DATE: 11 February 2009

DOC TYPE: PDF file

TITLE: Presentation on Modules

SOURCE [1]: Mr. Kris Kowal (FastSoft Inc.) and Mr. Ihab Awad (Google)

STATUS: presentation on a draft technical contribution

ACTION ID: FYI

NO. OF PAGES:

[1] *NOTE*
The source of this document grants permission to Ecma International to publish this document internally and / or externally (at least one option must be selected)

Back to first page

Modules

Ihab Awad, Google
Kris Kowal, FastSoft

HATE is all you need

Hermetic eval *

- empty scope chain
- no global object
- return last expression
- well-defined approach to primordials

* - Hermetically Air-Tight Evaluator

Q&A

What we can do

Straw proposal for a possible system

Lexical scoping bounds connectivity of "foreign" code

Module Loader

- resolves *module identifiers* ,
- fetches module text,
- evaluates text, and
- interns and returns a *module function*

Identifying a module

Module identifier - a name or location of a piece of code

Any arbitrary expression

Absolute or relative to "self" (or some other base)

"file:///usr/lib/js/util/linkedList.js"

"http://example.com/myModule.js"

```
{ name: "com.example.util.LinkedList",
  version: { min: "3.70", max: "4.00" } }
```

"../util/sortAlgorithms.js"

".util.sortAlgorithms"

Fetching a module

loader.fetch(id, opt_baseId)

Accepts a module identifier and an optional base.

Returns the module's text.

Evaluating a module

loader.evaluate(text)

Wrap in well-known envelope:

```
function evaluate(text) {
    return hermeticEval(
        "(function (require, exports) {" +
         text + ← ES Program production
        "})"
    );
};
```

Result: A (*closed*) module function

Inside a module function

Accepts:

- a mechanism for importing
- a mechanism for exporting
- an environment

```
(function (require, exports) {
    require( id ) ← importing
    exports.name  = ... ← exporting
    ... = require.env.name  ← environment
 })
```

