Refactor of import-related Host Hooks

`HostResolveImportedModule`

`HostImportModuleDynamically`

https://nicolo-ribaudo.github.io/modules-import-hooks-refactor

How does loading and evaluating a module work?

Graph loading

⏳ **HOST** wants to run a module. It loads it, it loads all its dependencies, and then contacts **ECMA-262**...

Graph linking

HOST: Hi! Can you prepare this module?

```
import {add} from "./math.js";

const two = add(1, 1);
```

ECMA-262: Sure, I just need the `./math.js` module.

HostResolveImportedModule

HOST: Here it is, I preloaded it earlier so that it's now ready for you :)

```
export function add(x, y) {
  return x + 1;
}
```

ECMA-262: The modules are valid and they can now be run!

Graph evaluation

HOST: Now, can you run this module?

```
import {add} from "./math.js";

const two = add(1, 1);
```

ECMA-262: Sure, I just need the `./math.js` module.

HostResolveImportedModule

HOST: Here it is, again :)

```
export function add(x, y) {
  return x + 1;
}
```

⏳ **ECMA-262**: Done!

How does loading and evaluating a module work? (1/2)

ECMA-262

ParseModule(*sourceText*, …):

1. Parse *sourceText*, and return if there are syntax errors
2. Analyze imports and build the list of imported specifiers

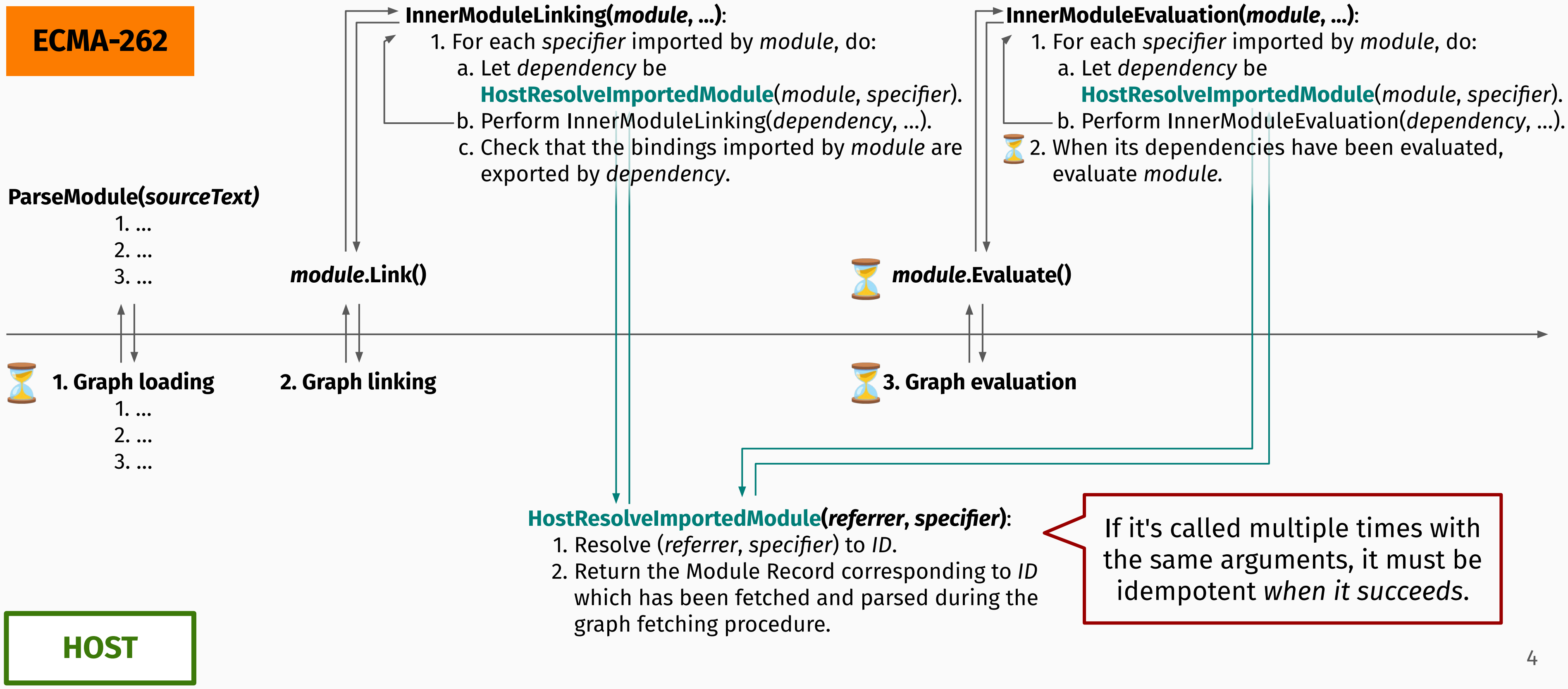
HOST

```
<script
  type="module"
  src="./main.js"
>
```

