Virtual Values for Language Extension

Thomas H. Austin Tim Disney Cormac Flanagan
University of California Santa Cruz
ICFP'11

$$x + y$$

Extensibility in Python is clean

```
class Complex(object):

  def __init__(self, real, imag):
    self.r = real
    self.i = imag

  def __add__(self, other):
    return Complex(self.r + other.r,
                   self.i + other.i)
```

```
x = Complex(2, 1)
y = Complex(3, 1)

x + y
```

Extensibility in JavaScript is ugly

```
function Complex(real, imag) {
  this.r = real;
  this.i = imag;
}
Complex.prototype.plus(other) {
  return new Complex(this.r + other.r,
                     this.i + other.i);
}
```

```
var x = new Complex(2, 1);
var y = new Complex(3, 1);

x.plus(y);
```

Even worse than ugly!

```
function matrixMult(a, b) { ... }

matrixMult([[-3,-8,3],
            [-2,1,4]])

matrixMult([[new Complex(4,1), new Complex(2,1)],
            [new Complex(5,1), new Complex(8,1)]])
```

```
x + y
```

vs.

```
x.plus(y)
```

Virtual Values:

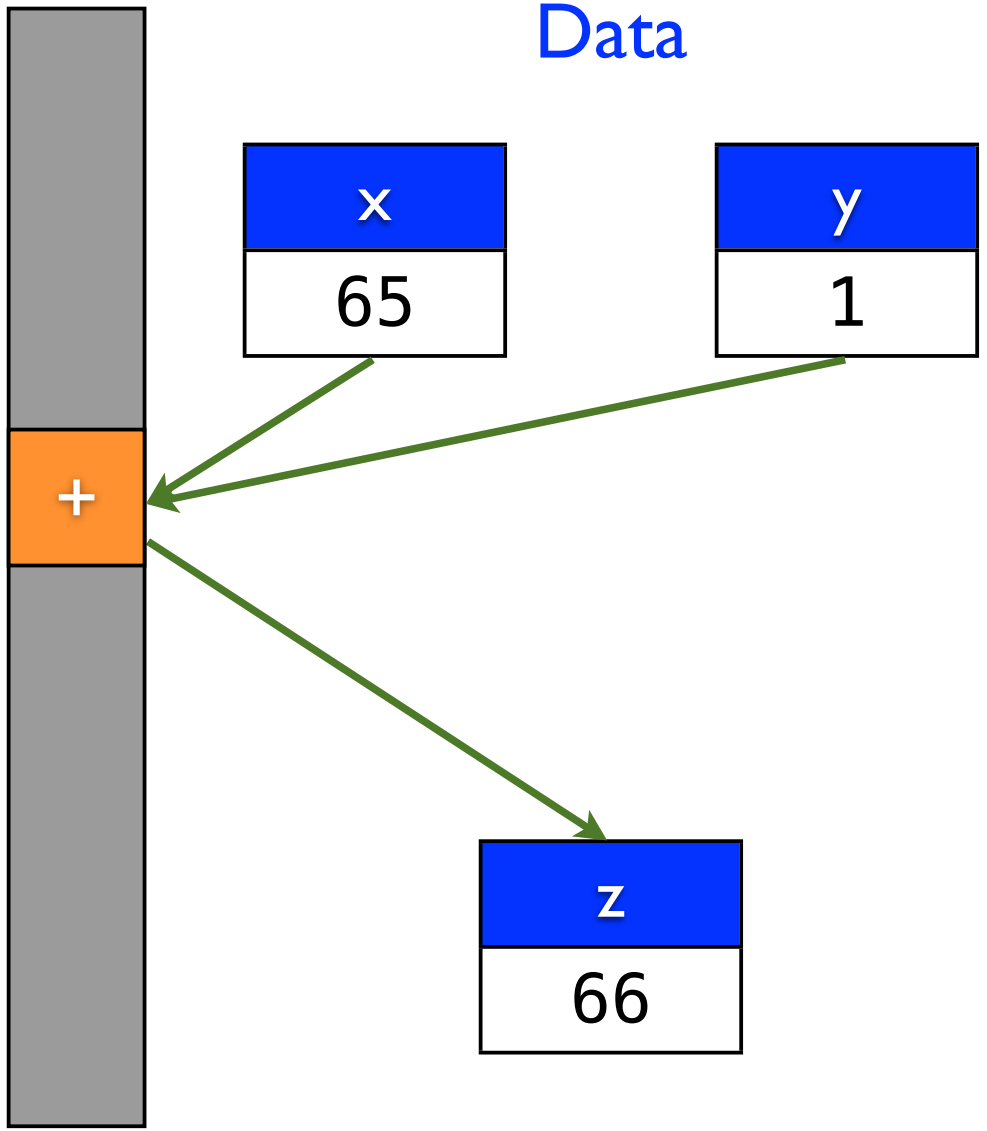
Virtualize the interface
between code and data

