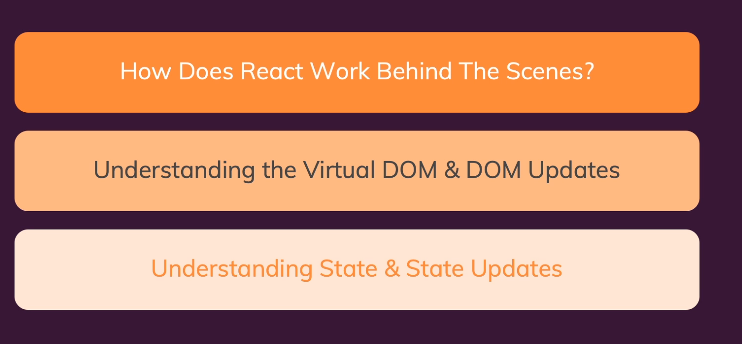
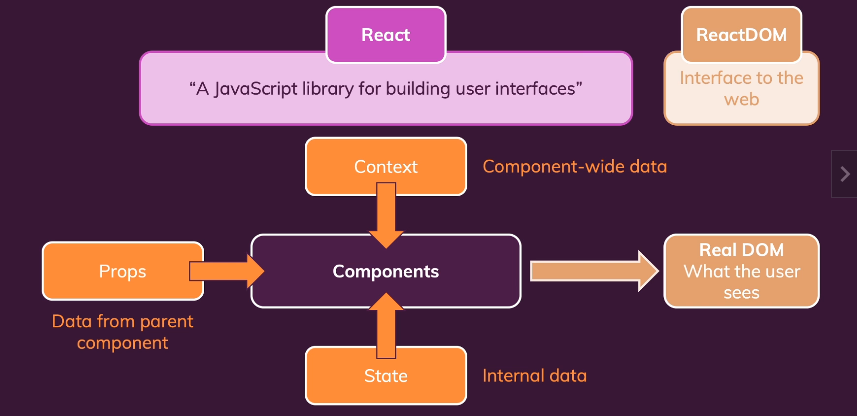
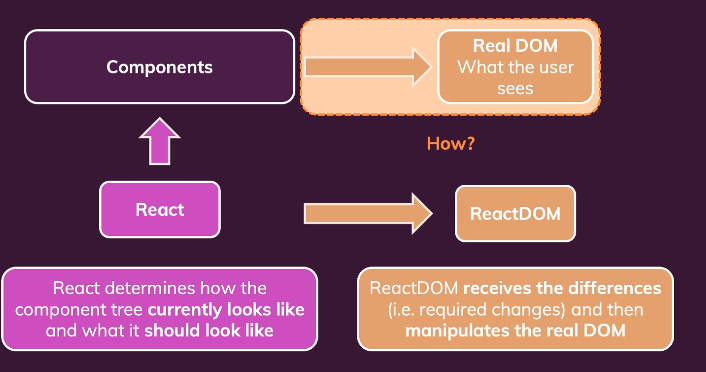
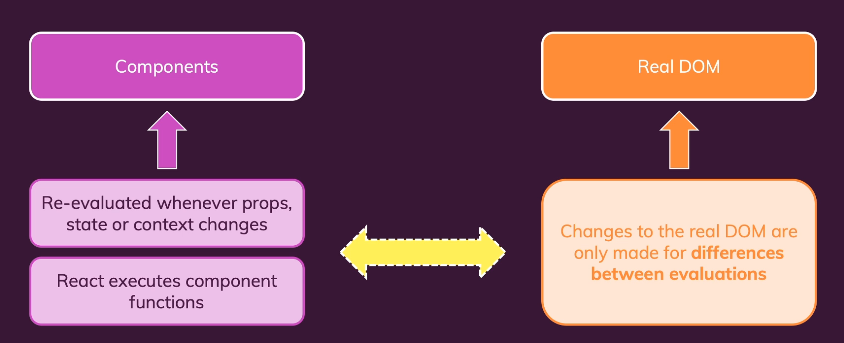
Sectioin 12: A Look behind the Scenes of react..

