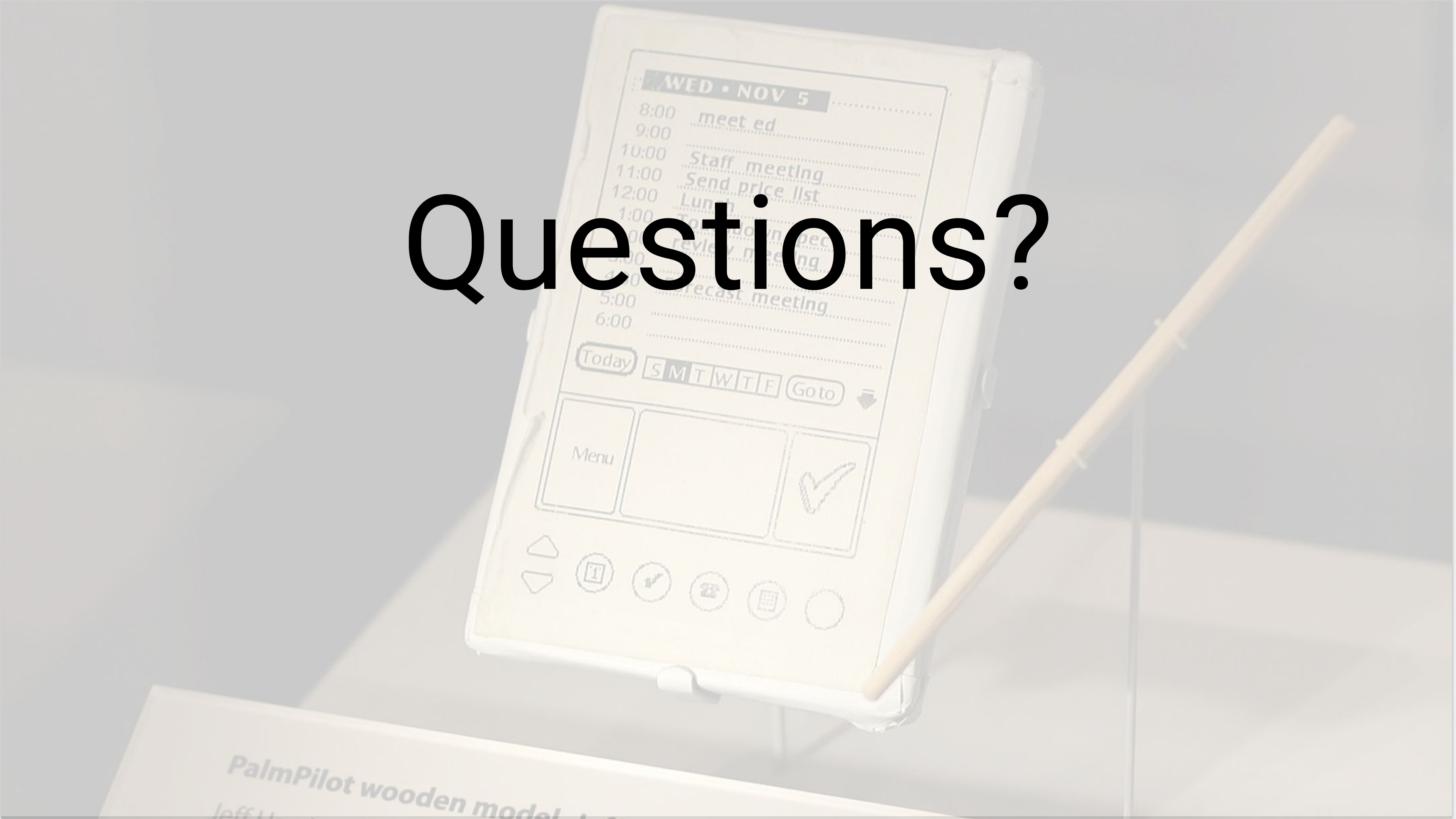
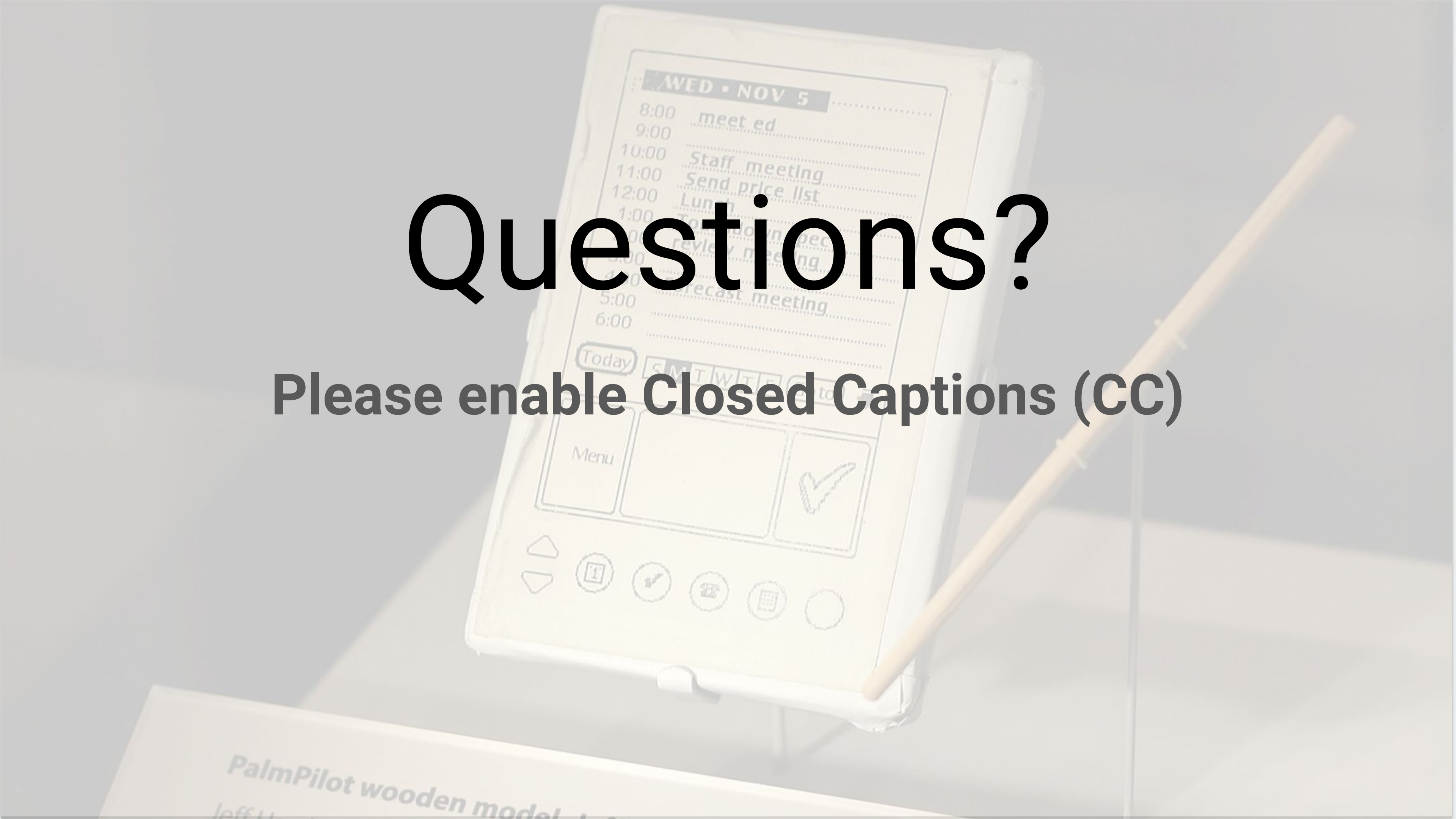
Asynchronous JavaScript

User Interface Development
EEcS 493 - Fall 2024

Questions?

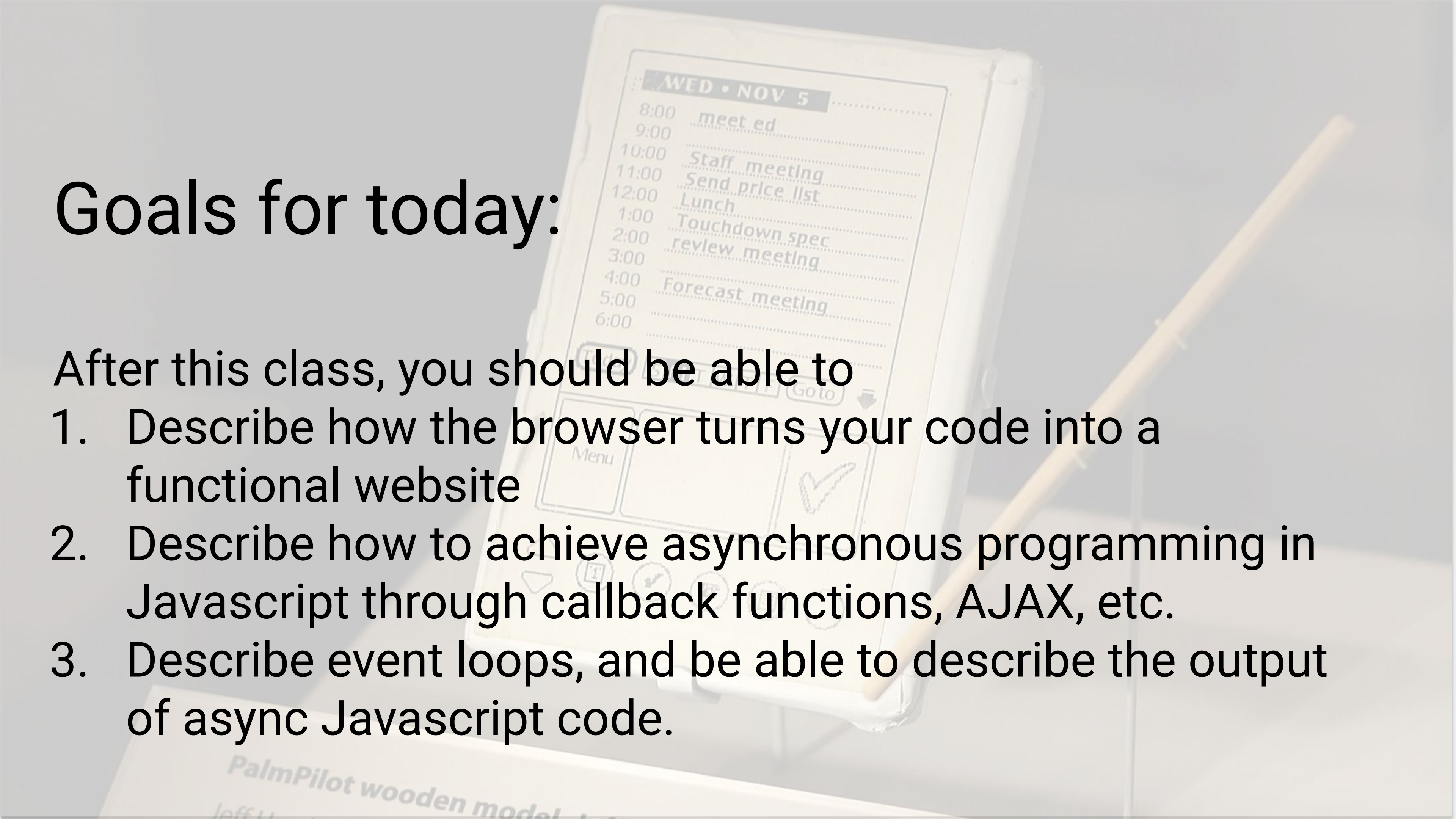

WED • NOV 5
8:00 meet ed
9:00
10:00 Staff meeting
11:00 Send price list
12:00 Lunch
5:00
6:00
Today
S M T W T F
Go to
Menu
PalmPilot wooden model



Questions?

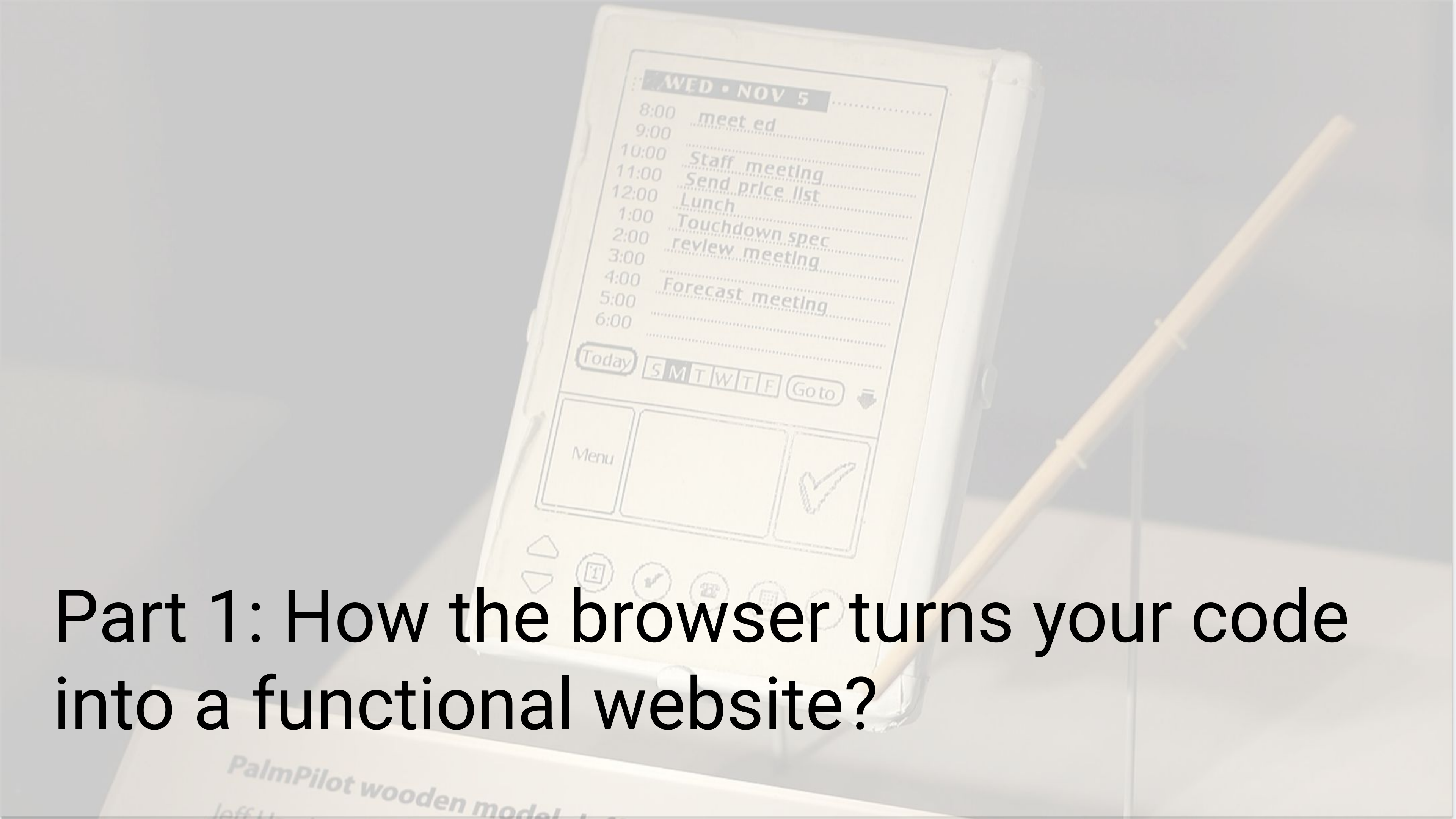
Please enable Closed Captions (CC)

Goals for today:



