In the **Next.js Pages Router**, Static Site Generation (SSG) is the gold standard for performance. It allows you to move the "work" of fetching data and rendering components from the user's request time to the developer's **build time**.

---

1. How SSG Works in Pages Router

When you run next build, Next.js identifies pages that use SSG functions. It calls your APIs, fetches the data, and generates a physical .html file and a matching .json file for every page.

The Two Core Functions

1. **getStaticProps**: This function runs on the server at build time. It fetches data and passes it to the component.
2. **getStaticPaths**: Used for **dynamic routes** (e.g., /post/[id]). Since Next.js needs to know *which* HTML files to generate at build time, this function returns a list of all possible IDs.

Code Implementation (Pages Router)

```
// pages/blog/[slug].js

// 1. Tell Next.js which dynamic pages to generate
export async function getStaticPaths() {
  const posts = await fetch('https://api.example.com/posts').then(res => res.json());

  const paths = posts.map((post) => ({
    params: { slug: post.slug },
  }));

  return { paths, fallback: false }; // fallback: false shows 404 if slug doesn't exist
}

// 2. Fetch data for each specific page at build time
export async function getStaticProps({ params }) {
  const res = await fetch(`https://api.example.com/posts/${params.slug}`);
  const post = await res.json();

  return {
    props: { post }, // Passed to the component
  };
}
```

```
export default function BlogPost({ post }) {
  return <h1>{post.title}</h1>;
}
```

---

2. Hydration in SSG

Hydration is the process of turning the "static" HTML into an "interactive" React application. In SSG, this is extremely efficient because the data is already embedded in the page.

The Step-by-Step Flow:

1. **The Delivery:** The server (or CDN) sends the pre-generated HTML file to the browser.
2. **The Fast Paint:** The browser renders the HTML immediately. The user sees the text and images instantly.
3. **The JSON Payload:** Next.js includes a hidden script tag: <script id="__NEXT_DATA__">. This contains the **JSON props** fetched during getStaticProps.
4. **The JS Download:** The browser downloads the React JavaScript bundle.
5. **The Reconciliation (Hydration):** React "wakes up." It reads the JSON from __NEXT_DATA__ to build its internal state. It then compares its virtual DOM with the existing HTML. Since they match perfectly (because they were both built from the same data), React simply attaches event listeners.

---

3. Why SSG + Hydration is Powerful

In a System Design interview, emphasize these three points:

- **Zero Database Latency:** Because the database was queried at build time, the user doesn't wait for a slow SQL query. The response time is purely the speed of the CDN.
- **Predictable Performance:** Every user gets the exact same file. There is no variance in speed based on server load.
- **Resilience:** If your database goes down, your site stays up. The HTML files are already built and sitting on a storage bucket/CDN.

---

4. Interview Q&A: SSG (Pages Router)

Q: What is the difference between SSG and SSR in the Pages Router?

A: The difference is **Timing**. SSG happens **once** at build time. SSR happens **every time** a user requests the page. Use SSG if the data is the same for all users; use SSR if the data is personalized or changes constantly.

Q: What happens if I have 1 million products? Is SSG still viable?

A: Pure SSG is not viable because the build would take days. Instead, use **Incremental Static Regeneration (ISR)** or set fallback: 'blocking' in getStaticPaths. This allows you to build the top 1,000 products at build time and generate the rest on-demand as users visit them.

Q: Why does Next.js send a JSON file along with the HTML?

A: When you navigate to an SSG page via a client-side link (using next/link), Next.js doesn't request the full HTML. It only requests that small .json file and uses it to render the page on the client. This makes navigation feel instantaneous.

Q: Can I use window or localStorage in an SSG page?

A: Only inside a useEffect or after a check for process.browser. If you use them in the main body of the component, the build will fail (or throw a hydration error) because the server doesn't have a window object when it's generating the HTML.
