#### **Current Frontiers**

#### Jacob Beal Lecture 5 of 5 on Spatial Computing ISC-PIF Summer School, 2009



# Agenda

- Predicting Approximation Error
- Dynamic Processes
- Compiling to Bacteria
- Local Checkability

# The Challenge of Composition

The Angenesing Status TEXAS INSTRUMENTS INCONFORMED Removed under Drive	
The TTL Data Book for Design Engineers	
Second Edition	
TEXAS INSTRUMENTS	R

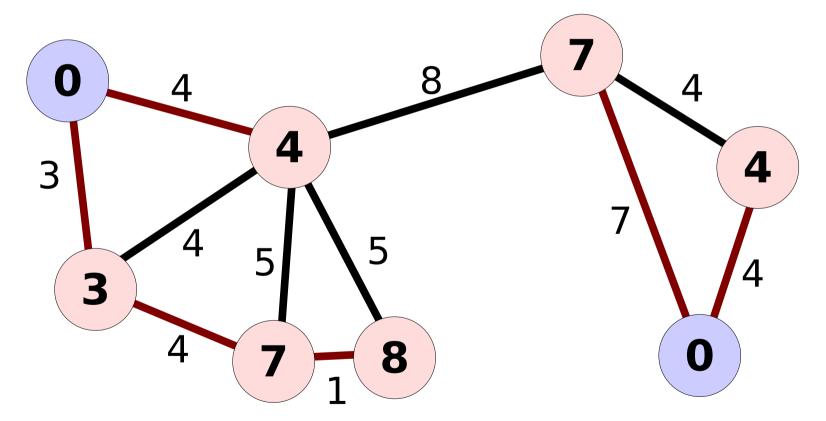
#### [Beal, et al. '08]

# **Discretization 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?

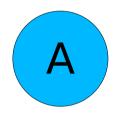
#### Distance-To

Distance from each device to nearest source



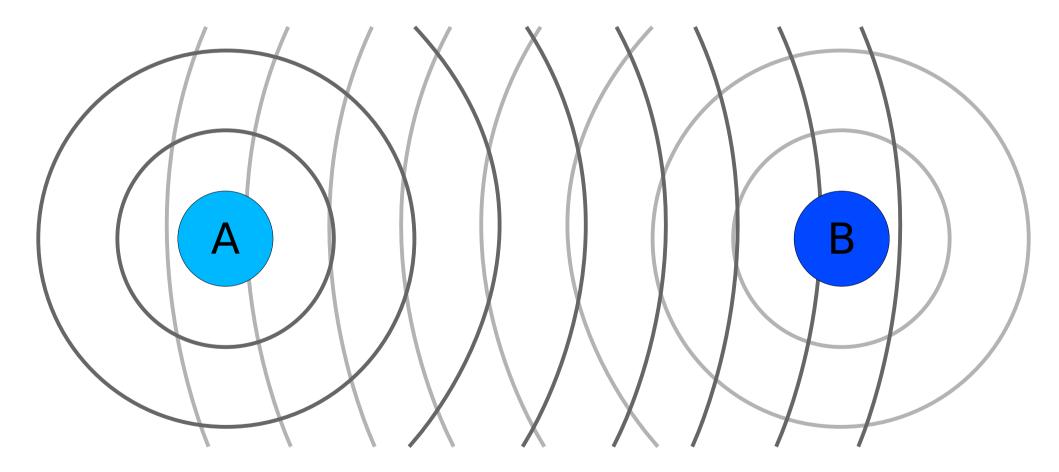
Distance in graph is proxy for real distance

# Geometric Program: Bisector

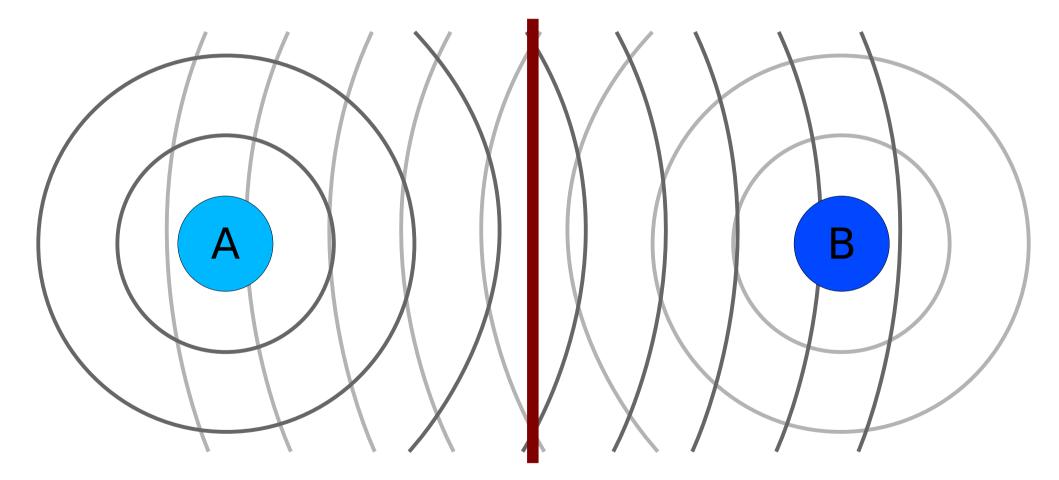




### Geometric Program: Bisector



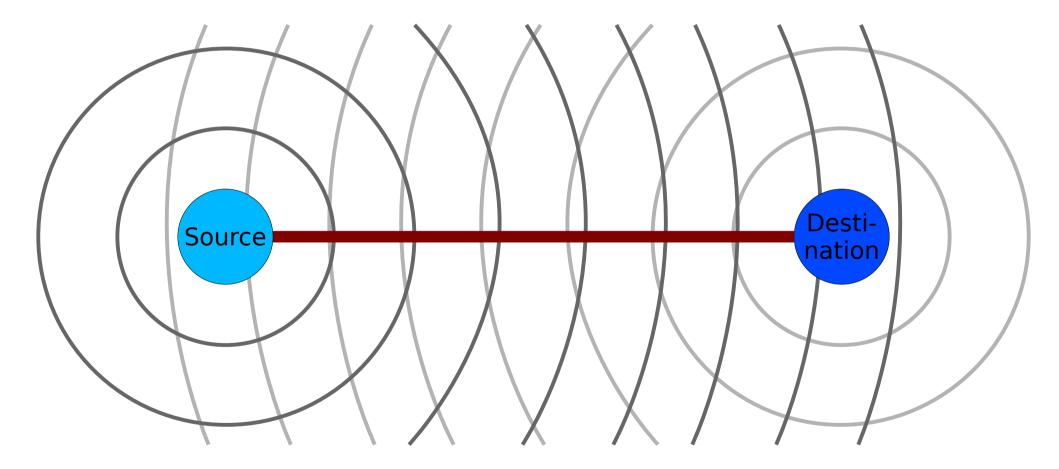
### Geometric Program: Bisector







(cf. Butera)



<sup>(</sup>cf. Butera)

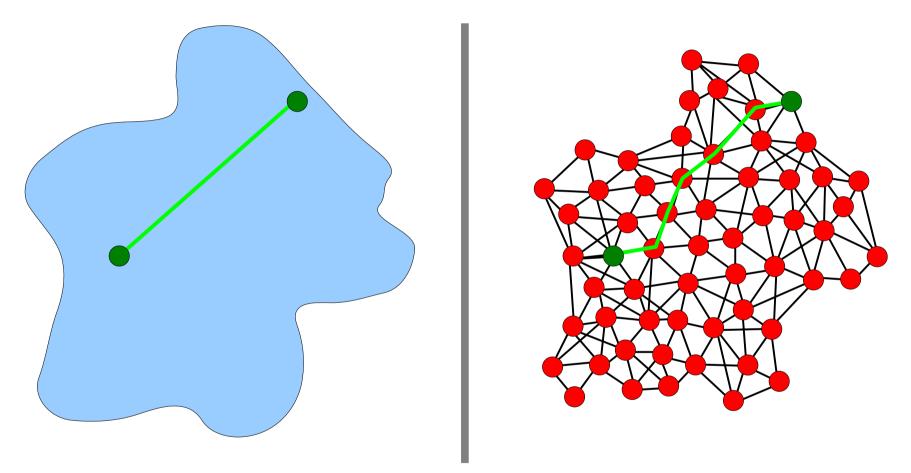


(cf. Butera)



(cf. Butera)

