

Crystals that think about how they're growing



David Doty

joint work with Damien Woods, Erik Winfree, Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin

UW Molecular Programming Seminar Series, April 22, 2021



Caltech



Inria Paris



UC Davis



Harvard

Acknowledgements



Caltech



Inria Paris



UC Davis



Harvard

Damien Woods
(co-first author)



Erik Winfree

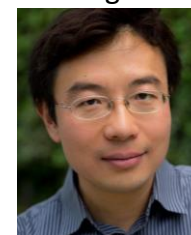


co-authors

Cameron Myhrvold



Peng Yin



Felix Zhou



Joy Hui



lab/science help

Sungwook Woo

Constantine Evans

Sarina Mohanty

Niranjan Srinivas

Deborah Fygenson

Yannick Rondolez

Mingjie Dai

Nikhil Gopalkrishnan

Chris Thachuk

Nadine Dabby

Jongmin Kim

Paul Rothmund

Bryan Wei

Cody Geary

Ashwin Gopinath



Diverse and robust molecular algorithms using reprogrammable DNA self-assembly.
Damien Woods[†], David Doty[†], Cameron Myhrvold, Joy Hui, Felix Zhou, Peng Yin, Erik Winfree.
Nature 2019. [†]*These authors contributed equally.*

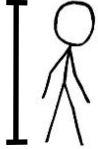


Ljubljana Marshes Wheel. 5k years old

Building things



Newgrange, Ireland. 5.2k years old

Building things by hand: use tools! Great for scale of $10^{\pm 2} \times$ 

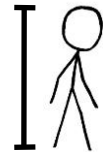


Ljubljana Marshes Wheel. 5k years old

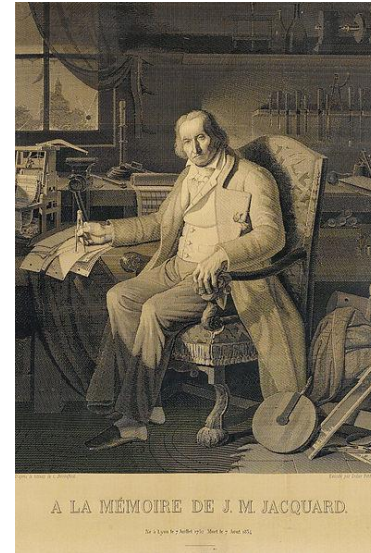
Building things



Newgrange, Ireland. 5.2k years old

Building things by hand: use tools! Great for scale of $10^{\pm 2} \times$ 

Building tools that build things: specify target object with a computer program



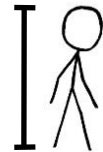


Ljubljana Marshes Wheel. 5k years old

Building things



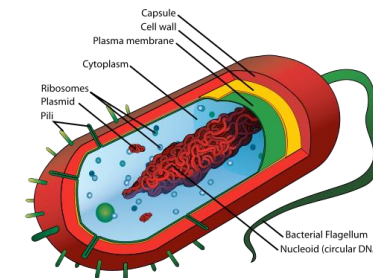
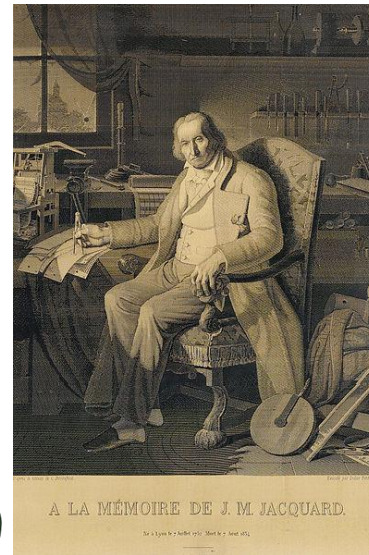
Newgrange, Ireland. 5.2k years old

Building things by hand: use tools! Great for scale of $10^{\pm 2} \times$ 

Building tools that build things: specify target object with a computer program



Programming things to build themselves: for building in small wet places where our hands or tools can't reach



Mariana Ruiz Villarreal

Things that build themselves

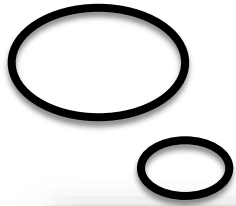


I want to stick below
blue & yellow and
above blue & green



Our topic: self-assembling molecules that compute as they build themselves

Things that build themselves



Our topic: self-assembling molecules that compute as they build themselves

Things that build themselves



Our topic: self-assembling molecules that compute as they build themselves

Hierarchy of abstractions

➡ Bits:	Boolean circuits compute
Tiles:	Tile growth implements circuits
DNA:	DNA strands implement tiles

Harmonious arrangement

0

1

1

0

1

1

Harmonious arrangement



0

1

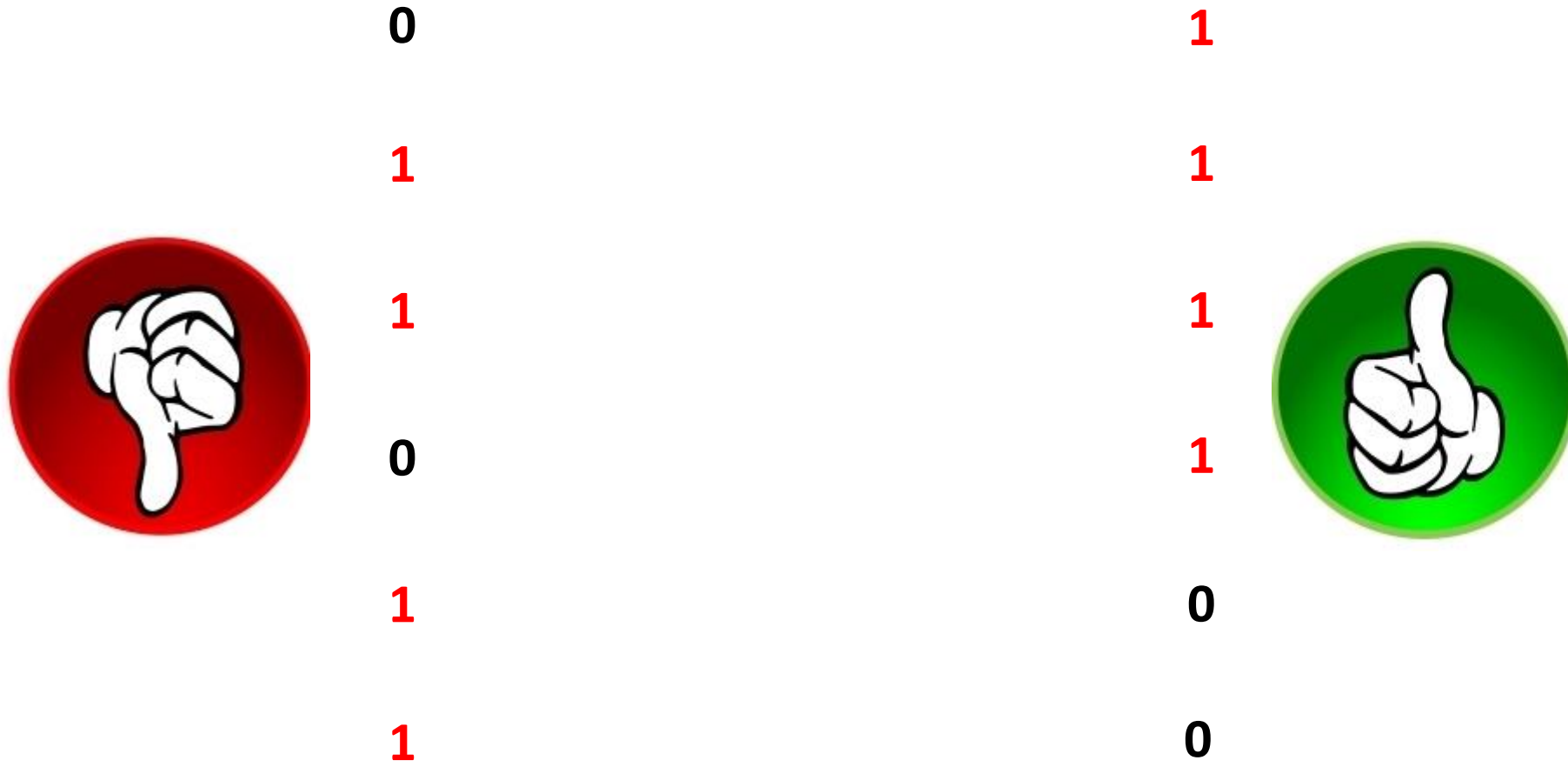
1

0

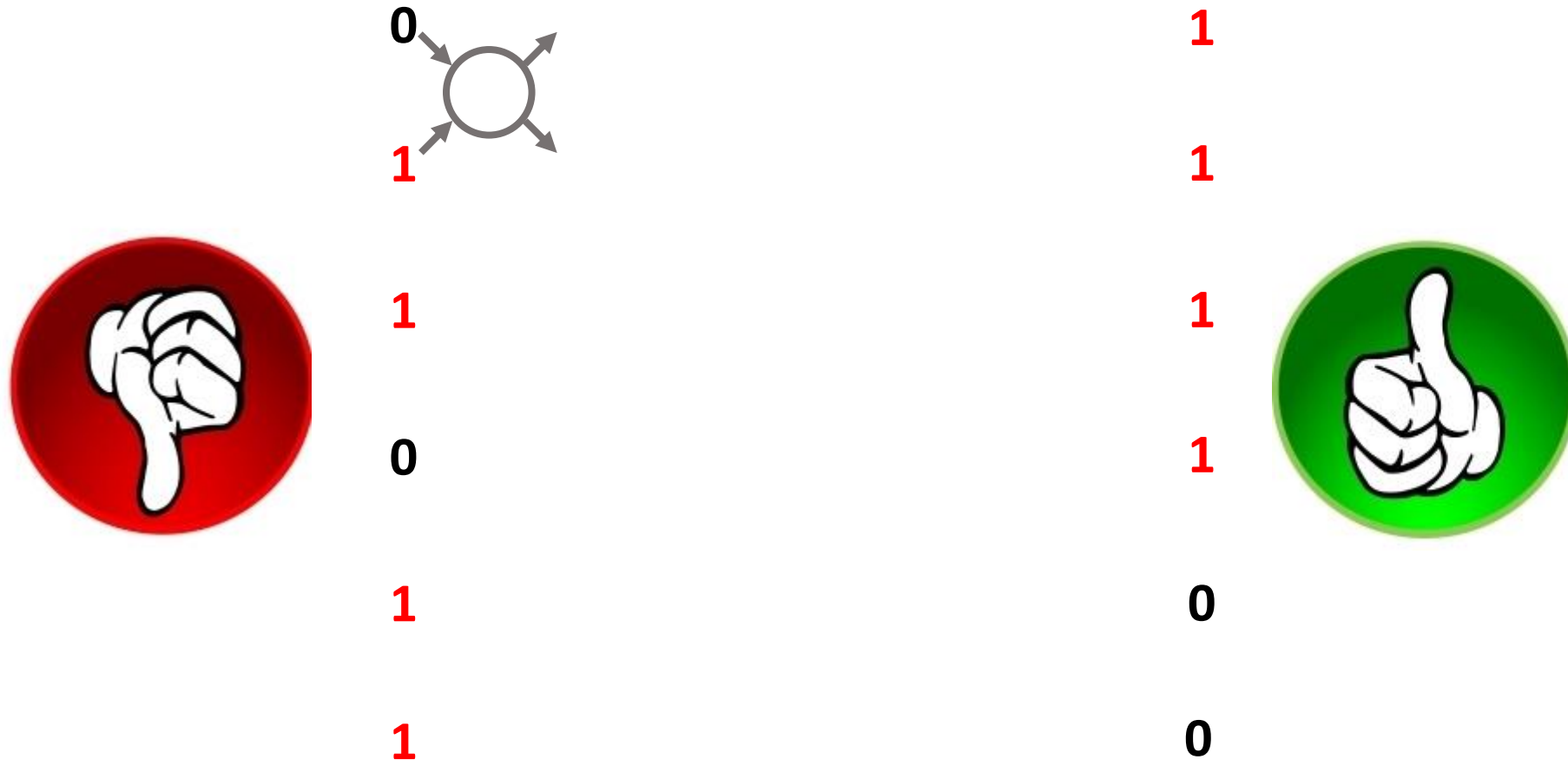
1

1

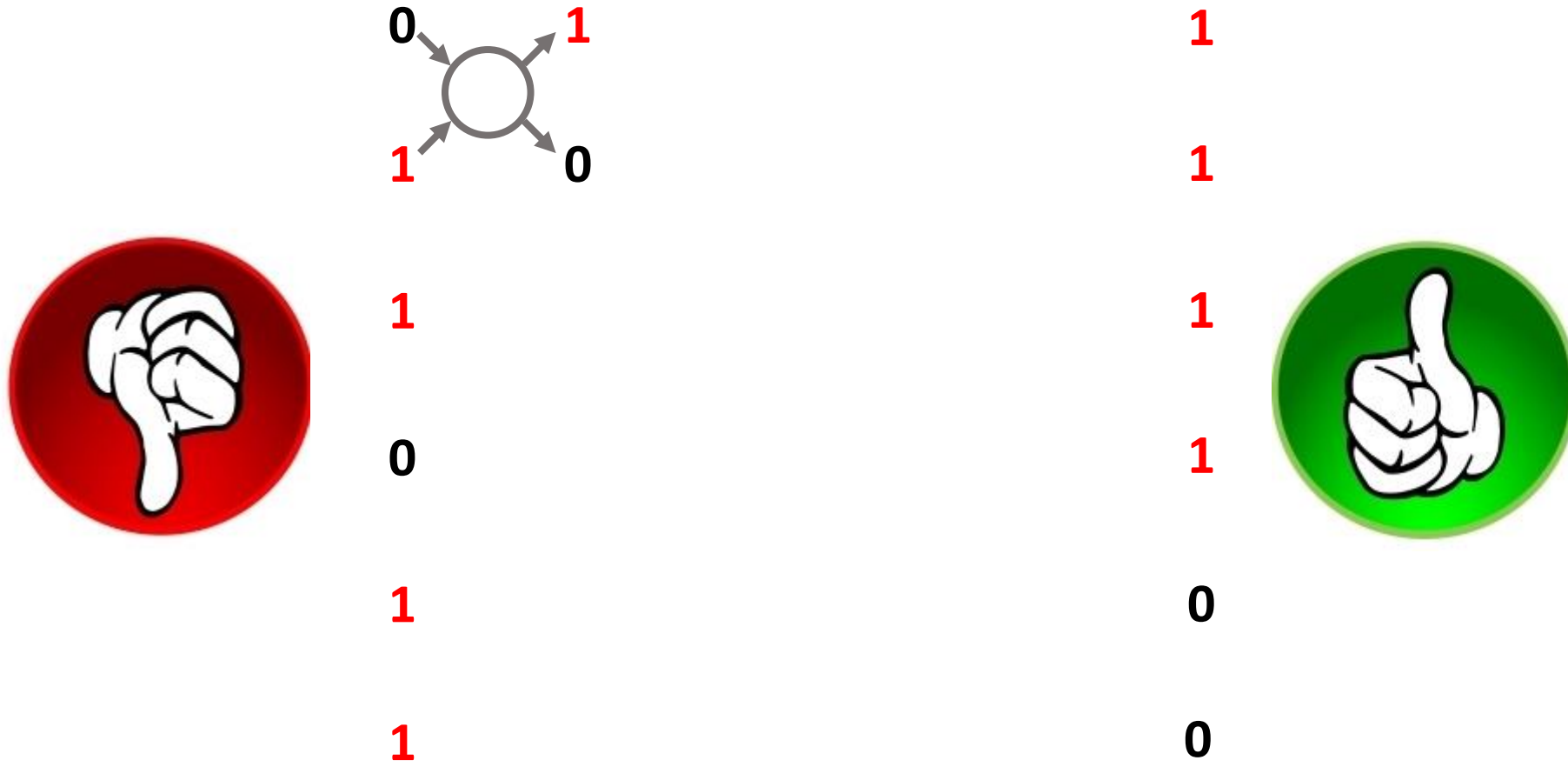
Harmonious arrangement



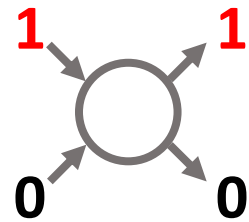
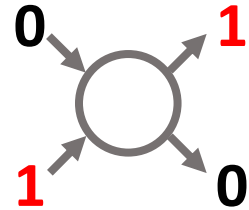
Harmonious arrangement



Harmonious arrangement



Harmonious arrangement



1

1

1

1

1

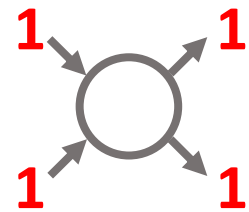
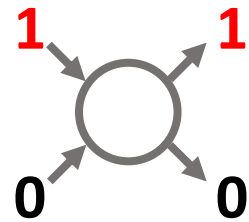
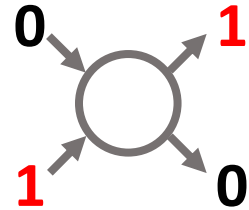
1

0

0



Harmonious arrangement



1

1

1

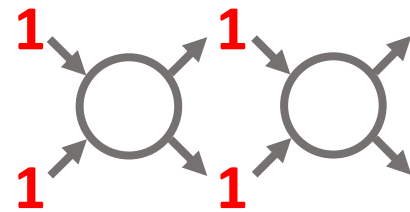
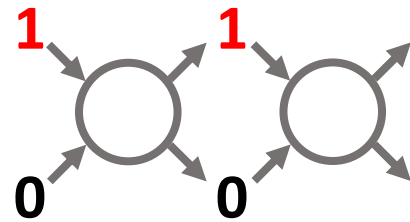
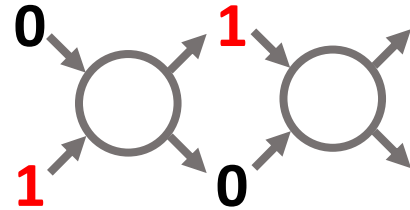
1

