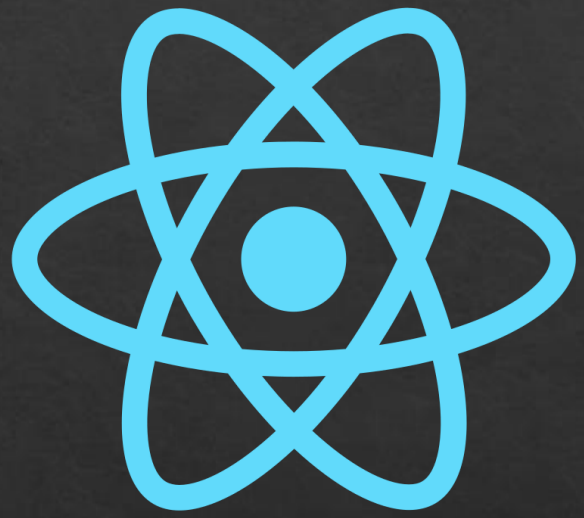
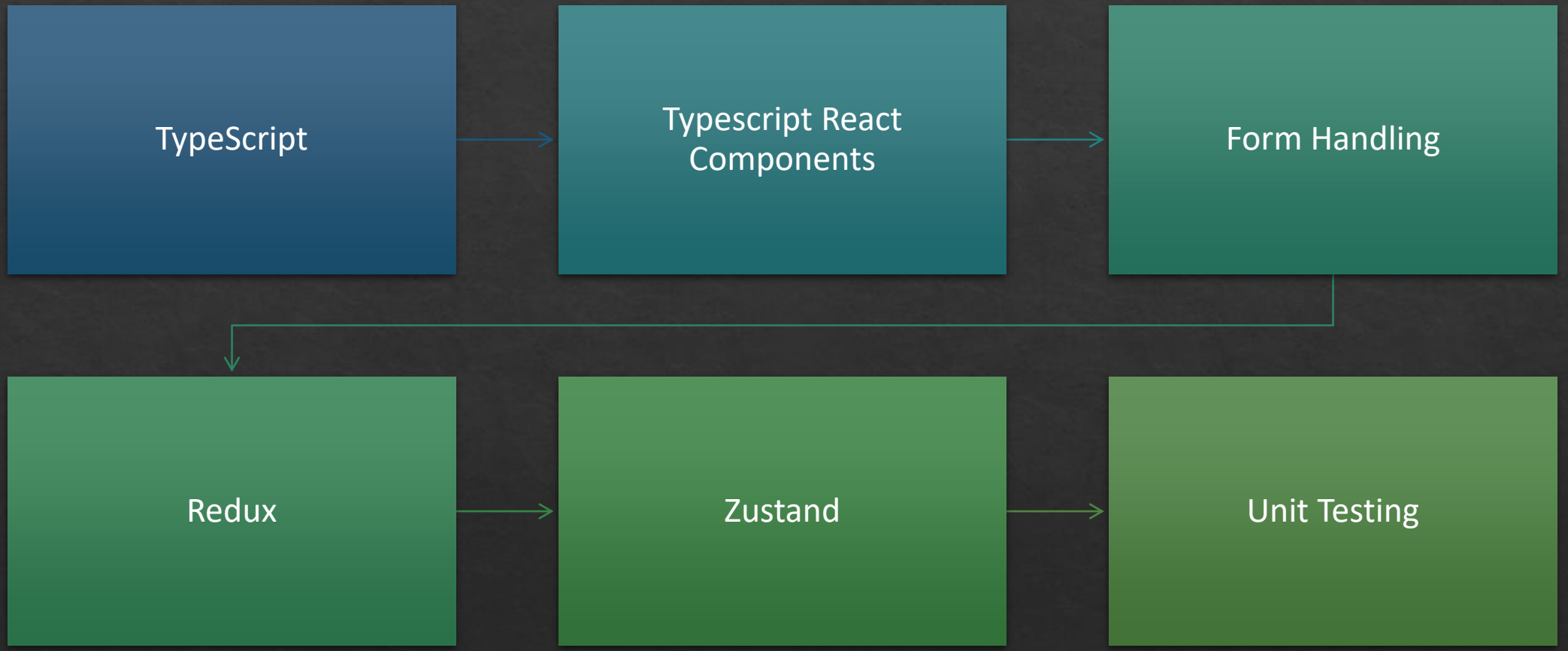
# React



ANIL JOSEPH

# Agenda


TypeScript
Typescript React Components
Form Handling
Redux
Zustand
Unit Testing

Anil Joseph

# Introduction

- Over 20 years of experience in the industry
- Technologies
  - C, C++
  - Java, Enterprise Java
  - .NET & .NET Core
  - **UI Technologies: React, Angular, jQuery, ExtJs**
  - Mobile: Native Android, React Native
- Worked on numerous projects
- Conducting trainings for corporates (700+)

# Software

Node.js & NPM

HTML, CSS, JavaScript, TypeScript Editor

**(Visual Studio Code)**

Browsers(Chromium)

# TypeScript

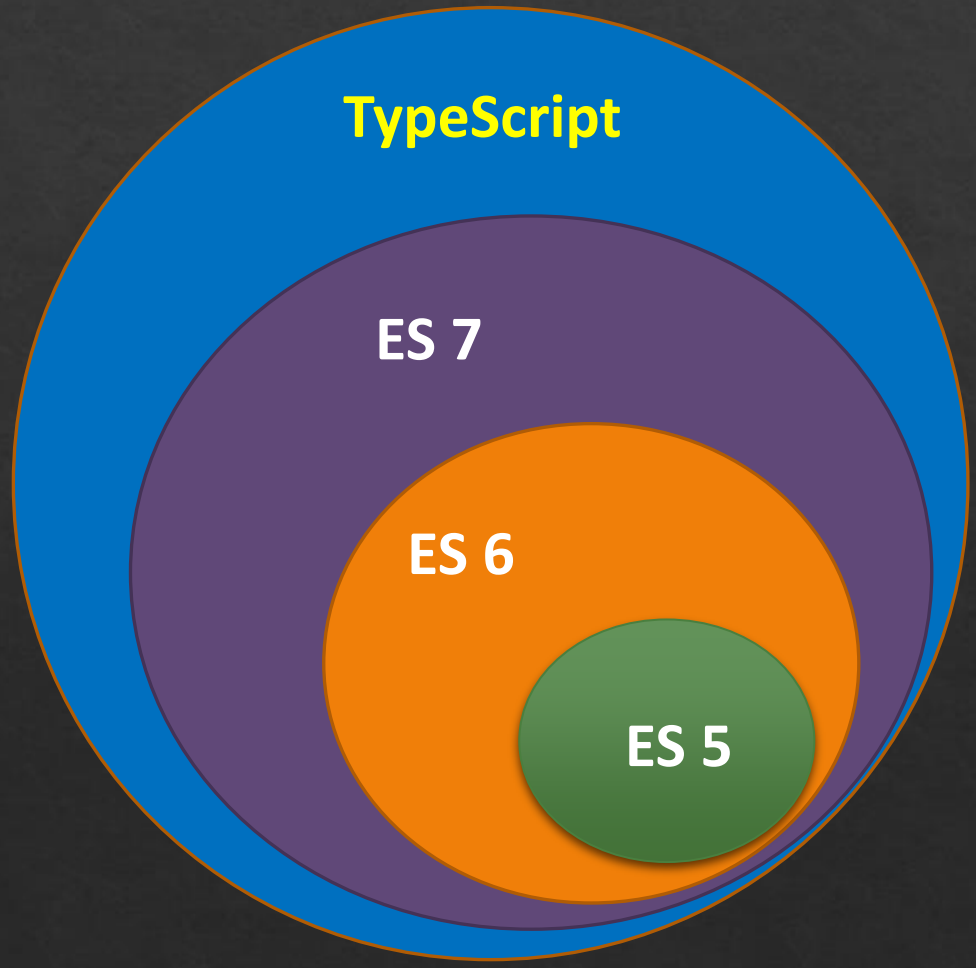
TypeScript is programming language developed and maintained by Microsoft.

TypeScript is a typed superset of JavaScript.

