Proxies

Design Principles for Robust OO Intercession APIs

Tom Van Cutsem

Mark S. Miller

Overview

- Context: a new meta-programming API for Javascript (based on ECMAScript 5th ed.)
  - Focus on intercession, intercepting meta-level operations on *proxies*
- Paper: generalized *design principles* for message-based OO meta-programming APIs

Ecmascript 5

- New reflection API
  - exposes *property attributes*

```
var point = {
  x: 5,
  get y() { return this.x; }
};
```

```
Object.defineProperty(point, ‘z’, {
  value: 42,
  writable: false,
  enumerable: true,
  configurable: true
});
```

Tamper-proof Objects

```
var point =
  { x: 5,
    get y() { return this.x; } };


Object.preventExtensions(point);
point.z = 0; // can't add new properties



Object.seal(point);
delete point.x; // can't delete properties



Object.freeze(point);
point.x = 7; // can't assign properties
```

Proxies

Intercession in Javascript today

- non-standard __noSuchMethod__ hook in Firefox
- modelled after Smalltalk's "doesNotUnderstand:"

```
function makeProxy(target) {
  return {
    __noSuchMethod__: function(name, args) {
      return target[name].apply(target, args);
    }
  };
}
```

__noSuchMethod__

- not stratified (defined as a regular application-level method)
- limited intercession (intercepts only missing method calls)

```
var p = makeProxy({foo: 42});
‘foo’ in p // false
p.__noSuchMethod__ // reveals the method
for (var name in p) {
  // reveals ‘__noSuchMethod__’ but not ‘foo’
}
```

Dynamic Proxies

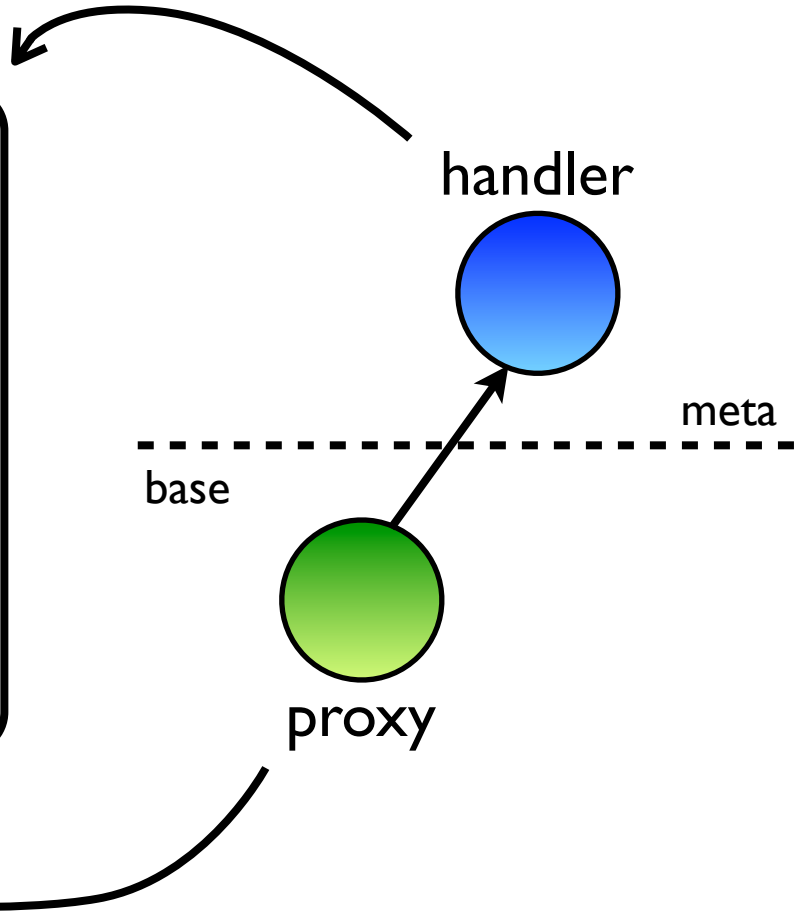
- Objects that "look and feel" like normal objects, but whose behavior is controlled by another Javascript object
- Some use cases:
  - Generic wrappers around existing objects: access control, tracing, profiling, contracts, ...
  - Virtual objects: remote objects, mock objects, ...

Tracing: dynamic proxies

```
function makeTracer(obj) {
  var proxy = Proxy.create({
    get: function(rcvr, name) {
      trace('get', name, obj);
      return obj[name];
    },
    set: function(rcvr, name, val) {
      trace('set', name, obj, val);
      obj[name] = val;
      return true;
    },
    ...
  }, Object.getPrototypeOf(obj));
  return proxy;
}
```

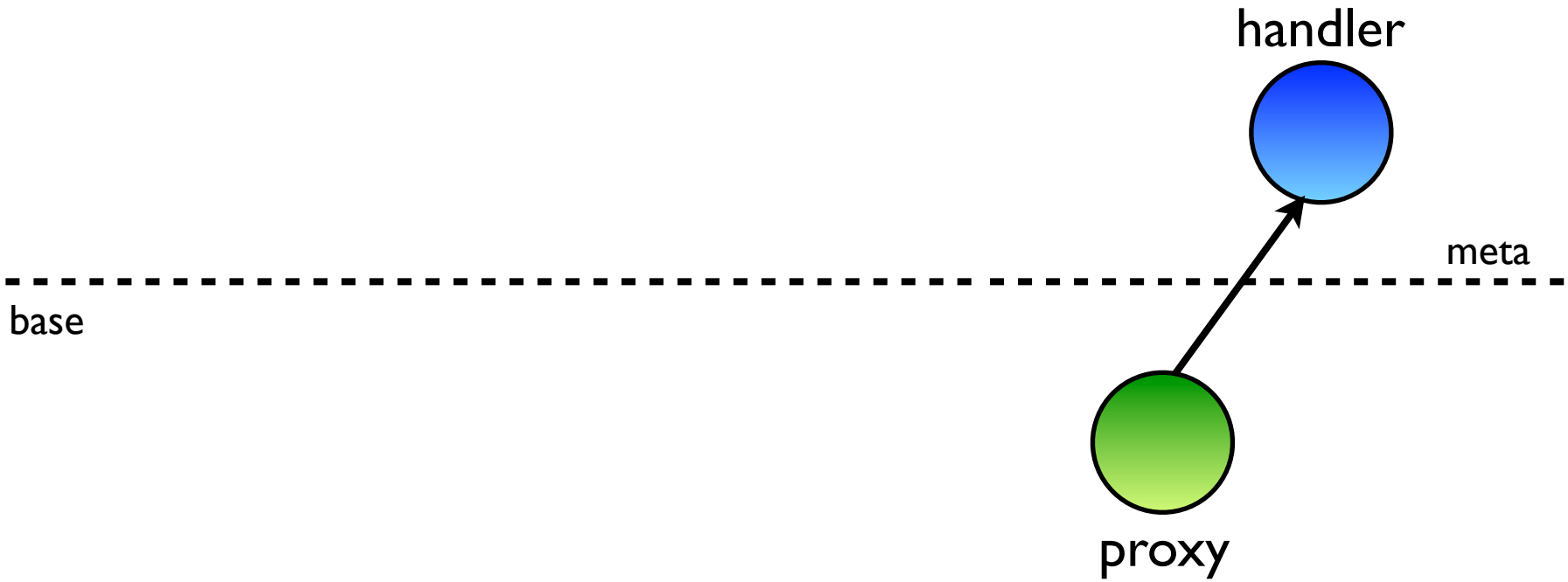
Tracing: dynamic proxies

```
function makeTracer(obj) {
  var proxy = Proxy.create({
    get: function(rcvr, name) {
      trace('get', name, obj);
      return obj[name];
    },
    set: function(rcvr, name, val) {
      trace('set', name, obj, val);
      obj[name] = val;
      return true;
    },
    ...
  }, Object.getPrototypeOf(obj));
  return proxy;
}
```



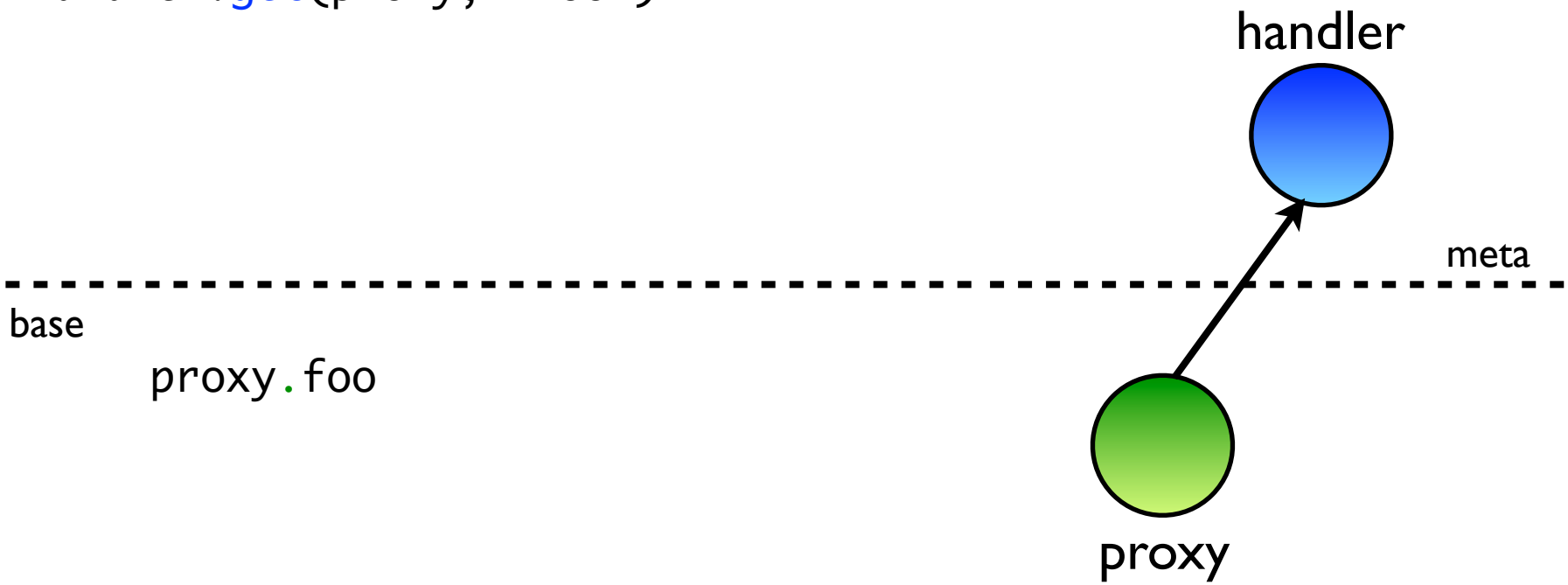
Stratified API

```
var proxy = Proxy.create(handler, proto);
```