⏳ **1. Graph loading** procedure with arguments (*referrer*, *specifier*):

1. Resolve (*referrer*, *specifier*) to *ID*.
2. ⏳ Fetch the source text corresponding to *ID*.
3. Let *module* the the Module Record obtained by parsing the source text.
4. For each *dependencySpecifier* imported by *module*, do:
   a. Perform the graph fetching procedure with arguments (*module*, *dependencySpecifier*).

How does loading and evaluating a module work? (2/2)



ECMA-262

InnerModuleLinking(*module*, …):

```
1. For each specifier imported by module, do:
   a. Let dependency be HostResolveImportedModule(module, specifier).
   b. Perform InnerModuleLinking(dependency, …).
   c. Check that the bindings imported by module are exported by dependency.
```

InnerModuleEvaluation(*module*, …):

```
1. For each specifier imported by module, do:
   a. Let dependency be HostResolveImportedModule(module, specifier).
   b. Perform InnerModuleEvaluation(dependency, …).
⏳ 2. When its dependencies have been evaluated, evaluate module.
```

ParseModule(*sourceText*)

1. …
2. …
3. …

***module*.Link()**

⏳ ***module*.Evaluate()**

⏳ **1. Graph loading**

1. …
2. …
3. …

2. Graph linking

⏳ **3. Graph evaluation**

HostResolveImportedModule(*referrer*, *specifier*):

```
1. Resolve (referrer, specifier) to ID.
2. Return the Module Record corresponding to ID which has been fetched and parsed during the graph fetching procedure.
```

If it's called multiple times with the same arguments, it must be idempotent *when it succeeds*.

HOST

How does loading and evaluating a module work?

- From an ECMA-262 point of view, modules loading is *synchronous*.

- From an host point of view, modules loading is *potentially asynchronous*, and it happens before calling the ECMA-262 module graph algorithms.

* This does not need to be true, but it's what happens at least in HTML, Node.js, and Deno.

HostResolveImportedModule synchronously loads dependencies when they are needed.from a cache that has been pre-populated.

Until ES2020!

How does `import()`ing a module work?

- With dynamic `import()` it's not possible for the host to pre-load all the necessary module, since they are not statically known.

- Instead of just relying on HostResolveImportedModule, ECMA-262 had to expose a new asynchronous host hook to let the host "perform whatever I/O operations are necessary to allow HostResolveImportedModule to synchronously retrieve the appropriate Module Record, and then calling its Evaluate concrete method":

HostImportModuleDynamically(*referrer*, *specifier*, …).

How does `import()`ing a module work?

ECMA-262: Hi host! Can you import dynamically `./main.js`?

HostImportModuleDynamically

Graph loading

⏳ **HOST** loads the module and all its dependencies, and then contacts **ECMA-262**...

Graph linking

HOST: Hi! Can you prepare this module? 📄

ECMA-262: Sure, I just need the `./math.js` module.

HostResolveImportedModule

HOST: Here it is, I preloaded it earlier so that it's now ready for you :) 📄

ECMA-262: The modules are valid and they can now be run!

⏳

Graph evaluation

HOST: Now, can you run this module? 📄

ECMA-262: Sure, I just need the `./math.js` module.

HostResolveImportedModule

HOST: Here it is, again :) 📄

ECMA-262: Done!

