Modern JavaScript and PhoneGap

PhoneGap Day EU 2017

Kerri Shotts • @kerrishotts

Image by Pete Linforth (https://pixabay.com/en/users/PeteLinforth-202249/), courtesy of Pixabay.com

Hi!

- Used PhoneGap for over six years
- Authored Five books about PhoneGap
- Apache Cordova committer
- One of many moderators at:
    - Cordova Google Group
    - PhoneGap Adobe Forums
- I love retro technology and ST:TNG 😉

Modern JavaScript Versions

Remember ECMAScript 5?

Release year: 2009

- The version we all know and love (~ish?)
- Supported by all modern mobile web views[1]
  - iOS 6+, IE 10+, Edge (forever), Android 4.4+
- Reasonably modern (`map` , `reduce` , getters/setters, etc.)

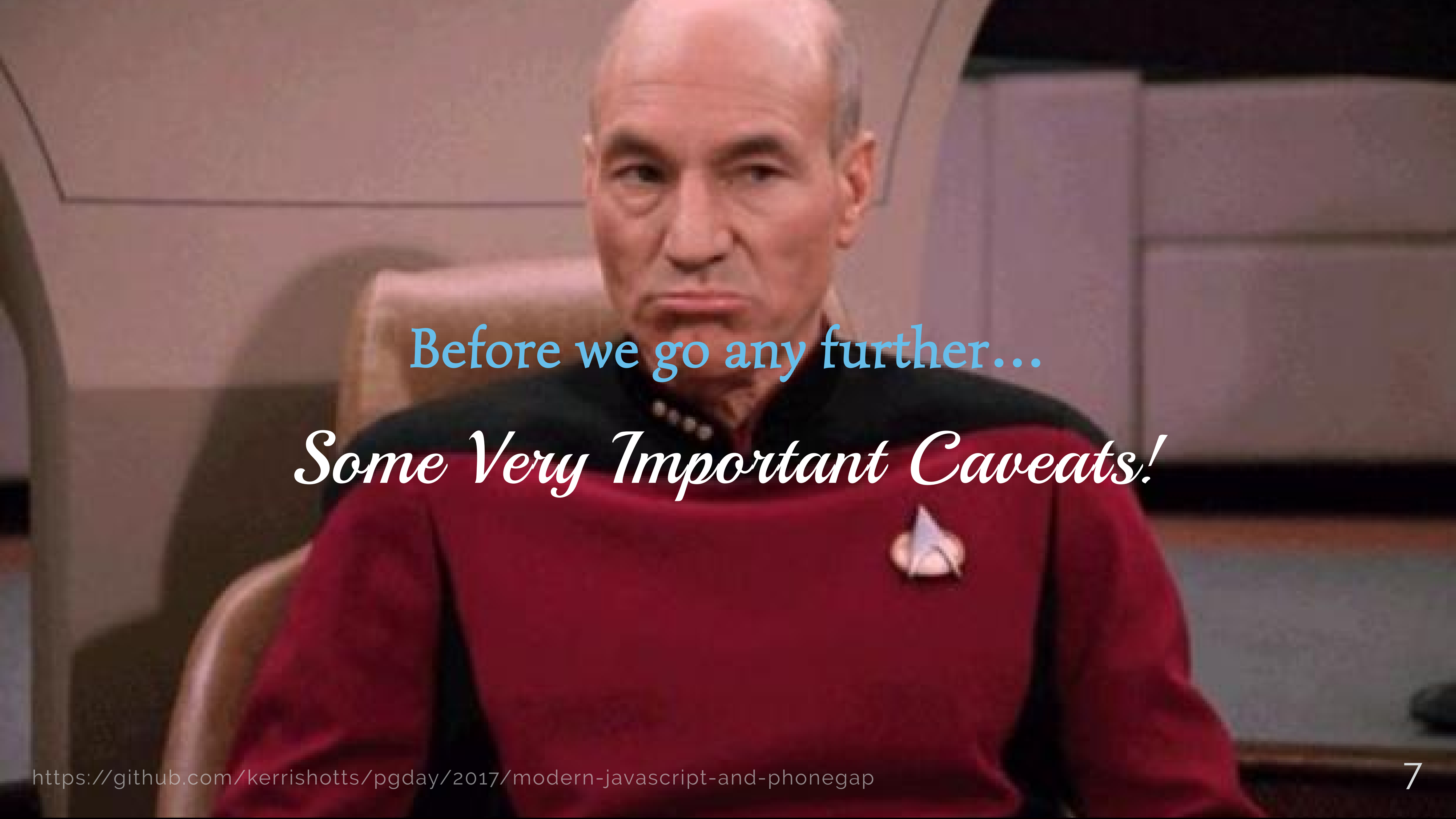
1. http://caniuse.com/#feat=es5

Things have changed a lot since then...

