PROMISES

ASYNCHRONOUS PROGRAMMING

Prof. Andrew Sheehan

Boston University
Computer Science Dept/MET

ES6 STANDARD

WHAT IS A PROMISE?

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

When a promise completes, it will invoke either a resolve or a reject function

FUNCTIONS ARE CALLBACKS

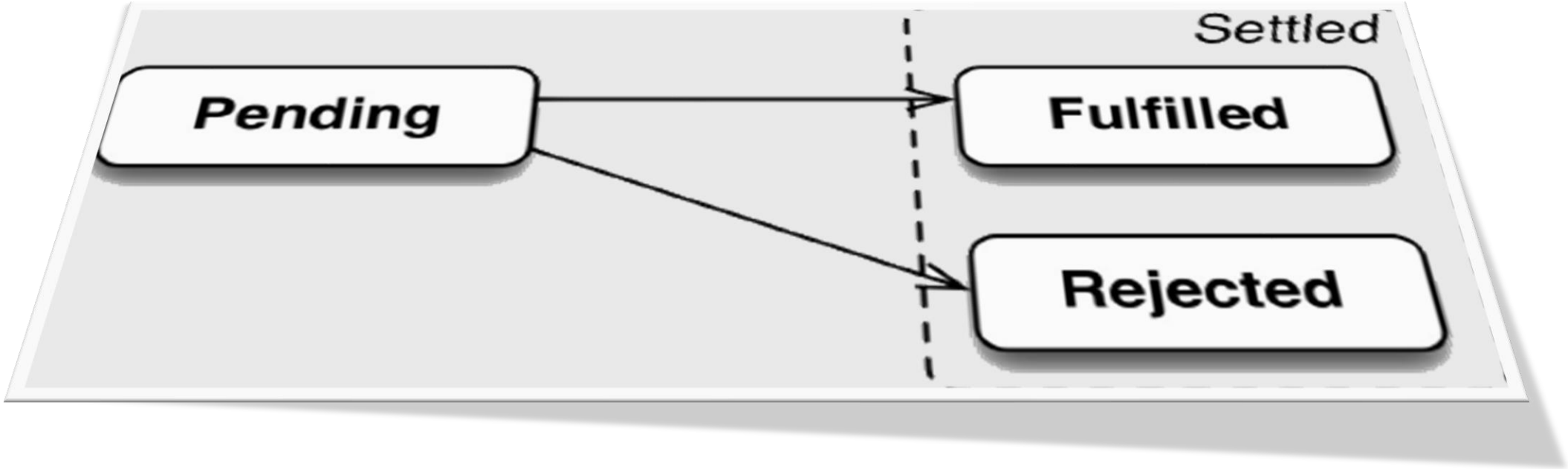
DIFFERENCES

Synchronous

Do one expression at a time.

Asynchronous

Next expression runs while the previous finishes up *at some point*.



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.

PROMISE STATES

```
let p = new Promise(function(resolve, reject) {
    if ( /* all good */ ) {
        resolve(/* return something…*/);
    } else {
        reject(/* with something */);
    }
});
```

CREATING A PROMISE

BASIC USE CASE

```
function example() {
    return new Promise(function (resolve, reject) {
        // do some code - successfully
        resolve(result);  // success!
        // or reject - failed
        reject(error);   // denotes failure
    });
}

example()
    .then(result => { ··· })
    .catch(error => { ··· });
```

Promise.resolve() ensures that the parameter used will be wrapped in a promise.

RESOLVE()

