Javascript’s Meta-object Protocol

Tom Van Cutsem

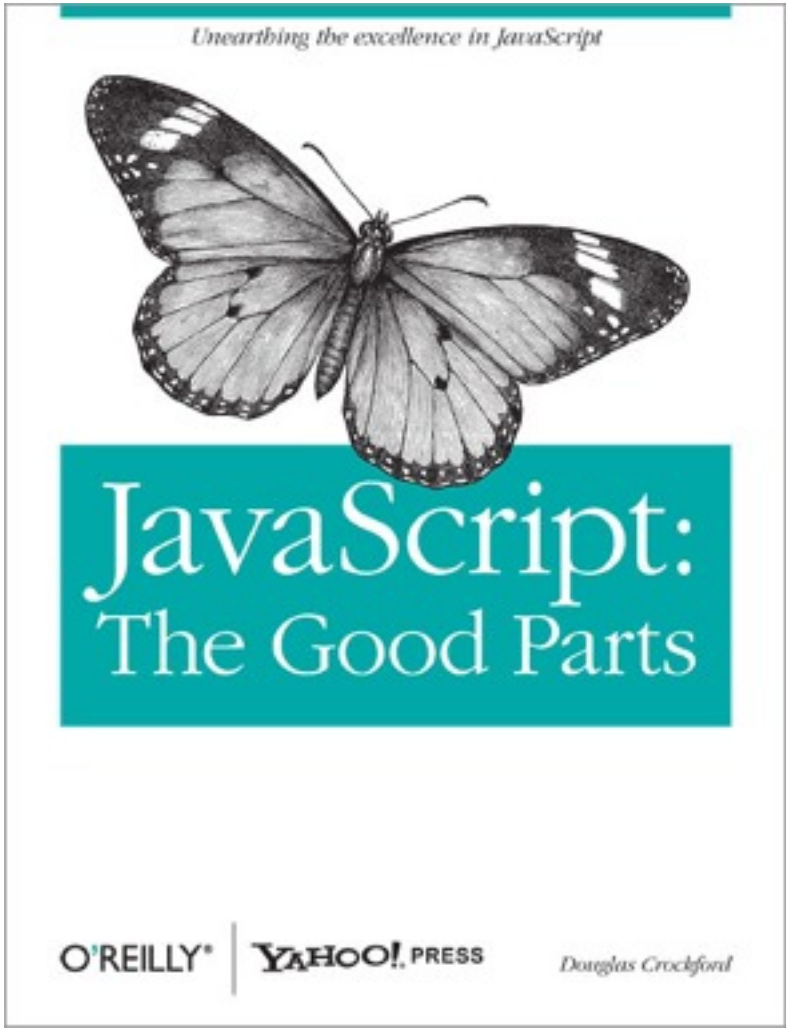
Software Languages.Lab

Vrije Universiteit Brussel

Talk Outline

- Brief walkthrough of Javascript
- Proxies in ECMAScript 6
- Meta-object protocols
- How proxies make Javascript’s MOP explicit
- Example: membranes

The world’s most misunderstood language



See also: “JavaScript: The World's Most Misunderstood Programming Language” by Doug Crockford at http://www.crockford.com/javascript/javascript.html

Good Parts

- Functions (closures, higher-order, first-class)

```
var add = function(a,b) {
  return a+b;
}

add(2,3);
```

```
function makeAdder(a) {
  return function(b) {
    return a+b;
  }
}

makeAdder(2)(3);
```

```
[1,2,3].map(function (x) { return x*x; })
node.addEventListener(‘click’, function (e) { clicked++; })
```

Good Parts

- Objects (no classes, literal syntax, arbitrary nesting)

```
var bob = {
  name: “Bob”,
  dob: {
    day: 15,
    month: 03,
    year: 1980
  },
  address: {
    street: “...”,
    number: 5,
    zip: 94040,
    country: “...”
  }
};
```

```
function makePoint(i,j) {
  return {
    x: i,
    y: j,
    toString: function() {
      return ‘(’+ this.x +‘,’+ this.y +‘)’;
    }
  };
}

var p = makePoint(2,3);
var x = p.x;
var s = p.toString();
```

A dynamic language...

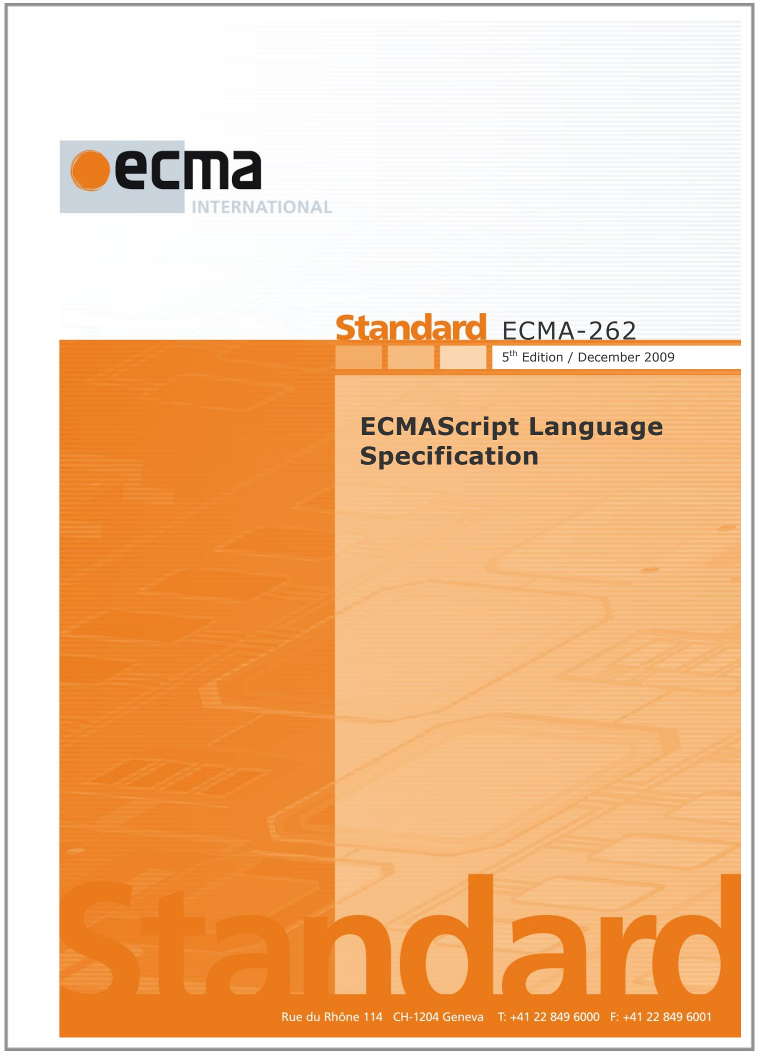
```
// computed property access and assignment
obj[“foo”]
obj[“foo”] = 42;

// dynamic method invocation
var f = obj.m;
f.apply(obj, [1,2,3]);

// enumerate an object’s properties
for (var prop in obj) { console.log(prop); }

// dynamically add new properties to an object
obj.bar = baz;

// delete properties from an object
delete obj.foo;
```

Bad Parts

- Dependence on global variables
- “var hoisting”: variables are not block-scoped
- `with`-statement breaks static scoping
- Automatic semicolon insertion
- Implicit type coercion
- …

ECMAScript

- "Standard" Javascript
  - 1st ed. 1997
  - 2nd ed. 1998
  - 3rd ed. 1999
  - 5th ed. 2009
  - *[6th ed. end of 2013 (tentative)]*



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

Functions

- Functions are objects

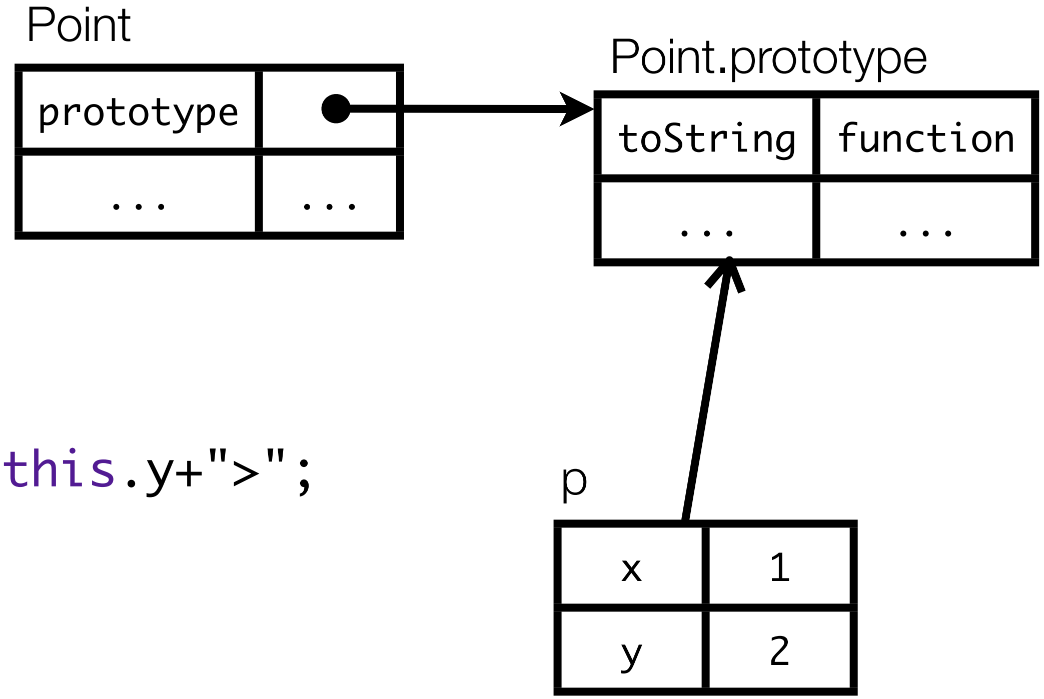
```
function add(x,y) { return x + y; }
add(1,2) // 3

add.apply(undefined, [1,2]) // 3
```

Objects

- No classes. Prototypes.
- Functions may act as object constructors.
- All objects have a “prototype”: object-based inheritance

Objects

```
function Point(x, y) {
  this.x = x;
  this.y = y;
}

Point.prototype = {
  toString: function() {
    return "<Point "+this.x+","+this.y+">";
  }
}

var p = new Point(1,2);
```



Functions / Methods

- Methods of objects are just functions

- When a function is called “as a method”, `this` is bound to the receiver object

```
var obj = {
  offset: 10,
  index: function(x) { return this.offset + x; }
}

obj.index(0); // 10
```

Functions / Methods

- Methods may be “extracted” from objects and used as stand-alone functions

```
var obj = {
  offset: 10,
  index: function(x) { return this.offset + x; }
}

var indexf = obj.index;

otherObj.index = indexf;

indexf(0) // error

indexf.apply(obj, [0]) // 10
```

Functions / Methods

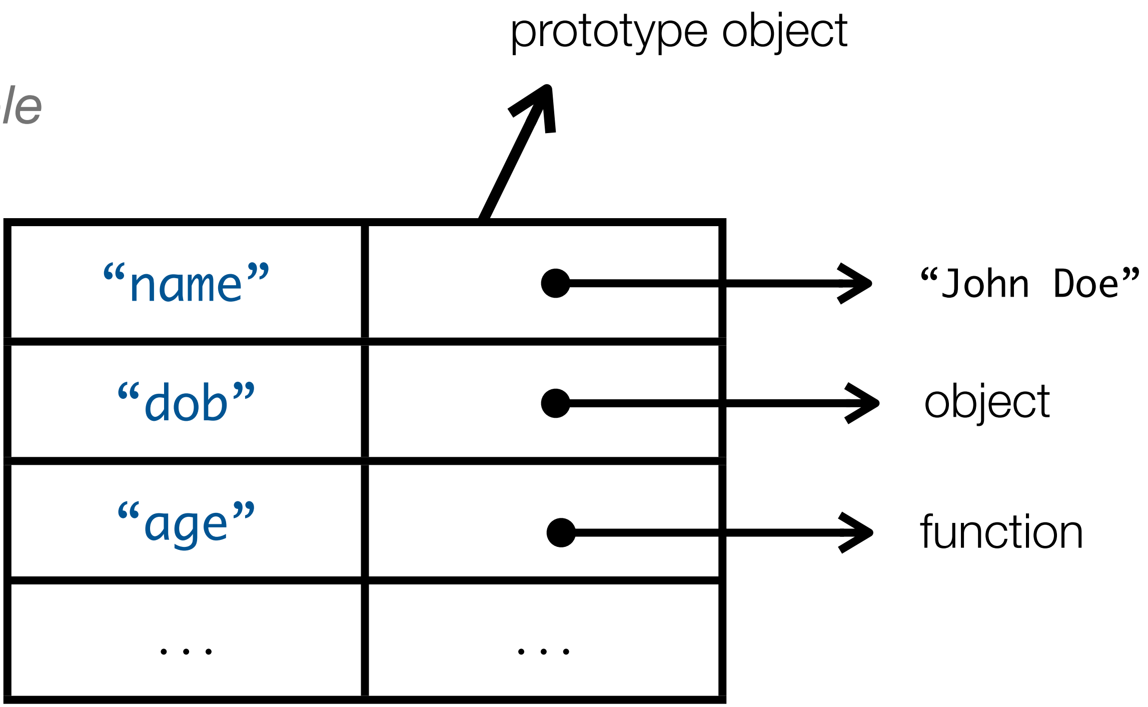
- Methods may be “extracted” from objects and used as stand-alone functions

```
var obj = {
  offset: 10,
  index: function(x) { return this.offset + x; }
}

var indexf = obj.index.bind(obj);

indexf(0) // 10
```

Javascript’s Object Model

- A Javascript object is a map of strings -> values + a prototype pointer
- Just a first approximation:
  - properties have hidden *attributes*
  - objects can be *non-extensible*

```
{ name: “John Doe”,
   dob: {...},
   age: function(){...},
   ...  };
```



Property attributes

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

```
Object.getOwnPropertyDescriptor(point, ‘x’);
```

```
  { value: 0,
    writable: true,
    enumerable: true,
    configurable: true }
```

Property attributes

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

```
Object.getOwnPropertyDescriptor(point, ‘x’);
  { value: 0,
    writable: true,
    enumerable: true,
    configurable: true }
```

```
Object.defineProperty(point, ‘x’,
  { value: 1,
    writable: false,
    enumerable: false,
    configurable: false });
```

Tamper-proof Objects

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

```
Object.preventExtensions(point);
point.z = 0; // error: can’t add new properties
```

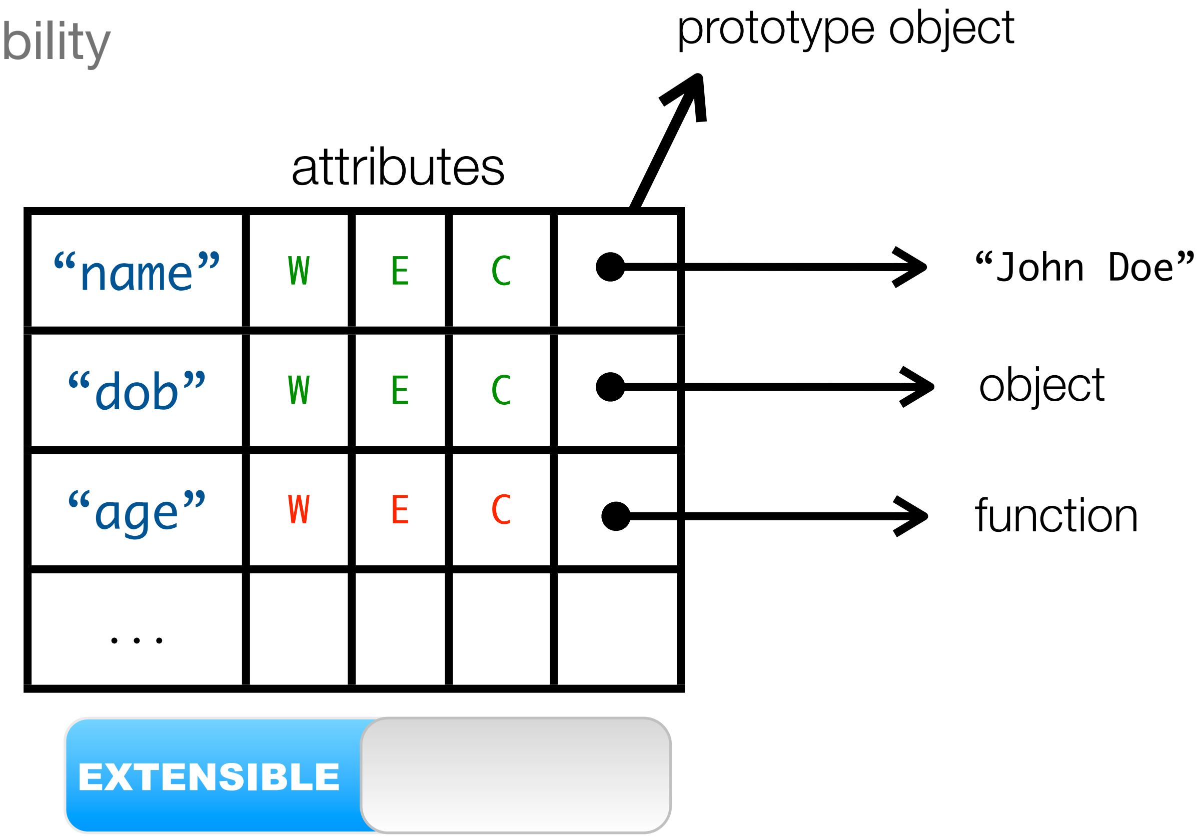
```
Object.seal(point);
delete point.x; // error: can’t delete properties
```

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

Javascript’s Object Model Revisited

- A Javascript object is a map of strings -> *property descriptors*
  - + a prototype pointer
  - + a flag indicating extensibility

```
{ name: “John Doe”,
   dob: {...},
   age: function(){...},
   ...  };
```



	attributes			
“name”	W	E	C	●
“dob”	W	E	C	●
“age”	W	E	C	●
...				

