ECMAScript 2015 and beyond

The Future of JavaScript is Now!

Tom Van Cutsem

JS.BE Meetup


@tvcutsem

Talk Outline

- Part I: 20 years of JavaScript (or, the long road to ECMAScript 6)
- Part II: a brief tour of ECMAScript 6
- Part III: ECMAScript 6 implementation progress
- Part IV: beyond ECMAScript 6
- Wrap-up

Part I
20 years of JavaScript (or, the long road to ES6)

JavaScript's origins

- Invented by Brendan Eich in 1995, to support client-side scripting in Netscape Navigator
- First called *LiveScript*, then *JavaScript*, then standardized as *ECMAScript*
- Microsoft "copied" JavaScript in IE JScript, "warts and all"

Brendan Eich,
Inventor of JavaScript

The world's most misunderstood language

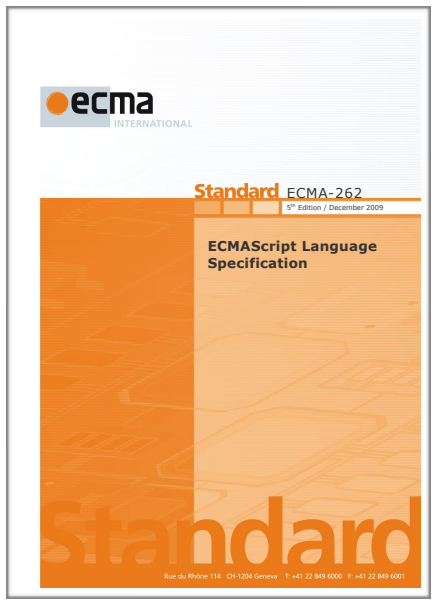
Douglas Crockford,
Inventor of JSON

See also: "JavaScript: The World's Most Misunderstood Programming Language" by Doug Crockford at http://www.crockford.com/javascript/javascript.html

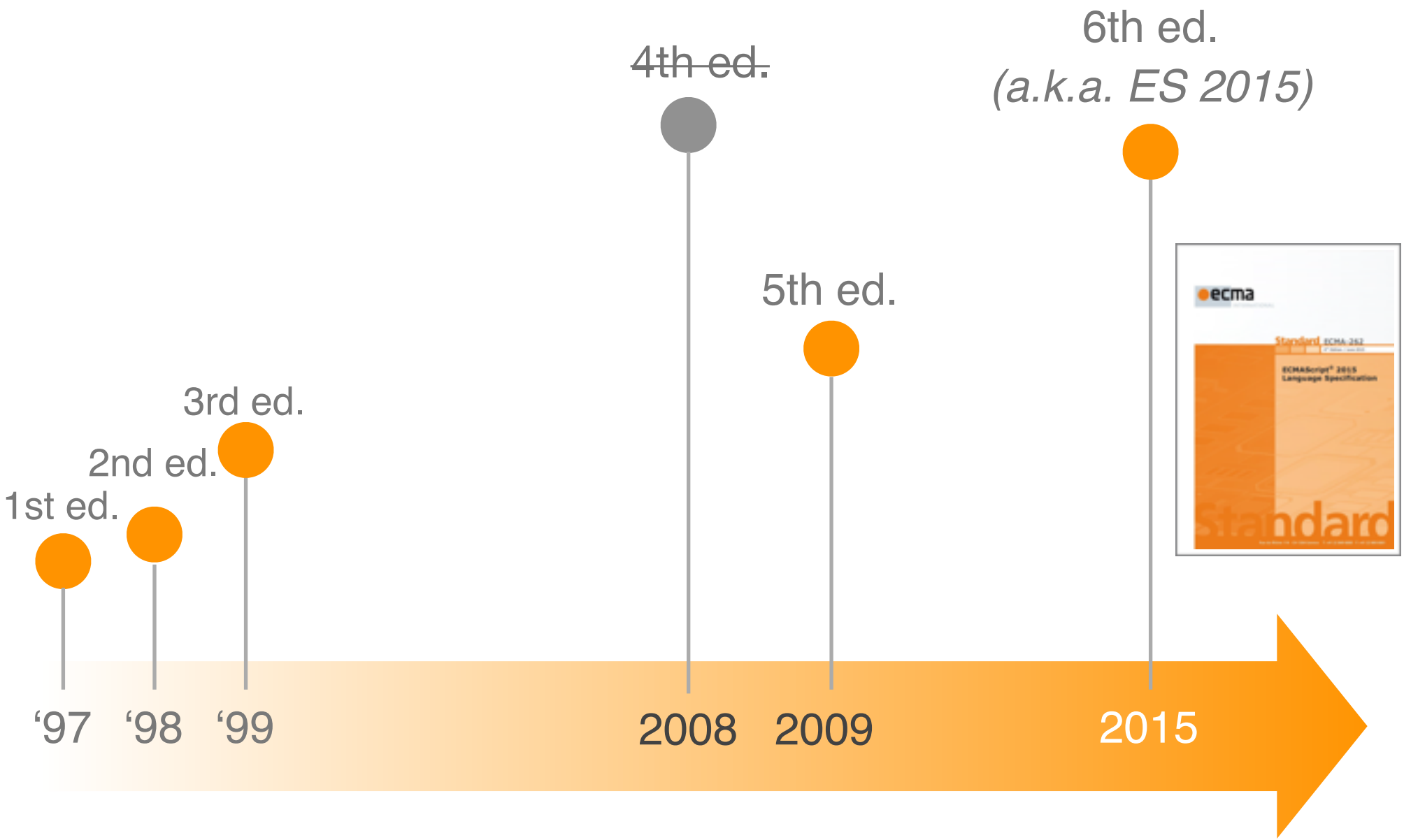
TC39: the JavaScript “standardisation committee”

- Representatives from major Internet companies, browser vendors, web organisations, popular JS libraries and academia. Meets bi-monthly.
- Maintains the ECMA-262 specification.
- The spec is a handbook mainly intended for language implementors.