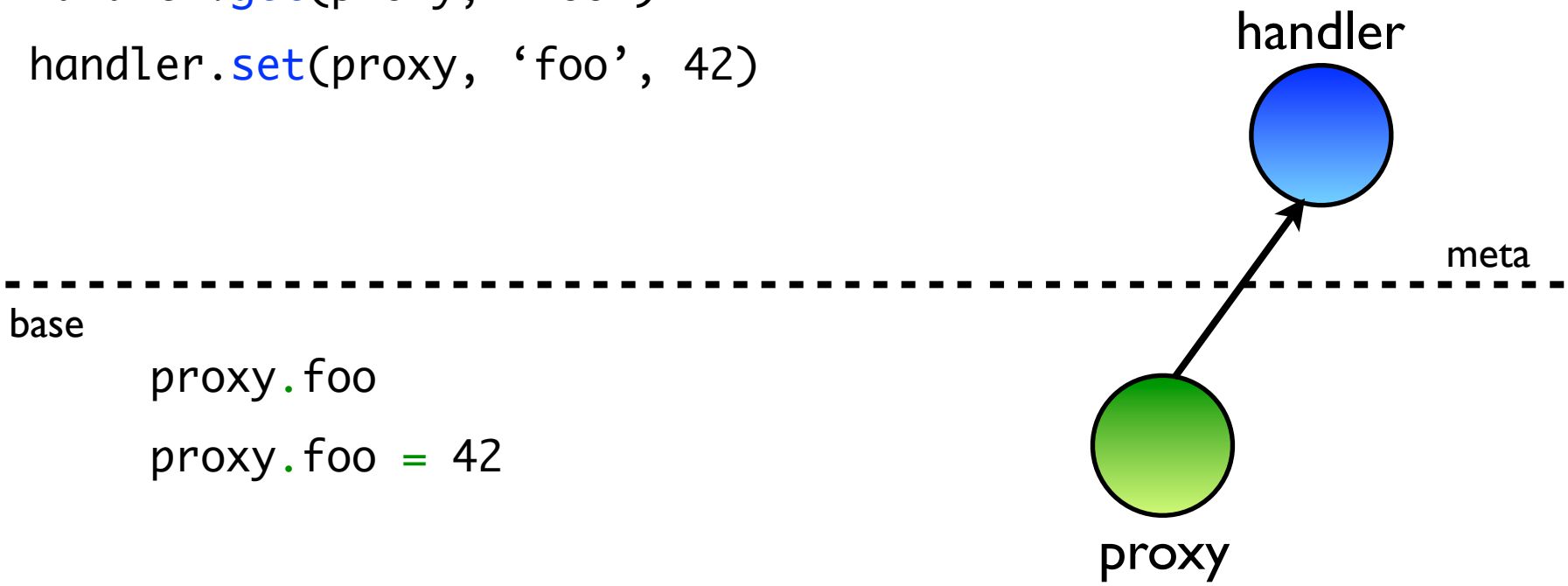
Stratified API

```
var proxy = Proxy.create(handler, proto);

handler.get(proxy, 'foo')
```



Stratified API

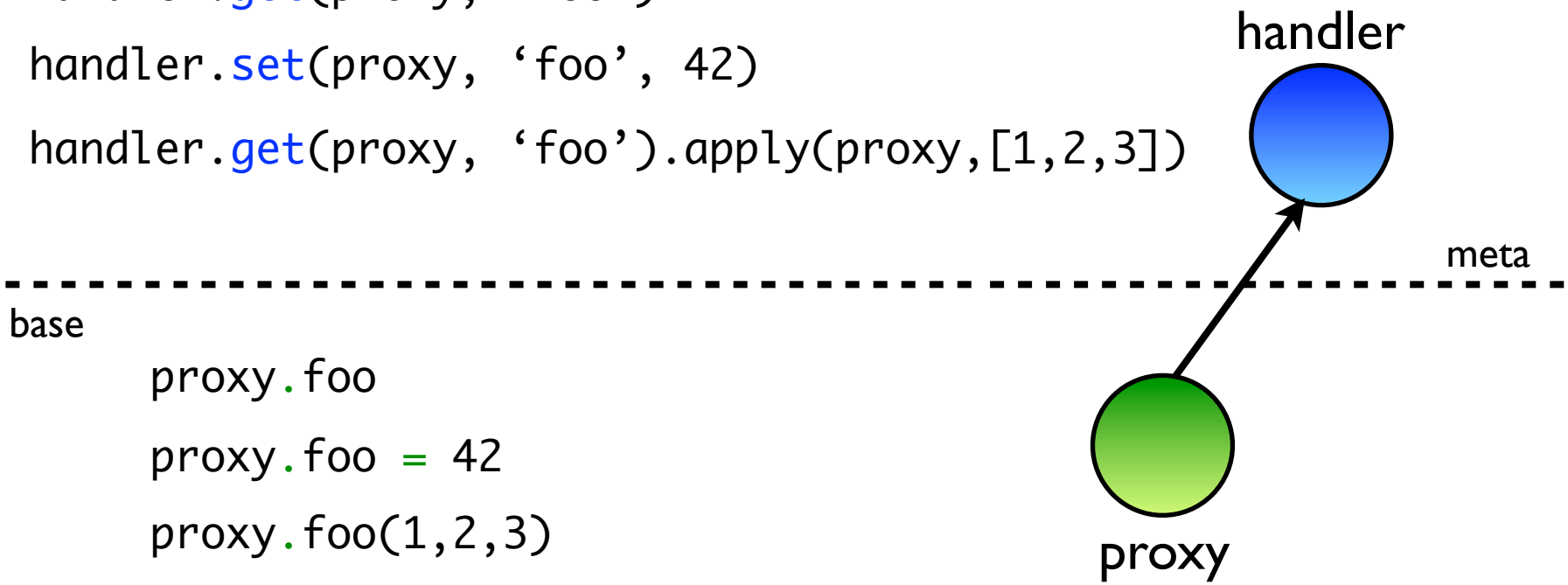
```
var proxy = Proxy.create(handler, proto);

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



Stratified API

```
var proxy = Proxy.create(handler, proto);
```

```
handler.get(proxy, 'foo')
handler.set(proxy, 'foo', 42)
handler.get(proxy, 'foo').apply(proxy,[1,2,3])
```



Stratified API

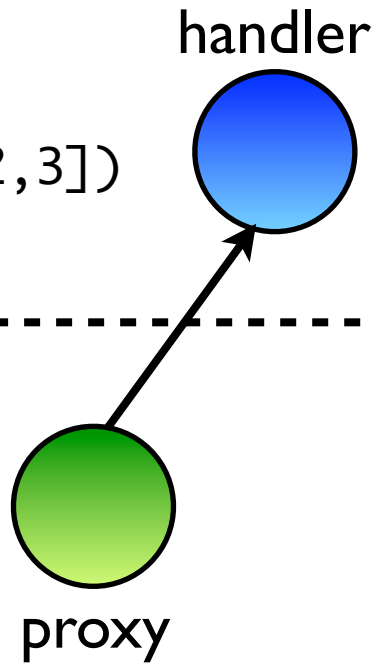
```
var proxy = Proxy.create(handler, proto);
```

```
handler.get(proxy, ‘foo’)
handler.set(proxy, ‘foo’, 42)
handler.get(proxy, ‘foo’).apply(proxy,[1,2,3])
handler.get(proxy, ‘get’)
```

meta

base

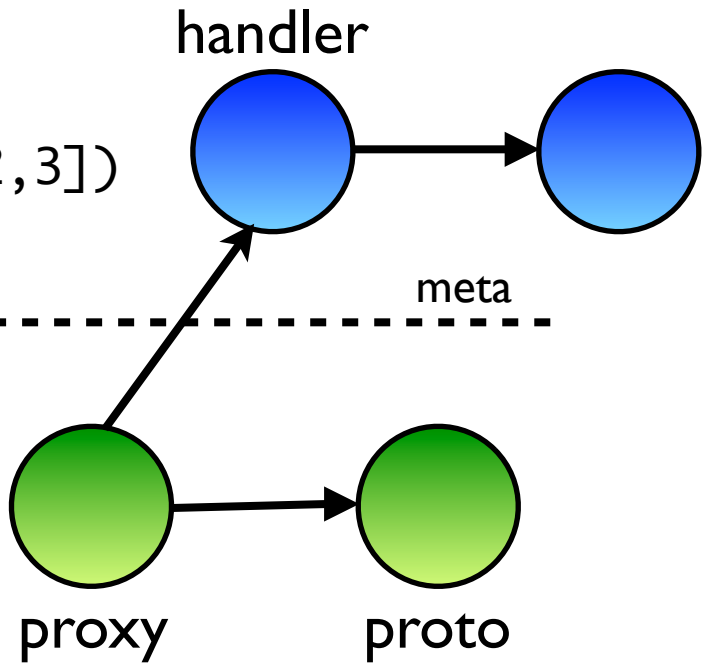
```
proxy.foo
proxy.foo = 42
proxy.foo(1,2,3)
proxy.get
```



Stratified API

```
var proxy = Proxy.create(handler, proto);
```

```
handler.get(proxy, 'foo')
handler.set(proxy, 'foo', 42)
handler.get(proxy, 'foo').apply(proxy,[1,2,3])
handler.get(proxy, 'get')
```

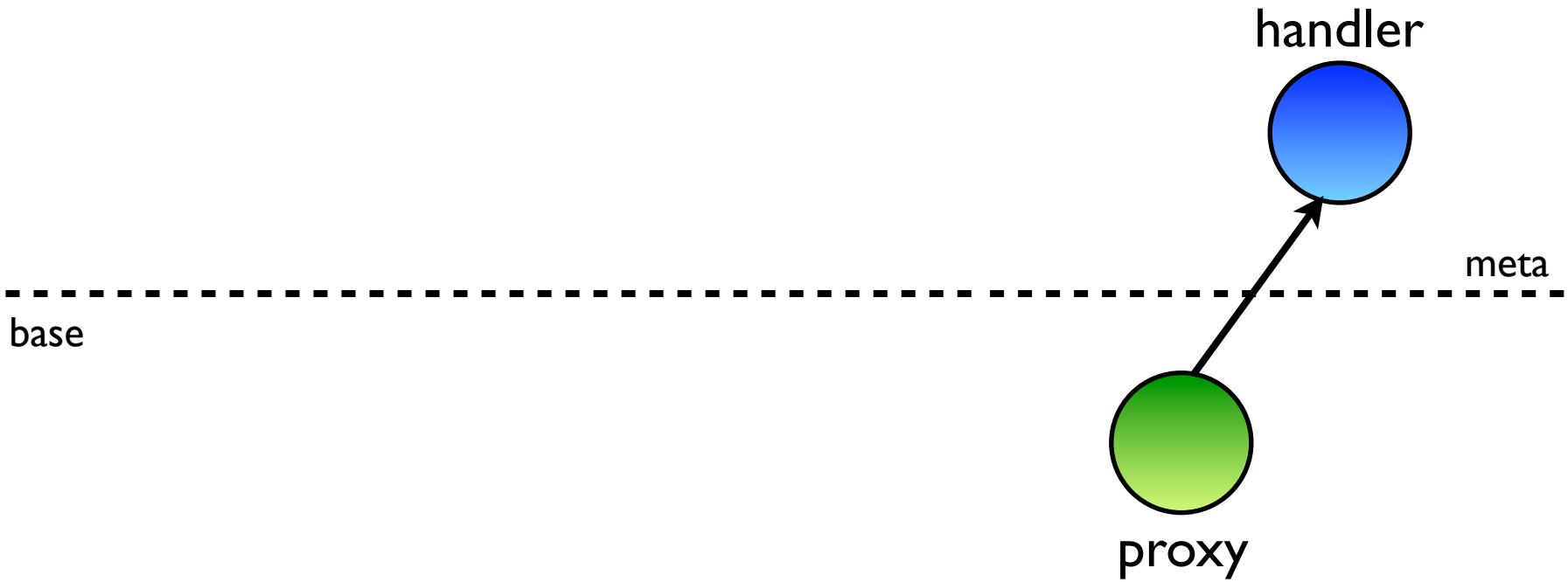


base

