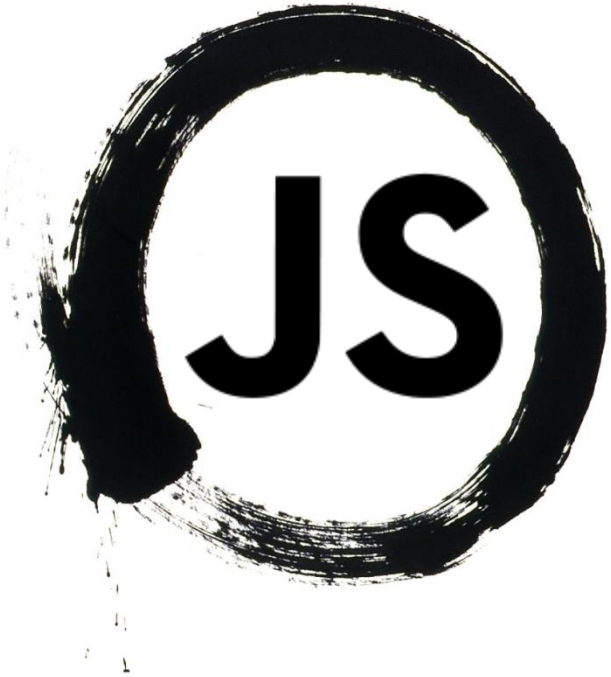
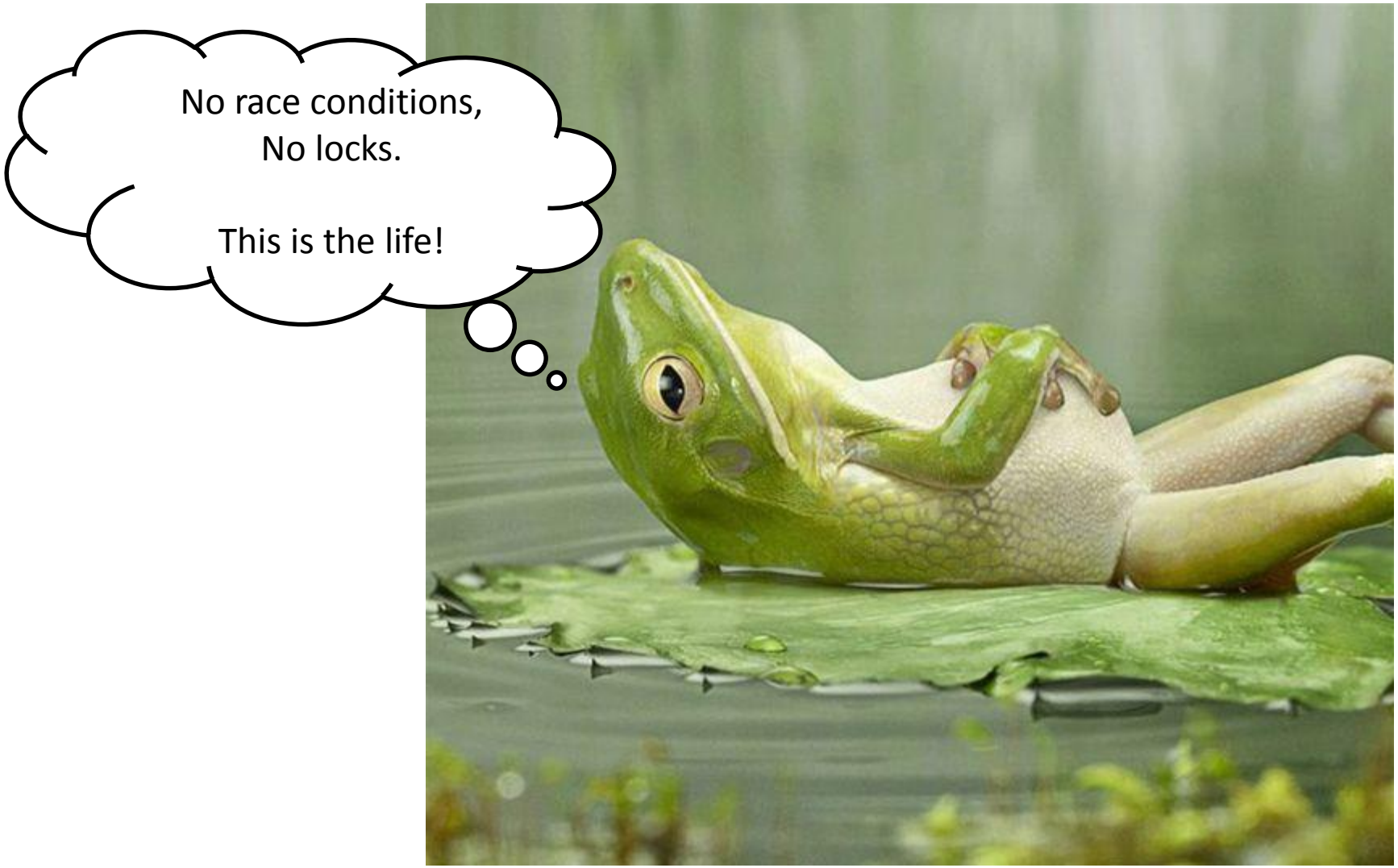
Loop into the Javascript Event Loop