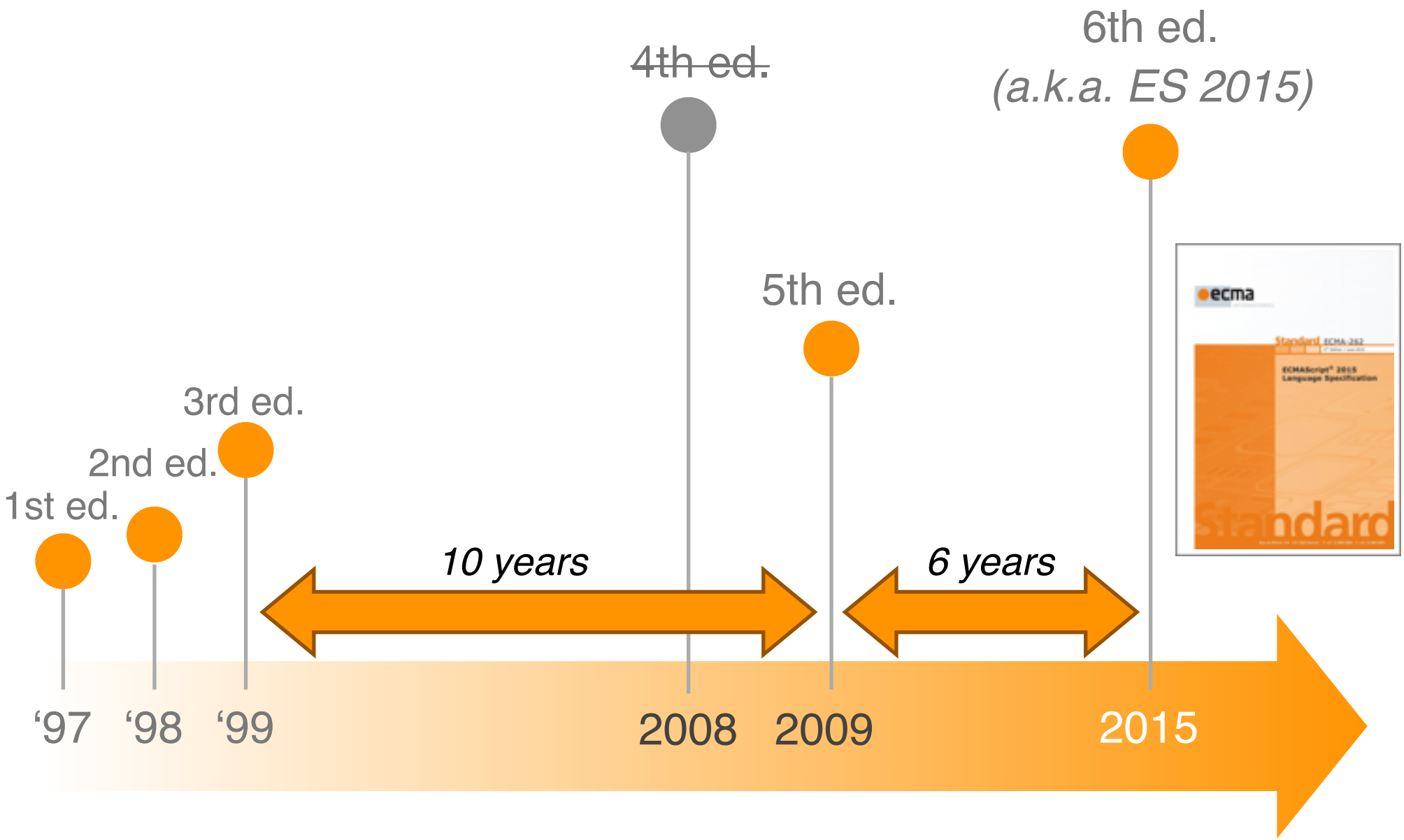
Allen Wirfs-Brock,
ECMA-262 technical editor (5th & 6th ed.)



A brief history of the ECMAScript spec



A brief history of the ECMAScript spec

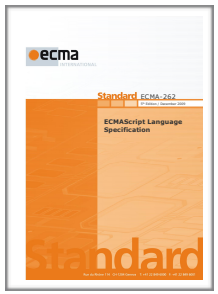


Part II
A brief tour of ECMAScript 6

ECMAScript 6

- Major update: many new features (too many to list here)
- Point-in-case:

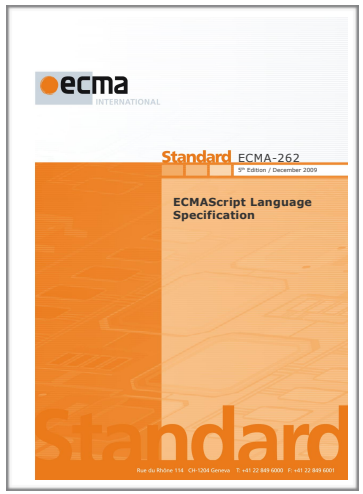
ES5.1



258 pages

x 2.2

ES6



566 pages

ECMAScript 6: shortlist

- Big Ticket items: Classes and Modules
- Control flow Goodness:
  - Iterators
  - Generators
  - Promises

ECMAScript 6: shortlist

- Big Ticket items: Classes and Modules
- **Control flow Goodness:**
  - **Iterators**
  - Generators
  - Promises

ECMAScript 6 Iterators

```
interface Iterator<T> {
  next() : IteratorResult<T>;
}

interface IteratorResult<T> {
  value : T;
  done  : bool;
}
```

ECMAScript 6 Iterators

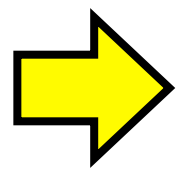
```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var temp = pre;
      pre = cur;
      cur = cur + temp;
      return { done: false, value: cur }
    }
  }
}

// generates 1, 1, 2, 3, 5, 8, 13, 21, …
```

ECMAScript 6 Iterators

```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var temp = pre;
      pre = cur;
      cur = cur + temp;
      return { done: false, value: cur }
    }
  }
}
```

ES5

```
var iter = fibonacci();
var nxt = iter.next();
while (!nxt.done) {
  var n = nxt.value;
  if (n > 100)
    break;
  print(n);
  nxt = iter.next();
}
```



ES6

```
for (var n of fibonacci()) {
  if (n > 100)
    break;
  print(n);
}
```

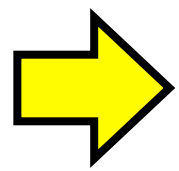
// generates 1, 1, 2, 3, 5, 8, 13, 21, …

ECMAScript 6 Iterators

```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var temp = pre;
      pre = cur;
      cur = cur + temp;
      return { done: false, value: cur }
    }
  }
}
```

ES5

```
var iter = fibonacci();
var nxt = iter.next();
while (!nxt.done) {
  var n = nxt.value;
  if (n > 100)
    break;
  print(n);
  nxt = iter.next();
}
```



ES6

```
for (var n of fibonacci()) {
  if (n > 100)
    break;
  print(n);
}
```

// generates 1, 1, 2, 3, 5, 8, 13, 21, …

ECMAScript 6: shortlist

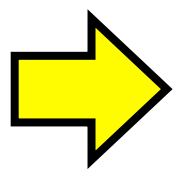
- Big Ticket items: Classes and Modules
- **Control flow Goodness:**
  - Iterators
  - **Generators**
  - Promises

ECMAScript 6 Generators

- A generator function implicitly creates and returns an iterator

ES5

```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var tmp = pre;
      pre = cur;
      cur = cur + tmp;
      return { done: false, value: cur }
    }
  }
}
```



ES6

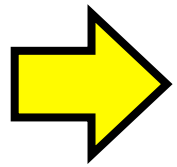
```
function* fibonacci() {
  var pre = 0, cur = 1;
  for (;;) {
    var tmp = pre;
    pre = cur;
    cur = cur + tmp;
    yield cur;
  }
}
```

ECMAScript 6 Generators

- A generator function implicitly creates and returns an iterator

ES5

```
function fibonacci() {
  var pre = 0, cur = 1;
  return {
    next: function() {
      var tmp = pre;
      pre = cur;
      cur = cur + tmp;
      return { done: false, value: cur }
    }
  }
}
```



ES6

```
function* fibonacci() {
  var pre = 0, cur = 1;
  for (;;) {
    var tmp = pre;
    pre = cur;
    cur = cur + tmp;
    yield cur;
  }
}
```

ECMAScript 6: shortlist

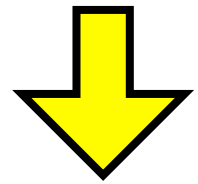
- Big Ticket items: Classes and Modules
- **Control flow Goodness:**
  - Iterators
  - Generators
  - **Promises**

ECMAScript 6 Promises

- A promise is a placeholder for a value that may only be available in the future

ES5

```
readFile("hello.txt", function (err, content) {
  if (err) {
    // handle error
  } else {
    // use content
  }
})
```



