Asynchronous JavaScript

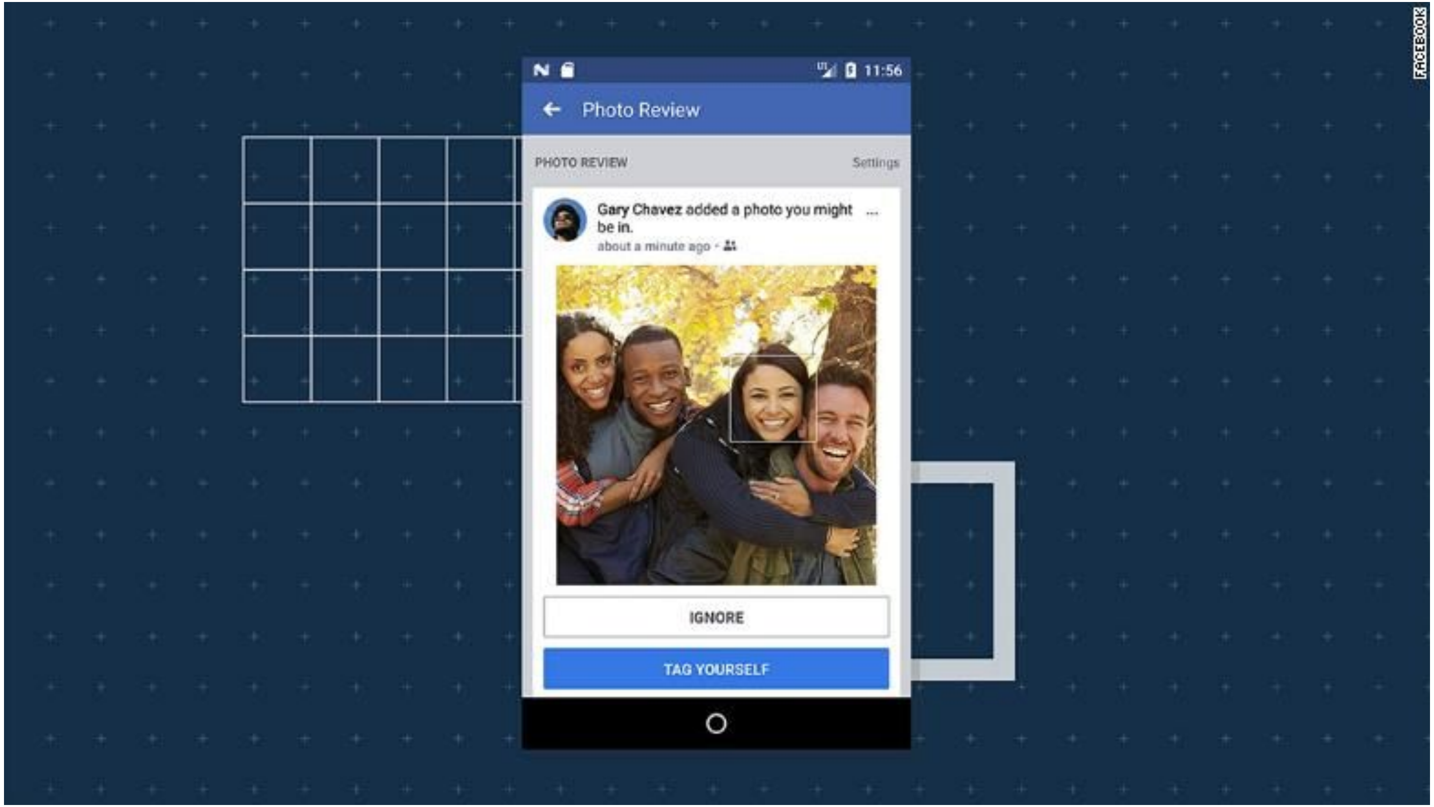
based on slides by David Herrera

Resources

Read the following resources along with these slides:

- [JavaScript Run-time]
- [Promises]
- [AngularJS promises]
- [Async Javascript - Callbacks vs. promises]
- [Promise Chaining]

Common Scenario

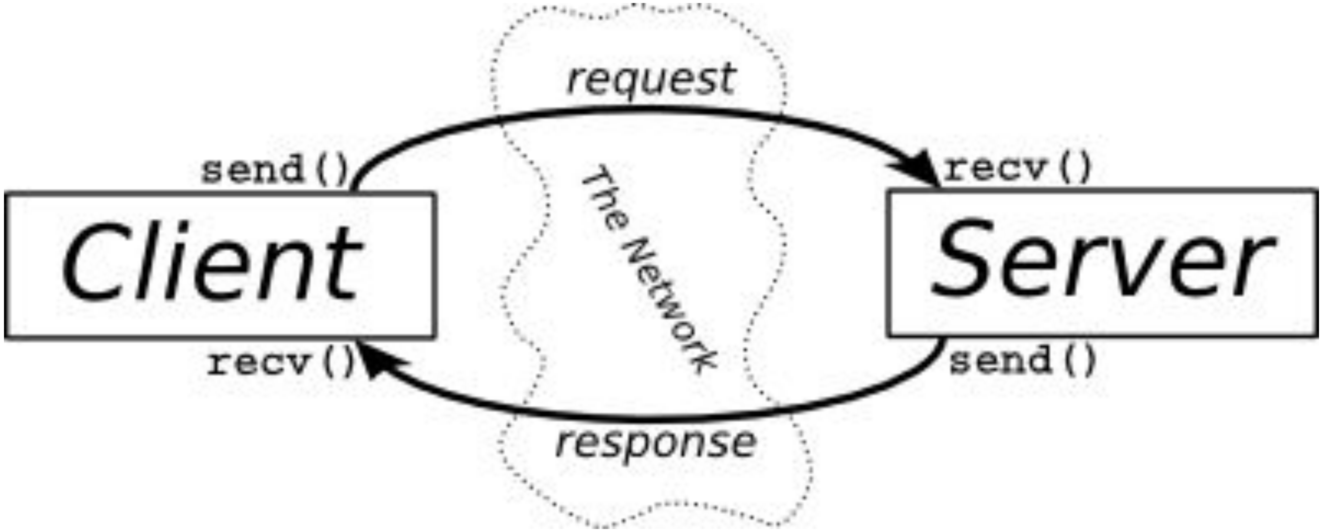


Gary Chavez added a photo you might be in.

