<body>

    <div *id*="root"></div>

    <script>

*// Creating HTML tag using Javascript*

*const* heading = document.createElement("h1");

      heading.innerHTML = "Hello World From Javascript!";

*const* root = document.getElementById("root");

      root.appendChild(heading);

    </script>

  </body>

**Q:-How does browser understand what is document,createElement,getElementById all these things how browser can Understand??**

* Browser has Javascript Engine that exectute this JavaScript
* But Browser Don’t Understand React so first we need to get react into our project

**# There is 2 way of adding react to our project**

CDN Links

<script *crossorigin* *src*="https://unpkg.com/react@18/umd/react.development.js"></script>

<script *crossorigin* *src*="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

1 and 2 links are different for a reason check BookPage-11

* **Via CDN links**
* react.development.js
* react-dom.development.js
* fetch from CDN is costly Operation, it will make a network call to unpkg.com and it will get react from unpkg.com
* suppose we have already react in my node\_module [more on page 7]
* **Via npm**
* npm install react
* npm install react-dom

**Q:- What is CORS (Cross-Origin Resource Sharing)**

* <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS>

**# Creating HTML Tag Element in React**

// Before root and after root all element work fine only affected those inside the root by react (but what is rendering matter, I’m just giving example of root)

    <div *id*="root">

*//Whatever inside the id root will be replaced after root.render*

</div>

    <script *crossorigin src*="https://unpkg.com/react@18/umd/react.development.js"></script>

    <script *Crossorigin* *src*="https://unpkg.com/react-dom@18/umd/react-dom.development.js"></script>

    <script>

*// Its the Job of 1st CDN Link react*

*const* heading = React.createElement("h1", {}, "Hello World From React!");

*// Its the Job of 2nd CDN Link react-dom*

*const* root = ReactDOM.createRoot(document.getElementById("root"));

      root.render(heading);

    </script>

* *const* heading = React.createElement("h1", {}, "Hello World From React!");
* this heading Element at the end of the is Javascript Object
* root.render(heading);
* this render method is basically responsible to take this “heading” object and put it up, Convert it into the “HTML heading tag” and put it up(root i.e mention in code) on the DOM

# NPM is not Node package Manager, in npm’s official website there is no place where it has written that npm is Node Package Manager.

NPM Does not have Full Form 😅, so basically npm manages packages but it does not stand for Node Package Manager.

**# Initializing a New Project**

To create a new project, navigate to your desired project folder and run the following command:

$ npm init

This command will prompt you to enter some basic information about your project, such as the name, version, description, and entry point. Once you've provided the required information, ***npm will generate a package.json file***, which will contain all your project's metadata and dependencies.

**# Understanding package.json**

The package.json file is the heart of your project, as it stores all the necessary information about your project, such as its name, version, description, dependencies, and more. Here's a simple example of a package.json file:

{

"name": "my-project",

"version": "1.0.0",

"description": "A simple example project",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"dependencies": {

"express": "^4.17.1"

}

}

In this example, the project is named "my-project" and has a single dependency: the Express.js framework. The dependencies object lists all the packages required for your project to run correctly.

**# what-is-npm-and-how-does-it-work**

( <https://reintech.io/blog/what-is-npm-and-how-does-it-work> )

**# Package.json is Configuration for npm**

(<https://heynode.com/tutorial/what-packagejson/#:~:text=Your%20package.,dependencies%20required%20by%20the%20application>)

* **Sometimes Packages also known as Dependencies**

**# React Bundler**

(<https://www.dhiwise.com/post/embark-an-enlightening-journey-with-react-bundler> )

* When we have HTML CSS and JS file our whole code needs to be bundled together, our whole code needs to be minified, whole code needs to be compressed and needs to be cleaned before it can be sent to production so Bundler helps you to do all that things(webpack, parcel, vite)
* These bundlers are the job of to basically bundles our app it packages our app properly so that it can be shipped to production
* That “create-react-app” behind the scene uses “webpack” bundler
* But in our project we are using “parcel” bundler

**# parcel Bundler**

* Parcel Bundler Ignite our App
* npm install -D parcel
* here -D is devDependency(search on net for more info)
* These are your development dependencies. Dependencies that you need at some point in the development workflow but not while running your code (e.g. Babel or Flow).
* ( <https://github.com/parcel-bundler/parcel> ) / (<https://parceljs.org/>)

**# Dependencies vs devDependency**

* (<https://medium.com/@reemshakes/devdependencies-vs-dependencies-in-reactjs-db7261e13012#:~:text=When%20your%20project%20needs%20code,list%20of%20your%20project's%20dependencies> )