ES6

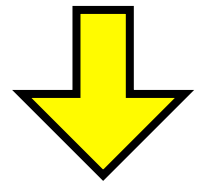
```
var pContent = readFile("hello.txt");
pContent.then(function (content) {
  // use content
}, function (err) {
  // handle error
});
```

ECMAScript 6 Promises

- A promise is a placeholder for a value that may only be available in the future

ES5

```
readFile("hello.txt", function (err, content) {
  if (err) {
    // handle error
  } else {
    // use content
  }
})
```



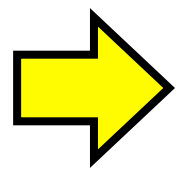
ES6

```
var pContent = readFile("hello.txt");
var p2 = pContent.then(function (content) {
  // use content
}, function (err) {
  // handle error
});
```

ECMAScript 6 Promises

- Promises can be *chained* to avoid callback hell

```
// step1(value, callback) -> undefined

step1(function (e,value1) {
    if (e) { return handleError(e); }
    step2(value1, function(e,value2) {
        if (e) { return handleError(e); }
        step3(value2, function(e,value3) {
            if (e) { return handleError(e); }
            step4(value3, function(e,value4) {
                if (e) { return handleError(e); }
                // do something with value4
            });
        });
    });
});
```



```
// step1(value) -> Promise

step1(value)
.then(step2)
.then(step3)
.then(step4)
.then(function (value4) {
    // do something with value4
})
.catch(function (error) {
    // handle any error here
});
```

Example adapted from https://github.com/kriskowal/q

ECMAScript 6 Promises

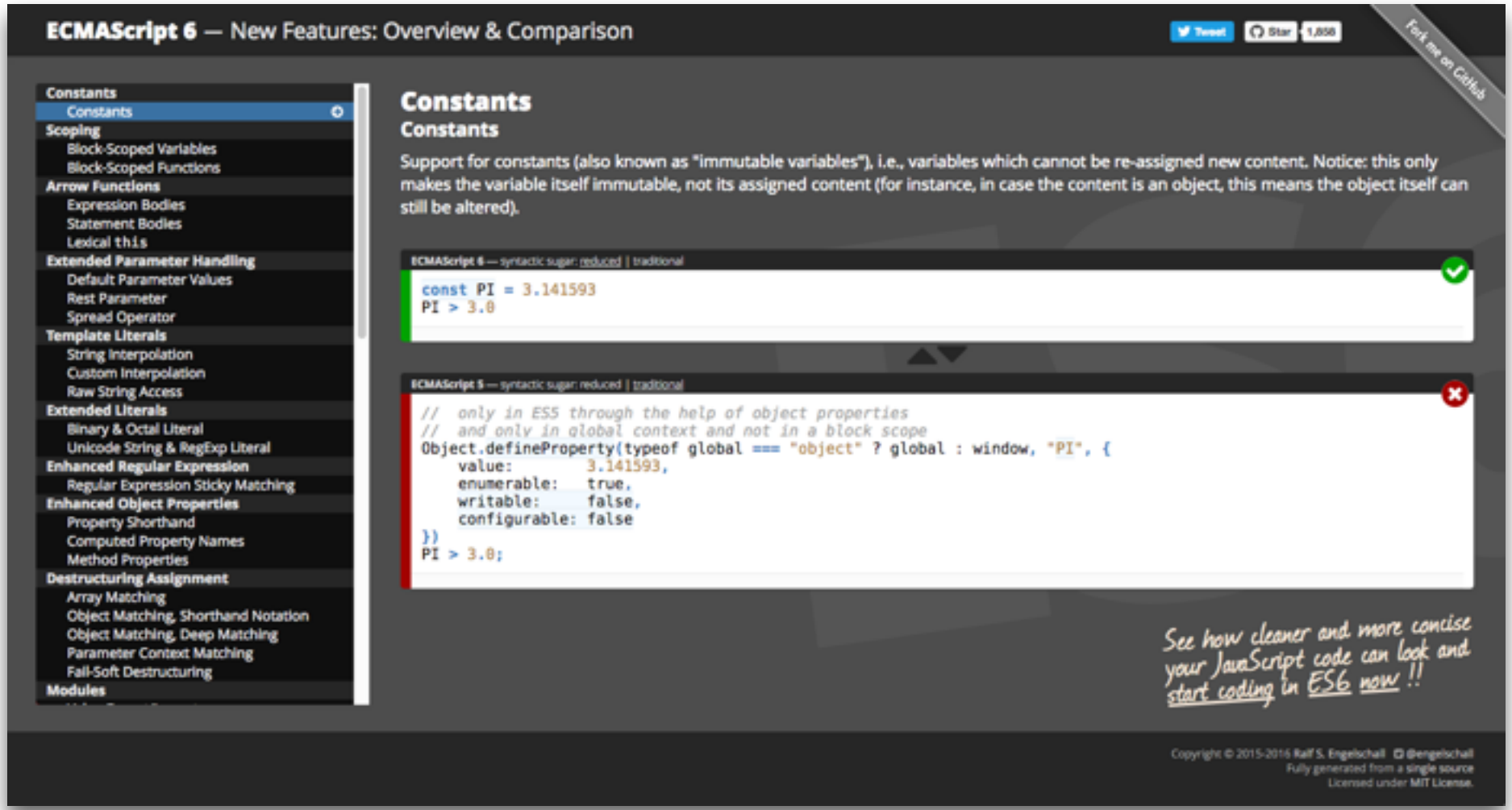
- Promises already exist as a library in ES5 (e.g. Q, Bluebird)
- Then why standardize?
  - Wide disagreement on a single Promise API. ES6 settled on an API called “Promises/A+”. See promisesaplus.com
  - Standard API allows platform APIs to use Promises as well
  - W3C’s latest DOM APIs already use promises

ECMAScript 6: shortlist

- Big Ticket items: Classes and Modules
- Control flow Goodness:
  - Iterators
  - Generators
  - Promises