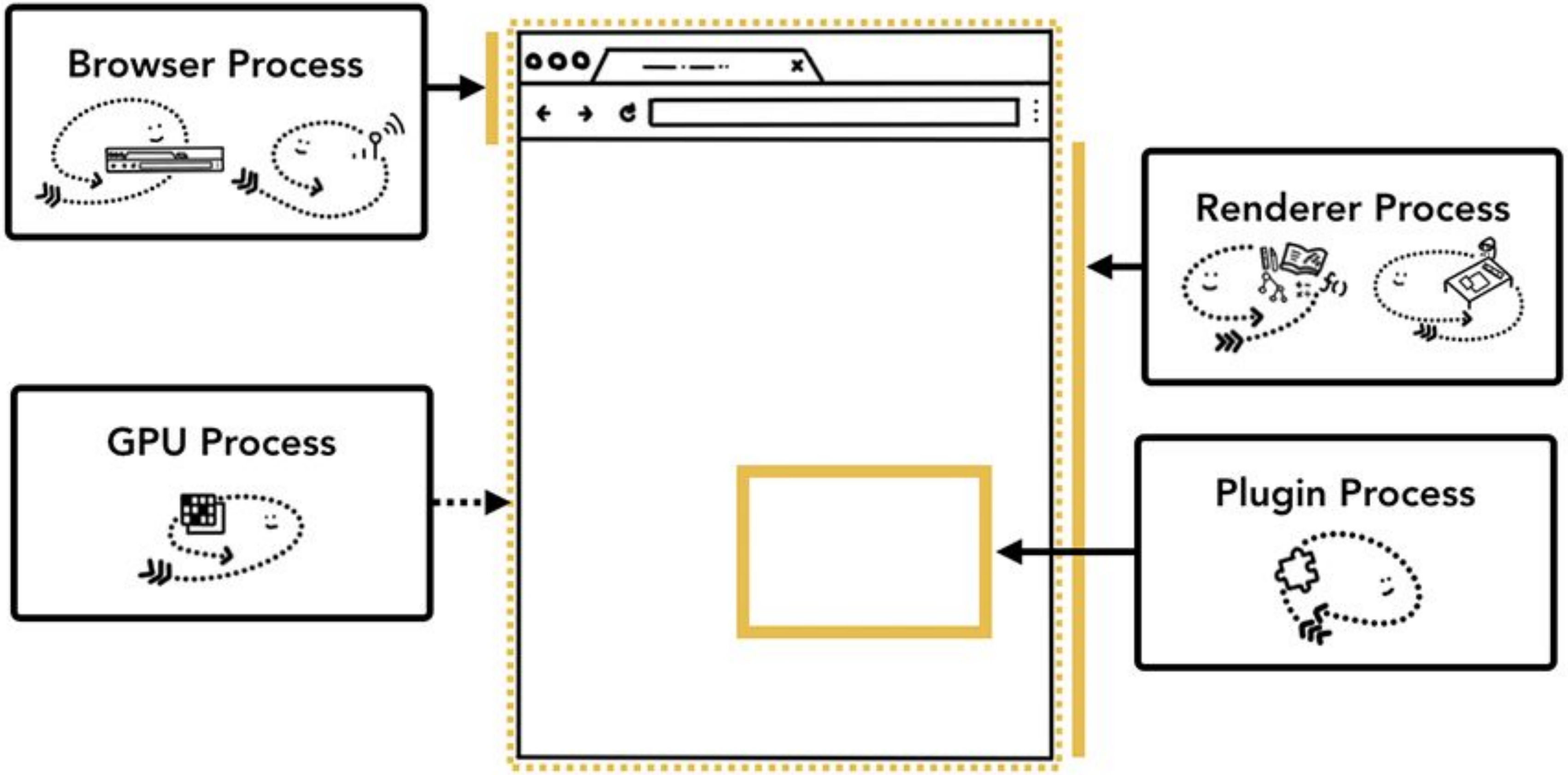
After this class, you should be able to

1. Describe how the browser turns your code into a functional website
2. Describe how to achieve asynchronous programming in Javascript through callback functions, AJAX, etc.
3. Describe event loops, and be able to describe the output of async Javascript code.



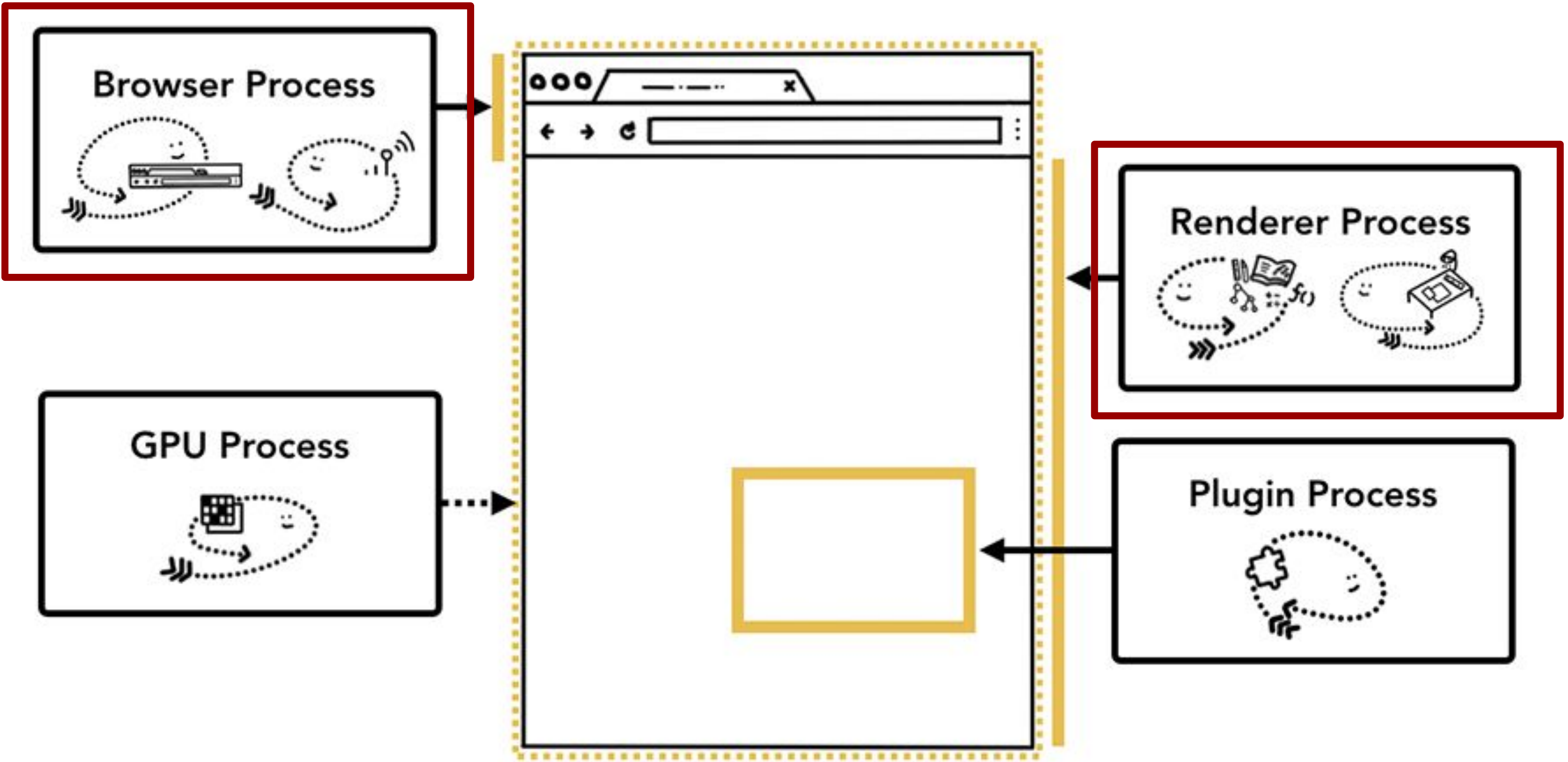
Part 1: How the browser turns your code into a functional website?

Web Browser - Chrome as an example



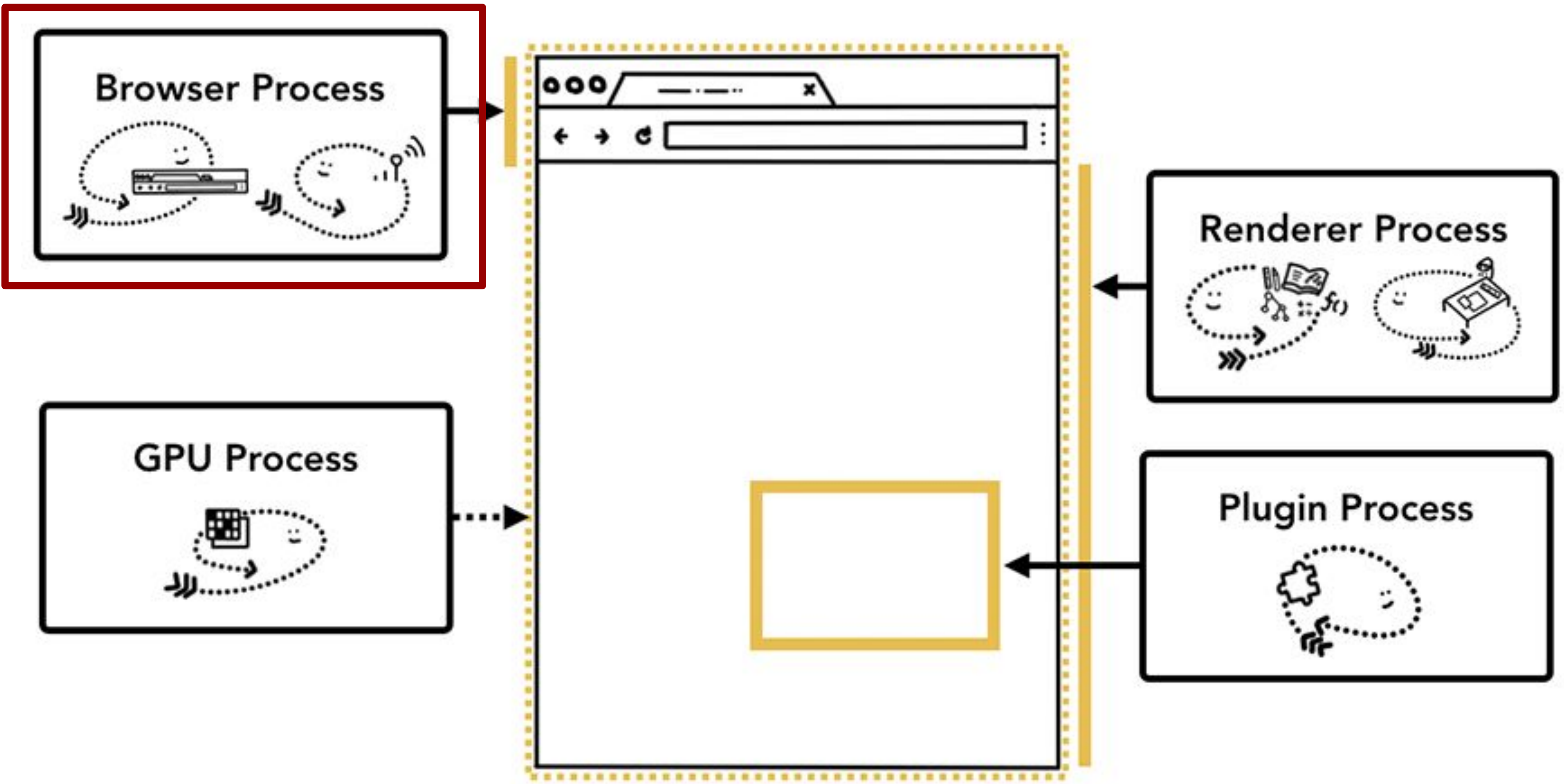
https://developers.google.com/web/updates/2018/09/inside-browser-part1

Web Browser - Chrome as an example



https://developers.google.com/web/updates/2018/09/inside-browser-part1

Browser Process



https://developers.google.com/web/updates/2018/09/inside-browser-part1

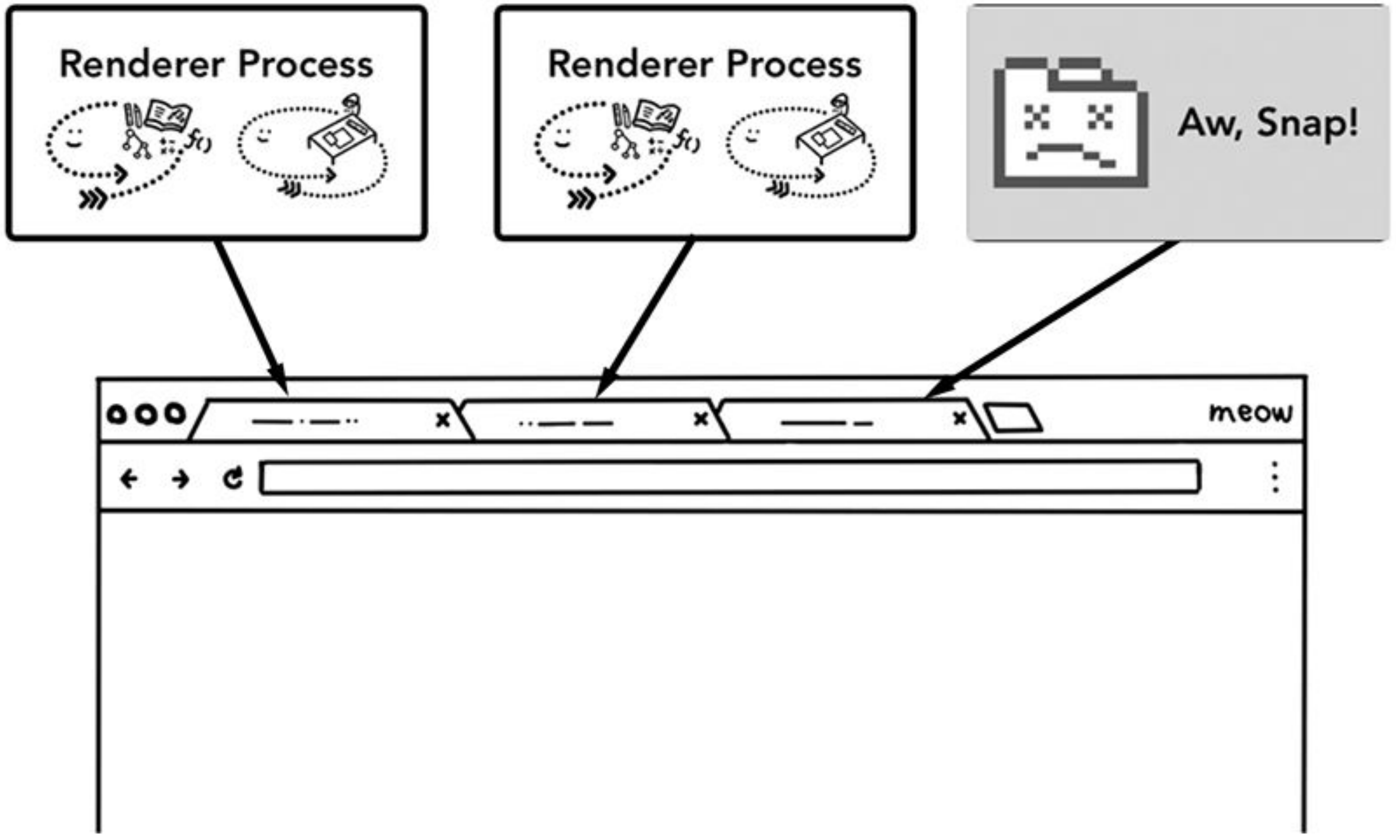
Browser Process - Controls User Interaction

- Main user interface, e.g., the address bar, bookmarks, forward and backward buttons
- Creation, switching and closing of tabs
- ...