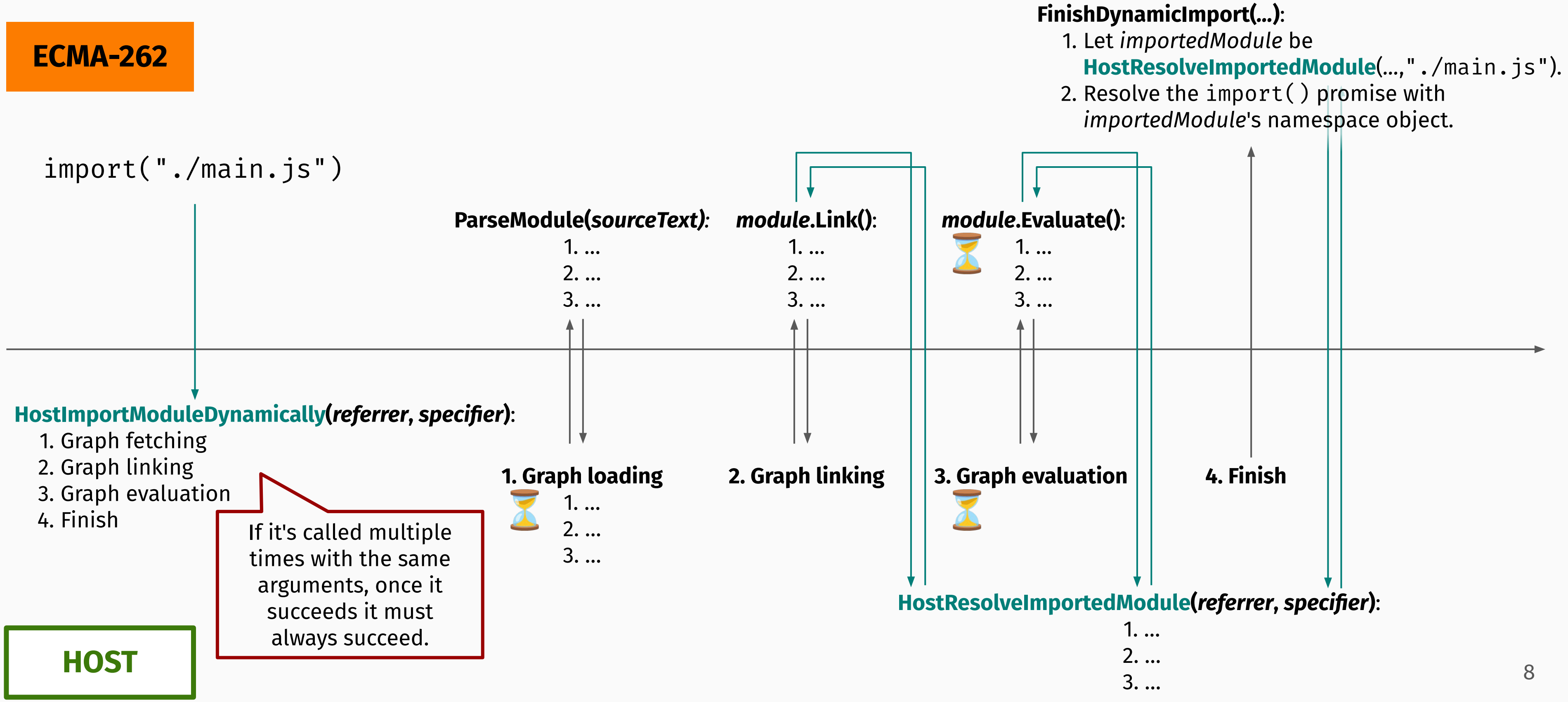
HOST: Hey ECMA-262, I finished the dynamic import.

ECMA-262: Thank you! Can you give me the `./main.js` module?

HostResolveImportedModule

HOST: Here it is :) 📄

How does `import()`ing a module work?



ECMA-262

```
import("./main.js")
```

HostImportModuleDynamically(*referrer*, *specifier*):
1. Graph fetching
2. Graph linking
3. Graph evaluation
4. Finish

If it's called multiple times with the same arguments, once it succeeds it must always succeed.

HOST

ParseModule(*sourceText*):
1. …
2. …
3. …

***module*.Link():**
1. …
2. …
3. …

***module*.Evaluate():** ⌛
1. …
2. …
3. …

1. Graph loading ⌛
1. …
2. …
3. …

2. Graph linking

3. Graph evaluation ⌛

4. Finish

FinishDynamicImport(…):
1. Let *importedModule* be **HostResolveImportedModule**(…,`"./main.js"`).
2. Resolve the `import()` promise with *importedModule*'s namespace object.

HostResolveImportedModule(*referrer*, *specifier*):
1. …
2. …
3. …

Current modules-related proposals and their needs

Module Blocks

- It allows creating inline modules that potentially import other modules:

```
const numbers = module {
     import { add } from "./math.js";
     export const two = add(1, 1);
};
const { two } = await import(numbers);
```

Load "./math.js", link it to numbers, and execute them.

- It needs to load the dependencies of a module that was not created by the host.

HostLoadModuleDependencies(*moduleRecord*)?

Current modules-related proposals and their needs

Import Reflection

- It allows loading a module without loading its dependencies or executing it yet:

```
import module numbers from "./numbers.js";
const { two } = await import(numbers);
```

Load `"./numbers.js"`'s dependencies, link it and execute it.

- It needs to load the the module record without loading its dependencies.

HostLoadUninitializedModule(*referrer*, *specifier*)?

- It needs to load the dependencies of a previously uninitialized module.

HostLoadModuleDependencies(*moduleRecord*)?

