

# AMORPHOUS AND CELLULAR COMPUTING

Hal Abelson

Radhika Nagpal

Jake Beal

Randy Rettberg

Lauren Clement

Erik Rauch

Chris Hanson

Gerald Jay Sussman

Attila Kondacs

Ron Weiss

Tom Knight



MIT Artificial Intelligence Laboratory

# A scientific and technological effort to identify

- Methods for obtaining coherent behavior from the cooperation of vast numbers of unreliable parts, that are interconnected in unknown, irregular, and time-varying ways
  - Organizational principles
  - Algorithms
  - Programming models
  - Compilation technology targeted to appropriate substrates
- New computing substrates, both from traditional silicon technology and from molecular biology



# Why is this interesting?

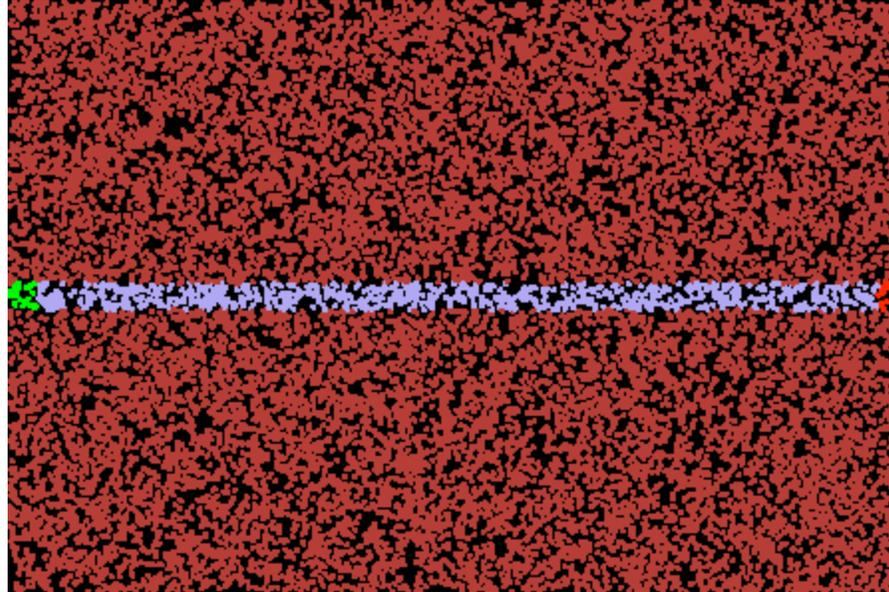
- Physically feasible at any scale
- Forces robustness of design
- Potentially extremely inexpensive
- Provides the possibility of *bulk computation*
  - smart paints
  - smart gels
  - concrete by the Megaflops
  - vast sensor networks
- *Our programming models run out when there are too many elements to program individually, or even to name.*

"It is unlikely that we could construct automata of a much higher complexity than the ones we now have, without possessing a very advanced and subtle theory of automata and information. This intellectual inadequacy certainly prevents us from getting much farther than we are now...A simple manifestation of this factor is our present relation to error checking...

With our artificial automata we are moving much more in the dark than nature appears to be with its organisms. We are, and apparently, at least at present, have to be much more 'scared' by the occurrence of an isolated error and by the malfunction which must be behind it. Our behavior is clearly that of overcaution, generated by ignorance."

-- John von Neumann (1948)

# Example of an Amorphous Computing Medium

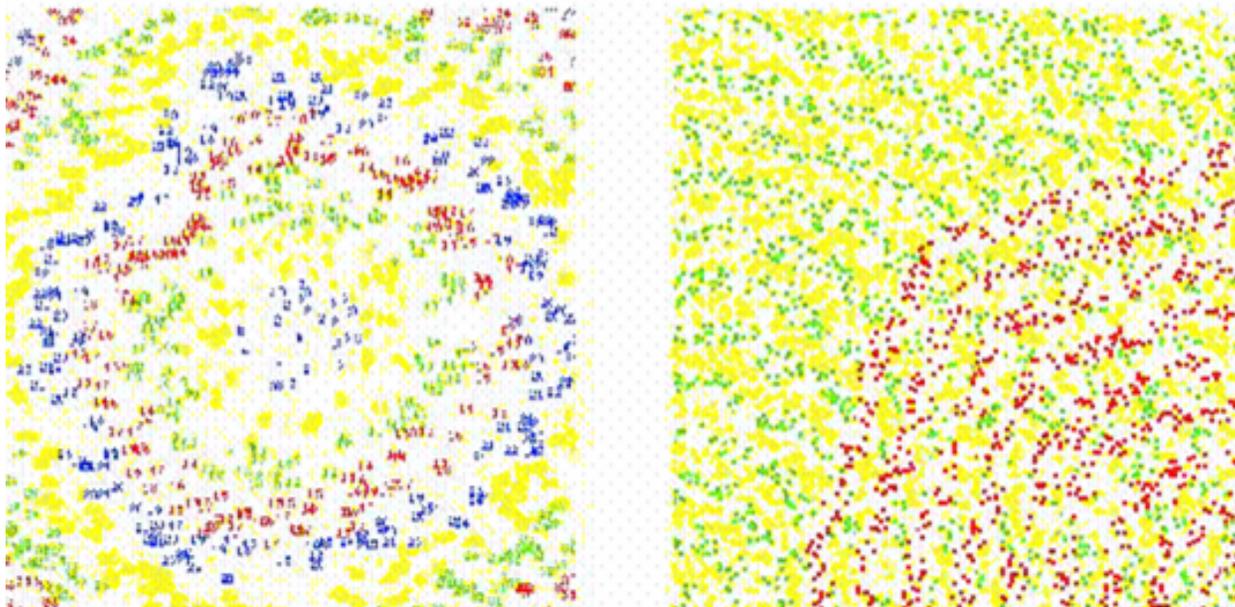


An amorphous medium has independent computational particles, all identically programmed. Each particle is represented by a spot in the picture, and different states are shown by different colors.

# Our amorphous computing model

- Computing elements sprinkled on a surface or in a volume
- Too many to individually program or even to name
- Each talks to a few neighbors, but not reliably
- Not synchronous, nor regularly arranged

Crude local coordinate systems can be constructed by intersecting countdown waves radiating from several loci.



$$E(d) = [\# hops] \cdot [com radius] \cdot H(N)$$

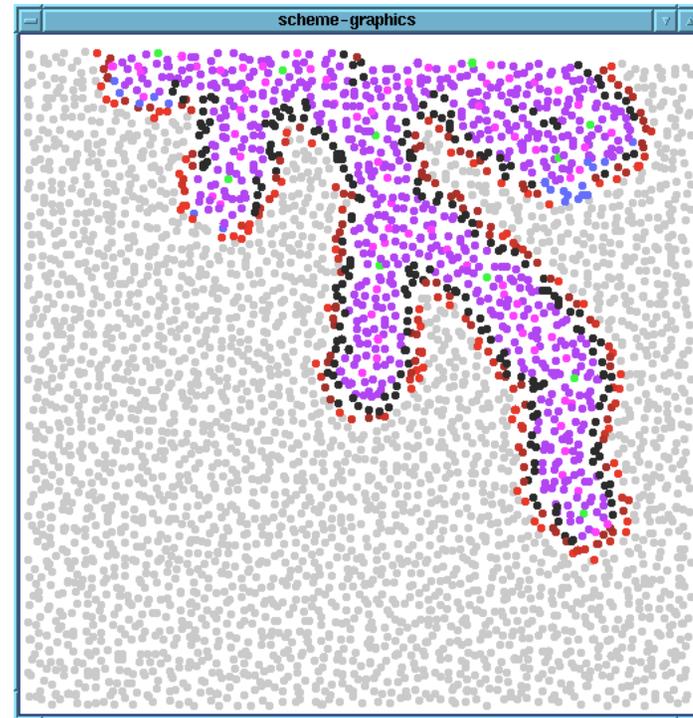
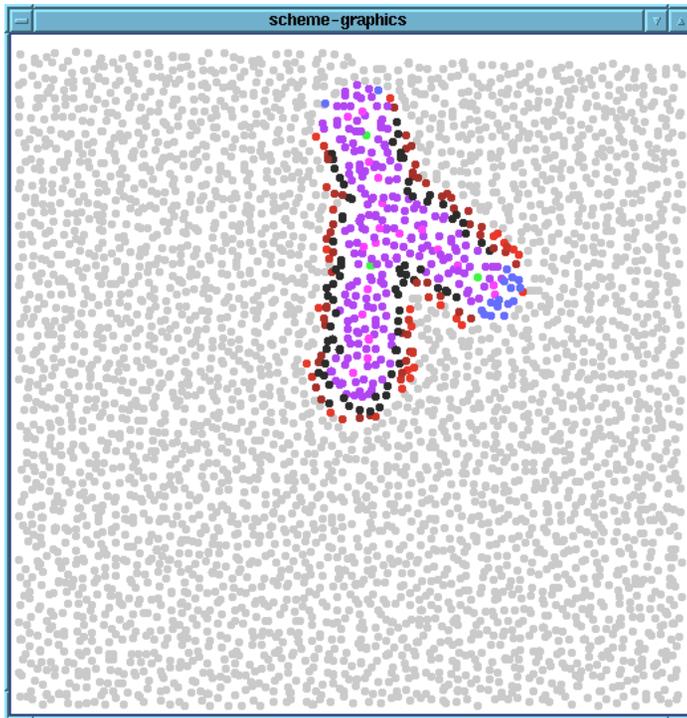
where  $H(N) = 1 + e^{-N} - \int_{-1}^1 e^{-\frac{N}{\pi}(\cos^{-1} t - t\sqrt{1-t^2})} dt$

Kleinrock and Silvester (1977)

# How can we program amorphous stuff?

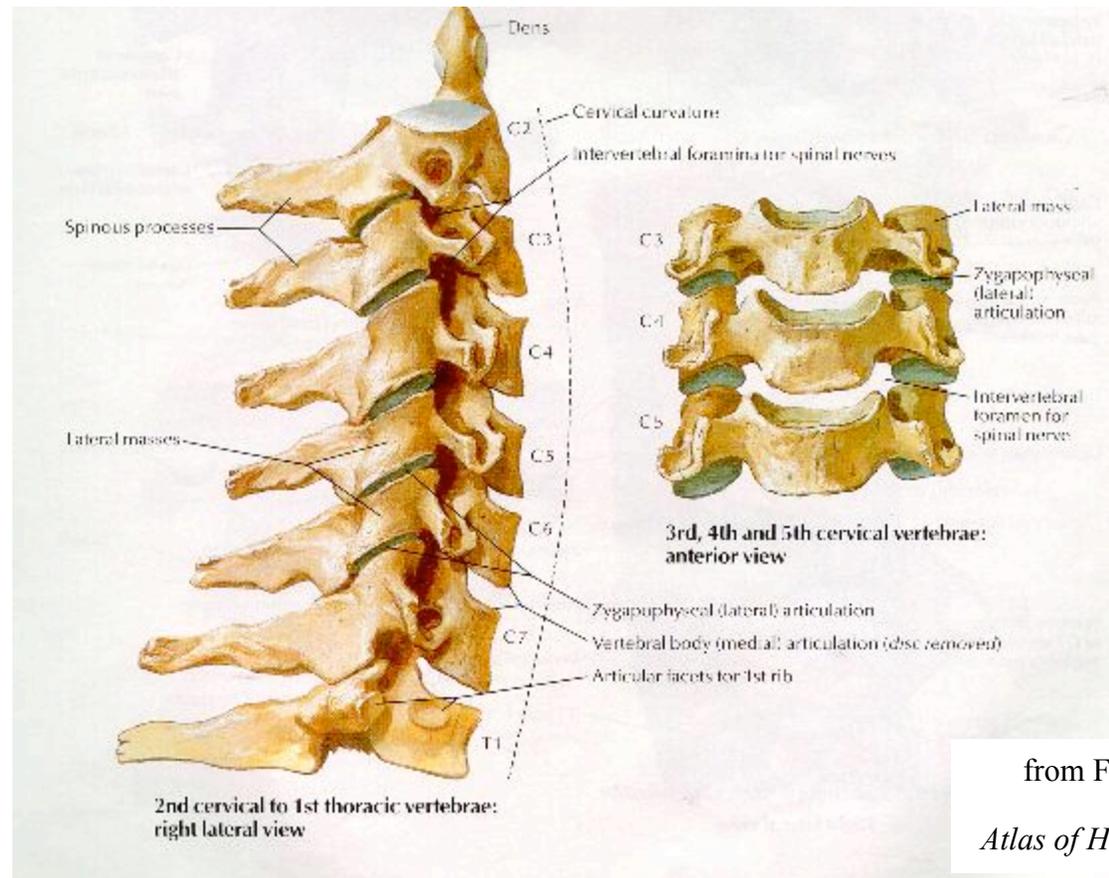
- We can look to biology for organizational metaphors, although we need not try to duplicate actual biological mechanisms
- We'll take examples from pattern formation to illustrate the point, and produce cartoon caricatures of biological morphogenesis

# Bifurcating Tubes: an example of emergent behavior

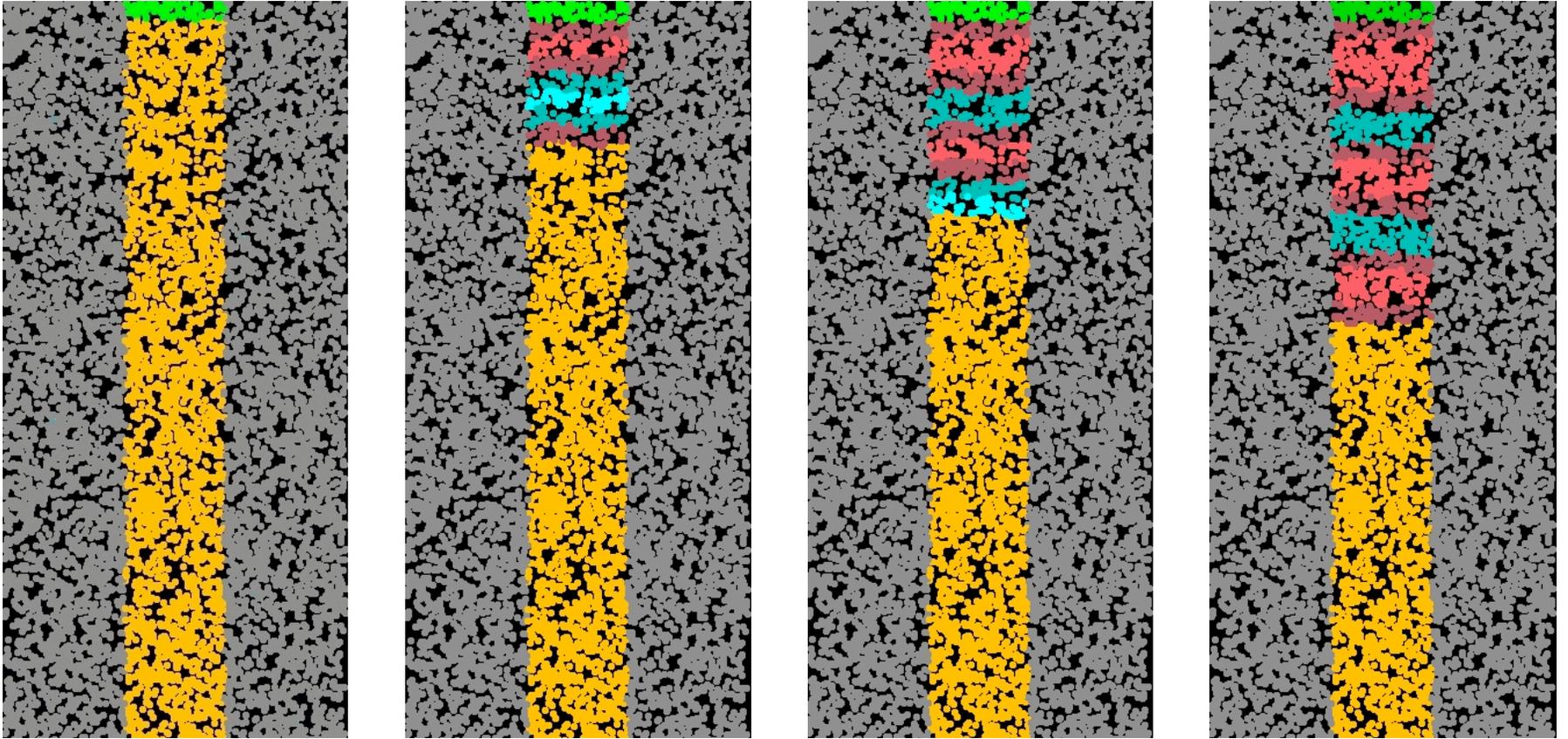


This is an amorphous physical simulation of a weak membrane bounding a pressure vessel. When a bulge appears, the membrane thins and the bulge expands. (by Radhika Nagpal)

Suppose we wanted to make something with a precisely specified geometry?



# Differentiation



To make a “spine”, the elements in an initial polarized tube must differentiate into bands of alternating C and D type segments.

## Local SIMD paradigm for programming differentiation and growth (Weiss)

- Each computing element's state includes some binary markers. Each computing element's program has many independent rules.
- Rules are triggered when messages are received. A rule is applicable if a certain boolean combination of markers is satisfied.
- When a rule is applied it may set markers and send further messages.
- Messages have hop counts that determine how far they will diffuse.
- Markers may have lifetimes after which they expire.

# Microbial Colony Language (MCL): Ron Weiss

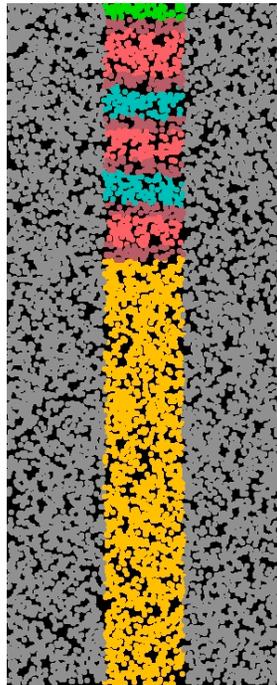
## *A program for creating segments:*

```
(start
  Crest
  ((send (make-seg C 1) 3))

((make-seg seg-type seg-index)
 (and Tube (not C) (not D))
 ((set seg-type)
  (set seg-index)
  (send created 3)))

((make-seg) (= 0))
Tube
((set Bottom))

((make-seg) (> 0))
Tube
((unset Bottom))
```



```
(created } message
(or C D) } condition
((set Waiting 10)) } actions

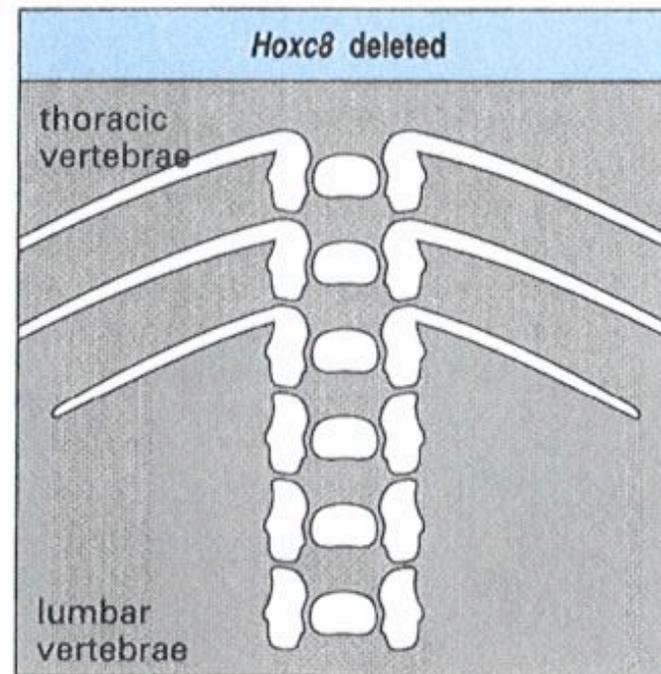
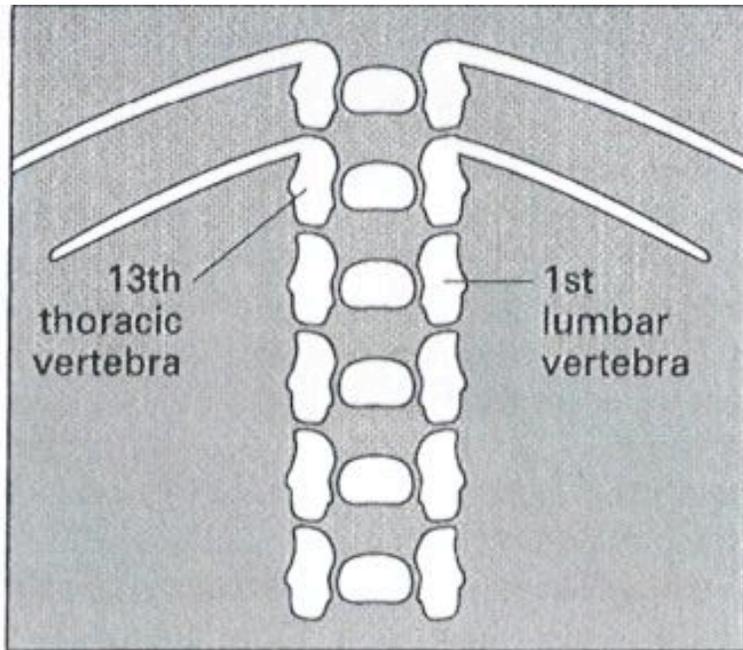
(*
 (and Bottom C 1 (Waiting (= 0)))
 ((send (make-seg D 1) 3))

(*
 (and Bottom D 1 (Waiting (= 0)))
 ((send (make-seg C 2) 3))

(*
 (and Bottom C 2 (Waiting (= 0)))
 ((send (make-seg D 2) 3))

(*
 (and Bottom D 2 (Waiting (= 0)))
 ((send (make-seg C 3) 3))
```

# Deleting Hox genes in mice



from Wolpert, *Principles of Development*,  
Oxford University Press, 2002, p. 124

# A botanical metaphor for pattern generation: (Daniel Coore)

Organize the process in terms of “growing points.”

Growing points are abstract structures that exhibit “tropisms” toward particular “chemical gradients.”

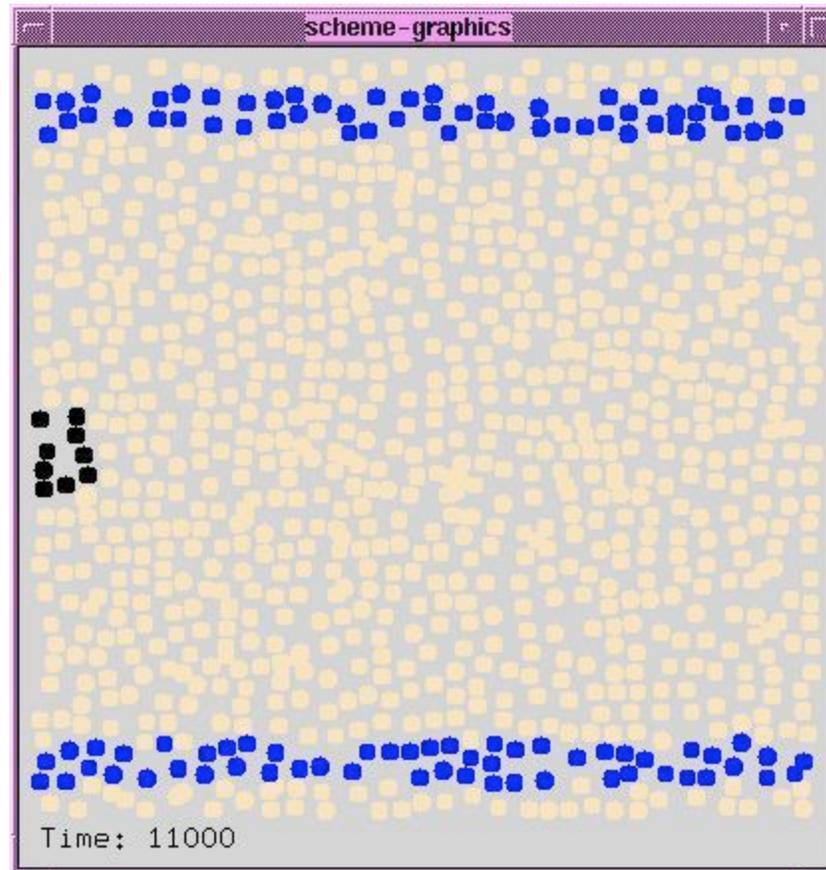
The growing points may lay down materials.

Materials may secrete pheromones that attract or repel other growing points.

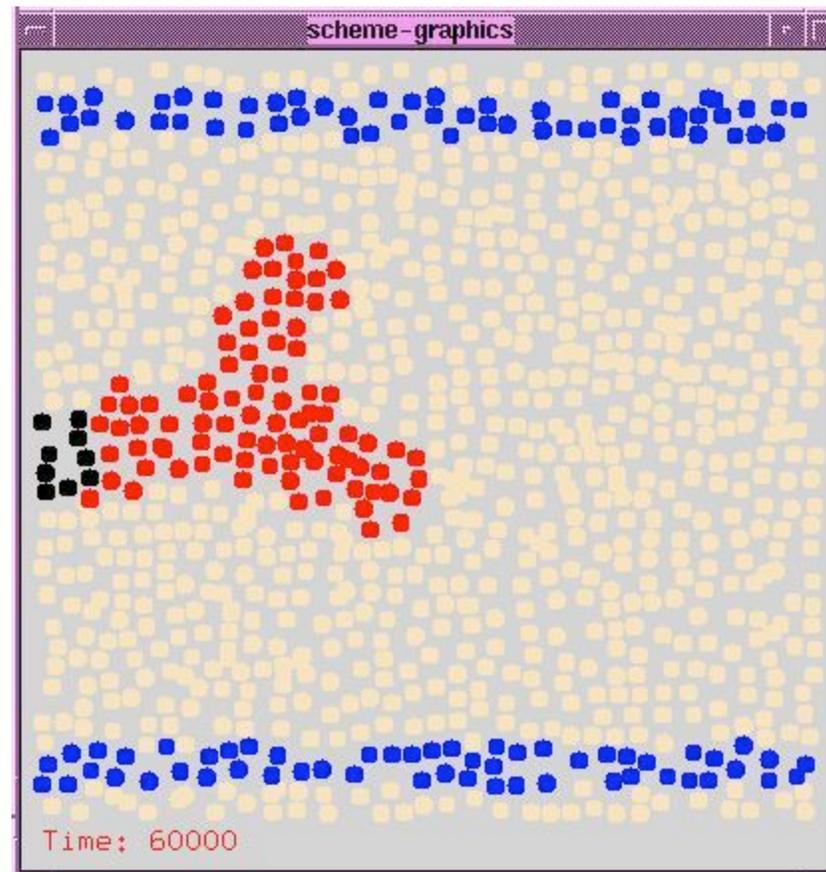
Growing points may split, die off, or join.

Support for this abstraction may be programmed as a uniform state machine in each computational particle.

