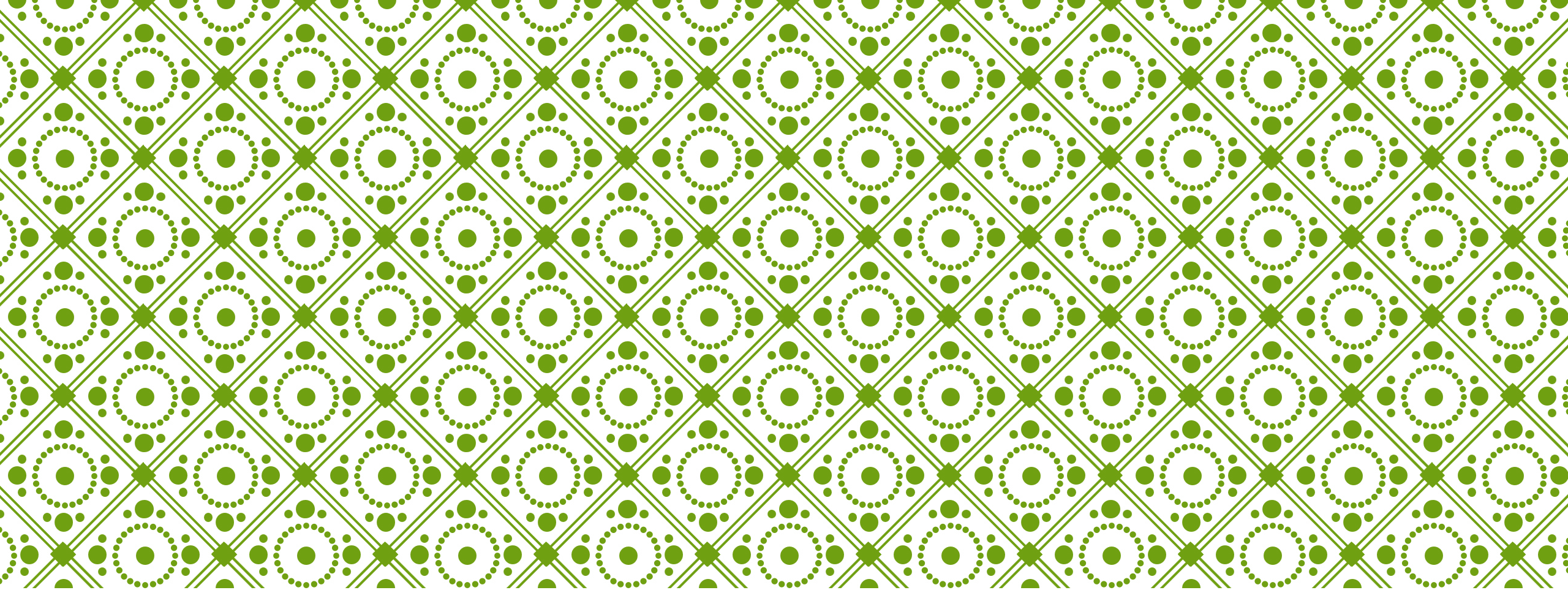
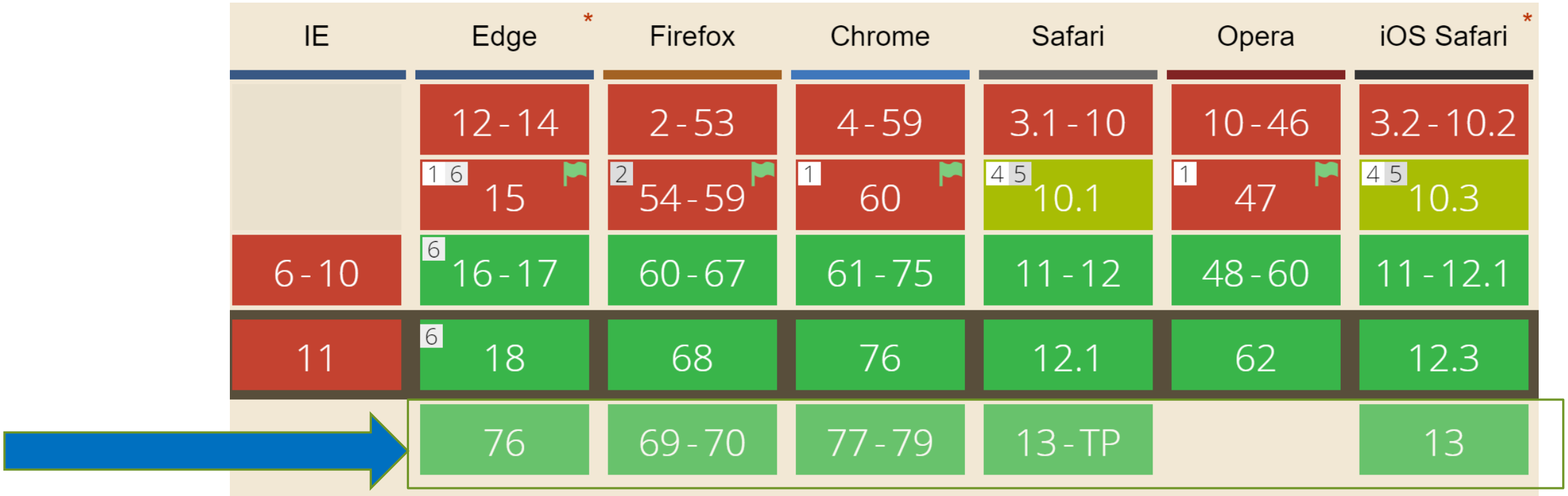


