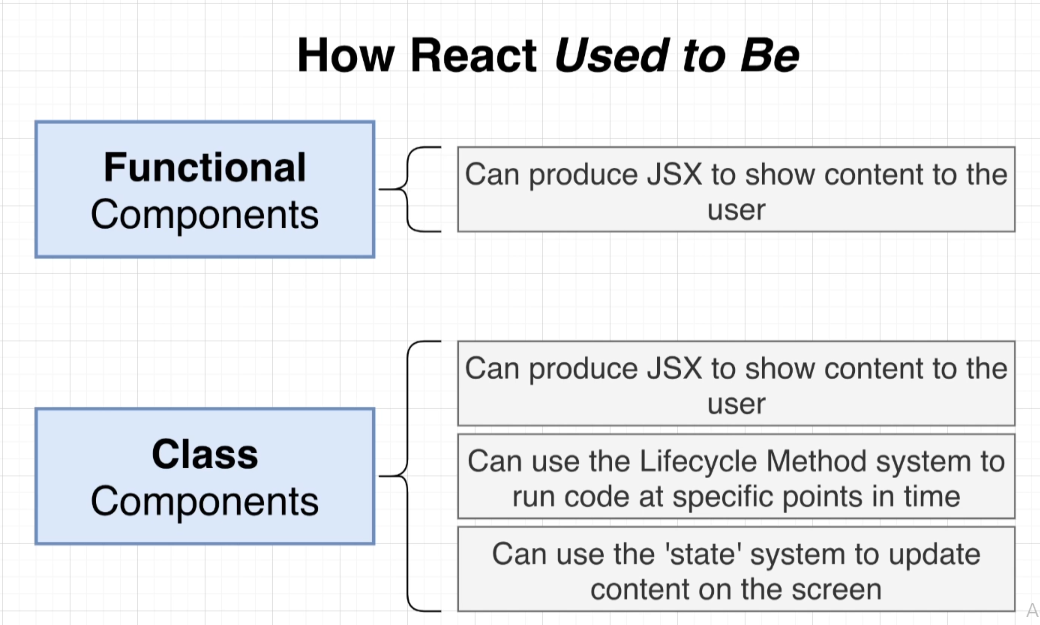
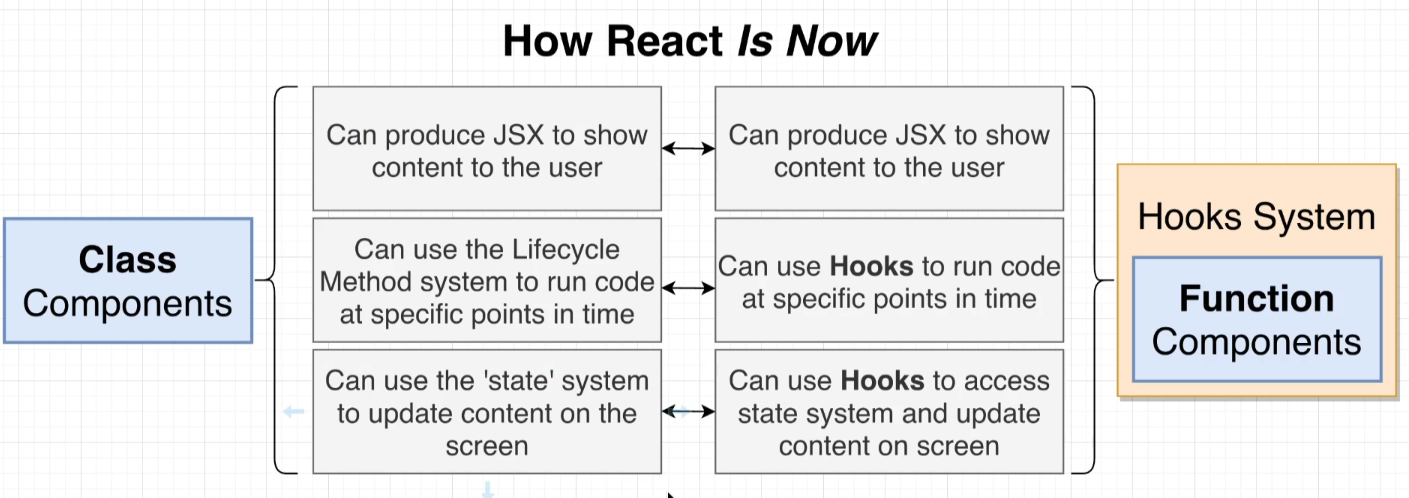
CLASS-BASED Components –



Functional components were much more restricted in nature – they did not have access to lifecycle method system or the state system. This meant we can only show content to our users.