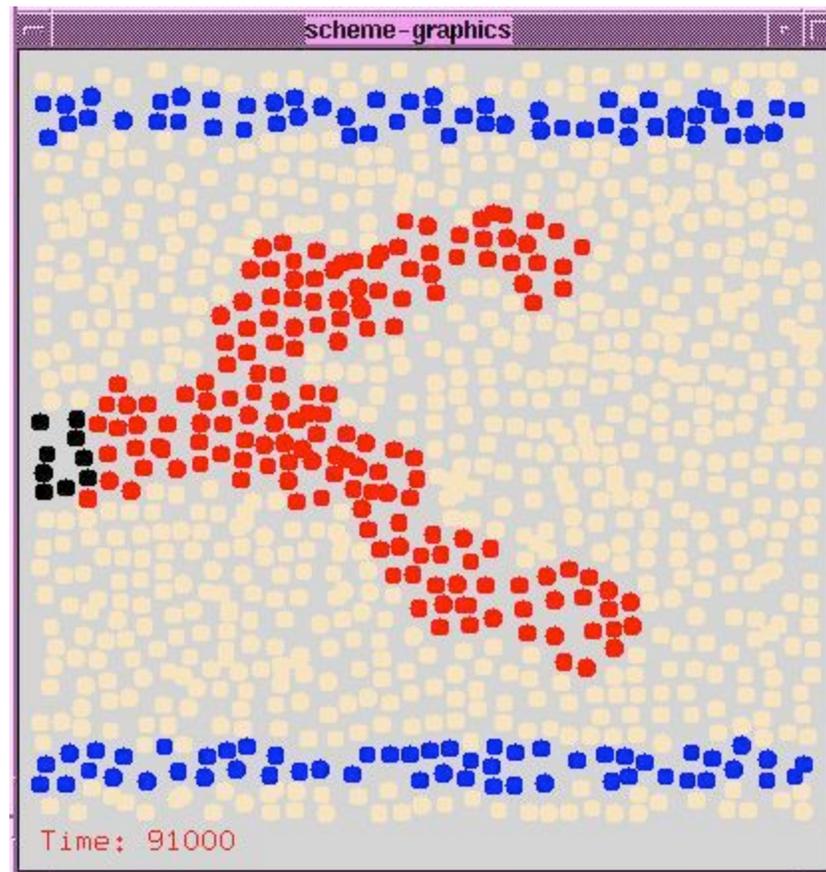
Start with “Vdd”, “Vss”, and a “Poly”  
Contact



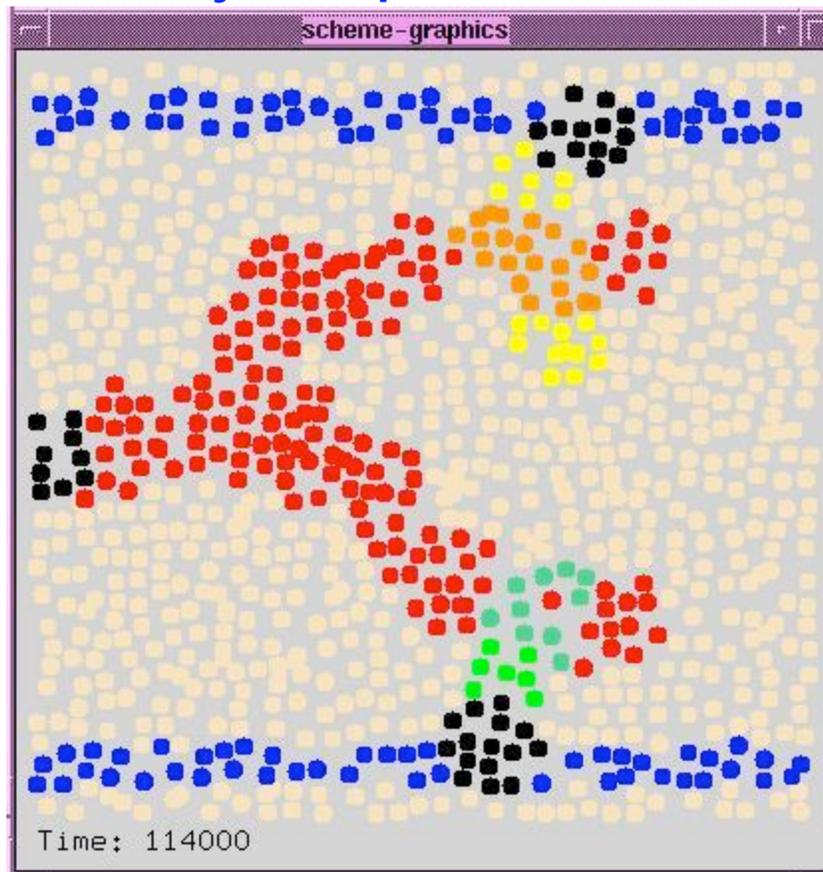
The poly contact sprouts a growing point that bifurcates and then grows toward the pheromones secreted by Vdd and Vss.



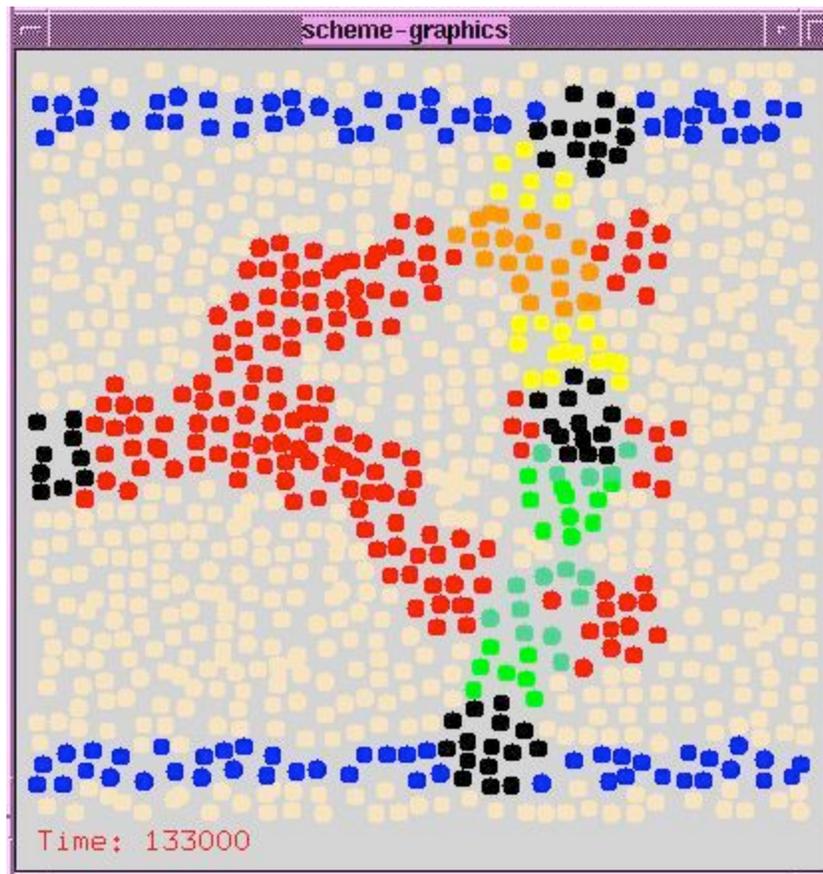
When the growing poly gets close to  $V_{dd}$  and  $V_{ss}$  it is stopped by a short-range inhibition



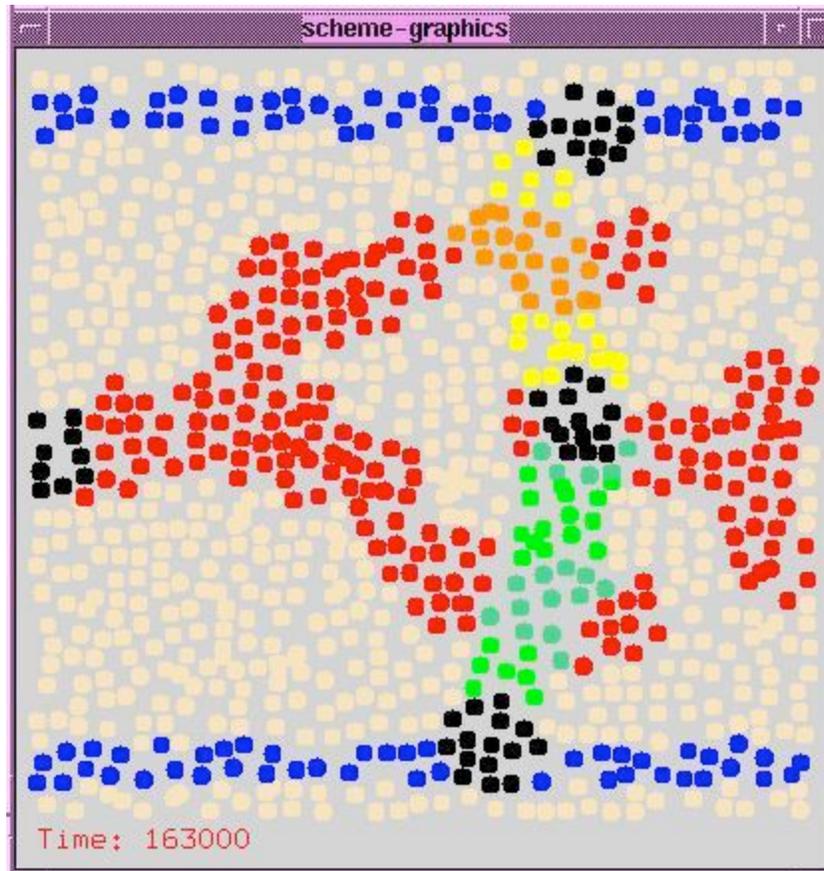
The poly growing points die off, but first they sprout P and N transistor diffusion growing points, which grow toward Vdd and Vss, where they drop contacts.



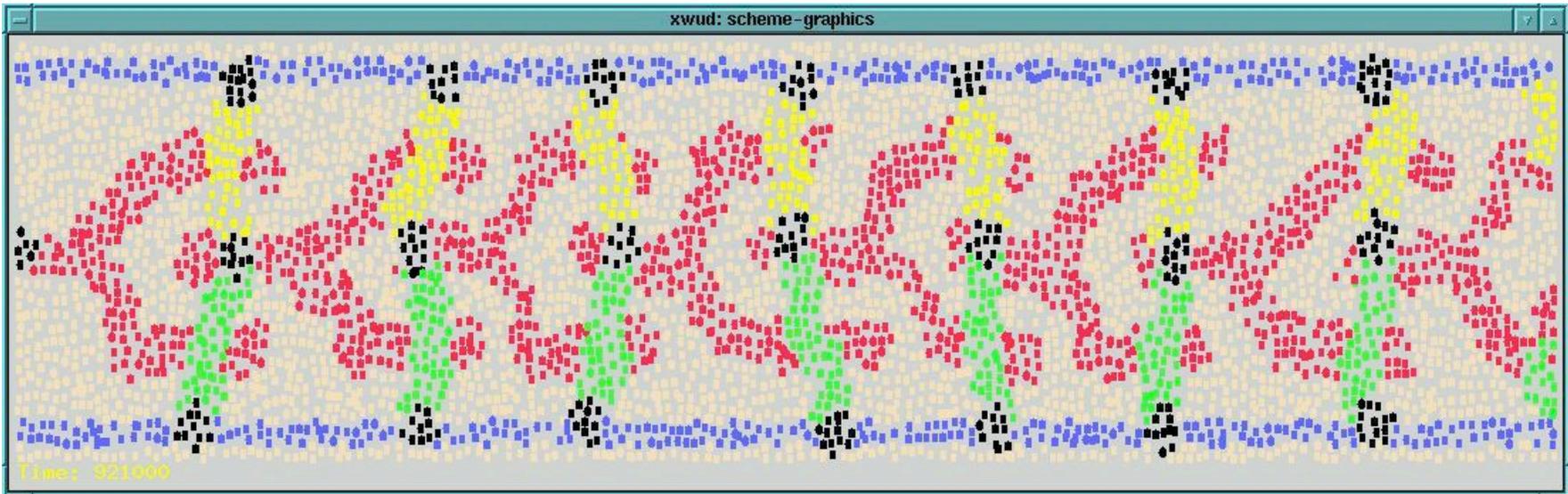
The diffusions also grow toward each other.  
When they hit they form a new poly contact and  
poly growing point.



The process then repeats, growing the next inverter.



This process repeats to make an arbitrarily long chain of ugly, but topologically correct inverters.

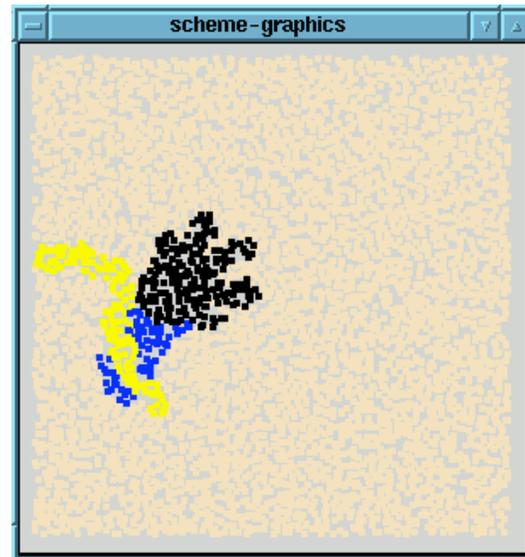
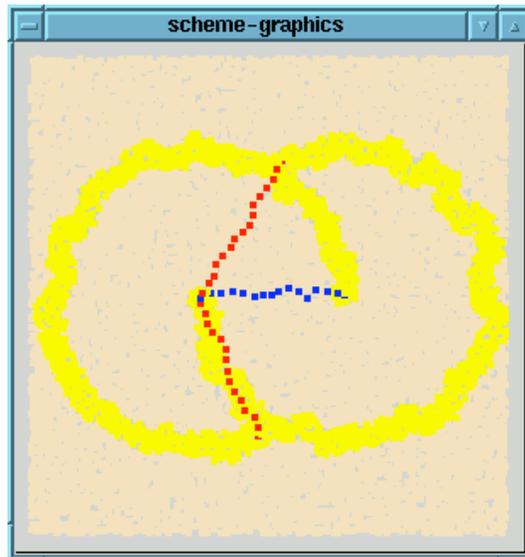
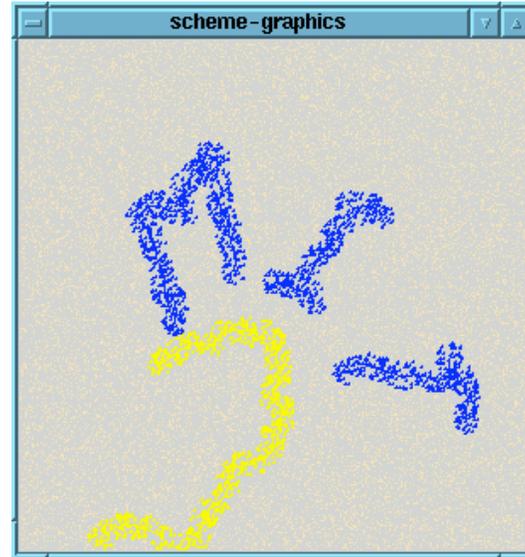
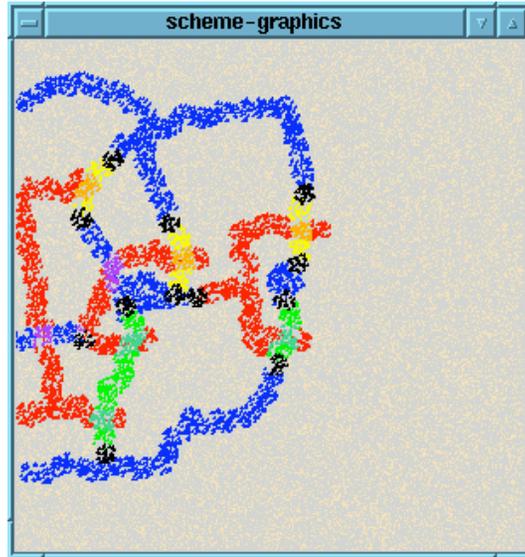


This demonstrates totally local control of precision topology.

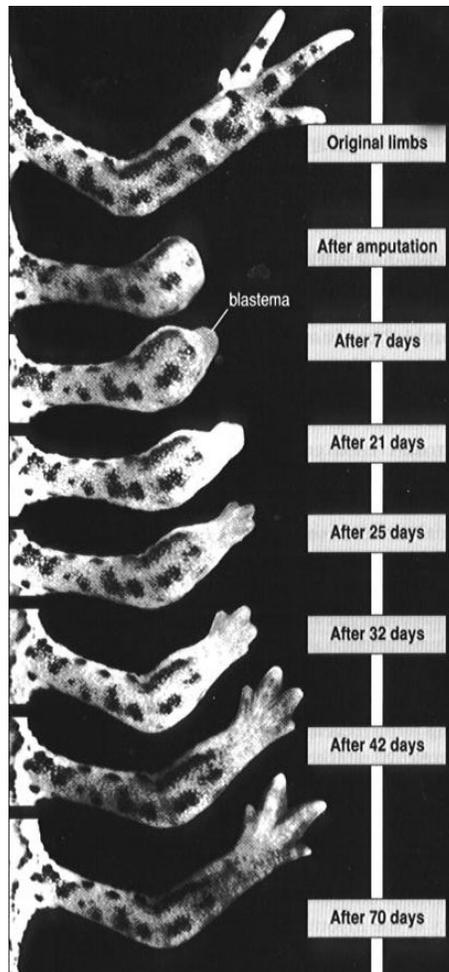
The growing points provide a serial locus of control, even though the implementation is in terms of a uniform state machine in each computational particle

```
(define-growing-point ((poly from-input-contact) Q-id)
  (material poly)
  (tropism constant Vdd-long Vss-long)
  (initialize lifetime 5)
  (secrete (poly-short inhibitor) Q-id)
  (when (= lifetime 0)
    (start-growing-point (poly up) Q-id)
    (start-growing-point (poly down) Q-id)
    (terminate)))
```

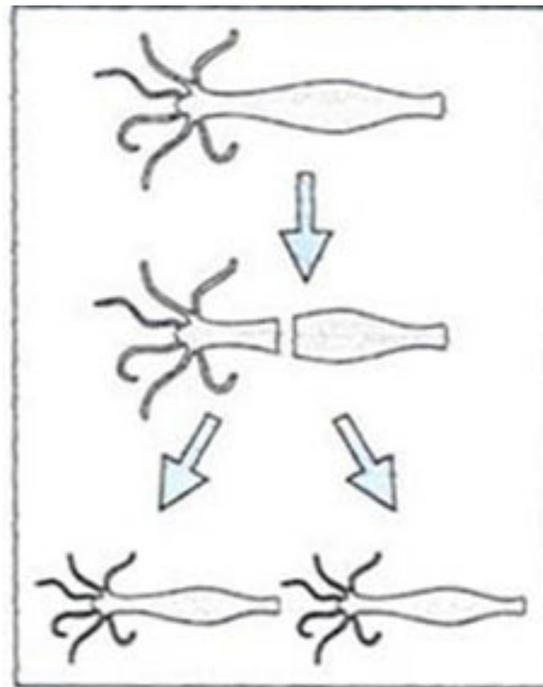
# Capabilities of GPL



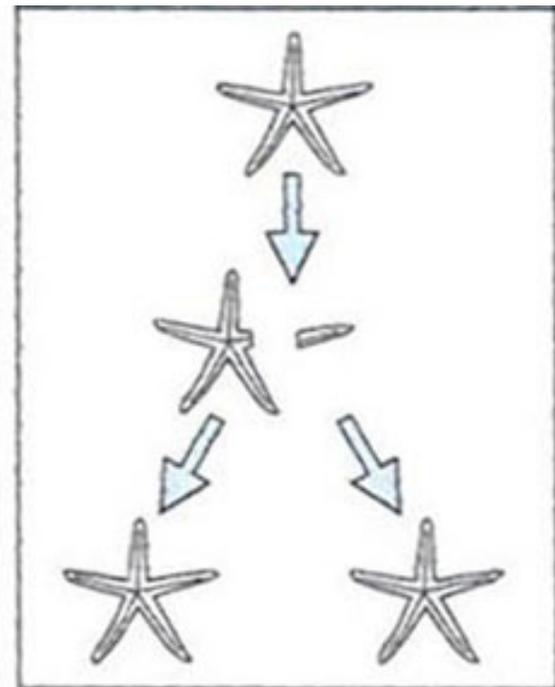
# Robustness and regeneration



Newt



Hydra

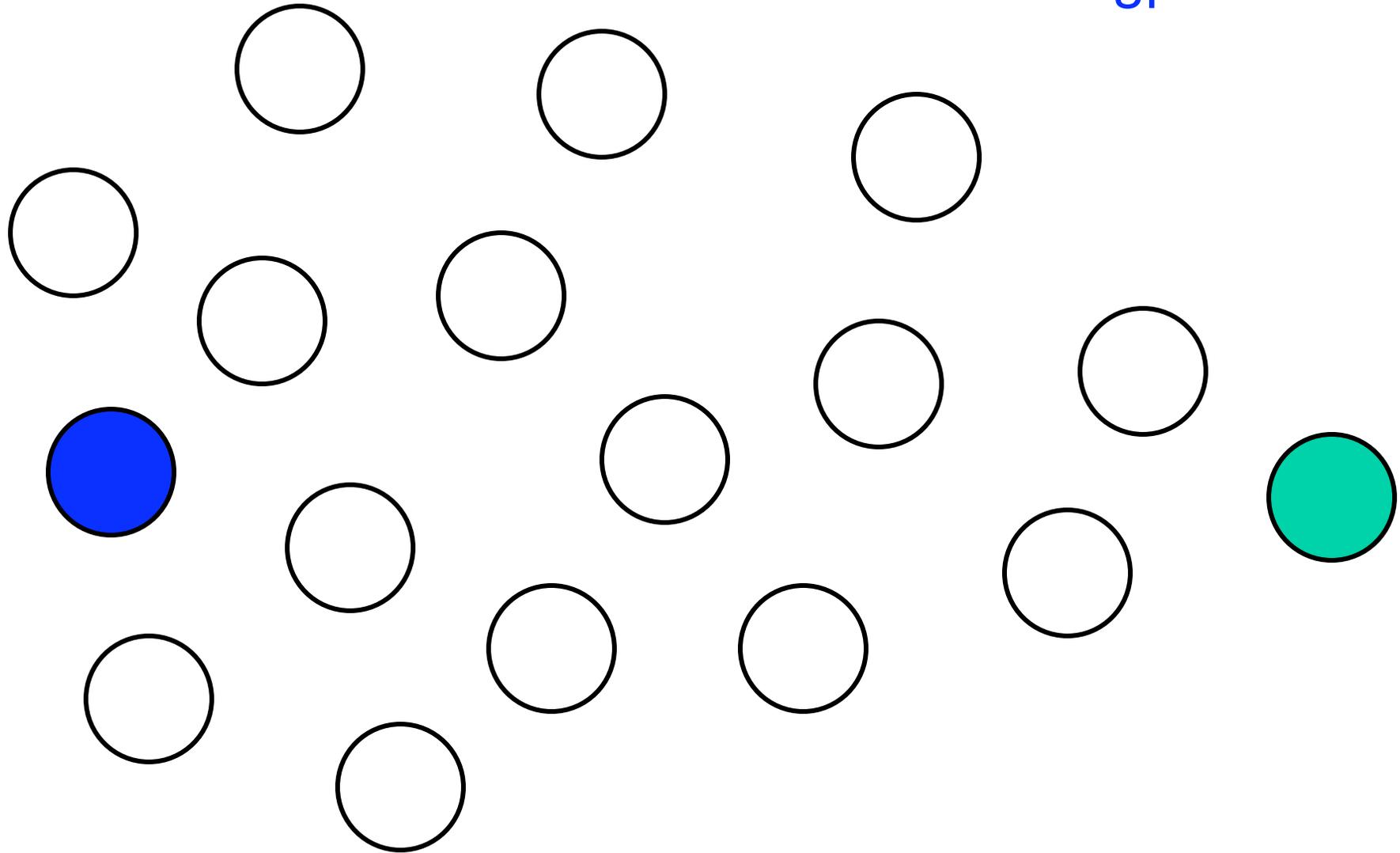


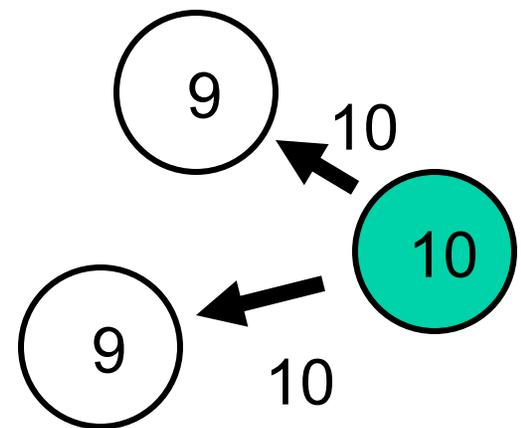
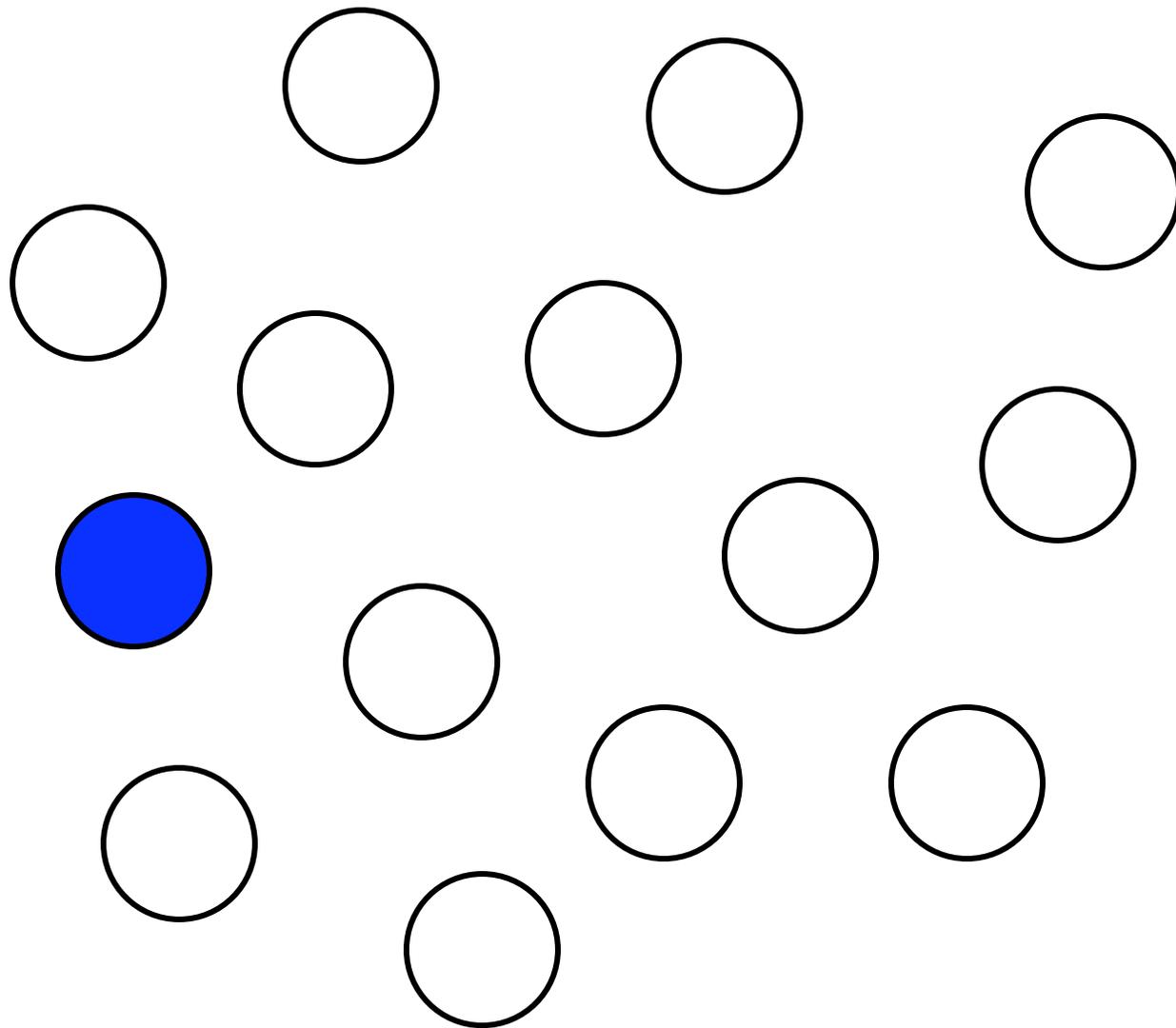
Starfish

