React.JS

(Yet another) Approach to create browser UI

By Sumit Khandelwal

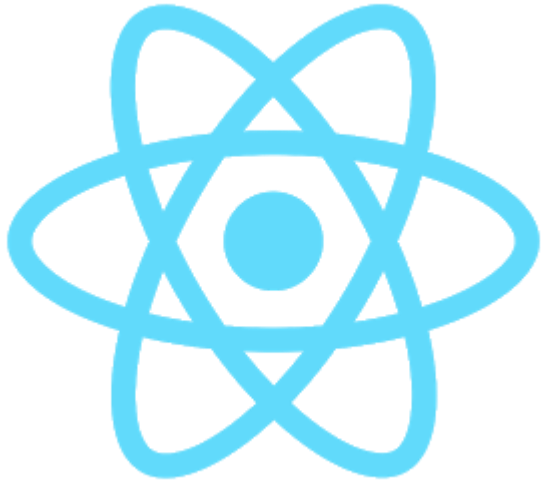
CGI

Trainer Profile

- Senior Consultant & Corporate Trainer having **12+ years of extensive consulting & Corporate Training Experience** in Java, UI & related technologies.
- Efficiently delivered over **4000 hours of corporate training** on JAVA SE, JAVA EE, UI/JavaScript, Intermediate and Advanced levels.
- Rich experience in delivering high-end technologies to diverse participants from Graduate Engineer Trainees (GET) to Project Managers.
- Empaneled Corporate Trainer in various organizations for Java, HTML, CSS, JavaScript & Other related technologies.

CGI

Training Agenda



- ES6+ features
- React – Overview
- Components
- JSX
- Routing
- Redux – Overview & Architecture
- Store
- Reducer
- Actions and Action Creator
- Unit Testing

Prerequisites -

- Good understanding of client-side web programming
- Working knowledge of HTML5
- Good understanding of JavaScript
- Any knowledge of server-side programming is optional

ES6 – New Features

- Arrow Functions
- Promises
- Block Scoping
- Rest & Spread Operators
- Destructuring
- Modules & Classes

Arrow Functions =>

Arrow functions are handy for one-liner functions

() => flavors	Without Curly braces (…args) => expression
	With curly braces (…args) => { body }
Limitations	Don’t have **this** keyword
	Don’t have **arguments** keyword
	Cant call with **new** operator

Arrow Function - Task

- Replace below function with arrow functions in the code:

```
function ask(question, yes, no){
    if(confirm(question)) yes();
    else no();
}

ask(
    "Do you agree?",
    function() { alert("You agreed."); },
    function(){ alert("You cancelled the execution."); }
);
```

Promises

A promise is a special JavaScript object that links the “*producing code*” and the “*consuming code*” together.

A “producing code” that does something and takes time. For instance the code loads a remote script.

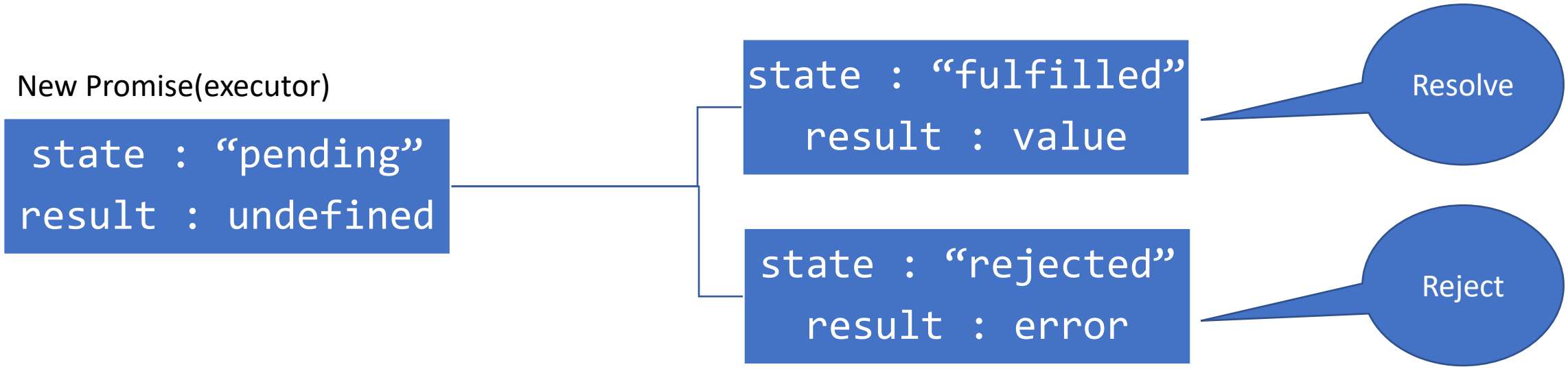
A “consuming code” that wants the result of the “producing code” once it’s ready.

