CRASH COURSE

ES6

SOMETIMES CALLED

ES.NEXT

SOMETIMES CALLED

ES.HARMONY

SOMETIMES CALLED

ES7

SOMETIMES CALLED

ES8

SOMETIMES CALLED

MANY OTHERS

I'M STICKING WITH

ES6

ES STANDS FOR

ECMA SCRIPT

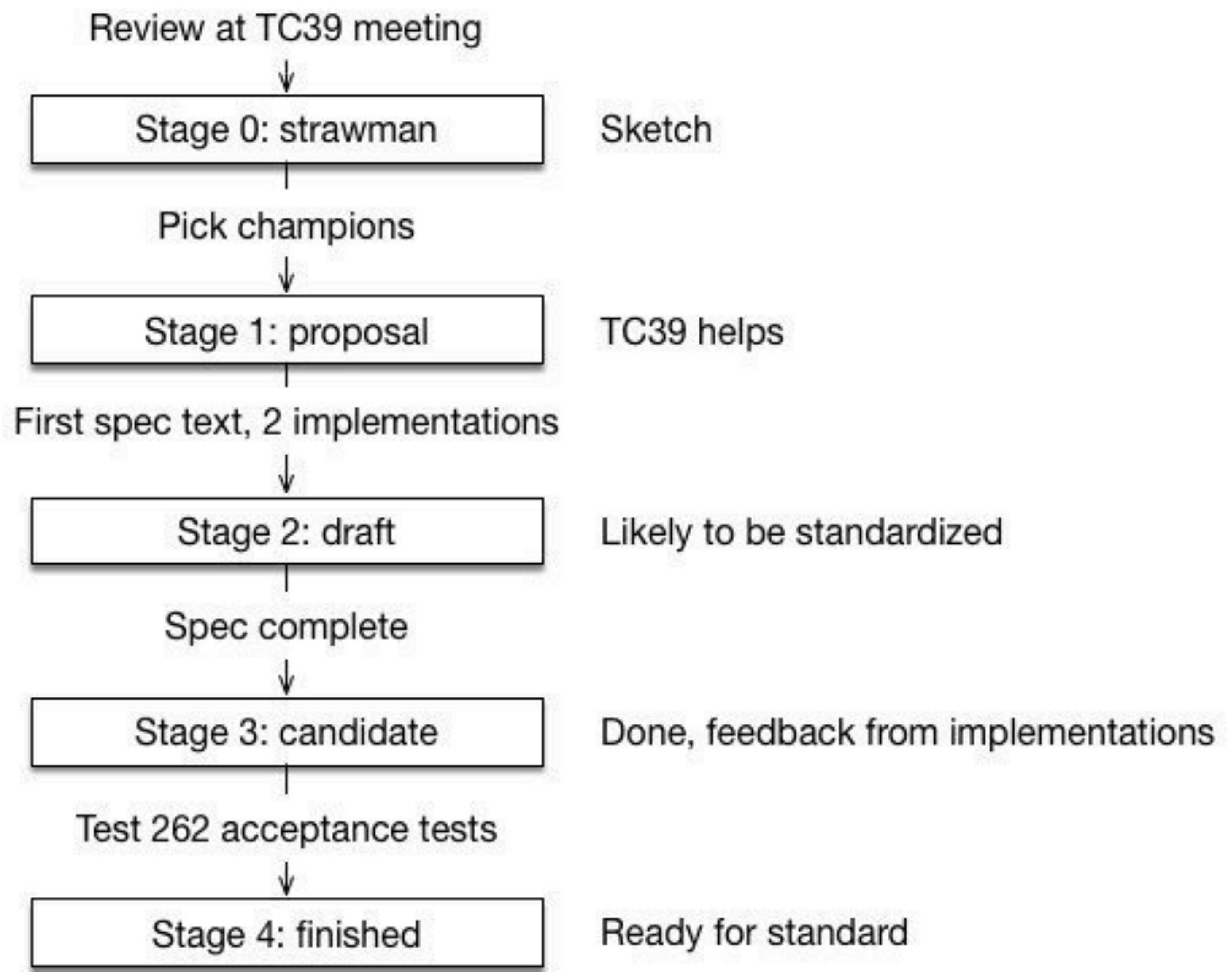
TODAY

- About ES6
- ES Modules
- Variables
- Destructuring
- Function expressions
- Promisses

