Asynchronous Programming

Javascript Plus Session-16

CLARUSWAY
WAY TO REINVENT YOURSELF

Table of Contents

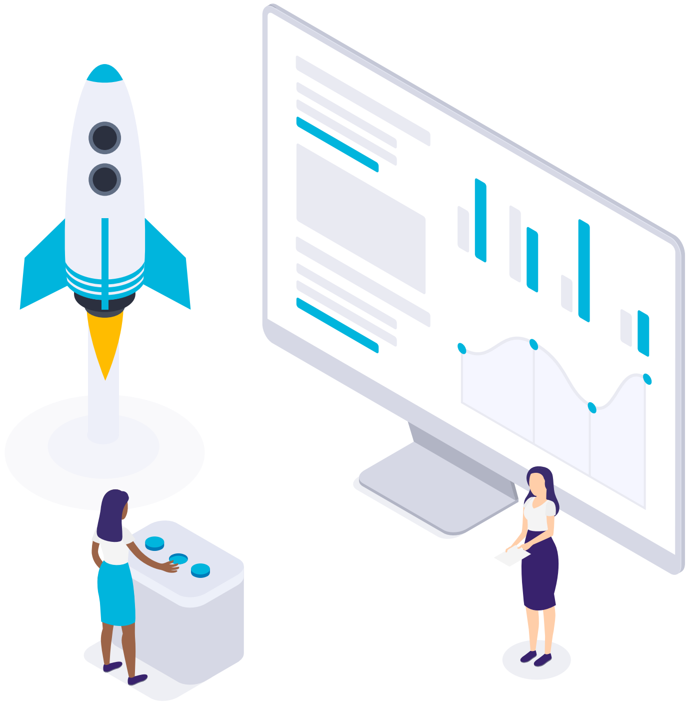
```
<script>
if(age > 19){
    alert("Adult");
} else{
    alert("Teenager");
}
</script>
```

Did you finish Javascript Plus pre-class material?

Play Kahoot!

1

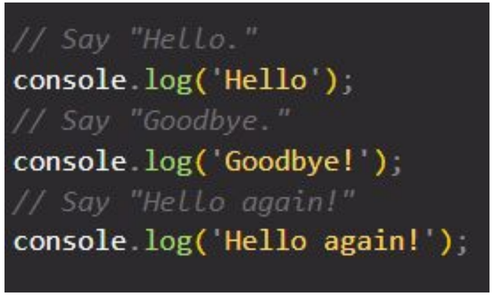
Intro to Asynchronous Programming



CLARUSWAY
WAY TO REINVENT YOURSELF

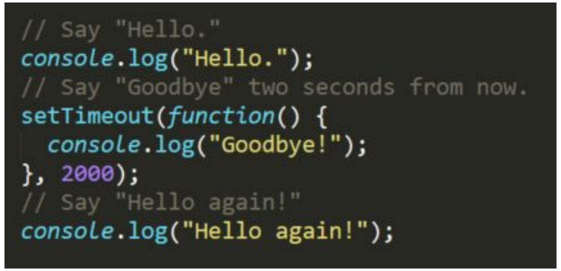
Intro to Asynchronous Programming

synchronous



```
// Say "Hello."
console.log('Hello');
// Say "Goodbye."
console.log('Goodbye!');
// Say "Hello again!"
console.log('Hello again!');
```

asynchronously



```
// Say "Hello."
console.log("Hello.");
// Say "Goodbye" two seconds from now.
setTimeout(function() {
  console.log("Goodbye!");
}, 2000);
// Say "Hello again!"
console.log("Hello again!");
```

- Say "Hello"
- Say "Goodbye"
- Say "Hello again"

- Say "Hello"
- Say "Hello again"
- Do nothing for two seconds
- Say "Goodbye"

JavaScript is single threaded. Only one thing can happen at a time, on a single main thread, and everything else is blocked until an operation completes.

alert() is a synchronous operation. It will block everything else from happening until it completes.

CLARUSWAY
WAY TO REINVENT YOURSELF

2

Async Callbacks

Async Callbacks

Async callbacks are functions that are specified as arguments when calling a function which will start executing code in the background.

When the background code finishes running, it calls the callback function to let you know the work is done or to let you know that something of interest has happened.

Using callbacks is slightly old-fashioned now, but you'll still see them in use in a number of older-but-still-commonly-used APIs.

