

# **Frozen Realms**

## **Proposed Standard Support for Safer JavaScript Plugins**

**Mark S. Miller (Google)**

`erights@google.com`

**Chip Morningstar (PayPal)**

`cmorningstar@paypal.com`

**Caridy Patiño (Salesforce)**

`cpatino@salesforce.com`

Verified Trustworthy Software Systems 2016

# Frameworks with fallible plugins

The screenshot displays the Google Earth Engine (GEE) Playground interface. The top navigation bar includes the Google Earth Engine logo, a search bar, and a user profile dropdown for 'nicholas.clinton'. The main interface is divided into three primary sections: Scripts, Playground.js, and the Inspector/Console/Logs panel.

**Scripts Panel:** A sidebar on the left lists various GEE methods under the heading 'Filter methods...'. The methods listed include ee.Algorithms, ee.Array, ee.Blob, ee.Date, ee.DateRange, ee.Dictionary, ee.ErrorMargin, ee.Feature, ee.FeatureCollection, ee.Filter, and ee.Geometry.

**Playground.js Panel:** The central area shows a JavaScript script being executed. The script is as follows:

```
1 Imports (5 entries)
2 // load the most recent MODIS composite
3 var modis = ee.Image(imageCollection
4   .sort('system:time_start', false)
5   .first());
6
7 // print metadata to the console
8 print(modis);
9
10 var sld = "\
11   <RasterSymbolizer>\
12   <ContrastEnhancement><Normalize></ContrastEnhancement>\
13   <ChannelSelection>\
14     <RedChannel>\
15       <SourceChannelName>sur_refl_b01</SourceChannelName>\
16     </RedChannel>\
17     <GreenChannel>\
18       <SourceChannelName>sur_refl_b04</SourceChannelName>\
```

**Inspector Panel:** The rightmost panel shows the execution results. It displays a 'Point (13.54, 23.56) at 20Km/px' and a 'MODIS composite: Image (3 bands)'. The 'Objects' section shows the 'MODIS composite' as an 'Image' with 3 bands and 5 properties. The 'DEM' is shown as an 'Image NOAA/NGDC/ETOP01 (2 bands)' with 2 properties.

**Map View:** The bottom section shows a satellite map of the world. A large, semi-transparent blue watermark with the text 'Fallible Plugins' is overlaid on the map. The map includes a 'Layers' panel on the right and a 'Map | Satellite | Deep' navigation bar at the bottom.

# Enablers of safer plugins

---

Memory safety

- + Encapsulation
  - + **Defensible** objects
  - + Effects **only** by using held references
  - + No **powerful** references by default
-

# Enablers of safer plugins

---

Memory safety

- + Encapsulation
  - + **Defensible** objects
  - + Effects **only** by using held references
  - + No **powerful** references by default
- 

Reference graph  $\equiv$  Access graph

Reachability limits effects

Abstraction boundary  $\equiv$  Enforcement mechanism

# Java?

---

- ✓ Memory safety
- ✓ Encapsulation
- ✓ **Defensible** objects

Effects **only** by using held references

No **powerful** references by default

# From Java to Joe-E

---

// Effects **only** by held references?

static Array mu = [];

import java.io.File;

// No **powerful** references by default?

...ClassLoader...

# From Java to Joe-E

---

// Effects **only** by held references

static Array mu = []; // prohibit

import java.io.File; // tame

// No **powerful** references by default

...ClassLoader... // restrict to subset

# Ancient JavaScript? (ES3, 1999-2009)

---

- ✓ Memory safety

  - Encapsulation

  - Defensible** objects

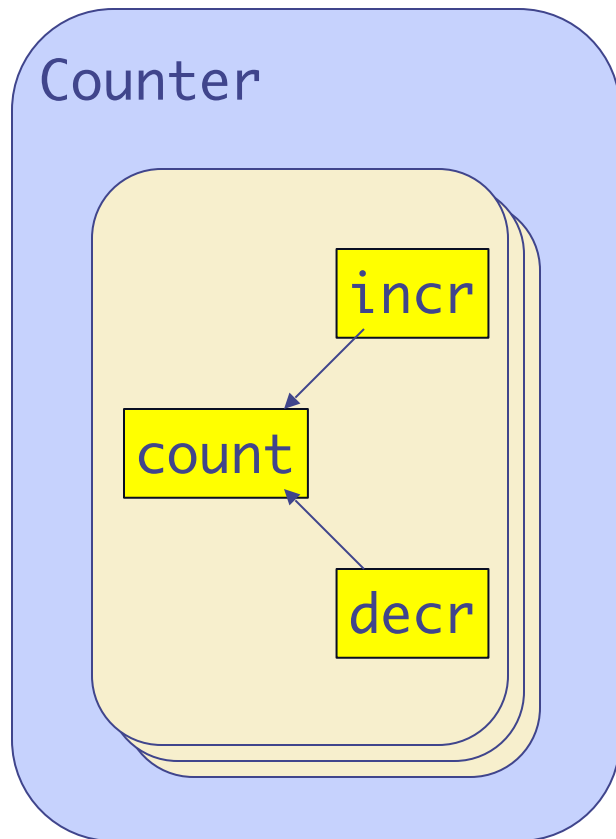
- ✓ Effects **only** by using held references

  - No **powerful** references by default



# Just Enough JavaScript

---



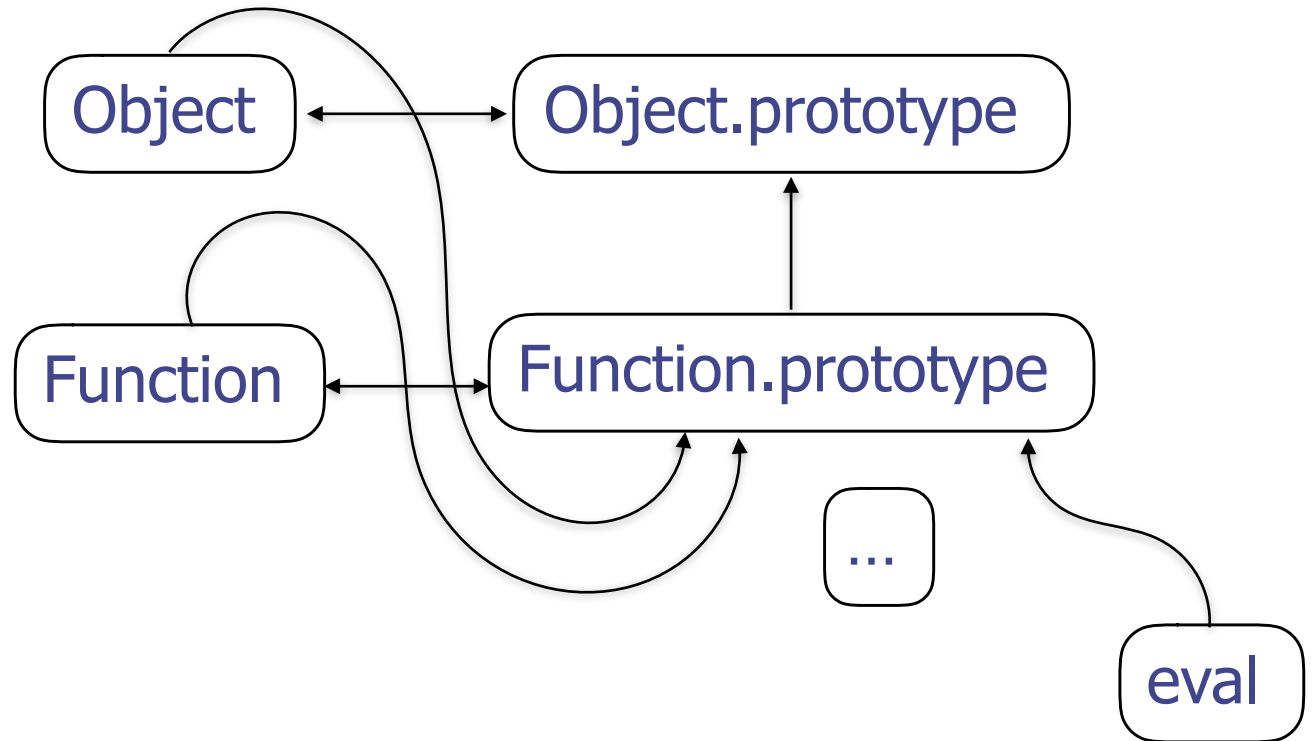
```
function Counter() {  
  let count = 0;  
  return {  
    incr: () => ++count,  
    decr: () => --count  
  };  
}
```

A record of closures hiding state  
is a fine representation of an  
object of methods hiding instance vars

# Primordials

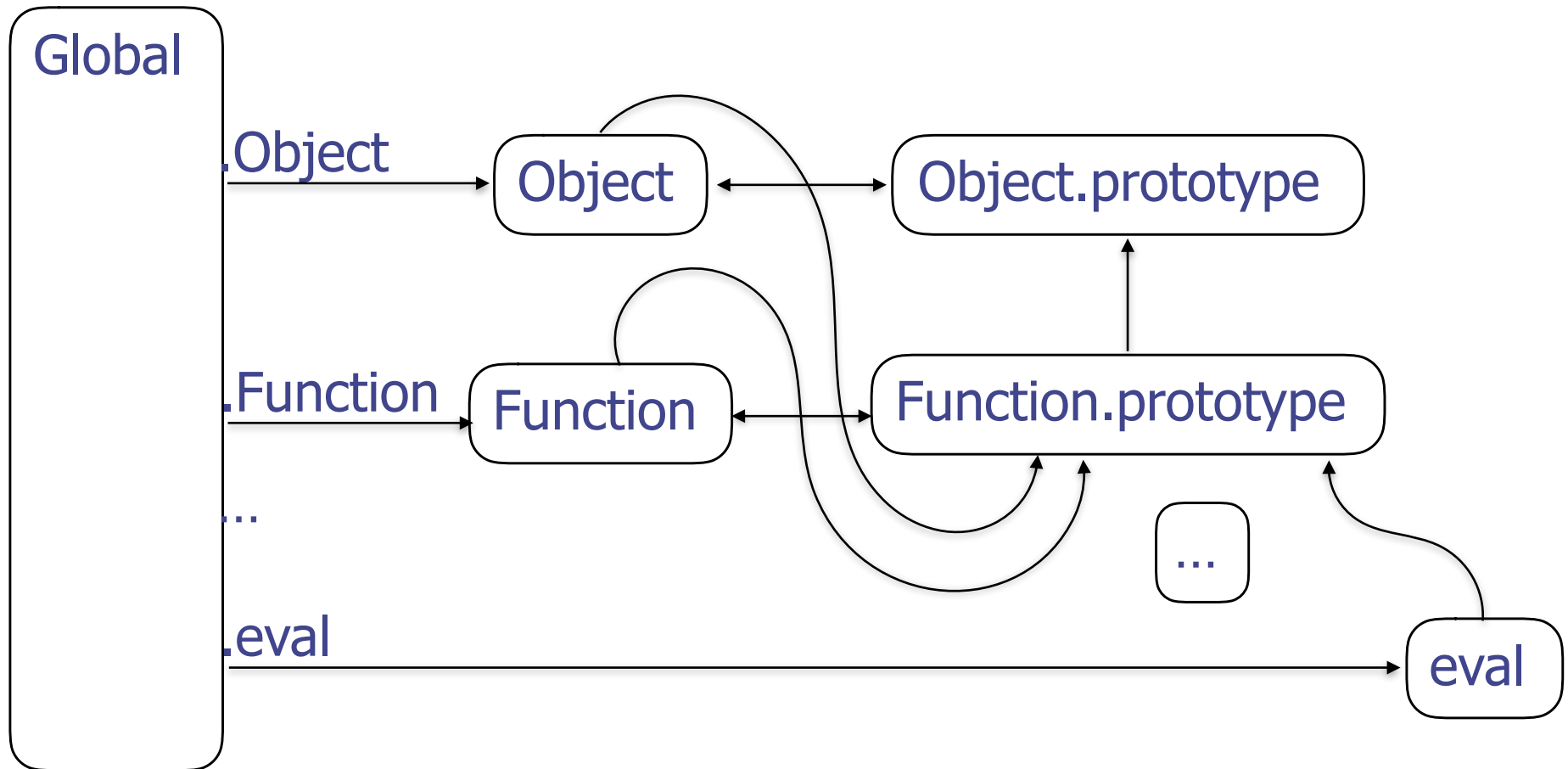
Objects that must exist before code runs

