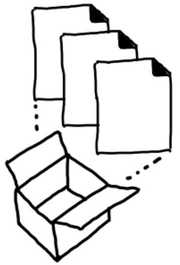
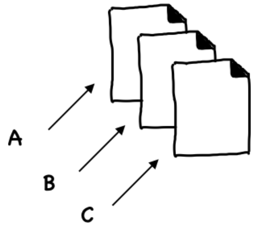
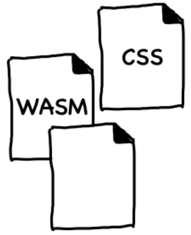
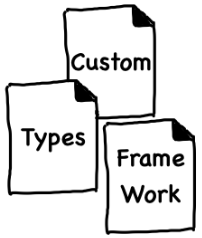
Re-thinking Modules

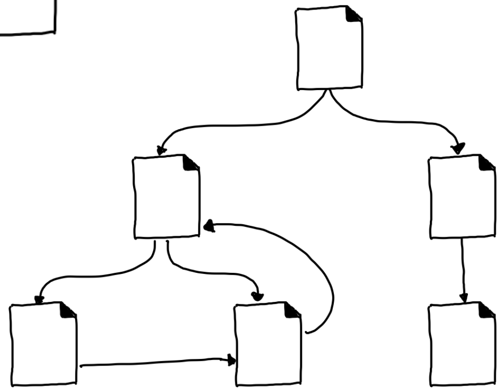


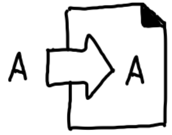
Yulia Startsev | bobkonf 2023

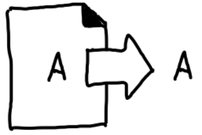


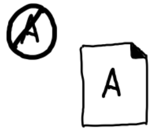


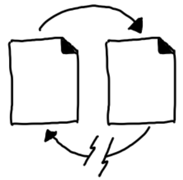






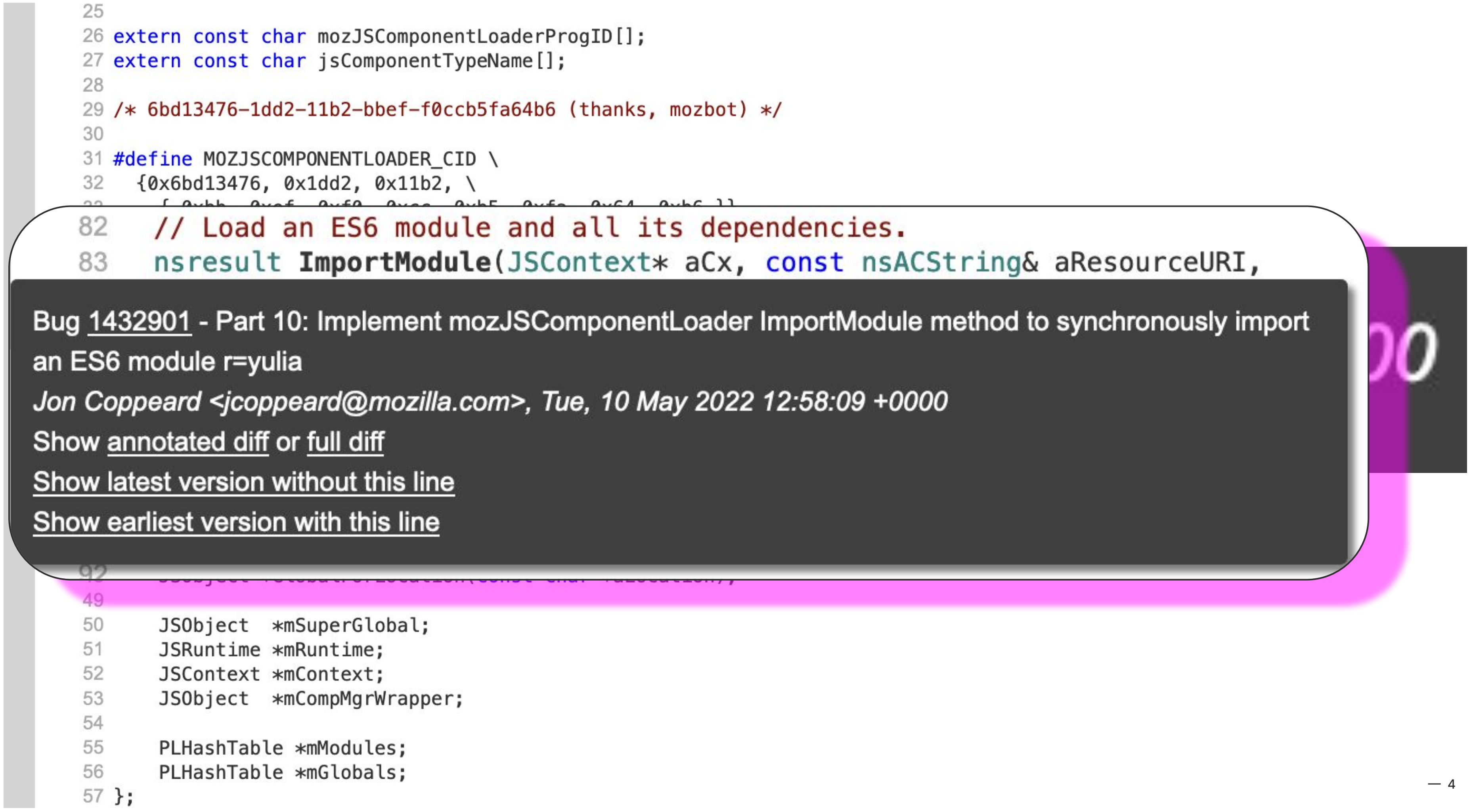






My name is Yulia.
I work on
Spidermonkey.

Short history of JS modules



```
extern const char mozJSComponentLoaderProgID[];
extern const char jsComponentTypeName[];

/* 6bd13476-1dd2-11b2-bbef-f0ccb5fa64b6 (thanks, mozbot) */

#define MOZJSCOMPONENTLOADER_CID \
  {0x6bd13476, 0x1dd2, 0x11b2, \
    { 0xbb, 0xef, 0xf0, 0xcc, 0xb5, 0xfa, 0x64, 0xb6 }}
```

```
  // Load an ES6 module and all its dependencies.
  nsresult ImportModule(JSContext* aCx, const nsACString& aResourceURI,
```

Bug 1432901 - Part 10: Implement mozJSComponentLoader ImportModule method to synchronously import an ES6 module r=yulia

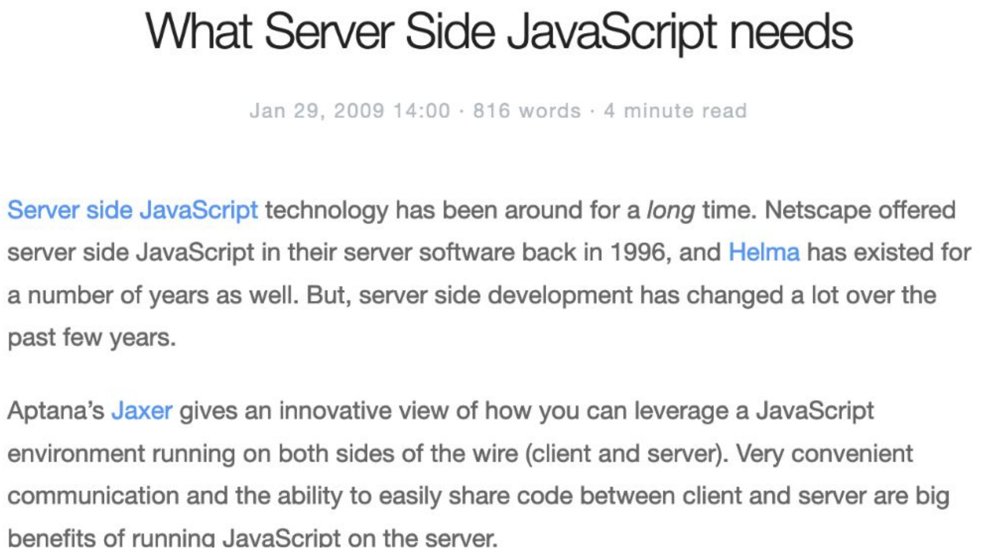
Jon Coppeard <jcoppeard@mozilla.com>, Tue, 10 May 2022 12:58:09 +0000

Show annotated diff or full diff

Show latest version without this line

Show earliest version with this line

```
    JSObject *GlobalForLocation(const char *aLocation);

    JSObject  *mSuperGlobal;
    JSRuntime *mRuntime;
    JSContext *mContext;
    JSObject  *mCompMgrWrapper;

    PLHashTable *mModules;
    PLHashTable *mGlobals;
};
```



What Server Side JavaScript needs

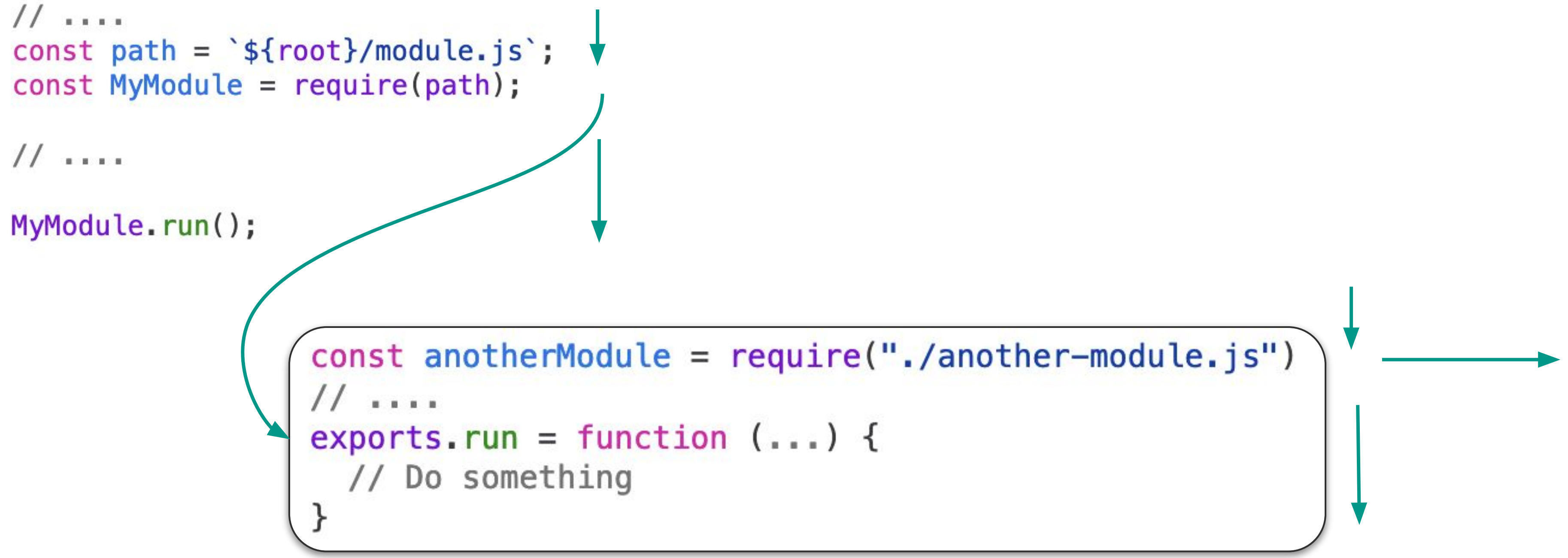
Jan 29, 2009 14:00 · 816 words · 4 minute read

Server side JavaScript technology has been around for a *long* time. Netscape offered server side JavaScript in their server software back in 1996, and Helma has existed for a number of years as well. But, server side development has changed a lot over the past few years.

Aptana's Jaxer gives an innovative view of how you can leverage a JavaScript environment running on both sides of the wire (client and server). Very convenient communication and the ability to easily share code between client and server are big benefits of running JavaScript on the server.

https://www.blueskyonmars.com/2009/01/29/what-server-side-javascript-needs/

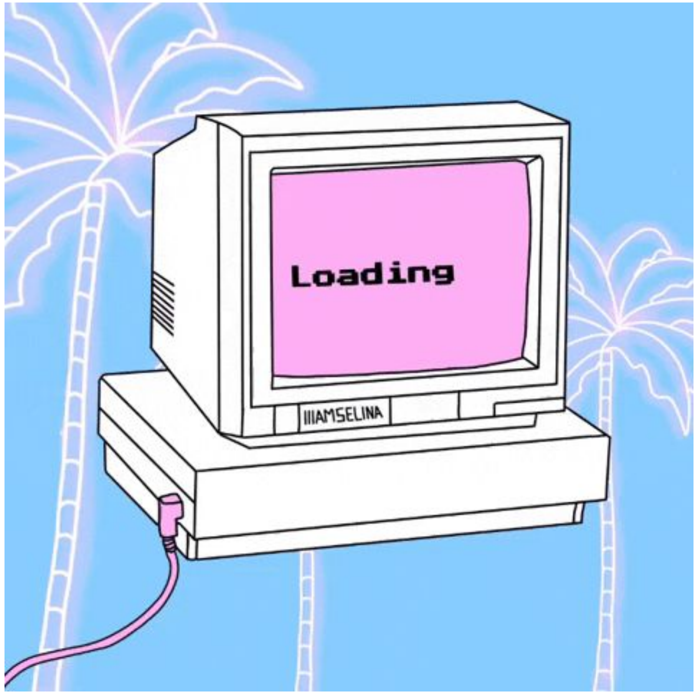
How to *load* a module (CommonJS)



```
// ....
const path = `${root}/module.js`;
const MyModule = require(path);

// ....

MyModule.run();
```

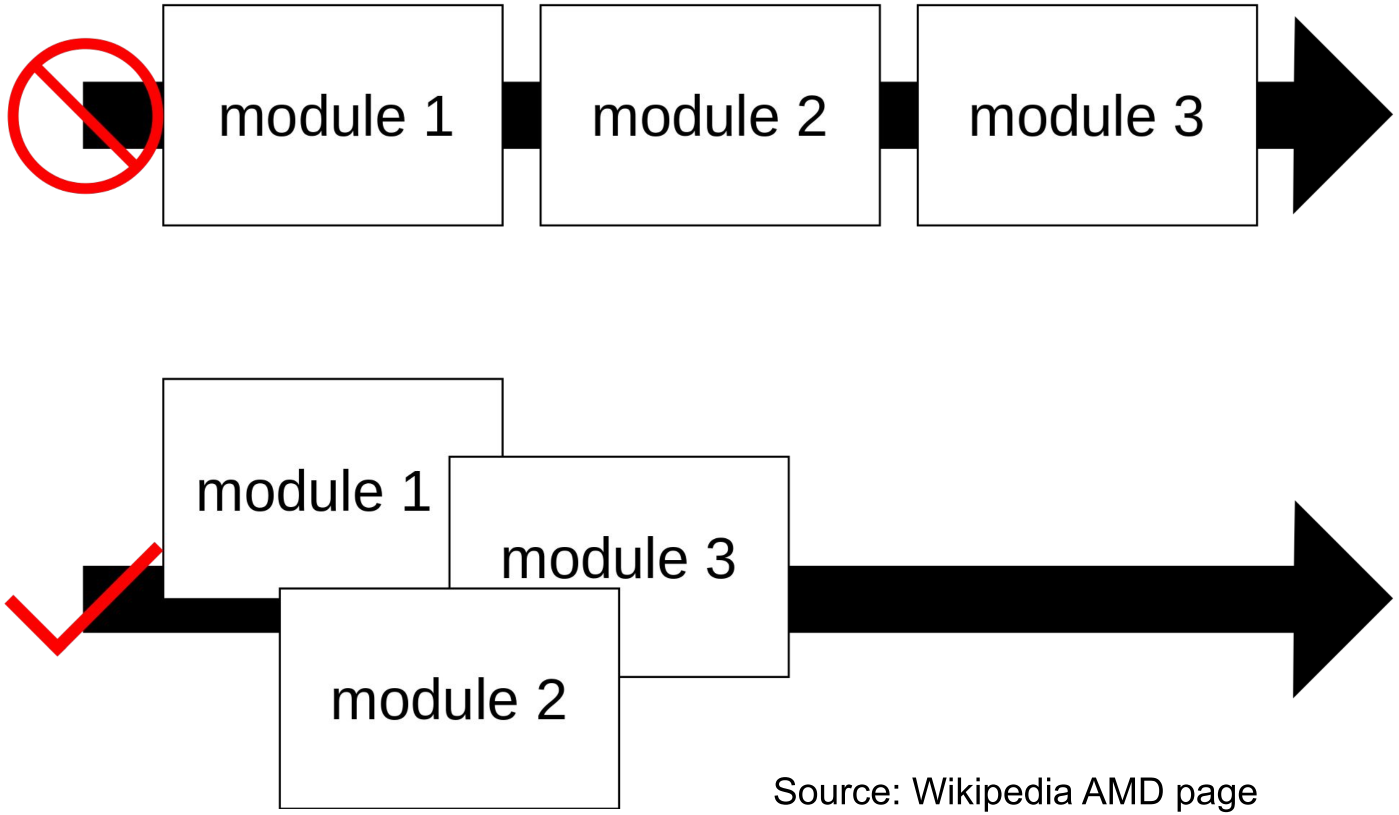
```
const anotherModule = require("./another-module.js")
// ....
exports.run = function (...) {
  // Do something
}
```