The “producing code” takes whatever time it needs to produce the promised result, and the “promise” makes that result available to all of the subscribed code when it’s ready.

The Promise Object

The resulting promise object has internal properties:

- state —initially “*pending*”, then changes to either “*fulfilled*” or “*rejected*”,
- result —an arbitrary value of your choice, initially “*undefined*”.



Promise : Task

- The function `delay(ms)` should return a promise. That promise should resolve after **ms** milliseconds, so that we can add .**then** to it :

```
function delay(ms) {
   // your code
}

delay(3000).then(() => alert('runs after 3 seconds'));
```

Block Scoping

Restricts the scope of variables to the nearest curly braces { }

Var Types -	***const***: converts the variable to a constant
	let : for all type of variables

Rest / Spread (…)

Rest Parameters	A function can be called with any number of arguments, no matter how it is defined.
	The rest parameters must be at the end.
	Usage : create functions that accept any number of arguments.
Spread Operator	Spread operator looks similar to rest parameters, also using (...), but does quite the opposite.
	It is used in the function call, it “expands” an iterable object into the list of arguments.
	Usage : pass an array to functions that normally require a list of many arguments.

Destructuring

Destructuring assignment is a special syntax that allows us to “unpack” arrays or objects into a bunch of variables.

Object Destructuring	We have an existing object at the right side, that we want to split into variables.
Array Destructuring	the array is destructured into variables, but the array itself is not modified.
Nested Destructuring	If an object or an array contain other objects and arrays, we can use more complex left side patterns to extract deeper portions.

React - Overview

React is a declarative, efficient, and flexible JavaScript library for building user interfaces.

Intended to be the View (“V”) or the user interface in MVC

Aims at effortless development of large scale Single Page App (SPA)

Components are defined and eventually becomes HTML

Why React?

- Easy to understand for developers with the knowledge of XML/HTML
- UI state becomes difficult to handle with Vanilla Javascript
- High Performance : renders quick view
- Focus on business logic
- Huge Ecosystem
- Active community
- Easy to test

React – Let’s do our hands dirty

To get started with React, install the React CLI Tool (create react app)

Run the below command to create new project :

- `npm install create-react-app -g`
- `create-react-app <APP_NAME>`
- `cd <APP_NAME>`
- `npm start`

React Internals – Virtual DOM

“ React abstract away the DOM from you, giving a simpler programming model and better performance ”

React Internals– Virtual DOM

Virtual DOM is in memory lightweight representation of actual DOM.

