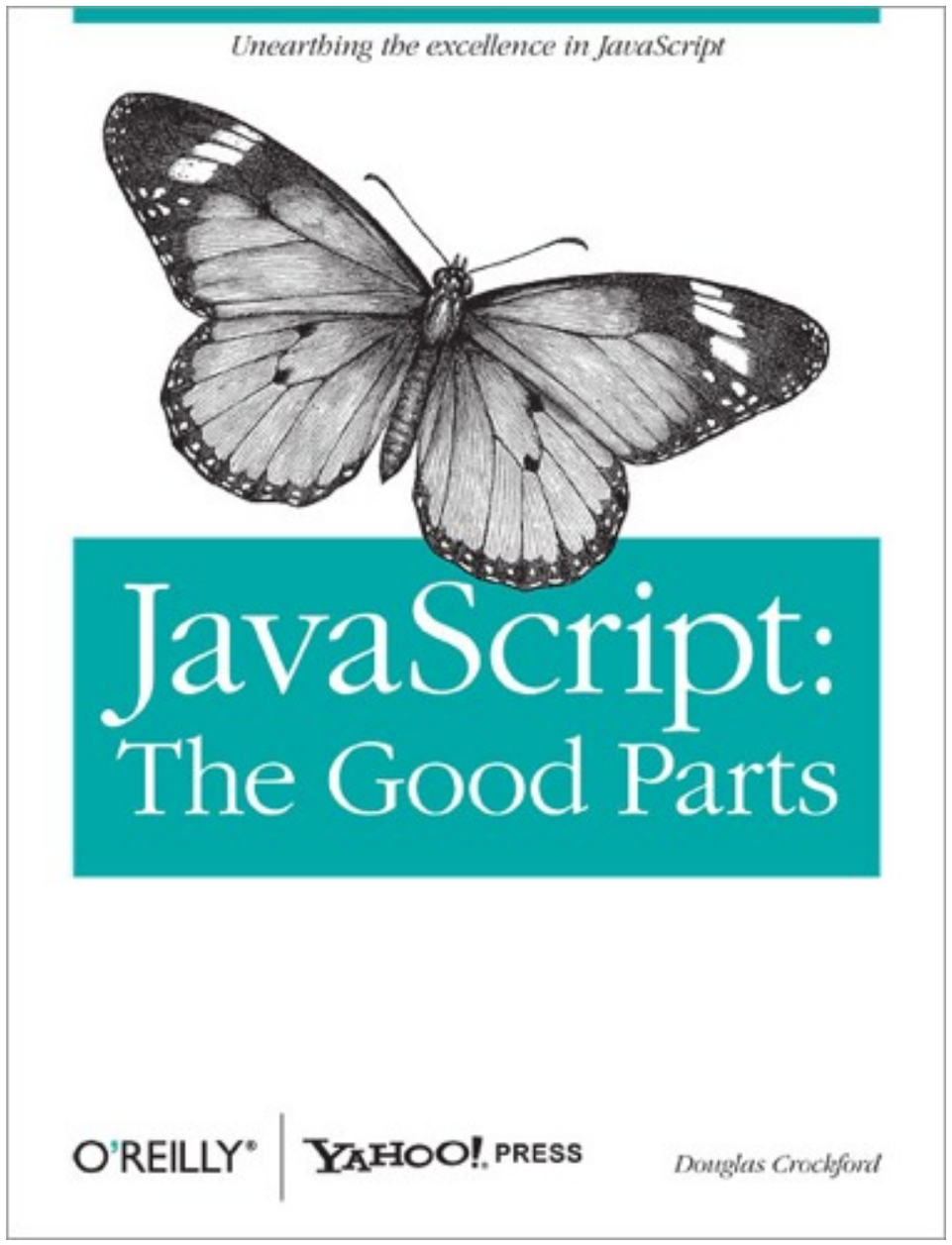
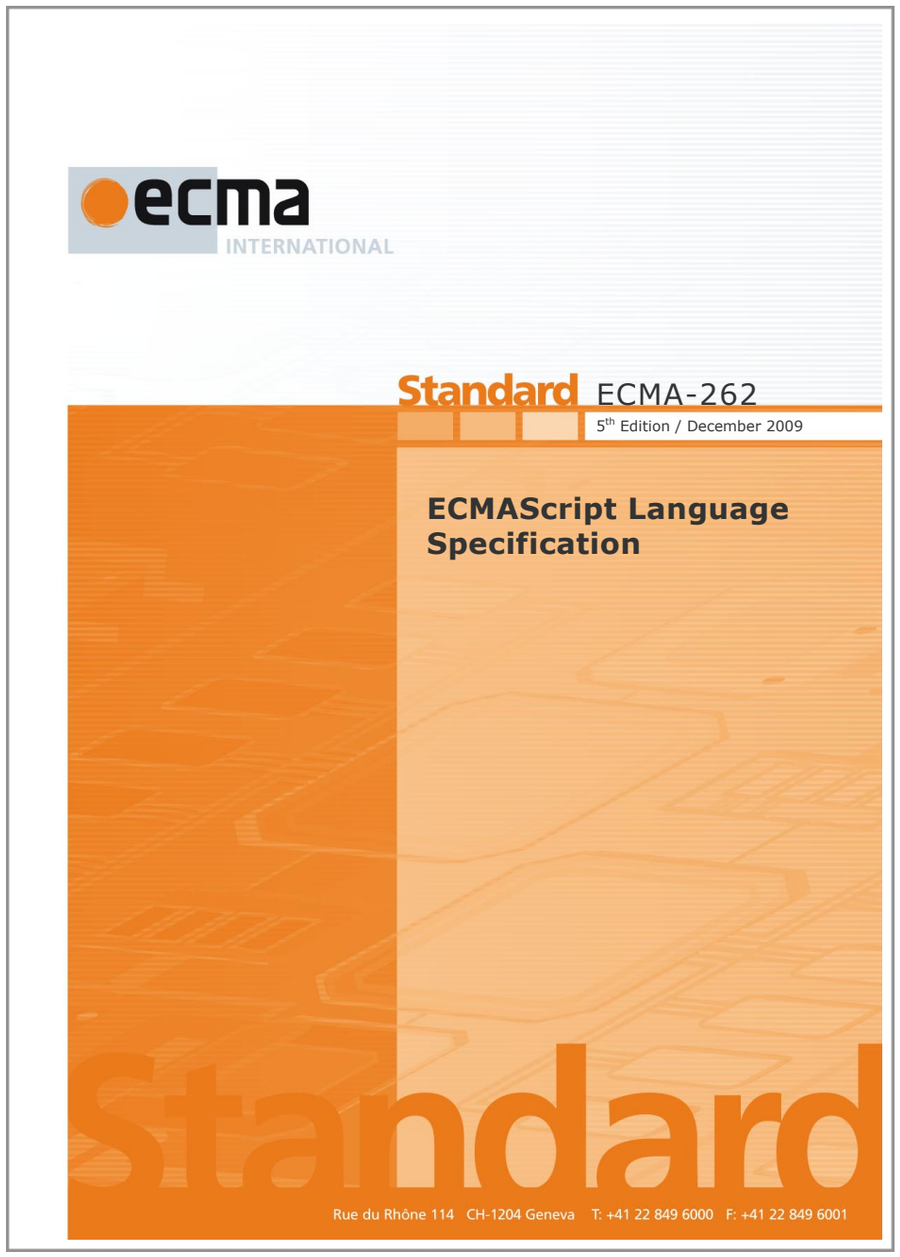
Trustworthy Proxies
Virtualizing Objects with Invariants

Tom Van Cutsem (VUB) and Mark S. Miller (Google)

Context


Unearthing the excellence in JavaScript
JavaScript:
The Good Parts
O'REILLY
YAHOO! PRESS
Douglas Crockford


ecma
INTERNATIONAL
Standard ECMA-262
5th Edition / December 2009
ECMAScript Language Specification
standard
Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

Preliminaries

- Proxies in JavaScript
- Invariants in JavaScript

Preliminaries

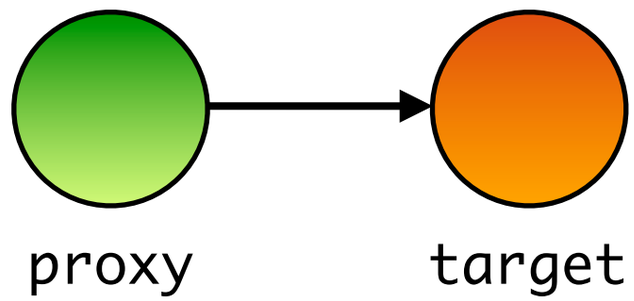
- **Proxies in JavaScript**
- Invariants in JavaScript