# **whats the difference between tilde(~) and caret(^) in package.json?**

* ~version **“Approximately equivalent to version”**, will update you to all future patch versions, without incrementing the minor version. ~1.2.3 will use releases from 1.2.3 to <1.3.0.
* ^version **“Compatible with version”**, will update you to all future minor/patch versions, without incrementing the major version. ^1.2.3 will use releases from 1.2.3 to <2.0.0.
* (<https://stackoverflow.com/questions/22343224/whats-the-difference-between-tilde-and-caret-in-package-json> )

**# Package.json vs Package-lock.json**

* What is the purpose of package.json?
* Tilde (~) and carat (^), and their difference
* What is package-lock.json?
* What is the purpose of package-lock.json?
* Comparing package.json and package-lock.json
* What is the role of npm-shrinkwrap.json in versioning?
* (<https://www.atatus.com/blog/package-json-vs-package-lock-json/> )
* "node\_modules/@babel/code-frame": {
* "version": "7.23.5",
* "resolved": "https://registry.npmjs.org/@babel/code-frame/-/code-frame-7.23.5.tgz",
* "integrity": "sha512-CgH3s1a96LipHCmSUmYFPwY7MNx8C3avkq7i4Wl3cfa662ldtUe4VM1TPXX70pfmrlWTb6jLqTYrZyT2ZTJBgA==",
* "dev": true,
* "dependencies": {
* "@babel/highlight": "^7.23.4",
* "chalk": "^2.4.2"
* },
* "engines": {
* "node": ">=6.9.0"
* }
* },
* Above data is from package-lock.json, **here the integrity meaning**
* This is Hash, have you heard of that thing it is working on my Local, I Don’t know how it break / it’s not working on Production
* So basically to avoid that package-lock.json keeps hash to verify that whatever is there in my machine is the same version which is being deployed onto the production.
* It’s very important file package-lock.json it keeps a track of all the exact versions of all dependencies
* When we install parcel there is one more thing that was created i.e node\_modules

Q:- What is node\_modules folder in react (BookPage-15)

* When creating a new React app your project will be populated with a bunch of new folders and files, and you may have noticed a **node\_modules** folder that contains an insane amount of folders.
* Because our needs parcel / project has dependency parcel. Now parcel as a project has it’s own dependencies and those dependencies can have it’s own dependencies, those dependencies can gave their own dependencies, this is known as “**Transitive Dependencies**”, Now this parcel can itself be dependent on lot of things. Parcel cannot do all these things on its own, parcel needs help of a lot of other packages, example parcel needs help of Babel also.(*Akshay Saini*)
* node\_modules are one of the most important directories in your React project as React requires node\_modules to run. The node\_modules directory is where all the dependancies packages are stored that are used to build and run your react project.
* So this is you'll find packages like React and React-DOM, your build packages like Vite, Babal or Webpack, and linters like ESLint or Prettier to name just a few. This directory can contain hundreds of dependancies!
* Q:- **Why aren't they included in version control?**
* => The main reason is the sheer size of this directory. Rather than including hundreds of package dependancies in version control, we can instead track a file called **package.json** which contains information about the project, and a list of dependencies required by the app. Other developers can use the package.json file and **npm install** to regenerate the node\_modules.

# **😱 when accidentally tracked node\_modules**

* Best practice would be to include the node\_modules in the **.gitignore** file in your project before pushing any code,
* which will prevent version control from tracking this folder - but if you've accidentally tracked and pushed the node\_modules to GitHub, like I did in my first project 🤦🏻‍♀️, simply follow the steps below.
* **Remove node\_modules from version control**
* Create a .gitignore file in your project and add node\_modules
* **Remove the node\_modules:**
* ***git rm -r --cached node\_modules***
* Commit and push without the node\_modules. The node\_modules should now be deleted from your repository.