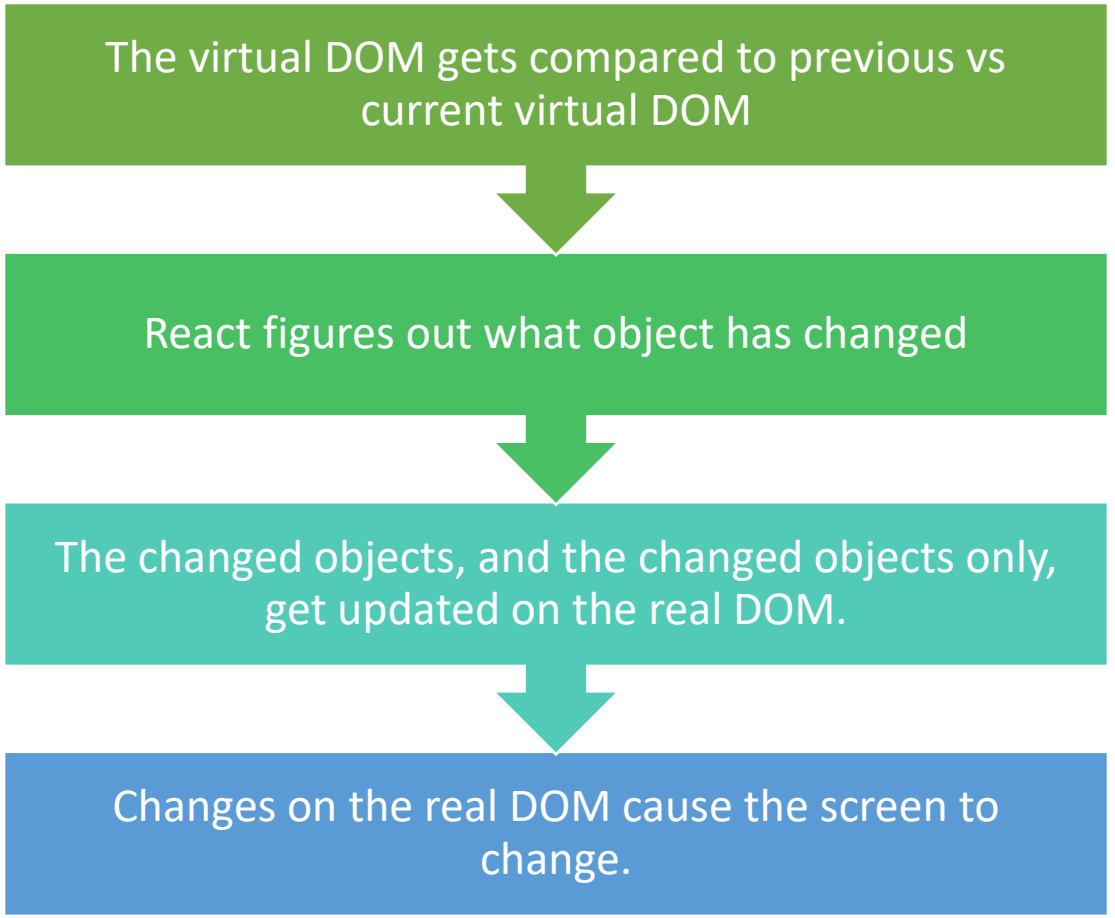
For every DOM object, there is a corresponding Virtual DOM object.

Pure JS intermediate representation.