Problem: Synchronous loading



- CommonJS was made for server-side JS.
  - Disk access is much faster than network!
- It implemented *Synchronous Loading*
- This doesn’t work very well for the web…

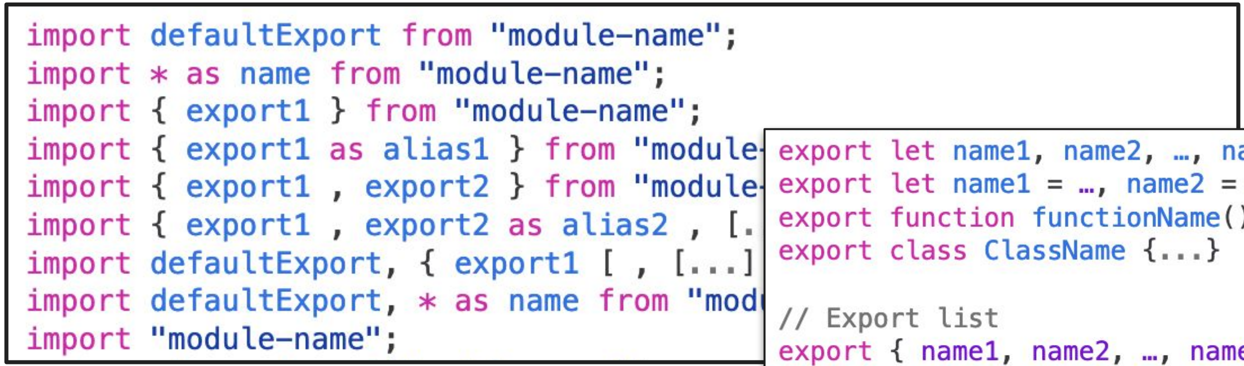
Asynchronous Module Definition (AMD)



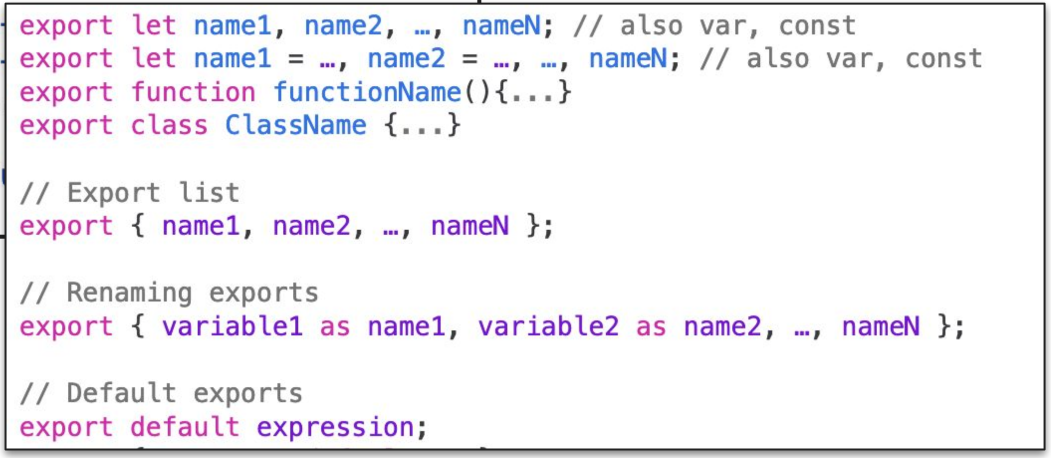
Source: Wikipedia AMD page

6 years later: ES6 Modules

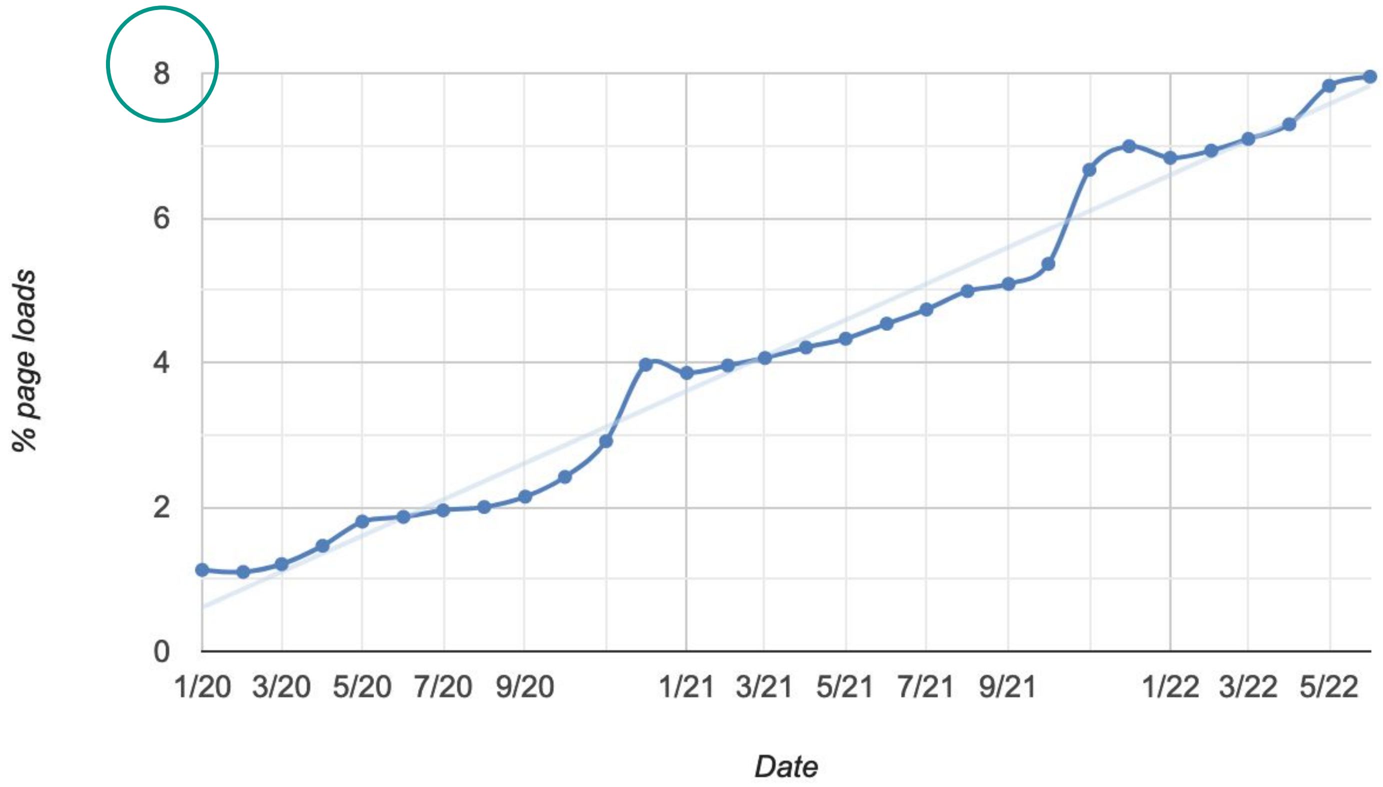
- Published in 2015, implemented by all browsers in 2018



```
import defaultExport from "module-name";
import * as name from "module-name";
import { export1 } from "module-name";
import { export1 as alias1 } from "module-
import { export1 , export2 } from "module-
import { export1 , export2 as alias2 , [.
import defaultExport, { export1 [ , [...]
import defaultExport, * as name from "modu
import "module-name";
```



```
export let name1, name2, …, nameN; // also var, const
export let name1 = …, name2 = …, …, nameN; // also var, const
export function functionName(){...}
export class ClassName {...}

// Export list
export { name1, name2, …, nameN };

// Renaming exports
export { variable1 as name1, variable2 as name2, …, nameN };

// Default exports
export default expression;
```

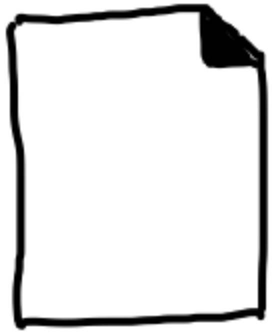

8
6
4
2
0
% page loads
1/20
3/20
5/20
7/20
9/20
1/21
3/21
5/21
7/21
9/21
1/22
3/22
5/22
Date

Agenda

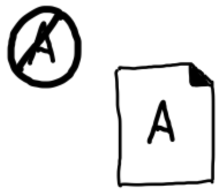
- Prelude (history, standards)
- How does the module system work?
- Why is it not being used more?
- Solution space!

(There are 96 slides. sorry!)

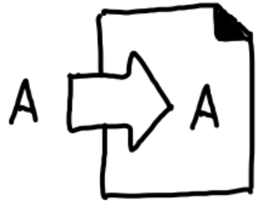
Ok. Let's talk modules.



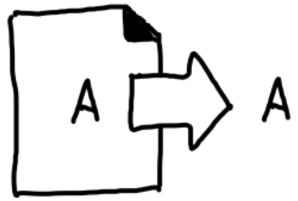
How do ES Modules Work?



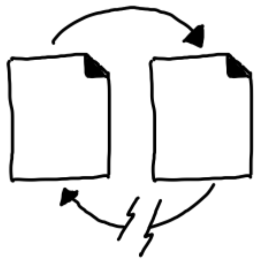
Namespacing



Importing objects



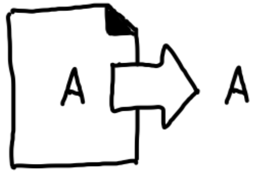
Exposing objects


Cycle breaking

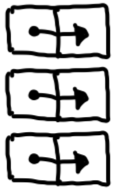
How do ES Modules Work?

```
window.A = "A"
window.B = "B"
      .
      .
      .
window.Z = "Z"
window.A = "A"
```

How do ES Modules Work?



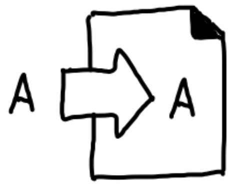
Exposing objects



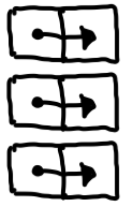
Export Entries

```
export const A = "AAAAA";
```

How do ES Modules Work?



Importing objects



Import entries

```
import a from "./a.js"
```

How to *load* a module (ES6)

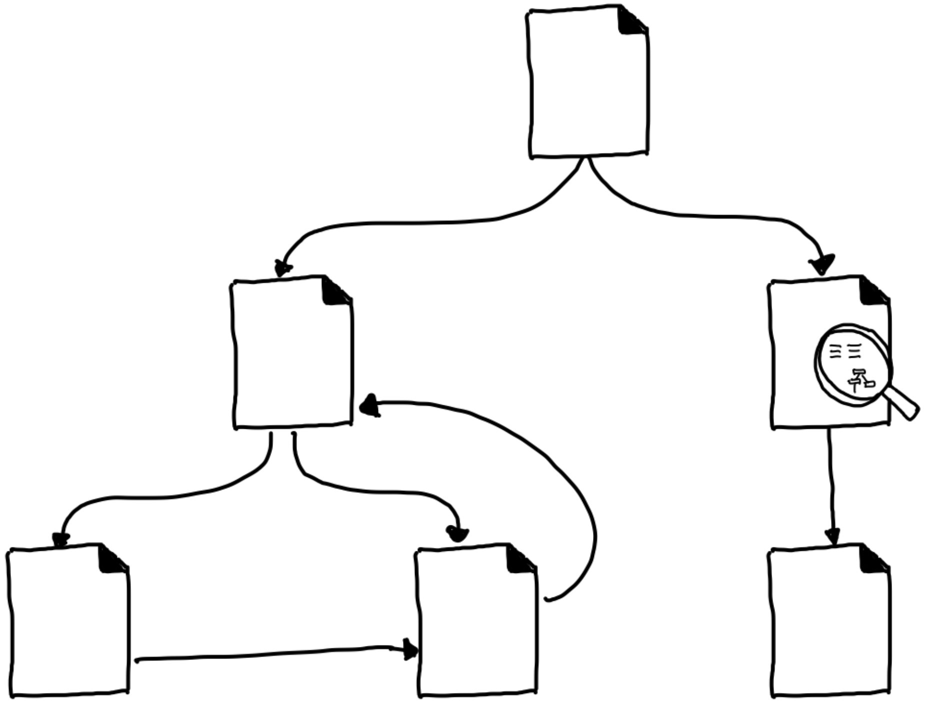
```
import MyModule from "./path.js"

// ...

MyModule.run()
```

Imports		Exports	
MyModule	“www.com /path.js”		

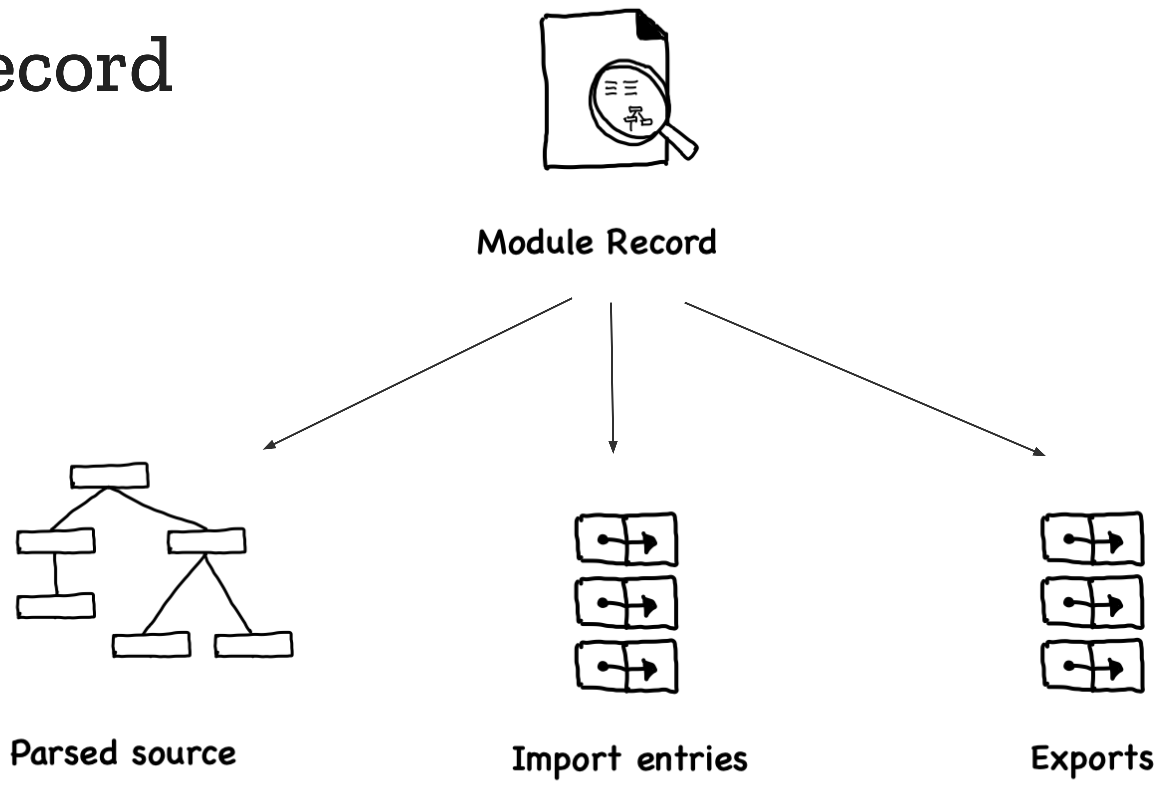
Imports		Exports	
anotherModule	“...”	run	run()

```
import anotherModule from "./another-module.js"

// ...

function run() { /*...*/ }

export default { run }
```

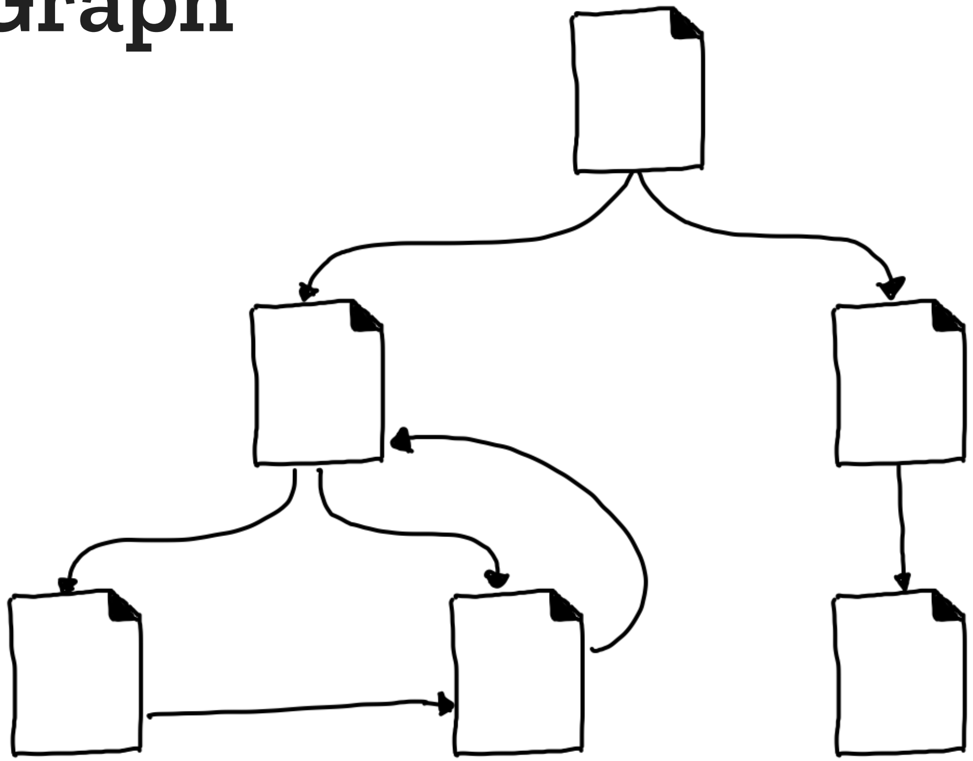
How do ES Modules Work?

