ES MODULES

What? More?

ES6 MODULES

ES6 MODULES

- When Node.js was introduced (2009), it adopted a module system using require and module.exports, like we know now (influenced by CommonJS spec)

ES6 MODULES

- When Node.js was introduced (2009), it adopted a module system using require and module.exports, like we know now (influenced by CommonJS spec)

ES6 MODULES

- When Node.js was introduced (2009), it adopted a module system using require and module.exports, like we know now (influenced by CommonJS spec)

- ECMAScript 6 (2015) approved a native module system for JavaScript using a different set of keywords:

ES6 MODULES

- When Node.js was introduced (2009), it adopted a module system using require and module.exports, like we know now (influenced by CommonJS spec)

- ECMAScript 6 (2015) approved a native module system for JavaScript using a different set of keywords:
  - import <thing> from ‘<file>’

ES6 MODULES

- **When Node.js was introduced (2009), it adopted a module system using require and module.exports, like we know now (influenced by CommonJS spec)**

- **ECMAScript 6 (2015) approved a native module system for JavaScript using a different set of keywords:**
  - import <thing> from '<file>'
  - export <thing>

ES6 MODULES

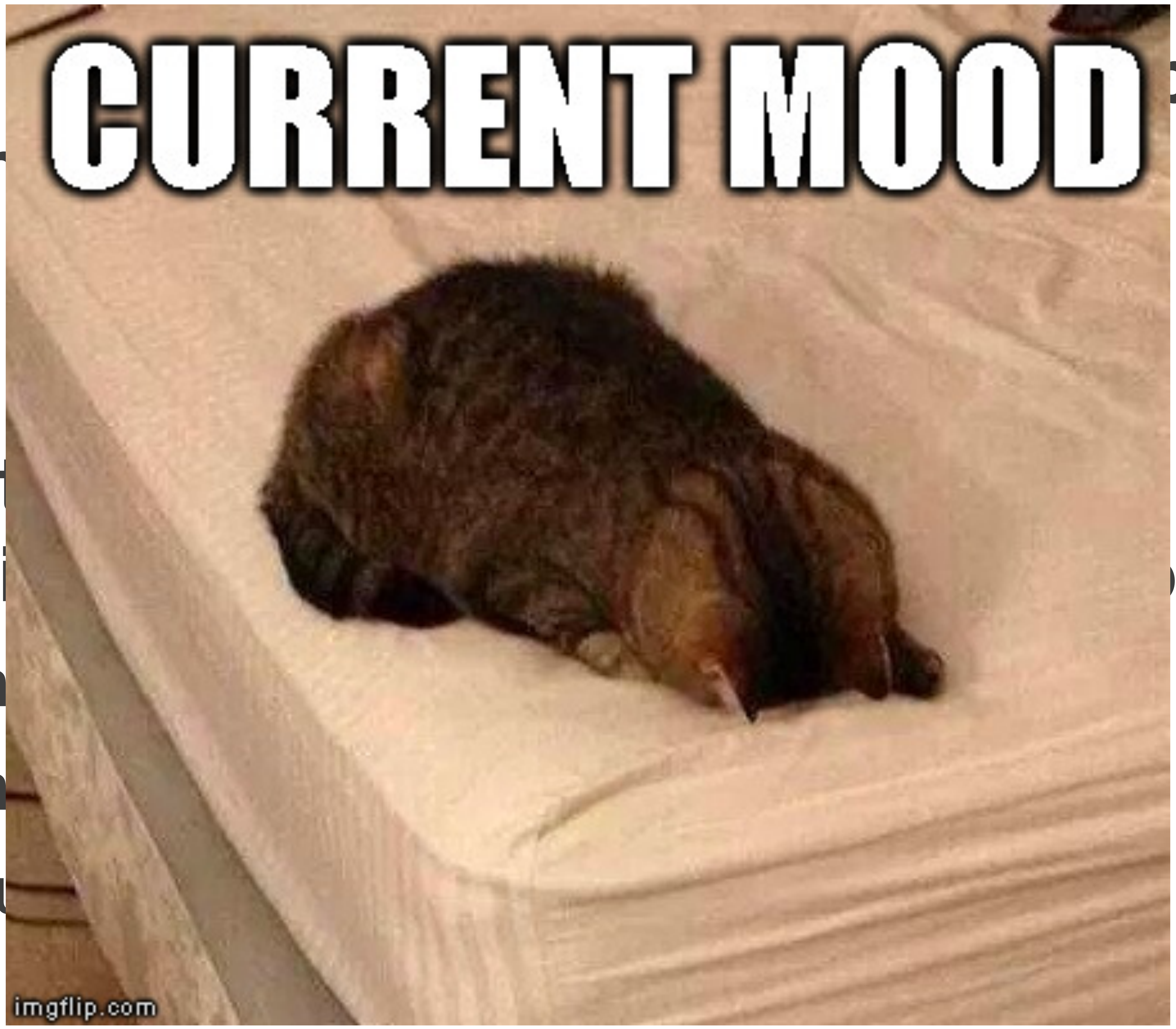
- **When Node.js was introduced (2009), it adopted a module system using require and module.exports, like we know now (influenced by CommonJS spec)**