```
proxy.foo
proxy.foo = 42
proxy.foo(1,2,3)
proxy.get
```

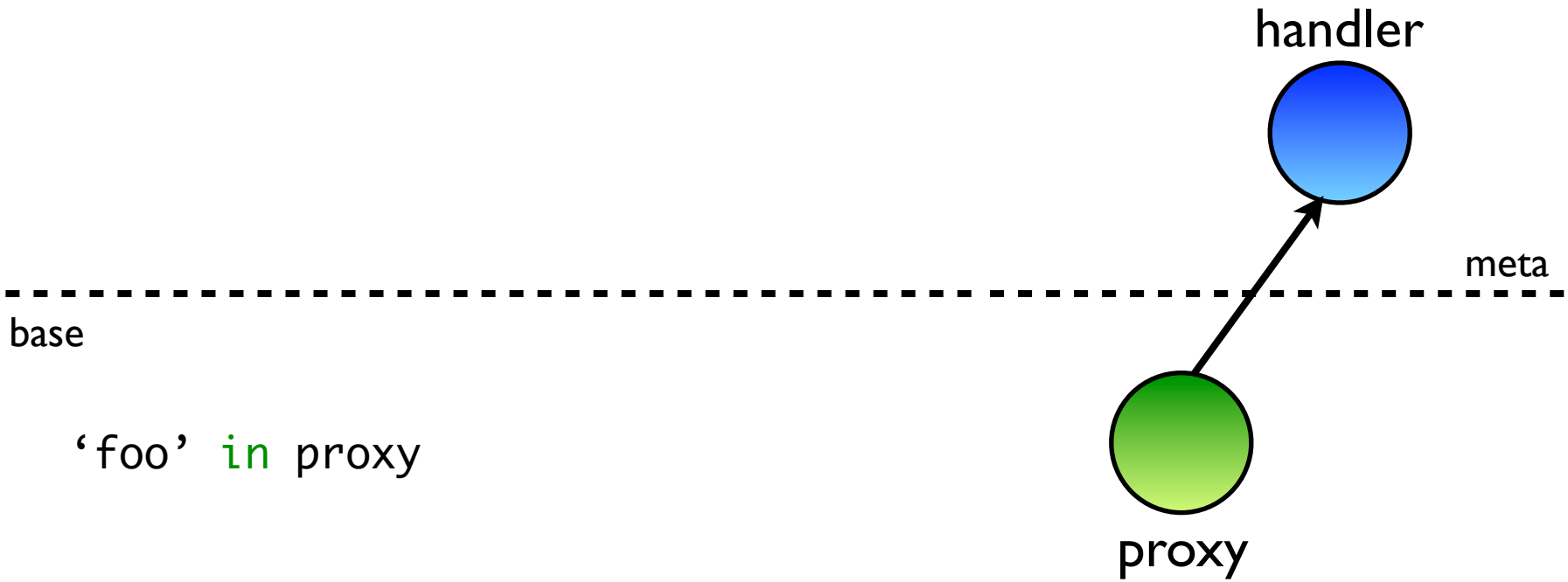
Not just property access

```
var proxy = Proxy.create(handler, proto);
```



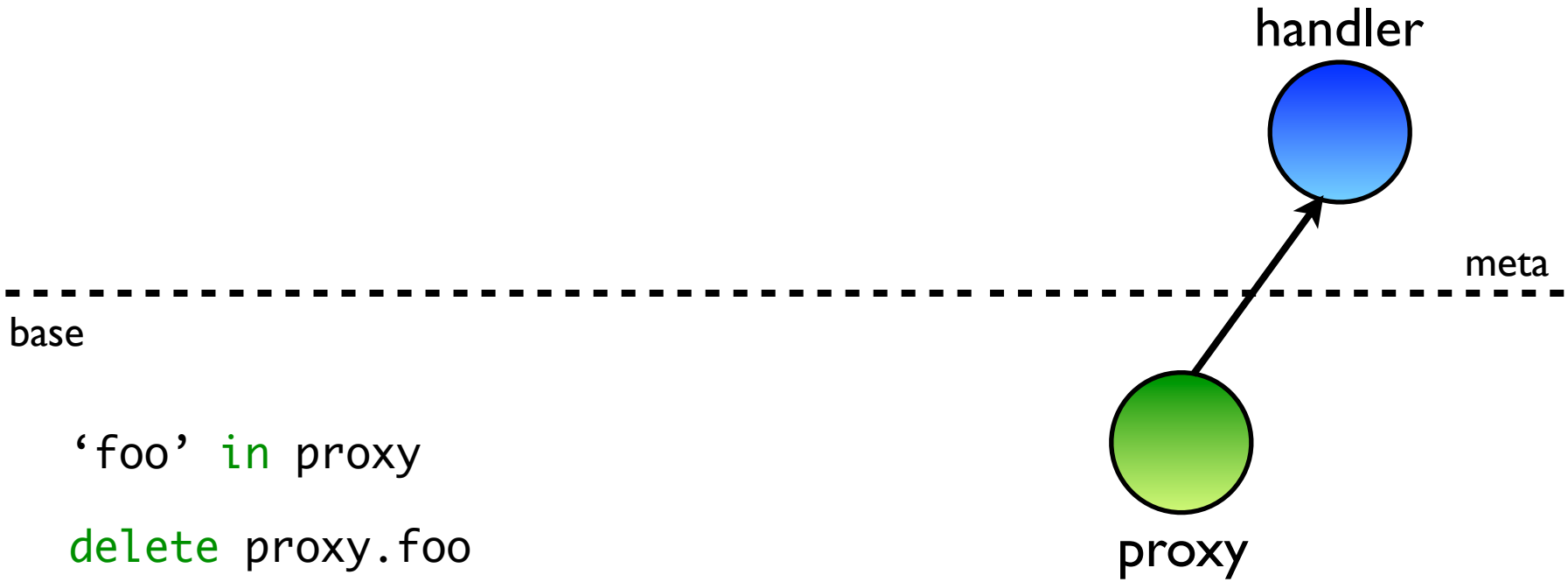
Not just property access

```
var proxy = Proxy.create(handler, proto);

  handler.has('foo')
```



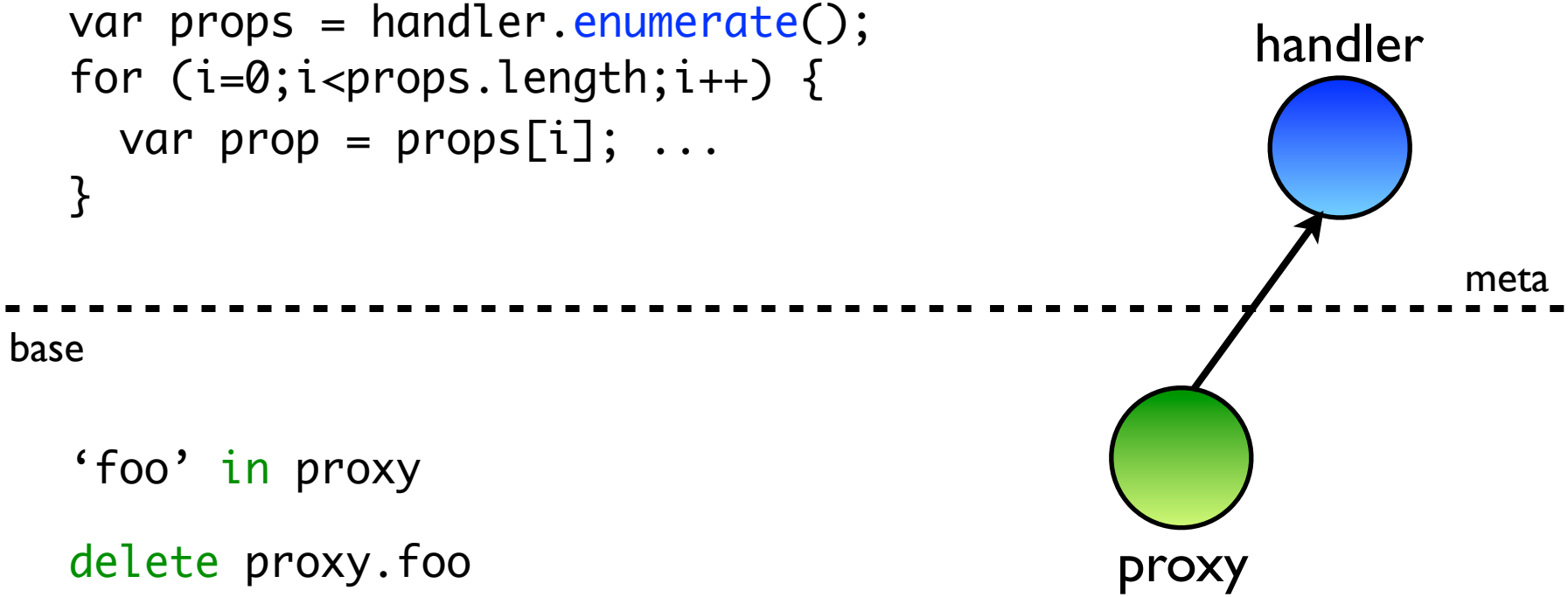
Not just property access

```
var proxy = Proxy.create(handler, proto);

  handler.has('foo')

  handler.delete('foo')
```



Not just property access

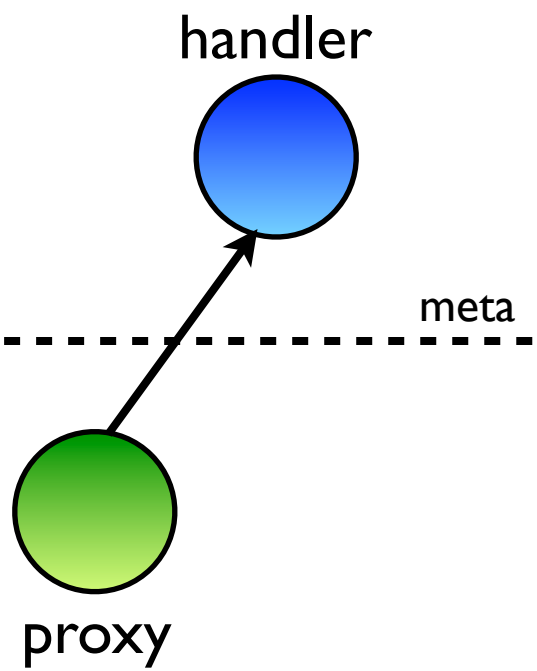
```
var proxy = Proxy.create(handler, proto);

  handler.has('foo')

  handler.delete('foo')

  var props = handler.enumerate();
  for (i=0;i<props.length;i++) {
    var prop = props[i]; ...
  }
```




