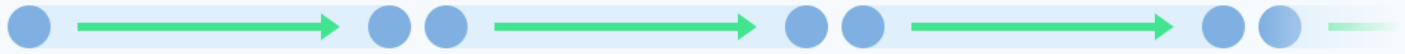
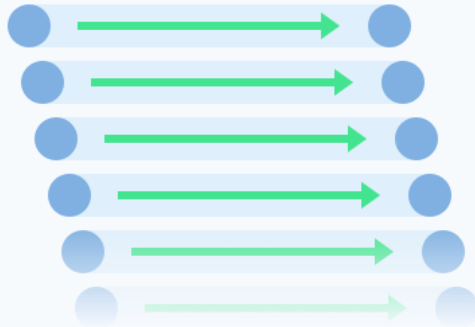


Javascript Asincrono

Synchronous



Asynchronous



Codice sincrono



```
const modal = document.querySelector('.modal');
modal.style.backgroundColor = 'red';
let val = confirm('Show Modal?');
if (val) modal.classList.add('show');
```

Codice sincrono



```
const modal = document.querySelector('.modal');
modal.style.backgroundColor = 'red';
let val = confirm('Show Modal?');
if (val) modal.classList.add('show');
```

Codice sincrono

Bloccante



```
const modal = document.querySelector('.modal');
modal.style.backgroundColor = 'red';
let val = confirm('Show Modal?');
if (val) modal.cl
```


127.0.0.1:5500 says

Show Modal?

Cancel

OK

Codice sincrono



```
const modal = document.querySelector('.modal');
modal.style.backgroundColor = 'red';
let val = confirm('Show Modal?');
if (val) modal.classList.add('show');
```

- Il codice è eseguito linea dopo linea
- Ogni linea aspetta che finisce l'esecuzione della precedente
- Le operazioni lunghe bloccano l'esecuzione del programma

Codice Asincrono




```
const modal = document.querySelector('.modal');
setTimeout(function () {
  modal.classList.add('show');
}, 2000);
modal.style.backgroundColor = 'red';
```

Codice Asincrono

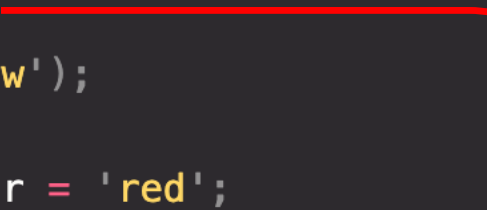


```
const modal = document.querySelector('.modal');
setTimeout(function () {
  modal.classList.add('show');
}, 2000);
modal.style.backgroundColor = 'red';
```


Codice Asincrono



```
const modal = document.querySelector('.modal');
setTimeout(function () {
  modal.classList.add('show');
}, 2000);
modal.style.backgroundColor = 'red';
```



Background tasks



- La funzione è eseguita alla fine dell'esecuzione del task
- Il codice sincrono continua la sua esecuzione

Codice Asincrono

```
const modal = document.querySelector('.modal');
setTimeout(function () {
  modal.classList.add('show');
}, 2000);
modal.style.backgroundColor = 'red';
```



Background tasks



- La funzione è eseguita alla fine dell'esecuzione del task
- Il codice sincrono continua la sua esecuzione

Codice Asincrono



```
const modal = document.querySelector('.modal');
setTimeout(function () {
  modal.classList.add('show');
}, 2000);
modal.style.backgroundColor = 'red';
```

Background tasks



- La funzione è eseguita alla fine dell'esecuzione del task
- Il codice sincrono continua la sua esecuzione

Codice Asincrono

```
const modal = document.querySelector('.modal');
setTimeout(function () {
  modal.classList.add('show');
}, 2000);
modal.style.backgroundColor = 'red';
```

Background tasks




- La funzione è eseguita alla fine dell'esecuzione del task
- Il codice sincrono continua la sua esecuzione

Le callback non rendono il codice asincrono:

```
const buttons = document.querySelectorAll('.btn');
buttons.forEach((el) => (el.style.backgroundColor = 'white'));
```

Codice Asincrono 2



```
const image = document.querySelector('.my-img');
image.src = 'mountain.jpg';
image.addEventListener('load', function () {
  image.classList.add('fadeIn');
});
modal.style.backgroundColor = 'white';
```

Codice Asincrono 2


```
const image = document.querySelector('.my-img');
image.src = 'mountain.jpg';
image.addEventListener('load', function () {
  image.classList.add('fadeIn');
});
modal.style.backgroundColor = 'white';
```

Background tasks



- Le immagini sono caricate in modo asincrono

Codice Asincrono 2



```
const image = document.querySelector('.my-img');
image.src = 'mountain.jpg';
image.addEventListener('load', function () {
  image.classList.add('fadeIn');
});
modal.style.backgroundColor = 'white';
```

Background tasks



- Le immagini sono caricate in modo asincrono
- Il load viene eseguito alla fine del caricamento

Codice Asincrono 2

```
const image = document.querySelector('.my-img');
image.src = 'mountain.jpg';
image.addEventListener('load', function () {
  image.classList.add('fadeIn');
});
modal.style.backgroundColor = 'white';
```



Background tasks



- Le immagini sono caricate in modo asincrono
- Il load viene eseguito alla fine del caricamento

