Some basic stuff regarding HTML and DOM:

What is the relationship between DOM and HTML… How they are related to each other?

The Document Object Model (DOM) and an HTML file serve different purposes but are closely related in the context of web development. Let's clarify the distinctions between the DOM and an HTML file:

**HTML File:**

1. **Static Document:**
   * An HTML (Hypertext Markup Language) file is a plain text file that contains markup tags defining the structure and content of a web document.
2. **Human-Readable Code:**
   * HTML files are created and edited by developers using a text editor. They consist of tags (e.g., **<html>**, **<head>**, **<body>**, **<p>**) that describe the elements and content of a web page.
3. **Stored on Disk:**
   * HTML files are static and are stored on the server's file system or distributed to clients. When a user requests a webpage, the server sends the corresponding HTML file to the user's browser.
4. **Not Interactive:**
   * An HTML file alone does not have dynamic or interactive capabilities. It represents the initial, static structure of a webpage before any user interactions or modifications.

**Document Object Model (DOM):**

1. **Dynamic Representation:**
   * The DOM is a dynamic, in-memory representation of an HTML document that a browser creates when it loads an HTML file.
2. **JavaScript Interaction:**
   * The DOM is manipulated and interacted with using JavaScript. JavaScript code can modify the DOM to dynamically update the content, structure, and style of a webpage in response to user actions or other events.
3. **Tree-like Structure:**
   * The DOM represents the HTML document as a tree-like structure where each HTML element is a node, and relationships between elements are defined by parent-child relationships.
4. **Live and Responsive:**
   * Unlike an HTML file, the DOM is responsive to changes made by JavaScript. When JavaScript modifies the DOM, the changes are immediately reflected in the displayed webpage without requiring a full page reload.

**Relationship:**

* **HTML as Source Code:**
  + The HTML file serves as the source code for a webpage. It provides the initial structure and content, including the markup tags that define elements and their relationships.
* **DOM as Runtime Representation:**
  + The DOM is a runtime representation of the HTML document. When a browser loads an HTML file, it creates the DOM, which is then used to render the webpage and respond to user interactions.

In summary, an HTML file is a static document that provides the initial blueprint for a webpage, while the DOM is a dynamic, in-memory representation that the browser creates based on the HTML file. The DOM can be manipulated and updated in real-time using JavaScript, enabling interactive and responsive web applications.

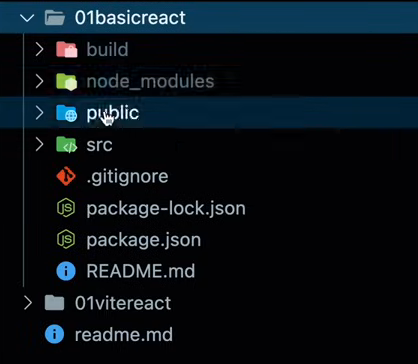
**IS an HTML element converted into javascript object at the end of the day while rendering?**

HTML elements themselves are not directly converted into JavaScript objects, JavaScript can interact with the DOM, treating DOM nodes as objects with properties and methods. DOM API presents methods and properties to manipulate them .This interaction allows developers to dynamically modify the content and structure of a web page, creating dynamic and interactive user interfaces.

**The Foundation**

Understanding the React flow and Structure :

**Understanding the Basic Folder structure of react jitha aapn kam karnar:**



Jevde pn dependencies mala package.json madhe distayt te download or install houn **node\_modules** madhe bastat

Most of the kam je asnar ahe te **source folder** madhe asnar kiva **public folder** madhe asnar

Talking in context of create-react-app wala react project

**In Public Folder:**

Public folder madhe important file ahe index.html which is where all the component code will be painted by react as and when needed …during runtime

