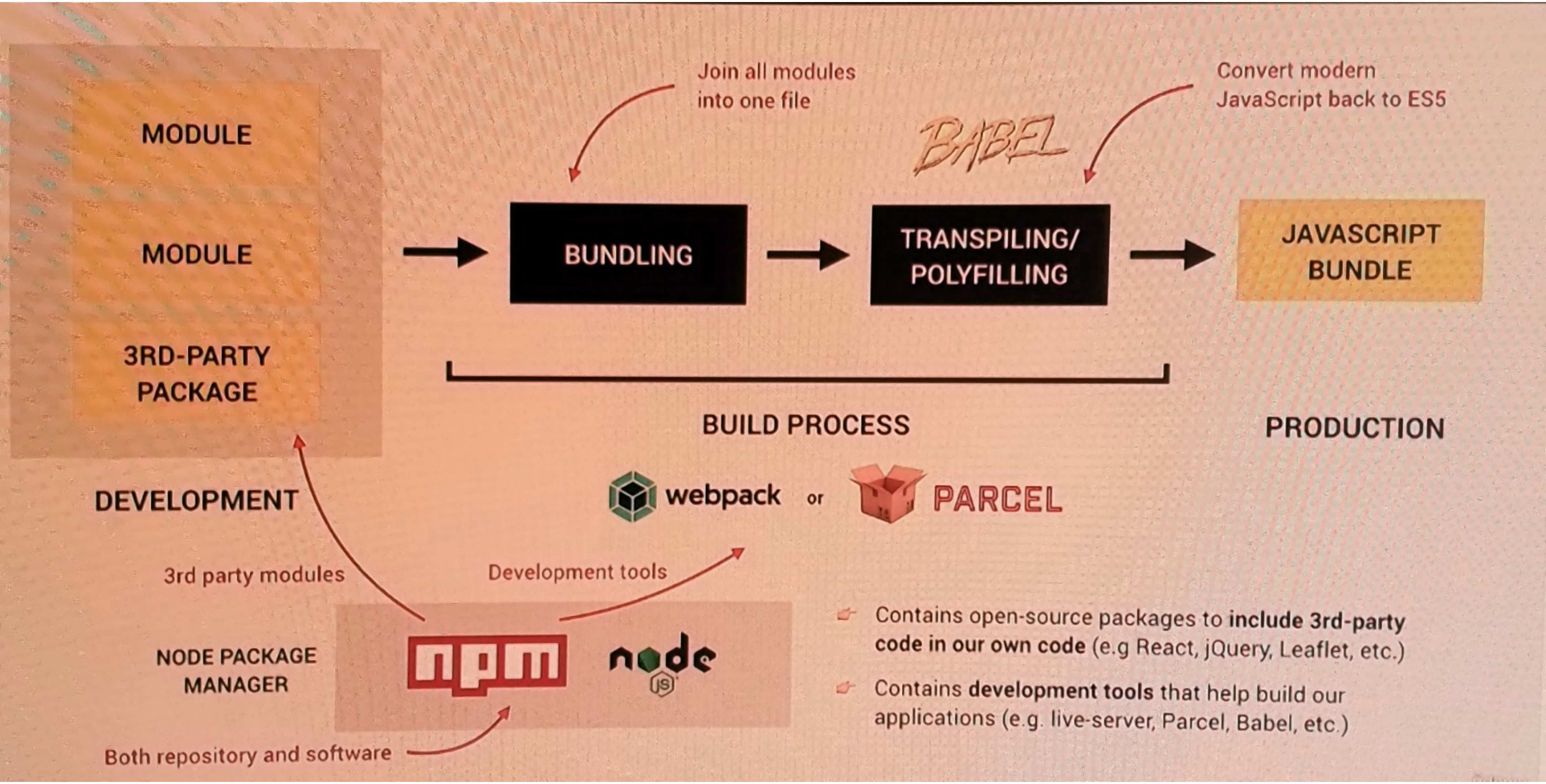


MODULE

MODULE

3RD-PARTY PACKAGE

DEVELOPMENT

Join all modules into one file

BUNDLING

BABEL

Convert modern JavaScript back to ES5

TRANSPILING/ POLYFILLING

JAVASCRIPT BUNDLE

BUILD PROCESS

PRODUCTION

webpack or PARCEL

3rd party modules

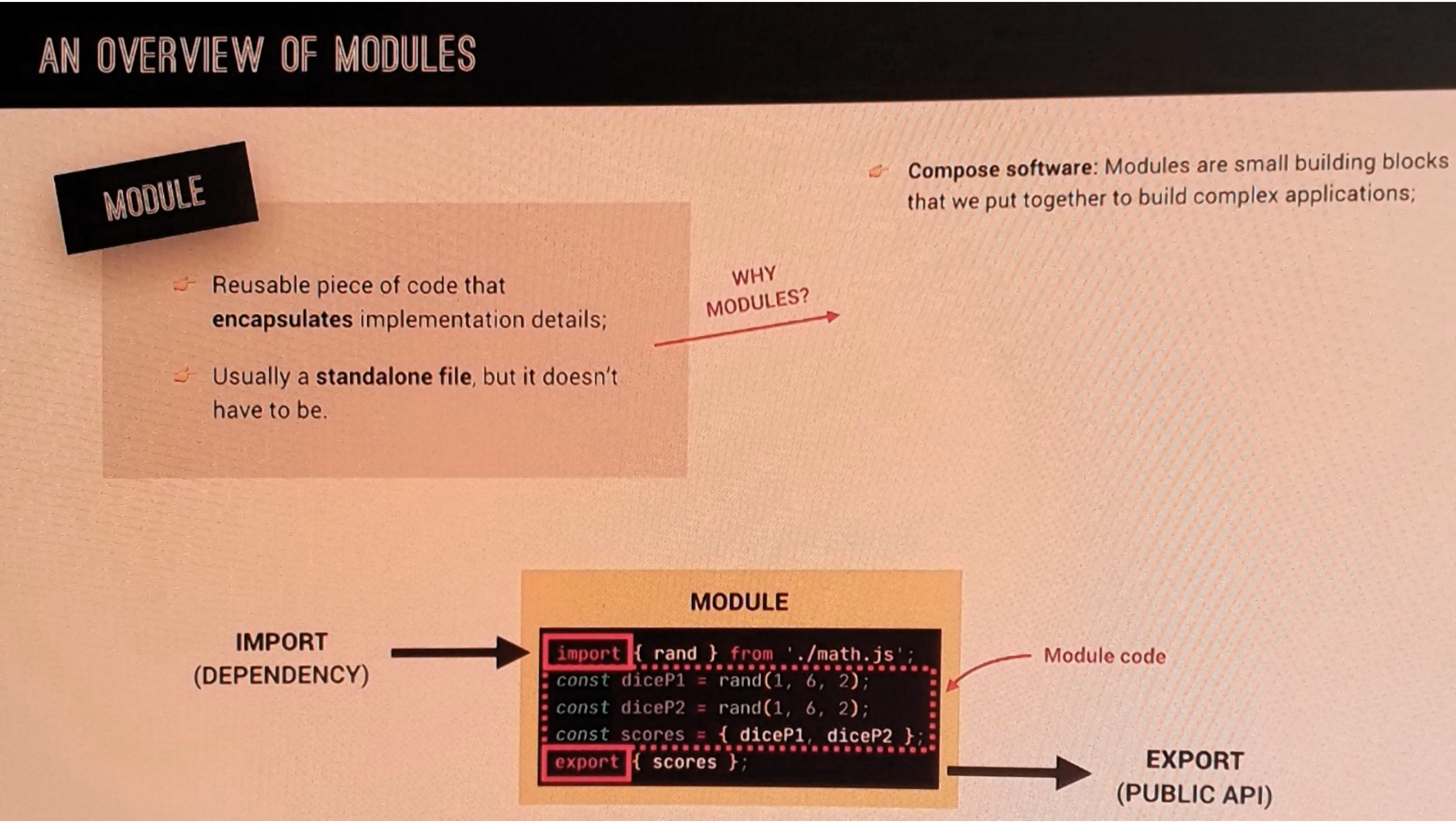
Development tools

NODE PACKAGE MANAGER

npm node

Both repository and software

- Contains open-source packages to **include 3rd-party code in our own code** (e.g React, jQuery, Leaflet, etc.)
- Contains **development tools** that help build our applications (e.g. live-server, Parcel, Babel, etc.)



AN OVERVIEW OF MODULES

MODULE

- Reusable piece of code that **encapsulates** implementation details;
- Usually a **standalone file**, but it doesn't have to be.

WHY MODULES?

- **Compose software:** Modules are small building blocks that we put together to build complex applications;

