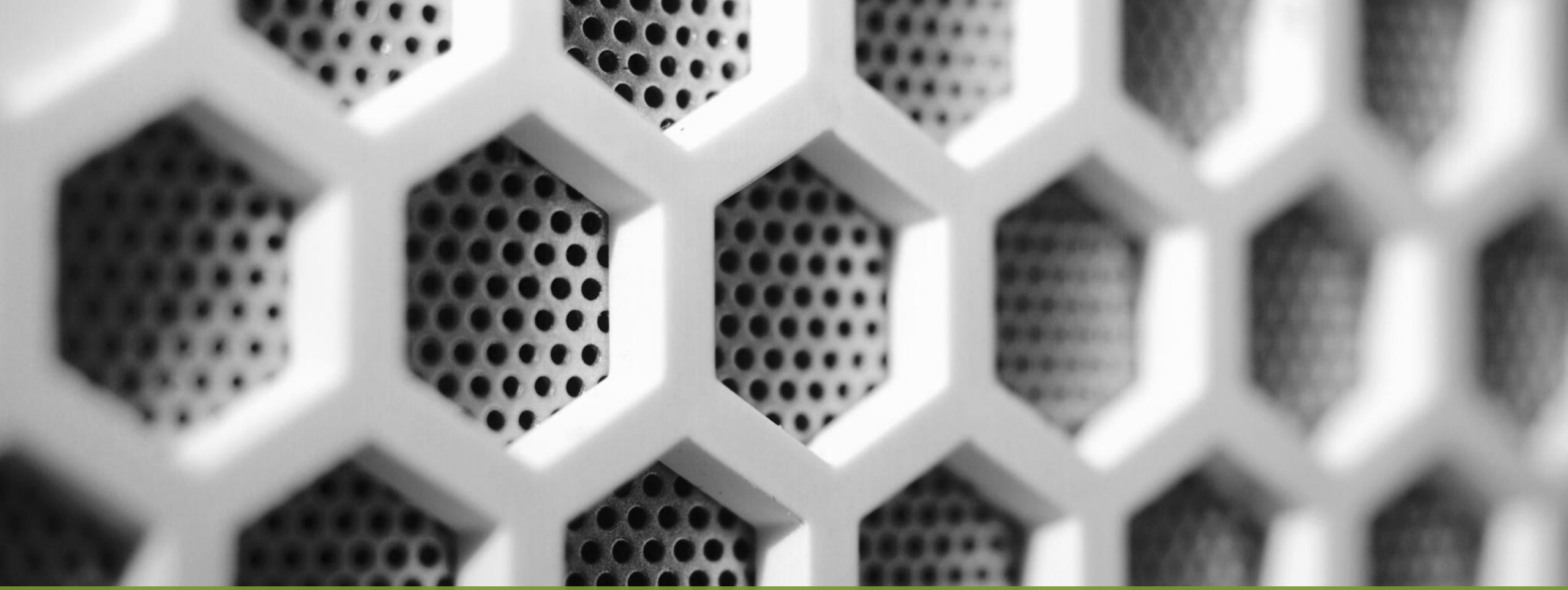