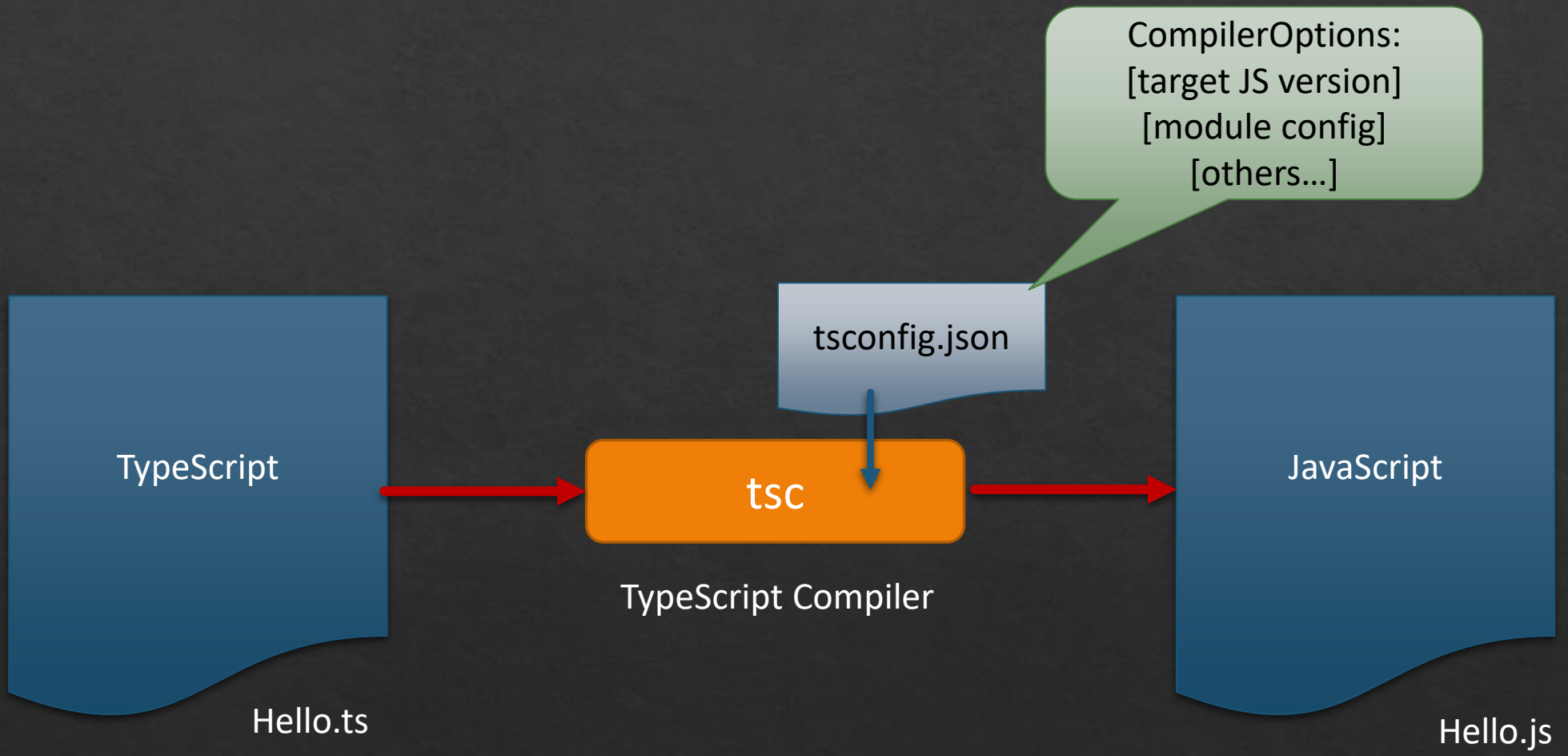
***Transcompiles*** to JavaScript.

Designed for development of large applications.

Open Source.



# TypeScript


CompilerOptions:
[target JS version]
[module config]
[others…]
tsconfig.json
TypeScript
tsc
JavaScript
TypeScript Compiler
Hello.ts
Hello.js

# TypeScript Features

- Type Annotations
- Compile-Time Type Checking
- Type Inference
- Interfaces
- Classes and Inheritance
- Namespaces and Modules
- Generics
- Decorators
- Arrow Functions

# TypeScript Installation

- Install NodeJs and NPM
- Run the command **npm install -g typescript**

# TypeScript Types

## Boolean

- **let** isAvailable:boolean = false

## Number

- **let** age: number = 16;

## String

- **let** name: string = “Anil";

## Array

- **let** list: number[] = [1, 2, 3];
- **let** list: Array<number> = [1, 2, 3];

# TypeScript Types

## Enum

- **enum** Color {Red, Green, Blue}
- **let** c: Color = Color.Green;

## Any

- **let** x: any = 4;
- x = “hello”

## void

```
function foo(): void {
  console.log(“foo");
}
```

## null and undefined

- var x: string = null;
- var y:string= undefined

# Defining New Types

- Type alias
  - Used to give a type a new name. It's like creating a shorthand for a more complex type.
- Interfaces
  - Used to define the shape of an object or the signature of a function. They are more extensible than type aliases because they can be reopened to add new properties.
- Classes
  - Define both the shape and the behavior of objects. A class can implement interfaces.
- Enums
  - Allow you to define a set of named constants. Using enums can make it easier to document intent, or create a set of distinct cases.

# Type aliases

- A type alias is declared using the ***type*** keyword followed by an identifier and a type annotation. Once defined, the alias can be used anywhere a type can be used.
- Flexibility: Type aliases can represent primitive types, unions, intersections, and any other valid TypeScript types.
- Readability: Using type aliases can make complex type definitions easier to work with and understand.

# Interfaces

- Interfaces are a powerful way of defining contracts.
- Example

```
interface Vehicle{

    name: string;
    speed: number;
    gear?: number;

    applyBrakes(decrement: number): void;
}
```

- Interfaces can extend interfaces
  - (keyword extends)
- Classes implements interfaces
  - (keyword implements)

# Classes

- Traditionally JavaScript uses functions and prototype-based inheritance to build up reusable components.
- Starting with ECMAScript 2015(ES6), JavaScript introduced the object-oriented class-based approach.
- Typescript supports classes that compile down JavaScript.
  - Works across all major browsers and platforms, without having to wait for the next version of the browser.

# Classes

- Access Modifiers
  - public, private, protected
- Constructors
- Properties
- Static Members
- Inheritance
- Abstract classes and methods

