


# Lecture 01 (Part-2): React CDN, React vs ReactDOM, and React 18 Rendering

---

## React को JS File में कैसे लाएँ?

React को JavaScript फ़ाइल में लाने के लिए हमें React और ReactDOM की **CDN Links** की ज़रूरत होती है।

👉 CDN वेबसाइट पर जाएँ:  
 <https://unpkg.com>

या फिर नीचे दिए गए `<script>` टैग्स को HTML में `<body>` से पहले paste करें:

```
<!--  React Core Library -->
```

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

```
<!--  React DOM Library -->
```

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

✅ इससे आप बिना Node.js और npm install किए React को browser में use कर सकते हैं।

---

## React और ReactDOM क्या हैं?

```
typeof React      //  "object"
```

```
typeof ReactDOM   //  "object"
```

दोनों JavaScript के objects होते हैं जो अलग-अलग काम करते हैं।

---

## React vs ReactDOM: क्या फर्क है?

## ◆ React

React एक **core UI library** है।

Platform-independent है (Mobile, Web, VR आदि)।

Component logic, state, props manage करता है।

Example: `React.createElement()`

## ◆ ReactDOM

ReactDOM **browser के लिए rendering library** है।

सिर्फ **browser-based DOM** को handle करता है।

DOM में render करने के लिए methods देता है।

Example: `ReactDOM.createRoot()`

### 🔧 Separation का फ़ायदा:

- React का **core छोटा और modular** बना रहता है।
- अलग-अलग platforms के लिए अलग rendering logic बनाया जा सकता है (जैसे React Native)।

---

## 📱 React Native vs ReactDOM

### React Native

Mobile Apps (iOS/Android) के लिए

Native UI components (Button, TextInput आदि)

Mobile-specific rendering logic

Uses `react-native` package

### React DOM

Web/Browser के लिए

HTML DOM elements (div, h1, p आदि)

Browser-specific rendering logic

Uses `react-dom` package

💡 React Native और React DOM दोनों **React core** को use करते हैं — बस rendering mechanism अलग होता है।

---

## ❌ ReactDOM.render() in React 18 — Deprecated

```
// ❌ This will give a warning in React 18+  
  
const element = React.createElement('h1', {}, "Hello React");  
  
ReactDOM.render(element, document.getElementById('root'));
```

⚠️ **React 18** में `ReactDOM.render()` को deprecated कर दिया गया है।

क्यों?

क्योंकि ये method **synchronous** और **blocking** थी — यानी:

- Rendering को बीच में रोका नहीं जा सकता था
- Browser का DOM एक ही thread में काम करता है — जिससे performance degrade होती थी

---

## ✅ React 18 में सही तरीका: createRoot()

```
// ✅ Correct Way in React 18+  
  
const element = React.createElement('h1', {}, "Hello React");  
  
  
// Step 1: Create root container  
  
const root = ReactDOM.createRoot(document.getElementById('root'));  
  
  
// Step 2: Render element inside root  
  
root.render(element);
```

---

## 💣 Netflix Example से समझो (Real-World Problem)

🖥️ पहले क्या होता था?

- Netflix पर **TV Shows** बटन दबाया → सारे shows एक साथ load होने लगे।
- जब तक पूरा data render नहीं होता था, UI freeze हो जाता था।
- अगर आप उस दौरान **Movies** पर click करते, तो कोई असर नहीं होता था।

❌ पुराना rendering synchronous था — UI को **response देने का मौका नहीं मिलता था**।

---

## 🔧 Solution: React 18 का Concurrent Rendering

```
const root = ReactDOM.createRoot(document.getElementById('root'));
```

👉 अब React root को एक **container object** में बदल देता है जिससे React:

- ✅ Rendering को बीच में रोक सकता है
- ✅ किसी नये user action के आने पर पुरानी rendering cancel कर सकता है
- ✅ Lazy loading या progressive rendering कर सकता है

### 🎯 Example:

2017 Netflix: पहले 50 cards render होते, फिर बाकी  
अब: आप बीच में **Movies** क्लिक करें तो TV Shows की rendering रुक जाती है