This is the reason why it is called as single page application

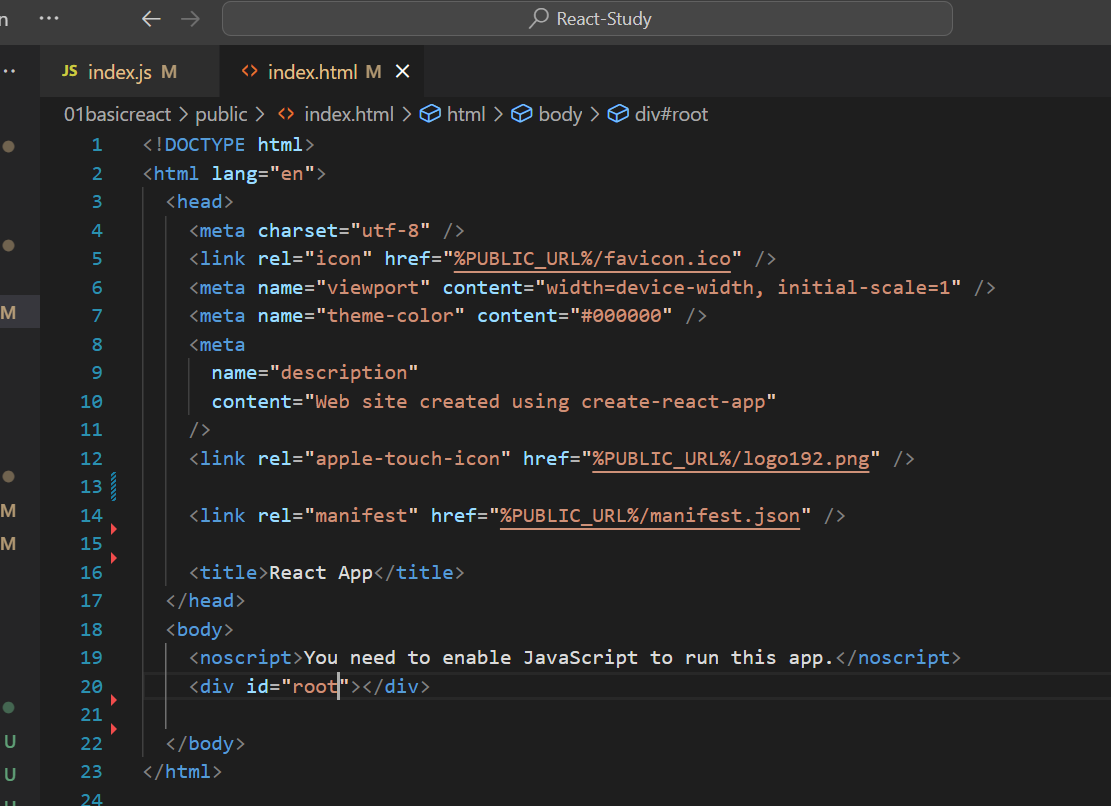
**In Source Folder:**

Index.js ani App.js are important files in this

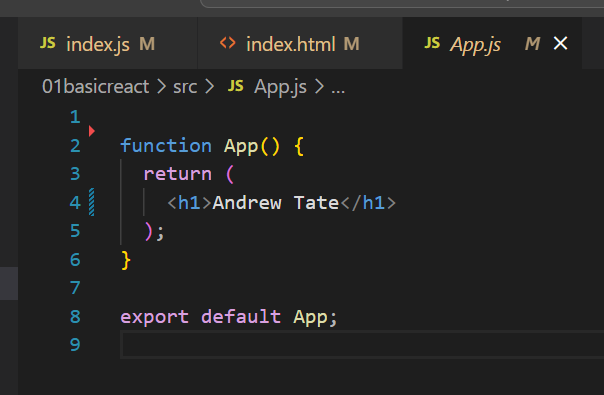
Index.js pasun ek virtual dom create hotay memory madhe and which maps whatever components are there in the browser dom.

App.js pasun aapn index.js madhe components pathvnar which will render them.

