Promises

Promises Basic

- Una Promise è un oggetto usato come placeholder per il risultato futuro di una 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**

https://javascript.info/promise-basics

I Promise a Result!

"Producing code" è un codice che può richiedere del tempo
"Consuming code" è il codice che deve attendere il risultato

Una promise è un oggetto che collega producing code and consuming code

I Promise a Result!

"Producing code" è un codice che può richiedere del tempo
"Consuming code" è il codice che deve attendere il risultato

Una promise è un oggetto che collega producing code and consuming code

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});
```

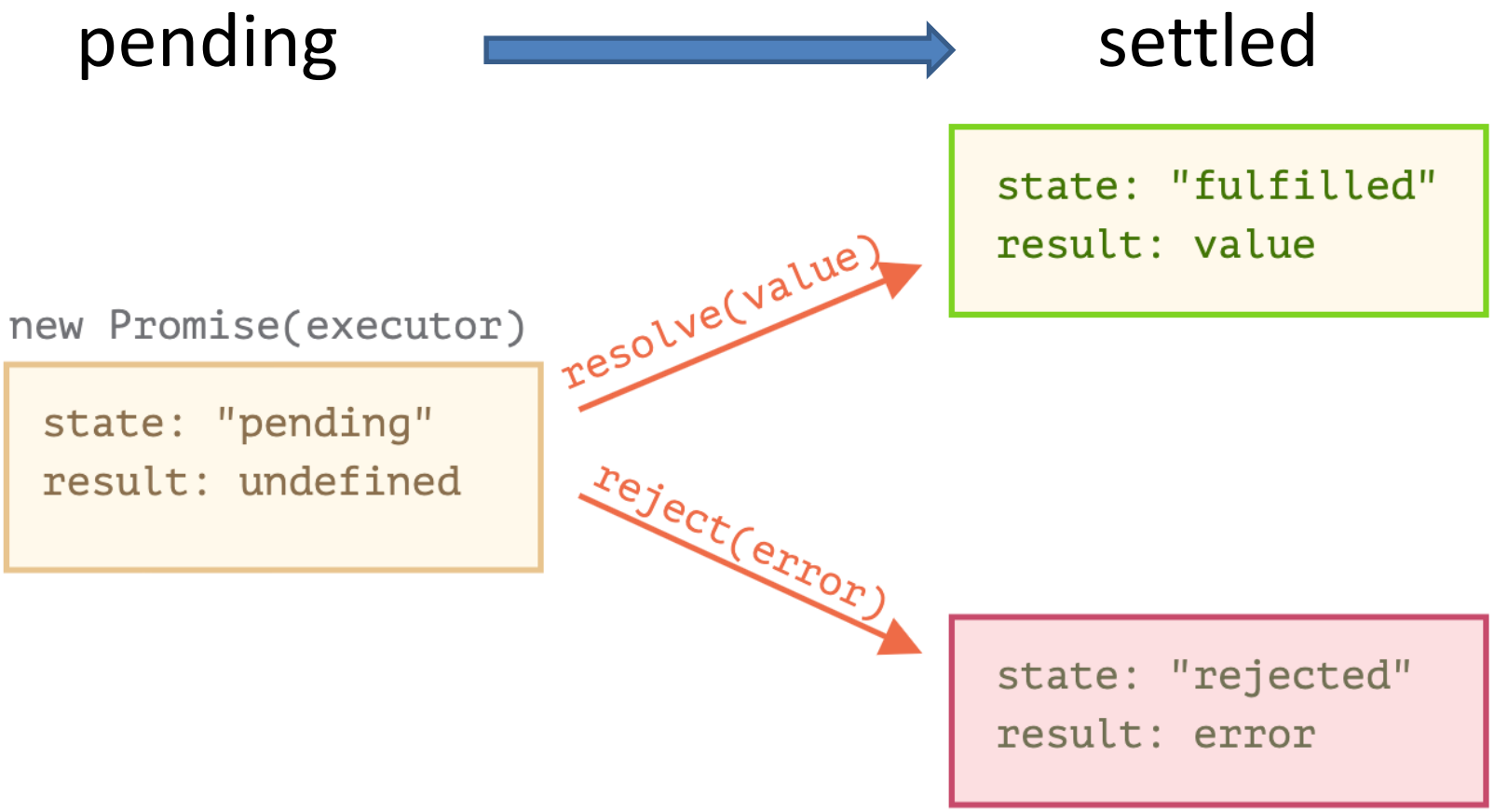
I Promise a Result!

"Producing code" è un codice che può richiedere del tempo
"Consuming code" è il codice che deve attendere il risultato

Una promise è un oggetto che collega producing code and consuming code

