Making JavaScript beautiful again with async/await

Kevin Bravestone

Kevin Bravestone

NERDERY

Helpful to Know

- JavaScript (ES6/ES2015)
- Basics of promises

The asynchronous world

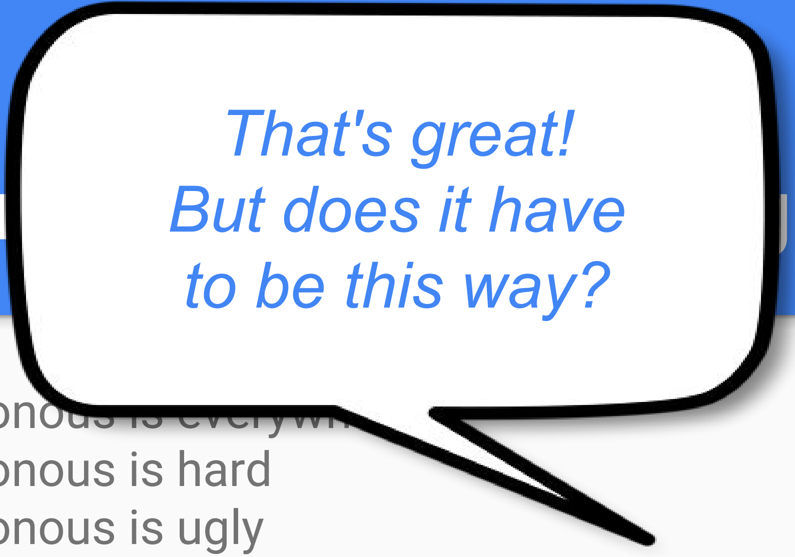
Kick off, but don't complete right away

- setTimeout()
- AJAX calls (fetch)
- Animations
- File access
- ...

Asynchronous programming 101

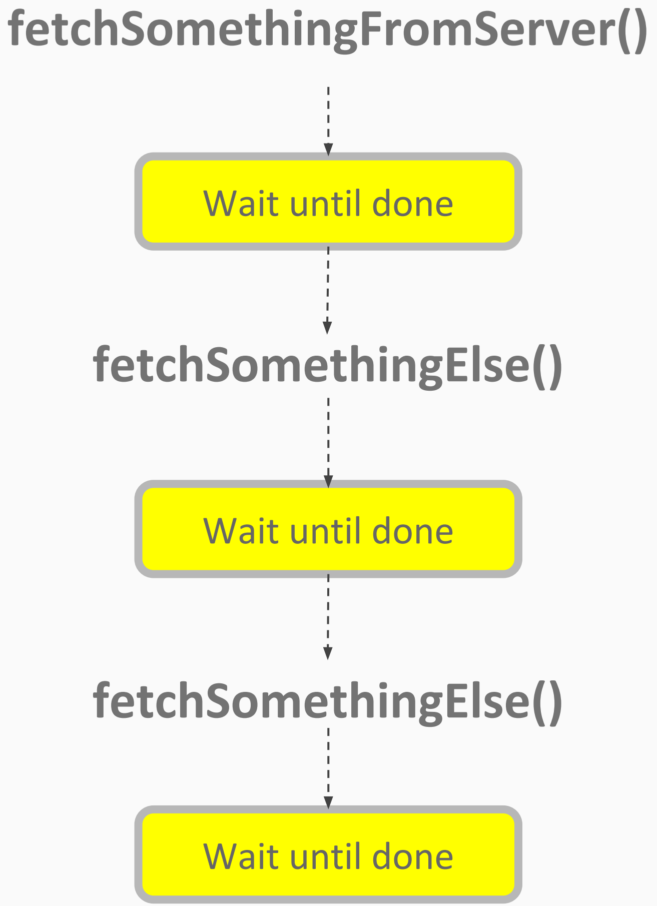
- Asynchronous is everywhere
- Asynchronous is hard
- Asynchronous is ugly

Asynchr 101



- Asynchronous is everyw
- Asynchronous is hard
- Asynchronous is ugly

Approaches!


fetchSomethingFromServer()
Wait until done
fetchSomethingElse()
Wait until done
fetchSomethingElse()
Wait until done

Approach #1: Callbacks

```
fetchSomethingFromServer(() => {
    fetchSomethingElse(() => {
        fetchSomethingElse(() => {
            fetchSomethingElse(() => {
                fetchSomethingElse(() => {
                });
            });
        });
    });
});
```

Approach #1: Callbacks

```
fetchSomethingFromServer(() => {
    fetchSomethingElse(() => {
        fetchSomethingElse(() => {
            fetchSomethingElse(() => {
                fetchSomethingElse(() => {
                });
            });
        });
    });
});
```

Approach #2: Promise chains

```
fetchSomethingFromServer()
    .then(fetchSomethingElse)
    .then(fetchSomethingElse)
    .then(fetchSomethingElse)
    .then(fetchSomethingElse);
```

Approach #2: Promise chains

```
fetchSomethingFromServer()
    .then(fetchSomethingElse)
    .then(result => {
        fetchSomethingElse(result)
            .catch(handleError)
            .then(fetchSomethingElse);
    });
```

But can still get ugly

Approach #2: Promise chains

```
fetchSomethingFromServer()
    .then(fetchSomethingElse)
    .then(result => {
        fetchSomethingElse(result)
            .catch(handleError)
            .then(fetchSomethingElse);
    });
```

Approach #3: Async/await

```
async function() {
    await fetchSomethingFromServer();
    await fetchSomethingElse();
    await fetchSomethingElse();
    await fetchSomethingElse();
    await fetchSomethingElse();
}
```

Approach #3: Async/await

```
async function() {
    await fetchSomethingFromServer();
    await fetchSomethingElse();
    await fetchSomethingElse();
    await fetchSomethingElse();
    await fetchSomethingElse();
}
```

Better!

Is asynchronous, looks synchronous

Syntax

```
async function() {
    const result = await <promise>;
}
```

Syntax

```
async function() {
    const result = await <promise>;
}
```

Waits for promise to resolve

Syntax

```
async function() {
    const result = await <promise>;
}
```

Result of promise
assigned to this variable

What it is

- Simpler & more ergonomic way to use Promises
- Clear, linear code
- "Suspends" execution in the middle of your code

What it isn't

- NOT multi-threaded code
- NOT a library -- native JavaScript!

Inspired by...