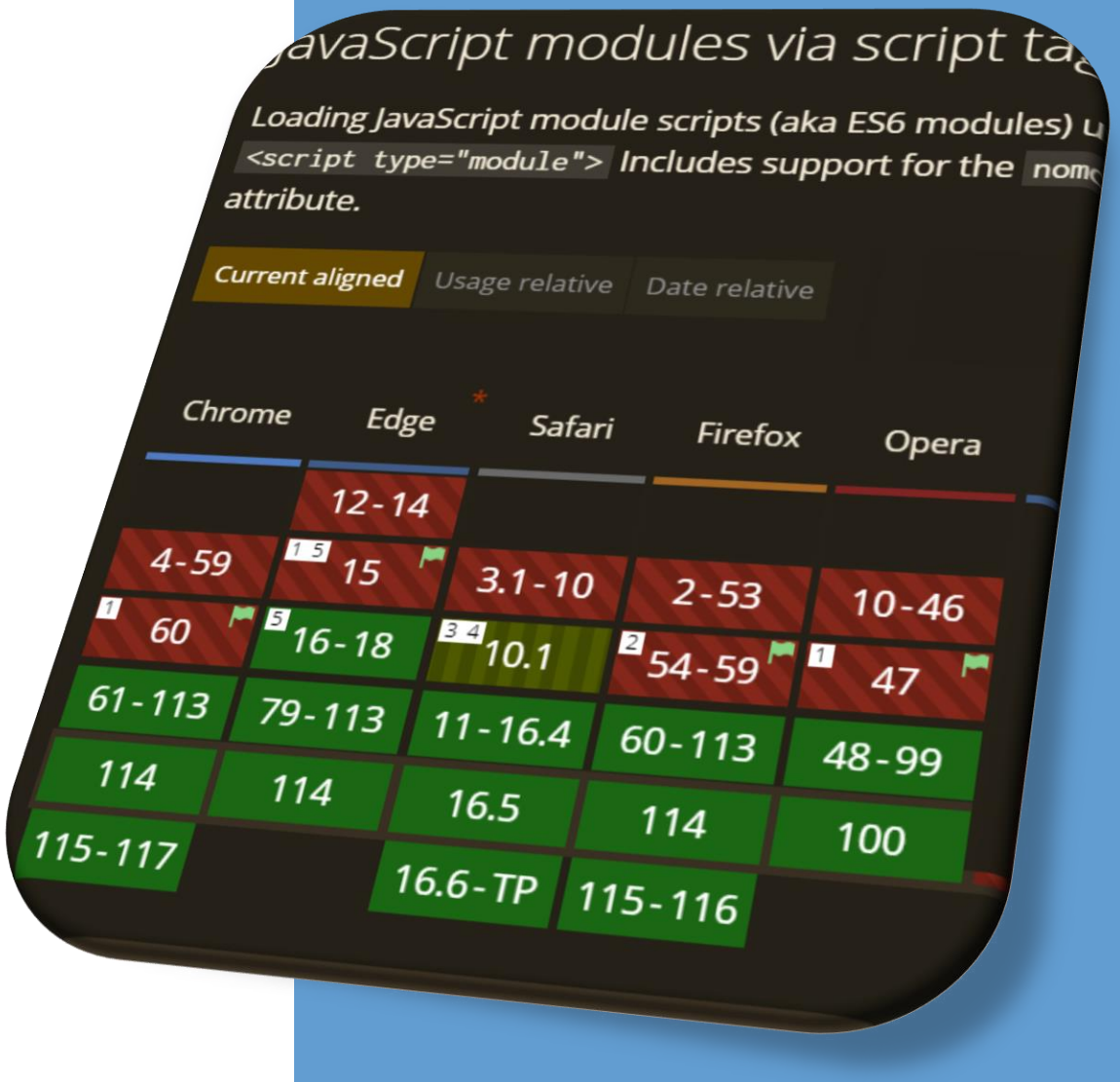


