JavaScript TUTORIAL

JavaScript setTimeout

Summary: in this tutorial, you will learn how to use the JavaScript `setTimeout()` that sets a timer and executes a callback function after the timer expires.

Introduction to JavaScript setTimeout()

The `setTimeout()` is a method of the `window` object. The `setTimeout()` sets a timer and executes a callback function after the timer expires.

The following illustrates the syntax of `setTimeout()`:

```
let timeoutID  = setTimeout(cb [,delay], arg1, arg2,...);
```

In this syntax:

- `cb` is a callback function to be executed after the timer expires.
- `delay` is the time in milliseconds that the timer should wait before executing the callback function. If you omit it, the `delay` defaults to 0.
- `arg1`, `arg2`, ... are arguments passed to the `cb` callback function.

The `setTimeout()` returns a `timeoutID` which is a positive integer identifying the timer created as a result of calling the method.

The `timeoutID` can be used to cancel timeout by passing it to the `clearTimeout()` method.

JavaScript setTimeout() example

The following creates two simple buttons and hooks them to the `setTimeout()` and `clearTimeout()`.

When you click the `Show` button, the `showAlert()` is invoked and shows an alert dialog after 3 seconds. To cancel the timeout, you click the `Cancel` button.

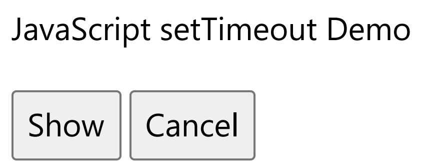
HTML

```html
<p>JavaScript setTimeout Demo</p>
<button onclick="showAlert();">Show</button>
<button onclick="cancelAlert();">Cancel</button>
```

JavaScript

```javascript
var timeoutID;

function showAlert() {
    timeoutID = setTimeout(alert, 3000, 'setTimeout Demo!');
}

function clearAlert() {
    clearTimeout(timeoutID);
}
```

Output


JavaScript setTimeout Demo

How JavaScript setTimeout() works

JavaScript is single-threaded therefore it can only do one task at a time. It means that it can only carry a single task a given time. Besides the JavaScript engine, the web browser has other components such as Event Loop, Call Stack, and Web API.

When you call the `setTimeout()`, the JavaScript engine creates a new function execution context and places it on the call stack.

The `setTimeout()` executes and creates a timer in the Web APIs component of the web browser. When the timer expires, the callback function that was passed in the `setTimeout()` is placed to the callback queue.

The event loop monitors both the call stack and the callback queue. It removes the callback function from the callback queue and places it to call stack when the call stack is empty.

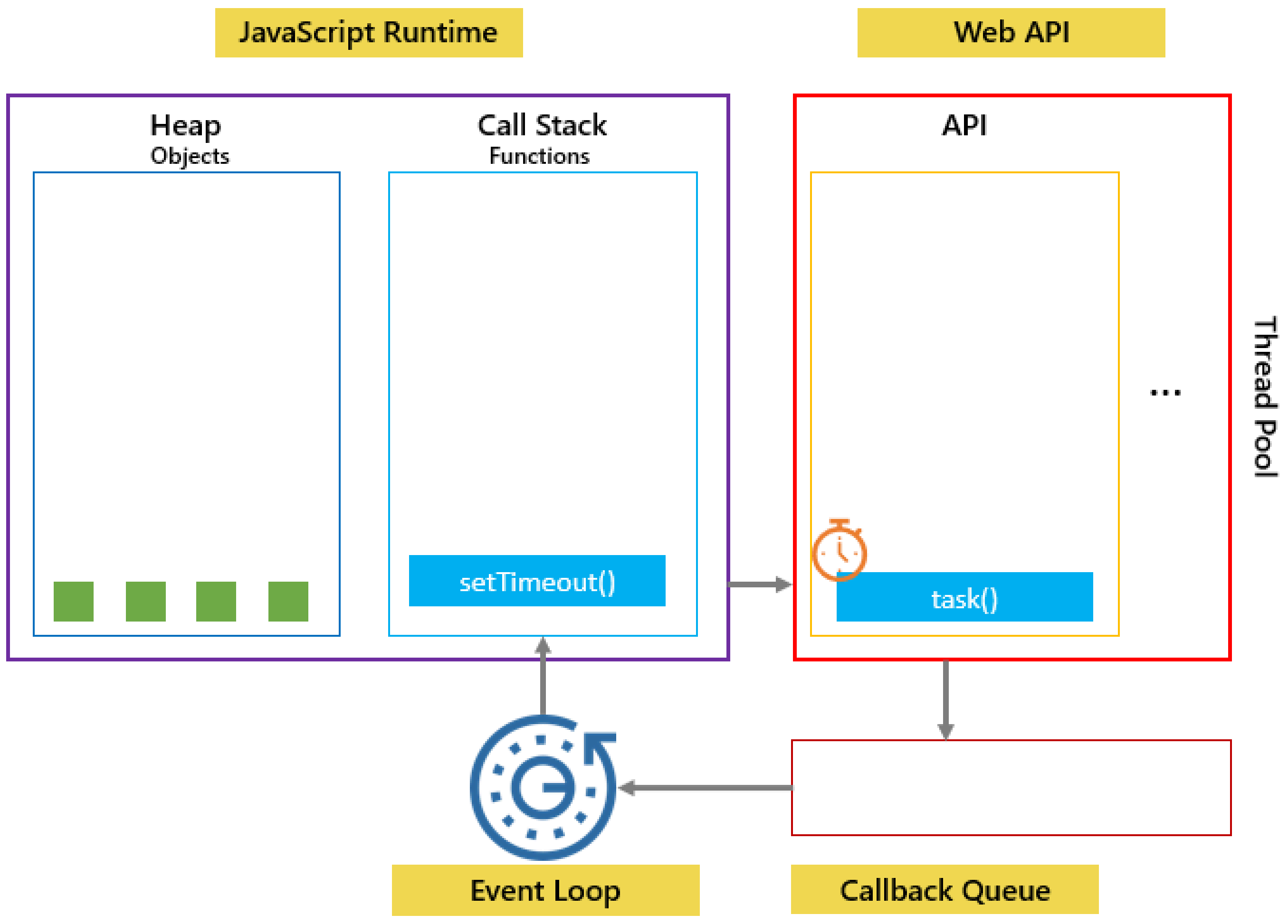
Once the callback function is on the call stack, it is executed.