Yonatan Mevorach

blog.cowchimp.com
github.com/cowchimp
@cowchimp

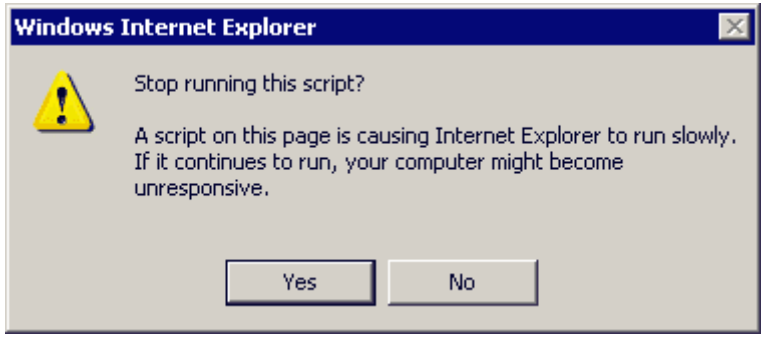
Javascript is single-threaded



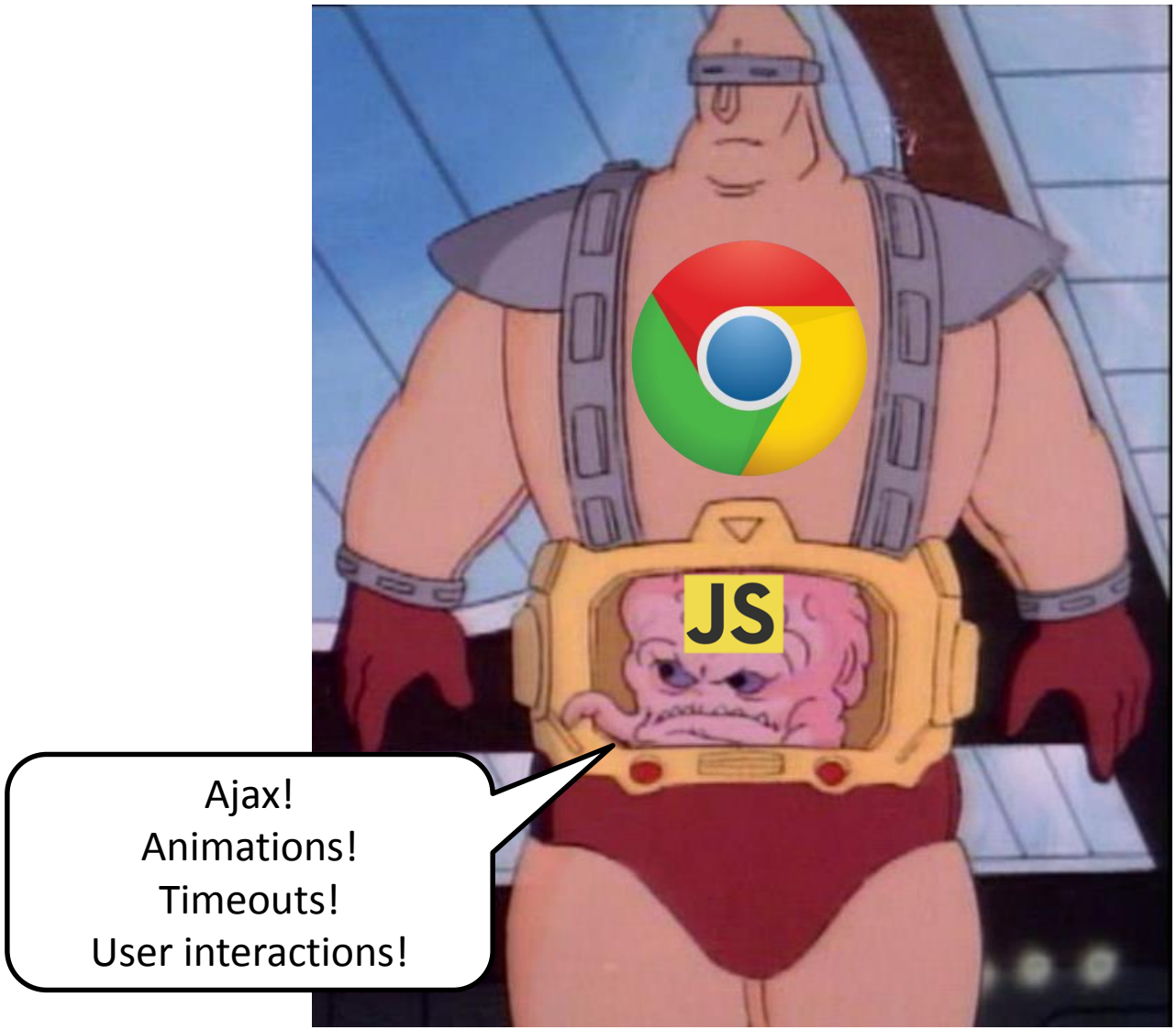
Javascript is blocking

```
for (var i = 0; i < 10000000; i++) { };
```