ES2015 and beyond

2015[1]	Block-scoped `let` & `const`	Destructuring and named parms
	Default parameters	Rest and Spread operator (`...`)
	`for...of` loops and Iterators	Arrow functions (`=>`)
	Template strings & interpolation	Improved literals (object, `0b10`)
	Generators (`*` / `yield`)	Symbols, Maps & Sets, Promises
	`class` syntactic sugar & `super`	Modules (`import` , `export`)
2016[2]	Exponent (`**`)	`Array.prototype.incudes()`
2017[3]	`async / await`	String padding 😉
	Shared memory	Atomics

1. https://github.com/lukehoban/es6features#readme; the list here is not a complete representation of *all* features
2. http://www.2ality.com/2016/01/ecmascript-2016.html
3. http://www.2ality.com/2016/02/ecmascript-2017.html



Before we go any further…

Some Very Important Caveats!

Caveats

- *NOT* a performance optimization
- Typically requires a build step
- Debugging can be interesting
- Some of the syntax is a little *sharp* — use with care

Performance Change from ES5	Chrome 55	Edge 15	Safari 10
Arrow functions	N/C	+1.2x	N/C
let compound	-1.6x	N/C	N/C
Classes	N/C	-1.5x	N/C
super	-4x	-1.7x	-15x
Destructuring	-16x	-53x	-23x
for … of array	-17x	-7x	-1.3x
for … of object	-1.8x	-4x	-2.3x
Map & Set	-4x	-23x	-8x
rest	+1.3x	+14x	-33x
spread	-22x	-1.7x	-5x
Template string	-1.2x	+1.4x	-18x

Source: https://kpdecker.github.io/six-speed/ (2017/01/04) | N/C: "no change"

So, why bother?

- Don’t let those numbers scare you!
  - Micro-benchmarks don’t always reflect the real world
  - Performance is steadily improving
- Frameworks are becoming increasingly dependant on ES2015
- Arrow functions, template strings, async/await
- More expressive & less boilerplate

Webviews & Performance

- `WKWebView` (iOS) single-core performance is impressive
  - iPad Pro 12.9″ can rival a MacBook Pro (Late 2014, 2.2GHz i7)
  - iPhone 6s is about half that; iPad Mini 4 is 2.5x slower
- Android Web View / Chrome is “meh”
  - OnePlus One is about 10%; Samsung Tab S 8.4″ about 3%.
- `UIWebView`: …

Note: Of course, this is *highly sensitive* to the ES2015+ features that you use. MacBook Pro: Late 2014, 2.2GHz i7 16GB RAM

UIWebView strikes again

Webviews & Performance (2)

