Report on the Realms Shim Security Review

JF Paradis, Salesforce
Brian Warner, Agoric
Mark S. Miller, Agoric
Dean Tribble, Agoric

Thanks to Realms, SES Meetings attendees

Overview

Realms

Shim

Security review

Overview

Realms — what and why?

Shim — how?

Security review — whether?

- Whether ready for production use now?

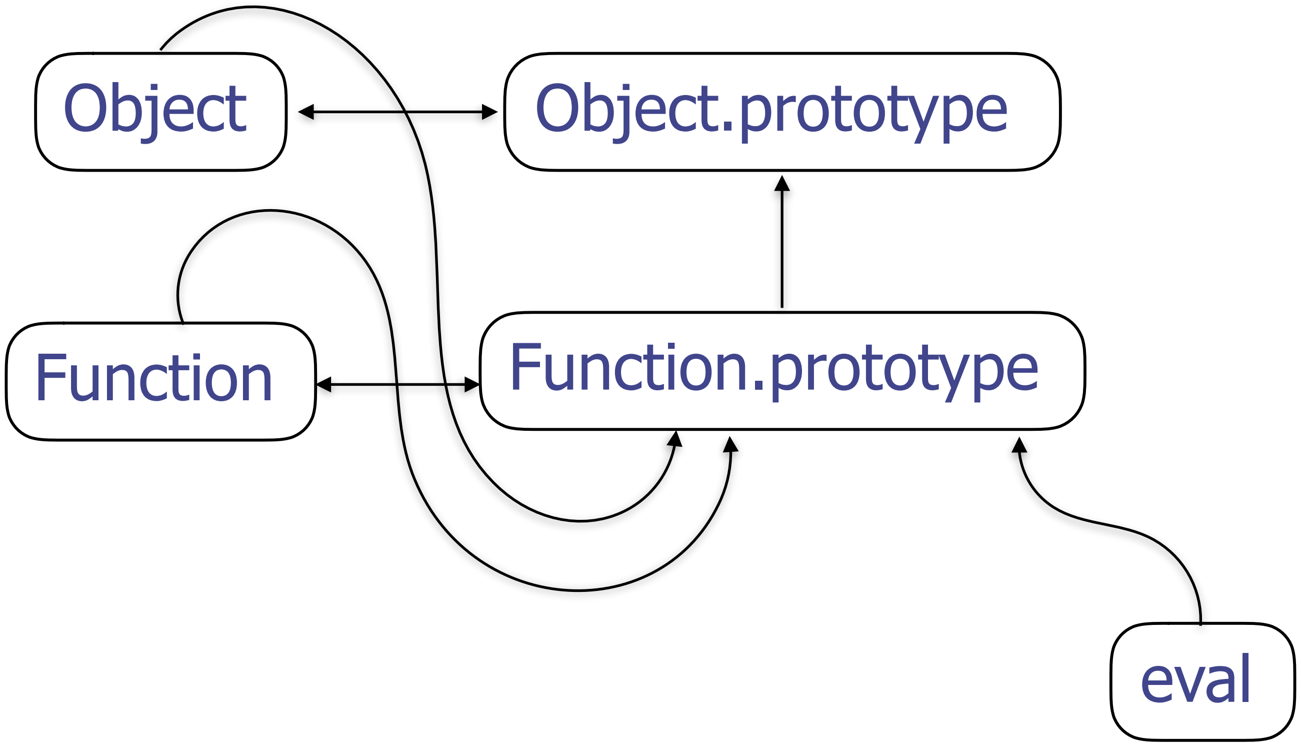
Realms

Realms

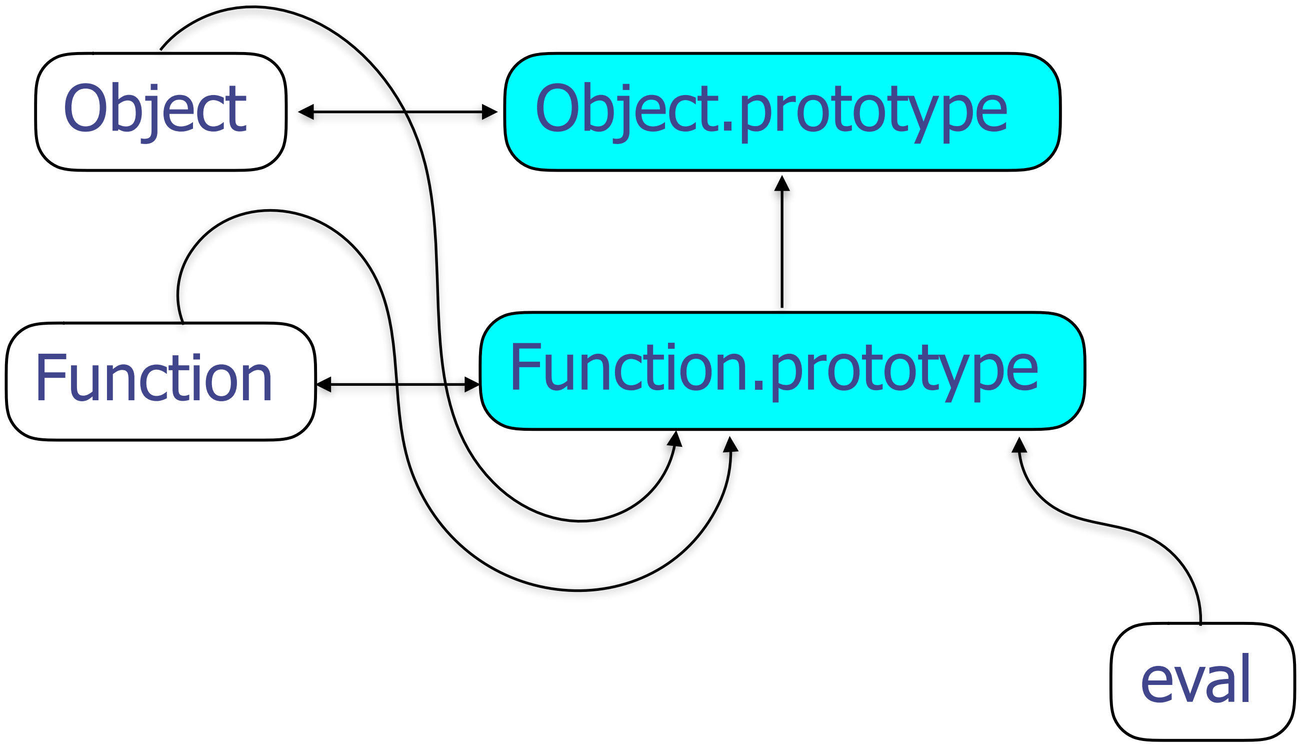
What and why?

