ECMAScript 2015

The Future of JavaScript is Now!

Tom Van Cutsem

SPLASH-I 2015

Talk Outline

- Part I: JavaScript's origins, and the long road to ECMAScript 6
- Part II: a brief tour of ECMAScript 6
- Part III: using ECMAScript 6 today
- Part IV: beyond ECMAScript 6
- Wrap-up

Part I
JavaScript’s origins, and the road to ECMAScript 6

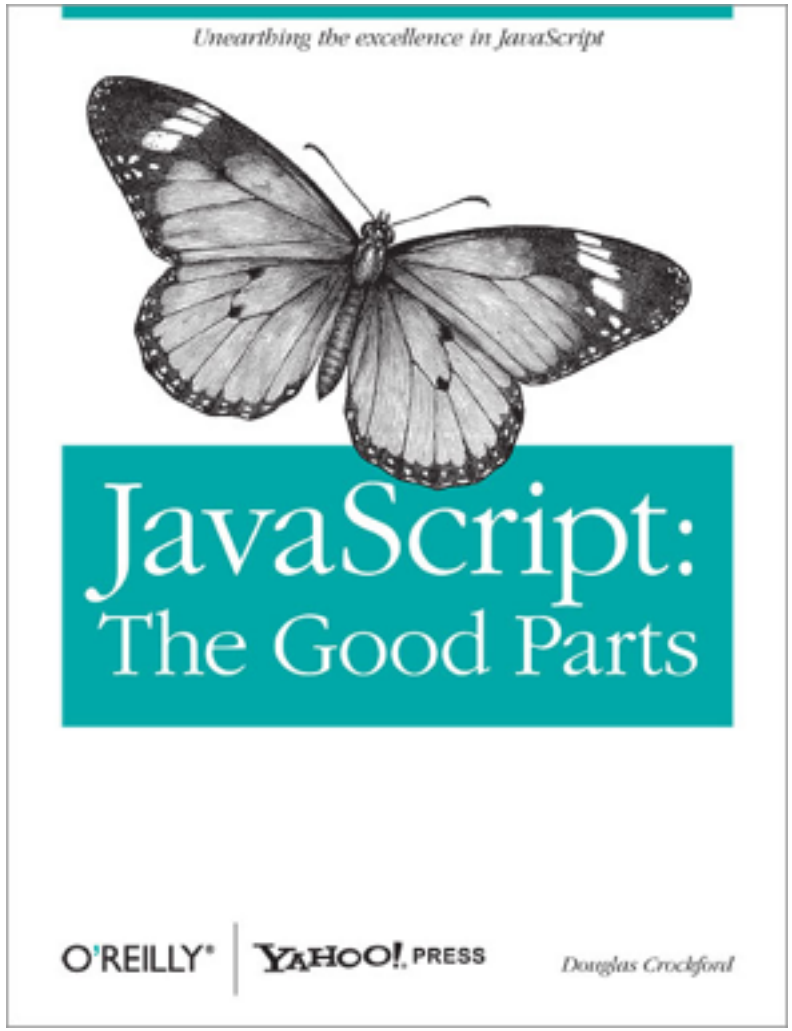
JavaScript's origins

- 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

Netscape®
TM

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

ECMAScript: “Standard” JavaScript

Carakan

Chakra

Spidermonkey

V8

JavaScriptCore

ECMAScript: “Standard” JavaScript

~~Carakan~~ V8

Chakra

Spidermonkey

V8

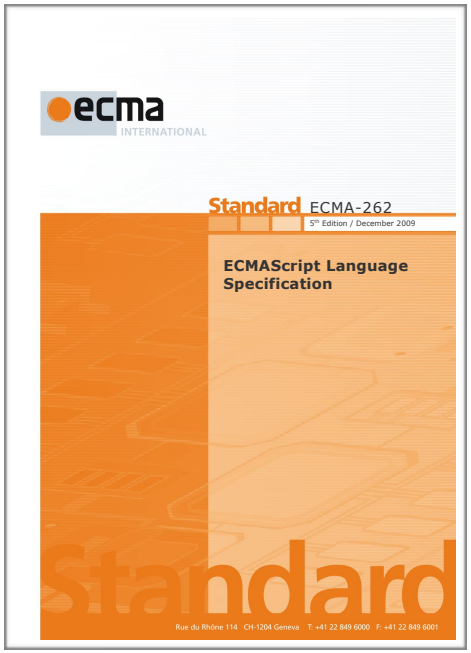
JavaScriptCore

(170.000+ npm packages!)