```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Ciclo di vita



https://javascript.info/promise-basics

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
```

Esempio

```
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

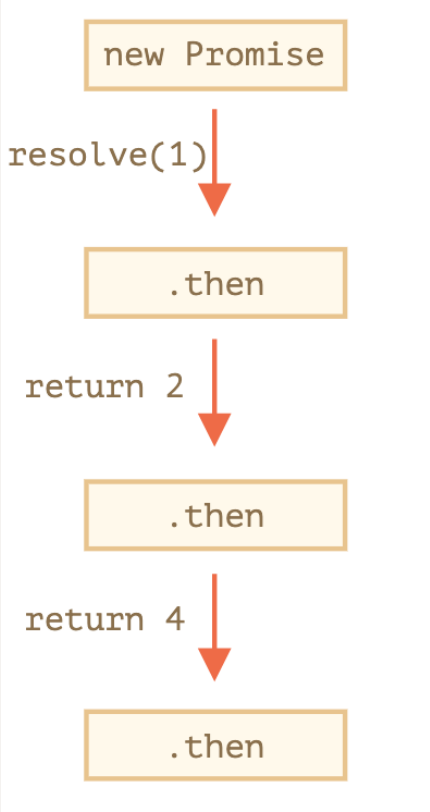
Esempio v2

```
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 1

```
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 2

```
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
```

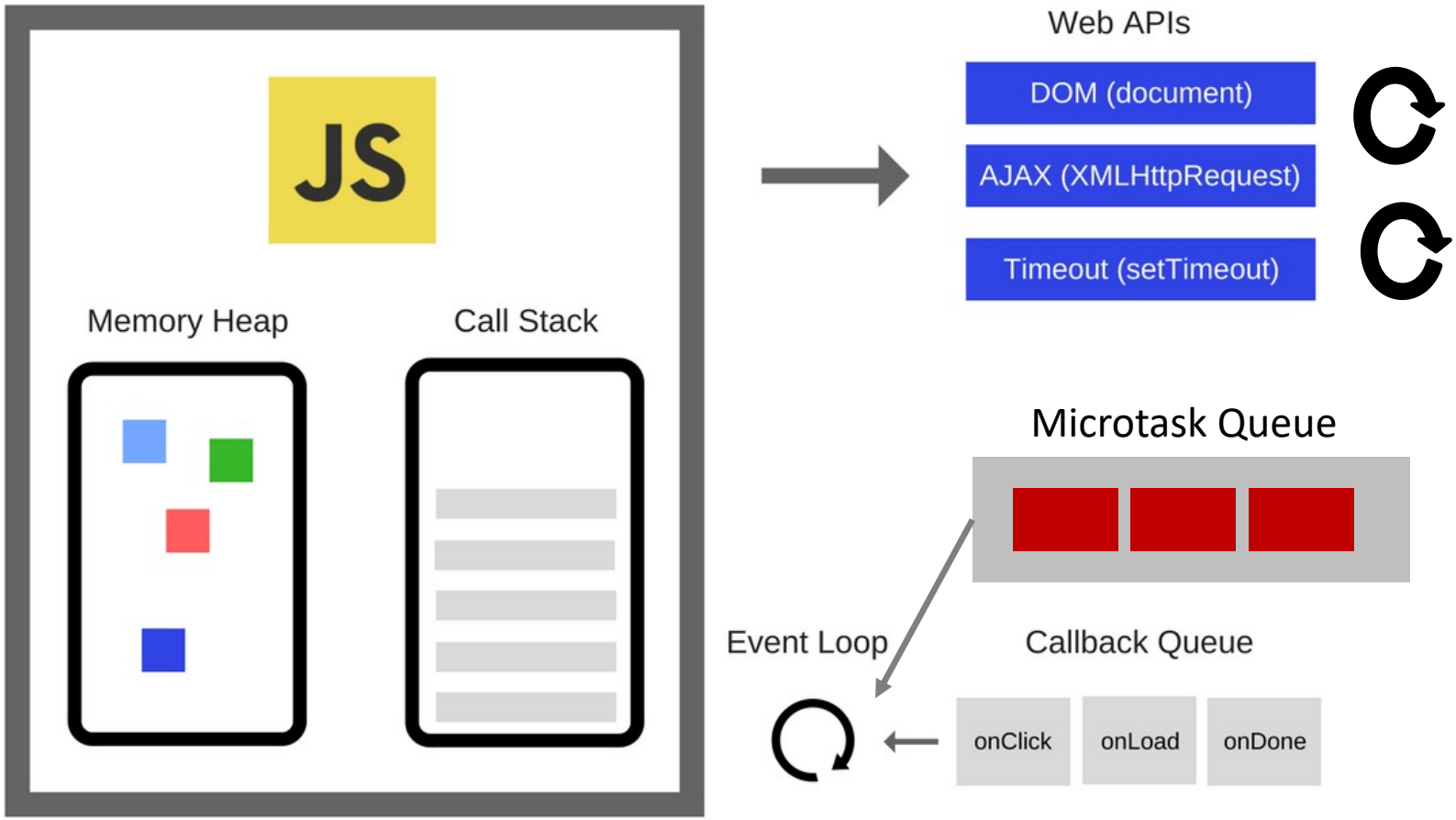
Callback Hell

```
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');
```

