

On the Evaluation of Space-Time Functions

Jacob Beal, Kyle Usbeck

Spatial Computing Workshop
@ SASO 2011

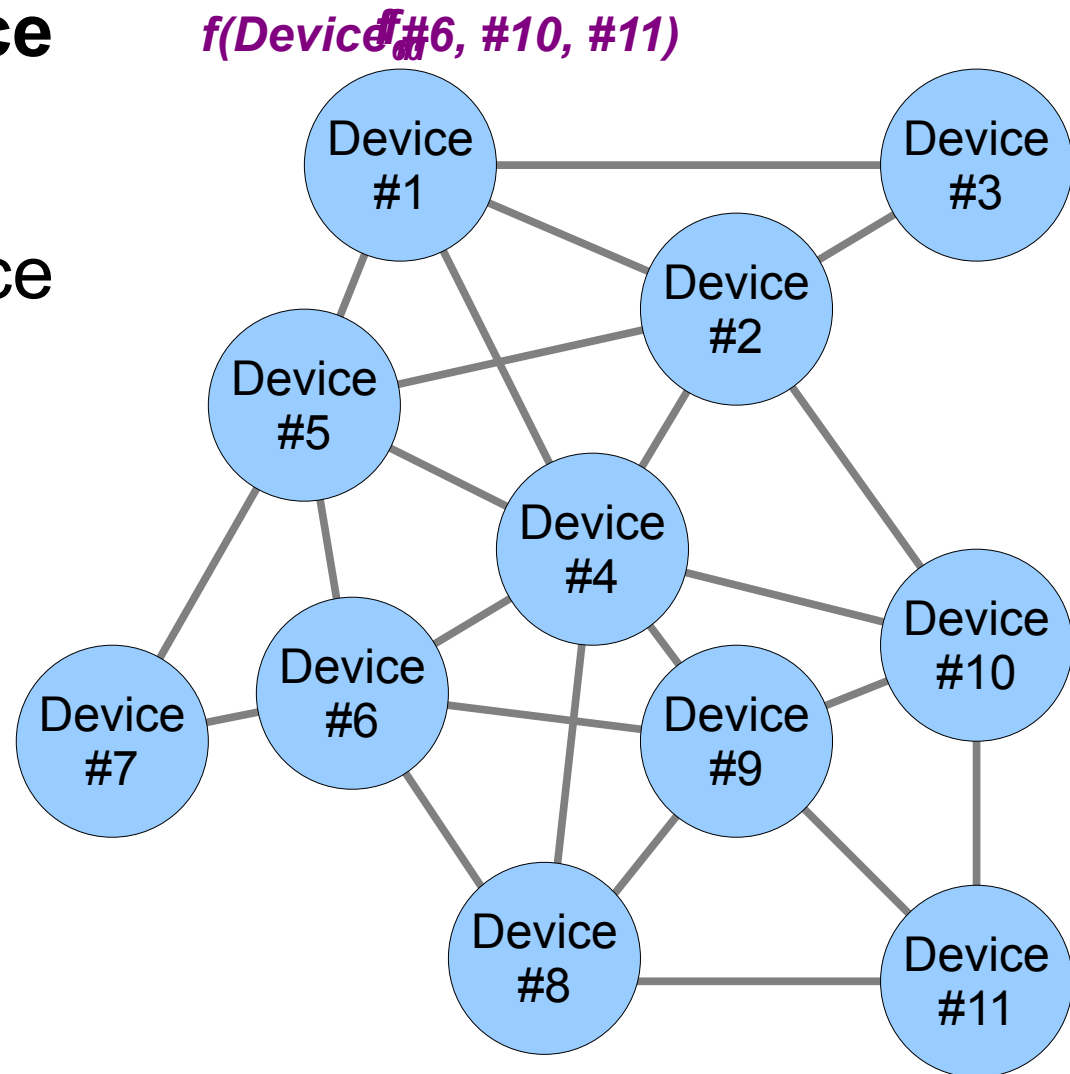
Raytheon
BBN Technologies

Distributed function calls are hard!

- Action at a distance
- Recursion
- Function equivalence

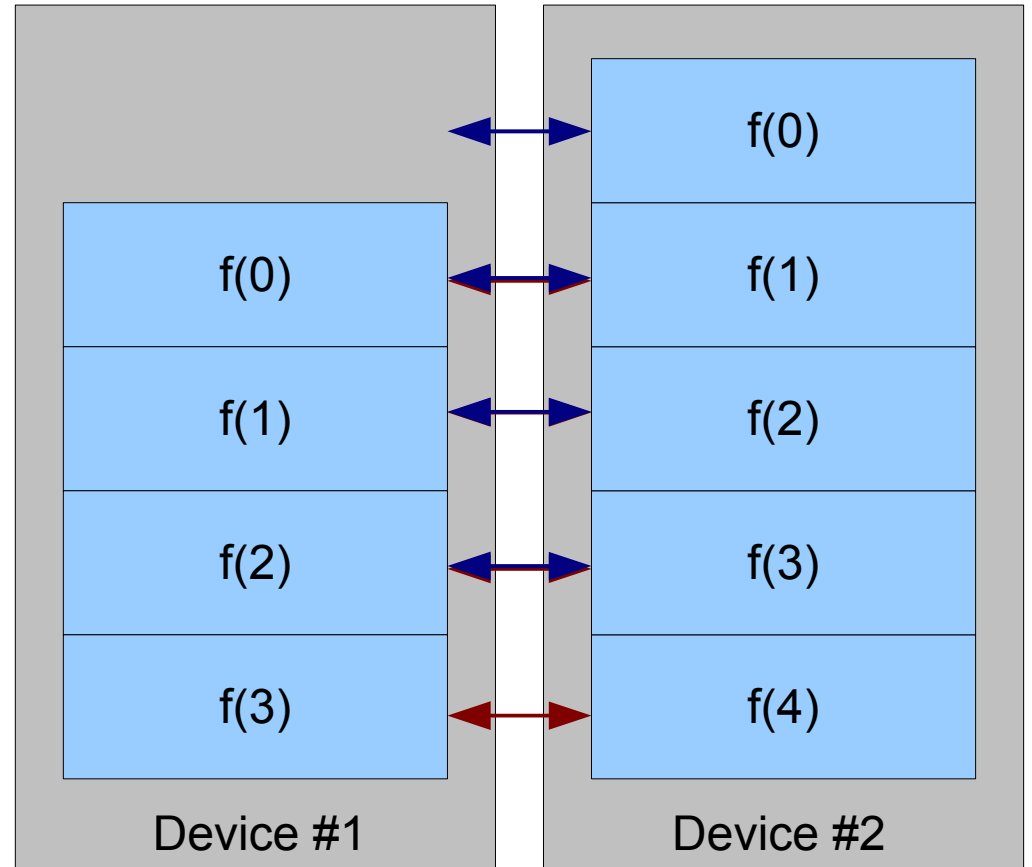
Distributed function calls are hard!

- **Action at a distance**
- Recursion
- Function equivalence



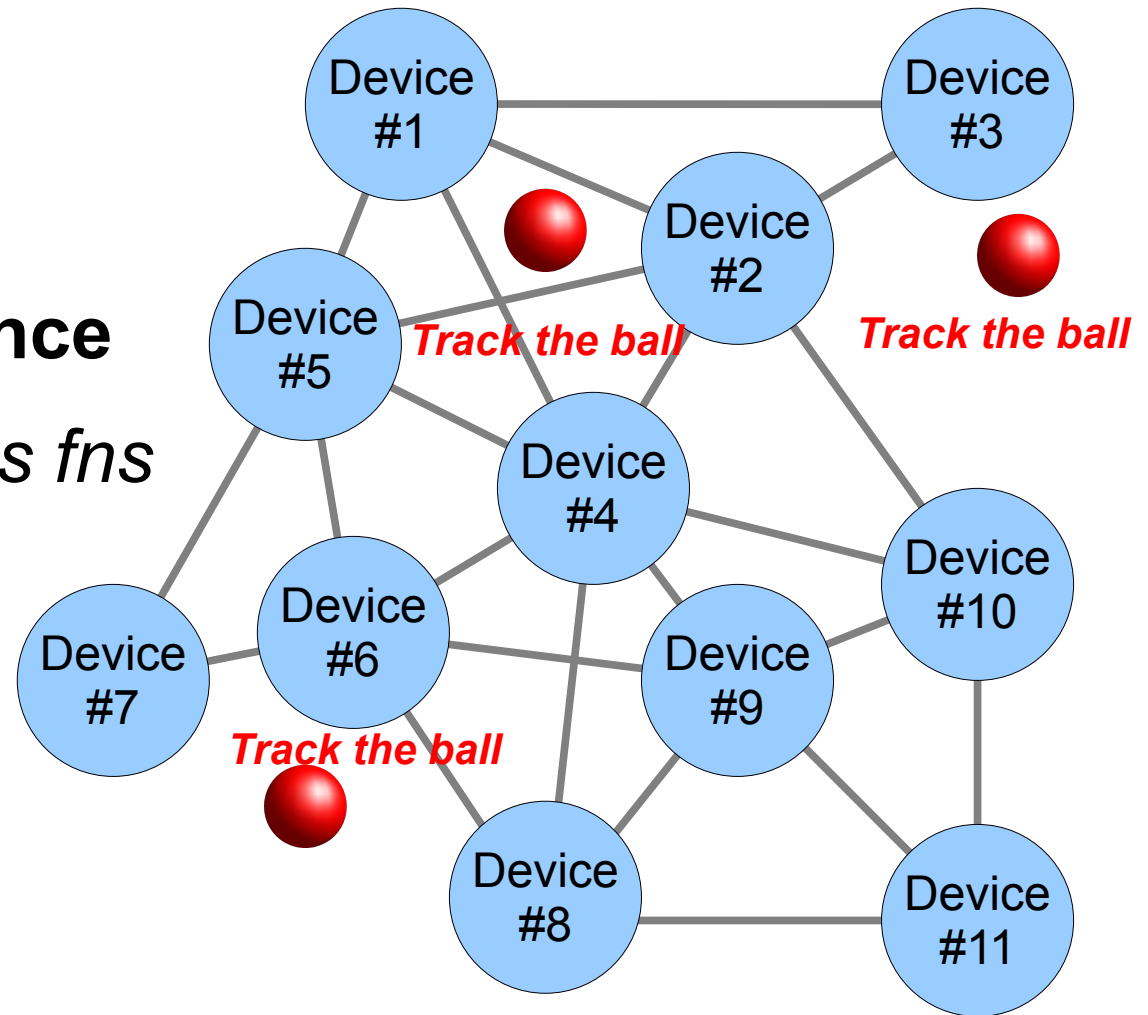
Distributed function calls are hard!

- Action at a distance
- **Recursion**
- Function equivalence



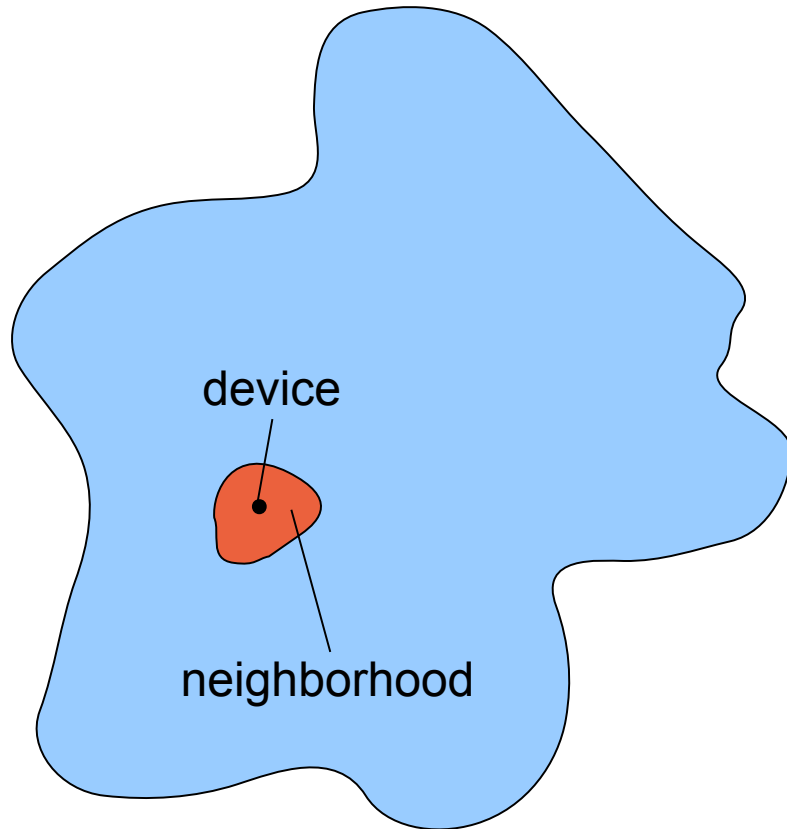
Distributed function calls are hard!

