Concurrency on the Web

JS Event Loop and Web Workers

Definitions

Concurrency *(n)* : Two or more events happening at the same time.

Thread *(n)* : Short for a **thread** of execution. **Threads** are a way for a program to divide itself into two or more simultaneously running tasks.

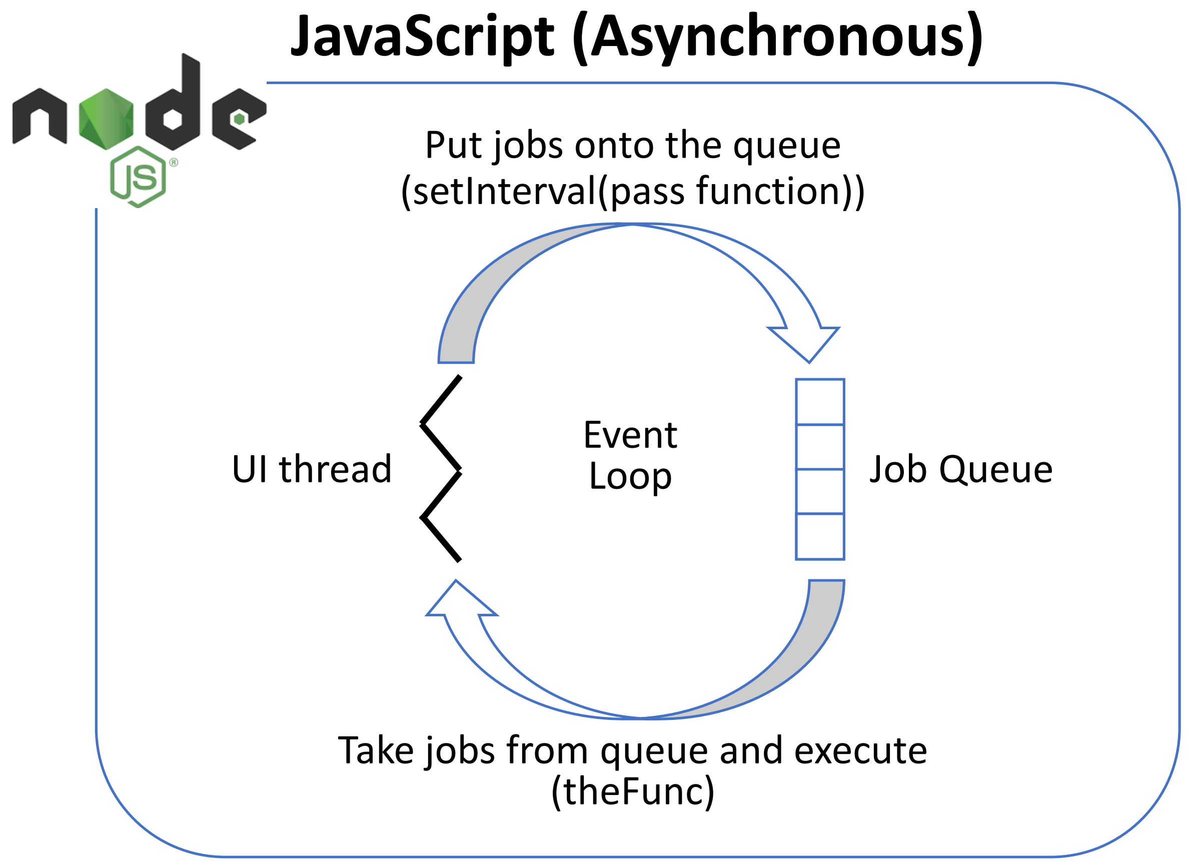
Asynchronous *(adj)* : When a specific operation begins upon receipt of an indication (signal) that the preceding operation has been completed.

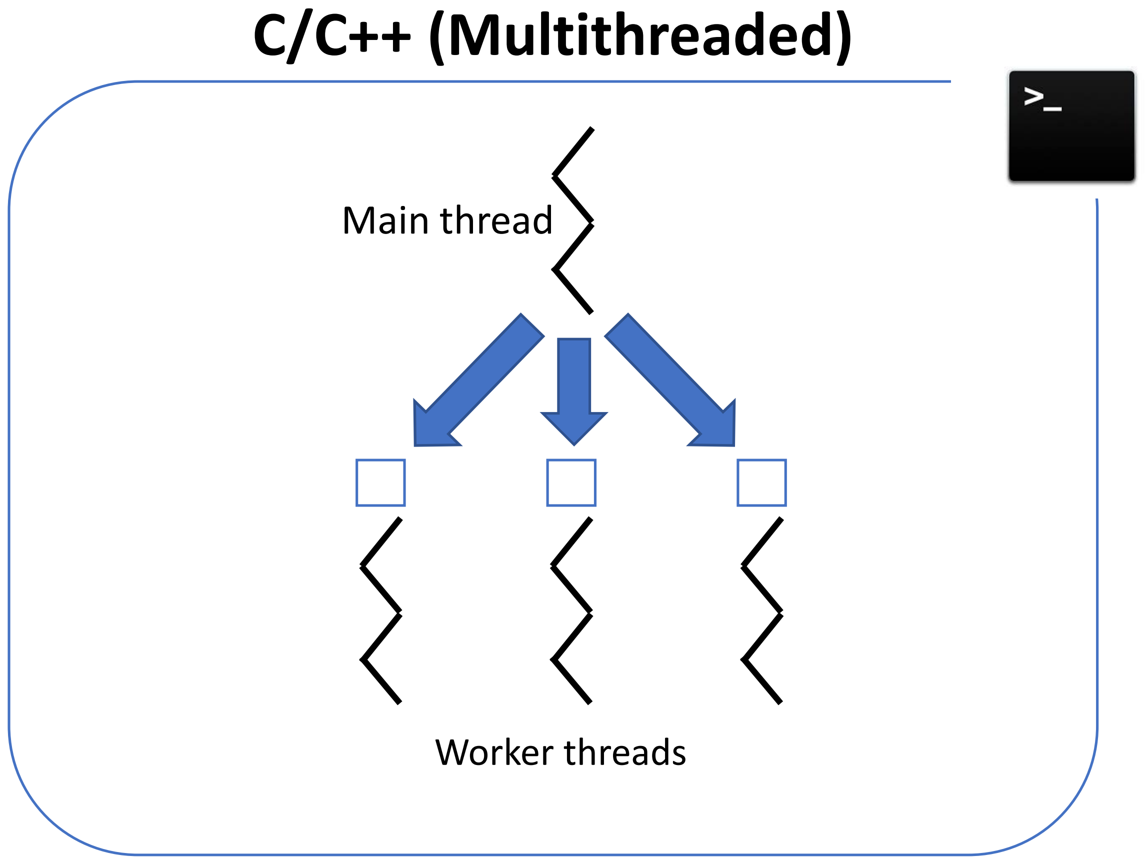
JavaScript is asynchronous, not multithreaded

Learning Objectives

- ✓ Understand asynchronous vs multithreaded execution
- Review JavaScript event loop code (asynchronous)
- ✓ Understand how to make JavaScript multithreaded by using Web Workers
- Review single thread vs Web Workers code (concurrent)

JavaScript is asynchronous, not multithreaded (for now)


JavaScript (Asynchronous)
Put jobs onto the queue
(setInterval(pass function))
UI thread
Event Loop
Job Queue
Take jobs from queue and execute
(theFunc)



Understand asynchronous vs multithreaded execution

The Event Loop enables asynchronous execution

Event Loop Steps

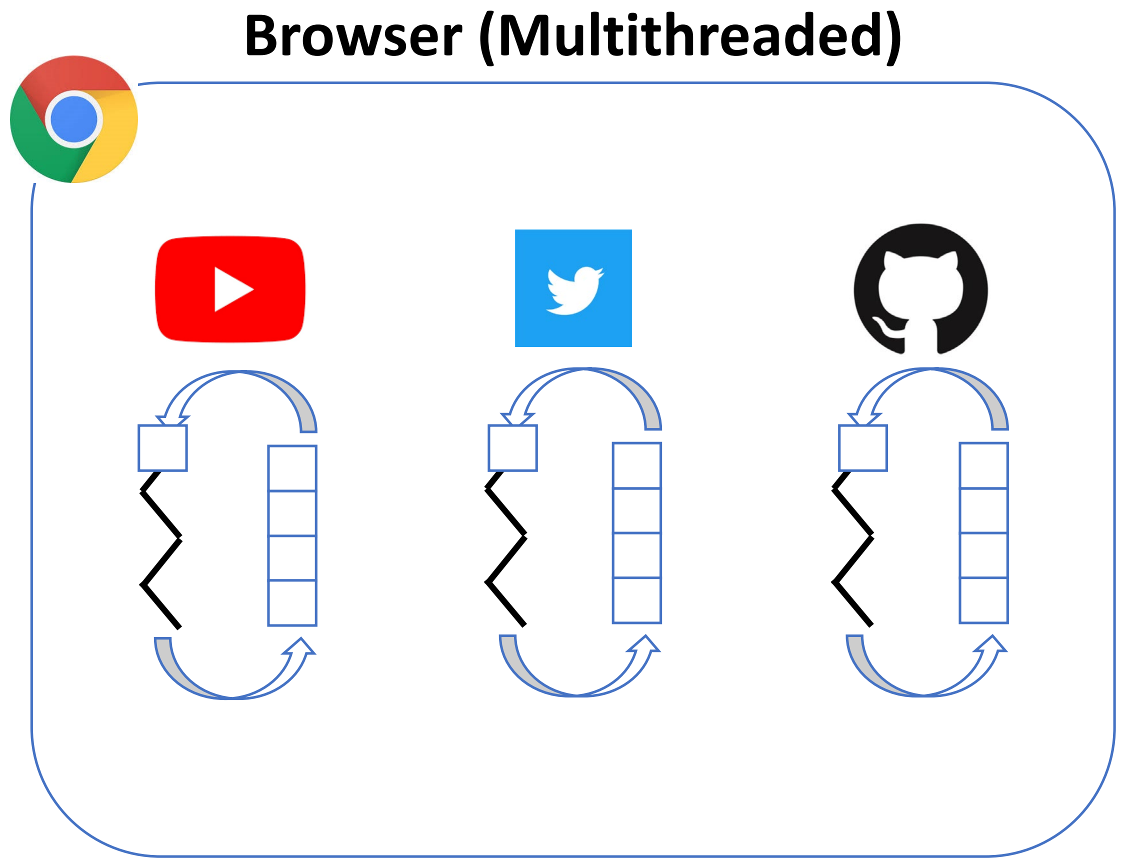
1. Pop call stack and execute function
2. If the function makes a call that must be handled later, put the new call on the queue
3. Try to pop the next call from the call stack. If the call stack is empty, take a function from the queue and push it to the call stack

