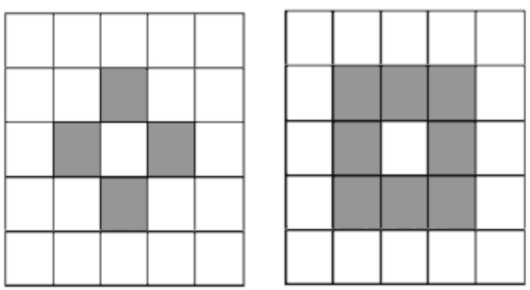
Module 3: Algorithmic self-assembly

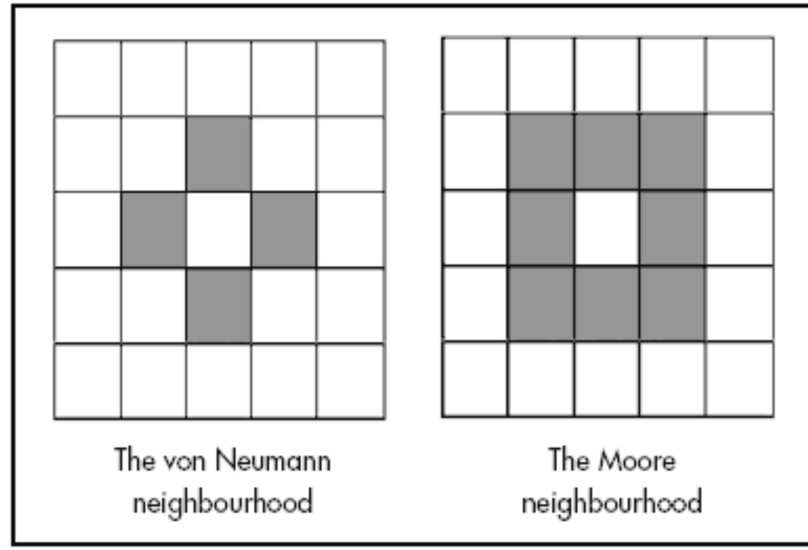
CSE590: Molecular programming and neural computation. Slides in this module are largely due to **Rebecca Schulman** and **Erik Winfree**.

Cellular automata

A lattice-based model of computation, where the lattice can be 1, 2 or any (finite) number of dimensions.



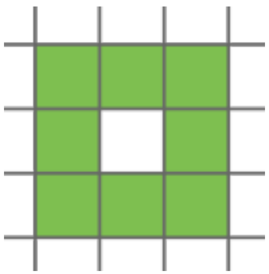
The model consists of a collection of cells, each in one of a finite number of states.



A cell has a neighborhood -- a finite set of cells that are defined to be “adjacent” to it.

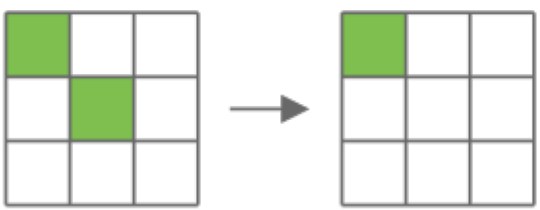
The cells evolve -- at each time step, the cell changes state (or stays the same) based on the states of its neighbors.

Conway's game of life

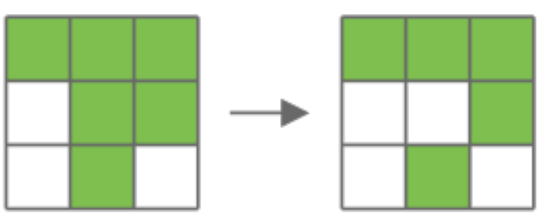


A 2d cellular automaton. Every cell interacts with its 8 neighbors. A cell is either live (colored) or dead (blank).

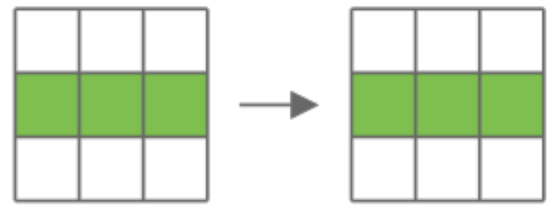
1. A live cell with 0 neighbors or 1 neighbor dies ("underpopulation").

