Navigating the Wild
Seas of ES6 Modules
August 2015

There be dragons!

The Node Dragon

Sonnet 130

The Dragon of Syntax

The Browser Dragon

The Dragon of Syntax

Creating a module

```
export function doIt() {}

export var meaningOfLife = 42;

function doThat() {}
```

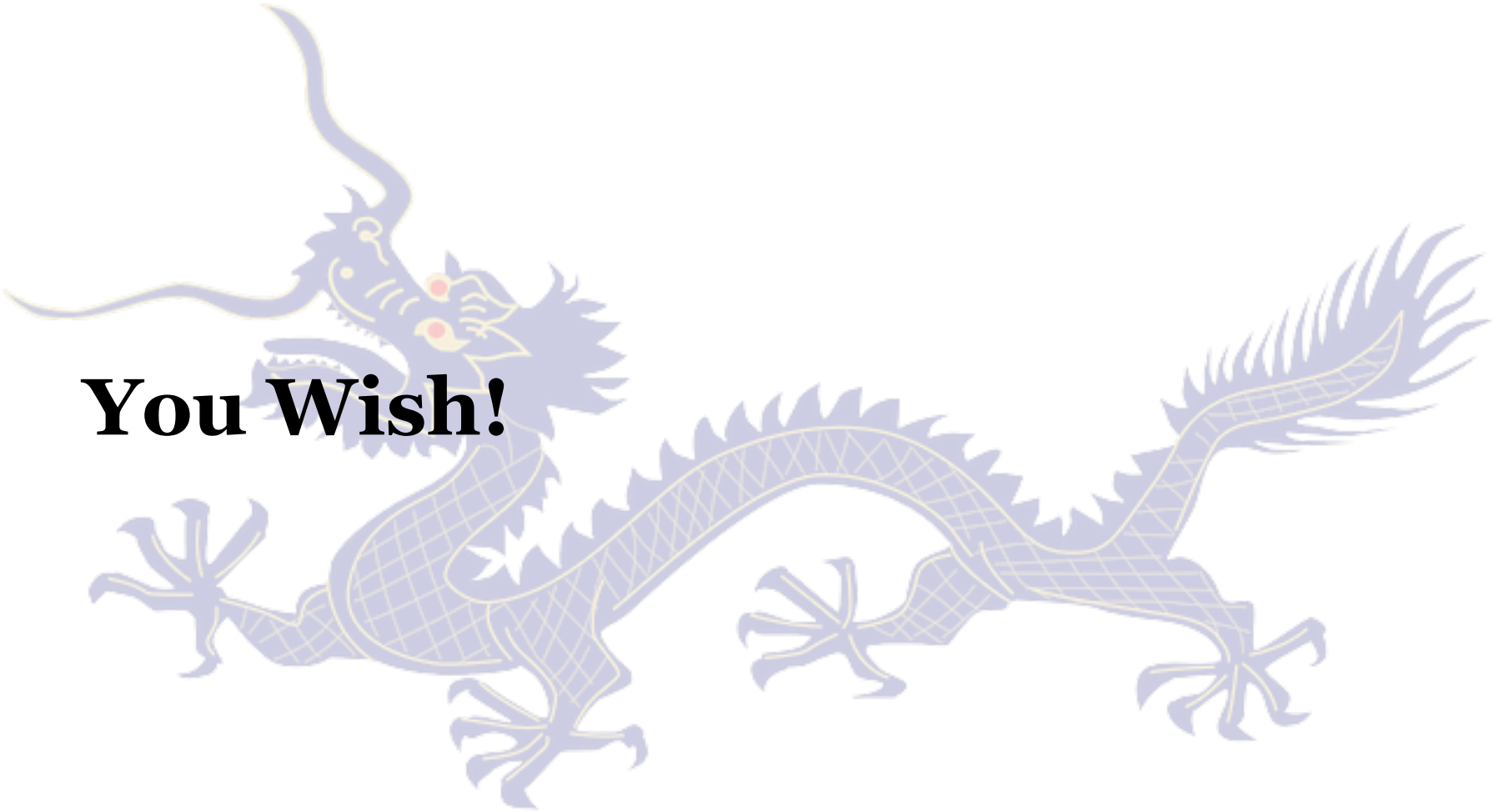
Using a module

a-module.js

```
export function doIt() {}

r meaningOfLife = 42

tion doThat() {}
```

Simple, right?

```
import {d          aningOfLife}
        from 'a-module';
```



You Wish!

TIMTOWTDI

There Is More Than One Way To Do It

```
function doIt() {}
var aVar = 42;
function doThat() {...}

export {doIt,
        aVar as meaningOfLife};
```

There Is More Than One Way To Do It

```
import * as lib from 'a-module';


lib.doIt();
lib.meaningOfLife;
```

AAAAARRRGGGGGHHH!!!