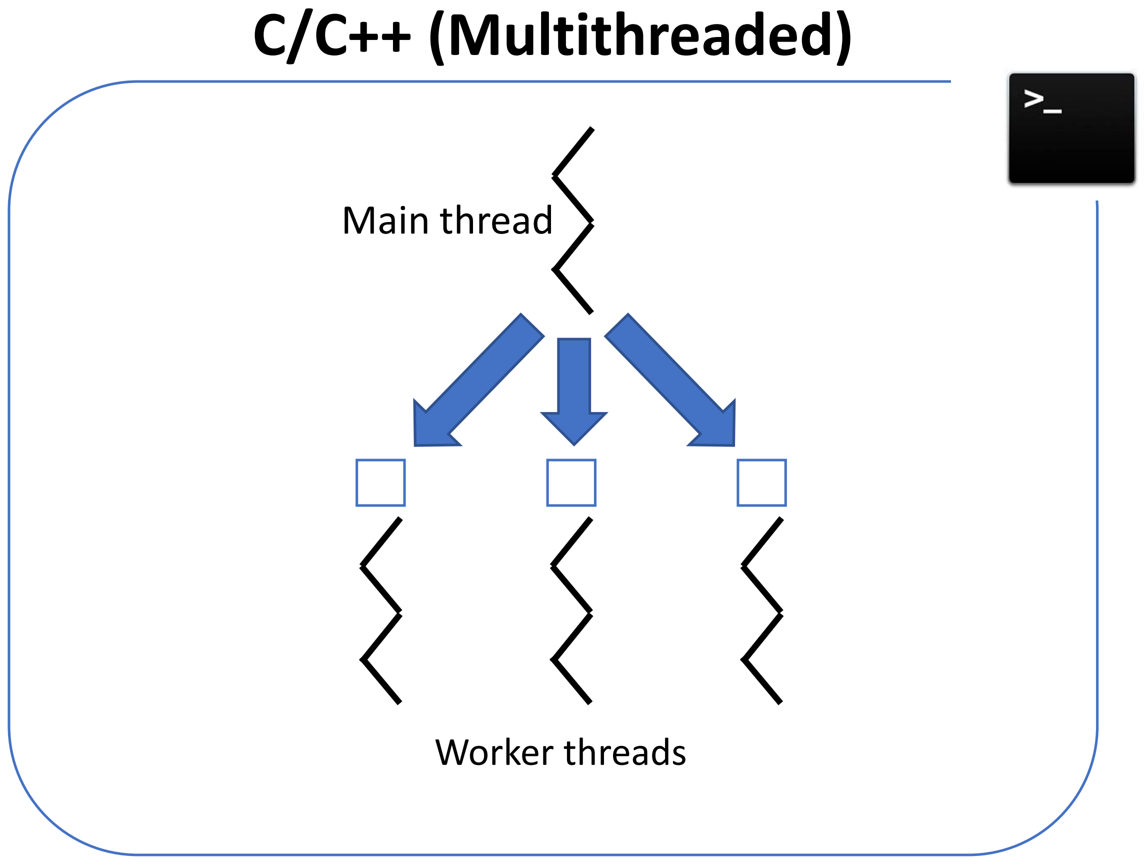
Data Structures

- Call Stack – First In, Last Out
  - Like stacking items
  - Pop() – get last call (top of the stack)
  - Push() – put call onto stack
- Queue – First In, First Out
  - Like putting items in a pipeline

Understand asynchronous vs multithreaded execution

JavaScript is single-threaded, and the browser is multithreaded





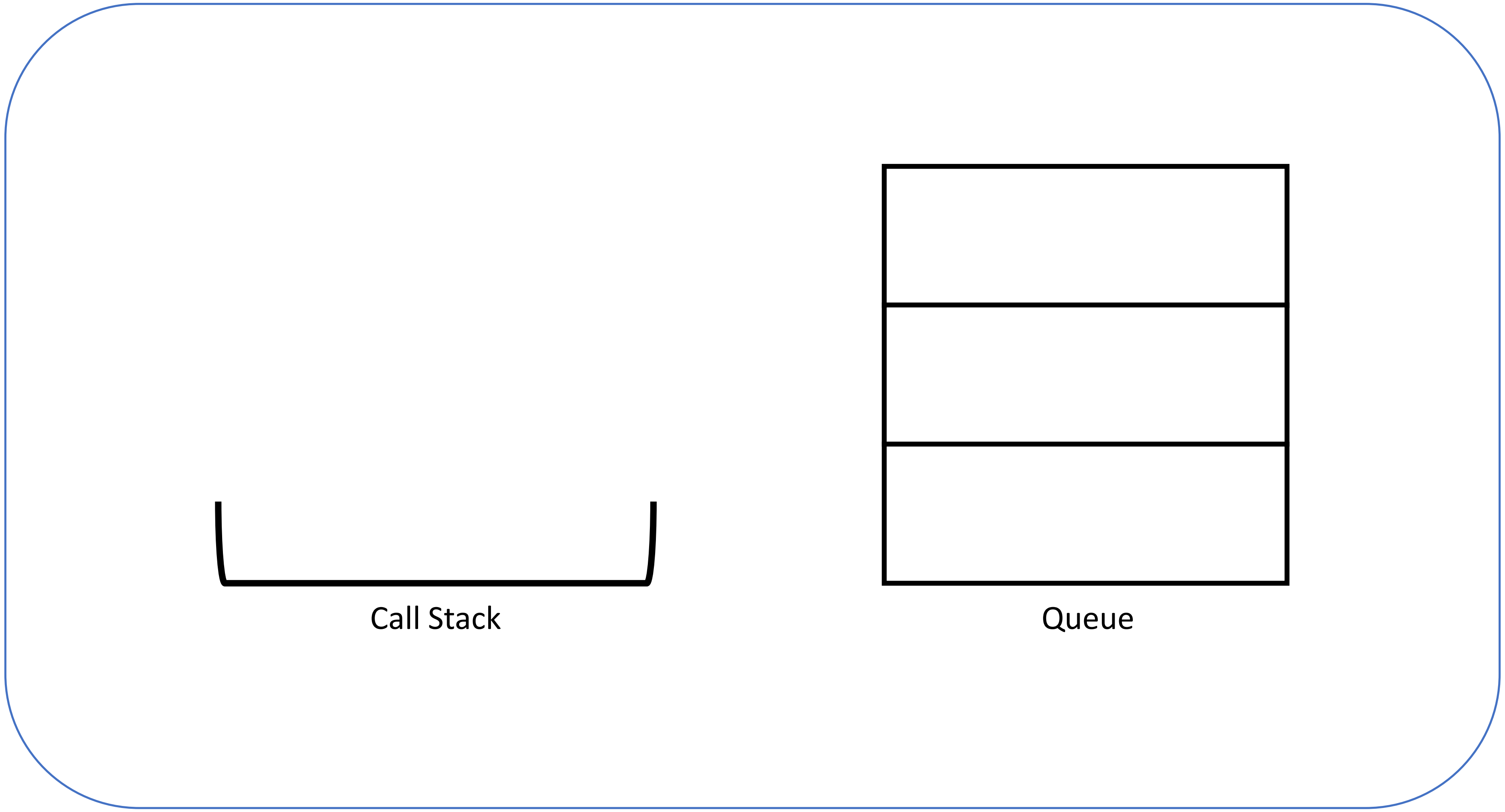
Understand asynchronous vs multithreaded execution

Event Loop code

```
const first  = () => console.log('The first function')
const second = () => console.log('The second function')
const third  = () => console.log('The third function')

first()

setTimeout(second, 0) // goes onto queue

third()
```

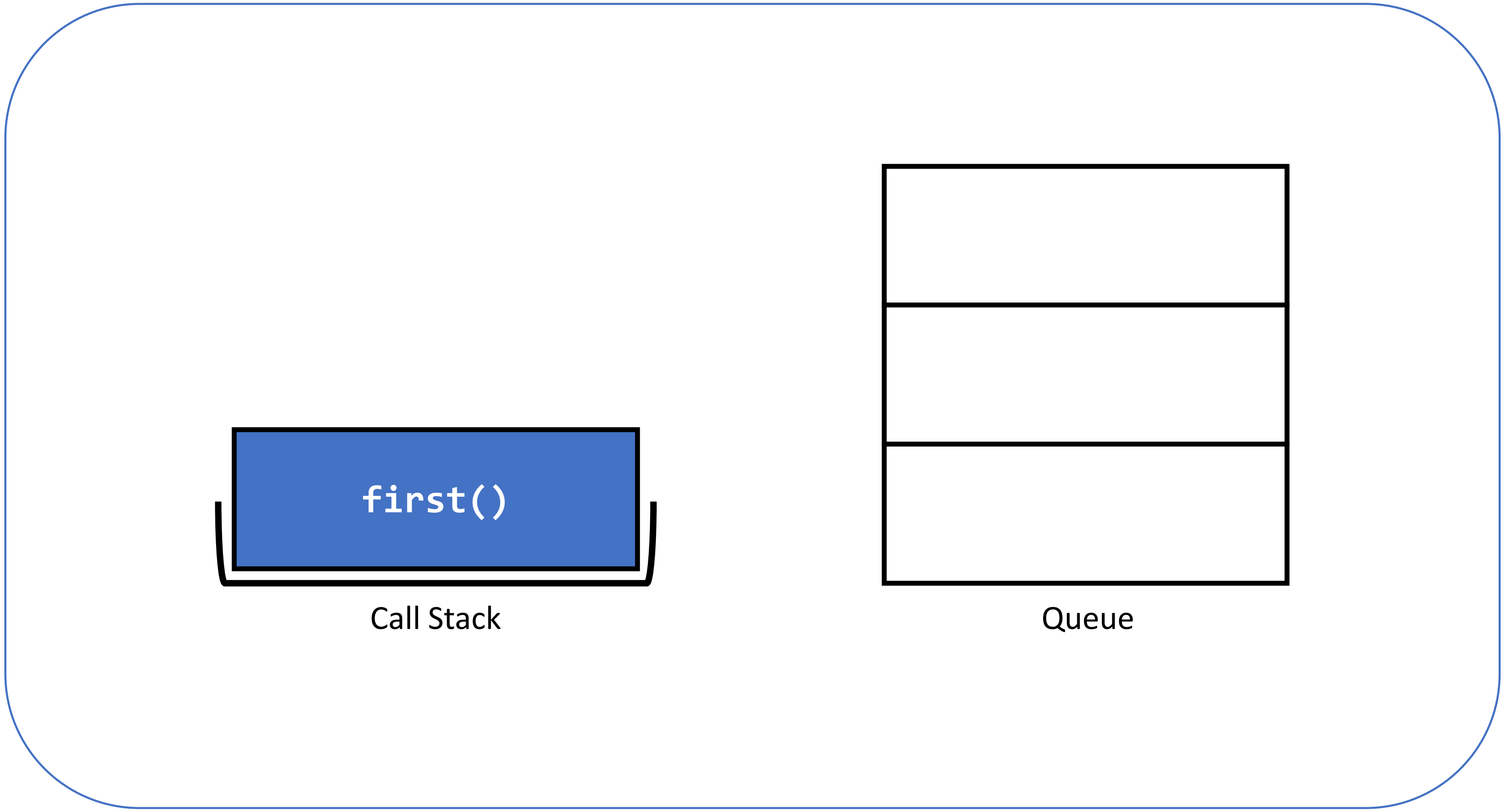
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