0

0



Harmonious arrangement



1

1

1

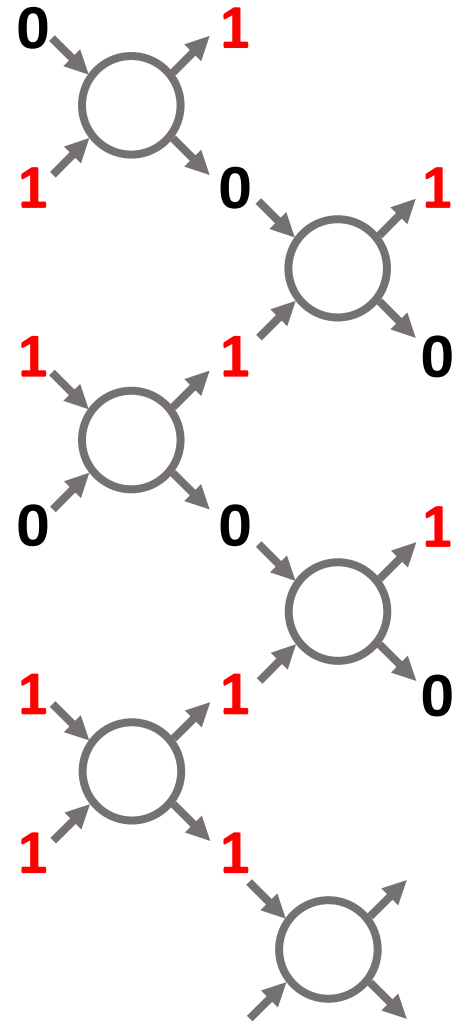
1

0

0



Harmonious arrangement



1

1

1

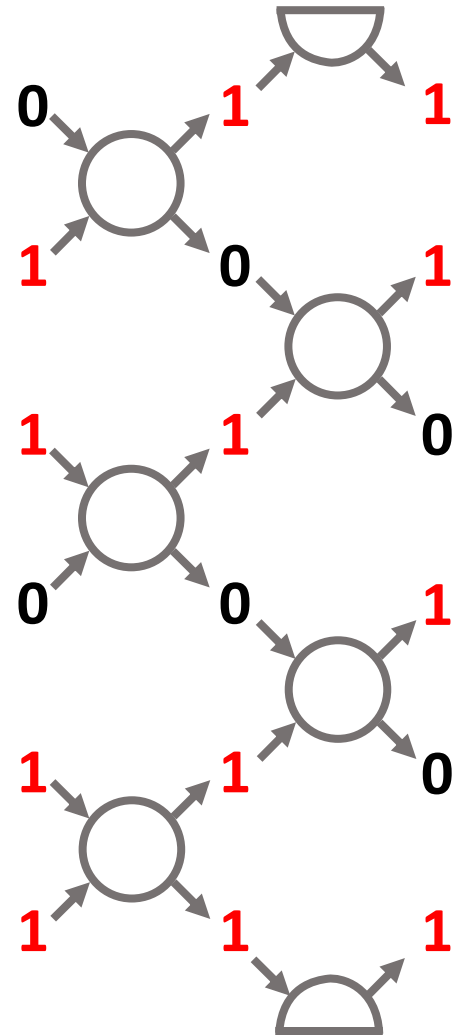
1

0

0



Harmonious arrangement



1

1

1

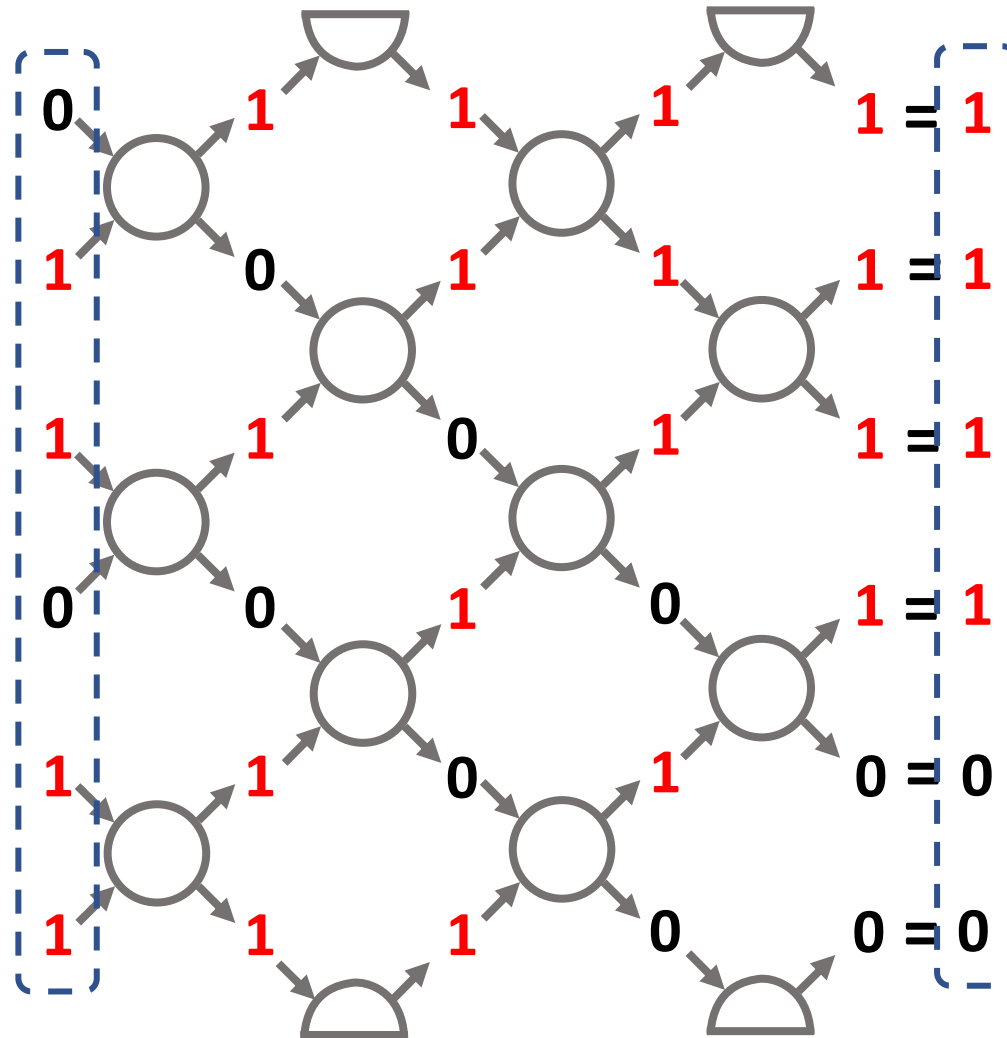
1

0

0



Harmonious arrangement



a.k.a. sorting



Odd bits

1

0

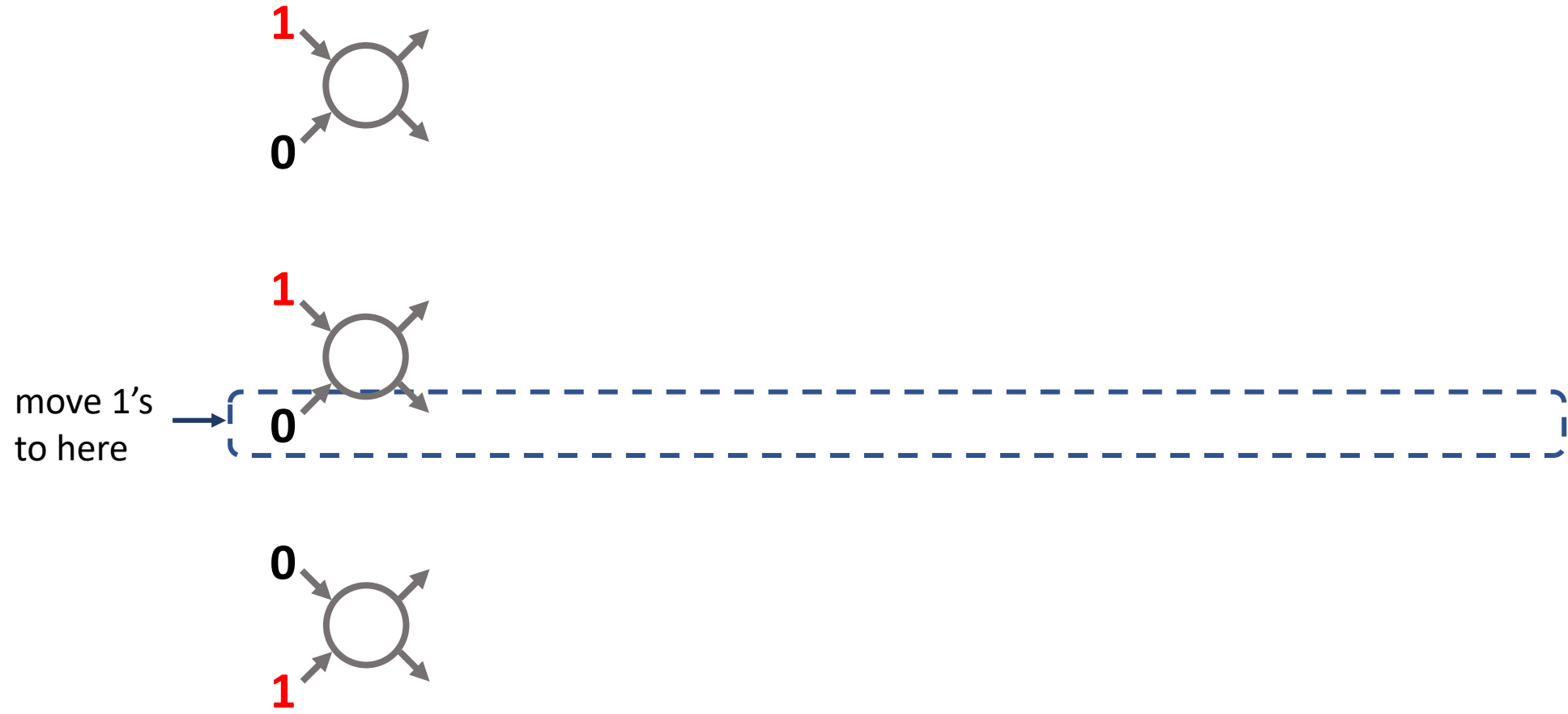
1

0

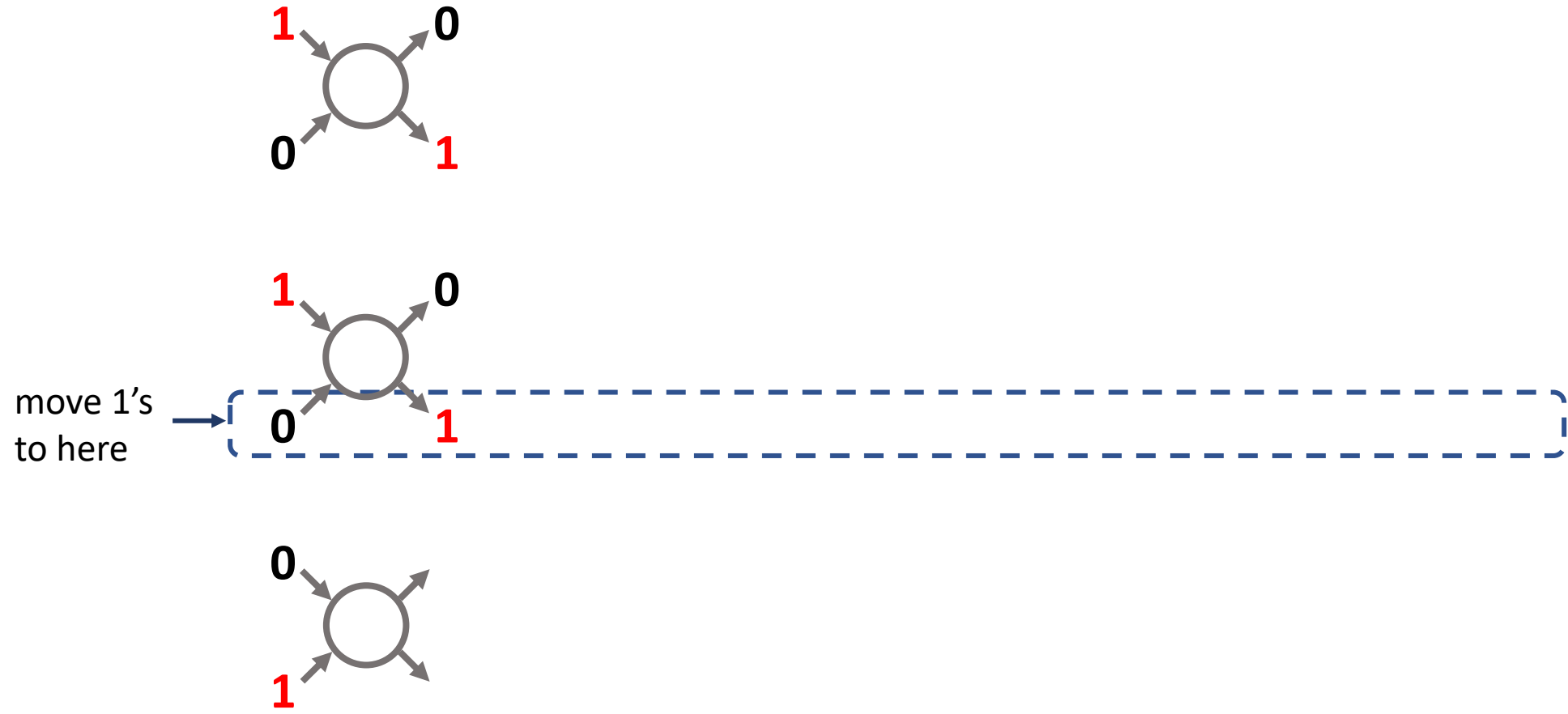
0

1

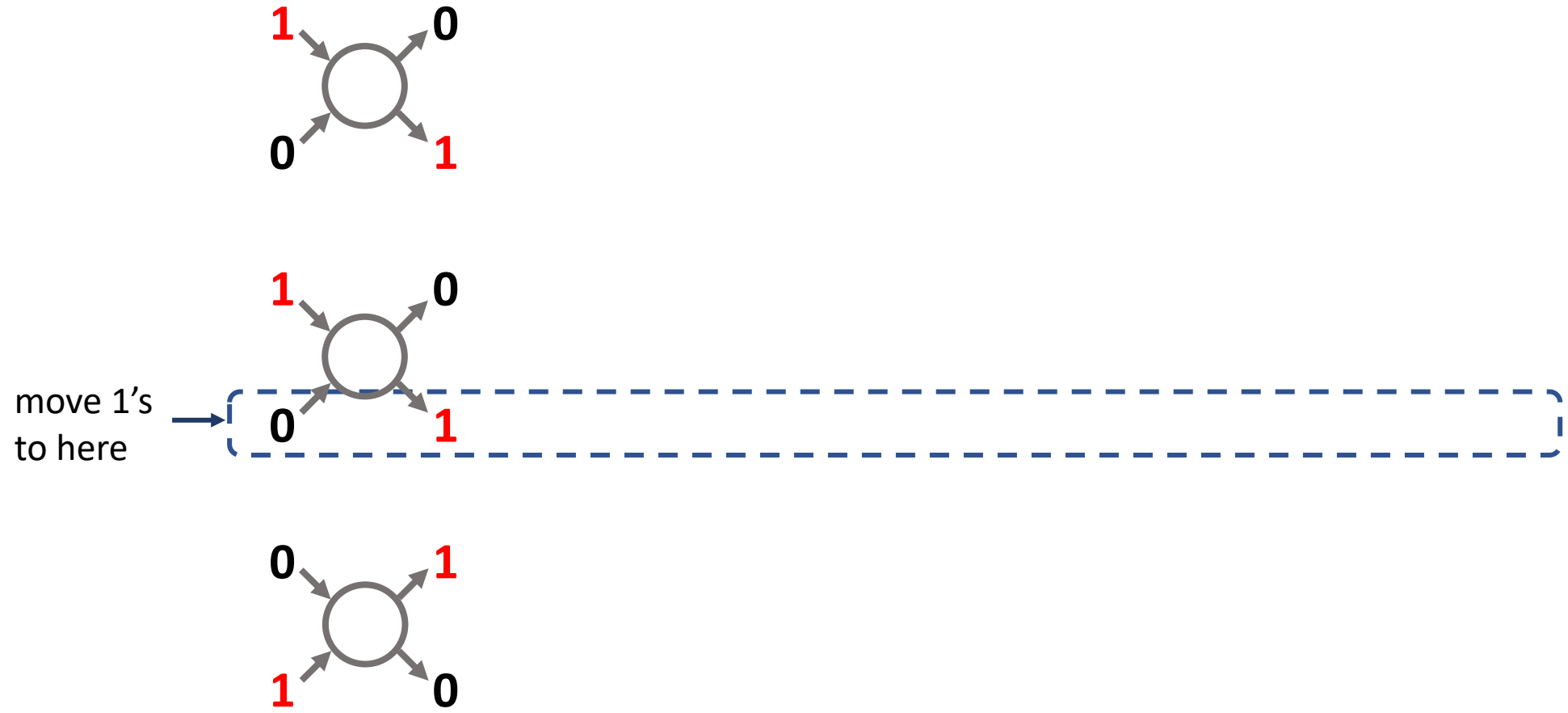
Odd bits



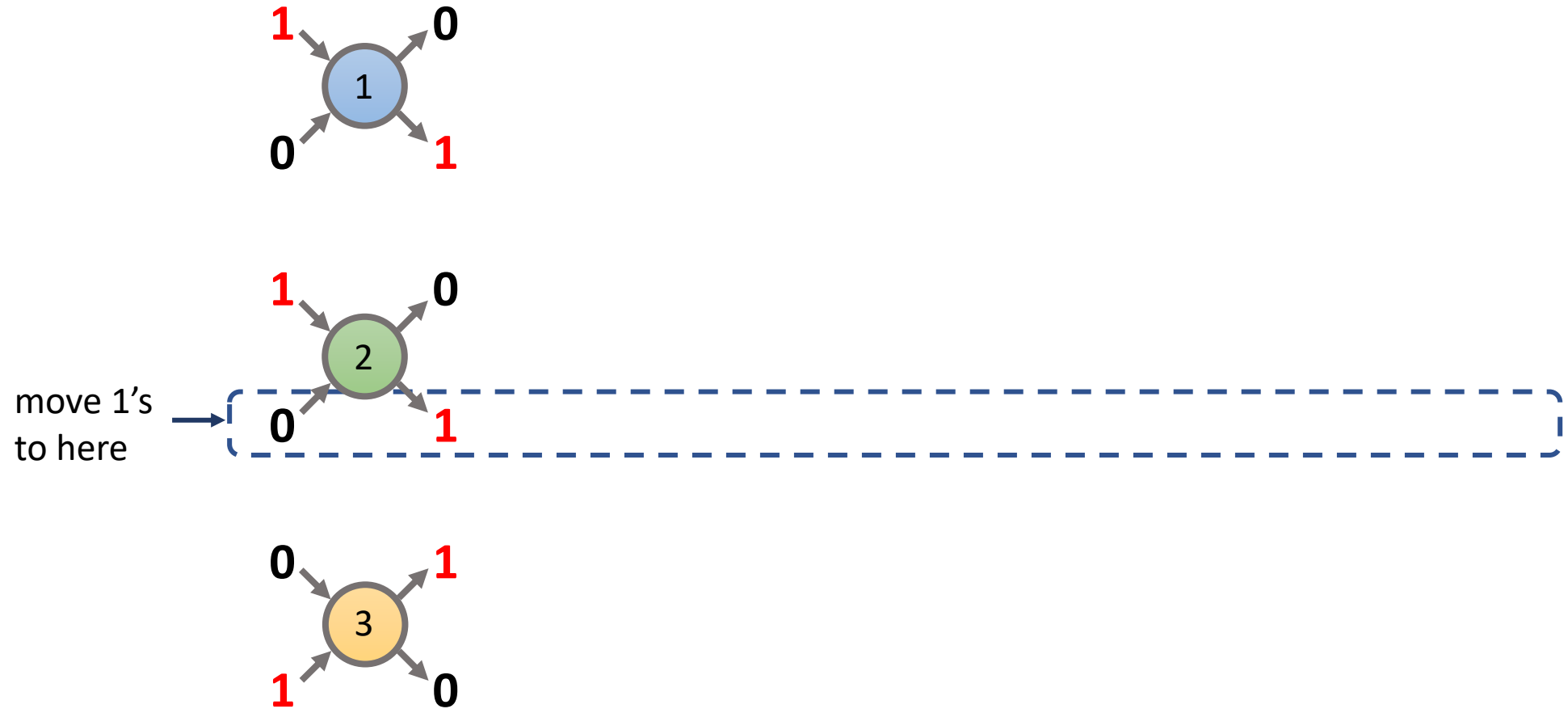
Odd bits



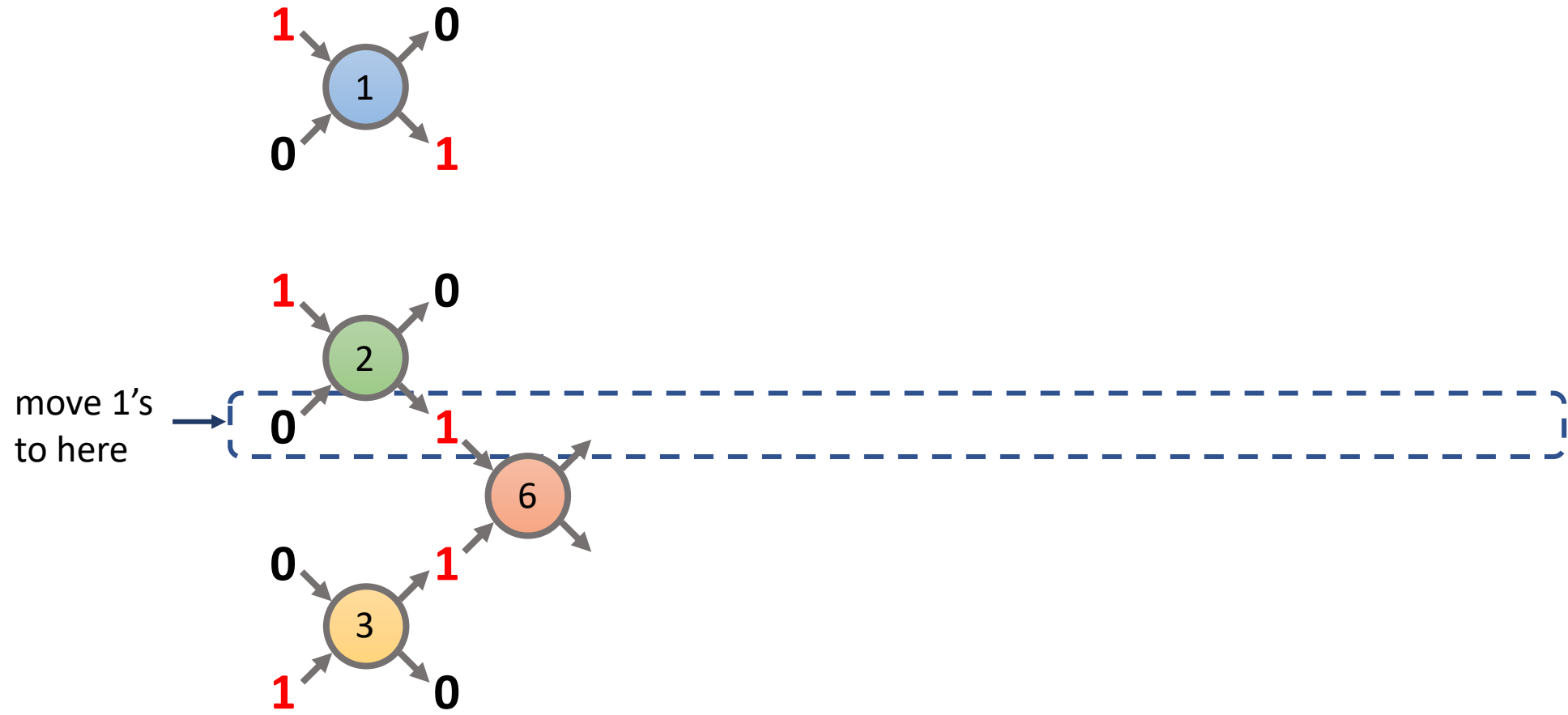
Odd bits



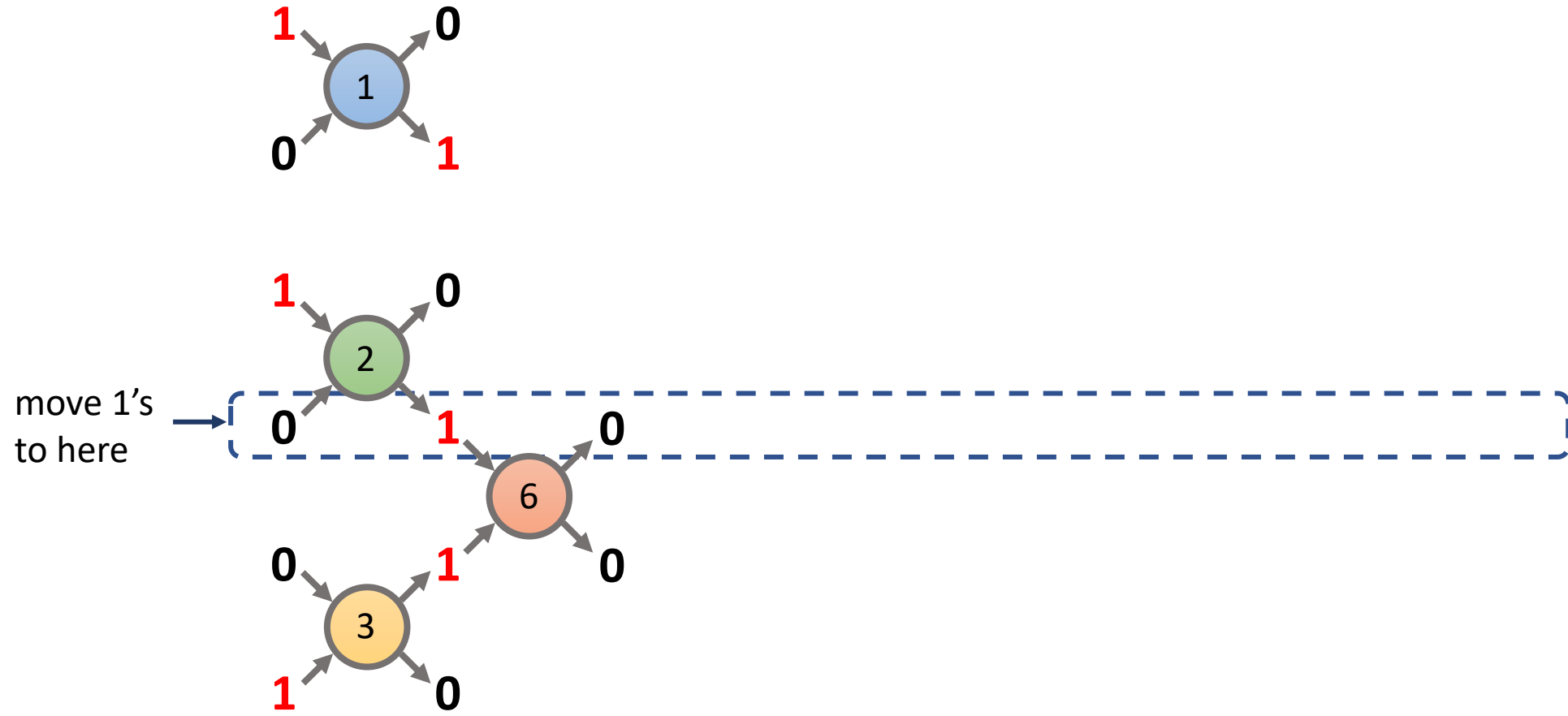
Odd bits



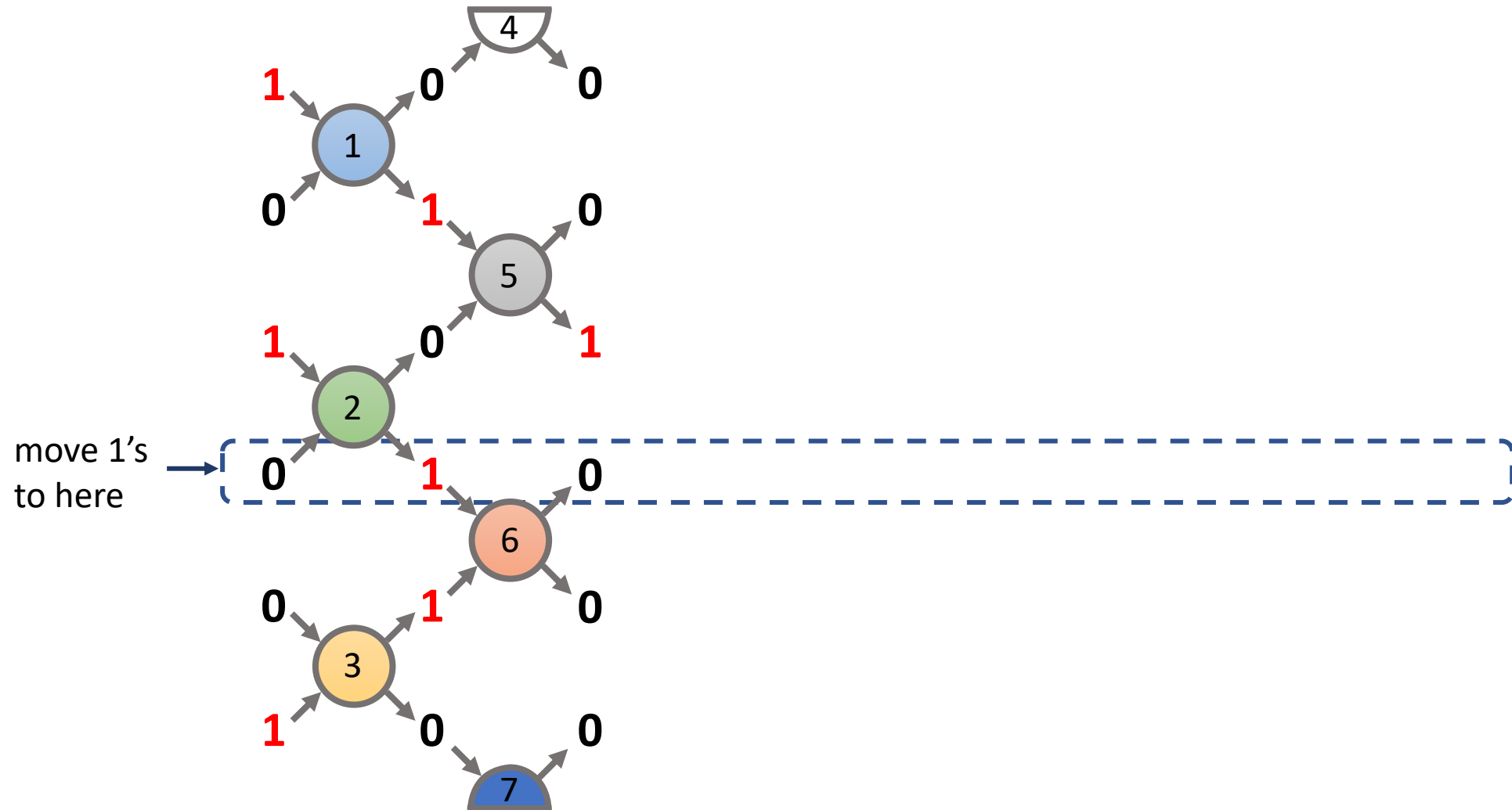
Odd bits



Odd bits

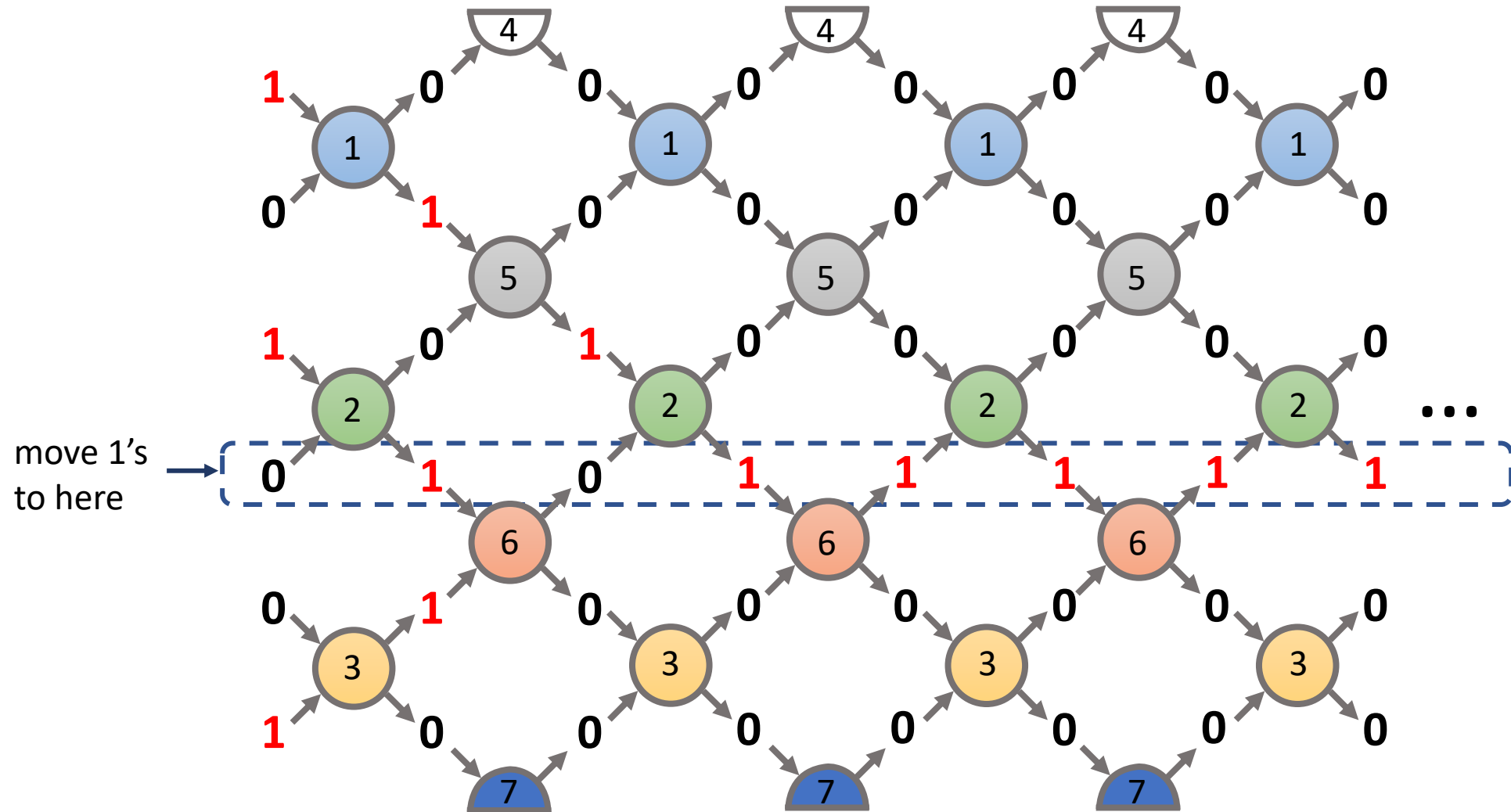


Odd bits



Odd bits

a.k.a. parity



Parity

1

0

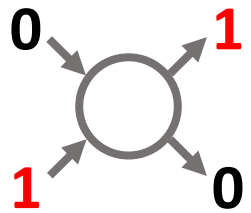
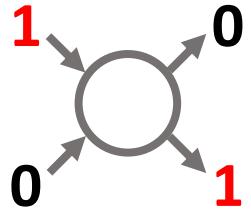
1

1

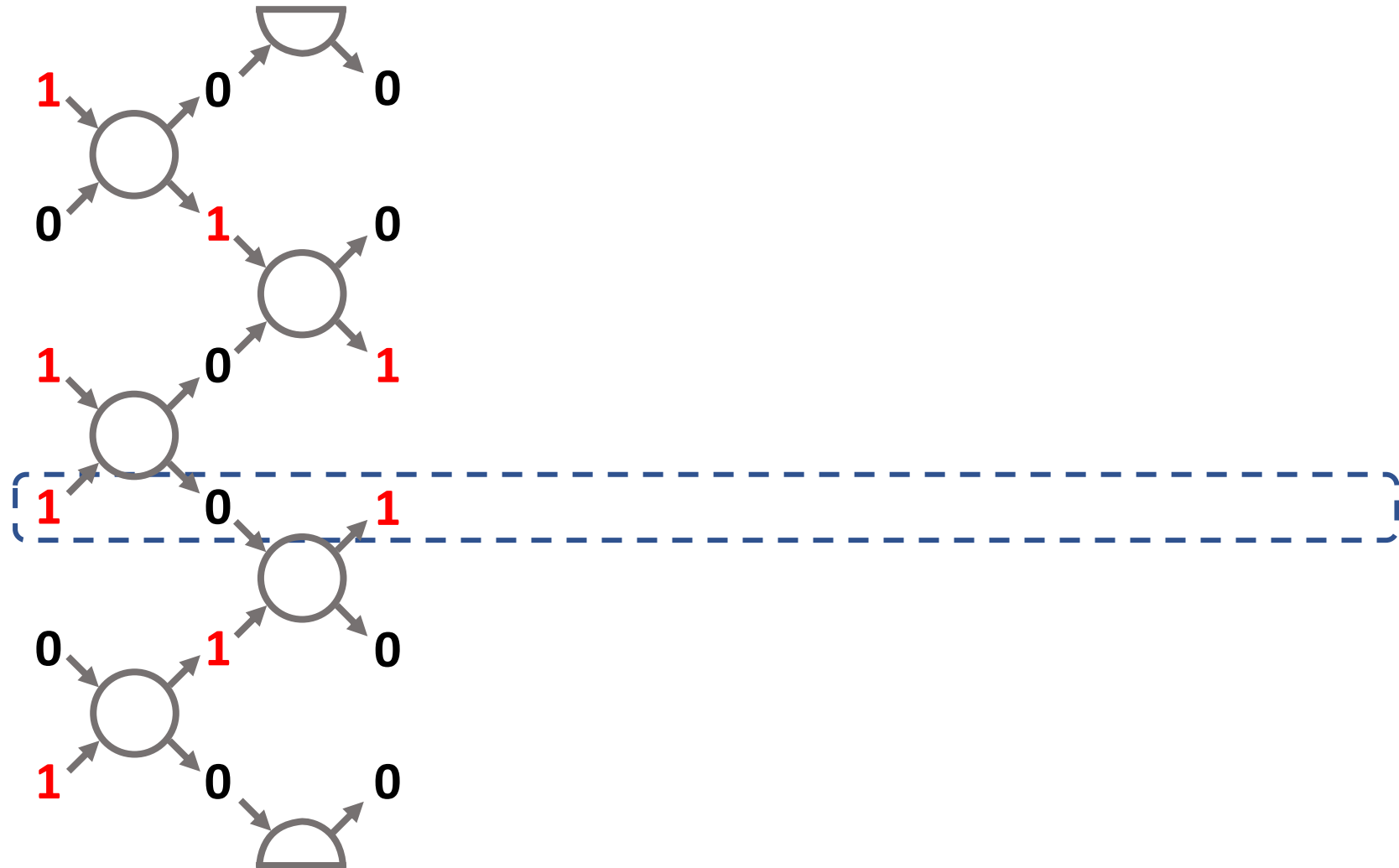
0

1

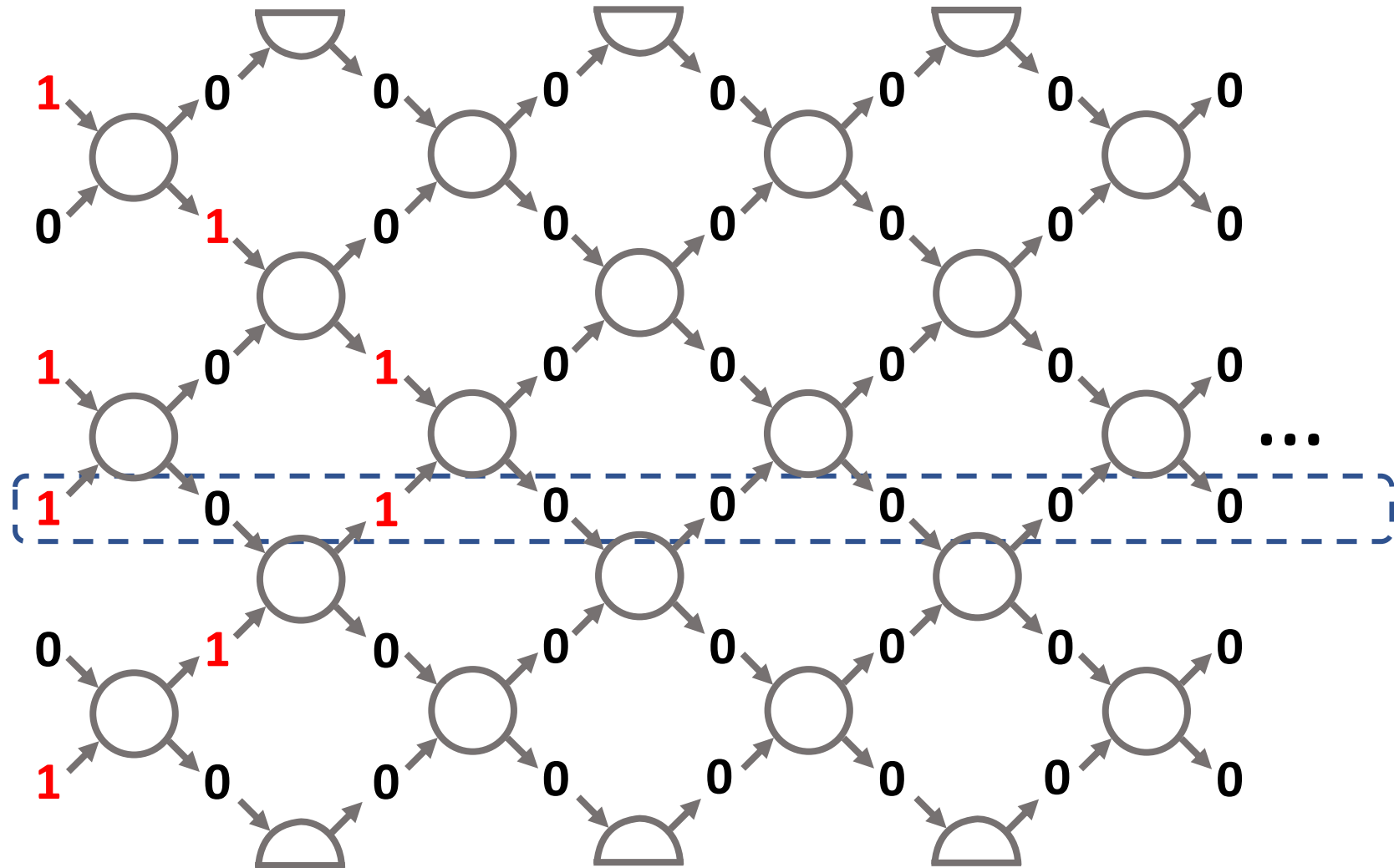
Parity



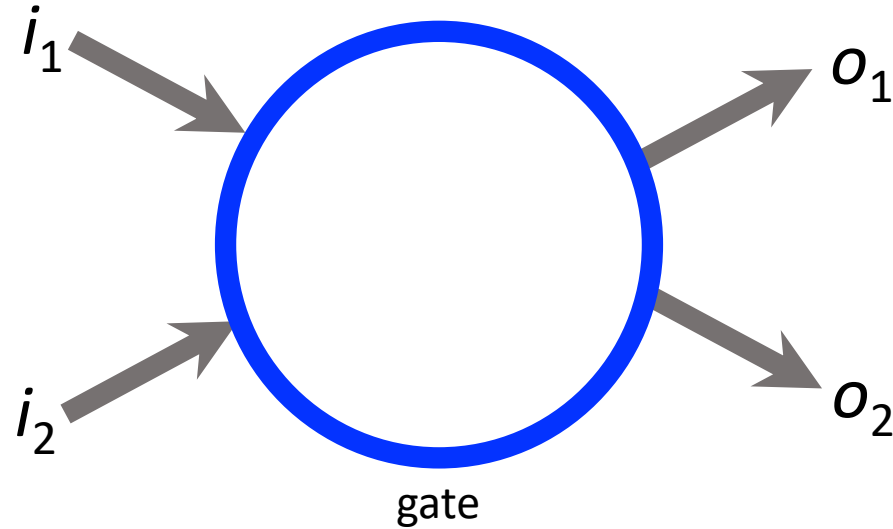
Parity



Parity

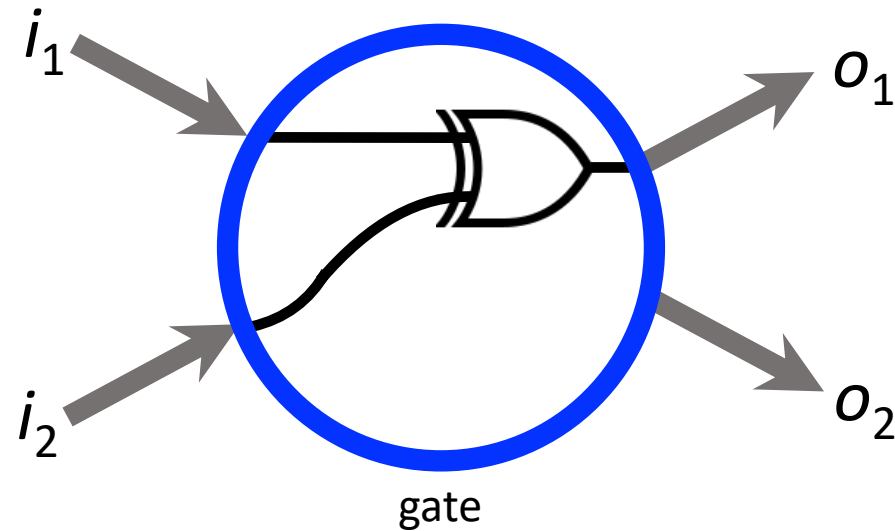


Circuit model



gate: function with two input bits i_1, i_2
and two output bits o_1, o_2

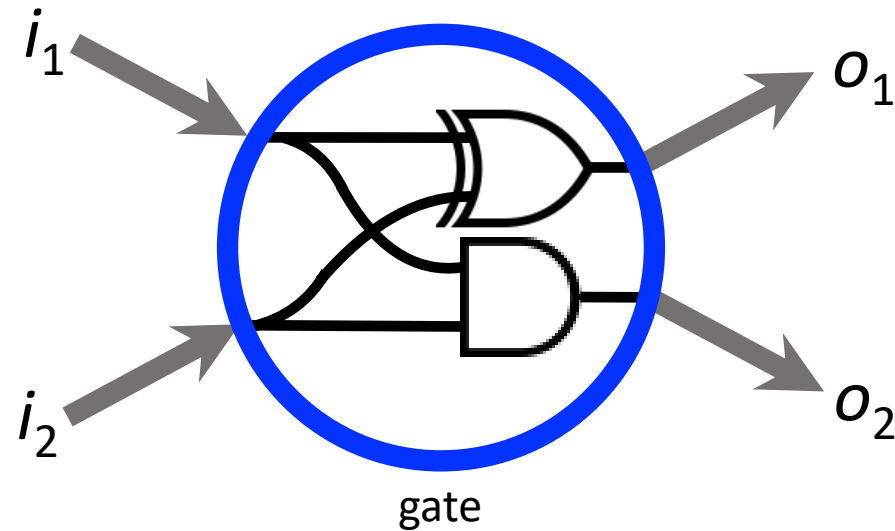
Circuit model



truth table			
i_1	i_2	o_1	o_2
0	0	0	
0	1	1	
1	0	1	
1	1	0	

gate: function with two input bits i_1, i_2
and two output bits o_1, o_2

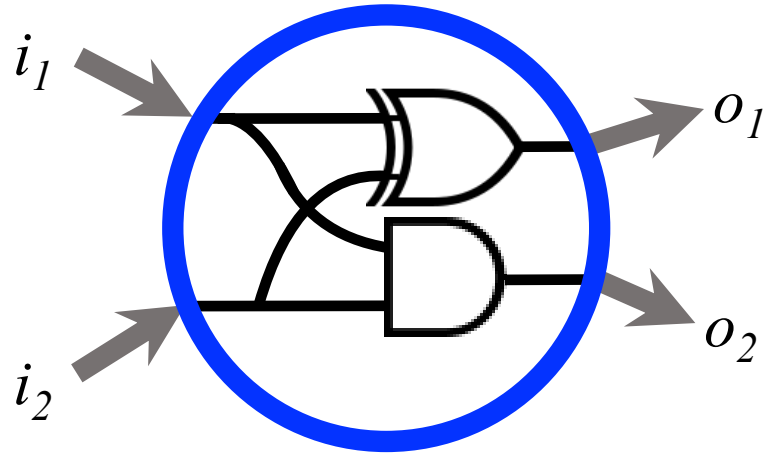
Circuit model



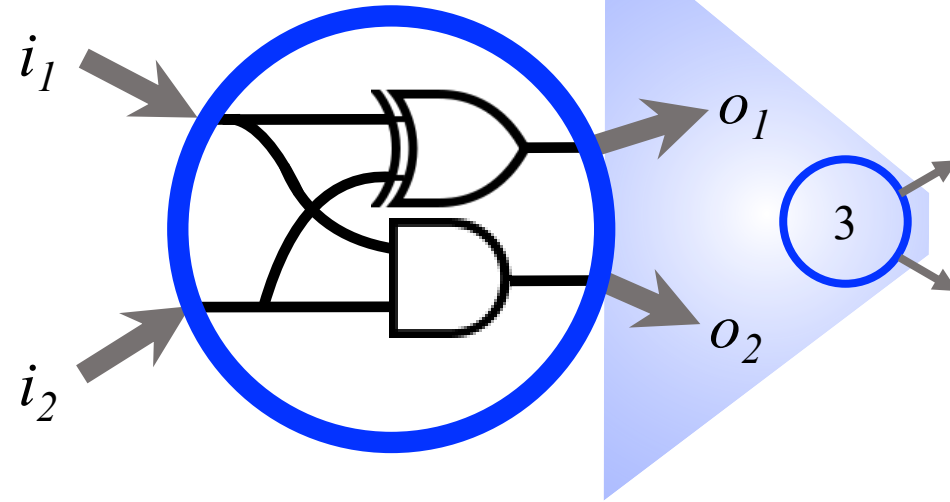
truth table			
i_1	i_2	o_1	o_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

gate: function with two input bits i_1, i_2
and two output bits o_1, o_2

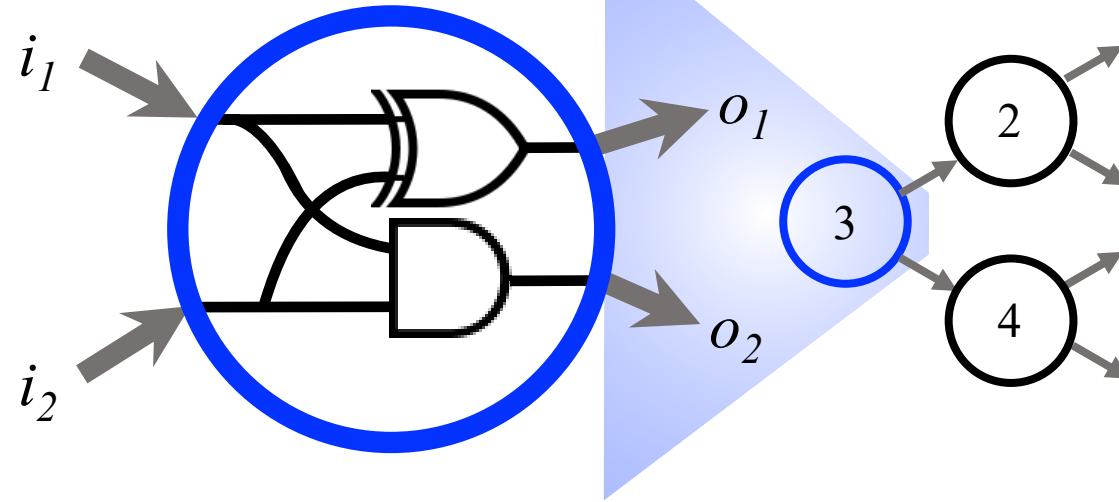
Circuit model



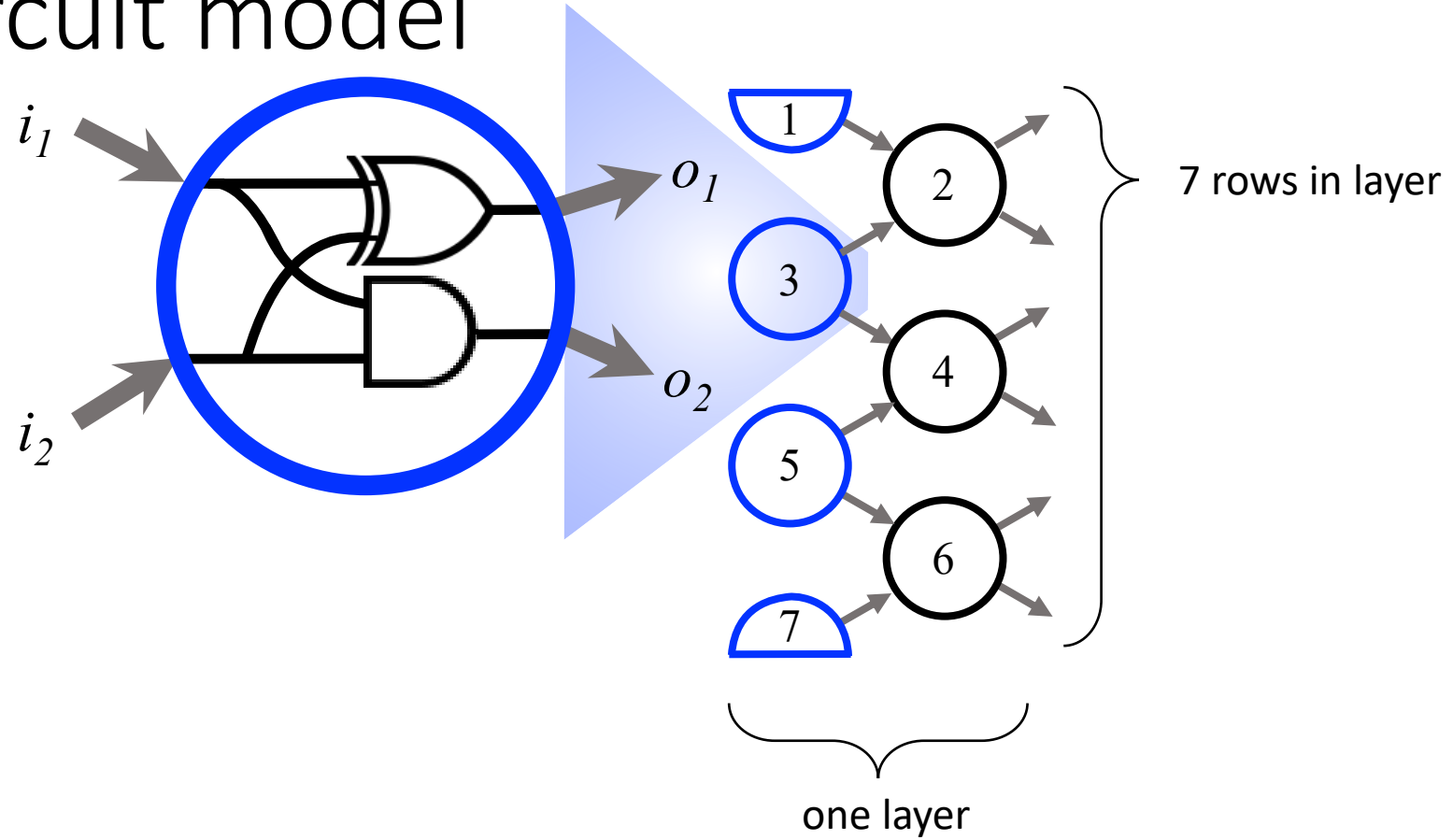
Circuit model



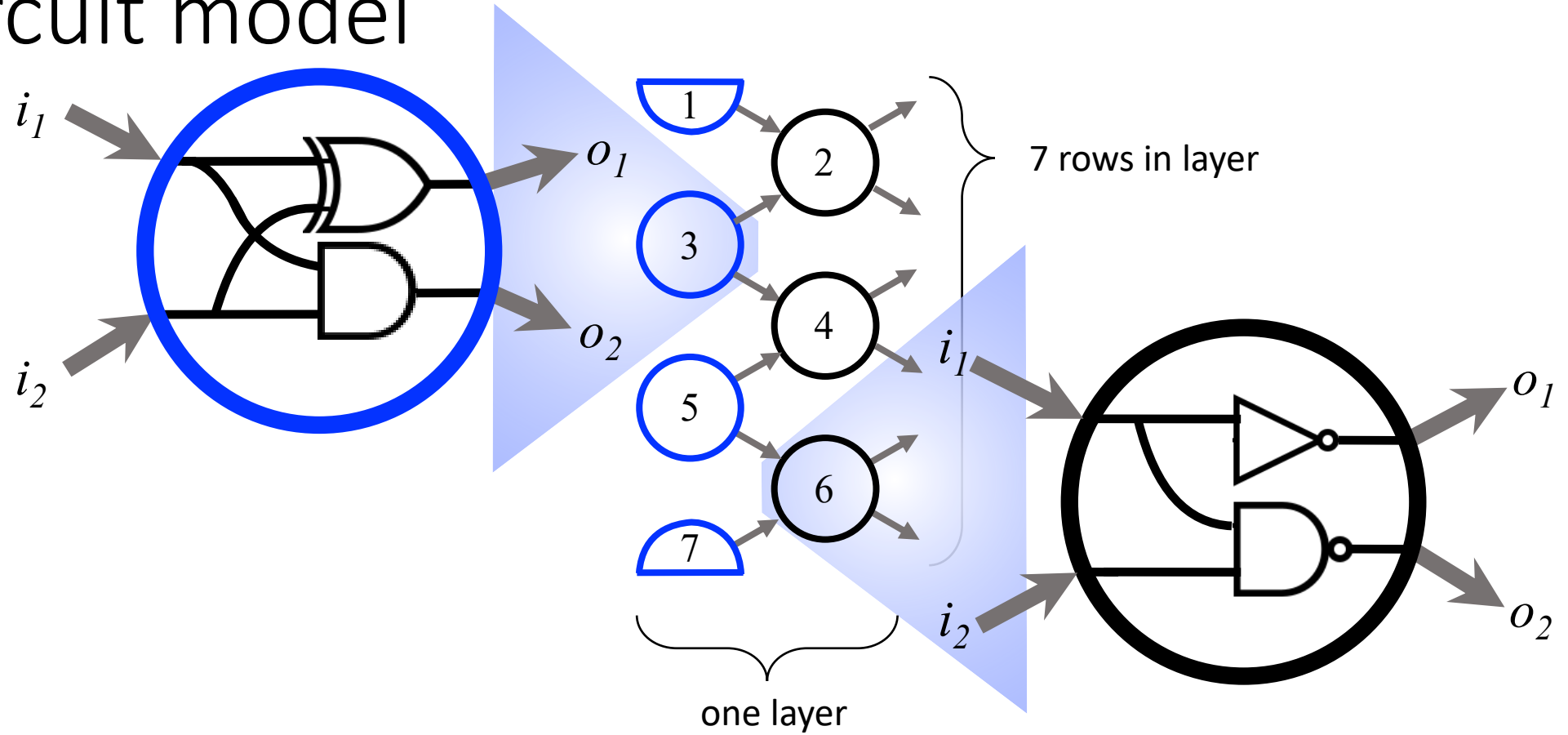
Circuit model



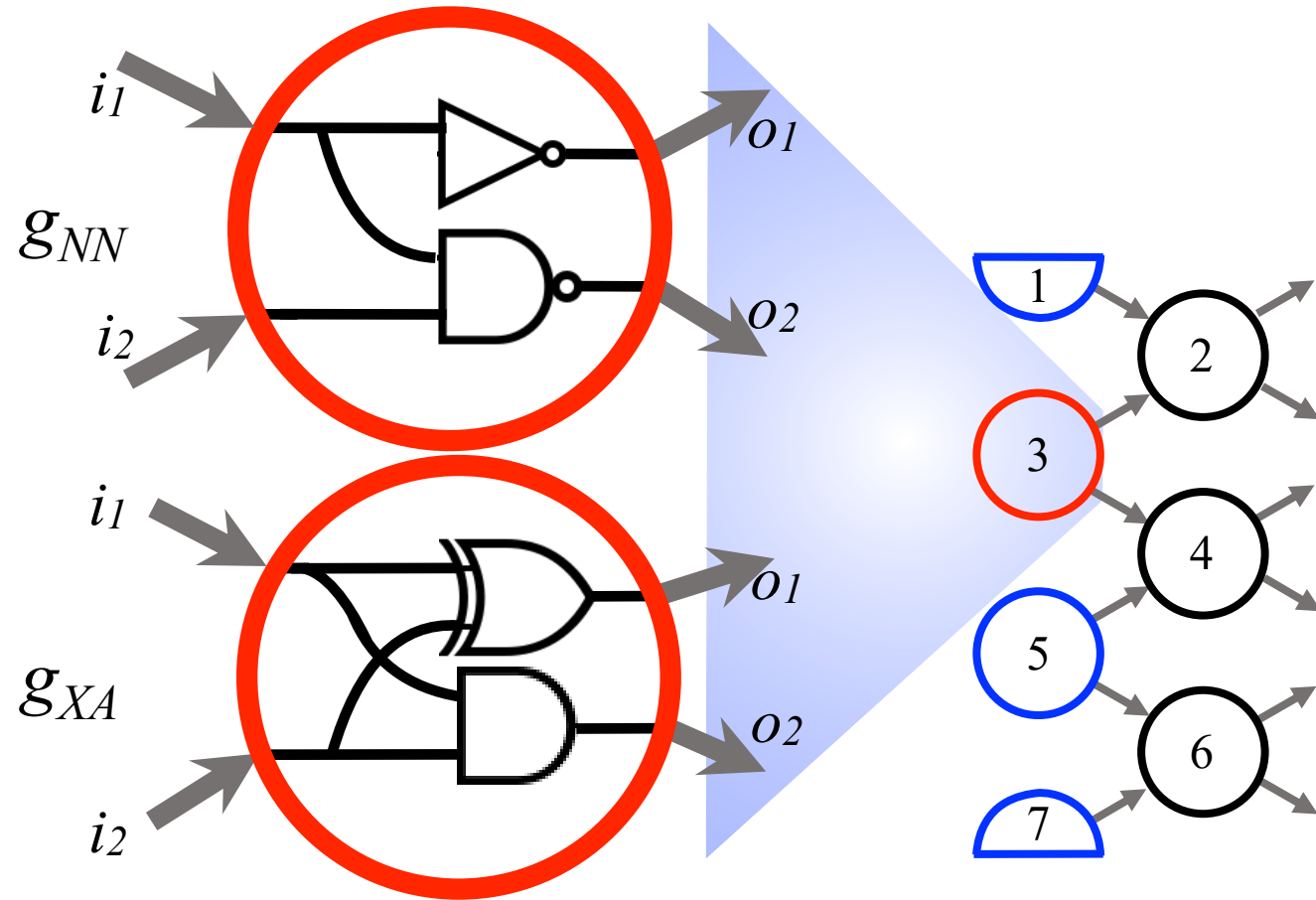
Circuit model



Circuit model



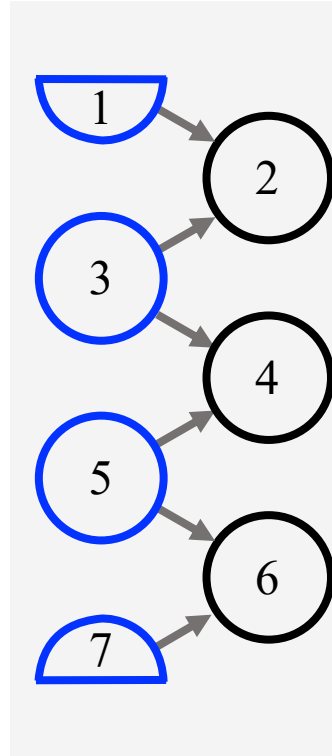
Circuit model



Randomization: Each row may be assigned ≥ 2 gates, with associated probabilities, e.g., $\Pr[g_{NN}] = \Pr[g_{XA}] = \frac{1}{2}$

Circuit model

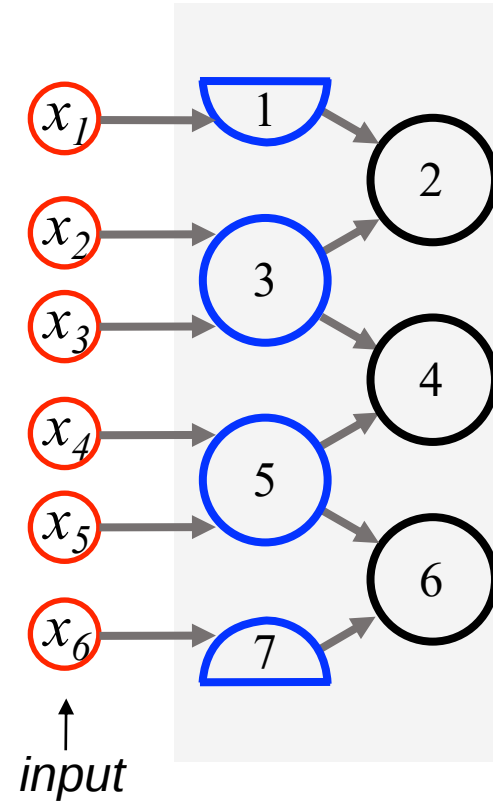
Programmer specifies layer:
gates to go in each row



Circuit model

Programmer specifies layer:
gates to go in each row

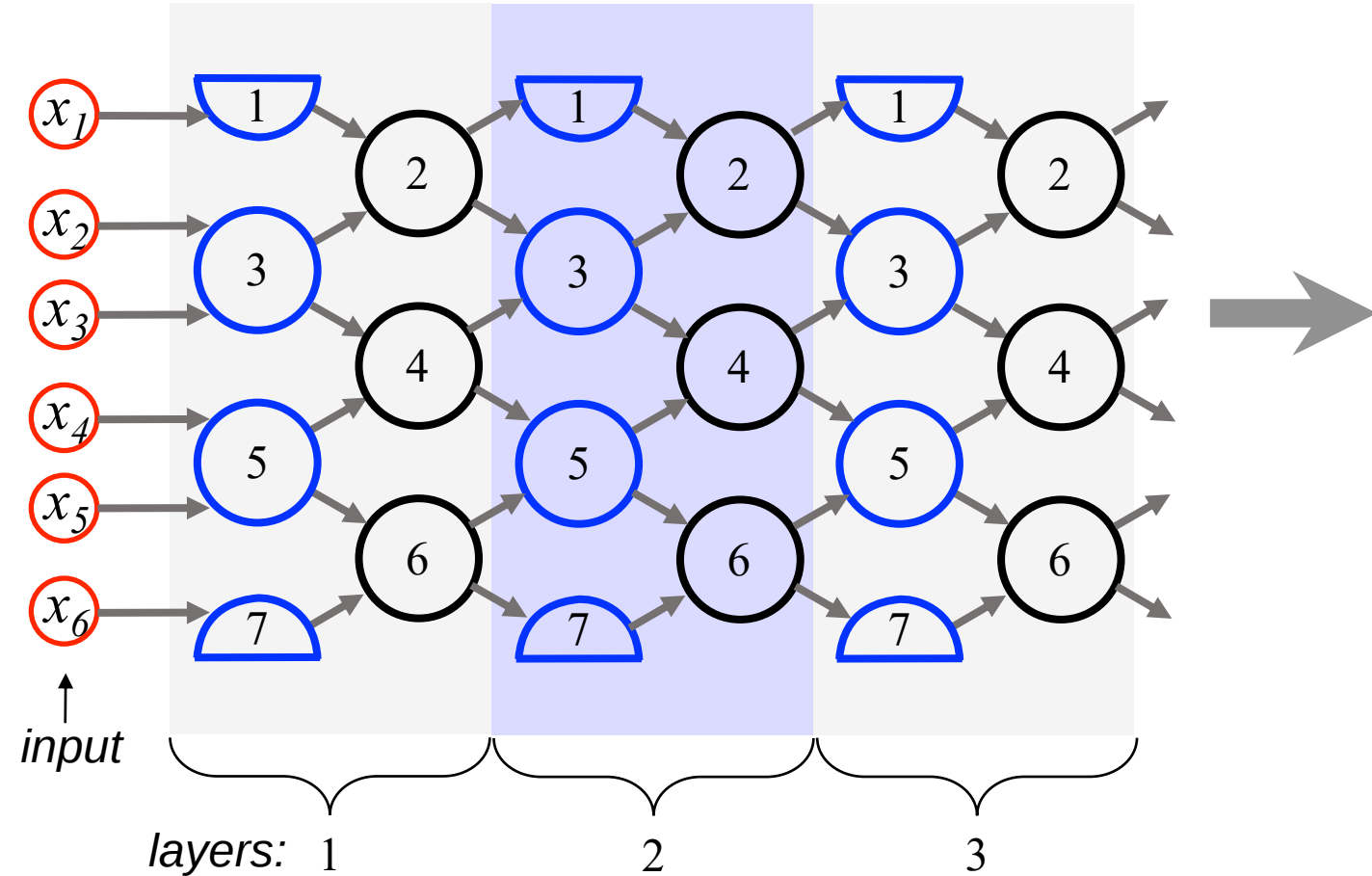
User gives n input bits



Circuit model

Programmer specifies layer:
gates to go in each row

User gives n input bits

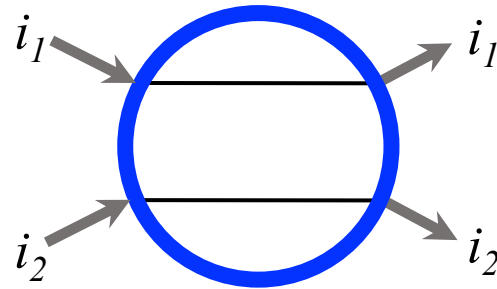


Example circuits with same gate in every row

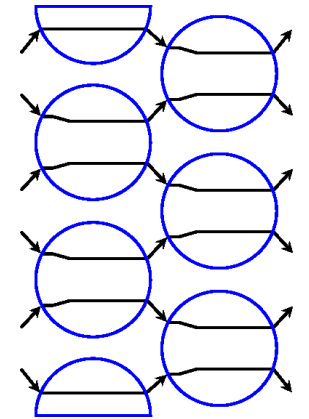
COPY



COPY gates

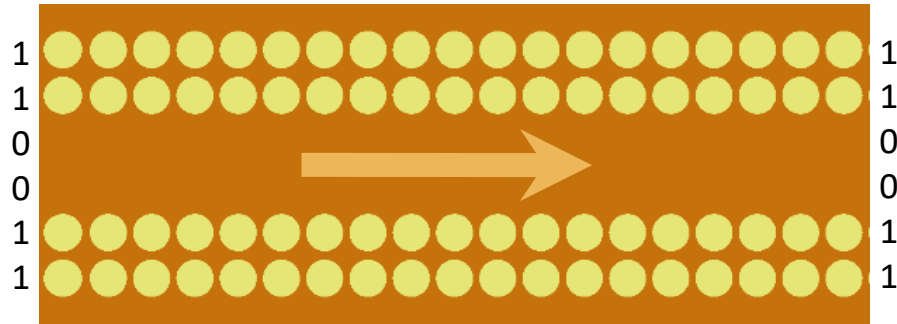


i_1	i_2	o_1	o_2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

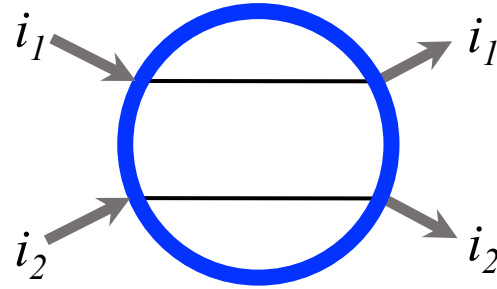


Example circuits with same gate in every row

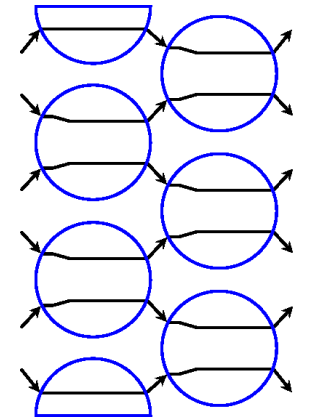
COPY



COPY gates

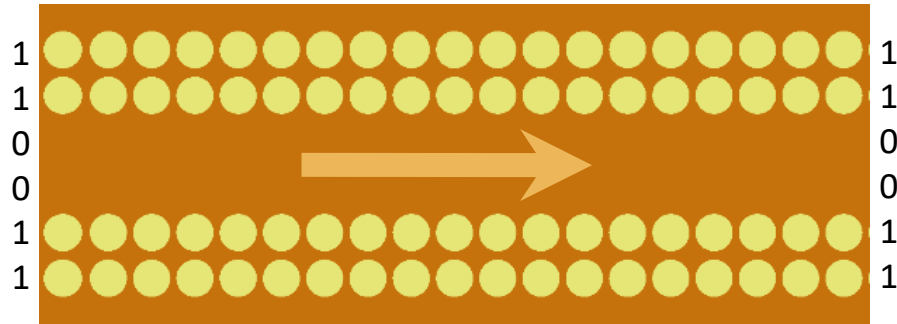


i_1	i_2	o_1	o_2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1

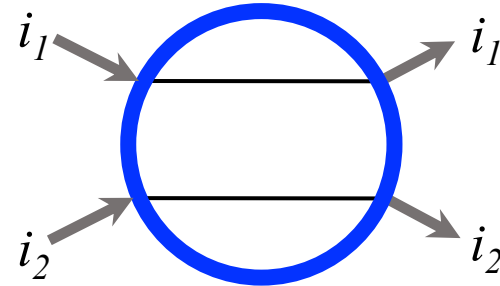


Example circuits with same gate in every row

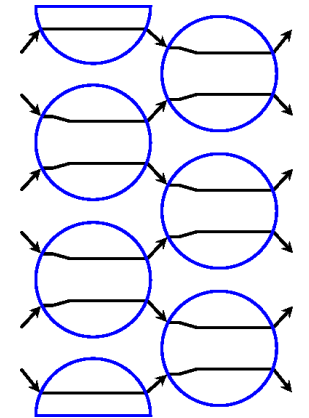
COPY



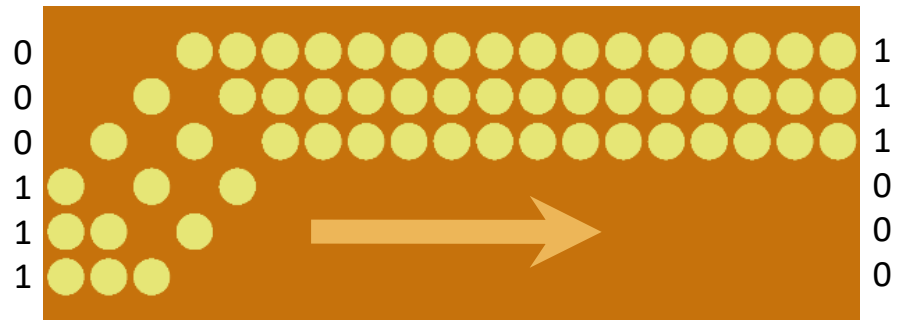
COPY gates



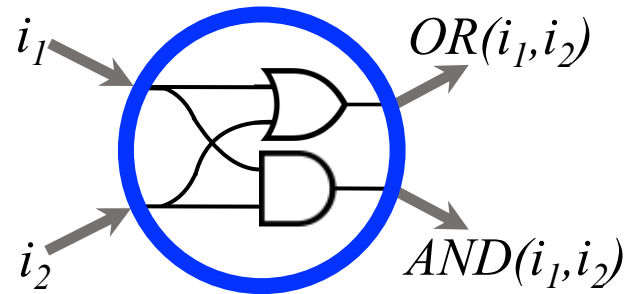
i_1	i_2	o_1	o_2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1