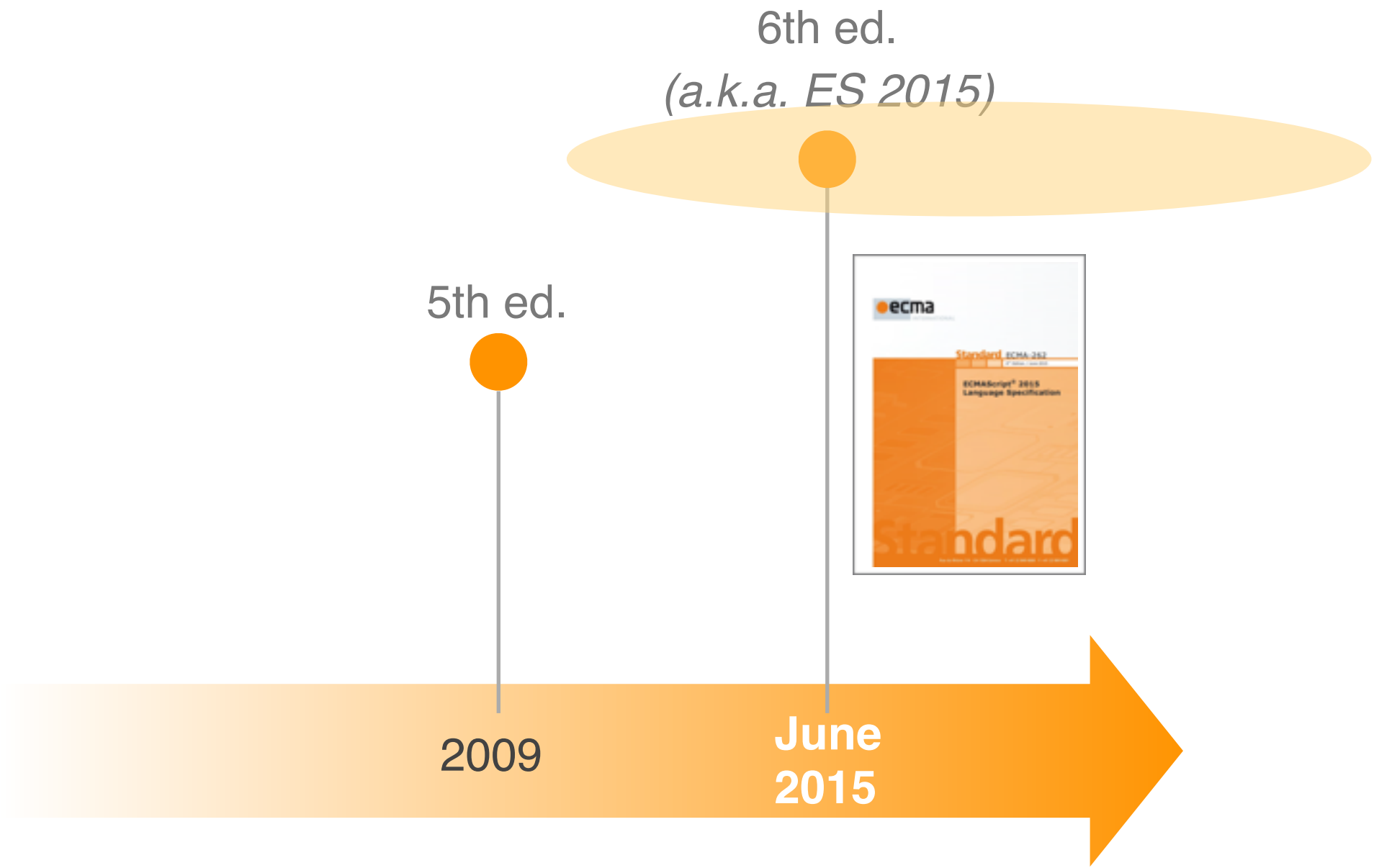
ECMAScript 6: longlist

http://es6-features.org/



Part III
ECMAScript 6: implementation progress

ECMAScript 6: timeline

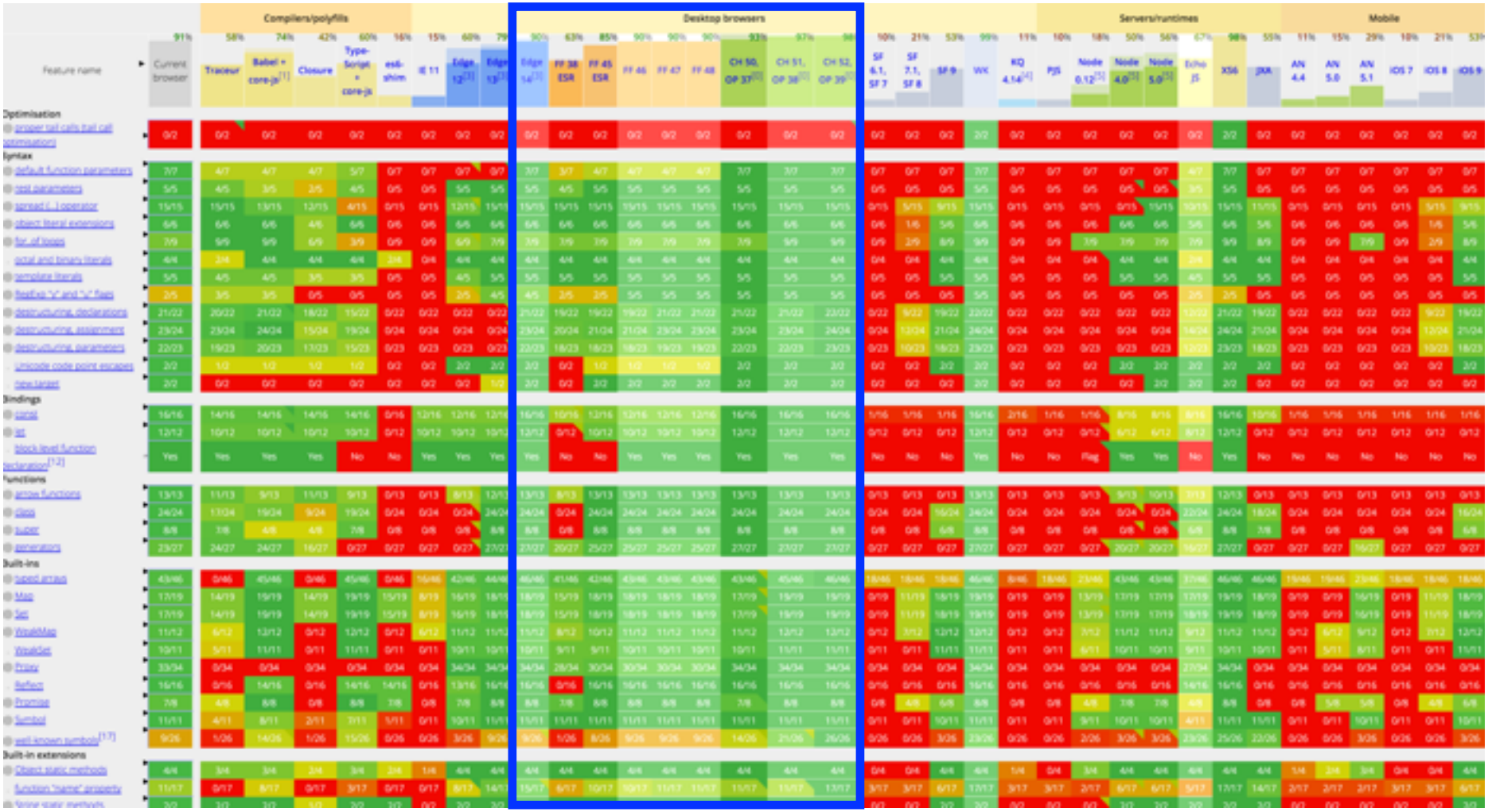


ECMAScript 6 support (april 2016)

(Source: Juriy Zaytsev (kangax)
http://kangax.github.io/es5-compat-table/es6)