See the following example:

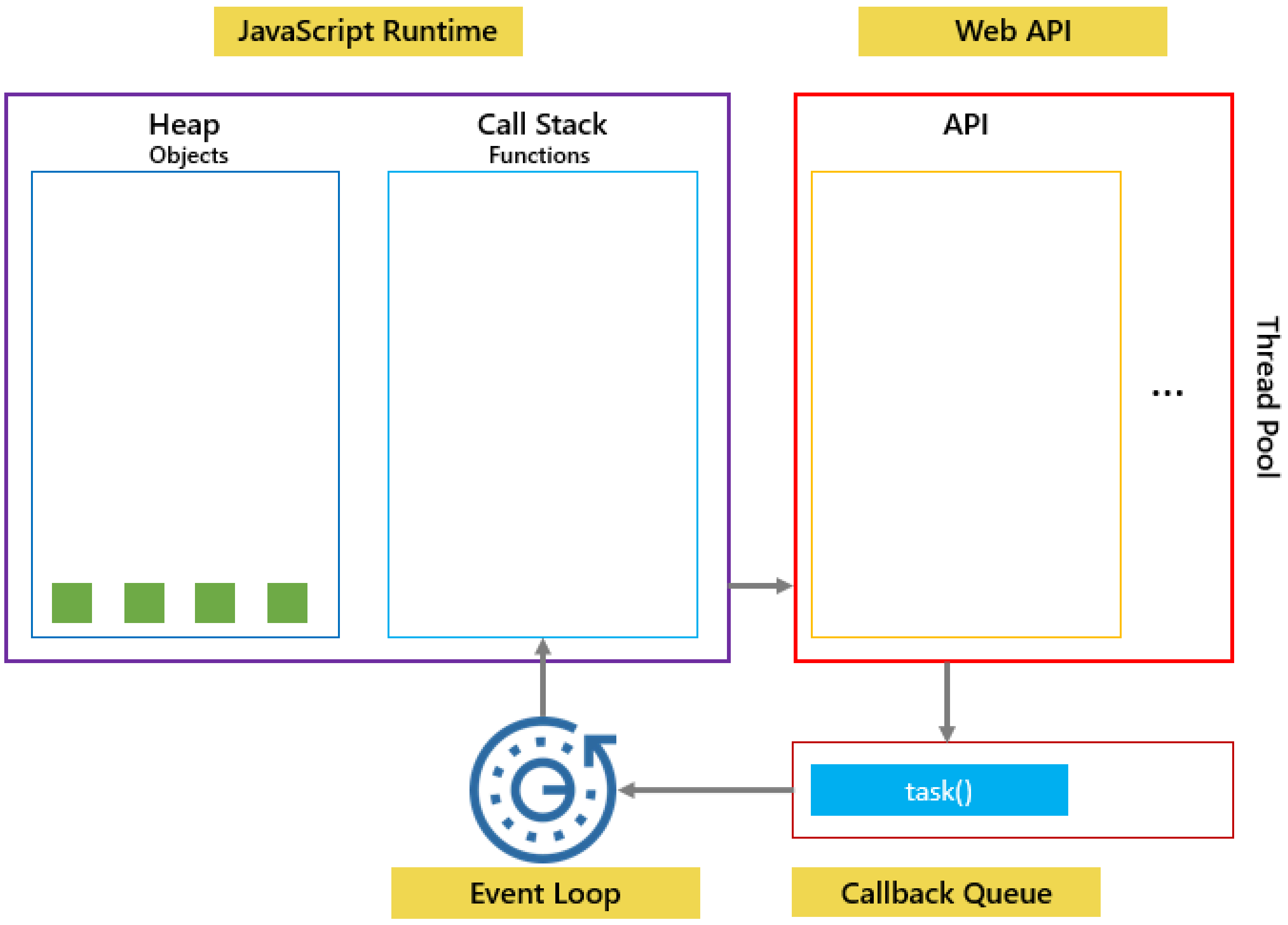
```
function task() {
    console.log('setTimeout Demo!')
}

setTimeout(task, 3000);
```