Host objects

- Objects provided by the host platform
- E.g. the **DOM**: a tree representation of the HTML document
- “look and feel” like Javascript objects, but are not implemented in Javascript (typically in C++)
- Odd behavior not always easy to faithfully emulate by wrapper libraries

```
var links = document.getElementsByTagName('a');
links.length // 2
document.body.appendChild(newLink);
links.length // now 3
```

Summary so far

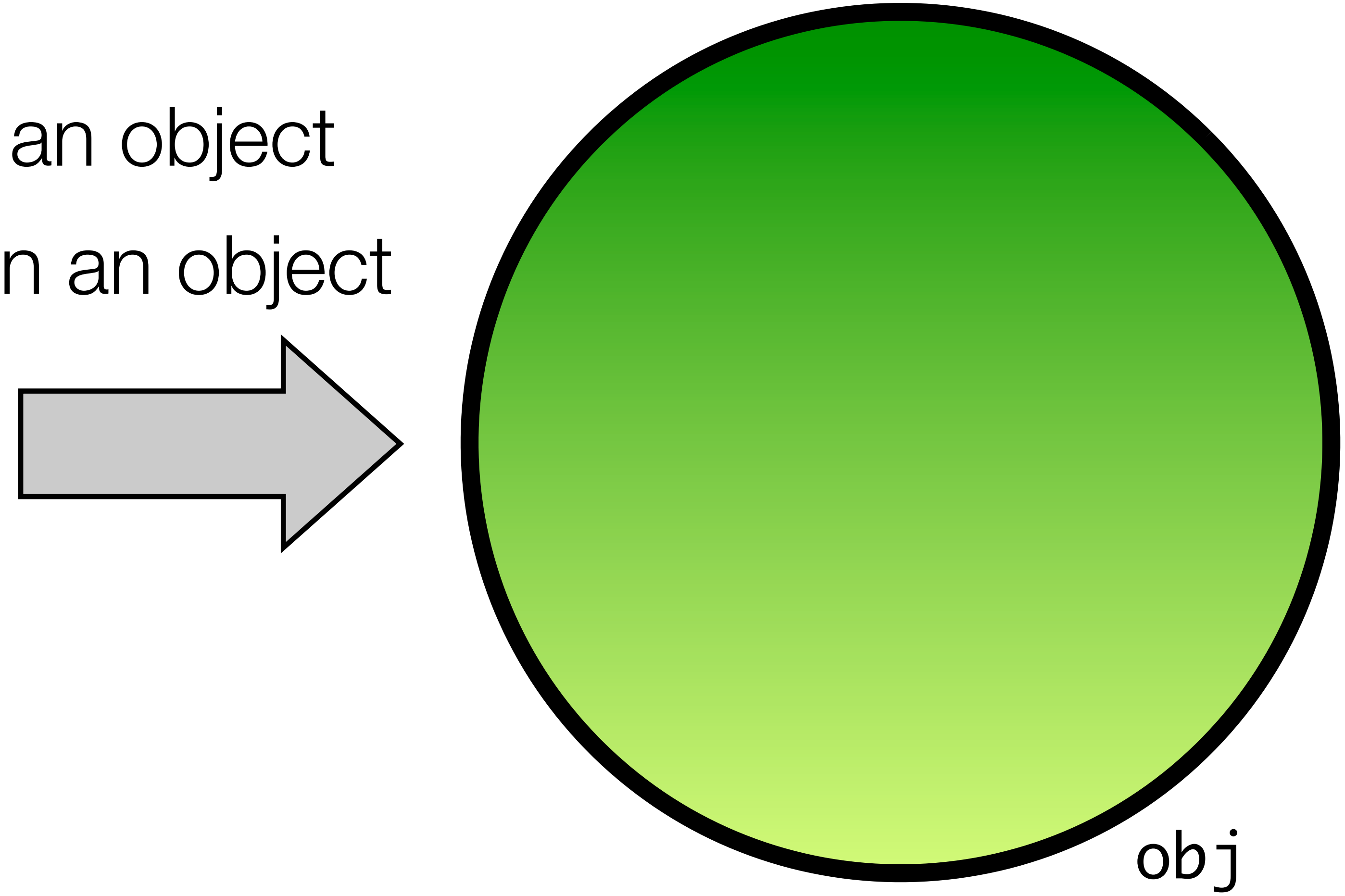
- Dynamic language, “Lisp in C’s clothing”
    - First-class functions, closures
    - Flexible objects, prototype-based inheritance
    - Beware of and avoid the “bad parts”
- Javascript object = property map + prototype link + extensible flag
- Javascript scripts interact with “host objects”

The Javascript MOP & Proxies

Introspection vs. Intercession

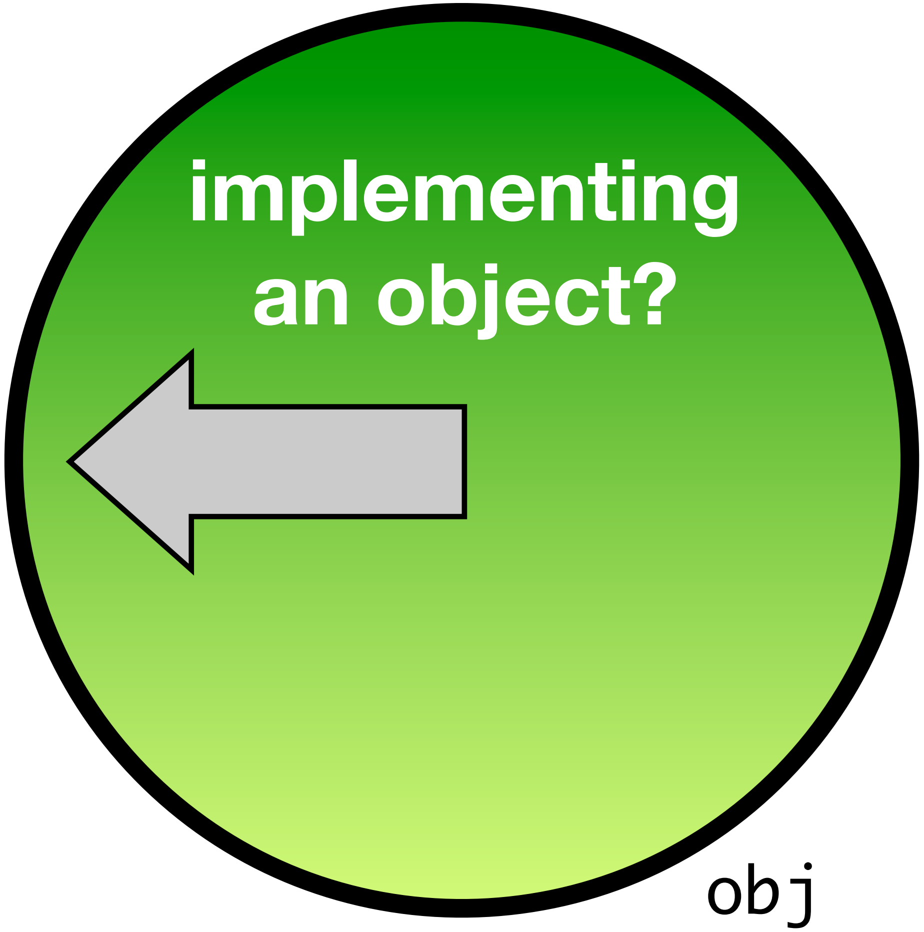
querying an object

acting upon an object



```
obj[“x”]
delete obj.x;
Object.getOwnPropertyDescriptor(obj,‘x’);
```

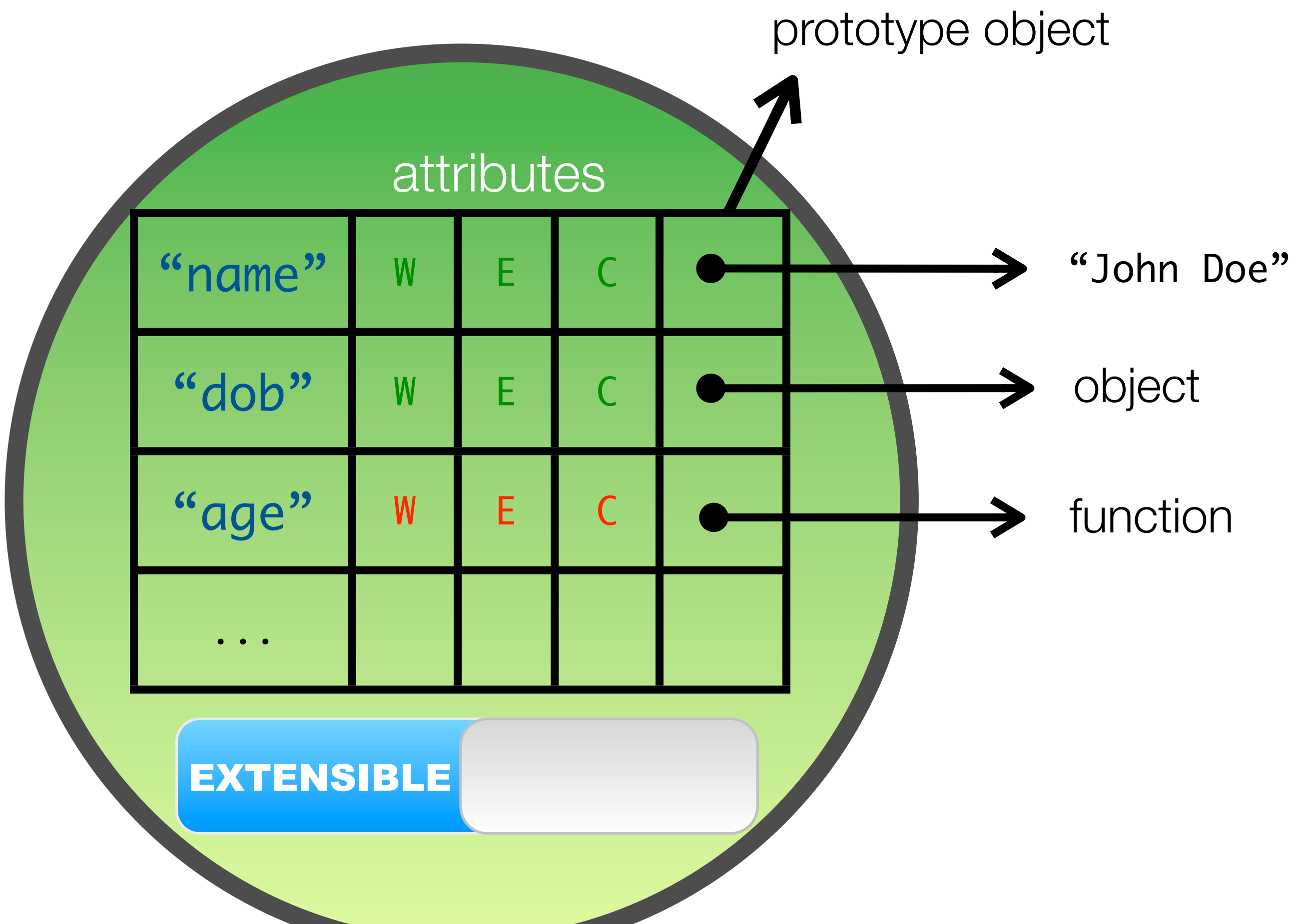
Introspection vs. Intercession



```
obj[“x”]
delete obj.x;
Object.getOwnPropertyDescriptor(obj,‘x’);
```

Javascript's Object Model: Recap

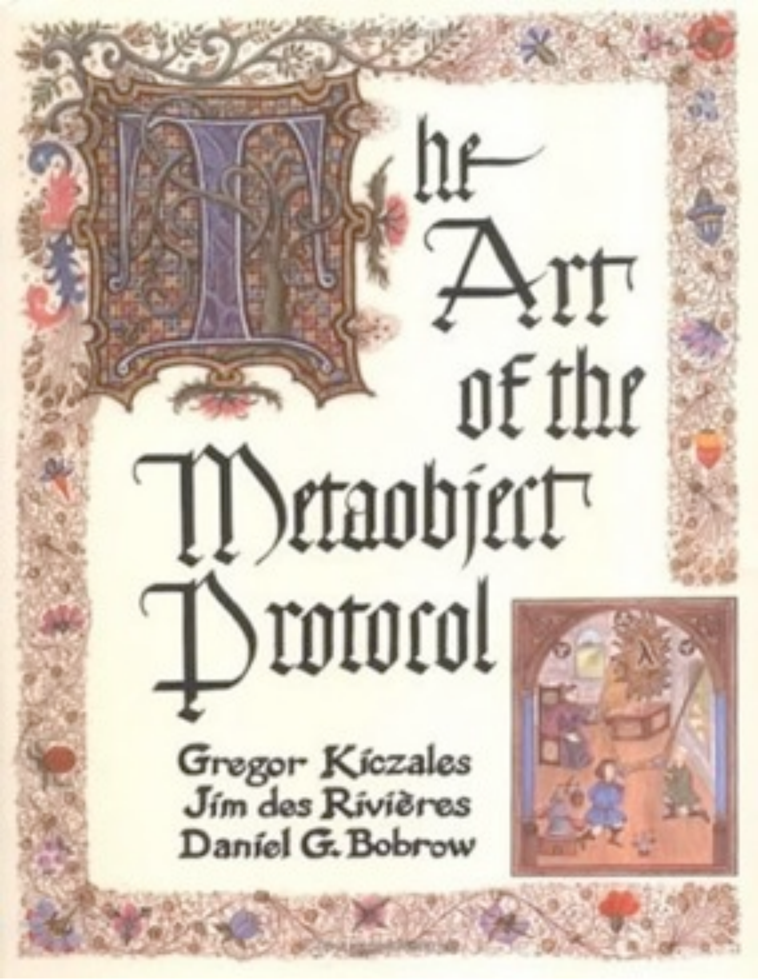
- The object model defines the “interface” of an object
- Any “sensible” implementation of this model is a valid Javascript object



“name”	W	E	C	●
“dob”	W	E	C	●
“age”	W	E	C	●
...				