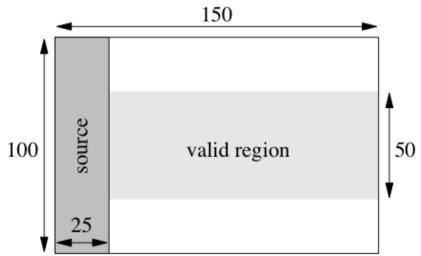
### **Discretization Error**



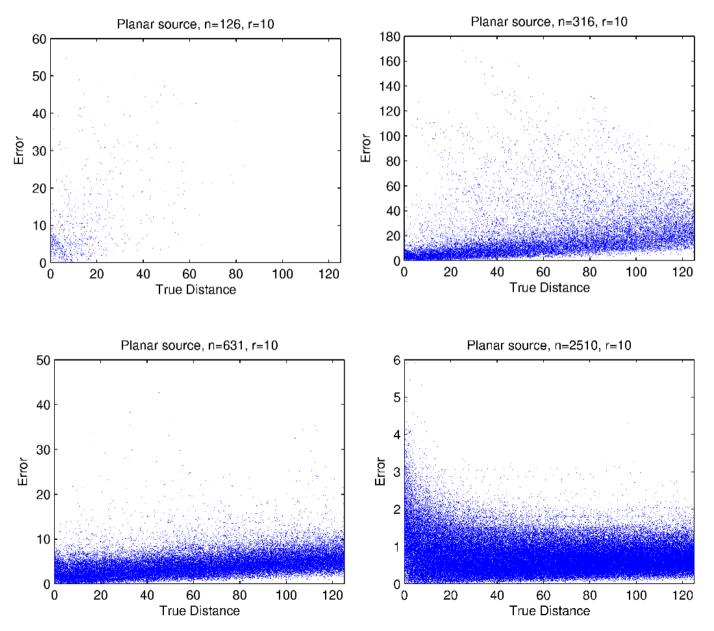
#### Prediction: $\varepsilon = \alpha \rho^{-2} d$

# Experimental Strategy

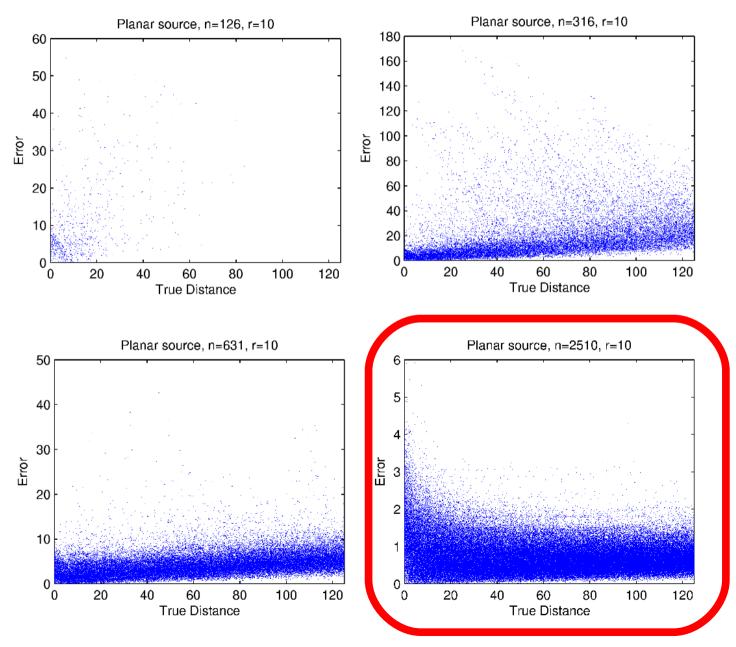
- Distribute *n* devices randomly in area *A*, communicating in *r* range, for density  $\rho$
- Perfect range information, no failures
- Survey wide range of parameters
  - 100 trials/combination, ~20K total



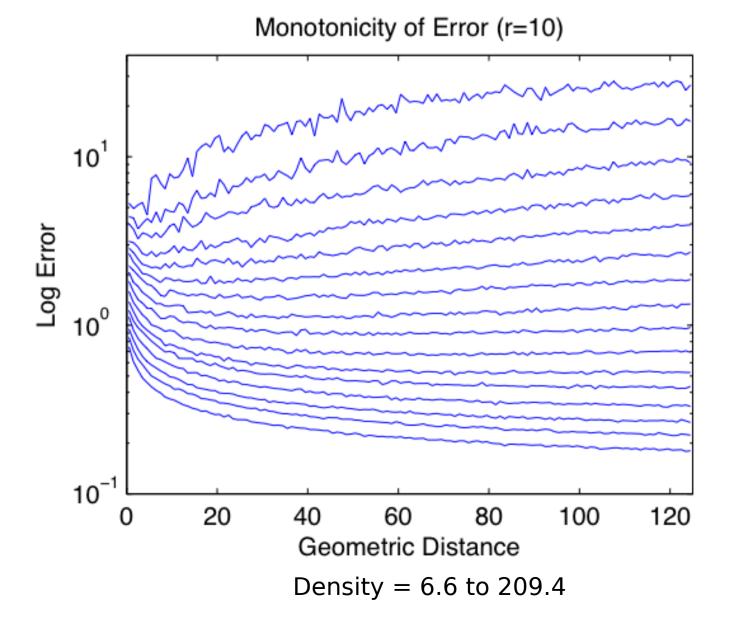
### Four Domains of Behavior



### Four Domains of Behavior



#### Density affects error monotonically



# Making an Empirical Model

$$\bar{\varepsilon_G} = \alpha d + \beta d^{-\gamma}$$

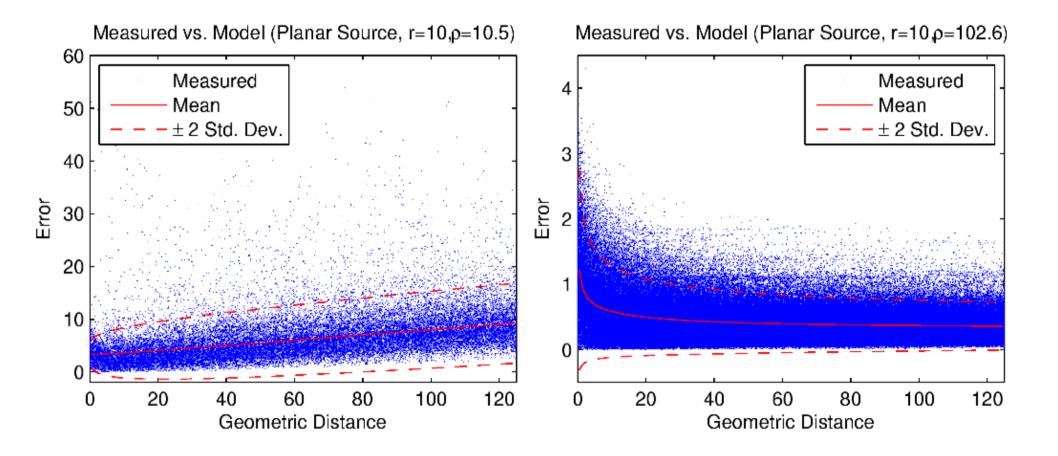
$$\bar{\varepsilon_G} = \alpha_1 \rho^{\alpha_2} d + \beta_1 \rho^{\beta_2} d^{(\gamma_1 + \gamma_2 \rho^{\gamma_3})}$$

$$\sigma_{\varepsilon_G} = \kappa + \lambda d^{-\mu}$$

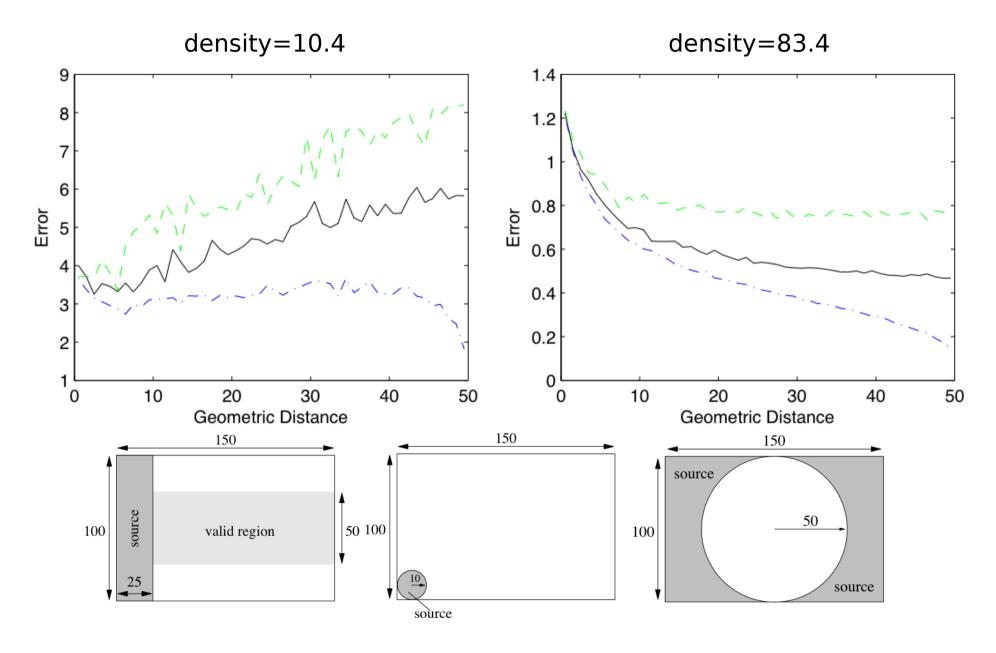
$$\sigma_{\varepsilon_G} = \kappa_1 \rho^{\kappa_2} + \lambda_1 \rho^{\lambda_2} d^{(\mu_1 + \mu_2 \rho^{\mu_3})}$$