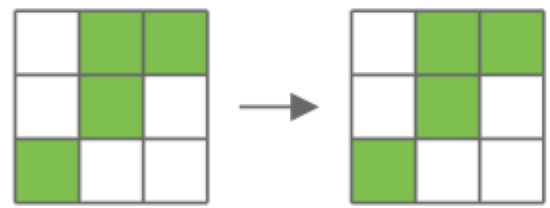


2. A cell with 4,5,6,7 or 8 neighbors dies ("overpopulation").

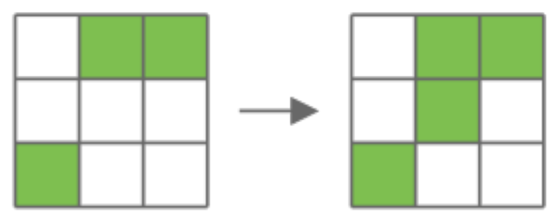


3. A live cell with 2-3 neighbors lives.



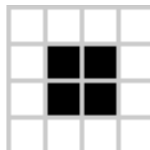


4. A dead cell with exactly 3 neighbors becomes live.

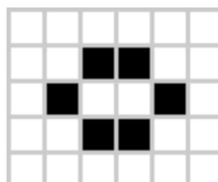


Inputs and outputs of cellular automata are structures

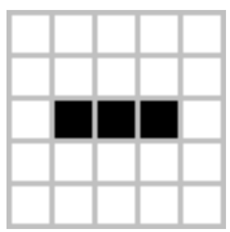
Block



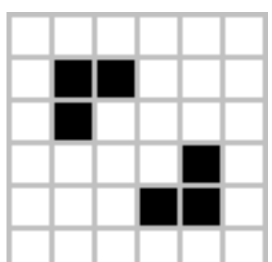
Beehive



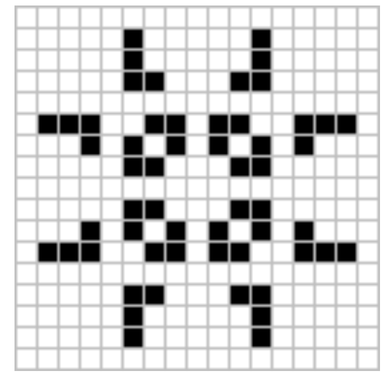
Blinker



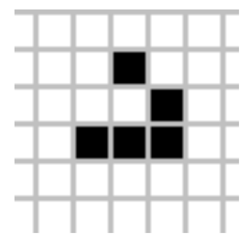
Beacon



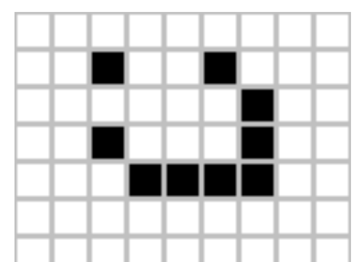
Pulsar



Glider



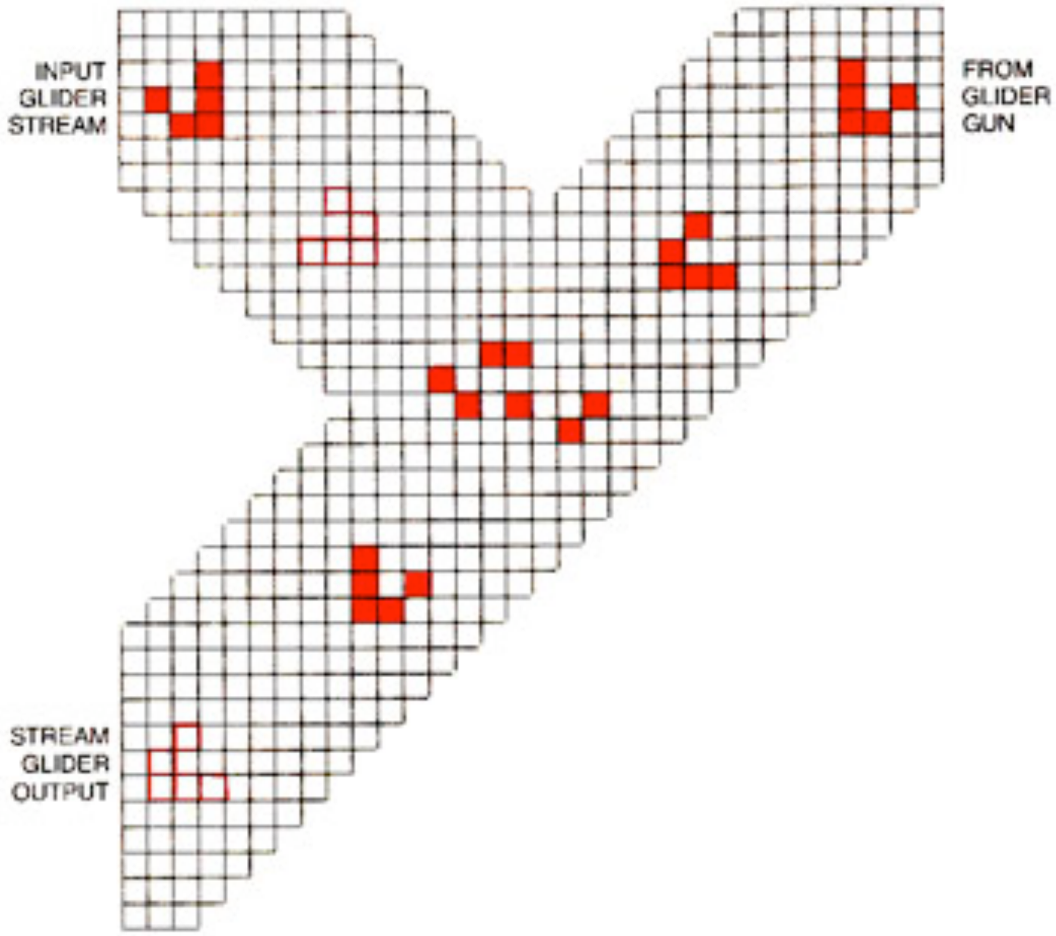
Spaceship



Conway's game of life can produce aperiodic patterns

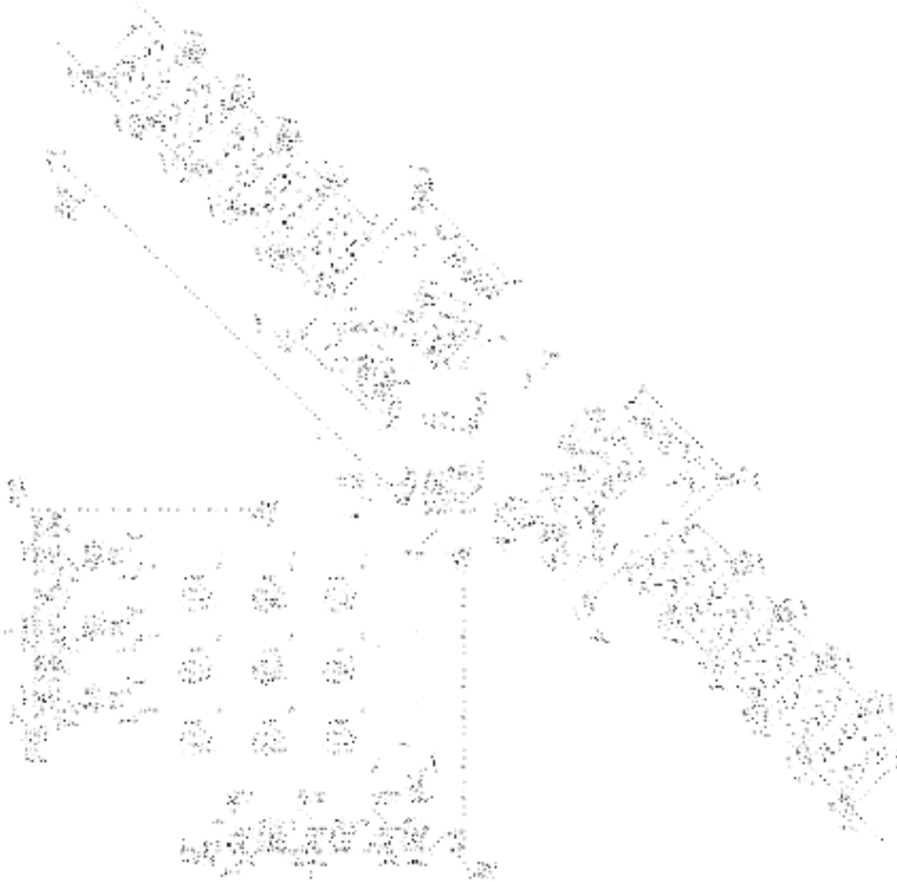
A glider gun

And it can do logic



Glider gun logic gates

Conway's game of life is Turing complete



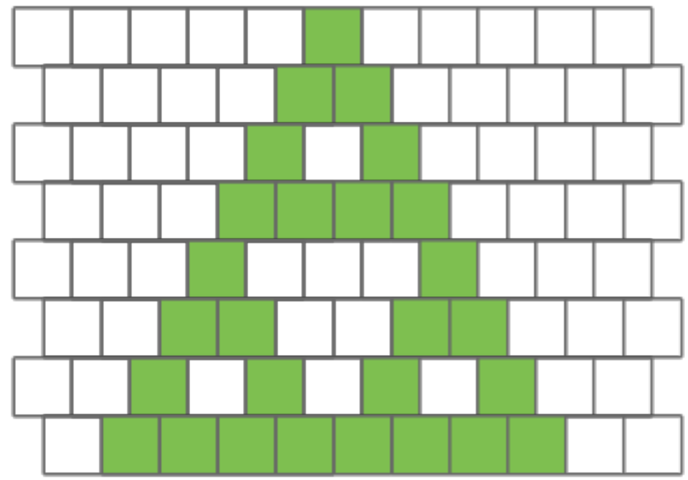
A zoom-out of a
Turing machine in action..

