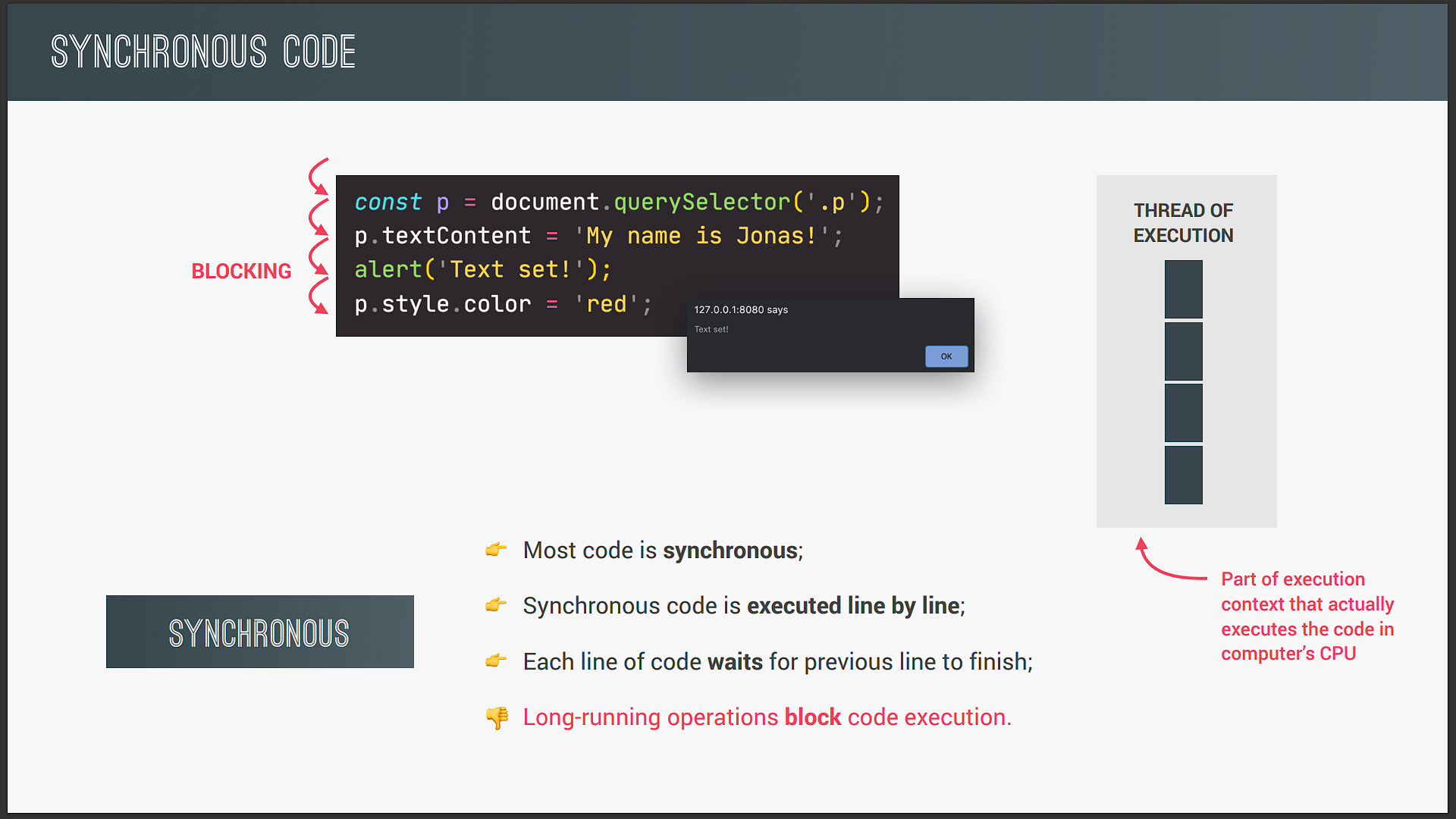
Let's start this section by understanding what asynchronous JavaScript actually is and also learn about the most popular use cases of asynchronous JavaScript which is basically to make socalled Ajax calls to APIs. Now to understand what asynchronous JavaScript code actually is we first need to understand what synchronous code is.