Name	Value	95% confidence bounds	Name	Value	95% confidence bounds
$\alpha_1$	7.8	(6.8, 8.7)	$\kappa_1$	-25000	(-52000, 2000)
$lpha_2$	-2.14	(-2.19, -2.10)	$\kappa_2$	-4.5	(-4.9, -4.0)
$eta_1$	11.2	(10.8, 11.5)	$\lambda_1$	7.40	(7.07, 7.73)
$eta_2$	-0.516	(-0.526,-0.505)	$\lambda_2$	-0.529	(-0.541,-0.517)
$\gamma_1$	-0.292	(-0.303, -0.282)	$\mu_1$	-0.278	(-0.283, -0.272)
$\gamma_2$	1.6	(1.3, 1.9)	$\mu_2$	11	(5, 16)
$\gamma_3$	-0.77	(-0.86, -0.69)	$\mu_3$	-1.38	(-1.54, -1.21)
Mean			Standard Deviation		

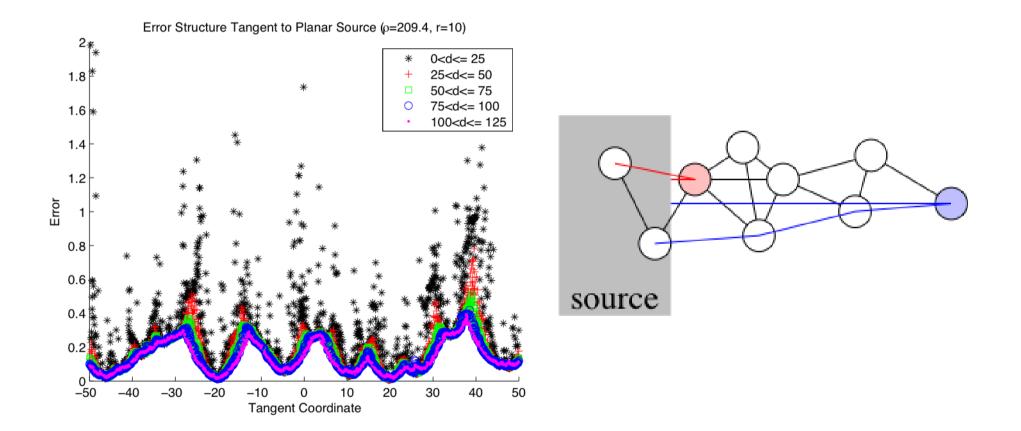
# Model Fit



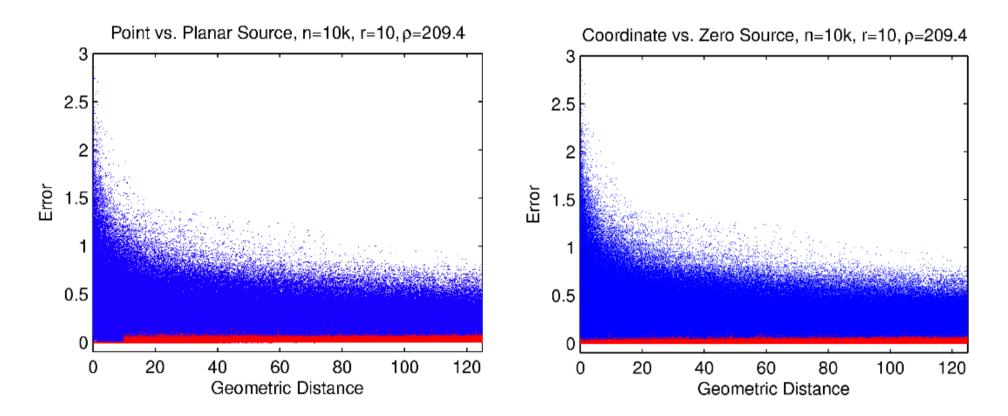
#### Source shape matters



# Understanding the Transient

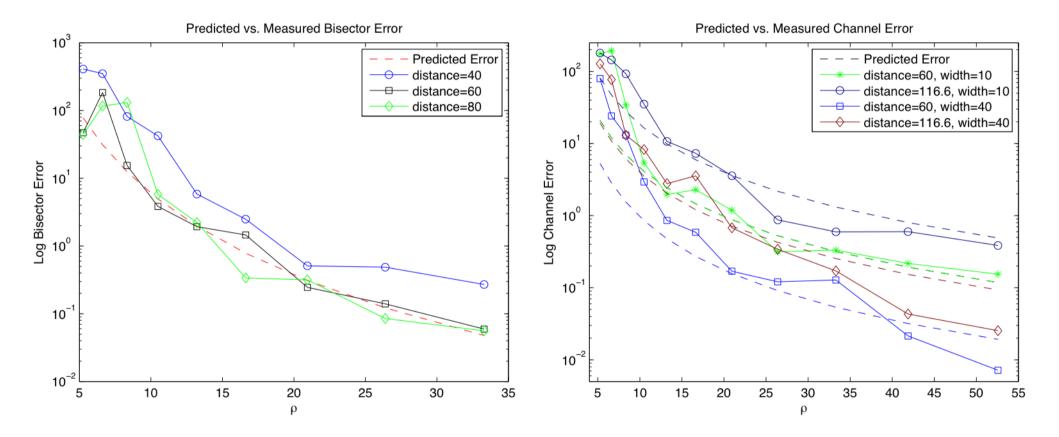


# **Transient Elimination**



# Point or "true depth" sources eliminate transient

### Model Predicts Channel/Bisector



## **Further Questions**

- What is a good model for the initial transient?
- How can the effect of source/medium shape be incorporated into the model?
- Can error prediction for gradient-based programs be automated?
- What other families of primitives can be predicted and composed?
- Can composing primitives ever reduce error?

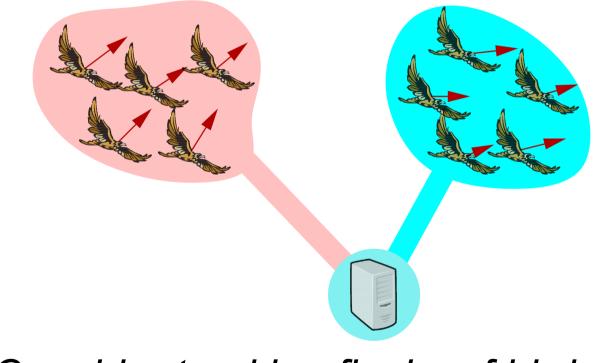
# Agenda

- Predicting Approximation Error
- Dynamic Processes
- Compiling to Bacteria
- Local Checkability

[Beal, in submission]

# **Dynamically Allocate State**

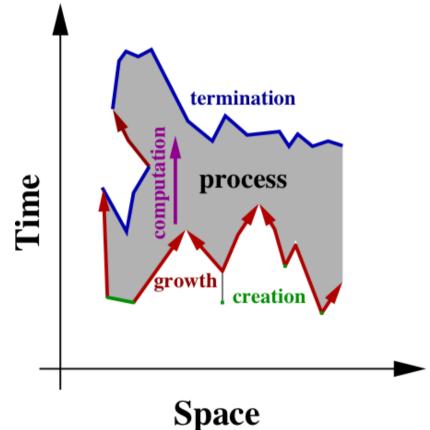
Many processes must create state (e.g. objects, processes) in response to their environment



Consider tracking flocks of birds...

# **Definition of Process**

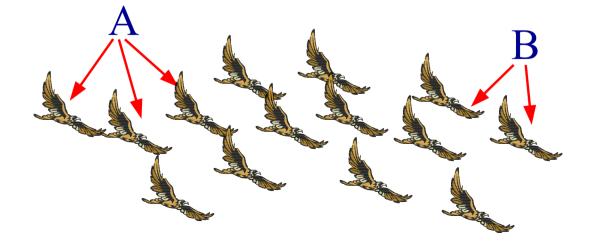
- Let *p* be an executing instance of a program at a point *m*
- p' on m' ∈ N(m) if in the same process if p can use state from p'
- Specifiable by 5 behaviors: creation, growth, sharing, computation, termination



#### **Problem of Independent Creation**



#### Are the visible birds part of the same flock?



# UIDs can't distinguish processes

**Theorem:** if instances of processes form an equivalence class  $\sim$ , no algorithm for creating program instances exists that can guarantee safe creation in less than O(diameter/c) time

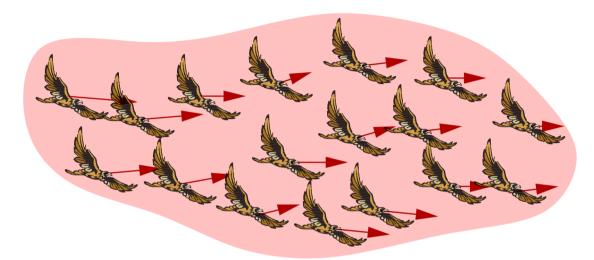
- Proof sketch:
  - Time bound  $\rightarrow$  space-like separation possible
  - choice of ~ only affected by causally related points
  - Algorithm must fail on one of:
    - m and m' create P
    - m and m' create P'
    - *m* creates *P*, *m*' creates *P*'

Solution: dynamically determined extent

Proposed new proto construct:

```
(procs (elt sources)
  ((var init evolve) ...)
 (same? run? &optional terminate?)
 . body)
```