- 2012: Microsoft adds async/await to .NET
- Staple of async programming

Asynchronous programming 201

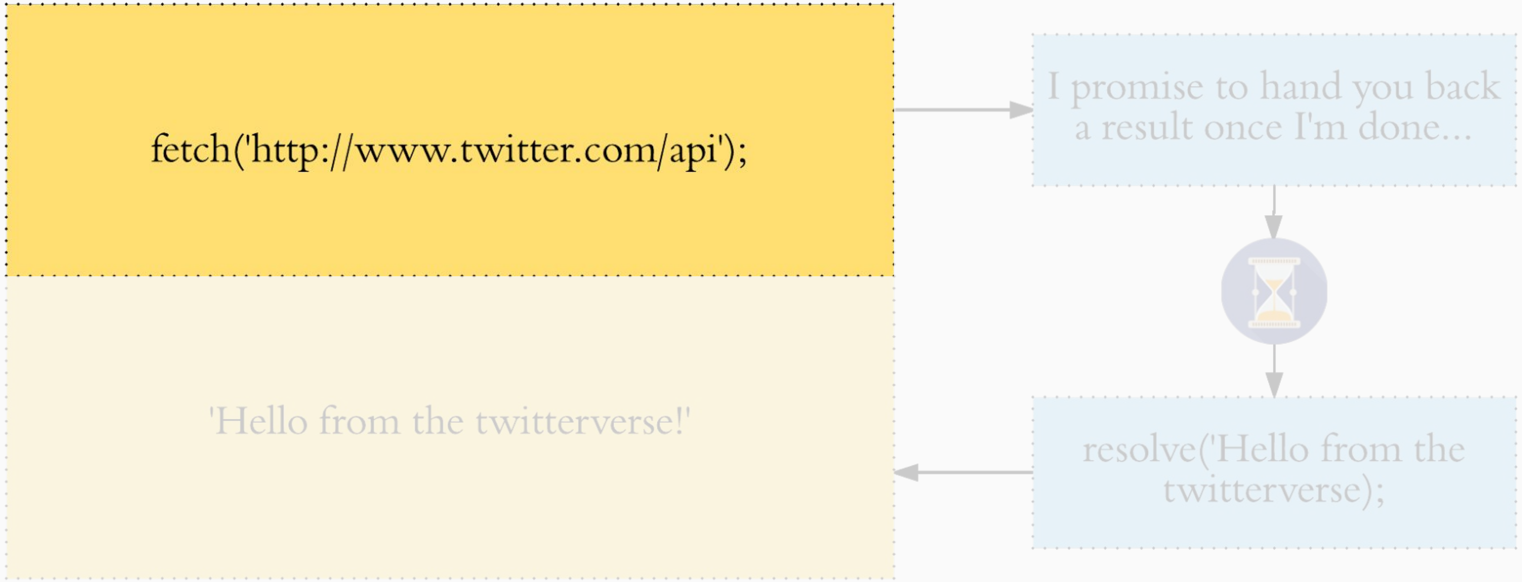
- Asynchronous is still everywhere
- Asynchronous is easy
- Asynchronous is pretty

Why this is awesome

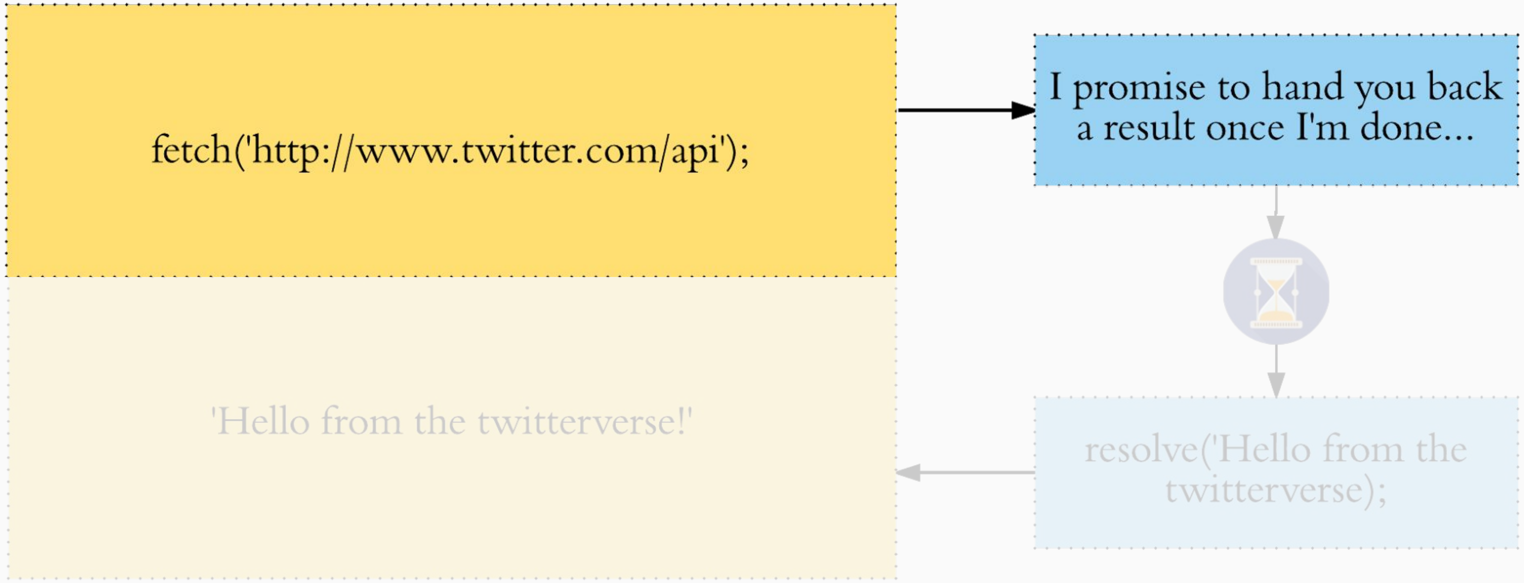
Promises

Promise?
Did You Get...
that thing I sent you?
Reject
Resolve

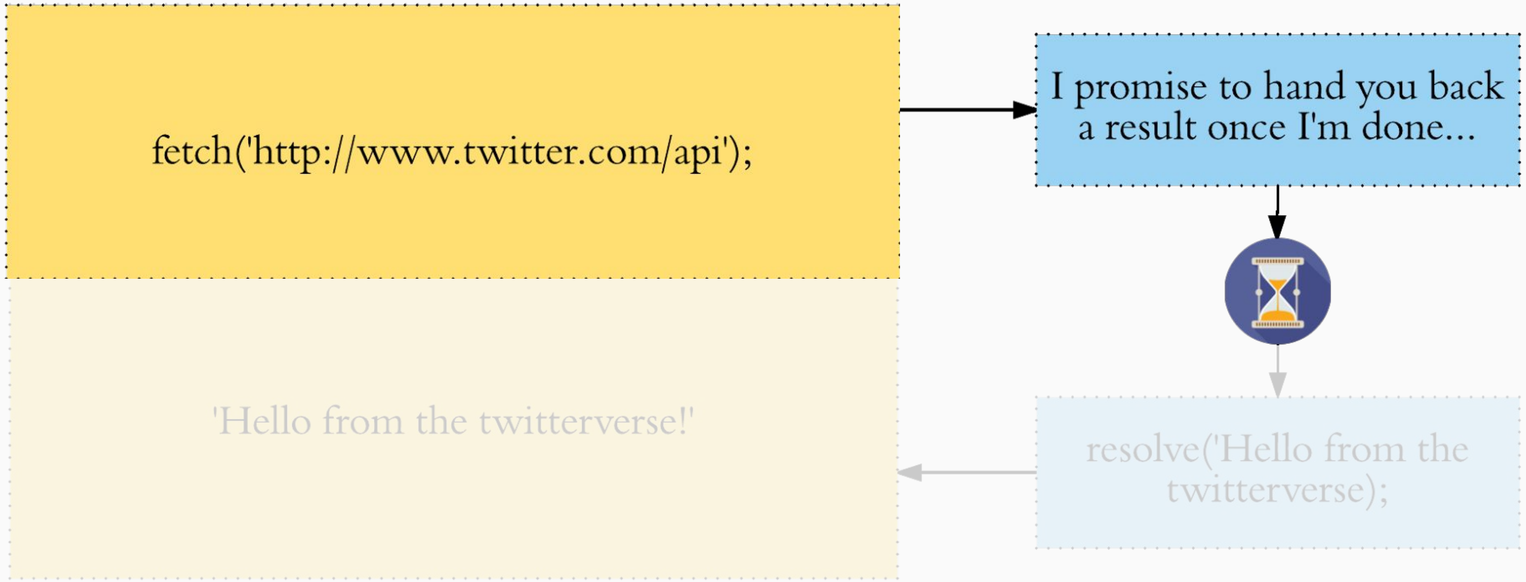
Promises


fetch('http://www.twitter.com/api');
'Hello from the twitterverse!'
I promise to hand you back a result once I'm done...
resolve('Hello from the twitterverse);