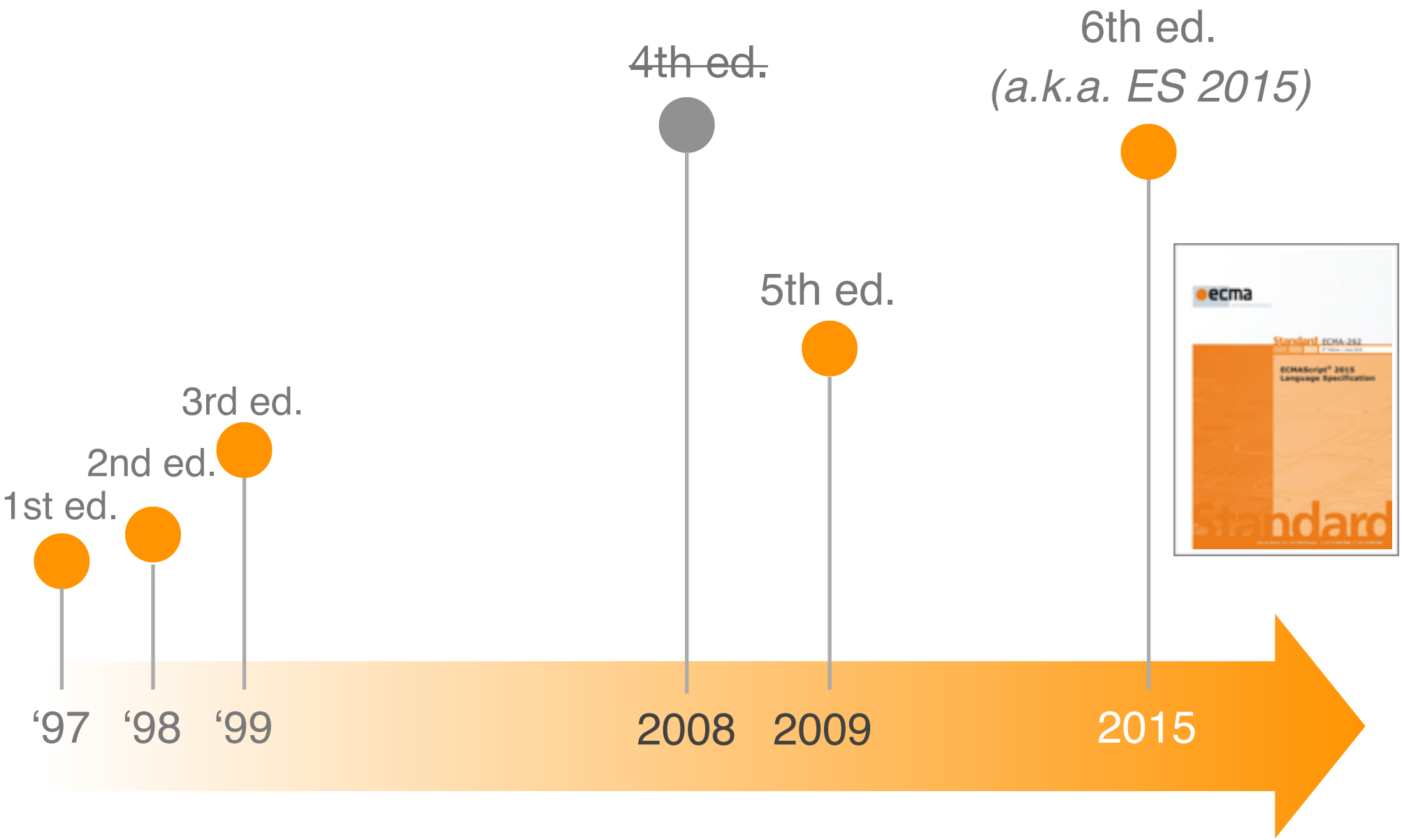
TC39: the JavaScript “standardisation committee”

- 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



A brief history of the ECMAScript spec



Part II
A brief tour of ECMAScript 6

ECMAScript 6

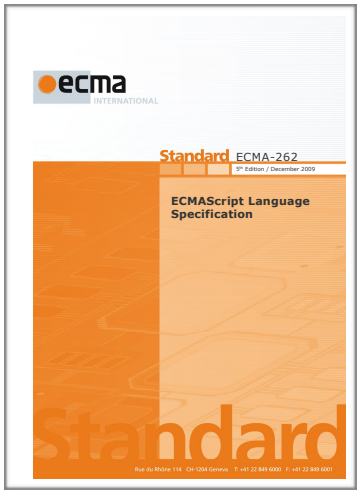
- Major update: many new features (too many to list here)
- Point-in-case:

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

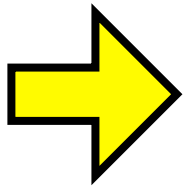
ECMAScript 6: shortlist

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

ECMAScript 6: classes

- All code inside a class is implicitly opted into strict mode!