Proxies

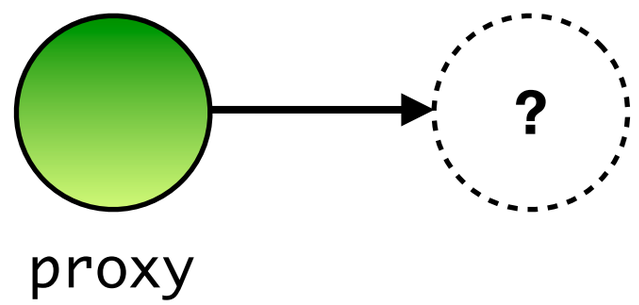
- Language mechanism to wrap objects independent of their type
- Can intercept operations applied to the proxy object
- Part of a new reflection API for ECMAScript 6
- Similar:
  - `java.lang.reflect.Proxy` in Java
  - Chaperones and Impersonators in Racket

Proxies: use cases

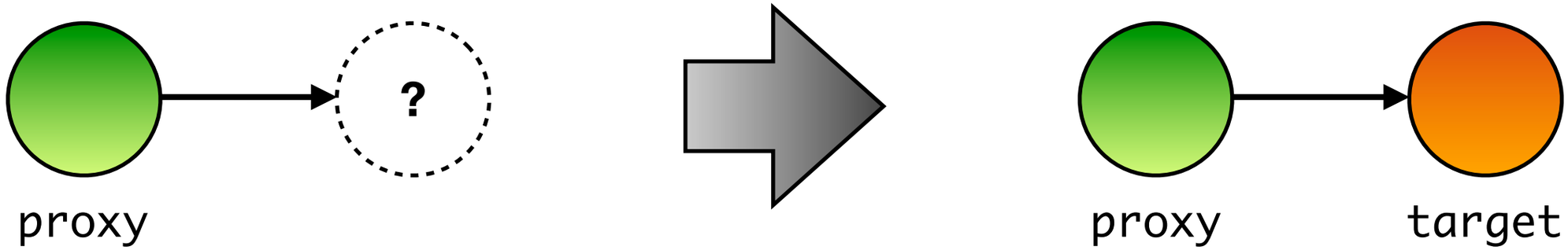
- Generic wrappers around existing objects: access control wrappers, tracing, profiling, contracts, taint tracking, ...



- Virtual objects: remote objects, mock objects, persistent objects, futures, lazily initialized objects, ...

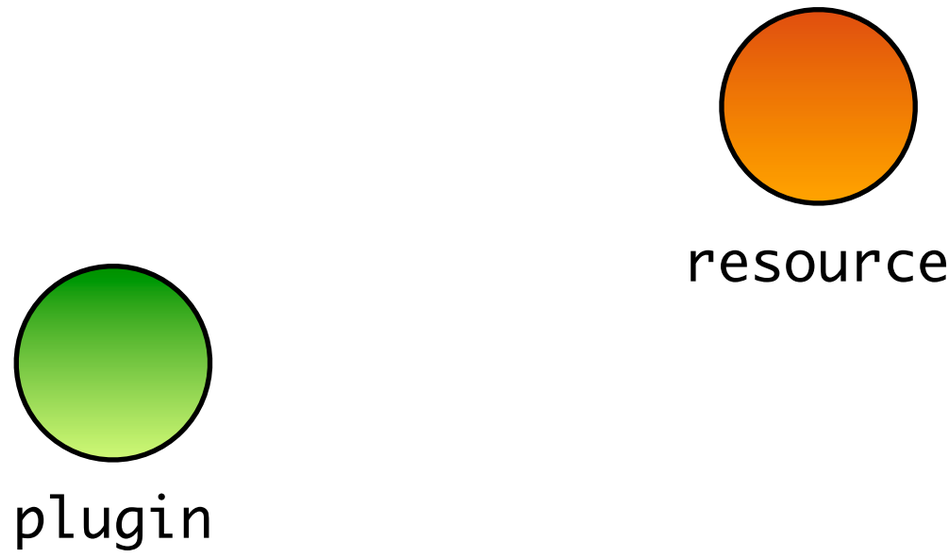


The rest of this talk



Proxy example: revocable references

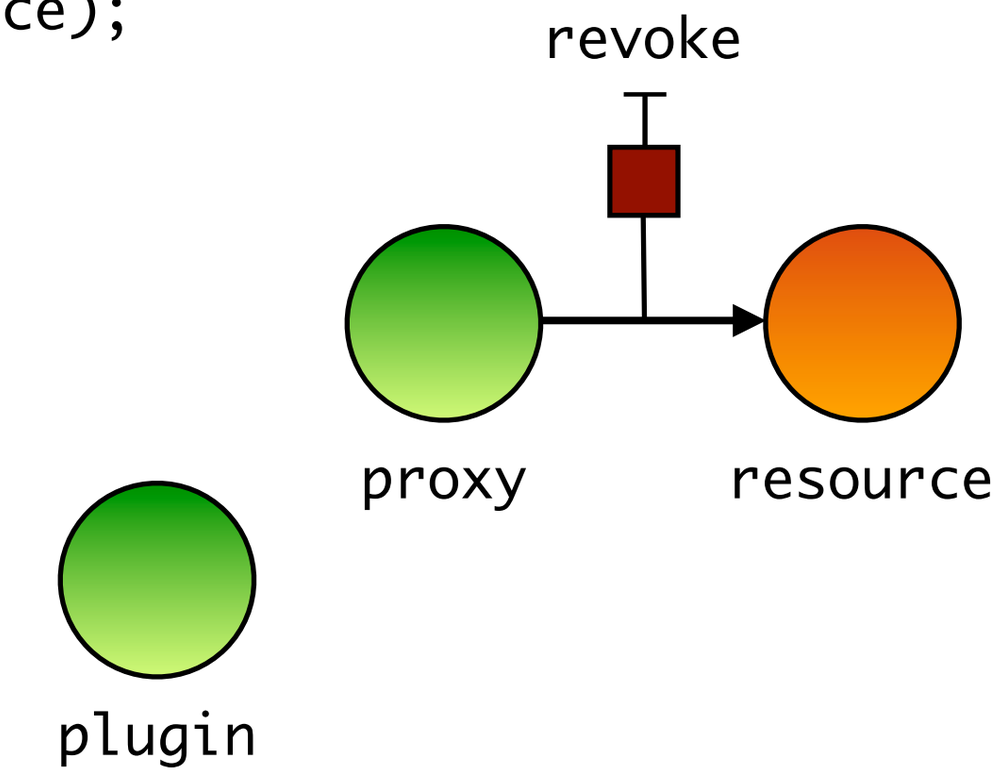
- Provide temporary access to a resource
- Useful for explicit memory management or expressing security policy



Proxy example: revocable references

- Provide temporary access to a resource
- Useful for explicit memory management or expressing security policy

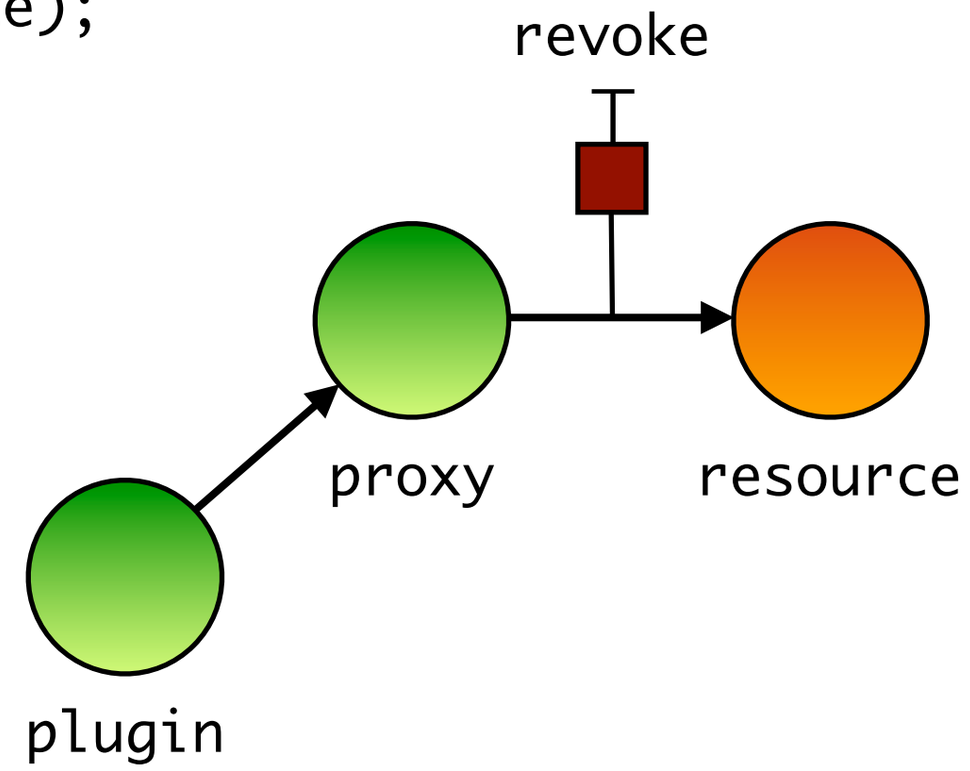
```
var {proxy,revoke} = makeRevocable(resource);
```



Proxy example: revocable references

- Provide temporary access to a resource
- Useful for explicit memory management or expressing security policy

