Modules

Use Cases, Semantics

@littlecalculist, @samth, @wycats


back


don’t give up! we’re almost there
http://www.flickr.com/photos/iwona_kellie/4903753390

```
// libs/string.js

var underscoreRegex1 = /([a-z\d])([A-Z]+)/g,
    underscoreRegex2 = /\-|\s+/g;

export function underscore(string) {
  return string.replace(underscoreRegex1, '$1_$2')
               .replace(underscoreRegex2, '_')
               .toLowerCase();
}

export function capitalize(string) {
  return string.charAt(0).toUpperCase() + string.substr(1);
}
```

```
// app.js

import { capitalize } from "libs/string";

var app = {
  name: capitalize(document.title)
};

export app;
```

```
import "fs" as fs;

fs.rename(oldPath, newPath, function(err) {
  // continue
});
```

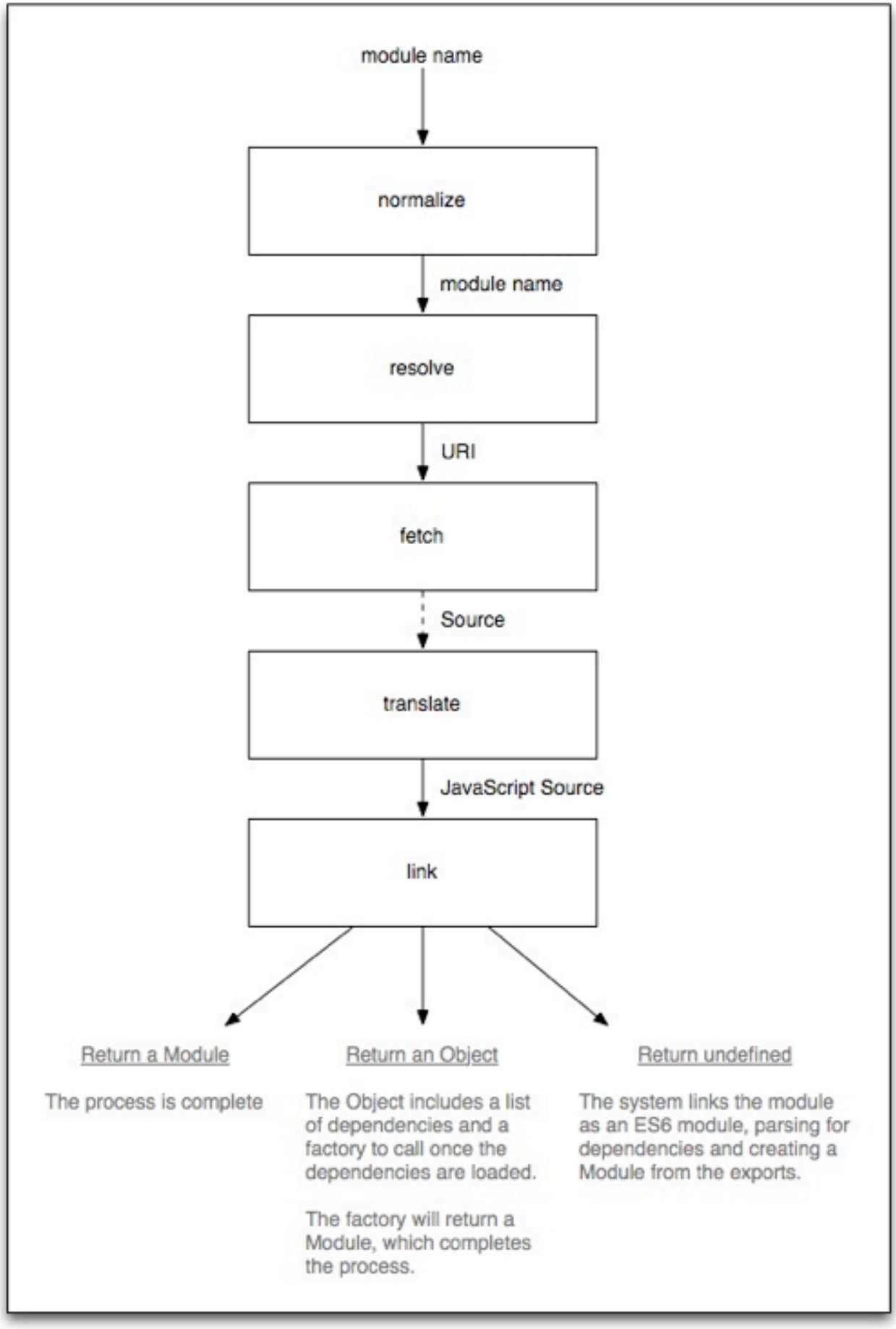
```
module "libs/string" {
  var underscoreRegex1 = /([a-z\d])([A-Z]+)/g,
      underscoreRegex2 = /\-|\s+/g;

  export function underscore(string) {
    return string.replace(underscoreRegex1, '$1_$2')
                 .replace(underscoreRegex2, '_')
                 .toLowerCase();
  }

  export function capitalize(string) {
    return string.charAt(0).toUpperCase() + string.substr(1);
  }
}

module "app" {
  import { capitalize } from "libs/string";

  var app = {
    name: capitalize(document.title)
  };

  export app;
}
```

