JONAS.IO
SCHMEDTMANN

NODE.JS, EXPRESS & MONGODB

THE COMPLETE BOOTCAMP

@JONASSCHMEDTMAN

SECTION

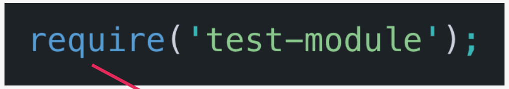
HOW NODE.JS WORKS: A LOOK BEHIND THE SCENES

LECTURE

HOW REQUIRING MODULES REALLY WORKS

THE COMMONJS MODULE SYSTEM

- Each JavaScript file is treated as a separate module;
- Node.js uses the **CommonJS module system**: `require()`, `exports` or `module.exports`;
- **ES module system** is used in browsers: `import/export`;
- There have been attempts to bring ES modules to node.js (`.mjs`).

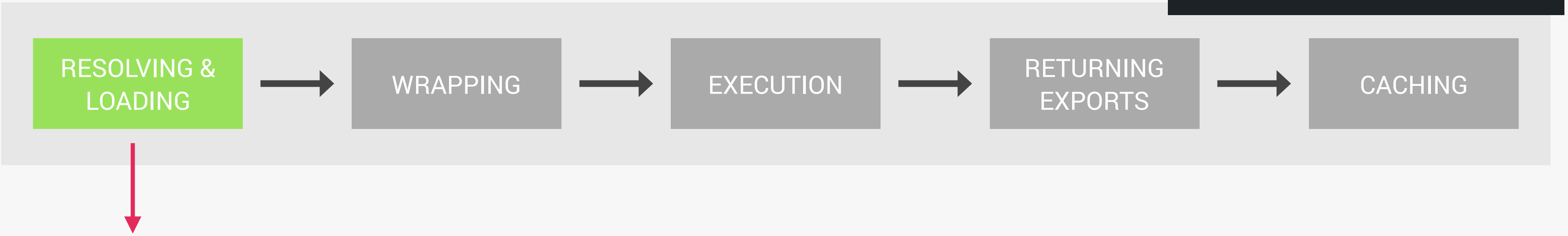


```
require('test-module');
```

Where does it come from?

WHAT HAPPENS WHEN WE REQUIRE() A MODULE

```
require('test-module');
```



RESOLVING & LOADING → WRAPPING → EXECUTION → RETURNING EXPORTS → CACHING

👉 **Core modules**

```
require('http');
```

👉 **Developer modules**

```
require('./lib/controller');
```

👉 **3rd-party modules (from NPM)**

```
require('express');
```

PATH RESOLVING: HOW NODE DECIDES WHICH MODULE TO LOAD

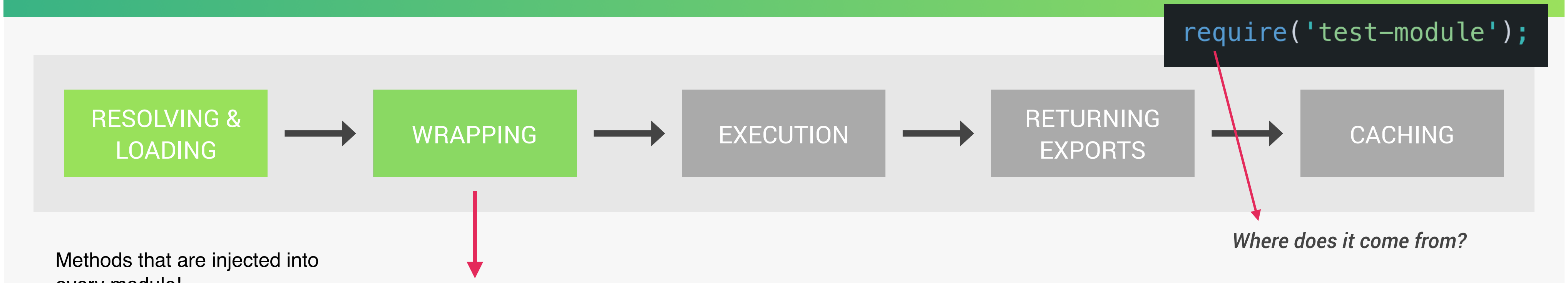
1️⃣ Start with **core modules**;

2️⃣ If begins with ‘`./`‘ or ‘`../`‘ 👉 Try to **load developer module**;

3️⃣ If no file found 👉 Try to **find folder** with `index.js` in it;

4️⃣ Else 👉 Go to **`node_modules/`** and try to find module there.

