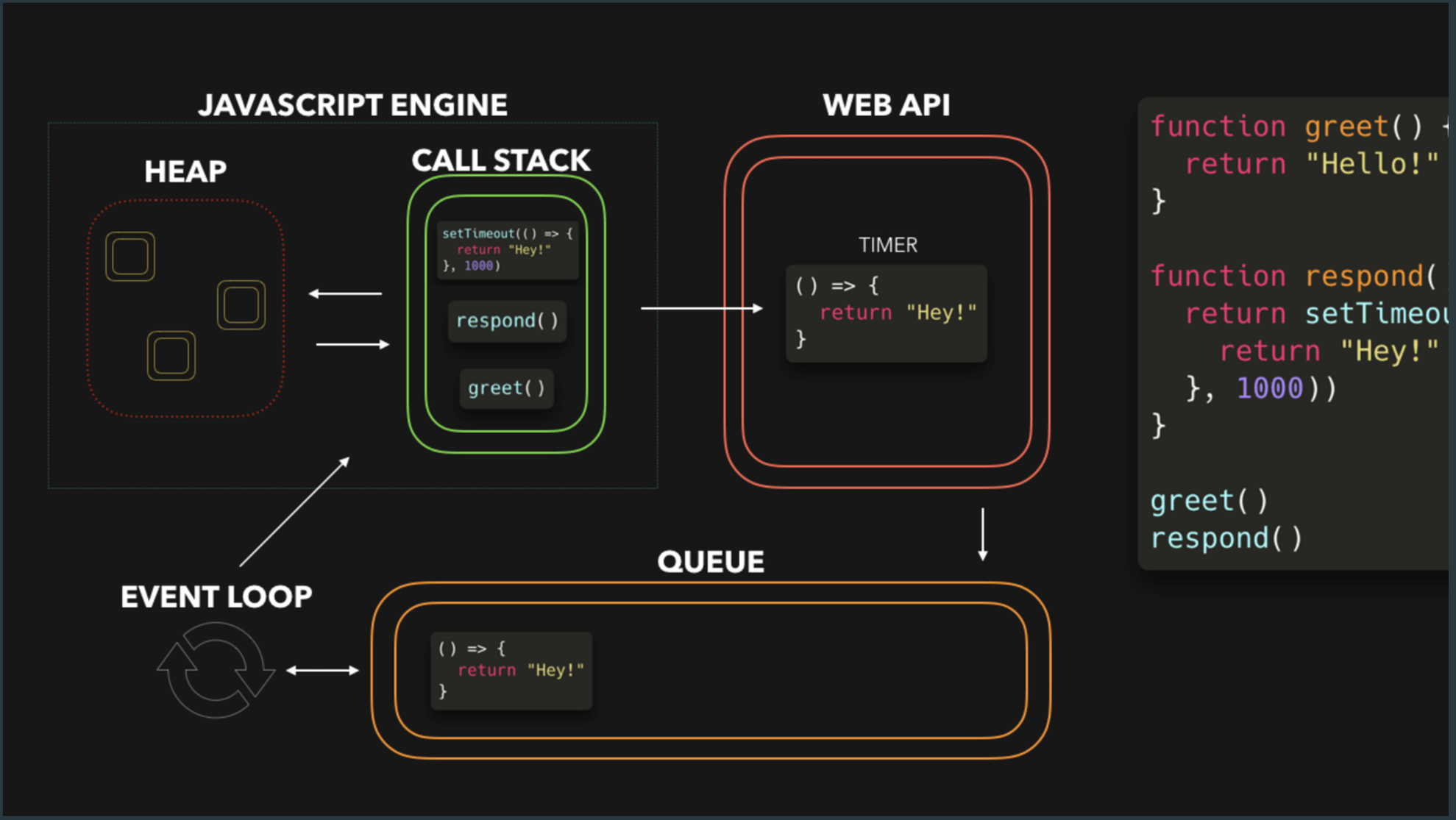
Асинхронность

JavaScript

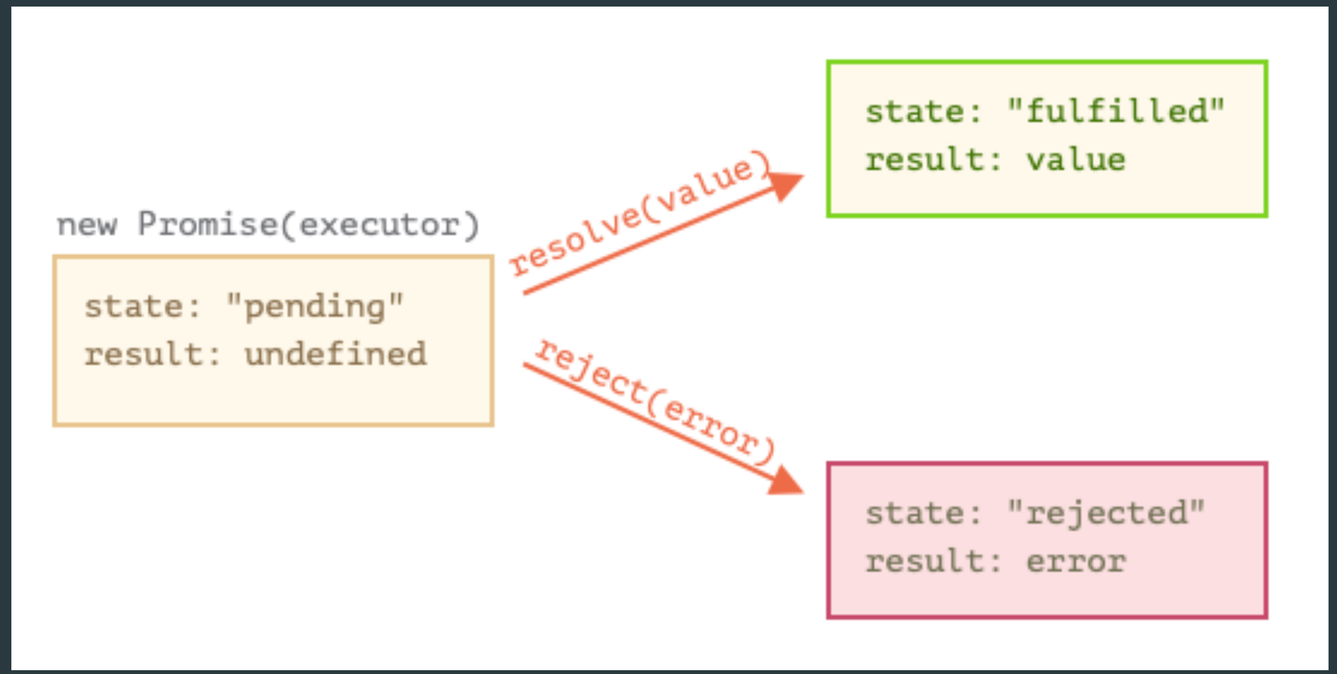
setTimeout & setInterval

EventLoop


JAVASCRIPT ENGINE
HEAP
CALL STACK
setTimeout(() => {
return "Hey!"
}, 1000)
respond()
greet()
WEB API
TIMER
() => {
return "Hey!"
}
QUEUE
() => {
return "Hey!"
}
EVENT LOOP
function greet() {
return "Hello!"
}
function respond() {
return setTimeout(() => {
return "Hey!"
}, 1000))
}
greet()
respond()

Promise

Объект, который представляет окончательное завершение или сбой асинхронной операции и ее результирующее значение



Создание промиса

```
let promise = new Promise(function(resolve, reject) {
  // эта функция выполнится автоматически, при вызове new Promise

  // через 1 секунду сигнализировать, что задача выполнена с результатом "done"
  setTimeout(() ⇒ resolve("done"), 1000);
});
```

Создание промиса с ошибкой

```
let promise = new Promise(function(resolve, reject) {
  // спустя одну секунду будет сообщено, что задача выполнена с ошибкой
  setTimeout(() => reject(new Error("Whoops!")), 1000);  Whoops!
});
```

Потребители: then

```
promise.then(
  function(result) { /* обработает успешное выполнение */ },
  function(error) { /* обработает ошибку */ }
);
```