With shim API we reviewed

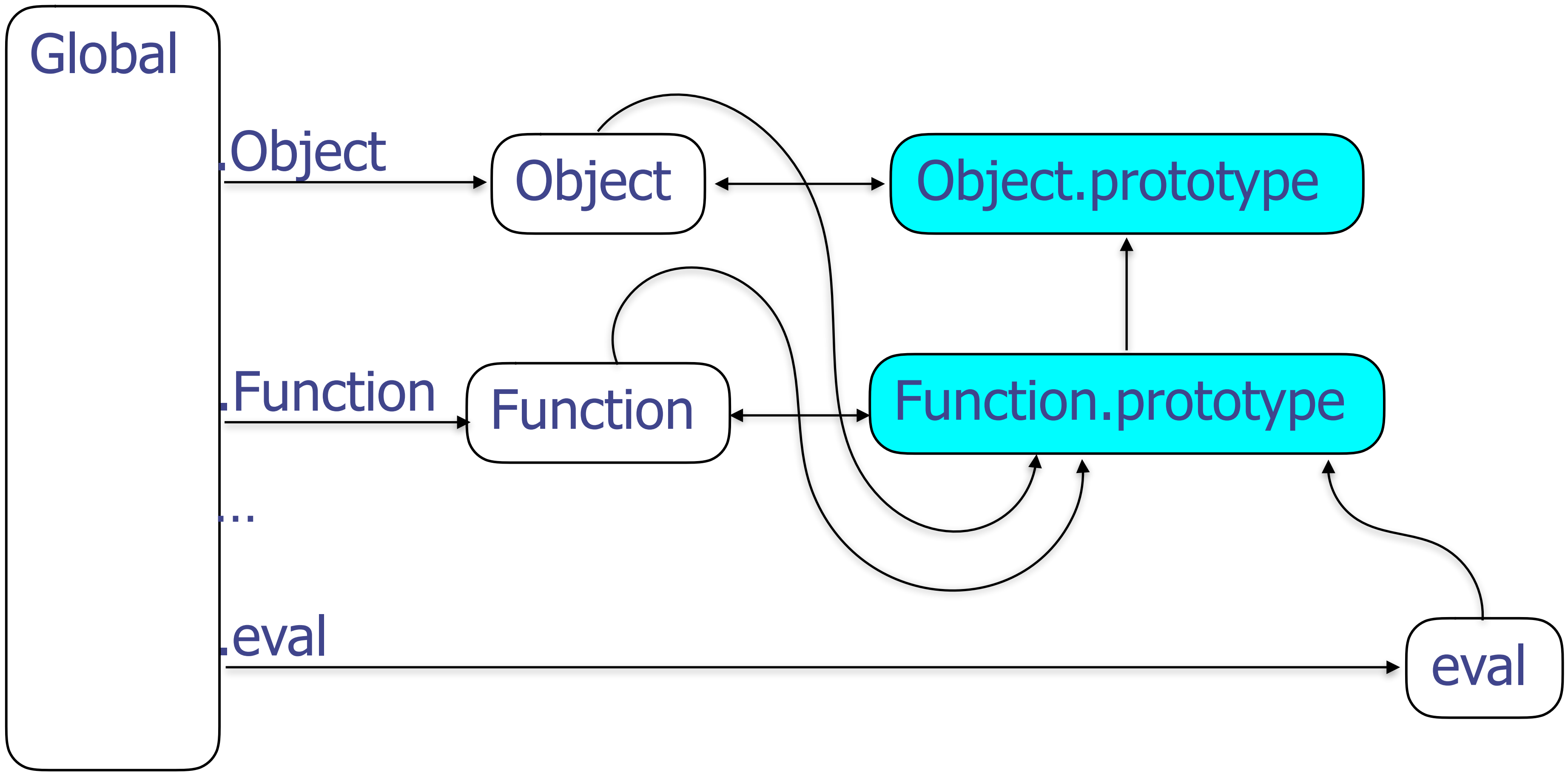
Primordials exist before code runs



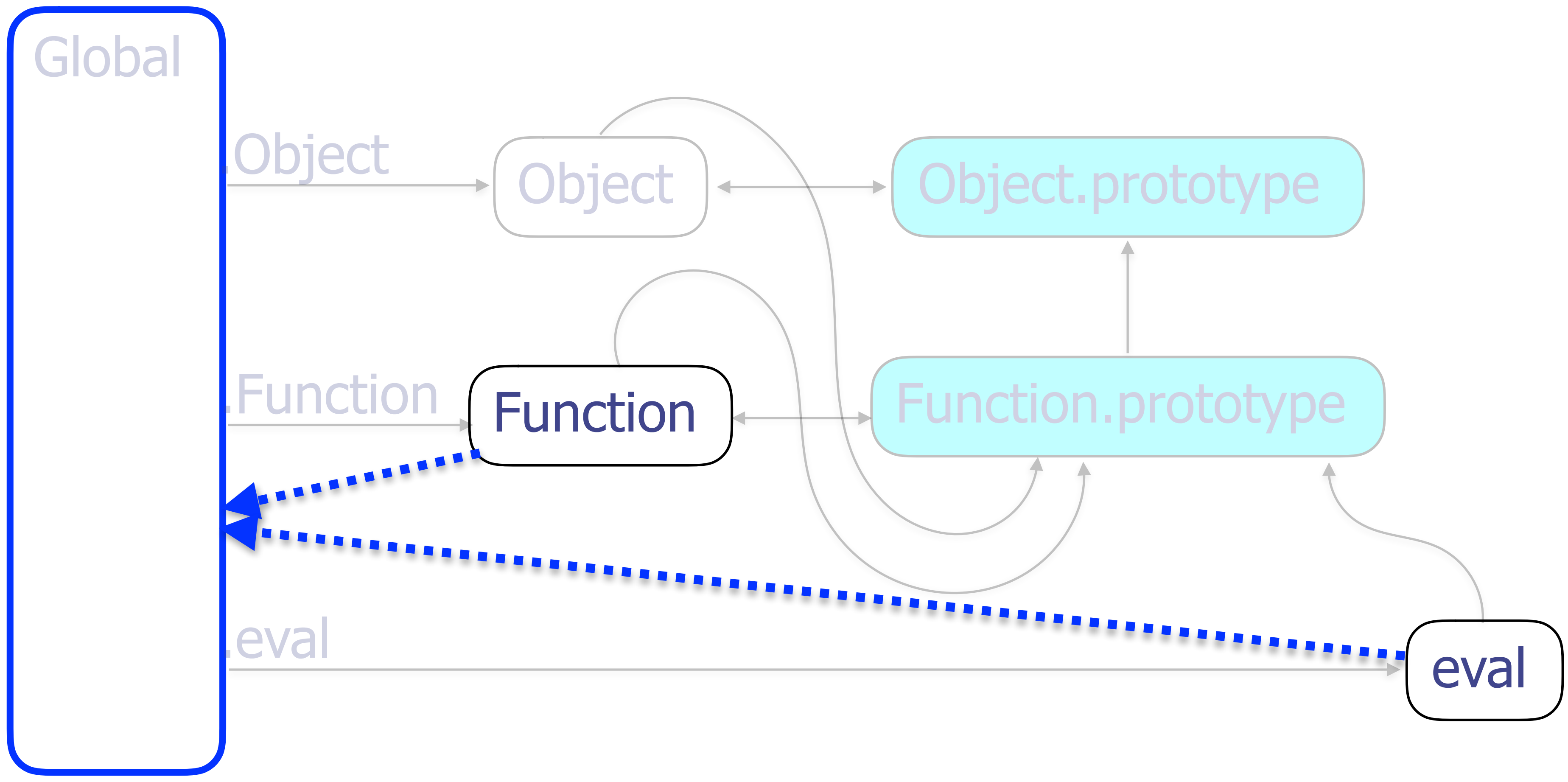
Undeniables are reachable by syntax



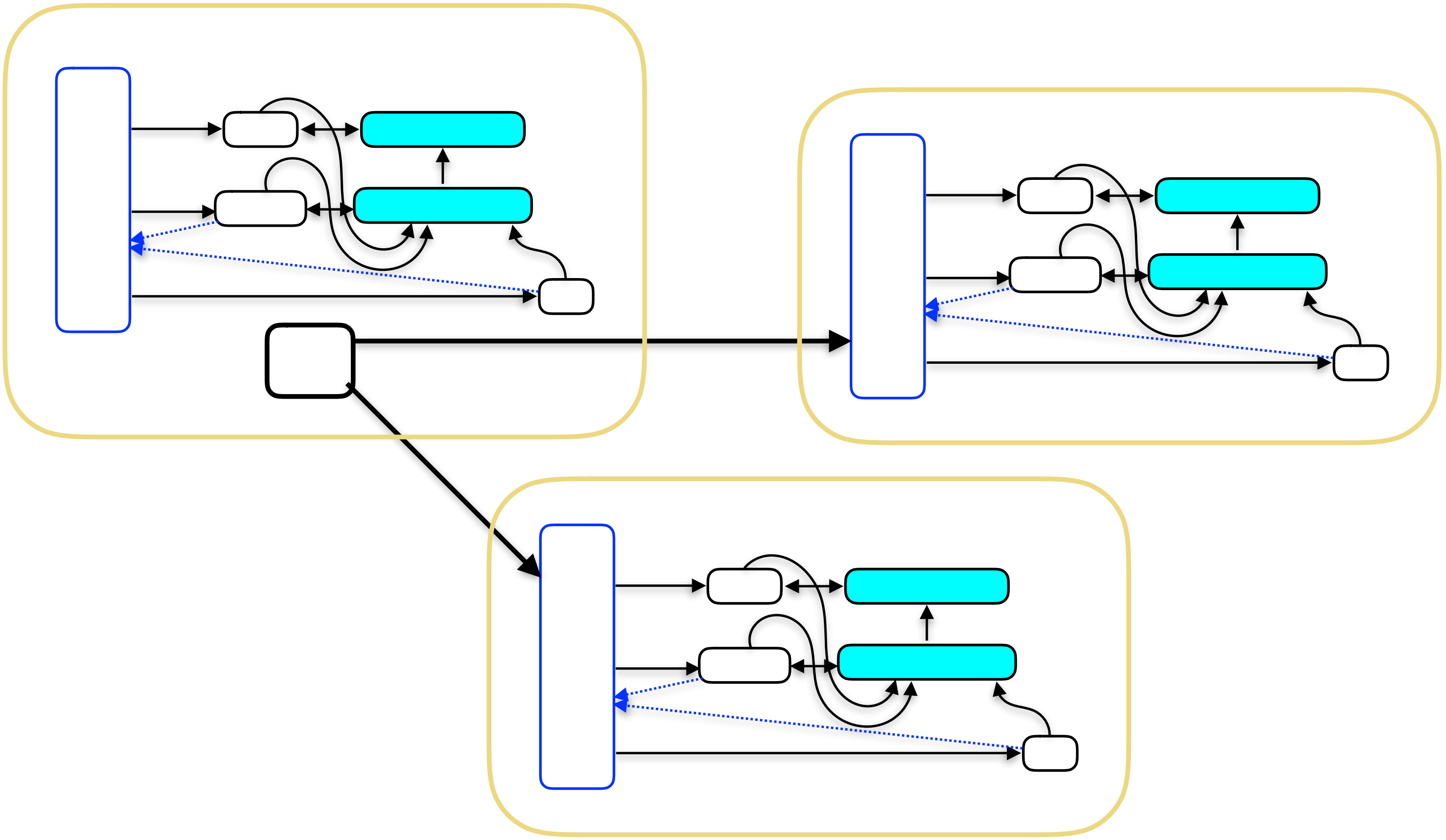
Global object



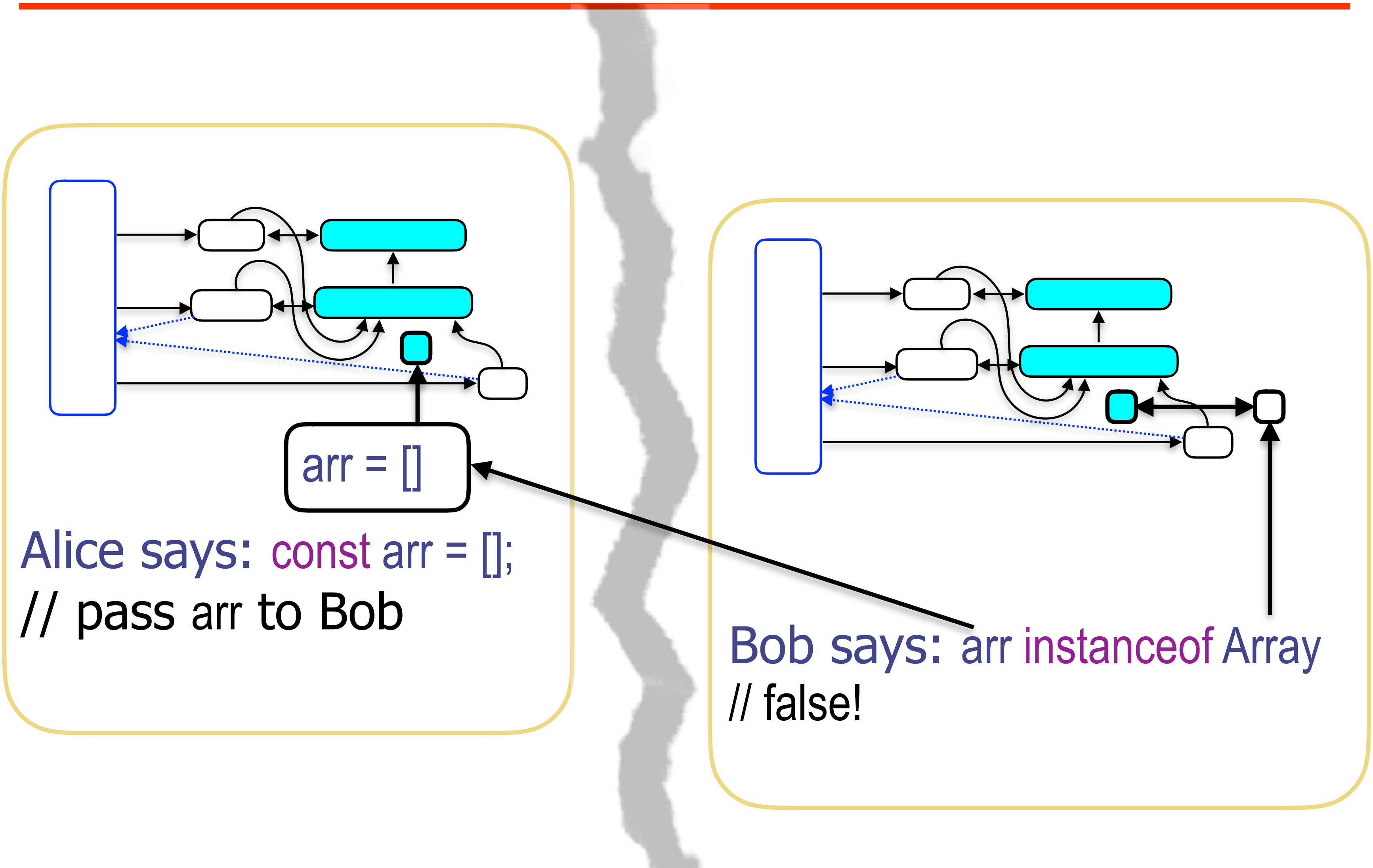
