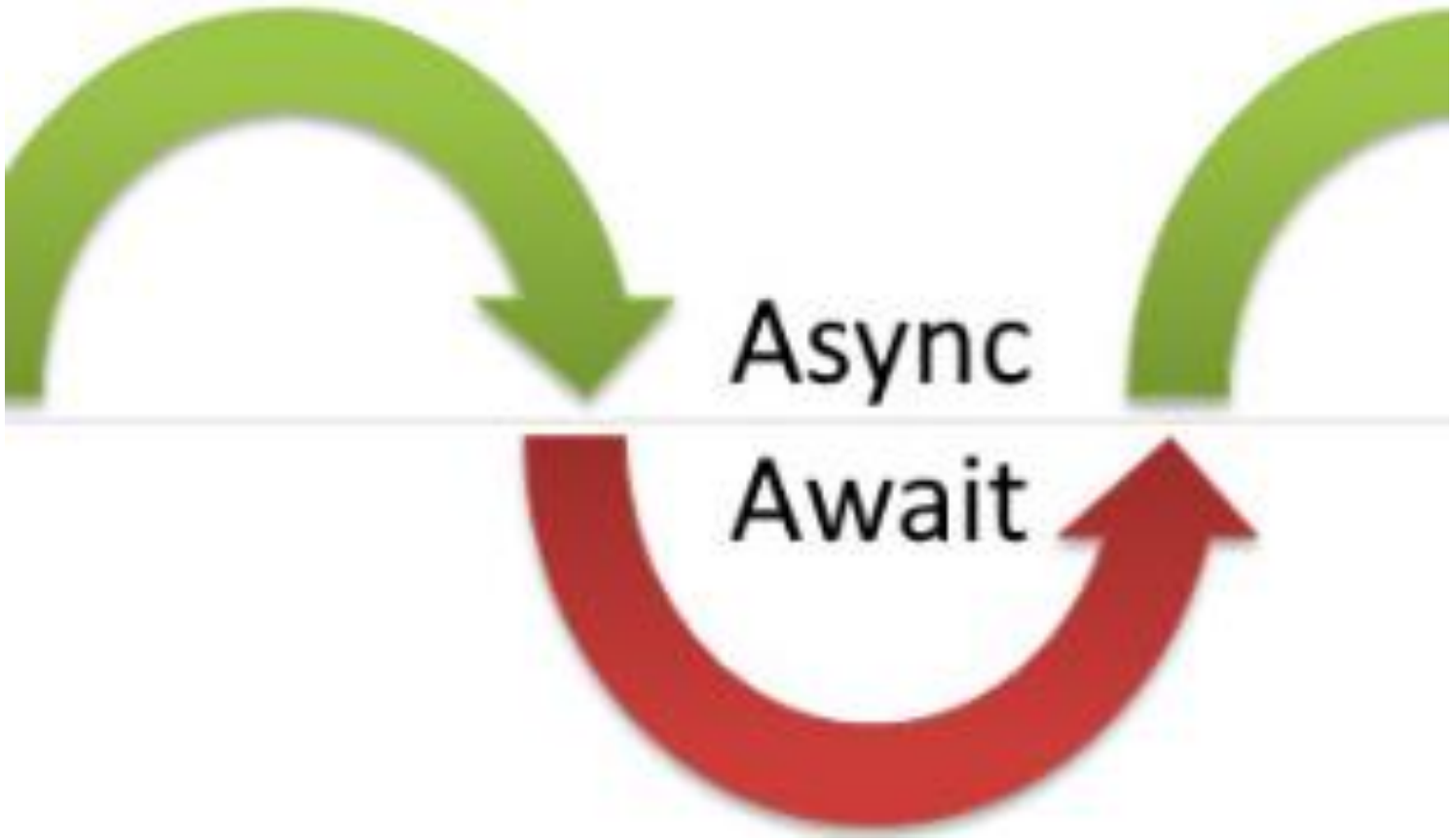
ASYNC & AWAIT

Andrew Sheehan

Boston University
Computer Science Dept.



WHAT ARE THEY?