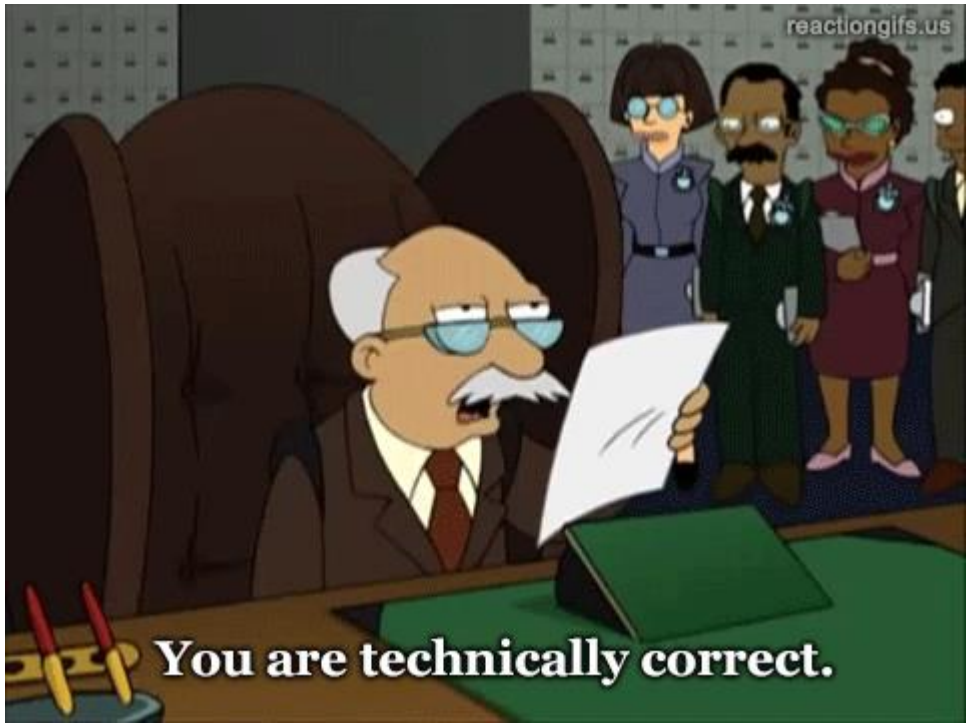
```
var now = new Date().getTime();
while (new Date().getTime() < now + 10000) { }
```