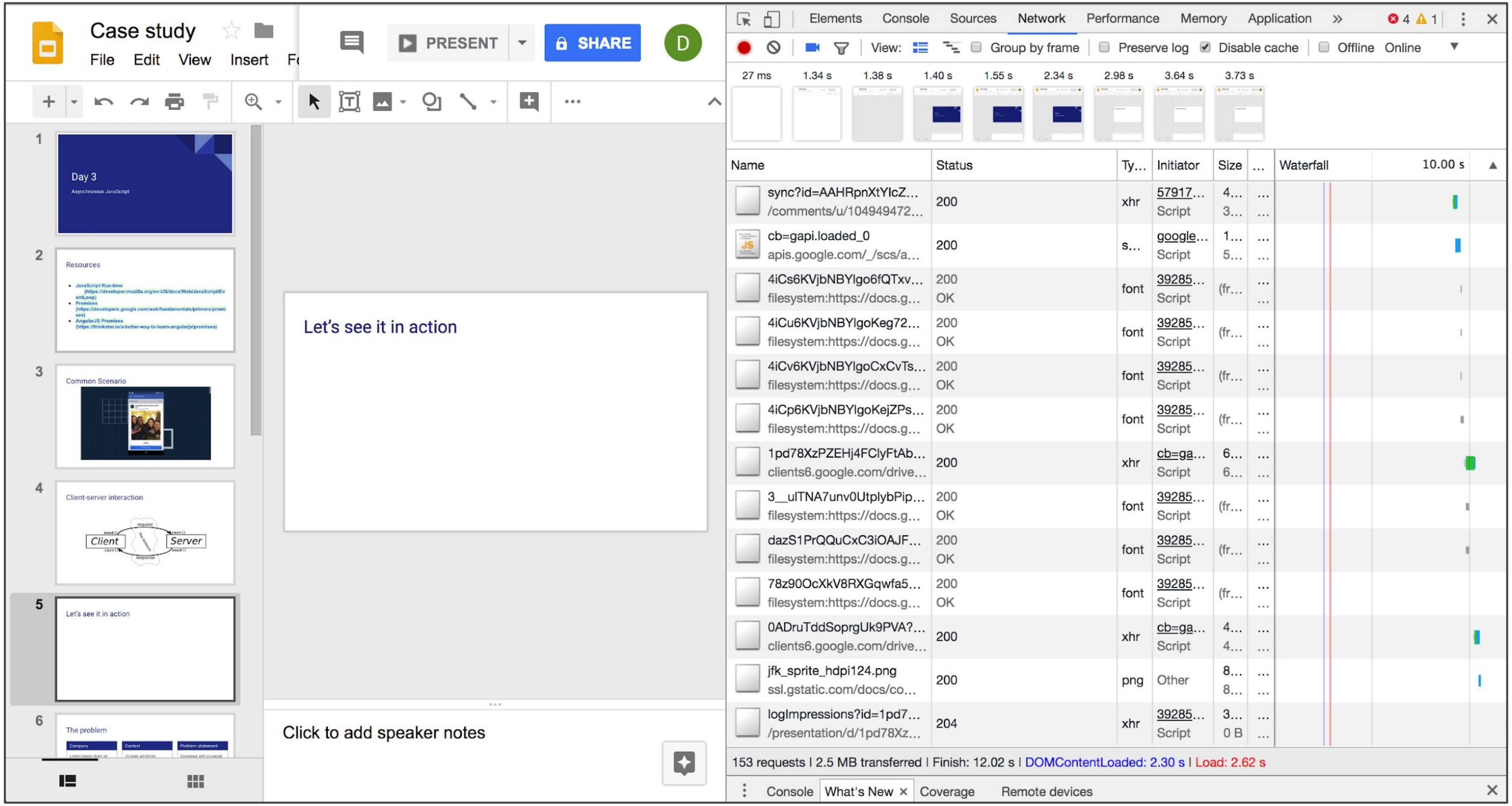
about a minute ago

FACEBOOK

Client-Server Interaction



Let’s See It in Action



1 Day 3
Asynchronous JavaScript

2 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)

3 Common Scenario

4 Client-server interaction

Client

Server

5 Let's see it in action

6 The problem

Company | Context | Problem statement

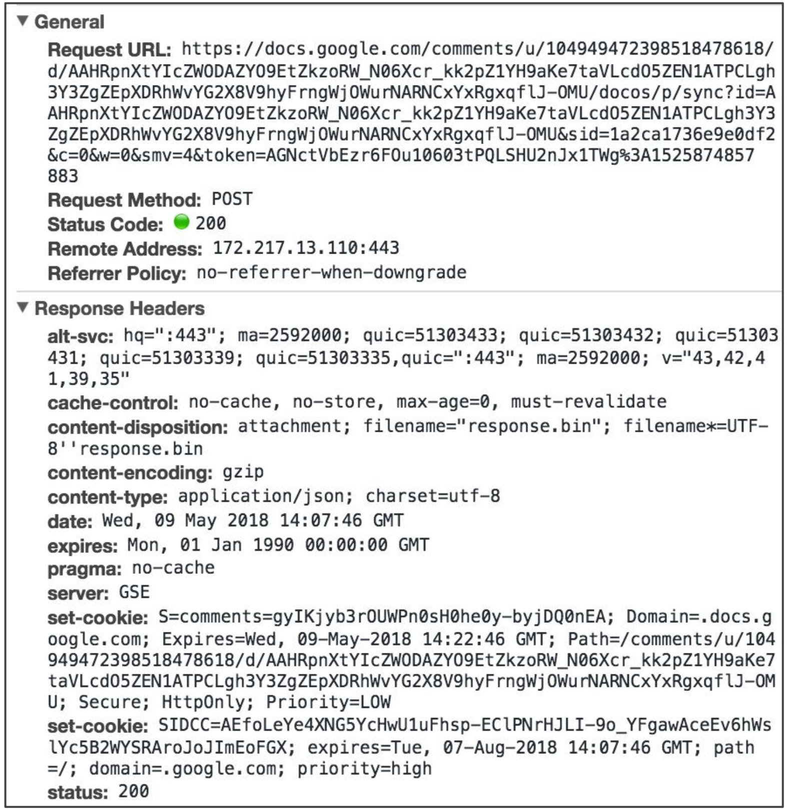
Let’s see it in action

Click to add speaker notes

Name	Status	Ty…	Initiator	Size	…	Waterfall 10.00 s
sync?id=AAHRpnXtYIcZ… /comments/u/104949472…	200	xhr	57917… Script	4… 3…	… …	
cb=gapi.loaded_0 apis.google.com/_/scs/a…	200	s…	google… Script	1… 5…	… …	
4iCs6KVjbNBYlgo6fQTxv… filesystem:https://docs.g…	200 OK	font	39285… Script	(fr…	… …	
4iCu6KVjbNBYlgoKeg72… filesystem:https://docs.g…	200 OK	font	39285… Script	(fr…	… …	
4iCv6KVjbNBYlgoCxCvTs… filesystem:https://docs.g…	200 OK	font	39285… Script	(fr…	… …	
4iCp6KVjbNBYlgoKejZPs… filesystem:https://docs.g…	200 OK	font	39285… Script	(fr…	… …	
1pd78XzPZEHj4FClyFtAb… clients6.google.com/drive…	200	xhr	cb=ga… Script	6… 6…	… …	
3__uITNA7unv0UtplybPip… filesystem:https://docs.g…	200 OK	font	39285… Script	(fr…	… …	
dazS1PrQQuCxC3iOAJF… filesystem:https://docs.g…	200 OK	font	39285… Script	(fr…	… …	
78z90OcXkV8RXGqwfa5… filesystem:https://docs.g…	200 OK	font	39285… Script	(fr…	… …	
0ADruTddSoprgUk9PVA?… clients6.google.com/drive…	200	xhr	cb=ga… Script	4… 4…	… …	
jfk_sprite_hdpi124.png ssl.gstatic.com/docs/co…	200	png	Other	8… 8…	… …	
logImpressions?id=1pd7… /presentation/d/1pd78Xz…	204	xhr	39285… Script	3… 0 B	… …	

