Cellular automata are idealized models of complex systems

- Large network of simple components
- Limited communication among components
- No central control
- Complex dynamics from simple rules
- Capability of information processing / computation
- Can be evolved via GAs

Terminology:

- Singular: “cellular automaton” (CA)
- Plural: “cellular automata” (CAs)

Pronunciation:

- American: “cellular au**TO**mata”
- British: “cellular auto**MA**ta”

The **Game of “Life”**: The world’s most famous cellular automaton.

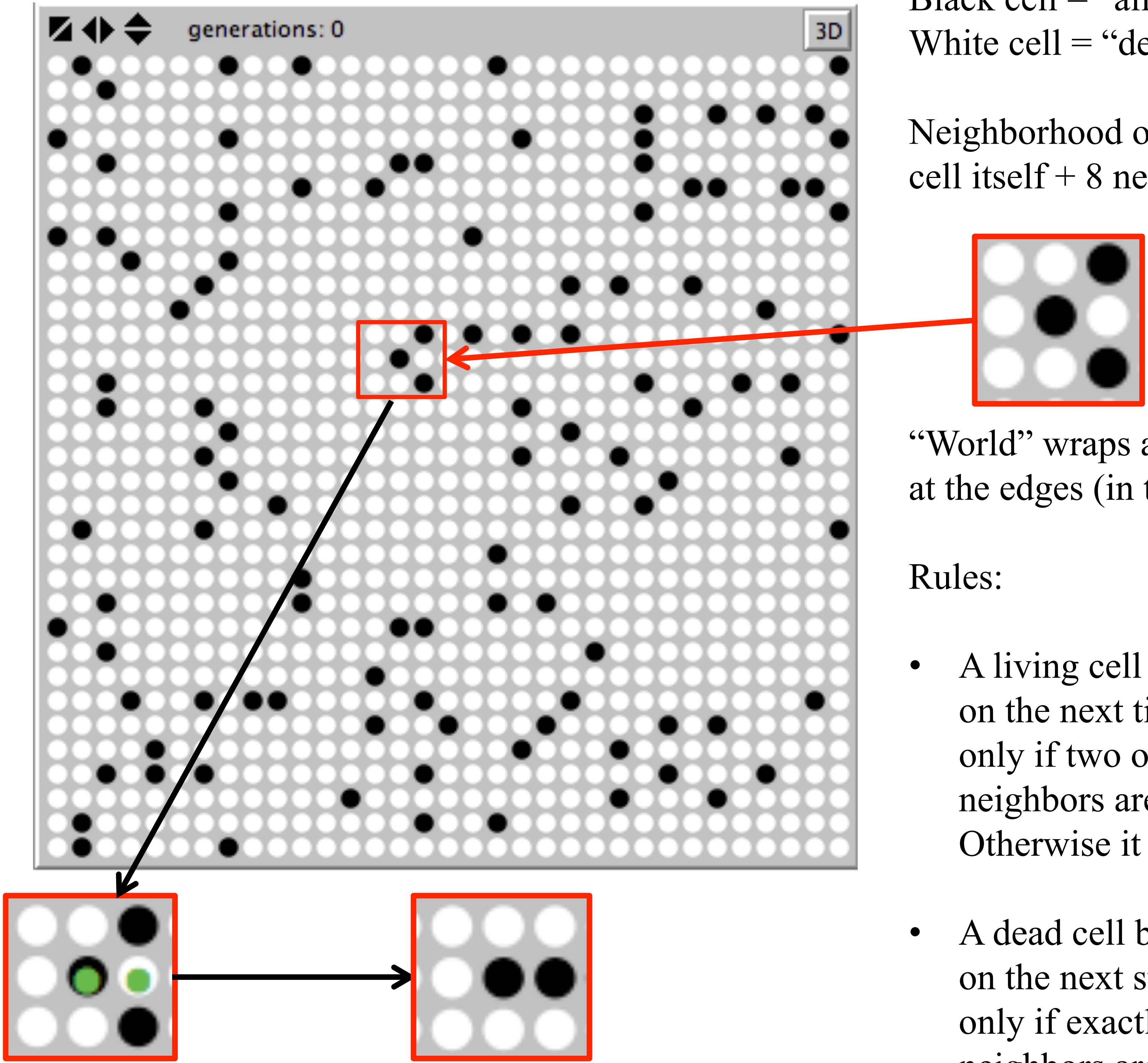
Not really a game.

Published in 1970 by British mathematician John Conway. via Martin Gardner’s “Mathematical Games” column in *Scientific American*.

John Conway

“Life”: Inspired by John von Neumann’s models of life-like processes in cellular automata.

Simple system that exhibits *emergence* and *self-organization*



Black cell = "alive"
White cell = "dead"

Neighborhood of a cell:
cell itself + 8 neighbors

"World" wraps around
at the edges (in this version)

Rules:

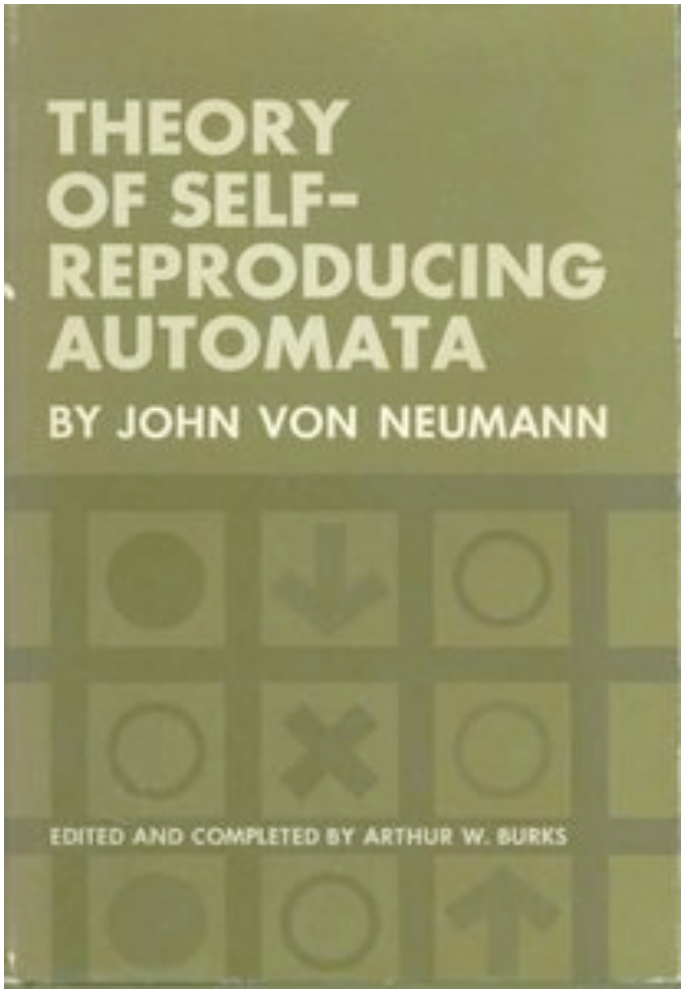
- A living cell remains alive on the next time step only if two or three neighbors are alive. Otherwise it dies.

- A dead cell becomes alive on the next step only if exactly three neighbors are alive.

Cellular automata were invented in the 1940s by **Stanislaw Ulam** and **John von Neumann** to prove that self-reproduction is possible in machines (and to further link biology and computation).

John von Neumann
1903-1957

Stanislaw Ulam
1909-1984



THEORY OF SELF-REPRODUCING AUTOMATA

BY JOHN VON NEUMANN

EDITED AND COMPLETED BY ARTHUR W. BURKS

Applications of CAs

- **Computer Science:** architecture for massively parallel computation, and for molecular scale computation

- **Complex Systems:**
  - Tool for modeling processes in physics, geology, chemistry, biology, economics, sociology, etc.

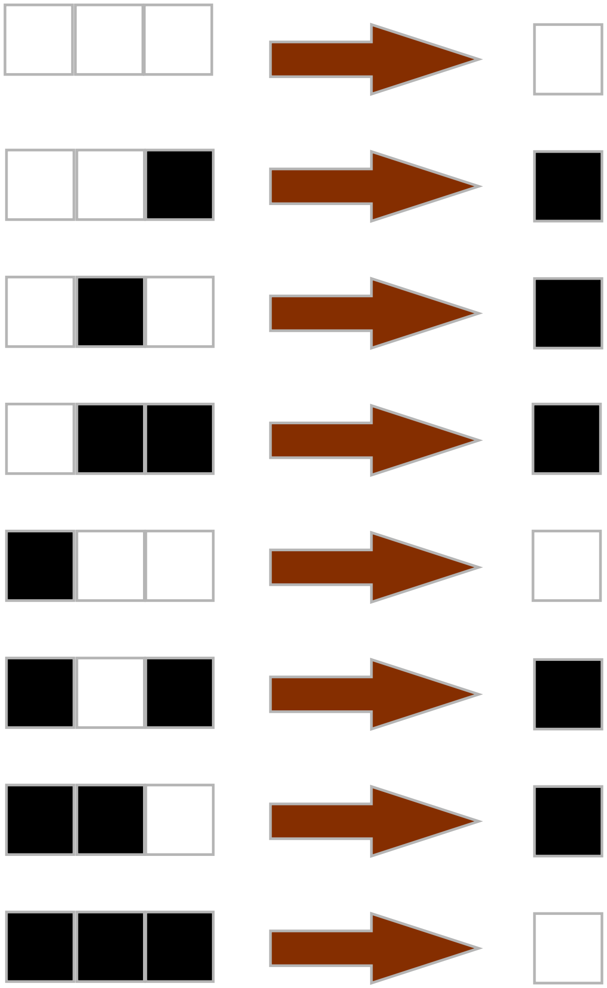
  - Tool for studying abstract notions of self-organization and emergent computation in complex systems

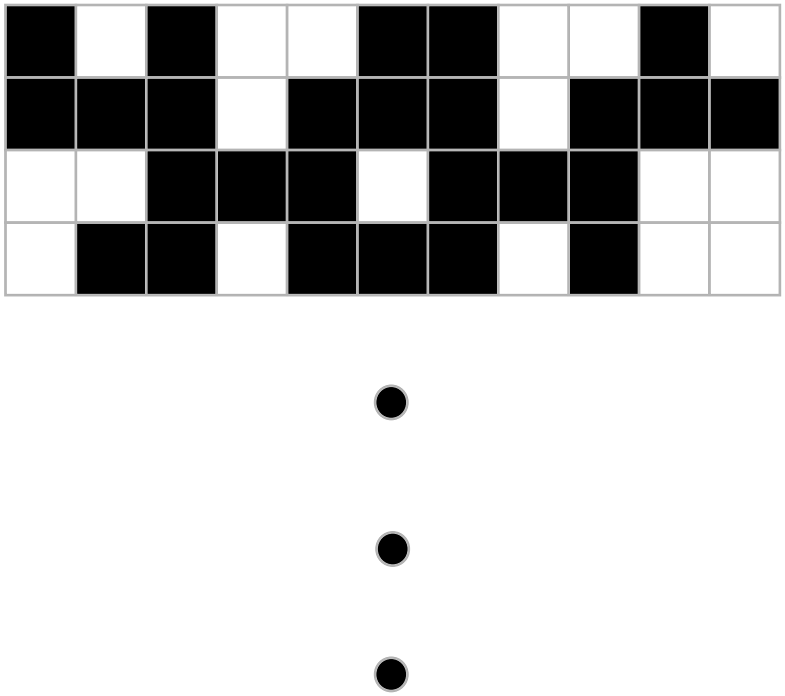
CAs are among the most common modeling tools in complex systems science!

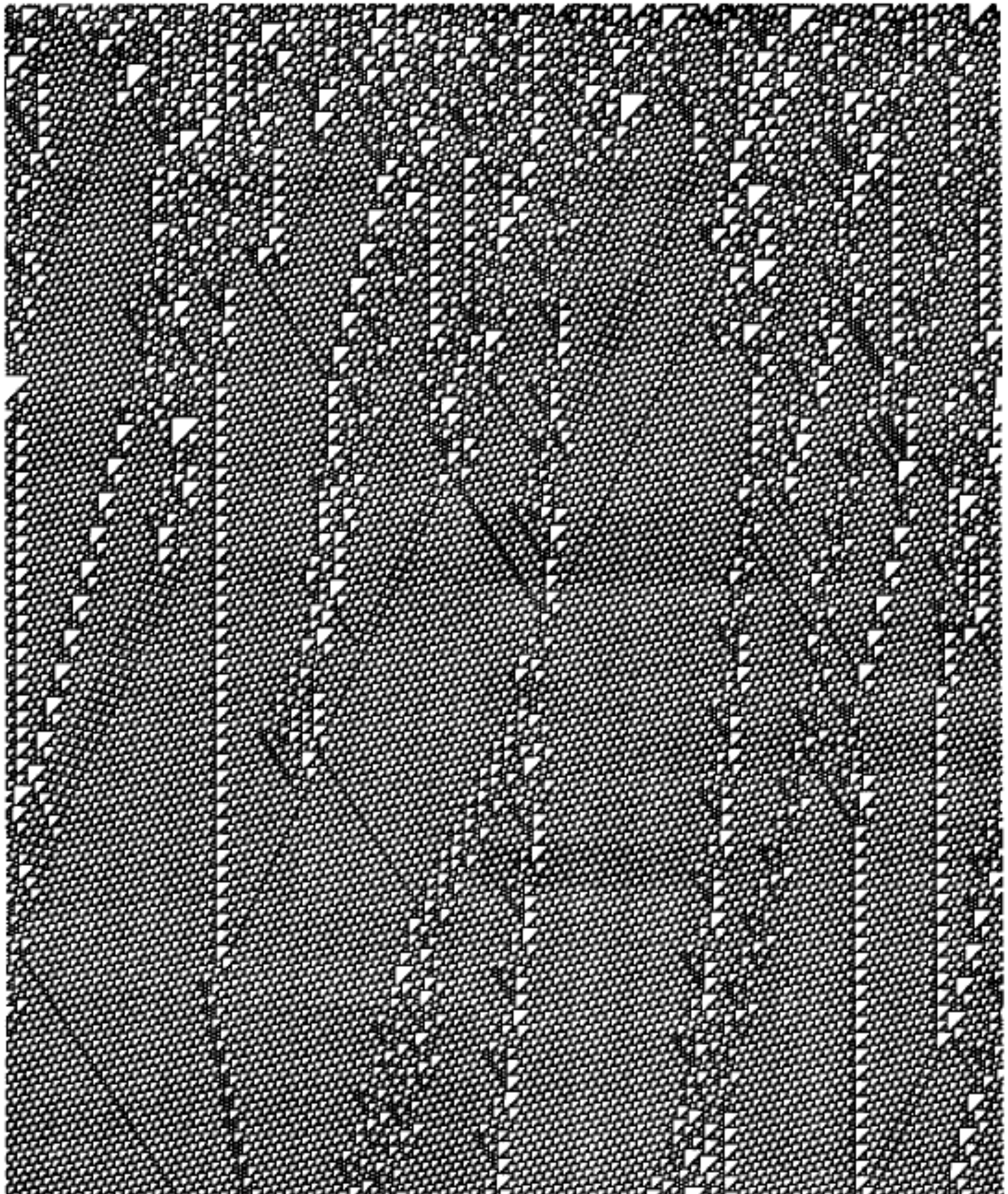
Elementary cellular automata

One-dimensional, two states (black and white)

