REACT

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

WHAT IS REACT?

React is a JavaScript library used to build the user interface for web applications

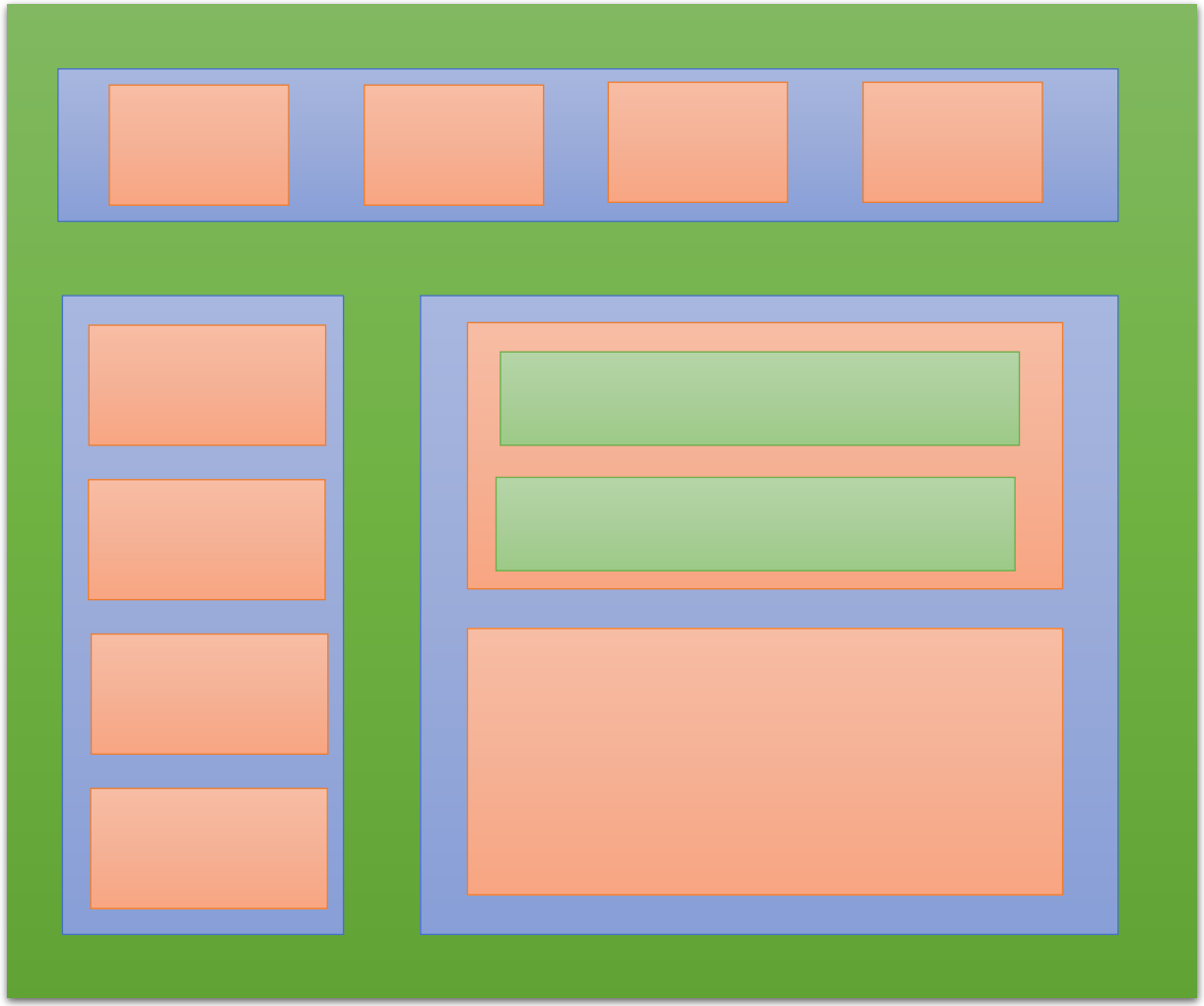
React was developed and maintained by the folks at Facebook

An open source project with an active developer community.

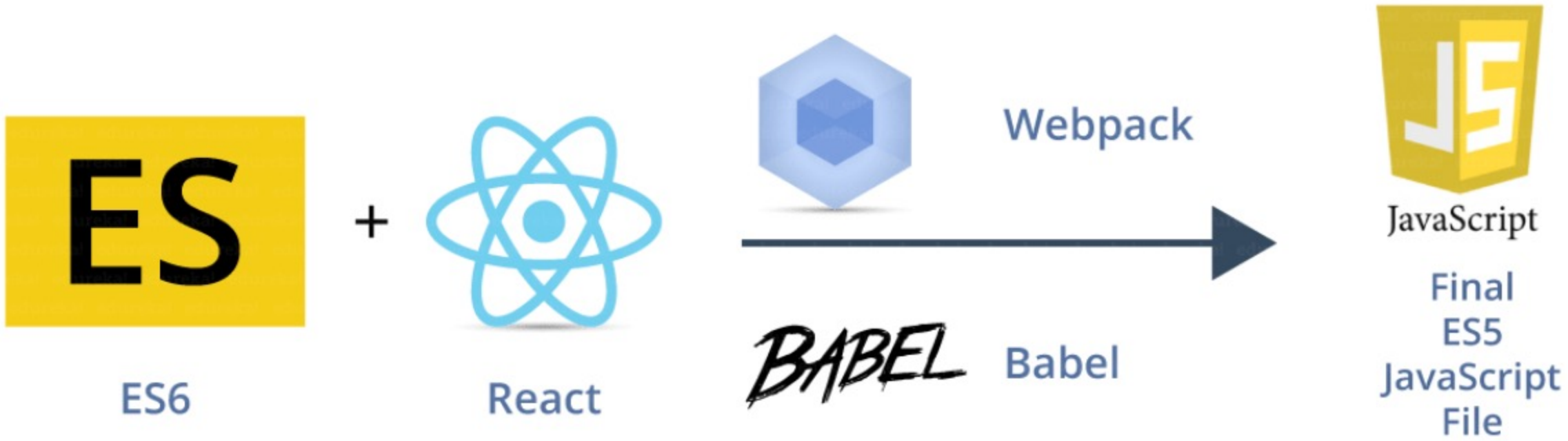
Write less and do more!!