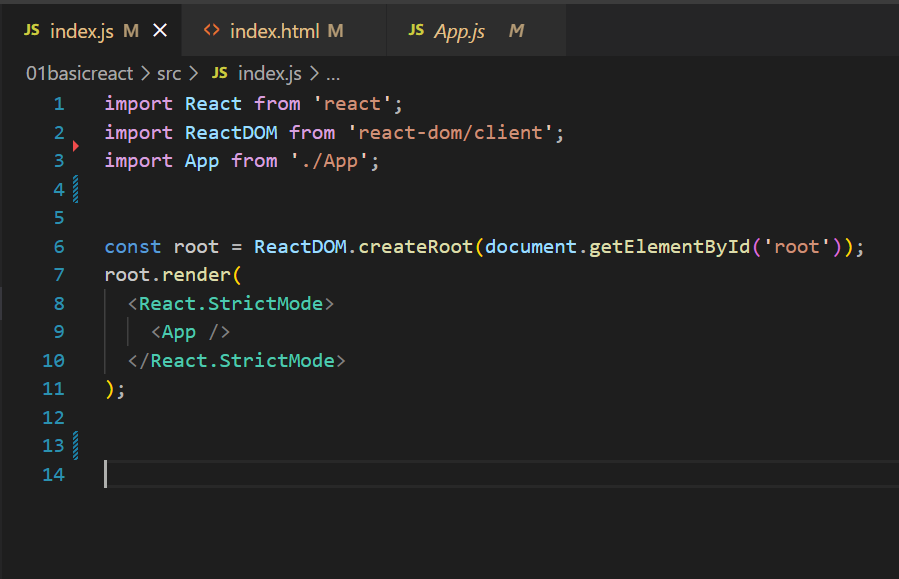
Index.html



App.js: Ithun aapn index.js la components export karnar…mhnje tithe index.js

Virtual dom madhe render karu shaknar

Index.js madhe virtual dom vrti render vhyla sodlay



**Virtual DOM cha reference aapn root variable madhe ghetoy and then aapn tyala render karyla sangtoy**

Jr webpage load karat astana ky langar asel tr react script te add karat jate necessary goshti (create-react-app chya context madhe)

**React madhe components banavna means what ?**

Eka jsx kiva js file madhe function banvun HTML cha code return karne is what creating components means

**Creating First React Component:**

**We will make projects in Vite+ react environment**

Steps:

1. Create a new jsx file which will denote our component
2. In that create a function which returns HTML … export default that
3. Import that in the App.jsx file and include that in an appropriate area in tag format
4. Run the application

Learnings:

It’s a good practice to keep the file name and component name starting with a capital letter

**Understanding JSX via creating our own react library**

**Through code we concluded the following**

1. Practically saw whatever the JSX which our functional components are returning they are first transpiled into JS objects or **react elements** (where they are loaded with properties which might be required for the future optimizations or algorithms ) and then given for the virtual DOM to render it
2. when you return JSX from a functional component, React internally processes it, converts it into React elements(JS object), and uses these elements to build and update the virtual DOM, which eventually leads to the efficient rendering of the actual DOM.
3. Since at the end we have an intermediary state where babble converts the JSX into React elements … **It is possible to create our own react elements and then give for the virtual dom to render it.**
4. **Another point to be noted here is that we can inject JS variables in our JSX code. JSX supports that functionality as well… Note here that only evaluated expression can be written inside the JSX**

**4.1 Custom react element creation Syntax:**

const element = React.createElement(

tag name,

props which is an object used to set the properties of the tag,

innerHTML ,

variables which can be injected

)

**How JSX and HTML are different?**

JSX and HTML look very similar, but they are distinct in a few key ways:

1. **Syntax:**
   * **JSX:** JSX is a syntax extension for JavaScript. It allows you to write XML-like code (HTML-like) within your JavaScript files. JSX tags resemble HTML, but they are not exactly the same.

const element = <div>Hello, JSX!</div>;

* + **HTML:** HTML is a markup language used to structure content on the web. It is a standalone language with its own syntax.

<div>Hello, HTML!</div>

1. **Attribute Names:**
   * **JSX:** Attribute names in JSX use camelCase, similar to how you would write attribute names in JavaScript.

const element = <input type="text" onChange={handleInputChange} />;

* + **HTML:** Attribute names in HTML are case-insensitive and typically written in lowercase.

<input type="text" onchange="handleInputChange()" />

1. **Class vs. className:**
   * **JSX:** To specify CSS classes in JSX, you use the **className** attribute.

const element = <div className="myClass">Styled with JSX</div>;

* + **HTML:** In HTML, you use the **class** attribute to specify CSS classes.

<div class="myClass">Styled with HTML</div>

1. **Comments:**
   * **JSX:** Comments in JSX are written as curly-brace-enclosed JavaScript comments.

const element = ( <div> {/\* This is a JSX comment \*/} Hello, JSX! </div> );

* + **HTML:** HTML comments are written using **<!-- -->**.

<div> <!-- This is an HTML comment --> Hello, HTML! </div>

1. **Usage:**
   * **JSX:** JSX is primarily used in React applications to describe the structure of UI components within JavaScript files.
   * **HTML:** HTML is the standard markup language for web pages and is used in static HTML files.

Despite these differences, JSX is designed to be familiar to developers who are used to writing HTML. **JSX gets transpiled to JavaScript by tools like Babel before being rendered by the browser. It allows developers to write UI components in a syntax that closely resembles HTML while leveraging the power and expressiveness of JavaScript.**

**Describe the internal working of Browser DOM and Virtual DOM?**

The Document Object Model (DOM) is a programming interface for web documents. It represents the structure of a document as a tree of objects, where each object corresponds to a part of the document, such as elements, attributes, and text. The DOM provides a way (by providing functions) for programs to manipulate the structure, style, and content of web documents dynamically. Here's a brief overview of how the browser DOM works:

