Callbacks, Promises, and Coroutines (oh my!)

Asynchronous Programming Patterns in JavaScript

Domenic Denicola
http://domenicdenicola.com
@domenicdenicola

In non-web languages,

most of the code we write is *synchronous*.

aka *blocking*

```
Console.WriteLine("What is your name?");
string name = Console.ReadLine();
Console.WriteLine("Hello, " + name);
```

```
var fileNames = Directory.EnumerateFiles("C:\
\");

foreach (var fileName in fileNames)
{
  using (var f = File.Open(fileName,
FileMode.Open))
  {
    Console.WriteLine(fileName + " " +
f.Length);
  }
}
```

```
using (var client = new WebClient())
{
  string html = client.DownloadString("http://
news.ycombinator.com");

  Console.WriteLine(html.Contains("Google"));
  Console.WriteLine(html.Contains("Microsoft"));
  Console.WriteLine(html.Contains("Apple"));
}
```

`Thread.Start`

`BackgroundWorker`

`Control.InvokeRequired`

This often causes us some pain…

… but hey, there's always threads!

`Dispatcher.Invoke`

`ThreadPool`

`.AsParallel()`

Q: What are these threads doing, most of the time?

A: *waiting*


WTF!?

In JavaScript, we do things differently.

There’s only one thread in JavaScript,

so we use that thread to get stuff done.

OK, let’s talk about…

- The event loop
- Callbacks
- Promises
- Coroutines


THE EVENT LOOP

You've seen event loops before:

```
int WINAPI WinMain(HINSTANCE hInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpCmdLine,
                   int nCmdShow)
{
  MSG msg;
  while (GetMessage(&msg, NULL, 0, 0) > 0)
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return msg.wParam;
}
```

```
this.btnOK.Click += this.btnOK_Click;


private void btnOK_Click(object sender,
                         EventArgs e)
{
   // ...
}
```

```
$("#ok-button").click(function () {
    // ...
});

setTimeout(function () {
    // ...
}, 100);

$.get("http://example.com", function (result) {
    // ...
});
```

Some event loop subtleties

- Yielding
- Async’s not sync
- Errors
- It’s not magic

Event Loop Subtleties

Yielding

```
console.log("1");

$.get("/echo/2", function (result) {
  console.log(result);
});

console.log("3");

// 1, 3, 2
```

Async's not sync

```
var hi = null;

$.get("/echo/hi", function (result) {
  hi = result;
});

console.log(hi);

// null
```

Errors

```
console.log("About to get the website...");

$.ajax("http://sometimesdown.example.com", {
  success: function (result) {
    console.log(result);
  },
  error: function () {
    throw new Error("Error getting the
website");
  }
});

console.log("Continuing about my business...");
```

It's not magic

```
function fib(n) {
    return n < 2 ? 1 : fib(n-2) + fib(n-1);
}

console.log("1");

setTimeout(function () {
   console.log("2");
}, 100);

fib(40);

// 1 ... 15 seconds later ... 2
```

http://teddziuba.com/2011/10/node-js-is-cancer.html

The event loop is tricky… but powerful.


CALLBACKS

What we’ve seen so far has been doing asynchronicity through *callbacks*.

Callbacks are OK for simple operations, but force us into *continuation passing style*.

Recurring StackOverflow question:

```
function getY() {
  var y;
  $.get("/gety", function (jsonData) {
    y = jsonData.y;
  });
  return y;
}

var x = 5;
var y = getY();

console.log(x + y);
```

Why doesn't it work???

After getting our data, we have to do everything else in a *continuation*:

```
function getY(continueWith) {
  $.get("/gety", function (jsonData) {
    continueWith(jsonData.y);
  });
}
```

CPS Headaches

- Doing things in sequence is hard
- Doing things in parallel is harder
- Errors get lost easily

CPS
Headaches

Doing things in sequence is hard

```
$("#button").click(function () {
  promptUserForTwitterHandle(function (handle) {
    twitter.getTweetsFor(handle, function
(tweets) {
      ui.show(tweets);
    });
  });
});
```

CPS
Headaches

Doing things in parallel is harder

```
var tweets, answers, checkins;

twitter.getTweetsFor("domenicdenicola", function (result)
{
  tweets = result;
  somethingFinished();
});
stackOverflow.getAnswersFor("Domenic", function (result) {
  answers = result;
  somethingFinished();
});
fourSquare.getCheckinsBy("Domenic", function (result) {
  checkins = result;
  somethingFinished();
});
```