Codice Asincrono 2

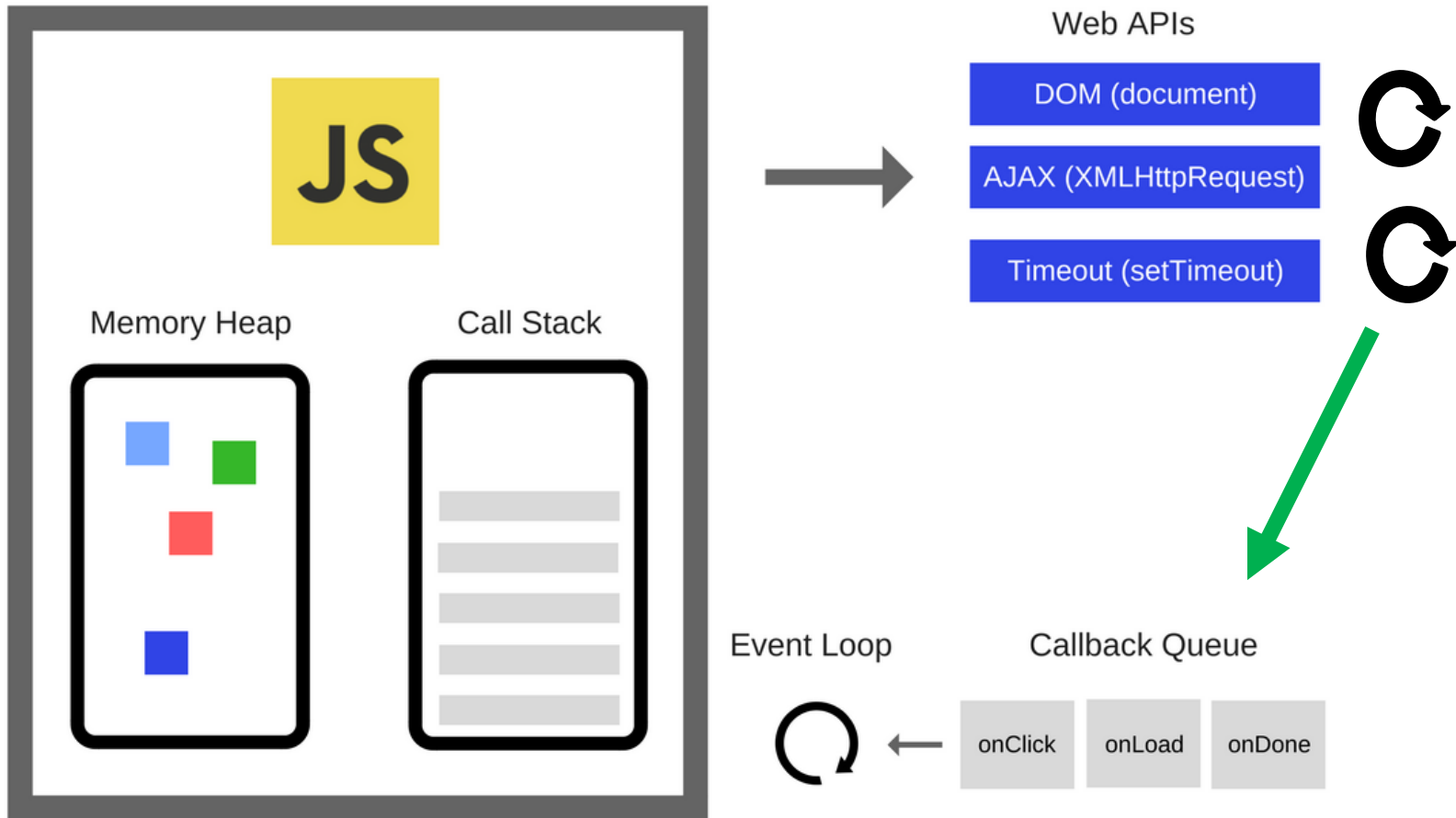
```
const image = document.querySelector('.my-img');
image.src = 'mountain.jpg';
image.addEventListener('load', function () {
    image.classList.add('fadeIn');
});
modal.style.backgroundColor = 'white';
```

Background tasks



- Le immagini sono caricate in modo asincrono
- Il load viene eseguito alla fine del caricamento

