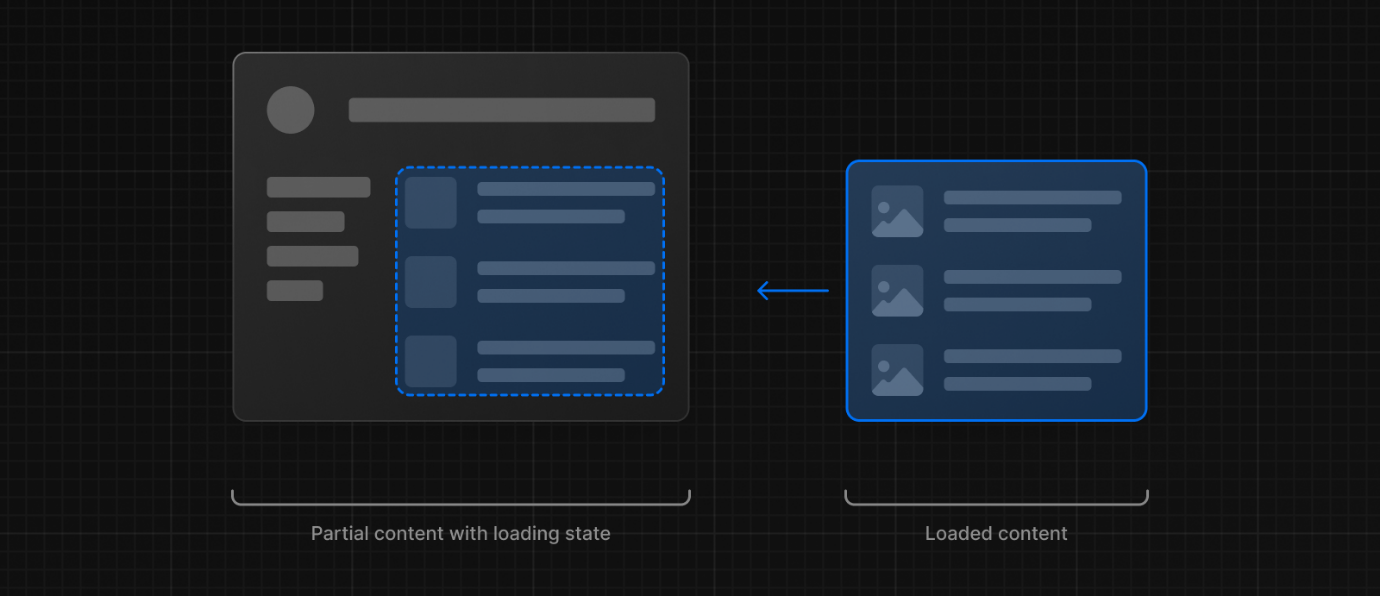
**Loading UI and Streaming**

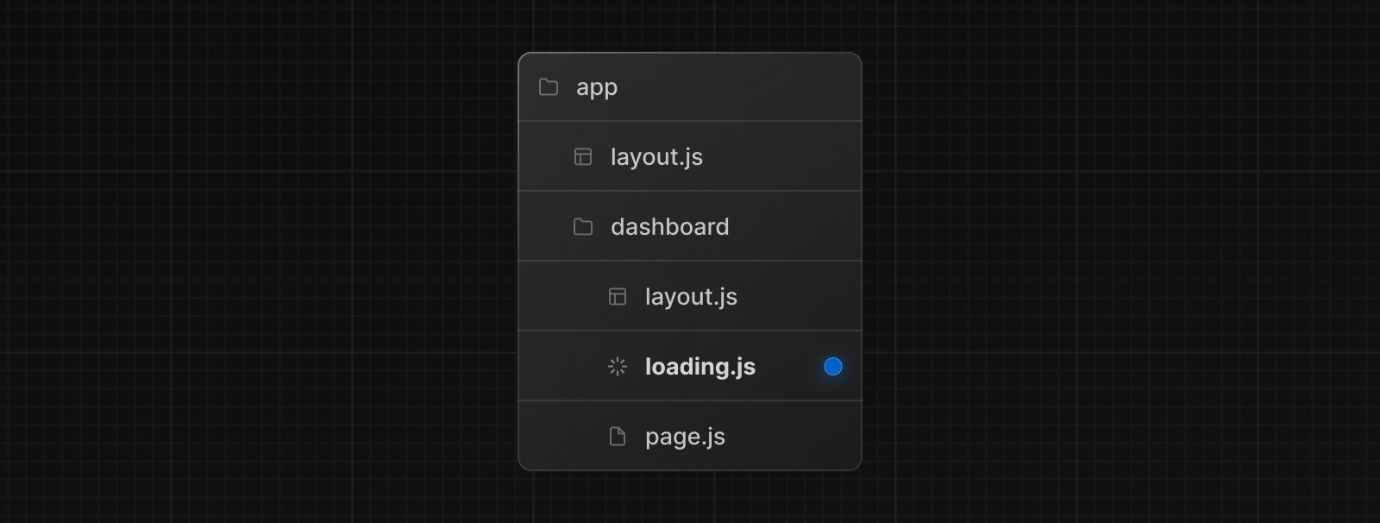
The special file loading.js helps you create meaningful Loading UI with React Suspense. With this convention, you can show an instant loading state from the server while the content of a route segment loads. The new content is automatically swapped in once rendering is complete.