minimalism – **in**

nested modules – **out**

```
var capitalize = System.get('libs/string').capitalize;
var app = System.get('app').app;
```

```
var sandbox = new Loader({ intrinsics: System });

sandbox.set('app', System.get('app'));
sandbox.get('app') === System.get('app'); // true

sandbox.eval("import { capitalize } from 'app'; capitalize('hi')"); // "Hi"
```


module name
normalize
module name
resolve
URI
fetch
Source
translate
JavaScript Source
link
Return a Module
The process is complete
Return an Object
The Object includes a list of dependencies and a factory to call once the dependencies are loaded.
The factory will return a Module, which completes the process.
Return undefined
The system links the module as an ES6 module, parsing for dependencies and creating a Module from the exports.

Use case

Module paths

(needs better name)

```
System.ondemand({
  "https://ajax.googleapis.com/jquery/2.4/jquery.module.js": "jquery",
  "backbone.js": ["backbone/events", "backbone/model"]
});
```

(needs better name)

```
System.ondemand({
  "https://ajax.googleapis.com/jquery/2.4/jquery.module.js": "jquery",
  "backbone.js": ["backbone/events", "backbone/model"]
});
```

≈

```
System.resolve = function(path) {
  switch (path) {
    case "jquery":
      return "https://ajax.googleapis.com/jquery/2.4/jquery.module.js";
    case "backbone/events":
    case "backbone/model":
      return "backbone.js";
  }
  // fall-through for default
}
```

Use case

Linting

```
import { JSHINT } from "jshint";
import { options } from "app/jshintrc"

System.translate = function(source, options) {
  var errors = JSHINT(source, options), messages = [options.actualAddress];

  if (errors) {
    errors.forEach(function(error) {
      var message = '';
      message += error.line + ':' + error.character + ', ';
      message += error.reason;
      messages.push(message);
    });

    throw new SyntaxError(messages.join("\n"));
  }

  return source;
};
```

Use case

Compiling to JS

```
System.translate = function(source, options) {
  if (!options.path.match(/\.coffee$/)) { return; }

  return CoffeeScript.translate(source);
};
```

Use case

AMD-style plugins

typo; should be mod

```
System.normalize = function(path) {
  if (/^text!/.test(mod)) {
    return { normalized: mod.substring(5) + ".txt", metadata: { type: 'text' } };
  }
  // fall-through for default behavior
}

System.translate = function(src, { metadata }) {
  if (metadata.type === 'text') {
    let escaped = escapeText(src);
    return `export let data = "${escaped}"`;
  }
  // fall-through for default behavior
}
```

```
import { data: foo } from "text!foo";

// logs "hello world"
console.log(foo);
```

Use case

Importing legacy libraries

```
var legacy = ["jquery", "backbone", "underscore"];

System.resolve = function(path, options) {
  if (legacy.indexOf(path) > -1) {
    return { name: path, metadata: { type: 'legacy' } };
  } else {
    return { name: path, metadata: { type: 'es6' } };
  }
}
```

typo; should be source

```
function extractExports(loader, original) {
  source =
    `var exports = {};
    (function(window) { ${source}; })(exports);
    exports;`

  return loader.eval(source);
}

System.link = function(source, options) {
  if (options.metadata.type === 'legacy') {
    return new Module(extractExports(this, source));
  }

  // fall-through for default
}
```

