I Promise asynchrony is easy!

The basis of modern JS 🤖 👽

Pieces of a cake

1. One `.js` file
2. Many `chunks`
3. Some will execute `now` and others `later`

The most basic unit of `chunk` is...

```
function foo() {}
```

FUNCTIONS!!

The curse of the now

1. `Later != After now`

```
const data = fetch("http://some.url.com");
console.log(data);
```

If something its going to take a time, by definition its completed asynchronously and we will not have blocking behavior as you might intuitively expect or want.

Call me back please 😢

The simplest way of `waiting` from **now** until **later** is to use a function, commonly called `callback`.

```
fetch("http://some.url.com", function callMeBack(data) {
  /** some ugly code */
});
```

Any time you wrap a portion of code into a function and specify that it should be executed in response to some event (`timer`, `mouse click`, `Ajax response`, etc.), you are creating a later chunk of your code, and thus introducing asynchrony to your program.

An infinite loop of ¿Pain? 😈

JavaScript itself has actually never had any direct notion of asynchrony built into it.

The JS engine itself has never done anything more than execute a single chunk of your program at any given moment, when asked to.

An infinite loop of ¿Pain? 😈 x2

1. JS runs in a `hosting environment` (a browser)
2. `Node.js` (server side)
3. The common thing in all the environments is the...

EVENT LOOP!! 💥

An infinite loop of ¿Pain? 😈 x3

- The JS engine has had no innate sense of time
- But has instead been an on-demand execution environment for any arbitrary snippet of JS.
- It's the surrounding environment that has always scheduled "events" (JS code executions).

GIVE ME CODE PLEASE!

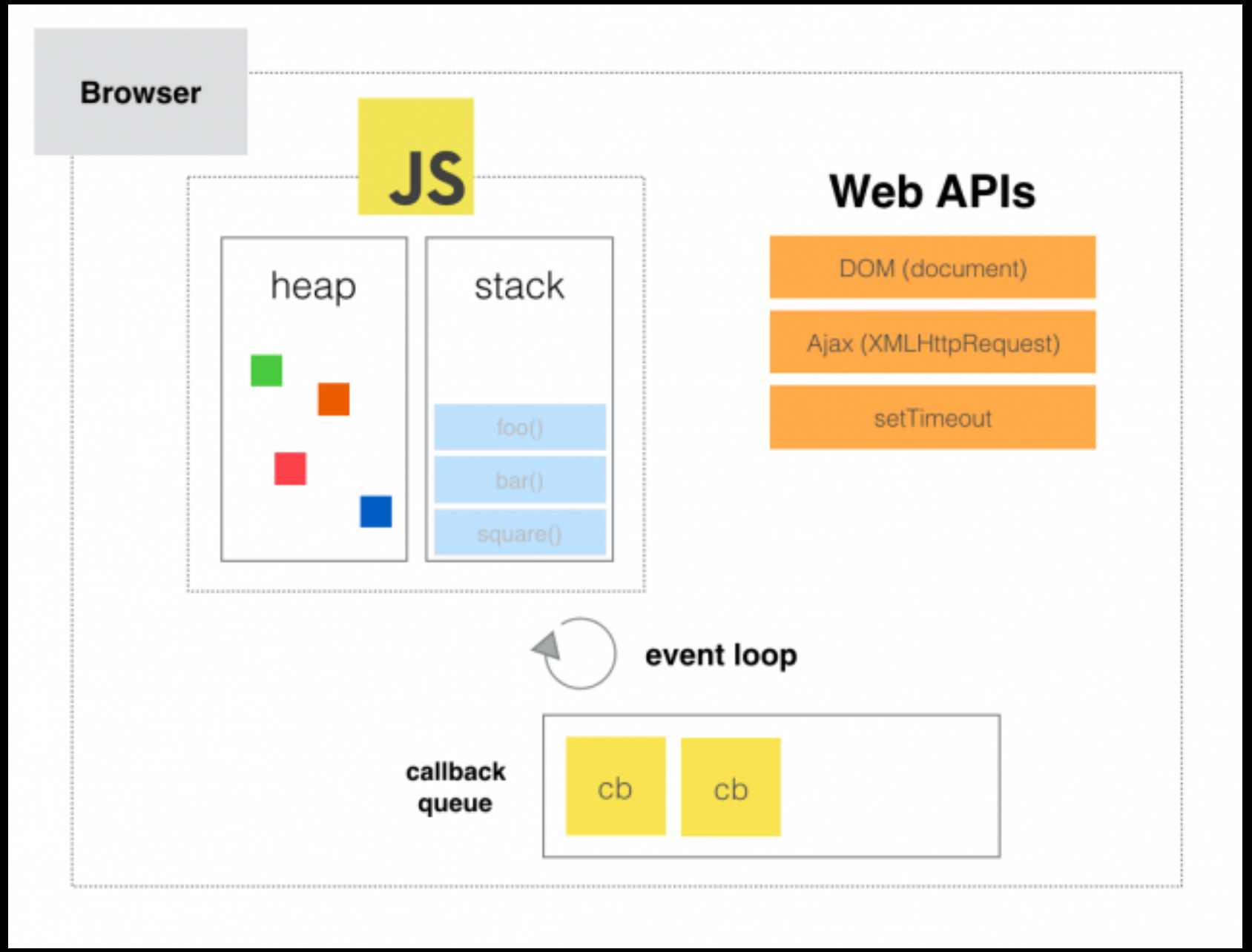
An infinite loop of ¿Pain? 😈 x4

How the event loop is implemented??

```
// `eventLoop` is an array that acts as a queue (first-in, first-out)
var eventLoop = [];
var event;

// keep going "forever"
while (true) {
  // perform a "tick"
  if (eventLoop.length > 0) {
    // get the next event in the queue
    event = eventLoop.shift();

    // now, execute the next event
    try {
      event();
    } catch (err) {
      reportError(err);
    }
  }
}
```


Browser
JS
heap
stack
foo()
bar()
square()
Web APIs
DOM (document)
Ajax (XMLHttpRequest)
setTimeout
event loop
callback queue
cb
cb

Paralellism, Concurrency, dude WTF?