---



# Global object ("window" in browsers)

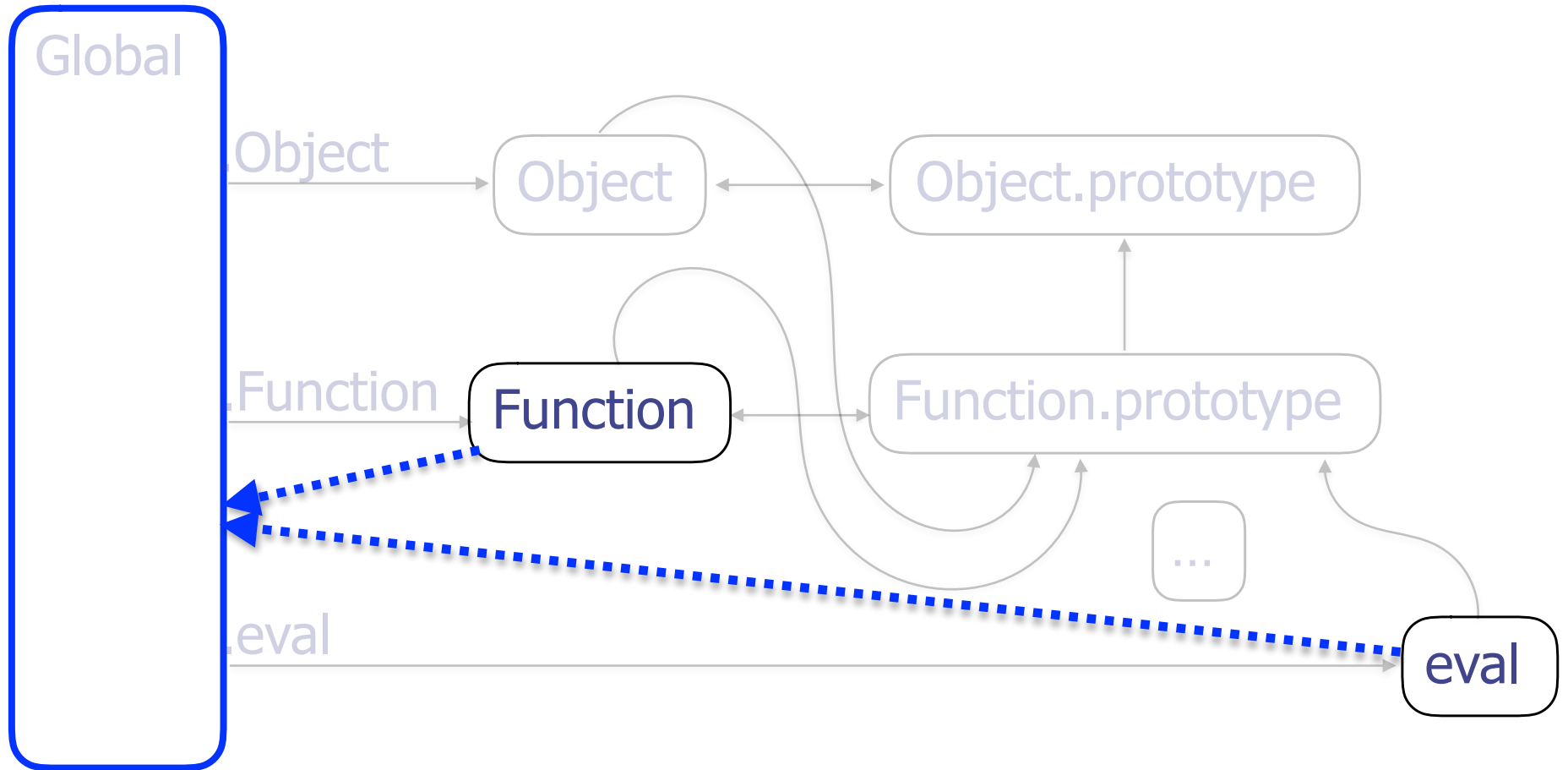
---



# Evaluators: eval, Function

## Evals code in scope of global's names

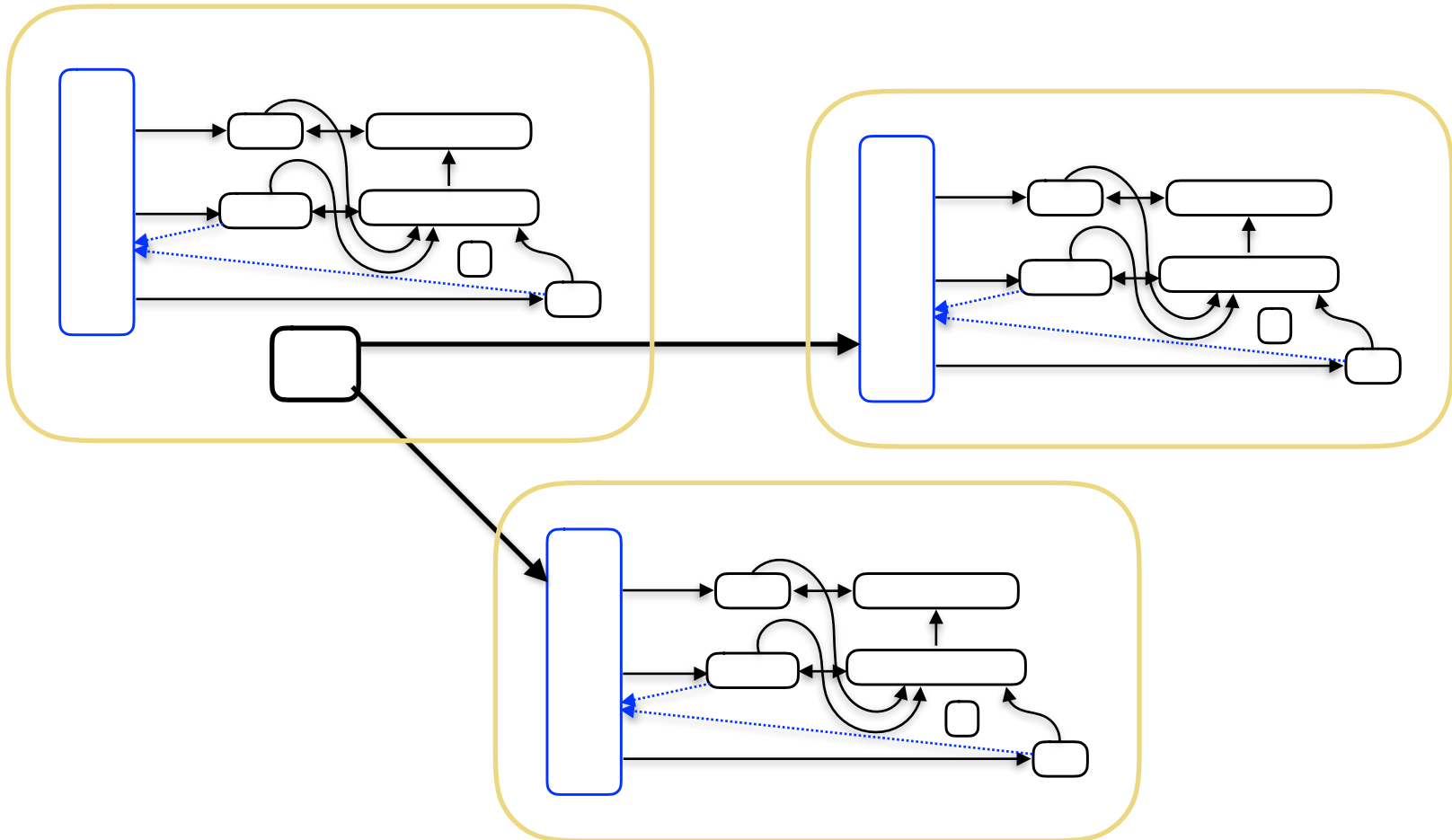
---



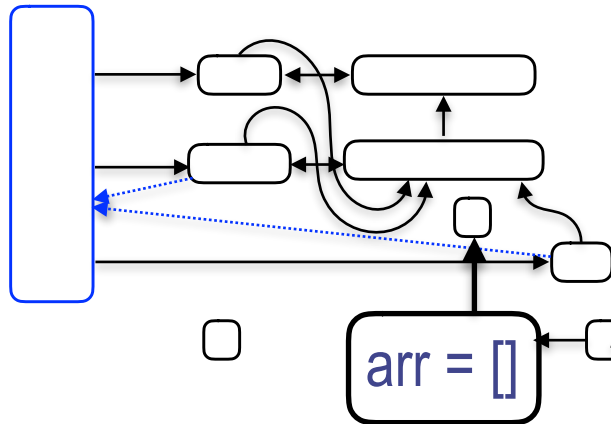
# Multiple Isolated Realms

## In browser: same-origin iframes

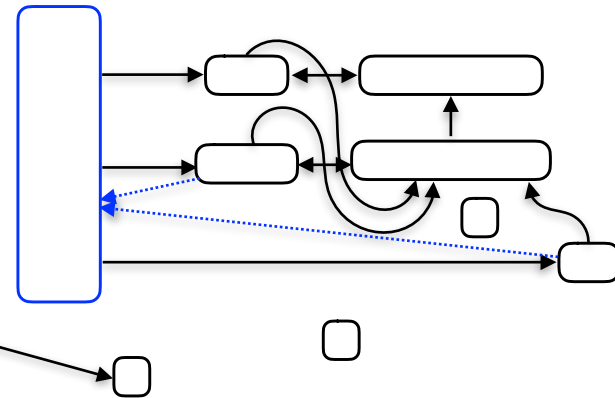
---



# Identity Discontinuities



```
Alice says: const arr = [];  
// pass arr to Bob
```



```
Bob says: arr instanceof Array
// false!
```

# “user mode” JS vs “system mode” host

---

JS by itself has no I/O.