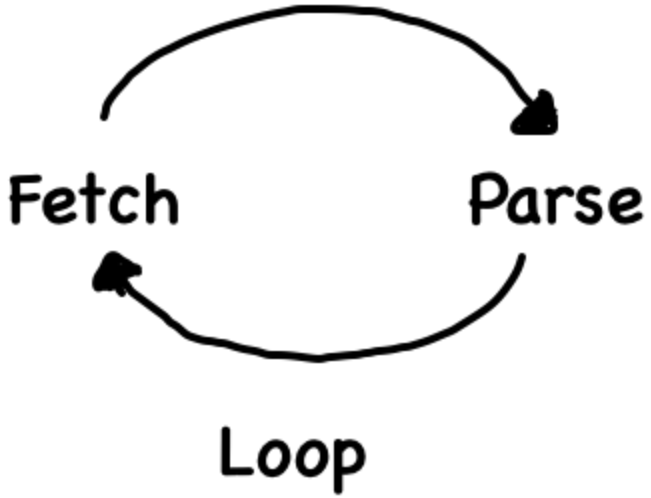


The Module Record

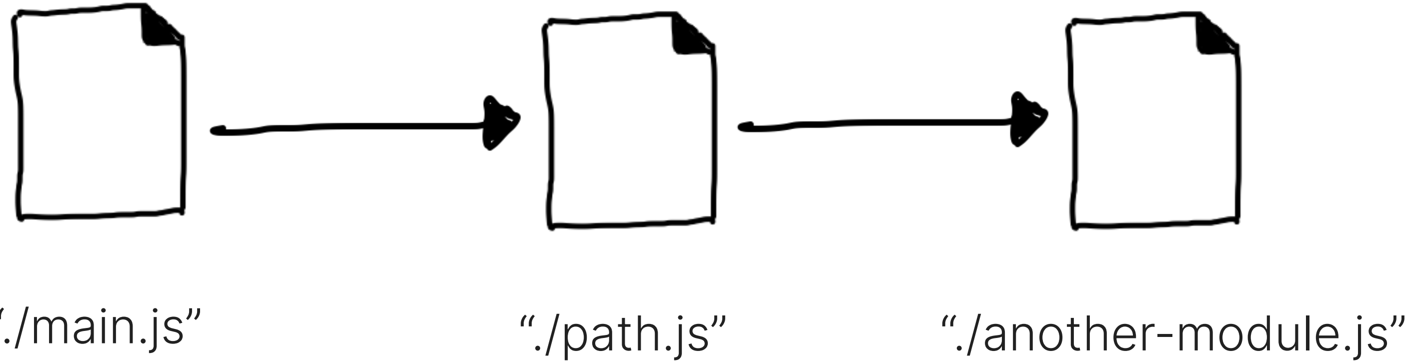


The ES Module Graph




Fetch
Parse
Loop

The Module Map



The Module Map

Resolved to URL Specifier	State
“./main.js”	unlinked
“./path.js”	unlinked
“./another-module.js”	unlinked

The Module Map

Resolved to URL Specifier	State
“./main.js”	**fetching**
“./path.js”	unlinked
“./another-module.js”	unlinked

The Module Map

Resolved to URL Specifier	State
“./main.js”	**fetched**
“./path.js”	unlinked
“./another-module.js”	unlinked

The Module Map

Resolved to URL Specifier	State
“./main.js”	**linking**
“./path.js”	unlinked
“./another-module.js”	unlinked

The Module Map

Resolved to URL Specifier	State
“./main.js”	linking
“./path.js”	**linking**
“./another-module.js”	unlinked

The Module Map

Resolved to URL Specifier	**State**
“./main.js”	linking
“./path.js”	linking
“./another-module.js”	**linking**

The Module Map

Resolved to URL Specifier	State
“./main.js”	linking
“./path.js”	linking
“./another-module.js”	**linked**

The Module Map

Resolved to URL Specifier	**State**
"./main.js"	linking
"./path.js"	**linked**
"./another-module.js"	linked

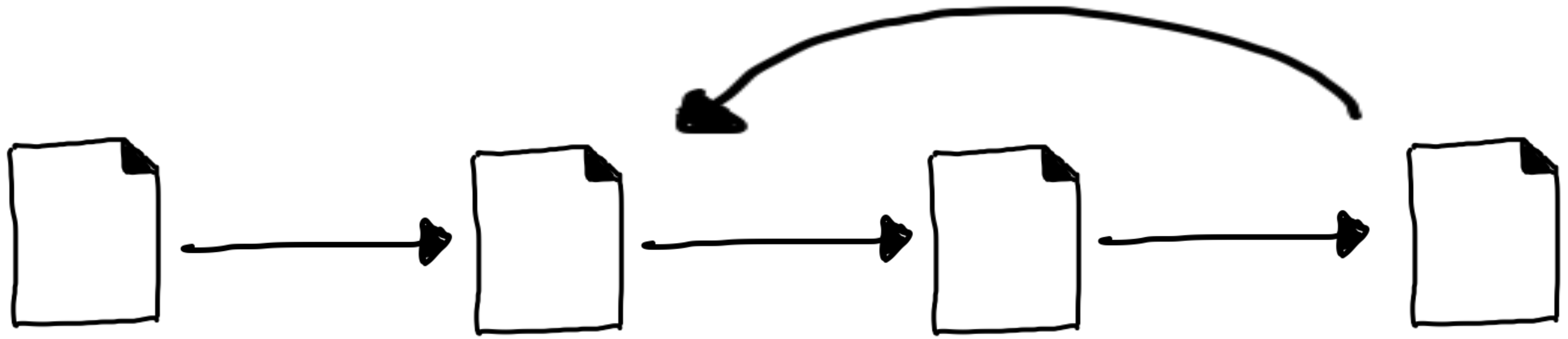
The Module Map

Resolved to URL Specifier	**State**
“./main.js”	**linked**
“./path.js”	linked
“./another-module.js”	linked

The Module Map

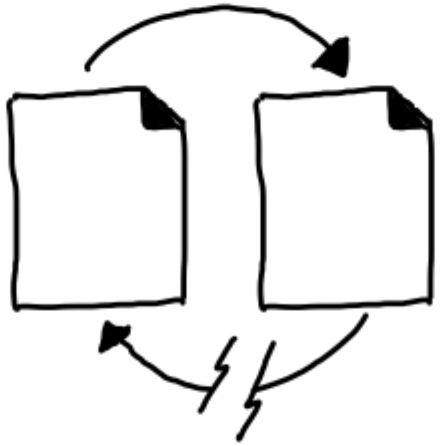
Resolved to URL Specifier	**State**
"./main.js"	linked
"./path.js"	linked
"./another-module.js"	linked
"./import-path.js"	**unlinked**

The Module Map



“./main.js” “./path.js” “./another-module.js” “./import-path.js”

The Module Map: Cycle breaking

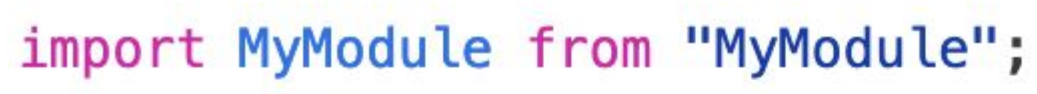


The Module Map: Cycle breaking

Resolved to URL Specifier	**State**
"./main.js"	linked
"./path.js"	**linked**
"./another-module.js"	linked
"./import-path.js"	**linking**

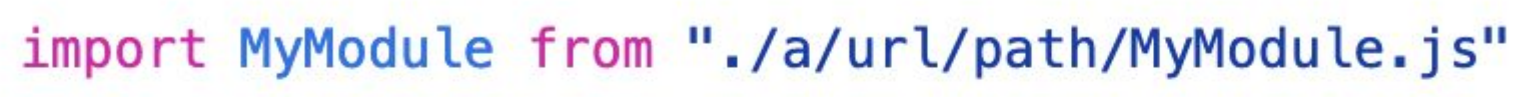
The Module Map: Cycle breaking

Resolved to URL Specifier	State
“./main.js”	linked
“./path.js”	**linked**
“./another-module.js”	linked
“./import-path.js”	**linked**



```
import MyModule from "MyModule";
```

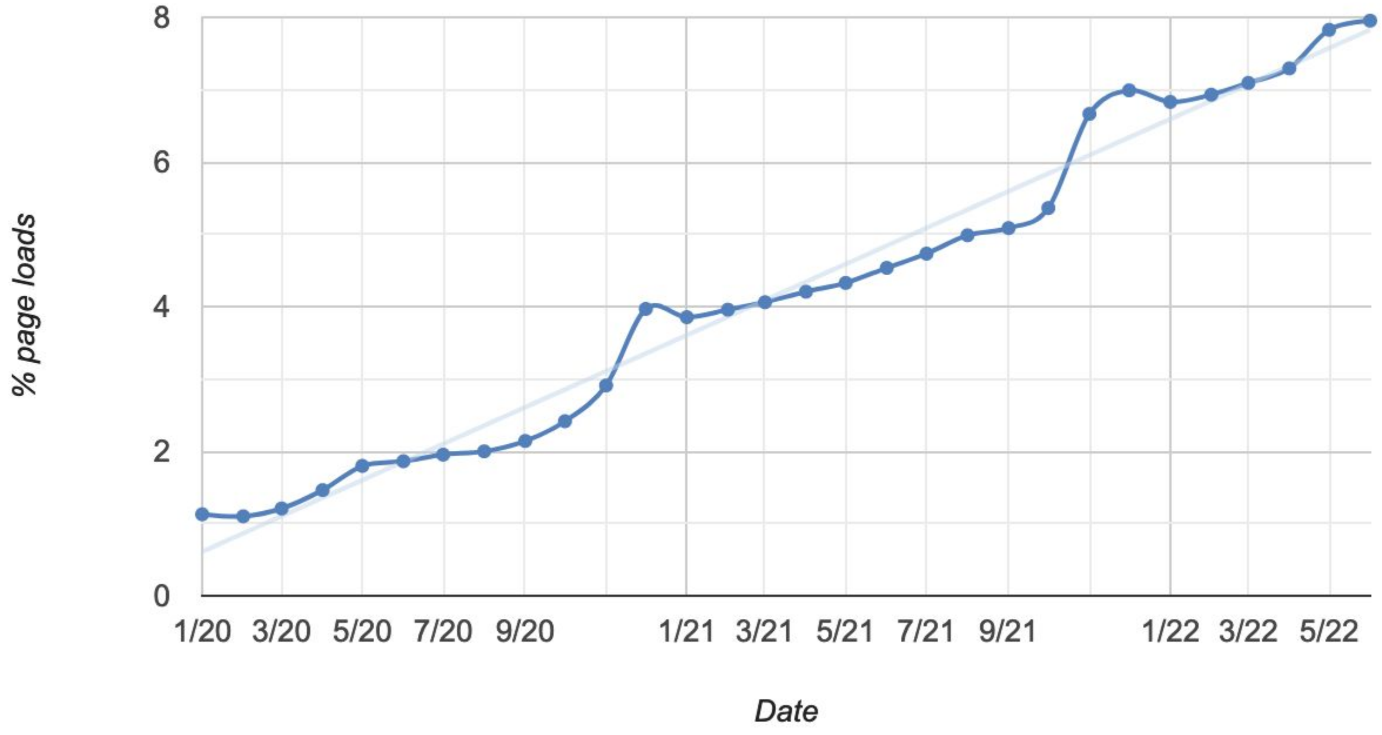
Why URLs?



```
import MyModule from "./a/url/path/MyModule.js"
```

Summary

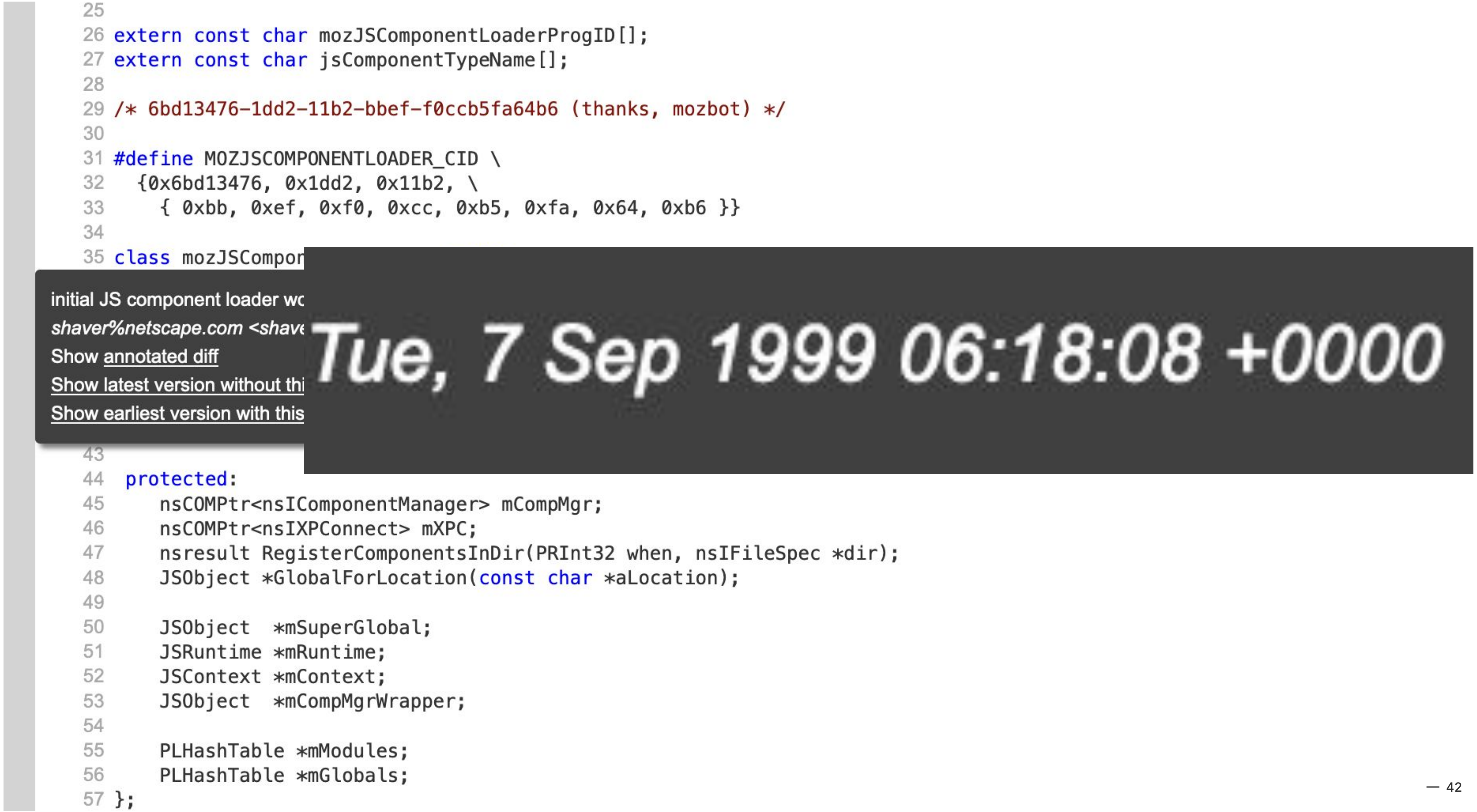
- The module system is
  - Async & Non-interrupting
  - Allows cyclic dependencies
  - Relies on the host’s security mechanisms
  - A global module map



8
6
4
2
0
% page loads
1/20
3/20
5/20
7/20
9/20
1/21
3/21
5/21
7/21
9/21
1/22
3/22
5/22
Date

Why?