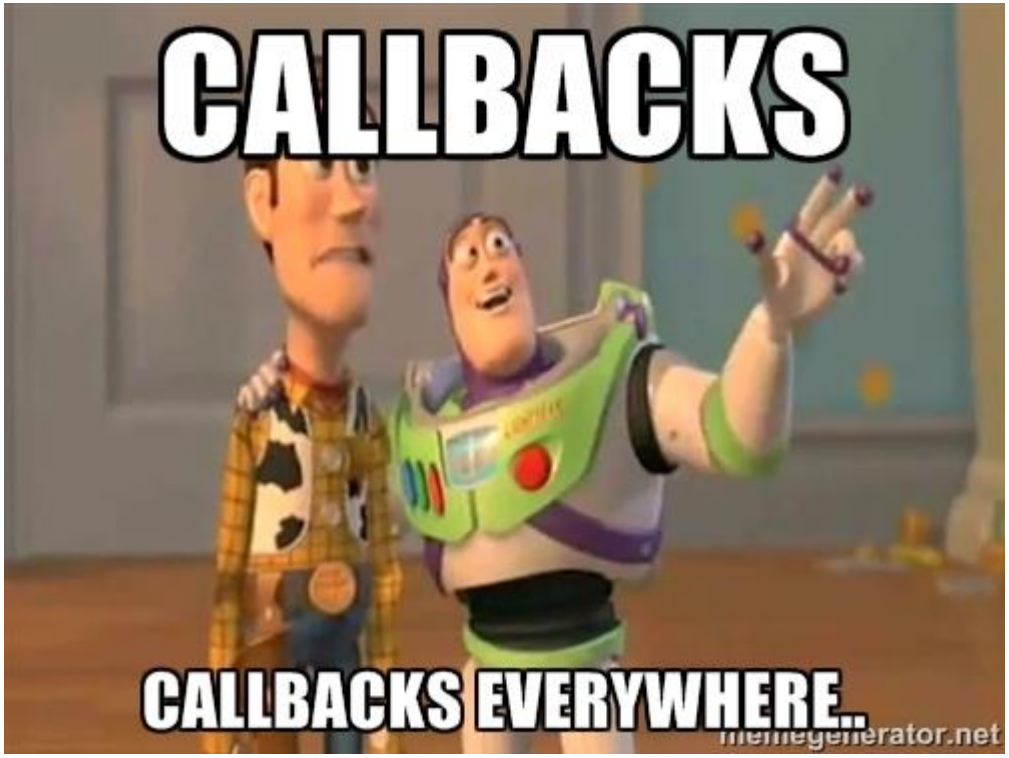
js is blocking, everything else isn't



Technically…