- Action at a distance
 - Recursion
 - **Function equivalence**
 - Worse w. *1st class fns*
- [c.f. Beal, 2009]

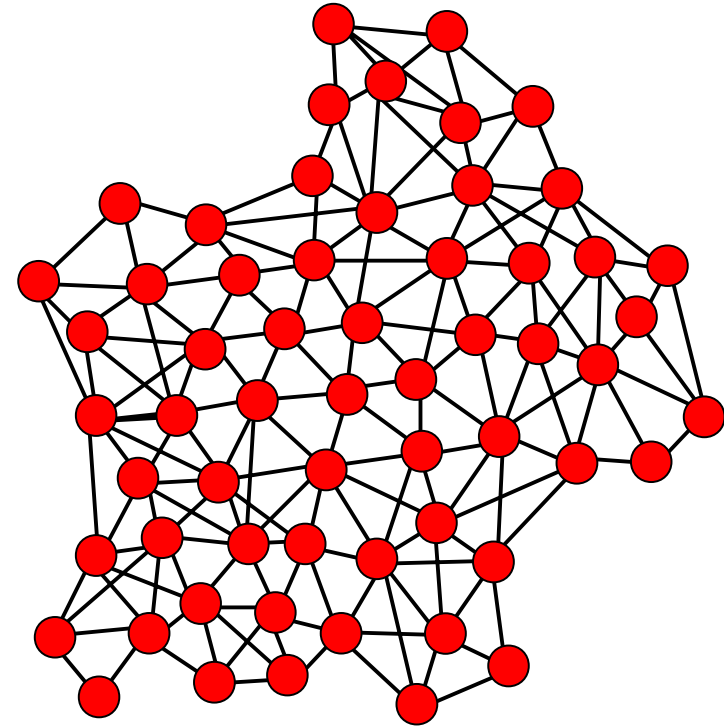


Most “distributed” calls avoid the last two...

Approach: Continuous Model



- Continuous space & time
- Infinite number of devices
- See neighbors' past state



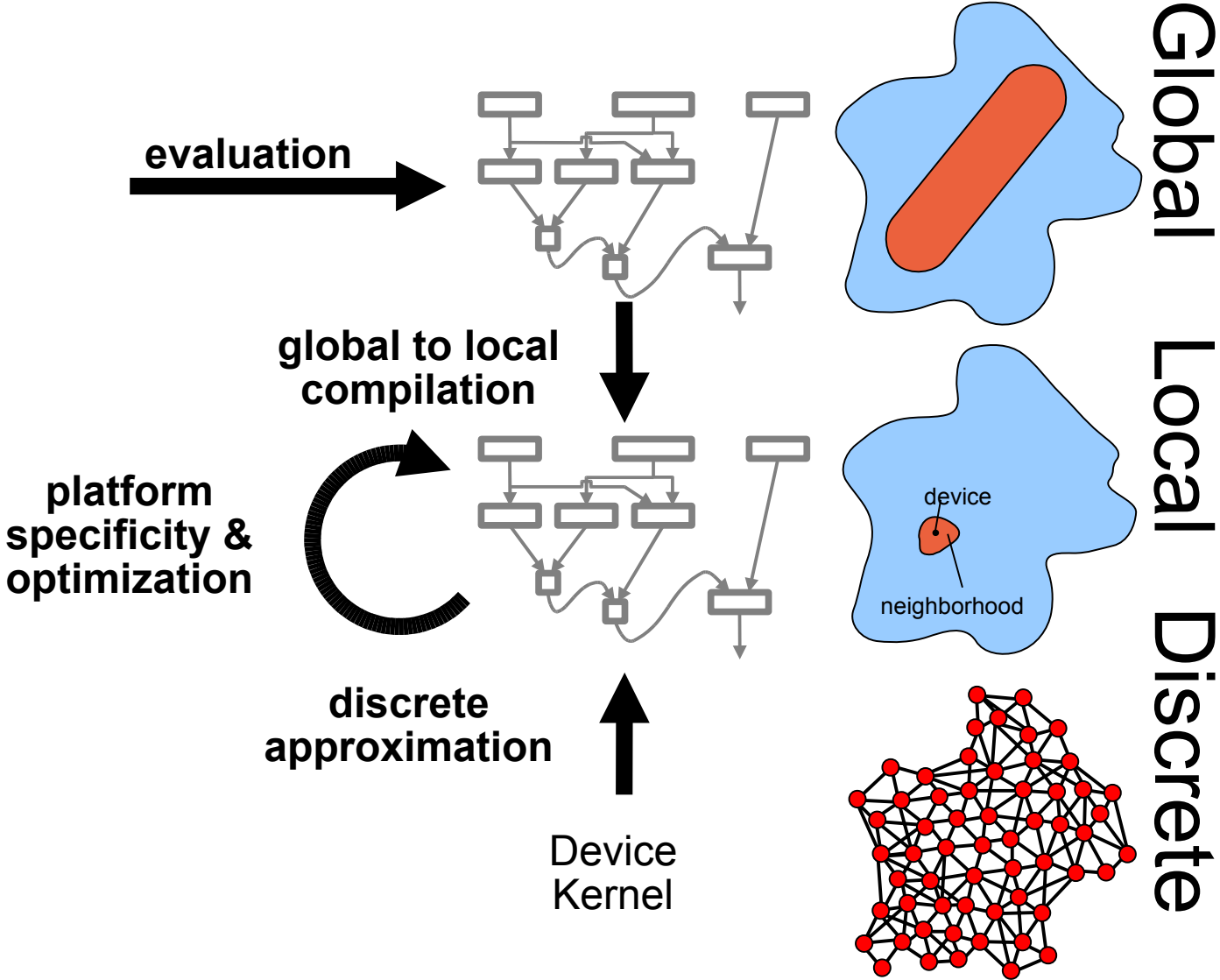
Approximate with:

- Discrete network of devices
- Signals transmit state

Advantages: simple, scalable, robust, adaptive...

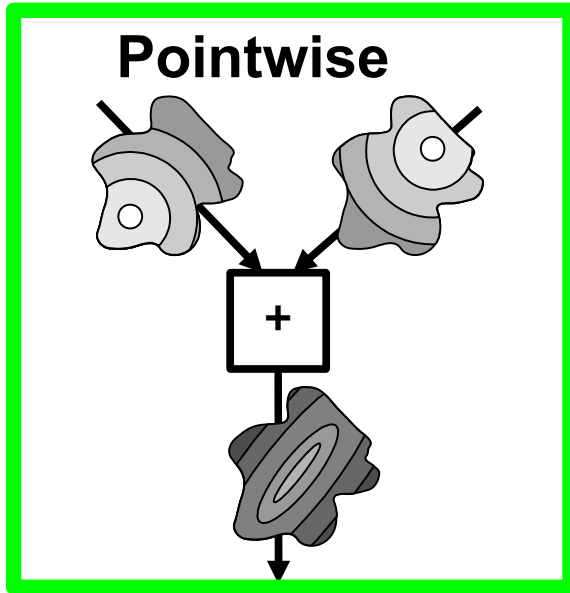
Proto

```
(def gradient (src) ...)  
(def distance (src dst) ...)  
(def dilate (src n)  
  (<= (gradient src) n))  
(def channel (src dst width)  
  (let* ((d (distance src dst))  
         (trail (<= (+ (gradient src)  
                       (gradient dst))  
                    d)))  
    (dilate trail width)))
```

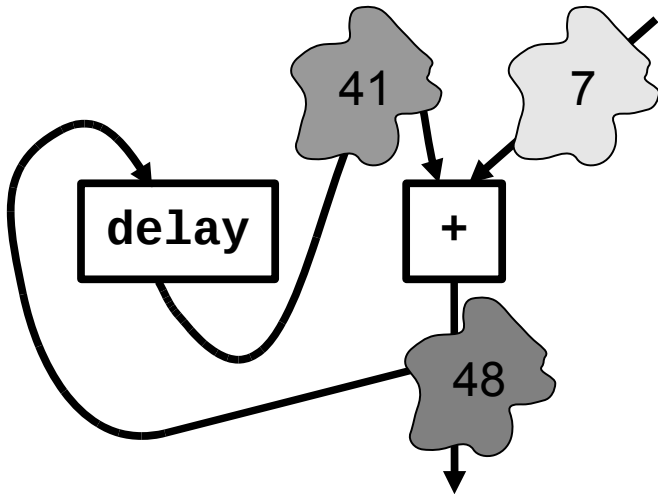


[Beal & Bachrach, '06]