Event Loop



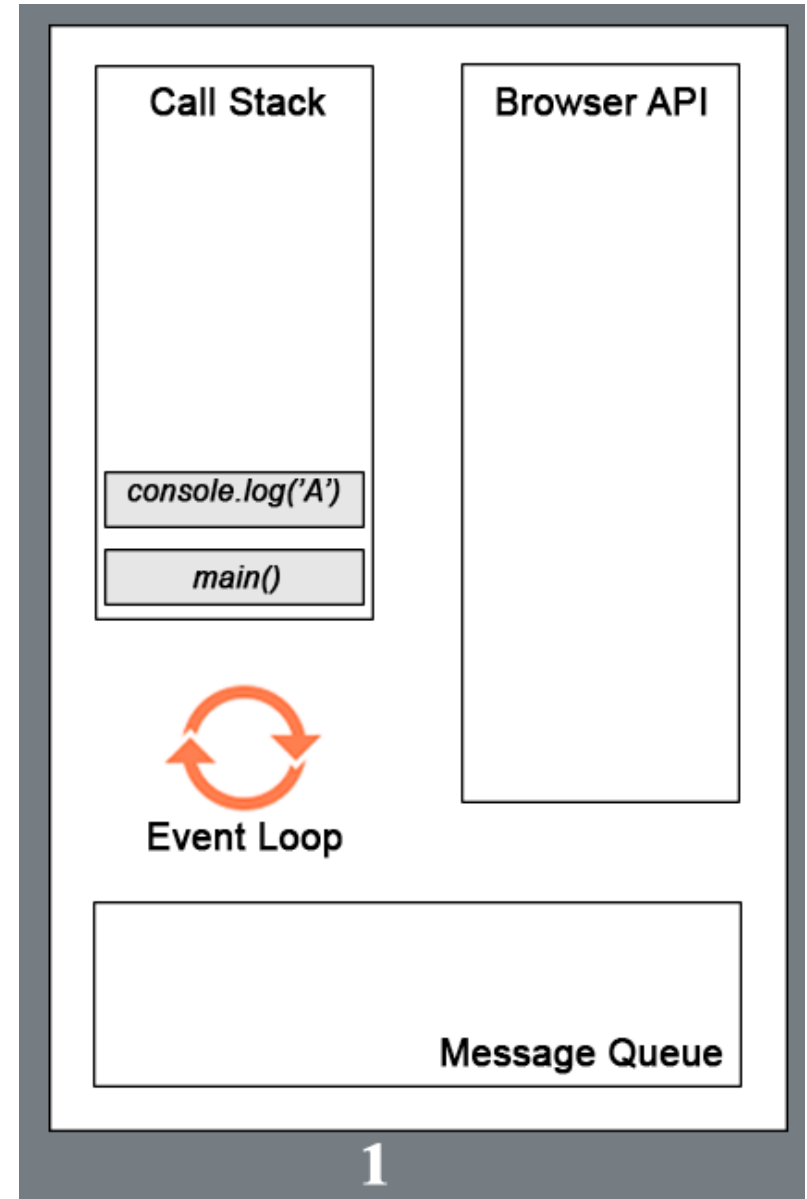
Esempio

```
function main() {
  console.log('A');
  setTimeout(function display() {
    console.log('B');
  }, 0);
  console.log('C');
}
main();
```

<https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4>

Esempio

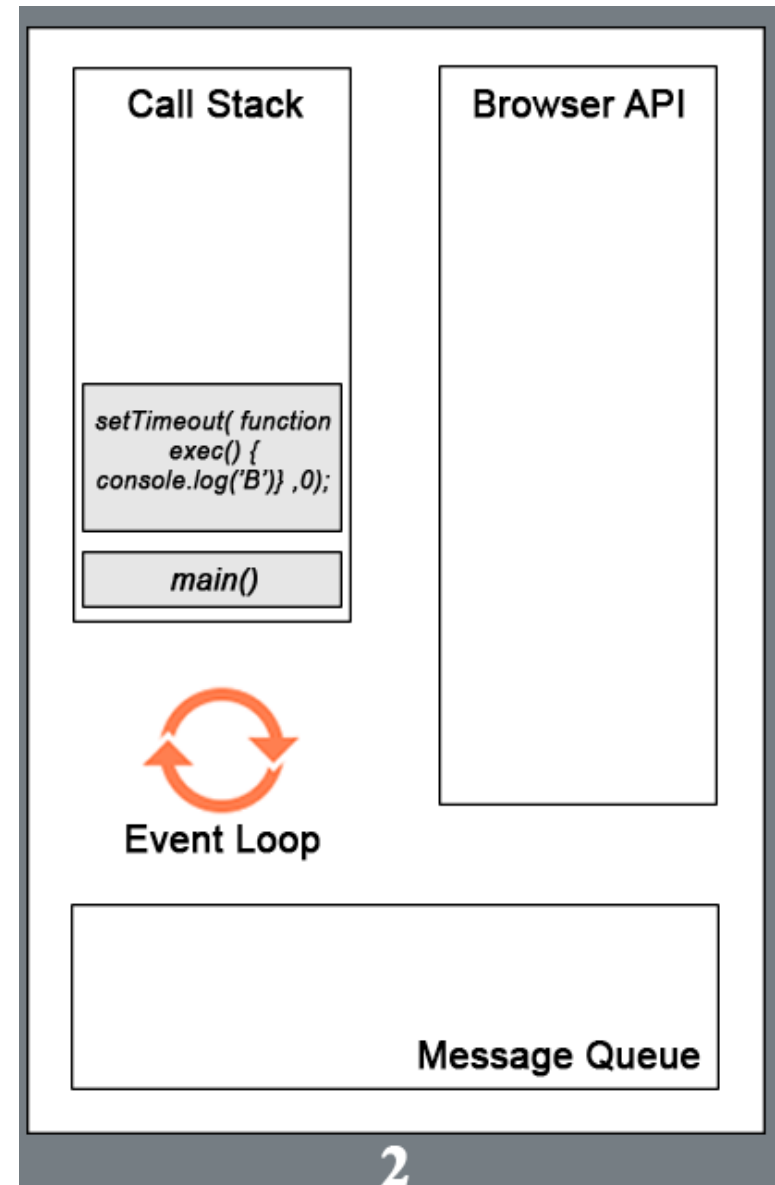
```
function main() {
  console.log('A');
  setTimeout(function display() {
    console.log('B');
  }, 0);
  console.log('C');
}
main();
```