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

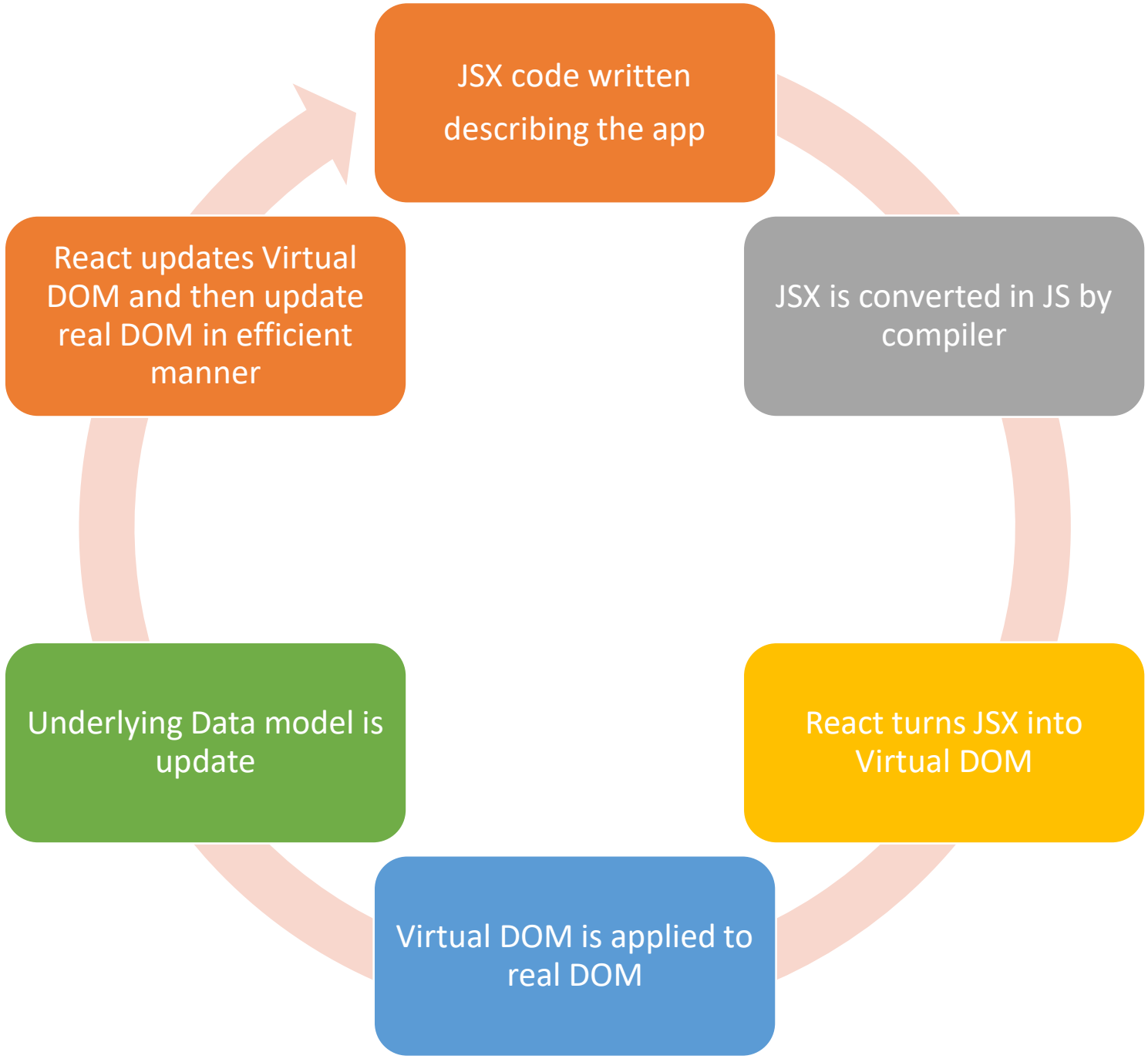
The process of updating any part of the DOM structure is called “reconciliation”

Virtual DOM Update



React Internals

- How React renders the View?



React - Components

Components are the core building block of React apps.

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

Each component needs to return/ render some JSX code

React should render to the real DOM in the end.