So basically the opposite of asynchronous. So most of the codes that we've been writing so far in the course has been synchronous code and synchronous simply means that the code is executed line by line in the exact order of execution that we defined in our code just like in this small example. So as the first line of code is reached in the execution it is simply executed in the execution of thread. Now don't worry about this execution of thread. It's not important here it's just to make a point of synchronous versus asynchronous code as you will see in the next slide. All you need to know is that the execution of thread is part of the execution context which does actually execute the code in the computer's processor. But anyway then the next line of code is executed and then the next one all in sequence. So each line of code always waits for the previous line to finish execution. Now this can create problems when one line of code takes a long time to run. For example in this current line of code we have an alert statement which creates this alert window. Now as we've experienced in the past this alert window will block the code execution right? So nothing will happen on the page until we click that OK Button. And only then the code can continue executing. And so the alert statement is a perfect example of a long running operation which blocks execution of the code. So again only after we click Okay the window disappears and the next line can run. So this is hopefully a nice illustration of the problem with synchronous code.

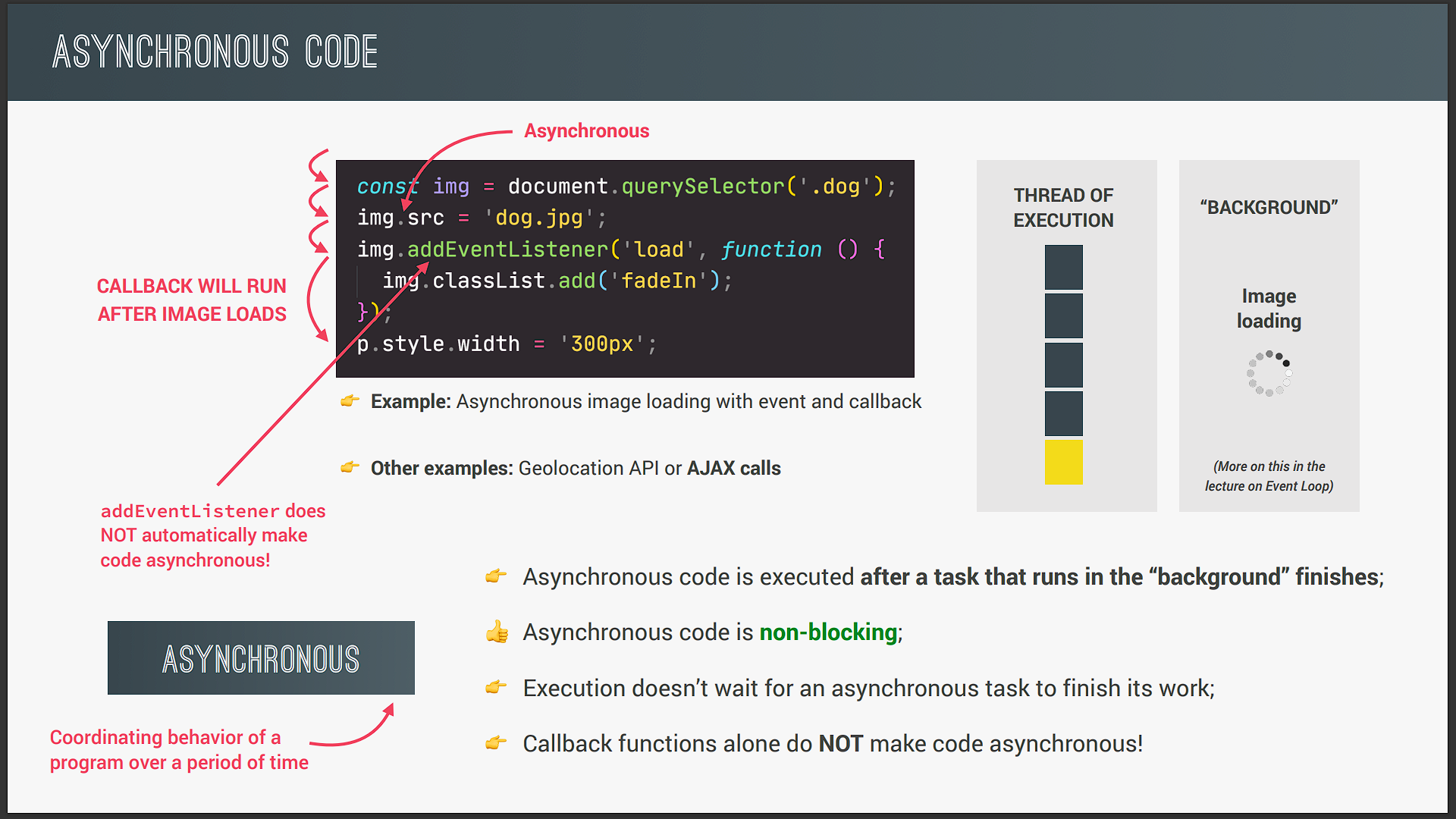
Now most of the time synchronous code is fine and makes perfect sense. But imagine that execution would have to wait for example for a five second timer to finish. That would just be terrible right? Because meanwhile nothing on the page would work during these five seconds. And so that's where asynchronous code comes into play. So this example contains the fivesecond timer that I just mentioned. And if you want you can pause the video here for a minute and analyze this code before we move on. Now anyway the first line of code is still synchronous here and we also move to the second line in a synchronous way. But here we encountered the set timeout function which will basically start a timer in an asynchronous way. So this means that the timer will essentially run in the background without preventing the main code from executing. We also register a callback function which will not be executed now but only after the timer has finished running. And we have actually already done this many times before in practice right? Now this callback function that I just mentioned is asynchronous JavaScript. And it is asynchronous because it's only going to be executed after a task that is running in the background finishes execution. And in this case that is the timer. So this callback that we just talked about is registered and then we immediately move on to the next line. So the main code is not being blocked and execution does not wait for the asynchronous timer to finish its work. And that's the big difference between synchronous and asynchronous code.