Exploring the Firefox case



```

extern const char mozJSComponentLoaderProgID[];
extern const char jsComponentTypeName[];

/* 6bd13476-1dd2-11b2-bbef-f0ccb5fa64b6 (thanks, mozbot) */

#define MOZJSCOMPONENTLOADER_CID \
  {0x6bd13476, 0x1dd2, 0x11b2, \
    { 0xbb, 0xef, 0xf0, 0xcc, 0xb5, 0xfa, 0x64, 0xb6 }}

class mozJSCompon

 protected:
    nsCOMPtr<nsIComponentManager> mCompMgr;
    nsCOMPtr<nsIXPConnect> mXPC;
    nsresult RegisterComponentsInDir(PRInt32 when, nsIFileSpec *dir);
    JSObject *GlobalForLocation(const char *aLocation);

    JSObject  *mSuperGlobal;
    JSRuntime *mRuntime;
    JSContext *mContext;
    JSObject  *mCompMgrWrapper;

    PLHashTable *mModules;
    PLHashTable *mGlobals;
};
```

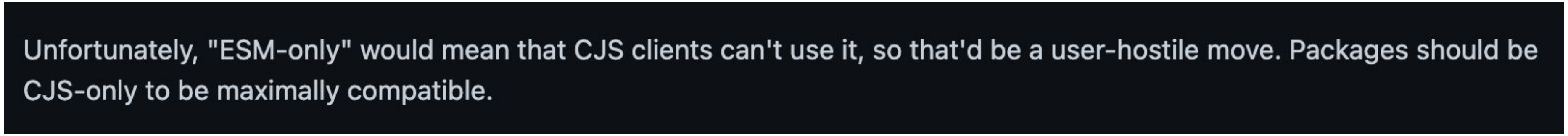
initial JS component loader wc
shaver%netscape.com <shave
Show annotated diff
Show latest version without thi
Show earliest version with this

Tue, 7 Sep 1999 06:18:08 +0000

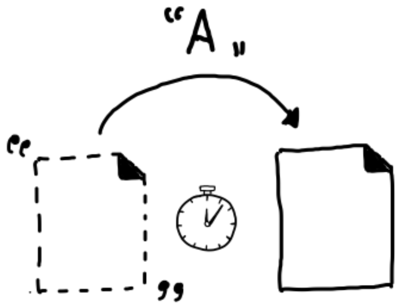
Deferred Module Evaluation

```js
import aMethod from "./a.js";

function rarelyUsedA() {
  // ...
  const aValue = aMethod();
}

function alsoRarelyUsedA() {
  // ...
  const aValue = aMethod();
}

// ...

function eventuallyCalled() {
  rarelyUsedA();
}
```

```js
async function lazyAMethod(...args) {
   const aMethod = await import("./a.js");
   return aMethod(...args);
}

async function rarelyUsedA() {
  // ...
  const aValue = await aMethod();
}

async function alsoRarelyUsedA() {
  // ...
  const aValue = await aMethod();
}

// ...

function eventuallyCalled() {
  rarelyUsedA();
}
```

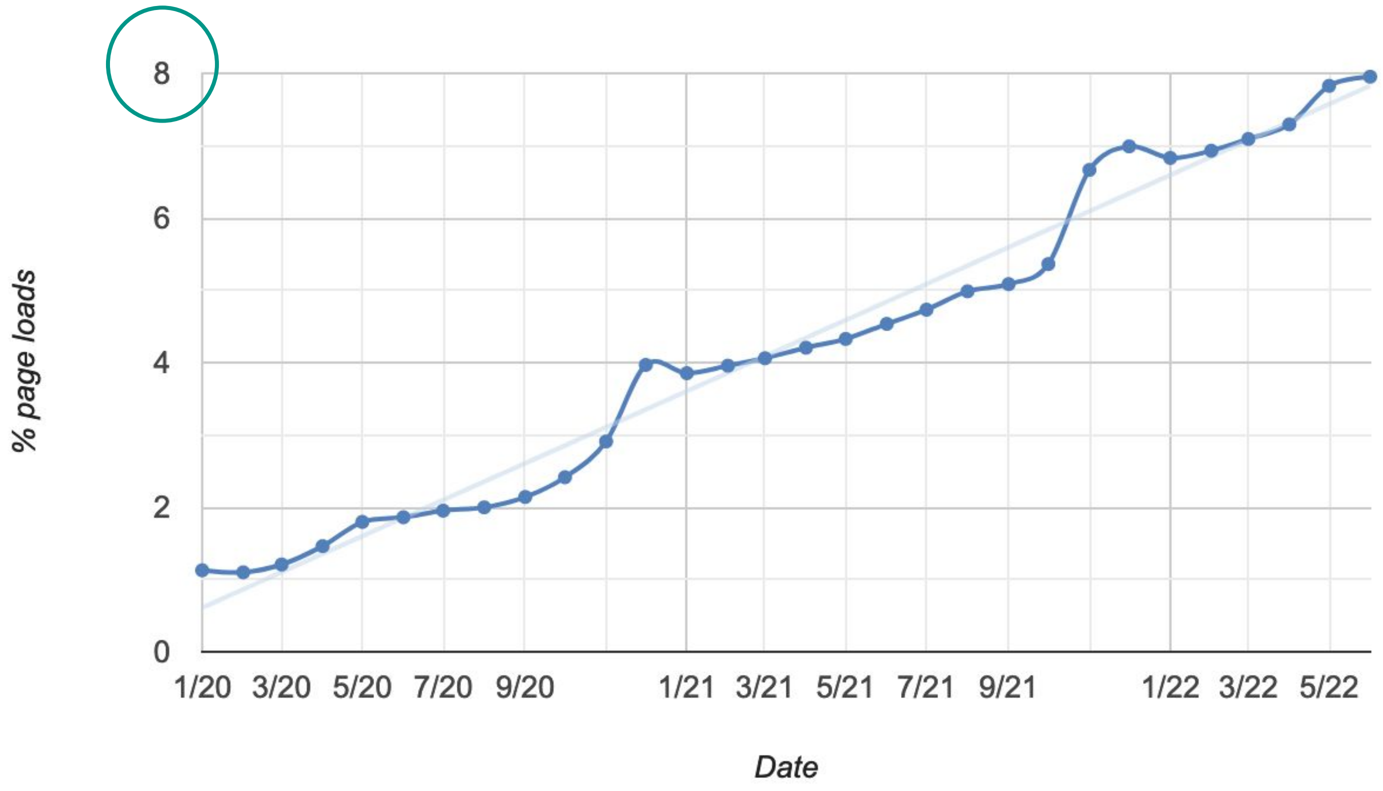


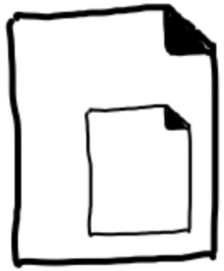
Unfortunately, "ESM-only" would mean that CJS clients can't use it, so that'd be a user-hostile move. Packages should be CJS-only to be maximally compatible.



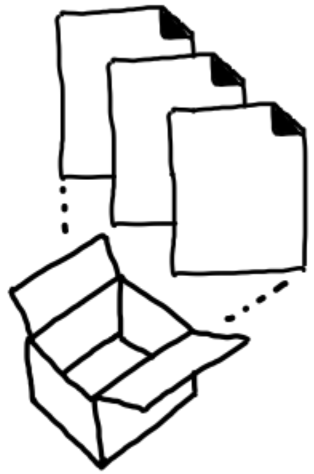
Deferred module evaluation

Exploring the Bundler case


8
6
4
2
0
% page loads
1/20
3/20
5/20
7/20
9/20
1/21
3/21
5/21
7/21
9/21
1/22
3/22
5/22
Date



In-line modules



Better bundling

Exploring the WASM case

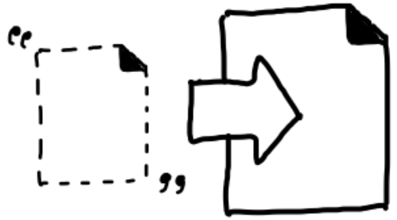
Import Reflection

```
import module x from "<specifier>"
```

Import Reflection

```
import module FooModule from "./foo.wasm";
FooModule instanceof WebAssembly.Module; // true

// For example, to run a WASI execution with an API like Node.js WASI:
import { WASI } from 'wasi';
const wasi = new WASI({ args, env, preopens });

const fooInstance = await WebAssembly.instantiate(FooModule, {
  wasi_snapshot_preview1: wasi.wasiImport
});

wasi.start(fooInstance);
```

import module "x"

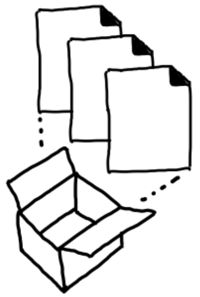


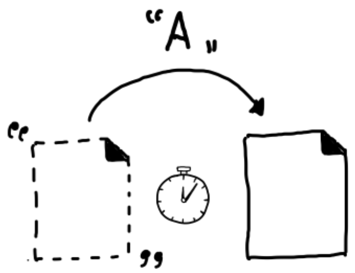
Import reflection

Limitations Summary

- No upgrade path from CommonJS!
- Loading time takes much more time than bundles
- Custom loading patterns are not supported

Components (WASM)


Better bundling

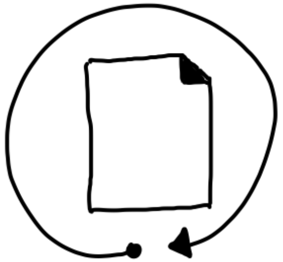


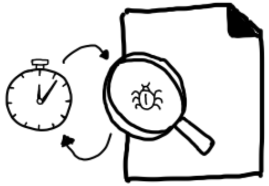
Deferred module evaluation

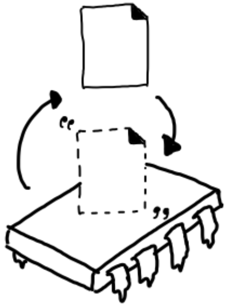
But wait, there's more!

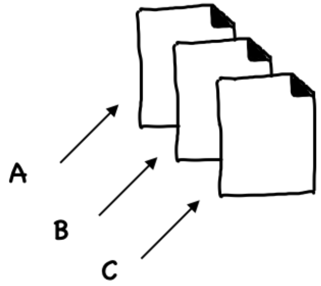
- The module system cannot:
    - Stop part way during instantiation
    - Re-instantiate a module
    - Remove a module (GC)

But wait, there's more!


Hot reloading

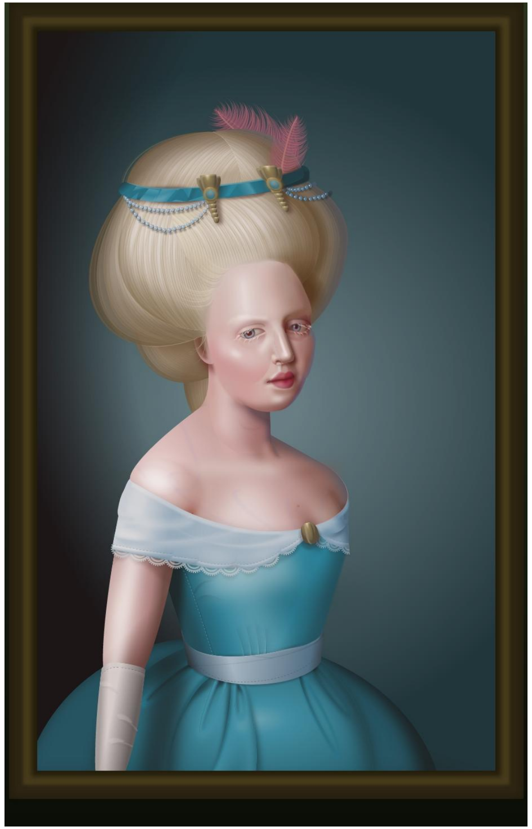

Persistent testing

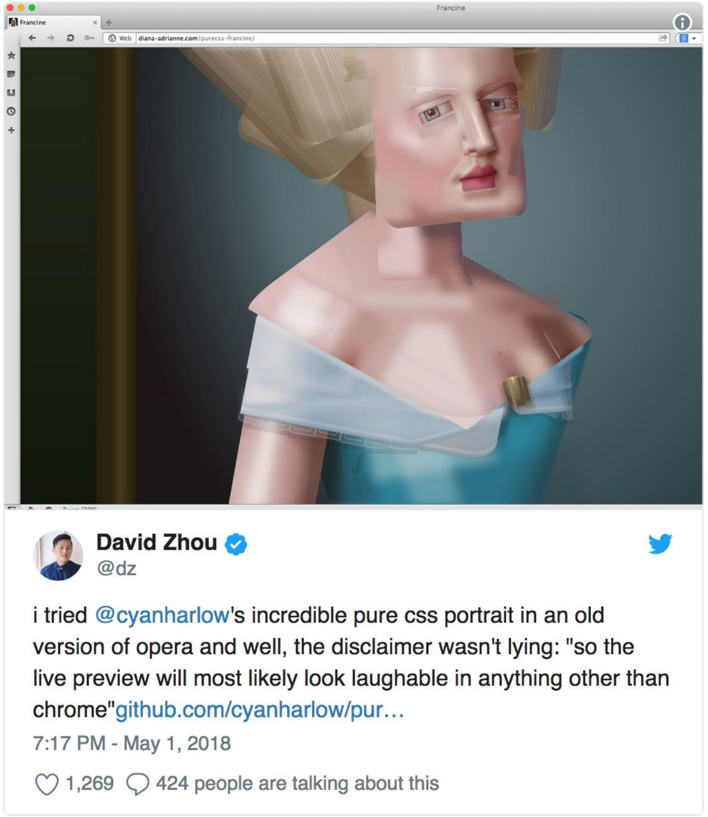

Low memory module reuse



Import map generation

How do we get our module system to do all this?



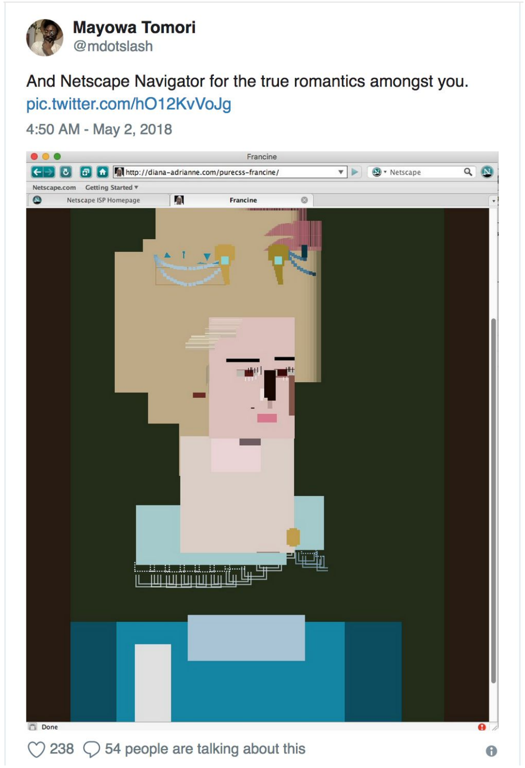


David Zhou
@dz

i tried @cyanharlow's incredible pure css portrait in an old version of opera and well, the disclaimer wasn't lying: "so the live preview will most likely look laughable in anything other than chrome"github.com/cyanharlow/pur…

7:17 PM - May 1, 2018

1,269 424 people are talking about this



Mayowa Tomori
@mdotslash

And Netscape Navigator for the true romantics amongst you.
pic.twitter.com/hO12KvVoJg

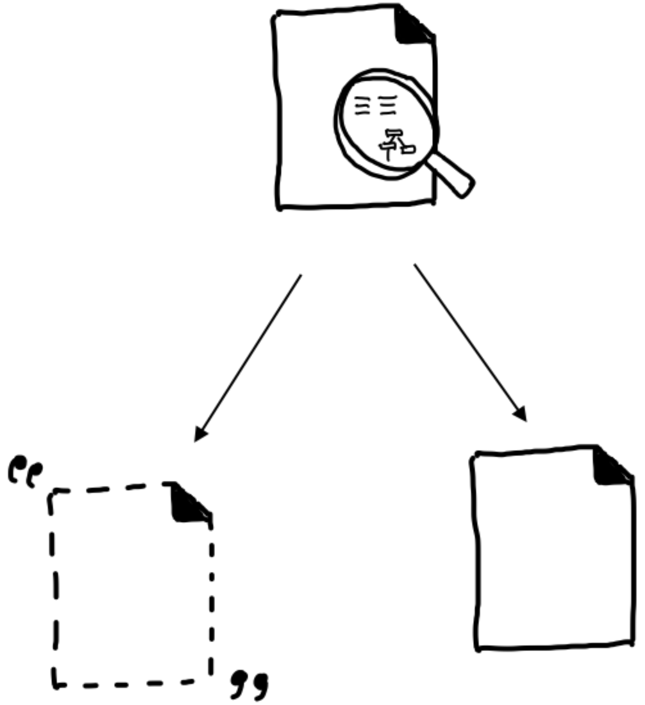
4:50 AM - May 2, 2018

Francine
http://diana-adrianne.com/purecss-francine/
Netscape
Netscape.com Getting Started
Netscape ISP Homepage
Francine
Done

