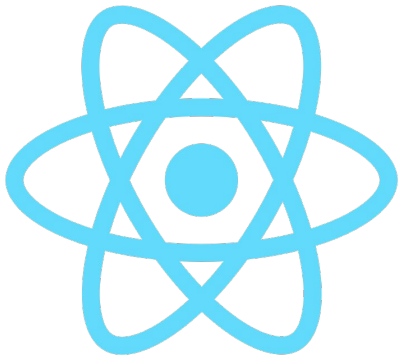


React & Node

Developed by,
Sivaranjini S

Software Engineer - HTC Global Services

The Webinar Curve

Theory – Intro and fundamentals of React 😒

Practical - Live code development with React 😀

Comparing the Theory and Practical 😊

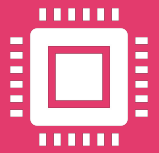
WHAT IS REACT ?

WHY SHOULD YOU USE/LEARN REACT ?

WHAT SHOULD YOU KNOW BEFORE LEARNING REACT ?

Introduction to React

What is React ?



React is a **JavaScript Library** for building user interfaces for front-end web and mobile applications.

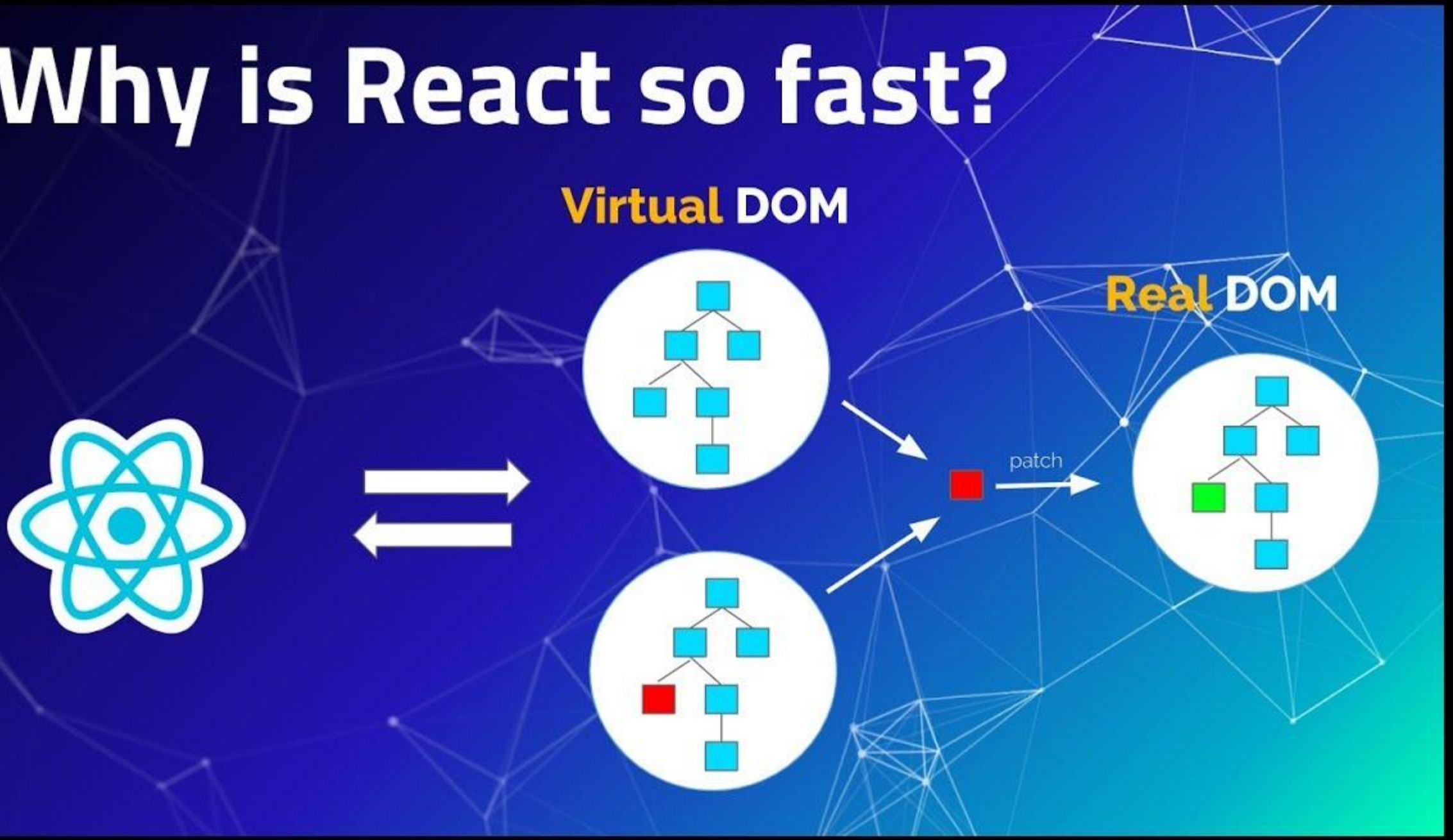
React is created by Jordan Walke, a Software Engineer in **Facebook** and it's initially used for their news feed feature in 2011 and later on Instagram in 2012.