ABOUT

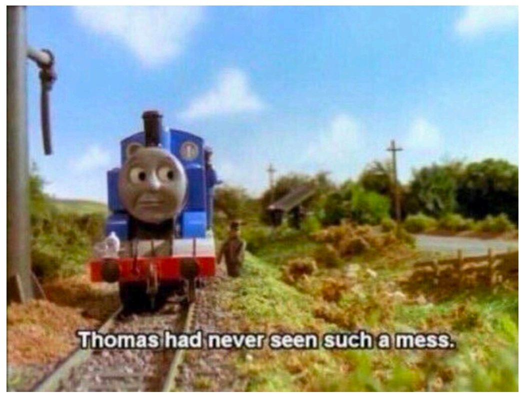
ES6

TC39 PROCESS



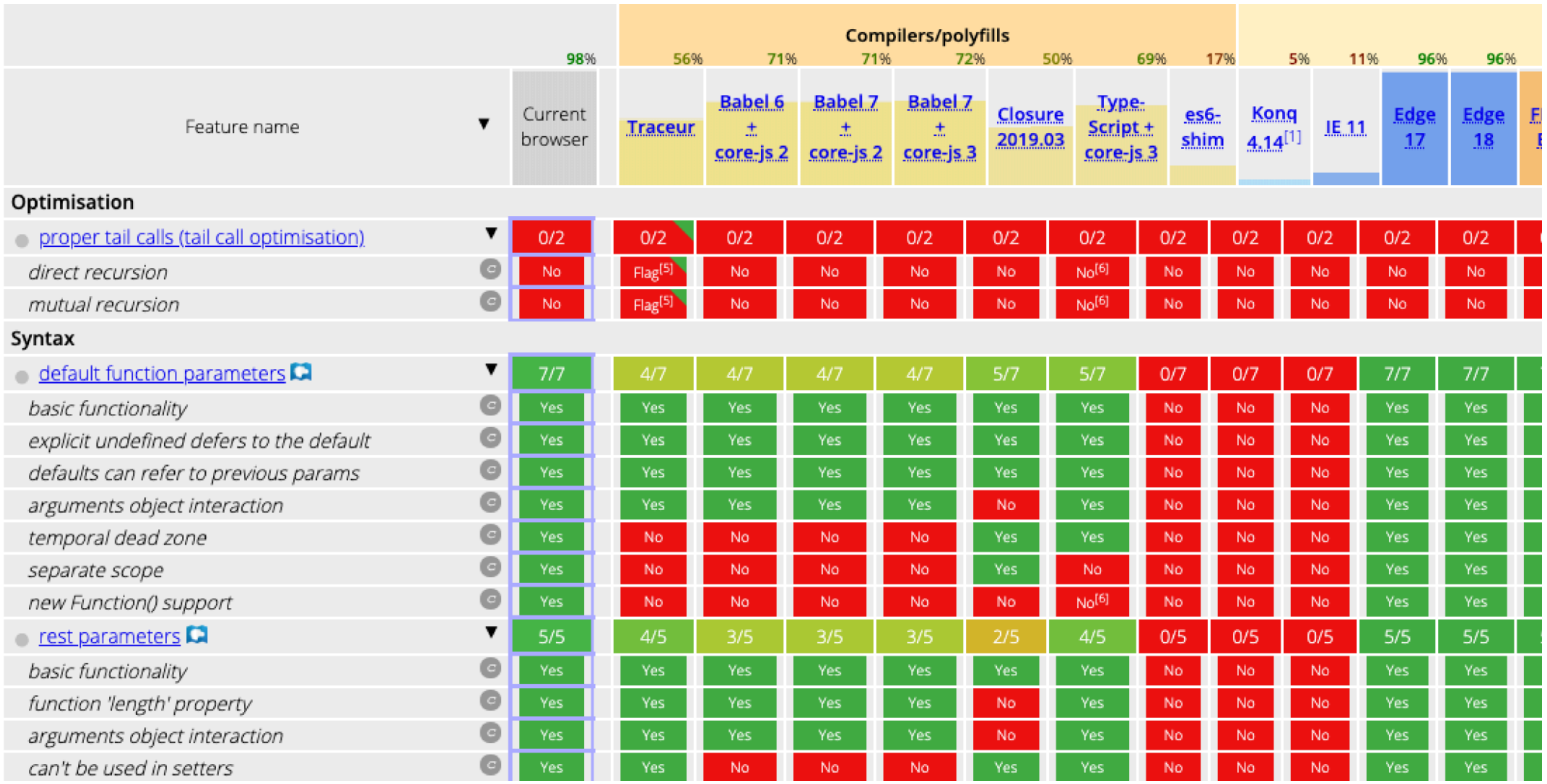
INTRODUCTION - RUNTIMES

When a backend developer tries out Javascript and its ecosystem



[HTTPS://EN.WIKIPEDIA.ORG/WIKI/LIST_OF_ECMASCRIPT_ENGINES](https://en.wikipedia.org/wiki/List_of_ECMAScript_engines)

INTRODUCTION - COMPATIBILITY



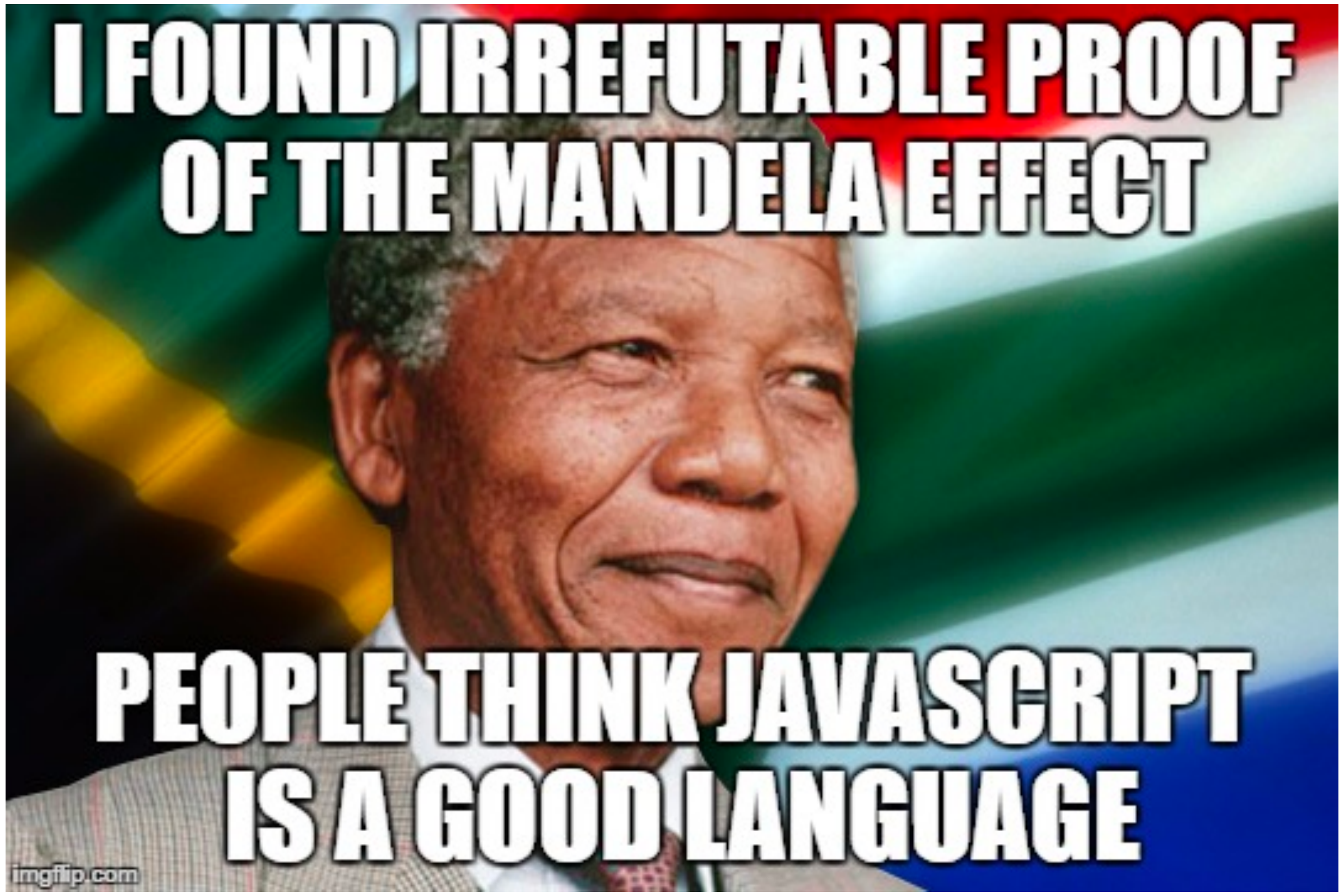
		Compilers/polyfills										
	98%	56%	71%	71%	72%	50%	69%	17%	5%	11%	96%	96%
Feature name ▼	Current browser	Traceur	Babel 6 + core-js 2	Babel 7 + core-js 2	Babel 7 + core-js 3	Closure 2019.03	Type-Script + core-js 3	es6-shim	Konq 4.14[1]	IE 11	Edge 17	Edge 18