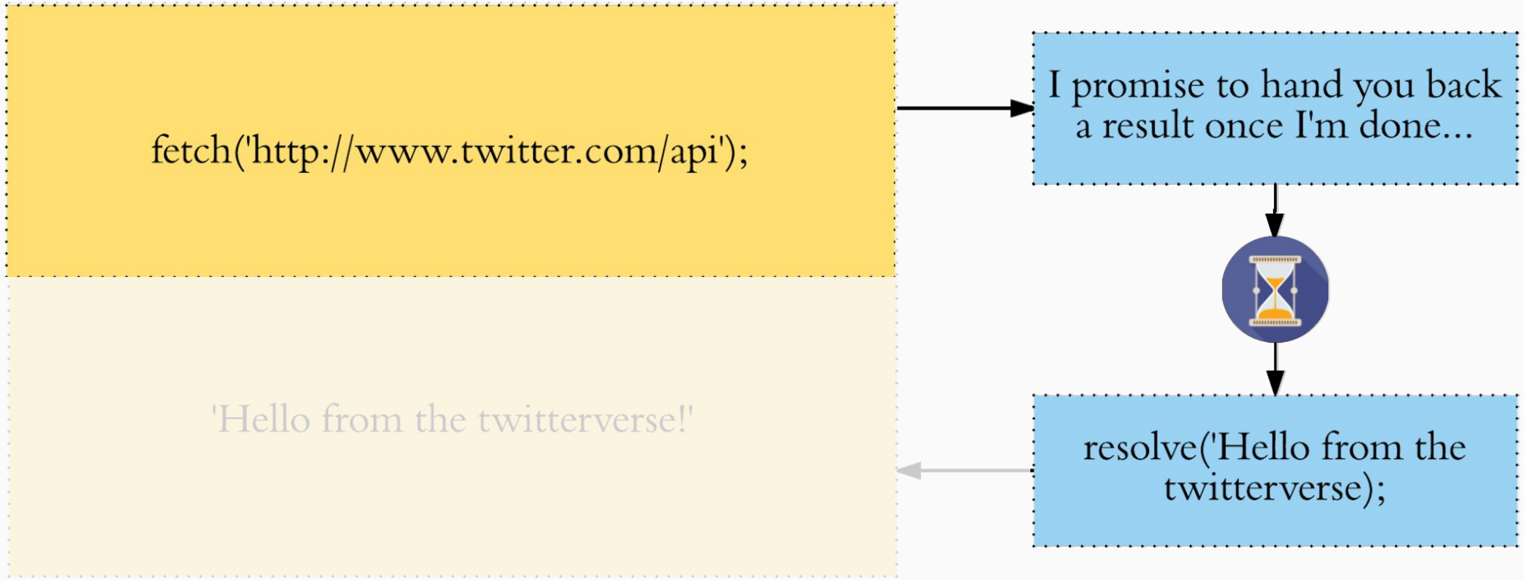
Promises


fetch('http://www.twitter.com/api');
I promise to hand you back a result once I'm done...
'Hello from the twitterverse!'
resolve('Hello from the twitterverse');

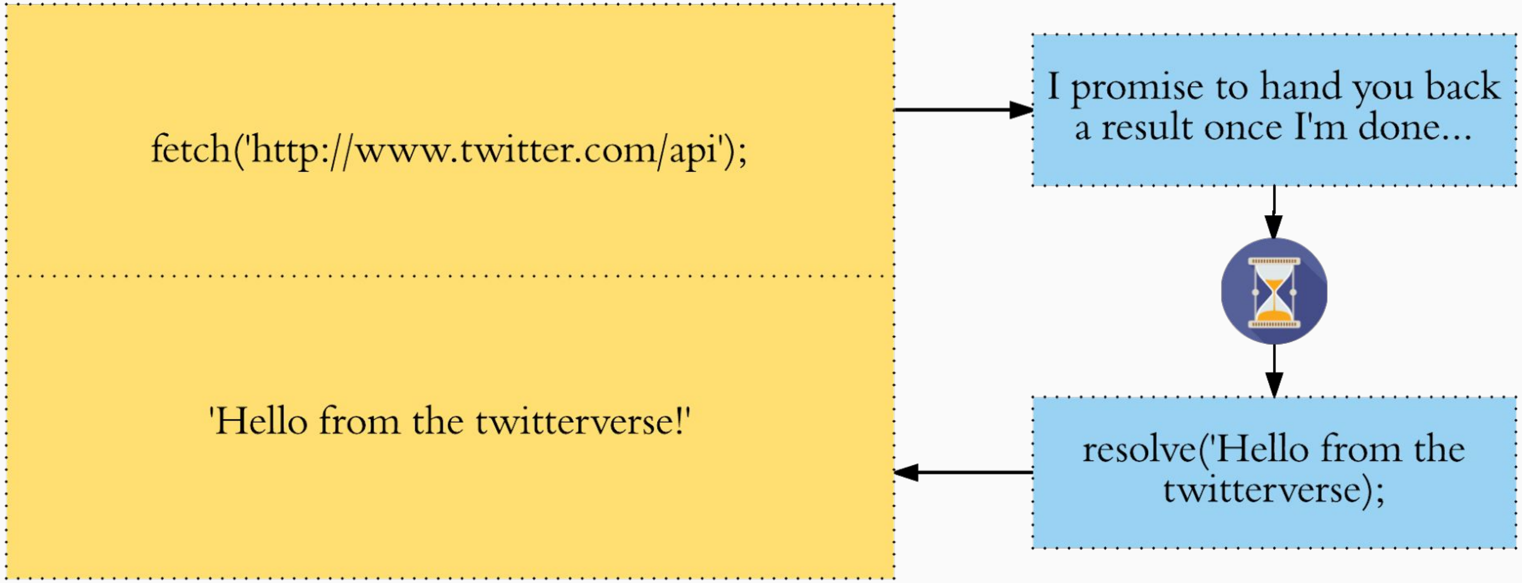
Promises


fetch('http://www.twitter.com/api');
I promise to hand you back a result once I'm done...
resolve('Hello from the twitterverse);
'Hello from the twitterverse!'

Promises


fetch('http://www.twitter.com/api');
I promise to hand you back a result once I'm done...
'Hello from the twitterverse!'
resolve('Hello from the twitterverse);

Promises


fetch('http://www.twitter.com/api');
'Hello from the twitterverse!'
I promise to hand you back a result once I'm done...
resolve('Hello from the twitterverse);

Using `then`

```
fetch('http://twitter.com/api');
```

Using `then`

```
fetch('http://twitter.com/api');
```

Could take awhile...

