**Git Commands**

echo “# React\_Counter” >> README.md

git init

git add README.md

git commit -m “first commit”

git branch -M main

git remote add origin <https://github>.com/MoizSethi/Repository.git

git push -u origin main

**React Installation**

**NPX Node Package Executor**

**NPM Node Package Manager**

**Old Method Non-Optimized with Node Pack Non-Restricted Version We can use both upper and lower cases in Naming**

npx create-react app AppName

**TO RUN APP**

npm run start

**The new method is React with Vite Optimized Without Node Only Necessary Dependency To React Strict with Syntax App Name must Start with Capital Letter.**

npm create-vite@latest

Once Installed we need to run an extra command

**NPM install or NPM i**

To install the node package for the project

**TO RUN APP**

npm run dev

**Hooks**

React Hooks are a new addition to React that allows you to use state and other React features without writing a class component. JavaScript functions known as “Hooks” allow you to “hook into” React state and lifecycle features from function components.



React controls UI updation we use hooks to Control states.

**Important Notes**

**TOPICS:**

* Virtual DOM
* React Fiber
* Reconciliation

**Steps browser follows to render our website**

1. **Parsing HTML [**Browser parses HTML and stores it as a tree structure also known as DOM. DOM methods allow programmatic access to the tree.**]**
2. **Parsing CSS [**The browser parses our CSS and stores it in memory as CSSOM (CSS Object Model). It is a web API that manipulates the CSS of our website.**]**
3. **Creating Render Tree [**The browser uses DOM and CSSOM to create a render tree. Render Tree represents everything that will be rendered on the browser (HTML nodes with their styles).**]**
4. **Layout Render Tree [**Place Elements according to their size and position**]**
5. **Painting [**Style the nodes**]**

**Virtual DOM:**

“createRoot” is an element that creates a virtual DOM or replicates the DOM structure. It will Compare the Main DOM with the Virtual DOM. When we refresh or change the page it will repaint the whole DOM. The benefit of using Virtual DOM when we refresh or navigate the page will only replace the DOM where necessary.

**React Fiber Architecture**

React Fiber is an ongoing reimplementation of React’s core algorithm.

**Goal**

Suitability for areas like Animation Layout and Gestures.

**Features:**

Pause, Abort, or reuse work as a new update in; the ability to assign priority to different types of updates; and new concurrency primitives.

**What is reconciliation?**

***Reconciliation***

The algorithm React uses to diff one tree with another to determine which parts need to be changed.

***Update***

A change in the data used to render a React app. Usually the result of `setState`. Eventually results in a re-render.

Reconciliation is the algorithm behind what is popularly understood as the “virtual DOM”.

Although Fiber is a ground-up rewrite of the reconciler, the high-level algorithm described in the React Docs will be largely the same.

The Key Points:

* Different component types are assumed to generate substantially different trees React will not attempt to diff them but rather replace the old tree completely.
* Diffing of the list is performed using keys. Keys should be “Stable, predictable, and unique.”

**Important Terminologies  
Reconciliation** is keeping 2 DOM Trees in sync by a library like React DOM. It is done by using a Reconciler and a Renderer.

**Reconciler** uses the Diffing Algorithm to find differences between the Current Tree and Work in Progress Tree and sends computed changes to the Renderer.

The **Renderer**is the one that updates the app’s UI. Different devices can have different Renderers while sharing the same Reconciler.

**The old reconciler** has been given the name **Stack Reconciler**. The problem with this approach is that it was **synchronous**and much execution happens all at once. It could cause frame drops in animations and poor UI experience. It used to have only 1 Virtual DOM tree, making some features like **Suspense**and **Concurrent Mode** impossible to implement because they rely on Reconciler’s ability to work asynchronously.

**React 16**

Uses a new data structure called fiber. Hence it is called a Fiber Reconciler. The main aim was to make the reconciler asynchronous and smarter by executing work based on priority.