Current modules-related proposals and their needs

Compartments

- It allows virtualizing the modules-related host behavior, supporting userland module loaders:

```
const mod = new Module(`export * from "./numbers.js"`, {
     async importHook(specifier) { /* … */ }
};
const { two } = await import(mod);
```

- It needs to specify the graph loading process, by delegating to an asynchronous userland `importHook` function to perform the actual loading of a single uninitialized module.

Can we avoid this host hooks proliferation, and the duplication of the loading algorithm between ECMA-262 and hosts?

One Hook to rule them all

One Hook to rule them all, One Hook to replace them; One Hook to substitute them all and in the darkness delete them.

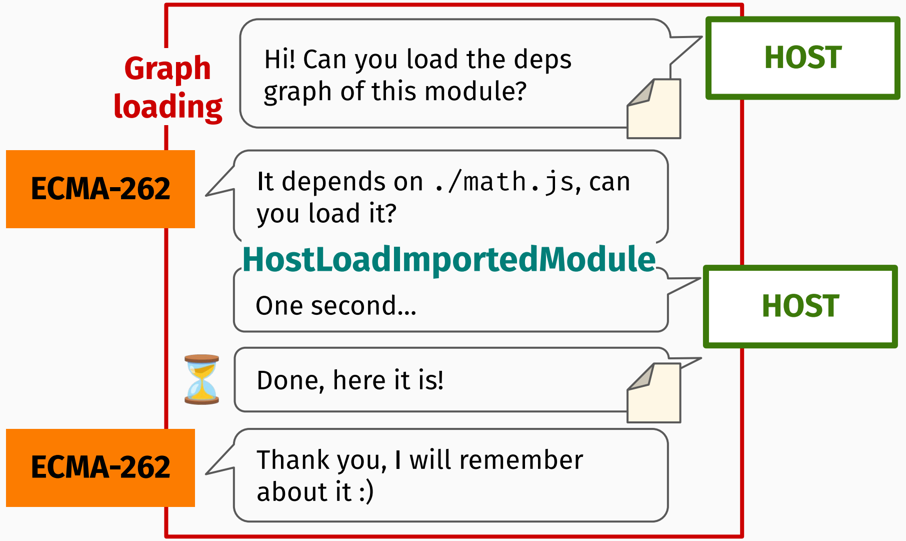
A hook to load a single module, potentially asynchronously, without recursing into its depedencies. A potentially asynchronous version of HostResolveImportedModule.

HostLoadImportedModule(*referrer*, *specifier*, …):

1. Resolve (*referrer*, *specifier*) to *ID*.
2. ⏳ Fetch the *source text* corresponding to *ID*.
3. Let *module* the the Module Record obtained by calling ParseModule(*source text*).
4. Return *module*.

How will loading and evaluating a module work?

⏳ **HOST** wants to run a module. It loads it, and then contacts **ECMA-262**...



Graph loading

HOST: Hi! Can you load the deps graph of this module?

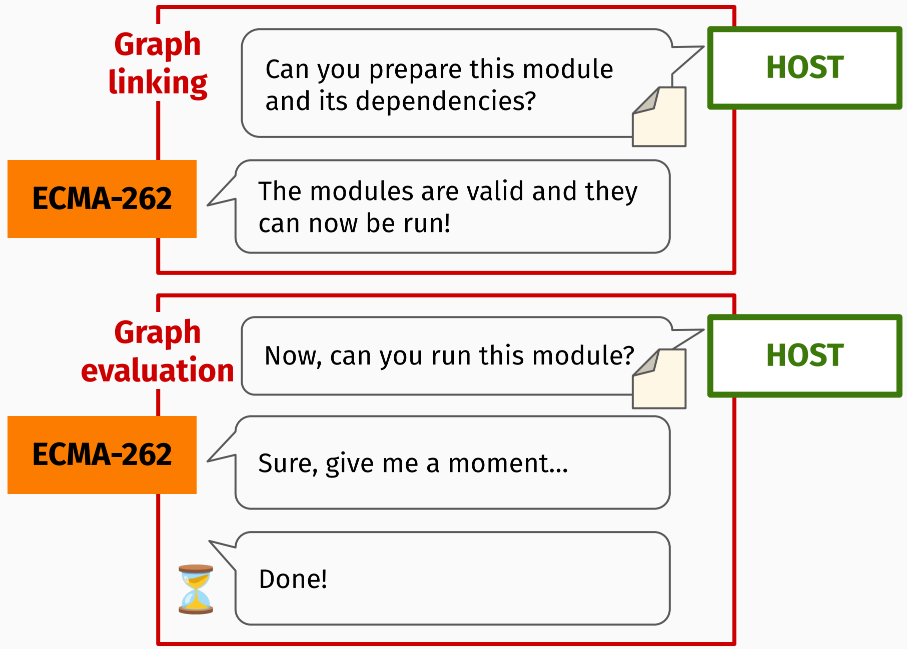
ECMA-262: It depends on `./math.js`, can you load it?