Async Callbacks

The `setTimeout()` method executes a block of code after the specified time. The method executes the code only once.

```
// program to display a text using setTimeout method

function hello() {
  console.log('Hello world');
}

let intervalId = setTimeout(hello, 3000);

console.log('This message is shown first');
console.log('Id: ' + intervalId);
```

```
OUTPUT:

This message is shown first
Id: 1
Hello world
```

`setTimeout()` method calls the `greet()` function after 3000 milliseconds (3 second).

Hence, the program displays the text Hello world only once after 3 seconds.

Async Callbacks

The `setInterval()` method repeats a block of code at every given timing event.

```
// program to display a text using setInterval method

function hello() {
  console.log('Hello world');
}
setInterval(hello, 2000);
```

```
OUTPUT:

Hello world
Hello world
Hello world
Hello world
Hello world
....
```

`setInterval()` method calls the `greet()` function every 2000 milliseconds(2 second). The program displays the text Hello world once every 2 seconds.

CLARUSWAY
WAY TO REINVENT YOURSELF

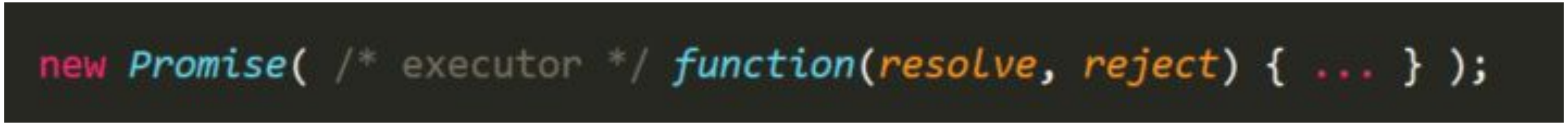
3

Promises

CLARUSWAY
WAY TO REINVENT YOURSELF

Promises

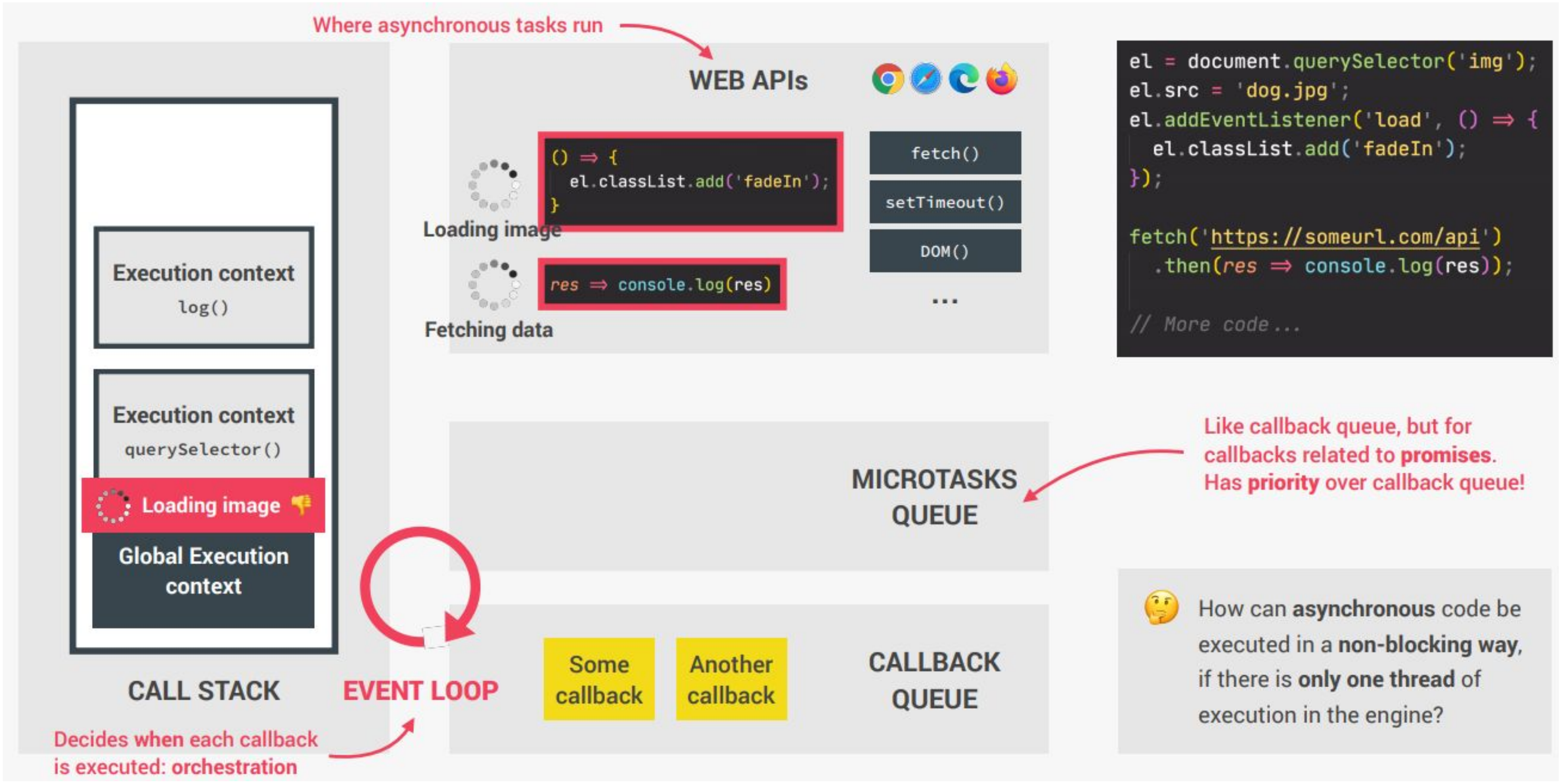
- Object that keeps track about whether a certain event has happened already or not;
- Determines what happens after the event has happened
- Implements the concept of a future value that we are expecting
- The promise object represents the eventual completion(or failure) of an asynchronous operation, and its resulting value.



```
new Promise( /* executor */ function(resolve, reject) { ... } );
```