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype = {
  toString: function() {
    return "[Point...]";
  }
}

var p = new Point(1,2);
p.x;
p.toString();
```

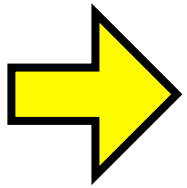


```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  toString() {
    return "[Point...]";
  }
}

var p = new Point(1,2);
p.x;
p.toString();
```

ECMAScript 6: classes

- All code inside a class is implicitly opted into strict mode!

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype = {
  toString: function() {
    return "[Point...]";
  }
}

var p = new Point(1,2);
p.x;
p.toString();
```

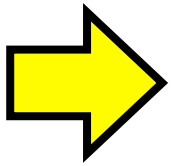


```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }

  toString() {
    return "[Point...]";
  }
}

var p = new Point(1,2);
p.x;
p.toString();
```

ECMAScript 6: modules

- All code inside a module is implicitly opted into strict mode!

```
<script>
var x = 0; // global
var myLib = {
  inc: function() {
    return ++x;
  }
};
</script>
```



```
<script type="module"
        name="myLib">
var x = 0; // local!
export function inc() {
  return ++x;
}
</script>
```

```
<script>
var res = myLib.inc();
</script>
```

```
<script type="module">
import { inc } from 'myLib';
var res = inc();
</script>
```

ECMAScript 6: modules

- All code inside a module is implicitly opted into strict mode!

```
<script>
var x = 0; // global
var myLib = {
  inc: function() {
    return ++x;
  }
};
</script>
```

```
<script type="module"
        name="myLib">
var x = 0; // local!
export function inc() {
  return ++x;
}
</script>
```

```
<script>
var res = myLib.inc();
</script>
```

```
<script type="module">
import { inc } from 'myLib';
var res = inc();
</script>
```

ECMAScript 6: shortlist

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

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

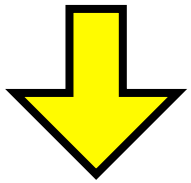
- Big Ticket items: Classes and Modules
- **Control flow Goodness:**
  - Iterators
  - Generators
  - **Promises**

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
  }
})
```

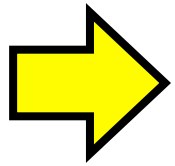
ES6

