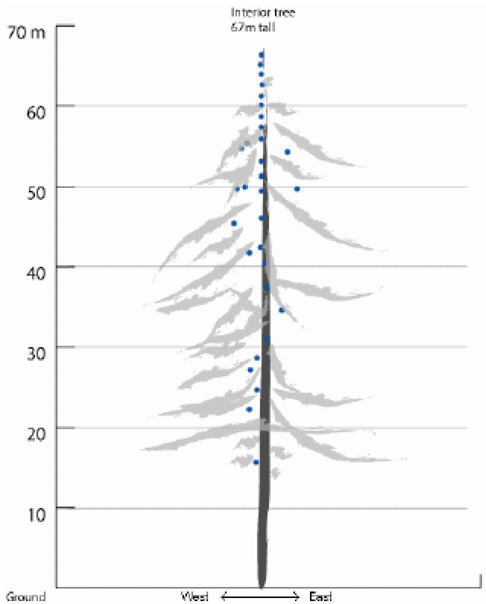
Programming Spatial Computers

Jonathan Bachrach & Jacob Beal
October, 2006

Space-filling Computers

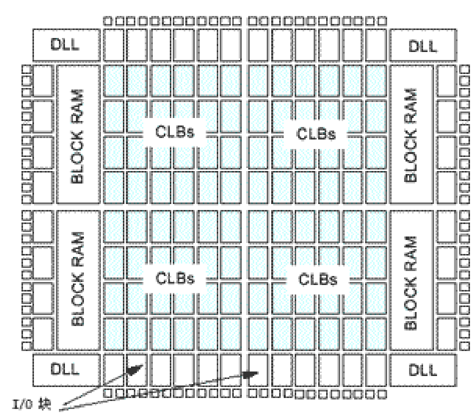


Sensor networks

Biological Computing

Robot Swarms

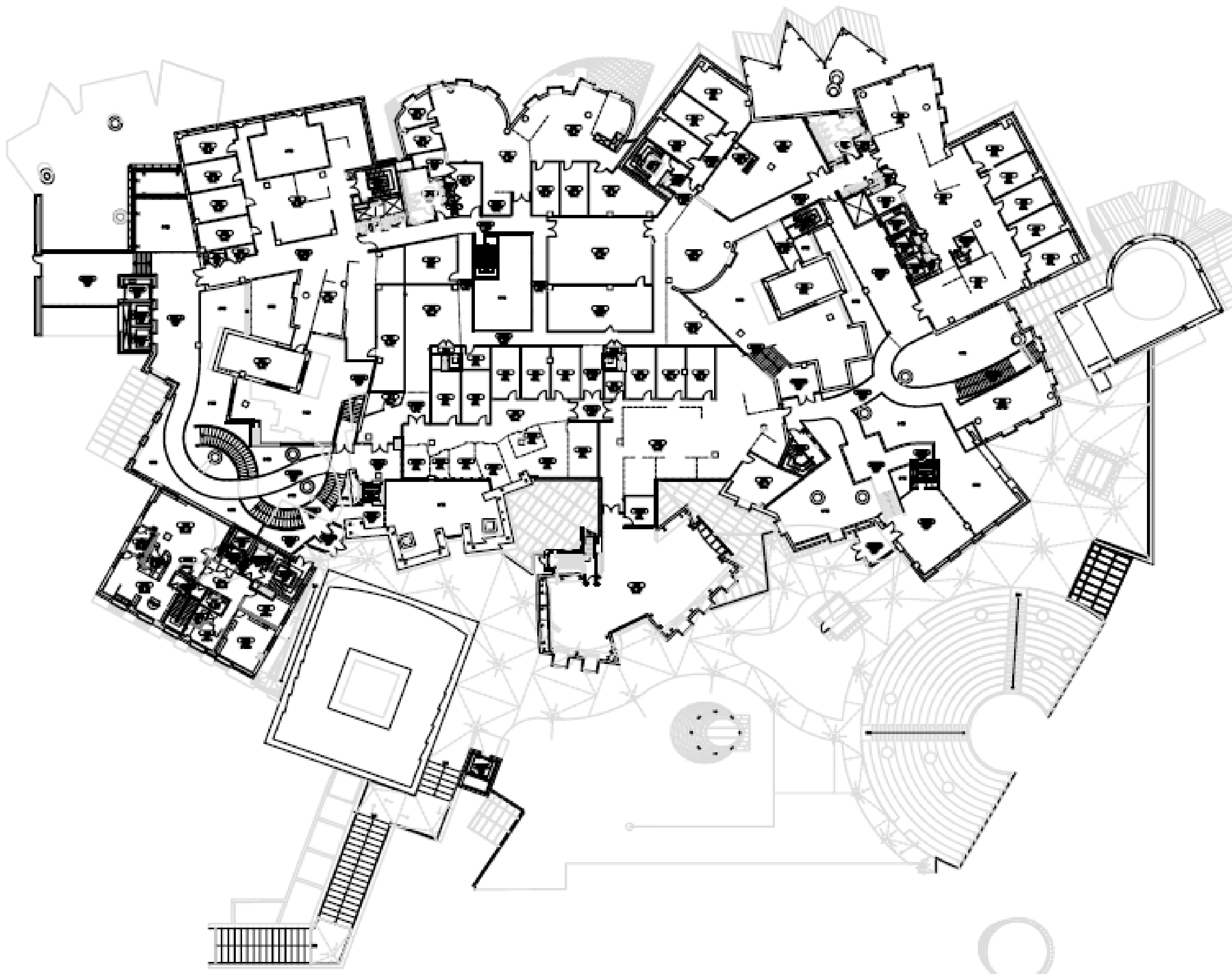
Distributed Control Systems



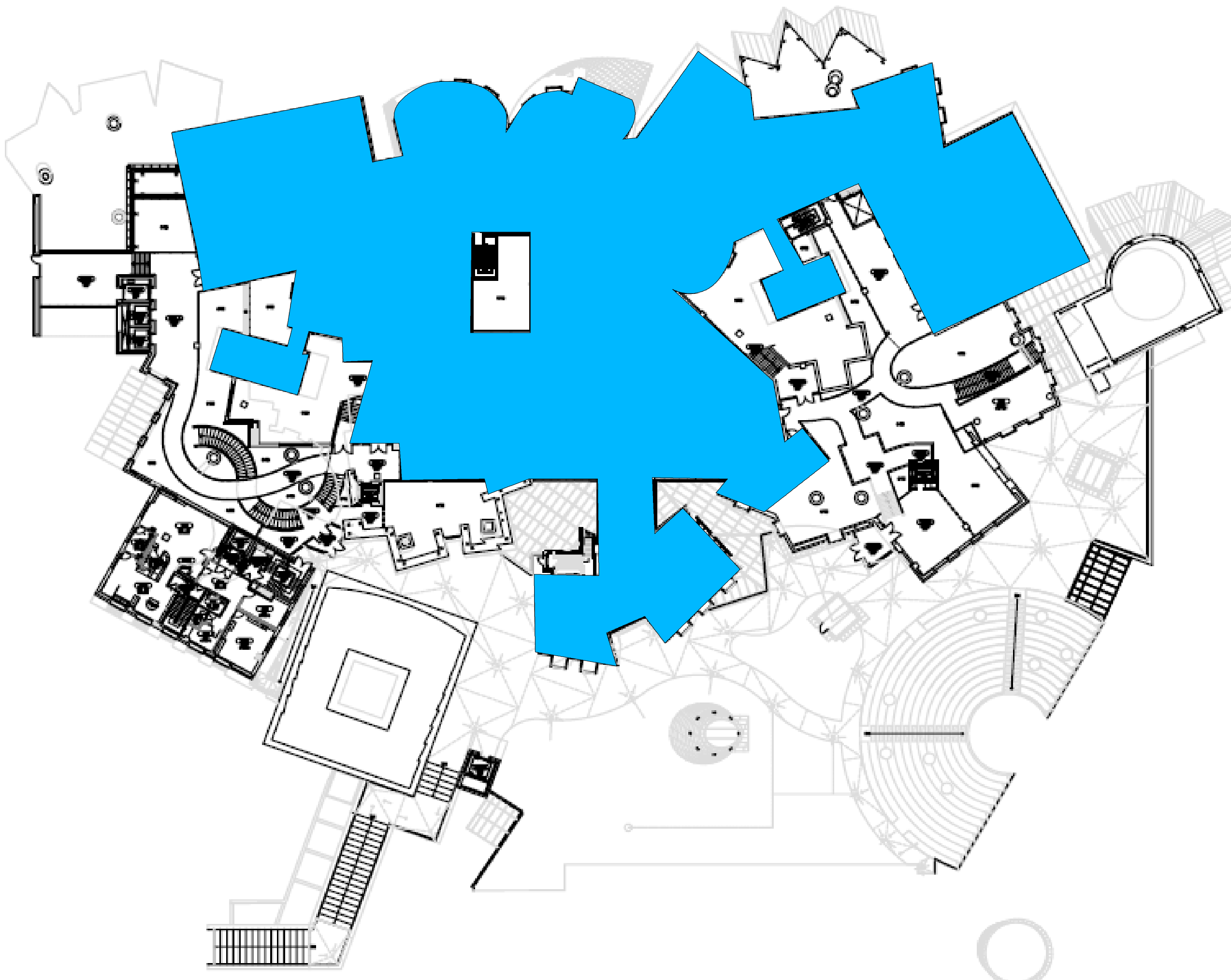
FPGAs

Programmable Matter

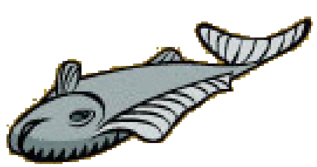
Amorphous Medium Approach



Amorphous Medium Approach



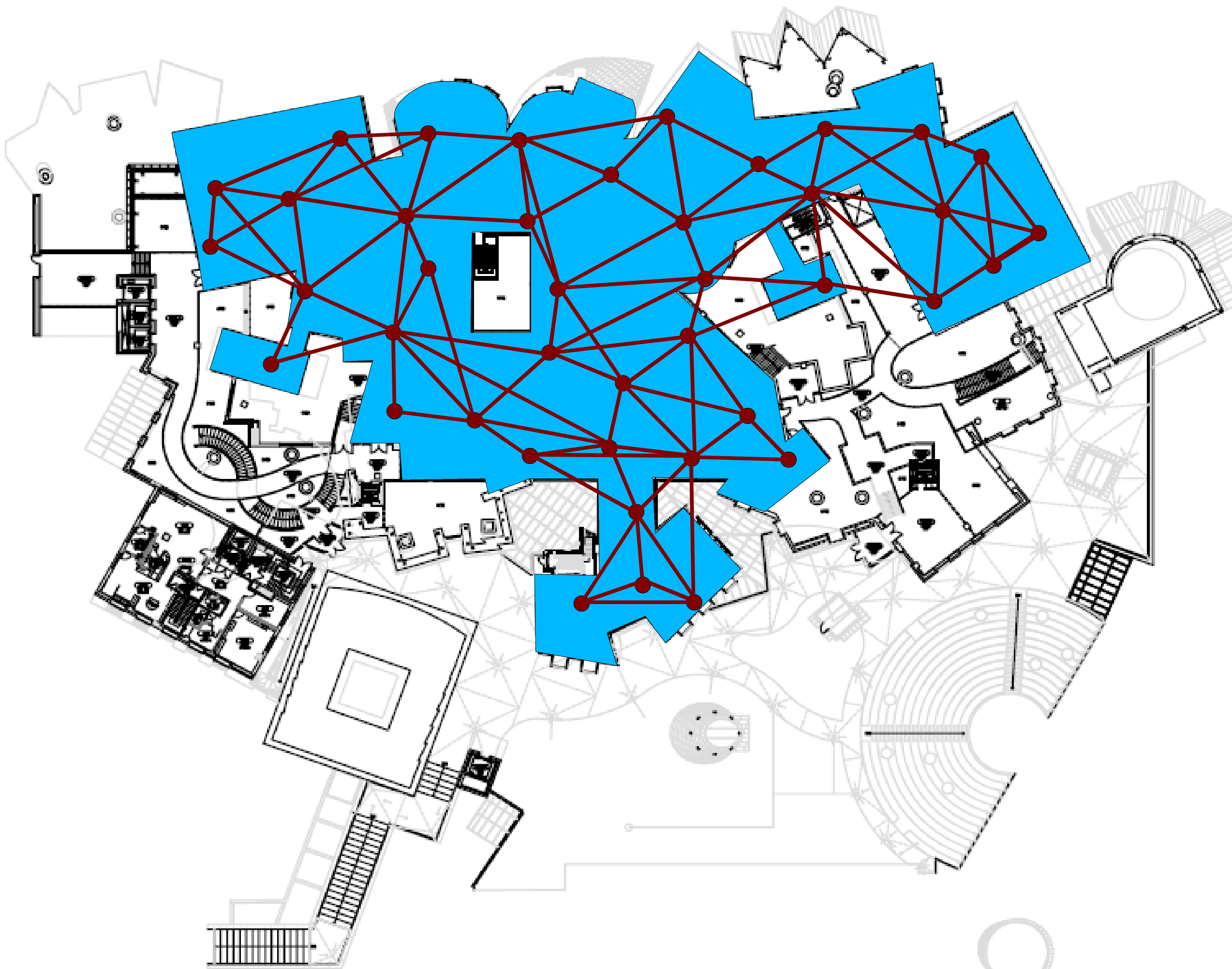
Amorphous Medium Approach



?

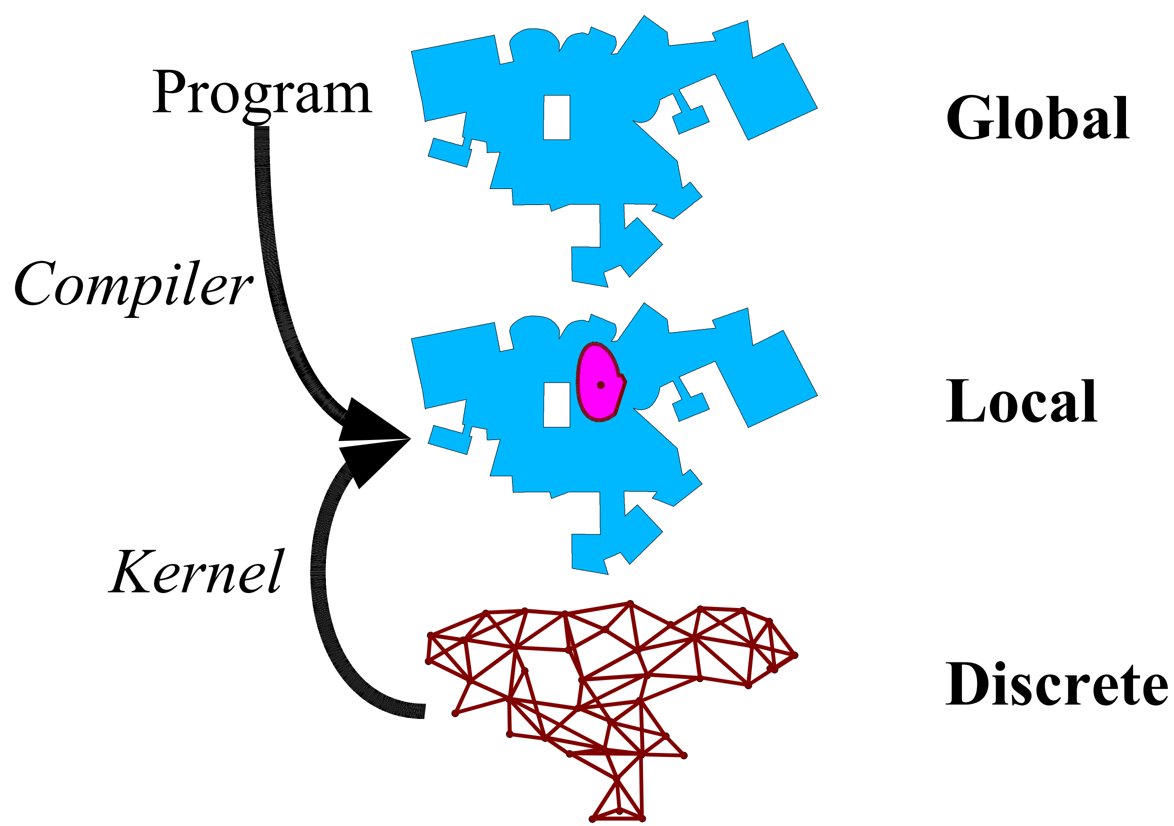