# Arrow Functions

Represents a function expression

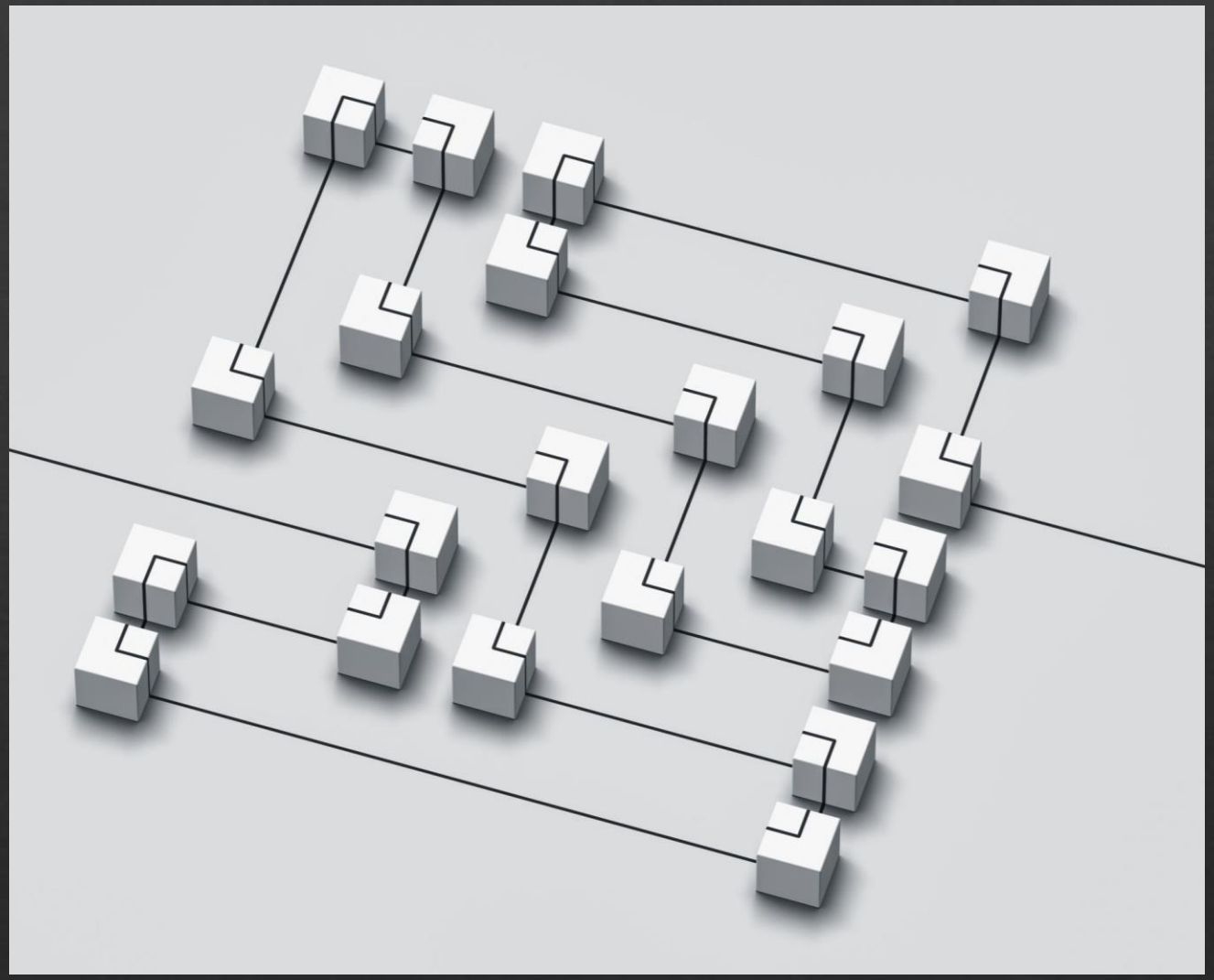
An arrow function expression has a shorter syntax than a function expression.

They do not receive the implicit arguments “this” and “arguments”.

Used widely for asynchronous and functional programming

# TypeScript Modules

- Starting with ECMAScript 2015(ES6), JavaScript has the concept of modules.
- Modules have a scope of their own
- In the module system every JS file is a module and all declarations in the file is scoped to that module.
- The same concept is shared in TypeScript



# Modules

- Use the import and export statements.

```
let foo = function(){
   //some code
}


export default foo;
```

one.js

```
import foo from ‘./one’;

foo();
```

two.js

```
import bar from ‘./one’;

bar();
```

three.js

# Modules

```
export let foo = function(){
   //some code
}

 export let bar = function(){
    //some code
 }
```

one.js

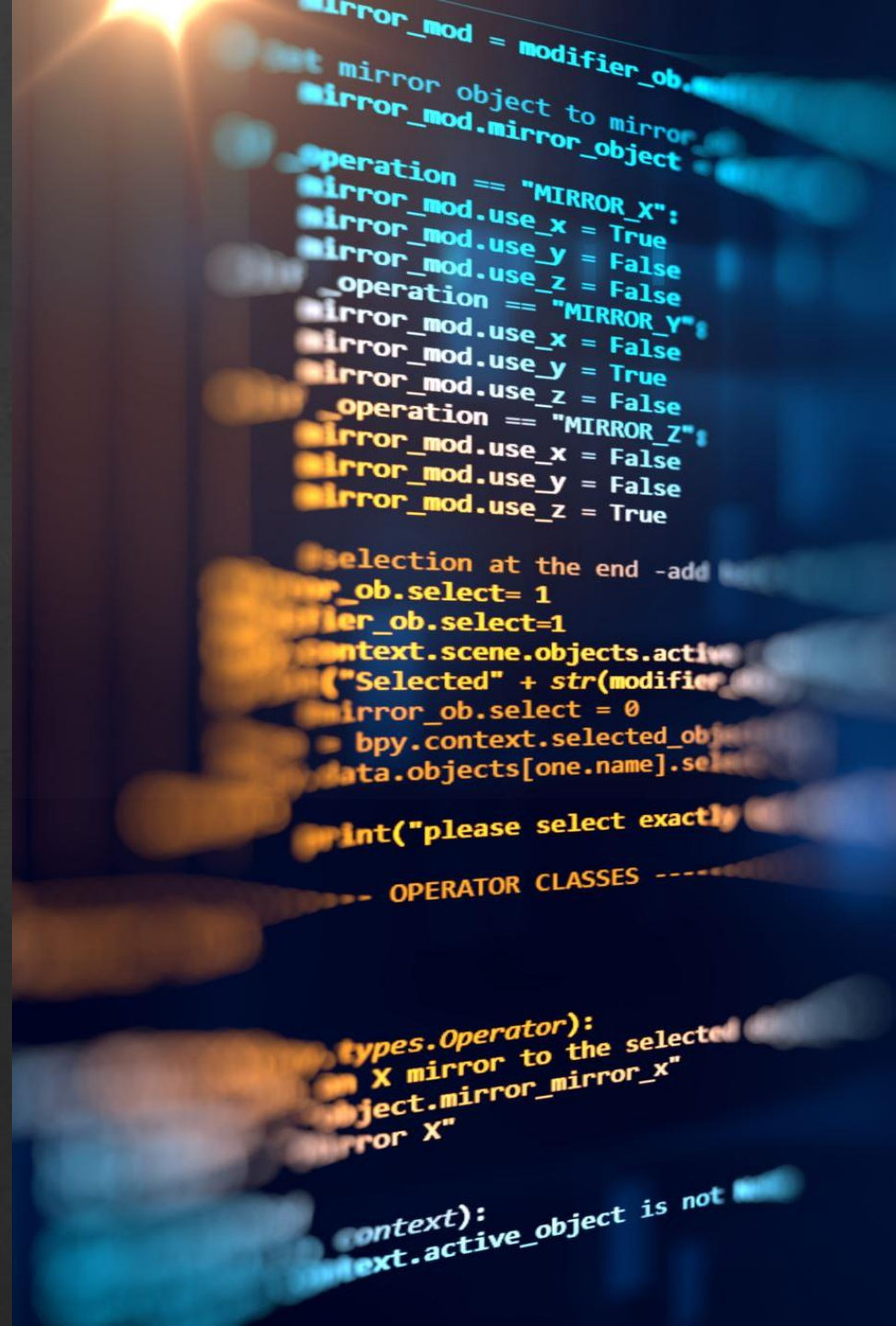
```
import {foo, bar} from ‘./one’;


foo();
bar();
```

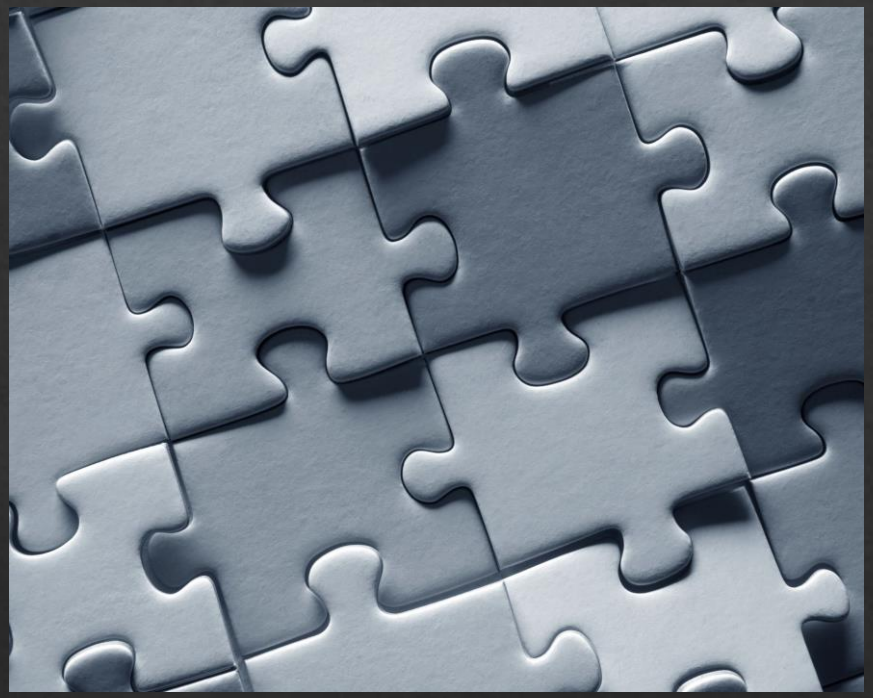
two.js

# Tool Chains

- A toolchain is a set of programming tools that is used to perform a complex software development task or to create a software product.(Wiki)
- Using a toolchain provides a better developer experience
- Advantages of Tool Chains
  - Scaling to many files and components.
  - Using third-party libraries from npm.
  - Detecting common mistakes early.
  - Live-editing CSS and JS in development.
  - Optimizing the output for production.