**# Igniting Our App / Running Our Application On a Development Server**

* npx parcel src/index.html
* Output
* ❯ npx parcel src/index.html
* Server running at http://localhost:1234
* ✨ Built in 5ms
* Parcel’s built-in development server is now running. The npx parcel command takes your entry point of src/index.html and builds your application with the necessary assets. The output also indicates that the application is running on <http://localhost:1234>.
* Just like we have npm similarly we have something known as npx that means executing a package, npm if we have to install a package we need to write npm install
* ***# How to bundle a web app with parcel***
* <https://www.digitalocean.com/community/tutorials/how-to-bundle-a-web-app-with-parcel-js>

**# installing React from npm**

* Suppose if we already have react in my node\_modules, how easy would it be to use inside our code
* So first thing is that we don’t want to make another network call to get react
* We will have it already in our node\_modules that is why we will install it
* The second thing is today <https://unpkg.com/react@18/umd/react.development.js> react@18, tomorrow React@19 came in, and some other version of react came in, what will happen is we will have to keep changing this ***URL.***
* So it’s better to just have it inside our package.json it is easier to manage all are dependecies and it is very easy to manage react also.
* As one of the dependency inside our npm package.json
* Command => npm install react or npm i react
* Command => npm install react-dom or npm i react-dom
* After this we got "react": "^18.2.0" and "react-dom": "^18.2.0"dependencies in package.json file
* See is react over here react version ^18.2.0 and ^ carret, suppose tomorrow 18.2.3 comes in it will be there in our code
* Package-lock.json will also have
* Now we no longer need CDN links
* Now we have to import React and ReactDOM
* In our HTML we are injecting this App.js, Browser thinks it’s normal javascript file and it does not understand what is import, normal javascript does not need import.
* <script *type*="module" *src*="/App.js"></script>
* That is why we need to tell the browser that this file is not a normal browser file, it’s module.
* We have to write *type*="module" over here

**# Create Prod Build**

* *Command=> npx parcel build index.html*
* Only need to add build for prod build
* But if we run this command we will get error
* If we are using parcel we need to remove **"main": "App.js"** from package.json file
* **PS C:\Users\riyaz\OneDrive\Desktop\React-Practice> npx parcel build index.html**
* **npm WARN config global `--global`, `--local` are deprecated. Use `--location=global` instead.**
* **✨ Built in 3.06s**
* **dist\index.html 378 B 1.20s**
* **dist\index.8d566482.css 84 B 785ms**
* **dist\index.47db806b.js 138.77 KB 1.56s**
* Suppose if we our app will have like 10, 20 files it will compress everything and minify everything to these 3 files and now these 3 files will contain all the code that we write these 3 files are the production ready code of our app
* It will bundle, it will minify it will put all those files inside new folder **“dist”** , but before production build the **“dist”**  folder contains the development build files
* When we write npx parcel index.html it generate a development build of our project and it host it onto **localhost:1234** and put it up into this **“dist”**  folder
* If we delete dist and .parcel-cache these are the temporary folders
* If we do *npx parcel index.html* or *npx parcel build index.html* these dist and .parcel-cache folder automatically regenerated
* So these things any code that we can automatically generate we don’t have to put in GitHub
* Put this things in .gitignore

**Babel in React**

( <https://www.scaler.com/topics/react/what-is-babel-in-react/> )

1. **Difference Between HTML and JSX**

( <https://www.freecodecamp.org/news/html-vs-jsx-whats-the-difference/> ) ( <https://codersera.com/blog/react-functional-components/> )

1. **Functional Componet**

( <https://codersera.com/blog/react-functional-components/> )

**Config-driven-UI**

[**https://www.freecodecamp.org/news/javascript-optional-chaining/**](https://www.freecodecamp.org/news/javascript-optional-chaining/)

**map array**

**# What is Browserslist**

* [**Browserslist**](https://browsersl.ist/) can specify which browsers your web application can run in, it provides a configuration for specifying browsers range. Browserslist has become a standard in the industry, it is used by libraries such as Autoprefixer, Babel, ESLint, PostCSS, SWC and Webpack.
* If we specify some browser’s list that means that listed browser’s definitely 100 of the times will work on these, it might or might not be work in rest of the browser’s
* (<https://modernjs.dev/builder/en/guide/advanced/browserslist> ) / (<https://github.com/browserslist/browserslist?tab=readme-ov-file#queries> )

**# Creating script for run server**

* Earlier we doing like this npx parcel index.html now instead of writing this command again and again we just create a simple script
* It is an npm script we need to create that script in our Package.json file
* So we can create different script for development and production build
* "scripts": {
* "start": "parcel index.html", // Development script
* "build": "parcel build index.html", // Production script
* "test": "jest"
* },
* Now when we have created these scripts we no longer have to write npx parcel index.html to run our code
* Now we can use npm run start / npm run build to start our script (here start and build are script name)
* if you go to company and you don’t know how to start the project, just goto their package.json find this script and you will get the exact command to run the code

**# React Element**

*// Creating React Element using React*

*- const* heading = React.createElement("h1",{ id: "heading" },"Namaste React 🚀");

* Lot’s of developer thinks this is React.createElement is basically a HTML element, NO it’s not an HTML element
* How its work => When we do React.createElement it gives React Element and this React Element is basically an Object. So React Element at the end of the day is an javascript Object. Then this JS Object is rendered as an HTML Element.
* *const* root = ReactDOM.createRoot(document.getElementById("root"));
* root.render(heading);
* This ReactDOM takes this Object and convert it to HTML and push it to the browser
* It will replace not append everything that is inside this root whatever it is it will be replace when rendering.