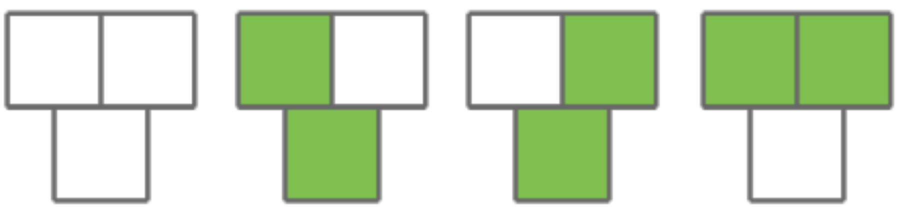
Conway's game of life can compute anything that a computer can compute

Sierpinski's triangle

A 1-dimensional block cellular automaton.

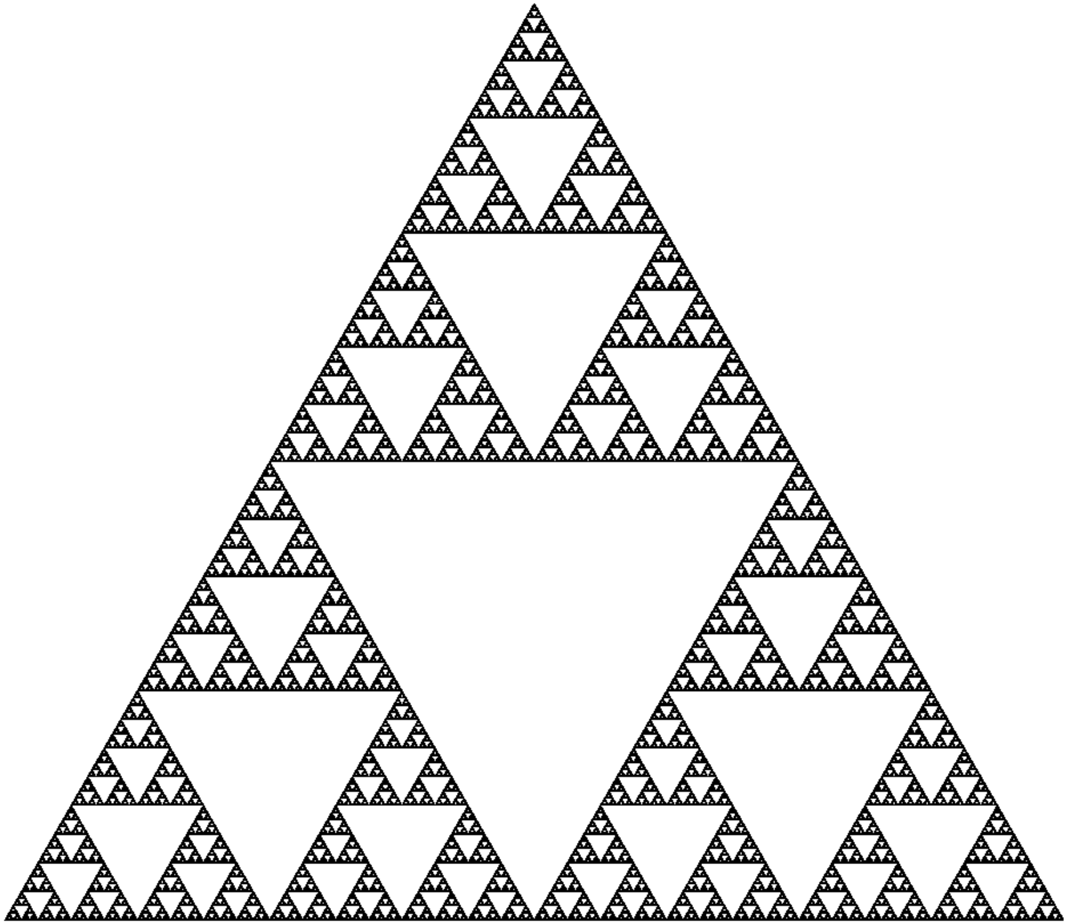
At each time step, the lattice staggers, and neighbors are above and to the left and right of the previous step.





The next state is the **XOR** of the two previous states.

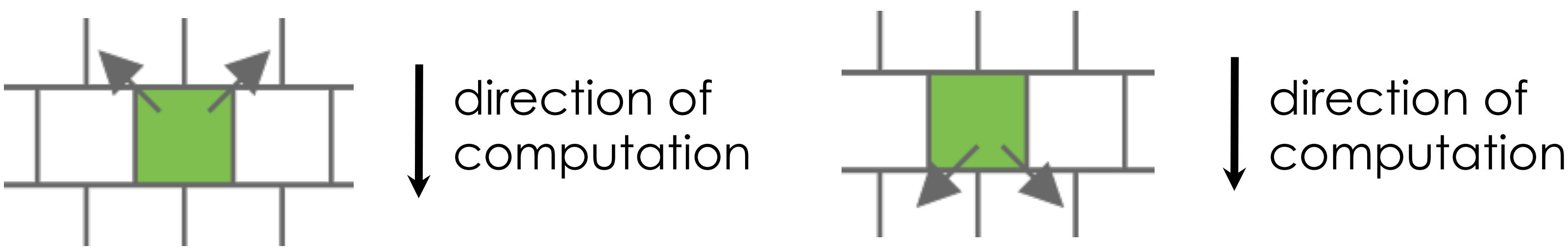
Sierpinski's triangle



Sierpinski's triangle

The next state is the **XOR** of the two previous states.

Each state in the fixed lattice requires knowledge of four surrounding states.

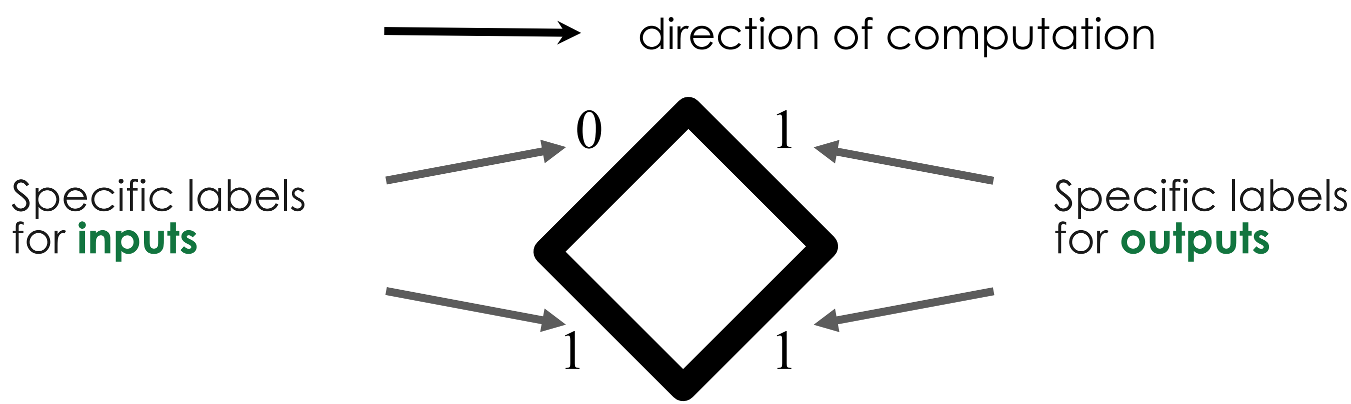


two inputs

two outputs

Yet even with this simple mechanism, block cellular automata are Turing complete.

Block cellular automata with tiles



Next we' ll introduce the **abstract tile assembly model**, where tiles start from a seed, and attach to a growing block.

