

# Aggregate Programming

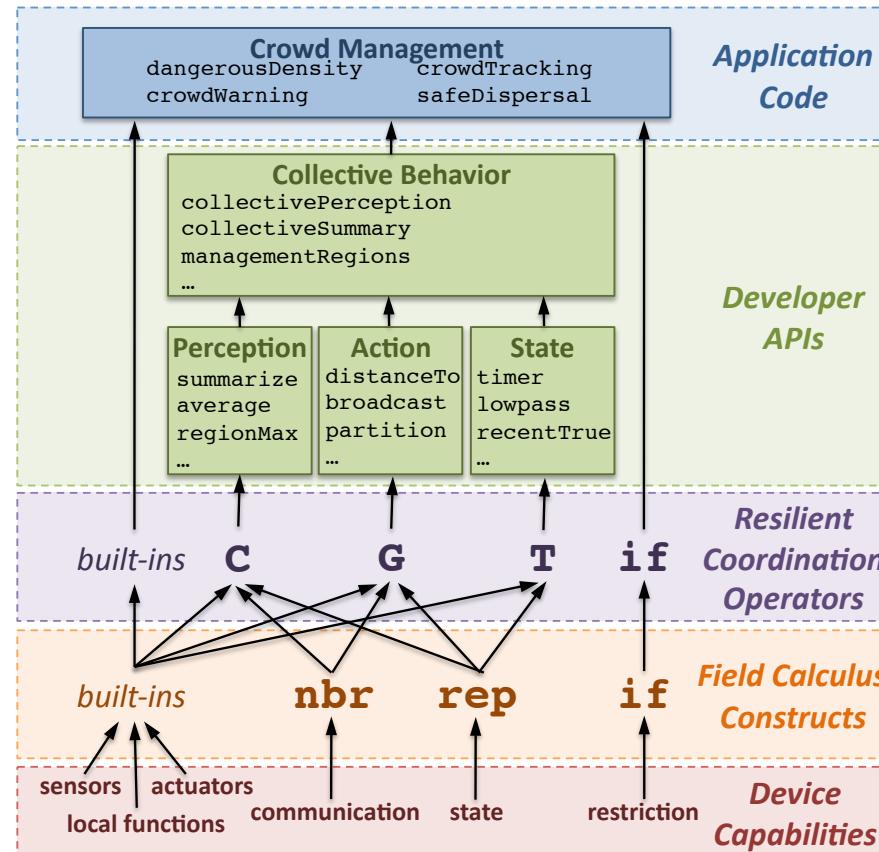
*Jacob Beal*

10<sup>th</sup> Swarm Intelligence Conf.  
September 2016



# A generative approach to safety

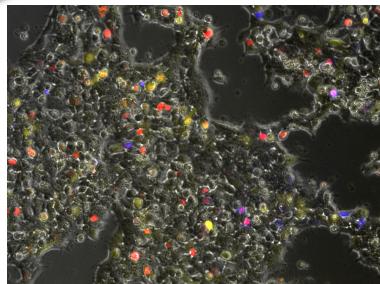
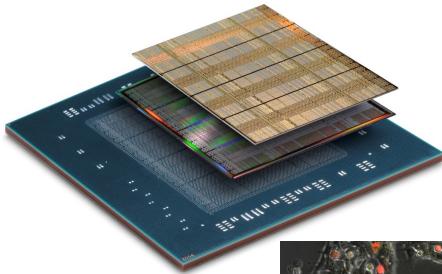
*Restrict your development environment...*



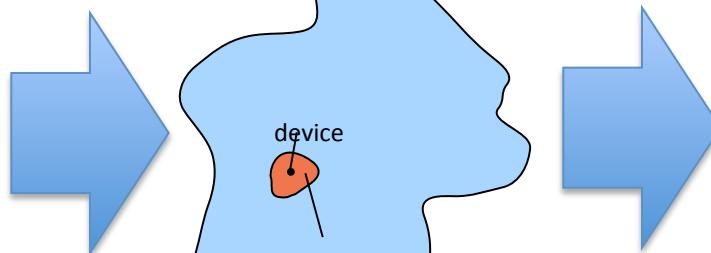
*... to contain only resilient distributed systems.*

# Dealing with challenging platforms

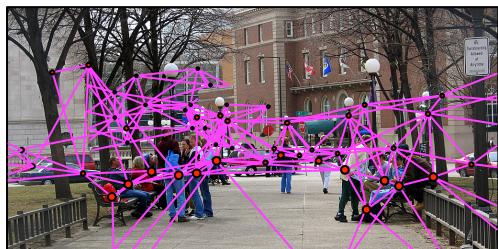
## *Emerging Computational Platforms*



## *Computational Field Programming Models*

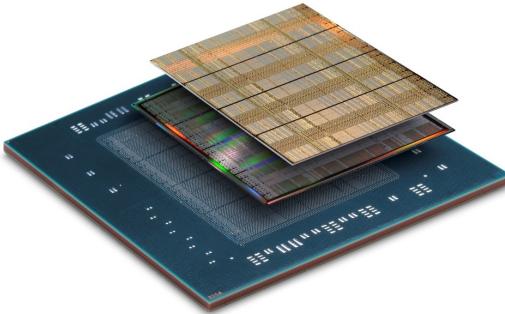


## *Inherently Resilient Distributed Systems*



*Pay a little efficiency, get a lot of programmability and resilience*

# Fundamentally different models



Isolate Systems  
Extremely High FLOPs



High Dispersion  
Moderate FLOPs



High Resolution Sense/Act  
Abysmal FLOPs

*How can we program aggregates adaptively & efficiently?  
Are there commonalities that cross substrates?*

# Example: Services for Mass Events

---

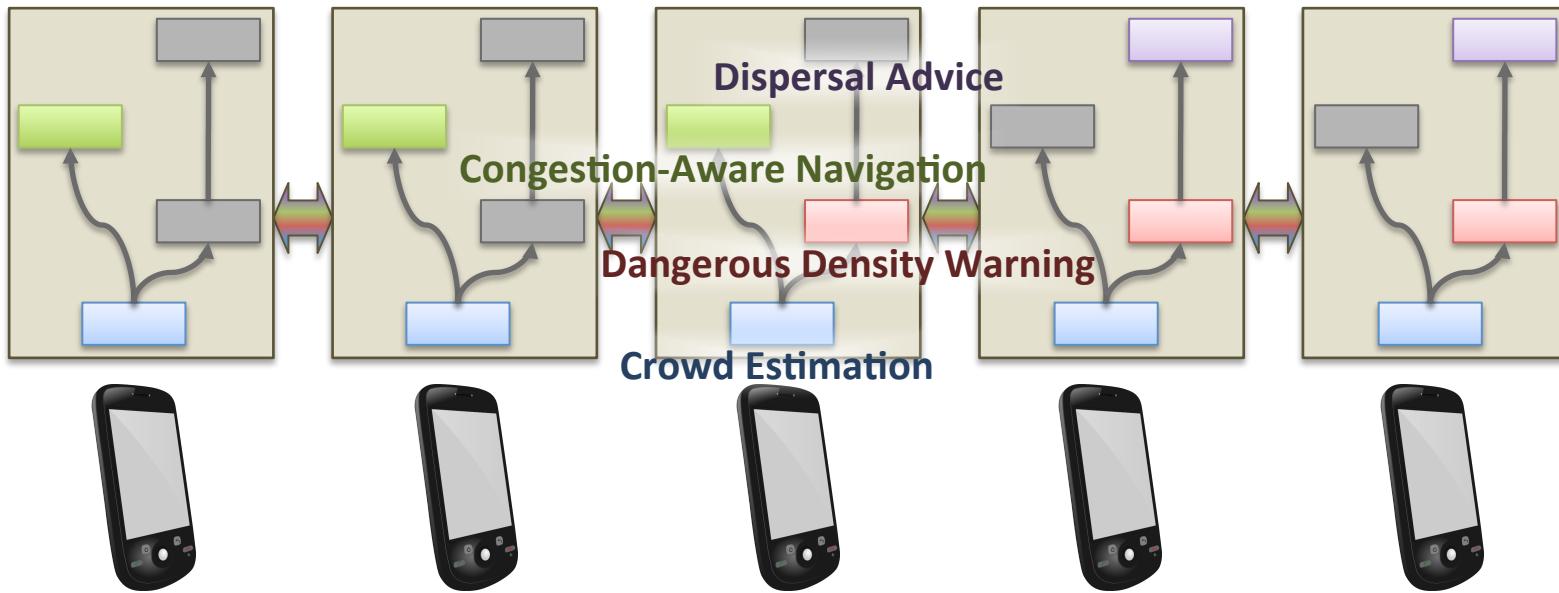


# Example: Managing Crowd Danger

---

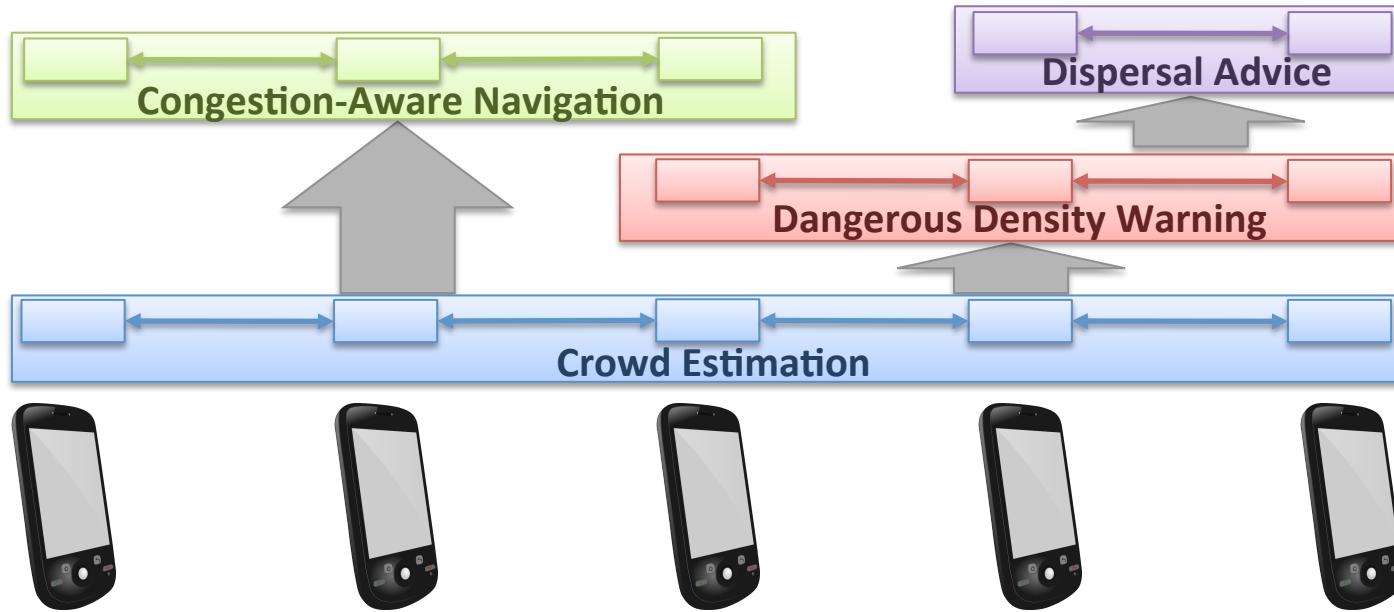


# Device-Centric Programming



- Explicit design of adaptation and communication
- Complex per-device multi-service application
- Intractable to ensure correct behavior

# Aggregate Programming

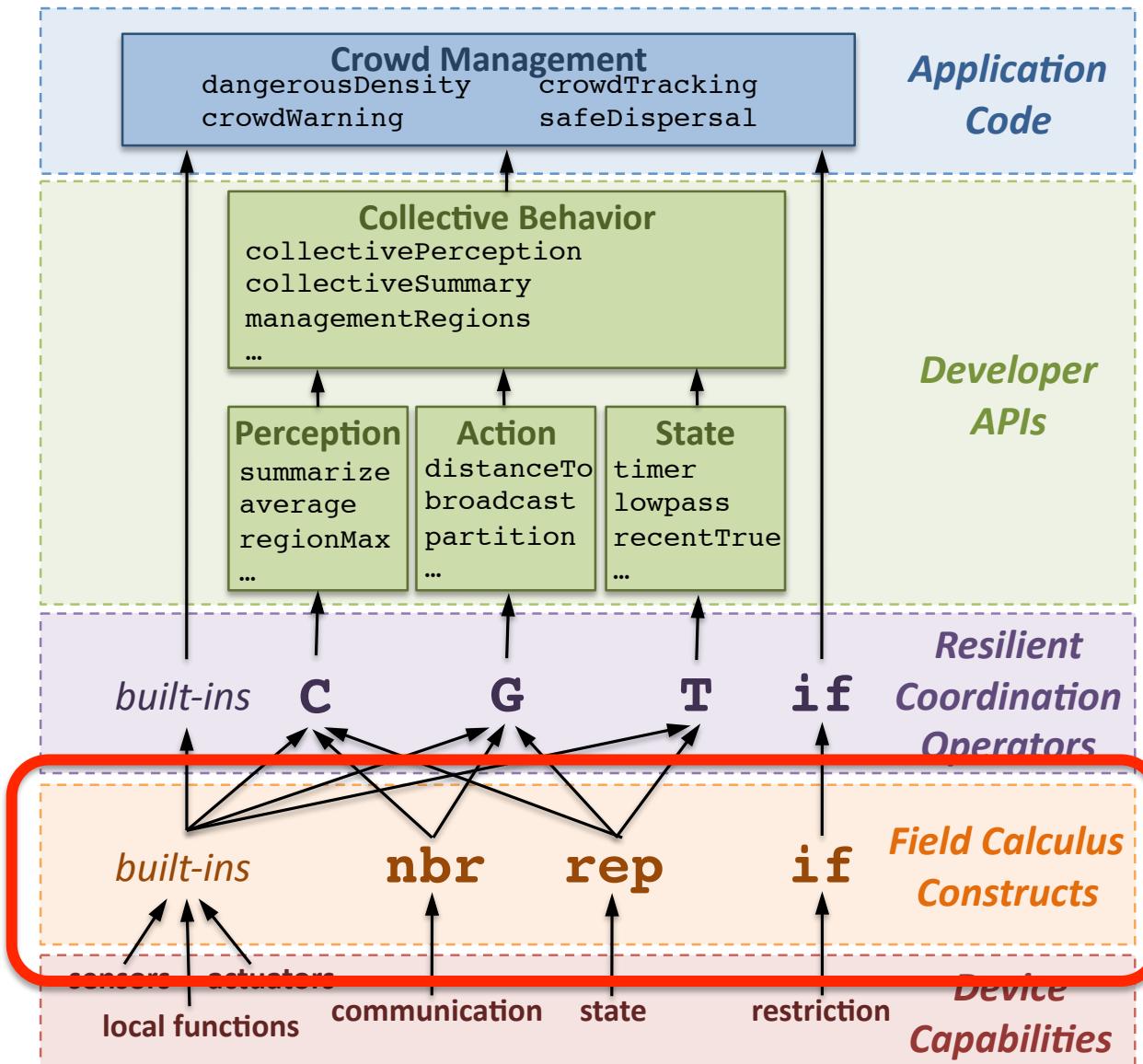


- Implicit adaptation and communication
- Code each collective service independently
- Compose via scope and information flow