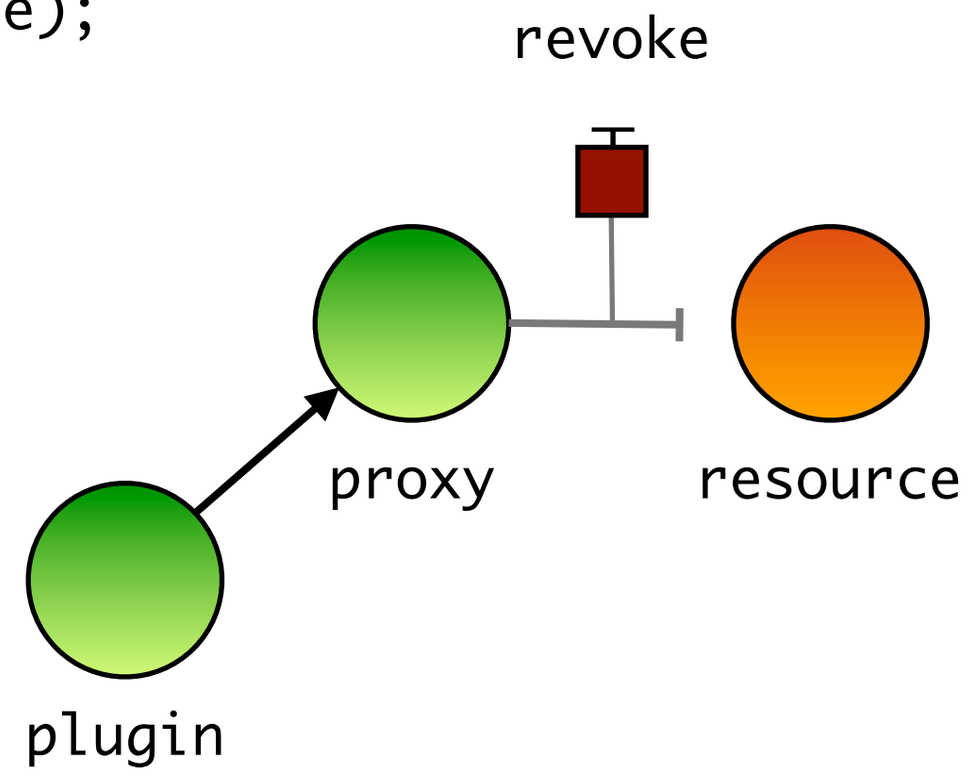
```
var {proxy,revoke} = makeRevocable(resource);

plugin.give(proxy)
```



Proxy example: revocable references

- Provide temporary access to a resource
- Useful for explicit memory management or expressing security policy

```
var {proxy,revoke} = makeRevocable(resource);


plugin.give(proxy)

...
revoke();
```



Revocable references

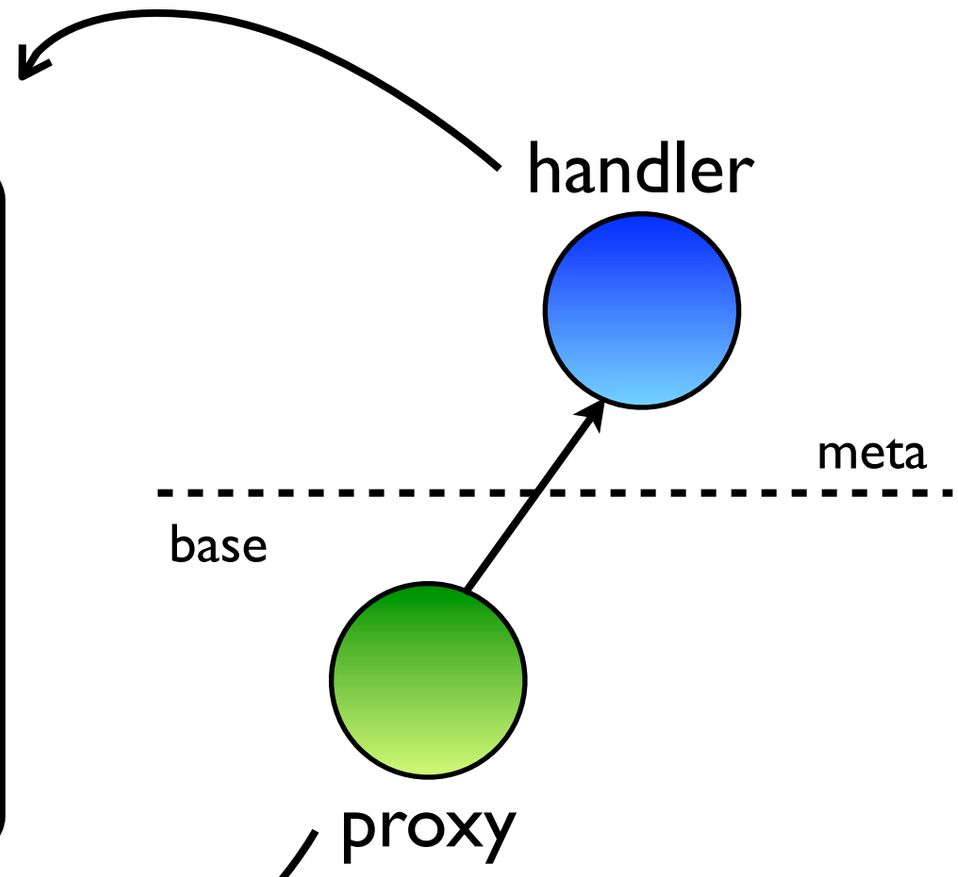
```
function makeRevocable(target) {
  var enabled = true;
  var proxy = Proxy({
  });
  return {
     proxy: proxy,
    revoke: function() { enabled = false; }
  }
}
```

Revocable references

```
function makeRevocable(target) {
  var enabled = true;
  var proxy = Proxy({
    get: function(name) {
      if (!enabled) throw Error(“revoked”)
      return target[name];
    },
    set: function(name, val) {
      if (!enabled) throw Error(“revoked”)
      target[name] = val;
    },
    ...
  });
  return {
     proxy: proxy,
    revoke: function() { enabled = false; }
  }
}
```

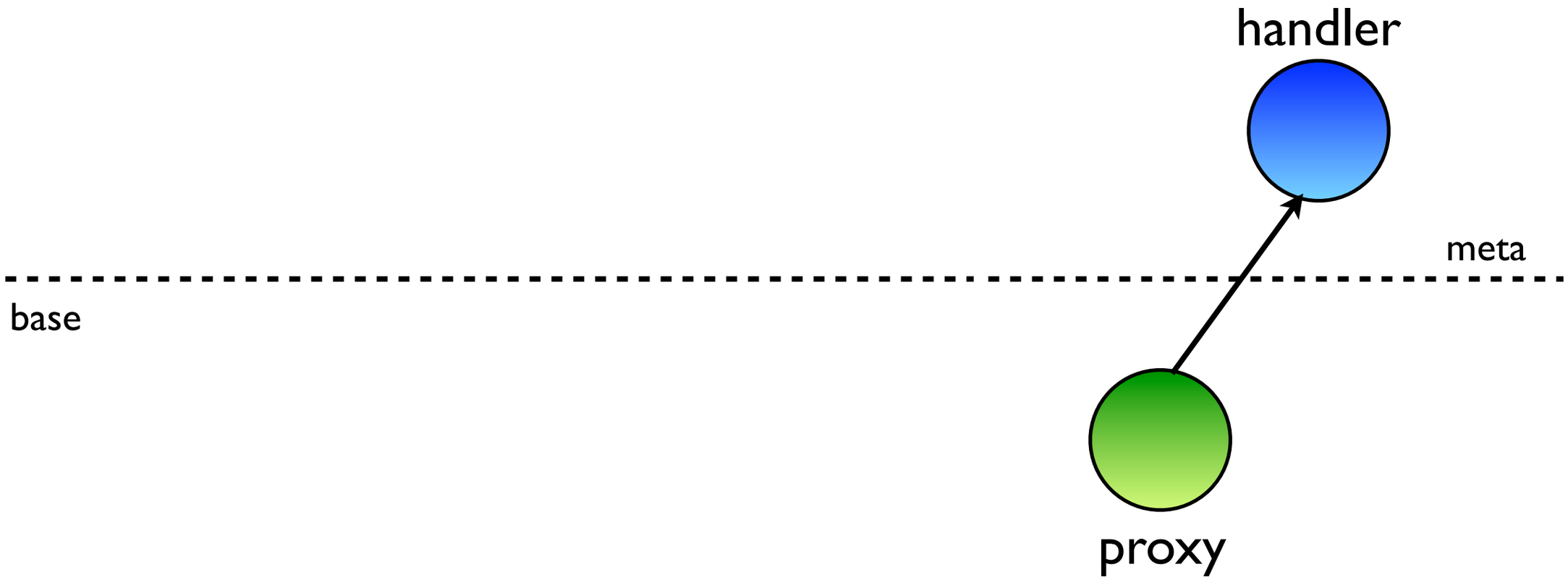
Revocable references

```
function makeRevocable(target) {
  var enabled = true;
  var proxy = Proxy({
    get: function(name) {
      if (!enabled) throw Error("revoked")
      return target[name];
    },
    set: function(name, val) {
      if (!enabled) throw Error("revoked")
      target[name] = val;
    },
    ...
  });
  return {
     proxy: proxy,
    revoke: function() { enabled = false; }
  }
}
```