#### Example: tracking a flock

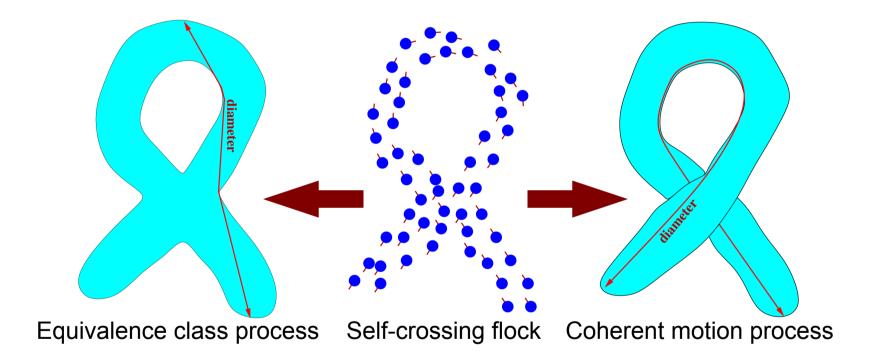


#### flock identity = similarly moving birds

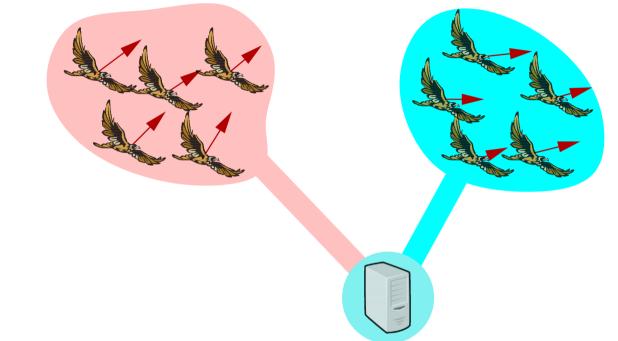
```
(def close-vec (base other err)
 (< (len (- base other)) (* err (len base))))</pre>
```

```
(def track-flocks ()
 (procs (bird-vec bird-vecs)
       ((flock-vec
           bird-vec
              (average (filter
                    (lambda (v) (close-vec flock-vec v 0.1))
                    bird-vecs))))
 ((close-vec flock-vec (nbr flock-vec) 0.1)
       (find-if (lambda (v) (close-vec flock-vec v 0.1))
                    bird-vecs))
 (measure-shape)))
```

#### Implication: self-crossing!



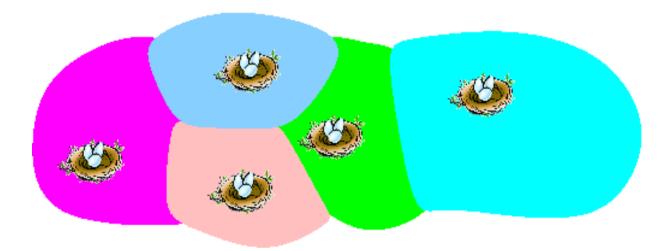
#### Example: reporting on flocks



#### flocks calculates reporting UID after forming

```
(report-data-stream (data-set base)
(procs (data data-set)
  ((uid (1st data) uid)
   (src true (find uid (map 1st data-set)) diameter))
  ((= uid (nbr uid))
  (dilate src diameter))
  (channelcast
   src base 2
  (2nd (find uid data-set :key 1st)))))
```

#### Example: finding the nearest nest



#### Processes compete on distance to nest

### **Further Questions**

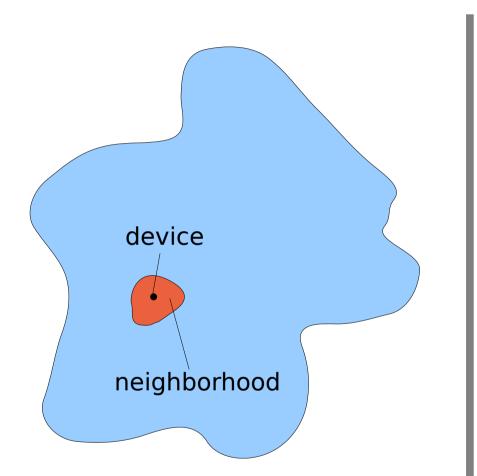
- What are good primitives for expressing dynamic process formation?
- What sorts of dynamic process-based
   algorithms are useful for various tasks?
- How can reportable identity be tracked for a process that splits and rejoins its parts?

# Agenda

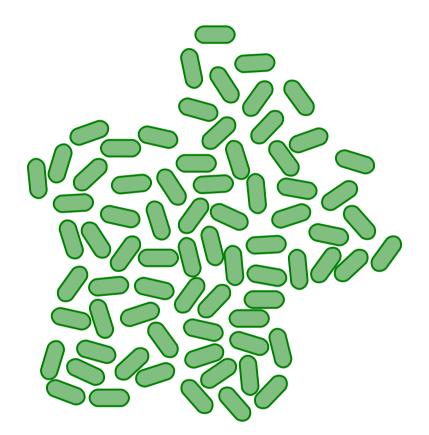
- Predicting Approximation Error
- Dynamic Processes
- Compiling to Bacteria
- Local Checkability

[Beal & Bachrach, '08]

### Amorphous Medium



Continuous space & timeInfinite number of devicesSee neighbors' past state



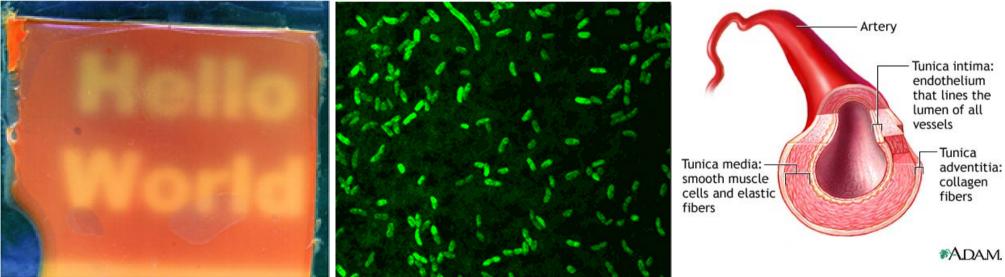
Approximate with:Discrete colony of cellsChemicals transmit state

# Why spatial computing?

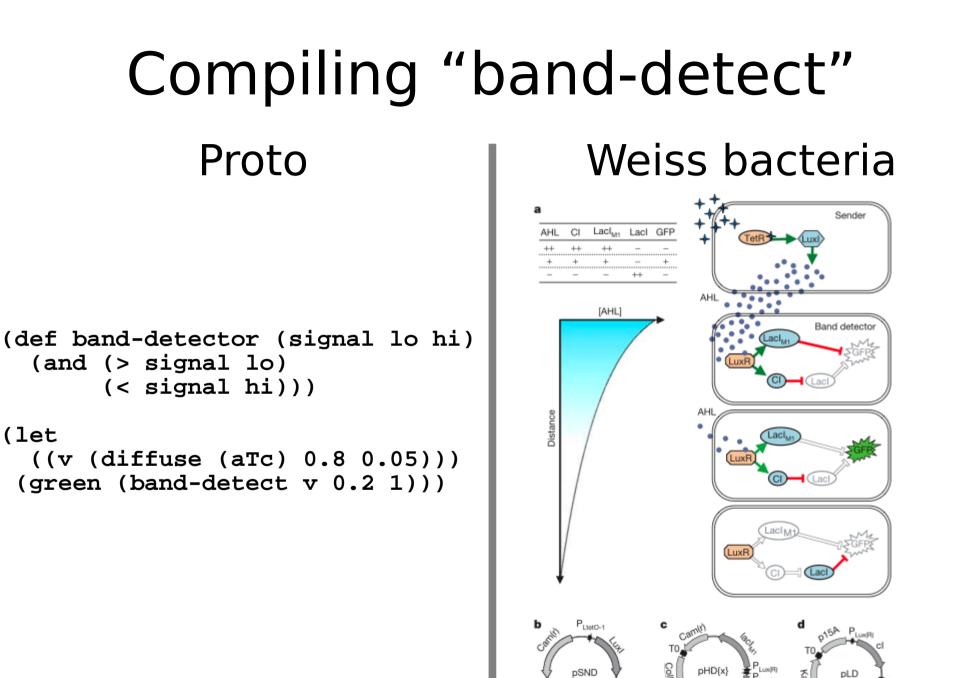
#### Pointwise

#### Global

#### Differentiated



(UT Austin) (v. fischeri Genome Project)