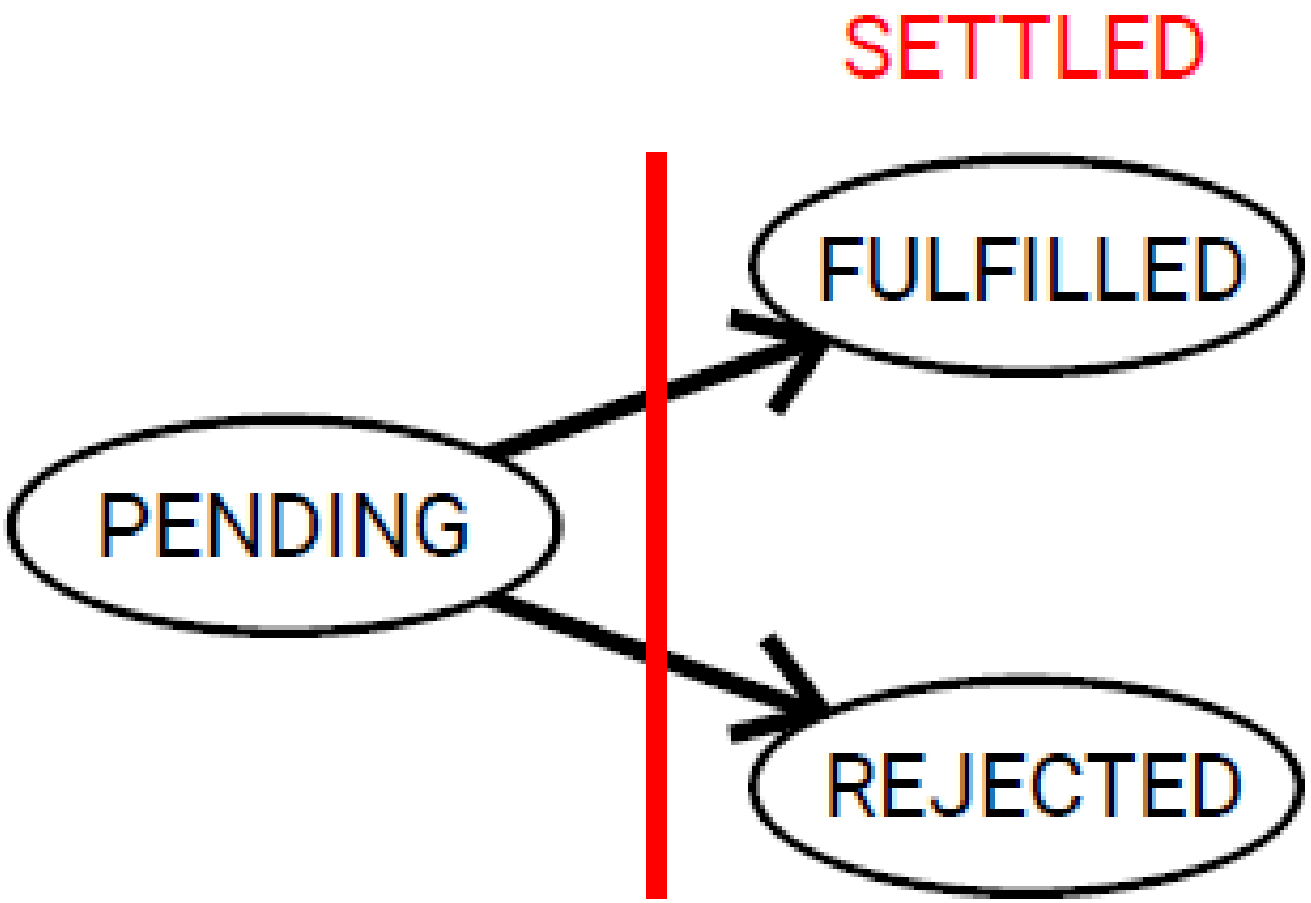
A different syntax to code **Promises.**

Became part of the standard in ECMAScript **2017**

THE CREATOR OF THE ASYNC MODULE

Caolan McMahon

RECAP: PROMISE STATES



- `async` is a keyword for the function declaration
- `await` is used during the promise handling
- `await` must be used within an `async` function, though Chrome now supports "top level" `await`
- `async` functions return a promise, regardless of what the `return` value is within the function
- `async` / `await` and promises are essentially the same under the hood

TL;DR

EXAMPLE

```
async function save(Something) {
  try {
    await Something.save()
  } catch (ex) {
    //error handling
  }
  console.log('success');
}
```

