

Day 3

Asynchronous JavaScript

Resources

- **JavaScript Run-time:**

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop>

- **Promises:**

<https://developers.google.com/web/fundamentals/primers/promises>

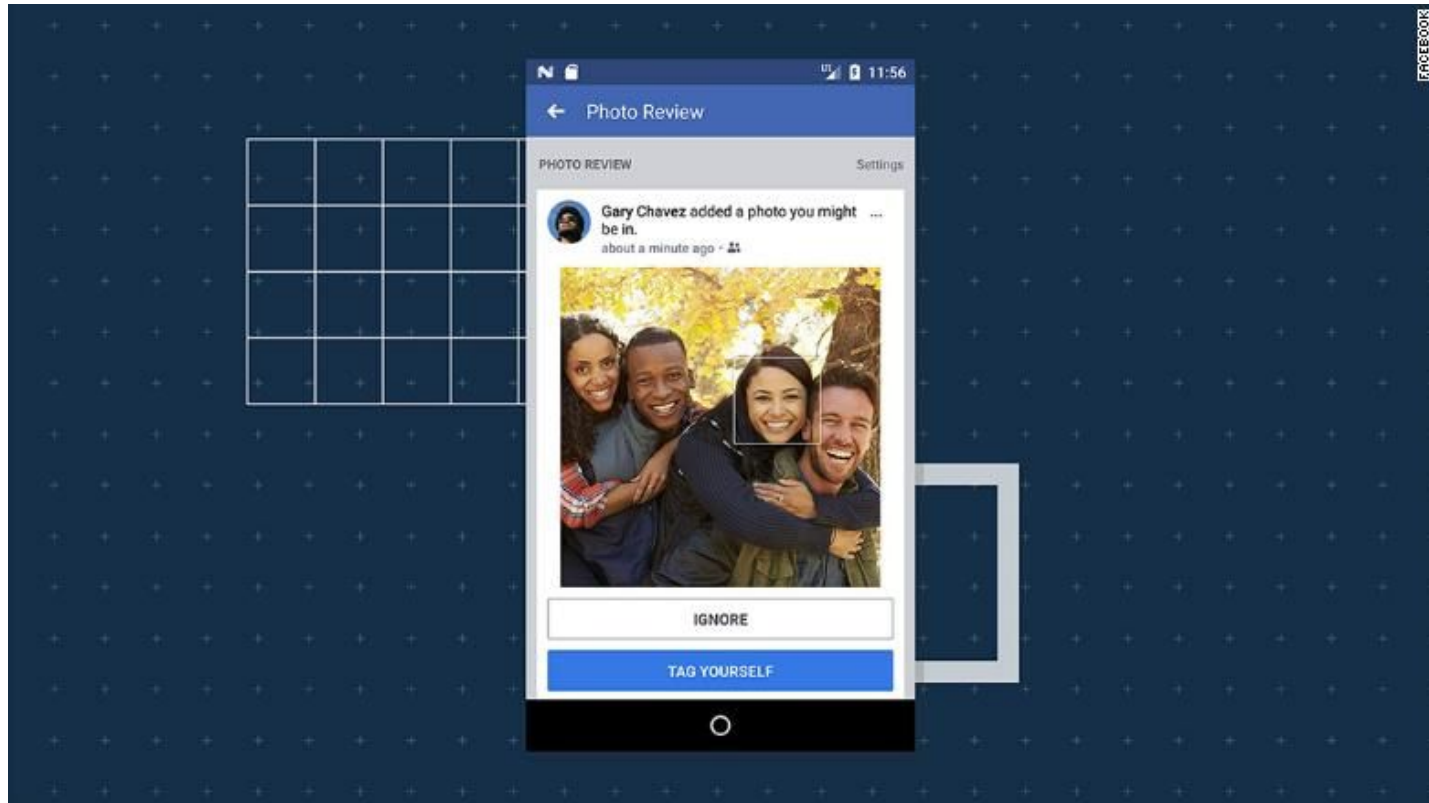
- **AngularJS promises:**

<https://thinkster.io/a-better-way-to-learn-angularjs/promises>

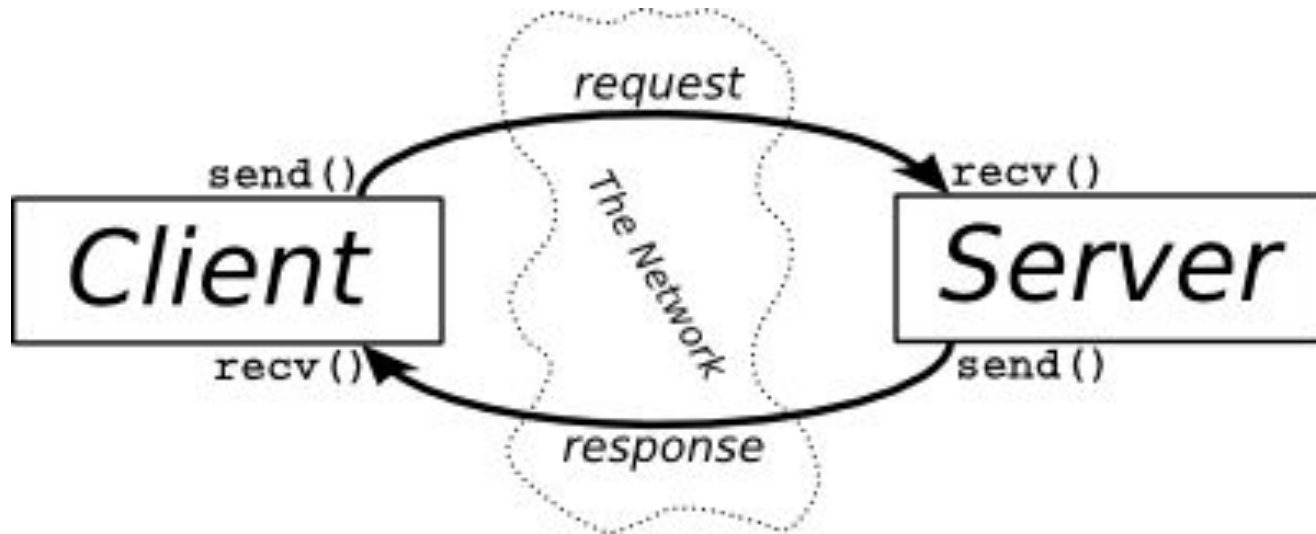
- **Async Javascript - callbacks vs. promises:**

<https://www.pluralsight.com/guides/introduction-to-asynchronous-javascript>

Common Scenario



Client-server interaction



Let's see it in action

The screenshot shows a Google Slides presentation titled "Case study" in the top left. The presentation has six slides. Slide 1 is titled "Day 3" and has the subtitle "Represent JavaScript". Slide 2 is titled "Resources" and lists several links. Slide 3 is titled "Common Scenario" and shows a diagram of a client-server interaction. Slide 4 is titled "Client-server interaction" and shows a diagram of a client-server interaction. Slide 5 is titled "Let's see it in action" and is currently selected. Slide 6 is titled "The problem" and has a subtitle "The problem".

The main content area of the presentation shows the text "Let's see it in action" on a blue background. Below the main content area, there is a text box that says "Click to add speaker notes".

