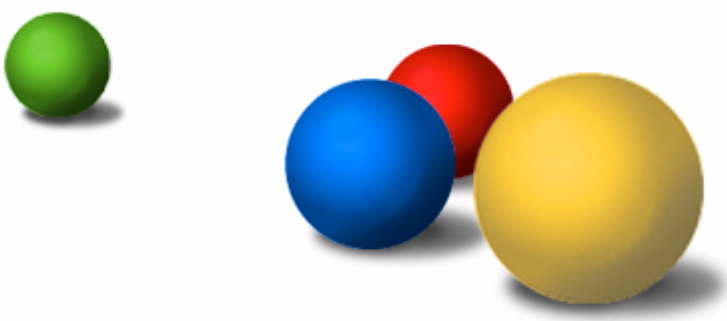
Google™

Vrije Universiteit Brussel

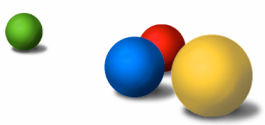
# Harmony Highlights Proxies & Traits

Tom Van Cutsem
Mark S. Miller



# Ecmascript 5

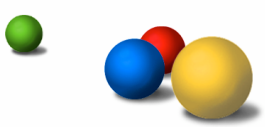
- Strict Mode:
  - guards against common pitfalls
  - rejects confusing features (e.g. "with" statement)
  - throws exceptions instead of failing silently
- Object-manipulation API
  - "javascript.lang.reflect"



# Property Descriptors

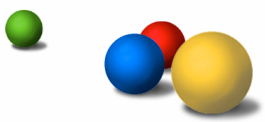
- Data and accessor properties
- Property attributes

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

Object.getOwnPropertyDescriptor(point, 'x');

  { value: 5,
    writable: true,
    enumerable: true,
    configurable: true }

Object.getOwnPropertyDescriptor(point, 'y');

  { get: function () { return this.x; },
    set: undefined,
    enumerable: true,
    configurable: true }
```



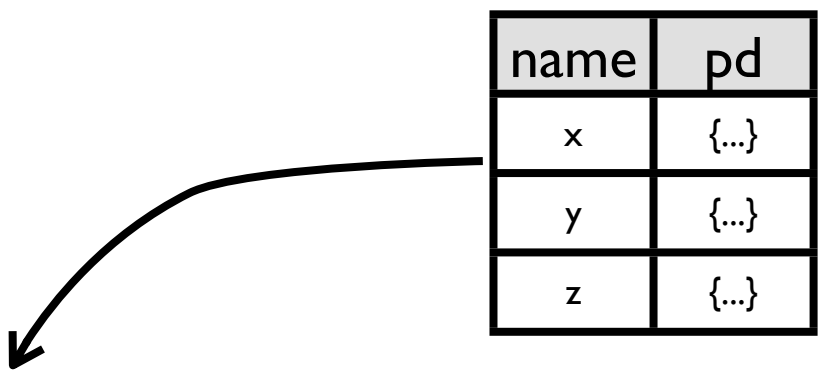
# Property Descriptor Maps

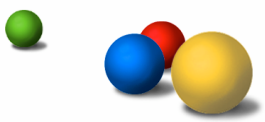
```
var point =
  { x: 5,
    get y() { return this.x; } };


Object.defineProperty(point, 'z', {
  value: 0,
  enumerable: true,
  writable: false,
  configurable: false });


var point = Object.create(
  Object.prototype,
  { x: { value: 5, enumerable: true, ... },
    y: { get: function() {...}, enumerable: true, ... },
    z: { value: 0, enumerable: true, ... }  });
```



# Property Descriptor Maps

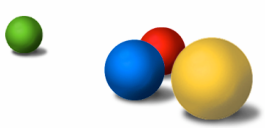
```
var point =
  { x: 5,
    get y() { return this.x; } };


Object.defineProperty(point, 'z', {
  value: 0,
  enumerable: true,
  writable: false,
  configurable: false });


var point = Object.create(
  Object.prototype,
  { x: { value: 5, enumerable: true, ... },
    y: { get: function() {...}, enumerable: true, ... },
    z: { value: 0, enumerable: true, ... }  });
```



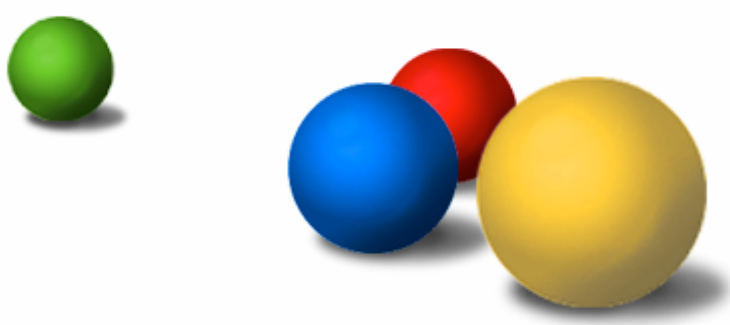



# Tamper-proof Objects

```
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
```

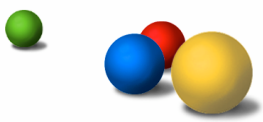


# Proxies



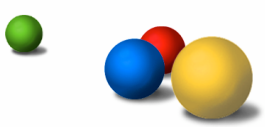
# Dynamic Proxies

- Generic handling of property access:
  - Generic wrappers: access control, tracing, profiling, ...
  - Stratified form of `__noSuchMethod__`
- Generic handling of other operations applicable to objects:
  - Virtual objects: persistent objects, remote objects, ...
  - Emulate host objects



# Tracing: static proxies

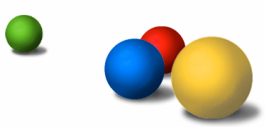
```
function makeTracer(obj) {
  var proxy = Object.create(Object.getProtoypeOf(obj), {});
  Object.getOwnPropertyNames(obj).forEach(function(name) {
    var pd = Object.getOwnPropertyDescriptor(obj, name);
    Object.defineProperty(proxy, name, {
      get: function() {
        trace('get', name, obj);
        return obj[name];
      },
      set: function(v) {
        trace('set', name, obj, v);
        obj[name] = v;
      },
      // copy attributes from pd
    });
  });
  return proxy;
}
```



# Tracing: static proxies

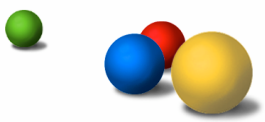
```
function makeTracer(obj) {
  var proxy = Object.create(Object.getProtoypeOf(obj), {});
  Object.getOwnPropertyNames(obj).forEach(function(name) {
    var pd = Object.getOwnPropertyDescriptor(obj, name);
    Object.defineProperty(proxy, name, {
      get: function() {
        trace('get', name, obj);
        return obj[name];
      },
      set: function(v) {
        trace('set', name, obj, v);
        obj[name] = v;
      },
      // copy attributes from pd
    });
  });
  return proxy;
}
```

- proxy doesn't reflect structural changes made to 'obj'
- structural changes made to proxy are not reflected in 'obj'
- structural changes:
  - added/deleted properties
  - changed property attributes



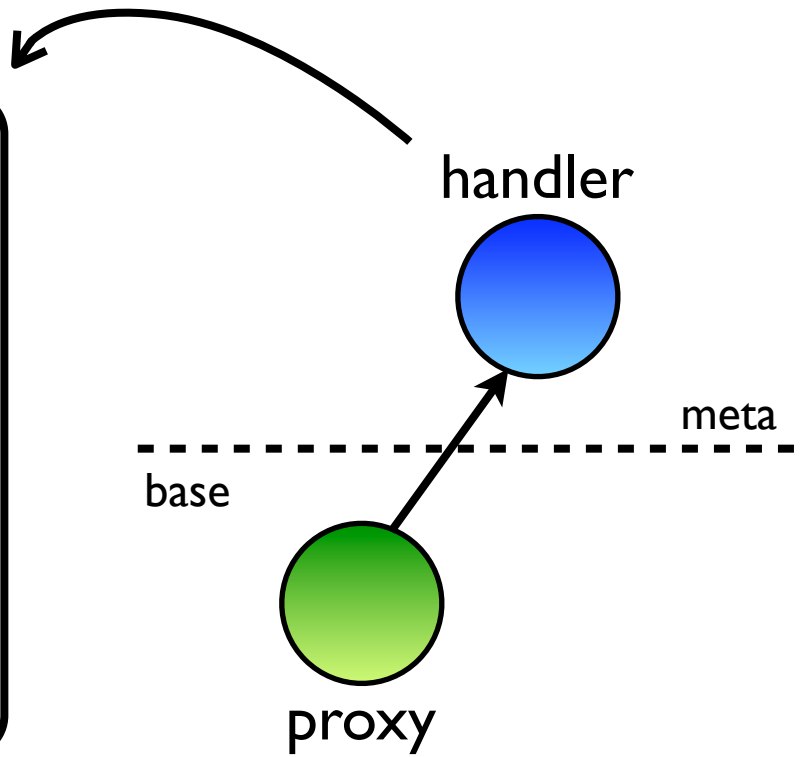
# Tracing: dynamic proxies

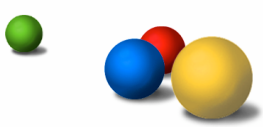
```
function makeTracer(obj) {
  var proxy = Proxy.create({
    get: function(rcvr, name) {
      trace('get', name, obj);
      return obj[name];
    },
    set: function(rcvr, name, val) {
      trace('set', name, obj, val);
      obj[name] = val;
      return true;
    },
    ...
  }, Object.getPrototypeOf(obj));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



# Tracing: dynamic proxies

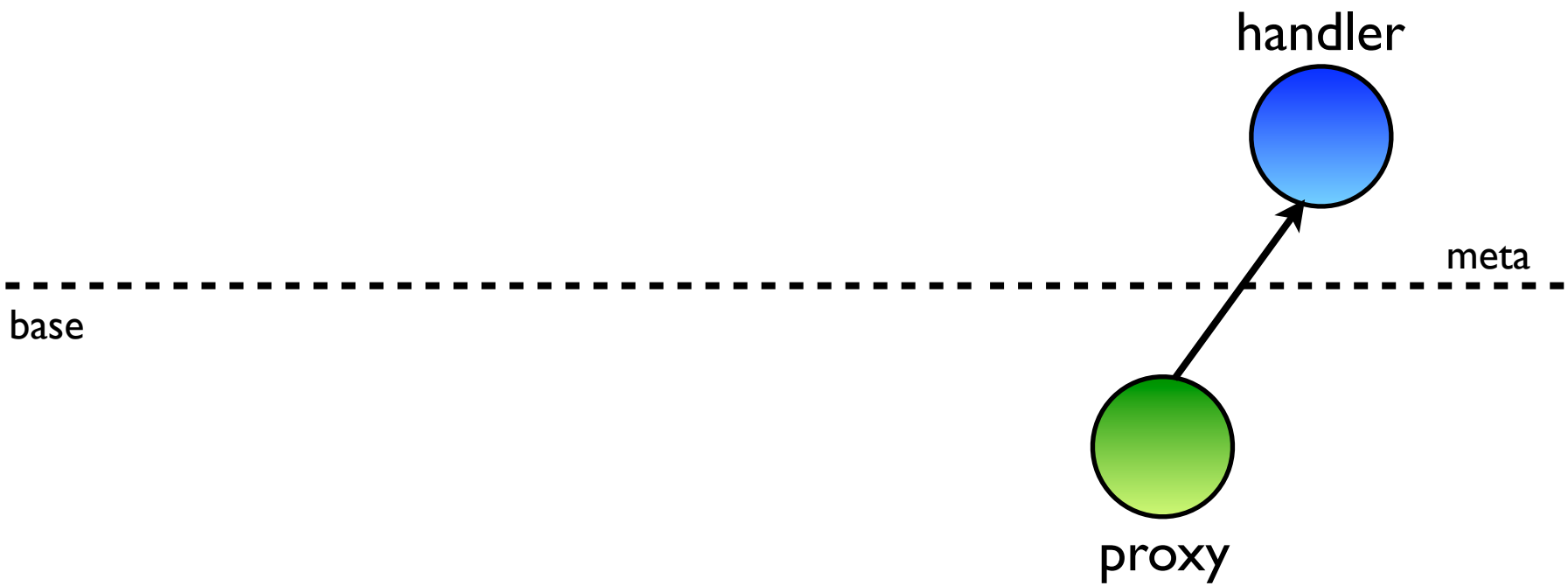
```
function makeTracer(obj) {
  var proxy = Proxy.create({
    get: function(rcvr, name) {
      trace('get', name, obj);
      return obj[name];
    },
    set: function(rcvr, name, val) {
      trace('set', name, obj, val);
      obj[name] = val;
      return true;
    },
    ...
  }, Object.getPrototypeOf(obj));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

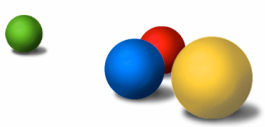


# Stratified API

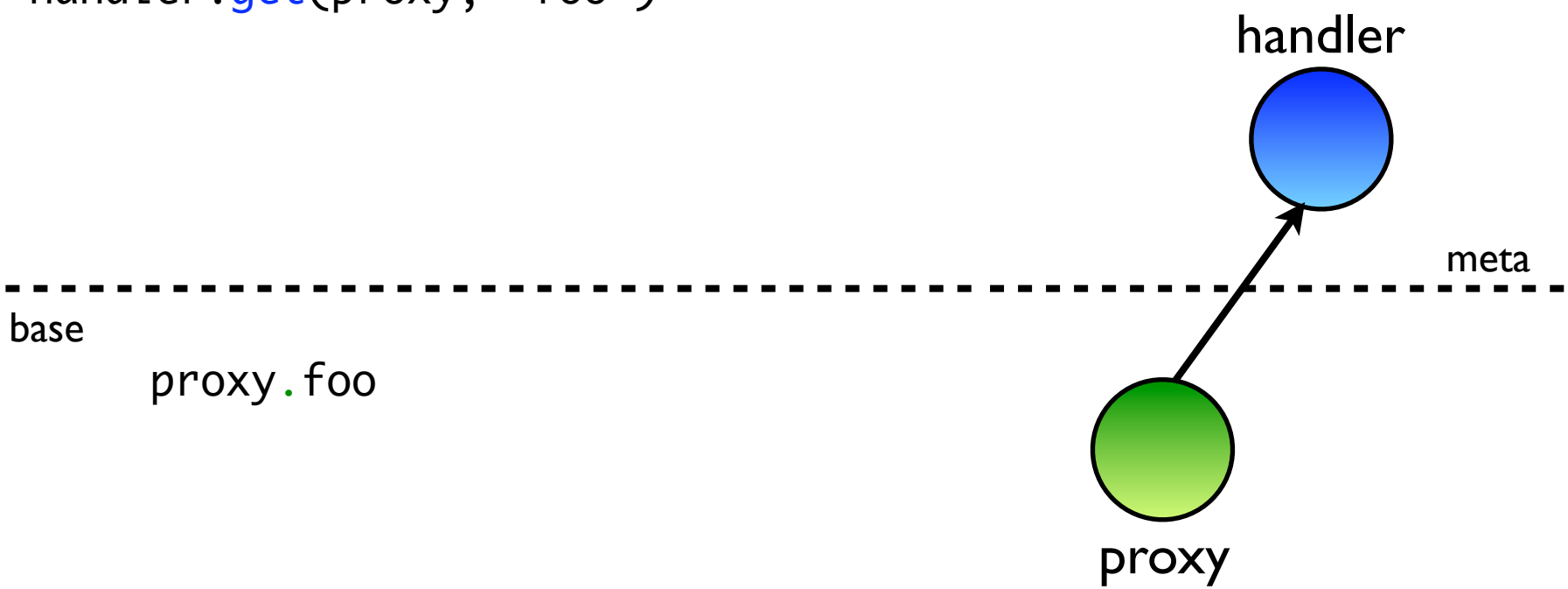
```
var proxy = Proxy.create(handler, proto);
```

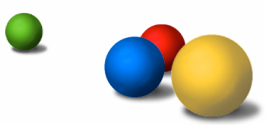


# Stratified API

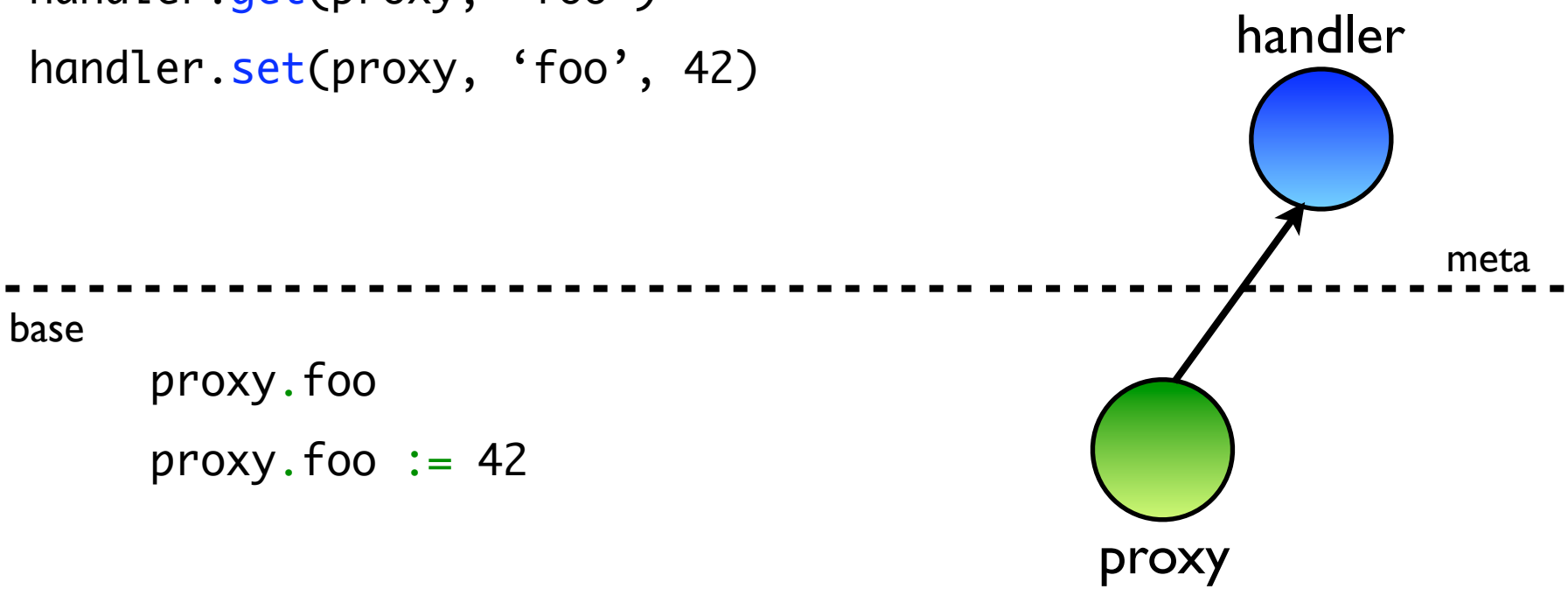
```
var proxy = Proxy.create(handler, proto);