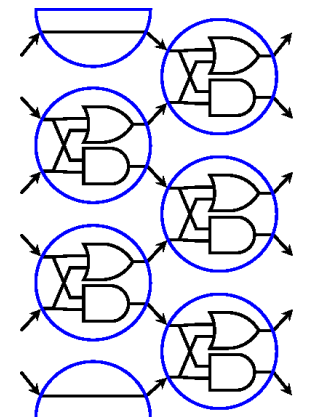
SORTING



SORTING gates

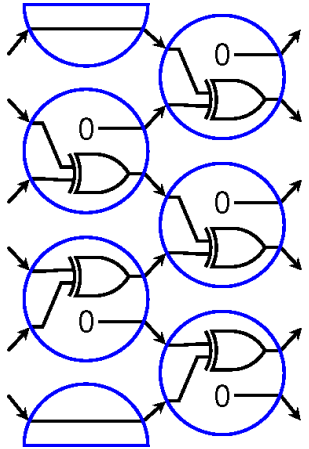


i_1	i_2	o_1	o_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1



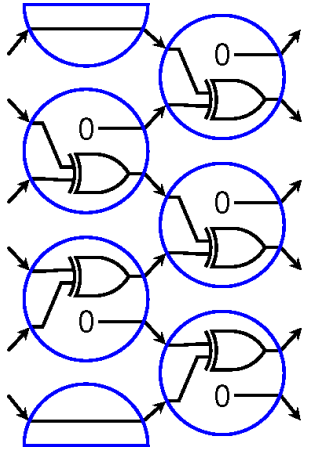
Example circuits with different gates in each row

PARITY



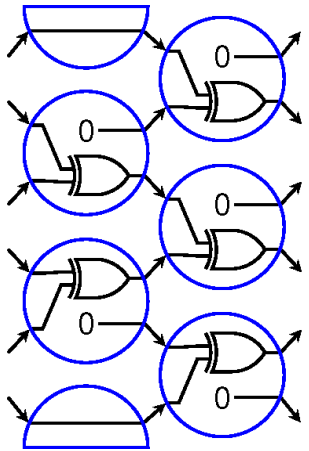
Example circuits with different gates in each row

PARITY

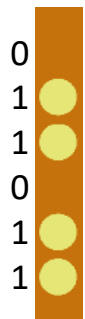
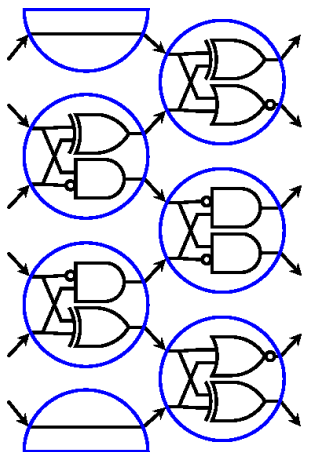


Example circuits with different gates in each row

PARITY



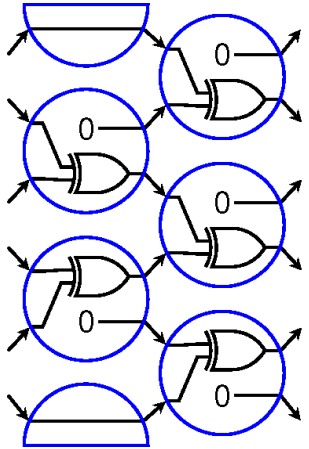
MULTIPLEOF3



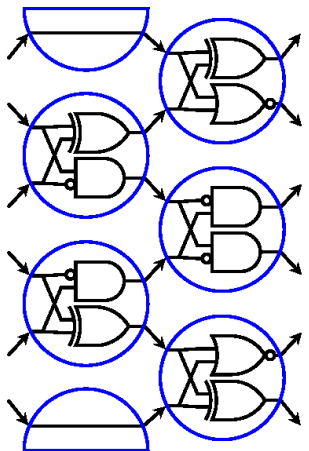
011011₂

Example circuits with different gates in each row

PARITY



MULTIPLEOF3

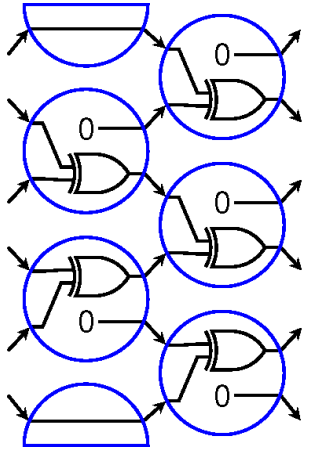


$$011011_2 = 27_{10} = 3 \cdot 9$$

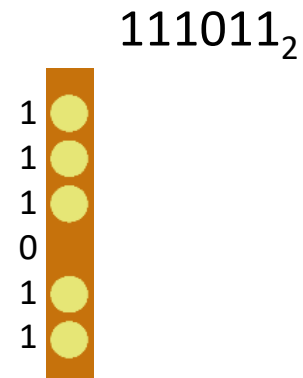
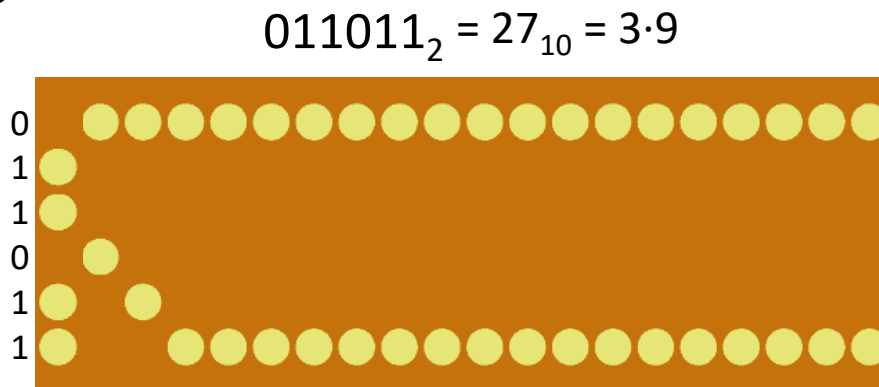
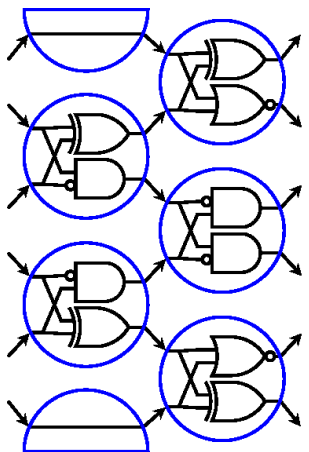


Example circuits with different gates in each row

PARITY

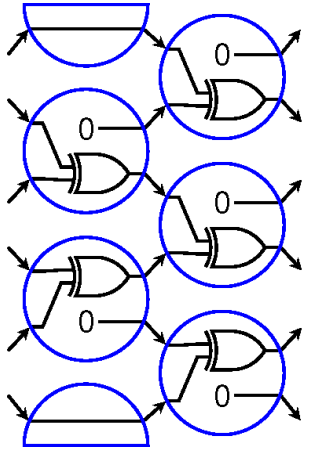


MULTIPLEOF3

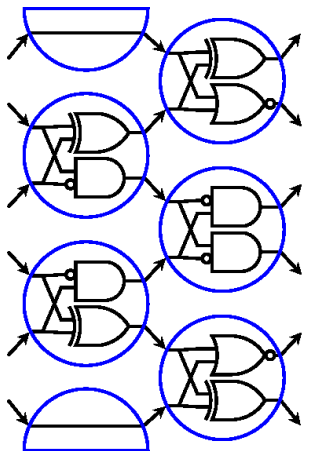


Example circuits with different gates in each row

PARITY



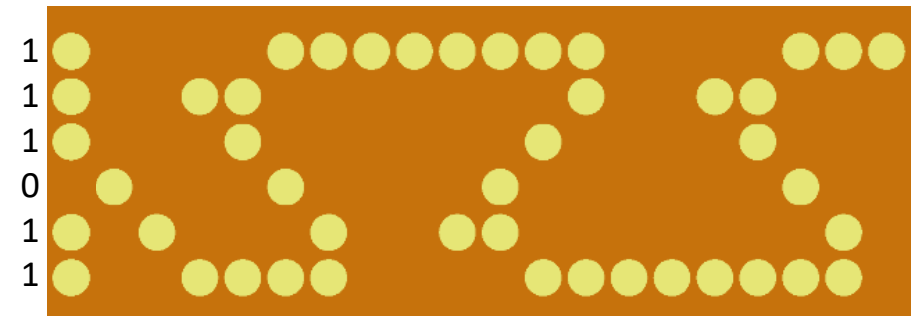
MULTIPLEOF3



$$011011_2 = 27_{10} = 3.9$$

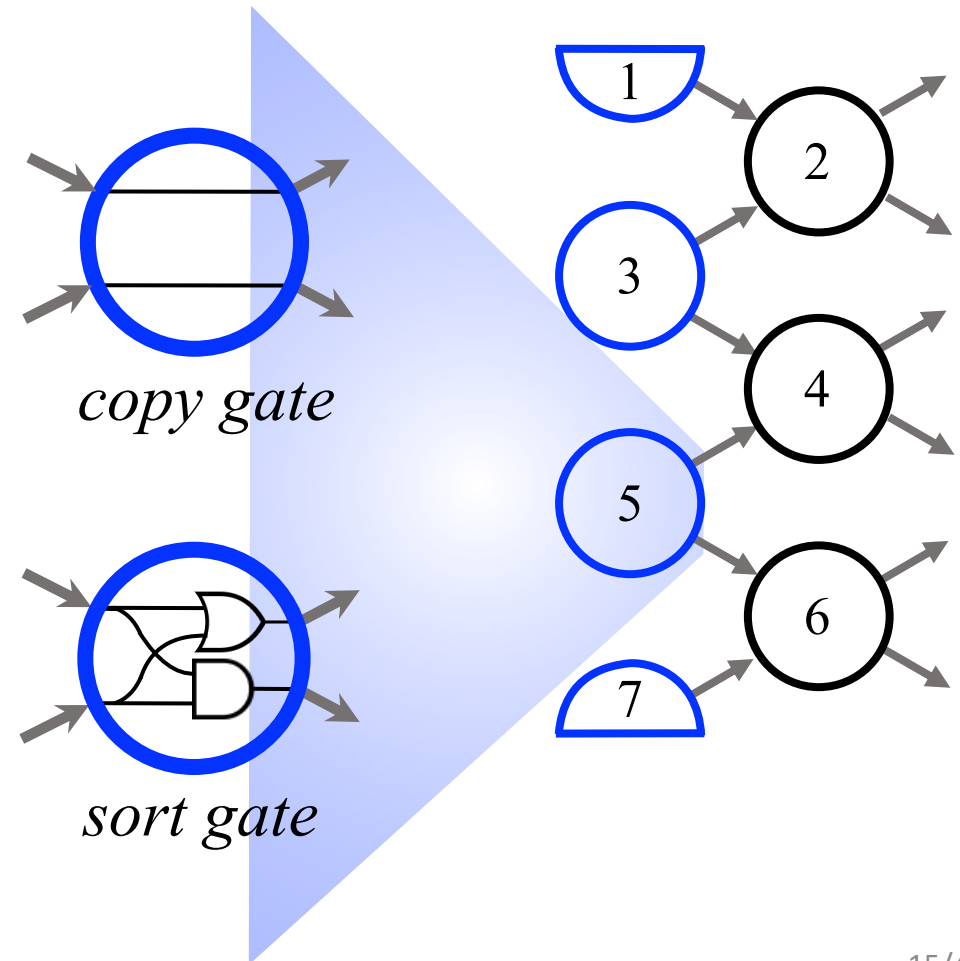


$$111011_2 = 59_{10} = 3 \cdot 19 + 2$$

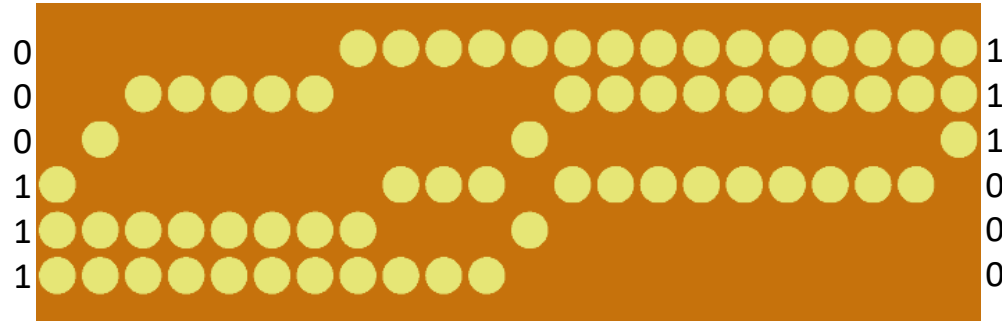


Randomization: “Lazy” sorting

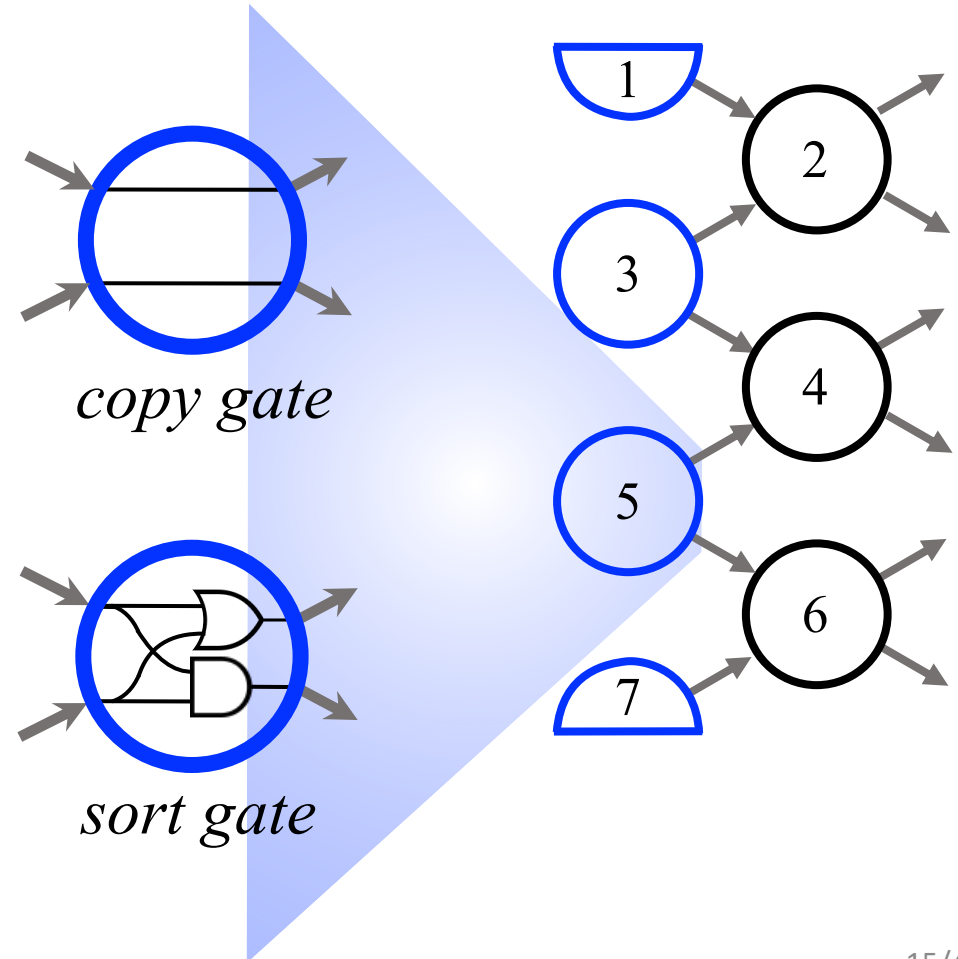
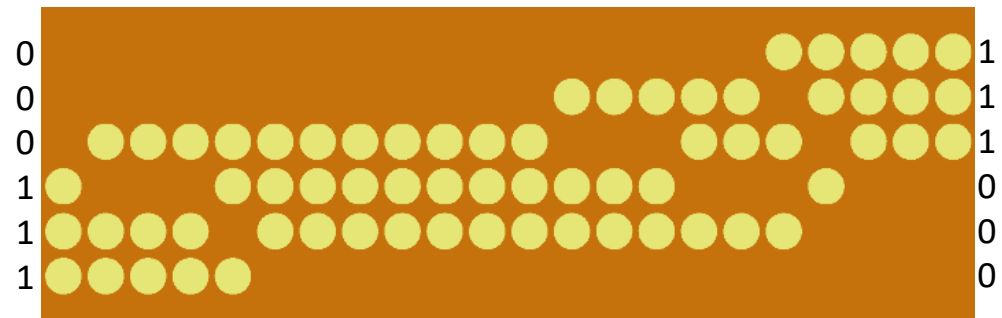
If 1 and 0 out of order, flip a coin to decide whether to swap them.



Randomization: “Lazy” sorting



If 1 and 0 out of order, flip a coin to decide whether to swap them.

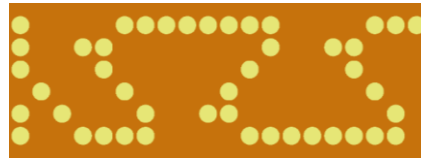


Deterministic circuits

PARITY



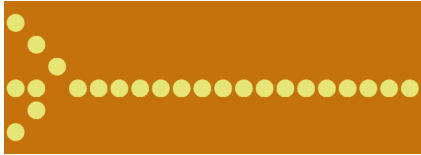
MULTIPLEOF3



answer yes/no question

Deterministic circuits

PARITY



MULTIPLEOF3



PALINDROME



yes



no

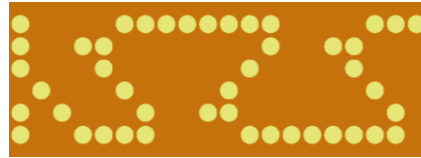
answer yes/no question

Deterministic circuits

PARITY



MULTIPLEOF3



PALINDROME

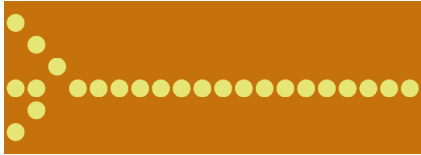


answer yes/no question



Deterministic circuits

PARITY



MULTIPLEOF3



PALINDROME

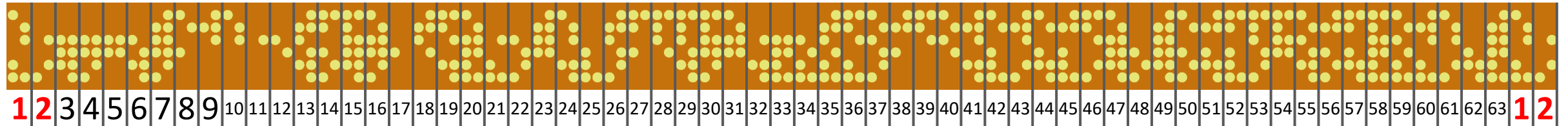


answer yes/no question



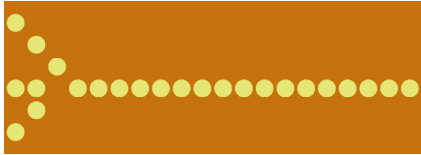
CYCLE63

"count" as high as possible



Deterministic circuits

PARITY



MULTIPLEOF3



PALINDROME

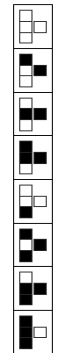
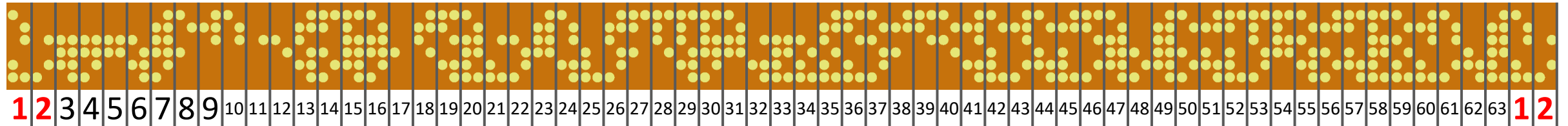


answer yes/no question



CYCLE63

"count" as high as possible

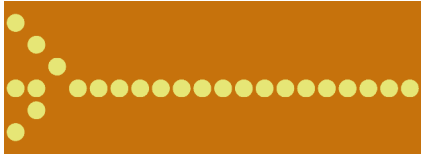


RULE110

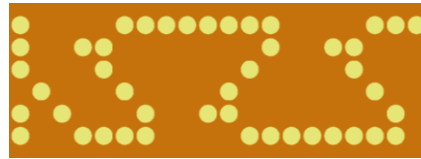
simulate cellular automata

Deterministic circuits

PARITY



MULTIPLEOF3



PALINDROME

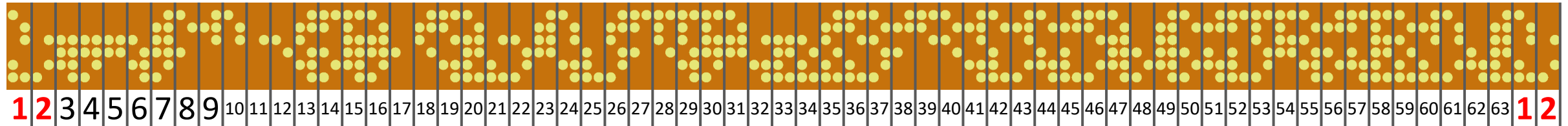


answer yes/no question



CYCLE63

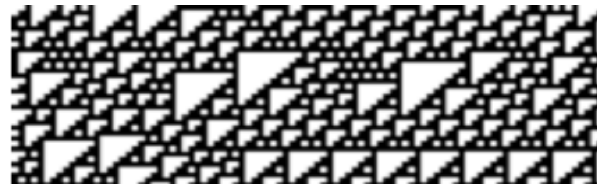
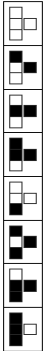
"count" as high as possible



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 1 2

RULE110

simulate cellular automata



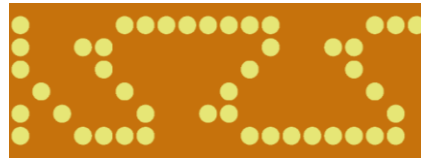
time →

Deterministic circuits

PARITY



MULTIPLEOF3



PALINDROME

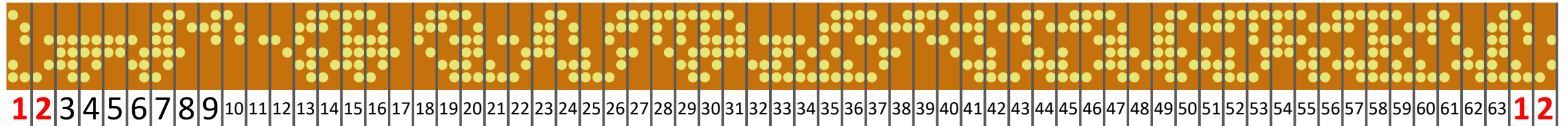


answer yes/no question



CYCLE63

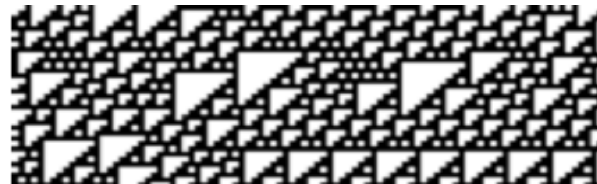
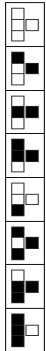
“count” as high as possible



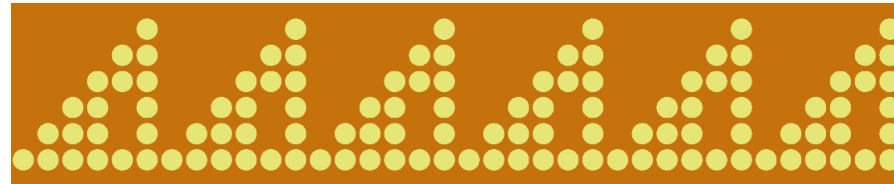
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 1 2

RULE110

simulate cellular automata



time →



Theorem: Rule 110 can efficiently execute any algorithm.

[Cook, Complex Systems 2004]

[Neary, Woods, ICALP 2006]

Randomized circuits

LAZYPARITY

Randomized circuits

LAZYPARITY



Randomized circuits

LAZYPARITY



RANDOMWALKINGBIT



Randomized circuits

LAZYPARITY



RANDOMWALKINGBIT



DIAMONDSAREFOREVER



Randomized circuits

LAZYPARITY



RANDOMWALKINGBIT



DIAMONDSAREFOREVER



FAIRCOIN

use biased coin to
simulate unbiased coin



Randomized circuits

LAZYPARITY



RANDOMWALKINGBIT



DIAMONDSAREFOREVER



FAIRCOIN

use biased coin to
simulate unbiased coin

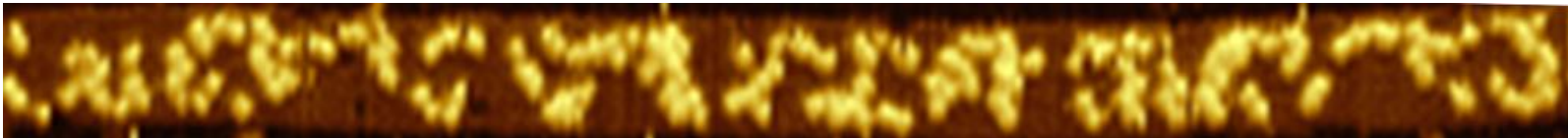
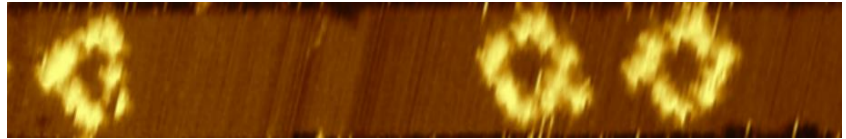
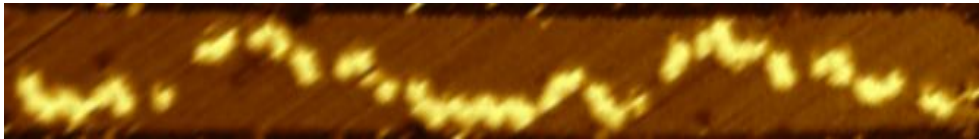
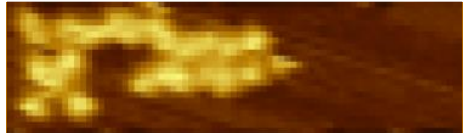
$$\Pr \left[\text{circuit diagram} \right] = \Pr \left[\text{circuit diagram} \right] = \frac{1}{2}$$

for **any** (positive) probabilities for the randomized gate





100 nm



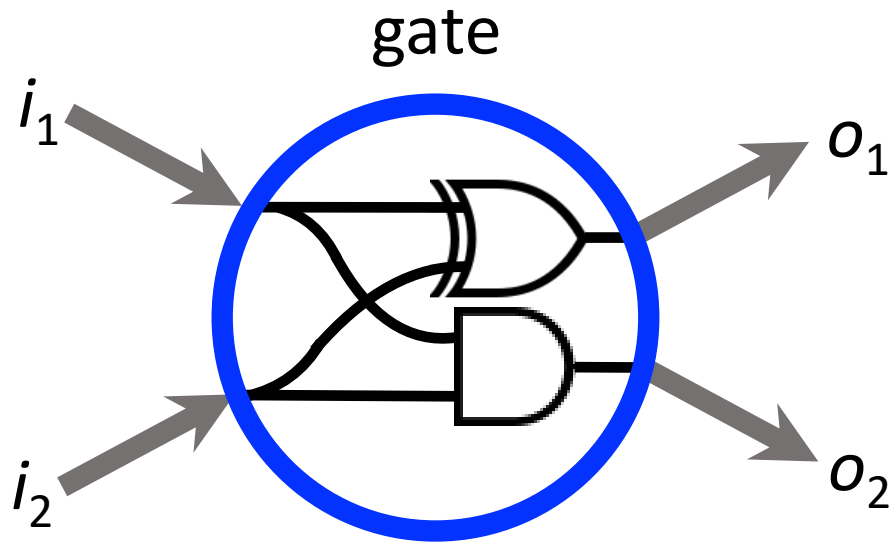
Hierarchy of abstractions

Bits: Boolean circuits compute

➔ **Tiles: Tile growth implements circuits**

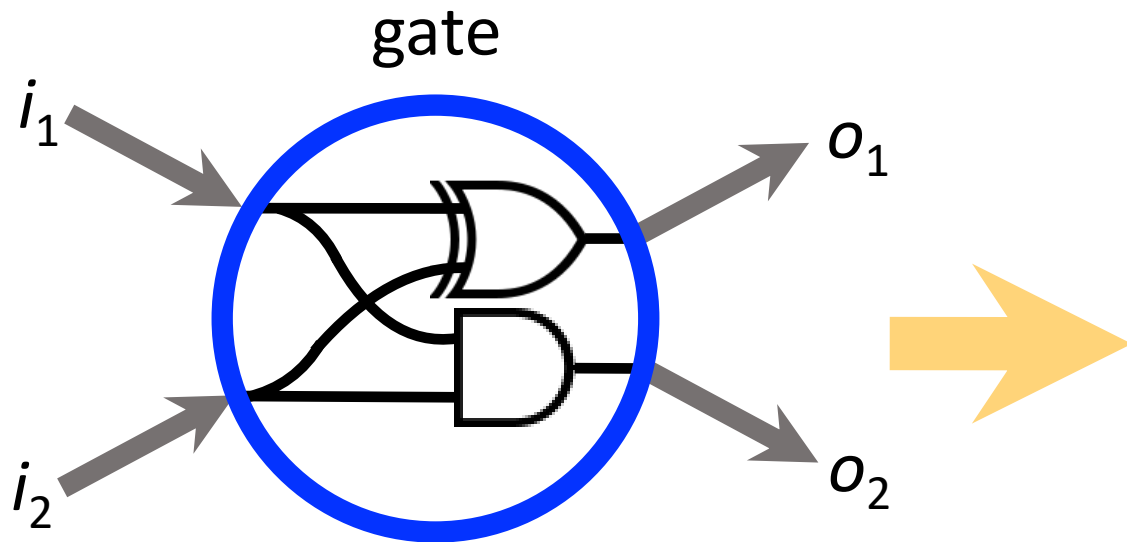
DNA: DNA strands implement tiles

Gates $\rightarrow$ Tiles



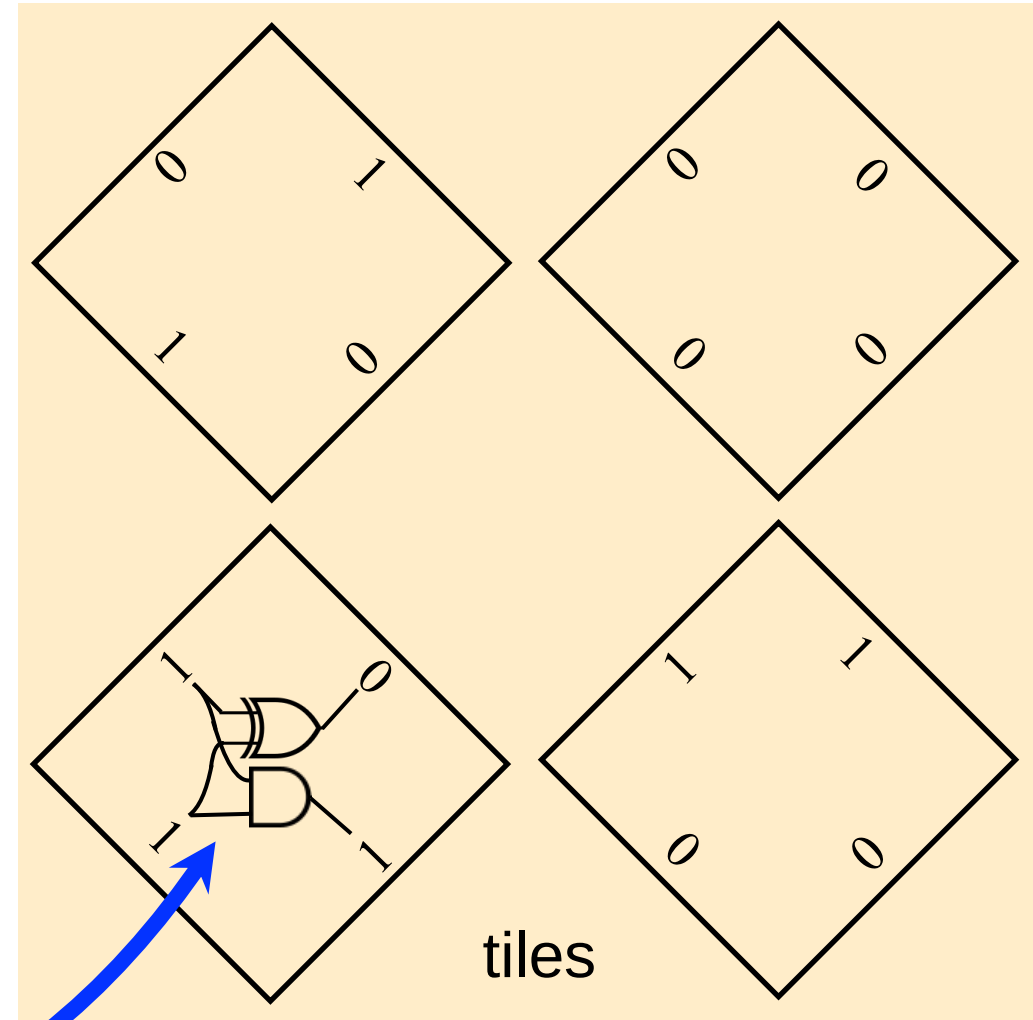
i_1	i_2	o_1	o_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Gates $\rightarrow$ Tiles

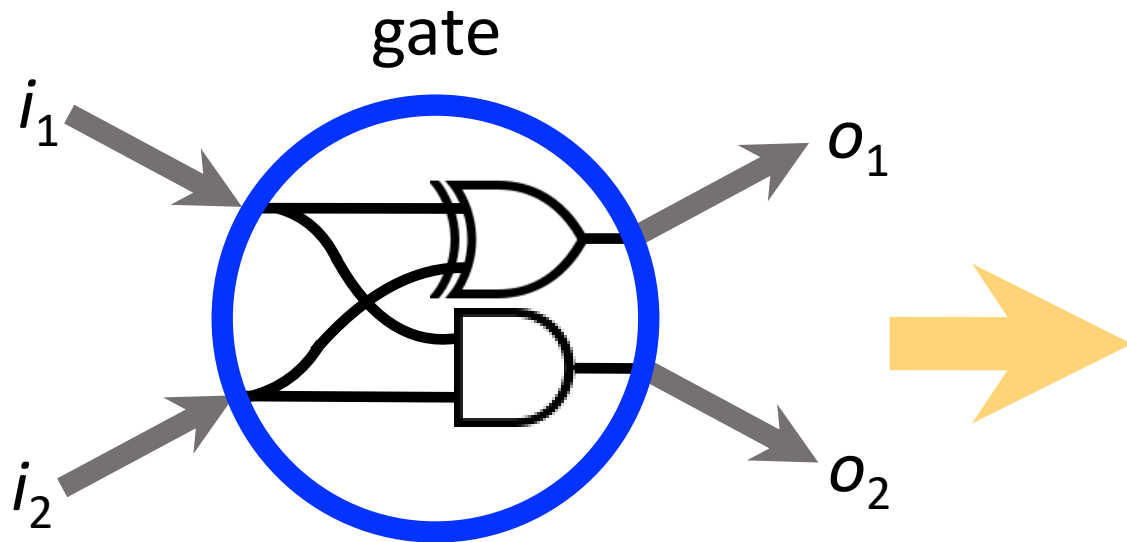


i_1	i_2	o_1	o_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

truth table row is
encoded by a **tile** with
4 **glues** encoding bits

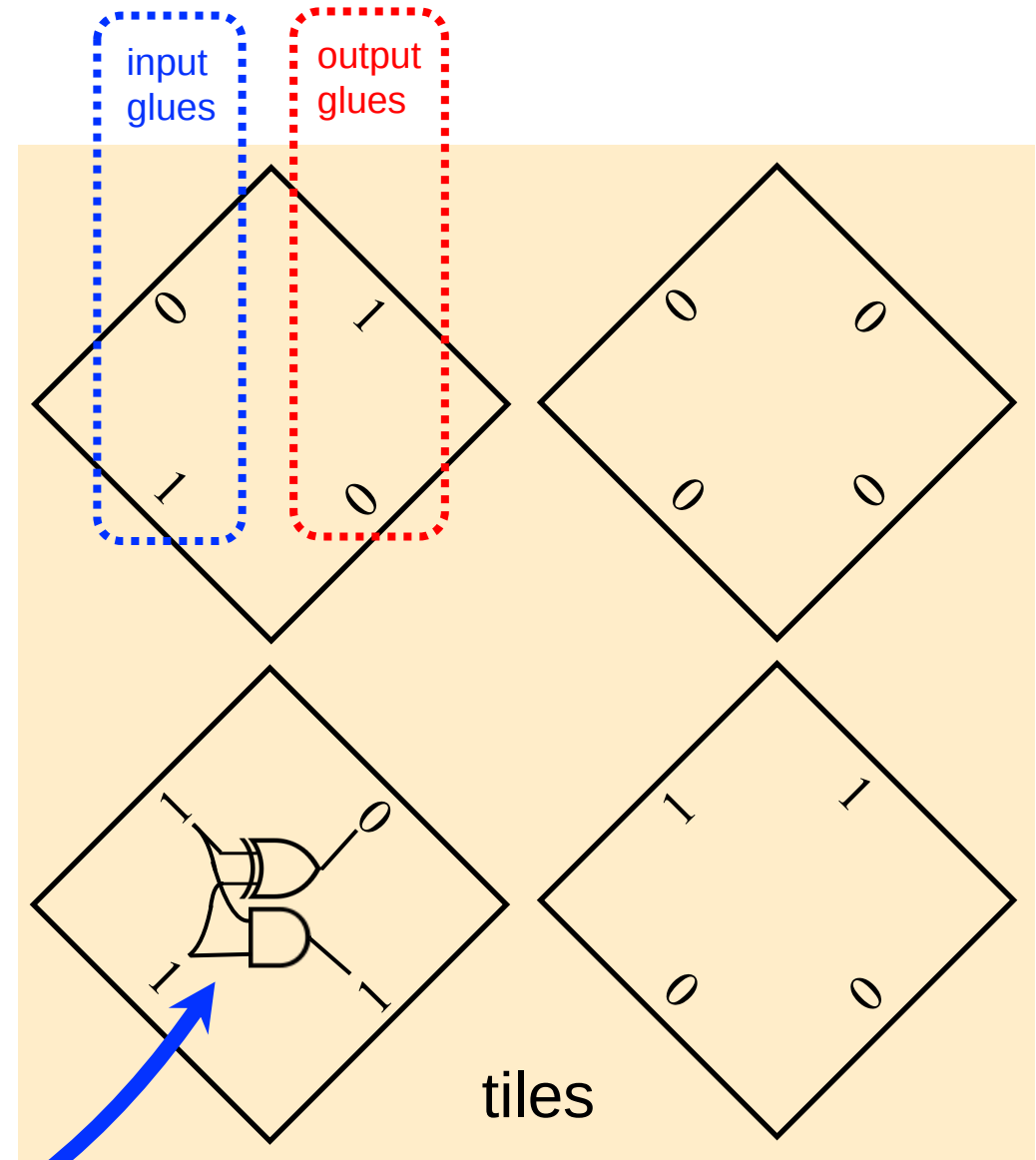


Gates $\rightarrow$ Tiles

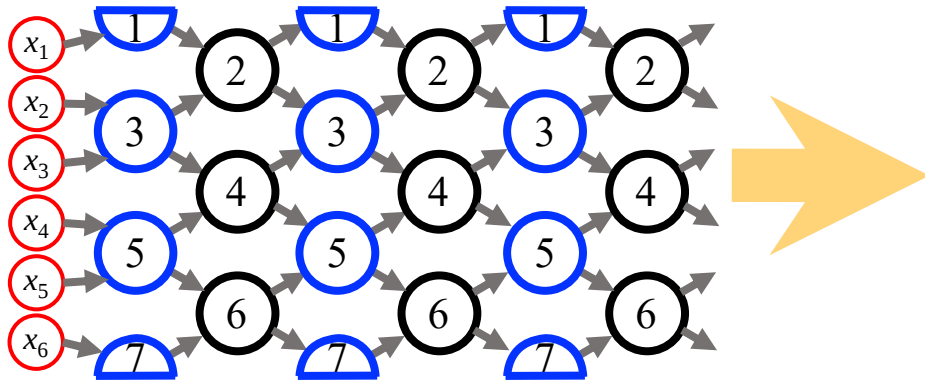


i_1	i_2	o_1	o_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

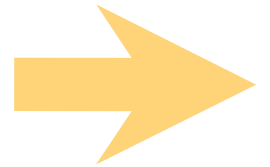
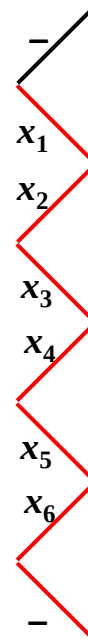
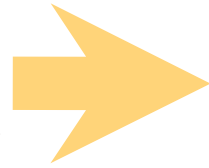
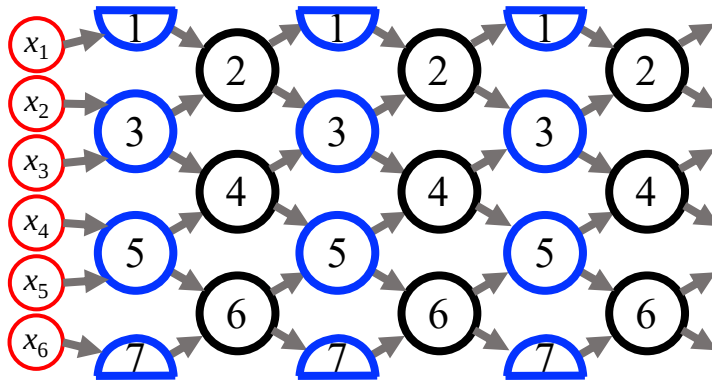
truth table row is
encoded by a **tile** with
4 **glues** encoding bits