The Chrome DevTools Network tab is open on the right side of the screen. It shows a list of network requests. The first request is a sync request to a Google Cloud Storage bucket. The second request is a script request to a Google Cloud Storage bucket. The third request is a font request to a Google Cloud Storage bucket. The fourth request is a script request to a Google Cloud Storage bucket. The fifth request is a font request to a Google Cloud Storage bucket. The sixth request is a script request to a Google Cloud Storage bucket. The seventh request is a font request to a Google Cloud Storage bucket. The eighth request is a script request to a Google Cloud Storage bucket. The ninth request is a font request to a Google Cloud Storage bucket. The tenth request is a script request to a Google Cloud Storage bucket. The eleventh request is a font request to a Google Cloud Storage bucket. The twelfth request is a script request to a Google Cloud Storage bucket. The thirteenth request is a font request to a Google Cloud Storage bucket. The fourteenth request is a script request to a Google Cloud Storage bucket. The fifteenth request is a font request to a Google Cloud Storage bucket. The sixteenth request is a script request to a Google Cloud Storage bucket. The seventeenth request is a font request to a Google Cloud Storage bucket. The eighteenth request is a script request to a Google Cloud Storage bucket. The nineteenth request is a font request to a Google Cloud Storage bucket. The twentieth request is a script request to a Google Cloud Storage bucket. The twenty-first request is a font request to a Google Cloud Storage bucket. The twenty-second request is a script request to a Google Cloud Storage bucket. The twenty-third request is a font request to a Google Cloud Storage bucket. The twenty-fourth request is a script request to a Google Cloud Storage bucket. The twenty-fifth request is a font request to a Google Cloud Storage bucket. The twenty-sixth request is a script request to a Google Cloud Storage bucket. The twenty-seventh request is a font request to a Google Cloud Storage bucket. The twenty-eighth request is a script request to a Google Cloud Storage bucket. The twenty-ninth request is a font request to a Google Cloud Storage bucket. The thirtieth request is a script request to a Google Cloud Storage bucket. The thirty-first request is a font request to a Google Cloud Storage bucket. The thirty-second request is a script request to a Google Cloud Storage bucket. The thirty-third request is a font request to a Google Cloud Storage bucket. The thirty-fourth request is a script request to a Google Cloud Storage bucket. The thirty-fifth request is a font request to a Google Cloud Storage bucket. The thirty-sixth request is a script request to a Google Cloud Storage bucket. The thirty-seventh request is a font request to a Google Cloud Storage bucket. The thirty-eighth request is a script request to a Google Cloud Storage bucket. The thirty-ninth request is a font request to a Google Cloud Storage bucket. The fortieth request is a script request to a Google Cloud Storage bucket. The forty-first request is a font request to a Google Cloud Storage bucket. The forty-second request is a script request to a Google Cloud Storage bucket. The forty-third request is a font request to a Google Cloud Storage bucket. The forty-fourth request is a script request to a Google Cloud Storage bucket. The forty-fifth request is a font request to a Google Cloud Storage bucket. The forty-sixth request is a script request to a Google Cloud Storage bucket. The forty-seventh request is a font request to a Google Cloud Storage bucket. The forty-eighth request is a script request to a Google Cloud Storage bucket. The forty-ninth request is a font request to a Google Cloud Storage bucket. The fiftieth request is a script request to a Google Cloud Storage bucket. The fifty-first request is a font request to a Google Cloud Storage bucket. The fifty-second request is a script request to a Google Cloud Storage bucket. The fifty-third request is a font request to a Google Cloud Storage bucket. The fifty-fourth request is a script request to a Google Cloud Storage bucket. The fifty-fifth request is a font request to a Google Cloud Storage bucket. The fifty-sixth request is a script request to a Google Cloud Storage bucket. The fifty-seventh request is a font request to a Google Cloud Storage bucket. The fifty-eighth request is a script request to a Google Cloud Storage bucket. The fifty-ninth request is a font request to a Google Cloud Storage bucket. The sixtieth request is a script request to a Google Cloud Storage bucket. The sixty-first request is a font request to a Google Cloud Storage bucket. The sixty-second request is a script request to a Google Cloud Storage bucket. The sixty-third request is a font request to a Google Cloud Storage bucket. The sixty-fourth request is a script request to a Google Cloud Storage bucket. The sixty-fifth request is a font request to a Google Cloud Storage bucket. The sixty-sixth request is a script request to a Google Cloud Storage bucket. The sixty-seventh request is a font request to a Google Cloud Storage bucket. The sixty-eighth request is a script request to a Google Cloud Storage bucket. The sixty-ninth request is a font request to a Google Cloud Storage bucket. The seventieth request is a script request to a Google Cloud Storage bucket. The seventy-first request is a font request to a Google Cloud Storage bucket. The seventy-second request is a script request to a Google Cloud Storage bucket. The seventy-third request is a font request to a Google Cloud Storage bucket. The seventy-fourth request is a script request to a Google Cloud Storage bucket. The seventy-fifth request is a font request to a Google Cloud Storage bucket. The seventy-sixth request is a script request to a Google Cloud Storage bucket. The seventy-seventh request is a font request to a Google Cloud Storage bucket. The seventy-eighth request is a script request to a Google Cloud Storage bucket. The seventy-ninth request is a font request to a Google Cloud Storage bucket. The eightieth request is a script request to a Google Cloud Storage bucket. The eighty-first request is a font request to a Google Cloud Storage bucket. The eighty-second request is a script request to a Google Cloud Storage bucket. The eighty-third request is a font request to a Google Cloud Storage bucket. The eighty-fourth request is a script request to a Google Cloud Storage bucket. The eighty-fifth request is a font request to a Google Cloud Storage bucket. The eighty-sixth request is a script request to a Google Cloud Storage bucket. The eighty-seventh request is a font request to a Google Cloud Storage bucket. The eighty-eighth request is a script request to a Google Cloud Storage bucket. The eighty-ninth request is a font request to a Google Cloud Storage bucket. The ninetieth request is a script request to a Google Cloud Storage bucket. The ninety-first request is a font request to a Google Cloud Storage bucket. The ninety-second request is a script request to a Google Cloud Storage bucket. The ninety-third request is a font request to a Google Cloud Storage bucket. The ninety-fourth request is a script request to a Google Cloud Storage bucket. The ninety-fifth request is a font request to a Google Cloud Storage bucket. The ninety-sixth request is a script request to a Google Cloud Storage bucket. The ninety-seventh request is a font request to a Google Cloud Storage bucket. The ninety-eighth request is a script request to a Google Cloud Storage bucket. The ninety-ninth request is a font request to a Google Cloud Storage bucket. The hundredth request is a script request to a Google Cloud Storage bucket.