Standard Addition

Code

```
x = 65
y = 1

z = x + y
```

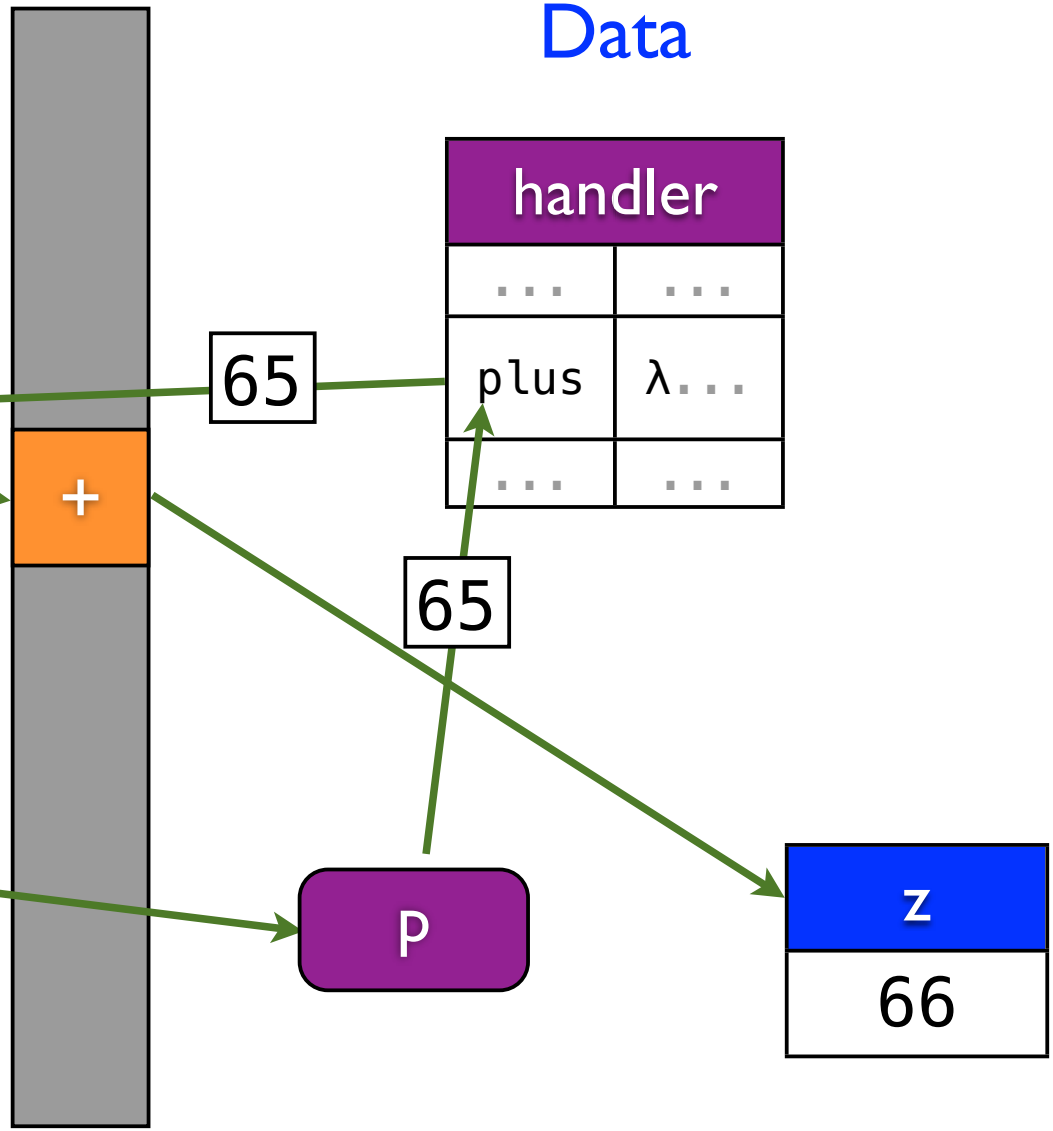
Data



Virtualized Addition

Code

```
handler = {
    ...
    plus: λr.
      1 + r
}

p = proxy(handler)

z = p + 65
```