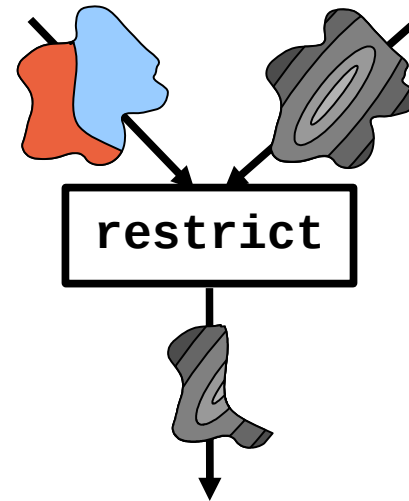
Proto's Families of Primitives



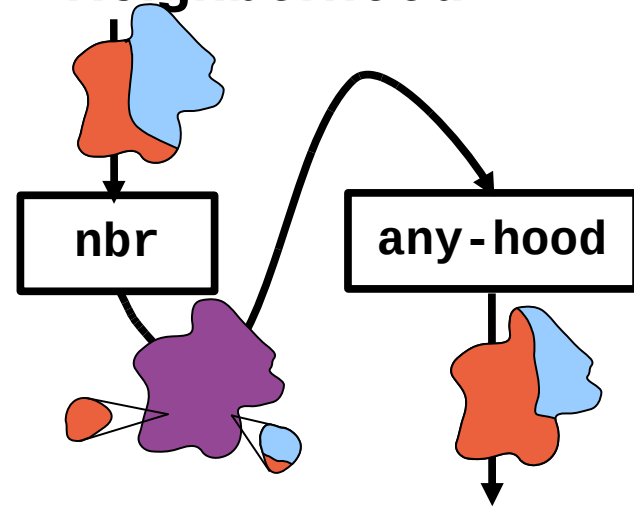
Feedback



Restriction

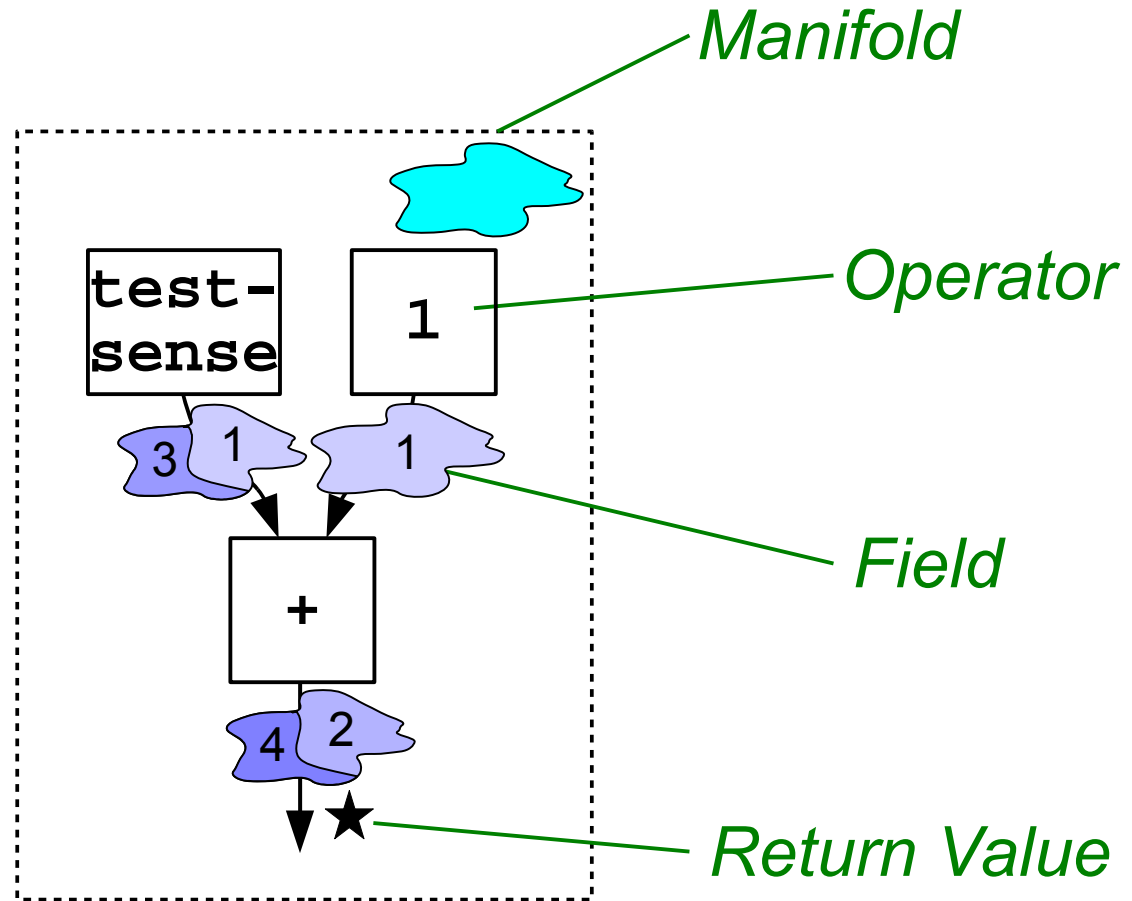


Neighborhood



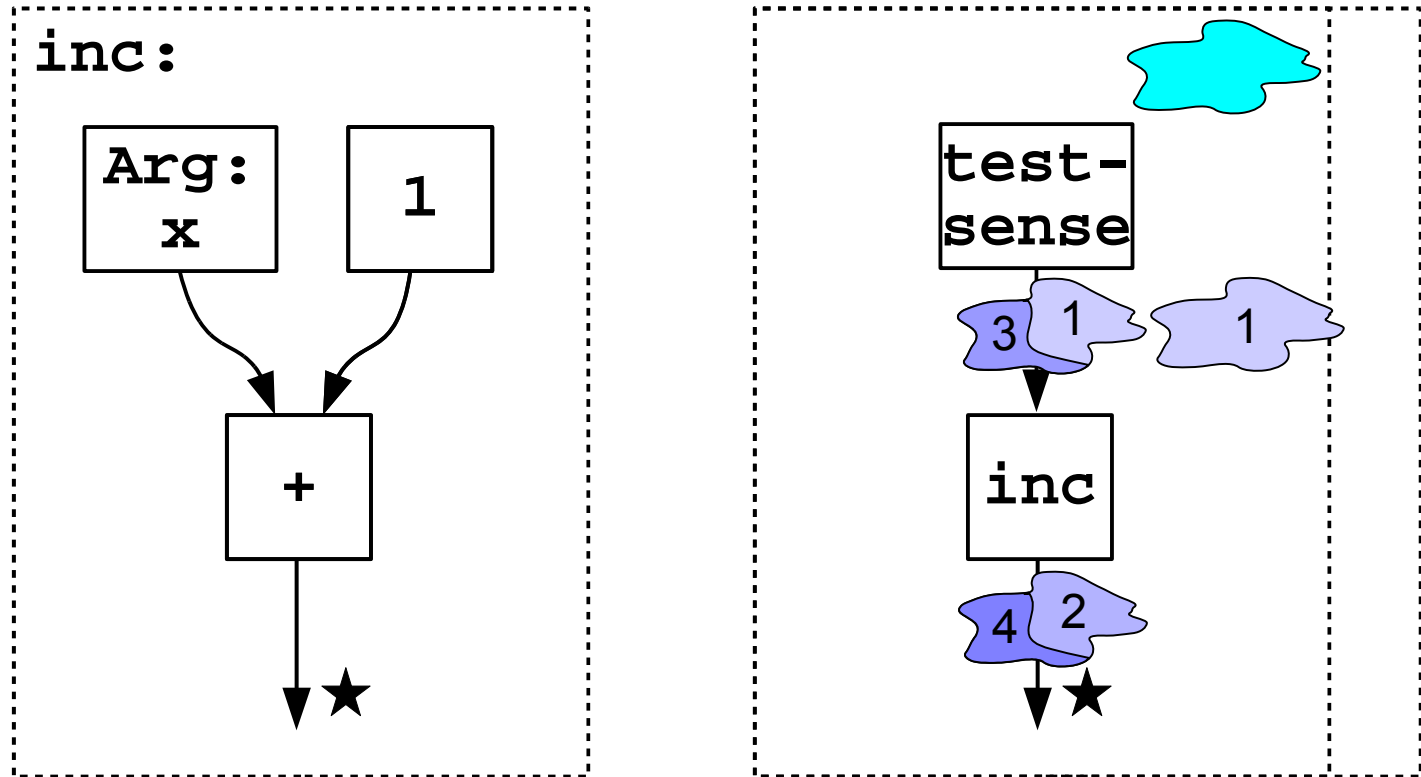
Continuous Space-Time Programs

*Well-defined iff:
each operator's
inputs and outputs
have same domain.*



`(+ 1 (test-sense))`

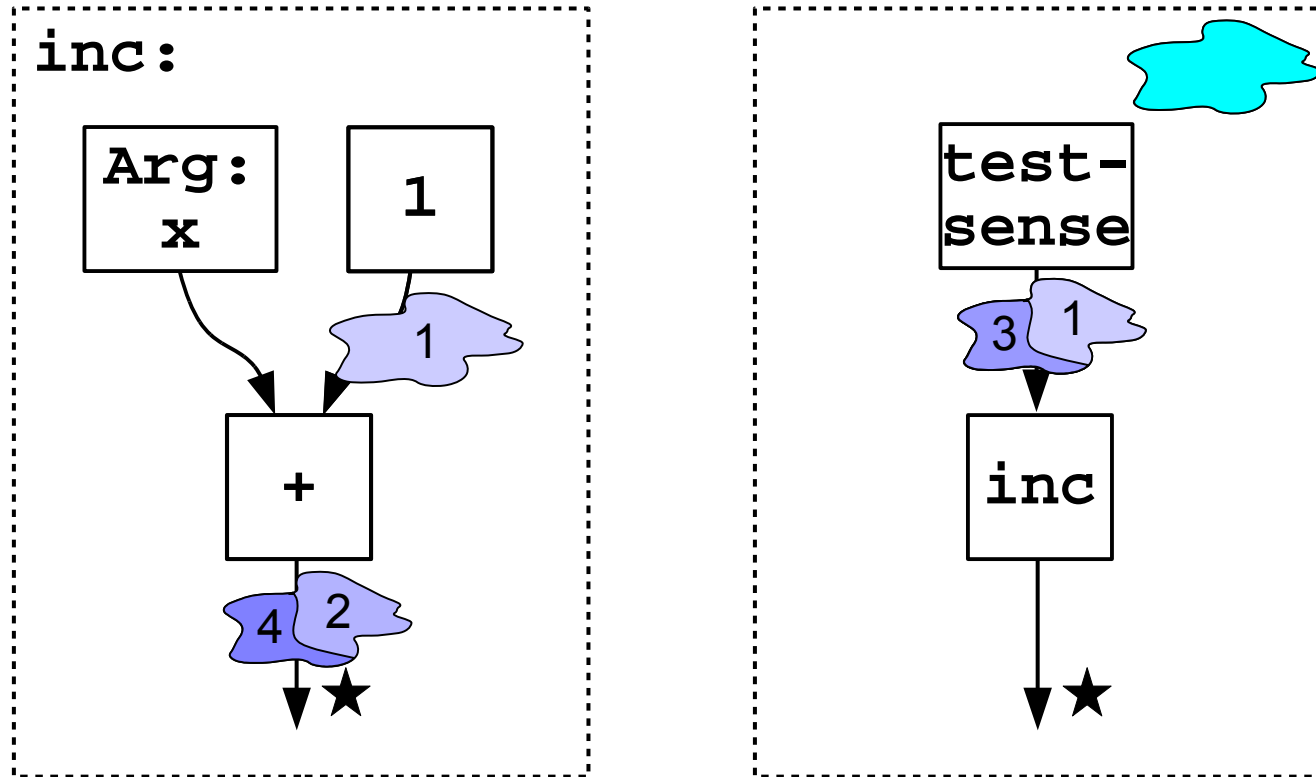
Functions: Substitution Model



```
(def inc (x) (+ x 1))  
(inc (test-sense))
```

Problems: code bloat, can't compile recursion or 1st class fns

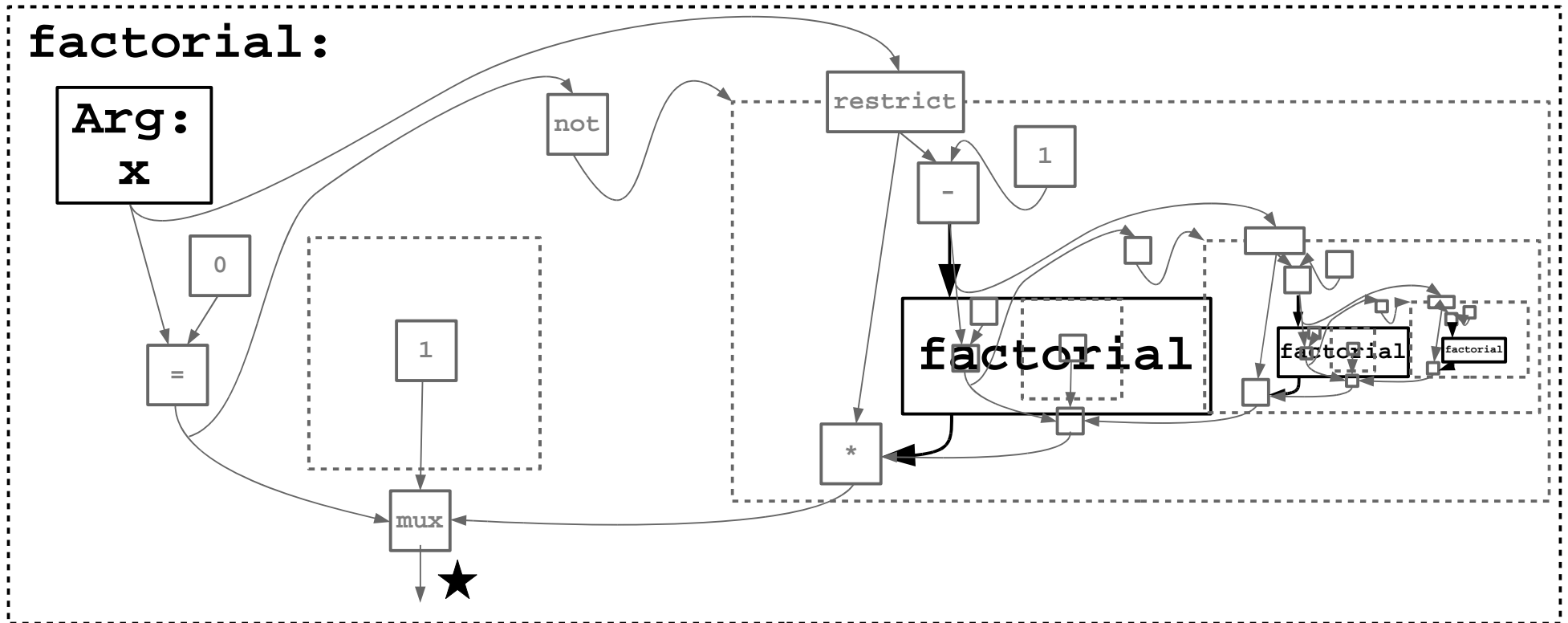
Functions: Call In-Place Model



```
(def inc (x) (+ x 1))  
(inc (test-sense))
```