MORE ABOUT REACT APP

In a React application, you should **break down** your site, page or feature **into smaller pieces** of components. It means that your site will be built by the combination of different components. These components are also built on the top of other components and so on.



KEY TERMINOLOGIES



KEY TERMINOLOGIES

JSX (JavaScript Extension)	JSX Allows us to include ‘HTML’ in the same file along with ‘JavaScript’ (HTML+JS=JSX). Each component in React generates some HTML which is rendered by the DOM
ES6 (ES2015)	The sixth version of JavaScript is standardized by ECMA International in 2015. Hence the language is referred to as ECMAScript.
ES5(ES2009)	The fifth JavaScript version and is widely accepted by all modern browsers,
Webpack	A module bundler which generates a build file joining all the dependencies
Babel	This is the tool used to convert ES6 to ES5. This is done because not all web browsers can render React (ES6+JSX) directly.

WHY REACT?

Easy to understand for developers with the knowledge of XML/HTML

UI state becomes difficult to handle with Vanilla JavaScript

High Performant Apps

Huge Ecosystem

Active community

Easy to test

REACT INTERNALS

UNDERSTANDING REACT INTERNALLY

REACT INTERNALS - VIRTUAL DOM

Virtual DOM is in memory lightweight representation of actual DOM.

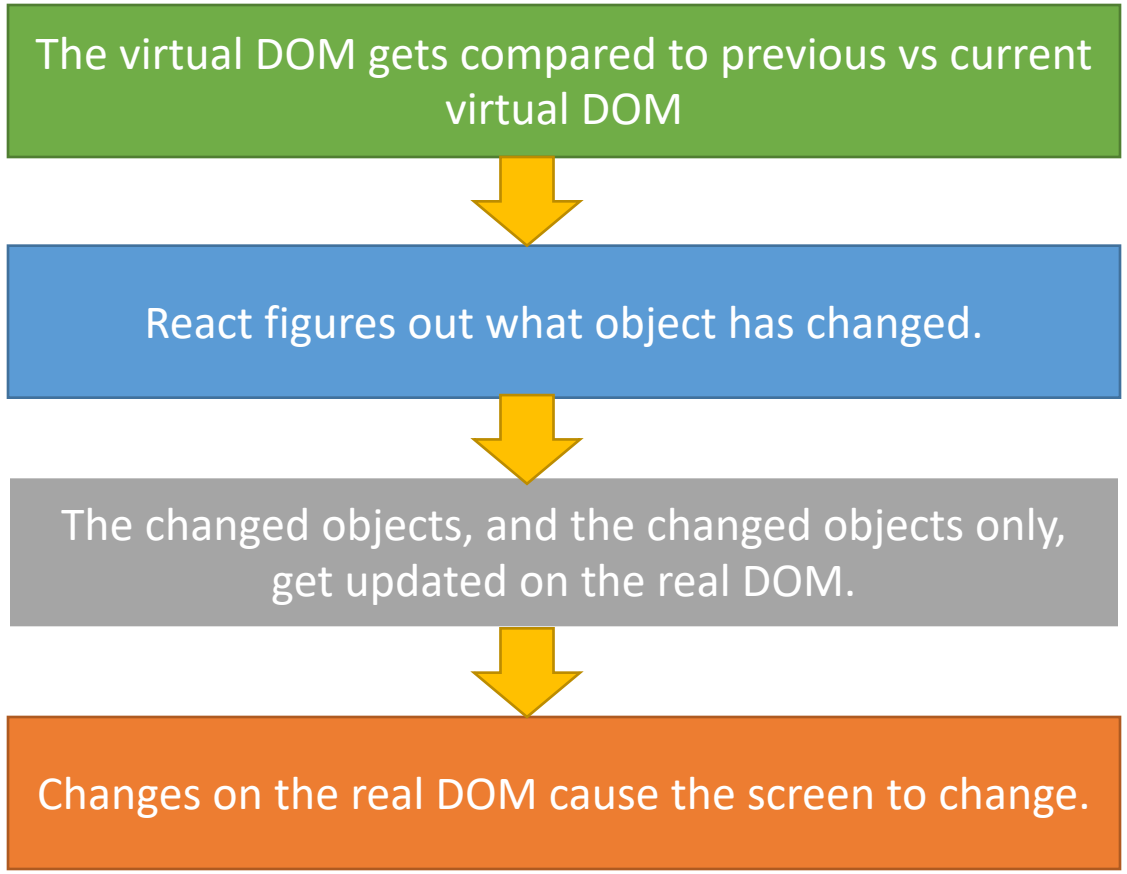
For every DOM object, there is a corresponding Virtual DOM object.