HostLoadImportedModule

HOST: One second...

⏳ Done, here it is!

ECMA-262: Thank you, I will remember about it :)



Graph linking

HOST: Can you prepare this module and its dependencies?

ECMA-262: The modules are valid and they can now be run!

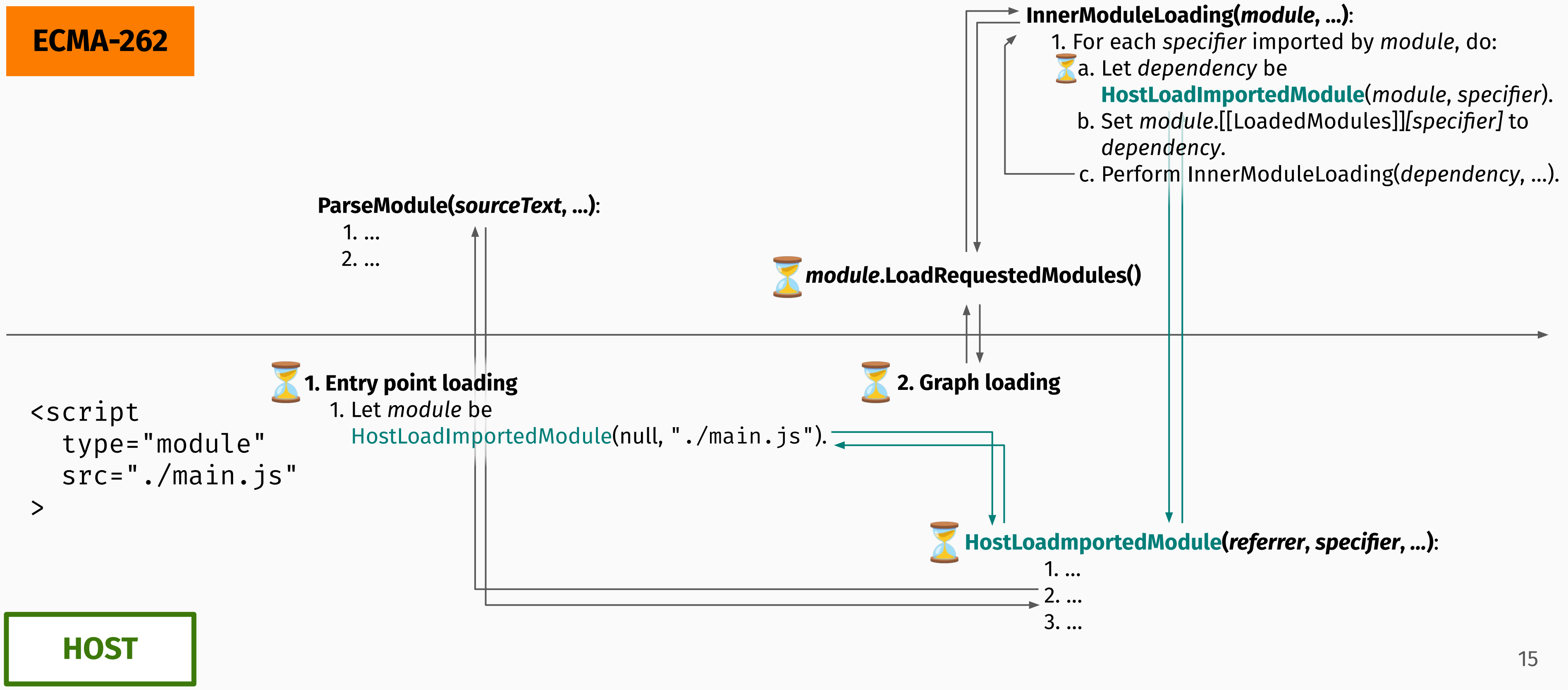
Graph evaluation

HOST: Now, can you run this module?

ECMA-262: Sure, give me a moment...

⏳ Done!