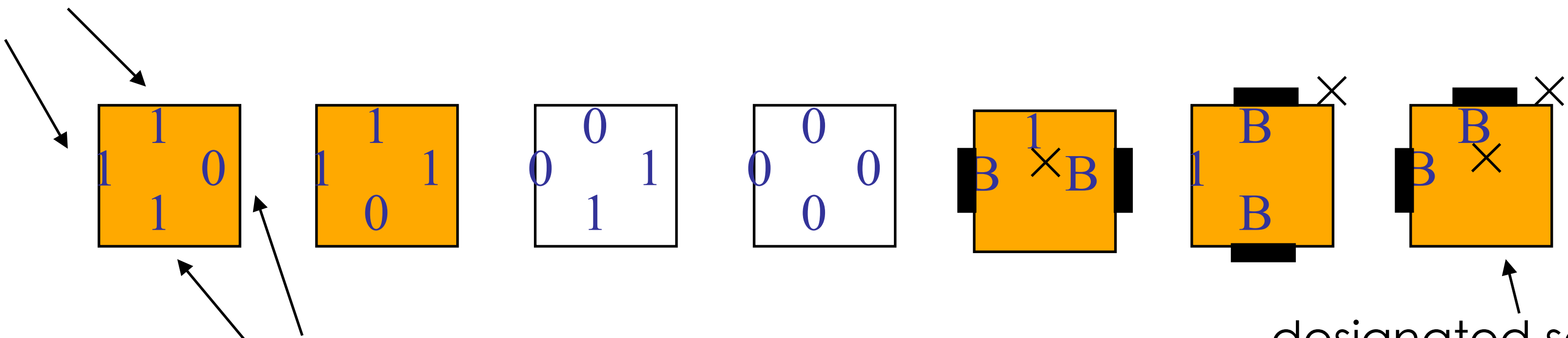
Computation occurs by adding tiles, which form rows of cells, but it is not necessary that rows be added one at a time.

Tiles can be added if their strength of attachment is greater than a threshold. In our case the threshold will be 2.

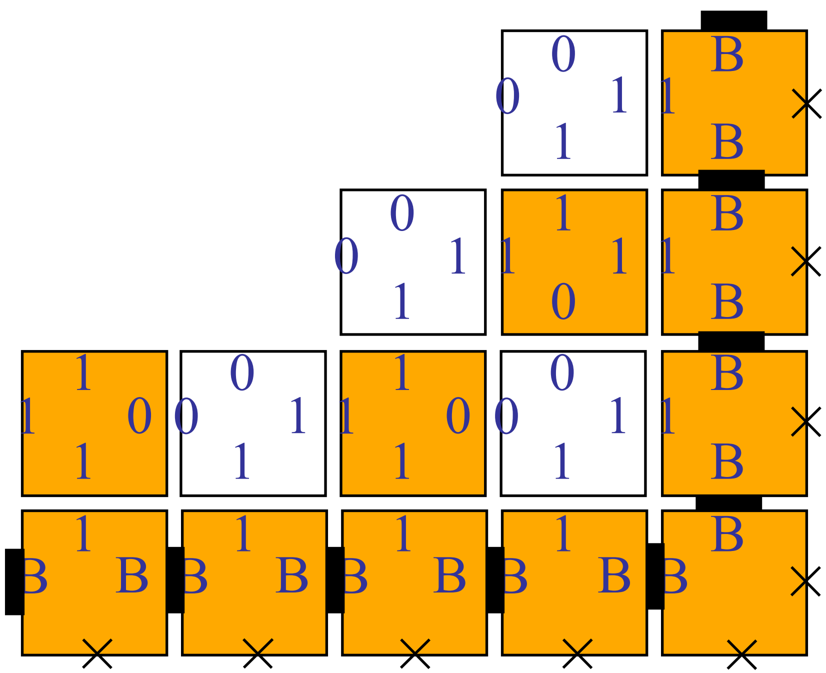
One can show that this processes simulates the execution of a cellular automaton.