Using `then`

```
fetch('http://twitter.com/api')
  .then(result => {
    return fetch('http://twitter.com/api/retweet/'+result[0].id);
  });
```

Here's what I want to do when operation is complete

Using `then`

```
fetch('http://twitter.com/api')
  .then(result => {
    return fetch('http://twitter.com/api/retweet/'+result[0].id);
  })
  .then(result => {
    console.log(result);
  });
```

The response

The same thing, with async/await

Using async/await

```
async function getTweets() {
 const tweets = await fetch('http://twitter.com/api');
 const result = await fetch('http://twitter.com/retweet/'+tweets[0].id);
 console.log(result);
}
```

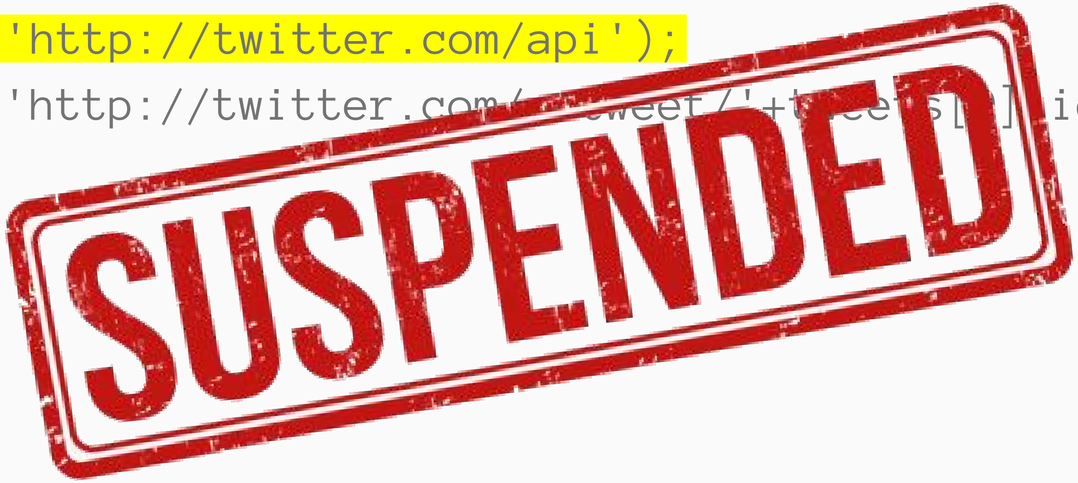
Using async/await

Could take awhile

```
async function getTweets() {
 const tweets = await fetch('http://twitter.com/api');
 const result = await fetch('http://twitter.com/retweet/'+tweets[0].id);
 console.log(result);
}
```

Using async/await

```
async function getTweets() {
 const tweets = await fetch('http://twitter.com/api');
 const result = await fetch('http://twitter.com/tweet/'+tweets[0].id);
 console.log(result);
}
```



Will not get to next line until promise is resolved

Using async/await

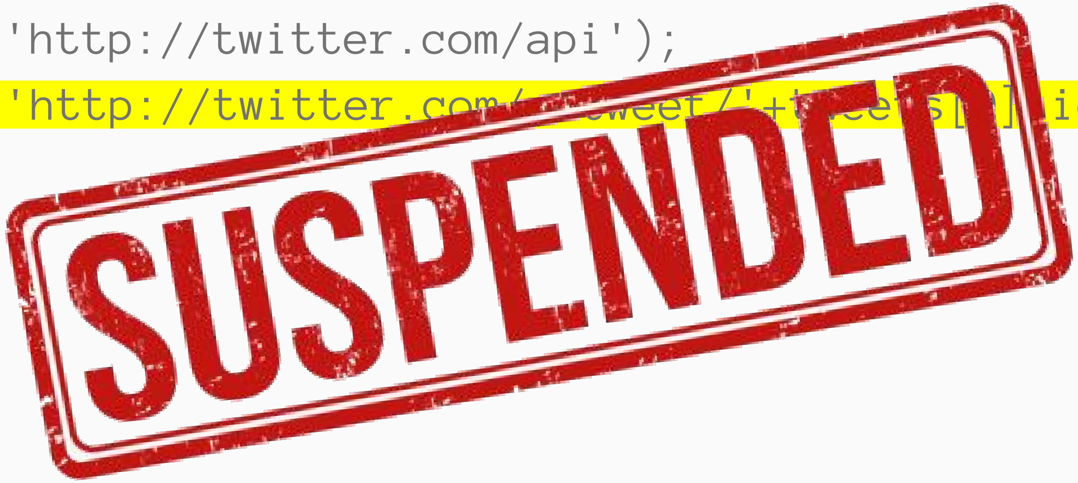
```
async function getTweets() {
 const tweets = await fetch('http://twitter.com/api');
 const result = await fetch('http://twitter.com/retweet/'+tweets[0].id);
 console.log(result);
}
```

Using async/await

```
async function getTweets() {
 const tweets = await fetch('http://twitter.com/api');
 const result = await fetch('http://twitter.com/retweet/'+tweets[0].id);
 console.log(result);
}
```

With async/await

```
async function getTweets() {
 const tweets = await fetch('http://twitter.com/api');
 const result = await fetch('http://twitter.com/retweet/'+tweets[0].id);
 console.log(result);
}
```



Will not get to next line until promise is resolved

Using async/await

```
async function getTweets() {
 const tweets = await fetch('http://twitter.com/api');
 const result = await fetch('http://twitter.com/retweet/'+tweets[0].id);
 console.log(result);
}
```

Using async/await

```
async function getTweets() {
 const tweets = await fetch('http://twitter.com/api');
 const result = await fetch('http://twitter.com/retweet/'+tweets[0].id);
 console.log(result);
}
```

```
fetch('http://twitter.com/api')
  .then(tweets => {
    return fetch('http://twitter.com/api/retweet/'+tweets[0].id);
  })
  .then(result => {
    console.log(result);
  });
```

```
const tweets = await fetch('http://twitter.com/api');
const result = await fetch('http://twitter.com/retweet/'+tweets[0].id);
console.log(result);
```

Works with loops too!