---

## 🧠 React.root के फायदे

Feature	Benefit
✅ Control over rendering	कोई नया action आये तो पुराना rendering pause/cancel
✅ Better performance	Load-balancing के साथ rendering
✅ Modular rendering	Multiple UI parts अलग-अलग load कर सकते हैं

---

## 📝 Notes & Key Points

- `React.createElement()` → Virtual DOM element बनाता है
  - `ReactDOM.createRoot()` → React 18 का नया rendering entry point है
  - `root.render()` → Final DOM में inject करता है
  - React Core: Platform-independent
  - ReactDOM: Browser-specific rendering
  - React Native: Mobile-specific rendering
- 

## Summary: Fast Revision

### Concept

### Detail

React CDN

Use `unpkg.com` to import React via script

React vs ReactDOM

React handles logic, ReactDOM handles rendering

React Native vs DOM

React Native = mobile, DOM = web

Deprecated Method

`ReactDOM.render()` ❌

New Method

`ReactDOM.createRoot()` + `root.render()` ✅

Advantage

Concurrent UI updates, faster, interruptible rendering

Real Use Case

Netflix - now better UI response due to new rendering

---

## Pro Tip:

🧠 अगर आप React को अंदर से master करना चाहते हैं, तो ReactDOM और rendering के इस concept को 100% समझना ज़रूरी है।

React 18 का **concurrent rendering** feature future का UI है!

## What is CDN? (Content Delivery Network)

**CDN** एक globally distributed network है जो वेबसाइट्स का static content users तक उनके **nearest server location** से पहुंचाता है — ताकि डेटा जल्दी लोड हो और वेबसाइट तेज़ चले।

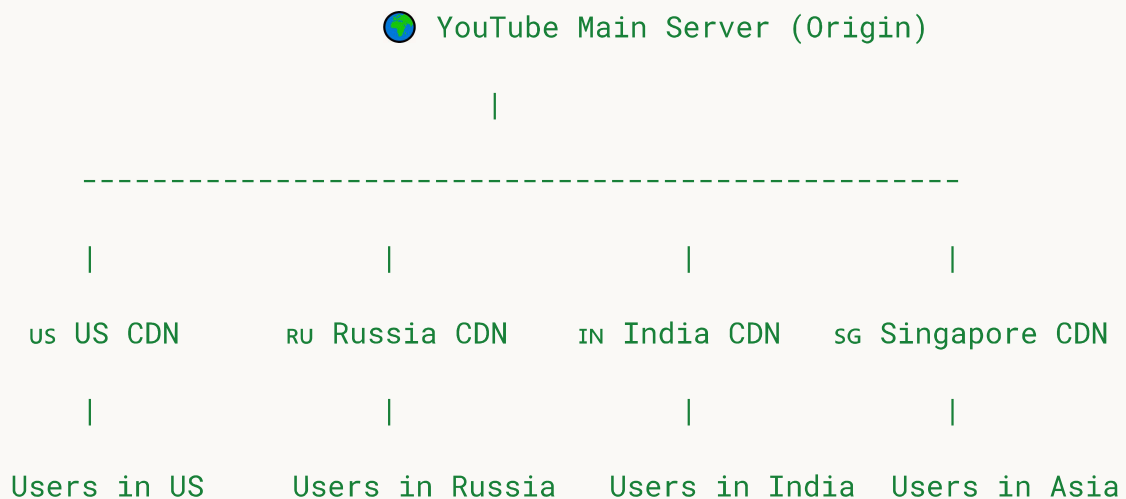
📌 Example:

- आपने YouTube पर वीडियो चलाया
- वो वीडियो America के server पर है
- लेकिन आप India में हो → तो CDN की वजह से वही वीडियो **India के नज़दीकी server** से stream होता है
- ⚡ Super fast performance

---

## Where is YouTube's Data Stored?

💡 Real Architecture (Simplified):



✅ All these are **CDN servers (also called POPs – Point of Presence)**

✅ हर लोकेशन पर data store होता है (cached)

📌 जब आप किसी content को पहली बार access करते हो:

- वो main server (origin) से आता है
- फिर local CDN पर cache हो जाता है
- अगली बार वहीं से serve होता है