In this example:

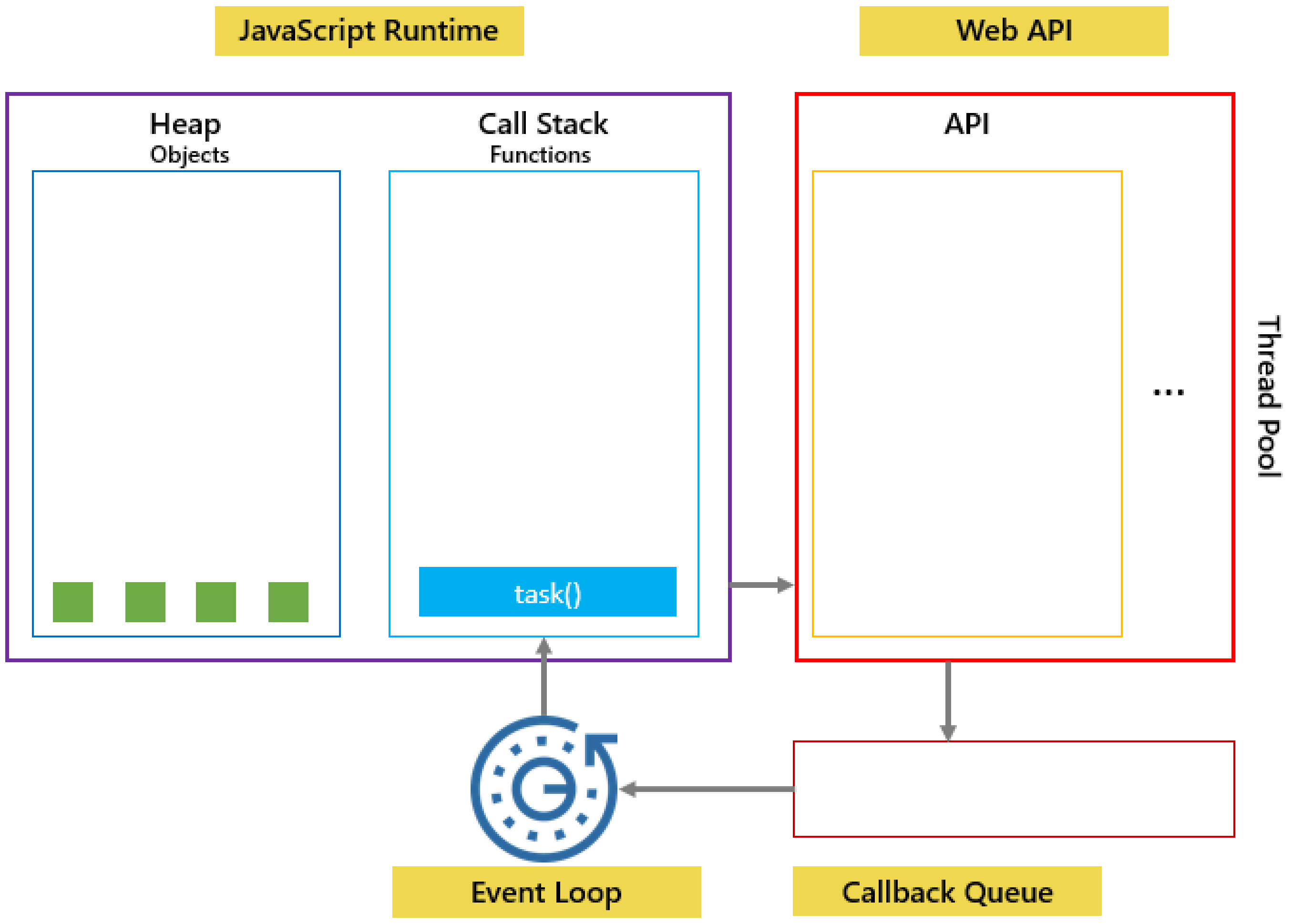
First, the `setTimeout()` is placed on the call stack. It creates a timer on Web API.