`for` loops

```
for (let i = 0; i < friends.length; i++) {
    await makeWallPost(friends[i]);
}
```

`while` loops

```
while (true) {
    await makeWallPost(friend);
}
```

`async` keyword

`async` keyword

```
const tweets = await fetch('https://twitter.com/api');
console.log(tweets);
```

```
Error: unexpected token
```

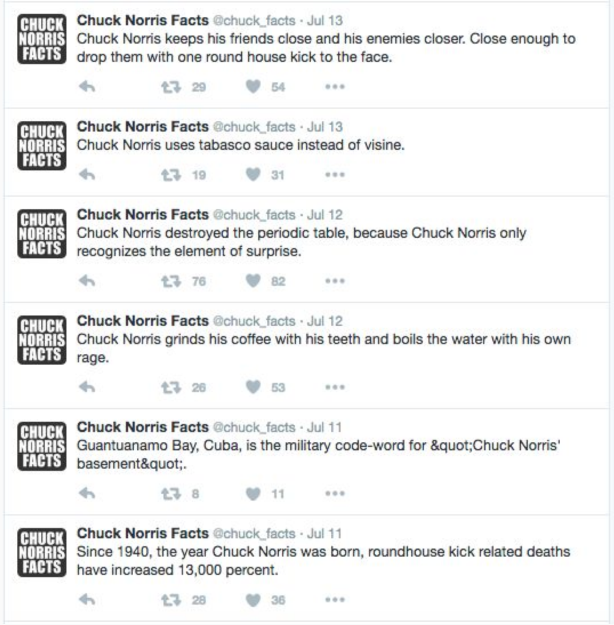
`async` keyword

```
async function fetchTweets() {
    const tweets = await fetch('https://twitter.com/api');
    console.log(tweets);
}
```

Rule of thumb

- Using await?
- Must be inside of function marked `async`

Live example



Chuck Norris Facts @chuck_facts · Jul 13
Chuck Norris keeps his friends close and his enemies closer. Close enough to drop them with one round house kick to the face.
29 54

Chuck Norris Facts @chuck_facts · Jul 13
Chuck Norris uses tabasco sauce instead of visine.
19 31

Chuck Norris Facts @chuck_facts · Jul 12
Chuck Norris destroyed the periodic table, because Chuck Norris only recognizes the element of surprise.
76 82

Chuck Norris Facts @chuck_facts · Jul 12
Chuck Norris grinds his coffee with his teeth and boils the water with his own rage.
26 53

Chuck Norris Facts @chuck_facts · Jul 11
Guantuanamo Bay, Cuba, is the military code-word for &quot;Chuck Norris' basement&quot;.
8 11

Chuck Norris Facts @chuck_facts · Jul 11
Since 1940, the year Chuck Norris was born, roundhouse kick related deaths have increased 13,000 percent.
28 36

Animation

Sequentially execute asynchronous stuff!

```
await moveTo(100, 100);
await sleep(500);
await spin();
await moveTo(300, 100);
await dance();
```

Implementing sleep()

```
await sleep(5000);
```

Implementing sleep()

```
await sleep(5000);

function sleep(milliseconds) {
    setTimeout(//something, milliseconds);
}
```

Implementing sleep()

```
await sleep(5000);

function sleep(milliseconds) {
    return new Promise(resolve => {
        ...
    });
}
```

Implementing sleep()

```
await sleep(5000);

function sleep(milliseconds) {
    return new Promise(resolve => {
        setTimeout(resolve, milliseconds);
    });
}
```

This causes `await` to move on to the next line

Animation Live Demo

Error handling

```
try {
    const result = await fetch('http://www.some404.com');
} catch(error) {
    // … Error: 404 NOT FOUND
}
```

Catch called on promise rejection

```
try {
    const result = await fetch('http://www.some404.com');
    const parsed = JSON.parse(result);
    document.getElementById('my-element').innerText = parsed;
} catch(error) {
    // … Error: 404 NOT FOUND
}
```

Might have other code in here too...

```
try {
    const result = await fetch('http://www.some404.com');
    const parsed = JSON.parse(result);
    document.getElementById('my-element').innerText = parsed;
} catch(error) {
    // … Error: 404 NOT FOUND
    // … Error: Could not parse JSON
    // … Error: Null reference error
}
```

Now many different errors could be caught!

```
try {
    const result = await fetch('http:/
    const parsed = JSON.parse(result);
    document.getElementById('my-elemen
} catch(error) {
    // … Error: 404 NOT FOUND
    // … Error: Could not parse JSON
    // … Error: Null reference error
}
```

`error` could be anything!

Keep try/catch blocks focused

```
let result = null;

try {
    result = await fetch('http://www.some404page.com');
} catch (error) {
    console.warn('Fetch failed!');
}

const parsed = JSON.parse(result)
document.getElementById('my-element').innerText = parsed;
```

Support

When can I use async/await?

~~Stage 0 ("Strawman")~~
~~Stage 1 ("Proposal")~~
~~Stage 2 ("Draft")~~
~~Stage 3 ("Candidate")~~
Stage 4 ("Finished")

When can I use async/await?

~~ES7~~

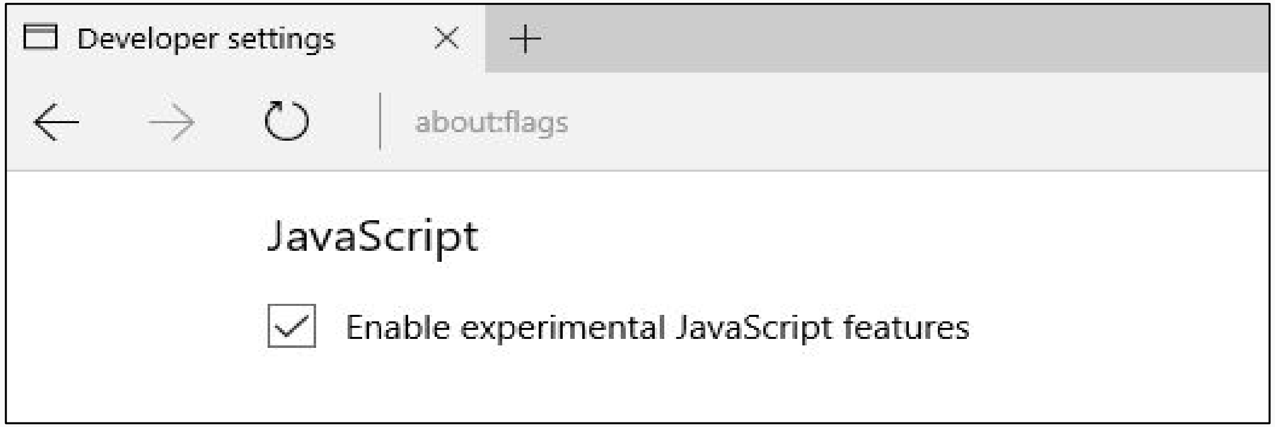
~~2016~~

ES8

2017

	Still working on it...
	Still working on it...
	Still working on it…
	Still working on it...
node JS™	Still working on it…

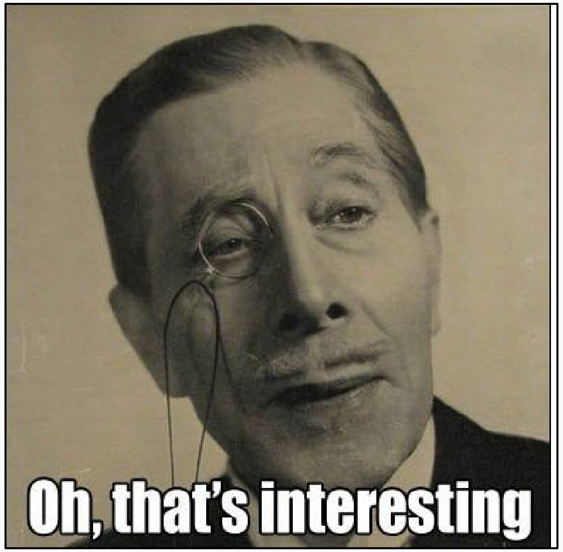
Experimental flag


Developer settings

about:flags

JavaScript

[✓] Enable experimental JavaScript features



BABEL

All the browsers!

package.json:

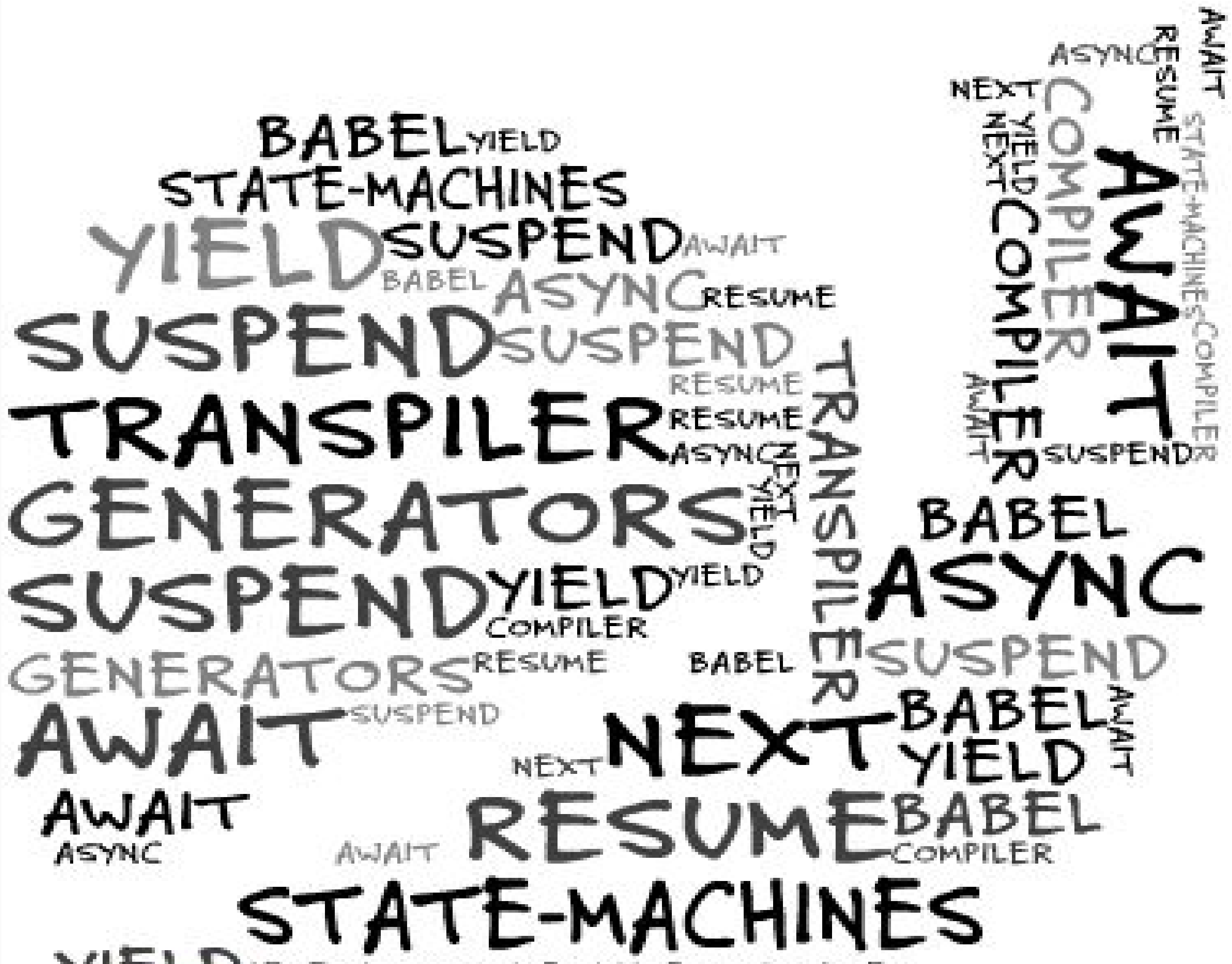
```
    "devDependencies": {
      "babel-plugin-transform-async-to-generator": "6.8.0"
    }
```

.babelrc:

```
    {
      "plugins": ["transform-async-to-generator"]
    }
```

So how exactly does this all work?

I"M NOT SAYING IT'S MAGIC
BUT MAGIC
H
HISTORY.COM


ASYNC
AWAIT
RESUME
NEXT
COMPILER
BABEL
YIELD
STATE-MACHINES
NEXT
YIELD
COMPILER
AWAIT
STATE-MACHINES
YIELD
SUSPEND
AWAIT
BABEL
ASYNC
RESUME
COMPILER
SUSPEND
SUSPEND
RESUME
AWAIT
TRANSPILER
RESUME
SUSPEND
ASYNC
NEXT
TRANSPILER
YIELD
GENERATORS
BABEL
ASYNC
SUSPEND
YIELD
YIELD
COMPILER
GENERATORS
RESUME
BABEL
SUSPEND
AWAIT
SUSPEND
NEXT
BABEL
AWAIT
NEXT
YIELD
AWAIT
RESUME
BABEL
ASYNC
AWAIT
COMPILER
STATE-MACHINES

Async/await

```
async function stuff() {
  await fetchSomething();
  await fetchSomethingElse();
  console.log('done!');
}
```

Generators

```
function *stuff() {
  yield fetchSomething();
  yield fetchSomethingElse();
  console.log('done!');
}
```

Generators

- Can suspend and then resume asynchronously!
- Async/await ="sugar syntax" on top of generators

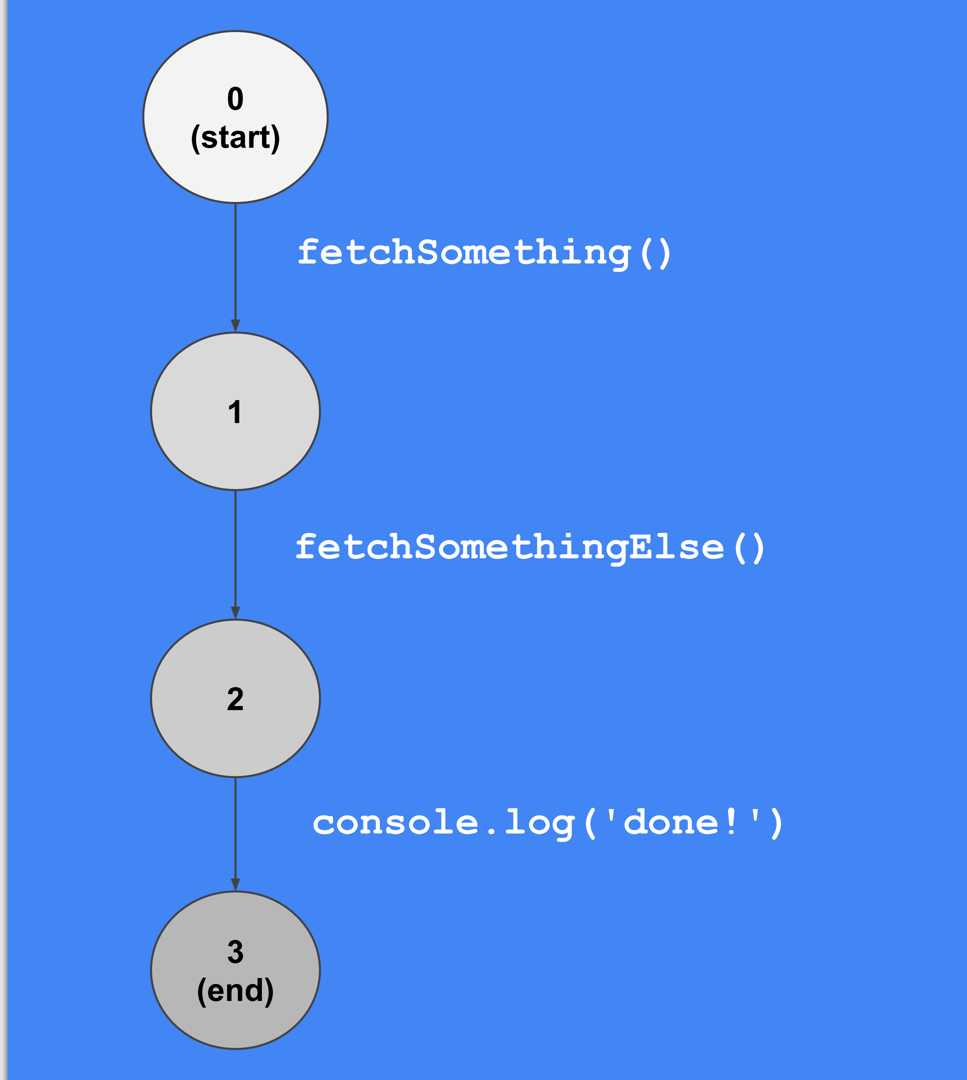
But how can a method "suspend"?!