Computes, invokes host-provided objects

All I/O via host-provided global variables

window, document, XMLHttpRequest, files, sockets

Hosts differ: browser, server (nodejs), devices

Computational libraries mostly in “user mode” JS

# ES5/strict (2009) — **Defensible** Objects

---

Full static scoping (almost lexical)

Impenetrable encapsulation

Safe reflection

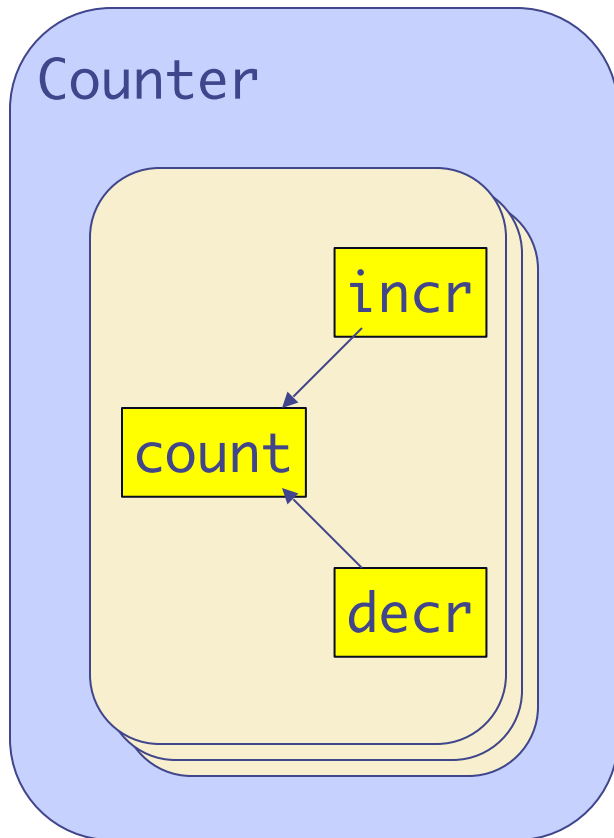
`Object.freeze(obj)`

tamper-proof API surface



# A Defensive Counter

---



```
function Counter() {  
  let count = 0;  
  return Object.freeze({  
    incr: Object.freeze(() => ++count),  
    decr: Object.freeze(() => --count)  
  });  
}
```

# ES2015 (ES6)

---

Full lexical scoping

let, const, function-in-block, modules

Classes (almost) as sugar

“Full Virtualizability”

Proxy, Reflect, WeakMap ==> Membranes

Virtualize host-provided “system mode” objects

# Modern JavaScript? (2009-present)

---

- ✓ Memory safety
- ✓ Encapsulation
- ✓ **Defensible** objects
- ✓ Effects **only** by using held references  
No **powerful** references by default

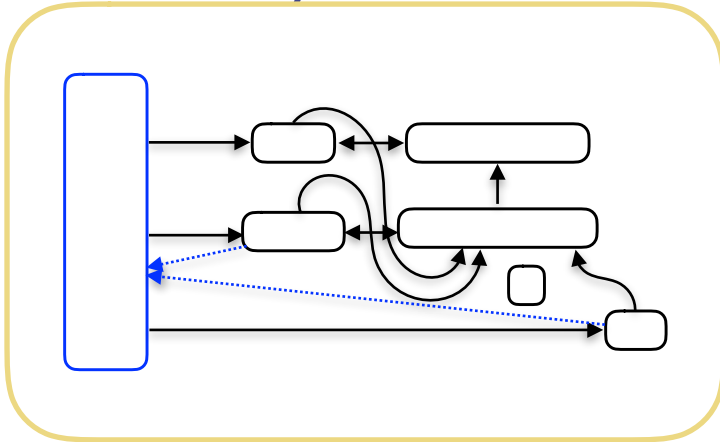
# Proposed Frozen Realm API

---

```
class Realm {  
    // From the old Realm API proposal  
    const global           // global object of this realm  
    eval(stringable)       // completion value  
  
    // New with this "Frozen Realm" proposal  
    static immutableRoot() // immutable root realm  
    spawn(endowments)      // lightweight child realm  
}
```