# Create React App



- Create React App is an officially supported way to ***create single-page React applications***.
- It offers a modern build setup with no configuration.
- Sets up the development environment to use the latest JavaScript features.
- Optimizes the application for production.
- Integrates the tools: NPM, Babel, Webpack, Webpack development server

# Create Project

1

## Install NodeJs

- Runtime to run/execute JavaScript on the machine/server
- This installation also installs npm(Node Package Manager)

2

## Create React Application

- **npx create-react-app the-react-app**

3

## Start the Application

- **cd the-react-app**
- **npm start**

# Create Project-Typescript

1

## Install NodeJs

- Runtime to run/execute JavaScript on the machine/server
- This installation also installs npm(Node Package Manager)

2

## Create React Application

- **npx create-react-app the-react-app –template typescript**

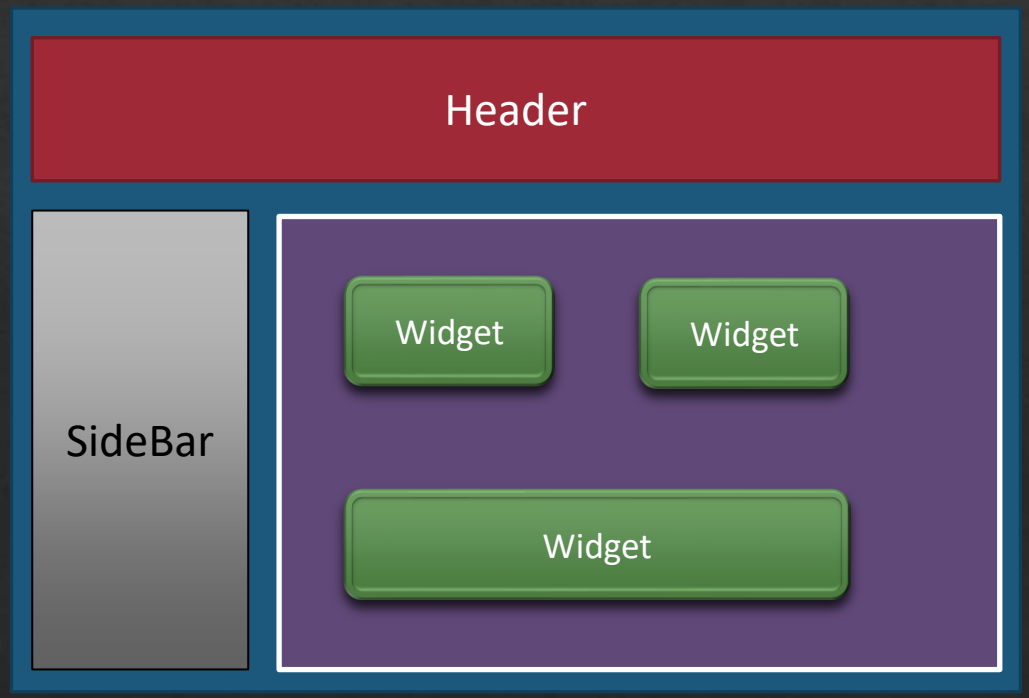
3

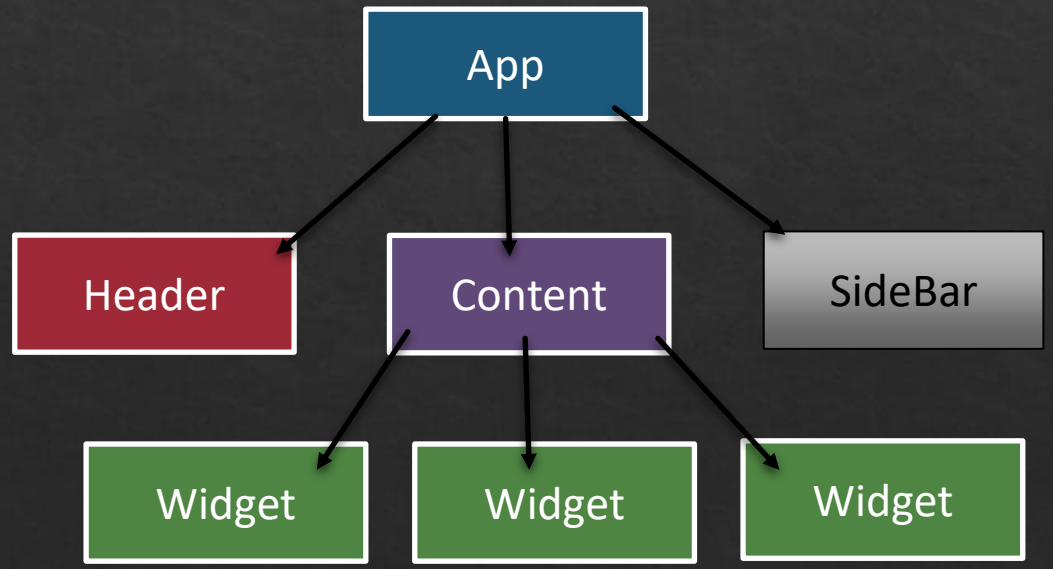
## Start the Application

- **cd the-react-app**
- **npm start**

# React Components

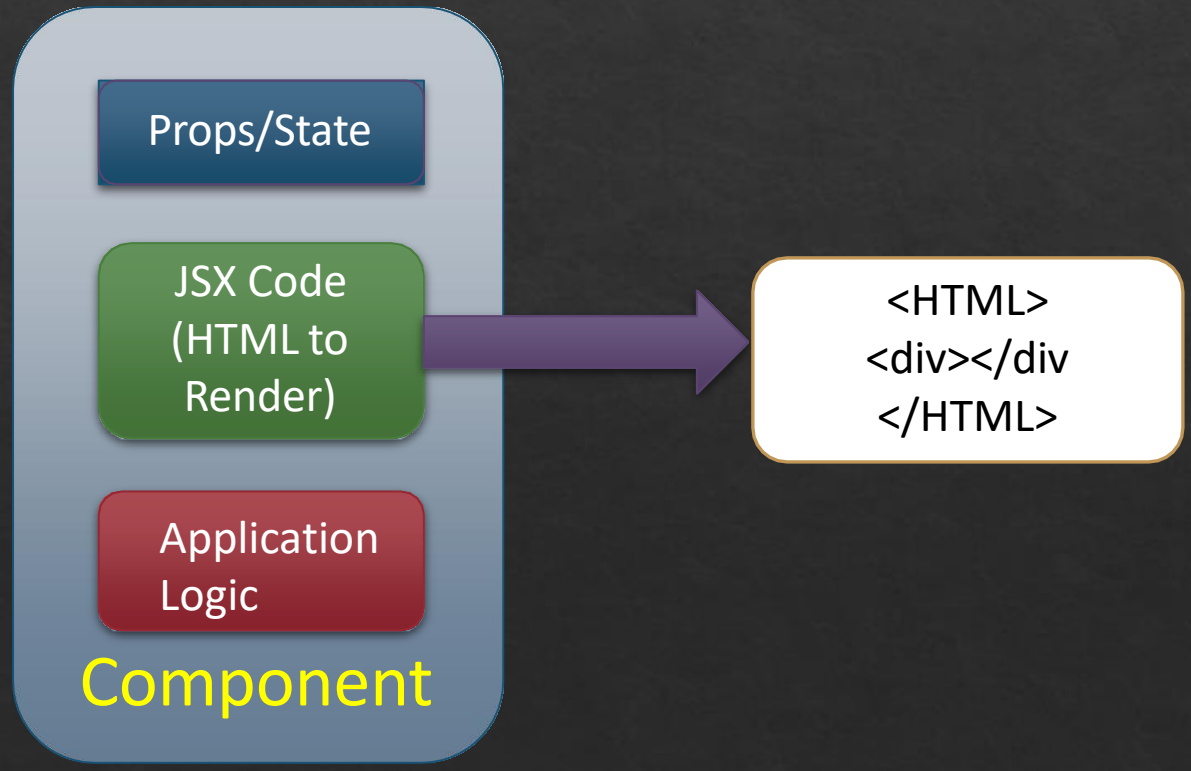
- Components are the core building blocks in React
- Creating a React applications is all about designing and implementing components
- A React application can be depicted as a component tree.