MODULES

Andrew Sheehan
MET CS602

NATIVE BROWSER SUPPORT



IE	Edge *	Firefox	Chrome	Safari	Opera	iOS Safari *
	12 - 14	2 - 53	4 - 59	3.1 - 10	10 - 46	3.2 - 10.2
	16 15	2 54 - 59	1 60	45 10.1	1 47	45 10.3
6 - 10	6 16 - 17	60 - 67	61 - 75	11 - 12	48 - 60	11 - 12.1
11	6 18	68	76	12.1	62	12.3
	76	69 - 70	77 - 79	13 - TP		13

MODULES
USE IN THE BROWSER

```
<script type="module">
  import { converter } from '../convert/converters.js';
  import { MEASUREMENTS } from '../values/units.js';

  const result = converter(2, MEASUREMENTS.binary);
</script>
```

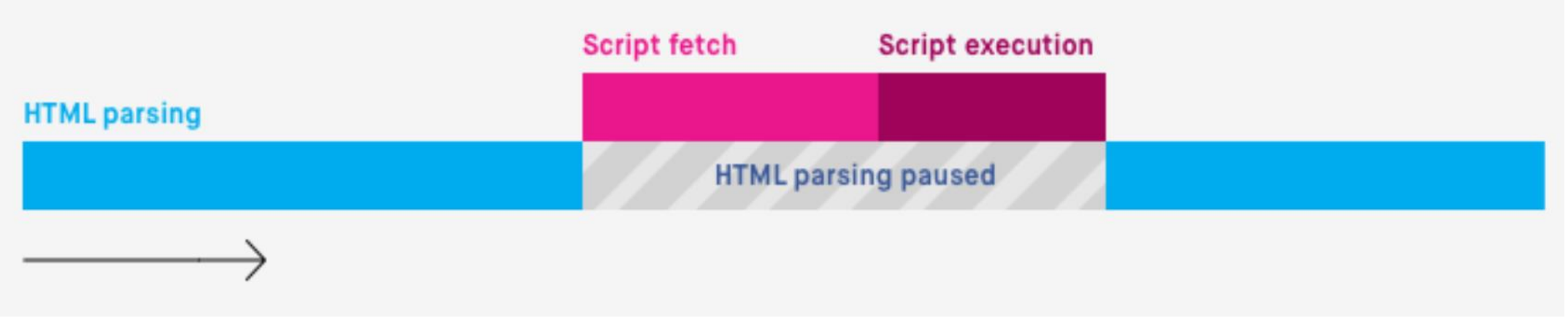
MODULES
LOAD: 1-TIME

Modules only load once (singletons)

```
<script type="module" src="1.mjs"></script>
<script type="module" src="1.mjs"></script>
<script type="module">
  import "./1.mjs";
</script>

<!-- Whereas classic scripts execute multiple times -->
<script src="2.js"></script>
<script src="2.js"></script>
```

MODULES
DEFER (BY DEFAULT)

When a <script> is loaded, whether internal or external, it stops HTML parsing to fetch and run the script.