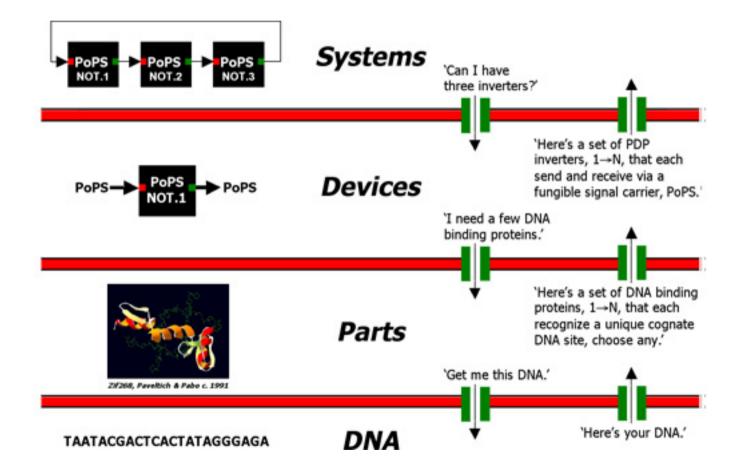


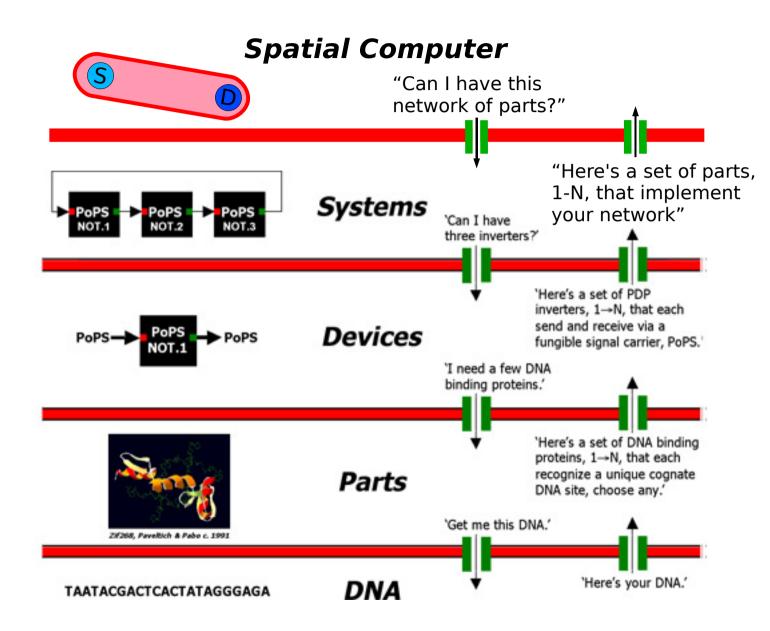
ColEt

## HLLs & Bacteria

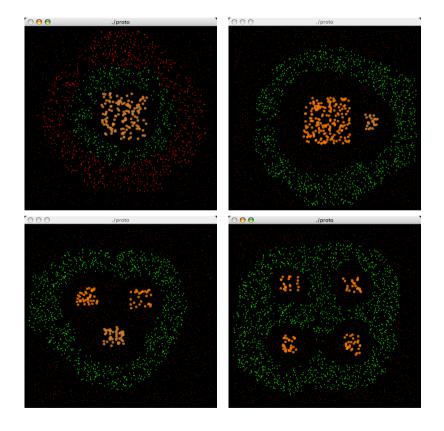
- High-level languages:
  - Shorter programs mean less efficient code
  - Optimizing compilers can help
- Bacteria
  - Extremely tight resource constraints
  - Inherently parallel chemical execution

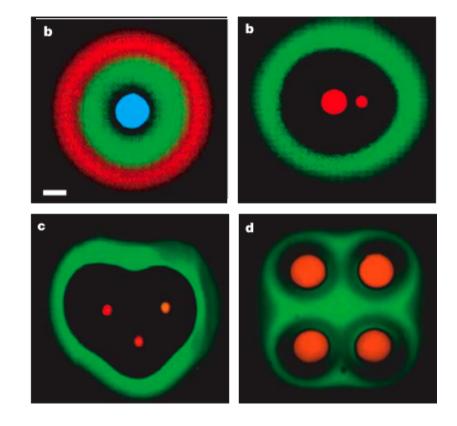
# Synthetic Biology Vision

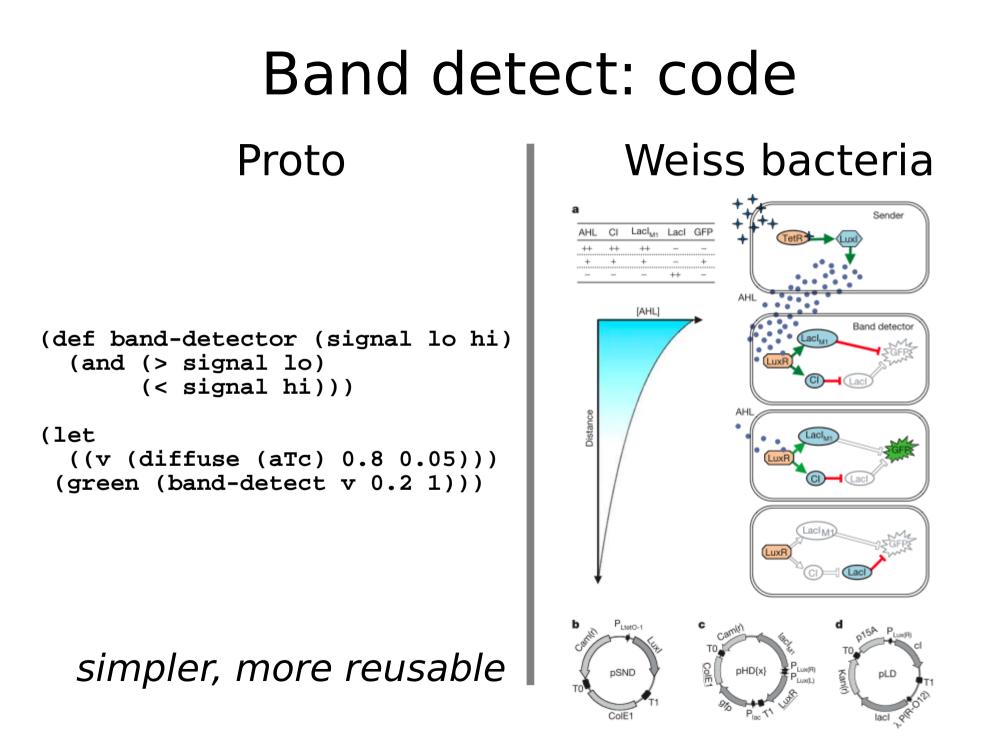




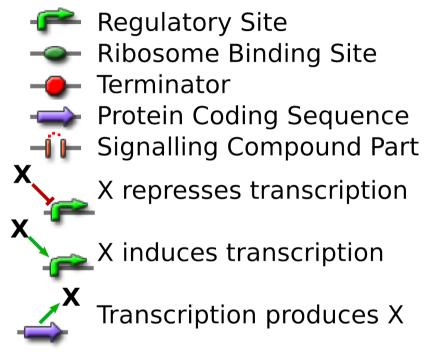
# Band detect: behavior Proto Weiss bacteria





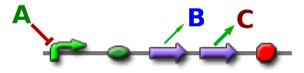


### **BioBrick Primitives**

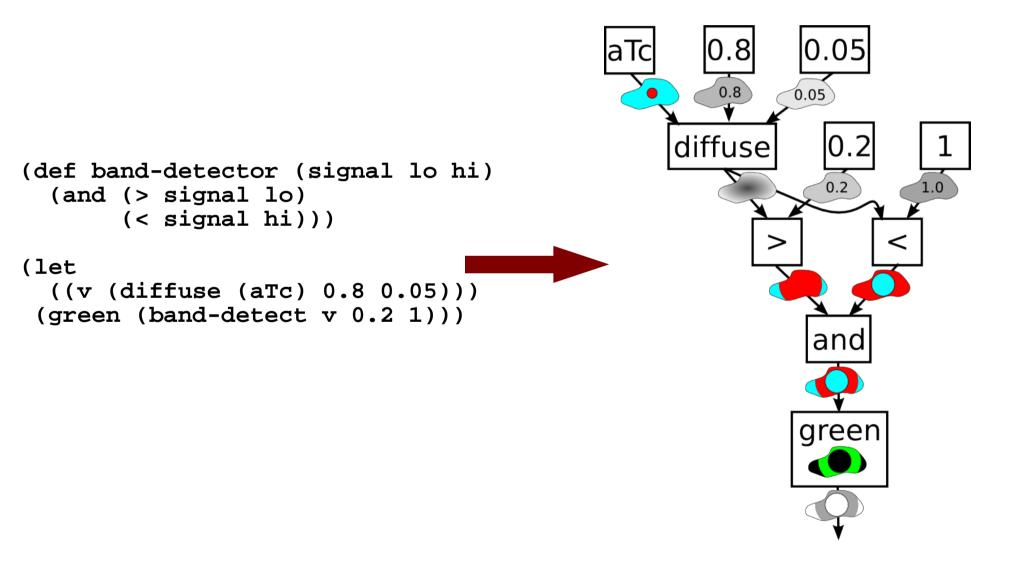


 $X+Y \rightarrow Z$  X and Y react, forming Z

Typical functional unit:



#### In Proto:



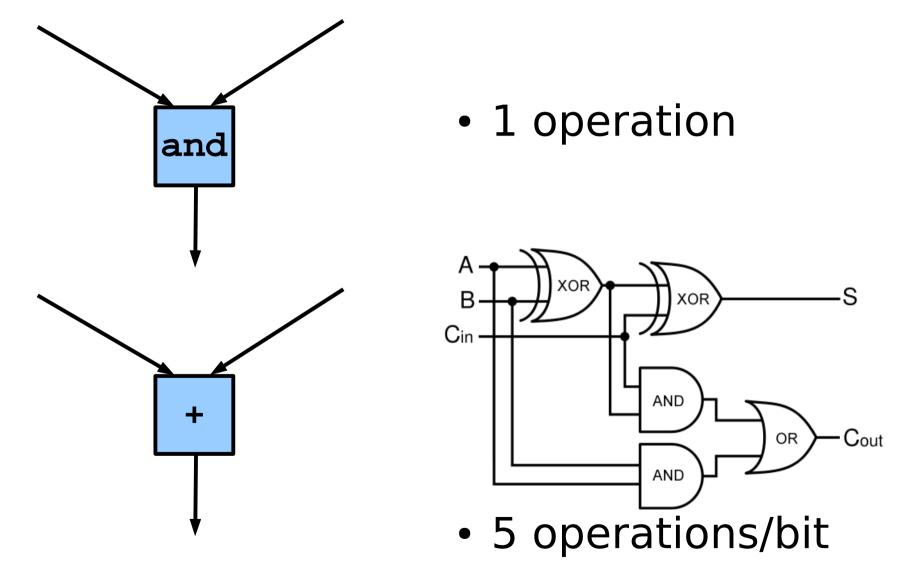
## Proto to GRNs: First Steps

- Logical: and, or, not 🗸
- Flow control: IF, MUX 🗸
- Arithmetic: +, -, \*, /, log, exp, ...
- Relational: >, <, =

Two possible implementations:

- Regulation
- Reaction

# **Digital Arithmetic is Expensive**



Use digital for booleans, analog for numbers

## Arithmetic

- *c*: constitutive expression
- (+ A в): same chemical represents both
- (- A B): A+B→C
- (log A), (exp A): lookup tables

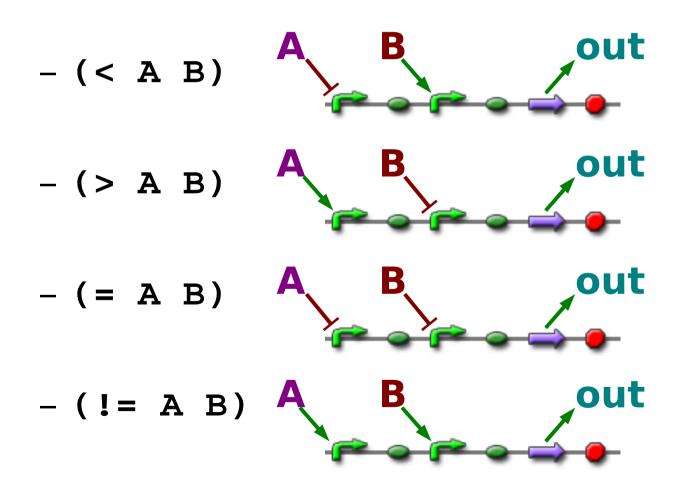
- approximate w. summary of > tests?

• (\* А в), (/ А в): log add, subtract

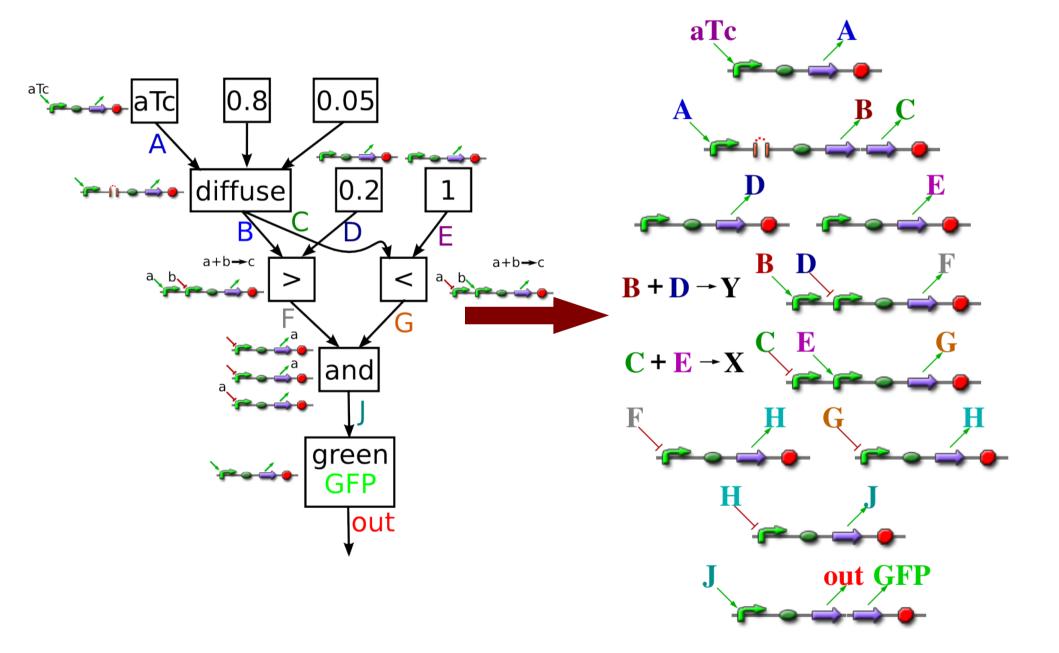
#### Range? How many bits?

#### Relational: A→D conversion

• **A**+**B**→**C** 



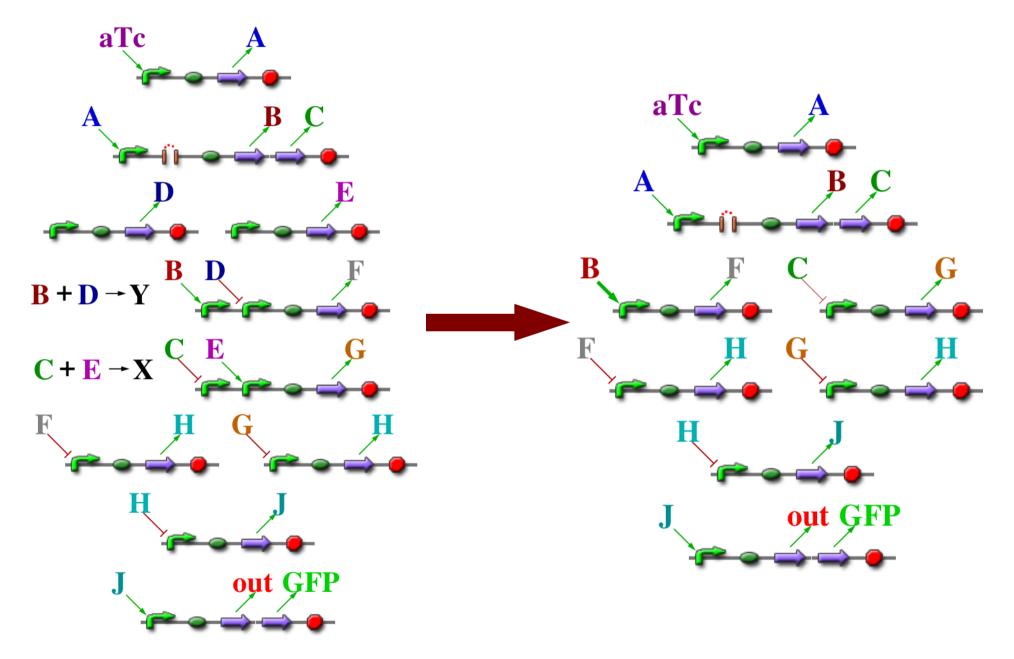
#### Naïve Implementation



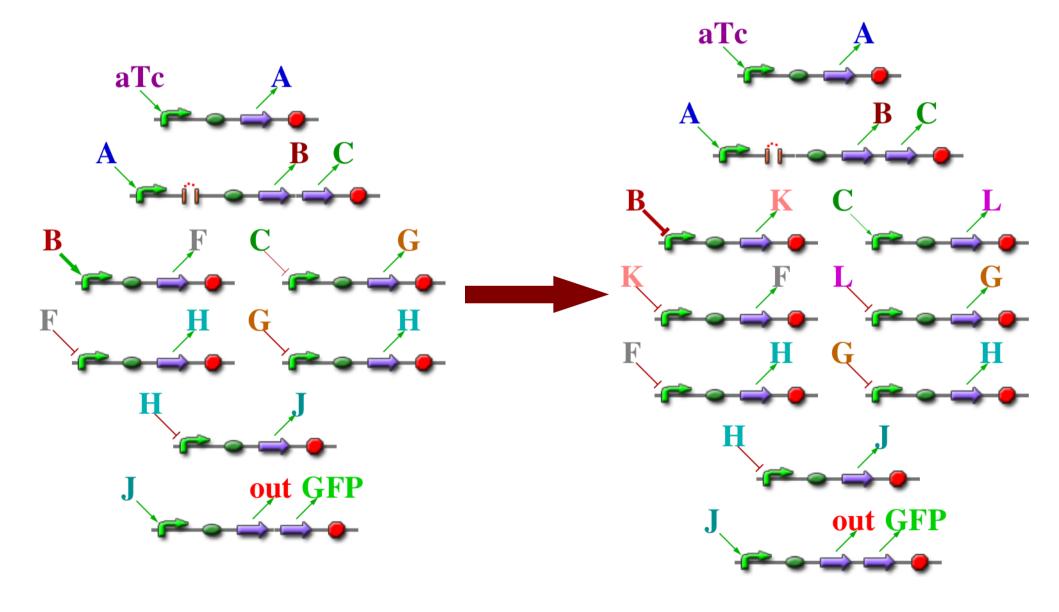
#### **Resources Required**

Resource	Hand Tuned	Naive
Signal-carrying chemical	3	11
Protein coding sequence	6	14
Promoters	5	14
Intercellular messengers	2	2
Chemical reactions	0	2

