Front-end Advances

Asynchronous JavaScript

Table of Contents

Section 1

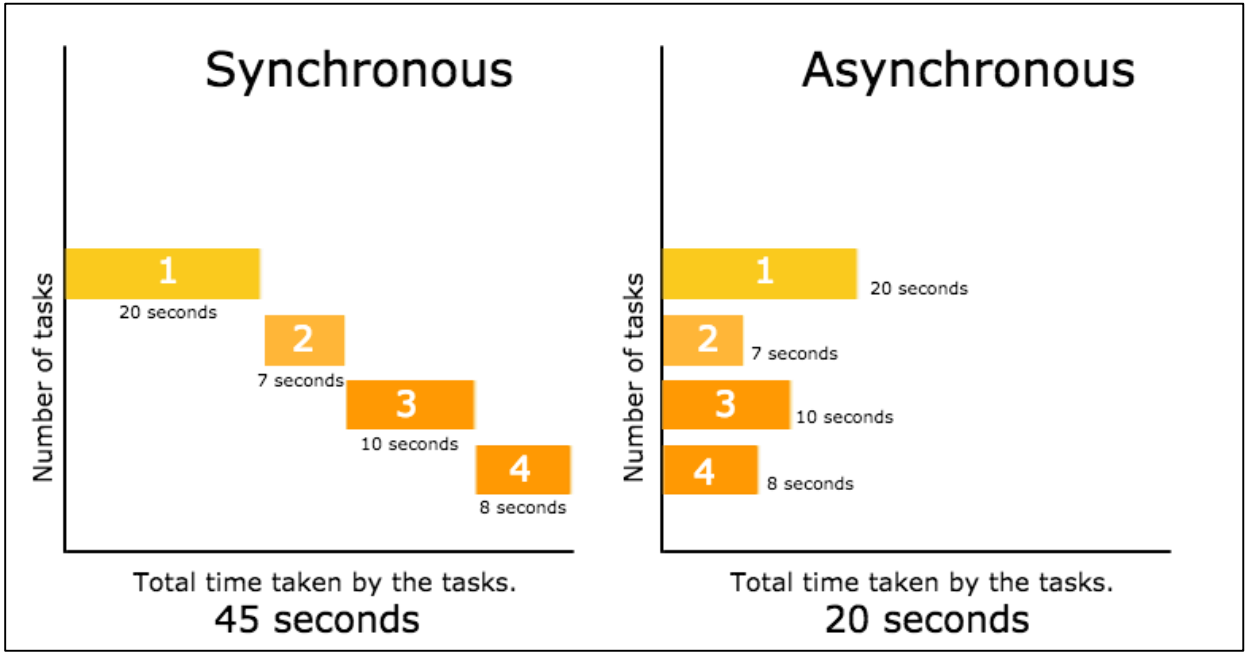
Asynchronous Overview

Asynchronous Overview

- What is a thread ?

Asynchronous Overview

- Synchronous model vs Asynchronous model



Asynchronous Overview

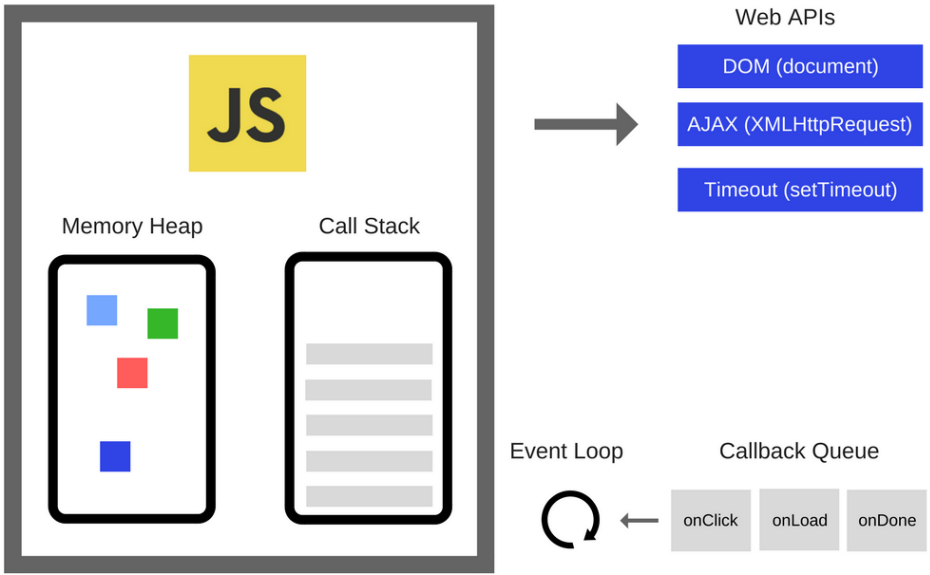
- Problem: Which programming model is JavaScript (synchronous or asynchronous ?)
  - Sadly JavaScript is **single-threaded**: only one task can run at a time
  - How to achieve concurrency with JavaScript?