**# JSX Syntax eXtension for JavaScript**

* JSX is a JavaScript syntax which is easier to create react elements
* Lot’s of people think’s JSX is a part of React, NO it’s not a part of React.
* React is Different JSX is Different, we can write React without JSX also But JSX makes our Developer life easy that is why we use JSX
* *JSX is not HTML* inside Javascript, JSX is Different than HTML. JSX is a HTML like Syntax
* JSX is a convention where we kind of merge HTML and JavaScript together
* JSX is just a **Syntax**
* *// Creating React Element using JSX*
* *const* jsxHeading = <h1 *id*="heading">Namaste React 🚀 from JSX</h1>;
* How its work => When we write JSX code is *Transpiled*(converting)to React.createElement it gives React Element and this React Element is basically an Object. So React Element at the end of the day is an javascript Object. Then this JS Object is rendered as an HTML Element.
* Babel is Converting our JSX to React.createElement
* jsxHeading Element is same as above heading Element both are object before Rendering

**# Babel**

* *const* jsxHeading = <h1 *id*="heading">Namaste React 🚀 from JSX</h1>
* So this JSX is a not a pure valid Javascript code.
* Any piece of Javascript code that JS Engine can understand.
* So the JS Engine not understand the Above JSX code, then how this code is working?
* Bundler(parcel) is doing the job behind the scene, Even before the whole code goes to Browser/JS-Engine it is Transpiled(it means this code is converted to the code that browser can understand).
* Then JS-Engine receives that Transpiled code so that browser can understand
* Then who is Transpiling the code Parcel itself? No no no Parcel is Gives the Responsibility of this Transpilation to a Package which is known as **Babel**
* Babel is a Package, Who is install Babel? We did not install then who install? => it’s a Parcel who install Babel
* Babel is present in node\_modules
* As soon as we write and save, So babe converted this code quickly to a code that JS engine will understand, Babel is transpiling ou code.
* *Babel in react is a crucial tool for React developers because it allows them to write code using the latest syntax and features while still ensuring that their code is compatible with all environments. This is particularly important because not all browsers support the latest JavaScript syntax, which could cause compatibility issues if developers tried to write React components using the latest syntax.*
* ( <https://www.scaler.com/topics/react/what-is-babel-in-react/> ) must visit and understand

**#** **Differece Between JSX and HTML**

* If we have to give an attributes to JSX we have to use camelCase (eg: className)
* (<https://www.freecodecamp.org/news/html-vs-jsx-whats-the-difference/> )

Q:- How to write JSX in multiple line

*// JSX in multipe line use () round braces*

*const* jsxHeading = (

  <h1 *id*="heading">

  Namaste React 🚀 from JSX

  </h1>

);

* We wrap JSX inside round brackets because Babel needs to understand from where JSX is starting and where is JSX ending.

**#Component**

* Everything is a component in React yes this statement is true
* A component is an independent, reusable code block which divides the UI into smaller pieces.
* For example, if we were building the UI of Twitter with React:
* 
* Rather than building the whole UI under one single file, we can and we should divide all the sections (marked with red) into smaller independent pieces. In other words, these are components.
* If we will see/create a webpage so a Button is a component, a Header is a component, a Footer is a component and a card, title, input box, search bar is component’s.
* There is 2 Types of Components
  + 1.Class Based Components (Old way of writing code)
  + 2.Function Based Components (new way of writing code)

**#Functional Component**

* React Functional Component is just a normal Javascript Function that must returns a React element (JSX).
* *// Functional Component*
* *const* HeadingComponent = () => {
* return <h1>Namaste React 🚀 From Functional Component</h1>;
* };
* *const* root = ReactDOM.createRoot(document.getElementById("root"));
* root.render(<HeadingComponent />);
* components are rendered like this *root.render(<HeadingComponent />);*
* this syntax that differentiate it’s a ReactElement or a Component that Babel understand
* always starts with a capital letter (naming convention)
* According to React's official docs, the function below is a valid functional component:
* function Welcome(props) {
* return <h1>Hello, {props.name}</h1>;
* }
* export default Welcome;
* Or
* const Welcome = (props) => {
* return <h1>Hello, {props.name}</h1>;
* }
* export default Welcome;
* To be able to use a component later, you need to first export it so you can import it somewhere else.
* (<https://www.freecodecamp.org/news/react-components-jsx-props-for-beginners/>) / (<https://codersera.com/blog/react-functional-components/> )

Q:- what is component Composition?