Block cellular automata with tiles

formal tiles may not be rotated



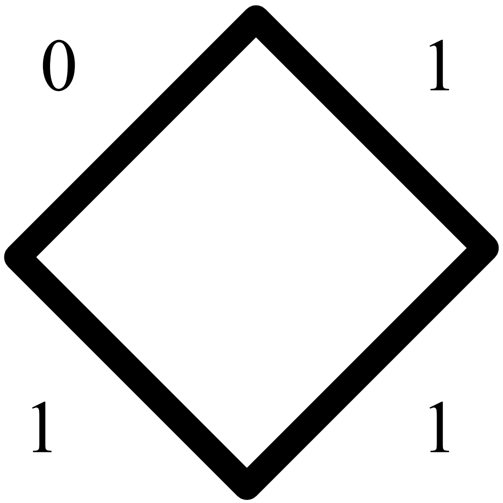
Growth:
1 strong bond
or
2 weak bonds

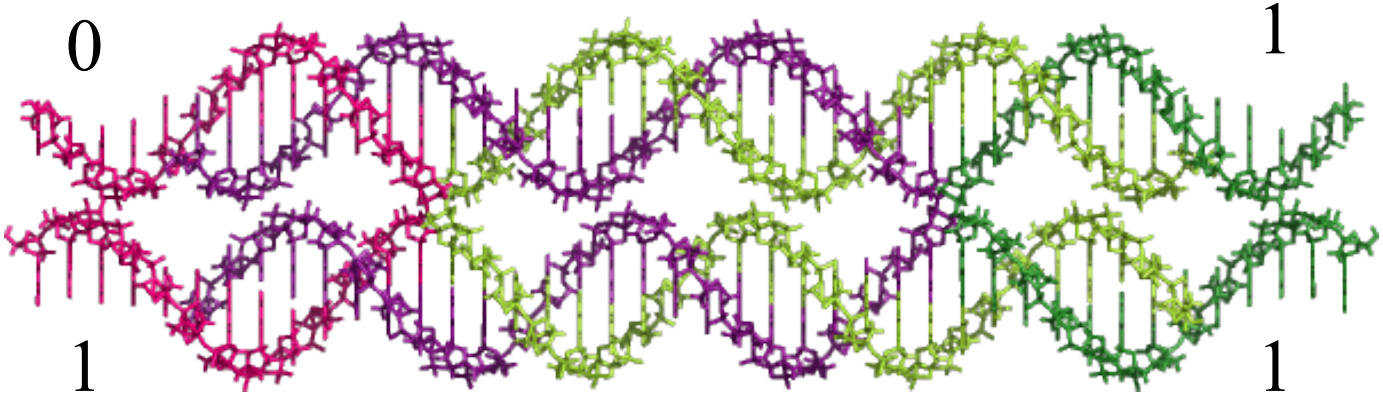


bond type	bond strength
"0"	1 = weak
"1"	1 = weak
"B"	2 = strong
	0 = null

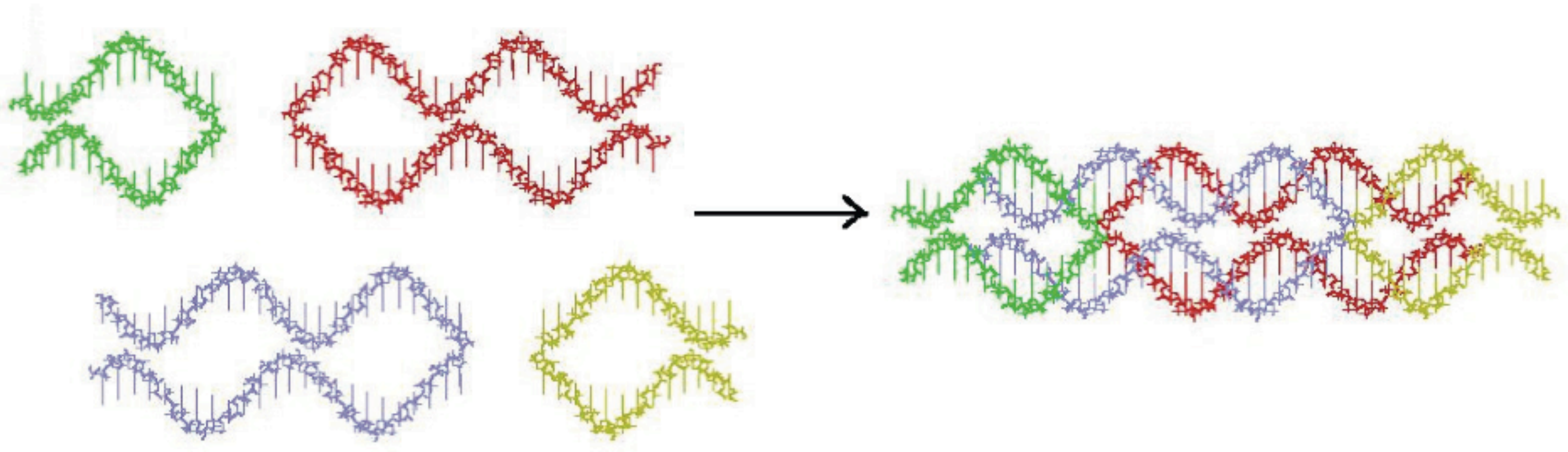
Winfree (1996,1998)

Block cellular automata with tiles



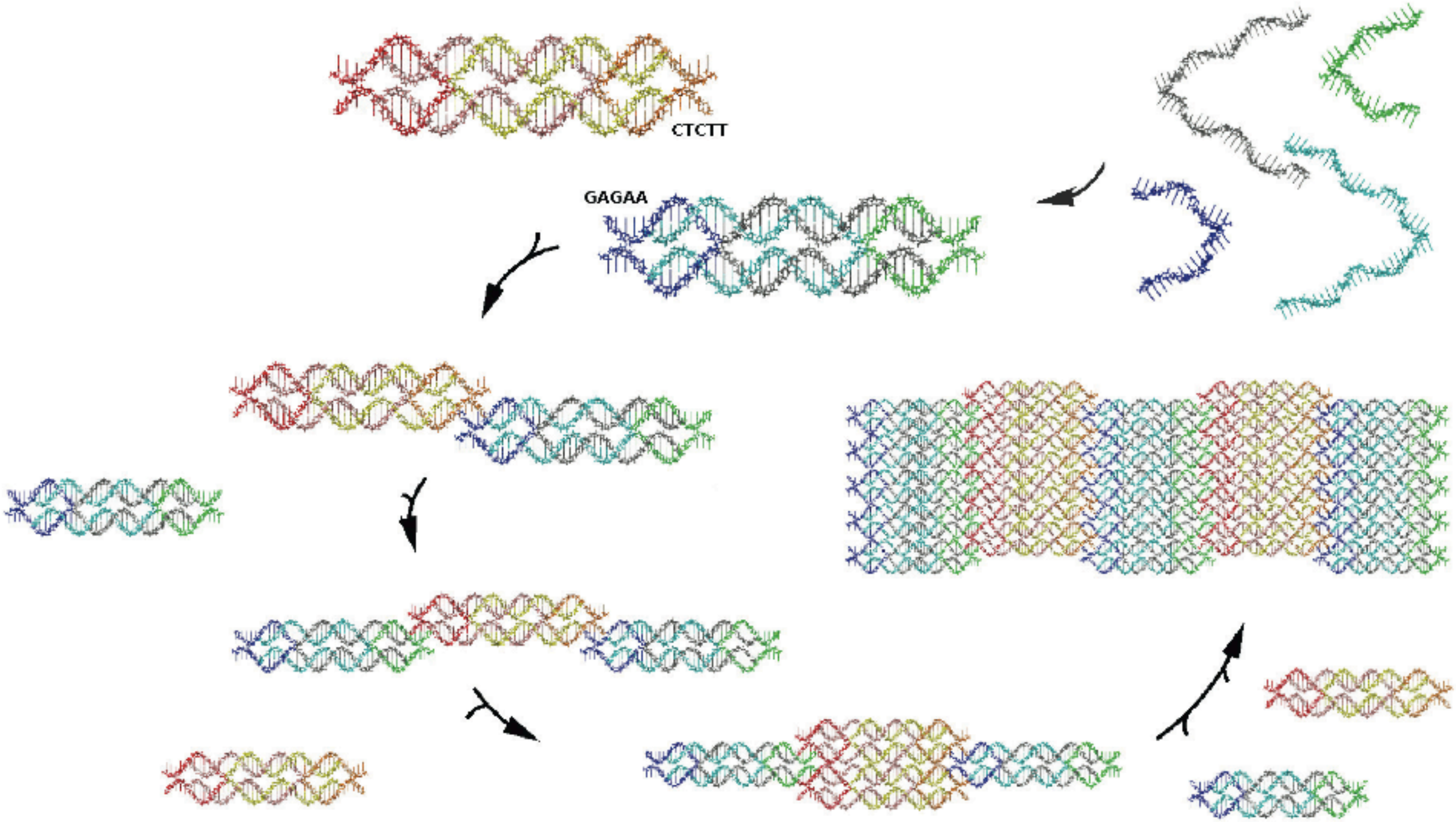


DNA tiles



DNA tiles are formed from four short, synthetic DNA strands

DNA tile assembly



DNA tiles will attach to each other via "sticky" ends that have complementary sequences.

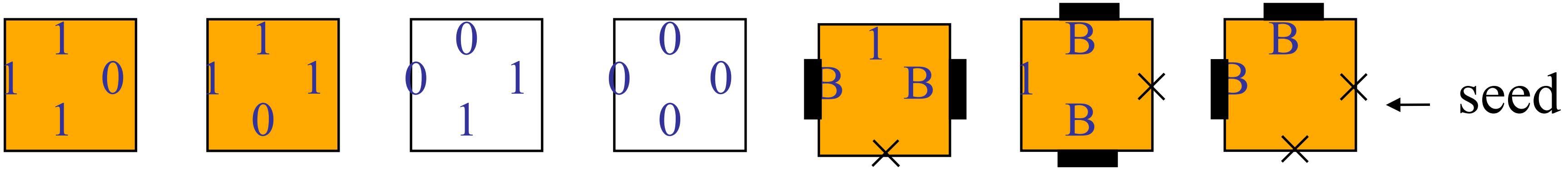
DNA tile assembly

Attachment of a block of the CA lattice <-> attachment of a DNA tile to a crystal of DNA tiles.

