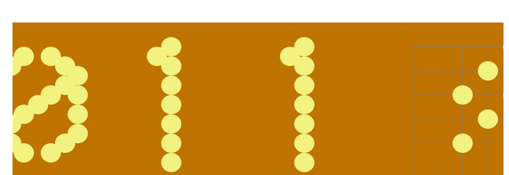
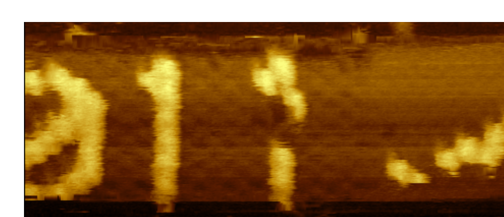
Diverse and robust molecular algorithms using reprogrammable DNA self-assembly

Damien Woods*, David Doty*, Cameron Myhrvold, Joy Hui,
Felix Zhou, Peng Yin, Erik Winfree

Maynooth University National University of Ireland Maynooth
Hamilton Institute
Caltech
Inria
UC Davis
Harvard

Theorem: Let *T* be a tile assembly system …

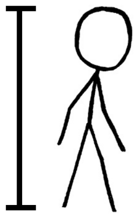


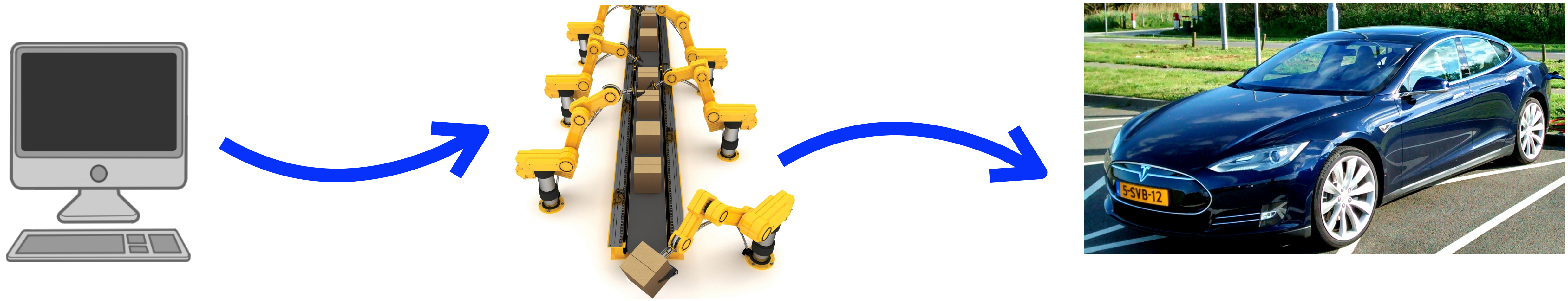


Building stuff

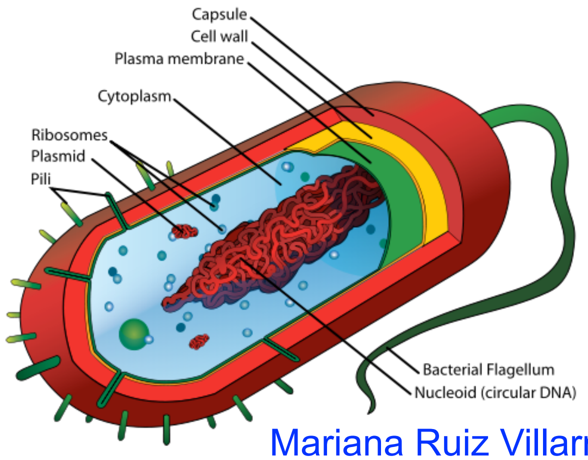
Ljubljana Marshes Wheel. 5k years old

Newgrange, Ireland. 5.2k years old

- **Building stuff by hand**: use tools! Great for scale of $10^{+/-2}$ x 
- **Building tools that build stuff**: specify target object with a computer program that then controls the manufacturing process

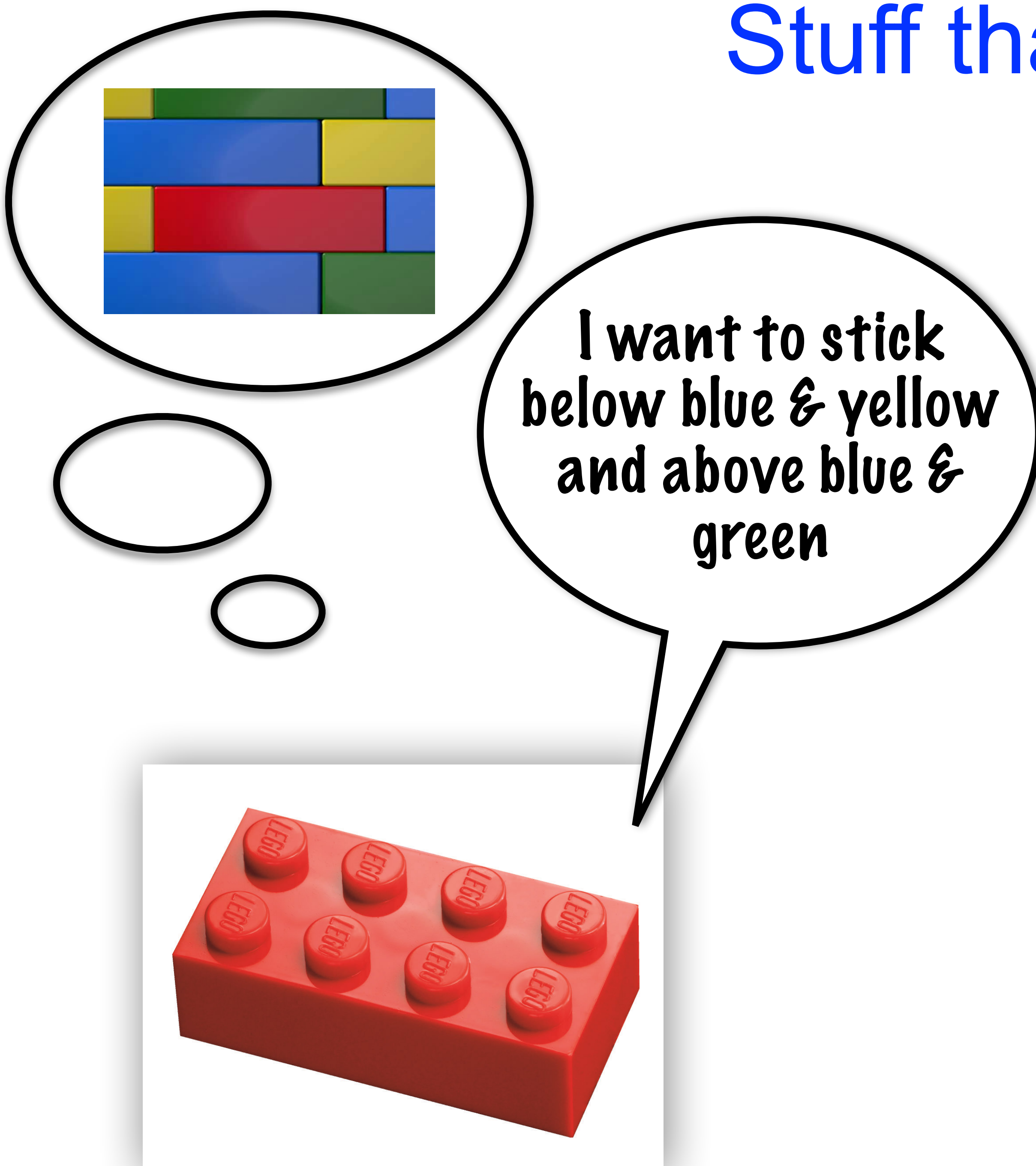


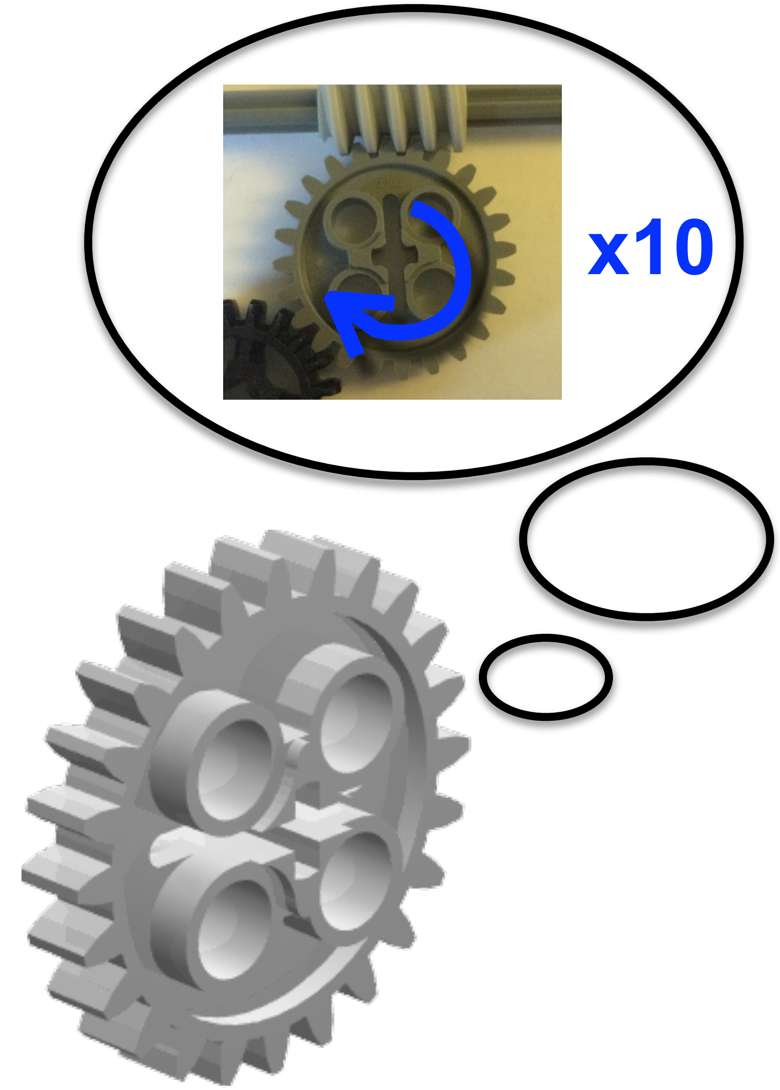
- Programming **stuff to build itself**: for building stuff in small wet places where our hands or tools can't reach



Mariana Ruiz Villarreal

Stuff that builds itself



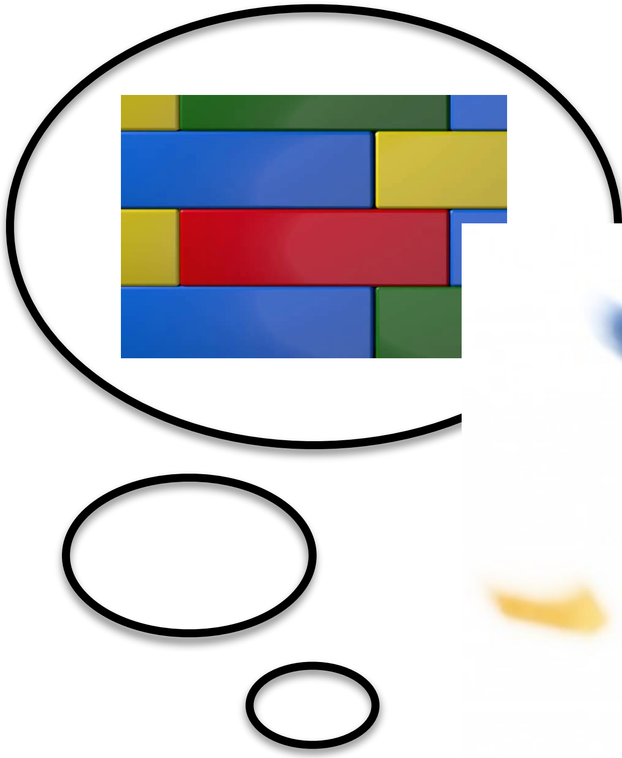


- Today you'll hear about self-assembling molecules that compute as they build themselves

Stuff that builds itself

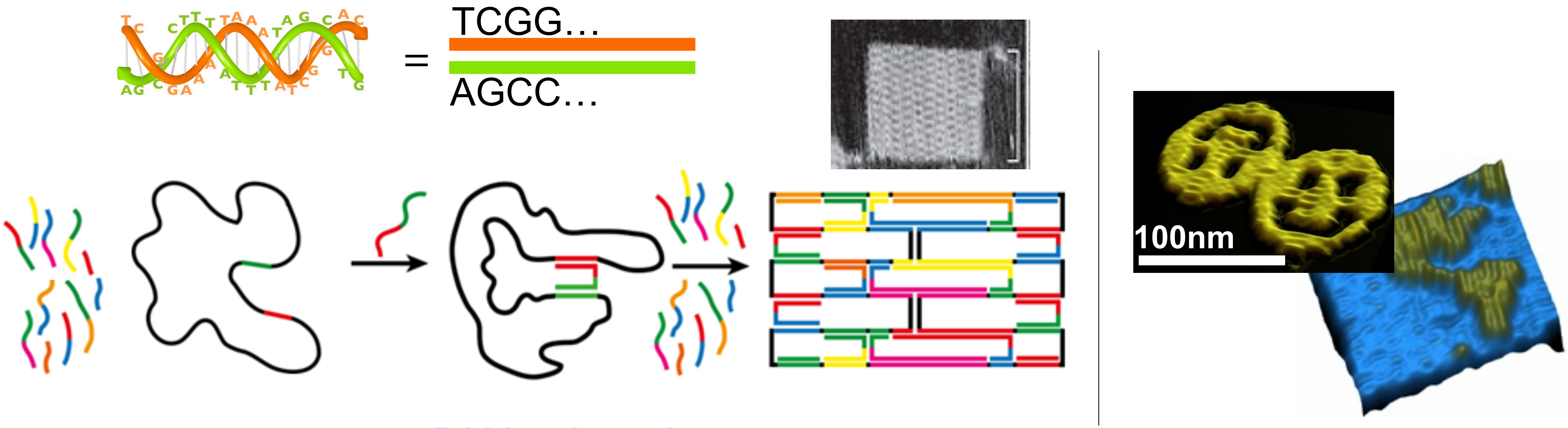
- Today you'll hear about self-assembling molecules that compute as they build themselves

Stuff that builds itself



- Today you'll hear about self-assembling molecules that compute as they build themselves

Background: DNA nanostructures



DNA origami

Rothemund. 2006 Nature

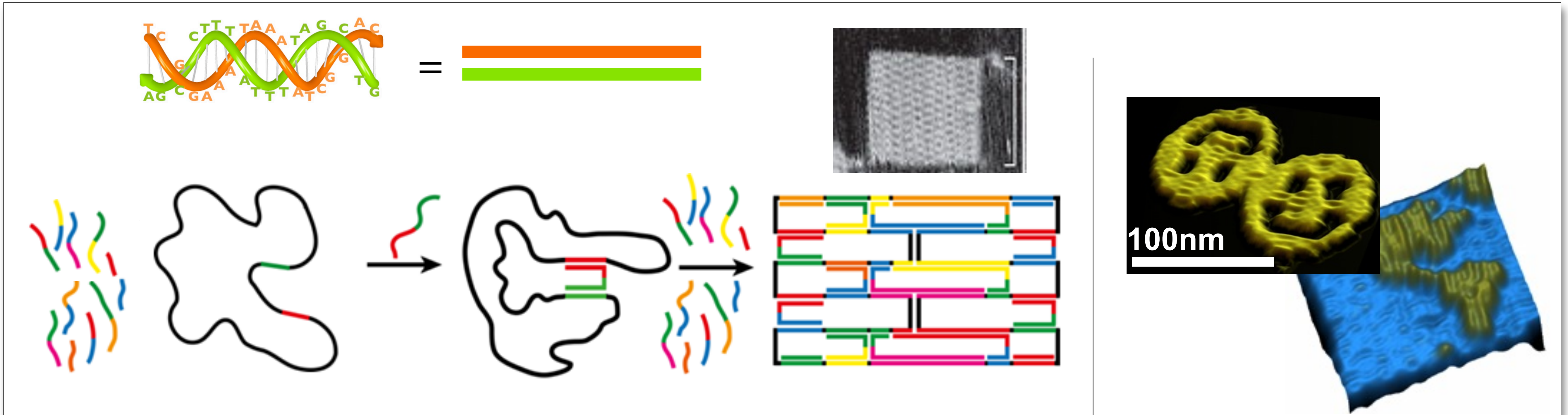
Example DNA nanostructure: DNA origami

Movie by Shawn Douglas

Example DNA nanostructure: DNA origami

Movie by Shawn Douglas

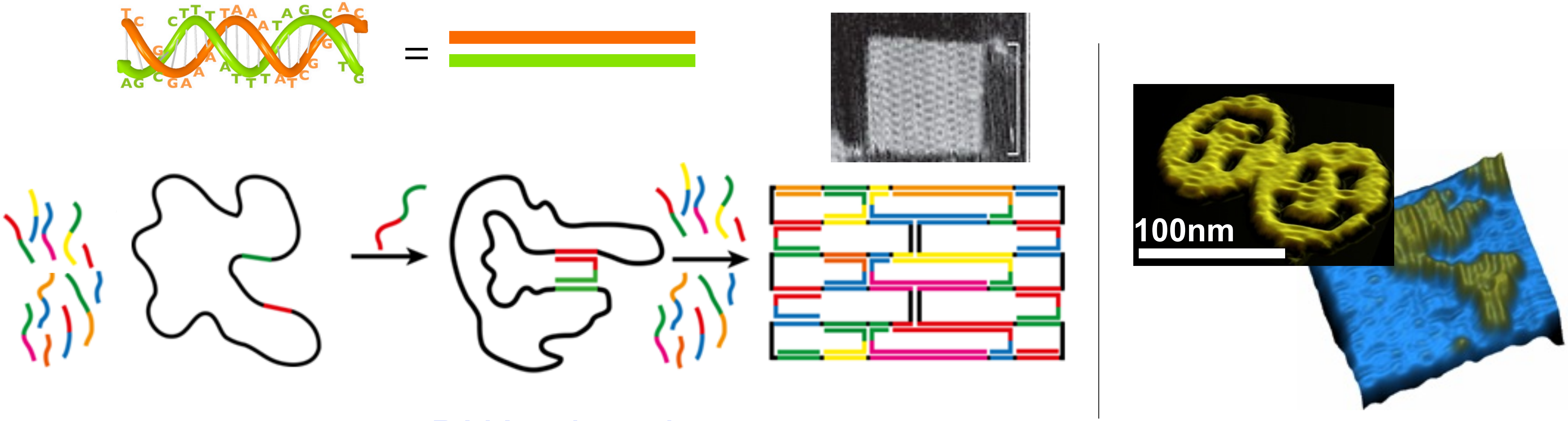
Background: DNA nanostructures



DNA origami

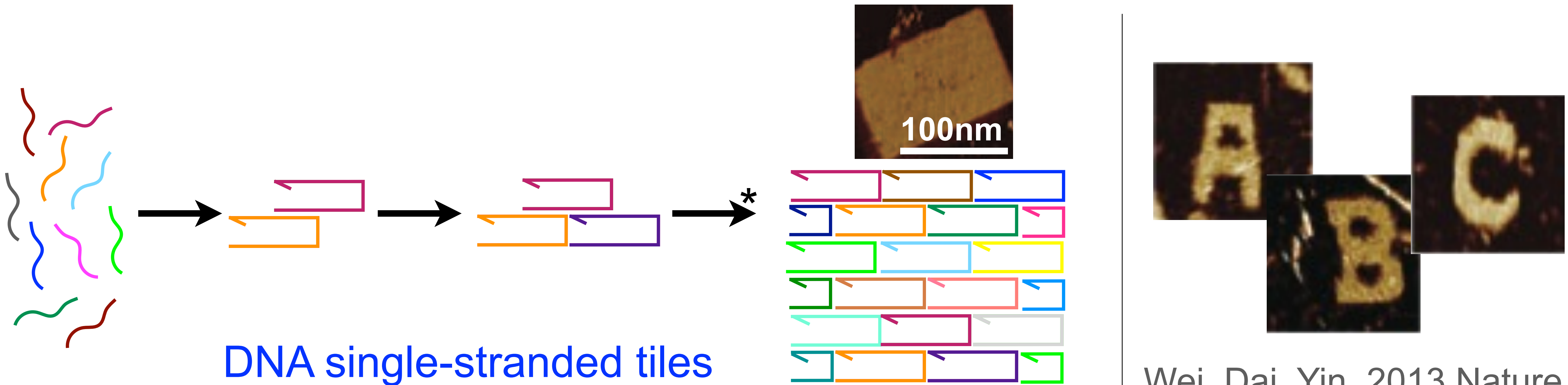
Rothemund. 2006 Nature

Background: DNA nanostructures



DNA origami

Rothemund. 2006 Nature

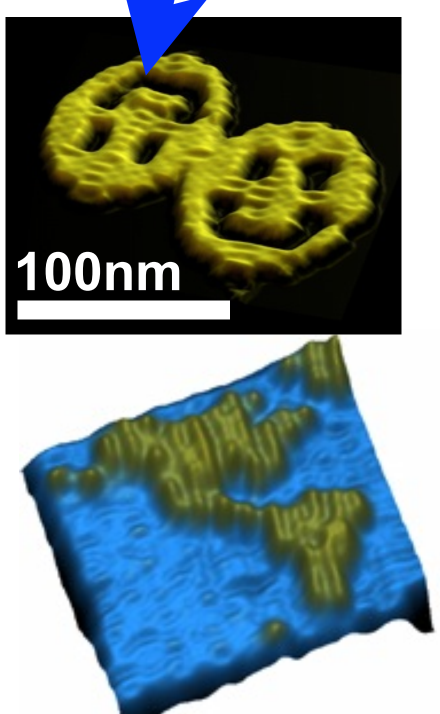


DNA single-stranded tiles

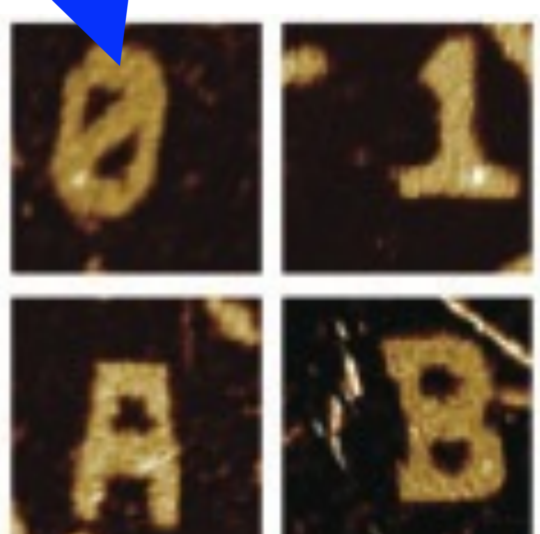
Wei, Dai, Yin. 2013 Nature

Nanostructure design via self-assembly

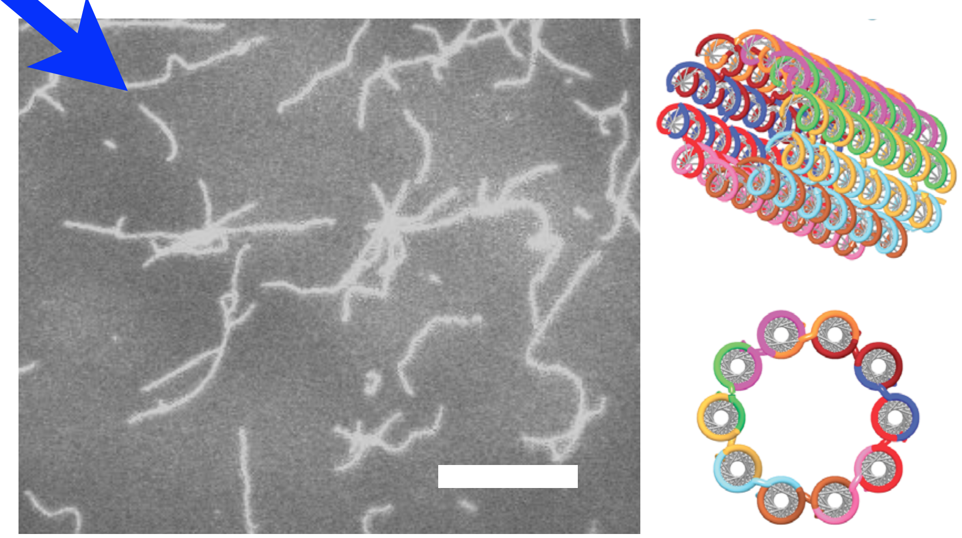
We tell the molecules **exactly** where to go