- `UIWebView`: *ugh*
  - 1% on an iPad Pro 12.9″
  - No JIT 😢

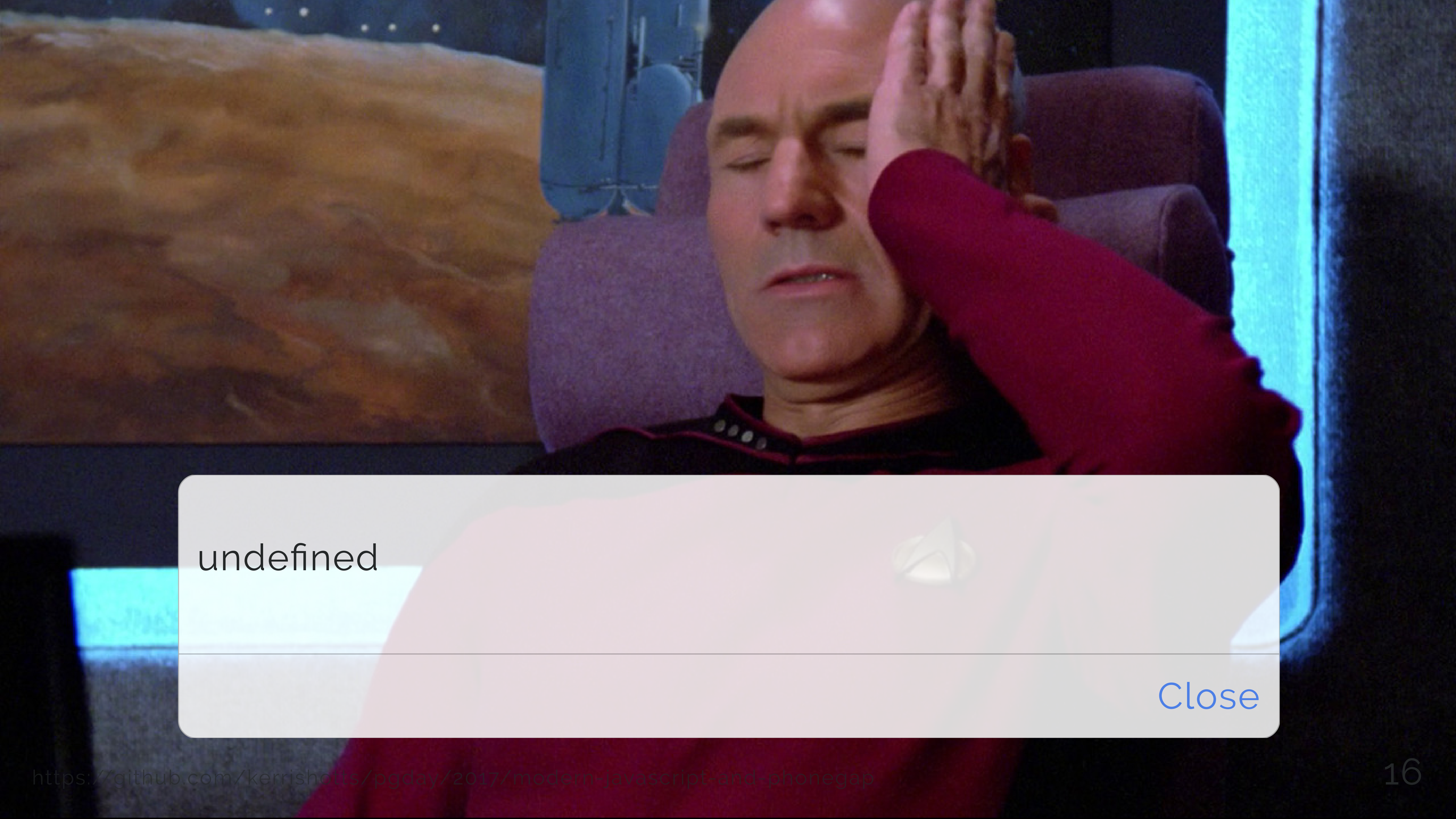
Note: Of course, this is *highly sensitive* to the ES2015+ features that you use. MacBook Pro: Late 2014, 2.2GHz i7 16GB RAM

A whirlwind tour

Dang it, *this!*

```
var app = {
  text: "Hello, PG Day Attendees!",
  sayHi: function() { alert(this.text); },
  start: function() {
    document.querySelector("#clickme")
      .addEventListener("click", this.sayHi, false);
  }
}

app.start();
```