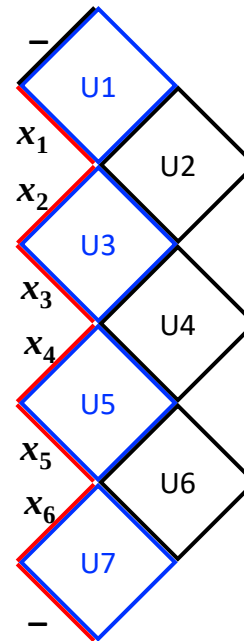
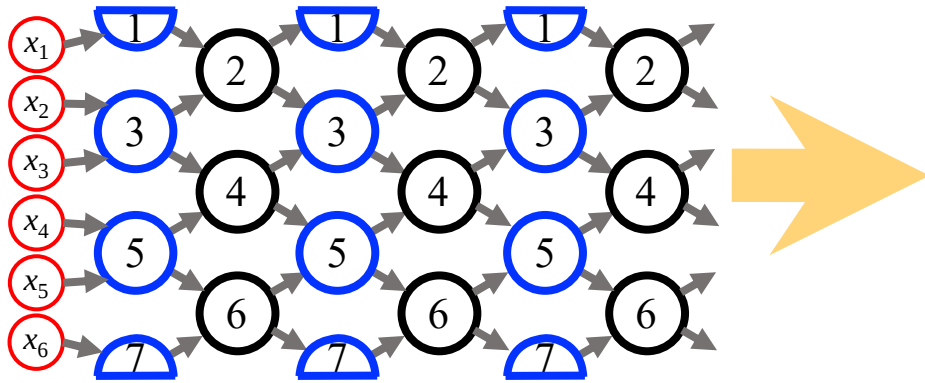
How tiles compute while growing (algorithmic self-assembly)



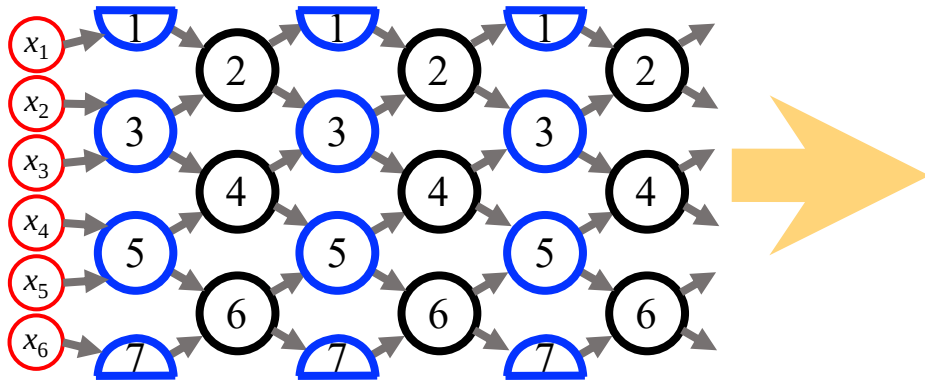
How tiles compute while growing (algorithmic self-assembly)



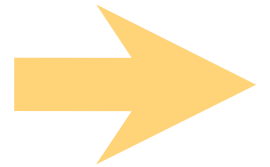
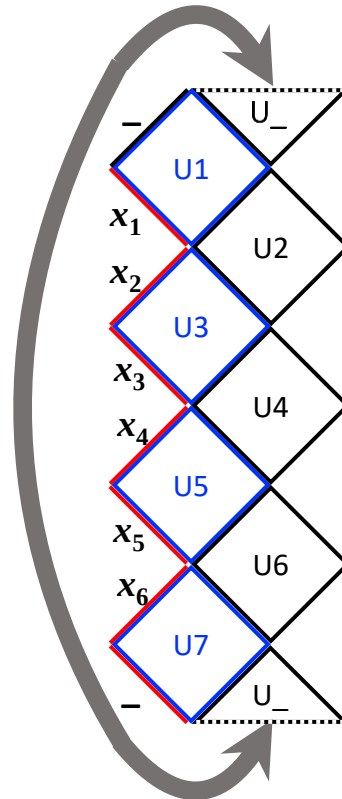
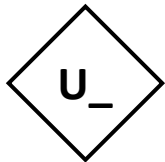
How tiles compute while growing (algorithmic self-assembly)



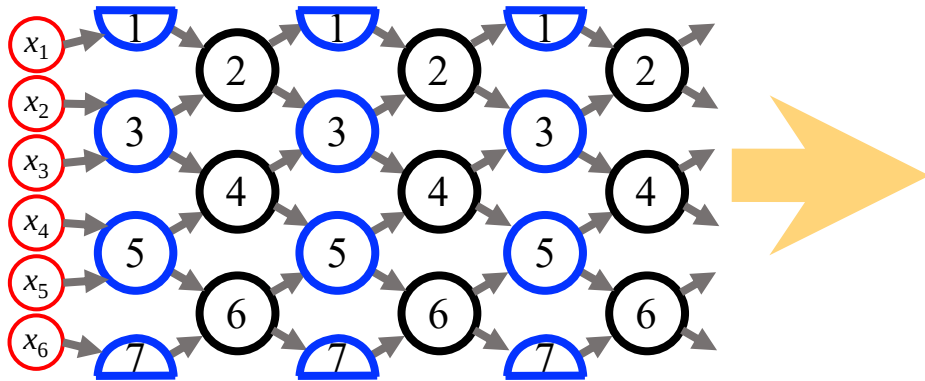
How tiles compute while growing (algorithmic self-assembly)



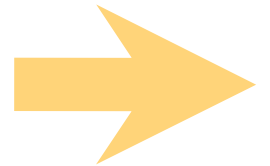
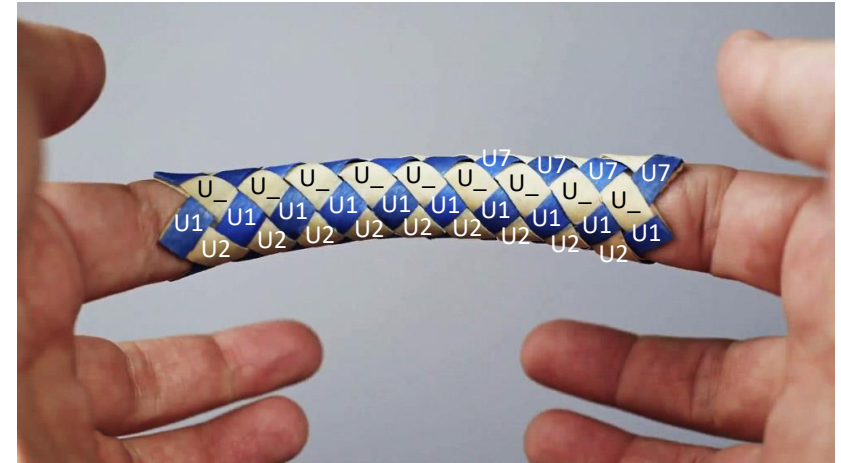
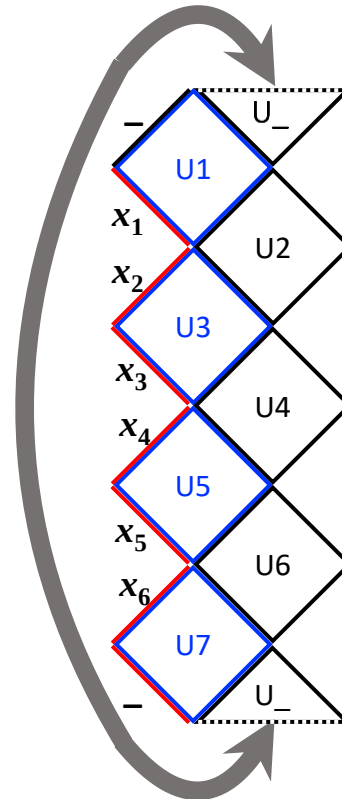
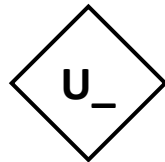
“data-free” tile wraps top
to bottom to form a tube



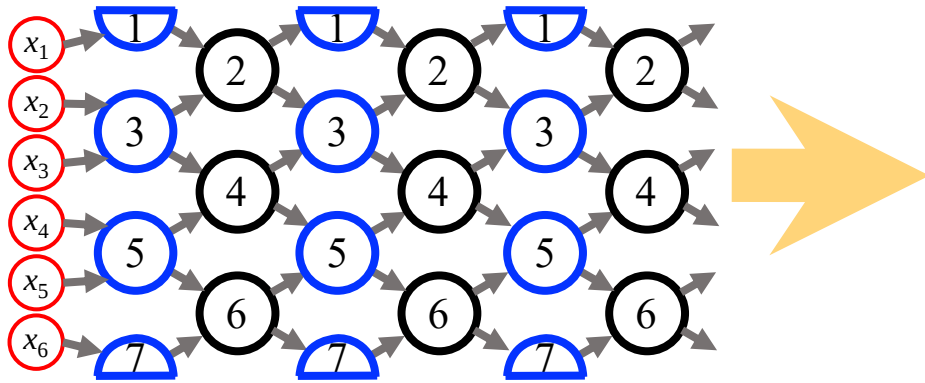
How tiles compute while growing (algorithmic self-assembly)



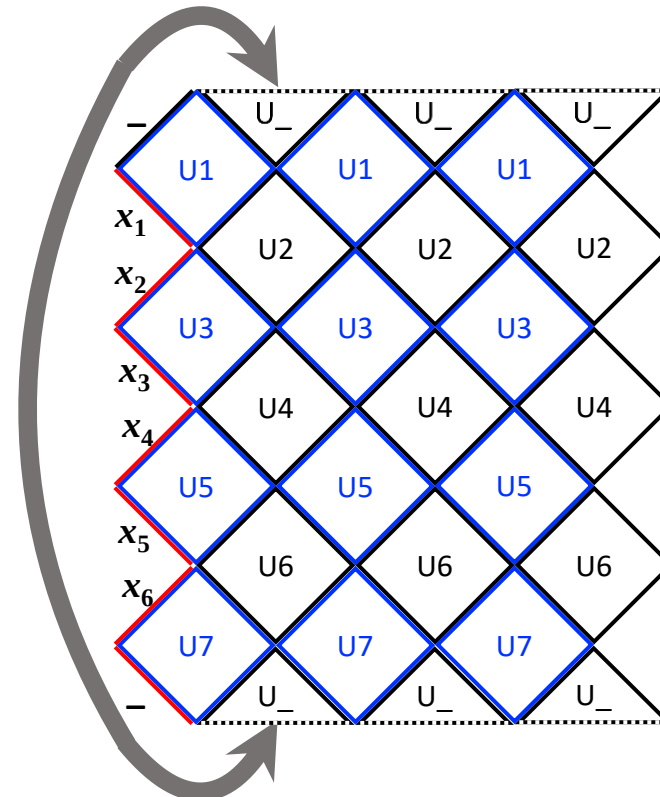
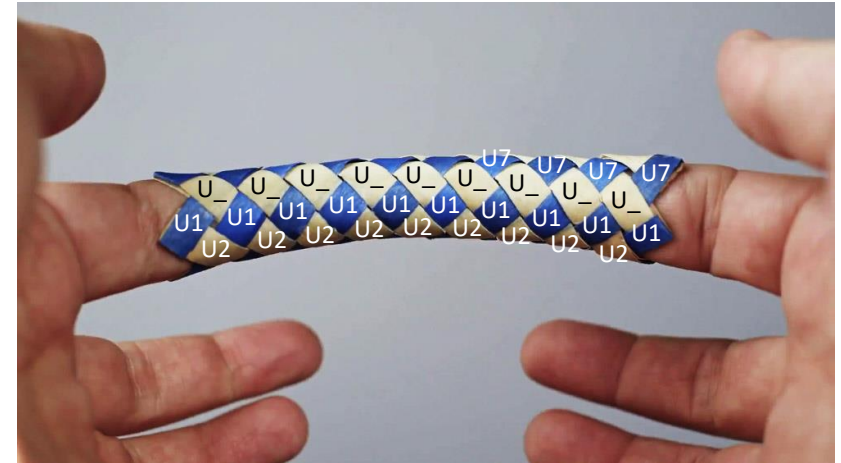
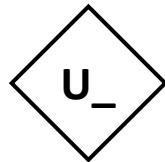
“data-free” tile wraps top
to bottom to form a tube



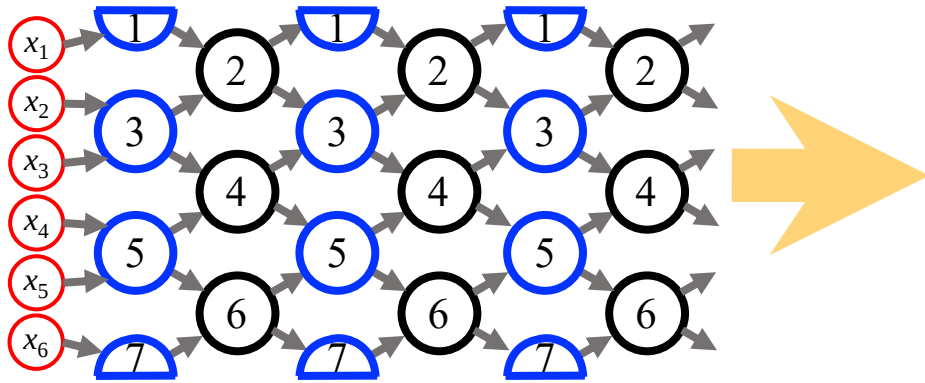
How tiles compute while growing (algorithmic self-assembly)



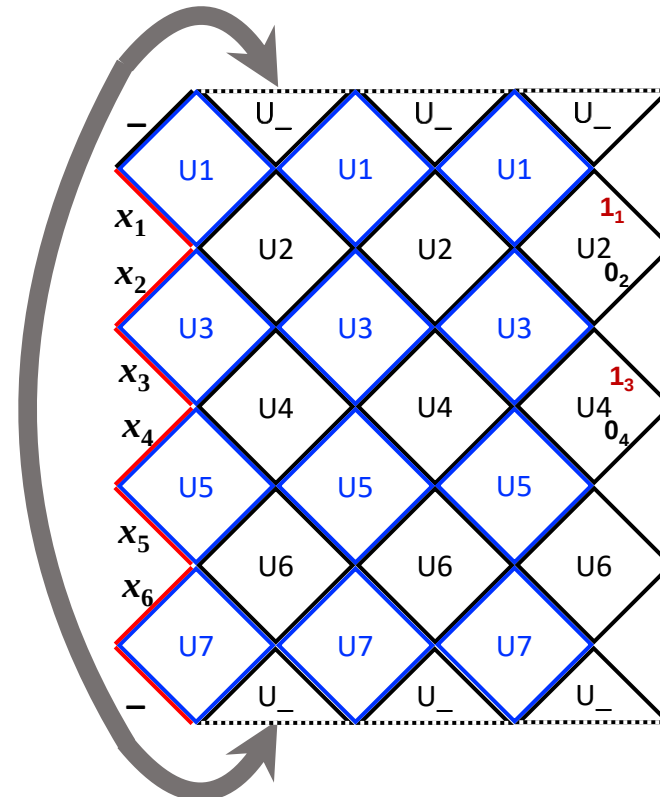
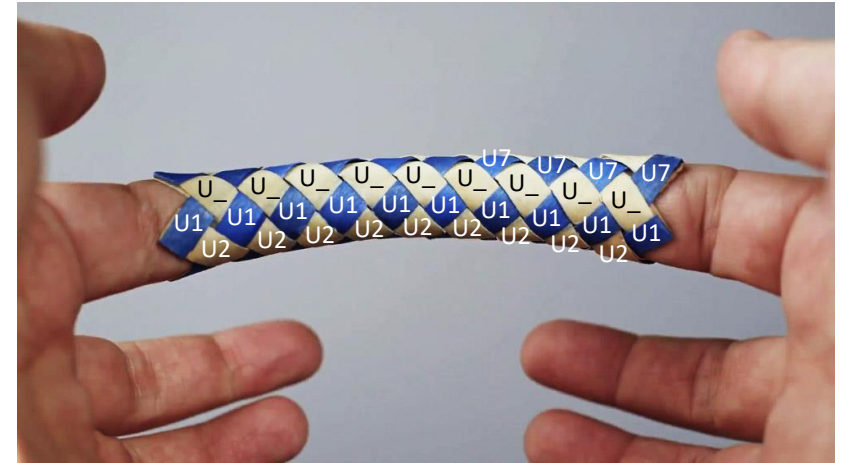
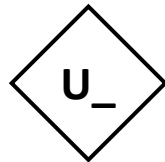
“data-free” tile wraps top
to bottom to form a tube



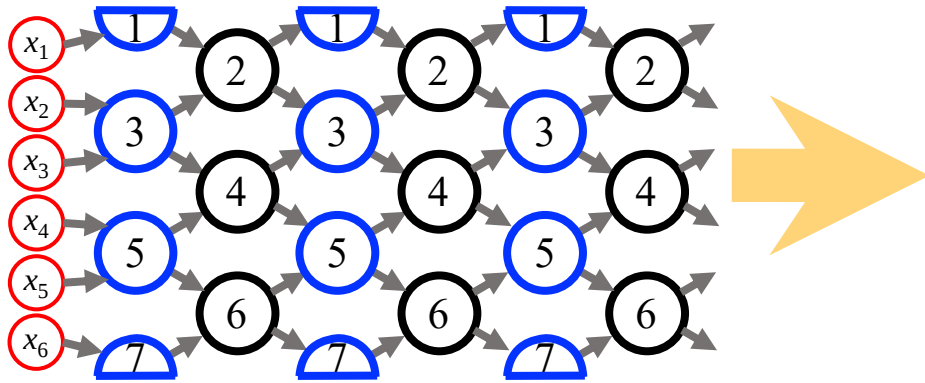
How tiles compute while growing (algorithmic self-assembly)



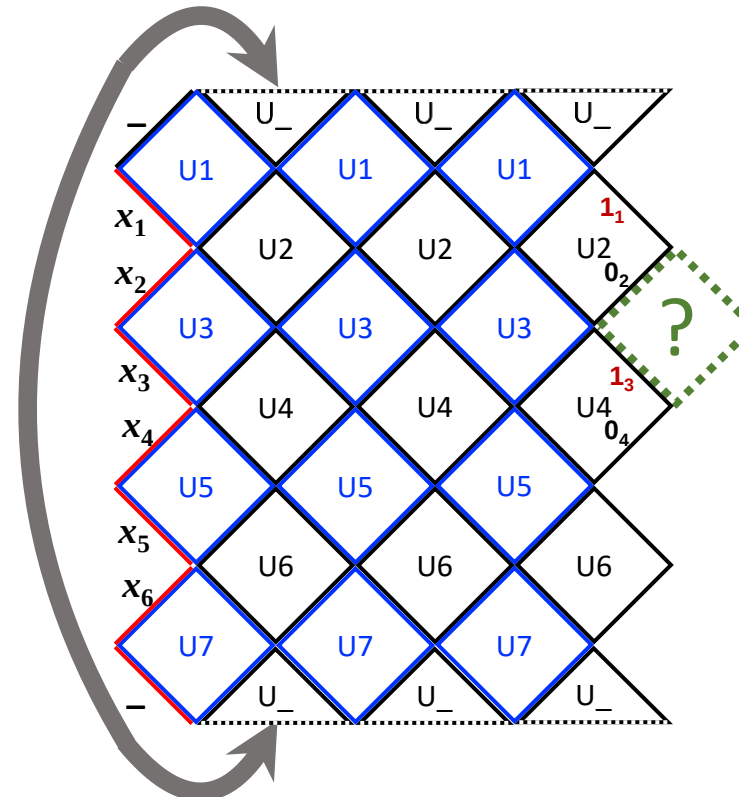
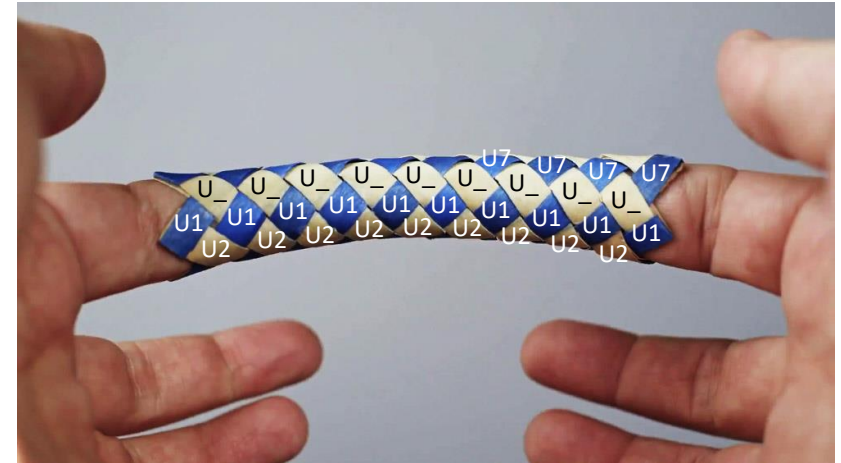
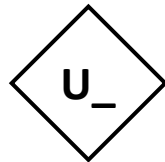
“data-free” tile wraps top
to bottom to form a tube



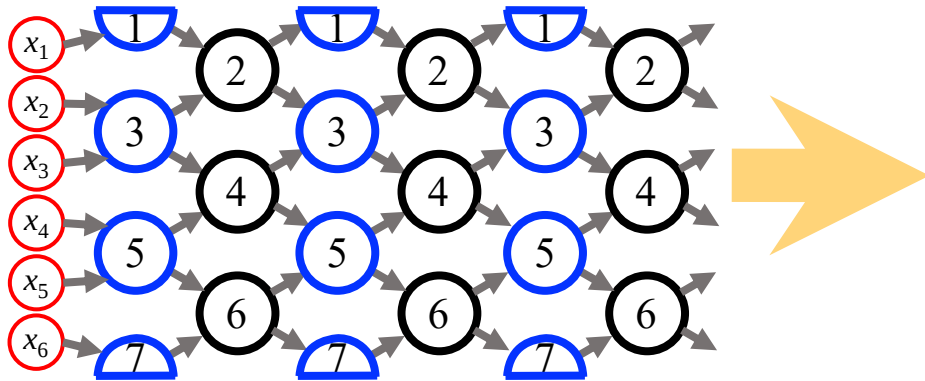
How tiles compute while growing (algorithmic self-assembly)



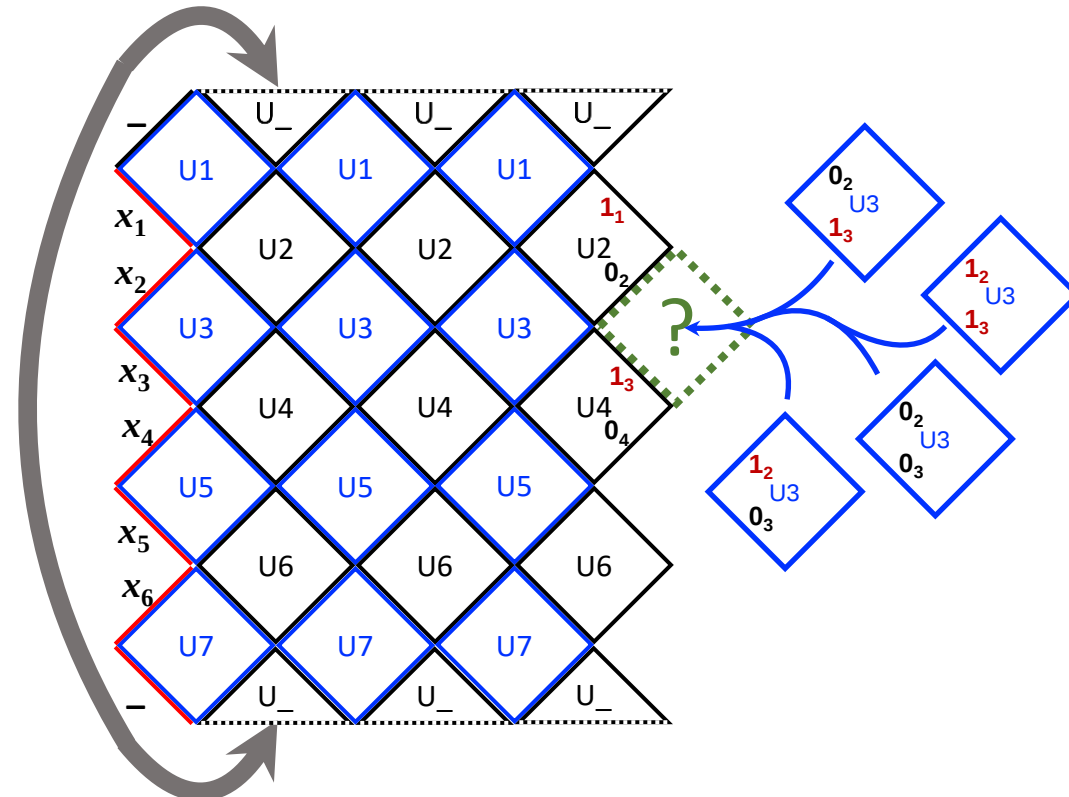
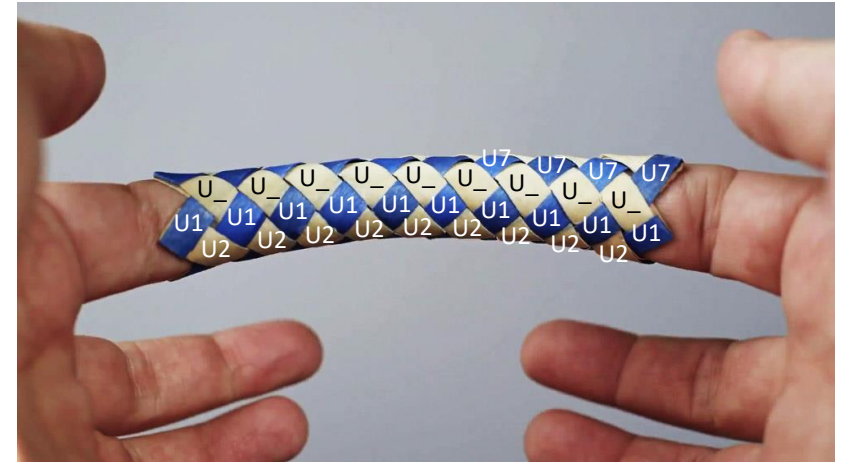
“data-free” tile wraps top
to bottom to form a tube



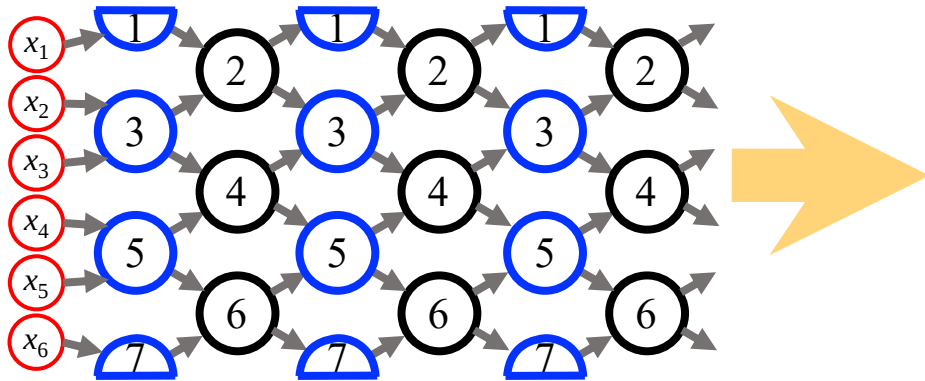
How tiles compute while growing (algorithmic self-assembly)



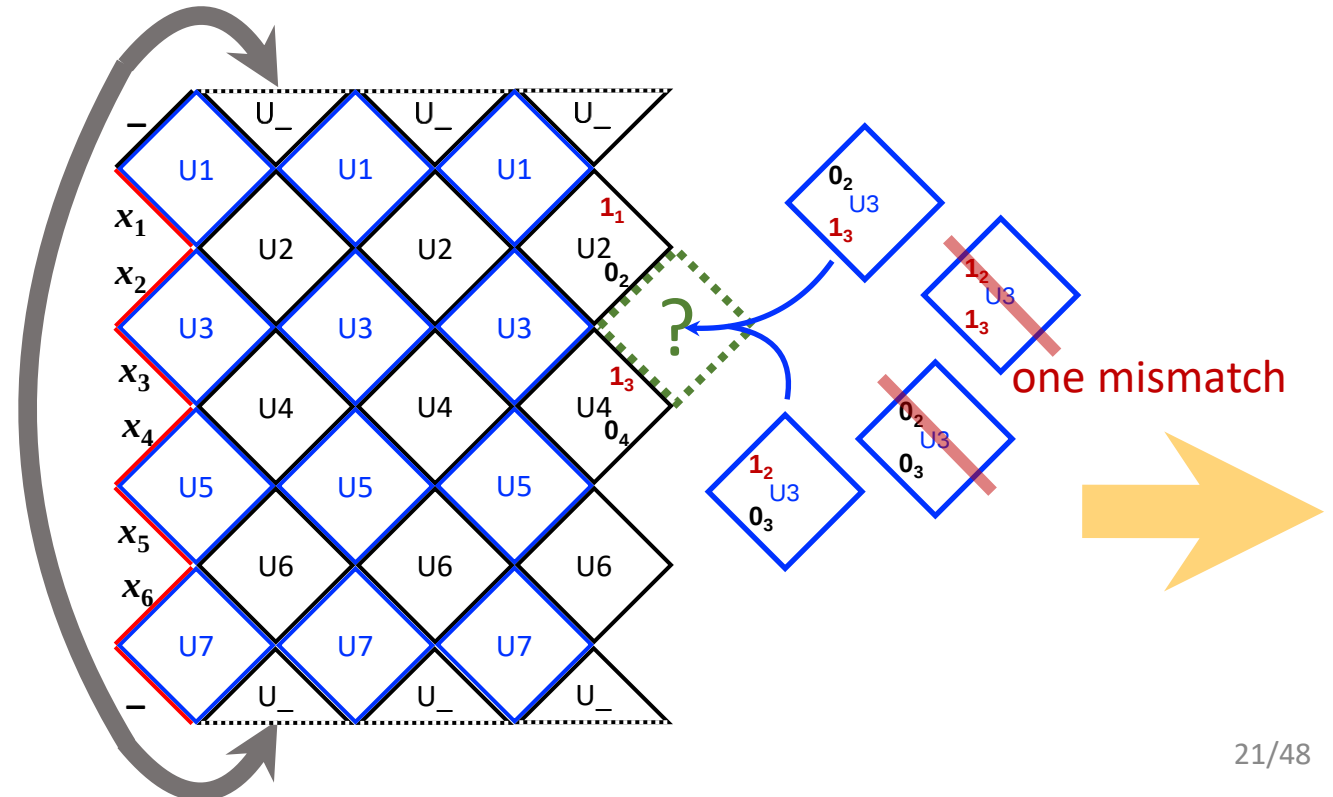
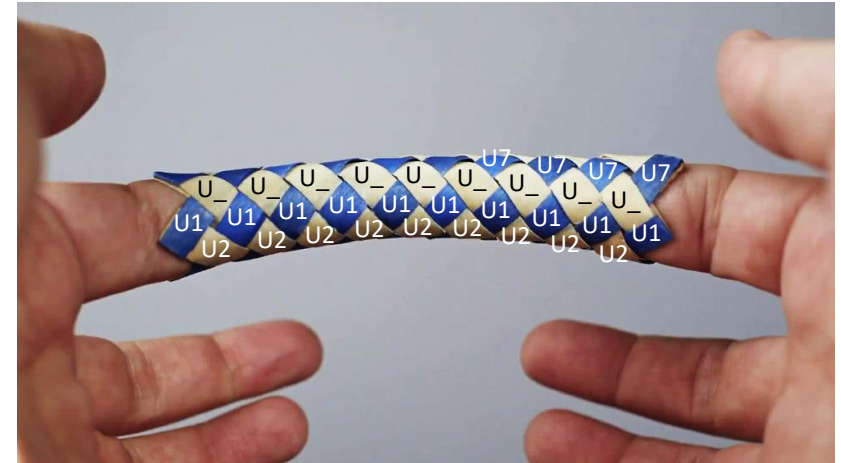
“data-free” tile wraps top
to bottom to form a tube



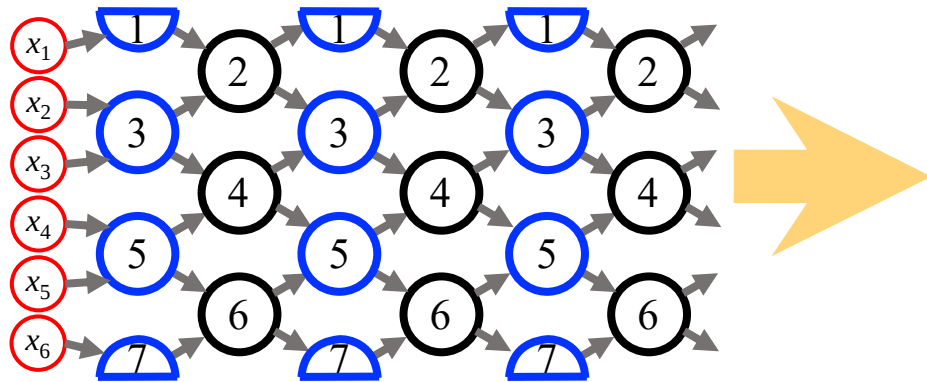
How tiles compute while growing (algorithmic self-assembly)



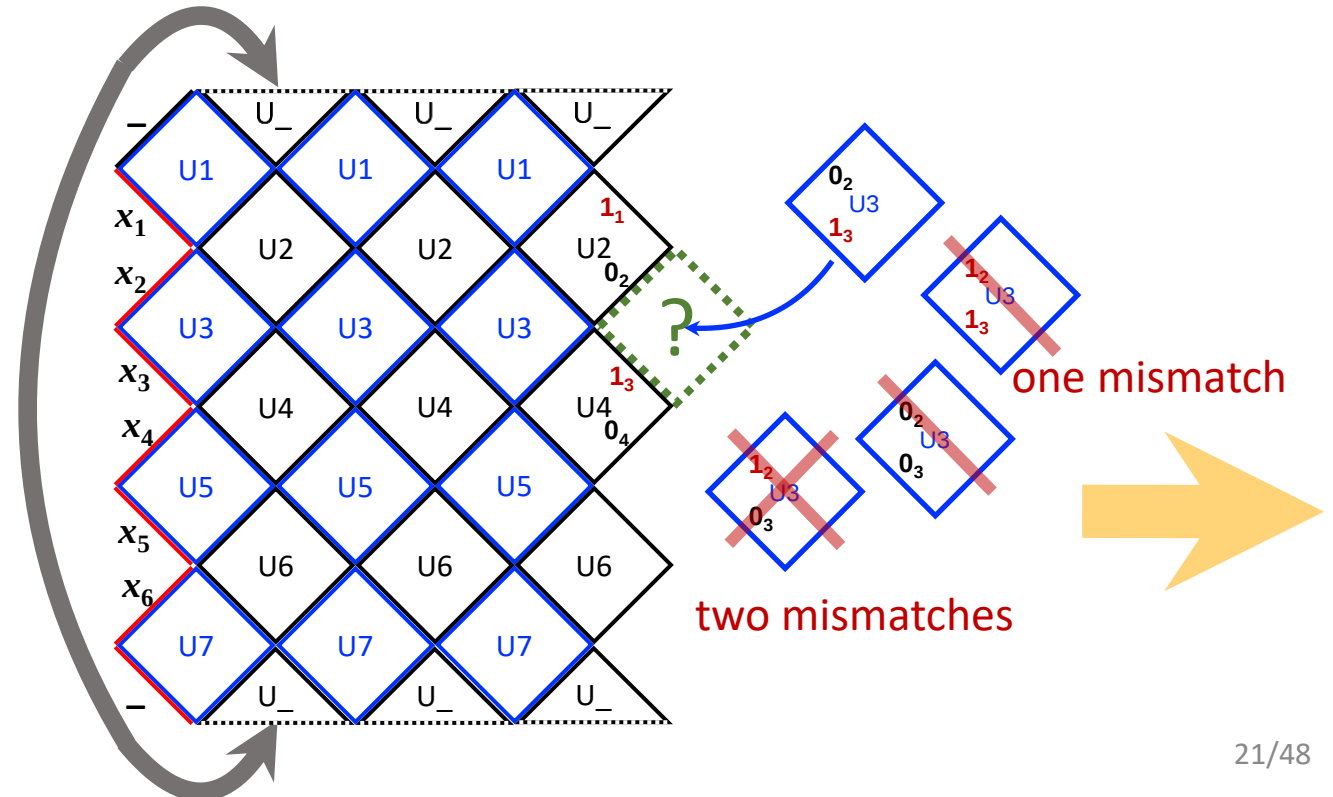
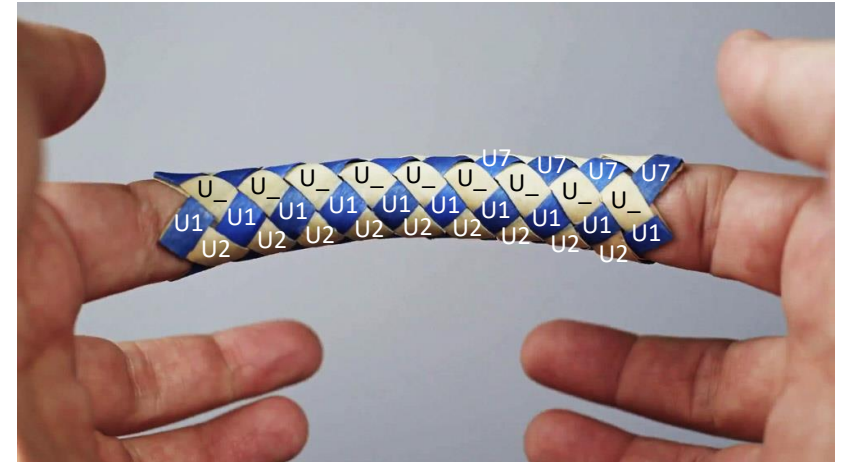
“data-free” tile wraps top
to bottom to form a tube



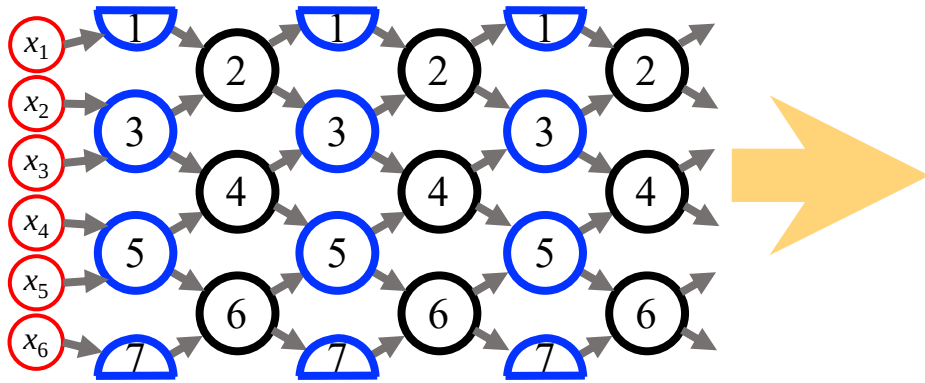
How tiles compute while growing (algorithmic self-assembly)



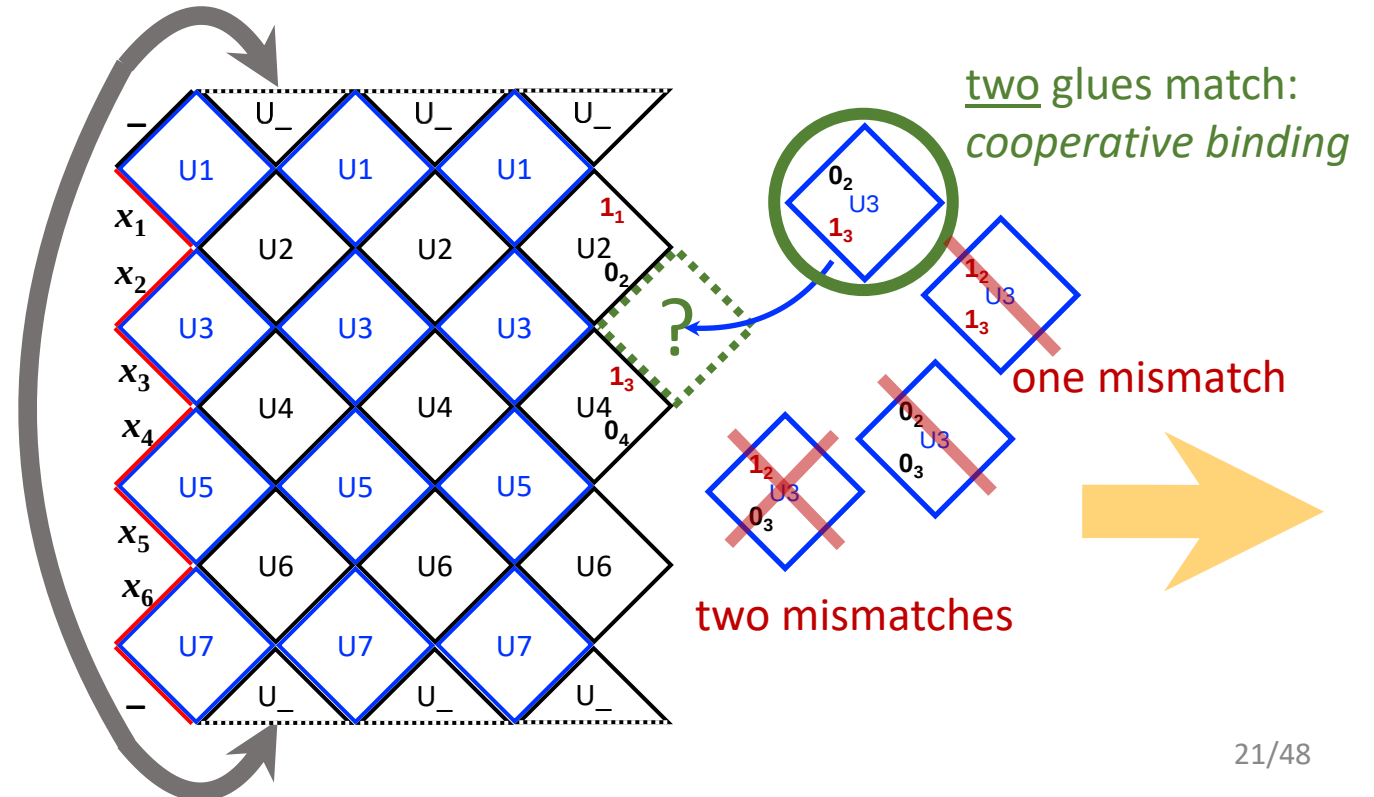
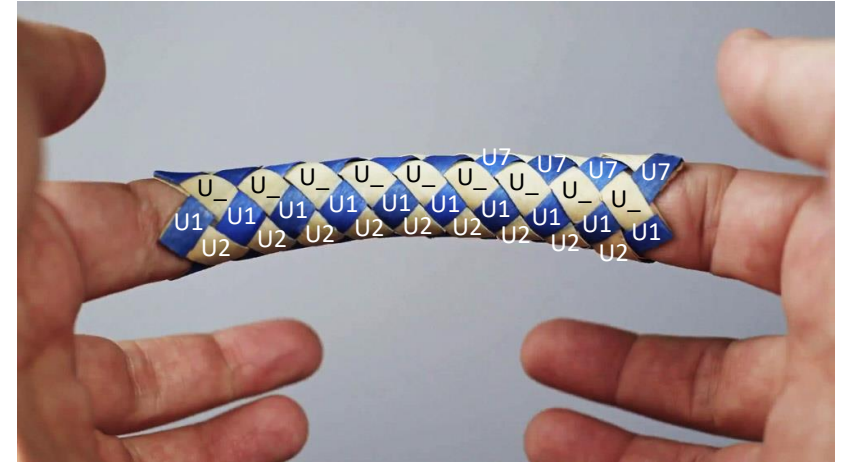
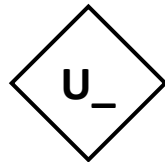
“data-free” tile wraps top
to bottom to form a tube



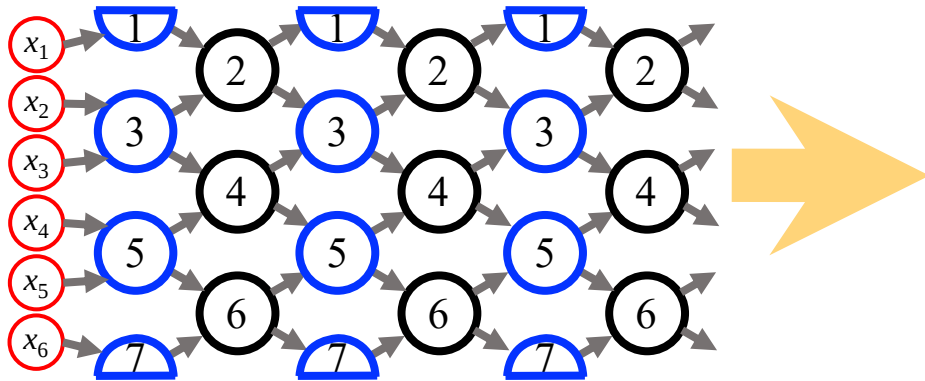
How tiles compute while growing (algorithmic self-assembly)