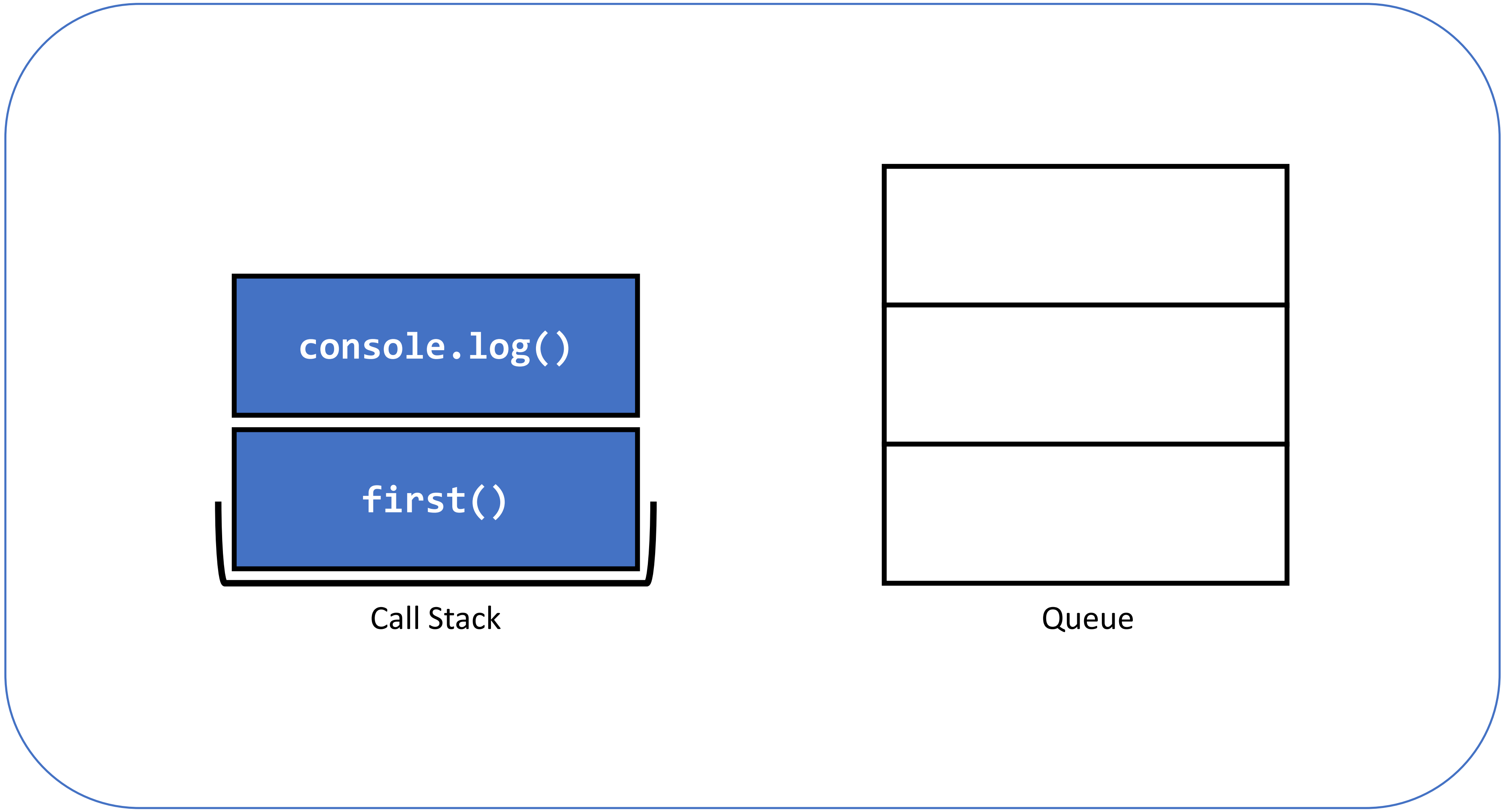
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



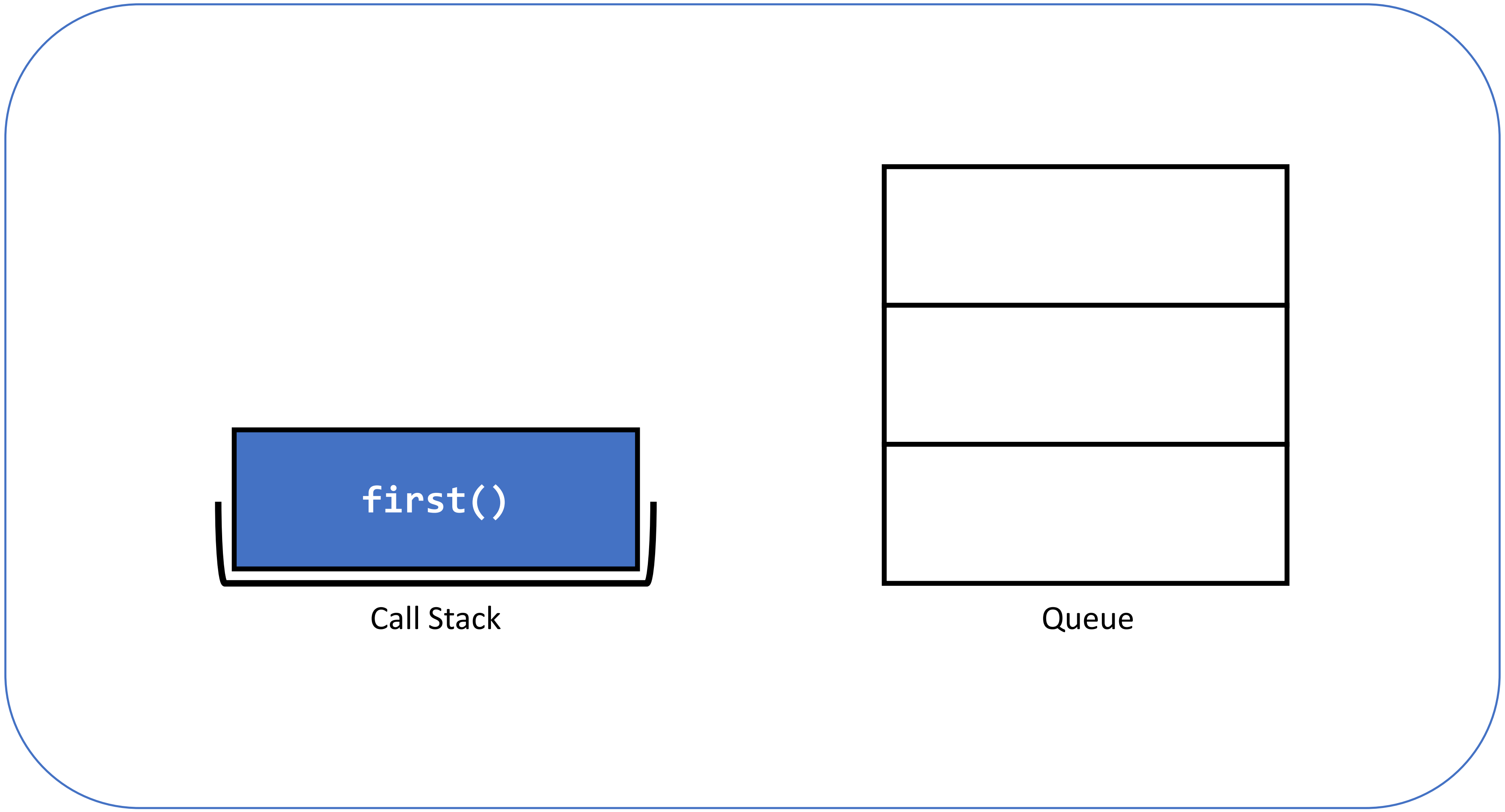
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



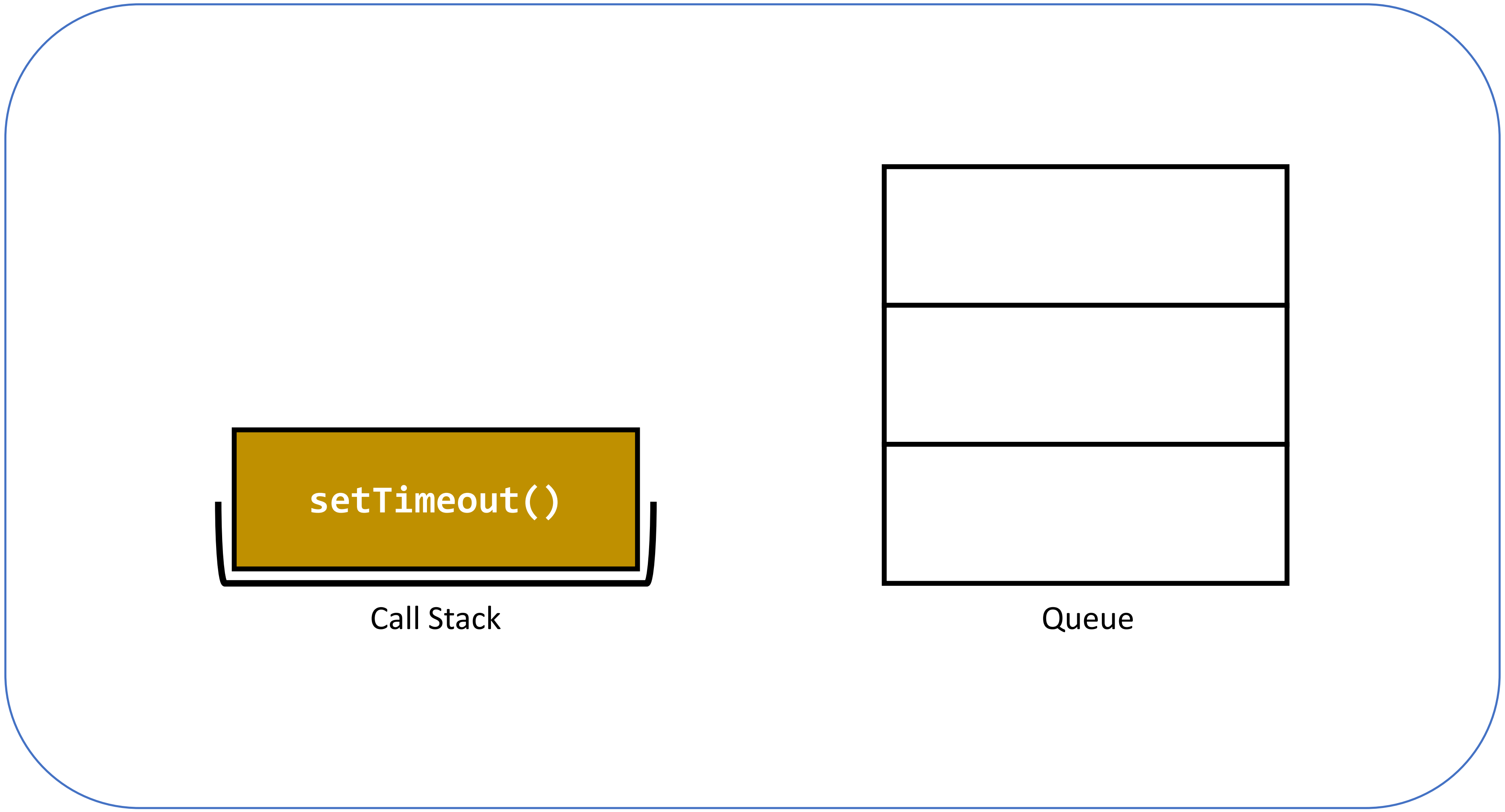
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