# Aggregate Programming



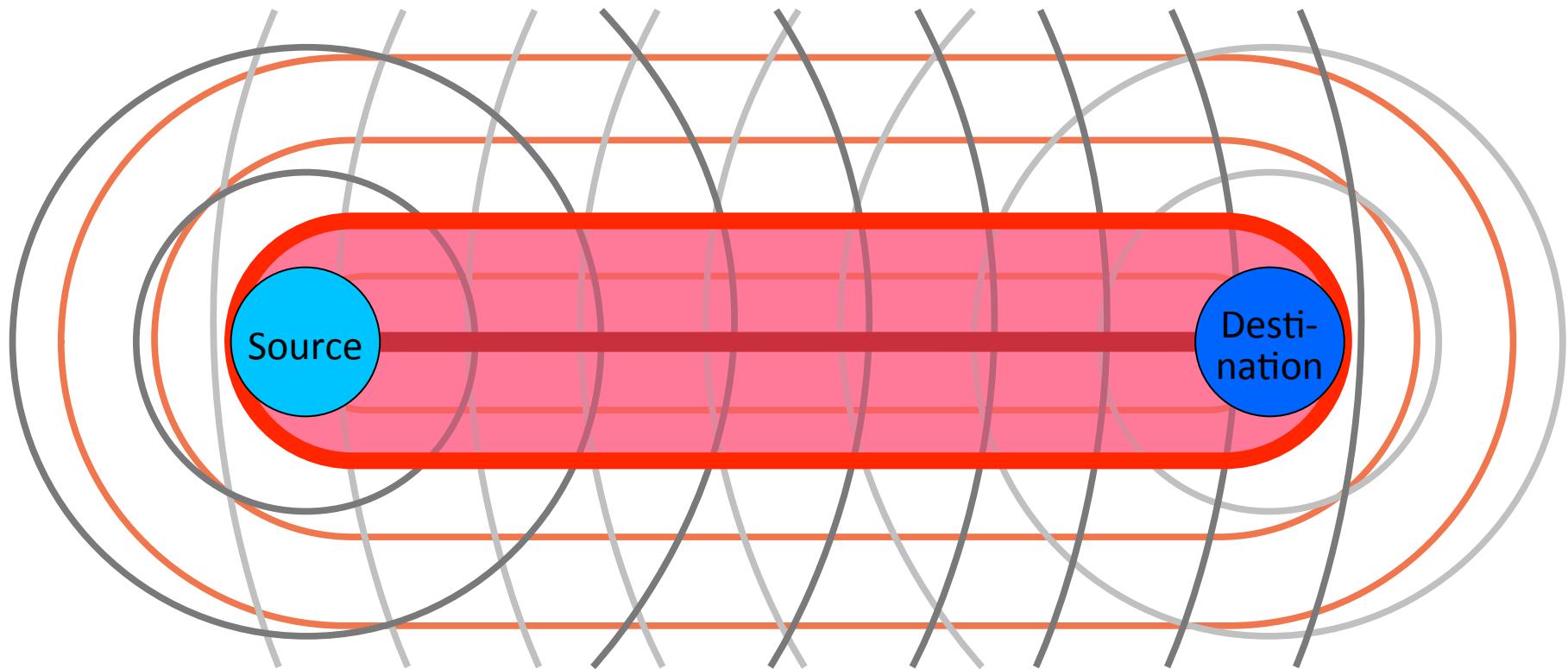
# Aggregate Programming Stack



# Example: Mesh-Network Cell Phones

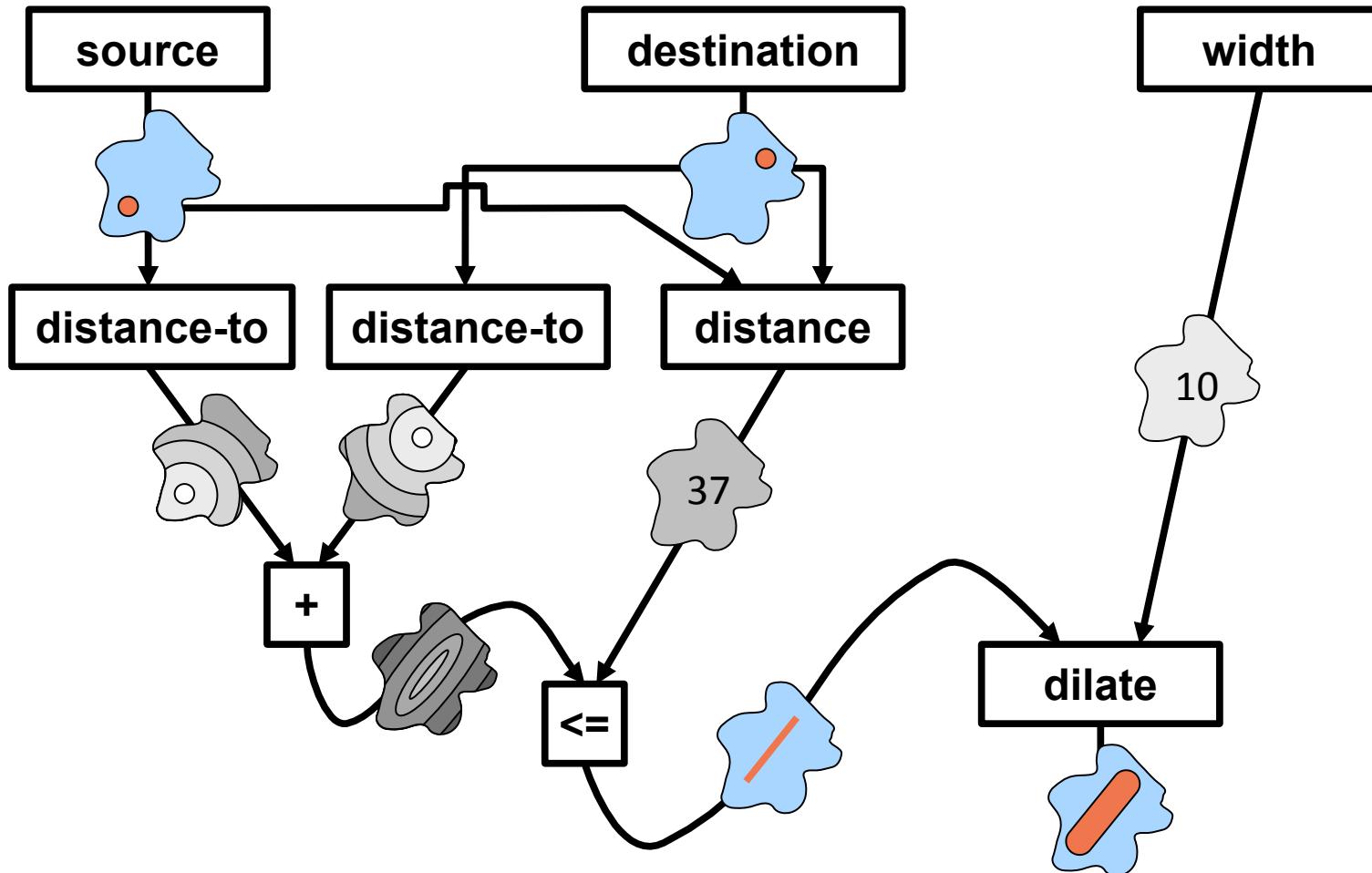


# Geometric Program: Channel

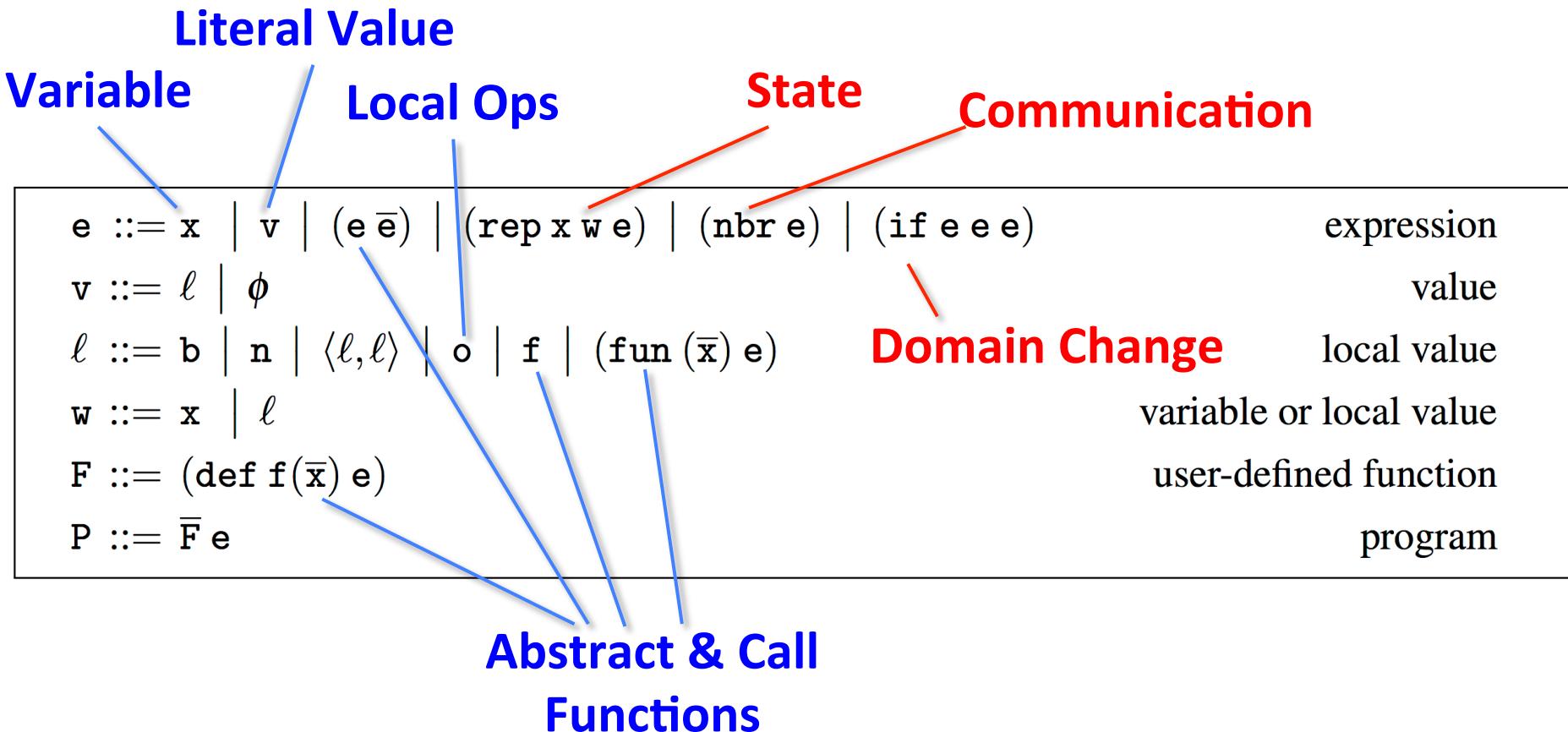


(cf. Butera)