```
  'foo' in proxy

  delete proxy.foo

  for (var prop in proxy) { ... }
```

Not just property access

```
var proxy = Proxy.create(handler, proto);

  handler.has('foo')

  handler.delete('foo')

  var props = handler.enumerate();
  for (i=0;i<props.length;i++) {
    var prop = props[i]; ...
  }
  handler.defineProperty('foo', pd)
```

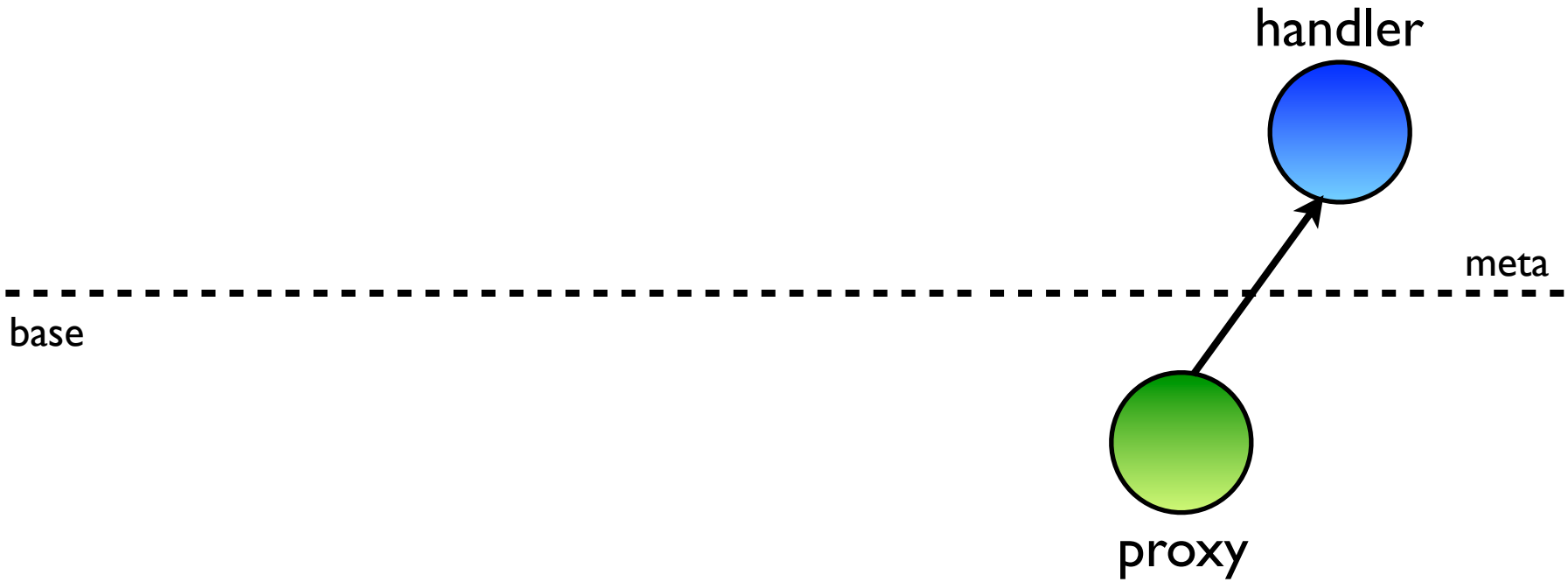


base

```
  'foo' in proxy

  delete proxy.foo

  for (var prop in proxy) { ... }

  Object.defineProperty(proxy,'foo', pd)
```

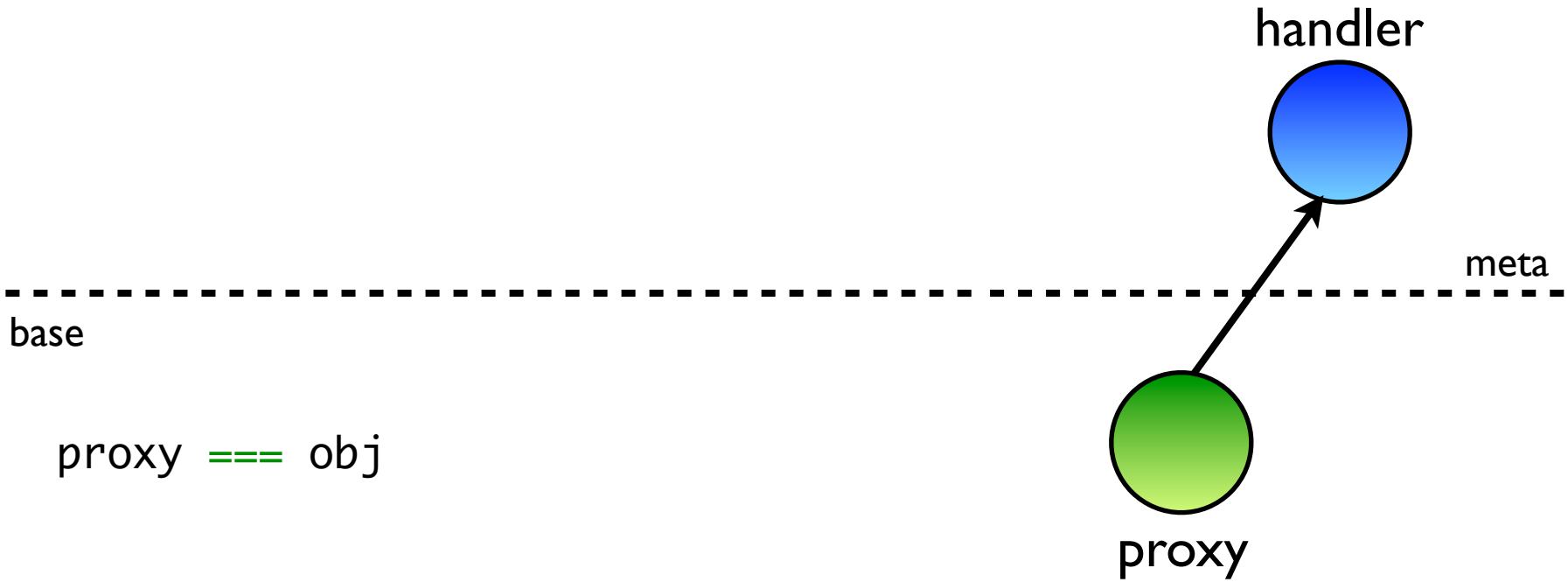
But not quite everything either

```
var proxy = Proxy.create(handler, proto);
```



But not quite everything either

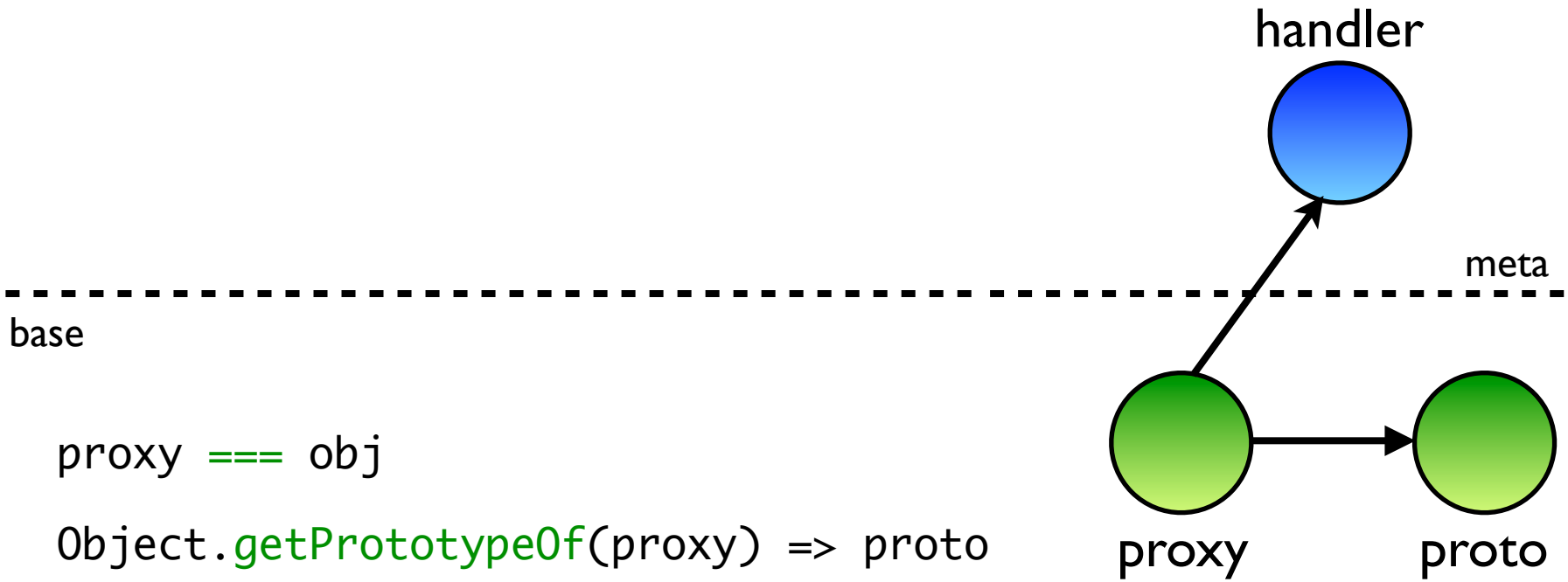
```
var proxy = Proxy.create(handler, proto);
```




```
proxy === obj
```

But not quite everything either

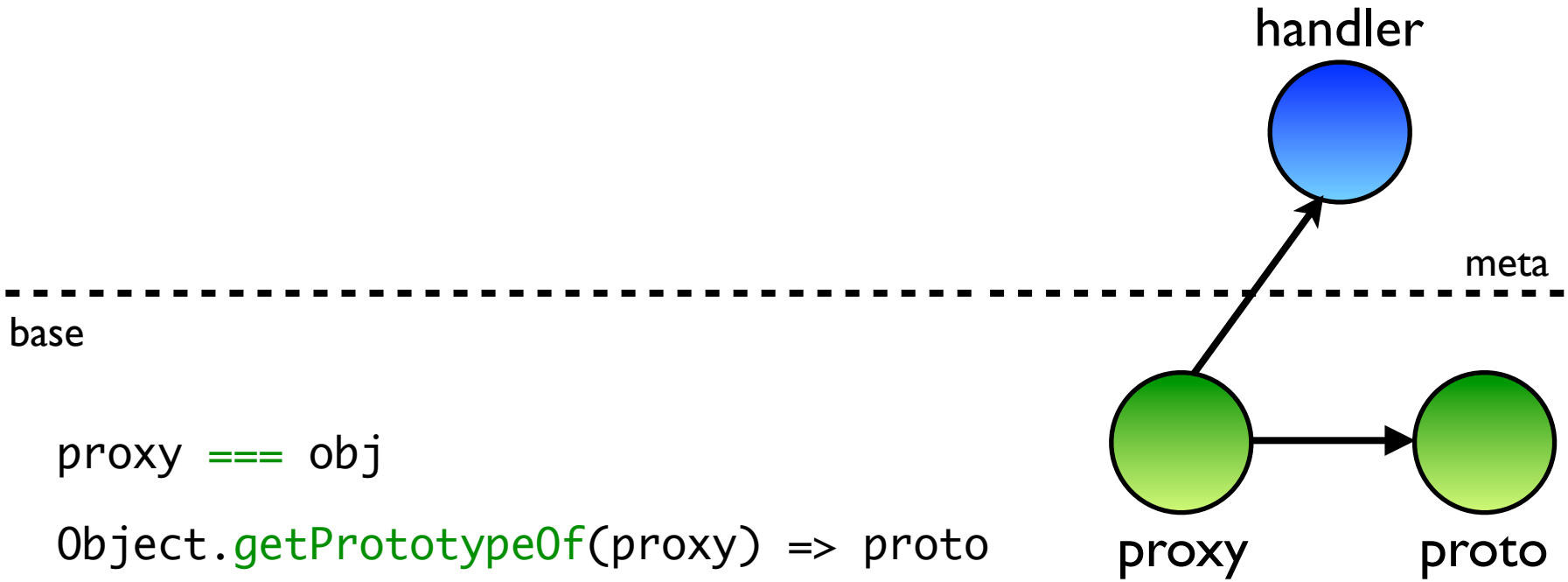
```
var proxy = Proxy.create(handler, proto);
```