The dreaded `default`

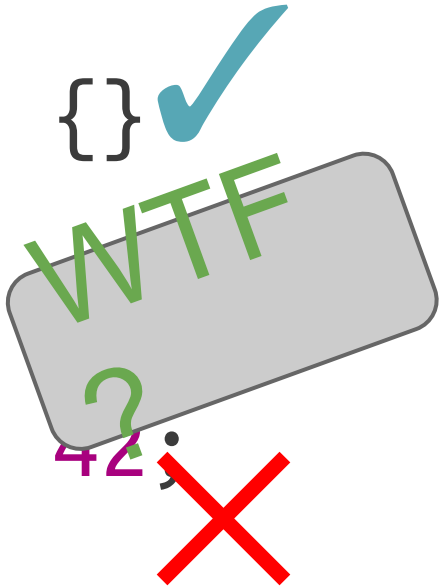
a-module.js

```
export default 42;
```

```
import meaningOfLife
        from 'a-module';
```

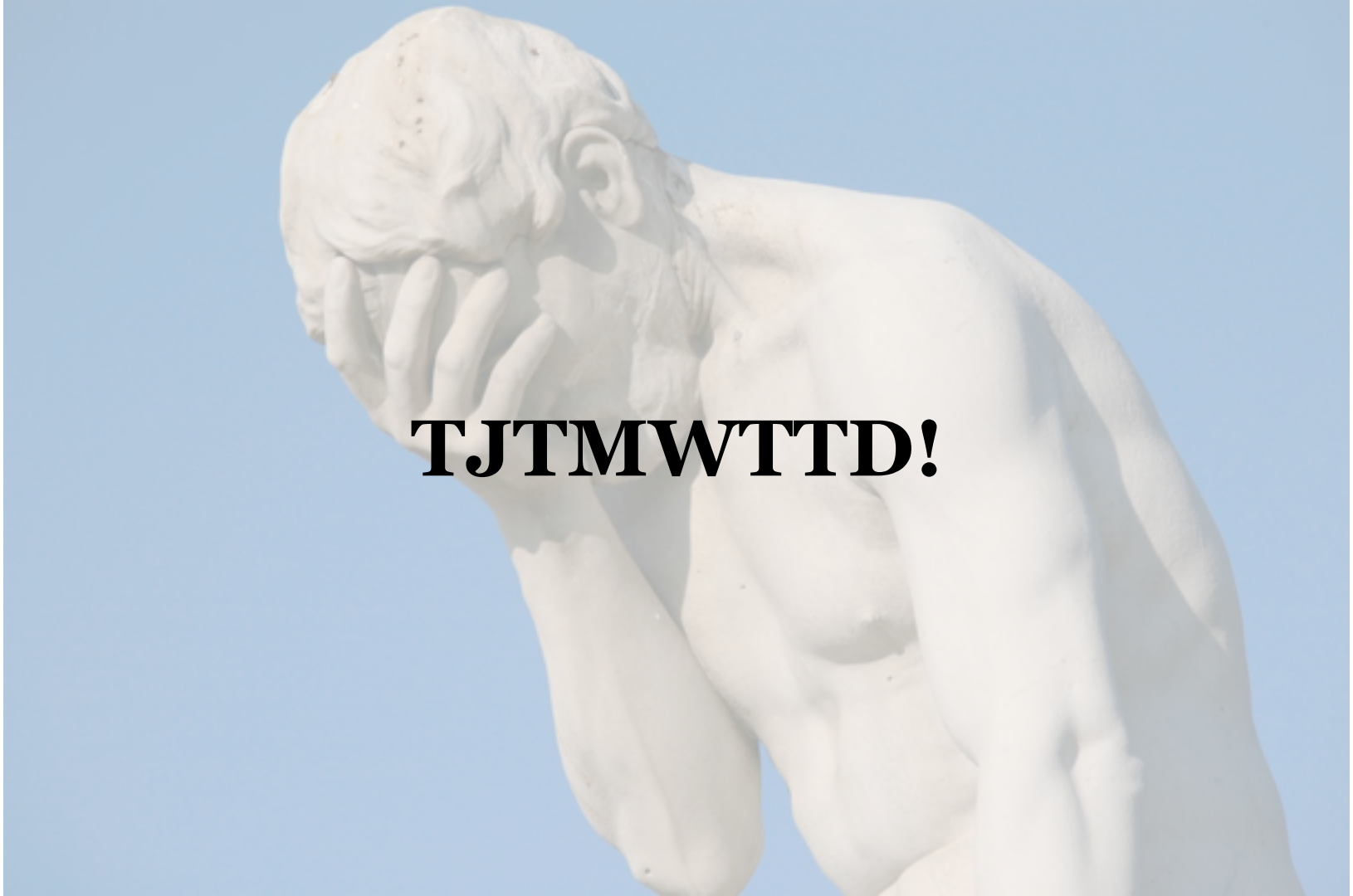
Confusion will be my epitaph

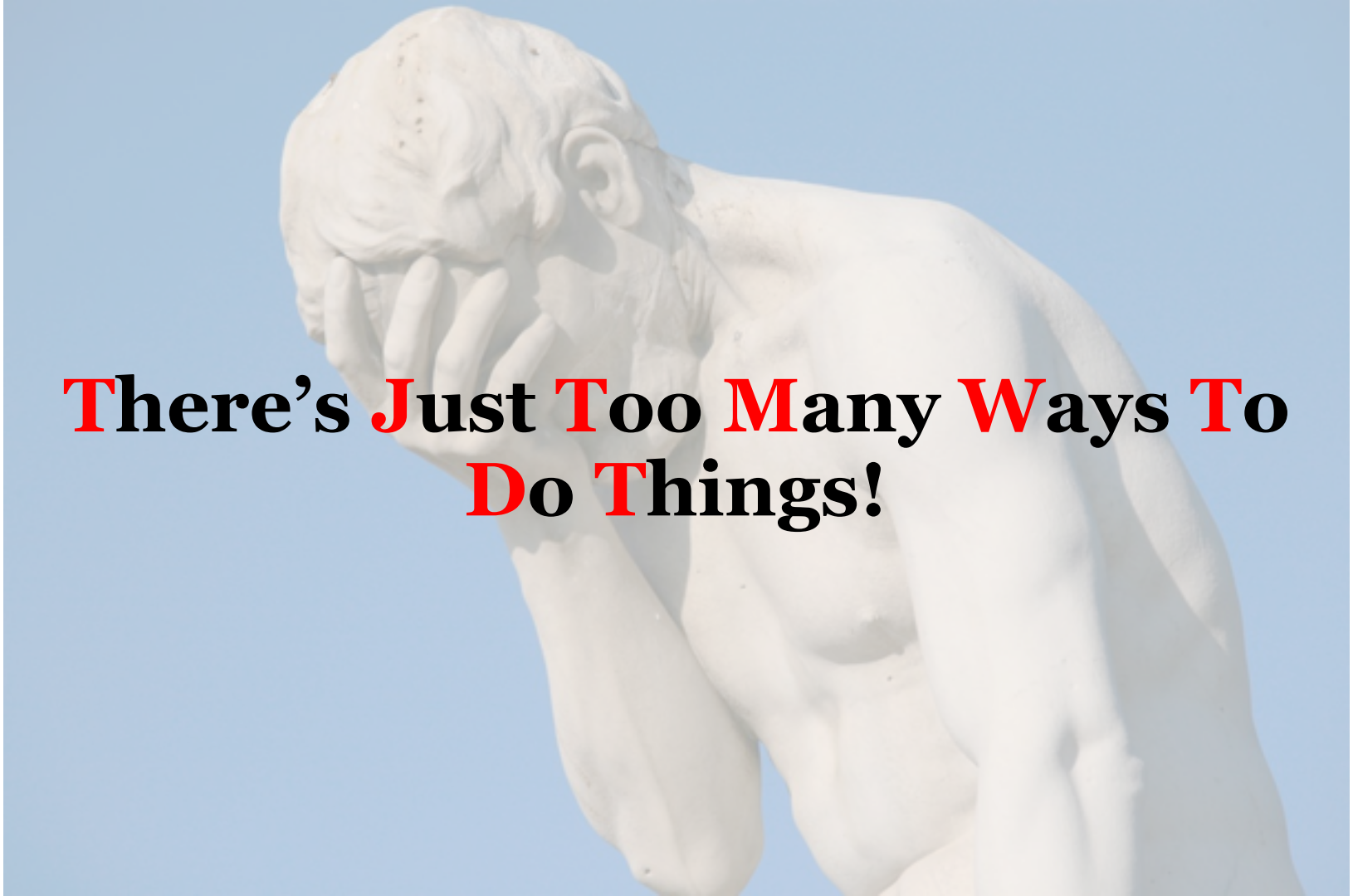
```
export default
       function doIt() {}

export default var
       meaningOfLife = 42;
```



`default` is just another import

```
import orOther, {something}
          from 'a-module';
import {something}, orOther
          from 'a-module';
import {something, default as orOther}
          from 'a-module';
```


TJTMWTTD!


There's Just Too Many Ways To
Do Things!

Why Did This Happen?

"A camel is a horse designed by a committee."

So?

Do we abandon ship?