COMPAT ES

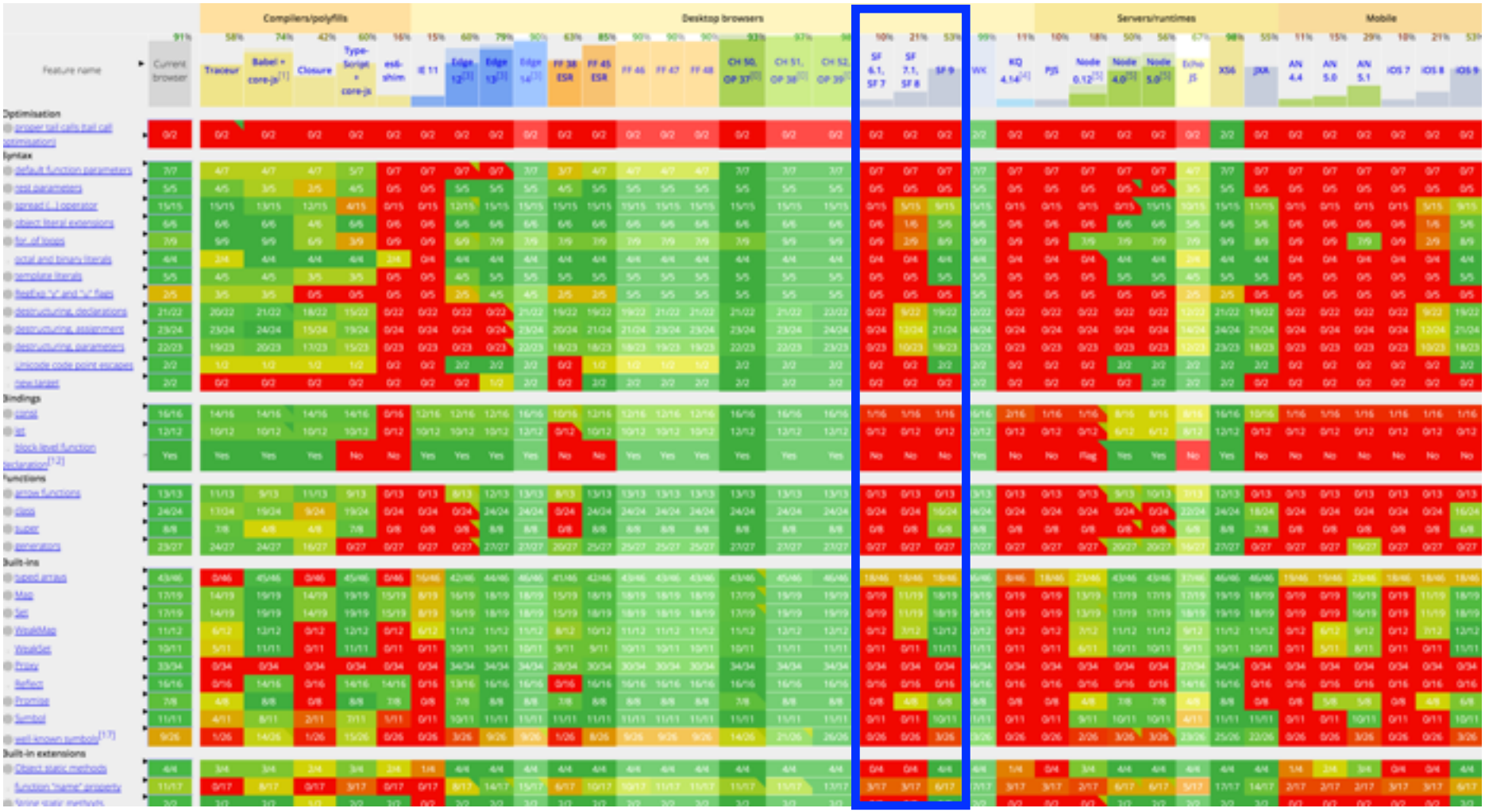
ECMAScript 6 support (april 2016)



Desktop browsers support nearly all of ES6…

COMPAT ES

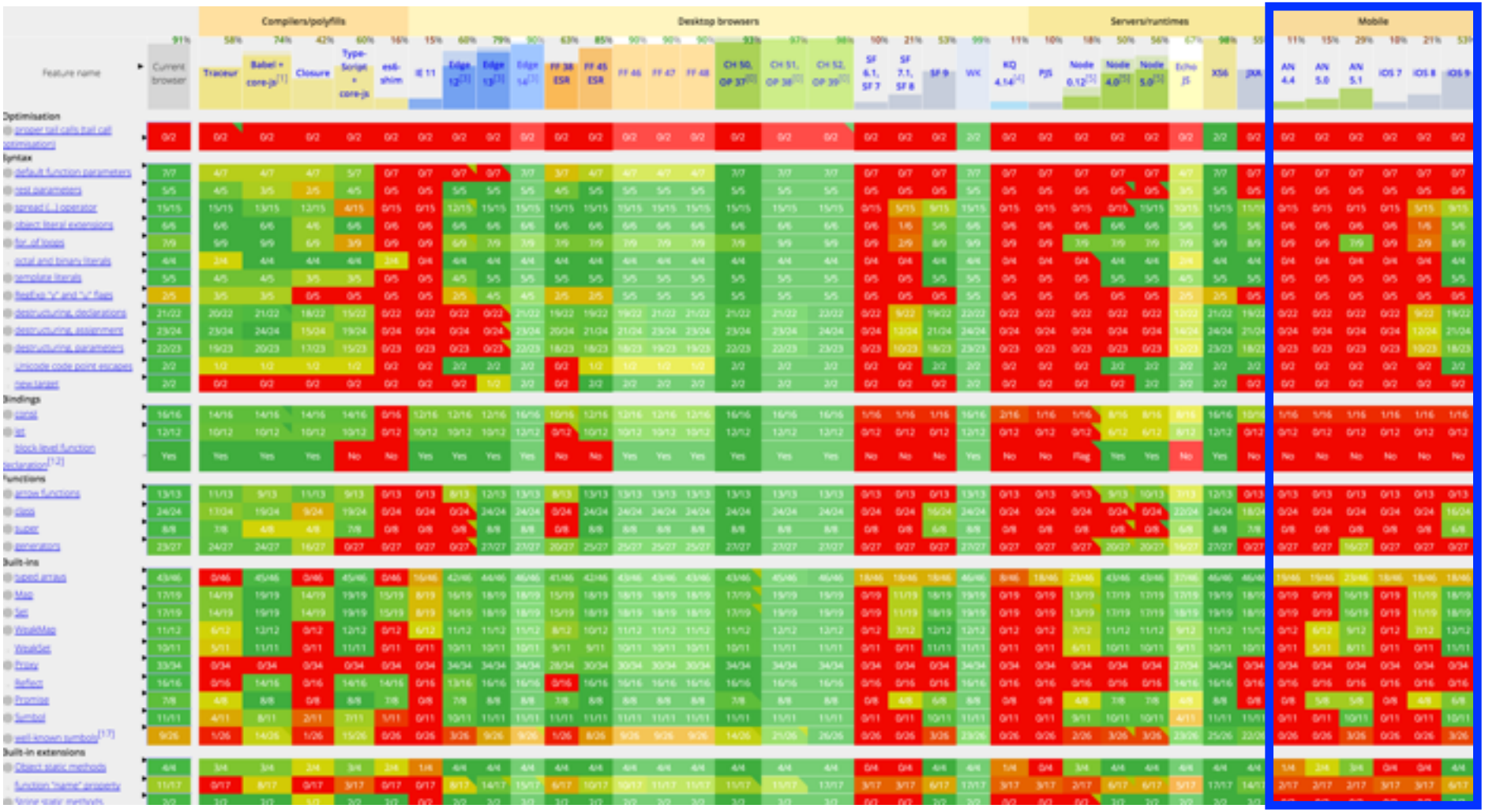
ECMAScript 6 support (april 2016)



… except for Safari (v9 at 53%)

COMPAT ES

ECMAScript 6 support (april 2016)



Mobile browsers are lagging behind…

COMPAT ES

ECMAScript 6 compilers

- Compile ECMAScript 6 to ECMAScript 5
- **Babel**: focus on producing readable (as-if hand-written) ES5 code. Supports JSX as well.

- Microsoft **TypeScript**: technically not ES6 but roughly a superset of ES6. Bonus: type inference and optional static typing.