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

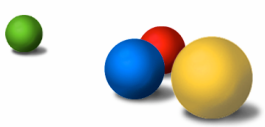


# Stratified API

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

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

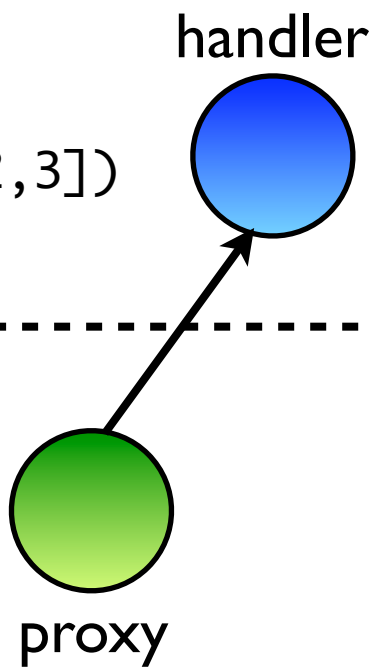


# Stratified API

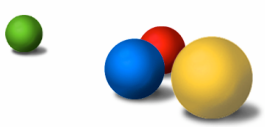
```
var proxy = Proxy.create(handler, proto);
```

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




base

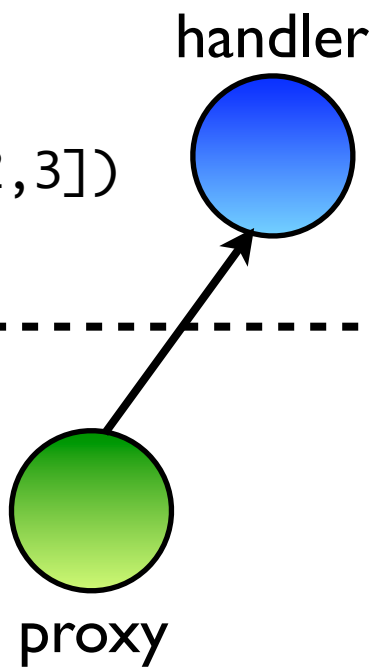
```
proxy.foo
proxy.foo := 42
proxy.foo(1,2,3)
```



# Stratified API

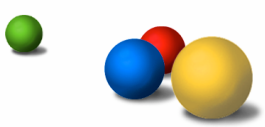
```
var proxy = Proxy.create(handler, proto);
```

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




base

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



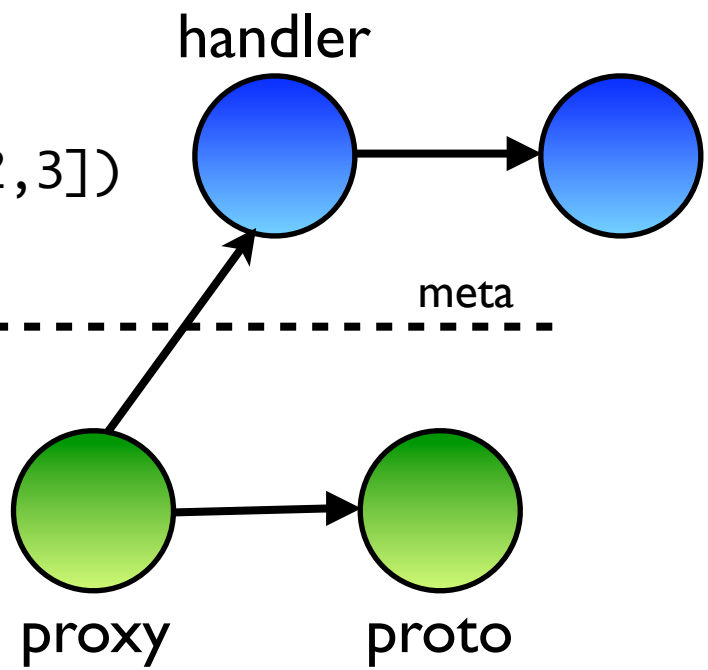
# Stratified API

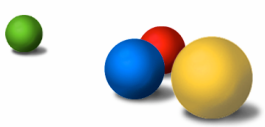
```
var proxy = Proxy.create(handler, proto);
```

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

base

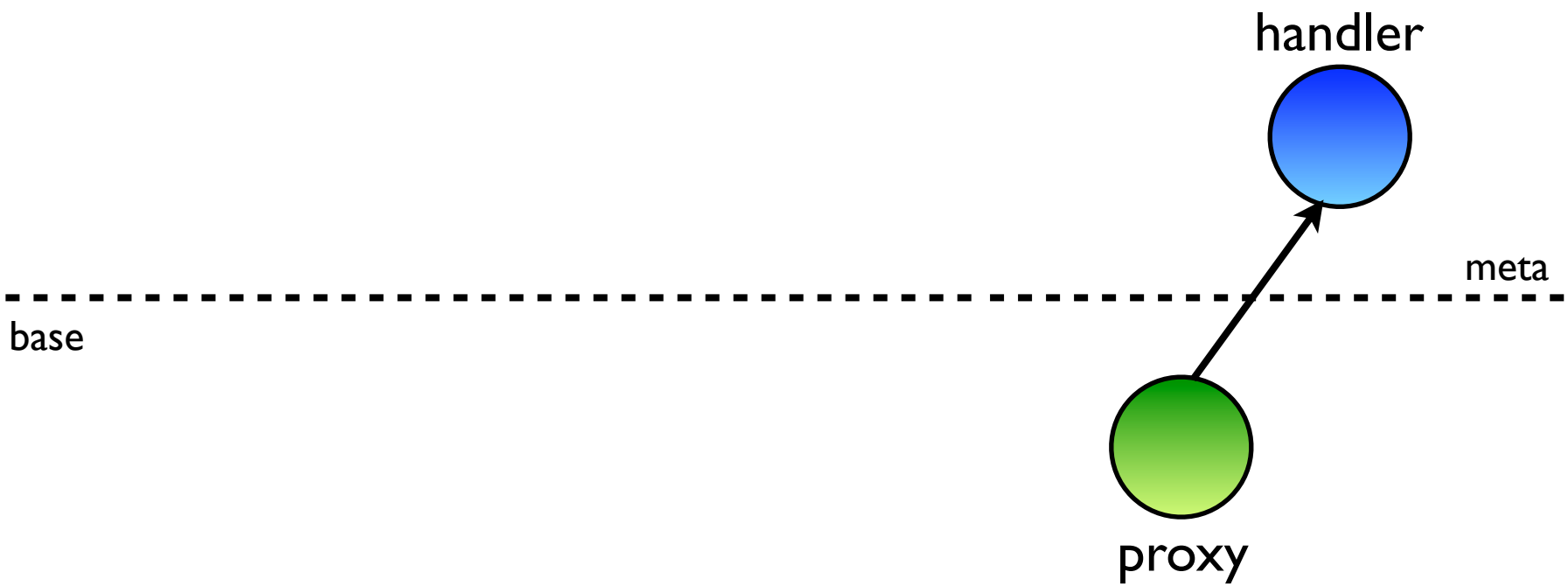
```
proxy.foo
proxy.foo := 42
proxy.foo(1,2,3)
proxy.get
```

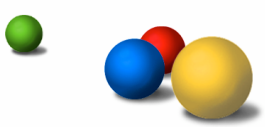


# Not just property access

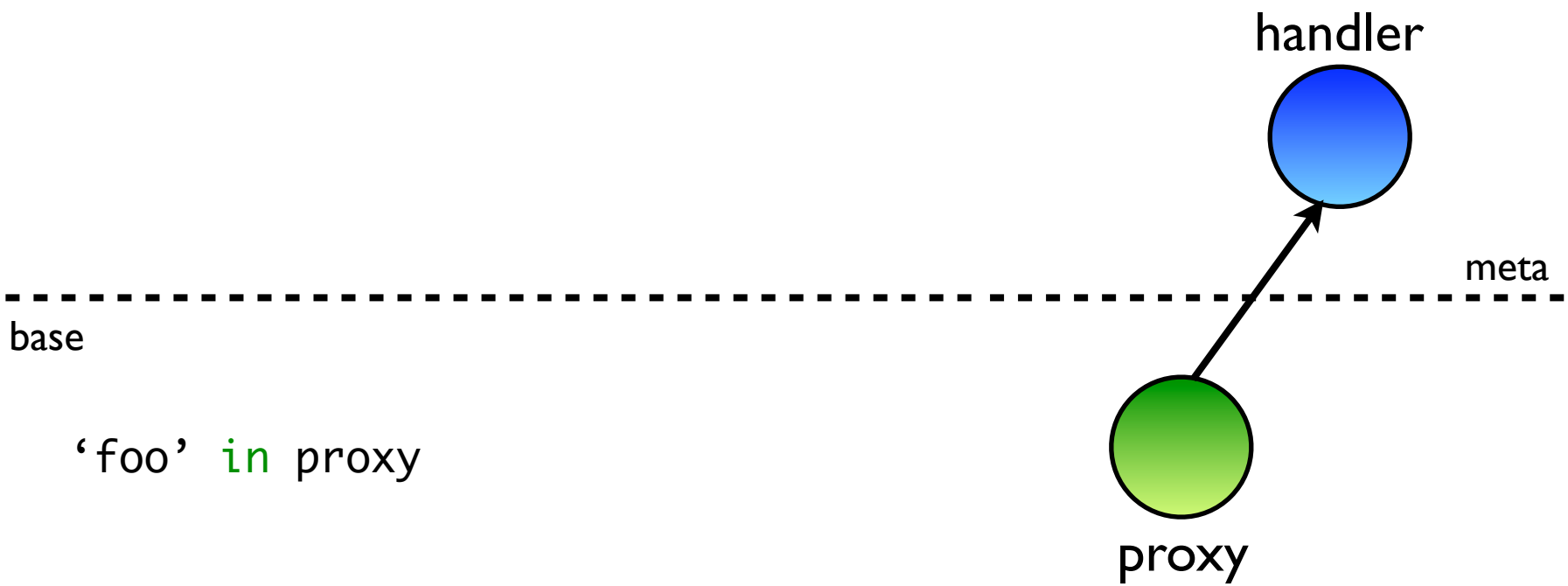
```
var proxy = Proxy.create(handler, proto);
```

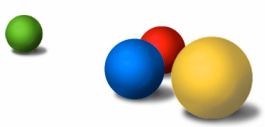


# Not just property access

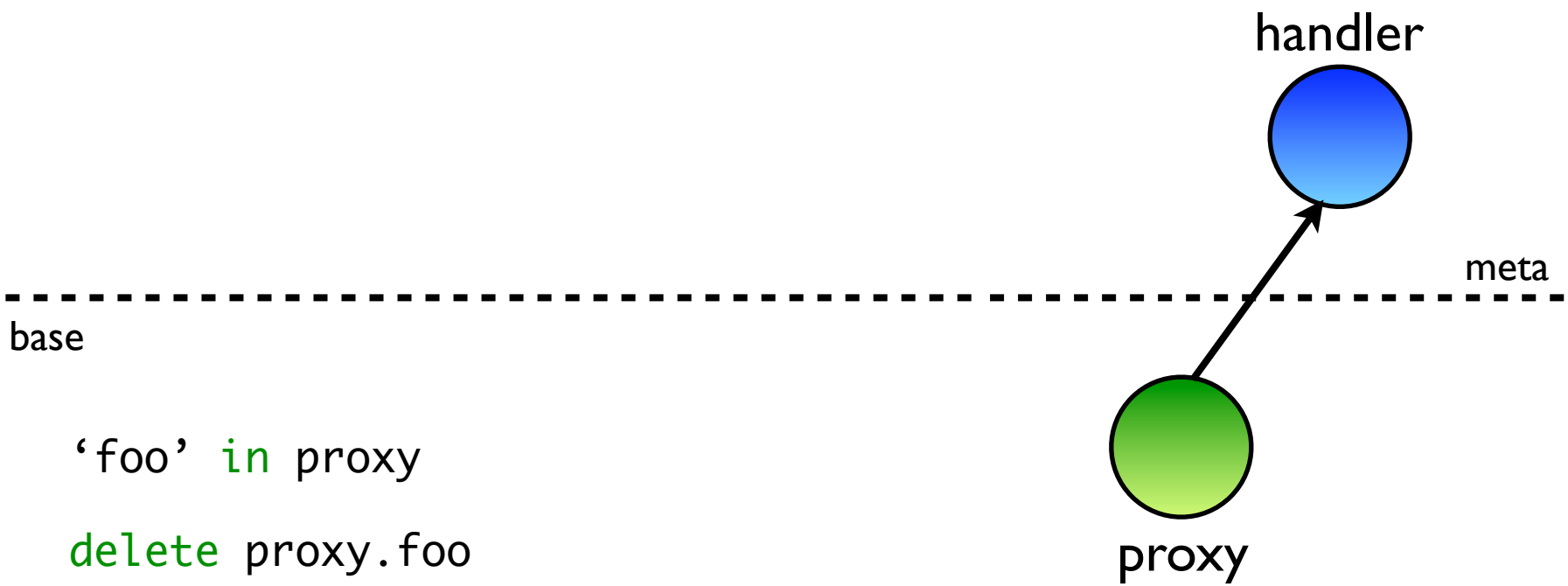
```
var proxy = Proxy.create(handler, proto);

  handler.has('foo')
```

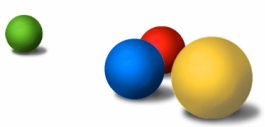


# Not just property access

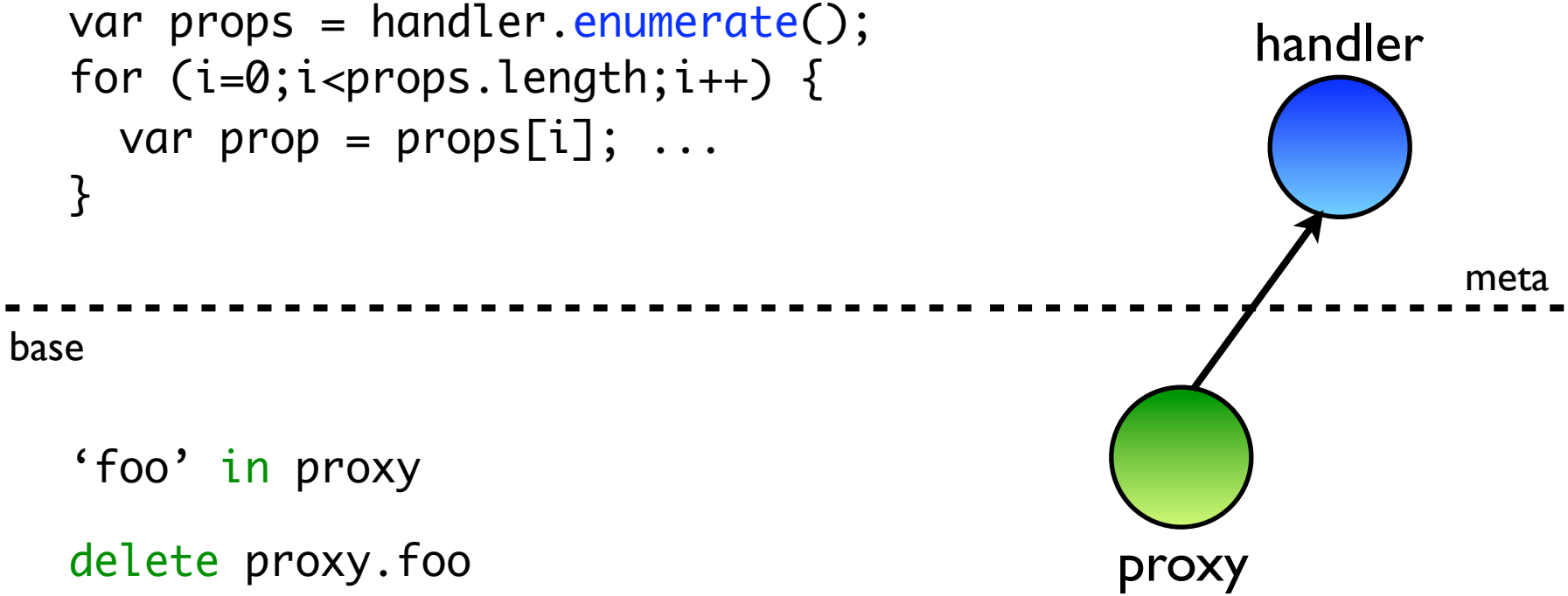
```
var proxy = Proxy.create(handler, proto);

  handler.has('foo')

  handler.delete('foo')
```