No!

Prefer `default` over specific exports

```
export default function
            factory() {}
```

or...

```
export default {
    doIt() {}
    meaningOfLife: 42
}
```

Prefer `const` var bindings

```
export const meaningOfLife = 42;
```

The Node Dragon

Sonnet 130

The Dragon of Syntax

The Browser Dragon

The Node Dragon

How do we use ES6 modules in Node?

Why would we do that?

Node has a *really* good module system

Node module

a-module.js

```
exports.doIt = function() {}
exports.meaningOfLife = 42;
```

So Why?

```
const {doIt, meaningOfLife} =
        require('a-module');
```

Two words: isomorphic code

So How?

Native ES6 modules: ongoing work

Till then…

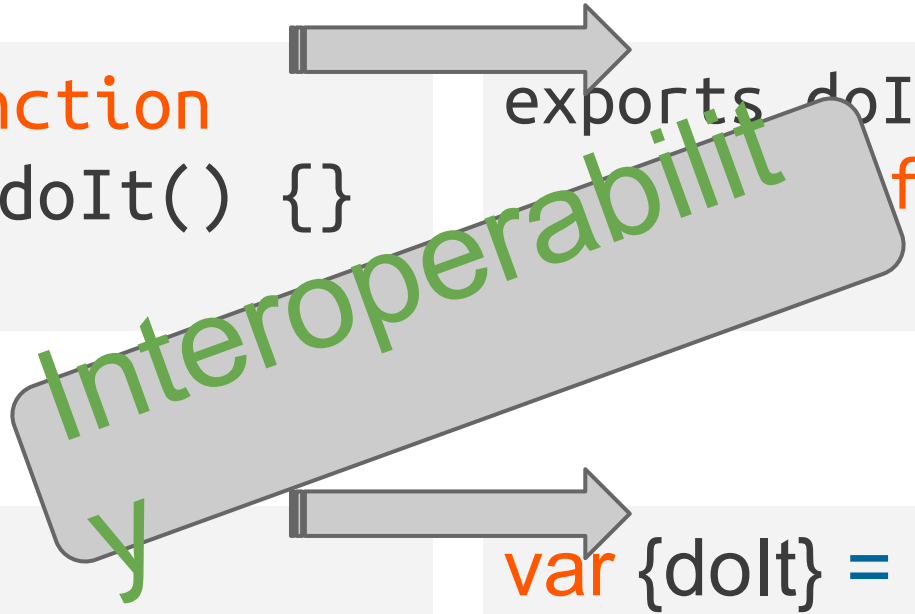
Babel!

```
require("babel/register");
```

Babel!

```
export function
          doIt() {}
```

```
exports doIt =
          function()
```



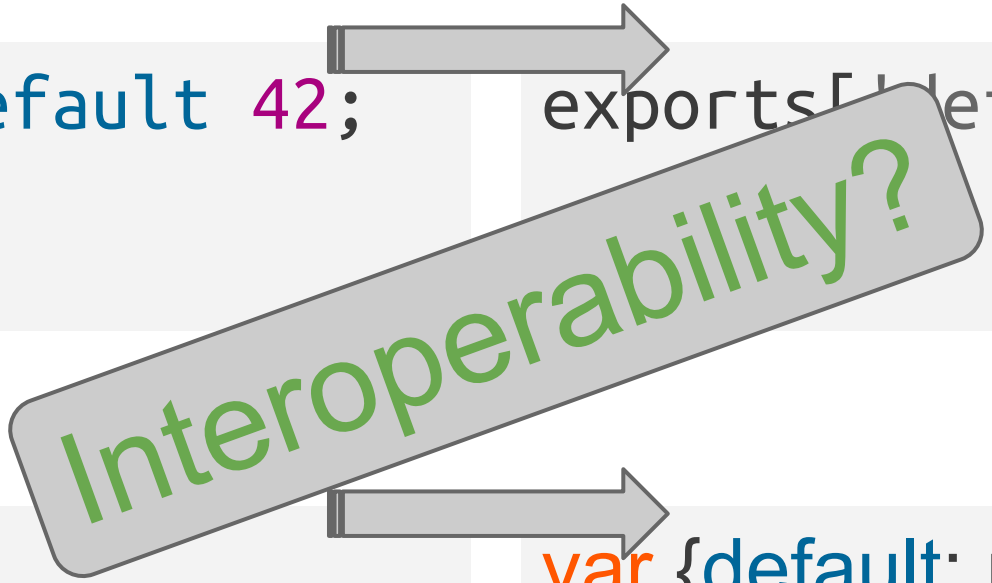

```
import {doIt}
          from 'a-
module'
```

```
var {doIt} =
 require('a-module')
```

default

```
export default 42;
```

```
exports['default'] =
```




```
import mol
                from 'a-
module'
```

```
var {default: mol} =
 require('a-module')
```

And this?

```
var aModule =
process.arch === 'wi32' ?
        requir dule-win32') :
        requir module')

//...
```

Huh? No dynamic loading of modules?