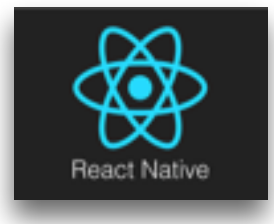
ECMAScript 6: server, desktop and mobile

- V8 5.0 (March 2016) implements over 93% of ES6

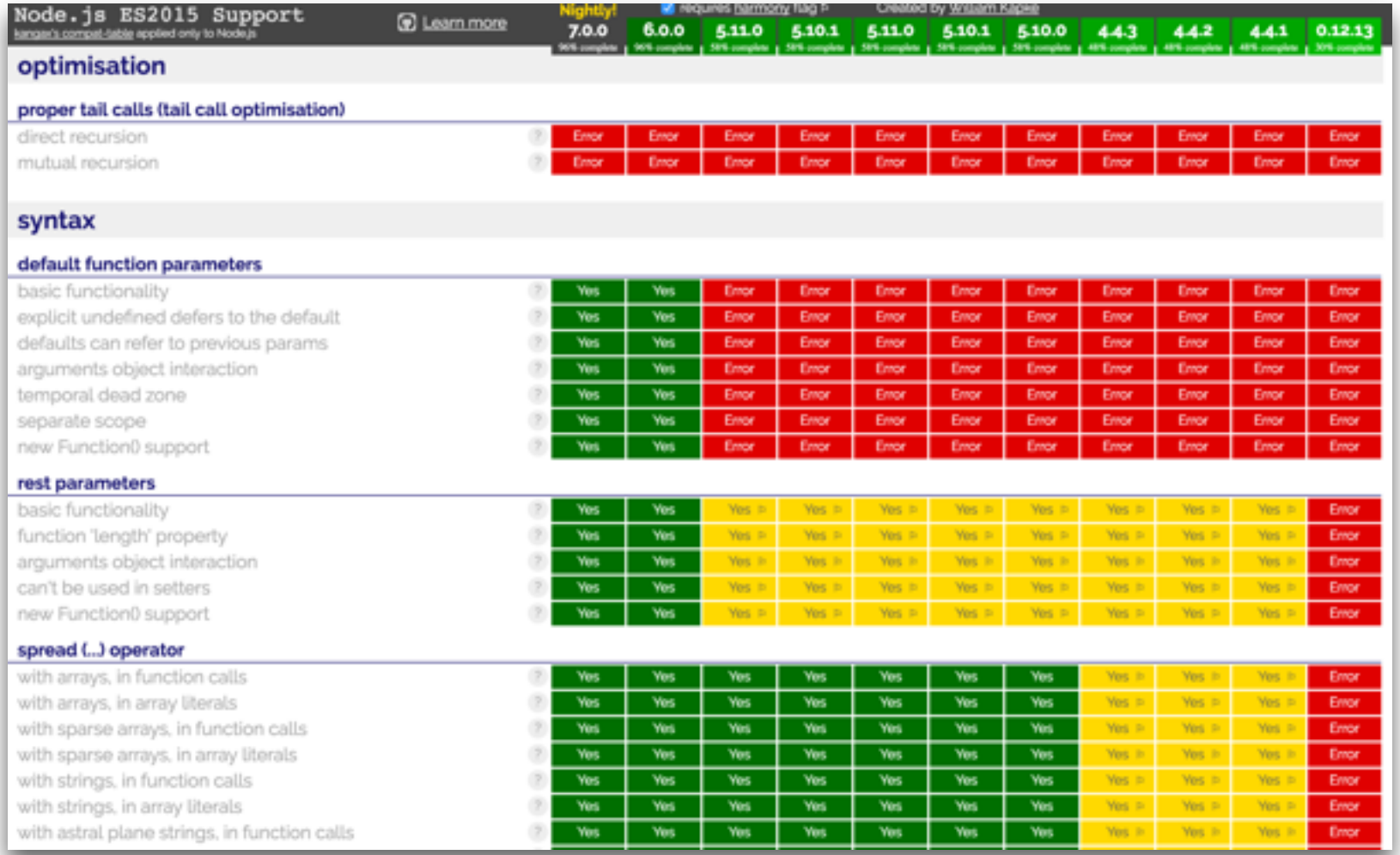
- Server: **node.js** v6.0.0 released 26th of April uses V8 5.0 and so is ES6-ready!

- Desktop: **NW.js** and **Electron** use node v5.x

- Mobile: **React-native** has Babel integration by default [on top of Safari's JSC]
  Apache **Cordova** runs on web views supported by mobile browsers, use Babel.



ECMAScript 6: server, desktop and mobile



Node.js ES2015 Support

kangax's compat-table applied only to Node.js

Learn more

Nightly! 7.0.0 (96% complete) | 6.0.0 (96% complete) | 5.11.0 (58% complete) | 5.10.1 (58% complete) | 5.11.0 (58% complete) | 5.10.1 (58% complete) | 5.10.0 (58% complete) | 4.4.3 (48% complete) | 4.4.2 (48% complete) | 4.4.1 (48% complete) | 0.12.13 (30% complete)

Requires harmony flag

Created by William Kapke

optimisation

proper tail calls (tail call optimisation)

	7.0.0	6.0.0	5.11.0	5.10.1	5.11.0	5.10.1	5.10.0	4.4.3	4.4.2	4.4.1	0.12.13
direct recursion	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error
mutual recursion	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error	Error

syntax

default function parameters

	7.0.0	6.0.0	5.11.0	5.10.1	5.11.0	5.10.1	5.10.0	4.4.3	4.4.2	4.4.1	0.12.13
basic functionality	Yes	Yes	Error	Error	Error	Error	Error	Error	Error	Error	Error
explicit undefined defers to the default	Yes	Yes	Error	Error	Error	Error	Error	Error	Error	Error	Error
defaults can refer to previous params	Yes	Yes	Error	Error	Error	Error	Error	Error	Error	Error	Error
arguments object interaction	Yes	Yes	Error	Error	Error	Error	Error	Error	Error	Error	Error
temporal dead zone	Yes	Yes	Error	Error	Error	Error	Error	Error	Error	Error	Error
separate scope	Yes	Yes	Error	Error	Error	Error	Error	Error	Error	Error	Error
new Function() support	Yes	Yes	Error	Error	Error	Error	Error	Error	Error	Error	Error

rest parameters

	7.0.0	6.0.0	5.11.0	5.10.1	5.11.0	5.10.1	5.10.0	4.4.3	4.4.2	4.4.1	0.12.13
basic functionality	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
function 'length' property	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
arguments object interaction	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
can't be used in setters	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
new Function() support	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error