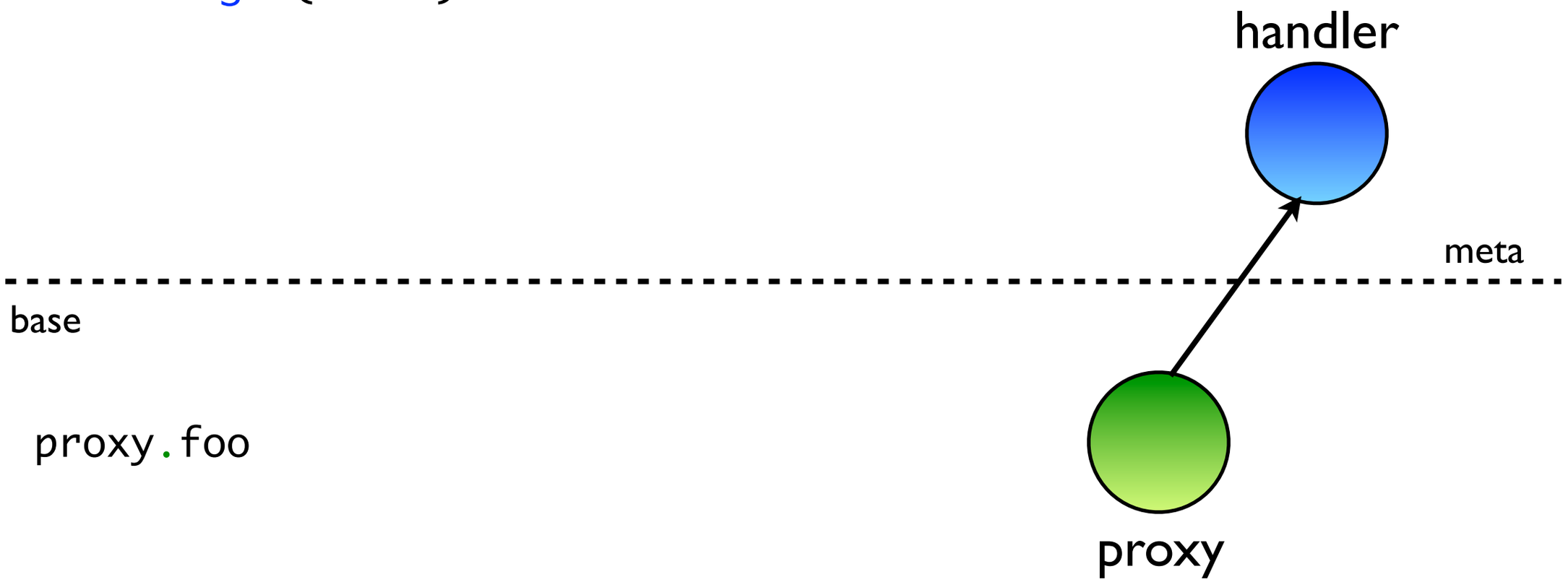
Proxy API

```
var proxy = Proxy(handler);
```



Proxy API

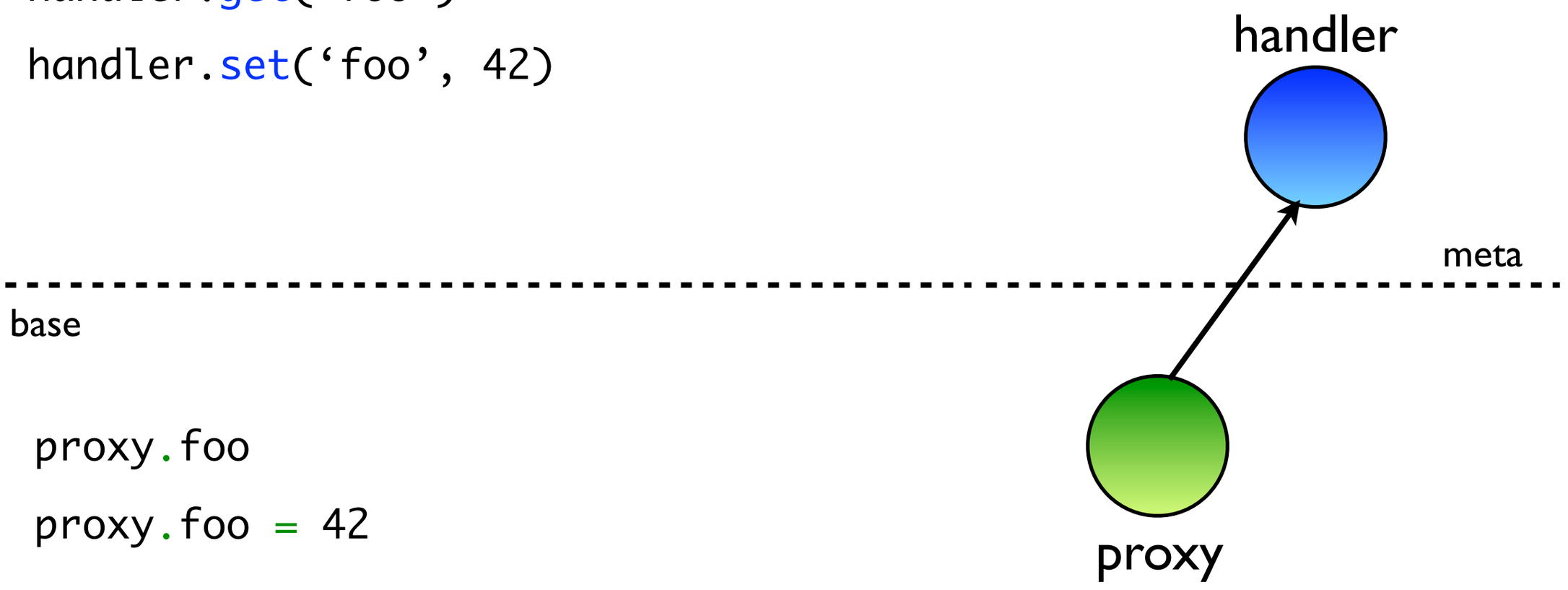
```
var proxy = Proxy(handler);

handler.get('foo')
```



Proxy API

```
var proxy = Proxy(handler);
```

```
handler.get('foo')
handler.set('foo', 42)
```




```
proxy.foo
proxy.foo = 42
```

Preliminaries

- Proxies in JavaScript
- **Invariants in JavaScript**

Frozen objects (since ECMAScript 5)

```
var point = { x: 0, y: 0 };

Object.freeze(point);

point.z = 0;    // error: can’t add new properties

delete point.x; // error: can’t delete properties

point.x = 7;    // error: can’t assign properties

Object.isFrozen(point) // true
```

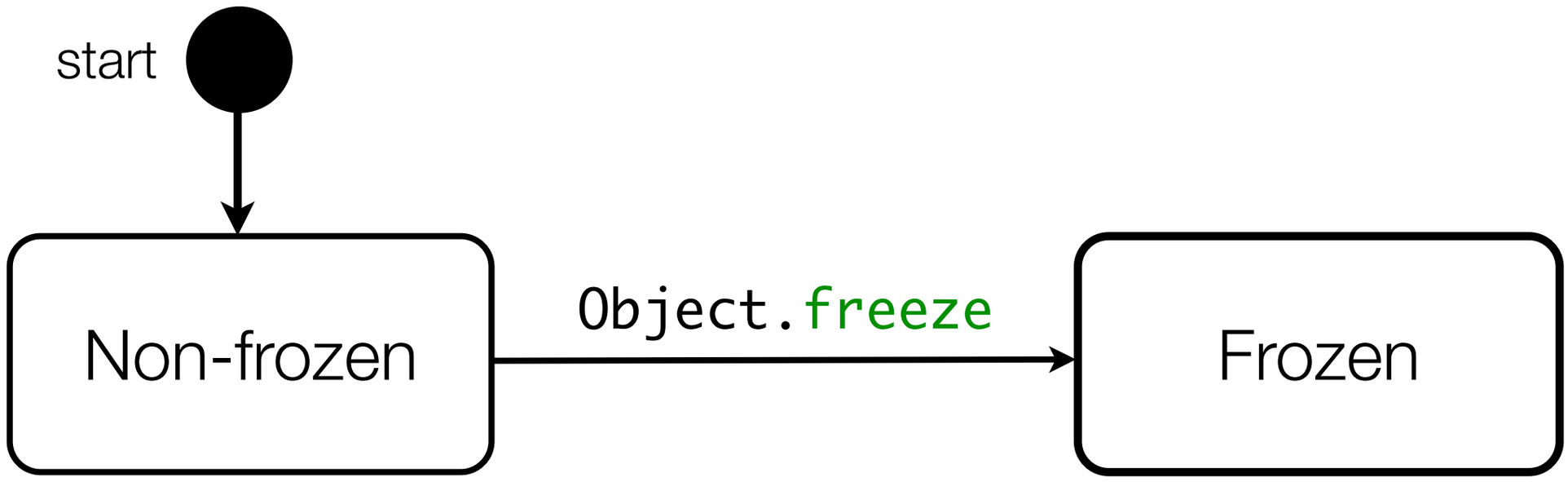
Frozen objects (since ECMAScript 5)

```
var point = { x: 0, y: 0 };

Object.freeze(point);

point.z = 0;    // error: can't add new properties
delete point.x; // error: can't delete properties

point.x = 7;    // error: can't assign properties

Object.isFrozen(point) // true
```

guarantee (**invariant**):
properties of a frozen object are immutable

Frozen objects (since ECMAScript 5)

freezing is permanent (**monotonic**) - there is no `defrost`



Preliminaries

- Proxies in JavaScript
- Invariants in JavaScript

How do they combine?