State machine!

```
await fetchSomething();
await fetchSomethingElse();
console.log('done!')
```

State machine!

```
await fetchSomething();
await fetchSomethingElse();
console.log('done!')
```



State machine!

```
await fetchSomething();
await fetchSomethingElse();
console.log('done!')
```

```
let state = 0;

function loop() {
    if (state < 3) {
        do(args).then(loop);
    }
}

function do() {
  switch(state) {
    case 0:
      state = 1;
      return fetchSomething();
    case 1:
      state = 2
      return fetchSomethingElse();
    case 2:
      state = 3;
      console.log('done!');
      return;
    }
}
```

Gotcha #1:

Uncaught errors are swallowed

No try/catch?

```
function addOne() {
    x + 1;
}

// ReferenceError: x is not defined
```

No try/catch?

```
async function addOne() {
    x + 1;
}
```

```
Unhandled promise rejection
ReferenceError: Can't find variable: x es6.promise.js:119
```

```
Uncaught (in promise)
ReferenceError: x is not defined source.js:2
```

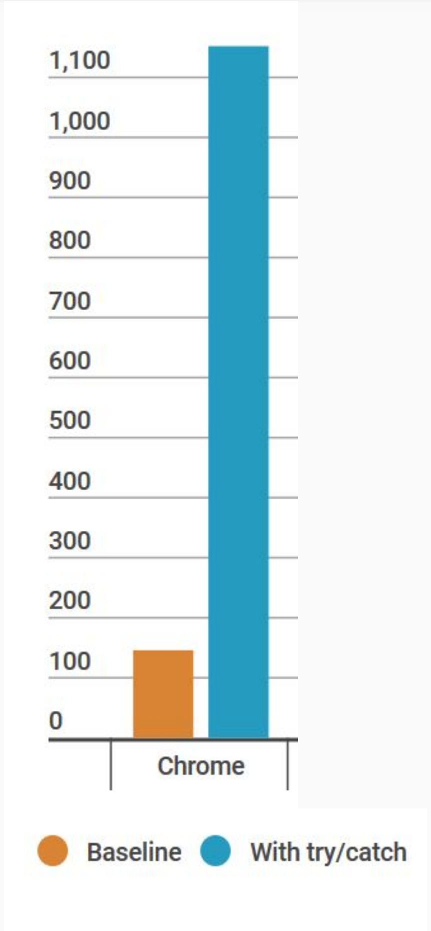
Gotcha #2:

Try/catch is slow

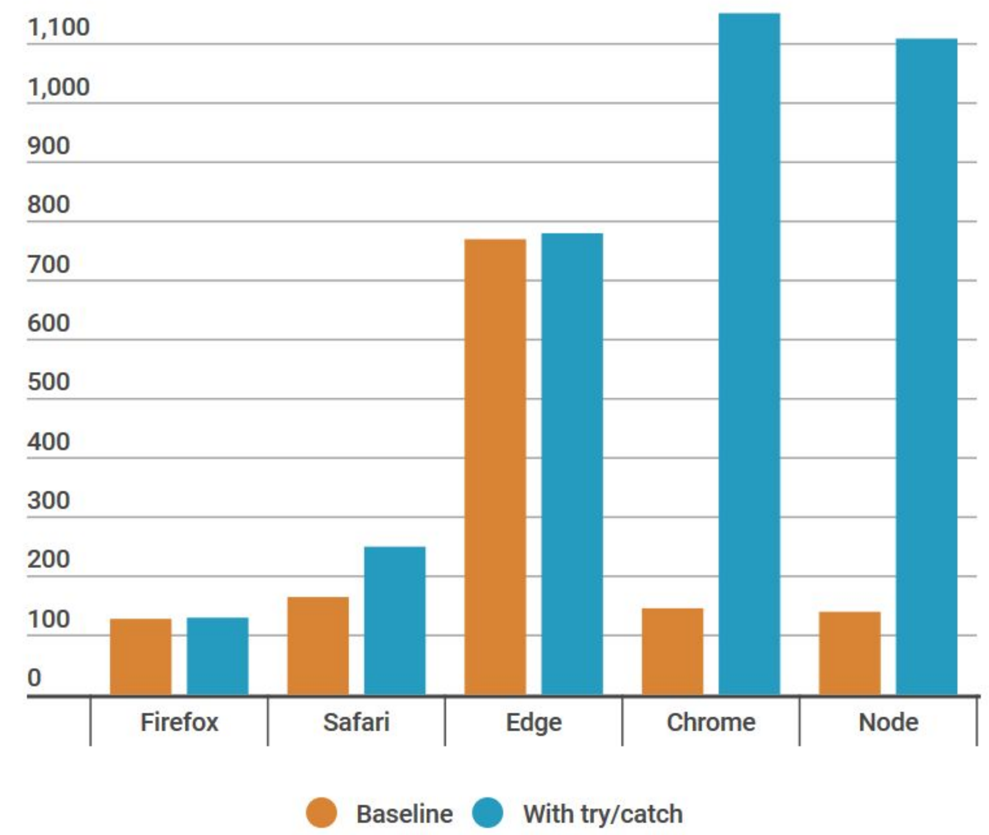
Try/Catch blocks not optimized!