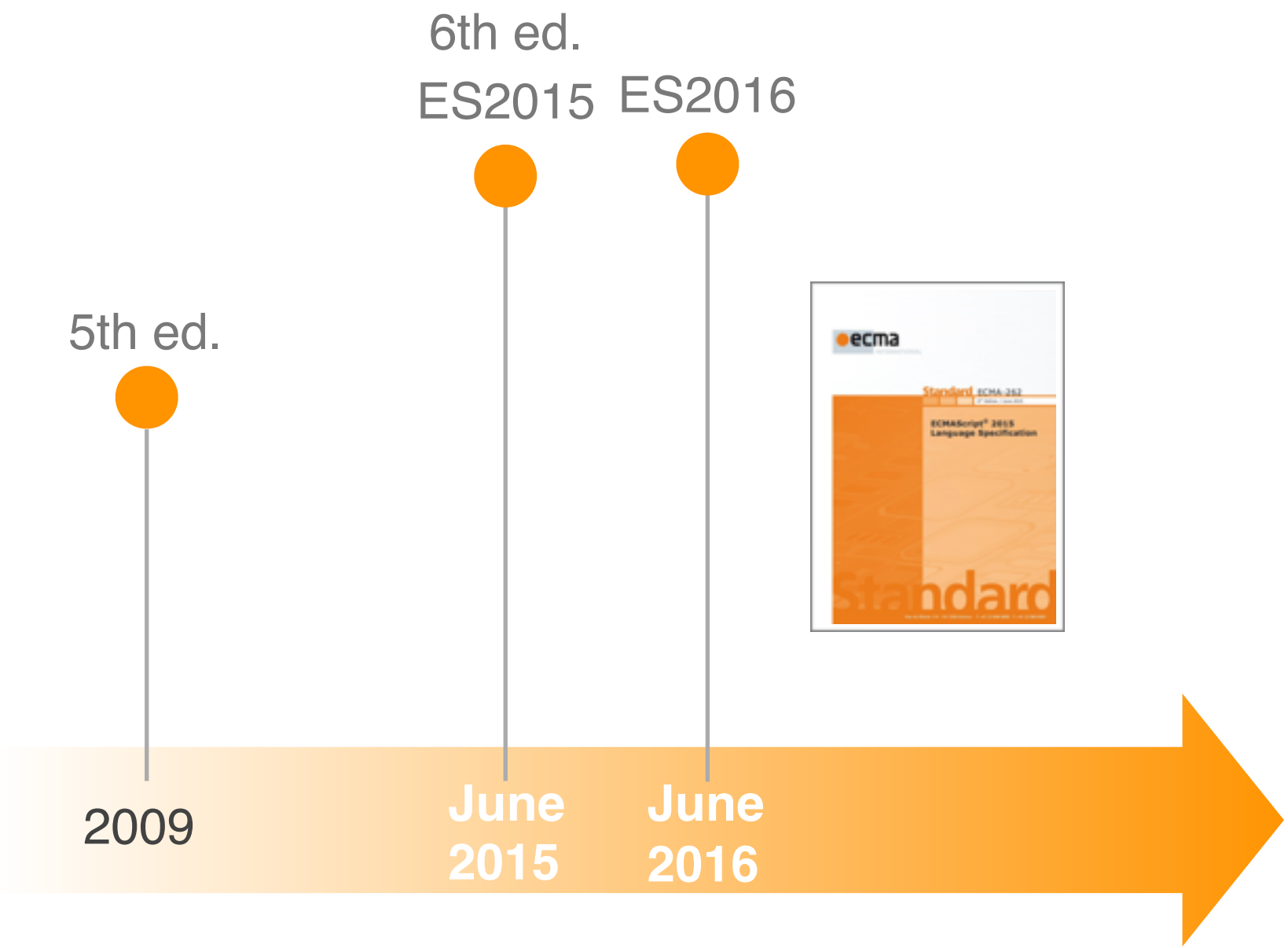
spread (...) operator

	7.0.0	6.0.0	5.11.0	5.10.1	5.11.0	5.10.1	5.10.0	4.4.3	4.4.2	4.4.1	0.12.13
with arrays, in function calls	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
with arrays, in array literals	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
with sparse arrays, in function calls	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
with sparse arrays, in array literals	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
with strings, in function calls	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
with strings, in array literals	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error
with astral plane strings, in function calls	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Error

http://node.green

Part IV
Beyond ECMAScript 6

Beyond ECMAScript 6: timeline



ES2016/2017 Completed Features

- All “stage 4” proposals thus far, mostly new standard methods

Proposal	Champion(s)	TC39 meeting notes
Array.prototype.includes	Domenic Denicola, Rick Waldron	November 2015
Exponentiation Operator	Rick Waldron	January 2016
Object.values/Object.entries	Jordan Harband	March 2016
String padding	Jordan Harband & Rick Waldron	May 2016
Object.getOwnPropertyDescriptors	Jordan Harband & Andrea Giammarchi	May 2016

ES2016 Completed Features

ES2016 feature

- Few fancy new features included in the spec in the last year
- Exponentiation operator: `x ** y`
- Array.prototype.includes: `[1,2,3].includes(1) === true`
- Bug fixes and minor details

ES2017 and beyond: proposals on the table

- Draft and Candidate proposals:

🚀	Proposal	Champion	Stage
	Object.values/Object.entries	Jordan Harband	4
	SIMD.JS - SIMD APIs + polyfill	John McCutchan, Peter Jensen, Dan Gohman, Daniel Ehrenberg	3
	Async Functions	Brian Terlson	3
	String padding	Jordan Harband & Rick Waldron	3
	Trailing commas in function parameter lists and calls	Jeff Morrison	3
	Object.getOwnPropertyDescriptors	Jordan Harband & Andrea Giammarchi	3
	`Function.prototype.toString` revision	Michael Ficarra	3
	Asynchronous Iterators	Kevin Smith	2
	function.sent metaproperty	Allen Wirfs-Brock	2
	Rest/Spread Properties	Sebastian Markbage	2
	Shared memory and atomics	Lars T Hansen	2
	System.global	Jordan Harband	2

See https://github.com/tc39/ecma262 for full list

ES2017 and beyond: proposals on the table

- Draft and Candidate proposals:

	Proposal	Champion	Stage
	Object.values/Object.entries	Jordan Harband	4
	SIMD.JS - SIMD APIs + polyfill	John McCutchan, Peter Jensen, Dan Gohman, Daniel Ehrenberg	3
	Async Functions	Brian Terlson	3
	String padding	Jordan Harband & Rick Waldron	3
	Trailing commas in function parameter lists and calls	Jeff Morrison	3
	Object.getOwnPropertyDescriptors	Jordan Harband & Andrea Giammarchi	3
	`Function.prototype.toString` revision	Michael Ficarra	3
	Asynchronous Iterators	Kevin Smith	2
	function.sent metaproperty	Allen Wirfs-Brock	2
	Rest/Spread Properties	Sebastian Markbage	2
	Shared memory and atomics	Lars T Hansen	2
	System.global	Jordan Harband	2

See https://github.com/tc39/ecma262 for full list

ECMAScript **2017**: SIMD

- Directly express Single-instruction, Multiple-data instructions
- Apply math on entire vectors.

```
var a = SIMD.float32x4(1.0, 2.0, 3.0, 4.0);
var b = SIMD.float32x4(5.0, 6.0, 7.0, 8.0);
var c = SIMD.float32x4.add(a,b);
```

- Useful for anything that requires number crunching (image processing, signal processing, …)
- JS as compilation target. Emscripten, asm.js, WebAssembly

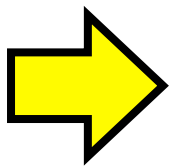
ECMAScript **2017**: async functions

ES2017 proposal

- A C# 5.0 feature that enables asynchronous programming using “direct style” control flow (i.e. no callbacks)

ES6

```
// step1(value) -> Promise

step1(value)
.then(step2)
.then(step3)
.then(step4)
.then(function (value4) {
    // do something with value4
})
.catch(function (error) {
    // handle any error here
});
```



ES2017

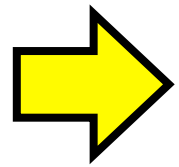
```
// step1(value) -> Promise

(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```

async functions in ECMAScript **6**

- Generators can be used as async functions, with some tinkering
- co npm library for node (4.0 or >= 0.11.x with `--harmony` flag)

ES2017

```
(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```



ES6

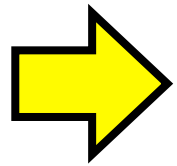
```
co(function*() {
  try {
    var value1 = yield step1();
    var value2 = yield step2(value1);
    var value3 = yield step3(value2);
    var value4 = yield step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
})
```

async functions in ECMAScript **6**

- Generators can be used as async functions, with some tinkering
- co npm library for node (4.0 or >= 0.11.x with `--harmony` flag)