MODULES

Prof Andrew Sheehan

Boston University/MET
Computer Science Dept.

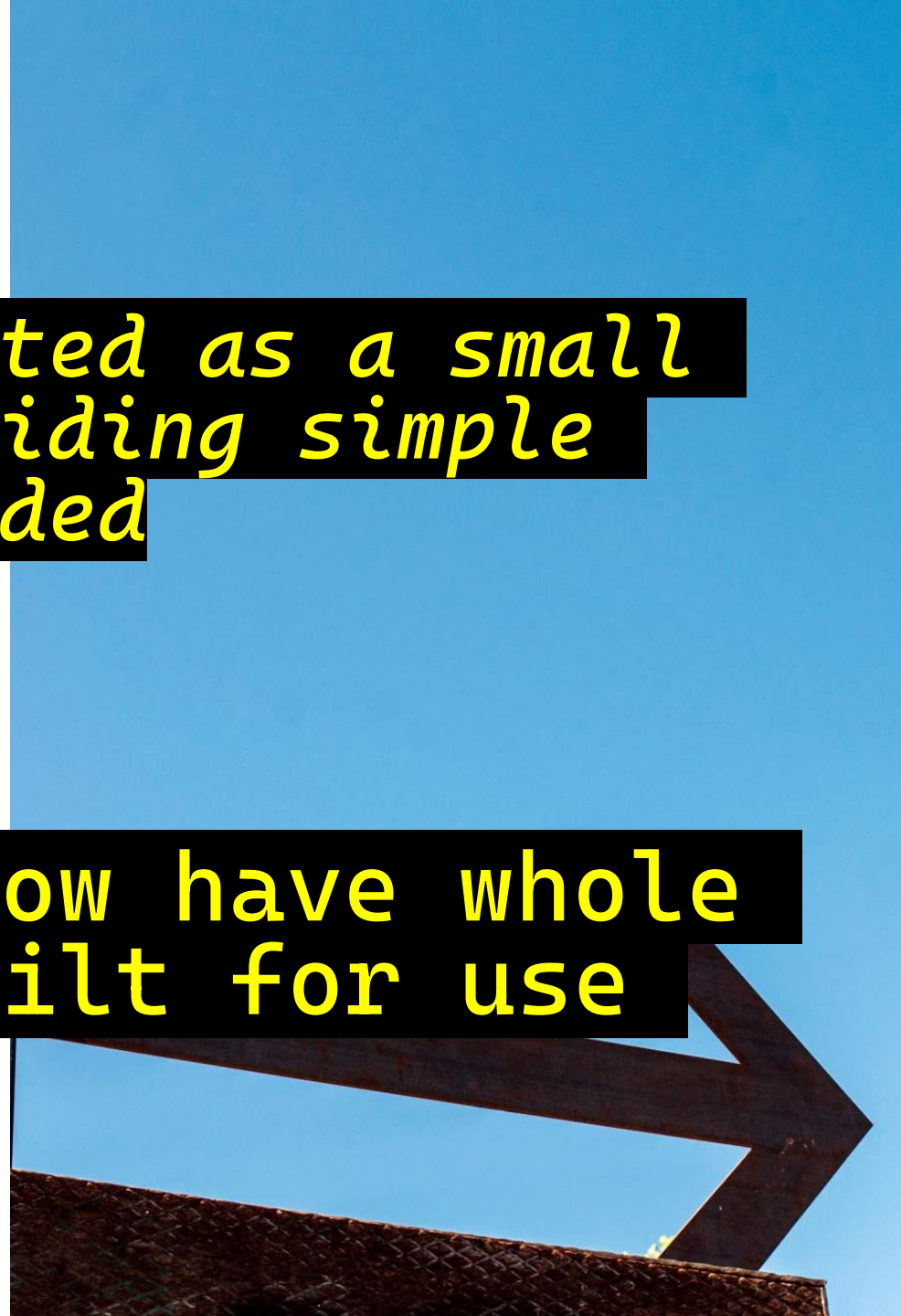


ES6 STANDARD

HISTORICALLY

Javascript programs started as a small part of a web page, providing simple event handling where needed

Fast forward and we now have whole application suites built for use with only browsers



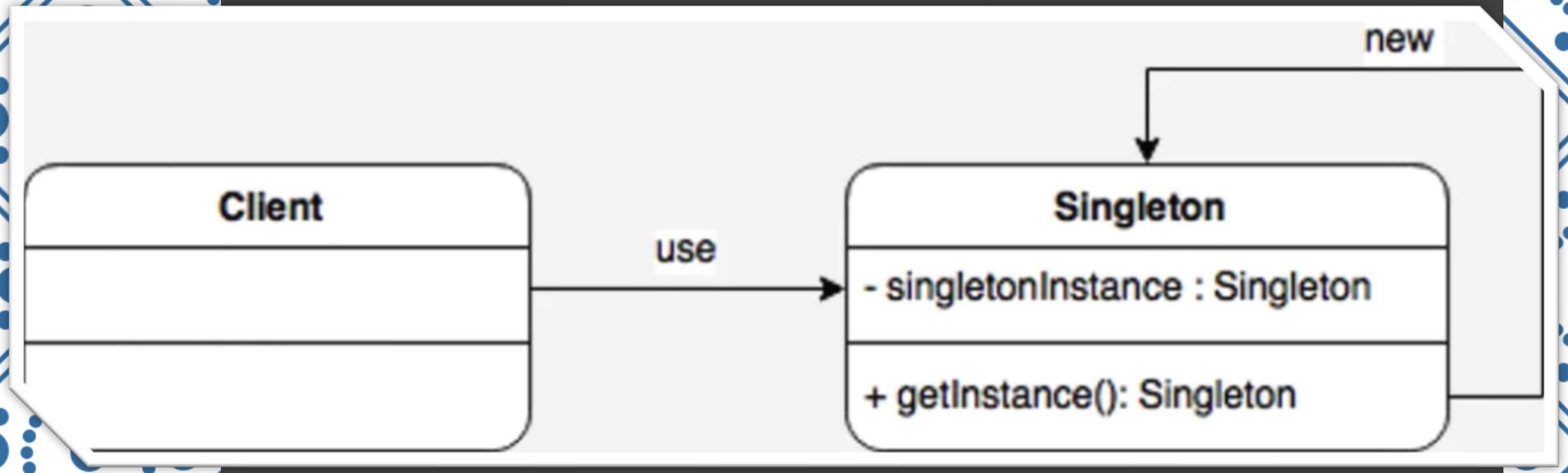
WHAT IS A MODULE?