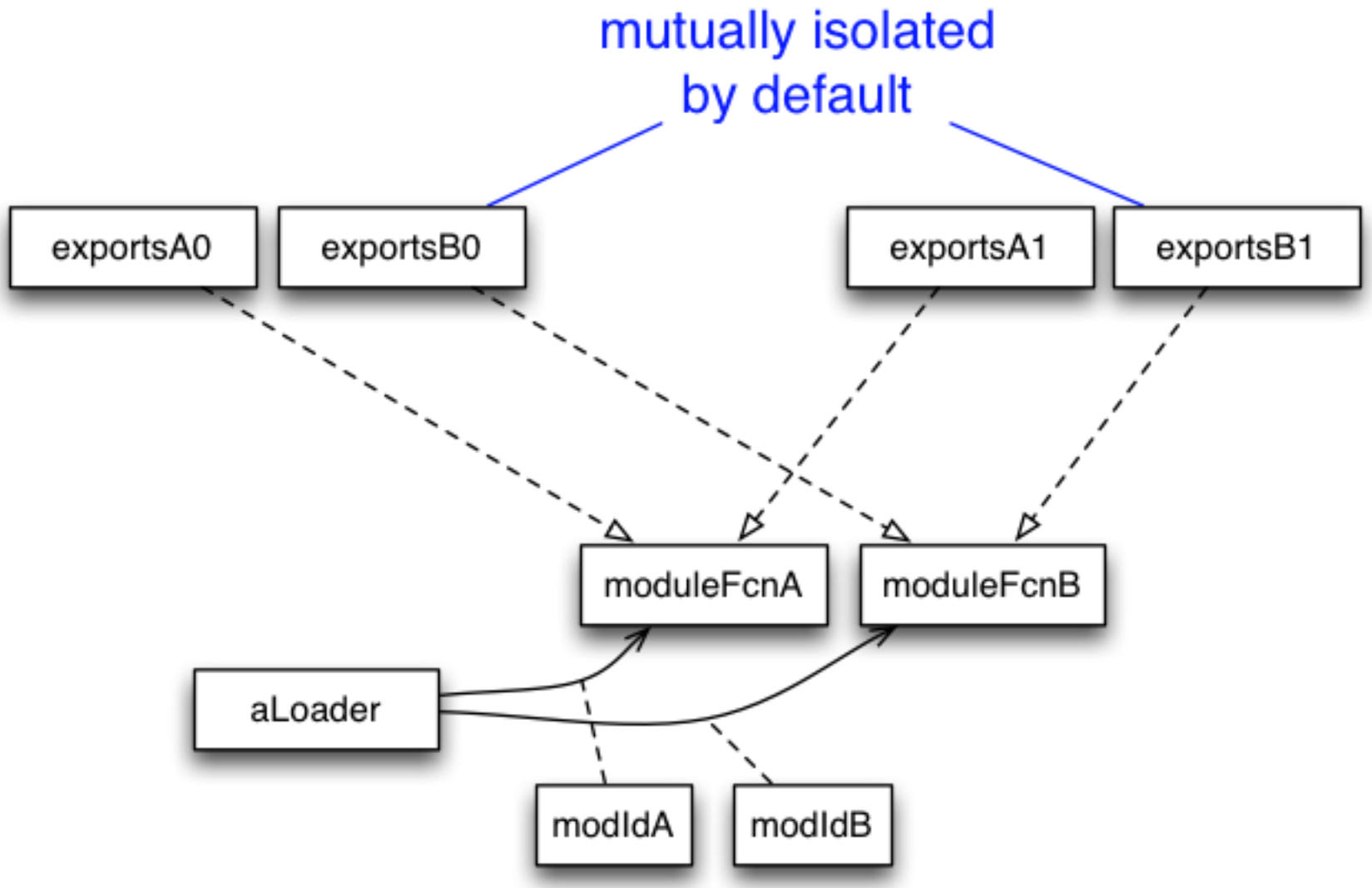
Module instance

The result of invoking a module function:

```
require.env = {
  document: aDocument, window: aDocument
};
let exports = {};
moduleFunction(require, exports);
exports.freeze();
```

exports ← is the module instance

Module instance



A brief moment's reflection...