λ_{proxy}

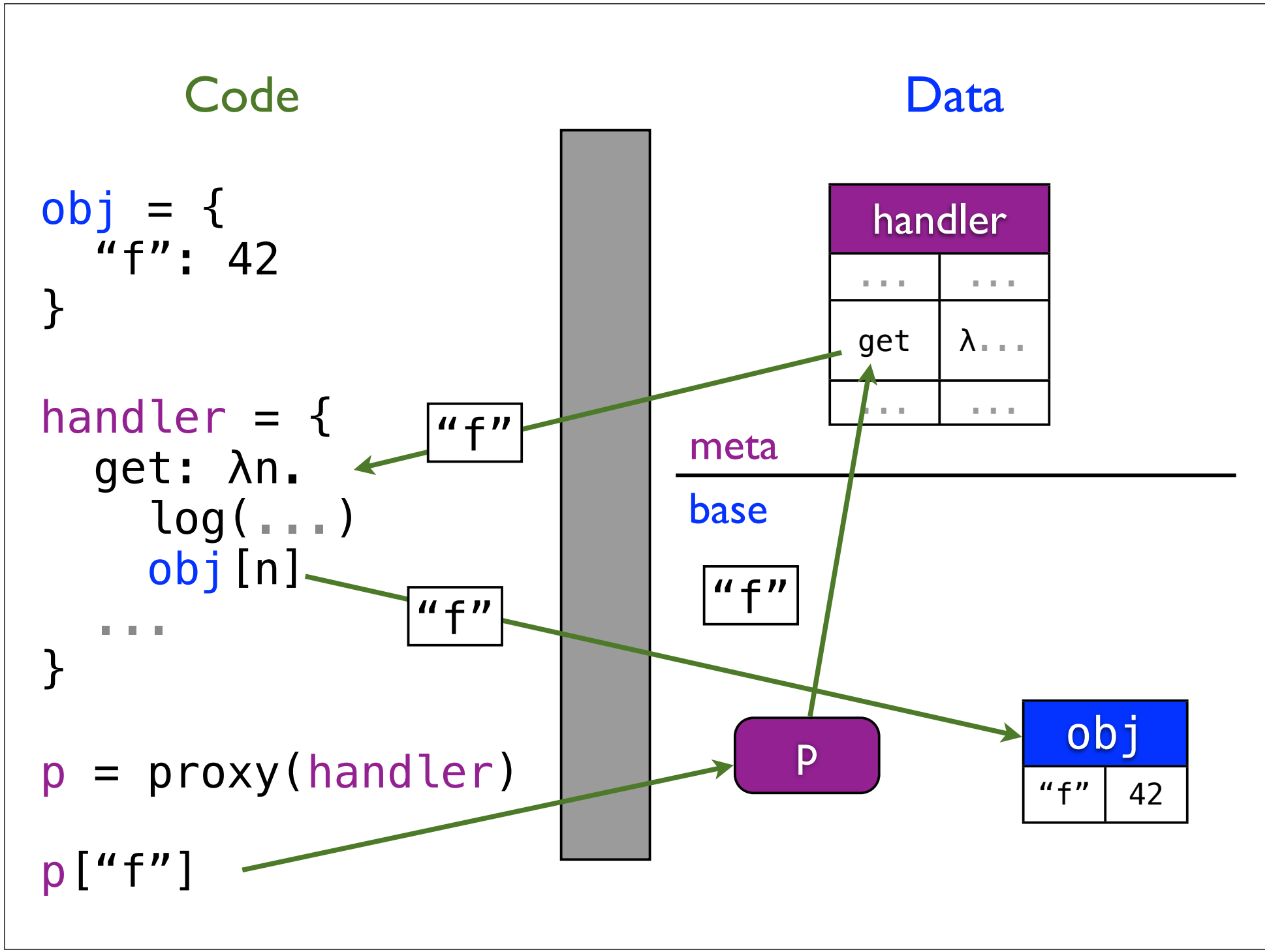
Idealized JavaScript-like language

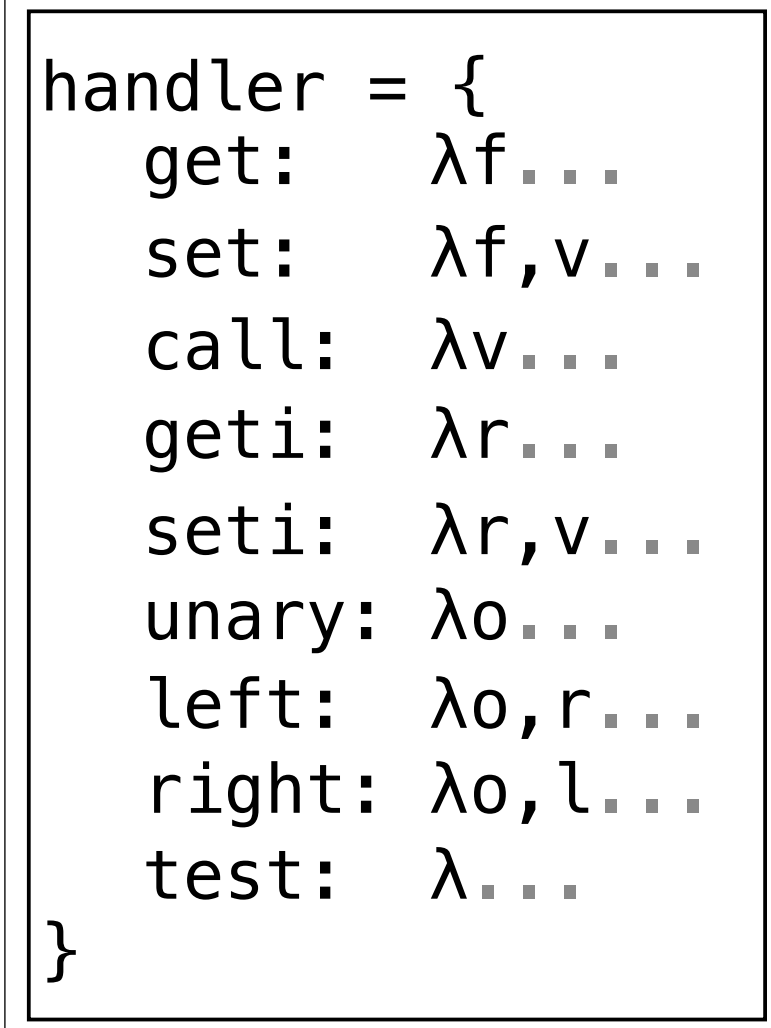
```
proxy(handler)
```

`λx. e`	`{ f : v }`	`24`
$e_1(e_2)$	`o[f]`	`true`
	`o[f] = v`	`!true`
		`24 + 42`
		`if b` e_1 e_2

```
handler = {
   get:   λf...
```

```
p = proxy(h)
p[f]        → h.get(f)
```

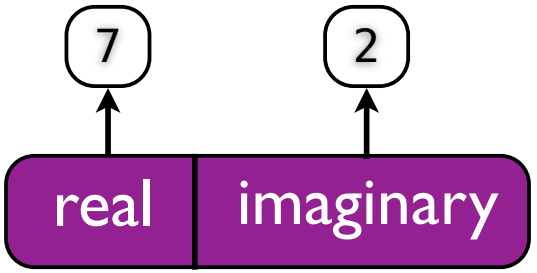

Code
Data
obj = {
"f": 42
}
handler = {
get: λn.
log(...)
obj[n]
...
}
p = proxy(handler)
p["f"]
"f"
"f"
handler
... ...
get λ...
... ...
meta
base
"f"
P
obj
"f" 42




```
p = proxy(h)
p[f]       → h.get(f)
p[f] = v   → h.set(f,v)
p(v)       → h.call(v)
r[p]       → h.geti(r)
r[p] = v   → h.seti(r,v)
!p         → h.unary("!")
p + x      → h.left("+",x)
x + p      → h.right("+",x)
if p e e   → if h.test() e e
```

```
1  private secret = {}
```