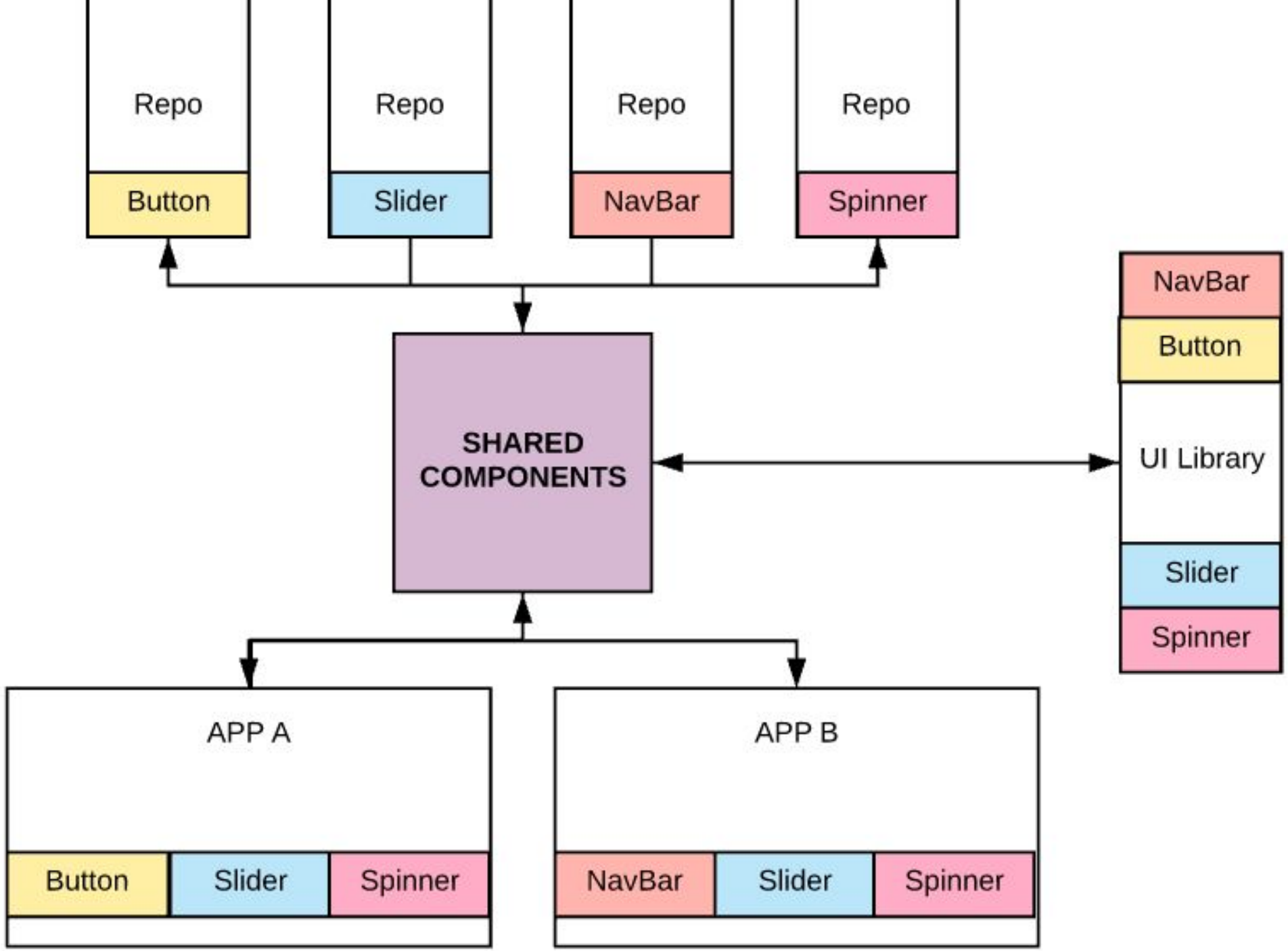
React is sometimes called as a **Framework** because of its features and behaviours.

Why should you use/learn React?

- Reusable and independent UI components.
- Virtual DOM
- JSX
- Currently the most desired technology for front end development.
- Single Page Application
- Can be used for mobile apps

Why is React so fast?




Repo
Button
Repo
Slider
Repo
NavBar
Repo
Spinner
SHARED COMPONENTS
NavBar
Button
UI Library
Slider
Spinner
APP A
Button
Slider
Spinner
APP B
NavBar
Slider
Spinner

JSX

JavaScript XML

```
const name = 'Sivaranjini'
const test = <p>Hello World {name} </p>;
```

What should you know before learning React?

01

JavaScript Basics (Objects, Arrays, Conditionals etc.,)

02

ES6 Features

- Classes
- Array and Object Destructuring
- Array Methods – forEach, map, filter
- Variables – let and const
- Arrow Functions
- Promises or Async and Await

03

Basic knowledge on NPM commands.

COMPONENTS

STATE AND PROPS

LIFECYCLE METHODS

React Fundamentals

React Components

Class

- Make use of ES6 **class** and extend the Component class in React.
- Called as "**stateful**" components as they tend to implement logic and state. – Make use of **state**.
- React lifecycle methods **can** be used inside class components.
- **Render** method must.

Functional

- Functional components are basic JavaScript **functions**.
- Called as "**stateless**" components as they simply accept data and display them in some form. –
- React lifecycle methods **cannot** be used inside class components.
- **Return** statement is must.

Class Component - Constructor

- First executed method in Class Component
- Default lifecycle method while mounting
- Has super() to give to access to this variable

Syntax:

```
constructor(props){
    super(props);
}
```

Class Component - Render

- The **render**() method **is** the only required method in a class component
- It is responsible for describing the view to be **rendered** to the browser
- May / mayn't contains return() method to display HTML components

Syntax:

```
render() {
   return (
     <div>This is Class component.</div>
   );
 }
```

Functional Component - Return

- In the **functional Components**, the **return** value is the JSX code to render to the DOM tree
- Return() is **must** in Functional Component as it need to return any HTML component

Syntax:

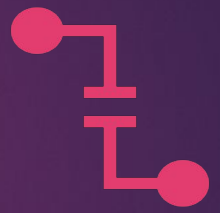
```
const Welcome=()=>
{
return (
    <h1>Welcome to GeeksforGeeks</h1>

    )
}
```

React State and Props

State

Object that holds a component's data and control its behavior.