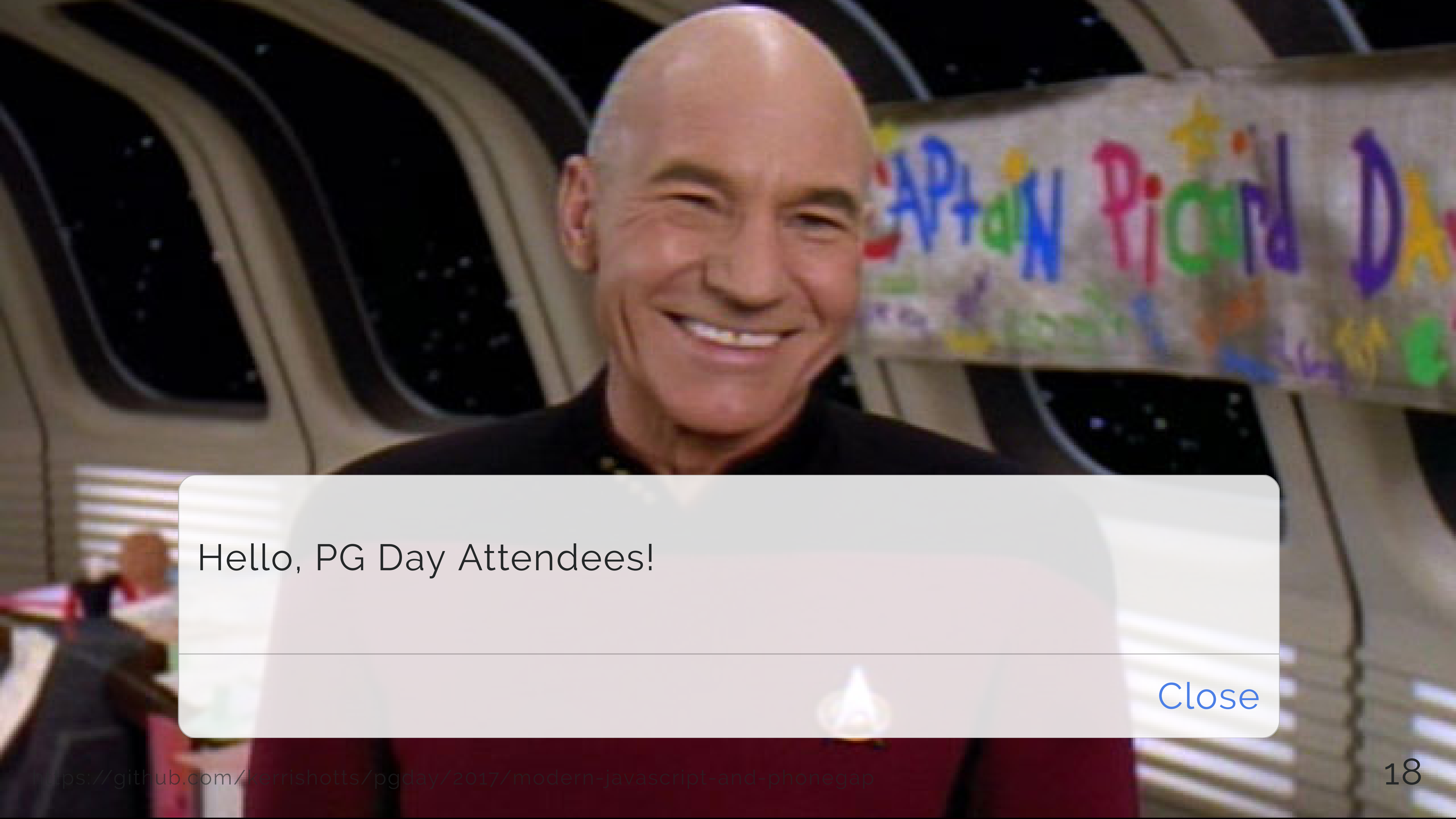


```
undefined
```

Arrow functions (=>) & Classes

```
 1  class App {
 2    constructor() { this.text = "Hello, PG Day Attendees!"; }
 3    sayHi() { alert(this.text); }
 4    start() {
 5      document.querySelector("#clickme")
 6        .addEventListener("click",() => this.sayHi(), false);
 7    }
 8  }
 9  const app = new App();
10  app.start();
```

Line 6 ES5 equivalent: `.addEventListener("click", (function() { this.sayHi(); }).bind(this), false)`



Hello, PG Day Attendees!

Close

Array-like conversion

ES5 requires `slice`:

```
var elList = document.querySelectorAll("a"),
    elArr = [].slice.call(elList, 0);
```

ES2015+ (with the standard library):

```
let elArr = Array.from(document.querySelectorAll("a"));
```

Spread/Rest is awesome (...)

Even shorter than `Array.from`:

```
let elArr = [...document.querySelectorAll("a")];
```

Easy variadic arguments:

```
function sum(start = 0, ...nums) {
  return nums.reduce((acc, val) => acc + val, start);
}
console.log(sum(1, 5, 10, 99)); /* 115 */
```

Destructuring

```
[a, b] = [b, a]  // swap!
```

“Multiple return values”:

```
function duplicate(str) {
  return {result: str + str,
          error: !str ? "no string" : null};
}
let {result, error} = someFunction("abc");
let {result:r, error:err} = someFunction("acb"); // you can rename
let {result} = someFunction("abc");              // or even ignore!
```

