# ES Modules (ESM) Lifecycle

# ~Lifecycle

**Note:** the following slides contain “host dependant behavior” this is used in order to enforce some rules of ESM such as those in ModuleDeclarationInstanciation and HostResolveImportedModule

# ~Lifecycle (**Resolve**, **Fetch**, **Parse**, **Link**, **Evaluate**)

1. **Resolve** (as *absolute URL*) => **Fetch** => **Parse**
    a. Make Module Record
    b. Place in Global Cache using Absolute URL*
    c. Errors remove records from Global Cache*
2. Traversal of import declarations recursively
    a. Ensure step 2 has been performed on the dependency
    b. Place dependency in Local Cache using Import Specifier*
    c. **Link** dependency to module
    d. Errors prevent *any* evaluation
3. **Evaluate** in post order traversal
    *a.* Errors prevent *further* evaluation

*Host dependent behavior

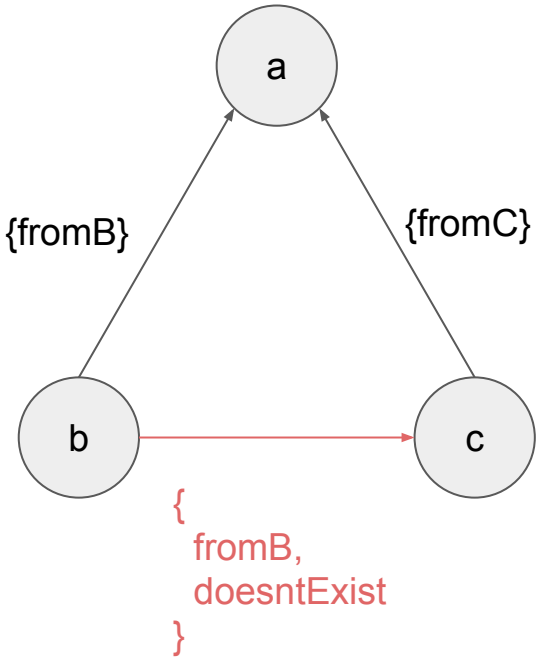
# ~Lifecycle Errors

```
// a (entry)
import {fromB} from 'b';
import {fromC} from 'c';
```

```
// b
export let fromB = 'b';
```

```
// c
import {fromB} from 'b';

// FIXME
import {doesntExist}
from 'b';

export let fromC = 'c';

throw Error();
```

# ~Lifecycle Link Error




NO EVAL

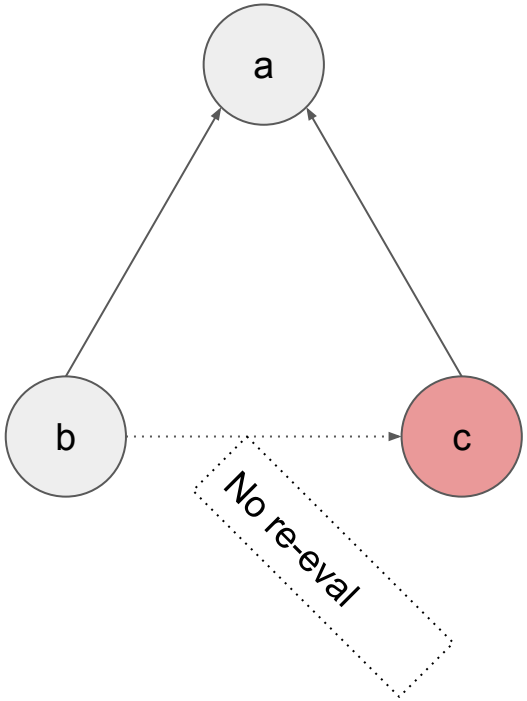
- Resolve / Fetch / Parse a
  - Place a into global cache*
  - Errors here remove from global cache*
- *In Parallel:* Resolve / Fetch / Parse a => [b,c]
  - Place [b,c] into global cache*
  - Place [b,c] into a's local cache*
  - Place b into c's local cache*
  - Errors here remove from global cache*