238 54 people are talking about this

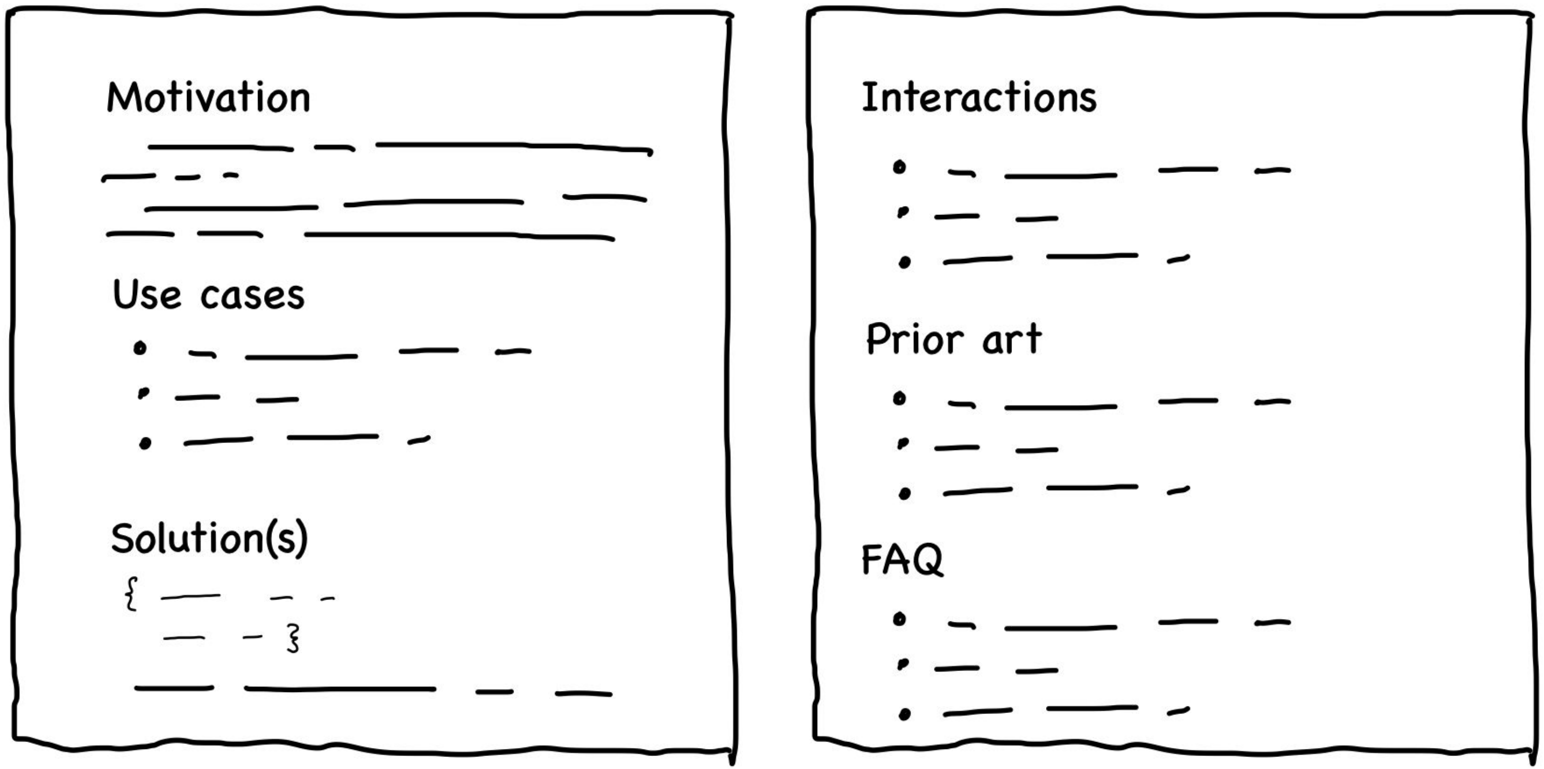
Pure CSS Francine by Diana Smith

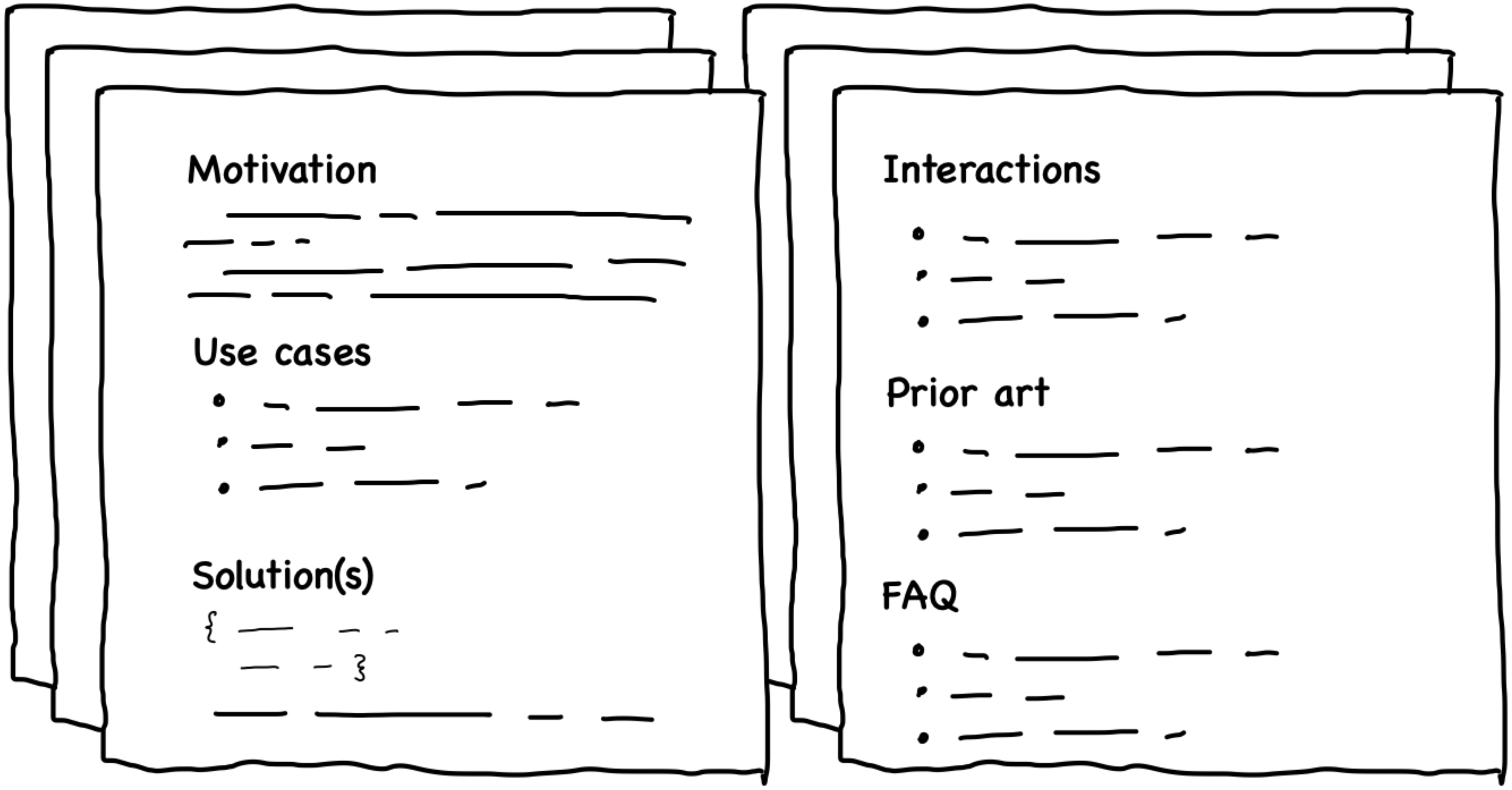
How do we get our module system to do all this?



TC
39

A Proposal

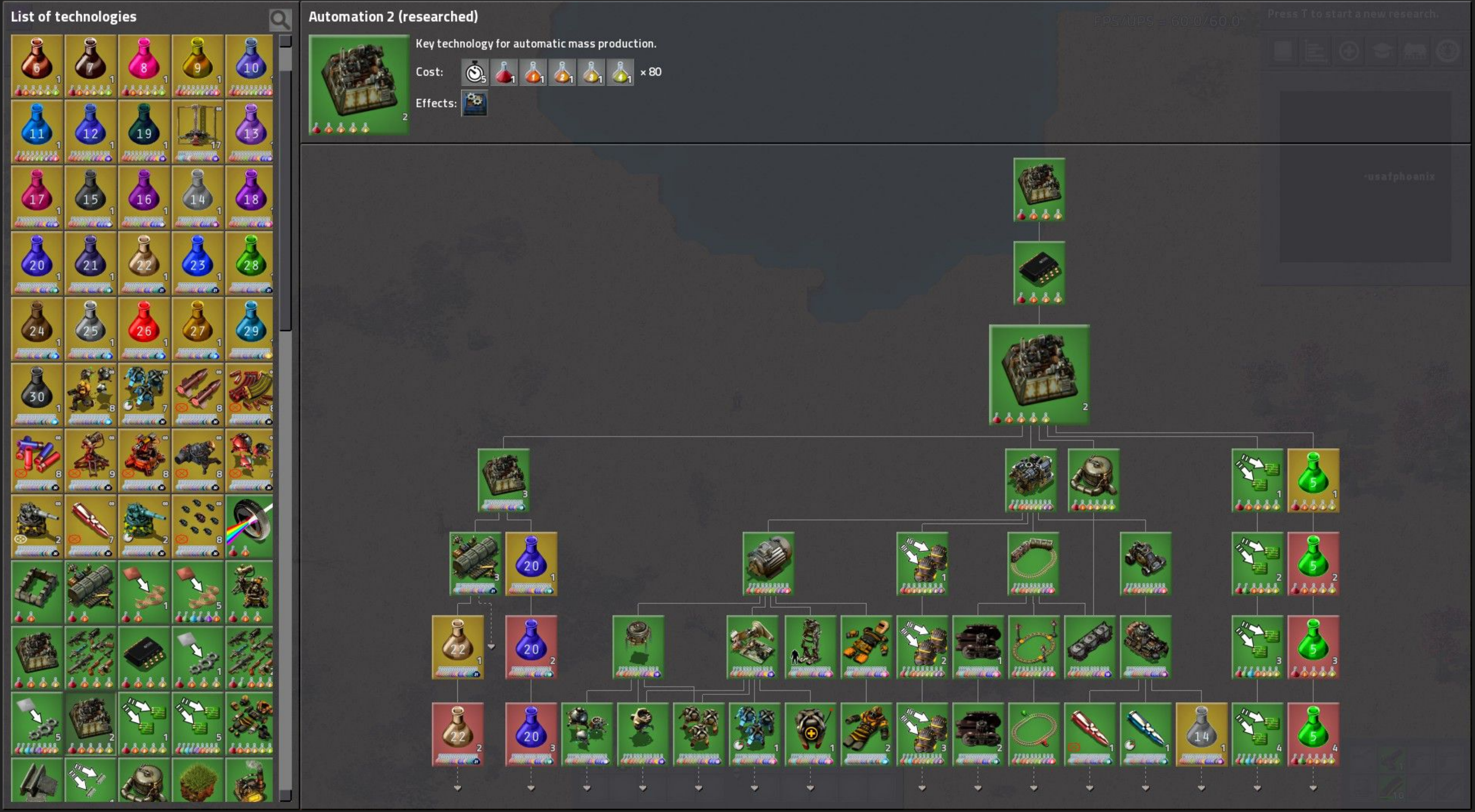

Motivation
Use cases
Solution(s)
Interactions
Prior art
FAQ

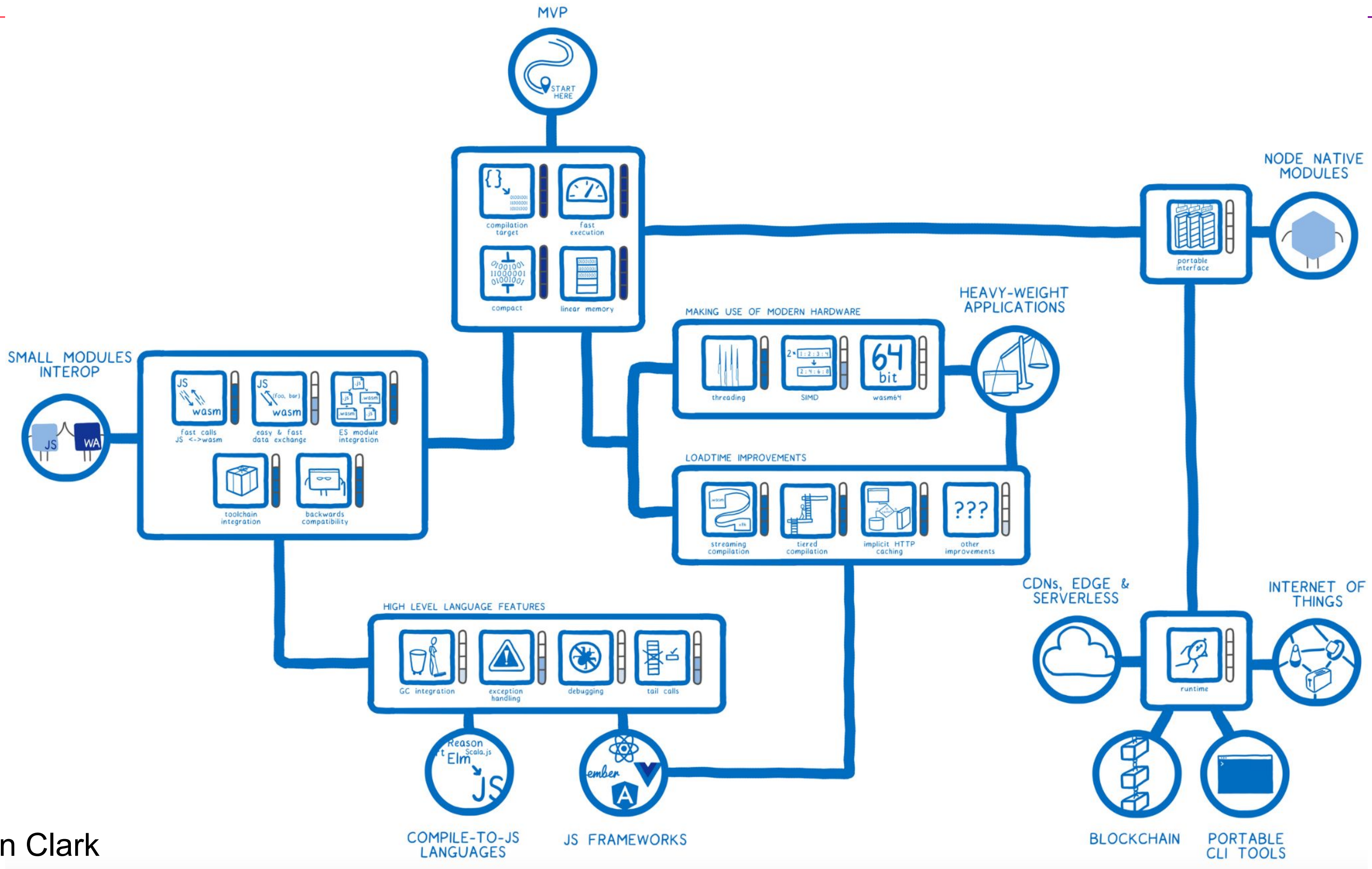

Motivation
Use cases
Solution(s)
{ }
Interactions
Prior art
FAQ

Structured Information & Facilitating Critique

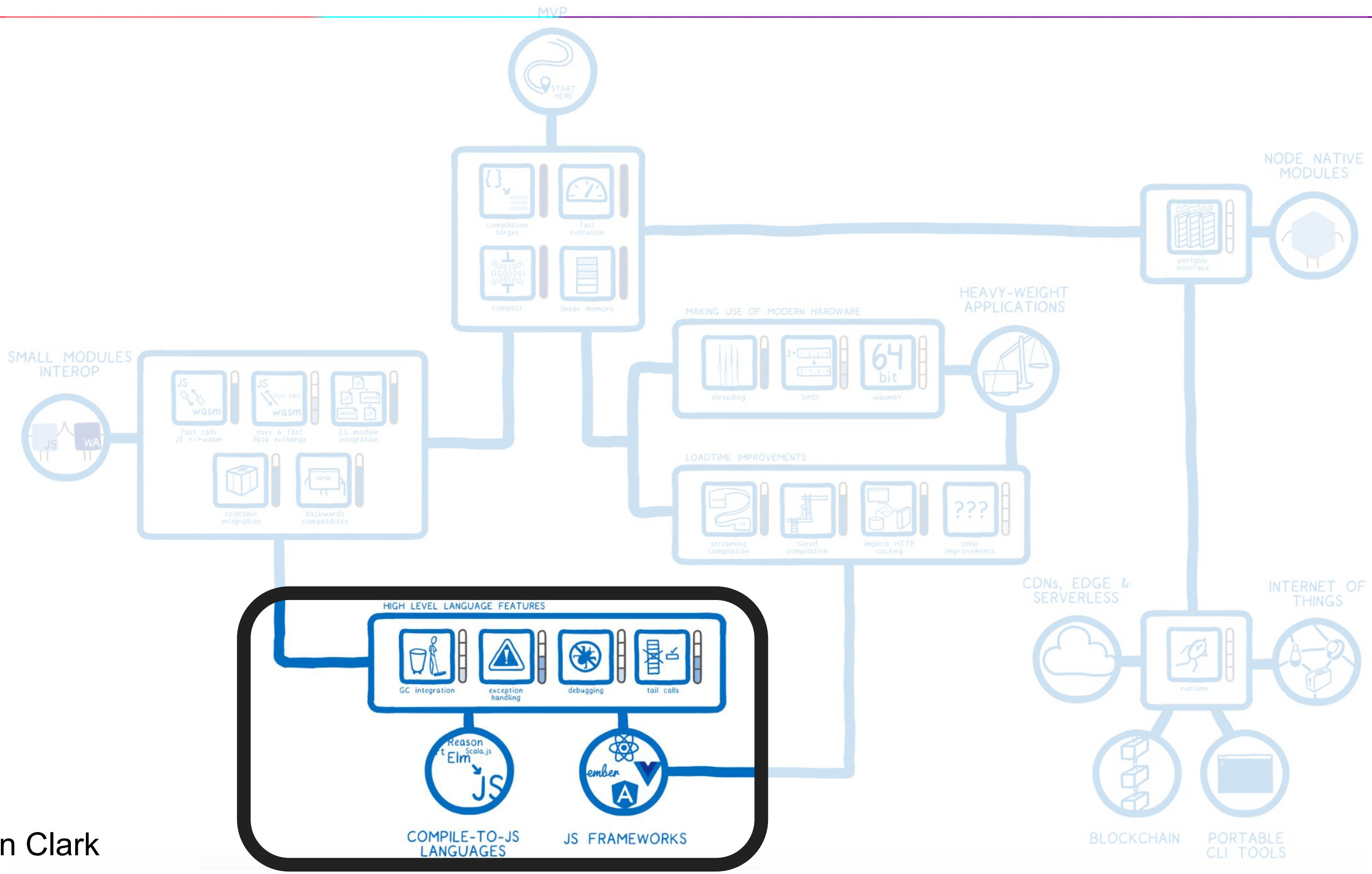
Software architecture is there to deal with all the *how* questions.