Renderer Process

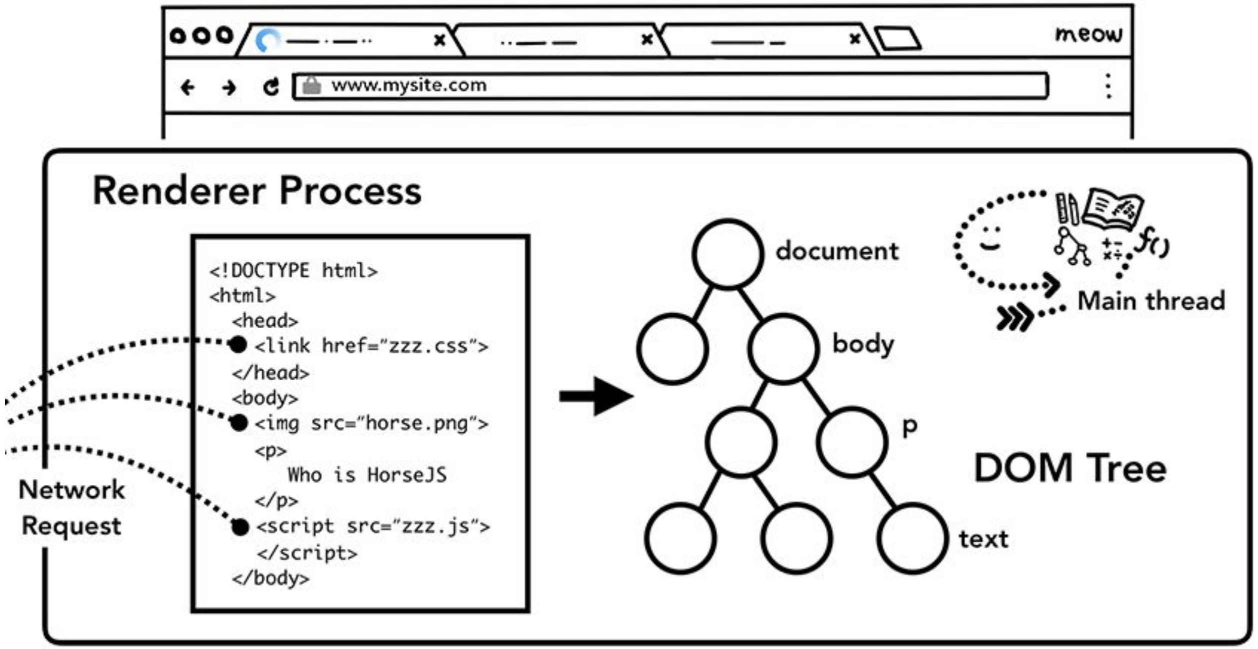
The render process is responsible for everything inside of a tab.

Renderer Process



Renderer Process - Creates DOM

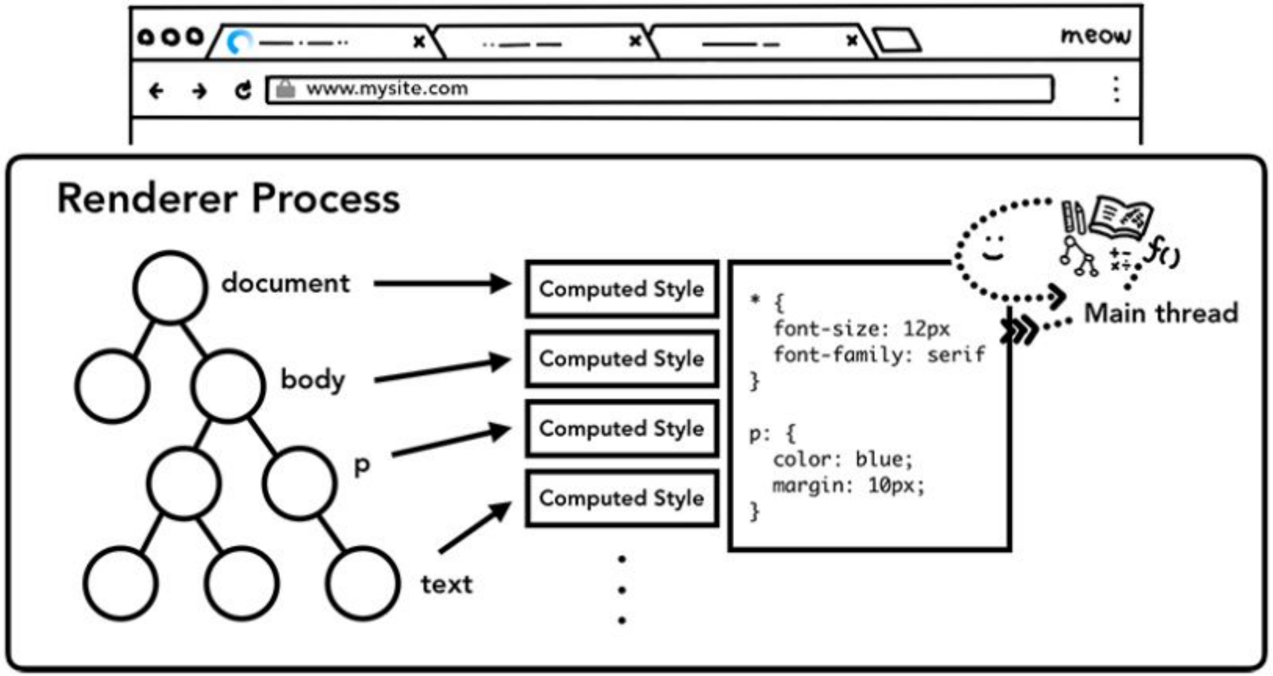
The renderer process parses the HTML code. When it finds a <script> tag, it pauses the parsing of HTML and load, parse, execute the JavaScript code first.



```
<!DOCTYPE html>
<html>
  <head>
  <link href="zzz.css">
  </head>
  <body>
  <img src="horse.png">
    <p>
      Who is HorseJS
    </p>
    <script src="zzz.js">
    </script>
  </body>
```

Renderer Process - Style and Layout

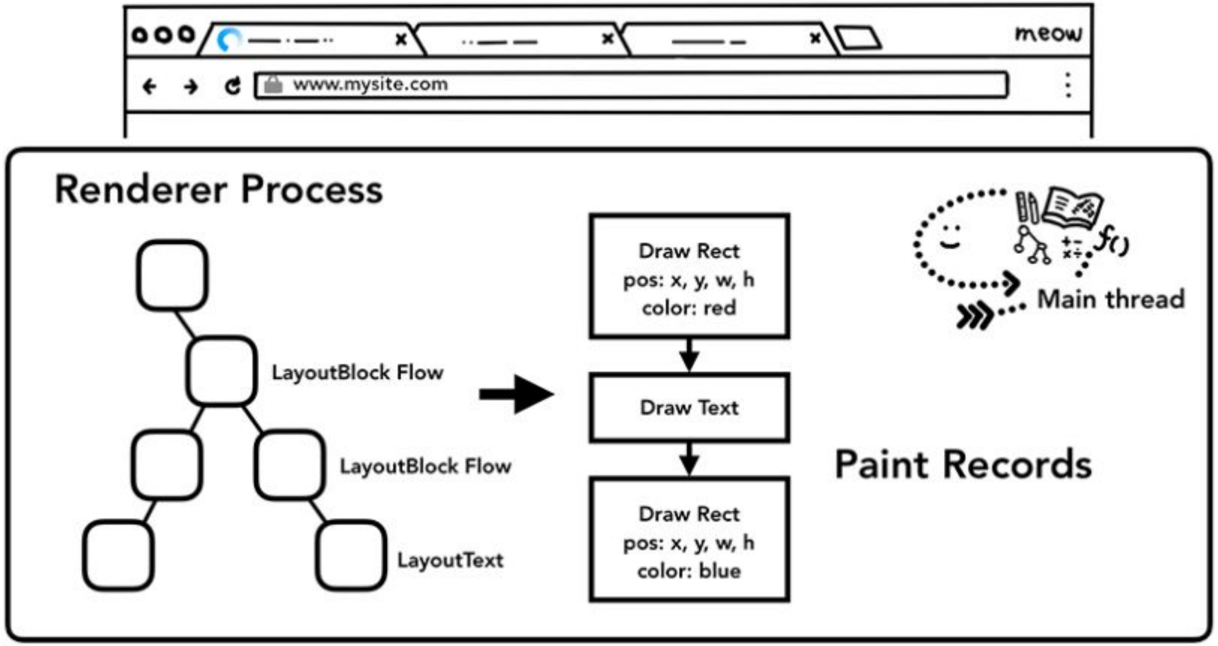
The render process parses the CSS code and determines the computed style and layout for each DOM node



```
* {
  font-size: 12px
  font-family: serif
}

p: {
  color: blue;
  margin: 10px;
}
```

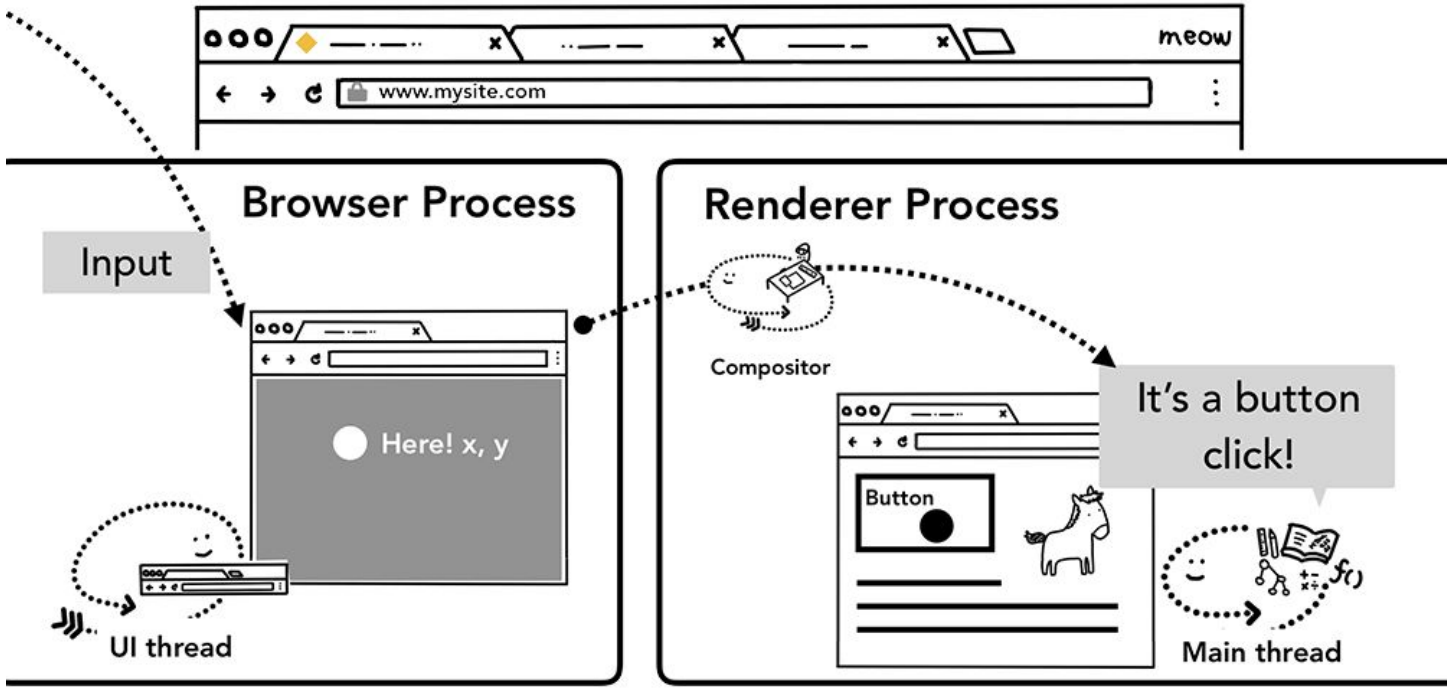
Renderer Process - Paint

The render process paints the records.

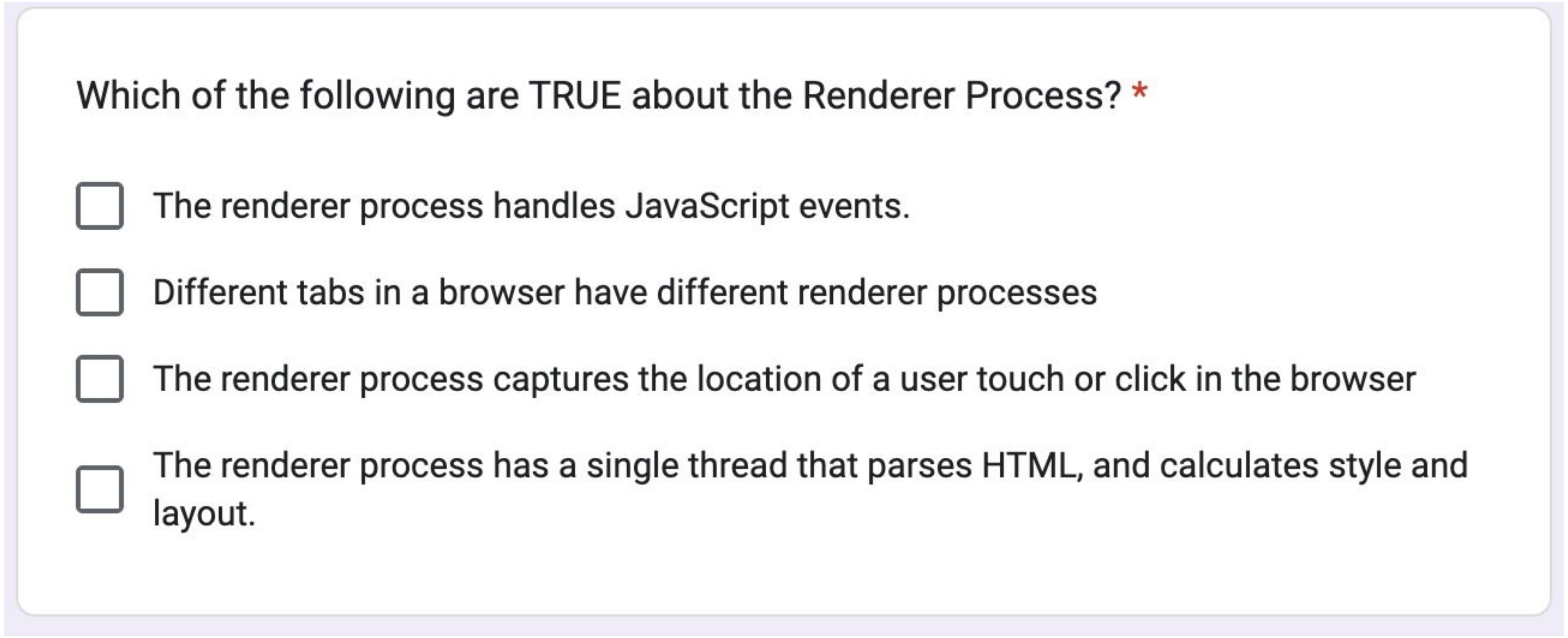


Browser Process Communicates with Renderer Process

The browser process takes user input (by position). Renderer process handles the event (through JavaScript)

