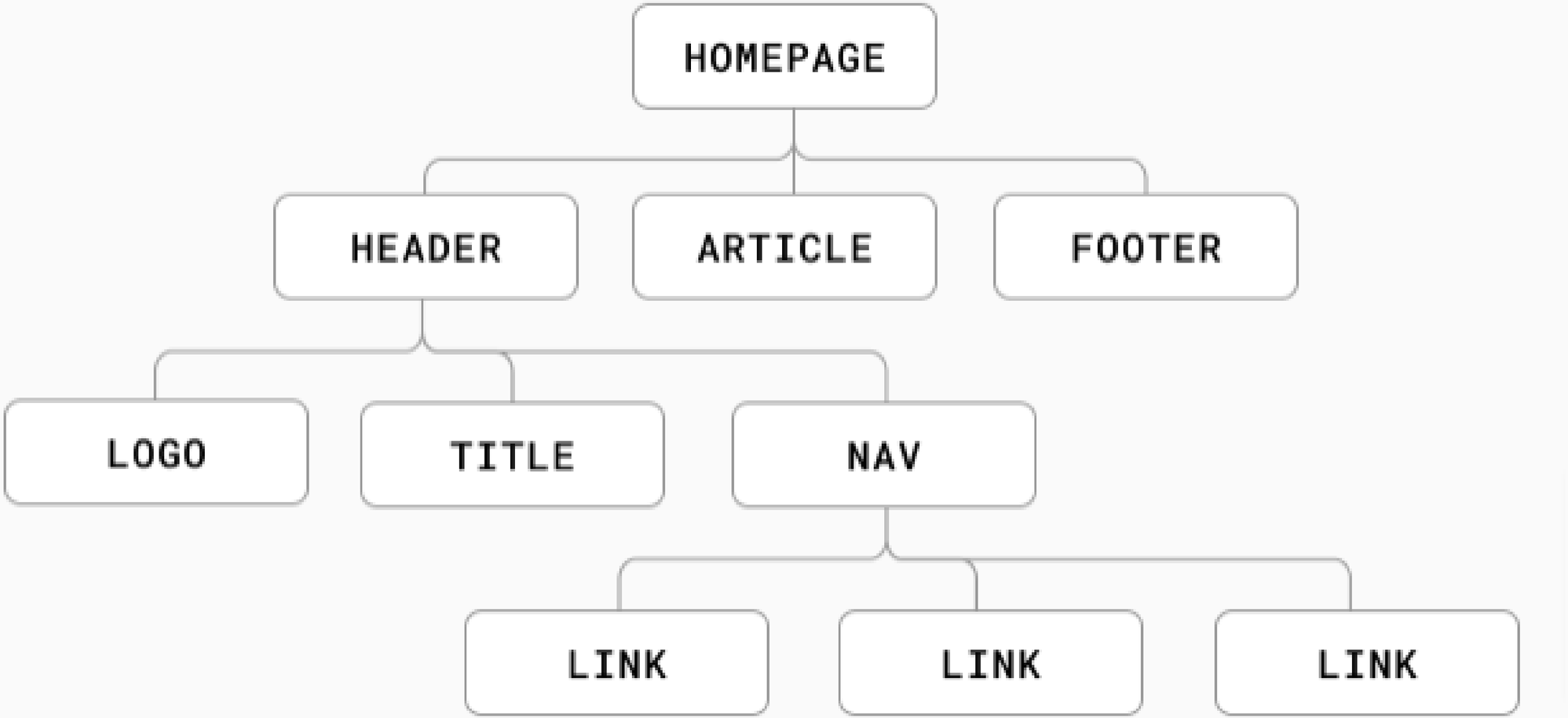


Description

Button



Component Tree



Props

In React, props (short for properties) are a mechanism for passing data from a parent component to a child component. Props are immutable, meaning they cannot be modified by the child component.

IMAGES

onClick

The onClick event handler in React is used to respond to click events on elements, such as buttons, links, or any other interactive elements in your application. It allows you to specify a function that will be executed when the element is clicked by the user.

```
function Button() {
  const handleClick = () => {
    console.log('Button clicked!');
  };

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

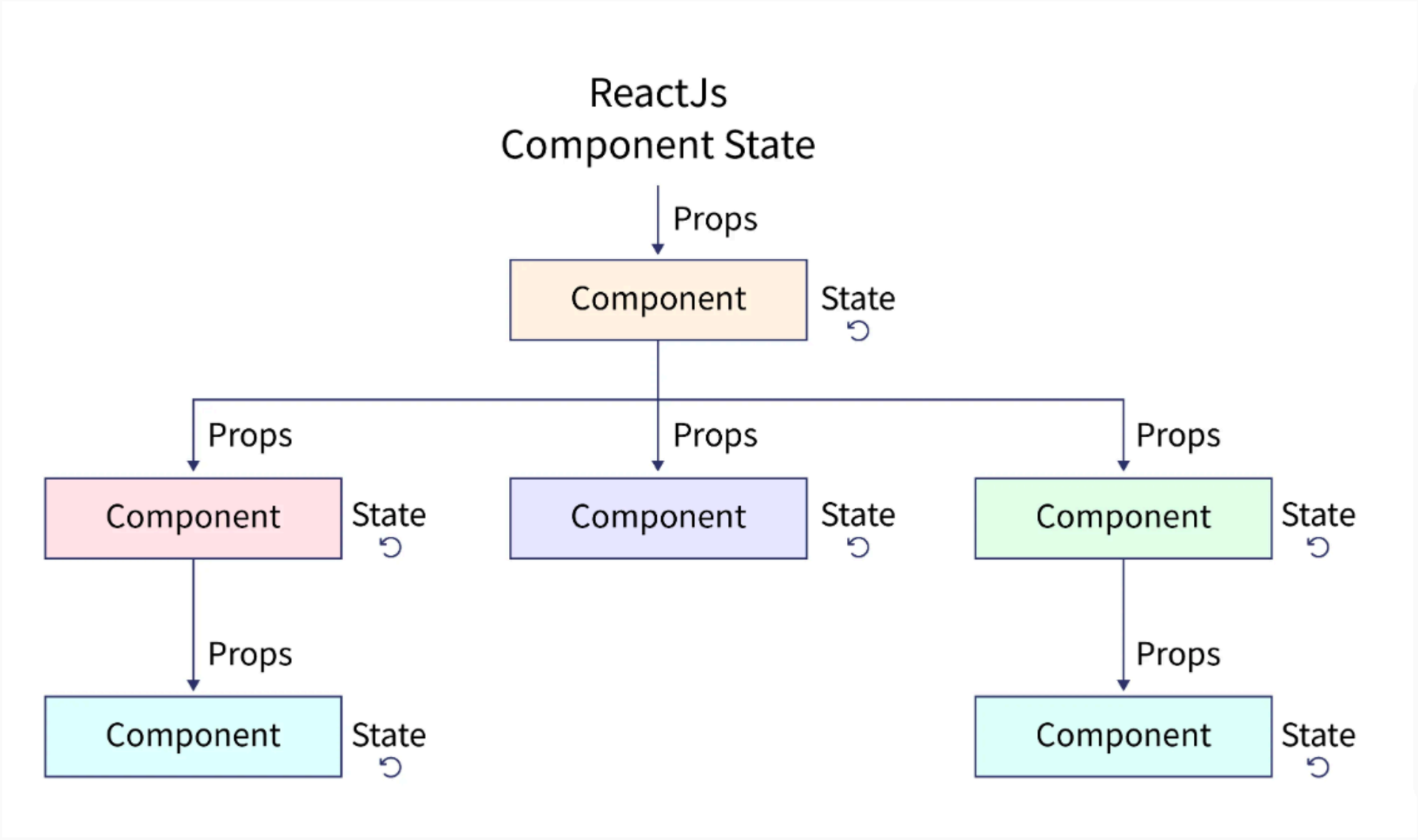
STATE

There is a constant flow of dynamic data through react components. Hence to manage this dynamic nature of data, we have state objects. The state object is used to store and manipulate the changing data of any component. State also ensures the re-rendering of the component to the browser in case of any change

Functional components and the useState() hook

we can use state in a functional component with the useState() hook provided by React. To use this hook we call the useState() function inside the component and pass in one argument which will be the initial value for the state.

From calling the useState() function we get two things back from React and we use array destructuring to deconstruct them into two variables.


ReactJs
Component State
Props
Component
State
Props
Component
State
Props
Component
State
Props
Component
State
Props
Component
State

onChange

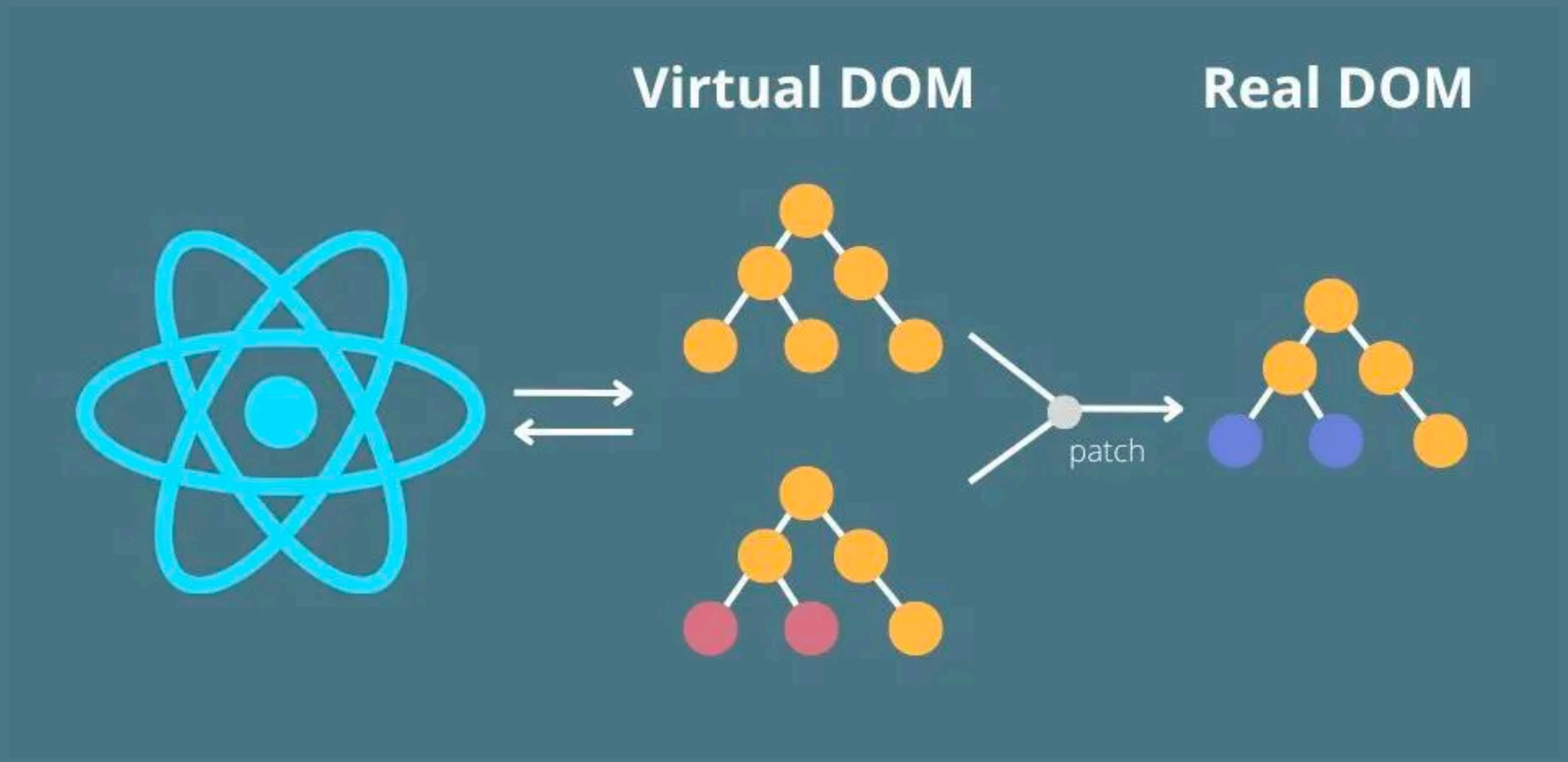
What is the onChange Event Handler?

JavaScript allows us to listen to an input’s change in value by providing the attribute onchange. React’s version of the onchange event handler is the same, but camel-cased.

onKeyDown

Changing the state object

The most reasonable way to handle changes is by using the set function in react. This method ensures that all the updates are made in an asynchronous manner and also ensures that react re-renders that component.


Virtual DOM
Real DOM
patch

The DOM is a way for programmers to interact with a web page. It's a tree-like representation of all the elements that make up the page, including the text, images, and buttons.

The React Virtual DOM (VDOM) is a lightweight representation of the real DOM. It is used by React to efficiently update the real DOM when state changes.

When a state change occurs in React, it first updates the VDOM. The VDOM is then compared to the real DOM to determine which elements have changed. React then only updates the changed elements in the real DOM.

This approach is much more efficient than updating the real DOM directly, because it only updates the elements that have actually changed. This can lead to significant performance improvements, especially for applications with large and complex user interfaces.

The VDOM is also diffed against the previous VDOM, which means that React only updates the elements that have changed since the last render. This further improves performance, as it prevents React from having to re-render the entire UI on every state change.

The VDOM is a key part of what makes React so fast and efficient. It allows React to update the UI quickly and smoothly, even for complex applications.

Mutation in terms of React state is any direct change to the value of a state variable.

If we **mutate** the state directly, it will change the reference of the state in the previous virtual DOM as well. So, React won't be able to see that there is a change of the state, and so it won't be reflected in the original DOM until we reload.

When you mutate the state in React, you are directly changing the value of the state variable. This can cause unexpected behavior in your React application.

If you mutate the state of a component, and that component has children, the children may not be re-rendered, even though the state has changed. This can lead to inconsistencies in the UI.

Additionally, mutating the state can make it difficult to debug your React application. When the state is mutated directly, it can be difficult to track down where and when the mutation occurred.

For these reasons, it is generally recommended to avoid mutating the state in React. Instead, you should use the setState() function to update the state.

This approach allows React to track changes to the state, and to ensure that all child components are re-rendered when the state changes.

Javascript has 5 data types that are passed by value: **Boolean**, **null**, **undefined**, **String**, and **Number**. We'll call these primitive types.

Variables that are assigned a non-primitive value are given a reference to that value. That reference points to the object's location in memory. The variables don't actually contain the value.

Variables	Values	Addresses	Objects
obj	<#234>	#234	{ first: 'reference' }

Avoiding array/object mutations

Never directly modify an object or array stored in useState. Instead, you should create a new updated version of the object or array and call setState with the new version.

Using Spread operator in ES6