Meta-object protocols

- Kiczales, early ‘90s
- Using OOP to structure the meta-level
- Common Lisp Object System (CLOS)
- Precursor to AOP

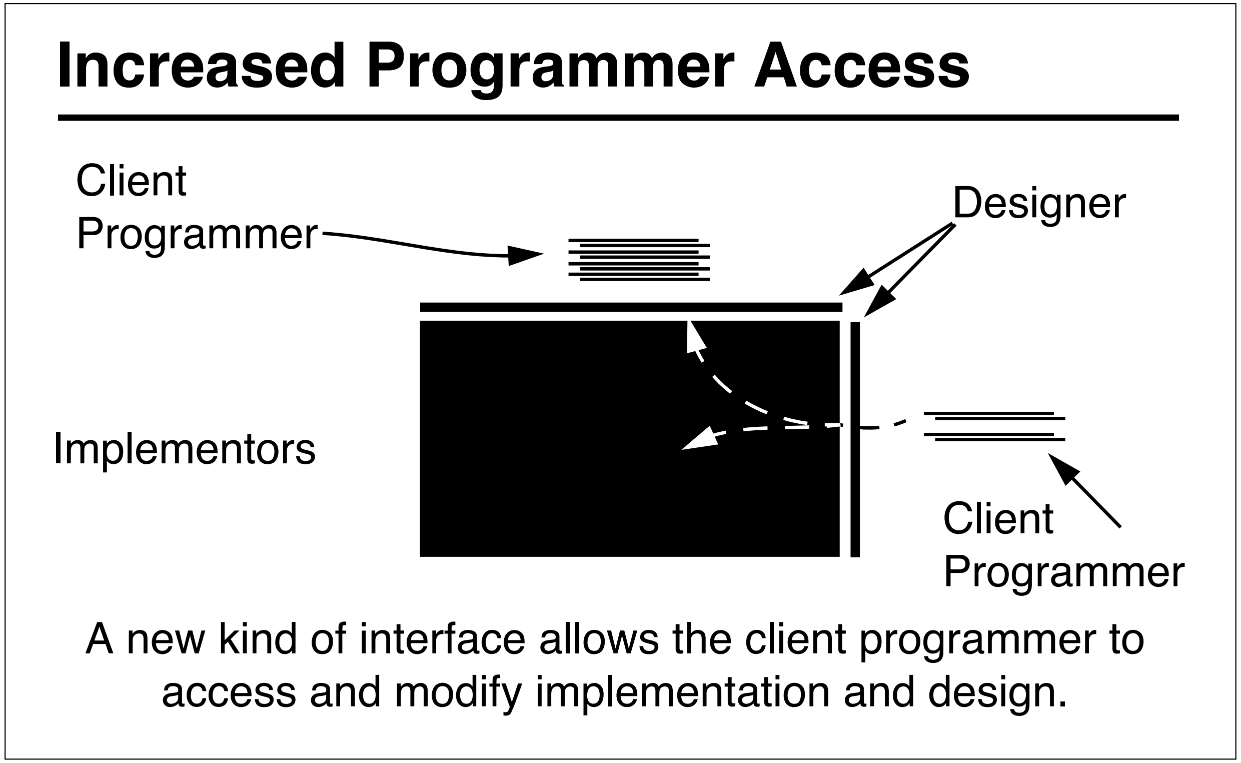


“Open implementations” philosophy

- Kiczales & Paepcke, early ‘90s

“Open implementations” philosophy

- Kiczales & Paepcke, early ‘90s



Increased Programmer Access

A new kind of interface allows the client programmer to access and modify implementation and design.

(Kiczales & Paepcke, Open Implementations & Metaobject Protocols)

Why implement your own object?

- Many use cases:
  - Generic wrappers around existing objects: access control wrappers (security), tracing, profiling, contracts, tainting, decorators, adaptors, ...
  - Virtual objects: remote objects, mock objects, persistent objects, futures, lazy initialization, ...

ECMAScript 5 does not support this

- ECMAScript 5 reflection API:
  - powerful control over structure of objects
  - limited control over behavior of objects
- Can’t intercept method calls, property access, ...
- Can’t implement ‘virtual’ properties

Limited intercession in some implementations

- non-standard `__noSuchMethod__` hook in Firefox
- modelled after Smalltalk’s `doesNotUnderstand:` method

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

__noSuchMethod__

- not “stratified” (part of base-level object interface)
- 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’
}
```

Proxies

- Objects that “look and feel” like normal objects, but whose behavior is controlled by *another* Javascript object
- Part of a new reflection API for ECMAScript 6
- Think `java.lang.reflect.Proxy` on steroids

Example: tracing

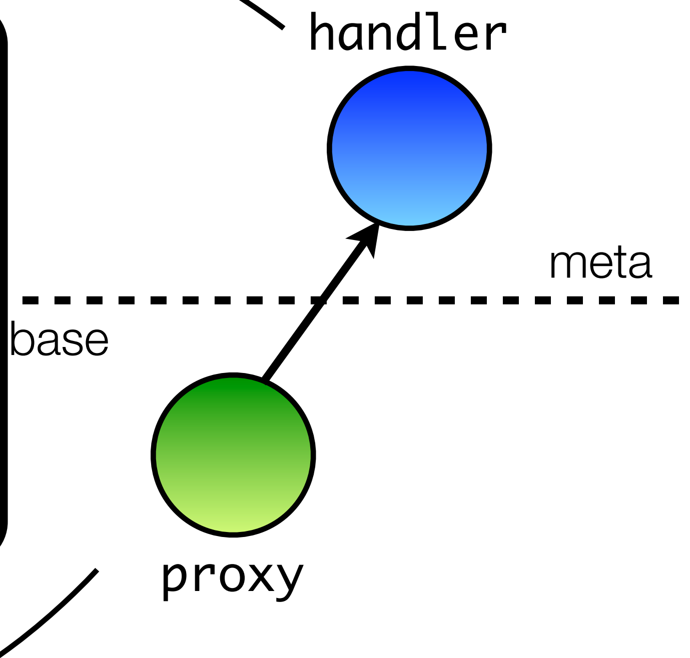
```
function makePoint(x, y) {
  return {
    x: x,
    y: y
  };
}


var p = makePoint(2,2);
var tp = makeTracer(p);
tp.x
// log(p, ‘get’, ‘x’);
// 2
tp.y = 3
// log(p, ‘set’, ‘y’, 3);
// 3
```

Example: tracing

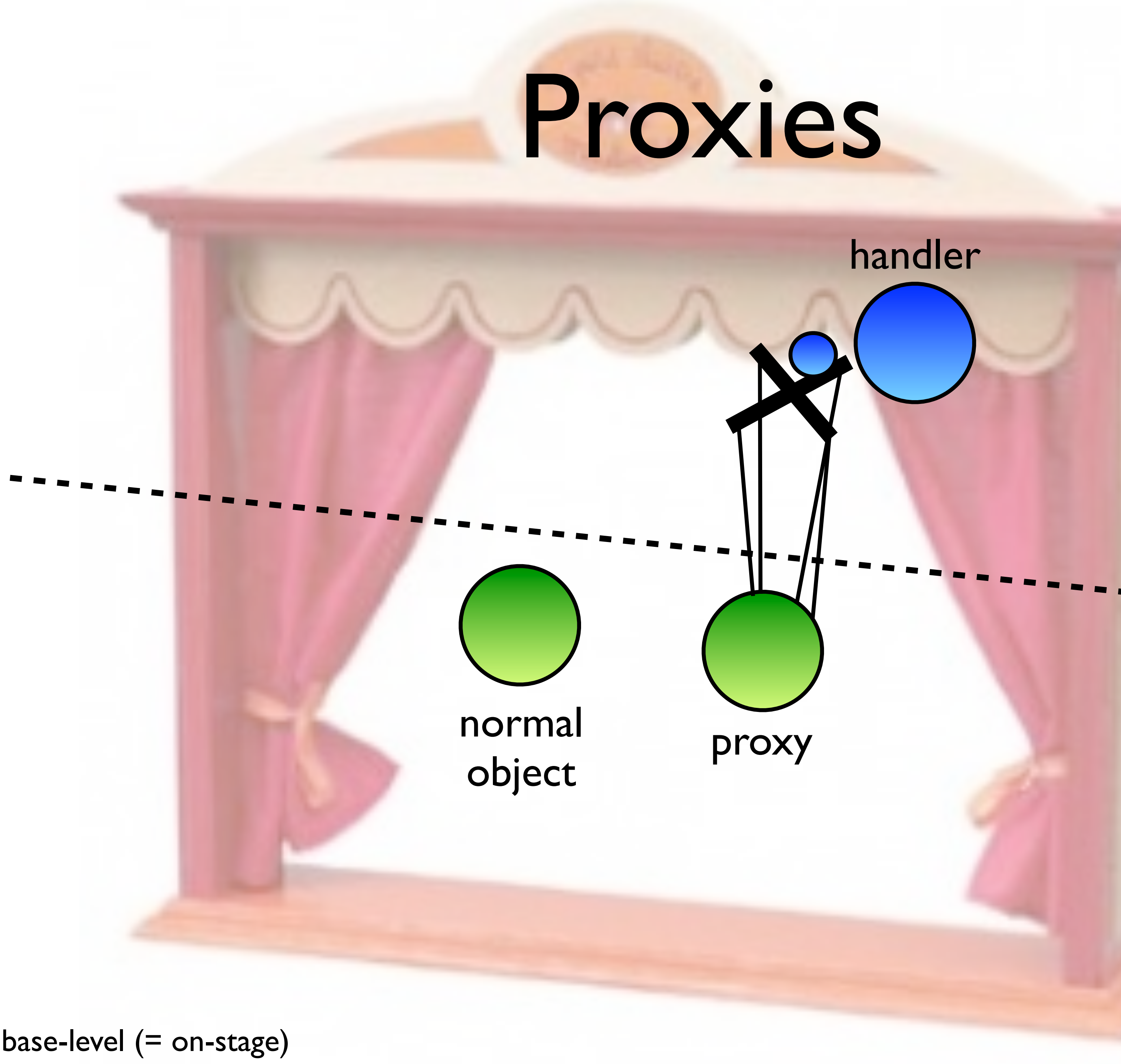
```
function makeTracer(obj) {
  var proxy = Proxy(obj, {
    get: function(tgt, name, rcvr) {
      console.log(tgt, ‘get’, name);
      return Reflect.get(tgt, name, rcvr);
    },
    set: function(tgt, name, val, rcvr) {
      console.log(tgt, ‘set’, name, val);
      return Reflect.set(tgt, name, val, rcvr);
    },
  });
  return proxy;
}
```

Example: tracing

```
function makeTracer(obj) {
  var proxy = Proxy(obj, {
    get: function(tgt, name, rcvr) {
      console.log(tgt, ‘get’, name);
      return Reflect.get(tgt, name, rcvr);
    },
    set: function(tgt, name, val, rcvr) {
      console.log(tgt, ‘set’, name, val);
      return Reflect.set(tgt, name, val, rcvr);
    },
  });
  return proxy;
}
```

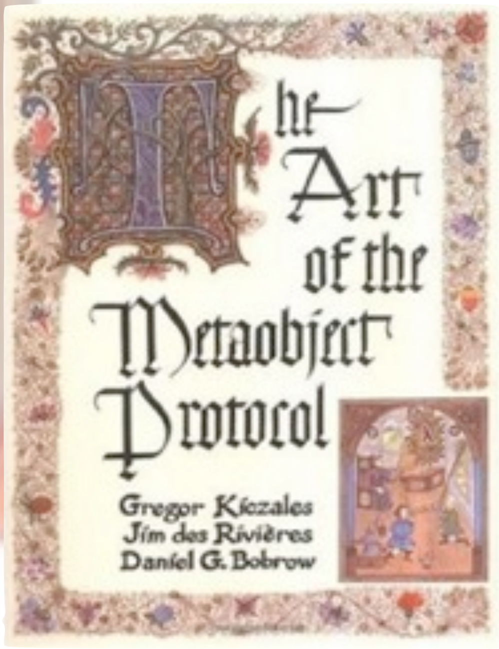


Proxies



meta-level (= backstage)

base-level (= on-stage)

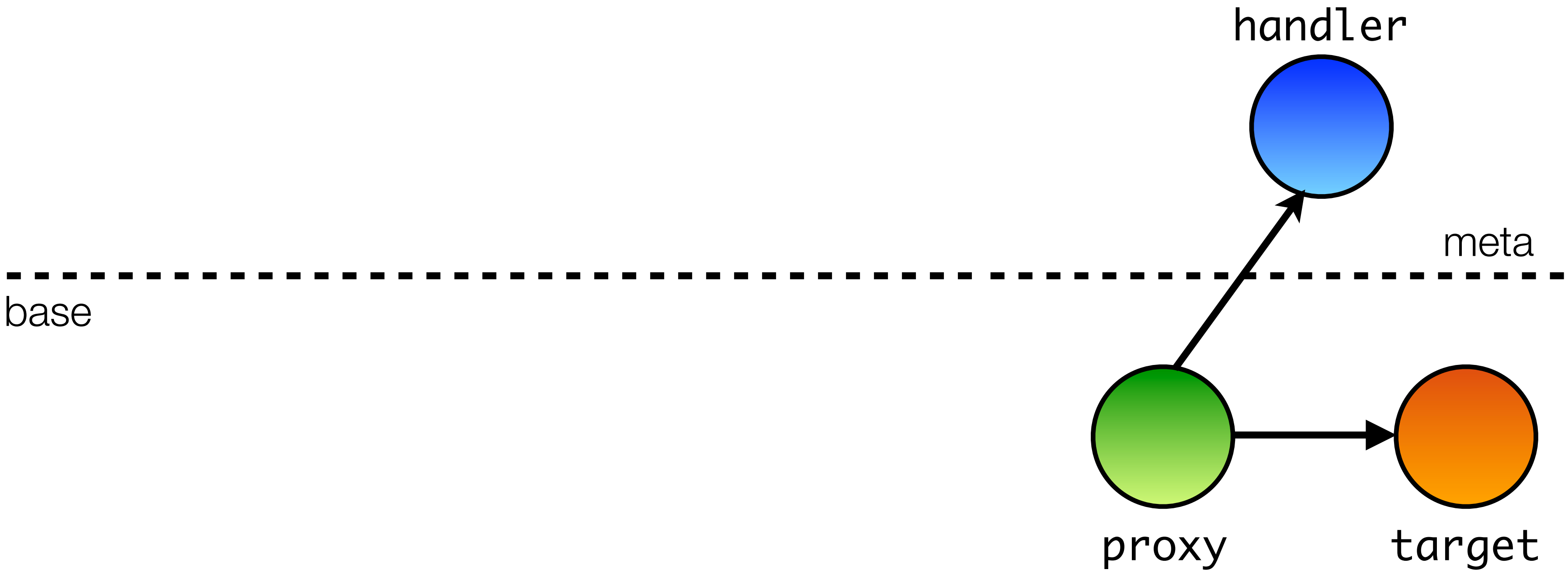


The Art of the Metaobject Protocol

Gregor Kiczales
Jim des Rivières
Daniel G. Bobrow

Stratified API

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



Stratified API

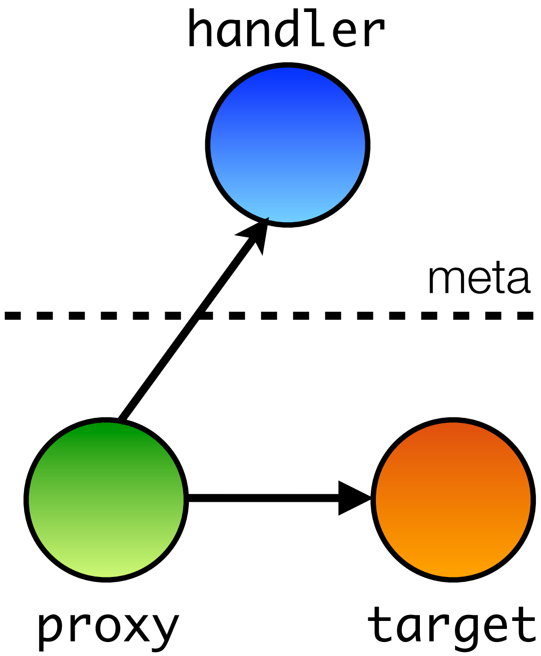
```
var proxy = Proxy(target, handler);
```

