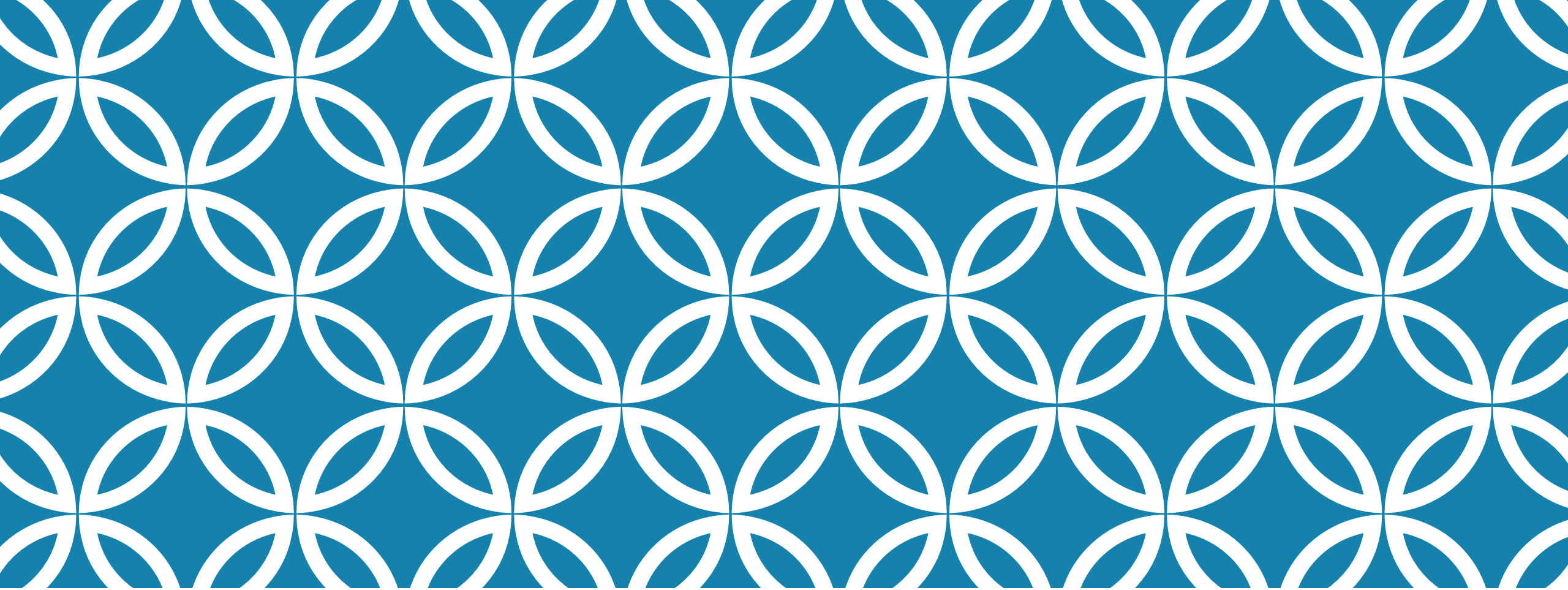


JAVASCRIPT MODULES

Andrew Sheehan

Boston University
Metropolitan College

ALSO KNOWN AS

ES Modules

~

ECMAScript Modules

SUPPORT

IE	Edge *	Firefox	Chrome	Safari	Opera	Safari on iOS *	Opera Mini *	Android Browser *
	12-14							
	[1] [6] 15	2-53	4-59	3.1-10	10-46	3.2-10.2		
	[6] 16-18	[2] 54-59	[1] 60	[4] [5] 10.1	[1] 47	[4] [5] 10.3		
6-10	79-91	60-90	61-91	11-14	48-77	11-14.4		2.1-4.4.4
11	92	91	92	14.1	78	14.7	all	92
		92-93	93-95	15-TP				

EXPOSE
BY EXPORTING

When something is made available to other modules or pages, it's called an **export**

PARTITION YOUR JS CODE

Provides a means to group your code.

Anything not exported is private.