```
const person = { name: 'John Doe', email: 'john@doe.com' };
const samePerson = { ...person, age: 27, nationality: 'Irish' };
```

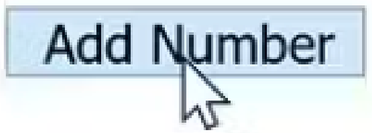
```
const numbers = [1, 2];
const moreNumbers = [...numbers, 3];
```

Functional components and the useState() hook

The first is called the **state** variable and we can give it any name we want.

The second value we get back is a function that allows up to update the state and similarly we can choose an appropriate name for it although the convention is to give it the same name as the state value but prefixed with the word "set"

```
const [ numbers, setNumbers ] = useState([0, 1, 2, 3]);
```



- 0
- 1
- 2
- 3
- 4

Lifting State

What happens when we have two different components and we would like them to react to the changes in state of another component.

Difference Between State and Props

Difference	Props	State
Component Data	Props get data from the parent component (i.e. from outside).	State is used to store and manipulate the data within the component (i.e. within).
Data Scope	Exchange of data between different components.	It is only used within a component.
Data Accessibility	Any data from parthe ent component is read-only and cannot be changed.	Data can be modified inside the component usthe ing setState() method.
Initial Value	Initial value is not required but can be passed the rom parent component.	Initial value is required and received believe the om parent component.

React Routing

React is an open-source front-end JavaScript library that allows developers to create user interfaces using UI components and **single-page** applications. One of the most important features we always want to implement when developing these applications is routing.

Routing is the process of redirecting a user to different pages based on their action or request. In React routing, you'll be using an external library called React router

SPA

A single-page application (SPA) is a web application or website that operates within a single web page, providing a more seamless and interactive user experience. In traditional web applications, navigating between different pages involves full page reloads, resulting in slower response times and less fluid interactions.

SPAs, on the other hand, load all the necessary HTML, CSS, and JavaScript resources when the user accesses the application initially. Afterward, any additional content or data is dynamically loaded and updated within the same page without requiring a full reload.

The key advantages of SPAs

1. **Improved performance**: Since SPAs only load the necessary resources once, subsequent interactions feel faster as there's no need to fetch and render entire pages again.
2. **Enhanced user experience**: SPAs offer a more fluid and desktop-like experience, as the application responds quickly to user input and dynamically updates the content without refreshing the whole page.
3. **Offline capabilities**: SPAs can use caching techniques to store necessary resources locally, allowing them to continue functioning even when the user is offline or experiencing a poor internet connection.
4. **Modular development**: SPAs can be developed using modular components, which enables code reusability, easier maintenance, and faster development cycles.

npm install react-router-dom

How to Setup React Router

To configure React router, navigate to the main.jsx file, which is the root file, and import BrowserRouter from the react-router-dom package that we installed

How to Configure Routes In React

We have now successfully installed and imported React router into our project; the next step is to use React router to implement routing. The first step is to configure all of our routes

```
import { Routes, Route } from 'react-router-dom';
```

How to Access Configured Routes with Links

We can create links within our application so that users can easily navigate, such as a navbar.

This is accomplished with the Link tag, just as it is with the <a> tag in HTML.

```
import { Link } from 'react-router-dom';
```

How to fix No Routes Found Error

When routing, a situation may cause a user to access an unconfigured route or a route that does not exist; when this occurs, React does not display anything on the screen except a warning with the message "No routes matched location."

How to Navigate Programmatically in React

Programmatic navigation is the process of navigating/redirecting a user as a result of an action on a route, such as a login or a signup action, order success, or when he clicks on a back button.

Going Back

```
<button className="btn" onClick={() => navigate(-1)}>
 Go Back
</button>
```

Parameters in routes

Parameters in routes are placeholders that can be matched by the URL and used to pass data to the component that is rendered by the route

Dynamically passing data through the URL is a powerful feature that can be used to create complex and user-friendly applications in React. It is a relatively simple concept to understand, but it can be used to implement a wide range of functionality.

HOOKS

In React, hooks are functions that allow you to use state and other React features in functional components.

useState

useState lets you use local state within a function component. You pass the initial state to this function and it returns a variable with the current state value (not necessarily the initial state) and another function to update this value.

useRef

useRef returns a “ref” object. Values are accessed from the .current property of the returned object. The .current property could be initialized to an initial value—useRef(initialValue), for example. The object is persisted for the entire lifetime of the component.

useEffect

With useEffect, you invoke side effects from within functional components, which is an important concept to understand in the React Hooks era.

What are the effects, really?

- Fetching data
- Reading from local storage

The useEffect hook in React allows you to perform side effects in function components. Side effects include actions like fetching data from an API.

The useEffect control flow at a glance

The useEffect hook is called after every render of the component by default.

The useEffect function takes two arguments: a function that represents the side effect logic and an optional array of dependencies. The function provided will be executed after every render of the component, including the initial render.

No dependency passed

```
useEffect(() => {
  //Runs on every render
});
```

An Empty Array

```
useEffect(() => {
  //Runs only on the first render
}, []);
```

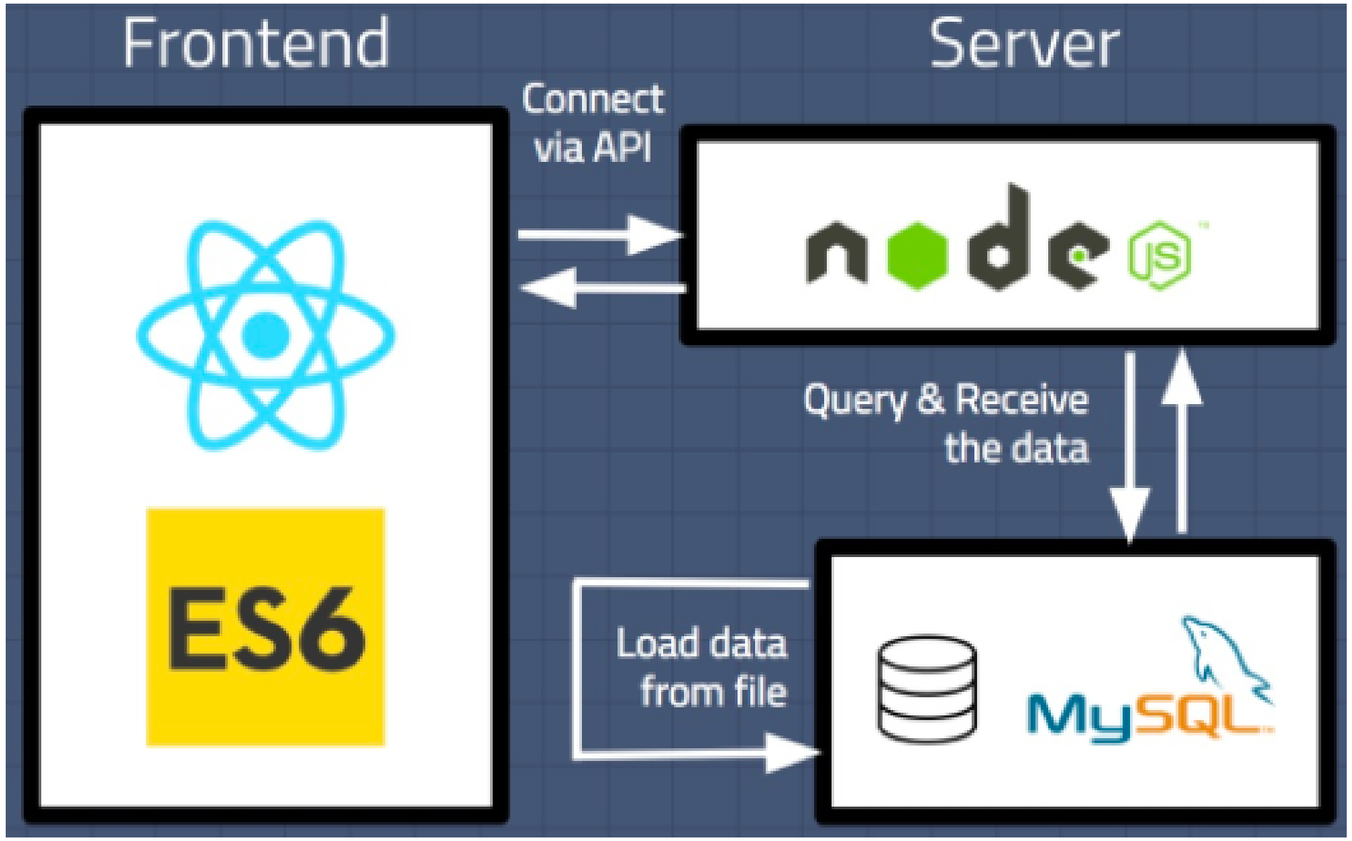
Props or State values

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [prop, state]);
```

BACKEND

What is a back-end API?

A back-end API is a programming interface that helps developers to interact with back-end services for example server.



HTTP

- HTTP stands for HyperText Transfer Protocol.
- It is a protocol used to access the data on the World Wide Web (www).
- The HTTP protocol can be used to transfer the data in the form of plain text, hypertext, audio, video, and so on.

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

POST

The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server.

PUT

The PUT method replaces all current representations of the target resource with the request payload.

PATCH

The PATCH method applies partial modifications to a resource.

DELETE

The DELETE method deletes the specified resource.

Uniform Resource Locator (URL)

- A client that wants to access the document in an internet needs an address .
- The URL defines four parts: method, host computer, port, and path.

- **Method**: The method is the protocol used to retrieve the document from a server. For example, HTTP.

- **Host:** The host is the computer where the information is stored, and the computer is given an alias name. Web pages are mainly stored in the computers and the computers are given an alias name that begins with the characters "www". This field is not mandatory.