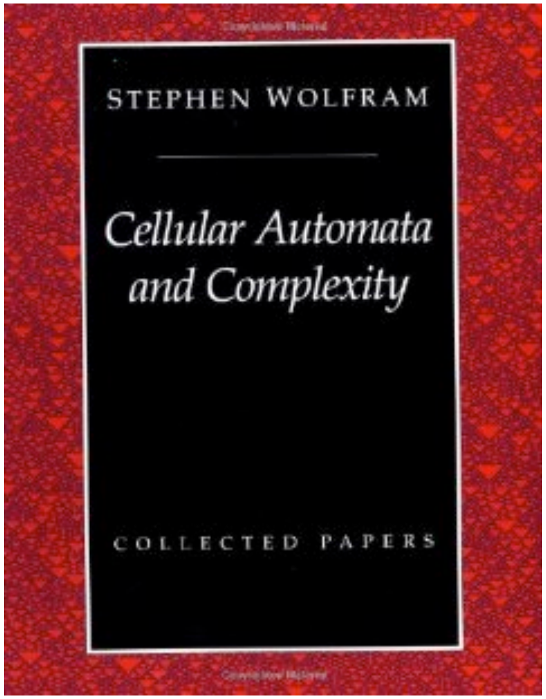
Rule:

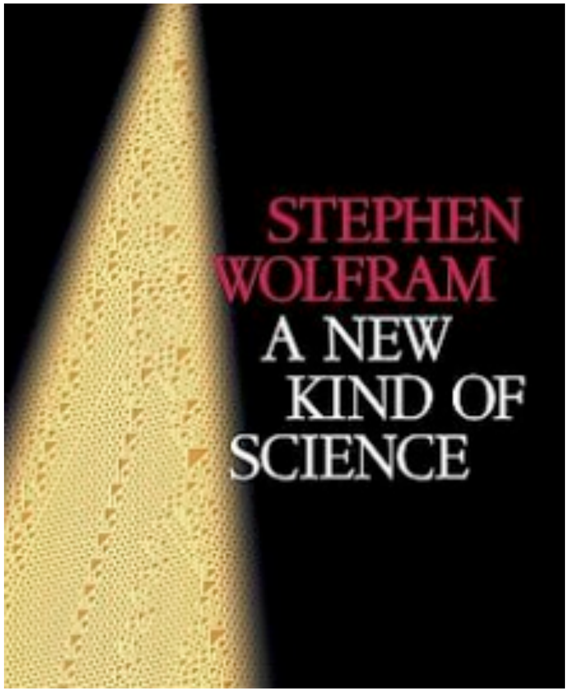






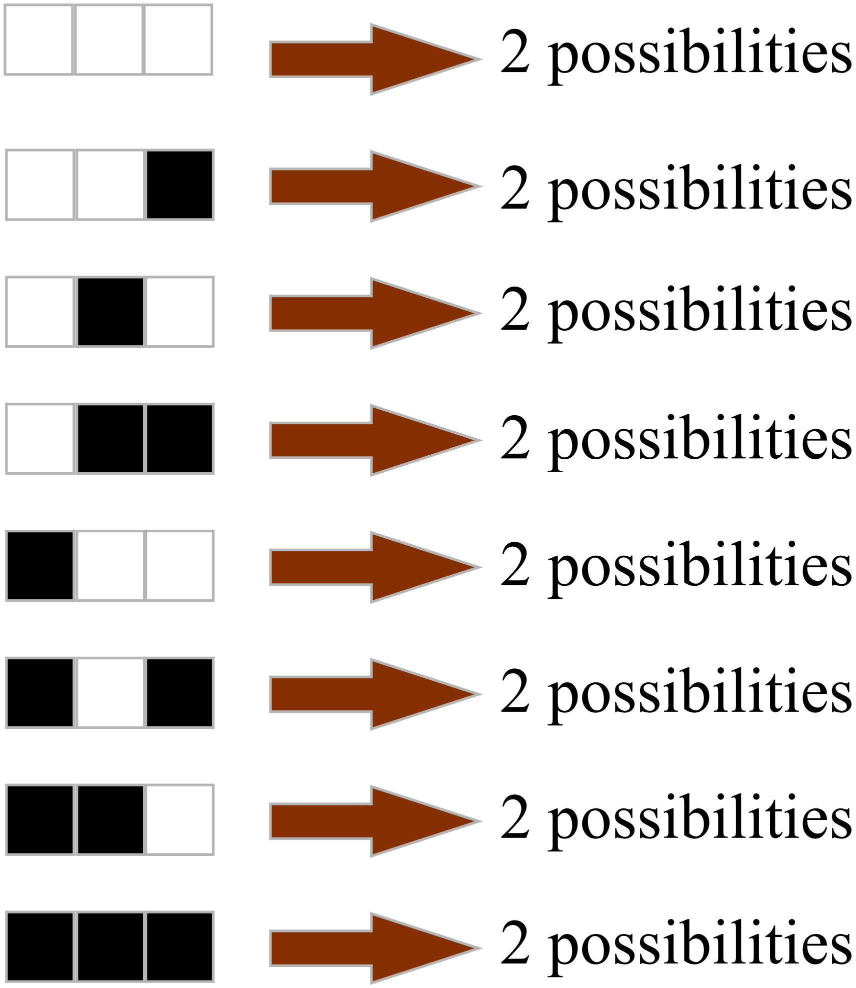
Stephen Wolfram





To define an ECA, fill in right side of arrows with black and white boxes:

Rule:



2 possibilities

2 possibilities

2 possibilities

2 possibilities

2 possibilities

2 possibilities

2 possibilities

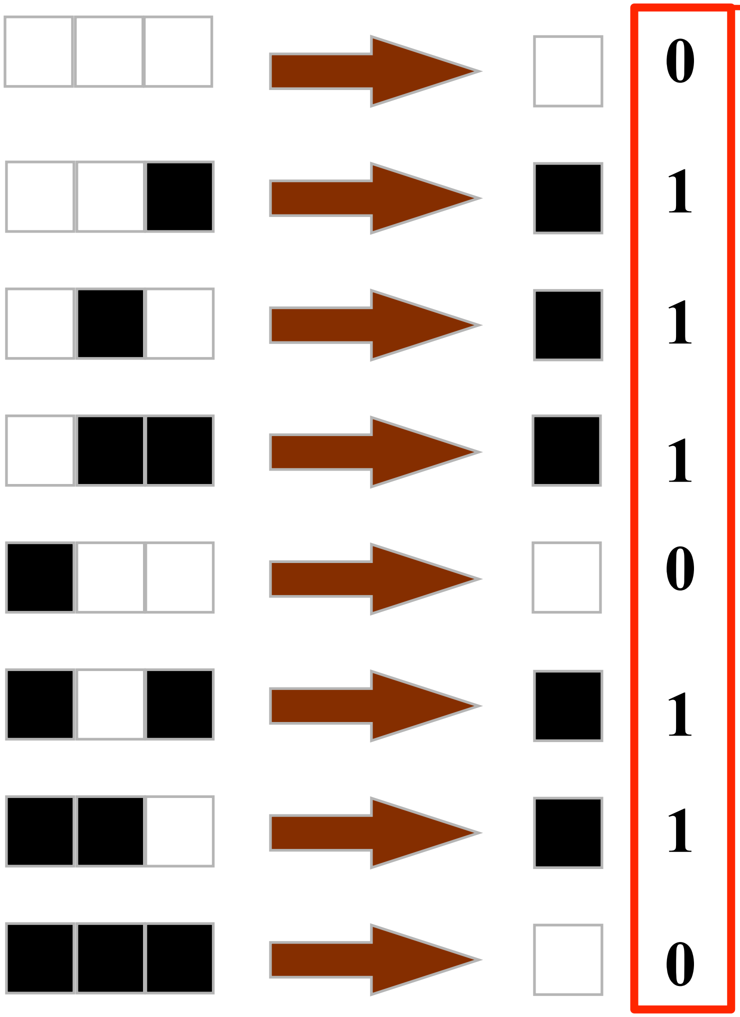
2 possibilities

Total: $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 2^8$

= 256 possible ECAs

Wolfram numbering:

Rule:



0 1 1 0 1 1 1 0

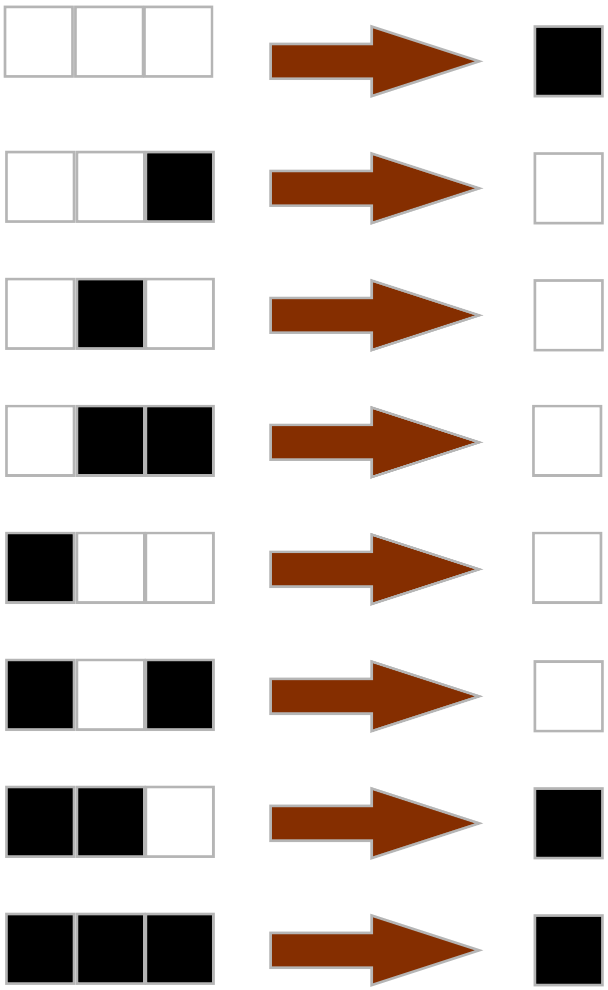
Interpret this as an integer in base 2:

$$(0\times 2^7)+(1\times 2^6)+(1\times 2^5)+(0\times 2^4)$$
$$+(1\times 2^3)+(1\times 2^2)+(1\times 2^1)+(0\times 2^0)$$
$$=110$$

"Rule 110"

Wolfram numbering:

Rule:



1

0

0

0

0

0

1

1

1 1 0 0 0 0 0 1

Interpret this as an integer in base 2:

$$(1\times 2^7)+(1\times 2^6)+(0\times 2^5)+(0\times 2^4)$$
$$+(0\times 2^3)+(0\times 2^2)+(0\times 2^1)+(1\times 2^0)$$
$$=128+64+1=193$$

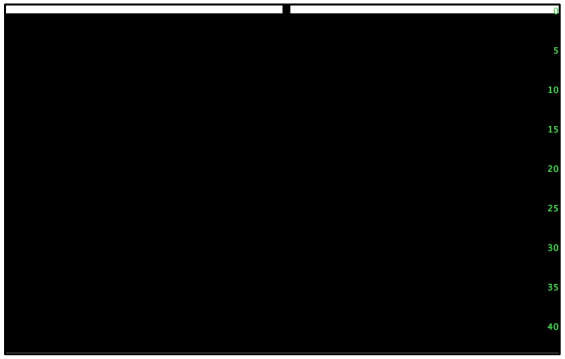
"Rule 193"

“The Rule 30 automaton is the most surprising thing I’ve ever seen in science....It took me several years to absorb how important this was. But in the end, I realized that this one picture contains the clue to what’s perhaps the most long-standing mystery in all of science: where, in the end, the complexity of the natural world comes from.”

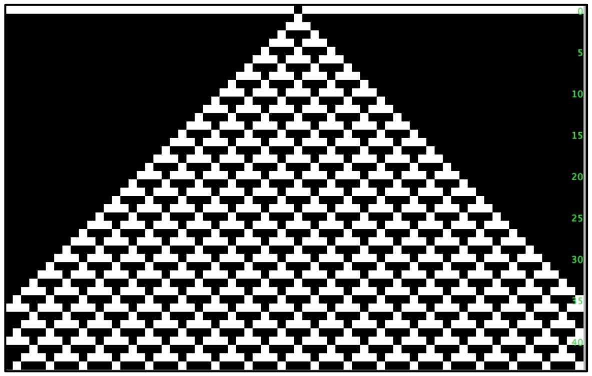
−−Stephen Wolfram (Quoted in *Forbes*)

Wolfram patented Rule 30’s use as a pseudo-random number generator!

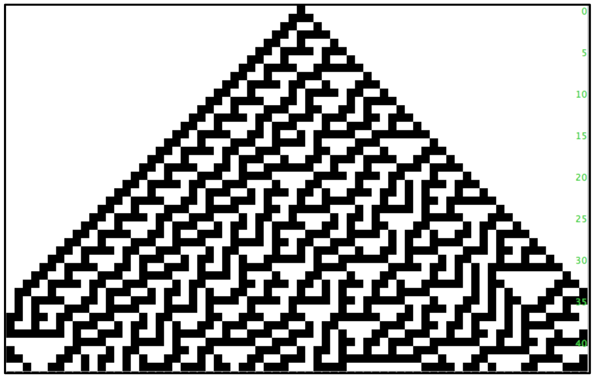
Wolfram's Four Classes of CA Behavior



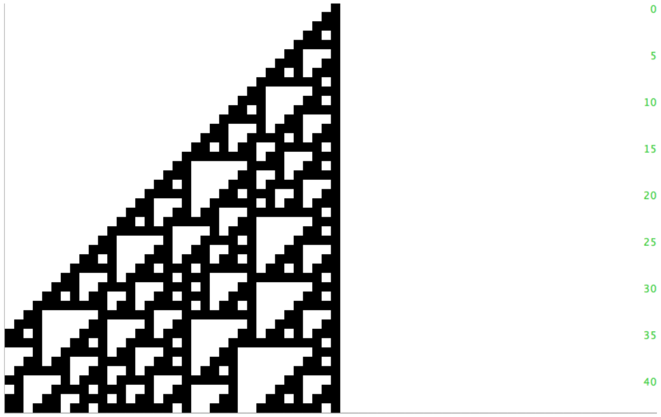
Class 1: Almost all initial configurations relax after a transient period to the same fixed configuration.