Rothemund 2006 Nature



Wei, Dai, Yin. 2013 Nature

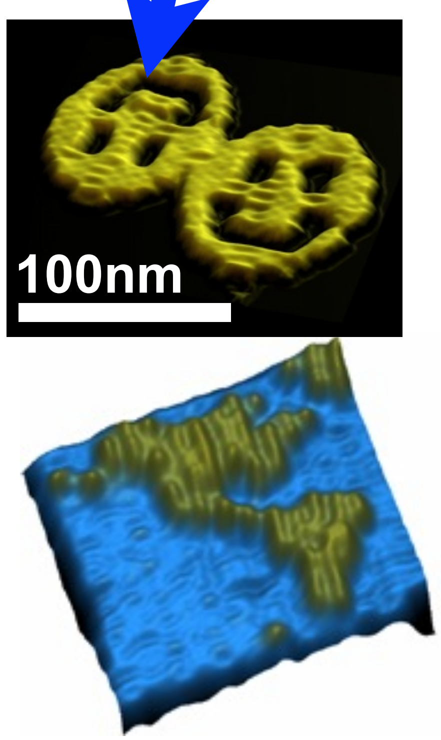


Yin et al 2008 Science

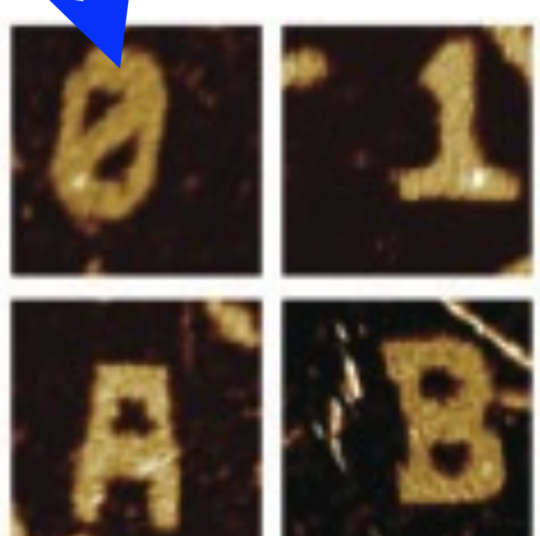
Nanostructure design via self-assembly

We tell the molecules **exactly** where to go

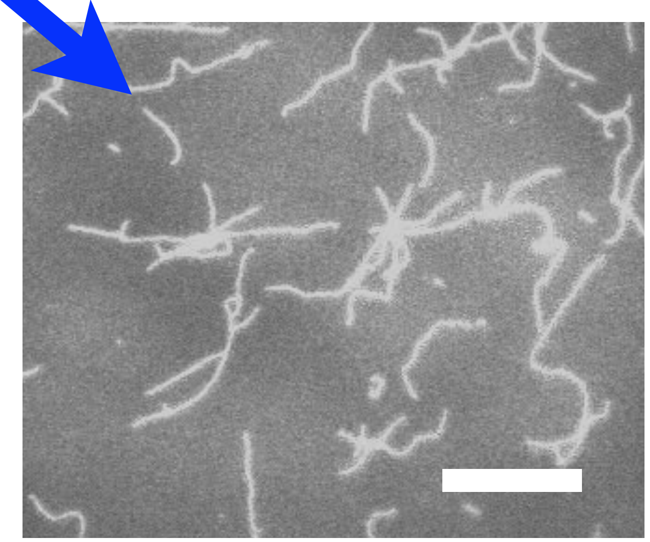
Can we have **smarter** molecules that decide where to go for themselves?

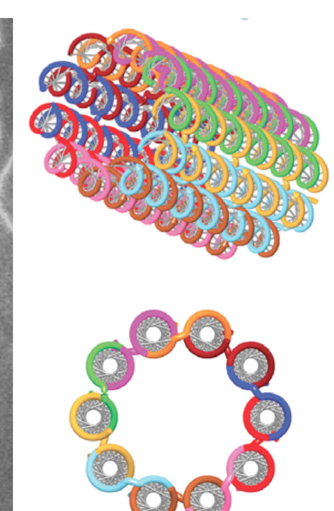


Rothemund 2006 Nature



Wei, Dai, Yin. 2013 Nature

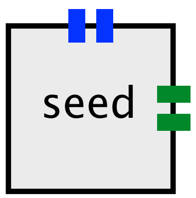


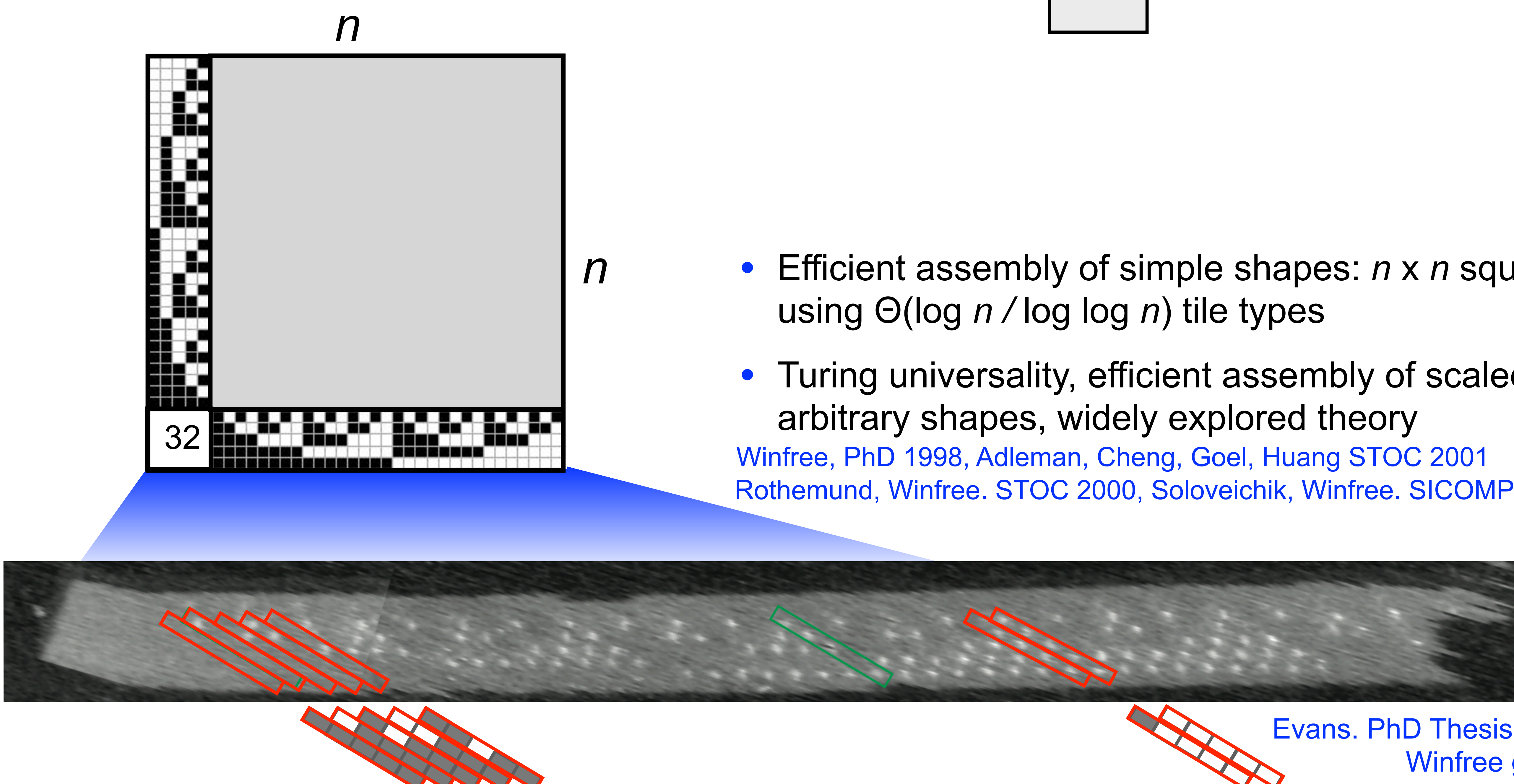


Yin et al 2008 Science

Algorithmic self-assembly: some previous work

- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





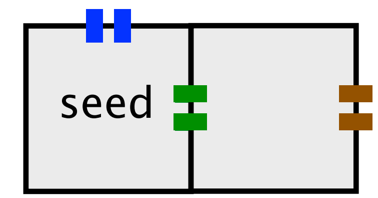
- Efficient assembly of simple shapes: *n* x *n* squares using Θ(log *n* / log log *n*) tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

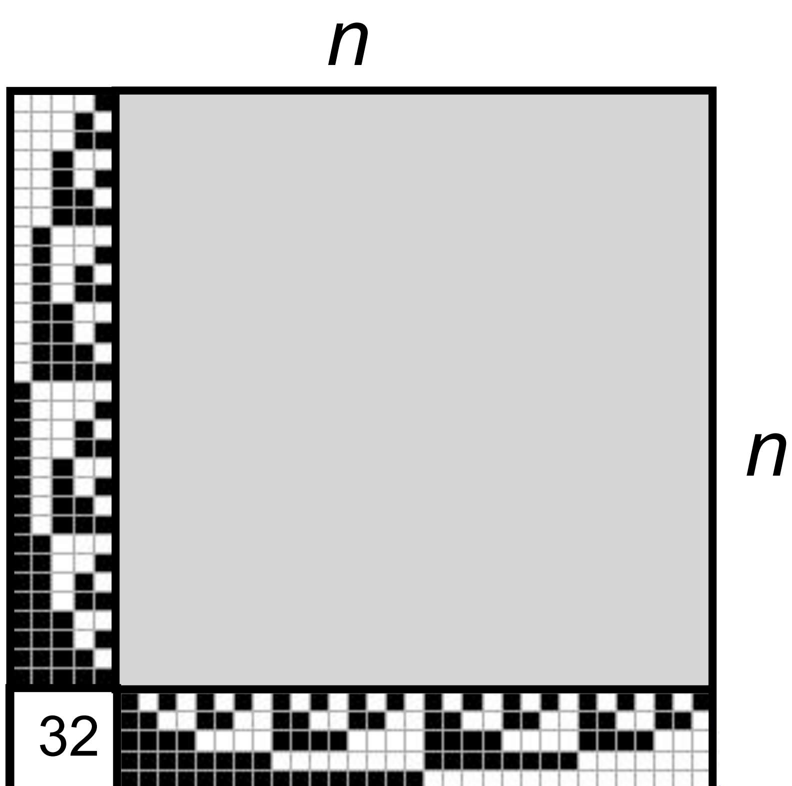
Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly: some previous work

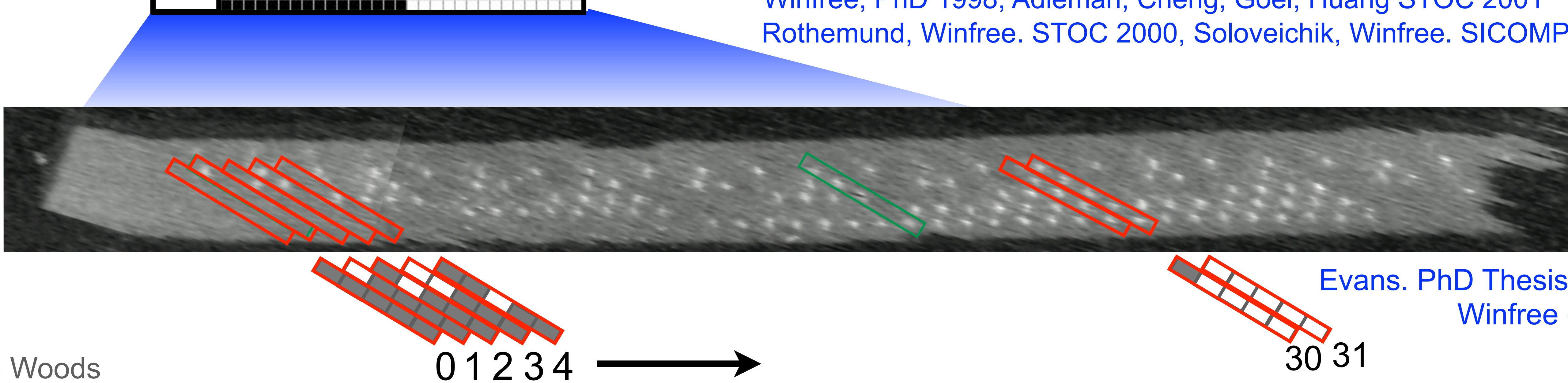
- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





- Efficient assembly of simple shapes: *n* x *n* squares using Θ(log *n* / log log *n*) tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

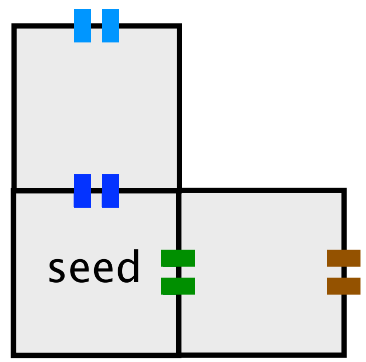
Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

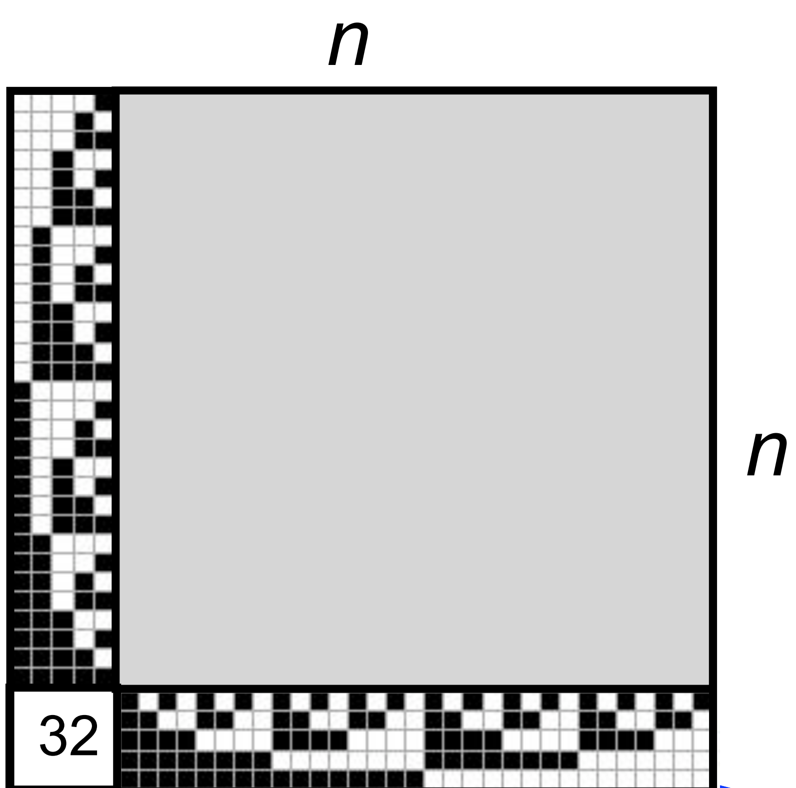


Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly: some previous work

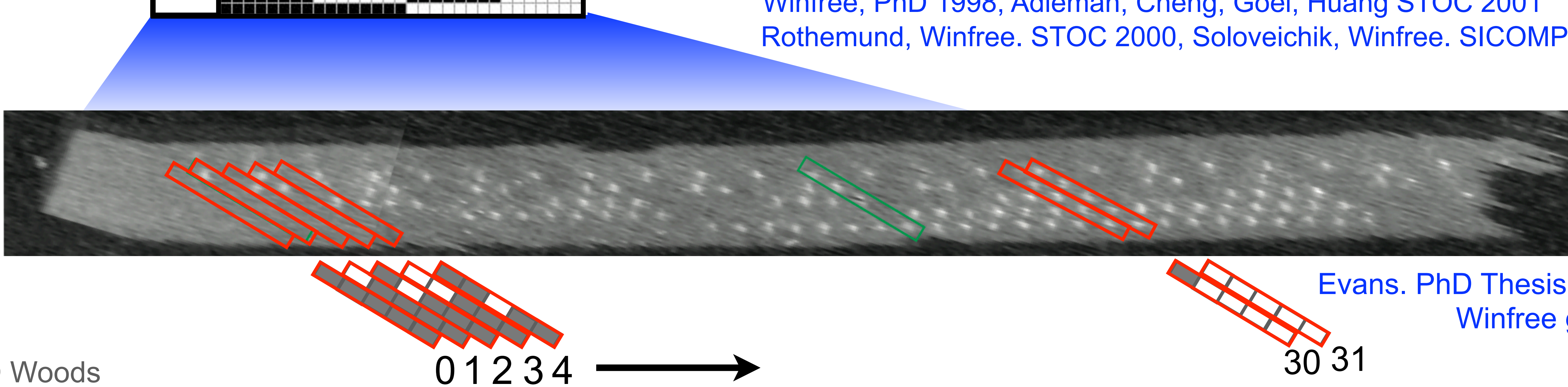
- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





- Efficient assembly of simple shapes: *n* x *n* squares using Θ(log *n* / log log *n*) tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

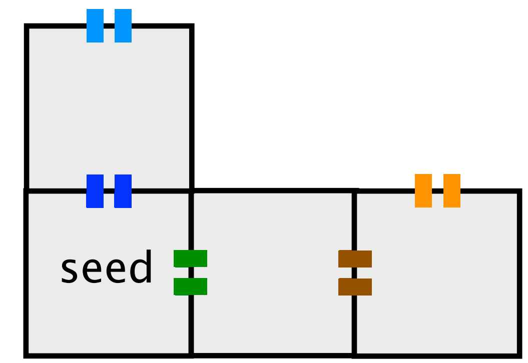
Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