```
handler.get(target, ‘foo’)
```

meta

base

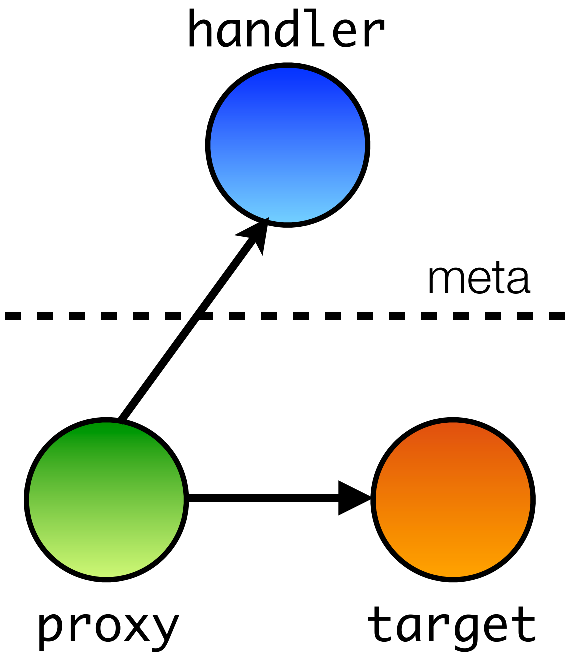
```
proxy.foo
```



Stratified API

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

```
handler.get(target, ‘foo’)
handler.set(target, ‘foo’, 42)
```



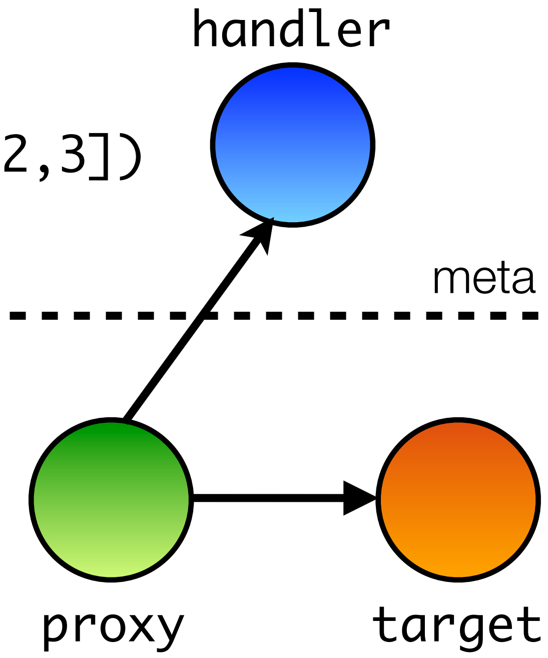
base

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

Stratified API

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

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



meta

base

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

Stratified API

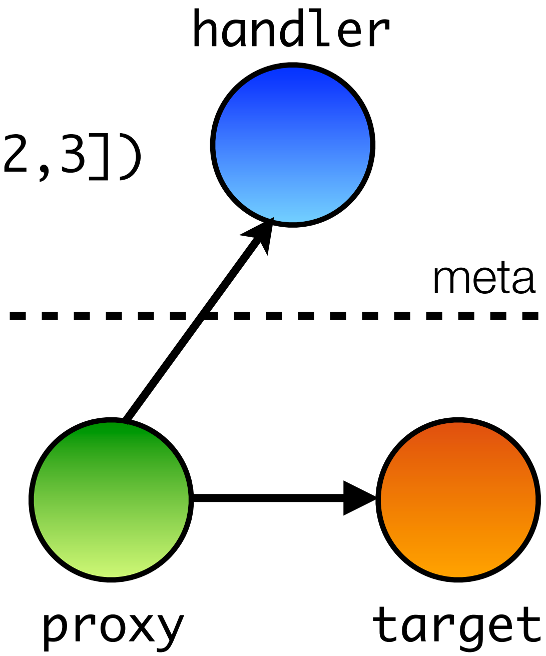
```
var proxy = Proxy(target, handler);
```

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

meta

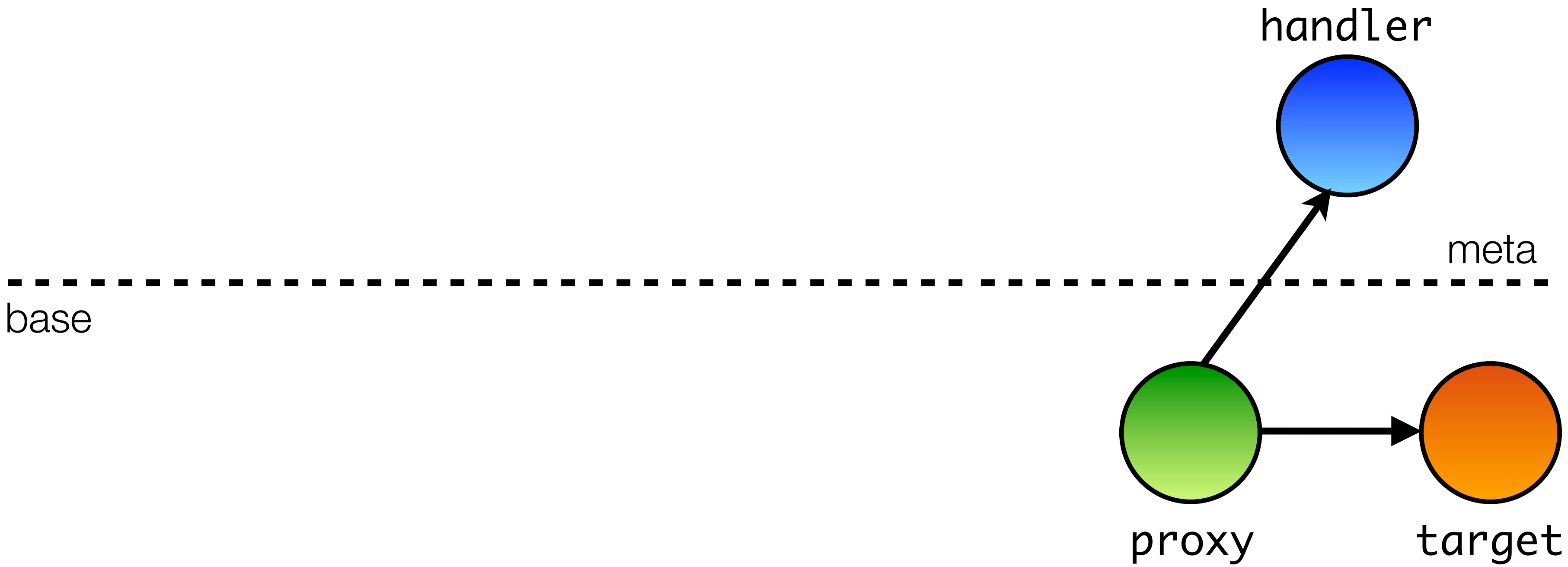
base

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



Not just property access

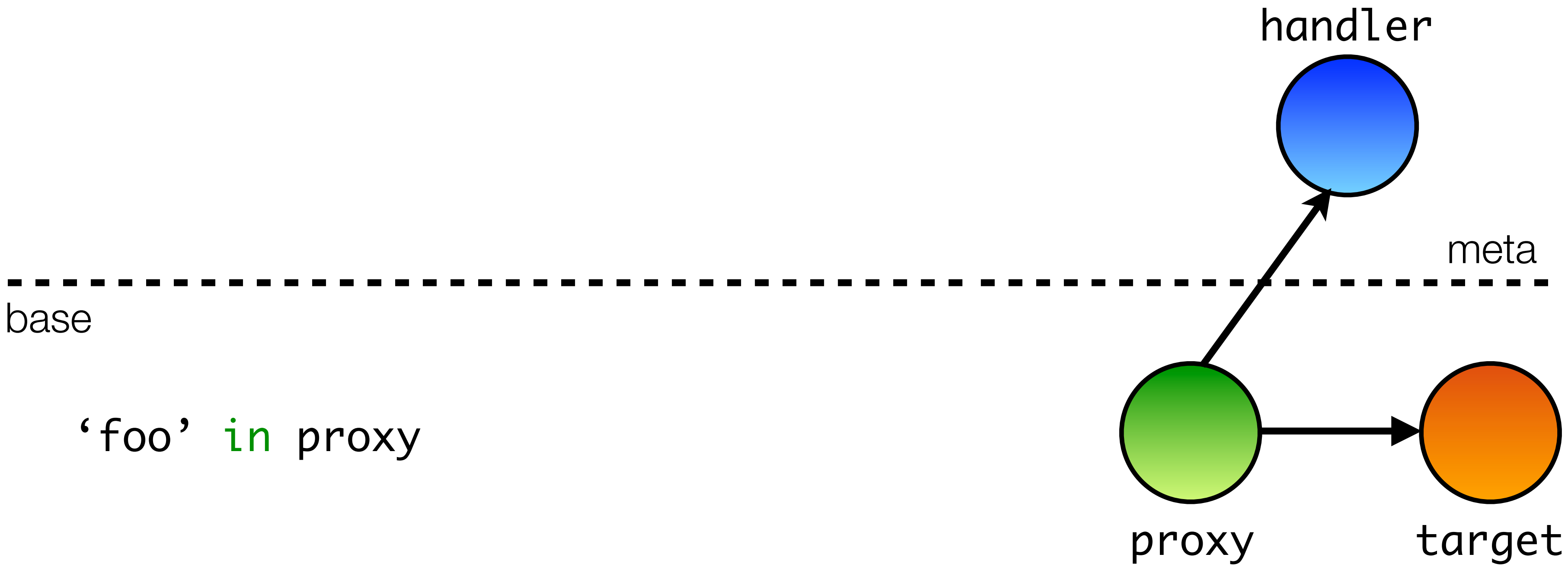
```
var proxy = Proxy(target, handler);
```



Not just property access

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

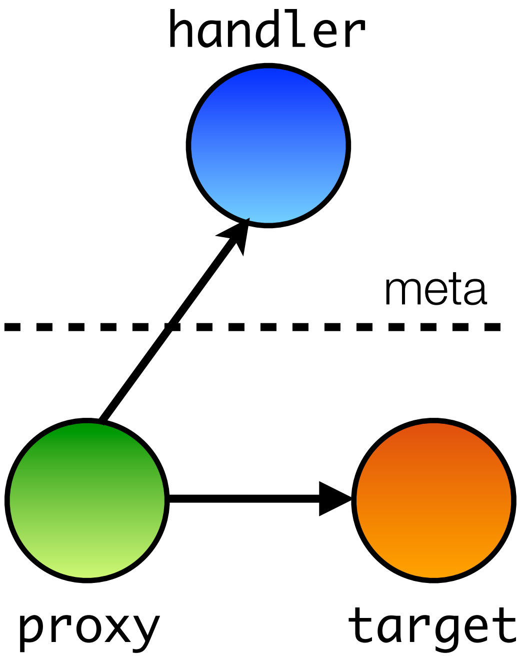
```
handler.has(target,‘foo’)
```



```
‘foo’ in proxy
```

Not just property access

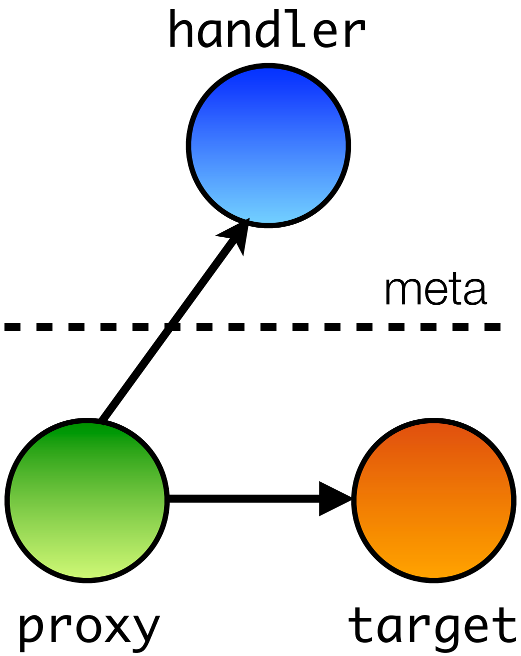
```
var proxy = Proxy(target, handler);

handler.has(target,‘foo’)

handler.deleteProperty(target,‘foo’)
```



```
‘foo’ in proxy

delete proxy.foo
```

Not just property access

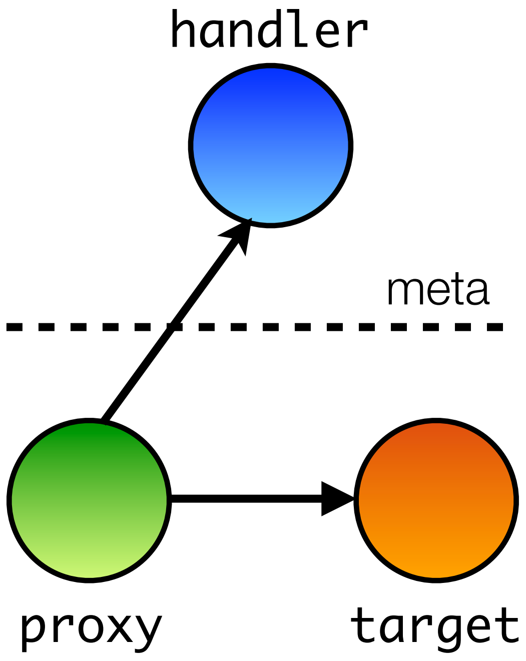
```
var proxy = Proxy(target, handler);


handler.has(target,‘foo’)

handler.deleteProperty(target,‘foo’)

var props = handler.enumerate(target);
for (var p in props) { ... }
```



```
‘foo’ in proxy

delete proxy.foo

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

Not just property access

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

handler.has(target,‘foo’)

handler.deleteProperty(target,‘foo’)

var props = handler.enumerate(target);
for (var p in props) { ... }

handler.defineProperty(target, ‘foo’, pd)
```

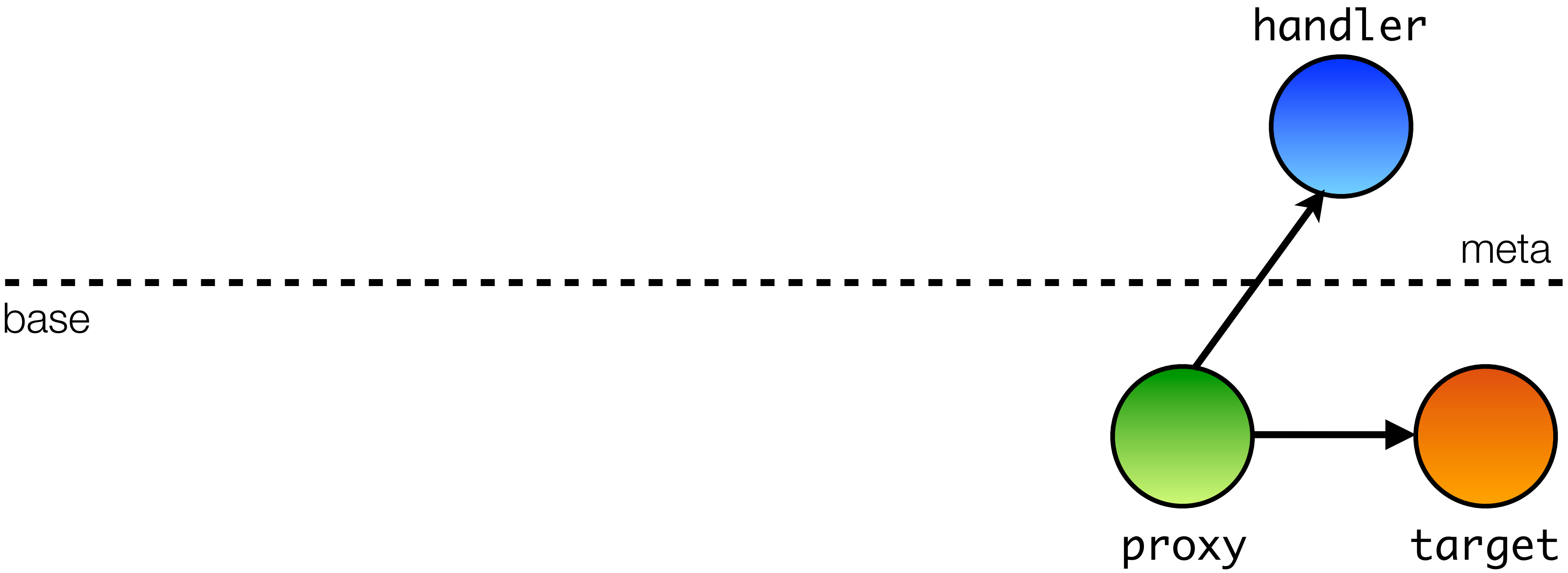


base