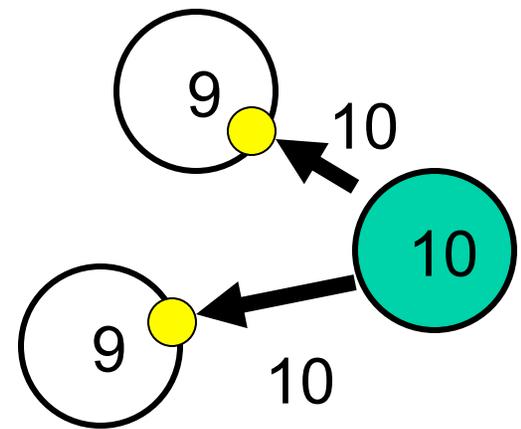
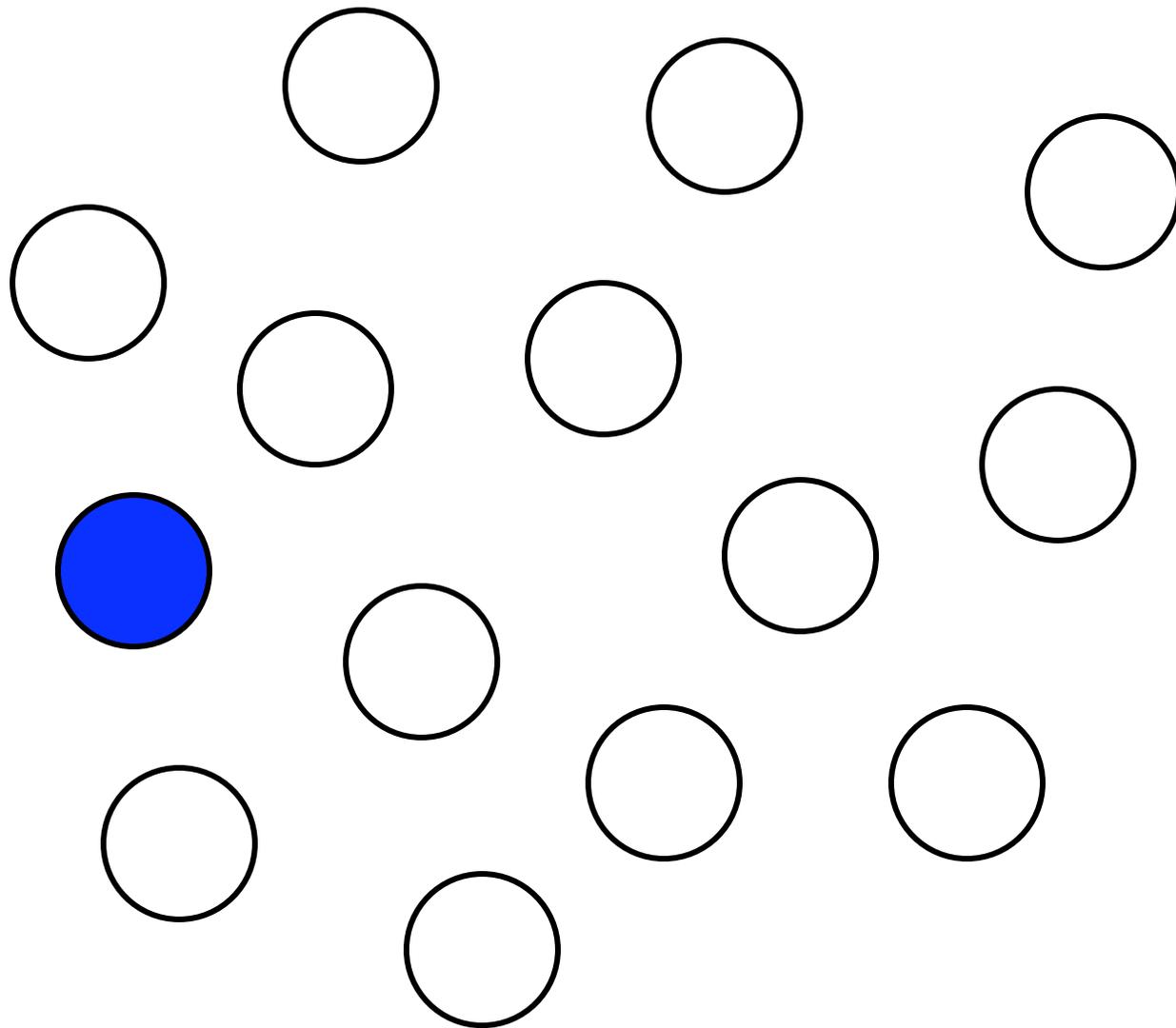
from Wolpert, *Principles of Development*,  
Oxford University Press, 2002, pp. 447, 450

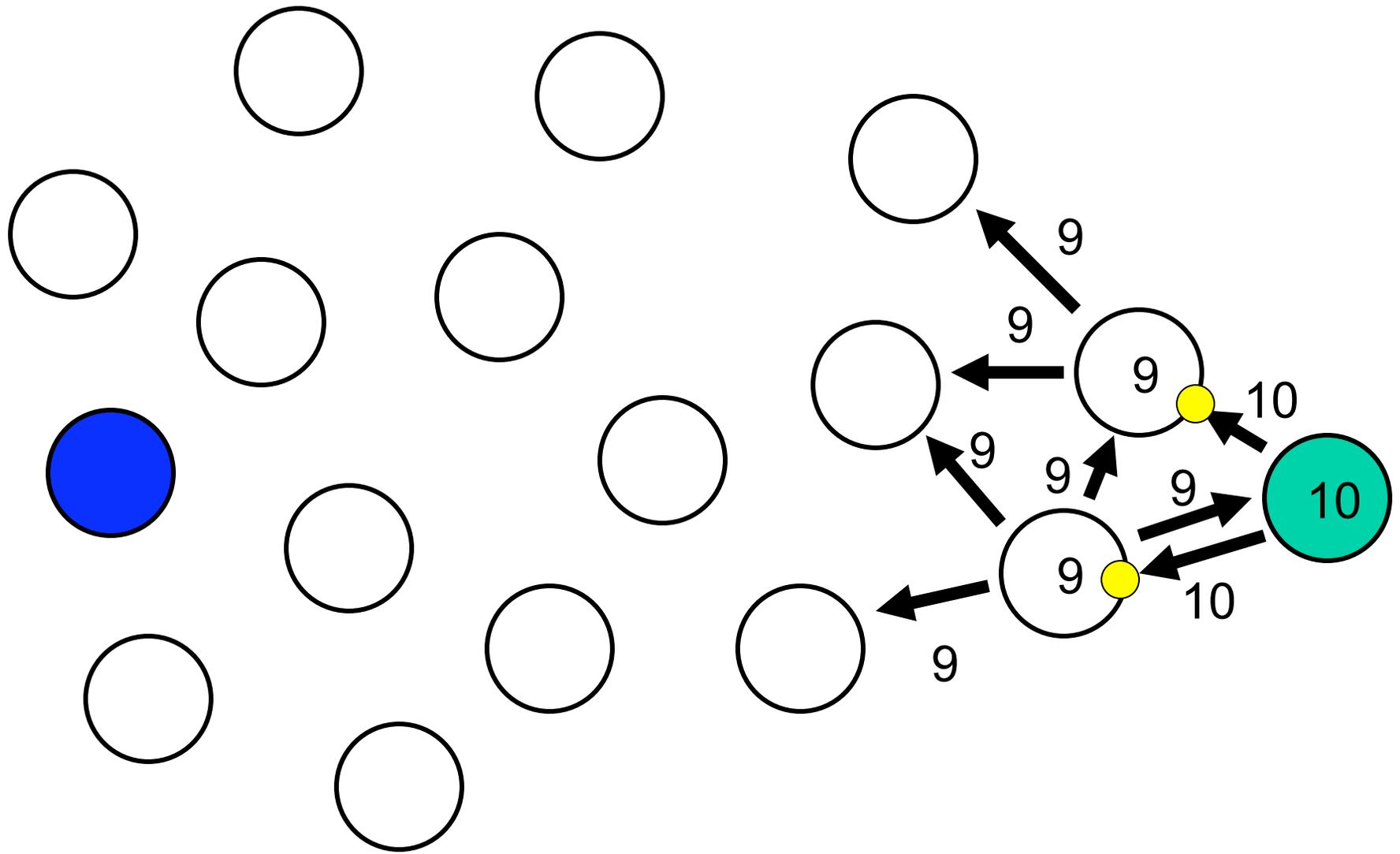
# Active gradients and self-repairing lines

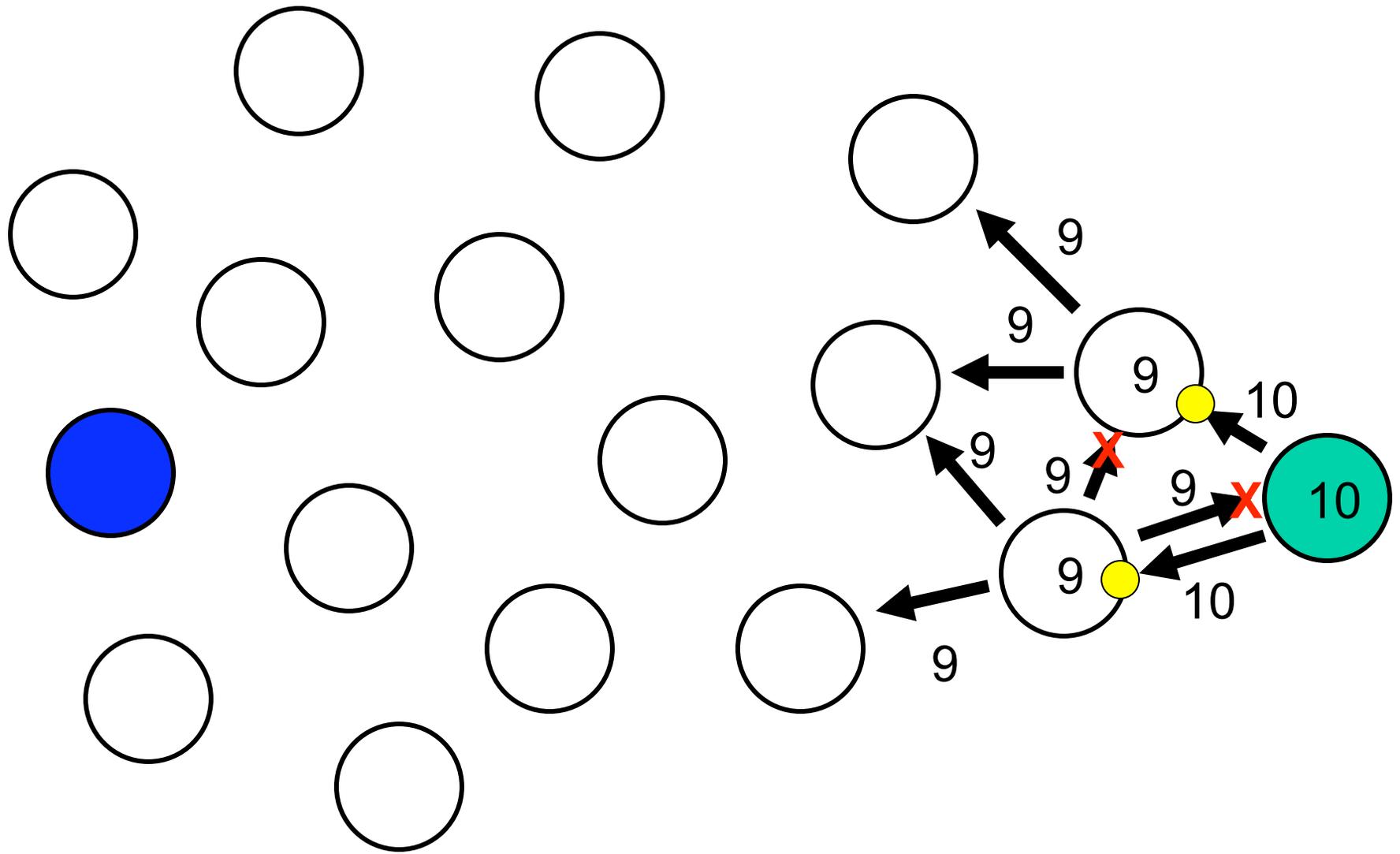
Lauren Clement and Radhika Nagpal

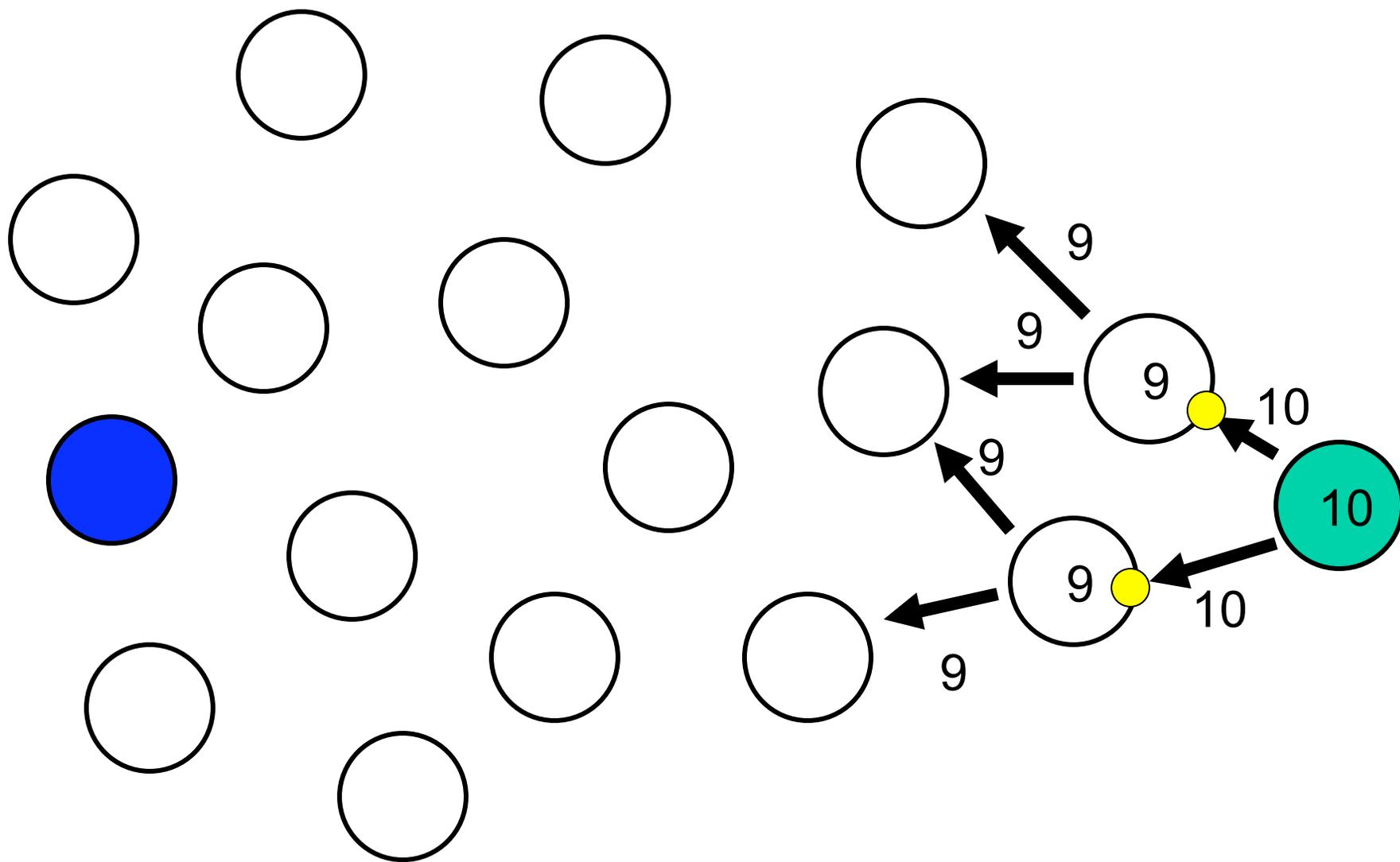


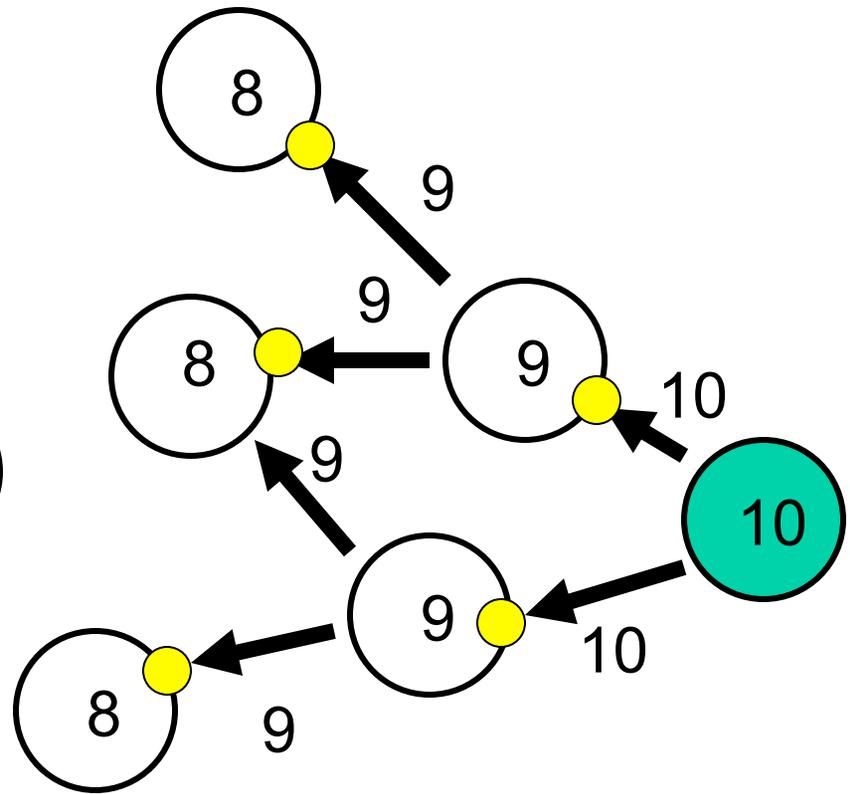
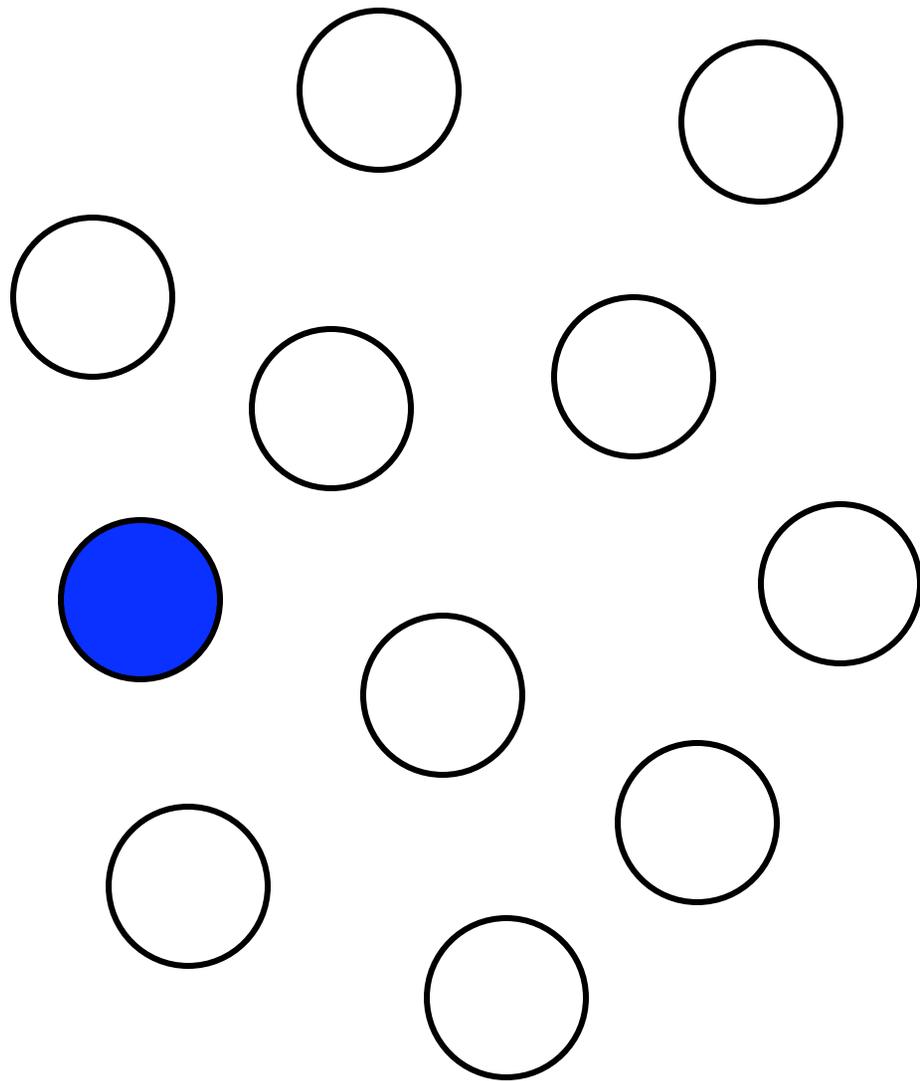