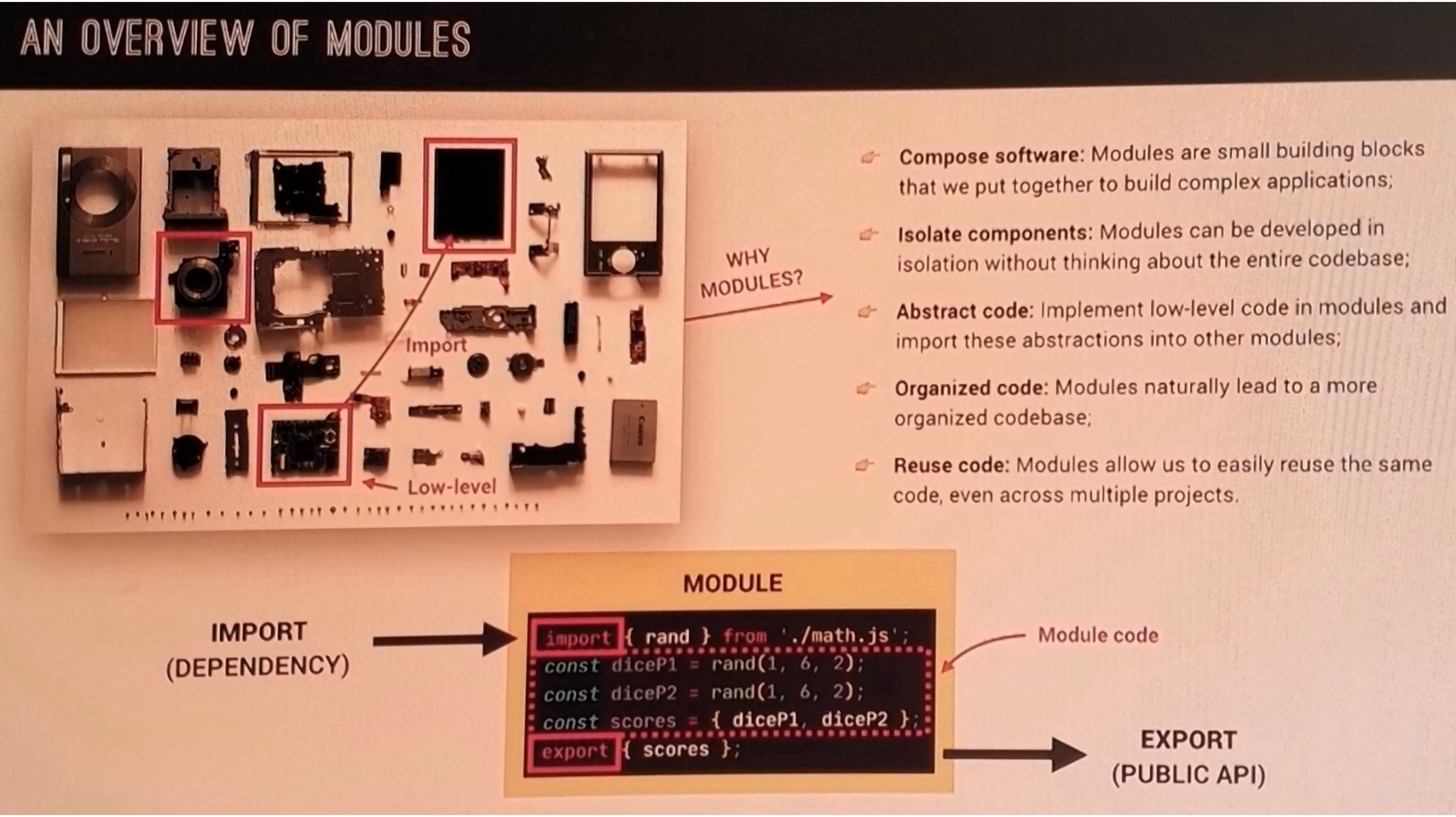
MODULE

IMPORT
(DEPENDENCY)

```
import { rand } from './math.js';
const diceP1 = rand(1, 6, 2);
const diceP2 = rand(1, 6, 2);
const scores = { diceP1, diceP2 };
export { scores };
```

Module code

EXPORT
(PUBLIC API)