Name	Status	Type	Initiator	Size	Waterfall	10.00 s
sync?id=AAHRpnXtYlcZ...	200	xhr	57917...	4...		
cb=gapi.loaded_0	200	s...	google...	1...		
4iCs6KVjbNBVigo6IQTxv...	200	font	39285...	(fr...		
4iCu6KVjbNBVigoKeg72...	200	font	39285...	(fr...		
4iCv6KVjbNBVigoCvTs...	200	font	39285...	(fr...		
4iCp6KVjbNBVigoKegZPs...	200	font	39285...	(fr...		
1pd78XzPZEhJ4FClyFtAb...	200	xhr	cb=98...	6...		
3_uITNA7unv0UtplybPip...	200	font	39285...	(fr...		
dazS1PrQQuCxC3iOAJF...	200	font	39285...	(fr...		
78z90OcXkV8RXGqwfia5...	200	font	39285...	(fr...		
0ADruTddSoprgUk9PVA?...	200	xhr	cb=98...	4...		
jfk_sprite_hdpi124.png	200	png	Other	8...		
logImpressions?id=1pd7...	204	xhr	39285...	3...		

153 requests | 2.5 MB transferred | Finish: 12.02 s | DOMContentLoaded: 2.30 s | Load: 2.62 s

Console What's New x Coverage Remote devices

Let's see it in action

▼ General

Request URL: https://docs.google.com/comments/u/104949472398518478618/d/AAHRpnXtYIcZW0DAZY09EtZkzoRW_N06Xcr_kk2pZ1YH9aKe7taVLcd05ZEN1ATPCLgh3Y3ZgZEpXDRhWvYG2X8V9hyFrngWj0WurNARNcXyXGxqfLJ-OMU/docs/p/sync?id=AAHRpnXtYIcZW0DAZY09EtZkzoRW_N06Xcr_kk2pZ1YH9aKe7taVLcd05ZEN1ATPCLgh3Y3ZgZEpXDRhWvYG2X8V9hyFrngWj0WurNARNcXyXGxqfLJ-OMU&sid=1a2ca1736e9e0df2&c=0&w=0&smv=4&token=AGNctVbEzr6F0u10603tPQLSHU2nJx1TWg%3A1525874857883