Evaluators: eval, Function
Evals code in scope of global's names



Multiple isolated realms today (same-origin iframes) (vm.createContext)



Identity Discontinuity

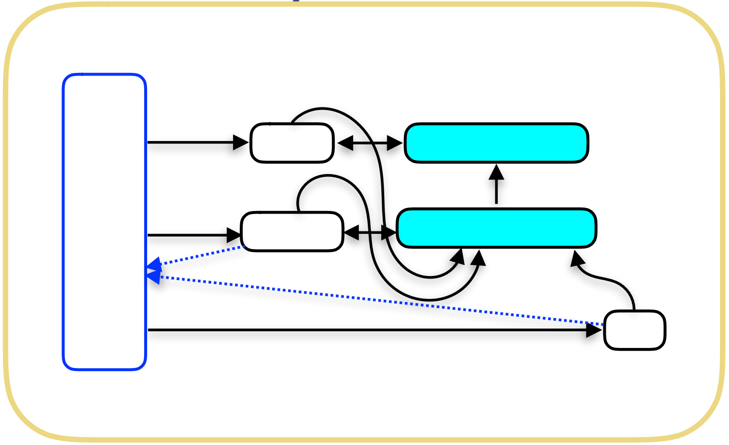


Alice says: const arr = [];
// pass arr to Bob

Bob says: arr instanceof Array
// false!