---

## "SPRUPR" = CDN Server (Symbolic Name)


आपके content में SPRUPR एक symbolic नाम की तरह इस्तेमाल हुआ है — इसे हम **CDN Node / Edge Server** कह सकते हैं।

Term	Meaning
SPRUPR	Representing a CDN server in this example
Origin Server	Original server जहाँ actual data है
Edge Server	Local CDN server जो users को तेज़ी से data देता है




---

## Why is CDN Important?

### Without CDN:

- YouTube data India में भी US से आएगा
-  Slow, laggy, buffering
- Server load बहुत ज्यादा हो जाएगा


### With CDN:

- Video once loaded → stored locally
-  Fast access next time
-  Reduces server load
-  Global reach, faster performance

---

## What is a Server? (Correct Definition)

**Server** एक high-performance computer होता है जो internet पर चलने वाली apps, websites, और services का **data, logic, और files** host करता है।

 Server कोई magic machine नहीं है — ये powerful computers होते हैं जो हमेशा on रहते हैं।


---


### Types of Servers:





Type	Purpose
Web Server	Websites run करना
App Server	Backend logic handle करना
CDN Server	Static content fast serve करना
Database Server	Data store करना (MySQL, MongoDB etc.)

---

## Static Data vs Dynamic Data in CDN

 **Static Data** rarely changes, so it's cached in CDN for fast delivery.

 **Dynamic Data** changes frequently and comes directly from the main server (Origin).

Type	Example	Stored in CDN?	Served From
 Static Data	Videos, Images, HTML, CSS	 Yes	CDN Server
 Dynamic Data	Likes, Comments, Messages	 No	Main Server (Origin)

---

### How it Works:



### ✅ First Time:

- Video request → Origin Server से आएगा
- फिर वो data **India के CDN** में save हो जाएगा

### ✅ Next Time:

- Same video → Direct **local CDN** से मिलेगा (super fast)

### 🔄 Dynamic Update:

- किसी ने comment किया?
  - वो CDN में नहीं जाएगा
  - वो request सीधे **main server** तक जाएगी

---

## ⚠️ If a CDN Server Fails?

- Suppose India का CDN server (SPRUPR) down हो गया
- 📍
  - Then nearest CDN (जैसे Singapore) से data serve होगा
  - ✅ **No downtime** for user

---

## 🧠 Key Terms Simplified

Term	Meaning
CDN	Content delivery network, fast content serving
Edge Server	Nearest server to user
Origin Server	Central server where main data lives

Static Data      Non-changing content (videos, images, files)

Dynamic Data      User-generated or real-time (likes, comments)

Server      Computer that hosts applications/data online

---



## Summary :

### 📌 React को CDN से JS में Import करें

React और ReactDOM को `<script>` के ज़रिए browser में लाया जा सकता है — बिना Node.js या npm के।

### 📌 React vs ReactDOM

- **React:** Component logic, state, UI handle करता है (Platform-independent)
- **ReactDOM:** Web के लिए rendering का काम करता है

### 📌 React Native vs ReactDOM

- **React Native:** Mobile rendering (Button, TextInput)
- **ReactDOM:** Browser rendering (div, h1)  
दोनों React Core को use करते हैं।

### 📌 React 18 Update

- ❌ `ReactDOM.render()` अब पुराना हो चुका है
- ✅ `ReactDOM.createRoot()` + `root.render()` है नया तरीका  
इससे UI responsive बनता है, rendering interrupt हो सकती है

### 📌 Netflix Example

पहले पूरे shows एकसाथ render होते थे → अब React root allows canceling & interrupting rendering mid-way

### 📌 React Root के फायदे

- UI performance बेहतर

- Lazy loading & concurrent updates
- बेहतर user experience

### 📌 CDN (Content Delivery Network)

Static data (HTML, CSS, videos) आपके नज़दीकी server (CDN node) से serve होता है  
Dynamic data (likes, comments) original/main server से आता है

### 📌 Server Meaning (Corrected)

Server = एक high-performance computer जो web apps और sites run करता है (कोई magic machine नहीं 😊)

### 📌 Static vs Dynamic Data in CDN

Type	Stored in CDN?	Example
Static Data	✅ Yes	Videos, Images, CSS
Dynamic Data	❌ No	Likes, Comments, Login