React component can be Stateful or Stateless

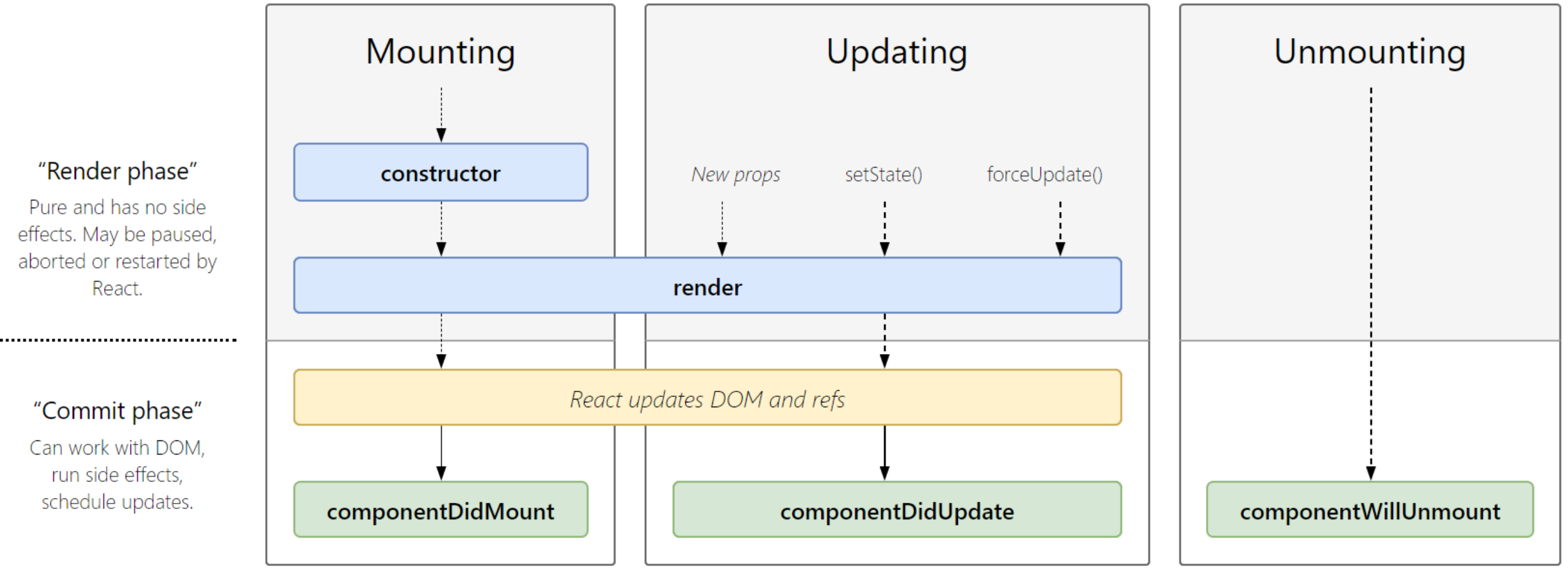
Components can be nested inside other components

React - Component Types

Functional Components

Class based Components (Container)

React – Component Life Cycle Process



React - JSX

JSX comes with the full power of JavaScript.

JSX produces React “elements”.

You can put any valid JavaScript expression inside the curly braces in JSX.

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

Since JSX is closer to JavaScript than to HTML, React DOM uses *camelCase* property naming convention instead of HTML attribute names.

React - Props

Props allow you to pass data from a parent (wrapping) component to a child (embedded) 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 “immutable”

Props are supplied as attribute to components

React - PropTypes

React.propTypes are used to run type checking the props of a component	
Allows to control the presence, or type of certain props passed to the child component	
After 15.5, prop types are moved to library ‘prop types’	
Validators:	String, number, function, Boolean, object, shape, element, any, required etc

```
> npm install prop-types --save
```

React - State

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() with an object map of keys which can be updated with new values. Keys that are not provided will not be affected.

setState() merges the new state with the old state.

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.

Unidirectional Data-flow

React follows unidirectional data flow via the state and props objects.

By keeping the data flow unidirectional you keep a single source of truth.

Clean dataflow architecture

State should be updated using *setState()* method to ensure that the UI is updated and resulting values should be passed down to child components using attributes that are accessible in said children via props.

Adding Keys for Dynamic Children

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

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.