# React Components

- The UI in a React Application is composed of components, the building blocks.
- Components are designed to be reusable



# Types of Components

## Functional

- Presentational
- Stateless (till React 16.8)
- Stateful (using React Hooks)

## Class based

- Containers
- Stateful

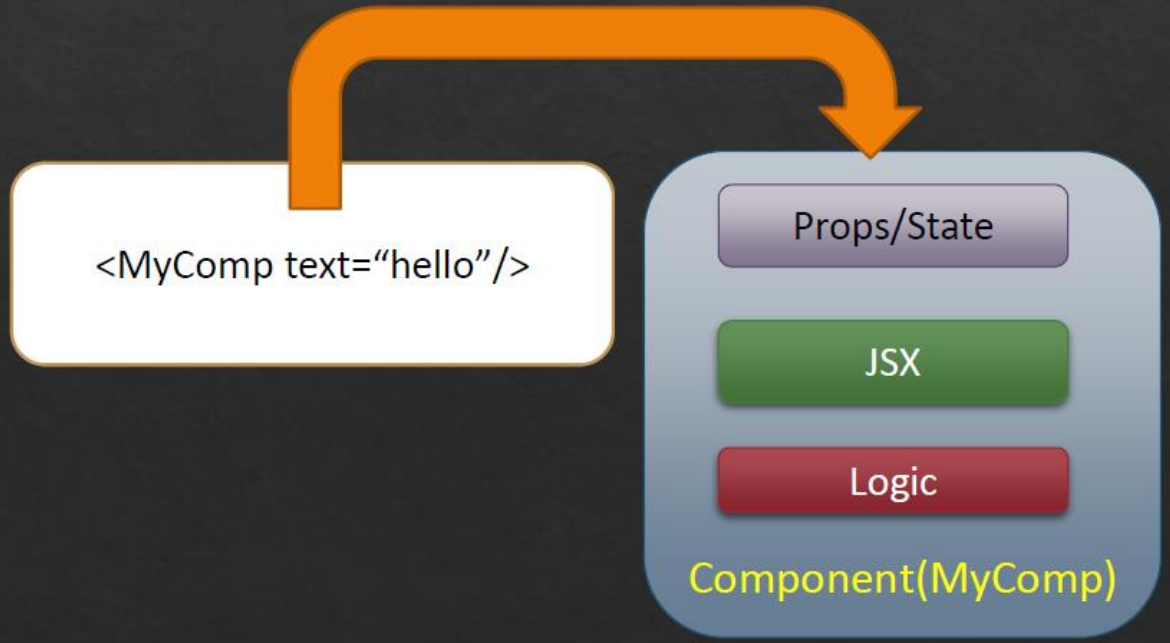
# Components: Properties(props)

Most components can be customized with different parameters when they are created.

These creation parameters are called ***"props"***.

***Props*** are used similar to HTML attributes.

***Changes to props will automatically re-render the component.***



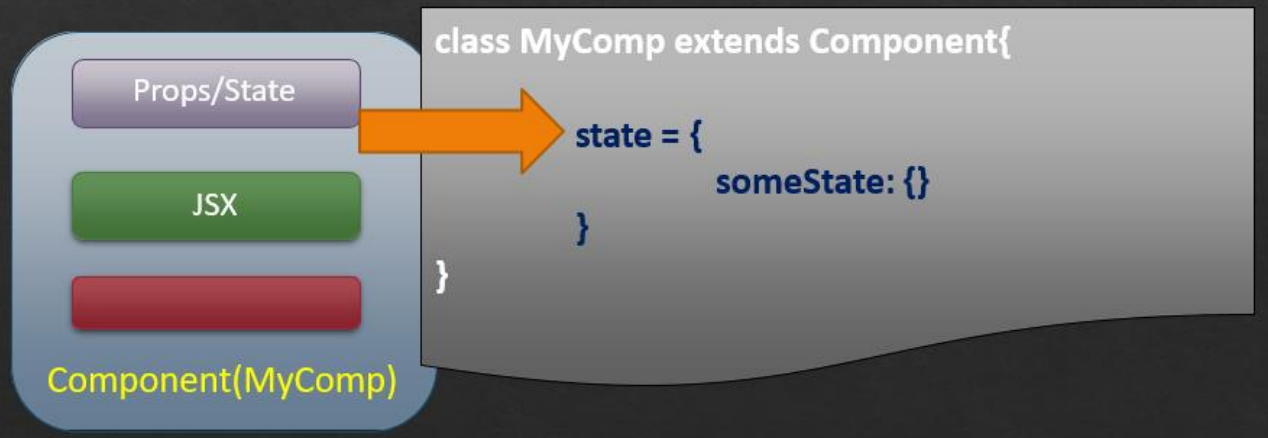
# Component State

State holds information about the component

State is used when a component needs to keep track of information between renderings.

State is created and initialized in the component itself.

**State updates trigger a rerender of the component.**



# Props and State

"props" and "state" are CORE concepts of React.

Only changes in "props" and/ or "state" trigger React to re-render the components and potentially update the DOM in the browser

props allow you to pass data from a parent (wrapping) component to a child (embedded) component.

State is used to change the component from within.

# React Hooks

React hooks was introduced in version 16.8

Many React features like state, lifecycle hooks etc. were available only with class-based components prior to 16.8.

Hooks let us use state and other React features in a functional component.

React team recommends the functional components over class-based since they can be highly optimized by the tools.

# React Hooks

## State Hooks(useState)

- Equivalent of state in class-based components

## Effect Hooks(useEffect)

- Used to perform side-effects in a function
- component
- Equivalent to lifecycle hooks in class-based components

## Context Hooks(useContext)

- Used to access the React Context;