- **Port:** The URL can also contain the port number of the server, but it's an optional field. If the port number is included, then it must come between the host and path and it should be separated from the host by a colon.

- **Path:** Path is the pathname of the file where the information is stored. The path itself contain slashes that separate the directories from the subdirectories and files.

Synchronous JavaScript

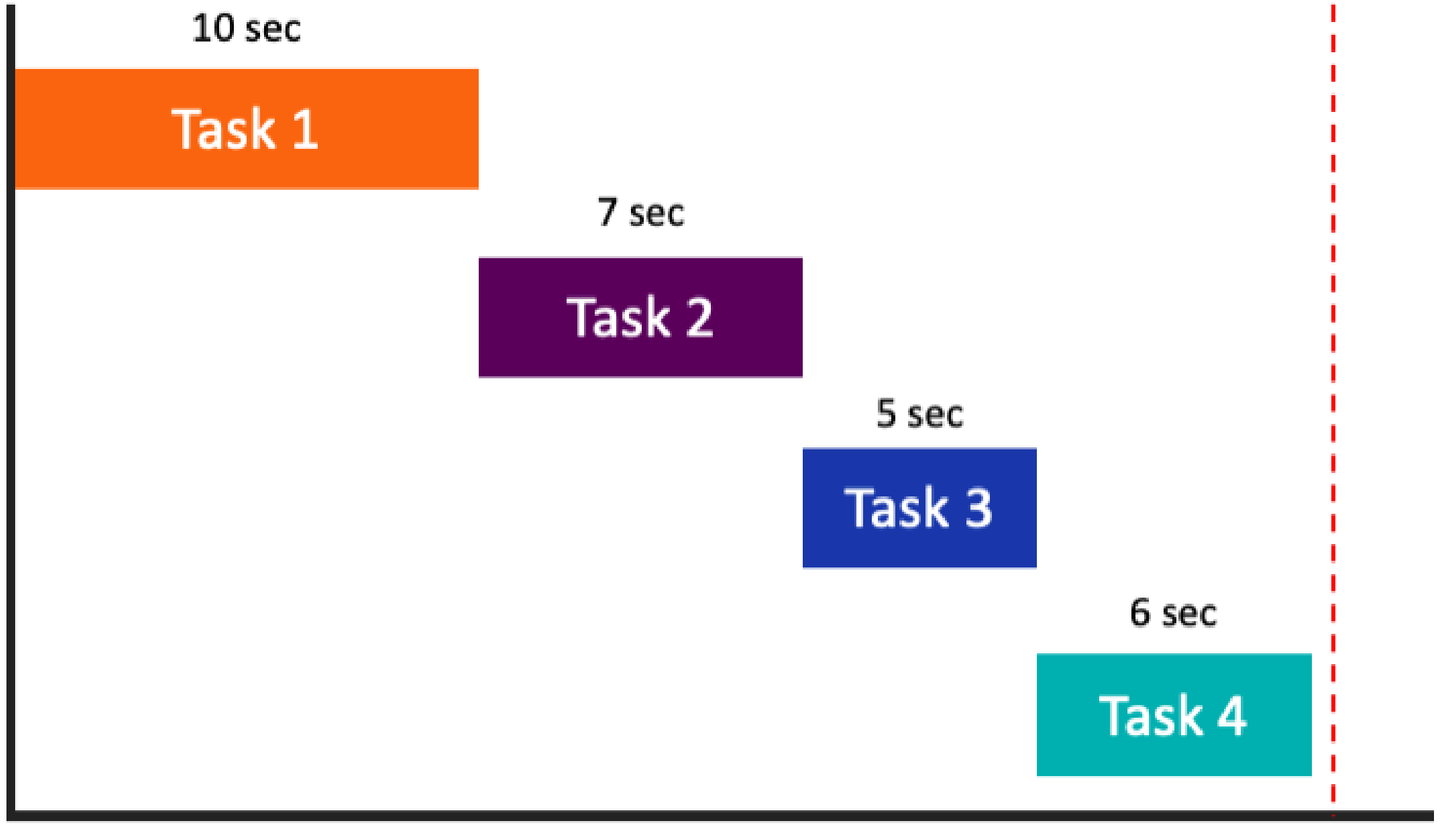
In synchronous programming, tasks are executed sequentially, one after another. Each task must complete before the next one starts. This can lead to blocking the execution thread and may cause the application to become unresponsive.

Asynchronous JavaScript

When we write a program in JavaScript, it executes line by line. When a line is completely executed, then and then only does the code move forward to execute the next line.

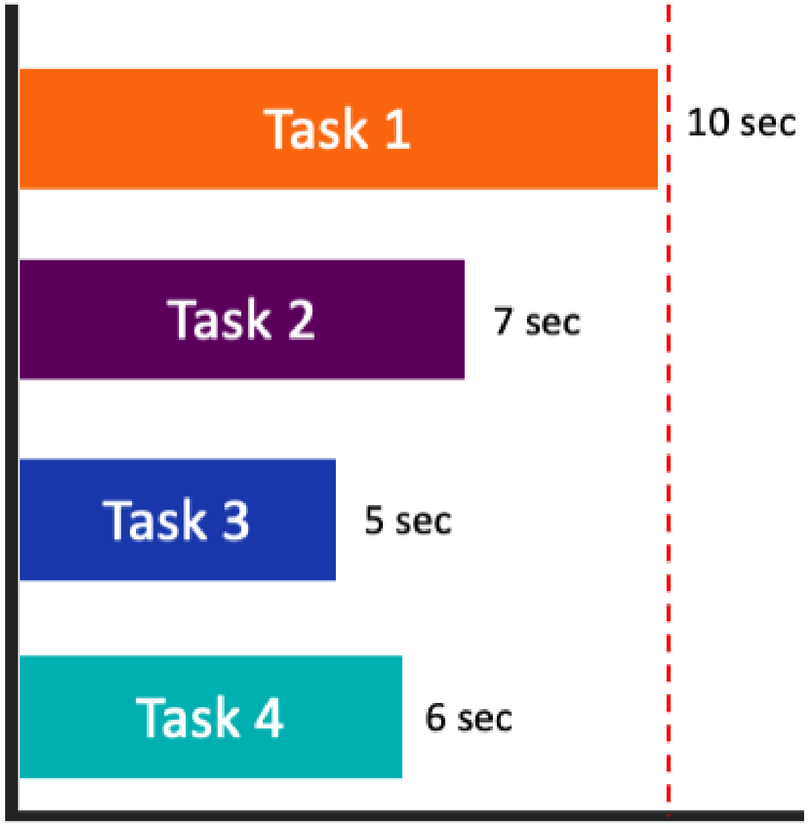
With asynchronous code, multiple tasks can execute at the same time while tasks in the background finish. This is what we call non-blocking code. The execution of other code won't stop while an asynchronous task finishes its work.

SYNCHRONOUS



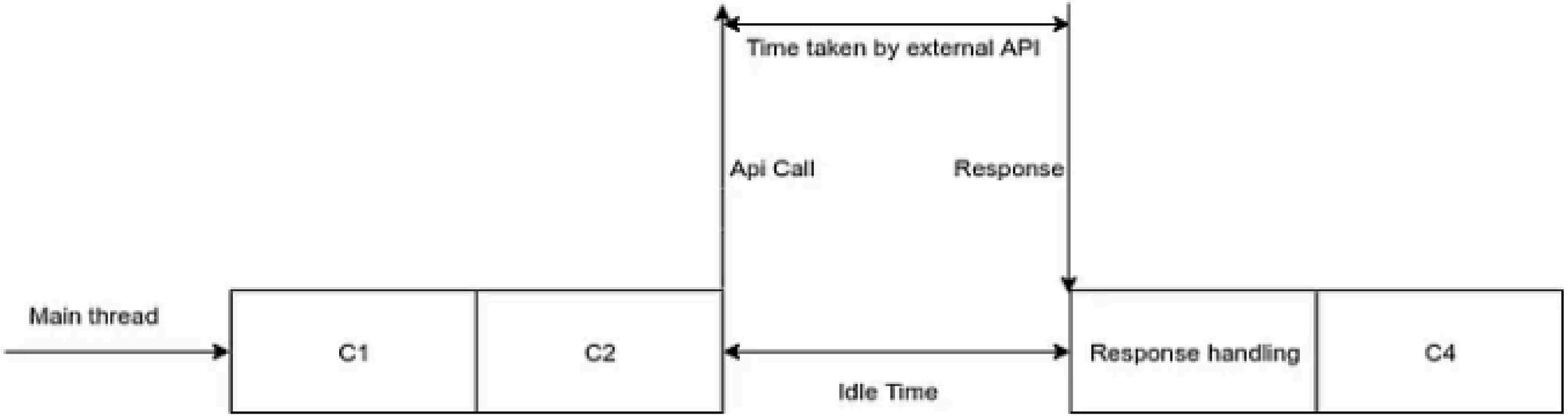
Time taken (28 sec)

ASYNCHRONOUS



Time taken (10 sec)

Synchronous code execution.



In asynchronous programming, tasks can be started and allowed to run independently of each other. The execution of the main thread continues, and once a task is completed, a callback or a promise is used to handle the result.

In asynchronous programming, tasks can be started and allowed to run independently of each other. The execution of the main thread continues, and once a task is completed, a callback or a promise is used to handle the result.

Promises

- Promises were introduced in ES6 to address the callback hell problem and improve code readability when dealing with asynchronous operations.
- A Promise is an object representing the eventual completion or failure of an asynchronous operation, and it has three states: pending, fulfilled, or rejected.
- Promises allow you to chain operations together using .then() and handle errors using .catch().

async/await

- Introduced in ES2017, async/await is a syntactical feature that provides a more elegant way to write asynchronous code using Promises.
- The async keyword is used to declare a function as asynchronous, and it always returns a Promise.
- The await keyword is used inside an async function to pause the execution until the Promise is resolved. It simplifies the syntax and makes the code look more like synchronous code.

Fake Store API

Fake store rest api for your ecommerce or shopping website prototype

Fake Store API

fetch api

Axios api

Axios is an HTTP client library that allows you to make requests to a given endpoint.

Why Use Axios in React

- Axios has function names that match any HTTP methods. To perform a GET request, you use the .get() method.
- Axios has better error handling. Unlike the Fetch API, Axios throws 400 and 500-range errors for you, where you have to check the status code and throw the error yourself.

npm install axios

Create an Axios Instance

```
const instance = axios.create({baseURL:
"https://jsonplaceholder.typicode.com"});
```

Ant Design

```
$ npm install antd --save
```

reacttoastify

```
import { ToastContainer, toast } from 'reacttoastify';
import 'react-toastify/dist/ReactToastify.css';
```