* Putting component inside a Component(nested) is known as Component Composition

*const* HeadingComponent = () => {

  return <h1>Namaste React 🚀 From Functional Component</h1>;

};

*const* MainComponent = () => (

  <div *className*="Container">

    <HeadingComponent />

    <h1>Namaste React from MAIN-COMPONENT 🚀🚀🚀🚀</h1>

  </div>

); // Javascript Expression

Q:- How to write Javascript inside JSX?

* We can run any piece of Javascript expression inside this { } curly braces
* *const* MainComponent = () => (
* <div *className*="Container">
* {heading} // Javascript Variable
* <h1>{10 + 2 + 196}</h1> // Javascript Expression
* {jsxHeading} // JSX Element
* <HeadingComponent /> // functional Component
* <h1>Namaste React from MAIN-COMPONENT 🚀🚀🚀🚀</h1>
* </div>
* );

Q:- What if API sent some Malicios Data.

* JSX is so amazing takes care of this injection attacks / will escape it
* When we getting some malicios data in {data}, whenever any data is wrapped inside these curly braces JSX wouldn’t blindly run it. It will sanitize your data whatever coming and then passing. It prevent cross-site scripting attacks for you.

Q:- How many ways of writing components inside component

*const* MainComponent = () => (

  <div *className*="Container">

    <HeadingComponent /> // 1

    <HeadingComponent></HeadingComponent> // 2

    {HeadingComponent()} // 3 calling functional component [at the end of the day it’s a function]

    <h1>Namaste React from MAIN-COMPONENT 🚀🚀🚀🚀</h1>

  </div>

);

**#Props in React Component**

* (<https://www.simplilearn.com/what-is-reactjs-props-article#:~:text=In%20ReactJS%2C%20the%20props%20are,components%20are%20read%2Donly%20components> )

**#Config Driven UI (understand this concept properly)**

* Configuration-driven UI, also known as config-driven UI or configuration-based UI - When you build real a world application so you want should your website work in many country or many place we control our fontend it is known as using Config-Driven UI . API or Backend Driven which is data Comming from API.
* or
* In a configuration-driven UI, the layout, styles, and other properties of UI elements are defined in a configuration file or database, which can be easily modified without requiring changes to the codebase. This approach makes it easier to customize the UI for different use cases or user groups, without the need for extensive coding.
* The configuration file or database may also define the data sources and the data to be displayed in the UI, as well as the interactions and behavior of the UI components. This allows for greater flexibility and adaptability of the UI to different use cases, as the configuration data can be easily modified or replaced without affecting the underlying application logic
* (<https://portal.gitnation.org/contents/config-driven-ui-using-reactjs>)

**FOOD ORDERING APP**

* First of all do planning before doing anything
* Don’t just blindly write code, plan it that how our app will look like

#Components in our App

* Header

. Logo

. Nav Items

* Body

. Search

. RestaurantContainer

. RestaurantCard

* Footer

. Copyright

. Links

. Address

. Contact

<div *className*="res-container">

          { resList.map( (*restaurant*) => ( <RestaurantCard *key*={restaurant.data.id} *resData*={restaurant} /> ) ) }

</div>

* resList.map function then this map function basically takes
* restList is an array so it will Loop over all of these restaurants.
* so for each restaurant we have to return a RestaurantCard
* we are passing dynamically data into this RestaurantCard *resData*={restaurant}
* ***whenever we are looping on to anything we have to always give a key over there***
* ***But why?***
* ***However, if you check the console log, you will see that there is a warning like, “Warning: Each child in a list should have a unique key prop.” Whenever you use a loop it is important to provide a unique key attribute. The reason is that React uses these keys to track if items were changed, added, or removed.***
* ( <https://www.telerik.com/blogs/beginners-guide-loops-in-react-jsx#:~:text=However%2C%20if%20you%20check%20the,changed%2C%20added%2C%20or%20removed>. )

**#Why not to use index as a key?**

* **React Official Page =>** We don’t recommend using indexes for keys if the order of items may change. This can negatively impact performance and may cause issues with component state. Check out Robin Pokorny’s article for an [in-depth explanation on the negative impacts of using an index as a key](https://robinpokorny.com/blog/index-as-a-key-is-an-anti-pattern/). If you choose not to assign an explicit key to list items then React will default to using indexes as keys.
* Here is an [in-depth explanation about why keys are necessary](https://legacy.reactjs.org/docs/reconciliation.html#recursing-on-children) if you’re interested in learning more.