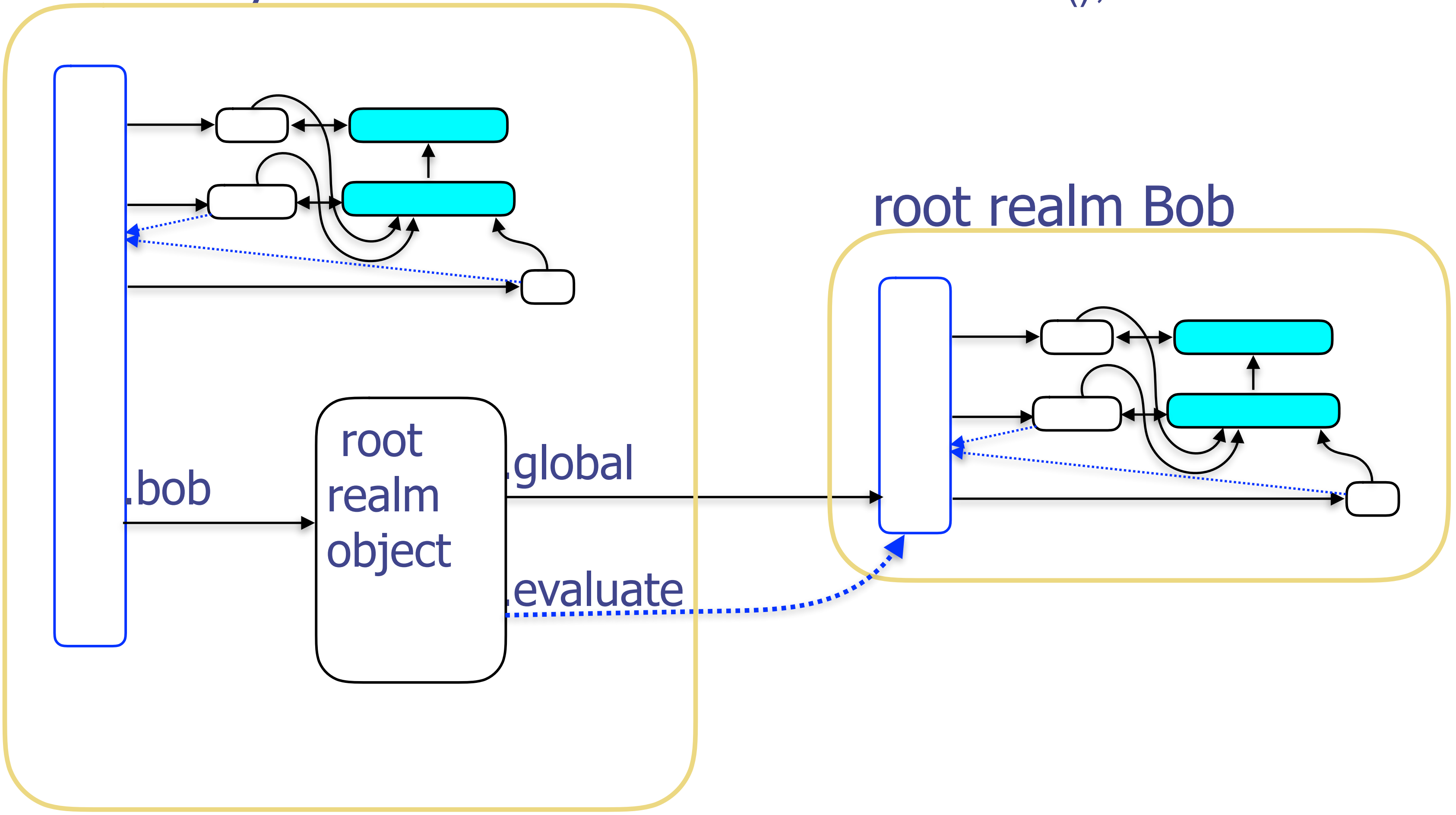
Make root realm Bob

Alice says: `const bob = Realm.makeRootRealm();`



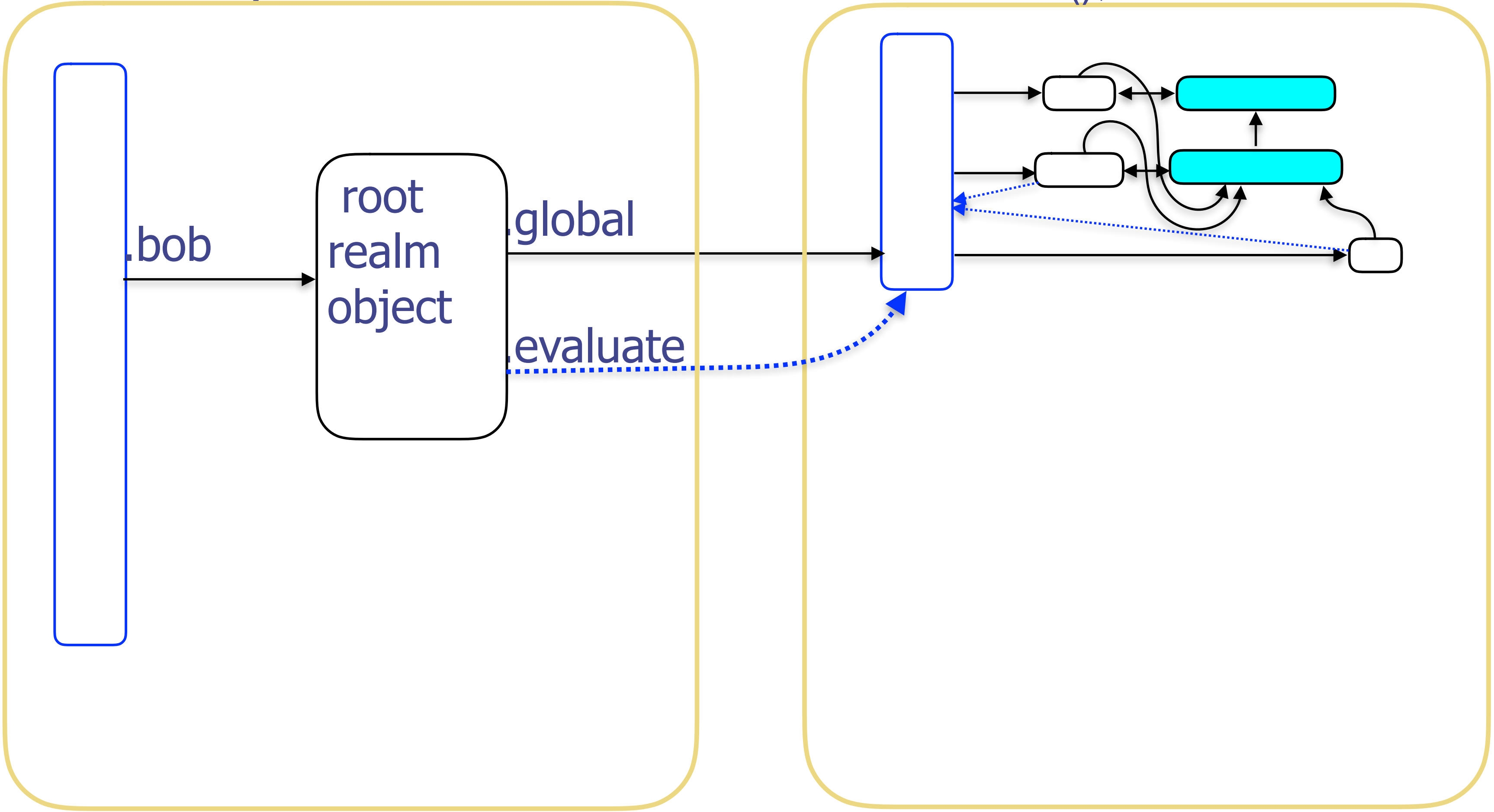
Make root realm Bob

Alice says: const bob = Realm.makeRootRealm();

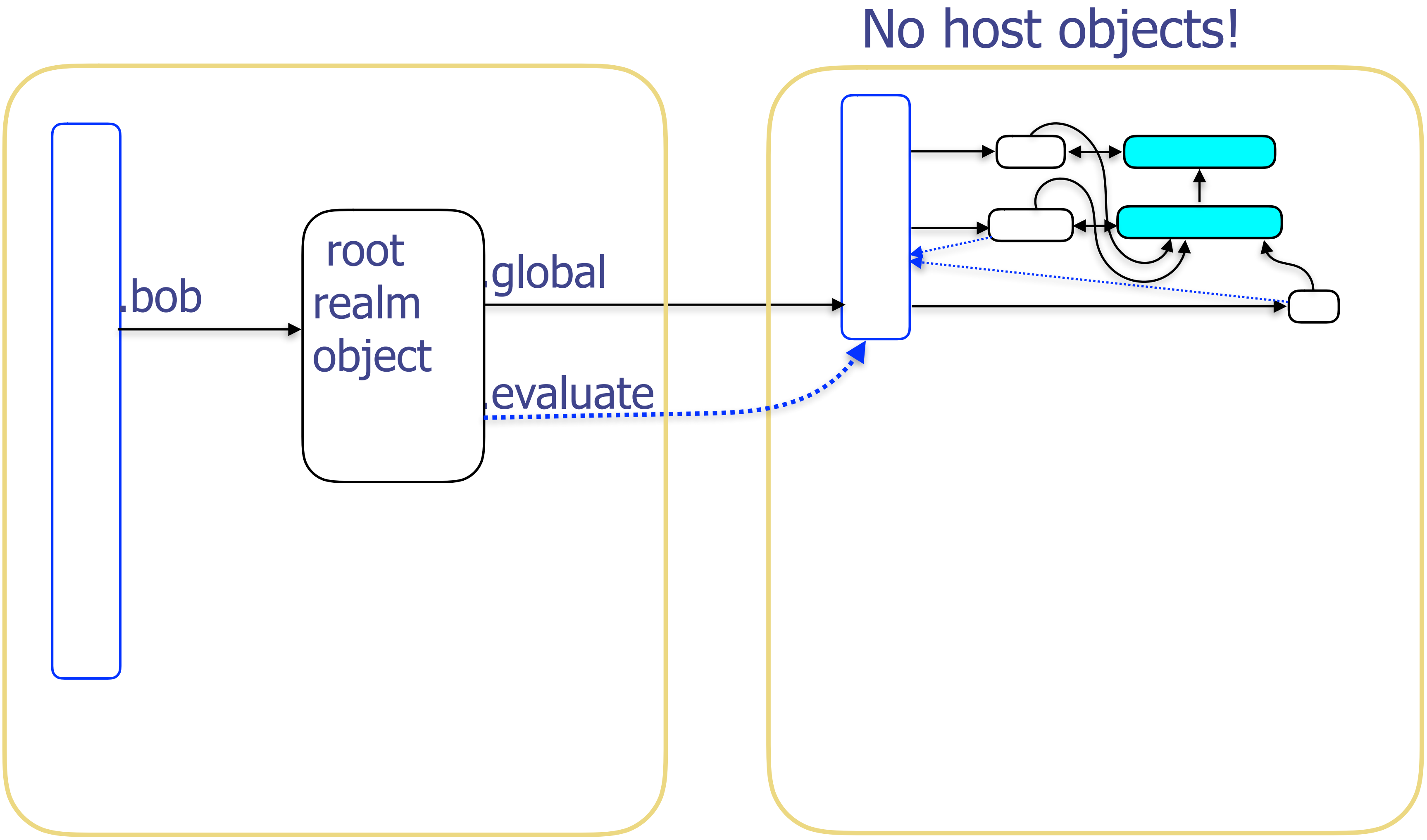


Make root realm Bob

Alice says: const bob = Realm.makeRootRealm();

