React Fiber (Home Task)

React Fiber is the **core engine** of React, introduced in React 16. Earlier, React's rendering system was simple, but it became slow for complex apps. Fiber made it **faster and more flexible**.

Purpose:

- To efficiently update the UI and handle update priorities.

Function:

- It divides React components into a tree structure and decides **which component should update first**.

Benefit:

- For heavy UI updates, Fiber allows React to **pause, resume, or abort updates**, keeping the app smooth.

Summary:

- Fiber is React's **smart scheduler** that optimizes rendering.

Virtual DOM vs Real DOM

Real DOM

- This is the **actual DOM in the browser** that displays the page.
- **Updates are slow** because any change can cause the entire page or element to re-render.
- Directly manipulates the UI.

Example:
If you want to change a single element, the entire parent element might re-render.

Virtual DOM

- This is React's **in-memory representation** of the Real DOM.
- Changes are first calculated in the **Virtual DOM**, then a **diffing algorithm** determines what actually needs to be updated.
- Only the **parts that changed** are updated in the Real DOM → fast and efficient.

Benefit:

- Better performance, smoother UI.

Summary:

- Real DOM = slow, direct updates
- Virtual DOM = fast, optimized updates using diffing

Real DOM vs Virtual DOM Example

HTML & JavaScript

```
 <h3>Hello</h3>
<h3>my friend</h3>
<h3 id="name">Ali Arsalan</h3>
<button>Click Me!</button>



<script>
let content = document.getElementById("name");
content.innerHTML = "<h3>Ali Ahmed</h3>";
</script>
```

Real DOM

- The **actual DOM** in the browser that displays the page.
- **Slow updates**: any change can cause the entire element or parent to re-render.
- Directly manipulates the UI.

Before JS:

```
 <h3>Hello</h3>
<h3>my friend</h3>
<h3 id="name">Ali Arsalan</h3>
<button>Click Me!</button>
```

After JS:

```
 <h3>Hello</h3>
<h3>my friend</h3>
<h3 id="name">Ali Ahmed</h3>
<button>Click Me!</button>
```

- Note: the entire node is replaced, inefficient.

Virtual DOM

- React's **in-memory representation** of the Real DOM.
- Changes are first calculated in the **Virtual DOM**, then a **diffing algorithm** determines what actually needs to be updated.
- Only the **changed part** is updated → fast and efficient.

Virtual DOM Before Update:

```
(type: content {h3: "Hello"; h3: "my friend"; h3: "Ali Arsalan"; button: "Click Me!"})
```

Virtual DOM After Update:

```
(type: content {h3: "Hello"; h3: "my friend"; h3: "Ali Ahmed"; button: "Click Me!"})
```

Benefit:

- Better performance, smoother UI, only the changed elements are updated.

Diffing Algorithm (Virtual DOM)

- **Purpose:** Efficiently update Real DOM by comparing old and new Virtual DOM.
- **How it works:**
    1. React creates a **new Virtual DOM** on state/props change.
    2. **Compare old vs new** Virtual DOM (diffing).
    3. Use **heuristics**:
        - Same element types are assumed similar.

        - Only compare same-level children.
        - Use **keys** for lists.

    4. **Update only changed parts** in Real DOM.

Example:

```
Old: <h3>Ali Arsalan</h3>
New: <h3>Ali Ahmed</h3>
Diff: Update only this node
```

React Fiber

- **Problem:** Large Virtual DOM trees can take long to compare (even ~200ms), causing UI freezes.
- **Solution:** React Fiber breaks the work into **small units**, allowing incremental rendering.

How it works:

1. Fiber **compares a small portion** of the Virtual DOM at a time.
2. If the tree is deep, it **pauses** rendering to let the browser handle new user updates.
3. **Resumes** later to continue updating remaining parts of the tree.

Benefits:

- Avoids freezing the app on big trees.
- Keeps UI **responsive** to user interactions.
- Stable enough for production; old apps don't crash.

React Fiber Root

```
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

- `createRoot()` → enables Fiber (concurrent rendering).
- `render()` → starts rendering component tree.
- Fiber **pauses/resumes** to keep UI responsive on big Virtual DOM trees.

React Components

Old Approach: Class Component

```
class Welcome extends React.Component {
    render() {
        return <h1>Hello, world!</h1>;
    }
}
```

- Uses `class` and `render()` method.
- More verbose, heavier for simple components.

New Approach: Function Component

```
const App = () => {
    return (
        <h1>Hello, world!</h1>
    );
}
```

- Uses **functions** and **hooks**.
- Lightweight, simpler, and preferred in modern React.

React Imports & Exports

Old Way

```
import React from 'react';  // mandatory in older React versions
```

New Way (React 17+)

- **Optional** to import React in function components.

Exporting Components

```
export default App;   // export the component
```

Importing Components

```
import App from './App';   // import from file location
```

- Works with both class and function components.
- Modern React prefers ES6 `import/export` style.

React Fragments

- In React, a component **cannot return multiple sibling elements** directly.
- Returning **two `<div>` elements side by side** causes an error.

Problem Example:

```
const App = () => {
    return (
        <div>
            <h1>Hello, world!</h1>
        </div>
        <div>
            <h1>Hello, world!</h1>
        </div>
    );
}
```

- Error: Multiple elements at the same level are not allowed in a single return.

Solution: Use Fragments

- Wrap multiple elements in a **fragment**:
  - `<React.Fragment>...</React.Fragment>`
  - Short syntax: `<>...</>`

Example:

```
const App = () => {
    return (
        <>
        <div>
            <h1>Hello, world!</h1>
        </div>
        <div>
            <h1>Hello, world!</h1>
        </div>
        </>
    );
}
```

- Fragments **don't add extra nodes** to the DOM.

React Relative Imports

Same Folder

```
./
|_ src
   |_ App.jsx
   |_ NavBar.jsx
```

In `App.jsx`:

```
import NavBar from './NavBar.jsx';
```

Sibling Folder

```
../
|_ src
   |_ App.jsx
|_ Images
   |_ Pic.jpg
```

In `App.jsx`:

```
import pic from '../Images/Pic.jpg';
```

Quick Notes

- `./` → current folder
- `../` → sibling folder
- Path after that → target file or folder