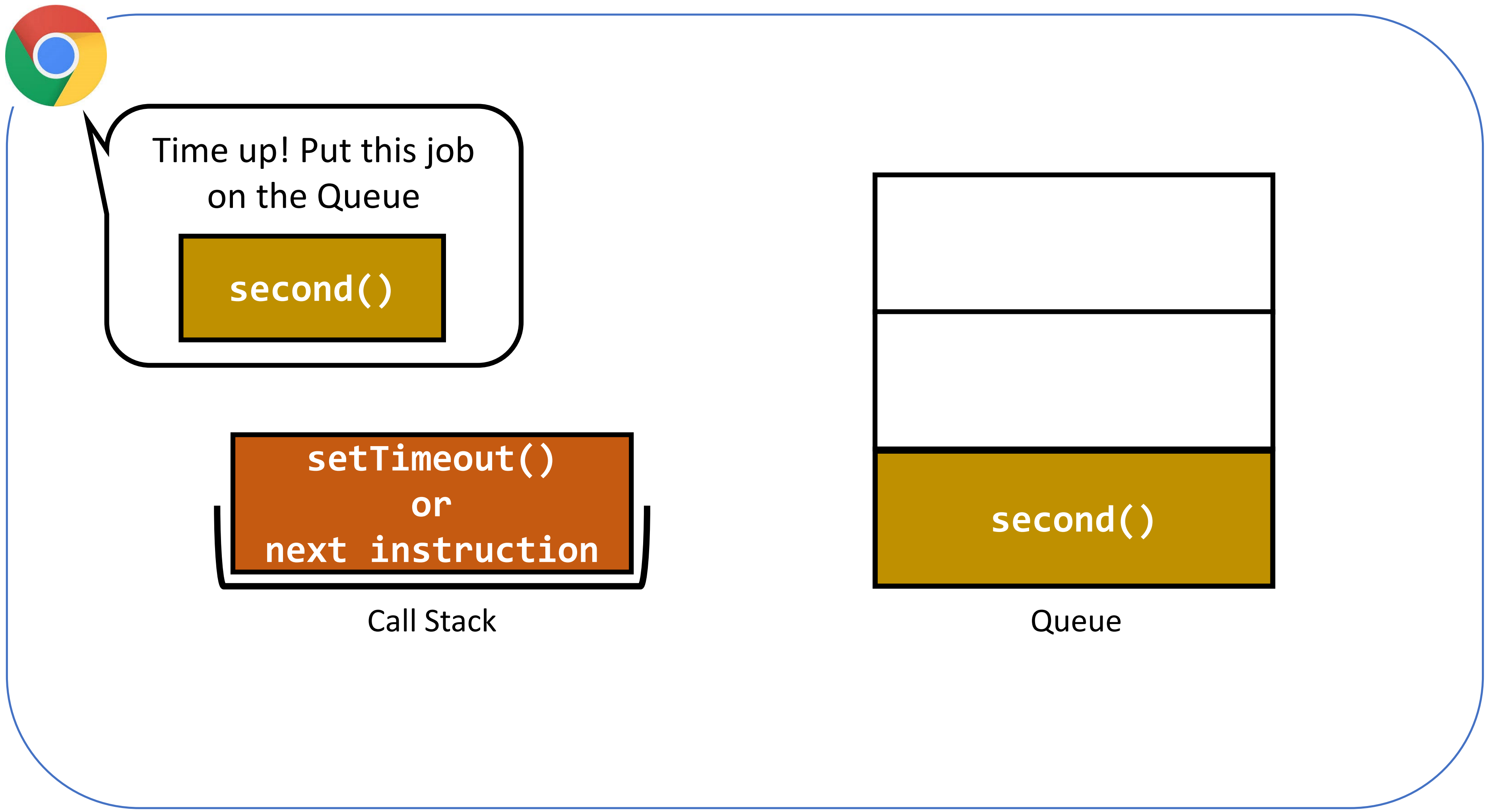
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



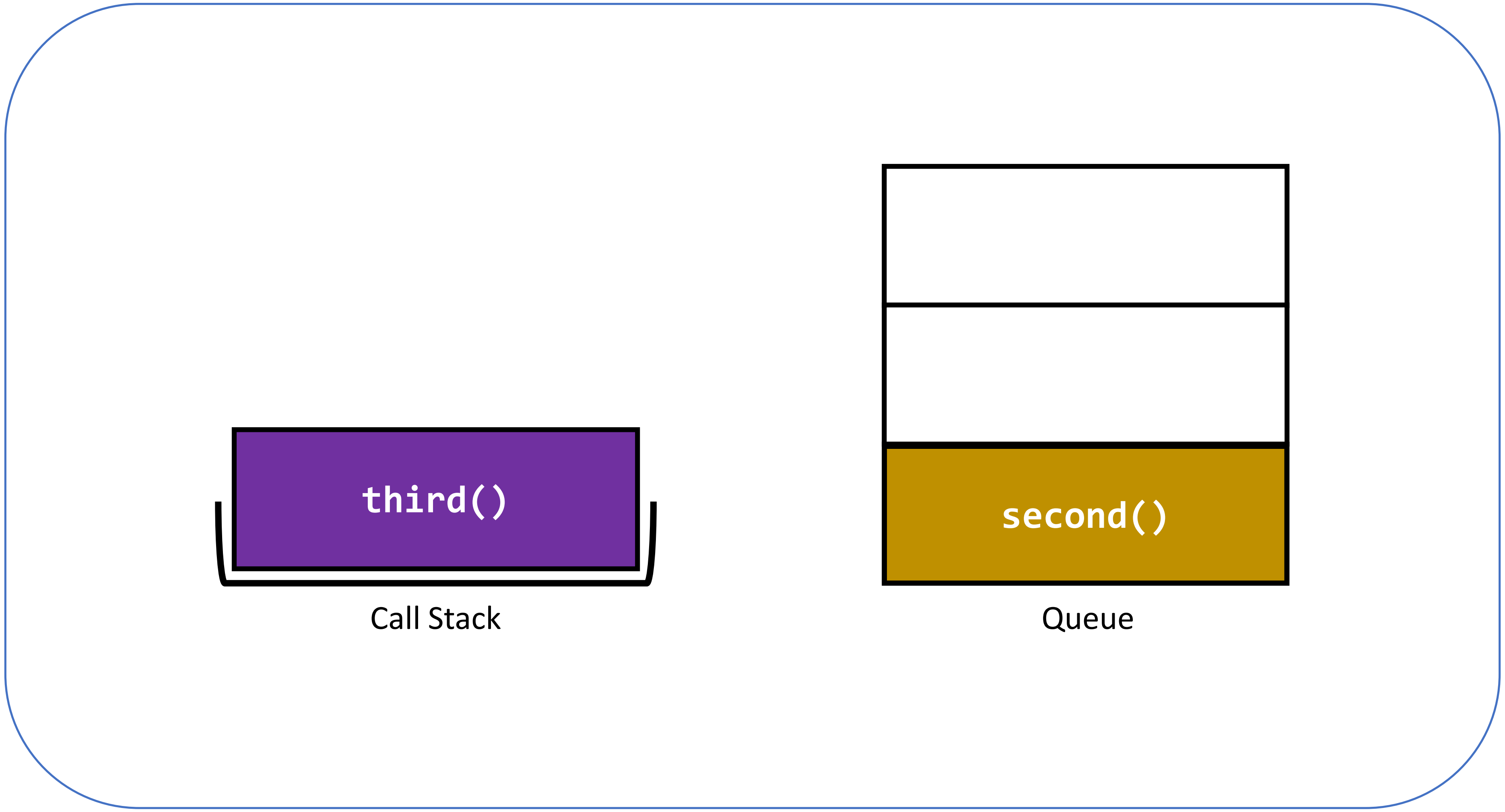
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



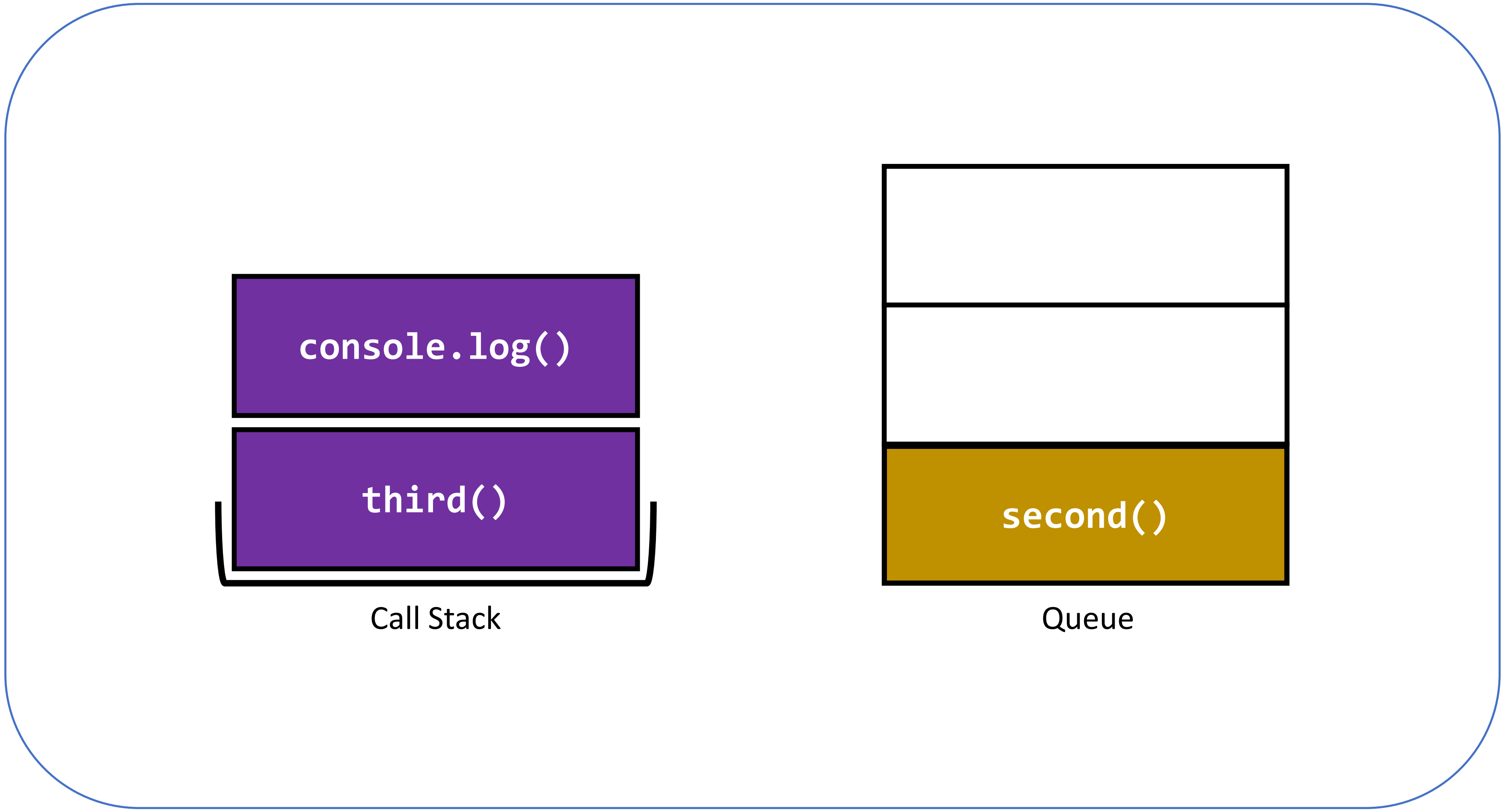
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