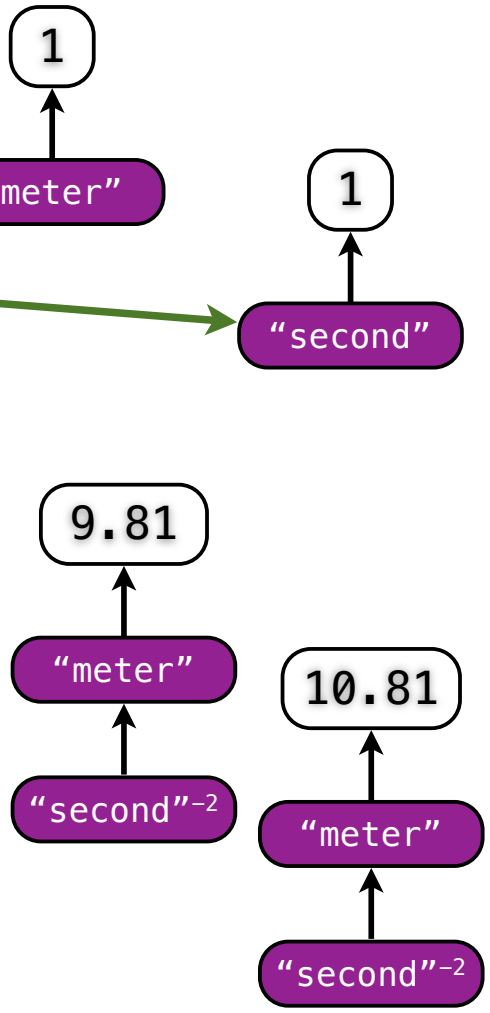
```
x = 4.0 + (1.0 * i)

y = 3.0 + (1.0 * i)

x + y
```




```
23                      else
24                          complexBinOps[o] r i y 0
25          right: λo,y. complexBinOps[o] y 0 r i
26          test : λ.    true // all Complex are non-false
27      }
28
29  isComplex :: Any → Bool = λx. if (unProxy secret x) true false
30
31  i :: Complex = makeComplex 0 1
32
33  Complex = Flat_c isComplex
```