ASYNC AWAIT

async

```
async function f() {
  return 1;
}
```

- async before a function means that a function always returns a promise

async

```
async function f() {
  return 1;
}
```

- async before a function means that a function always returns a promise

```
async function f() {
  return 1;
}

f().then(alert); // 1
```

async

```
async function f() {
  return 1;
}
```

- async before a function means that a function always returns a promise

```
async function f() {
  return 1;
}

f().then(alert); // 1
```

```
async function f() {
  return Promise.resolve(1);
}

f().then(alert); // 1
```

await

- Rende il codice asincrono ed aspetta la risposta

```
// works only inside async functions
let value = await promise;
```

await

- Rende il codice asincrono ed aspetta la risposta futura

```
// works only inside async functions
let value = await promise;
```

```
async function f() {

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

  let result = await promise; // wait until the promise resolves (*)

  alert(result); // "done!"
}

f();
```

error handling: try catch

- in the case of a rejection a promise throws the error
  - as if there were a throw statement at that line

error handling: try catch

- in the case of a rejection a promise throws the error
  - as if there were a throw statement at that line

- can catch that error using try..catch

```
async function f() {

  try {
    let response = await fetch('http://no-such-url');
  } catch(err) {
    alert(err); // TypeError: failed to fetch
  }
}

f();
```

Fetch with Async/Await

GET

```
async function fetchUsers(endpoint) {
  const res = await fetch(endpoint);
  let data = await res.json();

  data = data.map(user => user.username);

  console.log(data);
}

fetchUsers('https://jsonplaceholder.typicode.com/users');
```

GET V2

```
async function fetchUsers(endpoint) {
  const res = await fetch(endpoint);
  const data = await res.json();

  return data;
}

fetchUsers('https://jsonplaceholder.typicode.com/users')
  .then(data => {
    console.log(data.map(user => user.username));
  });
```

Errors

```
async function fetchUsers(endpoint) {
  const res = await fetch(endpoint);

  if (!res.ok) {
    throw new Error(res.status); // 404
  }

  const data = await res.json();
  return data;
}

fetchUsers('https://jsonplaceholder.typicode.com/usersZZZ')
  .then(data => {
    console.log(data.map(user => user.website));
  })
  .catch(err => console.log('Ooops, error', err.message));
```

Errors V2

```
async function fetchUsers(endpoint) {
  try {
    const res = await fetch(endpoint);
    if (!res.ok) {
      throw new Error(res.status); // 404
    }
    const data = await res.json();
    data = data.map(user => user.username);
    console.log(data);
  } catch (error) {
    // do somthing
  }
}

fetchUsers('https://jsonplaceholder.typicode.com/usersZZZ')
```

Esempio GitHub con async/await

```
const getInfoProf = async function(docente){
  try {

    const url = `http://pw.netgroup.uniroma2.it/docenti/${docente}.json`

    const res = await fetch(url);
    const json_data = await res.json()
    const url_user = `https://api.github.com/users/${json_data.name}`

    const res_gh = await fetch(url_user)
    const user_data = await res_gh.json()
```

CORS

CORS: cross-origin HTTP request

- un client richiede una risorsa di un differente dominio, protocollo o porta.