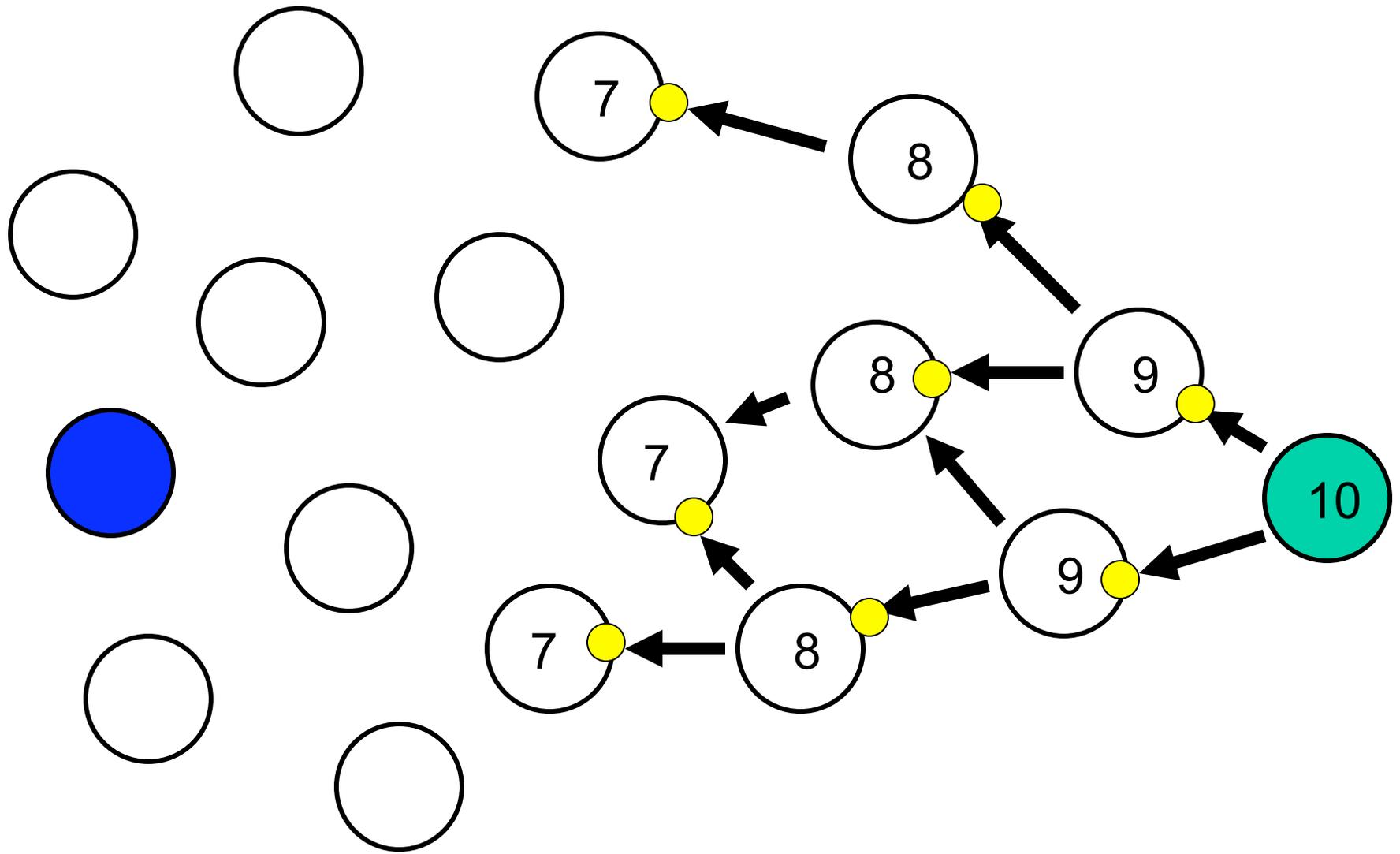


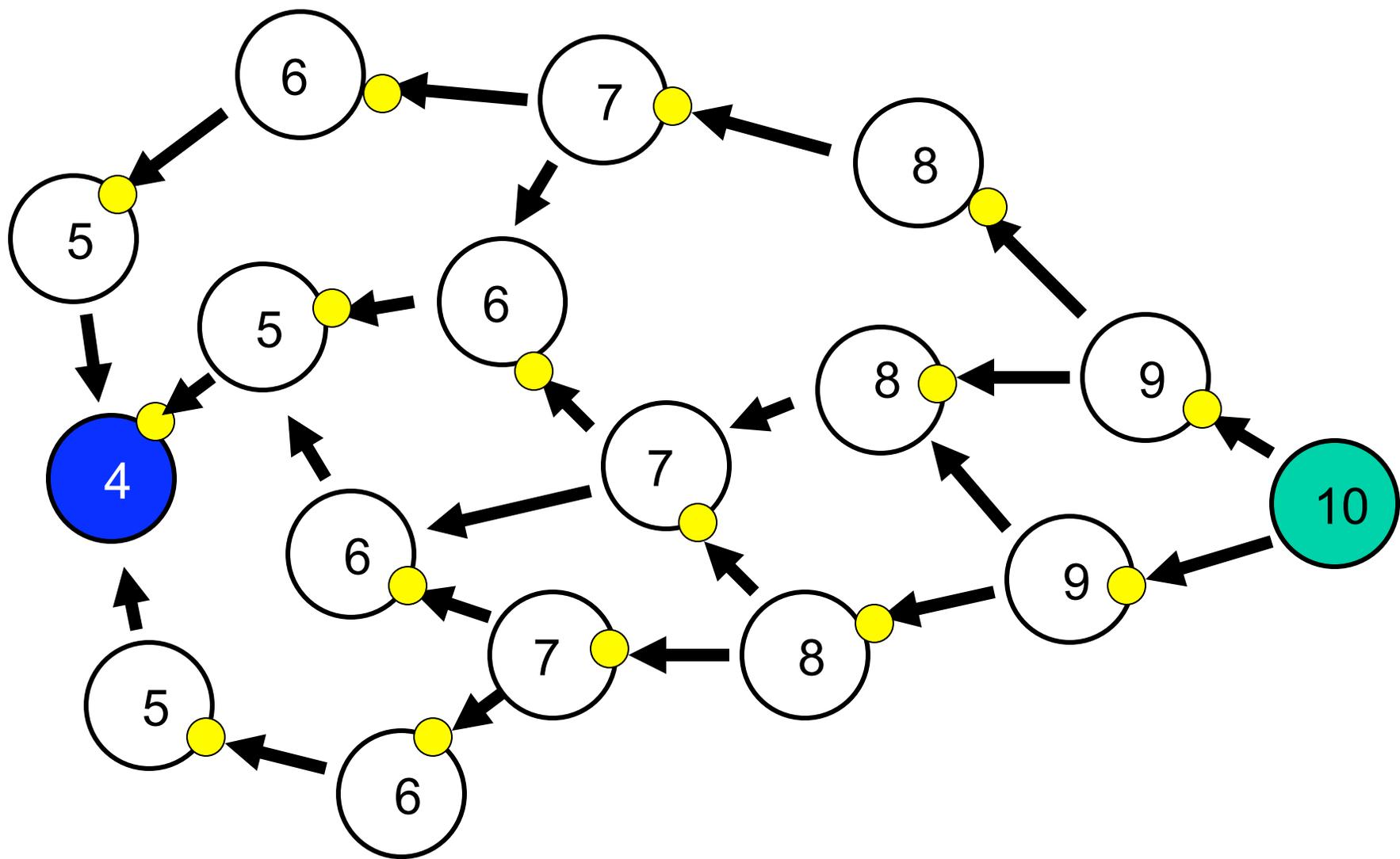


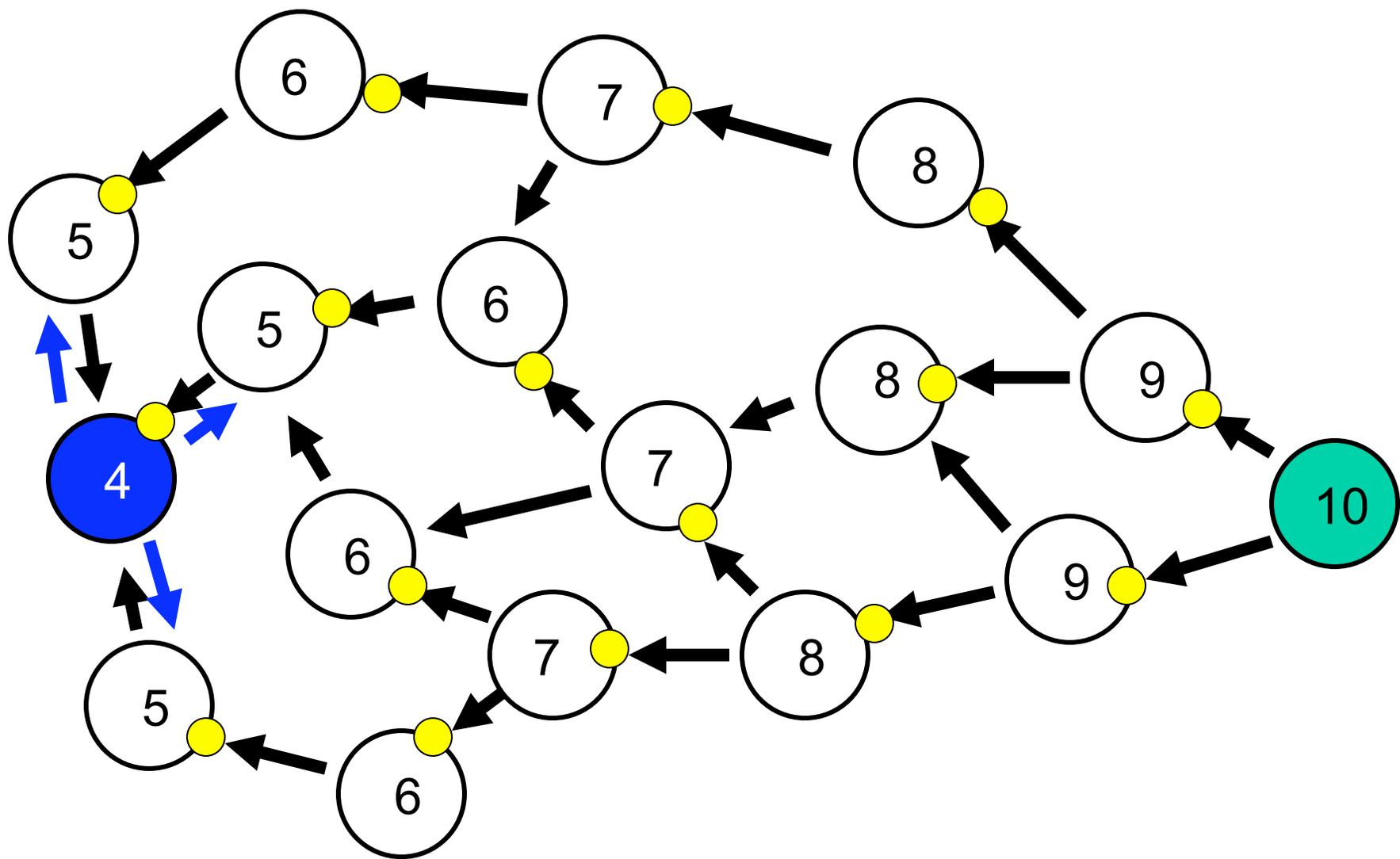


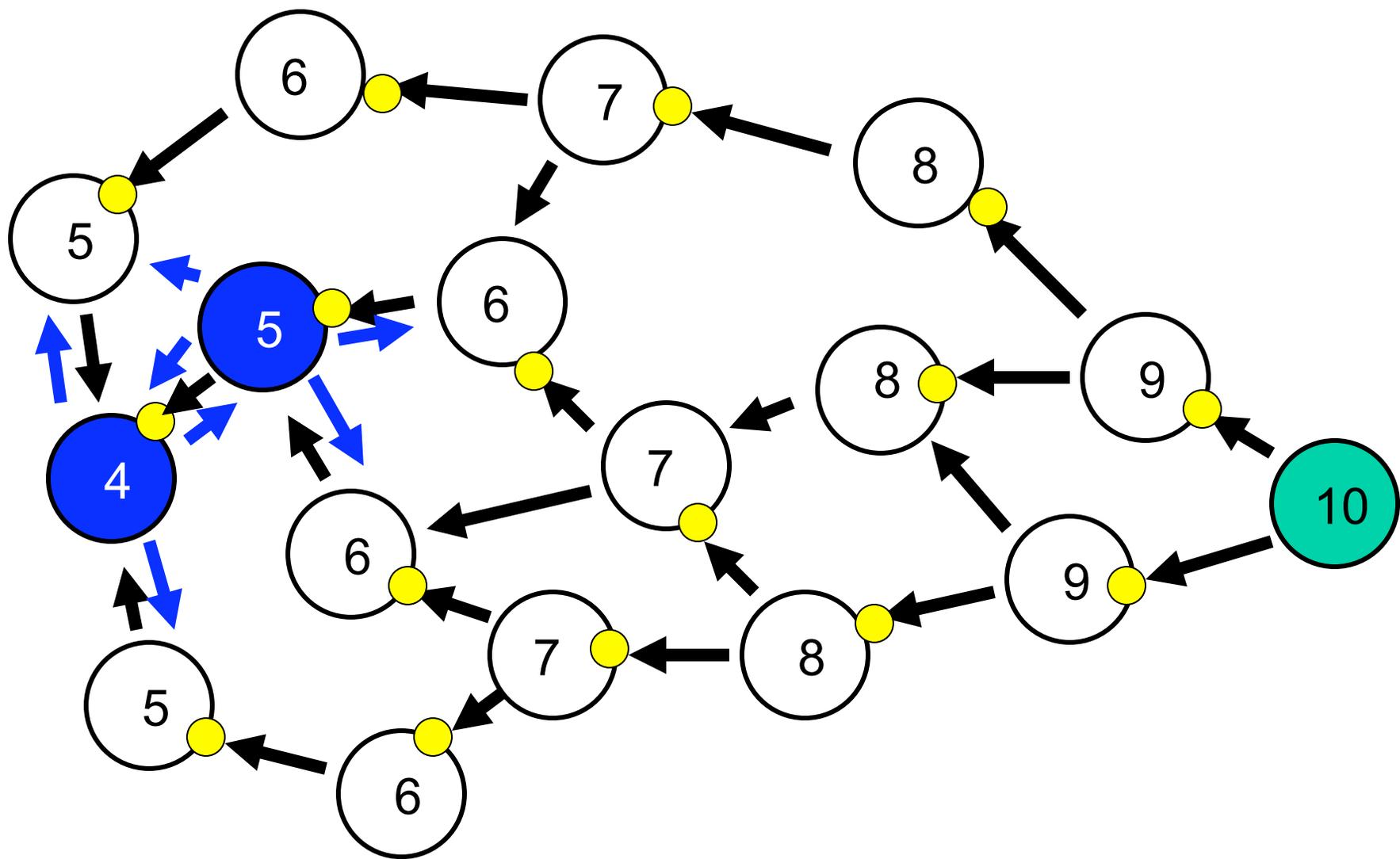


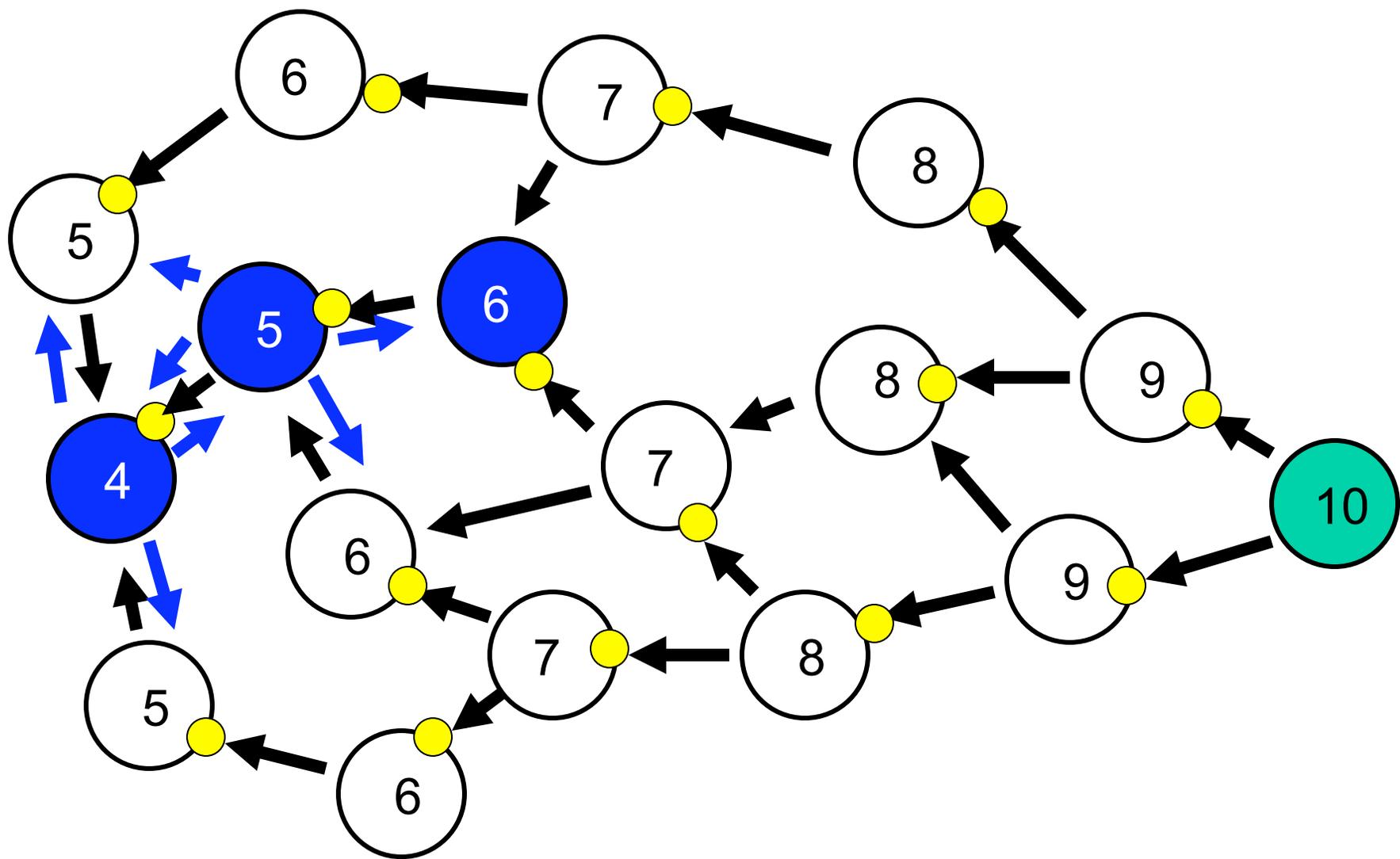




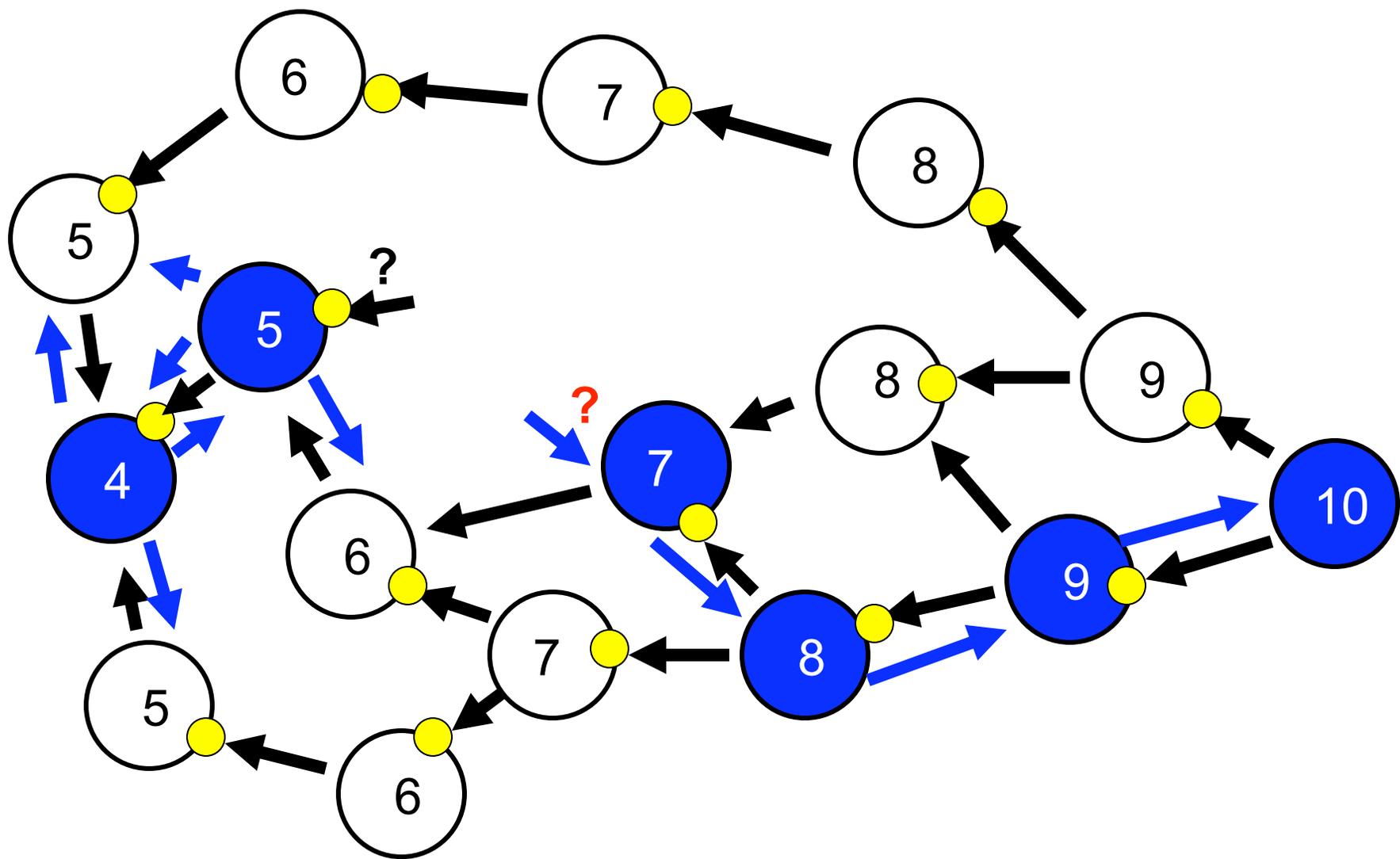




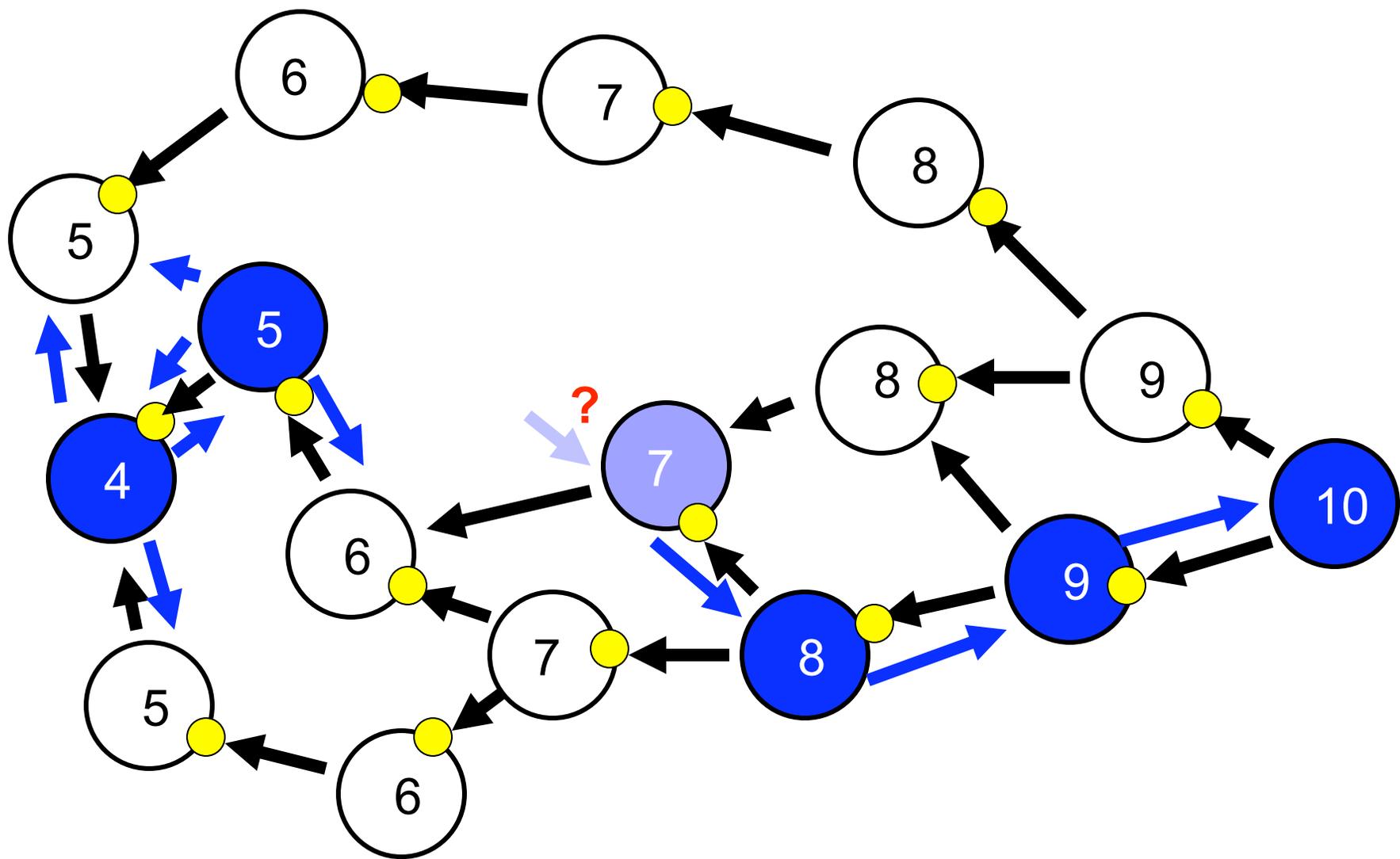


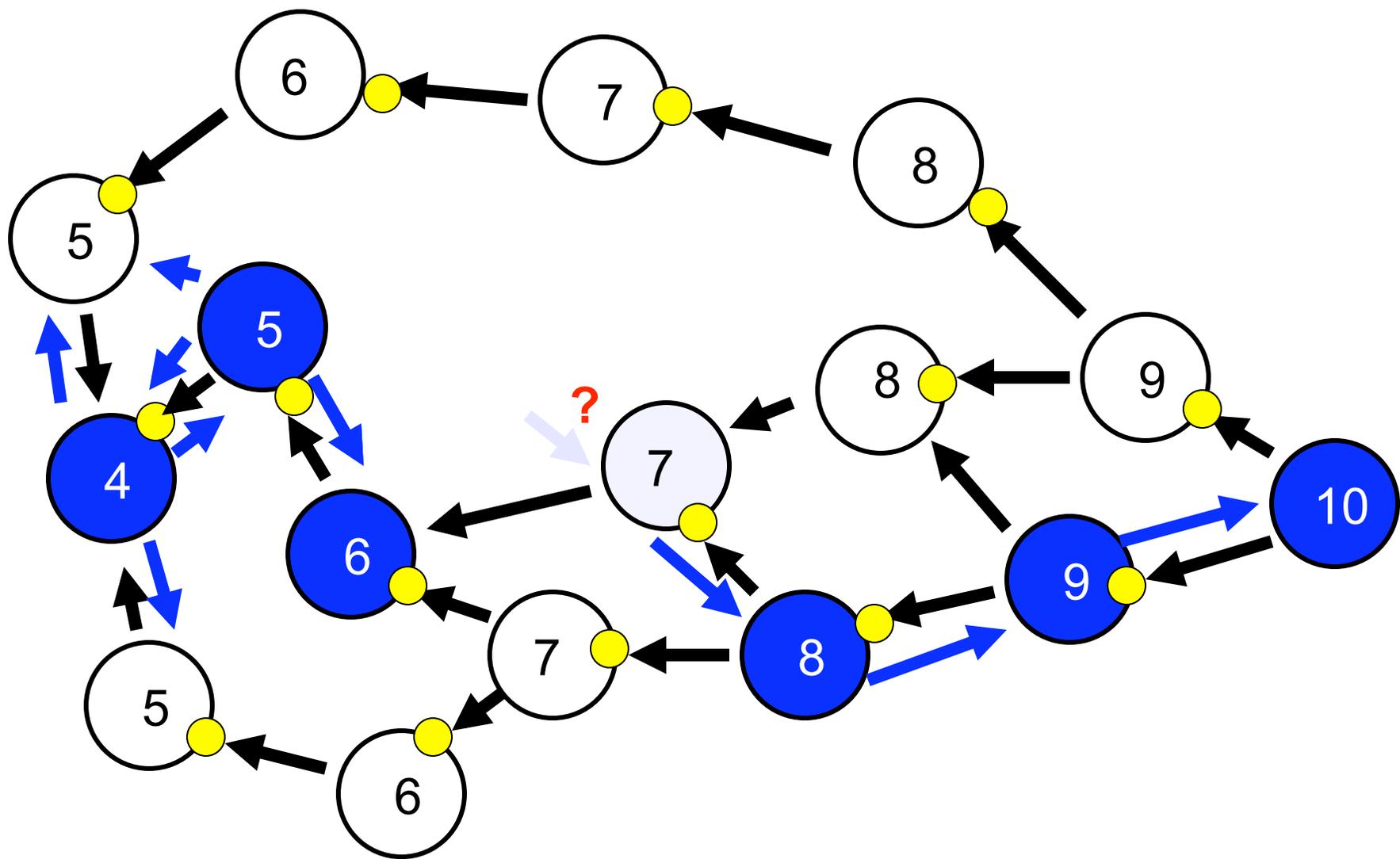




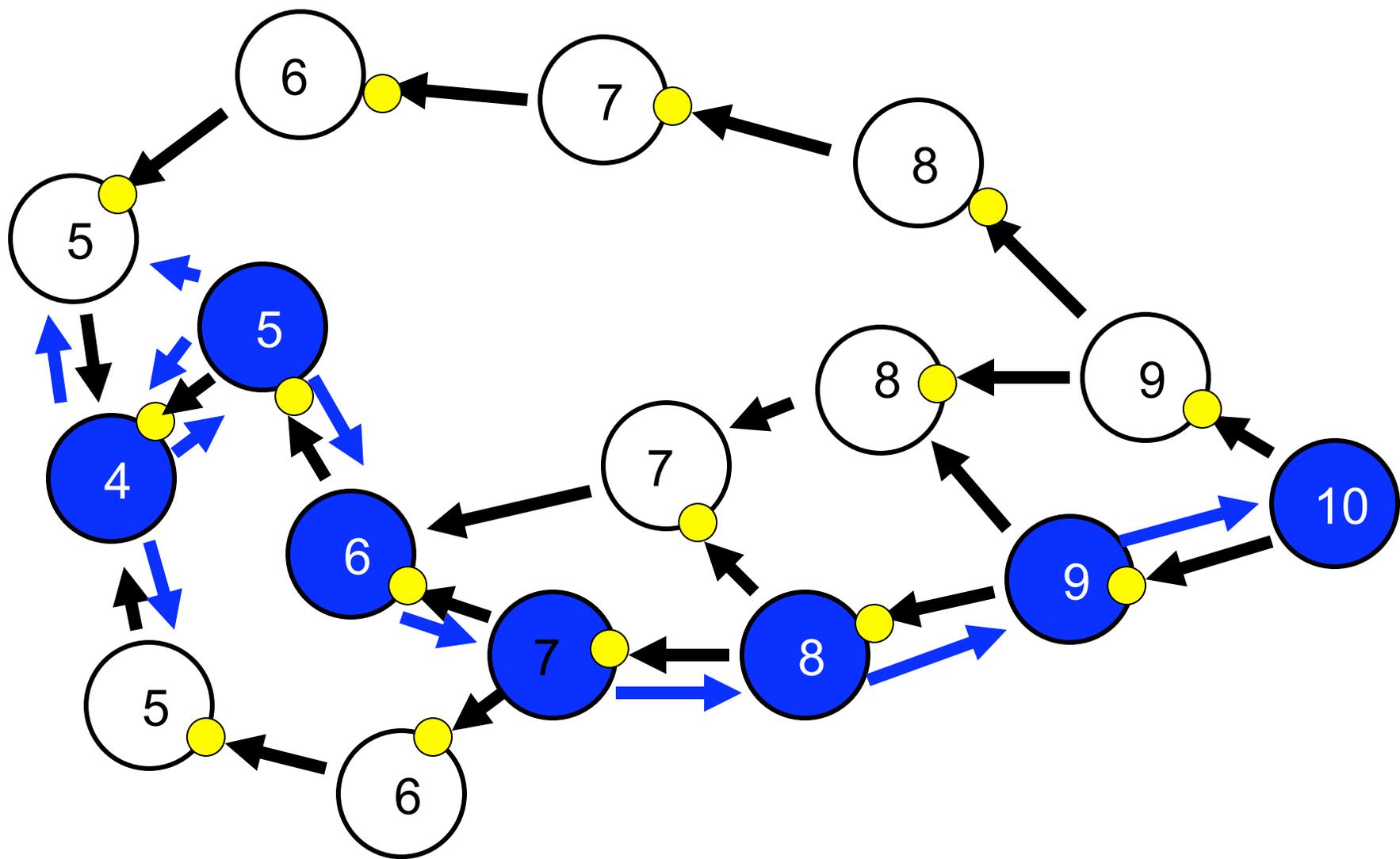




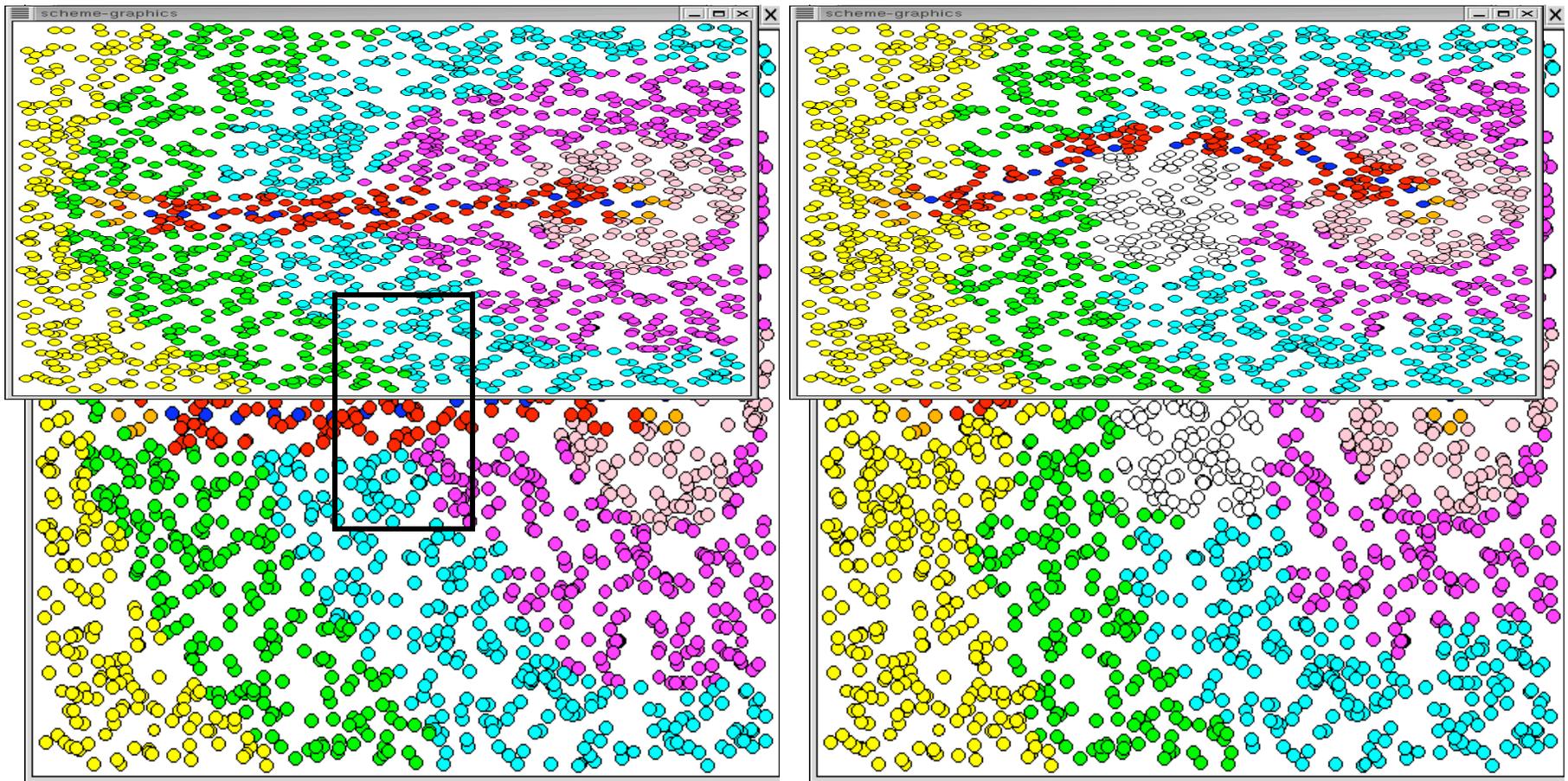




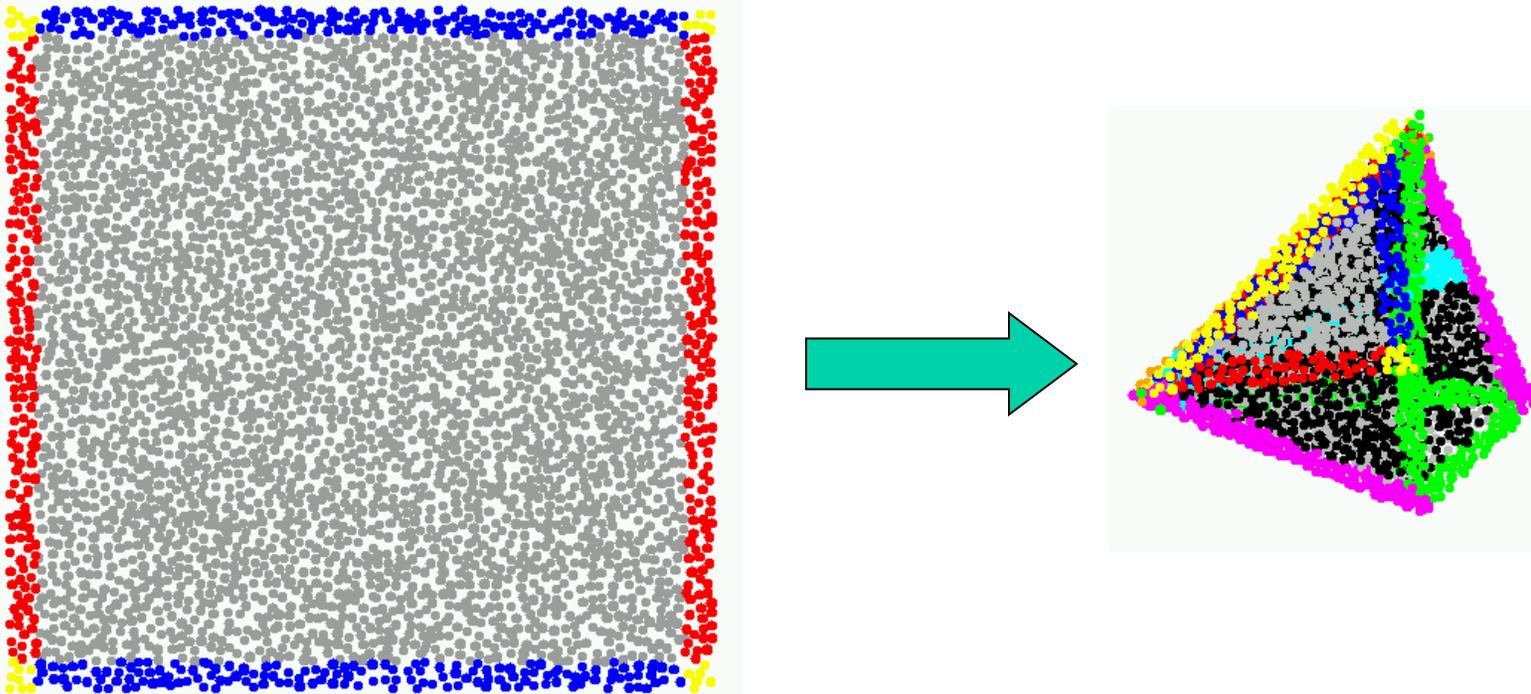




# Self-repairing line

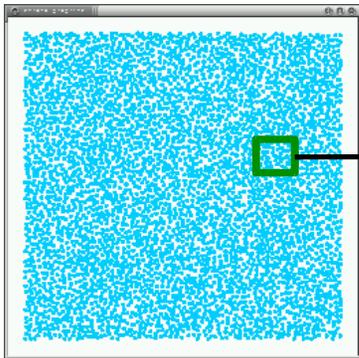


# Languages for programmable materials (Nagpal, 2001)

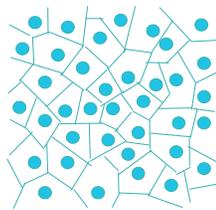


An amorphous sheet of programmable cells folds itself to form prespecified shapes

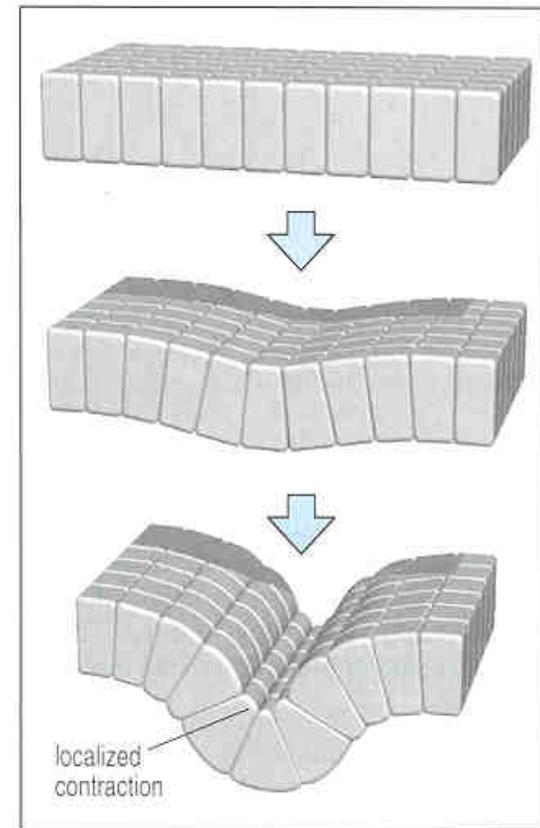
# Programmable Cell sheet



Randomly and densely distributed cells

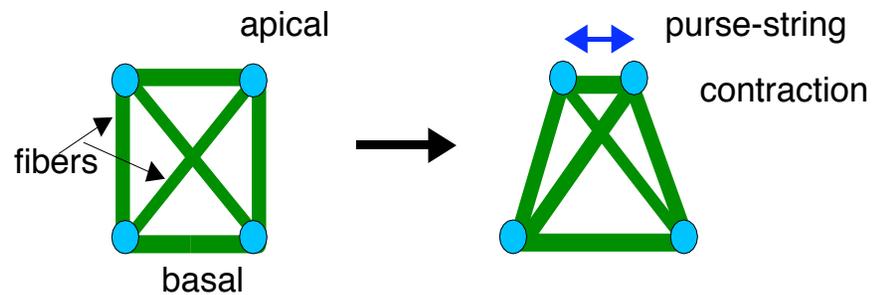
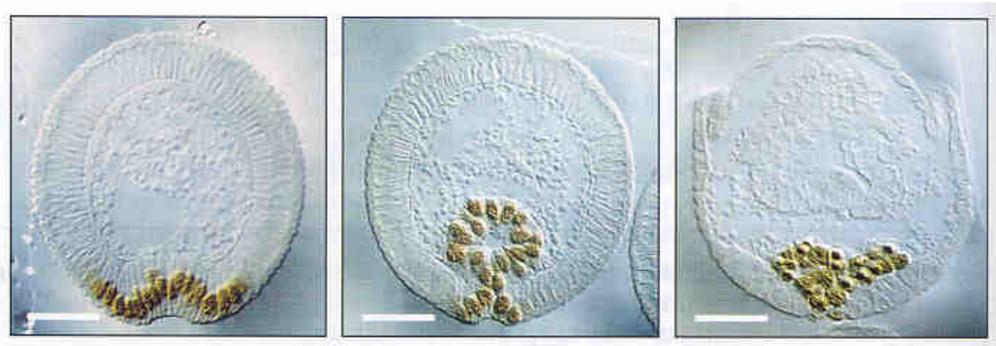


Irregularly shaped cells



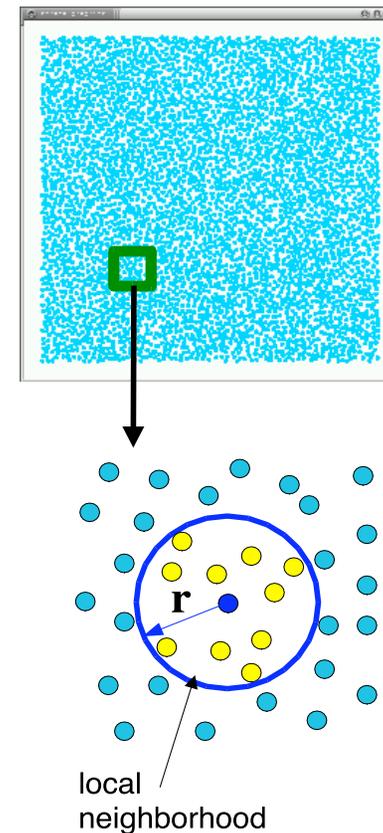
# Inspiration: Epithelial cell tissues

## Cell Model by Odell et al.



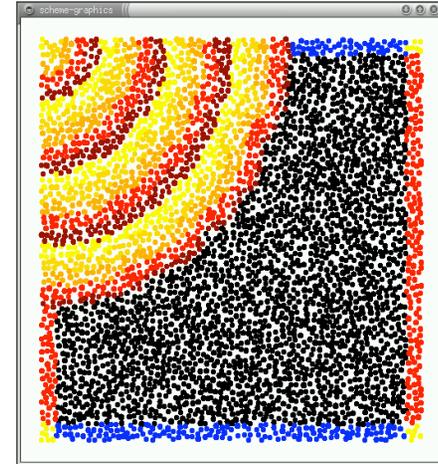
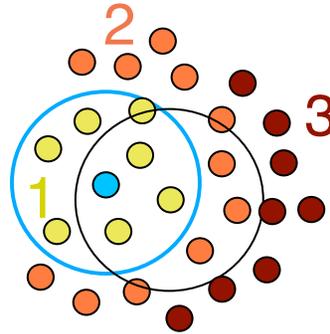
# Programmable cells

- Cell computation model
  - Autonomous
  - Identical program
  - Local communication
  - Local sensing, actuation
  - Limited resources, no global identifiers
  - No global coordinates
  - No global clock

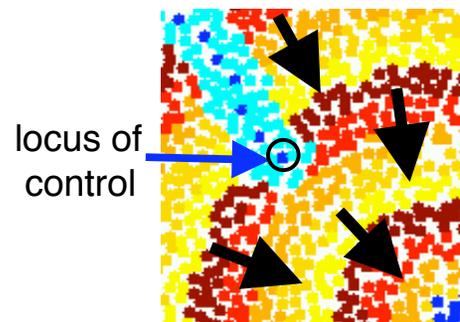


# Biologically-inspired primitives

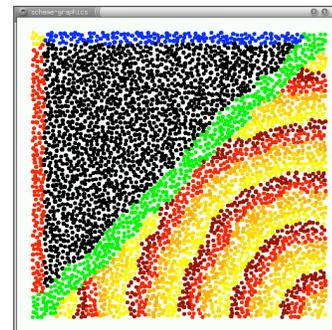
## Gradients



## Tropism

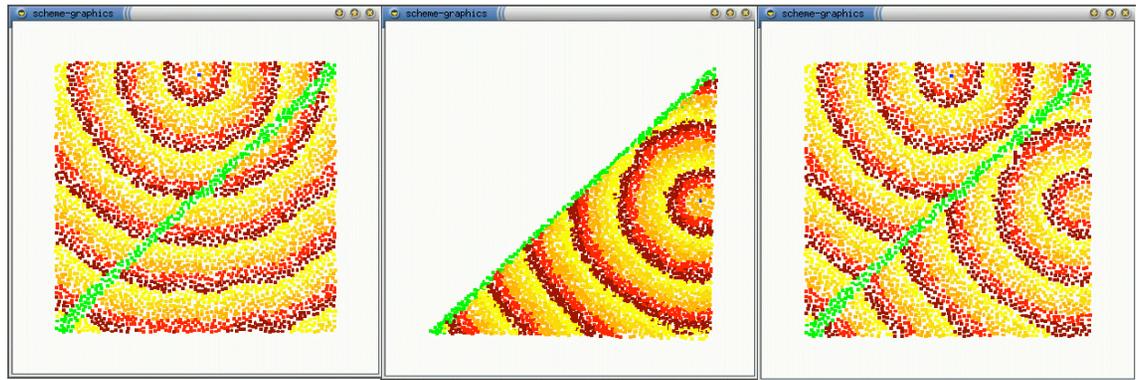
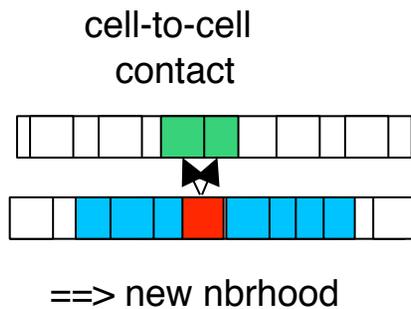


## Bounded

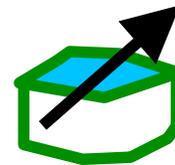


# More biologically-inspired primitives

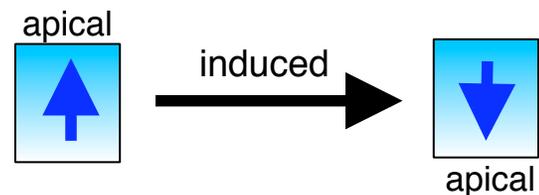
- Seepthru



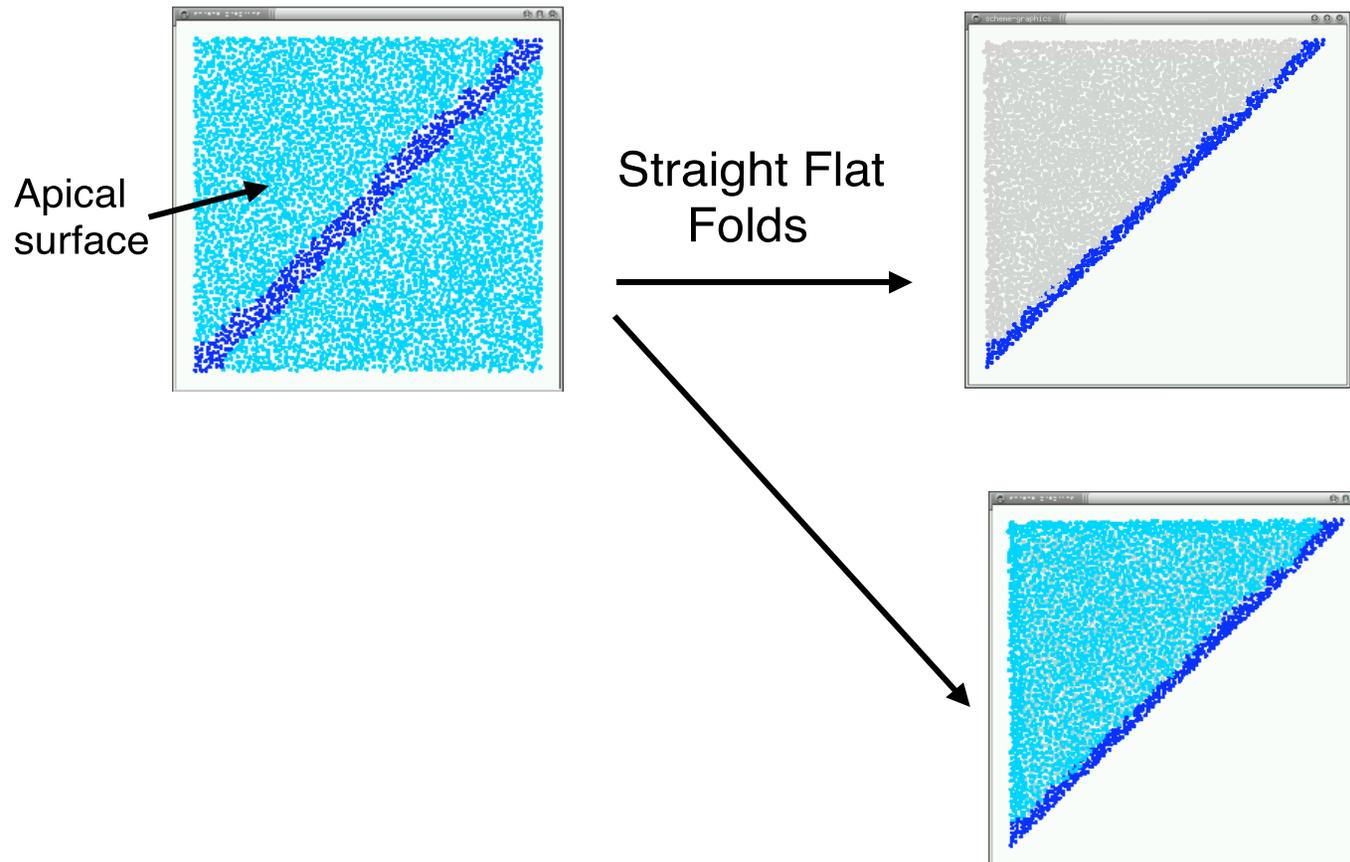
- Flexible Folding



- Polarity Induction