CLARUSWAY
WAY TO REINVENT YOURSELF

Promises



```
() => {
  el.classList.add('fadeIn');
}
```

```
res => console.log(res)
```

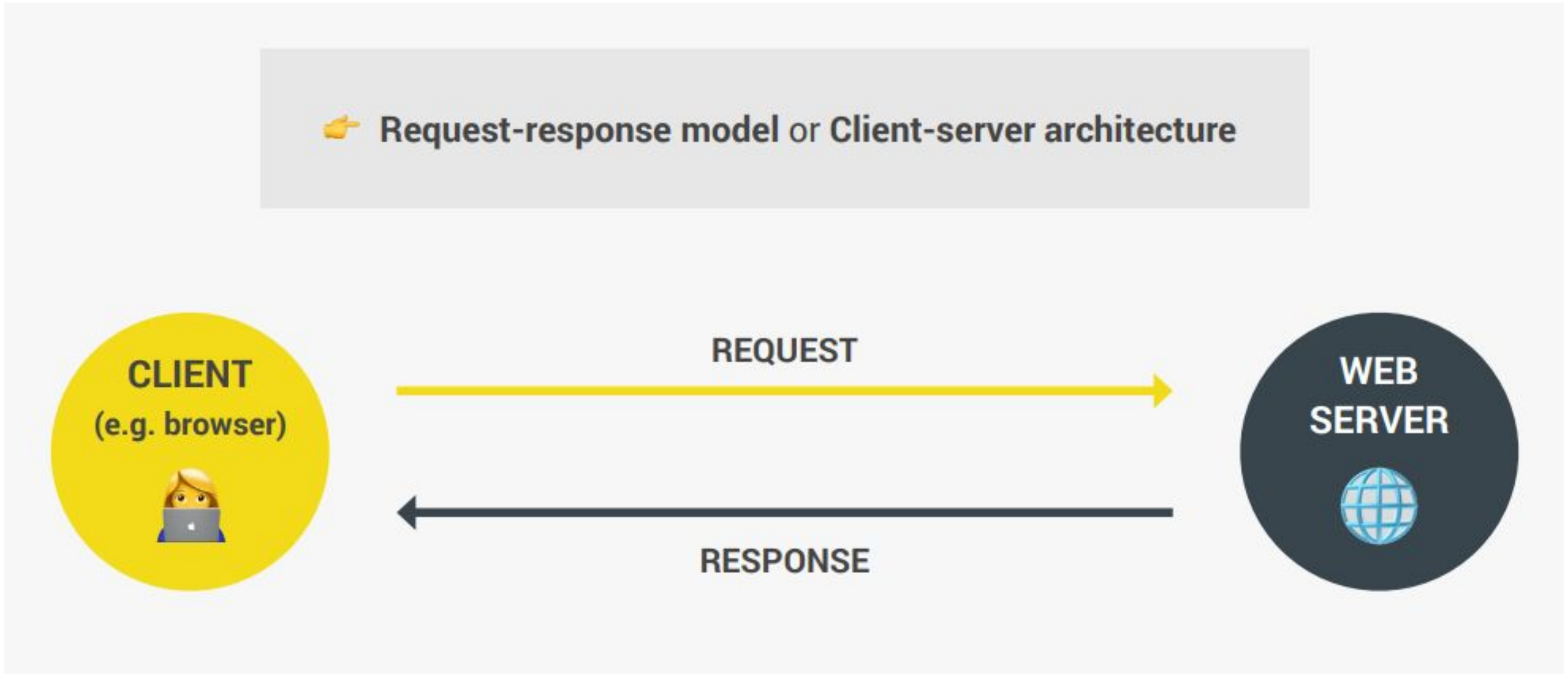
```
el = document.querySelector('img');
el.src = 'dog.jpg';
el.addEventListener('load', () => {
  el.classList.add('fadeIn');
});

fetch('https://someurl.com/api')
  .then(res => console.log(res));

// More code ...
```