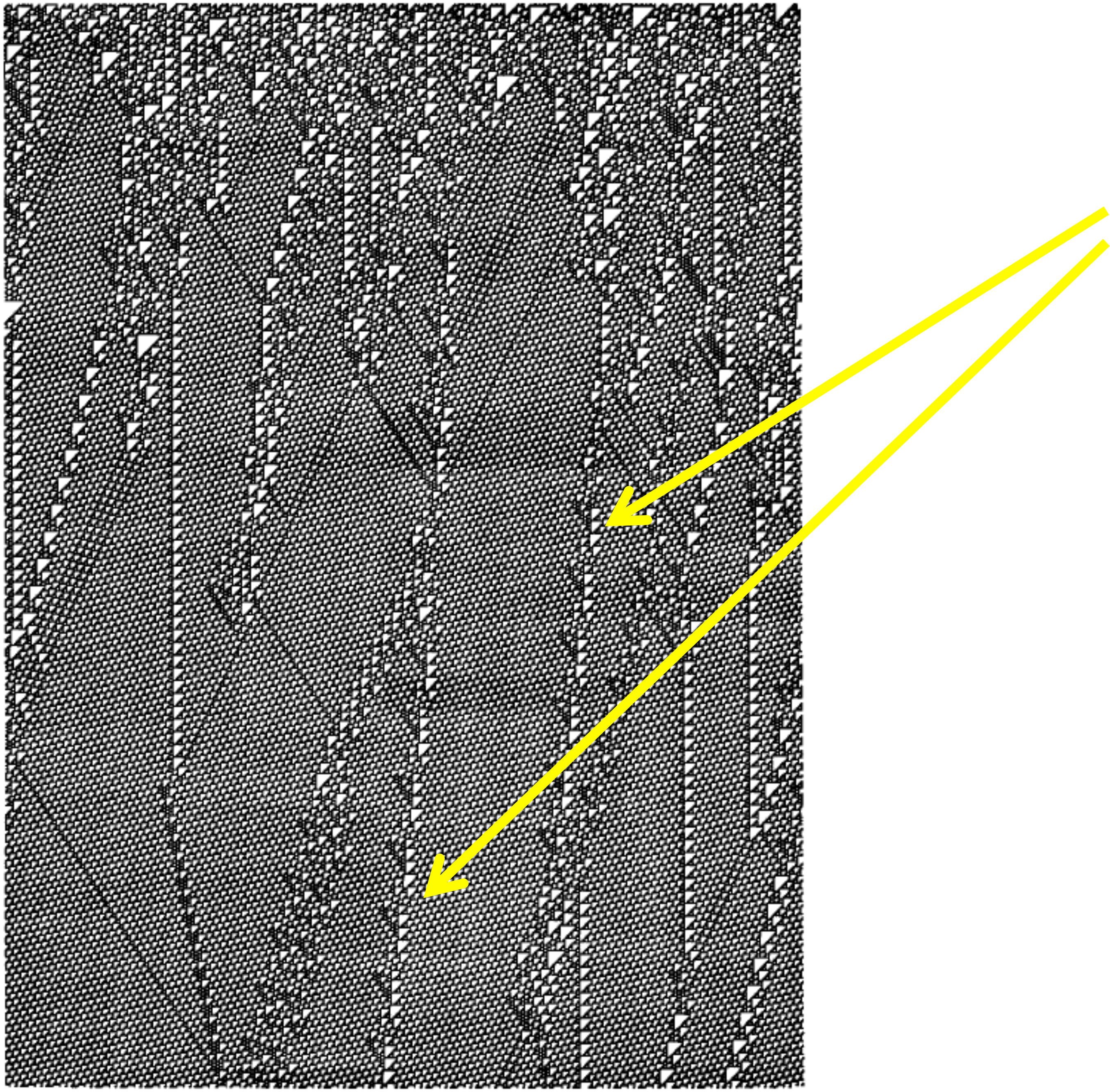
Class 2: Almost all initial configurations relax after a transient period to some fixed point or some periodic cycle of configurations, but which one depends on the initial configuration



Class 3: Almost all initial configurations relax after a transient period to chaotic behavior. (The term ``chaotic'‘ here refers to apparently unpredictable space-time behavior.)



Class 4: Some initial configurations result in complex localized structures, sometimes long-lived.



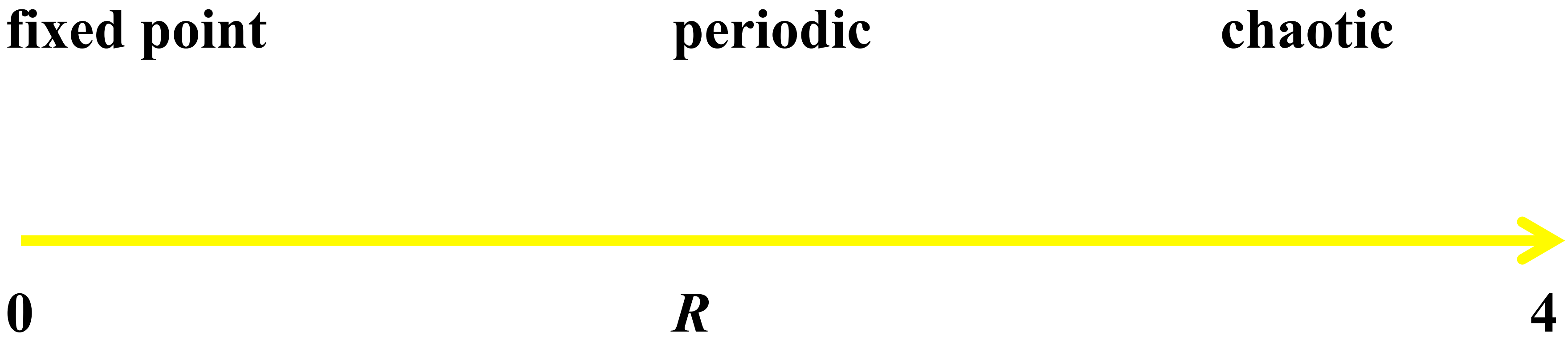
Examples of complex, long-lived localized structures

Rule 110

CAs as dynamical systems

(Analogy with logistic map)

Logistic Map	**Elementary Cellular Automata**
$x_{t+1} = f(x_t) = R\,x_t\left(1 - x_t\right)$	$lattice_{t+1} = f(lattice_t)$ [f = ECA rule)
Deterministic	Deterministic
Discrete time steps	Discrete time steps
Continuous “state” (value of *x* is a real number)	Discrete state (value of lattice is sequence of “black” and “white”)
Dynamics: Fixed point --- periodic ---- chaos	**Dynamics:** Fixed point – periodic – chaos
Control parameter: *R*	Control parameter: ?



fixed point periodic chaotic

0 R 4

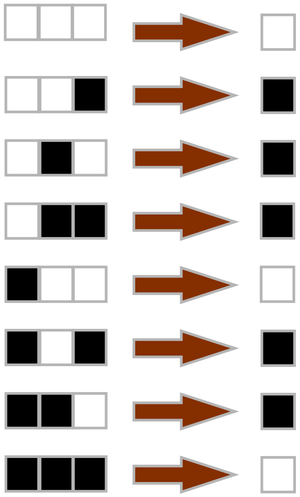
Langton's *Lambda* parameter as a proposed control parameter for CAs

Chris Langton

For two-state (black and white) CAs:

Lambda = fraction of black output states in CA rule table

For example:



□□□ → □

□□■ → ■

□■□ → ■

□■■ → ■

■□□ → □

■□■ → ■

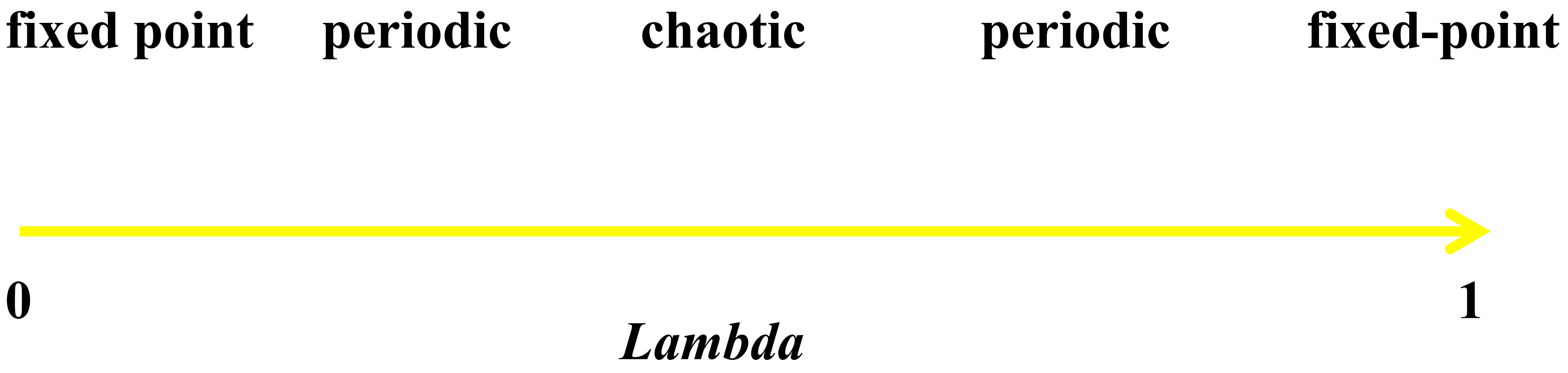
■■□ → ■

■■■ → □

Lambda = 5/8

Langton’s hypothesis:

“Typical” CA behavior (after transients):



fixed point periodic chaotic periodic fixed-point

0 **1**

Lambda

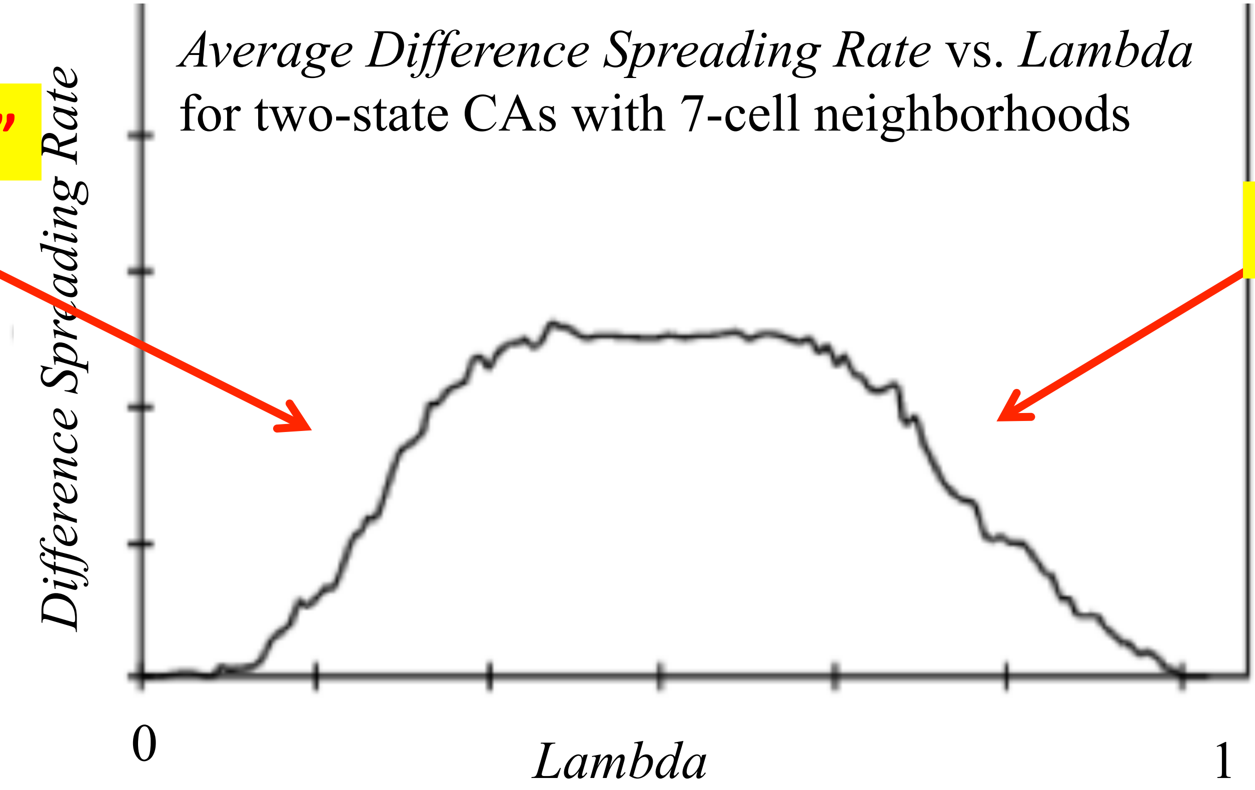
(for two-state CAs)

***Lambda* is a better predictor of behavior for neighborhood size > 3 cells**

“Edge of Chaos” applet

http://math.hws.edu/xJava/CA/EdgeOfChaosCA.html

From N. Packard,
“Adaptation Toward the Edge of Chaos”
1988



Average Difference Spreading Rate vs. *Lambda*
for two-state CAs with 7-cell neighborhoods

“Edge of chaos”

“Edge of chaos”

fixed point **periodic** **chaotic** **periodic** **fixed-point**

0 **1**

Summary

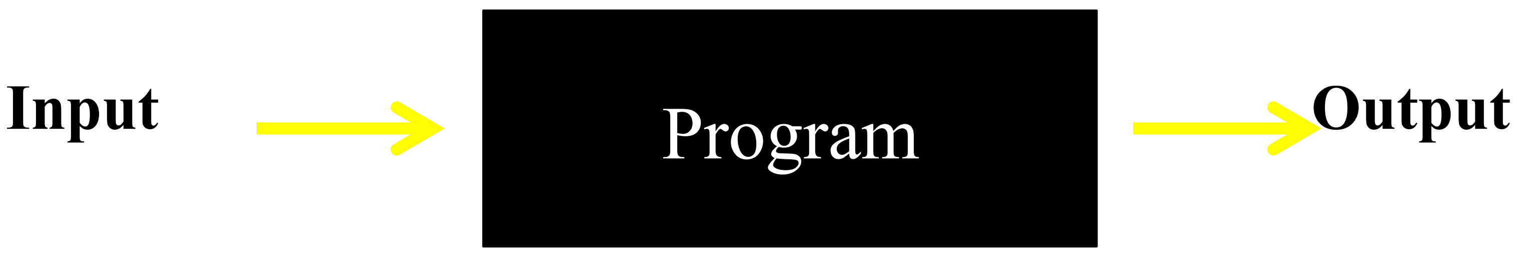
- CAs can be viewed as dynamical systems, with different attractors (fixed-point, periodic, chaotic, “edge of chaos”)
- These correspond to Wolfram’s four classes
- Langton’s *Lambda* parameter is one “control parameter” that (roughly) indicates what type of attractor to expect
- The Game of Life is a Class 4 CA!
- Wolfram hypothesized that Class 4 CAs are capable of “universal computation”

Computation: Information is

- input
- stored
- transferred
- combined (or “processed”)
- output

Computation: Information is

- input
- stored
- transferred
- combined (or "processed")
- output

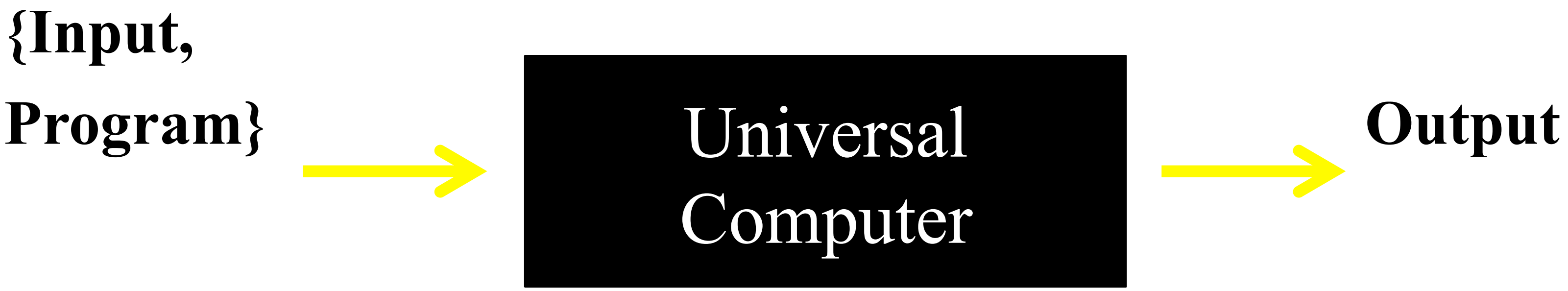


Input

Program

Output

Universal Computation (= *Programmable* Computers):