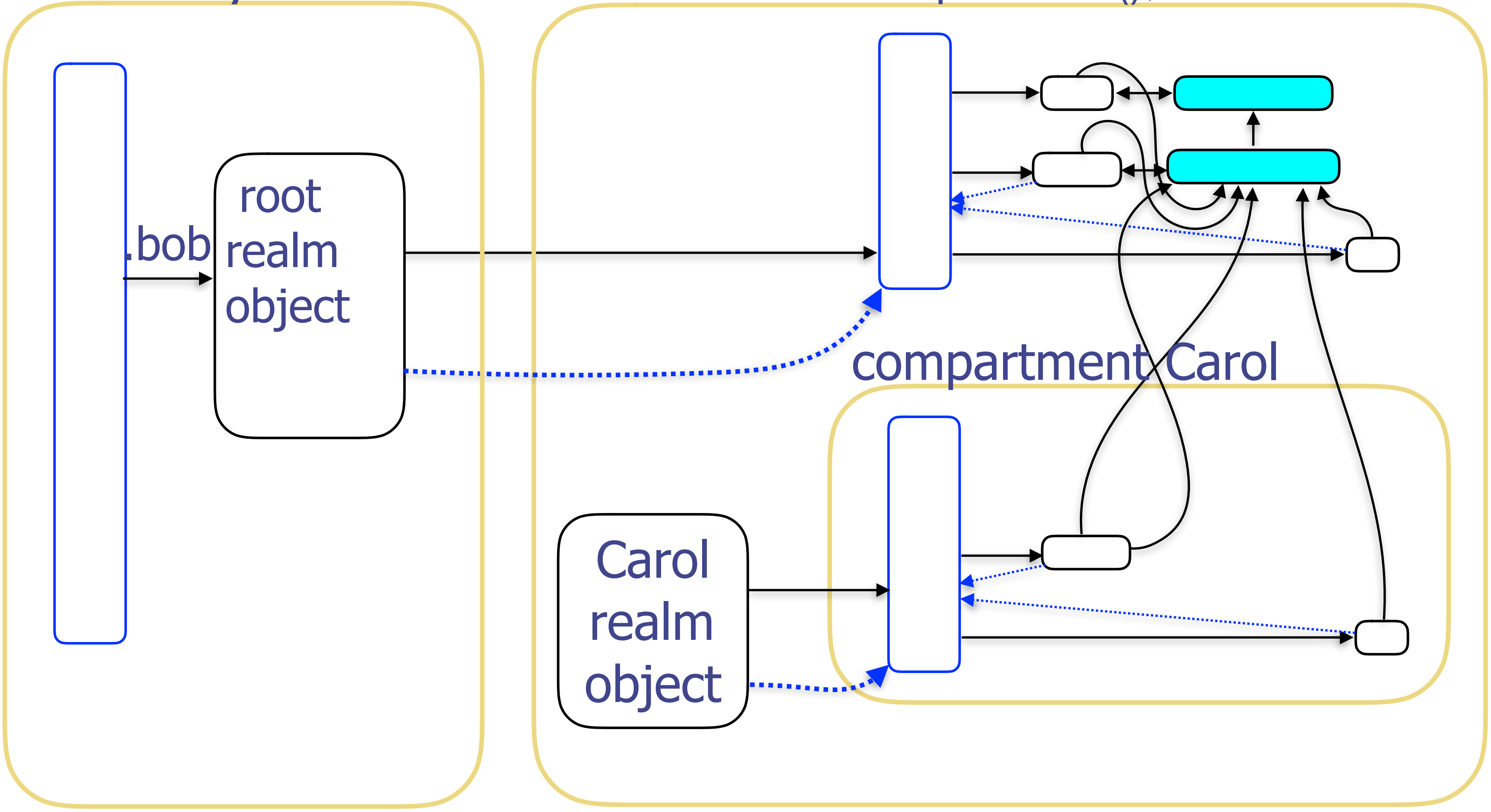


Perfect Sandbox



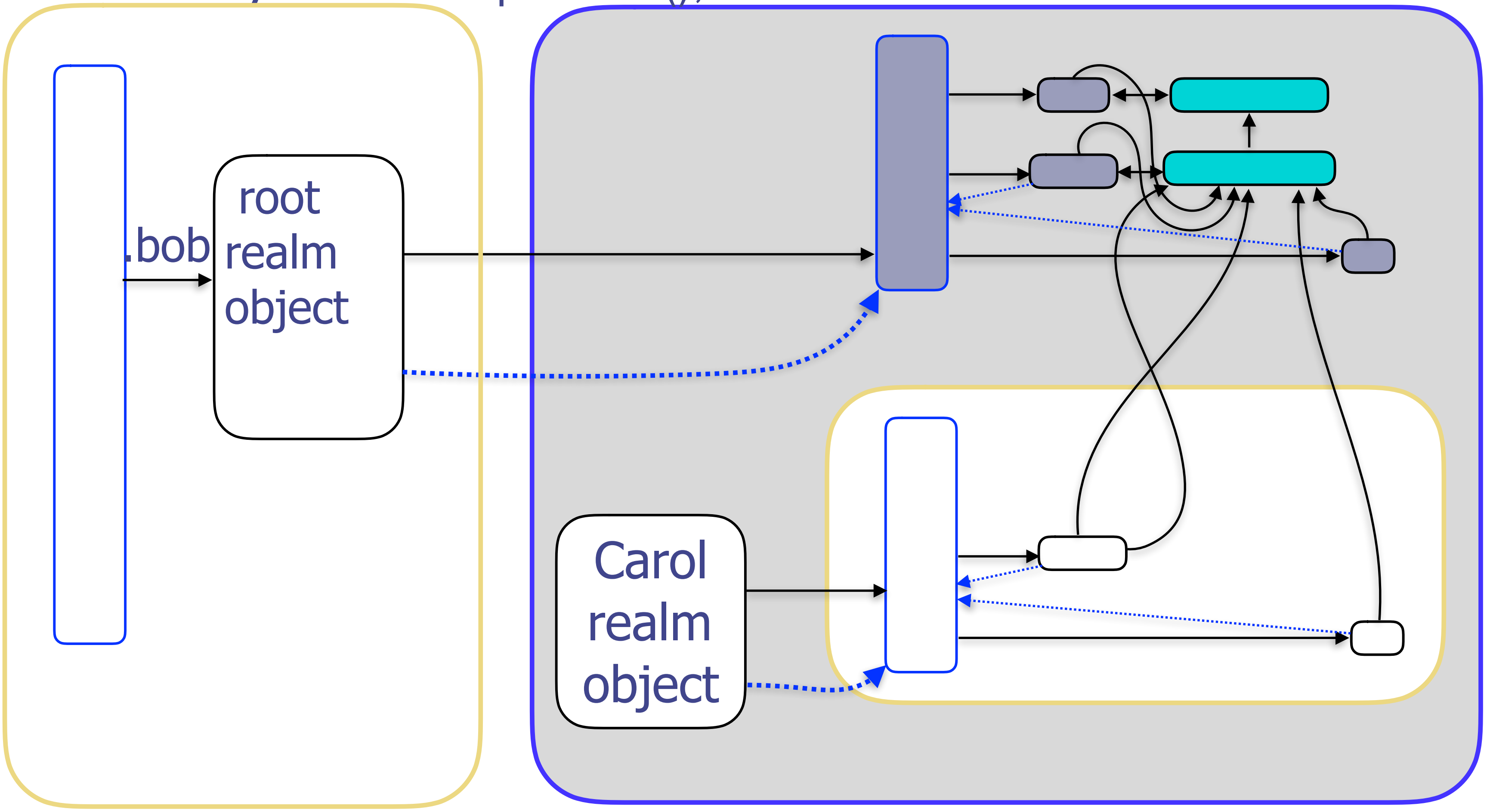
Make compartment carol

Bob says: `const carol = Realm.makeCompartment();`



Featherweight protection domains.
No identity discontinuity!

Alice says: bob.deepFreeze();



Shim

Shim

how?

On major platforms today

The heart of the shim

Evaluate code from user

- Redirect all free variable access + `this`
- Confine effects
- No parsing, but…
- No rewriting — evaluate Dr. Evil’s string as is
- Typical case is fast!

8 Magical Lines of JavaScript

```
return unsafeFunction(`
 with (arguments[0]) {
   ${ optimizer }
   return function() {
     “use strict”;
     return eval(arguments[0]);
   };
 }`);
```

Without optimization

```
return function() {
  with (arguments[0]) {

    return function() {
      “use strict”;
      return eval(arguments[0]);
    };
  }
};
```

Without optimization

```
return function() {
  with (arguments[0]) {

    return function() {
      "use strict";
      return eval(arguments[0]);
    };
  }
};
```

Direct eval

Direct eval — like inline anti-quote

```
…code A…;
eval(“code B”);
…code C…;
```

is like

```
…code A…;
{…code B…;}
…code C…;
```

Direct eval — like inline anti-quote

```
…code A…;
eval(“code B”); // but could be computed
…code C…;
```

is like

```
…code A…;
{…code B…;}
…code C…;
```

No extra variables in scope

```
return function() {
  with (arguments[0]) {

    return function() {
      “use strict”;
      return eval(arguments[0]);
    };
  }
};
```

Direct eval

No extra variables in scope

Sloppy

```
return function() {
  with (arguments[0]) {

    return function() {
      "use strict";
      return eval(arguments[0]);
    };
  }
};
```

Direct eval

Sloppy

No extra variables in scope

scopeProxy

```
return function() {
  with (arguments[0]) {

    return function() {
      "use strict";
      return eval(arguments[0]);
    };
  }
};
```

src string

Direct eval

Applying the magic

```
return function() {
 with (arguments[0]) {

  return function() {
   “use strict”;
   return eval(arguments[0]);
  };
 }
};
```

Applying the magic

```
return function() {
  with (arguments[0]) {
    return function() {
      “use strict”;
      return eval(arguments[0]);
    };
  }
};
```

Applying the magic

```
return Reflect.apply(f(scopeProxy), thisGlobal, [src]);
...
return function() {
  with (arguments[0]) {
    return function() {
      "use strict";
      return eval(arguments[0]);
    };
  }
};
```

Without scope safeguards

```
return f(scopeProxy).call(thisGlobal, src);
…
function f(scopeProxy) {
  with (scopeProxy) {
    return function(src) {
      “use strict”;
      return eval(src);
    };
  }
};
```