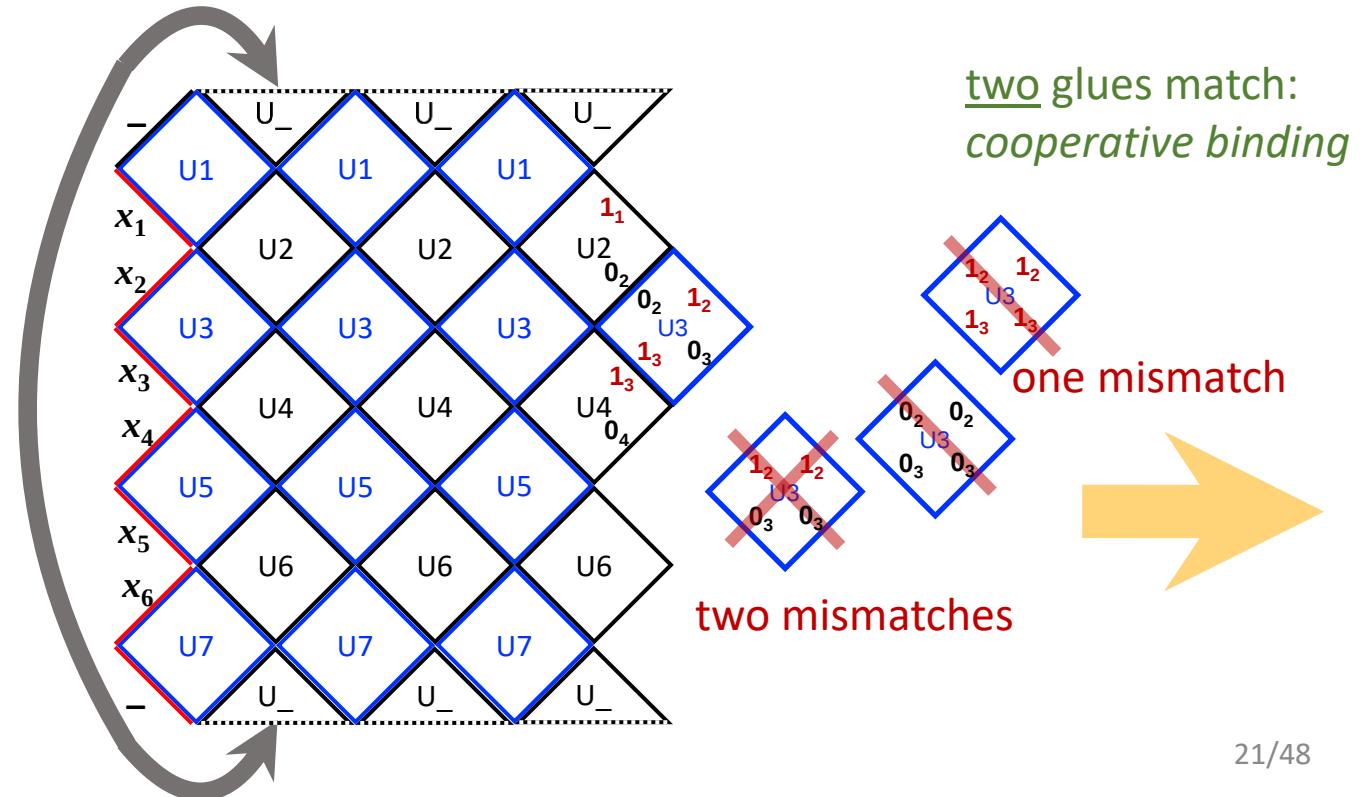
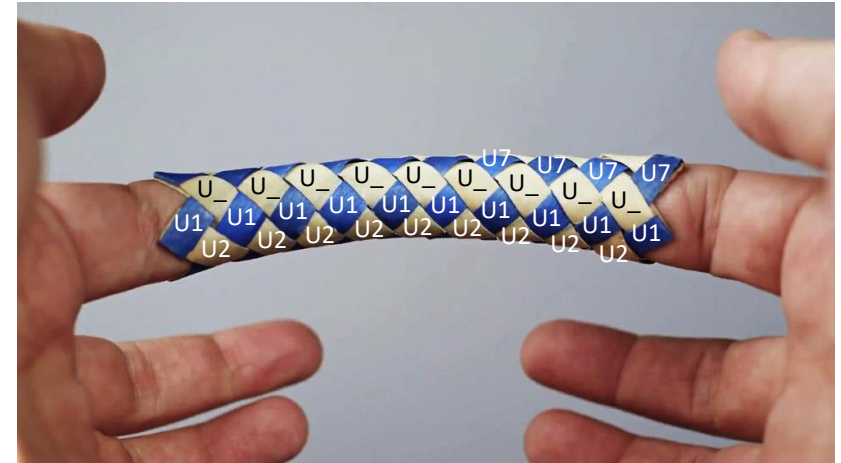
“data-free” tile wraps top
to bottom to form a tube



How tiles compute while growing (algorithmic self-assembly)



“data-free” tile wraps top
to bottom to form a tube



Hierarchy of abstractions

Bits: Boolean circuits compute

Tiles: Tile growth implements circuits

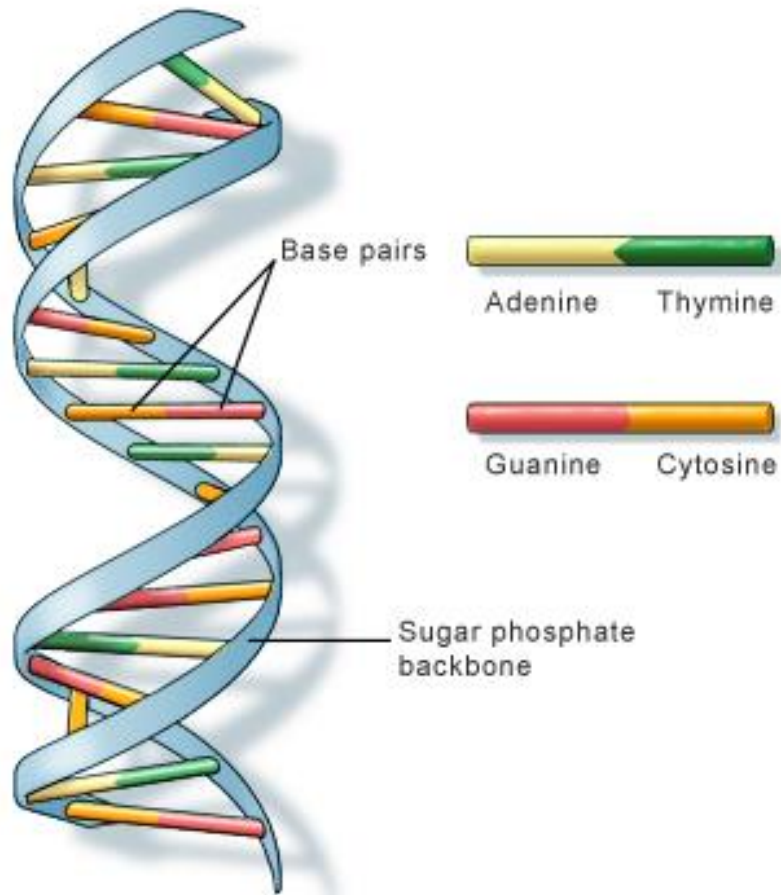
→ DNA: DNA strands implement tiles



Structural DNA nanotechnology

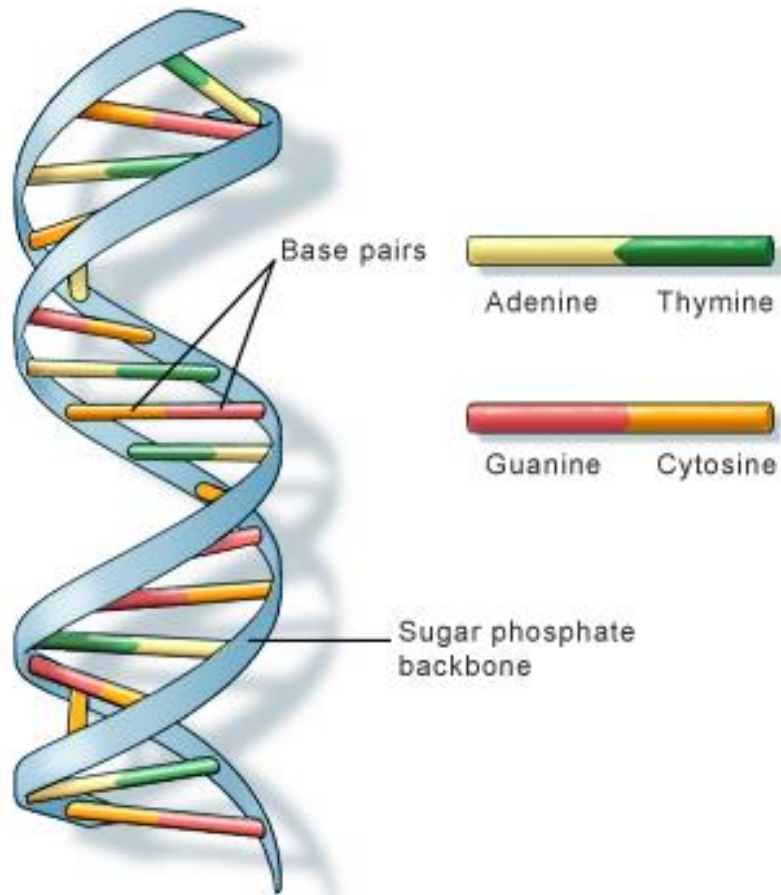
a.k.a. DNA carpentry

DNA as a building material



U.S. National Library of Medicine

DNA as a building material



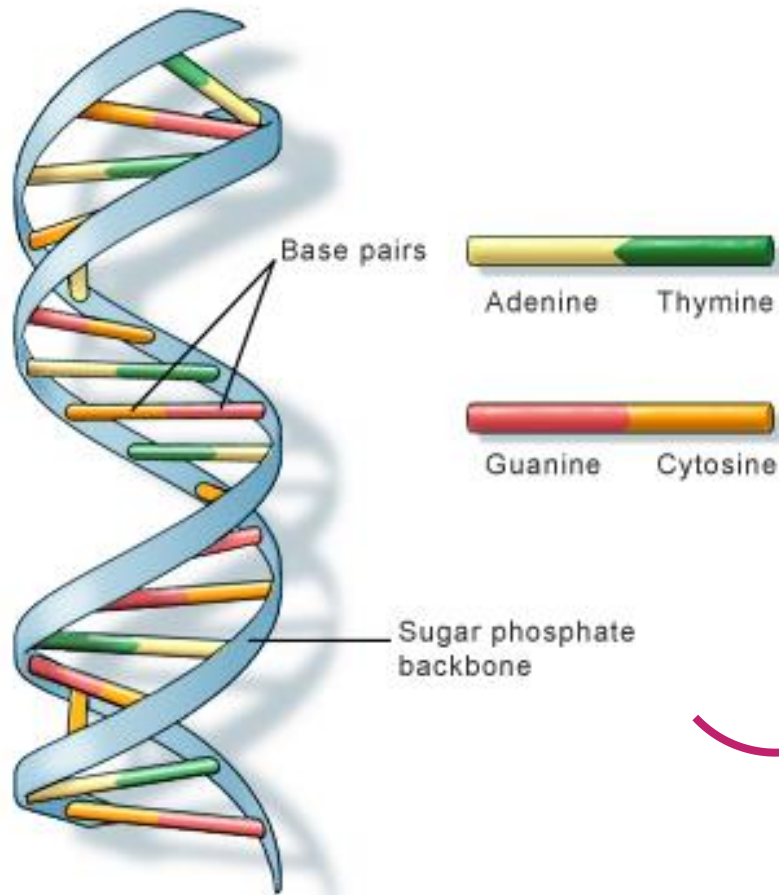
U.S. National Library of Medicine



=

TCGGAAATAAAATCGGAC
AGCCTTTATTTTAGCCTG

DNA as a building material



U.S. National Library of Medicine



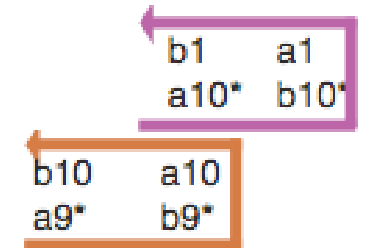
=



DNA strands can bind even if only part of strands are complementary:



=



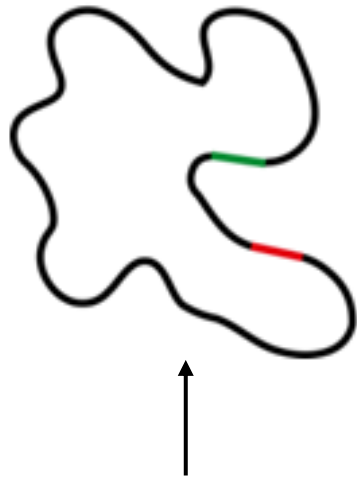
DNA origami

Paul Rothemund

Folding DNA to create nanoscale shapes and patterns

Nature 2006

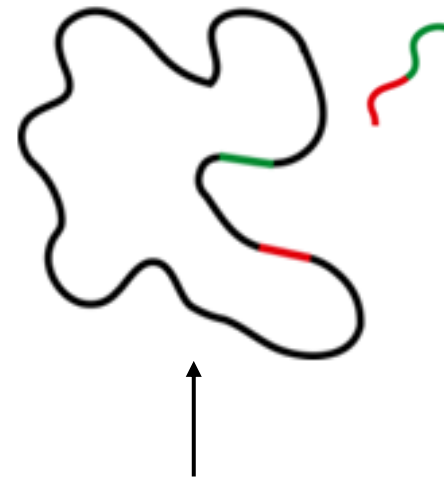
DNA origami



↑
scaffold DNA strand
(M13mp18 bacteriophage virus)

Paul Rothemund
Folding DNA to create nanoscale shapes and patterns
Nature 2006

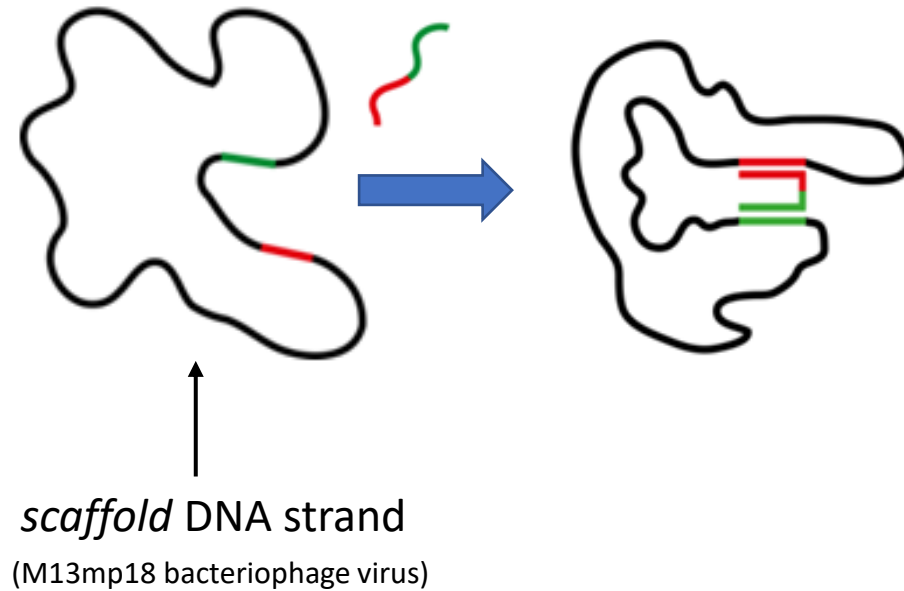
DNA origami



scaffold DNA strand
(M13mp18 bacteriophage virus)

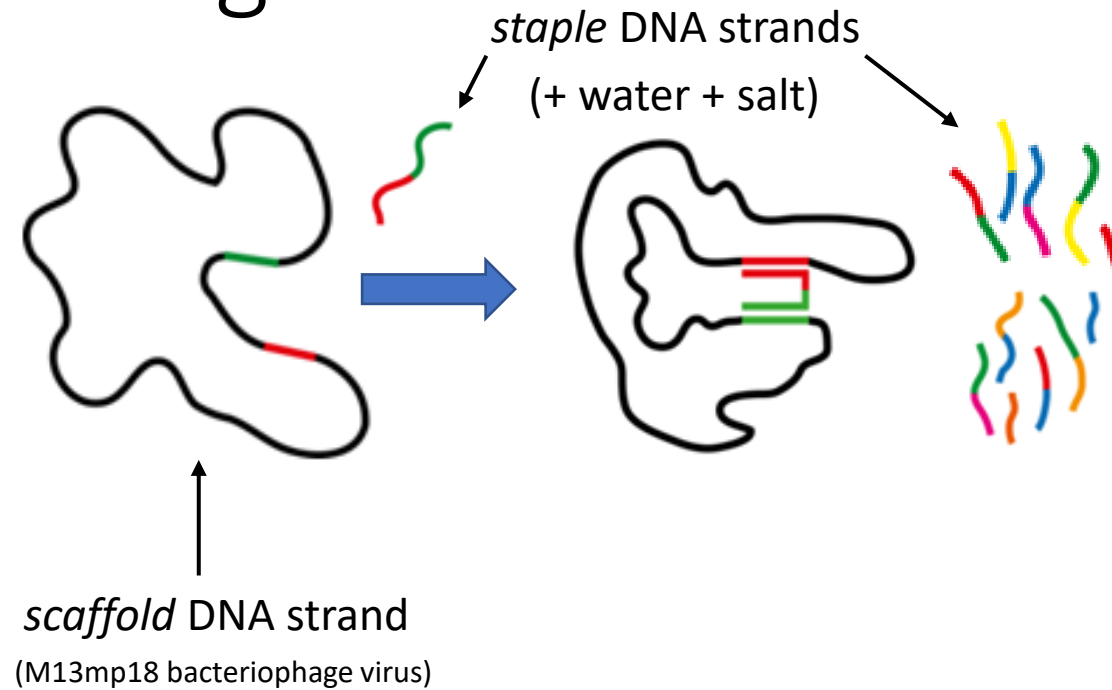
Paul Rothemund
Folding DNA to create nanoscale shapes and patterns
Nature 2006

DNA origami



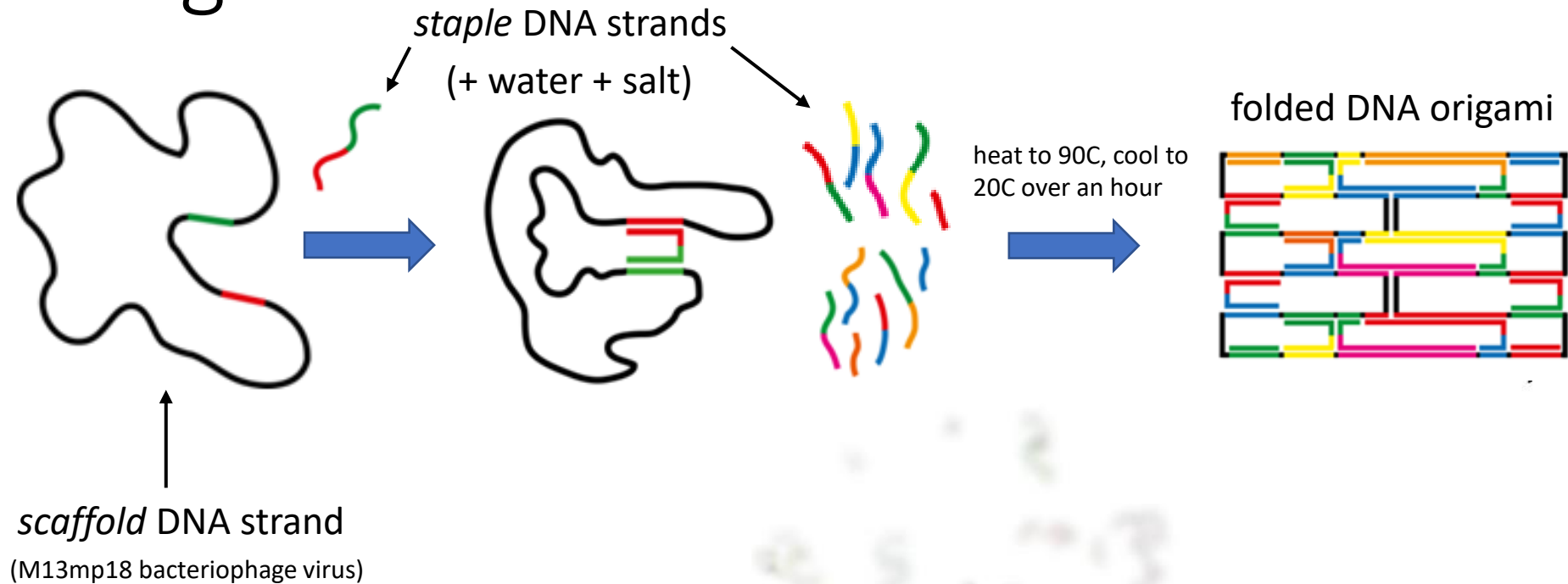
Paul Rothemund
Folding DNA to create nanoscale shapes and patterns
Nature 2006

DNA origami



Paul Rothemund
Folding DNA to create nanoscale shapes and patterns
Nature 2006

DNA origami



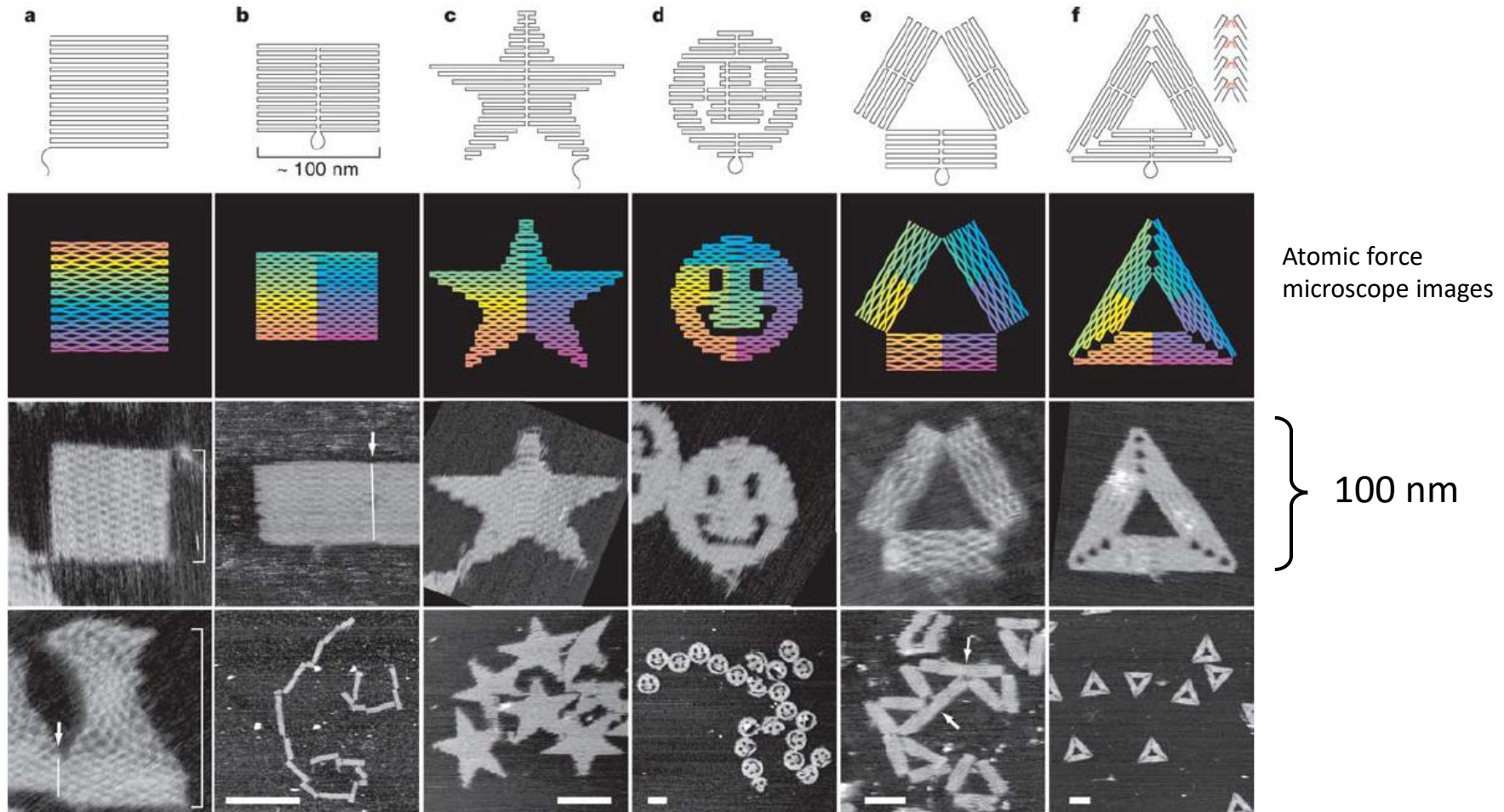
Paul Rothemund
Folding DNA to create nanoscale shapes and patterns
Nature 2006

DNA origami

Paul Rothemund

Folding DNA to create nanoscale shapes and patterns

Nature 2006



DNA nanotechnology applications

nonbiological:

- nanoscale resolution surface placement
- X-ray crystallization scaffolding
- molecular motors
- super-resolution imaging
- molecular circuits

biological:

- smart drugs
- mRNA detection
- cell surface marker detection
- genetically encoded structures

DNA nanotechnology applications

nonbiological:

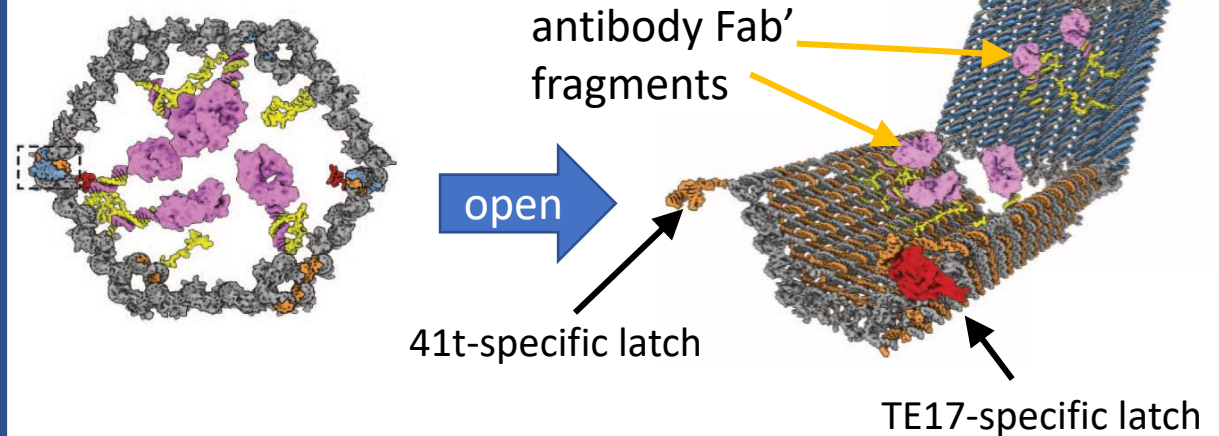
- nanoscale resolution surface placement
- X-ray crystallization scaffolding
- molecular motors
- super-resolution imaging
- molecular circuits

biological:

- smart drugs
- mRNA detection
- cell surface marker detection
- genetically encoded structures

example

DNA origami opens to deliver antibody only in presence of two protein antigens

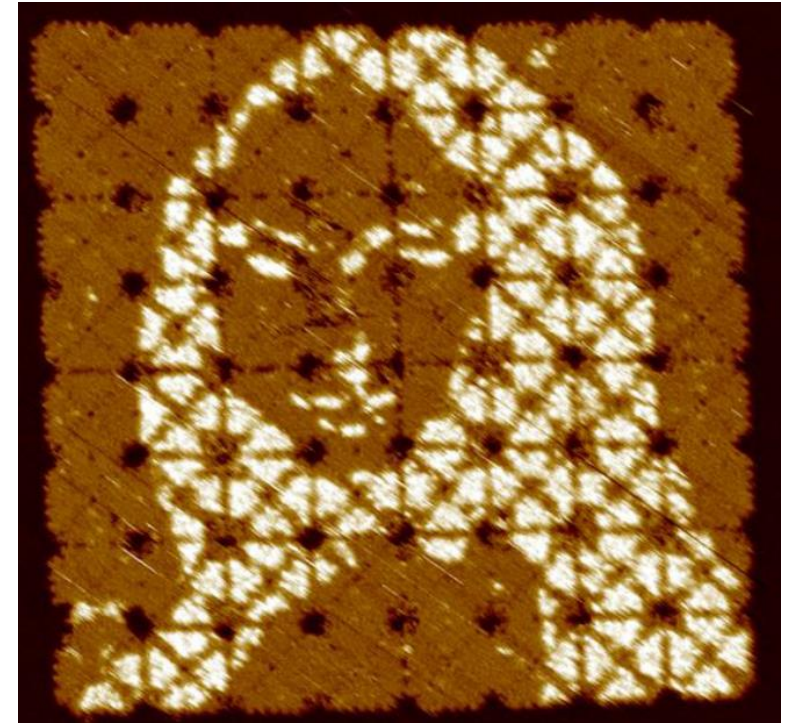
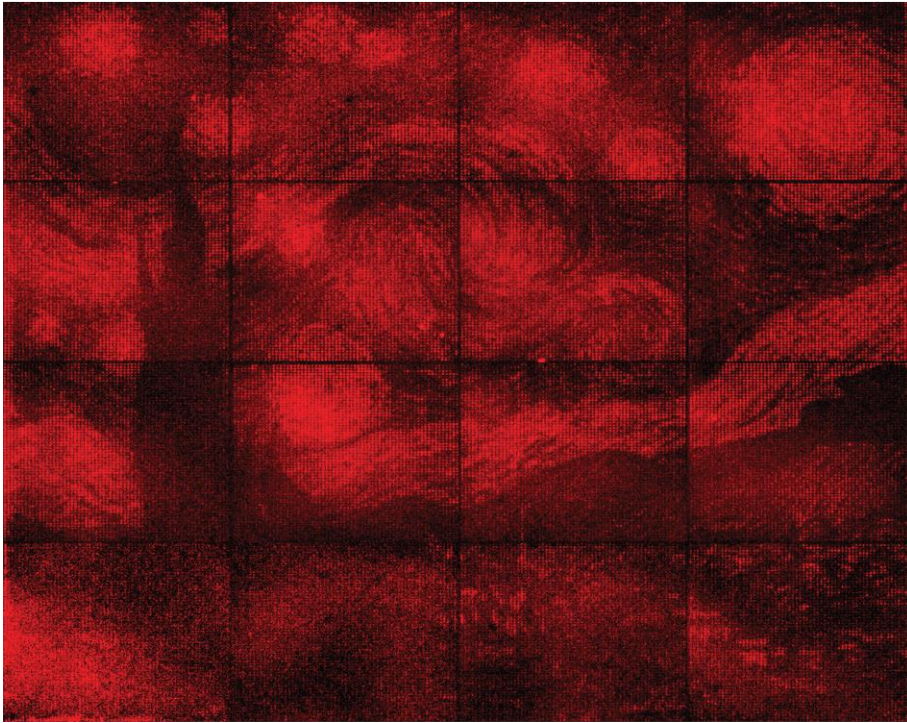


Shawn Douglas, Ido Bachelet, George Church. *A logic-gated nanorobot for targeted transport of molecular payloads*, Science 2012

DNA nanotechnology applications

nonbiological:

- **art**

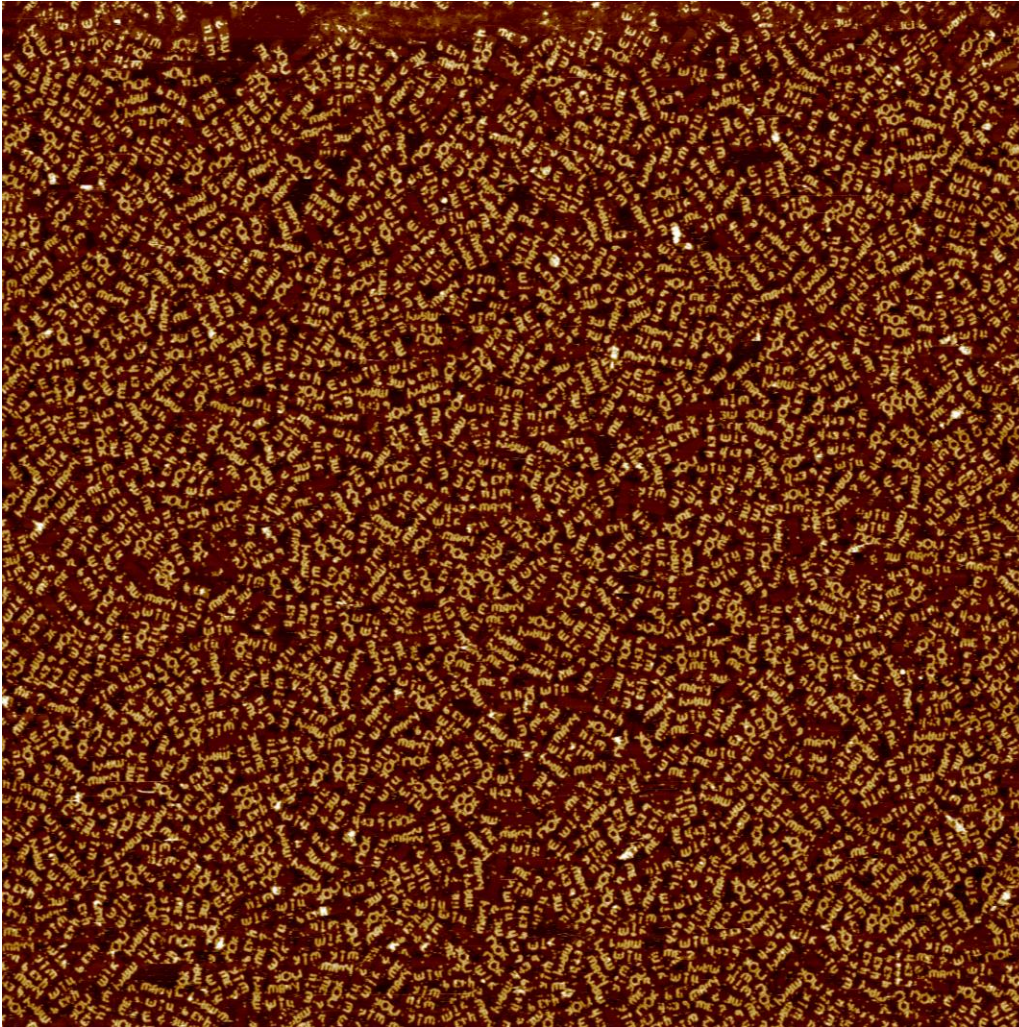


Ashwin Gopinath, Evan Miyazono, Andrei Faraon, Paul Rothemund. *Engineering and mapping nanocavity emission via precision placement of DNA origami*, Nature 2016

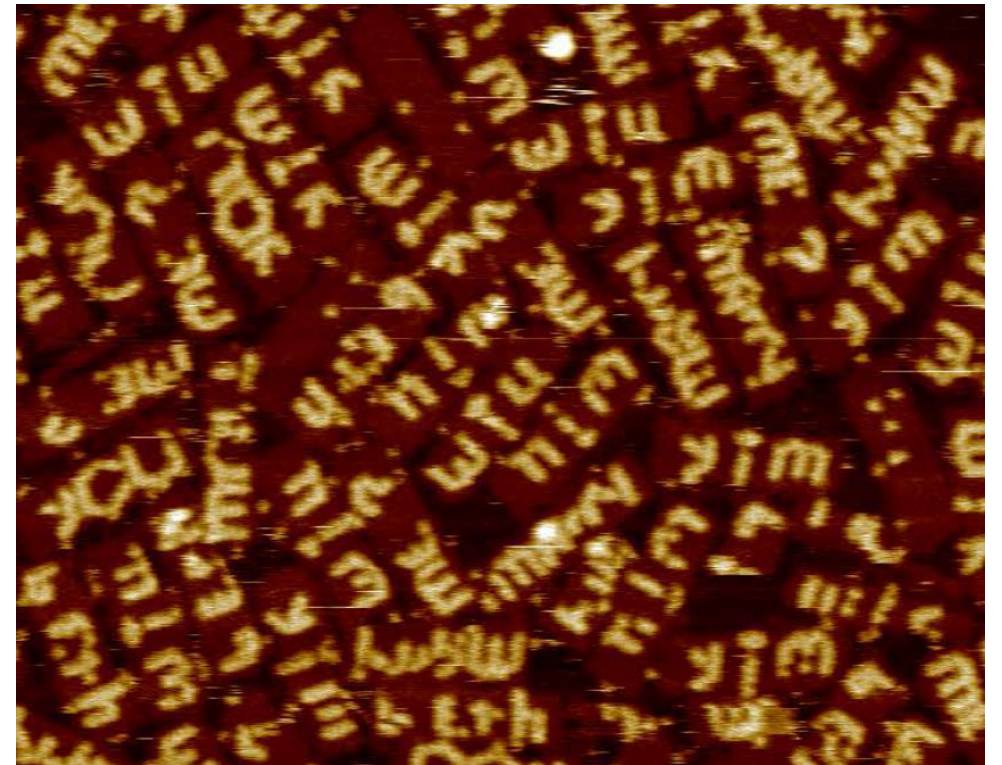
Grigory Tikhomirov, Philip Petersen, and Lulu Qian. *Fractal assembly of micrometre-scale DNA origami arrays with arbitrary patterns*. Nature 2017.

Other applications of DNA nanotechnology

4 μm wide scan



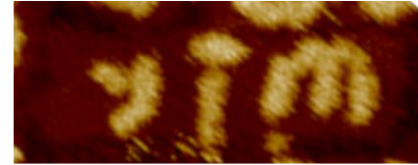
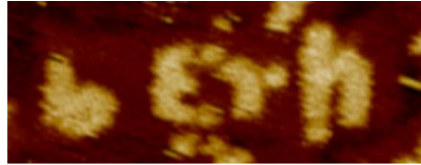
zoom in



A little proposal

100 nm

Beth Yim

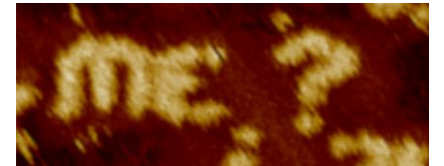
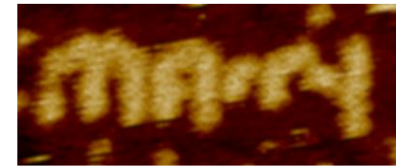
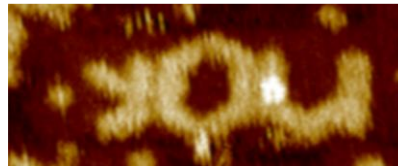
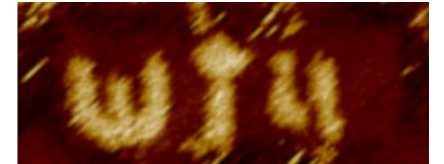
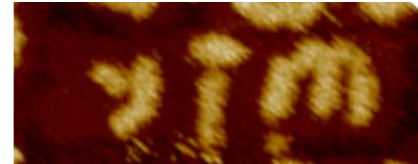
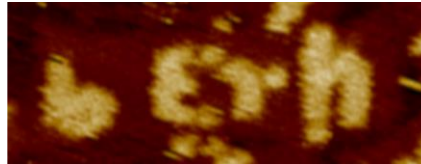


A little proposal

100 nm



Beth Yim

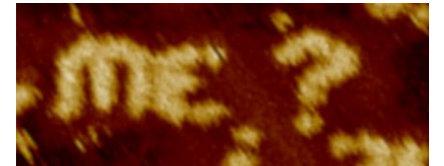
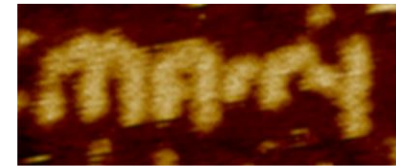
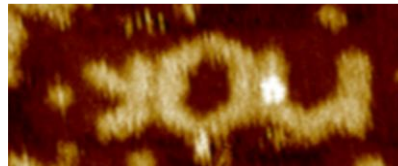
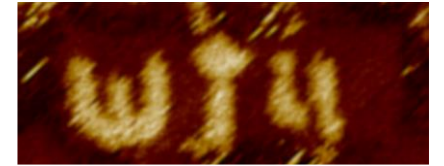
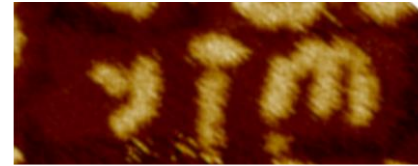
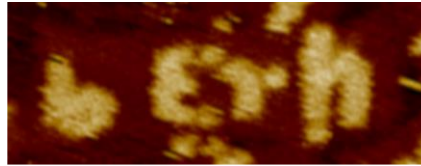


A little proposal

100 nm



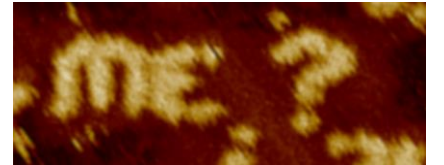
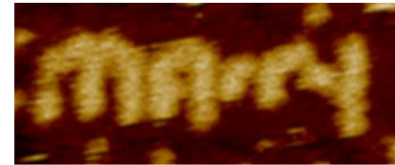
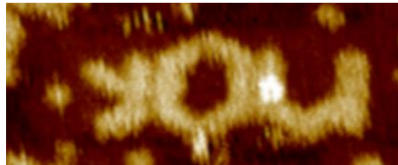
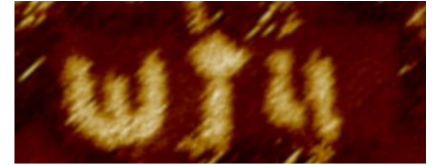
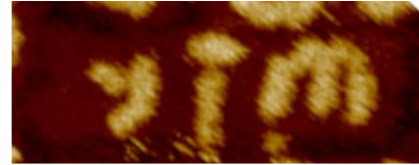
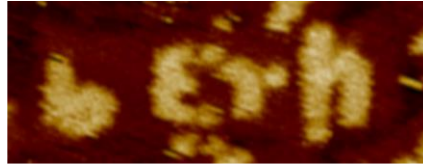
Beth Yim



A little proposal

100 nm

Beth Yim

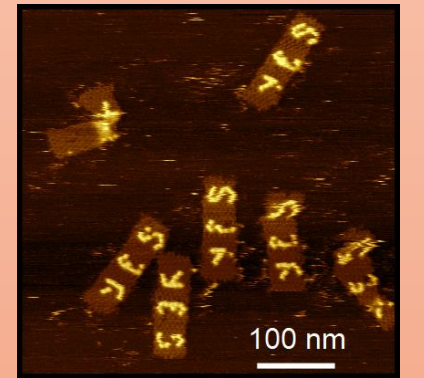
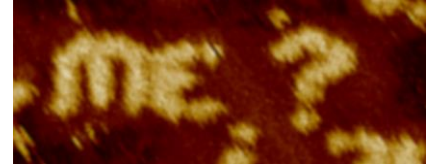
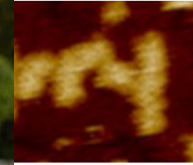
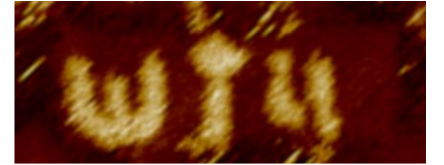


A little proposal and a little reply

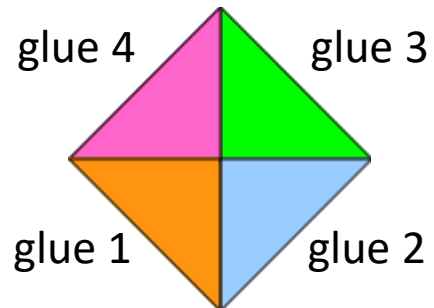
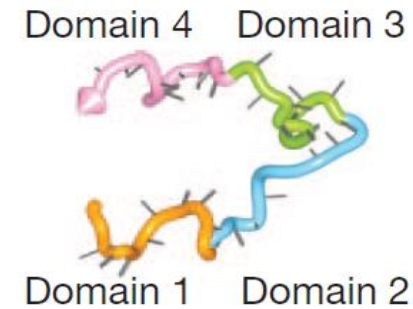
Beth Yim



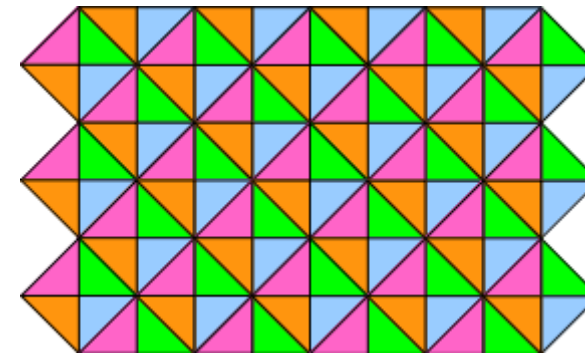
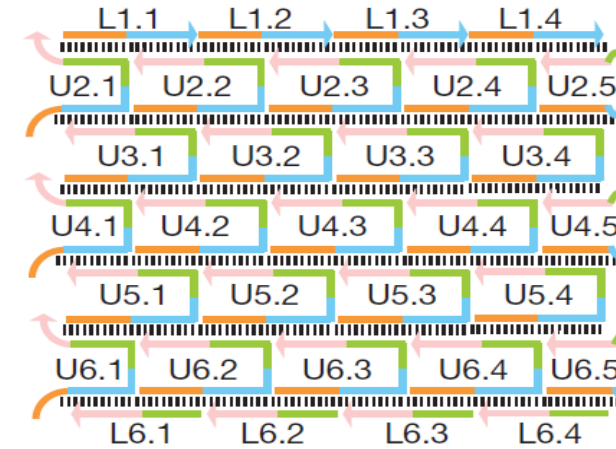
100 nm



DNA single-stranded tiles

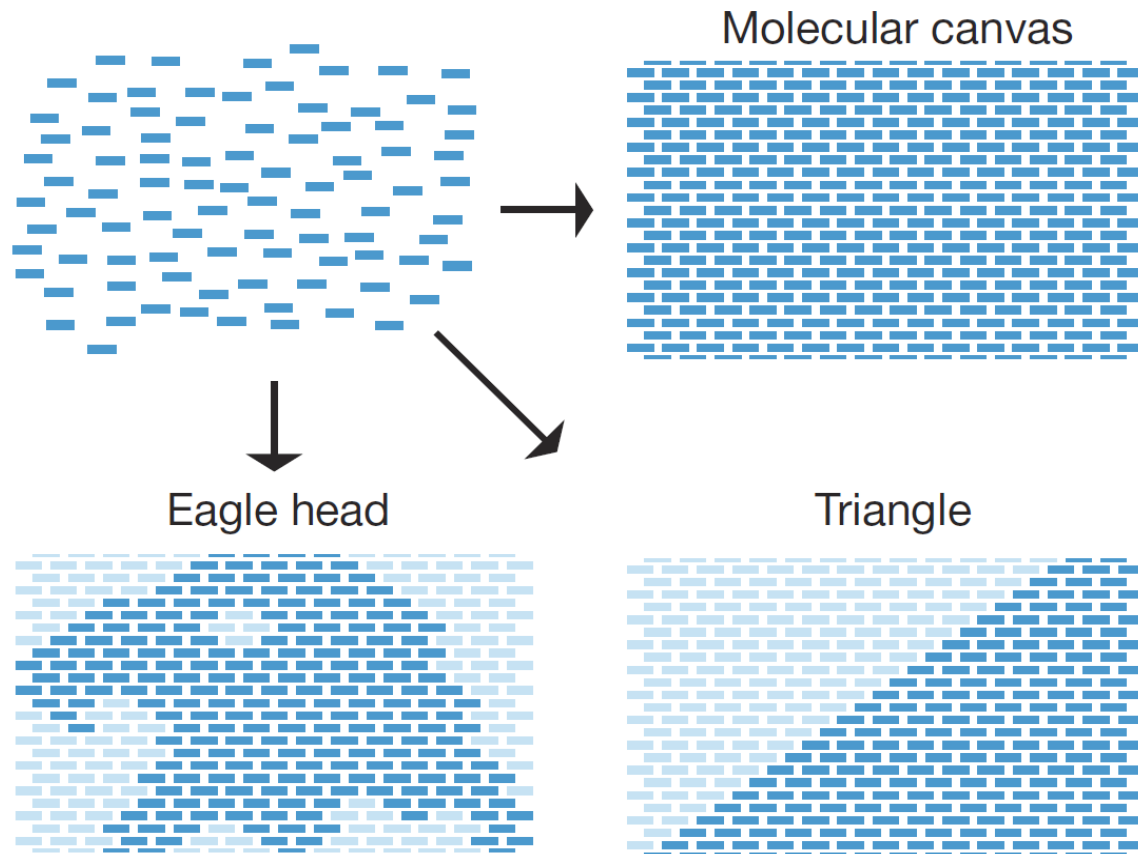


assembly



Yin, Hariadi, Sahu, Choi, Park, LaBean, and Reif.
Programming DNA tube circumferences.
Science 2008

Single-stranded tiles for making any shape



Bryan Wei, Mingjie Dai, and Peng Yin.
Complex shapes self-assembled from single-stranded DNA tiles.
Nature 2012.