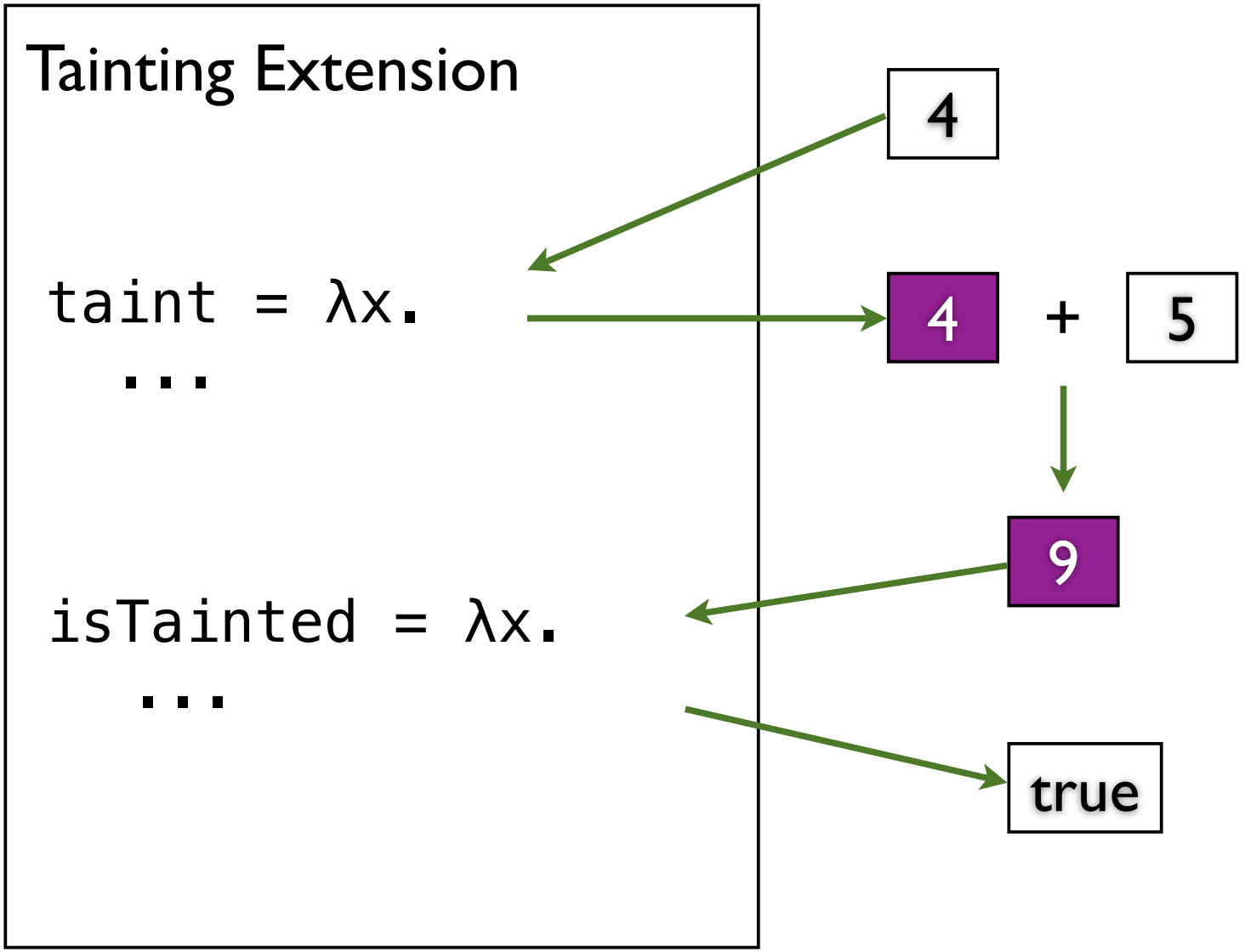
```
1  private secret = {}
2
3  private makeQuantity :: String → Int → Quantity → Quantity = λu,i,n.
4      let h = unProxy secret n
5      if (i = 0)                    // drop zero-ary unit
```

```
meter = makeUnit("meter")

second = makeUnit("second")

g = 9.81 * meter / second / second

print(g)  // "9.81 meters seconds^-2"

g + 1 // Error: Units not compatible!

g + 1 * meter / second / second
```




```
44 private dropUnit :: String → Int → Quantity → Quantity = λu,i,n.
45     let h = unProxy secret n
46     assert h != false && h.unit = u && h.index = i
47     h.value
48
49 makeUnit :: String → Quantity = λu. makeQuantity u 1 1
50 Quantity = Flat_c (λx. if (isNum x || unProxy secret x) true false)
```

```
isTainted(taint(4) + 5) == true
```