Program the space... approximate with a network

Amorphous Medium Approach

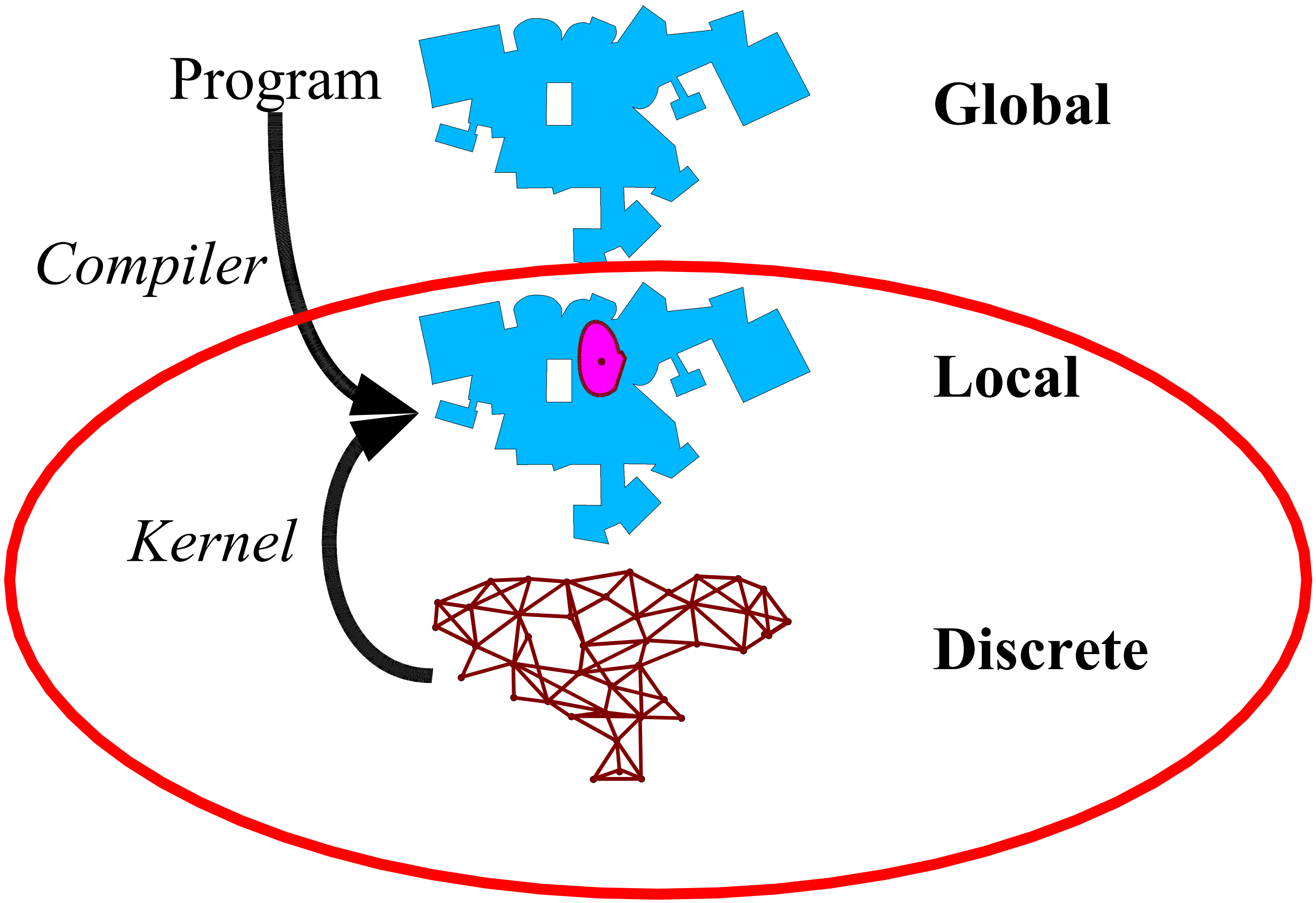


The discretization hardly matters!

Global v. Local v. Discrete



Global v. Local v. Discrete



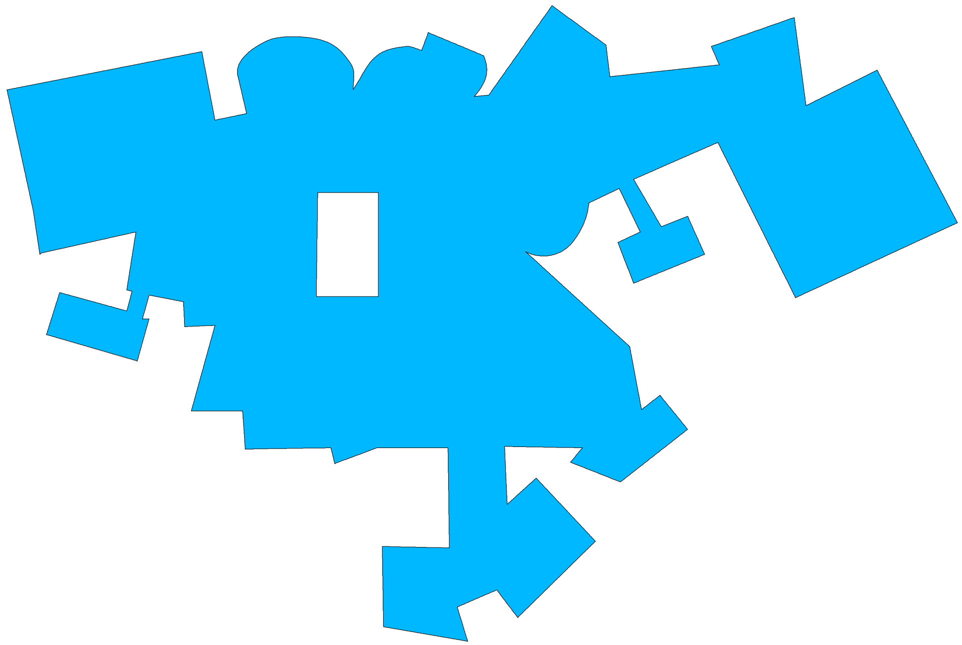
Discrete Model

- Dozens to billions of simple, unreliable agents
- Distributed through space, communicating by local broadcast
- Agents may be added or removed
- No guaranteed global services (e.g. time, naming, routing, coordinates)
- Relatively cheap power, memory, processing
- Partial synchrony