- **ECMAScript 6 (2015) approved a native module system for JavaScript using a different set of keywords:**
  - import <thing> from ‘<file>’
  - export <thing>
  - export default <thing>

ES6 MODULES

- **When Node** **pted a module system usir** **ke we know now**

- **ECMAScript** **ule system for JavaScri** **rds:**
  - import <thin
  - export <thin
  - export defau



RATIONALE

RATIONALE

- **The Node.js style modules:**

RATIONALE

- The Node.js style modules:
  - only support synchronous module loading

RATIONALE

- The Node.js style modules:
  - only support synchronous module loading
  - are dynamic

RATIONALE

- The Node.js style modules:
  - only support synchronous module loading
  - are dynamic

RATIONALE

- The Node.js style modules:
  - only support synchronous module loading
  - are dynamic

- Import/export were designed:

RATIONALE

- The Node.js style modules:
  - only support synchronous module loading
  - are dynamic

- Import/export were designed:
  - to support both sync and async module loading

RATIONALE

- The Node.js style modules:
  - only support synchronous module loading
  - are dynamic

- Import/export were designed:
  - to support both sync and async module loading
  - are not dynamic, which allows static analysis (i.e. tools can examine your code without running it)

- The Node.js s
  - only suppo ng
  - are dynam
- Import/expor
  - to support loading
  - are not dy lysis (i.e. tools can exami it)

a.js

```
const foo = () => {/* etc */}

module.exports = foo
```

b.js

```
const foo = require('./a')
```

a.js

```
const foo = () => {/* etc */}

module.exports = foo
```

```
const foo = () => {/* etc */}

export default foo
```

b.js

```
const foo = require('./a')
```

```
import foo from './a'
```

a.js

```
const foo = () => {/* etc */}

module.exports = foo
```

b.js

```
const foo = require('./a')
```

```
const foo = () => {/* etc */}

export default foo
```

```
import foo from './a'
```

a.js

```
const foo = () => {/* etc */}
const bar = 'bar'

module.exports = {
  foo: foo,
  bar: bar
}
```

b.js

```
const {foo, bar} = require('./a.js')
```

a.js

```
const foo = () => {/* etc */}
const bar = 'bar'

module.exports = {
  foo: foo,
  bar: bar
}
```

```
export const foo = () => {/* etc */}

export const bar = 'bar'
```

b.js

```
const {foo, bar} = require('./a.js')
```

```
import {foo, bar} from './a'
```

a.js

```
const foo = () => {/* etc */}
const bar = 'bar'

module.exports = {
  foo: foo,
  bar: bar
}
```

b.js

```
const {foo, bar} = require('./a.js')
```

```
export const foo = () => {/* etc */}

export const bar = 'bar'
```

```
import {foo, bar} from './a'
```

a.js

```
const foo =
const bar =

module.expor
  foo: foo,
  bar: bar
}
```

b.js

```
quire('./a.js')
```

```
export const f

export const b
```

```
 from './a'
```

CAN I USE IMPORT/EXPORT?

CAN I USE IMPORT/EXPORT?

- Import/Export has support in the latest versions of some browsers

CAN I USE IMPORT/EXPORT?

- Import/Export has support in the latest versions of some browsers
  - Not quite safe to depend on it without a build tool like webpack

CAN I USE IMPORT/EXPORT?

- Import/Export has support in the latest versions of some browsers
    - Not quite safe to depend on it without a build tool like webpack

- Node.js does not support import/export natively yet

WHAT SHOULD I USE?

WHAT SHOULD I USE?

- Browser-side JavaScript: use import/export and use webpack to compile your code

WHAT SHOULD I USE?

- Browser-side JavaScript: use import/export and use webpack to compile your code

WHAT SHOULD I USE?

- Browser-side JavaScript: use import/export and use webpack to compile your code
- Node: continue to use require and module.exports