```
proxy === obj

Object.getPrototypeOf(proxy) => proto
```

But not quite everything either

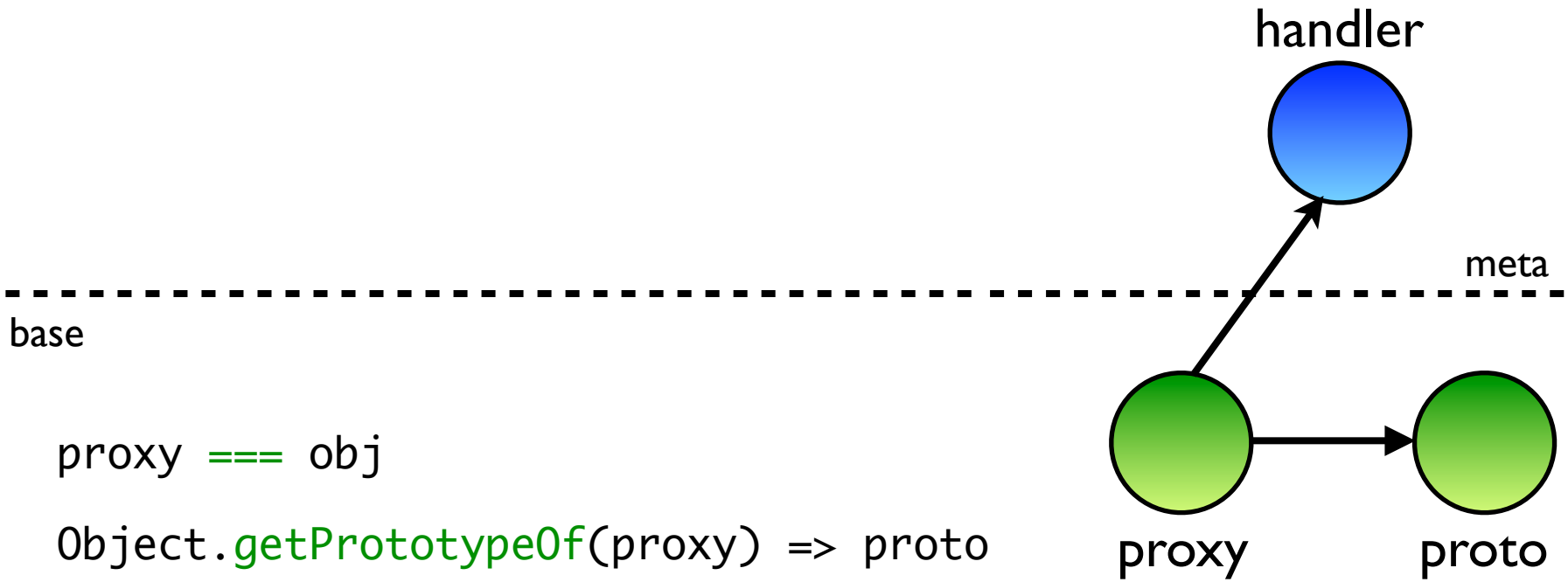
```
var proxy = Proxy.create(handler, proto);
```




```
proxy === obj
Object.getPrototypeOf(proxy) => proto
proxy instanceof Fun
```

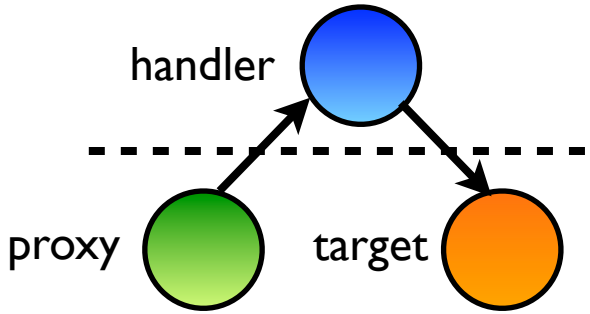
But not quite everything either

```
var proxy = Proxy.create(handler, proto);
```




```
proxy === obj
Object.getPrototypeOf(proxy) => proto
proxy instanceof Fun
typeof proxy => “object”
```

Example: no-op forwarding proxy




```
function ForwardingHandler(obj) {
  this.target = obj;
}

ForwardingHandler.prototype = {
  has: function(name) { return name in this.target; },
  get: function(rcvr,name) { return this.target[name]; },
  set: function(rcvr,name,val) { this.target[name]=val;return true; },
  delete: function(name) { return delete this.target[name]; }
  ...
}

var proxy = Proxy.create(new ForwardingHandler(o),
                         Object.getPrototypeOf(o));
```

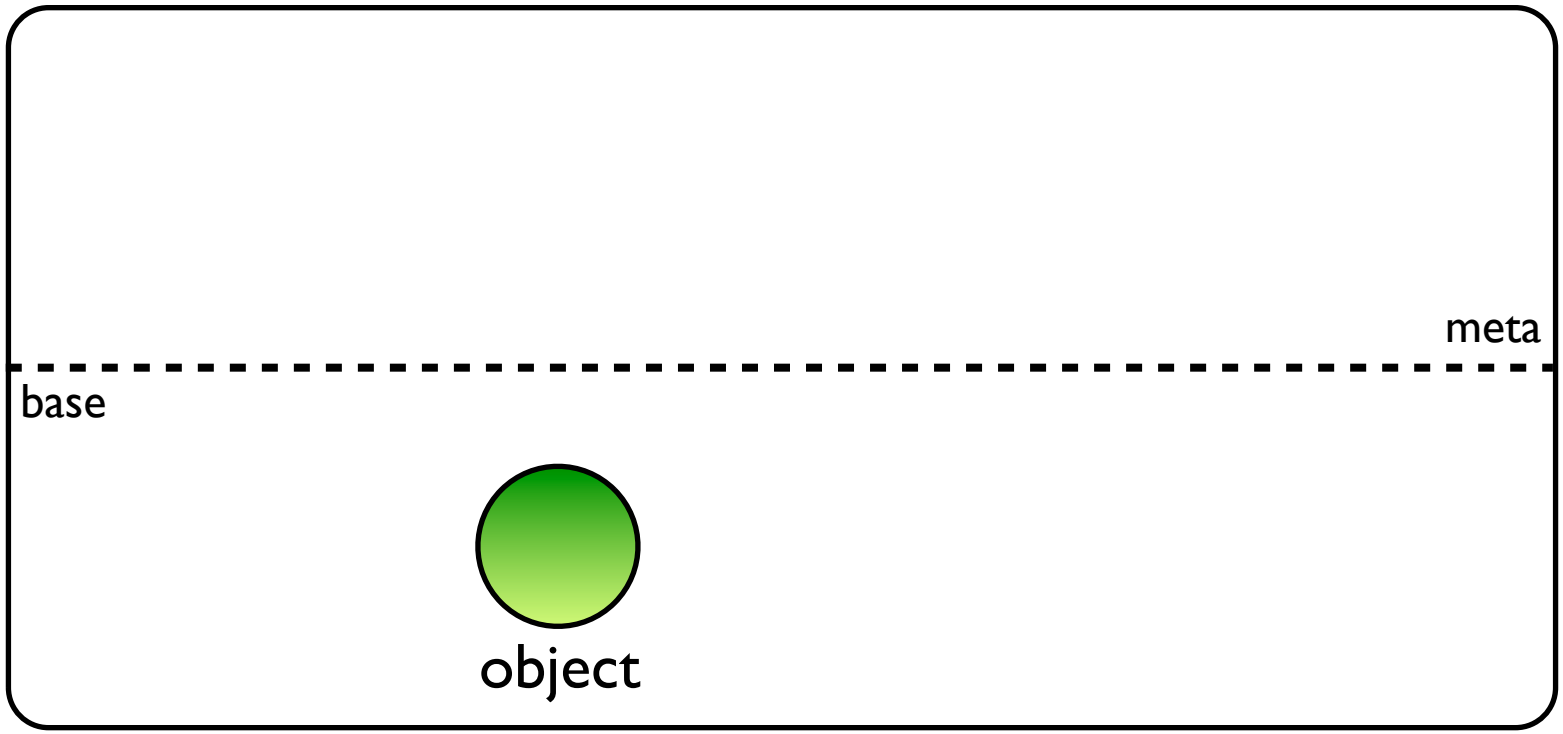
Example: counting property access

```
function makeSimpleProfiler(target) {
  var forwarder = new ForwardingHandler(target);
  var count = Object.create(null);
  forwarder.get = function(rcvr, name) {
    count[name] = (count[name] || 0) + 1;
    return this.target[name];
  };
  return {
    proxy: Proxy.create(forwarder,
                        Object.getPrototypeOf(target)),
    get stats() { return count; }
  }
}
```

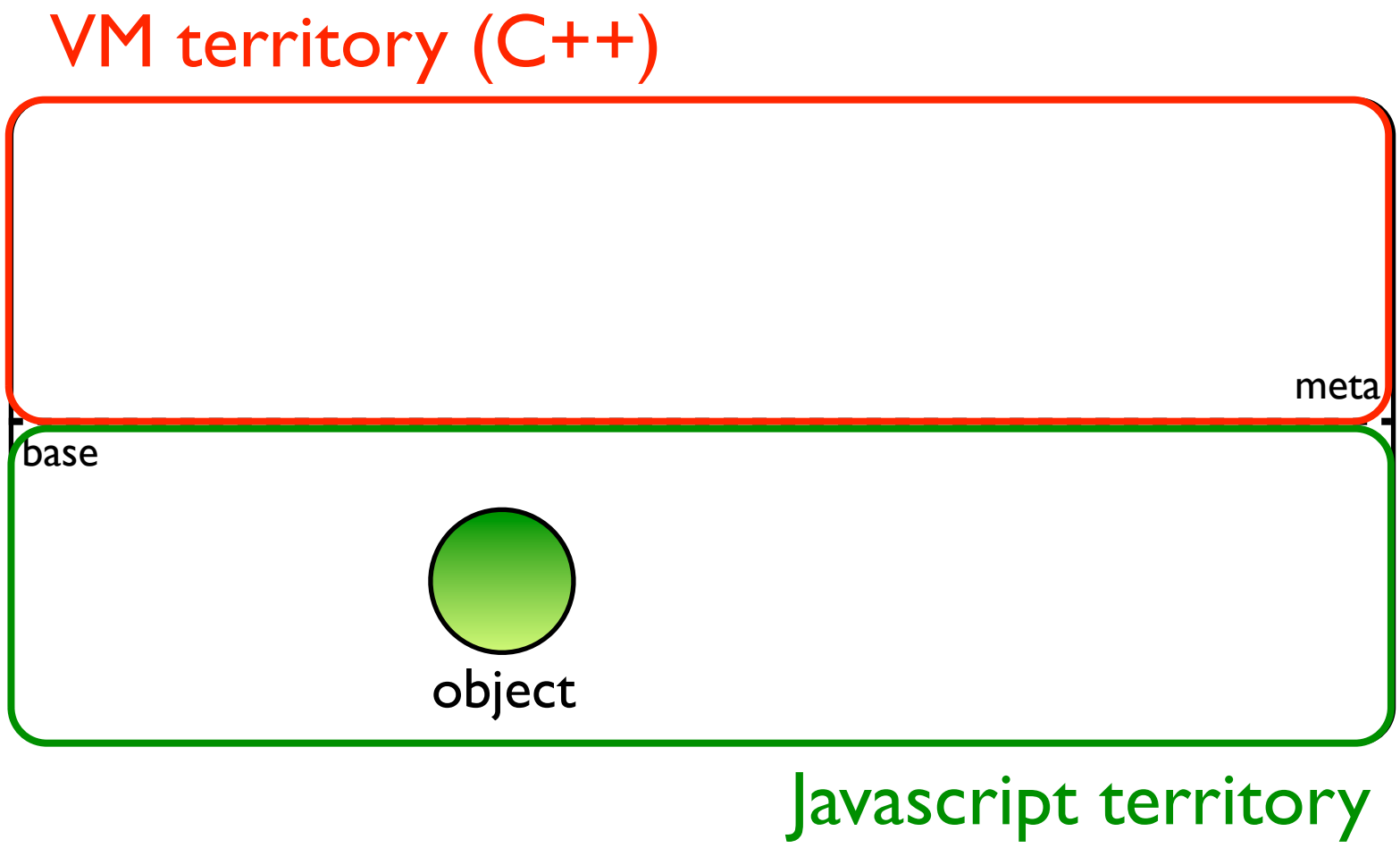
Example: counting property access