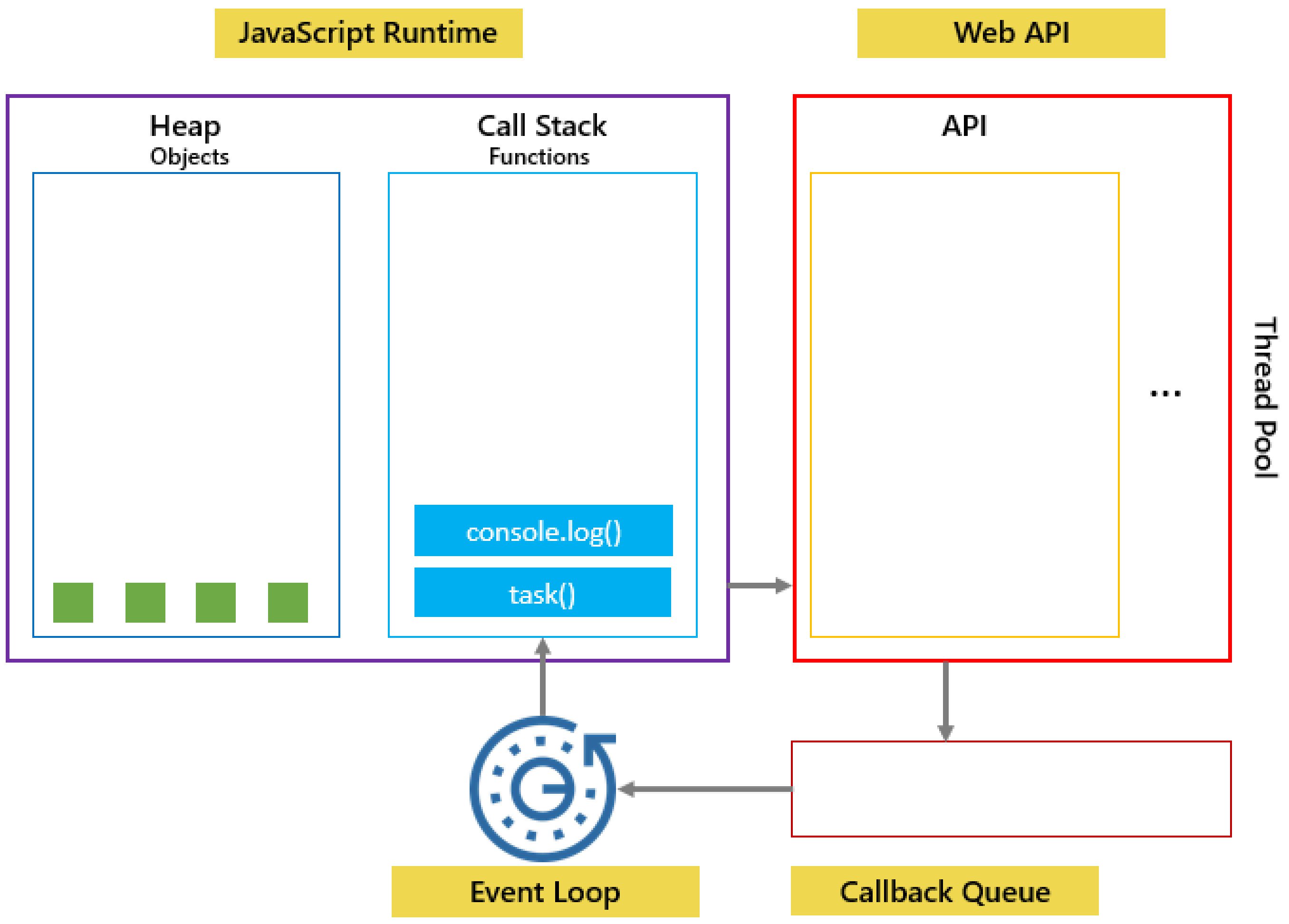
Second, after roughly 3 seconds, the timer expires, the `task` is pushed to the callback queue and waited for the next opportunity to execute.



Third, because the call stack is empty, the event loop removes the `task()` from the callback queue, places it on the call stack, and executes it:



Fourth, the `console.log()` in the `setTimeout()` executes that creates a new function execution context.



Finally, the `console.log()` and `task()` are popped out of the call stack once they are completed.

Summary

- `setTimeout()` is a method of the window object.
- `setTimeout()` sets a timer and executes a callback function when the timer expires.