**Instant Loading States**

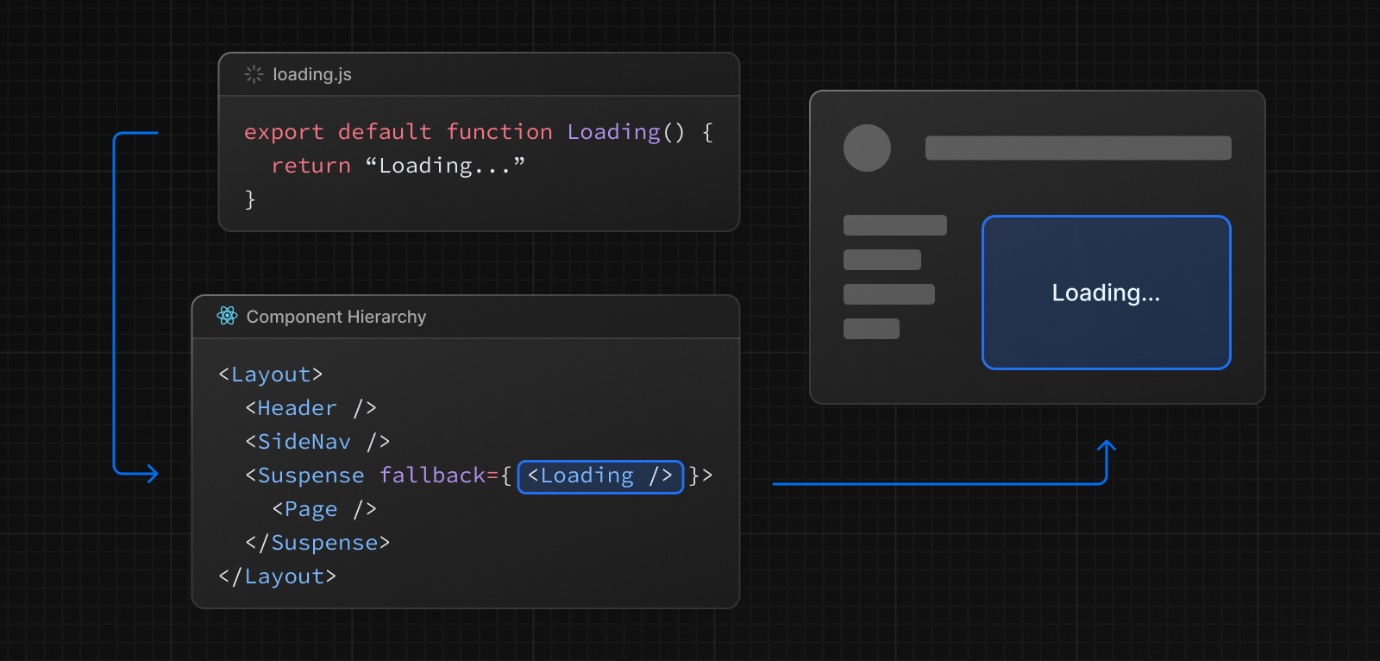
An instant loading state is a fallback UI that is shown immediately upon navigation. You can pre-render loading indicators such as skeletons and spinners, or a small but meaningful part of future screens such as a cover photo, title, etc. This helps users understand the app is responding and provides a better user experience.