# Computing with fields

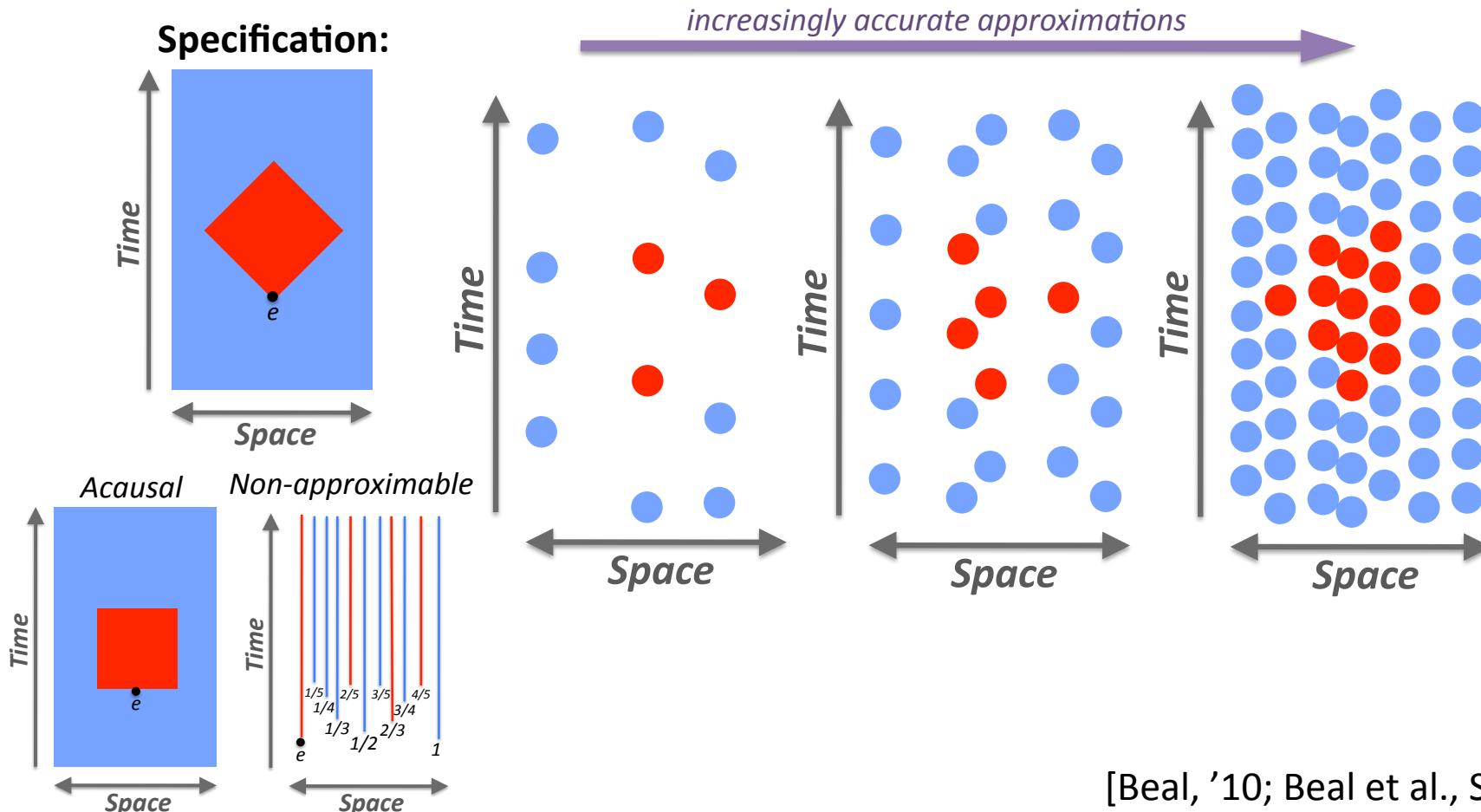


# (Higher Order) Field Calculus



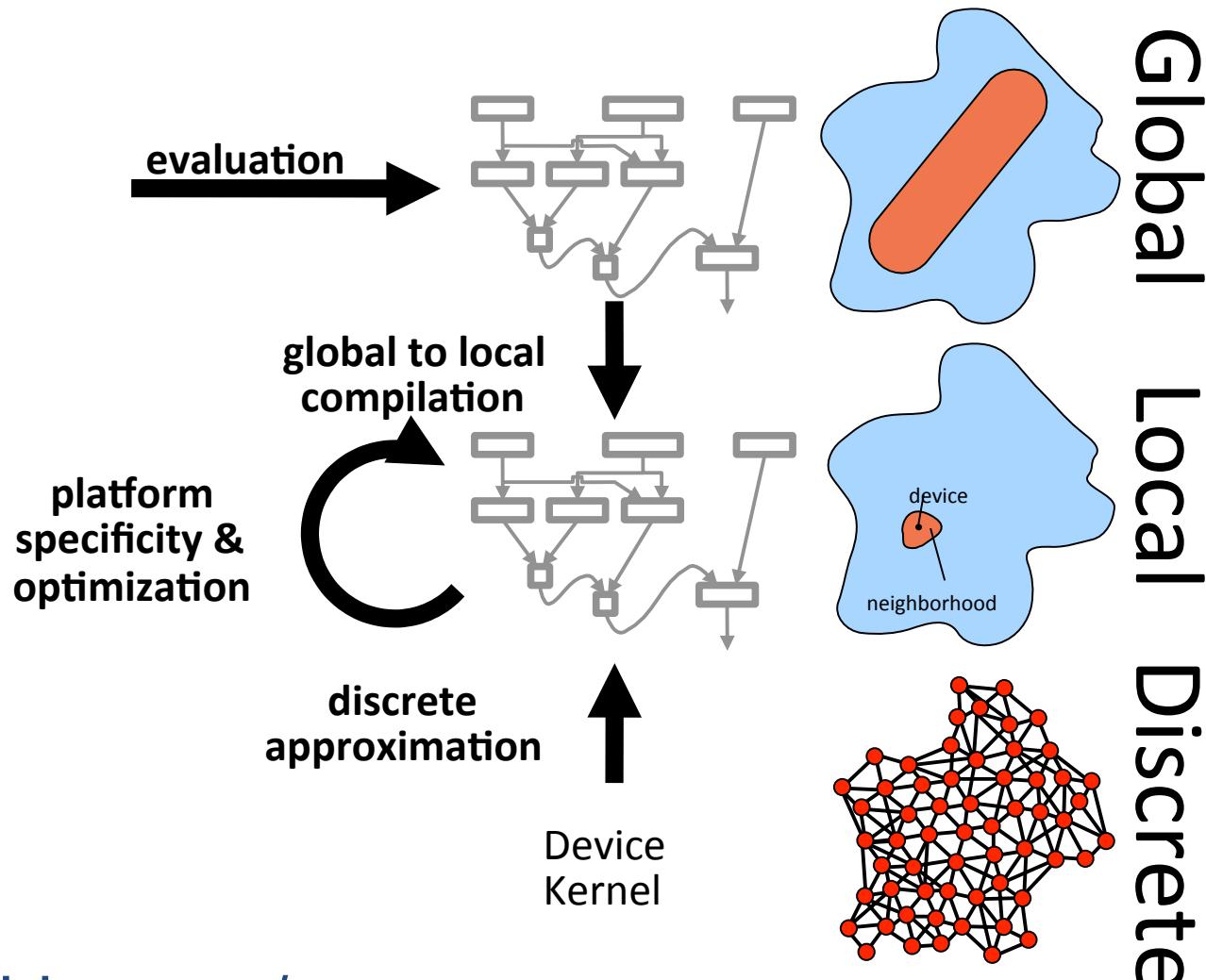
# Field Calculus is Space-Time Universal

*Space-time Universal = arbitrarily good approximation of any causal, finitely-approximable computation*