WHAT HAPPENS WHEN WE REQUIRE() A MODULE

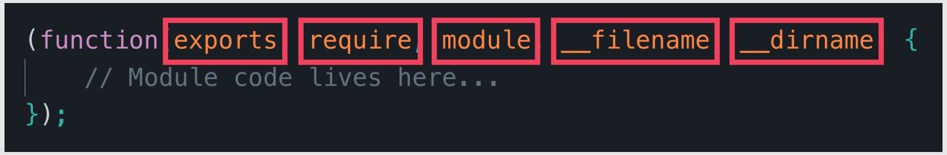


```js
require('test-module');
```

Where does it come from?

Methods that are injected into every module!

Importance:
1- Give access to developers to all these methods;
2- it keeps the top level variables in the current module private scope;

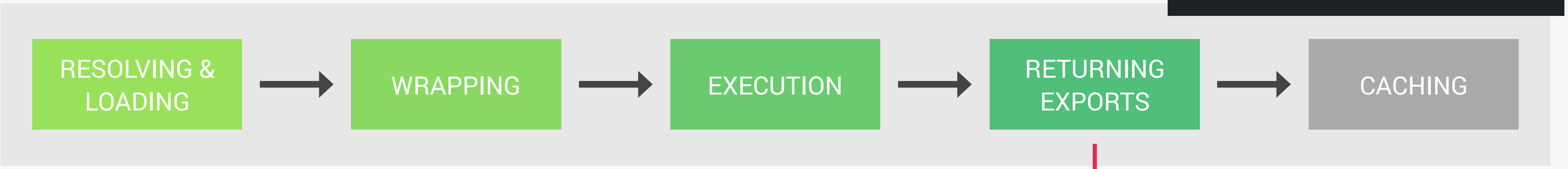


```js
(function(exports, require, module, __filename, __dirname) {
    // Module code lives here...
});
```

- 👉 **`require`**: function to require modules;
- 👉 **`module`:** reference to the current module;
- 👉 **`exports`**: a reference to `module.exports`, used to export object from a module;
- 👉 **`__filename`**: absolute path of the current module’s file;
- 👉 **`__dirname`**: directory name of the current module.

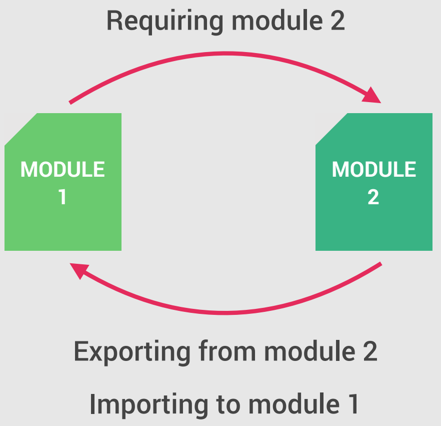
WHAT HAPPENS WHEN WE REQUIRE() A MODULE

```
require('test-module');
```



RESOLVING & LOADING → WRAPPING → EXECUTION → RETURNING EXPORTS → CACHING

- `require` function returns **exports** of the required module;
- `module.exports` is the returned object (important!);
- Use `module.exports` to export one single variable, e.g. one class or one function (`module.exports = Calculator`);
- Use `exports` to export multiple named variables (`exports.add = (a, b) => a + b`);
- This is how we import data from one module into another;



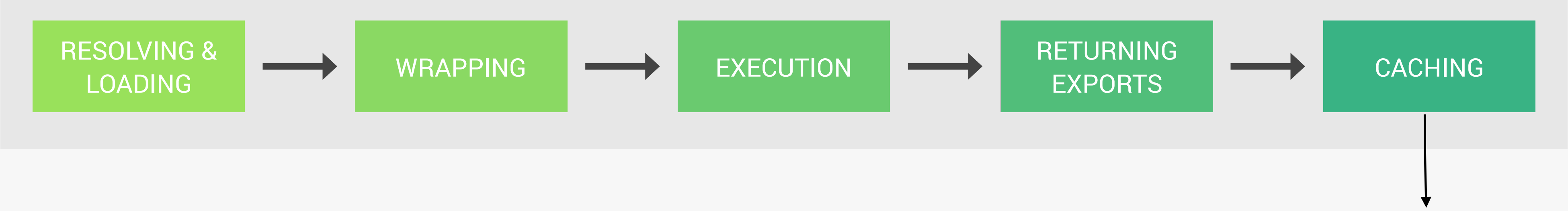
Requiring module 2

Exporting from module 2

Importing to module 1

WHAT HAPPENS WHEN WE REQUIRE() A MODULE

```
require('test-module');
```



If you require the same module multiple times you will always get the same result. It will only execute in the first require after that it will be retrieved from cache