Web Client - **#1 (Partie 1)**

[Ajax]

Cyril Rouyer

Sommaire

- Principes
- Historique
- Asynchronous
- XML (?)
- Sécurité

Principes

- Ajax : Asynchronous Javascript And XML
- Objectifs :
  - faire communiquer le client avec le serveur
  - Modifier la page mais sans recharger la page

Historique

- **XMLHttpRequest (XHR) :** Inventé par Microsoft (sorte d'ActiveX pour IE en 1998)

- Permet au navigateur d'exécuter une requête HTTP sans recharger la page

- Permet de traiter les résultats émis par le serveur en javascript

- Tous les autres navigateurs ont repris le concept depuis

Exemple

```
var url = "/albums"
const xhr = new XMLHttpRequest()

xhr.addEventListener('load', function() {
  if(this.readyState === XMLHttpRequest.DONE && this.status >=200 && this.status <=
  299){
    console.log(JSON.parse(this.responseText))
  }
  else {
    console.log(“Une erreur métier s’est produite”, this.responseText, this.status)
  }
})
xhr.addEventListener('error', function(err) {
    console.log(“Une erreur bas-niveau s'est produite”)
    console.log(err)
})
xhr.open(‘GET’, url, true)
xhr.send()
```

“Asynchronous”

- La requête n’est pas bloquante
- Javascript intercepte la réponse comme un **événement** (onreadystatechange)
- S’appuie sur le protocole HTTP pour détecter les erreurs (404, 403, …)

“XML (?)”

- Au départ, la réponse du serveur contenait du XML

- Mais le serveur peut renvoyer n’importe quel format : JSON, HTML, texte, …

- Le plus utilisé aujourd’hui est le JSON parce qu’on peut directement l’assimiler à du Javascript (objects, arrays)

JSON

- Objet litéral : { … }
- Primitives autorisées :
  - Number, String, Boolean, Array, Object
    - Pas de regexp, pas de function, pas de undefined
- null autorisé
- Number : pas de NaN, pas de Infinity
- String : pas de simple quotes
- Objects : les propriétés en tant que String

JSON 2


```
{
“prop_number": 25,
"prop_string": "test",
"prop_boolean": true,
"prop_array": [ 1, 2.3, "string"],
"prop_null": null,
"prop_ object": { "prop1": "test", "prop2": 2 },
"prop_complex": [ {"prop1": "test"}, {"prop1": "test"} ]
}
```

Asynchronisme

Dans le détail

Call Stack & Event Loop

- Runtime JS : mono-thread, une chose après l'autre
- Browser : Event Loop > gère une queue à côté pour pouvoir injecter les traitements **différés** quand la stack est vide

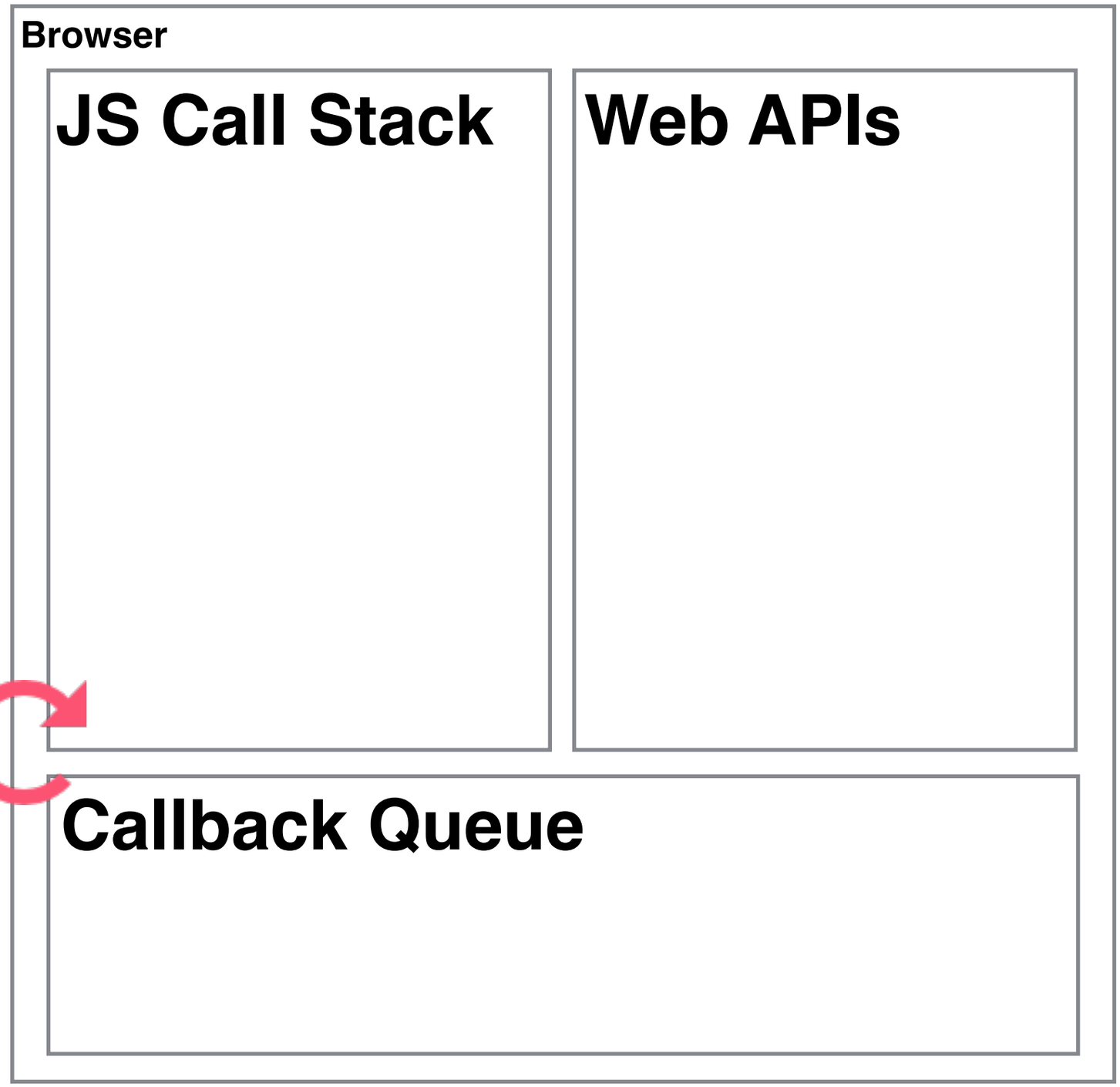
- Mais ça reste une seule boucle, toujours 1 seul thread

Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

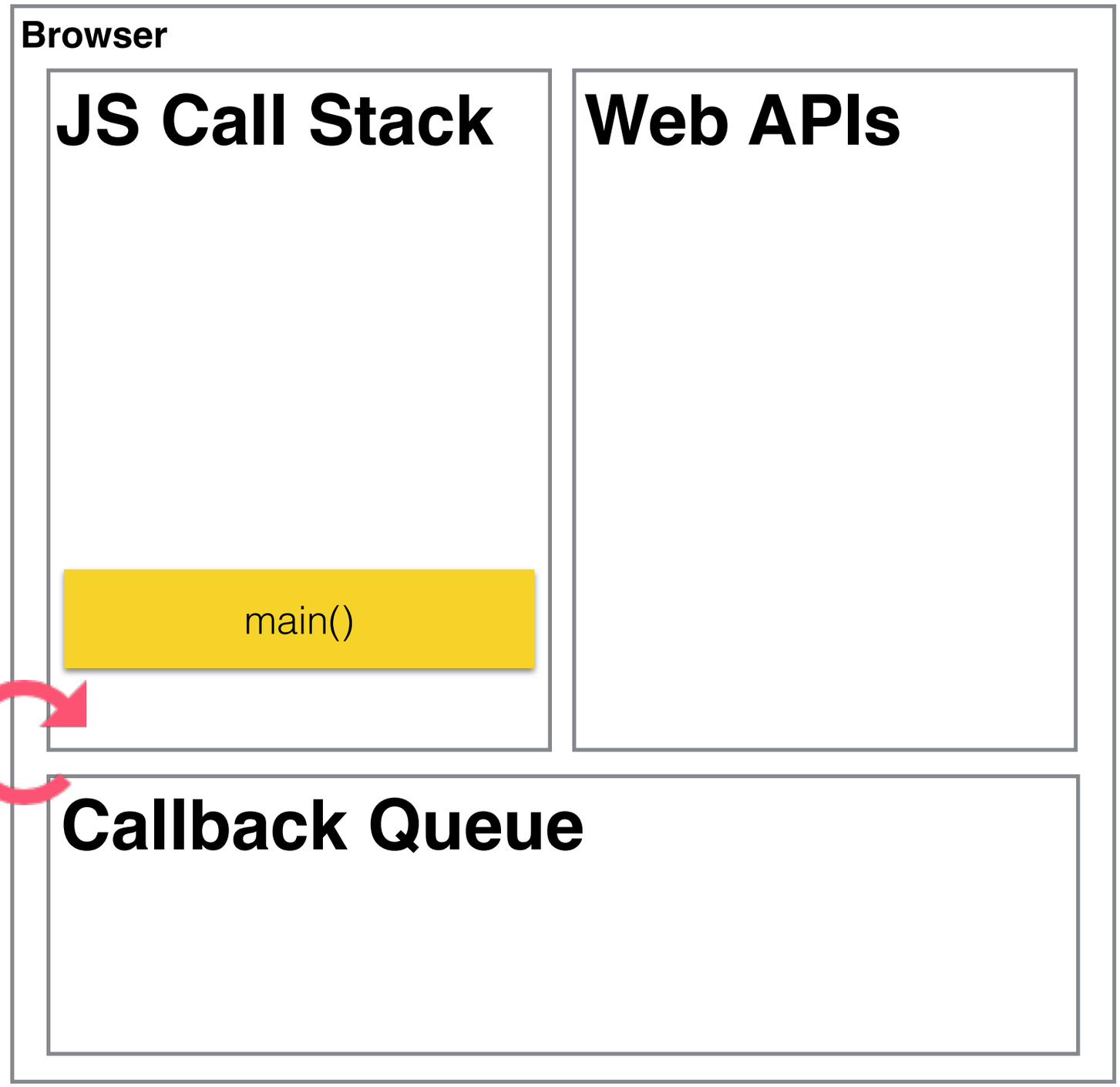


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

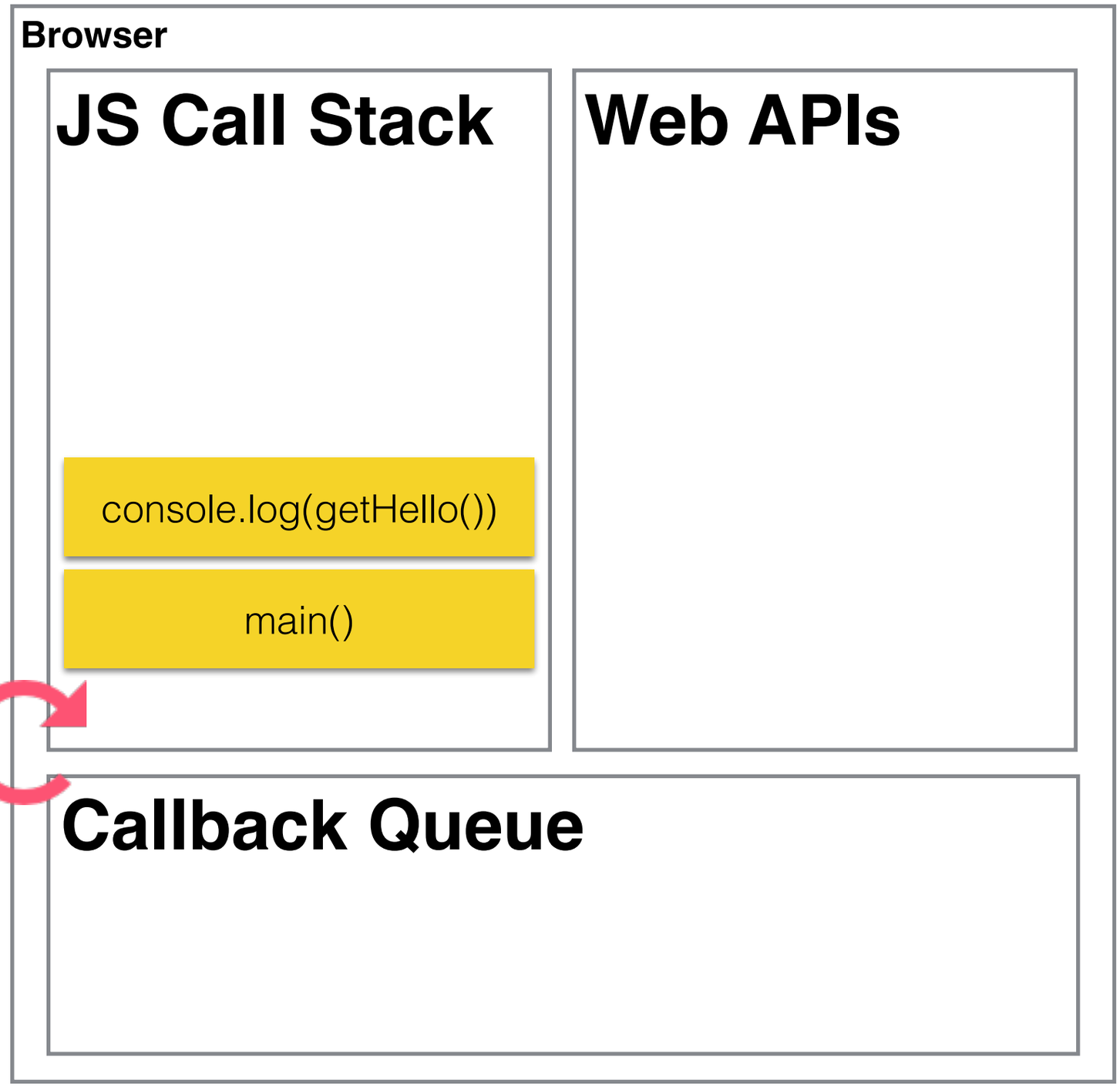


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

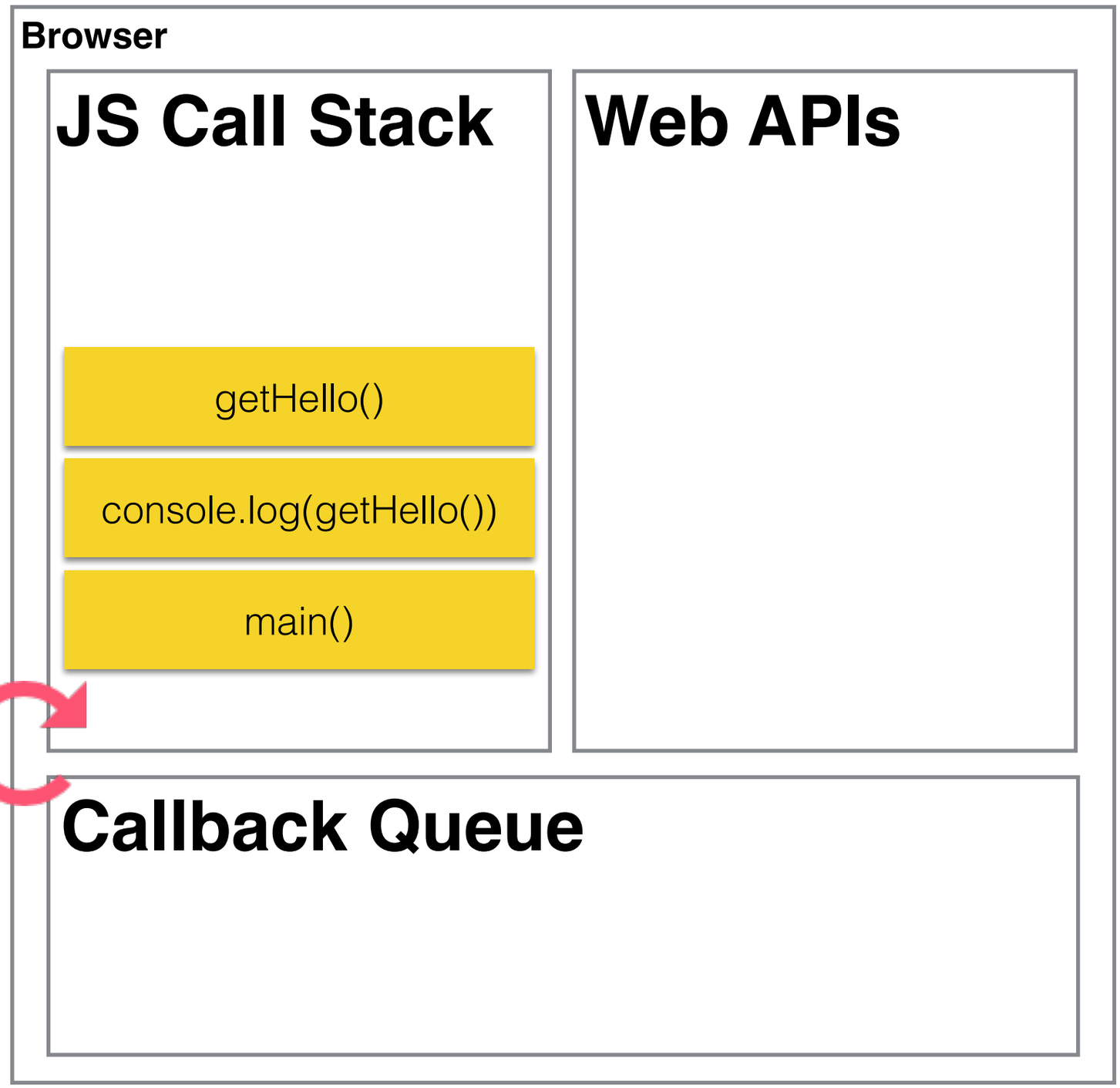


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

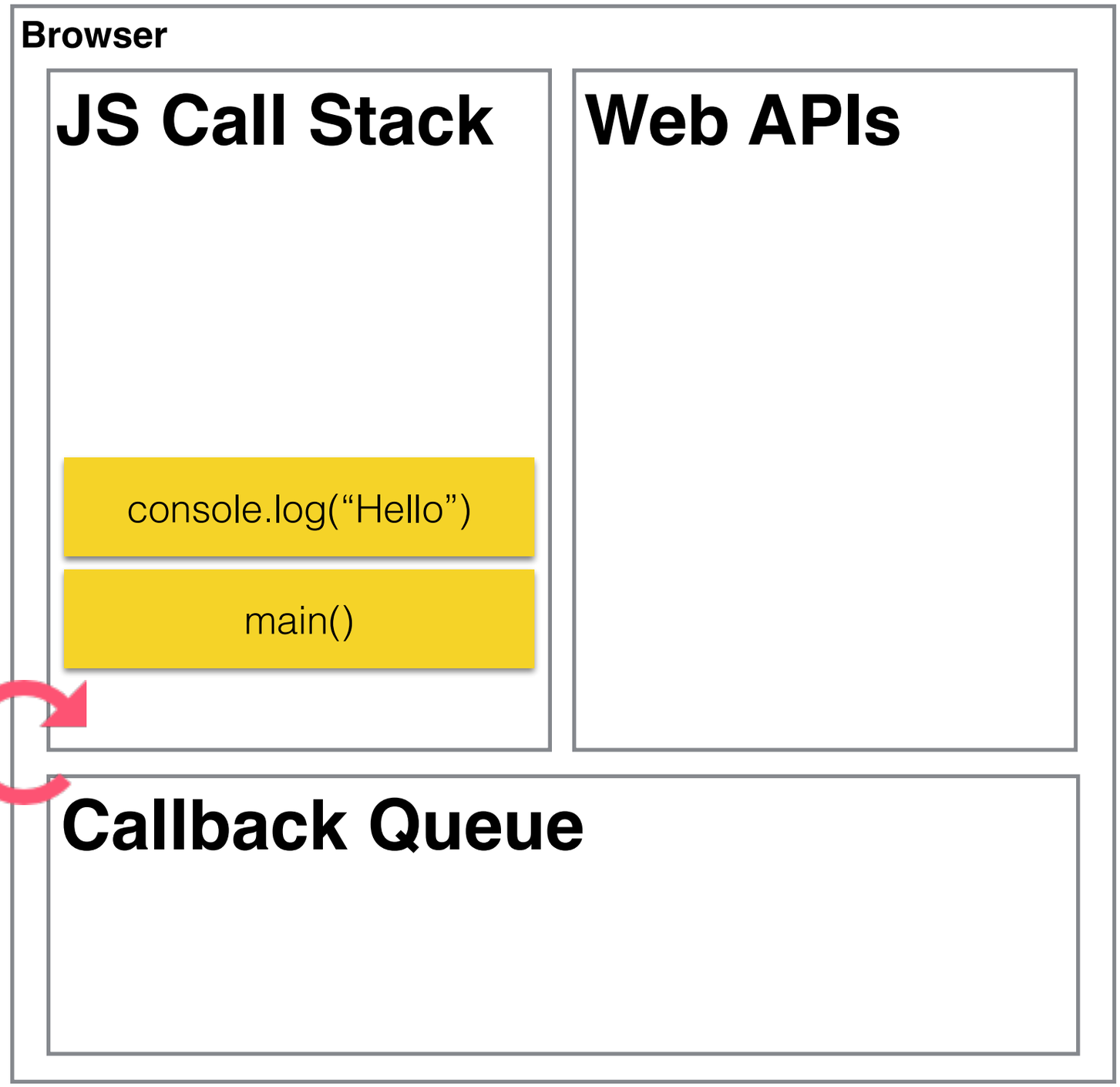


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

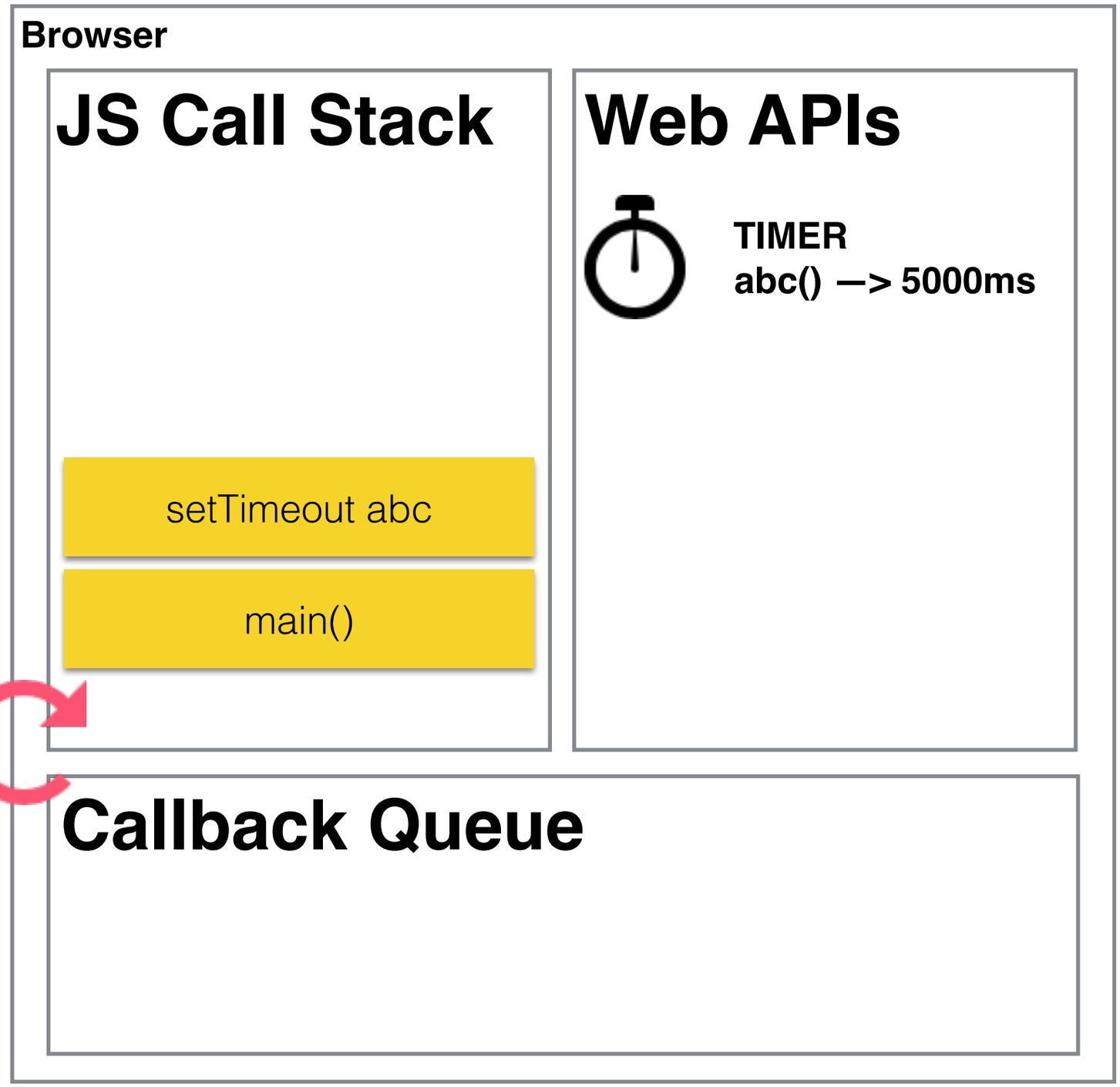


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