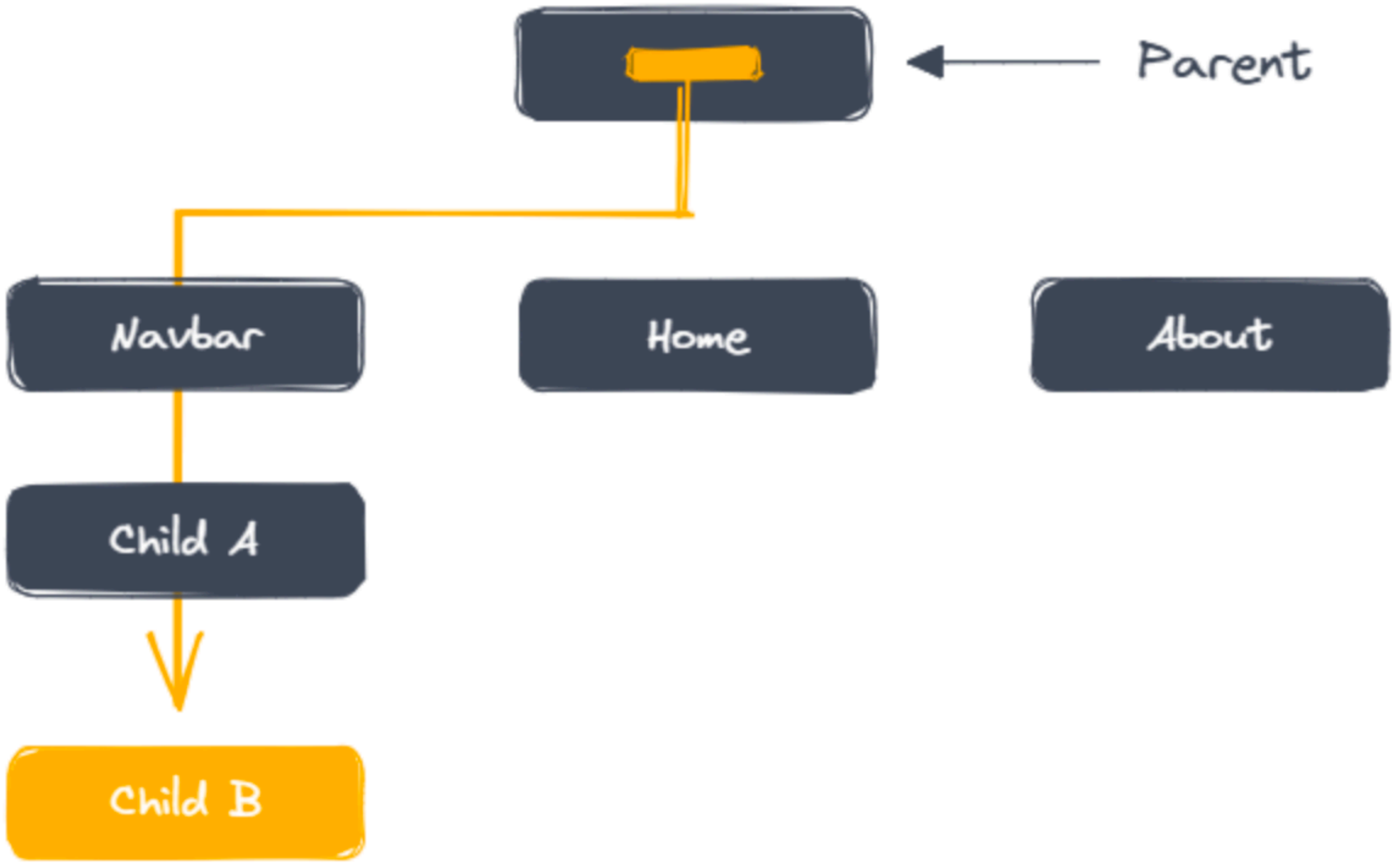
```
const showToastMessage = () => {
toast.success('Success Notification !', {
    position: toast.POSITION.TOP_RIGHT
        });
};
```

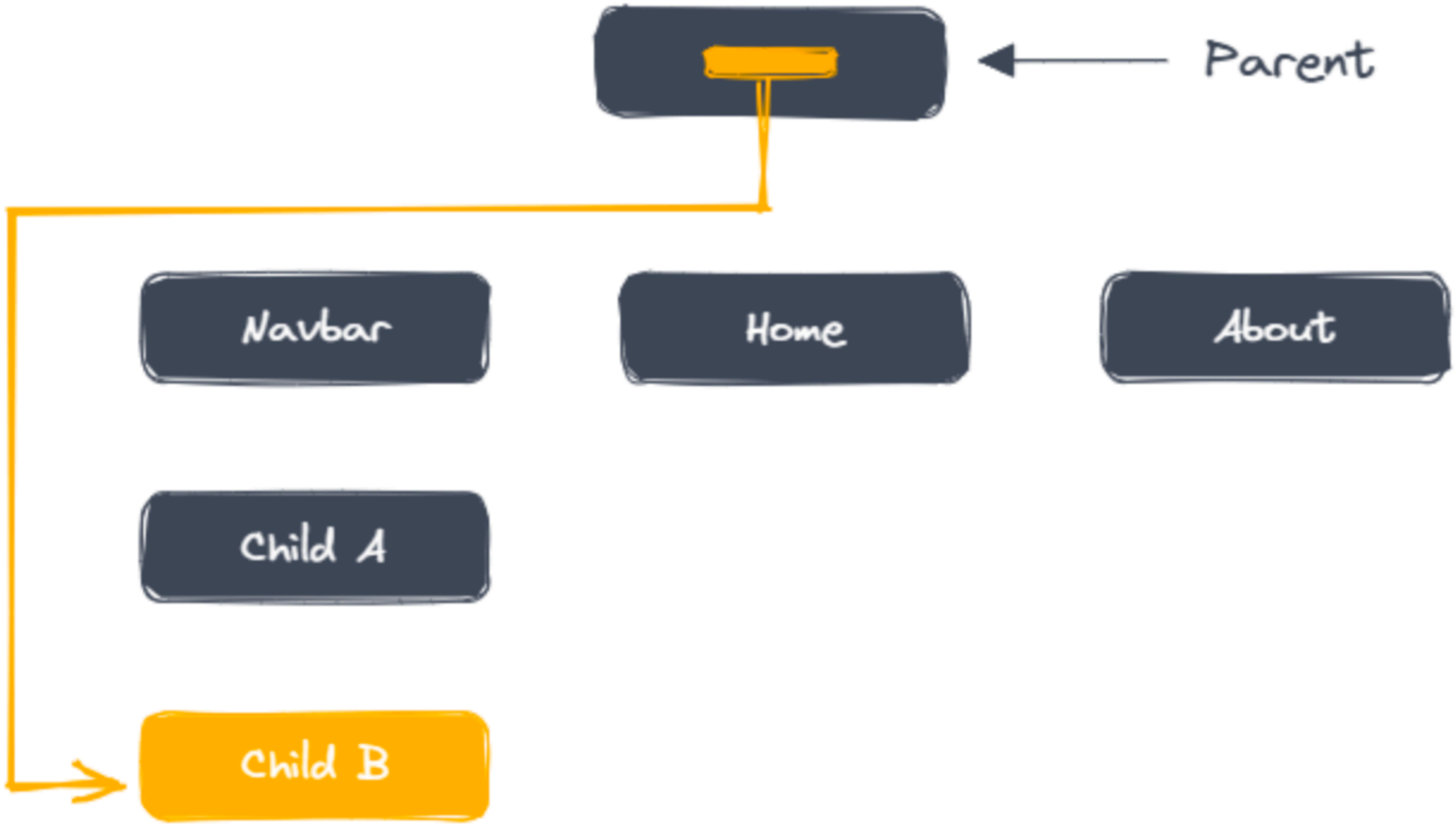
```
toast.success('Success Notification !', {
    position: toast.POSITION.TOP_RIGHT
});
toast.error('Error Notification !', {
    position: toast.POSITION.TOP_CENTER
});
toast.warning('Warning Notification !', {
    position: toast.POSITION.TOP_LEFT
});
toast.info('Information Notification !', {
    position: toast.POSITION.BOTTOM_CENTER
});
toast('Default Notification !', {
    position: toast.POSITION.BOTTOM_LEFT
});
toast('Custom Style Notification with css class!', {
     position: toast.POSITION.BOTTOM_RIGHT,
```

Context

The Problem with Passing Props

In many cases, passing props can be an effective way to share data between different parts of your application. But passing props down a chain of multiple components to reach a specific component can make your code overly cumbersome.

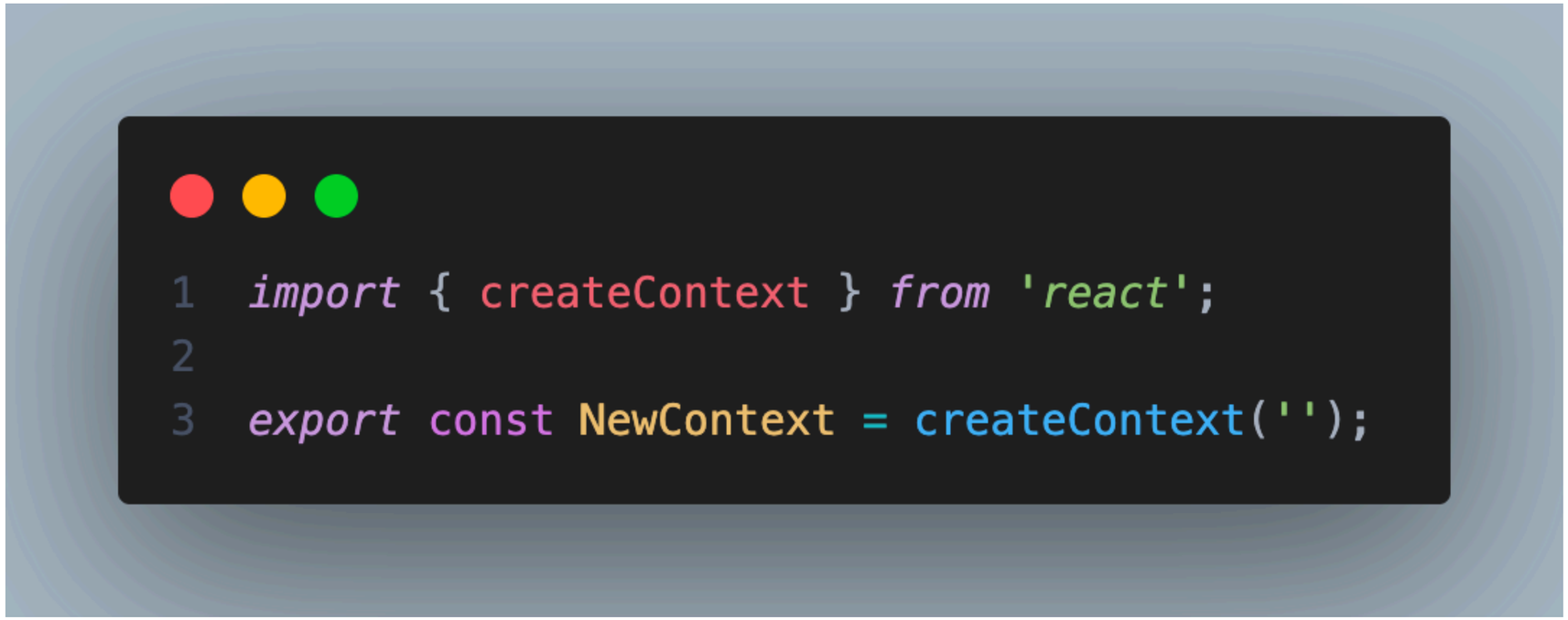

Parent
Navbar
Home
About
Child A
Child B


Parent
Navbar
Home
About
Child A
Child B

How the Context API Works

Context API allows data to be passed through a component tree without having to pass props manually at every level. This makes it easier to share data between components.