**#Why ReactJs is Fast?**

* When we say react is fast. React is fast in DOM Manipulation, React is efficient in DOM Manipulation.
* This is the exact problem that react is solving, suppose we have to keep our data and UI layer Consistent with each other tied to each other that is where react comes into picture and all the other Frameworks
* Angular, React, Vue all the other frameworks are trying to solve this. That UI Layer and Data Layer are work each with each other right properly sync in.
* If my data changes my UI Layer changes too, this is the thing which all these frameworks are like trying to solve updating the DOM efficiently
* React is fast because it can do faster DOM Manipulation it can do Efficient Dom Manipulation there is something known as Virtul-DOM, Diff Algorithm, Reconciliation and many more..

**#Hooks in ReactJs**

* Normal Javascript Utility Function written by Facebook Developer.
* They have write this utility function inside React(node\_module – react), when we did npm install react we got all these utility function into our code
* We have to import these utility functions, there are multiple React Hooks and there are 2 very important React Hook
* useState()
* useEffect()
* (<https://www.freecodecamp.org/news/react-hooks-fundamentals/> )

**#useState in ReactJs**

* Whenever a state variable updates React re-render’s the component
* This is the power of state variable
* (<https://www.freecodecamp.org/news/usestate-hook-3-different-examples/> )

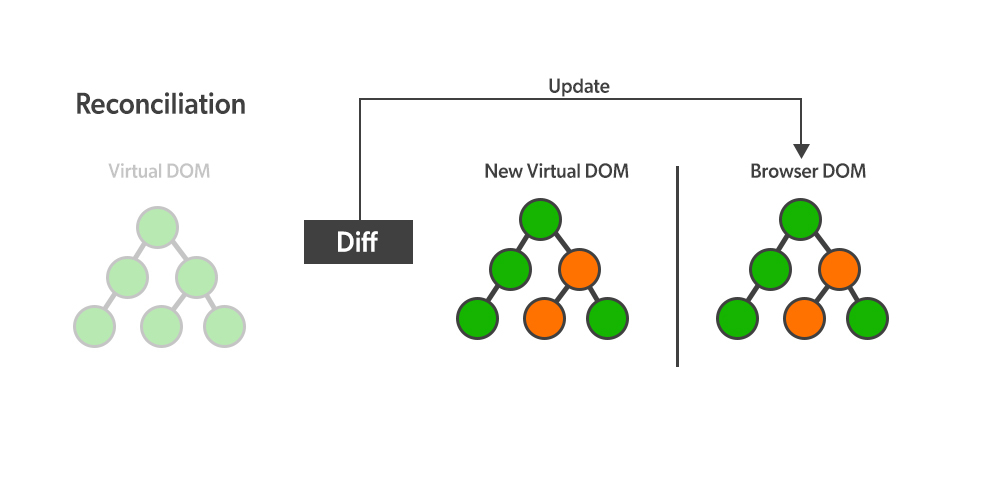
**#What is Reconciliation in ReactJs/React Fiber Architecture**

* **Introduction**
* **What is reconciliation?**
* **Reconciliation versus rendering**
* **Scheduling**
* **What is a fiber?**
* **Structure of a fiber(type & key, Child & Sibling, return, pendingProps & memoizedProps, pendingWorkPriority)**
* **(**[**https://github.com/acdlite/react-fiber-architecture**](https://github.com/acdlite/react-fiber-architecture) **)**
* React uses Reconciliation Algorithm is Also Known As React-Fiber, this has comes up in React16
* Fiber is a reconciliation algorithm used in the popular JavaScript library React to efficiently update a web application’s user interface (UI). It was introduced in [version 16.0 of React](https://reactjs.org/blog/2017/09/26/react-v16.0.html) in 2017 and has significantly improved the performance of React applications.
* In React, when the state of a component changes, the component needs to update its UI to reflect the new state. This process of updating the UI is called reconciliation.
* React uses a [Virtual DOM (VDOM)](https://reactjs.org/docs/faq-internals.html) to perform reconciliation, which is used to compare a component’s current and previous states.