```
‘foo’ in proxy

delete proxy.foo

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

Object.defineProperty(proxy,‘foo’, pd)
```

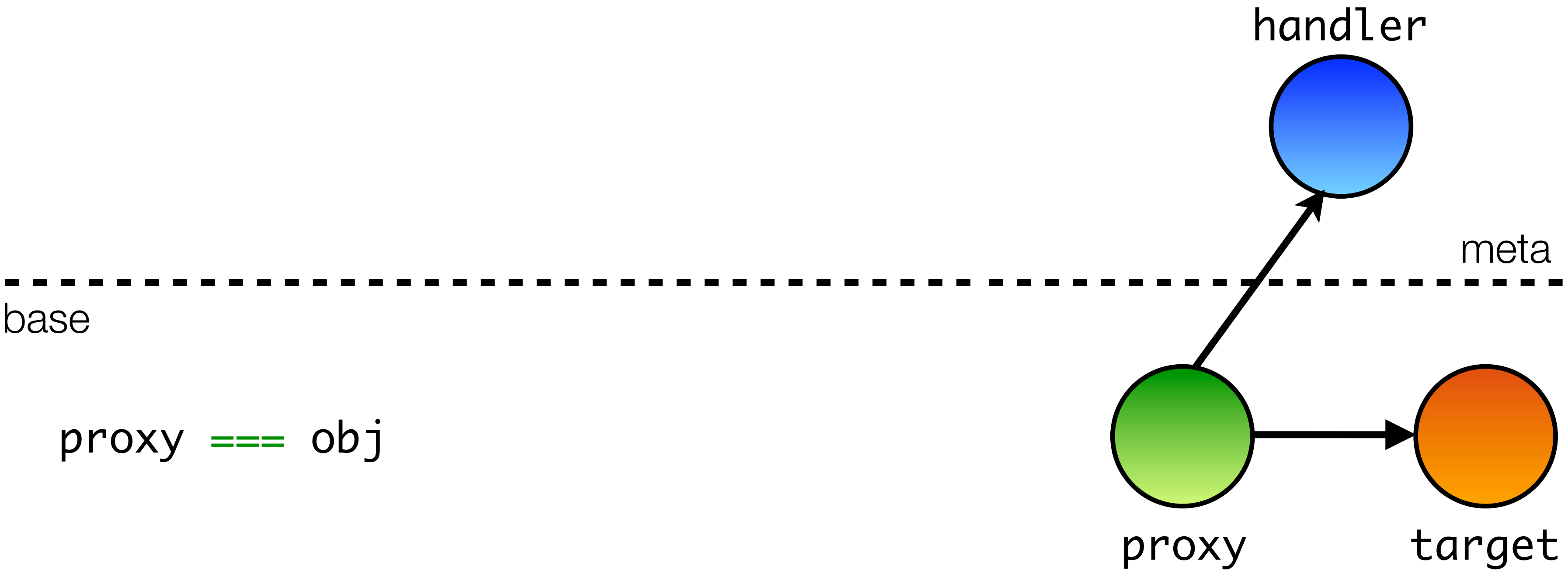
But not quite everything either

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



But not quite everything either

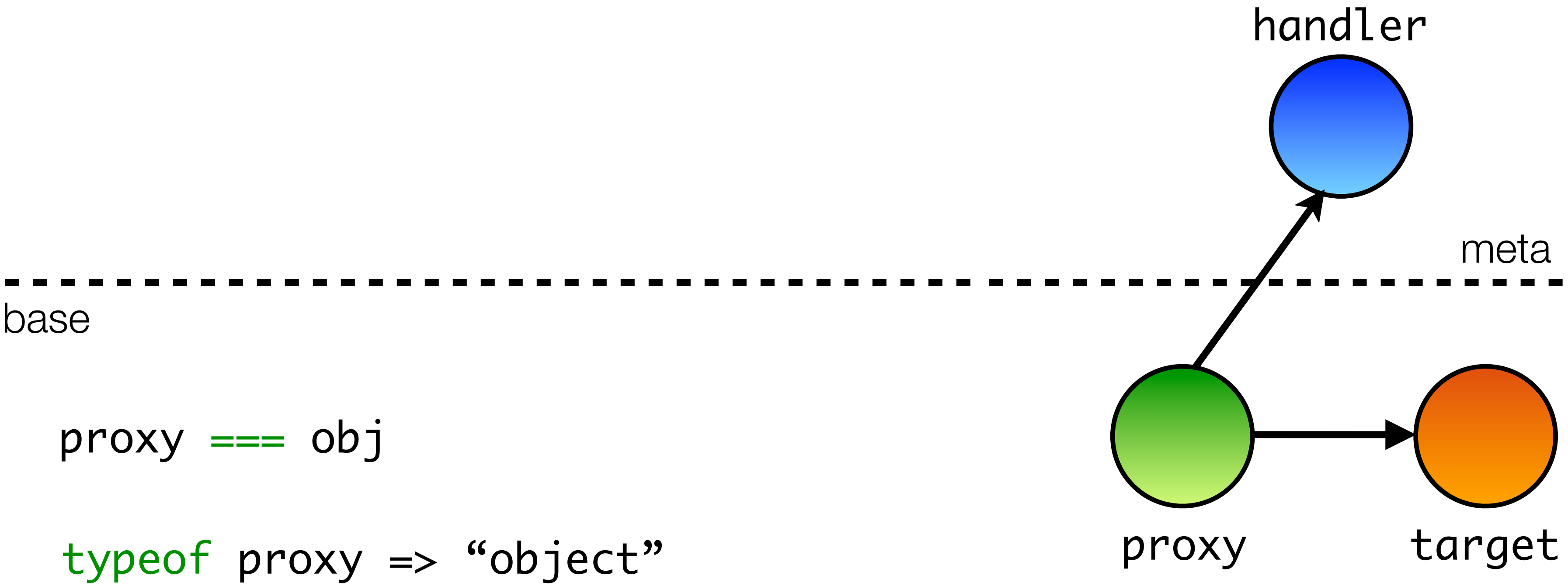
```
var proxy = Proxy(target, handler);
```



```
proxy === obj
```

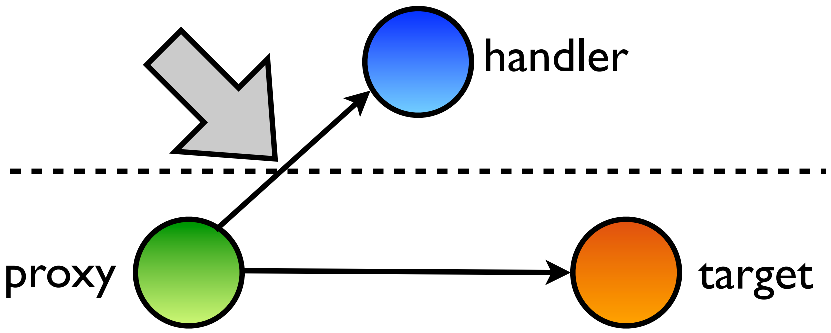
But not quite everything either

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



```
proxy === obj

typeof proxy => “object”
```

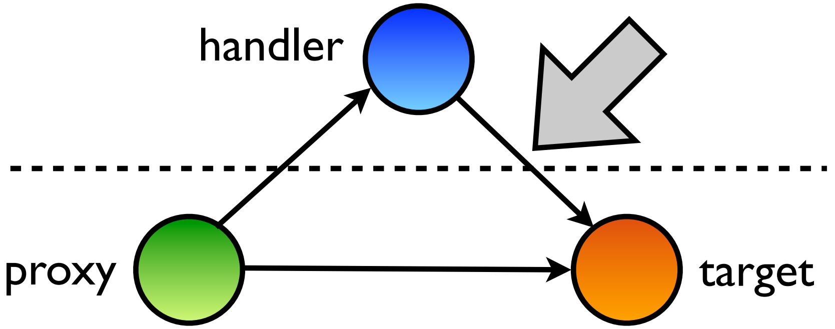
Full handler API (16 traps)



```
Object.getOwnPropertyDescriptor(proxy,name)
Object.defineProperty(proxy,name,pd)
Object.getOwnPropertyNames(proxy)
delete proxy.name
for (name in proxy) { ... }
for (name in Object.create(proxy)) { ... }
Object.{freeze|seal|preventExtensions}(proxy)
name in proxy
({}).hasOwnProperty.call(proxy, name)
Object.keys(proxy)
proxy.name
proxy.name = val
proxy(...args)
new proxy(...args)
```

```
handler.getOwnPropertyDescriptor(target,name)
handler.defineProperty(target,name,pd)
handler.getOwnPropertyNames(target)
handler.deleteProperty(target,name)
handler.iterate(target)
handler.enumerate(target)
handler.{freeze|seal|preventExtensions}(target)
handler.has(target,name)
handler.hasOwn(target,name)
handler.keys(target)
handler.get(target,name,receiver)
handler.set(target,name,value,receiver)
handler.apply(target,receiver,args)
handler.construct(target,args)
```

Reflect module



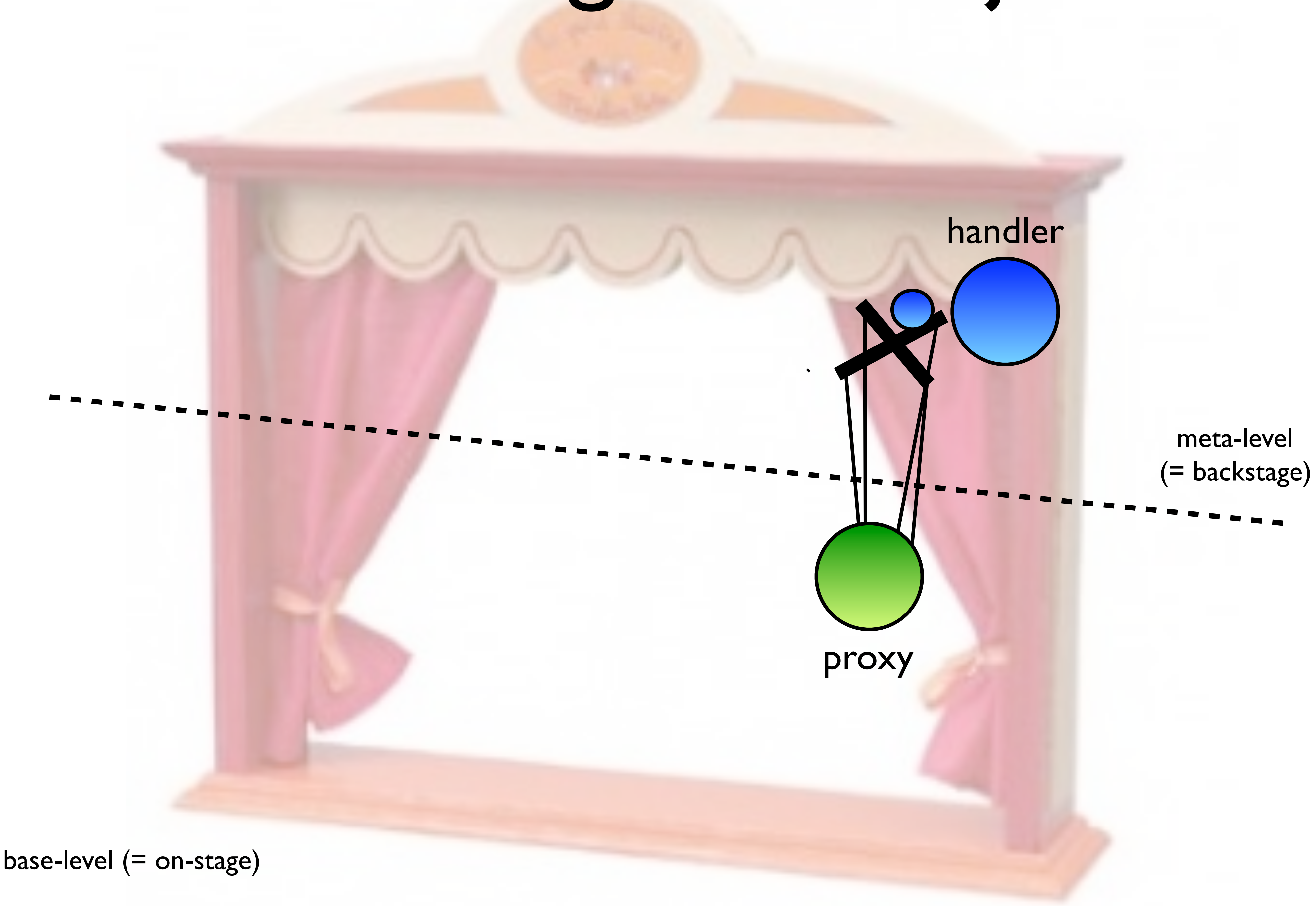
```
handler.getOwnPropertyDescriptor(target,name)
handler.defineProperty(target,name,pd)
handler.getOwnPropertyNames(target)
handler.deleteProperty(target,name)
handler.iterate(target)
handler.enumerate(target)
handler.{freeze|seal|preventExtensions}(target)
handler.has(target,name)
handler.hasOwn(target,name)
handler.keys(target)
handler.get(target,name,receiver)
handler.set(target,name,value,receiver)
handler.apply(target,receiver,args)
handler.construct(target,args)
```