Потребители: catch

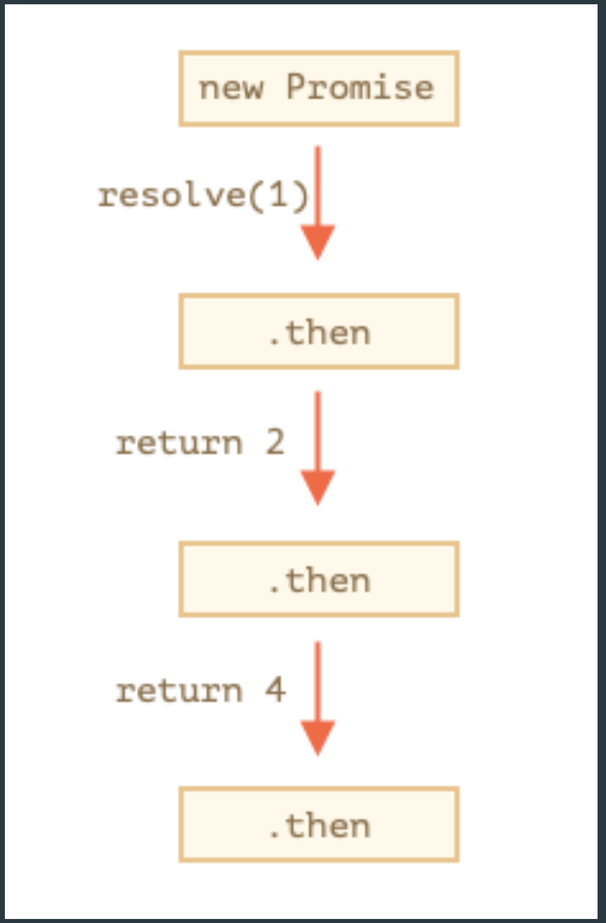
```
const promise = new Promise((resolve, reject) ⇒ {  n/a
  setTimeout(() ⇒ reject(new Error("Ошибка!")), 1000);
});

// .catch(f) это тоже самое, что promise.then(null, f)
promise.catch(alert); // выведет "Error: Ошибка!" спустя одну секунду
```

Потребители: finally

```
const promise = new Promise((resolve, reject) ⇒ {
  resolve('Result')
  /* сделать что-то, что займёт время, и после вызвать resolve/reject */
});


// выполнится, когда промис завершится, независимо от того, успешно или нет
promise
  .finally(() ⇒ { /*остановить индикатор загрузки*/ })
  .then(result ⇒ {}, err ⇒ {})
```

Цепочка промисов

```
new Promise(function(resolve, reject) {  alert is not defined

  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;

});
```



Параллельное исполнение

```
const promises = [
  new Promise(resolve ⇒ resolve('Первый || Promise')),
  new Promise(resolve ⇒ resolve('Второй || Promise')),
]

Promise
 .all(promises)
 .then(res ⇒ console.log(res)); // ["Первый || Promise", "Второй || Promise"]
```

Fetch

```
// GET запрос
let promise = fetch(url);


// POST запрос
let promise = fetch('/article/fetch/post/user', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json;charset=utf-8'
  },
  body: JSON.stringify(user)
});
```

Async/await

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

f().then(console.log); // 1
```

=>

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

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

Await

```
// работает только внутри async-функций
const value = await promise;


function f() {
  const promise = Promise.resolve(1);
  const result = await promise;          // SyntaxError
}
```

Await

Нельзя использовать на верхнем уровне вложенности

```
// SyntaxError на верхнем уровне вложенности
let response = await fetch('/article/promise-chaining/user.json');
let user = await response.json();

(async () => {
  const response = await fetch('/article/promise-chaining/user.json');
  const user = await response.json();
})();
```

Обработка ошибок

```
async function f() {
  await Promise.reject(
    new Error("Упс!")
  );
}
```

=>

```
async function f() {
  throw new Error("Упс!");
}
```

Обработка ошибок

```
async function f() {
  try {
    const response = await fetch('/no-user-here');
    const user = await response.json();
  } catch(err) {
    // перехватит любую ошибку в блоке try: и в fetch, и в response.json
    console.log(err);
  }
}

f();
```