Encapsulates code. Expose what you need.

Think of modules as packages (java) or namespaces (C#)

SINGLETONS

There is exactly one module per file



Module file names

filenames end with.js or .mjs*

Converter.mjs*

Converter.js

Make sure your web server has a mapping for this MIME Type (“text/javascript”) for .mjs files

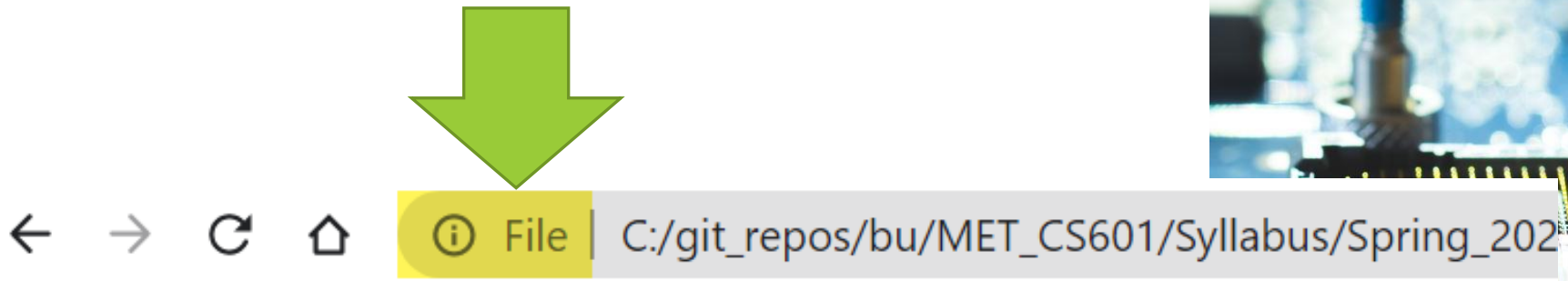
.mjs vs .js

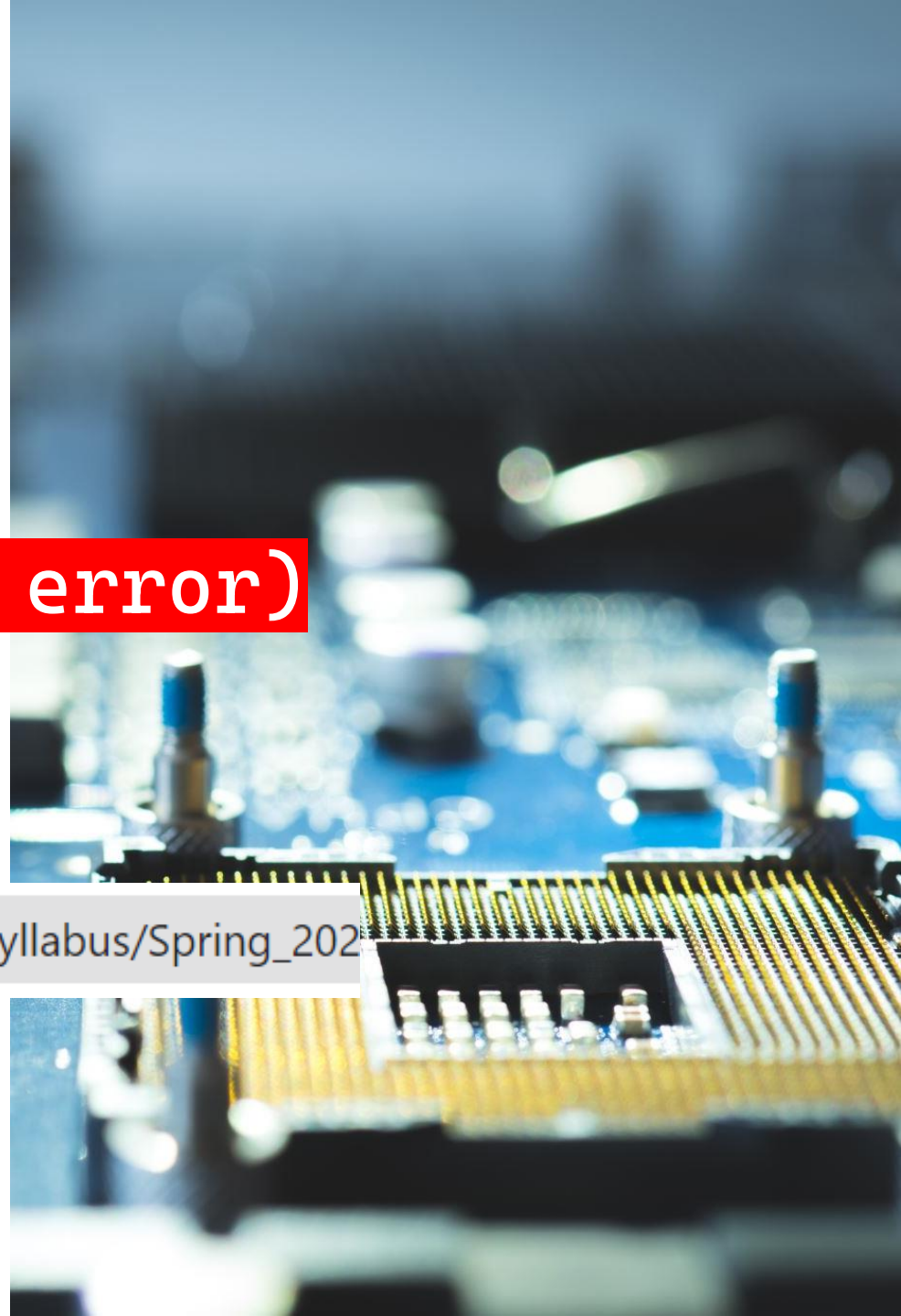
POINT #1: When you see .mjs, you know it is a module