Optimisation												
proper tail calls (tail call optimisation) ▼	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2	0/2
direct recursion	No	Flag[5]	No	No	No	No	No[6]	No	No	No	No	No
mutual recursion	No	Flag[5]	No	No	No	No	No[6]	No	No	No	No	No
Syntax												
default function parameters ▼	7/7	4/7	4/7	4/7	4/7	5/7	5/7	0/7	0/7	0/7	7/7	7/7
basic functionality	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes
explicit undefined defers to the default	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes
defaults can refer to previous params	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes
arguments object interaction	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No	Yes	Yes
temporal dead zone	Yes	No	No	No	No	Yes	Yes	No	No	No	Yes	Yes
separate scope	Yes	No	No	No	No	Yes	No	No	No	No	Yes	Yes
new Function() support	Yes	No	No	No	No	No	No[6]	No	No	No	Yes	Yes
rest parameters ▼	5/5	4/5	3/5	3/5	3/5	2/5	4/5	0/5	0/5	0/5	5/5	5/5
basic functionality	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes	Yes
function 'length' property	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No	Yes	Yes
arguments object interaction	Yes	Yes	Yes	Yes	Yes	No	Yes	No	No	No	Yes	Yes
can't be used in setters	Yes	Yes	No	No	No	Yes	Yes	No	No	No	Yes	Yes