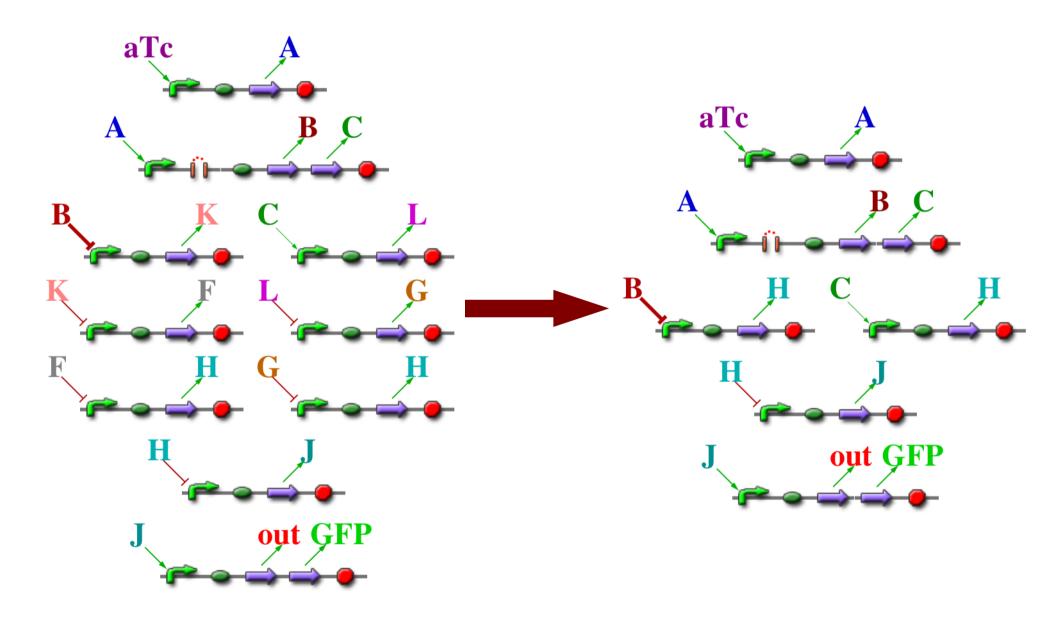
## **Optimize: Constant Elimination**



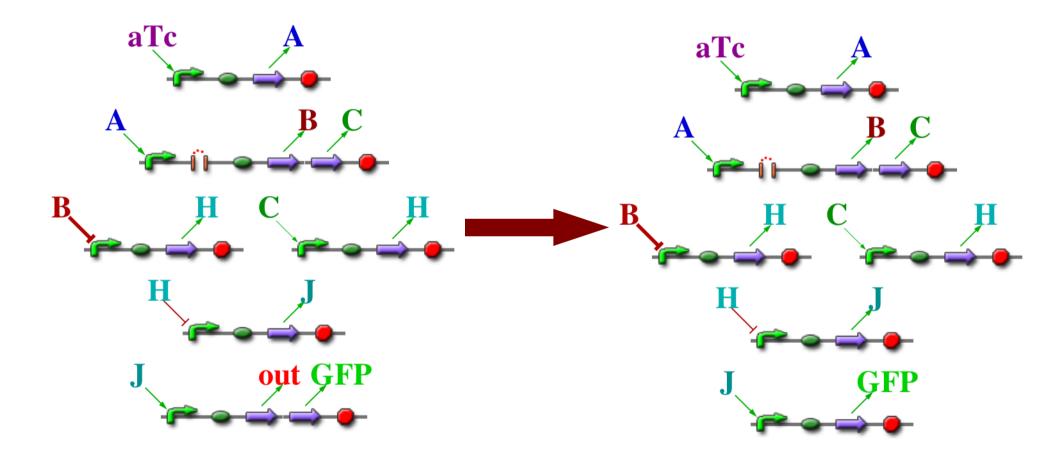
### Optimize: Algebraic Simp. (1/2)



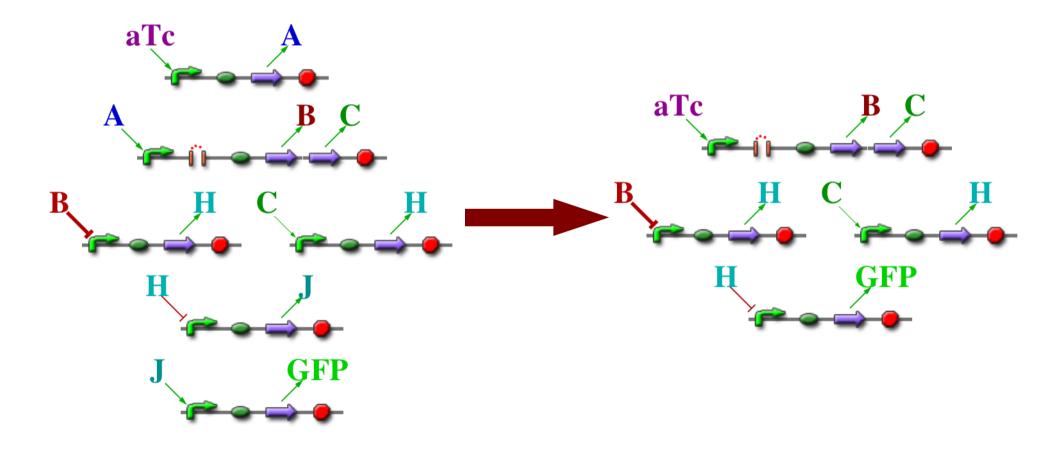
## Optimize: Algebraic Simp. (2/2)



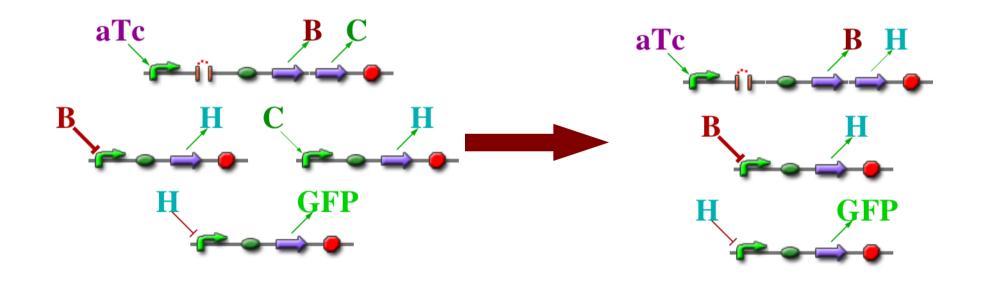
### **Optimize: Dead Code Elimination**



### **Optimize: Copy Propagation**



### **Optimize: Use-Definition Analysis**



#### **Resources Required**

Resource	Hand Tuned	Naive	Optimized
Signal-carrying chemical	3	11	3
Protein coding sequence	6	14	6
Promoters	5	14	5
Intercellular messengers	2	2	2
Chemical reactions	0	2	0

# **Further Questions**

- How many bits precision can be supported?
- What are good biological implementations of other operations?
- Are there useful bio-specific optimizations?
- How can timing be managed?
- Will it work?

# Agenda

- Predicting Approximation Error
- Dynamic Processes
- Compiling to Bacteria
- Local Checkability



## Model

Motivation: morphogenesis & high cost of state

- Crystalline network of asynchronous FSMs
- Boundary & directions are distinguishable
- *r*-hop neighborhood

$$\begin{array}{c} r=1 \\ \hline 1 \rightarrow 2 \rightarrow 0 \rightarrow 1 \rightarrow 1 \rightarrow 0 \rightarrow 2 \end{array}$$

*n* devices  $\cdot$  S states  $\rightarrow$  set C<sub>n,S</sub> of configurations

### Local Checkability

Let *T* be a pattern. A binary function  $\Theta: N_r \rightarrow \{T, F\}$ over neighborhoods of radius *r* is a **Local Check Scheme** (LCS) for *T* if:

- For all X in  $C_s$ ,  $\Theta(X) = 1 \rightarrow X \in T$
- For all *n*, such that  $T \cap C_{n,S} \neq \emptyset$ , there is  $X \in C_{n,S}$  such that  $\Theta(X)=1$

The Local Check Radius of T is minimum r for which an LCS exists.  $LCR(T) = \infty \rightarrow not$  locally checkable

#### Self-stabilization → Local Checkability

**Theorem:** If *F* is a self-stabilizing algorithm that makes *T* using *r* neighborhoods, then  $r \ge LCR(T)$ **Corollary:** For all repeat patterns  $T_q$ ,  $LCR(T_q) \le |q|/2$ 

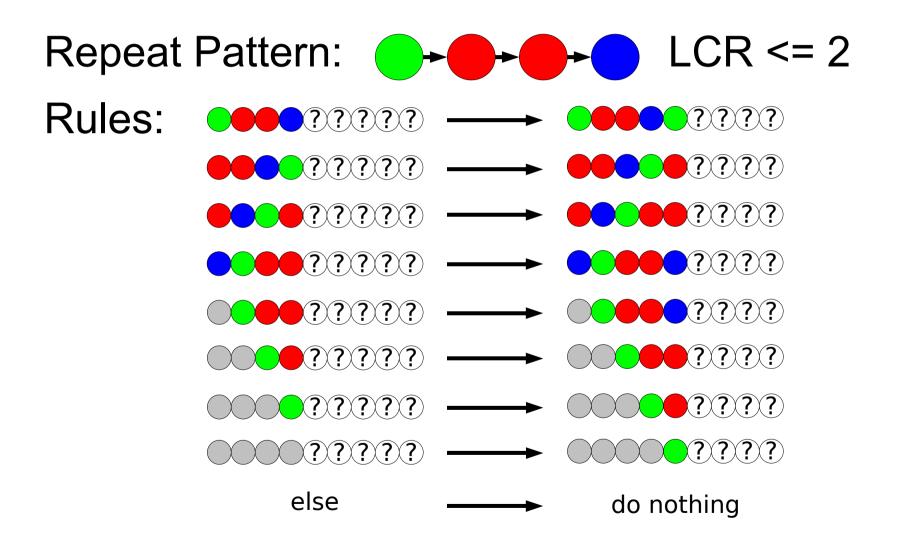
Intuition: if you can't locally check, the algorithm can't know if it's finished.