🤔 How can **asynchronous** code be executed in a **non-blocking way**, if there is **only one thread** of execution in the engine?

Promises



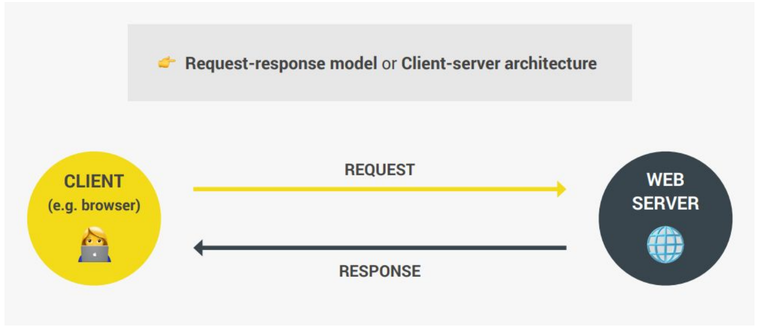
👉 **Request-response model** or **Client-server architecture**

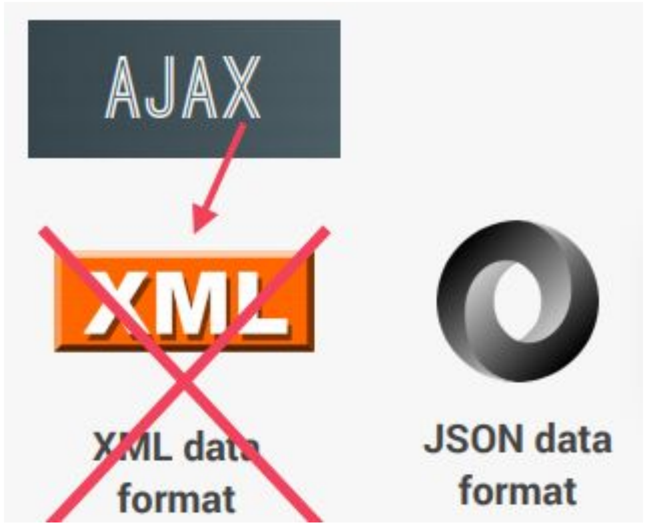
CLARUSWAY
WAY TO REINVENT YOURSELF

Promises

AJAX - Asynchronous JavaScript And XML:

Allows us to communicate with remote web servers in an asynchronous way. With AJAX calls, we can request data from web servers dynamically.