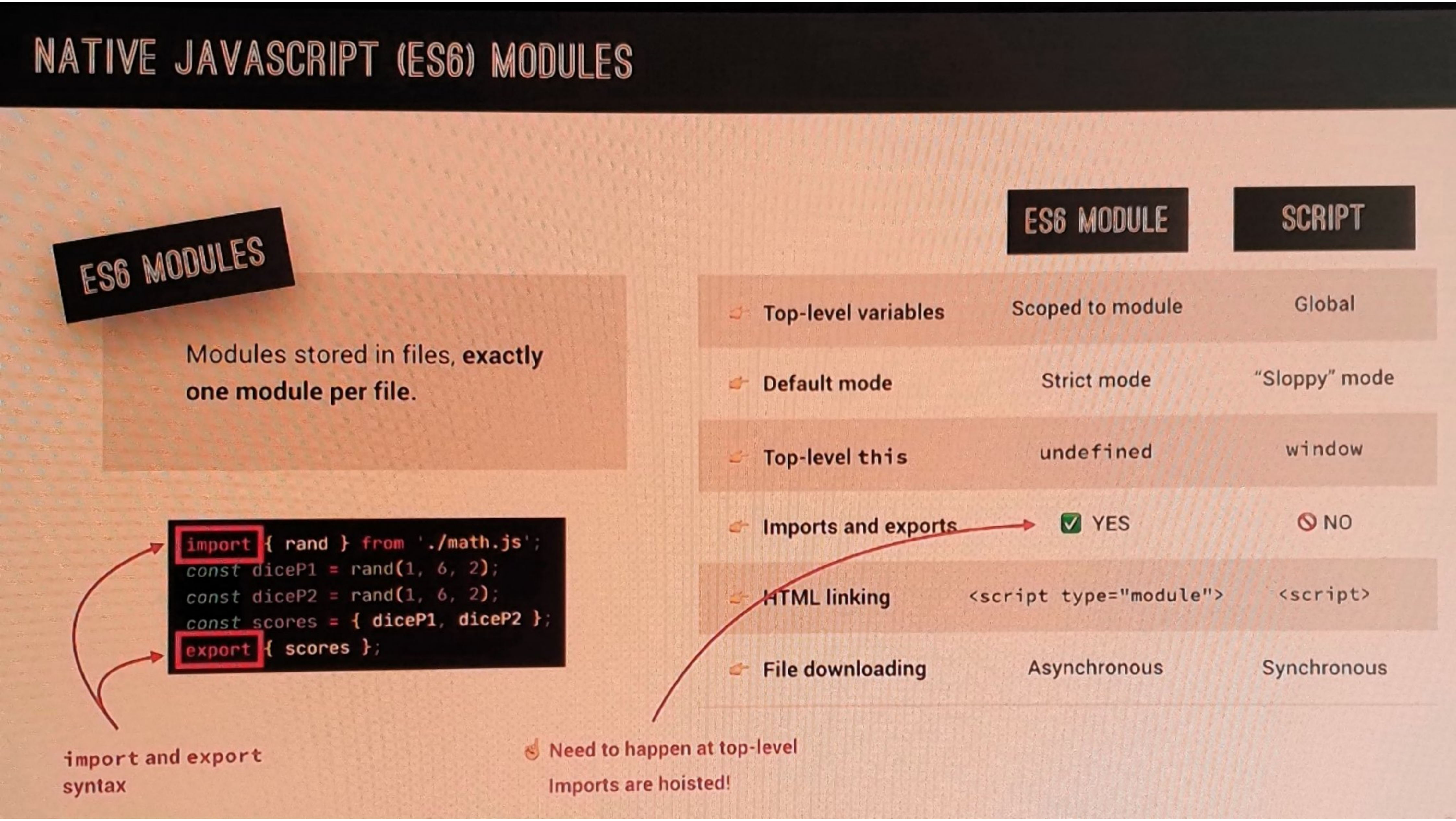


AN OVERVIEW OF MODULES

- **Compose software:** Modules are small building blocks that we put together to build complex applications;
- **Isolate components:** Modules can be developed in isolation without thinking about the entire codebase;
- **Abstract code:** Implement low-level code in modules and import these abstractions into other modules;
- **Organized code:** Modules naturally lead to a more organized codebase;
- **Reuse code:** Modules allow us to easily reuse the same code, even across multiple projects.

MODULE

```
import { rand } from './math.js';
const diceP1 = rand(1, 6, 2);
const diceP2 = rand(1, 6, 2);
const scores = { diceP1, diceP2 };
export { scores };
```



NATIVE JAVASCRIPT (ES6) MODULES

ES6 MODULES

Modules stored in files, **exactly one module per file.**

```
import { rand } from './math.js';
const diceP1 = rand(1, 6, 2);
const diceP2 = rand(1, 6, 2);
const scores = { diceP1, diceP2 };
export { scores };
```

`import` and `export` syntax

	ES6 MODULE	SCRIPT
Top-level variables	Scoped to module	Global
Default mode	Strict mode	"Sloppy" mode
Top-level `this`	`undefined`	`window`
Imports and exports	✅ YES	🚫 NO
HTML linking	`<script type="module">`	`<script>`
File downloading	Asynchronous	Synchronous