POINT #2: It ensures that your module files are parsed as a module by runtimes such as Node and build tools such as Babel

MUST USE A WEB SERVER

You cannot use the file:// protocol (CORS error)





STRICT MODE

Modules by design are always running in strict mode.

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

  const result = kelvin(2, MEASUREMENTS.atomic);
</script>
```

BASIC USE

- *All function and variables declared within a module are local to that module.*
- *Modules can use other modules.*
- *Modules are singletons.* Import it a million times – *It will only be loaded once* – over all your other modules.

RULES
SYNTAX

NEED BACKWARD COMPATIBILITY?

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

Any browser that understands modules will ignore the nomodule script

Module scripts do not block HTML processing. They load in parallel.

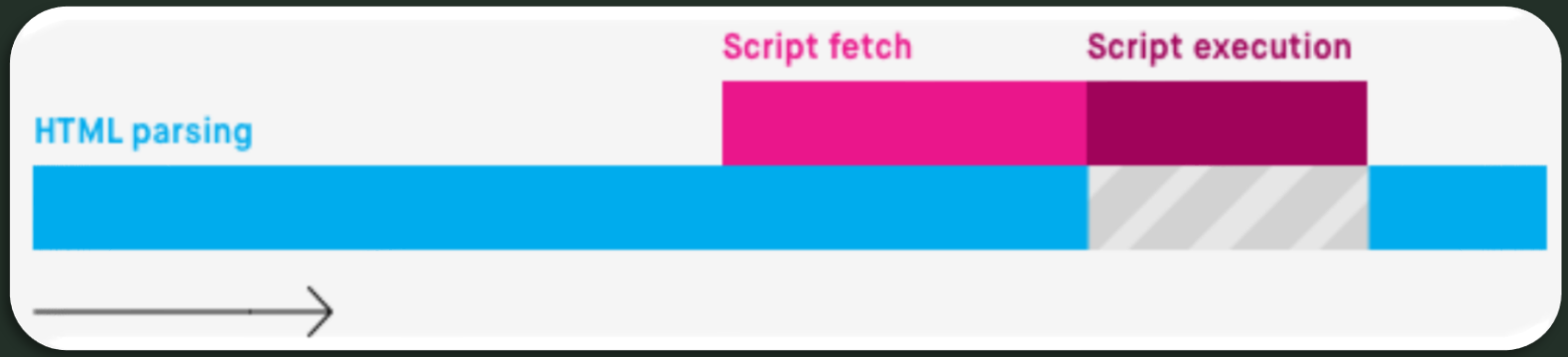
Defer: Module scripts wait until the HTML is loaded and then [the module] will execute.

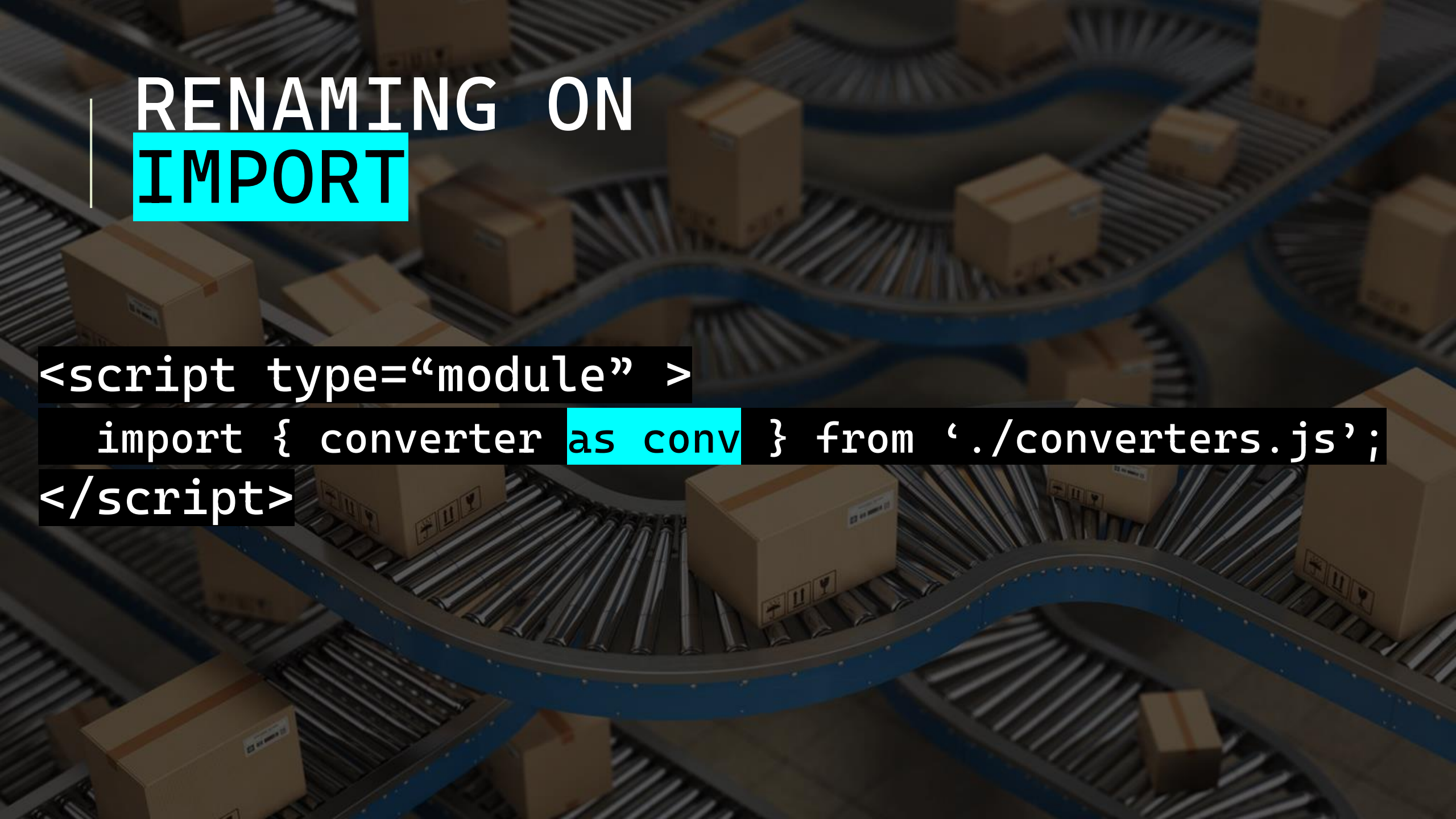
DEFER MODE BY DEFAULT

ASYNC