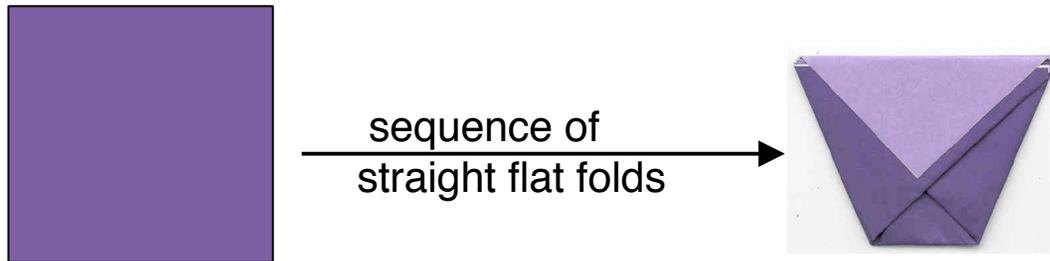


# Folding a cell sheet



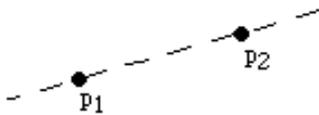
# Origami

- Origami as a *constructive* shape language



- Huzita's Axioms of Origami [1989]

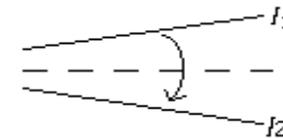
(1) fold-lbp



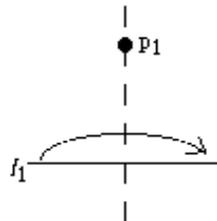
(2) fold-p2p



(3) fold-l2l



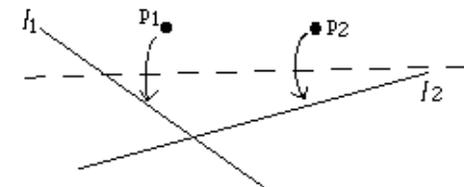
(4) fold-l2self



(5) fold-p2l

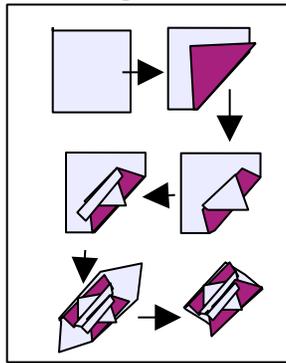


(6) fold-pp2ll



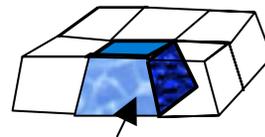
# Approach

Global Shape Program



compiled

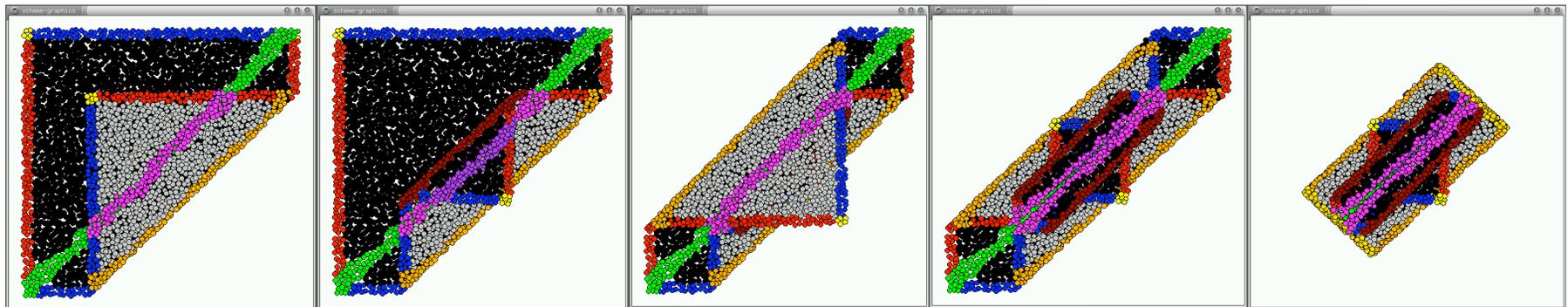
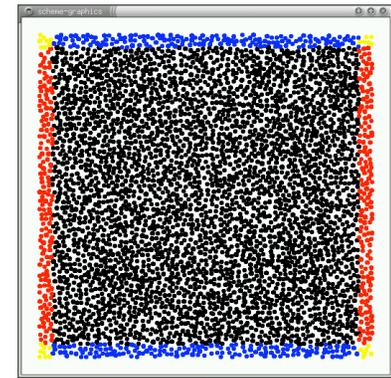
Cell Program



autonomous flexible cell

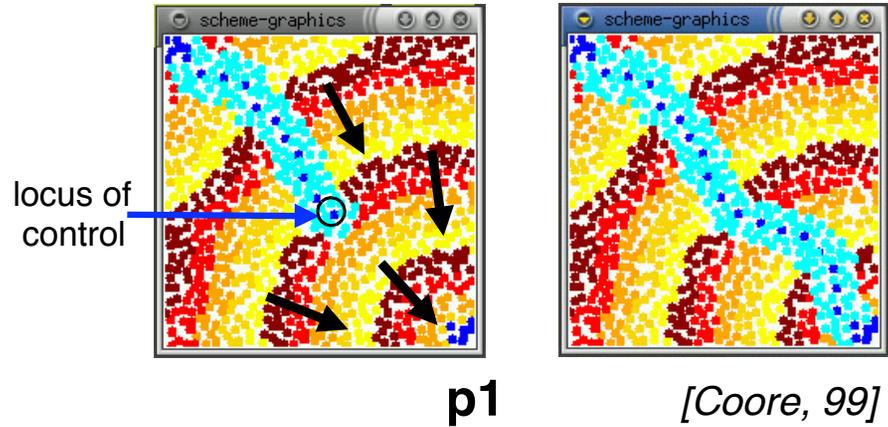
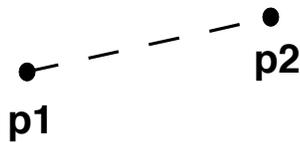
executed

Programmable Cell Sheet

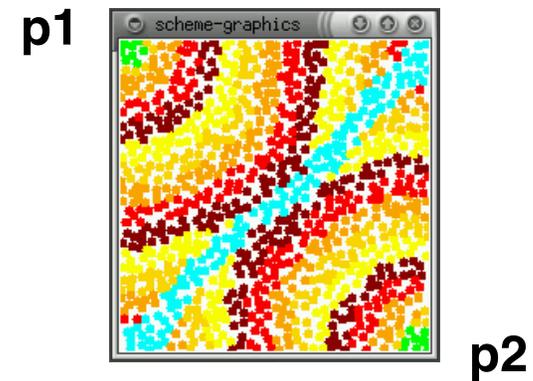
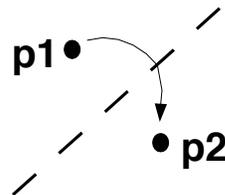


# Implementing the axioms

A1: (crease-lbp p1 p2)



A2:(crease-p2p p1 p2)



## GPL-like code for axiom 2

```
(define (axiom2-rule i1 i2 g1 g2 gend)
  (if i1 (create-gradient g1))
  (if i2 (begin (wait-for-gradient g1)
                (create-gradient g2)))
  (if i1 (begin (wait-for-gradient g1)
                (wait local-delay)
                (create-gradient gend))))

(wait-for-gradients gend)

(if (<= (abs (- g1 g2)) threshold)
    #t
    #f)
)

(define d1 (axiom2-rule c1 c3 gc1 gc2 gend))
```

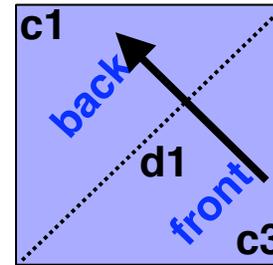
# Origami-based formation of global shape from biologically-inspired local interactions

- An language for self assembling predetermined global shape from a sheet of identically-programmed flexible autonomous cells
- A means of compiling that language into programs for the cells, using biologically-inspired primitives

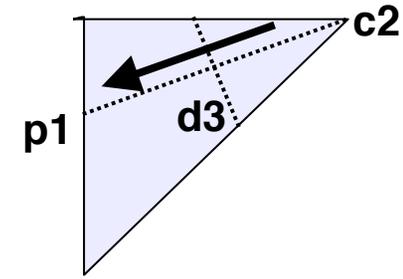
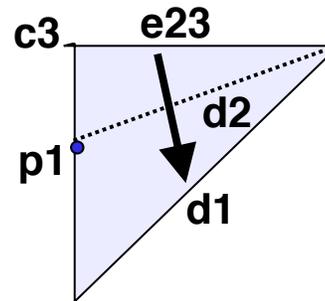
# Origami Shape Language