Request Method: POST

Status Code:  200

Remote Address: 172.217.13.110:443

Referrer Policy: no-referrer-when-downgrade

▼ Response Headers

alt-svc: hq=":443"; ma=2592000; quic=51303433; quic=51303432; quic=51303431; quic=51303339; quic=51303335, quic=":443"; ma=2592000; v="43,42,41,39,35"

cache-control: no-cache, no-store, max-age=0, must-revalidate

content-disposition: attachment; filename="response.bin"; filename*=UTF-8''response.bin

content-encoding: gzip

content-type: application/json; charset=utf-8

date: Wed, 09 May 2018 14:07:46 GMT

expires: Mon, 01 Jan 1990 00:00:00 GMT

pragma: no-cache

server: GSE

set-cookie: S=comments=gyIKjyb3r0UWPn0sH0he0y-byjDQ0nEA; Domain=.docs.google.com; Expires=Wed, 09-May-2018 14:22:46 GMT; Path=/comments/u/104949472398518478618/d/AAHRpnXtYIcZW0DAZY09EtZkzoRW_N06Xcr_kk2pZ1YH9aKe7taVLcd05ZEN1ATPCLgh3Y3ZgZEpXDRhWvYG2X8V9hyFrngWj0WurNARNcXyXGxqfLJ-OMU; Secure; HttpOnly; Priority=LOW

set-cookie: SIDCC=AEfoLeYe4XNG5YchWU1uFhsp-ECLPNrHJLI-9o_YFgawAceEv6hWslyc5B2wYSRAroJoJImEoFGX; expires=Tue, 07-Aug-2018 14:07:46 GMT; path=/; domain=.google.com; priority=high

status: 200

Let's see it in action

Name		×	Headers	Preview	Response	Cookies	Timing
 sync?id=AAHRpnXtYlcZWODAZY... /comments/u/1049494723985184...		1)]]'		
		2					
		3			[["sr", []		
 cb=gapi.loaded_0 apis.google.com/_/scs/abc-static/...		4			,1525874266808]		
		5]		

Requirements

- A browser is constantly loading data dynamically.
- We would like to have a “non-blocking” UI which always offers the user interaction, even as data is getting prepared in the background.

Requirements

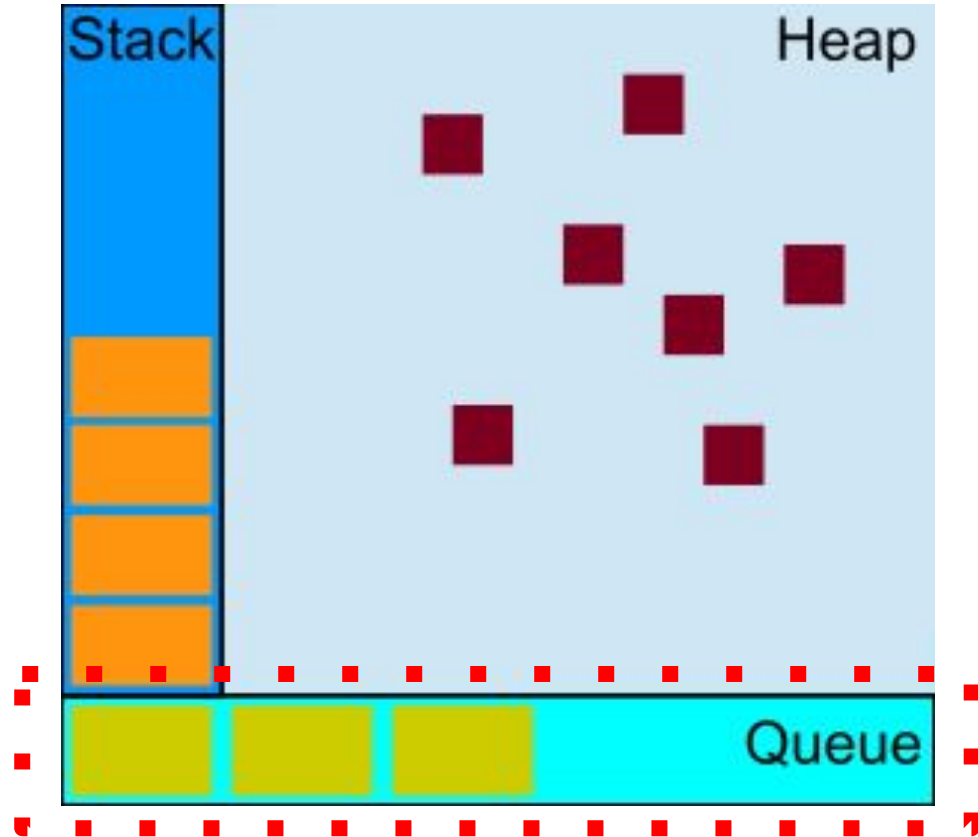
- A browser is constantly loading data dynamically.
- We would like to have a “non-blocking” UI which always offers the user interaction, even as data is getting prepared in the background.