Pure JS intermediate representation.

React never reads from real DOM, only writes to it.

Manipulating the DOM is slow. Manipulating the virtual DOM is much faster, because nothing gets drawn onscreen.

VIRTUAL DOM UPDATE PROCESS



VIRTUAL DOM IN NUTSHELL

To track down model changes and apply them on the DOM *(rendering)* we have to be aware of two important things:

When data has changed?

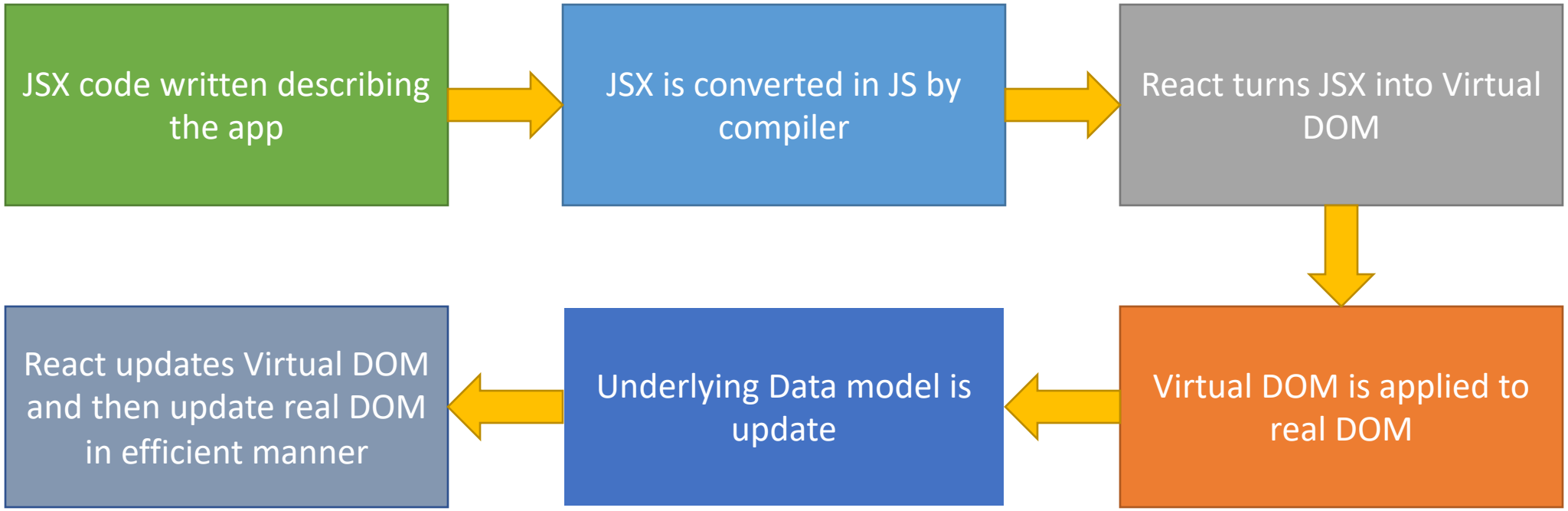
For the change detection, React uses an observer model instead of dirty checking *(continuous model checking for changes)*. That's why it doesn't have to calculate what is changed, it knows immediately.

Which DOM element(s) to be updated?

For the **DOM changing challenge**, React builds the tree representation of the DOM in the memory and calculates which DOM element should change.

React tries to keep as **much DOM elements untouched as possible**. Given the less DOM manipulation can be calculated faster based on the object representation, the costs of the DOM changes are reduced nicely

HOW REACT RENDERS THE VIEW?



COMPONENTS

REUSABLE PIECE OF CODE

COMPONENTS

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

React components are the core building blocks of a React application that represent how a particular element would be visualized on the User Interface.

Combining tens to thousands of components together makes up your application.

A component must be designed in a way that makes it reusable across any page or even projects.

A component may be responsible for performing a single task

A typical React app is a component tree having one root component ("App") and then a potentially infinite amount of nested child components.

CREATE-REACT-APP : THE REACT CLI TOOL

CRA involves configuring a combination of things such as webpack, babel, and react.

To install CRA

- npm install -g create-react-app

To create an App

- create-react-app <app-name>

CRA Commands

- npm run start
- npm run build
- npm run eject

JSX : A SYNTAX EXTENSION TO JAVASCRIPT

JSX is a Markup language used in order to create React elements.

JSX produces React “elements”.

You can put any valid JavaScript expression inside the curly braces in JSX, but not the reserved keywords eg class, for etc.

After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects.

PROPS

A React component is a reusable component which can be used over and over again in the UI, but not always we are going to render the same component with same data. Sometimes we have to change the data or content inside a component. That's why ***props*** are introduced in React.

Props allow you to pass data from a parent component to a child component

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

Props are considered as "immutable"

Props are supplied as attribute to components

STATE

A state in React Component is its own local state, the state cannot be accessed and modified outside the component and can only be used inside the component

React components can be made dynamic by adding state to it.

State is used when component needs to change independently of its parent.

Changes to state also trigger an UI update.

React component's state can be updated using setState()

Best practice : top level components are stateful which keep all interaction logic, manage UI state, and pass the state down to hierarchy to stateless components using props.