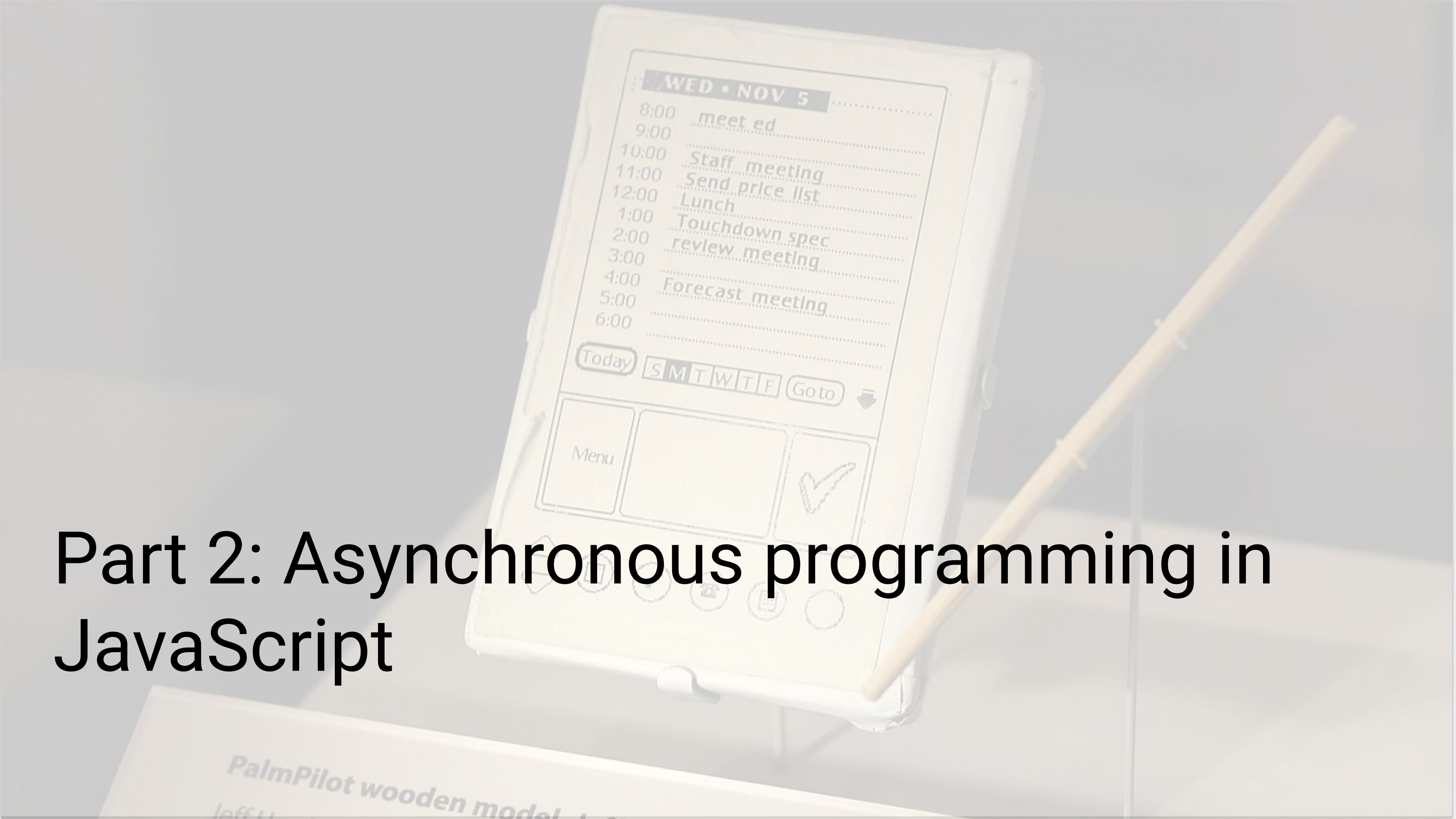


Lecture 6 - Survey 1



Which of the following are TRUE about the Renderer Process? *

- ☐ The renderer process handles JavaScript events.
- ☐ Different tabs in a browser have different renderer processes
- ☐ The renderer process captures the location of a user touch or click in the browser
- ☐ The renderer process has a single thread that parses HTML, and calculates style and layout.



Part 2: Asynchronous programming in JavaScript

JavaScript Execution

- JavaScript is single threaded
  - Run by the JavaScript runtime engine (V8) in the browser
- Ways to run JavaScript asynchronously:
  - **setTimeout()**
  - **Click events (event listeners)**
  - ajax()
  - Promise

setTimeout()

- A function in JavaScript to execute a piece of code after a specified delay.
- setTimeout(function(){}, delay)
- Returns an ID, which can be used to cancel the timeout.
- timeoutID = setTimeout(function(){}, delay)
- clearTimeout(timeoutID)

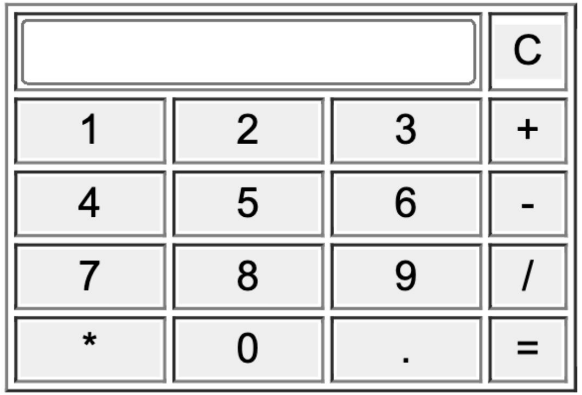
Live coding example

example.html

setTimeout()

- A function in JavaScript to execute a piece of code after a specified delay.
- setTimeout(function(){}, delay)
- Returns an ID, which can be used to cancel the timeout.
- timeoutID = setTimeout(function(){}, delay)
- clearTimeout(timeoutID)

Callback functions



Code:

<button onclick="handleClick()">,

or

document.querySelector(”myBtn").onclic
k = function(event) { … }

or

myBtn.addEventListener(“click”,
function(event));

AJAX (Asynchronous JavaScript and XML)

AJAX (Asynchronous JavaScript and XML)

- Allows a webpage to communicate with a server and updates parts of a web page without reloading the whole page.
- An event is triggered (e.g., a button is clicked)
- JavaScript creates an XMLHttpRequest object and sends a request to the web server.
- The server processes the request and sends a response back to the web page.
- The JavaScript on the web page reads the response and updates the page.

AJAX (Asynchronous JavaScript and XML)

- Allows a webpage to communicate with a server and updates parts of a web page without reloading the whole page.
- An event is triggered (e.g., a button is clicked)
- JavaScript creates an XMLHttpRequest object and sends a request to the web server.
- The server processes the request and sends a response back to the web page.
- The JavaScript on the web page reads the response and updates the page.
- **When the server is processing the request, the browser can do other stuff async.**

Live coding example

ajaxexample.html

Promise

- A powerful way to complete asynchronous tasks in JavaScript
- There are three states for a Promise
    - Pending: initial state
    - Fulfilled: when the operation was completed successfully
    - Rejected: when the operation failed

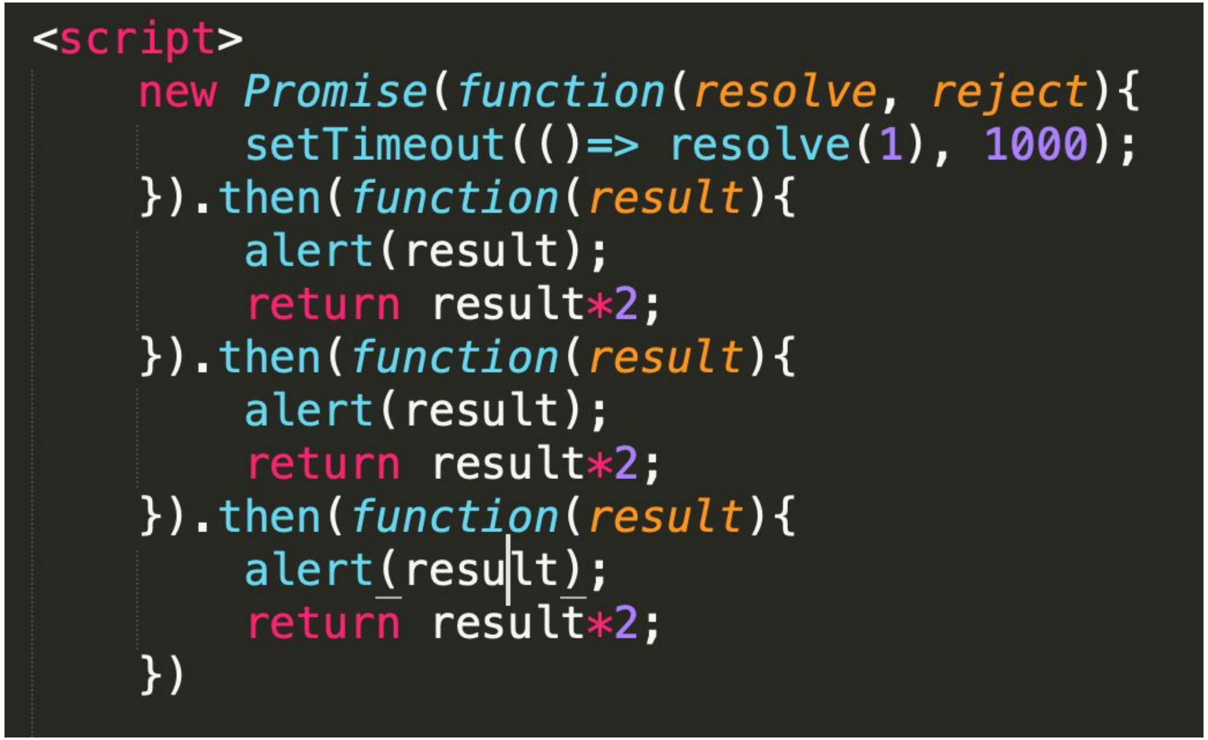
Live coding example

promise.html

Promise Chaining

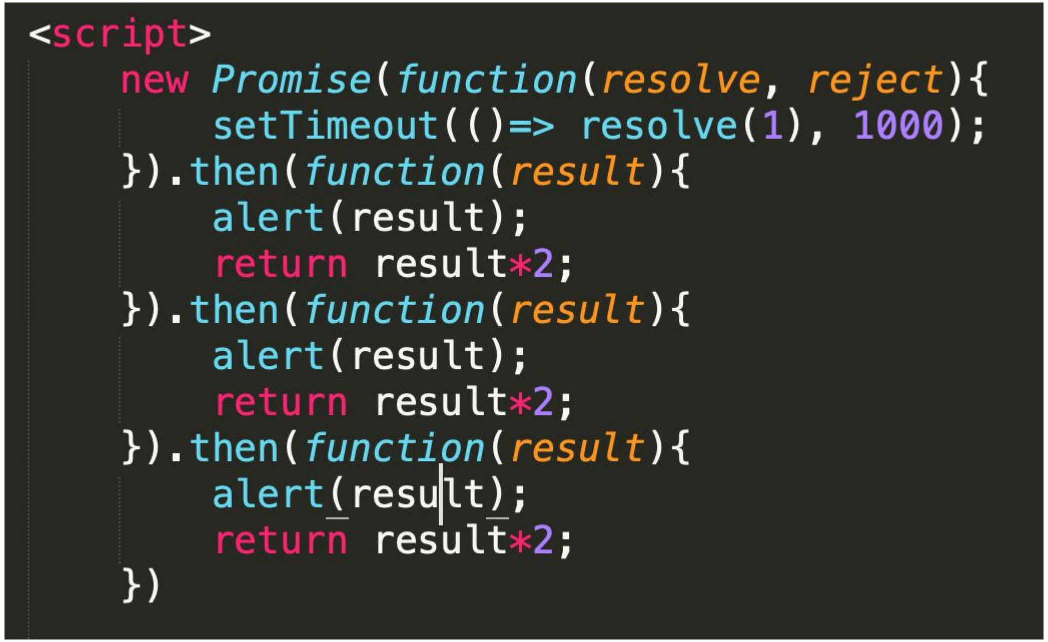
- Promise.then returns a promise

promisechaining.html

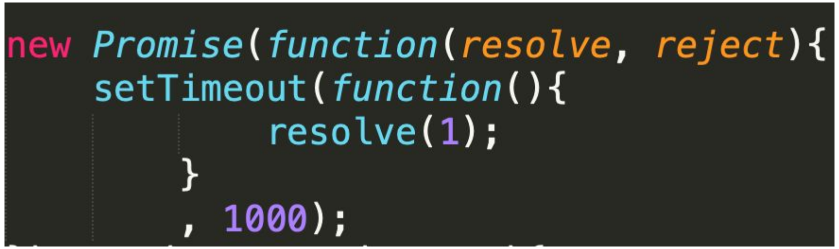


```
<script>
    new Promise(function(resolve, reject){
        setTimeout(()=> resolve(1), 1000);
    }).then(function(result){
        alert(result);
        return result*2;
    }).then(function(result){
        alert(result);
        return result*2;
    }).then(function(result){
        alert(result);
        return result*2;
    })
```

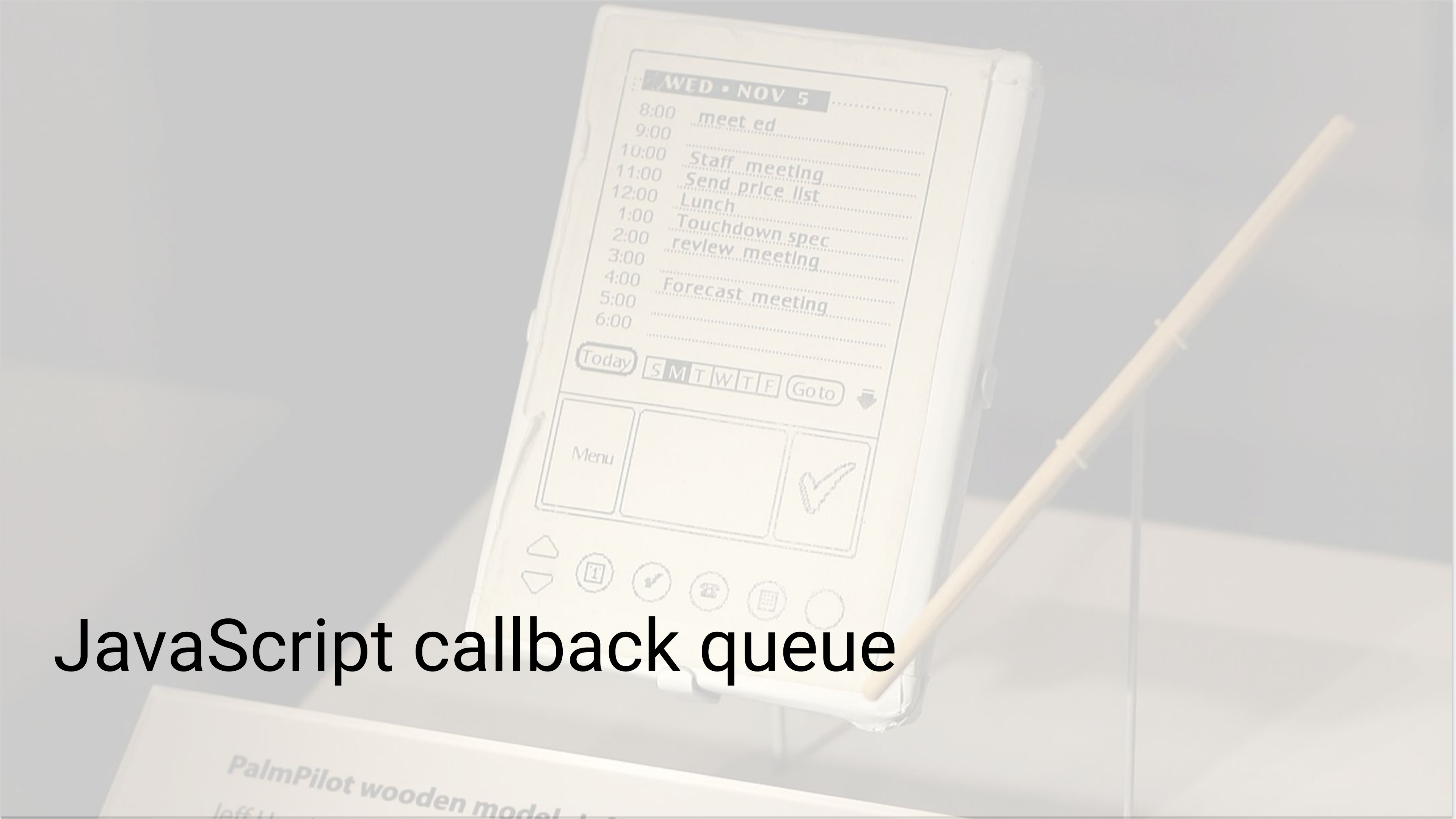
Promise Chaining - arrow functions



```
<script>
    new Promise(function(resolve, reject){
        setTimeout(()=> resolve(1), 1000);
    }).then(function(result){
        alert(result);
        return result*2;
    }).then(function(result){
        alert(result);
        return result*2;
    }).then(function(result){
        alert(result);
        return result*2;
    })
```