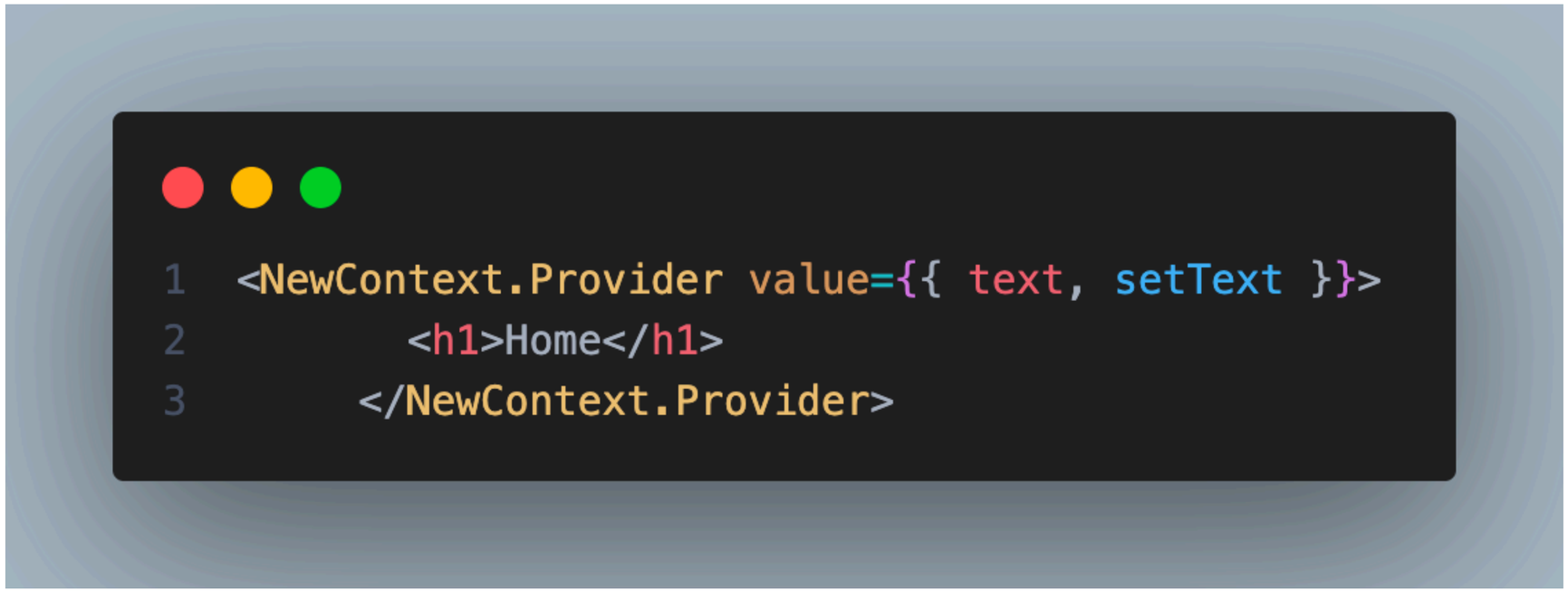
1. Create a Context Object



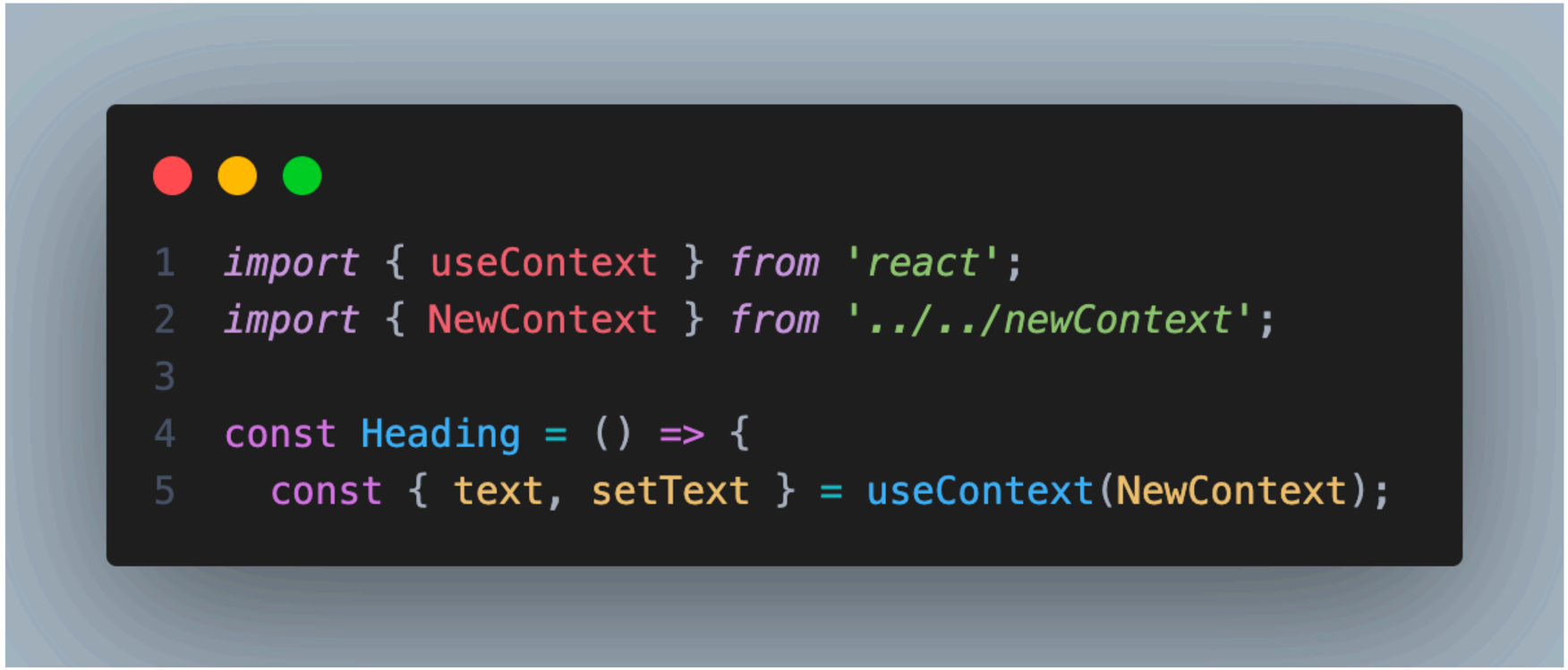
```
1  import { createContext } from 'react';
2
3  export const NewContext = createContext('');
```

2. Create a Context Provider

import { NewContext } from '../../newContext';



3. Use Context

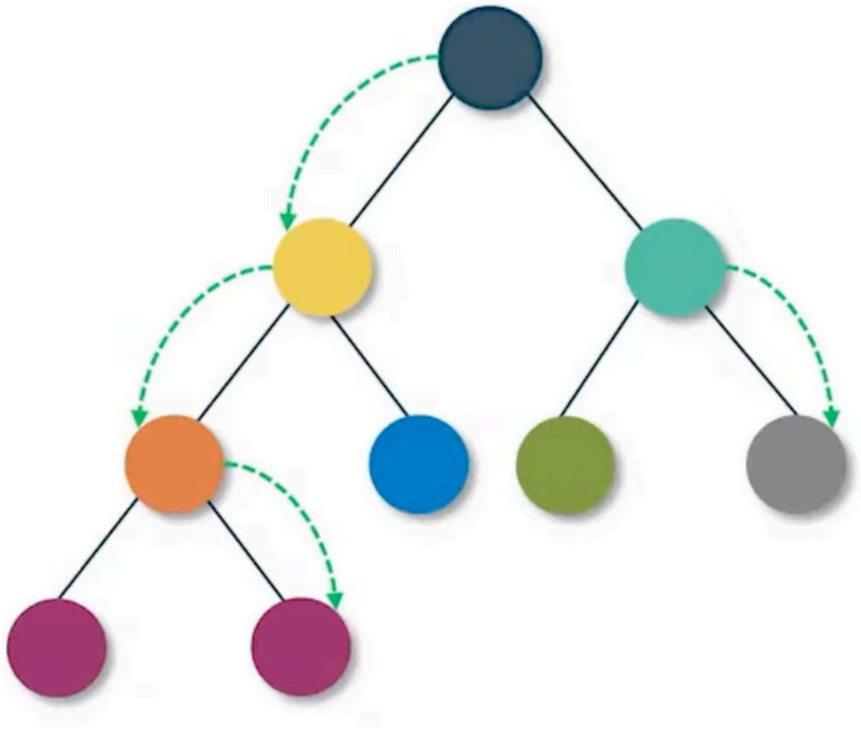


```
1  import { useContext } from 'react';
2  import { NewContext } from '../../newContext';
3
4  const Heading = () => {
5    const { text, setText } = useContext(NewContext);
```

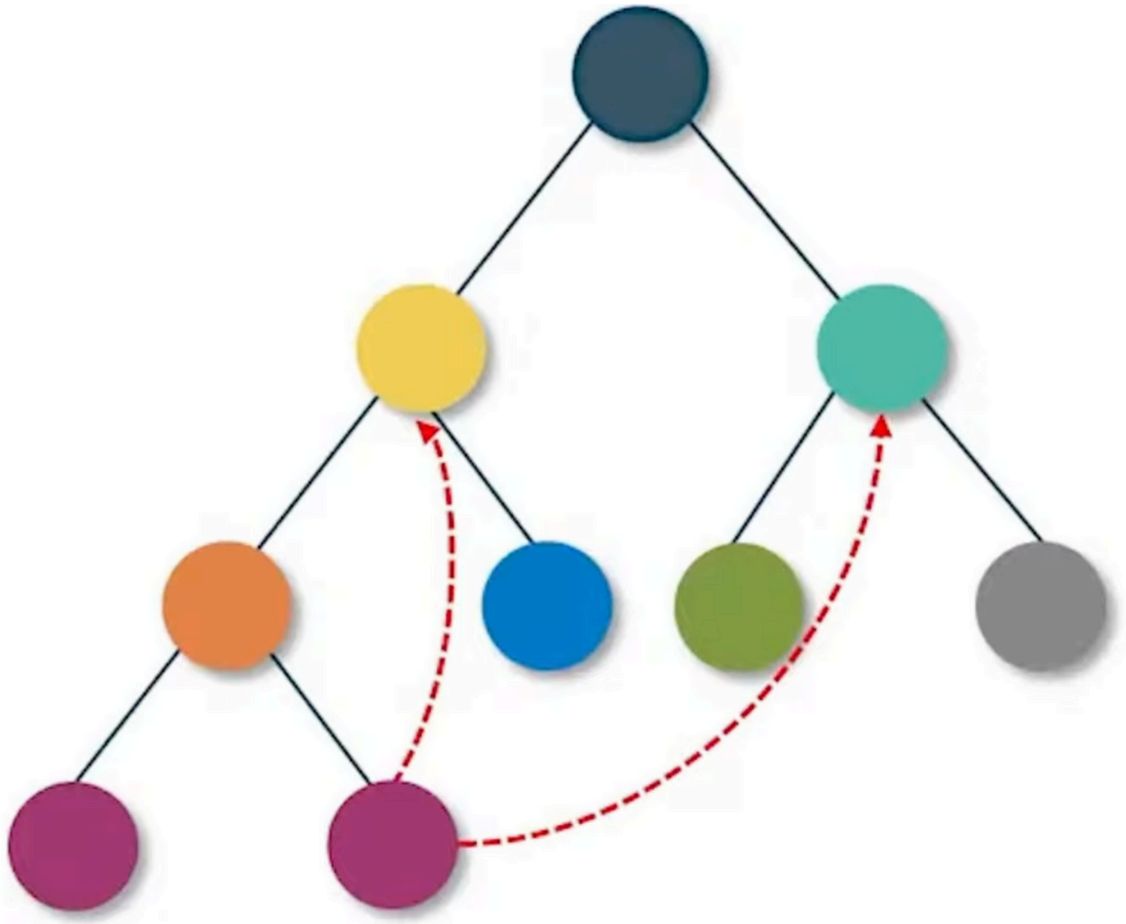
REDUX

Redux is a state management library for JavaScript applications. It provides a predictable way to manage the application state by using a single store that holds the entire state of the application.

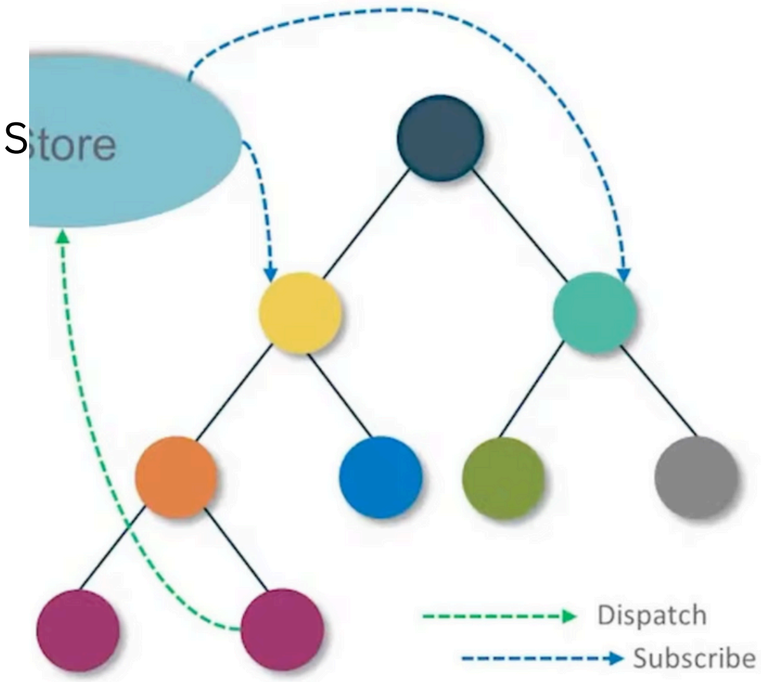
- In React the data flows through the Components
- It follows uni-directional data flow i.e. data always flows from parent to child component.