Section 2

Event Loop

Event Loop

- JavaScript has a concurrency model based on an **event loop**



Event Loop

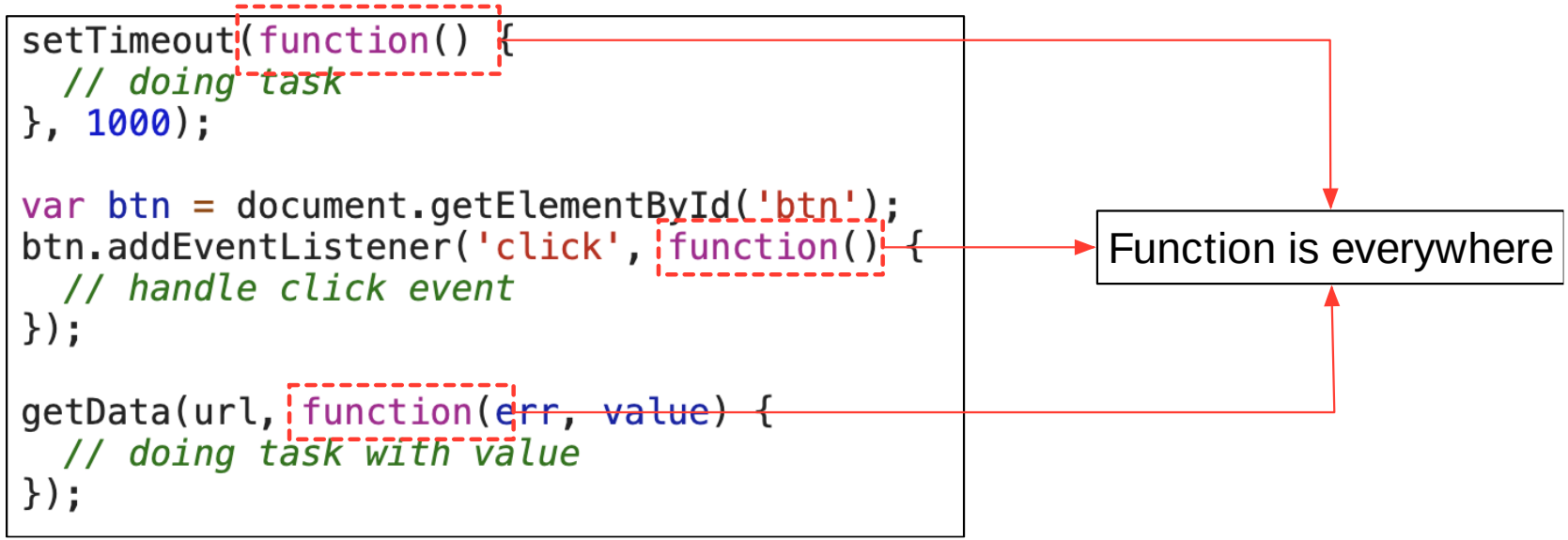
- Practice **setInterval**:
  - Implement the typewriter effect

Section 3

Callback

Callback

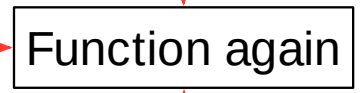
- Function plays a big role in **Event loop**



Callback

- How about this one?

```
var array = [1, 2, 3, 4];

array.map(function(e) {
  return e + 1;
});

array.filter(function(e) {
  return e % 2 === 1;
});

array.reduce(function(acc, e) {
  return acc + e;
});
```



Callback

- What the difference between those 2?

```
var array = [1, 2, 3, 4];

array.map(function(e) {
  return e + 1;
});

array.filter(function(e) {
  return e % 2 === 1;
});

array.reduce(function(acc, e) {
  return acc + e;
});
```

```
setTimeout(function() {
  // doing task
}, 1000);

var btn = document.getElementById('btn');
btn.addEventListener('click', function() {
  // handle click event
});

getData(url, function(err, value) {
  // doing task with value
});
```

Callback

- What the difference between those 2?

sync callback

```
var array = [1, 2, 3, 4];

array.map(function(e) {
  return e + 1;
});

array.filter(function(e) {
  return e % 2 === 1;
});

array.reduce(function(acc, e) {
  return acc + e;
});
```

async callback

```
setTimeout(function() {
  // doing task
}, 1000);

var btn = document.getElementById('btn');
btn.addEventListener('click', function() {
  // handle click event
});

getData(url, function(err, value) {
  // doing task with value
});
```

Callback

- Recall AJAX and JSON

```
function getData(url, cb) {
  var xhr = new XMLHttpRequest();

  xhr.onreadystatechange = function () {
    if (xhr.readyState == XMLHttpRequest.DONE) {
      // Change here
      if (xhr.status === 200) {
        cb(undefined, JSON.parse(xhr.responseText));
      } else {
        cb(new Error(xhr.statusText));
      }
    }
  };

  xhr.open('GET', url, true);
  xhr.send();
}
```