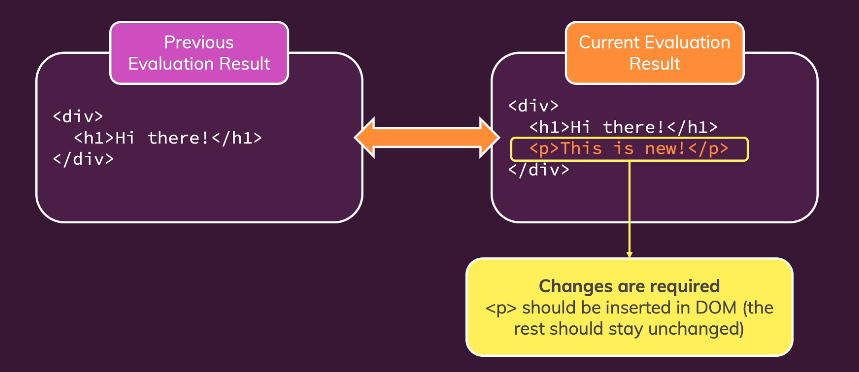


152. How React Relly Works









153. Component Updates In Action

Link source github: <https://github.com/academind/react-complete-guide-code/tree/12-a-look-behind-the-scenes/code>

Change app.js like this

import React, { useState } from 'react';

import Button from './components/UI/Button/Button';

import './App.css';

function App() {

  const [showParagraph, setShowParagraph] = useState(false);

  const toggleParagraphHandler = () => {

    setShowParagraph(prevShowParagraph => !prevShowParagraph);

  };

  return (

    <div className="app">

      <h1>Hi there!</h1>

      {showParagraph && <p>THis is new</p>}

      <Button onCLick={toggleParagraphHandler}>Toggle Paragraph!</Button>

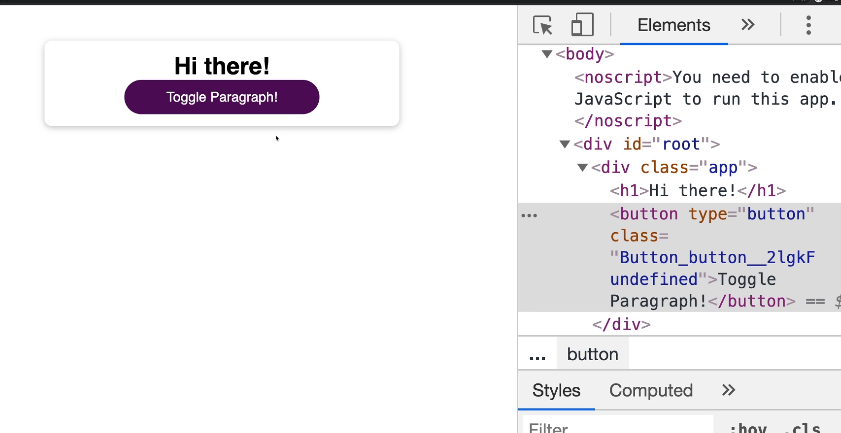
    </div>

  );

}

export default App;

note: only ‘{}’ affect to DOM and another is unchanged, and p always dispear when click button



154 A Closer Look At Child Component Re-Evaluation

Step 1 remove {showParagraph && <p>THis is new</p>} in app.js

Step 2 then create new folder components/Demo and DemoOutput.js

import React from 'react';

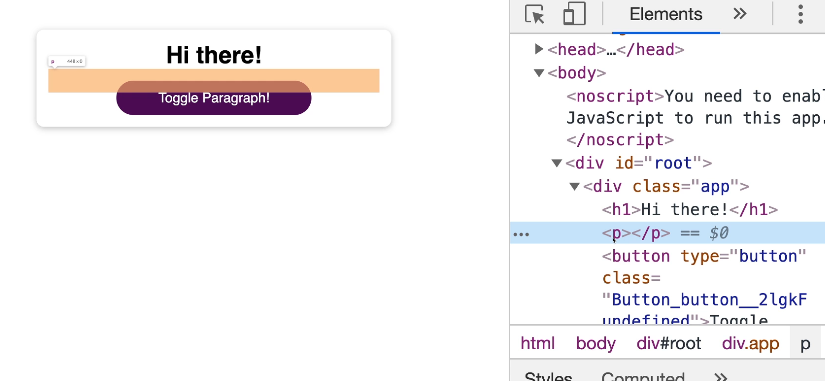
const DemoOutput = (props) => {

    return <p>{props.show ? 'This is new~' : ''}</p>

};

export default DemoOutput;

Step 3 note the p always there



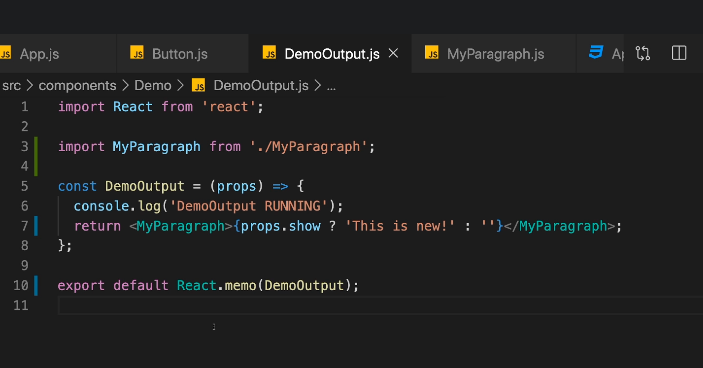
Note : When js execute all js in child will alse reload