Kernel

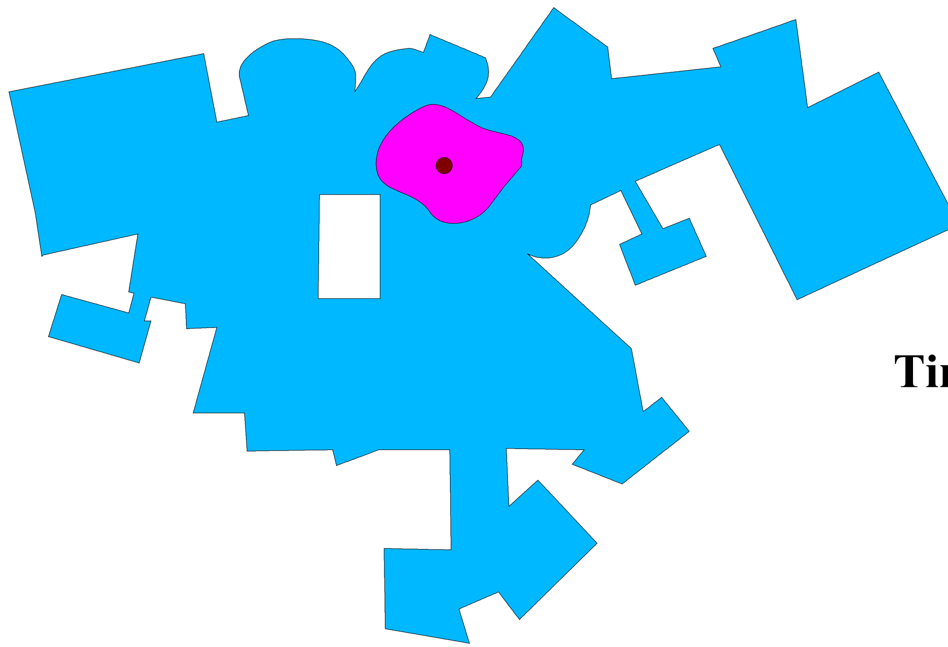
- Responsibilities:
  - Emulate amorphous medium
  - Time evolution
  - Interface with sensors, actuators
  - Viral reprogramming
- Current platforms:
  - Simulator
  - Mica2 Mote
  - McLurkin's Swarm

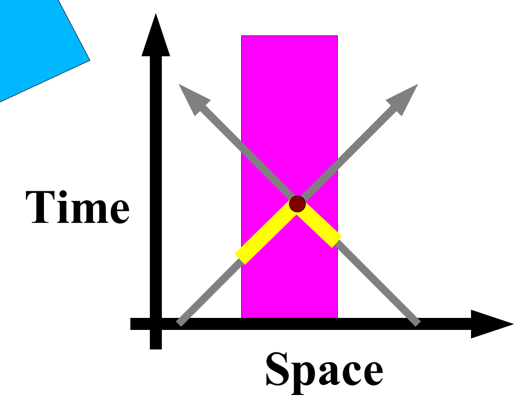
Amorphous Medium



- Manifold (locally Euclidean space)
  - Assume Riemannian, smooth, compact
  - Simple locally, complex globally

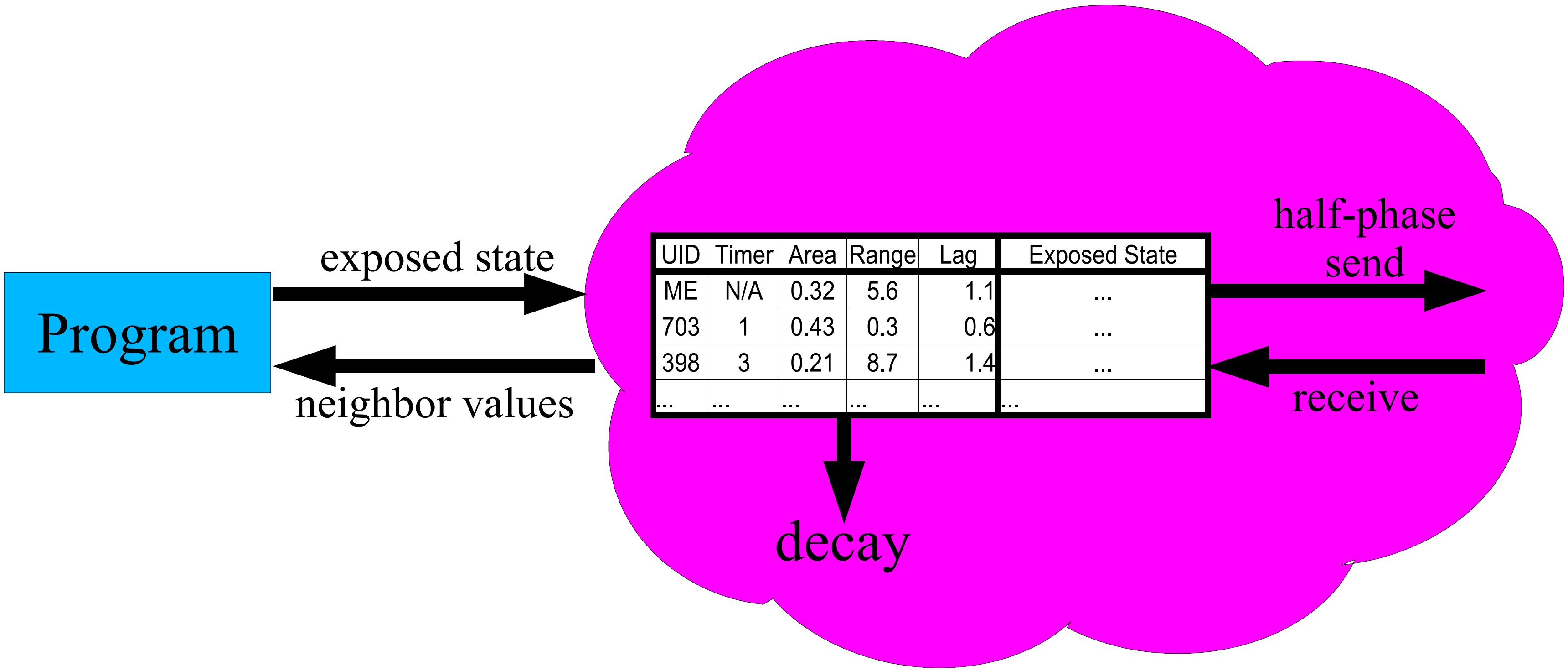
Amorphous Medium





- Points access past values in their neighborhood
  - Information propagates at a fixed rate *c*
- Evaluation is repeated at fixed intervals

Neighborhood Abstraction

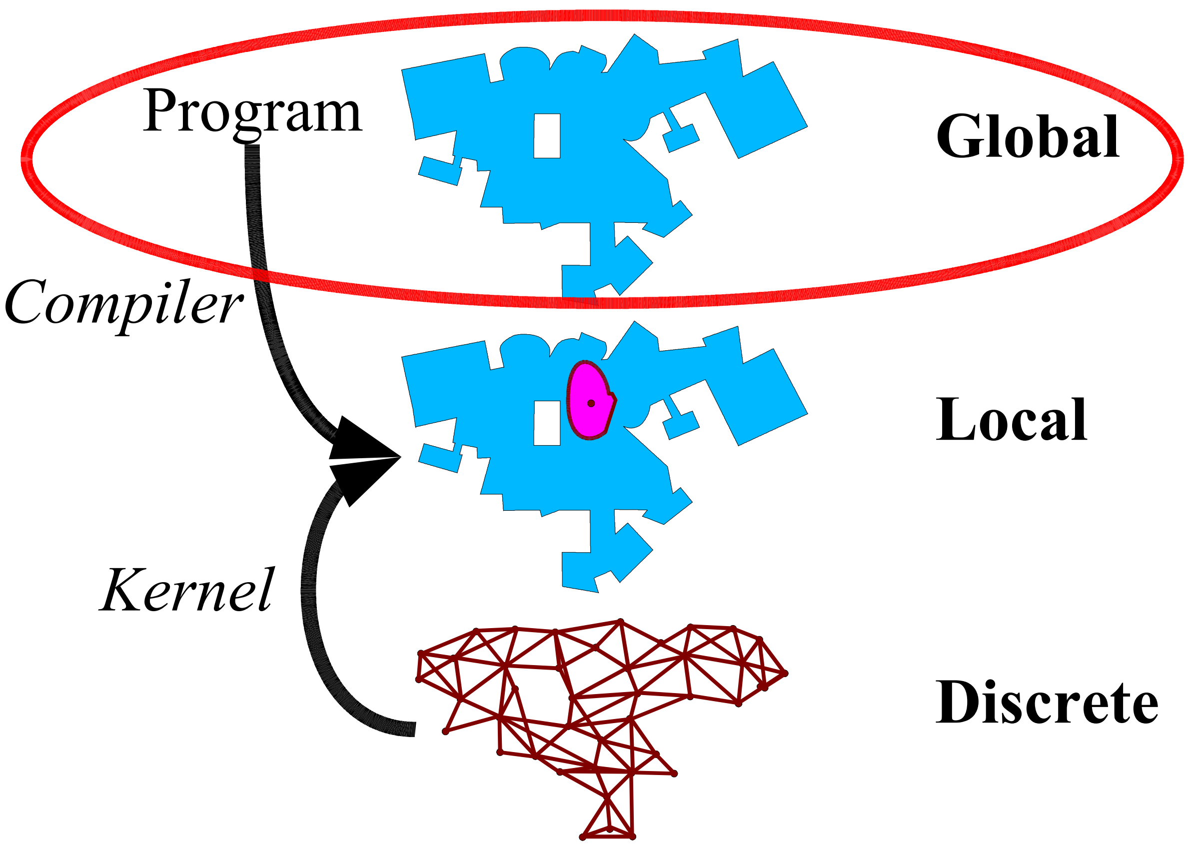


- Aggregate access to best-effort estimate of neighbor state, space-time properties
- Neighbors decay without updates

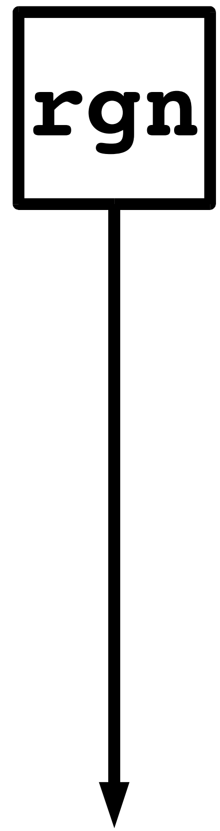
Kernel Open Questions

- What is the optimal replacement policy when there are more neighbors than table memory?
- What is the optimal decay rate?
- How much energy can be saved by throttling update and decay rates?
- What are good ways to expose the cost/responsiveness tradeoff to the programmer?