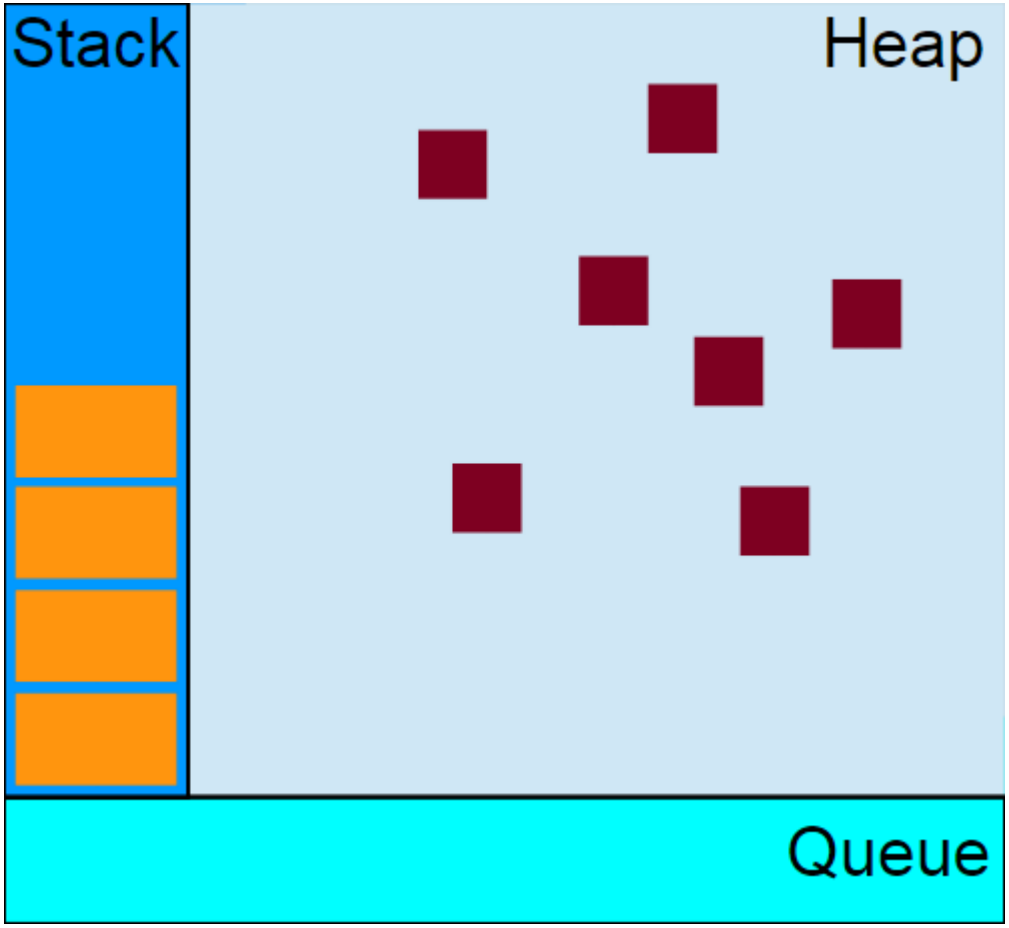
Callbacks



AND PROMISES…

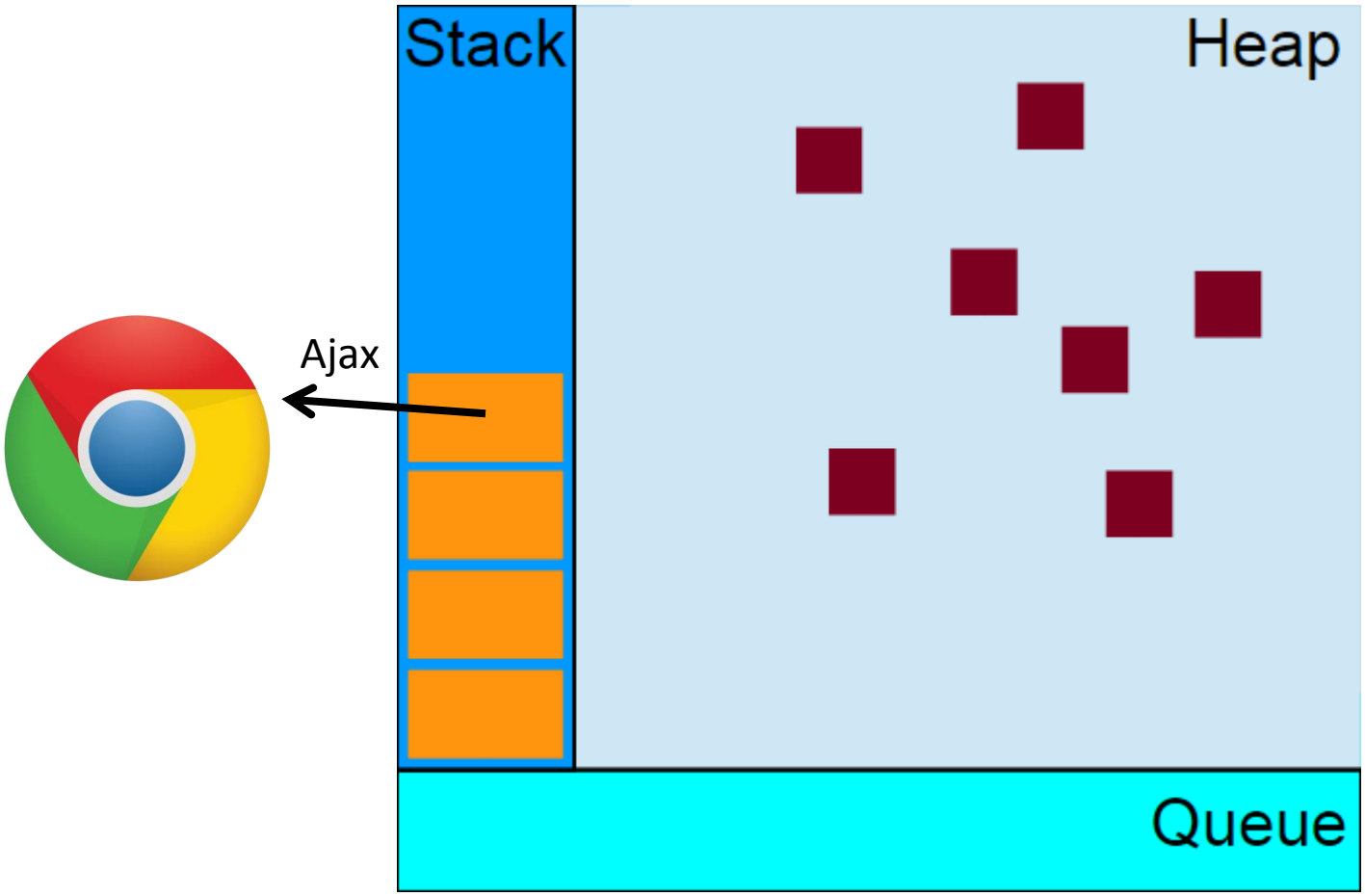
AND ASYNC\AWAIT…