So previously we had to wait for the user to click on the alert window to continue execution. And again that's because alert is blocking synchronous code but now with this timer the callback is actually asynchronous. And so it's only going to be executed after the timer has finished. And so therefore we say that it's nonblocking because in the meantime the rest of the code can keep running normally. Now when the timer finally finishes after five seconds the callback function will finally be executed as well. So you'll see that this callback runs after all the other code even though in the code it doesn't appear at the end. And so basically an action was deferred into the future here in order to make the code nonblocking. And actually we already saw this behavior happening before when we first learned about timers we just didn't know that this is called asynchronous and nonblocking code.

So in summary asynchronous programming is all about coordinating the behavior of our program over a certain period of time. And this is essential to understand. So asynchronous literally means not occurring at the same time. And so that's what asynchronous programming is all about. All right. Now as we saw in this example we need a callback function to implement this asynchronous behavior right? However that does not mean that callback functions automatically make code asynchronous. That is just not the case okay? For example the Array map method accepts a callback function as well but that doesn't make the code asynchronous. Only certain functions such as set timeout work in an asynchronous way. We just have to know which ones do and which ones don't okay?

But please understand this very important fact that callback functions alone do not make code asynchronous that's essential to keep in mind. But anyway in order to really understand this let's see another example.

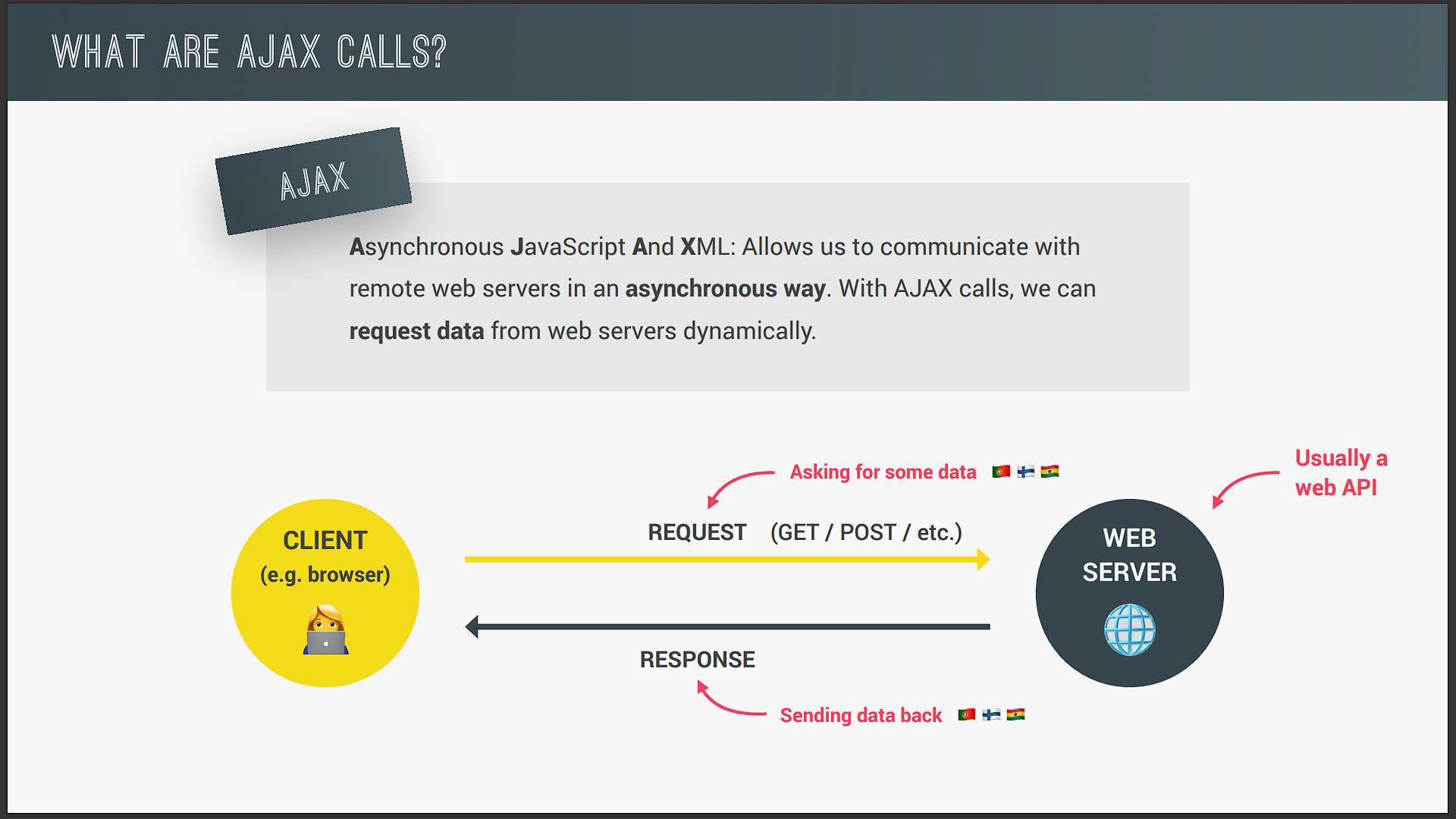


So this example is about loading an image. So the first two lines run in a synchronous way one after the other. Now in the second line we set the source attribute of the image that we selected in the first line. And this operation is actually asynchronous. So setting the source attribute of any image is essentially loading an image in the background while the rest of the code can keep running. And this makes sense right? Imagine that it's a huge image we wouldn't want our entire code to wait for the image to load. And that's why setting the source attribute was implemented in JavaScript in an asynchronous way.

Now once the image has finished loading a load event will automatically be emitted by JavaScript. And so we can then listen for that event in order to act on it. Listening for the load event is exactly what we do here in the next line as well. So here we use add event listener and to register a callback function for the load event. So just like in the previous example we provide a callback function that will be executed once the image has been loaded and not right away because again all this code is nonblocking. So instead of blocking execution moves on right to the next line immediately. Then once the image is completely loaded it's displayed on the webpage and the load event is admitted. And since we're listening for that event our callback function is finally executed.