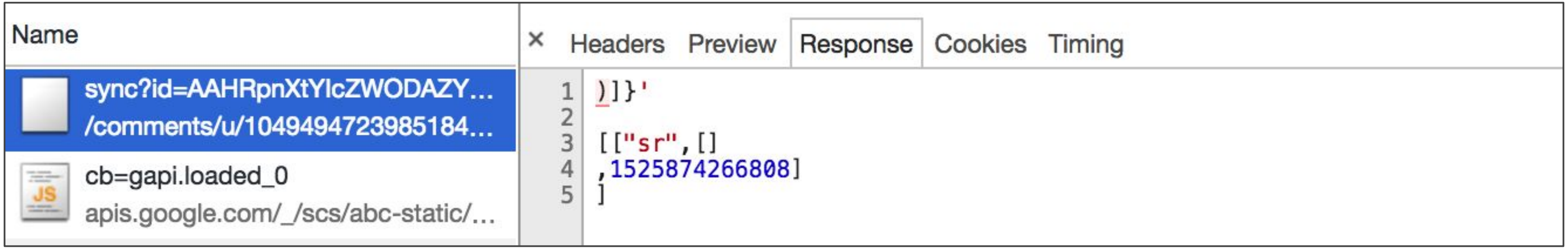
153 requests | 2.5 MB transferred | Finish: 12.02 s | DOMContentLoaded: 2.30 s | Load: 2.62 s

Let’s See It in Action



```
▼ General
    Request URL: https://docs.google.com/comments/u/104949472398518478618/
    d/AAHRpnXtYIcZWODAZYO9EtZkzoRW_N06Xcr_kk2pZ1YH9aKe7taVLcdO5ZEN1ATPCLgh
    3Y3ZgZEpXDRhWvYG2X8V9hyFrngWjOWurNARNCxYxRgxqflJ-OMU/docos/p/sync?id=A
    AHRpnXtYIcZWODAZYO9EtZkzoRW_N06Xcr_kk2pZ1YH9aKe7taVLcdO5ZEN1ATPCLgh3Y3
    ZgZEpXDRhWvYG2X8V9hyFrngWjOWurNARNCxYxRgxqflJ-OMU&sid=1a2ca1736e9e0df2
    &c=0&w=0&smv=4&token=AGNctVbEzr6FOu10603tPQLSHU2nJx1TWg%3A1525874857
    883
    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=51303
    431; quic=51303339; quic=51303335,quic=":443"; ma=2592000; v="43,42,4
    1,39,35"
    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=gyIKjyb3rOUWPn0sH0he0y-byjDQ0nEA; Domain=.docs.g
    oogle.com; Expires=Wed, 09-May-2018 14:22:46 GMT; Path=/comments/u/104
    949472398518478618/d/AAHRpnXtYIcZWODAZYO9EtZkzoRW_N06Xcr_kk2pZ1YH9aKe7
    taVLcdO5ZEN1ATPCLgh3Y3ZgZEpXDRhWvYG2X8V9hyFrngWjOWurNARNCxYxRgxqflJ-OM
    U; Secure; HttpOnly; Priority=LOW
    set-cookie: SIDCC=AEfoLeYe4XNG5YcHwU1uFhsp-EClPNrHJLI-9o_YFgawAceEv6hWs
    lYc5B2WYSRAroJoJImEoFGX; expires=Tue, 07-Aug-2018 14:07:46 GMT; path
    =/; domain=.google.com; priority=high
    status: 200
```

Let’s See It in Action



sync?id=AAHRpnXtYIcZWODAZY...
/comments/u/1049494723985184...

cb=gapi.loaded_0
apis.google.com/_/scs/abc-static/...

```
)]}'

[["sr",[]
,1525874266808]
]
```

Try it yourself

- Tutorial to Inspect Network Activity In Chrome DevTools

Requirements

- A browser is constantly loading data dynamically.

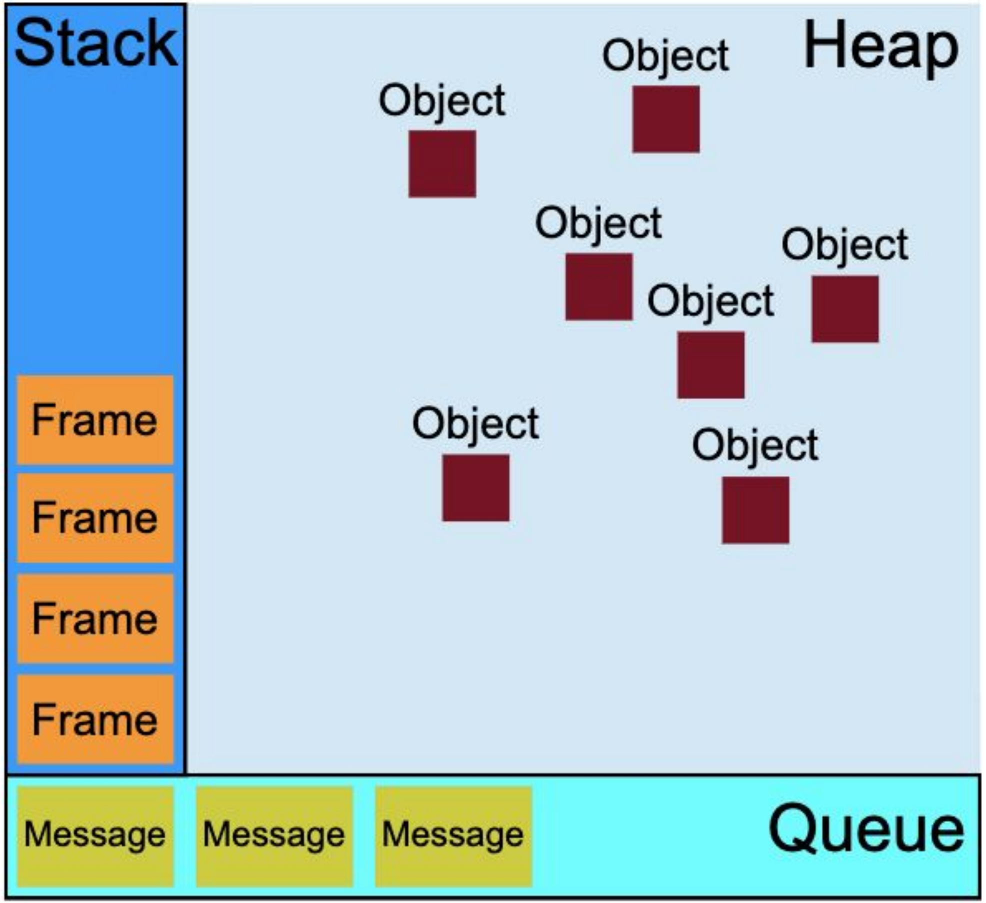
- We would like to have a “non-blocking” UI which always offers the user interaction, even as data is getting prepared in the background.