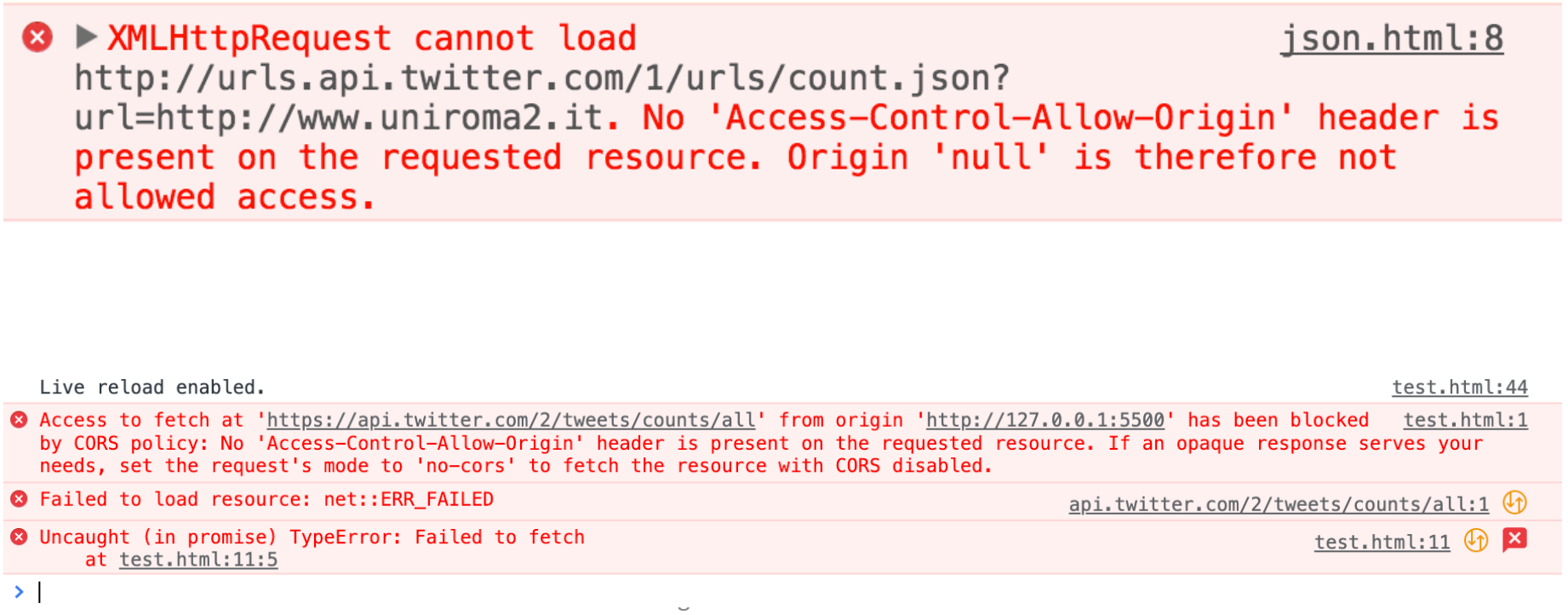
- Esempio
  - una web application con dominio X non può richiedere una risorsa ad un dominio Y tramite AJAX request se Y non ha abilitato il CORS.