Global v. Local v. Discrete



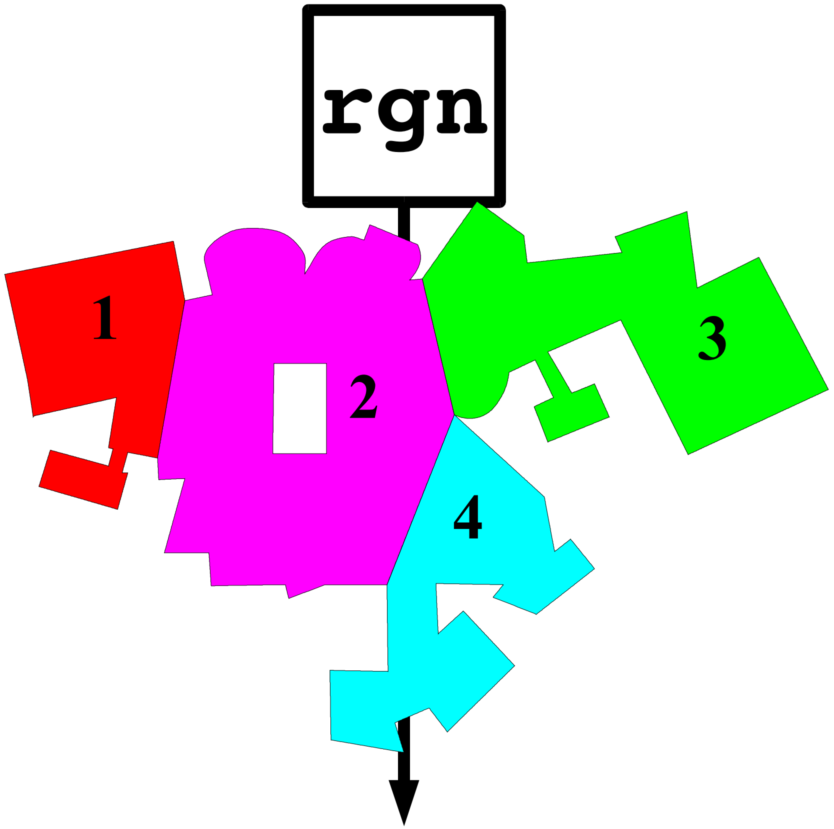
Expressions



- An expression maps a manifold to a field

  `rgn`: $\mathbf{M} \rightarrow (\mathbf{M} \rightarrow \mathrm{R})$

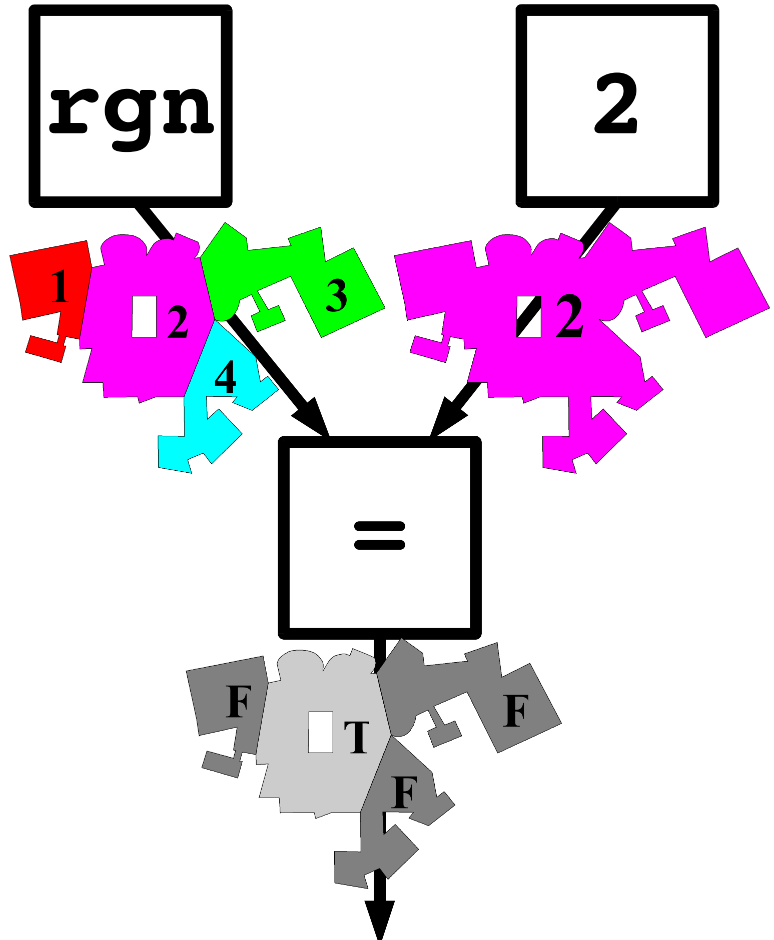
Expressions



- An expression maps a manifold to a field

`rgn`: $\mathbf{M} \rightarrow (\mathbf{M} \rightarrow \mathrm{R})$

Operators



- Operators map fields to fields **`(= rgn 2)`**

  `=`: $(\mathbf{M}\rightarrow \mathrm{R})\times(\mathbf{M}\rightarrow \mathrm{R}) \rightarrow (\mathbf{M}\rightarrow \mathrm{B})$

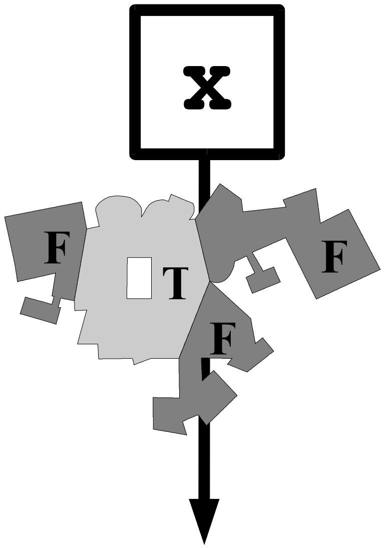
Composition & Abstraction

- Functional composition:
  - operator ◦ expressions = expression
  - operator ◦ operators = operator
  - *scope* ◦ expression = operator

Lambda!

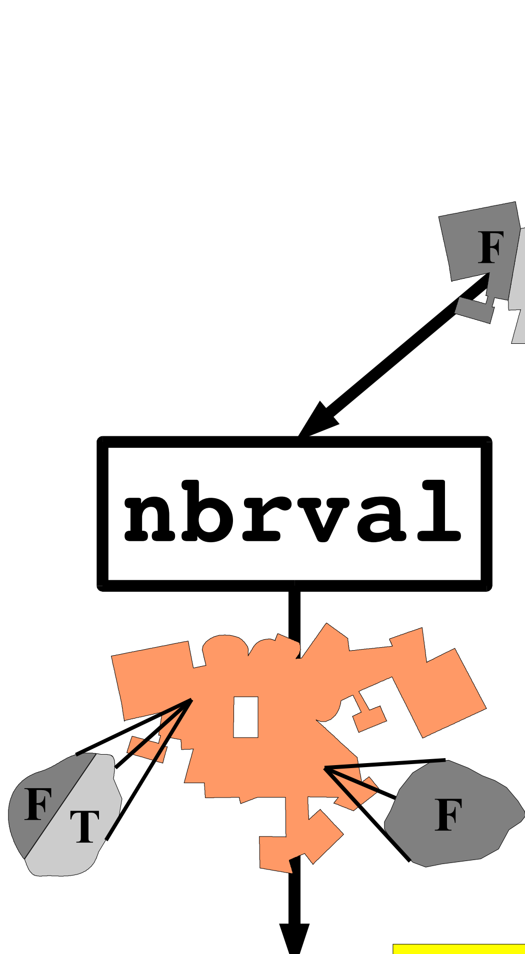
Purely functional pointwise computation

Computation over Neighborhoods



`(or ((`*n*` xor) (nbrval x) (local x)))`

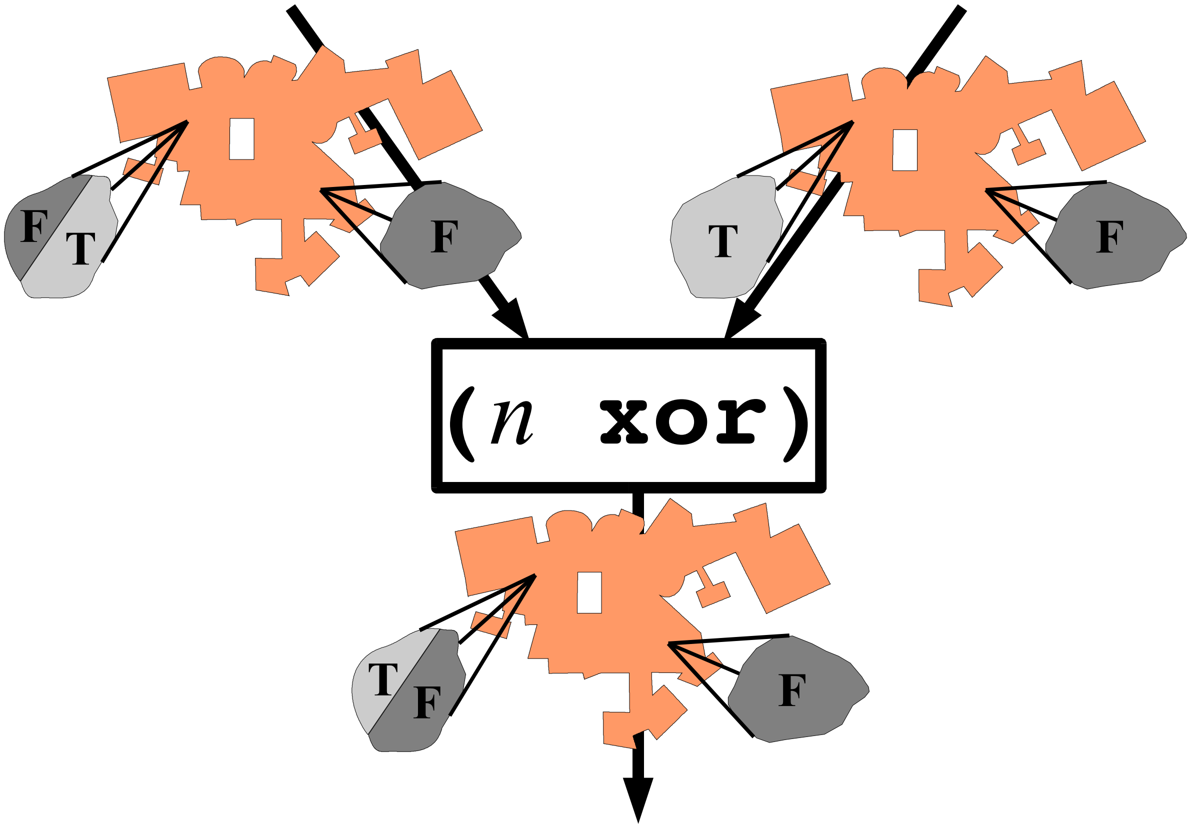
Computation over Neighborhoods



(or ((*n* xor) (nbrval x) (local x)))