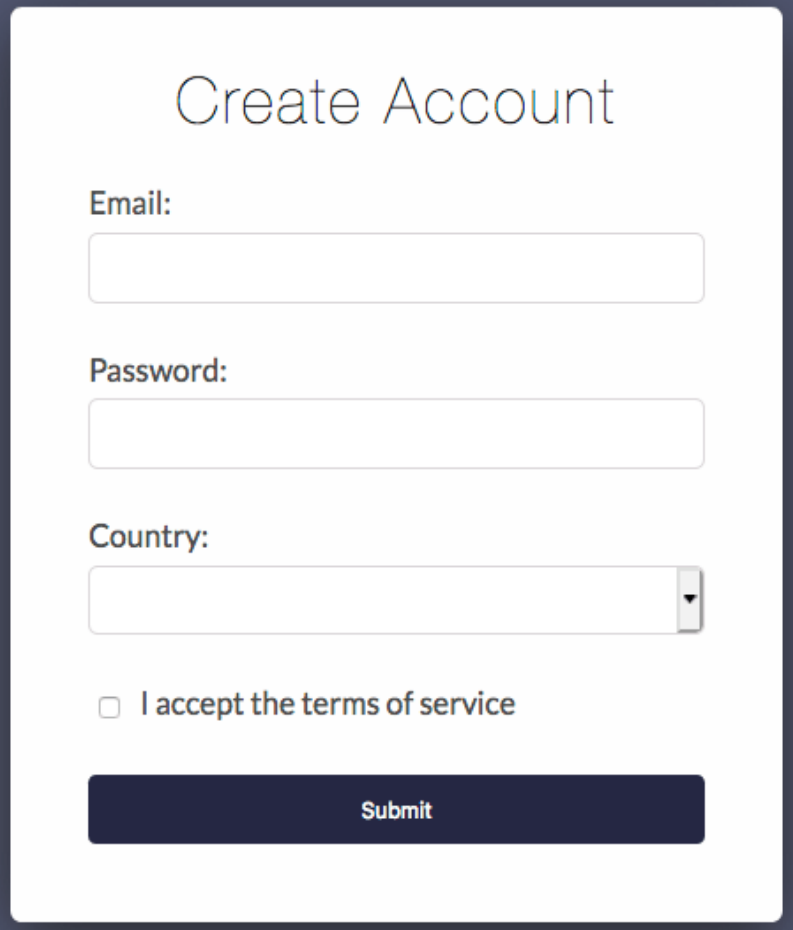
The key is not really about performance, its more about identity (which in turns leads to better performance)

React – Working with Forms

Form elements naturally keep some state internally.

Each form elements in HTML maintain their own state and updates it based on user input.

Form element whose value is controlled by React is called “Controlled Components”



Accessing User Input by Refs

React provides two standard ways to grab values from form elements

- Controlled Components
- Using Refs

Less code writing but hinders in optimized working of Babel inline plugin.

React - 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.

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

Only call Hooks **from React functional components**.

React – Navigation & Routing

Each React app has been a type of SPA.

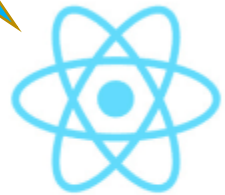
React router library gives us good foundation for building rich applications which have views and URL's.

Routing involves two functionality :	Modify the location of the app (the URL).
	Determining what component need to render at given location.

```
> npm install react-router react-router-dom --save
```