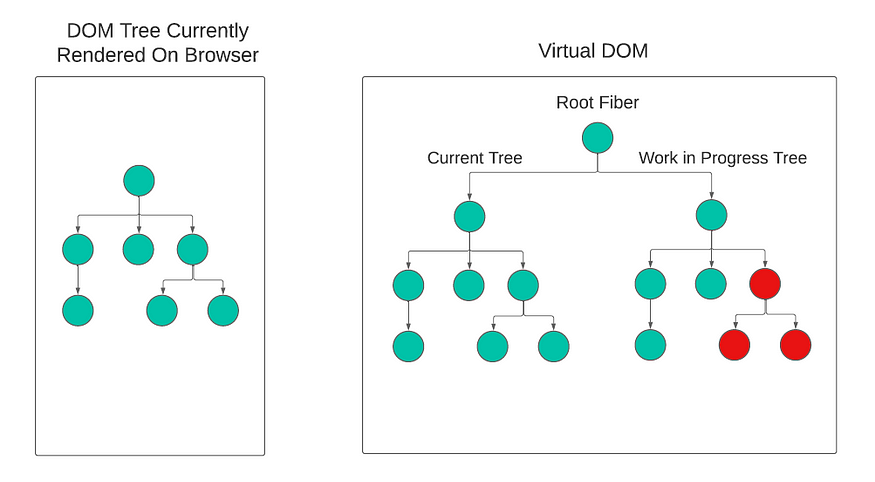
**React Fiber** needs to be asynchronous by taking advantage of cooperative scheduling and should be able to do:  
1. Pause work and come back to it later  
2. Assign priority to different types of work  
3. Reuse previously completed work  
4. Abort work if it’s no longer needed

**fiber** is a JavaScript Object that represents **a unit of work**. For each React Component and Element, React creates its fiber object. It has a one-to-one relationship with an instance and manages the work for an instance. And also keeps track of its relationships to other fiber objects in the Virtual DOM tree.

**Priority List:**React Fiber gives different importance to different updates and executes them according to their priority.

**Reconciliation Process**

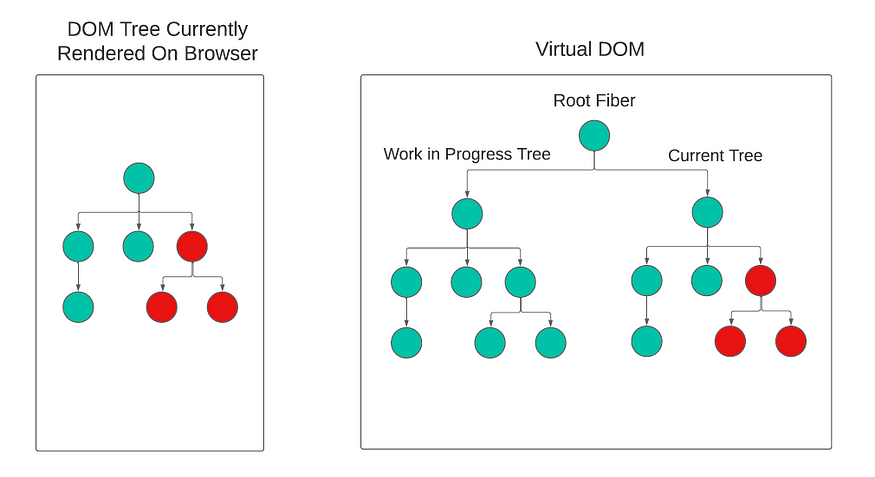
Browser’s Main thread is used for creating a Work In Progress Tree with React, processing user events, repainting, etc.



During Phase 1

Let’s see how all of this fits together:

1. When we make changes to our state, React waits for the Main Thread to become idle and then starts building a Work in Progress (WIP) Tree on it.
2. The WIP Tree is built using fiber and the Tree’s structure matches with the structure of the Components in our code.
3. This phase (**render/reconciliation phase**) of building the WIP Tree and figuring out changes to be made is **asynchronous** and can be paused if the Main Thread has some other work to complete. In that case, the main thread starts working on those updates according to **their priority in the Priority List**. Once the Main thread is idle again, it resumes building the WIP Tree where it last left it off.
4. The second phase (**commit phase**) starts once the whole WIP Tree is completed, this phase is **synchronous**and **can’t be interrupted**. In this phase, React will make those changes to the DOM. It does that by swapping pointers of the Current Tree and Work in Progress Tree. And then **flushing (rendering)** those fibers to the DOM.
5. After swapping, the new Work in Progress Tree can be used for any new future state changes.



Swapped Current Tree and Work In Progress Tree Pointers.

**Tailwind and Props in React**

**Install Tailwind with Vite**

**Create Tailwind Project**

npm create vite@latest my-project -- --template react

cd my-project

**Install Taiwind**

npm install -D tailwindcss postcss autoprefixer

npx tailwindcss init -p