In a Frontend System Design interview, **React Server Components (RSC)** is the "Endgame" of rendering patterns. It isn't just an evolution of SSR or SSG; it is a fundamental rewrite of how React interacts with the server.

To understand RSC, you have to understand the **"Waterfall Problem"** and the **"JavaScript Tax"** that plagued CSR, SSR, and SSG.

---

1. Why was RSC needed? (The Problem)

Even with SSR and SSG, we had two massive bottlenecks:

1. **The JavaScript Tax:** In SSR/SSG, even if a component is 100% static (like a footer), the server still sends the JavaScript for that component to the browser so React can "hydrate" it. As your app grows, your JS bundle grows, making the site slower for users on mobile devices.
2. **The Network Waterfall:** In CSR, a parent component renders, then fetches data, then renders a child, which then fetches more data. This creates a "waterfall" of slow network requests.
3. **Heavy Dependencies:** If you wanted to use a heavy library (like a Markdown parser or a Date formatting library) in a component, that whole library had to be downloaded by the user's browser.

RSC solves this by allowing components to stay on the server forever.

---

2. What is a React Server Component?

An RSC is a component that **only runs on the server.** * **It never hydrates:** It produces zero JavaScript for the client.

- **Direct Access:** It can talk directly to your database or file system because it's running on the server.
- **The Payload:** Instead of sending HTML or a JS bundle, the server sends a specialized **RSC Payload** (a serialized tree) that React on the client uses to "stitch" the UI together.

---

3. RSC vs. SSR (The "Big Confusion")

This is the most common interview question. **RSC is not SSR.**

Feature	Server-Side Rendering (SSR)	React Server Components (RSC)
Hydration	**Yes.** The whole page hydrates.	**No.** RSCs never hydrate.
JS Bundle	Includes code for all components.	**Excludes** code for Server Components.
State	Can use useState (after hydration).	**Cannot** use hooks or state.
Environment	Runs on Server, then Client.	**Runs ONLY on the Server.**
Result	Sends HTML string.	Sends a serialized UI "blueprint."

---

4. How it works: The Client-Server Boundary

In an RSC-based app (like Next.js App Router), you decide where the boundary is.

1. **Server Components (Default):** Great for data fetching, large static text, and heavy logic.
2. **Client Components ('use client'):** Great for interactivity (buttons, forms, state).

Because Server Components don't hydrate, you can have a 1MB Markdown library on the server, use it to render a blog post, and the user receives **0 bytes** of that library. They only get the final, formatted text.

---

5. The "Waterfall" Solution

In CSR, components and data are fetched like this:

- Browser fetches App.js -> App renders -> App fetches User.js -> User fetches Data.json.

In RSC, the "Waterfall" happens on the **Server**:

- The Server Component fetches the data directly from the database (latency is near zero) and sends the final UI structure to the client in one go.

---

6. Interview Q&A

Q: Can you call a Client Component from a Server Component?
A: Yes. This is the standard pattern. The Server Component renders the data and passes it as "props" to the Client Component for interactivity.

Q: Can you call a Server Component from a Client Component?
A: No. You cannot import a Server Component into a Client Component. However, you can pass a Server Component as children to a Client Component. This allows the Server Component to be rendered on the server while the Client Component manages the interactive shell around it.

Q: Why do Server Components improve "Time to Interactive" (TTI)?
A: Because there is less JavaScript to parse and execute. Since the static parts of the page (RSCs) don't need to be hydrated, the browser's main thread is free to handle user inputs immediately.

Q: Does RSC replace SSG?
A: It complements it. You can have a "Static Server Component." Next.js will run the component once at build time, save the result, and serve it like SSG—but with the added benefit of zero-hydration on the client.

Final Summary Table

Pattern	Where it Renders	Hydration?	Best For...
CSR	Browser	Full	Internal tools, heavy interactivity.
SSR	Server -> Browser	Full	SEO + dynamic/personalized data.
SSG	Build Time	Full	SEO + static content (Blogs).
ISR	Background	Full	Large scale sites with static content.
RSC	**Server Only**	**None**	**Everything else (Data, Layouts, Logic).**