# Instantiation: 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)))
```



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

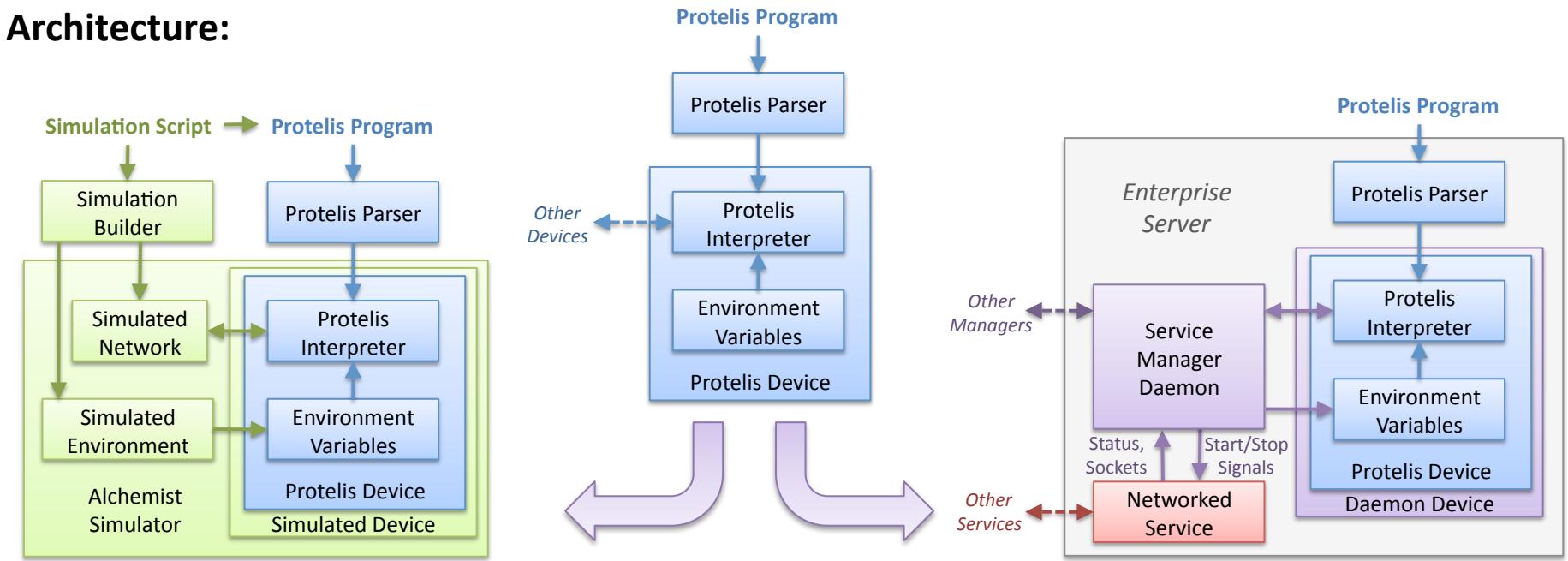
[Beal & Bachrach, '06]

# Instantiation: Protelis

- Java-hosted & integrated
- Java-like syntax
- Eclipse support

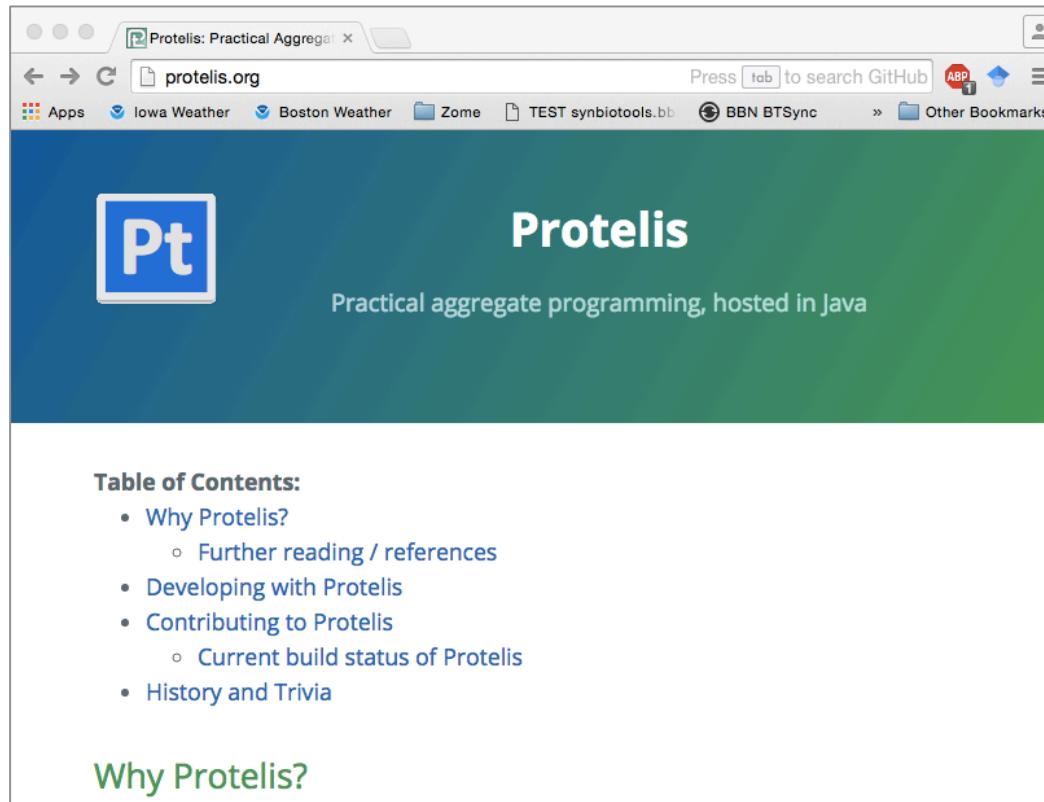
```
def distanceTo(source) {
    rep(d <- Infinity) {
        mux (source) { 0 }
        else { minHood(nbr{d} + nbrRange) }
    }
}
```

## Architecture:

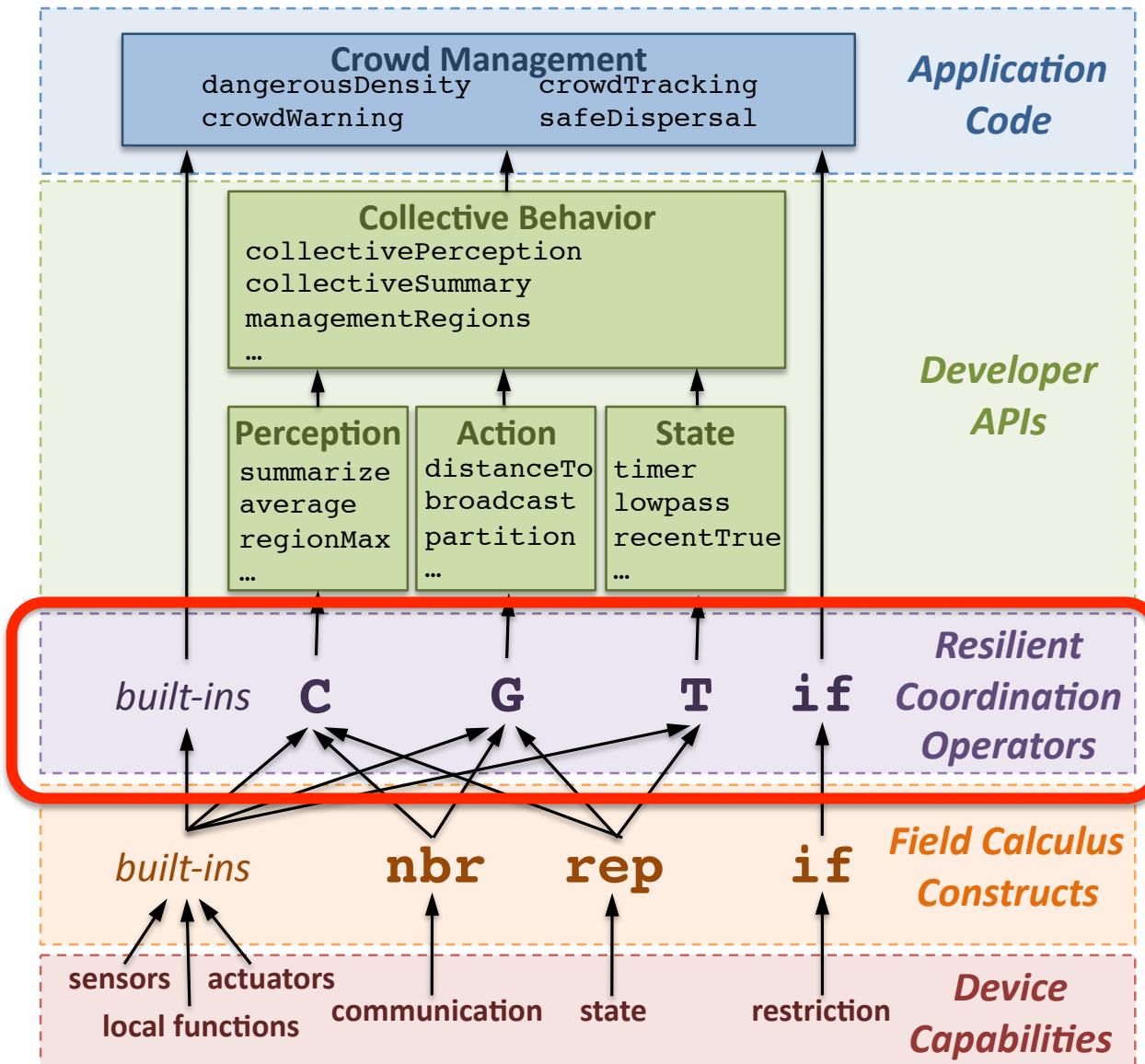


# Using Protelis in your projects

<http://protelis.org>



# Aggregate Programming Stack



# Example: Managing Crowd Danger

---

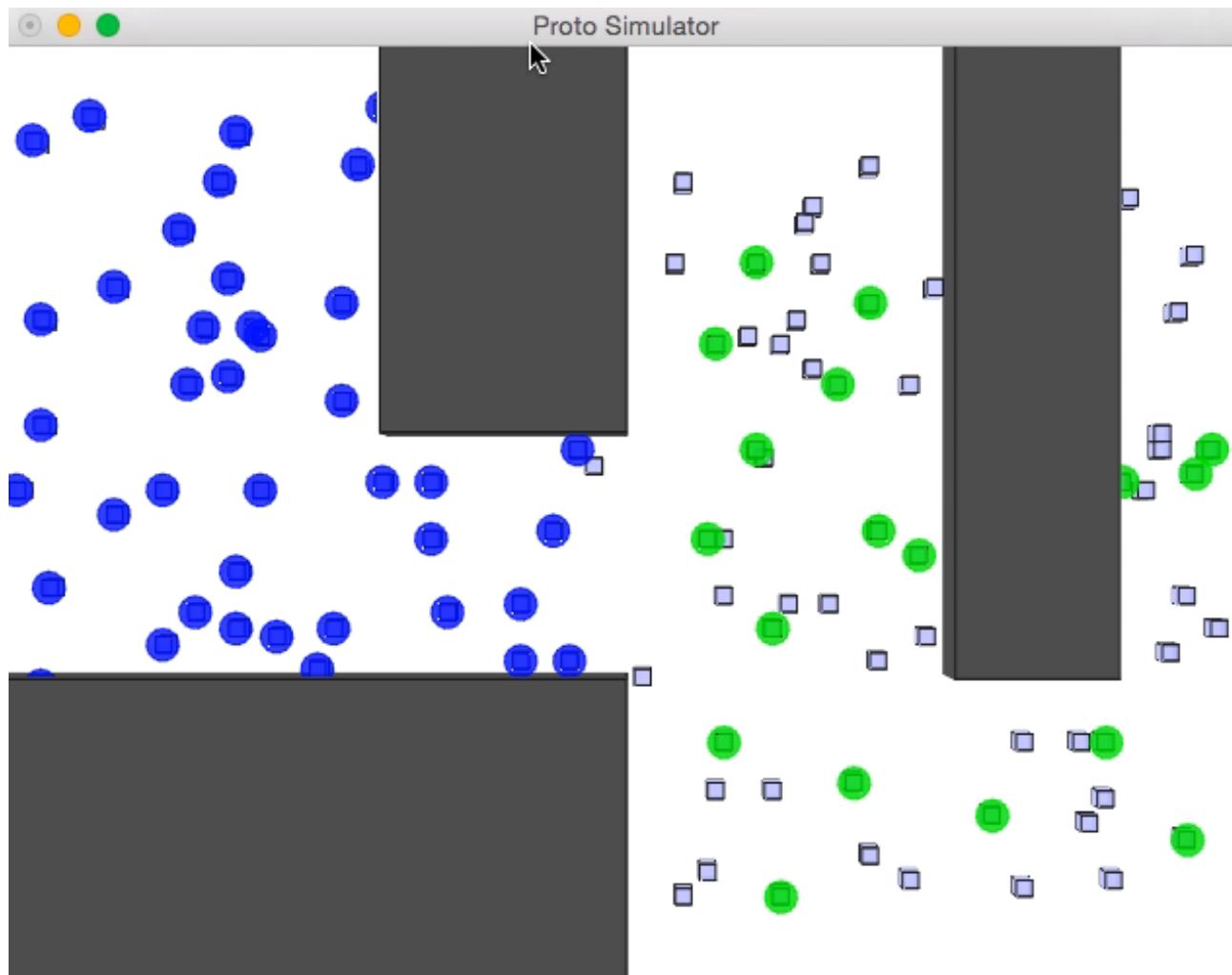


# Example of a complex service

---

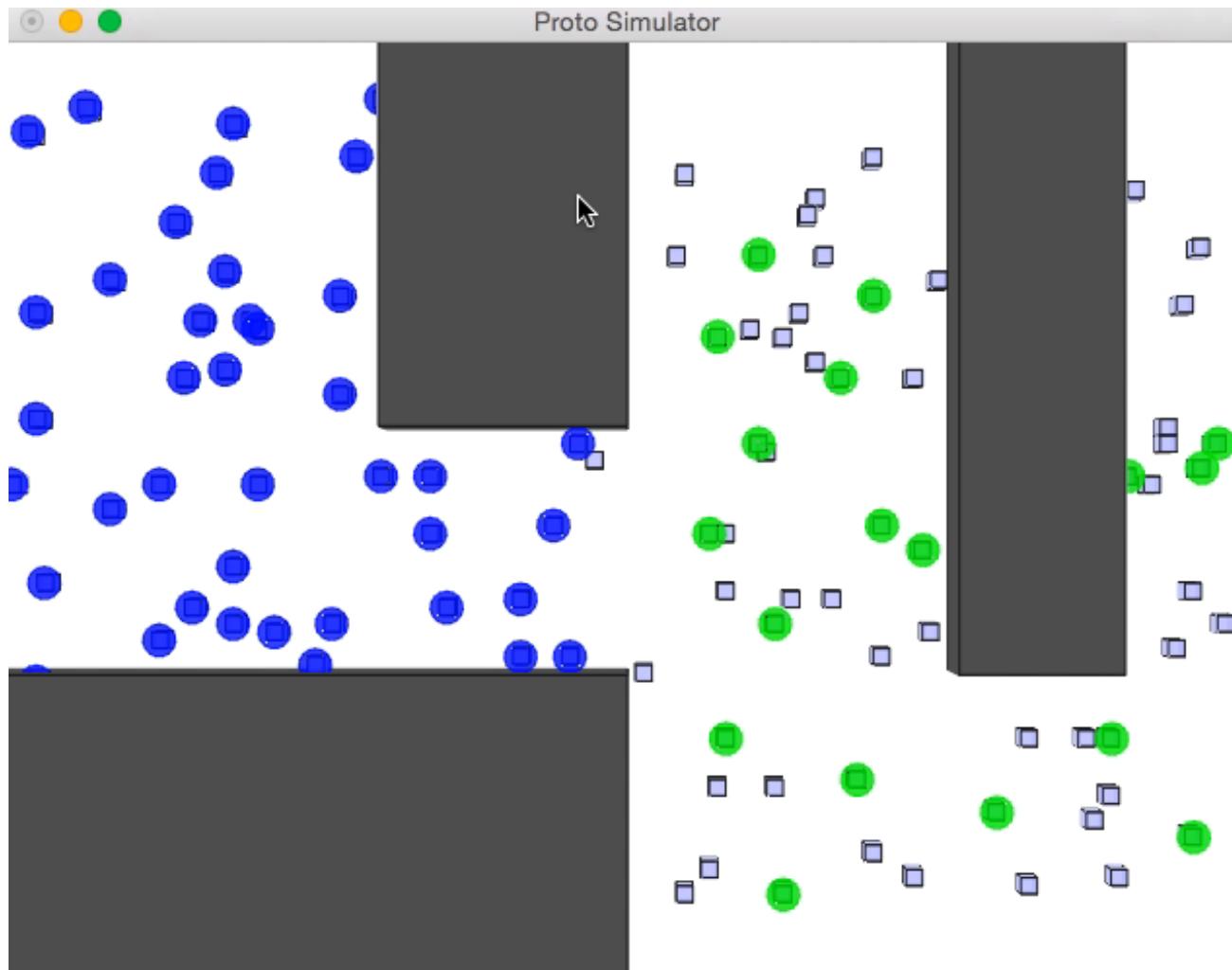
```
(def evacuate (zone coordinator alert)
  (let ((alerted
        (if zone
            (broadcast coordinator
              (collect-region
                (distance-to commander)
                alert)))
        0))))
  (* alerted
    (follow-gradient
      (distance-to (not zone))))))
```

# Self-stabilization is hard to get right



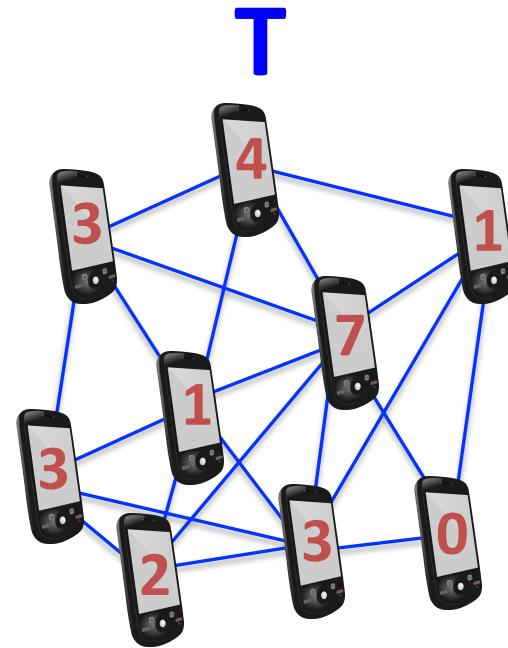
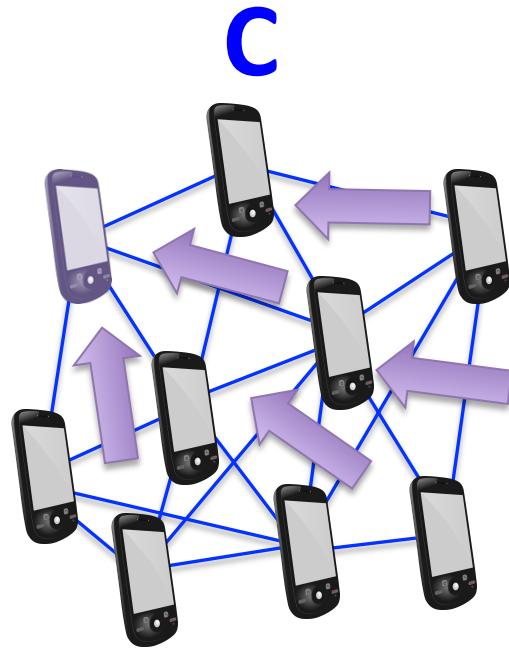
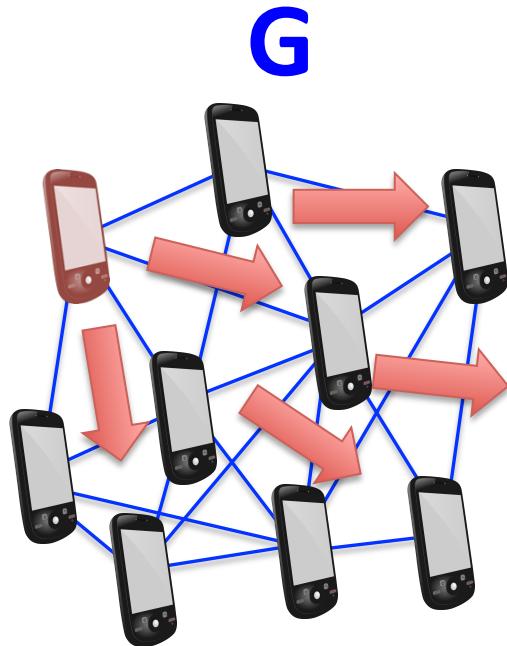
*Naïve geometry: when stationary, fine...*

# Self-stabilization is hard to get right



*... but doesn't correct properly for change.*