Doing things in parallel is harder

```
var finishedSoFar = 0;

function somethingFinished() {
  if (++finishedSoFar === 3) {
    ui.show(tweets, answers, checkins);
  }
}
```

CPS
Headaches

Errors get lost easily

```
function getTotalFileLengths(path, callback) {
  fs.readdir(path, function (err, fileNames) {
    var total = 0;

    var finishedSoFar = 0;
    function finished() {
      if (++finishedSoFar === fileNames.length) {
        callback(total);
      }
    }

    fileNames.forEach(function (fileName) {
      fs.readFile(fileName, function (err, file) {
        total += file.length;
        finished();
      });
    });
  });
}
```

You could write your own library to make this nicer…

```
function parallel(actions, callback) {
  var results = [];
  function finished(result) {
    results.push(result);
    if (results.length === actions.length) {
        callback(results);
    }
  }

  actions.forEach(function (action) {
    action(finished);
  });
}
```

```
parallel([
  function (cb) {
    twitter.getTweetsFor("domenicdenicola", cb);
  },
  function (cb) {
    stackOverflow.getAnswersFor("Domenic", cb);
  },
  function (cb) {
    fourSquare.getCheckinsFor("Domenic", cb);
  }
], function (results) {
  console.log("tweets = ", results[0]);
  console.log("answers = ", results[1]);
  console.log("checkins = ", results[2]);
});
```

And in fact many people have:

Control flow / Async goodies

- async.js — Async chaining and file system utilities. Async.js is to node's fs module, what jQuery is to the DOM.
- async — Comprehensive async map/reduce and control flow (parallel, series, waterfall, auto...) module that works in node a
- atbar — Async callback manager for javascript in nodejs and browser
- begin — Control flow library for node.js and CoffeeScript
- chainsaw — Build chainable fluent interfaces the easy way in node.js
- channels — Event channels for node.js
- Cinch — Write async code in sync form.
- deferred — Asynchronous control-flow with deferred and promises
- each — Chained and parallel async iterator in one elegant function
- fiberize — Node API wrapper for use with fibers.
- fibers — The closest thing to a thread you'll find in JavaScript
- fibers-promise — Small yet powerful promises based on fibers.
- first — A tiny control-flow library.
- flow-js — Continuation-esque contruct for expressing multi-step asynchronous logic
- funk — Asynchronous parallel functions made funky!
- futures — Asynchronous Method Queueing, Futures, Promises, Subscriptions, and other async goodies

https://github.com/joyent/node/wiki/modules#wiki-async-flow

The best of these (IMO) are based on an abstraction called “promises”


You promise?
PROMISES

Un-inverts the chain of responsibility:

instead of calling a passed callback, return a promise.

```
addWithCallback(a, b, function (result) {
  assert.equal(result, a + b);
});

var promise = addWithPromise(a, b);

promise.then(function (result) {
  assert.equal(result, a + b);
});
```

Why promises are awesome

- Cleaner method signatures
- Uniform return/error semantics
- Easy composition
- Easy sequential/parallel join
- Always async
- Exception-style error bubbling

Cleaner method signatures
Uniform return/error semantics

```
$.get(
  url,
  [data],
  [success(data, status, xhr)],
  [dataType]
)
```

Cleaner method signatures
Uniform return/error semantics

```
$.ajax(url, settings)

settings.success(data, status, xhr)
settings.error(xhr, status, errorThrown)
settings.complete(xhr, status)
```

Cleaner method signatures
Uniform return/error semantics

```
fs.open(
  path,
  flags,
  [mode],
  [callback(error, file)]
)
```

Cleaner method signatures
Uniform return/error semantics

```
fs.write(
  file,
  buffer,
  offset,
  length,
  position,
  [callback(error, written, buffer)]
)
```

Cleaner method signatures
Uniform return/error semantics

```
getAsPromise(url, [data], [dataType]).then(
  function onFulfilled(result) {
    var data = result.data;
    var status = result.status;
    var xhr = result.xhr;
  },
  function onBroken(error) {
    console.error("Couldn't get", error);
  }
);
```

Easy composition

```
function getUser(userName, onSuccess, onError) {
  $.ajax("/user?" + userName, {
    success: onSuccess,
    error: onError
  });
}
```

Easy composition

```
function getUser(userName) {
  return getAsPromise("/user?" + userName);
}
```

Easy composition

```
function getFirstName(userName, onSuccess,
onError) {
  $.ajax("/user?" + userName, {
    success: function successProxy(data) {
      onSuccess(data.firstName);
    },
    error: onError
  });
}
```