<https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4>

Esempio

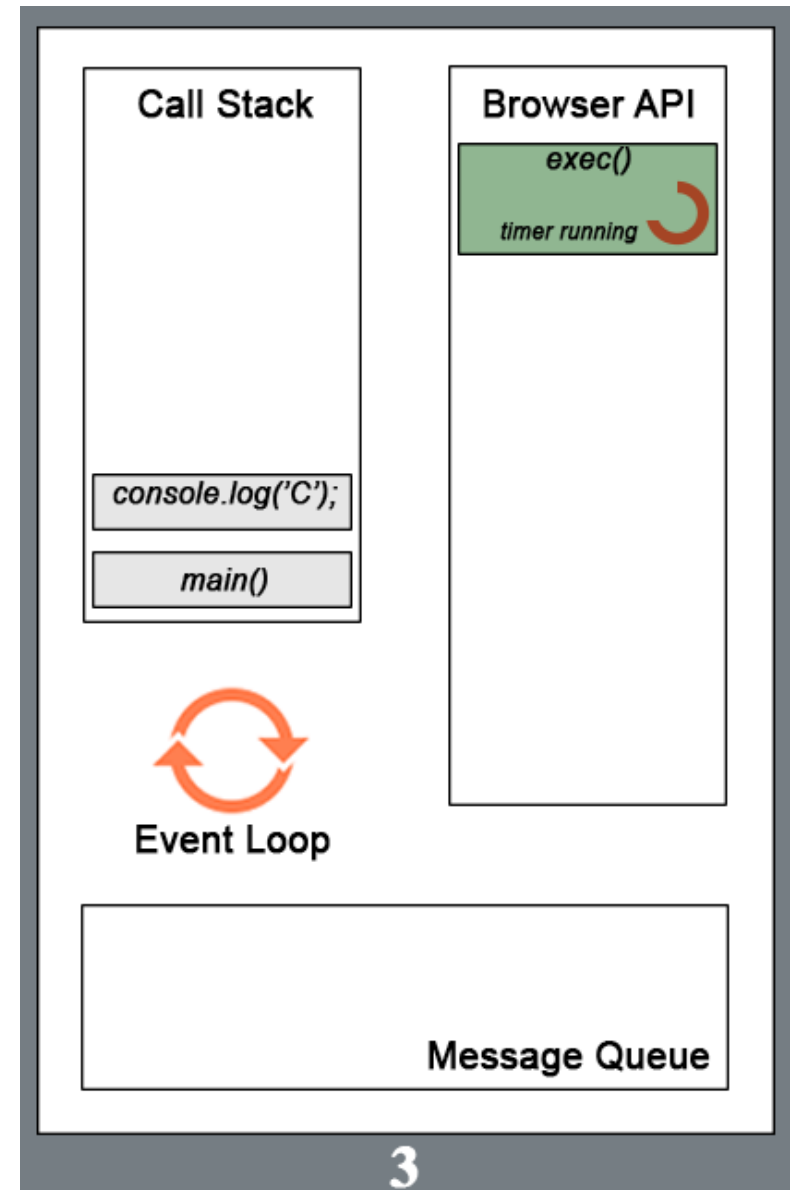
```
function main() {
  console.log('A');
  setTimeout(function display() {
    console.log('B');
  }, 0);
  console.log('C');
}
main();
```



<https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4>

Esempio

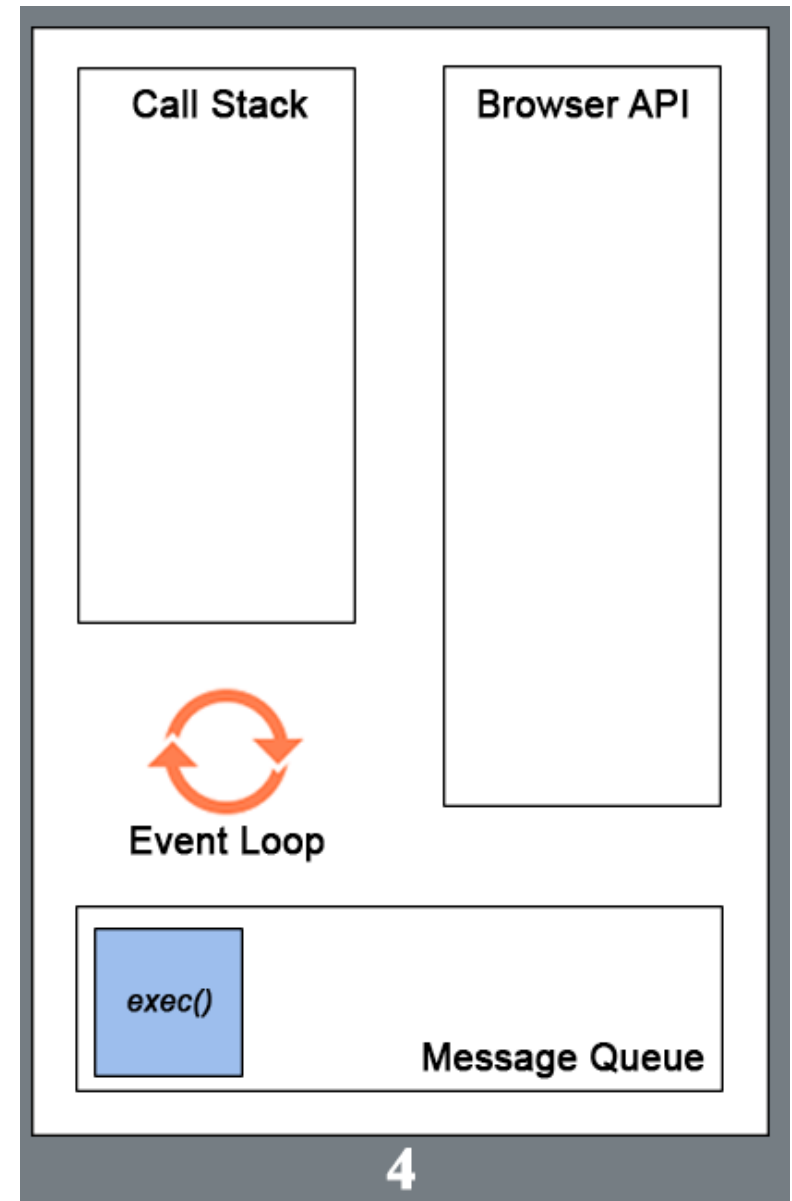
```
function main() {
  console.log('A');
  setTimeout(function display() {
    console.log('B');
  }, 0);
  console.log('C');
}
main();
```