Named Parameters & Defaults

```
class Button {
  constructor({type = "default", text = "",
              x = 0, y = 0, w = 100, h = 44} = {}) {
    this.type = type;
    this.text = text;
    this.frame = {x, y, w, h};
    this.bounds = {x: 0, y: 0, w, h};
  }
}
let button = new Button ({type: "round", text: "Click me",
                         x: 100, y: 100});
```

Template Strings

```
let x = 4, y = 10;
console.log(`x + y => ${x} + ${y} => ${x + y}`);
```

⇒ x + y => 4 + 10 => 14

Multi-line strings (preserving ↵):

```
let template=`<ul>
    <li><span></span></li>
</ul>`;
```

Promises, promises

Hopefully already familiar to you…

```
function requestFileSystem({type = window.PERSISTENT,
                            quota = 5 * 1024 * 1024} = {}) {
  return new Promise((resolve, reject) => {
    window.requestFileSystem(type, quota, resolve, reject);
  });
}
```

But ES2017 has something better…

async / await

```
async function readFile(name) {
  const fs = await requestFileSystem({
    type: window.PERSISTENT, quota: 10 * 1024 * 1024});
  return await readFile(await fs.getFile(name));
}

async function start() {
  try {
    const data = await readFile("poem.txt");
    readPoemAloud(data);
  } catch (err) { alert (err); }
}
```

Modules

Static Analysis, FTW!

📄 math.js:

```
export function add(a, b) {
    return a+b;
}
```

📄 index.js:

```
import {add} from "math.js";
console.log(add(4, 3)); /* 7 */
```

PhoneGap Examples

Geolocation with ES2017

```
function getPos(opts) {
  return new Promise((resolve, reject) => {
    navigator.geolocation.getCurrentPosition(resolve, reject, opts);
  });
}
async function start() {
  try {
    const {timestamp, coords:{latitude, longitude}} = await getPos();
    console.log(`At ${latitude}, ${longitude} on ${timestamp}`);
  } catch(err) {
    console.log(`Error ${err.code}: ${err.message}`);
  }
}
```

File Transfer with ES2017

```
function uploadFile({source, target, options} = {}) {
  return new Promise((resolve, reject) => (new FileTransfer()).
    upload(url, to, resolve, reject, options));
}
async function start() {
  try {
   const {responseCode, response, bytesSent} = uploadFile({
      url: "cdvfile://localhost/persistent/test.txt",
      to:  "http://www.example.com/upload.php",
      options: { mimeType: "text/plain",
                 fileKey:  "file", fileName: "test" }});
  } catch (err) { /* do something with the error */ }
}
```

Native support is a moving target

OS	ES2015	ES2016	ES2017
Android (Chrome)	97% (51+)	100% (55+)	53% (56+)
Edge 15	100%	100%	39%
Edge 14	93%	-	-
iOS 11*	100%	100%	98%
iOS 10	100%	61%	42%
iOS 9	54%	-	-

* Based on current status in Safari Technological Preview 11
Note: Some of the tests are based on existence, not completeness. **Sources**: ES2015, ES2016, ES2017

But, I want it everywhere!