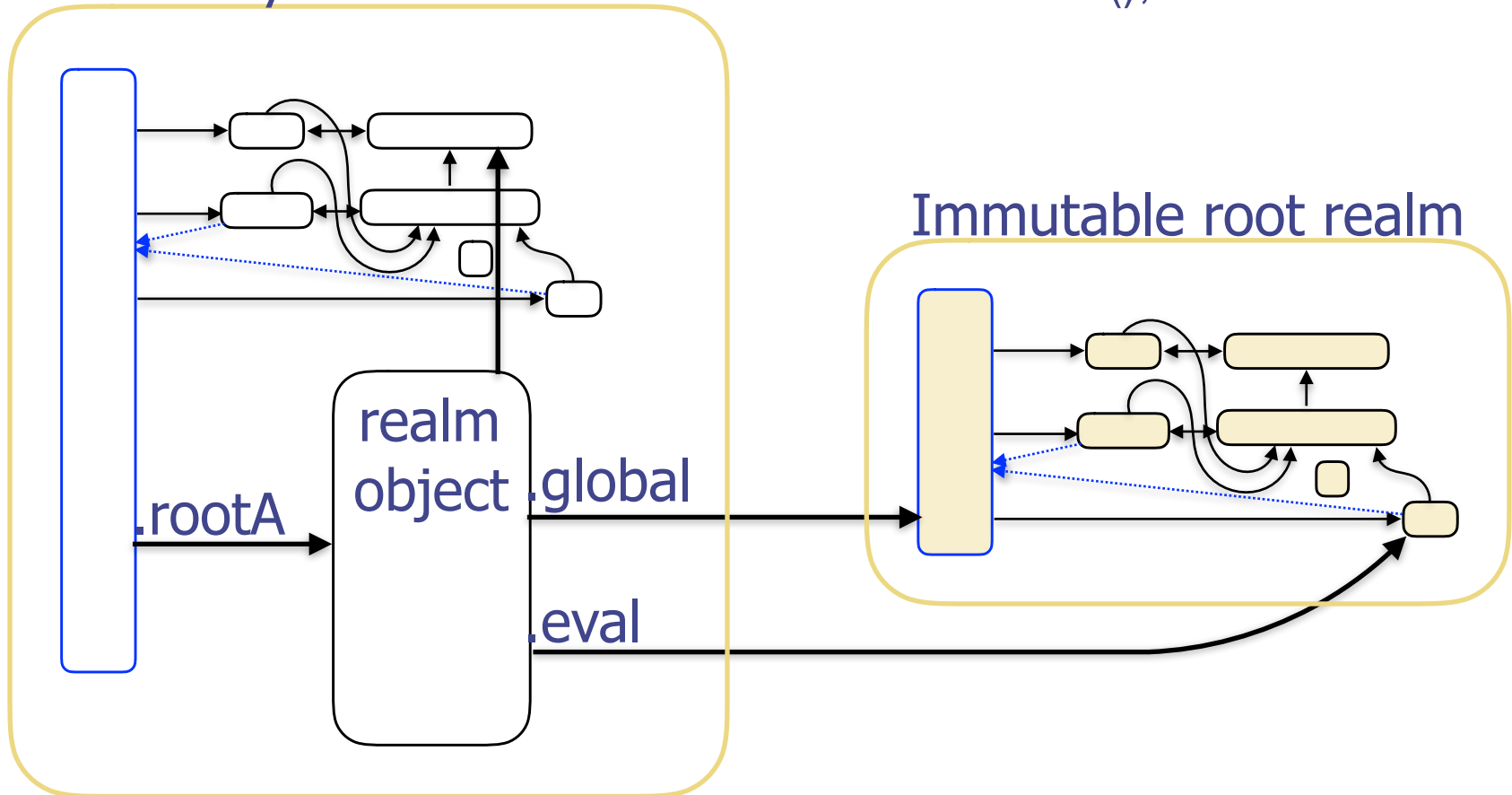
# Creating an immutable root realm

Alice says: `const rootA = Realm.immutableRoot();`



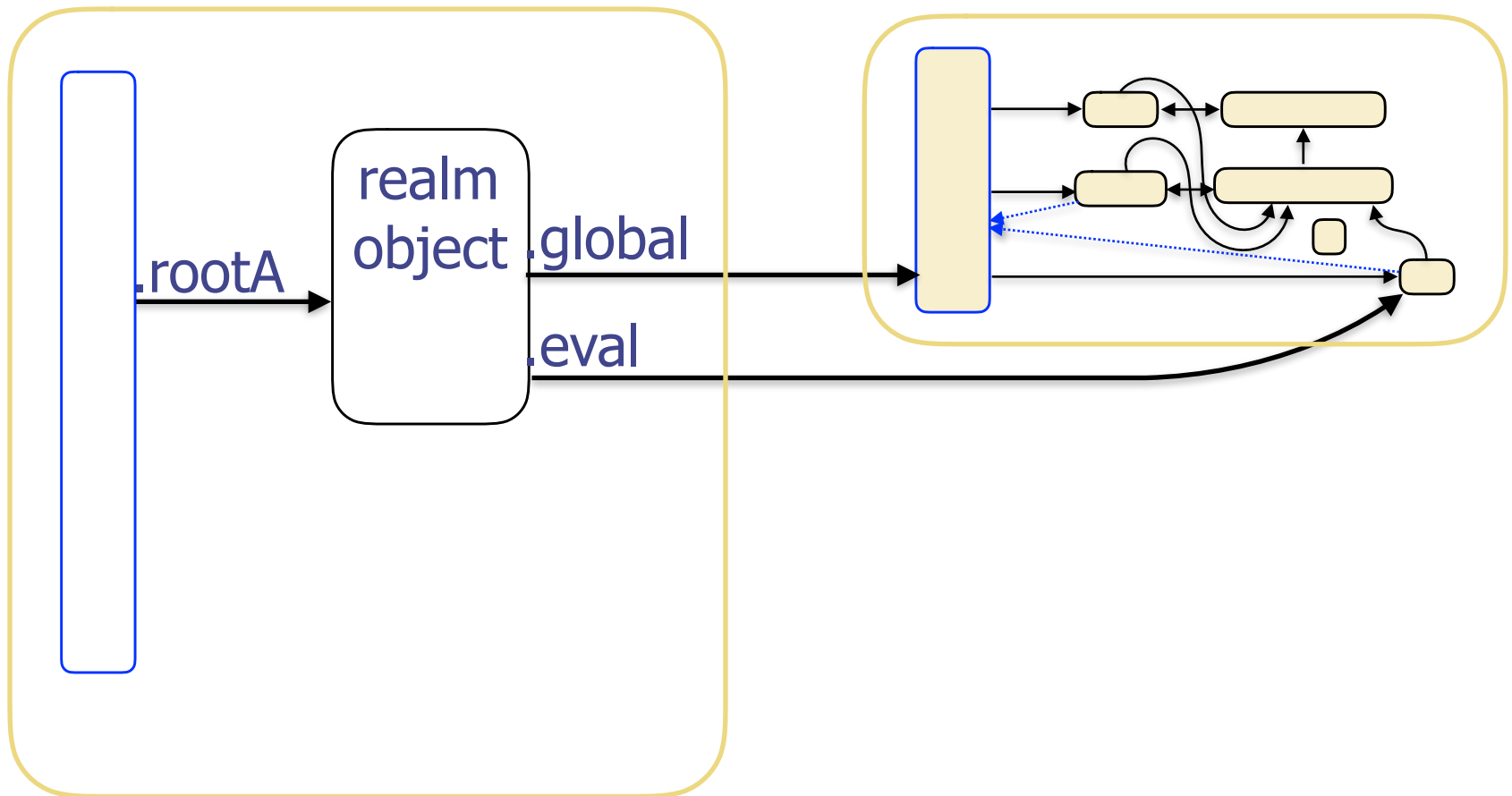
# Creating an immutable root realm

Alice says: `const rootA = Realm.immutableRoot();`



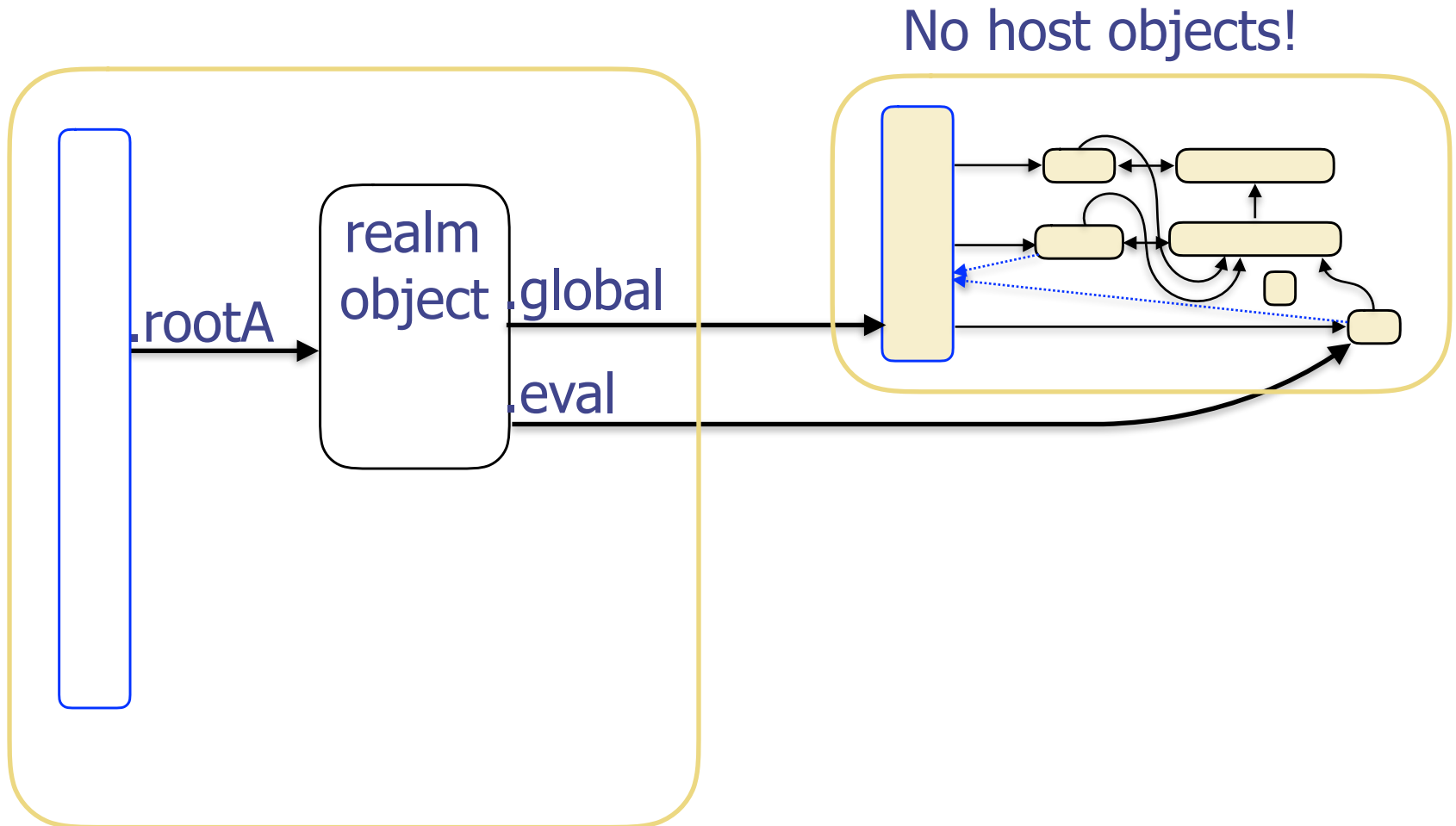
# Creating an immutable root realm

---



# Creating an immutable root realm

---

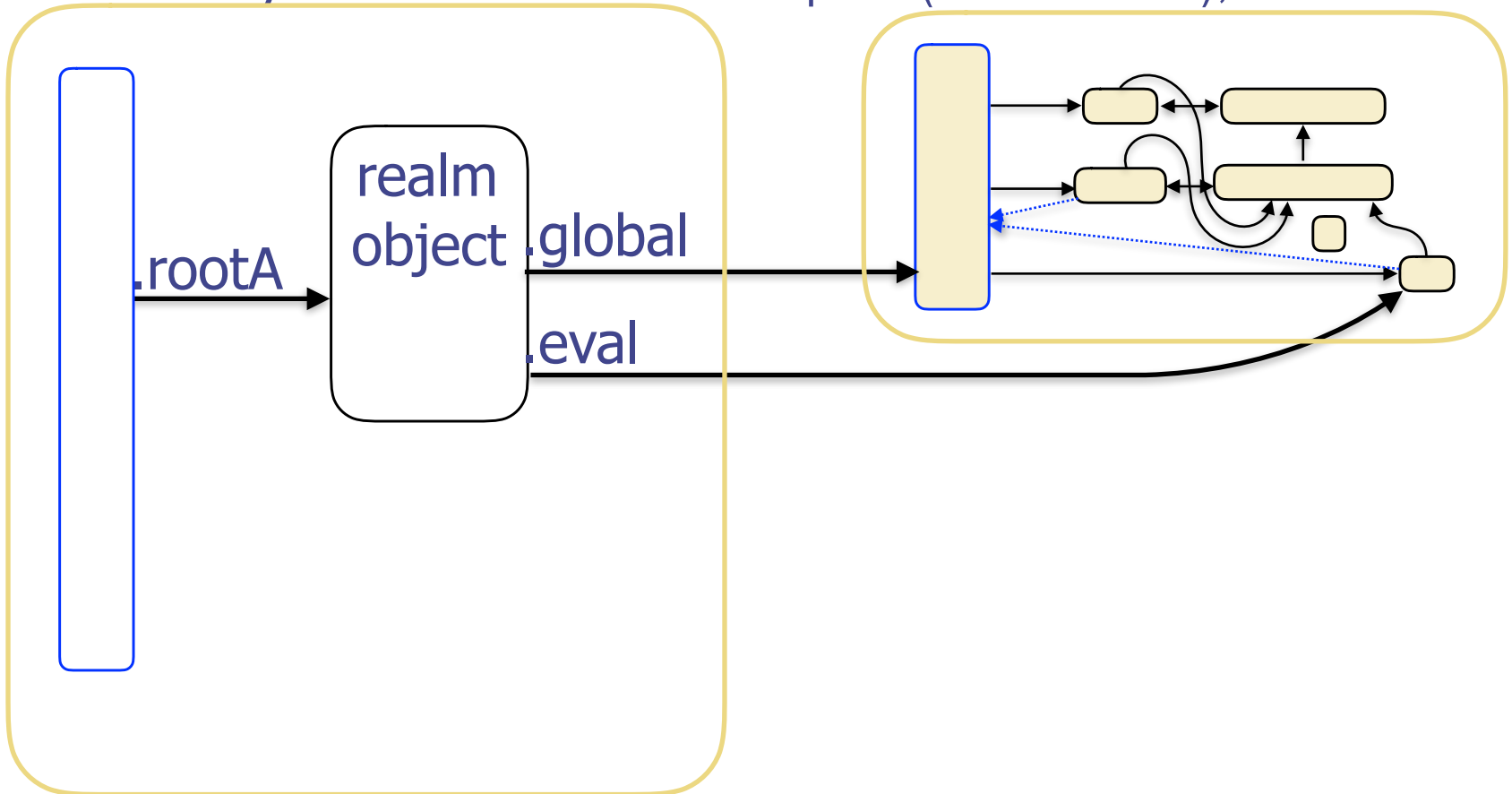




# Spawning a lightweight child realm

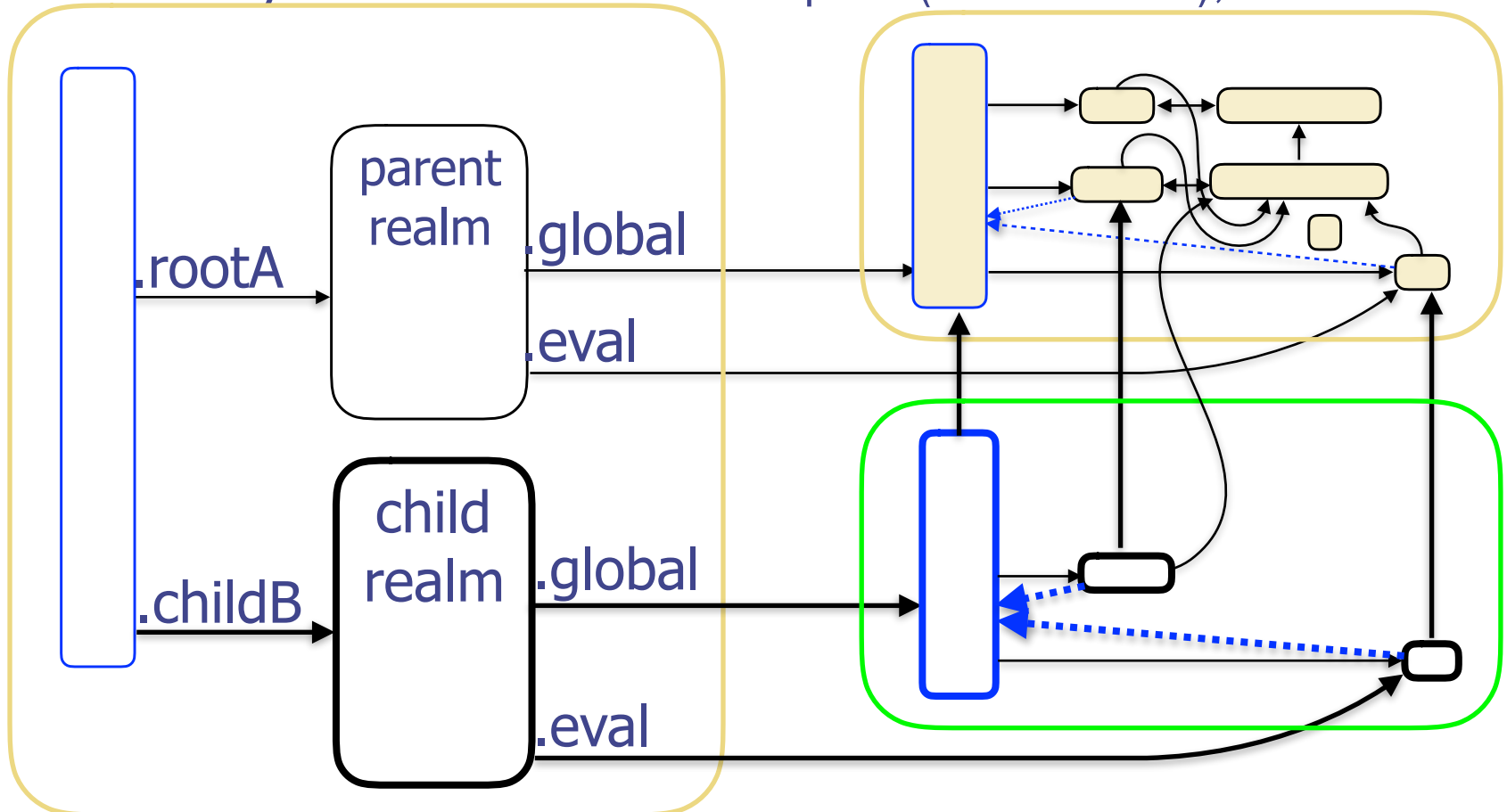
---

Alice says: `const childB = rootA.spawn(endowments);`