- Transpiles (compiles) ES6 to ES5
- Makes sure new language features run in older runtime environments

[HTTPS://BABELJS.IO/](https://babeljs.io/)

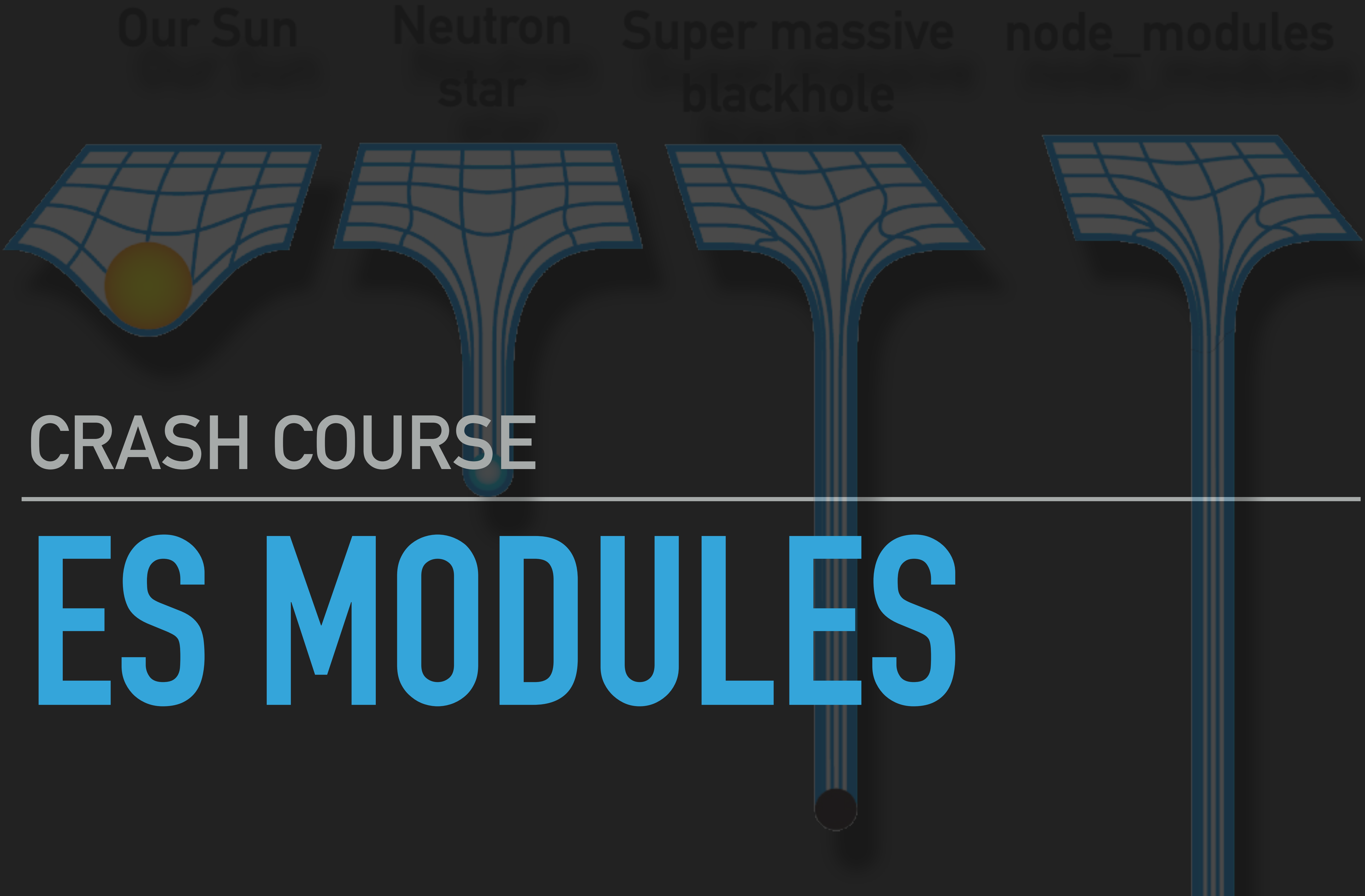
RECAP

- Can’t make up it’s mind on the name
- Design by committee is a slow process
- Many runtime environments with different implementation levels of standards
- Compatibility issues between those runtime environments
- Transpilers required


I FOUND IRREFUTABLE PROOF OF THE MANDELA EFFECT
PEOPLE THINK JAVASCRIPT IS A GOOD LANGUAGE
imgflip.com

REPEAT AFTER ME

JS IS A GOOD LANGUAGE



CRASH COURSE

ES MODULES

DEFAULT EXPORT / IMPORT

```
// subtract.js
export default function(x, y) {
  return x - y;
}
```

```
// main.js
import subtract from 'subtract';

console.log(substract(10, 5)); // 5
```

```
// main2.js
import minus from 'subtract';

console.log(minus(10, 5)); // 5
```

NAMED EXPORT / IMPORT

```
// mathLib.js
export function square(x) {
  return x * x;
}

export function sum(x, y) {
  return x + y;
}
```

```
// main.js
import { square, sum } from 'mathLib';

console.log(square(9)); // 81
console.log(sum(15, 1)); //16
```

NAMESPACE IMPORT

```
// mathLib.js
export function square(x) {
  return x * x;
}

export function sum(x, y) {
  return x + y;
}
```

```
// main.js
import * as math from 'mathLib';

console.log(math.square(9)); // 81
console.log(math.sum(15, 1)); //16
```

MIXING DEFAULT AND NAMED EXPORTS

```
export default class Math () {
  static function square(x) {
    return x * x;
  }

  static function sum(x, y) {
    return x + y;
  }

  static function subtract(x, y) {
    return x -y;
  }
}


export default Math;
export {
  Math.square as square,
  Math.sum as sum,
  Math.subtract as subtract
};
```