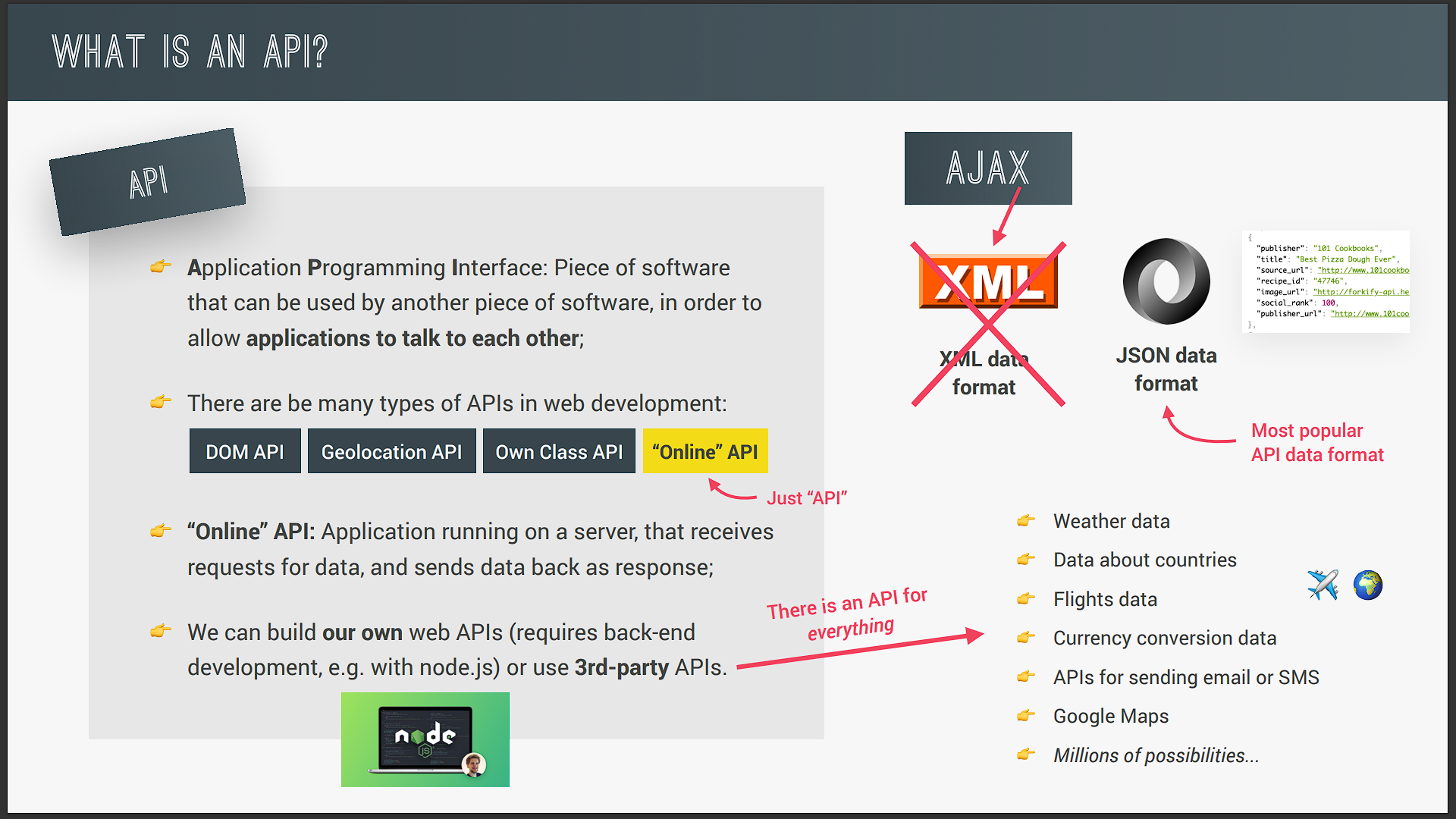
So once more we deferred an action into the future making the code asynchronous and nonblocking. All right. And so now at this point I believe that you have a pretty good understanding of asynchronous code. There's just one more important thing that I need to mention which is the fact that event listeners alone do not make code asynchronous just like callback functions alone do also not make code asynchronous. For example an event listener listening for a click on a button is not doing any work in the background. It's simply waiting for a click to happen but it's not doing anything. And so there is no asynchronous behavior involved at all. Now what makes this code example asynchronous is simply the fact that the image is loading asynchronously in the background but not the fact that we are listening for the load events to happen.

