Асинхронность в Javascript

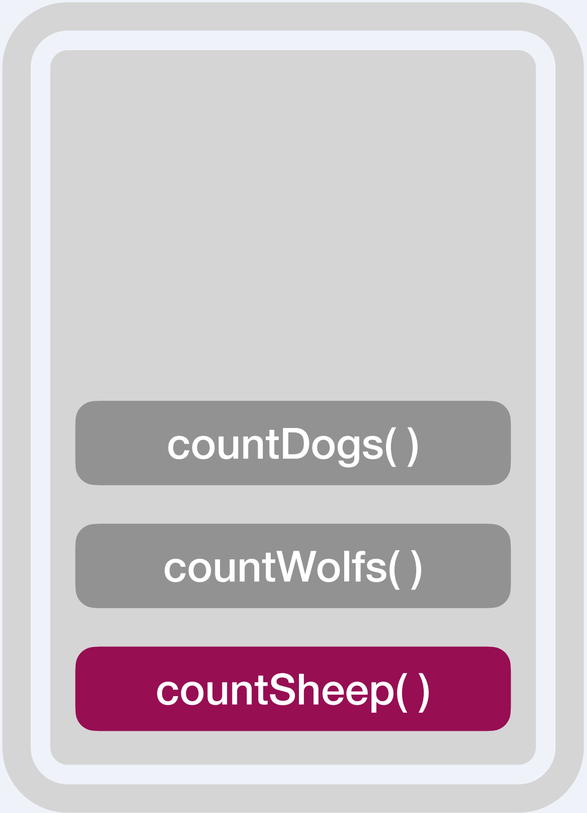
- Call-stack
- Timing functions
- Dom events
- Network requests

< Teach
Me
Skills / >

Урок

Синхронный Javascript

- **Javascript является однопоточным языком.**
- **В каждый момент времени может выполняться только одна команда, обрабатываемая в единственном — главном потоке.**
- **Все остальные действия блокируются до окончания выполнения текущей команды.**



Call-stack

< Teach
Me
Skills / >

Проблема синхронного кода

Шаг 1: Делаем запрос на сервер чтобы получить картинку

Шаг 2: Добавляем картинку в DOM дерево

Шаг 3: Ошибка !!!

```
// Псевдокод 🙈

const image = fetch('image.webp')


printImage(image)
```

Мы никогда не знаем сколько точно потребуется времени чтобы получить картинку, а вот Javascript сразу начнет выполнять следующую строку

< Teach Me Skills / >

Асинхронность в Javascript

Чтобы решить проблемы которые в себе несет синхронный код браузеры предоставляют нам функции и API, которые позволяют нам запускать функции асинхронно.

Функции задержки времени

События DOM

Сетевые запросы

Функции задержки времени

Функция setTimeout позволяет вызвать функцию один раз через определённый промежуток времени.

А функция setInterval периодически вызывает callback функцию через определённый промежуток времени.