Better: allows "normal" compiled function calls

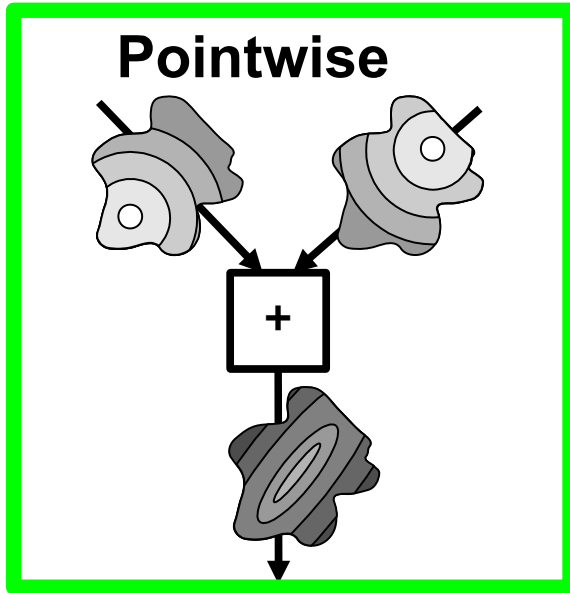
Recursion: Substitution vs. Call-in-Place



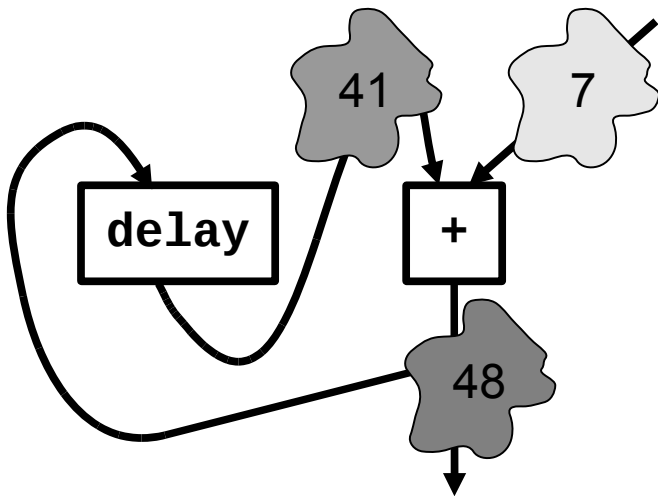
```
(def factorial (x)
  (if (= x 0)
      1
      (* x (factorial (- x 1)))))
```

To make call-in-place work, we'll need to be able to branch...

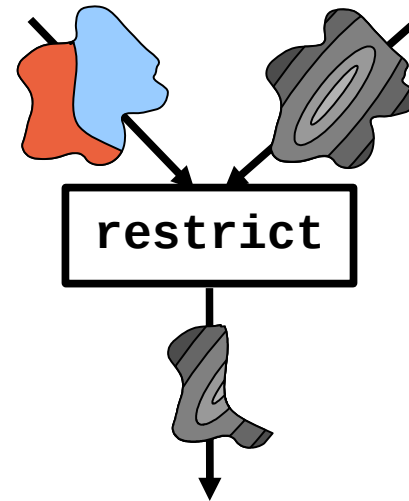
Proto's Families of Primitives



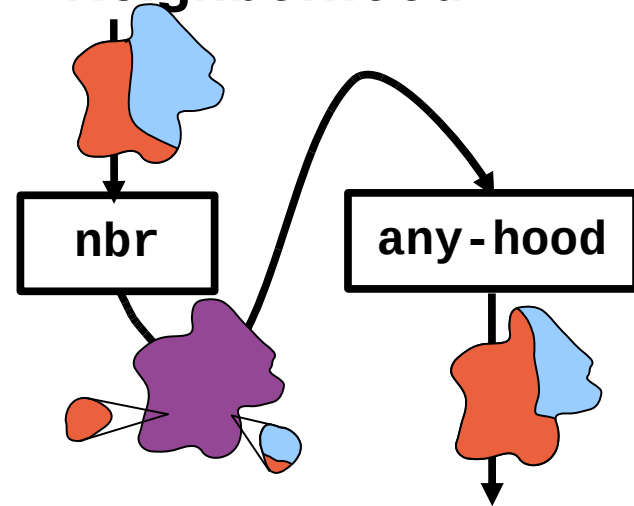
Feedback



Restriction



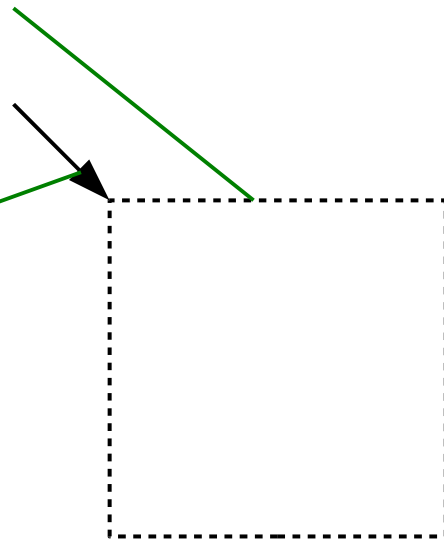
Neighborhood



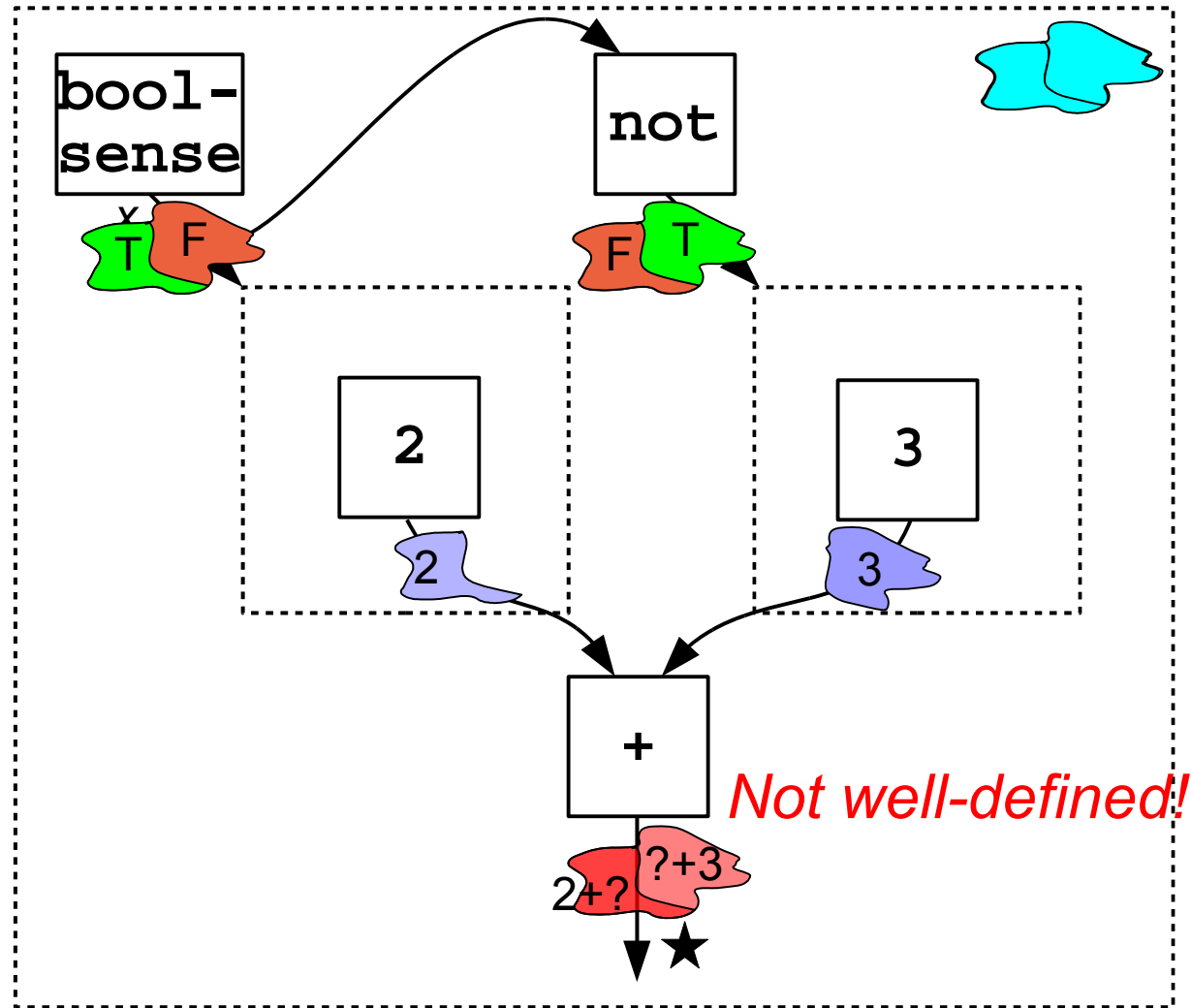
Different Spaces Interacting

Sub-manifold

Selector

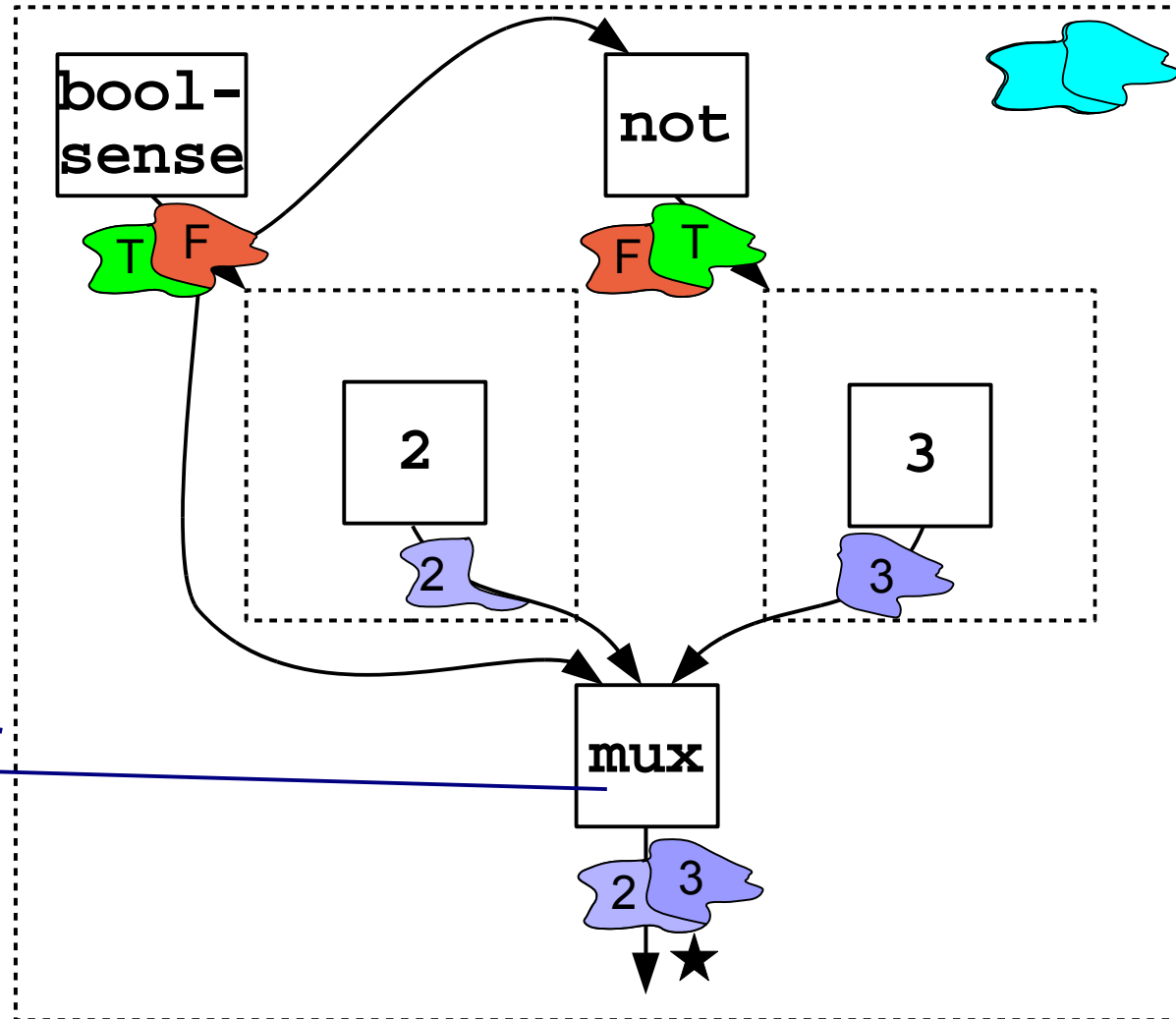


Different Spaces Interacting



```
(let ((x (bool-sense)))  
  (+ (restrict x 2)  
     (restrict (not x) 3)))
```