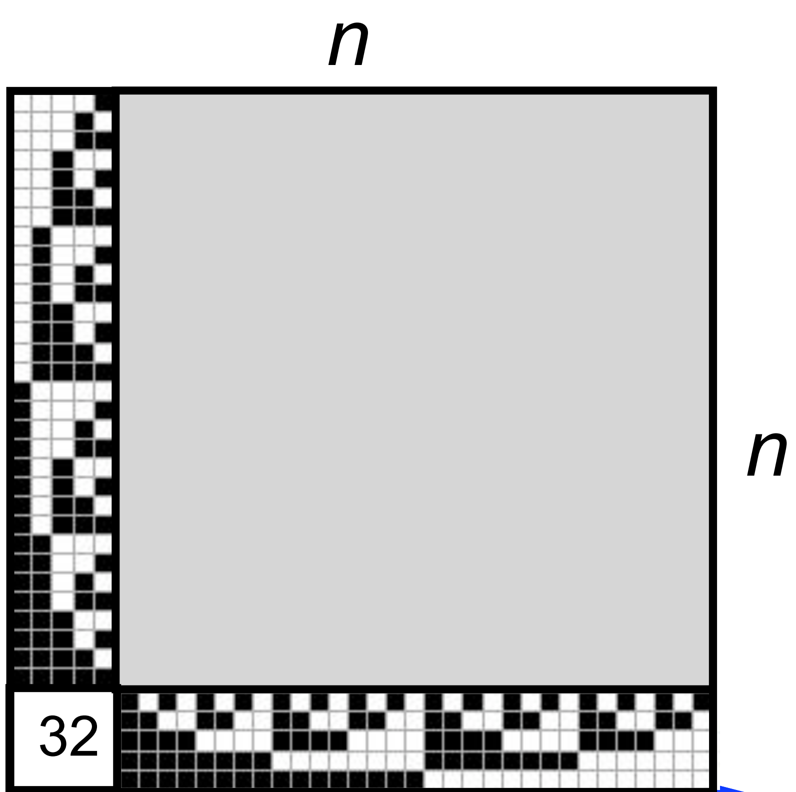


Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly: some previous work

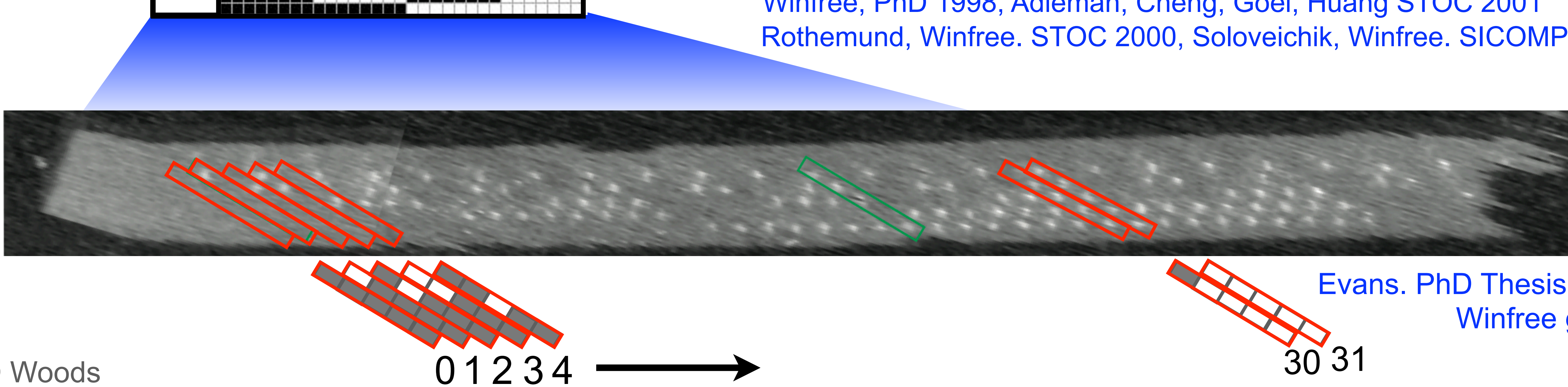
- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





- Efficient assembly of simple shapes: n x n squares using $\Theta(\log n / \log\log n)$ tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

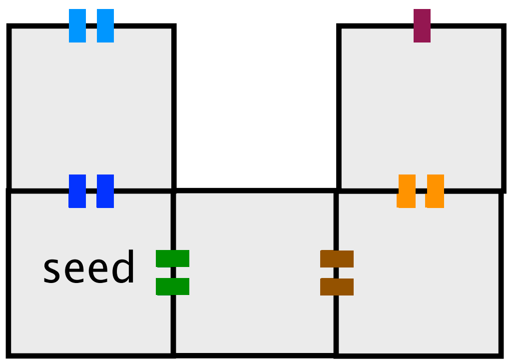
Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

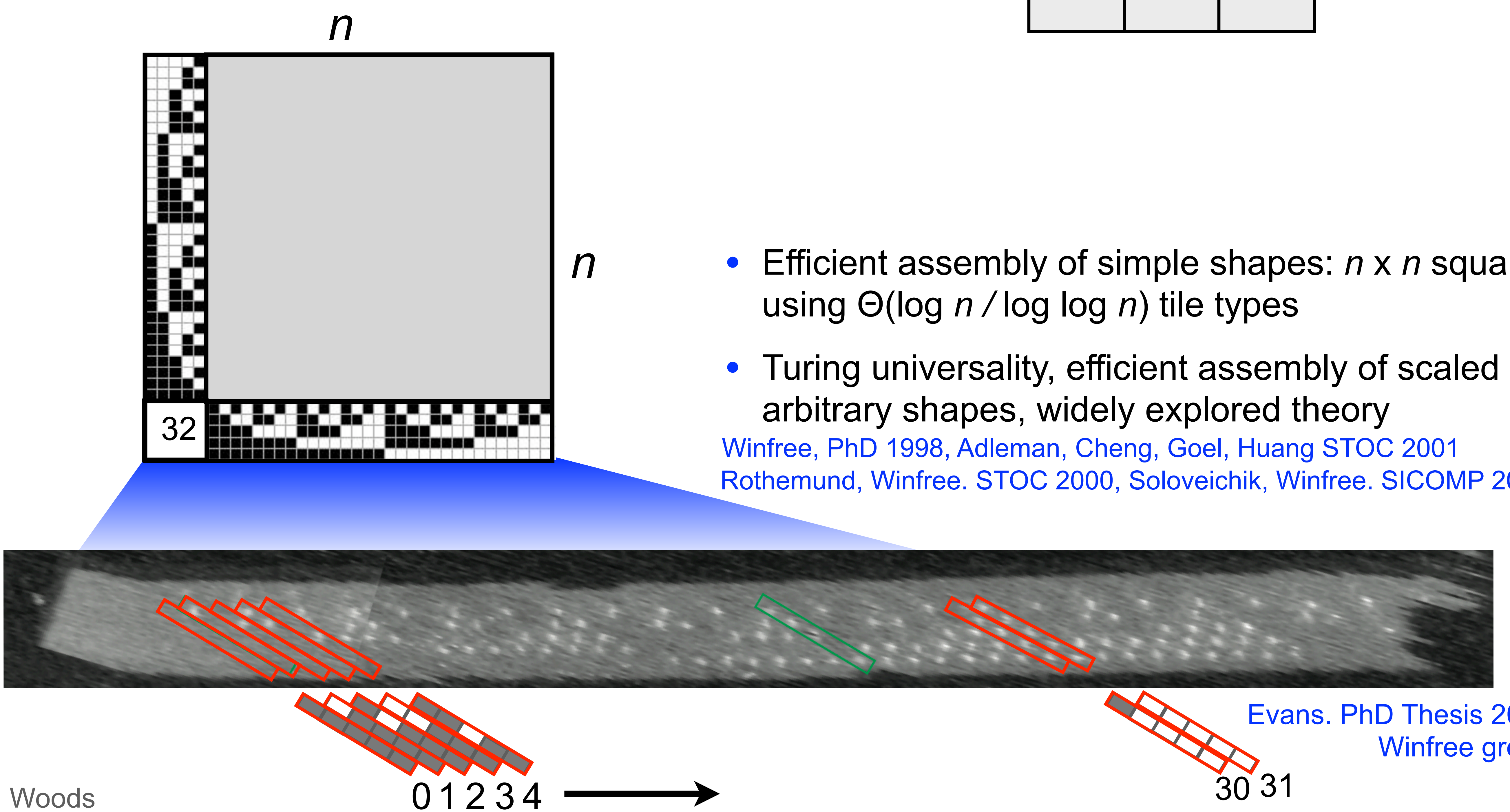


Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly: some previous work

- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





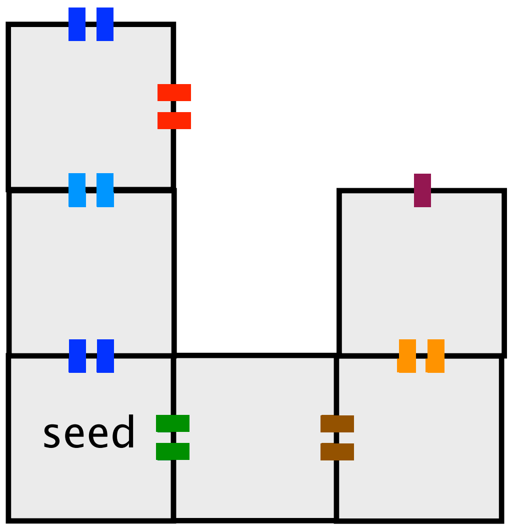
- Efficient assembly of simple shapes: *n* x *n* squares using $\Theta(\log n / \log\log n)$ tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

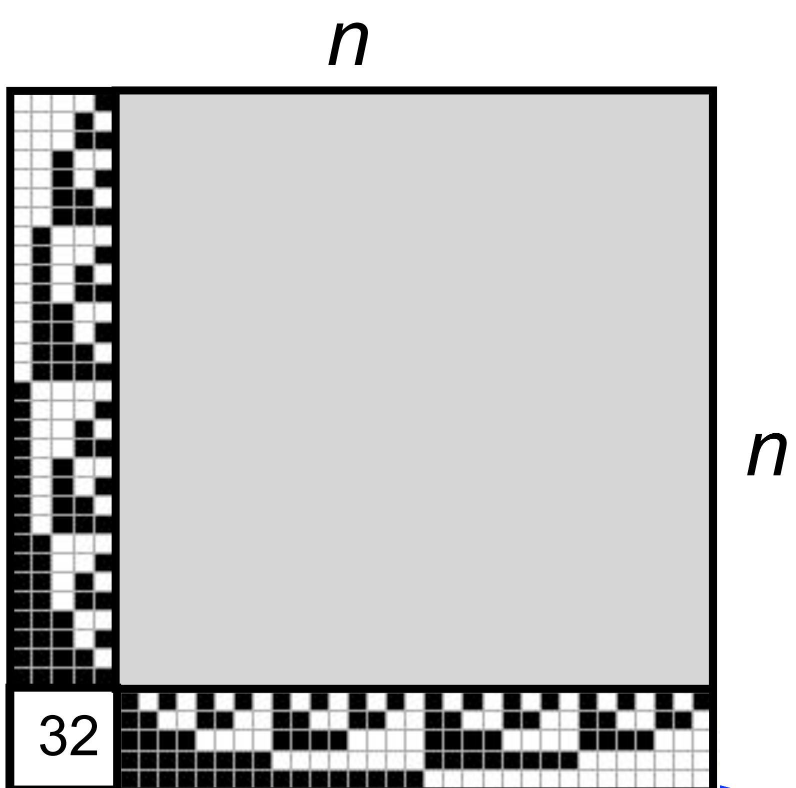
Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly: some previous work

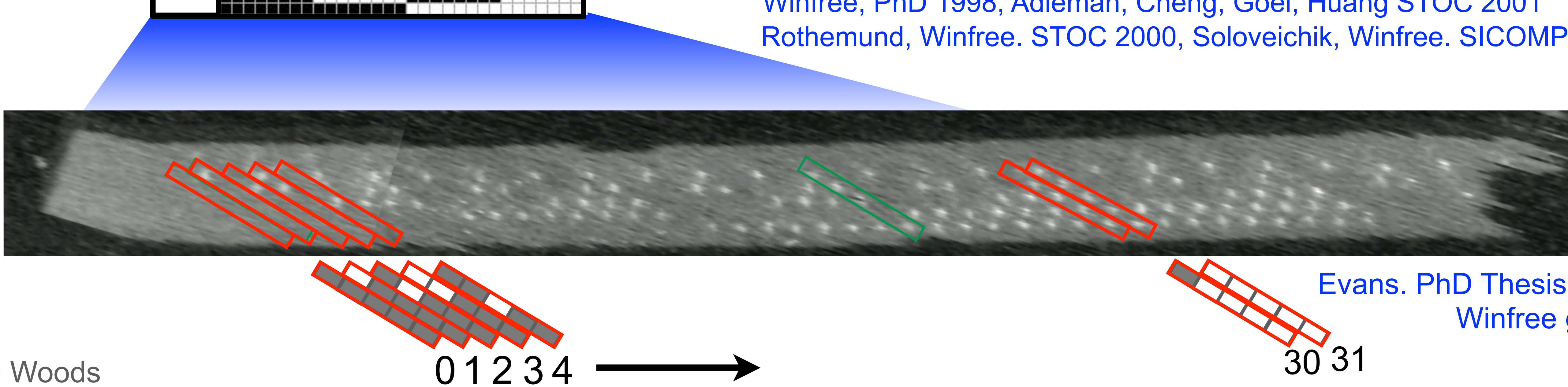
- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





- Efficient assembly of simple shapes: n x n squares using $\Theta(\log n / \log \log n)$ tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

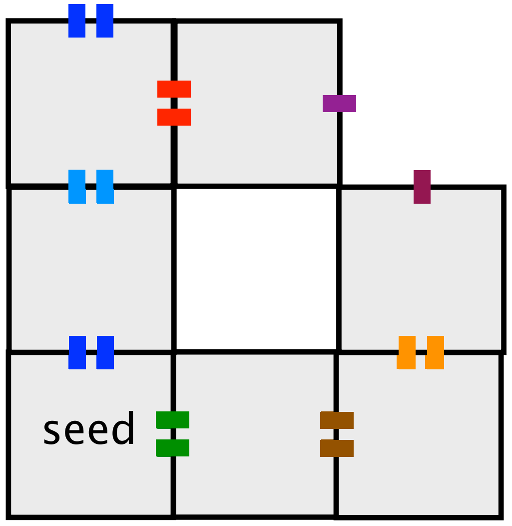
Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

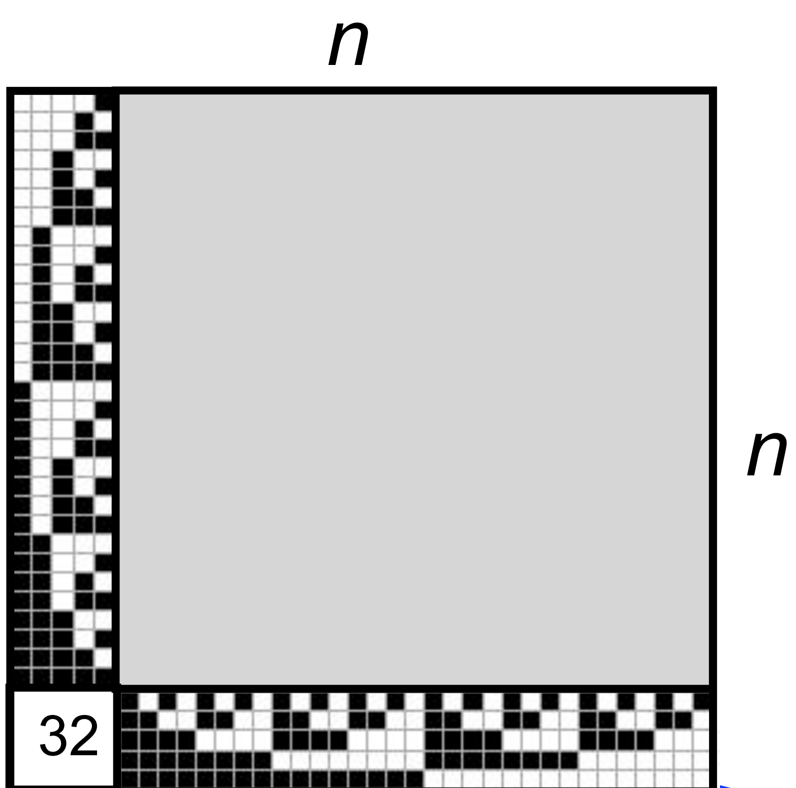


Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly: some previous work

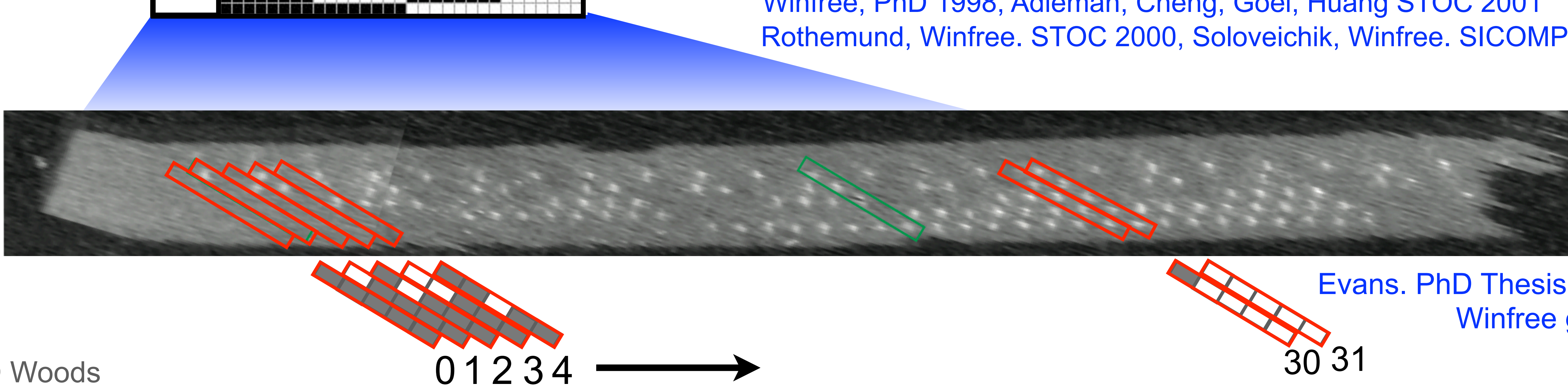
- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





- Efficient assembly of simple shapes: n x n squares using $\Theta(\log n / \log\log n)$ tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

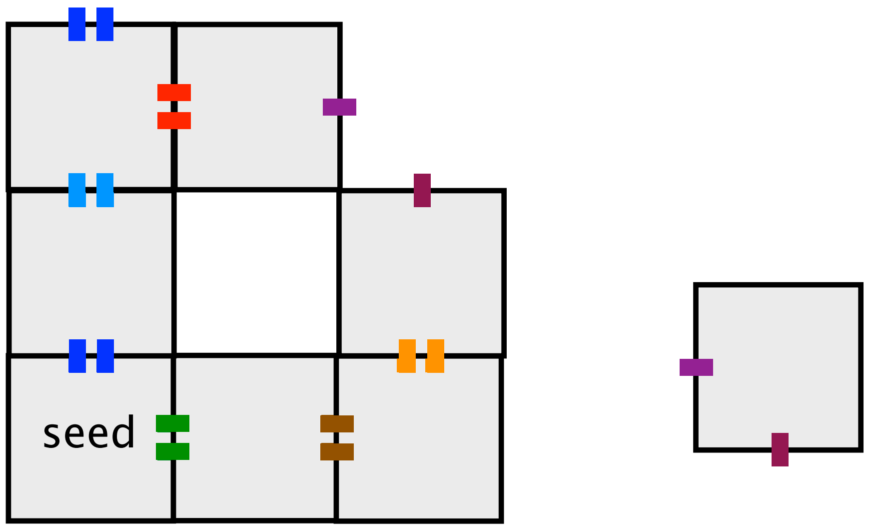
Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

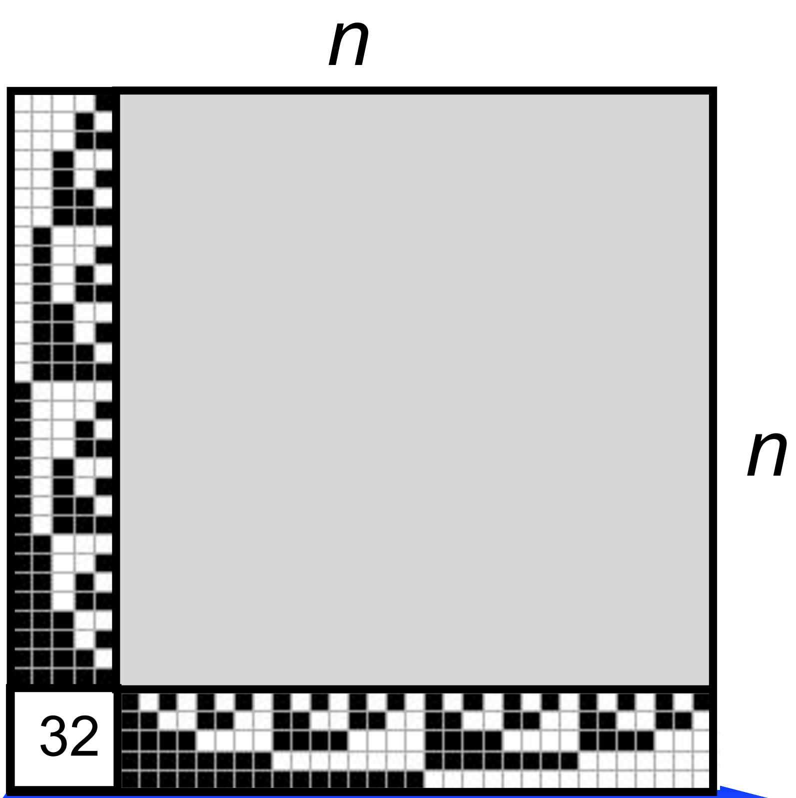


Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly: some previous work

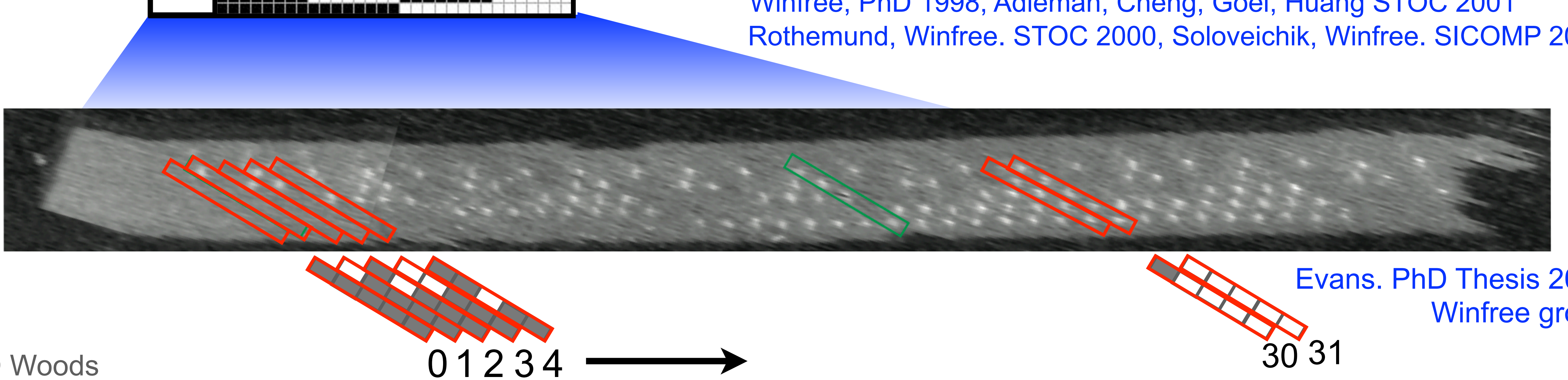
- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