Create a loading state by adding a loading.js file inside a folder.



export default function Loading() { // You can add any UI inside Loading, including a Skeleton. return <LoadingSkeleton />}

In the same folder, loading.js will be nested inside layout.js. It will automatically wrap the page.js file and any children below in a <Suspense> boundary.



**Good to know**:

**Navigation** is immediate, even with [server-centric routing](https://nextjs.org/docs/app/building-your-application/routing#server-centric-routing-with-client-side-navigation).

**Navigation** is interruptible, meaning changing routes does not need to wait for the content of the route to fully load before navigating to another route.

Shared layouts remain interactive while new route segments load.

**Recommendation:** Use the loading.js convention for route segments (layouts and pages) as Next.js optimizes this functionality.

[**Streaming with Suspense**](https://nextjs.org/docs/app/building-your-application/routing/loading-ui-and-streaming#streaming-with-suspense)

In addition to loading.js, you can also manually create Suspense Boundaries for your own UI components. The App Router supports streaming with [Suspense](https://react.dev/reference/react/Suspense) for both [Node.js and Edge runtimes](https://nextjs.org/docs/app/building-your-application/rendering/edge-and-nodejs-runtimes).

[**What is Streaming?**](https://nextjs.org/docs/app/building-your-application/routing/loading-ui-and-streaming#what-is-streaming)

To learn how Streaming works in React and Next.js, it's helpful to understand **Server-Side Rendering (SSR)** and its limitations.

With SSR, there's a series of steps that need to be completed before a user can see and interact with a page:

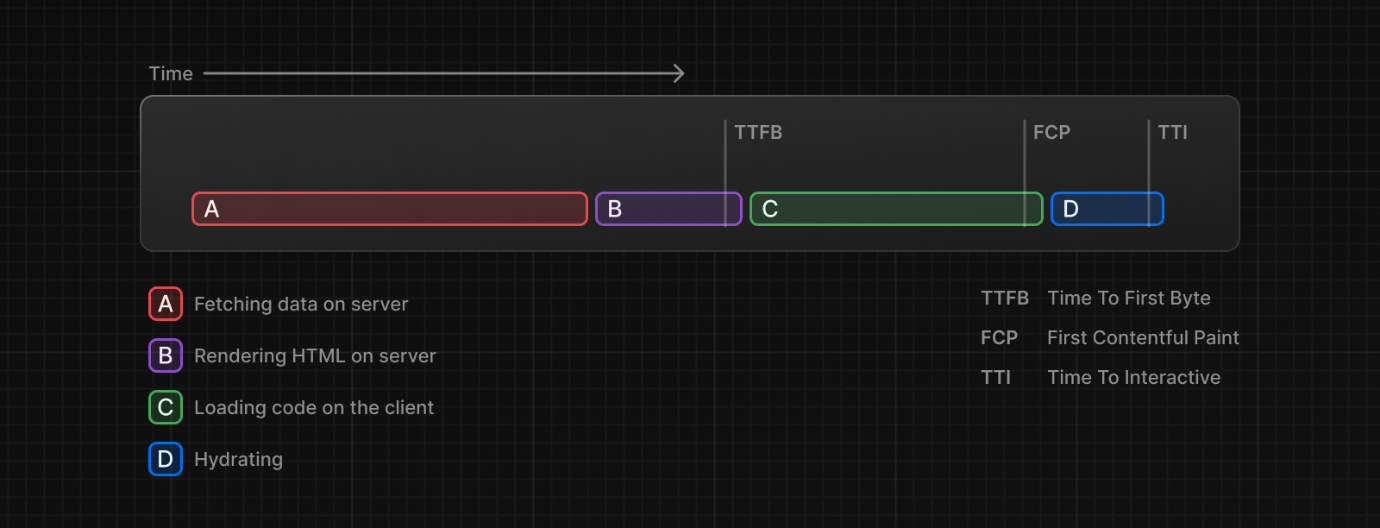
First, all data for a given page is fetched on the server.