- Link a using b,c
- Link b (noop)
- Link c using b
  - Error doesntExist does not exist (fix it)
  - c stays in cache in errored state*

*Host dependent behavior

# ~Lifecycle Eval Error

PARTIAL EVAL




- Resolve / Fetch / Parse / Link full graph
  - Place [a,b,c] into global cache*
- Eval b
- Eval c
  - Error
  - [a,c] stay in global cache in errored state*

*Host dependent behavior

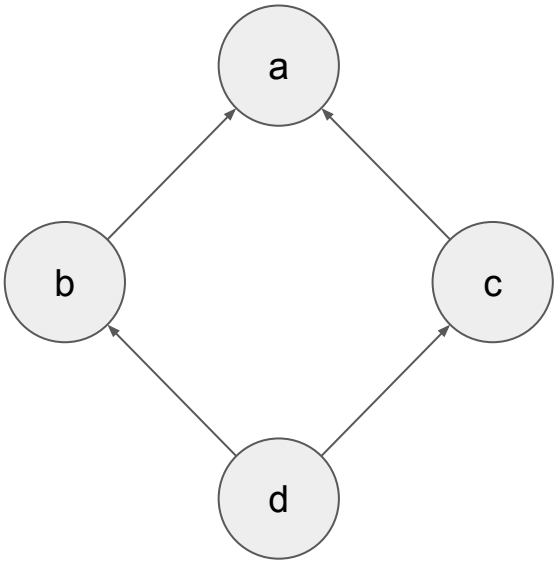
# ~Lifecycle Parallel Loading

```
// a (entry)
import 'b';
import 'c';
```

```
// b
import 'd';
```

```
// c
import 'd';
```

# ~Lifecycle Parallel Loading




- Resolve / Fetch / Parse full graph
  - Place [a,b,c,d] into global cache*

- Link a
- Link b
- Link d
- Link c

Note differing module orders in Link & Eval

- Eval d
  - Prior to *all* dependents
- Eval b
- Eval c
- Eval a

*Host dependent behavior

# ESM Spec Compat with Node CommonJS (NCJS)

# Conceptual vs Material

The following slides use the terms “conceptual” and “material” for a specific meaning.

- Conceptual - intent or design goals that are fundamentally at odds
- Material - specification or implementation mandates that are at odds

# Mode Detection

- Conceptual
  - Spec expects out of band declaration
  - Some new spec additions/reserved words are only toward Module
- Material
  - A source text can be ambiguous in some cases but evaluation can produce different values for some expressions

# Cache Data Structures

Cache data structures are used to enforce Host Dependant behavior required by the ESM specification.

- Values mimic the values of Module Map
  - {

    status: 'fetching' | 'uninstanciated' | 'instanciated' | 'error',

    module: Module Record

    }
- Global:
  - Key on Absolute URL (ephemeral URI like data: are not cached.)
- Local (associated with a Module Record):
  - Key on Import Specifier

# Cache

`import('f'); import('f');`

- Idempotent per specifier per ESM.
  - Conceptual
    - Each ESM has a local cache
    - ESM import declaration links *prior* to any eval => idempotent prior to eval
    - No retry when re-importing a specifier
      - Errors not always globally permanent (e.g. Fetch a single module script)
    - ESM is a graph not a tree
      - Things like `module.parent` no longer logical (need hooks)

# Linking

- Permanent per ESM.
  - Conceptual
    - Once linked ESM cannot be relinked
  - Material
    - ModuleDeclarationInstanciation is a no-op if the ESM has been called already, even if no evaluation has occurred
    - Some modules re-link their parents like meow

# Linking

- Both
  - Allow re-linking prior to evaluation
  - Remove HoistableDeclaration availability until evaluation

# Named Imports

```
import {readFile} from 'fs';
```

- ESM cannot named import from NCJS dependencies
  - Conceptual
    - Conflicting timing
      - ESM link *prior* to evaluation
      - NCJS declare export *after* evaluation
      - Load order needs to be maintained
    - ESM import/export lists are static, NCJS is dynamic (changing value/property list)
      - Exporting async, deleting, etc. cannot be respected (cannot form proper Proxy membrane since value is assigned from user code)
  - Material
    - Hoisting live properties means same guards as property access. This is a deopt nightmare in same vein as `with`.