ES2015+ ⇒ ES5 😄 💃

or, *The Rise of the Transpilers*

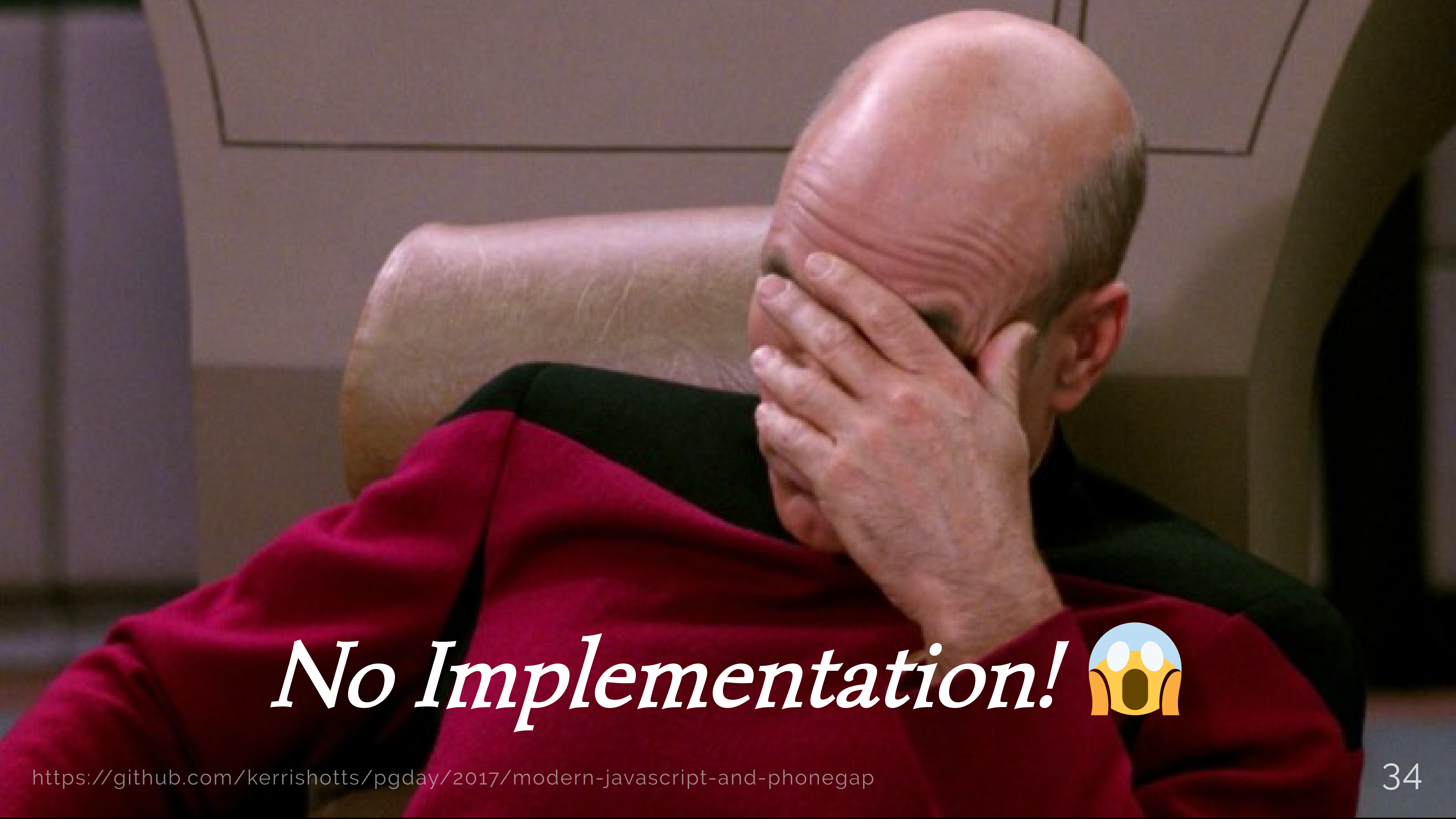
Common Transpilers

These can all transpile ES2015* (feature support may vary)

- Babel (née es6to5)
- TypeScript
- Bublé **
- Traceur

* **Note:** Not every ES2015+ feature can be transpiled effectively (if at all), such as proxies, shared memory, atomics, built-in subclassing, and tail call elimination. Also, most transpilers need core-js to polyfill the standard library.
** Doesn't attempt to transform non-performant or non-trivial ES6 features; *also very young*

Remember module syntax?



No Implementation! 😱

But we can fix that…

Module support using Bundling 🛍️

Dependency management & `import` / `export` (and CommonJS, AMD, etc.) support

- Webpack
- JSPM
- Browserify

PhoneGap Integration

- Manual
  - Just run each tool’s CLI… *every time…*
  - Error prone — you might forget!
- Automatic
  - `gulp` / `grunt` task runners
  - `npm run` scripts
  - Plugin / Project hooks

Setting up (npm run scripts)

- Install Webpack & Transpiler
- Configure Webpack & Transpiler
- Add build scripts to `package.json`

Install Webpack & Transpiler

```
[user@dev] $ | npm install --save-dev webpack
```

Typescript:

```
[user@dev] $ | npm install --save-dev ts-loader typescript core-js
```

Babel:

```
[user@dev] $ | npm install --save-dev babel-loader babel-core babel-polyfill \
             |   babel-preset-es2015 babel-preset-es2016 babel-preset-es2017 \
             |   babel-plugin-transform-runtime
```

Note: `core-js` is a standard library polyfill; depending on your feature use and targets you may not need it.