155. Preventing Unnecessary Re-Evaluations with React.memo()

In Parent



Then in child

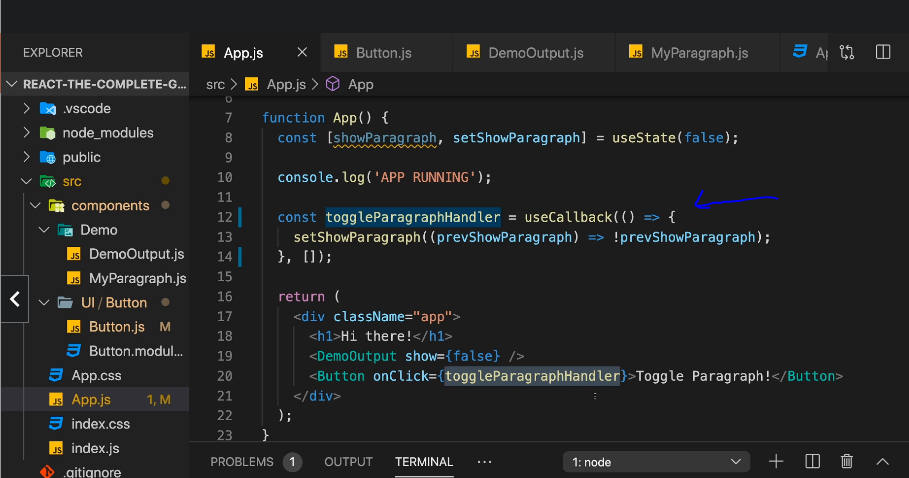


Note: Make component child no change (it will compare by props to check change or not) when component parent change, but will cost run heavy



156: Preventing Function Re-Creation with useCallback()

useCallBack make sure that function not execute (only when rerun)



157. useCallback() and its Dependencies

**useCallback** là được sử dụng để tối ưu quá trình **render** của **React functional components**. Nó sẽ rất hữu ích đối với trường hợp một thành phần (component) liên tục được hiển thị lại không cần thiết trong quá trình xử lý sự kiện người dùng và có hành vi chức năng phức tạp

Nào, bây giờ hãy xem xét ví dụ đơn giản qua **Counter** Component nhé:

import React, { useState, useCallback } from 'react'

function Counter() {

const [count, setCount] = useState(0);

const [countOther, setCountOther] = useState(0);

const increase = () => setCount(count + 1);

const decrease = () => setCount(count - 1);

const increaseOther = () => setCountOther(countOther + 1);

const decreaseOther = () => setCountOther(countOther + 1);

return (

<>

<div>Count: {count}</div>

<button onClick={increase}>+</button>

<button onClick={increase}>-</button>

<div>Count other: {countOther}</div>

<button onClick={increaseOther}>+</button>

<button onClick={decreaseOther}>-</button>

</>

)

}

export default Counter;

Điều này khá đơn giản, chúng ta có biến **2 state** nắm giữ số đếm và 4 hàm để thay đổi con số của 2 state trên. Tuy nhiên, vấn đề ở đây là mỗi lần thành phần **Counter** này re-render, tất cả 4 hàm, **increase**, **decrease**, **increaseOther**, **decreaseOther** sẽ bị khởi tạo lại.

Chúng ta có thể thấy điều đó bằng cách sử dụng **Set** và thêm các hàm vào bên trong **Set** mỗi lần **Counter** re-render. Tại sao lại là **Set**. Đây là đối tượng lưu trữ phần tử có tính duy nhất, không trùng lặp.

import React, { useState, useCallback } from 'react'

const storeSet = new Set();

function Counter() {

const [count, setCount] = useState(0);

const [countOther, setCountOther] = useState(0);

const increase = () => setCount(count + 1);

const decrease = () => setCount(count - 1);

const increaseOther = () => setCountOther(countOther + 1);

const decreaseOther = () => setCountOther(countOther + 1);

storeSet.add(increase);

storeSet.add(decrease);

storeSet.add(increaseOther);

storeSet.add(decreaseOther);

console.log(storeSet);

return (

<>

<div>Count: {count}</div>

<button onClick={increase}>+</button>

<button onClick={increase}>-</button>

<div>Count other: {countOther}</div>

<button onClick={increaseOther}>+</button>

<button onClick={decreaseOther}>-</button>

</>

)

}

export default Counter;

Nào, bây giờ cùng kiểm tra thử nhé?. Mỗi lần bạn **click** vào bất kì nút tăng giảm sẽ thấy hiển thị giá trị của **storeSet**. Bạn sẽ thấy chúng tăng giá trị mỗi lần hiển thị, điều này cho thấy mỗi lần thành phần **re-render** sẽ tạo ra phiên bản hoàn toàn mới của các hạm được tạo ra.