MORE ABOUT ONE WAY DATA FLOW

A state is always owned by one Component. Any data that's affected by this state can only affect Components below it: its children.

Changing state on a Component will never affect its parent, or its siblings, or any other Component in the application: just its children.

The state is often moved up in the Component tree, so that it can be shared between components that need to access it.

UNIDIRECTIONAL (ONE WAY) DATA FLOW

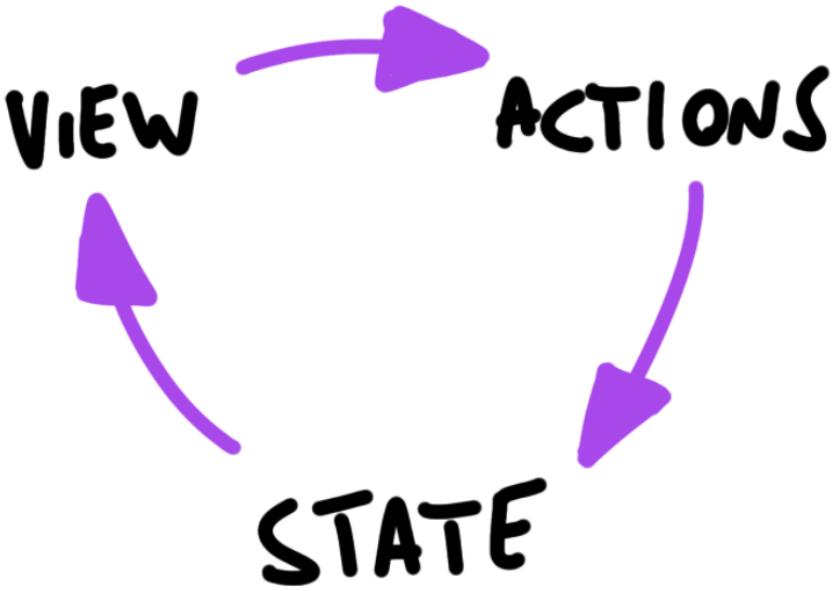
Unidirectional data flow means that data has one, and only one, way to be transferred to other parts of the application.

State is passed to the view and to child components

Actions are triggered by the view

Actions can update the state

The state change is passed to the view and to child components



COMPONENT TYPES

Functional Component

All function component is created using a JavaScript function.

Function component can be a named function or using an arrow function

Function components are also referred as Stateless components.

Class Based Component

Class components as its name suggests are created using the ES6 class syntax.

Every class that extends the React.Component class is obligated to implement the render() method.

The render() method returns a React element

FUNCTIONAL COMPONENTS

React Function Components are the equivalent of React Class Components but expressed as functions instead of classes.

The only constraint for a functional component is to accept ***props*** as an argument and return valid JSX.

In the past, it wasn't possible to use state or side-effects in Function Components – that's why they were called Functional Stateless Components – but that's not the case anymore with *React Hooks* which rebranded them to Function Components.

React Hooks were also introduced to bring side-effects to Function Components. We cover commonly used hooks later.

CLASS BASED COMPONENT

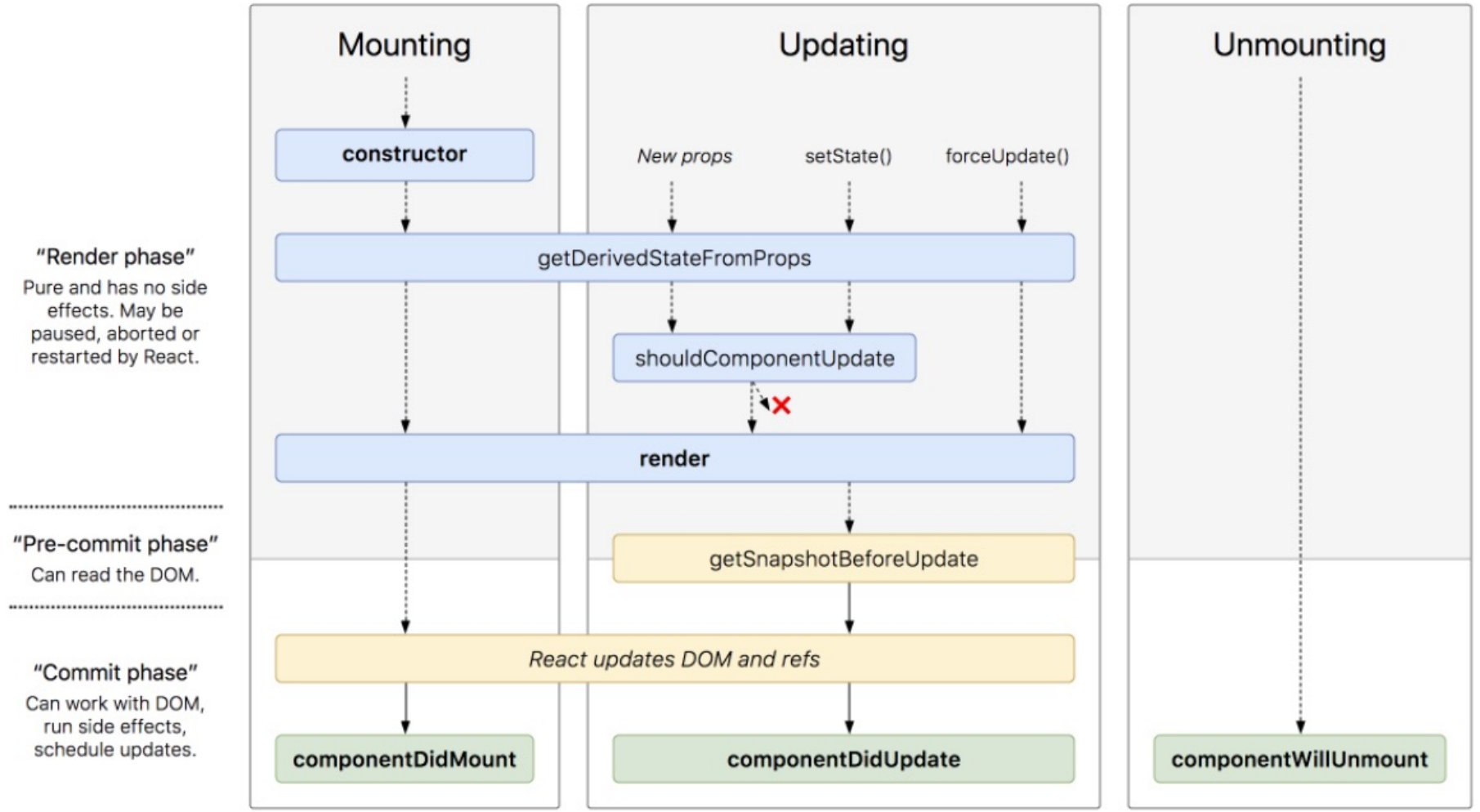
React Class Components were introduced with JavaScript ES6, because JS classes were made available to the language.