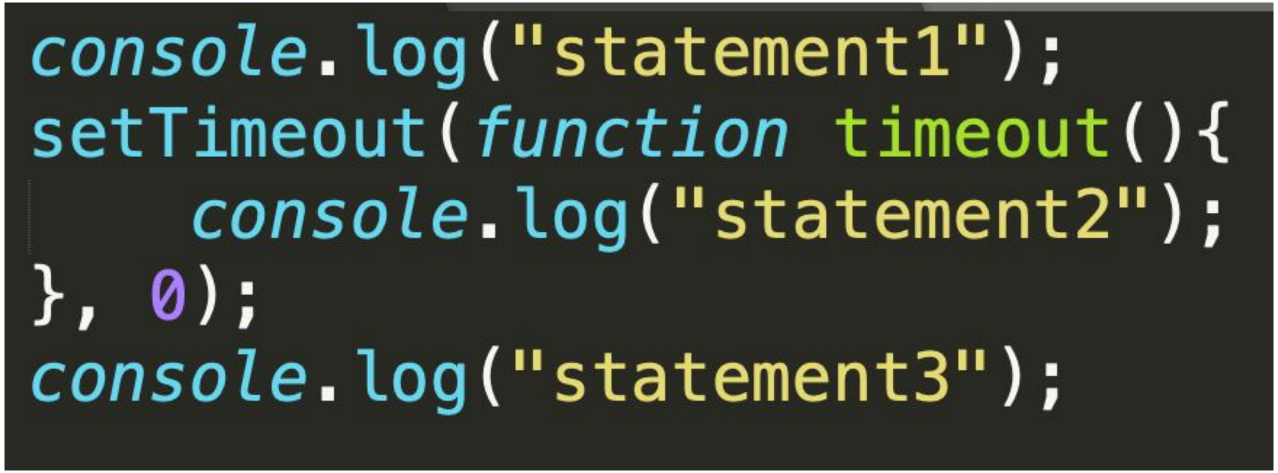


```
new Promise(function(resolve, reject){
    setTimeout(function(){
            resolve(1);
        }
        , 1000);
```



JavaScript callback queue

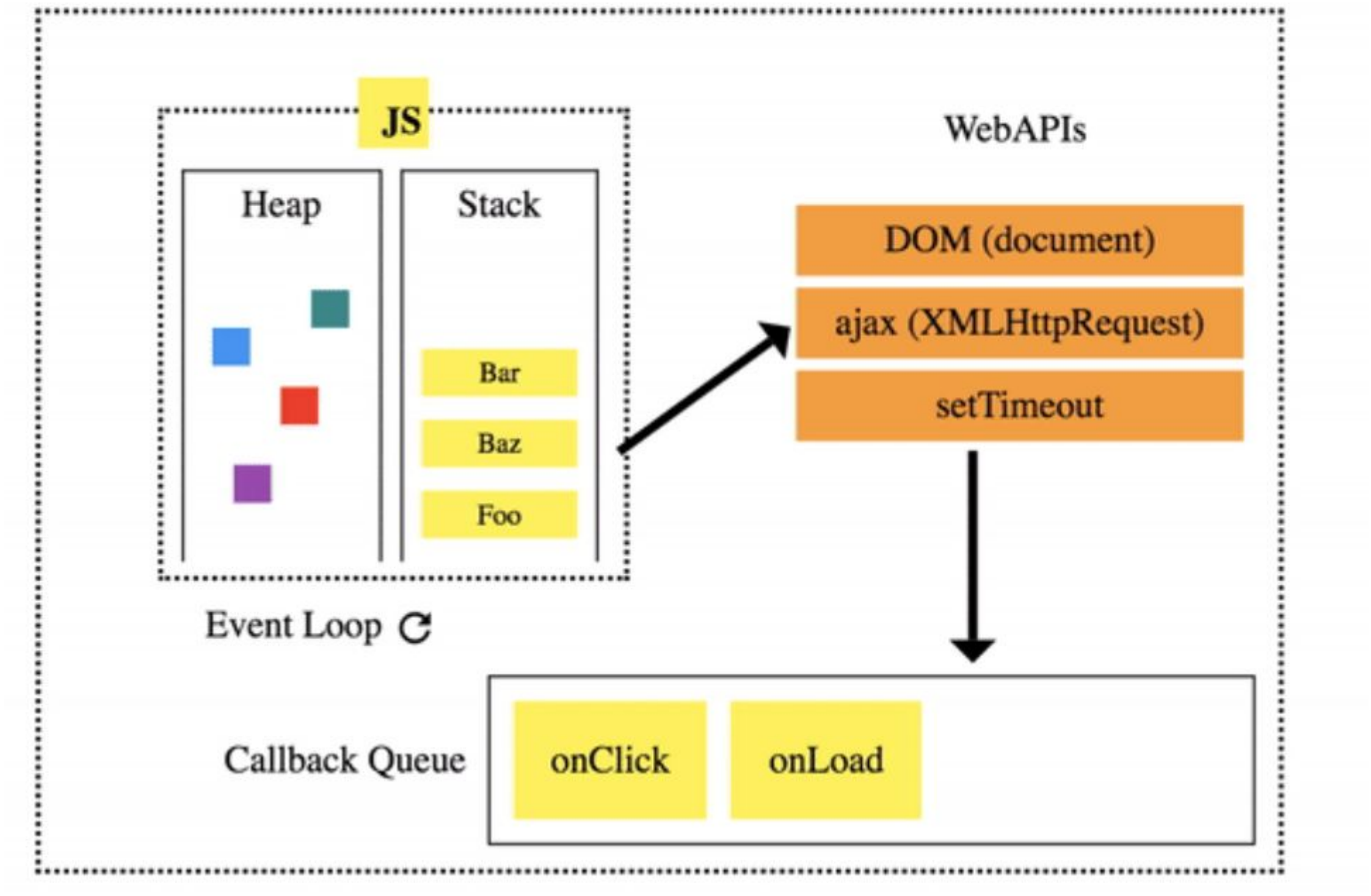
Predict the output



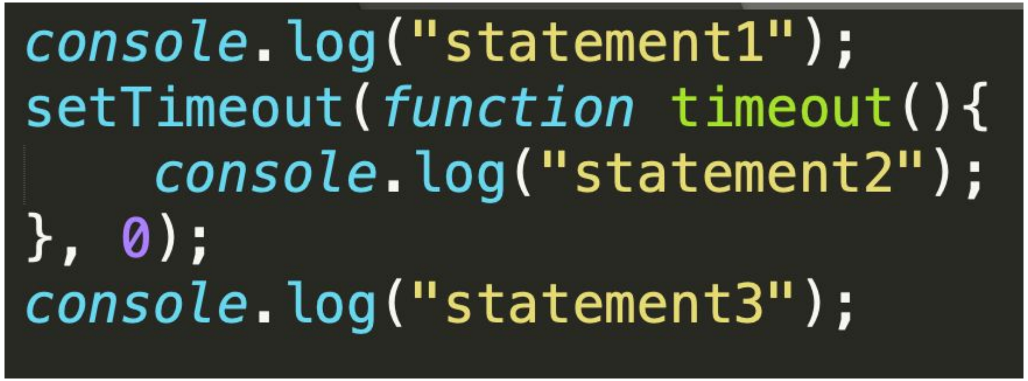
```
console.log("statement1");
setTimeout(function timeout(){
    console.log("statement2");
}, 0);
console.log("statement3");
```

Show it in console

How JavaScript does async programming?

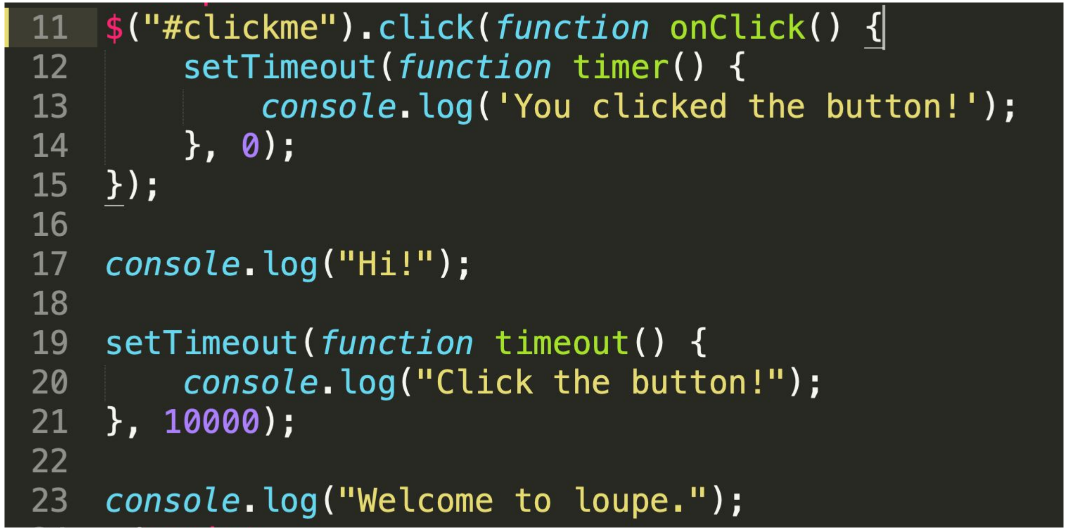


Visualize in Loupe



```
console.log("statement1");
setTimeout(function timeout(){
    console.log("statement2");
}, 0);
console.log("statement3");
```

http://latentflip.com/loupe/

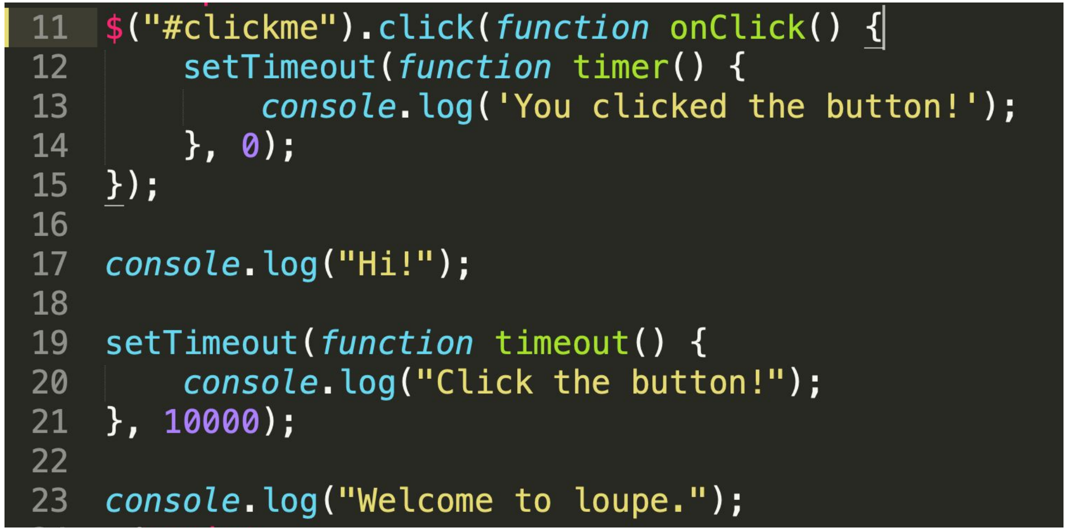


```
$("#clickme").click(function onClick() {
    setTimeout(function timer() {
        console.log('You clicked the button!');
    }, 0);
});

console.log("Hi!");

setTimeout(function timeout() {
    console.log("Click the button!");
}, 10000);

console.log("Welcome to loupe.");
```

Console
- Hi
- Welcome to Loupe

Callback Queue

WebAPIs
- timer of 10 seconds for timeout to enter the callback queue.



```
$("#clickme").click(function onClick() {
    setTimeout(function timer() {
        console.log('You clicked the button!');
    }, 0);
});

console.log("Hi!");

setTimeout(function timeout() {
    console.log("Click the button!");
}, 10000);

console.log("Welcome to loupe.");
```

Console

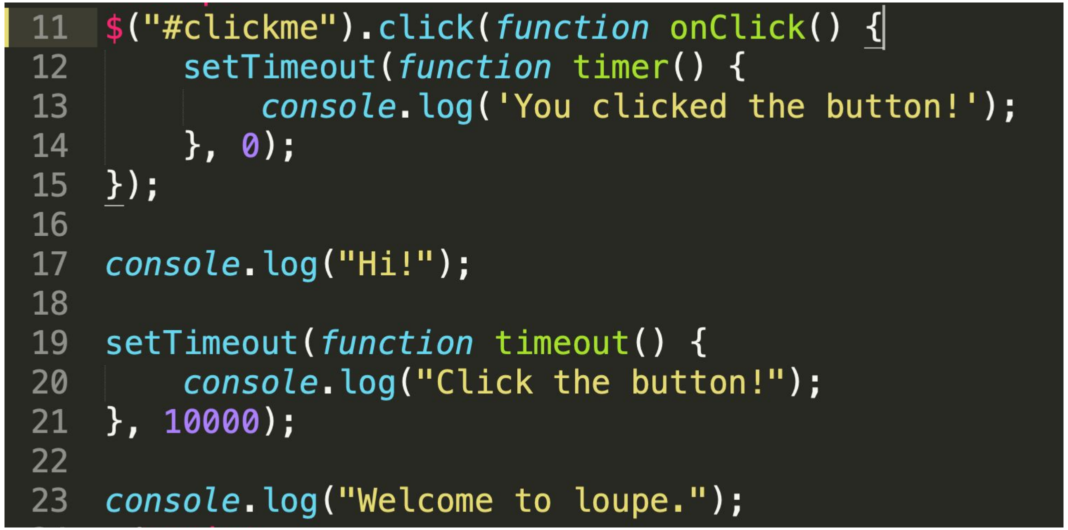
- Hi
- Welcome to Loupe

Callback Queue

- onClick()

WebAPIs

- timer of 10 seconds for timeout to enter the callback queue.
- timer of 0 seconds for timer() to enter the callback queue.

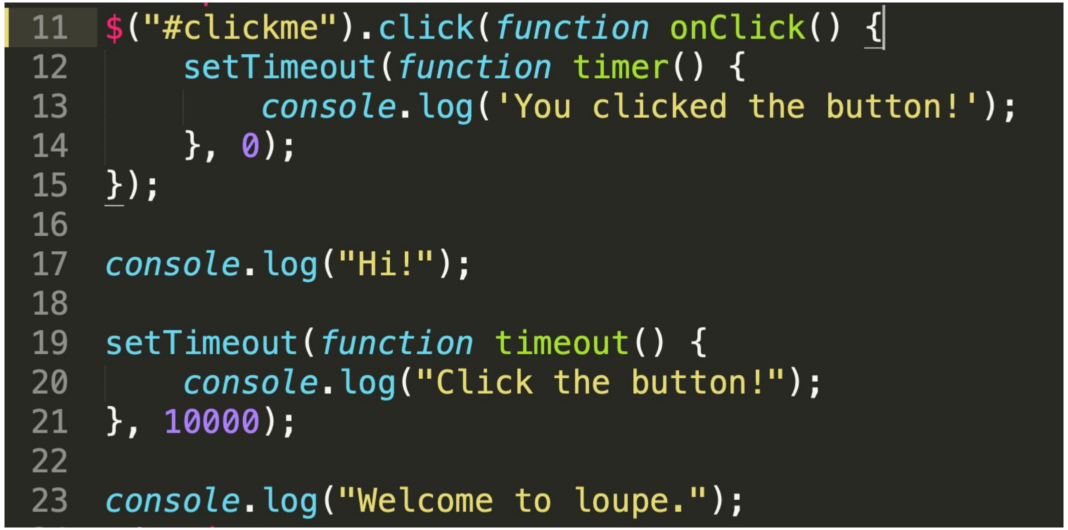


```
$("#clickme").click(function onClick() {
    setTimeout(function timer() {
        console.log('You clicked the button!');
    }, 0);
});

console.log("Hi!");

setTimeout(function timeout() {
    console.log("Click the button!");
}, 10000);

console.log("Welcome to loupe.");
```

Console
- Hi
- Welcome to Loupe
- You clicked the button!