## Origami Cup Program

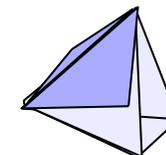
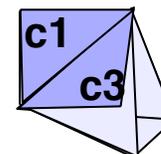
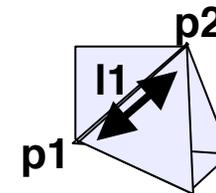
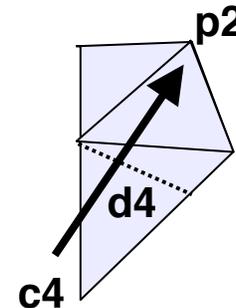
```
(define d1 (fold-p2p c3 c1))
(define-region front (c3 d1))
(define-region back (c1 d1))
(execute-fold d1 apical mark=c3)
```



```
(define d2 (fold-l2l e23 d1))
(define p1 (intersect d2 e34))
(define d3 (fold-p2p c2 p1))
(execute-fold d3 apical mark=c2)
(define p2 (intersect d3 e23))
(define d4 (fold-p2p c4 p2))
(execute-fold d4 apical mark=c4)
```



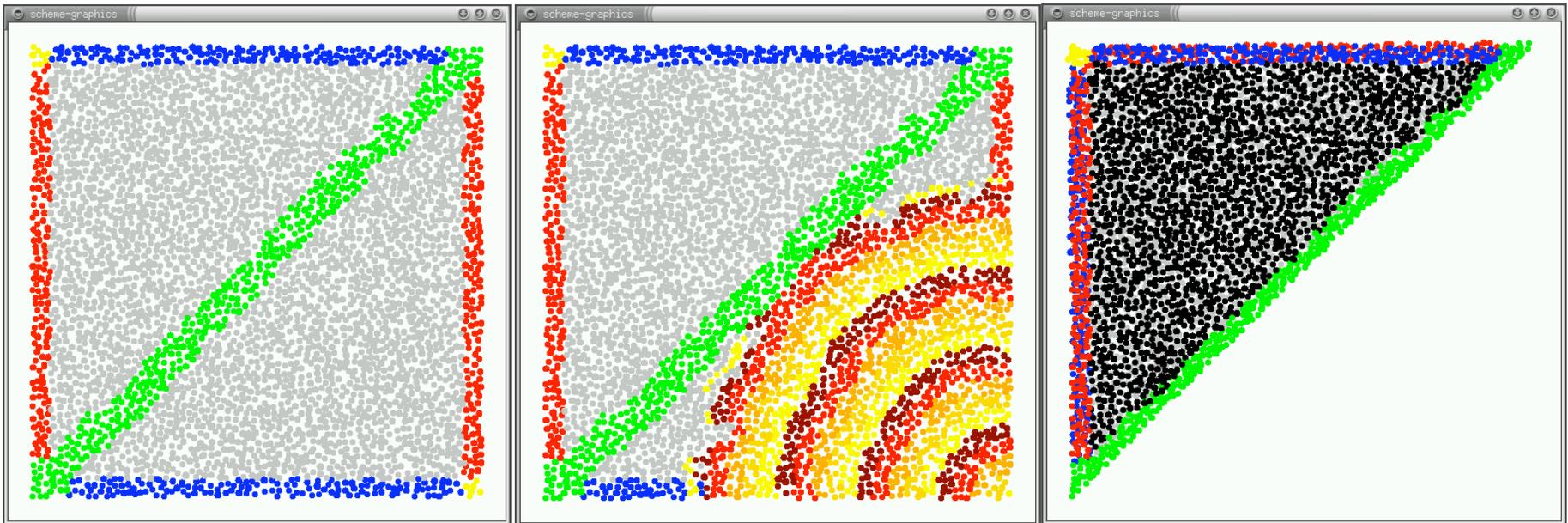
```
(define l1 (fold-lbp p1 p2))
(within-region front
  (execute-fold l1 apical mark=c3))
(within-region back
  (execute-fold l1 basal mark=c1))
```



# Cup Example: 1

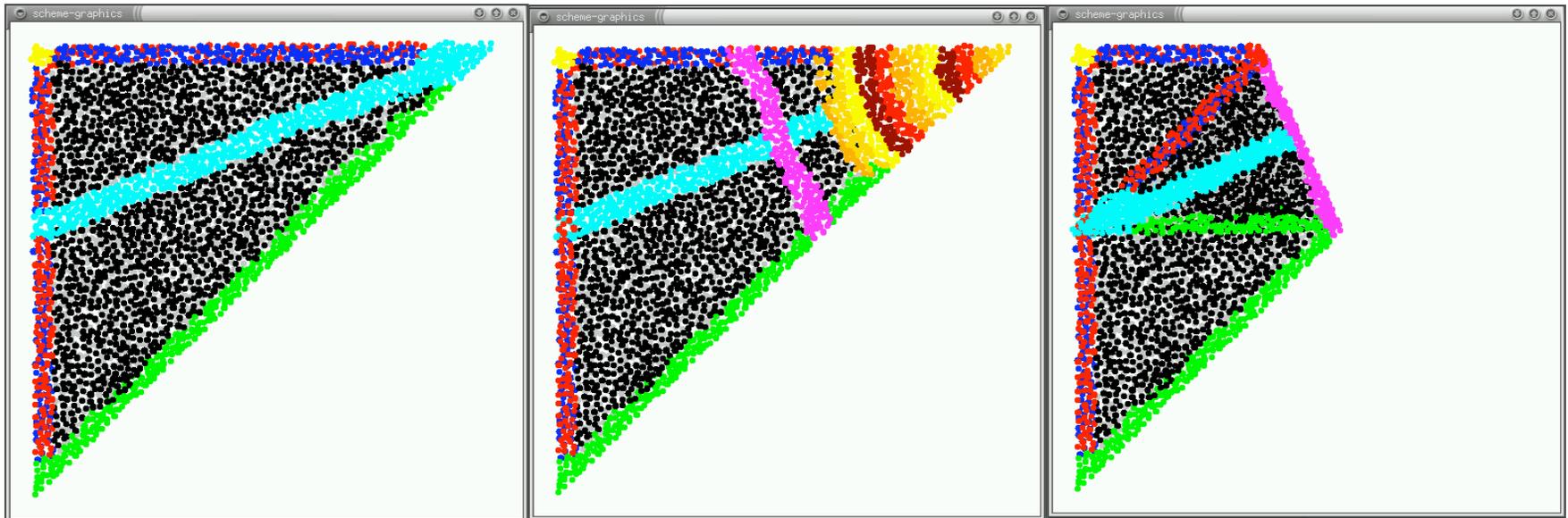
## Origami Cup Program

```
(define d1 (fold-p2p c1 c3 "green"))  
(define-region front (c3 d1))  
(define-region back (c1 d1))  
(execute-fold d1 apical landmark=c3)
```



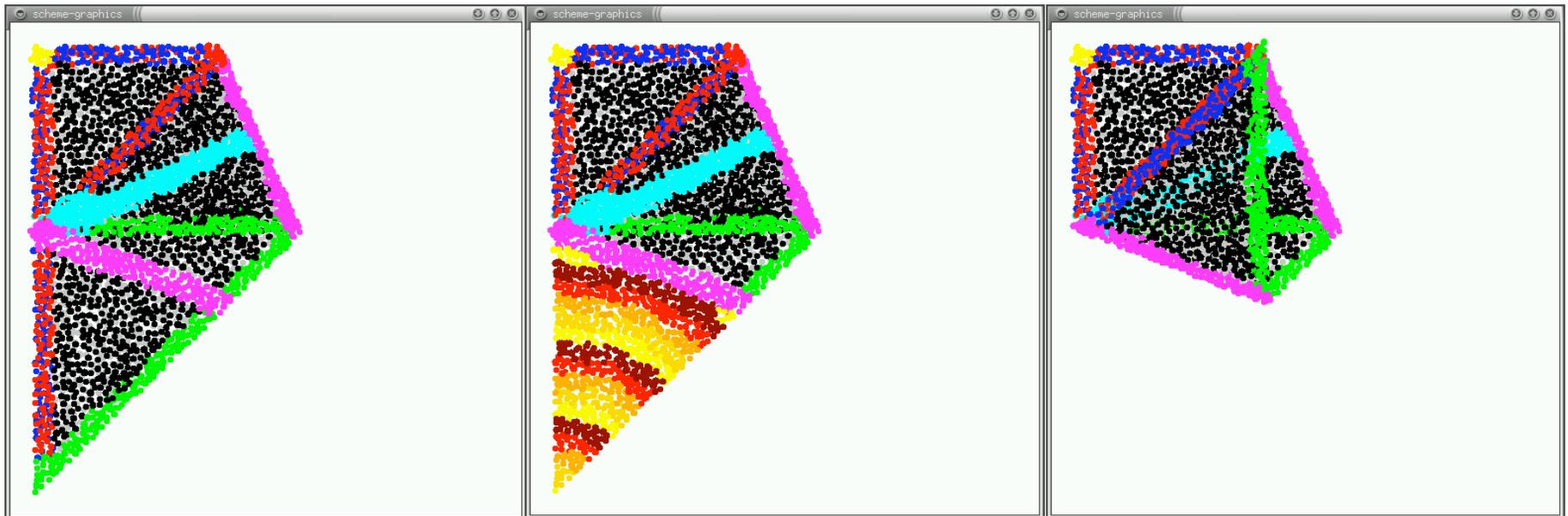
# Cup Example:2

```
(define d2 (fold-l2l e23 d1 "cyan"))  
(define p1 (intersect d2 e34))  
(define d3 (fold-p2p c2 p1 "magenta"))  
(execute-fold d3 apical landmark=c2)
```



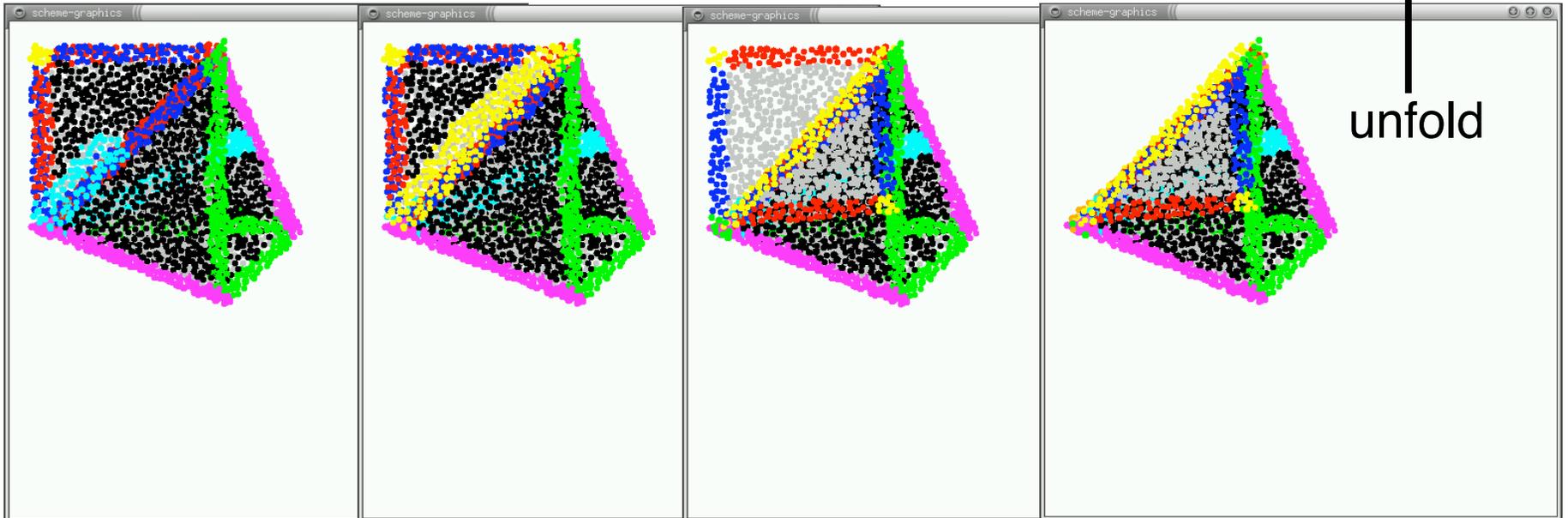
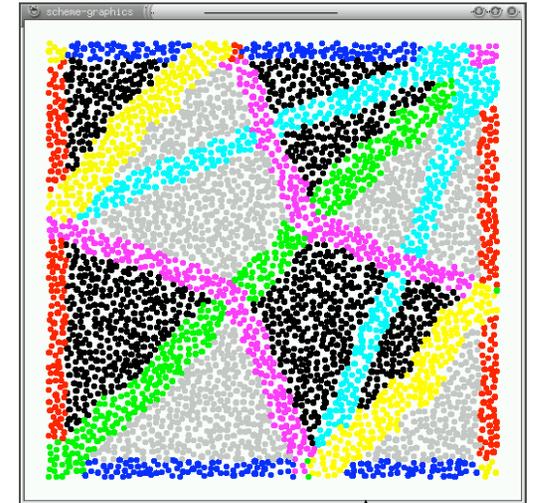
# Cup Example:3

```
(define p2 (intersect d3 e23))  
(define d4 (fold-p2p c4 p2 "magenta"))  
(execute-fold d4 apical landmark=c4)
```

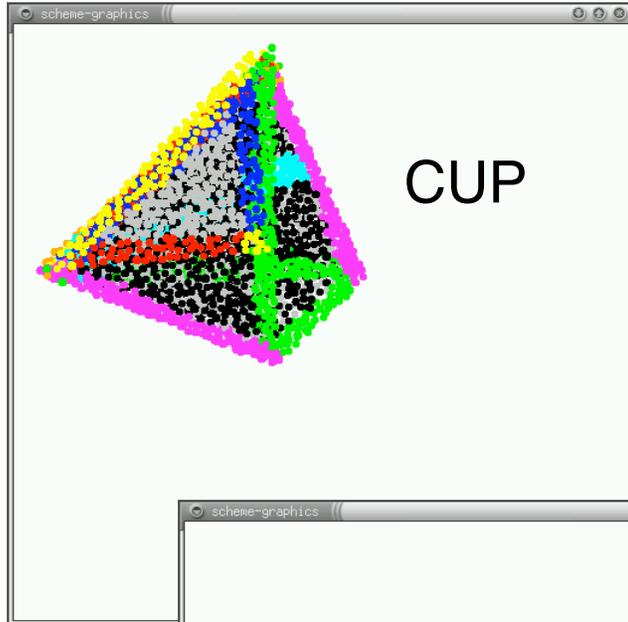


# Cup Example:4

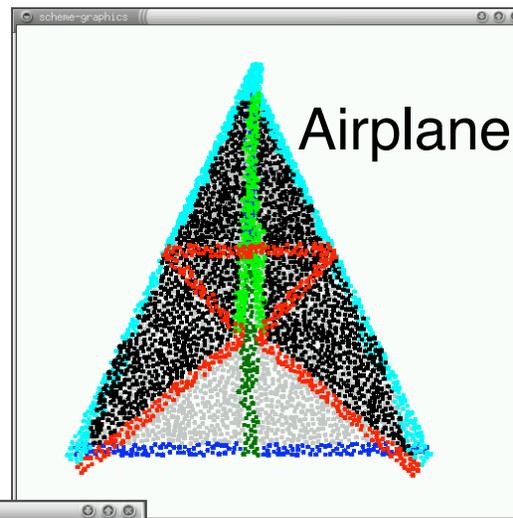
```
(define l1 (fold-lbp p1 p2 "yellow"))  
(within-region front  
  (execute-fold l1 apical landmark=c3))  
(within-region back  
  (execute-fold l1 basal landmark=c1))
```



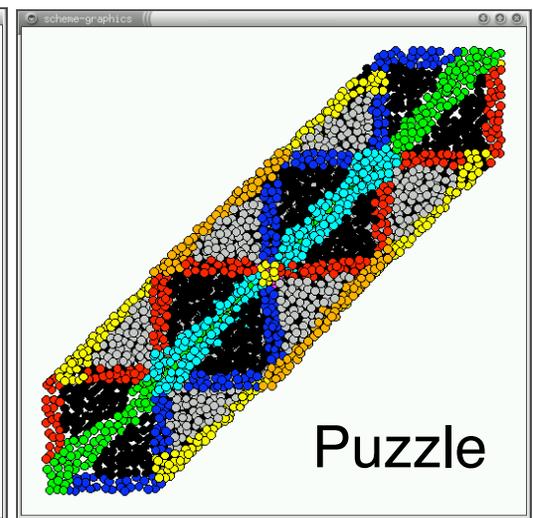
# Examples: Origami Shapes



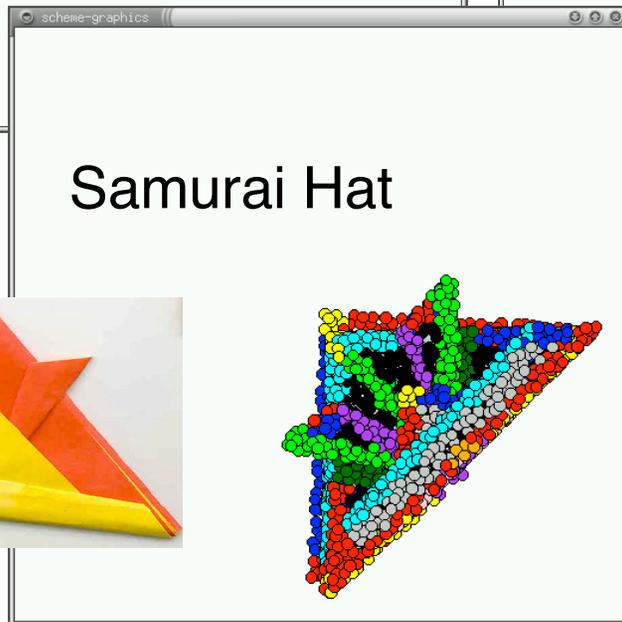
CUP



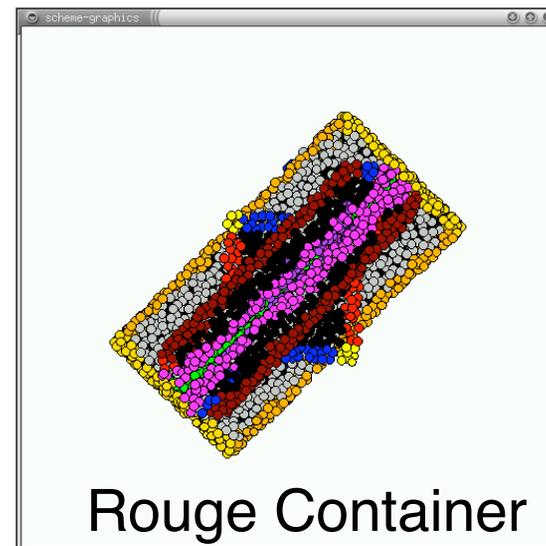
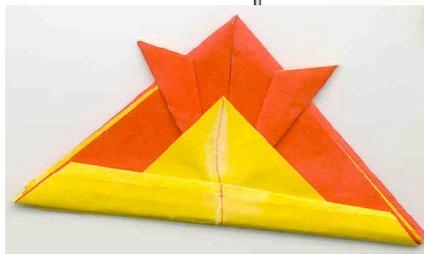
Airplane



Puzzle



Samurai Hat



Rouge Container

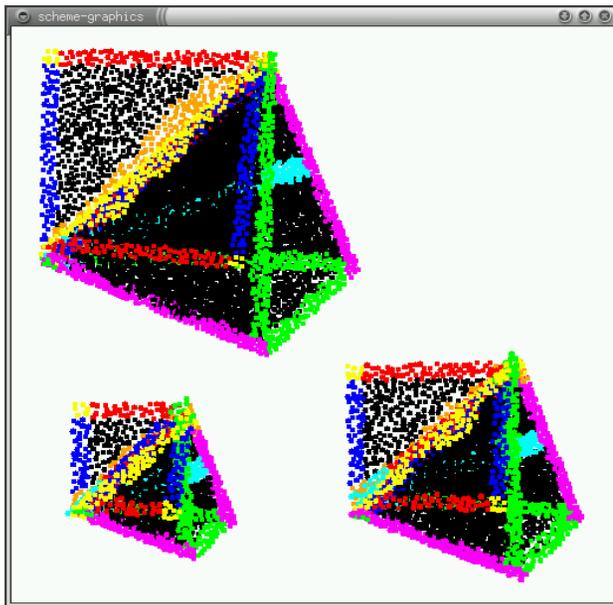


# The origami shape language is well-suited for amorphous systems

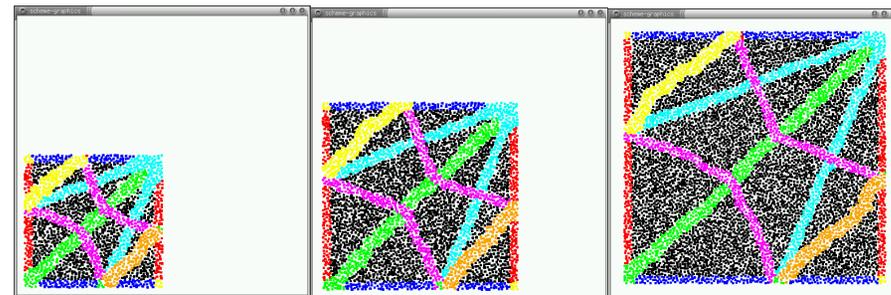
- Wide variety of predetermined global shapes, with only **local** communication and **local** computation
- Local rules are **automatically derived**
- Small set of biologically-inspired primitives
- **Robust** in the face of irregular cell placement, asynchronous cells, random cell death, etc
- Programs **are scale independent**

# Programs are scale independent

- Same shape at different scales
  - Program remains same, irrespective of number of cells



pattern *scales* with number of cells

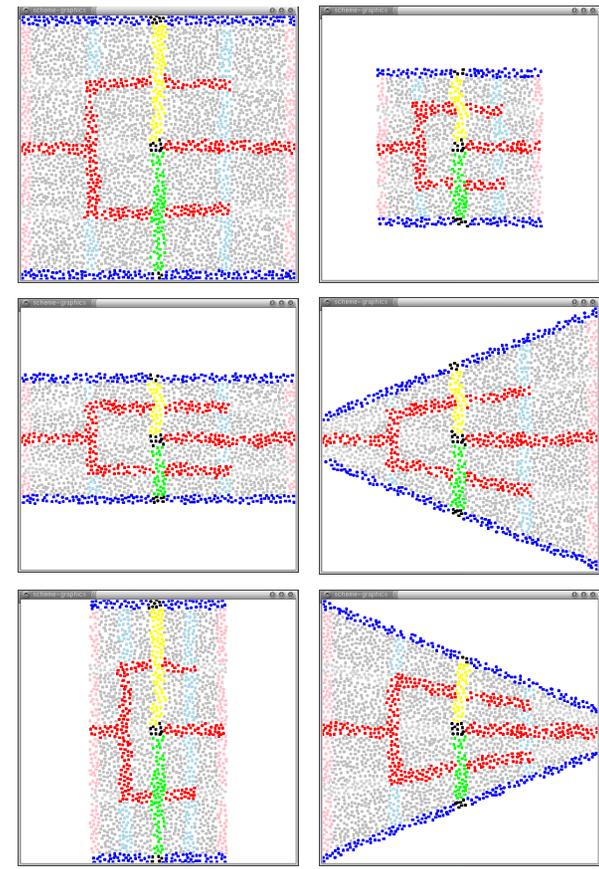
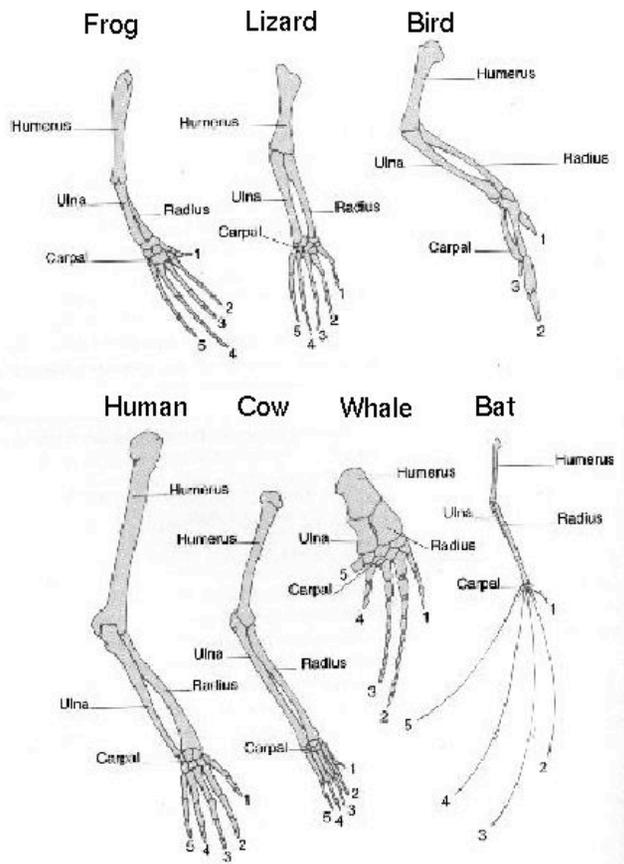