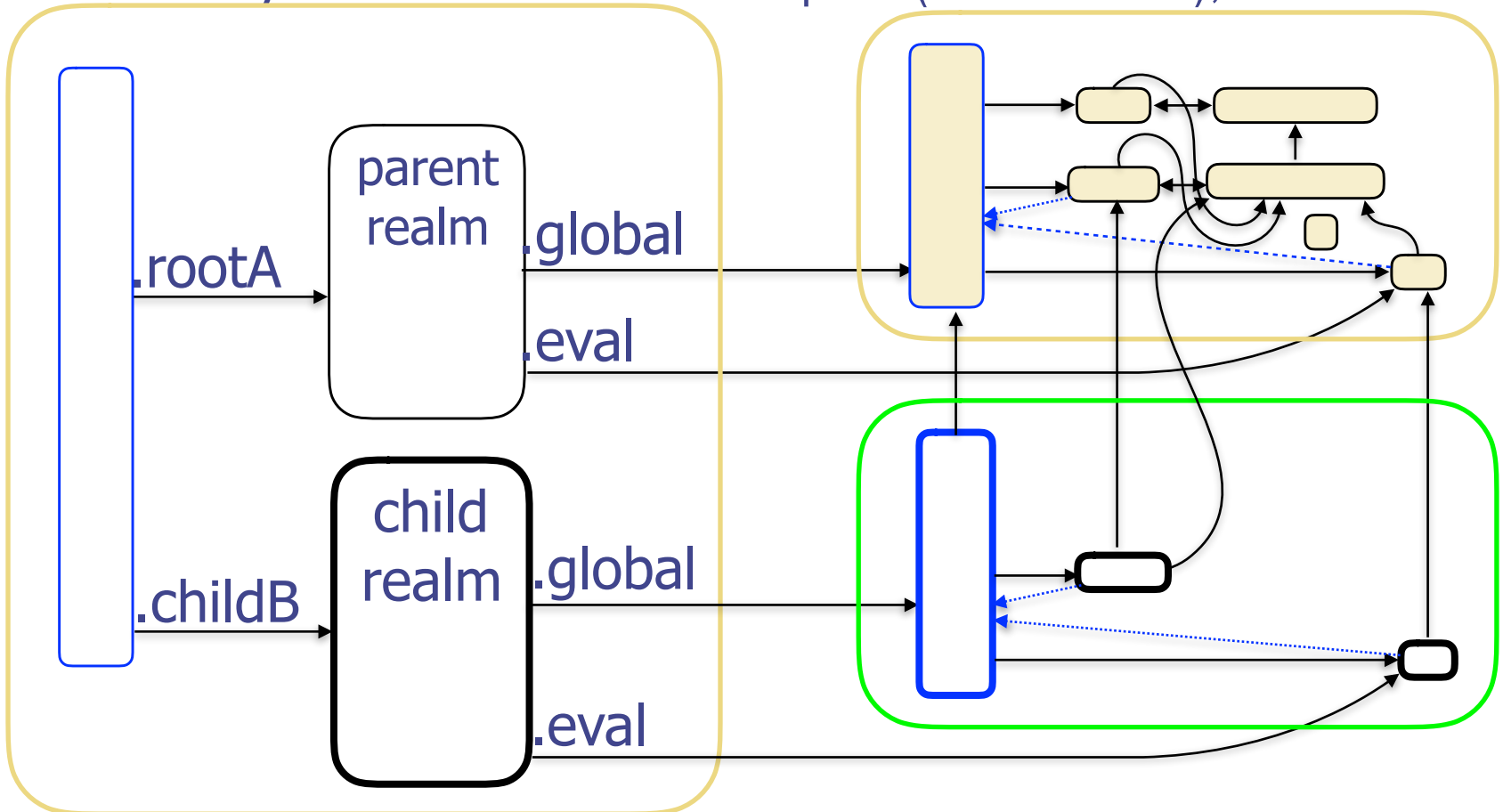
# Spawning a lightweight child realm

Alice says: `const childB = rootA.spawn(endowments);`



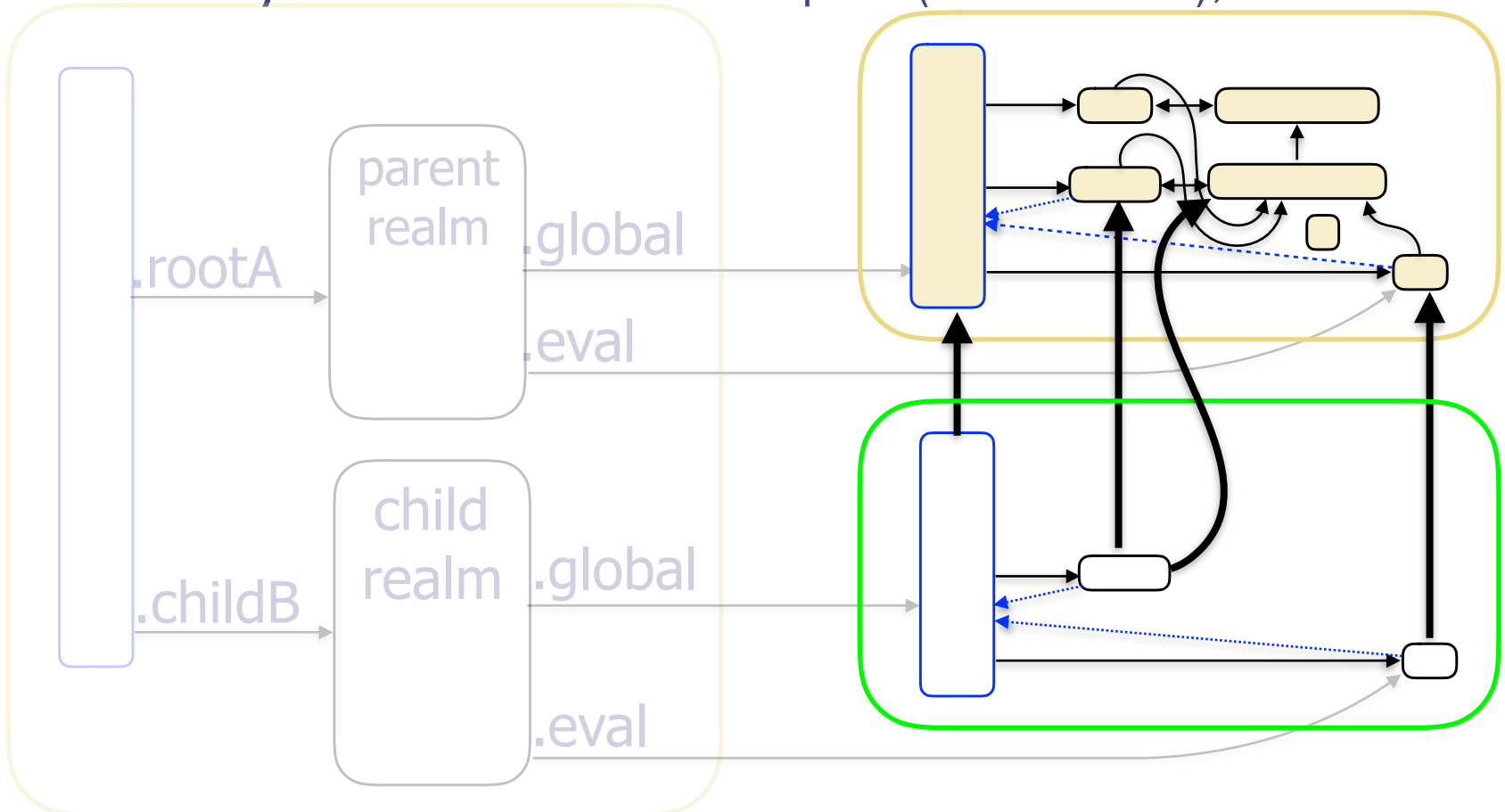
# Lightweight? Only four objects

Alice says: `const childB = rootA.spawn(endowments);`



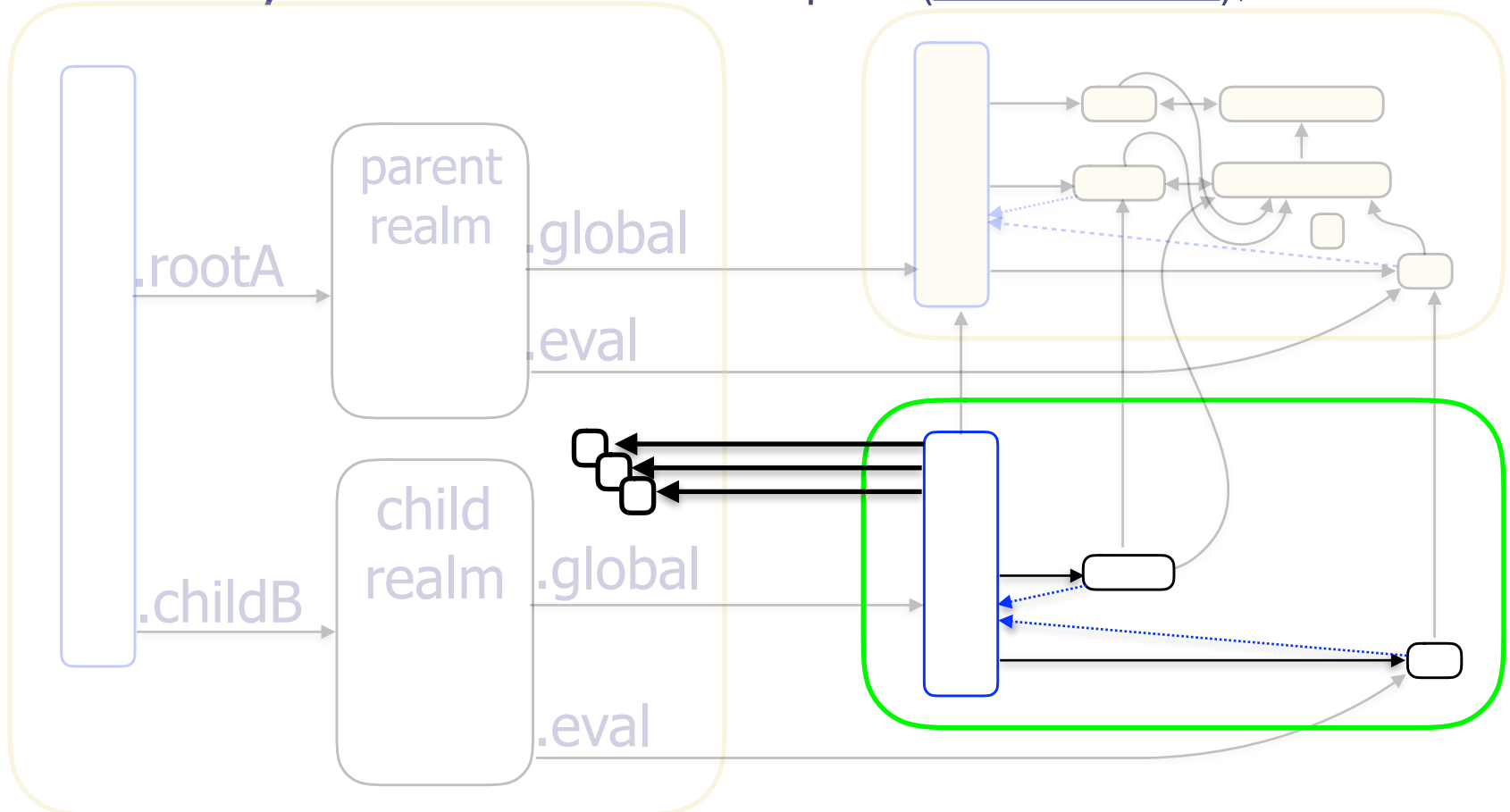
# No **powerful** references by default

Alice says: `const childB = rootA.spawn(endowments);`



# No powerful references **by default**

Alice says: `const childB = rootA.spawn(endowments);`



# Example: Overt Confinement

---

```
function confine(src, endowments) {  
  return rootA.spawn(endowments).eval(src);  