- Some JS engines
- *Entire* function is skipped!
- Result: performance hit

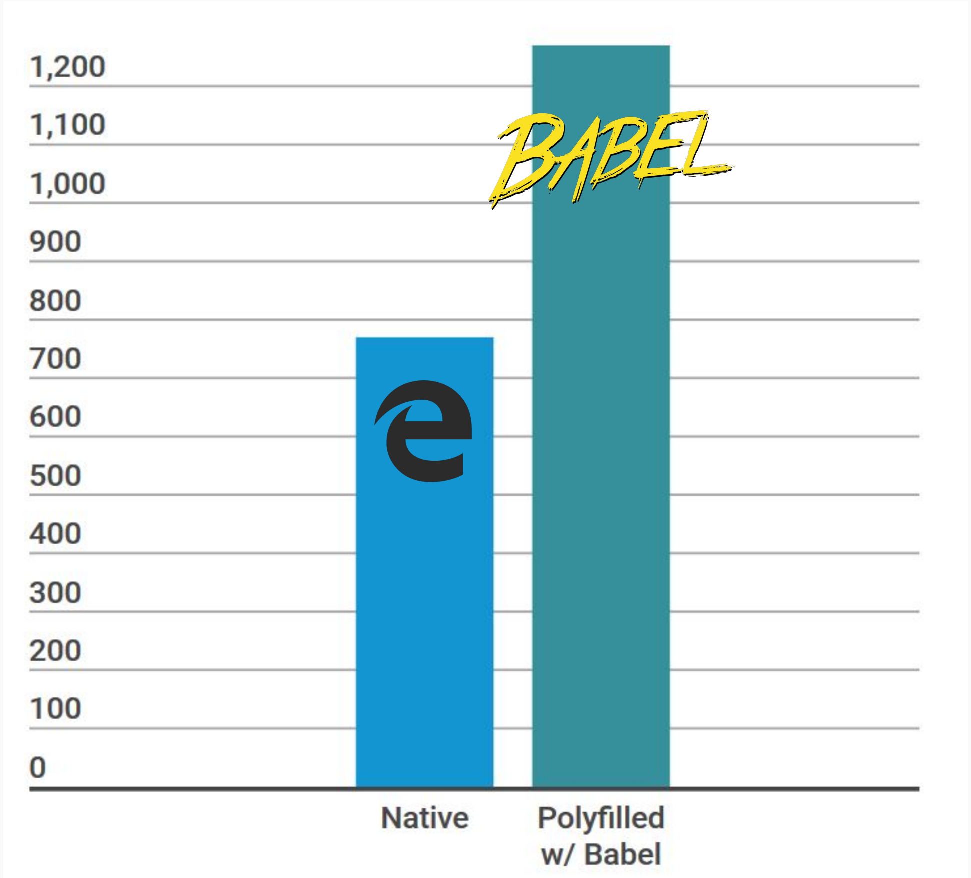
Execution time (ms)



Execution time (ms)



Execution time (ms)



Gotcha #3:

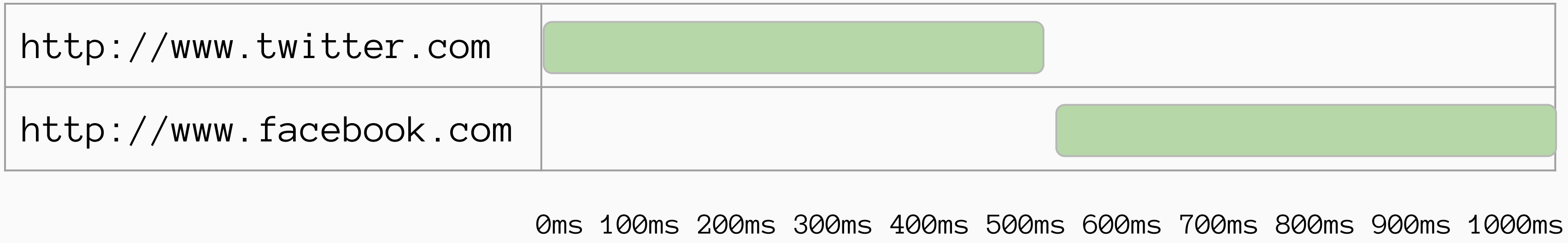
Parallelism

```
async function getPosts() {
    let twitter = await getTwitterPosts();
    let facebook = await getFacebookPosts();

    return {
        twitter,
        facebook
    };
}
```

Easy enough to get Facebook & Twitter posts, one after another

Network inspector

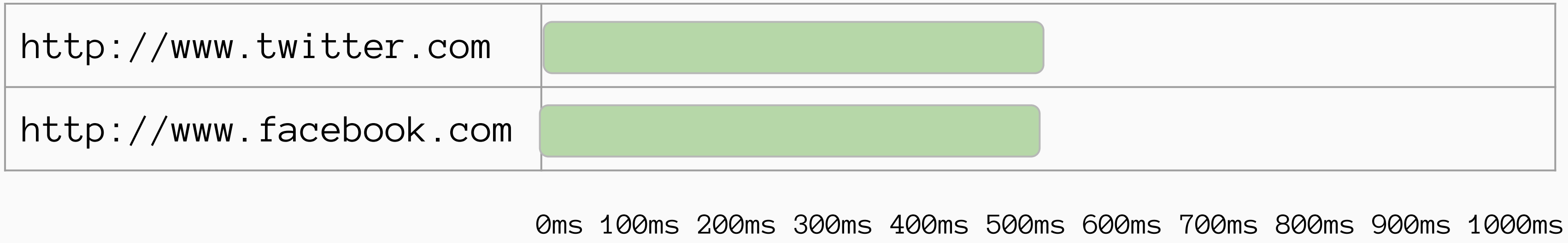


```
async function() {
    let [twitter, facebook] = await Promise.all(
        getTwitterPosts(),
        getFacebookPosts()
    );

    return {
        twitter,
        facebook
    };
}
```

Fetches both at same time!

Network inspector



Gotcha #4:

Higher order functions

Higher order functions

```
const urls = [
    'http://twitter.com/api',
    'http://facebook.com/api',
    'http://instagram.com/api'
];
```

Higher order functions

```
const urls = [
    'http://twitter.com/api',
    'http://facebook.com/api',
    'http://instagram.com/api'
];

urls.forEach(url => fetch(url));
```

Higher order functions

```
const urls = [
    'http://twitter.com/api',
    'http://facebook.com/api',
    'http://instagram.com/api'
];

urls.forEach(url => await fetch(url));
```

Higher order functions

```
const urls = [
    'http://twitter.com/api',
    'http://facebook.com/api',
    'http://instagram.com/api'
];

urls.forEach(url => await fetch(url));

// Unexpected token
```

Higher order functions

```
const urls = [
    'http://twitter.com/api',
    'http://facebook.com/api',
    'http://instagram.com/api'
];

urls.forEach(async url => await fetch(url));
```

Higher order functions

```
const urls = [
    'http://twitter.com/api',
    'http://facebook.com/api',
    'http://instagram.com/api'
];

urls.forEach(async url => await fetch(url));
```

Works!
But executes all of them in parallel!?

Higher order functions

```
const urls = [
    'http://twitter.com/api',
    'http://facebook.com/api',
    'http://instagram.com/api'
];

for (let i = 0; i < urls.length; i++) {
    await fetch(urls[i]);
}
```

Yep

Gotcha #5:

Angular digests

Angular

```
async function initialize() {
    $scope.user = await getUser();
    $scope.messages = await getMessagesFor($scope.user);
}
```

Angular

```
async function initialize() {
    $scope.user = await getUser();
    $scope.$apply();

    $scope.messages = await getMessagesFor($scope.user);
    $scope.$apply();
}
```

Angular

```
> npm install angular-async-await

$async(async function() {
    $scope.user = await getUser();
    $scope.messages = await getMessagesFor($scope.user);
});
```

Automatically calls apply()

Questions?