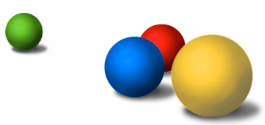



# Not just property access

```
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]; ...
  }
```



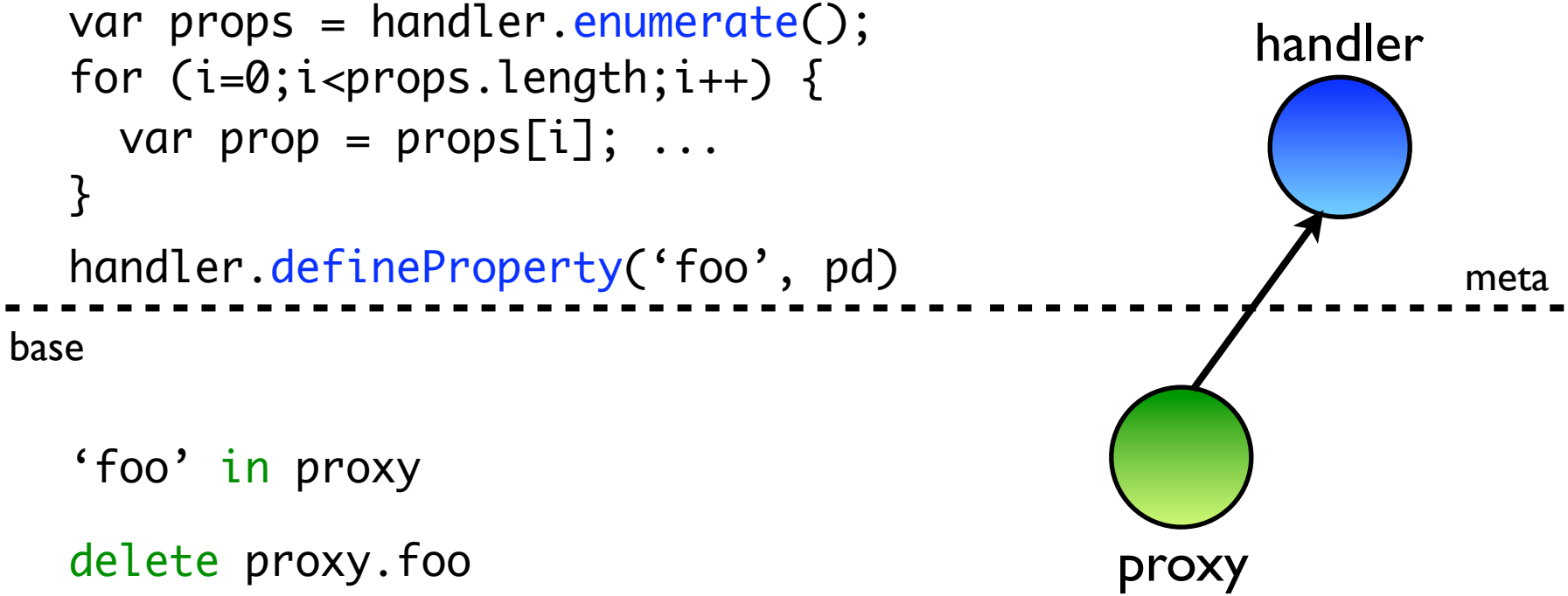

```
  'foo' in proxy

  delete proxy.foo

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



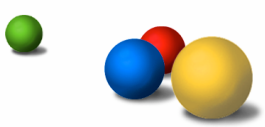
# Not just property access

```
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)
```

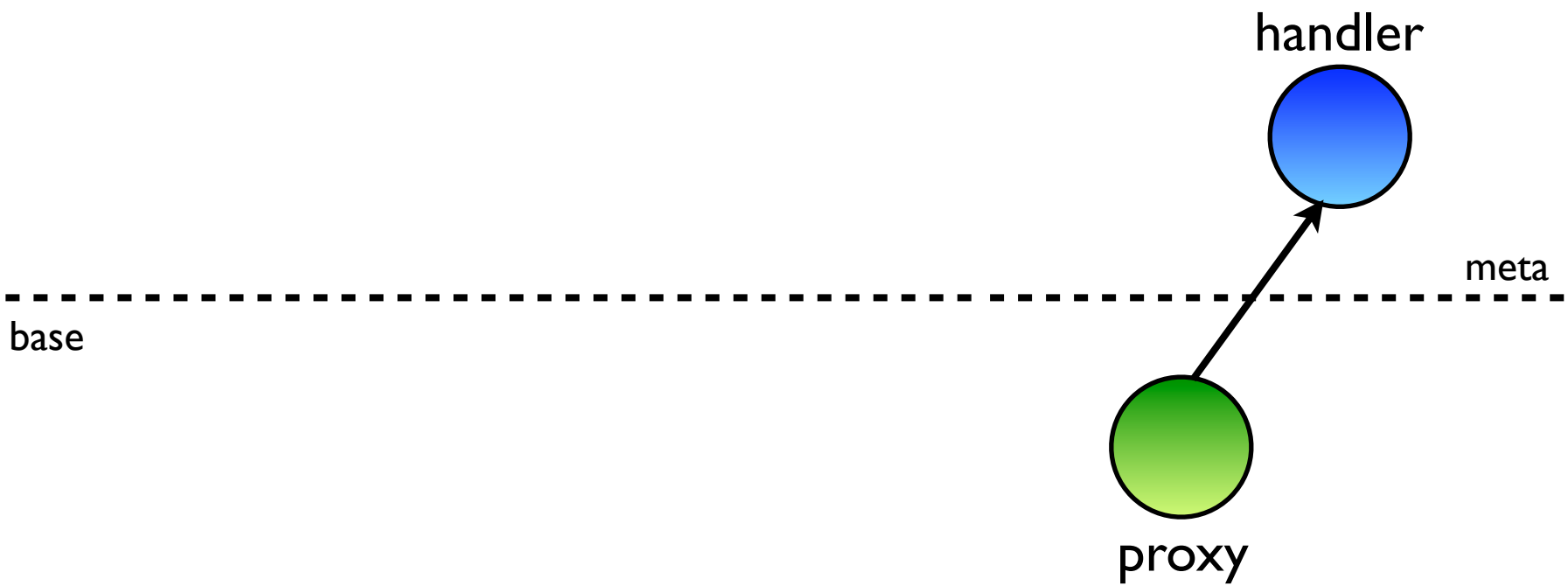


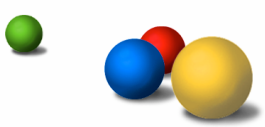

```
'foo' in proxy

delete proxy.foo

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

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



# But not quite everything either

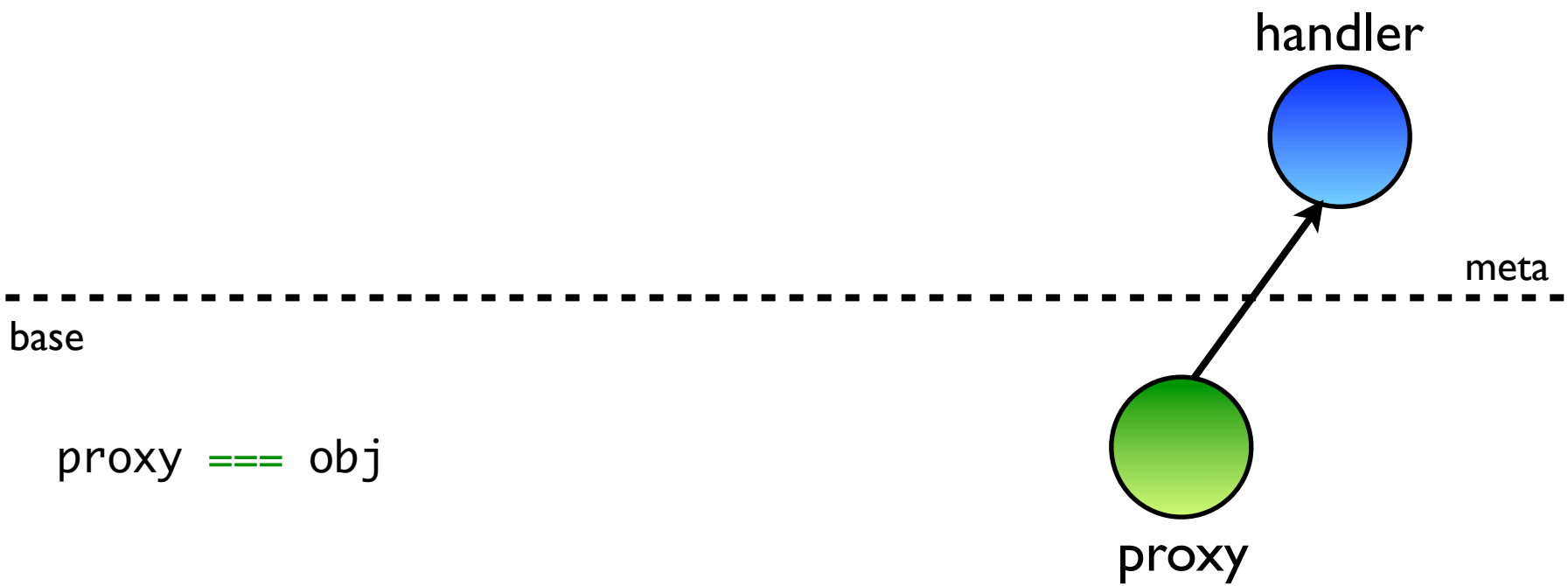
```
var proxy = Proxy.create(handler, proto);
```



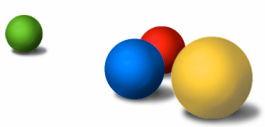



# But not quite everything either

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

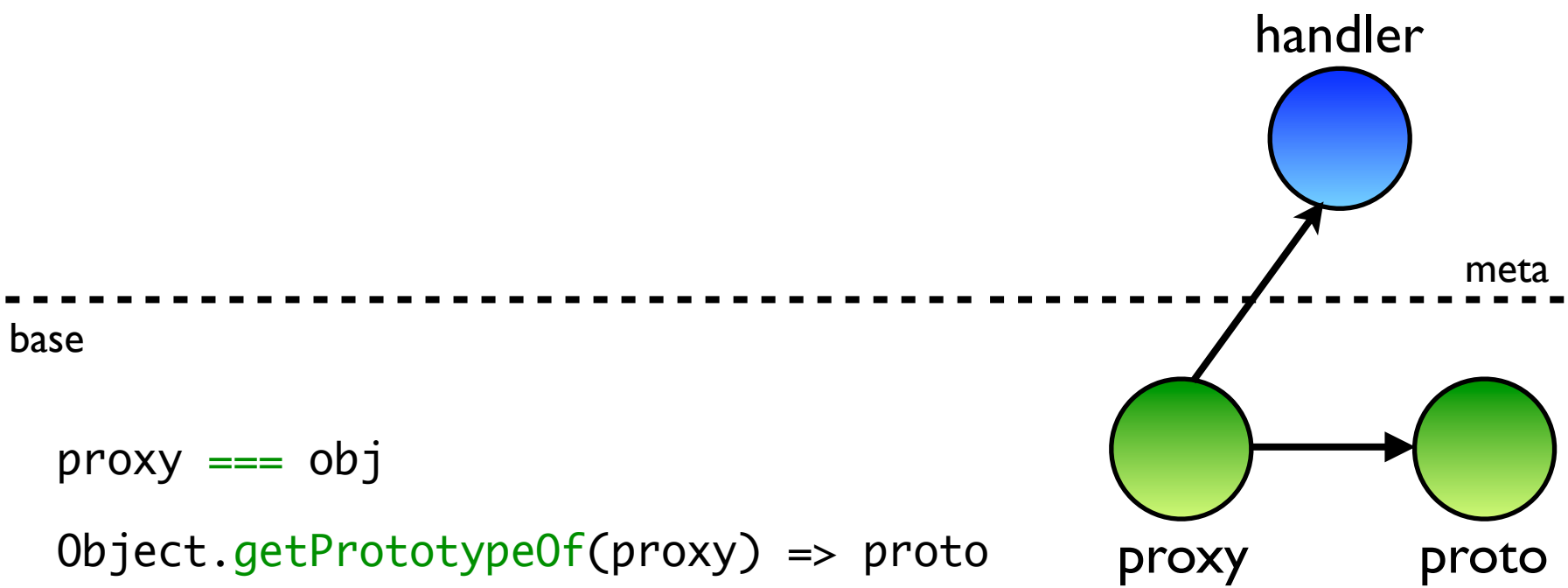



```
proxy === obj
```

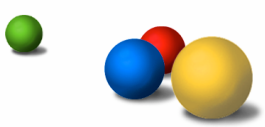


# But not quite everything either

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

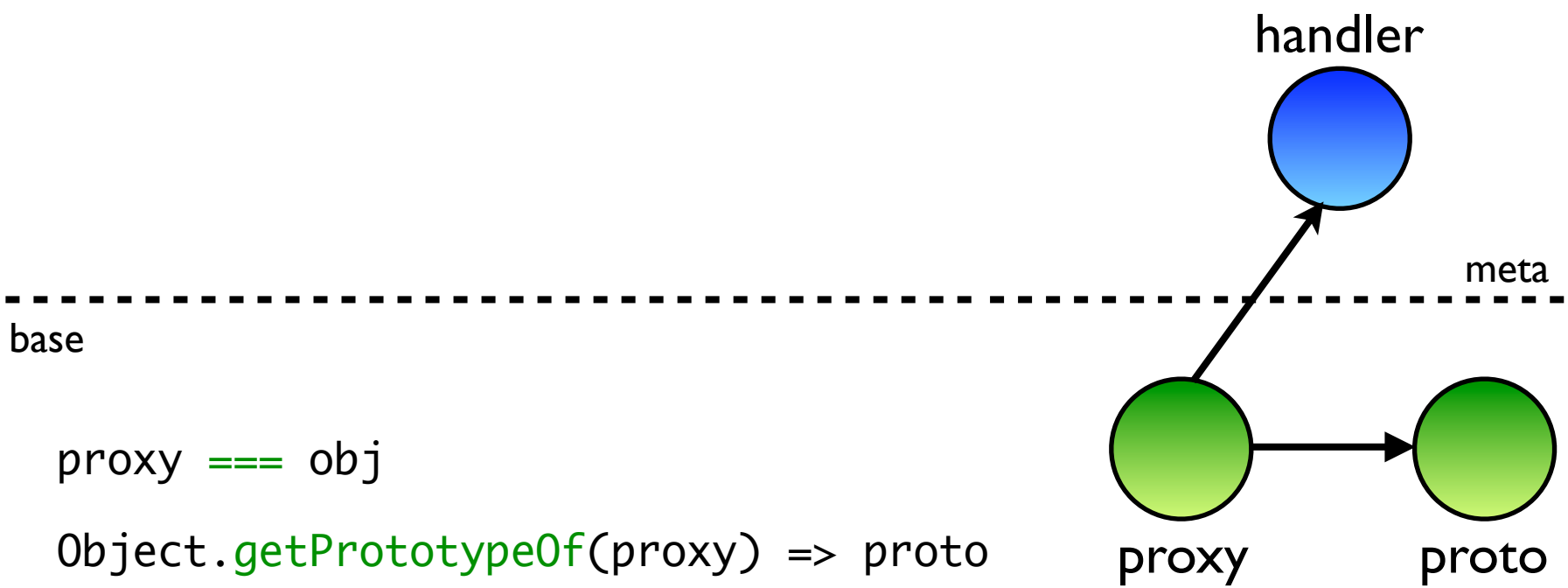



```
proxy === obj

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

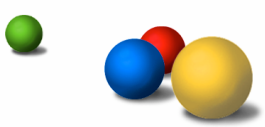


# But not quite everything either

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

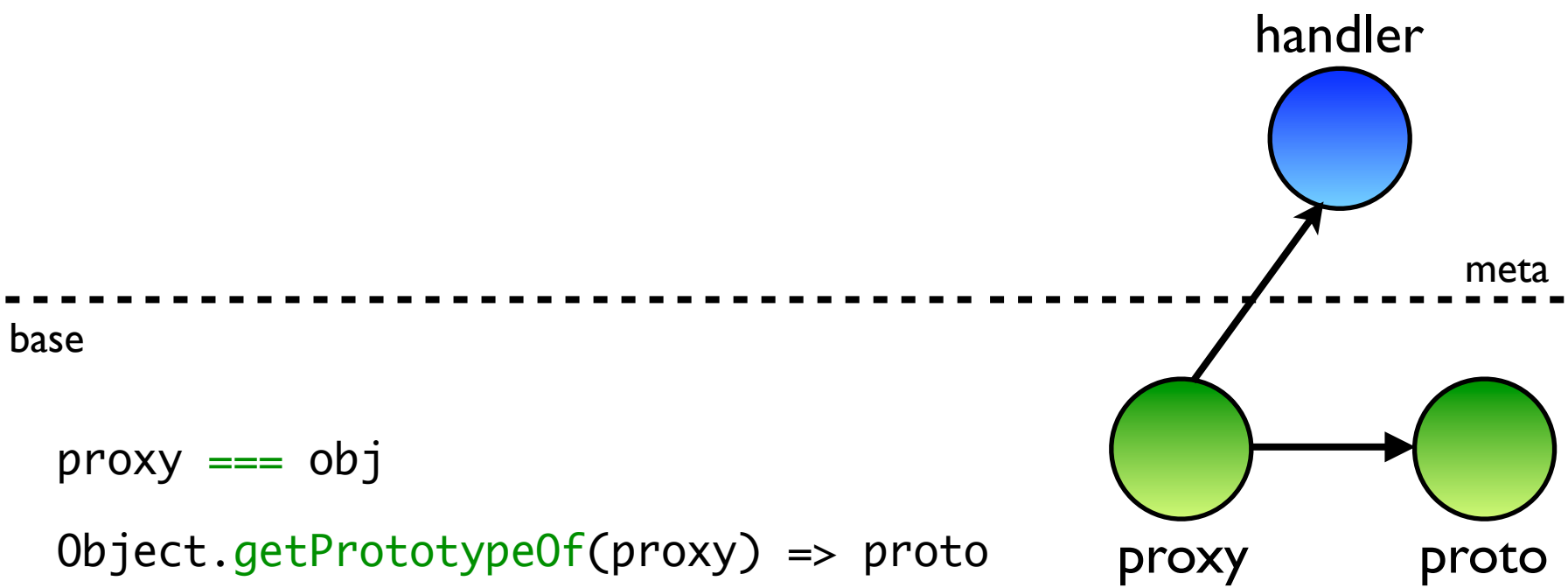