Về cốt lõi, vấn đề này là do cách **JavaScript** xác định bình đẳng hàm

Solved

Hãy cùng cập nhật cho các hàm increase/decrease sử dụng **useCallback**

const increase = useCallback(() => setCount(count + 1), [count]);

const decrease = useCallback(() => setCount(count - 1), [count]);

const increaseOther = useCallback(() => setCountOther(countOther + 1), [countOther]);

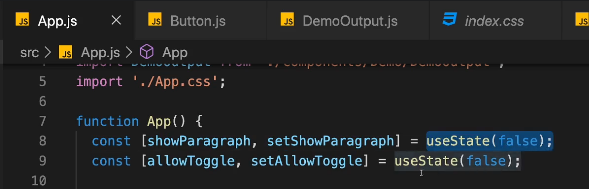
const decreaseOther = useCallback(() => setCountOther(countOther + 1), [countOther]);

Trong trường hợp này, khi chúng ta thử ấn button **increase** phía trên sau khi bọc các hàm setState lại bằng **useCallback** , kiểm tra hiển thị giá trị của **storeSet** chúng ta chỉ thấy duy nhất có một giá trị được tăng thêm, tiếp tục ấn button **increase** chúng ta vẫn không thấy giá trị thay đổi.

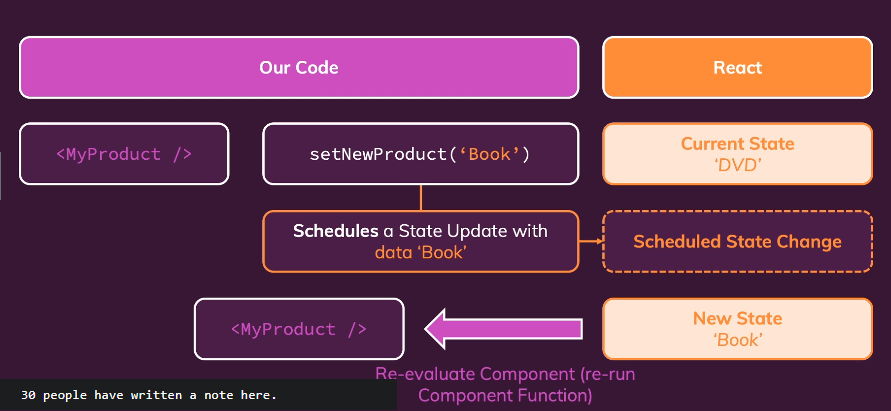
Mặc dù đây là một ví dụ rất đơn giản, nhưng chúng ta có thể thấy cách sử dụng **useCallback** để tối ưu hóa các thành phần có các chức năng phức tạp hoặc tốn nhiều tài nguyên.

159. A Closer Look At State & Component

Note: state never change, because state is managed by react



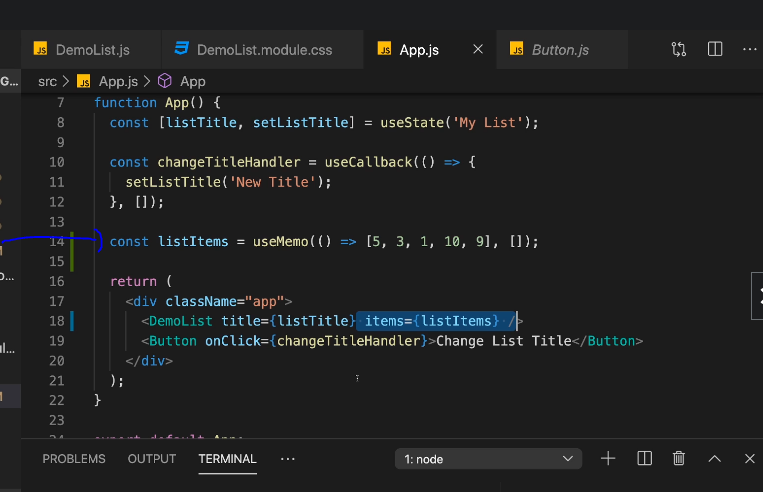
160. Understanding State Scheduling & Batching



161. Optimizing with useMemo()

Only run when props change

In parent



Then in Child