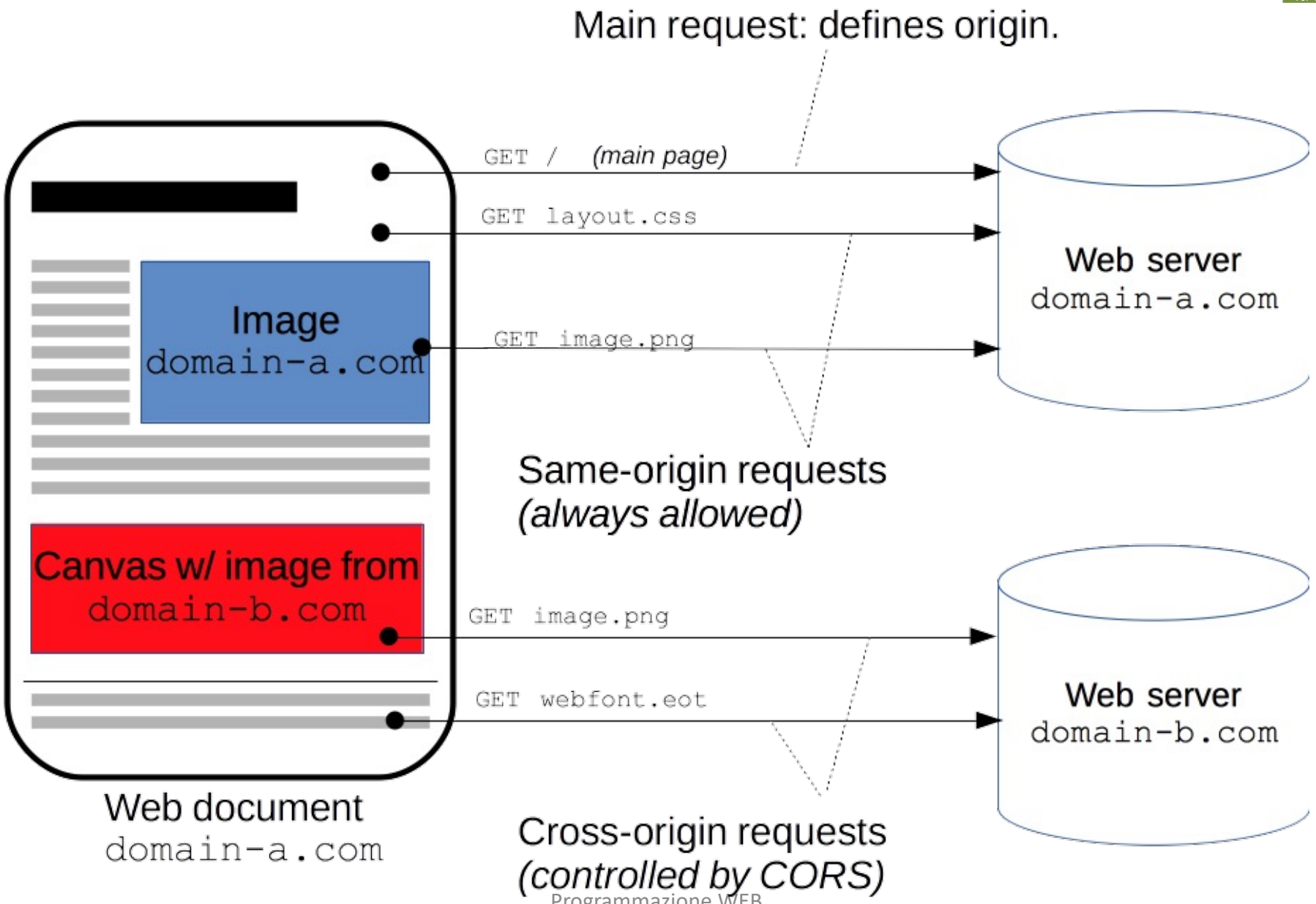
https://italiancoders.it/cors-in-dettaglio/

https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

Same origin policy

- Un browser permette agli script contenuti in una pagina web di accedere ai dati contenuti in un'altra risorsa web (altra pagina web, json ecc) solo se entrambe le pagine hanno la stessa origine




Main request: defines origin.
GET / (main page)
GET layout.css
GET image.png
Image
domain-a.com
Web server
domain-a.com
Same-origin requests
(always allowed)
Canvas w/ image from
domain-b.com
GET image.png
GET webfont.eot
Web server
domain-b.com
Web document
domain-a.com
Cross-origin requests
(controlled by CORS)

Cross Origin Resource Sharing

- Come fare se vogliamo espressamente permettere il resource sharing tra due siti diversi? → CORS
  - Standard W3C per condividere risorse tra domini diversi
  - Prevede richiesta di autorizzazione (client) e autorizzazione (server)