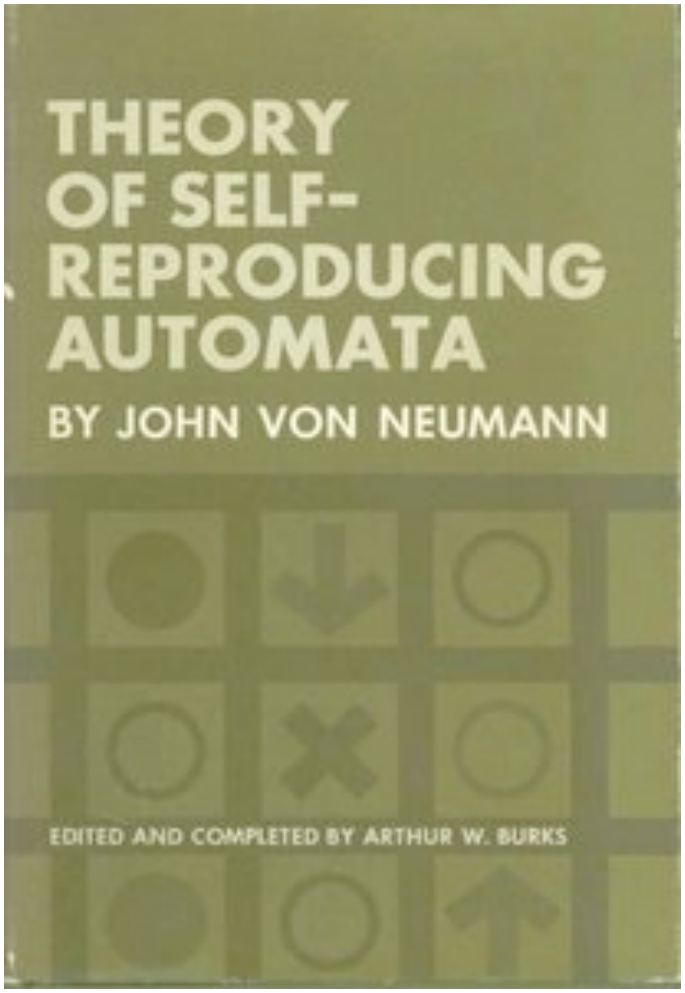
{Input,
Program}

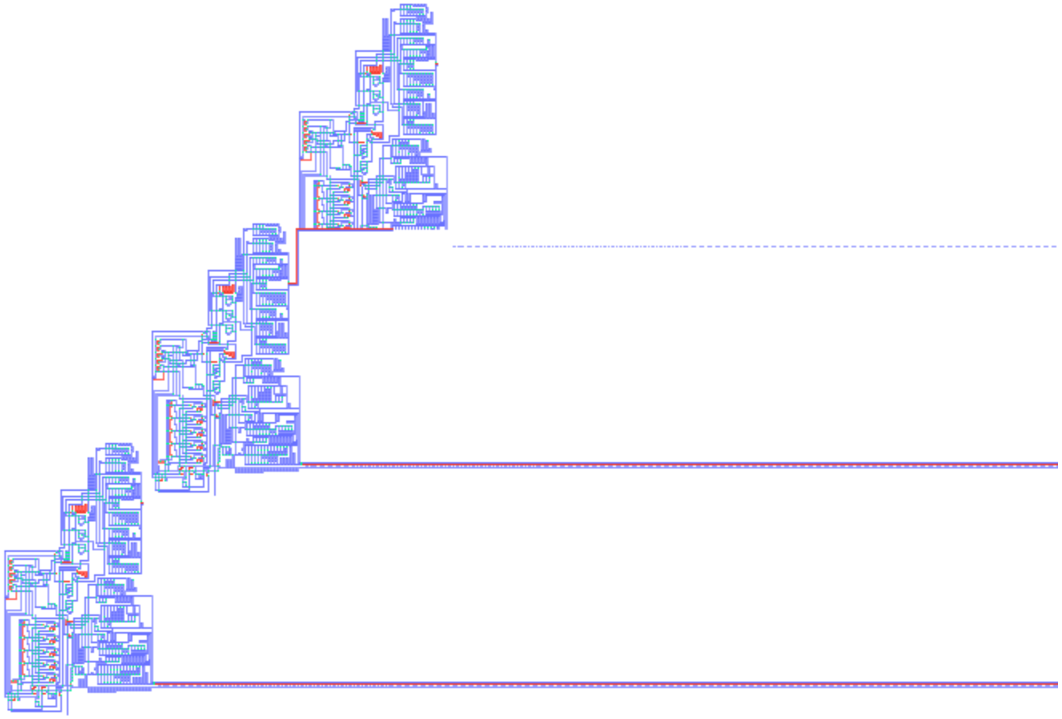
Universal
Computer

Output

Only a small set of logical operations is needed to support universal computation!

John von Neumann's Self-Reproducing Automaton


THEORY OF SELF-REPRODUCING AUTOMATA
BY JOHN VON NEUMANN
EDITED AND COMPLETED BY ARTHUR W. BURKS



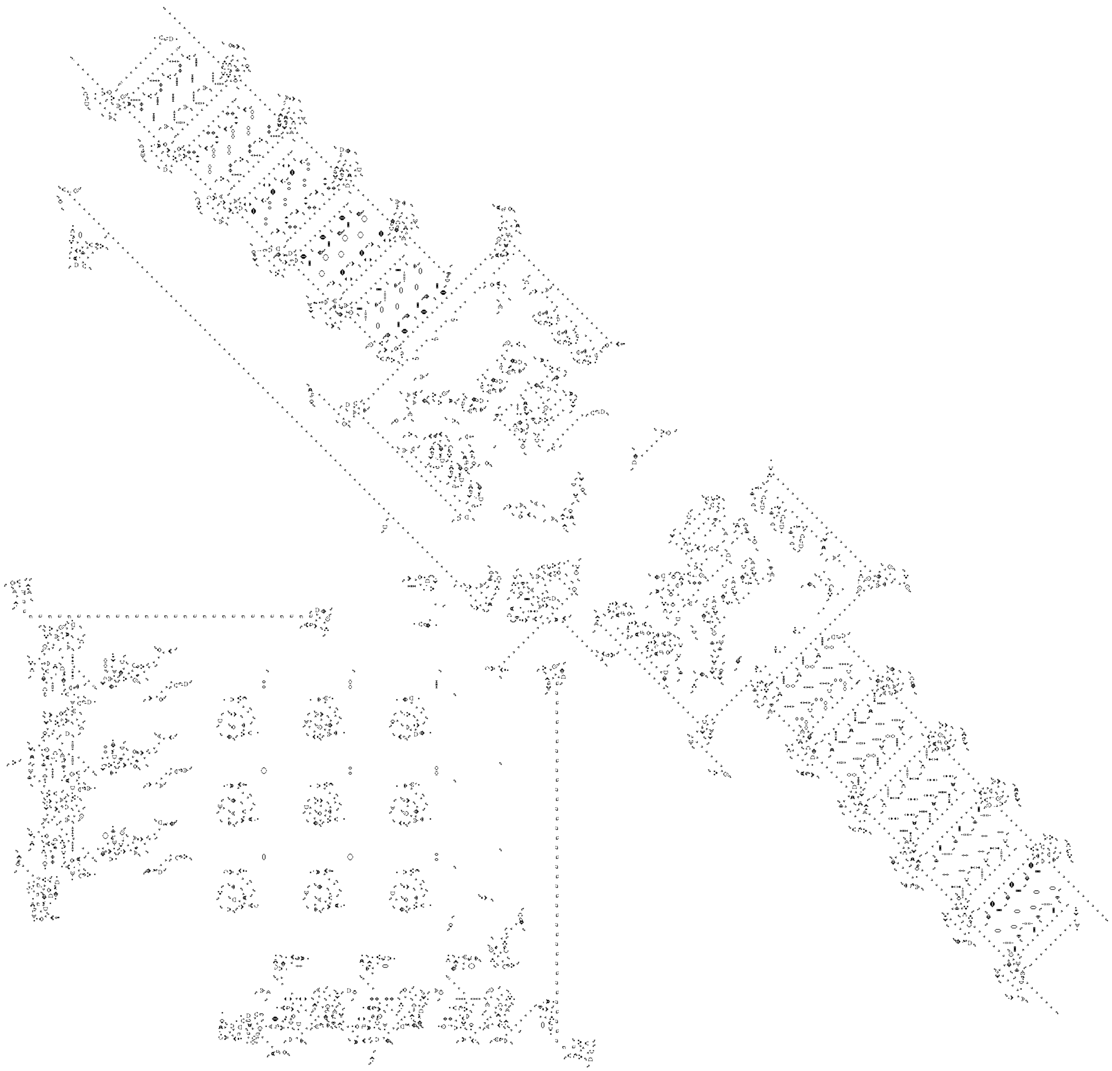
http://en.wikipedia.org/wiki/
File:Nobili_Pesavento_2reps.png

Two dimensional cellular automaton, 29 states. Universal replicator and computer.

The Game of Life as a Universal Computer

1970: Conway shows that *Life* can implement simple logic operations needed for universal computation, and sketches how a universal computer could be constructed.

1990s: Paul Rendall constructs universal computer in *Life*.



http://rendell-attic.org/gol/turing_js_r.gif

Computation in ECAs

Wolfram's hypothesis:

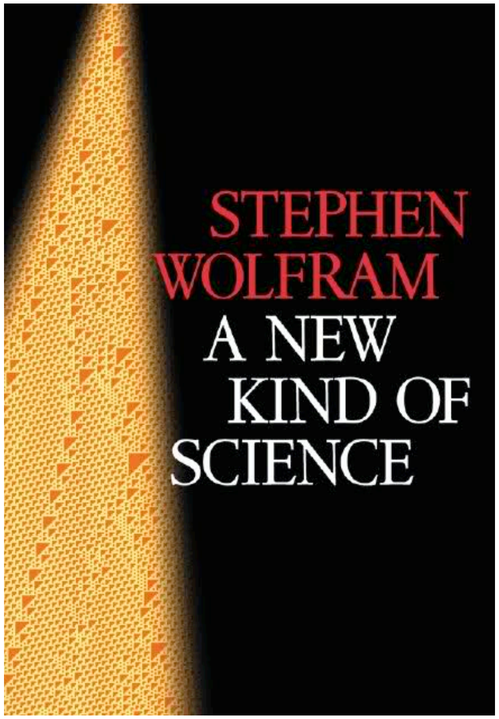
All class 4 CAs can support universal computation

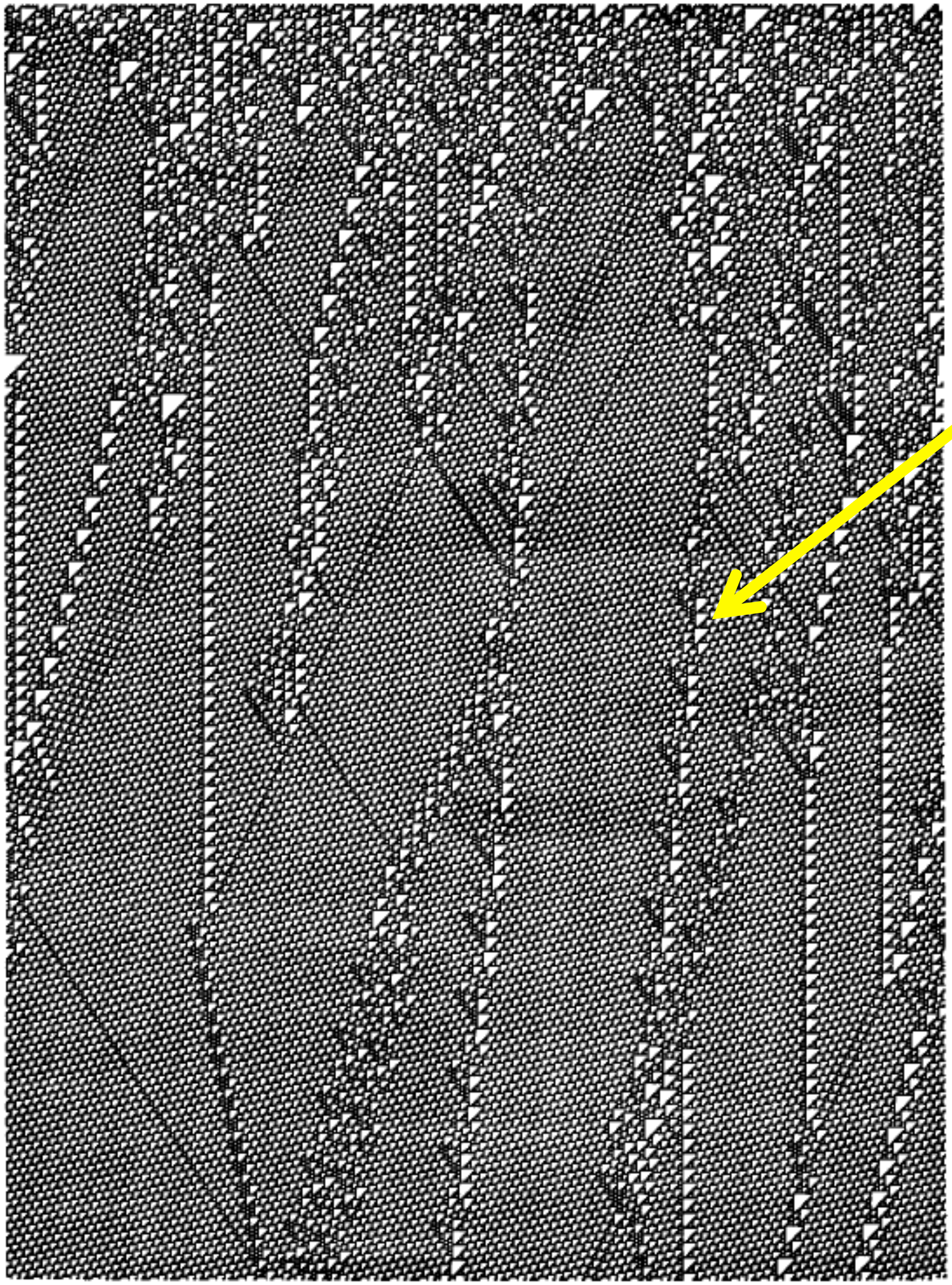
This hypothesis is hard to evaluate:

- No formal definition of class 4 CAs
- Hard to prove that something is capable of universal computation

Rule 110 as a Universal Computer

- Proved by Matthew Cook, 2002
- Described in





– Transfer of information: *moving particles*

– Integration of information from different spatial locations: *particle collisions*

From http://www.stephenwolfram.com/publications/articles/ca/86-caappendix/16/text.html

“Useful computation” in CAs

- Universal computation in CAs, while interesting and surprising, is not very practical.
  - Too slow, too hard to program.