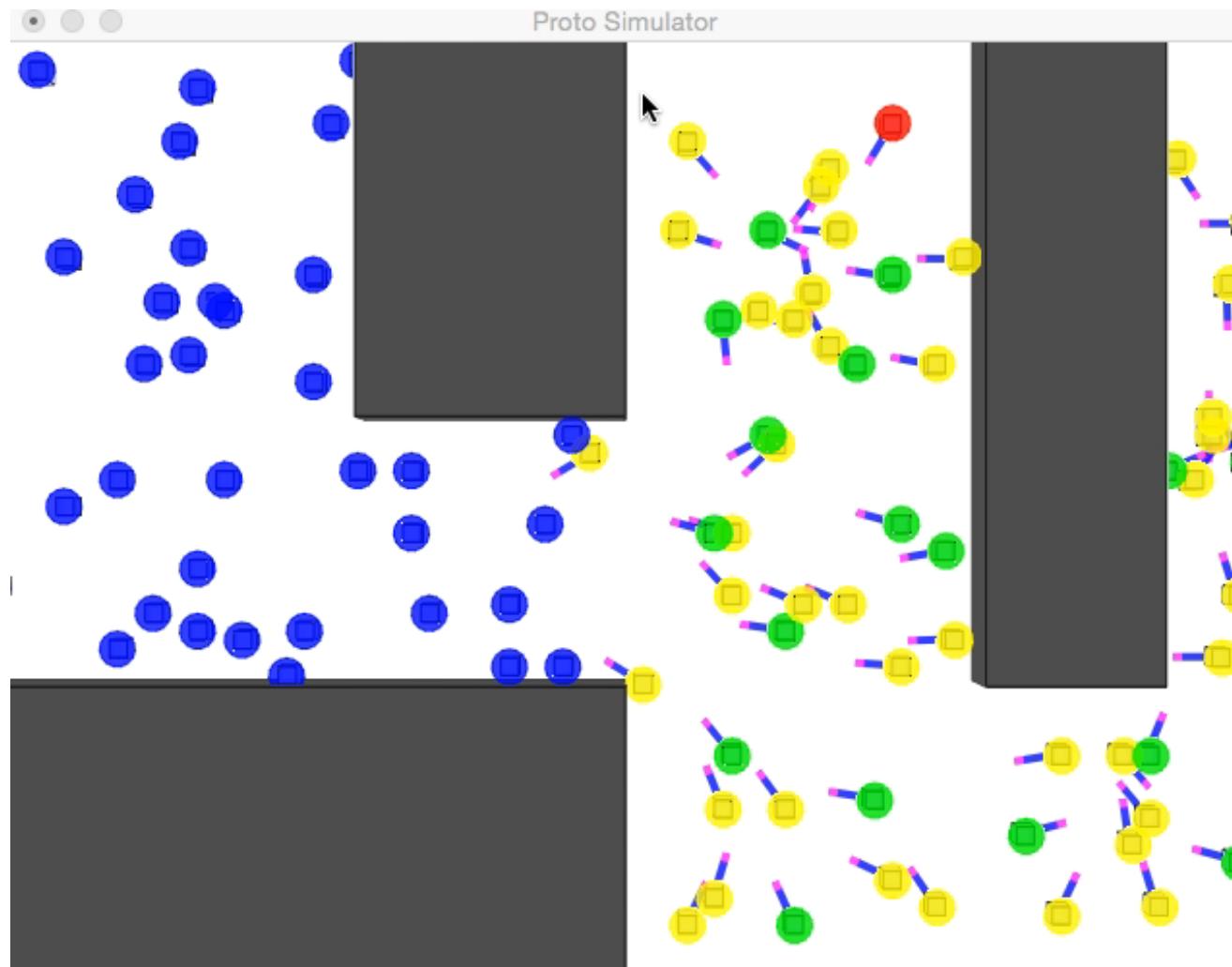
# Self-Stabilizing Building Blocks



*Information spreading    Information collection    Short-term memory*

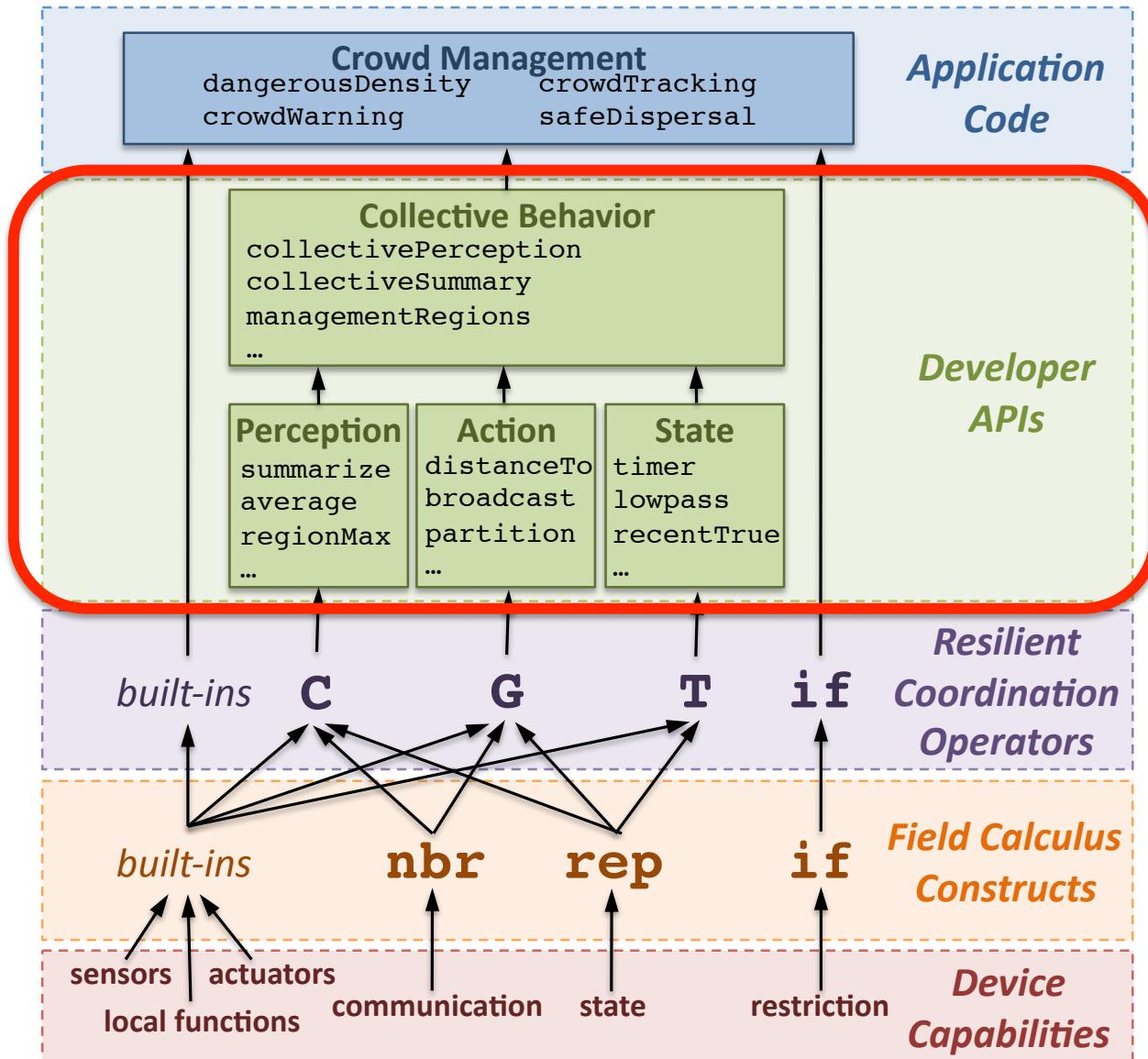
Resilience by construction: all programs from  
these building blocks are also self-stabilizing!

# All combinations are self-stabilizing!



*Now program rapidly converges following changes*

# Aggregate Programming Stack



# Applying building blocks:

---

## Example API algorithms from building blocks:

|  |                                 |
|--|---------------------------------|
| distance-to (source)                           | max-likelihood (source p)       |
| broadcast (source value)                       | path-forecast (source obstacle) |
| summarize (sink accumulate local null)         | average (sink value)            |
| integral (sink value)                          | region-max (sink value)         |
| timer (length)                                 | limited-memory (value timeout)  |
| random-voronoi (grain metric)                  | group-size (region)             |
| broadcast-region (region source value)         | recent-event (event timeout)    |
| distance-avoiding-obstacles (source obstacles) |                                 |

*Since based on these building blocks, all programs built this way are self-stabilizing!*

# Complex Example: Crowd Management