How will loading and evaluating a module work? (1/2)



ECMA-262

InnerModuleLoading(*module*, ...):

```
1. For each specifier imported by module, do:
   ⏳a. Let dependency be
        HostLoadImportedModule(module, specifier).
     b. Set module.[[LoadedModules]][specifier] to
        dependency.
     c. Perform InnerModuleLoading(dependency, ...).
```

ParseModule(*sourceText*, ...):

```
1. ...
2. ...
```

⏳ ***module*.LoadRequestedModules()**

```
<script
  type="module"
  src="./main.js"
>
```

⏳ **1. Entry point loading**

```
1. Let module be
   HostLoadImportedModule(null, "./main.js").
```

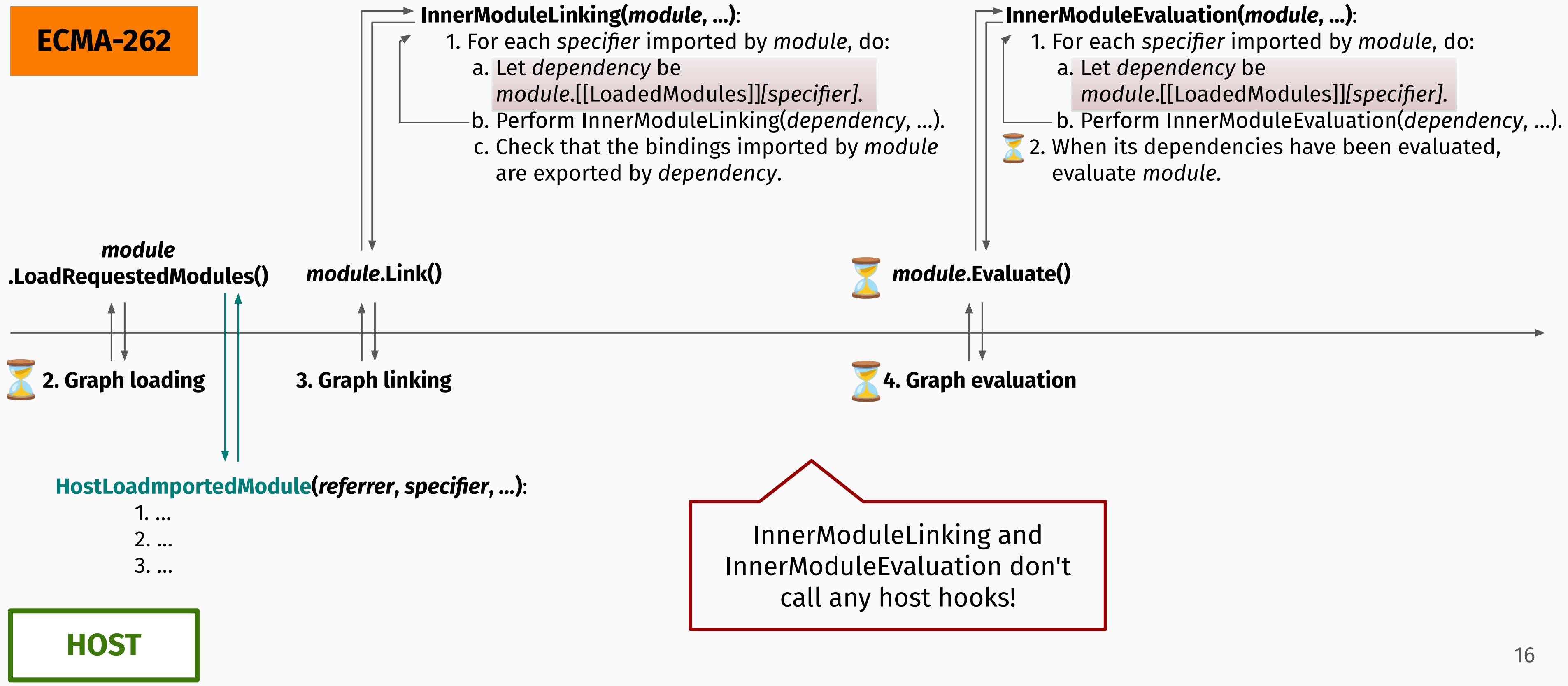
⏳ **2. Graph loading**

⏳ **HostLoadmportedModule(*referrer*, *specifier*, ...):**

```
1. ...
2. ...
3. ...
```

HOST

How will loading and evaluating a module work? (2/2)