<https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4>

Esempio

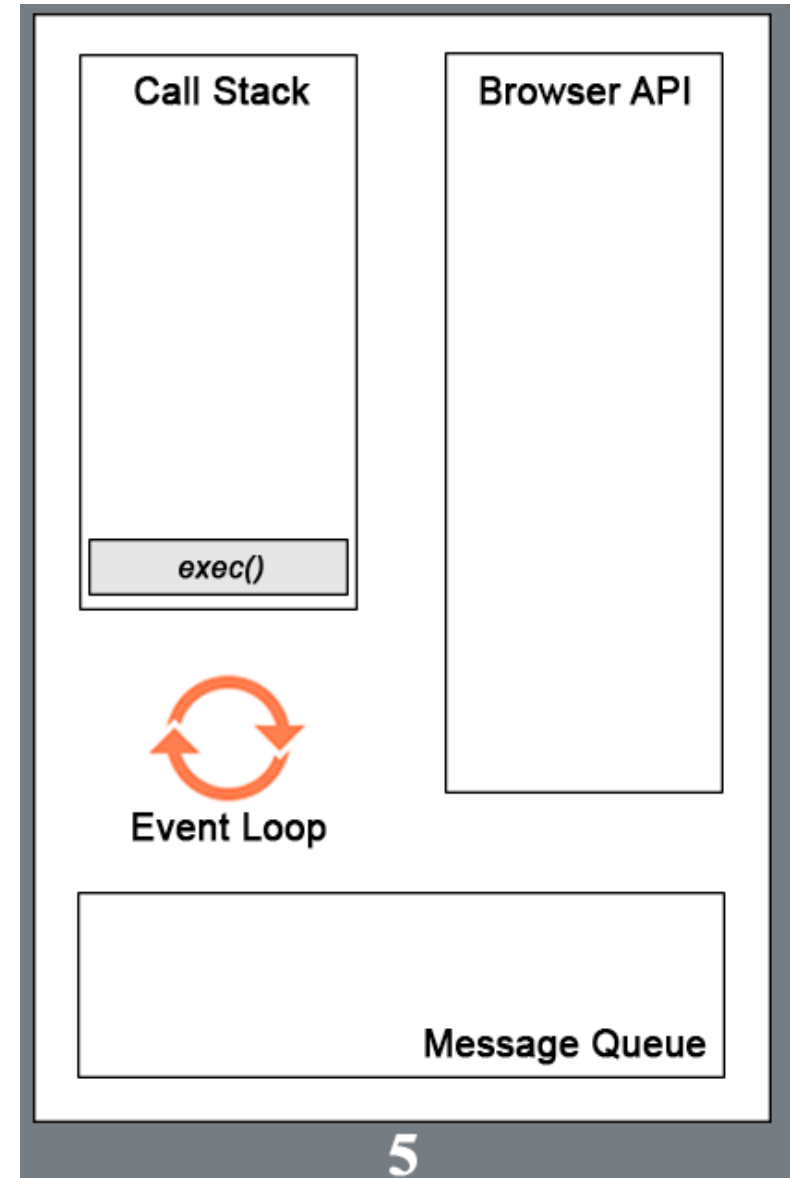
```
function main() {
  console.log('A');
  setTimeout(function display() {
    console.log('B');
  }, 0);
  console.log('C');
}
main();
```



<https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4>

Esempio

```
function main() {
  console.log('A');
  setTimeout(function display() {
    console.log('B');
  }, 0);
  console.log('C');
}
main();
```



<https://medium.com/front-end-weekly/javascript-event-loop-explained-4cd26af121d4>

Promises

Promises Basic

- Le Promise sono oggetti che rappresentano il **completamento** (o il fallimento) di un'operazione asincrona
 - Un contenitore per un valore assegnato in modo asincrono
 - Un contenitore per un valore futuro
- Vantaggi:
 - Non serve più un evento ed una callback per gestire il risultato asincrono
 - Le promises si possono concatenare evitando il **callback hell**

```
const fetchPromise = fetch('https://api.example.com/data');  
  
fetchPromise.then(response => response.json())  
  .then(data => console.log('Dati ricevuti:', data))  
  .catch(error => console.error('Errore nella richiesta:', error));
```

Background tasks



```
const fetchPromise = fetch('https://api.example.com/data');

fetchPromise.then(response => response.json())
  .then(data => console.log('Dati ricevuti:', data))
  .catch(error => console.error('Errore nella richiesta:', error));
```

Background tasks



```
const fetchPromise = fetch('https://api.example.com/data');
```

```
fetchPromise.then(response => response.json())  
  .then(data => console.log('Dati ricevuti:', data))  
  .catch(error => console.error('Errore nella richiesta:', error));
```