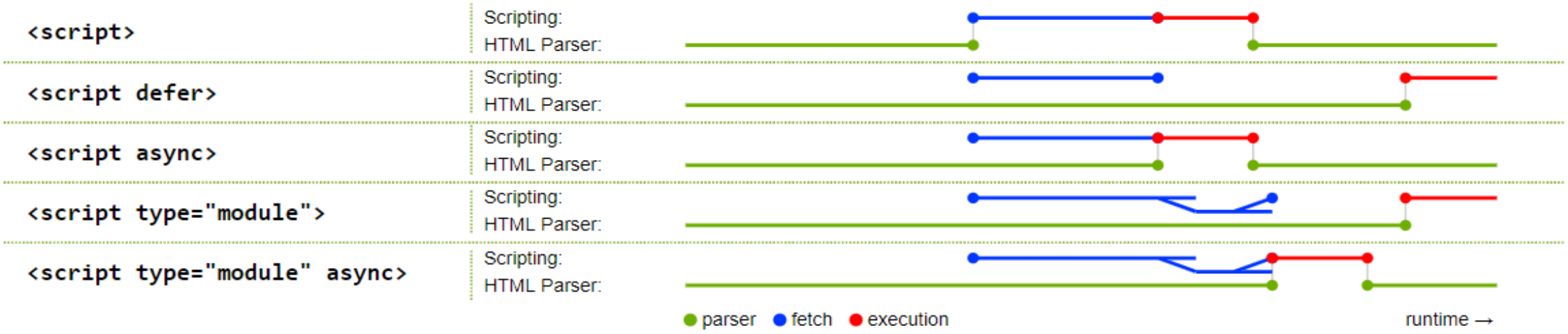
NOT USING MODULES?

When you include multiple .js files into your HTML, you are not provided any scope protection.

Most declarations go into the **global space**

MODULES ‘DEFER’

Classic `<script>`s block the HTML parser by default. You can work around it by adding the `defer` attribute, which ensures that the script download happens in parallel with HTML parsing.



1 MODULE ONE FILE

Declare 1 module per file.
That is the per specification.

DEFAULT EXPORT

NAMED EXPORTS

TYPES OF EXPORTS

NAMED EXPORTS

```
let company = "TutorialsPoint"

let getCompany = function(){
   return company.toUpperCase()
}

let setCompany = function(newValue){
   company = newValue
}

export {company,getCompany,setCompany}
```

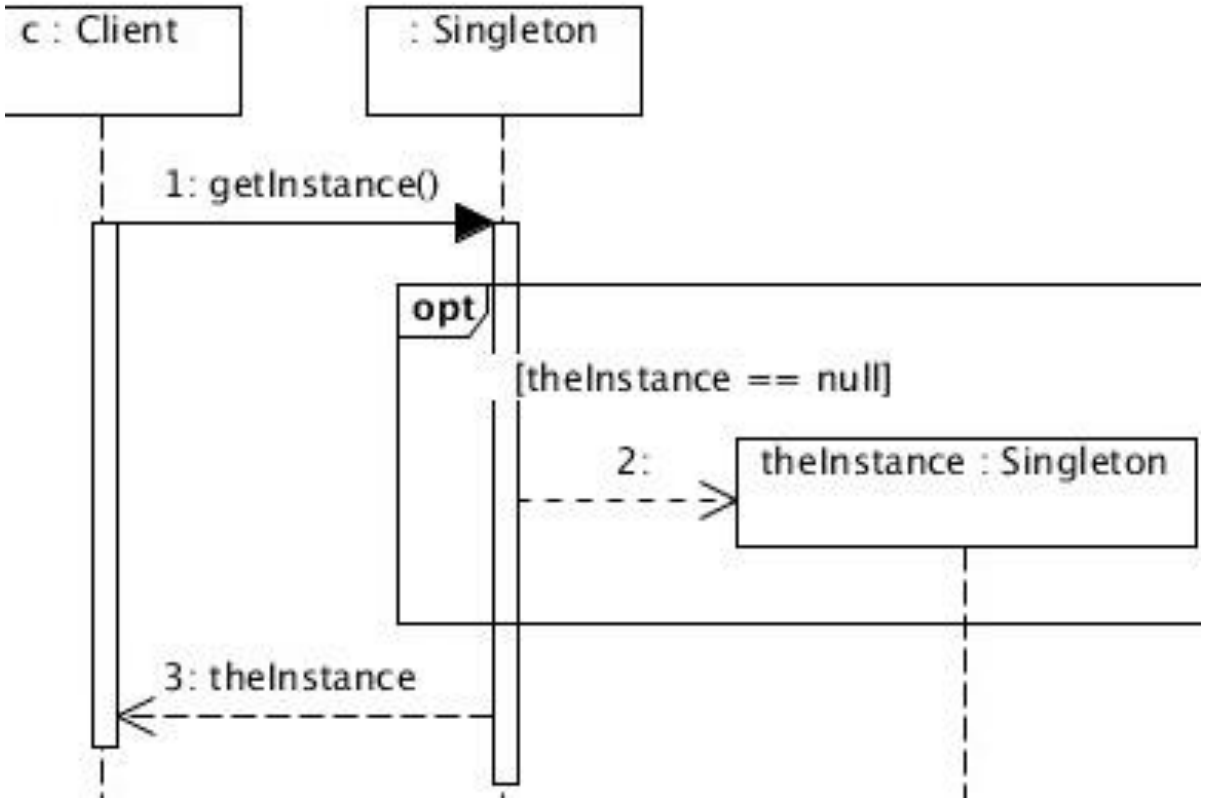
```
import {company as x, getCompany as y} from './company1.js'

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