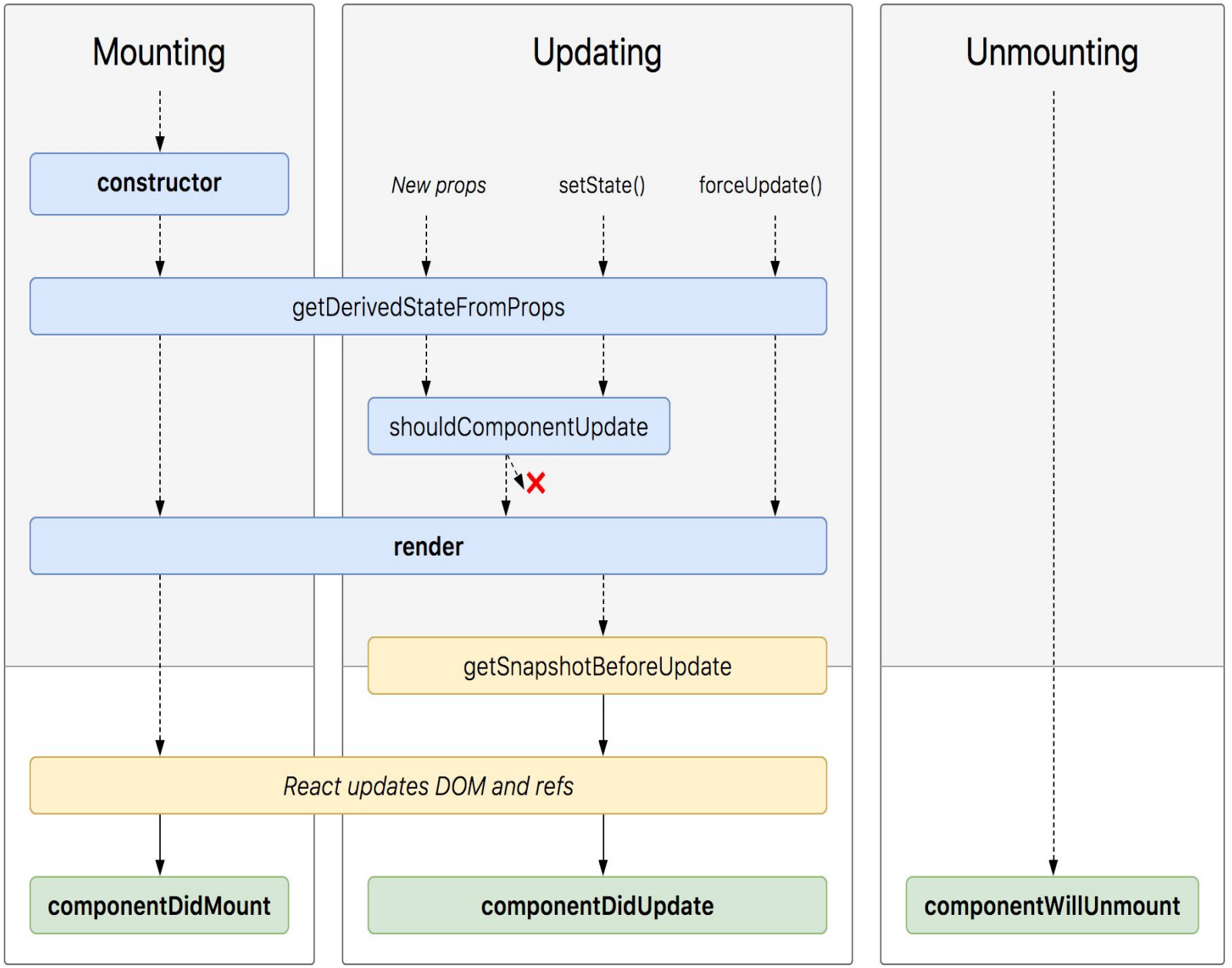
Can only be used in Class components only.



Props

Object that are passed from the parent which is read-only for the child.

Can be used in both class and functional component.



Lifecycle Methods

Installation – Node and create-react-app

- Steps:
- 1. Install Node JS in your System (https://nodejs.org/en/)
- 2. Check node installed by typing **node** in cmd prompt
- 3. To check node version: **nvm list**
- 4. Install create-react-app globally.
- **npm install create-react-app –g**
- 5. Create your react sample application
- **npx create-react-app app-name**
- 6. **cd app-name**
- 7. **yarn install :** To install dependencies
- **8. yarn start :** To start Application

Controlled Forms

1. Controlled data in a state variable
2. Single Source of Truth
3. Validation, Submission, Error Message handled manually

Uncontrolled Forms

1. Formik example of un-controlled Forms
2. Takes Initial Value, Validation, Submission methods as props
3. Return the value, error, handleChange and handleSubmit function which can be triggered in fields

```
<Formik
  initialValues={{ email: '', password: '' }}
  validate={values => {
    const errors = {};
    if (!values.email) {
      errors.email = 'Required';
    } else if (
      !/^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$/i.test(values.email)
    ) {
      errors.email = 'Invalid email address';
    }
    return errors;
  }}
  onSubmit={(values, { setSubmitting }) => {
    setTimeout(() => {
      alert(JSON.stringify(values, null, 2));
      setSubmitting(false);
    }, 400);
  }}
>
  {({
    values,
    errors,
    touched,
    handleChange,
    handleBlur,
    handleSubmit,
    isSubmitting,
    /* and other goodies */
  }) => (
    <form onSubmit={handleSubmit}>
      <input
        type="email"
        name="email"
        onChange={handleChange}
        onBlur={handleBlur}
        value={values.email}
      />
```

React Events

- Just like HTML, React can perform actions based on user events.
- React has the same events as HTML: click, change, mouseover etc.

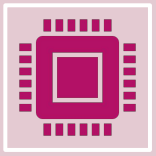
- HTML : <button **onchange**="search()" **class**="" **onclick**="submit()">Search</button>
- REACT: <button **onChange**={this.search} **className=**{} **onClick**={()=> {this.search()}}>Search</button>

Axios API

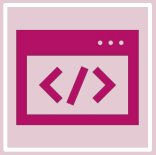
Syntax:

```
axios.get(`https://jsonplaceholder.typicode.com/users`)
.then(res => { })
```

Axios is a library that helps us make http request calls to external resources.



In our **React applications** we often need to retrieve data from external APIs so it can be displayed in our web pages



Axios is promise-based, which gives you the ability to take advantage of JavaScript's async and await for more readable asynchronous code.

Eg: We can make http GET, POST,PUT and DELETE calls via axios

Refs

Refs provide a way to access DOM nodes or React elements created in the render method.

```
myRef = React.createRef();
```

```
<input type="text" ref={myRef} />
```

Keys

Keys help React identify which items have changed, are added, or are removed.