Background tasks



```
const fetchPromise = fetch('https://api.example.com/data');
```

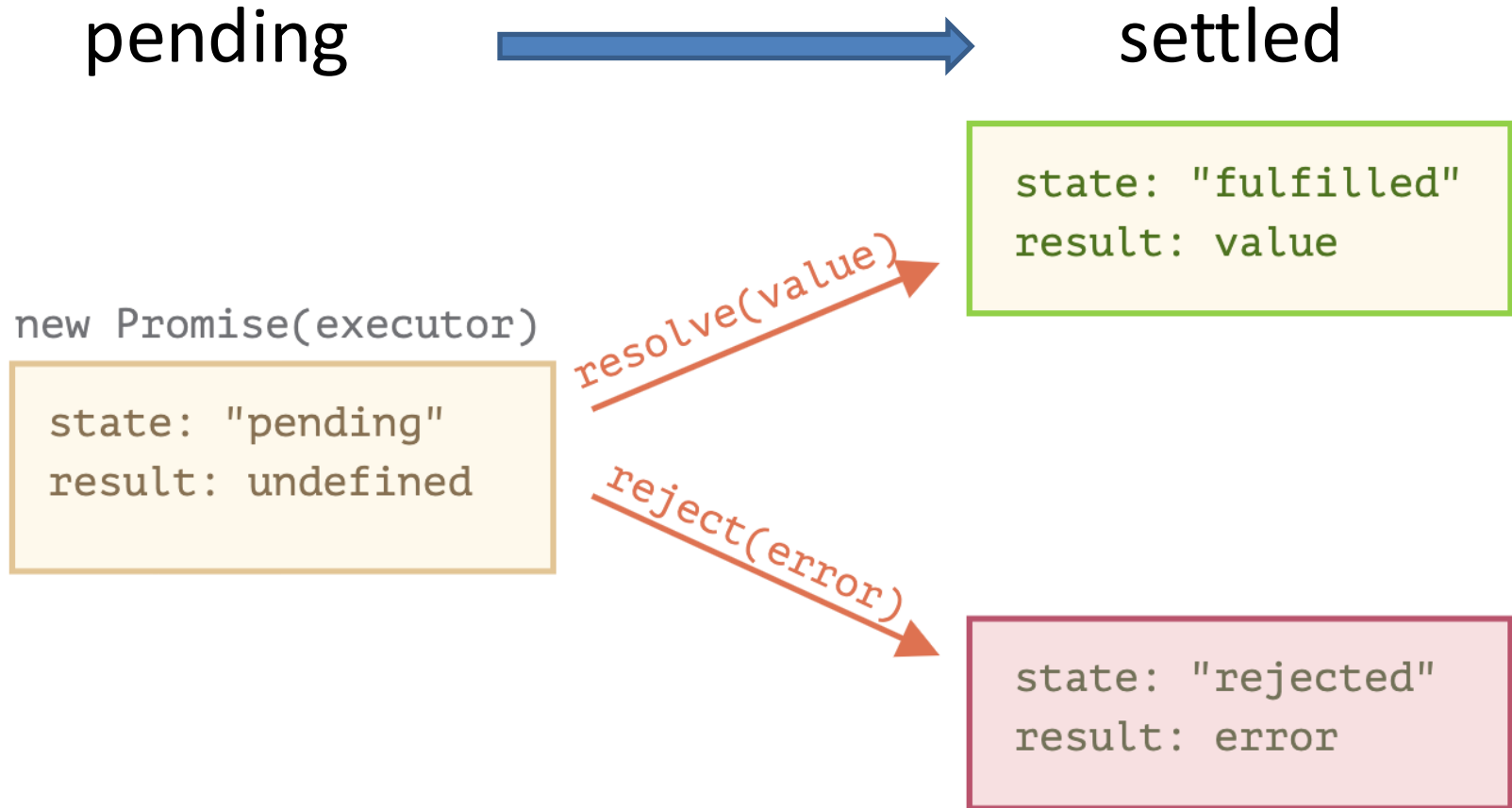
```
fetchPromise.then(response => response.json())
  .then(data => console.log('Dati ricevuti:', data))
  .catch(error => console.error('Errore nella richiesta:', error));
```

"Producing code" è un codice che può richiedere del tempo

"Consuming code" è il codice che deve attendere il risultato

Una promise è l'oggetto che collega producing code and consuming code

Ciclo di vita



Creare una promise

```
let promise = new Promise(function(resolve, reject) {
  // the function is executed automatically when the promise is constructed

  // after 1 second signal that the job is done with the result "done"
  setTimeout(() => resolve("done"), 1000);
});
```

new Promise(executor)

state: "pending"
result: undefined

resolve("done")

state: "fulfilled"
result: "done"

Creare una promise

```
let promise = new Promise(function(resolve, reject) {
  // the function is executed automatically when the promise is constructed

  // after 1 second signal that the job is done with the result "done"
  setTimeout(() => resolve("done"), 1000);
});
```

new Promise(executor)

state: "pending"
result: undefined

resolve("done")

state: "fulfilled"
result: "done"

```
let promise = new Promise(function(resolve, reject) {
  // after 1 second signal that the job is finished with an error
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});
```

new Promise(executor)

state: "pending"
result: undefined

reject(error)

state: "rejected"
result: error

Esempio

```
const randomPromise = new Promise(function (resolve, reject) {
  console.log('Inizia la Lotteria');
  setTimeout(function () {
    if (Math.random() >= 0.9999) {
      resolve('Hai Vinto');
    } else {
      reject(new Error('Mi Spiace, hai perso!!!'));
    }
  }, 3000);
});
```