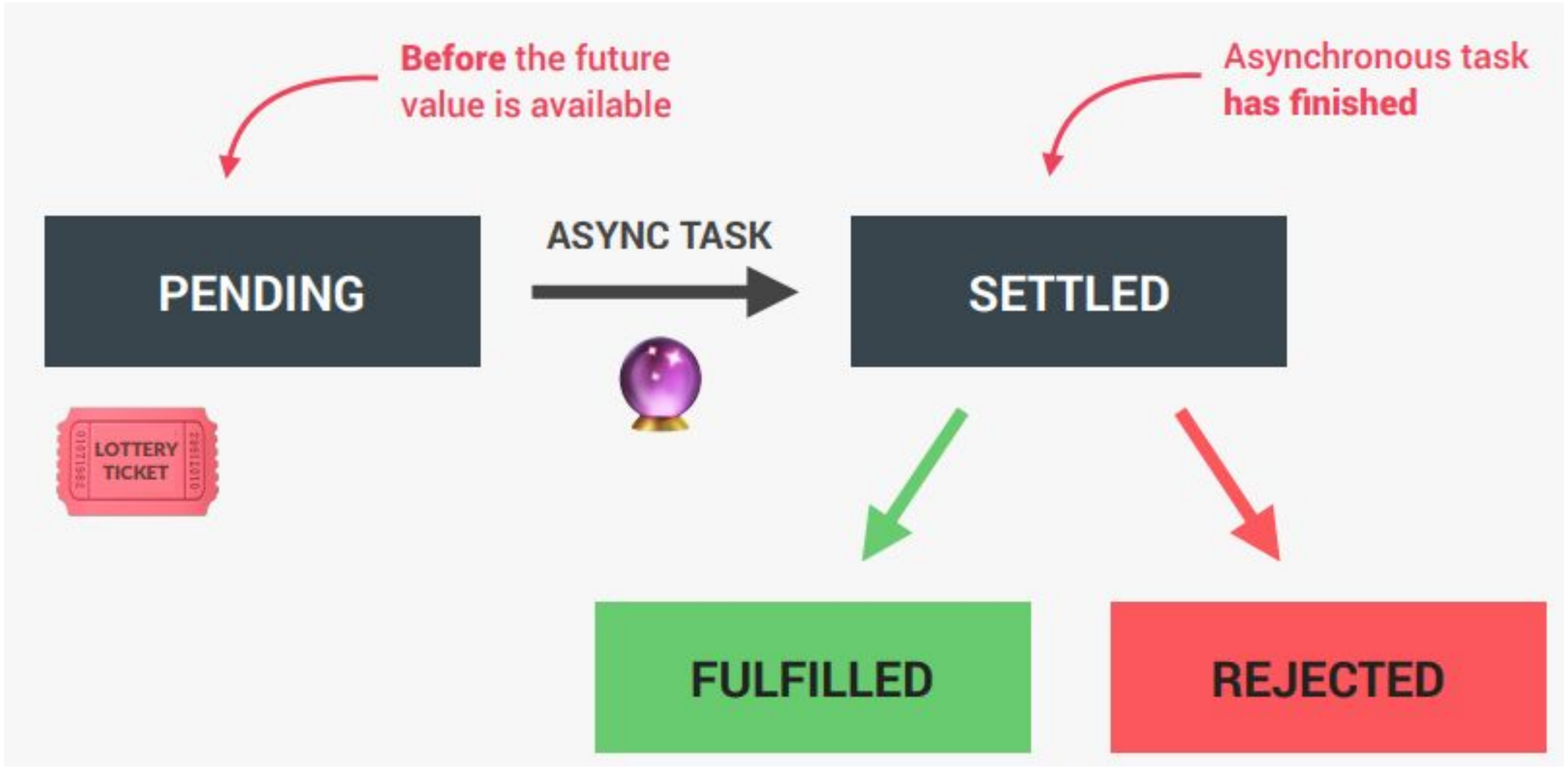
API - Application Programming Interface:

Piece of software that can be used by another piece of software, in order to allow applications to talk to each other.

Promises

- An object that is used as a placeholder for the future result of an asynchronous operation.
- A container for an asynchronously delivered value.
- **A container for a future value**.
- We **no longer need** to rely on **events and callbacks** passed into asynchronous functions to handle asynchronous results;
- Instead of nesting callbacks, we can chain promises for a sequence of asynchronous operations: **escaping callback hell**

Promises


Before the future value is available
Asynchronous task has finished
PENDING
ASYNC TASK
SETTLED
LOTTERY TICKET
FULFILLED
REJECTED

4

Async/await

async/await

- The newest way to write asynchronous code in JavaScript
- It is non blocking (just like promises and callbacks)
- Async/Await was created to simplify the process of working with and writing chained promises
- Async functions return a Promise. If the function throws error, the Promise will be rejected. If the function returns a value, the Promise will be resolved.

5

Parallel Promising

Parallel Promising

- **Promise.all(iterable)**

  Wait for all promises to be resolved, or for any to be rejected.

- **Promise.allSettled(iterable)**

  Wait until all promises have settled (each may resolve or reject).

- **Promise.any(iterable)**

  Takes an iterable of Promise objects and, as soon as one of the promises in the iterable **fulfills**, returns a single promise that resolves with the value from that promise.

- **Promise.race(iterable)**

  Wait until any of the promises is resolved or rejected.

Let's get our hands dirty!

THANKS!

Any questions?