The server then renders the HTML for the page.

The HTML, CSS, and JavaScript for the page are sent to the client.

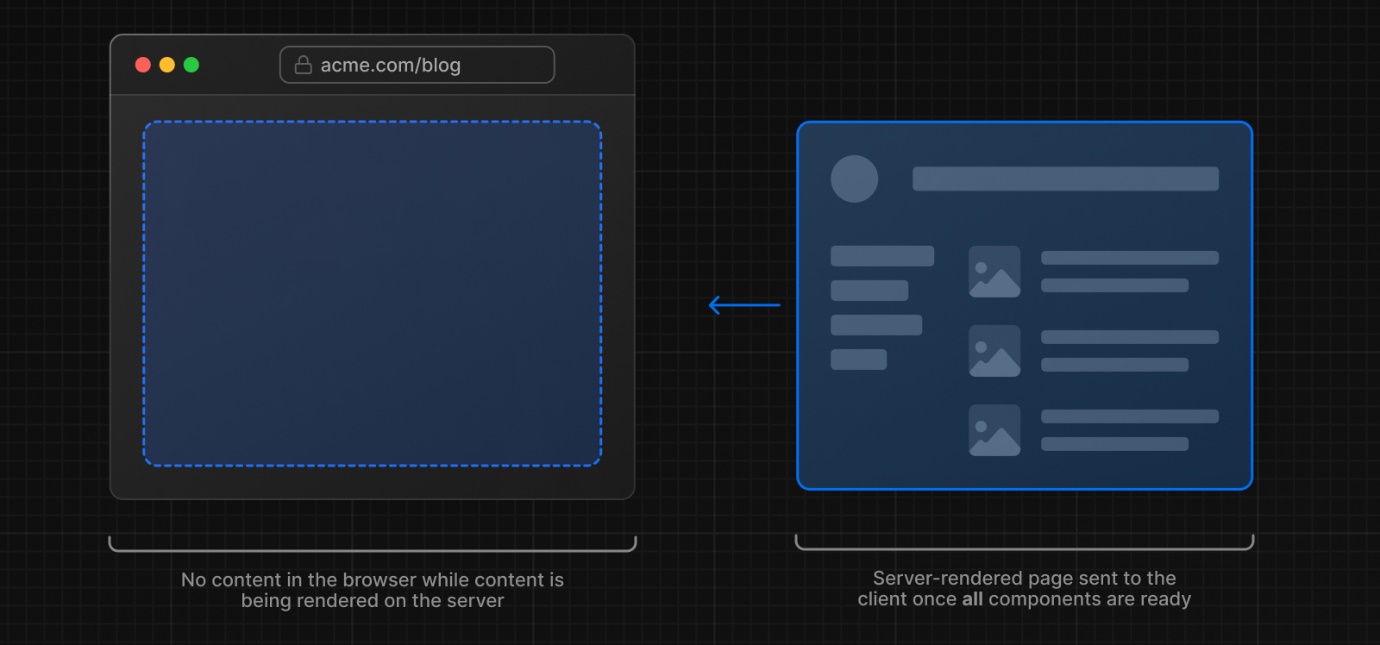
A non-interactive user interface is shown using the generated HTML, and CSS.

Finally, React [hydrates](https://react.dev/reference/react-dom/client/hydrateRoot#hydrating-server-rendered-html) the user interface to make it interactive.