# React Hooks

### Callback Hooks(useCallback)

- Returns a memoized callback.
- Used to optimize the components

### Memo Hooks(useMemo)

- Returns a memoized value.
- Used to optimize the components

### Ref Hooks(useRef)

- Returns a mutable ref object

### Custom Hooks

- A functions
- Uses Other Hooks

# State Management

**React Context**

- Available from React 16.3

**React Redux**

- Library to manage state

**Zustand**

- Library to manage state

**RxJs**

- Reactive Programming  using Observables

# Types of State

## Local UI State

- Show/Hide UI
- Handled By the Component

## Persistent State

- Orders, Blogs
- Stored on Server, can be managed by Redux or React Context

## Client State

- IsAuntenticated, Filter Information
- Managed by Redux or React Context

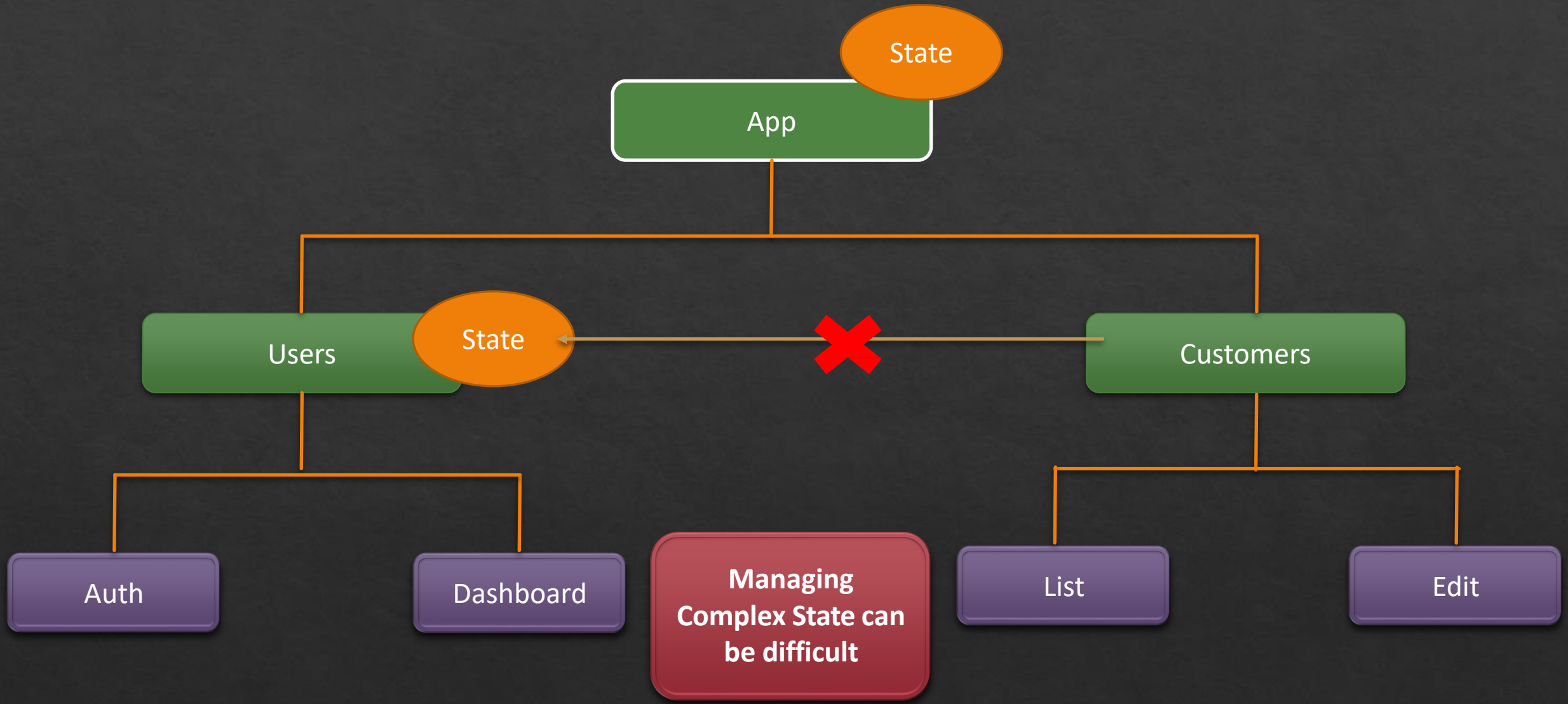
# Redux

Redux is an open-source JavaScript library designed for managing application state.

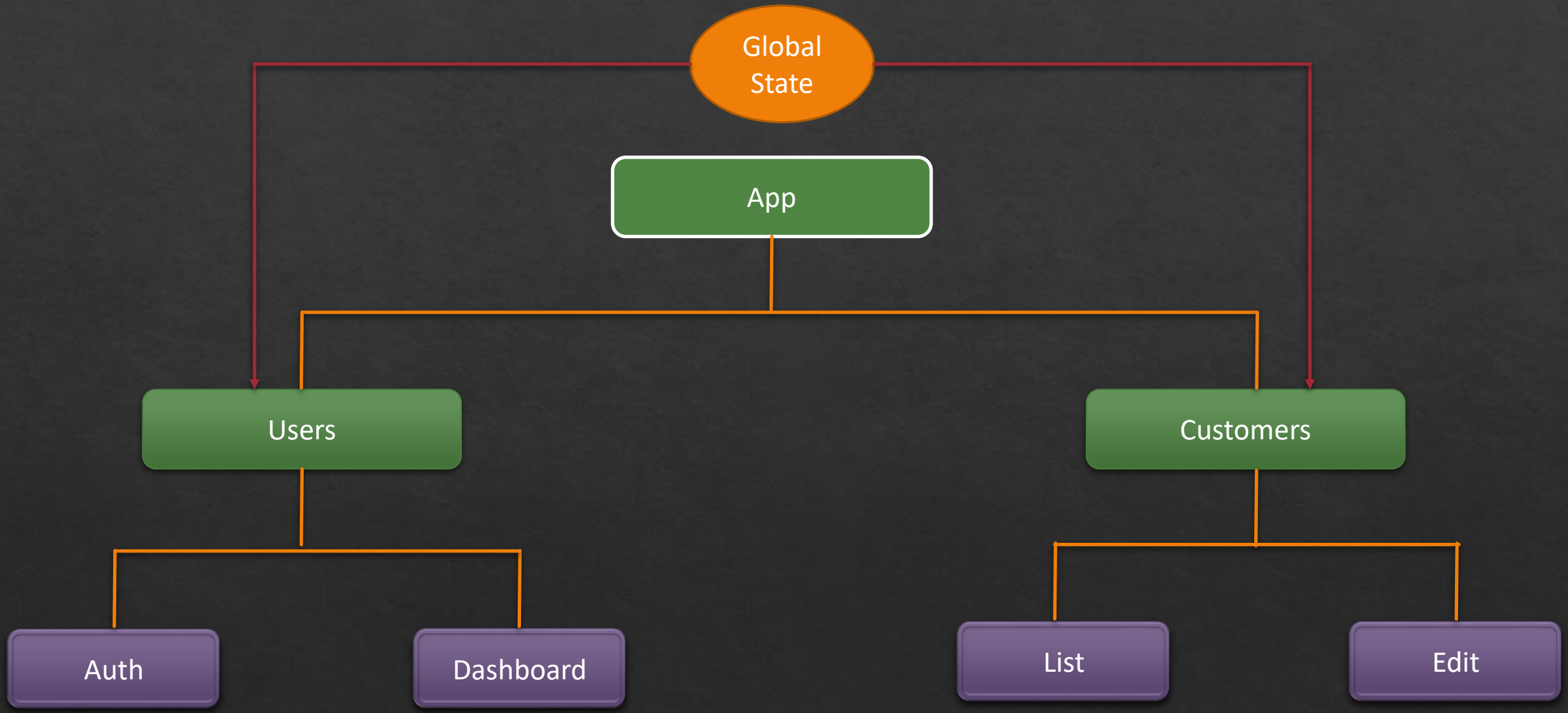
It is primarily used together with React or Angular for building user interfaces.

Redux was built on top of functional programming concepts.

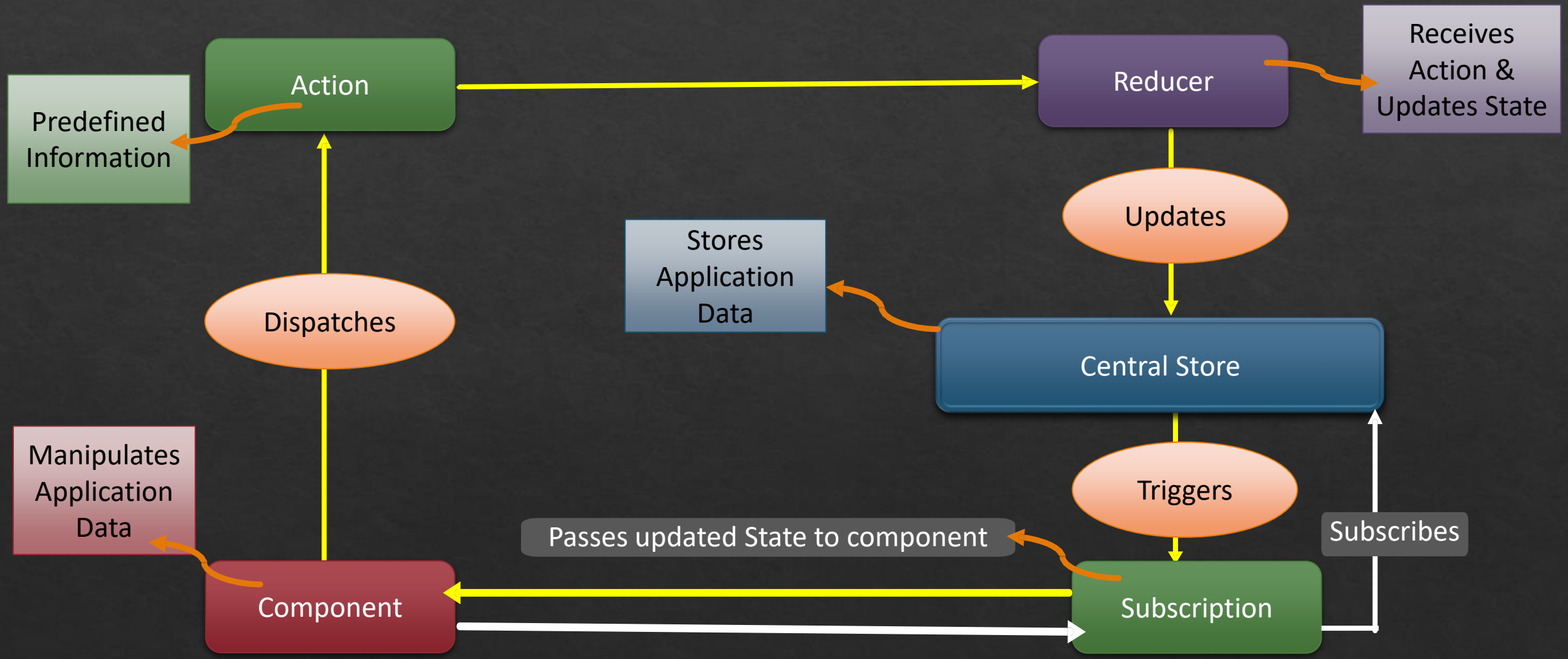
# Why Redux?


