This is the "aha!" moment for the App Router. The reason it's confusing is that for the last 10 years, React has taught us that **everything** must hydrate. The App Router changes the rules.

To understand this, you have to separate **"HTML Generation"** from **"Hydration."**

---

1. The Core Rule

- **Server Components (RSC):** Render to HTML on the server. They **never** send JavaScript to the browser. Therefore, they **never hydrate**.
- **Client Components ('use client'):** Render to HTML on the server (for the initial view) AND send JavaScript to the browser. They **do hydrate**.

Is this specific to the App Router?

Yes. In the Pages Router, every single component in your file was effectively a "Client Component." Even if it was just a static <h1>, React would still "hydrate" it. The App Router is the first time we can have "selective hydration."

---

2. When does Hydration happen? (The "Use" Rule)

In the App Router, you should visualize your page as a tree. The "roots" are Server Components, and the "leaves" are Client Components.

NO Hydration happens when:

A component is a **Server Component**. These are used for 80% of your app's structure:

- Fetching data from a database.
- Displaying static text, images, and links (<Link> in Next.js handles navigation without full hydration).

- Handling sensitive information (API keys).
- Large dependencies (e.g., a Markdown parser). The library stays on the server; only the resulting HTML goes to the user.

Hydration MUST happen when:

A component uses **Browser APIs** or **React Hooks**. You must mark these with 'use client':

- **Interactivity:** onClick, onChange, onSubmit.
- **State:** useState(), useReducer().
- **Lifecycle:** useEffect(), useLayoutEffect().
- **Browser APIs:** window, localStorage, document, navigator.

---

3. A Practical Example

Imagine a **Product Page**.

1. **ProductDescription (Server Component):** It fetches the description from a database. It's just a bunch of <p> and tags.
    - **Result:** Next.js turns this into HTML. **Zero JS** is sent for this. It **never hydrates**.
2. **AddToCartButton (Client Component):** It needs an onClick to add the item to a cart and useState to show a "Success!" message.
    - **Result:** Next.js turns it into HTML (for the initial look) AND sends the JS for this button. This component **hydrates**.

---

4. Why is "No Hydration" a good thing? (The "JS Tax")

In a traditional React app (Pages Router or Vite), you pay a **"JavaScript Tax"** for every component:

1. **Download Tax:** User waits to download the code for the static description.
2. **Parse Tax:** The browser's CPU has to read and understand that code.
3. **Hydration Tax:** React has to walk through the DOM and "attach" itself to the description, even though it's never going to change.

In the App Router, the "Tax" is only paid by the interactive parts.

---

5. Interview Q&A: "No Hydration" Specifics

Q: If a Server Component doesn't hydrate, how do links work?

A: Next.js uses a specialized Link component. It uses a small, global "router" on the client that intercepts the click. It doesn't need the entire page to be hydrated to perform a fast, client-side transition.

Q: Can a Server Component be a child of a Client Component?

A: No, not directly. If you import a Server Component into a 'use client' file, it becomes a Client Component and will hydrate.
The Workaround: You must pass the Server Component as children to the Client Component. This keeps the Server Component "static" while the parent is "interactive."

Q: How does the browser know what to do if there's no JS for the Server Components?

A: The browser treats it like old-school HTML. It just renders it. React only manages the "islands" of interactivity (the Client Components) that live within that HTML.

Q: What is the "RSC Payload" if there's no hydration?

A: Even though there's no hydration for Server Components, Next.js sends an **RSC Payload**. This is a lightweight map of the UI. If you navigate to another page, React uses this payload to "swap" the content without a full page reload, but it still doesn't "hydrate" the static parts.