#### Local Checkability — Self-Stabilization

Proof by construction:

- For "Single-choice" patterns:
  - Given LCR(T) = r, choose a neighborhood of 2r
  - If left neighbors are correct, set self to match
  - Otherwise, do nothing
- All others use a *self-organizing turing machine* that adds transient "marker" states to make the pattern "single choice."

### Example of "Single-Choice" Pattern

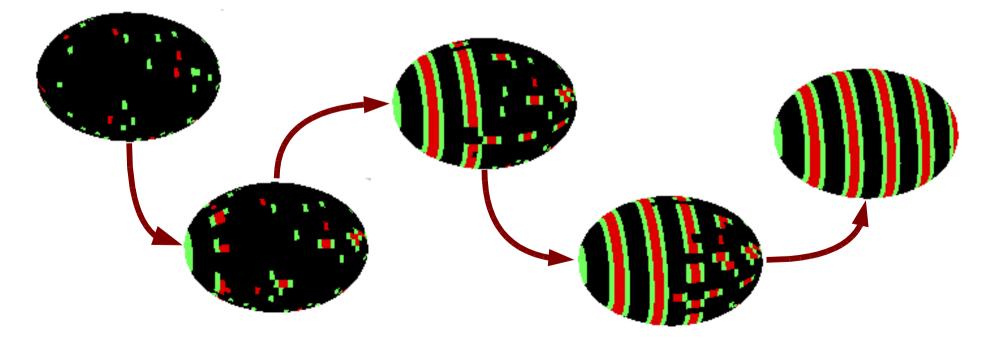


#### Example of Self-Organizing Turing Machine

- Pattern: T<sub>100</sub>T<sub>1000</sub>
- 10 rules, 3 extra states
- Example of execution:

#### **2D Pattern Primitives**

- Repeat patterns
- Proportionate patterns
- Fractal-generatable curves



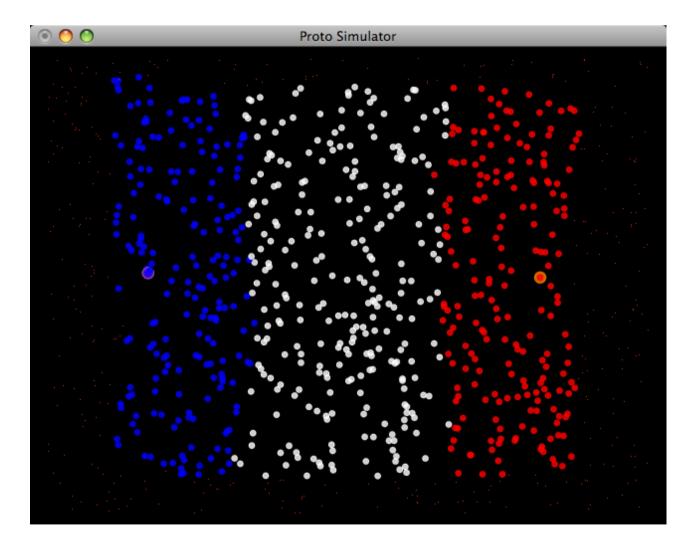
#### **Further Questions**

- Are there other useful basis patterns?
- Can these results be extended to irregular spaces or amorphous networks?
- What bounds are there for logarithmic state?
- What is an appropriate language for expressing the family of locally checkable patterns?

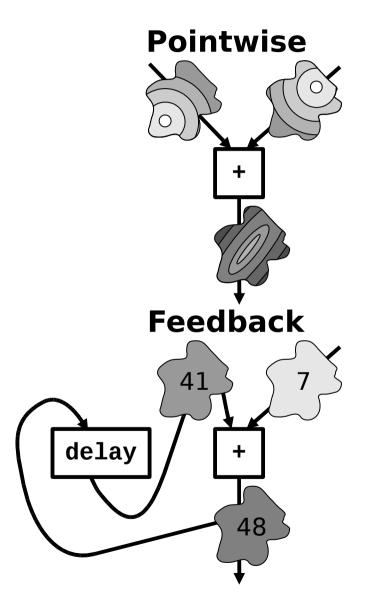
### What have we learned?

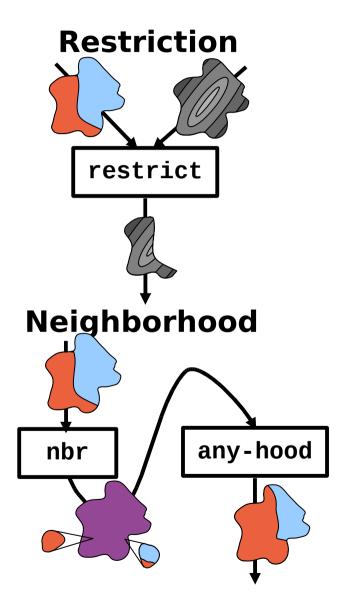
- Amorphous Medium abstraction simplifies
   programming of space-filling networks
- Appropriate space and time operations make it easy to compile global descriptions into local actions that approximate the global
- Geometric metaphors allow complex behaviors to be programmed with very short programs.
- Self-healing programs adjust to changes, and behave predictably when composed together.
- Spatial computing is filled with open questions and new frontiers for research.

#### Lecture 1: Spatial Computers & Fields

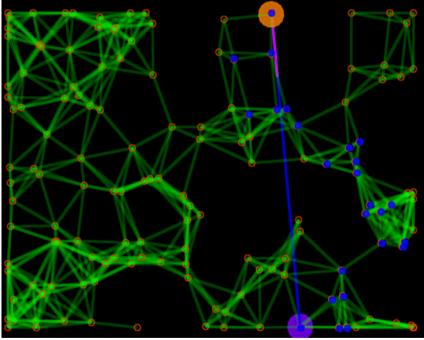


#### Lecture 2: Continuous Space-Time Programs

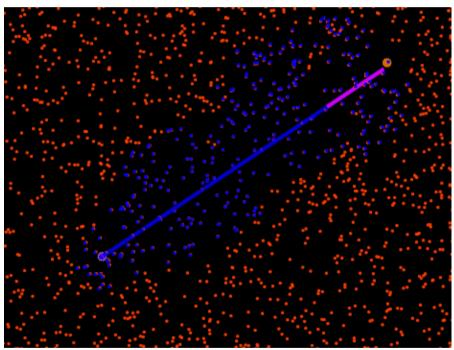




#### Lecture 3: Discrete Approximation & Self-Healing

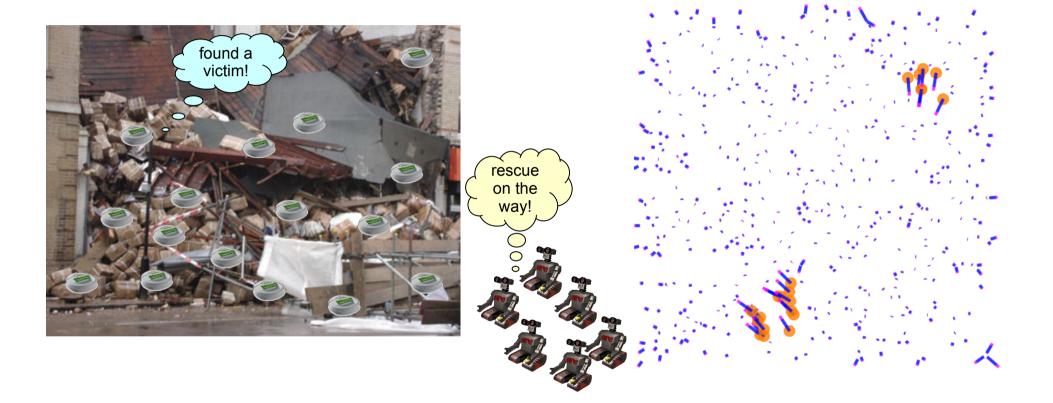


150 devices



2000 devices

#### Lecture 4: Moving Devices



#### *Robot motion* = *vector fields*

#### Lecture 5: Current Frontiers