### Browser DOM:

1. **Parsing HTML:** When a web page is loaded, the browser parses the HTML document to create the DOM tree. This involves breaking down the HTML into a hierarchical structure of elements.
2. **Object Representation:** Each HTML element, attribute, and piece of text is represented as an object in the DOM tree. These objects are organized in a hierarchical structure, reflecting the nesting of HTML elements.
3. **JavaScript Interaction:** JavaScript can be used to manipulate the DOM dynamically. **Developers can use methods and properties provided by the DOM API to add, modify, or remove elements and attributes. This allows for the creation of interactive and dynamic web pages.**
4. **Rendering:** The browser uses the DOM tree to render the web page. Styles applied through CSS are also considered during rendering, resulting in the final visual representation seen by users.

### Virtual DOM:

The Virtual DOM is a concept often associated with libraries and frameworks like React. It is an abstraction of the browser's DOM and is used to optimize the updating of the user interface. Here's how the Virtual DOM typically works:

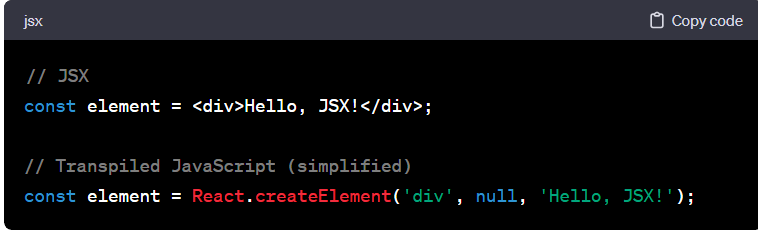
1. **Component Render:** In frameworks like React, user interfaces are structured as components. Each component has a representation in the Virtual DOM.
2. **Virtual DOM Tree:** When a component needs to update, React creates a virtual representation of the DOM tree, known as the Virtual DOM. This is a lightweight copy of the actual DOM.
3. **Diffing Algorithm:** The Virtual DOM uses a "diffing" algorithm to compare the new Virtual DOM with the previous one. It identifies the differences (or "diffs") between the two versions.
4. **Minimal Updates:** Instead of updating the entire real DOM with every change, React determines the minimal set of changes needed to bring the real DOM in sync with the new Virtual DOM.
5. **Batch Update:** React batches these changes and updates the real DOM in a more optimized way, reducing the number of direct manipulations to the actual DOM.

By using the Virtual DOM and optimizing the update process, frameworks like React aim to improve performance and provide a smoother user experience on web applications.

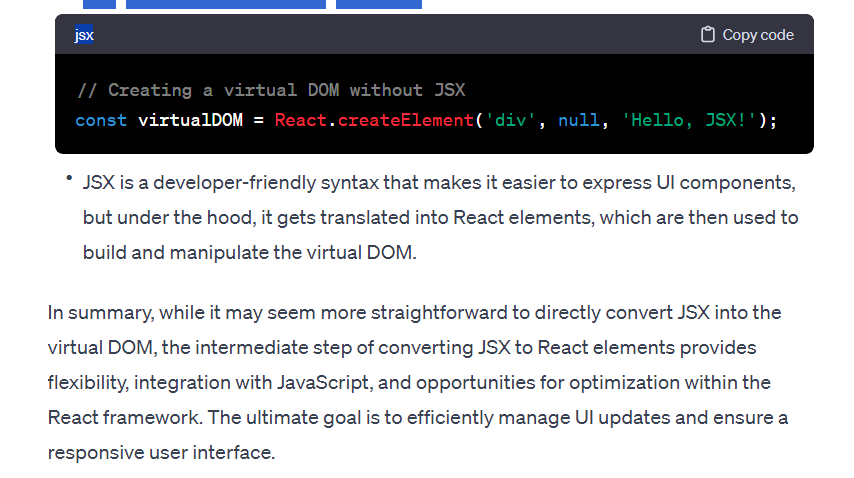
One question which i am having is that if HTML elements can easily be built into DOM tree ... then why JSX (which contains HTML tags) is firstly converted into javascript objects or react elements and then it is added to the virtual dom... why it is not directly converted into virtual dom through tags only .... Am i thinking in the right direction or is there any inconsistency in my understanding? (Read reasons for JSX transformations)

Your question is insightful, and there's no inconsistency in your understanding. Let me clarify the process and explain why JSX is first converted into React elements before being added to the virtual DOM.

1. **JSX to React Elements:**
   * JSX is a syntactic extension for JavaScript that looks similar to HTML. However, browsers don't understand JSX directly; they only understand HTML and JavaScript.
   * React, being a JavaScript library, provides a way to represent UI components as JavaScript objects called React elements.
   * JSX is processed by a tool like Babel, which transforms it into JavaScript code that creates React elements.

