React 18

React 18 is now available on npm!

*  React 18 Library Alpha (Complete)
*  React 18 Public Beta (Complete)
*  **React 18 RC (Available now)**
*  React 18 (~2-4 weeks after RC)

Many of the features in React 18 are built on top of our new concurrent renderer, a behind-the-scenes change that unlocks powerful new capabilities. Concurrent React is opt-in — it’s only enabled when you use a concurrent feature — but we think it will have a big impact on the way people build applications.

**What is Concurrent React?**

## Gradually Adopting Concurrent Features

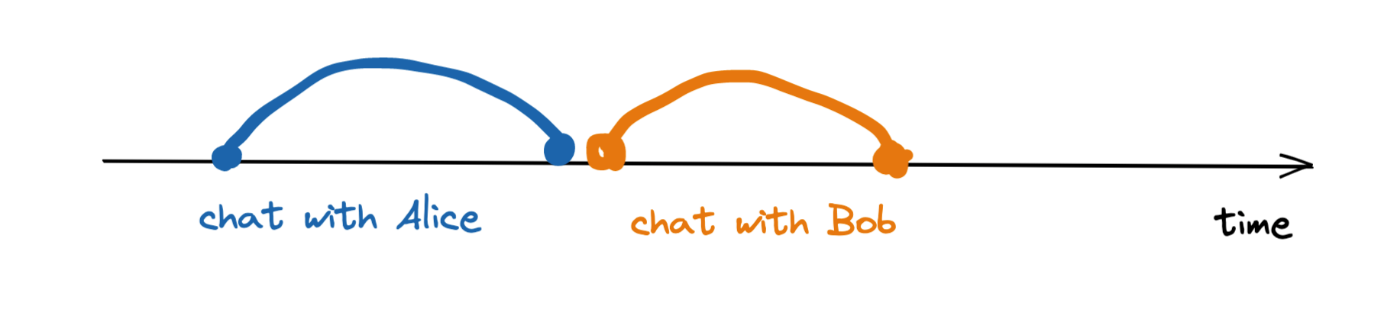
## Suspense in Data Frameworks

**Concurrency**

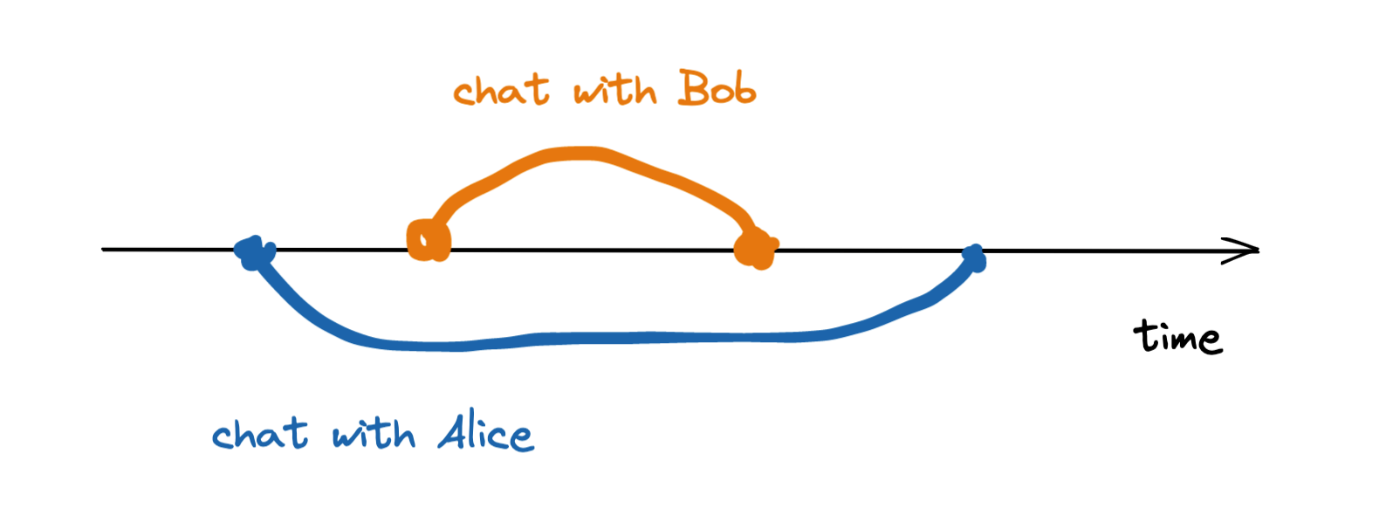
Concurrency means that tasks can overlap.