- Efficient assembly of simple shapes: n x n squares using $\Theta(\log n / \log \log n)$ tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

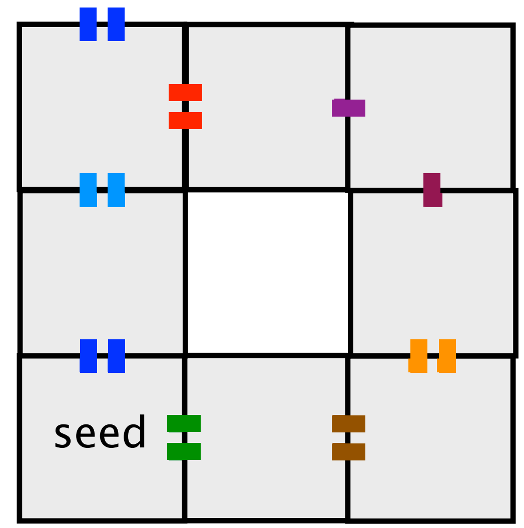
Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

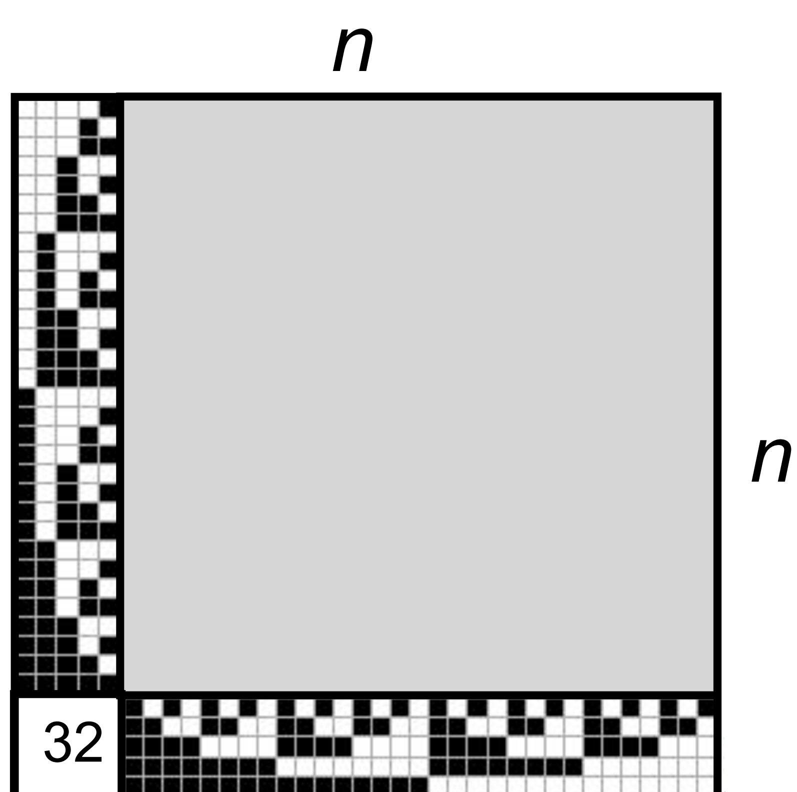


Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly: some previous work

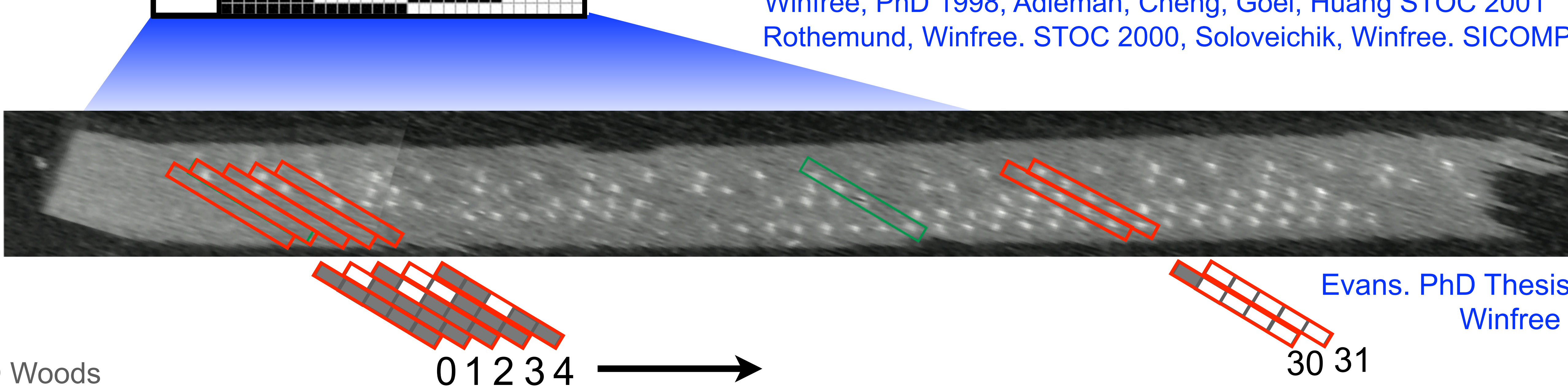
- Square **tiles** (finite set of tile types, unlimited supply of each, non-rotatable)
- Sides have a **glue** (colour) and **strength** (0,1,2,3,…)
- System has a **temperature** (e.g. 2)
- **Simple local binding rule**: A tile sticks to an assembly if enough of its glues match so that the sum of the strengths of the matching glues is at least the temperature





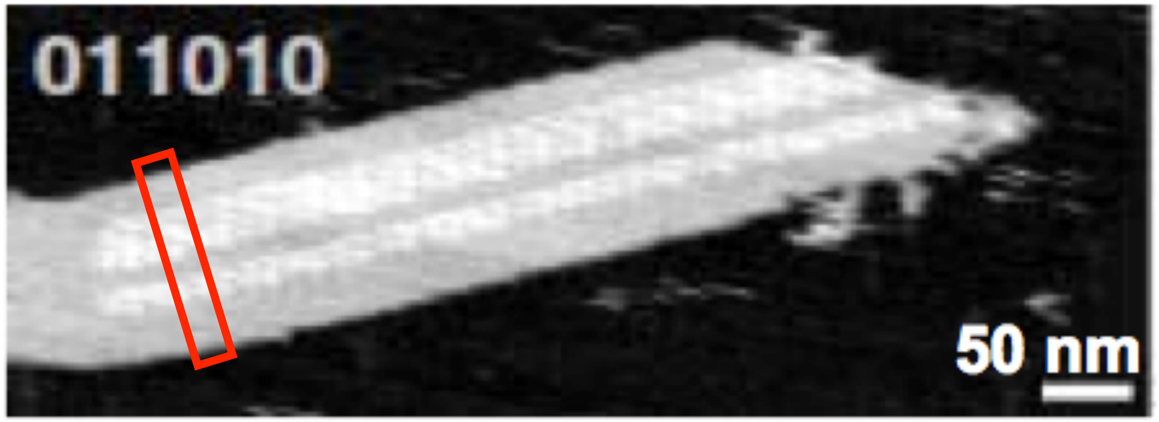
- Efficient assembly of simple shapes: *n* x *n* squares using Θ(log *n* / log log *n*) tile types
- Turing universality, efficient assembly of scaled arbitrary shapes, widely explored theory

Winfree, PhD 1998, Adleman, Cheng, Goel, Huang STOC 2001
Rothemund, Winfree. STOC 2000, Soloveichik, Winfree. SICOMP 2007

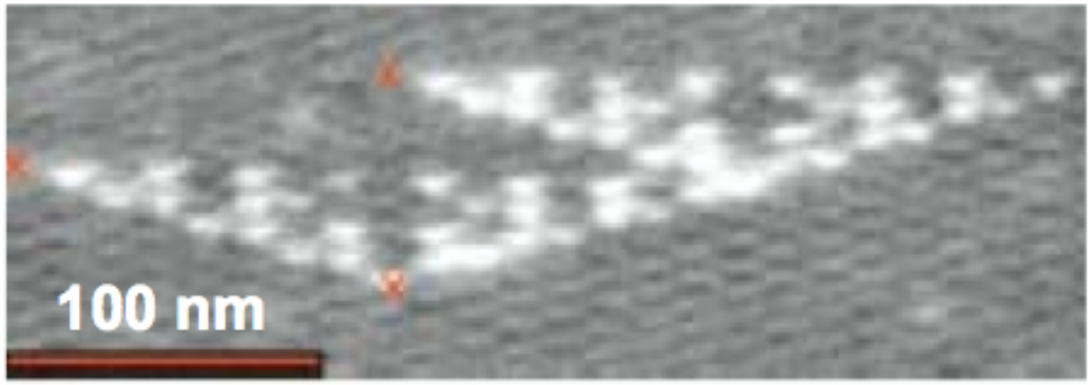


Evans. PhD Thesis 2014
Winfree group

Algorithmic self-assembly experiments: previous work

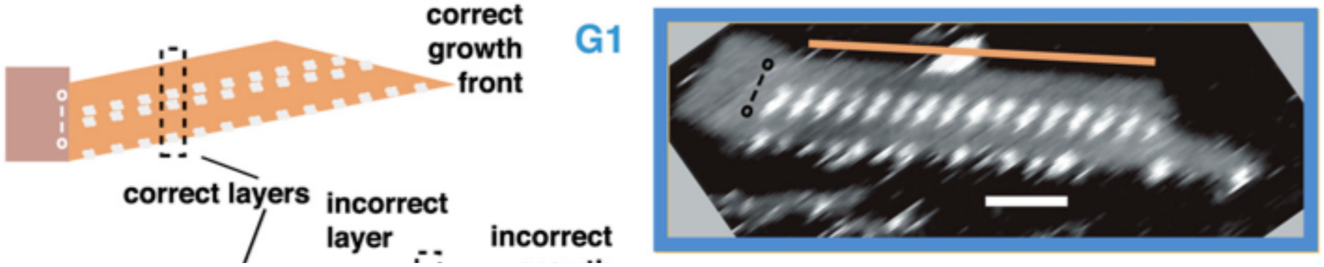


Bit Copying. Barish et al.2009

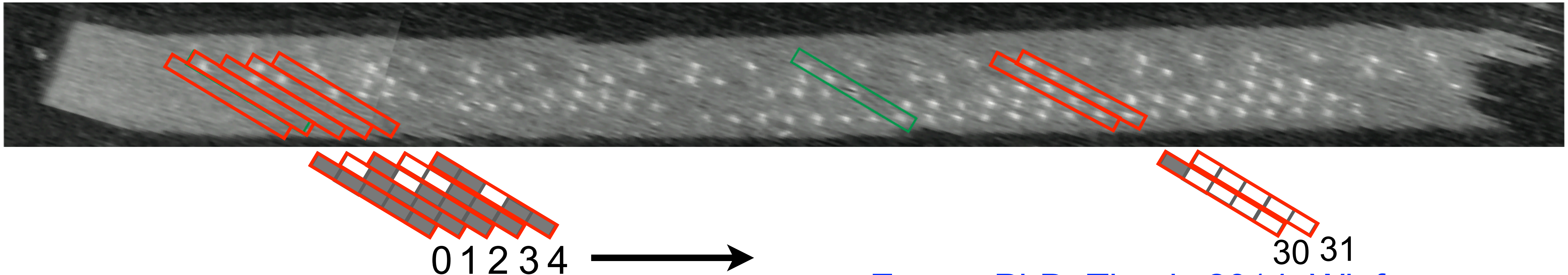


Sierpinski Triangles. Rothemund, Papadakis, Winfree. 2004

Counter. Barish et al. 2009



Copying & replication Schulman, Yurke, Winfree. PNAS. 2012



Evans. PhD. Thesis 2014. Winfree group.

Copying, Sierpinsky, binary counting to 31:
Can we run more algorithms?

Structure of talk

Copying, Sierpinski, binary counting to 31,
can we run more algorithms?