```
function makeSimpleProfiler(target) {
  var forwarder = new ForwardingHandler(target);
  var count = Object.create(null);
  forwarder.get = function(rcvr, name) {
    count[name] = (count[name] || 0) + 1;
    return this.target[name];
  };
  return {
    proxy: Proxy.create(forwarder,
                        Object.getPrototypeOf(target)),
    get stats() { return count; }
  }
}

var subject = { ... };
var profiler = makeSimpleProfiler(subject);
runApp(profiler.proxy);
display(profiler.stats);
```

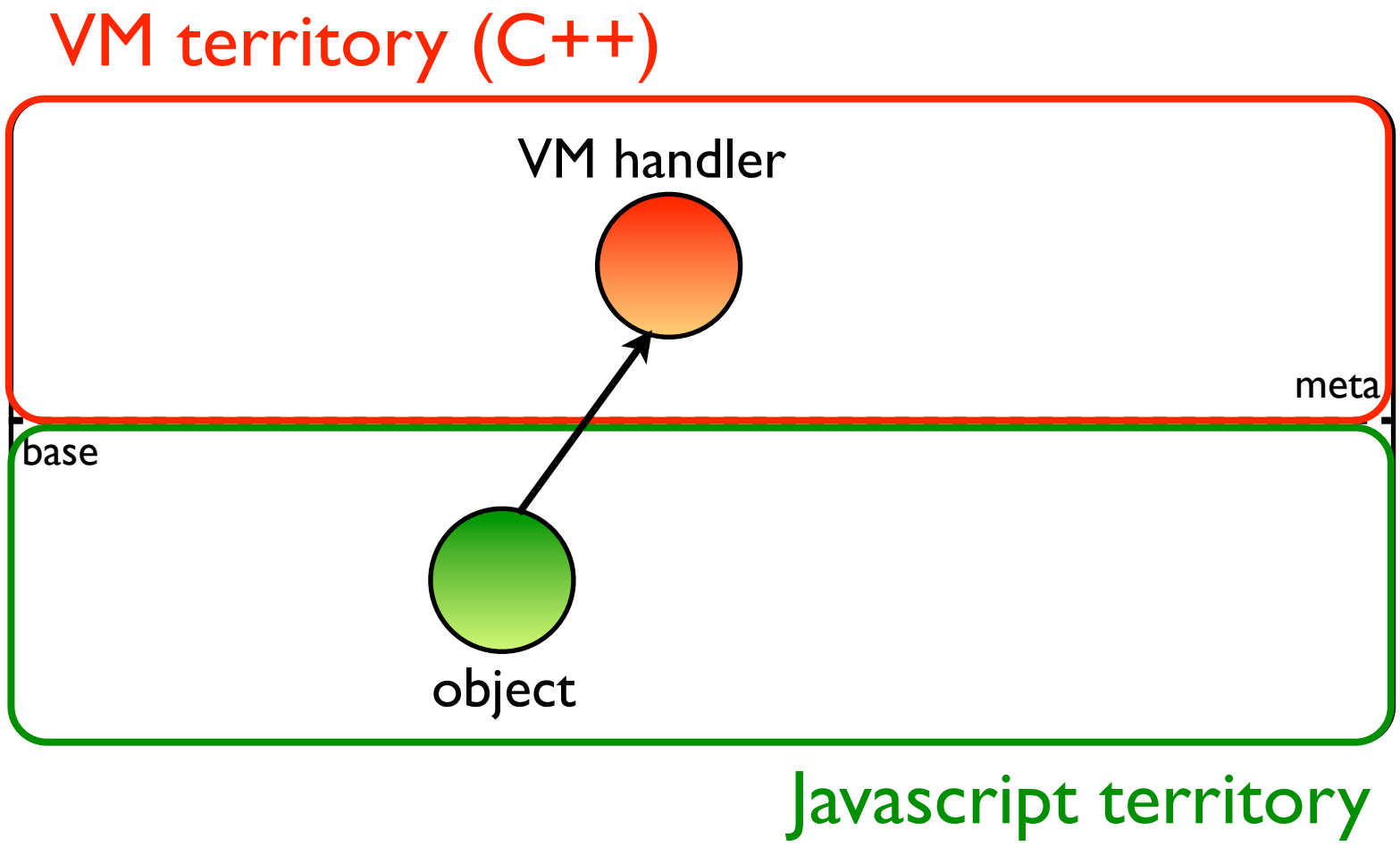
Selective interception



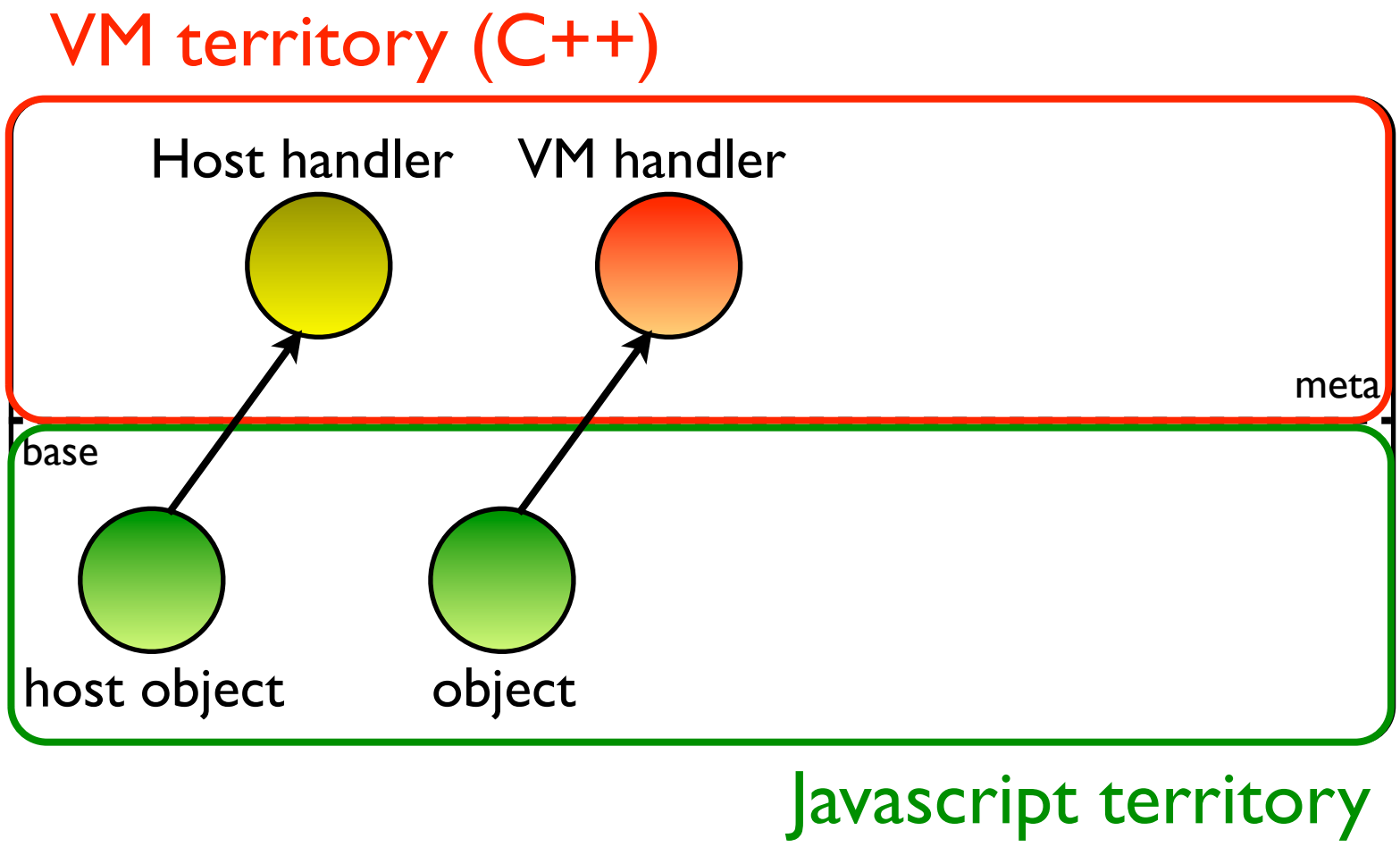
Selective interception



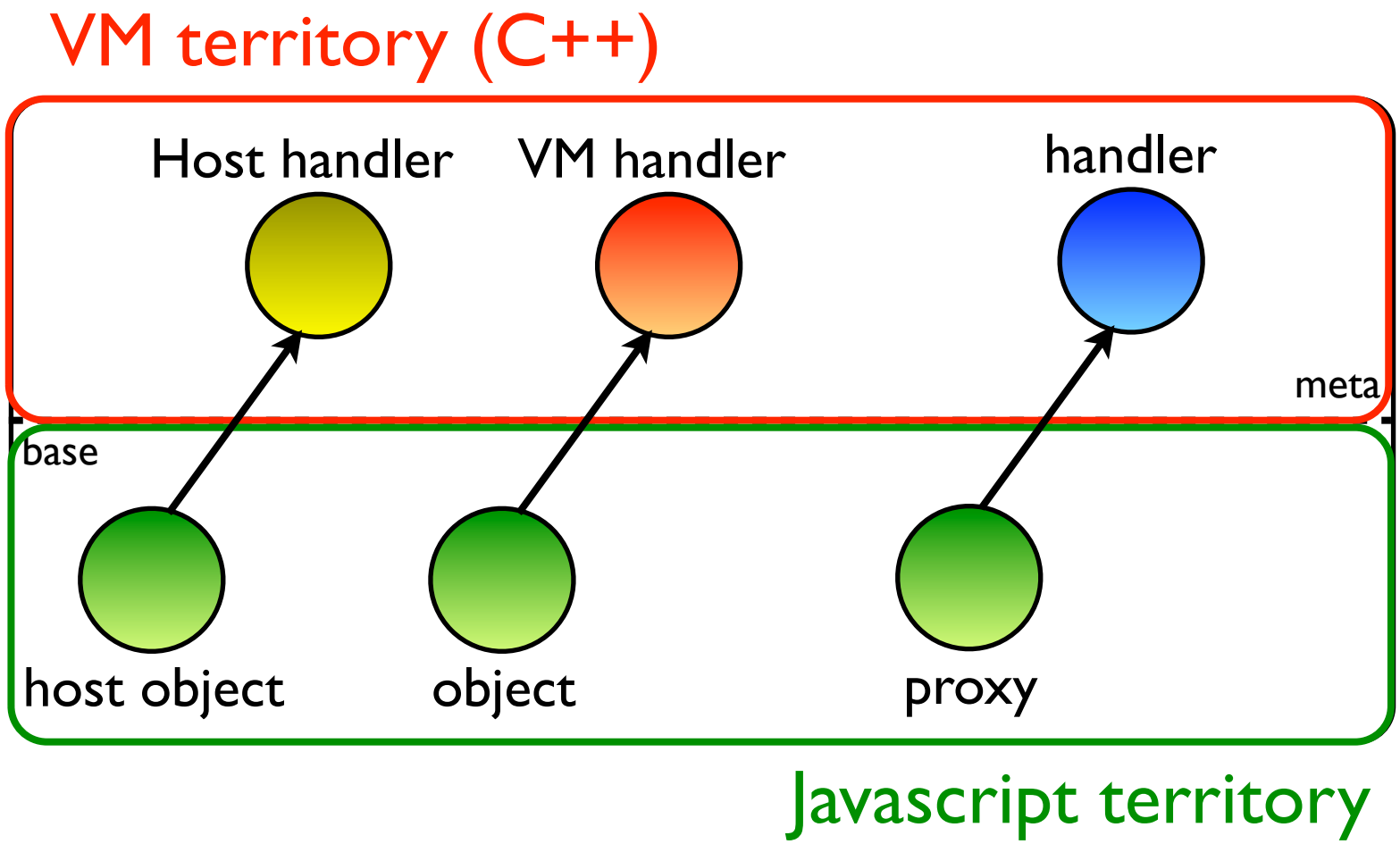
Selective interception



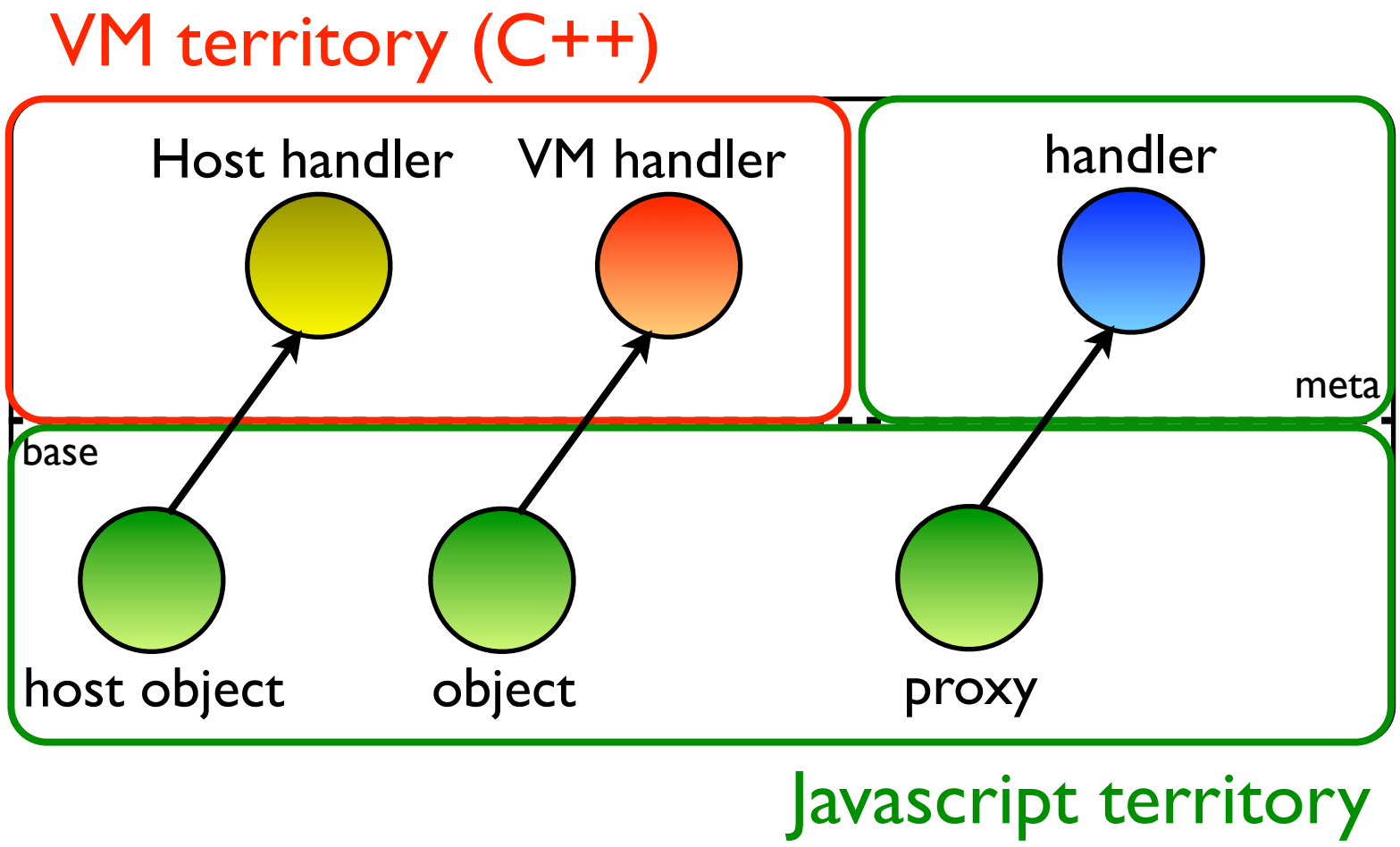
Selective interception



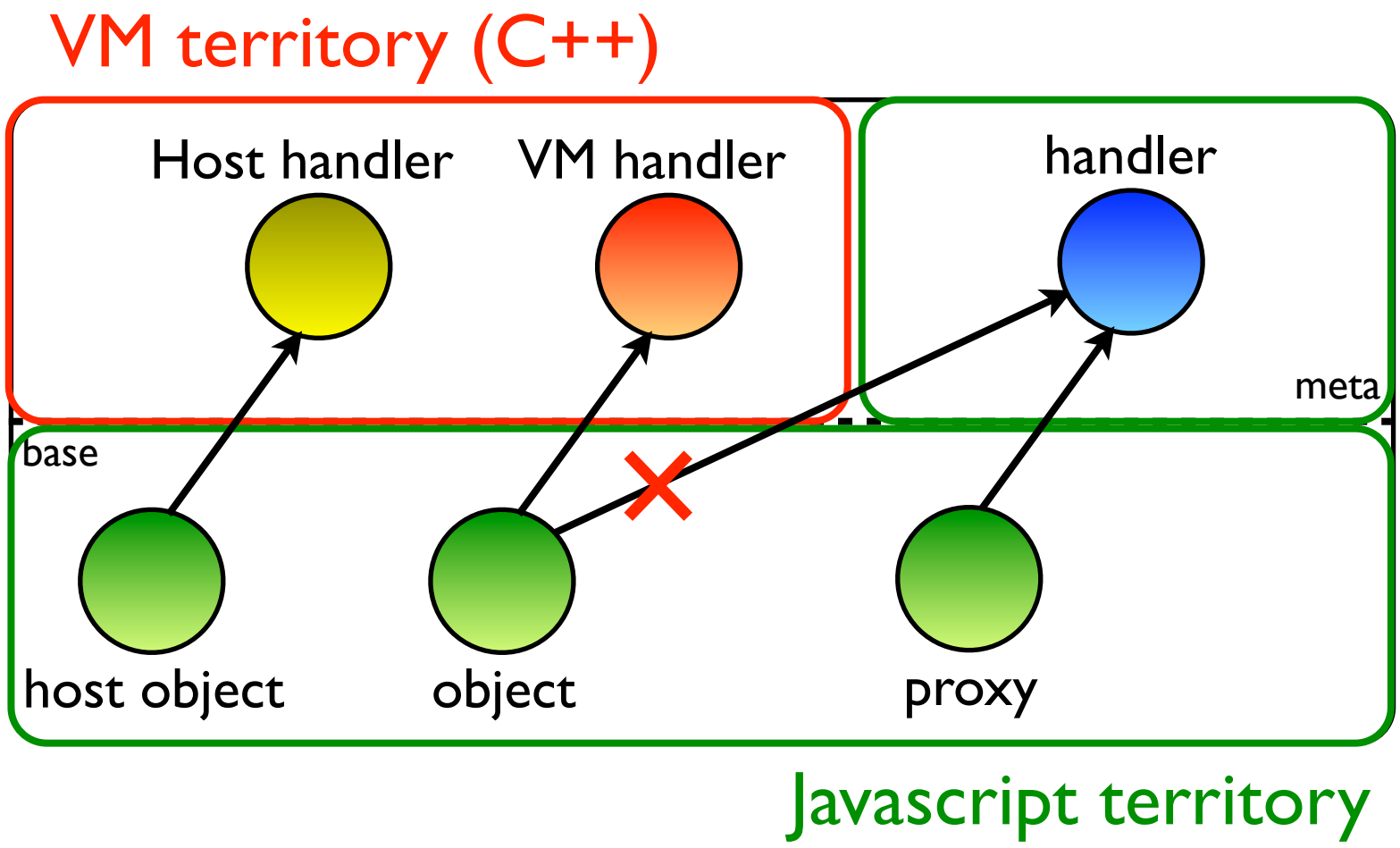
Selective interception



Selective interception

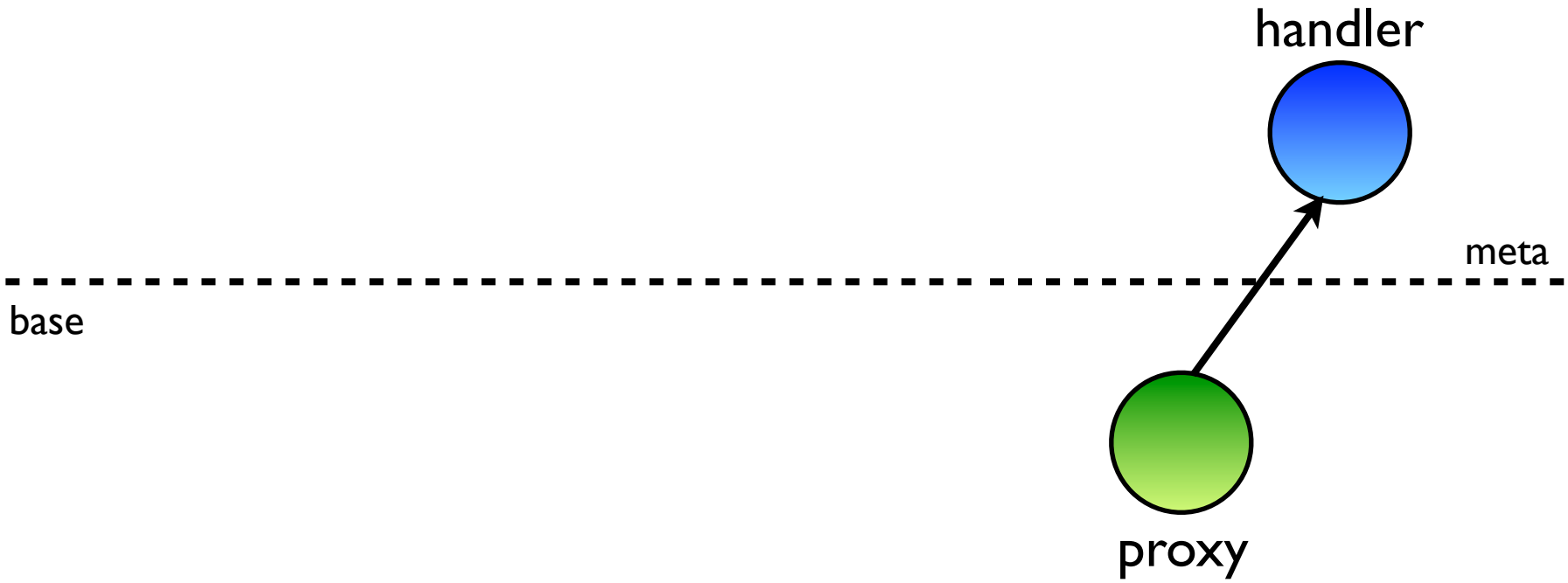


Selective interception



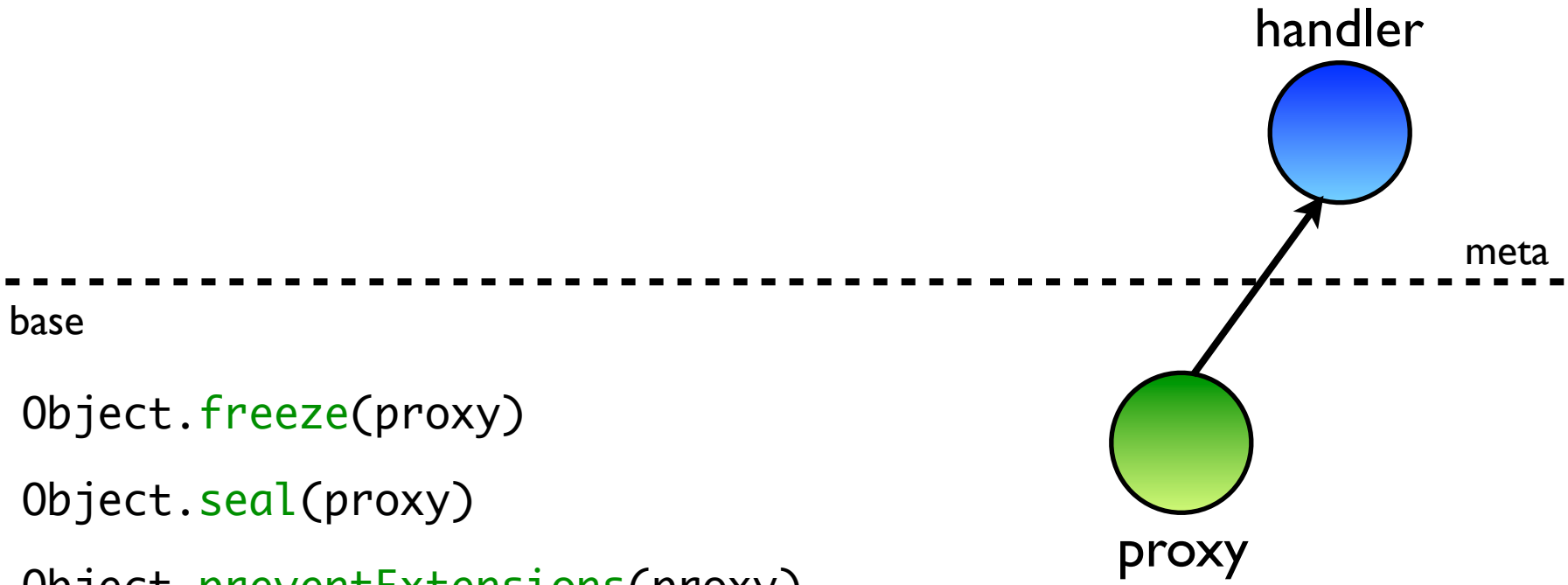
Fixing a Proxy (temporary intercession)