Keys should be given to the elements inside the array to give the elements a stable identity

```
<li key={number.toString()}> {number} </li>
```

React Router

- **React Router** is dynamic, client-side **routing**
- Build a single-page web application without the page refreshing
- Uses component structure to call components

```
import React from "react";
import ReactDOM from "react-dom";
import { BrowserRouter as Router, Route } from "react-router-dom";

// All route props (match, location and history) are available to
User
function User(props) {
  return <h1>Hello {props.match.params.username}!</h1>;
}

ReactDOM.render(
  <Router>
    <Route path="/user/:username" component={User} />
  </Router>,
  node
);
```

React Styling

Styling can be done in three different ways,

- Use traditional CSS framework (bootstrap, material etc) import and follow their semantics
- Use component-based approach MaterialUI and Reactstrap.
- Use your own styles.

Context

Allows you to share information to any component, by storing it in a central place and allowing access to any component that requests it

Provider => Provide the values, Consumers=> Consume the values

Context = React.createContext();

<Context.Provider>

<Context.Consumer>

HOC – Higher Order Components

- **A higher-order component is a function that takes a component and returns a new component.**

```
import HigherOrderComponent from './wrappedComponent';

function App(props) {
  return (
    <>
      <h3> This is {props.componentName} </h3>
    </>
  );
}

export default HigherOrderComponent(App);
```

```
function HigherOrderComponent(AppComponent) {
  return props => {
    return (
      <>
        <h3> Hello </h3>
        <AppComponent componentName={props.componentName}
      </>
    );
  };
}

export default HigherOrderComponent;
```

Pure Components

- Higher Performance
- Prevents re-render incase of same value
- Un-able to prevent re-render in state **objects**
- Applicable for class Component.

```
import React, { PureComponent } from 'react'
import PropTypes from 'prop-types'

class BoxBody extends PureComponent {
  render () {
    console.log('BoxBody rendered')
    return (
      <div style={this.props.options.style}>{t
    )
  }
}
```

REACT HOOKS - BASIC

How to use state variable in Functional Component? 😇

React Hooks helps us here…

Basic Hooks:

1. useState
2. useEffect
3. useContext

Additional

1. useReducer
2. useRef

useState

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

- Contains a Variable "**state**" and a function "**setState**" along with it to update the state.
- setState accepts a new state value and **re-render** component.
- Initial value of the state will be **initialState**, assigned by useState(5)
- By pasing a function to setState, the function will receive the previous value, and return an updated value

```
function Counter({initialCount}) {
  const [count, setCount] = useState(initialCount);
  return (
    <>
      Count: {count}
      <button onClick={() => setCount(initialCount)}>Reset</button>
      <button onClick={() => setCount(prevCount => prevCount - 1)}>-</button>
      <button onClick={() => setCount(prevCount => prevCount + 1)}>+</button>
    </>
  );
}
```

useEffect

```
useEffect(didUpdate);
```

- **Restrictions of Functional Component**
  - Mutations, subscriptions, timers, logging, and other side effects are not allowed inside the main body of a function component which leads to confusing bugs and inconsistencies in the UI
- useEffect solves above Restriction 
- The function passed to useEffect will run after the render is committed to the screen. By default, effects run after every completed render, but you can choose to fire them only when certain values have changed.

```
useEffect(() => {
  const subscription = props.source.subscribe();
  return () => {
    // Clean up the subscription
    subscription.unsubscribe();
  };
});
```

useContext

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

- Accepts : Context object **(React.createContext)**
- Returns : 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.
  - When the nearest <MyContext.Provider> above the component updates, this Hook will trigger a rerender with the latest context value passed to that MyContext provider.
- **Correct:** useContext(MyContext)
- **Incorrect:** useContext(MyContext.Consumer)
- **Incorrect:** useContext(MyContext.Provider)

```
const ThemeContext = React.createContext(themes.light);

function App() {
  return (
    <ThemeContext.Provider value={themes.dark}>
      <Toolbar />
    </ThemeContext.Provider>
  );
}

function Toolbar(props) {
  return (
    <div>
      <ThemedButton />
    </div>
  );
}

function ThemedButton() {
  const theme = useContext(ThemeContext);
  return (
    <button style={{ background: theme.background, color: theme.foreground }}>
      I am styled by theme context!
    </button>
  );
}
```

useReducer

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

- Alternative to useState.
- Accepts : reducer of type **(state, action) => newState**
- Returns : 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.
  - useReducer also lets you optimize performance for components that trigger deep updates because you can pass dispatch down instead of callbacks.

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default:
      throw new Error();
  }
}