All the internal React Component logic comes from the extends React.Componentvia object-oriented inheritance

React class components offer several lifecycle methods for the mounting, updating, and un-mounting of the component.

By using this.state, this.setState(), and lifecycle methods, state management and side-effects can be used side by side in a React class component

COMPONENT LIFE CYCLE METHODS



COMPONENT LIFE CYCLE METHODS

Lifecycle methods are various methods which are invoked at different phases of the lifecycle of a component.

Mounting

- Mounting refers to the loading of components on the DOM.
- This phase contains a set of methods which get invoked when the component is getting initialized, and loaded on to the DOM.

Updating

- This phase starts when the react component has taken birth on the browser and grows by receiving new updates.
- The component can be updated in two ways, sending new props from parents or updating the current state.

Unmounting

- In this phase, the component is not needed and the component will get unmounted from the DOM.

MOUNTING PHASE METHODS

constructor	This is the first method that gets called whenever a component is created. The constructor is called only once in the whole lifecycle of a component. It's used to set up the initial values of variables and component state.
render	This is the required method in a React component, as this method prepares the element that gets mounted on to the browser DOM.
componentDidMount	This is the hook method which is invoked immediately after the component **did** mount on the browser DOM.

UPDATING PHASE METHODS

shouldComponentUpdate	This method tells React that when the component is being updated, it should re-render or skip rendering altogether.
render	And then the component gets rendered
componentDidUpdate	executed when the newly updated component has been updated in the DOM.

UNMOUNTING PHASE METHOD

componentWillUnmount	This method is the last method in the lifecycle. This is executed just before the component gets removed from the DOM

WHY FUNCTIONAL COMPONENTS ?

Functional components make your code easier to read and understand

functional components are easier to test. No hidden state / props.

Avoid unnecessary overhead such as lifecycle events.

Functional components depend only on the props they are given to produce an output which makes debugging easier.

Functional components can reduce coupling

ADD KEYS : RENDERING LIST OF ITEMS

Each child in an array or iteration must be uniquely identified by assigning unique key with the help of “key” prop.

The key should always be supplied directly to the components in an array, not to the container element.

Identity and state of each component must be maintained across render passes.

The key is not really about performance, its more about identity (which in turns leads to better app performance)