Let's use phone calls as an analogy.

No concurrency means that I can only have one phone conversation at a time. If I talk to Alice, and Bob calls me, I have to finish the call with Alice before I can talk to Bob.

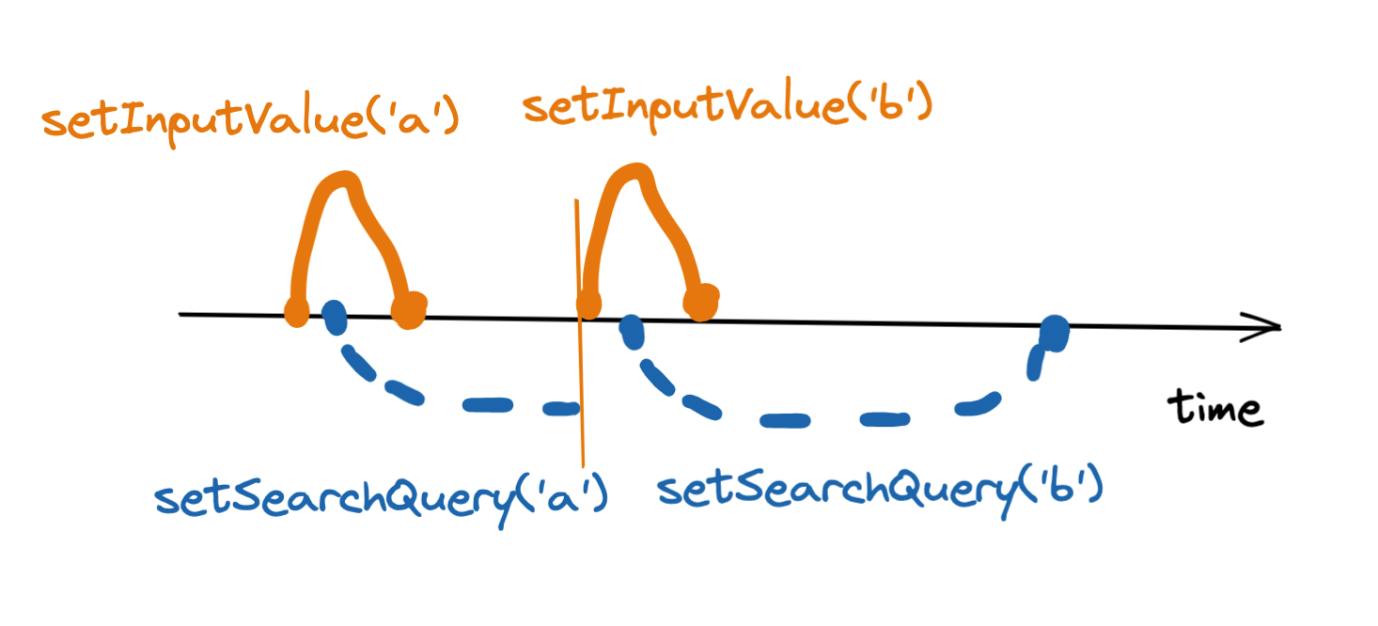
[](https://user-images.githubusercontent.com/810438/121394782-9be1e380-c949-11eb-87b0-40cd17a1a7b0.png)

Concurrency means that I can have more than one conversation at a time. For example, I can put Alice on hold, talk to Bob for a bit, and then switch back to talking to Alice.

[](https://user-images.githubusercontent.com/810438/121394880-b4ea9480-c949-11eb-989e-06a95edb8e76.png)

Note that concurrency doesn't necessarily mean I talk to *two people at once*. It just means that at any moment, I may be in multiple calls, and I choose who to talk to. For example, based on which conversation is more urgent.

Now, to translate the analogy, in React's case "phone calls" are your setState calls. Previously, React could only work on one state update at a time. So all updates were "urgent": once you start re-rendering, you can't stop. But with [startTransition](https://github.com/reactwg/react-18/discussions/41), you can mark a non-urgent update as a transition.a

[](https://user-images.githubusercontent.com/810438/121396132-f760a100-c94a-11eb-959c-b95a6647d759.png)

You can think of "urgent" setState updates as similar to urgent phone calls (e.g. your friend needs your help) while transitions are like relaxed conversations that can be put on hold or even interrupted if they're no longer relevant.