```
proxy === obj
Object.getPrototypeOf(proxy) => proto
proxy instanceof Fun
```

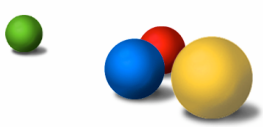


# But not quite everything either

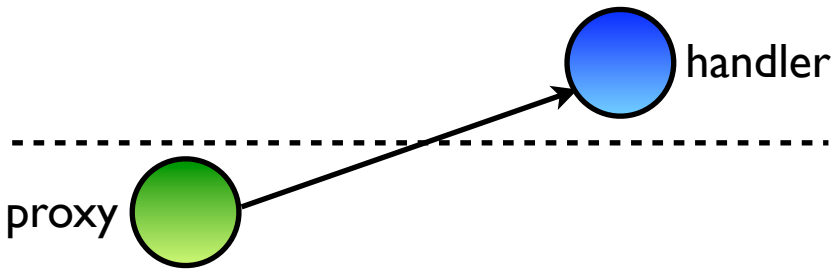
```
var proxy = Proxy.create(handler, proto);
```



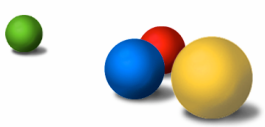

```
proxy === obj
Object.getPrototypeOf(proxy) => proto
proxy instanceof Fun
typeof proxy => “object”
```



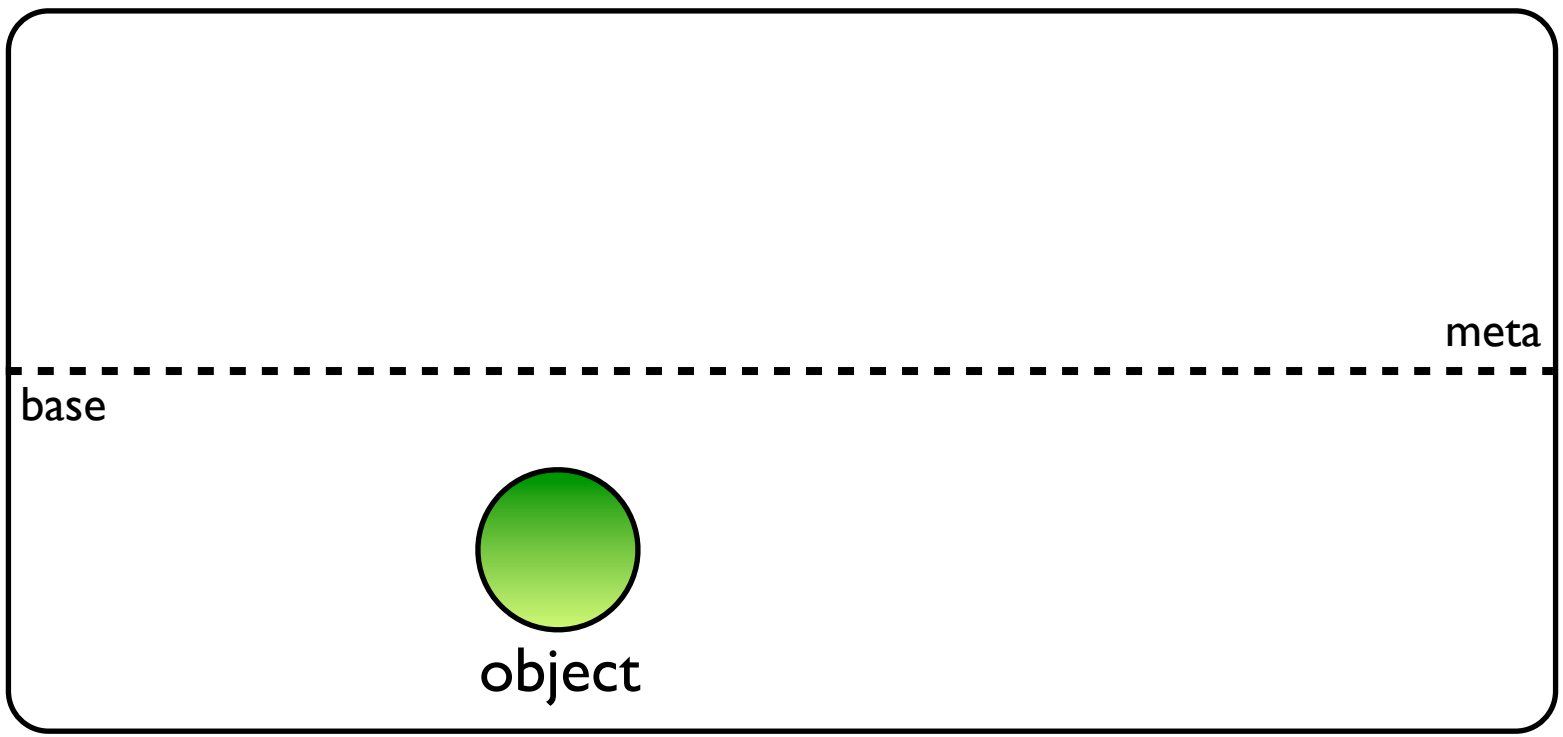
# Full Handler API

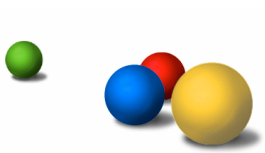



| | |
|---|---|
| `Object.getOwnPropertyDescriptor(proxy)` | `handler.getOwnPropertyDescriptor(name)` |
| `Object.getPropertyDescriptor(proxy)` | `handler.getPropertyDescriptor(name)` |
| `Object.defineProperty(proxy,name,pd)` | `handler.defineProperty(name, pd)` |
| `Object.getOwnPropertyNames(proxy)` | `handler.getOwnPropertyNames()` |
| `delete proxy.name` | `handler.delete(name)` |
| `for (name in proxy) { ... }` | `handler.enumerate()` |
| `Object.{freeze\|seal\|...}(proxy)` | `handler.fix()` |
| `name in proxy` | `handler.has(name)` |