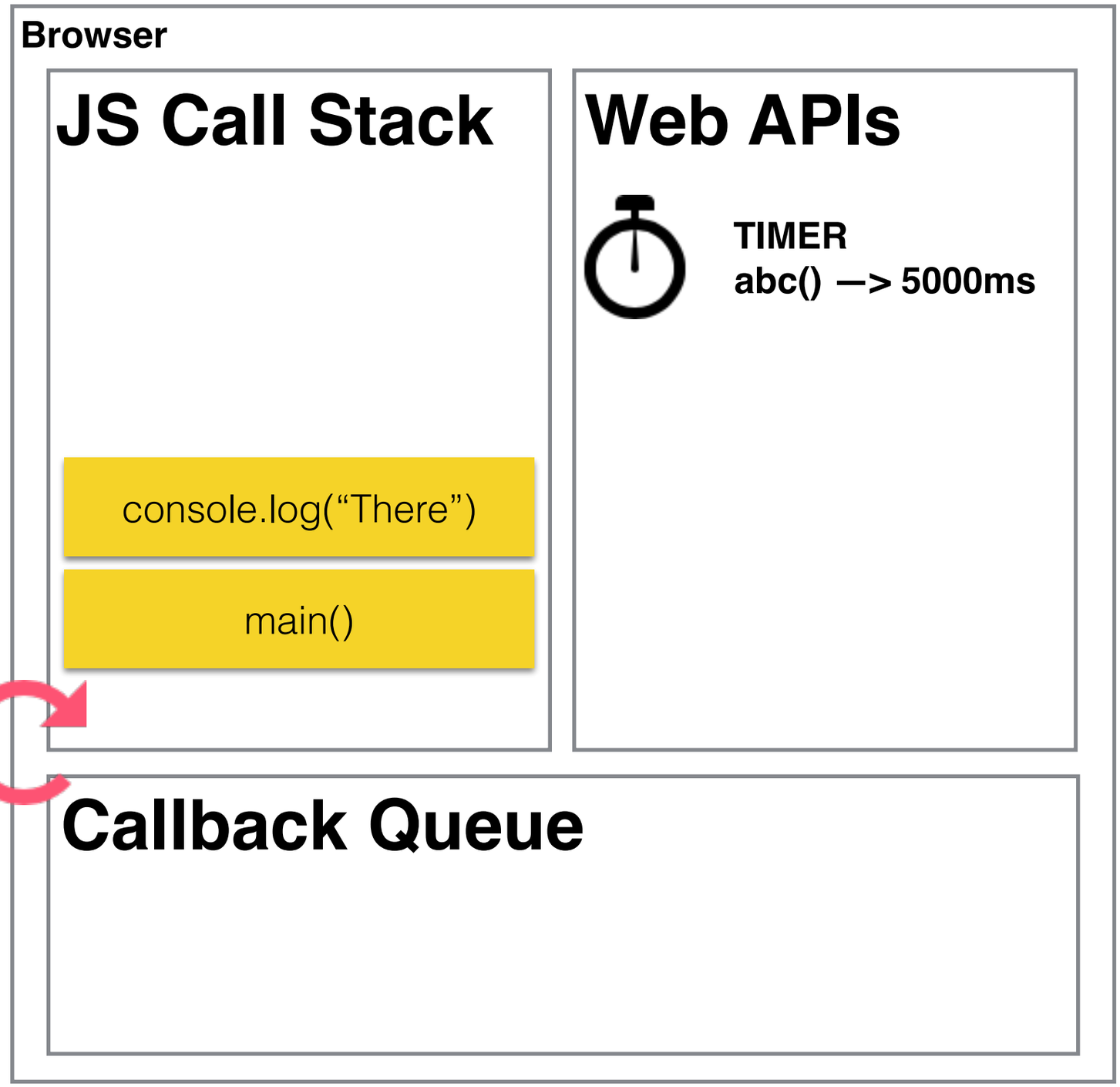


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

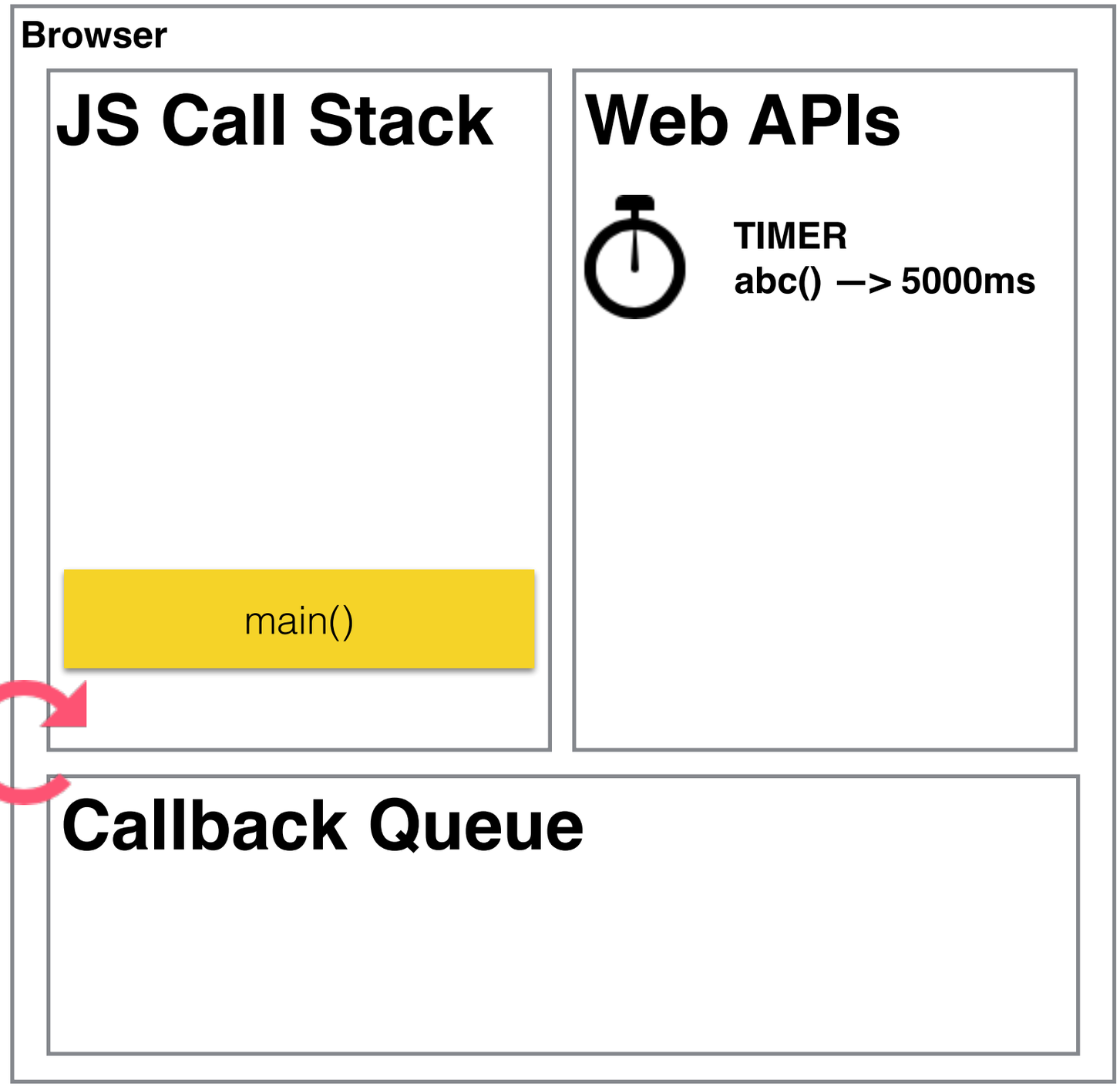


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

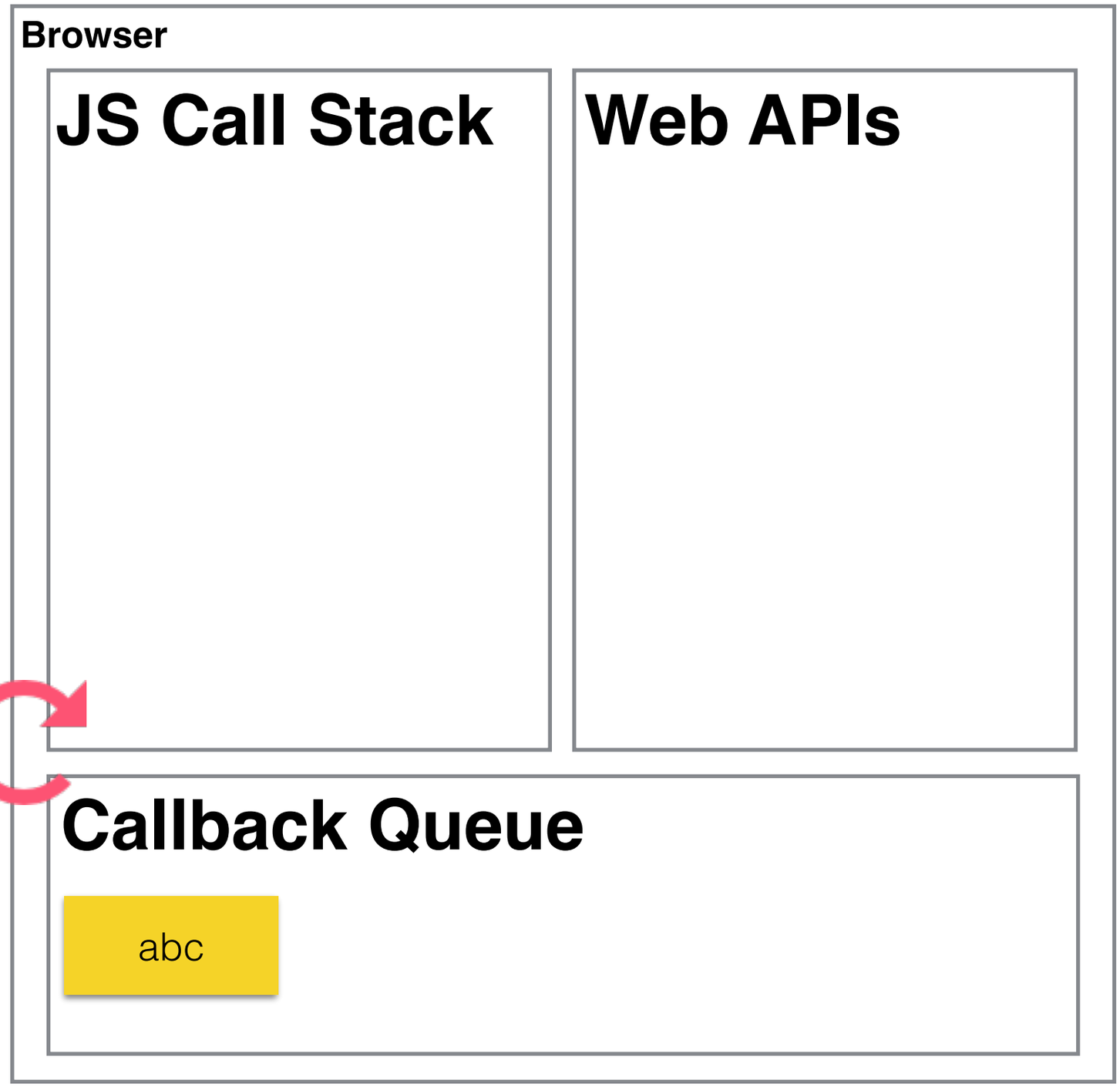


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

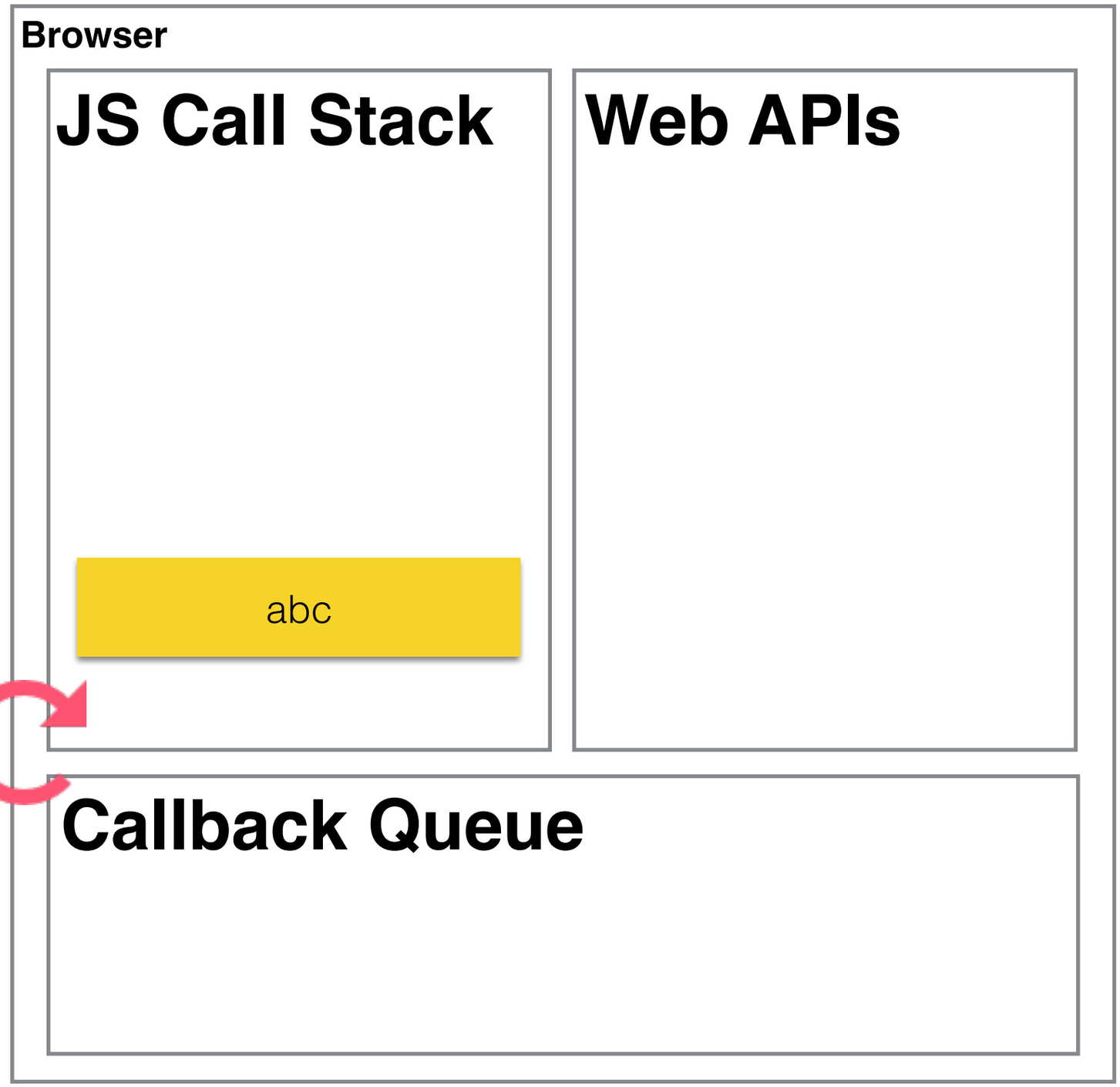


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

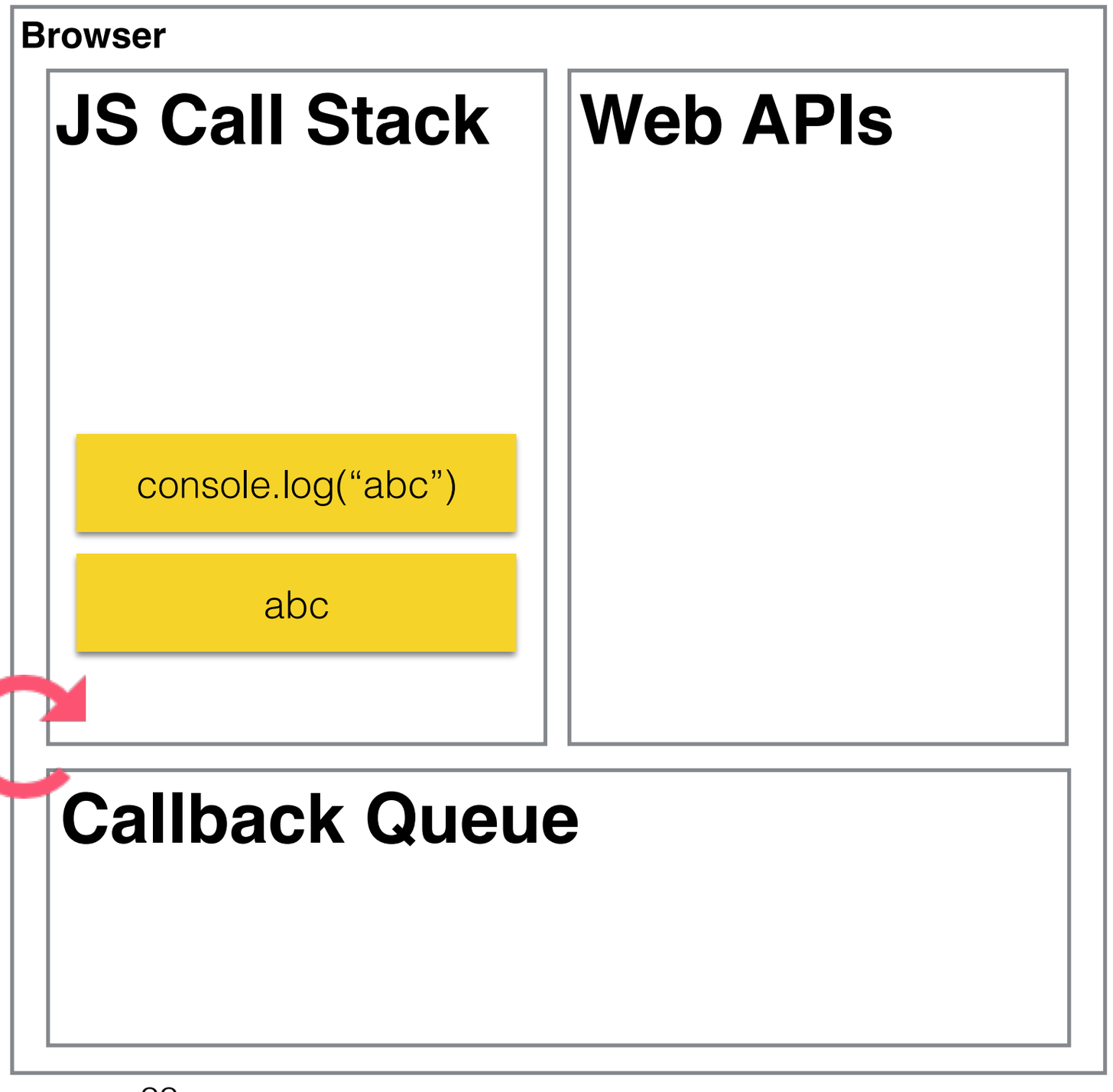


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

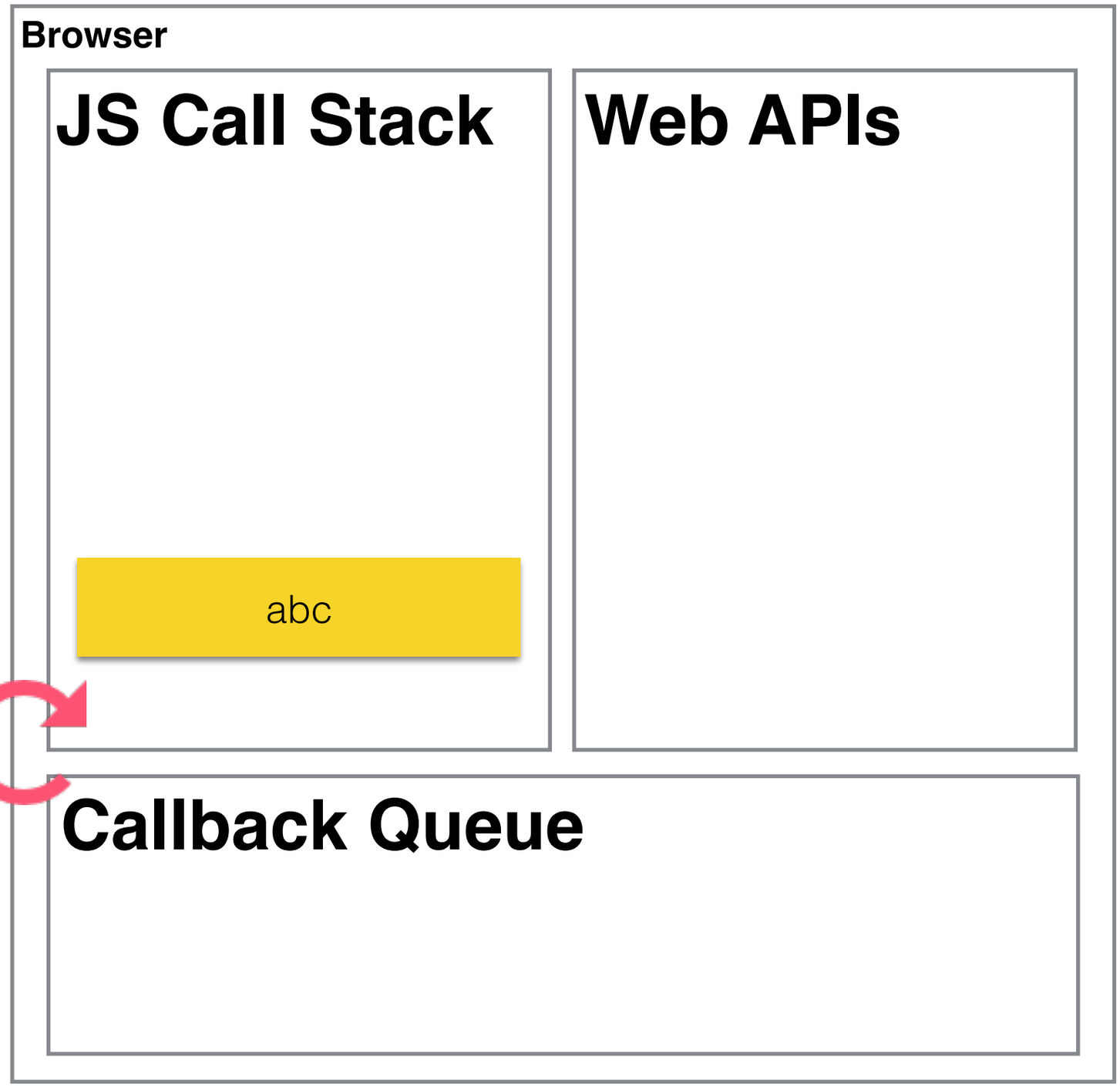