Callback Queue
- onClick()
- timer()

WebAPIs
- timer of 10 seconds for timeout to enter the callback queue.



```
$("#clickme").click(function onClick() {
    setTimeout(function timer() {
        console.log('You clicked the button!');
    }, 0);
});

console.log("Hi!");

setTimeout(function timeout() {
    console.log("Click the button!");
}, 10000);

console.log("Welcome to loupe.");
```

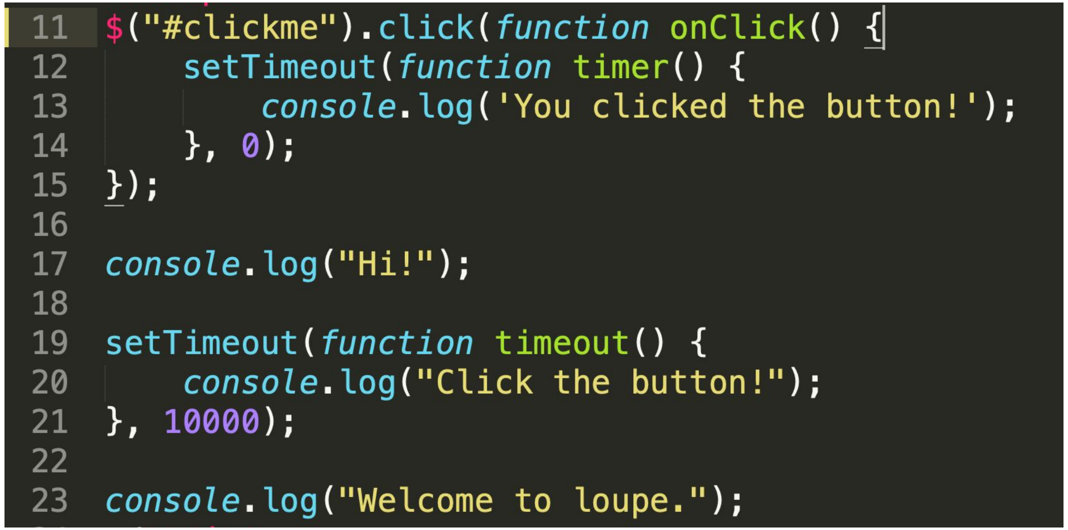
Console

- Hi
- Welcome to Loupe
- You clicked the button!

WebAPIs

Callback Queue

- onClick()
- timer()
- timeout()



```
$("#clickme").click(function onClick() {
    setTimeout(function timer() {
        console.log('You clicked the button!');
    }, 0);
});

console.log("Hi!");

setTimeout(function timeout() {
    console.log("Click the button!");
}, 10000);

console.log("Welcome to loupe.");
```

Console

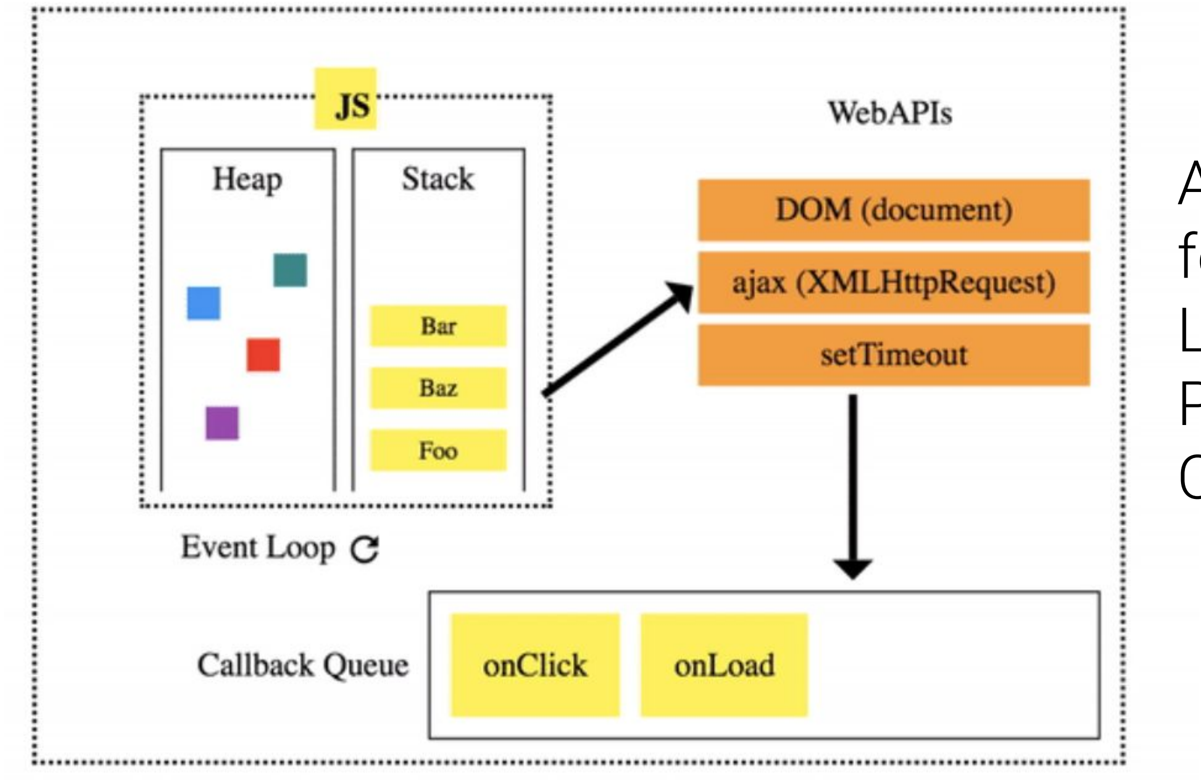
- Hi
- Welcome to Loupe
- You clicked the button!
- Click the button!

WebAPIs

Callback Queue

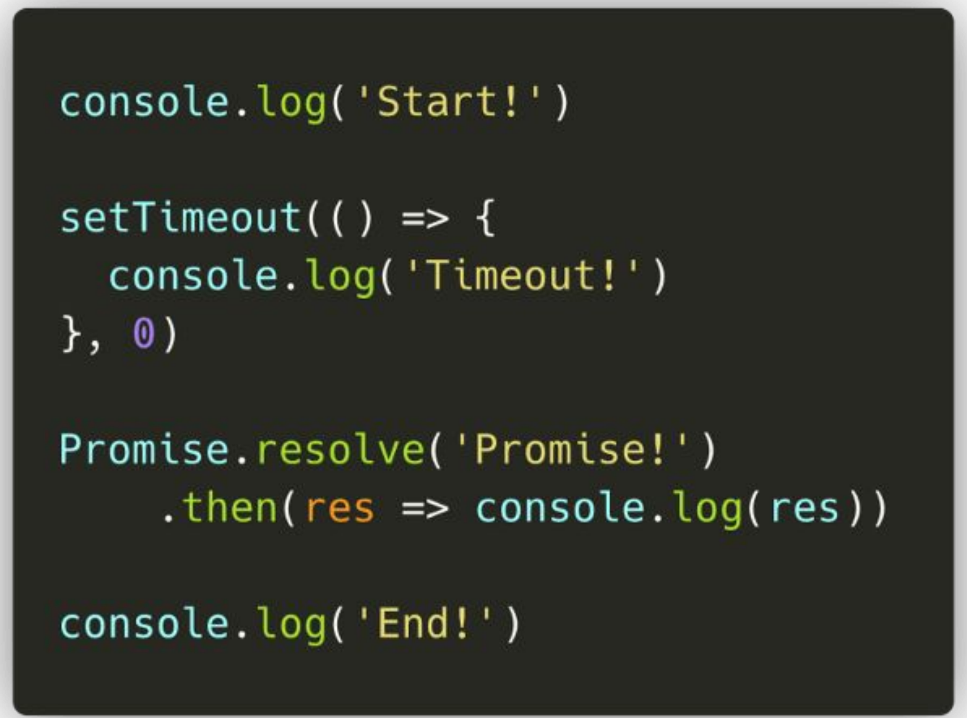
- onClick()
- timer()
- timeout()

Promises -> additional job queue

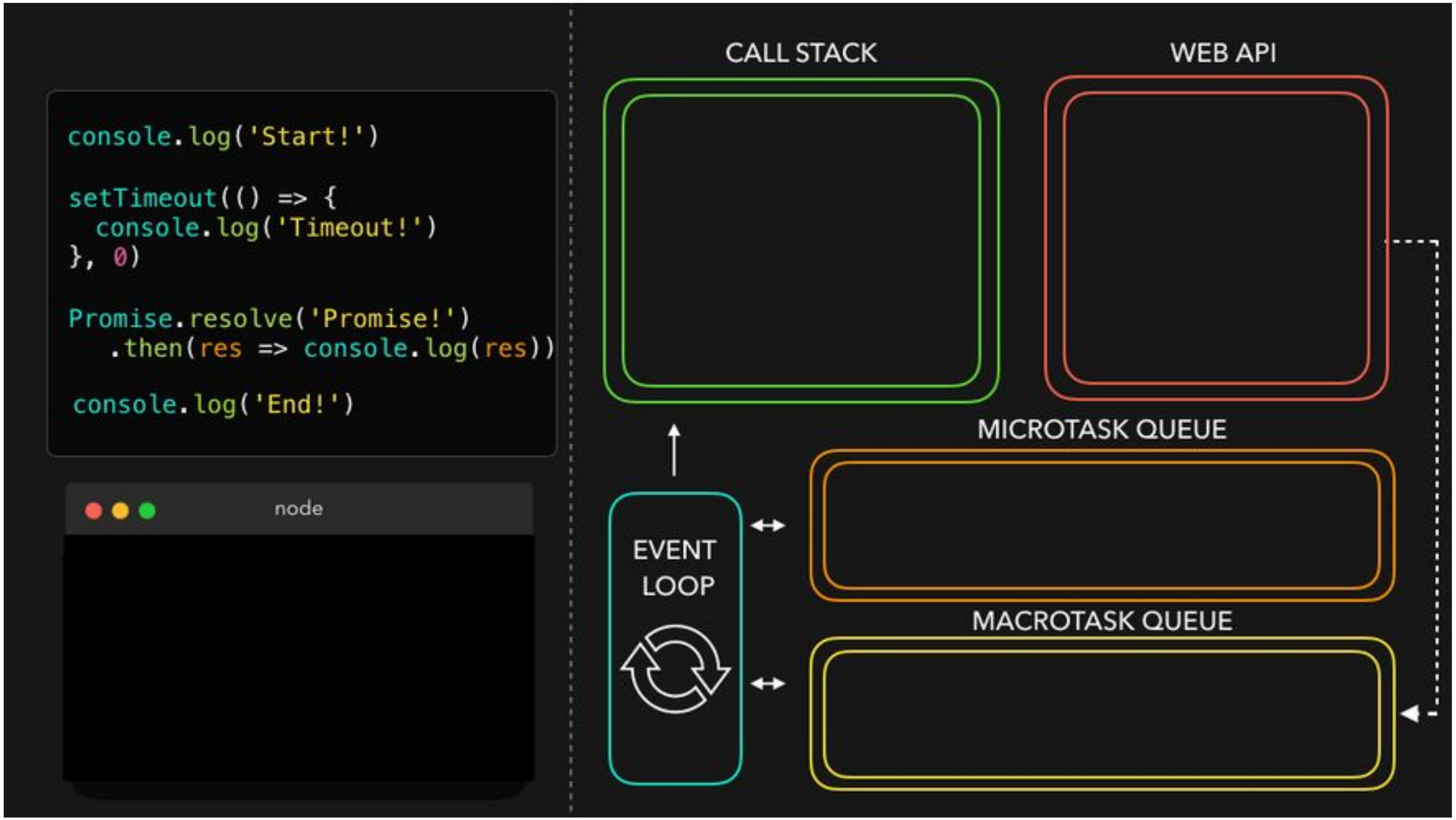


Additional job Queue for Promises: Event Loop prioritize Promises over Callback Queues

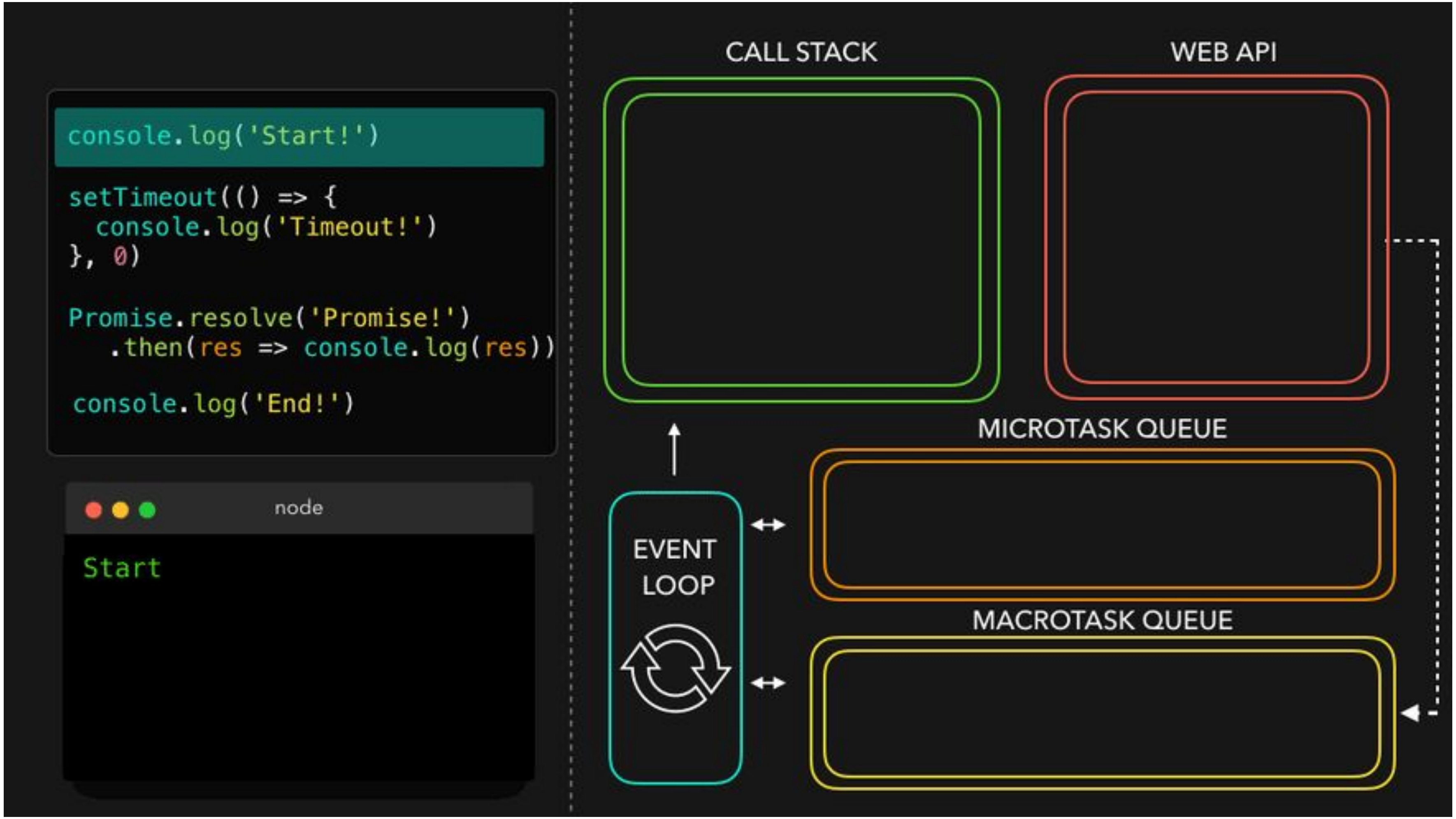
What is the order of output?