```
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

then

ECMAScript 6: shortlist

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

ECMAScript 6: longlist

- Luke Hoban has an excellent overview of all ES6 features at

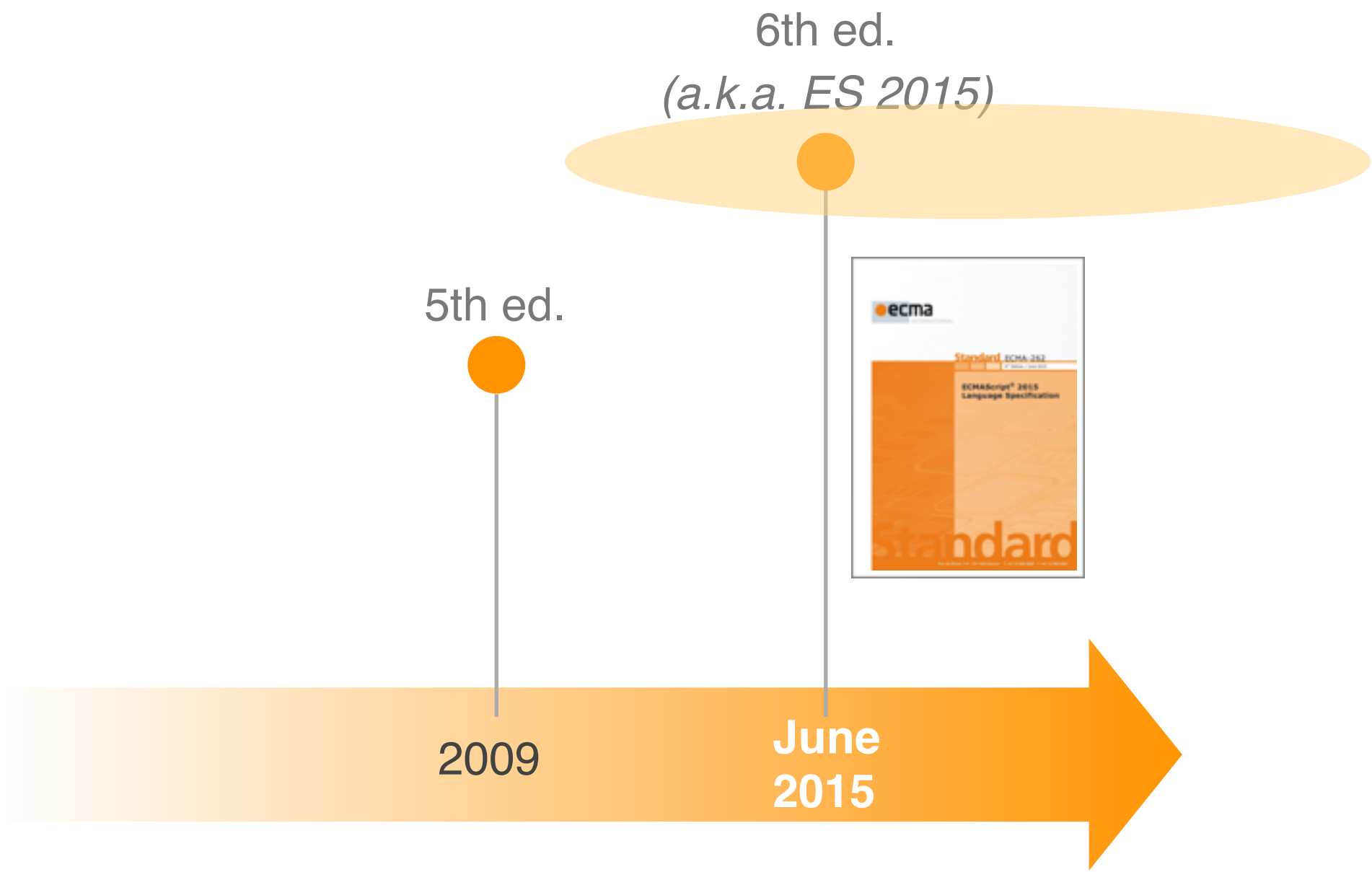
git.io/es6features

Luke Hoban,
Microsoft representative on TC39

- arrows
- classes
- enhanced object literals
- template strings
- destructuring
- default + rest + spread
- let + const
- iterators + for..of
- generators
- unicode
- modules
- module loaders
- map + set + weakmap + weakset
- proxies
- symbols
- subclassable built-ins
- promises
- math + number + string + array + object APIs
- binary and octal literals
- reflect api
- tail calls

Part III
Using ECMAScript 6 today

ECMAScript 6: timeline



ECMAScript 6 support (october 2015)