Callback

- Recall AJAX and JSON – Usage get data from server

```
// Sample code
var url = 'https://jsonplaceholder.typicode.com/todos/';

getData(url + 1, function (err, value) {
  if (err) {
    return console.log(err);
  }

  console.log(value);
});
// {userId: 1, id: 1, title: "delectus aut autem", completed: false}
```

Callback

- How to handle error?

sync callback

```
var array = [1, 2, 3, 4];

try {
  array.map(function (e) {
    if (e === 3) {
      throw new Error('Value is 3');
    }
    return e + 1;
  });
} catch (error) {
  console.log(error);
}
```

```
Error: Value is 3
    at <anonymous>:9:13
    at Array.map (<anonymous>)
    at <anonymous>:7:9
```

Easy

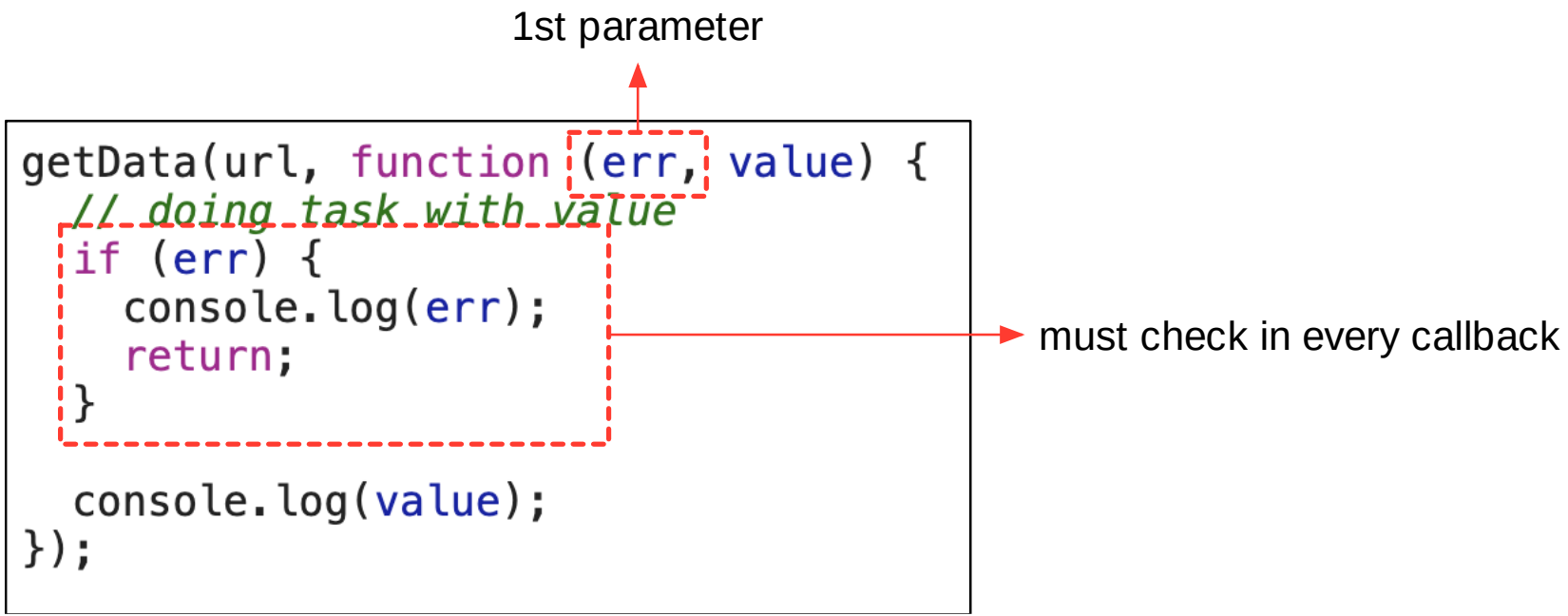
async callback

```
try {
  setTimeout(function () {
    throw new Error('Something went wrong');
  }, 1000);
} catch (err) {
  console.log(err);
}
```

```
2
▶ Uncaught Error: Something went wrong
    at <anonymous>:5:11
```