```
<script async type="module">
```

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





RENAMING ON IMPORT

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

HOW TO IMPORT EVERYTHING

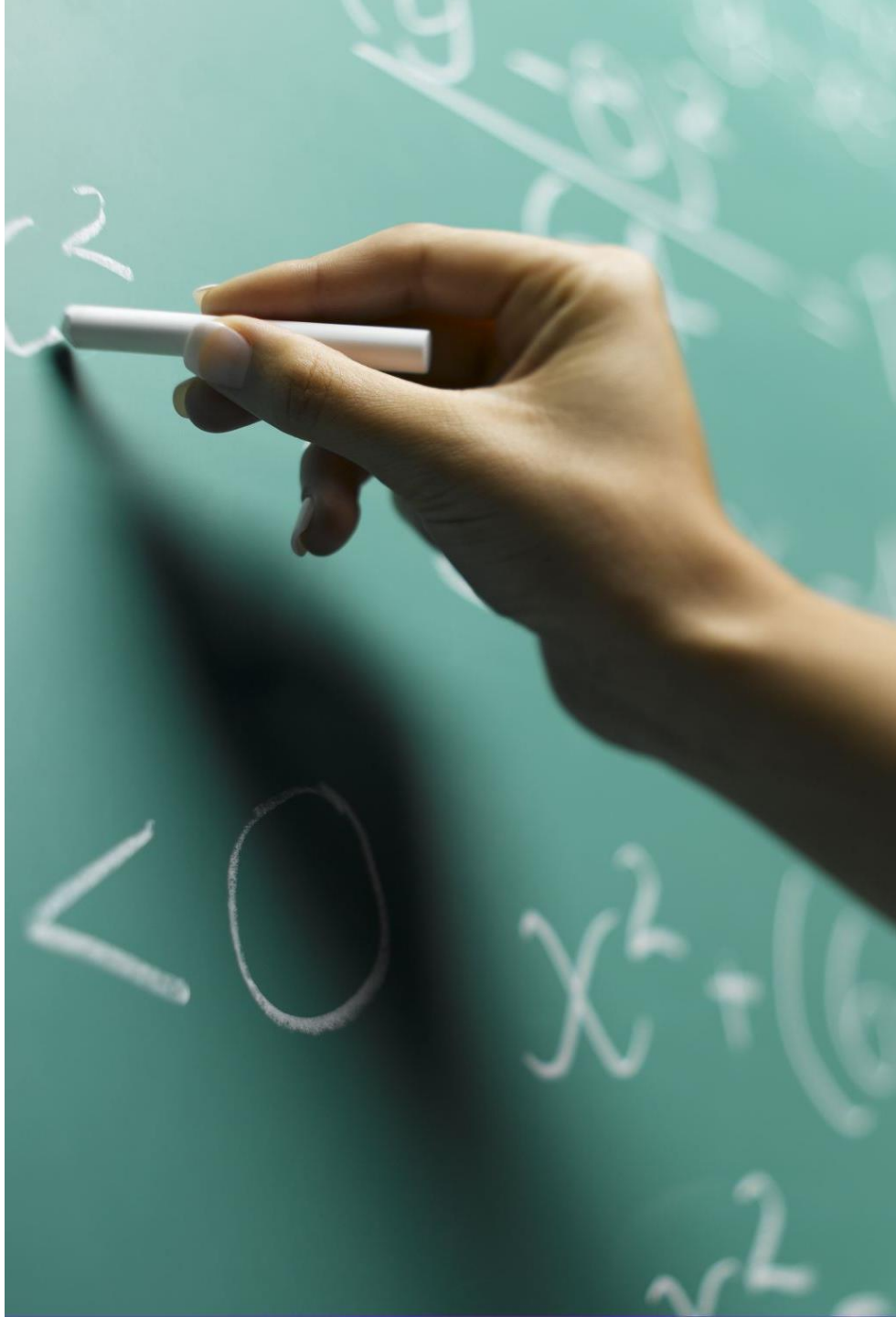
```
<script type="module">
  import * as CVT from
    '../converters/convert.js';
</script>
```

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>
```