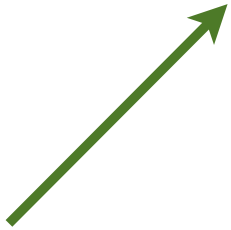
Security

Extensibility: wants to extend behavior of library extensions

Security: wants to restrict behavior of adversaries

Security

`isProxy(x)`



Always tells the truth

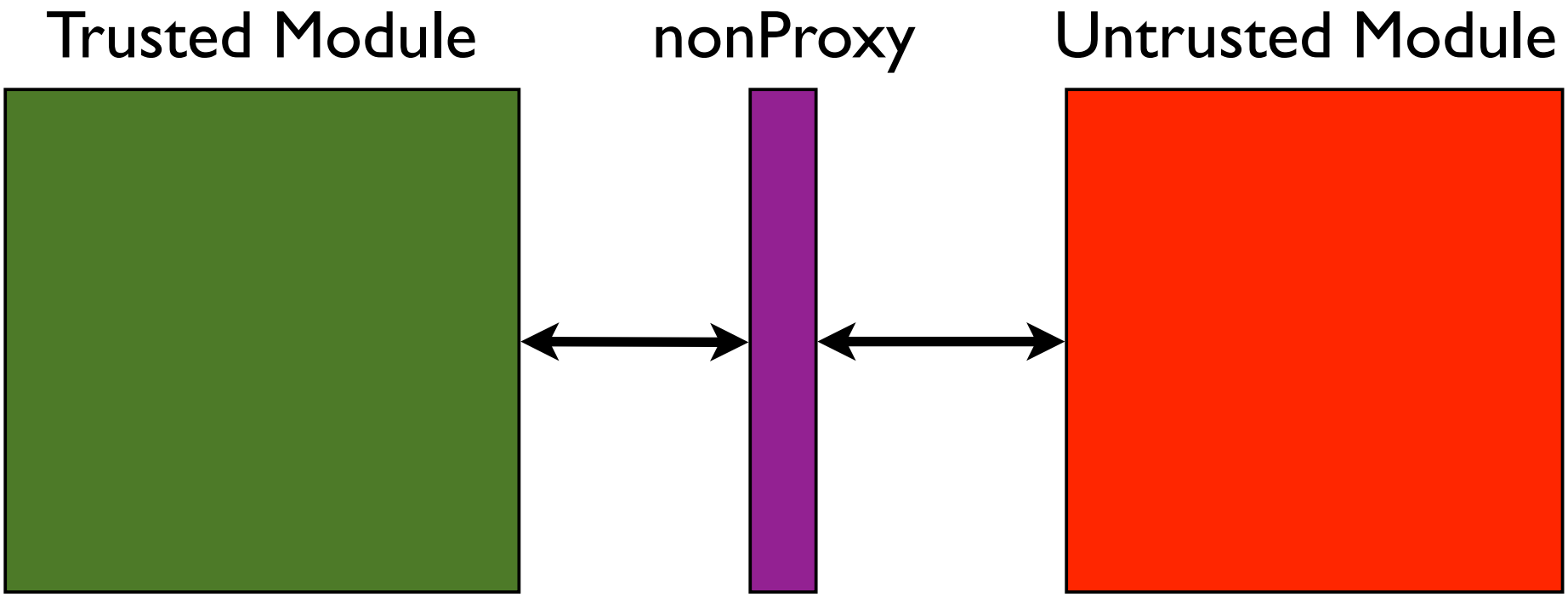
Stop proxies...

```
critical = λx.
  if isProxy(x)
   then err()
   else ...
```

...not quite

```
critical = λx.
  if isProxy(x)
   then err()
   else
    y = x()
    ...
```

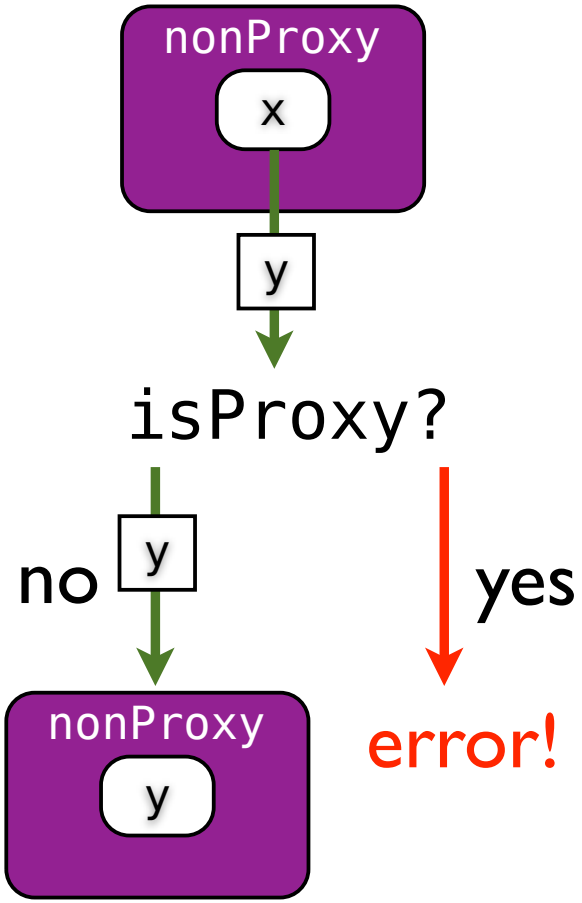
The nonProxy proxy!



The nonProxy proxy!

```
1  private secret = {}
2
17        unary: λo.    swap (unaryOps[o] x)
18        left : λo,r.  swap (binOps[o] x (swap r))
19        right: λo,l.  swap (binOps[o] (swap l) x)
20        test : λ.     if (x) true false
21      }
```

```
critical = λx.
   x = nonProxy(x)
   y = x()
      . . .
```



```
handler = {
    get:    ...
    set:    ...
    call:   ...

    geti:   ...
    seti:   ...
    unary:  ...
    left:   ...
    right:  ...
    test:   ...
}
```

JavaScript Proxies

contracts membranes

nonProxy

Virtual Values

complex units

taint tracking lazy evaluation

T.V. Cutsem and M. S. Miller. *Proxies: Design principles for robust object-oriented intercession APIs*