```
//using multiple export keyword
export function f() { };
export class g { … };
…
export const PI = 3.14;

//or -  using single export keyword
export {f, g, PI};
```

NAMED EXPORTS

```
//there can only be 1 default in your module…
export default function renderCircle(radius, fill, options) {…}
```

DEFAULT EXPORT



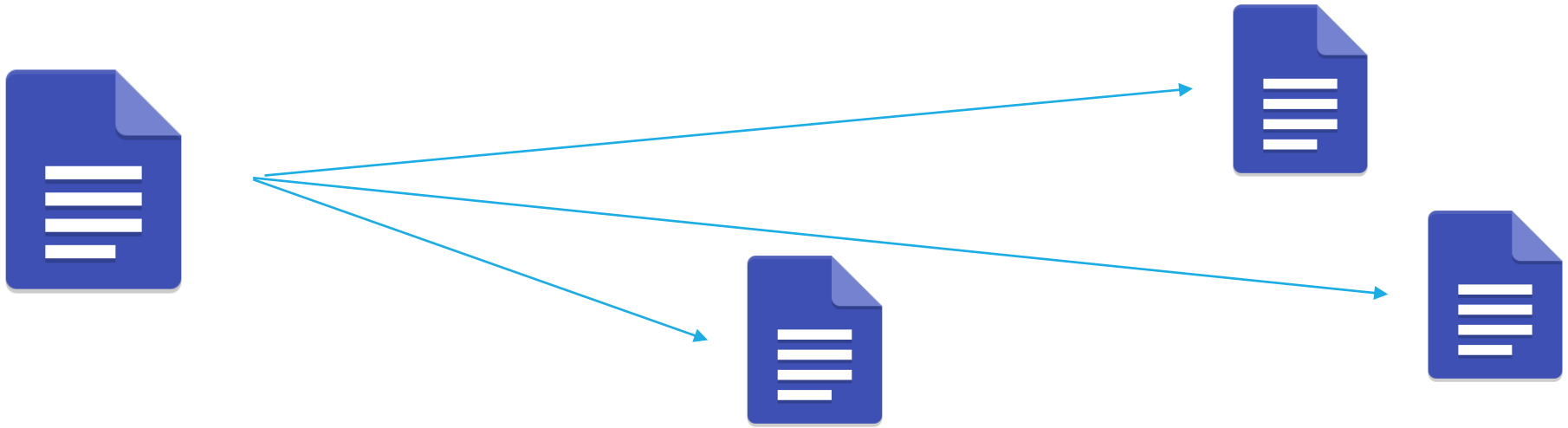
SINGLETON

Once you import a module, it will not be imported again.

*It will be **cached***

MODULES CAN HAVE
DIRECT DEPENDENCIES/TRANSITIVE DEPENDENCIES

When a module requires something from another module.



SAME DIRECTORY?

? In the same folder as the module

Use: './'

COMMONJS

USED BY NODE