Use case

Importing from AMD

```
System.link = function(source, options) {
  if (options.metadata.type !== 'amd') { return; }

  let loader = new Loader();
  let [ imports, factory ] = loader.eval(`
    let dependencies, factory;
    function define(dependencies, factory) {
      imports = dependencies;
      factory = factory;
    }
    ${source};
    [ imports, factory ];
  `);

  var exportsPosition = imports.indexOf('exports');
  imports.splice(exportsPosition, 1);

  function execute(...args) {
    let exports = {};
    args.splice(exportsPosition, 0, [exports]);
    factory(...args);
    return new Module(exports);
  }

  return { imports: imports, execute: execute };
};
```

Use case

Importing into Node

```
function extractNodeExports(loader, source) {
  var loader = new Loader();
  var exports = loader.eval(`
    var module = {};
    var exports = {};
    var require = System.get;
    ${source};
    { single: module.exports, named: exports };
  `);

  if (exports.single !== undefined) {
    return { exports: exports.single }
  } else {
    return exports.named;
  }
}
```

Use case

Single-export modules

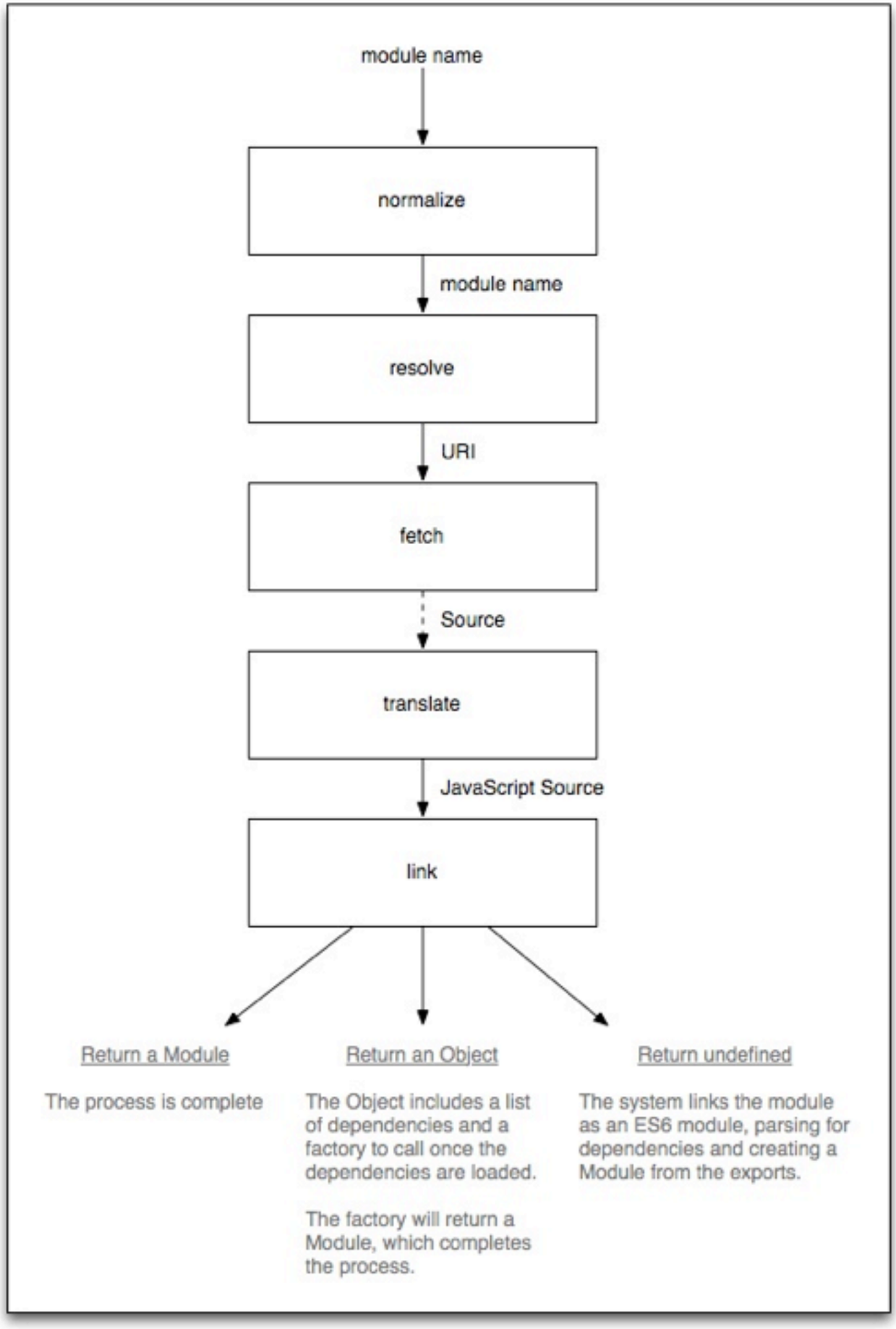
```
var isSingle = new Symbol();

function extractNodeExports(loader, source) {
  var loader = new Loader();
  var exports = loader.eval(`
    var module = {};
    var exports = {};
    var require = System.get;
    ${source};
    { single: module.exports, named: exports };
  `);

  if (exports.single !== undefined) {
    return { exports: exports.single, [isSingle]: true };
  } else {
    return exports.named;
  }
}

System.link = function(source, options) {
  if (options.context === 'node') {
    return new Module(extractNodeExports(this, source));
  }
}
```