- CAs have been harnessed for more practical parallel computation (e.g. image processing).

- Next subunit – evolving CAs with GAs to perform such computations.

Significance of CAs for Complex Systems

- Cellular automata can produce highly complex behavior from simple rules

- Natural complex systems can be modeled using cellular-automata-like architectures

- CAs give an framework for understanding how complex dynamics can produce collective information processing in a “life-like” system.

Evolving Cellular Automata with Genetic Algorithms: A Review of Recent Work

Melanie Mitchell
Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM 87501
mm@santafe.edu

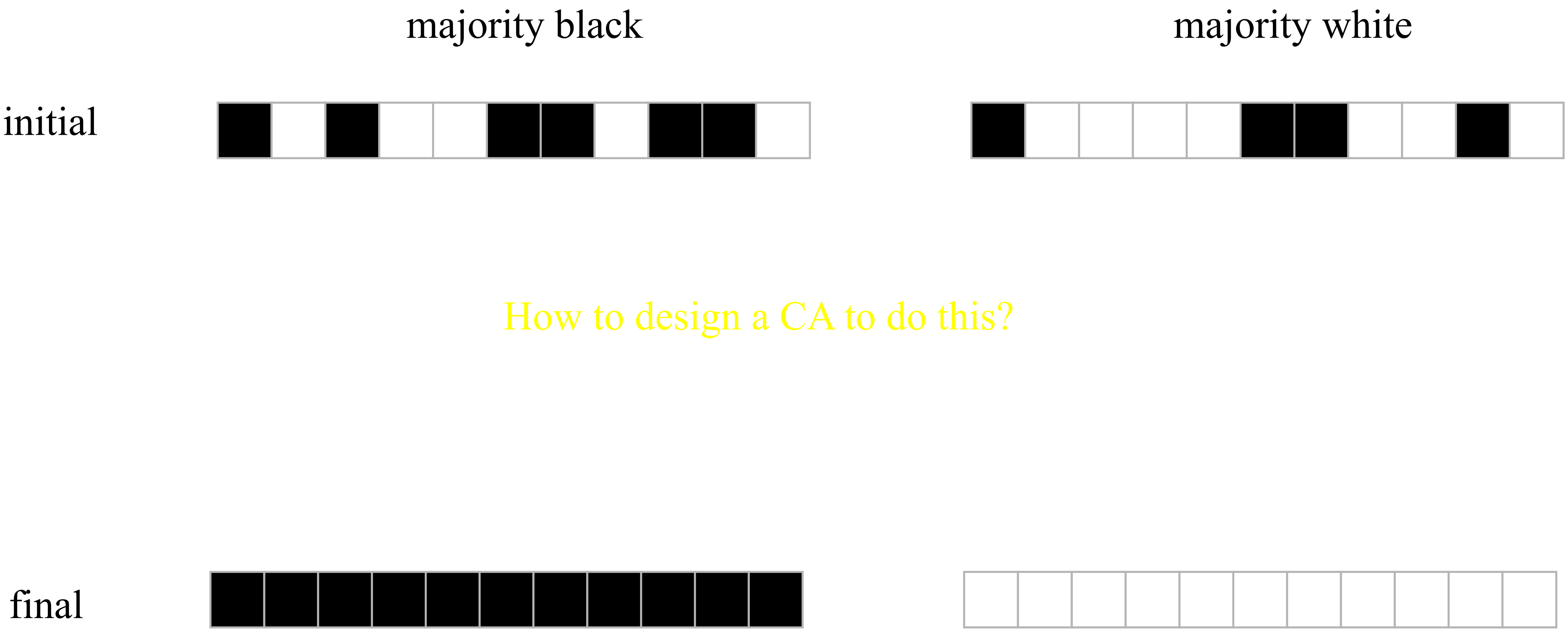
James P. Crutchfield[1]
Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM 87501
jpc@santafe.edu

Rajarshi Das
IBM Watson Research Ctr.
P.O. Box 704
Yorktown Heights, NY 10598
rajarshi@watson.ibm.com

In *Proceedings of the First International Conference on Evolutionary Computation and Its Applications (EvCA'96)*. Moscow, Russia: Russian Academy of Sciences, 1996.

A computational task for cellular automata

Design a cellular automaton to decide whether or not the initial pattern has a majority of black cells.



majority black

majority white

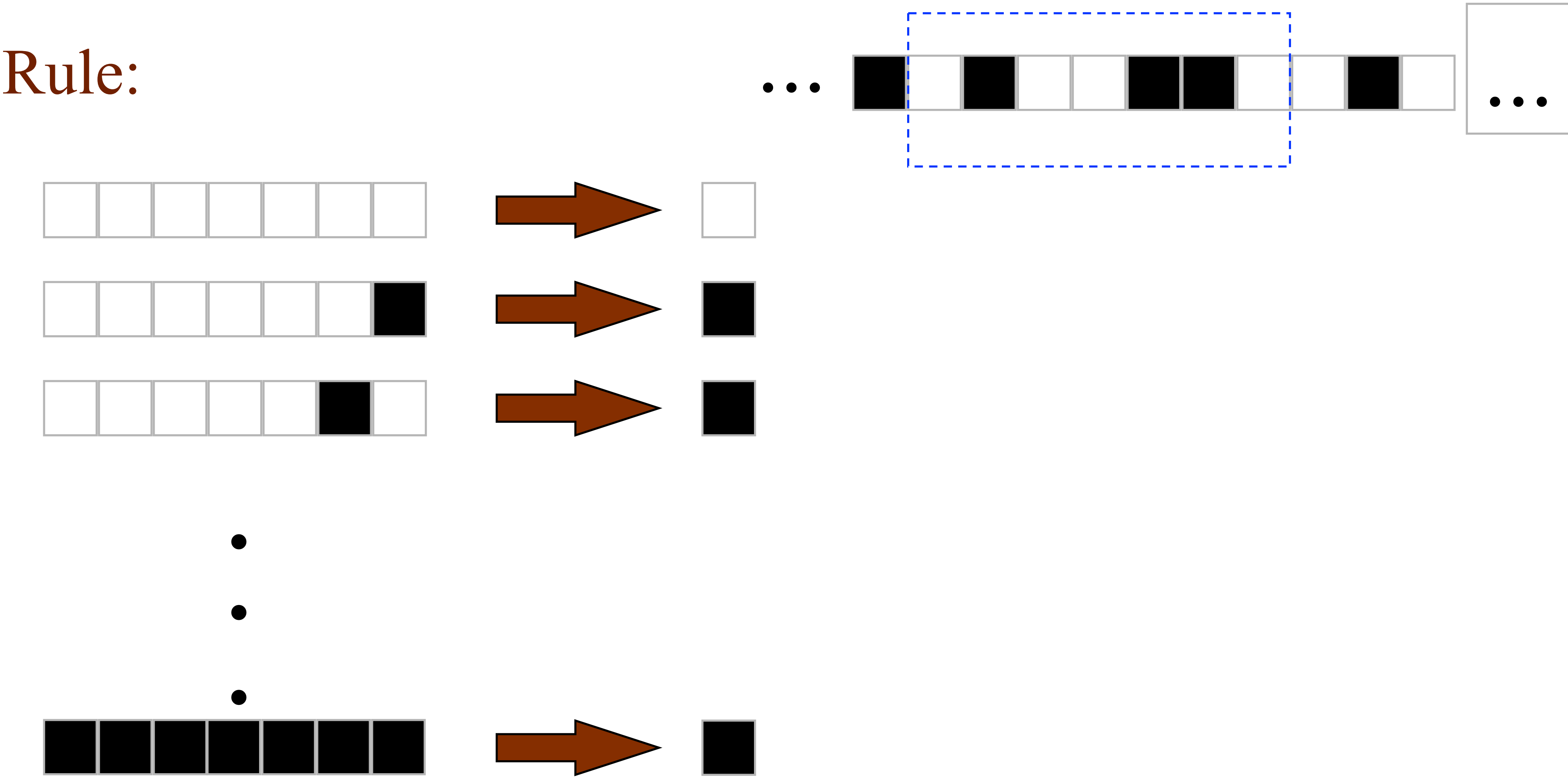
initial

How to design a CA to do this?

final

We used cellular automata with 6 neighbors for each cell:

Rule:



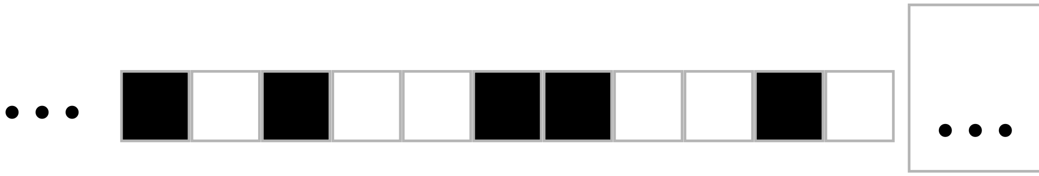
Quiz

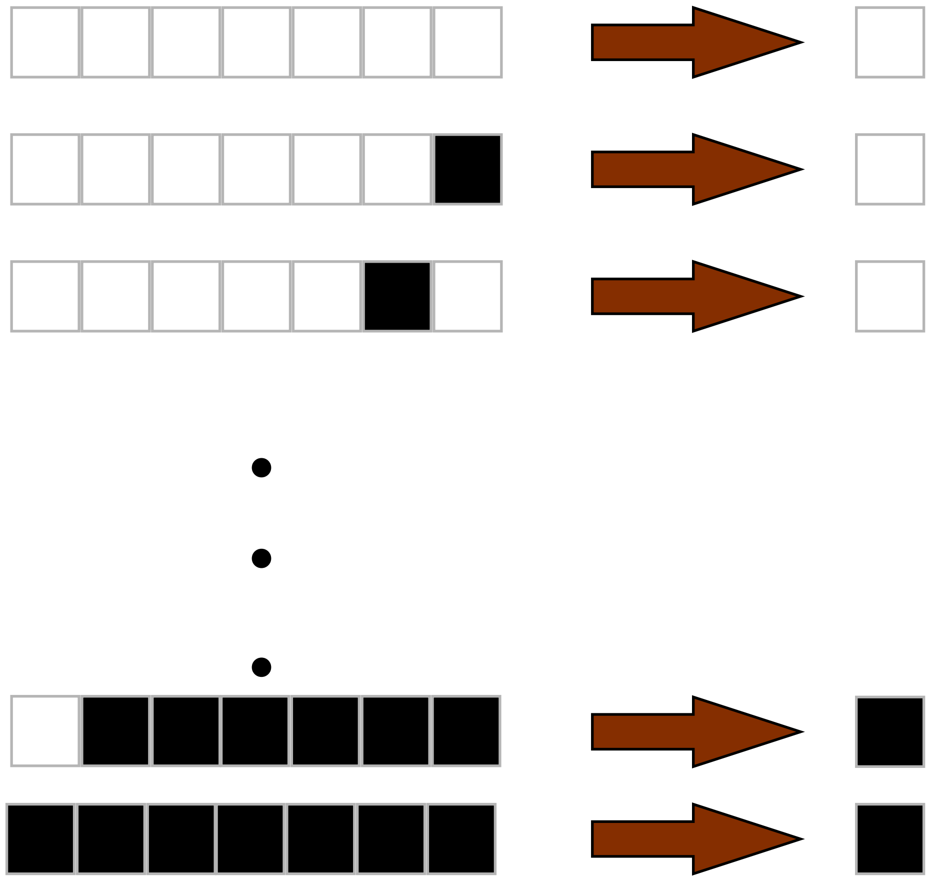
- how many neighborhoods?

- how many CAs

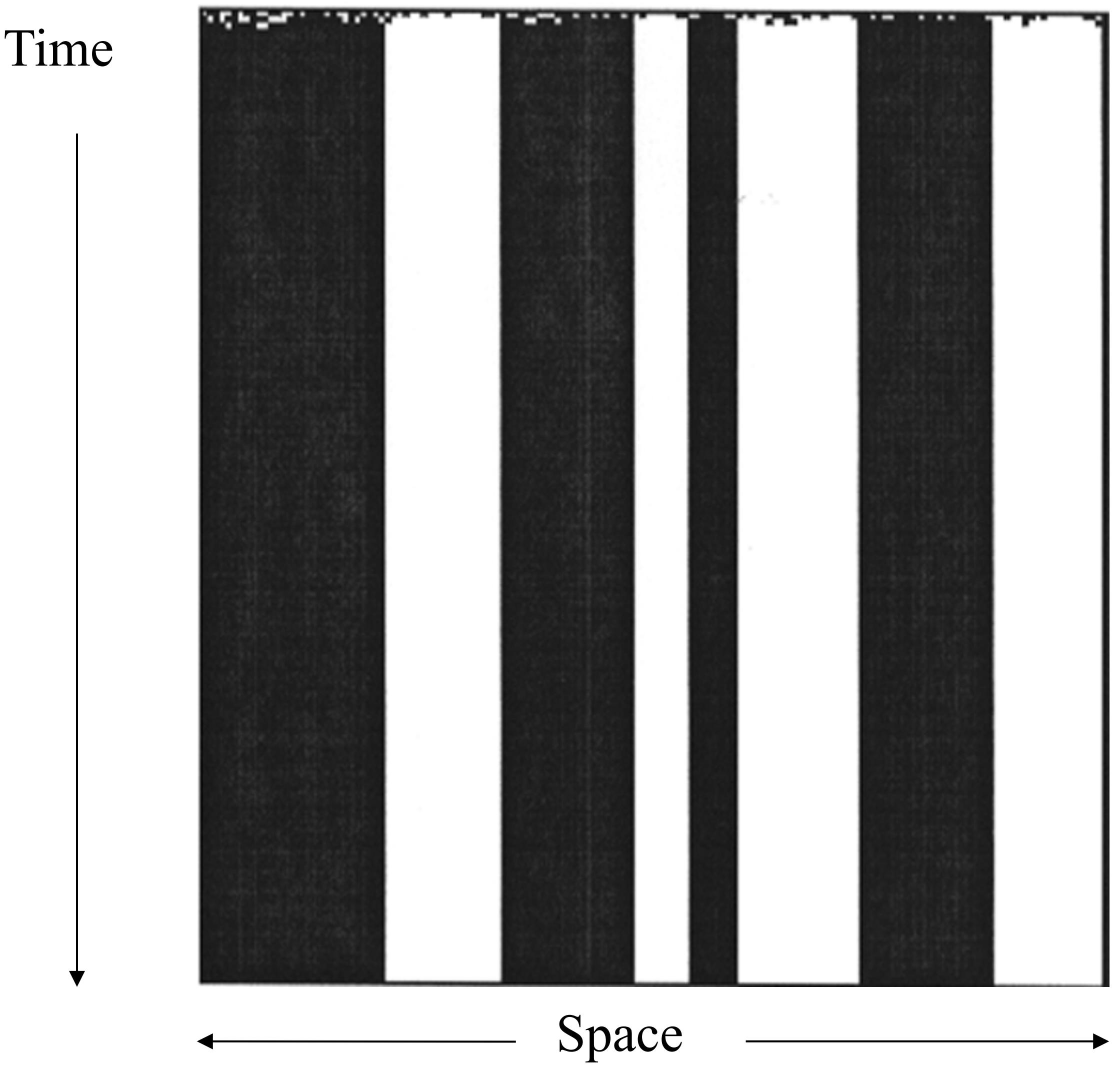
Naive Solution: Majority vote in each neighborhood

Rule:





Results of local majority voting CA: It doesn’t perform the task!