Uniquely addressed self-assembly versus algorithmic

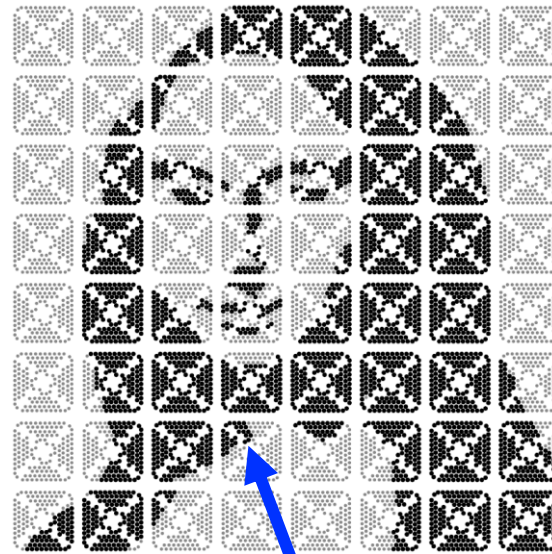
Unique addressing: each DNA “monomer” appears **exactly once** in final structure.

single DNA origami



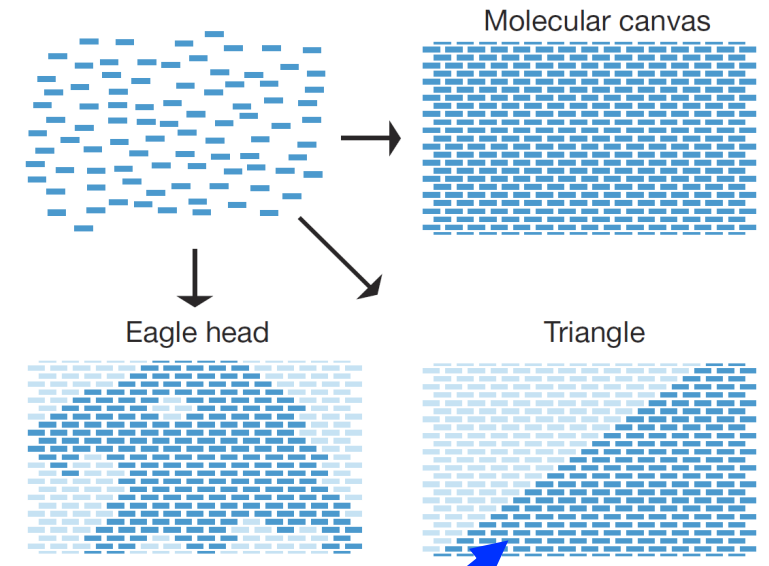
staple strand for position (4,2)

array of many DNA origamis



origami for position (4,2)

uniquely-addressed tiles



tile for position (4,2)

Uniquely addressed self-assembly versus algorithmic

Unique addressing: each DNA “monomer” appears **exactly once** in final structure.

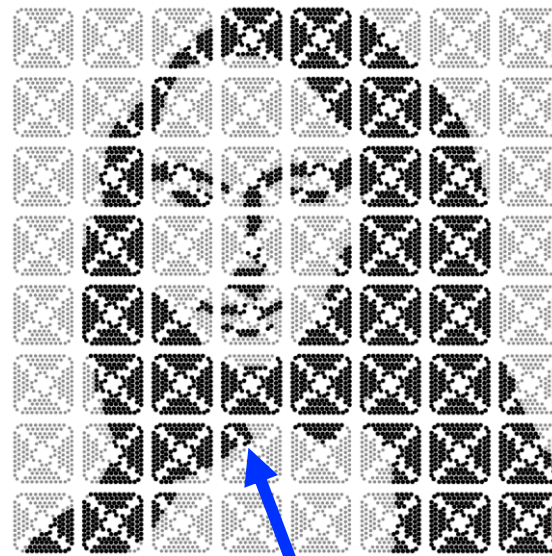
Algorithmic: DNA tiles are **reused** throughout the structure.

single DNA origami



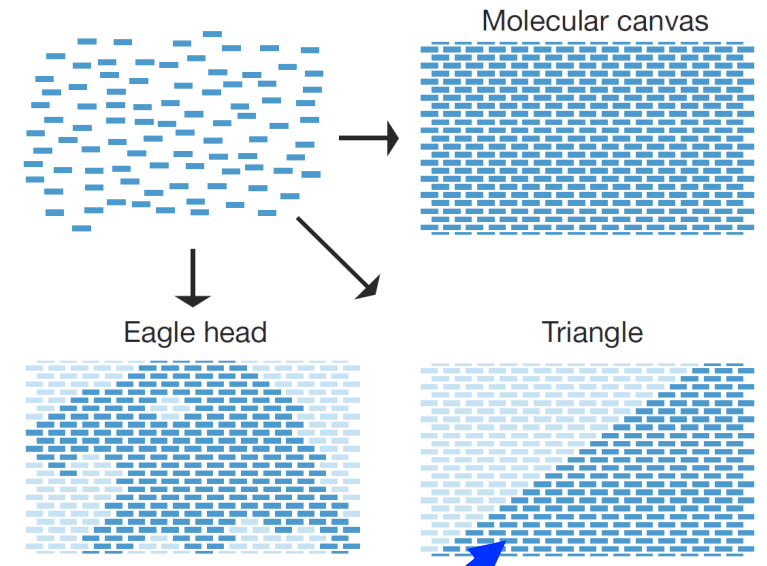
staple strand for position (4,2)

array of many DNA origamis



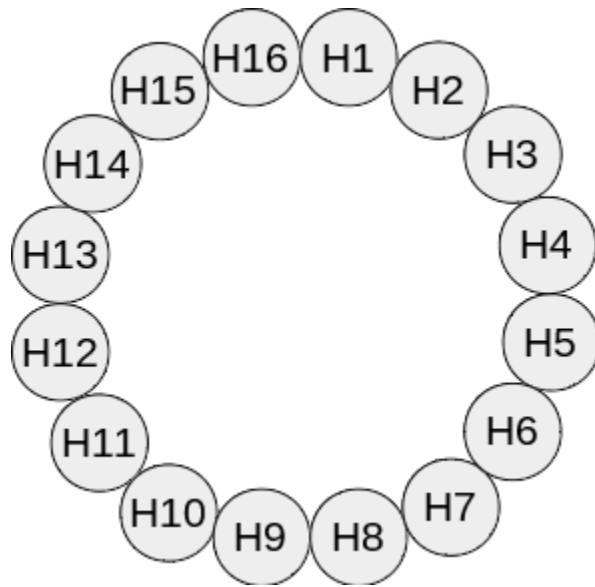
origami for position (4,2)

uniquely-addressed tiles

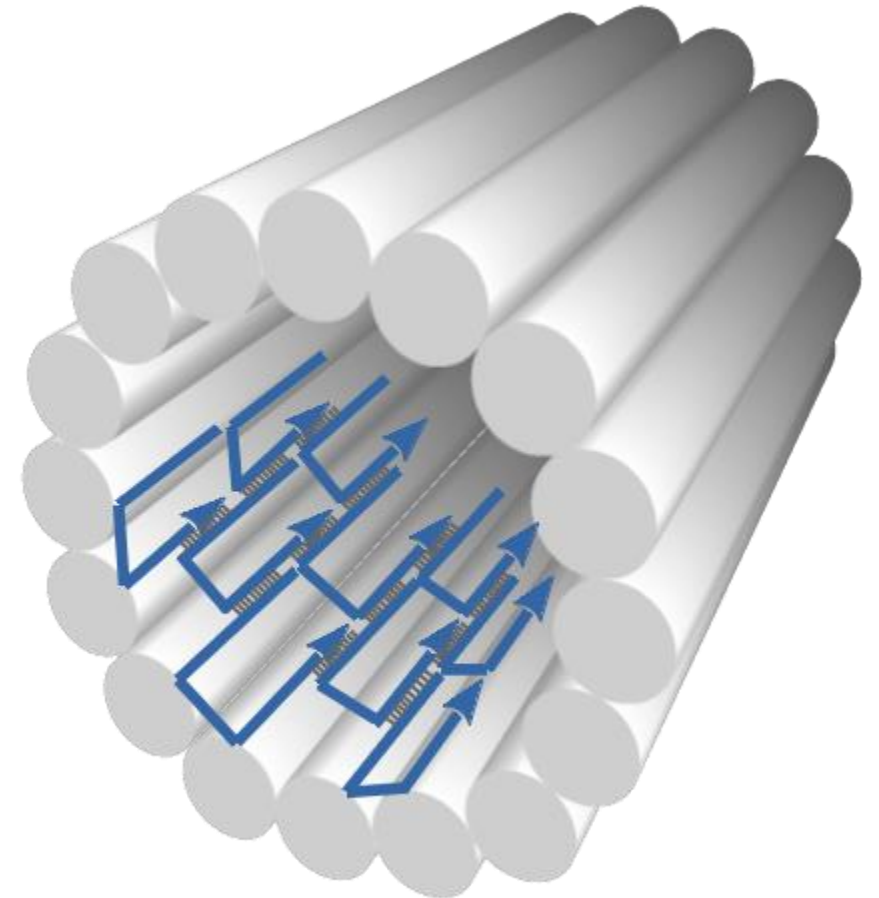


tile for position (4,2)

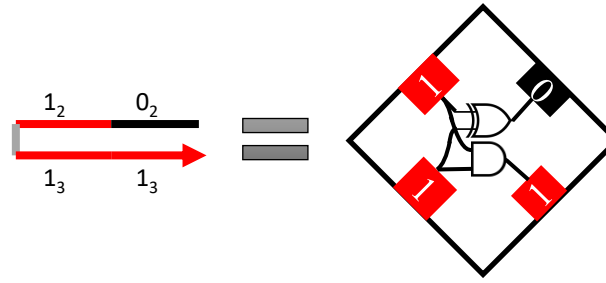
Single-stranded tile tubes



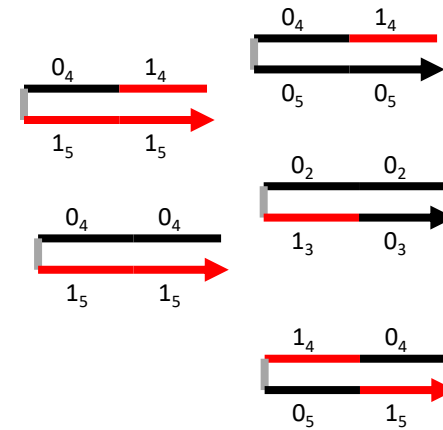
DNA-level diagram of 20-helix tube



Seeded growth



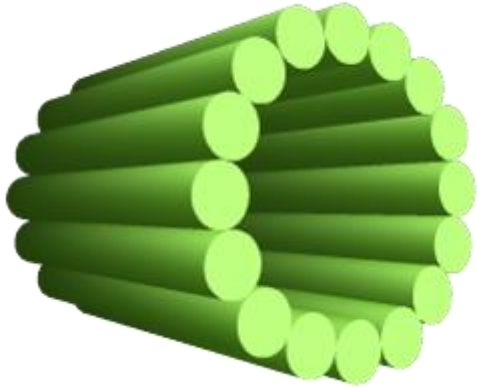
single-stranded tiles
implementing circuit gates



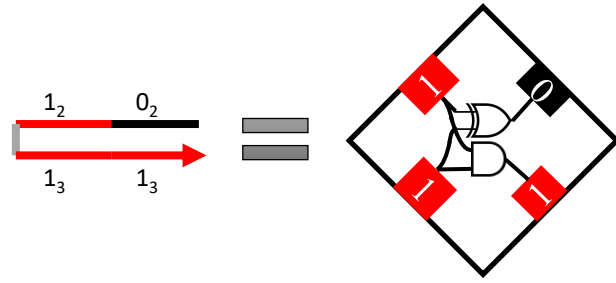
need barrier to nucleation
(tile growth without seed);
[tile]=100 nM;
temperature=50.9° C

Seeded growth

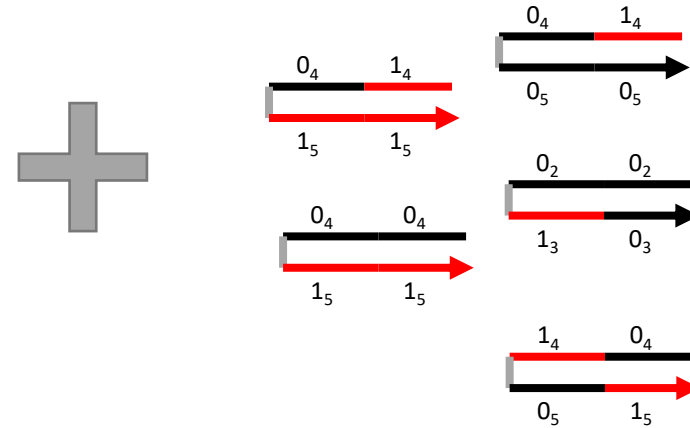
DNA origami seed



need barrier to nucleation
(tile growth without seed);
[tile]=100 nM;
temperature=50.9° C



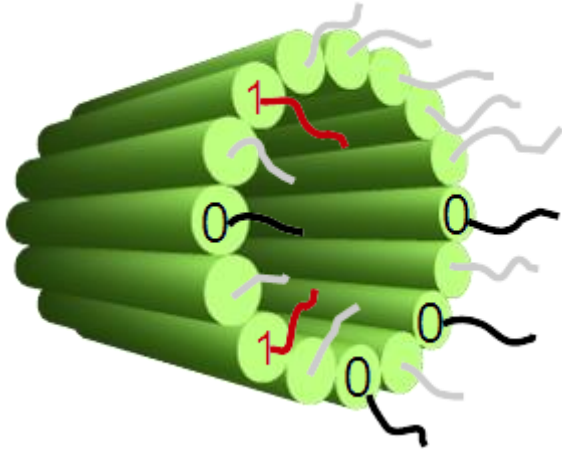
single-stranded tiles
implementing circuit gates



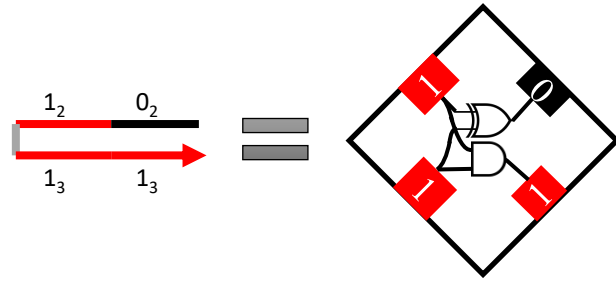
Seeded growth

DNA origami seed

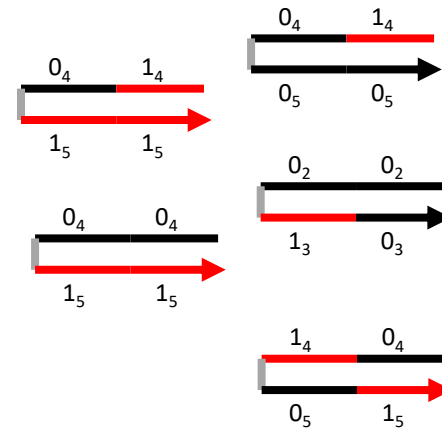
single-stranded “input-adapter”
extensions encoding 6 input bits



need barrier to nucleation
(tile growth without seed);
[tile]=100 nM;
temperature=50.9° C



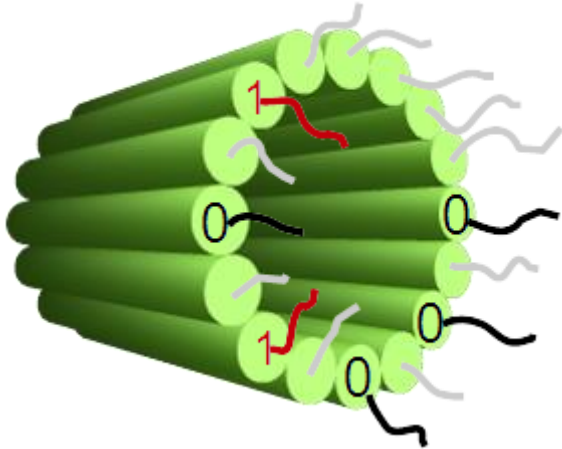
single-stranded tiles
implementing circuit gates



Seeded growth

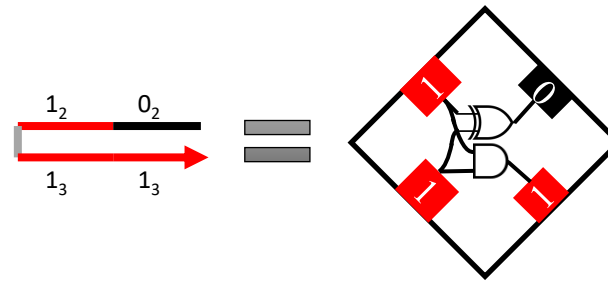
DNA origami seed

single-stranded “input-adapter”
extensions encoding 6 input bits

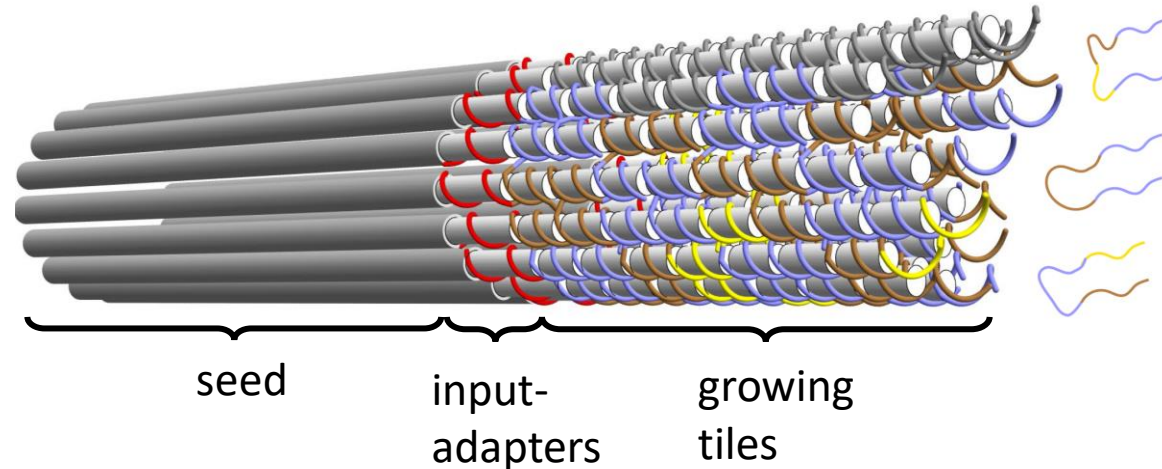
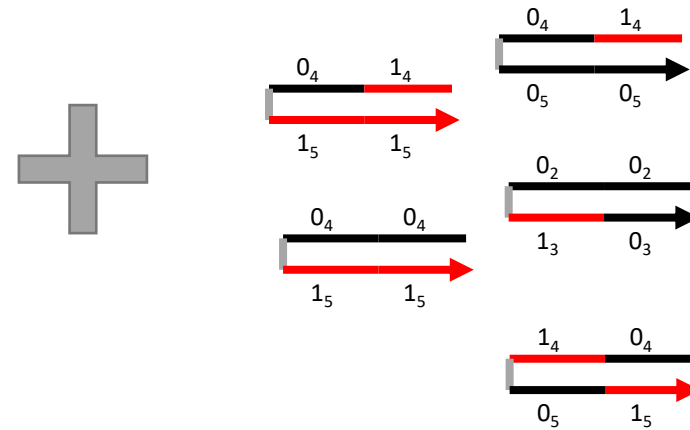


need barrier to nucleation
(tile growth without seed);
[tile]=100 nM;
temperature=50.9° C

hold 8-48 hours



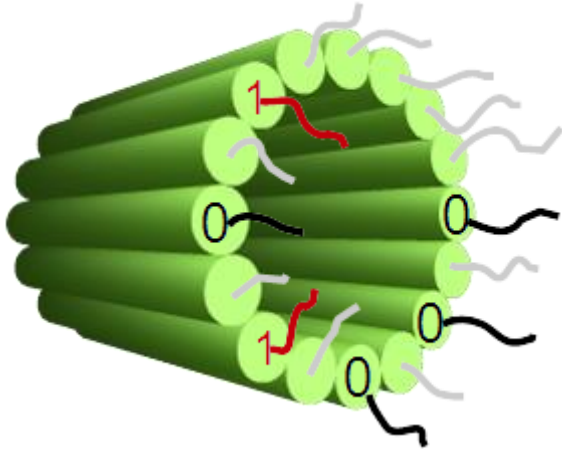
single-stranded tiles
implementing circuit gates



Seeded growth

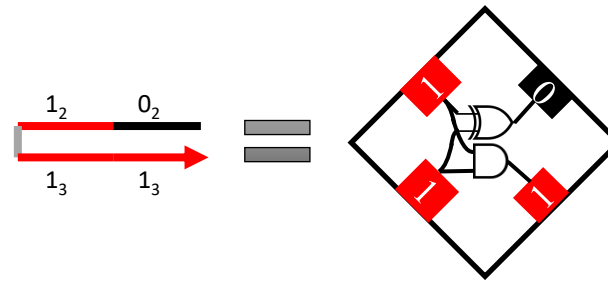
DNA origami seed

single-stranded “input-adapter”
extensions encoding 6 input bits

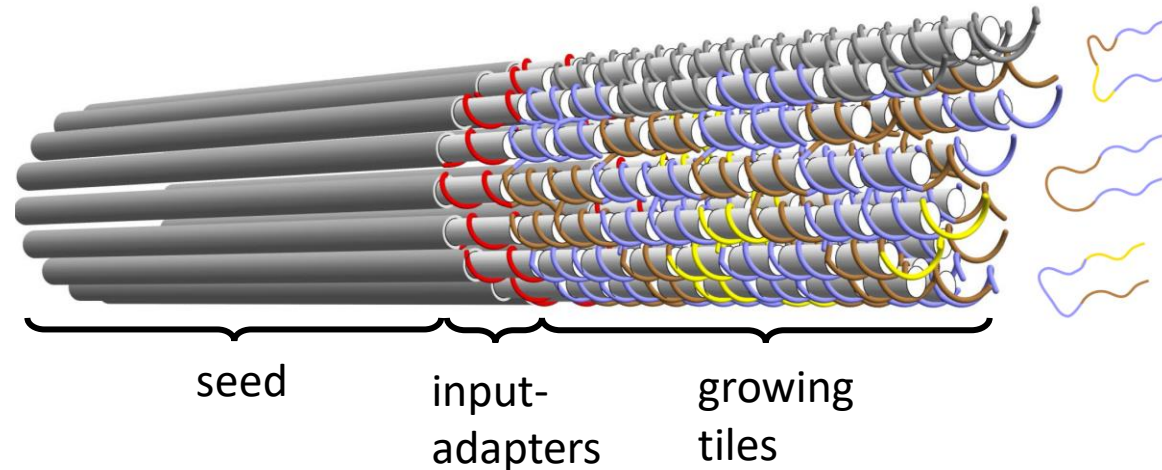
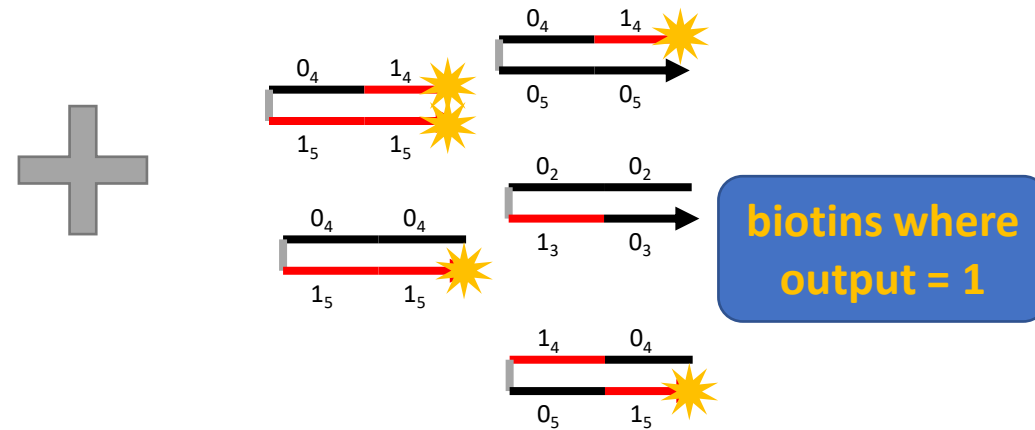


need barrier to nucleation
(tile growth without seed);
[tile]=100 nM;
temperature=50.9° C

hold 8-48 hours



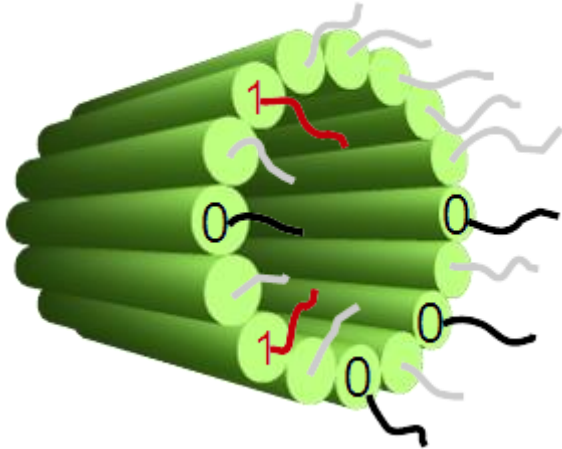
single-stranded tiles
implementing circuit gates



Seeded growth

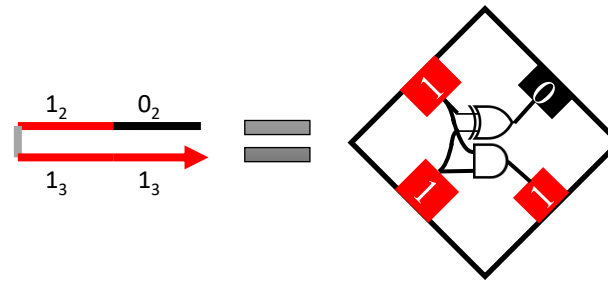
DNA origami seed

single-stranded “input-adapter”
extensions encoding 6 input bits

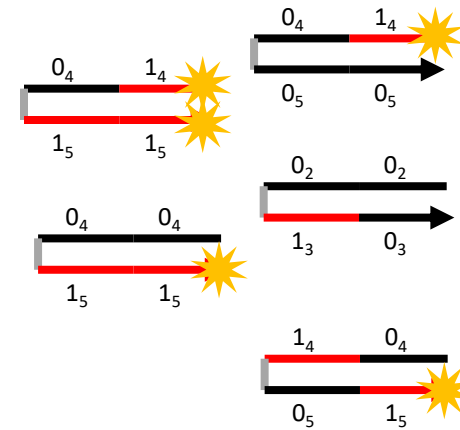


need barrier to nucleation
(tile growth without seed);
[tile]=100 nM;
temperature=50.9° C

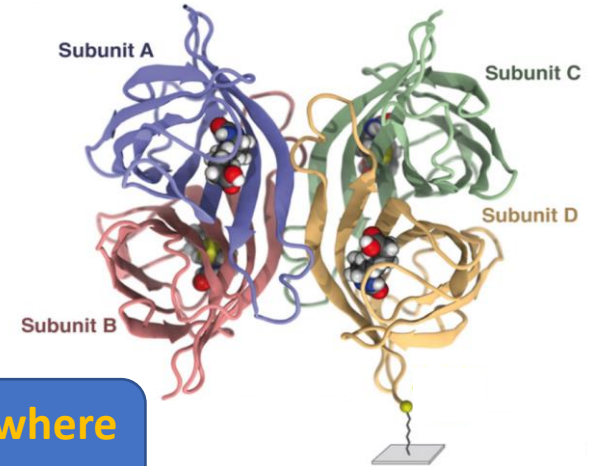
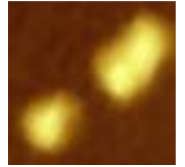
hold 8-48 hours



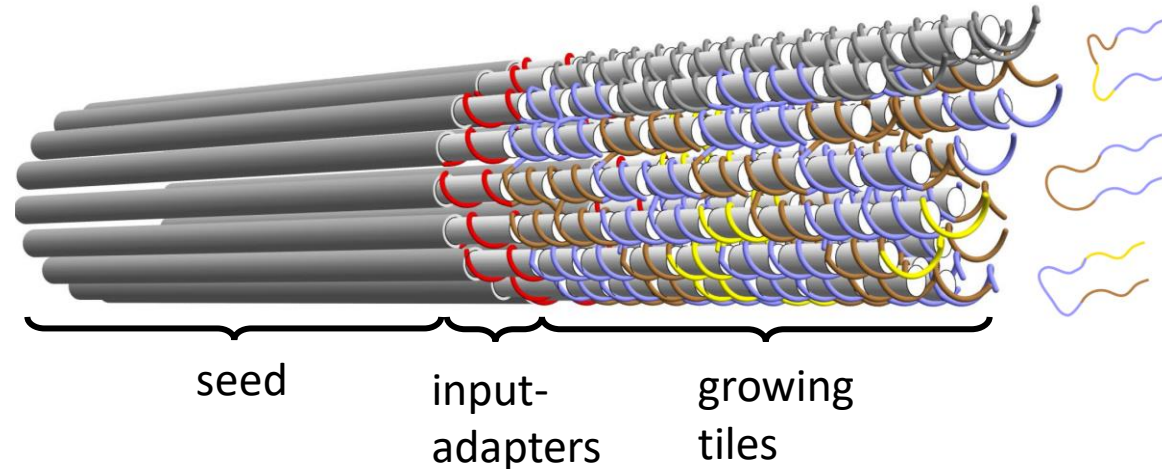
single-stranded tiles
implementing circuit gates



can later add streptavidin (5 nm
wide protein) to bind biotins and
visualize where the 1's are

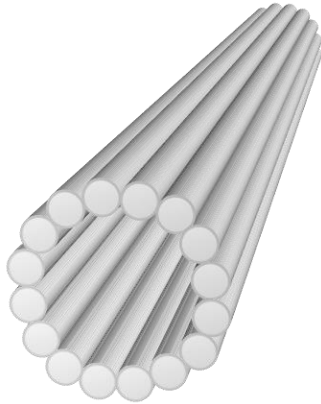


biotins where
output = 1

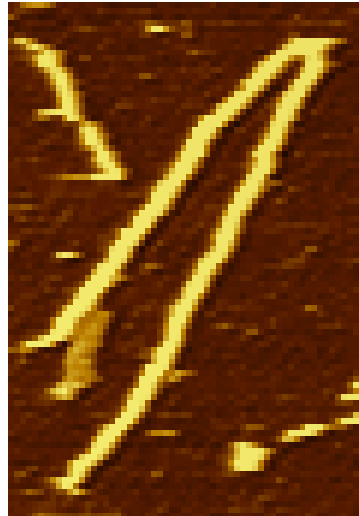


Tubes to ribbons

tube



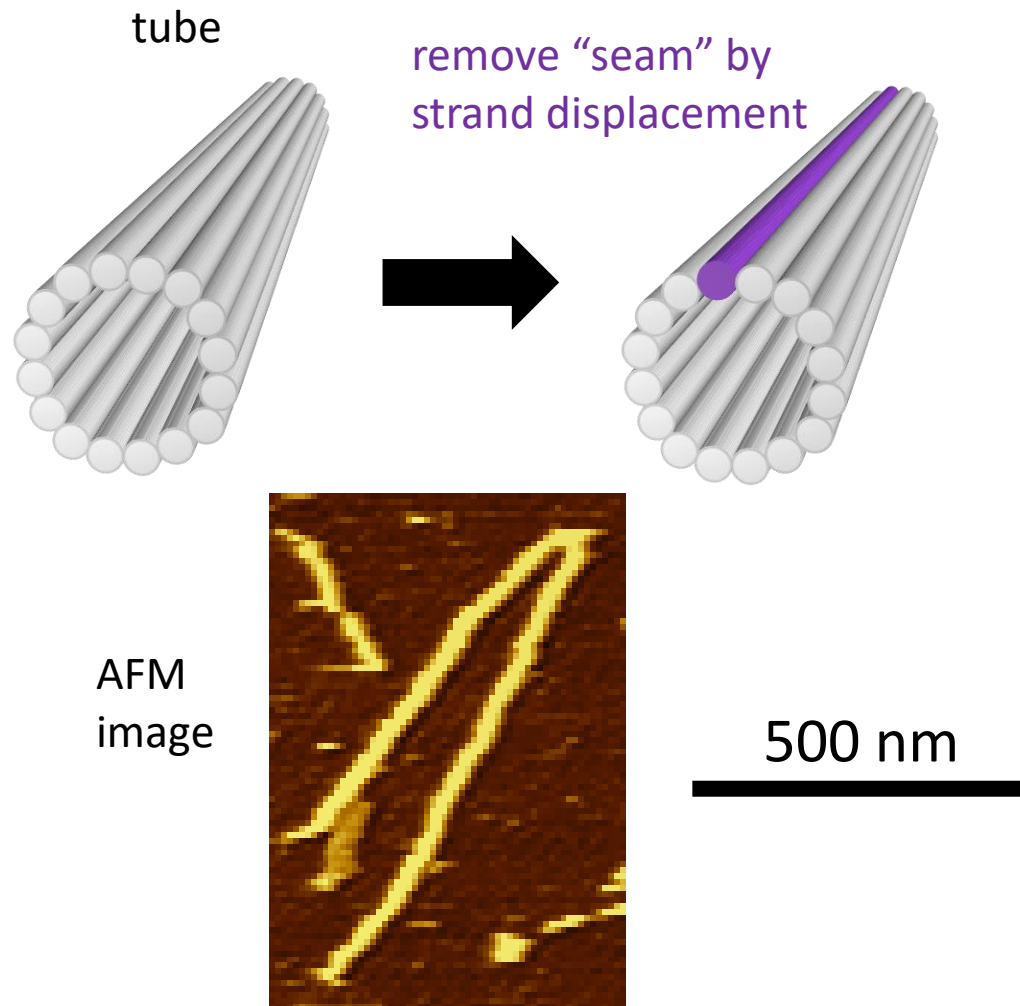
AFM
image



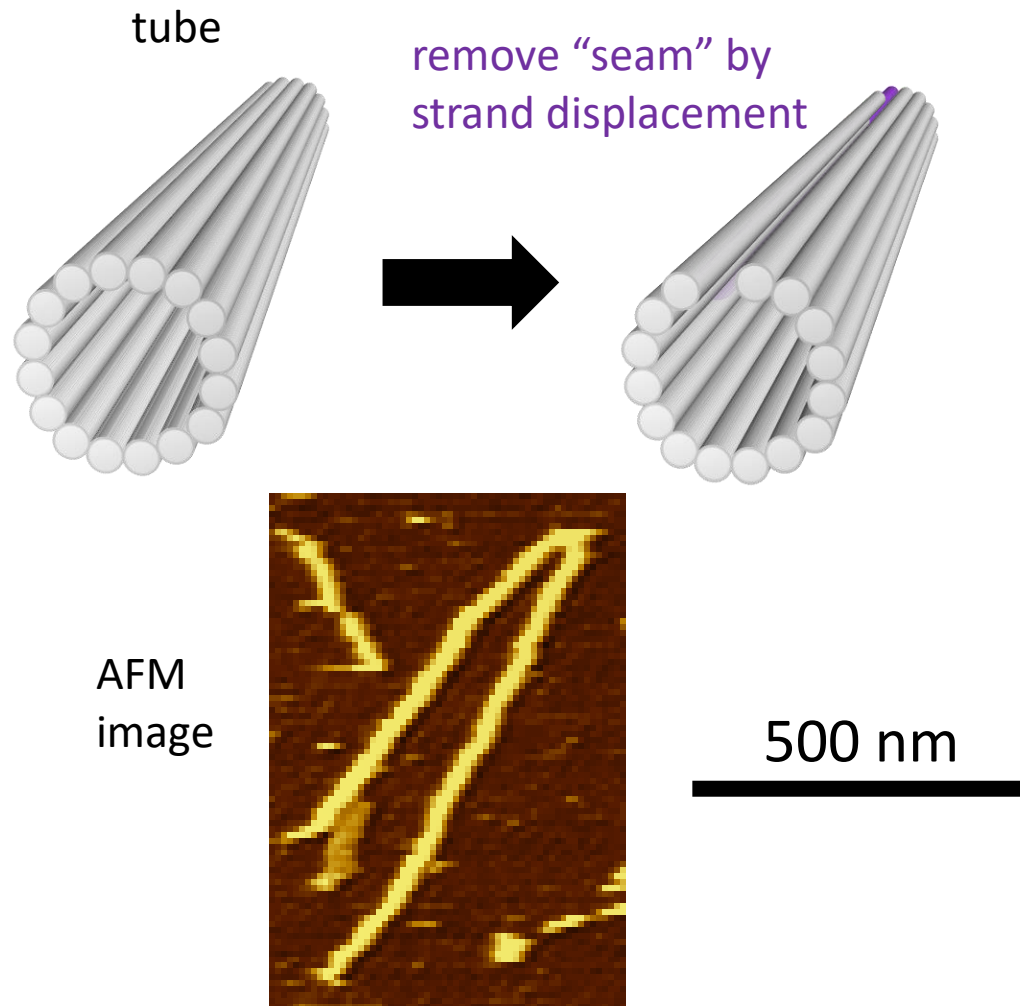
500 nm



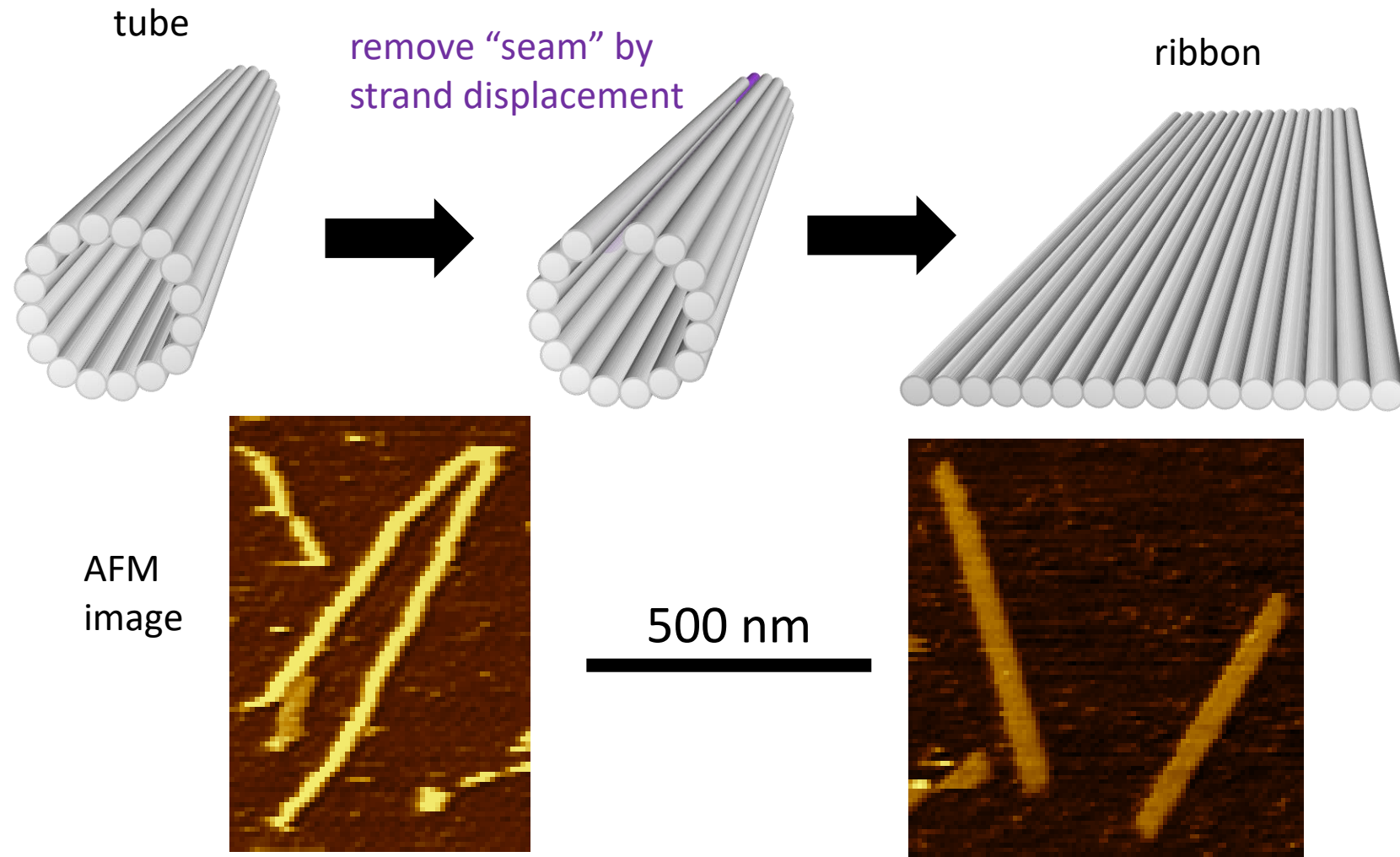
Tubes to ribbons



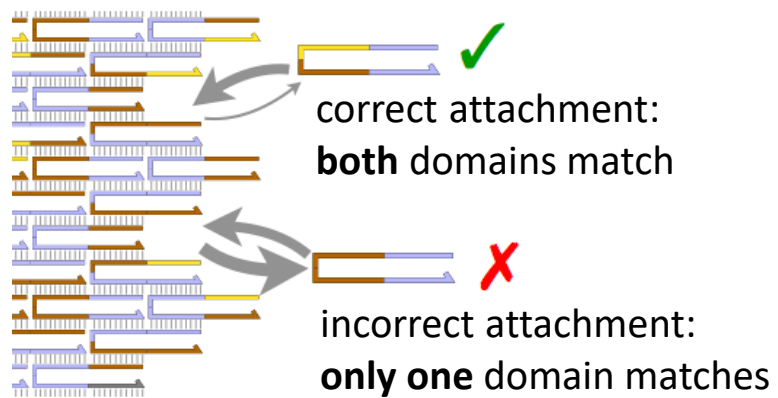
Tubes to ribbons



Tubes to ribbons



DNA sequence design

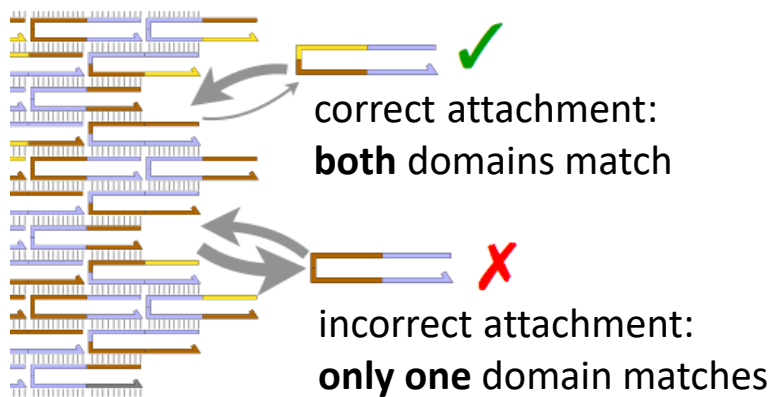
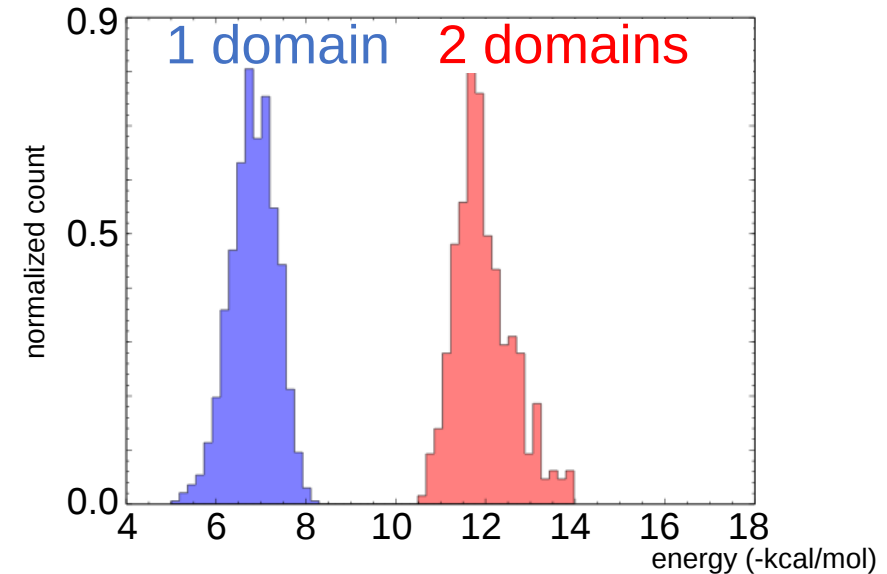
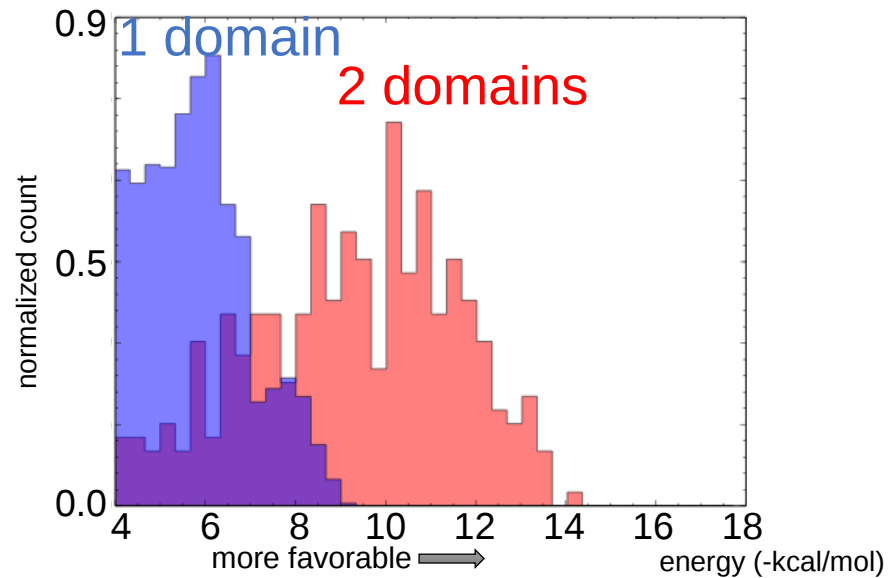