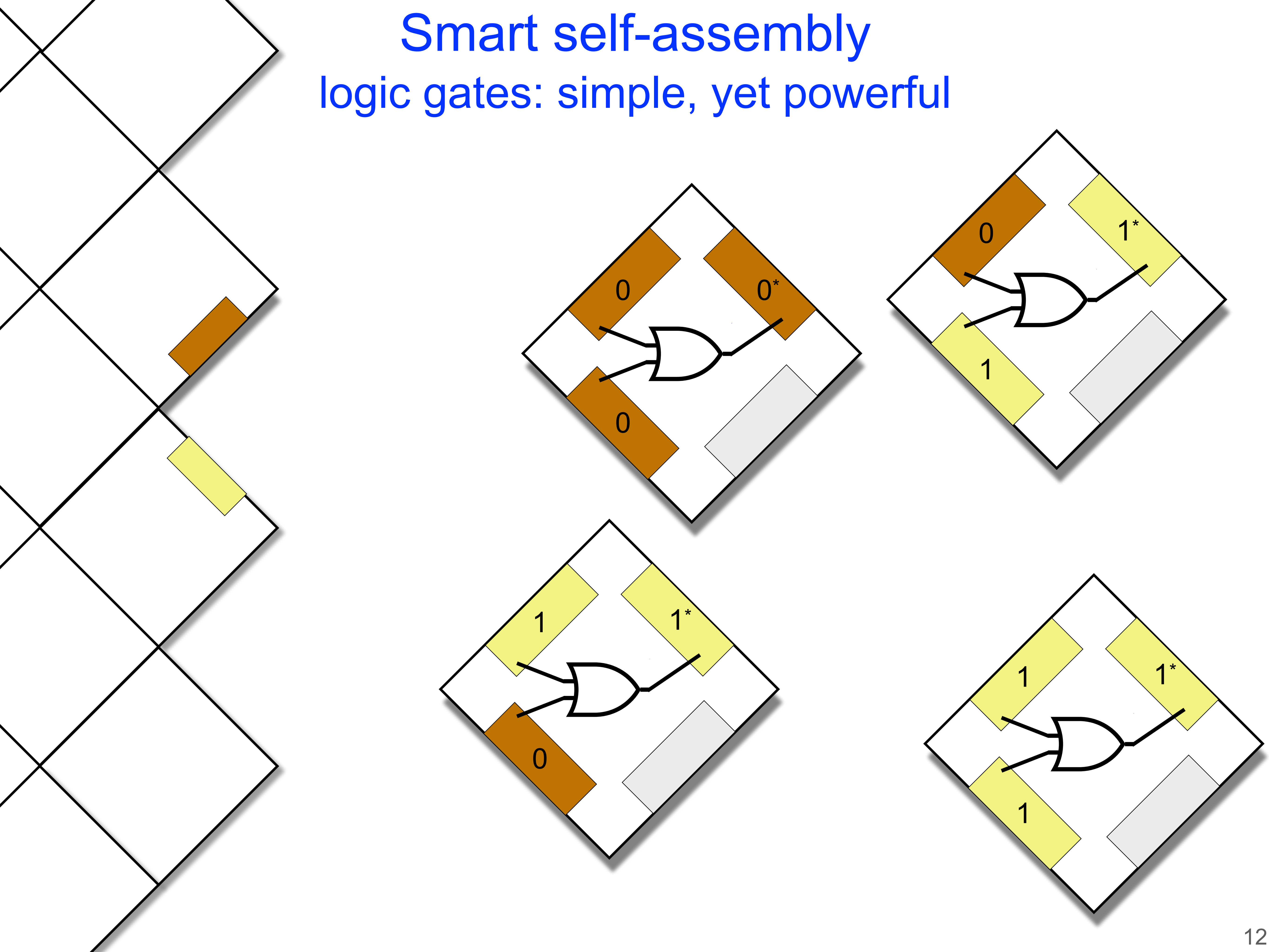
Theoretical circuit model

How it works: design and implementation

Experimental results

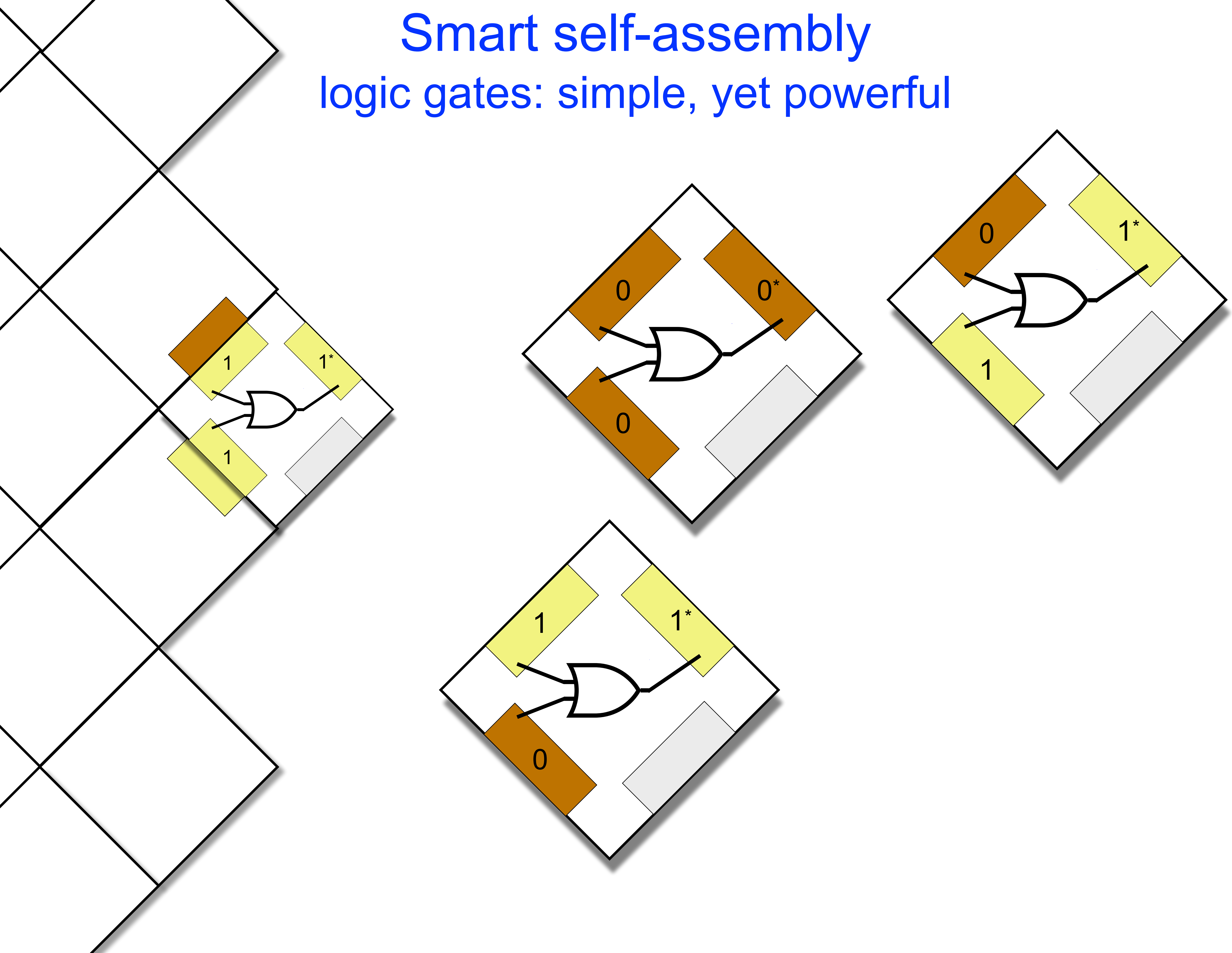
Smart self-assembly

logic gates: simple, yet powerful



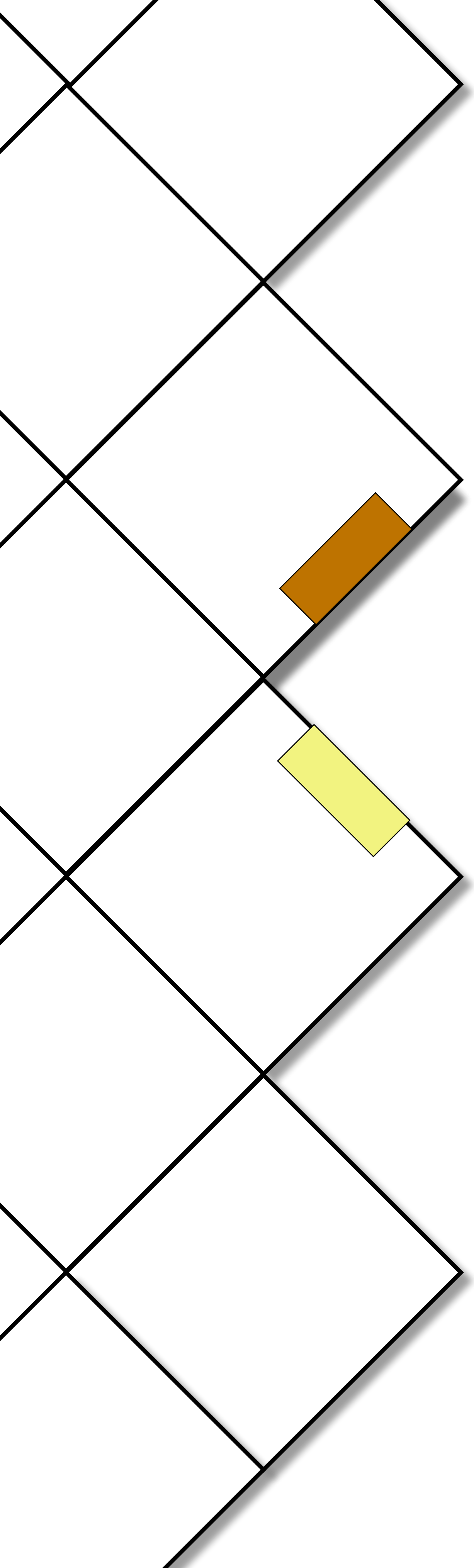
Smart self-assembly

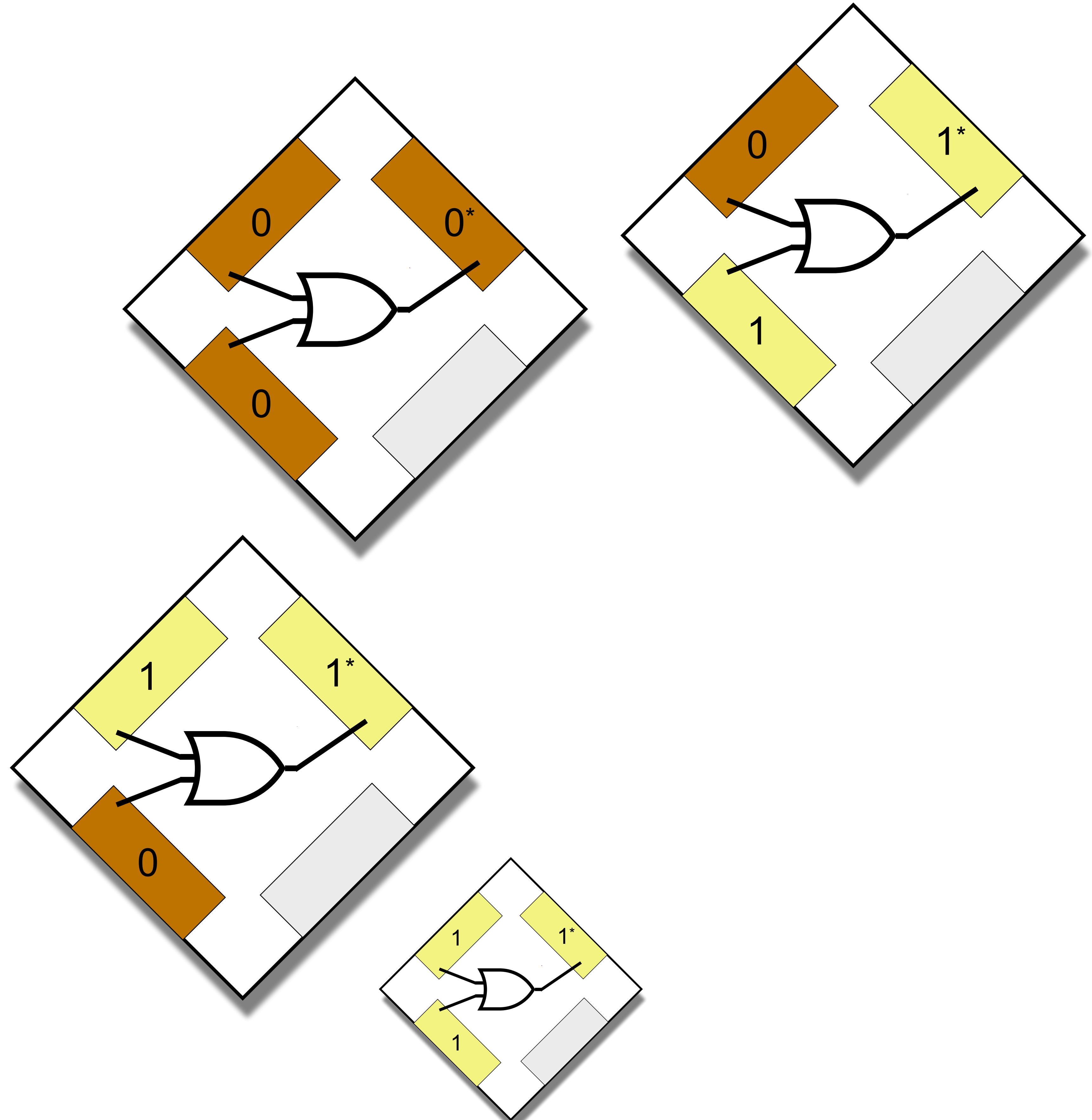
logic gates: simple, yet powerful



Smart self-assembly

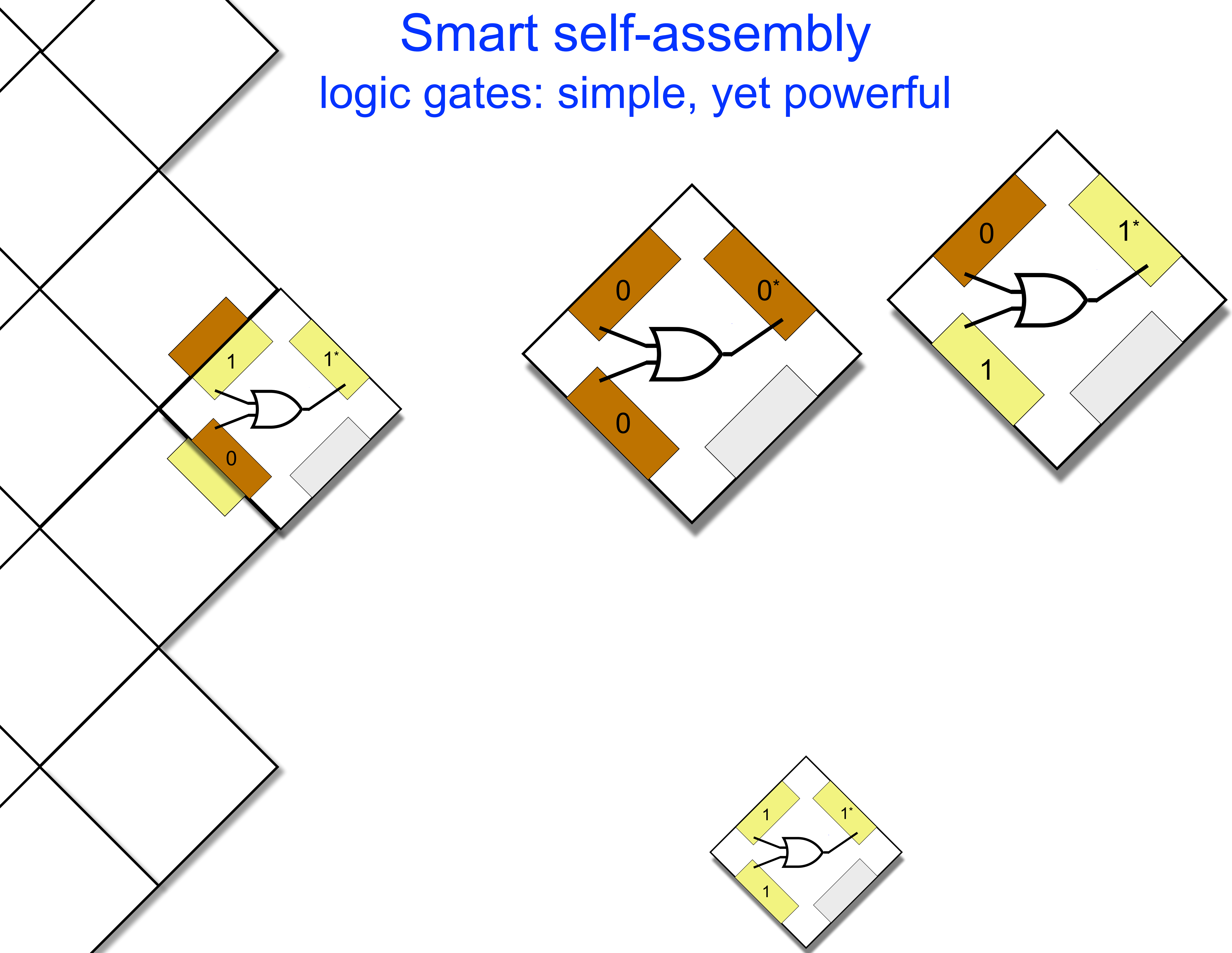
logic gates: simple, yet powerful





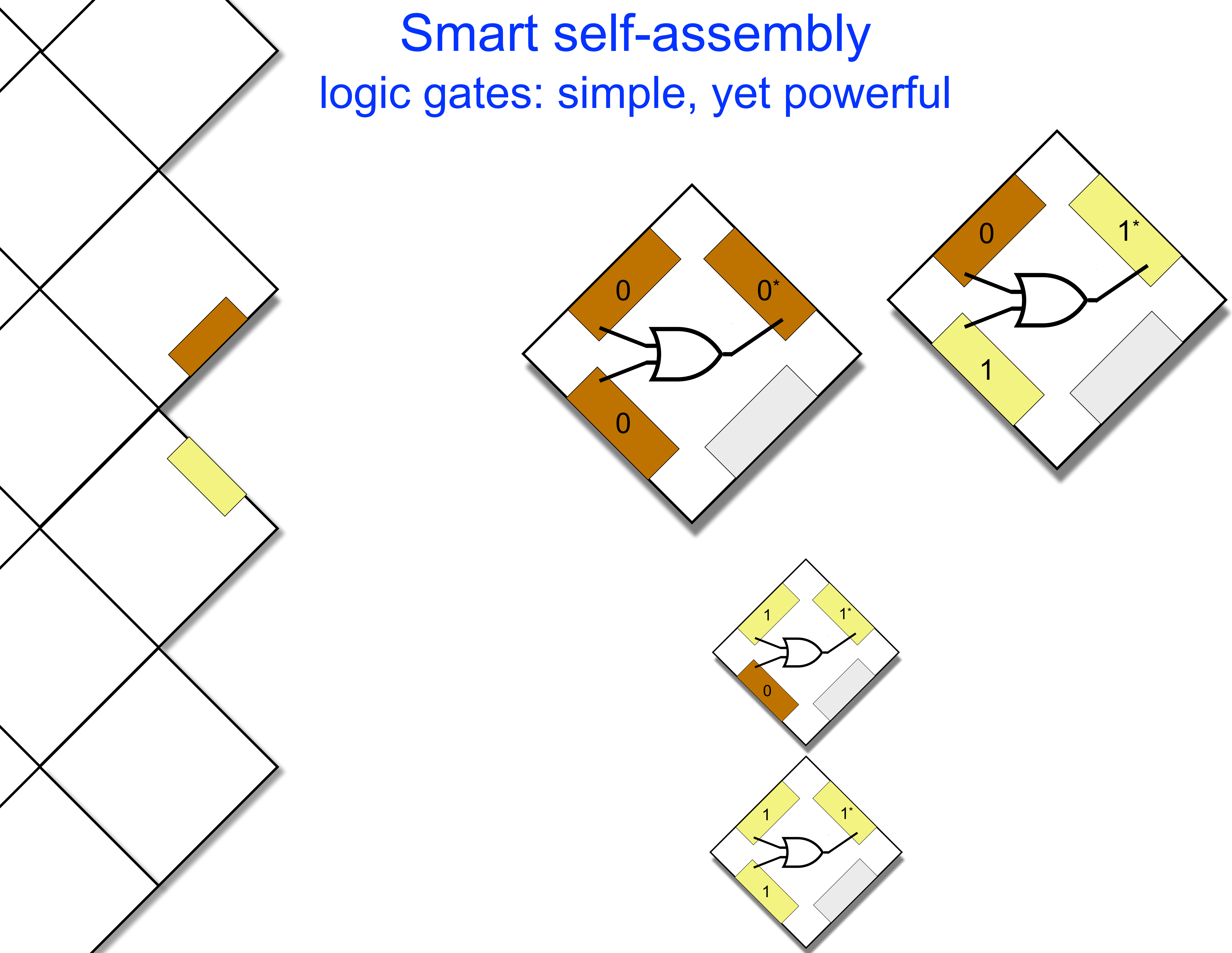
Smart self-assembly

logic gates: simple, yet powerful



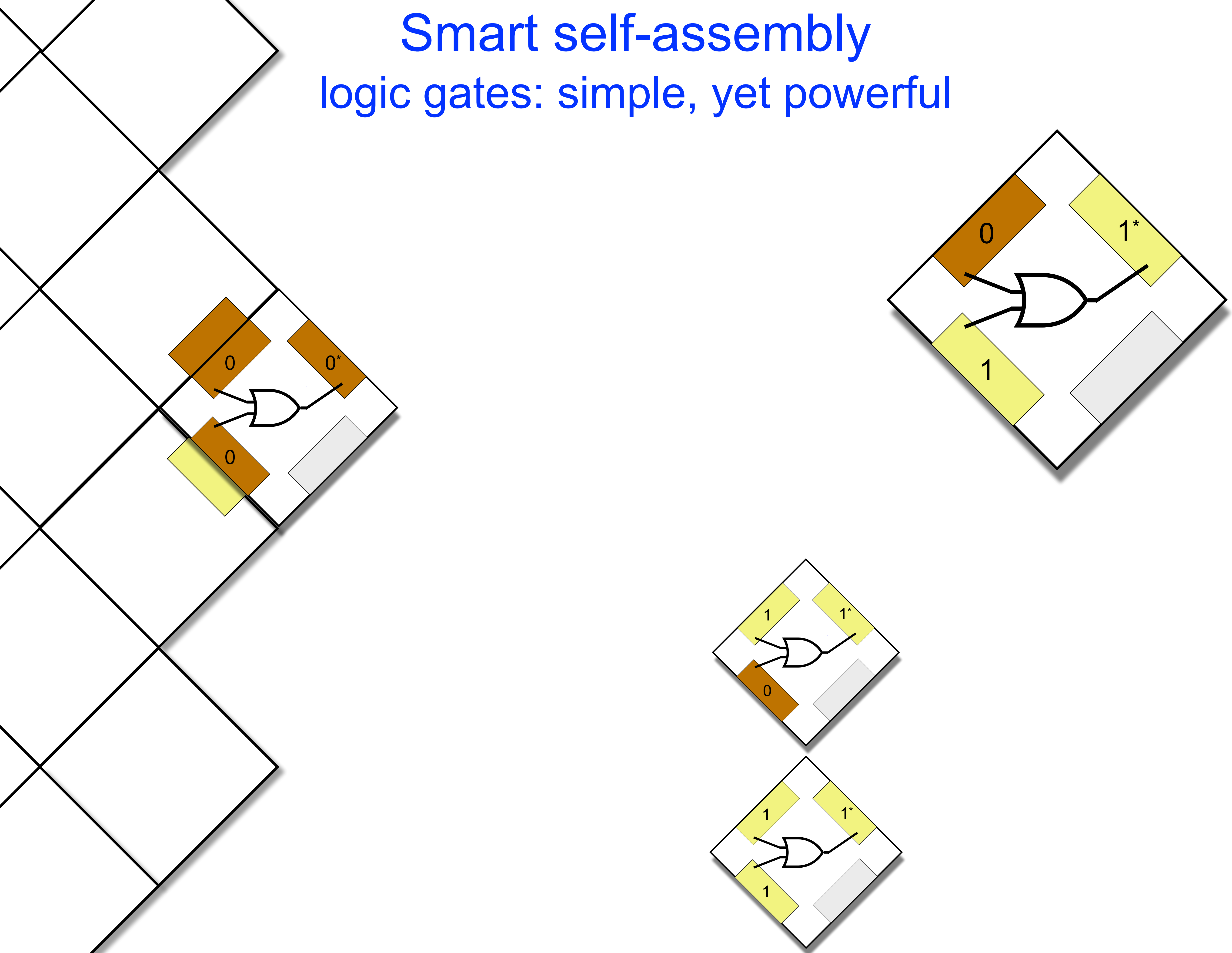
Smart self-assembly

logic gates: simple, yet powerful