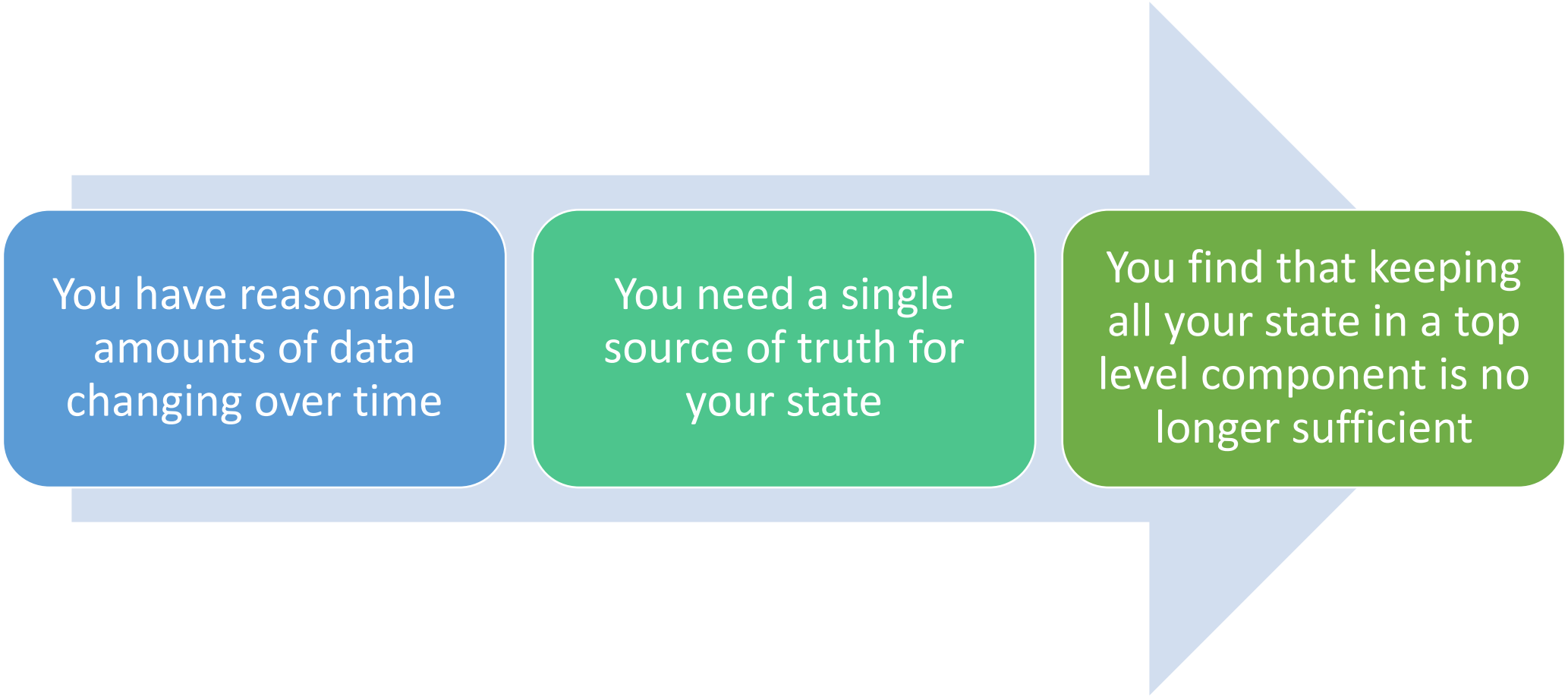
Approach for Creating Browser UI

A Predictable State Management Container For JavaScript Apps



React + Redux

Should I use Redux?



Redux – An Overview

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.

```
> npm install redux react-redux --save
```

Redux : Three Principles

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

Redux - 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,
  text: ‘Hello Redux’
}
```

Redux - Reducers

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

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

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

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

Redux – Do not's 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

Redux - Store

The Store is the single object that has the following responsibilities:

- Holds application state.
- Allows access to state via getState().
- Allows state to be updated via dispatch(action).
- Registers listeners via subscribe(listener).
- Handles unregistering of listeners via the function returned by subscribe(listener).

Creating Store with Root Reducer