Solution

- A language whose semantic model is built to accommodate this nature. This is where **JavaScript** comes in.

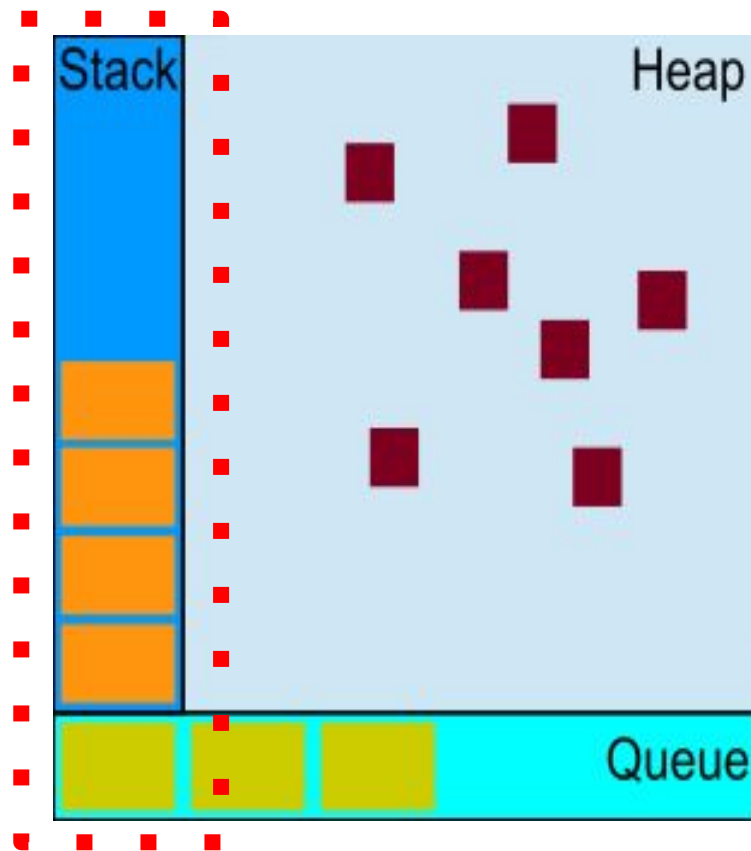
JavaScript run-time

The Run-time



The stack

- Controls actual execution in JavaScript
- Functions are pushed onto it as execution progresses.
- Only one function is executed at a time
- If a function is long-lasting it **will** block the UI
- Code executes as you would expect, **in order**.



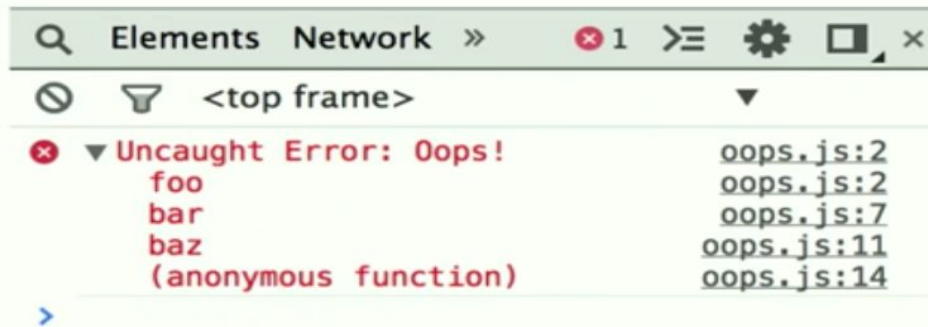
The stack

```
function foo() {  
  throw new Error('Oops!');  
}
```

```
function bar() {  
  foo();  
}
```

```
function baz() {  
  bar();  
}
```

```
baz();
```