Solution

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

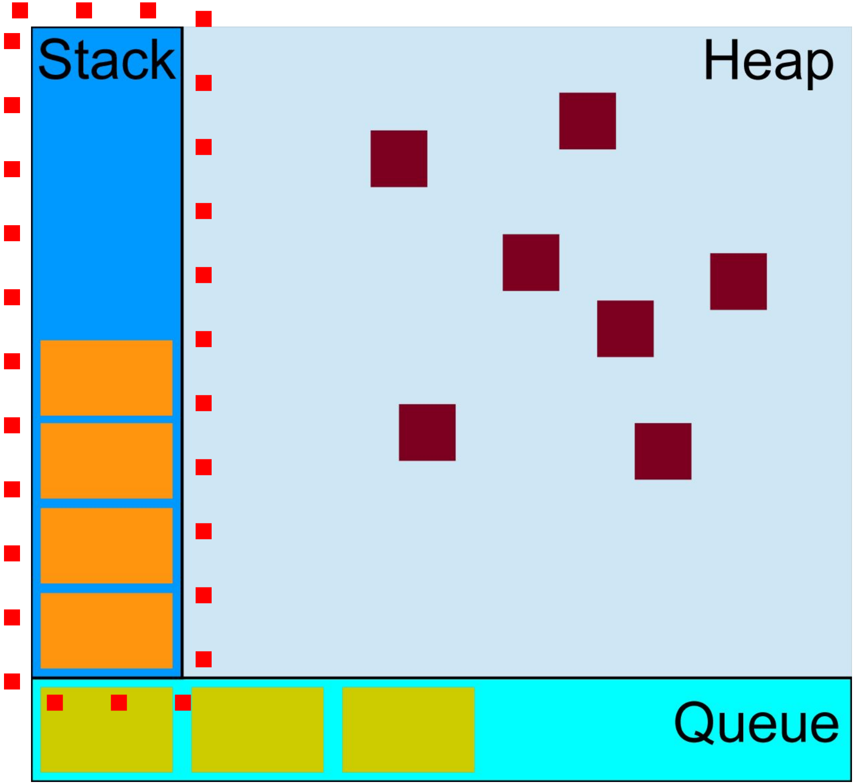
JavaScript Run-Time

The Run-Time

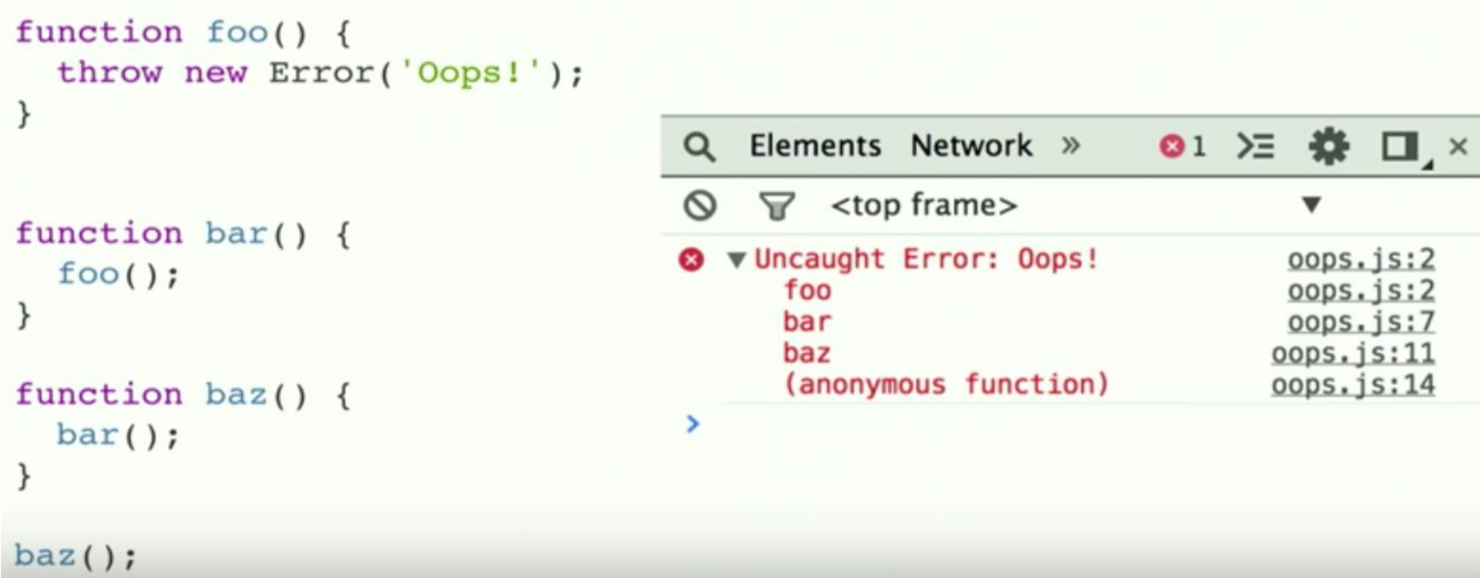

Stack
Heap
Object
Object
Object
Object
Object
Object
Object
Frame
Frame
Frame
Frame
Message
Message
Message
Queue