		Compilers/polyfills						Desktop browsers													
	74%	59%	71%	30%	18%	51%	17%	10%	20%	67%	80%	66%	70%	71%	72%	63%	63%	12%	21%	54%	70%	13%
Feature name	Current browser	Traceur	Babel + core-js[1]	Closure	JSX[2]	TypeScript + core-js	es6-shim	IE 10	IE 11	Edge 12[3]	Edge 13[3]	FF 38 ESR	FF 41	FF 42	FF 43	CH 46, OP 33[4]	CH 47, OP 34[4]	SF 6.1, SF 7	SF 7.1, SF 8	SF 9	WK	KQ 4.14[5]
Optimisation																						
proper tail calls (tail call optimisation)	0/2	0/2	1/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
Syntax																						
default function parameters	0/7	4/7	6/7	4/7	0/7	4/7	0/7	0/7	0/7	0/7	0/7	3/7	3/7	3/7	4/7	0/7	0/7	0/7	0/7	0/7	7/7	0/7
rest parameters	5/5	4/5	4/5	2/5	3/5	3/5	0/5	0/5	0/5	5/5	5/5	4/5	4/5	4/5	5/5	0/5	0/5	0/5	0/5	0/5	0/5	0/5
spread (...) operator	15/15	15/15	13/15	3/15	2/15	4/15	0/15	0/15	0/15	12/15	12/15	15/15	15/15	15/15	15/15	15/15	15/15	0/15	5/15	9/15	9/15	0/15
object literal extensions	6/6	6/6	6/6	4/6	5/6	6/6	0/6	0/6	0/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	6/6	0/6	1/6	5/6	6/6	0/6
for..of loops	7/9	9/9	9/9	6/9	2/9	3/9	0/9	0/9	0/9	6/9	6/9	7/9	7/9	7/9	7/9	7/9	7/9	0/9	2/9	8/9	8/9	0/9
octal and binary literals	4/4	2/4	4/4	2/4	0/4	4/4	2/4	0/4	0/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	4/4	0/4	0/4	4/4	4/4	0/4
template strings	5/5	4/5	4/5	3/5	4/5	3/5	0/5	0/5	0/5	4/5	4/5	5/5	5/5	5/5	5/5	5/5	5/5	0/5	0/5	5/5	5/5	0/5
RegExp "y" and "u" flags	0/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	0/4	2/4	2/4	2/4	2/4	2/4	2/4	0/4	0/4	0/4	0/4	0/4	0/4	0/4
destructuring	0/36	30/36	33/36	20/36	15/36	26/36	0/36	0/36	0/36	0/36	0/36	27/36	28/36	28/36	28/36	0/36	0/36	0/36	18/36	30/36	32/36	0/36
Unicode code point escapes	2/2	1/2	1/2	1/2	0/2	1/2	0/2	0/2	0/2	2/2	2/2	0/2	1/2	1/2	1/2	2/2	2/2	0/2	0/2	2/2	2/2	0/2
new.target	2/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	2/2	0/2	2/2	2/2	2/2	2/2	2/2	0/2	0/2	0/2	1/2	0/2
Bindings																						
const	5/8	6/8	6/8	6/8	0/8	6/8	0/8	0/8	8/8	8/8	8/8	8/8	8/8	8/8	8/8	5/8	5/8	1/8	1/8	1/8	8/8	2/8
let	5/10	8/10	8/10	8/10	0/10	6/10	0/10	0/10	8/10	8/10	8/10	0/10	0/10	0/10	0/10	5/10	5/10	0/10	0/10	0/10	10/10	0/10
block-level function declaration[13]	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes	Yes	No	No	No	No	Yes	Yes	No	No	No	No	No
Functions																						
arrow functions	11/12	10/12	9/12	9/12	7/12	8/12	0/12	0/12	0/12	9/12	11/12	8/12	10/12	10/12	11/12	10/12	11/12	0/12	0/12	0/12	9/12	0/12
class	23/23	16/23	19/23	9/23	15/23	16/23	0/23	0/23	0/23	0/23	23/23	0/23	0/23	0/23	0/23	0/23	0/23	0/23	0/23	15/23	23/23	0/23
super	8/8	7/8	6/8	4/8	7/8	7/8	0/8	0/8	0/8	0/8	8/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	0/8	6/8	8/8	0/8
generators	21/25	22/25	22/25	16/25	0/25	0/25	0/25	0/25	0/25	0/25	0/25	20/25	20/25	20/25	21/25	19/25	19/25	0/25	0/25	0/25	0/25	0/25
Built-ins																						
typed arrays	43/46	0/46	0/46	0/46	0/46	0/46	0/46	16/46	16/46	42/46	44/46	41/46	41/46	41/46	41/46	43/46	43/46	18/46	18/46	18/46	43/46	8/46
Map	16/18	13/18	18/18	0/18	0/18	18/18	14/18	0/18	7/18	15/18	16/18	14/18	15/18	16/18	16/18	16/18	16/18	0/18	10/18	17/18	17/18	0/18

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

COMPAT
ES

ECMAScript 5 support (october 2015)