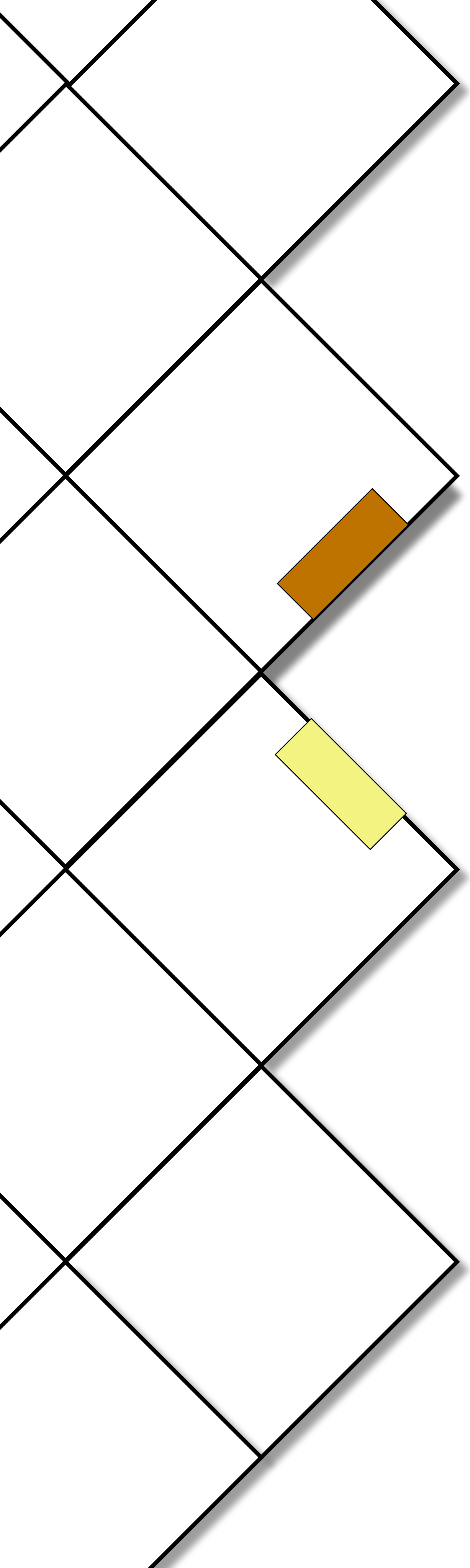
Smart self-assembly

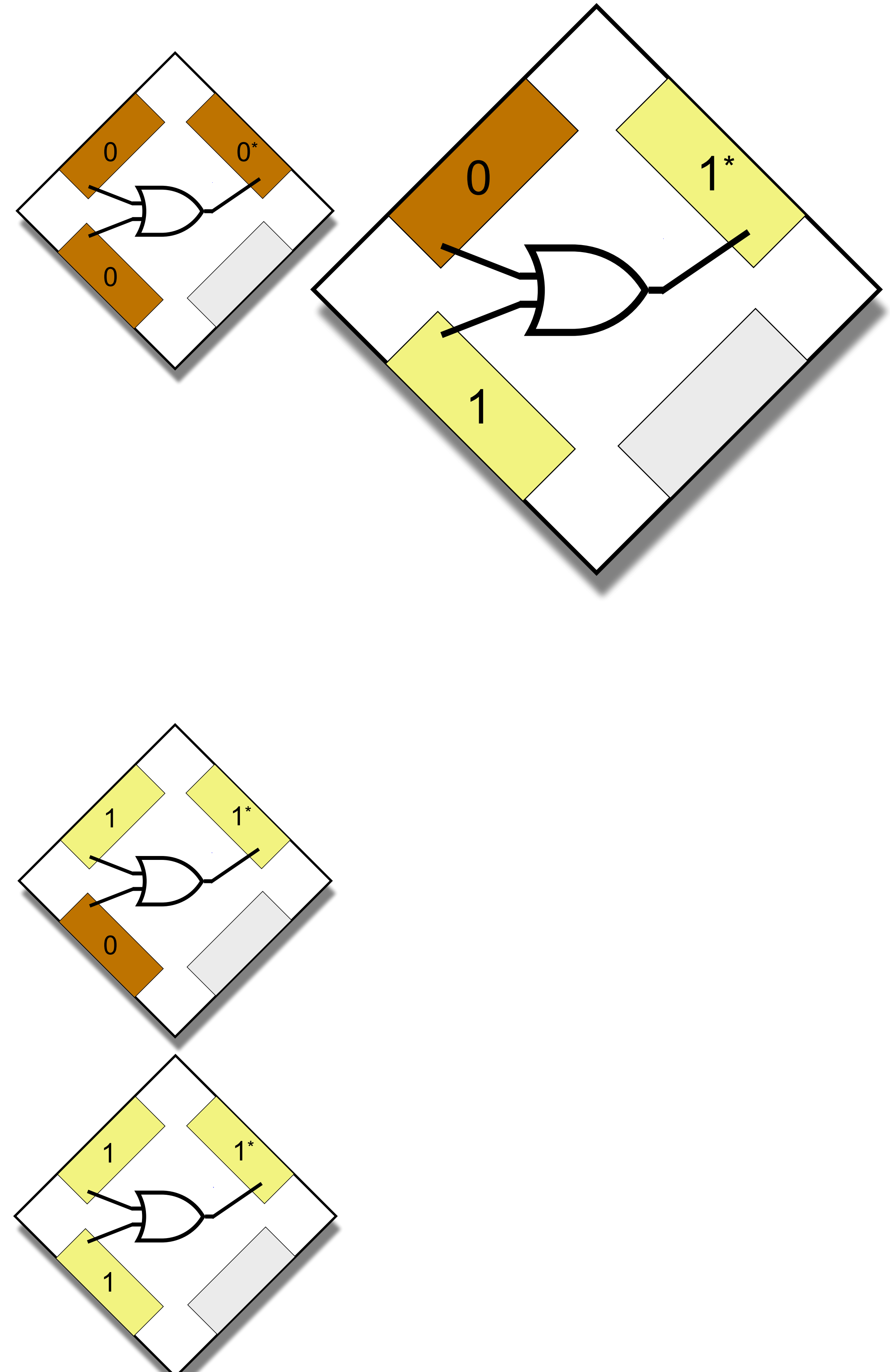
logic gates: simple, yet powerful



Smart self-assembly

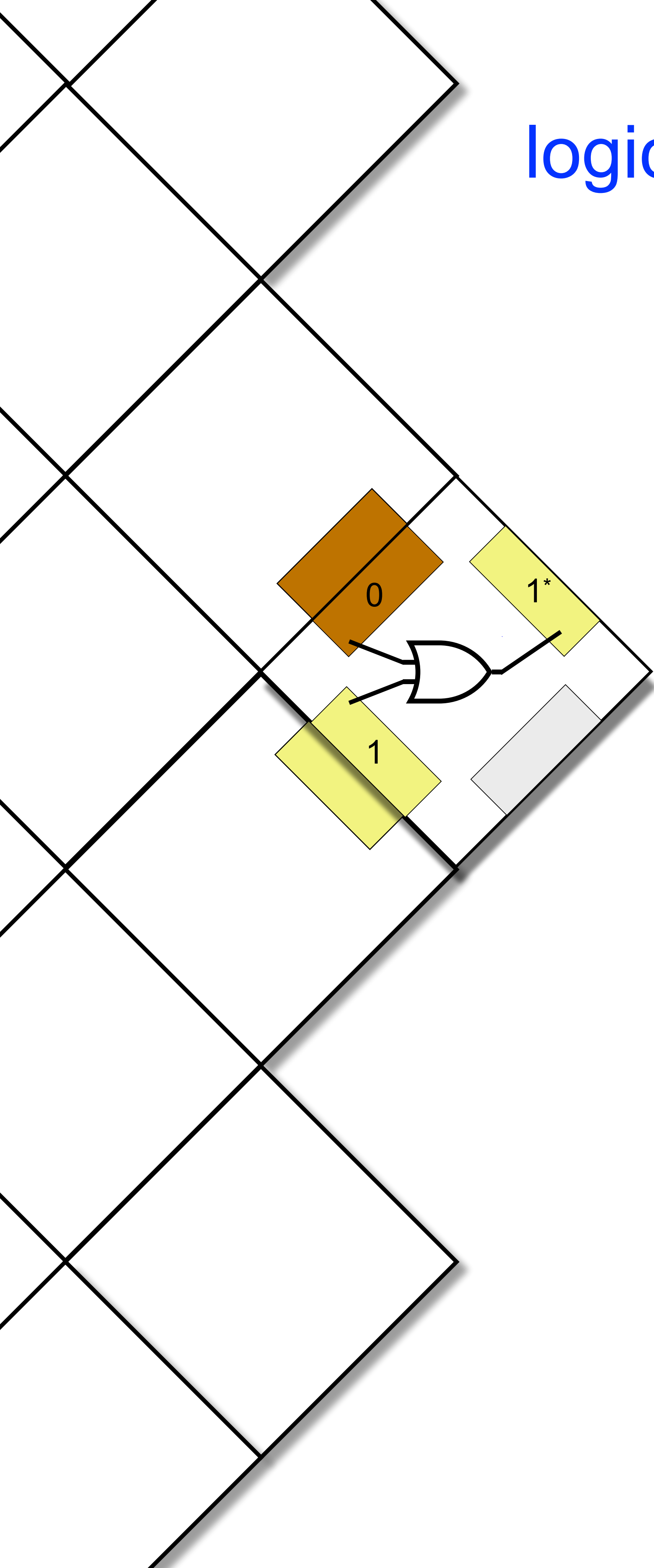
logic gates: simple, yet powerful

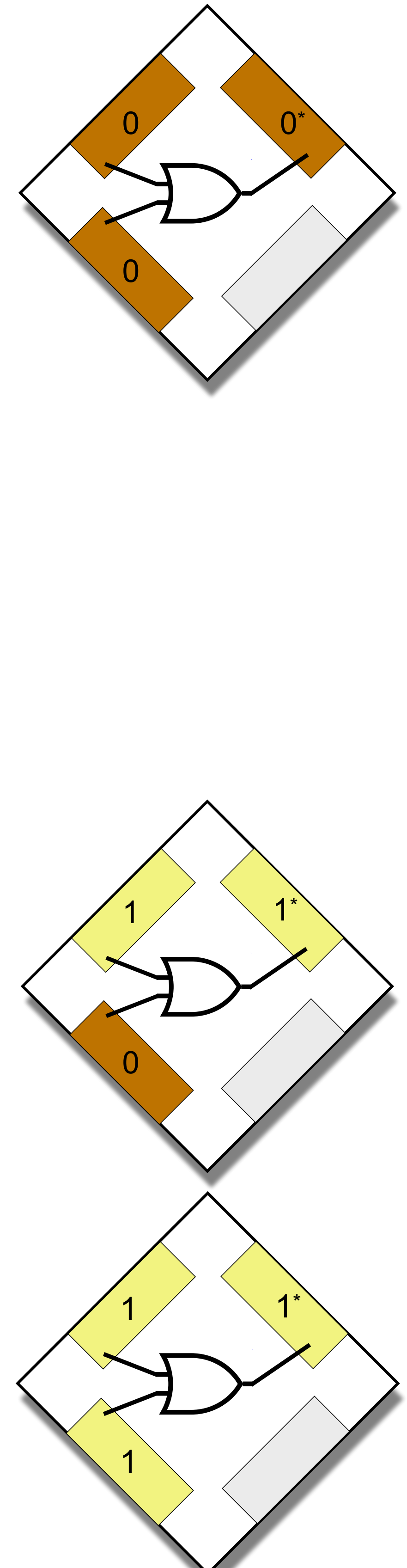




Smart self-assembly

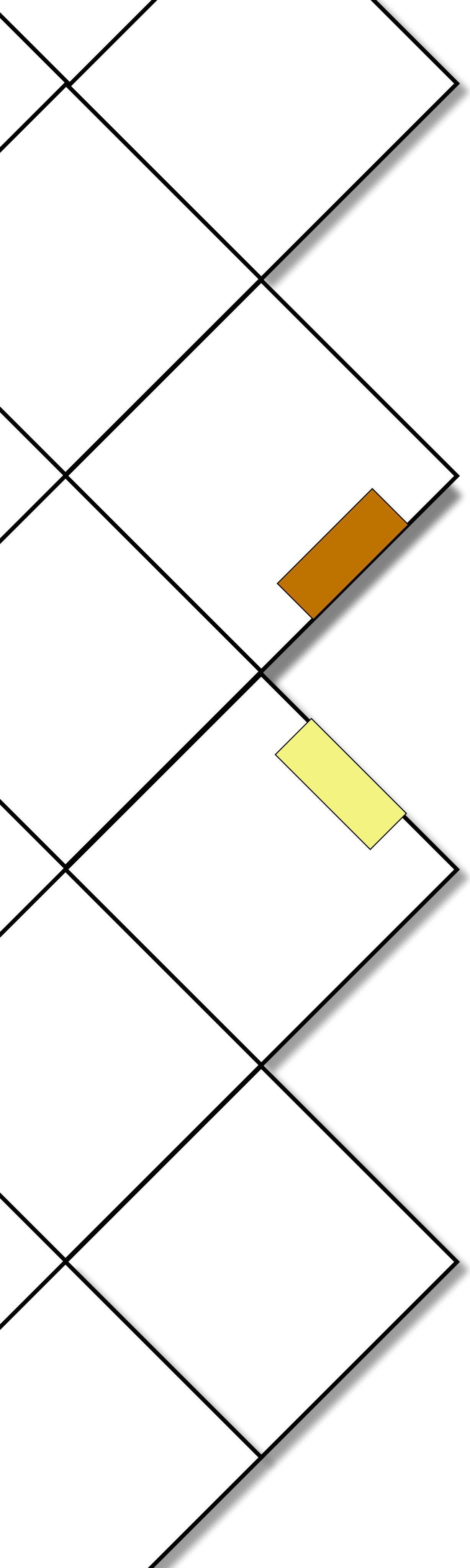
logic gates: simple, yet powerful

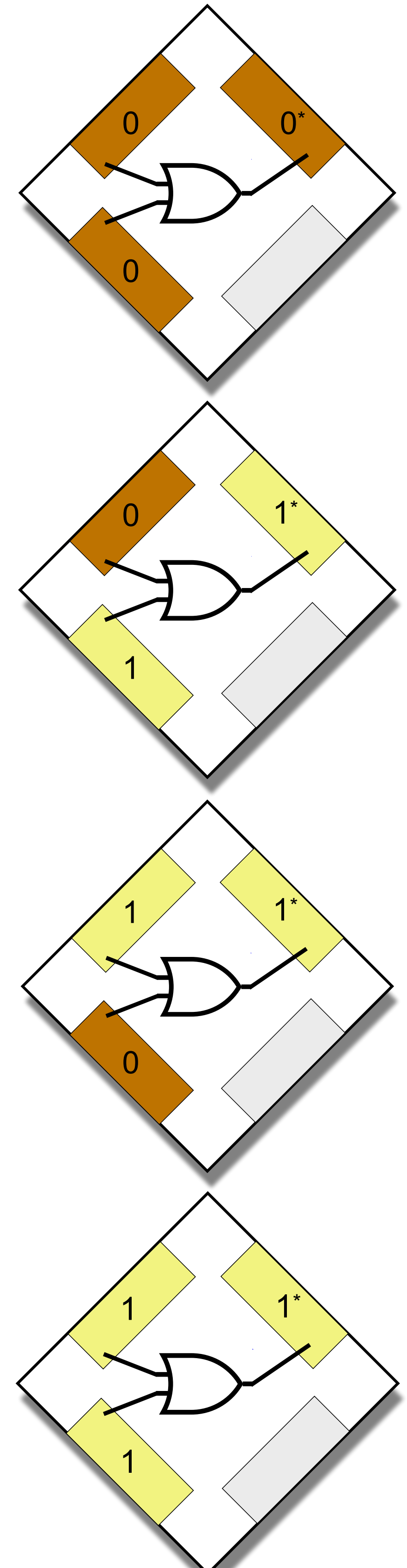




Smart self-assembly

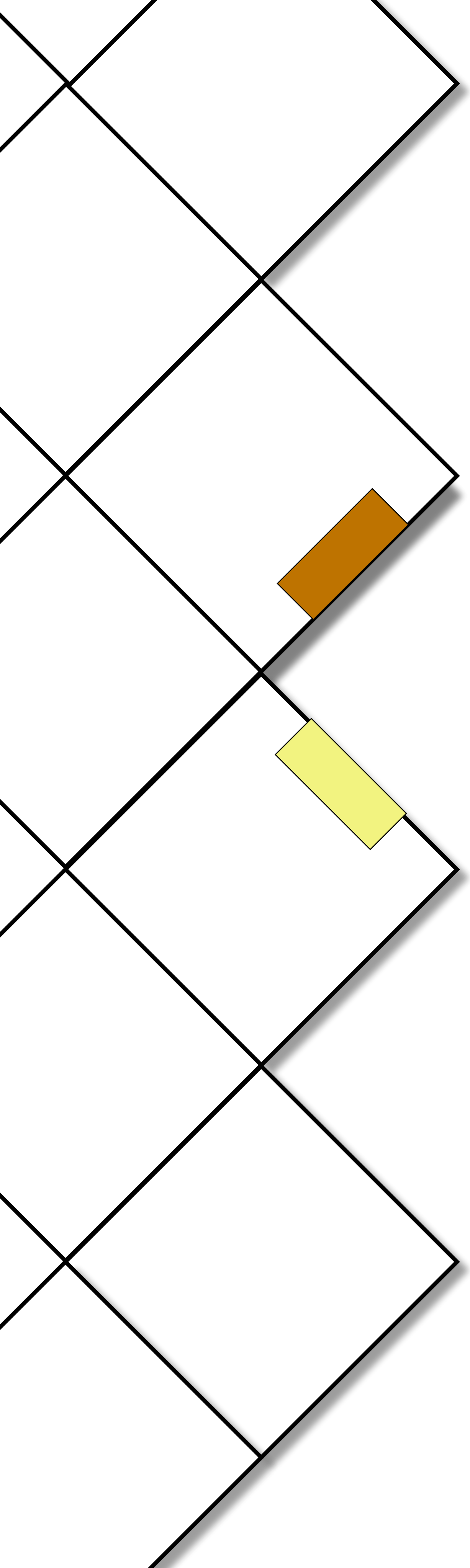
logic gates: simple, yet powerful

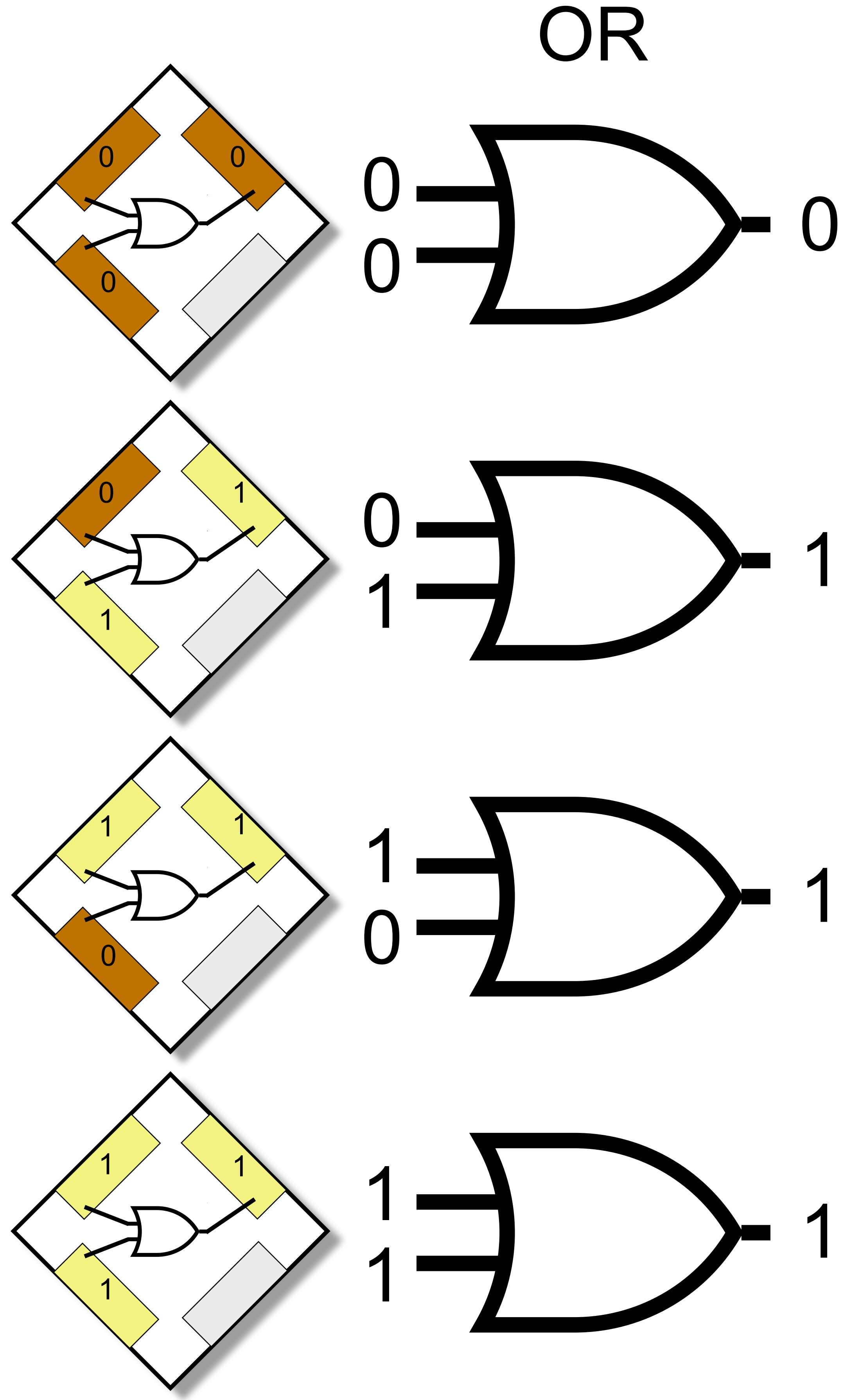




Smart self-assembly

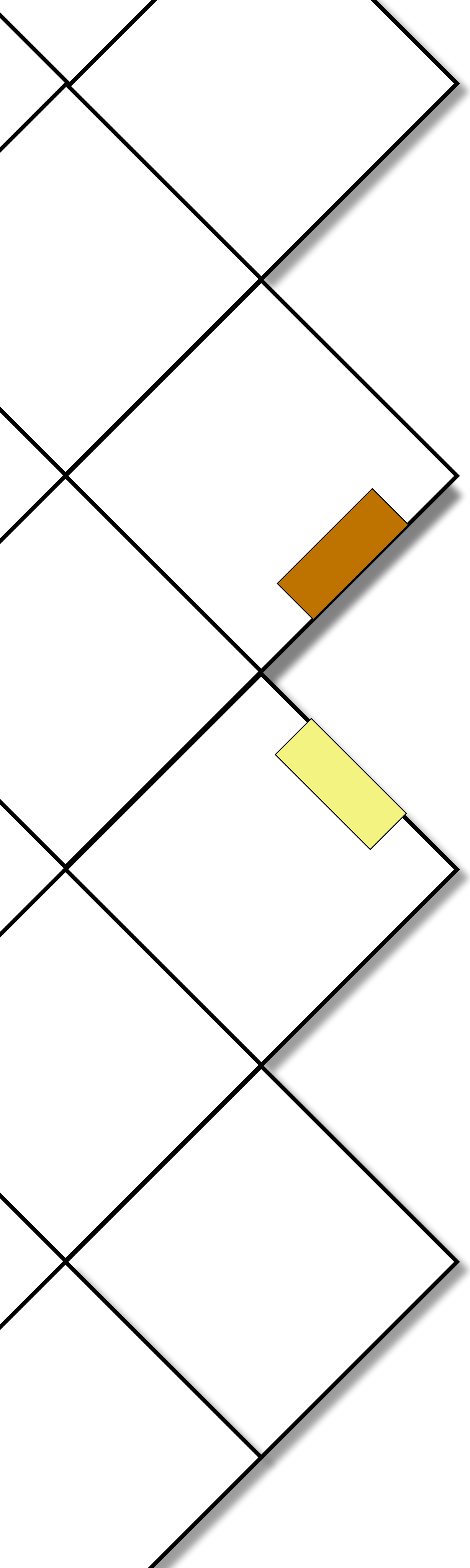
logic gates: simple, yet powerful