Promises are Awesome

Easy composition

```
function getFirstName(userName) {
  return getAsPromise("/user?" + userName)
           .get("firstName");
}
```

Easy sequential join

```
$("#button").click(function () {
  promptUserForTwitterHandle(function (handle) {
    twitter.getTweetsFor(handle, function
(tweets) {
      ui.show(tweets);
    });
  });
});
```

Easy sequential join

```
$("#button").clickPromise()
  .then(promptUserForTwitterHandle)
  .then(twitter.getTweetsFor)
  .then(ui.show);
```

Promises are Awesome

Easy parallel join

```
var tweets, answers, checkins;

twitter.getTweetsFor("domenicdenicola", function (result) {
  tweets = result;
  somethingFinished();
});
stackOverflow.getAnswersFor("Domenic", function (result) {
  answers = result;
  somethingFinished();
});
fourSquare.getCheckinsBy("Domenic", function (result) {
  checkins = result;
  somethingFinished();
});
```

Promises are Awesome

Easy parallel join

```
Q.all([
  twitter.getTweetsFor("domenicdenicola"),
  stackOverflow.getAnswersFor("Domenic"),
  fourSquare.getCheckinsBy("Domenic")
]).then(function (results) {
  console.log(results[0], results[1],
results[2]);
});
```

Promises are Awesome

Easy parallel join

```
Q.all([
  twitter.getTweetsFor("domenicdenicola"),
  stackOverflow.getAnswersFor("Domenic"),
  fourSquare.getCheckinsBy("Domenic")
]).spread(function (tweets, answers, checkins) {
  console.log(tweets, answers, checkins);
});
```

Always async

```
function getUser(userName, onSuccess, onError) {
  if (cache.has(userName)) {
    onSuccess(cache.get(userName));
  } else {
    $.ajax("/user?" + userName, {
      success: onSuccess,
      error: onError
    });
  }
}
```

Always async

```
console.log("1");

getUser("ddenicola", function (user) {
  console.log(user.firstName);
});

console.log("2");

// 1, 2, Domenic
```

Always async

```
console.log("1");

getUser("ddenicola", function (user) {
  console.log(user.firstName);
});

console.log("2");

// 1, Domenic, 2
```

Always async

```
function getUser(userName) {
  if (cache.has(userName)) {
    return Q.ref(cache.get(userName));
  } else {
    return getWithPromise("/user?" + userName);
  }
}
```

Always async

```
console.log("1");

getUser("ddenicola").then(function (user) {
  console.log(user.firstName);
});

console.log("2");

// 1, 2, Domenic (every time)
```

Exception-style error bubbling

```
getUser("Domenic", function (user) {
  getBestFriend(user, function (friend) {
    ui.showBestFriend(friend);
  });
});
```

Exception-style error bubbling

```
getUser("Domenic", function (err, user) {
  if (err) {
    ui.error(err);
  } else {
    getBestFriend(user, function (err, friend) {
      if (err) {
        ui.error(err);
      } else {
        ui.showBestFriend(friend);
      }
    });
  }
});
```

Exception-style error bubbling

```
getUser("Domenic")
  .then(getBestFriend, ui.error)
  .then(ui.showBestFriend, ui.error);
```

Exception-style error bubbling

```
getUser("Domenic")
  .then(getBestFriend)
  .then(ui.showBestFriend, ui.error);
```

Exception-style error bubbling

```
ui.startSpinner();
getUser("Domenic")
  .then(getBestFriend)
  .then(
    function (friend) {
      ui.showBestFriend(friend);
      ui.stopSpinner();
    },
    function (error) {
      ui.error(error);
      ui.stopSpinner();
    }
  );
```

Exception-style error bubbling

```
ui.startSpinner();
getUser("Domenic")
  .then(getBestFriend)
  .then(ui.showBestFriend, ui.error)
  .fin(ui.stopSpinner);
```

Exception-style error bubbling

```
function getBestFriendAndDontGiveUp(user) {
  return getUser(user).then(
    getBestFriend,
    function (error) {
      if (error instanceof TemporaryNetworkError) {
        console.log("Retrying after error", error);
        return getBestFriendAndDontGiveUp(user);
      }
      throw error;
    });
}
```

Sounds great. How do I get in on this action?