Callback

- How to handle error? Use **error** 1st callback style



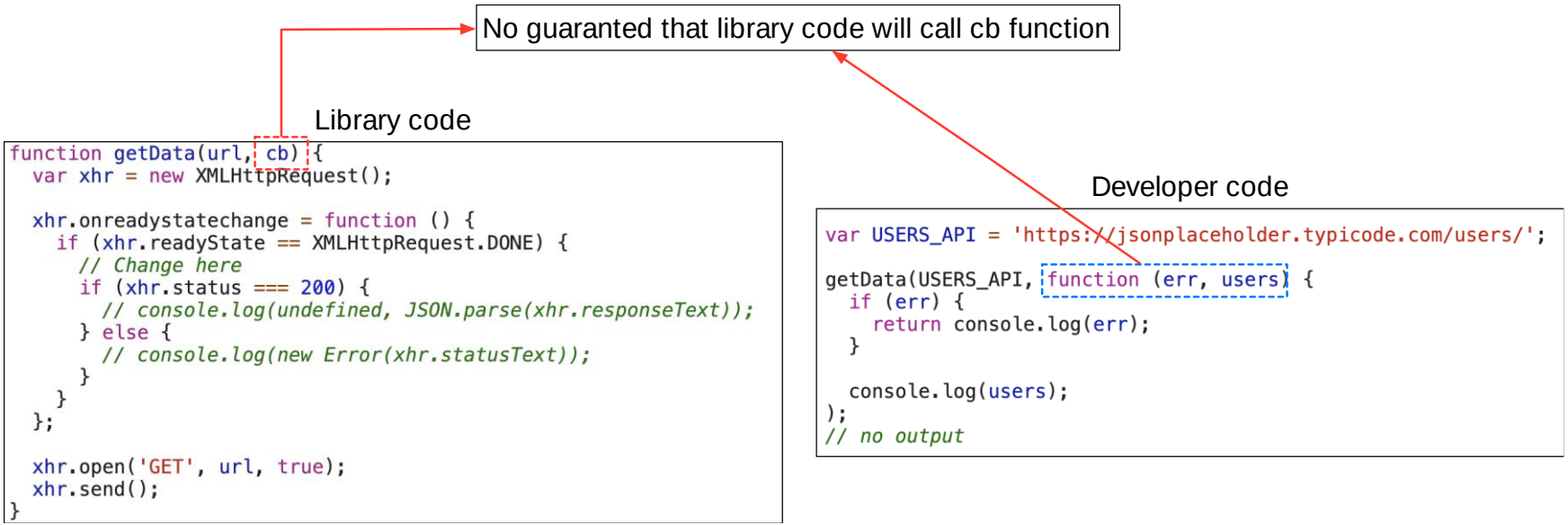
Callback – Draw back

- What if we want to request many resouces from server?

```
var USERS_API = 'https://jsonplaceholder.typicode.com/users/';
var POSTS_API = 'https://jsonplaceholder.typicode.com/posts/';

// get all users
getData(USERS_API, function (err, users) {
  if (err) {
    return console.log(err);
  }

  // get detail of 1st user
  getData(USERS_API + users[0].id, function (err, user) {
    if (err) {
      return console.log(err);
    }

    // get post by user id
    getData(POSTS_API + user.id, function (err, posts) {
      if (err) {
        return console.log(err);
      }
    });
  });

  console.log(value);
});
```

Callback – Draw back

- Why my callback is not running ?



Callback

- Callback function is core mechanism behind Event Loop
- There are 2 types of callback: sync and async
- Callback has 3 main disadvantages:
  1. There is no guaranted that callback function is called exactly 1 (unless you use built-in or well-known library)
  2. Hard to handle error in async callback
  3. Coding styling is ugly when work with multiple callback