There is!

```
var moduleName =
      process.arch === 'win32' ?
            'a-module-win32' : 'a-module’

System.import(‘a-module-win32’).then(
          aModule => { /* ... */})
```

But it's async!

the node people and ES6 people are currently discussing how to resolve this

So?

Do we abandon ship?

No!

Do this

- Use `import`/`export` when you can
- Fall back to `require` if you must

The Node Dragon

Sonnet 130

The Dragon of Syntax

The Browser Dragon

The Browser Dragon

The fiercest dragon of them all

A whole talk unto itself

But maybe a glimpse?

This is illegal

```
<script>
    import {doIt} from 'a-module';
  // …
</script>
```

You have to do this

```
<script>
    System.import('a-module').then(
        ({doIt}) =>
            // ...
    );
</script>
```

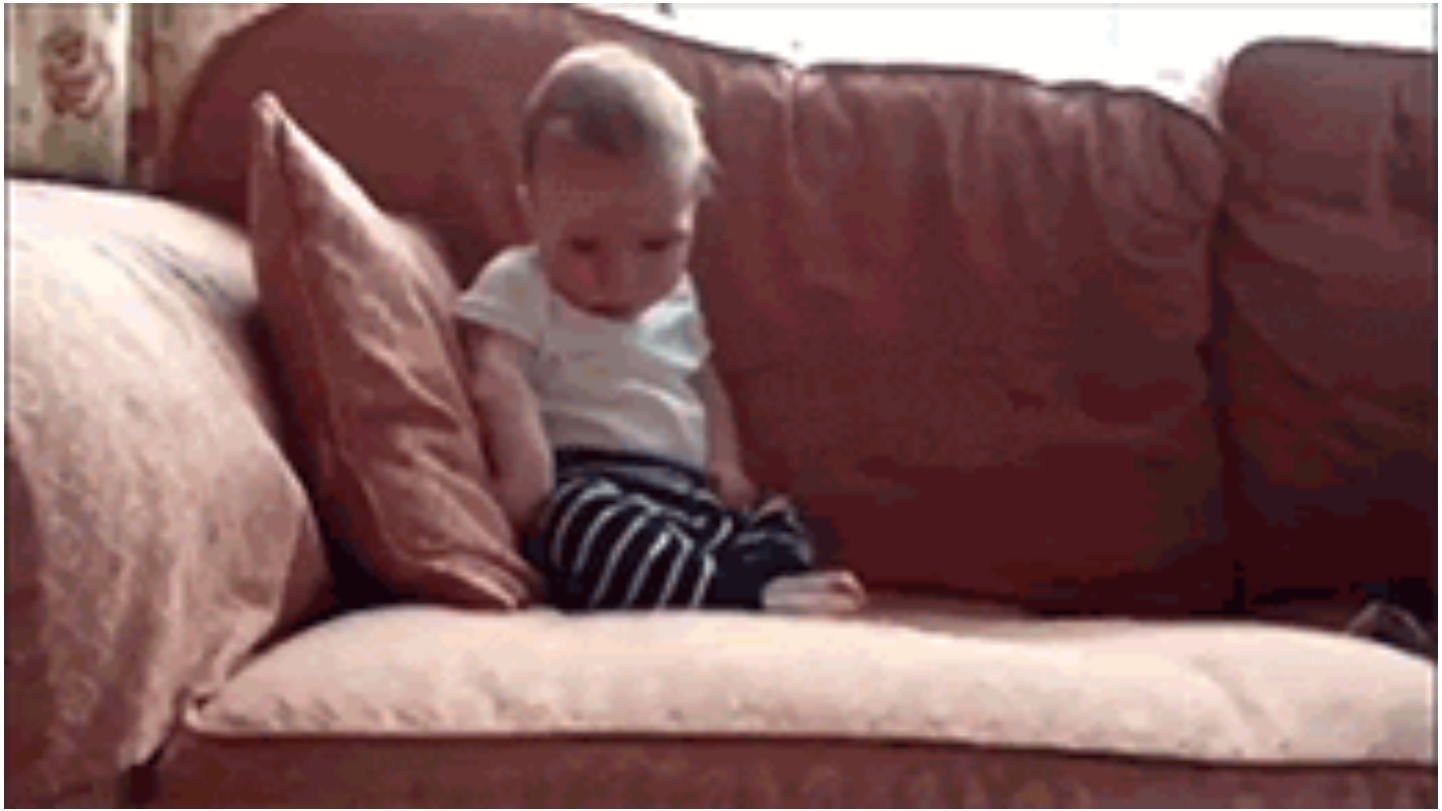
Or…

```
<script type=module>
    import {doIt} from 'a-module';
  // …
</script>
```

Or maybe…

```
<module>
    import {doIt} from 'a-module';
  // …
</module>
```

Who knows? The spec hasn’t been updated in 10 months!