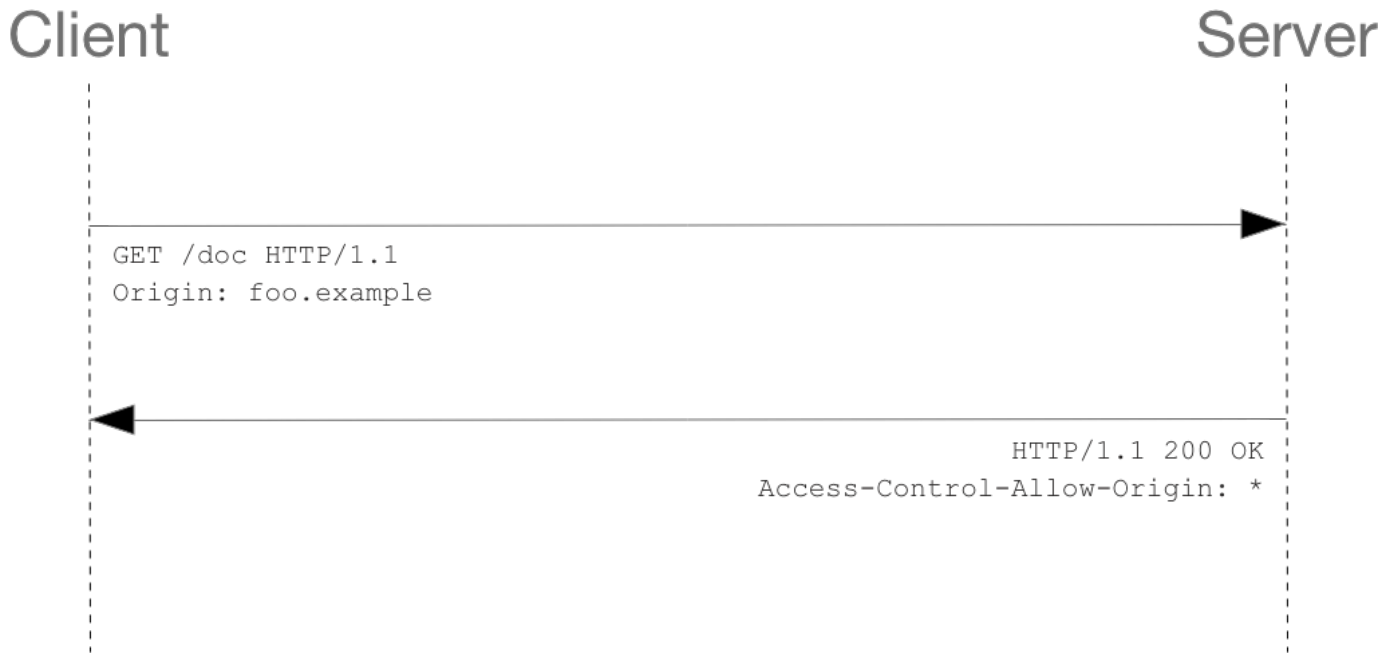
- Viene implementato inviando degli header HTTP in req/resp
  - Richieste Semplici
  - Richieste Preflight

Simple request

- Metodi Ammessi
  - GET
  - HEAD
  - POST
- Header pemessi
  - Accept
  - Accept-Language
  - Content-Language
  - Content-Type (but note the additional requirements below)
  - …
- Valori ammessi per header Content-Type:
  - application/x-www-form-urlencoded
  - multipart/form-data
  - text/plain

Simple request

```
const xhr = new XMLHttpRequest();
const url = 'https://bar.other/resources/public-data/';

xhr.open('GET', url);
xhr.onreadystatechange = someHandler;
xhr.send();
```



Access-Control-Allow-Origin: * means that the resource can be accessed by **any** origin.

Pre-flight request

```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
xhr.setRequestHeader('X-PINGOTHER', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
xhr.onreadystatechange = handler;
xhr.send('<person><name>Arun</name></person>');
```

Pre-flight request

```
const xhr = new XMLHttpRequest();
xhr.open('POST', 'https://bar.other/resources/post-here/');
xhr.setRequestHeader('X-PINGOTHER', 'pingpong');
xhr.setRequestHeader('Content-Type', 'application/xml');
xhr.onreadystatechange = handler;
xhr.send('<person><name>Arun</name></person>');
```