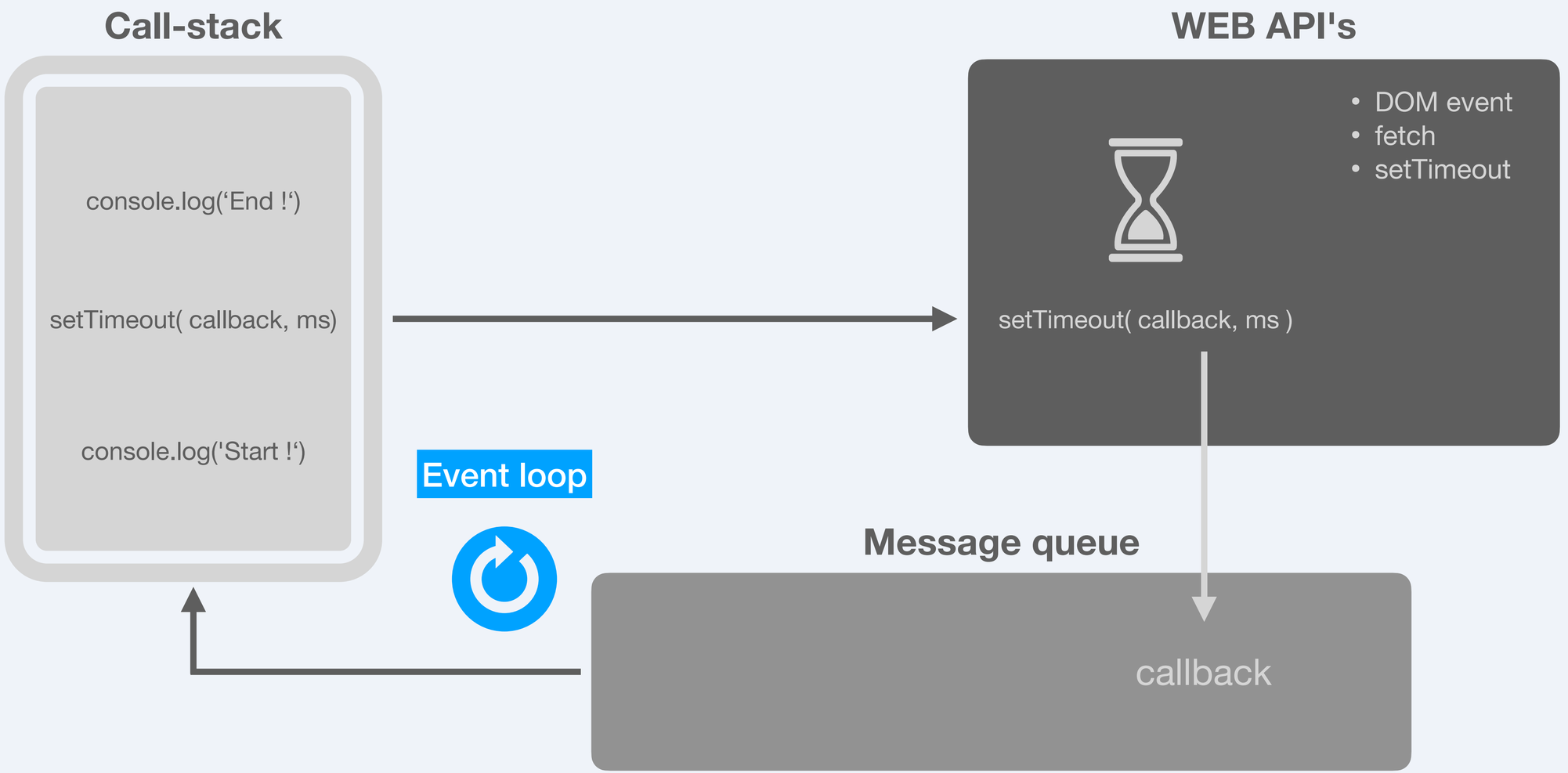
Принимают два аргумента:

- **Функцию callback которую надо вызвать**
- **Задержку в ms**

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


setTimeout(() => {
    console.log('Async Code');
}, 1000)


console.log('End !')