}
```

```
confine("x + y", {x: 3, y: 4}) // 7
```

```
confine("Array", {}) // Array constructor of immutable root realm
```

```
confine("window", {}) // ReferenceError: window not in scope
```

# Alice loads fallible plugins

---

```
// Alice makes API surface
```

```
const counter = Counter();
```

```
// Alice loads fallible plugins bobSrc and carolSrc
```

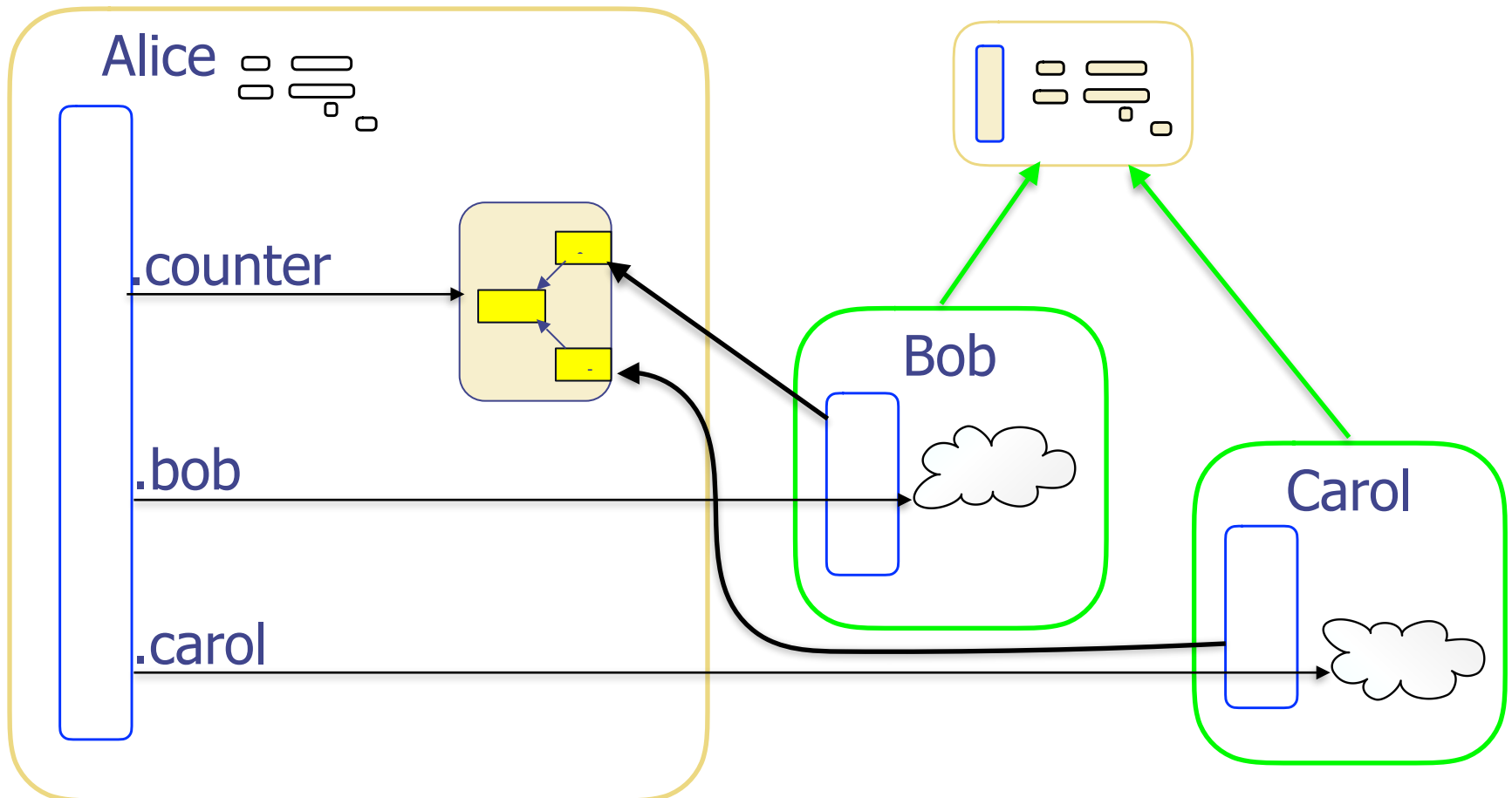
```
const bob = confine(bobSrc, {change: counter.incr});
```

```
const carol = confine(carolSrc, {change: counter.decr});
```

```
// Alice uses bob, carol, counter
```