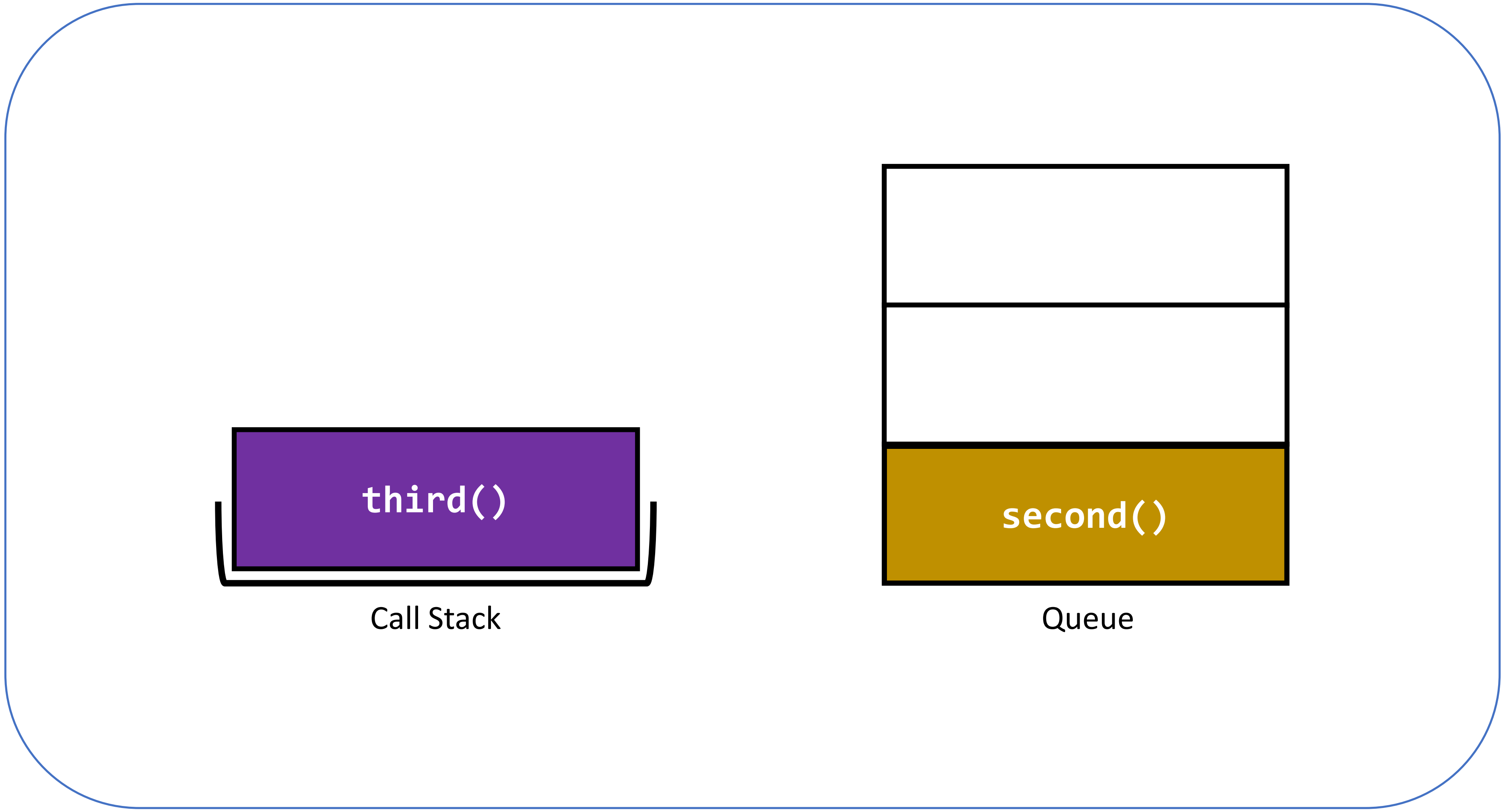
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



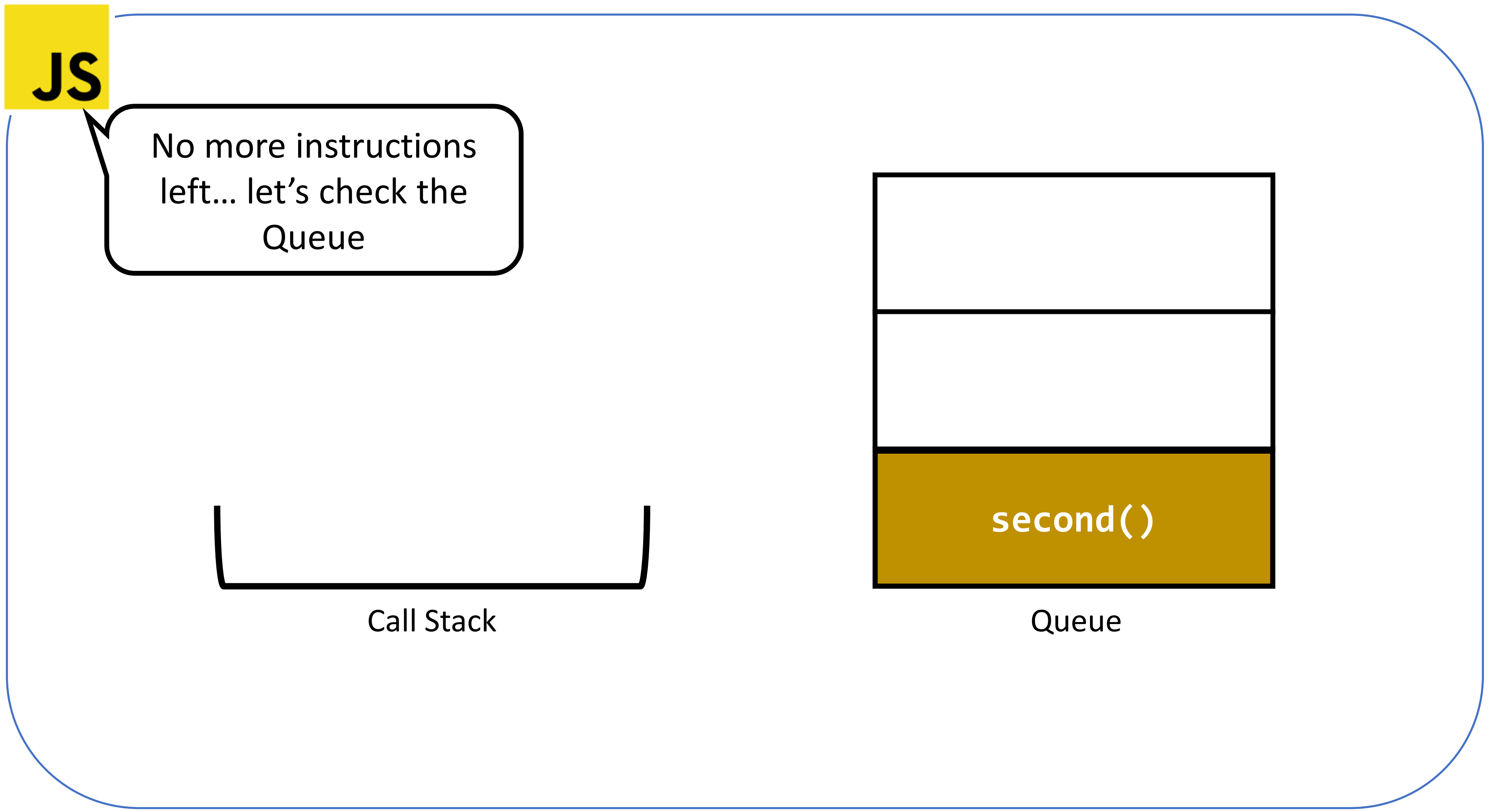
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



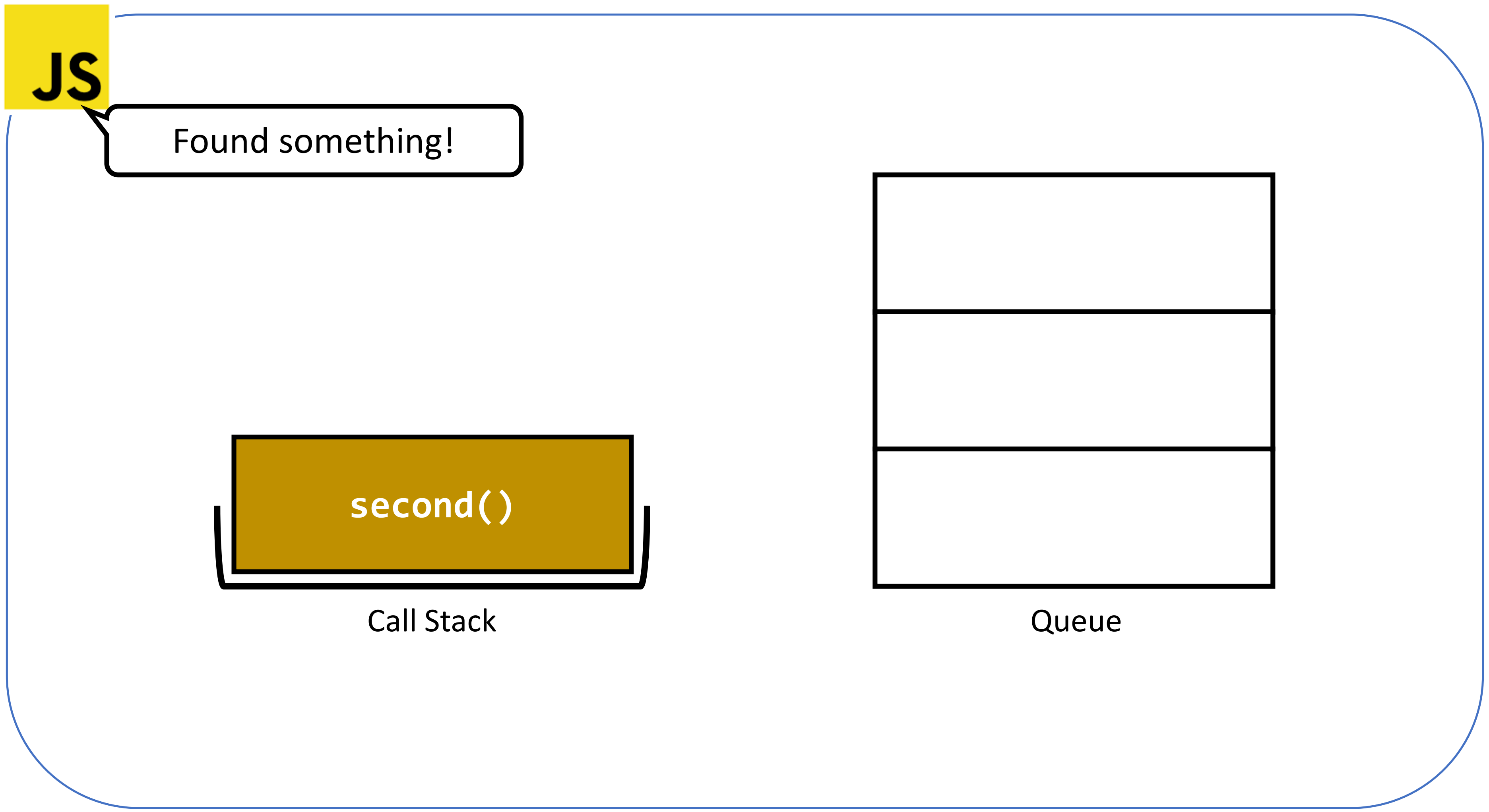
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



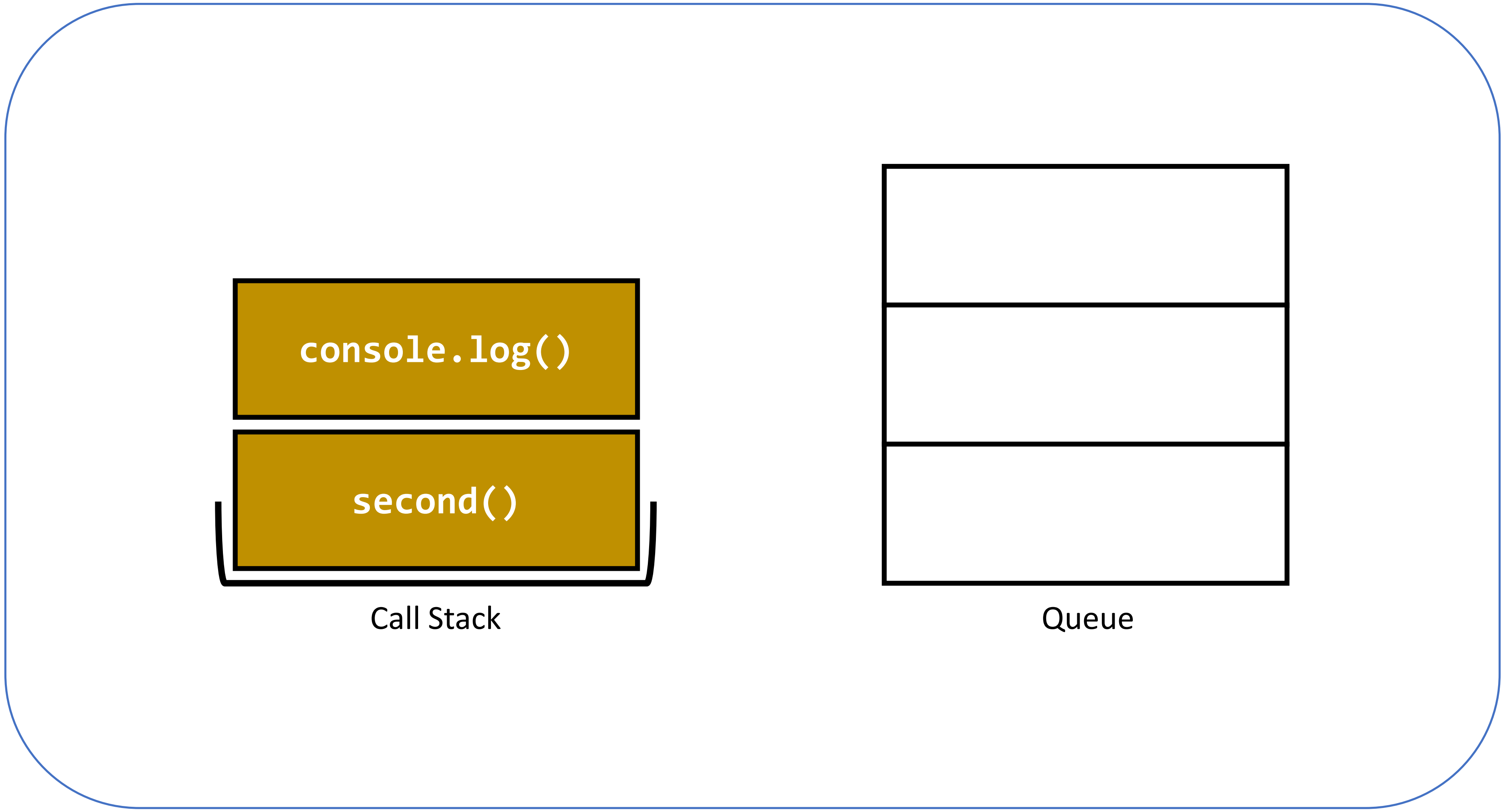
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