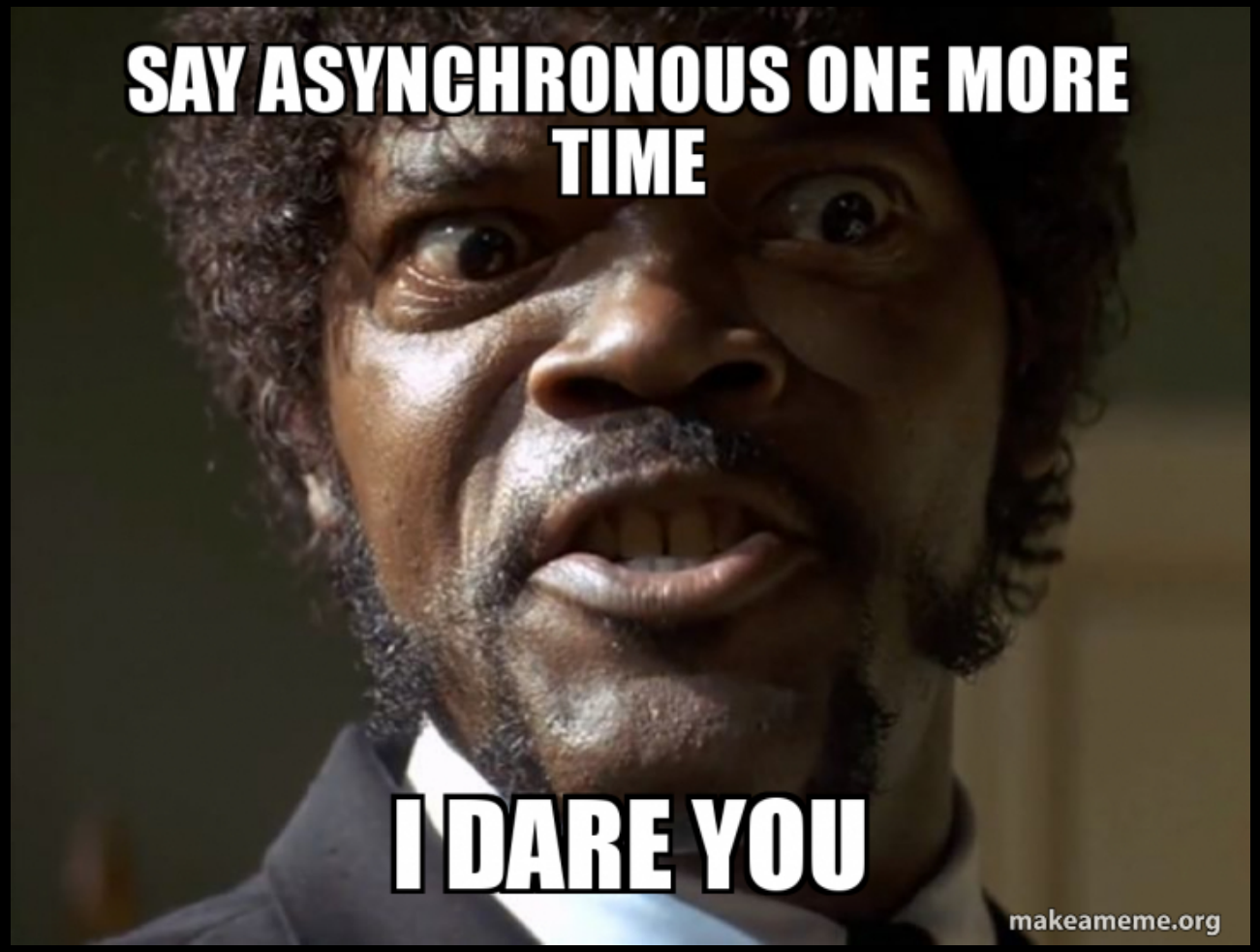
- Async. `now` and `later`
- Parallel. Things being able to occur `simultaneously`.
  - Processes and threads
- Concurrency. Concurrency is when two or more "processes" are executing simultaneously over the same period.
- Event loop. Breaks the work in Tasks and executes them in serial.

TL;DR

1. Whenever there are events to run, the event loop runs until the queue is empty. Each iteration of the event loop is a "tick." User interaction, IO, and timers enqueue events on the event queue.
2. At any given moment, only one event can be processed from the queue at a time.
3. Concurrency is when two or more chains of events interleave over time, such that from a high-level perspective, they appear to be running simultaneously (even though at any given moment only one event is being processed).

Links

1. 9.4: JS timeout function
2. The best explanation about event loop


SAY ASYNCHRONOUS ONE MORE TIME
I DARE YOU
makeameme.org