MODULES
WHAT DOES DEFER DO?

The defer attribute tells the browser to only execute the script once the HTML document has been fully parsed.

```
<script defer src="script.js">
```

MODULES
BACKWARD COMPATIBILITY

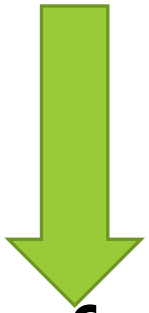
```
<script type="module">
  import { converter } from './converters.js';
  import { MEASUREMENTS } from '../values/units.js';

  const result = converter(2, MEASUREMENTS.binary);
</script>
<script nomodule src="fallback.js"></script>
```

MODULES
BACKWARD COMPATIBILITY

```
<script nomodule src="fallback.js"></script>
```

Any browser that understands modules (and loads) will ignore the nomodule script

MODULES
DEFER (BY DEFAULT)



This is implied. No need to code it.

```
<script defer type="module">
 import { converter } from './converters.js';
 import { MEASUREMENTS } from '../values/units.js';
</script>
```

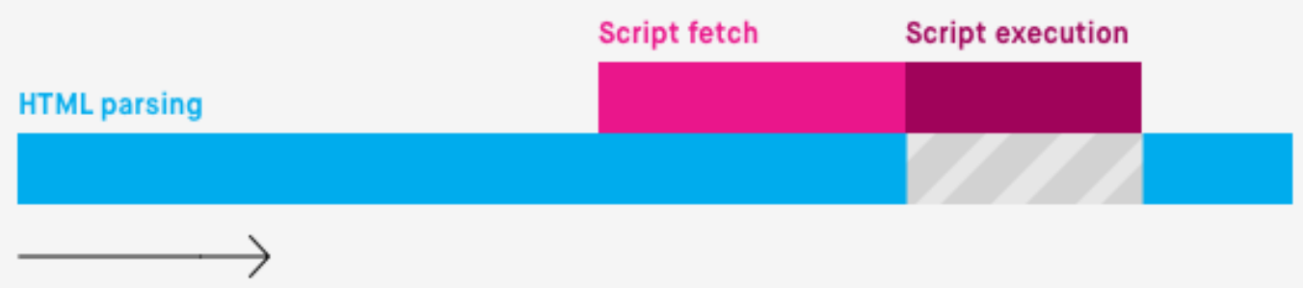
MODULES
ASYNC

```
<script async type="module">
  import { converter } from './converters.js';
  import { MEASUREMENTS } from '../values/units.js';
</script>
```

MODULES
ASYNC

The async attribute is used to indicate to the browser that the script file can be executed asynchronously.

The HTML parser does not need to pause at the point it reaches the script tag to fetch and execute



MODULES
ASYNC

```
<!-- This executes as soon as its imports have fetched -->
<script async type="module">
  import {addTextToBody} from './utils.mjs';

  addTextToBody('Inline module executed.');
</script>

<!-- This executes as soon as it & its imports have fetched -->
<script async type="module" src="1.mjs"></script>
```

MODULES EXAMPLE

render.js

```
export const name = 'square';

export function draw(ctx, length, x, y, color) {
  ctx.fillStyle = color;
  ctx.fillRect(x, y, length, length);

  return {
    length: length,
    x: x,
    y: y,
    color: color
  };
}
```

MODULES EXAMPLE

render.js

```
export const name = 'square';

export function draw(ctx, length, x, y, color) {
  ctx.fillStyle = color;
  ctx.fillRect(x, y, length, length);

  return {
    length: length,
    x: x,
    y: y,
    color: color
  };
}
```

Or you can do this

```
export { name, draw };
```

MODULES
IMPORTING YOUR MODULE

```
<script type="module">
  import { draw, name } from './render.js';
</script>
```

MODULES
USE A WEB SERVER

Cannot use file://

Meaning, you need a web server on your workstation to test.