- Author forgotten by my husband

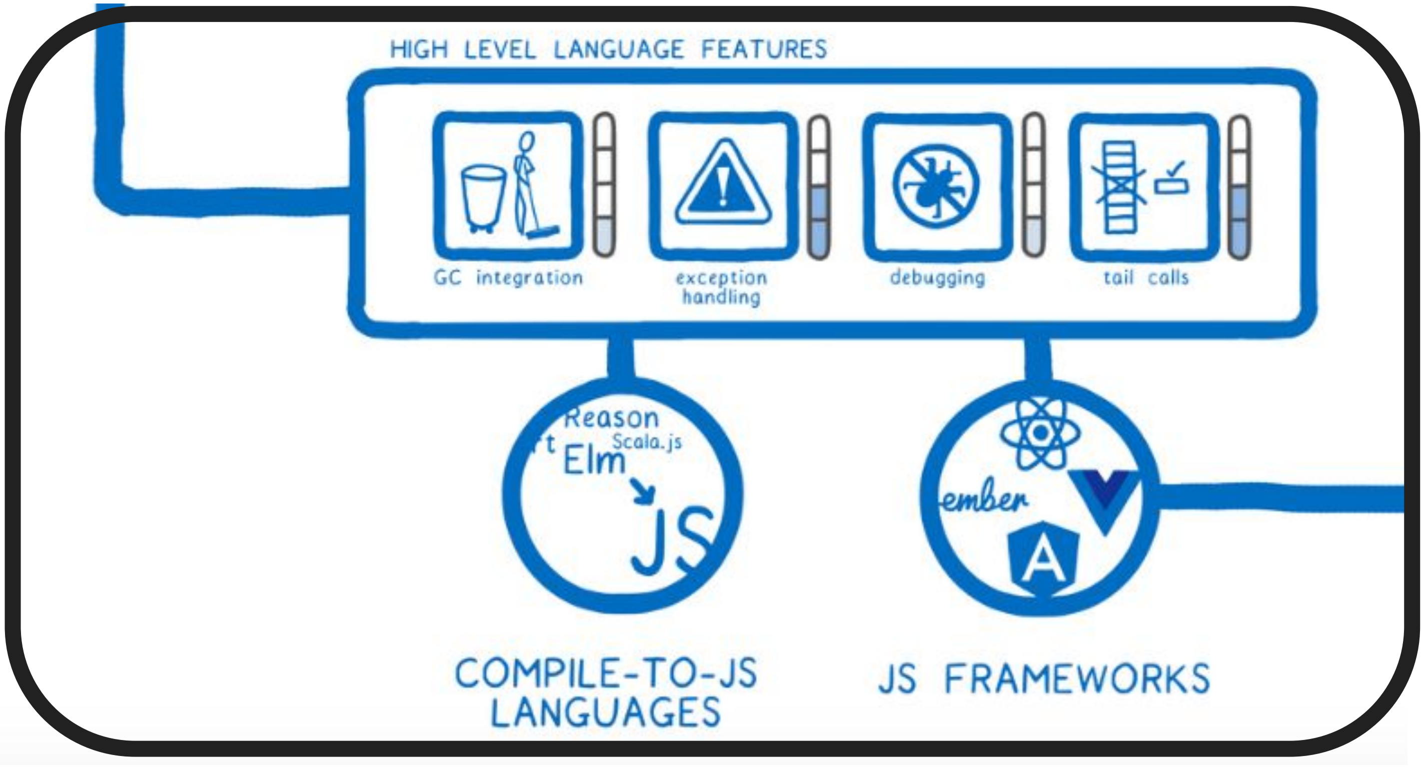

List of technologies
Automation 2 (researched)
Key technology for automatic mass production.
Cost:
× 80
Effects:
Press T to start a new research.
usafphoenix


MVP
START HERE
compilation target
fast execution
compact
linear memory
SMALL MODULES INTEROP
JS
WA
fast calls JS <->wasm
easy & fast data exchange
ES module integration
toolchain integration
backwards compatibility
MAKING USE OF MODERN HARDWARE
threading
SIMD
64 bit
wasm64
HEAVY-WEIGHT APPLICATIONS
LOADTIME IMPROVEMENTS
streaming compilation
tiered compilation
implicit HTTP caching
???
other improvements
NODE NATIVE MODULES
portable interface
HIGH LEVEL LANGUAGE FEATURES
GC integration
exception handling
debugging
tail calls
Reason
Scala.js
Elm
JS
COMPILE-TO-JS LANGUAGES
ember
JS FRAMEWORKS
CDNs, EDGE & SERVERLESS
runtime
INTERNET OF THINGS
BLOCKCHAIN
PORTABLE CLI TOOLS

By: Lin Clark

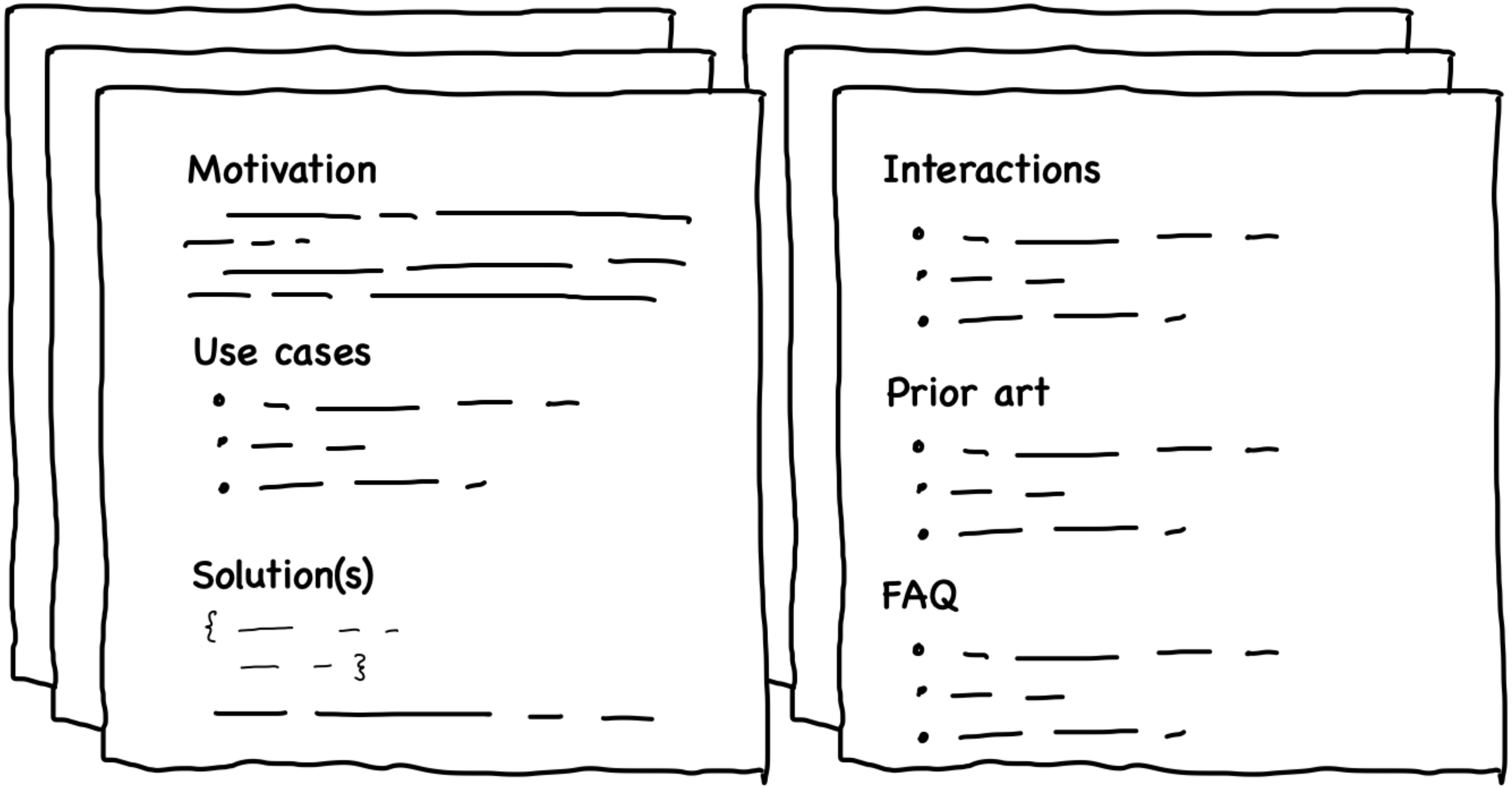

MVP
START HERE
compilation target
fast execution
compact
linear memory
SMALL MODULES INTEROP
JS
WA
fast calls JS <->wasm
easy & fast data exchange
ES module integration
toolchain integration
backwards compatibility
MAKING USE OF MODERN HARDWARE
threading
SIMD
wasm64
HEAVY-WEIGHT APPLICATIONS
LOADTIME IMPROVEMENTS
streaming compilation
tiered compilation
implicit HTTP caching
other improvements
NODE NATIVE MODULES
portable interface
HIGH LEVEL LANGUAGE FEATURES
GC integration
exception handling
debugging
tail calls
Reason
Scala.js
Elm
JS
ember
COMPILE-TO-JS LANGUAGES
JS FRAMEWORKS
CDNs, EDGE & SERVERLESS
INTERNET OF THINGS
runtime
BLOCKCHAIN
PORTABLE CLI TOOLS

By: Lin Clark


HIGH LEVEL LANGUAGE FEATURES
GC integration
exception handling
debugging
tail calls
Reason
Scala.js
Elm
JS
ember
COMPILE-TO-JS LANGUAGES
JS FRAMEWORKS

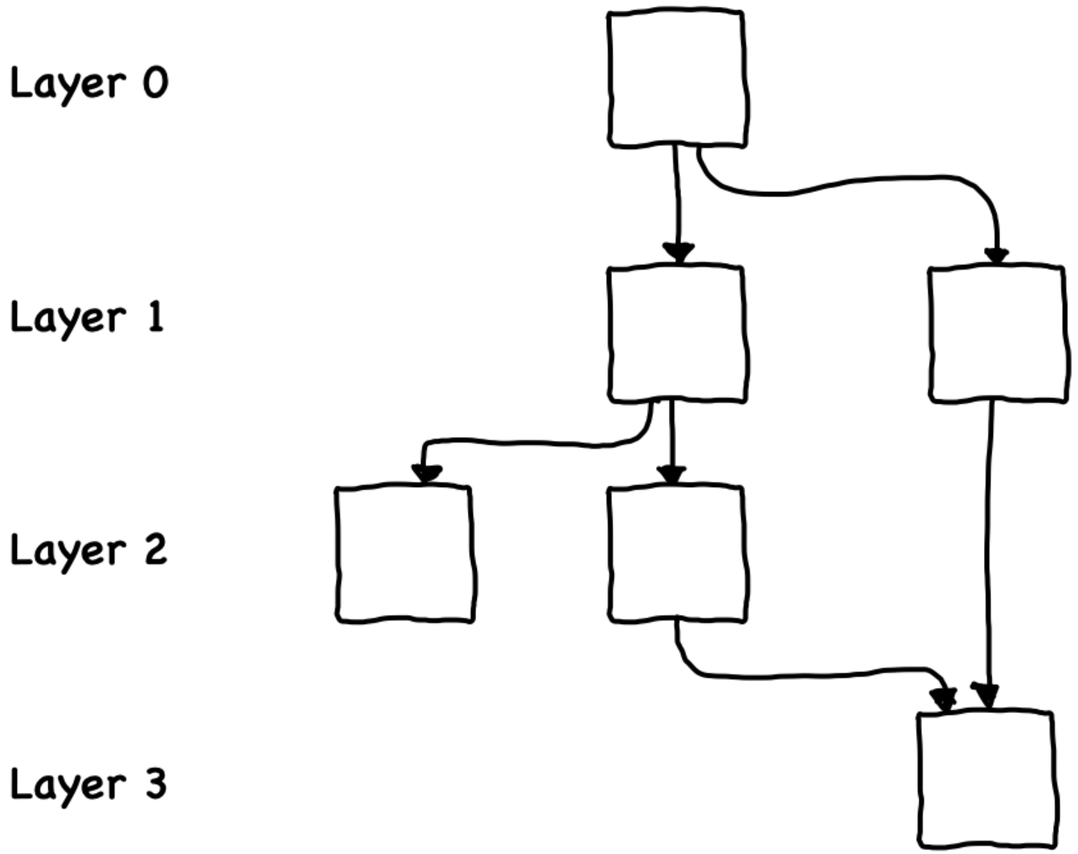
By: Lin Clark

Layered Proposal



Organizing Principle:
Structure is shaped by Dependencies

Layered Proposal



Defining a Layer

- A layer is a **collection** of work that
  - Has the same dependencies
  - Or must be worked on in parallel
- A layer is **complete when all of it's proposals advance**

Defining a Layer

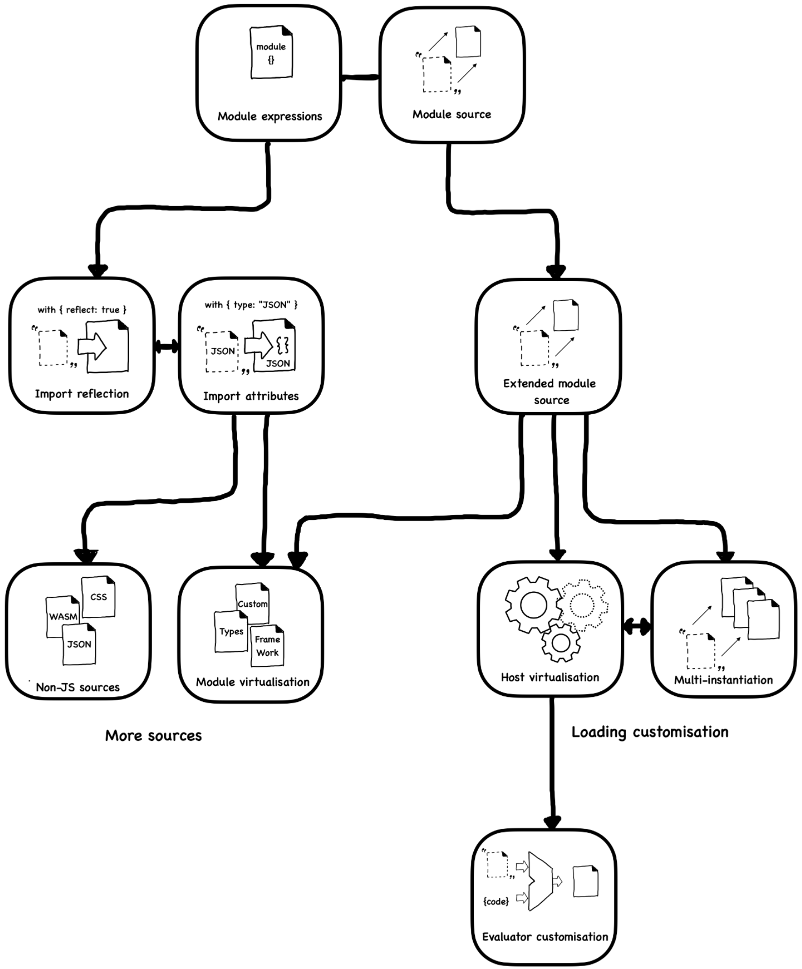
- Work can continue on proposals blocked by a layer
- Work blocked by a layer is easily identifiable
- Importance of a layer or layer component can be identified by its dependencies