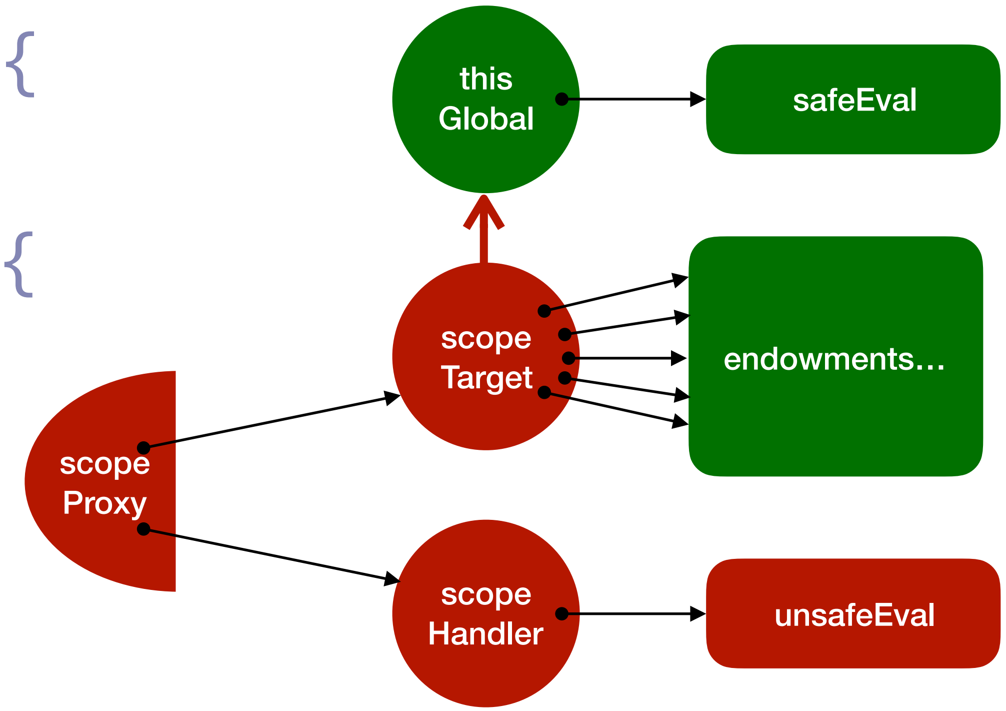
Without scope safeguards

```
return f(scopeProxy).call(thisGlobal, src);
…
function f(scopeProxy) {
  with (scopeProxy) {
    return function(src) {
      "use strict";
      return eval(src);
    };
  }
};
```

See slide
"meta-programming"

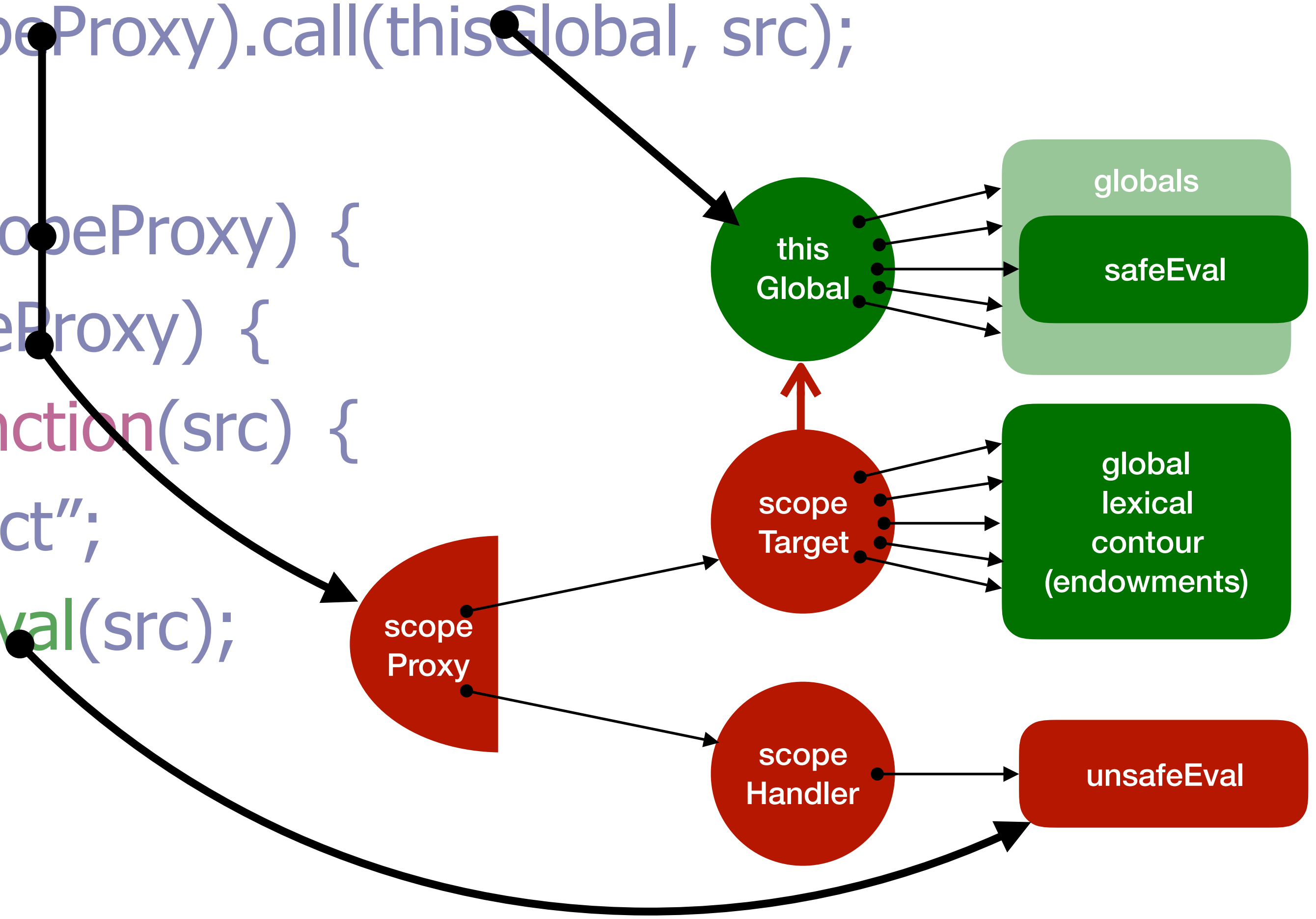
Without scope safeguards

```
return f(scopeProxy).call(thisGlobal, src);
…
function f(scopeProxy) {
  with (scopeProxy) {
    return function(src) {
      "use strict";
      return eval(src);
    };
  }
};
```



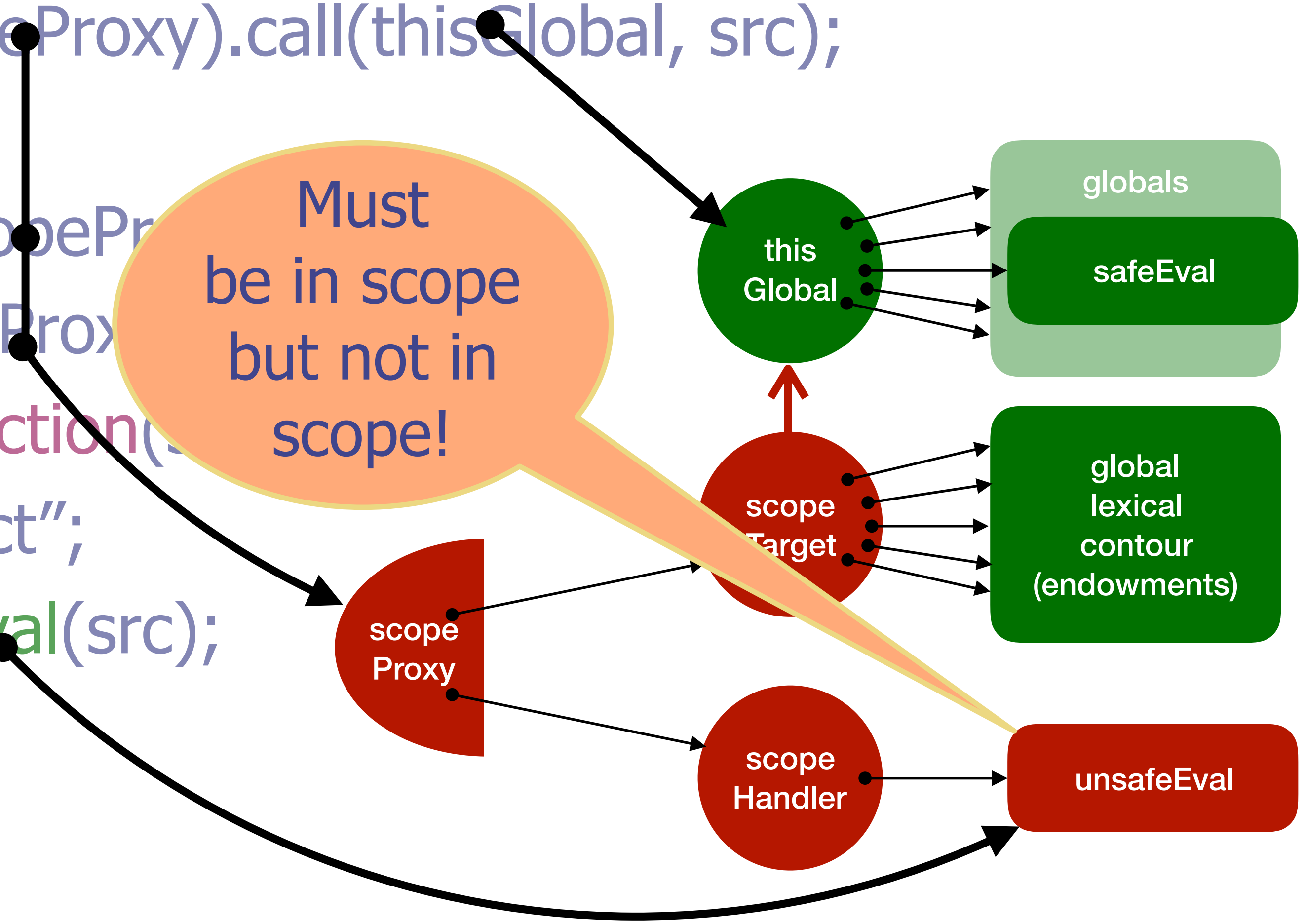
The heart of the shim

```
return f(scopeProxy).call(thisGlobal, src);
…
function f(scopeProxy) {
  with (scopeProxy) {
    return function(src) {
      "use strict";
      return eval(src);
    };
  }
};
```