EXAMPLE:

AWAITING YOUR FETCH() CALLS.

```
async function fetchContent() {
  // Instead of using fetch().then, use await
  let content = await fetch('/');
  let text = await content.text();

  // Inside the async function text is the request body
  console.log(text);

  // Resolve this async function with the text
  return text;
}

// Use the async function
var promise = fetchContent().then(...);
```

BEFORE AND *AFTER*


```javascript
fetch('/users.json')
  .then(response => response.json())
  .then(json => {
    console.log(json);
  })
  .catch(e => { console.log('error!'); })

// After: no more callbacks!
async function getJson() {
  try {
    let response = await fetch('/users.json');
    let json = await response.json();
    console.log(json);
  }
  catch(e) {
    console.log('Error!', e);
  }
}
```

```javascript
async function test() {
  // This function will print "Hello, World!" after 1 second.
  await new Promise(resolve => setTimeout(() => resolve(), 1000));
  console.log('Hello, World!');
}

test();
```

EXAMPLE USING A TIMEOUT

```javascript
async function test() {
  // Wait 100ms 10 times. This function also prints after 1 sec
  for (let i = 0; i < 10; ++i) {
    await new Promise(resolve => setTimeout(resolve, 100));
  }
  console.log('Hello, World!');
}

test();
```

YOU CAN USE AWAIT IN IF STATEMENTS, FOR LOOPS, AND TRY/CATCH BLOCKS

AWAIT

Await on the fetch(); await on the json();

```
async function showAvatar() {

  // read our JSON
  let response = await fetch('/article/promise-chaining/user.json');
  let user = await response.json();

  // read github user
  let githubResponse = await fetch(`https://api.github.com/users/${user.name}`);
  let githubUser = await githubResponse.json();

  // show the avatar
  let img = document.createElement('img');
  img.src = githubUser.avatar_url;
  img.className = "promise-avatar-example";
  document.body.append(img);

  // wait 3 seconds
  await new Promise((resolve, reject) => setTimeout(resolve, 3000));

  img.remove();
```

TYPICAL ERROR SITUATION

There is one major restriction for using await: You can only use await within the body of **a function that's marked async**

```
function test() {
  const p = new Promise(resolve => setTimeout(resolve, 1000));
  // SyntaxError: Unexpected identifier
  await p;
}
```

DECLARING

Anonymous Async Function

```
let main = (async function() {
  let value = await fetch('/');
})();
```

Async Function Declaration

```
async function main() {
  let value = await fetch('/');
};
```

Async Function Assignment

```
let main = async function() {
  let value = await fetch('/');
};
```

Async Function as Argument

```
document.body.addEventListener('click', async function() {
  let value = await fetch('/');
});
```

PASSING AS ARGUMENTS

OBJECTS AND METHODS

```javascript
// Object property
let obj = {
  async method() {
    let value = await fetch('/');
  }
};

// Class methods
class MyClass {
  async myMethod() {
    let value = await fetch('/');
  }
}
```

PARALLELISM

```
// Will take 1000ms total!
async function series() {
  await wait(500);
  await wait(500);
  return "done!";
}
```

```
// Would take only 500ms total!
async function parallel() {
  const wait1 = wait(500);
  const wait2 = wait(500);
  await wait1;
  await wait2;
  return "done!";
}
```

- Trigger both wait calls and then use await.
- Allows the async functions to happen concurrently

PROMISE.ALL()
EQUIVALENT

```
let [foo, bar] = await Promise.all([getFoo(), getBar()]);
```

ERROR HANDLING

If there is an error, you can use a standard:

try { … } catch() { … }

```
async function test() {
  try {
    const p = Promise.reject(new Error('Oops!'));
    // The below `await` throws
    await p;
  } catch (error) {
    console.log(error.message); // "Oops!"
  }
}
```

ERROR HANDLING

This code:

```
1 async function f() {
2   await Promise.reject(new Error("Whoops!"));
3 }
```

...Is the same as this:

```
1 async function f() {
2   throw new Error("Whoops!");
3 }
```

We can catch that error using `try..catch`,

```
async function f() {

  try {
    let response = await fetch('http://no-such-url');
  } catch(err) {
    alert(err); // TypeError: failed to fetch
  }
}

f();
```