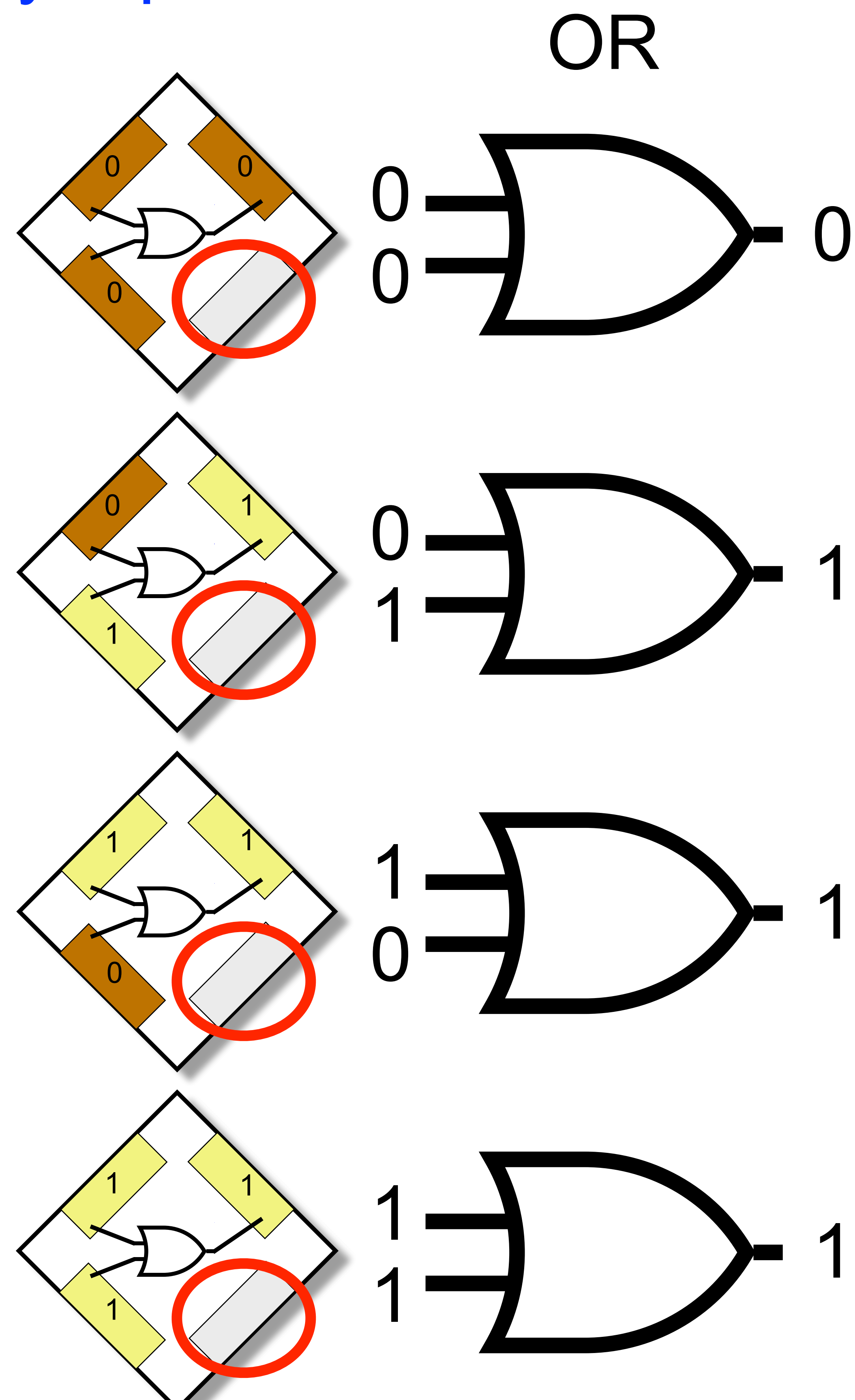




Smart self-assembly

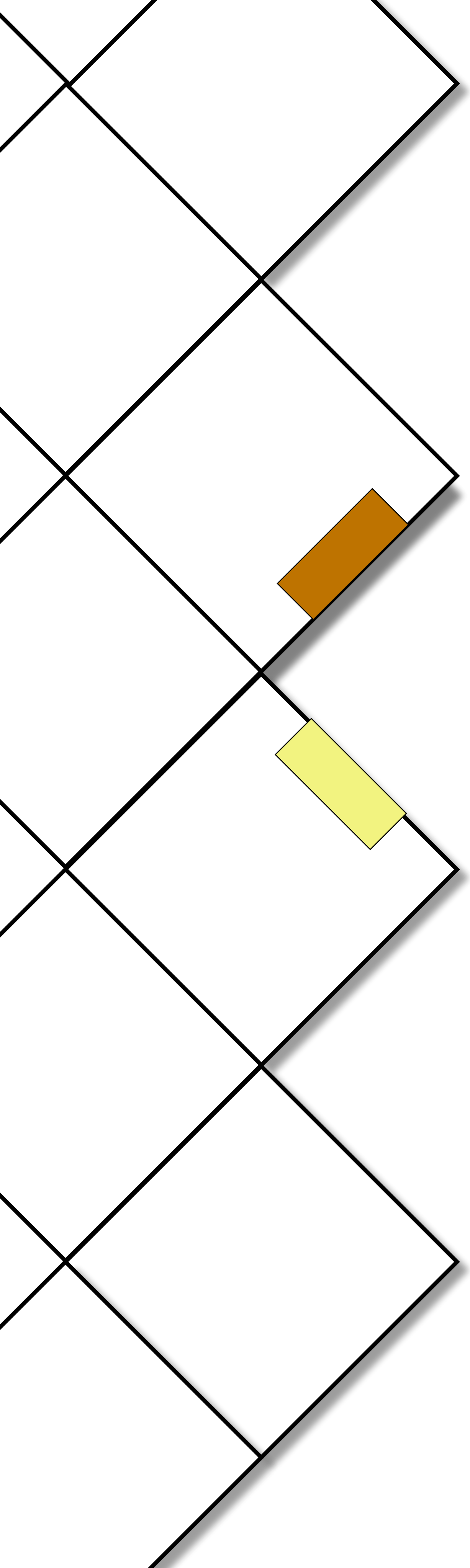
logic gates: simple, yet powerful

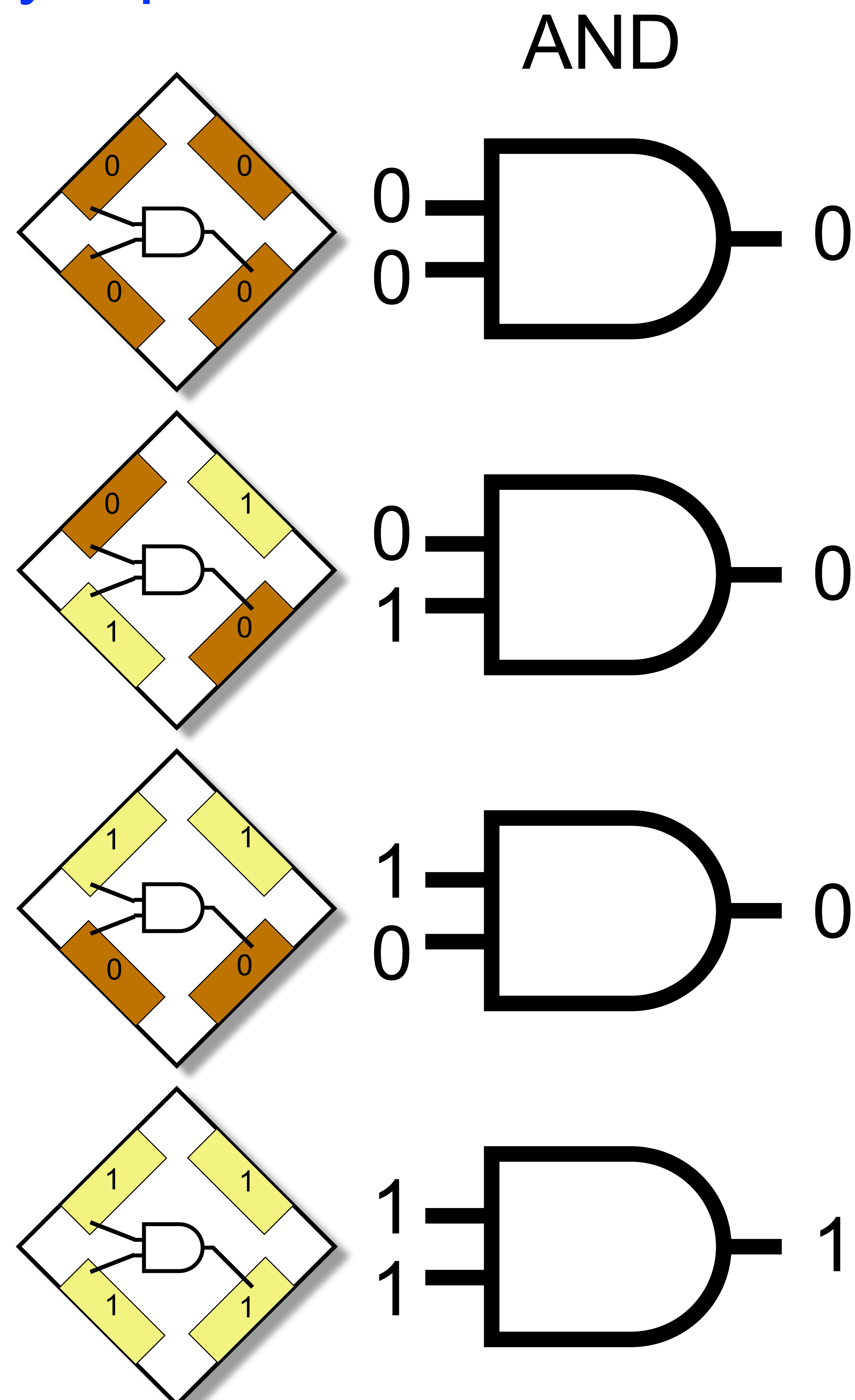




Smart self-assembly

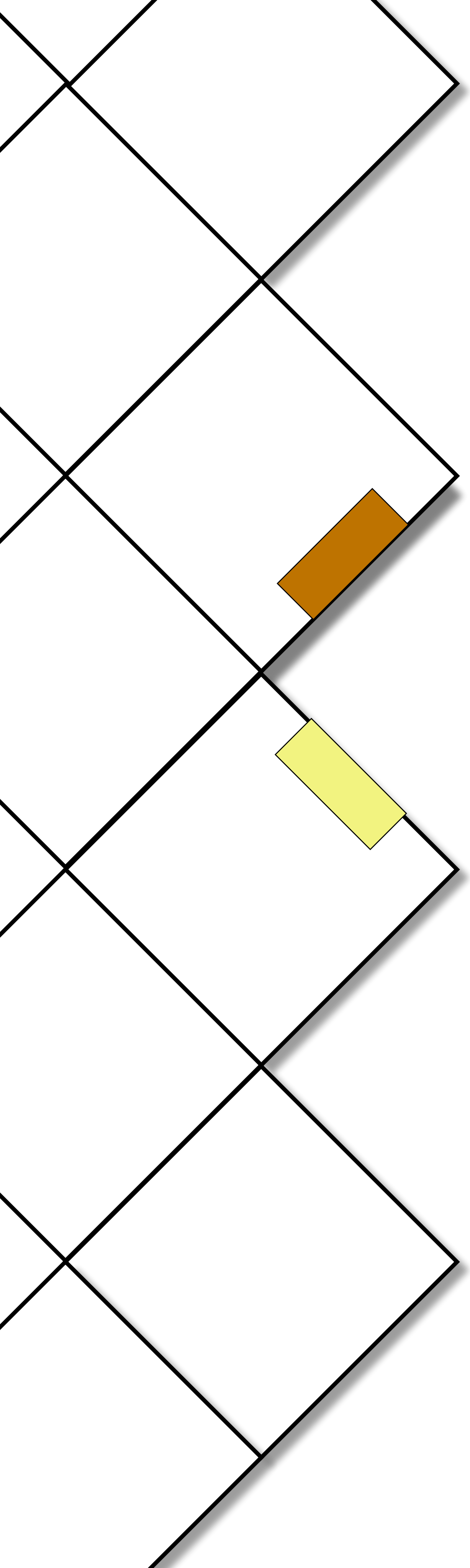
logic gates: simple, yet powerful

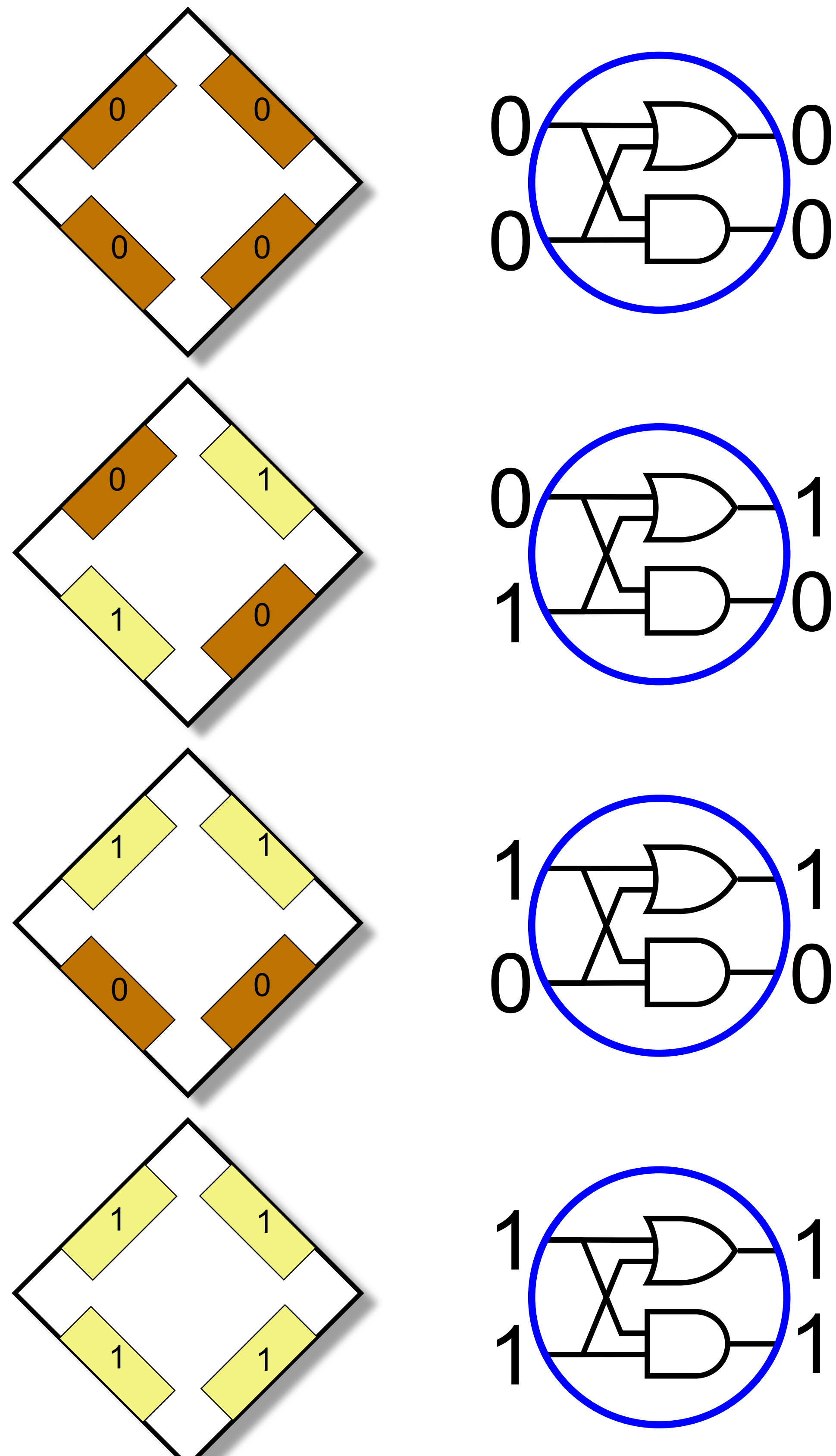




Smart self-assembly

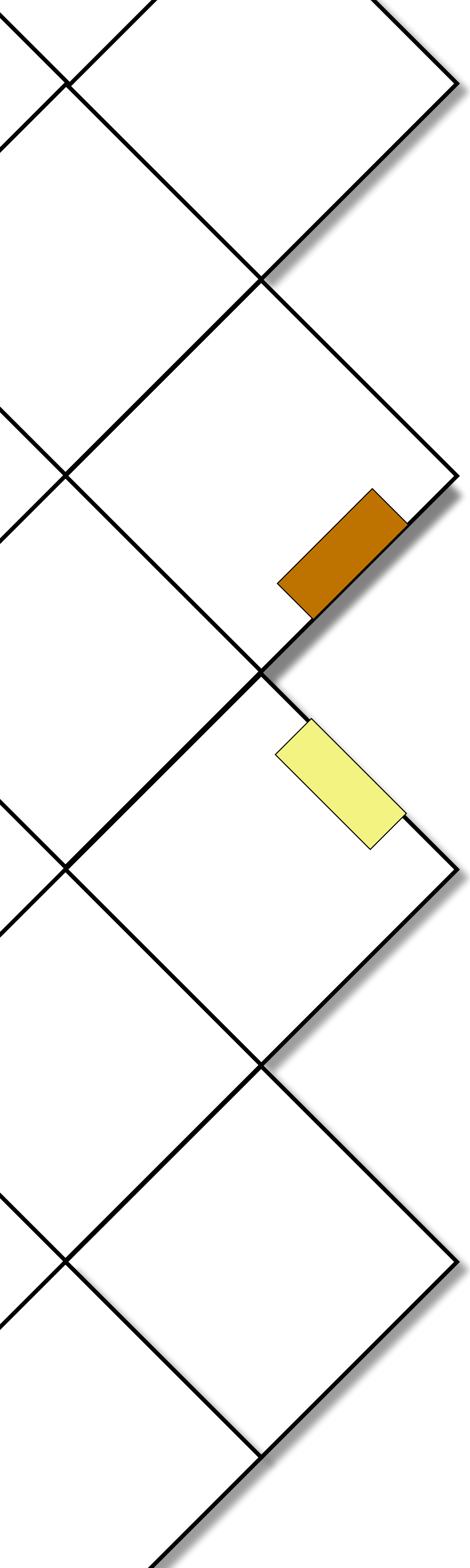
logic gates: simple, yet powerful

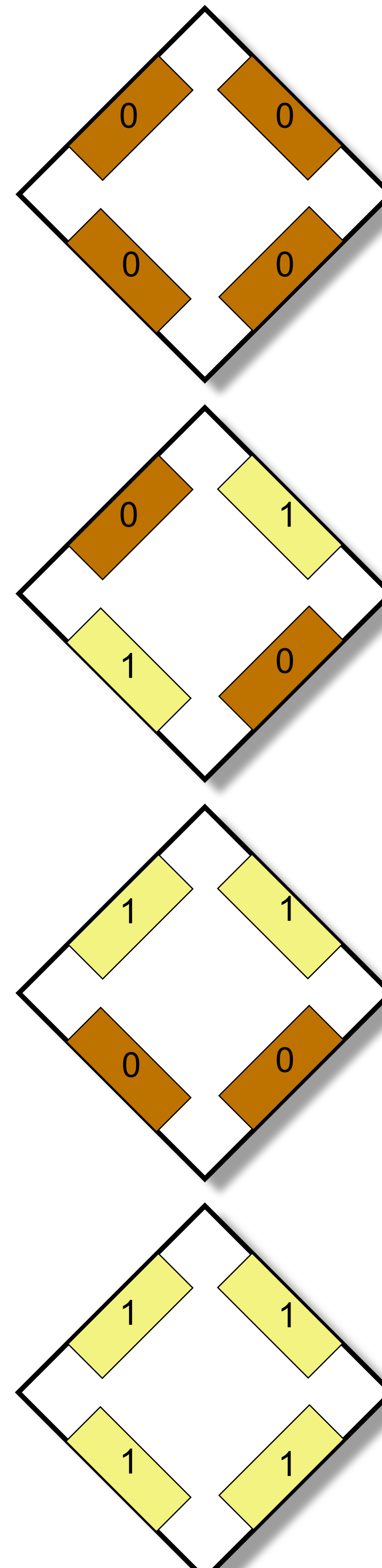


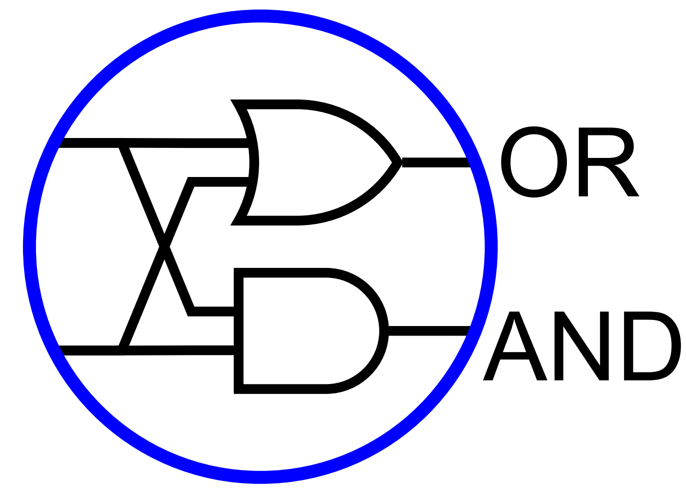


Smart self-assembly

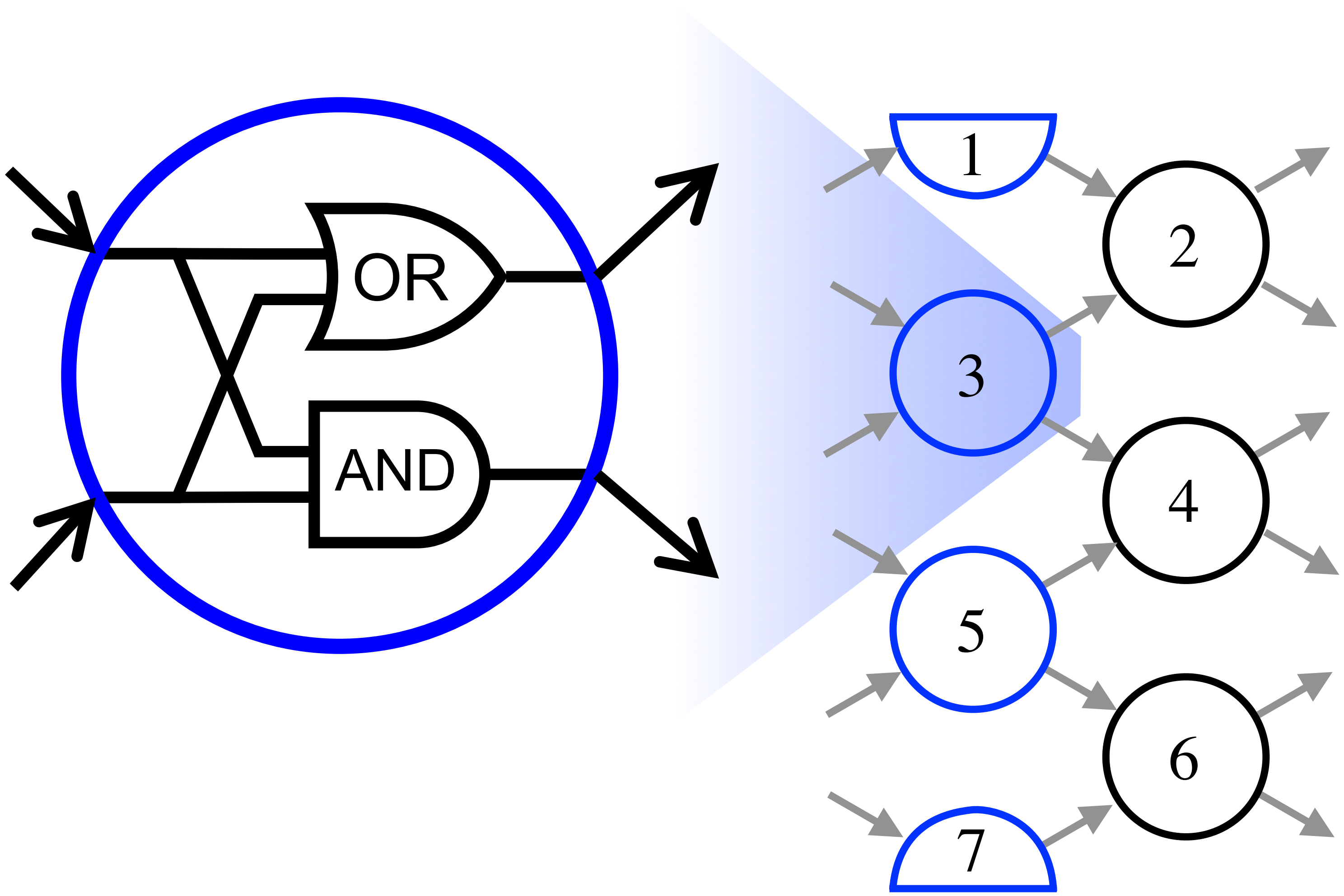
logic gates: simple, yet powerful



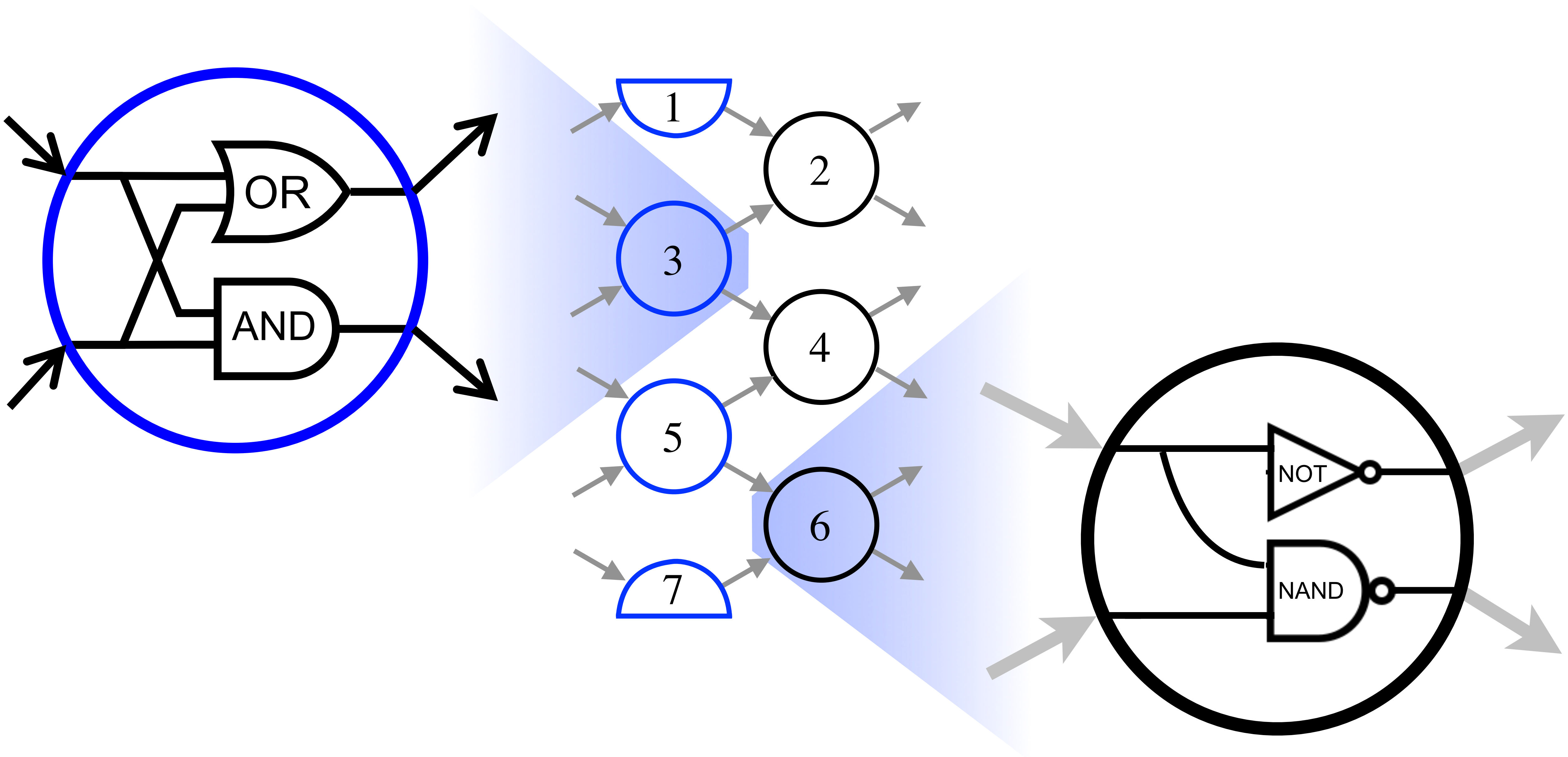




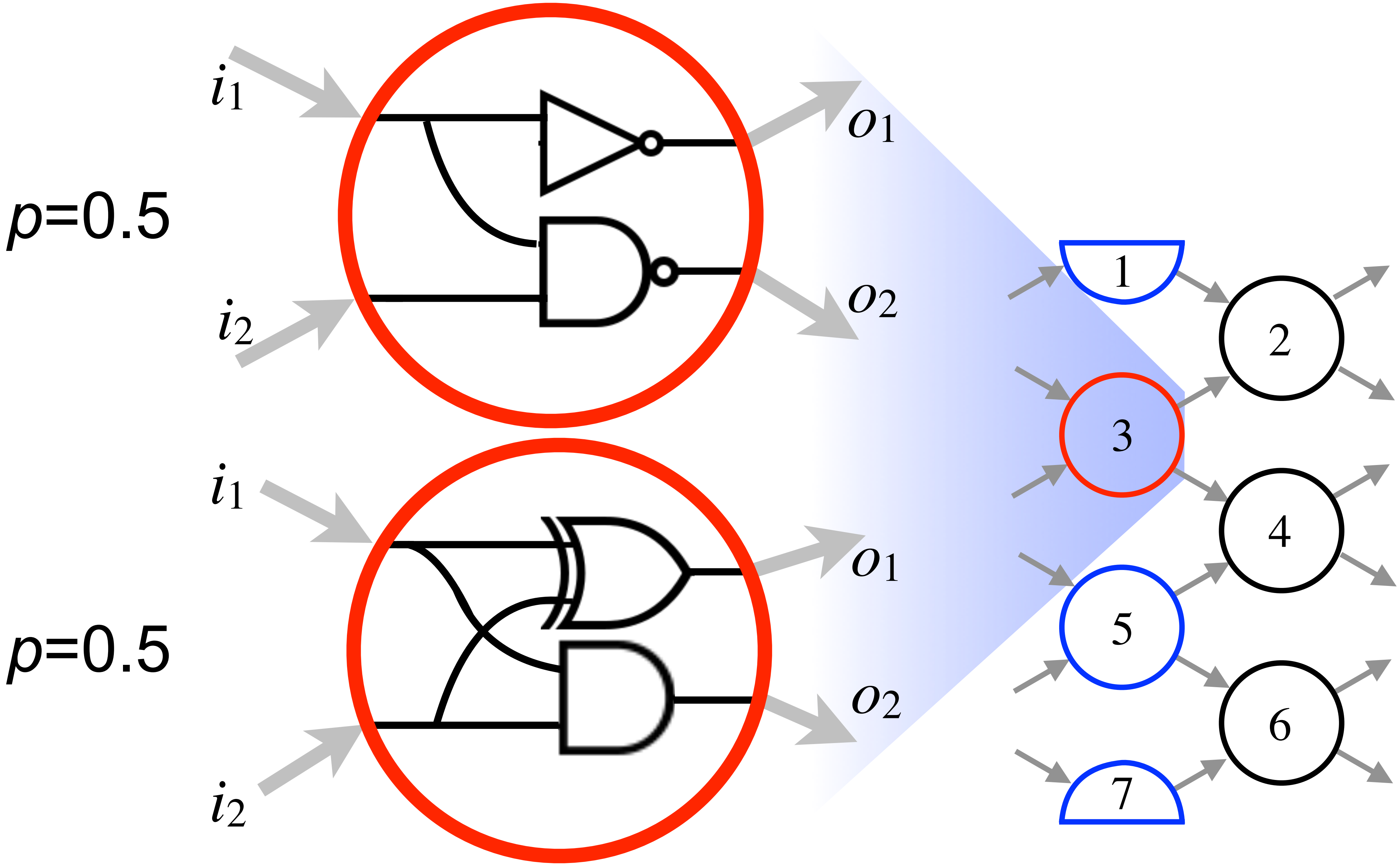