Javascript Runtime

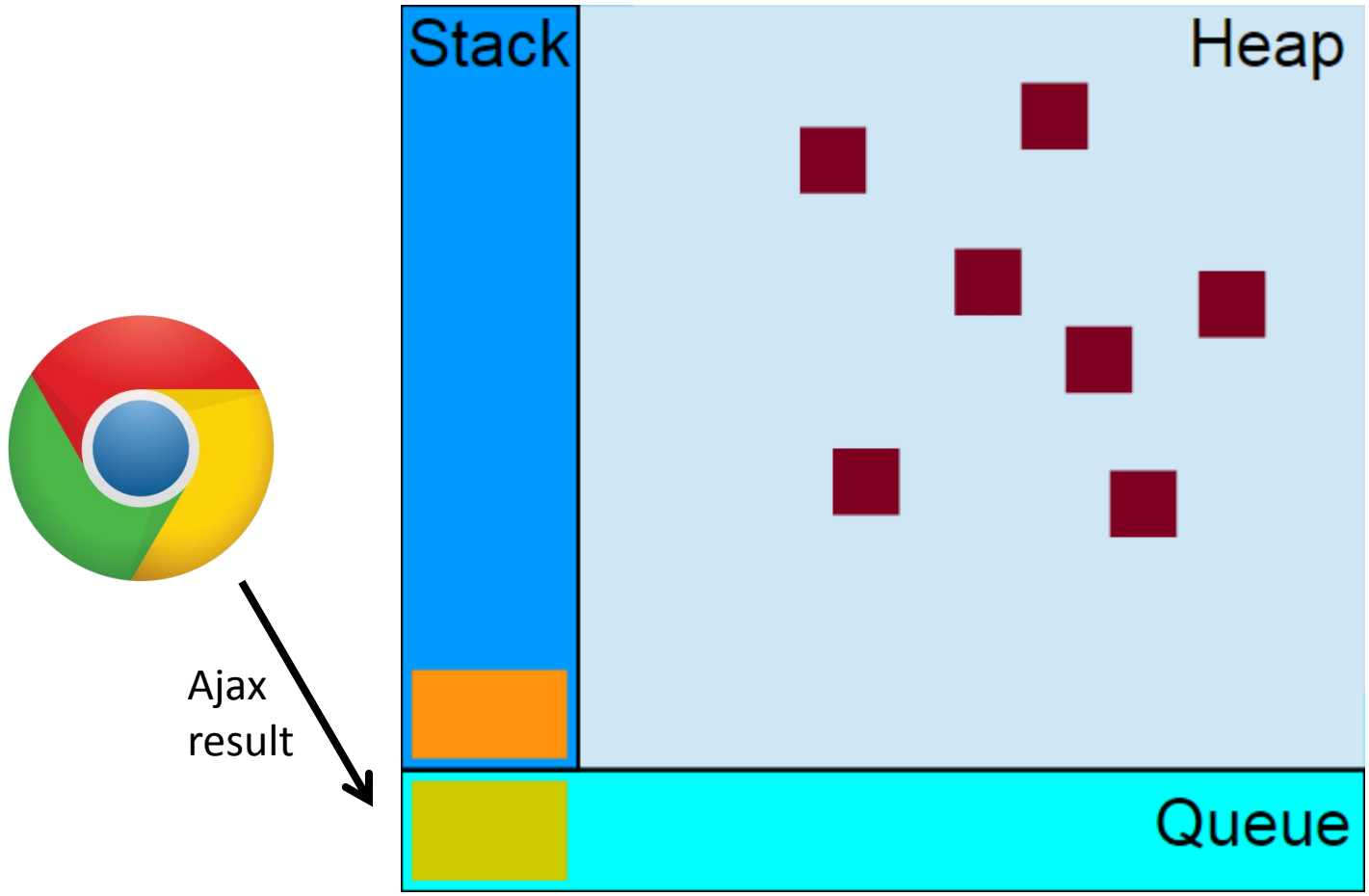


Source: [MDN Concurrency model and Event Loop]