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```

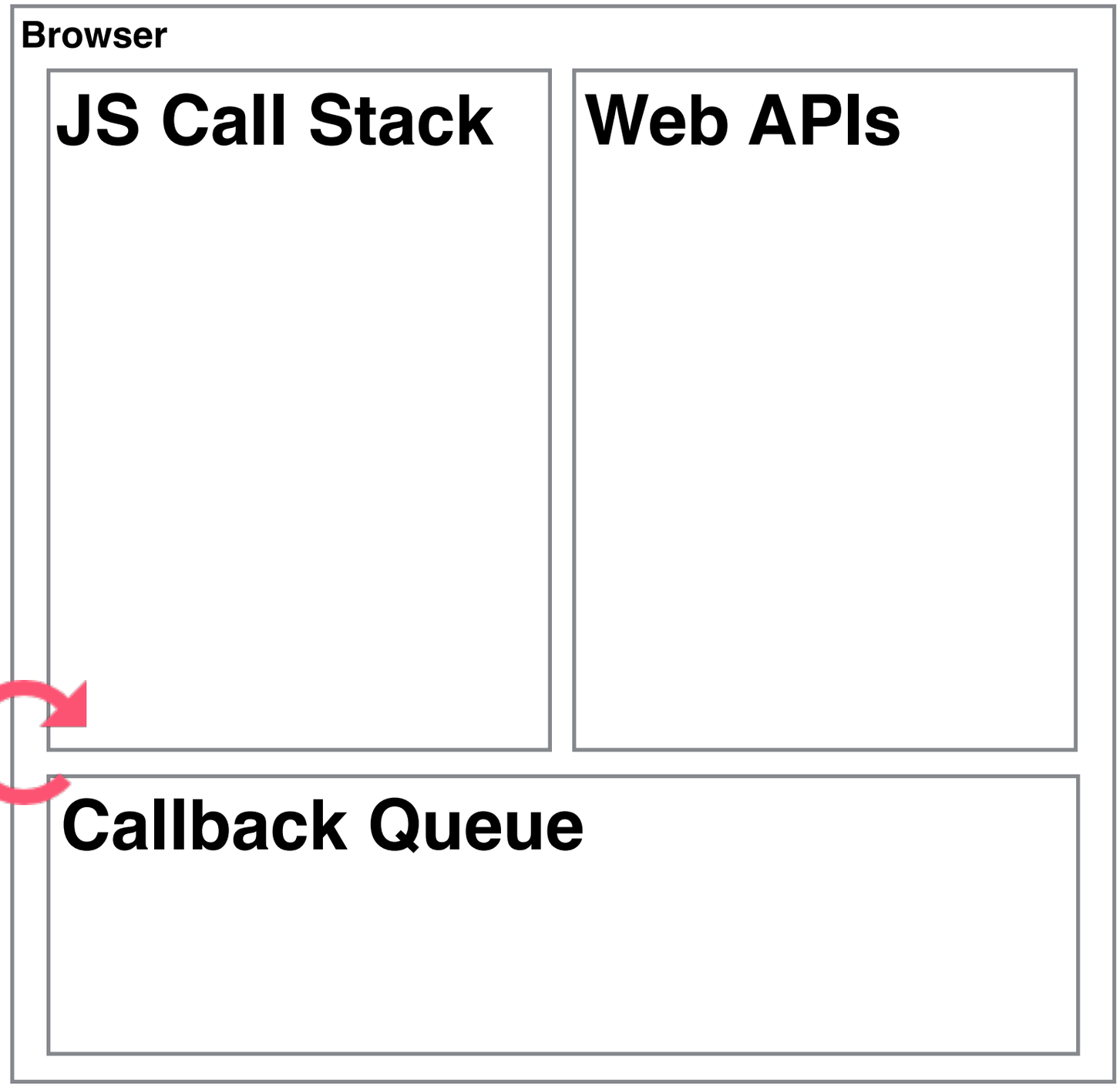


Call Stack & Event Loop 2

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 5000);

console.log(“There”);
```



Call Stack & Event Loop 2

Pour la science, il va se passer quoi ?

```
function getHello() {
  return “Hello”;
}
console.log(getHello());

setTimeout(function abc() {
  console.log(‘abc’);
}, 0);

console.log(“There”);
```

Call Stack & Event Loop 3

- **Traitements différés**
  - Timers
  - Browser events (ex : onload, …)