```
Reflect.getOwnPropertyDescriptor(target,name)
Reflect.defineProperty(target,name,pd)
Reflect.getOwnPropertyNames(target)
Reflect.deleteProperty(target,name)
Reflect.iterate(target)
Reflect.enumerate(target)
Reflect.{freeze|seal|preventExtensions}(target)
Reflect.has(target,name)
Reflect.hasOwn(target,name)
Reflect.keys(target)
Reflect.get(target,name,receiver)
Reflect.set(target,name,value,receiver)
Reflect.apply(target,receiver,args)
Reflect.construct(target,args)
```

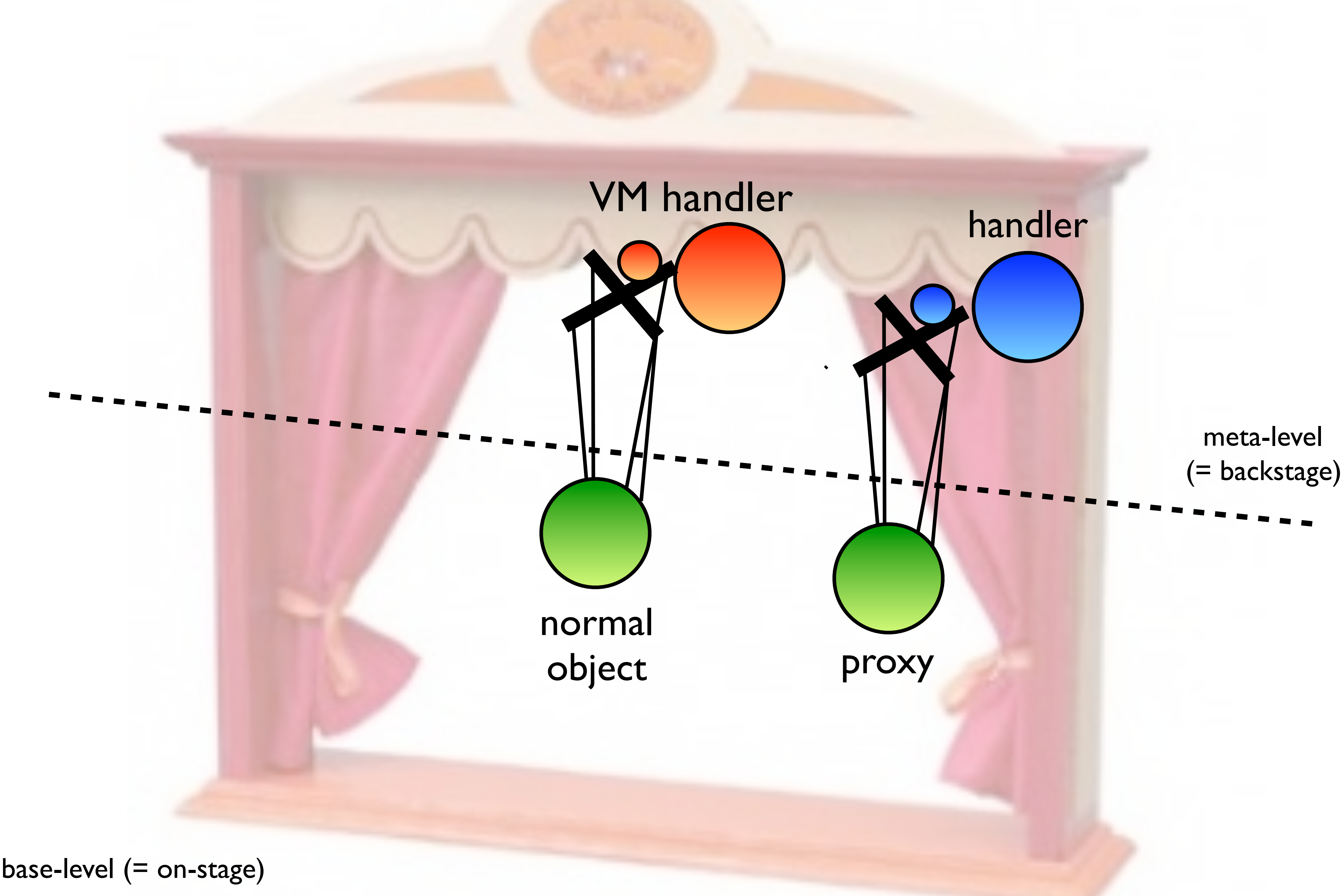
Example: profiling

```
function makeProfiler(target) {
  var count = new Map();
  return {
    proxy: Proxy(target, {
      get: function(target, name, receiver) {
        count.set(name, (count.get(name) || 0) + 1);
        return Reflect.get(target, name, receiver);
      }
    }),
    stats: count;
  }
}
```

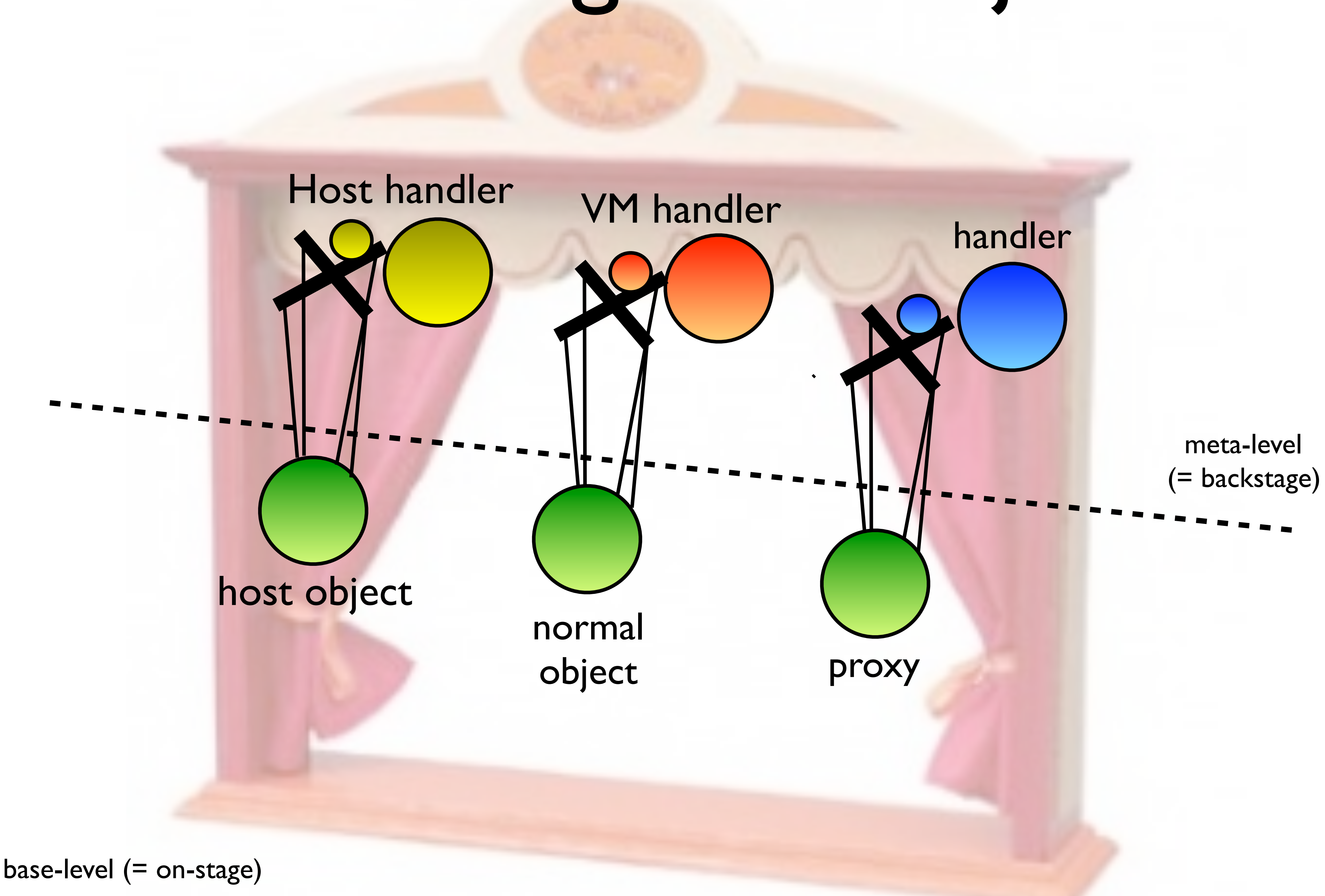
Retrofitting host objects



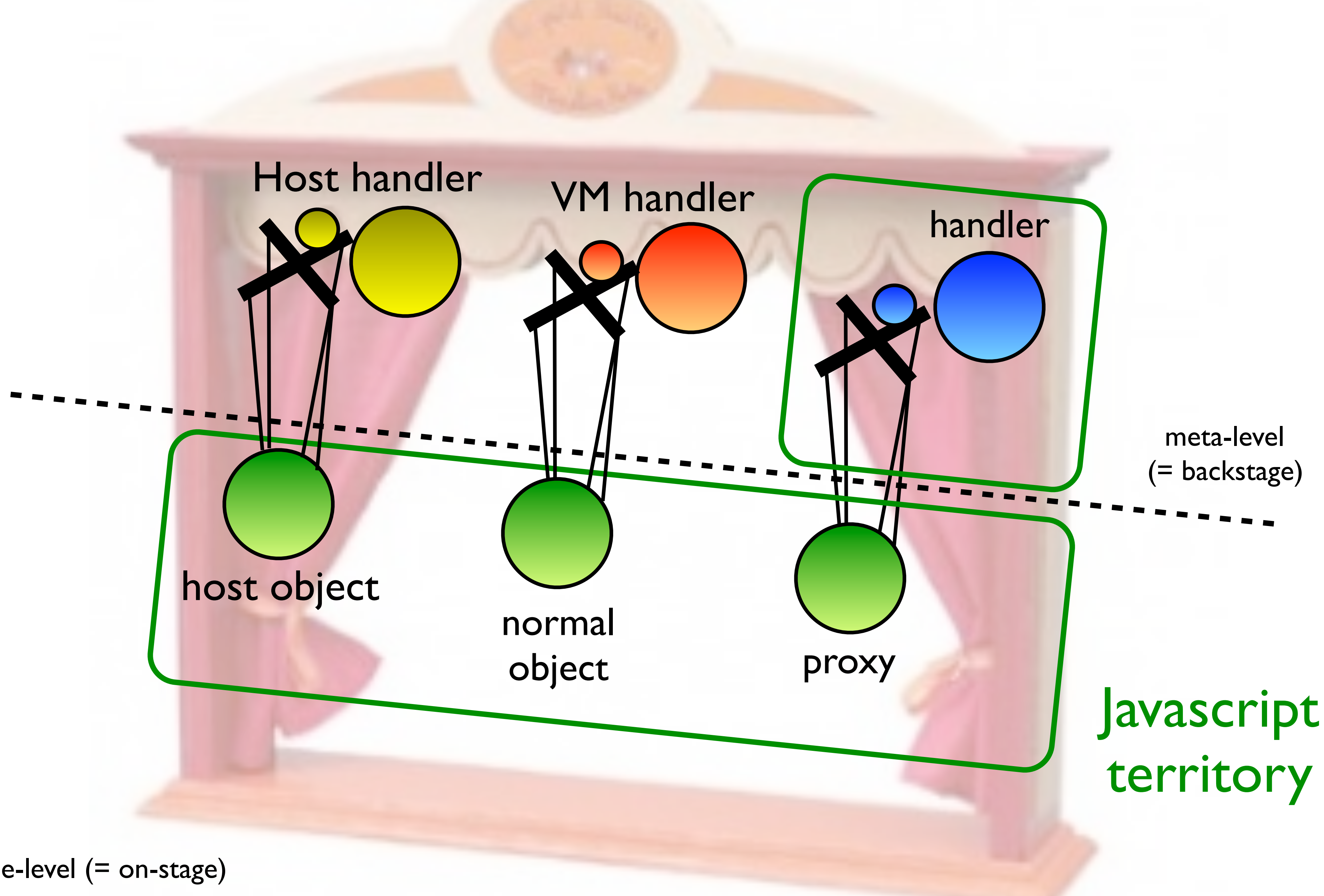
Retrofitting host objects



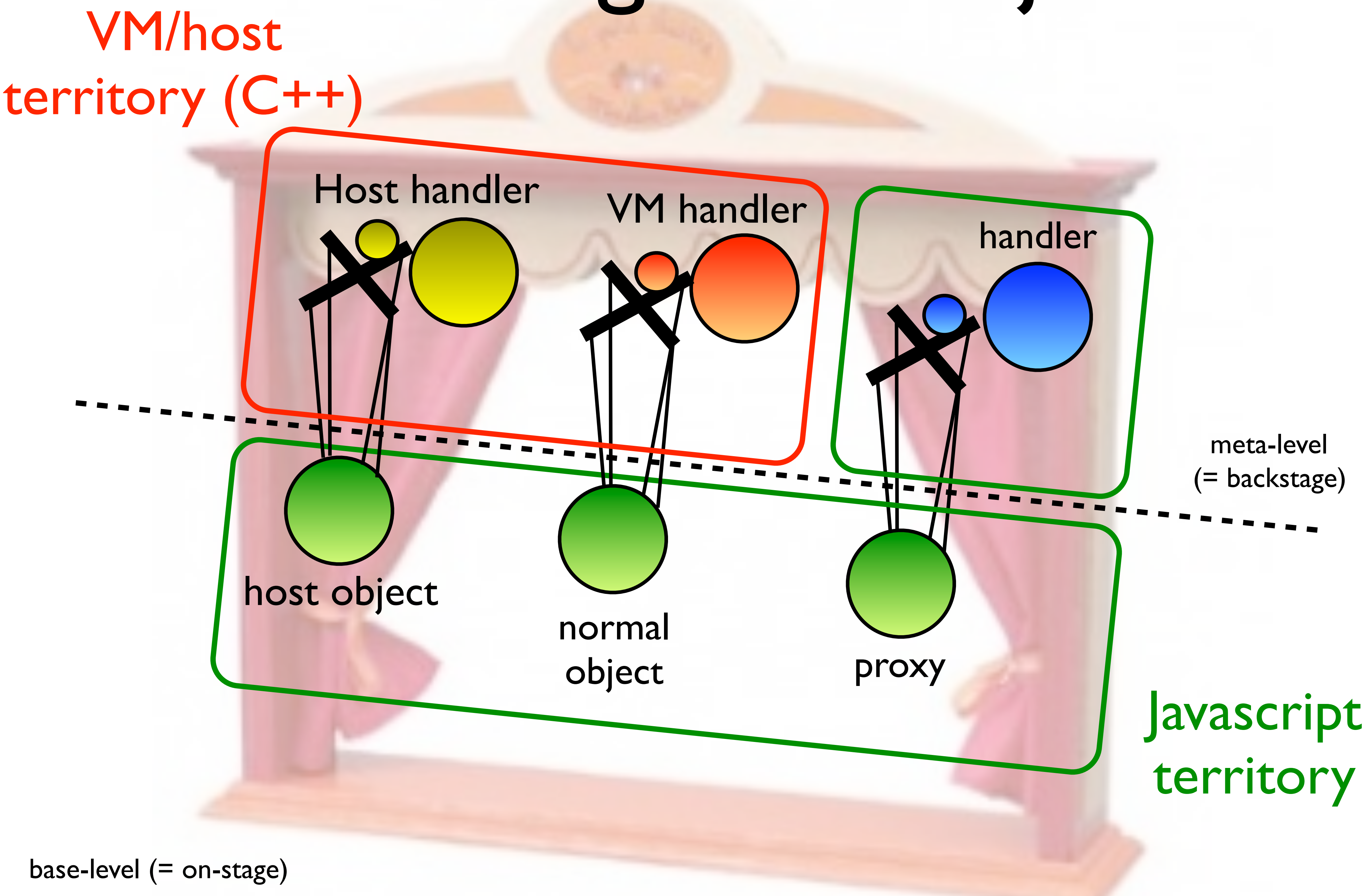
Retrofitting host objects



Retrofitting host objects



Retrofitting host objects



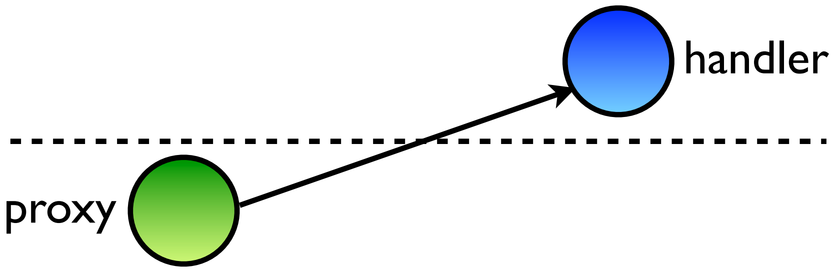
Proxies & frozen objects

- Frozen objects have strong invariants

- Proxies can emulate frozen objects, but handlers can’t violate these invariants