function Counter() {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'decrement'})}>-</button>
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
    </>
  );
}
```

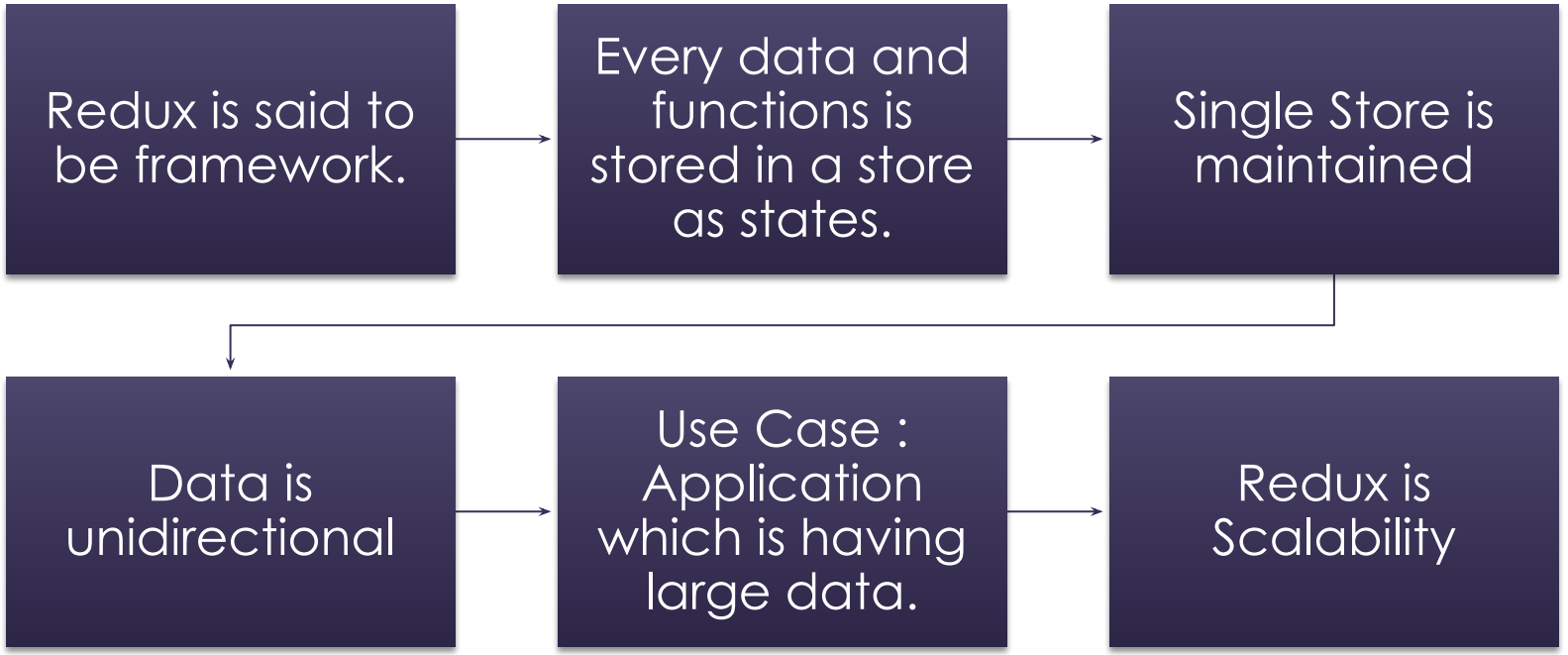
useRef

```
const refContainer = useRef(initialValue);
```

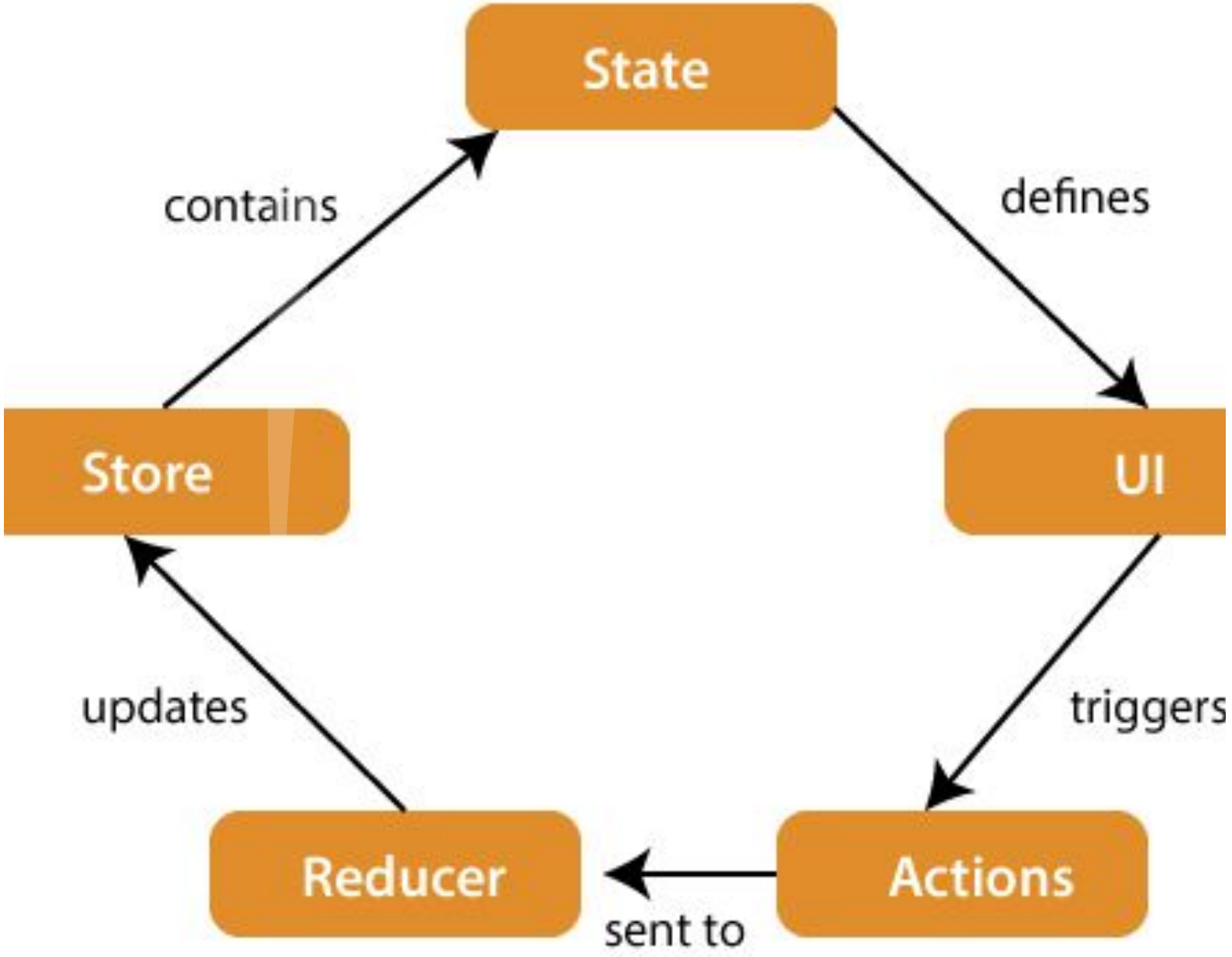
- useRef returns a mutable ref object whose **.current** property is initialized to the passed argument (initialValue).
- The returned object will persist for the full lifetime of the component.
- Essentially, useRef is like a “box” that can hold a mutable value in its **.current** property.

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    // `current` points to the mounted text input element
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>Focus the input</button>
    </>
  );
}
```

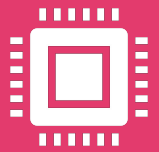
REDUX - Features



- REDUX


State
contains
defines
Store
UI
updates
triggers
Reducer
Actions
sent to

Node JS



Node JS is a server side Javascript Library.