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.**

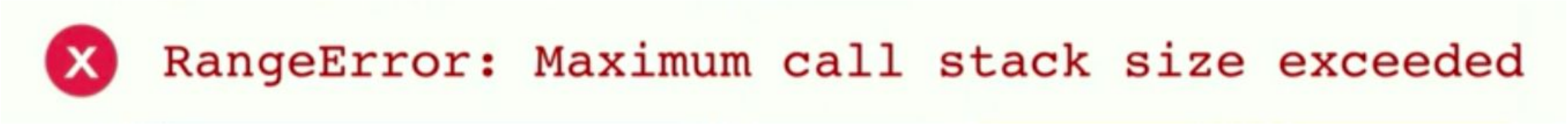


The Stack



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


function bar() {
  foo();
}

function baz() {
  bar();
}


baz();
```

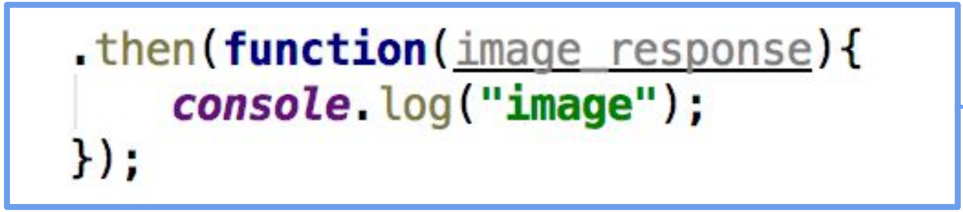
```
Uncaught Error: Oops!            oops.js:2
  foo                            oops.js:2
  bar                            oops.js:7
  baz                            oops.js:11
  (anonymous function)           oops.js:14
>
```



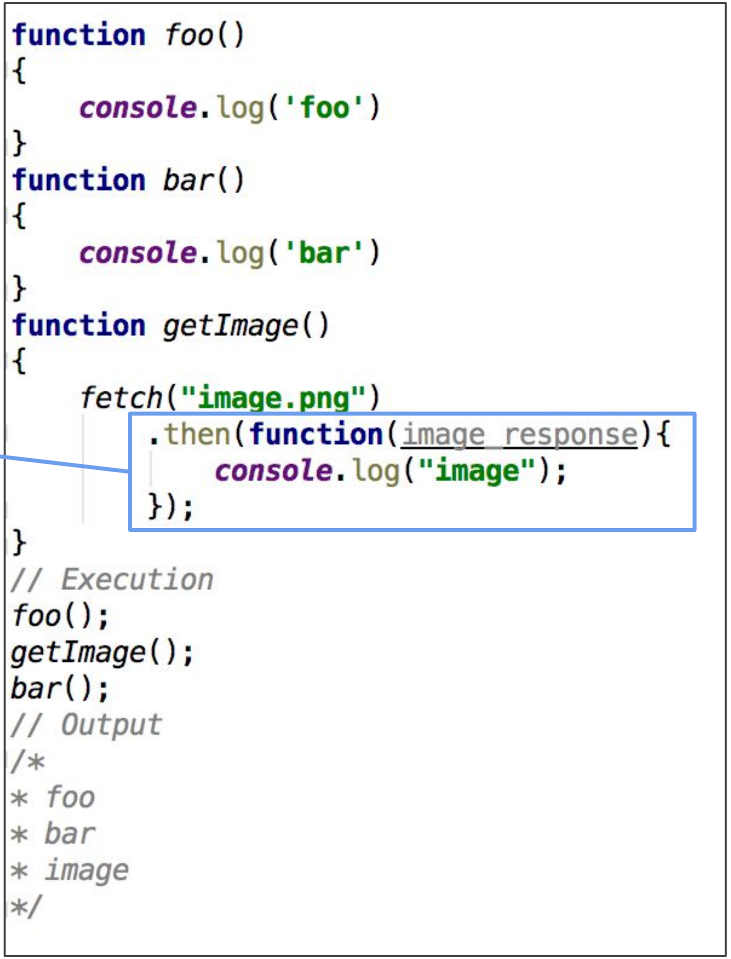
```
RangeError: Maximum call stack size exceeded
```

Asynchronous Code

Callback



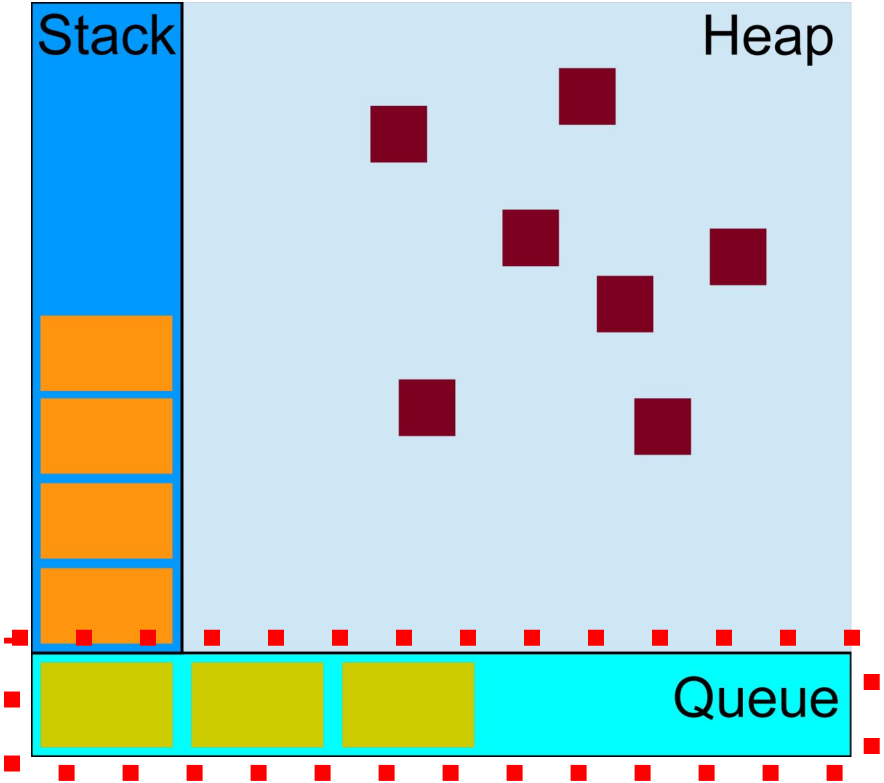
```
.then(function(image_response){
    console.log("image");
});
```



```
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
*/
```

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

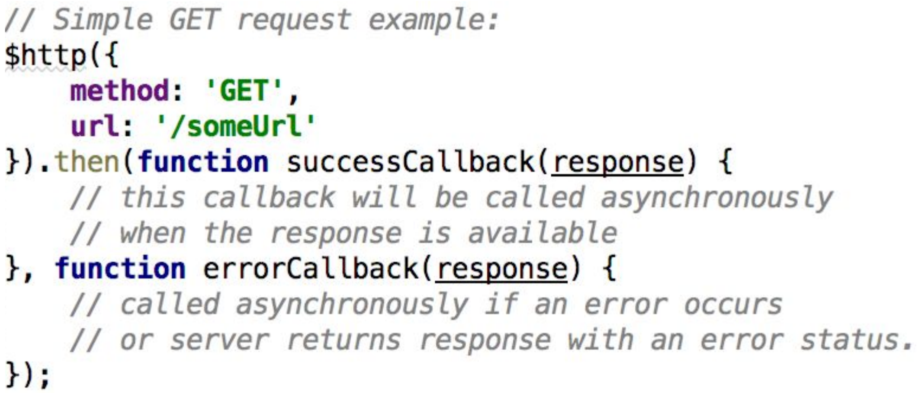
- We have many asynchronous calls made continuously or at the same time. How do we handle this in JavaScript?

Solution

- Many ways to do this in JavaScript
  - Events and callbacks
  - **Promises**
  - await/sync
- **AngularJS** uses promises, so we will focus on this one.
- You can transform all of these representations into one another. The goal is to achieve **expressivity** and **clarity**!
- Reference: 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." (ref)
- A promise can have three states:
    - Pending
    - Resolved
    - Failed



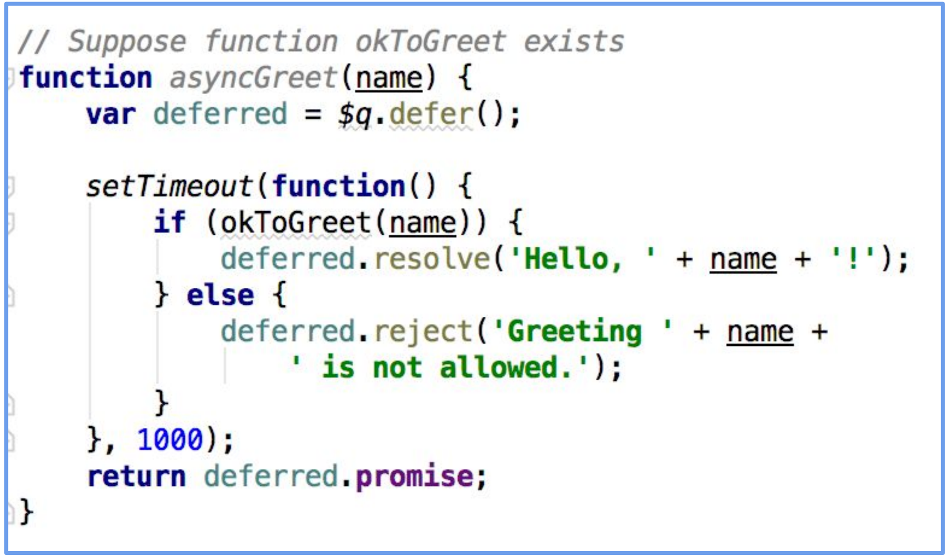
```
// Simple GET request example:
$http({
    method: 'GET',
    url: '/someUrl'
}).then(function successCallback(response) {
    // this callback will be called asynchronously
    // when the response is available
}, function errorCallback(response) {
    // called asynchronously if an error occurs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

How Do We ***Promisify***?