- Child components can't pass data to their parent component
- The non-parent components can't communicate with each other
- React doesn't recommend direct component-to-component communication this way.



- Redux offers a solution of storing all your application state in one place, called a "store"
- Components then "dispatch" state changes to the store, not directly to other components.
- The components that need to be aware of state changes can "subscribe" to the store



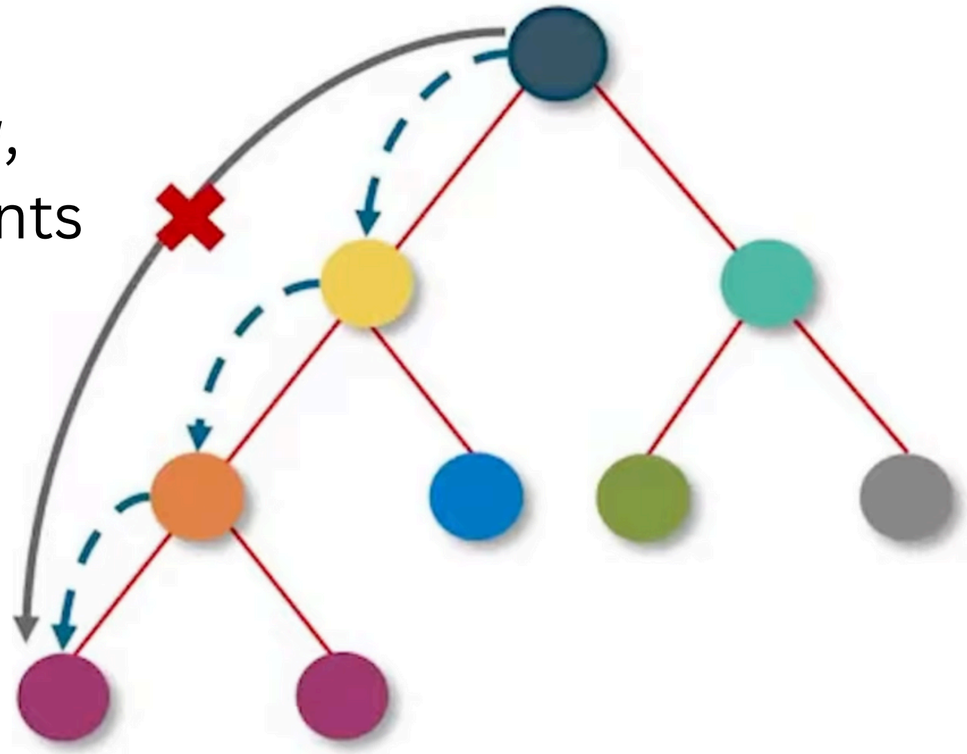
- Redux one of the hottest libraries for front-end development
- Redux is a predictable state container for JavaScript apps
- Mostly used for applications State Management
- Developed by Dan Abramov and Andrew Clark in 2015
- It is inspired by Facebook's Flux and influenced by the functional programming language Elm

Principles of Redux

1. **Single Store**
2. **State is read-only**
3. **Change using pure functions**

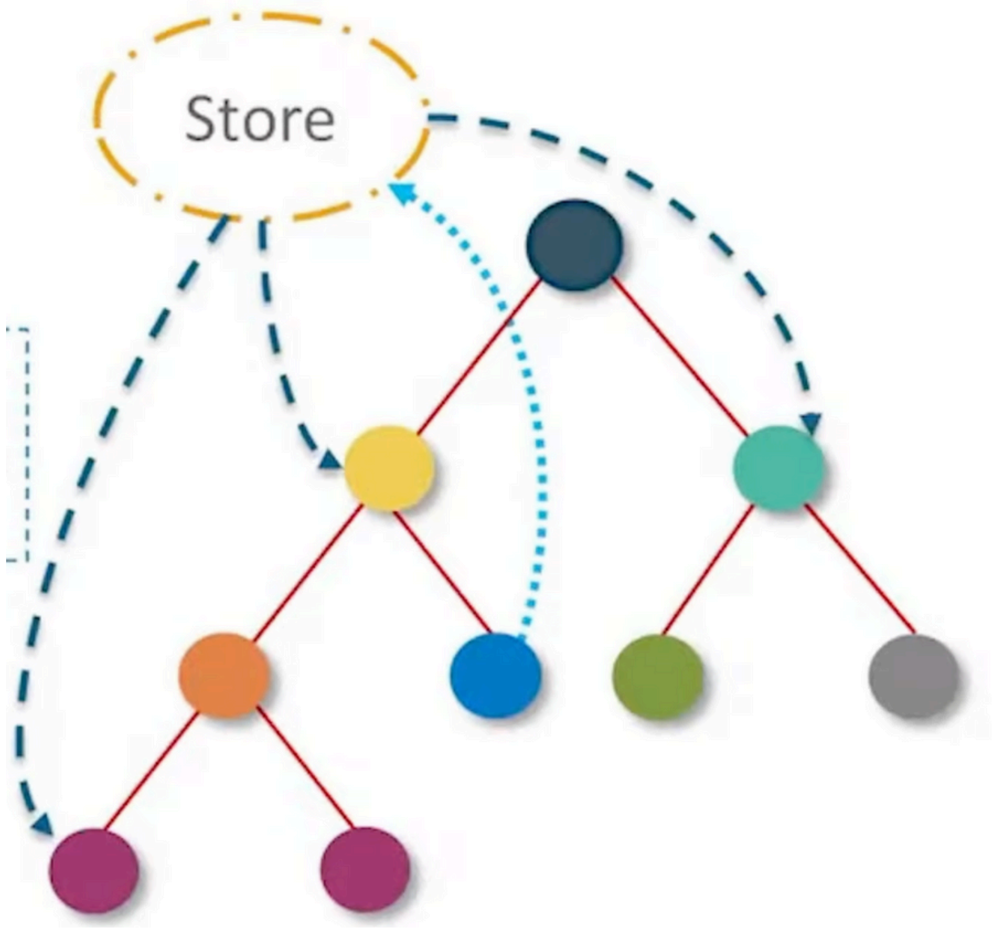
Single Store

With only React Uni-directional Data Flow, direct communication between components is not allowed



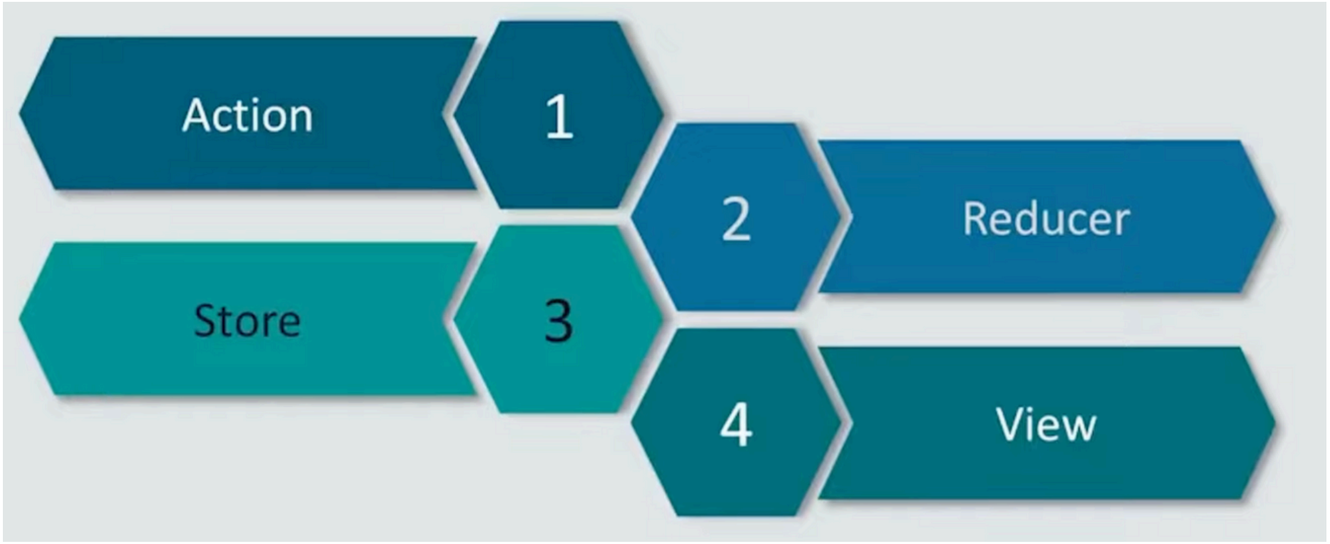
Single Store

Redux uses Store for storing all the application state at one place. Components state is stored in the Store and they receive updates from the store itself.



You can change the state only by triggering an action which is an object describing what happened.

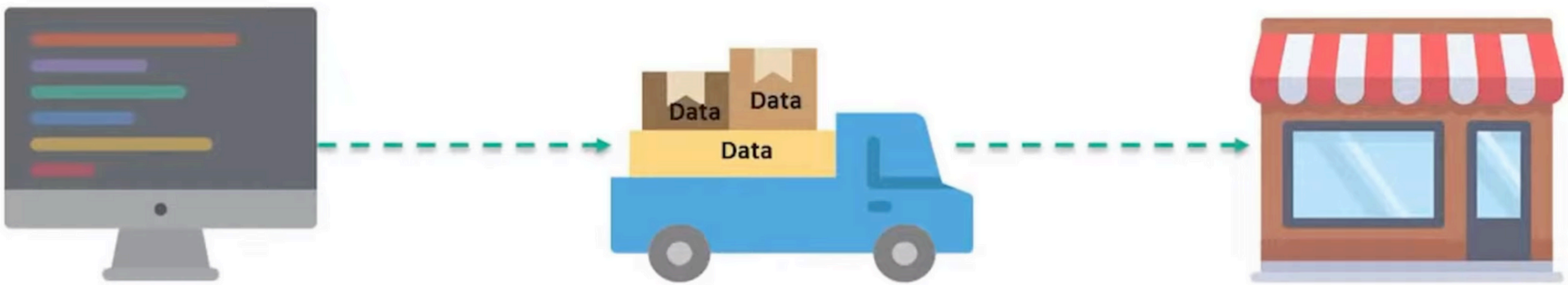
Pure functions called Reducers are used to indicate how the State has been transformed by the Action


Action
1
2
Reducer
Store
3
4
View

Actions

Actions are payloads of information that send data from your application to your store. They are the only source of information for the store. You send them to the store using store.dispatch()

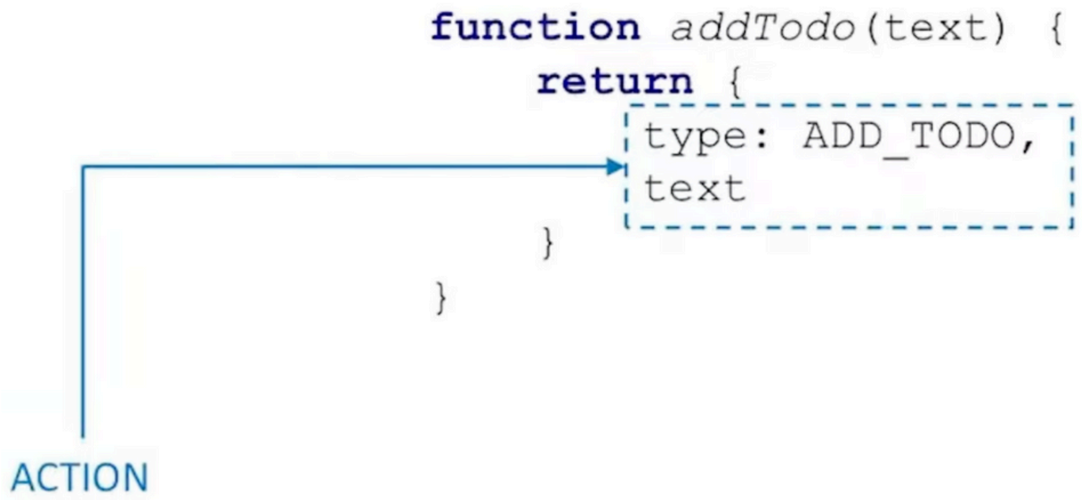
Actions describe the fact that something happened but don't specify how the application's state changes in response.
This is the job of reducers.