State
App
Users
State
Customers
Auth
Dashboard
Managing Complex State can be difficult
List
Edit

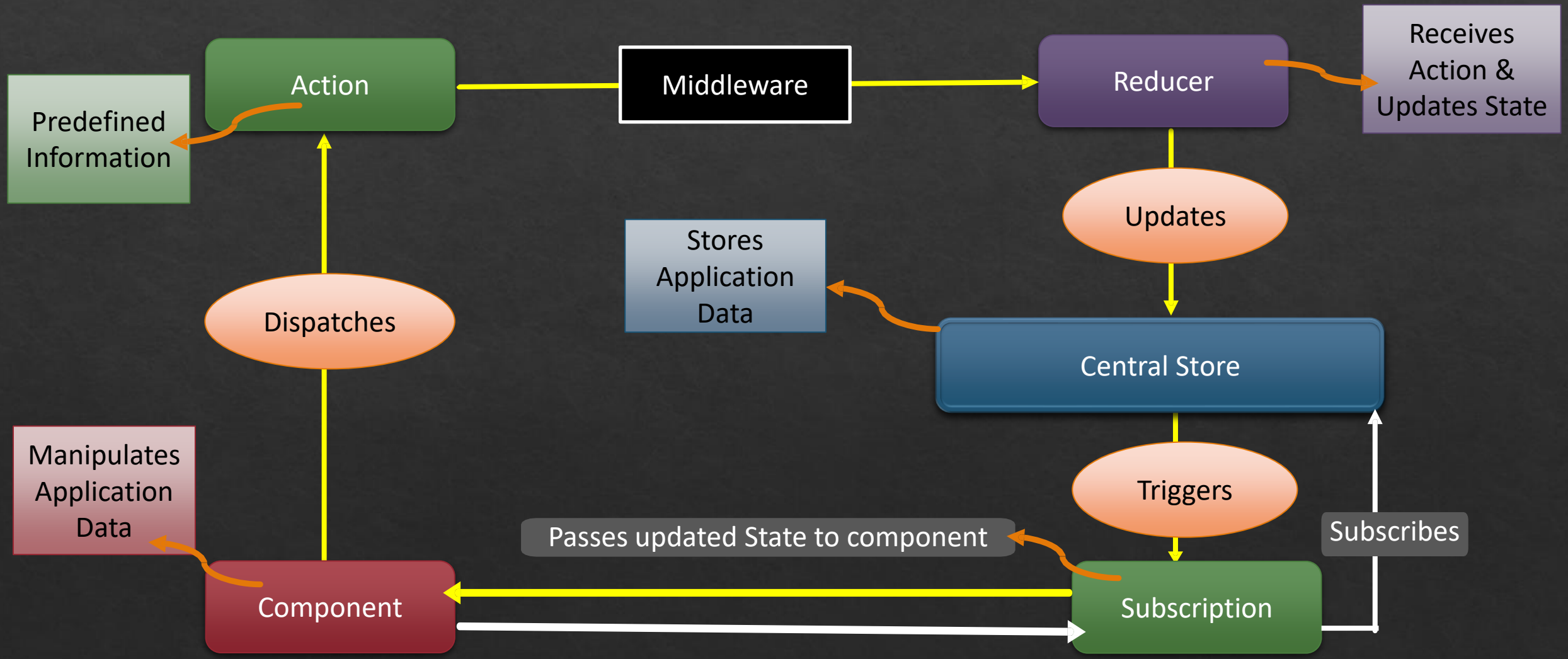
# Why Redux?



# Redux Flow

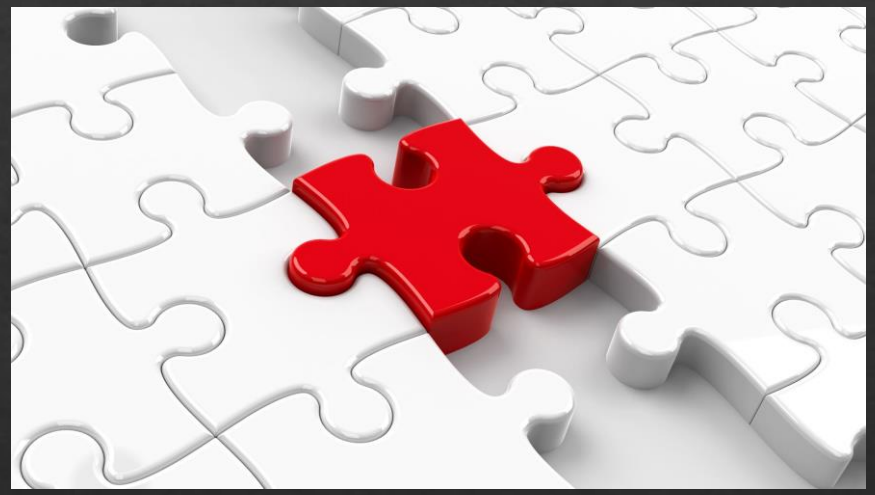

Action
Predefined Information
Reducer
Receives Action & Updates State
Updates
Stores Application Data
Dispatches
Central Store
Manipulates Application Data
Triggers
Passes updated State to component
Subscribes
Component
Subscription

# Redux Flow


Action
Middleware
Reducer
Receives Action & Updates State
Predefined Information
Updates
Stores Application Data
Dispatches
Central Store
Manipulates Application Data
Triggers
Passes updated State to component
Subscribes
Component
Subscription

# React Redux

- Redux react is a library that integrates Redux to a React  Application
- Comprises of Components & Functions
- Installation
  - npm install react-redux



# Zustand

- Minimalist API: Zustand provides a very simple and straightforward API that makes setting up and using your state management straightforward.
- Hooks-based: It utilizes React hooks, making it easy to integrate and use within functional React components.
- No boilerplate: Unlike some other state management libraries, Zustand requires very little boilerplate code, making your codebase cleaner and easier to maintain.
- Centralized Stores: You can create multiple stores if needed, but typically a single store is used, which can be divided into slices of state.