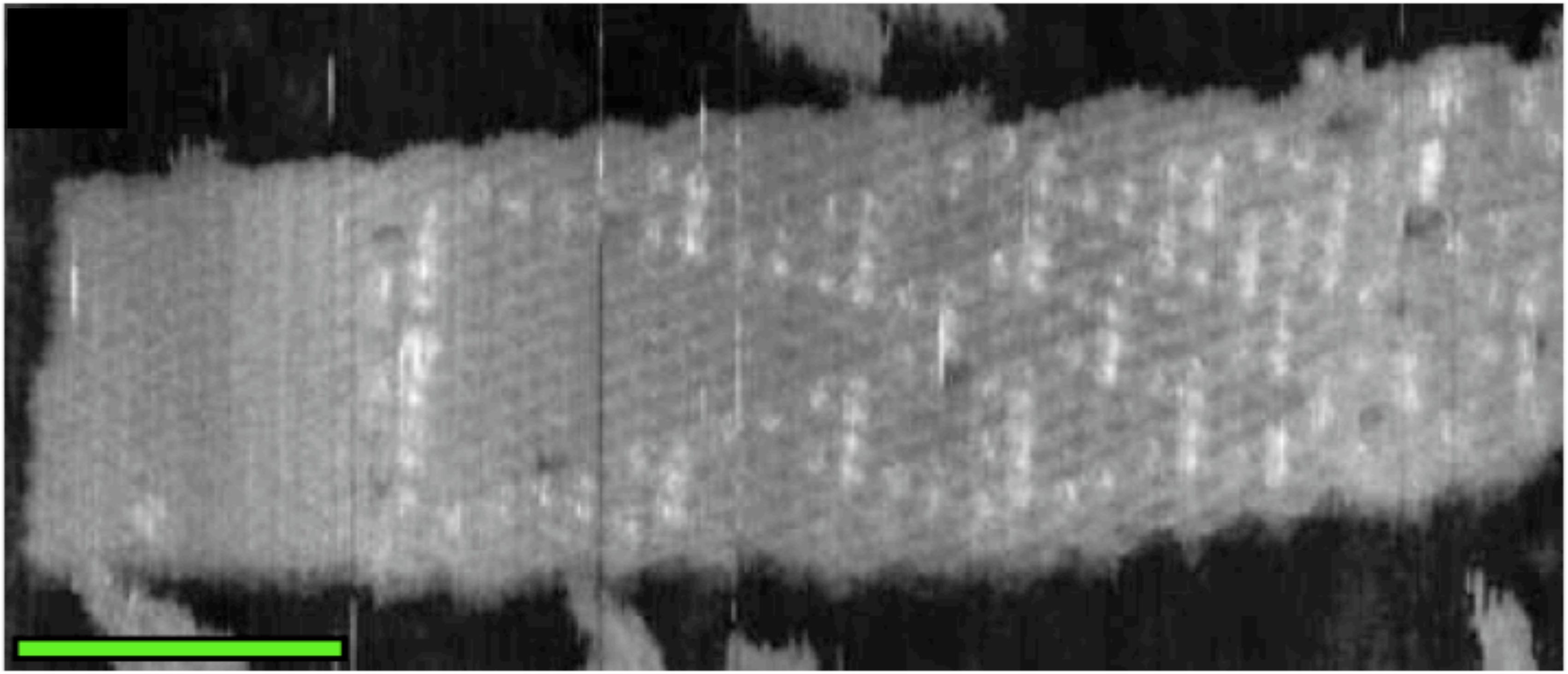
This should only happen if the sticky ends match, and there are enough sticky ends that this is a favorable reaction.

The result: we can program a set of “tiles”, make them out of DNA, then make the assembly we predict into a real object!

Simulated assembly of a DNA Sierpinski triangle



A self-assembled DNA object

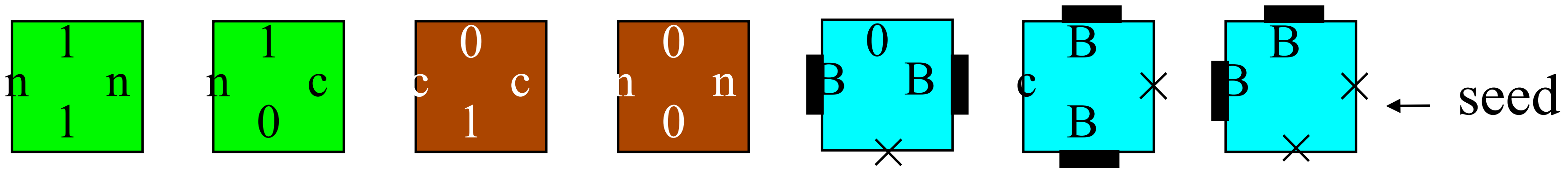


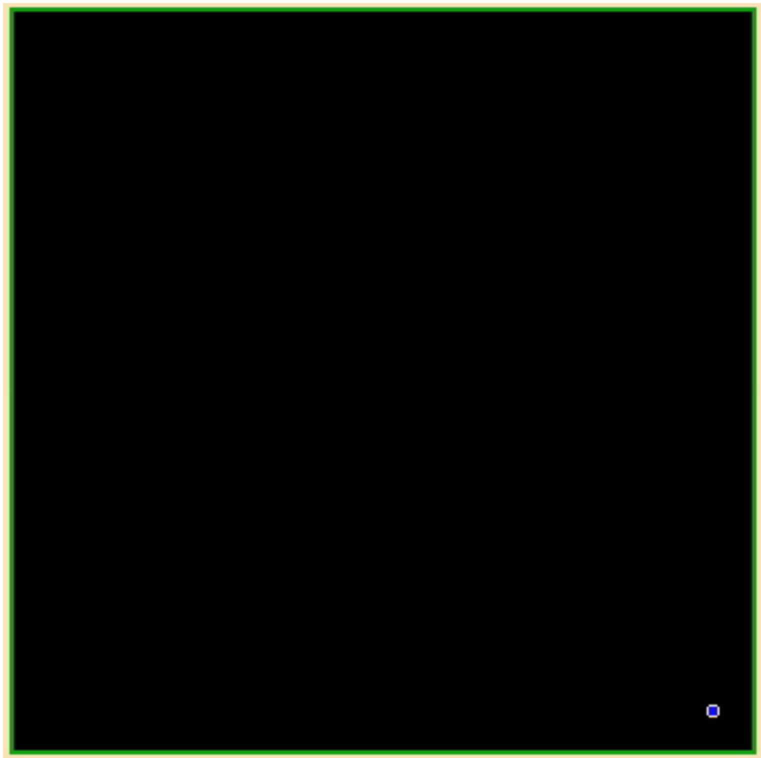
100 nm (atomic force microscope image)

- Kenichi Fujibayashi, Rizal Hariadi, Sung Ha Park, Erik Winfree, Satoshi Murata (Nano Letters, 2008)