DNA sequence design

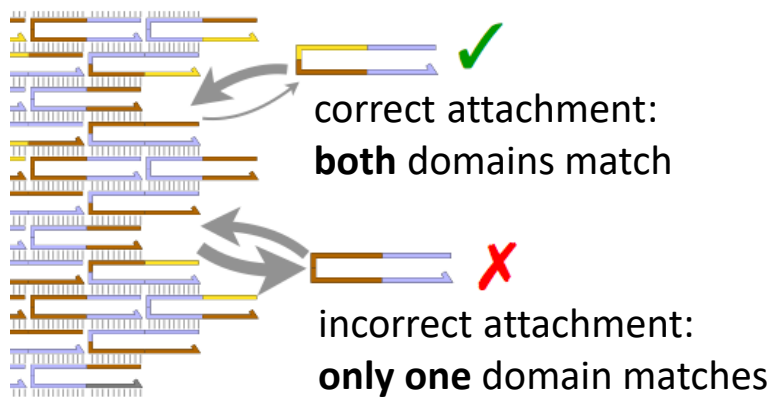
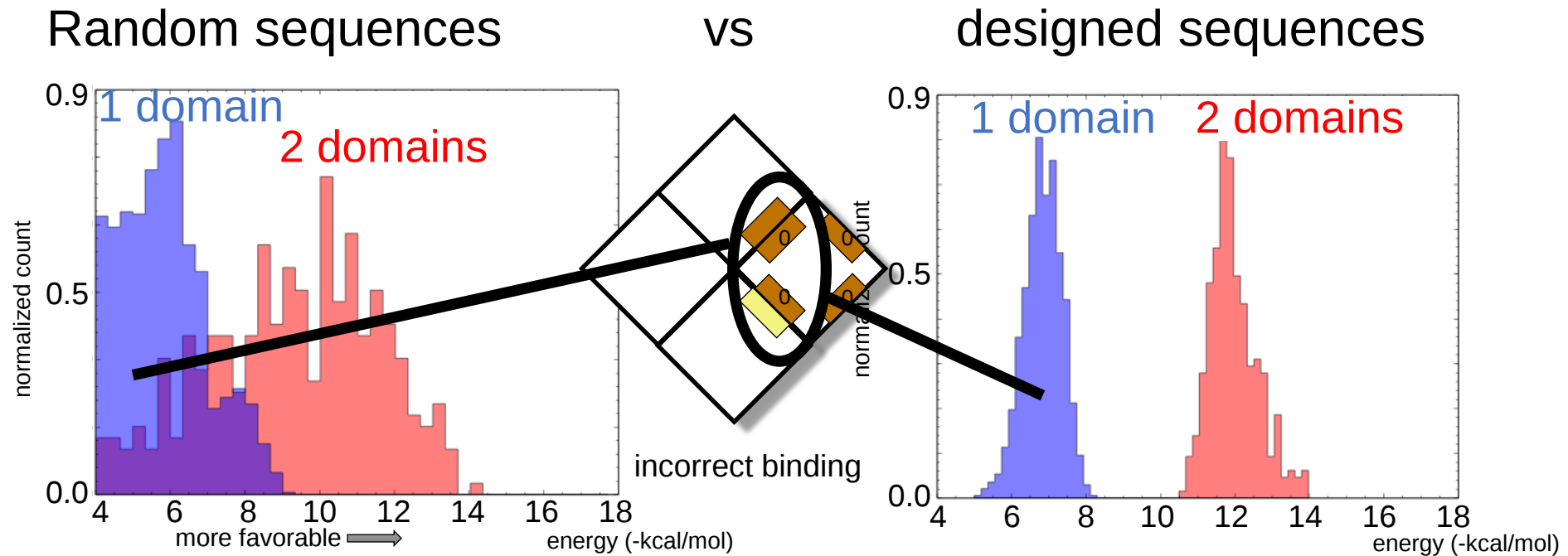
Random sequences

vs

designed sequences



DNA sequence design

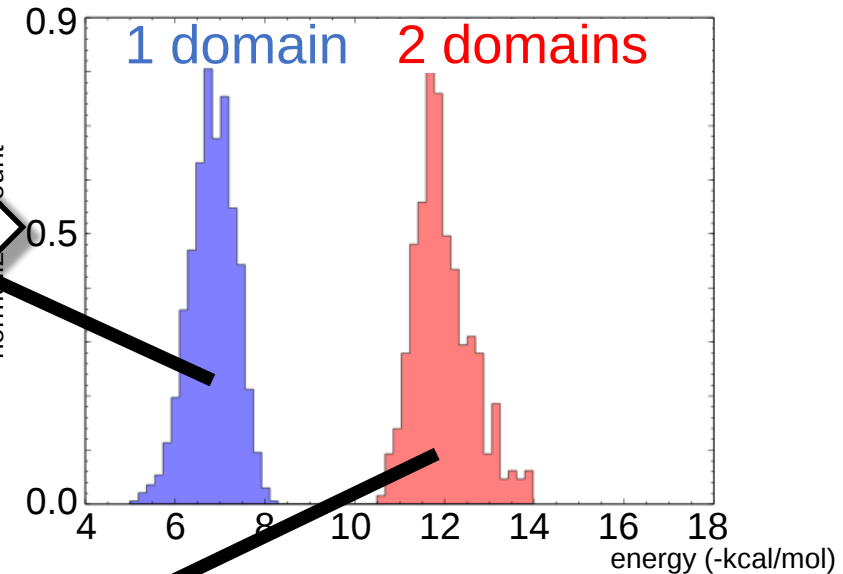
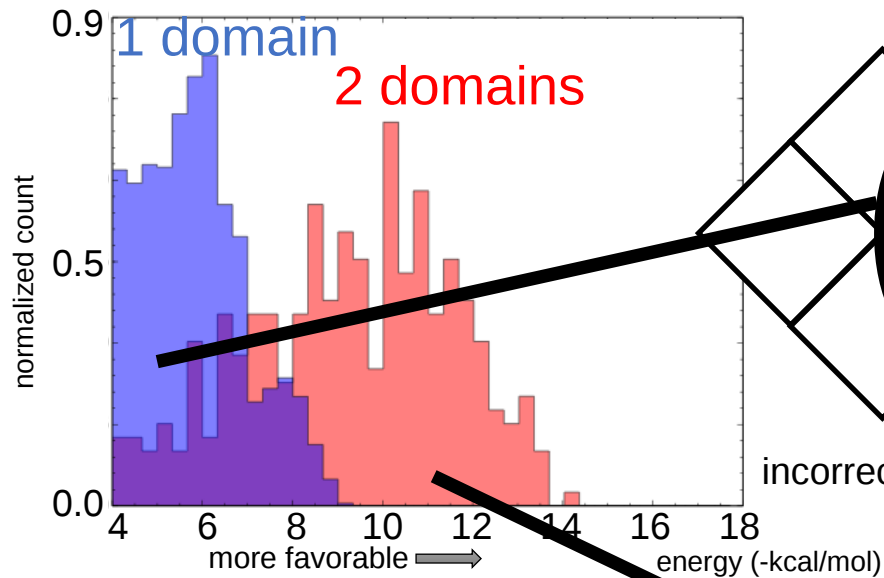


DNA sequence design

Random sequences

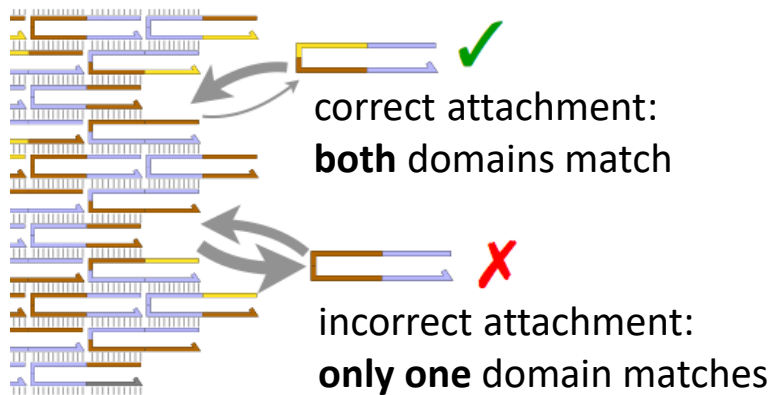
vs

designed sequences



incorrect binding

correct binding

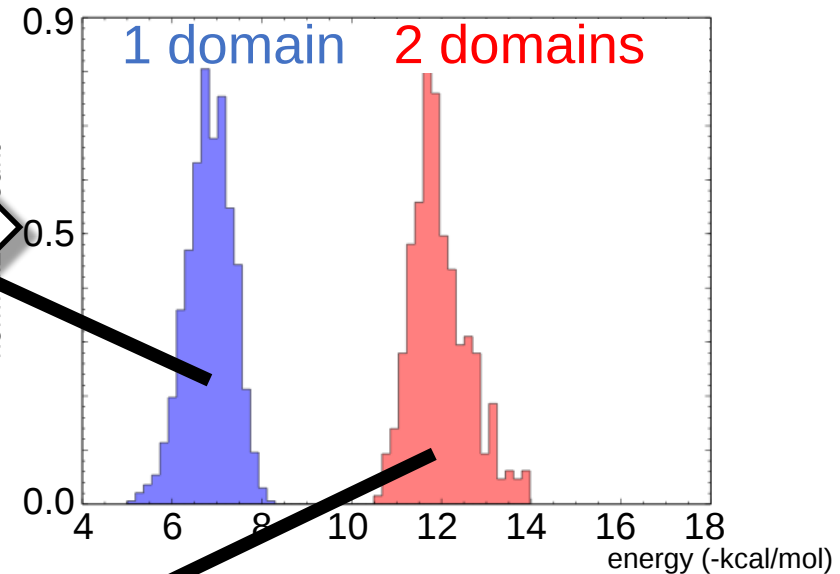
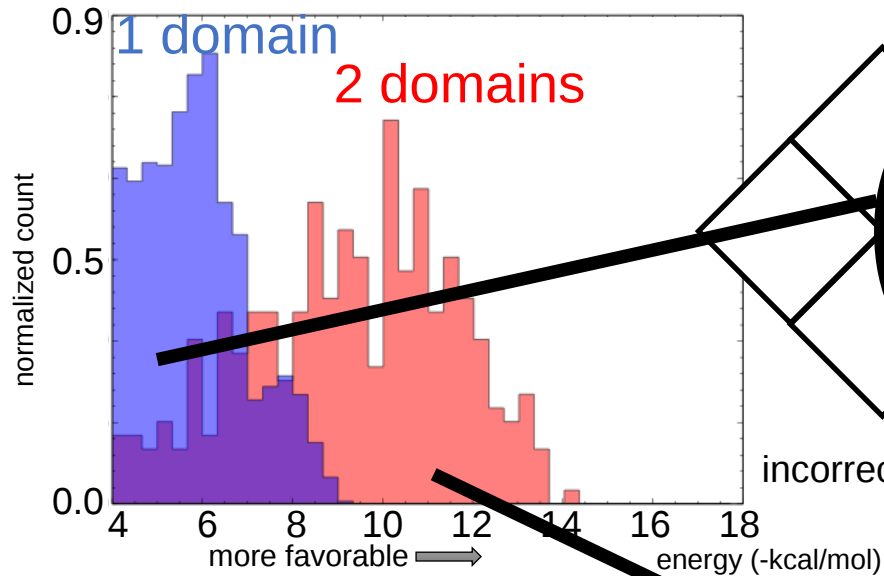


DNA sequence design

Random sequences

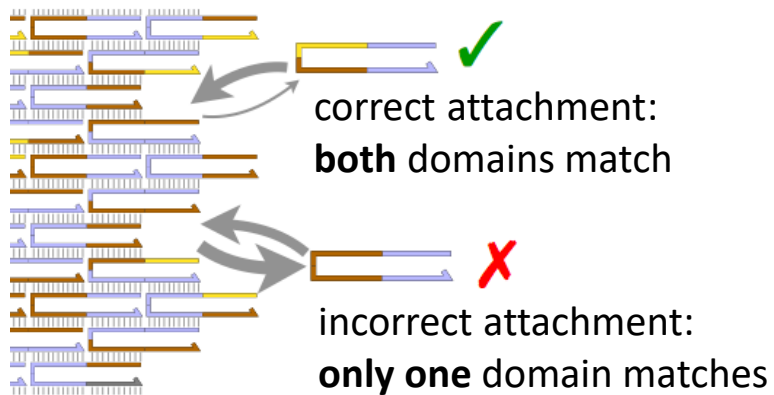
vs

designed sequences



incorrect binding

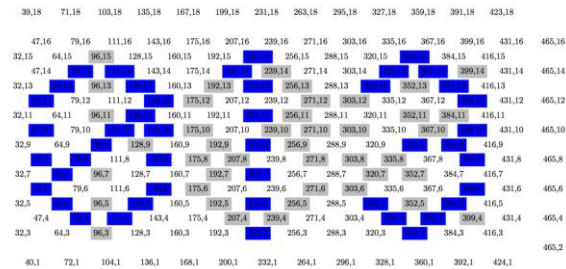
correct binding



Other goals:

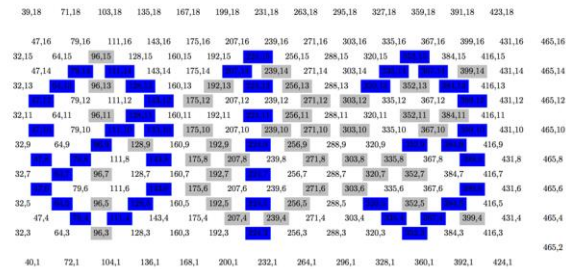
- low strand secondary structure
- low interaction between strands

Bar-coding origami seed for imaging multiple samples at once



some staples of origami seed
have version with a biotin

Bar-coding origami seed for imaging multiple samples at once



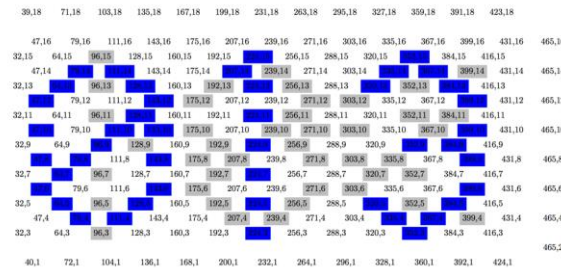
some staples of origami seed have version with a biotin

Generate plate map



```
*****
* pos3wu, coding staples, no biotin (14 staples)
16H barrel stap 2
1 2 3 4 5 6 7 8 9 10 11 12
A
B
C
D
E
F
G
H
pos3wu, coding staples, biotin (16 staples)
16H 3bit biotin staples
1 2 3 4 5 6 7 8 9 10 11 12 1 2 3 4 5 6 7 8 9 10 11 12
A
B
C
D
E
F
G
H
*****
```

Bar-coding origami seed for imaging multiple samples at once



Generate
plate map

some staples of origami seed
have version with a biotin

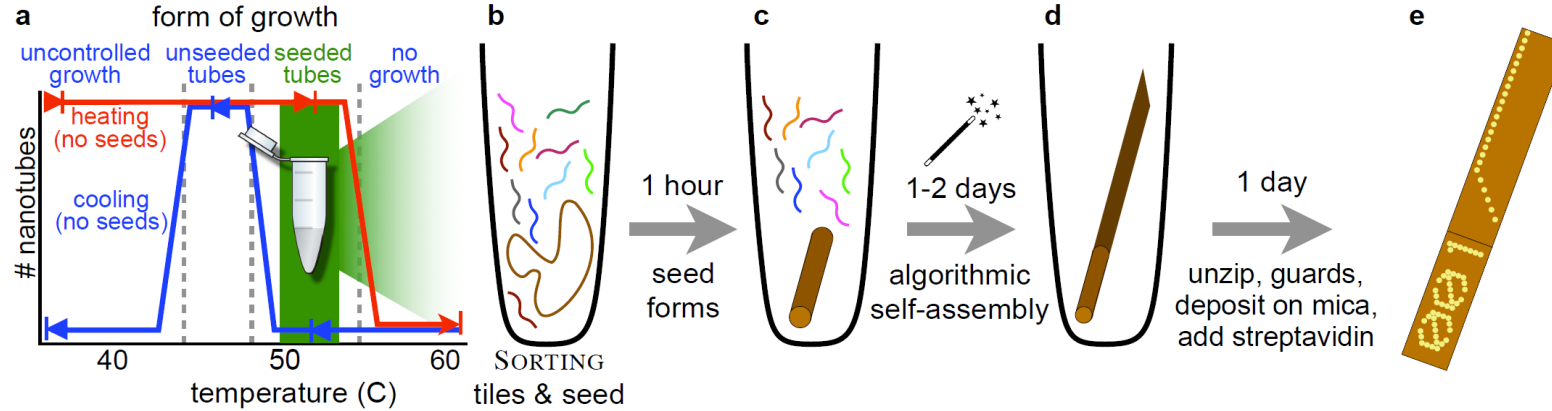
```
*****
* pos3su, coding staples, no biotin (14 staples)
16H barrel stap 2
1 2 3 4 5 6 7 8 9 10 11 12
A      X X
B      X X
C      X X
D      X X
E      X X
F      X X
G      X X
H
pos3su, coding staples, biotin (16 staples)
16H 3bit biotin staples
1 2 3 4 5 6 7 8 9 10 11 12
A      X X
B      X X
C      X X
D      X X
E      X X
F      X X
G      X X
H      X
*****
```

represents some combination of
circuit and input, e.g.,
013 = “parity circuit, input=011010”

label with
streptavidin



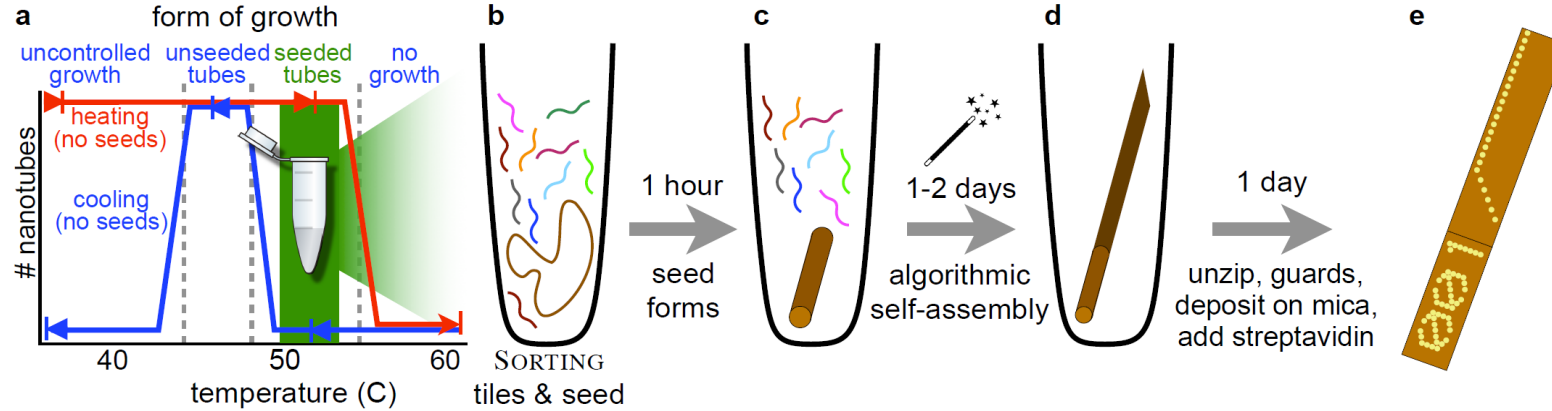
Experimental protocol



To execute circuit γ on input $x \in \{0,1\}^*$:

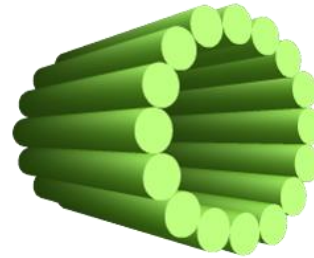
- Mix

Experimental protocol

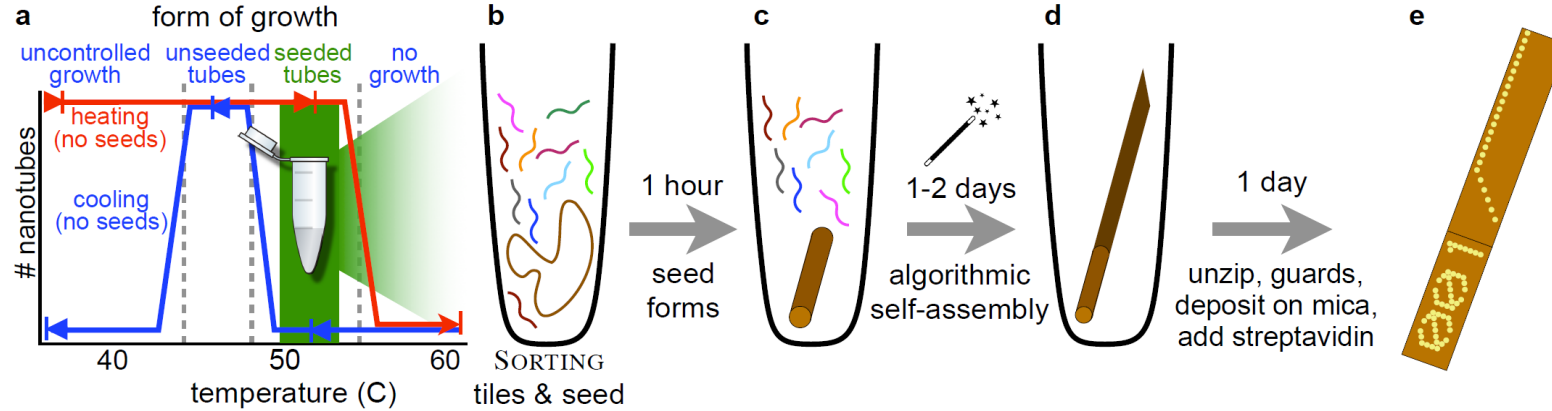


To execute circuit γ on input $x \in \{0,1\}^*$:

- Mix
 - origami seed (bar-coded to identify γ and x)

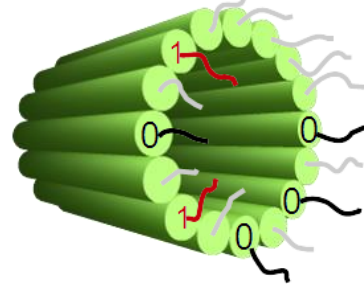


Experimental protocol

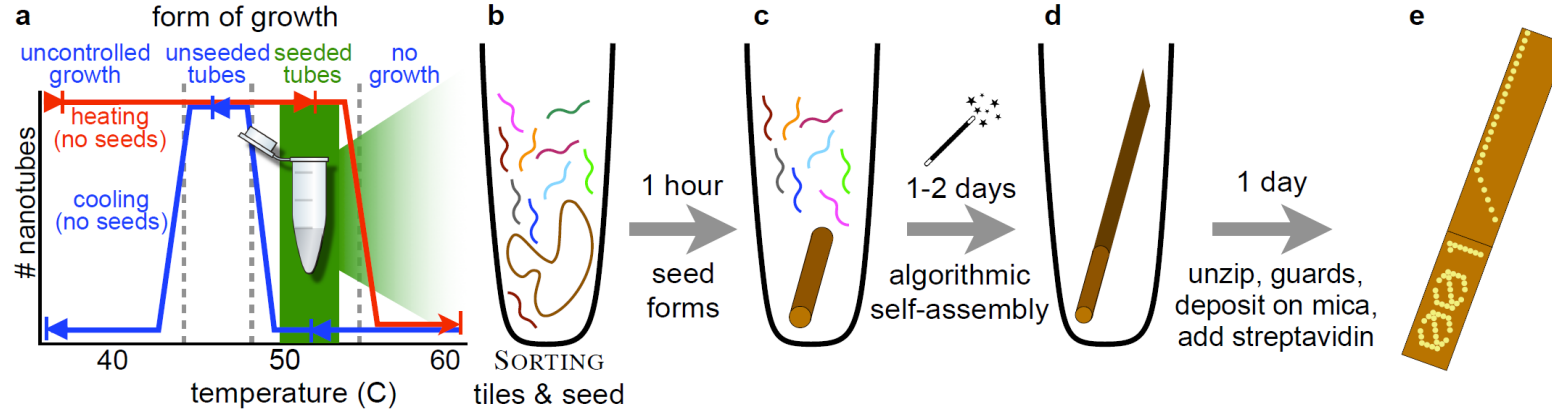


To execute circuit γ on input $x \in \{0,1\}^*$:

- Mix
 - origami seed (bar-coded to identify γ and x)
 - “adapter” strands encoding x



Experimental protocol

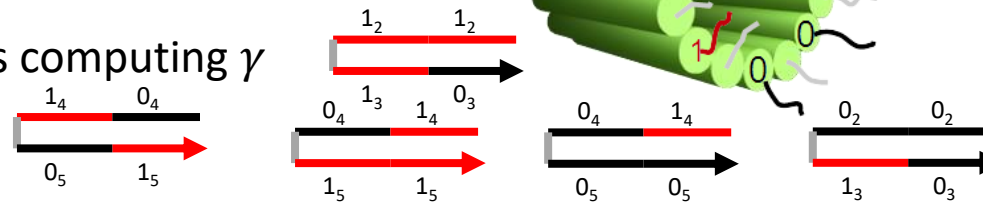


To execute circuit γ on input $x \in \{0,1\}^*$:

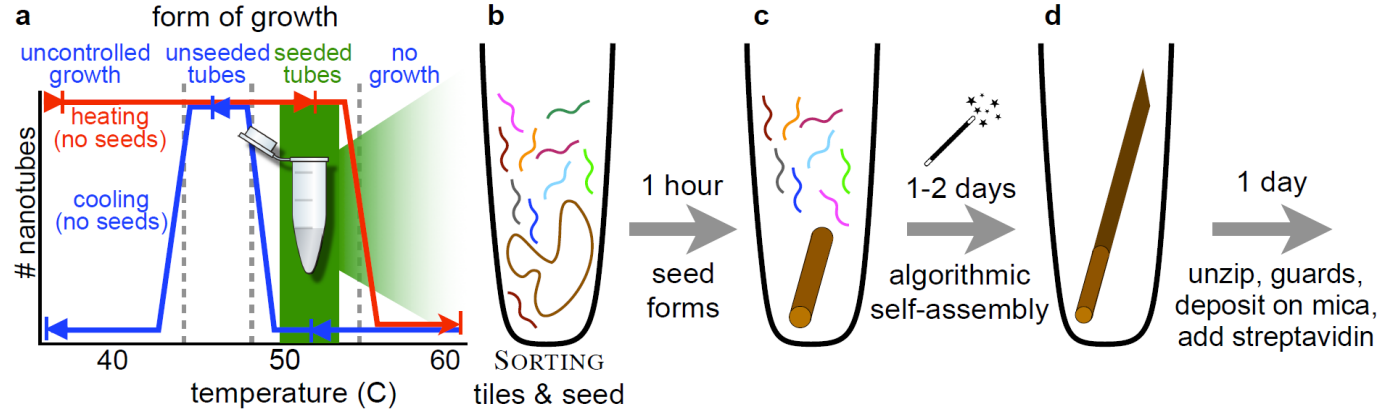
- Mix
 - origami seed (bar-coded to identify γ and x)

- “adapter” strands encoding x

- tiles computing γ



Experimental protocol



To execute circuit γ on input $x \in \{0,1\}^*$:

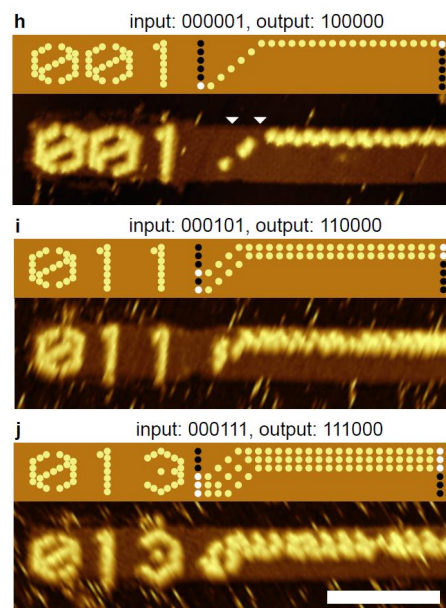
- Mix
 - origami seed (bar-coded to identify γ and x)
 - “adapter” strands encoding x
 - tiles computing γ
-
- The diagram shows the assembly of DNA nanotubes. On the left, four horizontal bars represent different DNA strands with binary labels (0s and 1s) at their ends. These strands are shown joining to form a 3D structure of green cylinders (nanotubes). Each cylinder has a red line (seam) and a black line (guard) on its side. The labels on the strands are: 1₂, 1₂, 0₄, 1₄, 0₅, 1₅, 0₄, 1₃, 1₄, 0₃, 0₄, 1₄, 0₅, 0₅, 0₂, 0₂, 1₃, 0₃.
- Anneal 90° C to 50.9° C in 1 hour (*origami seeds form*)
 - Hold at 50.9° C for 1-2 days (*tiles grow tubes from seed*)
 - Add “unzipper” strands (remove seam to convert tube to ribbon)
 - Add “guard” strands (complements of output sticky ends, to deactivate tiles)
 - Deposit on mica, buffer wash, add streptavidin, AFM



Results

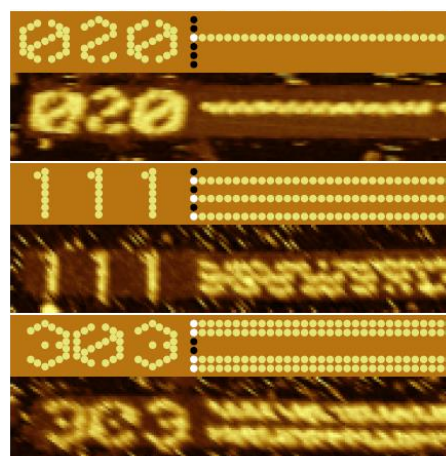
```
def test_parity():  
    actual = parity('100101')  
  
    expected =   
    assertEquals(expected, actual)
```

SORTING



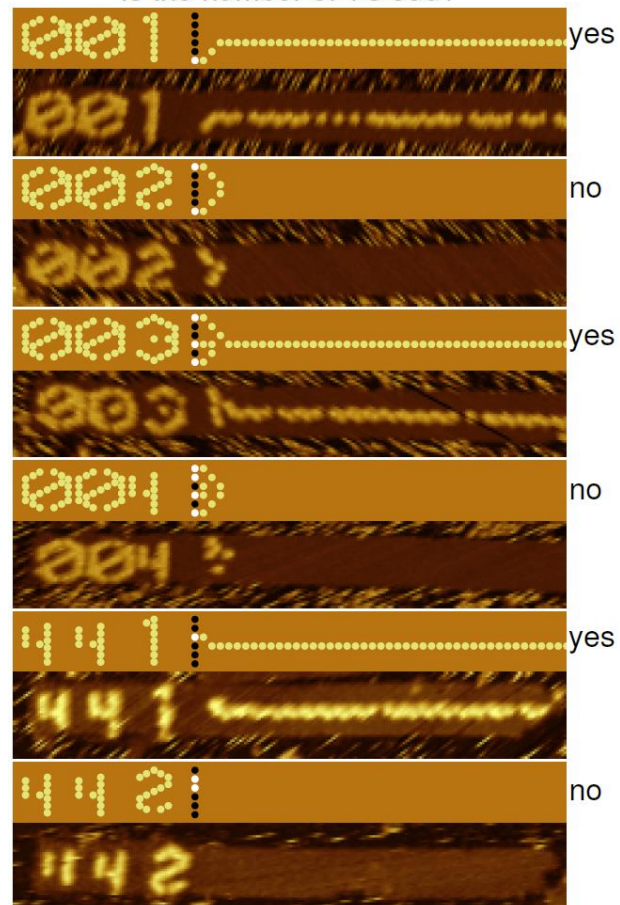
100 nm

COPY



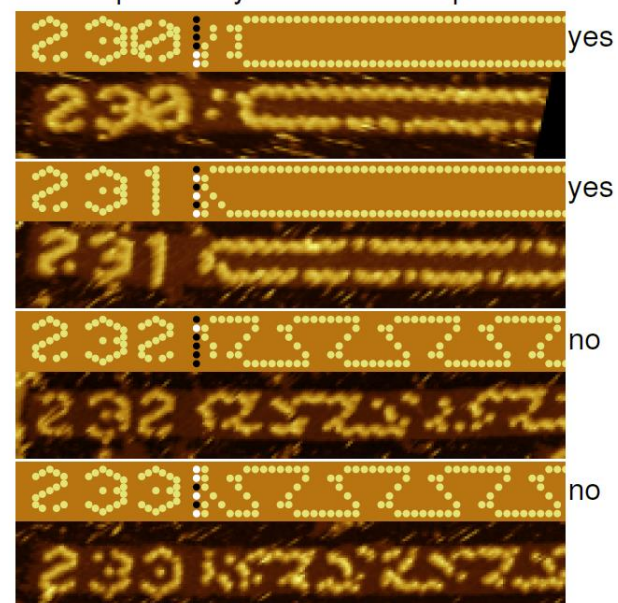
PARITY

Is the number of 1's odd?



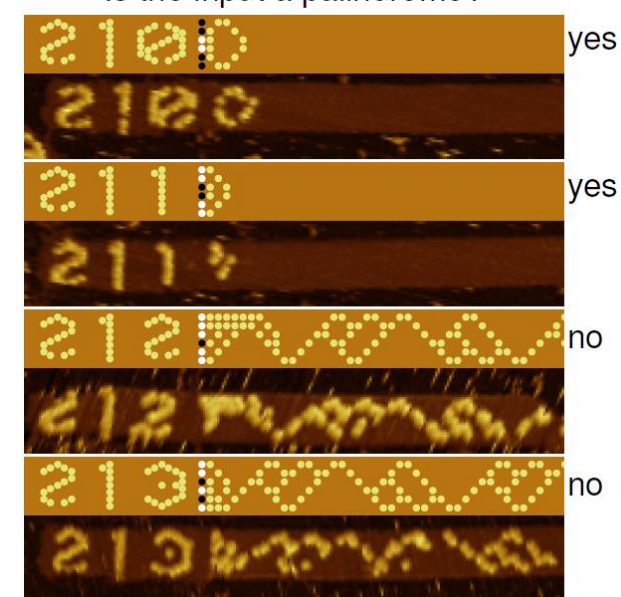
MULTIPLEOF3

Is the input binary number a multiple of 3?



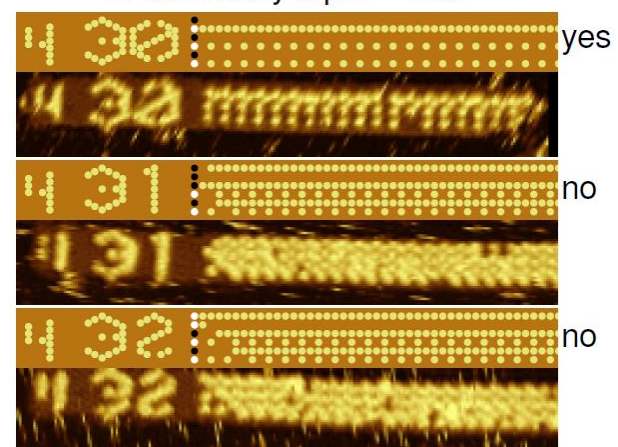
PALINDROME

Is the input a palindrome?



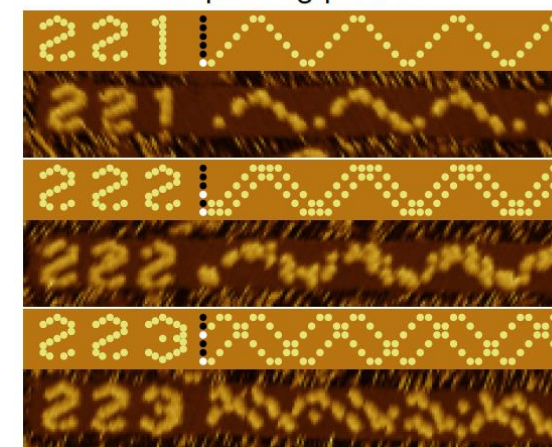
RECOGNISE21

Is the binary input = 21?

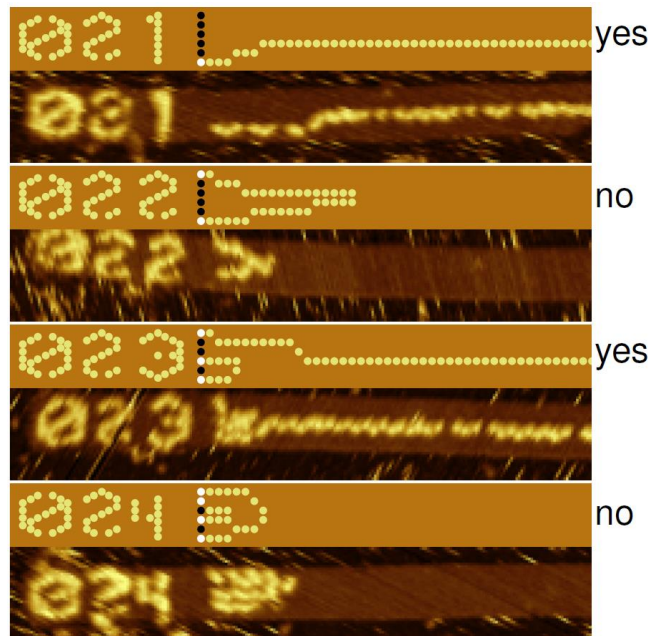


ZIG-ZAG

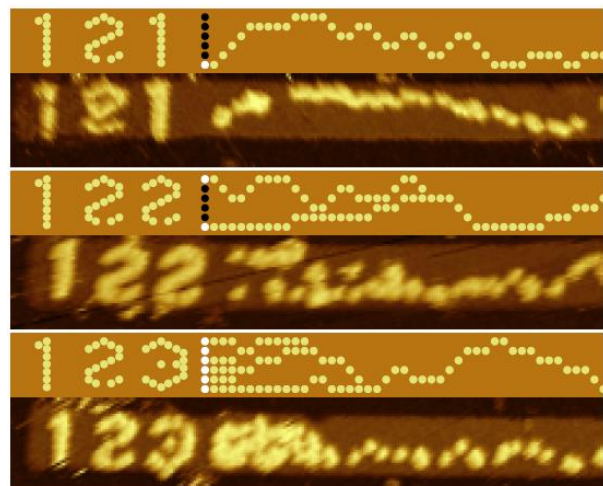
Repeating pattern



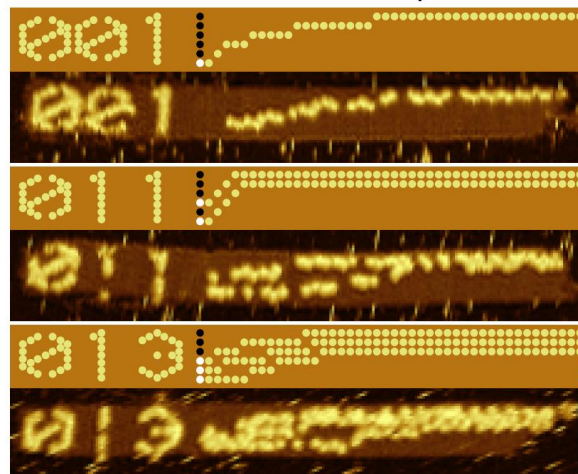
LAZYPARITY



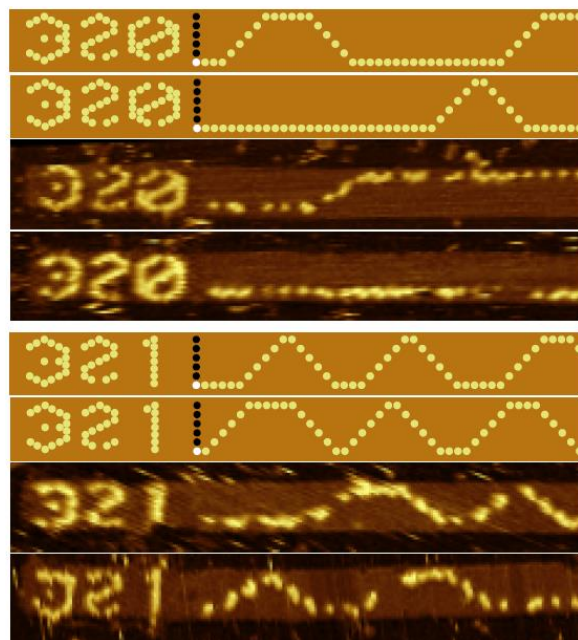
LEADERELECTION



LAZYSORTING



WAVES

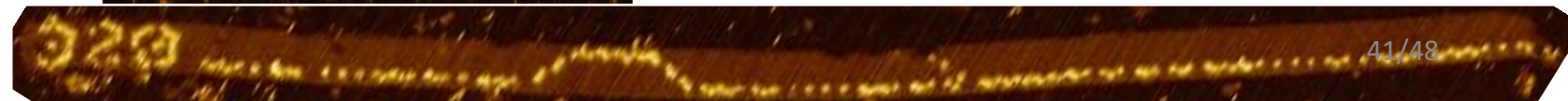
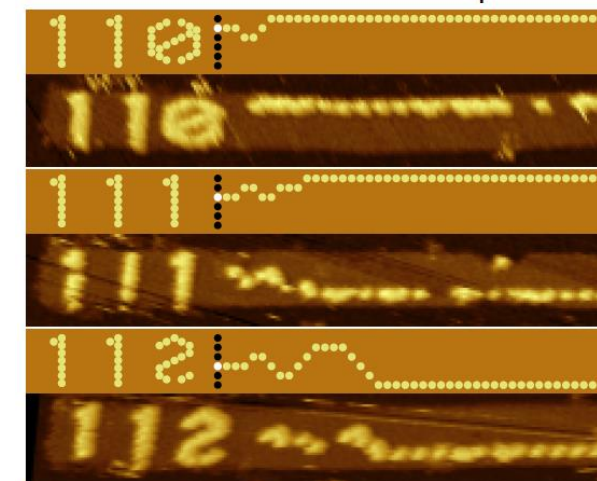


RANDOMWALKINGBIT



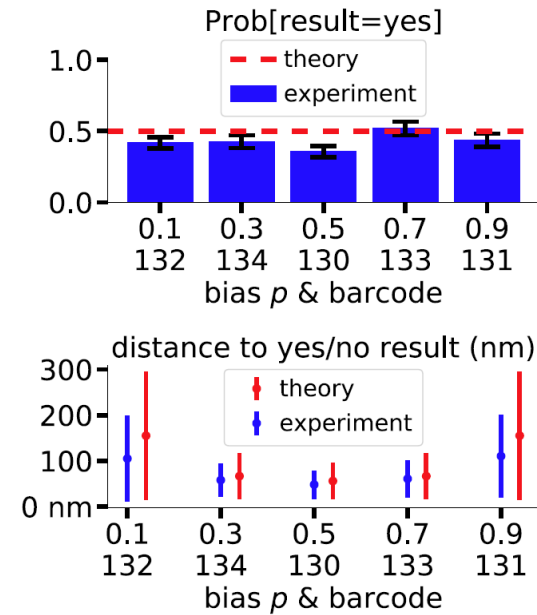
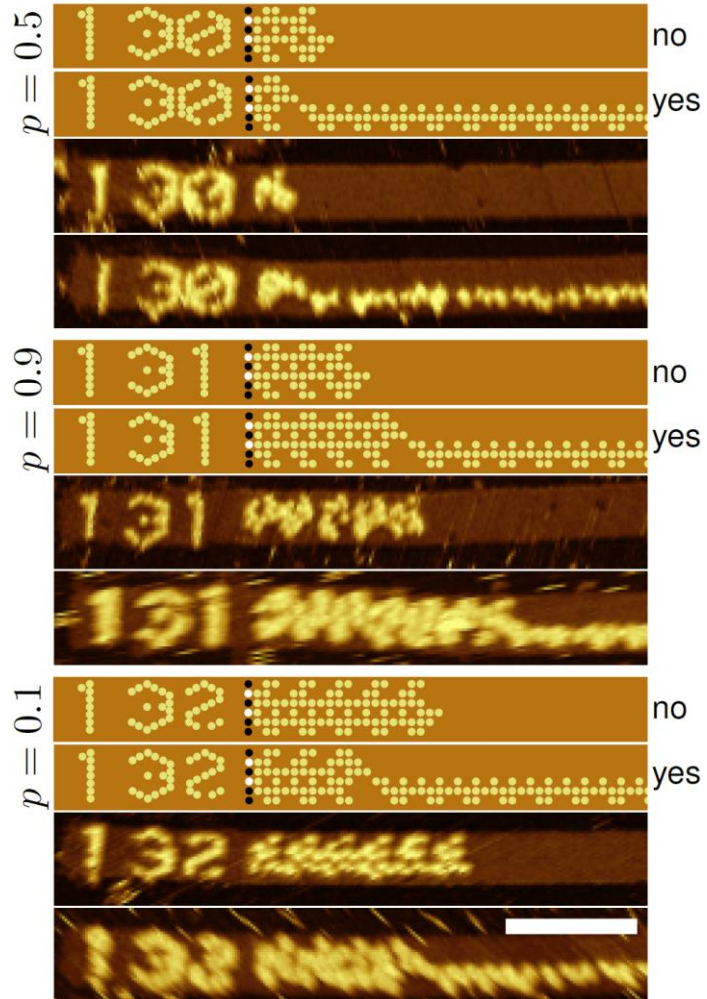
ABSORBINGRANDOMWALKINGBIT

Random walker absorbs to top/bottom



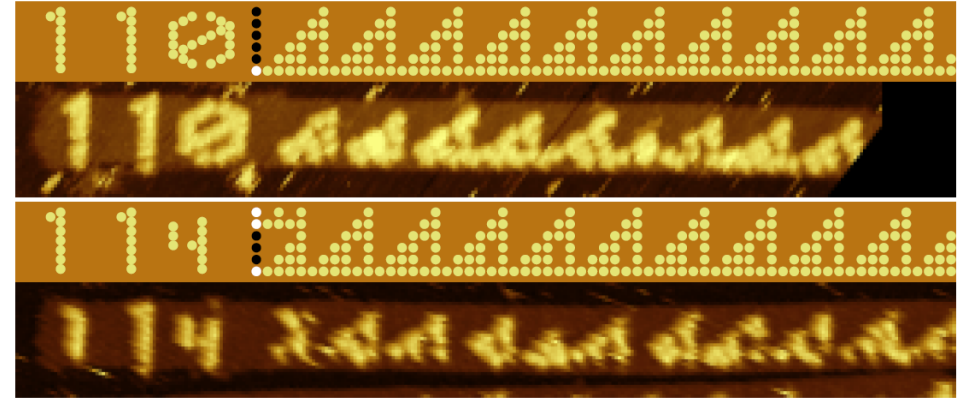
FAIRCOIN

Unbiasing a biased coin



RULE110

Simulation of a cellular automaton



Counting to 63

Circuit with 63 distinct strings



Is there a 64-counter?