X RangeError: Maximum call stack size exceeded

Asynchronous code

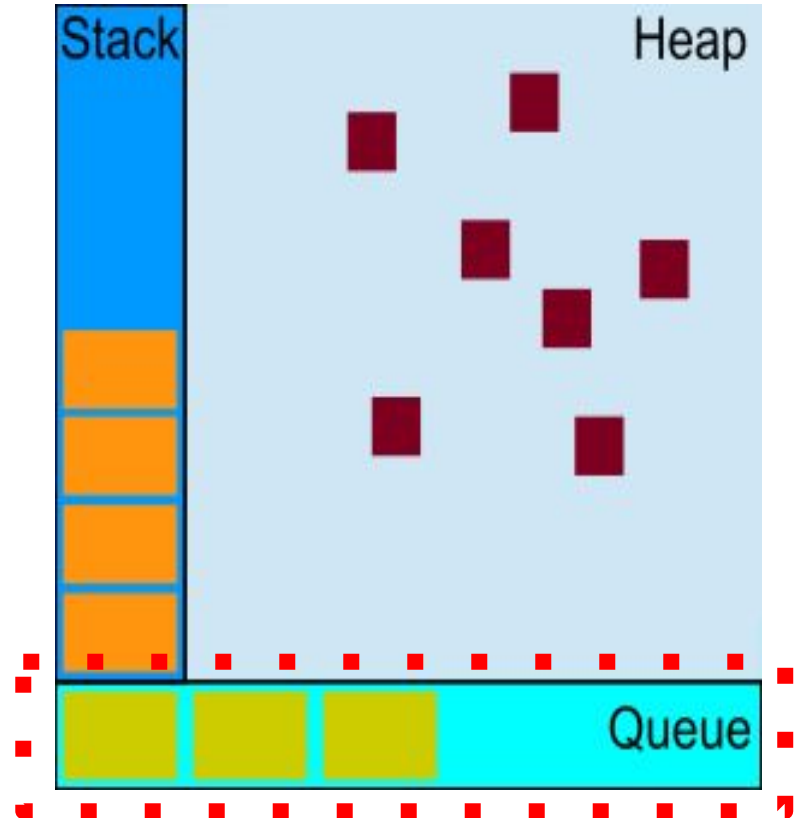
```
function foo()
{
    console.log('foo')
}
function bar()
{
    console.log('bar')
}
function getImage()
{
    fetch("image.png")
        .then(function(image_response){
            console.log("image");
        });
}
// Execution
foo();
getImage();
bar();
// Output
/*
* foo
* bar
* image
*/
```

Callback

```
.then(function(image_response){
    console.log("image");
});
```

The queue

- The **queue** keeps track of all the **callbacks** in asynchronous requests.
- Waits for the stack to be empty before requesting to place the **callback** back onto the stack to be executed



Handling Async code

The problem

- We have many asynchronous call made continuously or at the same time. How do we handle this in JavaScript?

Solution

- Many ways to do this in JavaScript
 - Events and callbacks
 - **Promises**
 - await/sync
- **AngularJS** uses promises, we will focus on this one.
- You can transform all these representations into one another. Is about **expressivity** and **clarity!!!**
- Reference:
<https://www.pluralsight.com/guides/introduction-to-asynchronous-javascript>

What is a promise?

- **Definition:** A promise represents the eventual result of an asynchronous operation. It is a placeholder into which the successful result value or reason for failure will materialize
- A promise contains three states:
 - Pending
 - Resolved
 - Failed

```
// Simple GET request example:  
$http({  
  method: 'GET',  
  url: '/someUrl'  
}).then(function successCallback(response) {  
  // this callback will be called asynchronously  
  // when the response is available  
}, function errorCallback(response) {  
  // called asynchronously if an error occurs  
  // or server returns response with an error status.  
});
```

How do we *promisify*?

Promisifying: Converting async code into a promise.