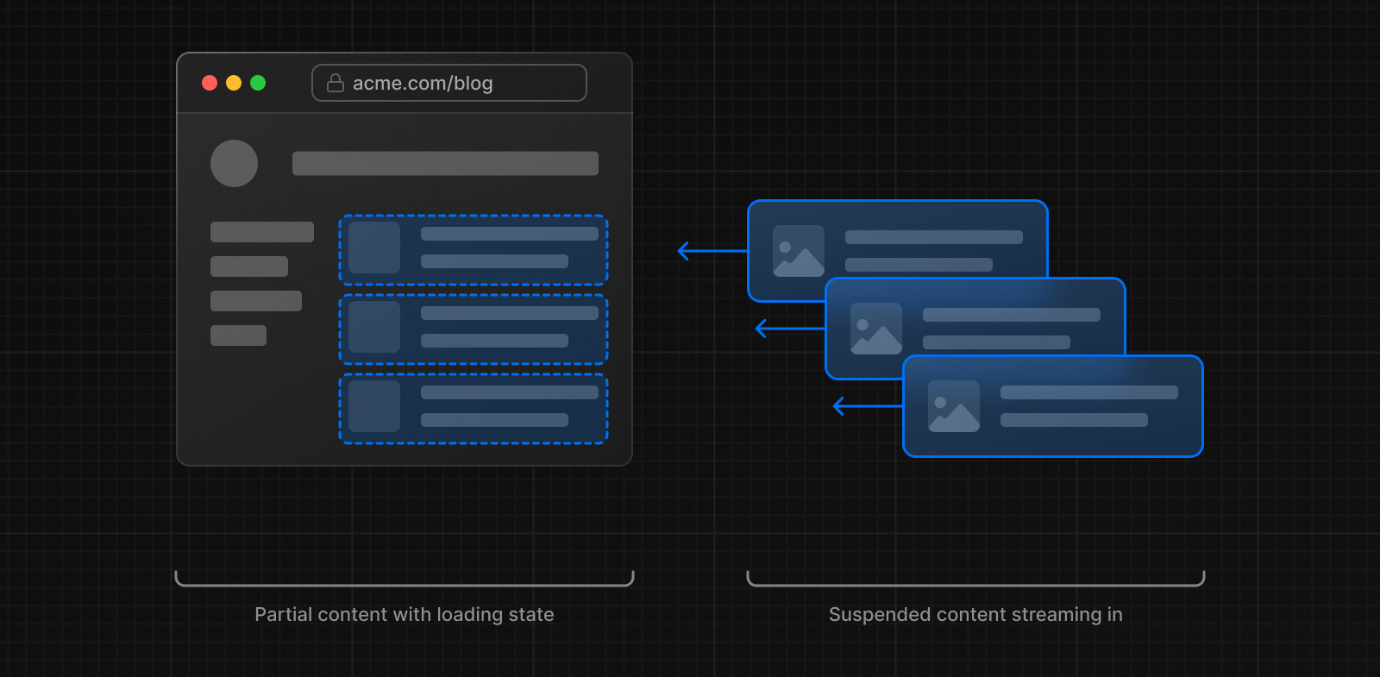
These steps are sequential and blocking, meaning the server can only render the HTML for a page once all the data has been fetched. And, on the client, React can only hydrate the UI once the code for all components in the page has been downloaded.

SSR with React and Next.js helps improve the perceived loading performance by showing a non-interactive page to the user as soon as possible.