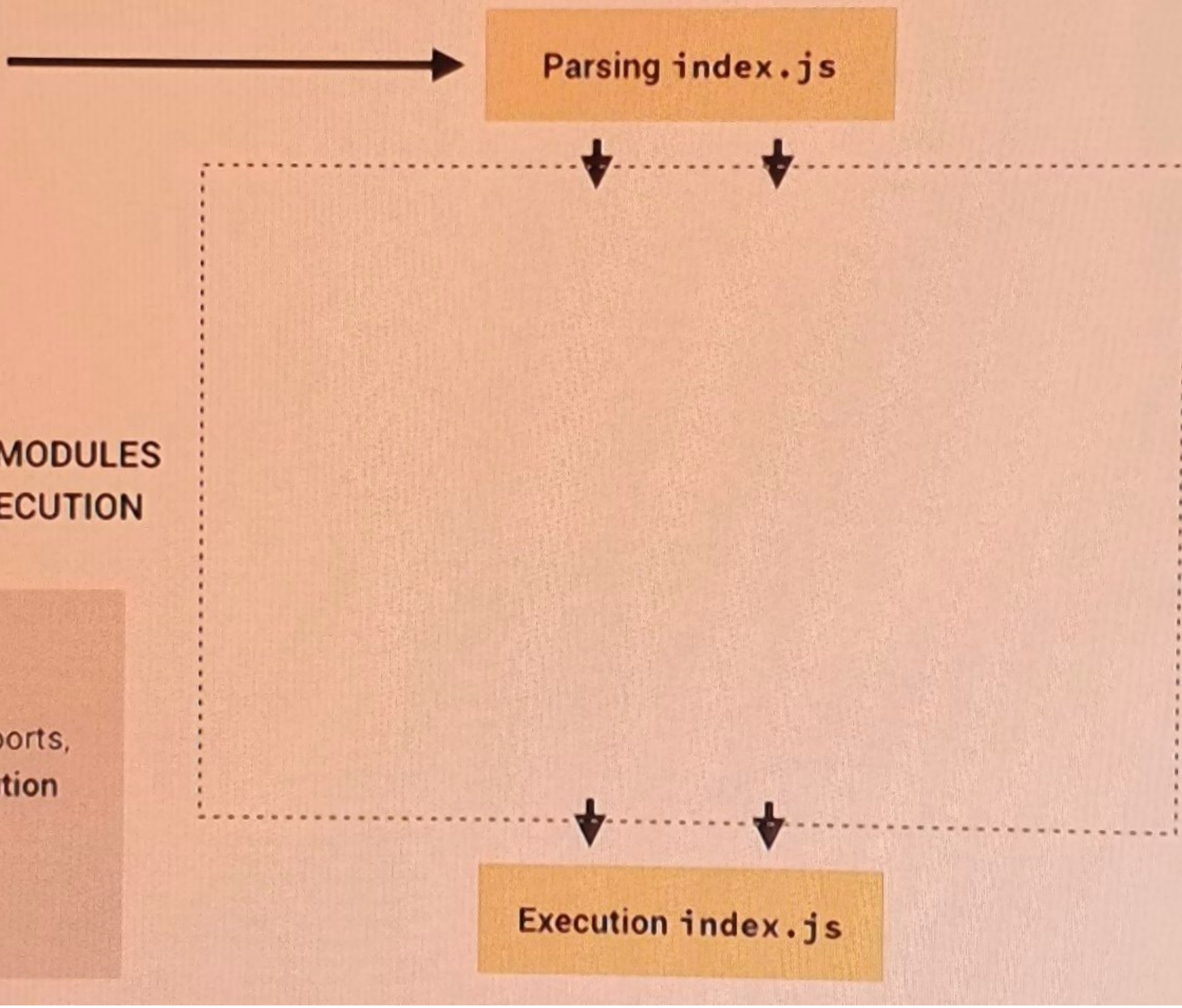
☝ Need to happen at top-level

Imports are hoisted!