```
(def crowd-tracking (p)
  ;; Consider only Fruin LoS E or F within last minute
  (if (recently-true (> (density-est p) 1.08) 60)
    ;; Break into randomized "cells" and estimate danger of each
    (+ 1 (dangerous-density (sparse-partition 30) p))
    0))

(def recently-true (state memory-time)
  ;; Make sure first state is false, not true...
  (rt-sub (not (T 1 1)) state memory-time))
(def rt-sub (started s m)
  (if state 1 (limited-memory s m)))

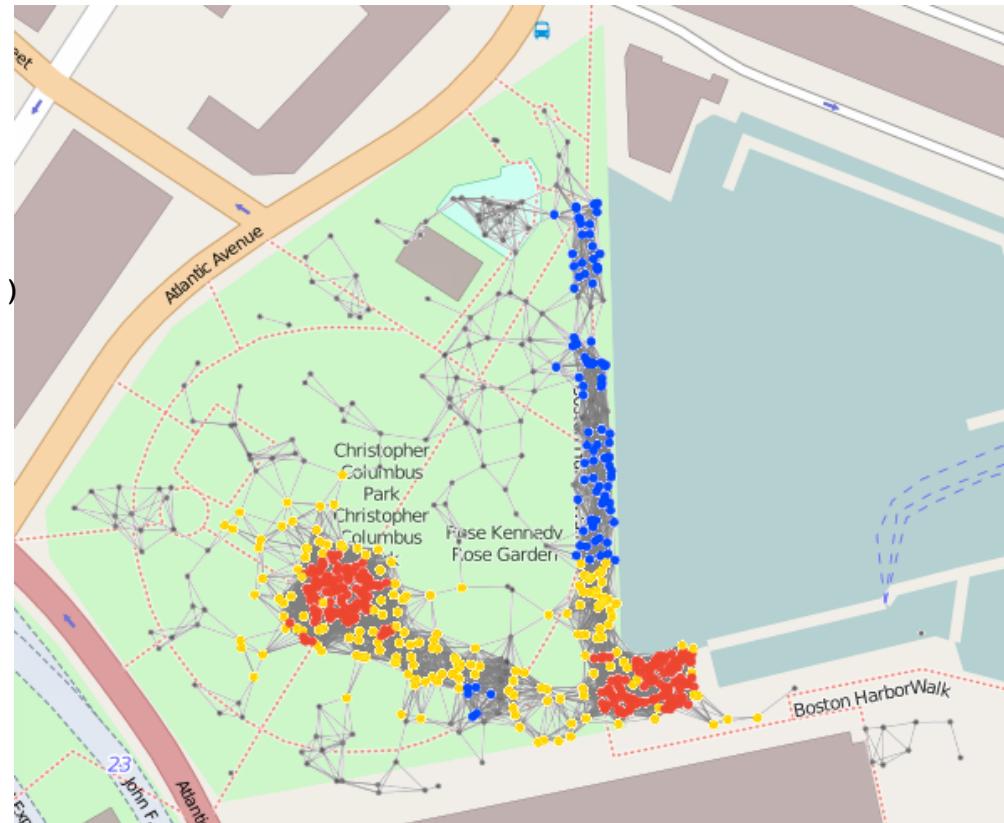
(def dangerous-density (partition p)
  ;; Only dangerous if above critical density threshold...
  (and
    (> (average partition (density-est p)) 2.17)
    ;; ... and also involving many people.
    (> (summarize partition + (/ 1 p) 0) 300)))
```

*18 lines non-whitespace code  
10 library calls (21 ops)*

*IF: 3 G: 11 C: 4 T: 3*

```
(def crowd-warning (p range)
  (> (distance-to (= (crowd-tracking p) 2))
      range))