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

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
    .then(res => console.log(res))

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

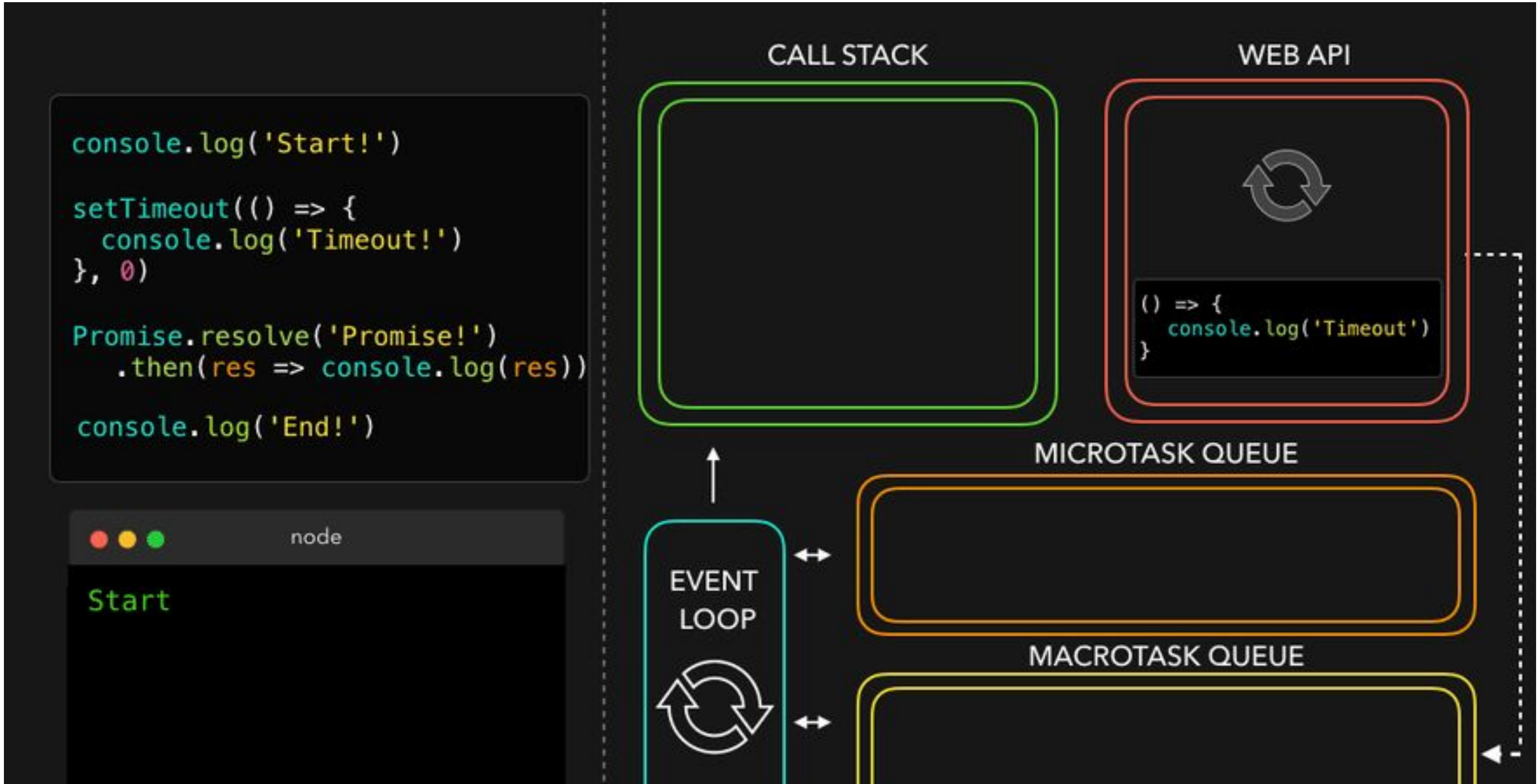


```
console.log('Start!')

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
   .then(res => console.log(res))

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



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

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
   .then(res => console.log(res))

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

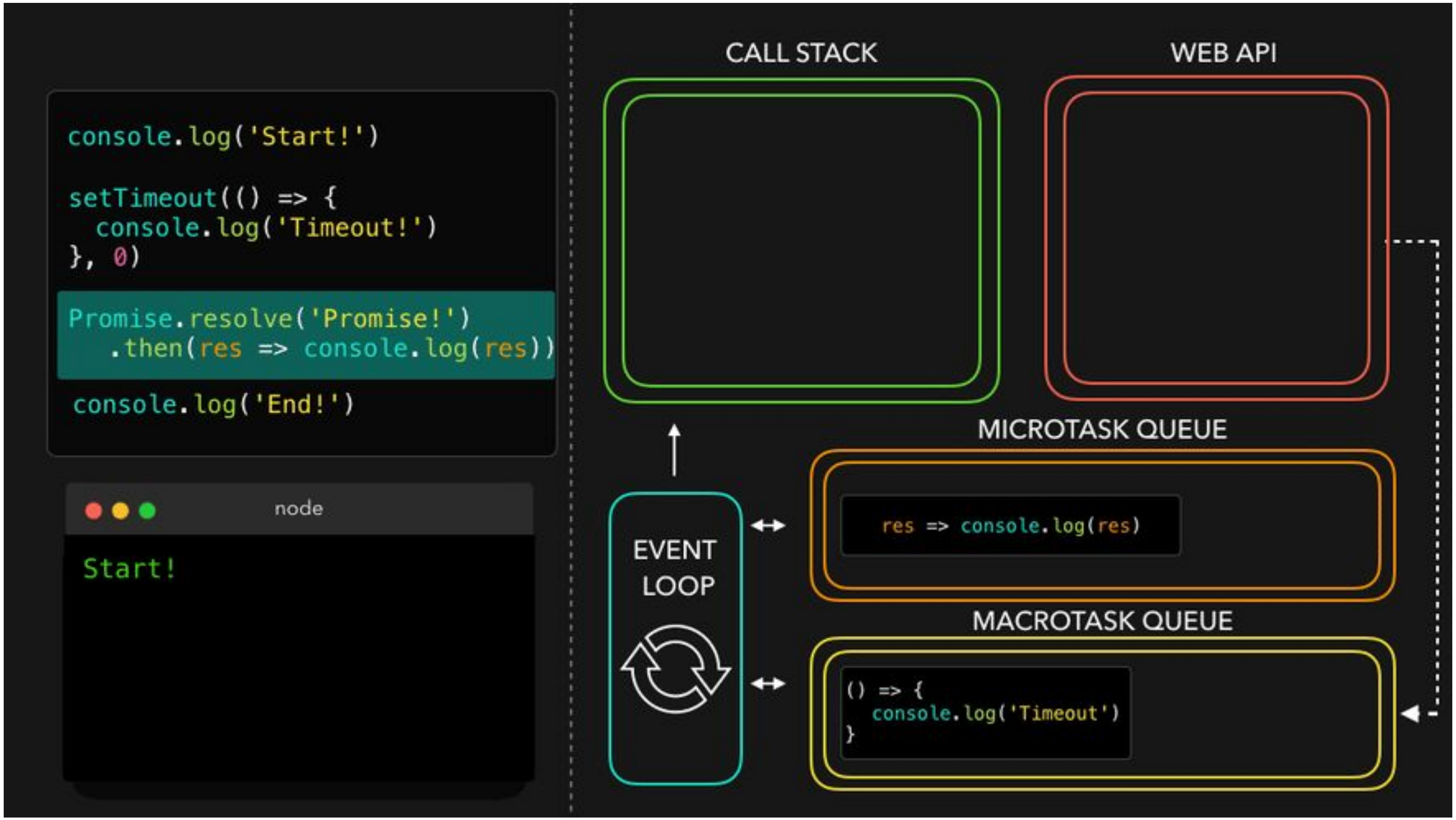
```
Start
```



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

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
   .then(res => console.log(res))

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

```
Start
```

```
() => {
  console.log('Timeout')
}
```

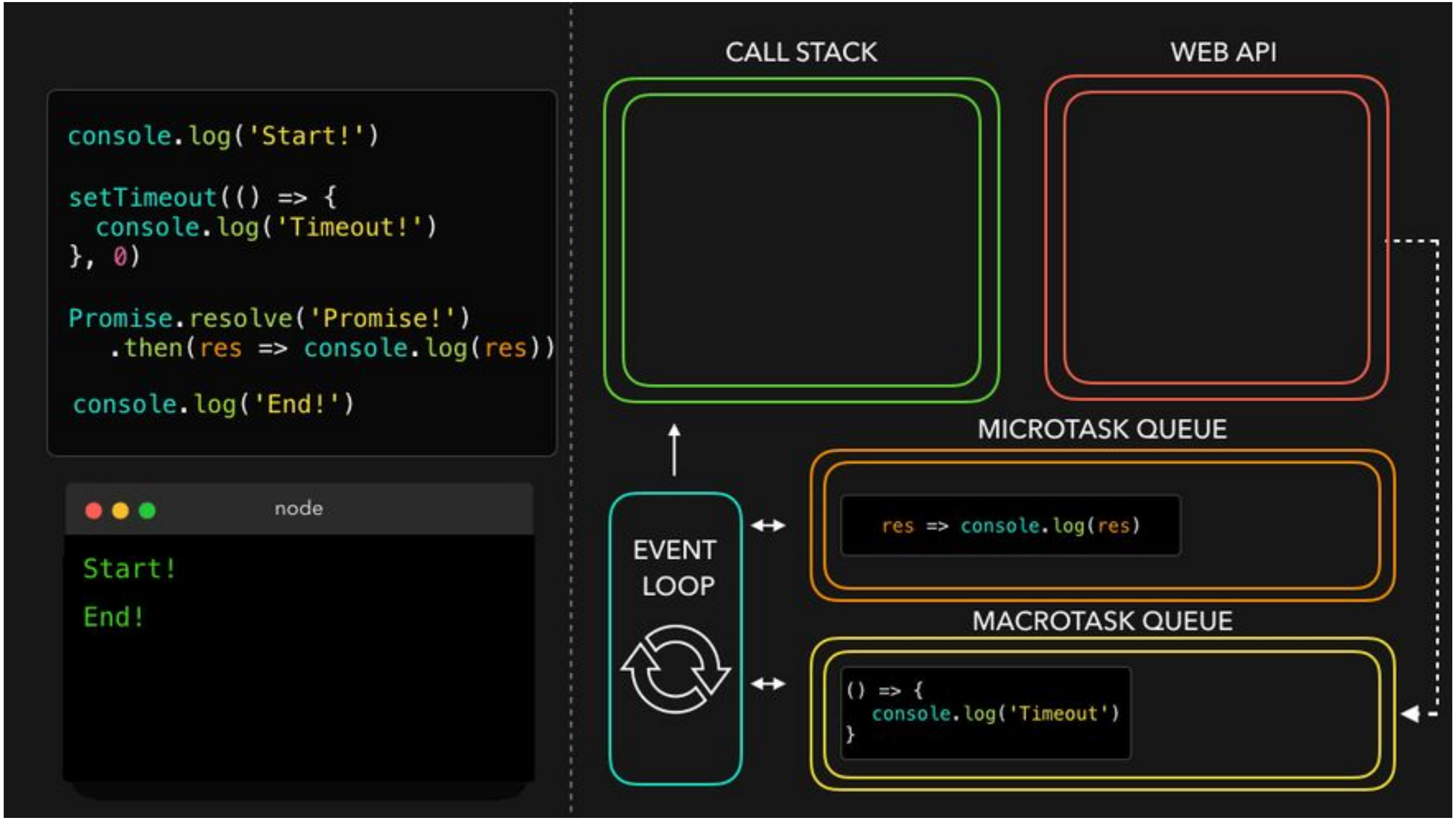


```
console.log('Start!')

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
   .then(res => console.log(res))

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

```
Start!
```

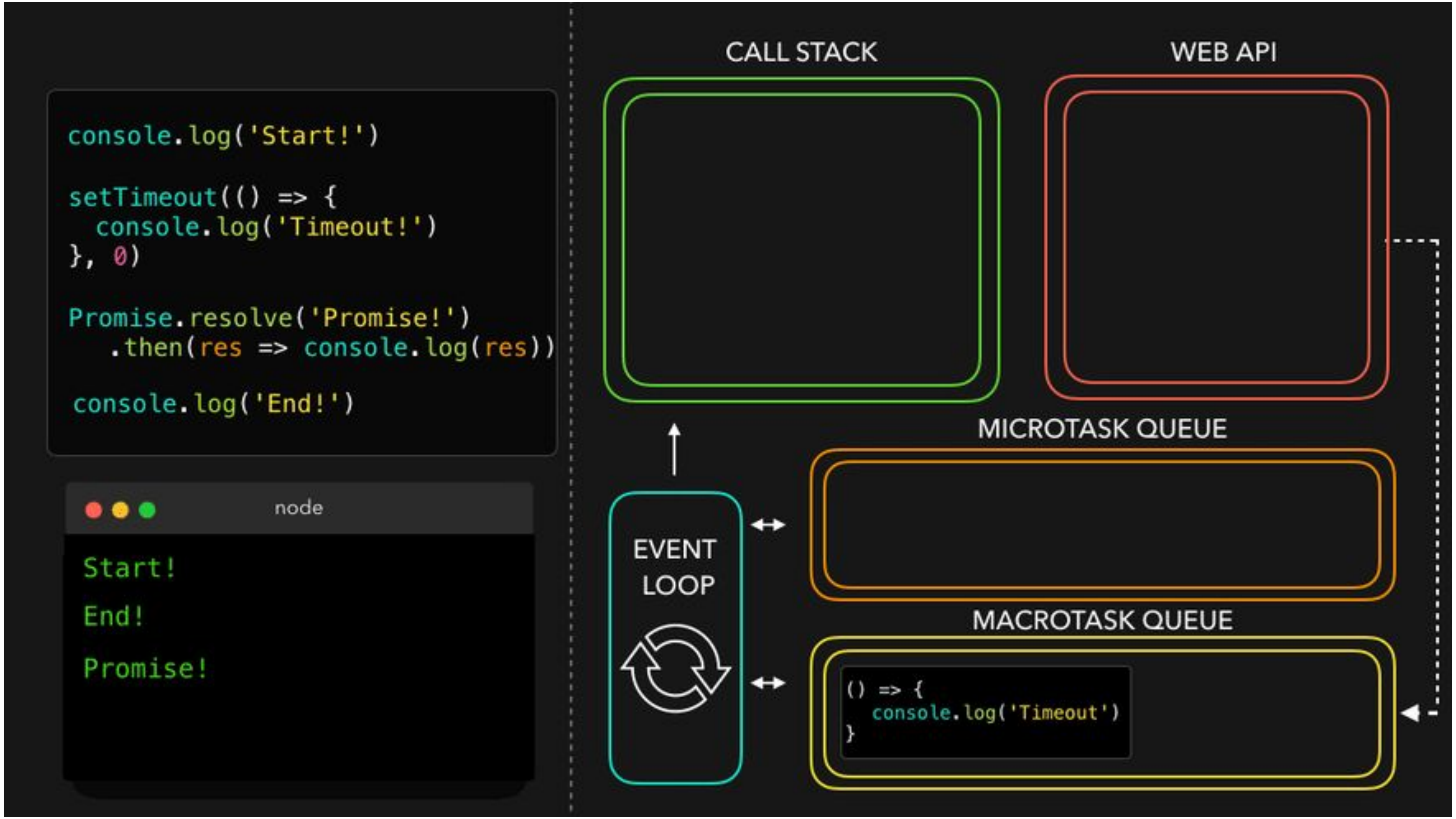
```
res => console.log(res)
```

```
() => {
  console.log('Timeout')
}
```



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

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
   .then(res => console.log(res))