Consumare una promise: then

```
promise.then(  
  function(result) { /* handle a successful result */ },  
  function(error) { /* handle an error */ }  
);
```

Consumare una promise: then

```
promise.then(
  function(result) { /* handle a successful result */ },
  function(error) { /* handle an error */ }
);
```

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => resolve("done!"), 1000);
});

// resolve runs the first function in .then
promise.then(
  result => alert(result), // shows "done!" after 1 second
  error => alert(error) // doesn't run
);
```

Gestire gli errori: catch

```
let promise = new Promise(function(resolve, reject) {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// reject runs the second function in .then
promise.then(
  result => alert(result), // doesn't run
  error => alert(error) // shows "Error: Whoops!" after 1 second
);
```

```
let promise = new Promise((resolve, reject) => {
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});

// .catch(f) is the same as promise.then(null, f)
promise.catch(alert); // shows "Error: Whoops!" after 1 second
```

Esercizio 1

- Costruisce una promise che si risolve dopo 5 secondi.
 - Il valore della promessa dovrebbe essere:
"la mia prima promise è stata un successo!".
- Al momento dell'adempimento, il messaggio deve essere stampato nella console

Esercizio 2

```
function orderExecutor(resolve, reject) {
  console.log('Pizza ordered...');
  setTimeout(function () {
    resolve("Here's your pizza!");
  }, 1000);
}

let orderPizza = new Promise(orderExecutor);
orderPizza.then(console.log);
console.log("Waiting for my pizza!");
```

Qual è l'ordine delle dichiarazioni stampate nella console?
Provate a farlo senza incollarlo nella console

Esercizio 3

```
function orderExecutor(resolve, reject) {
  console.log('Pizza ordered...');
  resolve("Here's your pizza!");
}

let orderPizza = new Promise(orderExecutor);
orderPizza.then(console.log);
console.log("Waiting for my pizza!");
```

Qual è l'ordine delle dichiarazioni stampate nella console?
Provate a farlo senza incollarlo nella console

Esercizio 4

```
function buttonExecutor(resolve, reject) {
  let myBtn = document.querySelector('button');
  myBtn.addEventListener('click', resolve);
  setTimeout(reject, 5000);
}

let betterClick = new Promise(buttonExecutor);
betterClick
  .then(function () { console.log('Option A'); })
  .catch(function () { console.log('Option B'); });
```

Cosa succede se il pulsante non viene cliccato entro 5 secondi?
Provate a farlo senza incollarlo nella console

Esercizio 4 v2

```
function buttonExecutor(resolve, reject) {
  let myBtn = document.querySelector('button');
  myBtn.addEventListener('click', function() {
    resolve();
    console.log('clicked!');
  });
  setTimeout(reject, 5000);
}

let betterClick = new Promise(buttonExecutor);
betterClick
  .then(function () { console.log('Option A'); })
  .catch(function () { console.log('Option B'); });
```

Cosa succede se il pulsante viene cliccato dopo 5 secondi?

Si noti la modifica del eventListener.

Provate a farlo senza incollarlo nella console

Concatenare Promises

```
new Promise(function(resolve, reject) {

    setTimeout(() => resolve(1), 1000); // (*)

}).then(function(result) { // (**)

    alert(result); // 1
    return result * 2;

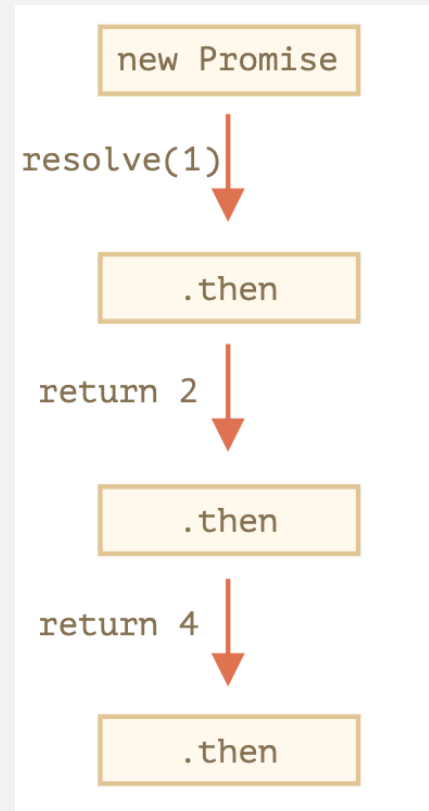
}).then(function(result) { // (***)

    alert(result); // 2
    return result * 2;

}).then(function(result) {

    alert(result); // 4
    return result * 2;

});
```



Esercizio 5

```
function executor(resolve, reject) {
  resolve(1);
}

function add(value) {
  return value + 5;
}

function multiply(value) {
  return value * 6;
}

let myPromise = new Promise(executor);
myPromise
  .then(add)
  .then(multiply)
  .then(console.log);
```

Cosa stampa?

Provate a farlo senza
incollarlo nella
console

Esercizio 6

```
function executor(resolve, reject) {
  resolve(1);
}

function add(value) {
  return new Promise(function(resolve, reject) {
    resolve(value + 5);
  });
}

let myPromise = new Promise(executor);
myPromise
  .then(add)
  .then(console.log);
```

Cosa stampa?

Provate a farlo senza incollarlo nella console

... finally