(def safe-navigation (destination p)
  (distance-avoiding-obstacles
    destination (crowd-warning p)))
```



# Generalization: Self-Stabilizing Calculus

|   |                         |
|---|-------------------------|
| $e ::= x \mid v \mid (e \bar{e}) \mid (\text{rep } x \ w \ e) \mid (\text{nbr } e) \mid (\text{if } e \ e \ e)$ | expression              |
| $v ::= \ell \mid \phi$  | value                   |
| $\ell ::= b \mid n \mid \langle \ell, \ell \rangle \mid o \mid f \mid (\text{fun } (\bar{x}) \ e)$              | local value             |
| $w ::= x \mid \ell$   | variable or local value |
| $F ::= (\text{def } f(\bar{x}) \ e)$  | user-defined function   |
| $P ::= \bar{F} \ e$   | program                 |

*Restrict field calculus by replacing  $e$  with  $s$ :*

|  |   |
|--|---|
| $s ::= \ell \mid x \mid (s \bar{s}) \mid (\text{nbr } s) \mid (\text{if } s \ s \ s)$                      |   |
| $  \quad \mathbf{T} (\text{rep } x \ w (\pi^{\text{MB}} x \ \bar{s}))$                                     | $x \notin \mathbf{FV}(\bar{s})$   |
| $  \quad \mathbf{C} (\text{rep } x \ w (\pi^F s^A (\text{nbr } (s \ x)) \bar{s}))$                         | $x \notin \mathbf{FV}(s, \bar{s}, s^A)$   |
| $  \quad \mathbf{G} (\text{rep } x \ w (\pi (\pi' (\text{nbr } (\pi'' x \ \bar{s}'')) \bar{s}') \bar{s}))$ | $\pi' \circ \pi = \pi^{\text{MD}}, \pi'' \circ \pi' = \pi^{\text{MBP}}, x \notin \mathbf{FV}(\bar{s}, \bar{s}', \bar{s}'')$ |

# Self-Stabilization → Substitution

Self-Stabilising  
Calculus



(Higher-Order)  
Field Calculus



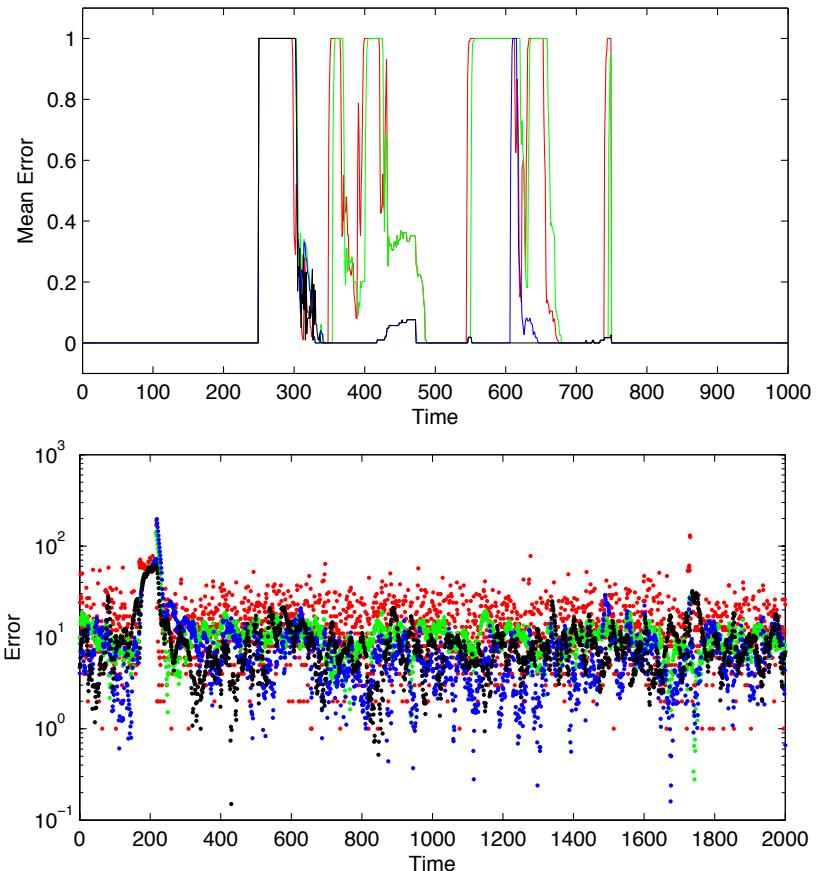
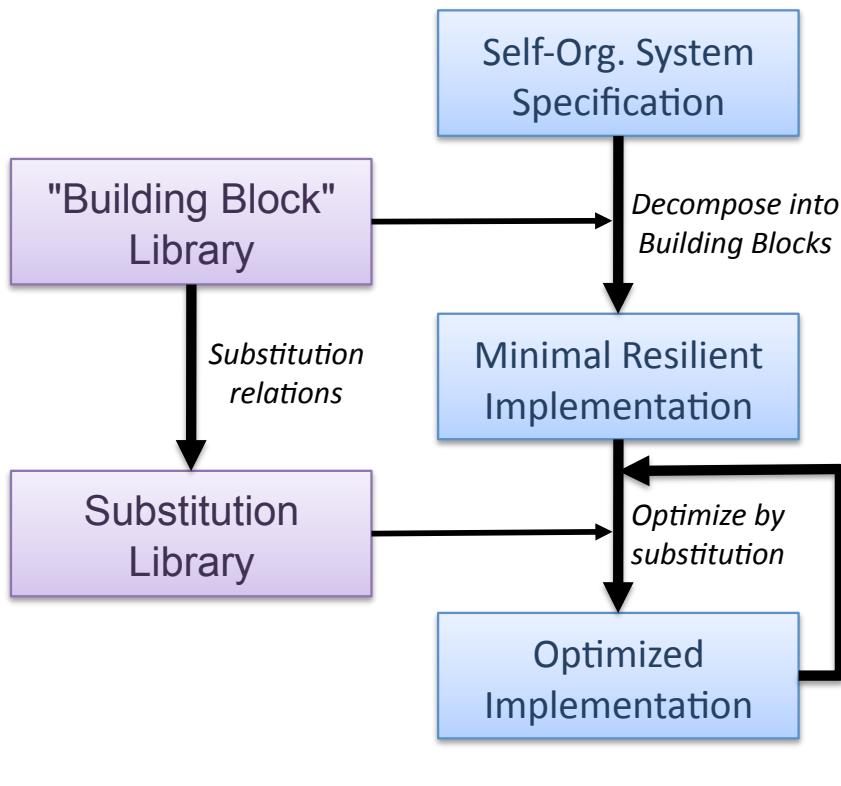
*Building Block  
Operator*



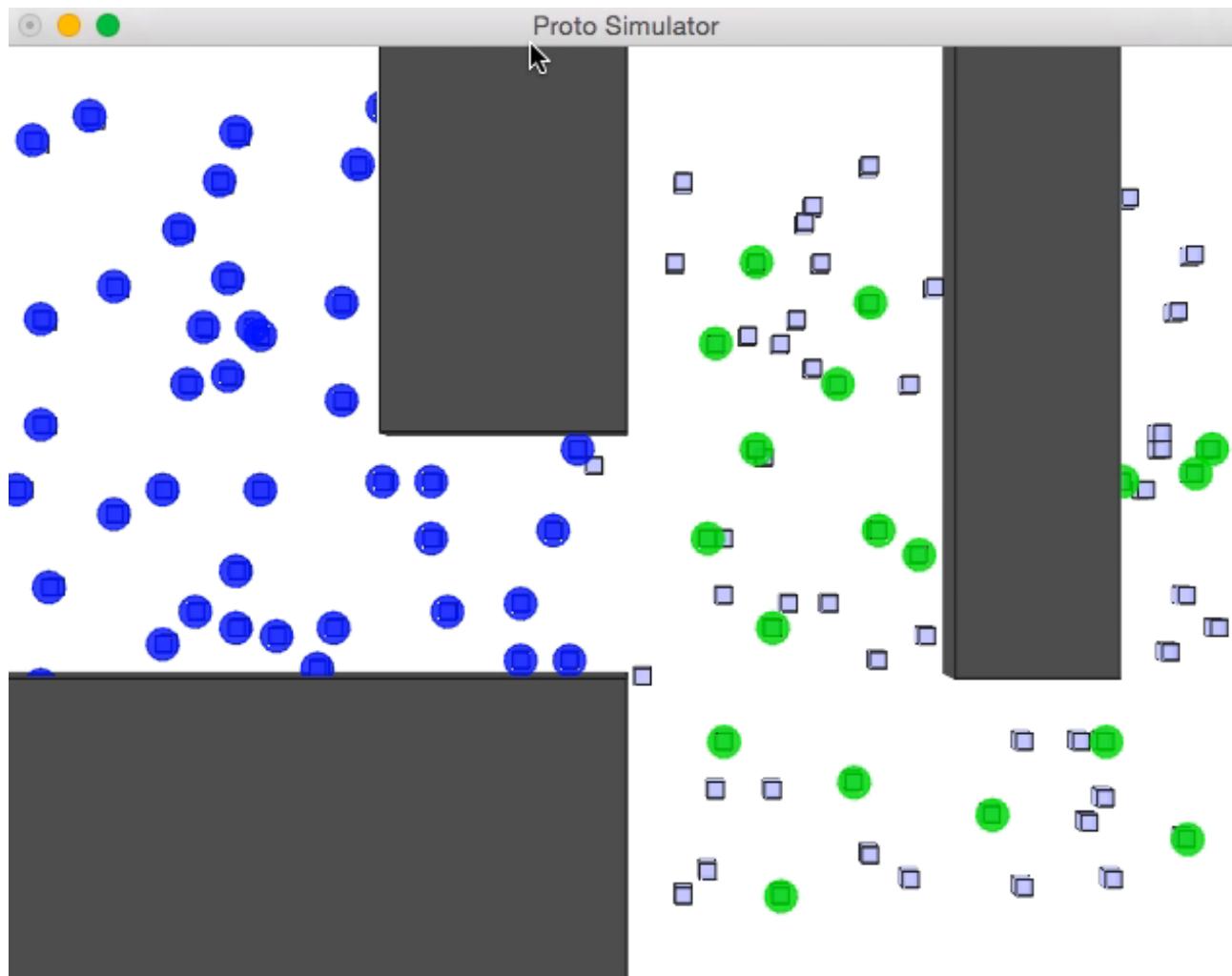
*Coordination  
Mechanism*

*Given functions  $\lambda, \lambda'$  with same type,  $\lambda$  is **substitutable** for  $\lambda'$  iff for any self-stabilising list of expressions  $e$ ,  $(\lambda e)$  always self-stabilises to the same value as  $(\lambda' e)$ .*

# Optimization of Dynamics

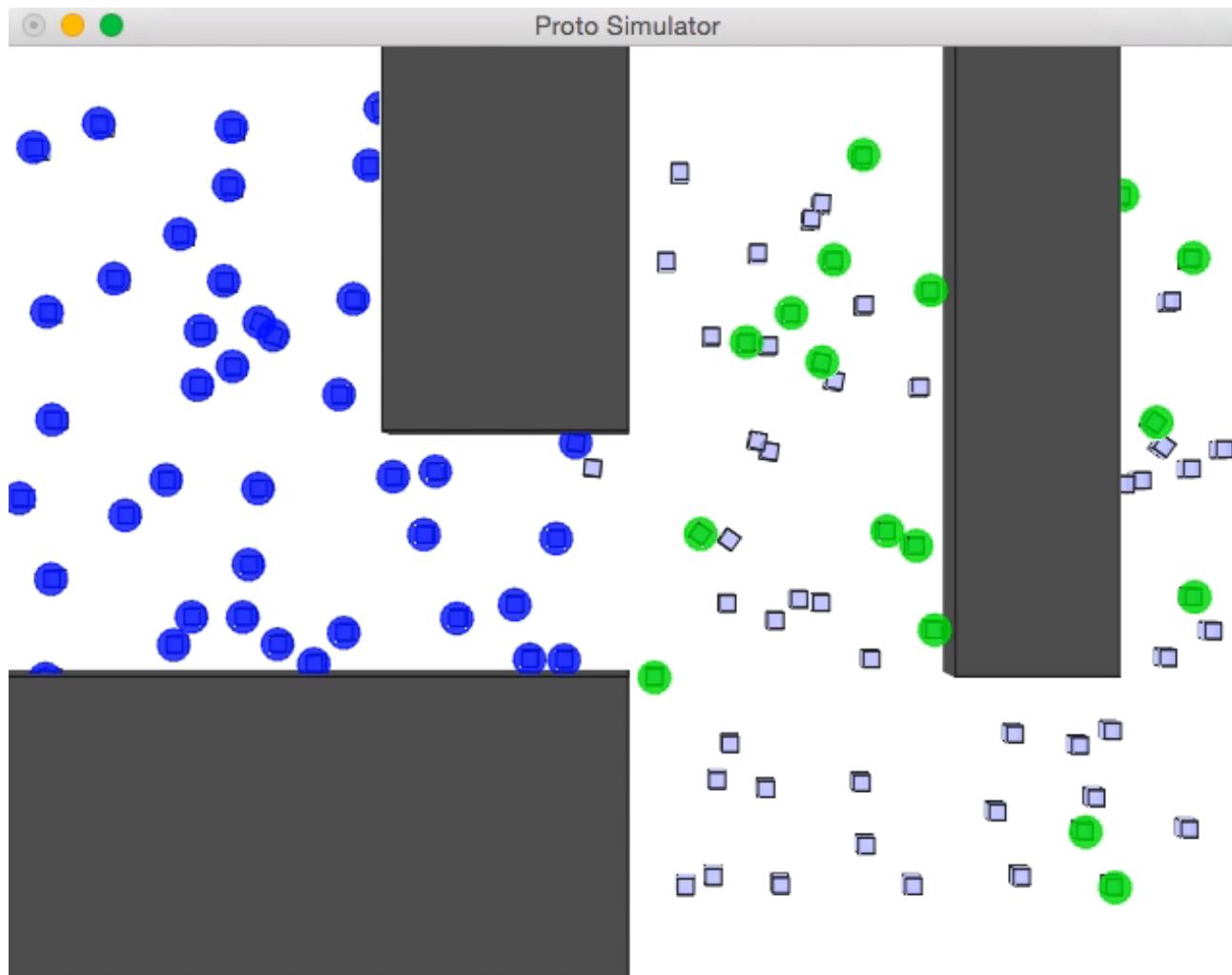


# Optimization Example: Crowd Alert



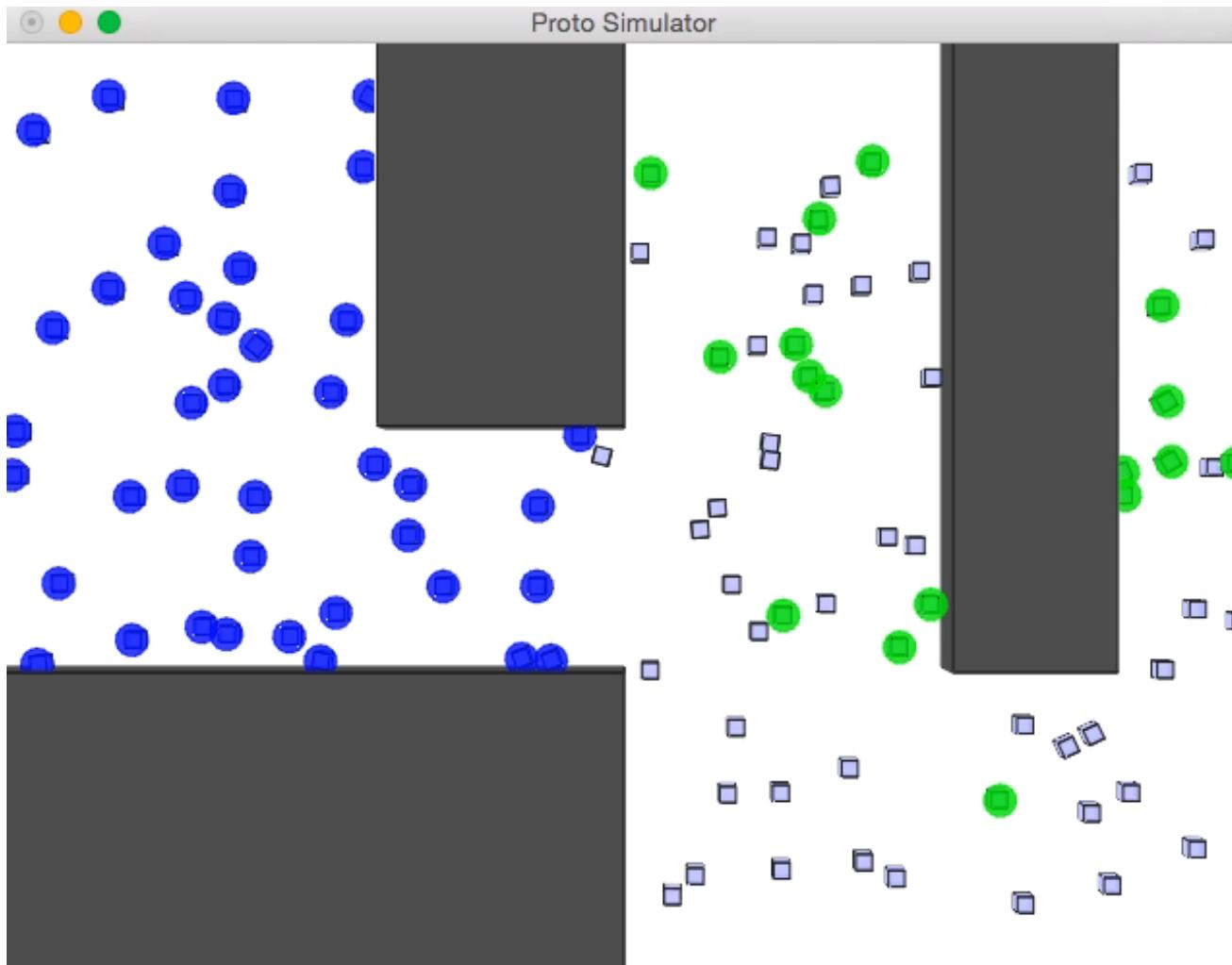
*Naïve algorithm: when stationary, fine...*

# Optimization Example: Crowd Alert



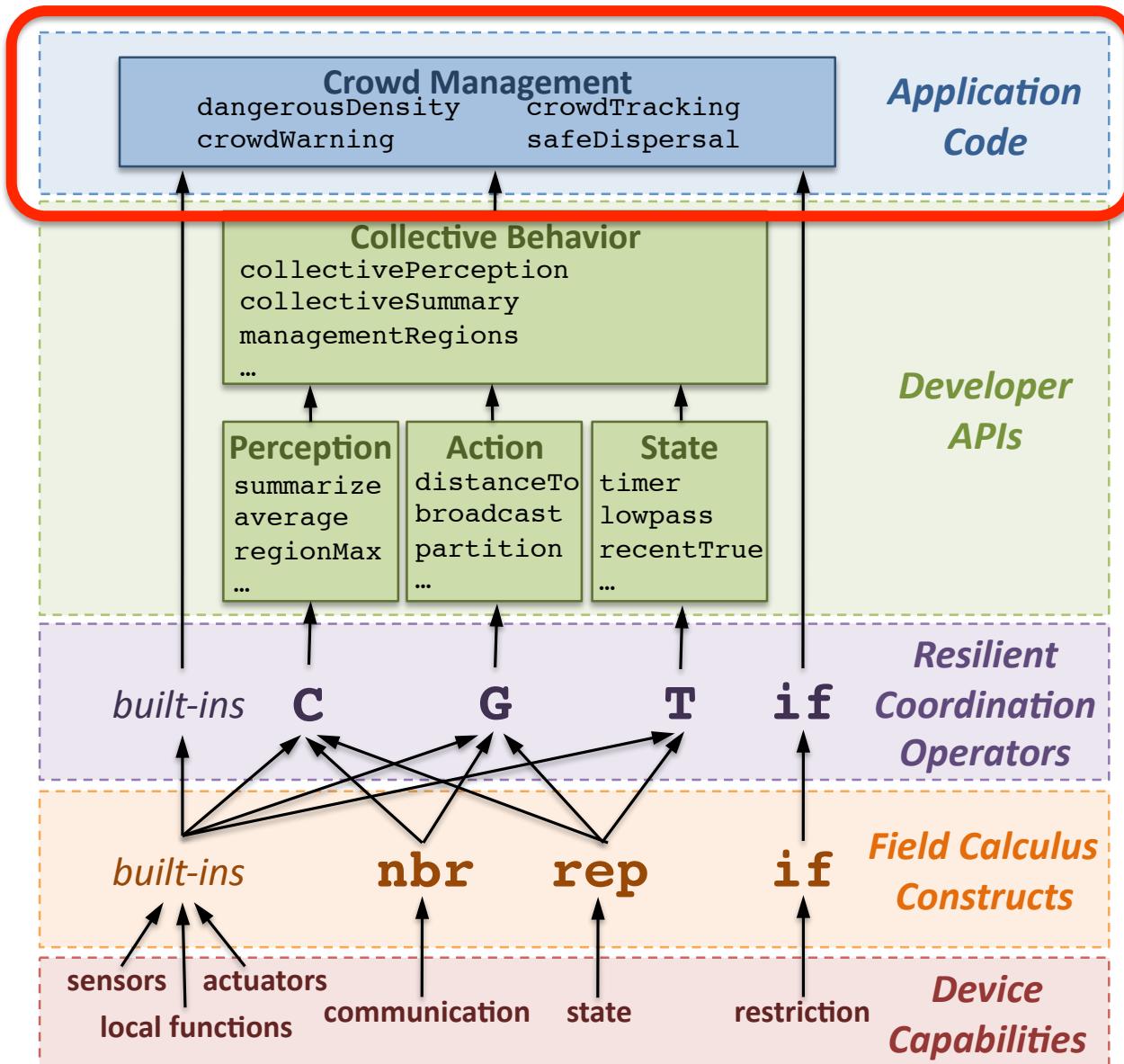
*... but dynamics can't keep up with fast mobility.*

# Optimization Example: Crowd Alert

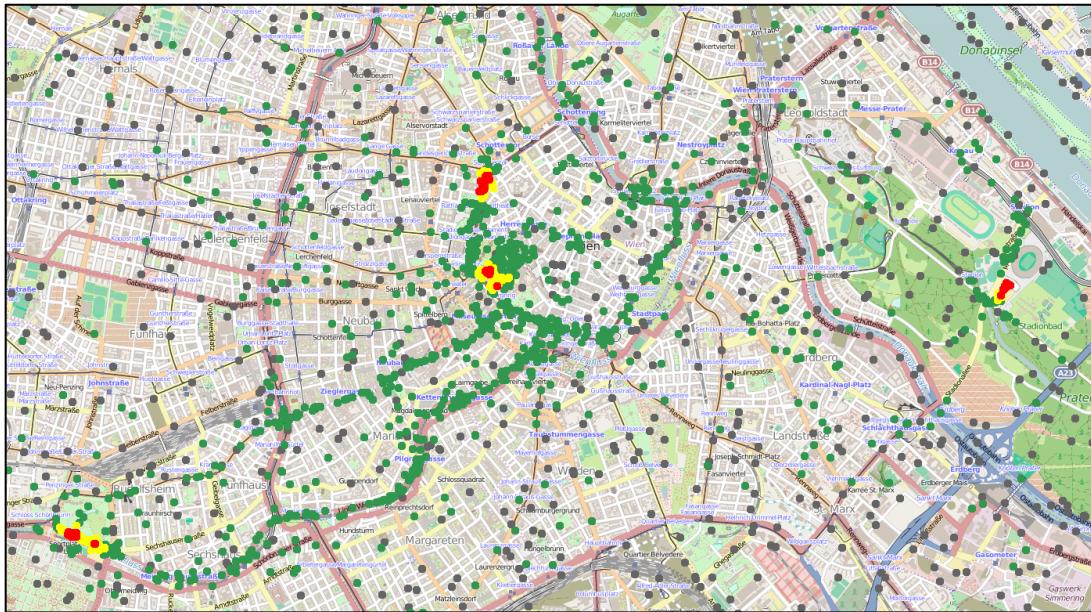


*Optimized dynamics, however, work well.*

# Aggregate Programming Stack



# Crowd Safety Services

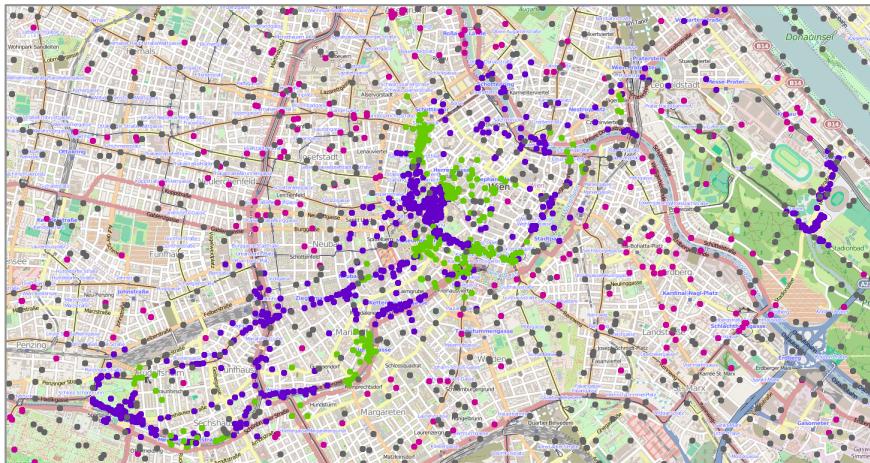