# Bob and Carol are mostly confined

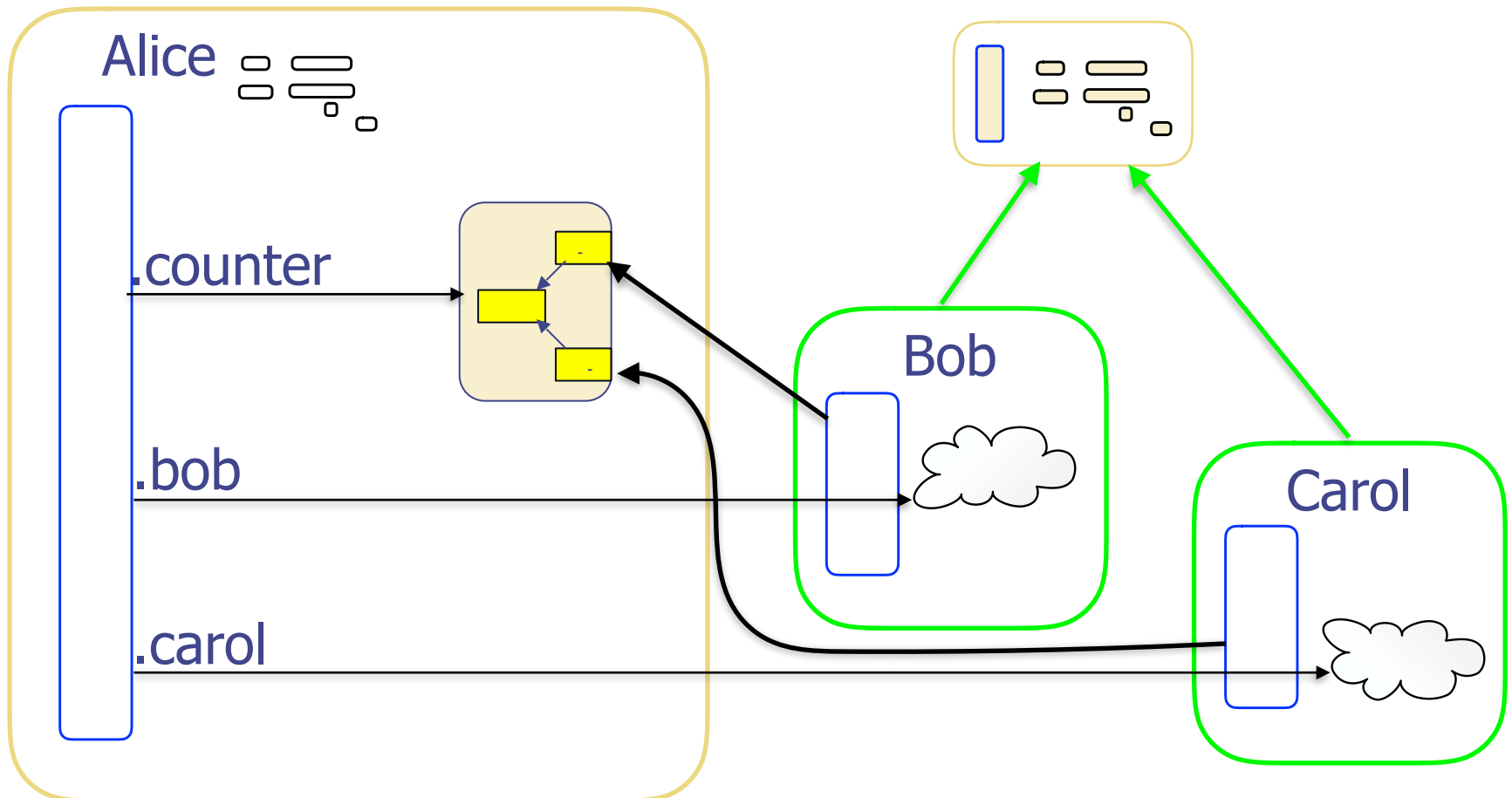
---





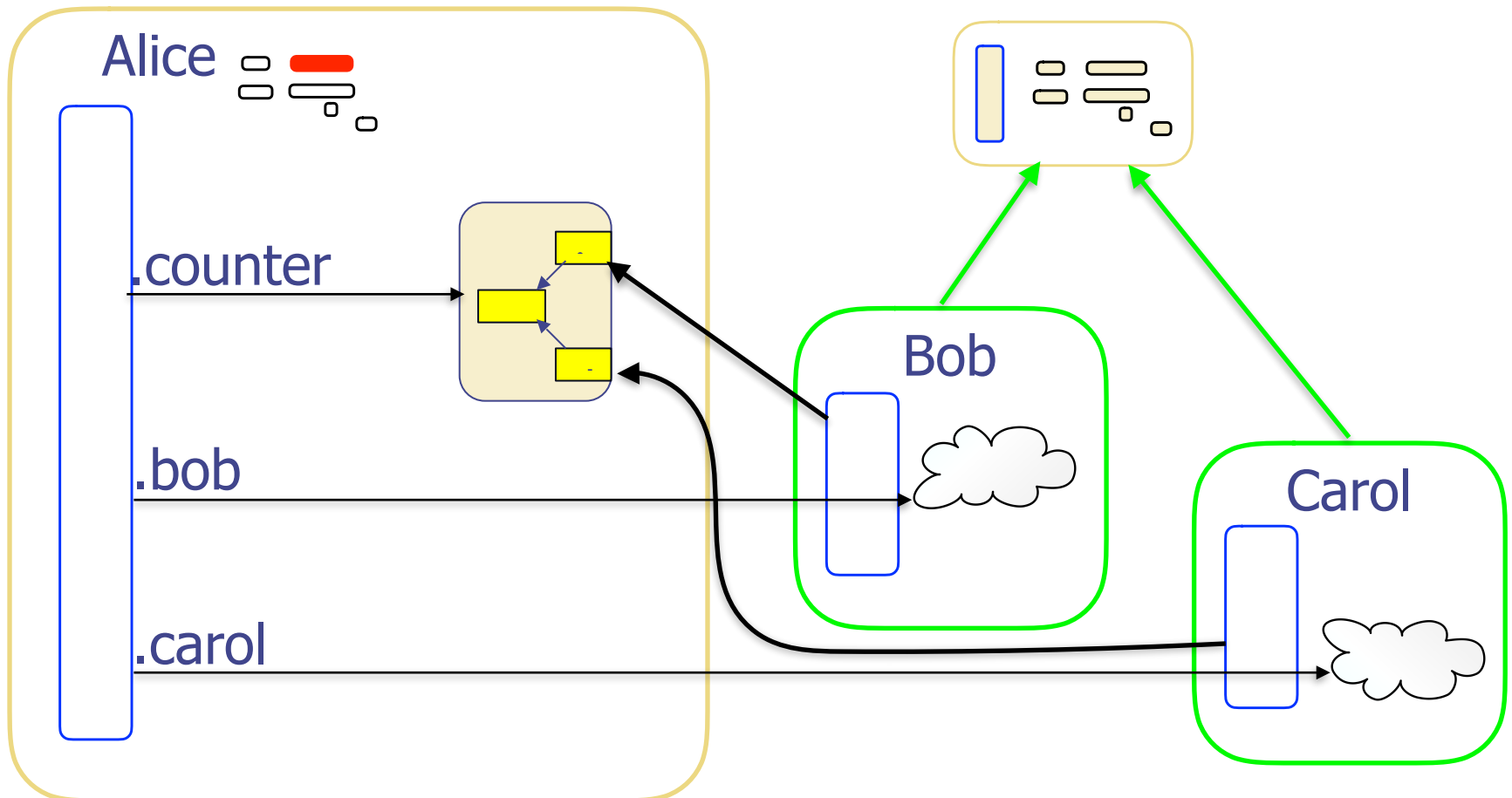
# Is Alice's API surface defensive?

---



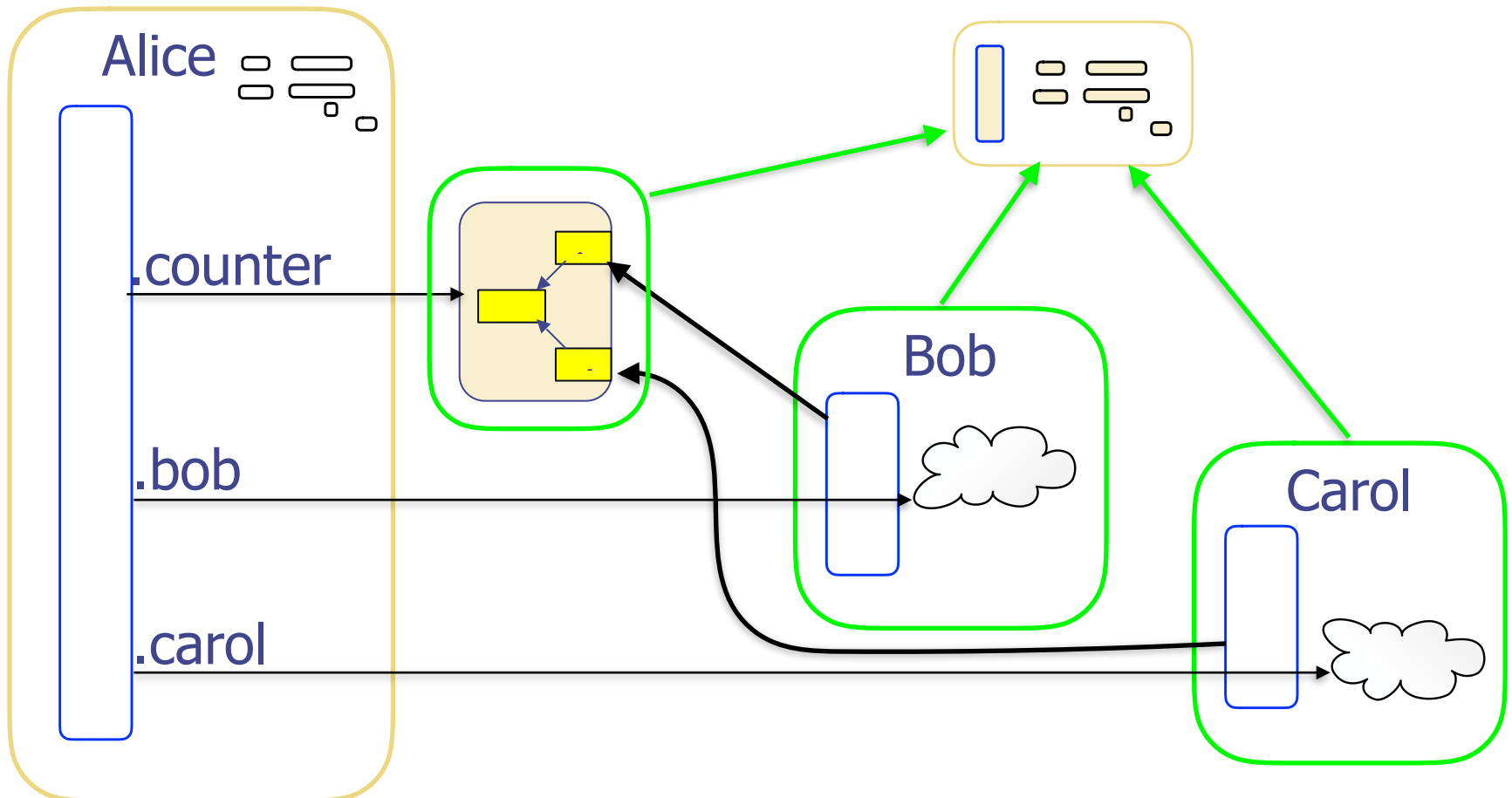
# Is Alice's API surface defensive?

Bob says: `change.__proto__.__proto__.toString = function() { stash = this; };`



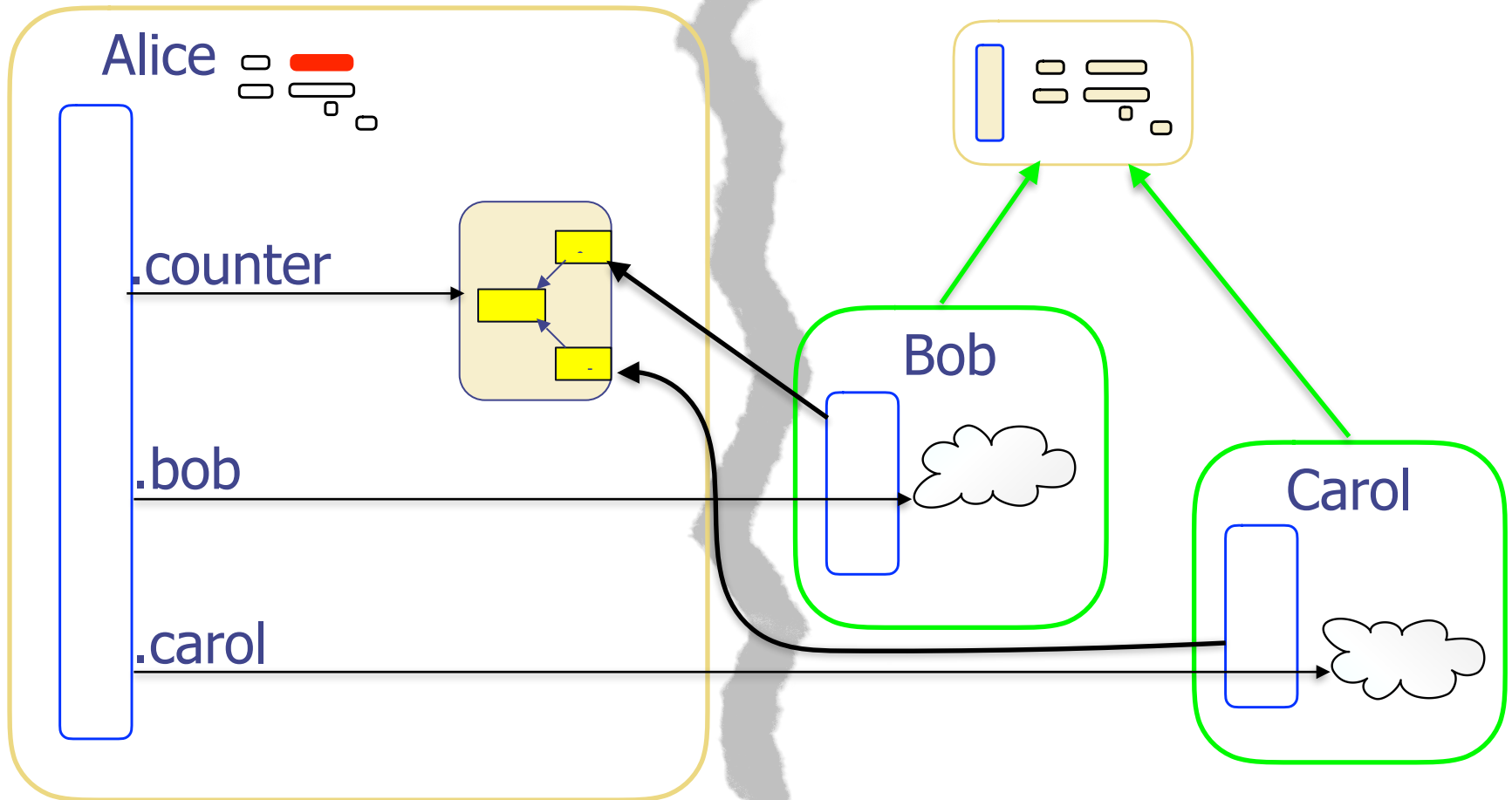
# Alice's API surface is defensive...