**What is the Virtual DOM?**

* Virtual-DOM is not Actual-DOM, Virtual-DOM is representation of an Actual-DOM, VirtualDOM basically an Object
* The VDOM is a lightweight in-memory representation of the actual DOM.
* When the state of a component changes, React compares the VDOM of the last and current states and calculates the minimum number of DOM operations required to update the actual DOM to match the current VDOM.

**What is Diffing Algorithm in React ?**

* Diffing Algorithm in React JS differentiates the updated and previous DOM of the application. DOM stores the components of a website in a tree structure. React uses virtual DOM which is a lightweight version of the DOM. The only difference is the ability to write the screen like the real DOM, in fact, a new virtual DOM is created after every re-render.
* Diffing short for Differences Algorithm is used to differentiate the DOM Tree for efficient updates. React utilize diffing algorithm to identify the changes in the newly created virtual dom and previous version of dom after any changes are made.
* (<https://www.geeksforgeeks.org/what-is-diffing-algorithm/> )
* What is Diff Algorithm => it Basically finds out the difference between two virtual DOM’s, Updated VDOM and the previous VDOM

**The Catch**

* This helps reduce the number of DOM manipulations and improve the application’s performance. However, there are specific scenarios where the reconciliation process can become inefficient.
* *For example, suppose a component has many elements that need to be updated. In that case, the reconciliation process can take a long time and cause the UI to become unresponsive – This is where Fiber comes in.*

**What is Fiber?**

* Fiber is a new reconciliation algorithm introduced in React 16.0 that aims to improve the performance of React applications by making the reconciliation process more efficient.
* It does that by allowing the reconciliation process to be broken down into smaller chunks and scheduled over multiple frames rather than being completed in a single frame.
* Fiber divides the reconciliation work into smaller units called “fibers“.

**What is a Fiber?**

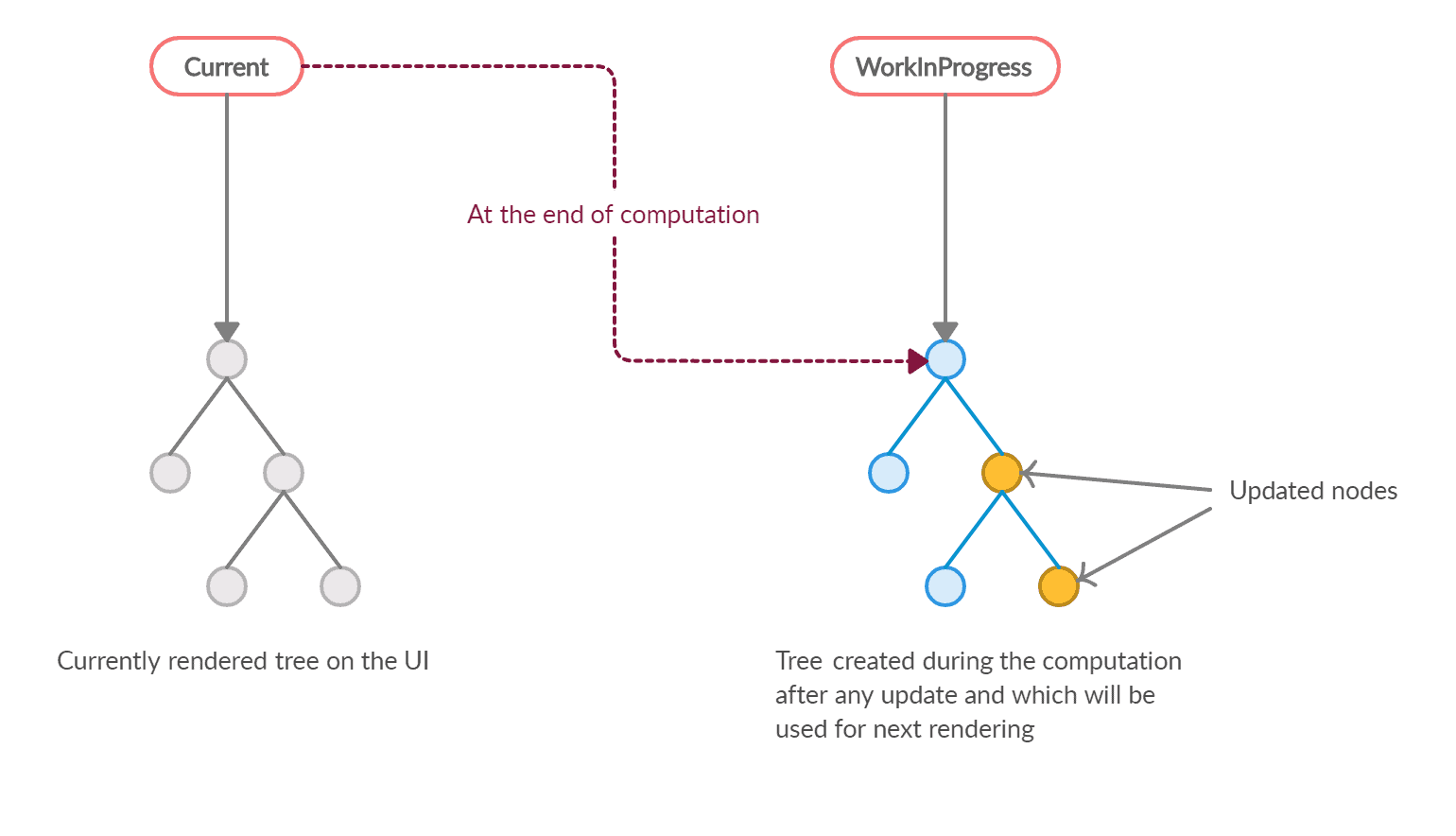
* Each Fiber represents a single element in the VDOM tree, and the reconciliation process is performed on each Fiber individually.
* This allows React to prioritize the reconciliation of certain fibers over others, depending on the importance of the updates.
* For example, suppose a component has many elements that need to be updated. In that case, React can prioritize the reconciliation of the elements that are visible to the user while deferring the reconciliation of the other elements until later.
* This helps ensure that the UI remains responsive even when there are a large number of updates.