How to combine proxies with frozen objects?

- Can a proxy emulate the “frozen” invariant of the object it wraps?

```
var point = { x: 0, y: 0 };
Object.freeze(point);
var {proxy,revoke} = makeRevocable(point);

Object.isFrozen(point) // true
Object.isFrozen(proxy) // ?
```

How to combine proxies with frozen objects?

- Can a proxy emulate the “frozen” invariant of the object it wraps?

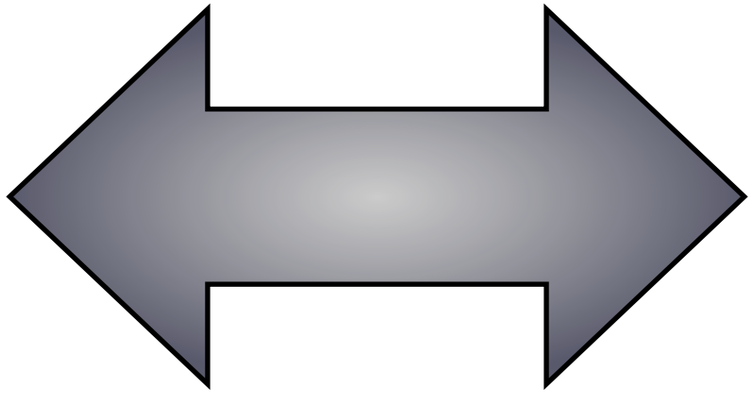
```
function wrap(target) {
  return Proxy({
    get: function(name) { return Math.random(); }
  });
}

var point = { x: 0, y: 0 };
Object.freeze(point);

var proxy = wrap(point);

Object.isFrozen(point) // true
Object.isFrozen(proxy) // can’t be true!
```

The “solution”

- Proxies can’t emulate frozen objects
- `Object.isFrozen(proxy)` always returns `false`
- Safe, but overly restrictive

Tradeoff

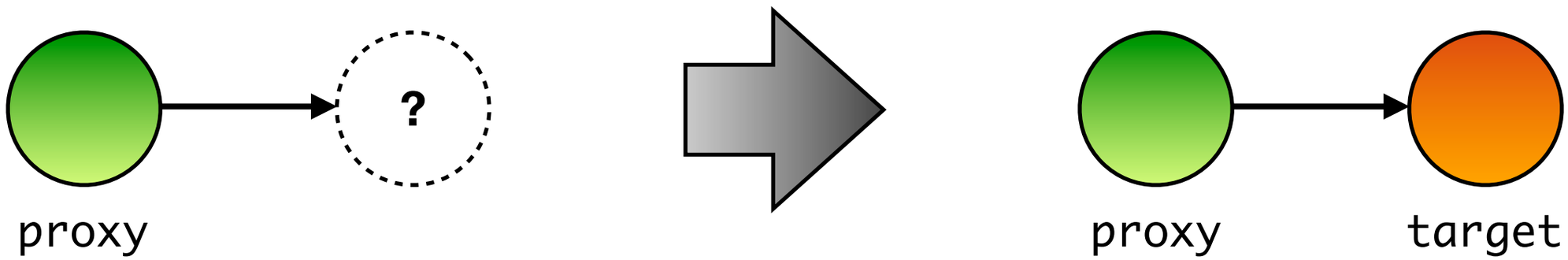
Powerful proxies that can virtualize frozen objects



Strong language invariants that can’t be spoofed

Second iteration: direct proxies

- Proxy now has *direct* pointer to the object it wraps
- `Proxy(target, handler)`
- `Object.isFrozen(proxy)` <=> `Object.isFrozen(target)`



Revocable references (original API)

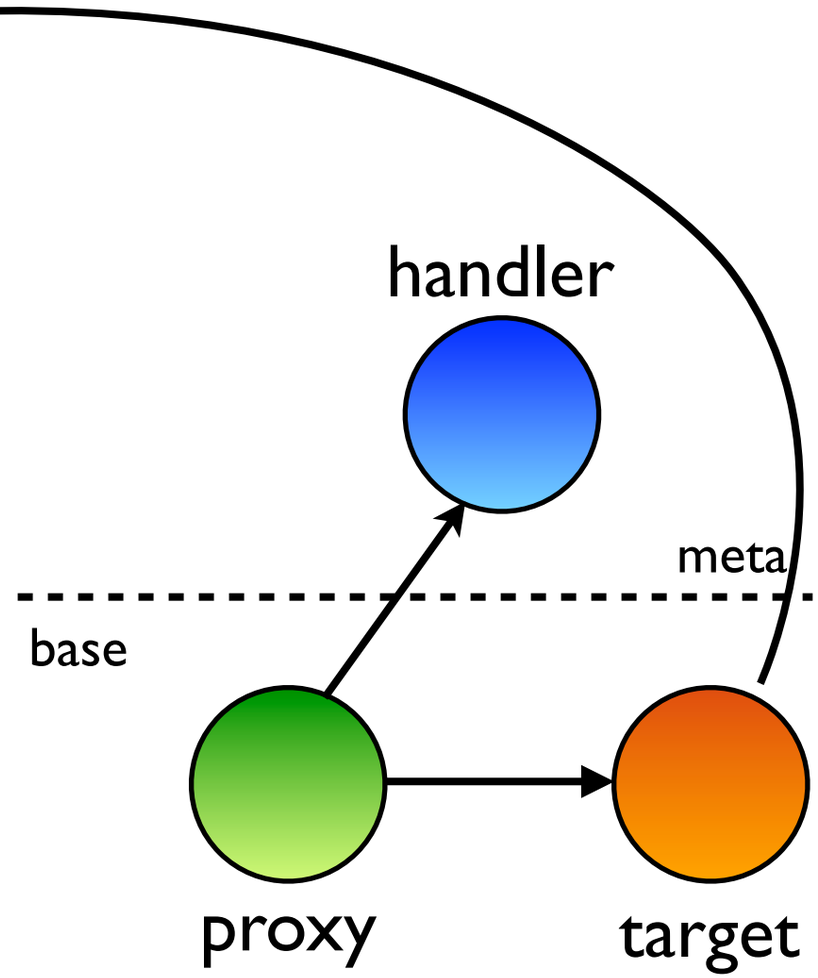
```
function makeRevocable(target) {
  var enabled = true;
  var proxy = Proxy({
    get: function(name) {
      if (!enabled) throw Error(“revoked”)
      return target[name];
    },
    set: function(name, val) {
      if (!enabled) throw Error(“revoked”)
      target[name] = val;
    },
    ...
  });
  return {
     proxy: proxy,
    revoke: function() { enabled = false; }
  }
}
```

Revocable references (new API)

```
function makeRevocable(target) {
  var enabled = true;
  var proxy = Proxy(target, {
    get: function(tgt, name) {
      if (!enabled) throw Error(“revoked”)
      return target[name];
    },
    set: function(tgt, name, val) {
      if (!enabled) throw Error(“revoked”)
      target[name] = val;
    },
    ...
  });
  return {
     proxy: proxy,
    revoke: function() { enabled = false; }
  }
}
```