- **local**, **nbrval** select fields of neighborhood fields

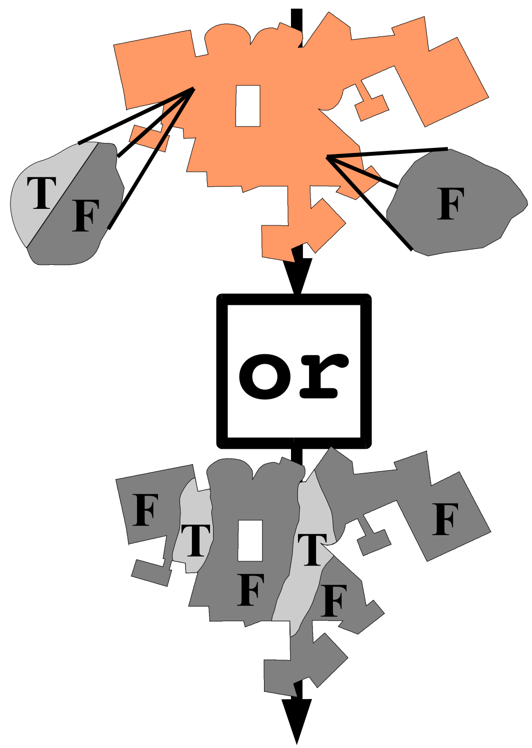
Computation over Neighborhoods



`(or ((`*n*` xor) (nbrval x) (local x)))`

– *n* applies an operator to neighborhood fields

Computation over Neighborhoods



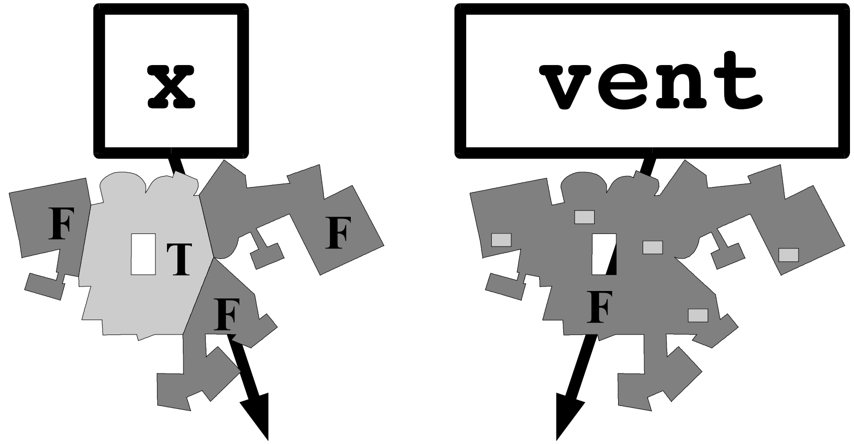
(or ((*n* xor) (nbrval x) (local x)))

- Measures (e.g. **or**, **integral**) reduce fields to values
- Sugar: **(reduce-nbrs or (xor x (local x)))**

Neighborhood Open Questions

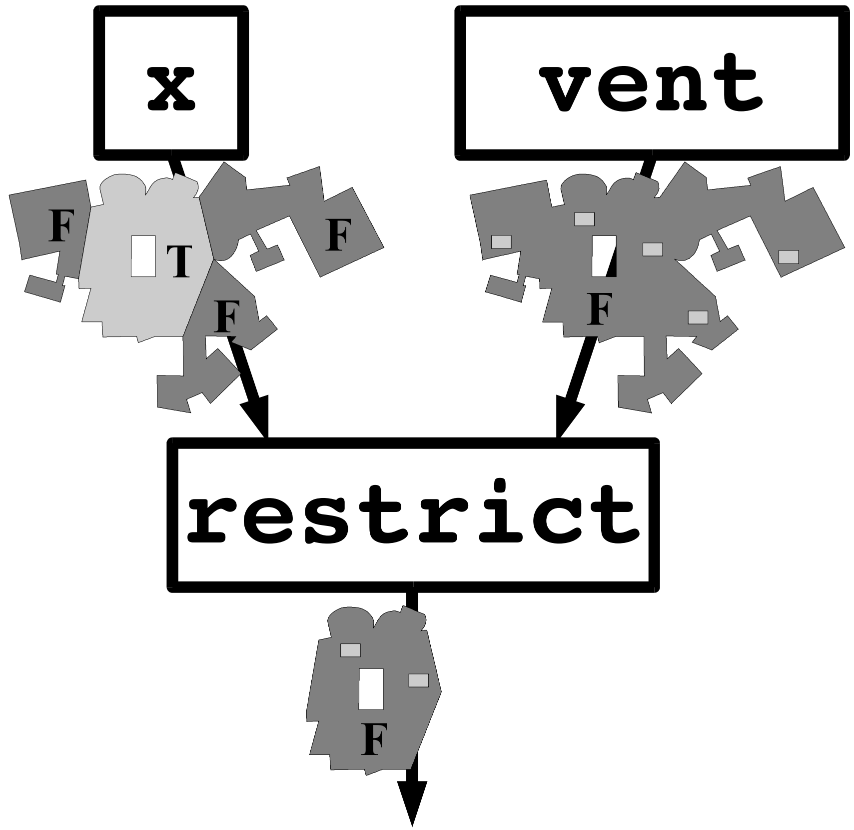
- Are the summary operations **`and`**, **`or`**, **`min`**, **`max`**, and **`integral`** sufficient for all approximatable continuous neighborhood computations?
- Are the field primitives **`local`**, **`nbrval`** **`nbr-range`**, **`nbr-lag`**, and **`nbr-bearing`** sufficient sources of neighborhood data?
- What is the discretization error of arbitrary composite neighborhood computations?

Conditional Computation



`(mux x (or (nbrval (restrict x vent)) #F)`

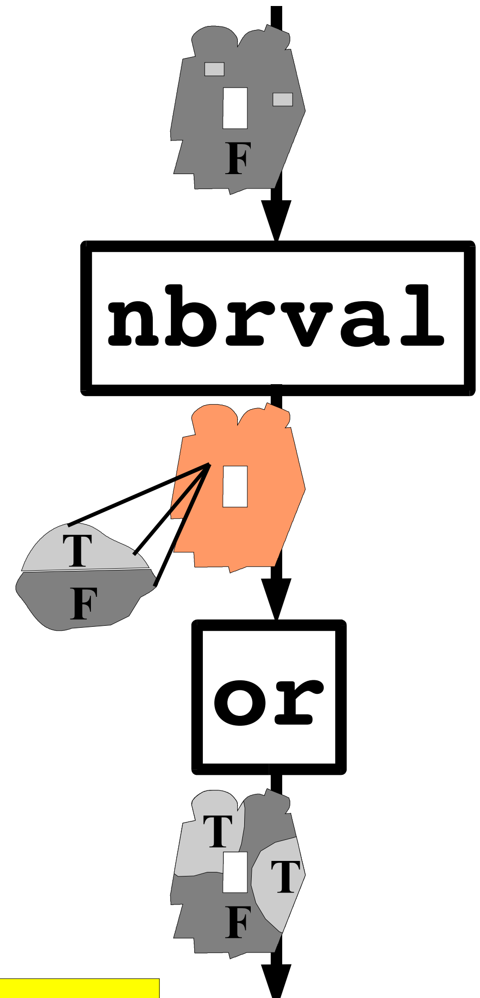
Conditional Computation



`(mux x (or (nbrval (restrict x vent)) #F)`

– **`restrict`** limits the domain of a field

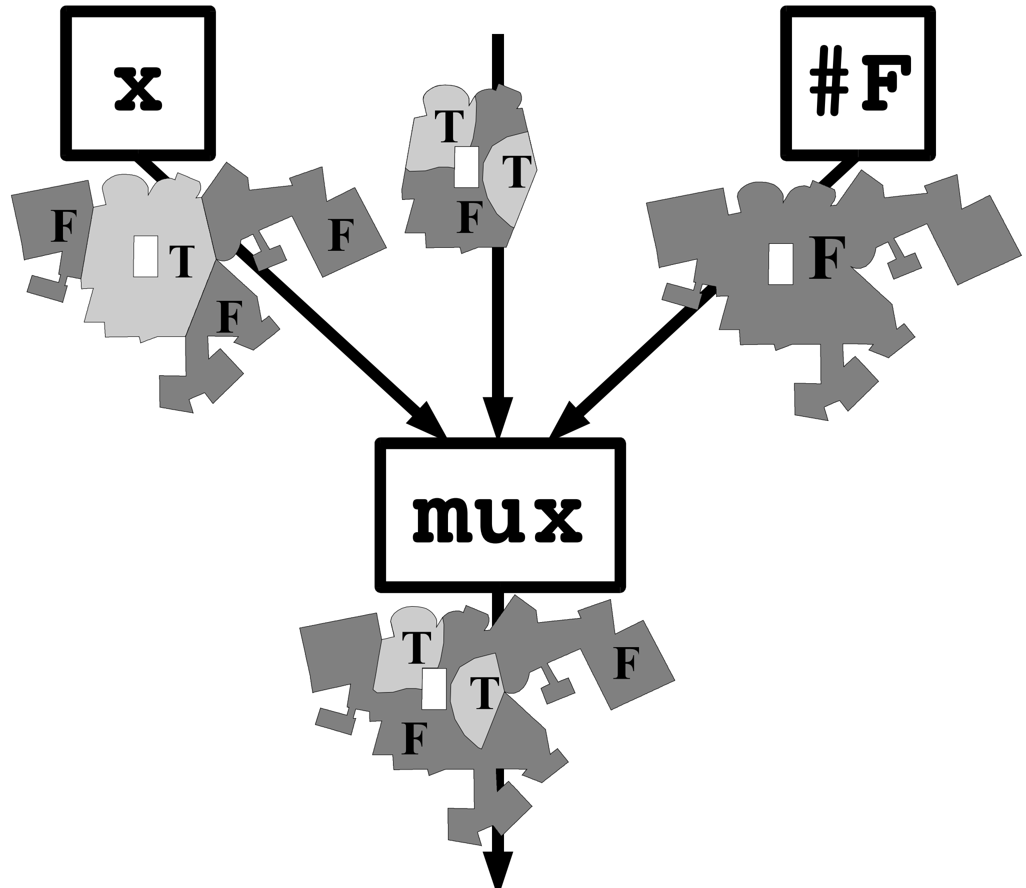
Conditional Computation



(mux x (or (nbrval (restrict x vent)) #F)

– operations proceed normally in the restricted field

Conditional Computation

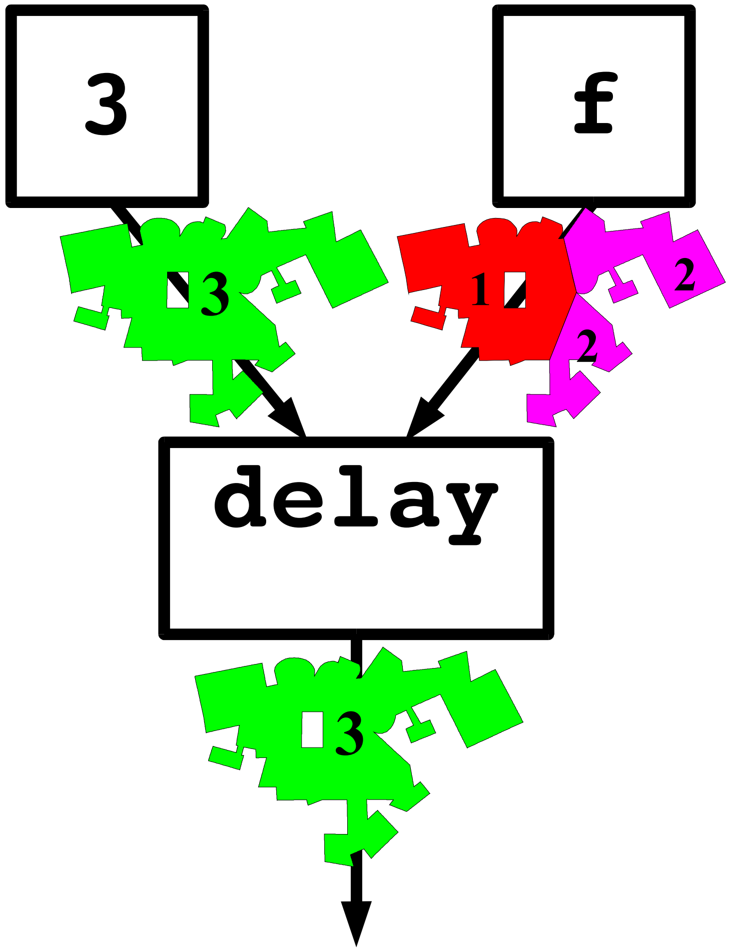


(mux x (or (nbrval (restrict x vent)) #F)