Evolving cellular automata with genetic algorithms

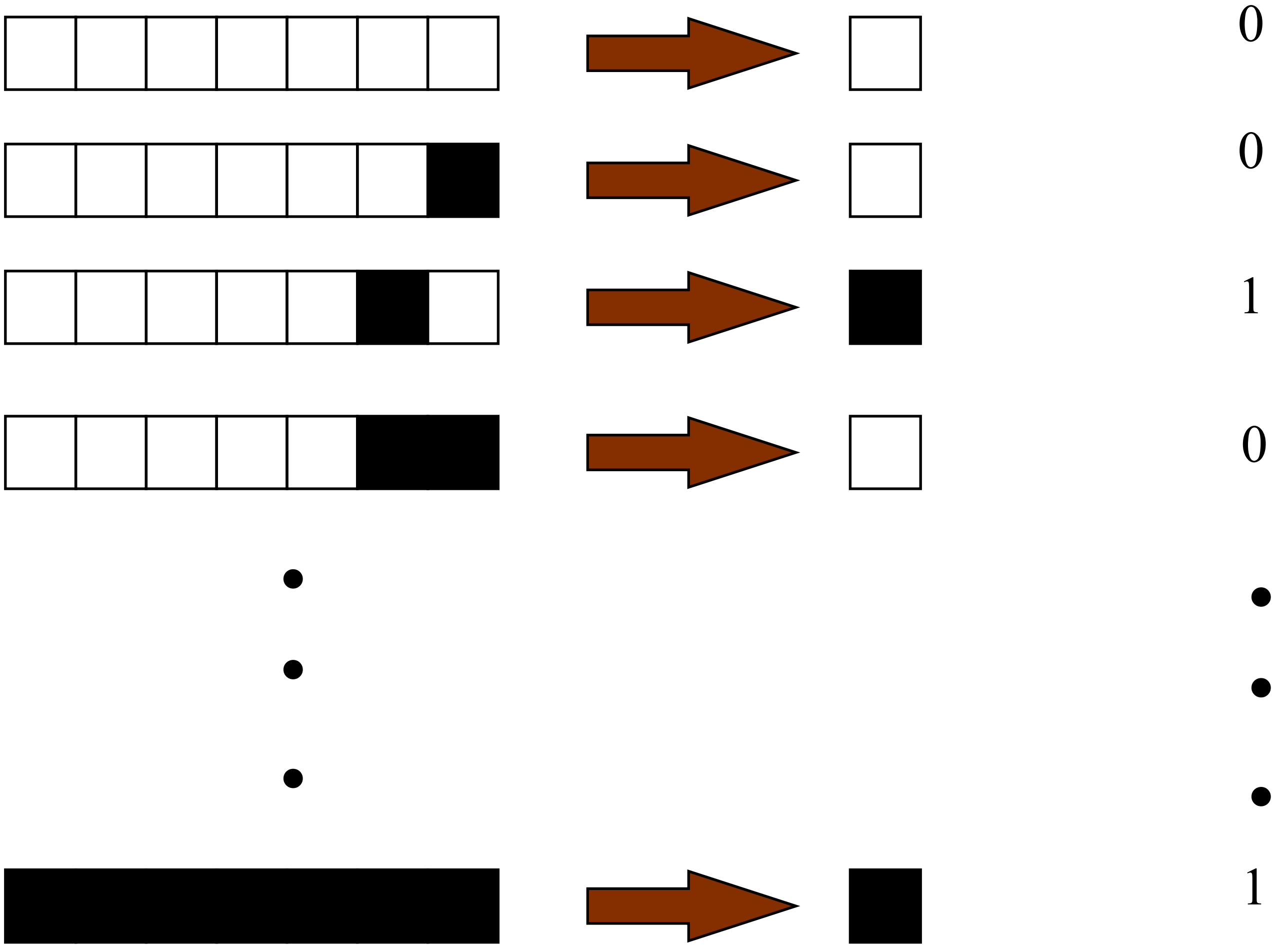
- Create a random population of candidate cellular automata rules.

- The “fitness” of each cellular automaton is how well it performs the task.

- The fittest cellular automata get to reproduce themselves, with mutations and crossovers.

- This process continues for many generations.

The “DNA” of a cellular automaton is an encoding of its rule table:



Create a random population of candidate cellular automata rules:

rule 1: 0010001100010010111100010100110111000...
rule 2: 0001100110101011111111000011101001010...
rule 3: 1111100010010101000000011100010010101...

.

.

.

rule 100: 0010111010000001111100000101001011111...

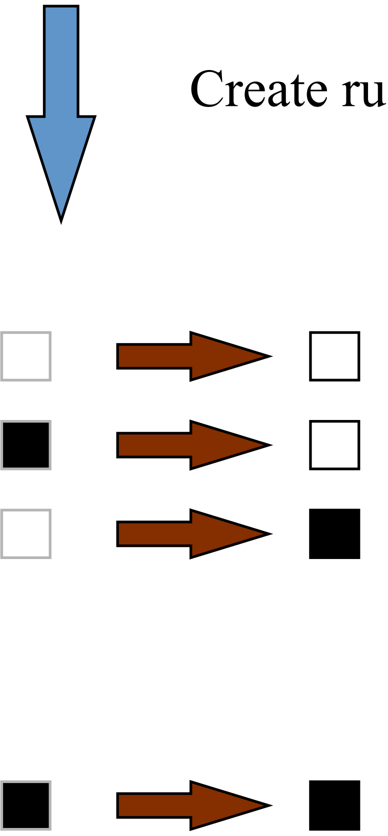
Calculating the Fitness of a Rule

- For each rule, create the corresponding cellular automaton. Run that cellular automaton on many initial configurations.

- Fitness of rule = fraction of correct classifications

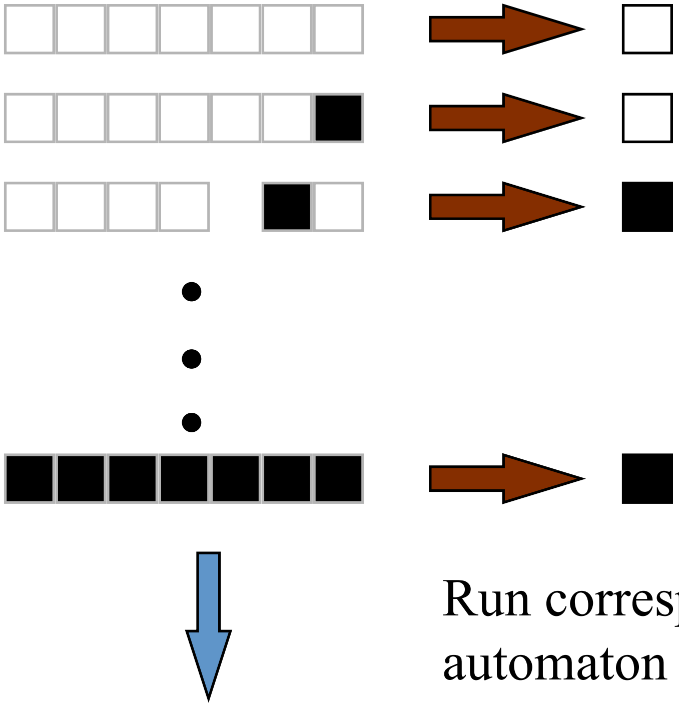
For each cellular automaton rule in the population:

rule 1: 0010001100010010111100010100110111000...1

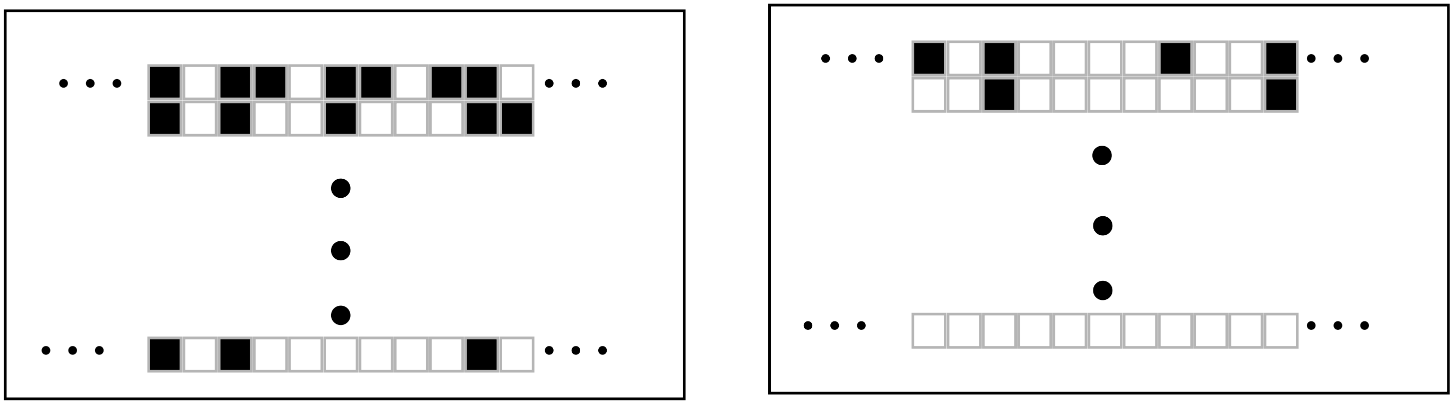


Create rule table

rule 1 rule table:



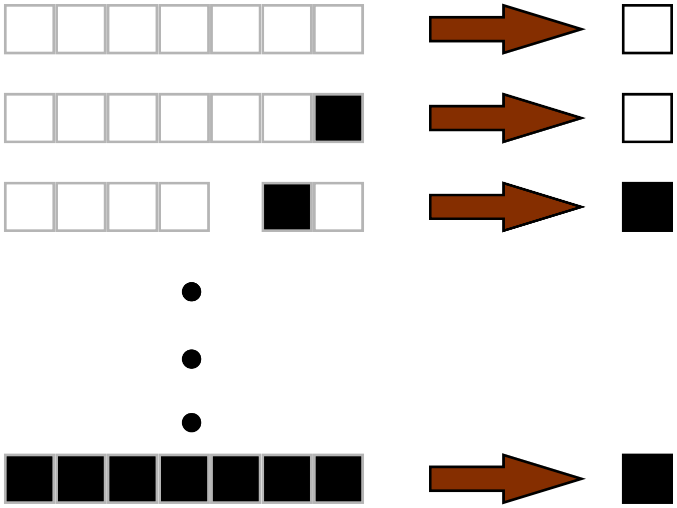
Run corresponding cellular automaton on many random initial lattice configurations



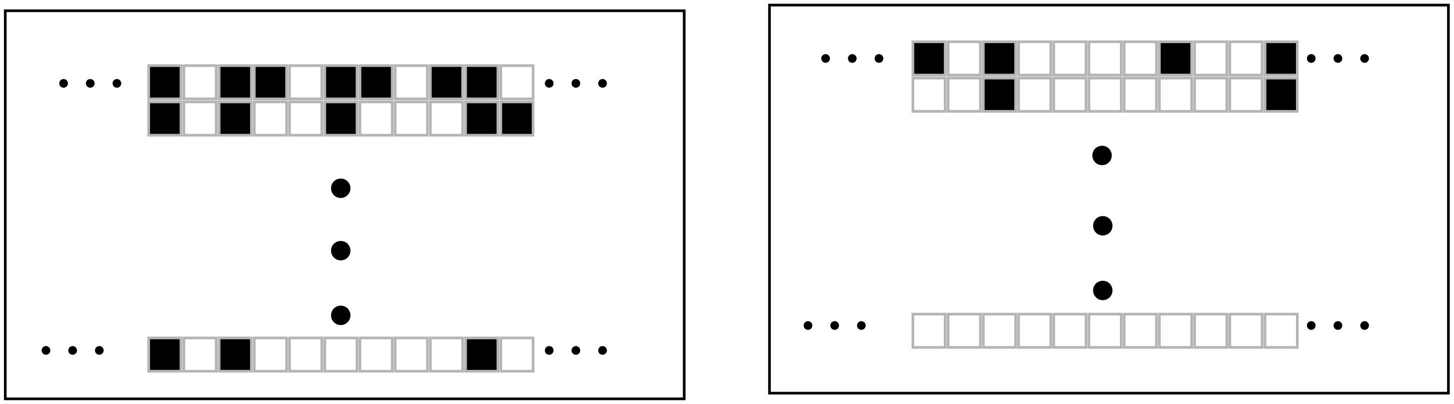
incorrect

correct

rule 1 rule table:



Run corresponding cellular automaton on many random initial lattice configurations

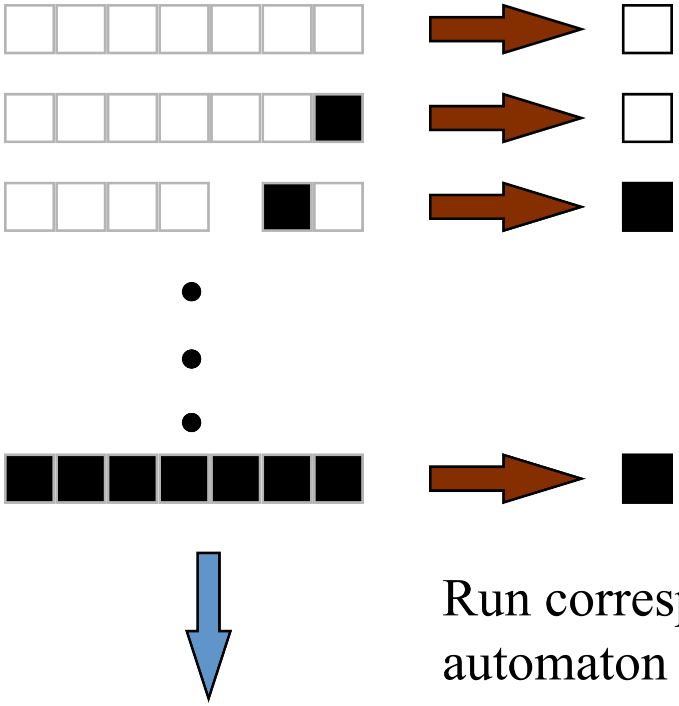


incorrect

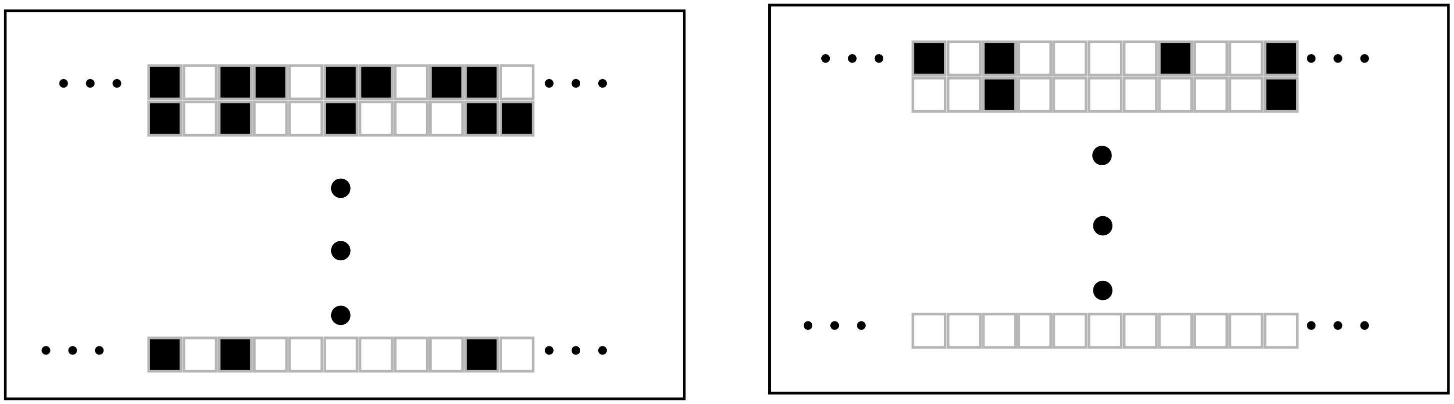
correct

etc.

rule 1 rule table:



Run corresponding cellular automaton on many random initial lattice configurations



incorrect

correct

etc.