MIXING DEFAULT AND NAMED EXPORTS

```js
// main.js
import mathametics, { square, sum } from 'mathLib';

console.log(mathametics.square(9)); // 81
console.log(mathametics.sum(15, 1)); //16

console.log(square(9)); // 81
console.log(sum(15, 1)); //16
```

ALIASING IMPORTS

```
export function a() { return 'a' };
```

```
import { a as b } from 'a';
b(); //a
```

ALIASING IMPORTS

```
import { MyModule as MyOtherModule } from 'module';
import { MyOtherModule as MyModule } from 'module';
```

SYNTAX OVERVIEW

```
import something from 'module-name';
import * as name from 'module-name';
import { export } from 'module-name';
import { export as alias } from 'module-name';
import { export1 , export2 } from 'module-name';
import { foo , bar } from 'module-name/path/to/specific/un-exported/file';
import { export1 , export2 as alias2 , [...] } from 'module-name';
import defaultExport, { export [ , [...] ] } from 'module-name';
import defaultExport, * as name from 'module-name';
import 'module-name';
```

CRASH COURSE

VARIABLES

HOISTING

A mechanism where variable declarations are moved to the top of their scope

```
console.log(hoistedVariable);

var hoistedVariable = "I'm hoisted";
```

What is the output of console.log()?

1. Uncaught ReferenceError: hoistedVariabled is not defined
2. I'm hoisted
3. Undefined

THREE WAYS OF DECLARING A VARIABLE

```
var myVar;
let myOtherVar;
const myConst = [];
```

VAR

```
var i = 36;

function aFunction() {
  var i = 22;

  for (var i = 0; i < 10; i++) {
    console.log(i); // 0, 1, 2, 3, ...
  }
  // `i` get overwritten in the for loop
  console.log(i); // 10
}
aFunction();

console.log(i); // 36
```

LET

```
const aFunction = () => {
  let i = 22;

  for (let i = 0; i < 10; i++) {
    console.log(i); // 0, 1, 2, 3, ...
  }
  console.log(i); // 22
}
aFunction();
```

CONST

```
const myObj = {};
const myObj = []; // SyntaxError: Identifier 'myObj' has already been
declared
```

CONST

```
const myObj = {}:
myObj.someProperty = '123';

console.log(myObj.someProperty); // 123
```

CONST

```
const myObj = Object.freeze({});
myObj.someProperty = '123'; // TypeError
```

POP QUIZ

```
function aFunction() {
  someVar = 'some value';
}
aFunction();

console.log(someVar);
```

What is the output of console.log()?

1. Uncaught ReferenceError: someVar is not defined
2. "some value”
3. Undefined

CRASH COURSE

DESTRUCTURING

DESTRUCTURING AN ARRAY

```
const name = ['John', 'Doe'];
let [ firstname, lastname ] = name;

console.log(name[0], name[1]); // John Doe
console.log(firstname, lastName); // John Doe
```

DESTRUCTURING AN OBJECT

```
const myObject = {
  aKey: 'a value',
  aFunction: function() { return 'return value'; }
}

const { aKey, aFunction } = myObject;
console.log(aKey); // 'a value'
console.log(aFunction()); // 'return value'
```

NESTED DESTRUCTURING

```
const student = {
  name: 'John the Programmer',
  age: 19,
  testScores: {
    php: 89,
    javascript: 55,
  }
};

const { name, testScores: { javascript, php } } = student;
console.log(javascript, php); // 89 55
```

REST PROPERTIES

```
const student = {
  name: 'John the Programmer',
  age: 19,
  testScores: {
    php: 89,
    javascript: 55,
  }
};

const { name, ...theRest } = student;
console.log(theRest); // {age: 19, testScores: {php: 89, javascript: 55 }}
```

SPREAD PROPERTIES

```
const student = {
   name: 'John the Programmer',
   age: 19,
   testScores: {
      php: 89,
      javascript: 55,
   },
};
```

```
const updatedStudent = {
   ...student,
   testScores: {
      ...student.testScores ,
      lolcode: 100,
   },
};

console.log(updatedStudent);
```

SPREAD PROPERTIES

```
{
  "name":"John the Programmer",
  "age":19,
  "testScores": {
    "php":89,
    "javascript":55,
    "lolcode":100
  }
}
```