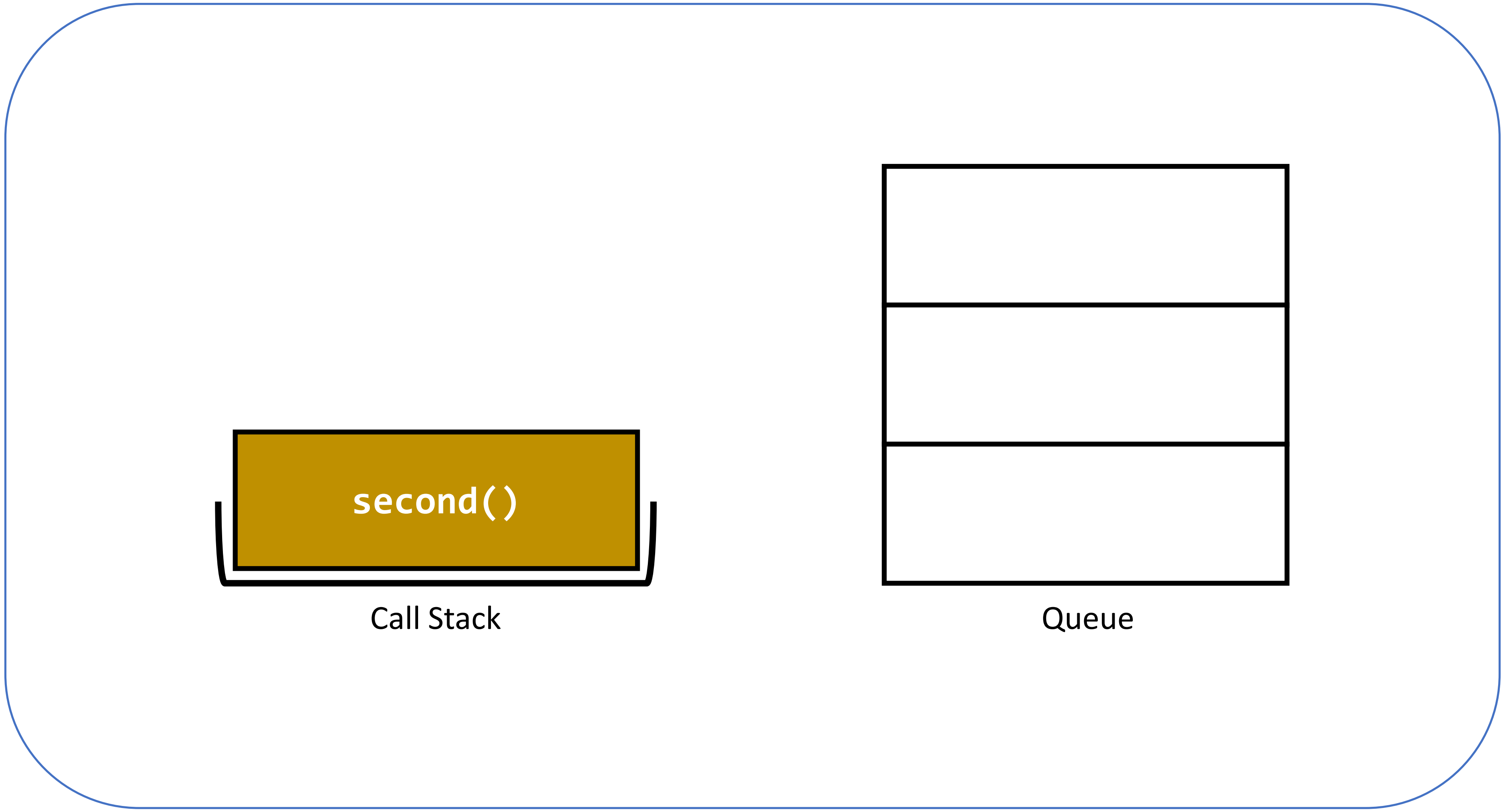
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



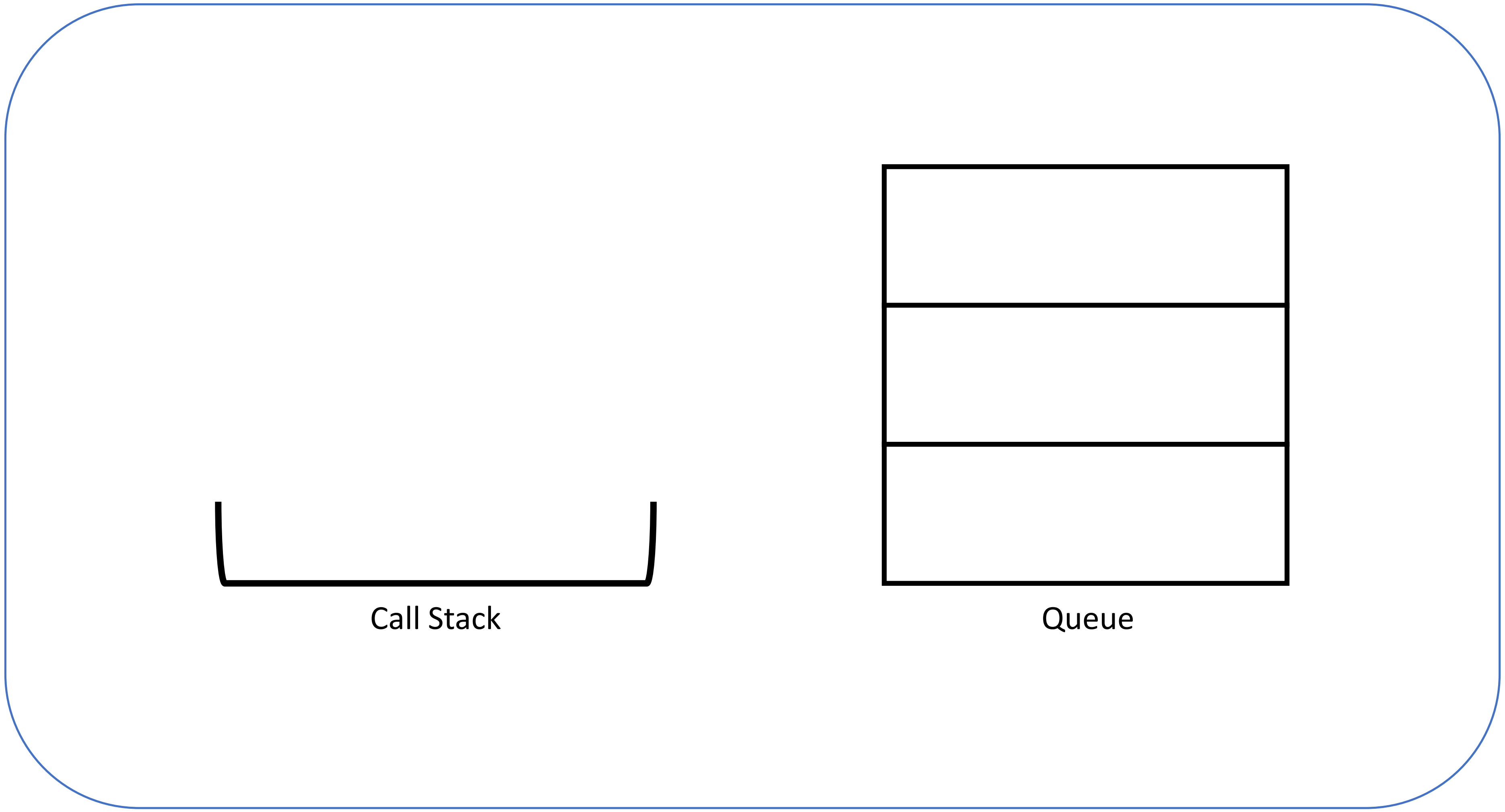
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



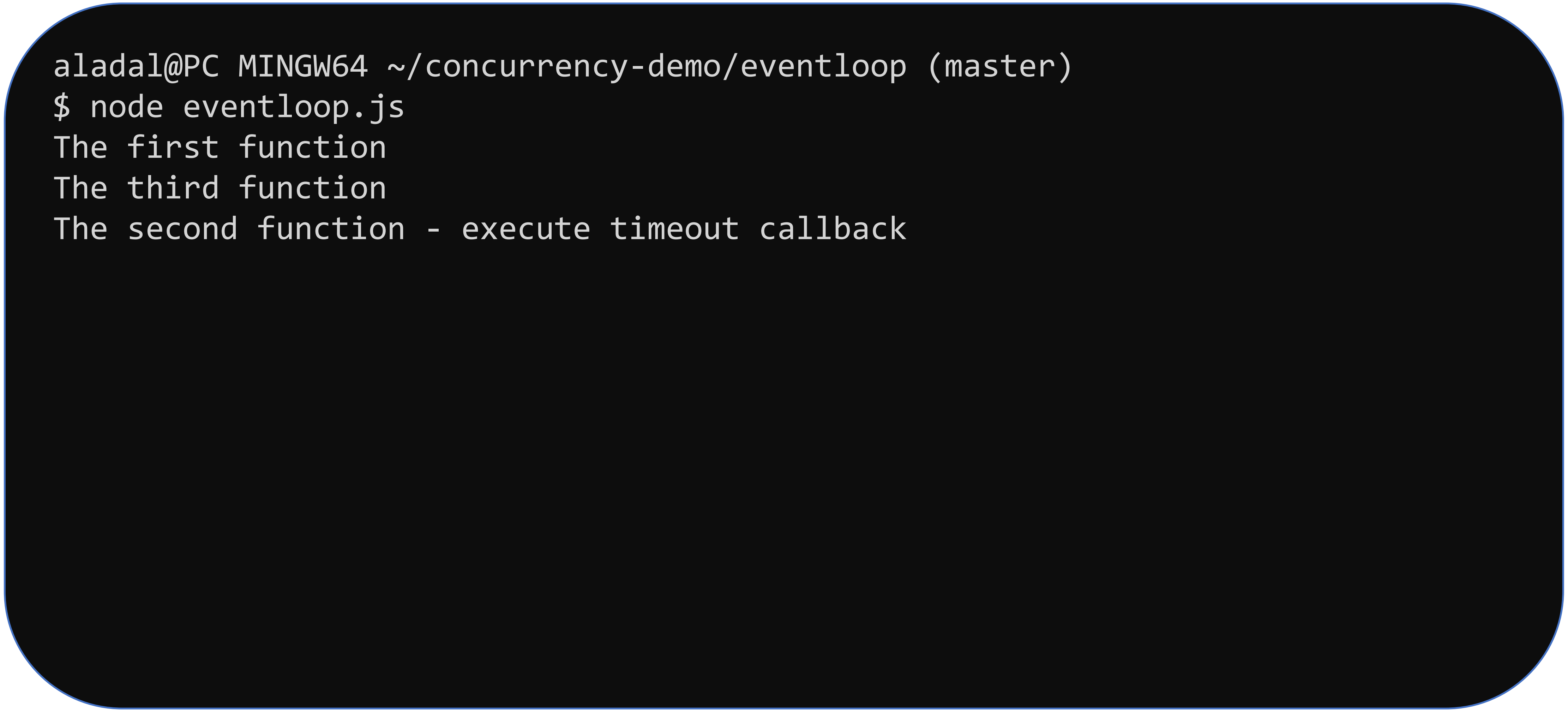
Review JavaScript event loop code (asynchronous)