typo; should be metadata

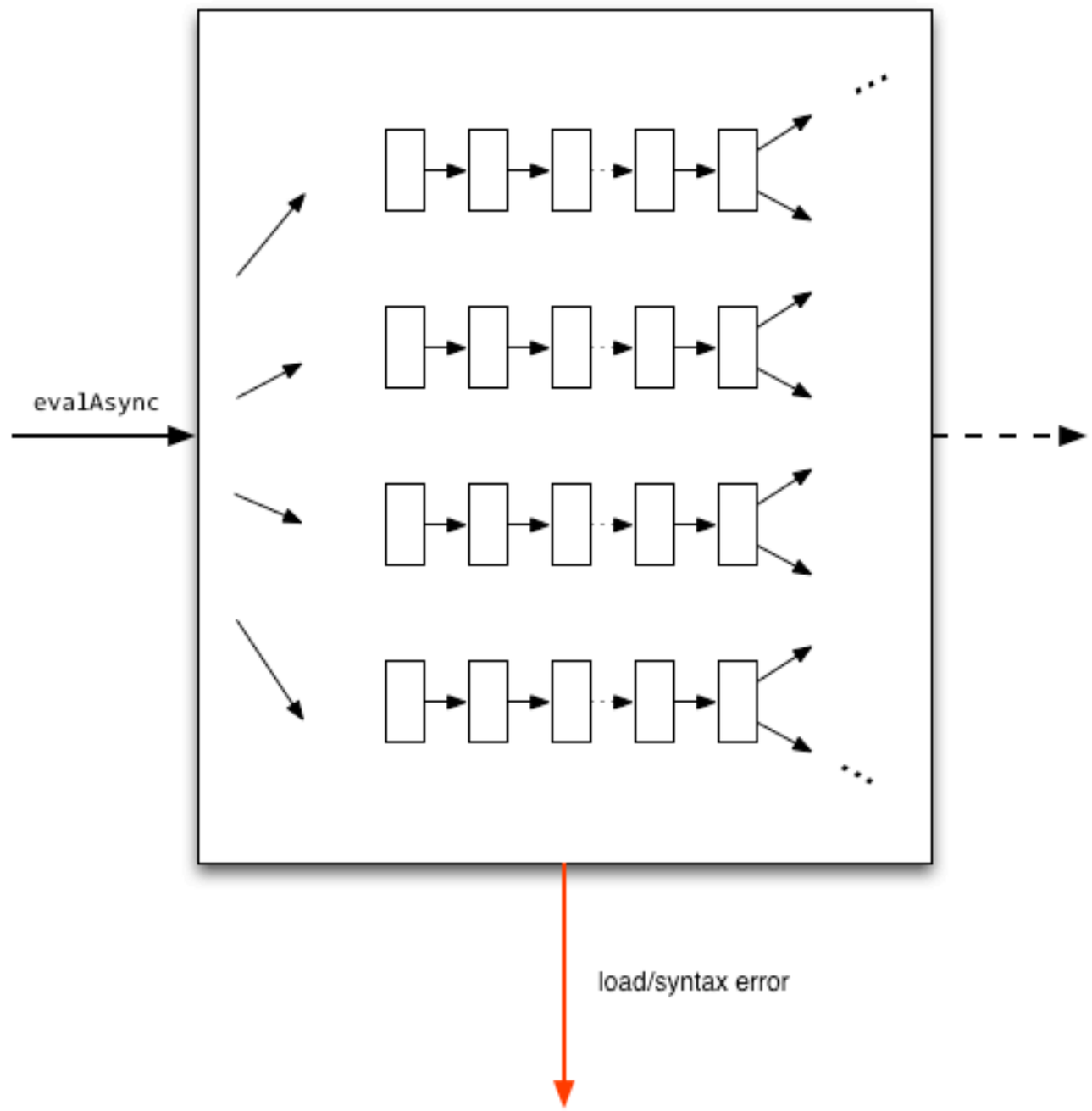
can, should do better

goal – simple sugar

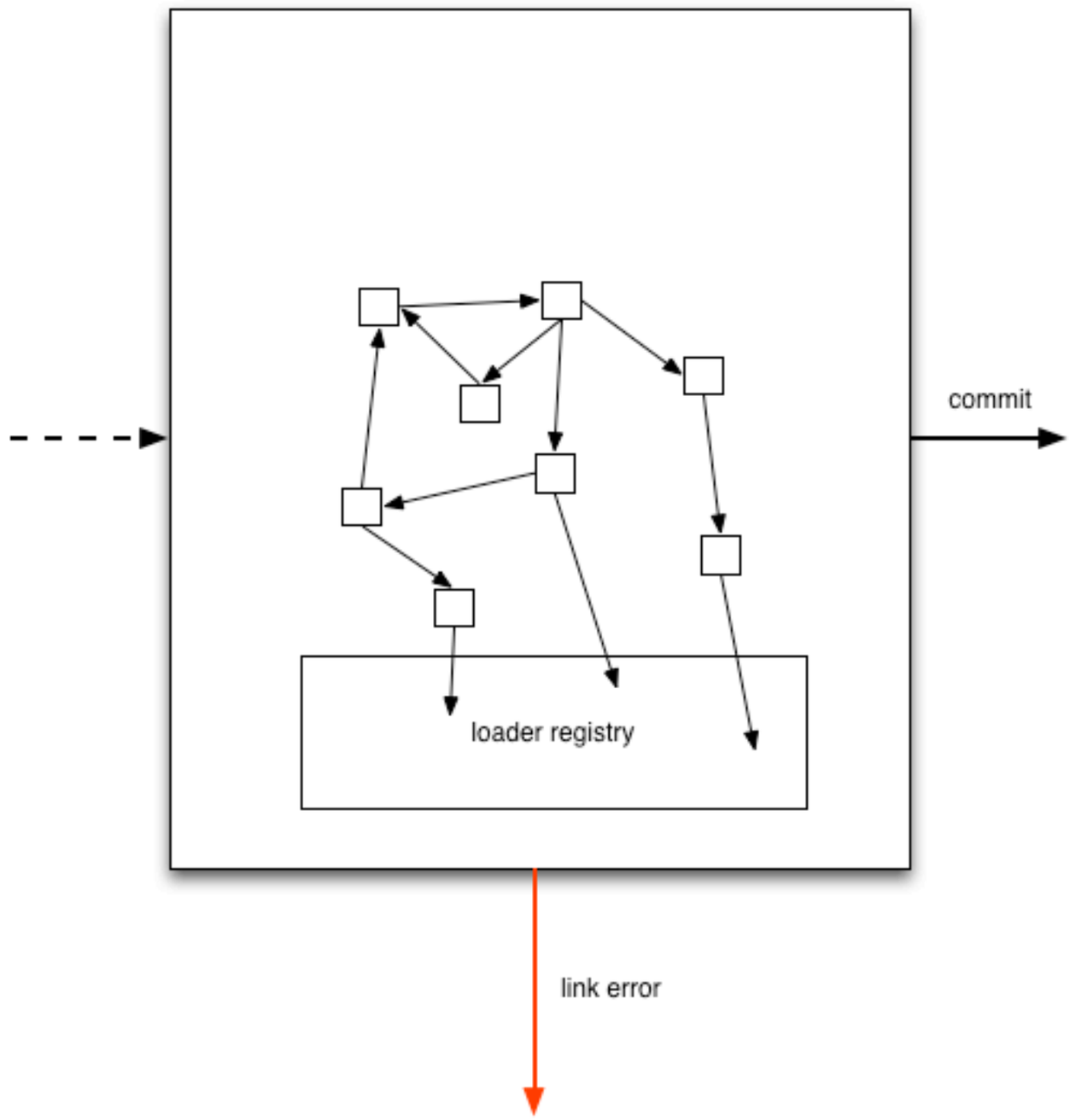
Big picture


module name
normalize
module name
resolve
URI
fetch
Source