FORMS

ACCEPTING USER INPUTS

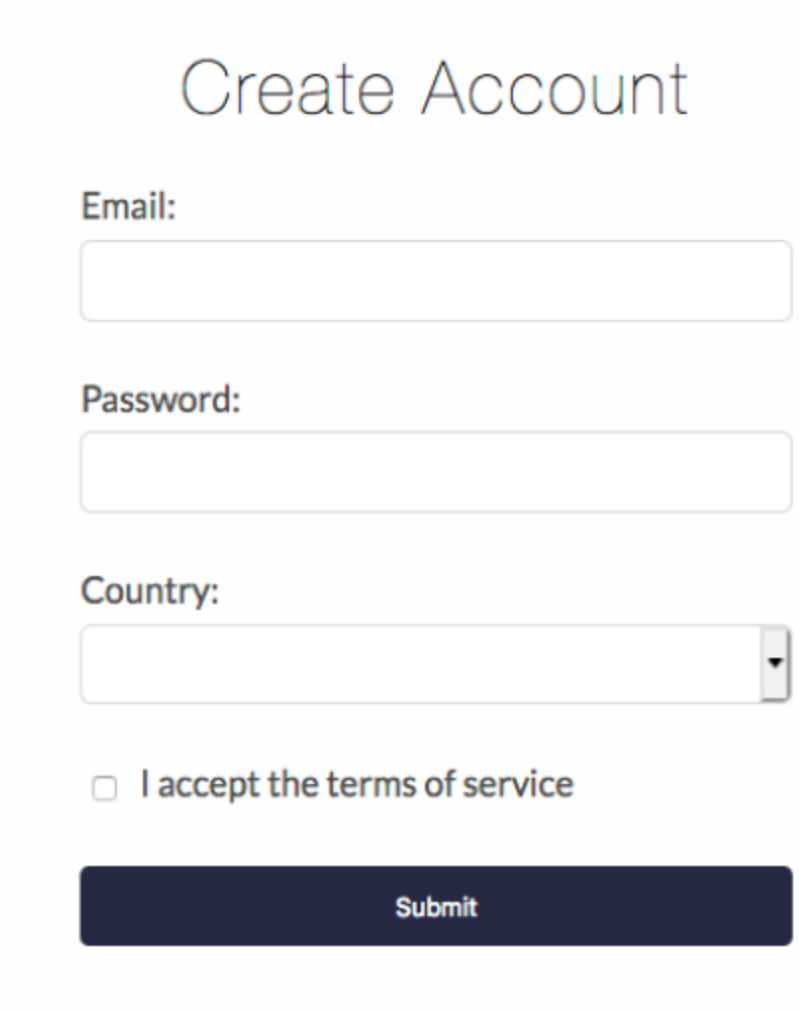
WORKING WITH FORMS

Form elements naturally keep some internal state.

An input form element whose value is controlled by React in this way is called a “controlled component”.

With a controlled component, the input’s value is always driven by the React state.

A controlled component takes its current value through props and notifies the changes through callbacks like an onChange event.



ACCESSING USER INPUT USING REFS

There are mainly two types of form input in React.

- Controlled : Element state managed by React Component
- Uncontrolled : Element state managed by DOM itself

HTML elements maintain their own state that will be updated when the input value changes.

For Uncontrolled elements, there is no need to write an event handler for every state update. You can use a ***ref*** to access the input field value of the form from the DOM

HOOKS

MAKING FUNCTIONAL COMPONENTS POWERFUL

HOOKS

Hooks are a new addition in React 16.8. e.g. useState, useEffect, useContext, useReducer and many more.

They let you use state and other React features without writing a class component.

Don’t call Hooks inside loops, conditions, or nested functions. Only call Hooks at the top level.

Only call Hooks from React functional components

Hook : useState()

```
const [state, setState] = useState(initialState);
```

Returns a stateful value, and a function to update it.

During the initial render, the returned state is the same as the value passed as the first argument (initialState).

The setState function is used to update the state. It accepts a new state value and re-render the component

If the new state is computed using the previous state, you can pass a function to setState. The function will receive the previous value, and return an updated value.

Hook : useEffect()

```
useEffect(didUpdate);
```

Accepts a function that contains imperative, possibly effectful code.

All side-effects can run eg. Mutations, subscriptions, timers, logging, and other possible side effects.

By default, effects run after every completed render, but you can choose to fire them only when certain values have changed.

useEffect may return a clean-up function.

Hook : useContext()

```
const value = useContext(MyContext);
```

Accepts a context object (the value returned from React.createContext) and returns the current context value for that context.

The current context value is determined by the value prop of the nearest <MyContext.Provider> above the calling component in the tree.

The argument to useContext must be the context object itself.

Hook : useReducer()

```
const [state, dispatch] = useReducer(reducer,
                                     initialArg, init);
```

Accepts a reducer of type (state, action) => newState, and returns the current state paired with a dispatch method.

useReducer is usually preferable to useState when you have complex state logic that involves multiple sub-values or when the next state depends on the previous one

Hook : useCallback()

```
const memoizedCallback = useCallback(cb,[]);
```

Returns a memoized callback.

Pass an inline callback and an array of dependencies. useCallback will return a memoized version of the callback that only changes if one of the dependencies has changed.

This is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders

useCallback(fn, deps) is equivalent to useMemo(() => fn, deps).

Hook : useMemo()

```
const memoizedValue = useMemo(() =>
computeExpensiveValue(a, b), [a, b]);
```

Pass a “create” function and an array of dependencies.

The function passed to useMemo runs during rendering.

useMemo will only recompute the memoized value when one of the dependencies has changed.

If no array is provided, a new value will be computed on every render.

Routing & Navigation

Creating Single Page Apps

ROUTING IN REACT

```
npm install react-router-dom
```

React Router is a standard library for routing in React

It enables the navigation among views of various components in a React Application.

React Router allows changing the browser URL and keeps the UI in sync with the URL.

MAIN COMPONENTS OF REACT ROUTER

BrowserRouter	BrowserRouter is a router implementation that uses the HTML5 history API to keep your UI in sync with the URL.
Route / Element	Route is the conditionally shown component that renders some UI when its path matches the current URL.
Link	Link component is used to create links to different routes and implement navigation around the application. It works like HTML anchor tag.

REDUX

PREDICTABLE STATE MANAGEMENT CONTAINER

SHOULD I USE REDUX ?

You have reasonable amounts of data changing over time

You need a single source of truth for your state

You find that keeping all your state in a top level component is no longer sufficient

REDUX : OVERVIEW

npm install redux react-redux

Redux is a predictable state container for JavaScript apps.

It helps you write applications that behave consistently, run in different environments (client, server, and native), and are easy to test.

It provides a great developer experience, such as live code editing combined with a time traveling debugger.

THREE PRINCIPLES OF REDUX

Single source of truth

The state of your whole application is stored in an object tree within a single store.

State is read only

The only way to change the state is to emit an action , an object describing what happened.