...if Alice had said: `const counter = confine(String(Counter), {})(());`



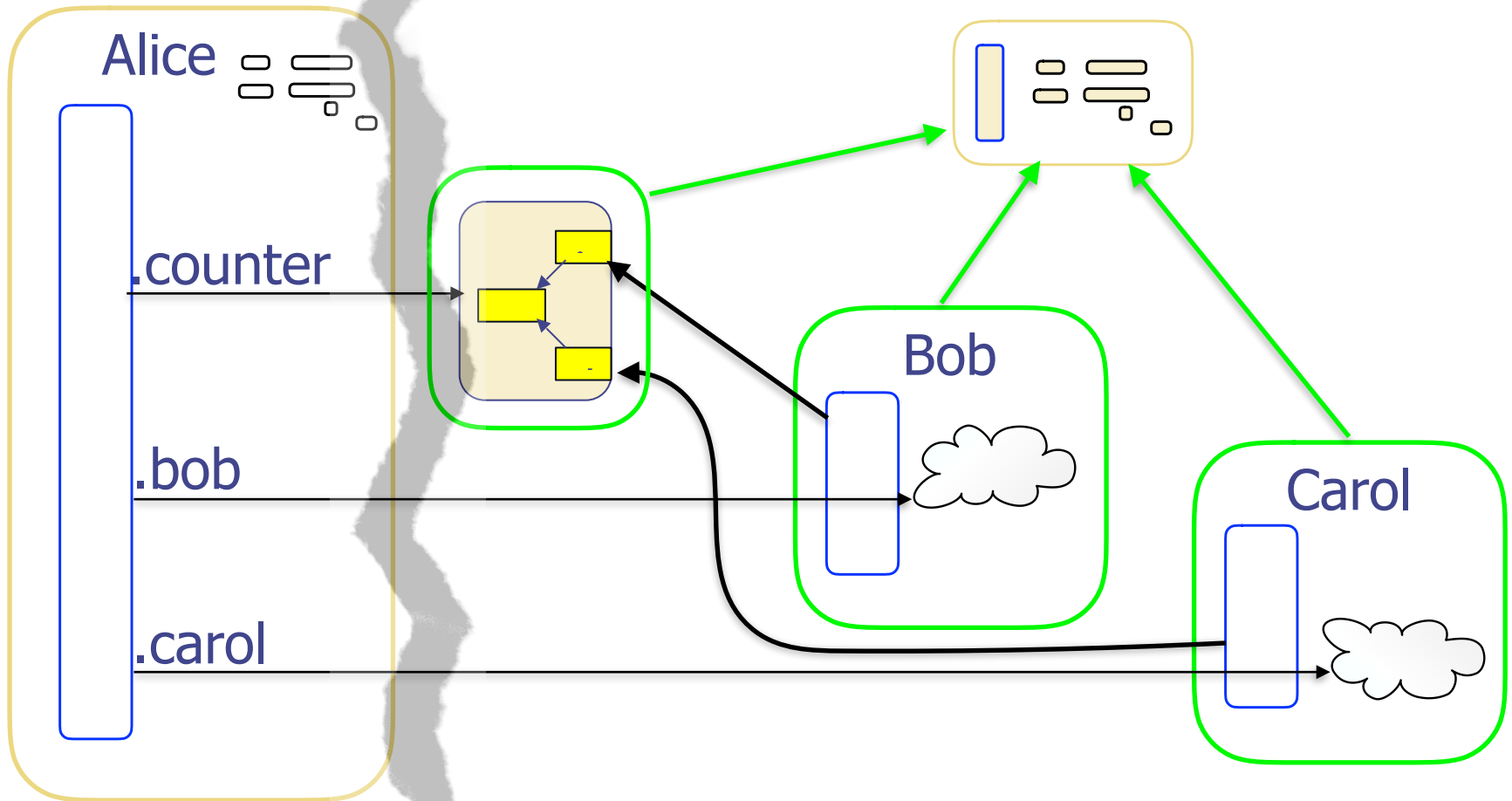
# Discontinuity between API & plugin

---



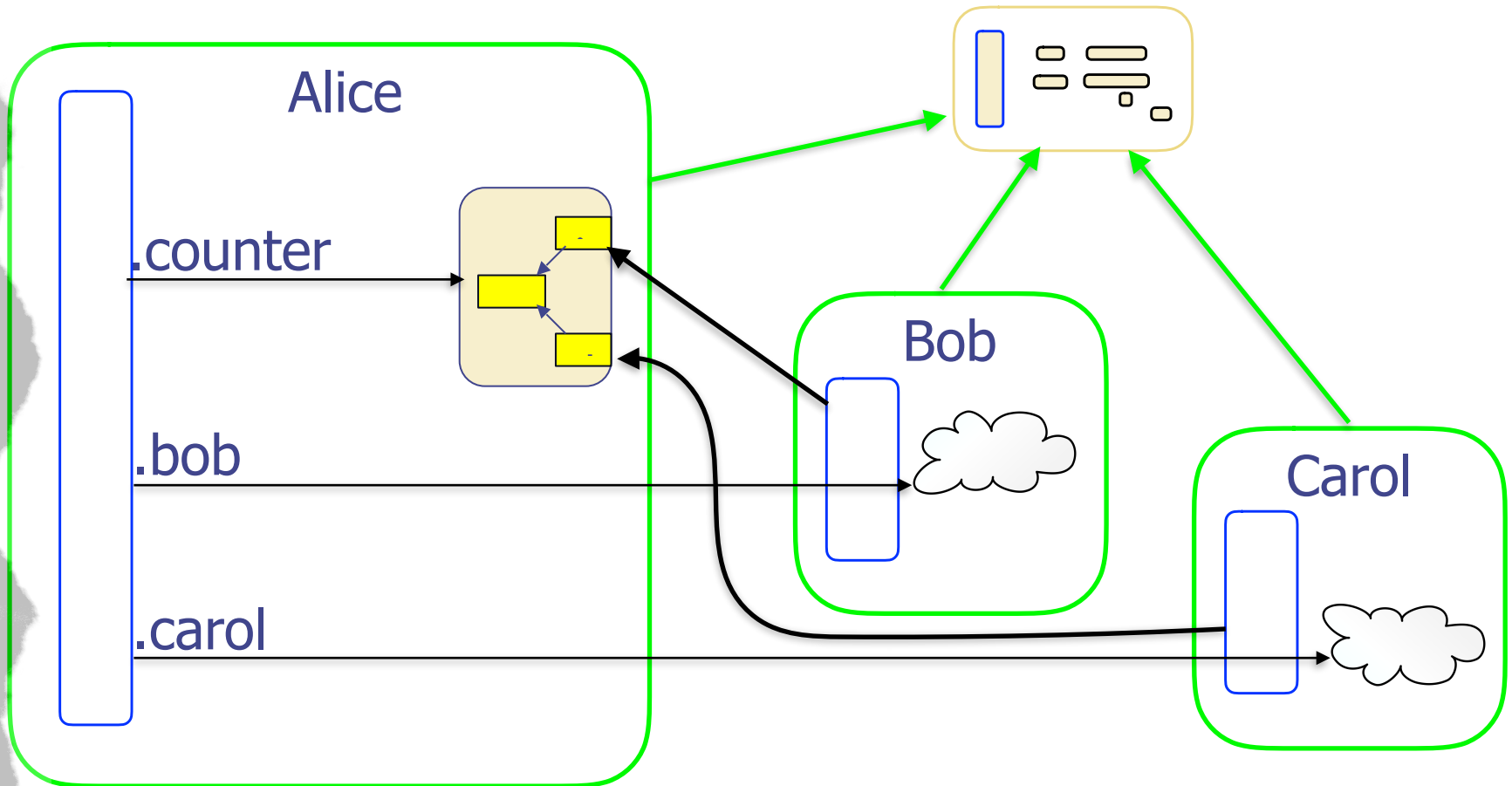
# Discontinuity between Alice & API

---



# Discontinuity between init & Alice

---



# Links

---

**Draft proposal document, more examples**

[github.com/FUDCo/frozen-realms](https://github.com/FUDCo/frozen-realms)

**Relevant ES5-era formal semantics**

[research.google.com/pubs/pub37199.html](https://research.google.com/pubs/pub37199.html)

**ES5-era shim — similar functionality, different API**

[github.com/google/caja/tree/master/src/com/google/caja/ses](https://github.com/google/caja/tree/master/src/com/google/caja/ses)

**ES2015-era shim coming soon**

# Questions?

---

**Draft proposal document, more examples**

[github.com/FUDCo/frozen-realms](https://github.com/FUDCo/frozen-realms)

**Relevant ES5-era formal semantics**

[research.google.com/pubs/pub37199.html](https://research.google.com/pubs/pub37199.html)

**ES5-era shim — similar functionality, different API**

[github.com/google/caja/tree/master/src/com/google/caja/ses](https://github.com/google/caja/tree/master/src/com/google/caja/ses)

**ES2015-era shim coming soon**



---

# Prior language retrofits

---

Scheme

W7

Java

J-Kernel, Joe-E

Mozart/Oz (constraints)

Oz-E

OCaml (ML)

Emily

Squeak (Smalltalk)

Squeak-E

Python

Monte

Pict ( $\pi$  calculus)

Tamed Pict

ES3 (Early JavaScript)

FBJS, MS WebSandbox  
ADSafe, Jacaranda, JSand,  
Cajita, Valija, ES5/3

# Why do we retrofit JavaScript?

---

Because that's where the attack surface is.

(with apologies to Willie Sutton)

# What JavaScript always got right

---

“user-mode” vs “system-mode” separation

ECMA vs W3C jurisdiction boundary

organizations which design systems ... produce  
designs which are copies of the communication  
structures of these organizations

—Conway’s Law

# What JavaScript always got right

---

No undefined behaviors!

Few implementation-defined behaviors

Little room for accidental non-determinism

# What JavaScript always got right

---

Effects **only** by held references

No “import” (yet)

I/O reached only by scoping (window, document, ...)

Multiple Isolated Realms

mobile code as protocol

Great concurrency model

Shared-nothing Communicating Event Loops