| `({}).hasOwnProperty.call(proxy, name)` | `handler.hasOwn(name)` |
| `receiver.name` | `handler.get(receiver, name)` |
| `proxy.name = val` | `handler.set(proxy, name, val)` |
| `Object.keys(proxy)` | `handler.enumerateOwn()` |

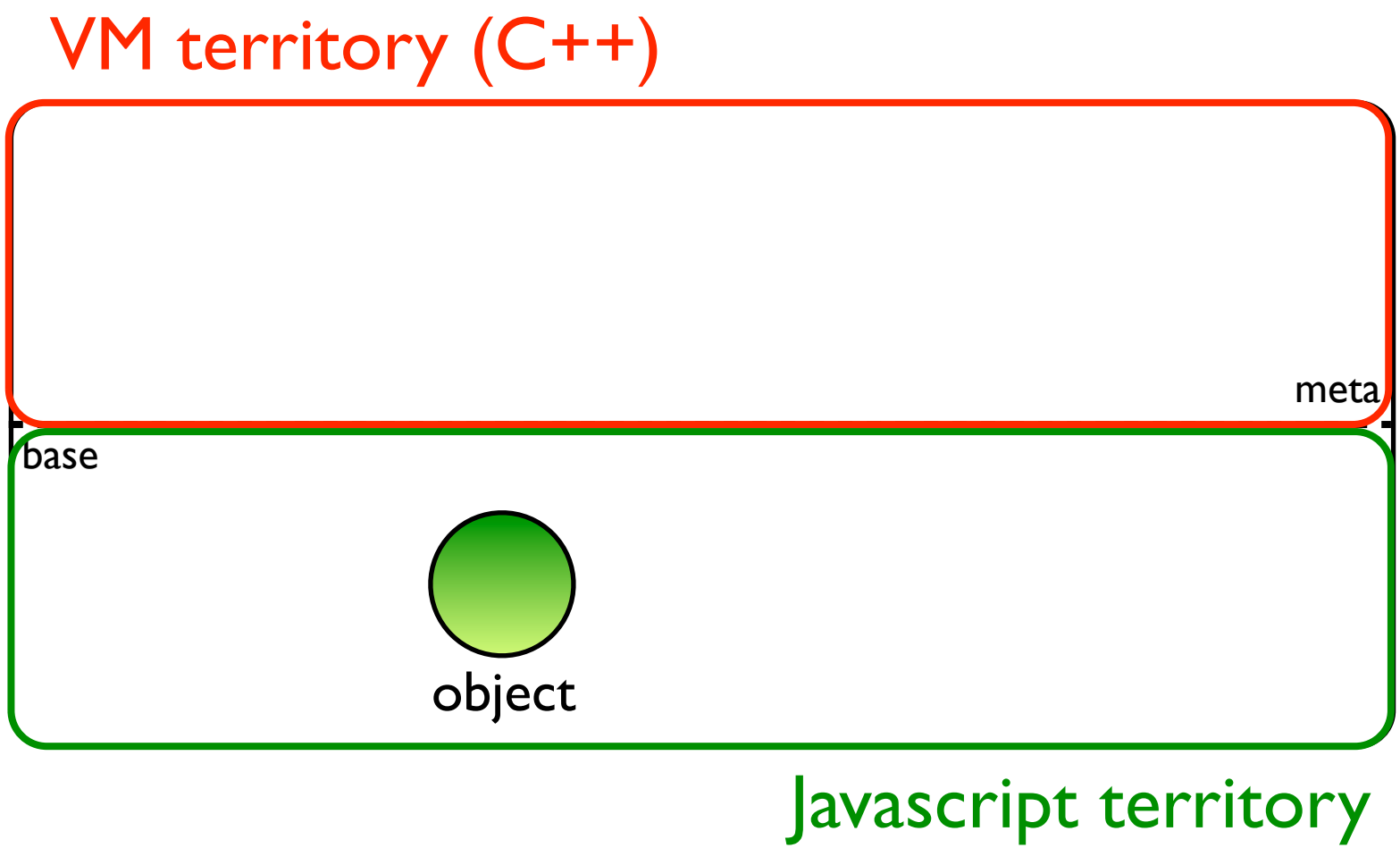


# Selective interception

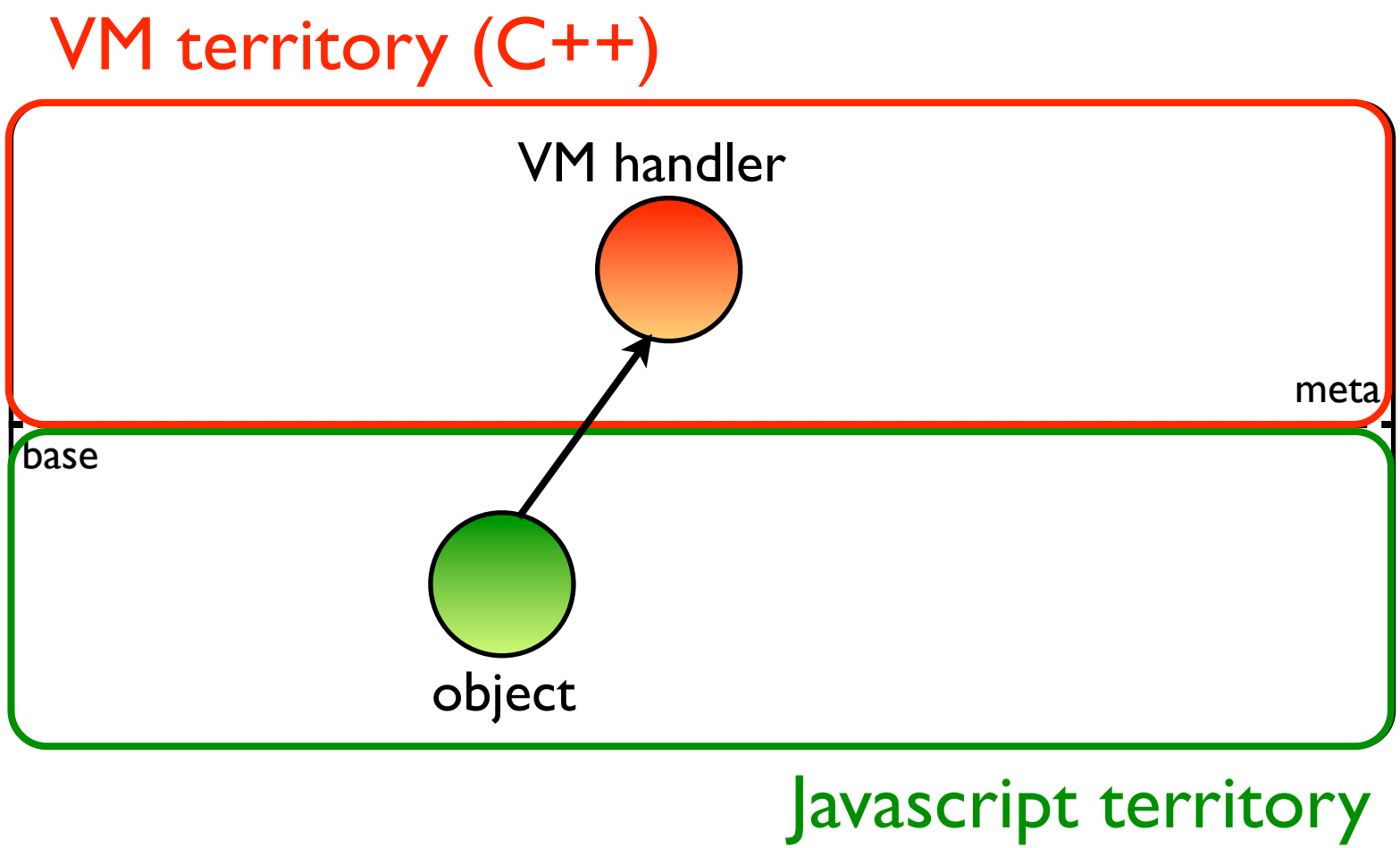


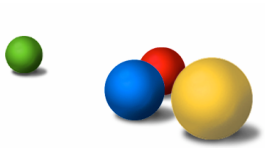



# Selective interception

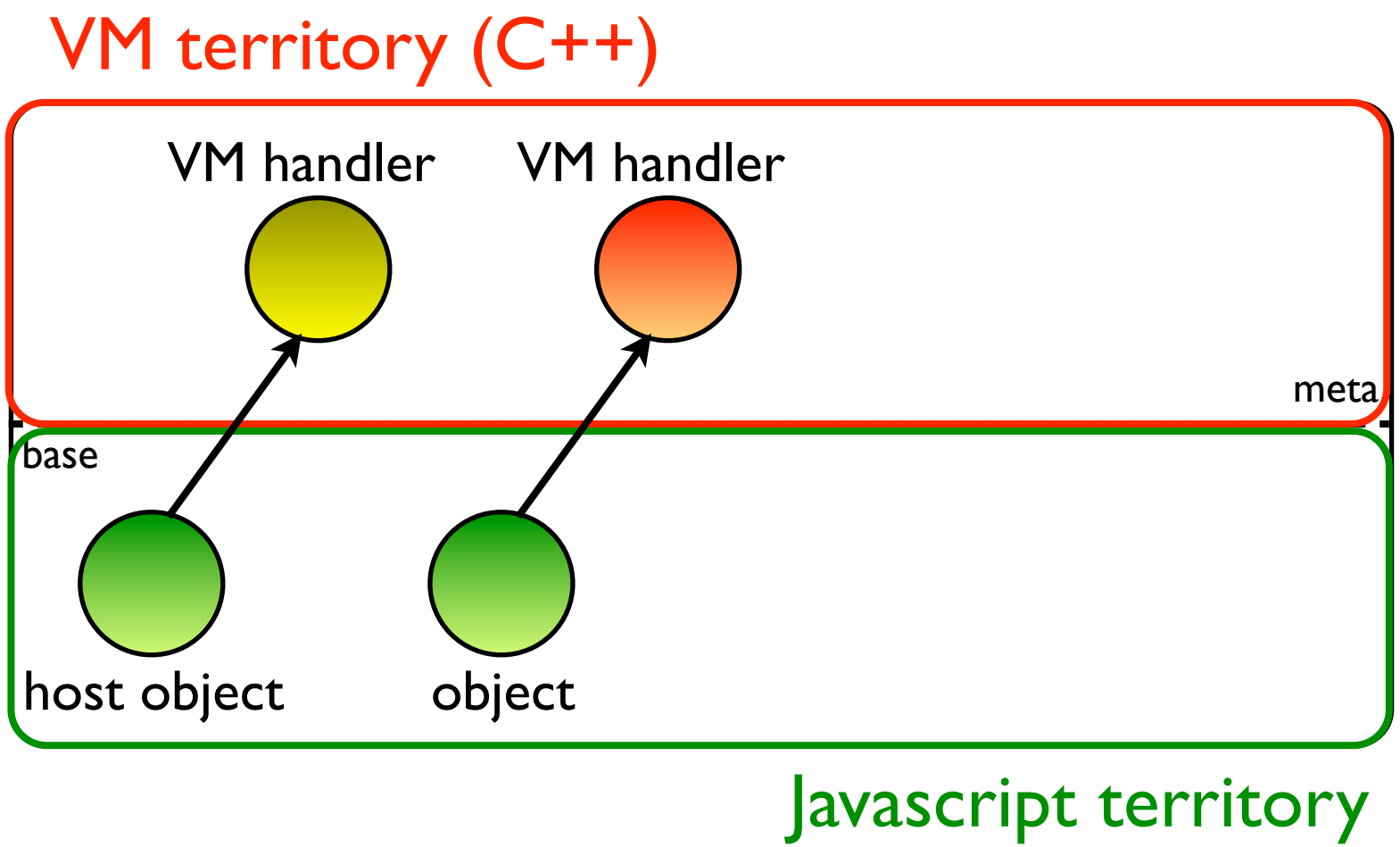


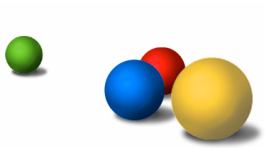
# Selective interception



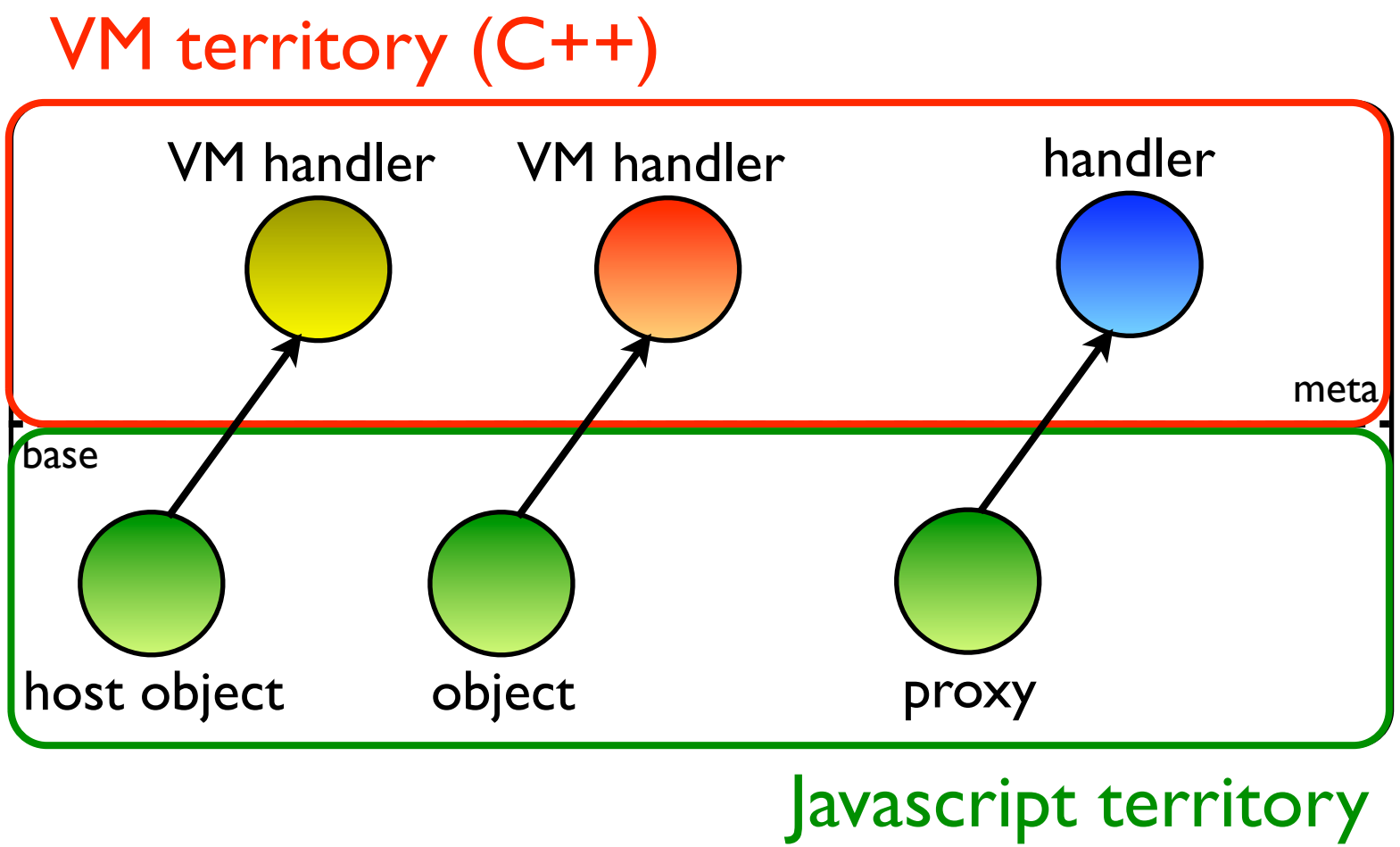



# Selective interception

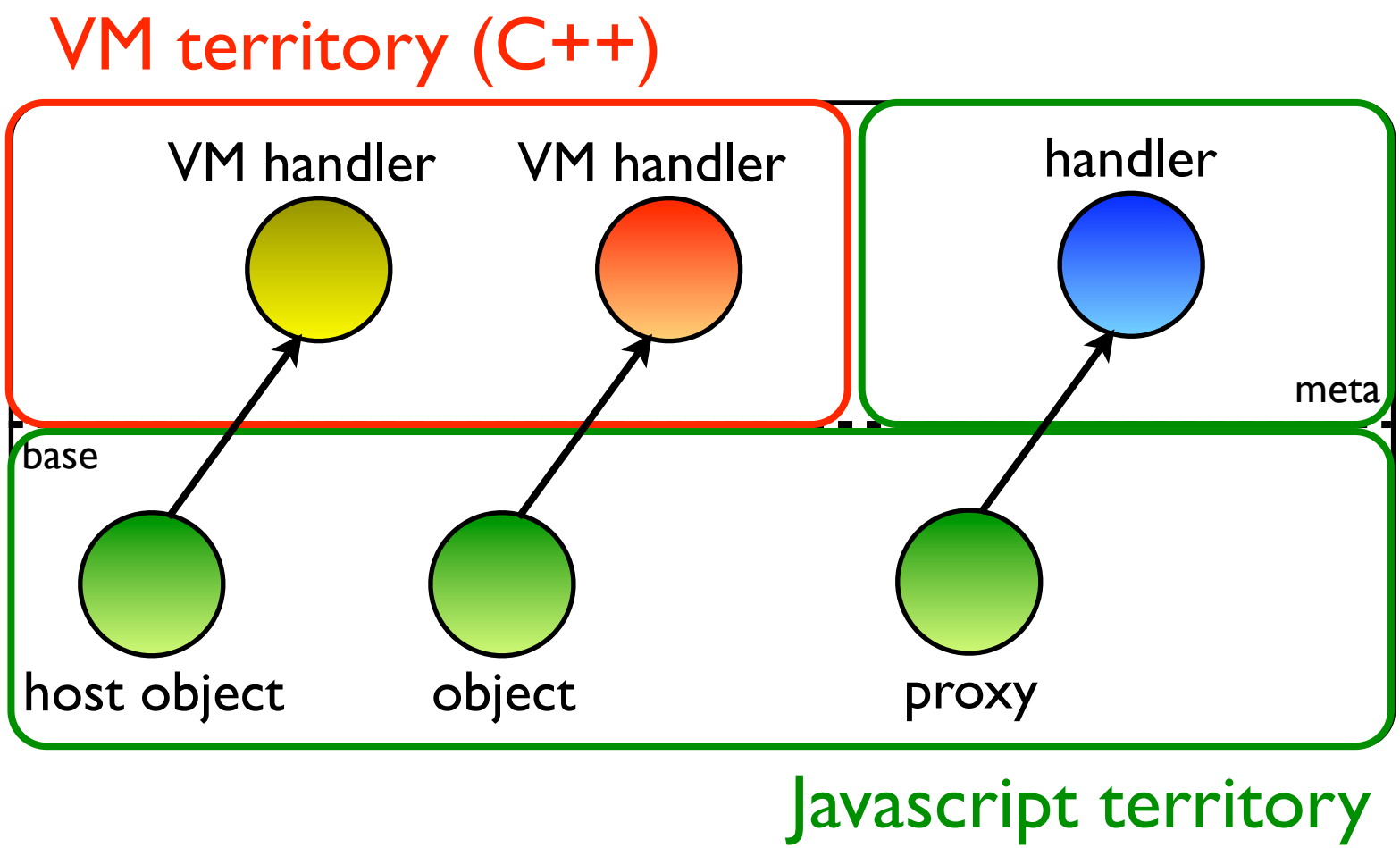


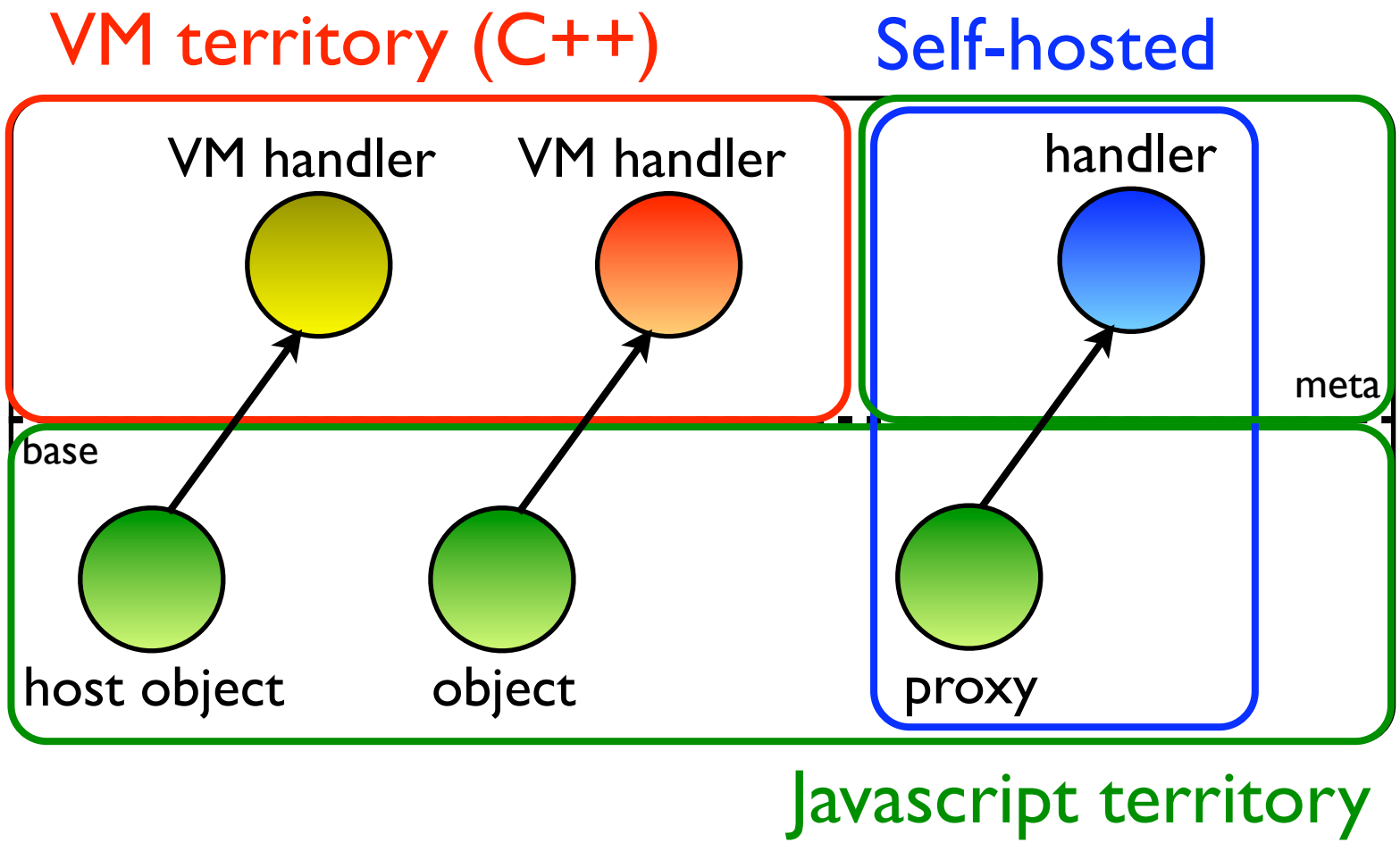
# Selective interception

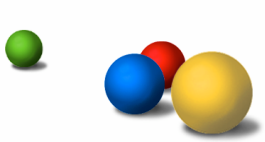


# Selective interception

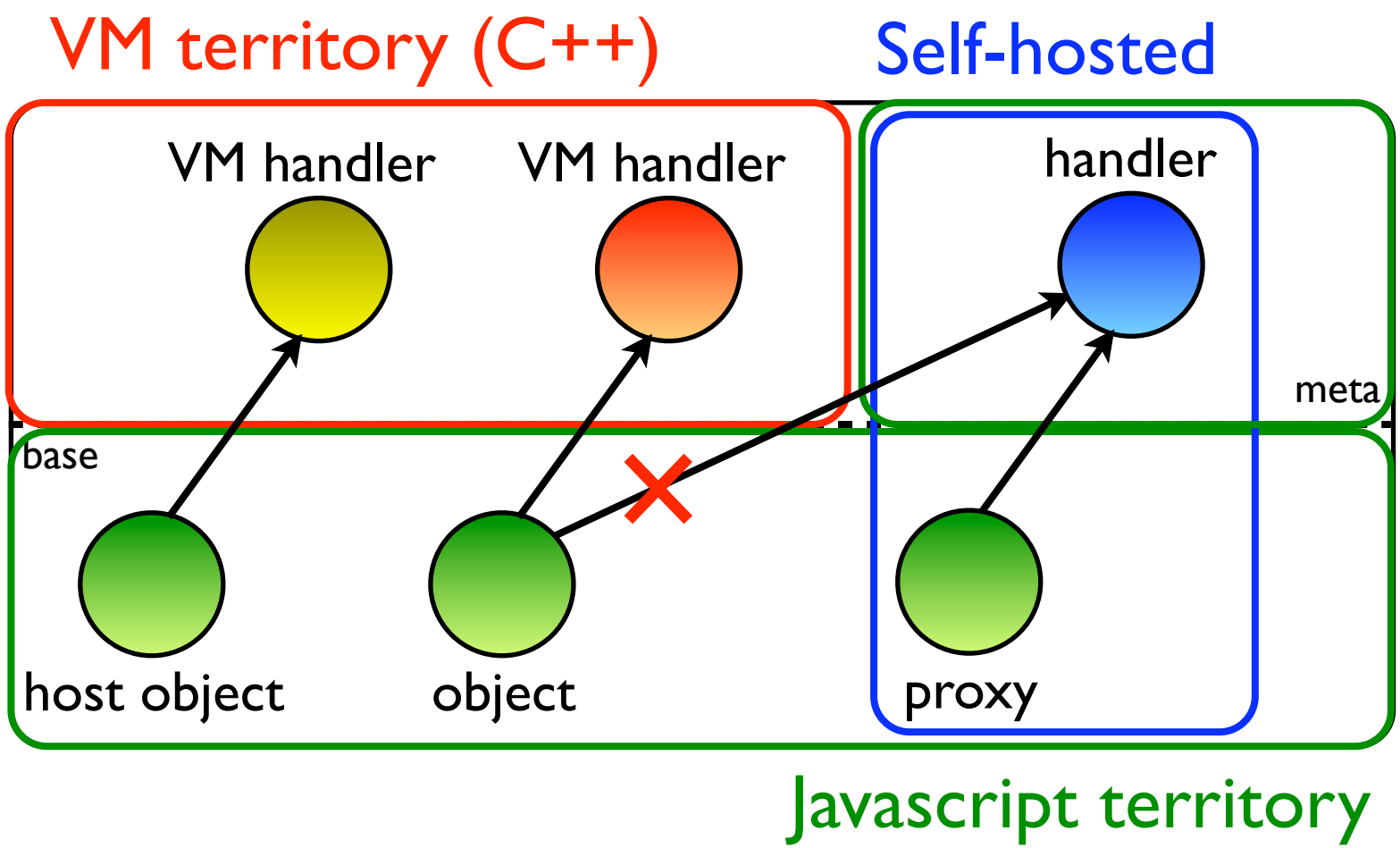


# Selective interception

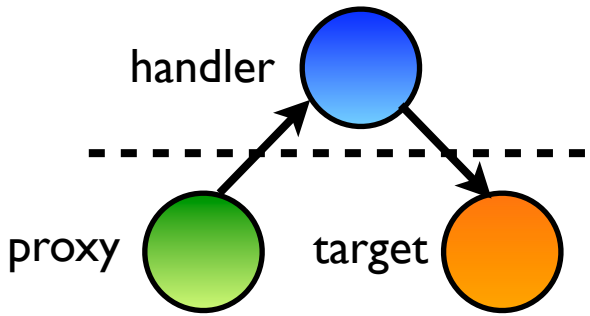


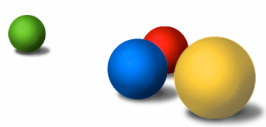
# Selective interception



# Example: no-op forwarding proxy

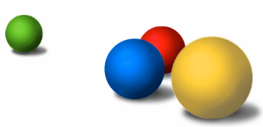