Changes are made with pure functions

To specify how the state tree is transformed by actions, you write pure reducers

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()

```
{
    type : “ADD_TODO”,
    payload : “Hello”
}
```

REDUCER

The reducer is a pure function that takes the previous state and an action , and returns the next state.

Reducers specify how the application's state changes in response to actions sent to the store.

Actions only describe what happened, but don't describe how the application's state changes.

```
(prevState, action) => newState
```

DON’TS FOR REDUCERS

Things you should never do inside a reducer:

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

STORE

The Store is the single object.

Holds application state.

Allows access to state.

Allows state to be updated via dispatching action

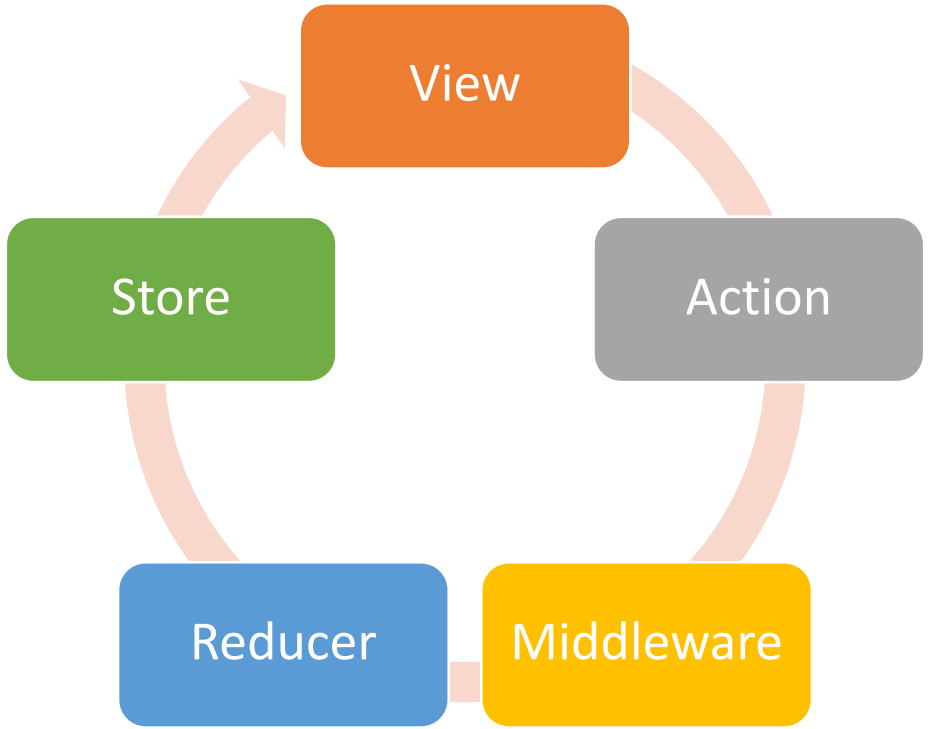
Registers listeners via subscribe(listener).

DATA FLOW IN REDUX

Redux architecture revolves around a strict unidirectional data flow.

The data lifecycle in any Redux app follows these 4 steps:

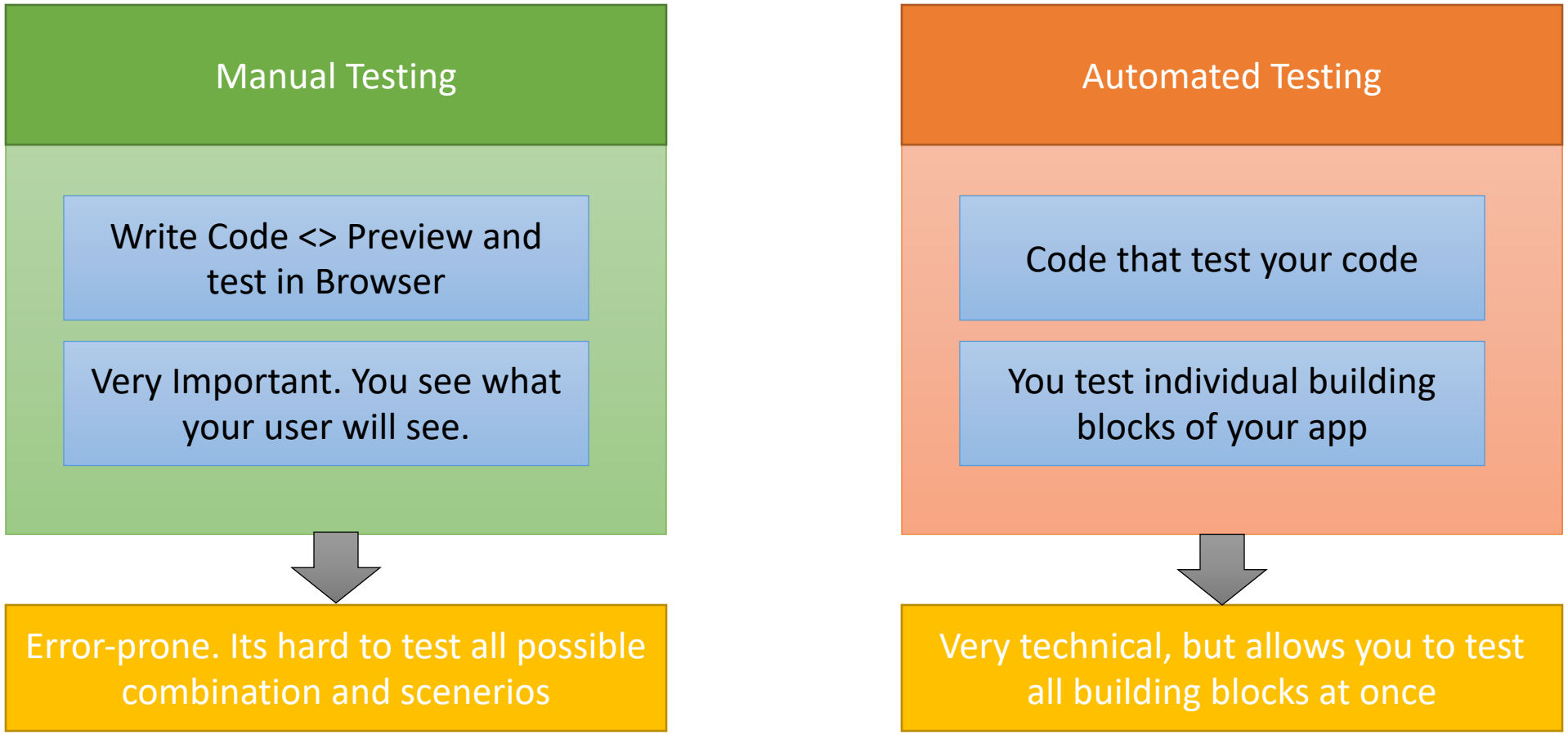
- Your view will dispatch the Action
- The Redux store calls the reducer function.
- The root reducer may combine the output of multiple reducers into a single state tree.
- The Redux store saves the complete state tree returned by the root reducer.