Configure TypeScript

```
// tsconfig.json
{
  "compilerOptions": {
    "allowJs": true,
    "target": "es5",          // es2015, es5, es3
    "module": "es2015",       // required for tree shaking
    "lib": ["es6", ...]       // Features you're using*
    "inlineSourceMap": true
  },
  "include": ["www(.src)/(es|ts)/**/*"] // adjust as appropriate
}
```

* Don't forget to import `core-js` in your `index.?s` if targeting older runtimes.

Configure Babel

Create `.babelrc`:

```
{
  "presets": [
    ["es2015", {
      "loose": true,   // best performance
      "modules": false // required for tree shaking
    }], "es2016", "es2017"
  ], "plugins": ["transform-runtime"] // reduces repetition
}
```

* Don't forget to import `babel-polyfill` in your `index.js` if targeting older runtimes.

Configure Webpack

```
// Create `webpack.config.js`:
module.exports = {
  devtool: "inline-source-map",
  context: path.resolve(__dirname, "www.src"), // if sibling, use   __dirname, "www"
  entry: "./" + path.join("(e|t)s", "index.(j|t)s"), // will fail without ./!
  output: { filename: "bundle.js",
            path: path.resolve(__dirname, "www", "js") },
  module: { loaders: [{
                test: /\.(ts|js|jsx)$/,          // remove ts for babel
                loader: 'ts-loader',             // or babel-loader
                exclude: /node_modules/,
                options: { entryFileIsJs: true } // only for js with typescript
            }]
  }
}
```

Add run script to package.json

```
"scripts": {
  "build:ios": "webpack && cordova build ios"
}
```

```
[user@dev] $ npm run build:ios
```

Magic!

cordova-plugin-webpack-transpiler can do this on `prepare`.

```
[user@dev] $  cordova plugin add --save \
                cordova-plugin-webpack-transpiler \
                --variable CONFIG=typescript|babel
```

Templates work too:

- Typescript: cordova-template-webpack-ts-scss
- Babel: cordova-template-webpack-babel-scss

What about tests?

… and code coverage?

… and linting?

Tests

```
[user@dev] $ npm install --save-dev mocha chai
[user@dev] $ npm install --save-dev ts-node         # for TypeScript
[user@dev] $ npm install --save-dev babel-register # for Babel
```

Add `test` to `package.json:scripts`*

```
"test": "mocha" // TypeScript (need ./test/_bootstrap.js)
"test": "mocha --compilers js:babel-register"   // Babel
```

Then `npm test`

* Assumes tests are in `./test`
_bootstrap.js: `require("ts-node").register();`

Code coverage (Babel)

`npm install --save-dev instanbul`, then in `.babelrc`:

```
{
  "presets": ["es2015", ...],
  "plugins": ["transform-es2015-modules-commonjs", ...]
  "env": {
    "test": {
      "plugins": ["istanbul"]
    }
  }
}
```

Code coverage (Babel, 2)

`npm install --save-dev cross-env nyc`, then:

```
// package.json
"nyc": {
  "require": ["babel-register"], "reporter": ["text", "html"],
  "sourceMap": false, "instrument": false
}
```

And create a `npm run` script:

```
"cover": "cross-env NODE_ENV=test nyc npm test"
```

Linting

`eslint` works just fine with ES2015! (`tslint` for Typescript)

```
[user@dev] $ | npm install --save-dev eslint
```

📄 `package.json`:

```
"scripts": {
    "lint": "eslint www.src test"
}
```

```
[user@dev] $ | npm run lint     # or, write a plugin /
             |                  # project-level hook! ;-)
```

Tips

Tips

- You don't have to convert overnight — a little at a time is fine
- Use `for...of` instead of `for...in` & `hasOwnProperty()`
- Don't assume `=>` functions are drop-in replacements
  - Careful using arrow functions with `describe` & `it` in your tests
- `var` hasn't gone away
- Try to declare `let` / `const` at the top of each scope (for Chrome's benefit)

Tips (2)

- Use `var` instead of `let` where performance is critical (e.g., tight, nested loops)
- *Do* minify & tree shake — reduces file size and startup time
    - Don't count on minified code as a performance optimization (results highly variable)

And we're done!

Thanks!

https://github.com/kerrishotts/pgday/2017/modern-javascript-and-phonegap

@kerrishotts

This slide intentionally left blank