```
function ForwardingHandler(obj) {
  this.target = obj;
}
ForwardingHandler.prototype = {
  has: function(name) { return name in this.target; },
  get: function(rcvr,name) { return this.target[name]; },
  set: function(rcvr,name,val) { this.target[name]=val;return true; },
  delete: function(name) { return delete this.target[name]; }
  enumerate: function() {
    var props = [];
    for (name in this.target) { props.push(name); };
    return props;
  },
  ...
}

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



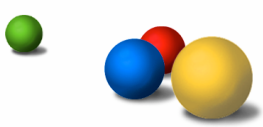
# Example: counting property access

```
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

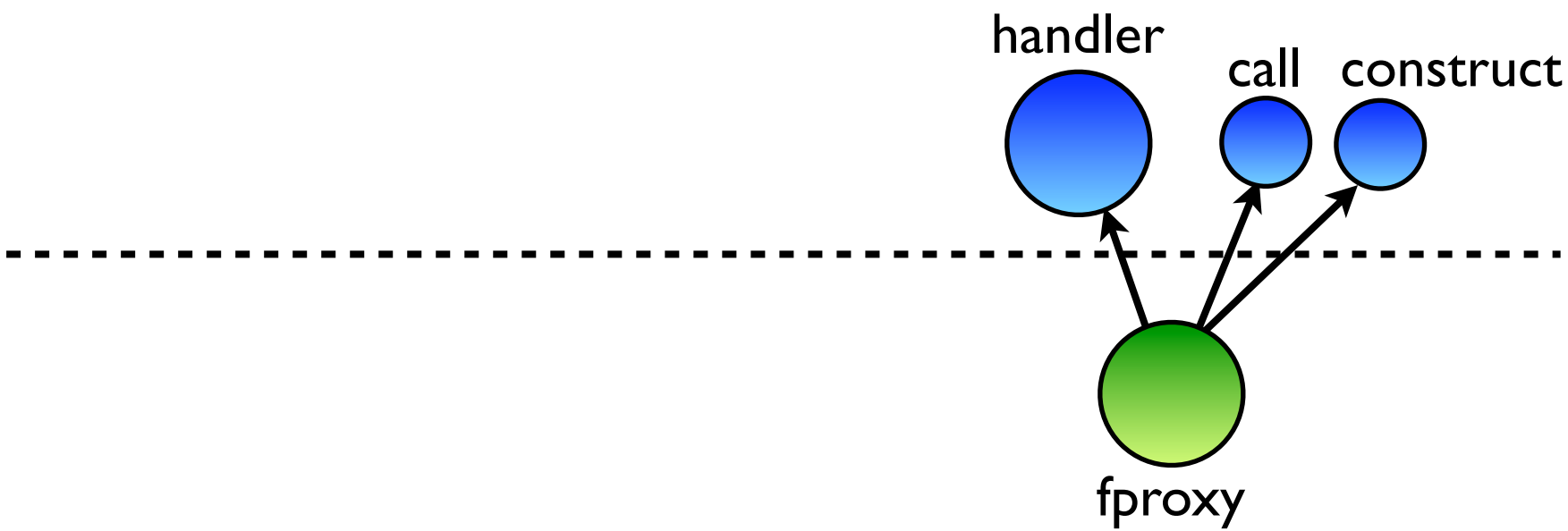
```
function makeSimpleProfiler(target) {
  var forwarder = new ForwardingHandler(target);
  var count = Object.create(null);
  forwarder.get = function(rcvr, name) {
    count[name] = (count[name] || 0) + 1;
    return this.target[name];
  };
  return {
    proxy: Proxy.create(forwarder,
          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

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

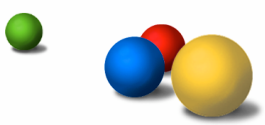


# Function Proxies

Javascript functions are objects. Additionally, they are also 'callable' and 'constructable'

```
var call = function() { ... };
var construct = function() { ... };
var fproxy = Proxy.createFunction(handler, call, construct);
```



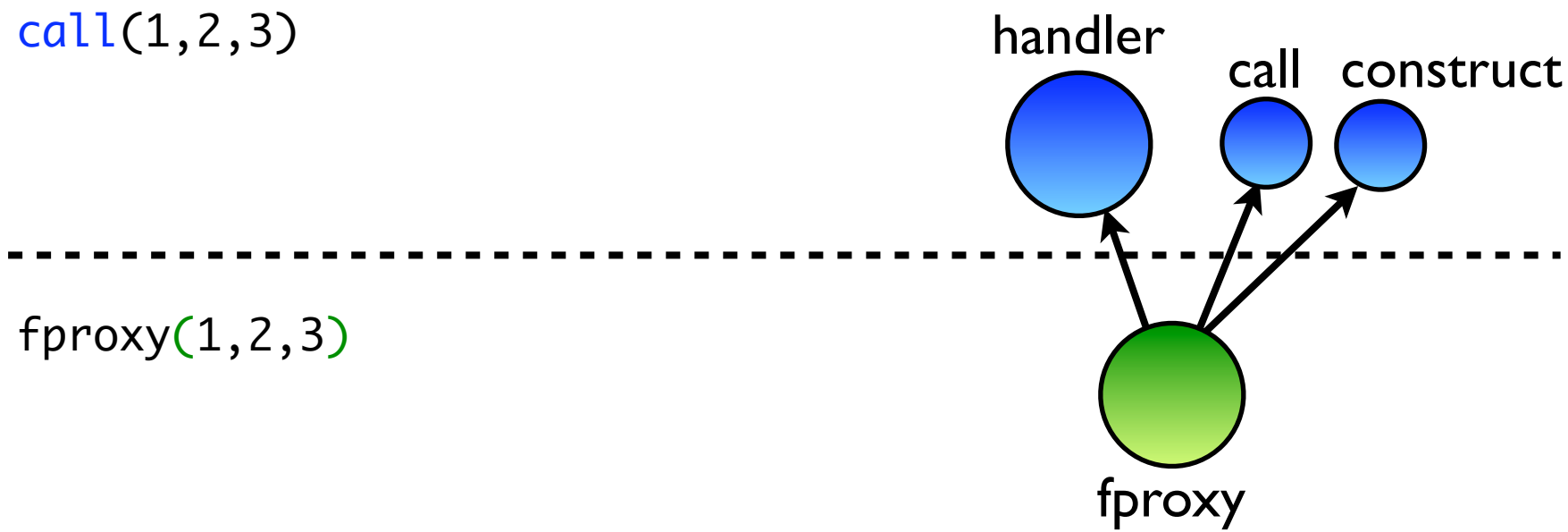



# Function Proxies

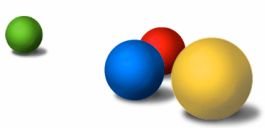
Javascript functions are objects. Additionally, they are also 'callable' and 'constructable'

```
var call = function() { ... };
var construct = function() { ... };
var fproxy = Proxy.createFunction(handler, call, construct);
```

```
call(1,2,3)
```




```
fproxy(1,2,3)
```

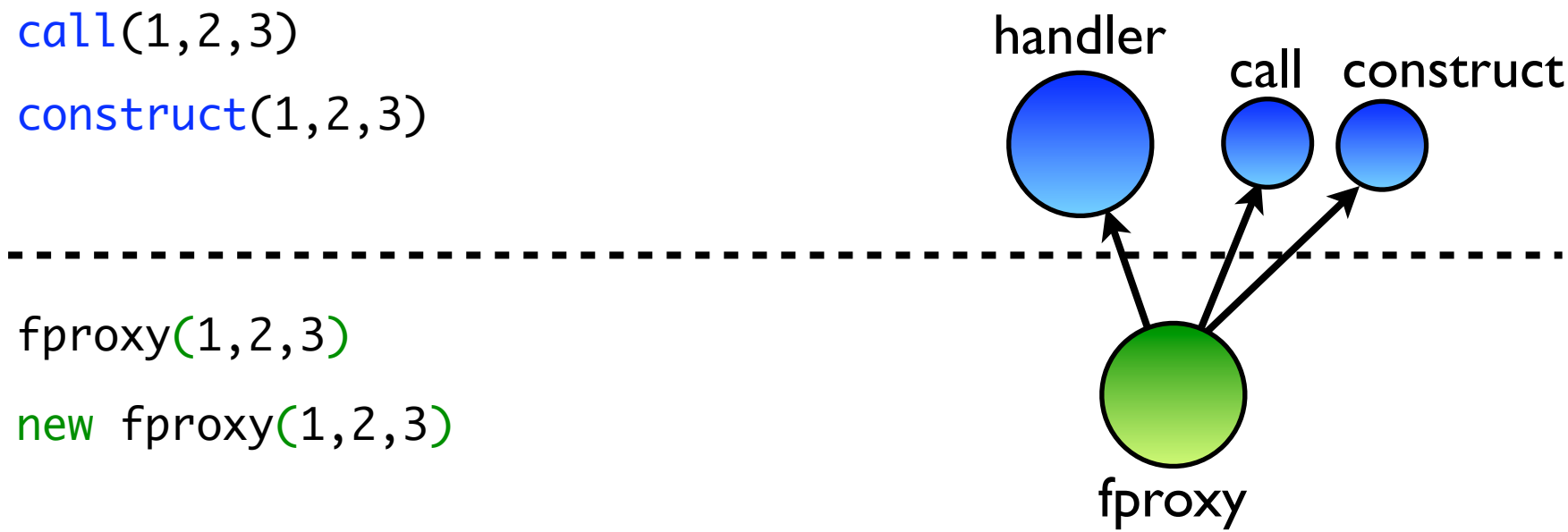


# Function Proxies

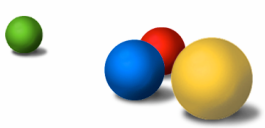
Javascript functions are objects. Additionally, they are also 'callable' and 'constructable'

```
var call = function() { ... };
var construct = function() { ... };
var fproxy = Proxy.createFunction(handler, call, construct);
```

```
call(1,2,3)
construct(1,2,3)
```




```
fproxy(1,2,3)
new fproxy(1,2,3)
```

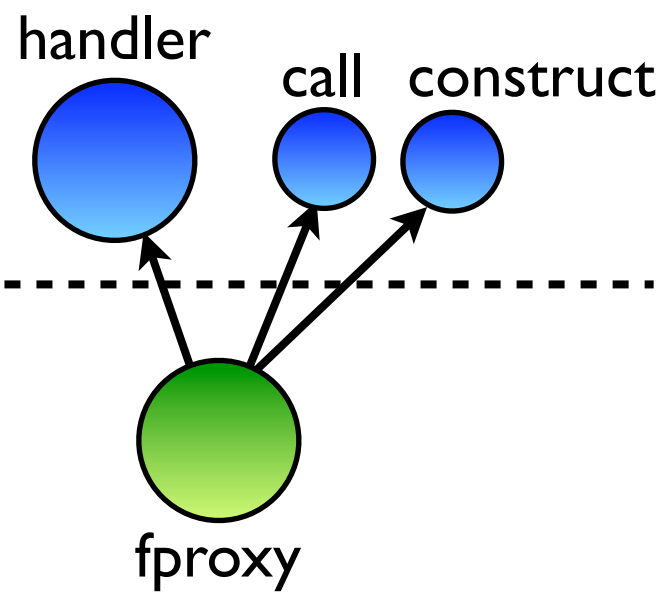


# Function Proxies

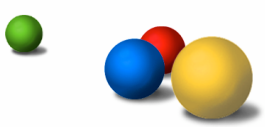
Javascript functions are objects. Additionally, they are also 'callable' and 'constructable'

```
var call = function() { ... };
var construct = function() { ... };
var fproxy = Proxy.createFunction(handler, call, construct);
```

```
call(1,2,3)
construct(1,2,3)
handler.get(fproxy,'prototype')
```




```
fproxy(1,2,3)
new fproxy(1,2,3)
fproxy.prototype
```

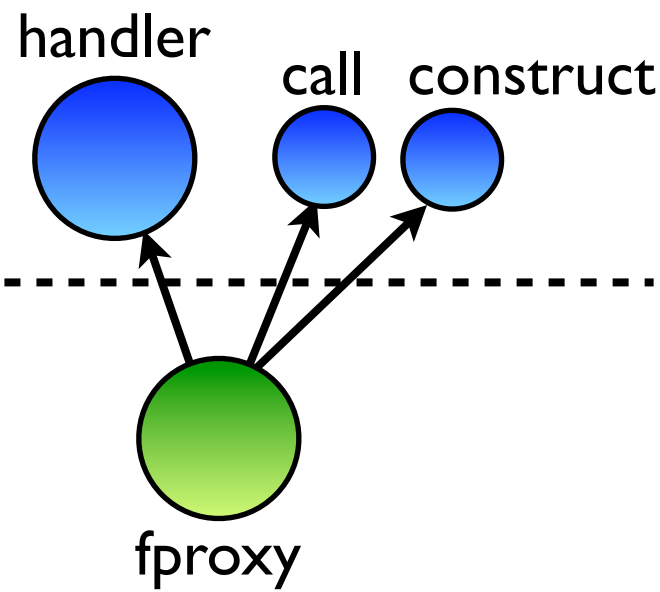


# Function Proxies

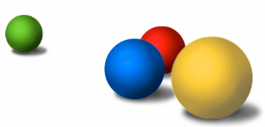
Javascript functions are objects. Additionally, they are also 'callable' and 'constructable'

```
var call = function() { ... };
var construct = function() { ... };
var fproxy = Proxy.createFunction(handler, call, construct);
```

```
call(1,2,3)
construct(1,2,3)
handler.get(fproxy,'prototype')
```




```
fproxy(1,2,3)
new fproxy(1,2,3)
fproxy.prototype
typeof fproxy => "function"
```

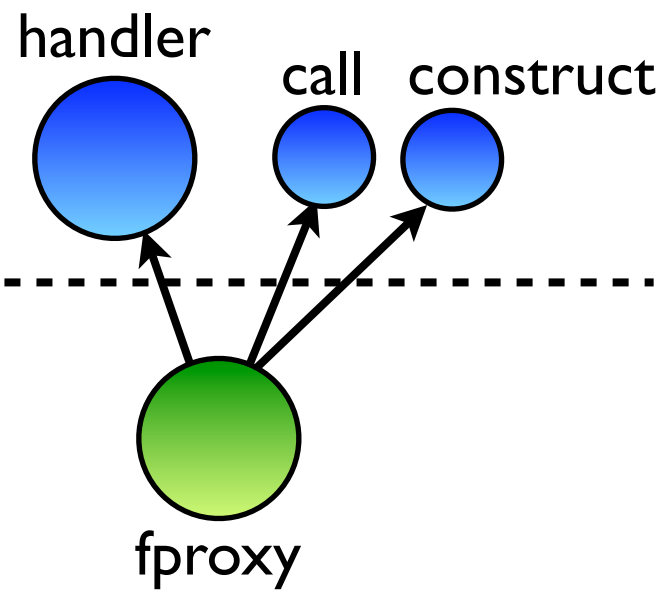


# Function Proxies

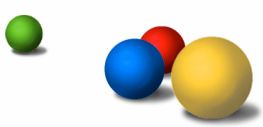
Javascript functions are objects. Additionally, they are also 'callable' and 'constructable'

```
var call = function() { ... };
var construct = function() { ... };
var fproxy = Proxy.createFunction(handler, call, construct);
```

```
call(1,2,3)
construct(1,2,3)
handler.get(fproxy,'prototype')
```

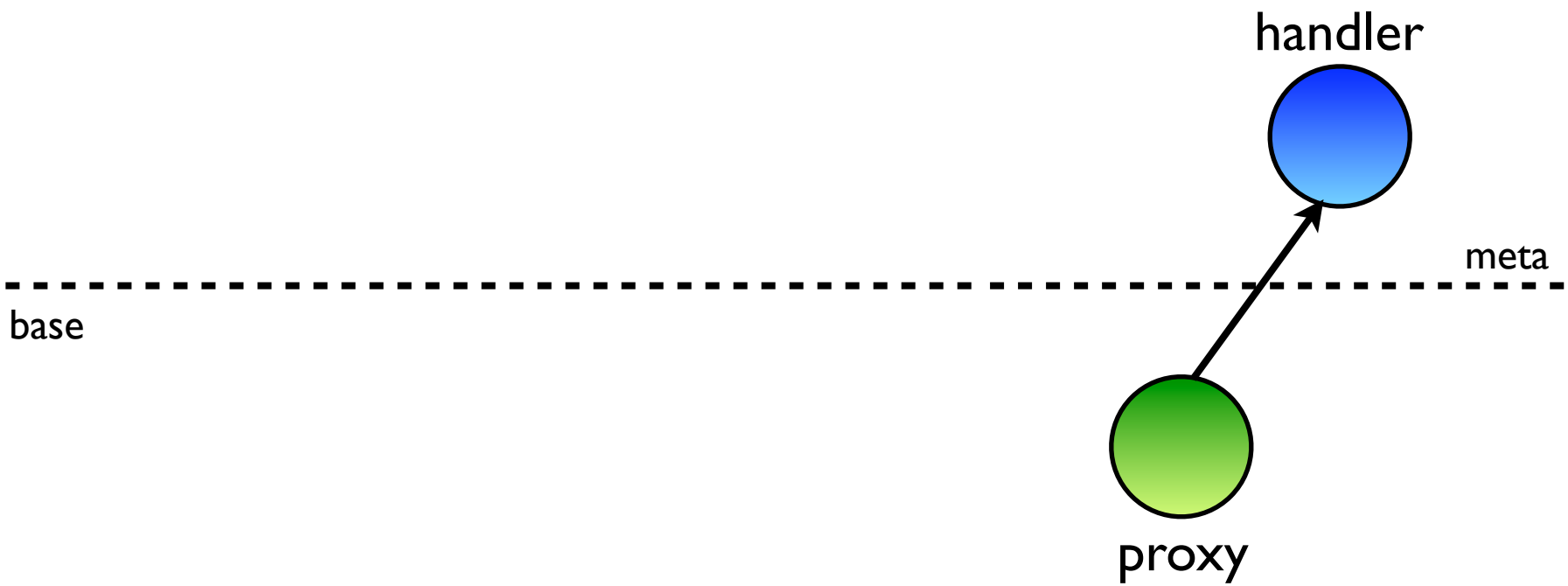


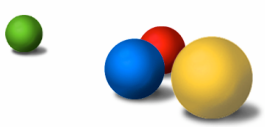

```
fproxy(1,2,3)
new fproxy(1,2,3)
fproxy.prototype
typeof fproxy => "function"
Object.getPrototypeOf(fproxy) => Function.prototype
```