  - User events (ex : click, …)
  - Réseau : **Ajax**

Call Stack & Event Loop 4

- **LIMITES**
  - Pas de vraie garantie sur les temps
  - Slow Script Warning :
    - Quand un script prend trop de temps sans laisser la main à l'Event Loop
    - —> message du navigateur pour demander à l'utilisateur s'il veut killer le script

Sécurité

Ajax

Sécurité 1

- **XMLHttpRequest :**
  - Ne peut faire des requêtes HTTP **que** sur le domaine d'où la page a été chargée
  - Same Origin Policy

Sécurité 2

- Same-Origin Policy : restreint la capacité du navigateur à utiliser des ressources sur une autre origine que celle de l'appel.

- L'origine = protocole (ex : http, https) + nom de domaine + le port.

- http://www.mesalbums.com/albums

- http://www.mesalbums.com/albums/1/photos/3

C'est OK. Mais avec :

- http**s**://www.mesalbums.com**:8080**/albums/1/photos/3

C'est plus bon.

Sécurité 3

- **Exceptions :**
  - <img>
  - <link>
  - <script>

Sécurité 4

- Solutions : JSONP, CORS
- JSONP : JSON with Padding -> permet de récupérer une ressource sur une autre origine avec passage de callback, mais uniquement des requêtes GET, pas pratique.
- CORS (XMLHttpRequest**2**) : Cross-Origin Ressource Sharing :
  - plus sécurisé
  - passe par les HEADERs pour échanger les droits de communication entre le serveur et le client
  - Toutes les actions HTTPs **si** elles sont autorisées par le serveur (GET, POST, PUT, PATCH, DELETE, **OPTIONS :** requête de PREFLIGHT)

Sécurité 5

CORS & Headers

Request	Response
Origin	Access-Control-Allow-Origin
Access-Control-Request-Method	Access-Control-Allow-Method
Access-Control-Request-Headers	Access-Control-Allow-Headers

Sécurité 6

- Pour le FrontEnd, il n’y a pas de souci. Tous les navigateurs récents envoient les headers et la requête de preflight pour vous.

- Côté Backend, il faut penser à répondre avec les headers attendus.

Nom	Domaine	Type	Méthode	Schéma	État
allowed	localhost	XHR	OPTIONS	HTTP	200
allowed	localhost	XHR	GET	HTTP	200
info	localhost	XHR	GET	HTTP	200
services	localhost	XHR	OPTIONS	HTTP	200
services	localhost	XHR	GET	HTTP	200
favicon.ico	localhost	XHR	GET	HTTP	200