This illustrates how UI libraries, including React, typically work today. Once they start rendering an update, including creating new DOM nodes and running the code inside components, they can’t interrupt this work. We’ll call this approach “blocking rendering”.

In Concurrent Mode, rendering is not blocking. It is interruptible. This improves the user experience. It also unlocks new features that weren’t possible before. Before we look at concrete examples in the [next](https://17.reactjs.org/docs/concurrent-mode-suspense.html) [chapters](https://17.reactjs.org/docs/concurrent-mode-patterns.html), we’ll do a high-level overview of new features.

# How to Upgrade to React 18

To install the latest version of React:

npm install react react-dom

**ReactDOM.render is no longer supported in React 18. Use createRoot instead. Until you switch to the new API, your app will behave as if it’s running React 17. Learn more:**[**https://reactjs.org/link/switch-to-createroot**](https://reactjs.org/link/switch-to-createroot)

## Updates to Client Rendering APIs

React 18 introduces a new root API which provides better ergonomics for managing roots. The new root API also enables the new concurrent renderer, which allows you to opt-into concurrent features.



We’ve also changed unmountComponentAtNode to root.unmount:

Shape

Description automatically generated with medium confidence

We’ve also removed the callback from render, since it usually does not have the expected result when using Suspense:

Text

Description automatically generated

## Updates to TypeScript definitions

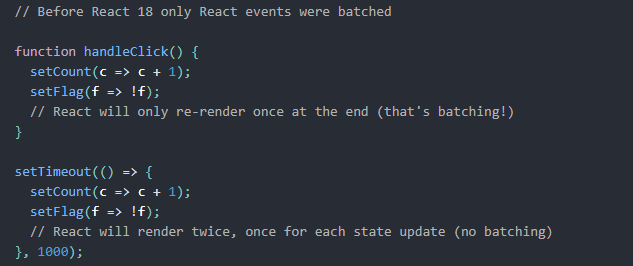
If your project uses TypeScript, you will need to update your @types/react and @types/react-dom dependencies to the latest versions. The new types are safer and catch issues that used to be ignored by the type checker. The most notable change is that the children prop now needs to be listed explicitly when defining props, for example:

Graphical user interface, text, application

Description automatically generated

Automatic Batching

React 18 adds out-of-the-box performance improvements by doing more batching by default. Batching is when React groups multiple state updates into a single re-render for better performance. Before React 18, we only batched updates inside React event handlers. Updates inside of promises, setTimeout, native event handlers, or any other event were not batched in React by default:



Starting in React 18 with createRoot, all updates will be automatically batched, no matter where they originate from. This means that updates inside of timeouts, promises, native event handlers or any other event will batch the same way as updates inside of React events:

Text

Description automatically generated

This is a breaking change, but we expect this to result in less work rendering, and therefore better performance in your applications. To opt-out of automatic batching, you can use flushSync:

Text

Description automatically generated

For more information, see the [Automatic batching deep dive](https://github.com/reactwg/react-18/discussions/21).

### **Transitions**

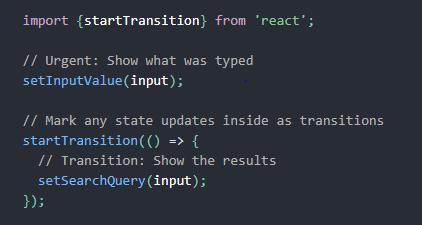
A transition is a new concept in React to distinguish between urgent and non-urgent updates.

* **Urgent updates** reflect direct interaction, like typing, clicking, pressing, and so on.
* **Transition updates** transition the UI from one view to another.

Urgent updates like typing, clicking, or pressing, need immediate response to match our intuitions about how physical objects behave. Otherwise they feel “wrong”. However, transitions are different because the user doesn’t expect to see every intermediate value on screen.

For example, when you select a filter in a dropdown, you expect the filter button itself to respond immediately when you click. However, the actual results may transition separately. A small delay would be imperceptible and often expected. And if you change the filter again before the results are done rendering, you only care to see the latest results.