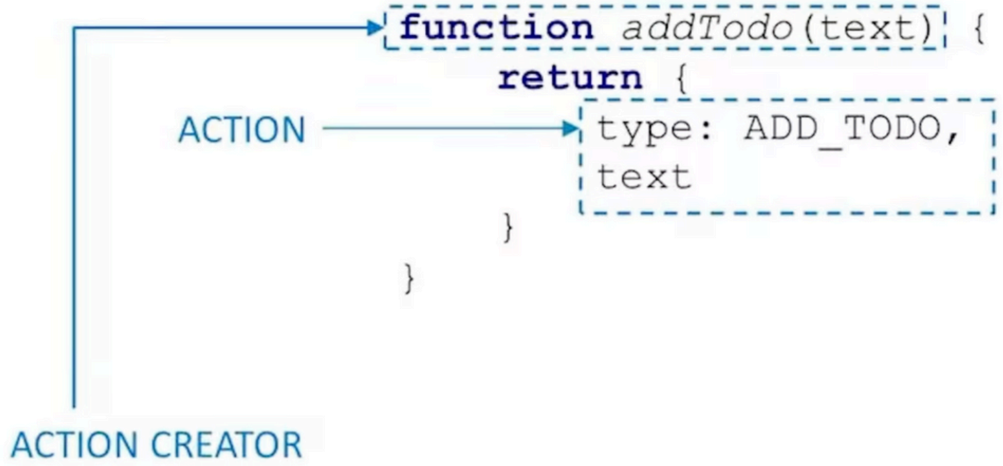
Application

ACTION

Store



- Must have type property that indicates the type of ACTION being performed.
- They must be defined as String constant.
- You can add more properties to it.

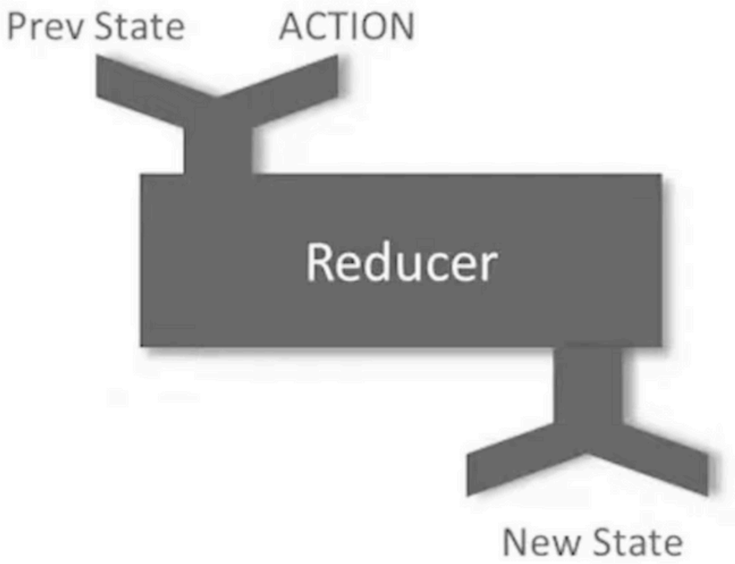


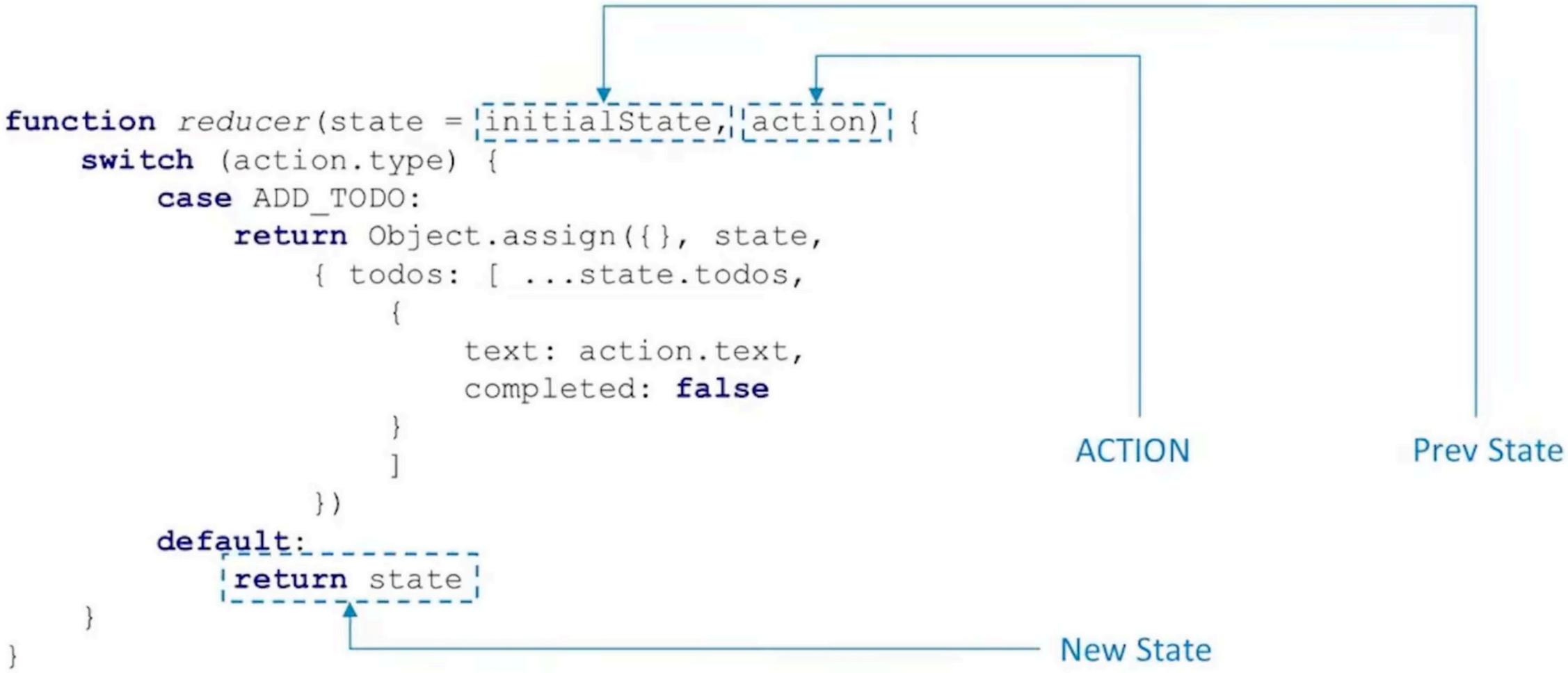
- Functions which create ACTIONS
- Takes in the message, converts it into system understandable format and returns a formatted **Action Object**.

Reducers

Reducers are pure functions which specify how the applications state changes in response to an ACTION

- They do not change the value of the input parameter
- Determines what sort of update needs to be done based on the type of the action, and returns new values
- It returns the previous state if no work needs to be done
- The root reducer slices up the state based on the State Object Keys and passes them to their respective specialized reducers
- Reducers don't manipulate the original state passed to them but make their own copies and then updates them.



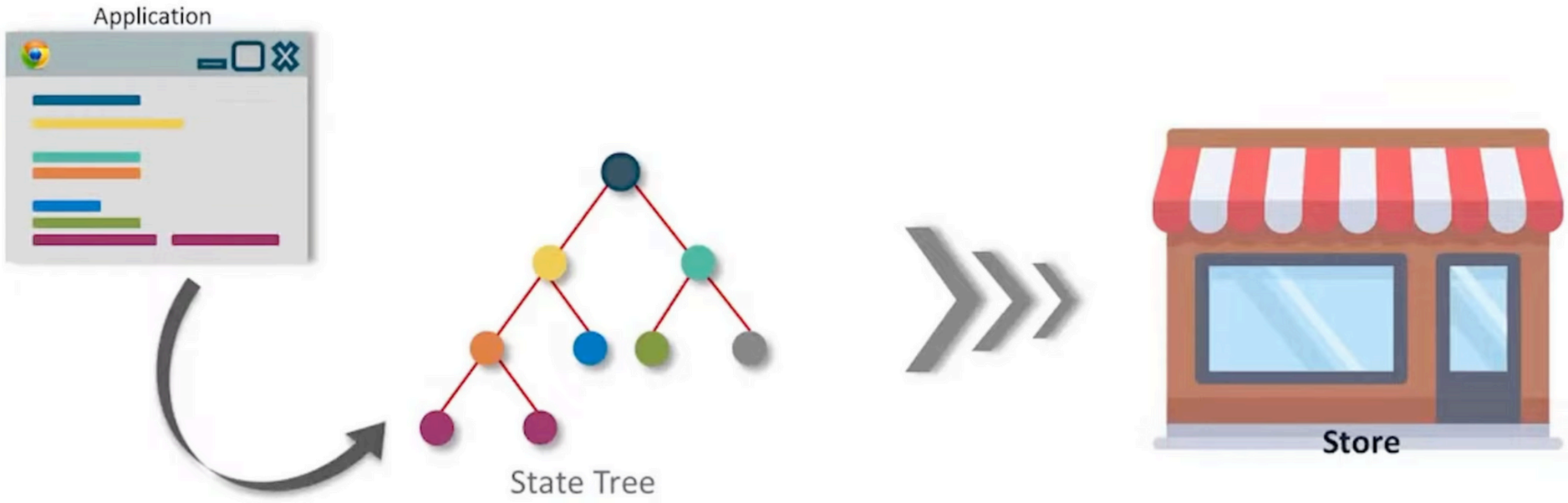

function reducer(state = initialState, action) {
switch (action.type) {
case ADD_TODO:
return Object.assign({}, state,
{ todos: [...state.todos,
{
text: action.text,
completed: false
}
]
})
default:
return state
}
}
ACTION
Prev State
New State

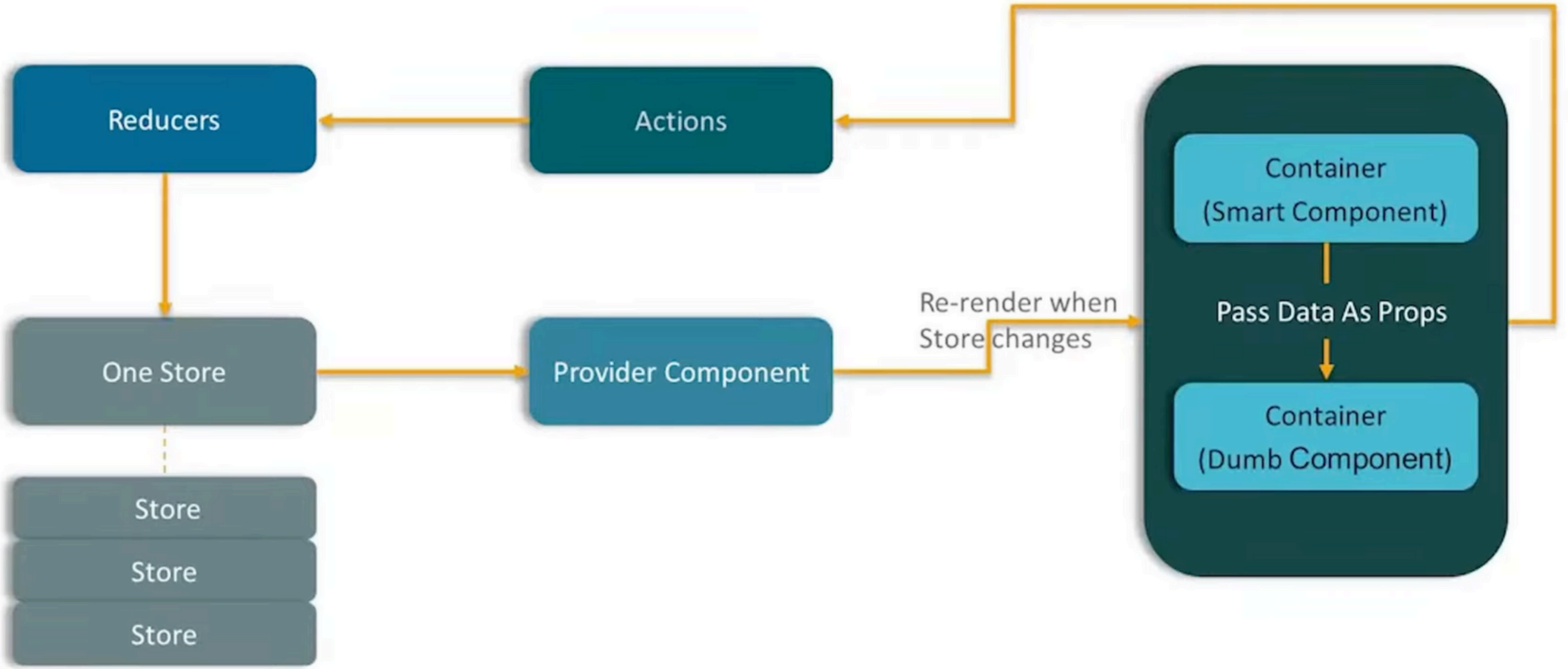
Things you should **NEVER** do inside a reducer:

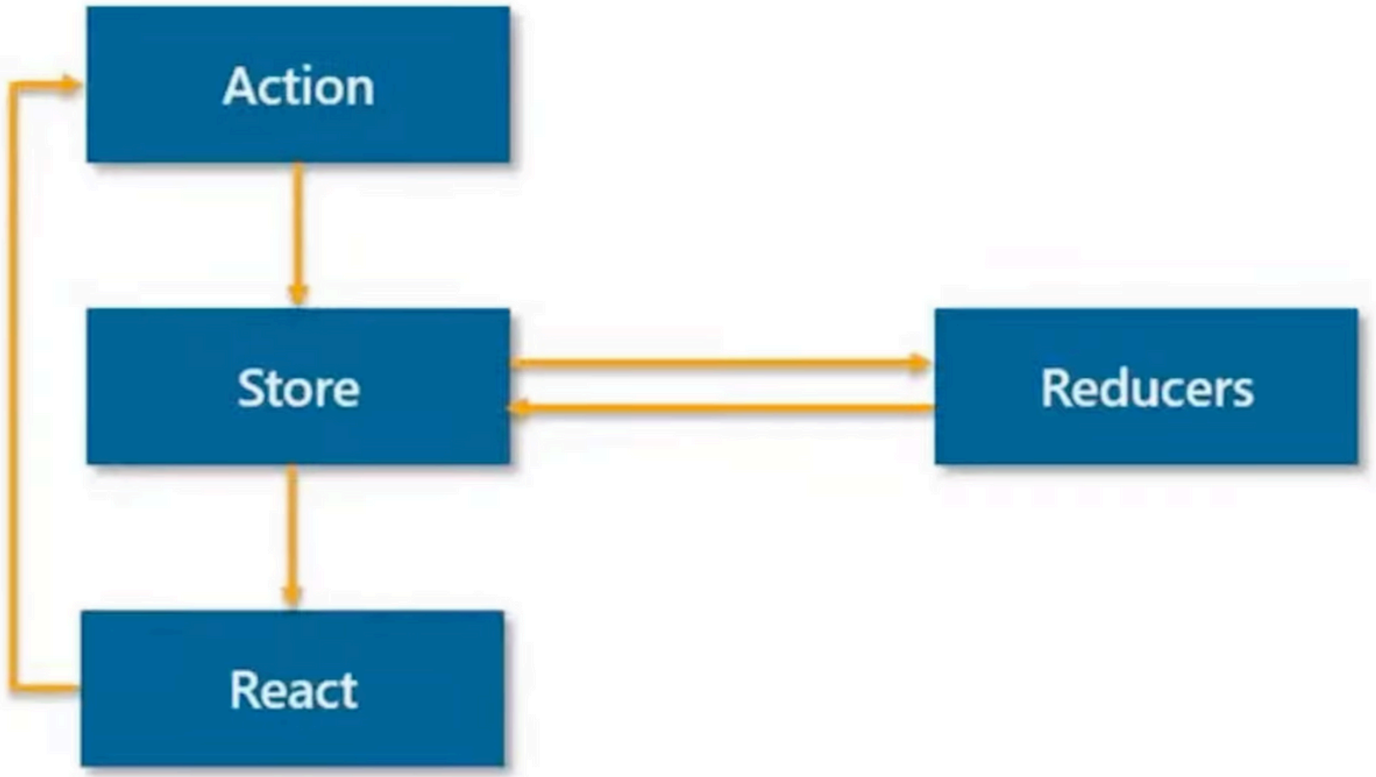
- ✗ Mutate its arguments
- ✗ Perform side effects like API calls and routing transitions
- ✗ Call non-pure functions like `Date.now()` or `Math.random()`

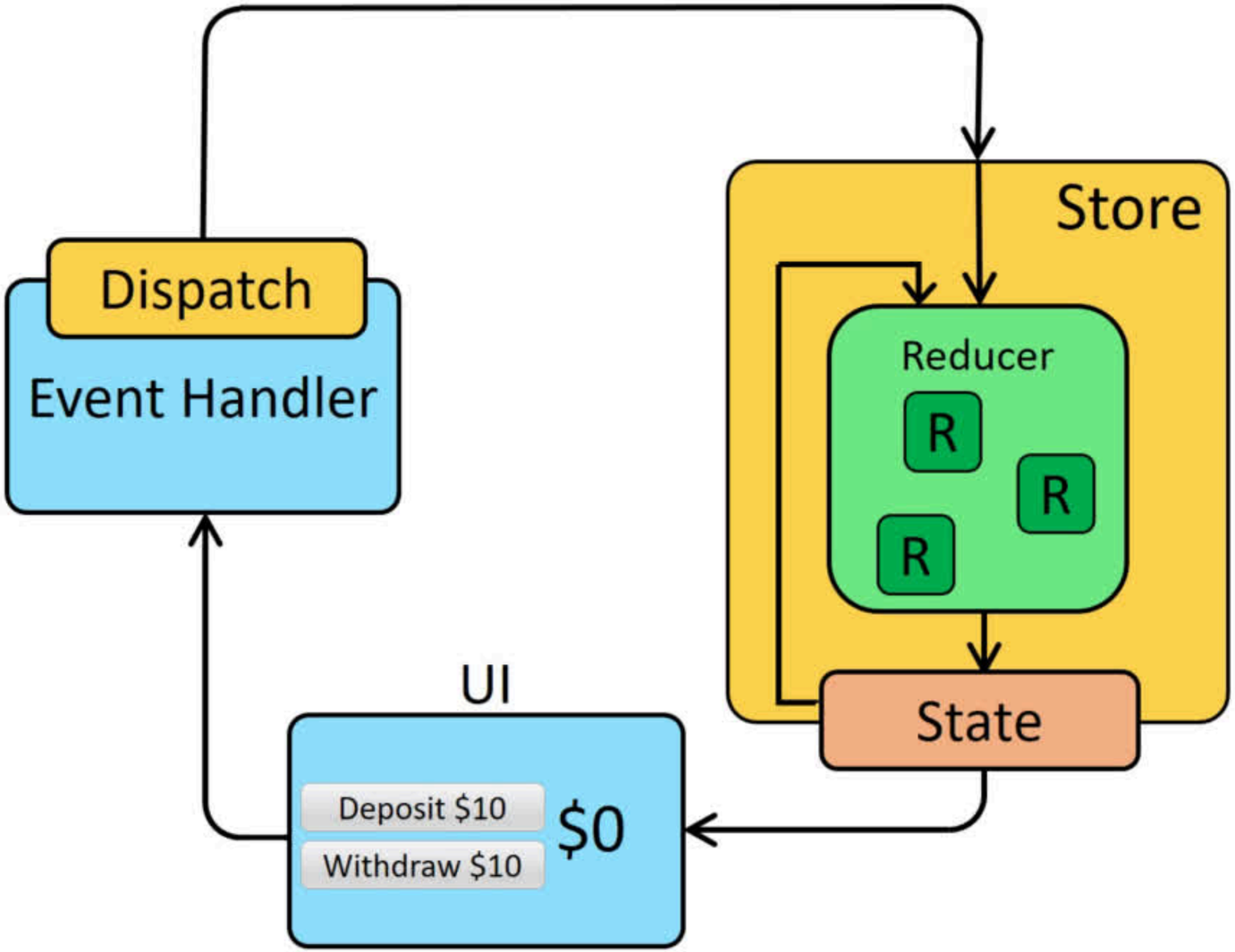
Store

Store is an object which brings all the components to work together. It calculates the state changes and then notifies the root reducer about it.



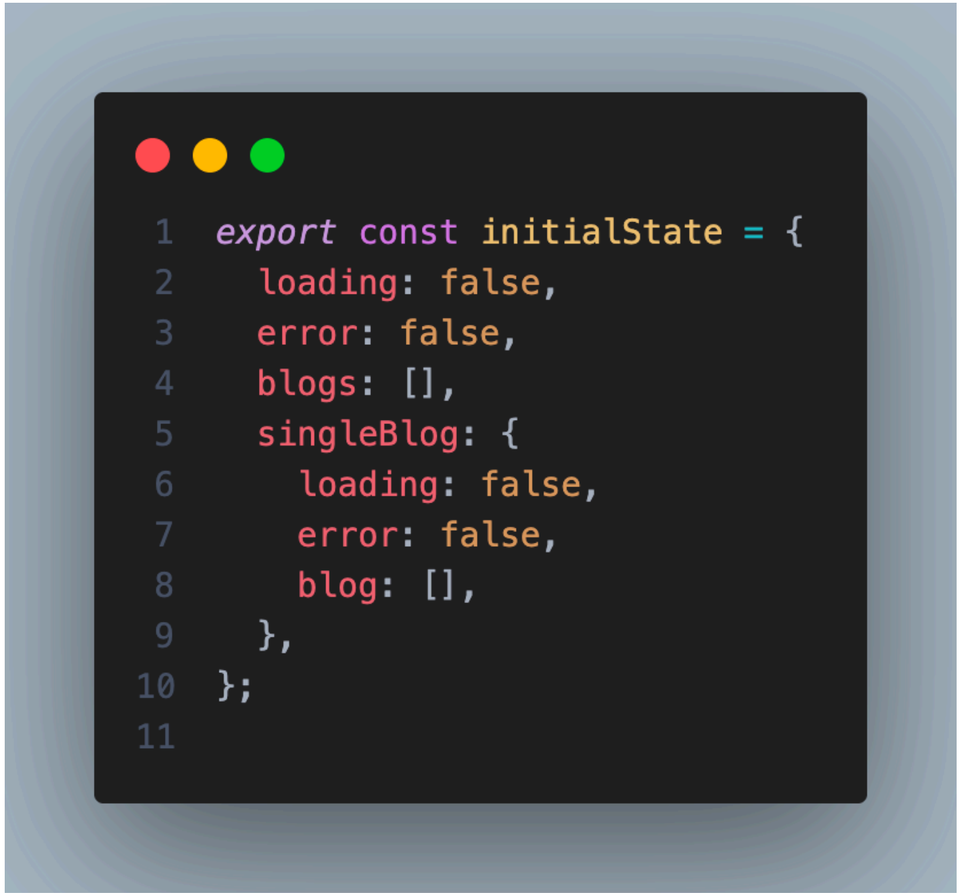

Reducers
Actions
One Store
Provider Component
Re-render when
Store changes
Container
(Smart Component)
Pass Data As Props
Container
(Dumb Component)
Store
Store
Store


Action
Store
Reducers
React


Dispatch
Event Handler
Store
Reducer
R
R
R
State
UI
Deposit $10
Withdraw $10
$0

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Router from './route';

// REDUX
import { Provider } from 'react-redux';
import store from './redux/store';
import './index.css';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Provider store={store()}>
    <Router />
  </Provider>
);

```

```
import { legacy_createStore as createStore } from 'redux';

import rootReducer from './reducers';

export default function configureStore(preloadedState) {
  const store = createStore(rootReducer, preloadedState);

  return store;
}
```


export const initialState = {
loading: false,
error: false,
blogs: [],
singleBlog: {
loading: false,
error: false,
blog: [],
},
};

```
import { initialState } from './initialState';

const reducer = (state = initialState, action) => {
  console.log('reducer');
  switch (action.type) {
    case 'CHECK':
      return {
        ...state,
        data: action.payload,
      };
    default:
      return state;
  }
};

export default reducer;

```

```
import { combineReducers } from 'redux';

const rootReducer = combineReducers({})

export default rootReducer
```