```
// add.js
function add (a, b) {
  return a + b
}

module.exports = add
```

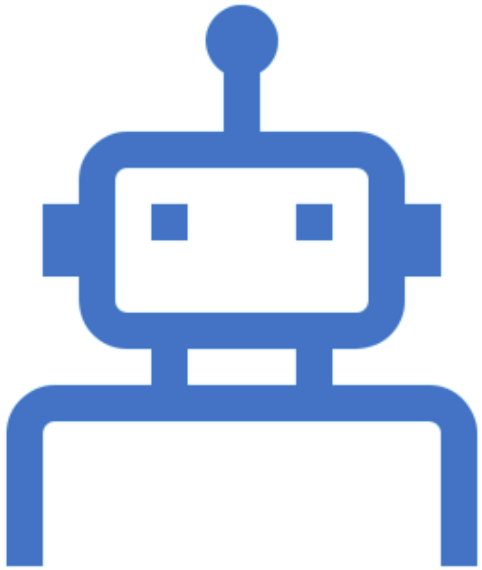
```
// index.js
const add = require('./add')

console.log(add(4, 5))
//9
```

REQUIRE()
PATHNAMES

'../components'

Up a directory from the module,
then use a folder called 'components'

import React from ‘react’;



You will not use require()

ES6

MODULES

STRICTNESS

No need for ‘use strict’ in your modules (it’s the default)

Everything inside an ES6 module is private by default, and runs in strict mode (there’s no need for `'use strict'`). Public variables, functions and classes are exposed using `export`.

CONDITIONAL IMPORT

You cannot…

```
if (Math.random()) {
    import 'foo'; // SyntaxError
}

// You can't even nest `import` and `export`
// inside a simple block:
{
    import 'foo'; // SyntaxError
}
```

SMALLER MODULES => PERFORMANCE GAINS

- Smaller is better with any download..
- Optimize your code
- Strict mode – all the time with modules!

```
import { sum } from './lib.js';



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

ES6 SINGLE EXPORT

DYNAMIC IMPORT()

```
<script type="module">
  (async () => {
    const moduleSpecifier = './lib.mjs';
    const {repeat, shout} = await import(moduleSpecifier);
    repeat('hello');
    // → 'hello hello'
    shout('Dynamic import in action');
    // → 'DYNAMIC IMPORT IN ACTION!'
  })();
</script>
```

When you do not want to load a module upfront, but on-demand

You can load your modules when you need to use them.

Performance improvement!

.MJS EXTENSION

The Chrome V8 team recommends it's use:

Still, we recommend using the `.mjs` extension for modules, for two reasons:

1. During development, the `.mjs` extension makes it crystal clear to you and anyone else looking at your project that the file is a module as opposed to a classic script. (It's not always possible to tell just by looking at the code.) As mentioned before, modules are treated differently than classic scripts, so the difference is hugely important!
2. It ensures that your file is parsed as a module by runtimes such as Node.js and `d8`, and build tools such as Babel. While these environments and tools each have proprietary ways via configuration to interpret files with other extensions as modules, the `.mjs` extension is the cross-compatible way to ensure that files are treated as modules.

```
// Supported:
import {shout} from './lib.mjs';
import {shout} from '../lib.mjs';
import {shout} from '/modules/lib.mjs';
import {shout} from 'https://simple.example/modules/lib.mjs';
```

```
// Not supported (yet):
import {shout} from 'jquery';
import {shout} from 'lib.mjs';
import {shout} from 'modules/lib.mjs';
```

MODULE SPECIFIERS MUST BE FULL URL'S OR RELATIVE URL'S WITH /, ./, OR ../

MODULES

YOUR HTML SCRIPT TAG

```js
// html.js
export function tag (tag, text) {
  const el = document.createElement(tag)
  el.textContent = text

  return el
}
```

```html
<script type="module">
  import { tag } from './html.js'

  const h1 = tag('h1', ' Hello Modules!')
  document.body.appendChild(h1)
</script>
```

MODULES: START USING THEM TODAY

You do not need to wait.

Works everywhere.

(Nobody cares about IE anymore…)