# Named Imports Example

```
import {Component} from 'React';
```

# Timing

`require('esm') || import('ncjs');`

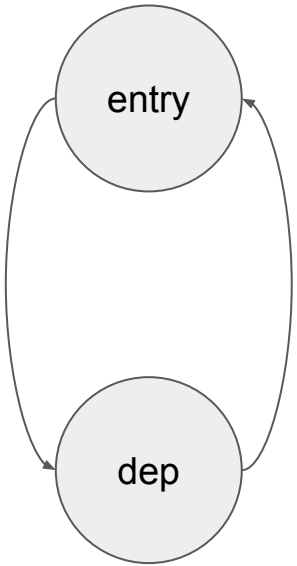
- *always* async
  - Conceptual
    - Intent for async IO in browsers
    - Linking phase separated from evaluation
    - Top level await requires async
  - Material
    - In order to do sync would need to recurse in HostResolveImportedModule
      - Breaks linking assumptions
        - Partial linkage / Errors prior to eval
      - Some circular deps don’t work (NCJS -> ESM -> NCJS)
    - Match browser, unify

# Timing Example - Circular

```
// entry (CJS)

require('dep');

module.exports = null;
```

```
// dep (ESM)

import foo from 'entry';
```

# ~Sync Timing Error - Link



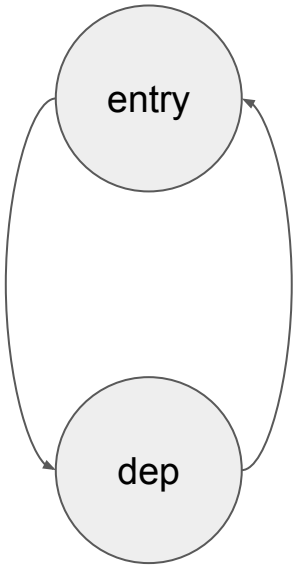

- Evaluate entry
- Fetch / Load dep
- Link dep
  - entry has not finished evaluation / has no shape

# Timing Example - Hoistable

```
// entry (ESM)

import 'dep';

export function foo() {}
```

```
// dep (ESM)

import foo from 'entry';

foo();
```

# ~Sync Timing Error - Hoistable




- Fetch / Load entry
- Fetch / Load dep
- Link dep
  - entry has not finished initialization of foo
- Evaluate dep

# Hoistable fix

- Either
  - Re-order ModuleDeclarationInstanciation
    - Initialize LexicallyScopedDeclarations prior to linking dependency subgraphs
  - Remove HoistableDeclaration availability until evaluation

# ESM Doable Needs

# Context

```
import {url} from 'js:context'; //(bikeshed)
```

- Remove NCJS “magic” variables
- **Available**
  - `url` (initial go ahead from browsers to match)
- Early error trying to use non-existant context variables
- Allows *hosts* to selectively expose certain things
  - Loader
  - Important if you have nested ESM Loaders (jsdom)
- Should be accessible if you have a Module Record

# Hooks

- Need to be able to ensure hookup prior to any of desired dep-graph evaluation
- Need some specific hooks (should disableable)
  - Local/Global Cache manipulation
    - Allow populating Local/Global Cache (Module Record, or other loading job)
      - Error to populate Local Cache *after* if specifier already linked
    - Allow removing *errors* from Global Cache
  - Resolution notification (request and result)
    - No manipulation (use cache manip upon request notification)
  - Source code transform
    - No module system swapping (use cache manip upon resolution result notification)

# Userland Loaders

- Should enforce the constraints of ESM, by API design, and early error if violation is possible
  - WHATWG Loader Spec
    - Looser constraints than Node’s Loader