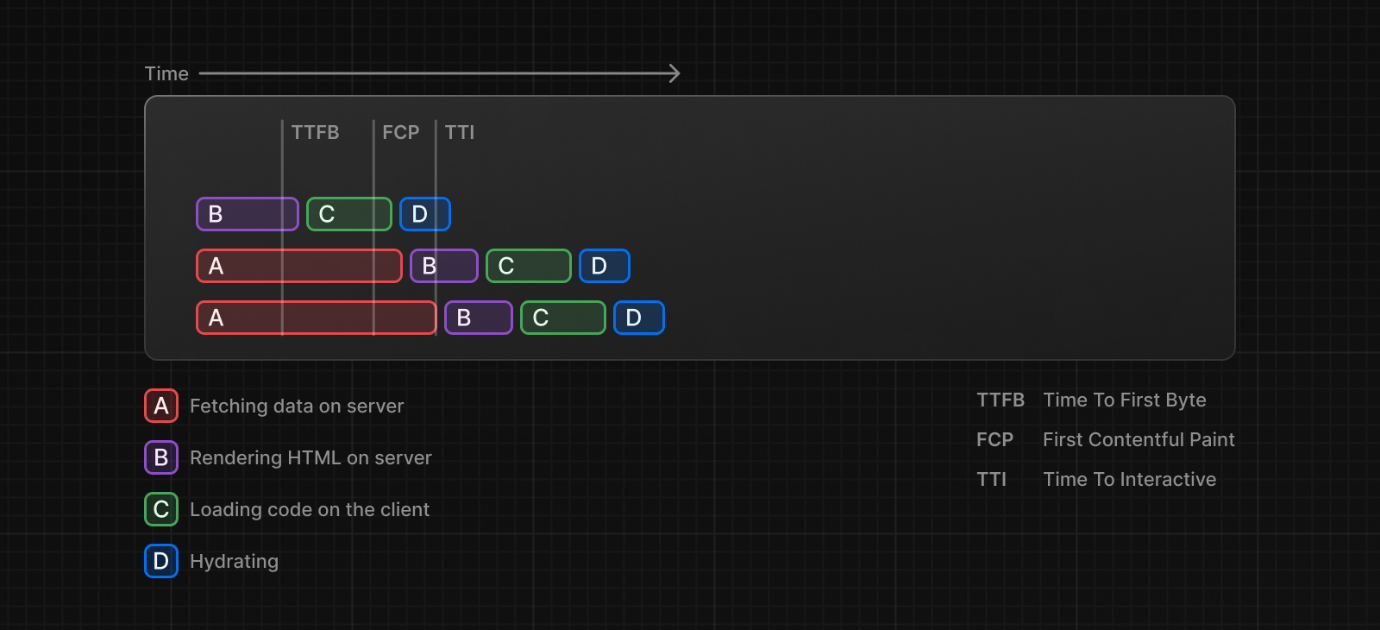
However, it can still be slow as all data fetching on server needs to be completed before the page can be shown to the user.

**Streaming** allows you to break down the page's HTML into smaller chunks and progressively send those chunks from the server to the client.



This enables parts of the page to be displayed sooner, without waiting for all the data to load before any UI can be rendered.

Streaming works well with React's component model because each component can be considered a chunk. Components that have higher priority (e.g. product information) or that don't rely on data can be sent first (e.g. layout), and React can start hydration earlier. Components that have lower priority (e.g. reviews, related products) can be sent in the same server request after their data has been fetched.



Streaming is particularly beneficial when you want to prevent long data requests from blocking the page from rendering as it can reduce the [Time To First Byte (TTFB)](https://web.dev/ttfb/) and [First Contentful Paint (FCP)](https://web.dev/first-contentful-paint/). It also helps improve [Time to Interactive (TTI)](https://developer.chrome.com/en/docs/lighthouse/performance/interactive/), especially on slower devices.

**Example**

<Suspense> works by wrapping a component that performs an asynchronous action (e.g. fetch data), showing fallback UI (e.g. skeleton, spinner) while it's happening, and then swapping in your component once the action completes.

import { Suspense } from 'react'

import { PostFeed, Weather } from './Components'

export default function Posts() {

return (

<section>

<Suspense fallback={<p>Loading feed...</p>}>

<PostFeed />

</Suspense>

<Suspense fallback={<p>Loading weather...</p>}>

<Weather />

</Suspense>

</section>

)

}

By using Suspense, you get the benefits of:

1. Streaming Server Rendering - Progressively rendering HTML from the server to the client.
2. Selective Hydration - React prioritizes what components to make interactive first based on user interaction.

For more Suspense examples and use cases, please see the React Documentation(<https://react.dev/reference/react/Suspense> ).

**SEO**

Next.js will wait for data fetching inside generateMetadata (<https://nextjs.org/docs/app/api-reference/functions/generate-metadata> ) to complete before streaming UI to the client. This guarantees the first part of a streamed response includes <head> tags.

Since streaming is server-rendered, it does not impact SEO. You can use the Mobile Friendly(<https://search.google.com/test/mobile-friendly> ) Test tool from Google to see how your page appears to Google's web crawlers and view the serialized HTML (<https://web.dev/rendering-on-the-web/#seo-considerations> ).