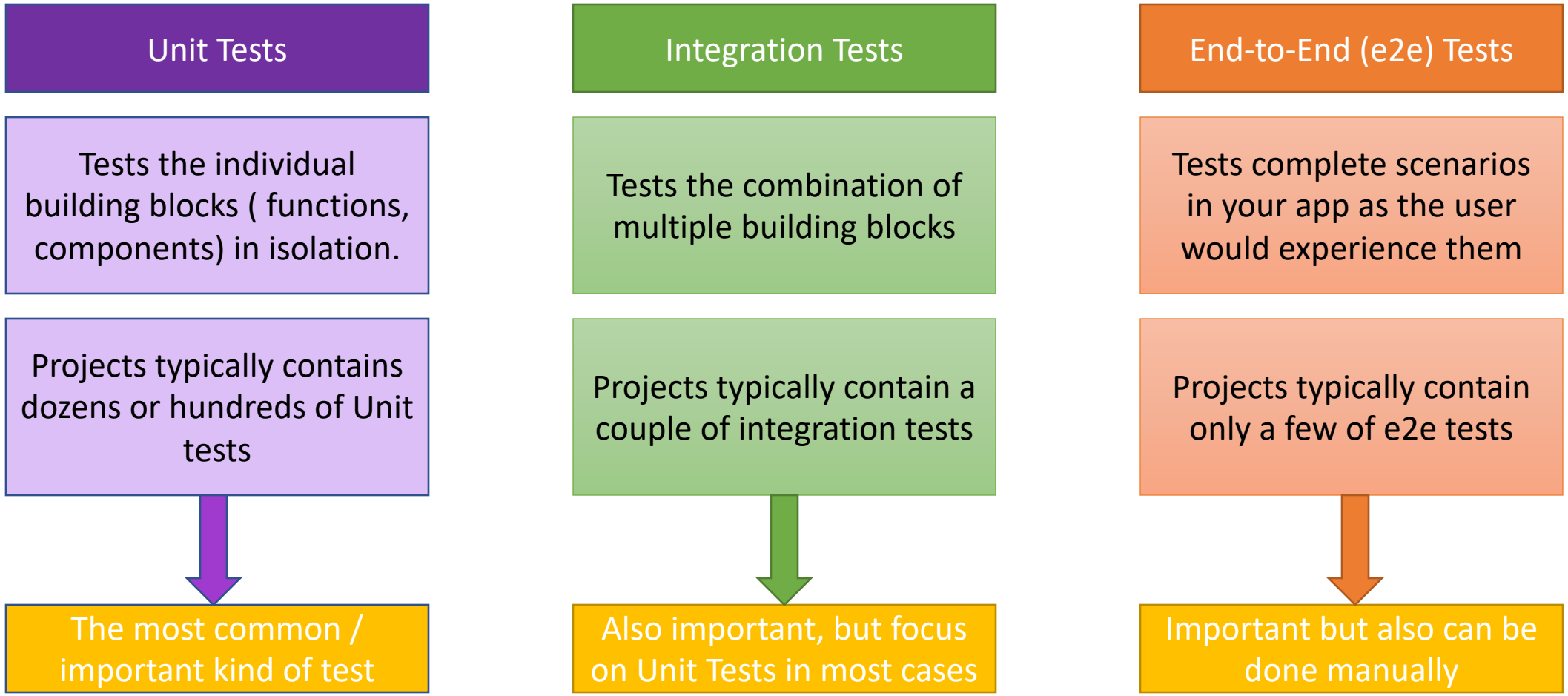
TESTING

GETTING STARTED WITH TESTS

WHAT IS TESTING ?



DIFFERENT KIND OF AUTOMATED TESTS



Thank You

Now you have required foundation to go ahead on you exciting and lucrative journey.

ALL THE BEST