Use Q

- By Kris Kowal, @kriskowal
- https://github.com/kriskowal/q
- Can consume promises from jQuery etc.
- Implements various CommonJS standards

If you’re already using jQuery’s promises, switch to Q:

https://github.com/kriskowal/q/wiki/jQuery


I DON'T ALWAYS USE PROMISES
BUT WHEN I DO I USE Q
memegenerator.net

Creating promises with Q

Fulfilling promises

```
// We have:
setTimeout(doSomething, 1000);

// We want:
delay(1000).then(doSomething);
```

Creating promises with Q

Fulfilling promises

```
function delay(ms) {
  var deferred = Q.defer();
  setTimeout(deferred.resolve, ms);
  return deferred.promise;
}

delay(1000).then(doSomething);
```

Creating promises with Q

Breaking promises

```
function getWithTimeout(url, ms, onSuccess, onError) {
  var isTimedOut = false, isHttpErrored = false;

  setTimeout(function () {
    if (!isHttpErrored) {
      isTimedOut = true;
      onError(new Error("timed out"));
    }
  }, ms);

  $.ajax(url, {
    success: function (result) {
      if (!isTimedOut) { onSuccess(result); }
    },
    error: function (xhr, status, error) {
      if (!isTimedOut) {
        isHttpErrored = true;
        onError(error);
      }
    }
  });
}
```

Creating promises with Q

Breaking promises

```
function getWithTimeout(url, ms) {
  var deferred = Q.defer();

  setTimeout(function () {
    deferred.reject(new Error("timed out"));
  }, ms);

  $.ajax(url, {
    success: deferred.resolve,
    error: deferred.reject
  });

  return deferred.promise;
}
```

Creating promises with Q

Building abstractions

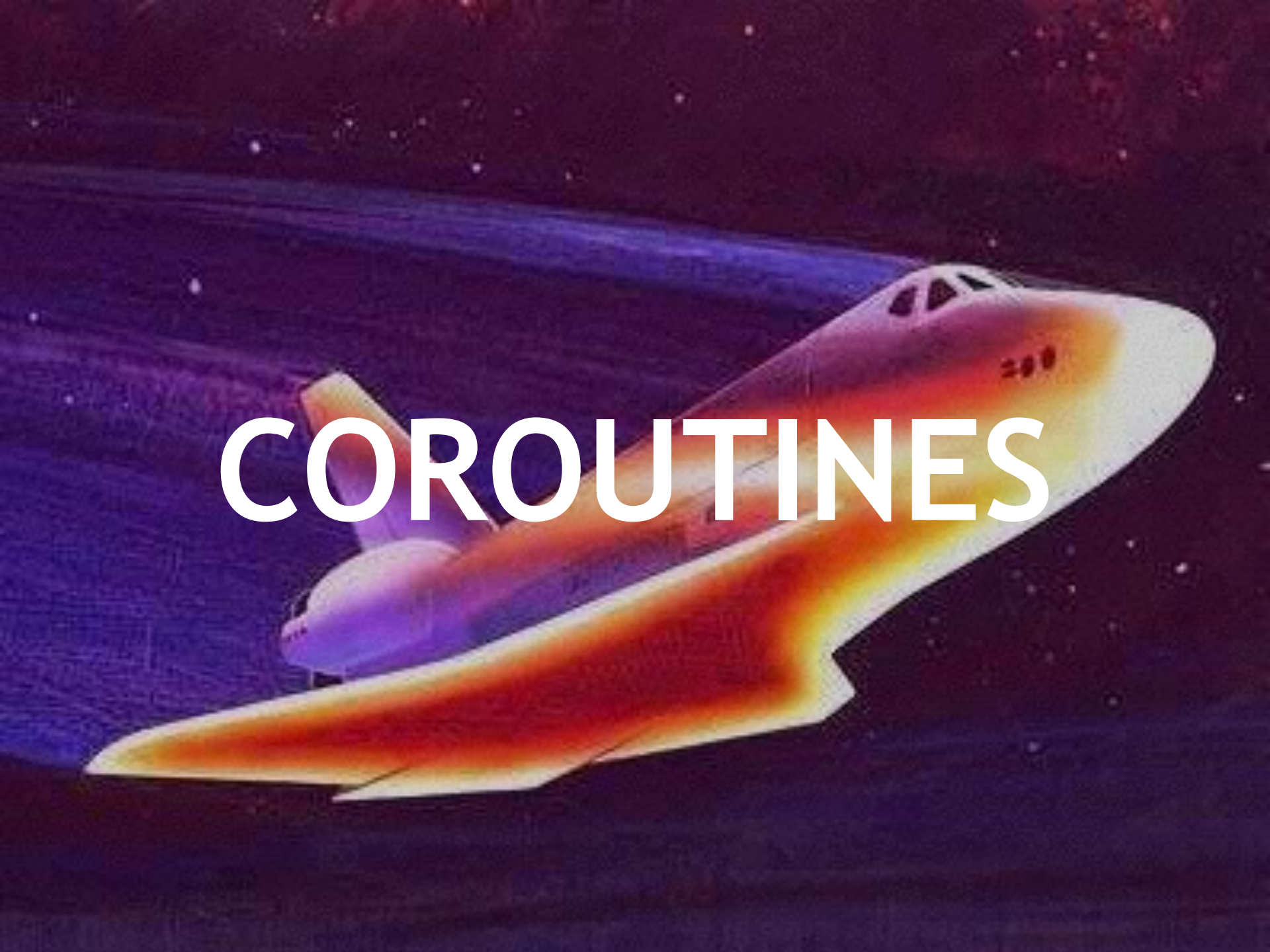
```
function timeout(promise, ms) {
  var deferred = Q.defer();
  promise.then(deferred.resolve, deferred.reject);

  setTimeout(function () {
    deferred.reject(new Error("timed out"));
  }, ms);

  return deferred.promise;
}

function getWithTimeout(url, ms) {
  return timeout(getAsPromise(url), ms);
}
```