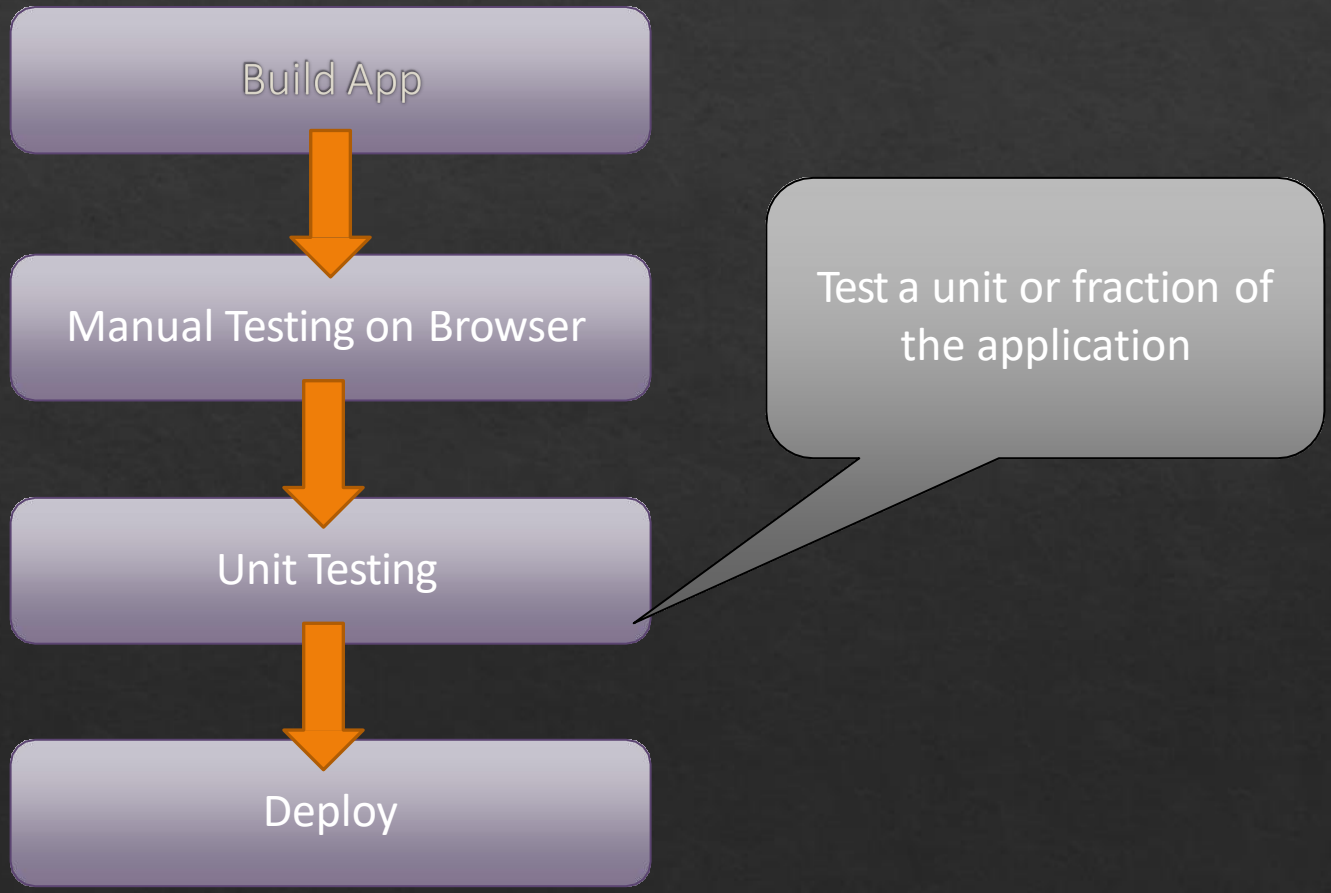
# Zustand

- Immutable Updates: Zustand enforces immutable update patterns which is beneficial for predictable state management.
- Devtools Support: It supports Redux DevTools, allowing you to inspect state changes and actions for debugging purposes.
- Flexible: It works with both React and non-React applications, offering flexibility across different setups.
- Optimized for Performance: Zustand only causes re-renders in components that specifically subscribe to pieces of the state, making it very performance efficient.

# React Context

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.
- Example of props to be passed down
  - Theme
  - Locale
  - Authenticated user
- API
  - React.createContext

# Testing


Build App
Manual Testing on Browser
Unit Testing
Deploy
Test a unit or fraction of the application

# Testing Tools

Testing API
&
Test Runner

Write the Unit Test.
Executes the Unit Tests.

Jest
(Built over Jasmine)

Test Utilities

Simulate the React App

react-test-renderer

enzyme

testing-library/react

# Jest

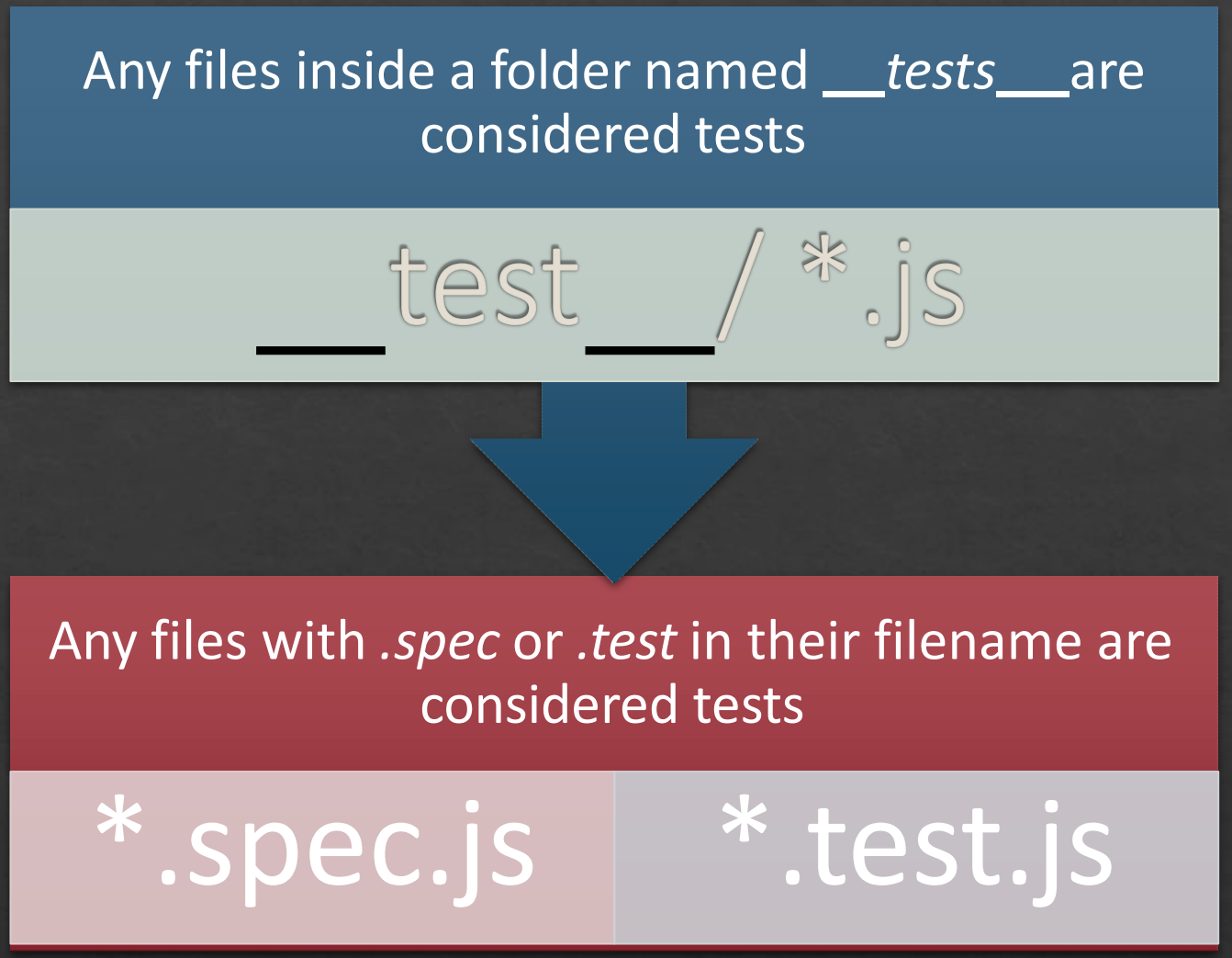
A testing library and test runner with handy features

Created by the members of the React team and the recommended tool for unit testing react

Built on top of Jasmine/Mocha

Additional features like mocking and snapshot testing

# Jest: Identifying Test files



# Jest Global Methods

## it

- Method which you pass a function to, that function is executed as block of tests by the test runner.
- Alias name: test

## describe

- An optional method for grouping any number of *it* or *test* statements
- Alias name: suite

# Setup & Teardown Global functions

| | |
|---|---|
| beforeEach | BeforeEach runs a block of code before each test |
| afterEach | Runs a block of code after each test |
| beforeAll | BeforeAll runs code just once, before the first test |
| afterAll | Runs a block of code after the last test) |

# What to Test?

Test Isolated Components

Test Conditional Outputs

Don’t Test Library Functions

Don’t Test Complex Connection

# Thank You

ANIL JOSEPH

anil.jos@gmail.com