Typically, for the best user experience, a single user input should result in both an urgent update and a non-urgent one. You can use startTransition API inside an input event to inform React which updates are urgent and which are “transitions”:



Updates wrapped in startTransition are handled as non-urgent and will be interrupted if more urgent updates like clicks or key presses come in. If a transition gets interrupted by the user (for example, by typing multiple characters in a row), React will throw out the stale rendering work that wasn’t finished and render only the latest update.

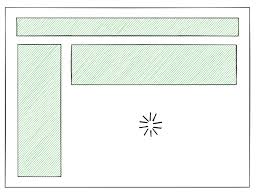
* useTransition: a hook to start transitions, including a value to track the pending state.
* startTransition: a method to start transitions when the hook cannot be used.

Transitions will opt in to concurrent rendering, which allows the update to be interrupted. If the content re-suspends, transitions also tell React to continue showing the current content while rendering the transition content in the background (see the [Suspense RFC](https://github.com/reactjs/rfcs/blob/main/text/0213-suspense-in-react-18.md) for more info).

[See docs for transitions here](https://reactjs.org/docs/react-api.html#transitions).

### **Suspense Features**

Suspense lets you declaratively specify the loading state for a part of the component tree if it’s not yet ready to be displayed.



Suspense makes the “UI loading state” a first-class declarative concept in the React programming model. This lets us build higher-level features on top of it.

We introduced a limited version of Suspense several years ago. However, the only supported use case was code splitting with React.lazy, and it wasn’t supported at all when rendering on the server.

In React 18, we’ve added support for Suspense on the server and expanded its capabilities using concurrent rendering features.

Suspense in React 18 works best when combined with the transition API. If you suspend during a transition, React will prevent already-visible content from being replaced by a fallback. Instead, React will delay the render until enough data has loaded to prevent a bad loading state.

For more, see the RFC for [Suspense in React 18](https://github.com/reactjs/rfcs/blob/main/text/0213-suspense-in-react-18.md).

### **New Strict Mode Behaviors**

In the future, we’d like to add a feature that allows React to add and remove sections of the UI while **preserving state.** For example, when a user tabs away from a screen and back, React should be able to immediately show the previous screen. To do this, React would unmount and remount trees using the same component state as before.

To help surface these issues, React 18 introduces a new development-only check to Strict Mode. This new check will automatically unmount and remount every component, whenever a component mounts for the first time, restoring the previous state on the second mount.

### Ensuring reusable state

To demonstrate the development behavior you’ll see in Strict Mode with this feature, consider what happens when React mounts a new component. Without this change, when a component mounts, React creates the effects:

Text

Description automatically generated

With Strict Mode starting in React 18, whenever a component mounts in development, React will simulate immediately unmounting and remounting the component:

Text

Description automatically generated

On the second mount, React will restore the state from the first mount. This feature simulates user behavior such as a user tabbing away from a screen and back, ensuring that code will properly handle state restoration.

When the component unmounts, effects are destroyed as normal:

Text

Description automatically generated

Unmounting and remounting includes:

* componentDidMount
* componentWillUnmount
* useEffect
* useLayoutEffect
* useInsertionEffect

Timeline

Description automatically generated

### **New Hooks**

#### **useId**

useId is a new hook for generating unique IDs on both the client and server, while avoiding hydration mismatches. It is primarily useful for component libraries integrating with accessibility APIs that require unique IDs. This solves an issue that already exists in React 17 and below, but it’s even more important in React 18 because of how the new streaming server renderer delivers HTML out-of-order.

**Note :** useId is **not** for generating [keys in a list](https://reactjs.org/docs/lists-and-keys.html#keys). Keys should be generated from your data.

#### **useTransition**

useTransition and startTransition let you mark some state updates as not urgent. Other state updates are considered urgent by default. React will allow urgent state updates (for example, updating a text input) to interrupt non-urgent state updates (for example, rendering a list of search results).