– **mux** constructs a field piecewise from inputs

– Sugar: **(if x (or (nbrval vent)))**

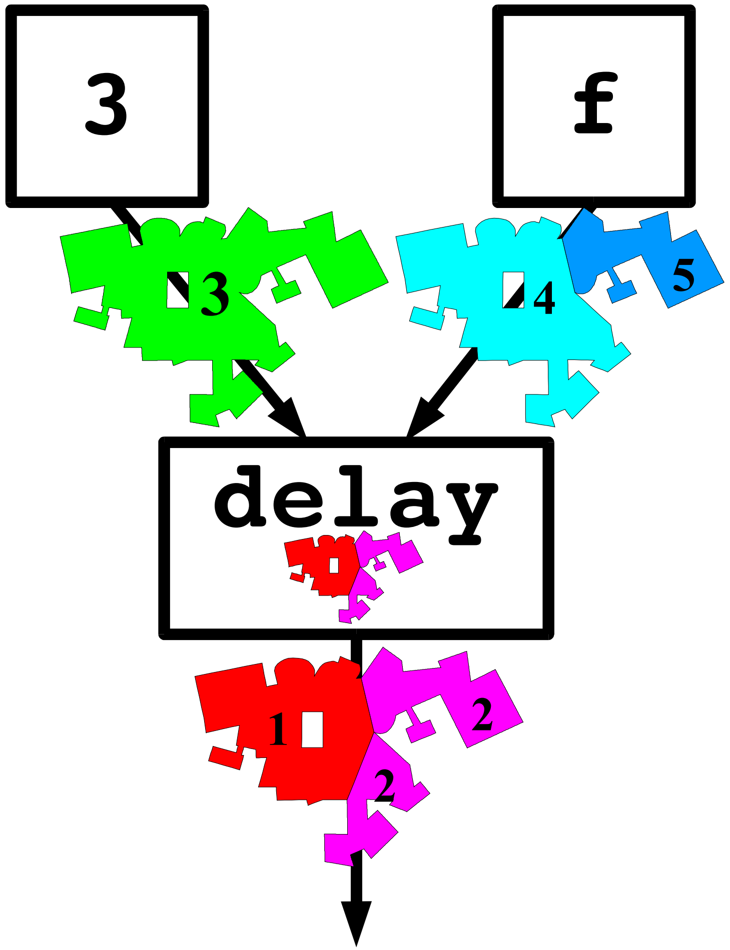
Computation with State



`(delay default init)`

– Previous values, current domain

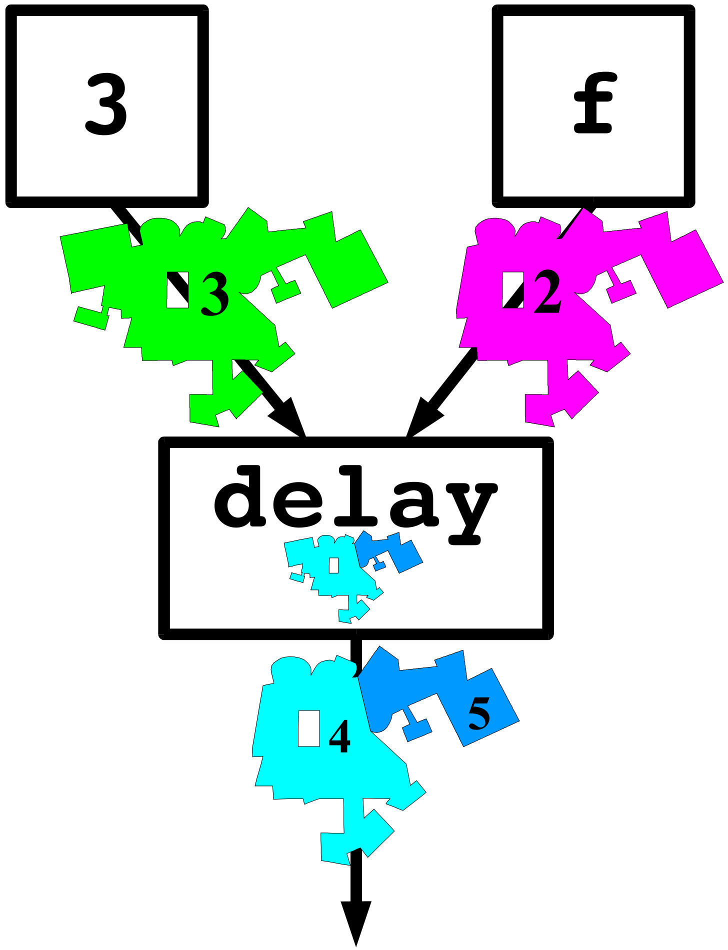
Computation with State



(delay default init)

- Previous values, current domain

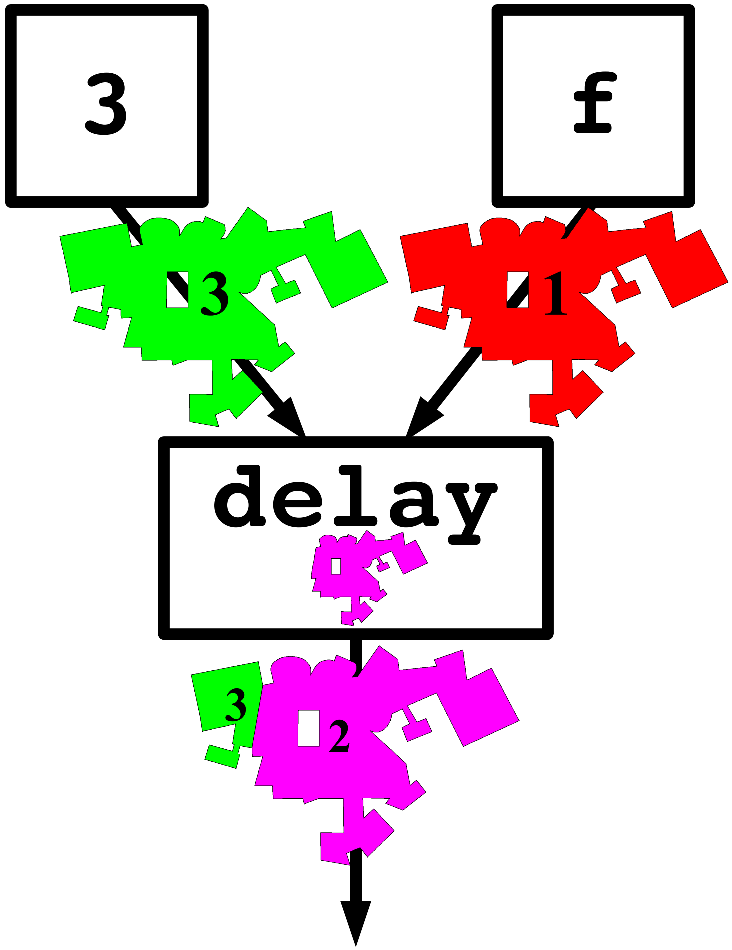
Computation with State



(delay default init)

– Previous values, current domain

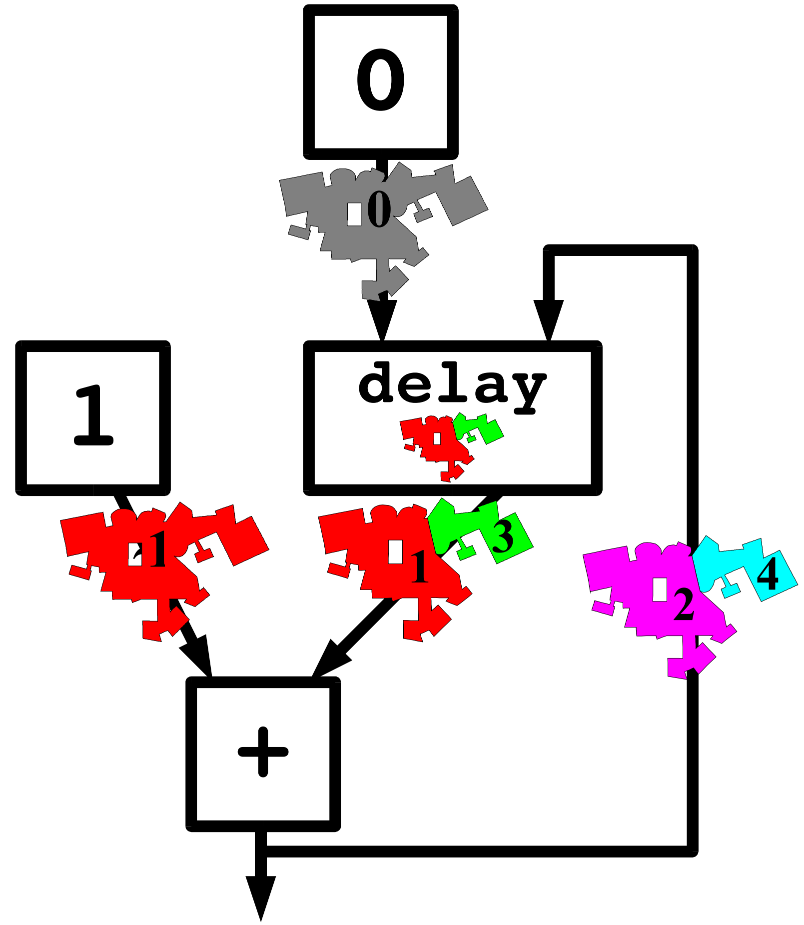
Computation with State



(delay default init)