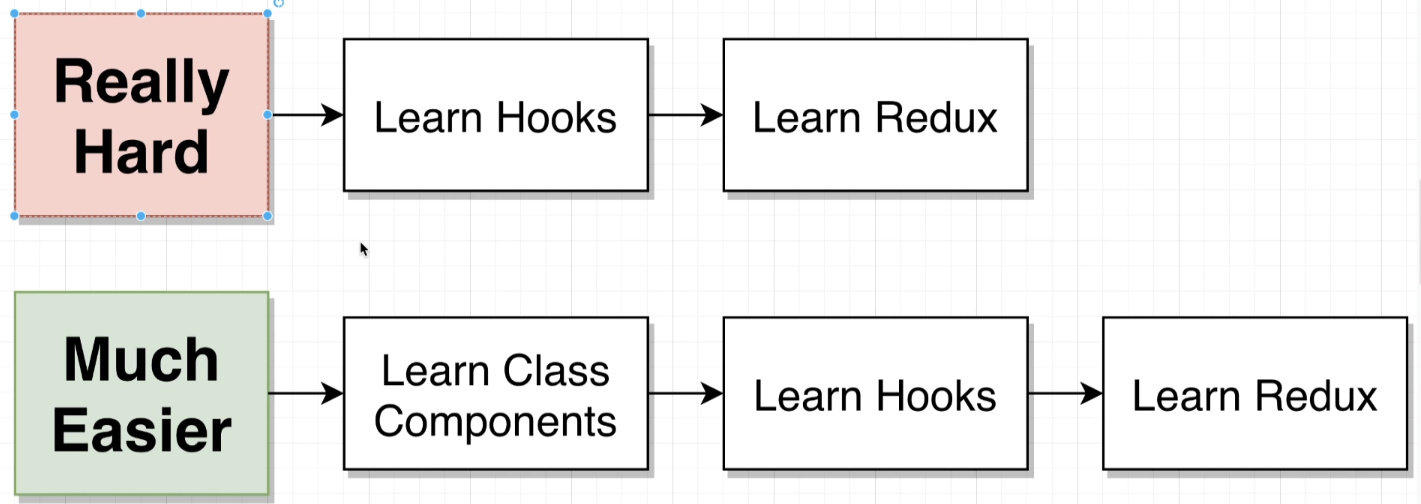
Now, it’s much later in the development cycle of React and functional components have gotten a bit fancier over time – through the introduction of **Hooks** system.

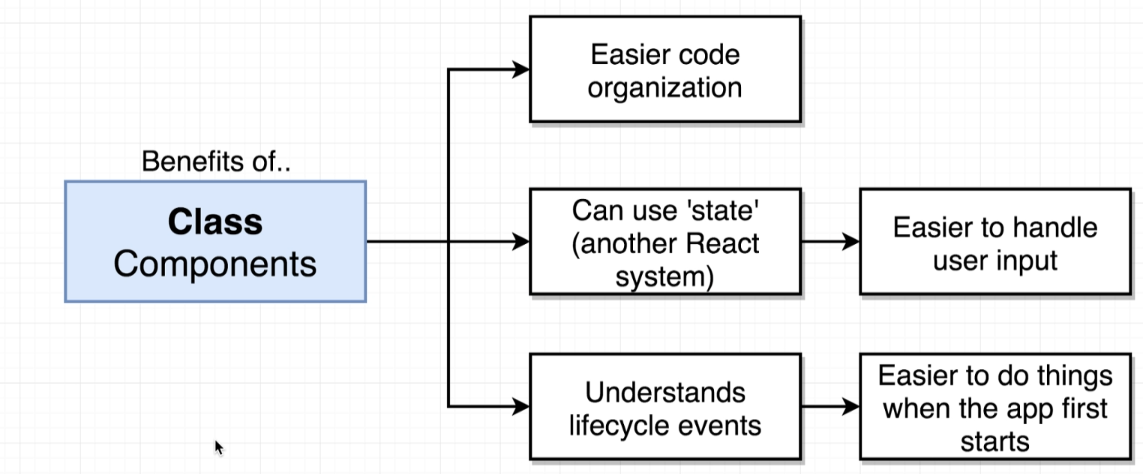


🡨--🡪 : roughly equivalent to.