MODULES
STRICT MODE

Modules by nature are always running in strict mode.

“use strict”

MODULES
RENAMING ON IMPORT

```
<script type="module" >
  import { converter as conv } from './converters.js';
</script>
```

MODULES
RENAMING ON EXPORT

```
export {
    strangeJump as badHop,
    doublePlay as aroundTheHorn,
    homeRun as HOMAH
};
```

MODULES
DEFAULT EXPORT (EXAMPLE)

```
let address = {
   fullAddress: "",
   dateCreated: null
}

export { address as default };
```

MODULES
EXPORTING

There are three types of exports

1. Named Exports (Zero or more exports per module)
2. Default Exports (One per module)
3. Hybrid Exports

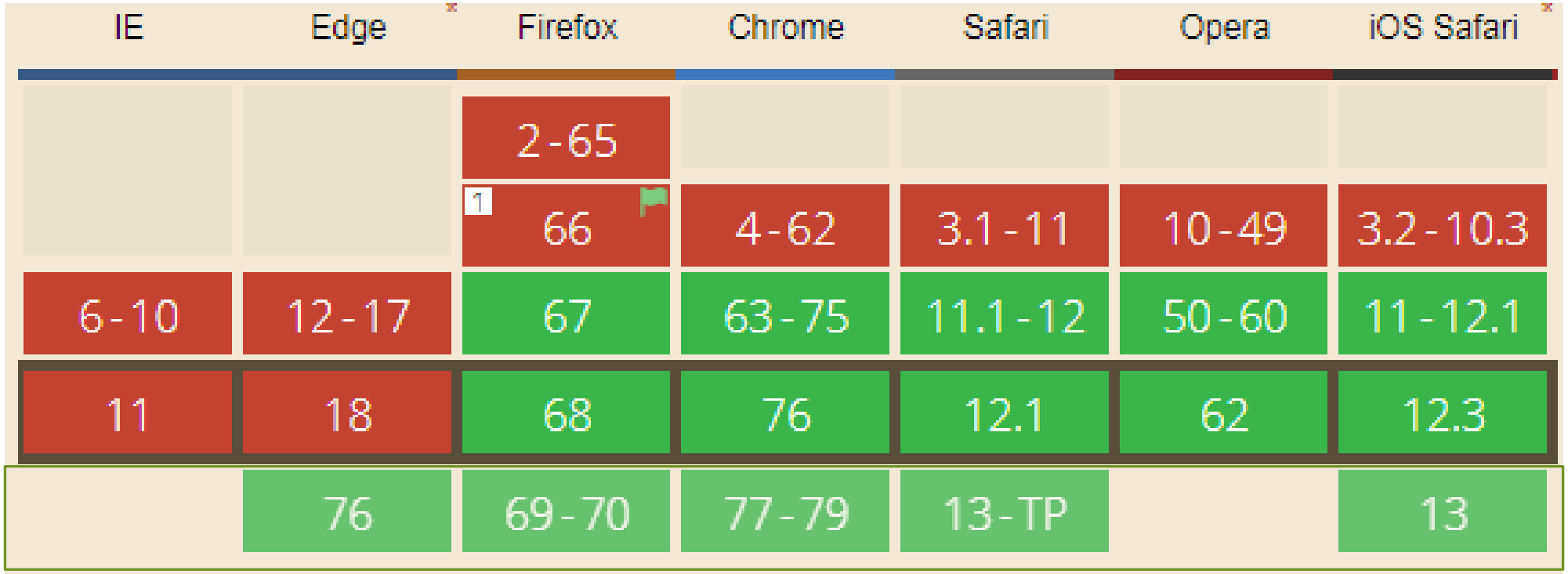
MODULES
EXPORTING

```
// Exporting individual features
export let name1, name2, …, nameN; // also var, const
export let name1 = …, name2 = …, …, nameN; // also var, const
export function functionName(){...}
export class ClassName {...}

// Export list
export { name1, name2, …, nameN };

// Renaming exports
export { variable1 as name1, variable2 as name2, …, nameN };

// Exporting destructured assignments with renaming
export const { name1, name2: bar } = o;

// Default exports
export default expression;
export default function (…) { … } // also class, function*
export default function name1(…) { … } // also class, function*
export { name1 as default, … };

// Aggregating modules
export * from …;
export { name1, name2, …, nameN } from …;
export { import1 as name1, import2 as name2, …, nameN } from …;
export { default } from …;
```