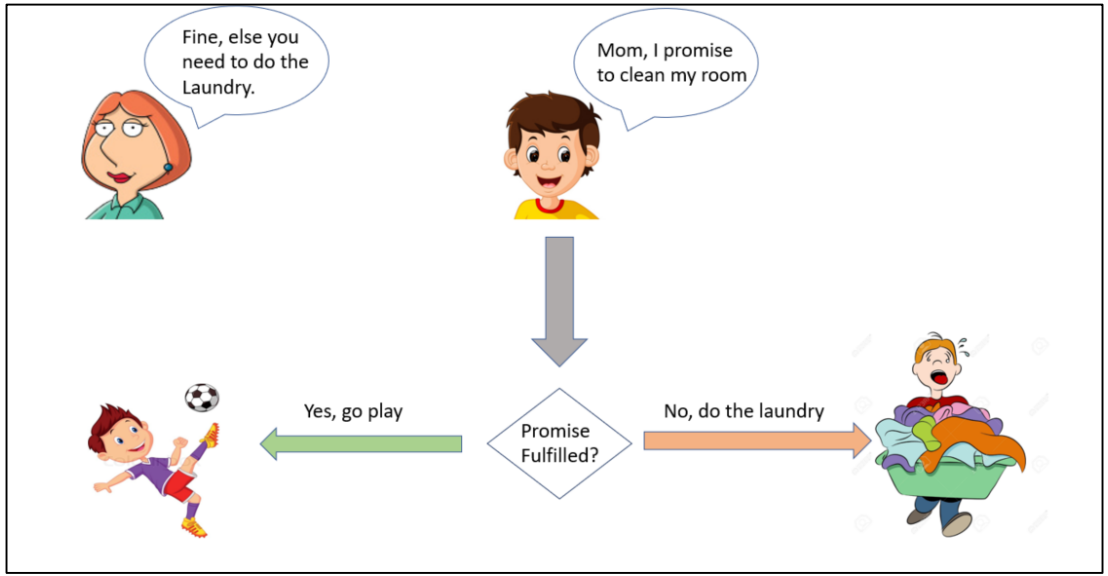
Section 4

Promise

Promise

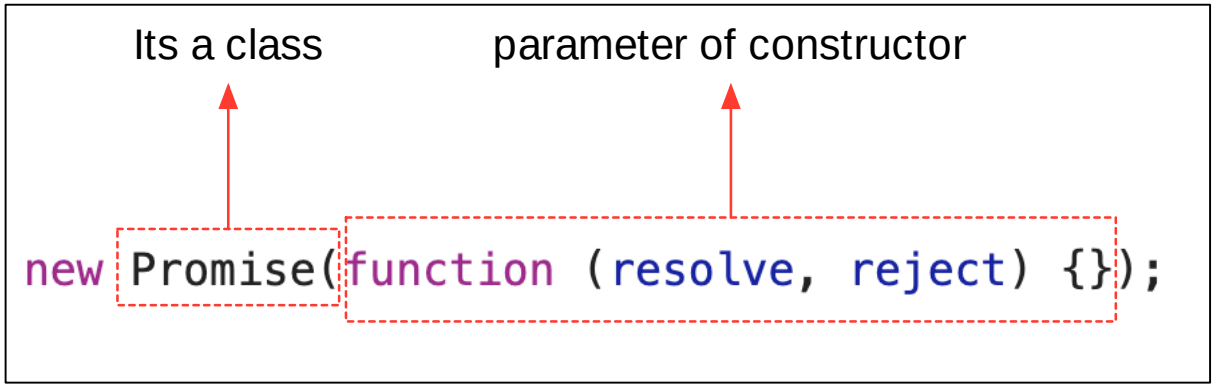
- What is a **Promise**?

  Promise object representing the eventual **completion** (**fulfil**) or **failure** (**reject**) of an asynchronous operation.



Promise

- How JS represent a **Promise** ?



Promise

- Usage: represent network request with **Promise**

```
var url = 'https://jsonplaceholder.typicode.com/todos/';

var p = new Promise(function (resolve, reject) {
  getData(url + '1', function (err, value) {
    if (err) {
      return reject(err);
    }

    resolve(value);
  });
});
```

Promise

- Usage: how to get the value of that network request ?

```
var url = 'https://jsonplaceholder.typicode.com/todos/';

var p = new Promise(function (resolve, reject) {
  getData(url + '1', function (err, value) {
    if (err) {
      return reject(err);
    }

    resolve(value);
  });
});

p.then(function (data) {
  console.log(data);
});
```

the callback function will receive the data of asynchronous operation

Use .then() and give it a callback function

Promise

- Usage: how to handle error?

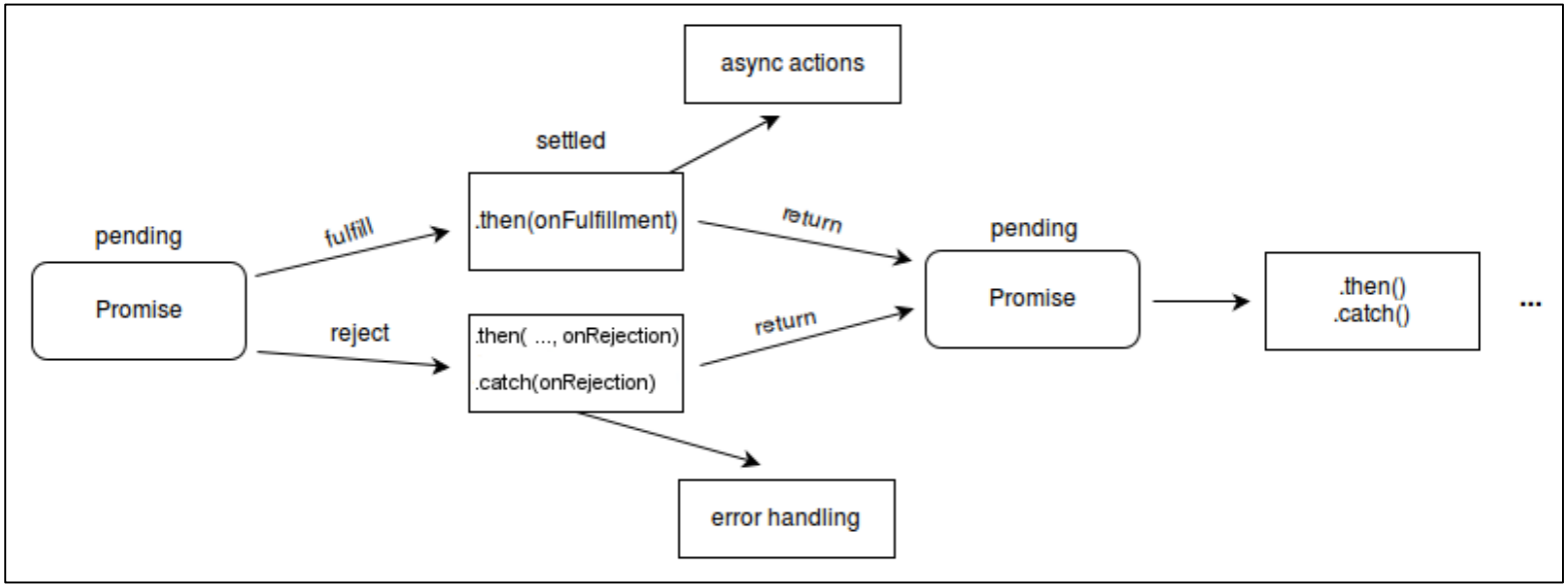
```
var url = 'https://jsonplaceholder.typicode.com/todos/';

var p = new Promise(function (resolve, reject) {
  getData(url + '0', function (err, value) {
    if (err) {
      return reject(err);
    }

    resolve(value);
  });
});

p.then(
  function (data) {
    console.log(data);
  },
  function (err) {
    console.log(err);
  }
);
```