No!

Proof by Tristan Stérin, Maynooth University

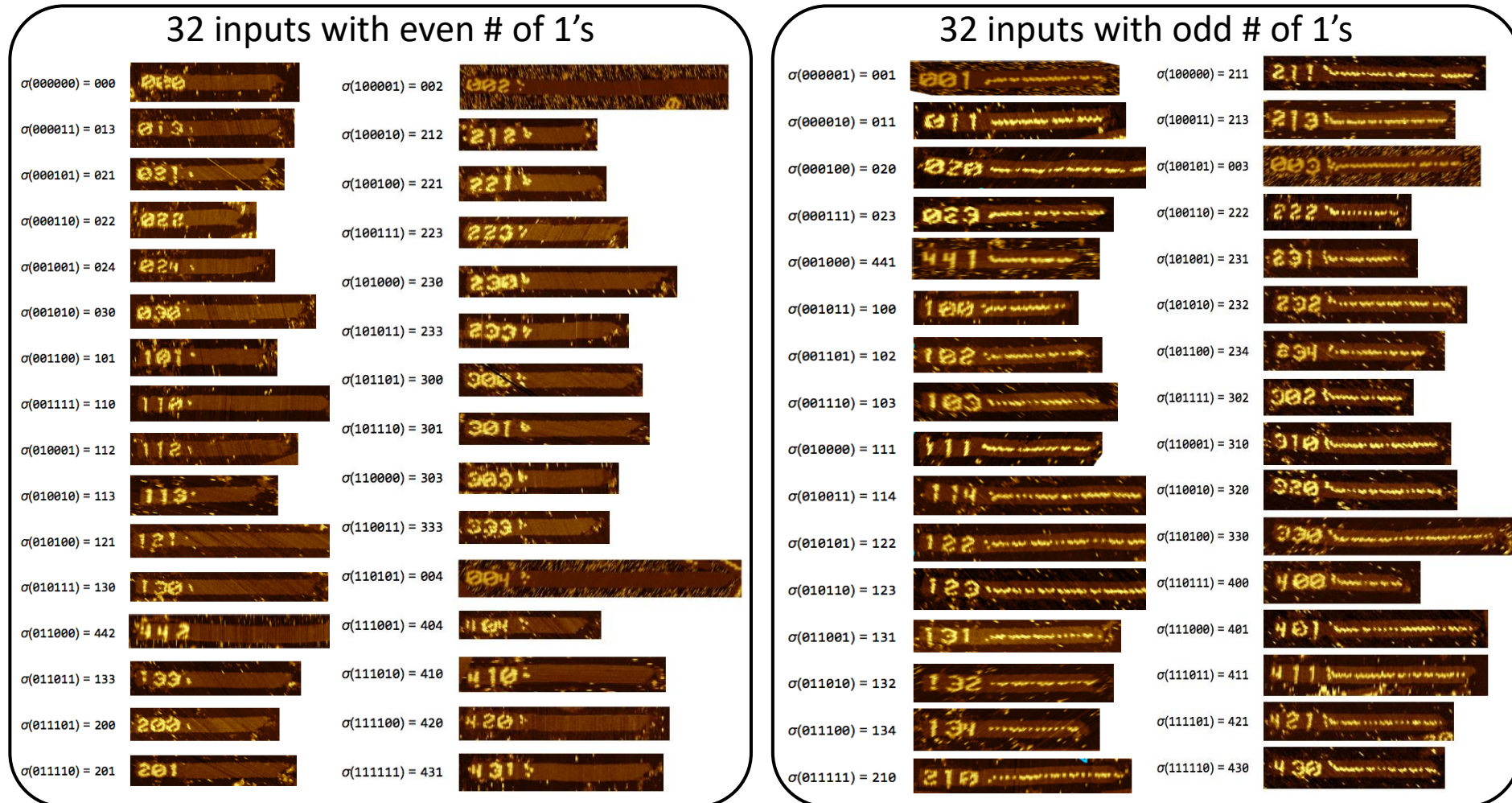
Consequence of following theorem:

*No Boolean function computes an odd permutation
if some output bit does not depend on all input bits.*



Parity tested on all inputs

$2^6 = 64$ inputs with 6 bits

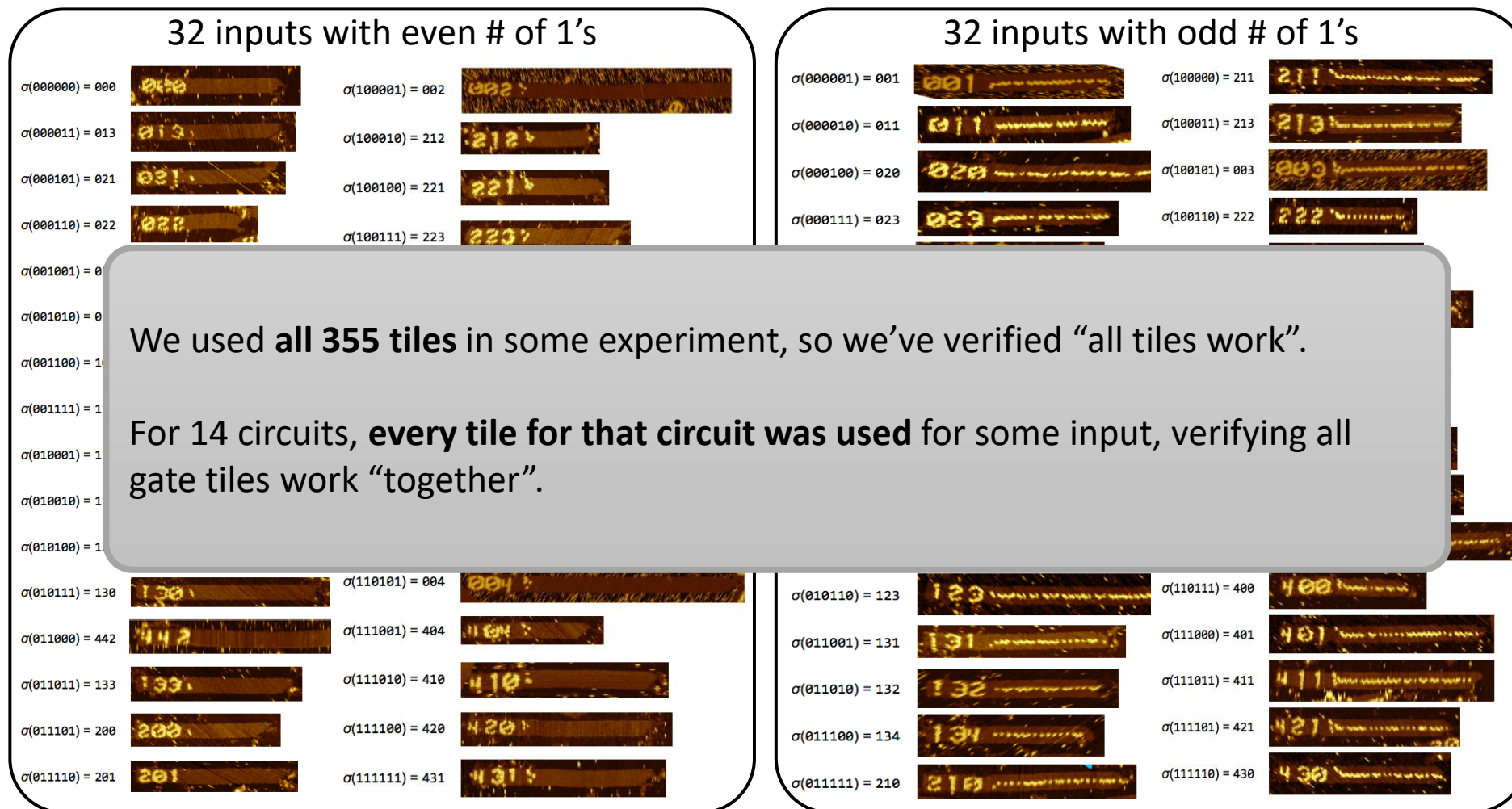


$\sigma(6\text{-bit input}) = 3\text{-digit barcode representing that input}$

150 nm

Parity tested on all inputs

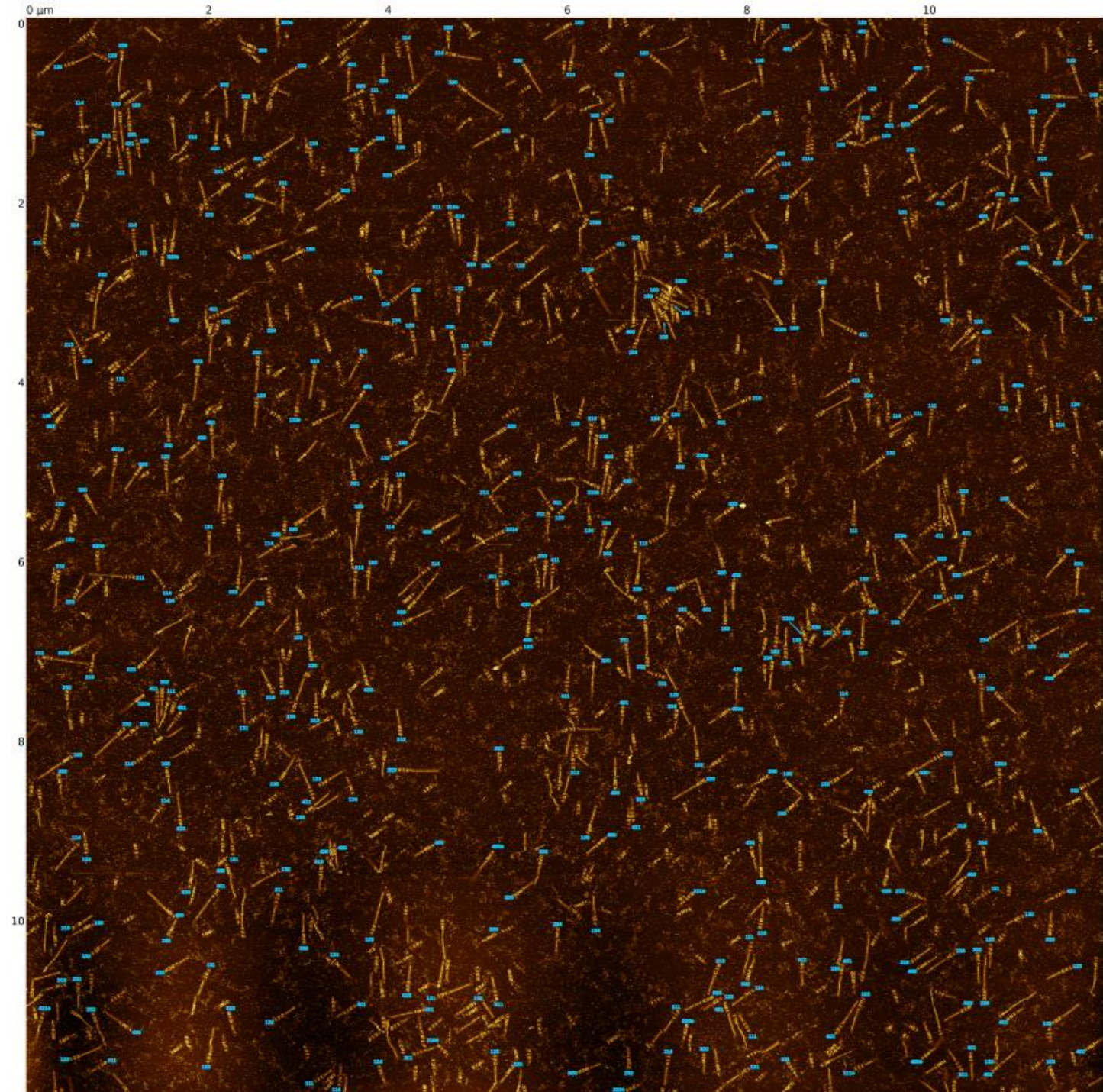
$2^6 = 64$ inputs with 6 bits



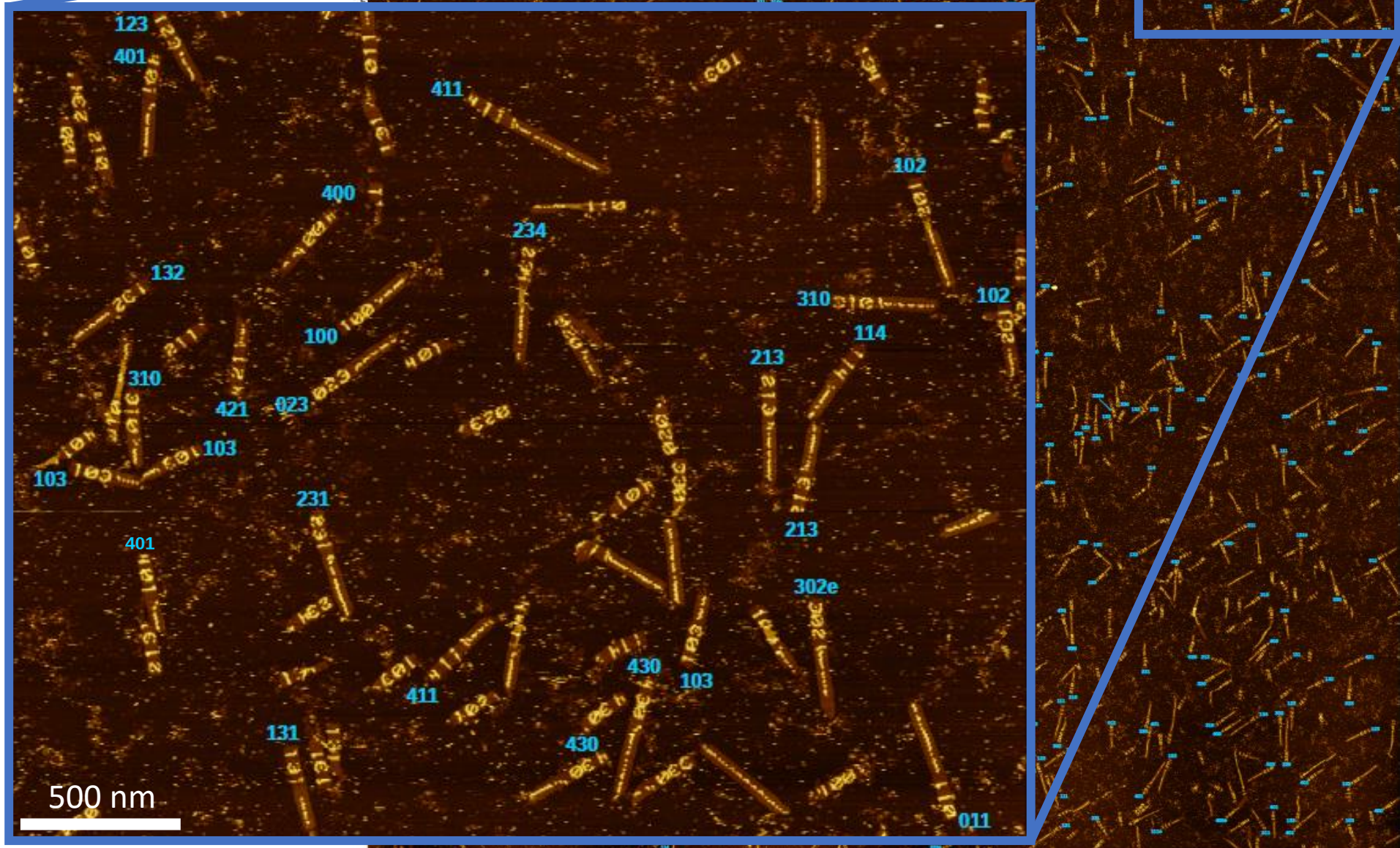
σ (6-bit input) = 3-digit barcode representing that input

150 nm

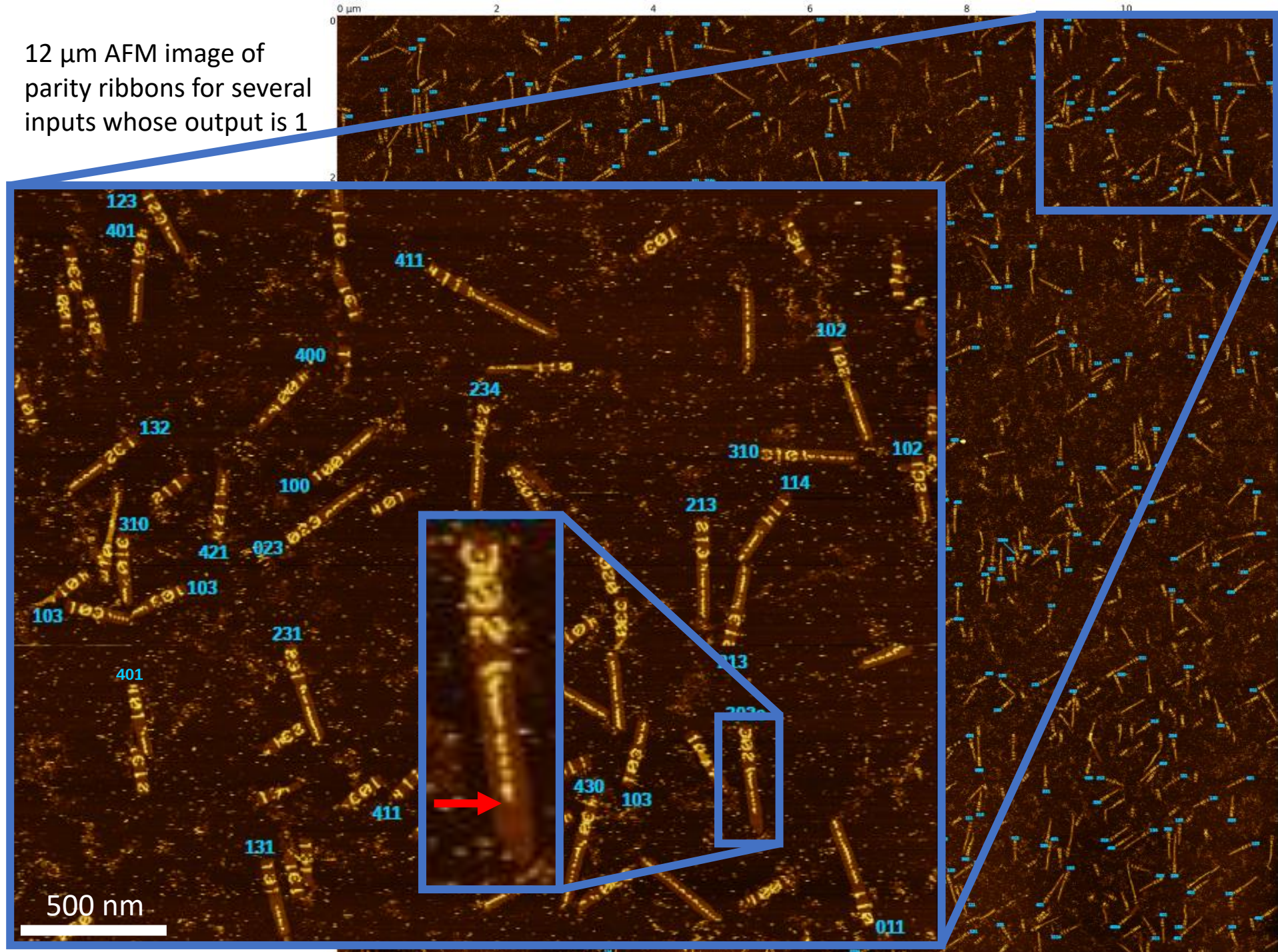
12 μm AFM image of
parity ribbons for several
inputs whose output is 1



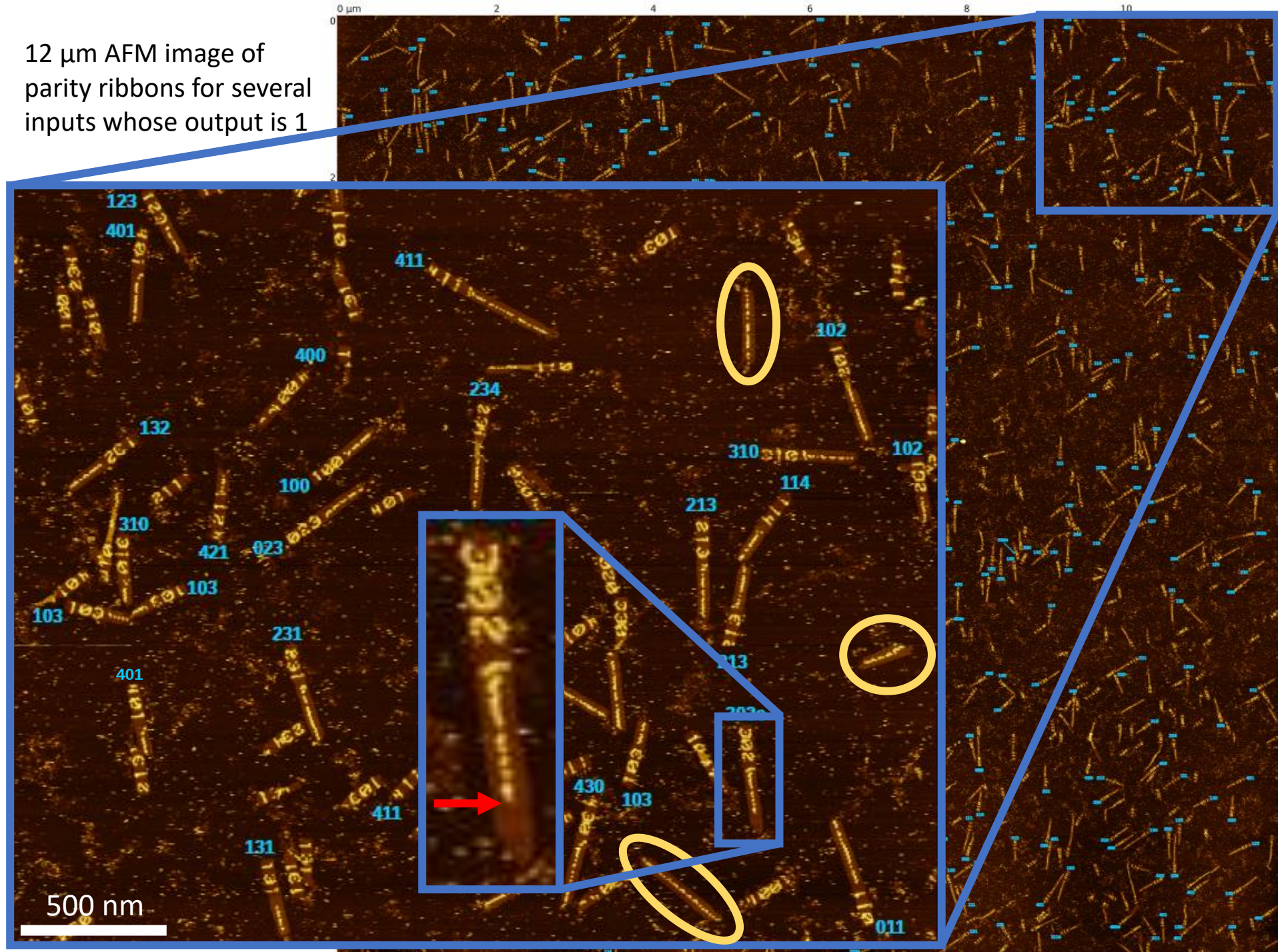
12 μm AFM image of
parity ribbons for several
inputs whose output is 1



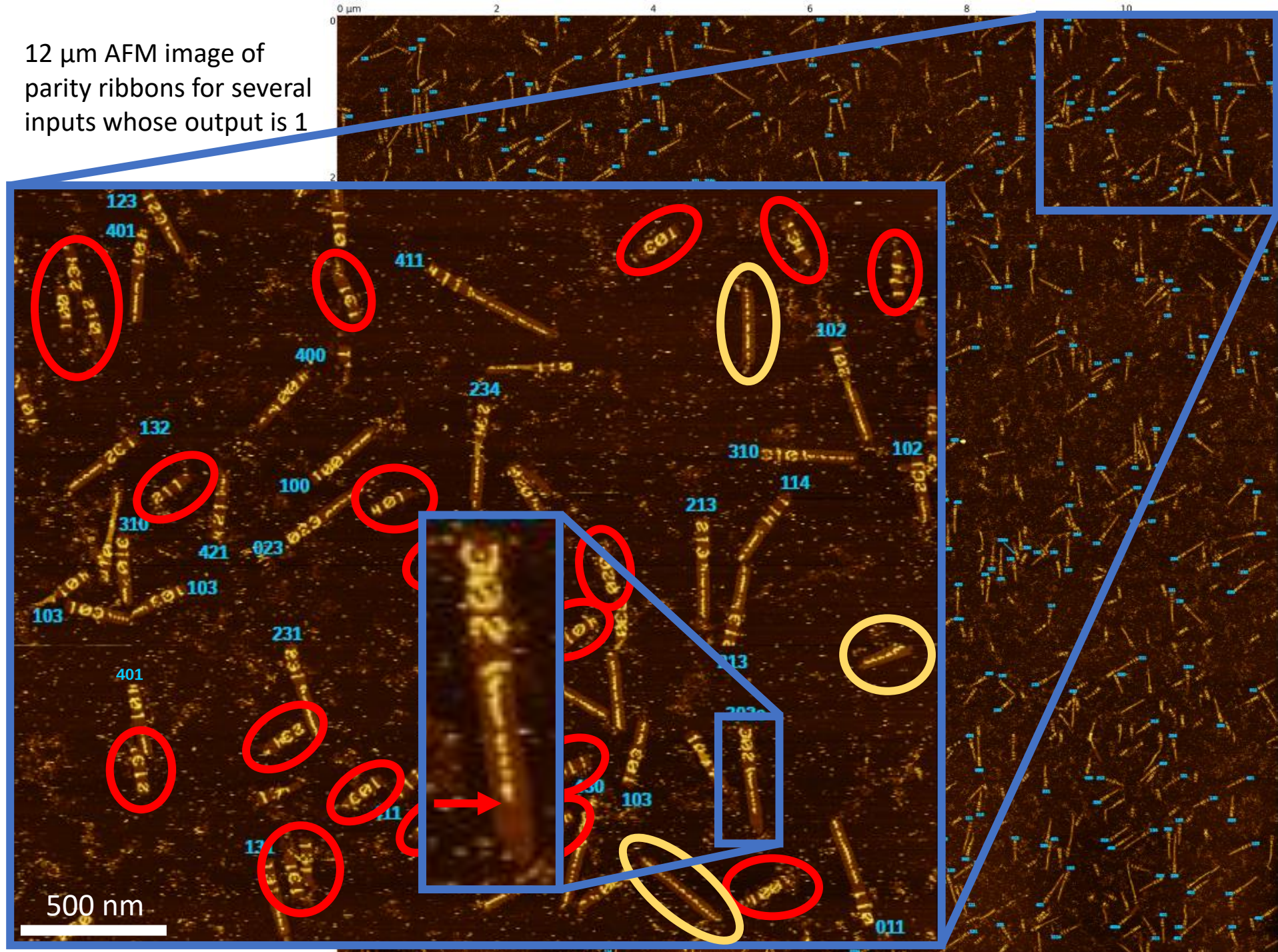
12 μm AFM image of
parity ribbons for several
inputs whose output is 1



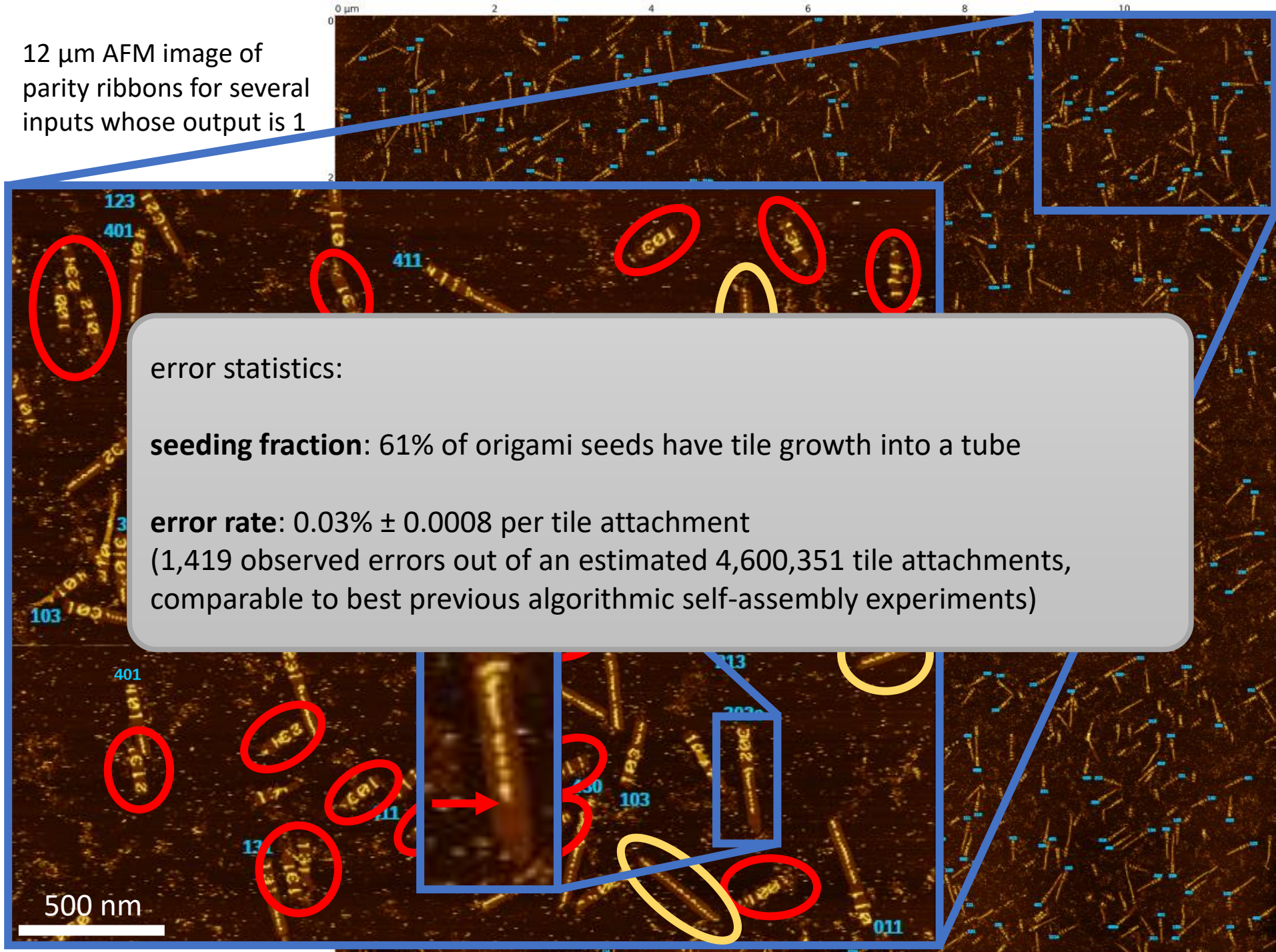
12 μm AFM image of
parity ribbons for several
inputs whose output is 1



12 μm AFM image of
parity ribbons for several
inputs whose output is 1



12 μm AFM image of
parity ribbons for several
inputs whose output is 1



What did we learn?

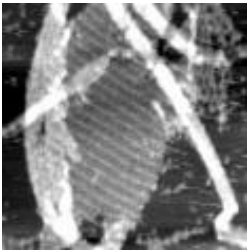
A small(ish) library of molecules can be reprogrammed to self-assemble reliably into many complex patterns, by processing information as they grow.

What did we learn?

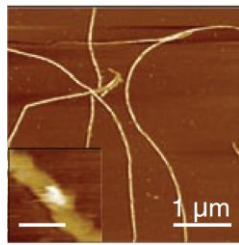
A small(ish) library of molecules can be reprogrammed to self-assemble reliably into many complex patterns, by processing information as they grow.

Contrasting with other self-assembly work:

more algorithmic control
than **periodic** self-assembly



2D tile lattices
(Winfree et al.,
Nature 1998)



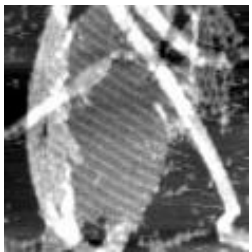
1D tile tubes
(Yin et al.,
Science 2008)

What did we learn?

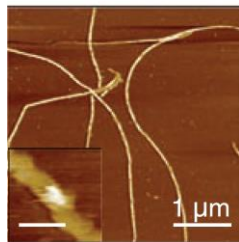
A small(ish) library of molecules can be reprogrammed to self-assemble reliably into many complex patterns, by processing information as they grow.

Contrasting with other self-assembly work:

more algorithmic control
than **periodic** self-assembly

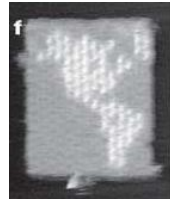


2D tile lattices
(Winfree et al.,
Nature 1998)



1D tile tubes
(Yin et al.,
Science 2008)

fewer types of DNA strands
required than **uniquely-
addressed** self-assembly



DNA origami
(Rothemund,
Nature 2006)



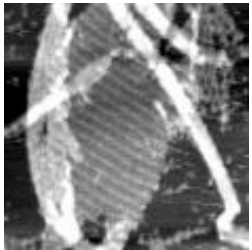
hard-coded tile
lattice (Wei et al.,
Nature 2012)

What did we learn?

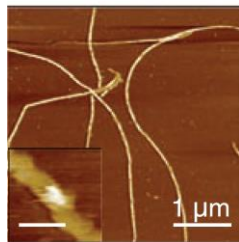
A small(ish) library of molecules can be reprogrammed to self-assemble reliably into many complex patterns, by processing information as they grow.

Contrasting with other self-assembly work:

more algorithmic control
than **periodic** self-assembly

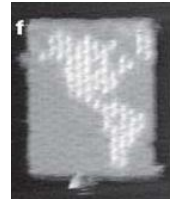


2D tile lattices
(Winfree et al.,
Nature 1998)



1D tile tubes
(Yin et al.,
Science 2008)

fewer types of DNA strands
required than **uniquely-**
addressed self-assembly



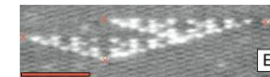
DNA origami
(Rothemund,
Nature 2006)



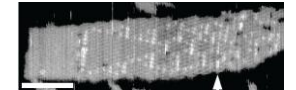
hard-coded tile
lattice (Wei et al.,
Nature 2012)

order of magnitude more tile
types available than previous
algorithmic self-assembly

double-crossover tile lattices



(Rothemund et al.,
PLoS Bio 2004)



(Fujibayashi et al.,
Nano Letters 2008)



(Barish et al., *PNAS*
2009)



(Evans, *Ph.D. thesis*
2014)

Next big challenge: Algorithmically control shape

We “drew” interesting patterns on a boring shape (infinite rectangle)



Can we run algorithms to
grow interesting shapes?

Next big challenge: Algorithmically control shape

We “drew” interesting patterns on a boring shape (infinite rectangle)



Can we run algorithms to
grow interesting shapes?

Theorem: There is a single set T of tile types, so that, for any finite shape S , from an appropriately chosen seed σ_S “encoding” S , T self-assembles S .

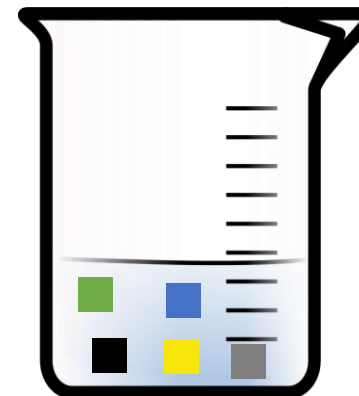
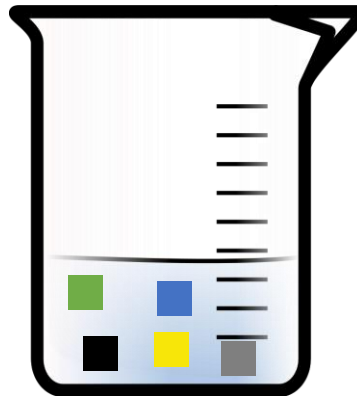
Next big challenge: Algorithmically control shape

We “drew” interesting patterns on a boring shape (infinite rectangle)



Can we run algorithms to
grow interesting shapes?

Theorem: There is a single set T of tile types, so that, for any finite shape S , from an appropriately chosen seed σ_S “encoding” S , T self-assembles S .



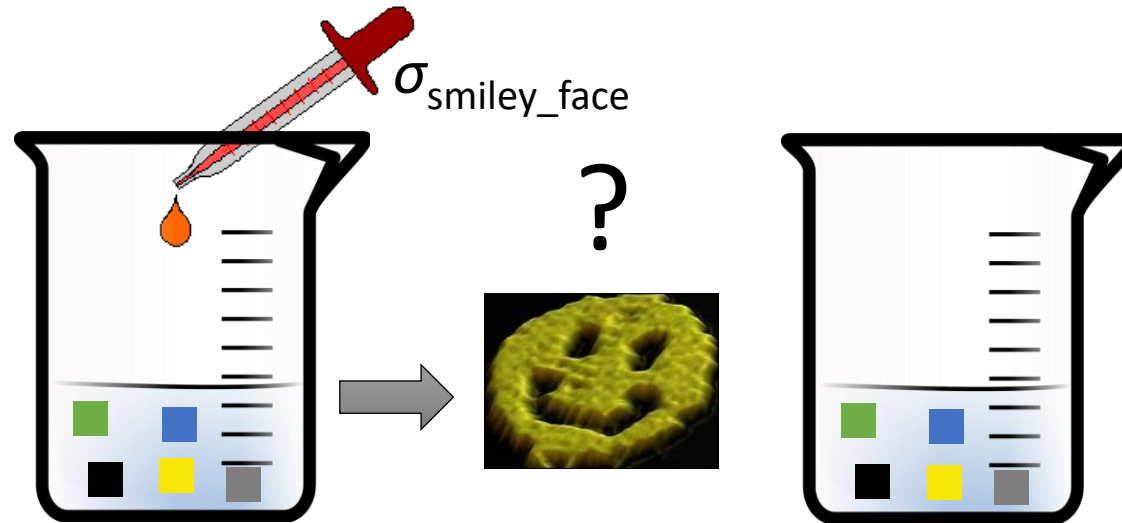
Next big challenge: Algorithmically control shape

We “drew” interesting patterns on a boring shape (infinite rectangle)



Can we run algorithms to
grow interesting shapes?

Theorem: There is a single set T of tile types, so that, for any finite shape S , from an appropriately chosen seed σ_S “encoding” S , T self-assembles S .



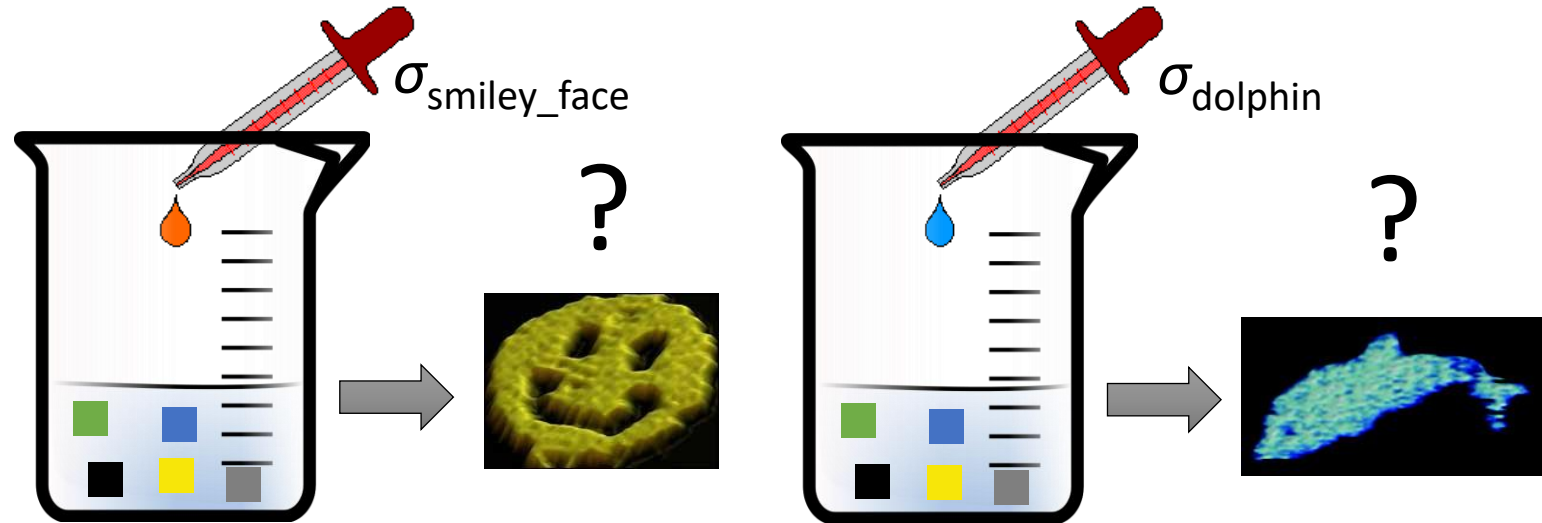
Next big challenge: Algorithmically control shape

We “drew” interesting patterns on a boring shape (infinite rectangle)



Can we run algorithms to
grow interesting shapes?

Theorem: There is a single set T of tile types, so that, for any finite shape S , from an appropriately chosen seed σ_S “encoding” S , T self-assembles S .



These tiles are **universally programmable** for building any shape.

Thank you!