```
var target = { x: 0 };
Object.freeze(target); // now target.x should be immutable

var y = 0;
var proxy = Proxy(target, {
  get: function(tgt, name, rcvr) {
    return ++y;
  }
});

Object.isFrozen(proxy) // true!
proxy.x // error: cannot report inconsistent value for ‘x’
```

Meta-level shifting

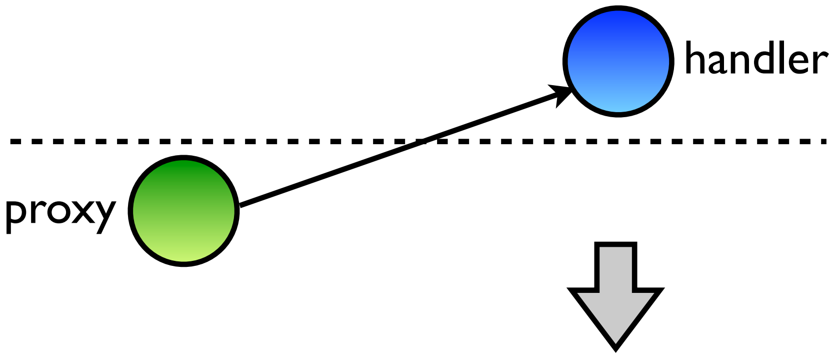


```
Object.getOwnPropertyDescriptor(proxy,name)        handler.getOwnPropertyDescriptor(target,name)
Object.defineProperty(proxy,name,pd)               handler.defineProperty(target,name,pd)
Object.getOwnPropertyNames(proxy)                  handler.getOwnPropertyNames(target)
delete proxy.name                                  handler.deleteProperty(target,name)
for (name in proxy) { ... }                        handler.iterate(target)
for (name in Object.create(proxy)) { ... }         handler.enumerate(target)
Object.{freeze|seal|preventExtensions}(proxy)      handler.{freeze|seal|preventExtensions}(target)
name in proxy                                      handler.has(target,name)
({}).hasOwnProperty.call(proxy, name)              handler.hasOwn(target,name)
Object.keys(proxy)                                 handler.keys(target)
proxy.name                                         handler.get(target,name,receiver)
proxy.name = val                                   handler.set(target,name,value,receiver)
proxy(...args)                                     handler.apply(target,receiver,args)
new proxy(...args)                                 handler.construct(target,args)
```

base-level: many operations on objects

meta-level: all operations reified as invocations of traps

Meta-level shifting



```
Object.getOwnPropertyDescriptor(proxy,name)
Object.defineProperty(proxy,name,pd)
Object.getOwnPropertyNames(proxy)
delete proxy.name
for (name in proxy) { ... }
for (name in Object.create(proxy)) { ... }
Object.{freeze|seal|preventExtensions}(proxy)
name in proxy
({}).hasOwnProperty.call(proxy, name)
Object.keys(proxy)
proxy.name
proxy.name = val
proxy(...args)
new proxy(...args)
```

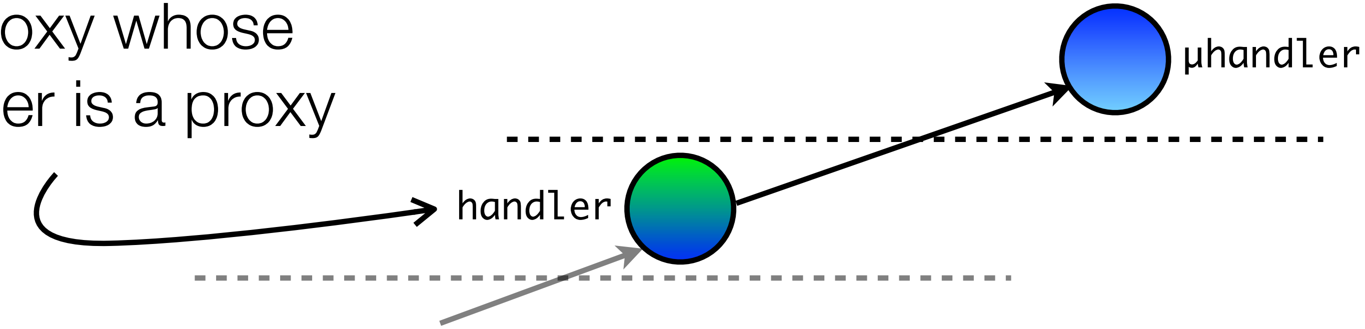
```
handler.getOwnPropertyDescriptor(target,name)
handler.defineProperty(target,name,pd)
handler.getOwnPropertyNames(target)
handler.deleteProperty(target,name)
handler.iterate(target)
handler.enumerate(target)
handler.{freeze|seal|preventExtensions}(target)
handler.has(target,name)
handler.hasOwn(target,name)
handler.keys(target)
handler.get(target,name,receiver)
handler.set(target,name,value,receiver)
handler.apply(target,receiver,args)
handler.construct(target,args)
```

base-level: many operations on objects

meta-level: all operations reified as invocations of traps

Meta-level shifting

a proxy whose
handler is a proxy



```
handler.getOwnPropertyDescriptor(target,name)
handler.defineProperty(target,name,pd)
handler.getOwnPropertyNames(target)
handler.deleteProperty(target,name)
handler.iterate(target)
handler.enumerate(target)
handler.{freeze|seal|preventExtensions}(target)
handler.has(target,name)
handler.hasOwn(target,name)
handler.keys(target)
handler.get(target,name,rcvr)
handler.set(target,name,value,rcvr)
handler.apply(target,rcvr,args)
handler.construct(target,args)
```