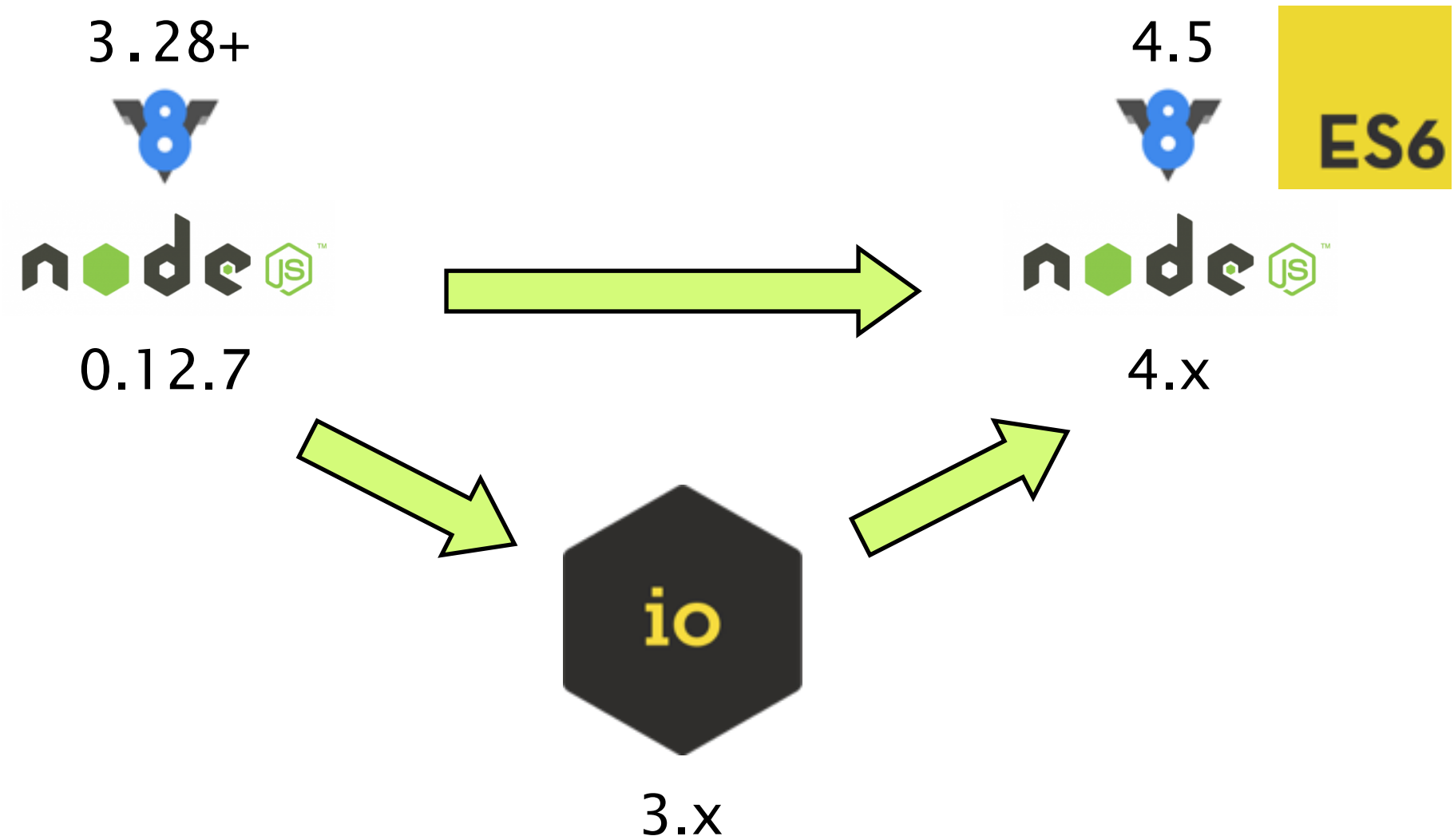
Feature name	Current browser	es5-shim	IE 9	IE 10+	FF 21+	SF 6+	WebKit	CH 23+, OP 15+	OP 12.10	Kanq 4.13	BESEN	Rhino 1.7	PhantomJS 2.0	EJS	Android 4.4+	iOS7/8
	97%	47%	95%	97%	97%	97%	100%	97%	97%	92%	100%	92%	100%	95%	97%	97%
Object.create	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.defineProperty	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.defineProperties	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.getPrototypeOf	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.keys	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.seal	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.freeze	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.preventExtensions	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isSealed	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isFrozen	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.isExtensible	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.getOwnPropertyDescriptor	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Object.getOwnPropertyNames	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Date.prototype.toISOString	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Date.now	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Date.prototype.toJSON	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No
Date.parse on invalid dates	No	Yes	No	No	No	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Array.isArray	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
JSON	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Function.prototype.bind	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
String.prototype.trim	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.indexOf	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.lastIndexOf	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.every	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.some	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.forEach	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.map	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.filter	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.reduce	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Array.prototype.reduceRight	Yes	Yes[4]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Getter in property initializer	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Setter in property initializer	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Property access on strings	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Reserved words as property names	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes
Zero-width chars in identifiers	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
parseInt() ignores leading zeros	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	Yes	Yes	Yes	Yes
Immutable undefined	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Strict mode	Yes	No	No	Yes[6]	Yes[7]	Yes	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes

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

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-side support



Part IV
Beyond ECMAScript 6

ES7 Proposals on the table

- Draft and Candidate proposals:

🚀	Proposal	Champion	Stage
	Exponentiation Operator	Rick Waldron	3
	Array.prototype.includes	Domenic Denicola, Rick Waldron	3
	SIMD.JS - SIMD APIs + polyfil	John McCutchan, Peter Jensen, Dan Gohman, Daniel Ehrenberg	3
	Async Functions	Brian Terlson	3
	Object.observe	Adam Klein	2
	function.sent metaproperty	Allen Wirfs-Brock	2
	Object.values/Object.entries	Jordan Harband	2
	Rest/Spread Properties	Sebastian Markbage	2
	Trailing commas in function parameter lists and calls	Jeff Morrison	2

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

ES7 Proposals on the table

- Draft and Candidate proposals:

🚀	Proposal	Champion	Stage
	Exponentiation Operator	Rick Waldron	3
	Array.prototype.includes	Domenic Denicola, Rick Waldron	3
	SIMD.JS - SIMD APIs + polyfil	John McCutchan, Peter Jensen, Dan Gohman, Daniel Ehrenberg	3
	Async Functions	Brian Terlson	3
	Object.observe	Adam Klein	2
	function.sent metaproperty	Allen Wirfs-Brock	2
	Object.values/Object.entries	Jordan Harband	2
	Rest/Spread Properties	Sebastian Markbage	2
	Trailing commas in function parameter lists and calls	Jeff Morrison	2

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

ECMAScript **7**: async functions

- 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
});
```

ES7

```
// 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)

ES7

```
(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)

ES7

```
(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** (!)

- BabelJS async functions plug-in based on Facebook Regenerator
- Works in browsers too!

facebook.github.io/regenerator

github.com/lukehoban/ecmascript-asyncawait

ES7

```
(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
  }
}())
```

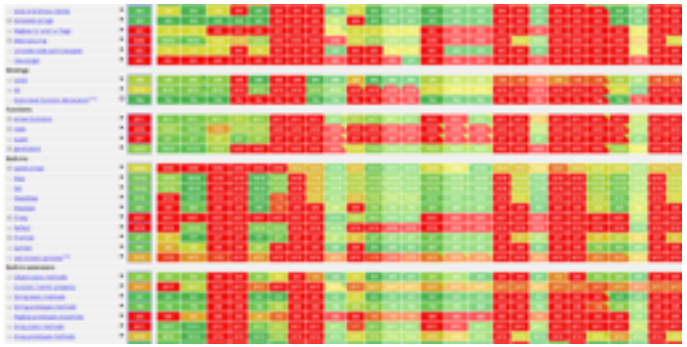
ES5

```
(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]]);
})();
```

Wrap-up

Take-home messages

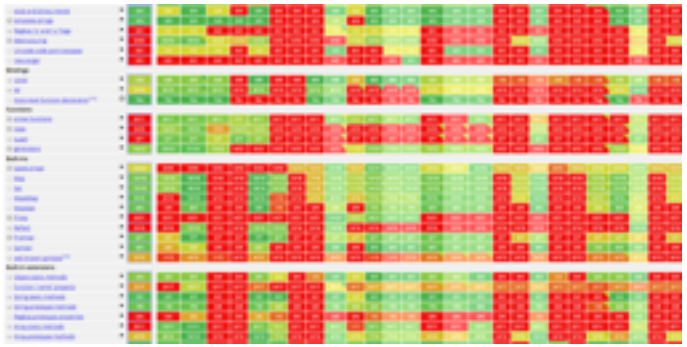
ES6 is a *major* upgrade



Expect engines to upgrade gradually

Take-home messages

ES6 is a *major* upgrade



Expect engines to upgrade gradually

BABEL

TypeScript

Use ES6 to ES5 compilers to bridge the gap

ES6

You can start using ES6 today!

Thanks for listening!

ECMAScript 2015

The Future of JavaScript is Now!

Tom Van Cutsem

SPLASH-I 2015

@tvcutsem