So what matters is the asynchronous behavior of a task like running a timer or loading an image. And there are more examples of asynchronous behavior in JavaScript like the Geolocation API that we used before or Ajax calls. And Ajax calls are probably the most important use case of asynchronous JavaScript. And so let's see what Ajax is all about.



So Ajax stands for asynchronous JavaScript and XML and basically it allows us to communicate with remote web servers in an asynchronous way. Now in practice we make Ajax calls in our code in order to request some data from a web server dynamically. So without reloading the page so that we can use that data in our application dynamically. For example right in the next video we're going to make Ajax calls to request data about countries. And we can then use that data about countries to build a small application that shows us information about the country that we're currently in. And the possibilities and use cases are endless but more about that in the next slide. For now let's quickly understand how Ajax works.

So let's say that we have our JavaScript application running in the browser which is also called the Client. And we want the application to get some data from a web server. And let's say the data about countries that I was talking about earlier. So with Ajax we can do an HTTP request to the server which has this data. And the server will then set back a response containing that data that we requested. And this back and forth between Client and server all happens asynchronously in the background just the way we learned before. And there can even be different types of requests like get requests to receive data or post requests to send data to a server. But there is a whole lecture about this a bit later to really show you how it all works in detail. Now when we're asking a server to send us some data this server usually contains a web API. And this API is the one that has the data that we're asking for. So an API is something pretty important and so let's now see what an API and web APIs actually are.



So first of all API stands for Application Programming Interface. Now in general terms and on the very high level of obstruction an API is basically a piece of software that can be used by another piece of software in order to basically allow applications to talk to each other and exchange information. And that's true not only for web development and JavaScript but for programming in general. Now in JavaScript and web development there are countless types of APIs like the DOM API or the Geolocation API that we have been using. So these are called APIs because they are a selfcontained piece of software that allow other pieces of software to interact with them. For example our Mapty application that we built in the previous section. Also we can always implement a small and simple API in a class where we make some methods available as a public interface. Remember?

So again objects made from a class can be seen as selfcontained encapsulated pieces of software that other pieces of software can interact with. And so that fits the definition of API right? But now let's talk about the important type of API that we are actually interested in when we use Ajax. And that are APIs that I like to call Online APIs.

So an online API is essentially an application running on a web server which receives requests for data then retrieves this data from some database and then sends it back to the client. So just as we saw in the last slide right? Now when building applications in practice we simply call these online APIs API and many people will also call these APIs Web APIs or again just simply API. So the term Online API is actually a term that I came up with myself because the term Web API is actually also used for other things. Now of course we can build or own Online APIs but that requires backend development. So development with servers and databases and all that. And this is something really interesting that you can learn after this course and I actually have a course on Node.js myself that you can check out if you're interested. But anyway for now we are interested in using rdparty APIs.

So APIs that other developers make available for us most of the times for free. So let's now imagine that you're building a traveling application and you have a database with different destinations and tours that you're offering. So on your own server you could build your own API that can receive requests from your front end application in JavaScript and send back the results. So that would be your own API hosted on your own server. But that alone would probably not be enough to build a complete application. And so you could also use some rdparty APIs. And there are really APIs for everything. So in our example travel application you could use an API to get weather data in your destinations data about the destination countries themselves data about flights about currency conversion. And you could even use APIs to send emails or text messages or embed Google maps into your applications. So as you see the possibilities are really endless with APIs and we can even say that APIs is what made the modern web as you know it possibly in the first place.

So using APIs in JavaScript is super popular and everyone does it all the time and so that's why I'm explaining you all these details. Okay. But now just to finish let's quickly talk about API data formats. So Ajax stands for asynchronous JavaScript and XML. Remember? So the X there stands for XML and XML is a data format which used to be widely used to transmit data on the web. However these days basically no API uses XML data anymore.

The term Ajax is just an old name that got very popular back in the day and so it's still used today even though we don't use XML anymore. So instead most APIs these days use the JSON data format. So JSON is the most popular data format today because it's basically just a JavaScript object but converted to a string. And so therefore it's very easy to send across the web and also to use in JavaScript once the data arrives. All right. So now that we know what asynchronous JavaScript is and also what Ajax and APIs are let's finally use all this in practice and make our very first Ajax call in the next video.