```
Import { createStore } from 'redux’
Import rootReducer from './reducers’

const store = createStore(rootReducer)
```

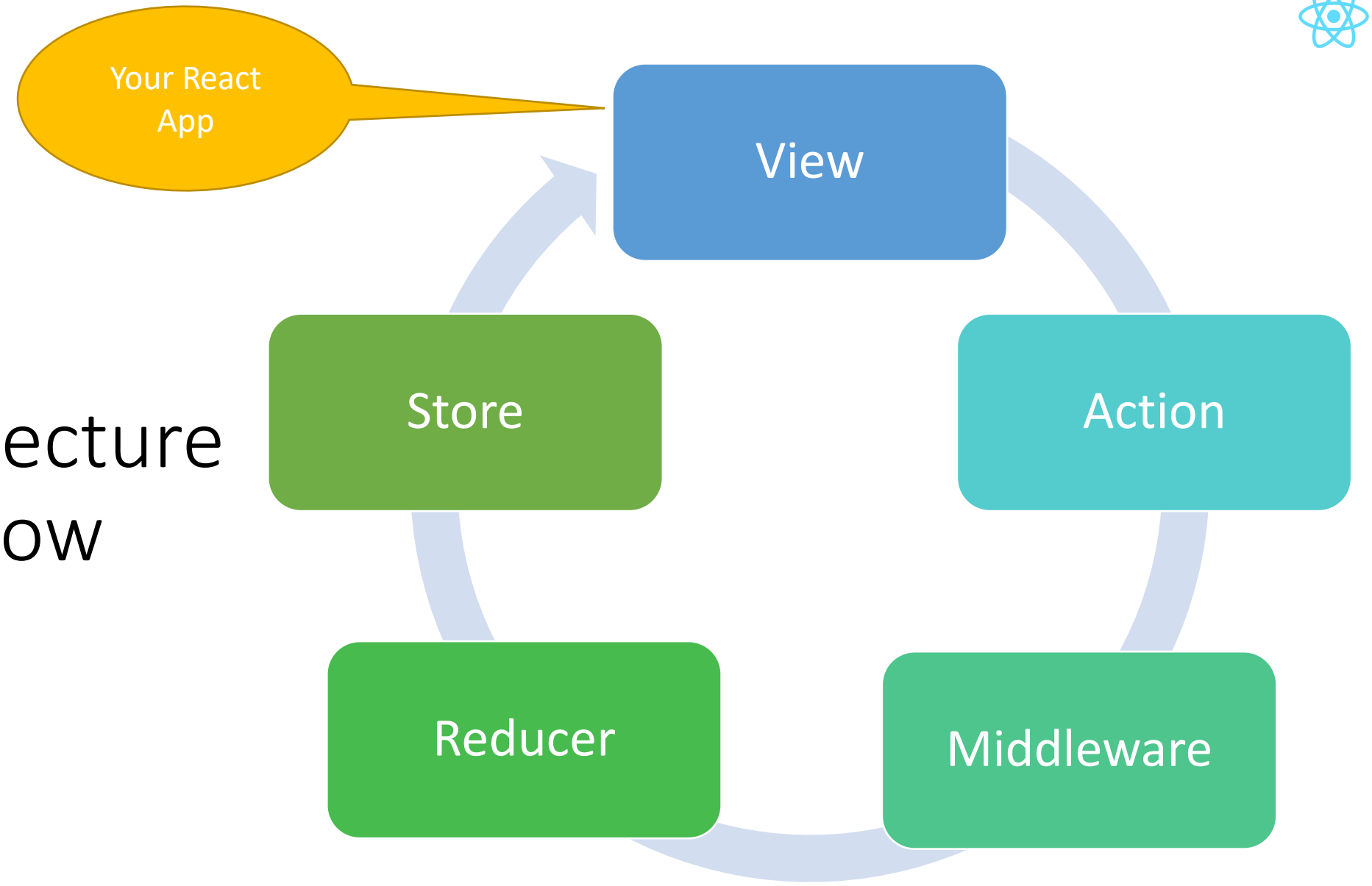
Redux – Data Flow

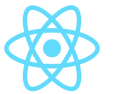
Redux architecture revolves around a strict unidirectional data flow

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

- You call store.dispatch (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.

Redux Architecture & Data Flow





References

- http://javascript.info
- https://reactjs.org
- https://redux.js.org
- https://www.npmjs.com
- https://gist.github.com/danharper/3ca2273125f500429945
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Symbol