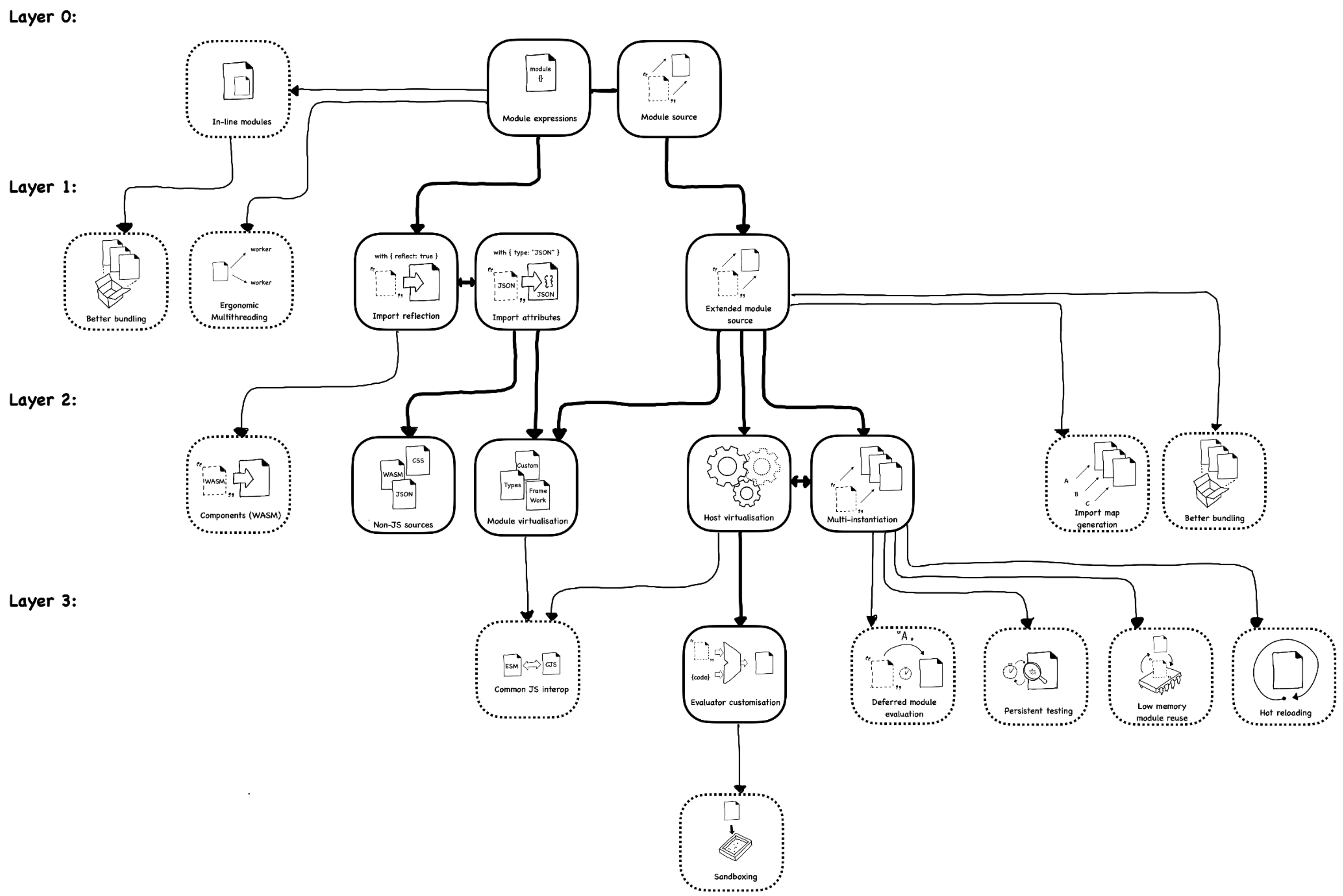
Layer 0:
Exposing the module object

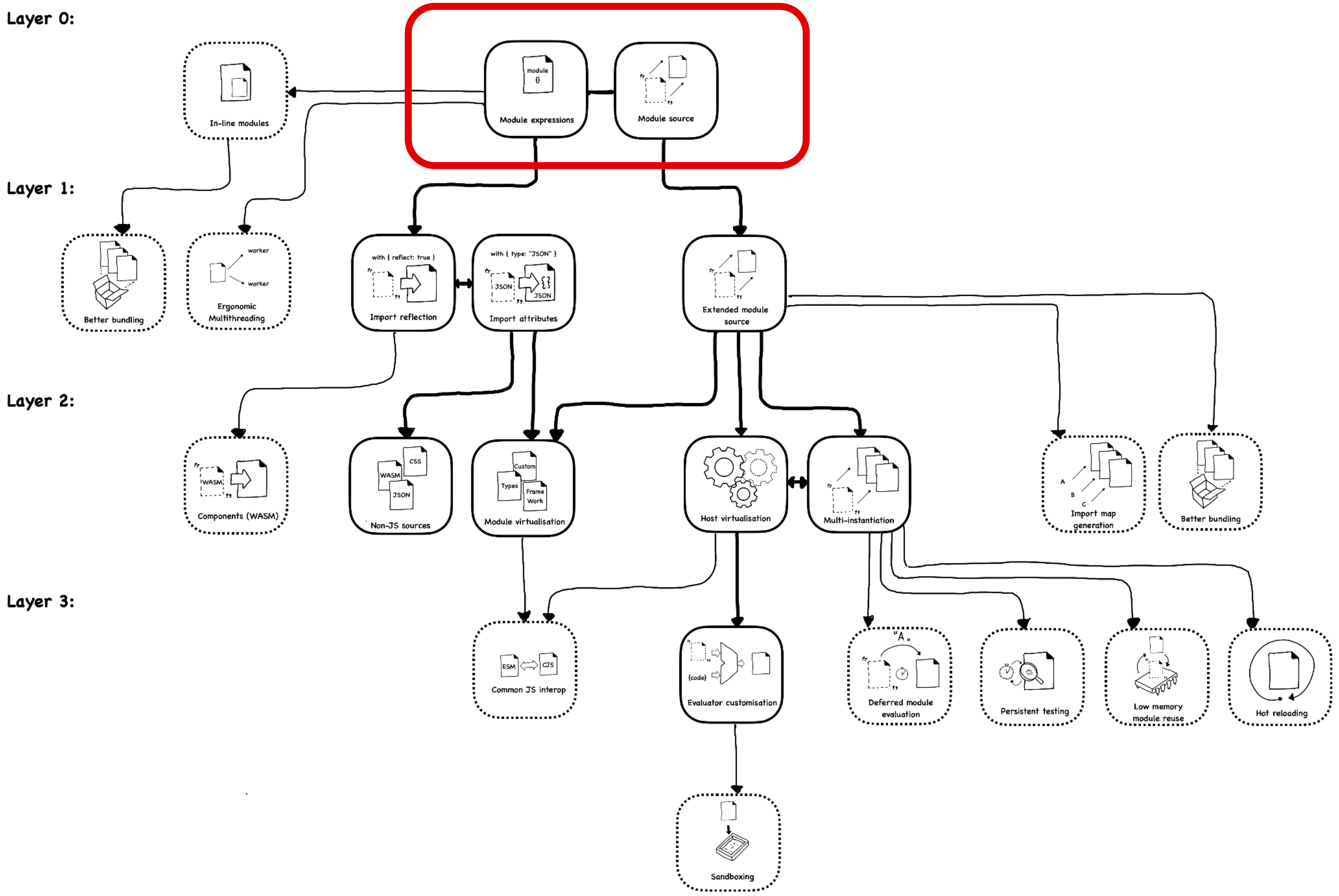
Layer 1:
Import/Export modification

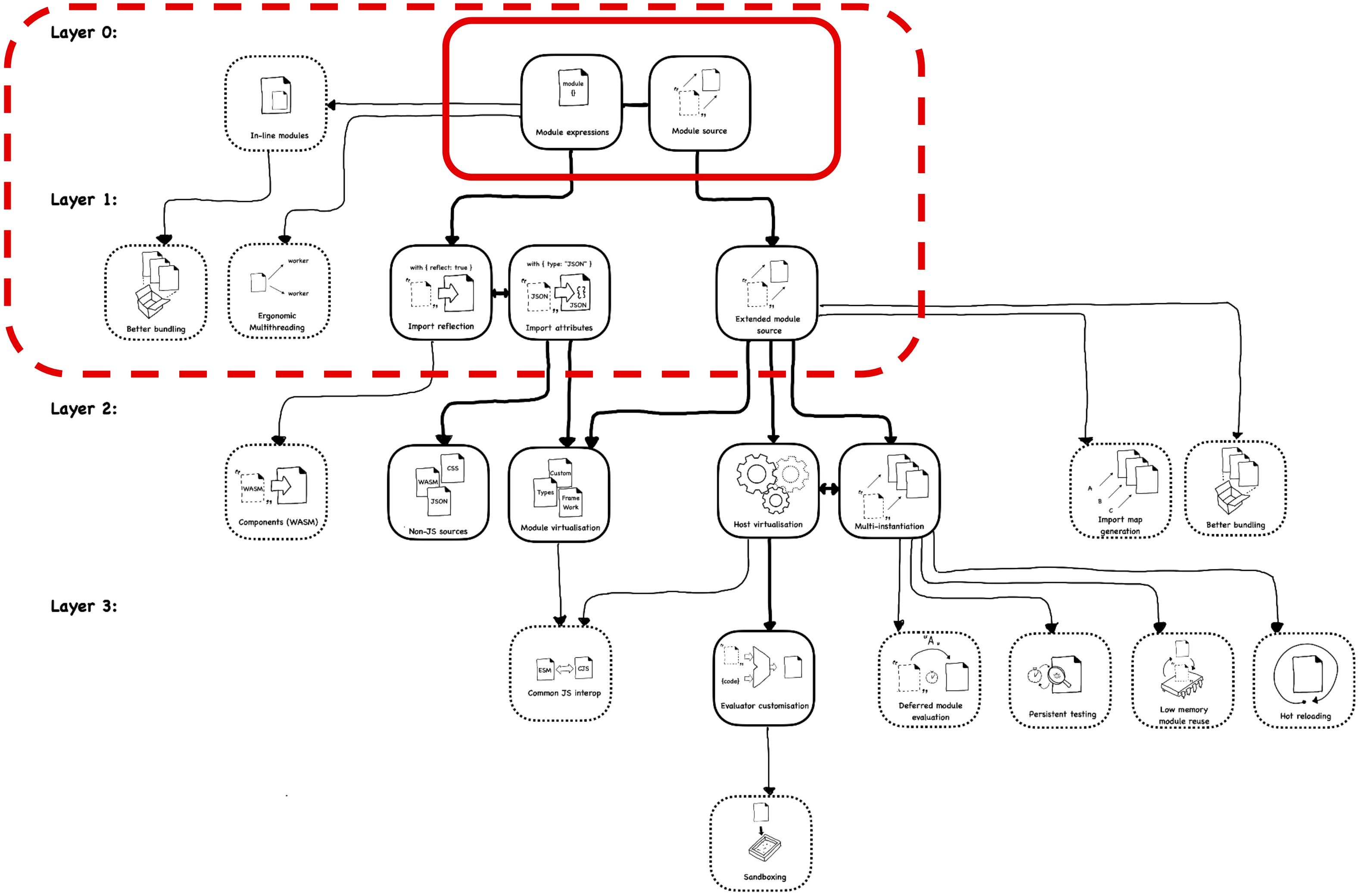
Layer 2:
Virtualisation

Layer 3:
TBD

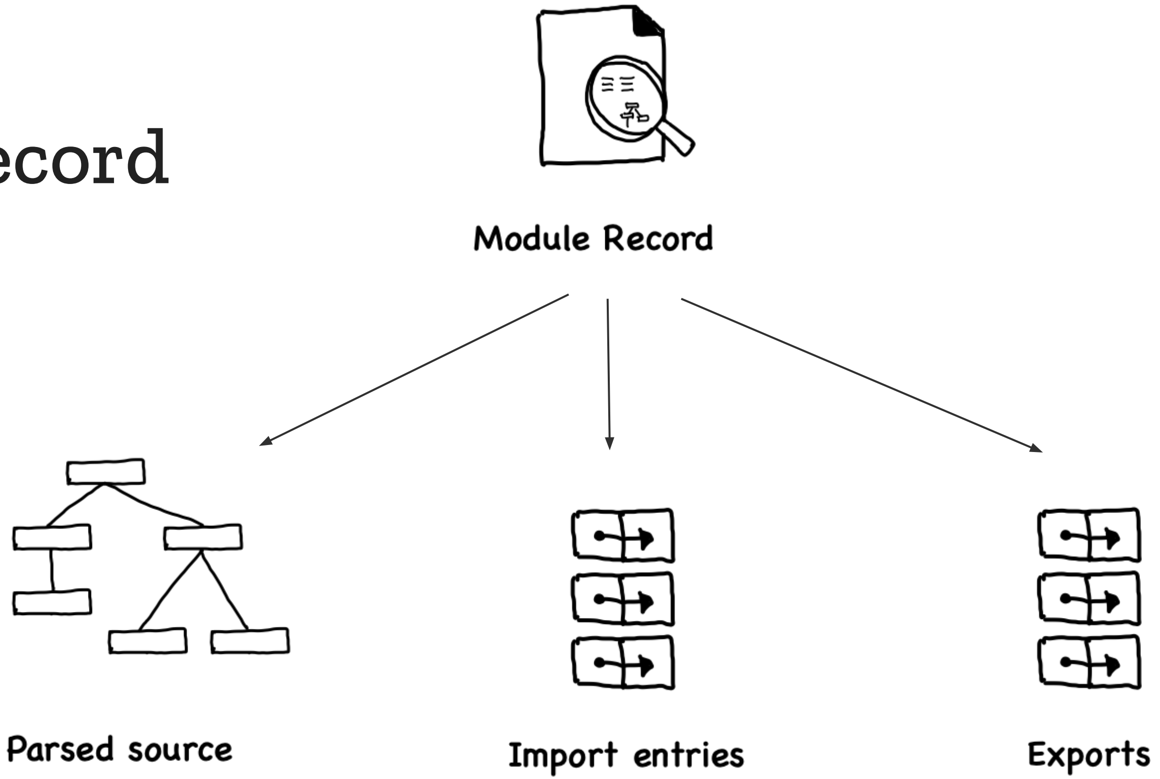



Layer 0:
In-line modules
module {}
Module expressions
Module source
Layer 1:
Better bundling
worker
worker
Ergonomic Multithreading
with { reflect: true }
Import reflection
with { type: "JSON" }
JSON
{}
JSON
Import attributes
Extended module source
Layer 2:
WASM
Components (WASM)
CSS
WASM
JSON
Non-JS sources
Custom
Types
Frame Work
Module virtualisation
Host virtualisation
Multi-instantiation
A
B
C
Import map generation
Better bundling
Layer 3:
ESM
CJS
Common JS interop
{code}
Evaluator customisation
"A"
Deferred module evaluation
Persistent testing
Low memory module reuse
Hot reloading
Sandboxing


Layer 0:
In-line modules
module {}
Module expressions
Module source
Layer 1:
Better bundling
worker
worker
Ergonomic Multithreading
with { reflect: true }
Import reflection
with { type: "JSON" }
JSON
JSON
Import attributes
Extended module source
Layer 2:
WASM
Components (WASM)
CSS
WASM
JSON
Non-JS sources
Custom
Types
Frame Work
Module virtualisation
Host virtualisation
Multi-instantiation
A
B
C
Import map generation
Better bundling
Layer 3:
ESM
CJS
Common JS interop
{code}
Evaluator customisation
A
Deferred module evaluation
Persistent testing
Low memory module reuse
Hot reloading
Sandboxing

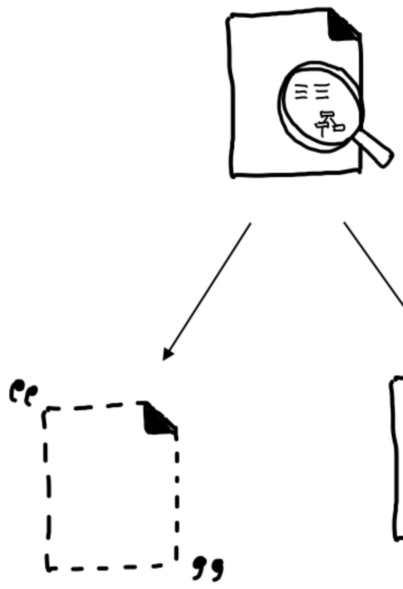

Layer 0:
Layer 1:
Layer 2:
Layer 3:
In-line modules
module {}
Module expressions
Module source
Better bundling
worker
worker
Ergonomic Multithreading
with { reflect: true }
Import reflection
with { type: "JSON" }
JSON
Import attributes
Extended module source
WASM
Components (WASM)
CSS
JSON
Non-JS sources
Custom
Types
Frame Work
Module virtualisation
Host virtualisation
Multi-instantiation
A
B
C
Import map generation
Better bundling
ESM
CJS
Common JS interop
{code}
Evaluator customisation
"A"
Deferred module evaluation
Persistent testing
Low memory module reuse
Hot reloading
Sandboxing