This is *all* we need ← ya rly this time

This is all *we* need

We and some DI geeks

And a few ocap geeks

But → this is not how *most* people program

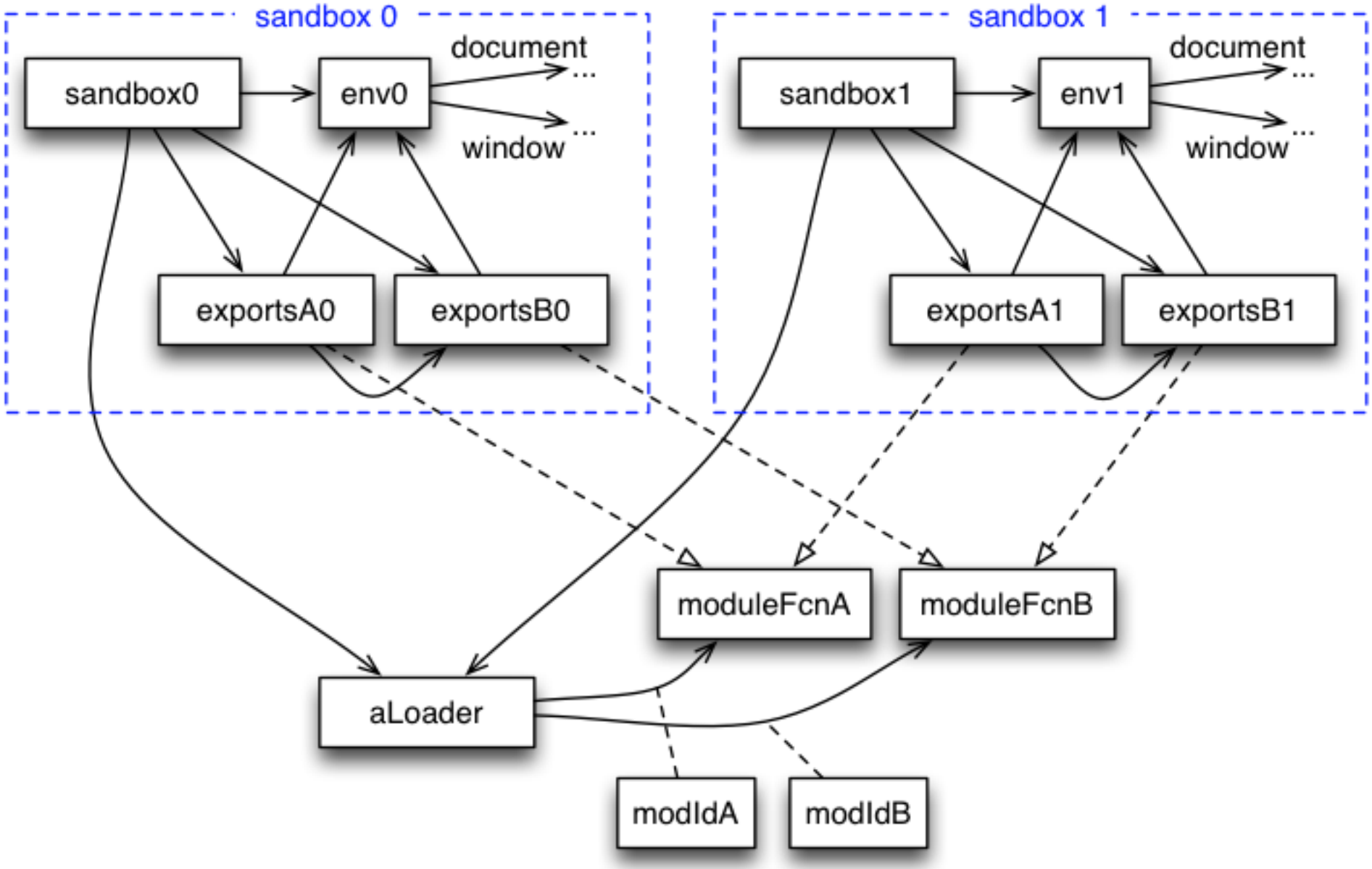
include "foo.js" ← expect a singleton instance

Principled way to do both styles?

Module Sandbox

- an environment
- a loader
- interned module exports (~ *instances*)