# Fixing a Proxy

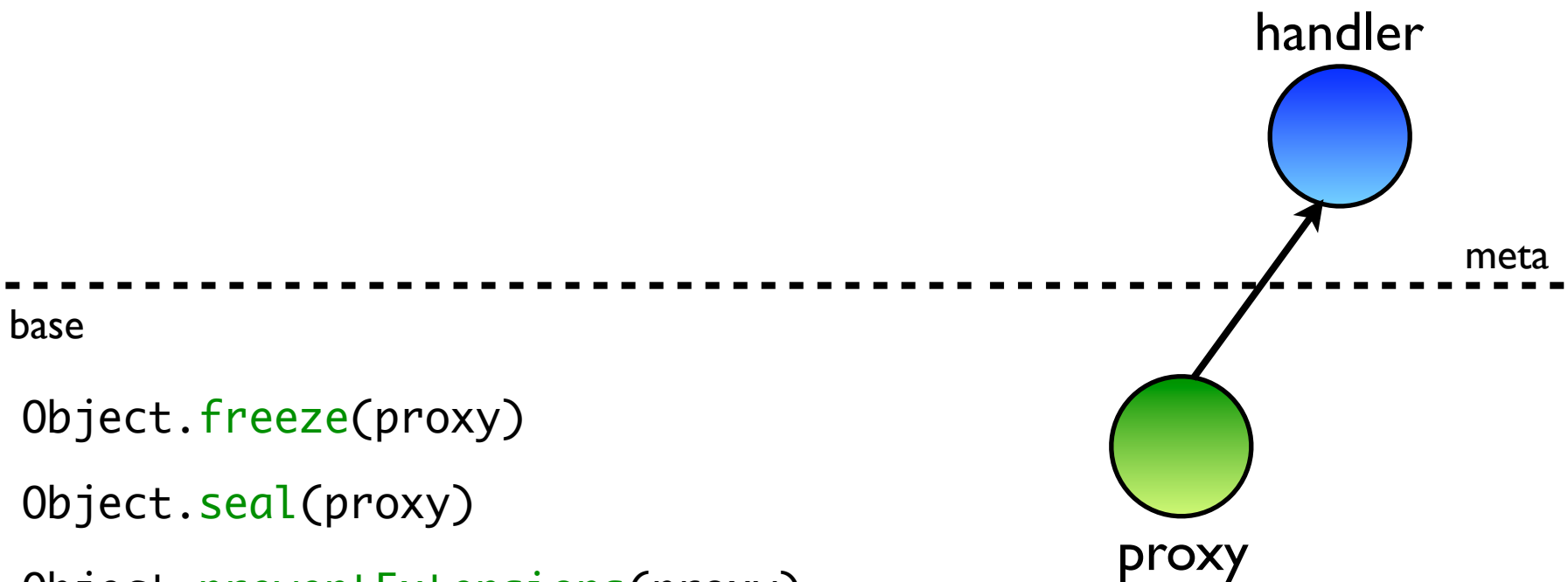
```
var proxy = Proxy.create(handler, proto);
```



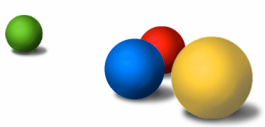



# Fixing a Proxy

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



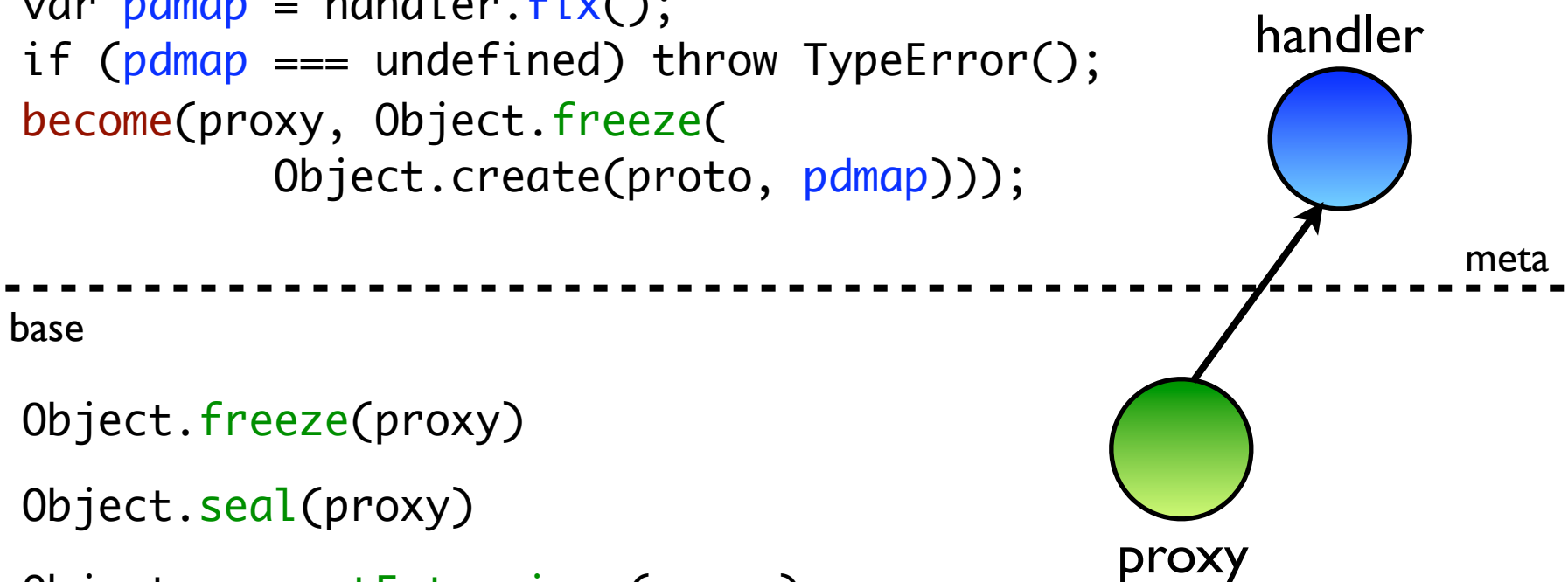

```
Object.freeze(proxy)
Object.seal(proxy)
Object.preventExtensions(proxy)
```



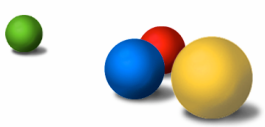
# Fixing a Proxy

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

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



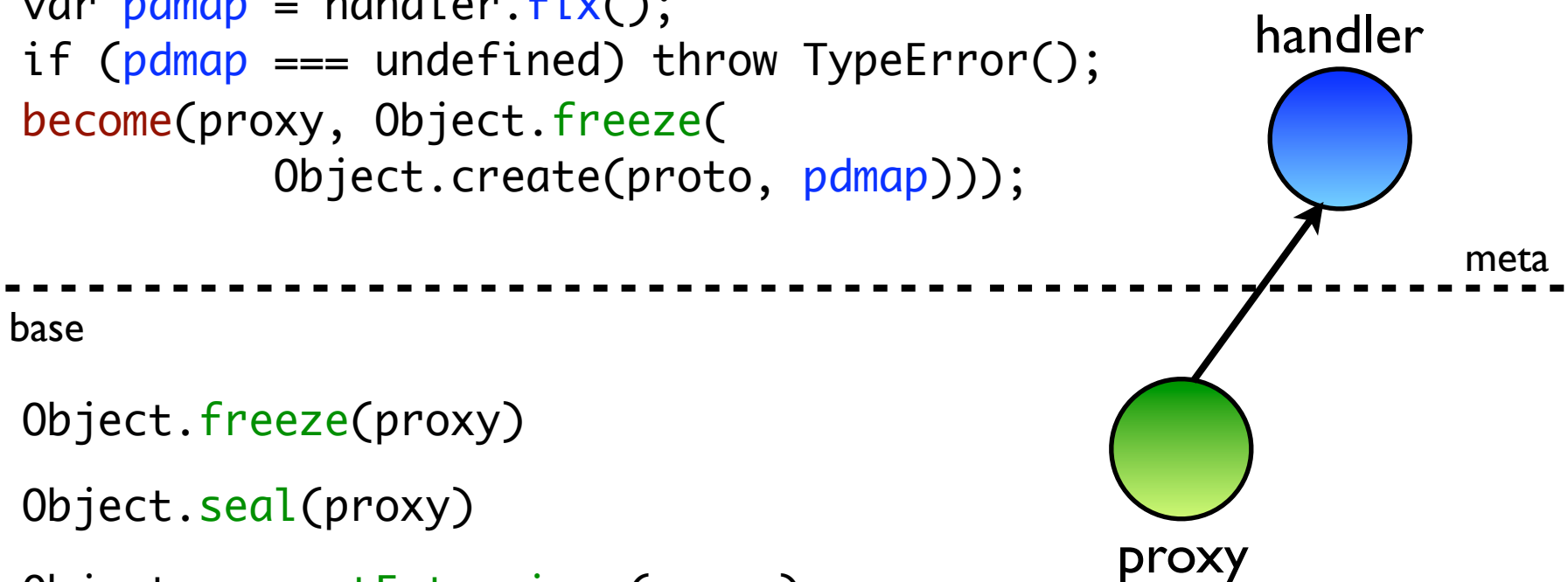

```
Object.freeze(proxy)

Object.seal(proxy)

Object.preventExtensions(proxy)
```



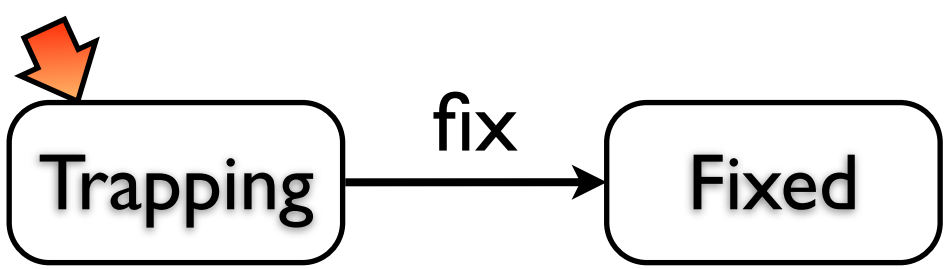
# Fixing a Proxy

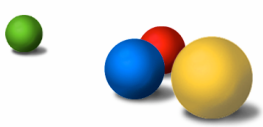
```
var proxy = Proxy.create(handler, proto);
```

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




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



# Fixing a Proxy

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

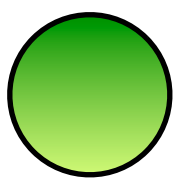
```
var pdmap = handler.fix();
if (pdmap === undefined) throw TypeError();
become(proxy, Object.freeze(
          Object.create(proto, pdmap)));
```

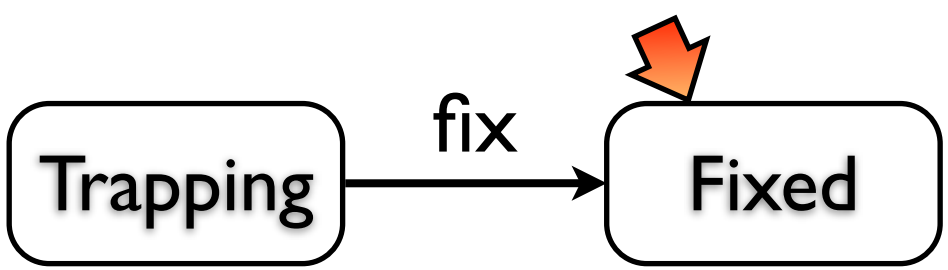
meta

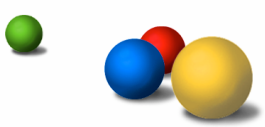
---

base

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

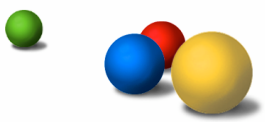


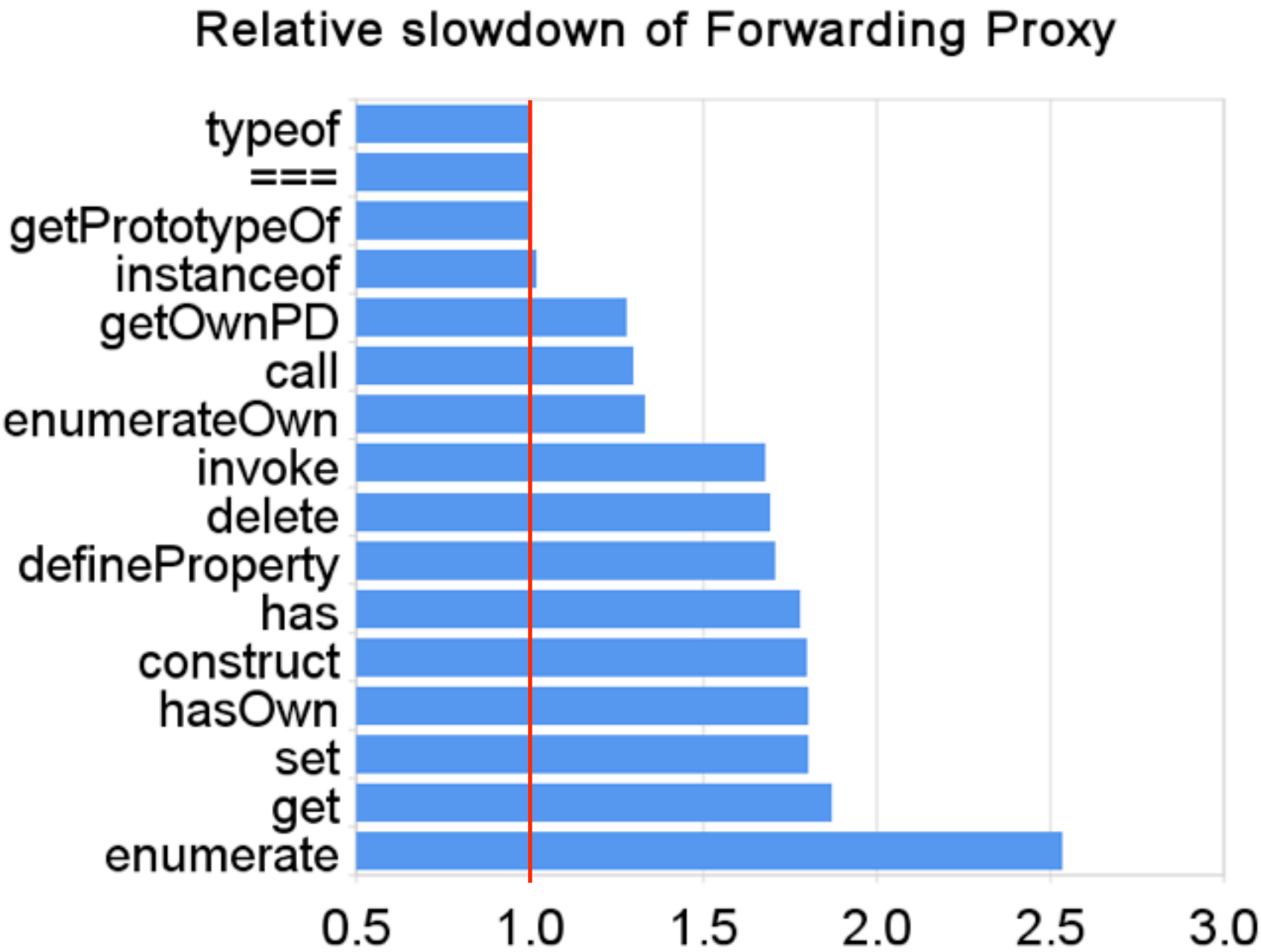



# Status

- Presented at ECMA TC-39 meetings
- Proposal for ES-Harmony
- tinyurl.com/harmony-proxies
- Andreas Gal (Mozilla) implemented a prototype as an extension of Tracemonkey

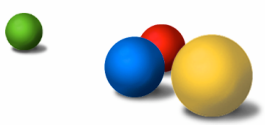


# Micro-benchmark

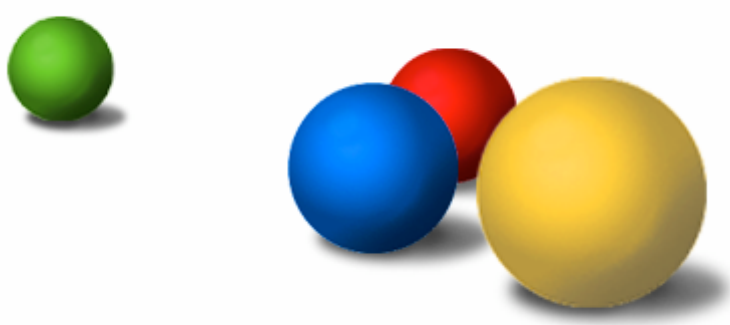


# Proxies: summary

- Dynamic proxies enable:
  - Generic wrappers: access control, profiling, adaptors, ...
  - Virtual objects: persistent objects, remote 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

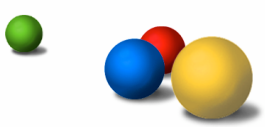


# Traits



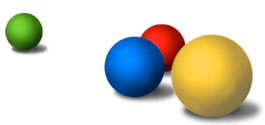
# Traits in a nutshell

- An alternative to mixins & multiple inheritance
- A trait provides and requires a set of methods
- Robust composition:
  - Name clashes lead to conflicts that must be resolved
  - Trait composition is commutative & associative: composition order is irrelevant
- First in Squeak Smalltalk (~2003), adoption in Perl 6, PLT Scheme, Fortress, ...