The heart of the shim

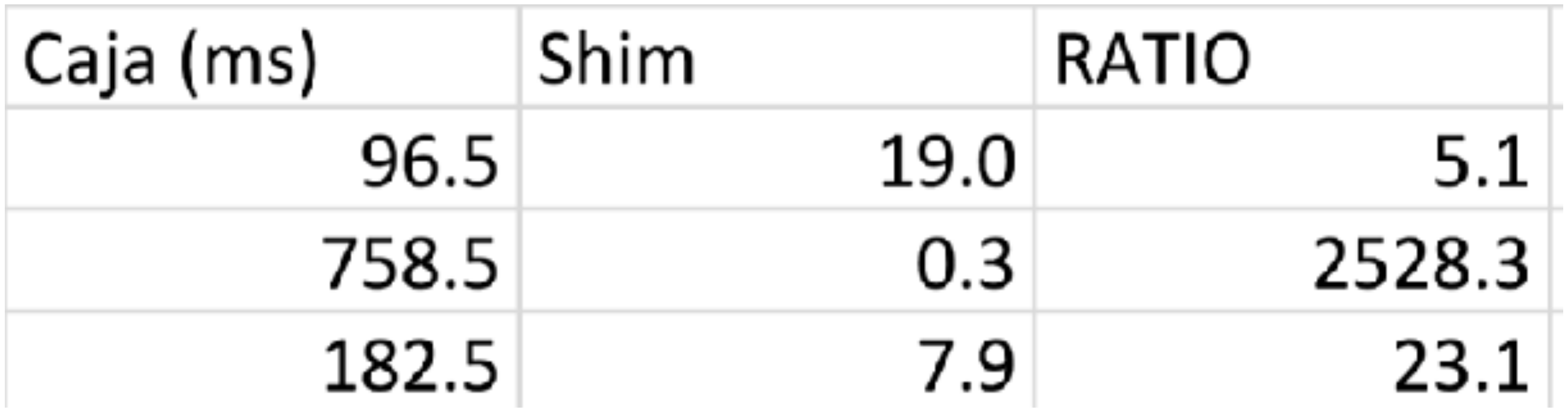
```
return f(scopeProxy).call(thisGlobal, src);
…
function f(scopePr
  with (scopeProx
    return function(
      “use strict”;
      return eval(src);
    };
  }
};
```



Fast (but need more measurements)

Typical case, hardly there

Worst cases, tremendously better



Caja (ms)	Shim	RATIO
96.5	19.0	5.1
758.5	0.3	2528.3
182.5	7.9	23.1

Security review

Security review

Whether ready for production use now?

Yes, but…

Findings

- Green vs. Red(unsafe) one bit type system
- Identity discontinuity much worse than expected
- Realm vs. Root Realm vs. Compartment
- No modules yet, but…
- Override mistake is **expensive**
- Secure meta-programming is **free**
- Shim is secure & useful **now**, but…
- No waterfall between spec and shim

Findings

Challenge

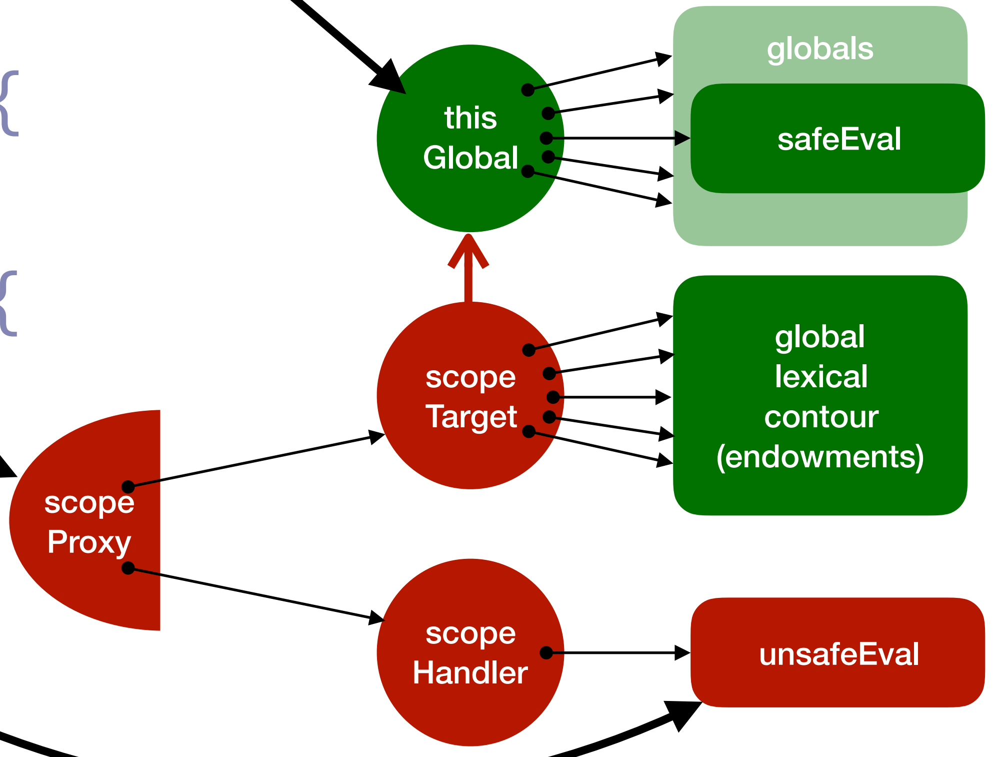
https://rawgit.com/Agoric/SES/master/demo/?dateNow=enabled

Need

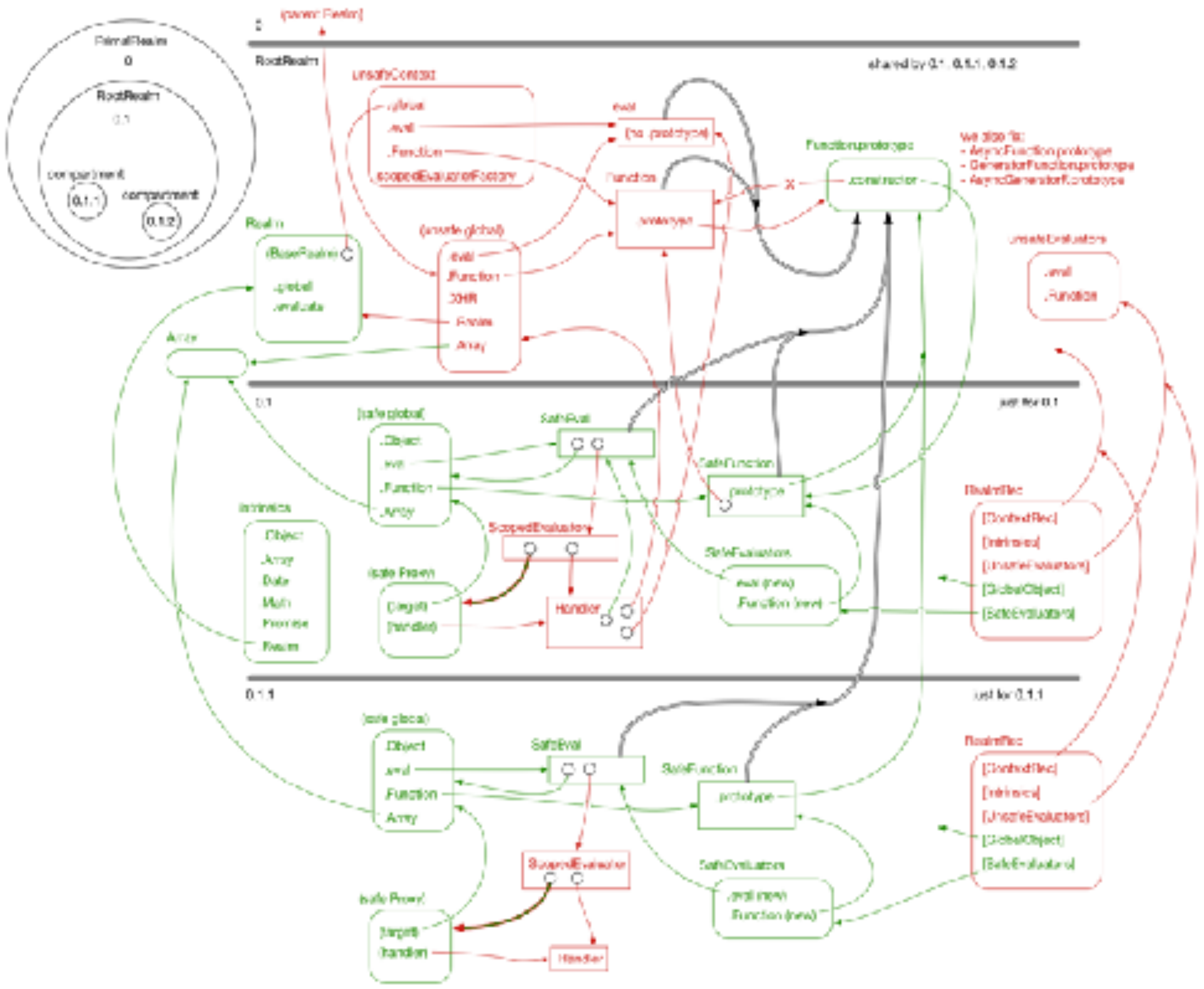
- bounties, both for bugs and proofs
- responsible disclosure process
- more feedback
- more experience