Event Loop model – Stack & Queue



Review JavaScript event loop code (asynchronous)

Event Loop result



```
aladal@PC MINGW64 ~/concurrency-demo/eventloop (master)
$ node eventloop.js
The first function
The third function
The second function - execute timeout callback
```

Review JavaScript event loop code (asynchronous)

Event Loop takeaways

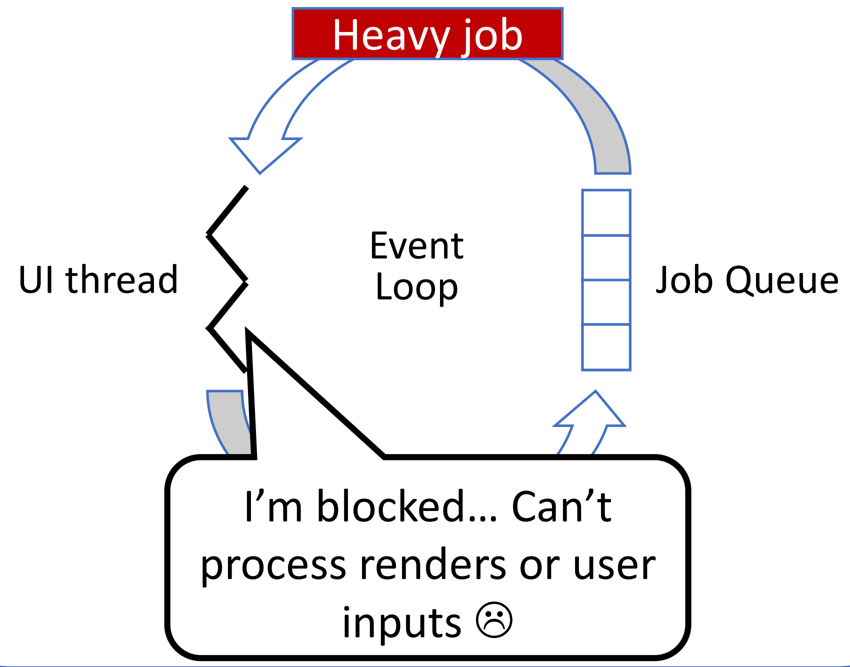
- The Event Loop enables JavaScript to execute functions in an interleaved way. This is how JS works on multiple things seemingly at the same time.

- For more info on the JS Event Loop:
- https://flaviocopes.com/javascript-event-loop/

Review JavaScript event loop code (asynchronous)

UI thread dangers

JavaScript Event Loop



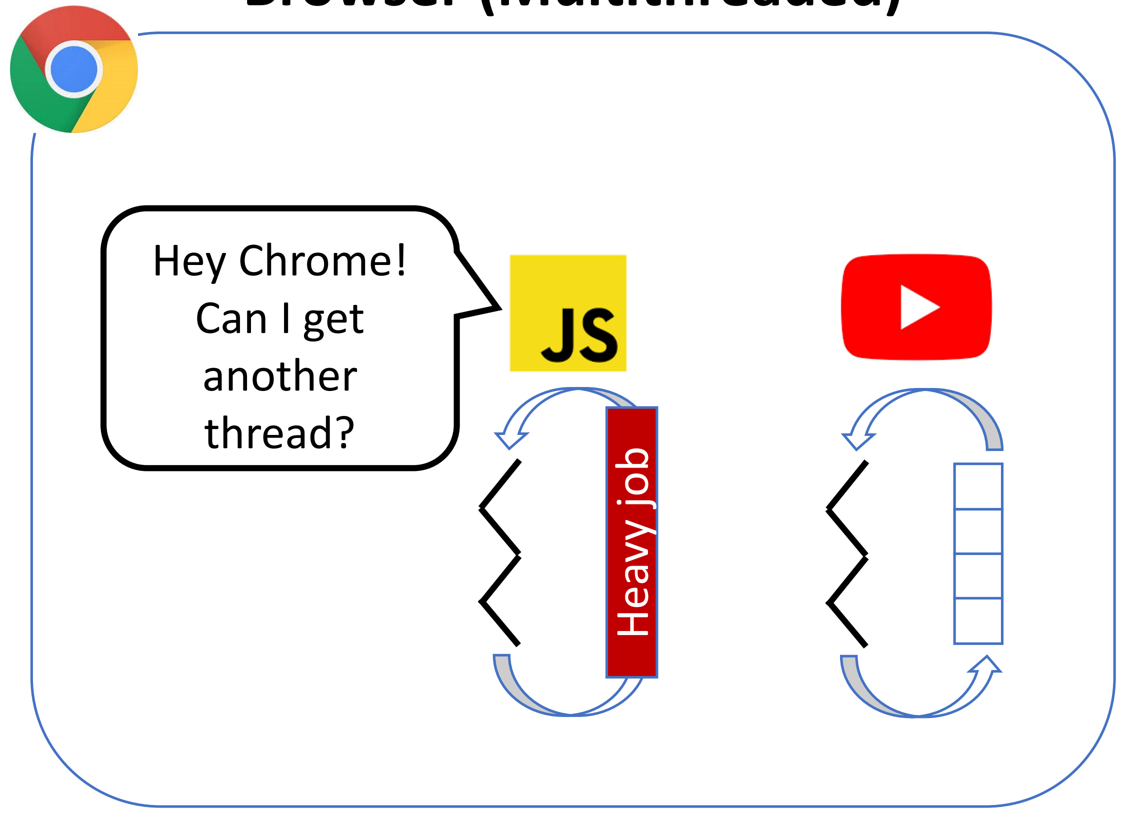
What if there is a very heavy job?

- Any task that is computationally intensive can block the UI thread
  - Sorting through or processing lots of data
  - Displaying lots of images
  - Handling large files
- The UI thread is responsible for rendering, user input, and executing all function calls so if there's a chance that it could be blocked we should avoid it

Understand how to make JavaScript multithreaded by using Web Workers

Solution: Ask the Browser for another thread

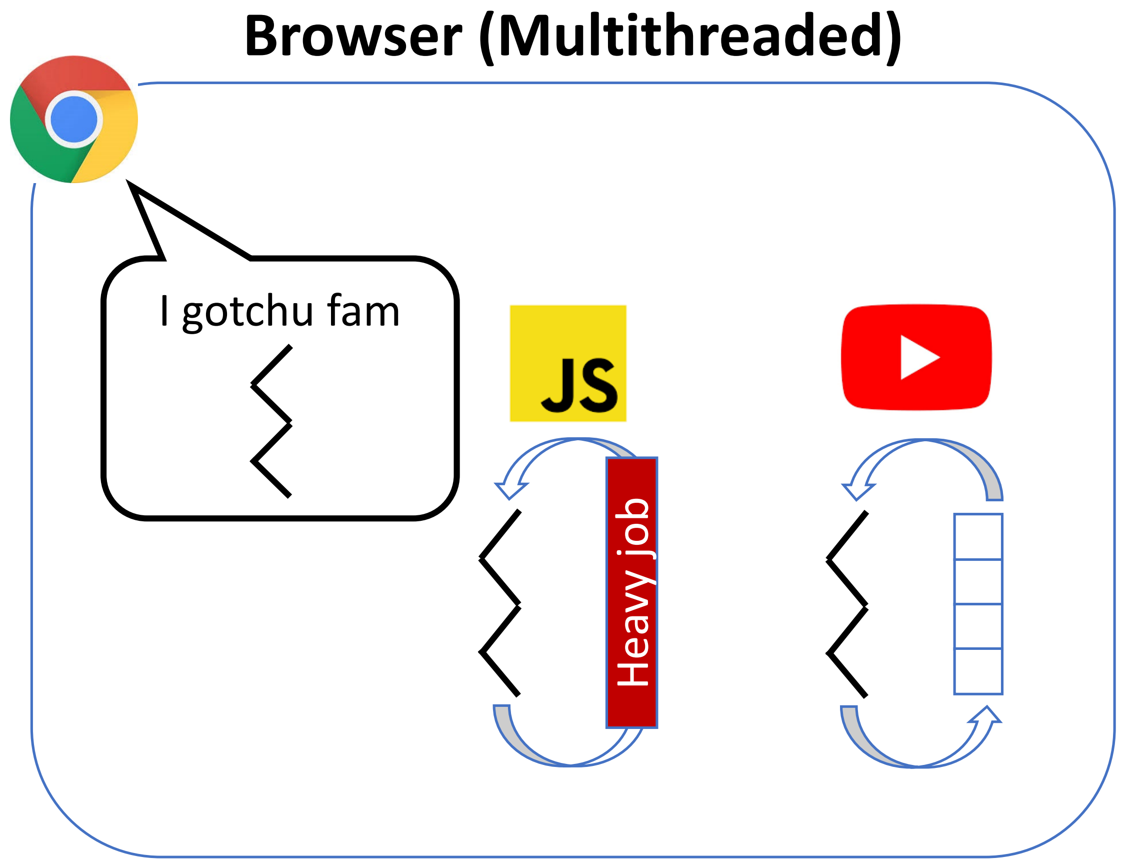
Browser (Multithreaded)



- The browser is a multithreaded program. It can request an additional thread from the computer if your code needs it