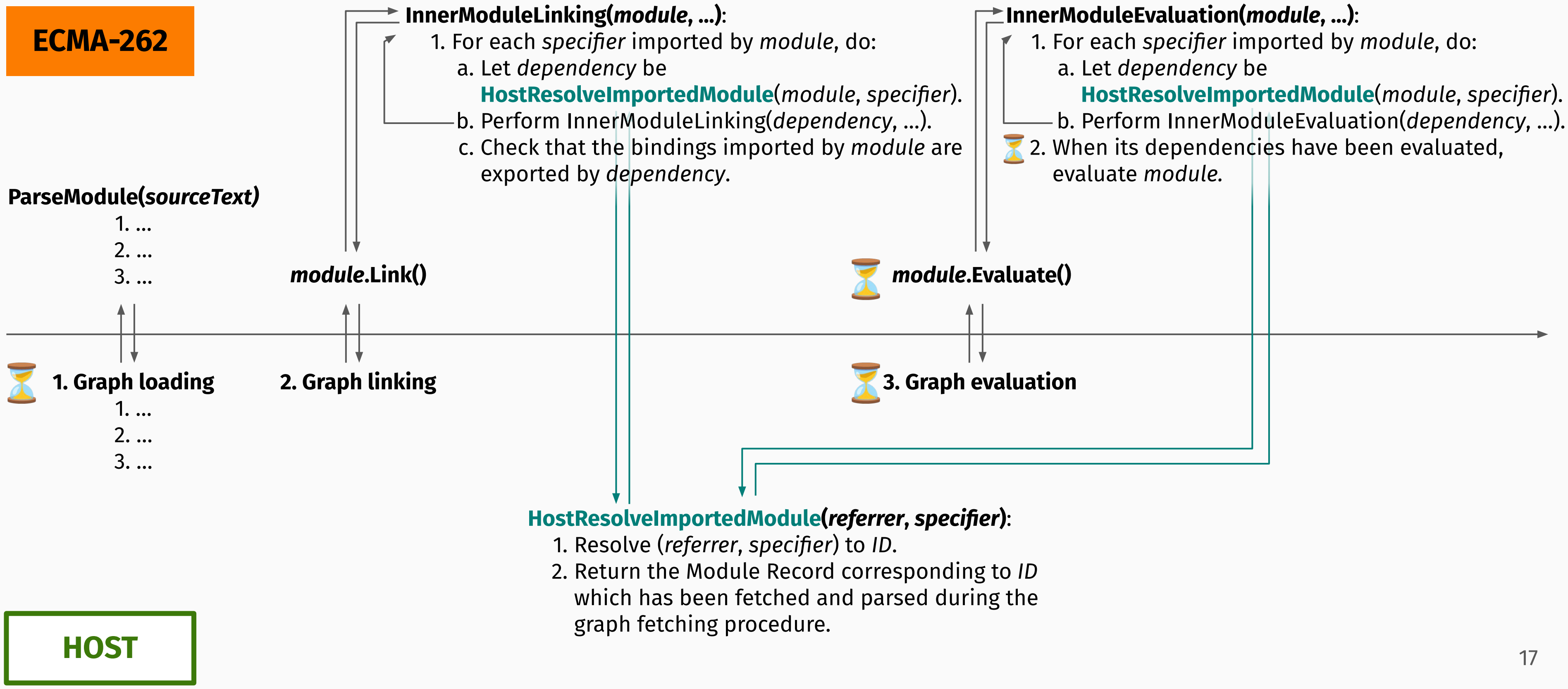
```
InnerModuleLinking(module, …):
    1. For each specifier imported by module, do:
        a. Let dependency be module.[[LoadedModules]][specifier].
        b. Perform InnerModuleLinking(dependency, …).
        c. Check that the bindings imported by module are exported by dependency.
```

```
InnerModuleEvaluation(module, …):
    1. For each specifier imported by module, do:
        a. Let dependency be module.[[LoadedModules]][specifier].
        b. Perform InnerModuleEvaluation(dependency, …).
⏳  2. When its dependencies have been evaluated, evaluate module.
```

```
HostLoadmportedModule(referrer, specifier, …):
    1. …
    2. …
    3. …
```

InnerModuleLinking and InnerModuleEvaluation don't call any host hooks!

(module loading with the current spec, for comparison)



ECMA-262

ParseModule(*sourceText*)
1. …
2. …
3. …

***module*.Link()**

InnerModuleLinking(*module*, …):
1. For each *specifier* imported by *module*, do:
   a. Let *dependency* be **HostResolveImportedModule**(*module*, *specifier*).
   b. Perform InnerModuleLinking(*dependency*, …).
   c. Check that the bindings imported by *module* are exported by *dependency*.

⏳ ***module*.Evaluate()**

InnerModuleEvaluation(*module*, …):
1. For each *specifier* imported by *module*, do:
   a. Let *dependency* be **HostResolveImportedModule**(*module*, *specifier*).
   b. Perform InnerModuleEvaluation(*dependency*, …).
2. ⏳ When its dependencies have been evaluated, evaluate *module*.