translate
JavaScript Source
link
Return a Module
The process is complete
Return an Object
The Object includes a list of dependencies and a factory to call once the dependencies are loaded.
The factory will return a Module, which completes the process.
Return undefined
The system links the module as an ES6 module, parsing for dependencies and creating a Module from the exports.

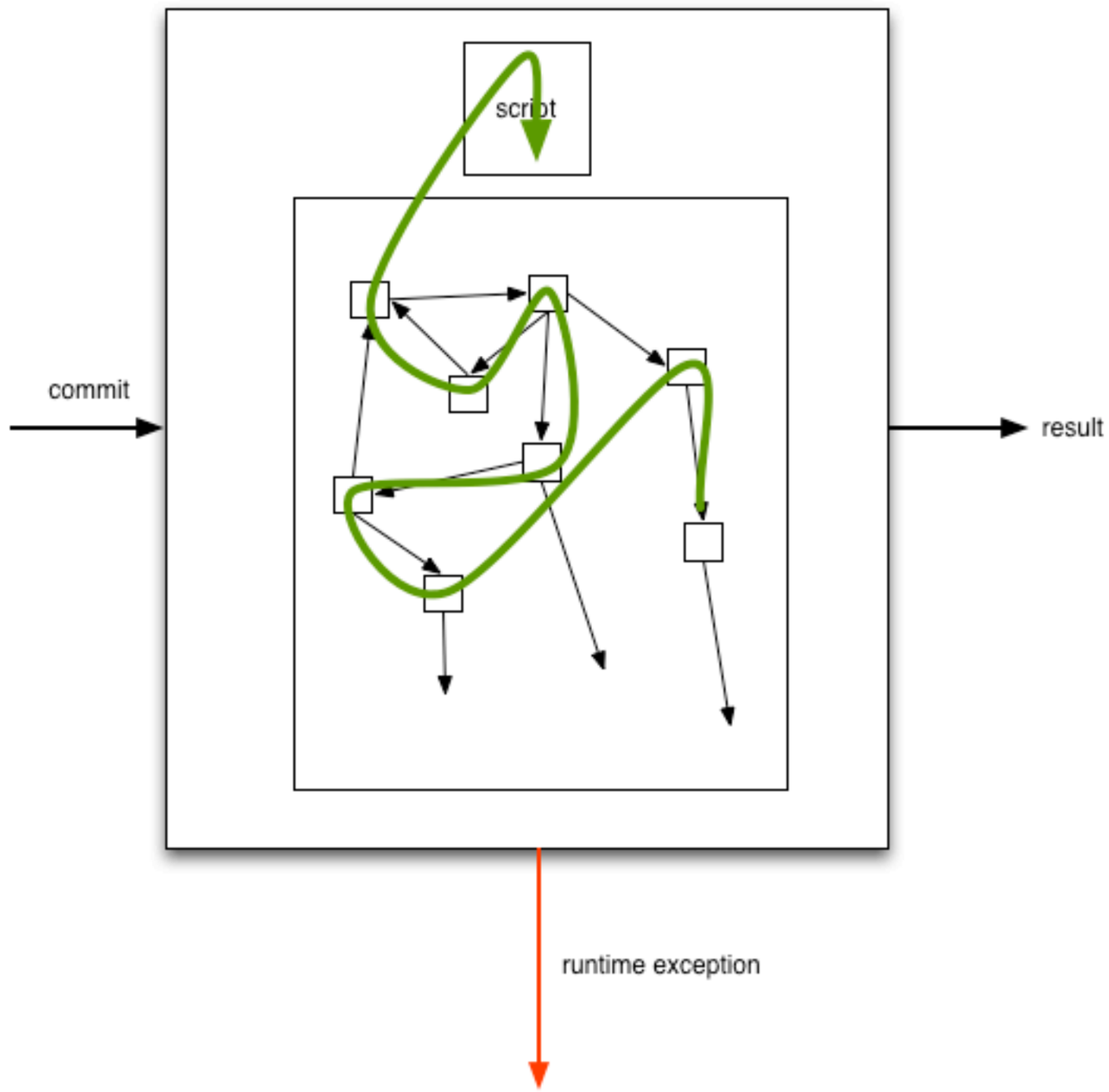
1. Load and process dependencies.