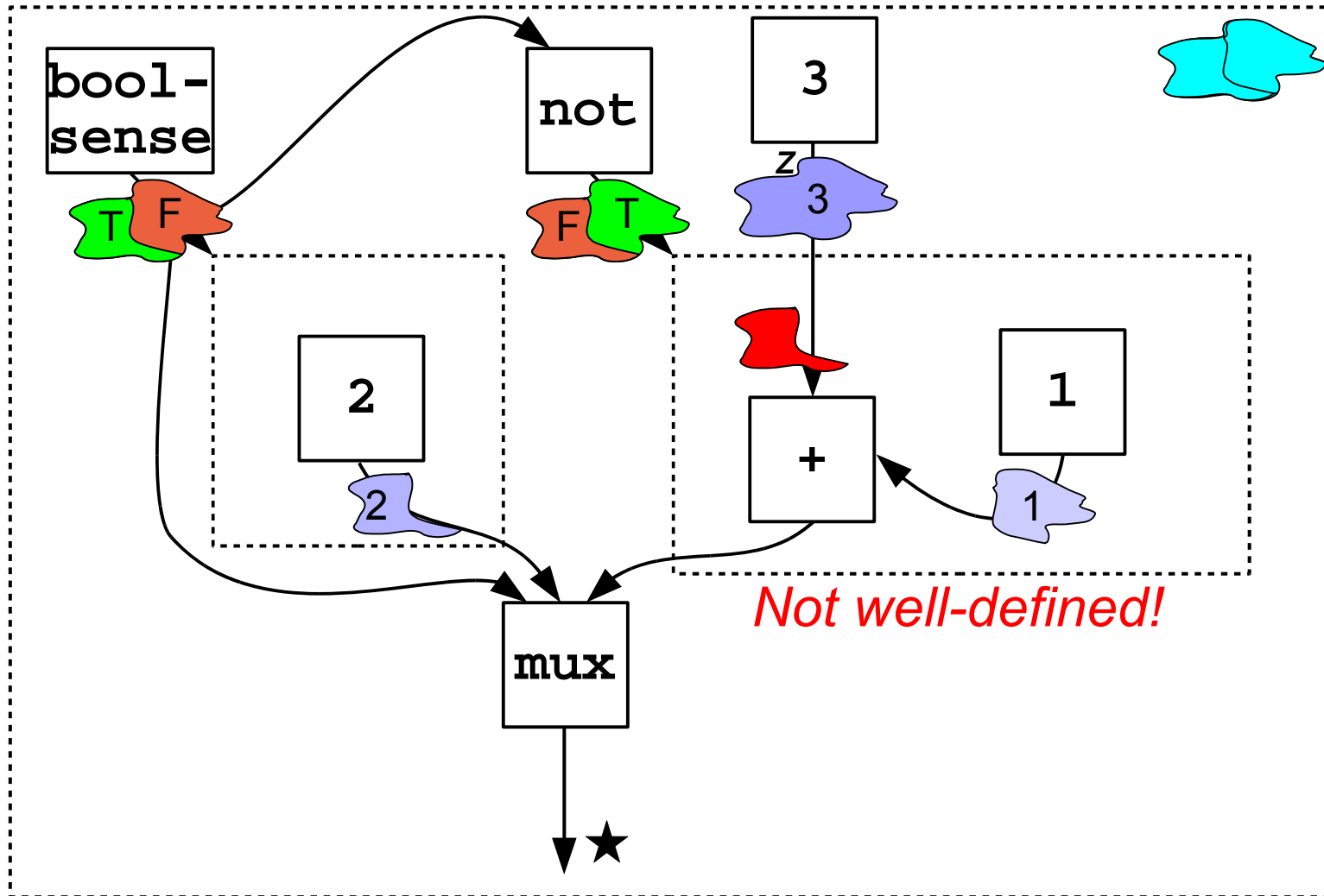
Syntactic safety: if & mux



*Well-defined iff
selected inputs
cover output.*

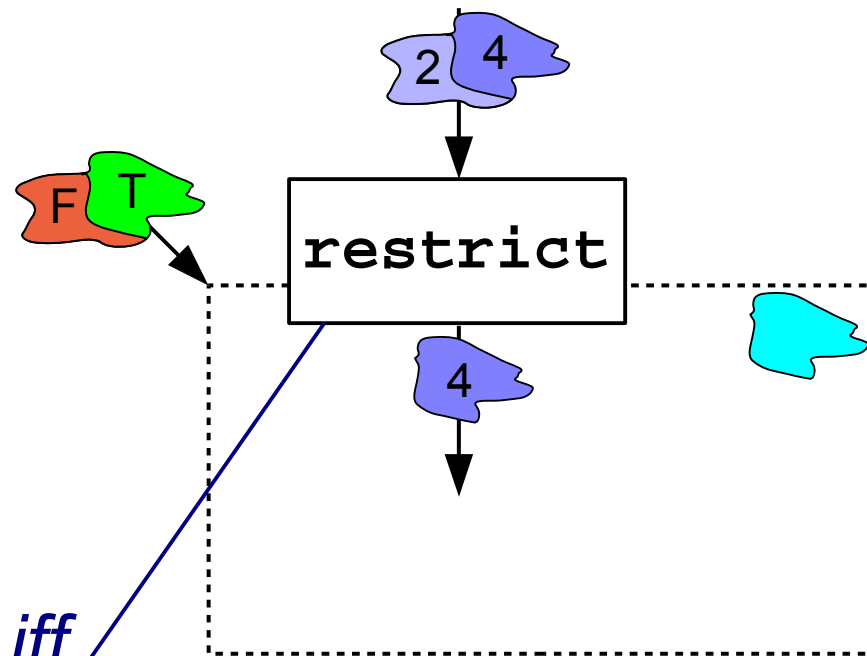
`(if (bool-sense) 2 3)`

Changing Field Domains



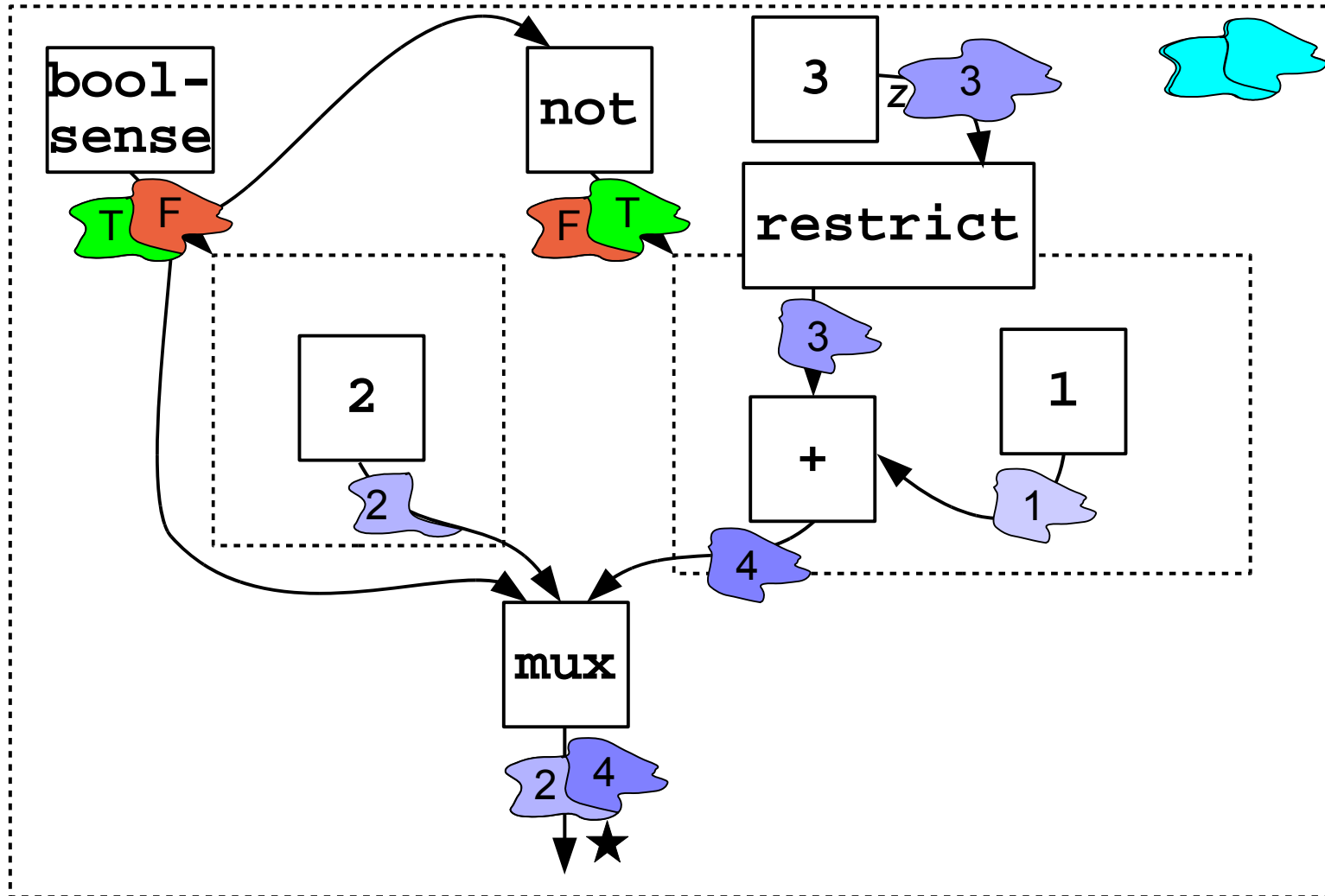
```
(let ((z 3))  
  (if (bool-sense) 2 (+ z 1)))
```