****

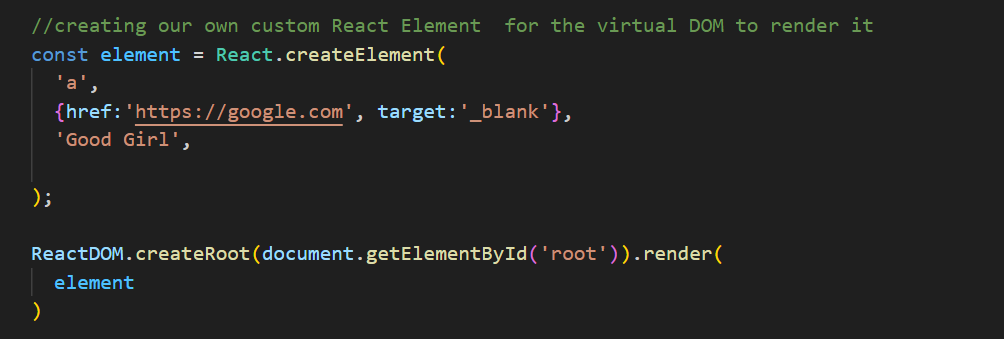
1. **Reasons for JSX Transformation:**
   * **Integration with JavaScript:** JSX allows seamless integration with JavaScript. By converting JSX into JavaScript objects (React elements), React can leverage the full power of JavaScript to manage and manipulate components.
   * **Optimizations and Diffing:** Converting JSX to React elements before reaching the virtual DOM allows React to perform optimizations during the diffing process. React can efficiently compare the new and old virtual DOMs, identifying which parts of the DOM need updating without directly manipulating the actual DOM for every change.
   * **Component Lifecycle and State Management:** Converting JSX to React elements enables React to manage component lifecycles, state, and other aspects of component behavior. It provides a consistent and predictable way to handle component updates and rendering.
2. **Direct Virtual DOM Creation:**
   * While React introduces an additional step in converting JSX to React elements, it's worth noting that you can create a virtual DOM directly without using JSX by using the **React.createElement** function.



a

**Creating Custom react elements**

We learnt, At the end JSX is converted into react elements / JS objects … before being rendered into virtual DOM … **You can even create your own react elements and give for the virtual DOM to render it…**



**Notes related to hooks are written in the code file 02CounterUseState**

**What are hooks?**

They are basically functions which help us utilize the react features

**Some few important points from sir:**

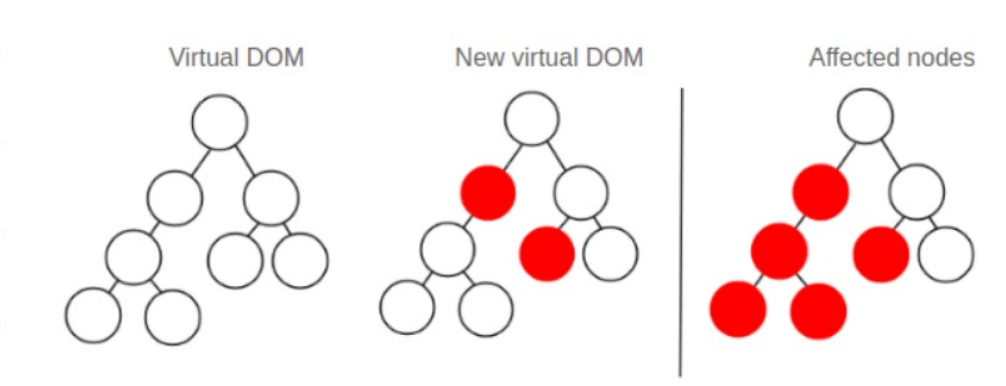
1. We can keep Javascript variables different and state variables different….
2. Jr UI madhe update honar asel tr React will only be responsible to do that ...
3. Jr library banavnyat yet asel tr modular way ne banavli jate ... such that ek task eka function kade dila jato... which on later helps in debugging as well
4. Hooks la actually study kasa karna he yayla pahije and use kasa karna hechyavr focus pahije
5. **The change in the state(variable) will be propagated to all the UI components of the DOM using useState hook**