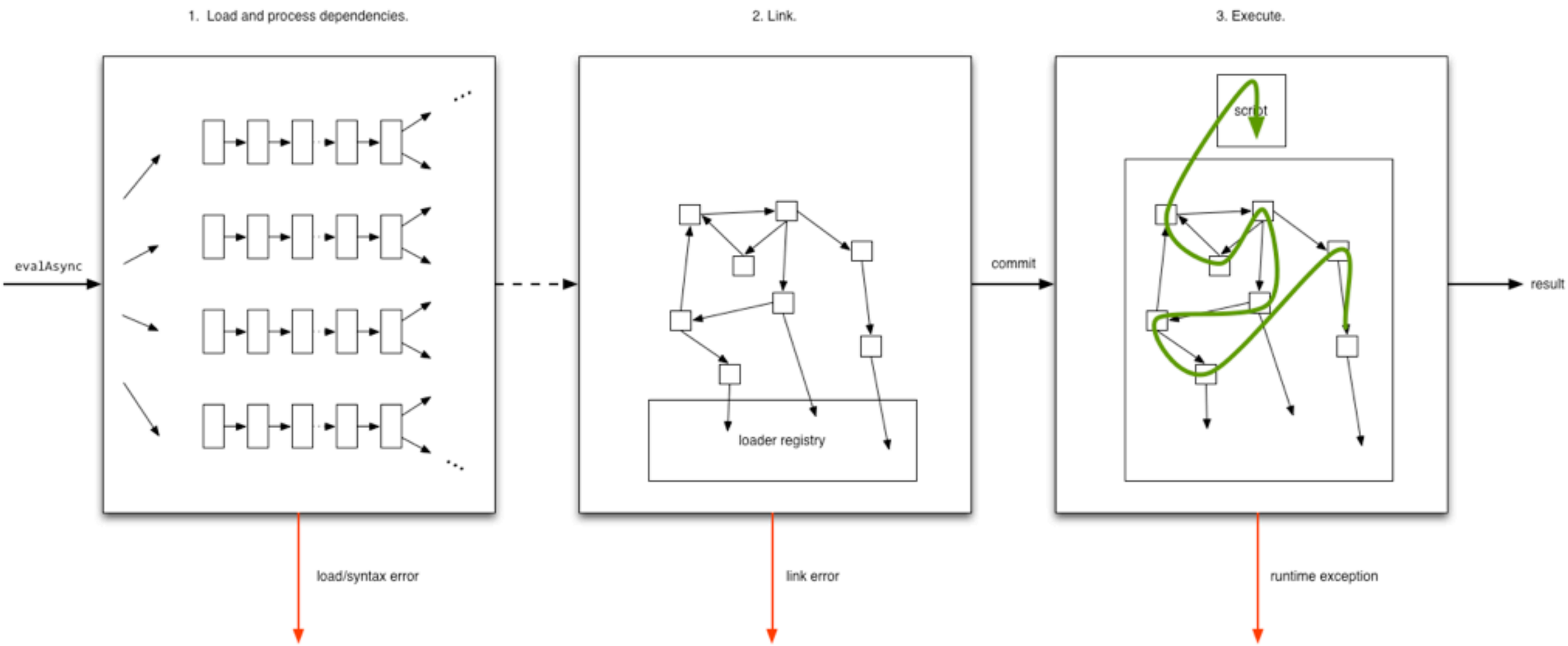


2. Link.



3. Execute.




1. Load and process dependencies.
2. Link.
3. Execute.
evalAsync
commit
result
loader registry
script
load/syntax error
link error
runtime exception