**Fiber in Asynchronous Programming**

Fiber also introduces a new concept called “suspense”. Suspense allows React components to “wait” for a specific condition to be met before rendering.

This can improve the performance of applications that rely on asynchronous data, such as fetching data from a server.

**How does Fiber work?**



**An Introduction to React Fiber - The Algorithm Behind React (**<https://www.velotio.com/engineering-blog/react-fiber-algorithm> )

* Fiber works by maintaining a linked list of fibers called the “fiber tree“.
* Each Fiber in the tree represents a single element in the VDOM tree and contains information about the element, such as its type, props, and state.
* When a component’s state changes, React begins the reconciliation process by creating a new fiber tree based on the current state of the component. It then compares the new fiber tree to the previous fiber tree to determine the minimum number of changes required to update the actual DOM.
* During the reconciliation process, React traverses the fiber tree and updates each Fiber individually. It does this by starting at the root fiber and working its way down the tree, updating each Fiber in a depth-first order.
* As it traverses the fiber tree, React can pause the reconciliation process at any point and schedule the remaining work for the next frame. This allows React to prioritize the reconciliation of certain fibers over others and ensure that the UI remains responsive.

**Benefits of Fiber**

Fiber brings several benefits to React applications, including:

**Improved Performance**

As mentioned earlier, Fiber allows the reconciliation process to be broken down into smaller chunks and scheduled over multiple frames, which helps improve React applications’ performance, especially when a large number of elements need to be updated.

**Better user experience**

By prioritizing the reconciliation of certain fibers over others, Fiber helps ensure that the UI remains responsive, even when there are a large number of updates. This leads to a better user experience.

**Asynchronous rendering**

The introduction of Suspense in Fiber allows React components to “wait” for a specific condition to be met before rendering, which can be used to improve the performance of applications that rely on asynchronous data.

*If you’d like to know more about Suspense in React, which is related to the Asynchronous rendering mentioned earlier, I’d recommend checking the article we’ve written*[*here*](https://upmostly.com/tutorials/how-to-add-lazy-loading-to-react-components)*.*

**Better concurrency**

Fiber allows React to interrupt the reconciliation process at any point and schedule the remaining work for the next frame. This helps improve the concurrency of React applications, as it allows React to perform updates in parallel with other tasks.

**Summary**

In summary, Fiber is a reconciliation algorithm used in React to improve the performance of web applications.

It does this by allowing the reconciliation process to be broken down into smaller chunks and scheduled over multiple frames and by introducing the concept of suspense, which allows React components to “wait” for a specific condition to be met before rendering.

Fiber brings several benefits to React applications, including improved performance, a better user experience, asynchronous rendering, and better concurrency.