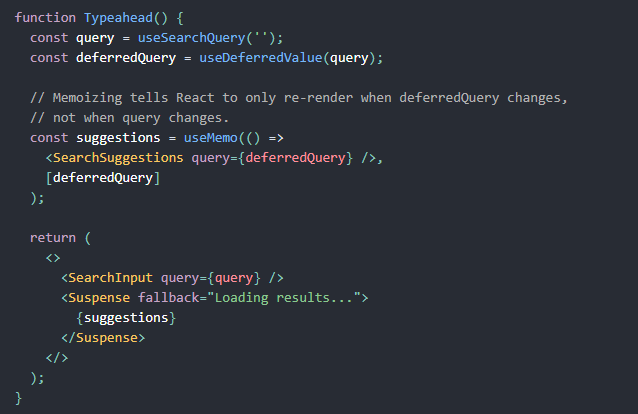
**useDeferredValue**

useDeferredValue lets you defer re-rendering a non-urgent part of the tree. It is similar to debouncing, but has a few advantages compared to it.

useDeferredValue accepts a value and returns a new copy of the value that will defer to more urgent updates. If the current render is the result of an urgent update, like user input, React will return the previous value and then render the new value after the urgent render has completed.

#### Memoizing deferred children

useDeferredValue only defers the value that you pass to it. If you want to prevent a child component from re-rendering during an urgent update, you must also memoize that component with [React.memo](https://reactjs.org/docs/react-api.html" \l "reactmemo) or [React.useMemo](https://reactjs.org/docs/hooks-reference.html" \l "usememo):



Memoizing the children tells React that it only needs to re-render them when deferredQuery changes and not when query changes. This caveat is not unique to useDeferredValue, and it’s the same pattern you would use with similar hooks that use debouncing or throttling.

#### **useSyncExternalStore**

useSyncExternalStore is a new hook that allows external stores to support concurrent reads by forcing updates to the store to be synchronous. It removes the need for useEffect when implementing subscriptions to external data sources, and is recommended for any library that integrates with state external to React.

#### **useInsertionEffect**

useInsertionEffect is a new hook that allows CSS-in-JS libraries to address performance issues of injecting styles in render. Unless you’ve already built a CSS-in-JS library we don’t expect you to ever use this. This hook will run after the DOM is mutated, but before layout effects read the new layout. This solves an issue that already exists in React 17 and below, but is even more important in React 18 because React yields to the browser during concurrent rendering, giving it a chance to recalculate layout.

**Deprecations**

* react-dom: ReactDOM.render has been deprecated. Using it will warn and run your app in React 17 mode.
* react-dom: ReactDOM.hydrate has been deprecated. Using it will warn and run your app in React 17 mode.
* react-dom: ReactDOM.unmountComponentAtNode has been deprecated.
* react-dom: ReactDOM.renderSubtreeIntoContainer has been deprecated.
* react-dom/server: ReactDOMServer.renderToNodeStream has been deprecated.

## Other Notable Changes

* **Components can now render undefined:** React no longer warns if you return undefined from a component. This makes the allowed component return values consistent with values that are allowed in the middle of a component tree. We suggest to use a linter to prevent mistakes like forgetting a return statement before JSX.
* **In tests, act warnings are now opt-in:** If you’re running end-to-end tests, the act warnings are unnecessary. We’ve introduced an [opt-in](https://github.com/reactwg/react-18/discussions/102) mechanism so you can enable them only for unit tests where they are useful and beneficial.
* **No warning about setState on unmounted components:** Previously, React warned about **memory leaks** when you call setState on an unmounted component. This warning was added for subscriptions, but people primarily run into it in scenarios where setting state is fine, and workarounds make the code worse. We’ve [removed](https://github.com/facebook/react/pull/22114) this warning.
* **No suppression of console logs:** When you use Strict Mode, React renders each component twice to help you find unexpected side effects. In React 17, we’ve suppressed console logs for one of the two renders to make the logs easier to read. In response to [community feedback](https://github.com/facebook/react/issues/21783) about this being confusing, we’ve removed the suppression. Instead, if you have React DevTools installed, the second log’s renders will be displayed in grey, and there will be an option (off by default) to suppress them completely.
* **Improved memory usage:** React now cleans up more internal fields on unmount, making the impact from unfixed memory leaks that may exist in your application code less severe.

## Dropping Support for Internet Explorer

In this release, React is dropping support for Internet Explorer, which is [going out of support on June 15, 2022](https://blogs.windows.com/windowsexperience/2021/05/19/the-future-of-internet-explorer-on-windows-10-is-in-microsoft-edge). We’re making this change now because new features introduced in React 18 are built using modern browser features such as microtasks which cannot be adequately polyfilled in IE.

**If you need to support Internet Explorer we recommend you stay with React 17.**

## 

## Thank You

Abhishek Negi Inn-304