Module sandbox



Module environment

{window, document, print, file}

Shared by all modules in a sandbox

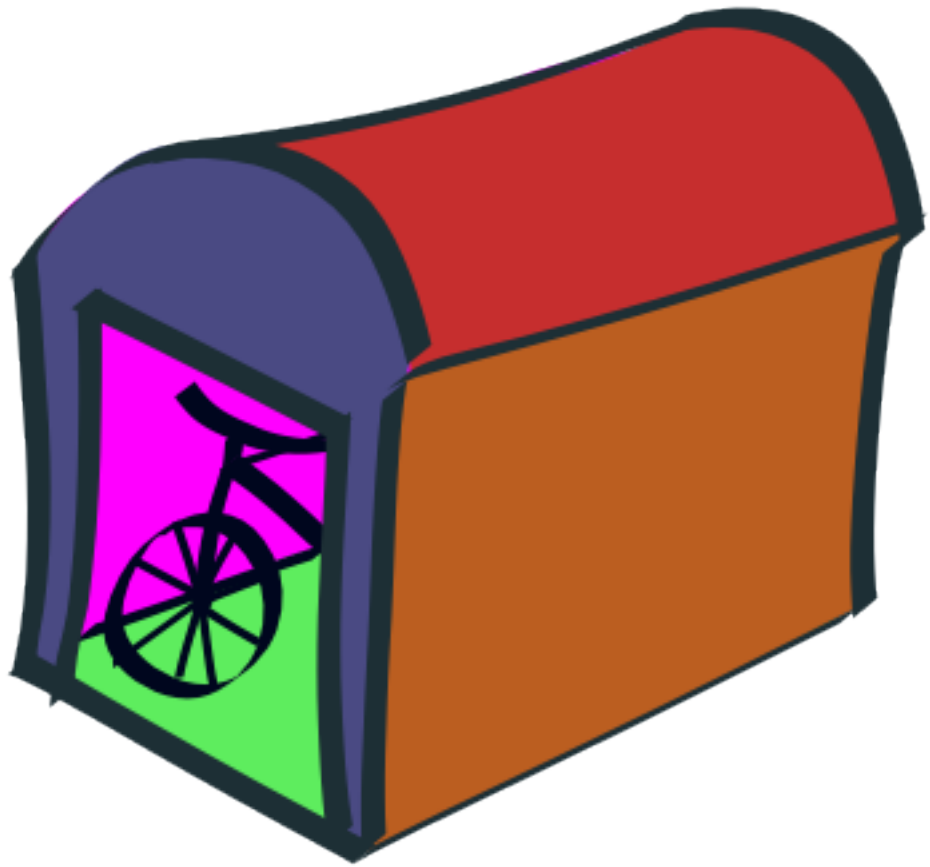
The only conduit to side-effect the world (e.g., I/O)

New sandbox →
    New world of module instances
    ... to which creator is boundedly vulnerable

Module exports memo

Sandbox → 1 module instance per module function

Module functions and loaders may be safely shared



Syntax

Roadmap

- current, global script
- global scripts that can also be used as modules
- modules that can be used with current ES and future ES
- modules that have syntactic sugar

Syntactic variants

- sugared and desugared
- seasoned with salt
- legacy global

- exporting
- importing
- environment

Sugared & Desugared - Exporting

Sugar

```
export X =
  function (n) {
    return n + 1;
  };
export Y =
  function (n) {
    return X(n) + 1;
  };
```

Desugared

```
const X = exports.X =
  function (n) {
    return n + 1;
  };
const Y = exports.Y =
  function (n) {
    return X(n) + 1;
  };
```

Sugared & Desugared - Importing

Sugared

```
import "X" as Y;
import "X" as Y,
  "W" as Z;
from "X" import A;
from "X" import A, B;
from "X" import
  A as F;
```

Desugared

```
const Y =
  require("X");
const W =
  require("Z");
const {A} =
  require("X");
const {A, B} =
  require("X");
const {A: F} =
  require("X");
```

Sugared & Desugared - Environment

```
const {window, document} = require.env;
```

No special syntactic sugar.

Transitional ("Salt") - Export

Resembles the desugared form (in ES3 syntax) with one convenience:

```
exports.X = function (n) { return n + 1; }
exports.Y = function (n) { return X(n) + 1; }
```

Using a `with` block under the covers, we allow exported variables like `X` to appear in scope automatically.

Transitional ("Salt") - Import

Just like the desugared case, but without Harmony syntax.

Transitional ("Salt") - Environment

Just like the desugared case, but without Harmony syntax.

```
var window = require.env.window;
var document = require.env.document;
```

Legacy Global - Export

```
const X = this.X =
  function (n) {
    return n + 1;
  };
const Y = this.Y =
  function (n) {
    return X(n) + 1;
  };
```

To support this syntax, the sandbox must call the module function with the module exports bound to "this".

Legacy Global - Import

Say what?

Legacy Global - Environment

```
var window =
  typeof require == "undefined" ?
  window :
  require.env.window;
```

In legacy global scripts, environment is promiscuous

HATE is all you need

Hermetic eval *

- empty scope chain
- no global object
- return last expression
- well-defined approach to primordials

* - Hermetically Air-Tight Evaluator

Q&A