Fitness of rule = fraction of correct classifications

GA Population:

rule 1:   0010001100010010111100010100110111000... Fitness = 0.5
rule 2:   0001100110101011111111000011101001010... Fitness = 0.2
rule 3:   11111000100101010000000011100010010101... Fitness = 0.4

.

.

.

rule 100:00101110100000011111000000101001011111... Fitness = 0.0

Select fittest rules to reproduce themselves

rule 1:   0010001100010010111100010100110111000... Fitness = 0.5
rule 3:   11111000100101010000000011100010010101... Fitness = 0.4

etc.

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 × 100010010111100010100110111000...
rule 3: 1111100 010010101000000011100010010101...

Children:

Mutate:

0010001010010$\boxed{1}$01000000011100010010101...

1111100100010010111100010100110111000..

Create new generation via crossover and mutation:

Parents:

rule 1: 0010001 ╳ 100010010111100010100110111000...
rule 3: 1111100 010010101000000011100010010101...

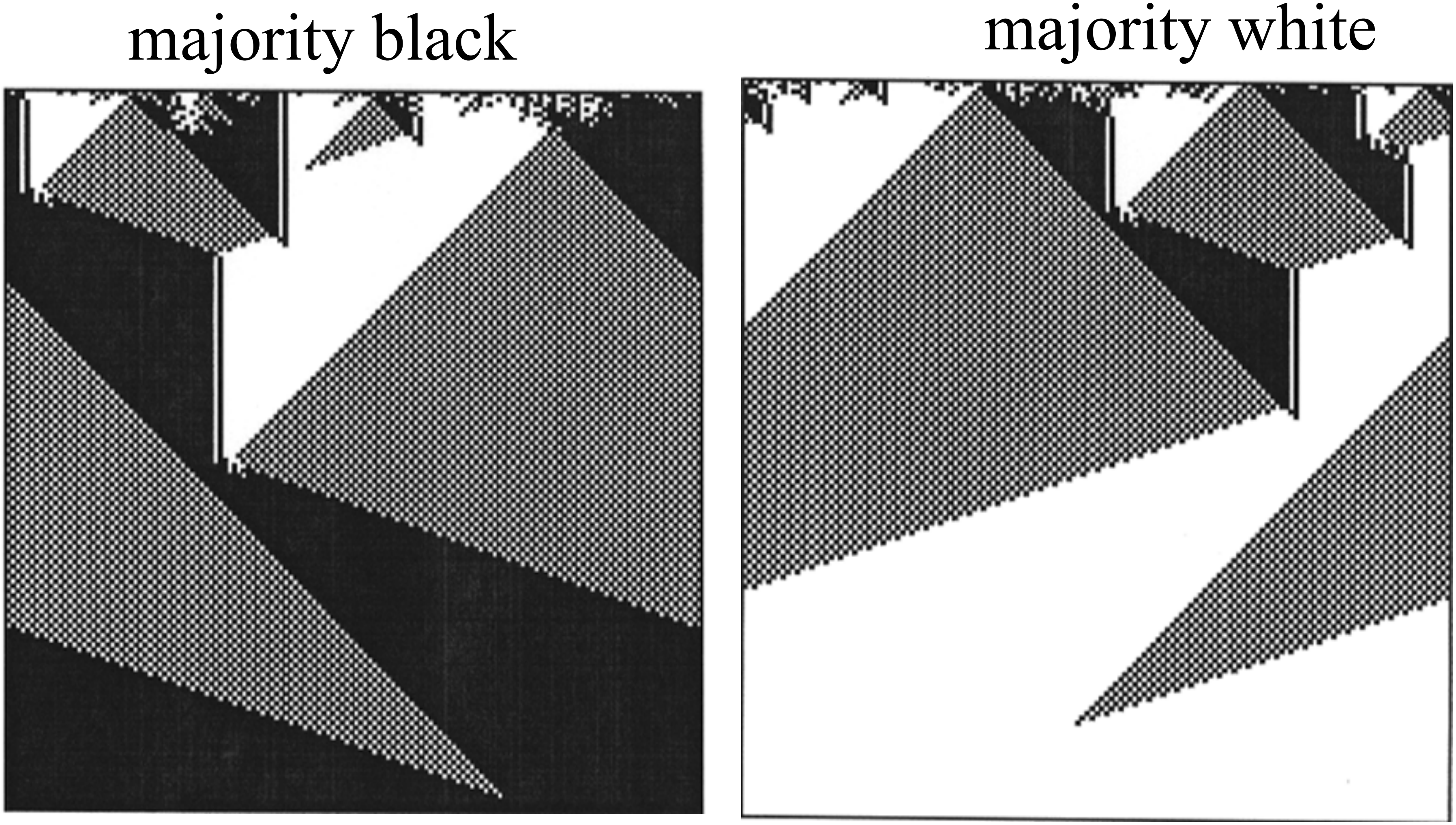
Children:

0010001010010001000000011100010010101...

1111100100010010111100010100110111000..

Continue this process until new generation is complete.
Then start over with the new generation.

Keep iterating for many generations.

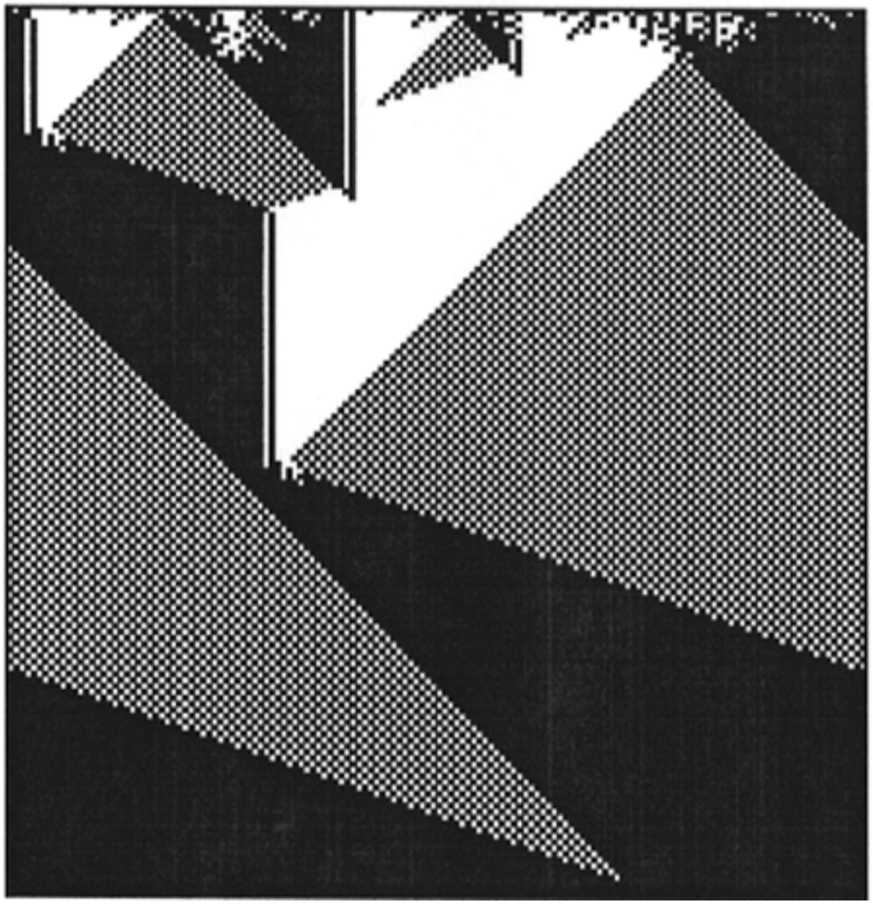


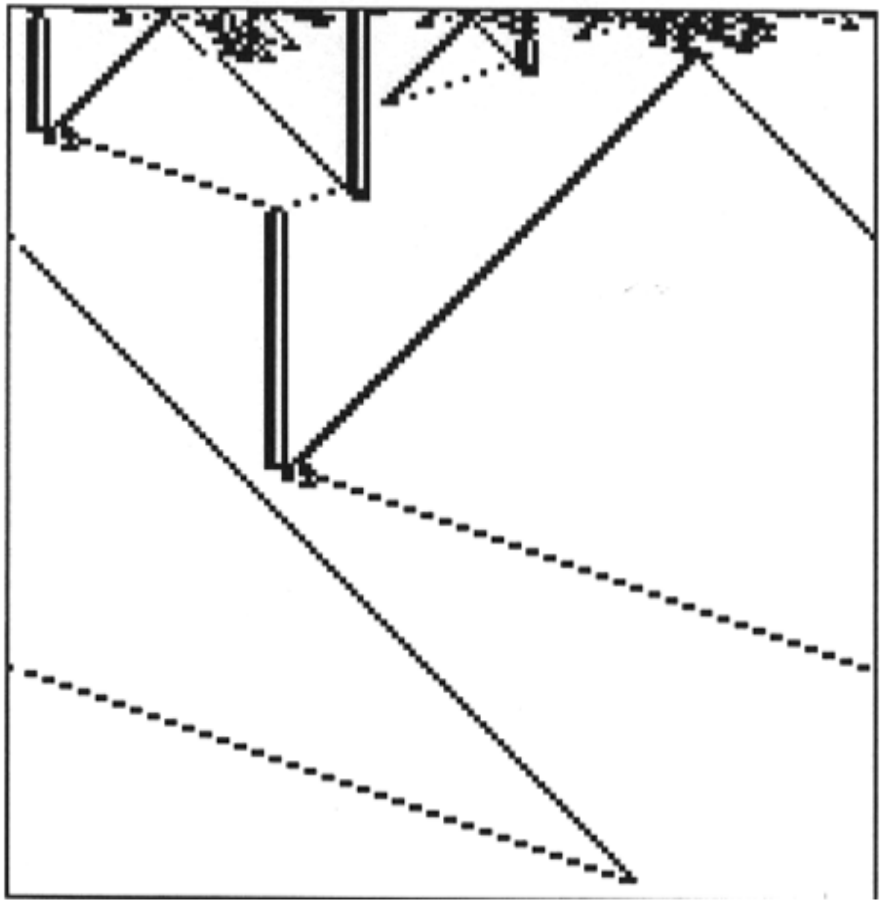
majority black

majority white

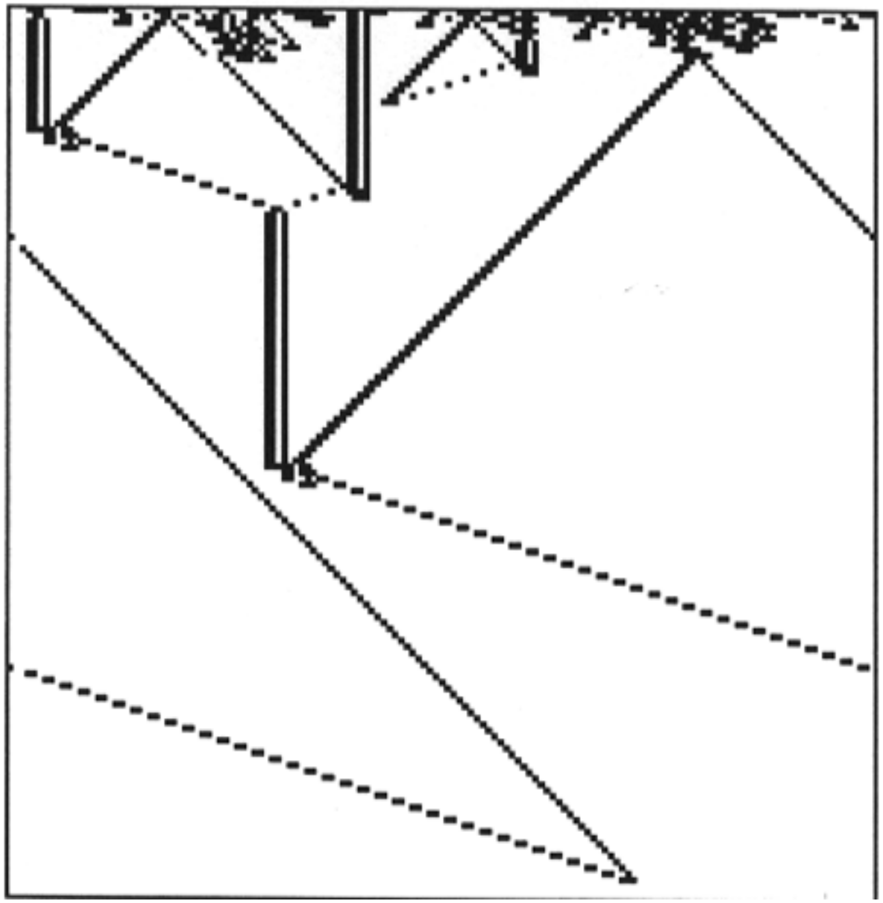
A cellular automaton evolved by the genetic algorithm

How do we describe information processing in complex systems?