Changing Field Domains



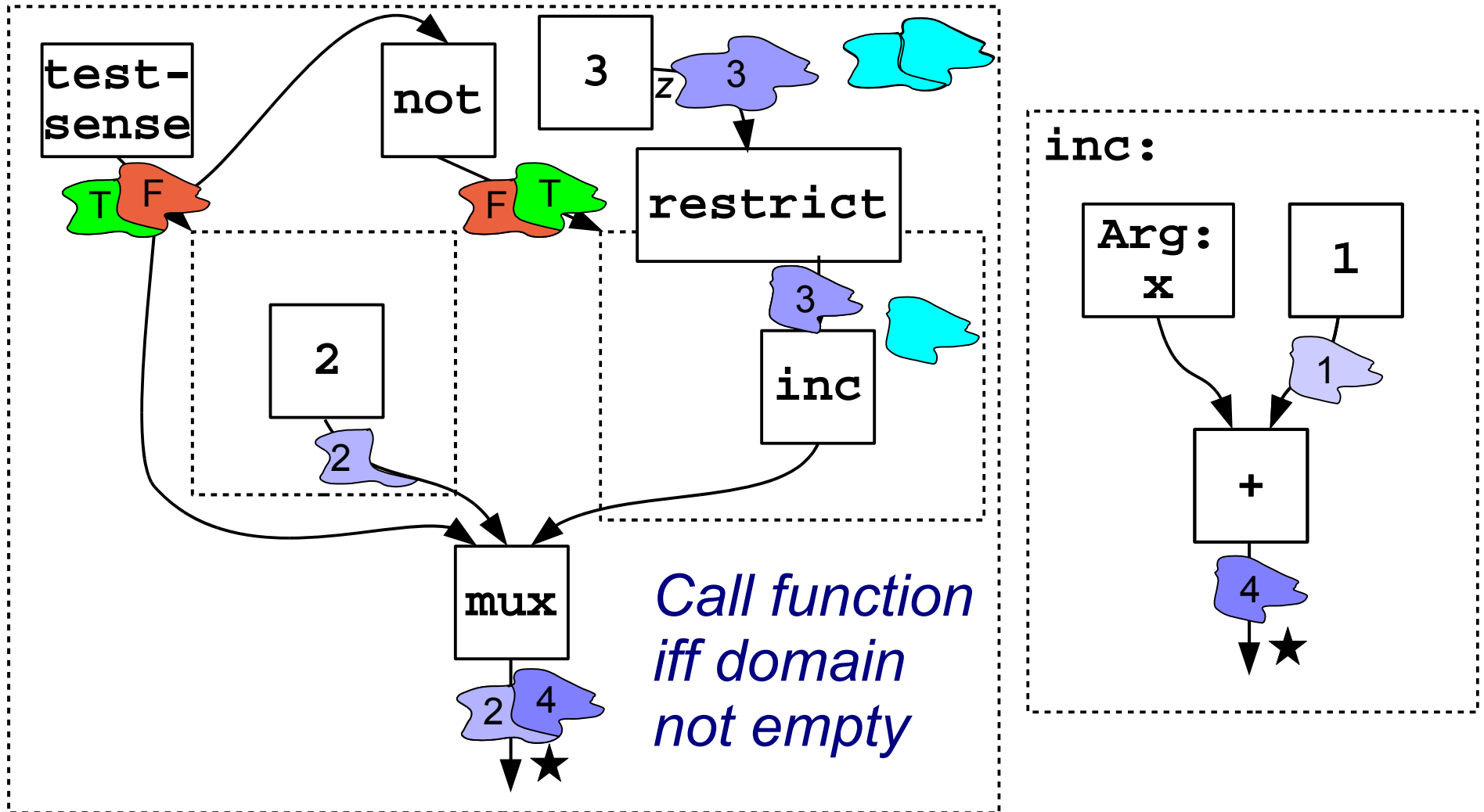
*Well-defined iff
output domain becomes
subspace of input domain*

Changing Field Domains



```
(let ((z 3))  
  (if (bool-sense) 2 (+ z 1)))
```

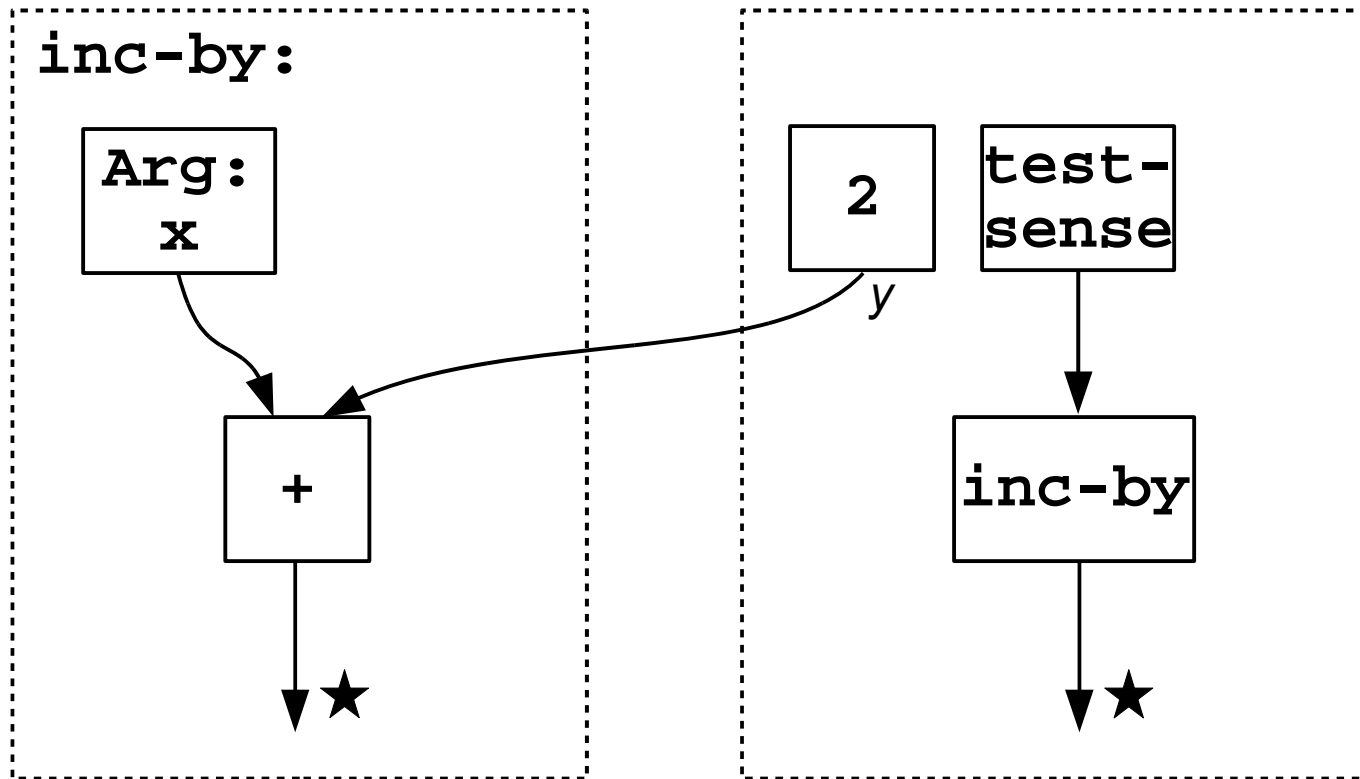
Branches and Function Calls



```
(let ((z 3)) (if (bool-sense) 2 (inc z)))
```

Manifold restriction → well-defined distributed function calls!

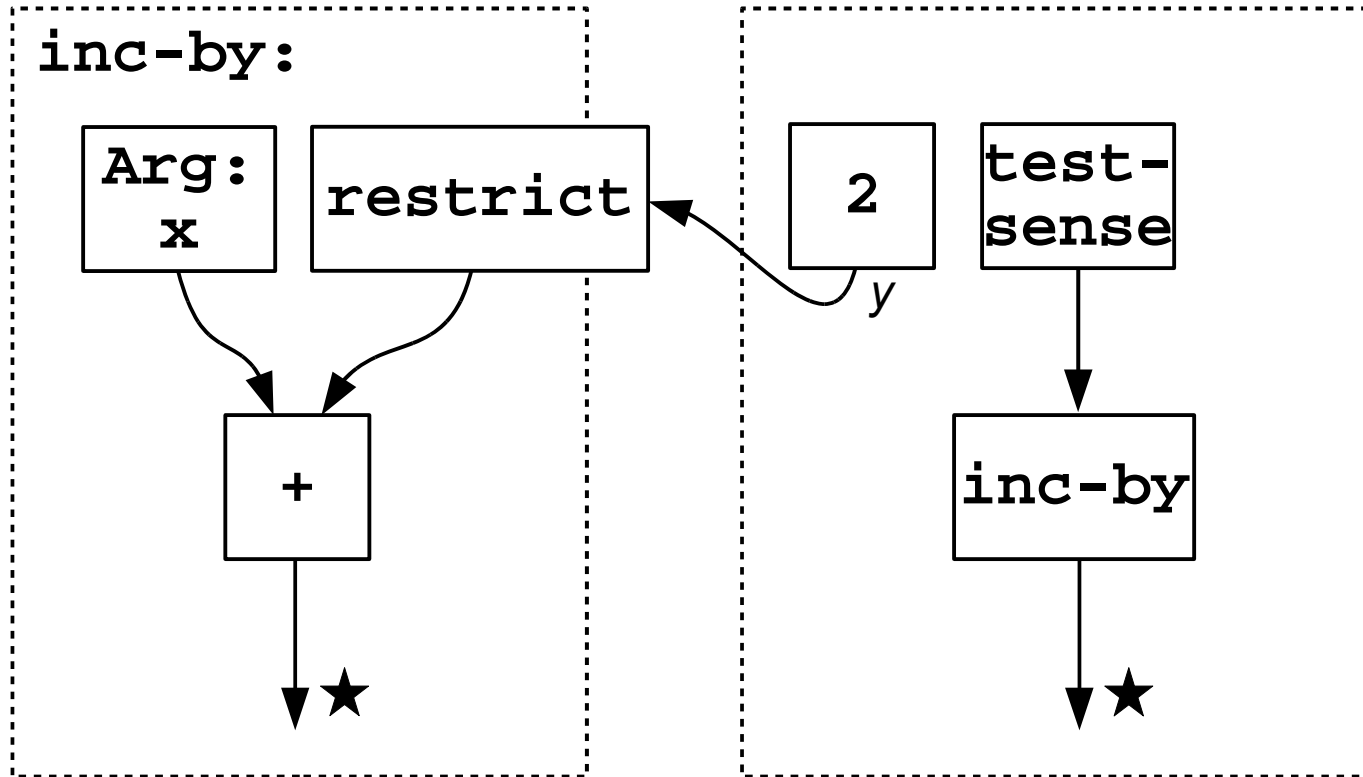
Branches → Function Closure



```
(let ((y 2))  
  (def inc-by (x) (+ x y))  
  (inc-by (test-sense)))
```

*Problem: what if the function call was in an **if**?*

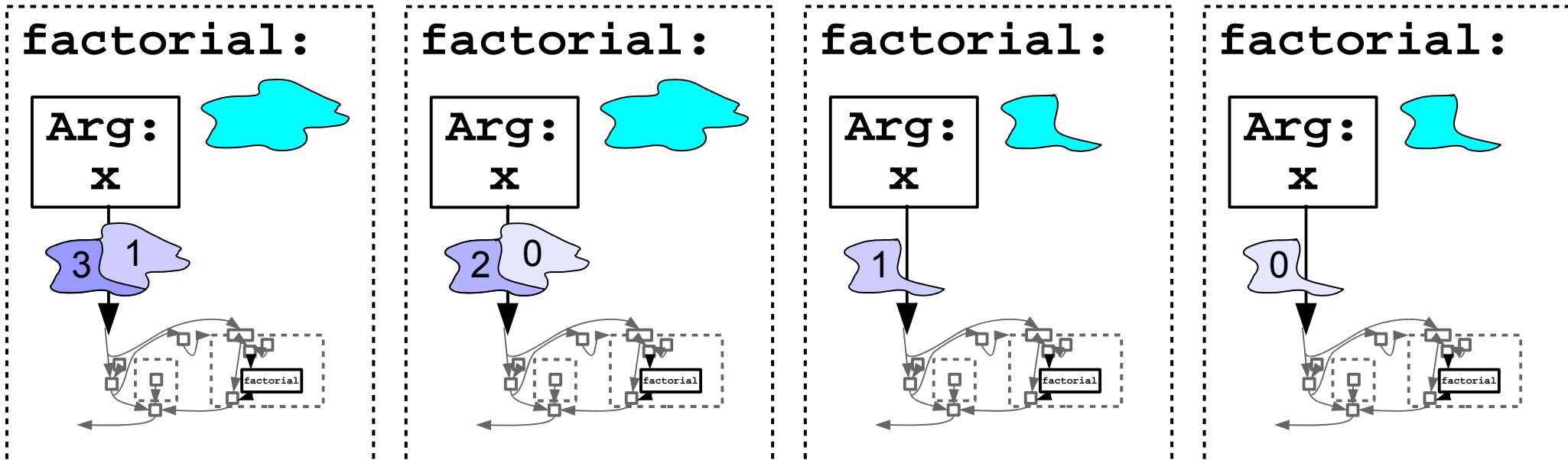
Branches → Function Closure



```
(let ((y 2))
  (def inc-by (x) (+ x y))
  (inc-by (test-sense)))
```