Revocable references

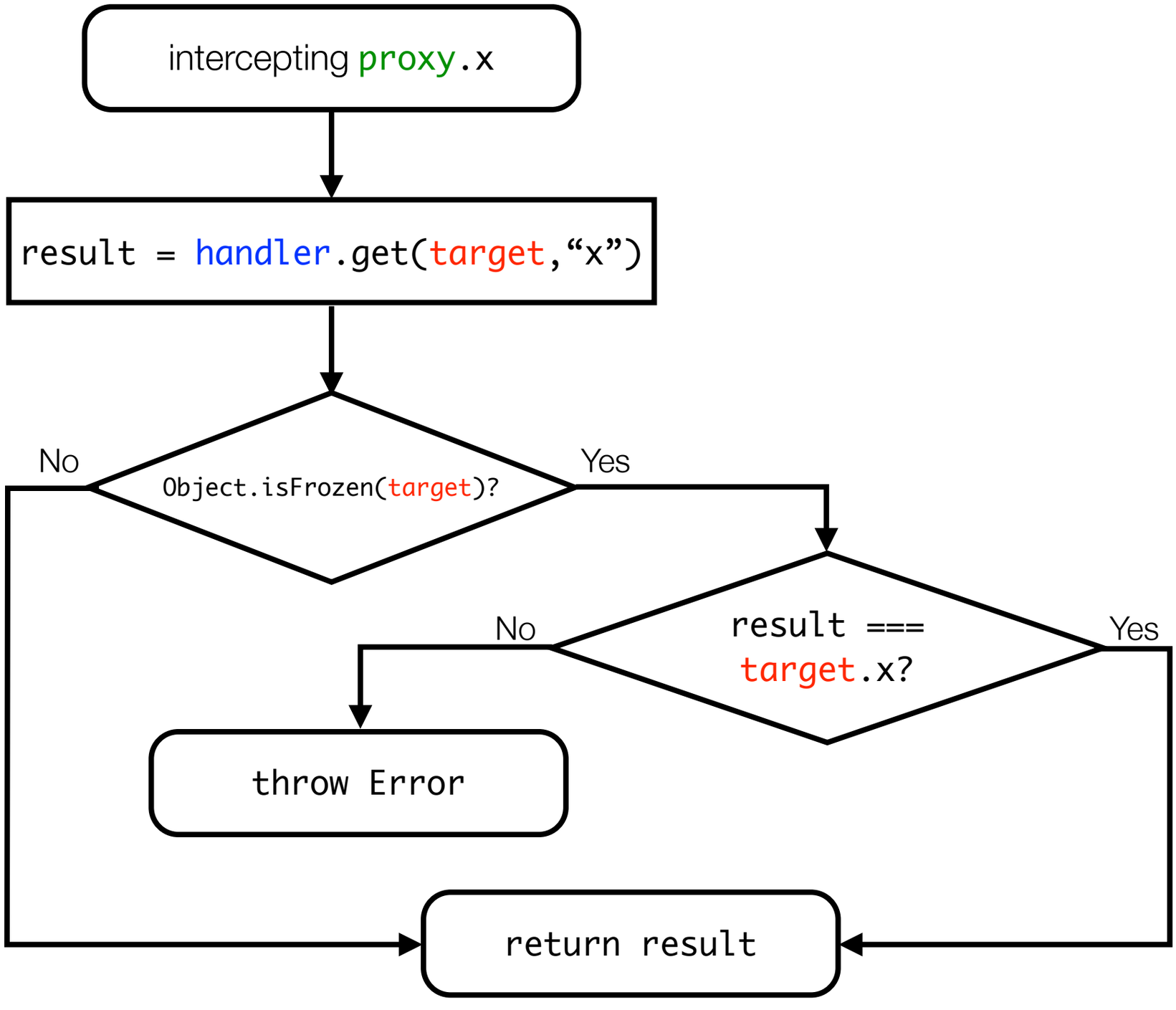
```
function makeRevocable(target) {
  var enabled = true;
  var proxy = Proxy(target, {
    get: function(tgt, name) {
      if (!enabled) throw Error("revoked")
      return target[name];
    },
    set: function(tgt, name, val) {
      if (!enabled) throw Error("revoked")
      target[name] = val;
    },
    ...
  });
  return {
     proxy: proxy,
    revoke: function() { enabled = false; }
  }
}
```

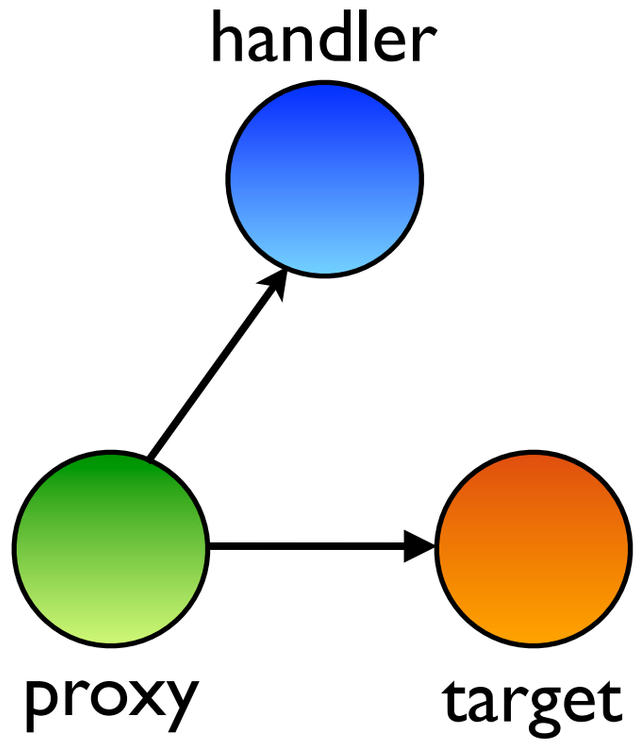


Direct proxies

```
var point = { x: 0, y: 0 };
Object.freeze(point);

var {proxy,revoke} = makeRevocable(point);

Object.isFrozen(point) // true
Object.isFrozen(proxy) // true!
```

Direct proxies enforce invariants via runtime assertions

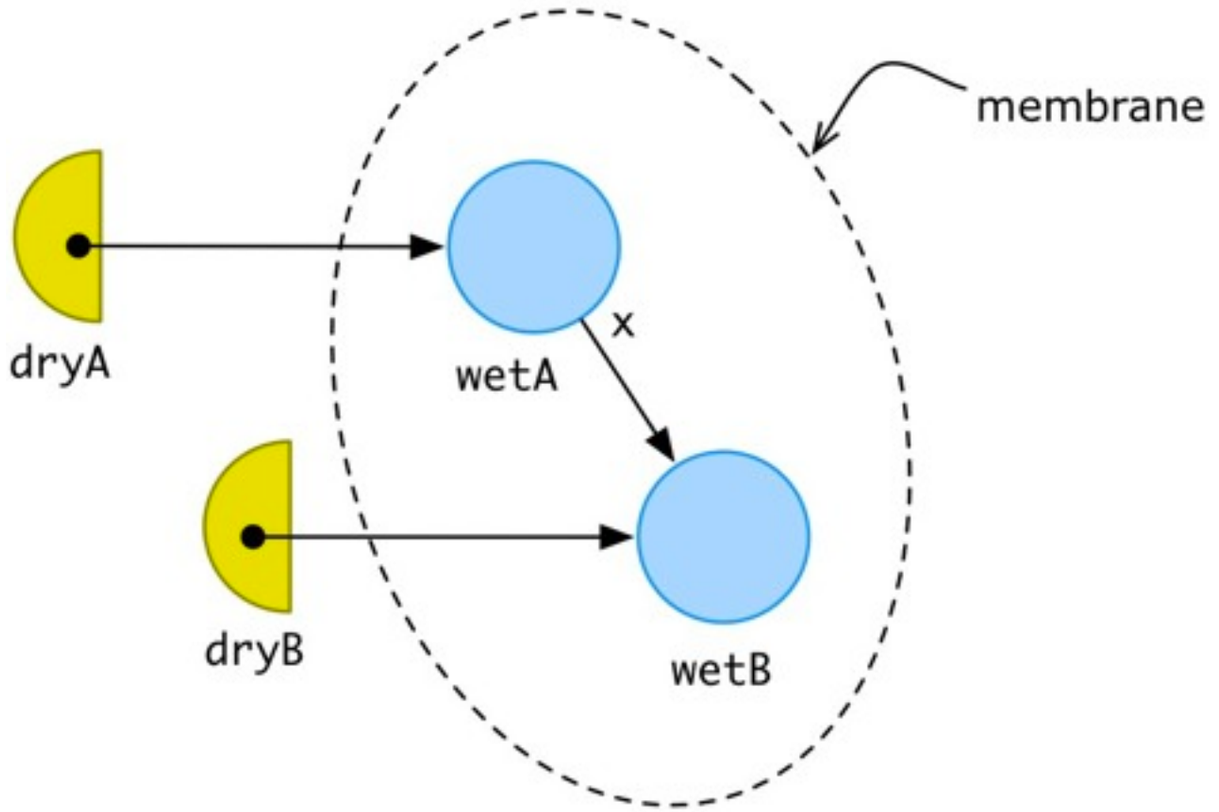




Case study: Membranes

- Goal: isolate two object graphs
- Litmus test for expressiveness of proxies (e.g. contracts require membranes)
- Must be transparent: maintain invariants on both sides of the membrane

```
var wetB = {};
var wetA = { x: wetB };

var dryA = wet2dry(wetA);
var dryB = dryA.x;
```