- Previous values, current domain

Computation with State

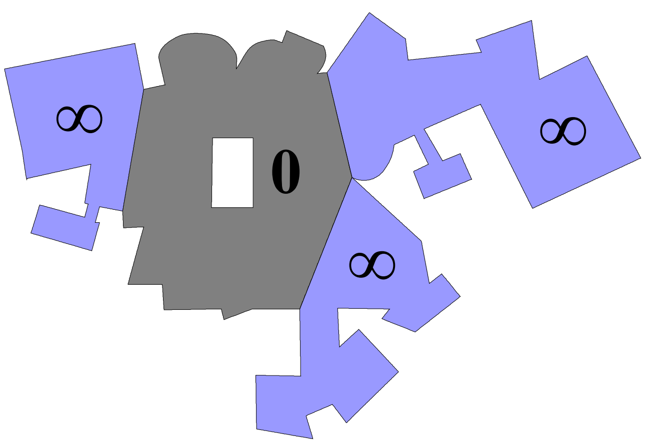


`(letfed ((n 0 (+ n 1))) n)`

Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

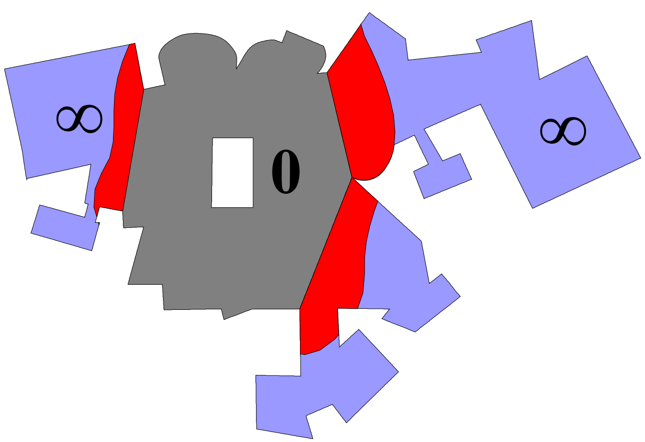
```
(lambda (src)
  (letfed
      ((d ∞ (mux src 0
              (reduce-nbrs min (+ d nbr-range)))))
    d))
```



Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

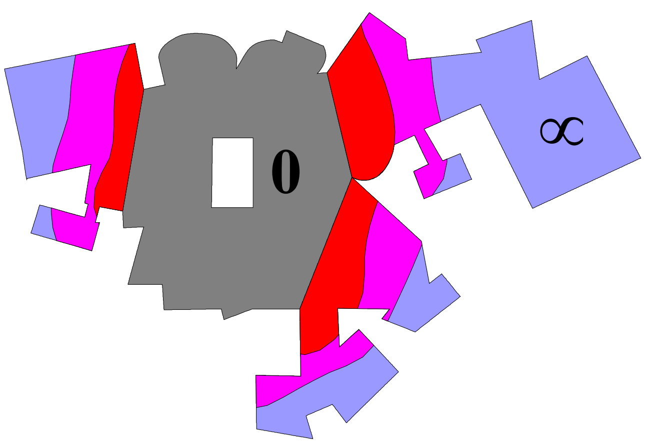
```
(lambda (src)
  (letfed
      ((d ∞ (mux src 0
                 (reduce-nbrs min (+ d nbr-range)))))
    d))
```



Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

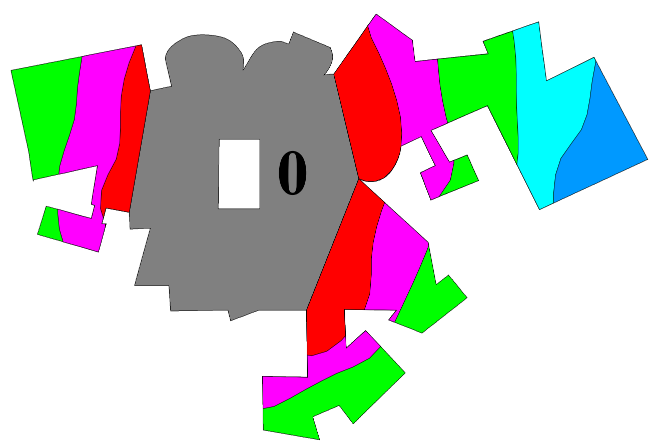
```
(lambda (src)
  (letfed
     ((d ∞ (mux src 0
              (reduce-nbrs min (+ d nbr-range)))))
   d))
```



Putting it all together

- State chains neighborhoods to arbitrary regions
  - Example: relaxation to calculate distance

```
(lambda (src)
  (letfed
      ((d ∞ (mux src 0
               (reduce-nbrs min (+ d nbr-range)))))
    d))
```

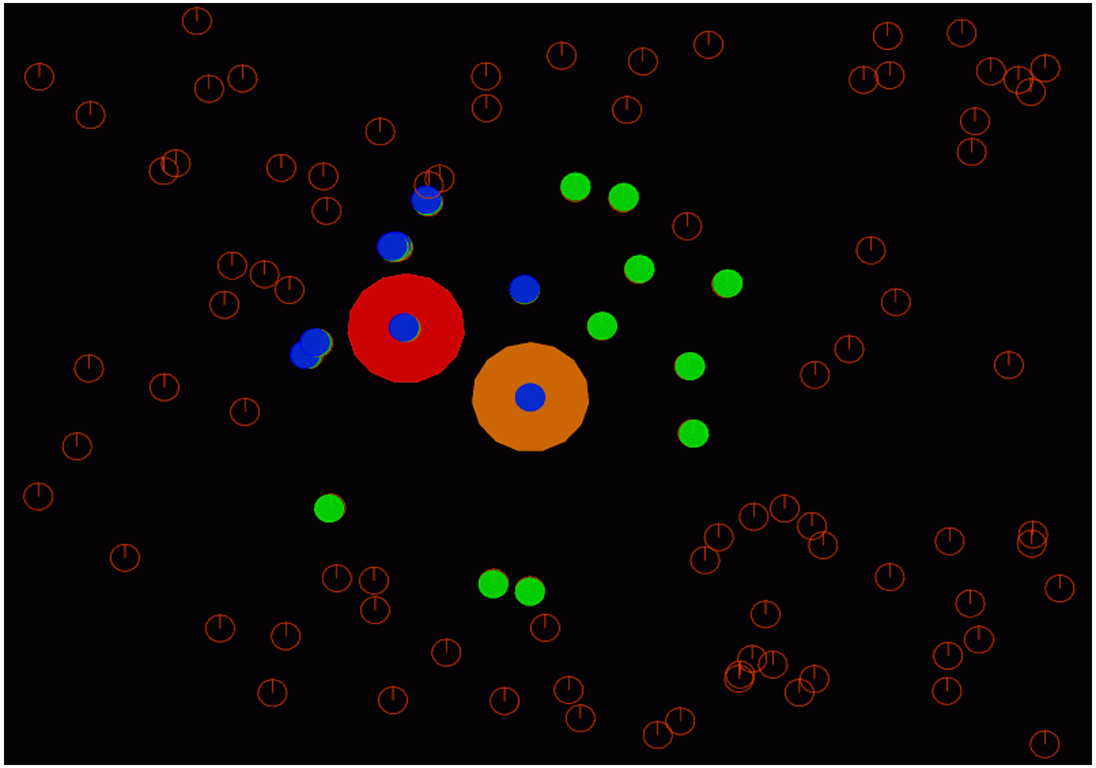


Program Open Questions

- Under what conditions does continuous convergence imply discrete convergence?
- How do convergence properties compose?
- Given a continuous program and desired error bounds, what discretization will suffice?
- Given a continuous program and a discretization, what will the error bounds be?

Programs scale gracefully

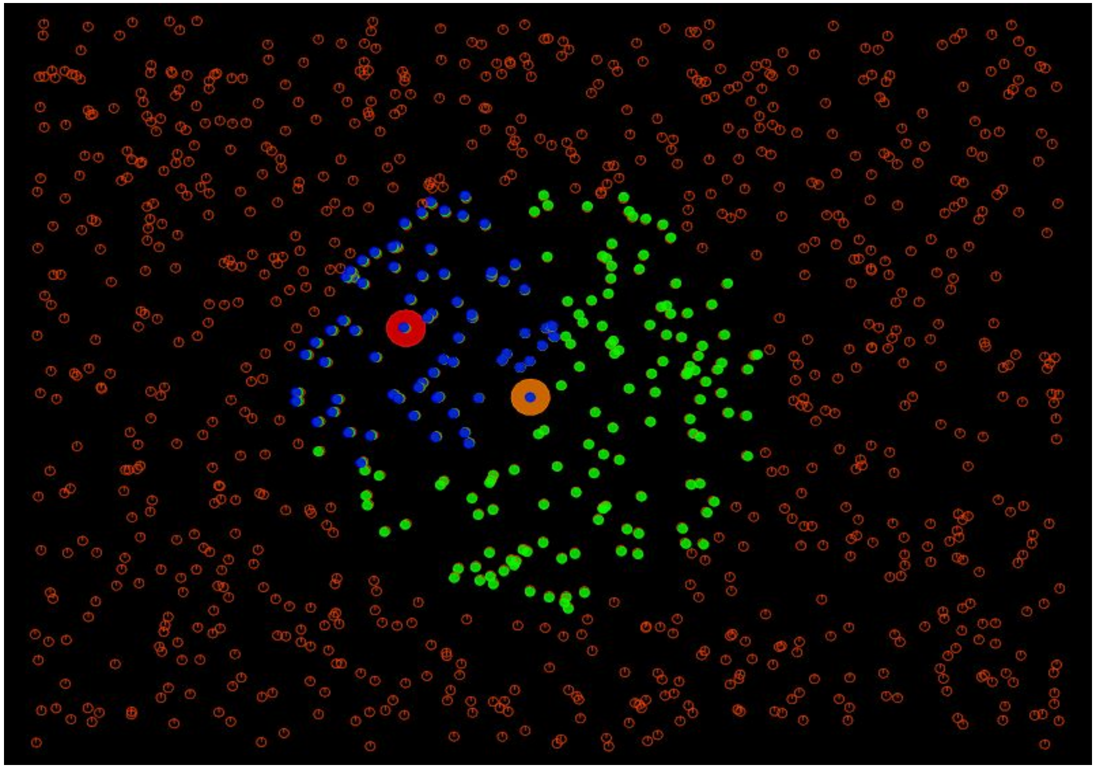
100 nodes



```
(and (green (dilate (sense 1) 30))
     (blue (dilate (sense 2) 20)))
```

Programs scale gracefully

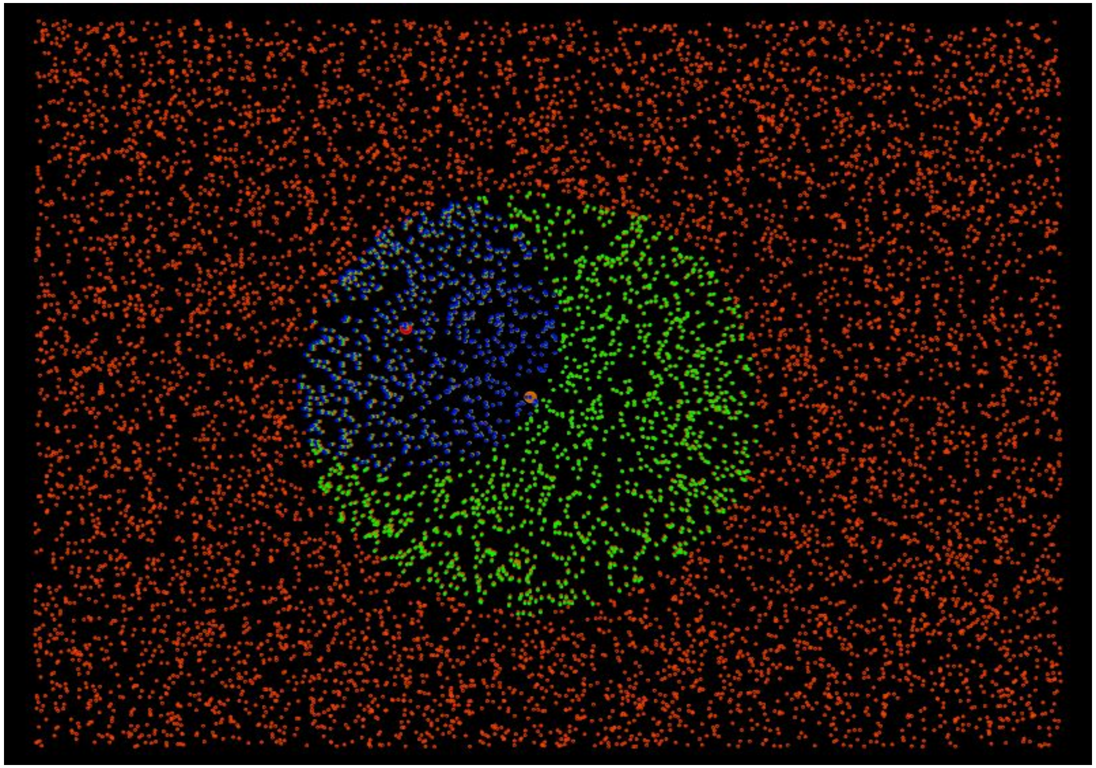
1,000 nodes



```
(and (green (dilate (sense 1) 30))
     (blue (dilate (sense 2) 20)))
```