Understand how to make JavaScript multithreaded by using Web Workers

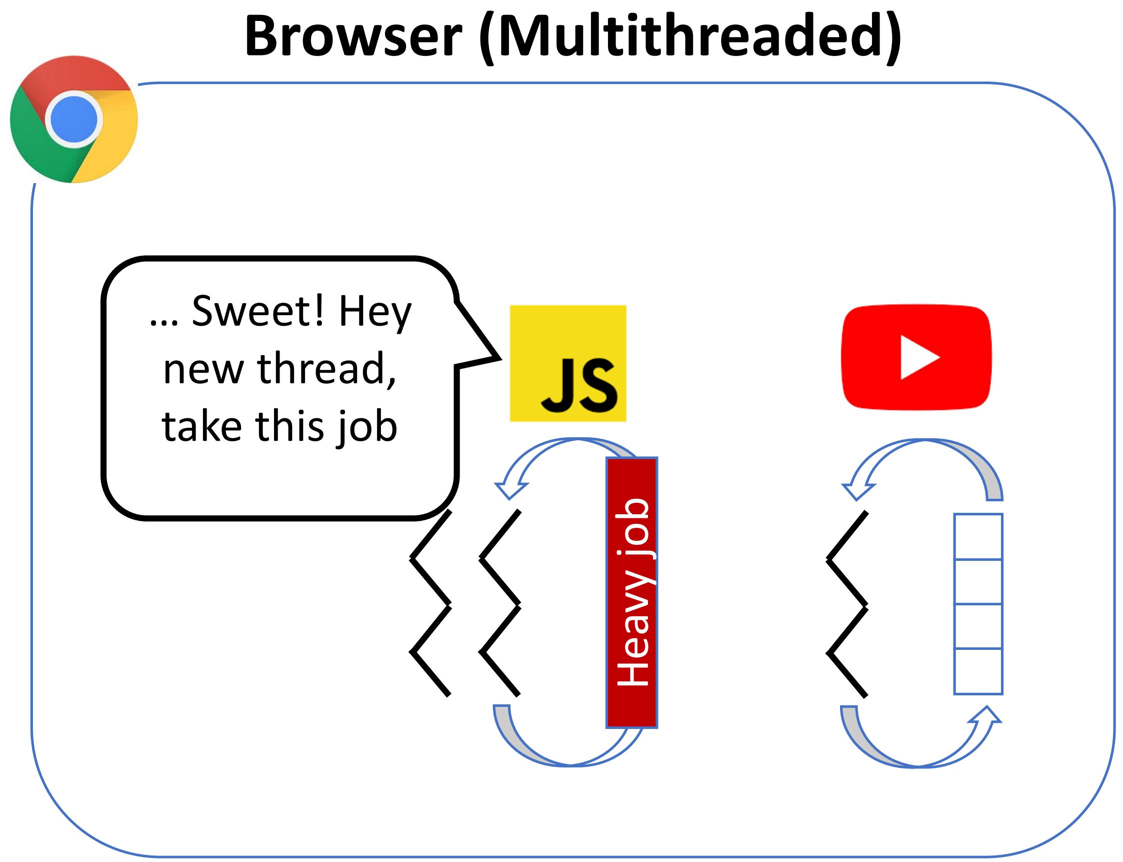
Solution: Ask the Browser for another thread



- The browser is a multithreaded program. It can request an additional thread from the computer if your code needs it

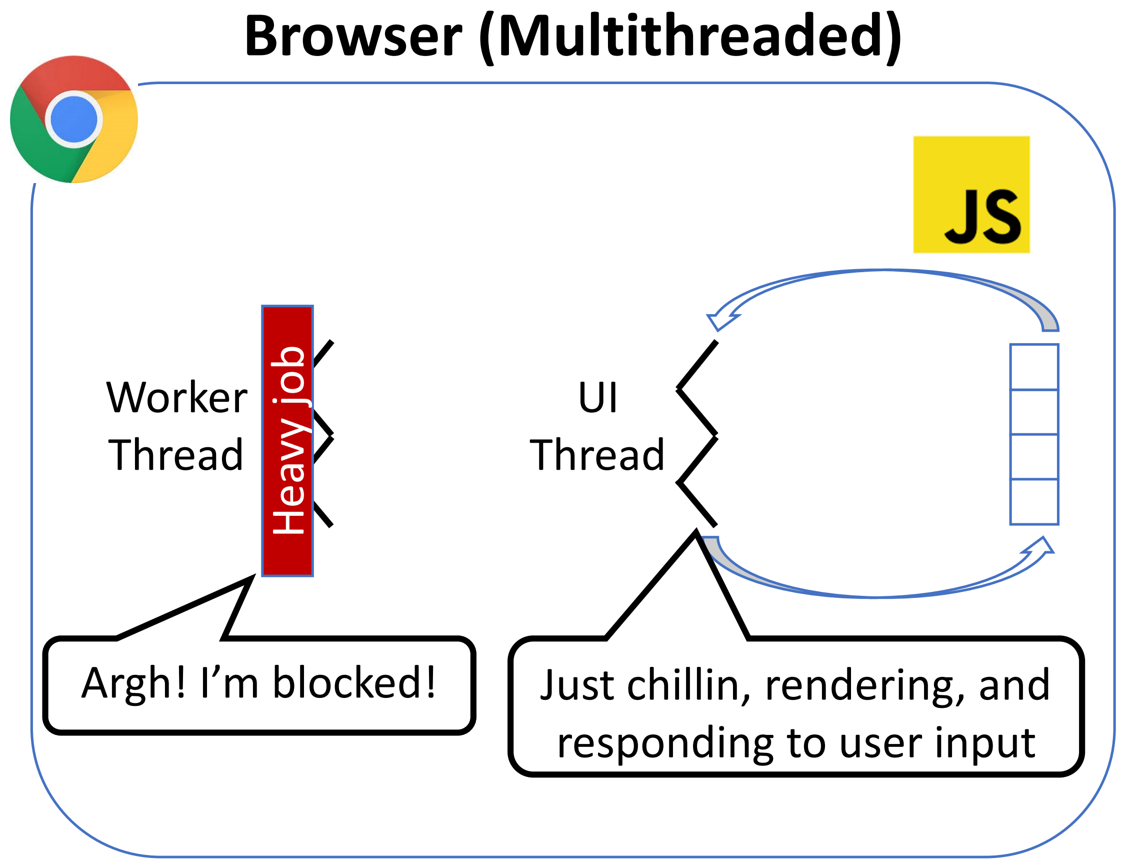
Understand how to make JavaScript multithreaded by using Web Workers

Solution: Ask the Browser for another thread



- Once your code receives the additional thread (Web Worker), you can use message passing to activate the worker

Solution: Ask the Browser for another thread



- Heavy calculations will block the worker thread, but your UI thread remains free and the user doesn't see any rendering hiccups

Understand how to make JavaScript multithreaded by using Web Workers

Web Worker: Main/UI Thread

```
var worker = new Worker('worker.js')

worker.addEventListener('message', function(e) {
    if (e.data) {
        this.postMessage({ workload: workload.pop() })
    }
}, false)

worker.postMessage({ workload: workload.pop() })
```

Review code that uses a single thread vs Web Workers

Quick Detour: postMessage API

- “The window.postMessage() method safely enables cross-origin communication between Window objects; e.g., between a page and a pop-up that it spawned, or between a page and an iframe embedded within it.” --- Mozilla Developer Network

- Many use cases such as Web Sockets, Web Workers, cross-origin communication

- Security is a big deal – if we don’t specify the targetOrigin and validate what is in the message, anyone can send a message to our listening Web Worker

Review code that uses a single thread vs Web Workers

Quick Detour: postMessage API

- Send data: targetWindow.postMessage(message, targetOrigin, [transfer]);
  - Message parameter must be something that is eligible for structured cloning. It must translate into valid JSON.
  - Any JSON object is ok, but functions and maps are not.

- Listen for data: target.addEventListener(type, listener [, options]);

- For more info on the postMessage API & structured cloning/deep copying:
- https://developer.mozilla.org/en-US/docs/Web/API/Window/postMessage
- https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener
- https://dassur.ma/things/deep-copy/

Review code that uses a single thread vs Web Workers

Web Worker: Main/UI Thread

```
var worker = new Worker('worker.js')

worker.addEventListener('message', function(e) {
    if (e.data) {
        this.postMessage({ workload: workload.pop() })
    }
}, false)

worker.postMessage({ workload: workload.pop() })
```

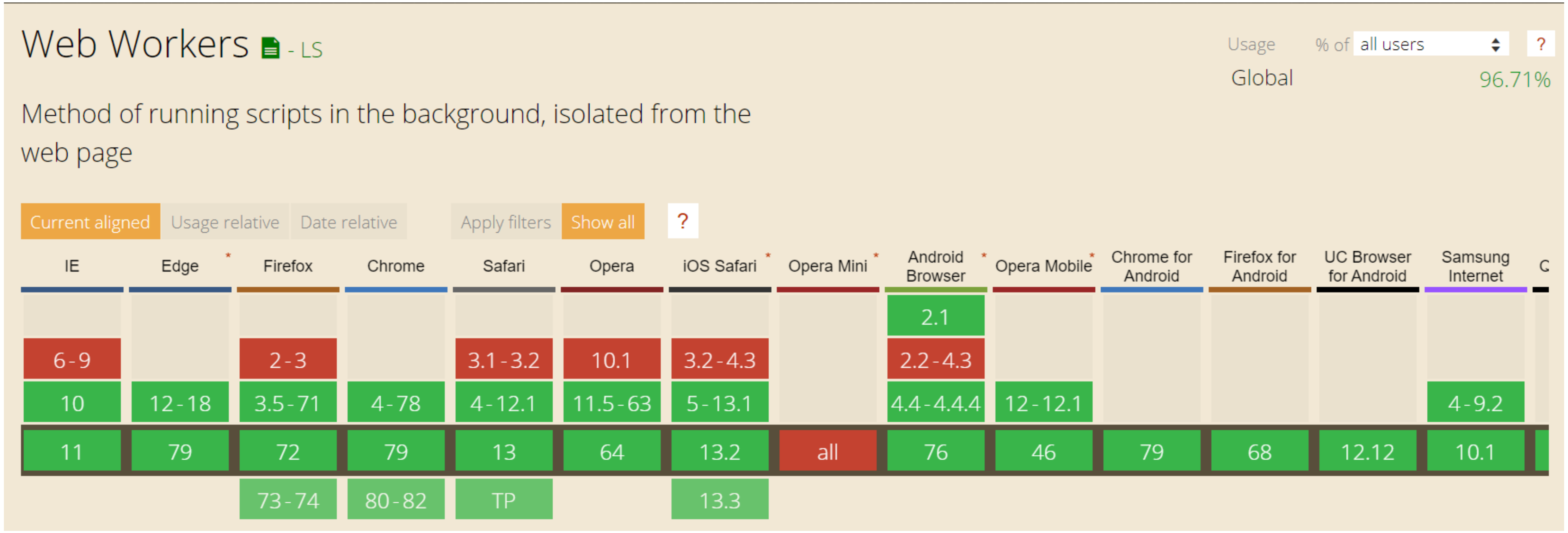
Review code that uses a single thread vs Web Workers

Web Worker: Worker Thread

```
self.addEventListener('message', function(e) {
    if (e.data) {
        // process the workload
        this.self.postMessage({ ready: true })
    }
}, false)
```

Review code that uses a single thread vs Web Workers

Can I Use This?



Web Workers - LS

Usage % of all users ?

Global 96.71%

Method of running scripts in the background, isolated from the web page

Current aligned | Usage relative | Date relative | Apply filters | Show all | ?

IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *	Opera Mini *	Android Browser *	Opera Mobile *	Chrome for Android	Firefox for Android	UC Browser for Android	Samsung Internet	C
								🟩 2.1						
🟥 6-9		🟥 2-3		🟥 3.1-3.2	🟥 10.1	🟥 3.2-4.3		🟥 2.2-4.3						
🟩 10	🟩 12-18	🟩 3.5-71	🟩 4-78	🟩 4-12.1	🟩 11.5-63	🟩 5-13.1		🟩 4.4-4.4.4	🟩 12-12.1				🟩 4-9.2	
🟩 11	🟩 79	🟩 72	🟩 79	🟩 13	🟩 64	🟩 13.2	🟥 all	🟩 76	🟩 46	🟩 79	🟩 68	🟩 12.12	🟩 10.1	🟩
		🟩 73-74	🟩 80-82	🟩 TP		🟩 13.3								

Review code that uses a single thread vs Web Workers

Web Worker takeaways

- Any task that is computationally intensive can block the main thread, and we should use web workers to avoid this blocking. The following examples are use cases for web workers:

  - **Filtering –** Filter large data sets without blocking the UI thread and without making a full round trip to the server.
  - **Proxy for other Js library APIs –** You can use web workers to load and run Javascript libraries in separate threads so that none of the downloading or parsing is handled on the main thread

- Using Web Workers in React:
- https://www.fullstackreact.com/articles/introduction-to-web-workers-with-react/

- Using Web Workers in Angular:
- https://angular.io/guide/web-worker

Review code that uses a single thread vs Web Workers