Extra! Extra! Read all about it!

Modules have their own scope

So forget about this

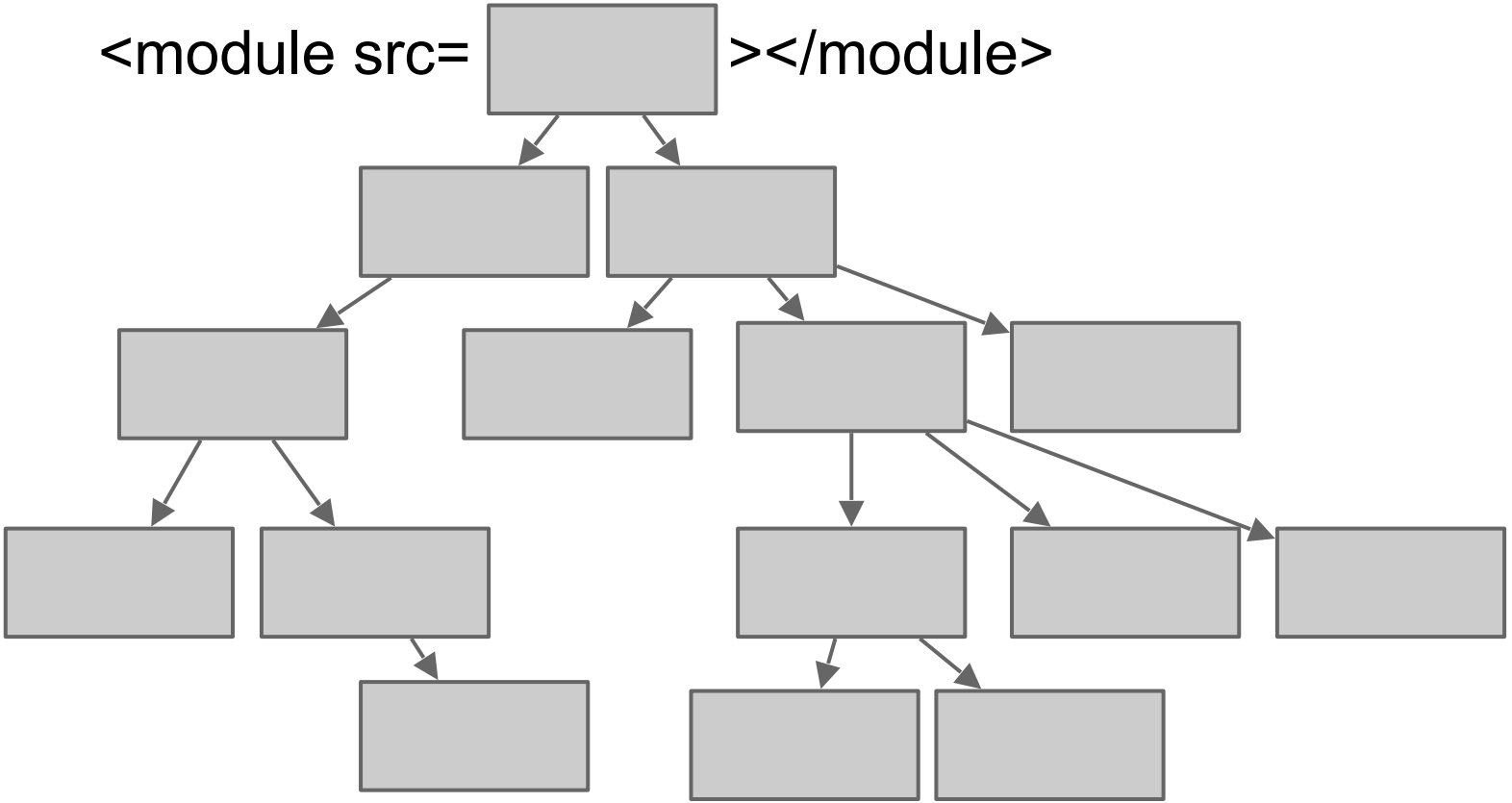
```
<module>
     import {doIt} from 'a-module';
   // …
</module>
<div onclick=doit>...</div>
```