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

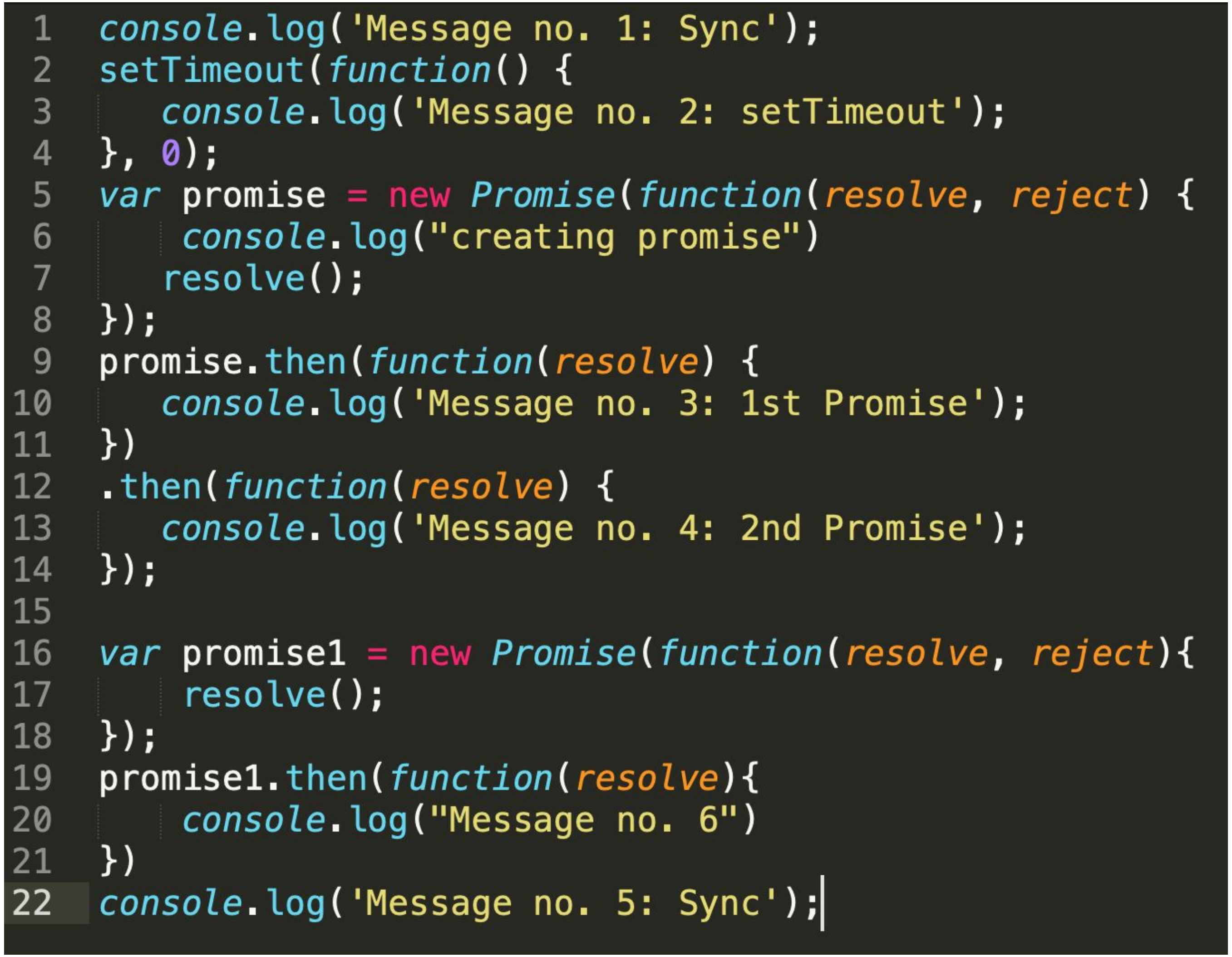
```
Start!
End!
```



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

setTimeout(() => {
  console.log('Timeout!')
}, 0)

Promise.resolve('Promise!')
   .then(res => console.log(res))

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

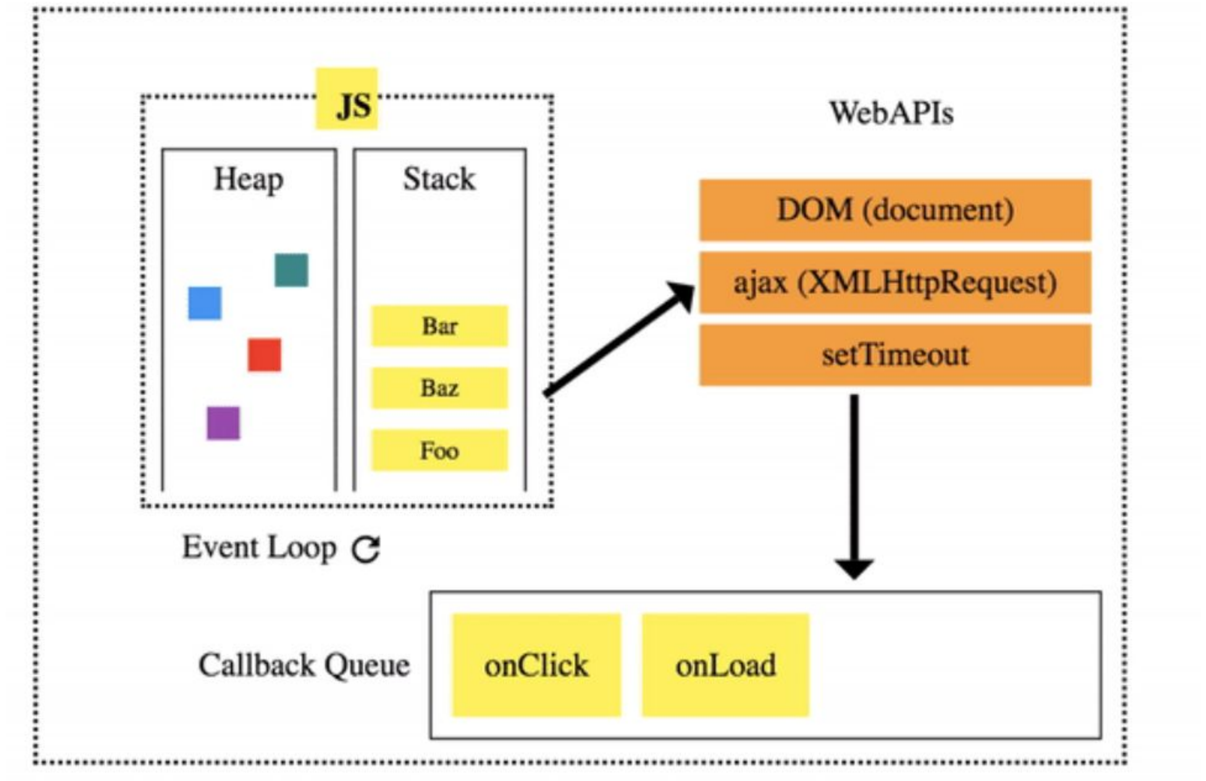
```
Start!
End!
Promise!
```

```
() => {
  console.log('Timeout')
}
```



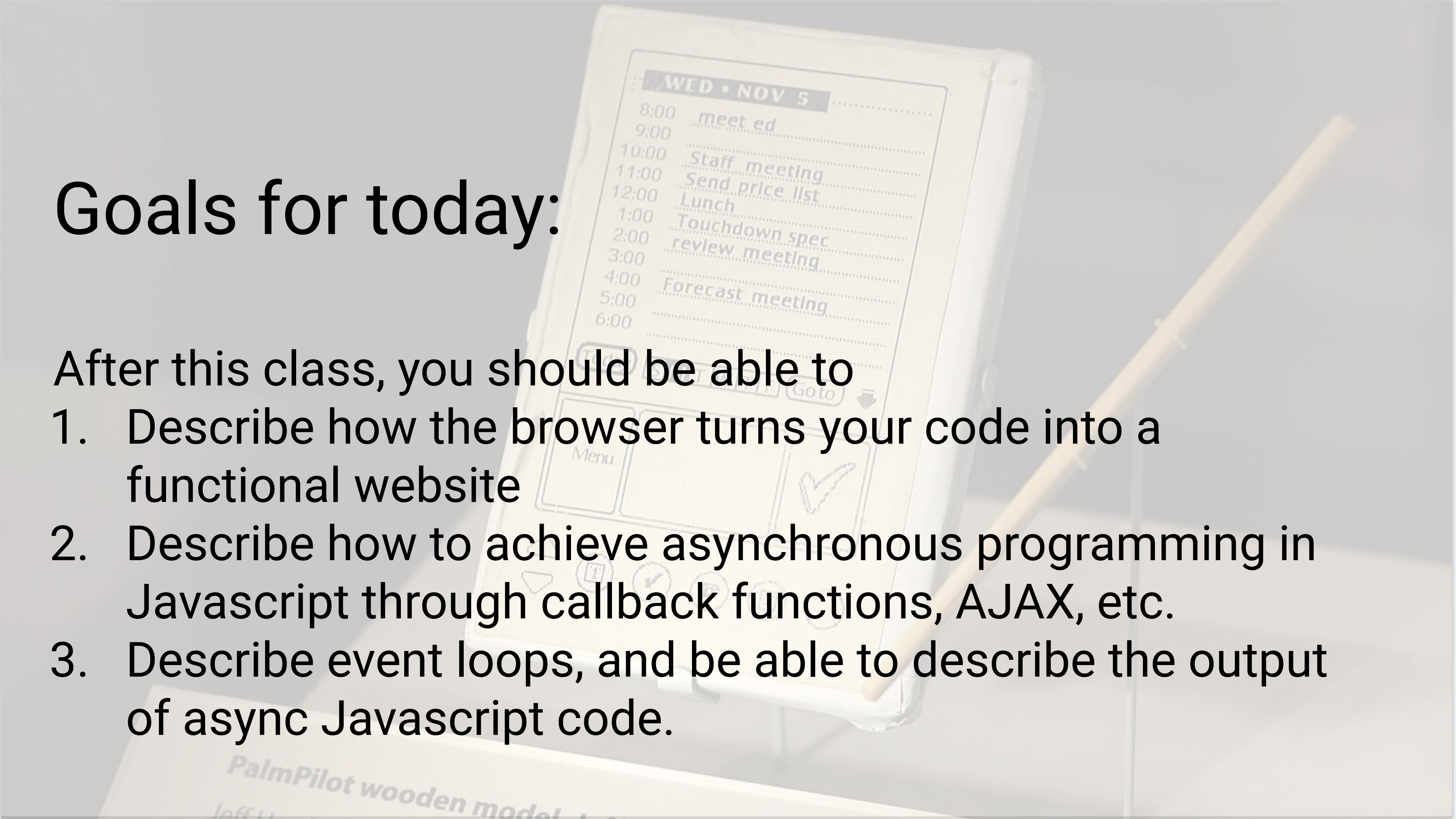
```javascript
console.log('Message no. 1: Sync');
setTimeout(function() {
   console.log('Message no. 2: setTimeout');
}, 0);
var promise = new Promise(function(resolve, reject) {
    console.log("creating promise")
   resolve();
});
promise.then(function(resolve) {
   console.log('Message no. 3: 1st Promise');
})
.then(function(resolve) {
   console.log('Message no. 4: 2nd Promise');
});

var promise1 = new Promise(function(resolve, reject){
    resolve();
});
promise1.then(function(resolve){
    console.log("Message no. 6")
})
console.log('Message no. 5: Sync');
```

Recap - JavaScript Event Loop



Recap - JavaScript Event Loop

- Event Loop is constantly running to check the stack and the callback queues. When the stack is empty, it pushes functions from callback queues to stack.
- setTimeout() - 10 seconds
  - At least there's a wait of 10 seconds, possibly it'll be executed in more than 10 seconds

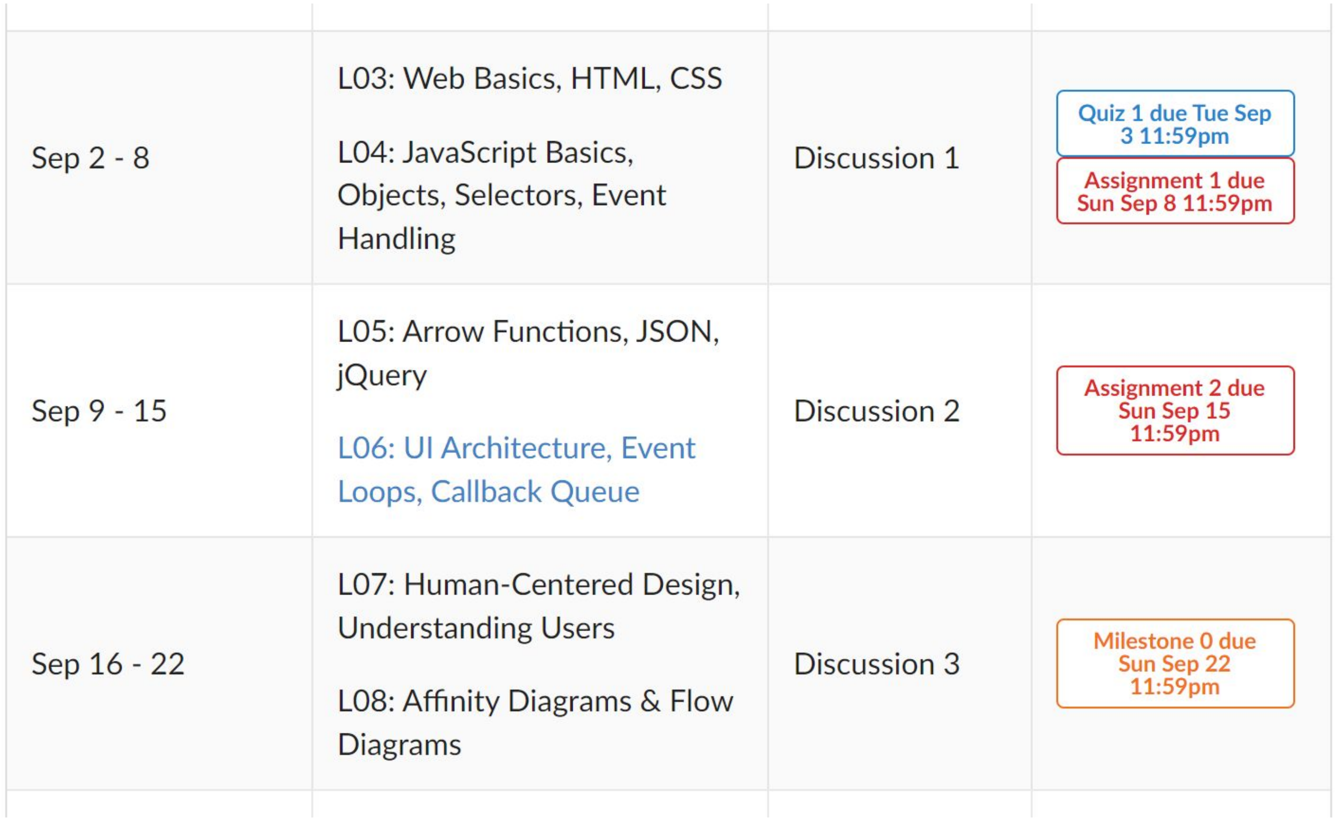


Goals for today:

After this class, you should be able to

1. Describe how the browser turns your code into a functional website
2. Describe how to achieve asynchronous programming in Javascript through callback functions, AJAX, etc.
3. Describe event loops, and be able to describe the output of async Javascript code.

Where we stand in class…



Sep 2 - 8	L03: Web Basics, HTML, CSS L04: JavaScript Basics, Objects, Selectors, Event Handling	Discussion 1	Quiz 1 due Tue Sep 3 11:59pm Assignment 1 due Sun Sep 8 11:59pm
Sep 9 - 15	L05: Arrow Functions, JSON, jQuery L06: UI Architecture, Event Loops, Callback Queue	Discussion 2	Assignment 2 due Sun Sep 15 11:59pm
Sep 16 - 22	L07: Human-Centered Design, Understanding Users L08: Affinity Diagrams & Flow Diagrams	Discussion 3	Milestone 0 due Sun Sep 22 11:59pm