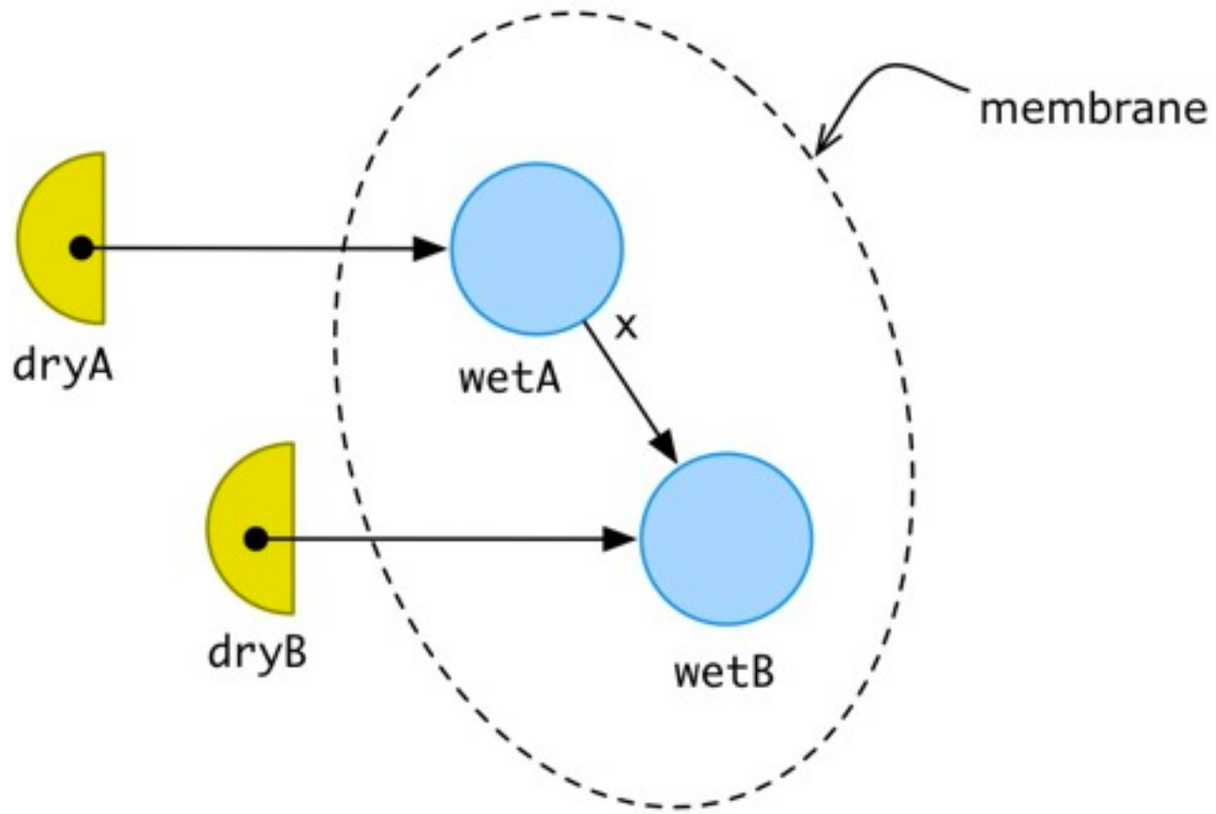


Membranes with direct proxies: try 1

- Works fine as long as `wetTarget` doesn't have any invariants (i.e. not frozen)

```
function wet2dry(wetTarget) {
  ...
  var dryProxy = Proxy(wetTarget, {
    ...
    get: function(wetTarget, name) {
      return wet2dry(wetTarget[name]);
    }
  });
  ...
}
```

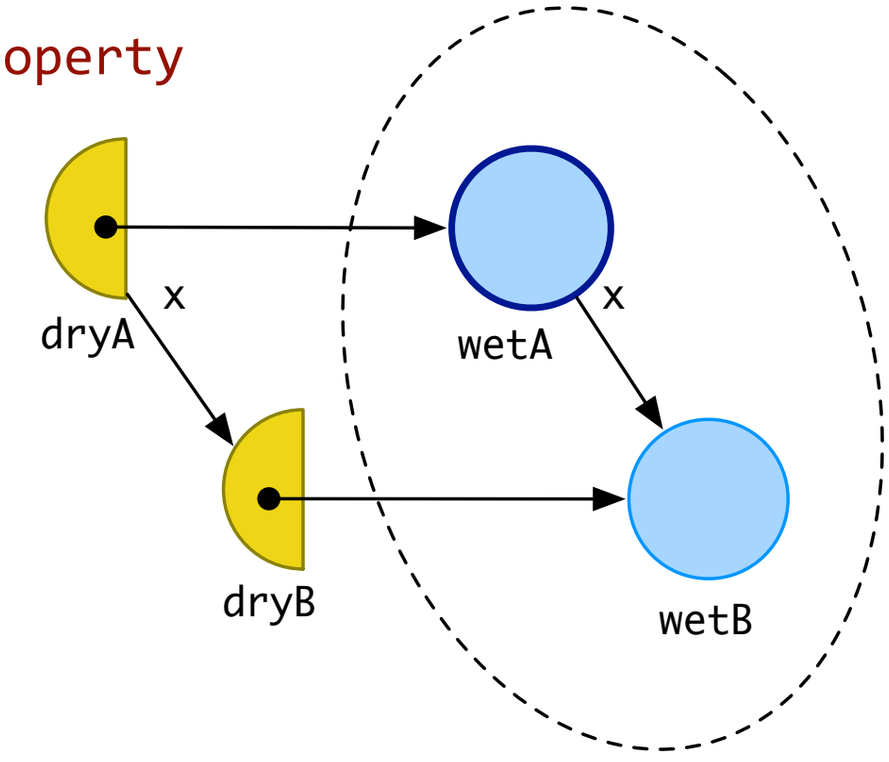
```
var wetB = {};
var wetA = { x: wetB };

var dryA = wet2dry(wetA);
var dryB = dryA.x;
```



Membranes with direct proxies: try 1

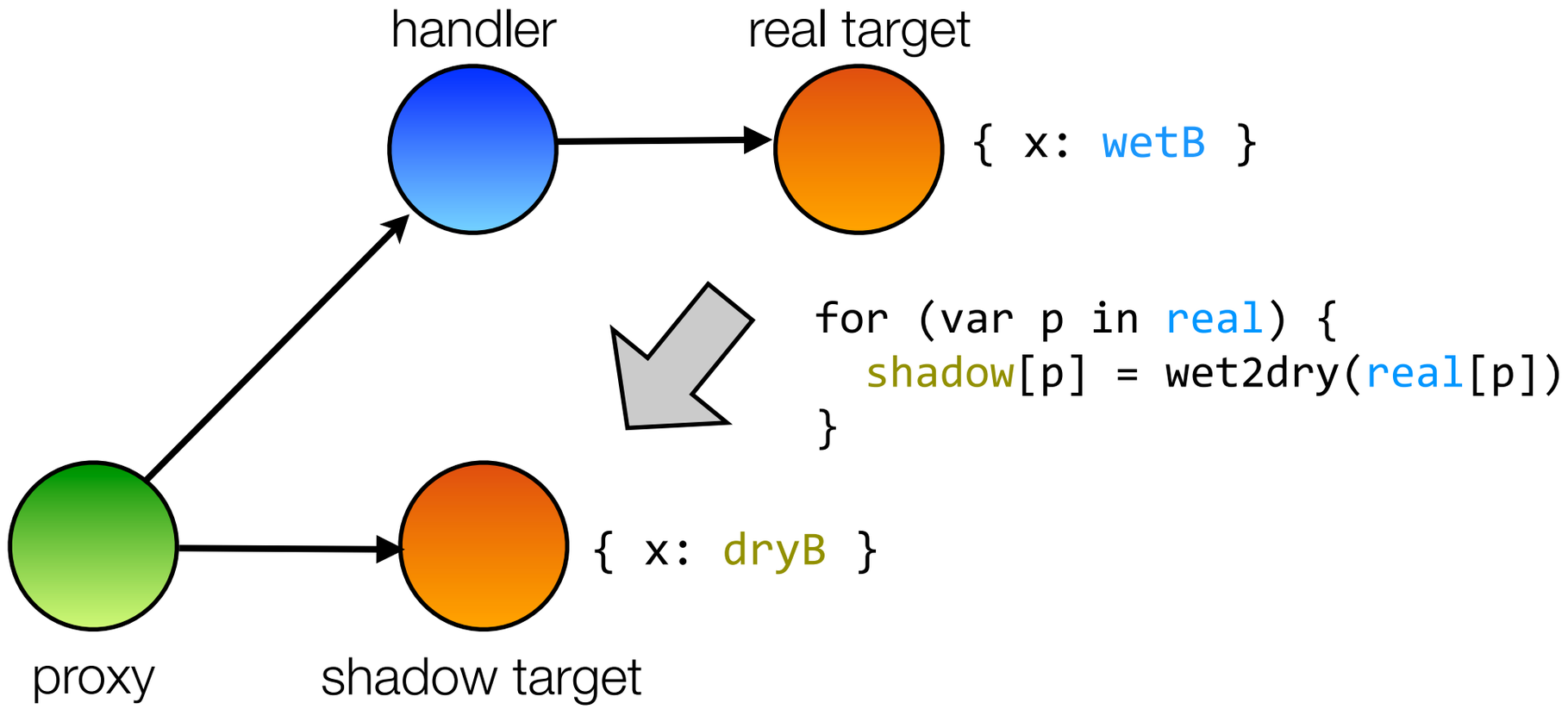
- Now assume `wetTarget` is frozen
- Because `wetA.x` is a frozen property, and `wetA.x === wetB`, the proxy asserts that `dryA.x === wetB`

```
var wetB = {};
var wetA = Object.freeze({ x: wetB });

var dryA = wet2dry(wetA);
dryA.x // Error: inconsistent value for frozen property
```