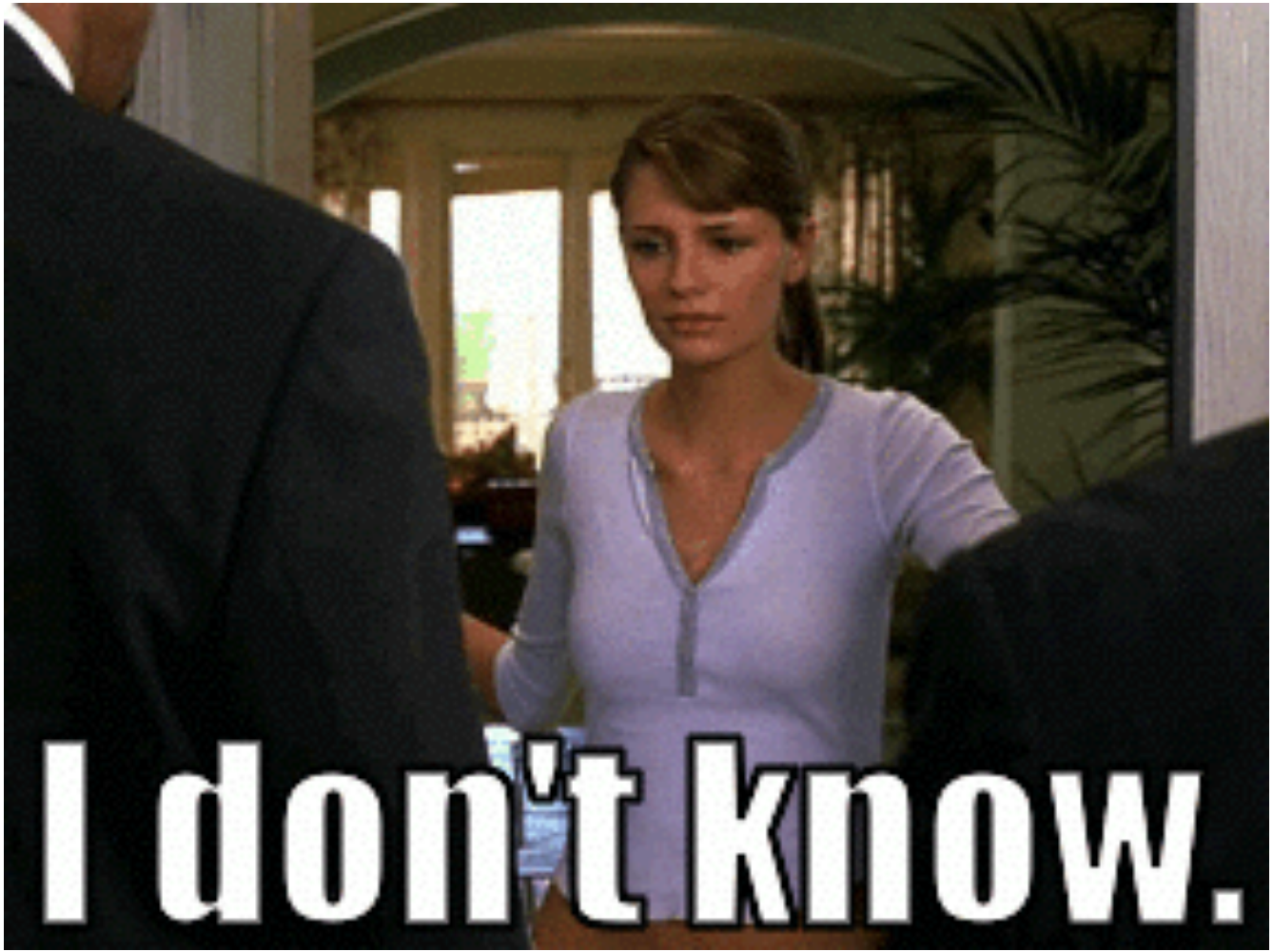
Interoperability with AMD

- e.g. RequireJS?

- Maybe. There's *another* spec, the Loader spec, which is dealing with this
- Work in progress. Not yet close to ready.
- And has interoperability problems with Node

And how would this work efficiently?




I don't know.

So?

Do we abandon ship?

No!