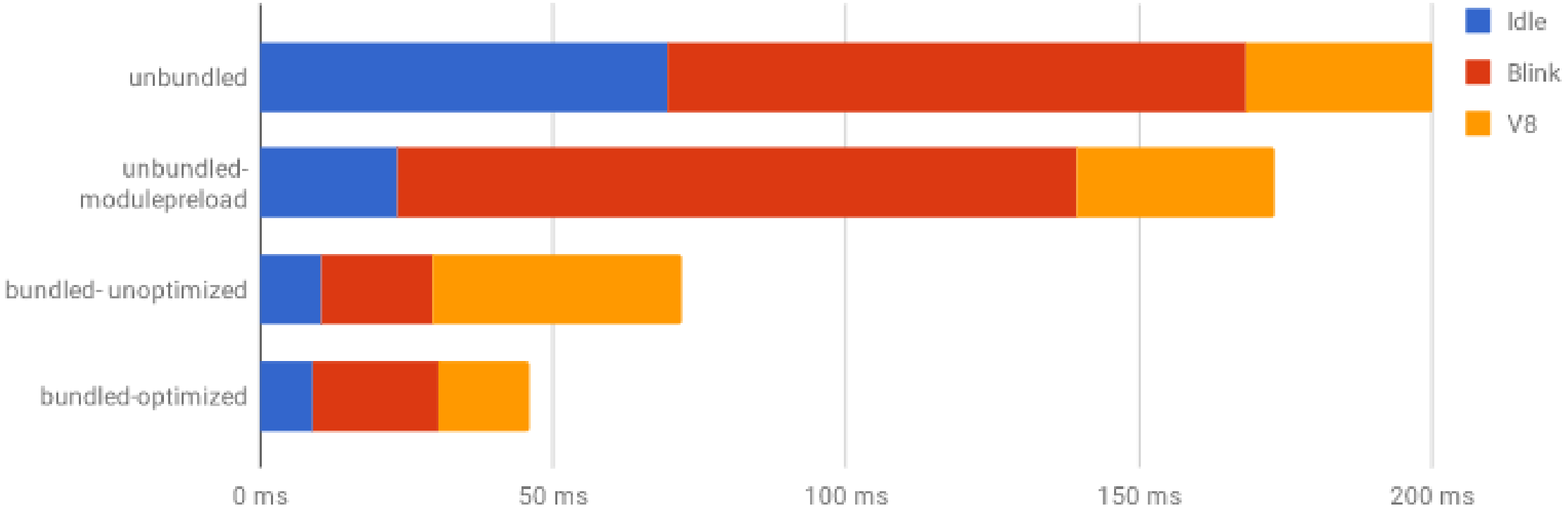
MODULES
DYNAMIC IMPORT

```
<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>
```

MODULES
DYNAMIC IMPORT



MODULES
BETTER PERFORMANCE



MODULES PRELOADING

```
<link rel="modulepreload" href="lib.mjs">
<link rel="modulepreload" href="main.mjs">
<script type="module" src="main.mjs"></script>
<script nomodule src="fallback.js"></script>
```

Without rel="modulepreload", the browser needs to perform multiple HTTP requests to figure out the full dependency tree.

However, if you declare the full list of dependent module scripts with rel="modulepreload", the browser doesn't have to discover these dependencies.