2000

4000

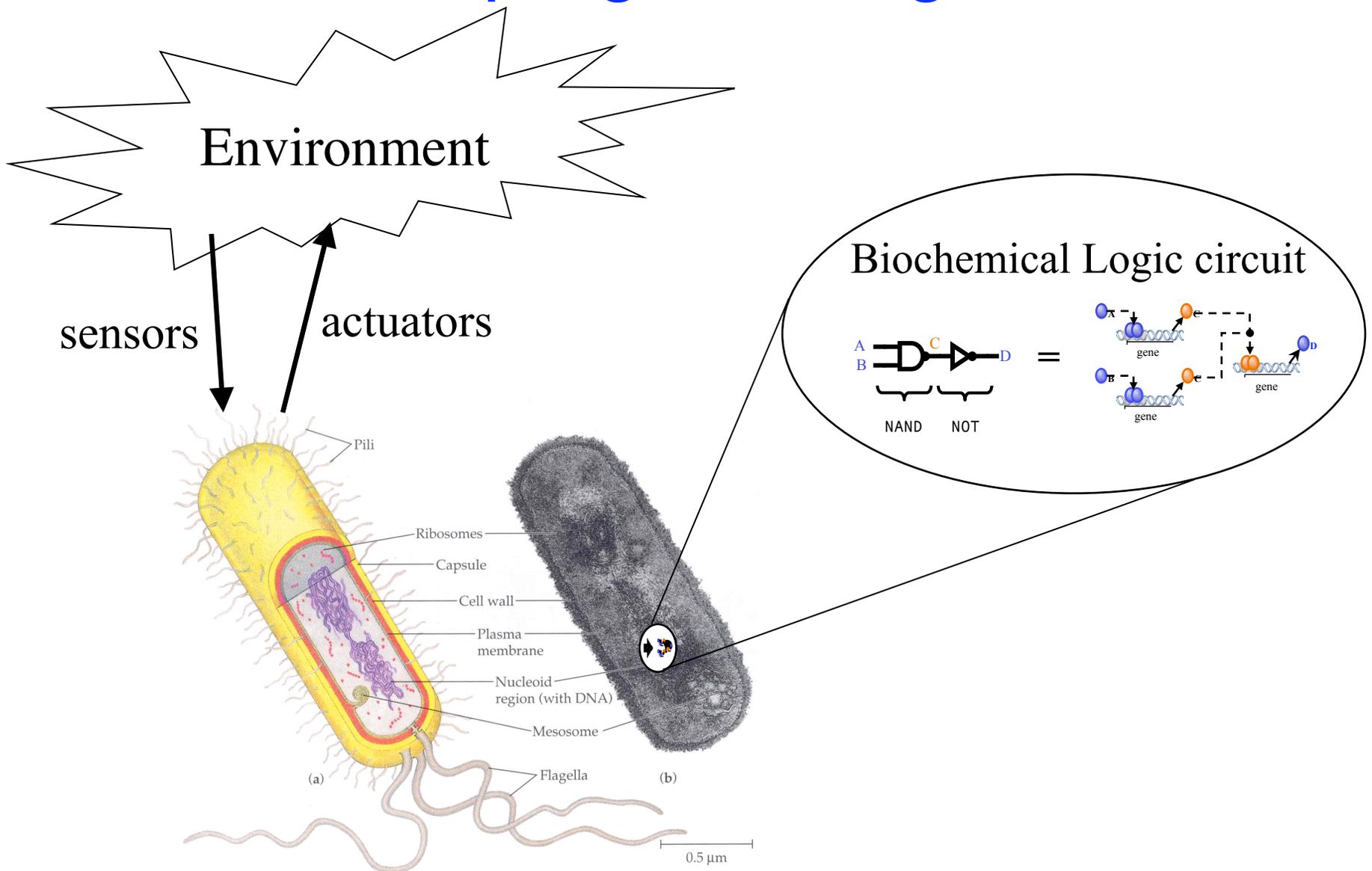
8000

# Similar programs produce homologous structures



Ridley (1997) *Evolution*

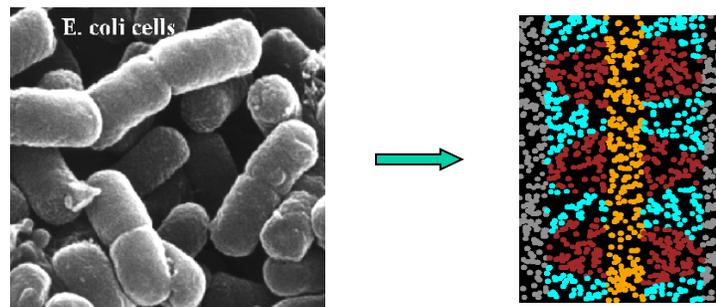
# Cells as a programming substrate



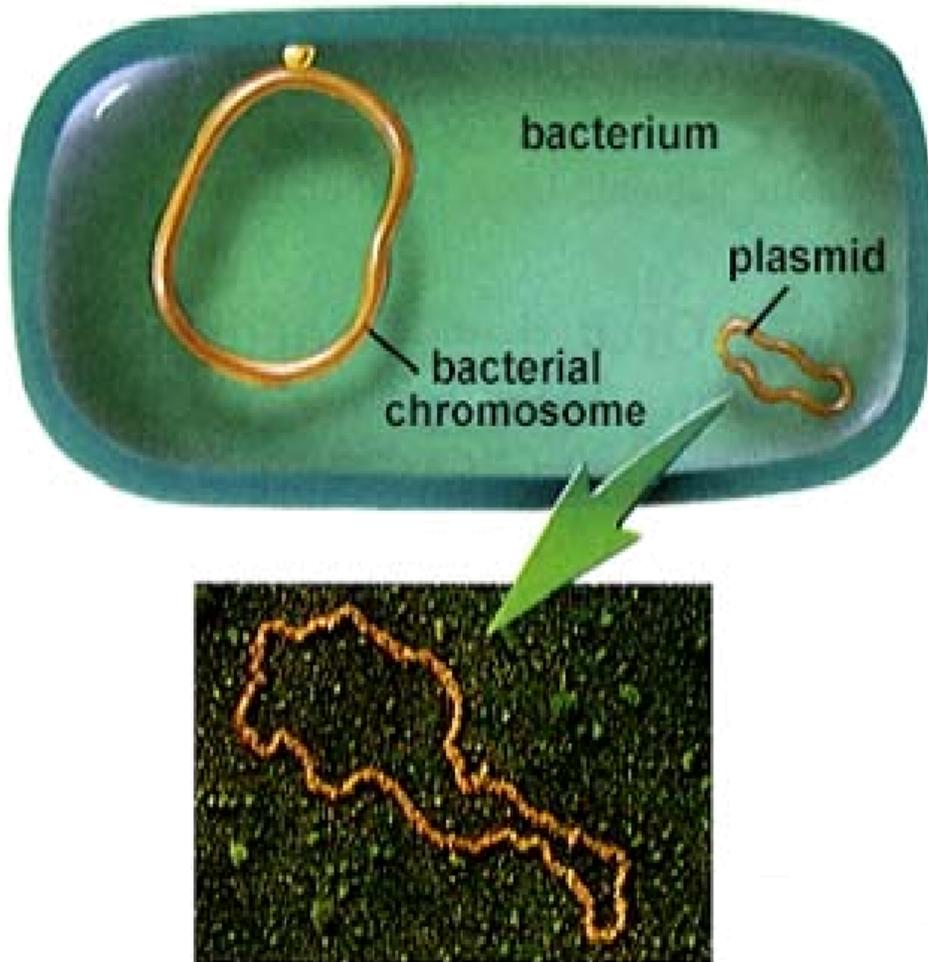
# Vision

- A new substrate for engineering: living cells
  - interface to the chemical world
  - cell as a factory / robot
  - possible biomedical applications

Challenge: engineer complex, predictable behavior



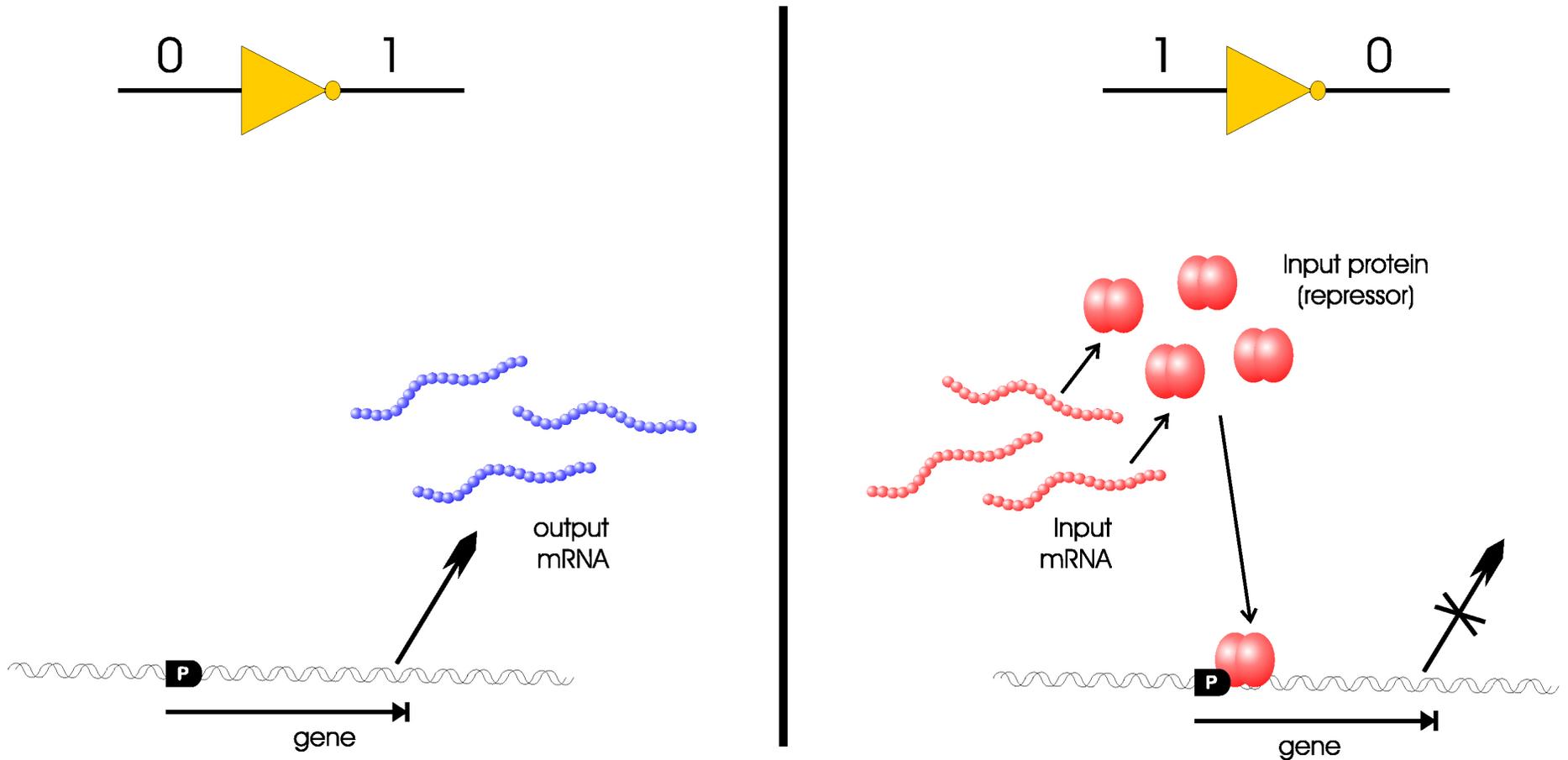
# Programming Cells



- Compile logic circuits into pieces of DNA that encode genetic regulatory networks
- Logic signals are represented as concentrations of proteins (mRNA)
- Action of the genetic regulatory network in a living cell implements the desired logic function

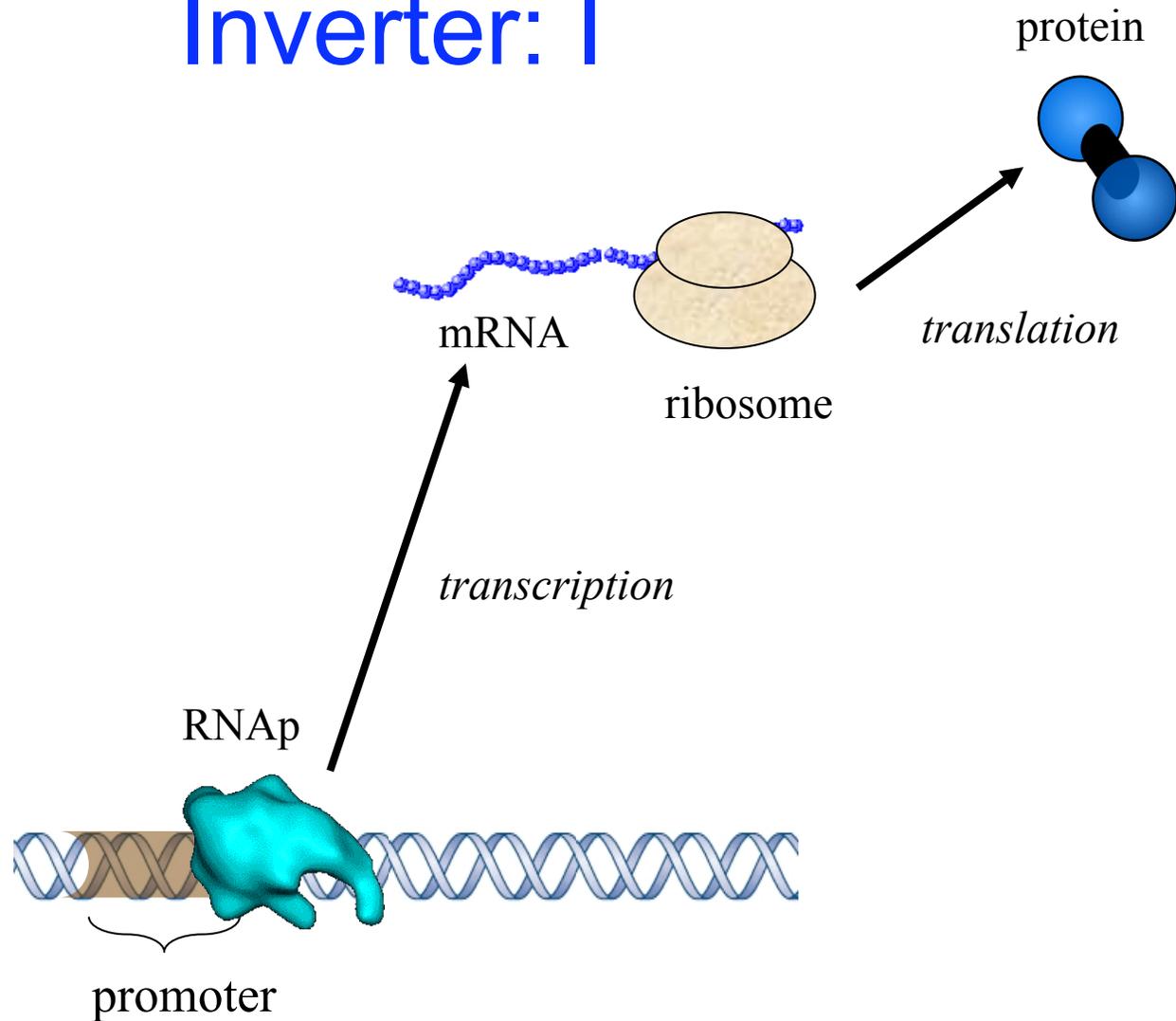
plasmid = “user program”

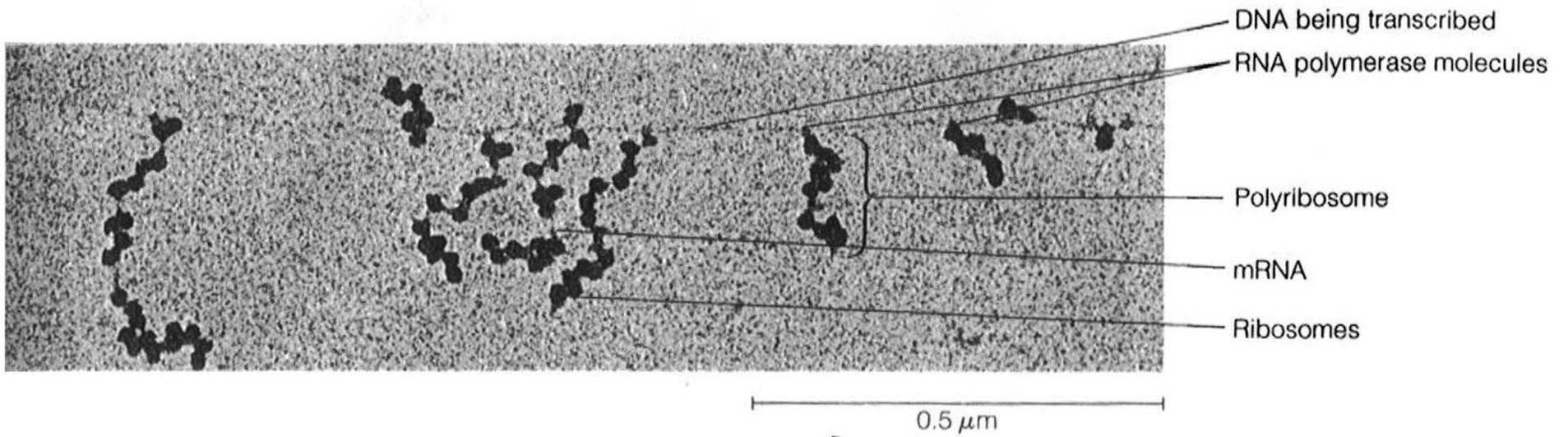
# Biochemical Inverter



signal = concentration of specific proteins (mRNA)  
computation = regulated protein synthesis + decay

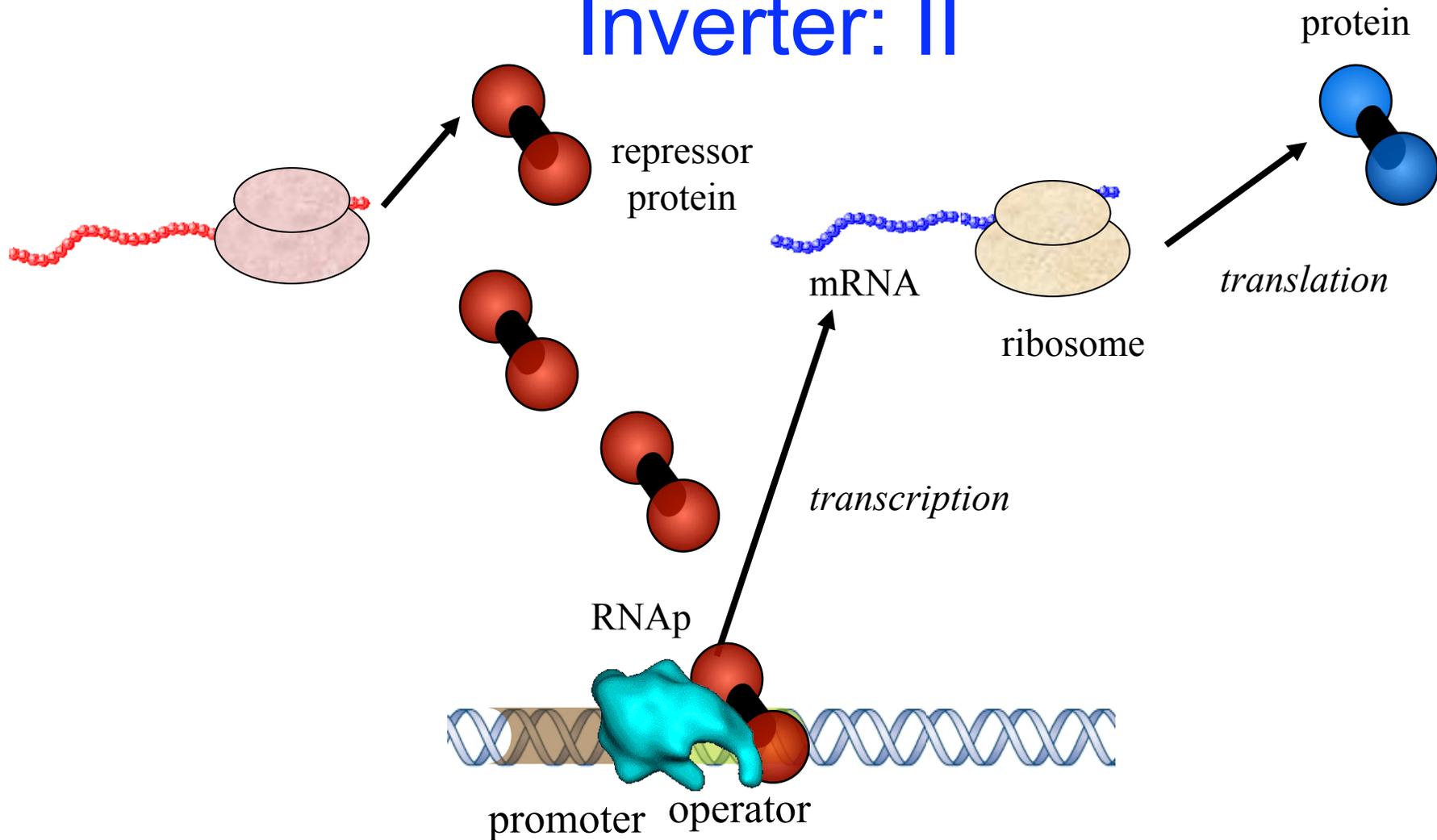
# A More In-Depth Model of the Inverter: I



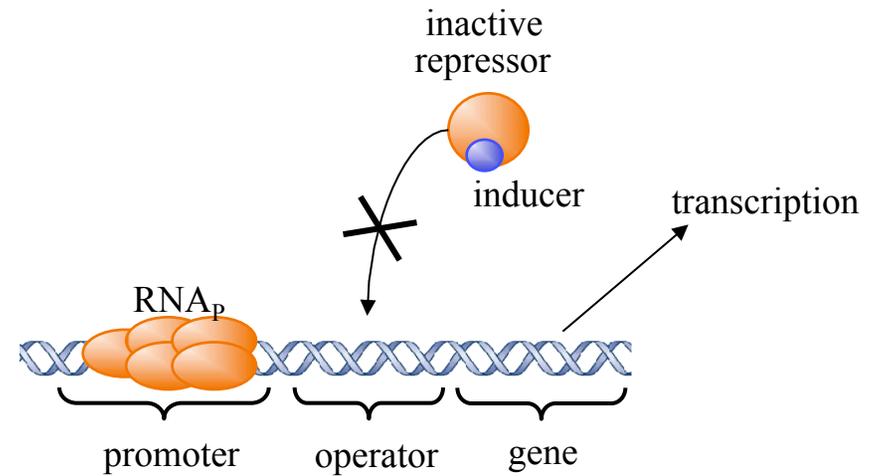
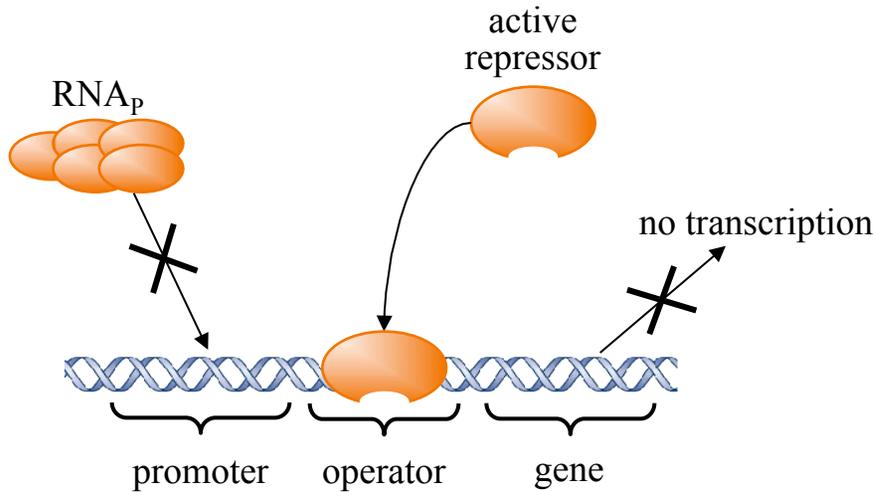


from Becker, Reece, and Poenie, *The World of the Cell*, Benjamin/Cummings, 1996, p. 559

# A More In-Depth Model of the Inverter: II

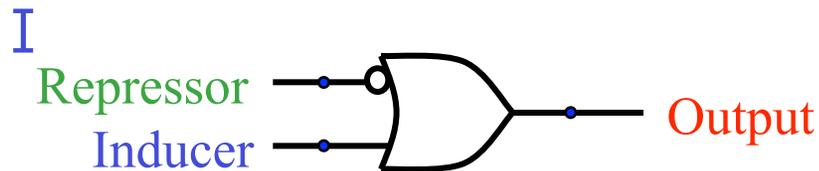


# Inducers



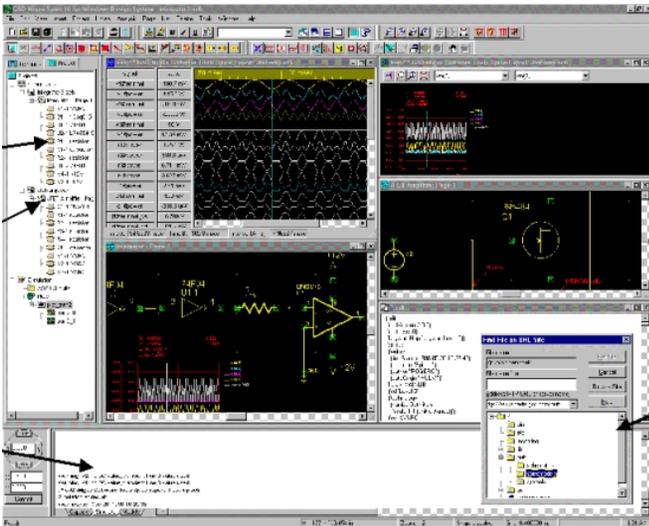
- Use as a logical *Implies* gate:

(NOT **R**) OR



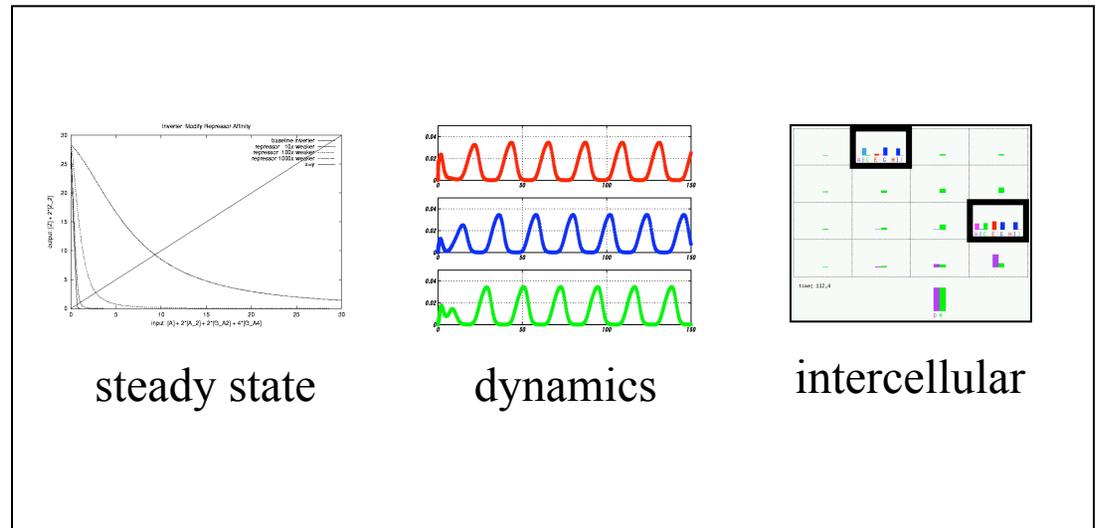
Repressor	Inducer	Output
0	0	1
0	1	1
1	0	0
1	1	1

# BioCircuit Computer-Aided Design



SPICE

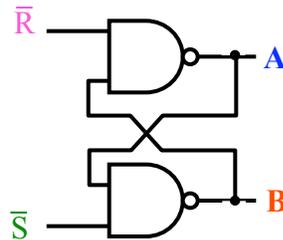
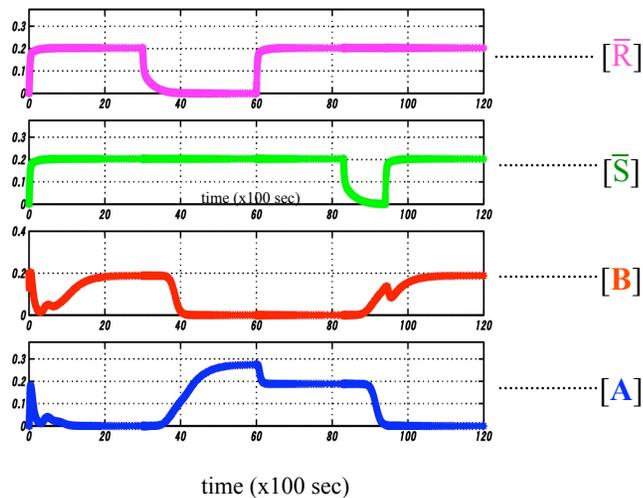
- BioSPICE: a prototype biocircuit CAD tool
  - simulates protein and chemical concentrations
  - intracellular circuits, intercellular communication
  - single cells, small cell aggregates



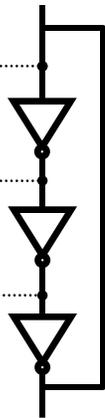
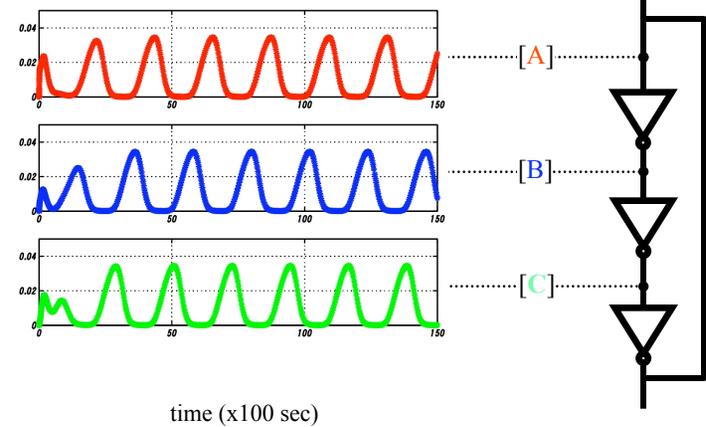
BioSPICE