Membranes with direct proxies: try 2

- Use a shadow target: a dummy target object to store wrapped properties

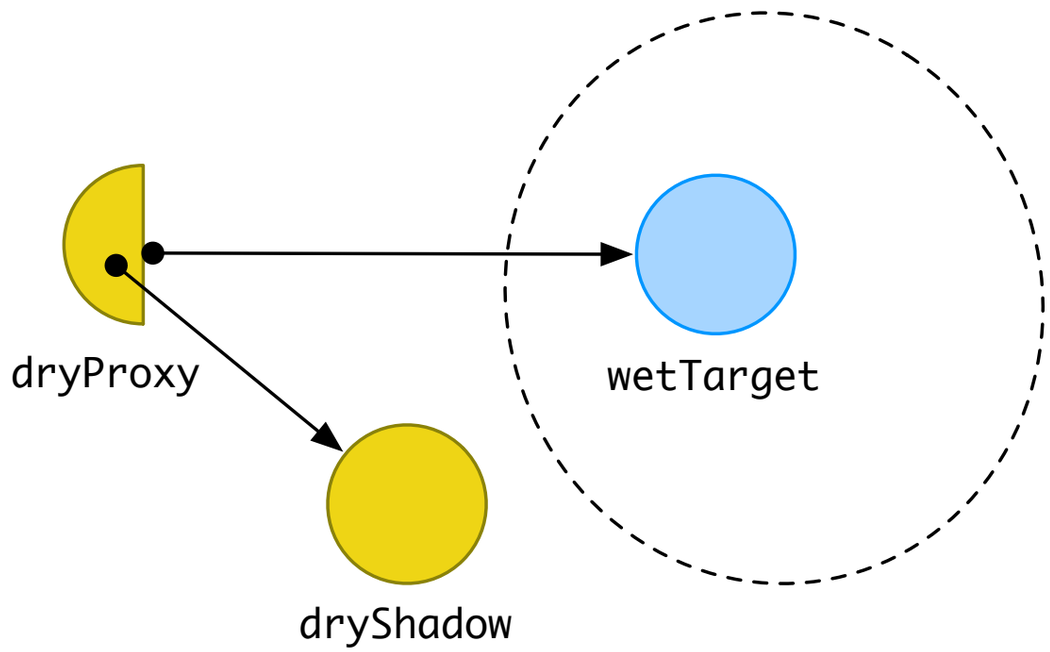


- General solution, not specific to membranes

Membranes with direct proxies: try 2

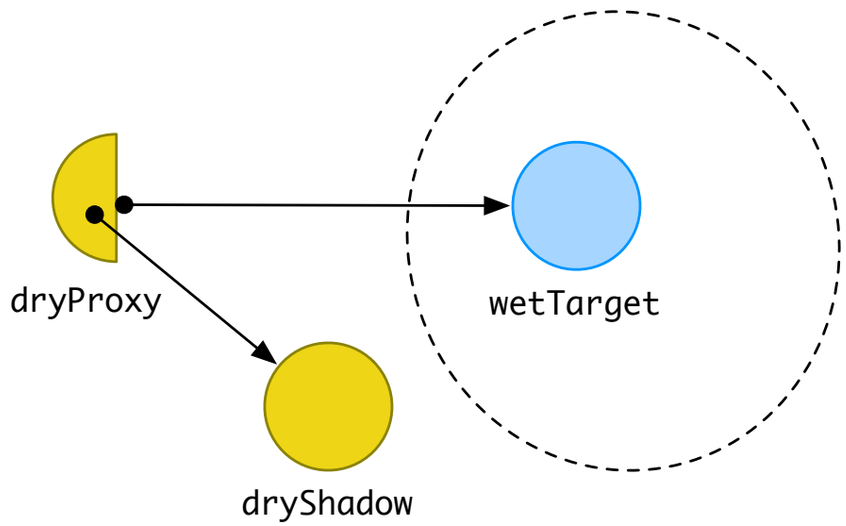
- In the case of membranes: shadow and real target are on opposite sides of the membrane

```
var wetB = {};
var wetA = Object.freeze({ x: wetB });

var dryA = wet2dry(wetA);
var dryB = dryA.x; // ok: shadowTarget.x === dryB
```



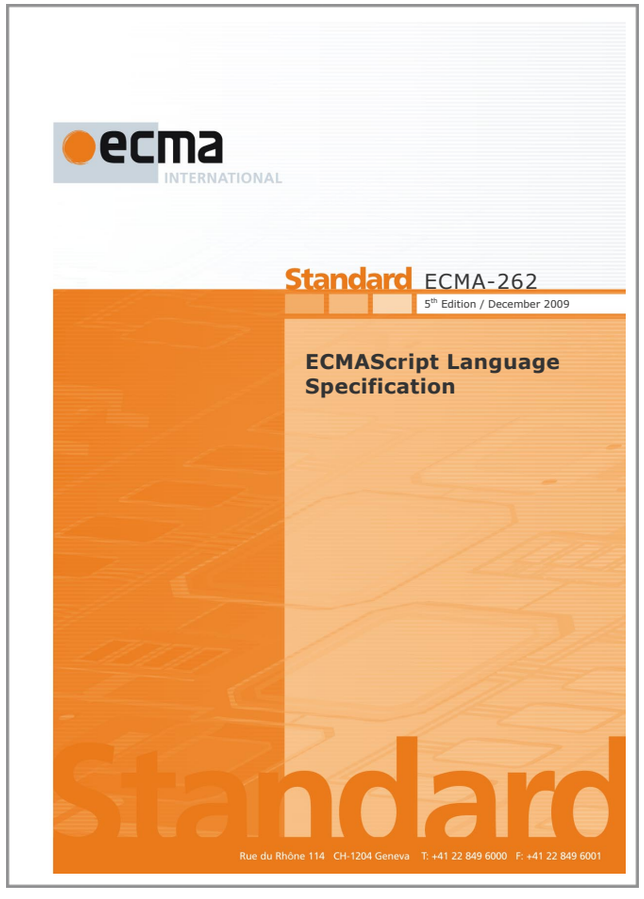
Membranes with direct proxies: conclusion

- Direct proxies at least as expressive as original proxies
- Can maintain invariants transparently across a membrane
- But, the shadow target technique required to maintain invariants...
- ... is cumbersome: must keep shadow and real target “in sync”
- ... is more expensive: extra allocated object per wrapped object



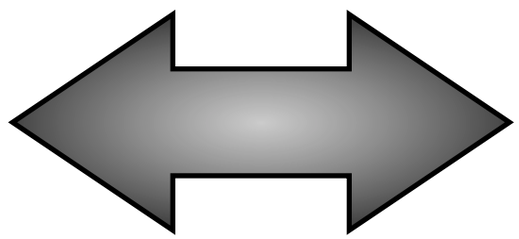
Direct proxies: status

- Part of the ECMAScript 6 draft standard
- Implemented in Firefox 18+

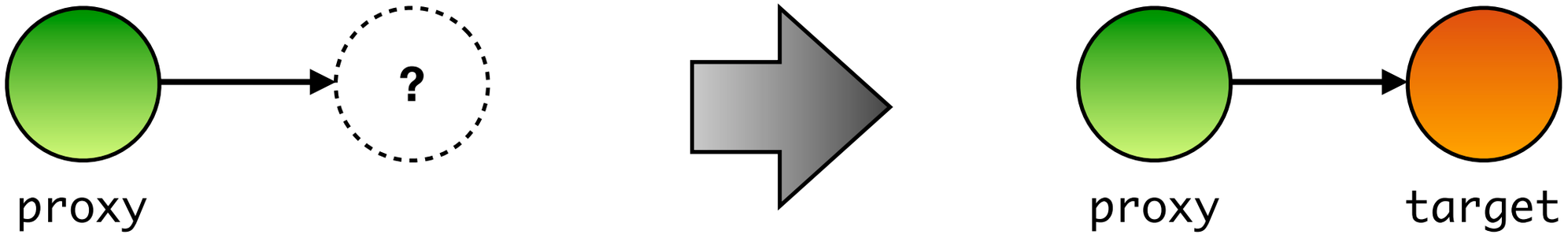


Conclusion

Powerful reflection API
(Proxies)



Strong language invariants
(Frozen Objects)



- Direct proxies are trustworthy, they do not violate language invariants
- Ensured by inserting runtime invariant assertions