SPREAD PROPERTIES

```
function sum(x, y, z) {
  return x + y + z;
}
const numbers = [1,2,3];
```

```
const [x, y, z] = numbers;
sum(x, y, z); // 6
sum(...numbers); // 6
```

```
console.log(numbers); // [1, 2, 3]
console.log(...numbers); // "1 2 3"
console.log(x, y, z); // "1 2 3"
```

SHALLOW COPY

```
const arr1 = ['a', 'b', 'c'];
const arr2 = [...arr1, 'apple'];
const arr3 = [...arr1, ...arr2];

console.log(arr1); // ['a', 'b', 'c']
console.log(arr2); // ['a', 'b', 'c', 'apple']
console.log(arr3); // ['a', 'b', 'c', 'a', 'b', 'c', 'apple']
```

SHALLOW COPY ONLY WORKS ONE LEVEL DEEP

```
const arr = [
  {
    somekey: 'some value',
    nestedObject: {
      nestedKey: 'nested value'
    }
  },
  'pear'
];

const otherArr = [...arr];
otherArr[0].nestedObject.nestedKey = 'different value';

console.log(arr[0].nestedObject.nestedKey); // 'different value'
```

POP QUIZ

```
const avengers = {
  operation: 'Assemble',
  members: [
    { ironMan: 'Tony Stark' },
    { captainAmerica: 'Steve Rogers' },
    { blackWidow: 'Natasha Romanoff' }
  ]};

const { operation, members:[, batman] } = avengers;

console.log(batman);
```

What is the output of console.log()?

```
{ captainAmerica: 'Steve Rogers' }
```

CRASH COURSE

FUNCTION EXPRESSIONS

BASIC SYNTAX

```
(param1, param2) => { statements };
```

- Can’t be used as constructor methods
- Don’t have their own `this`, `super`, `arguments`, or `new.target`
- `this` is lexical scope, picked up from their surrounding
- Best suited class methods or anonymous callbacks

THIS

```
var Bear = {
  name: 'Winnie the Pooh',

  foods: ['honey', 'strawberries'],

  sayName: function() {
    console.log(this.name); // Winnie the Pooh
  },

  eat: function() {
    this.foods.forEach(function(food) {
      console.log(this.name + ' eats ' + food);
    })
  }
}

Bear.sayName(); // Winnie the Pooh
Bear.eat() // eats honey,  eats strawberries
```

FUNCTION EXPRESSIONS

THIS

```
var Bear = {
  name: 'Winnie the Pooh',

  foods: ['honey', 'strawberries'],

  sayName: function() {
    console.log(this.name); // Winnie the Pooh
  },

  eat: function() {
      this.foods.forEach((food) => {
        console.log(this.name + ' eats ' + food);
      })
  }
}

Bear.sayName(); // Winnie the Pooh
Bear.eat() // Winnie the Pooh eats honey,  Winnie the Pooh eats strawberries
```

REST PARAMETERS

```
function multiply(multiplier, ...theArgs) {
  return theArgs.map(element => multiplier * element);
}

console.log(multiply(2, 16, 100)); // 32, 200
```

```
function sum(...nums) {
    return nums.reduce((total, num)=> total + num);
}

console.log(sum(1 ,2 ,3 ,4)); // 10
```

SYNTAX VARIANTS

```
// ES5 function expression
var powerof2 = function(x) {
  return x * x;
}

// Remove function keyword, add arrow
var powerof2 = (x) => {
  return  x * x;
}

// Return is implied in this case, remove return keyword and brackets
var powerof2 = (x) => x * x;

// Only one parameter, parens not required
var powerof2 = x => x * x;
```