Green vs. Red(unsafe) one bit type system

```
return f(scopeProxy).call(thisGlobal, src);
...
function f(scopeProxy) {
  with (scopeProxy) {
    return function(src) {
      "use strict";
      return eval(src);
    };
  }
};
```



Green vs. Red(unsafe) one bit type system



Green vs. Red(unsafe) one bit type system

Wrote transitive deep-freeze-like graph walker to verify separation

Green vs. Red(unsafe) one bit type system

Wrote transitive deep-freeze-like graph walker to verify separation

Found no leakage that we didn’t already suspect

Identity discontinuity: Much worse than expected

From spec’s own README:

```
class FakeWindow extends Realm {
  init() {
    super.init();  // install the standard primordials
    let global = this.global;
    global.document = new FakeDocument(...);
    global.alert = new Proxy(fakeAlert, { ... });
    ...
  }
}
```

Identity discontinuity: Much worse than expected

From spec's own README:

```
class FakeWindow extends Realm {
  init() {
    super.init();  // install the standard primordials
    let global = this.global;
    global.document = new FakeDocument(...);
    global.alert = new Proxy(fakeAlert, { ... });
    ...
  }
}
```

Leaks references across realms!

Identity discontinuity: Much worse than expected

Old API: customize from outside

- Manipulate intrinsics
- Call to `.init()` subclass override
- Most frequent loss of confinement, by everyone!

Shims customize from inside

- Use options/handler object
- Accepts existing shims
- Still need external endowments (whitelist?)

Realm vs. Root Realm vs. Compartment

Smorgasbord of micro-choices didn’t work

`Realm` constructor per Root Realm

- statics work from inside

`Realm` instance per Realm

- methods work from outside
- Only Root Realms create Realms

No modules yet, but…

Script code, Eval code, Module code

Module loaders are hard

- Good ideas in separate spec (Dave Herman, Caridy Patino)
- Shim modules correctly without parsing? Unlikely

…but, build tools (rollup) good enough for now

- Node CommonJS modules are scripts
- Google Caja
- Salesforce Locker Service
- Agoric Dr. SES

Override mistake is **expensive**

```
Object.freeze(Object.prototype);
…



function Point(x, y) {
  this.x = x; this.y = y;
}
Point.prototype.toString = function() {   // throws
    return `<${this.x},${this.y}>`;
};
```

Override mistake is **expensive**

```
const original = Object.prototype.toString;
defineProperty(Object.prototype, ‘toString’, {
~~  get() { return original; },~~  // slow
  set(override) { … },
  …
});
…
Point.prototype.toString = function() {   // ok
    return `<${this.x},${this.y}>`;
};
```

Secure meta-programming

```
push.call(thisValue, arg0, arg1)
```

Secure meta-programming

push**.call**(thisValue, arg0, arg1)

Failure
to emulate spec

Function.prototype.call = () => { throw …; };

// Prevented on Frozen Realms, SES

// But threatens (non-Frozen) shim accuracy

Secure meta-programming is **free**

```
push.call(thisValue, arg0, arg1)

Reflect.construct(push, thisValue, [arg0, arg1])

// …early…
const pushFn = uncurryThis(Array.prototype.push)
// …later…
pushFn(thisValue, arg0, arg1)

// Faster on most. Slightly slower on others.
```

Secure meta-programming still tricky

```
// …early…
const g22r = Object.getOwnPrototypeDescriptor;
// …later…
const desc = g22r(…, …);
…
if (‘value’ in desc) {        vs        if (!(‘get’ in desc)) {
```

Secure meta-programming still tricky

```
// …early…
const g22r = Object.getOwnPrototypeDescriptor;
// …later…
const desc = g22r(…, …);
…
if ('value' in desc) {        vs        if (!('get' in desc)) {
Object.prototype.value = null;

// Again, prevented on Frozen Realms, SES
// But threatens (non-Frozen) shim accuracy
```

Shim is secure & useful **now**, but…

Small, Solid, Fast

- Lessons from Google Caja, Salesforce Locker Service
- < 700 lines, hardcore review, 90% test coverage
- …but known issues being fixed

No trusted parser

- Rewrite only needed for full compat, not safety
- …but for `import(…)` expression. Conservative regexp
- …but virtualized direct eval impossible

…but must know all undeniables, evaluators

- Only platform support enables safe growth

No waterfall between spec and shim

Iterative development, Complementary lessons

- All activity informs all other activity
- Discovered what customizations work, and don’t

Working to reconcile, as conformant subset

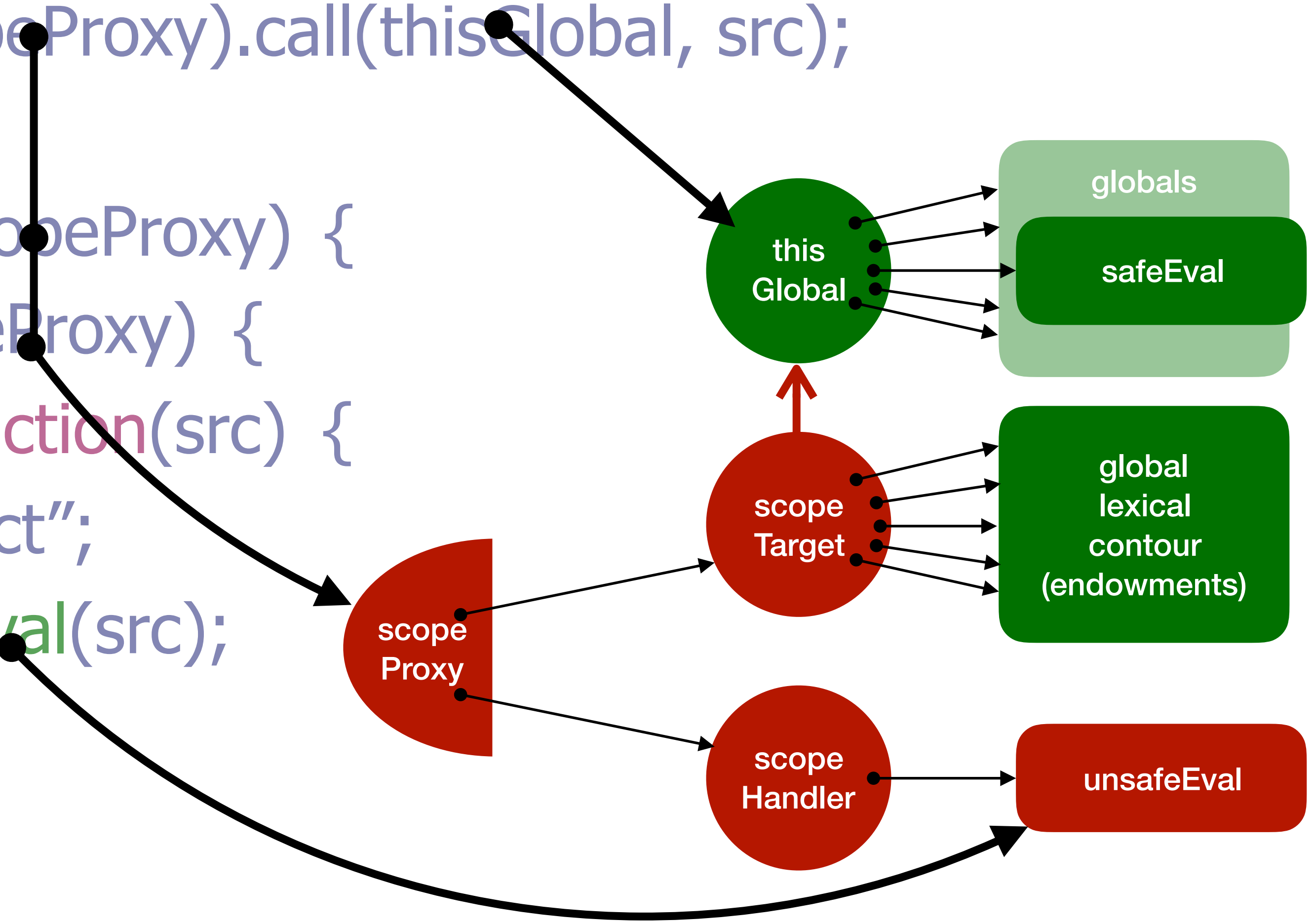
Try the challenge

https://rawgit.com/Agoric/SES/master/demo/?dateNow=enabled

Please report all bugs, responsibly. Thanks.

Questions?

```
return f(scopeProxy).call(thisGlobal, src);
…
function f(scopeProxy) {
  with (scopeProxy) {
    return function(src) {
      "use strict";
      return eval(src);
    };
  }
};
```


globals
this Global
safeEval
scope Target
global lexical contour (endowments)
scope Proxy
scope Handler
unsafeEval