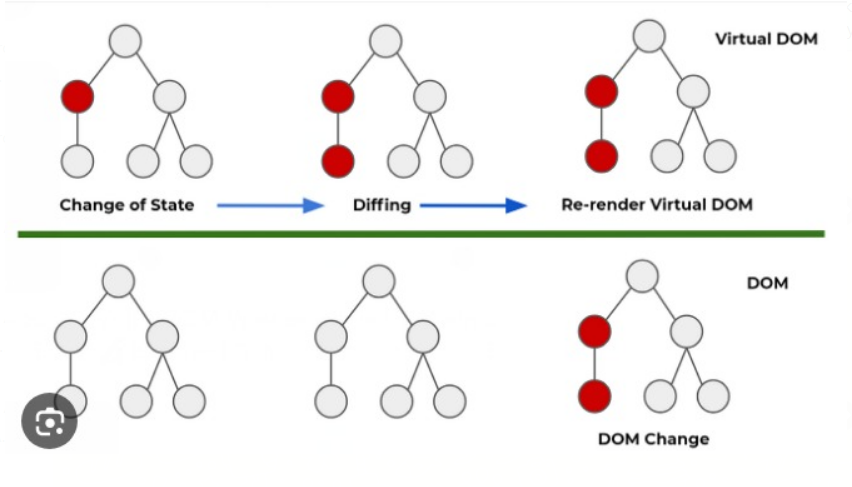
Practically implemented a counter project using useState hooks

**Virtual DOM and React Fibre:**

**What is reconciliation process?**

We can say that Reconcilation is a process which internally uses diffing algorithm.

Diffing algo identifies all the differences between the the old Virtual DOM and new Virtual DOM 

****

Reconciliation is the algorithm behind what is popularly understood as the "virtual DOM." A high-level description goes something like this: when you render a React application, a tree of nodes that describes the app is generated and saved in memory. This tree is then flushed to the rendering environment — for example, in the case of a browser application, it's translated to a set of DOM operations. When the app is updated (usually via setState), a new tree is generated. The new tree is diffed with the previous tree to compute which operations are needed to update the rendered app.

**What is react fibre?**

React Fiber is a term associated with the internal implementation of React, a JavaScript library for building user interfaces. React Fiber was a reimplementation of the core algorithm used by React to enable better performance, improved rendering, and the ability to handle concurrent updates.

React Fiber was introduced to address some limitations in the original React stack, particularly in terms of handling large and complex user interfaces. The goal was to make React more adaptable to modern web applications with dynamic and interactive user interfaces.

Some key features and goals of React Fiber include:

1. **Incremental rendering:** React Fiber enables the rendering of components in a more incremental and interruptible way. This means that React can work on rendering a portion of the UI, yield to other high-priority tasks, and then resume rendering.
2. **Better support for asynchronous rendering:** React Fiber is designed to better handle asynchronous updates, allowing for smoother interactions and improved performance.
3. **Improved priority and scheduling:** React Fiber introduces a new scheduling algorithm that allows React to prioritize and schedule updates based on their urgency, which is crucial for creating responsive user interfaces.
4. **Support for concurrent rendering:** React Fiber lays the groundwork for concurrent rendering, which means that React can potentially work on multiple tasks concurrently, leading to better overall performance.

It's important to note that React Fiber is not a separate version of React but rather a new internal architecture. From a developer's perspective, most of the API and usage of React remained the same. React Fiber improvements were intended to be transparent to developers, providing a better user experience without requiring significant changes in the way React applications are built.

**What is traditional rendering and incremental rendering?**

Traditional Rendering vs. Incremental Rendering:

Traditional Rendering:

In a traditional rendering approach, when an update occurs in a React application (such as a state change or a prop update), React re-evaluates the entire virtual DOM and compares it with the previous one to determine the differences (reconciliation). Once the differences are identified, React updates the actual DOM to reflect the new state of the application.

This process is synchronous and can be time-consuming, especially for large and complex UIs. During this rendering process, the UI might become unresponsive, causing delays in user interactions.

Incremental Rendering with React Fiber:

React Fiber introduces the concept of incremental rendering to make the rendering process more flexible and interruptible. Instead of completing the entire rendering process in one go, React can now work on rendering a small "chunk" or portion of the UI and then pause or yield to other tasks.