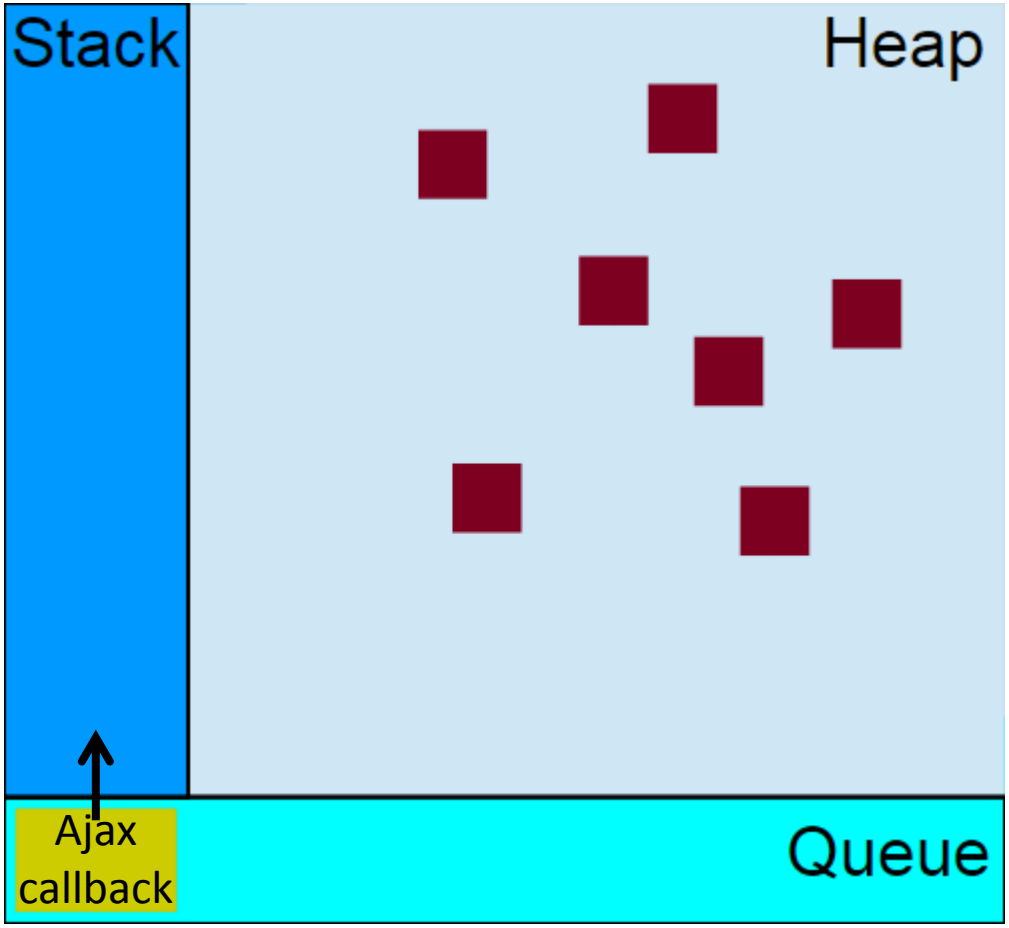
Javascript Runtime



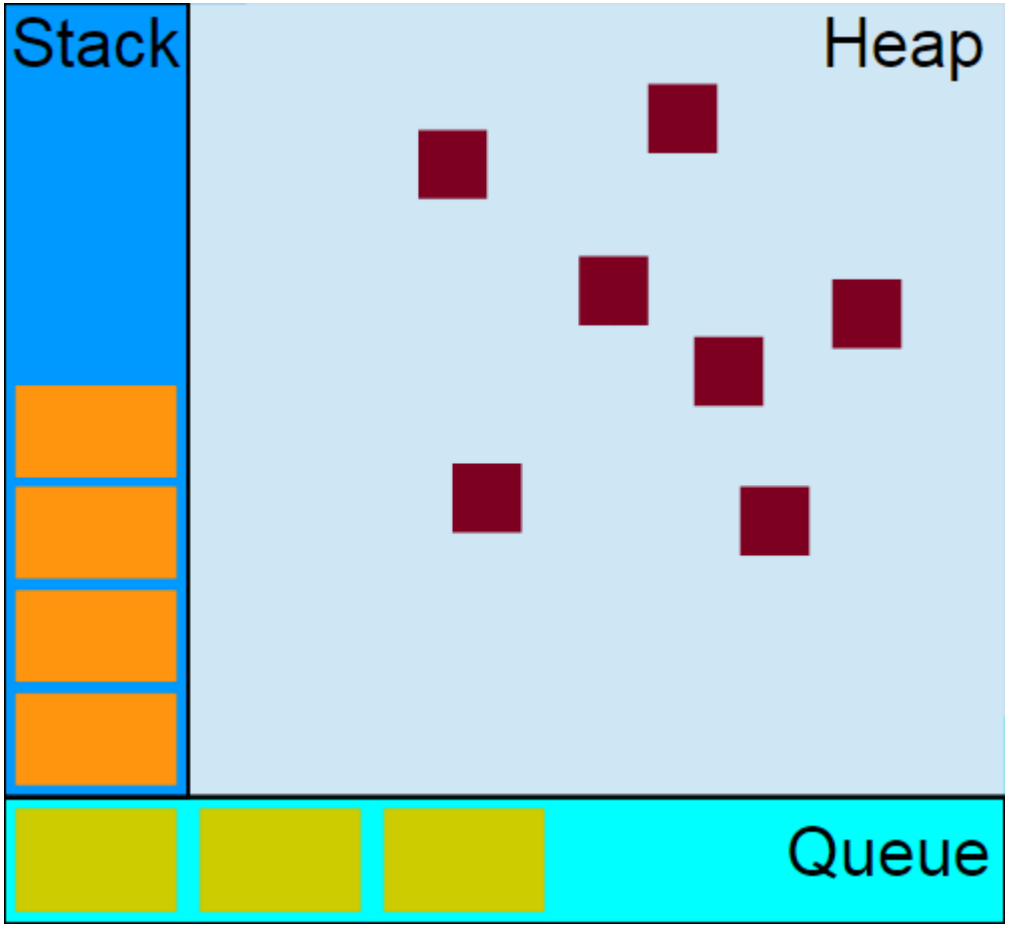
Javascript Runtime



Javascript Runtime



Javascript Runtime



The Event Loop

```
while (queue.waitForMessage()) {
  queue.processNextMessage();
}
```

Can js operations be non-blocking?