CRASH COURSE

PROMISES

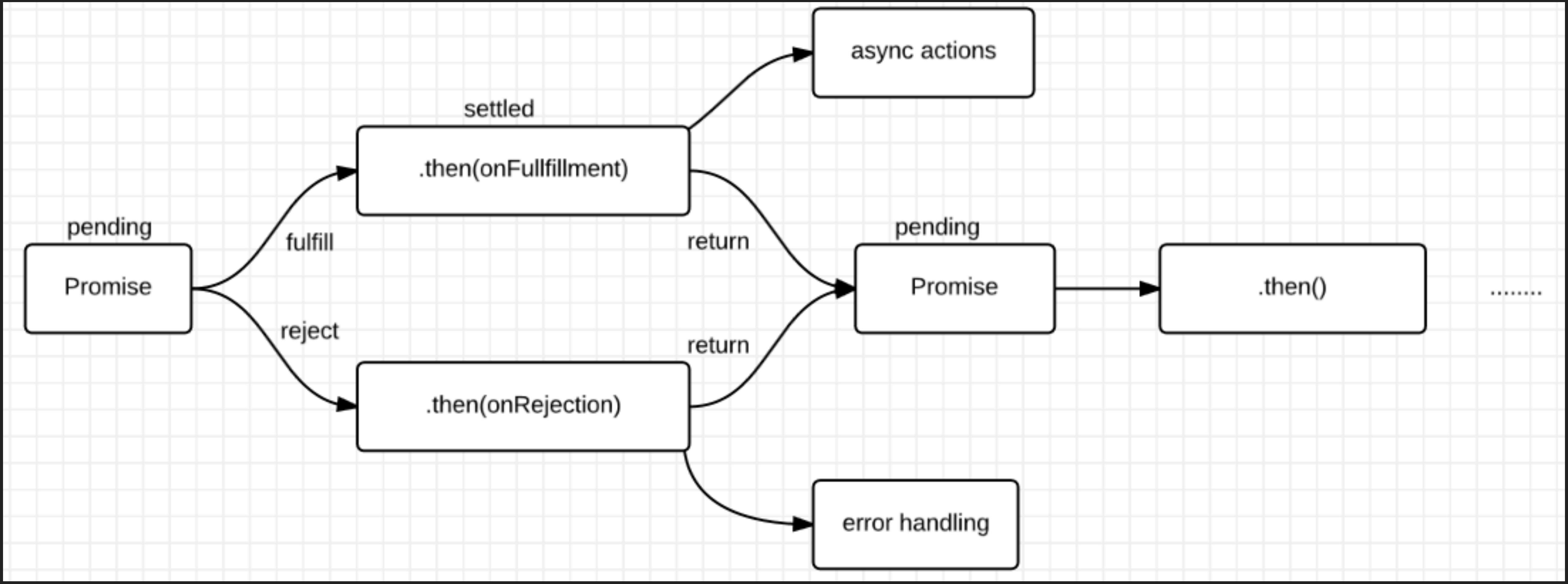
WHAT IS A PROMISE

A Promise is an object representing the eventual completion or failure of an asynchronous operation.

WHY DO WE NEED THEM?

Imagine you are a kid and want a new phone

STATES


pending
Promise
fulfill
reject
settled
.then(onFullfillment)
.then(onRejection)
async actions
error handling
return
return
pending
Promise
.then()
....

MOM, CAN I HAVE A NEW PHONE?

```
const isMomHappy = !!(Math.random() >= 0.5);

// Promise
const willIGetNewPhone = new Promise((resolve, reject) => (
   isMomHappy
      ? resolve({
         brand: 'Samsung',
         color: 'black',
      })
      : reject(Error('Mom isn\'t happy'))));

// call promise
const askMom = () => {
  willIGetNewPhone
    .then(fulfilled => console.log(fulfilled))
    .catch(error => console.log(error.message));
};

askMom();
```

MOM, CAN I HAVE A NEW PHONE?

```
const isMomHappy = !!(Math.random() >= 0.5);

// Promise
const willIGetNewPhone = new Promise((resolve, reject) => (
     isMomHappy
          ? resolve({
               brand: 'Samsung',
               color: 'black',
          })
          : reject(Error('Mom isn\'t happy'))));


// call promise
const askMom = () => {
  willIGetNewPhone
    .then(fulfilled => console.log(fulfilled))
      .catch(error => console.log(error.message));
};

askMom();
```

EXAMPLE: MAKE A HTTP REQUEST

```
let isLoading = true;

fetch('https://url.tld')
    .then(response => response.json())
    .then(jsonData => JSON.stringify(jsonData))
    .catch(error => console.log(error))
    .finally(() => { isLoading = false; });
```

QUESTIONS?

ES6