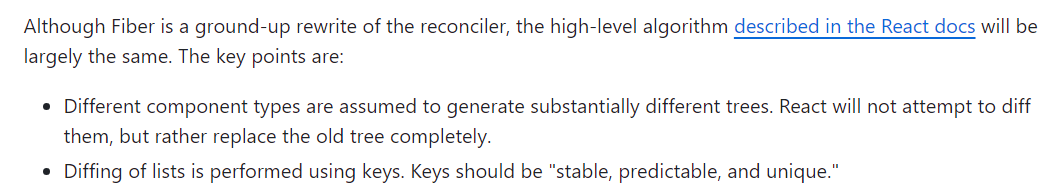
Here's how incremental rendering works in React Fiber:

1. Work in Units (Fibers): React Fiber breaks down the work into smaller units called "fibers." Each fiber represents a unit of work, such as rendering a component or updating the DOM.
2. Priority and Scheduling: Each fiber has an associated priority, indicating its urgency or importance. React can schedule and work on fibers based on their priority. High-priority tasks (e.g., user interactions) are handled first.
3. Yielding and Resuming: While rendering a fiber, React can yield control to the browser, allowing it to perform other high-priority tasks or respond to user inputs. This makes the rendering process interruptible.
4. Time Slicing: React Fiber introduces the concept of time slicing, where the work is divided into small time slices. The renderer works on a slice, yields, and then picks up where it left off in the next slice. This helps ensure a more responsive user interface.

Benefits of Incremental Rendering:

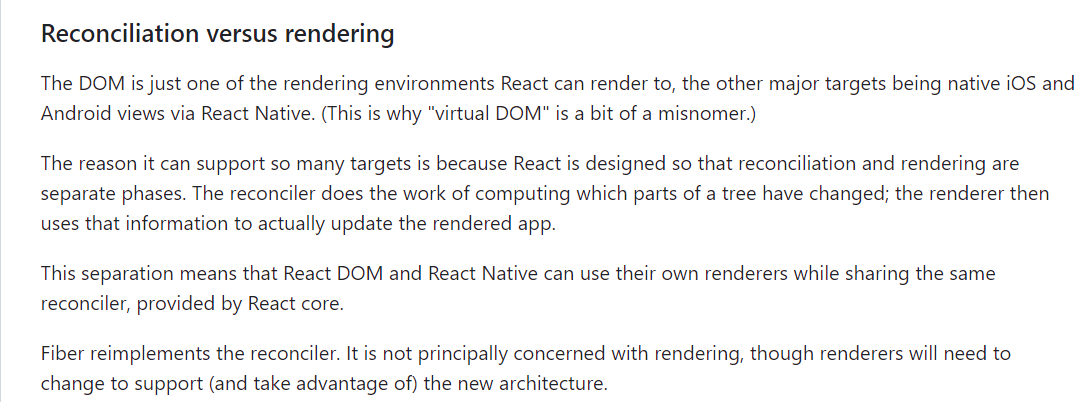
1. Improved Responsiveness: Incremental rendering allows React to respond to user interactions more quickly, preventing the UI from freezing during lengthy rendering tasks.
2. Better Concurrency: The ability to pause and resume rendering tasks enables React to work on multiple tasks concurrently, improving overall performance.
3. Adaptability to Different Environments: Incremental rendering makes React more adaptable to different environments, such as low-powered devices or slower network connections, by allowing the application to remain responsive.

In summary, React Fiber's incremental rendering is a strategy to make the rendering process more flexible and responsive by breaking it into smaller units, assigning priorities, and allowing interruptions for other high-priority tasks. This contributes to a smoother user experience in React applications.



If suppose complex component asel tr whole virtual tree is replaces with the new one

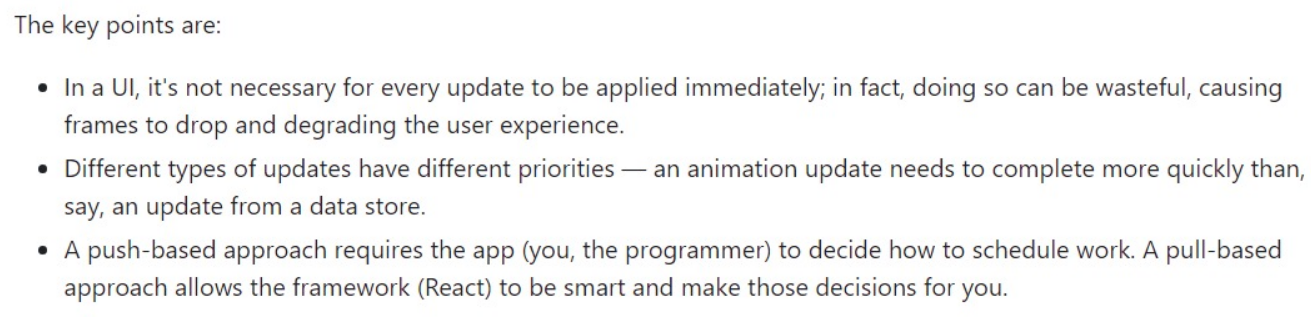
**Something about reconciliation and rendering**