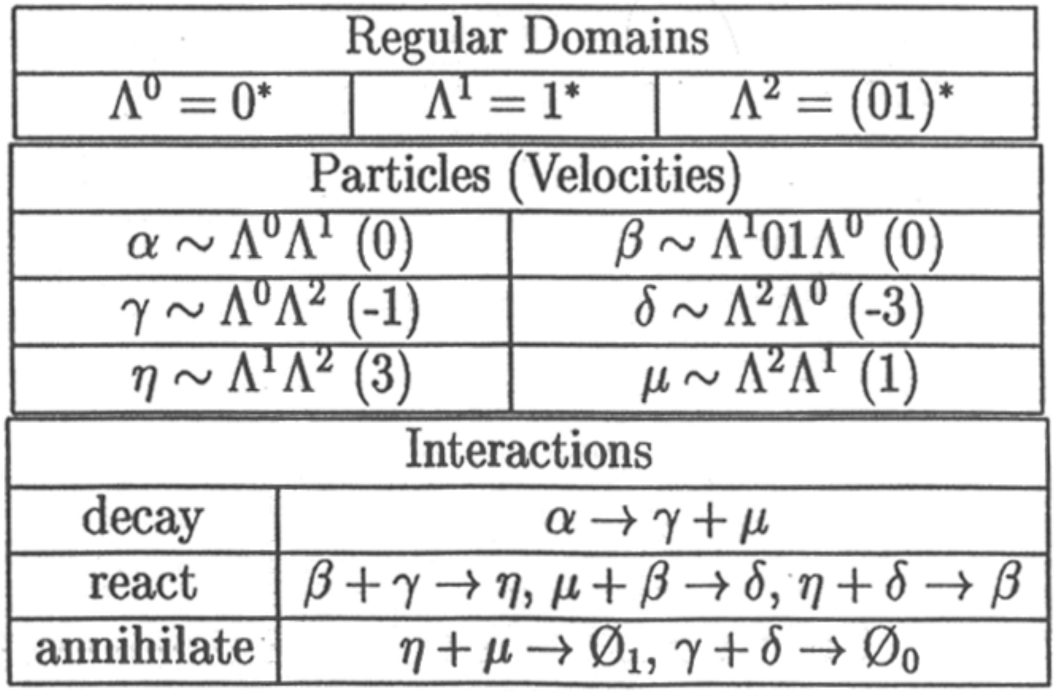




Simple patterns
filtered out



“particles”

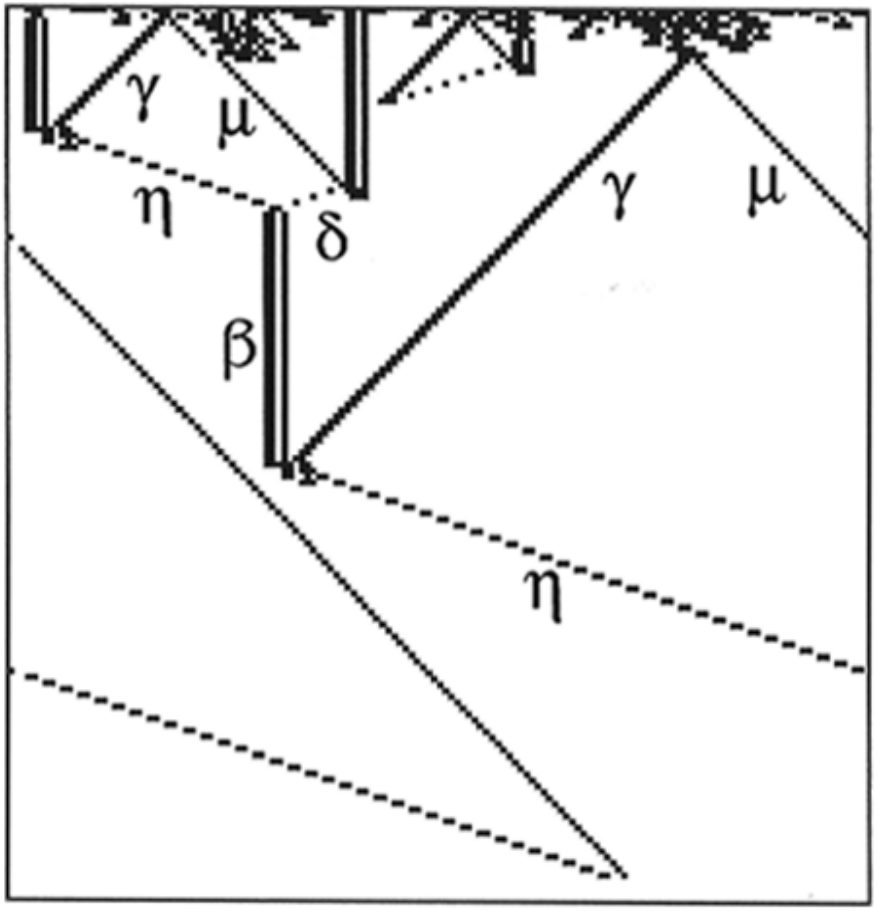


Regular Domains		
$\Lambda^0 = 0^*$	$\Lambda^1 = 1^*$	$\Lambda^2 = (01)^*$

Particles (Velocities)	
$\alpha \sim \Lambda^0\Lambda^1$ (0)	$\beta \sim \Lambda^1 01 \Lambda^0$ (0)
$\gamma \sim \Lambda^0\Lambda^2$ (-1)	$\delta \sim \Lambda^2\Lambda^0$ (-3)
$\eta \sim \Lambda^1\Lambda^2$ (3)	$\mu \sim \Lambda^2\Lambda^1$ (1)

Interactions	
decay	$\alpha \rightarrow \gamma + \mu$
react	$\beta + \gamma \rightarrow \eta$, $\mu + \beta \rightarrow \delta$, $\eta + \delta \rightarrow \beta$
annihilate	$\eta + \mu \rightarrow \emptyset_1$, $\gamma + \delta \rightarrow \emptyset_0$

laws of
“particle physics”

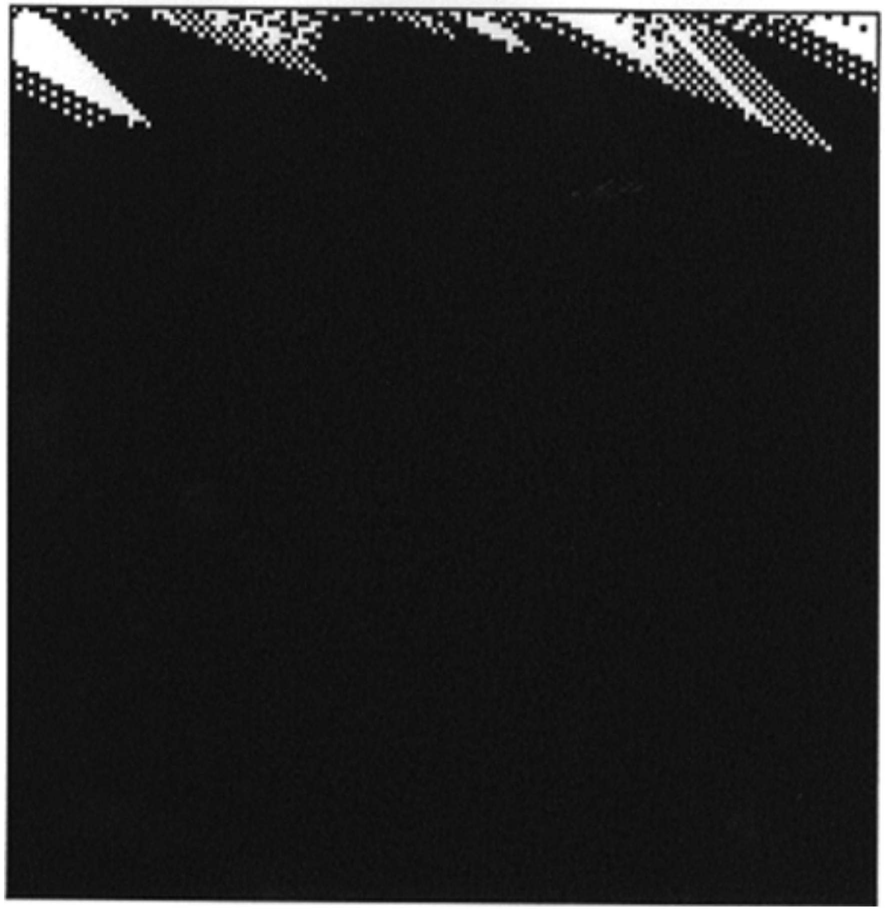


“particles”

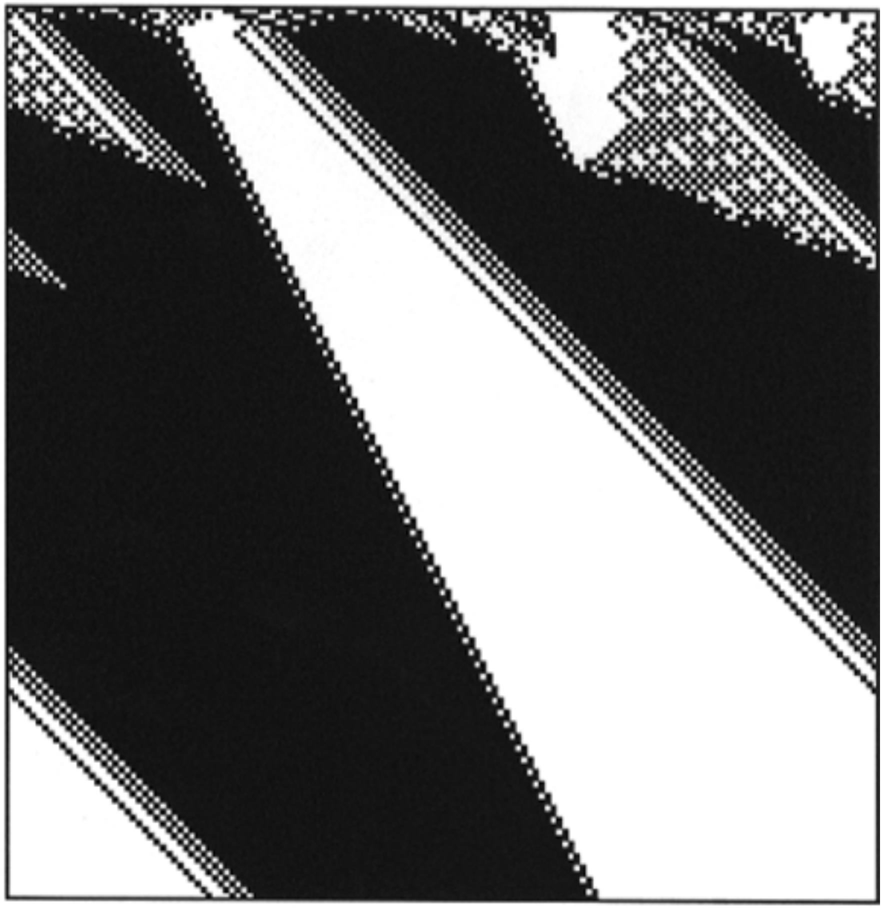
- Level of particles can explain:
  - Why one CA is fitter than another
  - What mistakes are made
  - How the GA produced the observed series of innovations

- Particles give an “information processing” description of the collective behavior

How the genetic algorithm evolved cellular automata

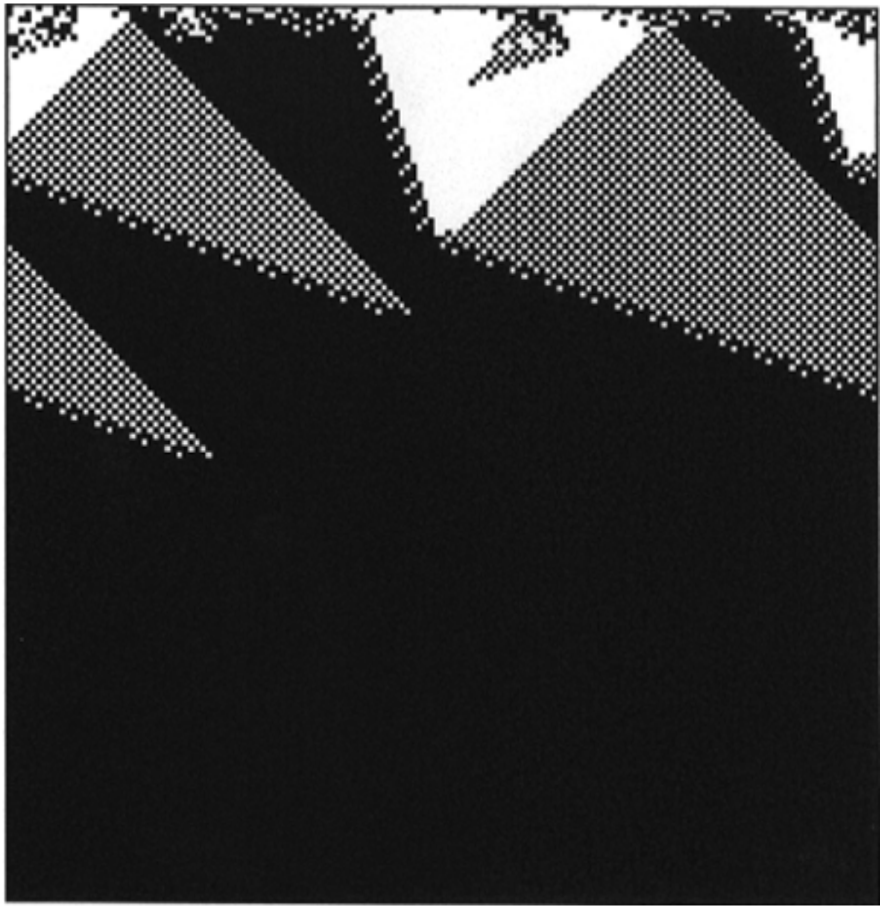


generation 8

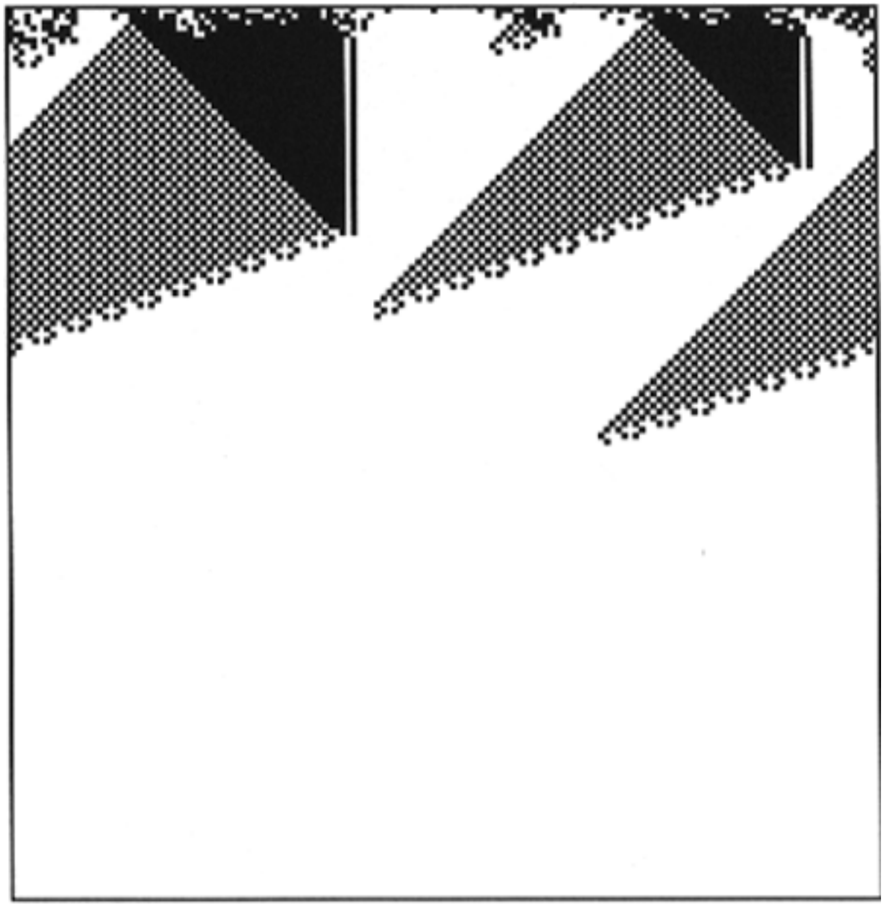


generation 13

How the genetic algorithm evolved cellular automata



generation 17



generation 18