OW ES6 MODULES ARE IMPORTED

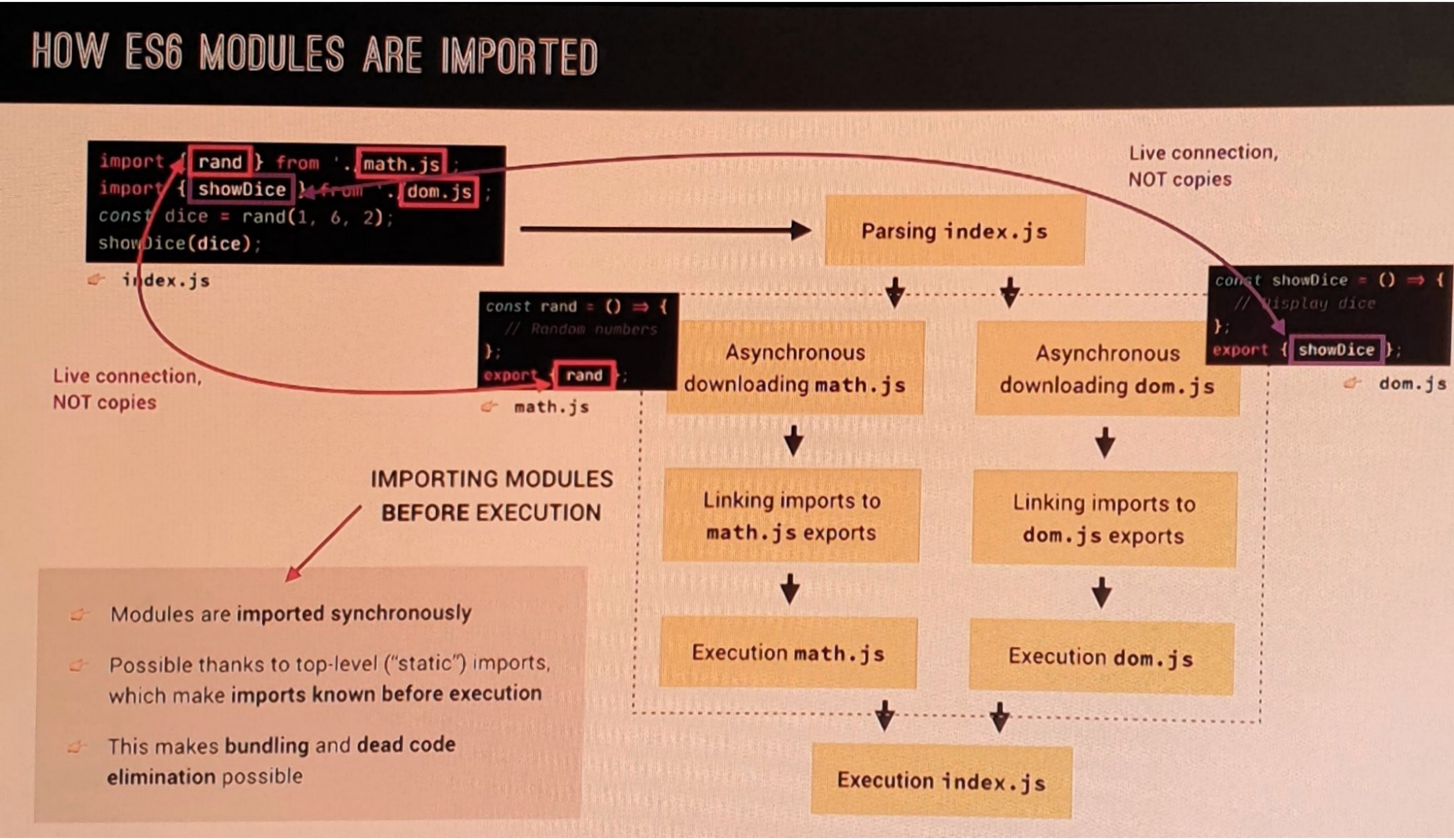
```
import { rand } from './math.js';
import { showDice } from './dom.js';
const dice = rand(1, 6, 2);
showDice(dice);
```

- `index.js`



IMPORTING MODULES BEFORE EXECUTION

- Modules are **imported synchronously**
- Possible thanks to top-level (“static”) imports, which make **imports known before execution**
- This makes **bundling** and **dead code elimination** possible



HOW ES6 MODULES ARE IMPORTED

```
import { rand } from './math.js';
import { showDice } from './dom.js';
const dice = rand(1, 6, 2);
showDice(dice);
```

index.js

```
const rand = () => {
  // Random numbers
};
export { rand };
```

math.js

```
const showDice = () => {
  // Display dice
};
export { showDice };
```

dom.js

Live connection, NOT copies

Live connection, NOT copies

Parsing index.js

Asynchronous downloading math.js

Linking imports to math.js exports

Execution math.js

Asynchronous downloading dom.js

Linking imports to dom.js exports

Execution dom.js

Execution index.js

IMPORTING MODULES BEFORE EXECUTION

- Modules are **imported synchronously**
- Possible thanks to top-level ("static") imports, which make **imports known before execution**
- This makes **bundling** and **dead code elimination** possible