Reconciler and rendering phases …firstly reconciliation happens which generates information on which part of tree was updated

And the renderer uses that info to actually render the change

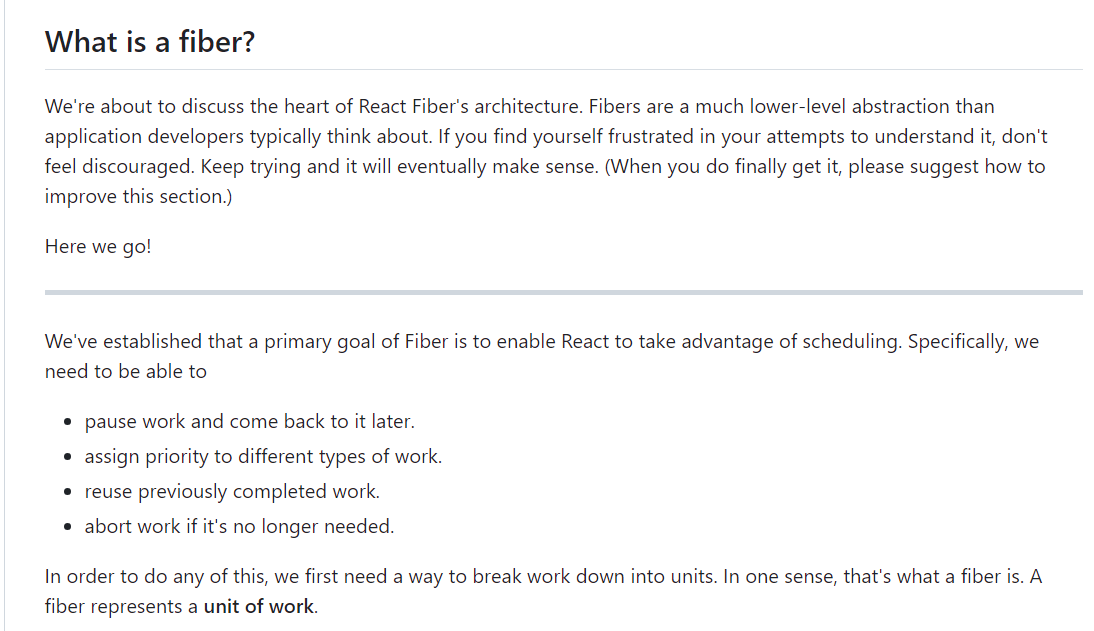
**Something about Scheduling:**

****

first point : jaruri nahi ki instantaneously new virtual dom create karycha ... thoda time thambun nantr bulk madhe pn states update karu shakto

Second point: Kahi updates chi priority jasta aste …

1. **Update Priorities in UI:**
   * Imagine your UI as a series of tasks or updates that need to be applied.
   * Some updates are more time-sensitive than others. For example, an animation update needs to happen quickly to ensure smooth motion, while an update from a data store might not be as urgent.
2. **Push-Based Approach:**
   * In a push-based approach, you, as the programmer, decide when and how to schedule updates.
   * It's like you actively "push" or initiate the updates based on your programming decisions.
   * You have to manage the timing and order of updates, which can be challenging, especially in complex applications.
3. **Pull-Based Approach:**
   * In a pull-based approach, the framework (like React) takes charge of scheduling updates.
   * Instead of you pushing updates, the framework pulls or decides when to apply updates based on their priorities.
   * React, being smart about it, can prioritize updates such as animations over less critical updates.
4. **Why Prioritize Updates:**
   * Not every update needs to be applied immediately. For instance, if you're scrolling through a list, updating every millisecond might be wasteful and degrade the user experience.
   * Prioritizing updates ensures that more critical tasks, like animations, get processed quickly, providing a smoother user experience.



Some other important points from sir

1. React madhle internal concepts and ideas change zale ahet ...
2. browser jyaveli aapn refresh karto .. tyaveli browser Dom parat repaint hoto / reconstruct hoto...
3. Ui update instantaneously hoat nahi ... only changes are being reflected from virtual Dom to actual dom
4. As a good engineer, you should be interested in performance

**Props (First see the code file tailwindcssprops)**

Props are responsible for making the components reusable

Some imp points from sir:

* 1. props initially gives an empty object which is also linked with one of the prototype object
  2. props madhe key value aapn string directly pass karu shaktoy but when it comes to passing arrays or objects

 or functions or variables we have to pass the reference of it

* 1. Its a good practice ki props chya thikane object destructuring karne