Promisifying: Converting async code into a promise.

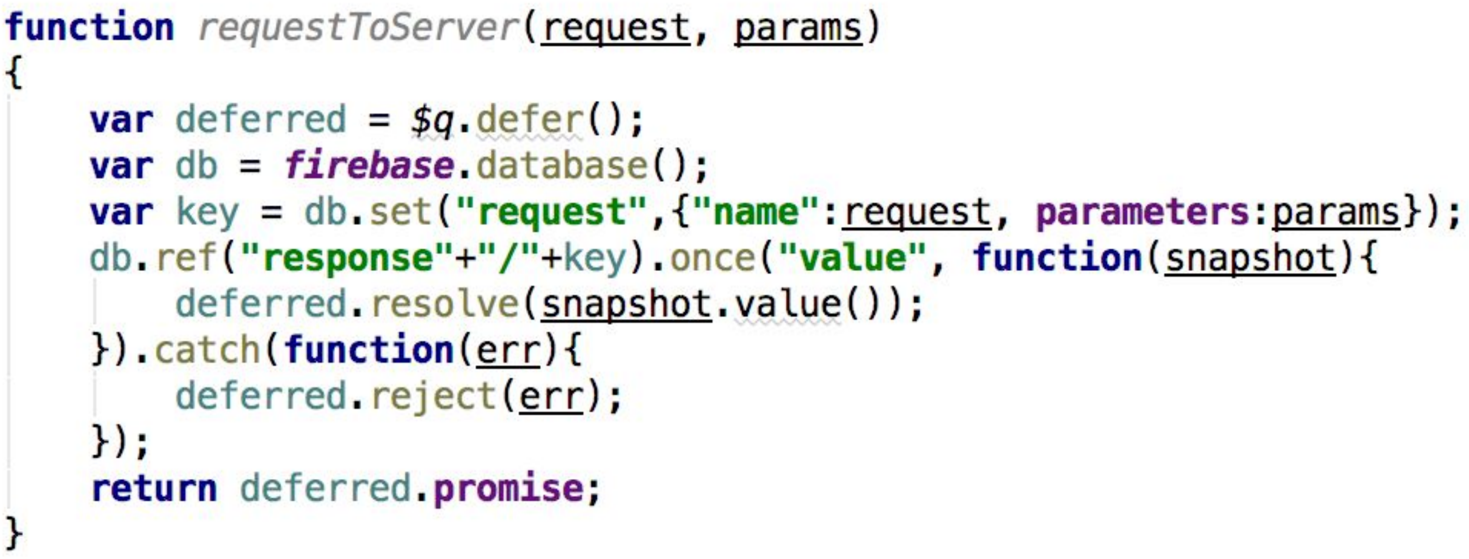
Procedure:

- Wrap a promise around the async code.
- In AngularJS we use the **$q** dependency. In, Node.js, we use **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;
}
```

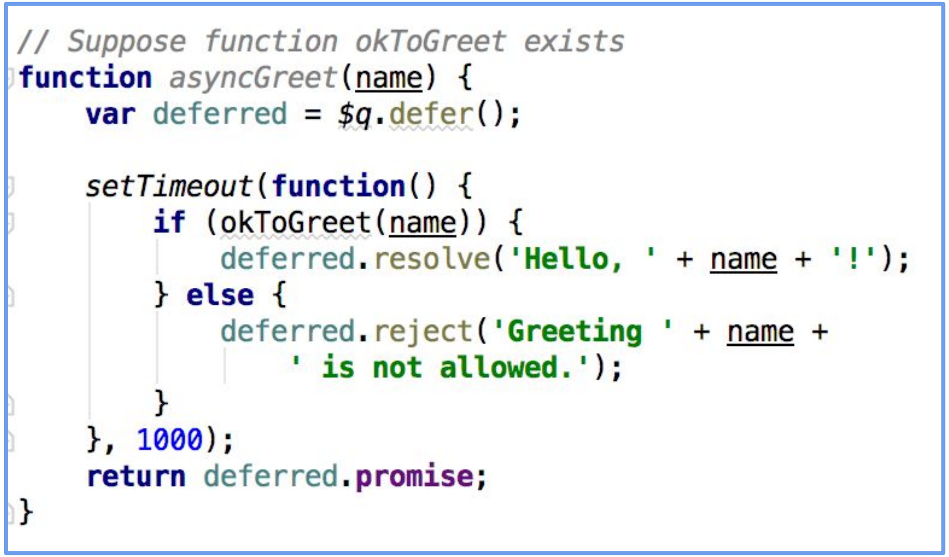
Opal Promise Creation Example



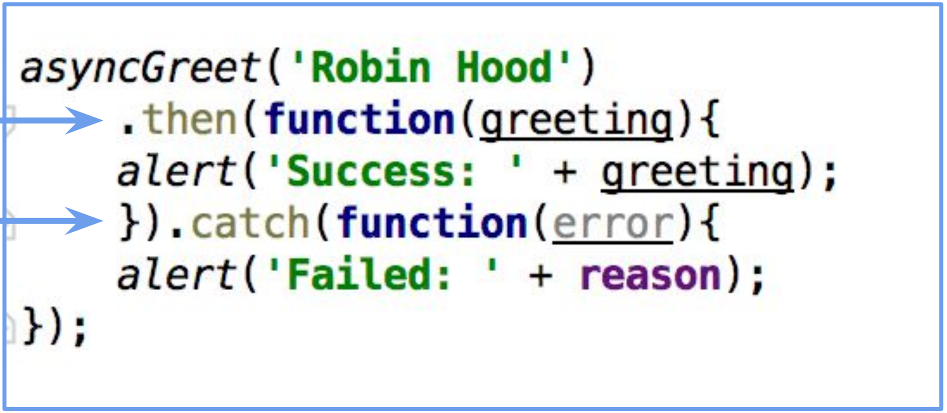
```
function requestToServer(request, params)
{
    var deferred = $q.defer();
    var db = firebase.database();
    var key = db.set("request",{"name":request, parameters:params});
    db.ref("response"+"/"+key).once("value", function(snapshot){
        deferred.resolve(snapshot.value());
    }).catch(function(err){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