There is no todos/0

2nd parameter of .then is used to handle error

Promise

- Promise API:



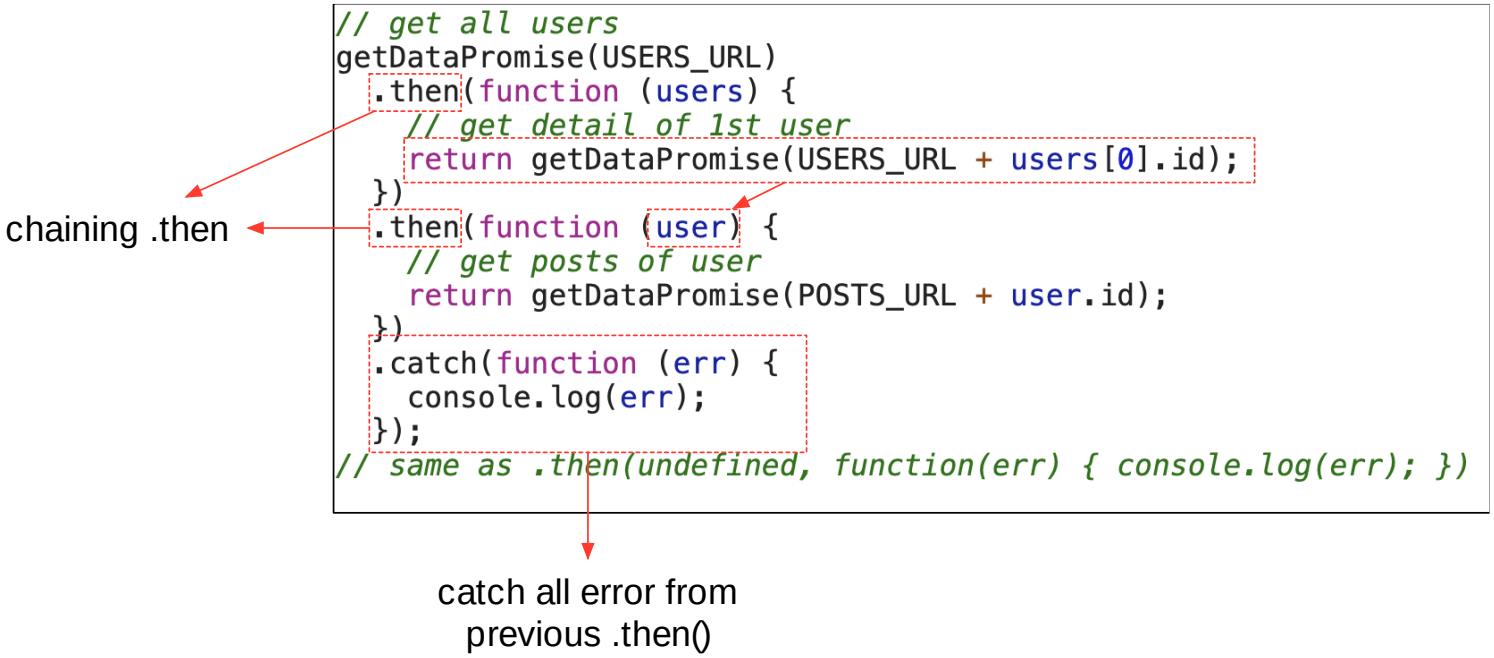
Promise

- Promise Usage:

```
var USERS_URL = 'https://jsonplaceholder.typicode.com/users/';
var POSTS_URL = 'https://jsonplaceholder.typicode.com/posts/';

function getDataPromise(url) {
  return new Promise(function (resolve, reject) {
    getData(url, function (err, value) {
      if (err) {
        reject(err);
        return;
      }

      resolve(value);
    });
  });
}
```