# “Proof of Concept” Circuits

## RS-Latch (“flip-flop”)

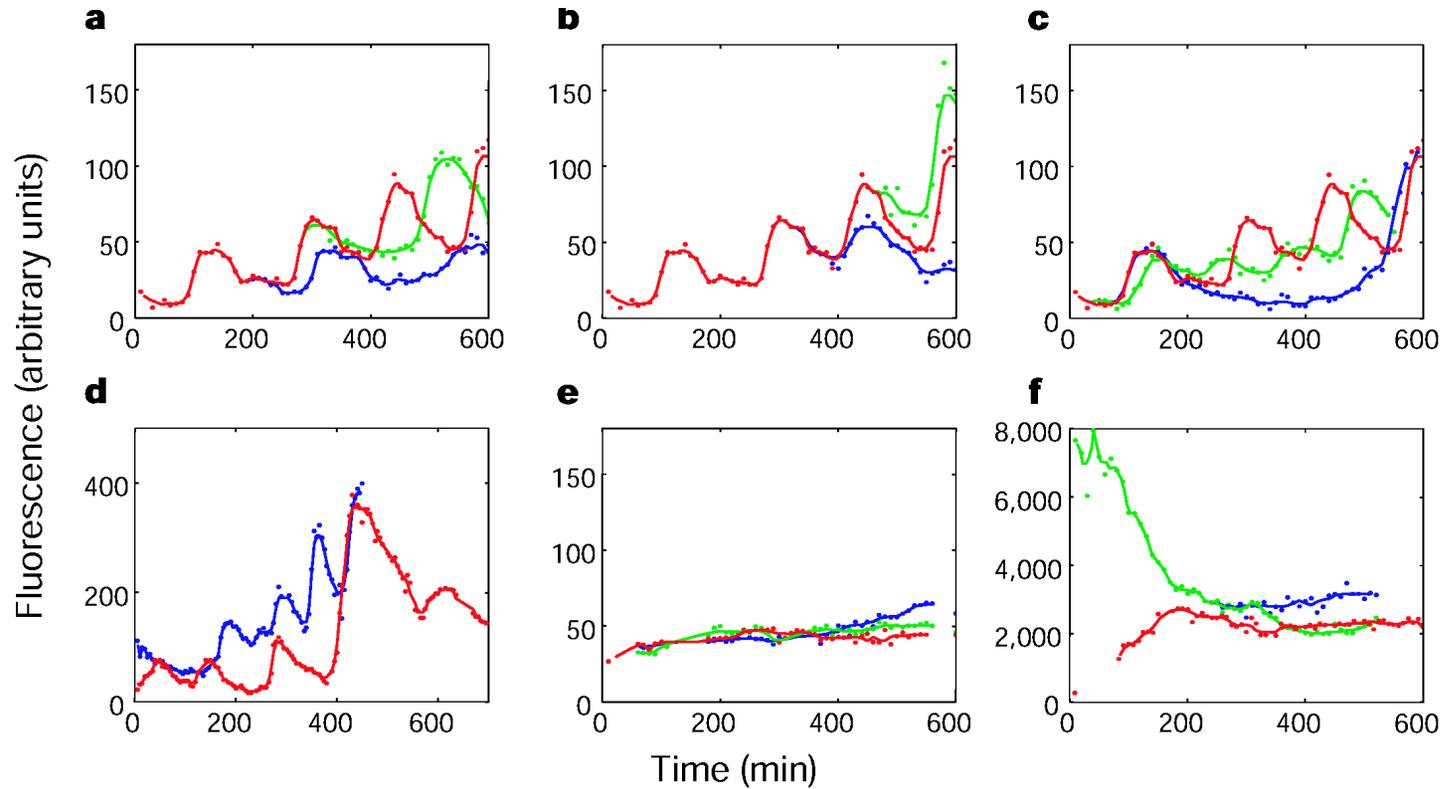


## Ring oscillator



- Work in BioSPICE simulations [Weiss, Homsy, Nagpal, 1998]
- They work in vivo
  - Flip-flop [Gardner & Collins, 2000], Ring oscillator [Elowitz & Leibler, 2000]
- Models poorly predict their behavior

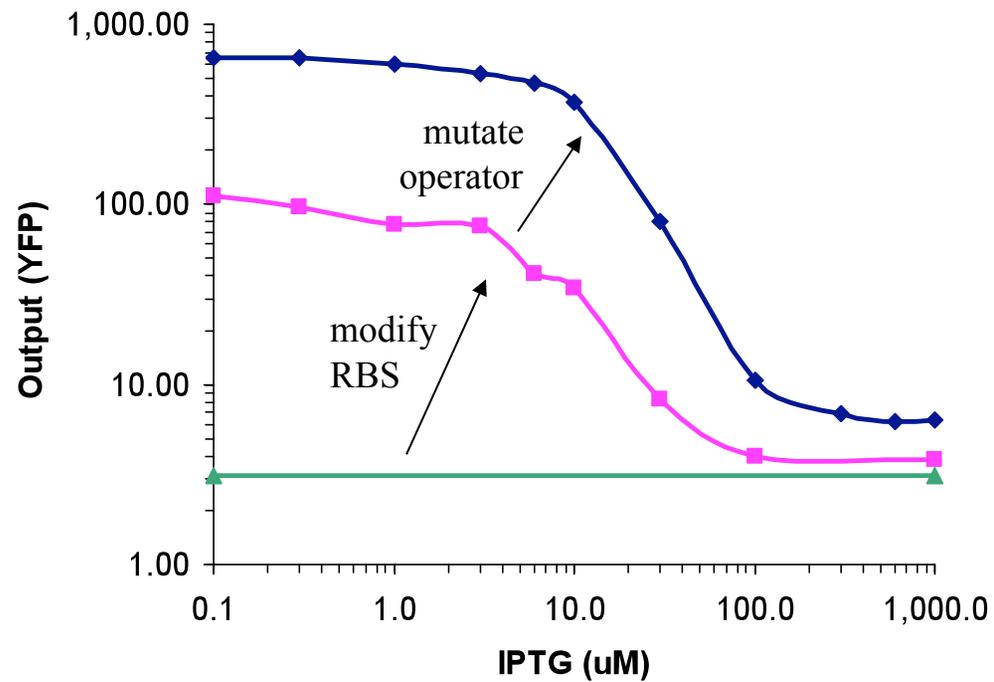
# Measurements of a Ring Oscillator



[Elowitz & Leibler, 2000]

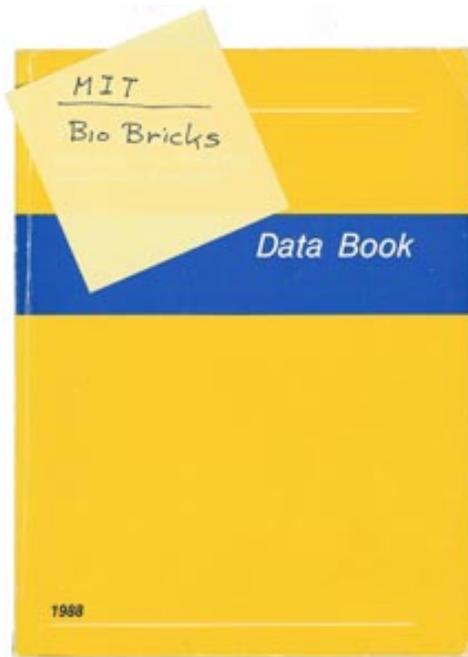
# Genetic Process Engineering

(Ron Weiss)



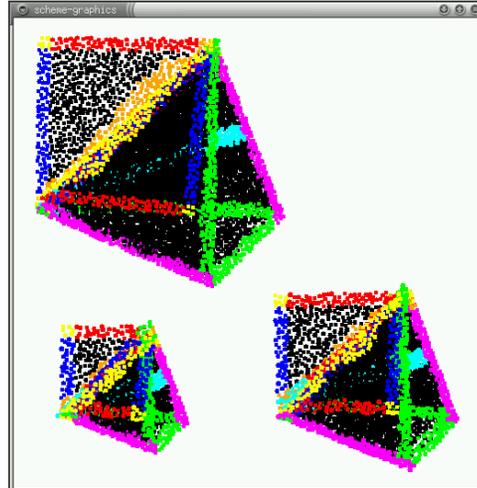
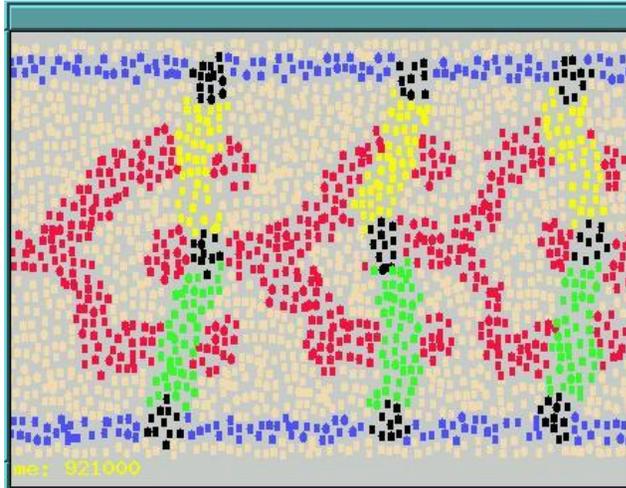
# BioBricks: Future Data Book for Cellular Robotics?

(Randy Rettberg and Tom Knight)



BB Device Number	Description
*BBa D0001	<b>Base Plasmid for Structured Assembly</b>
<u>BBa D0001</u>	<b>Inverter</b>
*BBa D0002	<b>Inverter</b>
*BBa D0003	<b>Inverter</b>
*_BBa D0004	<b>Inverter</b>
*BBa D0001	<b>Green Fluorescence Output</b>
*_BBa D0002	<b>Cyan Fluorescence Output</b>
*BBa D0003	<b>Yellow Fluorescence Output</b>
...	...

# Putting it all together?



Robust  
gradients?

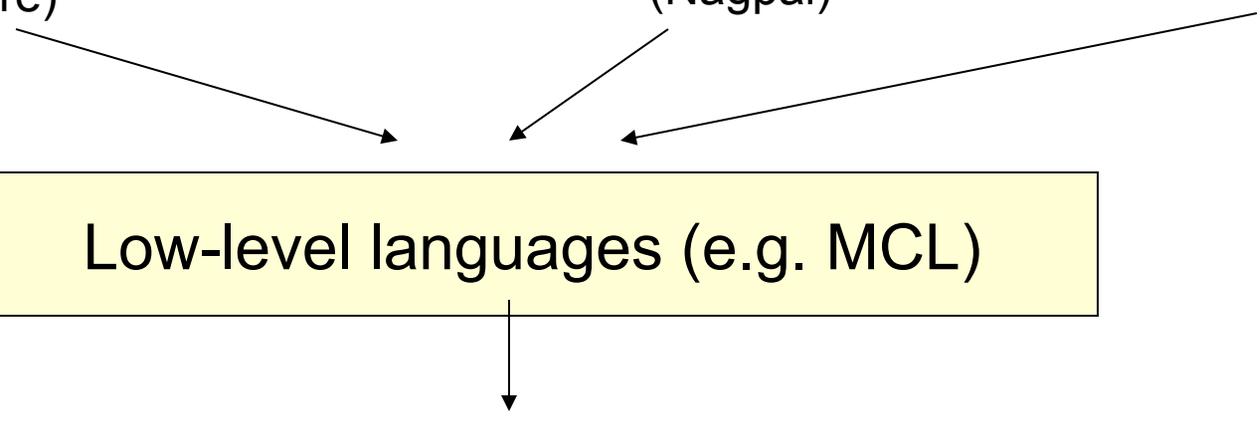


Growing Point Language  
(Coore)

Origami Shape Language  
(Nagpal)

Low-level languages (e.g. MCL)

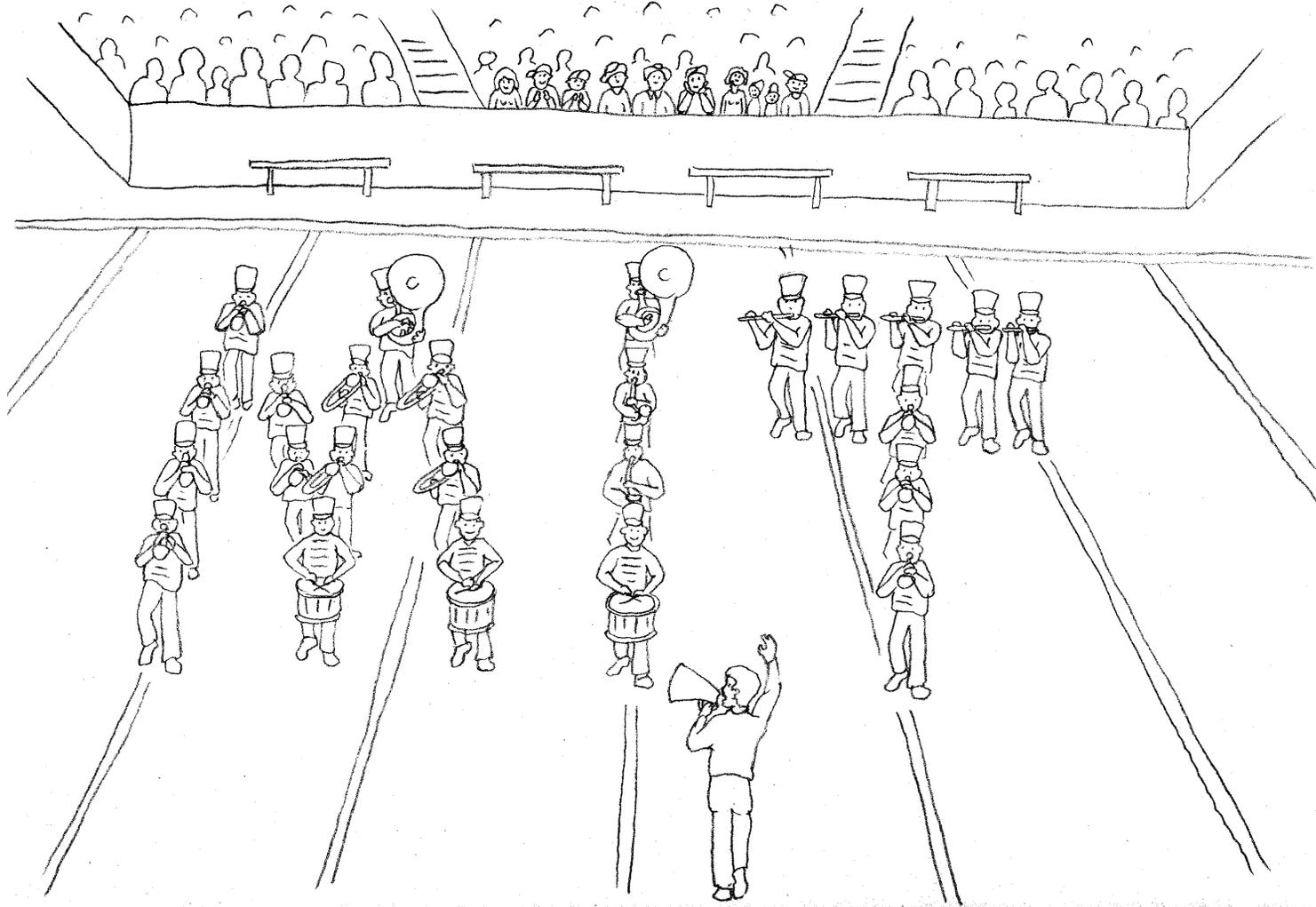
Genetic Circuit



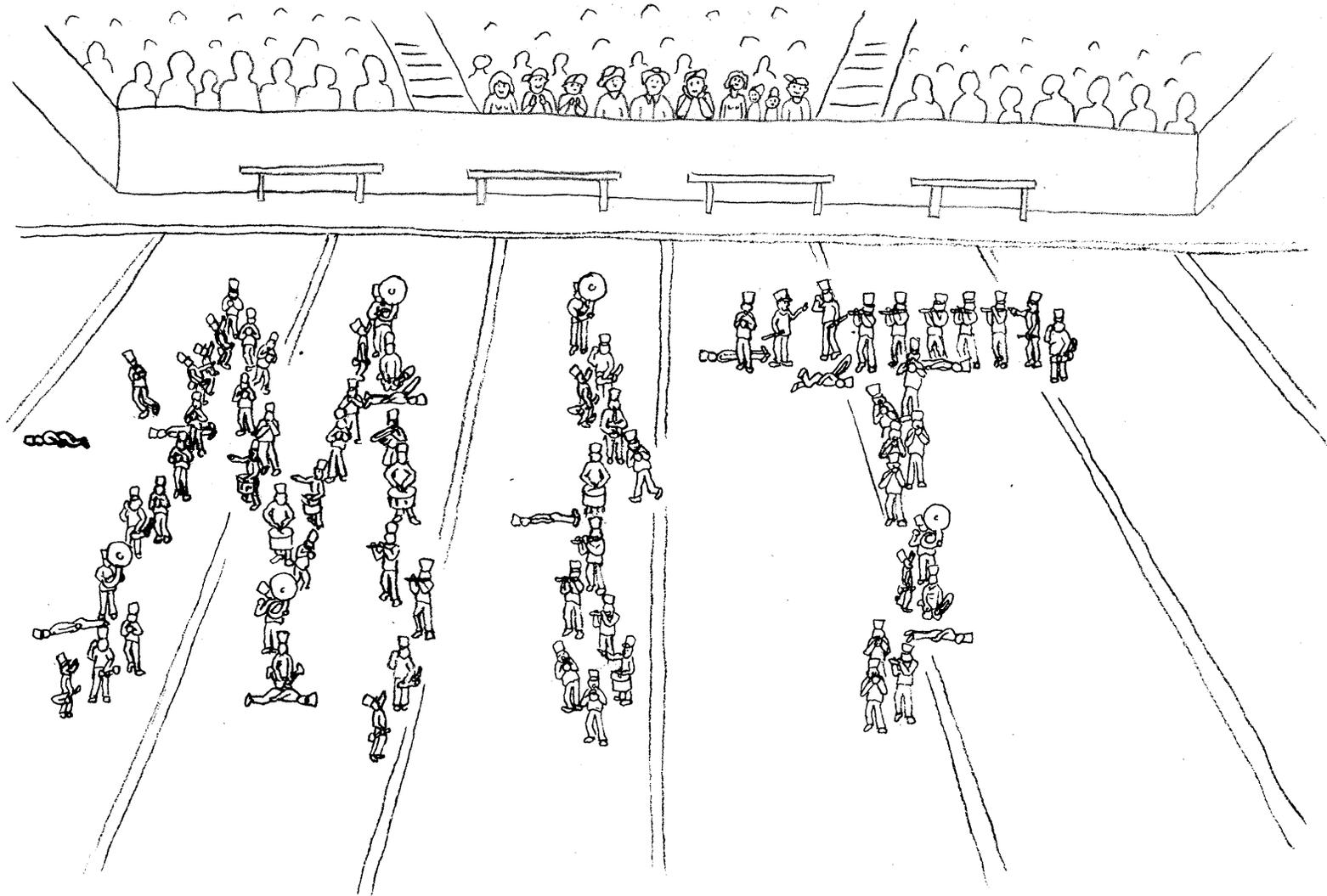
# The Challenge of Amorphous Computing

- To reliably obtain a desired behavior by engineering the cooperation of myriads of computing elements, without assuming any precision interconnect or precision geometrical arrangement of the elements.
- To invent the computational substrates that can support this kind of engineering

# The Old-fashioned Way

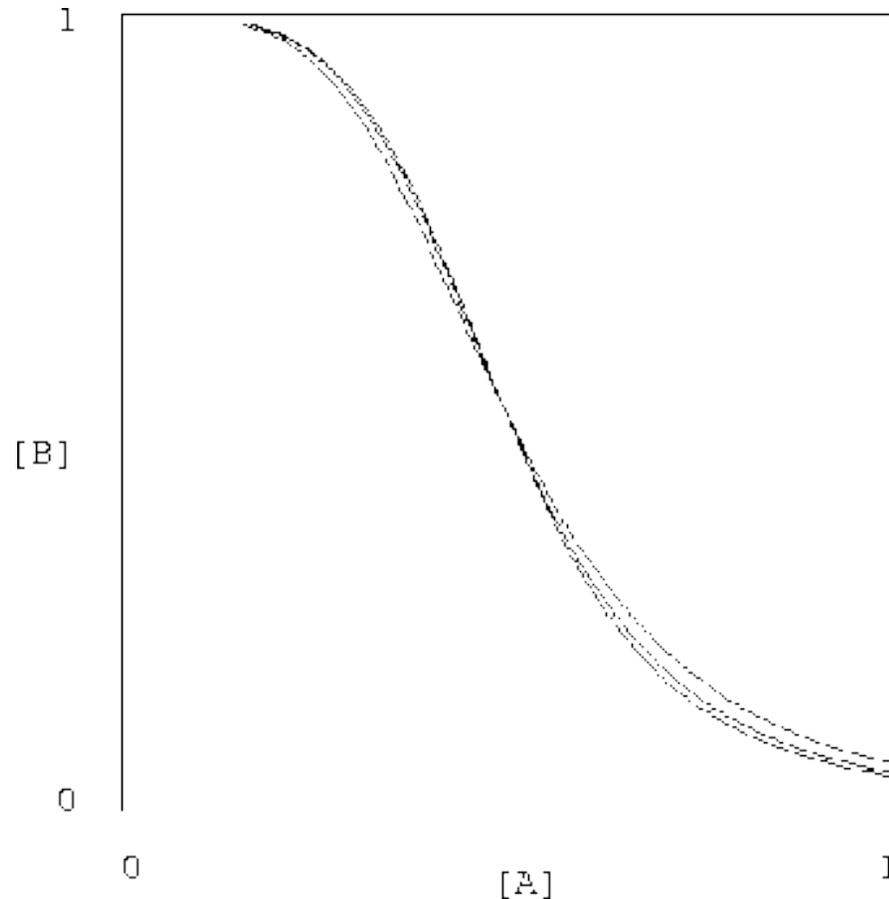
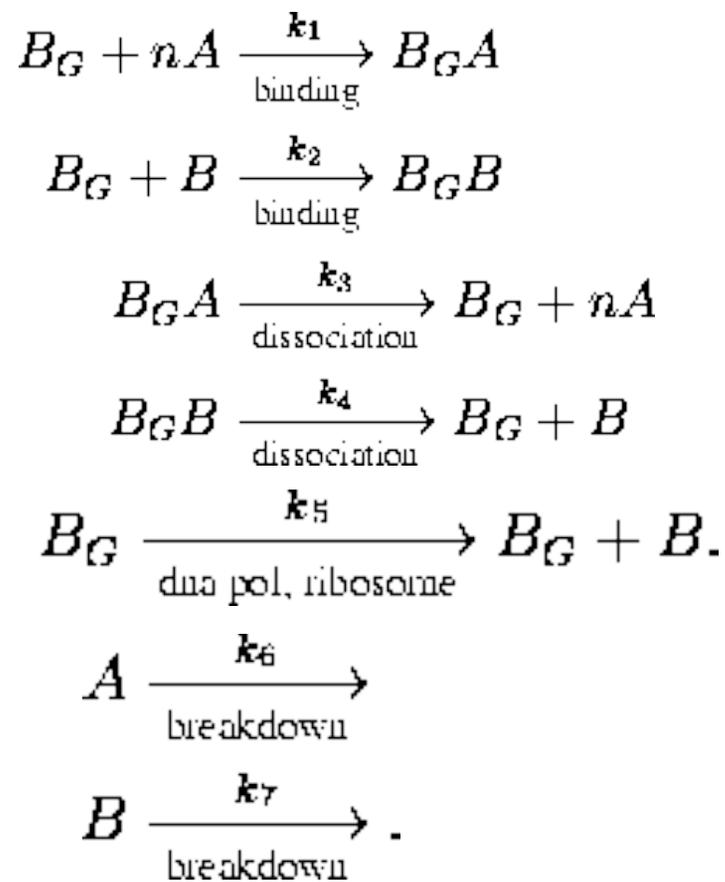


# The Amorphous Way



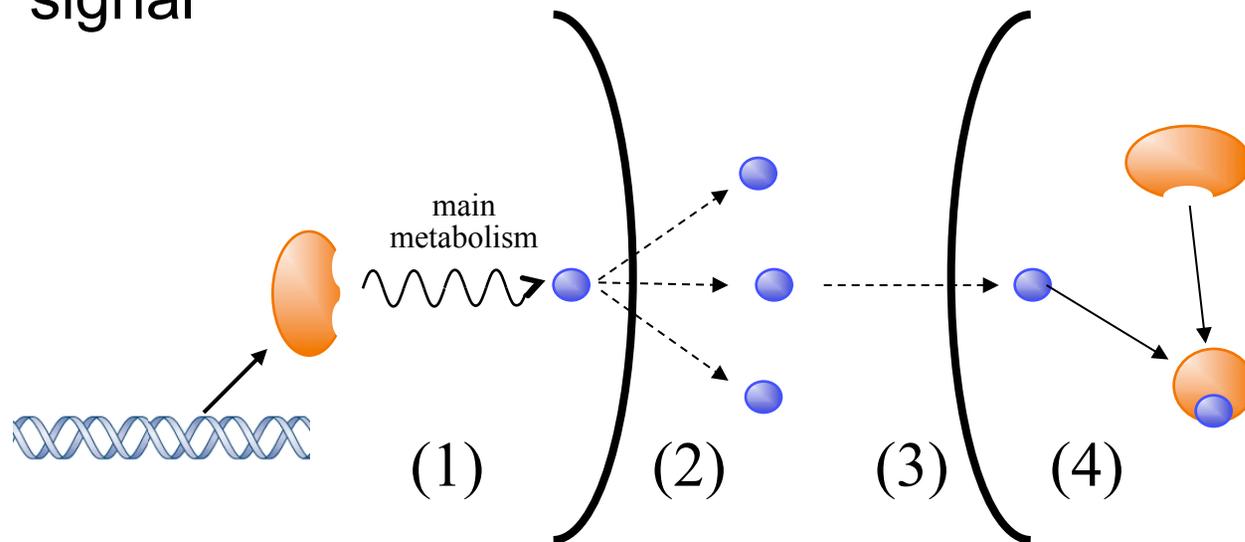
END

# Chemical mechanism for inverter

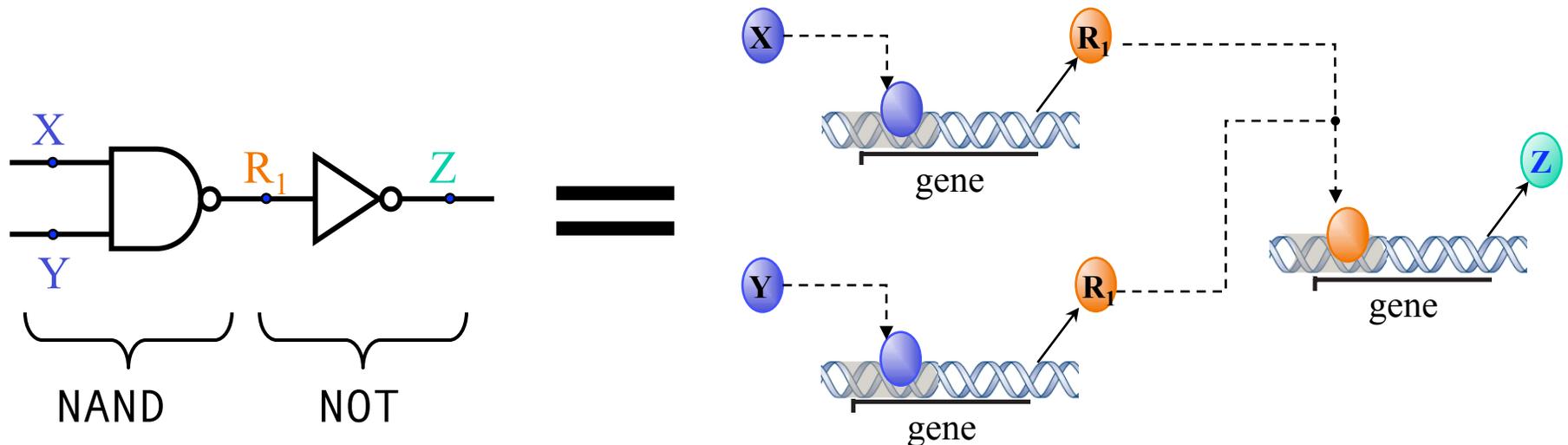


# Intercellular Communications

- Certain inducers useful for communications:
  1. A cell produces inducer
  2. Inducer diffuses outside the cell
  3. Inducer enters another cell
  4. Inducer interacts with repressor/activator → change signal



# Logic Circuits based on Inverters



- Proteins are the wires/signals
- Promoter + decay implement the gates
- NAND gate is a universal logic element:
  - any (finite) digital circuit can be built!