```

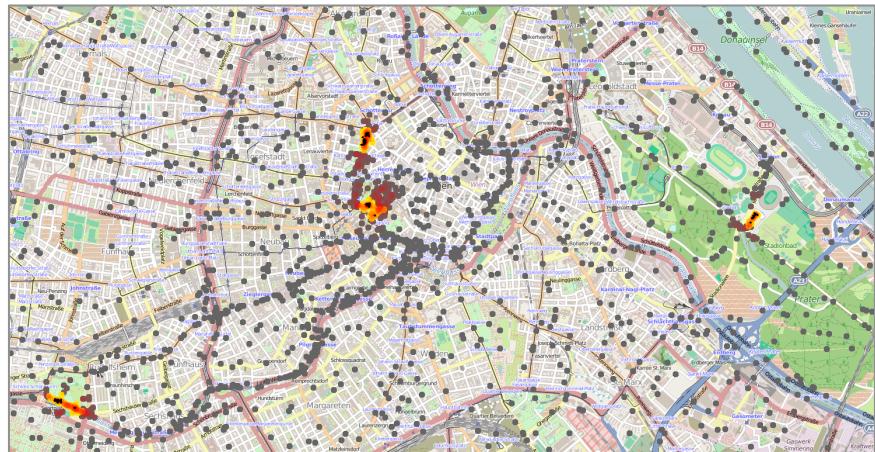
def dangerousDensity(p, r) {
    let mr = managementRegions(r*2, () -> { nbrRange });
    let danger = average(mr, densityEst(p, r)) > 2.17 &&
        summarize(mr, sum, 1 / p, 0) > 300;
    if(danger) { high } else { low }
}
def crowdTracking(p, r, t) {
    let crowdRgn = recentTrue(densityEst(p, r)>1.08, t);
    if(crowdRgn) { dangerousDensity(p, r) } else { none };
}
def crowdWarning(p, r, warn, t) {
    distanceTo(crowdTracking(p,r,t) == high) < warn
}

```

Dissemination of new versions

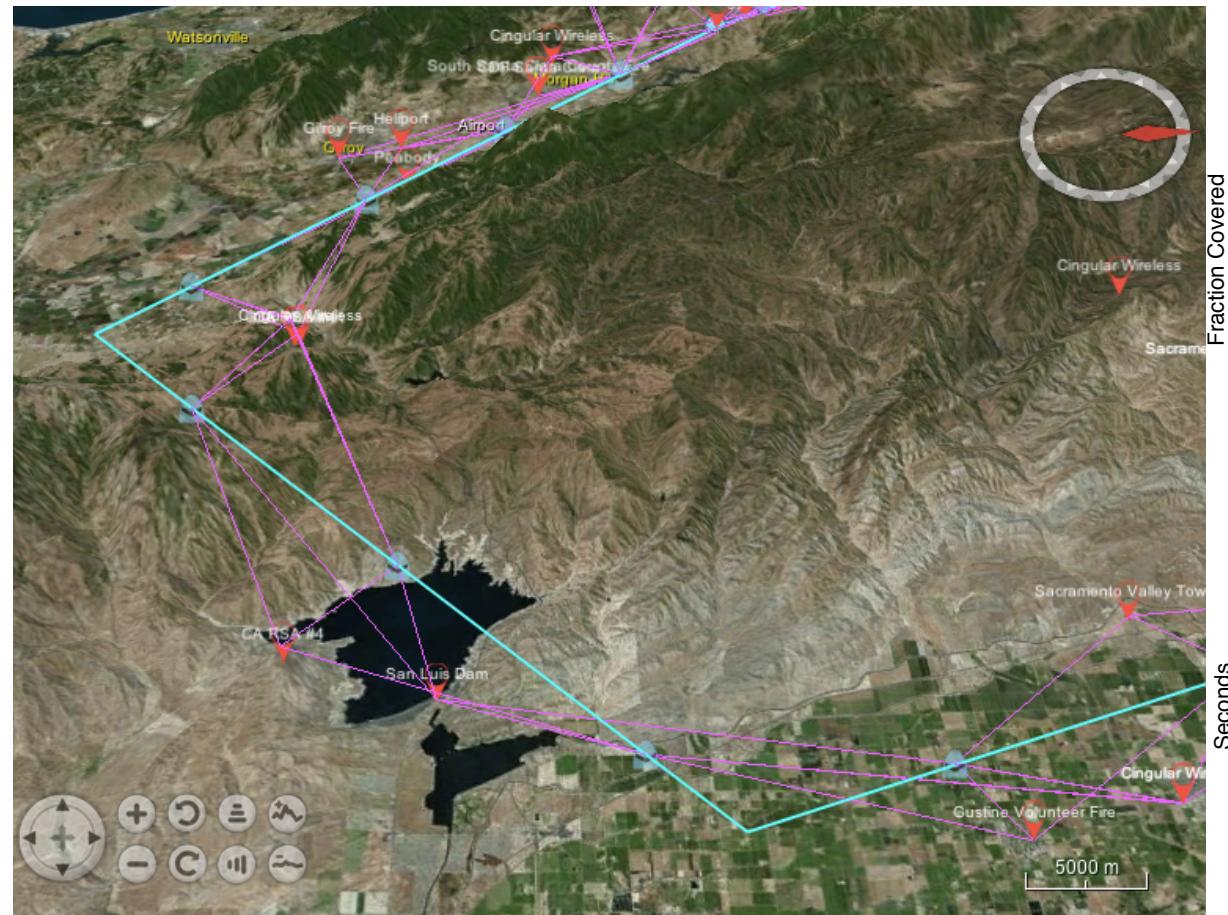


Pre-emptive modulation of priorities

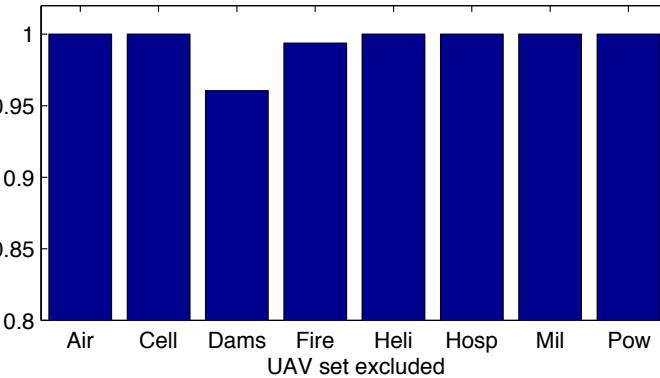


# Opportunistic Airborne Sensor Sharing

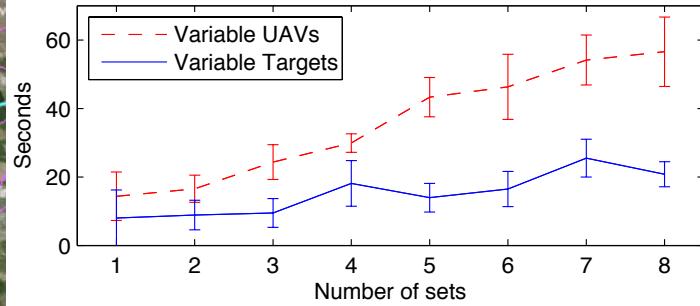
## GIS-integrated adaptive mission planning



## Highly effective sharing

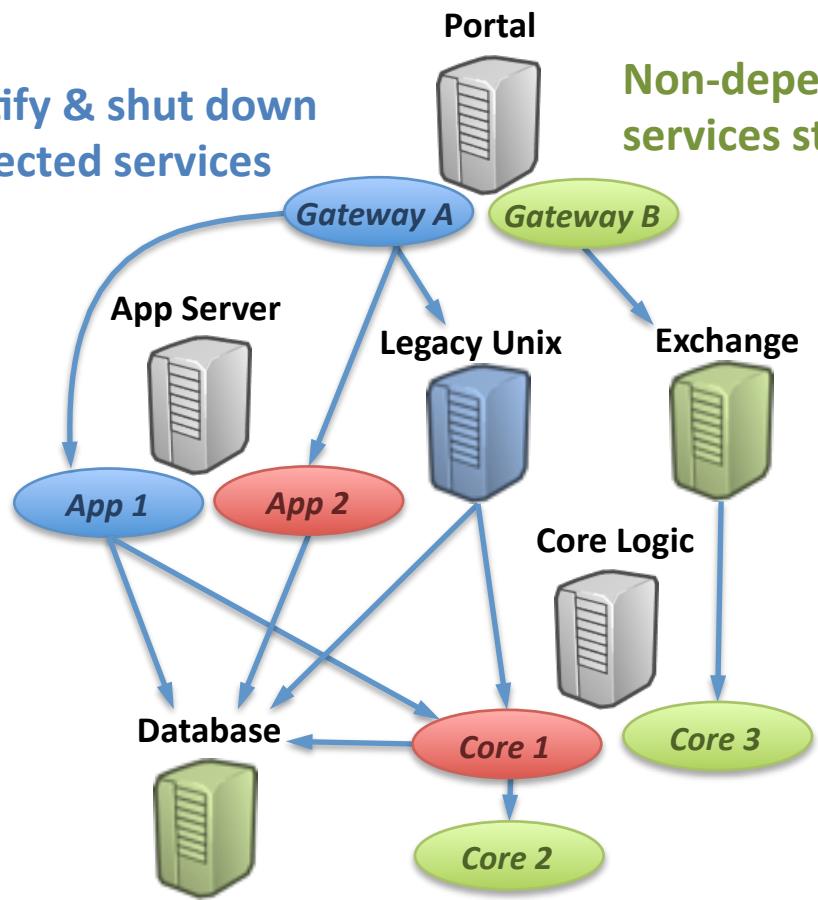


## Low computational cost



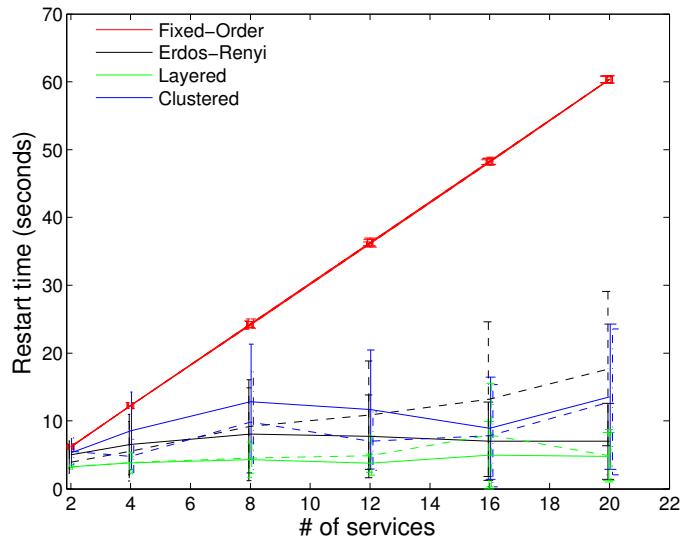
# Dependency-Directed Recovery

Identify & shut down  
affected services

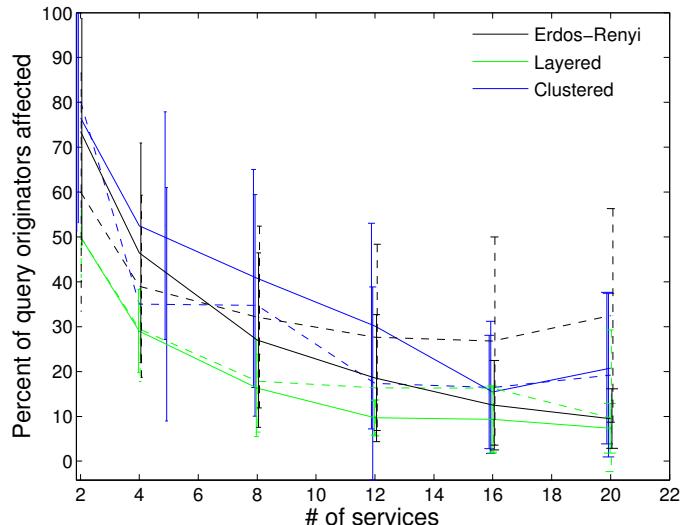


Non-dependent  
services still run

Dramatically better recovery time

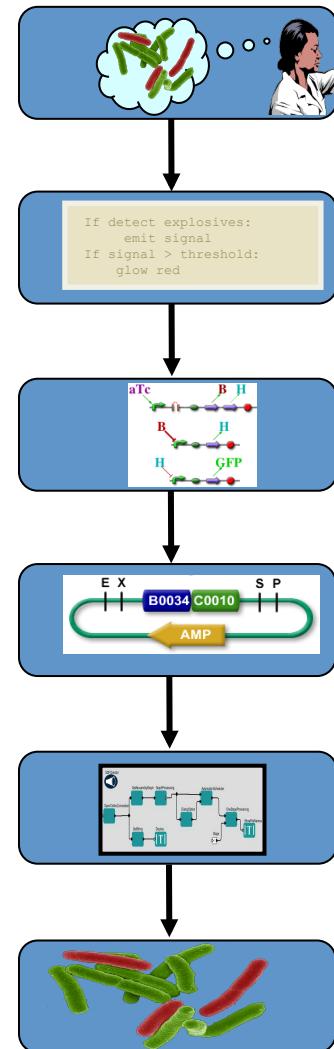
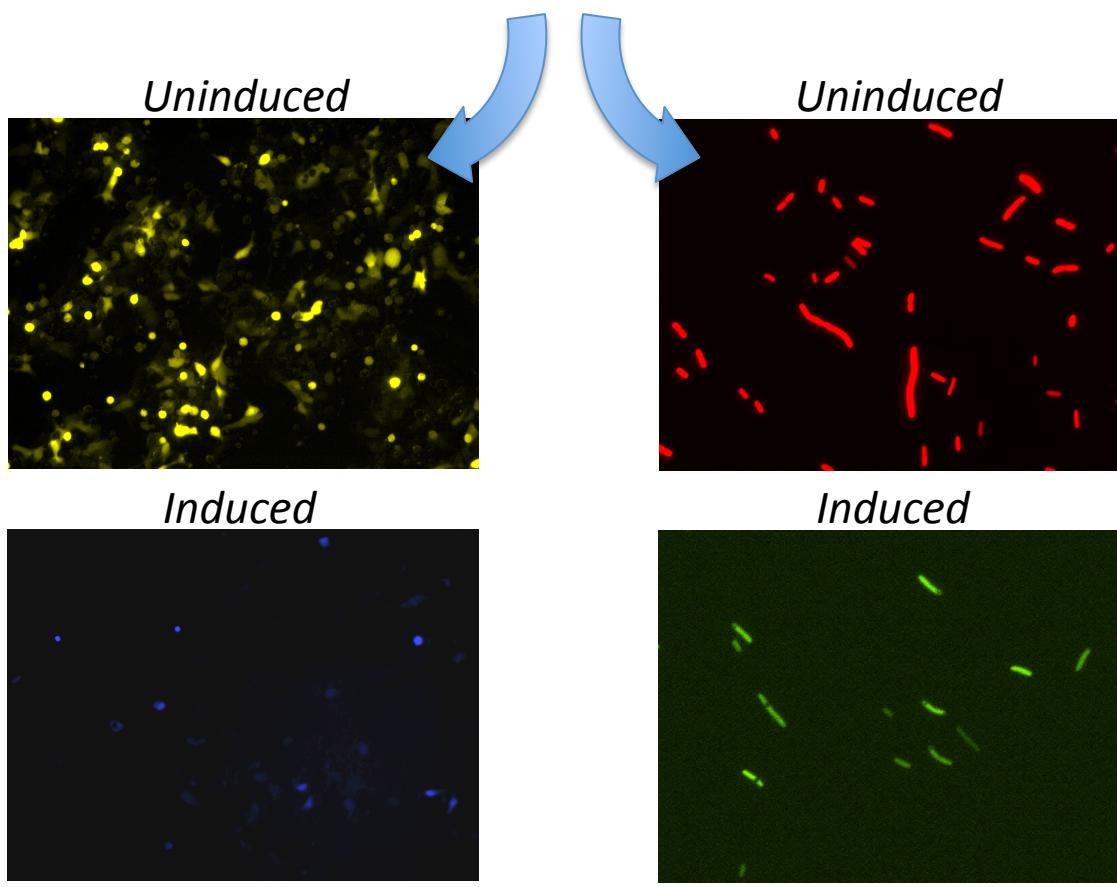


Fewer services disrupted



# Engineering Biological Systems

```
(def simple-sensor-actuator ()
  (let ((x (test-sensor)))
    (debug-1 x)
    (debug-2 (not x))))
```

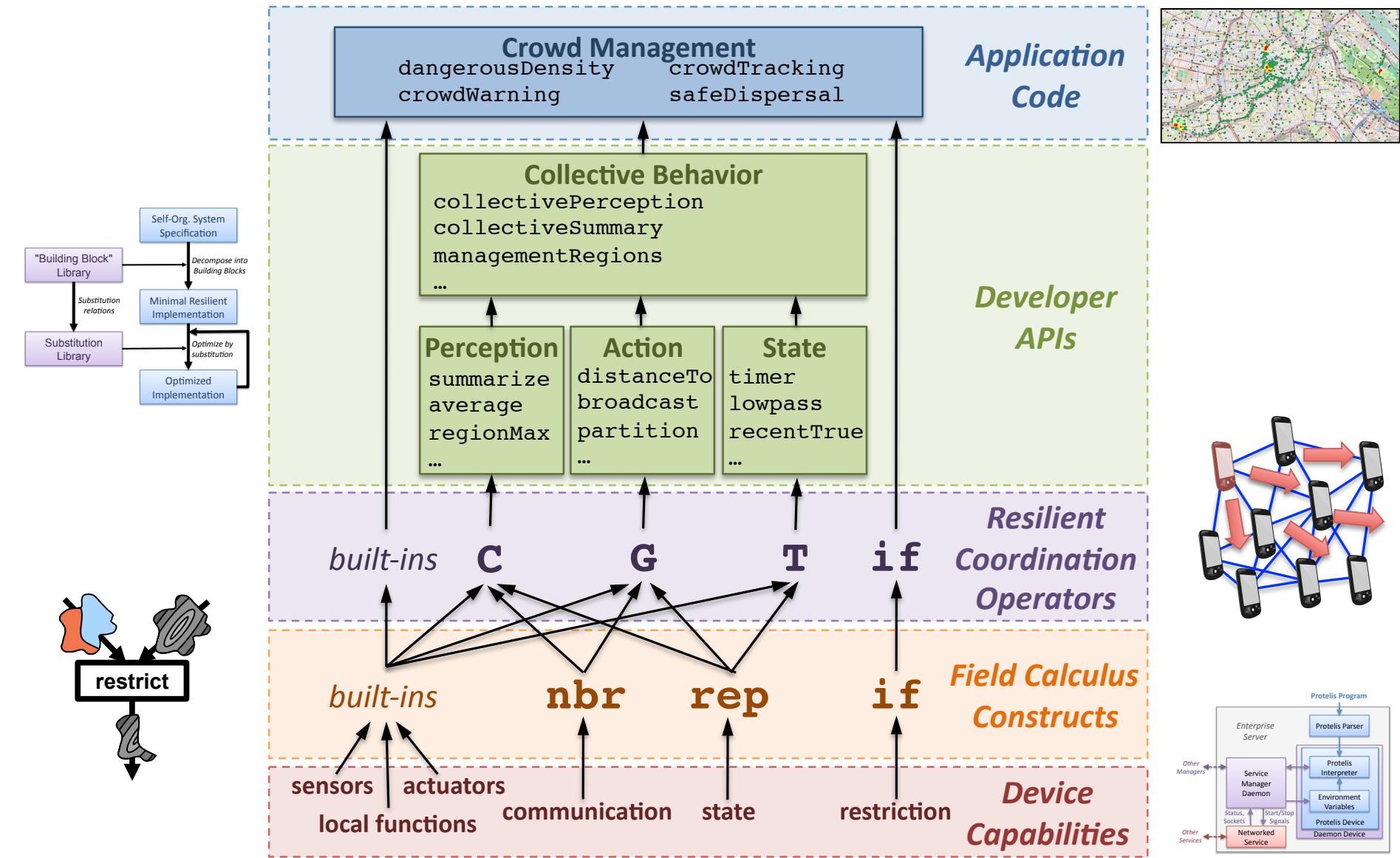


Mammalian Target

*E. coli* Target

[Beal et al., 2012]

# Summary: Aggregate Methodology



# Summary

---

- Major technological trends are all driving towards a world filled with distributed systems
- Aggregate programming aims at rapid and reliable engineering of complex distributed systems
- Field calculus provides a universal theoretical foundation for aggregate programming
- Resilient systems design can be simplified by an emerging self-organization toolbox
- Functional composition allows modulation, predictable convergence

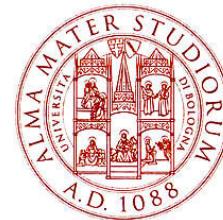
# Acknowledgements

## **Raytheon** **BBN Technologies**

- Shane Clark
- Partha Pal
- Kyle Usbeck



- Soura Dasgupta
- Raghu Mudumbai
- Amy Kumar



- Mirko Viroli
- Danilo Pianini



- Ferruccio Damiani