*Solution: external references pass through **restrict***

Distributed Recursion

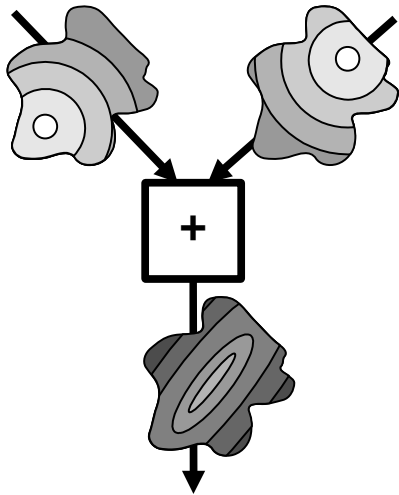


```
(def factorial (x)
  (if (= x 0)
      1
      (* x (factorial (- x 1)))))
```

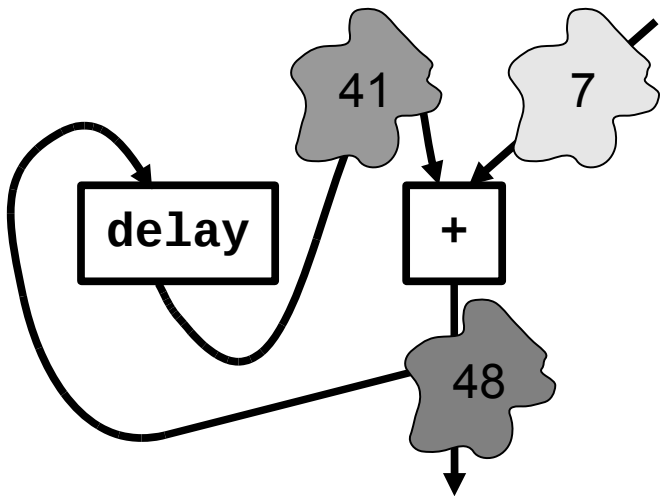
Solution: external references pass through restrict