Procedure

- Wrap the async code around a promise.
- In AngularJS we use the `$q` dependency.
- Reference:
 - [https://docs.angularjs.org/api/ng/service/\\$q](https://docs.angularjs.org/api/ng/service/$q)

```
// Suppose function okToGreet exists
function asyncGreet(name) {
    var deferred = $q.defer();

    setTimeout(function() {
        if (okToGreet(name)) {
            deferred.resolve('Hello, ' + name + '!');
        } else {
            deferred.reject('Greeting ' + name +
                ' is not allowed.');
        }
    }, 1000);
    return deferred.promise;
}
```

Opal promise creation example

```
function requestToServer(request, params)
{
  var deferred = $q.defer();
  var db = firebase.database();
  var key = db.set("request", {"name": request, parameters: params});
  db.ref("response"+"/"+key).once("value", function(snapshot){
    deferred.resolve(snapshot.value());
  }).catch(function(err){
    deferred.reject(err);
  });
  return deferred.promise;
}
```

How do we call a promise?

- Once we have *'promisified'* a function, how do we call it?
 - Use the **then/catch** promise semantics

```
// Suppose function okToGreet exists
function asyncGreet(name) {
    var deferred = $.defer();

    setTimeout(function() {
        if (okToGreet(name)) {
            deferred.resolve('Hello, ' + name + '!');
        } else {
            deferred.reject('Greeting ' + name +
                ' is not allowed.');
        }
    }, 1000);
    return deferred.promise;
}
```

```
asyncGreet('Robin Hood')
    .then(function(greeting){
        alert('Success: ' + greeting);
    }).catch(function(error){
        alert('Failed: ' + reason);
    });
```

Common Opal promises

- \$http.get
- Firebase
- The whole back-end!

```
// Simple GET request example:
$http({
  method: 'GET',
  url: '/someUrl'
}).then(function successCallback(response) {
  // this callback will be called asynchronously
  // when the response is available
}, function errorCallback(response) {
  // called asynchronously if an error occurs
  // or server returns response with an error status.
});
```

```
requestToServer("GetConversations",{userId:1})
  .then(function(response){
    // Handle conversations
  })
  .catch(function(error){
    // Handle error
  });
```



Common async scenarios

Cases

- **Scenario 1:** One simple async request (Done)
- **Scenario 2:** Two or more simple requests that **depend** on one another
- **Scenario 3:** Two or more simple requests that **do not depend** on one another.
- Every other scenario is a combination of this tree.

Scenario 1

- **Description:** A simple async request
- **Procedure:**
 1. Promisify request (if not promisifyied)
 2. Use **then/catch**

Scenario 1 - Example

```
fetchUrlContent(imageUrl)  
  .then(function(content) {  
  
    }).catch(function(error) {  
  
    });
```

Scenario 2

- **Description:** Two or more simple async requests that depend on one another
- **Procedure:**
 1. Promisify the requests (if not promisifyied)
 2. Chain them one after the other.

Scenario 2 - example

```
// Assume getImages function exists, which fetches  
// the images from conversations  
requestToServer("GetConversations",{userId:1})  
  .then(function(response){  
      
    return getImages(response.data.conversations);  
  }).then(function(conversationsWithImages){  
    // Handle conversations  
  })  
  .catch(function(error){ alert(error); });
```

Scenario 3

- **Description:** Two or more simple async requests that do not depend on one another. (order of arrival does not matter)
- **Procedure**
 1. Promisify requests (if not promisifyied)
 2. Create array of unresolved promises with each image
 3. Use `$q.all()`
- **Common cases:**
 - Fetching images for each of the conversations. These do not depend on one another but must all return before you can use conversations.

Scenario 3 - Example

```
function getImages(conversations){  
  var promiseArray = [];  
  for(var i = 0; i < conversations.length; i++)  
  {  
    promiseArray.push(fetchUrlContent(conversations[i].imageUrl));  
  }  
  return $q.all(promiseArray).then(function(images){  
    images.forEach(function(image,index){  
      conversations[index].image = image;  
    });  
    return images;  
  });  
}
```

Last comments

- You will encounter this repeatedly as a web-developer!
- Read references at the beginning of presentation
- Exercises will come soon!
- I will also put together a README.md giving more context and detail.
- If you master this, you are well on your way with JavaScript :)



The End - Thank you!