// 'Start !'
// 'End !'
// 'Async Code'
```

< Teach Me Skills / >

Цикл событий



События DOM и цикл событий

Браузерные события работают похожим образом.

Шаг 1: Слушатель событий попадает в WEB API’s и ждет когда сработает событие.

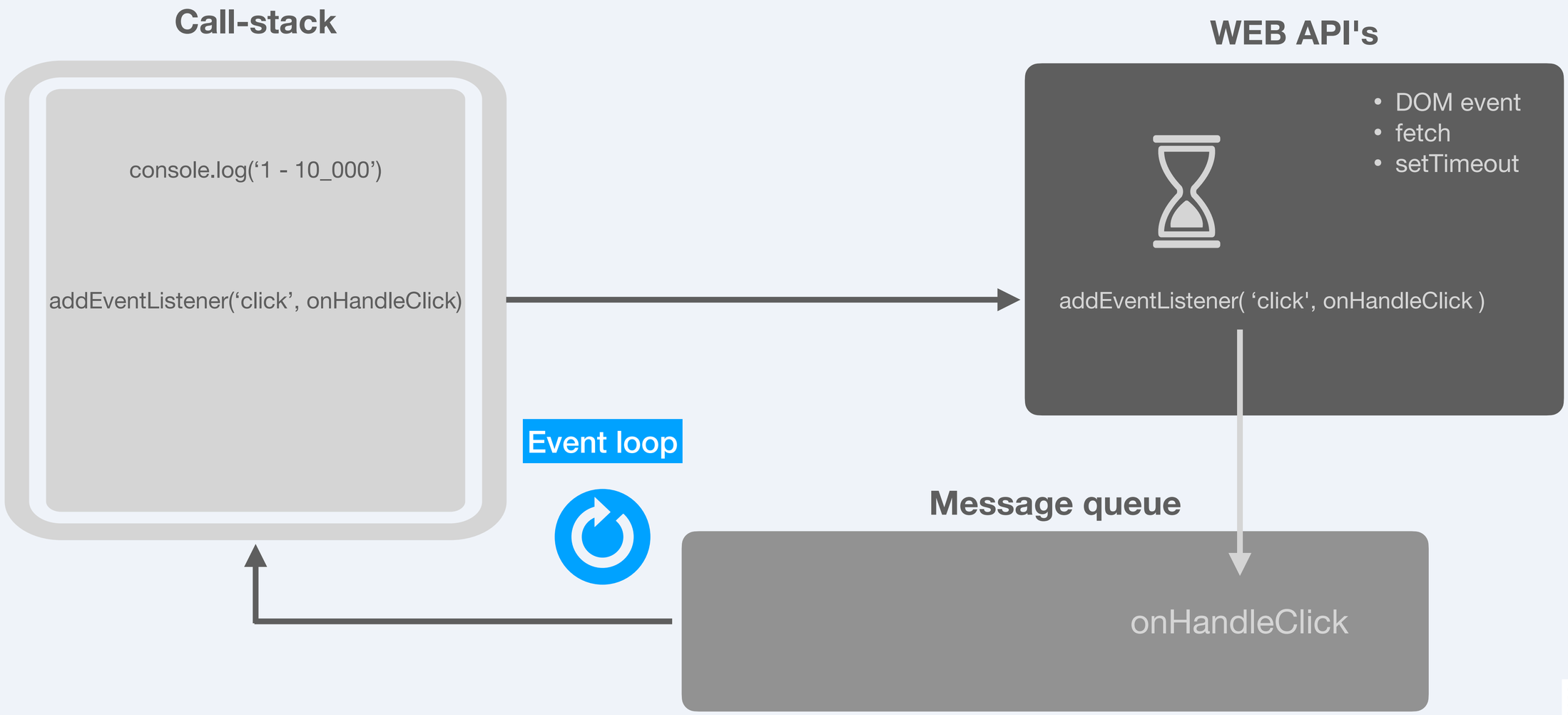
Шаг 2: Как только событие срабатывает - функция callback попадает в message queue.

Шаг 3: В message queue callback ожидает пока за ним придет event loop и переместит ее в call-stack.

```
btn.addEventListener('click', () => {
    console.log('DOM Events !');
})

for (let i = 0; i < 10_000; i++) {
    console.log(i);
}

// 1 … 10_000
// ‘DOM Events !’
```

< Teach Me Skills / >

Цикл событий



Сетевые запросы

Также сетевые запросы работают по похожему алгоритму.

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

fetch('https://xxx.com')

console.log('End Code !');

// 'Start Code !'
// 'End Code !'
// Результат запроса
```