



PROMISES

ASYNCHRONOUS PROGRAMMING

Prof. Andrew Sheehan

Metropolitan College
Boston University

WHAT IS A PROMISE?

A Promise is a proxy for a value not necessarily known when the promise is created.

A promise represents WORK that needs to be done at some point.

Promises - OTHER

A promise represents the eventual result of an operation.

Current aligned

Usage relative

Date relative

Chrome	Edge [*]	Safari	Firefox
4-31			2-26
32		3.1-7	27-28
33-104	12-104	7.1-15.5	29-103
105	105	15.6	104
106-108		16.0-TP	105-106

USEABLE
EVERYWHERE

You can use it
everywhere
but not with
Internet
Explorer.



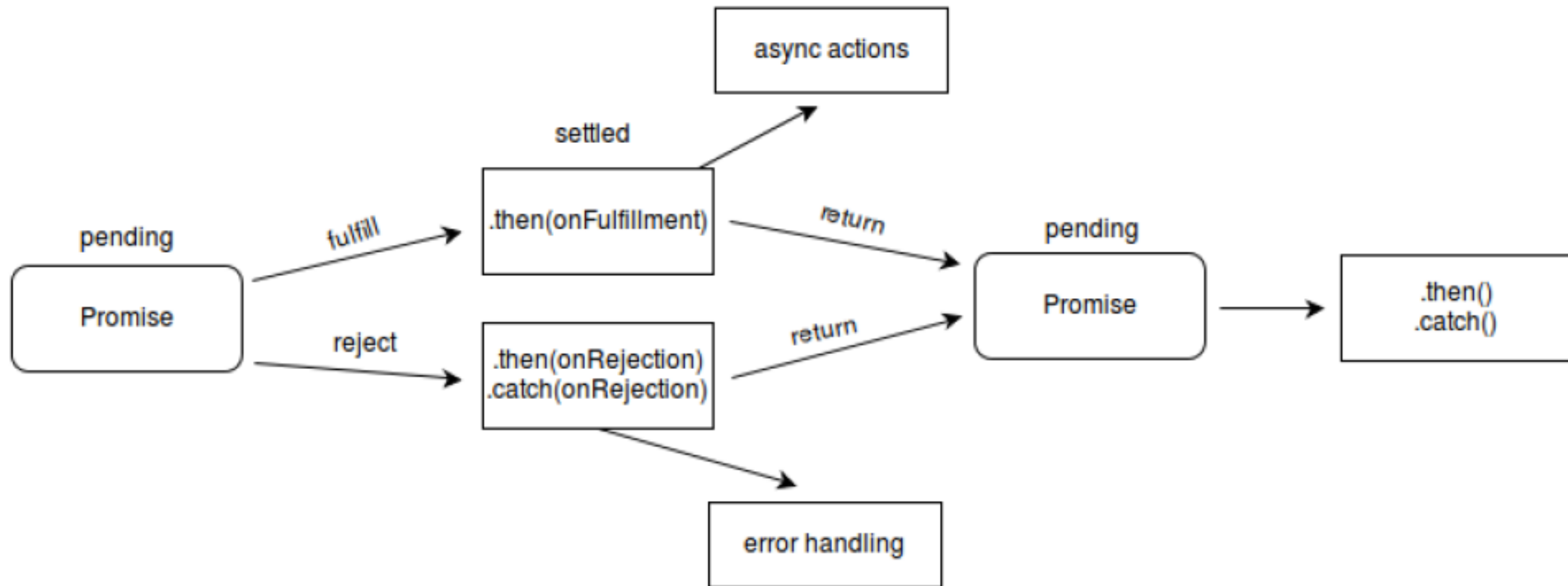
Synchronous

One expression at a time.

Asynchronous

The next expression runs while the previous finishes up.

DIFFERENCES



PROMISE STATES |

STANDARD EXAMPLE

```
var promise = new Promise(function(resolve, reject) {  
  // do a thing, possibly async, then...  
  
  if (/* everything turned out fine */) {  
    resolve("Stuff worked!");  
  }  
  else {  
    reject(Error("It broke"));  
  }  
});
```

A `Promise` is in one of these states:

- *pending*: initial state, neither fulfilled nor rejected.
- *fulfilled*: meaning that the operation was completed successfully.
- *rejected*: meaning that the operation failed.

THEN() - CAN BE USED MULTIPLE TIMES

```
> const myPromise = new Promise( (resolve, reject) => {  
  resolve('Hello world!');  
});
```

```
myPromise.then( value => console.log(value));  
myPromise.then( value => console.log(value));  
myPromise.then( value => console.log(value));
```

```
Hello world!
```

```
Hello world!
```

```
Hello world!
```

```
◀ ▶ Promise {<resolved>: undefined}
```



```
const myPromise = new Promise((resolve, reject) => {  
  setTimeout(() => {  
    resolve('foo');  
  }, 300);  
});
```

```
myPromise  
  .then(handleResolvedA)  
  .then(handleResolvedB)  
  .then(handleResolvedC)  
  .catch(handleRejectedAny);
```

CHAINING PROMISES

Every use of
.then() **will**
return a
Promise
that you can
use **again.**

PROMISE.ALL()

```
<script>
  var prom3000 = new Promise(function(resolve, reject) {
    setTimeout(function() { resolve(" 3 seconds out!"); }, 3000);
  });
  var prom6000 = new Promise(function(resolve, reject) {
    setTimeout(function() { resolve(" 6 seconds out!"); }, 6000);
  });
  var prom9000 = new Promise(function(resolve, reject) {
    setTimeout(function() { resolve(" 9 seconds out"); }, 9000);
  });
  $(function() {
    $("button").on("click", function() {
      Promise.all([prom3000, prom6000, prom9000]).then(function(results) {
        console.log("All promises done" + results);
      });
    });
  });
</script>
```

PROMISE.ALL()

The `all()` method takes an iterable of promises as input, and returns a single Promise that resolves to an array of the results of the input promises, in order.

Promise.all(iterable)

Wait for all promises to be resolved, or for any to be rejected.

If the returned promise resolves, it is resolved with an aggregating array of the values from the resolved promises, in the same order as defined in the iterable of multiple promises.

If it rejects, it is rejected with the reason from the first promise in the iterable that was rejected.

.ALL()

RACE() - WAIT UNTIL ANY PROMISE WORKS OR FAILS

Promise.race(iterable)

Wait until any of the promises is fulfilled or rejected.


If the returned promise resolves, it is resolved with the value of the first promise in the iterable that resolved.

If it rejects, it is rejected with the reason from the first promise that was rejected.

CATCHING ERRORS

The **catch()** returns a Promise. Deals w/rejected cases only.

```
1 const promise1 = new Promise((resolve, reject) => {  
2   throw 'Uh-oh!';  
3 });  
4  
5 promise1.catch((error) => {  
6   console.error(error);  
7 });  
8 // expected output: Uh-oh!
```



CATCH() IS JUST A FORM OF THEN()

The `catch()` is just like `then()` without a callback function for success.