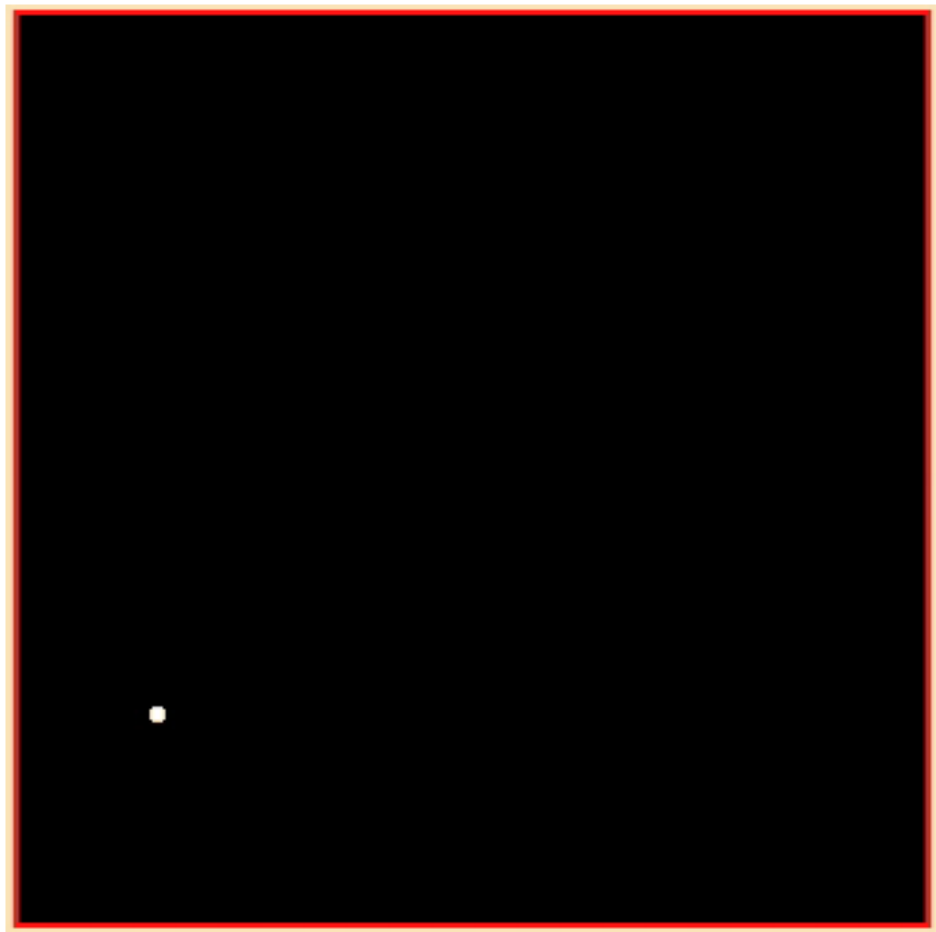
Programming self-assembly

Tile set 2: Binary counter tile set



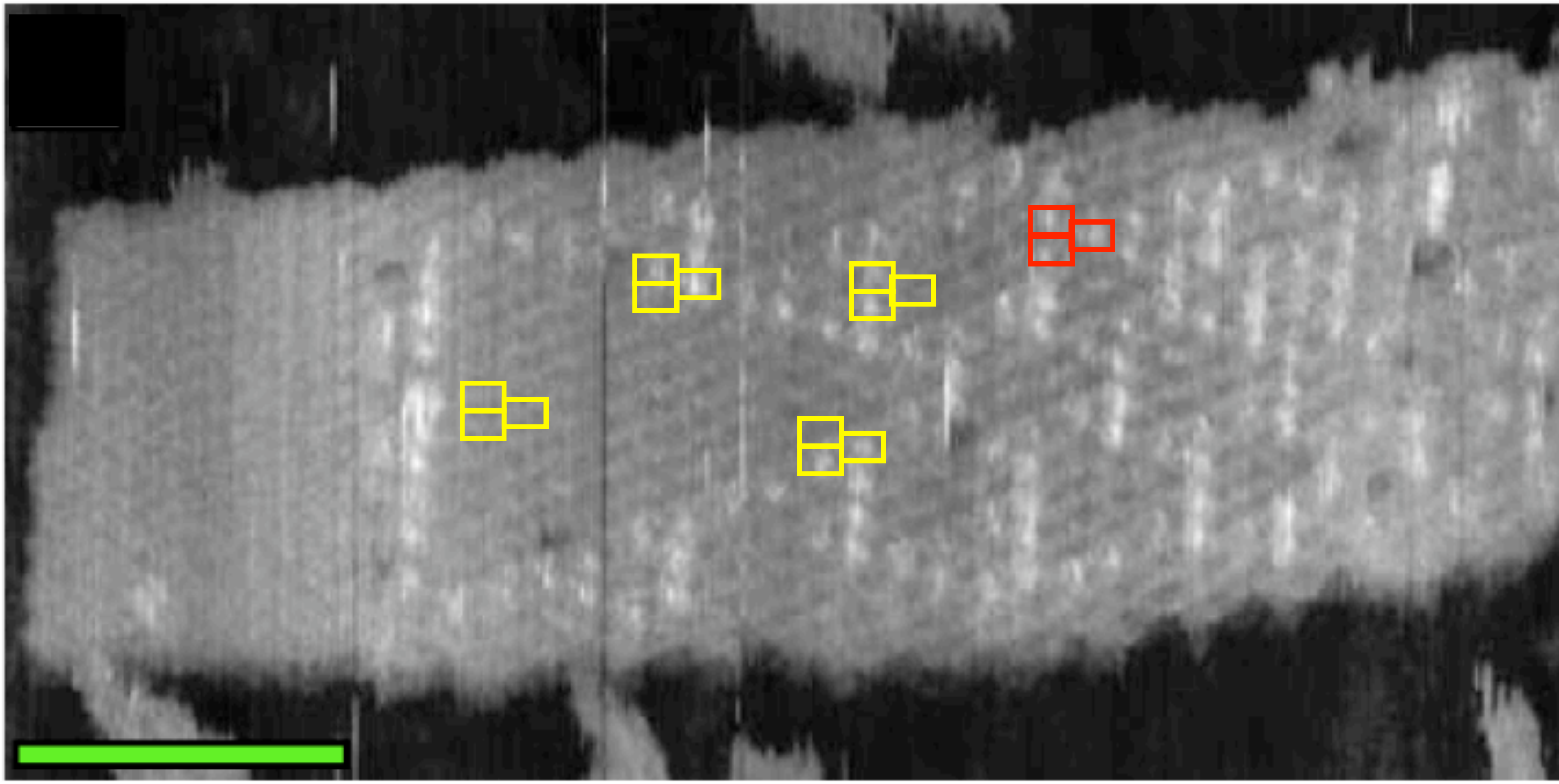


Programming self-assembly



Tile set 3: 59 tile types, 28 bond types

Unfortunately, DNA sometimes makes mistakes



100 nm

An even number of white dots in each triangle!