# traits.js library

- Motivation:
  - More robust than existing prototypical inheritance & mixin patterns
  - Easier creation of tamper-proof objects
- Based on ES5 property descriptor API
- Backwards-compatible with ES3
- Works in browser and stand-alone



# Minimal core API

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

## Trait construction

```
var t = Trait({
  a: Trait.required,
  b: function(){ return this.a + 1; }
})
```

## Trait composition

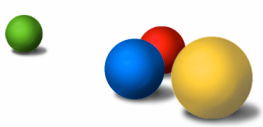
```
var tc = Trait.compose(t1, t2, ...)
```

## Trait resolution

```
var tr = Trait.resolve({ a: 'c',
                         b: undefined }, t)
```

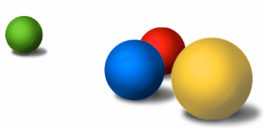
## Trait instantiation

```
var o = Trait.create(proto, t)
```



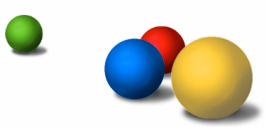
# Enumerable trait

```
var EnumerableTrait = Trait({
  forEach: Trait.required,
  map: function(fun) {
    var seq = [];
    this.forEach(function(e) { seq.push(fun(e)); });
    return seq;
  },
  filter: function(pred) {
    var seq = [];
    this.forEach(function(e) {
      if (pred(e,i)) { seq.push(e); }
    });
    return seq;
  },
  reduce: function(init, fun) {
    var result = init;
    this.forEach(function(e) { result = fun(result, e); });
    return result;
  }
});
```



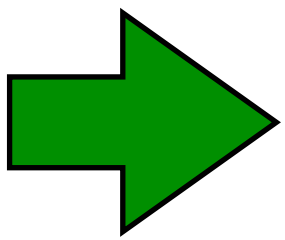
# Enumerable interval

```
function makeInterval(min, max) {
  return Trait.create(Object.prototype,
    Trait.compose(
      EnumerableTrait,
      Trait({
        start: min,
        end: max,
        contains: function(e) { return (min <= e) && (e < max); },
        forEach: function(consumer) {
          for (var i = min; i < max; i++) { consumer(i); }
        }
      })));
}


var int = makeInterval(0,5);
int.reduce(0, function(a,b){return a+b;}); // 10
```



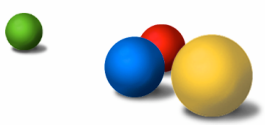
# Trait construction

```
Trait({
    a: Trait.required,
    b: function() {...},
    c: 42,
    get d() {...}
});
```



```
{ a: {
    value: undefined,
    required: true,
    enumerable: false
  },
  b: {
    value: function() {...},
    method: true,
    enumerable: true
  },
  c: {
    value: 42,
    enumerable: true
  },
  d: {
    get: function(){...},
    enumerable: true
  }
}
```

Traits are property descriptor maps



# Trait composition: conflicts

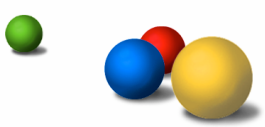
T2

| | |
|---|---|
| a | <methodP> |
| b | <methodP> |

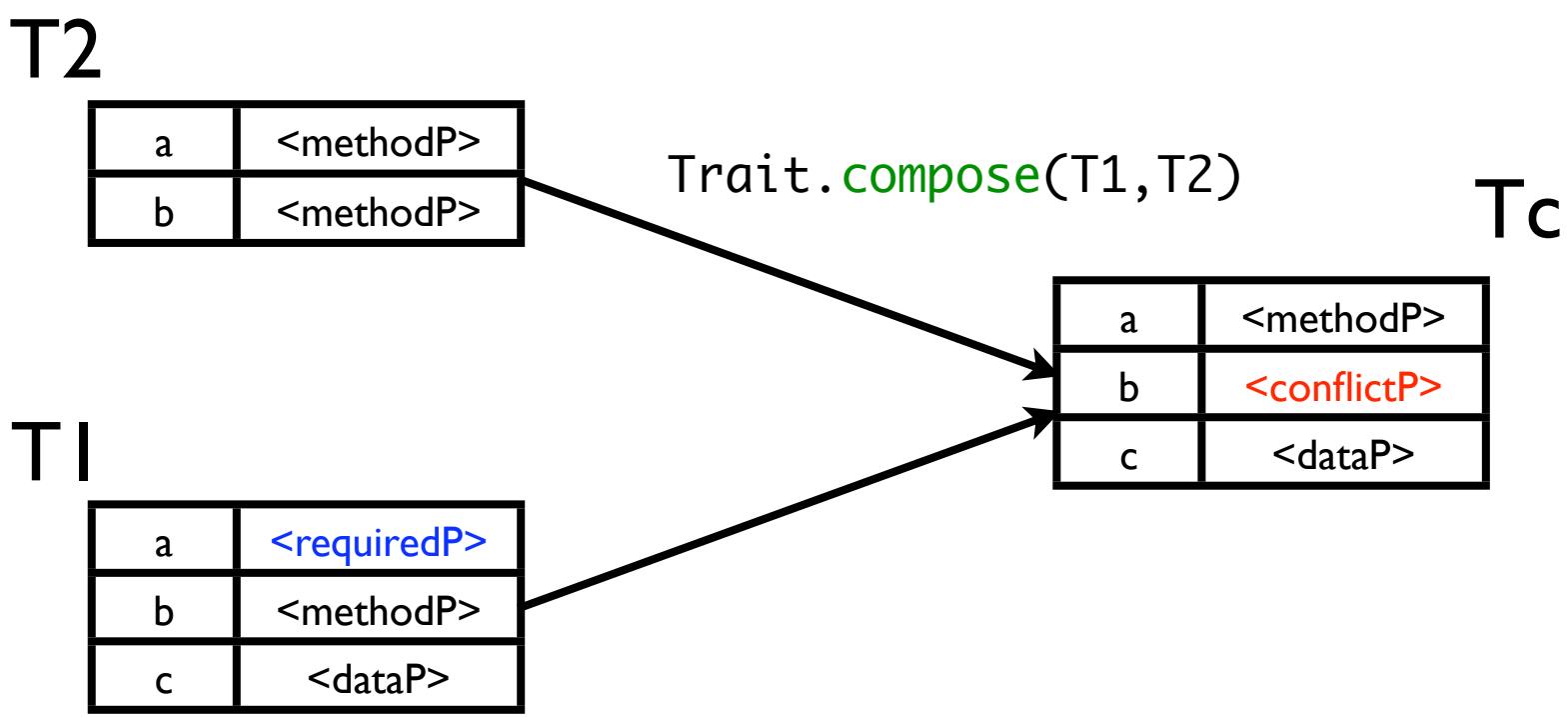
T1

| | |
|---|---|
| a | <requiredP> |
| b | <methodP> |
| c | <dataP> |

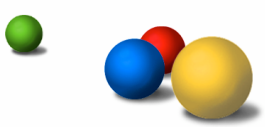
```
Trait({
  a: Trait.required,
  b: function() { ...this.a... },
  c: 42
})
```



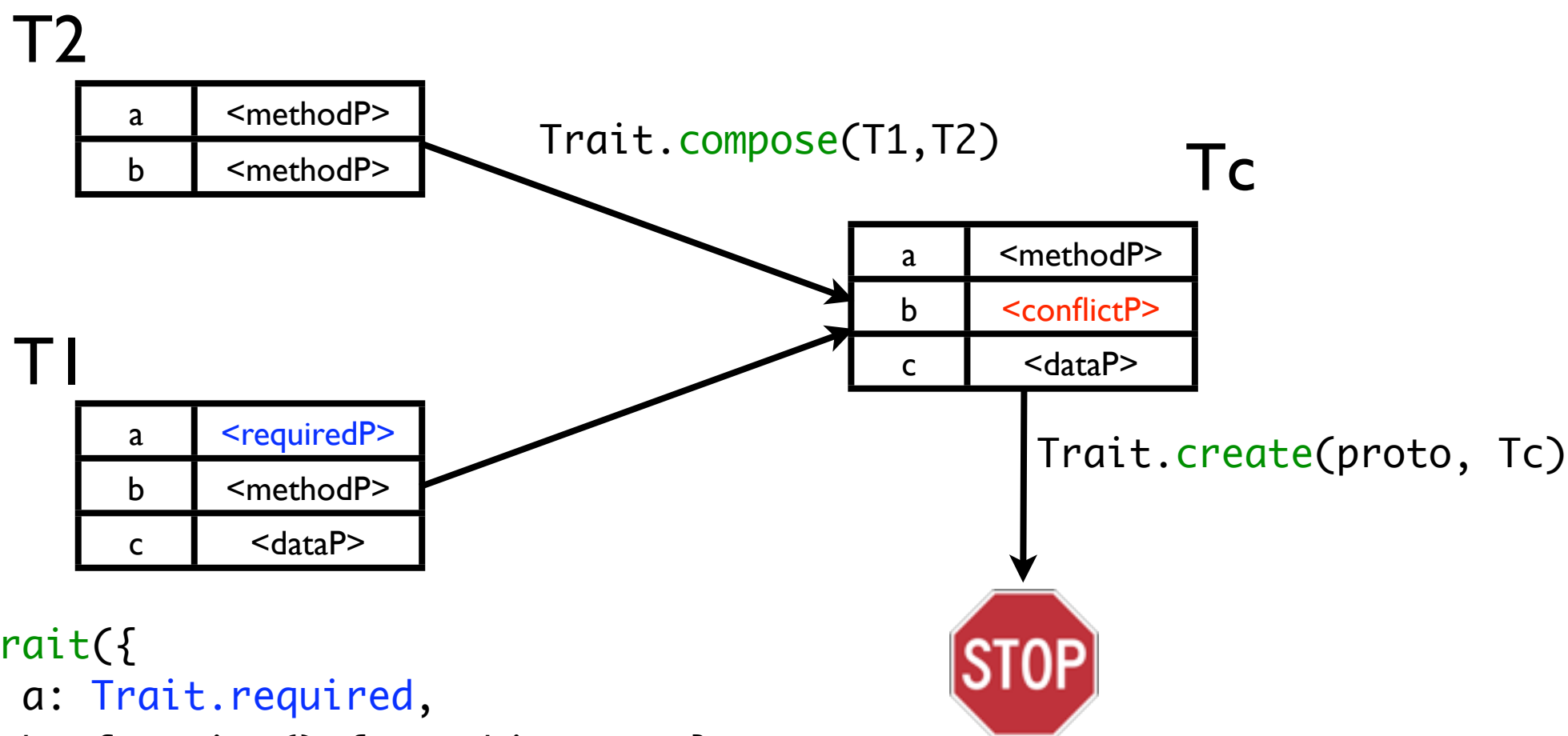
# Trait composition: conflicts



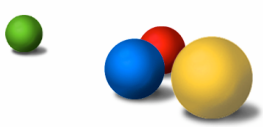

```
Trait({
  a: Trait.required,
  b: function() { ...this.a... },
  c: 42
})
```



# Trait composition: conflicts




```
Trait({
  a: Trait.required,
  b: function() { ...this.a... },
  c: 42
})
```



# Trait resolution

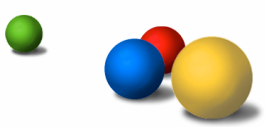
T2

| | |
|---|---|
| a | <methodP> |
| b | <methodP> |

T1

| | |
|---|---|
| a | <requiredP> |
| b | <methodP> |
| c | <dataP> |

```
Trait({
  a: Trait.required,
  b: function() { ...this.a... },
  c: 42
})
```



# Trait resolution

```
Trait.resolve(
  {b:undefined}, T2);
```

T2

| | |
|---|---|
| a | <methodP> |
| b | <methodP> |

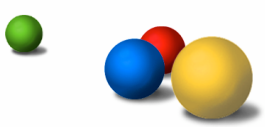
Tr

| | |
|---|---|
| a | <methodP> |
| b | <requiredP> |

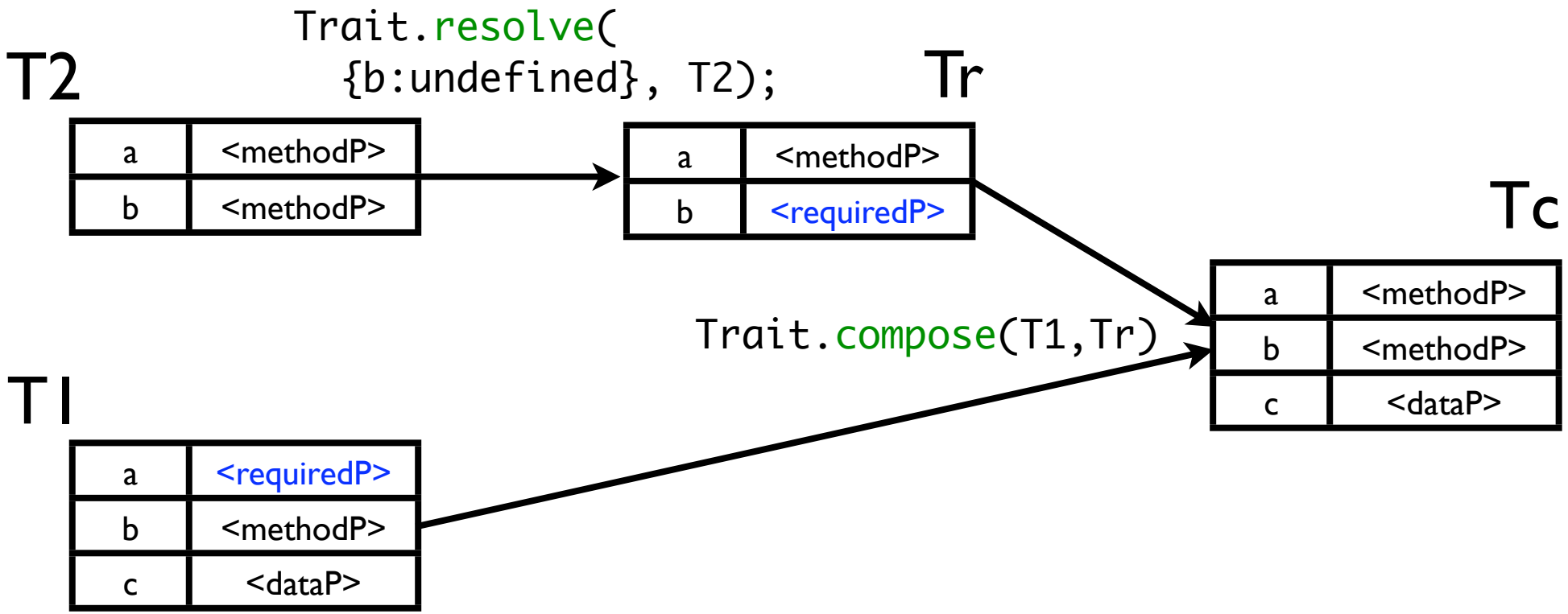
T1

| | |
|---|---|
| a | <requiredP> |
| b | <methodP> |
| c | <dataP> |

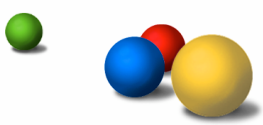
```
Trait({
  a: Trait.required,
  b: function() { ...this.a... },
  c: 42
})
```



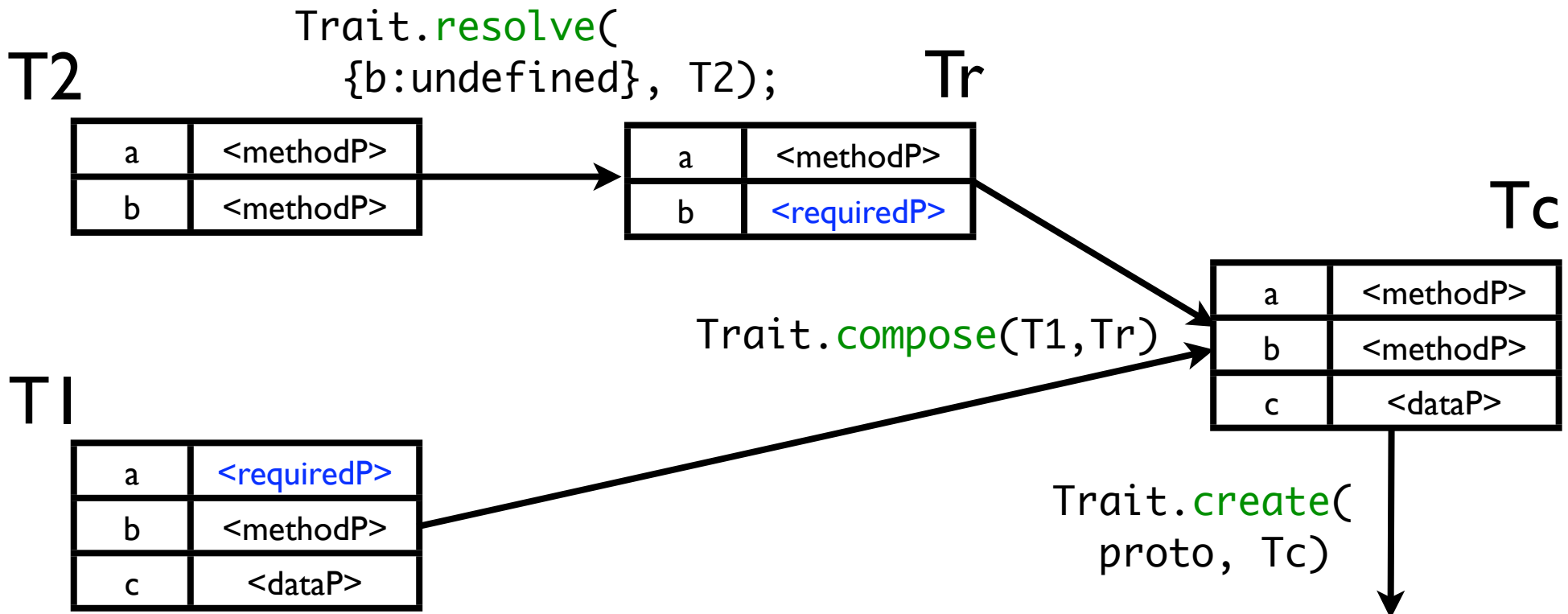
# Trait resolution



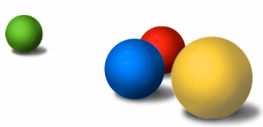

```
Trait({
  a: Trait.required,
  b: function() { ...this.a... },
  c: 42
})
```



# Trait resolution

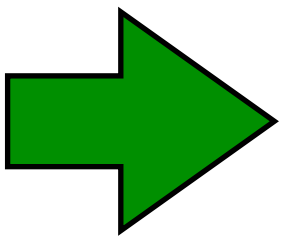



```
Trait({
  a: Trait.required,
  b: function() { ...this.a... },
  c: 42
})
```



# Trait instantiation

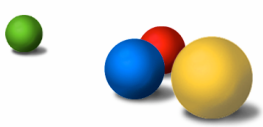
```
Trait.create(proto, {
  a: {
    value: function() {
      ... this.b ...
    },
    method: true
  },
  b: {
    value: 42
  }
})
```



```
Object.create(proto, {
  a: {
    value: function() {
      ... this.b ...
    },
  },
  b: {
    value: 42
  }
})
```

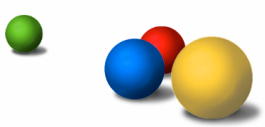
Trait instances are tamper-proof

+ throws on conflict/required
+ binds 'this' of methods
+ freezes object and methods



# Traits: summary

- traits.js: a minimal trait composition library
- Traits as property descriptor maps:
  - Object.create generates flexible objects
  - Trait.create generates tamper-proof objects
- Open issue: sharing bound methods between trait instances
- www.traitsjs.org



# Conclusion

- ES5-strict is a robust programming language
- Proxies: robust metaprogramming API for ES-harmony
- Traits: robust composition API for ES5
- Next: abstractions for robust event-driven programming

es-lab.googlecode.com