Proto's Families of Primitives

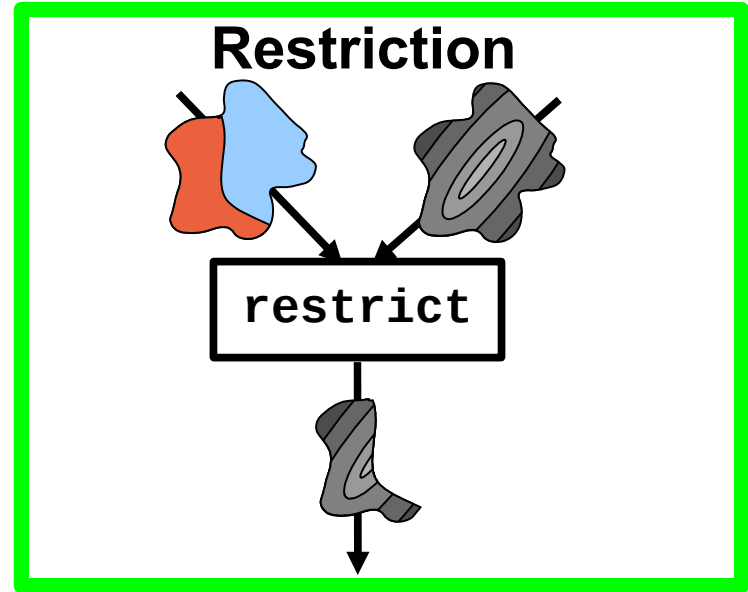
Pointwise



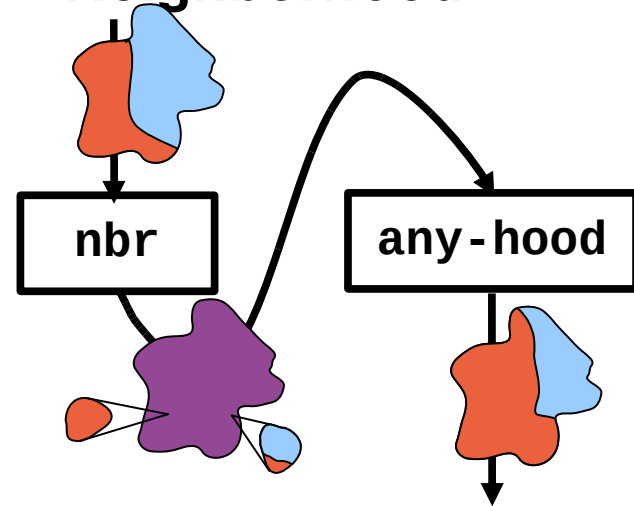
Feedback



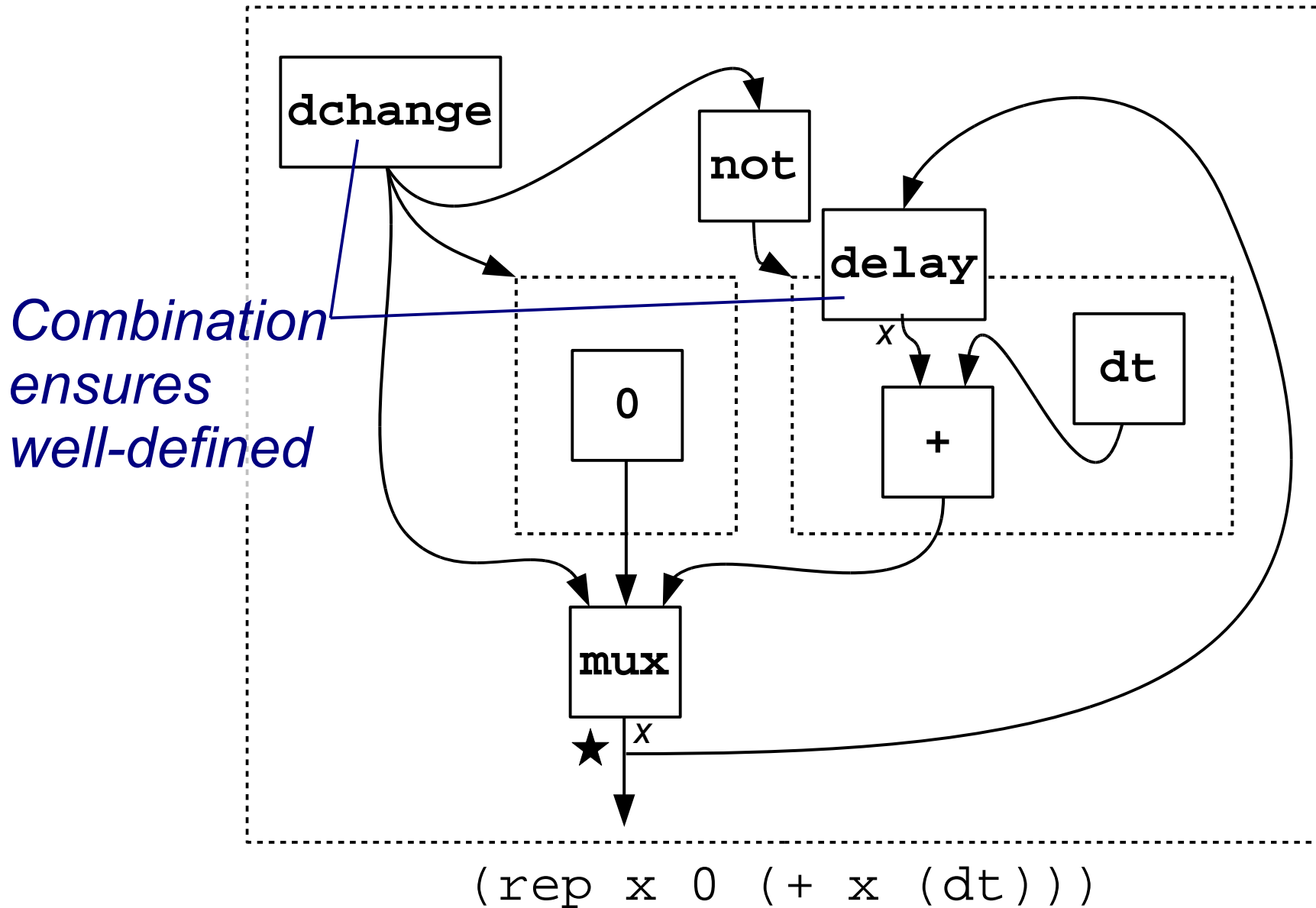
Restriction



Neighborhood

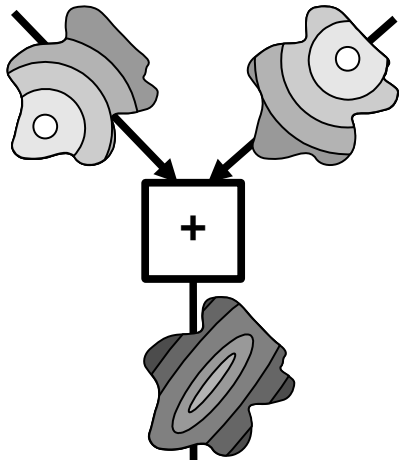


Branches → Feedback Operators

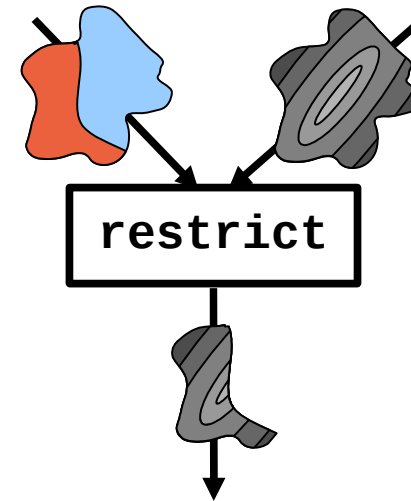


Proto's Families of Primitives

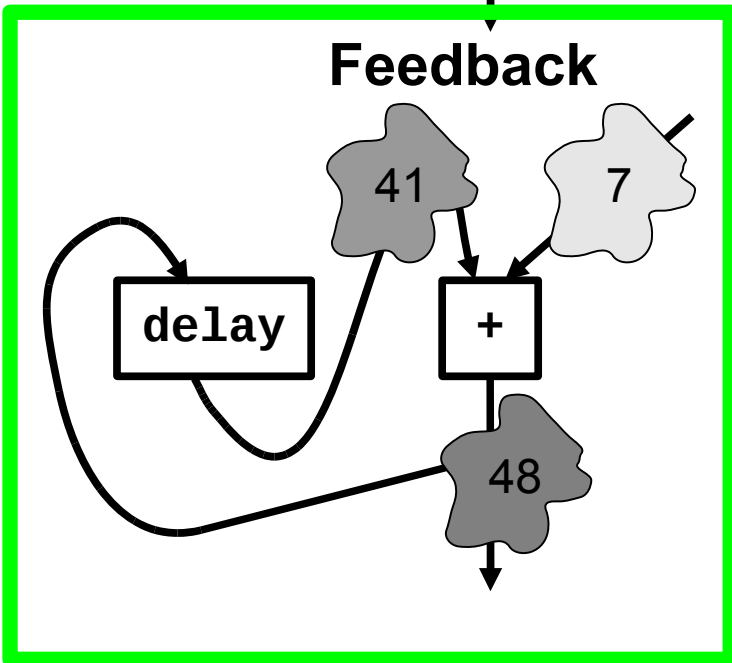
Pointwise



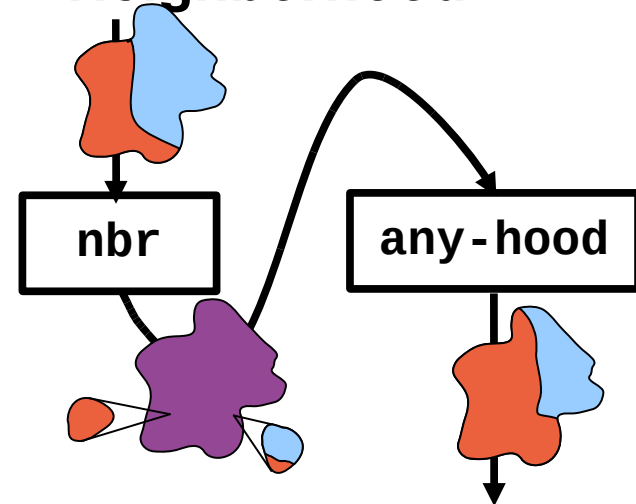
Restriction



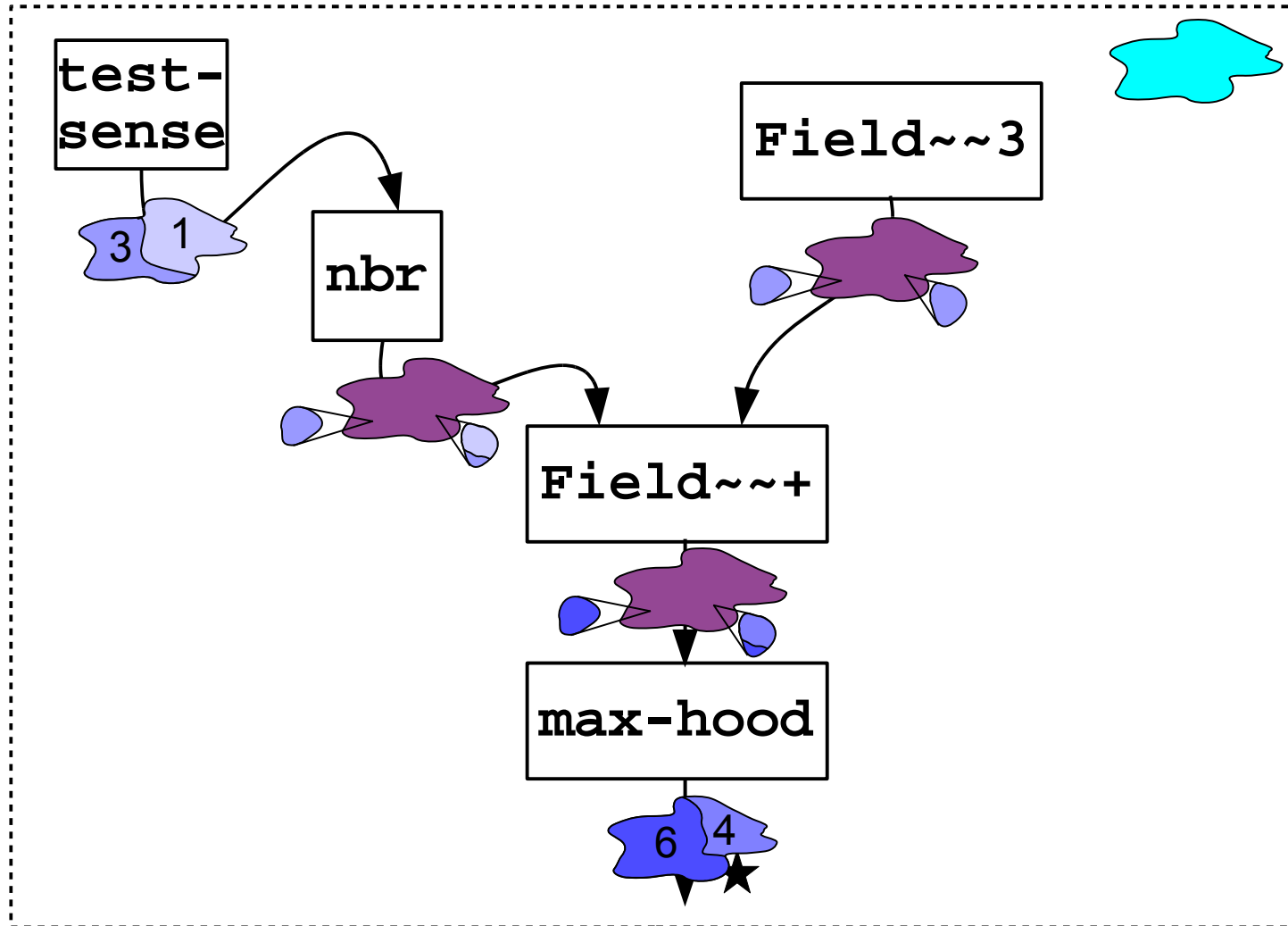
Feedback



Neighborhood

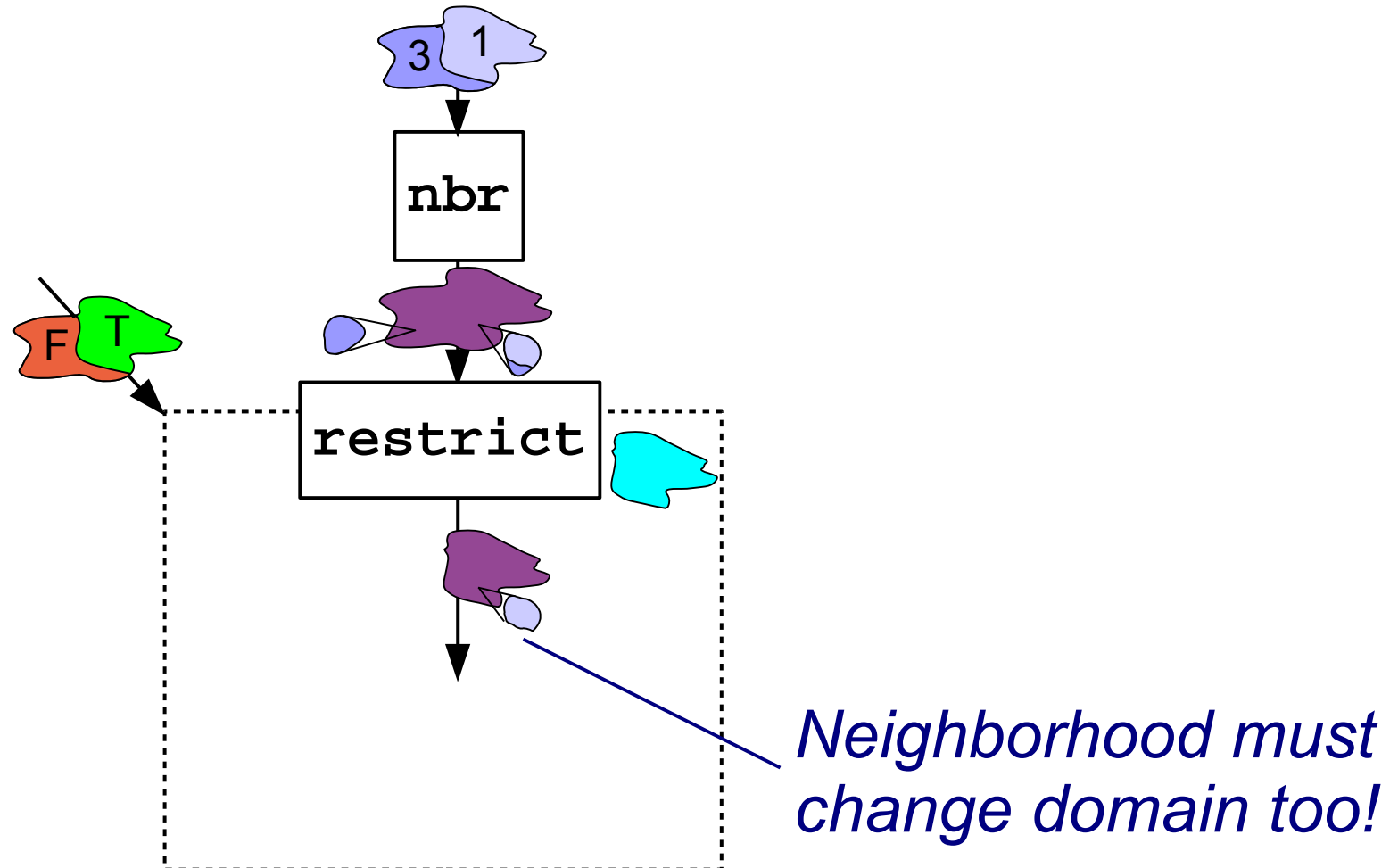


Neighborhood Ops



`(max-hood (+ (nbr (test-sense)) 3))`

Restricting Neighborhoods



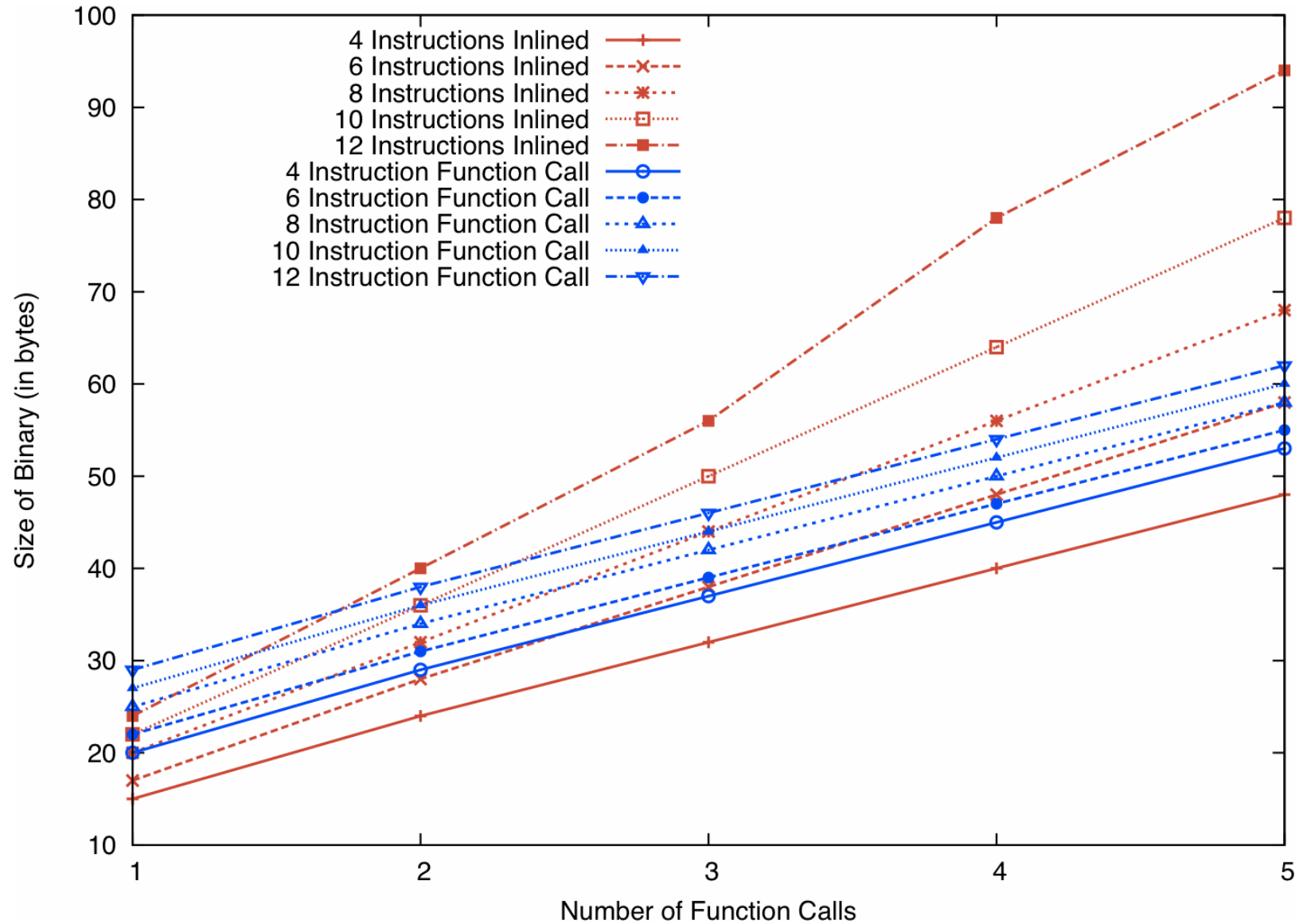
Same problem as for restriction without if construct

Solution: compile-time error checking

Implementation in Proto

- Previously, all function calls were inlined!
- Upgrades:
 - Function call opcodes added to VM
 - External references → implicit arguments
 - Error checking for bad `nbr` / `if` interactions

Verification of Smaller Code Size



Distributed function call problems:

- Action at a distance
 - Recursion
 - Function equivalence
 - *1st class fns*
- [c.f. Beal, 2009]



(partly implemented)

*(except part
of 1st class)*

Contributions & Future Work

- Evaluation model for space-time function calls
- Analysis of Proto operator interactions
- Implementation of call-in-place model in Proto
- Future work: 1st-class fns, implement recursion

<http://proto.bbn.com/>