ES2017

```
(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```



ES6

```
co(function*() {
  try {
    var value1 = yield step1();
    var value2 = yield step2(value1);
    var value3 = yield step3(value2);
    var value4 = yield step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
})
```

async functions in ECMAScript **5** (!)

- Babel plug-in based on Facebook Regenerator

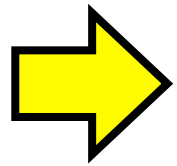
facebook.github.io/regenerator

- Also in TypeScript 1.7+

github.com/lukehoban/ecmascript-asyncawait

ES2017

```
(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```



ES5

```
(function callee$0$0() {
  var value1, value2, value3, value4;
  return regeneratorRuntime.async(function callee$0$0$(context$1$0) {
    while (1) switch (context$1$0.prev = context$1$0.next) {
      case 0:
        context$1$0.prev = 0;
        context$1$0.next = 3;
        return regeneratorRuntime.awrap(step1());

      case 3:
        value1 = context$1$0.sent;
        context$1$0.next = 6;
        return regeneratorRuntime.awrap(step2(value1));

      case 6:
        value2 = context$1$0.sent;
        context$1$0.next = 9;
        return regeneratorRuntime.awrap(step3(value2));

      case 9:
        value3 = context$1$0.sent;
        context$1$0.next = 12;
        return regeneratorRuntime.awrap(step4(value3));

      case 12:
        value4 = context$1$0.sent;
        context$1$0.next = 17;
        break;

      case 15:
        context$1$0.prev = 15;
        context$1$0.t0 = context$1$0["catch"](0);

      case 17:
      case "end":
        return context$1$0.stop();
    }
  }, null, this, [[0, 15]]);
})();
```

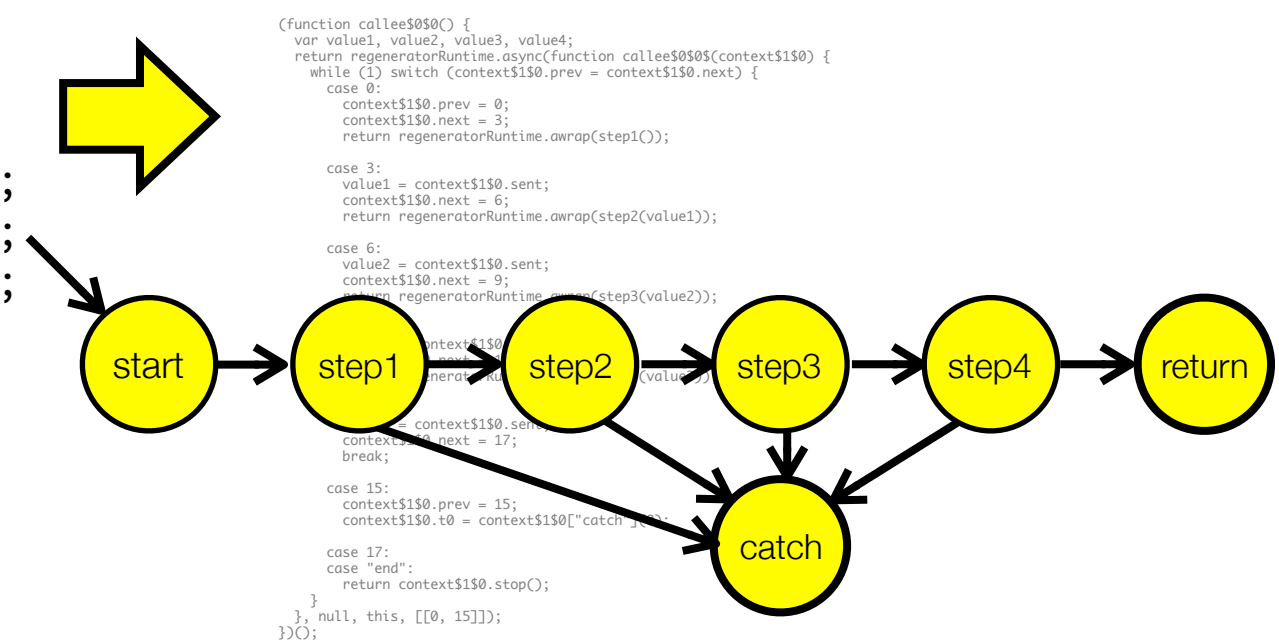
async functions in ECMAScript **5** (!)

- Async functions (and generators) can be compiled into state machines. Each intermediate state represents a “yield point”.

ES2017

```
(async function() {
  try {
    var value1 = await step1();
    var value2 = await step2(value1);
    var value3 = await step3(value2);
    var value4 = await step4(value3);
    // do something with value4
  } catch (error) {
    // handle any error here
  }
}())
```

ES5

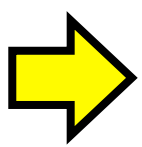


Async Iterators

ES2017 proposal

- ES6 iterator and generator protocols are synchronous…
- …but many Iterable sources are asynchronous in JS

```
interface Iterator<T> {
  next() : IteratorResult<T>;
}

interface IteratorResult<T> {
  value : T;
  done  : bool;
}
```



```
interface AsyncIterator<T> {
  next() : Promise<IteratorResult<T>>;
}
```

https://github.com/tc39/proposal-async-iteration

Async Iterators

ES2017 proposal

- Async for-of loop can be used in an async function to consume an async iterator

```
function readLines(path: string): AsyncIterator<string>;


async function printLines() {
  for await (let line of readLines(filePath)) {
    print(line);
  }
}
```

Async Generators

ES2017 proposal

- Async generators can await, and yield promises

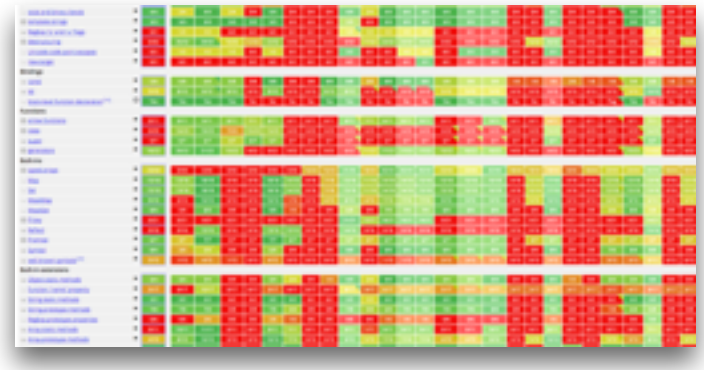
```
function readLines(path: string): AsyncIterator<string>;

async function* readLines(path) {

  let file = await fileOpen(path);

  try {
    while (!file.EOF) {
      yield file.readLine();
    }
  } finally {
    await file.close();
  }
}
```

Polyfill implemented in FB Regenerator

Wrap-up

Take-home messages

ES is alive and kicking.
The new yearly release process works!



Server and desktop browser engines have mostly transitioned, mobile is lagging behind.

Take-home messages

Use ES6 to ES5 compilers to bridge the gap

The days of Callback Hell are numbered…

(Photo credit: https://www.twilio.com/blog/2015/10/asyncawait-the-hero-javascript-deserved.html)

Thanks for listening!

ECMAScript 2015 and beyond

The Future of JavaScript is Now!

Tom Van Cutsem

JS.BE Meetup

@tvcutsem