⏳ **1. Graph loading**
1. …
2. …
3. …

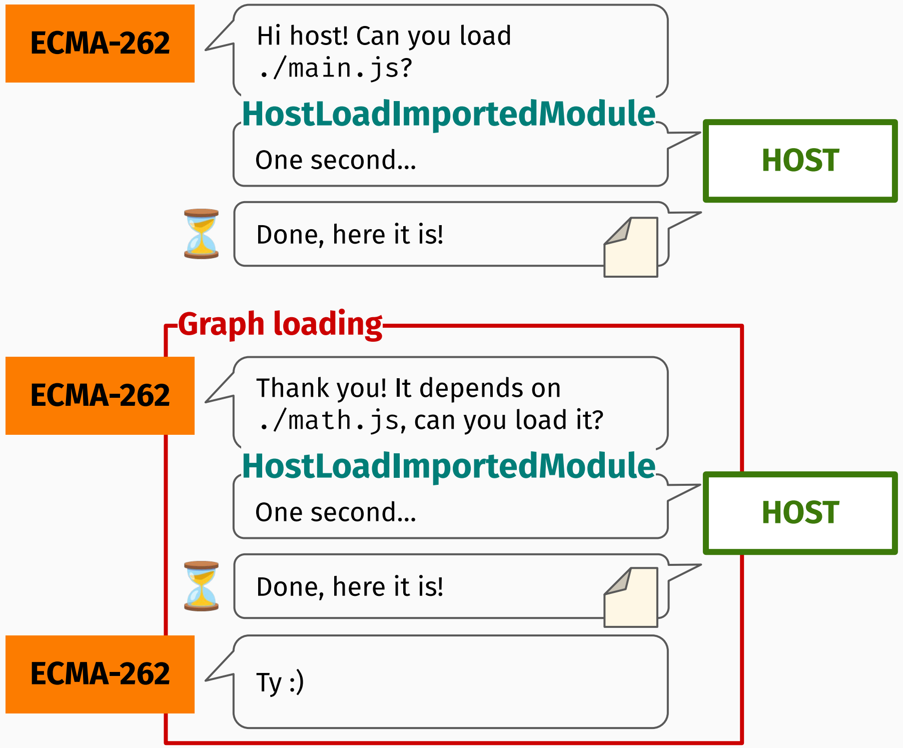
2. Graph linking

⏳ **3. Graph evaluation**

HostResolveImportedModule(*referrer*, *specifier*):
1. Resolve (*referrer*, *specifier*) to *ID*.
2. Return the Module Record corresponding to *ID* which has been fetched and parsed during the graph fetching procedure.

HOST

How will `import()`ing a module work?



⏳ **ECMA-262** now **links** and **evaluates** `./main.js`, withuot needing to talk with the host anymore.

How will `import( )`ing a module work?

ECMA-262

```
import("./main.js")
```

1. Let *module* be HostLoadImportedModule(
   referrer,
   `"./main.js"`
).

**2. *module*
.LoadRequestedModule():**
⏳
1. …
2. …
3. …

3. *module*.Link():
1. …
2. …
3. …

4. *module*.Evaluate():
⏳
1. …
2. …
3. …

5. Resolve the `import( )` promise with *module*'s namespace object.

HostLoadmportedModule(*referrer*, *specifier*, …):
⏳
1. …
2. …
3. …

If it's called multiple times with the same arguments, it must be idempotent *once it succeeds*.

HOST

Integration with modules-related proposals

- **Module Blocks** can call *module*.LoadRequestedModules() to load the dependencies of an inline module.

- **Import Reflection** can call HostLoadImportedModule(*referrer*, *specifier*) to load a module without loading its dependencies.

- **Compartments** can reuse the LoadRequestedModules() algorithm, adjusting it to call the `importHook` function instead of HostLoadImportedModule.

Links

- Spec: https://nicolo-ribaudo.github.io/modules-import-hooks-refactor
- Repo: https://github.com/nicolo-ribaudo/modules-import-hooks-refactor
- HTML PR: https://github.com/whatwg/html/pull/8253

Bonus: avoiding unnecessary host hook calls

I'm not asking for consensus on this *now*!

- Even for dynamic import, we should only call HostLoadImportedModule(*referrer*, *specifier*) if *referrer*.[[LoadedModules]] doesn't contain an entry corresponding to *specifier* yet.

```
await import("./dependency.js");
await import("./dependency.js");
```

If the first import succeeds, the second import would now be guaranteed to take exactly 2 promise ticks.

```
import "./dependency.js";
await import("./dependency.js");
```

If the first import succeeds, the second import would now be guaranteed to succeed and to take exactly 2 promise ticks.