2 WAYS TO EXPORT

1. Named
2. Default



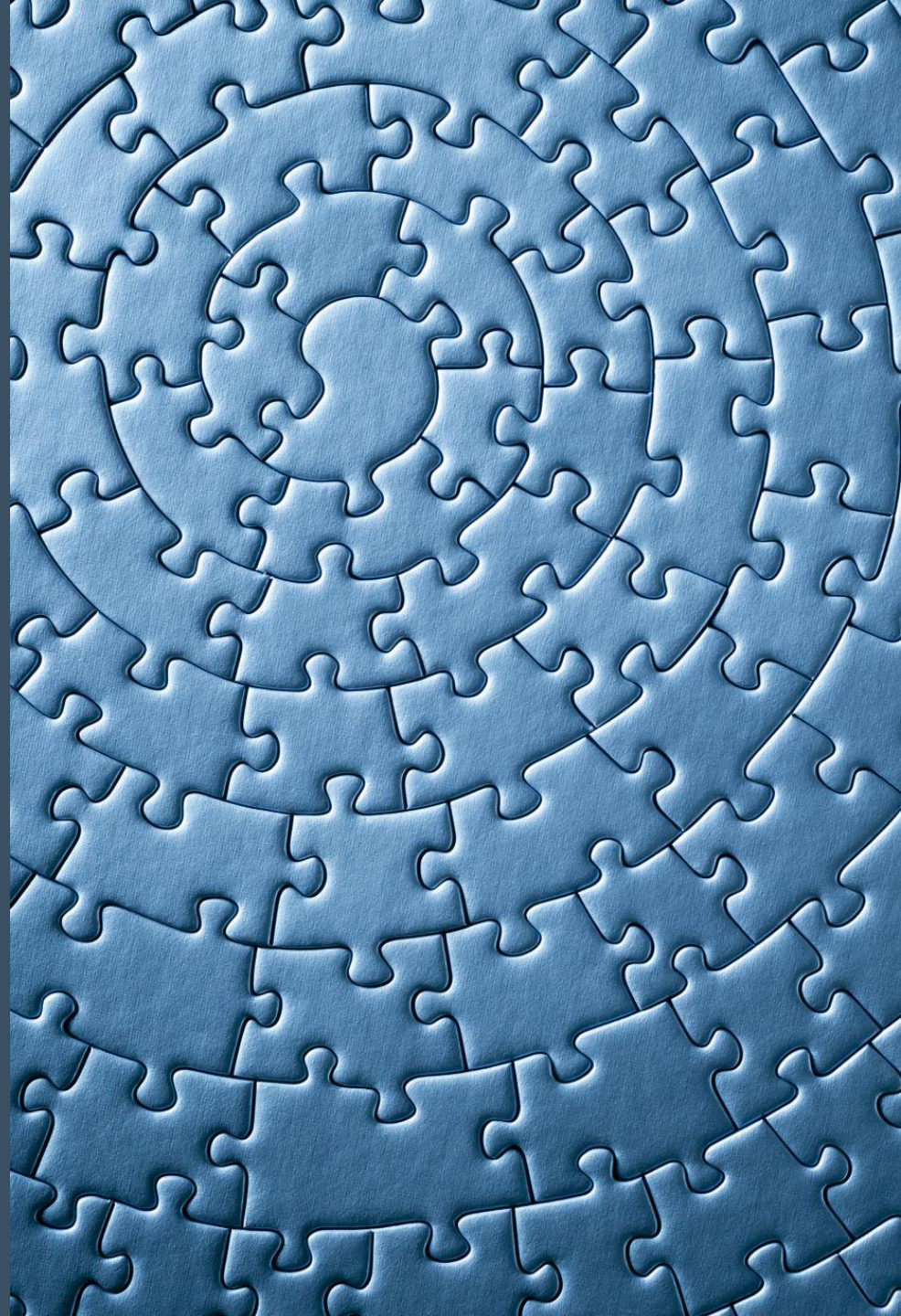
EXAMPLE

You could have several named exports

Can only use one as the **default export**

```
//------ lib.js ------
export const sqrt = Math.sqrt;
export function square(x) {
    return x * x;
}
export function diag(x, y) {
    return sqrt(square(x) + square(y));
}

//------ main.js ------
import { square, diag } from 'lib';
console.log(square(11)); // 121
console.log(diag(4, 3)); // 5
```



RENAMING ON EXPORT

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

EXPORT DECLARATION EXAMPLES

```
export let name1, name2;
export const name1 = 1, name2 = 2;
export function functionName() { /* … */ }
export class ClassName { /* … */ }
export const { name1, name2: bar } = o;
export const [ name1, name2 ] = array;
```

EXPORT MAPPING EXAMPLES

```
export { name1, …};
export {
  variable1 as name1,
  variable2 as name2, …
};
export { variable1 as ‘string name’, … };
export { name1 as default, …};
```

IMPORT MAPS RULES

Only 1 importmap per HTML document

Must be declared before any use of script elements

Must be formatted using JSON

IMPORT MAPS

```
<script type="importmap">
  {
    "imports": {
      "shapes": "./shapes/square.js",
      "shapes/square": "./modules/shapes/square.js",
      "https://example.com/shapes/": "/shapes/square/",
      "https://example.com/shapes/square.js": "./shapes/square.js",
      "../shapes/square": "./shapes/square.js",
    }
  }
</script>
```

```
import { name as squareNameOne } from "shapes";
import { name as squareNameTwo } from "shapes/square";
```

See more:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules

With this map you can now use the property names above as module specifiers.
If there is no trailing forward slash on the module specifier key then the whole module specifier key is matched and substituted

IMPORT MAPS EXAMPLE