Node JS is created by Ryan Dahl in 2009

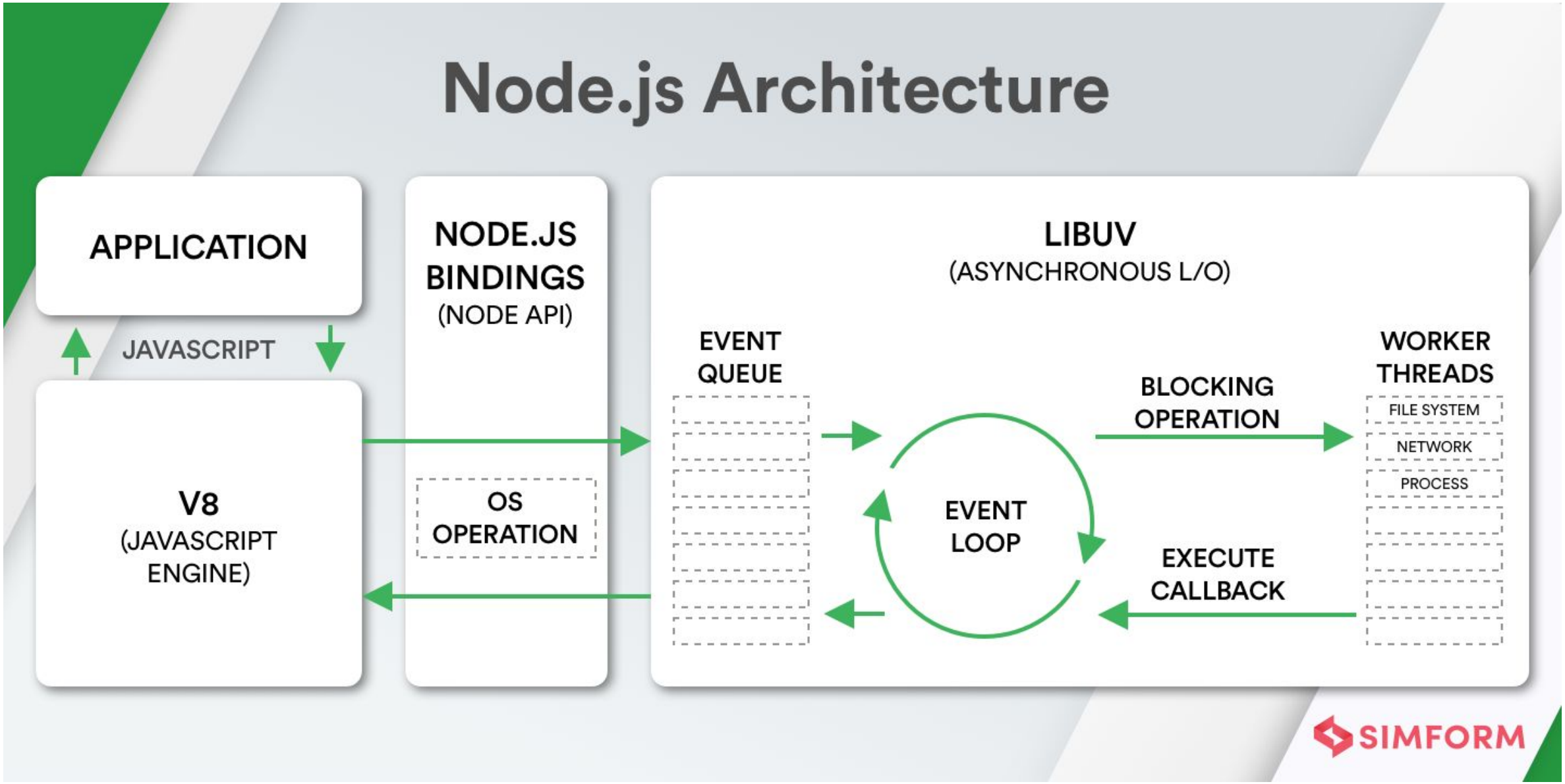
Node JS runs on V8. V8 is an open source JavaScript engine developed by Google.

Node JS is a single threaded application which makes it light weight. It uses event-driven , non-blocking I/O model.

Node JS - Async

- **Asynchronous** programming in **Node**. **js**. **Asynchronous** I/O is a form of input/output processing that permits other processing to continue before the transmission has finished.

Node.js Architecture



SIMFORM


asynchronous.js
asynchronous.js
const getUserAsyncById = require("./getUserAsyncById");
const user1 = getUserAsyncById(1, user1 => {
console.log(user1);
});
const user2 = getUserAsyncById(1, user2 => {
console.log(user2);
});
const sum = 4 + 5;
console.log(sum);
0 0
Ln 2, Col 1
Spaces: 2
UTF-8
LF
JavaScript
Prettier

```
const fetch = require('node-fetch');


async function asyncajaxawait(x)
{
  const res = await fetch('https://api.github.com/users/KrunalLathiya')
  const data = await res.json();
  console.log(data.name);
}


asyncajaxawait(10);
```

Node JS - NPM

NPM – Node Package Manager

- Used to **Maintain Dependencies**
- Packages are **open source** and anyone can import and export
- Provides command line utility for **package installation, version management, and dependency management**.

https://www.npmjs.com/

Promises and Callbacks

Promises

- Has three states **Resolve**, **Reject**, **Pending**
- **Promises** give you back that control.
- Node JS supports different promises.
- **Promises** are one-shot devices and cannot be **used** for repeat notifications.

Callbacks

- Doesn't have states
- Immediately call backed next function.
- **Call back hell**
- Synchronous notifications (such as the **callback** for Array. ... Notifications that may occur more than once

Node JS - Express

WHAT IS EXPRESS ?