```
.finally(() => alert("Promise ready"))  
.then(result => alert(result)); // <-- .then handles the result
```

```
.finally(() => alert("Promise ready"))  
.catch(err => alert(err)); // <-- .catch handles the error object
```

Callback Hell

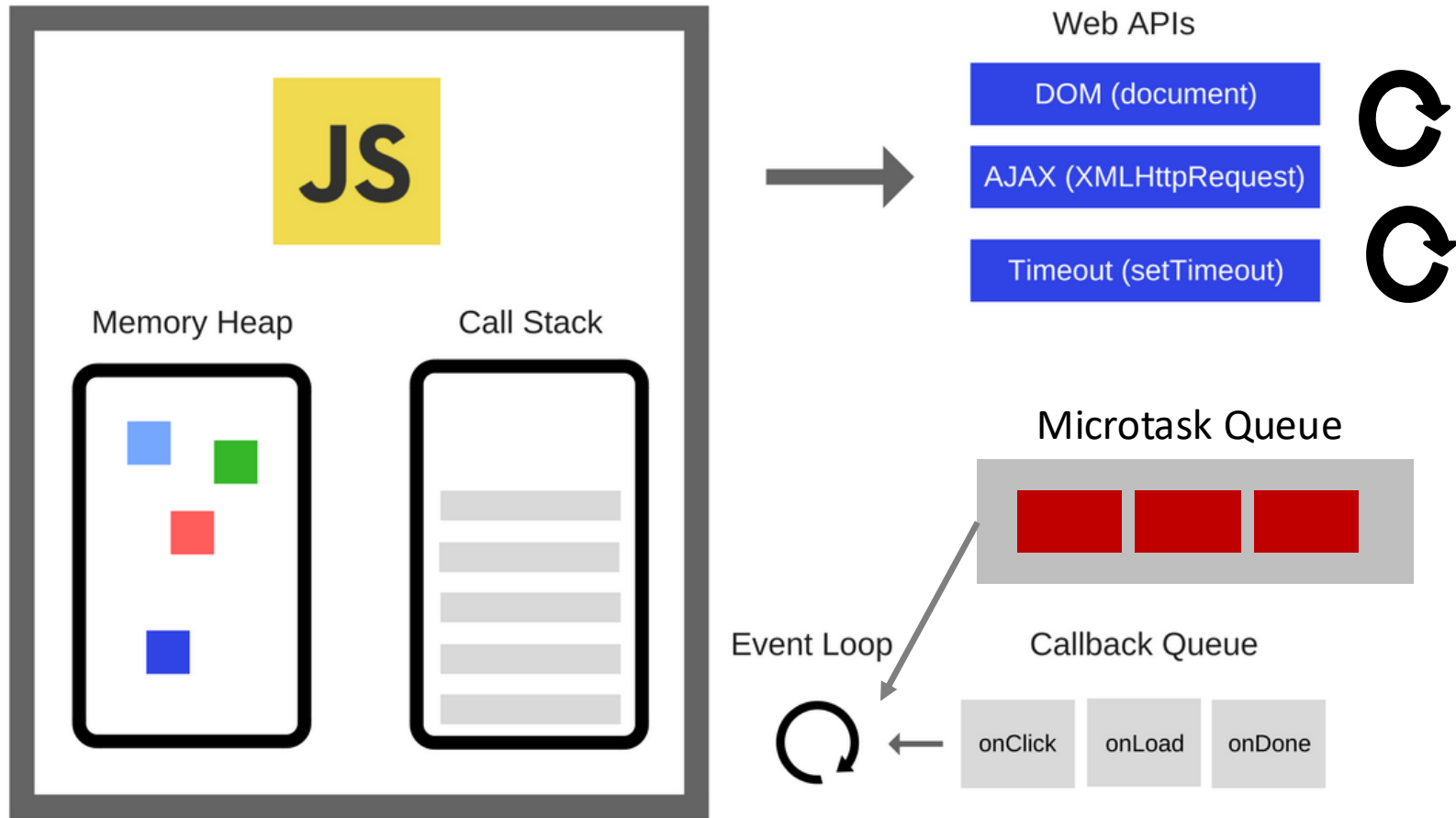
```
setTimeout(() => {
  console.log('1 second passed');
  setTimeout(() => {
    console.log('2 seconds passed');
    setTimeout(() => {
      console.log('3 second passed');
      setTimeout(() => {
        console.log('4 second passed');
      }, 1000);
    }, 1000);
  }, 1000);
}, 1000);
```

```
// Promisifying setTimeout
const wait = function (seconds) {
  return new Promise(function (resolve) {
    setTimeout(resolve, seconds * 1000);
  });
};

wait(1)
  .then(() => {
    console.log('1 second passed');
    return wait(1);
  })
  .then(() => {
    console.log('2 second passed');
    return wait(1);
  })
  .then(() => {
    console.log('3 second passed');
    return wait(1);
  })
  .then(() => console.log('4 second passed'));
```


MICROTASK

Microtask queue



- La coda dei Microtask ha priorità su quella delle callback

Esempio priorità

```
console.log('Start');
setTimeout(() => console.log('Timer 0'), 0);
Promise.resolve('resolved Promise 1').then((res) => {
  console.log(res);
});
Promise.resolve('resolved Promise 2').then((res) => {
  for (let index = 0; index < 10000000000; index++) {}
  console.log(res);
});

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