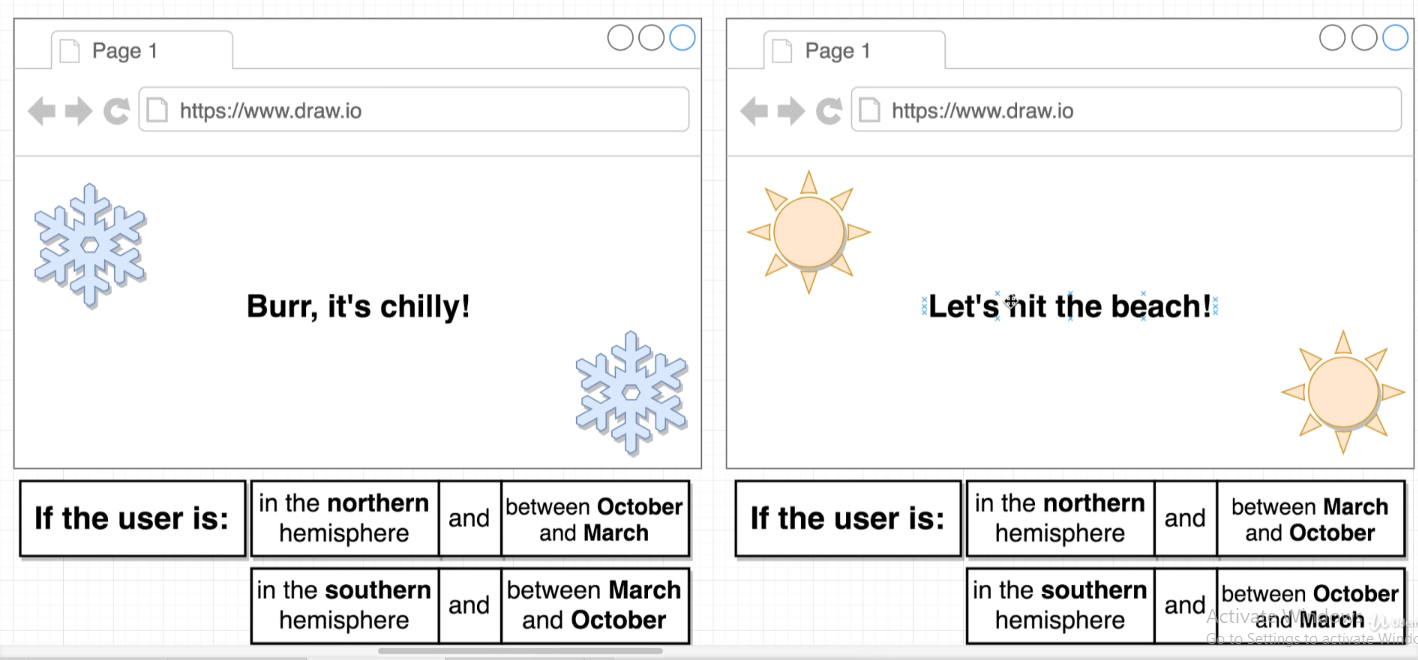
We need to understand both the approaches i.e. functional components and class-based components to get a good grasp on React.js

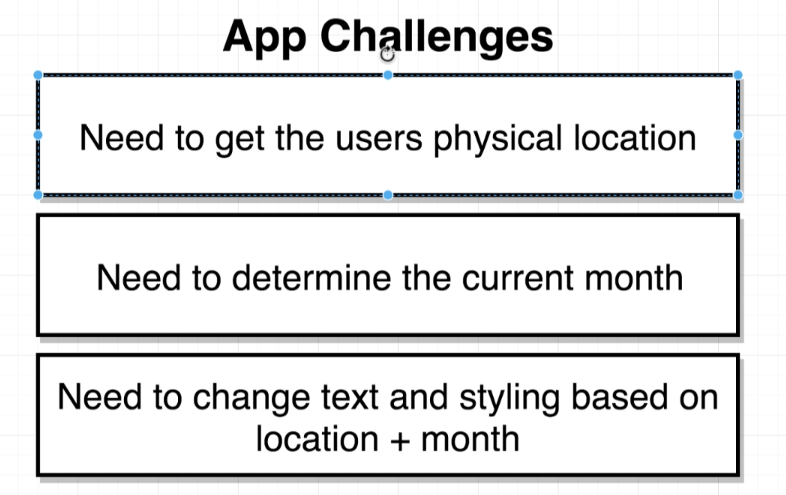
Stripe (a 36B$ company) uses both functional and class components.