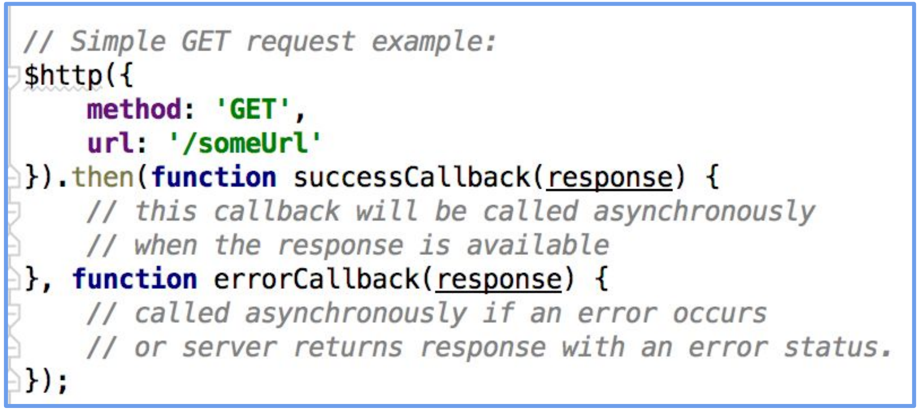
```
// Suppose function okToGreet exists
function asyncGreet(name) {
    var deferred = $q.defer();

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



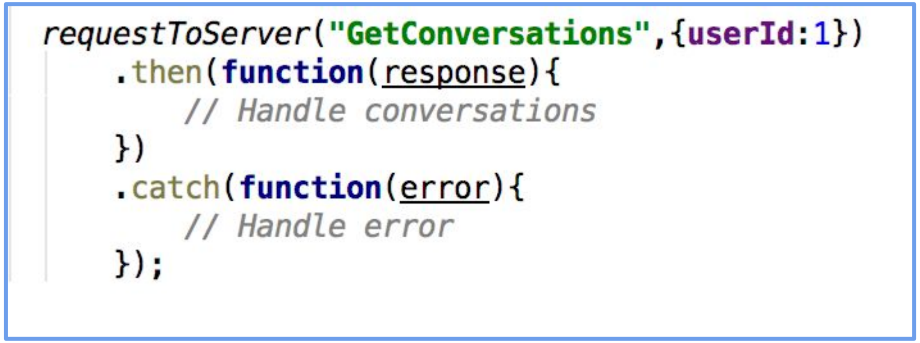
```
asyncGreet('Robin Hood')
    .then(function(greeting){
    alert('Success: ' + greeting);
    }).catch(function(error){
    alert('Failed: ' + reason);
});
```

Common Opal Promises

- $http.get
- Firebase
- All calls to the 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

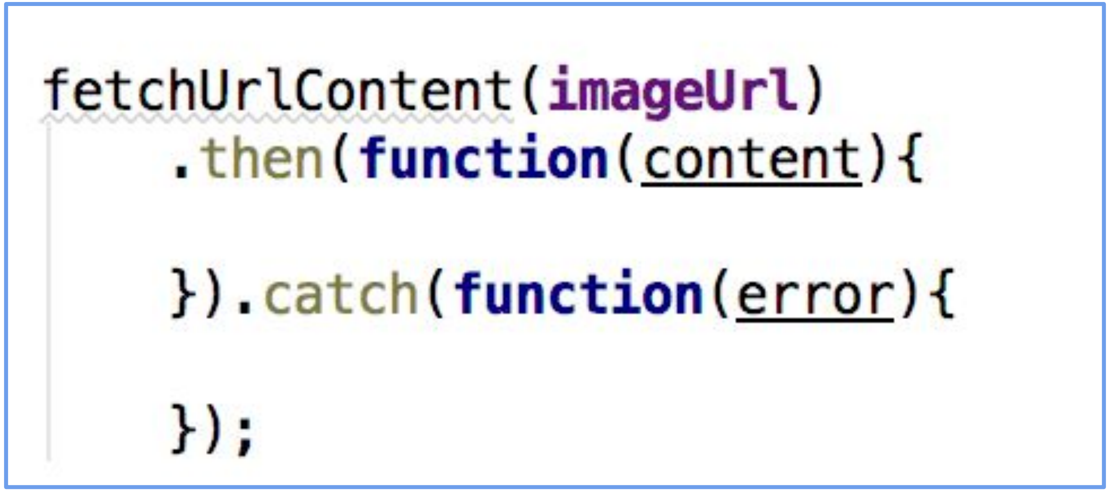
- **Scenario 1:** One simple async request (shown previously).
- **Scenario 2:** Two or more simple requests that **depend** on one another.
- **Scenario 3:** Two or more simple requests that **do not depend** on one another.
- Every other scenario is a combination of these three.

Scenario 1

- **Description:** A simple async request.

- **Procedure:**
    1. Promisify the request (if not promisified).
    2. Use **then/catch.**

Scenario 1 - Example