Promise

- Promise Usage:



Promise

- Handle concurrent requests:

```
var p1 = getDataPromise(USERS_URL + 1); // Promise<User1>
var p2 = getDataPromise(USERS_URL + 2); // Promise<User2>
var p3 = getDataPromise(USERS_URL + 3); // Promise<User3>

Promise.all([p1, p2, p3]).then(function (users) {
  console.log(users); // [User1, User2, User3];
});
```

3 independent requests

use Promise.all on array of Promise

Promise

- Promise advantages:
  1. Callback is guaranted to executed (exactly 1)
  2. Built-in error handling mechanism
  3. Coding style is OK (not like callback)

Promise

- Promise disadvantages:

```
getDataPromise(USERS_URL)
  .then(function (users) {
    return getDataPromise(USERS_URL + users[0].id);
  })
  .then(function (user) {
    return getDataPromise(POSTS_URL + user.id);
  })
  .then(function (posts) {
    console.log(posts);
  })
  .catch(function (err) {
    console.log(err);
  });
```

can't resuse users

Section 5

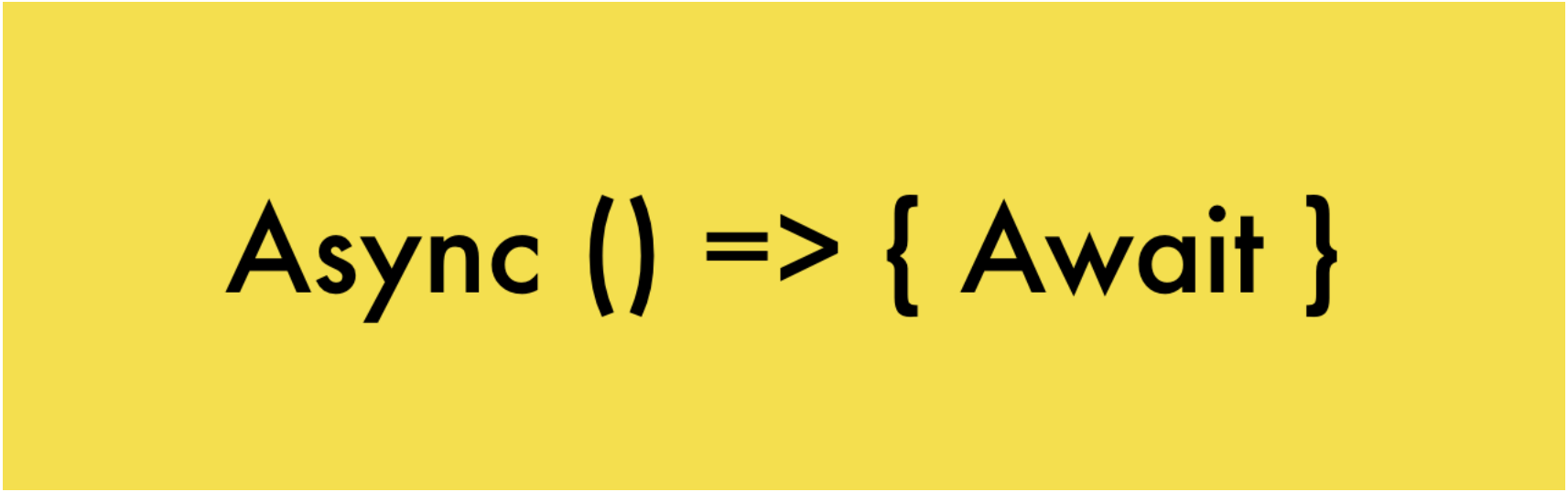
Generator

Section 6

async/await

async/await

- Promise is nice but its contructor/syntax is still too hard ? Is there only mechanism to handle async operation ?



async/await

- Syntax:

```
async function main() {
  console.log('before');
  var t = await getDataPromise(USERS_URL + 1);
  console.log('after');

  console.log(t); // Promise<User1>
}

main();
console.log('end')
// before
// end
// after
// User1
```

async/await

- async keyword:

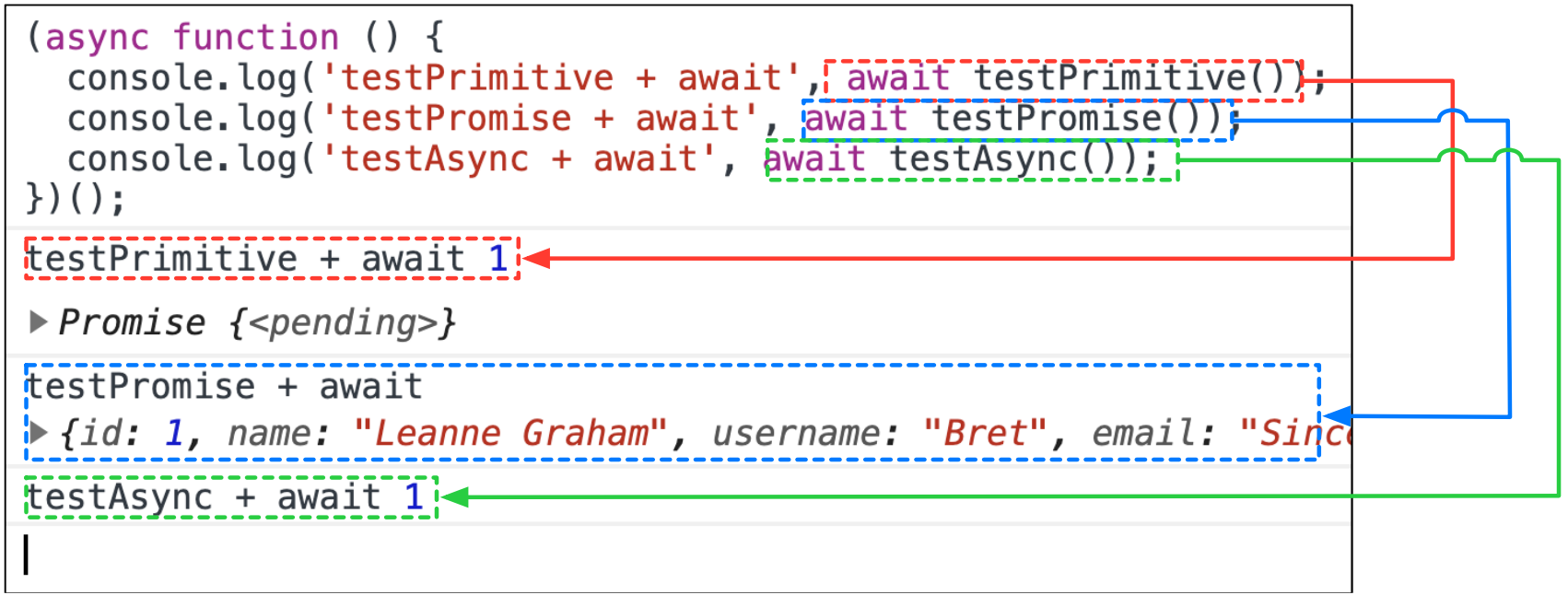
```
async function testPrimitive() {
  return 1;
}

async function testPromise() {
  return getDataPromise(USERS_URL + 1); // Promise<User1>
}

async function testAsync() {
  return testPrimitive(); // Promise<1>
}

console.log('testPrimitive', testPrimitive());
console.log('testPromise', testPromise());
console.log('testAsync', testAsync());
```

```
testPrimitive ▶ Promise {<resolved>: 1}
testPromise ▶ Promise {<pending>}
testAsync ▶ Promise {<pending>}
```

How to print the value inside Promise ?

async/await

- async keyword:



async/await

- Handle error ?

```
async function main() {
  try {
    var u1 = await getDataPromise(USERS_URL + 1);
    console.log(u1); // Promise<User1>

    var u2 = await getDataPromise(USERS_URL + 2);
    console.log(u2); // Promise<User2>
  } catch (e) {
    console.log(e);
  }
}

main();
```

Easy, just like sync code

async/await

- But can we **await** concurrent request ?

```
async function main() {
  try {
    var u1 = getDataPromise(USERS_URL + 1);
    var u2 = getDataPromise(USERS_URL + 2);
    var u3 = getDataPromise(USERS_URL + 2);

    var users = await [u1, u2, u3]; //
    console.log(users); // [Promise, Promise, Promise]

  } catch (e) {
    console.log(e);
  }
}

main();
```

Not working

async/await

```
async function main() {
  try {
    var u1 = getDataPromise(USERS_URL + 1);
    var u2 = getDataPromise(USERS_URL + 2);
    var u3 = getDataPromise(USERS_URL + 2);

    var users = await Promise.all([u1, u2, u3]);
    console.log(users);

  } catch (e) {
    console.log(e);
  }
}

main();
```

just wrap in Promise.all

console log

```
(3) [{…}, {…}, {…}]
▶ 0: {id: 1, name: "Leanne Graham", username:
▶ 1: {id: 2, name: "Ervin Howell", username: '
▶ 2: {id: 2, name: "Ervin Howell", username: '
  length: 3
▶ __proto__: Array(0)
```

Summary

- **JavaScript** is single-thread, synchronous programming language
- **Browsers** add asynchronous (concurrent) model to **JavaScript** via **Event Loop**
- **Functions** play **big** role in Callback style
- **Promise and async/await (recommended)** are created to solve the problem of Callback

Thank you