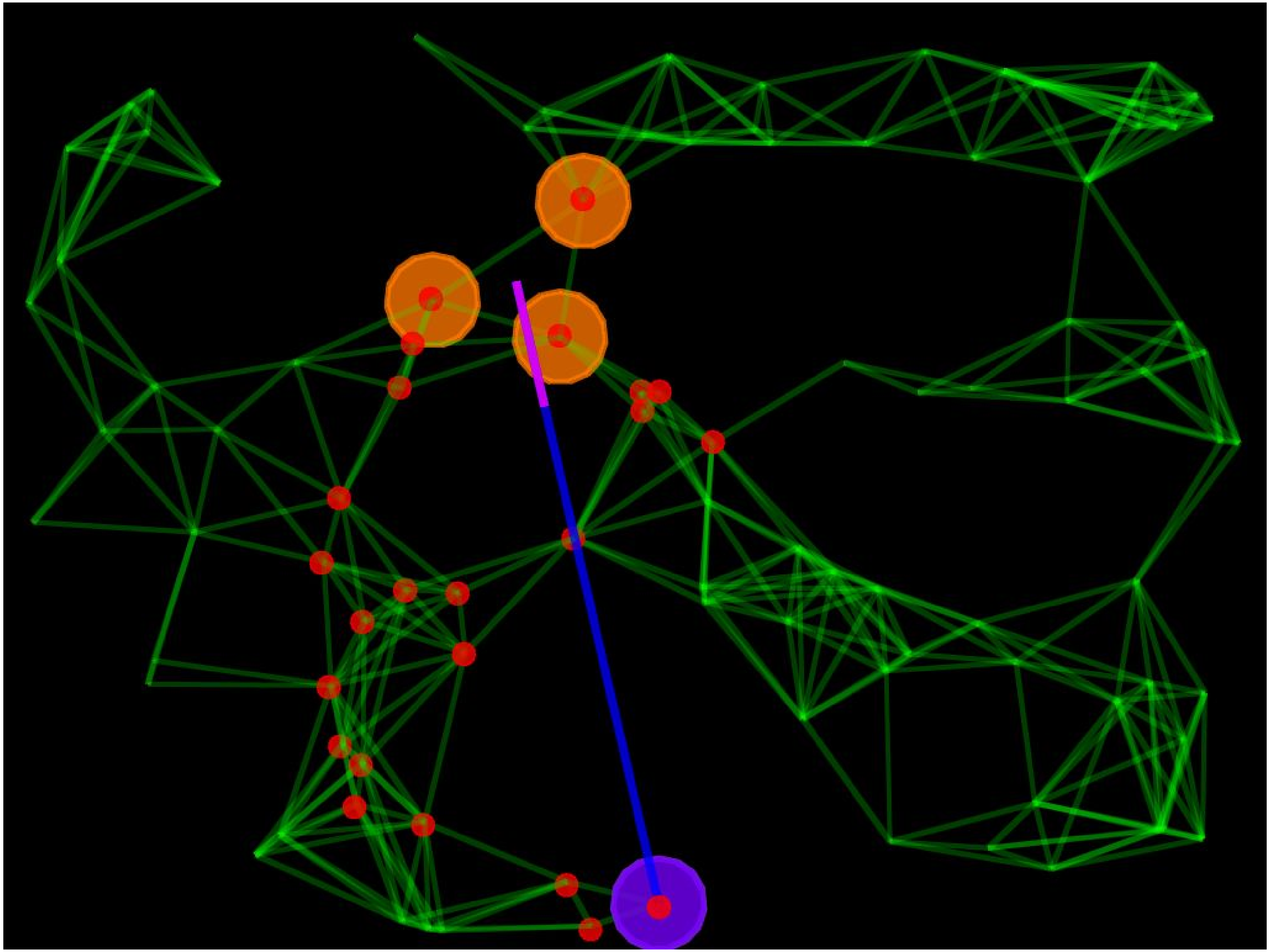
Programs scale gracefully

10,000 nodes



```
(and (green (dilate (sense 1) 30))
     (blue (dilate (sense 2) 20)))
```

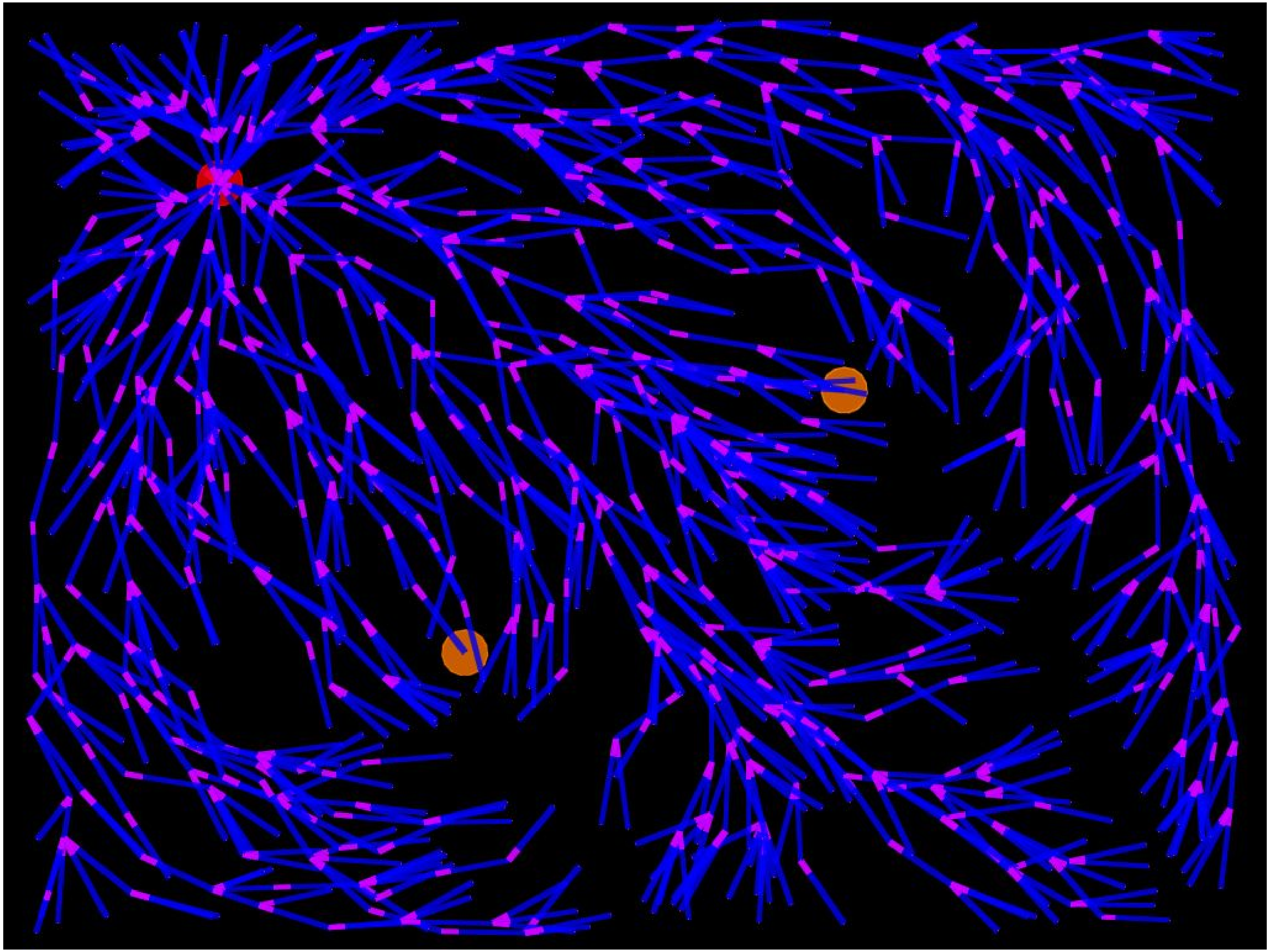
Target Tracking



Target Tracking

```
(def local-average (v) (/ (reduce-nbrs v integral) (reduce-nbrs integral 1)))
(def gradient (src)
  (letfed ((n infinity
              (+ 1 (mux src 0 (reduce-nbrs min (+ n nbr-range))))))
    (- n 1)))
(def grad-value (src v)
  (let ((d (gradient src)))
    (letfed ((x 0 (mux src v (2nd (reduce-nbrs min (tup d x))))))
      x)))
(def distance (p1 p2) (grad-value p1 (gradient p2)))
(def channel (src dst width)
  (let* ((d (distance src dst))
         (trail (<= (+ (gradient src) (gradient dst)) d)))
    (dilate width trail)))
(def track (target dst coord)
  (let ((point
         (if (channel target dst 10)
           (grad-value target
             (mux target
                  (tup (local-average (1st coord))
                       (local-average (2nd coord)))
                  (tup 0 0)))
           (tup 0 0))))
    (mux dst (vsub point coord) (tup 0 0))))
```

Threat Avoidance



Threat Avoidance

```
(def exp-gradient (src d)
  (letfed ((n src (max (* d (reduce-nbrs max n)) src)))
    n))
(def sq (x) (* x x))
(def dist (p1 p2)
  (sqrt (+ (sq (- (1st p1) (1st p2)))
           (sq (- (2nd p1) (2nd p2))))))
(def l-int (p1 v1 p2 v2)
  (pow (/ (- 2 (+ v1 v2)) 2) (+ 1 (dist p1 p2))))
(def max-survival (dst v p)
  (letfed
      ((ps 0 (mux dst
                  1
                (reduce-nbrs max (* (l-int p v (local p) (local v)) ps)))))
    ps))
(def greedy-ascent (v coord)
  (- (2nd (reduce-nbrs max (tup v coord))) coord))
(def avoid-threats (dst coords)
  (greedy-ascent
   (max-survival
    dst
    (exp-gradient (sense :threat) 0.8) coords) coords))
```

Future Directions

- Continuous time evaluation
- Analysis of distortion from space discretization
- Evaluation on a changing manifold
- Actuation of the manifold
- Applications!