The One-Two Punch

Punch #1

We've already seen it (Babel)

```
export function
       doIt() {}
```

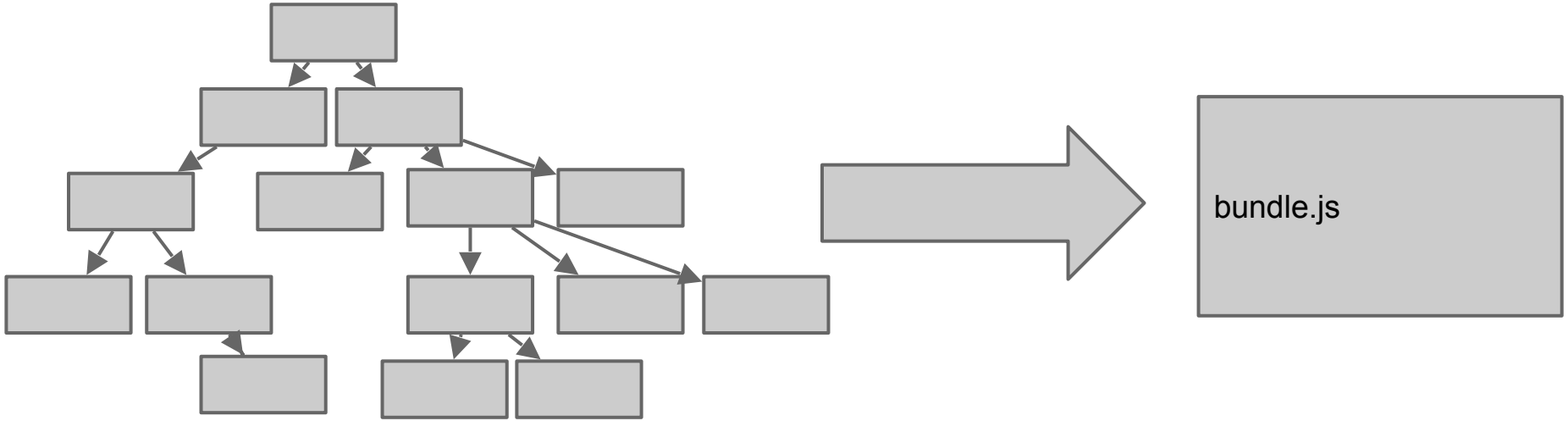
```
exports.doIt =
     function() {}
```

```
import {doIt}
    from 'a-module'
```

```
var {doIt} =
 require('a-module')
```

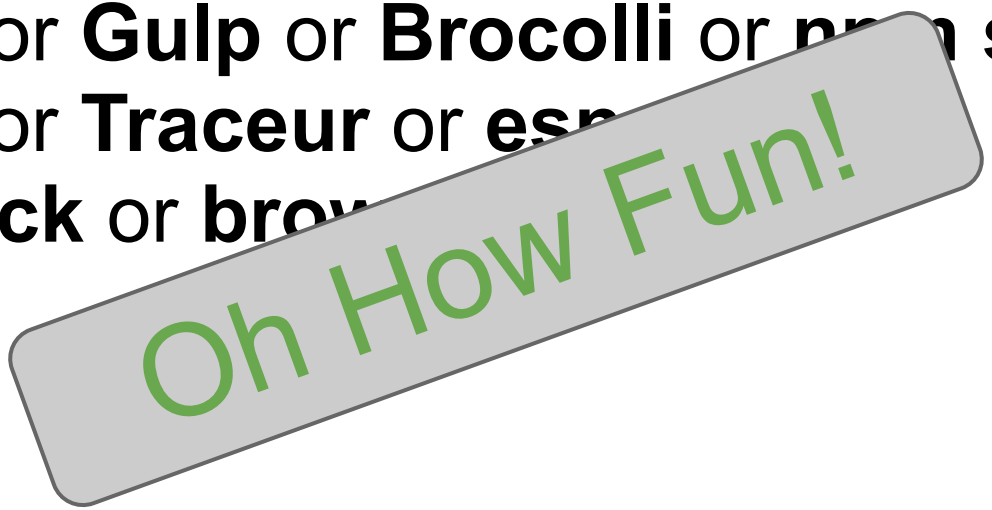
Punch #2

Bundling CommonJS modules



Use…

- **Grunt** or **Gulp** or **Brocolli** or **npm scripts**
- **Babel** or **Traceur** or **esp**
- **webpack** or **brow**



But if you do all this, you get a very sweet carrot

Isomorphic code between Node and the Browser

The Node Dragon

Sonnet 130

The Dragon of Syntax

The Browser Dragon

Sonnet 130

My mistress' eyes are nothing like the sun;
Coral is far more red than her lips' red;
If snow be white, why then her breasts are dun;
If hairs be wires, black wires grow on her head.
I have seen roses damasked, red and white,
But no such roses see I in her cheeks;
And in some perfumes is there more delight
Than in the breath that from my mistress reeks.
I love to hear her speak, yet well I know
That music hath a far more pleasing sound;
I grant I never saw a goddess go;
My mistress when she walks treads on the ground.
And yet, by heaven, I think my love as rare
As any she belied with false compare.

So Go…

...and love ES6 modules like you love your beloved:

it’s never perfect,
but it’s still wonderful,
and it gets better over time

Thank You

gil@tayar.org, @giltayar