```
var proxy = Proxy.create(handler, proto);
```



Fixing a Proxy (temporary intercession)

```
var proxy = Proxy.create(handler, proto);
```

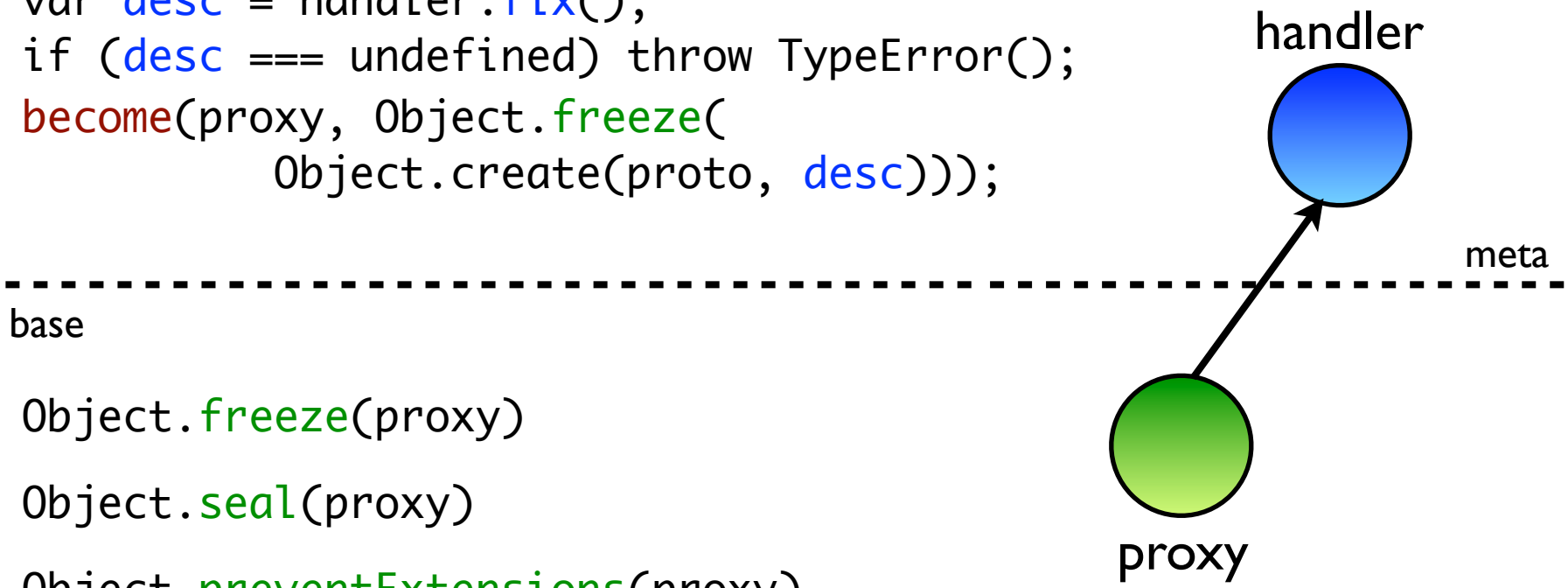



```
Object.freeze(proxy)
Object.seal(proxy)
Object.preventExtensions(proxy)
```

Fixing a Proxy (temporary intercession)

```
var proxy = Proxy.create(handler, proto);
```

```
var desc = handler.fix();
if (desc === undefined) throw TypeError();
become(proxy, Object.freeze(
          Object.create(proto, desc)));
```

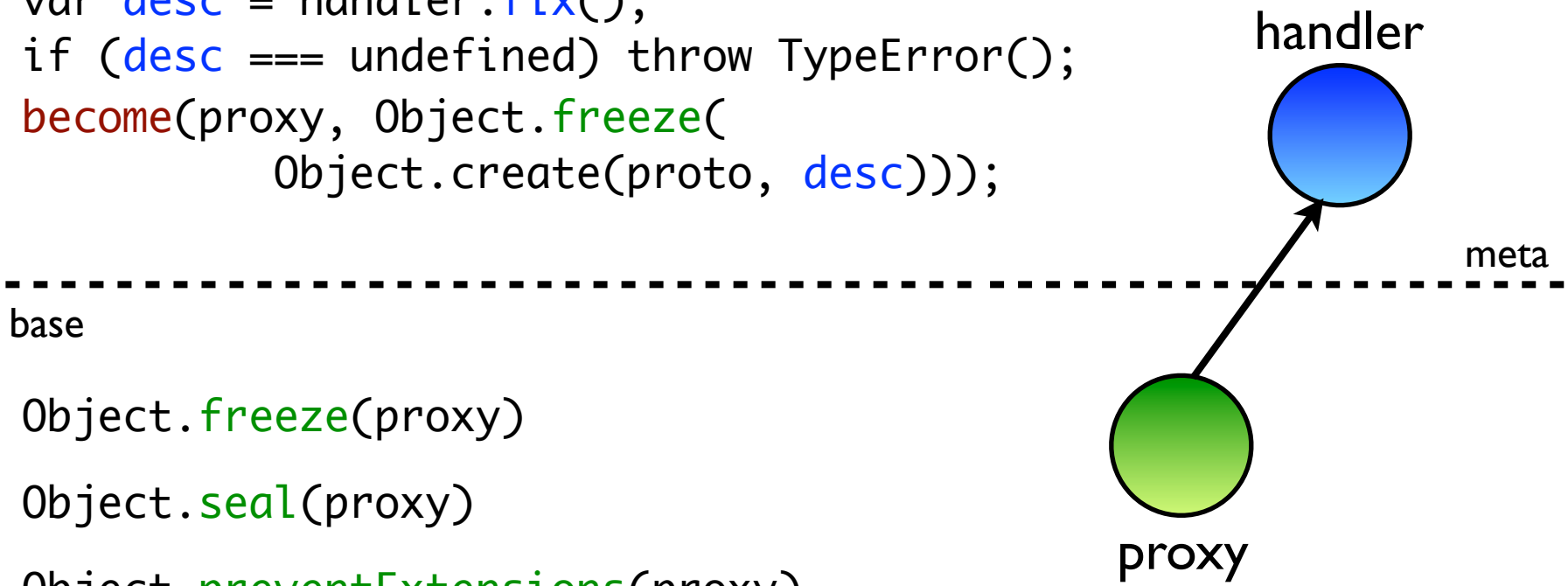



```
Object.freeze(proxy)

Object.seal(proxy)

Object.preventExtensions(proxy)
```

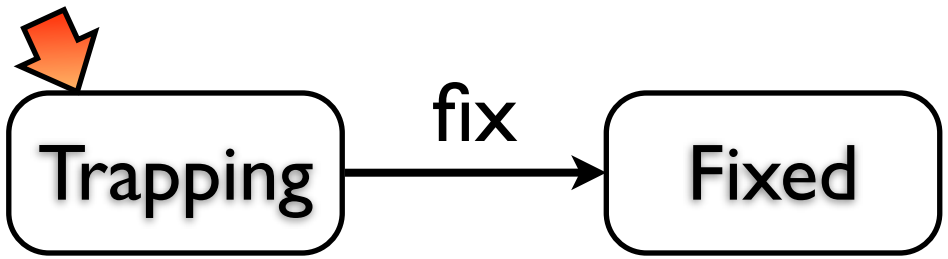
Fixing a Proxy (temporary intercession)

```
var proxy = Proxy.create(handler, proto);
```

```
var desc = handler.fix();
if (desc === undefined) throw TypeError();
become(proxy, Object.freeze(
          Object.create(proto, desc)));
```




```
Object.freeze(proxy)
Object.seal(proxy)
Object.preventExtensions(proxy)
```



Fixing a Proxy (temporary intercession)

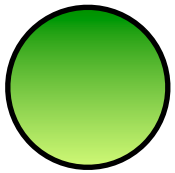
```
var proxy = Proxy.create(handler, proto);
```

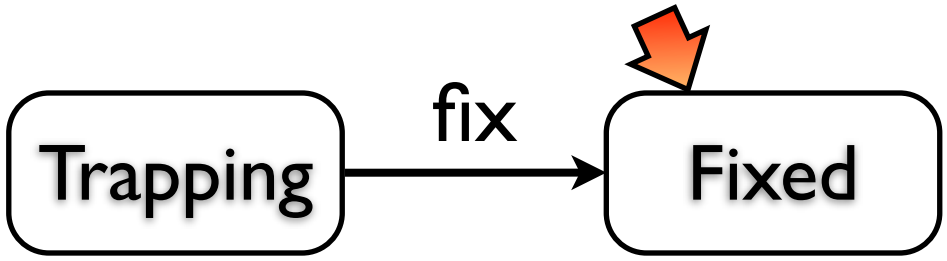
```
var desc = handler.fix();
if (desc === undefined) throw TypeError();
become(proxy, Object.freeze(
          Object.create(proto, desc)));
```

meta

base

```
Object.freeze(proxy)
Object.seal(proxy)
Object.preventExtensions(proxy)
```





Status

- Accepted proposal for ES-Harmony
- Draft specification at tinyurl.com/harmony-proxies
- Andreas Gal (Mozilla) prototyped an implementation in Tracemonkey

Status

- Available in Firefox 4 Betas today:

Summary

- Dynamic proxies enable:
  - Generic wrappers: access control, profiling, adaptors, ...
  - Virtual objects: remote objects, mock objects, emulated host objects, ...
- API:
  - Robust (stratified, not all operations intercepted)
  - Secure (can't trap non-proxy or fixed objects)
  - Performance: no overhead for non-proxy objects

es-lab.googlecode.com