- Kenichi Fujibayashi, Rizal Hariadi, Sung Ha Park, Erik Winfree, Satoshi Murata (Nano Letters,2008)

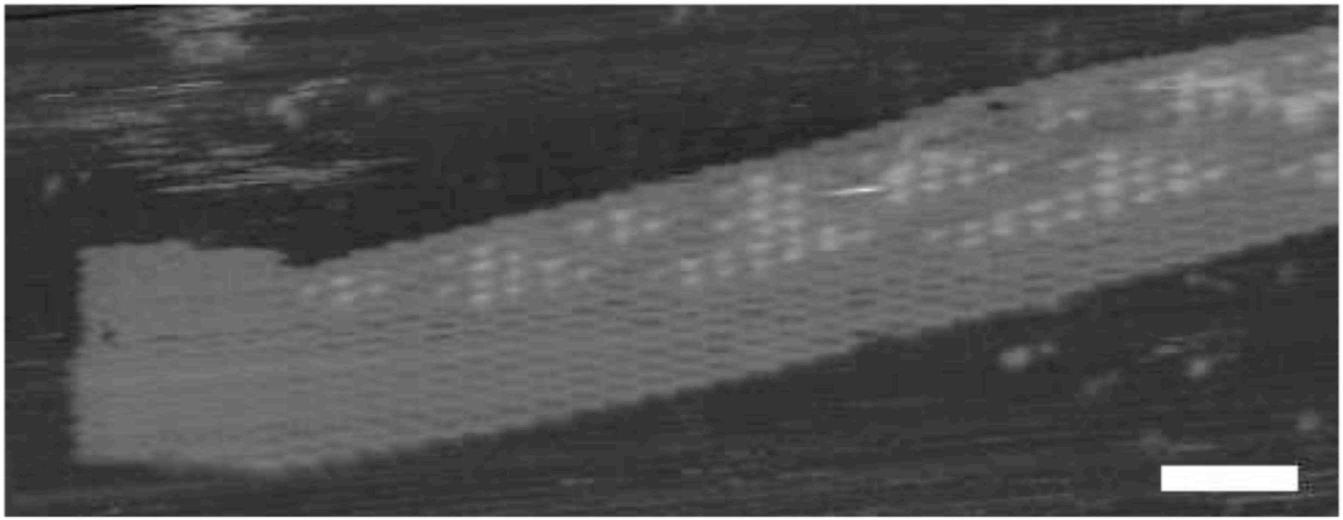
Unfortunately, DNA sometimes makes mistakes

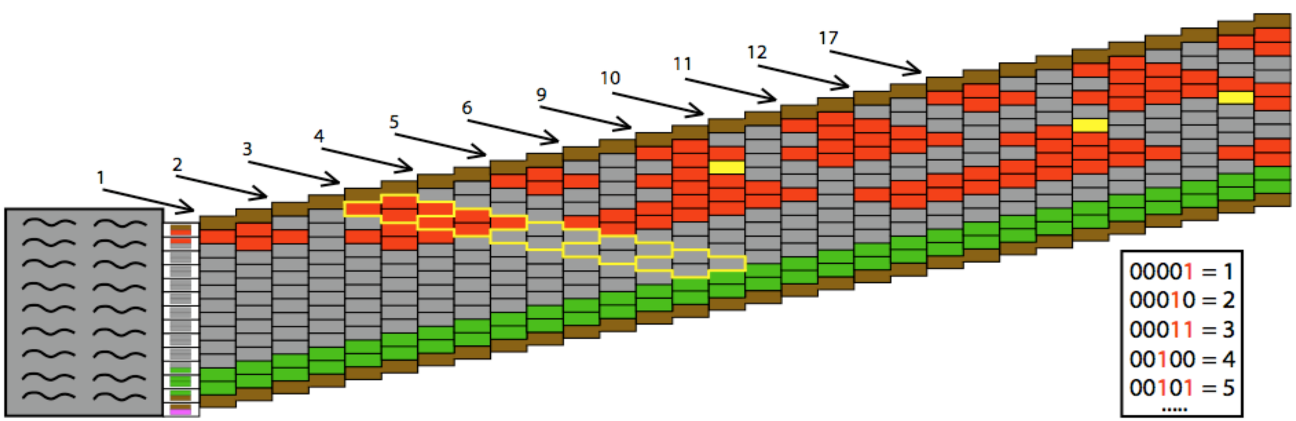
A big challenge in DNA self-assembly is to get DNA to follow instructions:

Error rate in your computer's logic system 1 in 10^{23}

Error rate in DNA tile assembly 1 in 10^{2}!

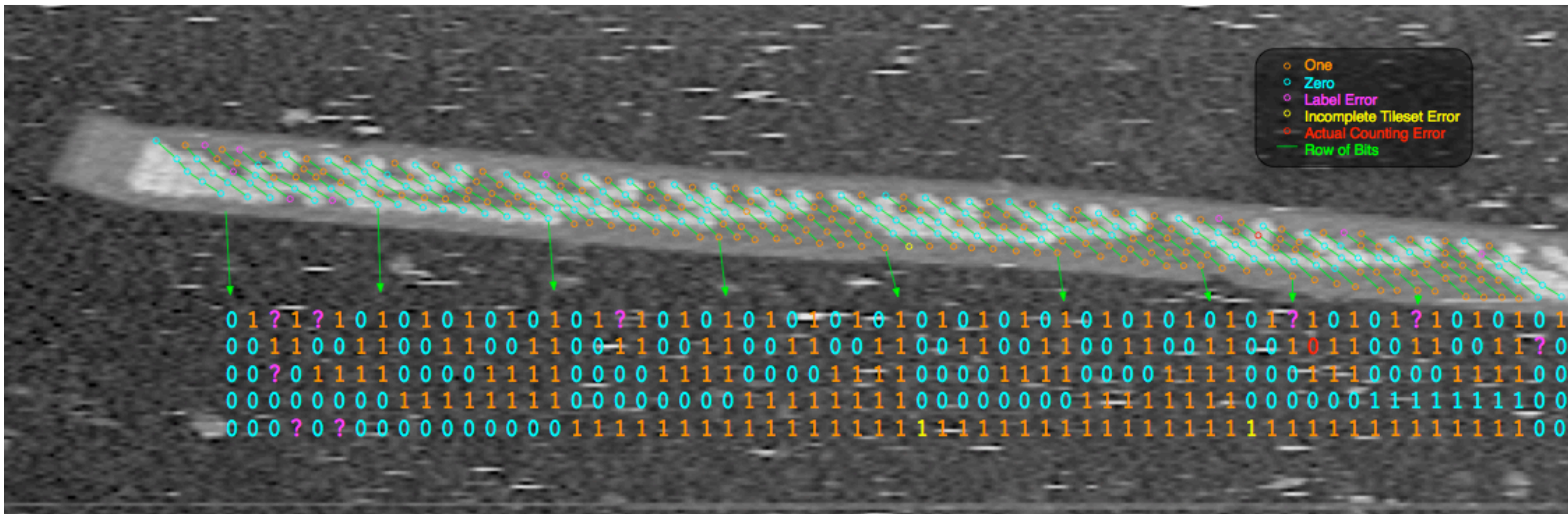
A single error in a crystal can be disastrous





- Robert Barish, Rebecca Schulman, Paul Rothemund, Erik Winfree (PNAS, 2009)

5-bit binary counter



22 tile types, error rate 0.05%

- Optimized growth conditions
- Optimized concentrations
- Optimized labels

Constantin Evans and Erik Winfree, in preparation.

Current research focuses on how to make assembly accurate

1. Designing "robust" tile sets can make it harder for an error to stick.

2. Optimizing physical conditions can improve error rates because crystallization happens with fewer defects under some physical conditions than others.

3. Combining both these approaches has allowed us to reach an error rate of less than 1 in 10^4. How much lower can we go?

?