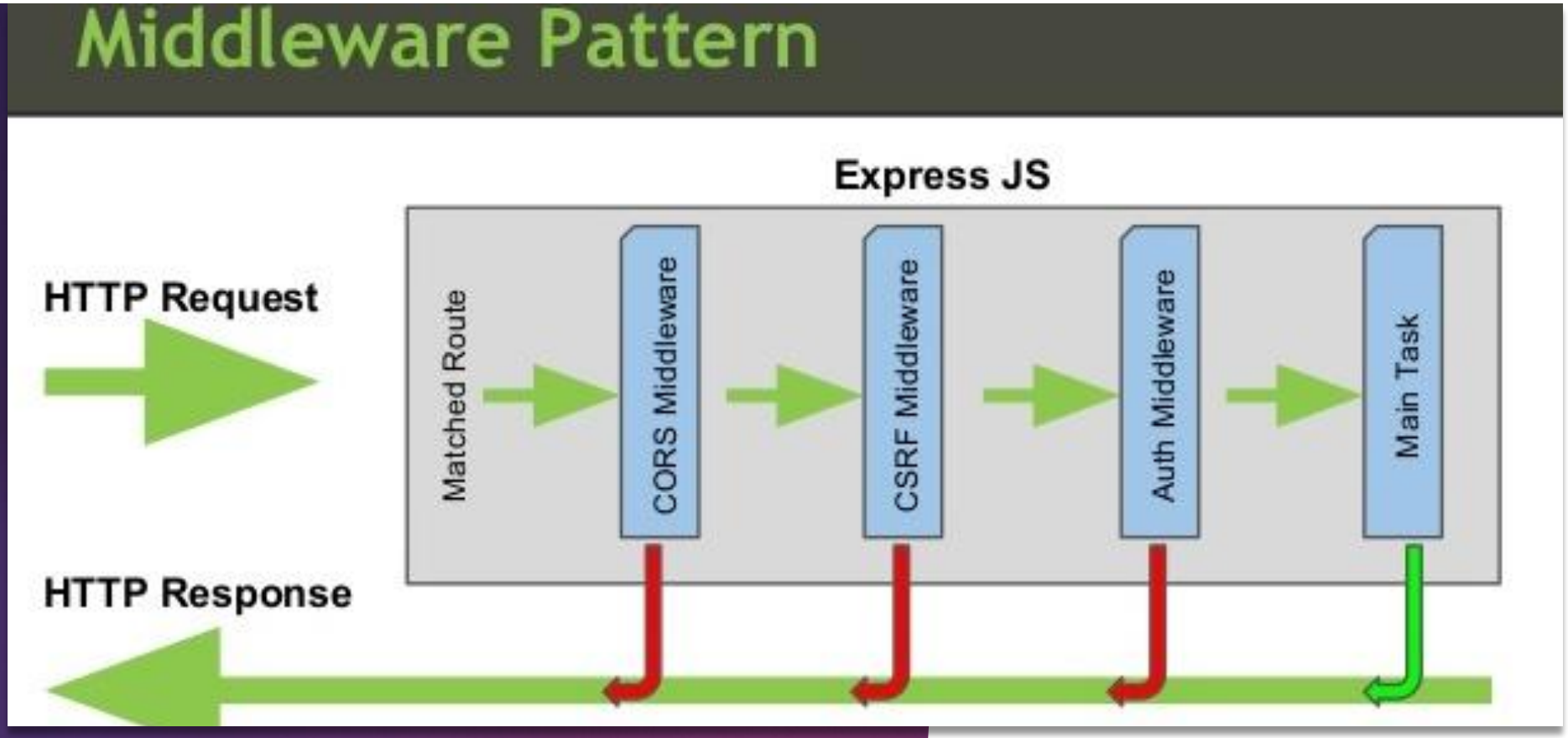
Express is a minimal and flexible **Node.js** web application framework that provides a robust **set of features** to develop web and mobile applications.

WHY EXPRESS?

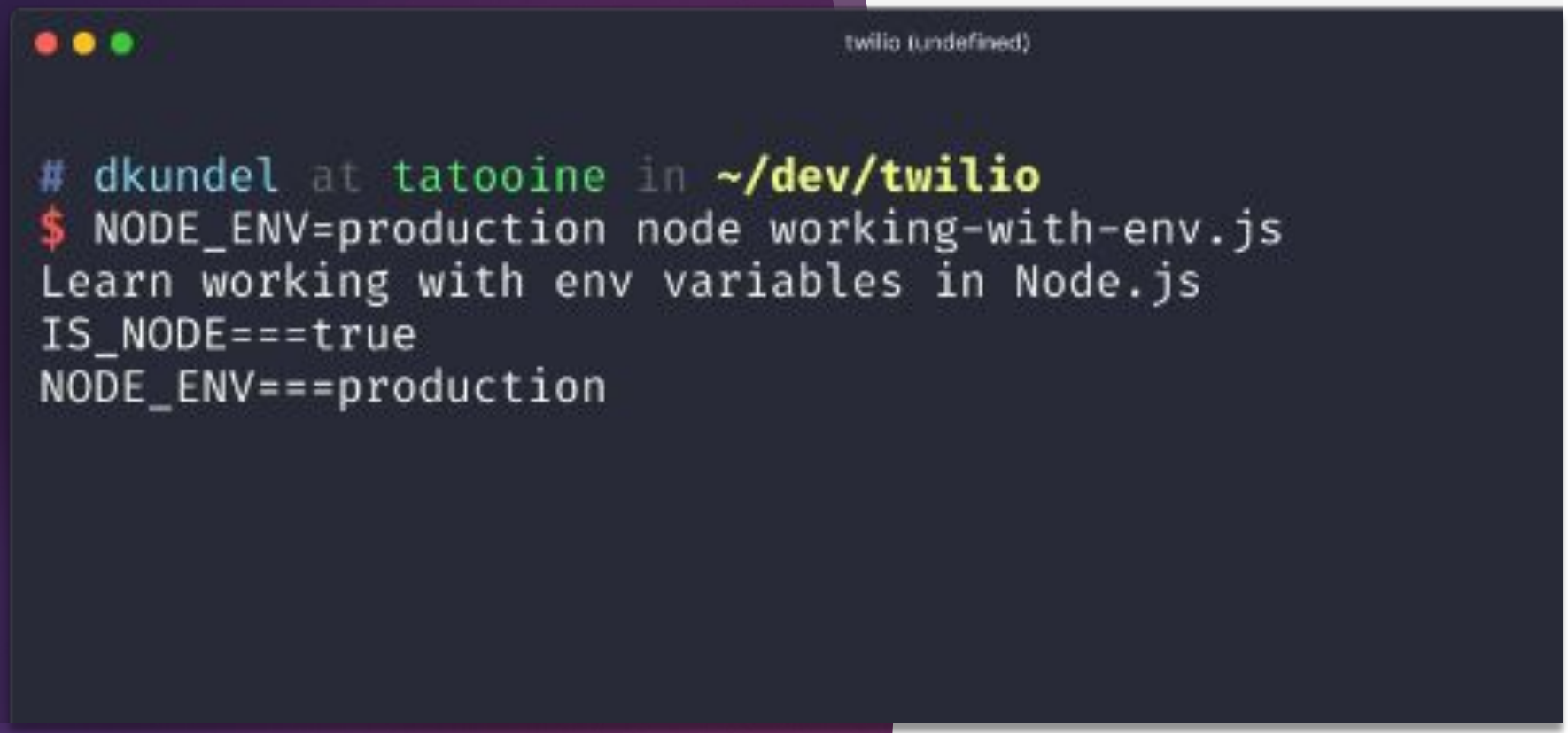
The **express** framework is **built on top** of the node. js framework and helps in fast-tracking development of server-based applications.