```
fetchUrlContent(imageUrl)
    .then(function(content){

    }).catch(function(error){

    });
```

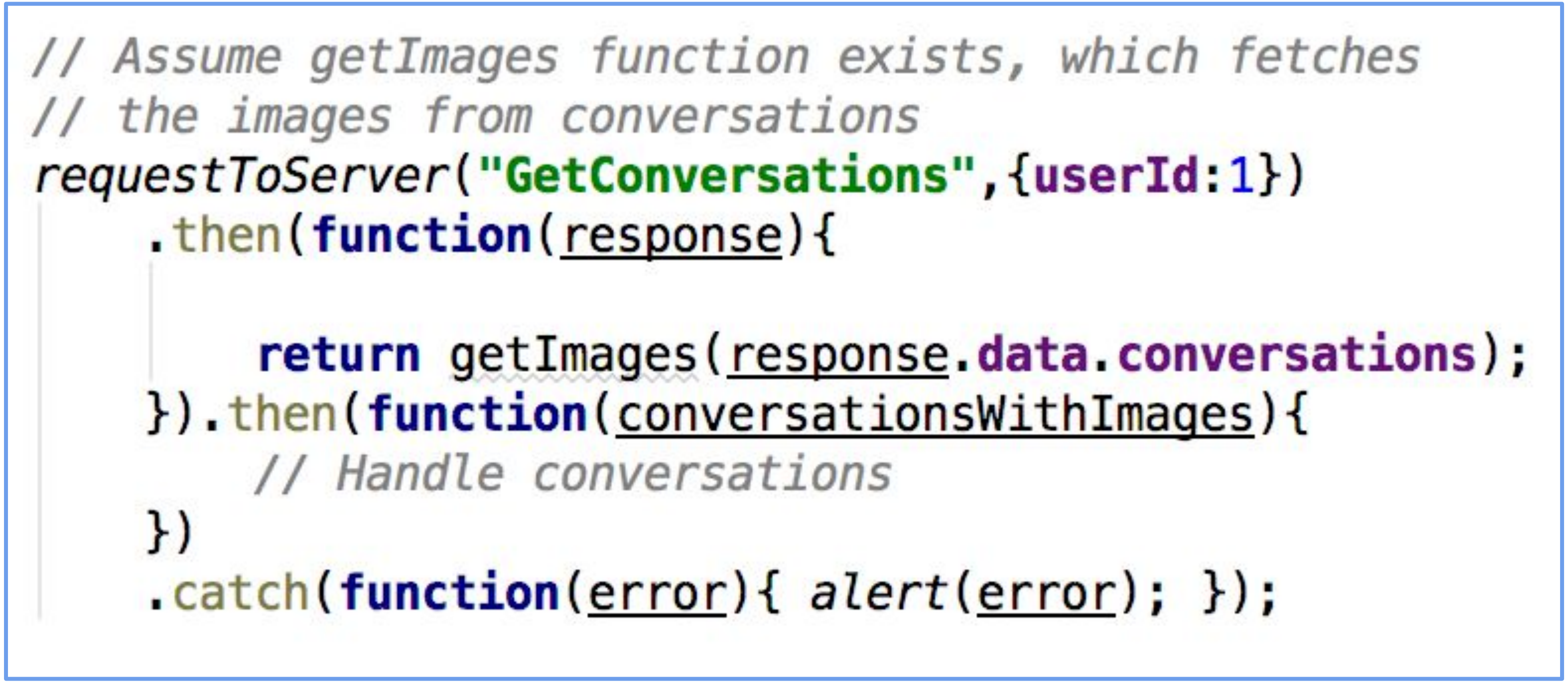
Scenario 2

- **Description**: Two or more simple async requests that depend on one another.

- **Procedure:**
  1. Promisify the requests (if not promisified).
  2. Chain them one after the other, using **return** to launch the next promise in the chain.

Read more on promise chaining here.

Scenario 2 - Example

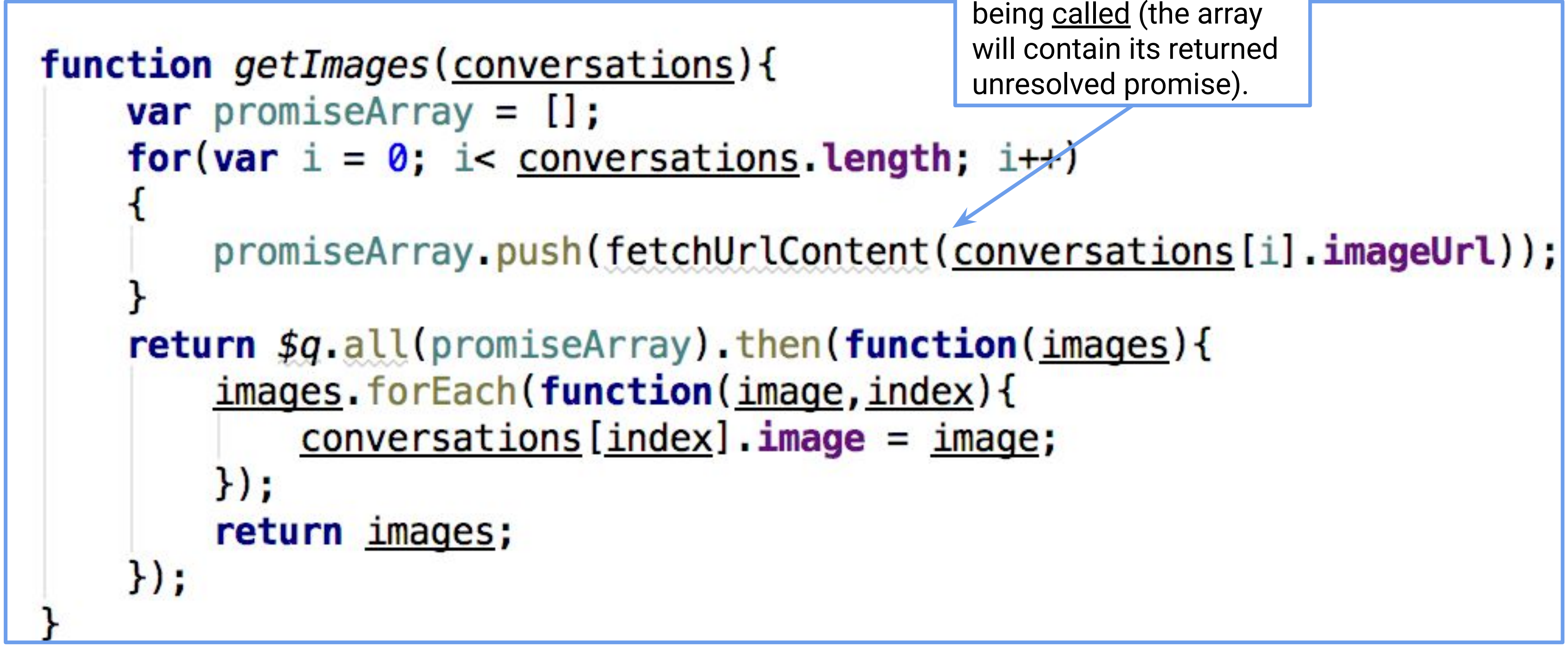


```
// Assume getImages function exists, which fetches
// the images from conversations
requestToServer("GetConversations",{userId:1})
    .then(function(response){

        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 don't depend on one another (order of response arrival doesn't matter).
- **Procedure**
  1. Promisify the requests (if not promisified).
  2. Launch all requests, saving their responses (unresolved promises) in an array.
  3. Use **$q.all()** on the array.
- **Common case:**
  - Fetching images for a list of conversations (these don't not depend on each other but must all return before you can use the conversations.

Scenario 3 - Example



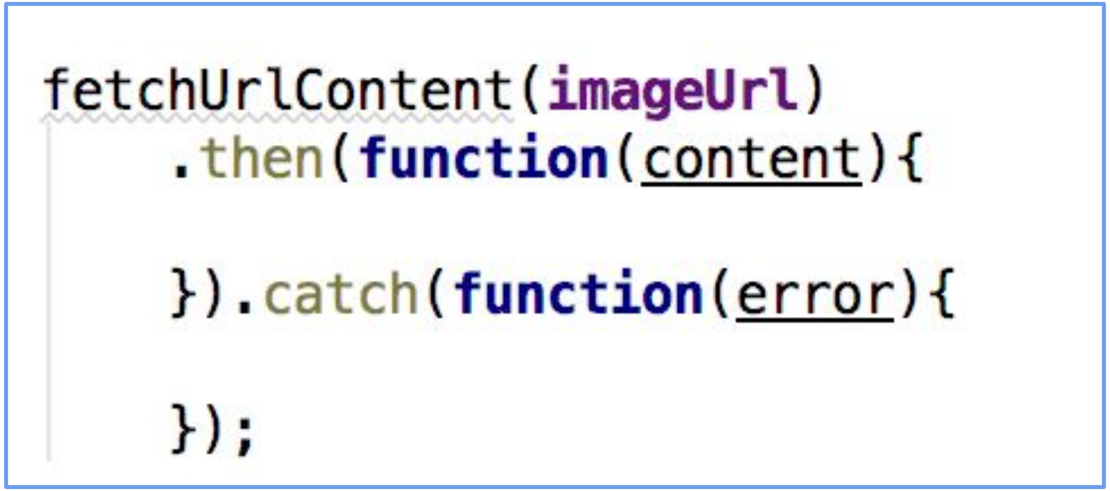
```
function getImages(conversations){
    var promiseArray = [];
    for(var i = 0; i< conversations.length; i++)
    {
        promiseArray.push(fetchUrlContent(conversations[i].imageUrl));
    }
    return $q.all(promiseArray).then(function(images){
        images.forEach(function(image,index){
 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

Notice that the promisified function is being called (the array will contain its returned unresolved promise).

Last Comments

- You will encounter this concept repeatedly in web development!
- Read the references at the beginning of the presentation.
- Do the assignment containing exercises on async js.
- If you master this, you are well on your way to becoming an expert in JavaScript :)

End of Asynchronous JavaScript



```
fetchUrlContent(imageUrl)
    .then(function(content){

    }).catch(function(error){

    });
```