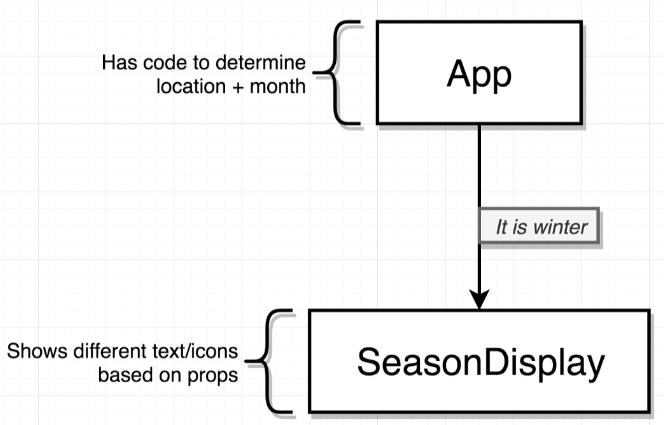


**Application we’re going to build –** (detect what season user is currently experiencing)





Flow of Action -



1. Somehow in the browser, we need to detect what the user’s physical location is.
2. Determining the current month – easy to use JS API’s to get date and time.
3. Based on (1) and (2), we need to dynamically change text and the styling.

In general, we need two components – an App component, bunch of code that’s going to be in-charge of determining the location of user and current month.

This App component, is now going to take the current season – winter or summer – and pass it as a prop as a second component, the SeasonDisplay component.

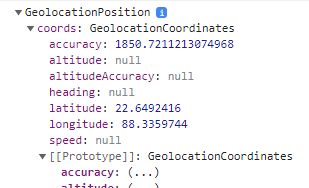
The SeasonDisplay component is in-charge of showing some actual icons and text based on season.

**Determining user’s physical location –** (Geolocation API)

We’re going to be using the **Geolocation API**, built into most modern Browsers. Using the MDN documentation, we’re going to get the user’s current position.

This is a function that is going to make an educated guess about where the user is physically located depending upon a couple of different parameters – IP address, different Wi-Fi networks available etc.





Now, we’ll use the position object’s latitude value to see whether we’re in the Northern or Southern Hemisphere.

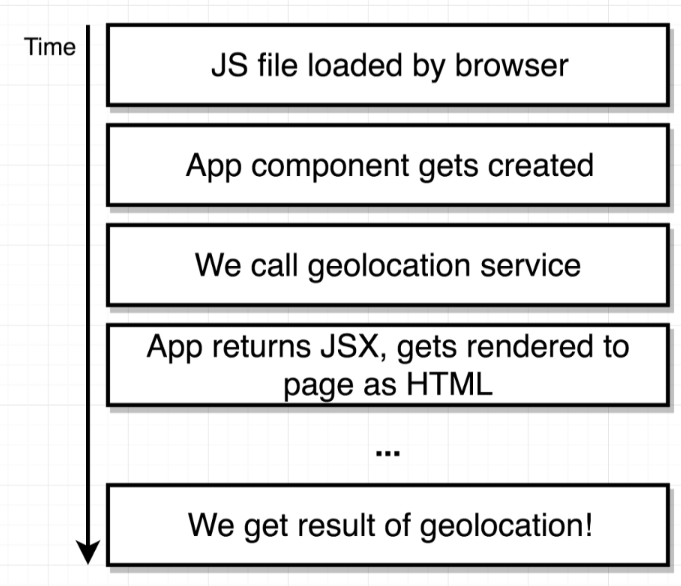
Let’s say we want to display the latitude value on the page now – instead of Hi There!

That’s pretty much the question that’ll lead us to understand why we have to use class-based components.



**LIFECYCLE –**

What is exactly happening inside our application as it starts up on the Browser? – This will make us understand why it will be really challenging for us to somehow put our latitude in the <div>…</div> if we continue using functional component.