Our focus:
The Module Record



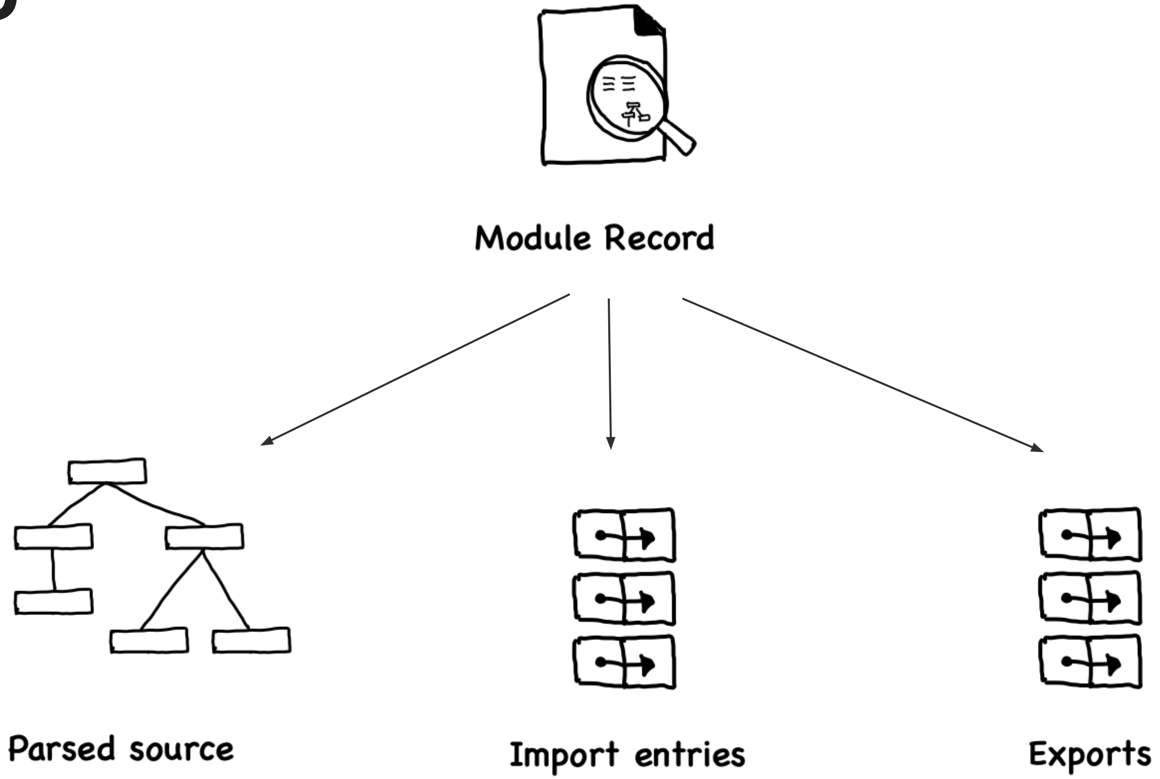
Layer 0 Walkthrough

Layer 0

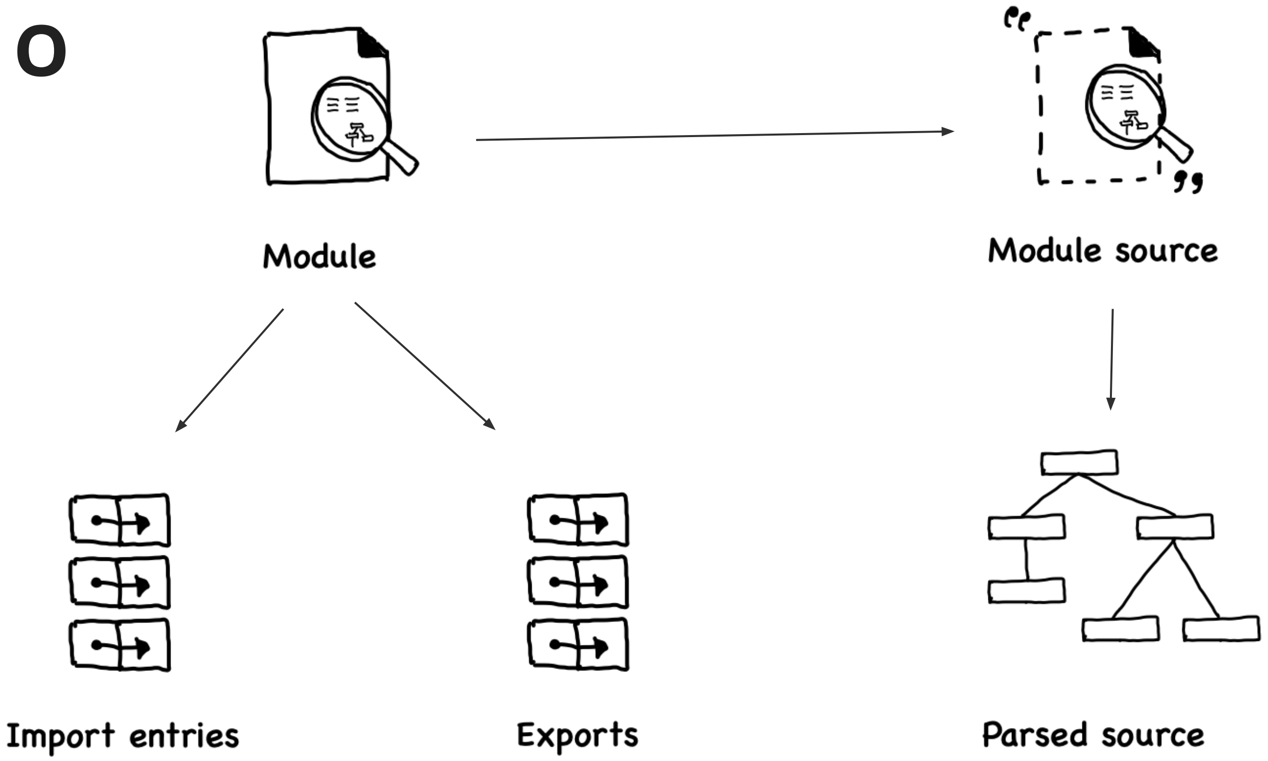


Refactoring module record

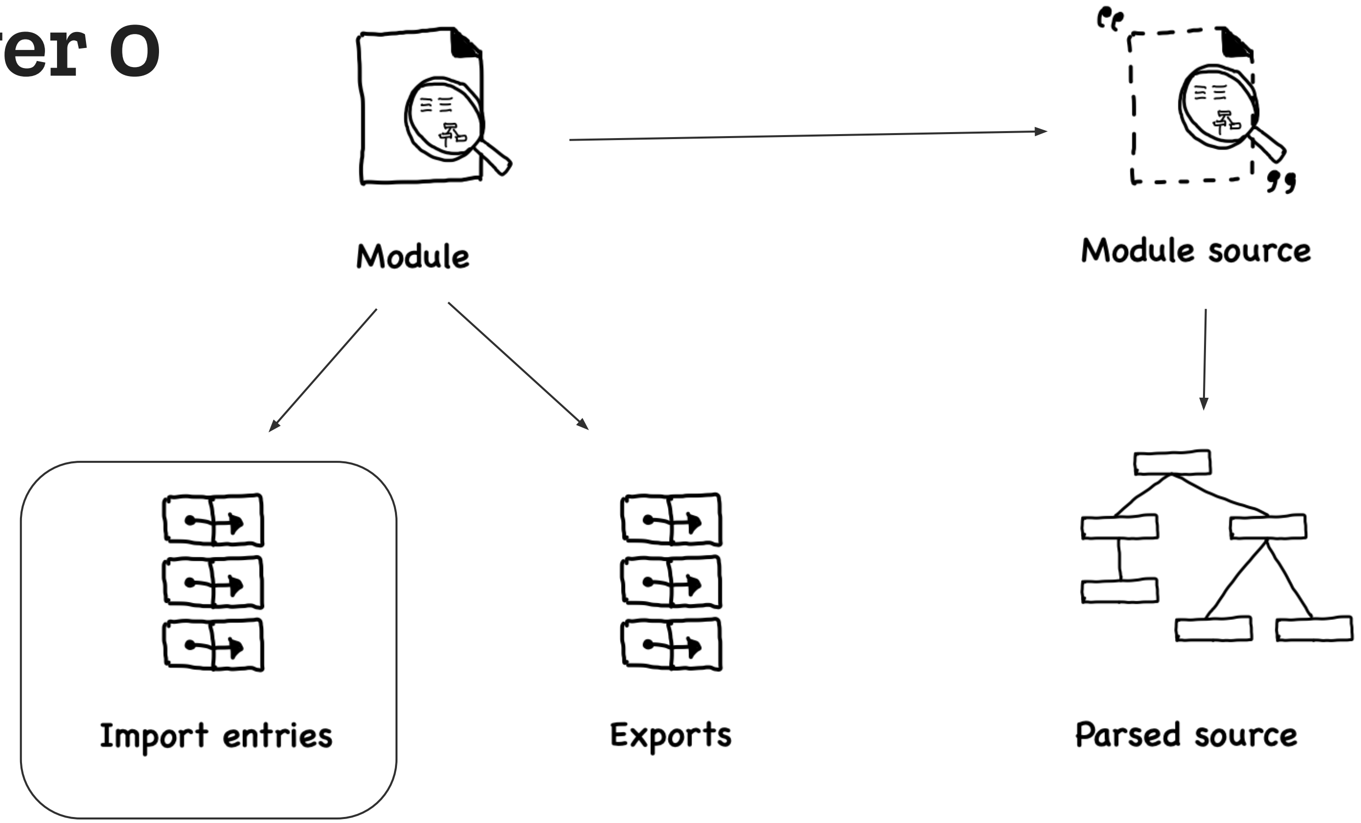
Layer 0



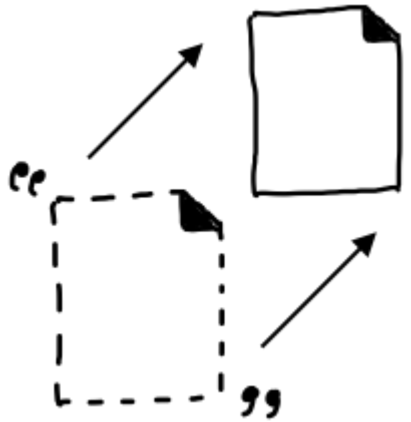
Layer 0


Module
Module source
Import entries
Exports
Parsed source

Layer 0

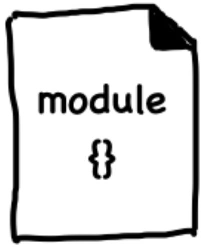

Module
Module source
Import entries
Exports
Parsed source

Layer 0: Module Source



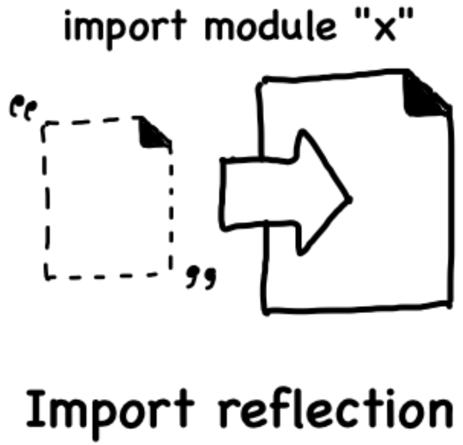
- A module source contains statically analyzable information

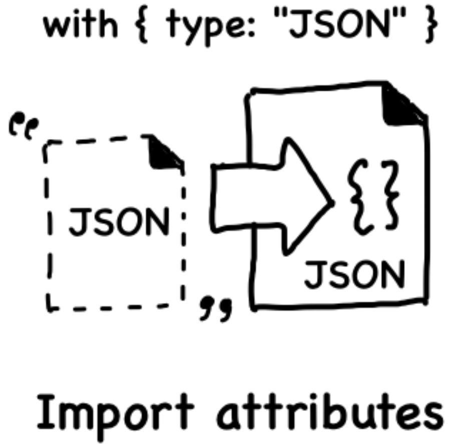
Layer 0:
Module expressions / declarations



- Module expressions expose a bound module object that can be executed
- Two proposals

Unlocked Layer 1 Capabilities

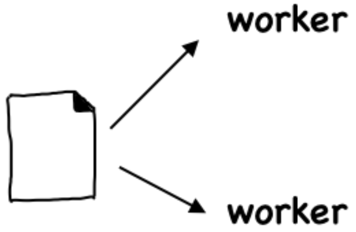




Unlocked Layer 1 Capabilities

```
import <load modifier> x from "y" with { type: "..." }
```

Unlocked Capabilities

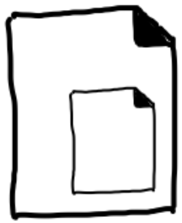


Ergonomic Multithreading

Unlocked Capabilities

```
let workerBlock = module {
  onmessage = async function({data}) {
    let mod = await import(data);
    postMessage(mod.default());
  }
};

let worker = new Worker({type: "module"}).addModule(workerBlock);
worker.onmessage = ({data}) => alert(data);
worker.postMessage(module { export default function() { return "hello!" } });
```

Unlocked Capabilities

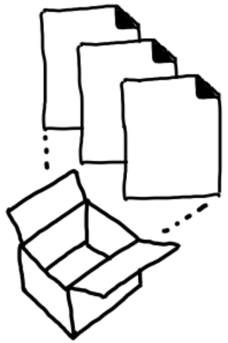


In-line modules

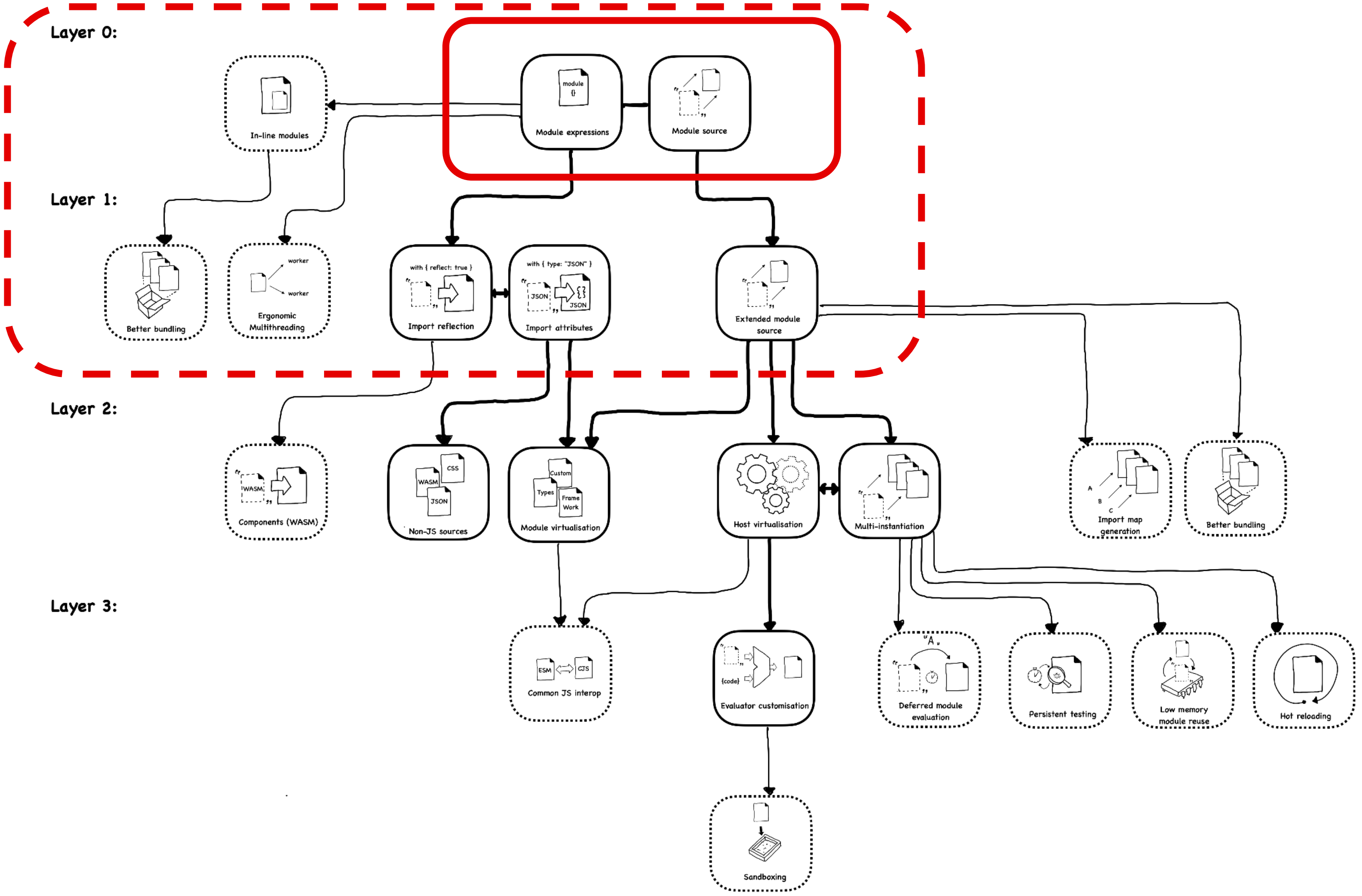
Unlocked Capabilities

```
module countModule {
  // ...
}

module uppercaseModule {
  // ...
}

import { count } from countModule;
import { uppercase } from uppercaseModule;
```

Unlocked Capabilities



Better bundling


Layer 0:
Layer 1:
Layer 2:
Layer 3:
module {}
Module expressions
Module source
In-line modules
Better bundling
worker
worker
Ergonomic Multithreading
with { reflect: true }
Import reflection
with { type: "JSON" }
JSON
JSON
Import attributes
Extended module source
WASM
Components (WASM)
CSS
WASM
JSON
Non-JS sources
Custom
Types
Frame Work
Module virtualisation
Host virtualisation
Multi-instantiation
A
B
C
Import map generation
Better bundling
ESM
CJS
Common JS interop
{code}
Evaluator customisation
"A"
Deferred module evaluation
Persistent testing
Low memory module reuse
Hot reloading
Sandboxing

Links to the proposals

Layer 0:

- Module Source
- Module Expressions

Reading More:

- Module Declarations
- Import Reflection
- Import Assertions
- Compartments

Questions

@codehag@mastodon.social

moz://a