WHAT SHOULD YOU KNOW BEFORE LEARNING REACT ?

It's a web framework that let's you structure a web application to handle multiple different **http requests** at a **specific url. Routes** are used to divert users to different parts of the web applications based on the request made.


Middleware Pattern
Express JS
HTTP Request
Matched Route
CORS Middleware
CSRF Middleware
Auth Middleware
Main Task
HTTP Response

Middleware


twilio (undefined)
dkundel at tatooine in ~/dev/twilio
$ NODE_ENV=production node working-with-env.js
Learn working with env variables in Node.js
IS_NODE===true
NODE_ENV===production

PROCESS ENV

DataBase

A DATABASE IS AN ORGANIZED COLLECTION OF STRUCTURED INFORMATION, OR DATA, TYPICALLY STORED ELECTRONICALLY IN A COMPUTER SYSTEM.

Types of Database

RELATIONAL

NON-RELATIONAL (NOSQL)

MySQL

MYSQL IS A RELATIONAL DATABASE MANAGEMENT SYSTEM BASED ON SQL – STRUCTURED QUERY LANGUAGE.

DataTypes

CHAR(SIZE)

VARCHAR(SIZE)

BINARY()

BLOB(SIZE)

ENUM(VAL1, VAL2, VAL3, ...)

BIT(*SIZE*)

BOOL

TINYINT(*SIZE*)

INT(*SIZE*)

FLOAT(*SIZE*)

DATETIME(*FSP*)

TIMESTAMP(*FSP*)

DDL : Data Definition Language

- CREATE
- ALTER
- DROP

DML : Data Manipulation Language

- INSERT
- UPDATE
- DELETE
- TRUNCATE

DQL : Data Query Language

- SELECT
  - WHERE CLAUSE
  - DISTINCT
  - ORDER BY
  - LIMIT
  - OFFSET
  - ALIAS
- CONDITION
  - AND
  - OR
  - NOT
  - IN
  - BETWEEN
  - LIKE

1. COUNT
2. SUM
3. AVG
4. MIN
5. MAX

Aggregate Functions

KEYS AND CONSTRAINTS

KEYS

- PRIMARY KEY
- FOREIGN KEY (MINIMISE DATA REDUNDANCY)
- COMPOSITE KEY

CONSTRAINT

- UNIQUE
- NOT NULL
- DEFAULT

LEFT

RIGHT

INNER

JOINS

NORMALIZATION

A TECHNIQUE OF ORGANIZING THE DATA INTO MULTIPLE RELATED TABLES, TO MINIMIZE THE DATA REDUNDANCY

DATA REDUNDANCY

- REPETATION OF DATA
- DISADVANTAGES
  - INSERTION
  - UPDATION
  - DELETION

1st Normal Form

SCALABLE TABLE DESIGN WHICH CAN BE EASILY EXTENDED

RULES

EACH FIELD SHOULD CONTAIN ATOMIC VALUE

A COLUMN SHOULD CONTAIN VALUES THAT ARE OF SAME TYPE

EACH COLUMN MUST CONTAIN UNIQUE NAME

ORDER IN WHICH DATA IS SAVED DOESN'T MATTER

AVOID **PARTIAL DEPENDENCY**

2ND Normal Form

AVOID **TRANSITIVE DEPENDENCY**

3rd Normal Form

MongoDB

MONGODB IS A DOCUMENT-ORIENTED DATABASE WHICH STORES DATA IN JSON-LIKE DOCUMENTS WITH DYNAMIC SCHEMA. IT MEANS YOU CAN STORE YOUR RECORDS WITHOUT WORRYING ABOUT THE DATA STRUCTURE SUCH AS THE NUMBER OF FIELDS OR TYPES OF FIELDS TO STORE VALUES. MONGODB DOCUMENTS ARE SIMILAR TO JSON OBJECTS.

EG:- MONGODB, COUCHDB, CASSANDRA ETC.,

Collections, Fields, Documents

TABLES - COLLECTIONS

COLUMNS - FIELDS

ROWS - DOCUMENTS

CREATING/SWITCHING TO DB

DELETING DATABASE

CREATE COLLECTION

DROP COLLECTION

DATATYPES

STRING - "DASASD", "DADAS"

INTEGER - 12, 342

BOOLEAN - TRUE, FALSE

ARRAY - [1,2,3], [{ NAME: "GUVI" }, { NAME: "TEST" }]

OBJECT - { AGE: 21 }

DATE

DOUBLE - 34.44, 43.43242, 689

TIMESTAMP

$group
$project
$match
$limit
$skip
$sort

AGGREGATE FUNCTIONS