Promises are cool.

They clean up our method signatures.

They’re composable, they’re joinable, and they’re dependably async.

But… we still have to write in CPS.


COROUTINES

“Coroutines are computer program components that generalize subroutines to allow multiple entry points for suspending and resuming execution at certain locations.”

http://en.wikipedia.org/wiki/Coroutine


WTF!?

Nice:

```
var xP = getX();
var yP = getY();
var zP = getZ();

Q.all([xP, yP, zP]).spread(function (x, y, z) {
  console.log(x + y + z);
});
```

Nicer:

```
var [x, y, z] = await Q.all([getX(), getY(),
getZ()]);

console.log(x + y + z);
```

Nice:

```
$("#button").clickPromise()
  .then(promptUserForTwitterHandle)
  .then(twitter.getTweetsFor)
  .then(ui.show);
```

Nicer:

```
await $("#button").clickPromise();

var handle = await promptUserForTwitterHandle();
var tweets = await twitter.getTweetsFor(handle);

ui.show(tweets);
```

Q: Can’t the compiler do this for me?

A: yes… if you are willing to introduce a compiler.

Several options, none perfect

- Kaffeine: http://weepy.github.com/kaffeine/
- Traceur: http://tinyurl.com/traceur-js
- TameJS: http://tamejs.org/
- Node fork: http://tinyurl.com/node-async

Q: OK well… can’t the interpreter do this for me?

A: yes… if you’re willing to wait for the next version of JS.

The next version of JavaScript (“ECMAScript Harmony”) has a limited form of coroutines that can be twisted to do something like what we want.

ECMAScript Harmony generators

```
function* fibonacci() {
  var [prev, curr] = [0, 1];
  for (;;) {
    [prev, curr] = [curr, prev + curr];
     yield curr;
  }
}

for (n of fibonnaci()) {
  console.log(n);
}
```

http://wiki.ecmascript.org/doku.php?id=harmony:generators

ECMAScript Harmony generators

```
var eventualAdd = Q.async(function* (pA, pB) {
  var a = yield pA;
  var b = yield pB;
  return a + b;
});
```

https://github.com/kriskowal/q/tree/master/examples/async-generators

ECMAScript Harmony generators

```
// Can only use yield as we want to within
// Q.async'ed generator functions

Q.async(function* () {
  // Talk to the server to get one and two.
  var three = yield eventualAdd(getOne(),
getTwo());

  assert.equal(three, 3);
})();
```

https://groups.google.com/d/topic/q-continuum/7PWKbgeFA48/discussion

ECMAScript Harmony generators

```
// Given promise-returning delay(ms) as before:

var animateAsync = Q.async(function* (el) {
  for (var i = 0; i < 100; ++i) {
    element.style.left = i;
    yield delay(20);
  }
});
```

http://wiki.ecmascript.org/doku.php?id=strawman:async_functions

ECMAScript Harmony generators

```
Q.async(function* () {
  var el = document.getElementById("my-
element");

  yield animateAsync(el);

  console.log("it's done animating");
})();
```

https://groups.google.com/d/topic/q-continuum/7PWKbgeFA48/discussion

So coroutines are a bit of a mess,

but we'll see how things shape up.

Recap

- Async is here to stay
- But you don’t have to dive into callback hell
- Use promises
- Use Q
- Maybe use coroutines if you’re feeling brave

Thanks for listening!