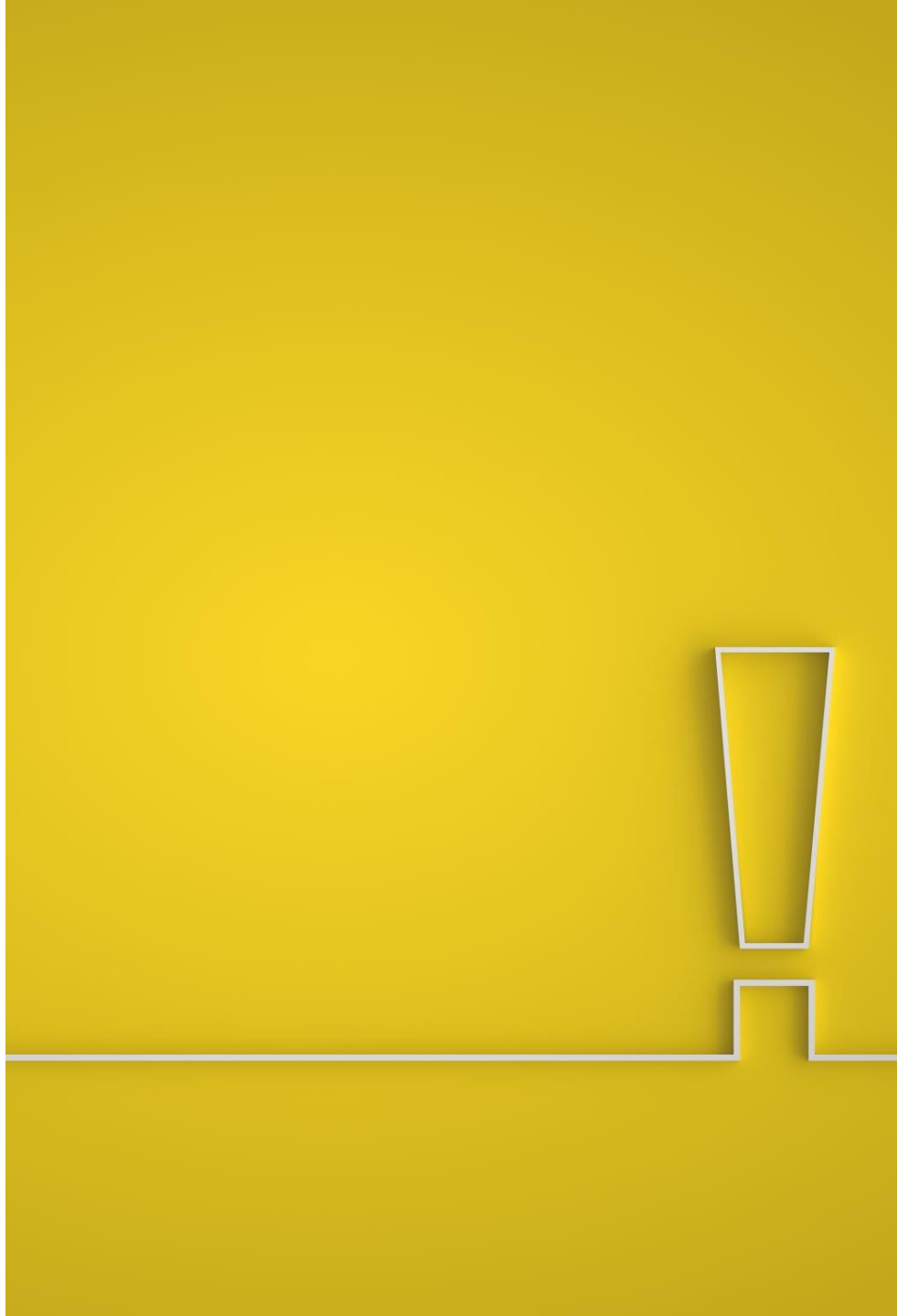
Resolve() Example

```
const today =
  new Promise.resolve(new Date());


today.then(value => {
    console.log(value);
});
```

PROMISE.REJECT()

Ensures that the parameter used will be wrapped in a rejected promise.

```
let badRequest =
      Promise.reject({'error': 'bad request'});

badRequest
   .then( result => {
         console.log(shouldNotBeHere);
}).catch(error => {
         console.error(result.error);
});
```



```
new Promise( (_, reject) => reject(new Error(“Testing…")))
  .then(value => {
     console.log(“Success handler…"); // won’t happen..
}).catch(reason => {
    console.log(`Failure: ${reason}`);
    return “Continue";
}).then(value => {
    console.log(`After rejection: ${value}.`);
});
```

```
// Failure: Testing…
// After rejection: Continue
```

```javascript
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 invoke the Promise** that you can use **again**.

FINALLY()

- Takes no arguments
- Similar to then()
- resolved or rejected: finally will be invoked, *If you use it…*

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(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.

PROMISE .RACE()

```
const promise1 = new Promise((resolve, reject) => {
    setTimeout(resolve, 500, 'one');
});
const promise2 = new Promise((resolve, reject) => {
    setTimeout(resolve, 100, 'two');
});

Promise.race([promise1, promise2]).then((value) => {
    console.log(value);
});
```

// Expected output: "two“ Both resolve, but promise2 is faster

CHOICES

In the end, there will be only 1 result:

Either resolve() or reject().