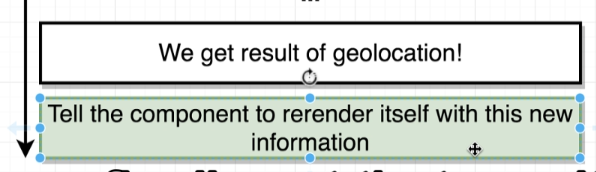


Timeline of what happens in our application on the user’s Browser –

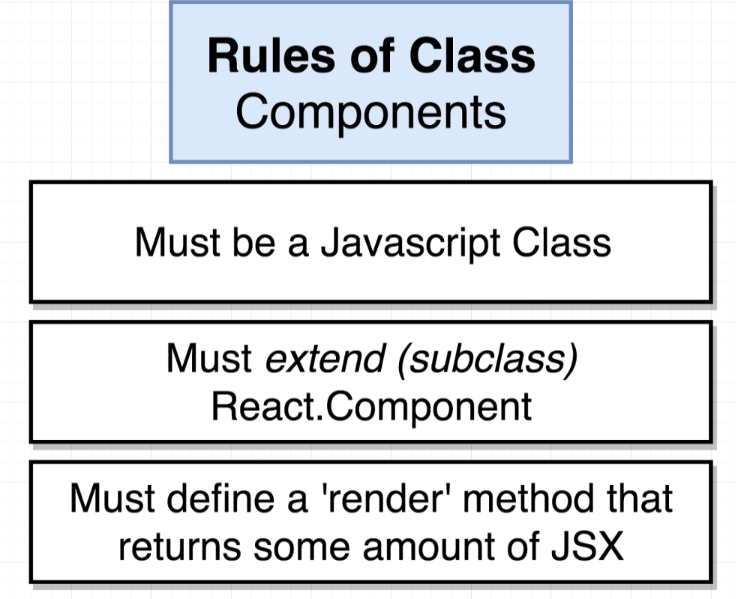
JS file gets loaded into our Browser -> Browser then interprets our JS file and sees that we’re trying to create an App component -> the App component gets created 🡪 at some point of time App component gets invoked -> when the function gets invoked, we the call Geolocation service -> but getting the geolocation result back takes some amount of time -> upon calling the API, our App instantly returns some amount of JSX (the JSX is eventually turned into HTML and then rendered out on the screen -> we see content on the screen far sooner than we ever get a result from geolocation API.

Thus, with this functional component, we don’t really have a good way of waiting until the success callback (from the Geolocation API) finishes and then taking the position and sticking it into some JSX and returning it.

The **solution** is a class-based component in conjunction with React state system.



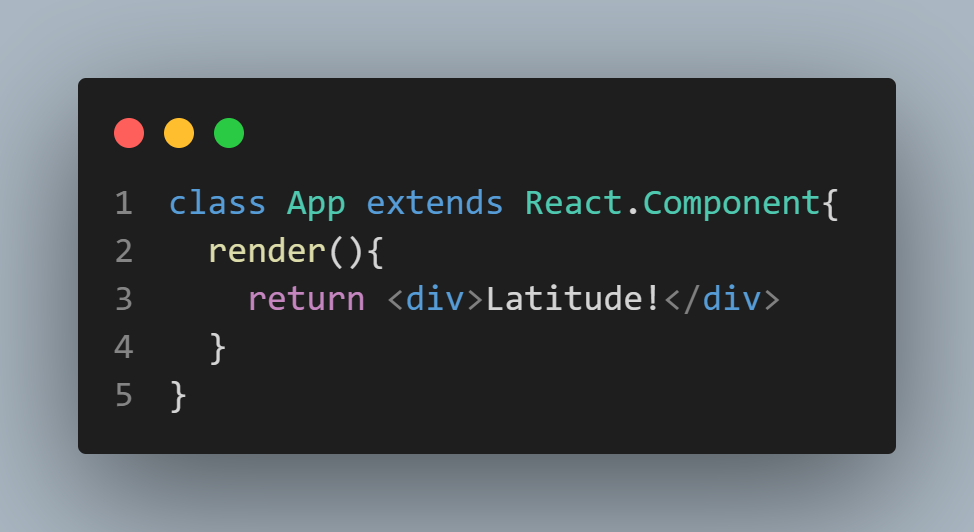
After we finally get the result, we want to tell our component to re-render itself – update the content on the screen with new information.



JavaScript inheritance works on prototypal inheritance, which is a little bit different than classic inheritance featured in Java or Ruby or similar OOP languages.

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain>

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes>



The reason we’re extending React.Component is that it allows us to pull a ton of built-in functionality from Component class to our class -> fancy way of saying we’re sub-classing React.Component (borrowing functionality into our App class).