```
μhandler.get(tgt,‘getOwnPr..’)(target,name)
μhandler.get(tgt,‘definePr..’)(target,name,pd)
μhandler.get(tgt,‘getOwnPr..’)(target)
μhandler.get(tgt,‘deletePr..’)(target,name)
μhandler.get(tgt,‘iterate’)(target)
μhandler.get(tgt,‘enumerate’)(target)
μhandler.get(tgt,‘freeze’|..)(target)
μhandler.get(tgt,‘has’)(target,name)
μhandler.get(tgt,‘hasOwn’)(target,name)
μhandler.get(tgt,‘keys’)(target)
μhandler.get(tgt,‘get’)(target,name,rcvr)
μhandler.get(tgt,‘set’)(target,name,value,rcvr)
μhandler.get(tgt,‘apply’)(target,rcvr,args)
μhandler.get(tgt,‘construct’)(target,args)
```

meta-level: all operations reified
as invocations of traps

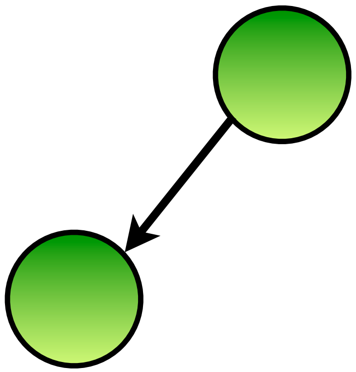
meta-meta-level: all operations
reified as invocations of ‘get’ trap

Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane

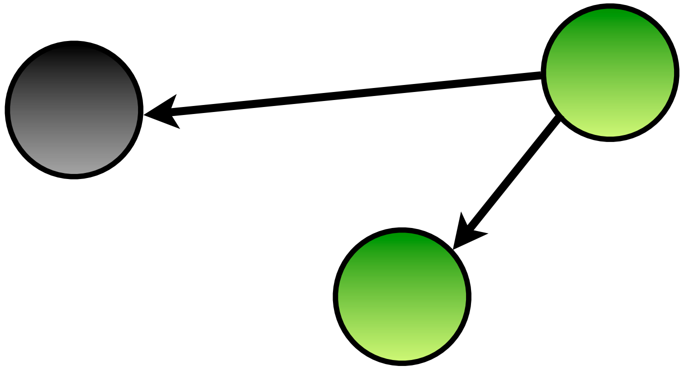
Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane



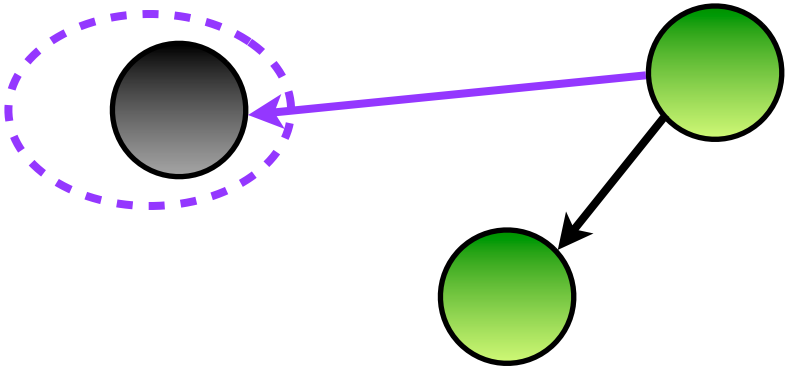
Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane



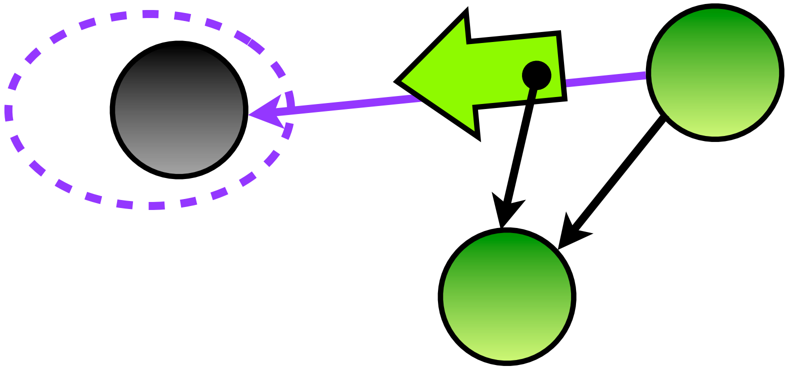
Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane



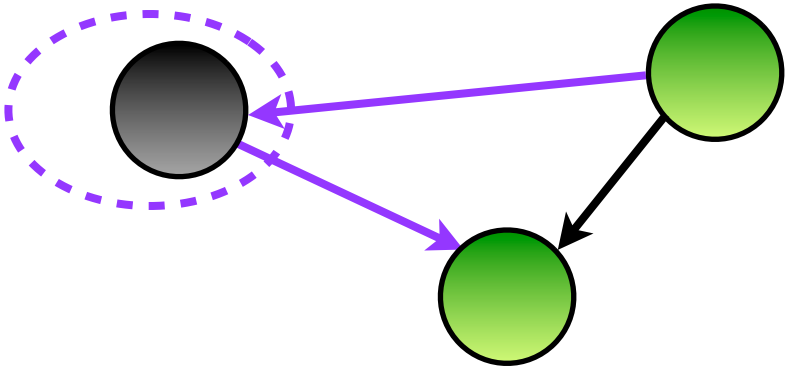
Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane



Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane

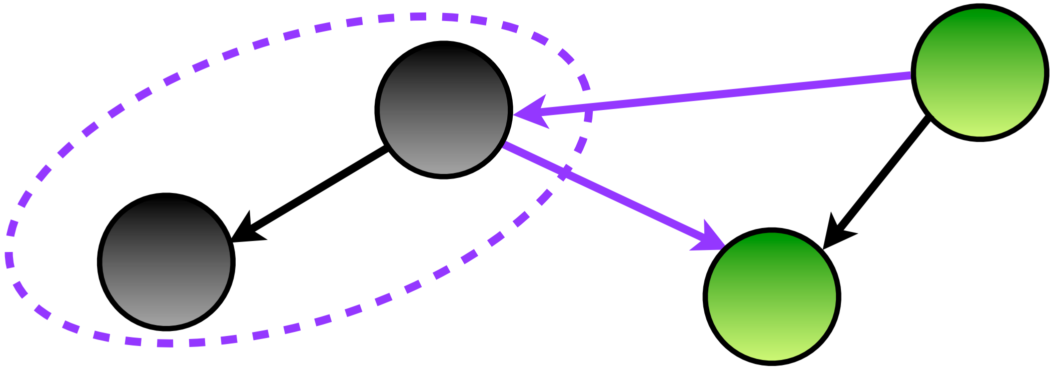


Example: membranes

- Caja: capability-secure subset of Javascript

- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects

- References can be made revocable through a membrane

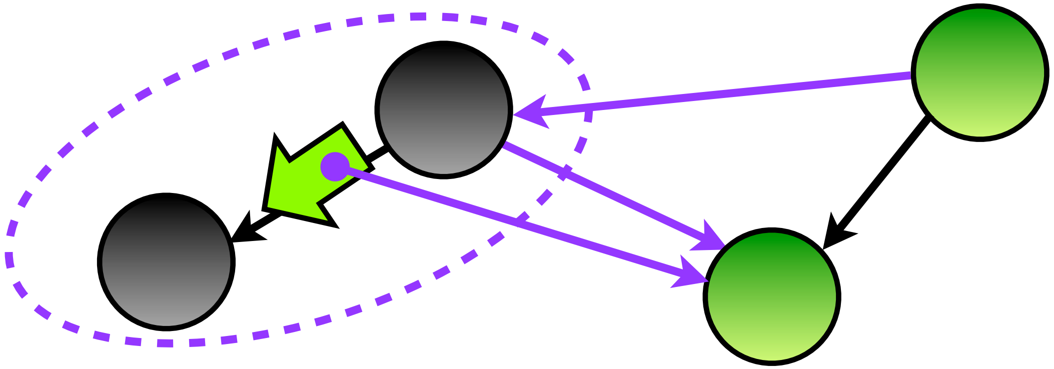


Example: membranes

- Caja: capability-secure subset of Javascript

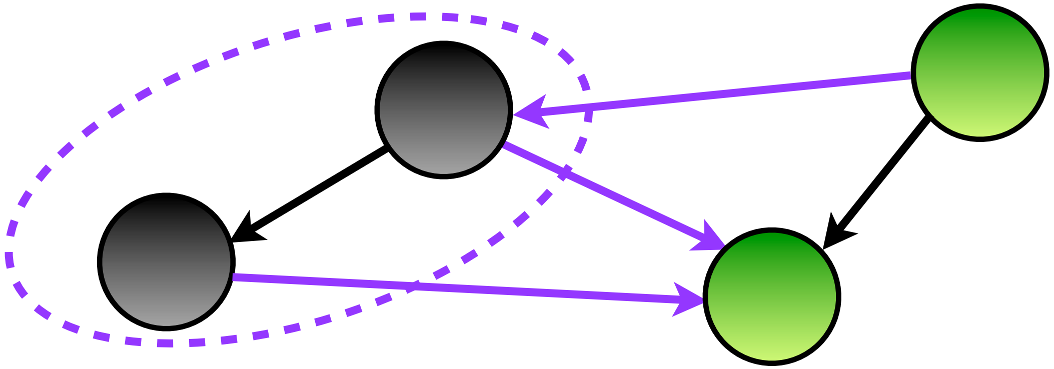
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects

- References can be made revocable through a membrane



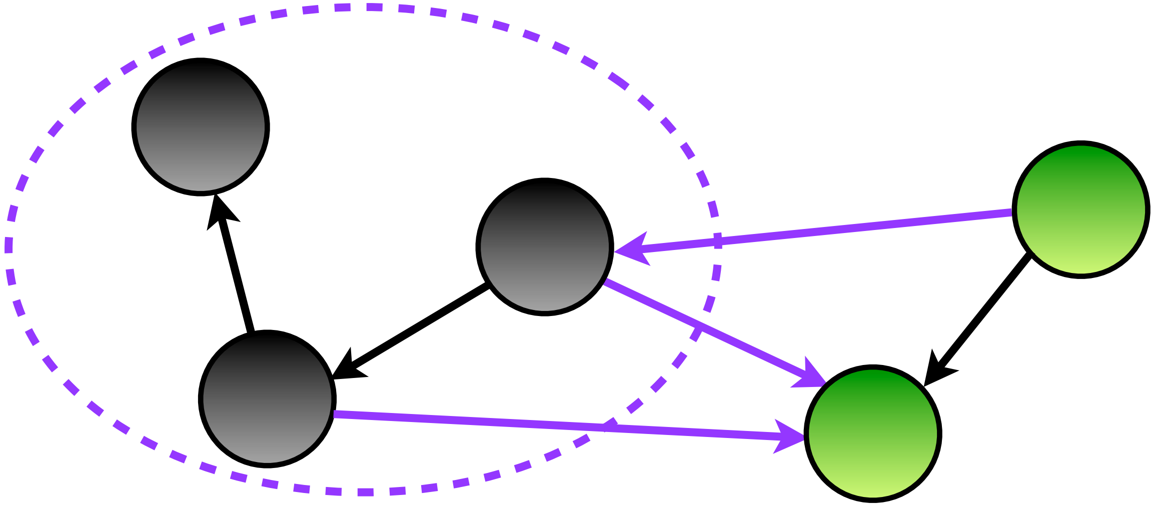
Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane



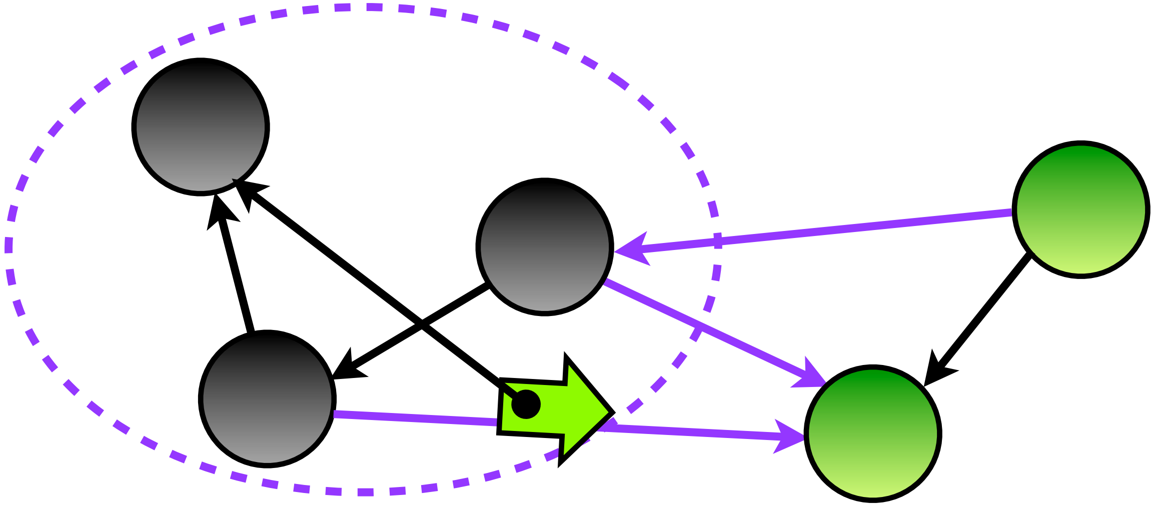
Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane



Example: membranes

- Caja: capability-secure subset of Javascript
- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects
- References can be made revocable through a membrane

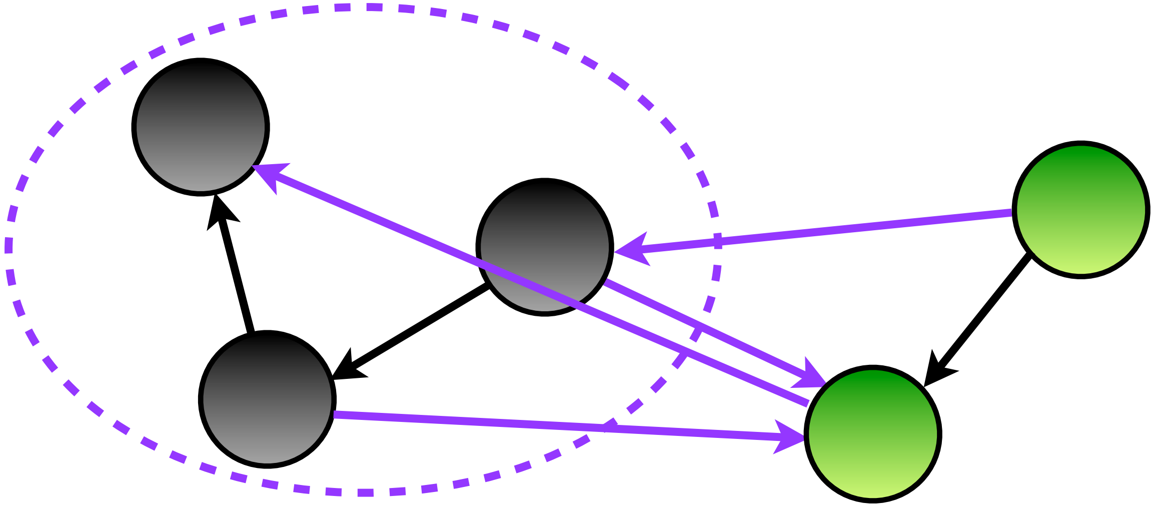


Example: membranes

- Caja: capability-secure subset of Javascript

- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects

- References can be made revocable through a membrane

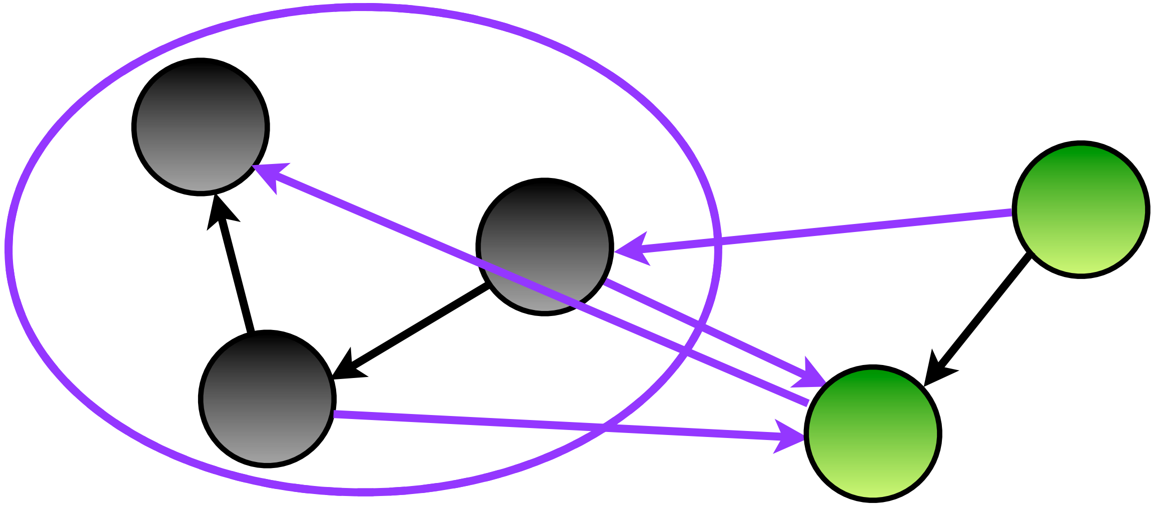


Example: membranes

- Caja: capability-secure subset of Javascript

- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects

- References can be made revocable through a membrane

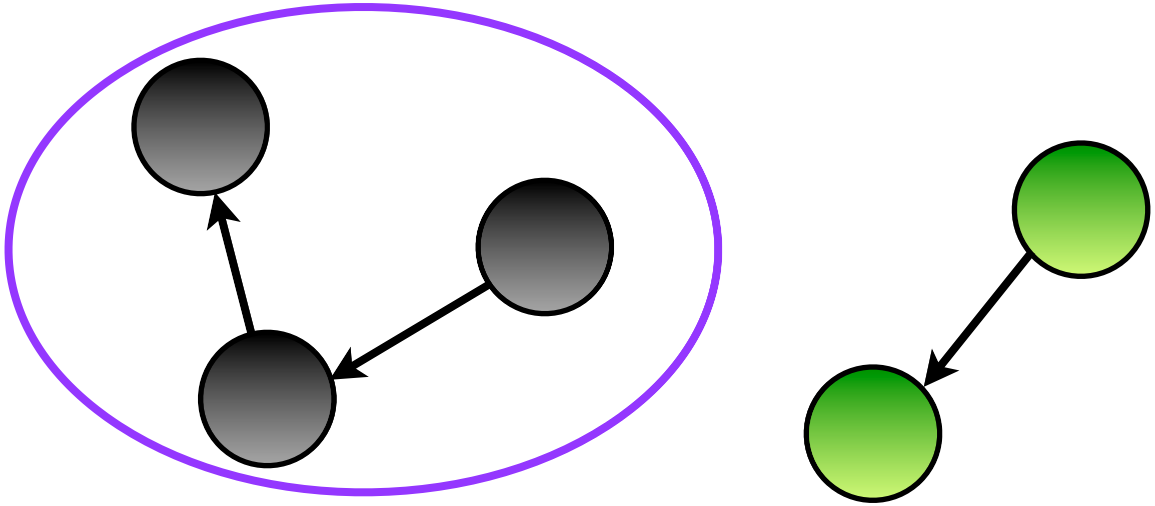


Example: membranes

- Caja: capability-secure subset of Javascript

- In the object-capability paradigm, an object is powerless unless given a reference to other (more) powerful objects

- References can be made revocable through a membrane



Example: membranes

```
function makeMembrane(initTarget) {
  var enabled = true;




















}
```

Example: membranes

```
function makeMembrane(initTarget) {
  var enabled = true;

  return {
    wrapper: wrap(initTarget),
    revoke: function() { enabled = false; }
  };
}
```

Example: membranes

```
function makeMembrane(initTarget) {
  var enabled = true;
  function wrap(target) {
    if (isPrimitive(target)) { return target; }









    return Proxy(target, metaHandler);
  }
  return {
    wrapper: wrap(initTarget),
    revoke: function() { enabled = false; }
  };
}
```

Example: membranes

```
function makeMembrane(initTarget) {
  var enabled = true;
  function wrap(target) {
    if (isPrimitive(target)) { return target; }
    var metaHandler = Proxy(target, {
      get: function(target, trapName) {
        if (!enabled) { throw new Error("revoked"); }
        return function(...args) {
          return wrap(Reflect[trapName](...args.map(wrap)));
        }
      }
    });
    return Proxy(target, metaHandler);
  }
  return {
    wrapper: wrap(initTarget),
    revoke: function() { enabled = false; }
  };
}
```

Pitfalls / Limitations of Proxies

- Proxy objects have their own identity.
  - “Two-body problem”
- Cannot turn regular objects into proxies.
  - cf. Smalltalk’s `become:` primitive
  - Security gotcha’s
  - VM implementation gotcha’s

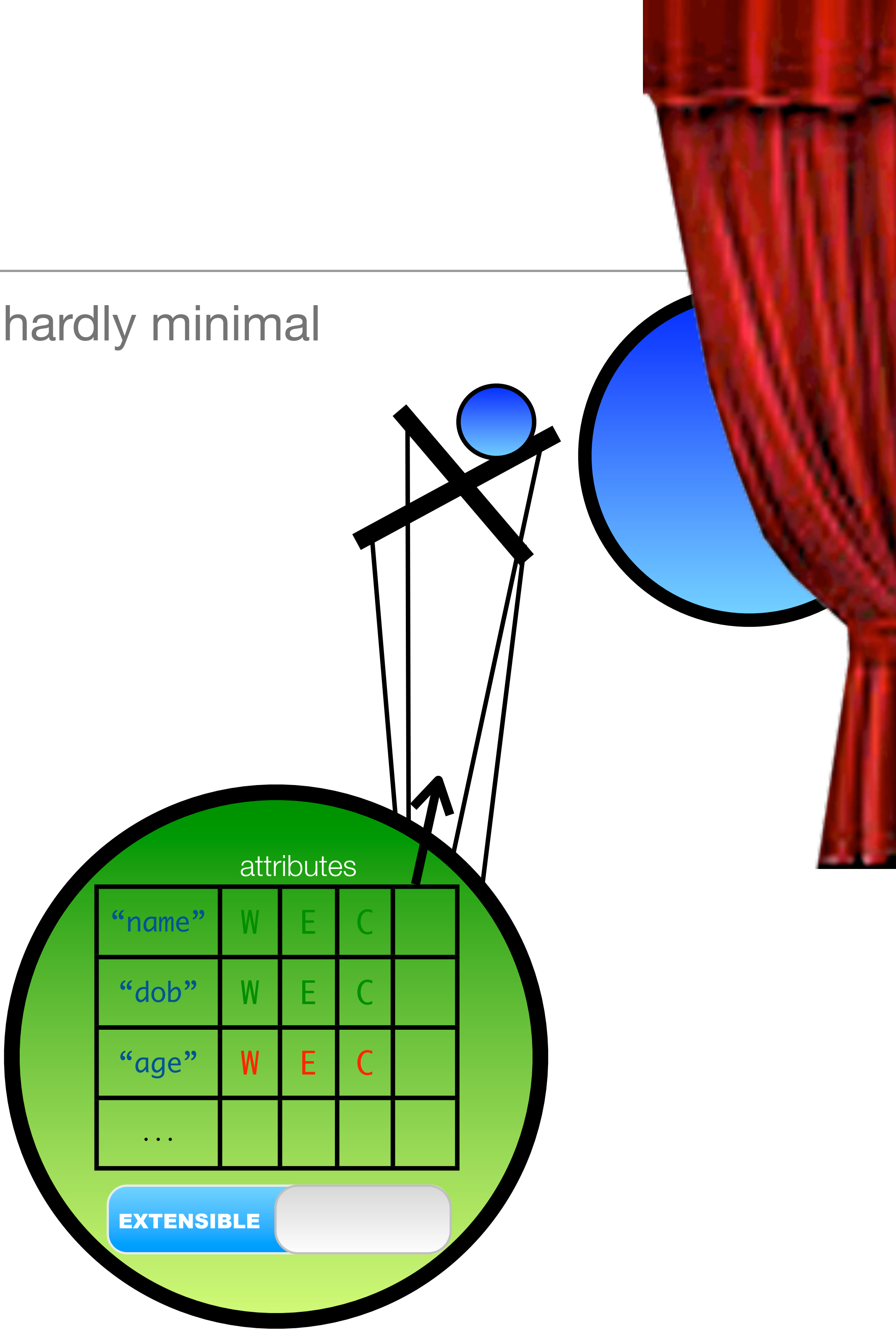
Availability

- Both Firefox and Chrome currently implement an earlier prototype of the Proxy API
- Library that implements current API on top of the old API
- Available on Github: https://github.com/tvcutsem/harmony-reflect

```
<script src="reflect.js"></script>
```

Conclusion

- Javascript: dynamic, flexible, but hardly minimal
- Proxies in ECMAScript 6
- Makes the Javascript “MOP” explicit for the first time
- Stratified API



“name”	W	E	C	
“dob”	W	E	C	
“age”	W	E	C	
. . .				