`setTimeout` to the resuce

```
function chunk(array, process, context) {
  var items = array.concat();   //clone the array
  setTimeout(function () {
    var item = items.shift();
    process.call(context, item);

    if (items.length > 0) {
      setTimeout(arguments.callee, 100);
    }
  }, 100);
}
```

Source: Nicholas C. Zakas' "Timed array processing in JavaScript"

“Asyncify”™ sync operations

```
setTimeout(myFunc, 0);
```

“Asyncify”™ sync operations

```
setTimeout(myFunc, 0);
```

```
setImmediate(myFunc);
```

DEMO

“Asyncify”™ sync operations

```
setTimeout(myFunc, 0);
```

```
setImmediate(myFunc);
```

https://github.com/YuzuJS/setImmediate

```
requestAnimationFrame(myFunc);
```

```
requestIdleCallback(myFunc);
```

```
process.nextTick(myFunc);
```

```
window.addEventListener("message", myFunc);
```

```
Promise.resolve().then(myFunc);
```

Quiz time

```
setTimeout(function () {
  console.log('hello');
}, 0);

['world'].forEach(function (str) {
  console.log(str);
});

console.log('!');
```

Quiz time # 2

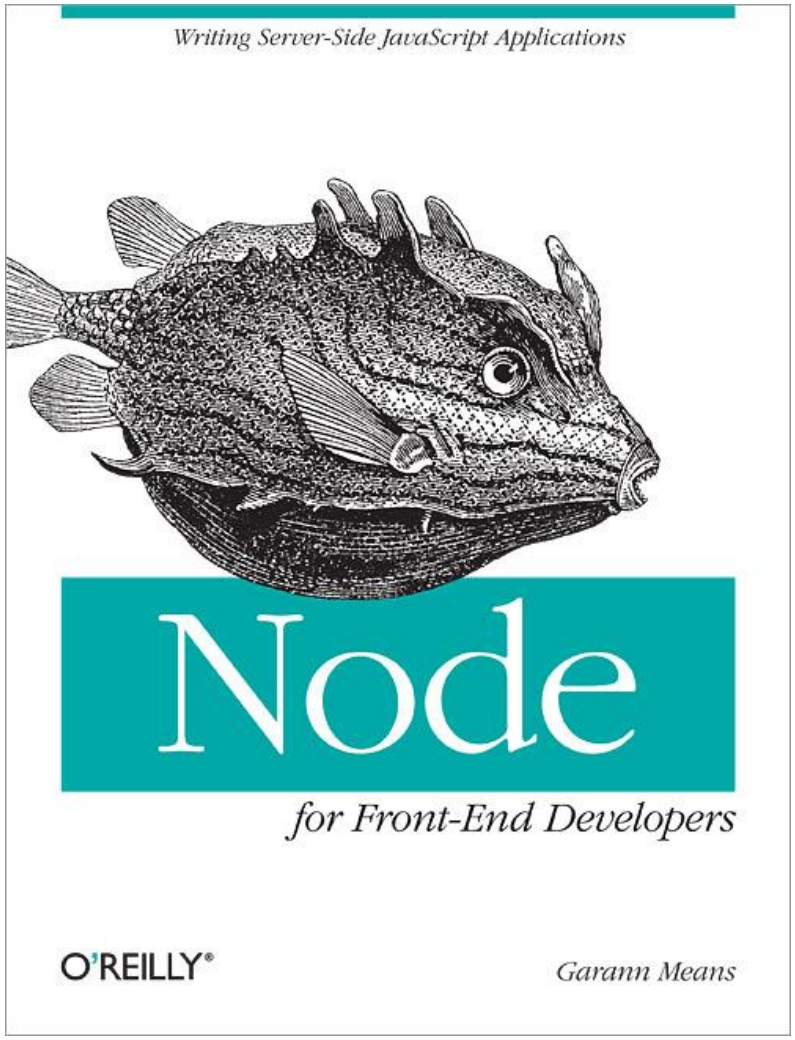
```
console.log('script start');

setTimeout(function () {
  console.log('setTimeout');
}, 0);

Promise.resolve().then(function () {
  console.log('promise1');
}).then(function () {
  console.log('promise2');
});

console.log('script end');
```

Source: Jake Archibald's "Tasks, microtasks, queues and schedules"

Web Workers

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

What about node.js?



Further reading

- What the heck is the event loop anyway?
- Writing a Non-blocking JavaScript Quicksort
- The case for setImmediate()
- Dissecting the HTML 5 Event Loop - Loops, Task Queues and Tasks
- The Basics of Web Workers

Questions?

Yonatan Mevorach

 blog.cowchimp.com  @cowchimp

 yonatan@sears.co.il