A local Boolean circuit model



A local Boolean circuit model

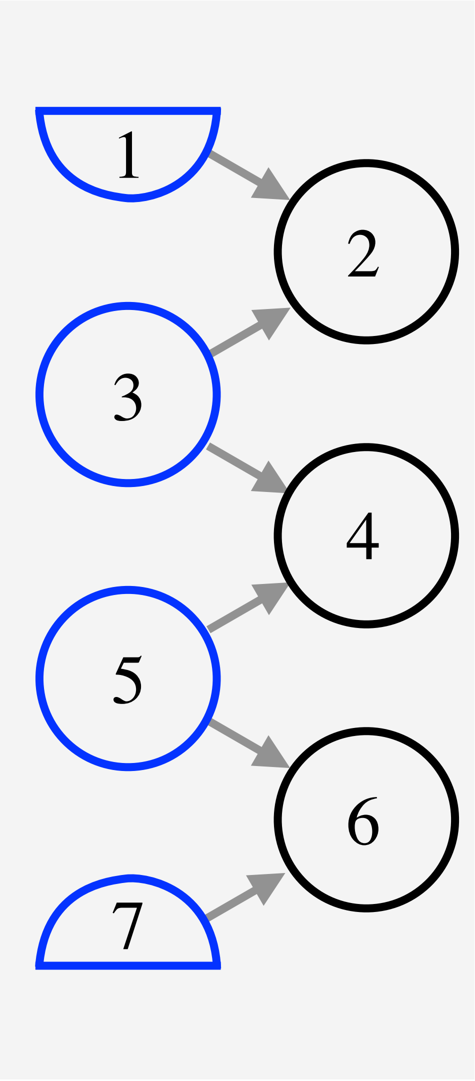


A local circuit model: randomised gates



A local Boolean circuit model

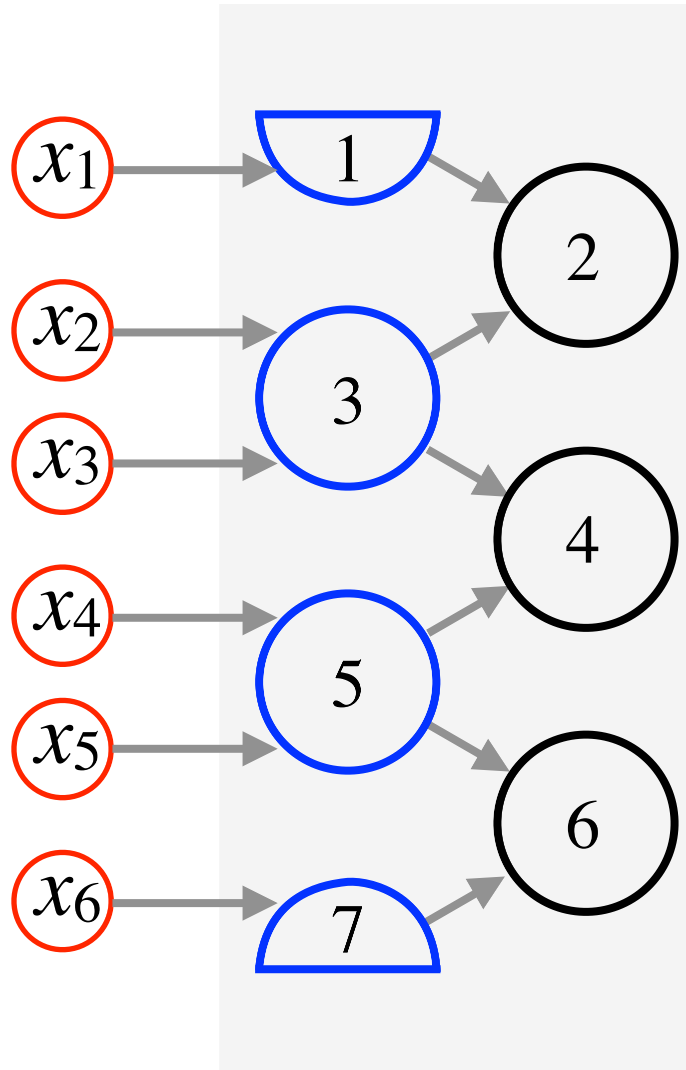
Programmer
specifies a layer



A local Boolean circuit model

Programmer specifies a layer

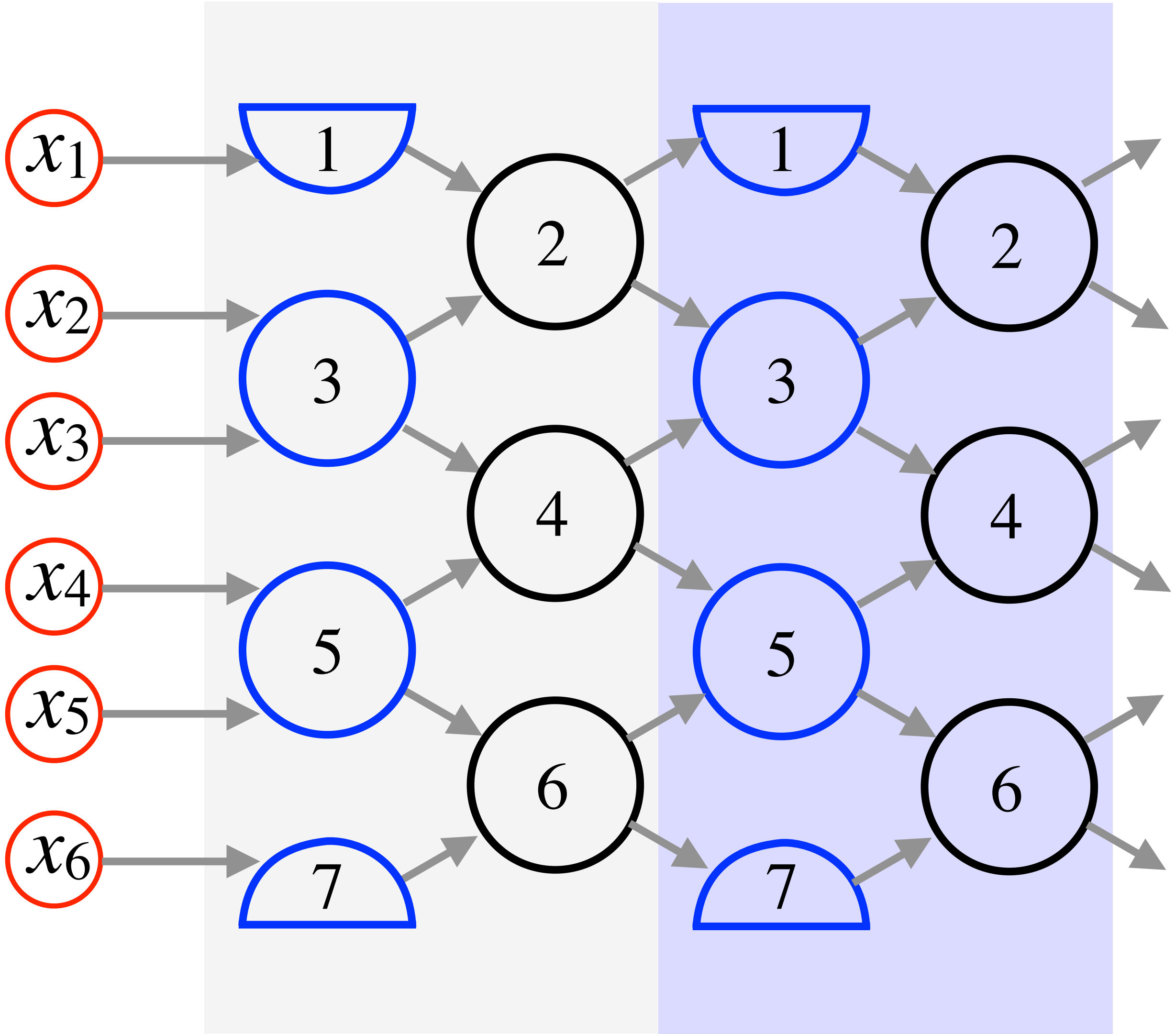
User gives *n* input bits $x_k \in \{0,1\}$



A local Boolean circuit model

Programmer specifies a layer

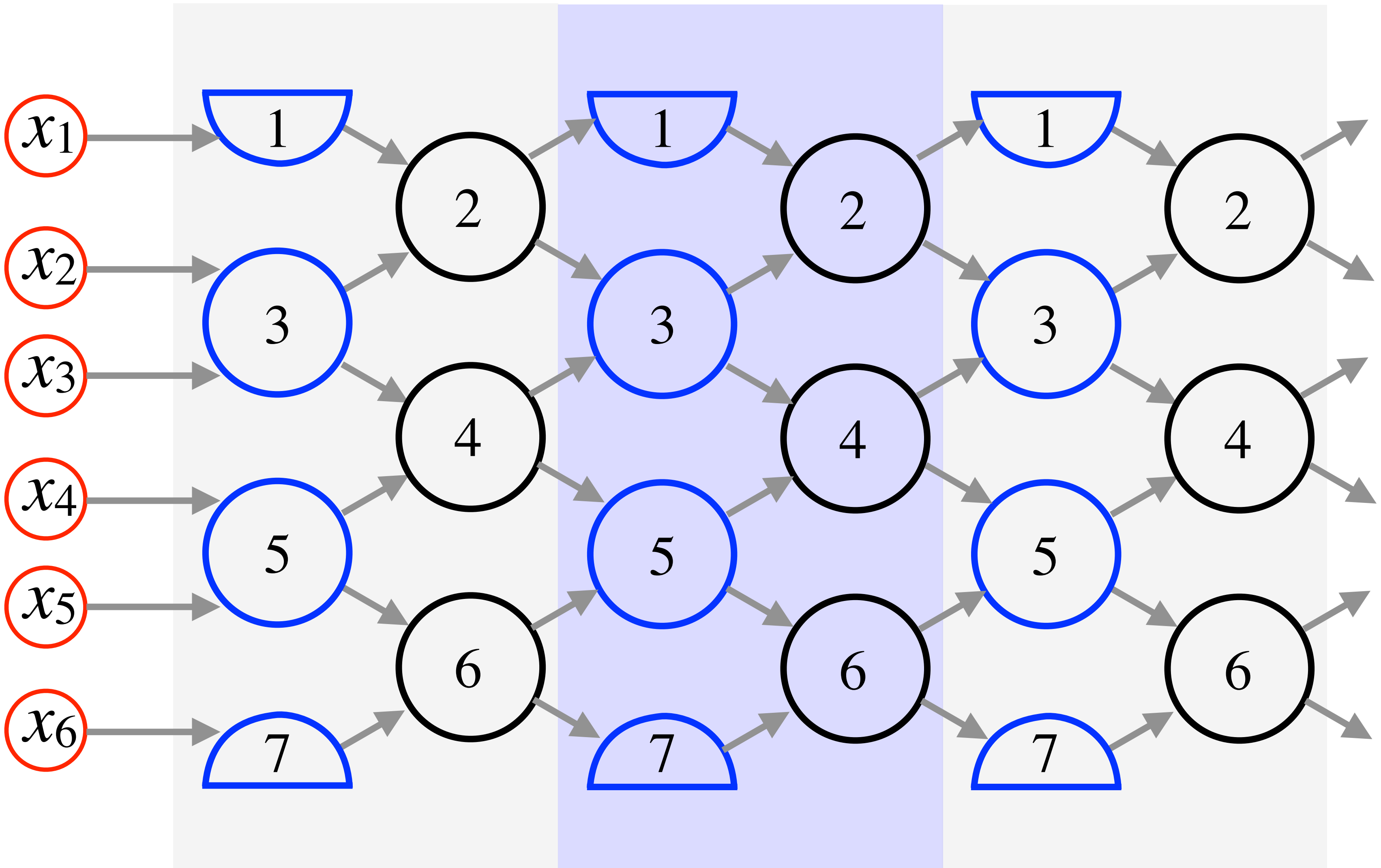
User gives *n* input bits $x_k \in \{0,1\}$



A local Boolean circuit model

Programmer specifies a layer

User gives *n* input bits $x_k \in \{0,1\}$

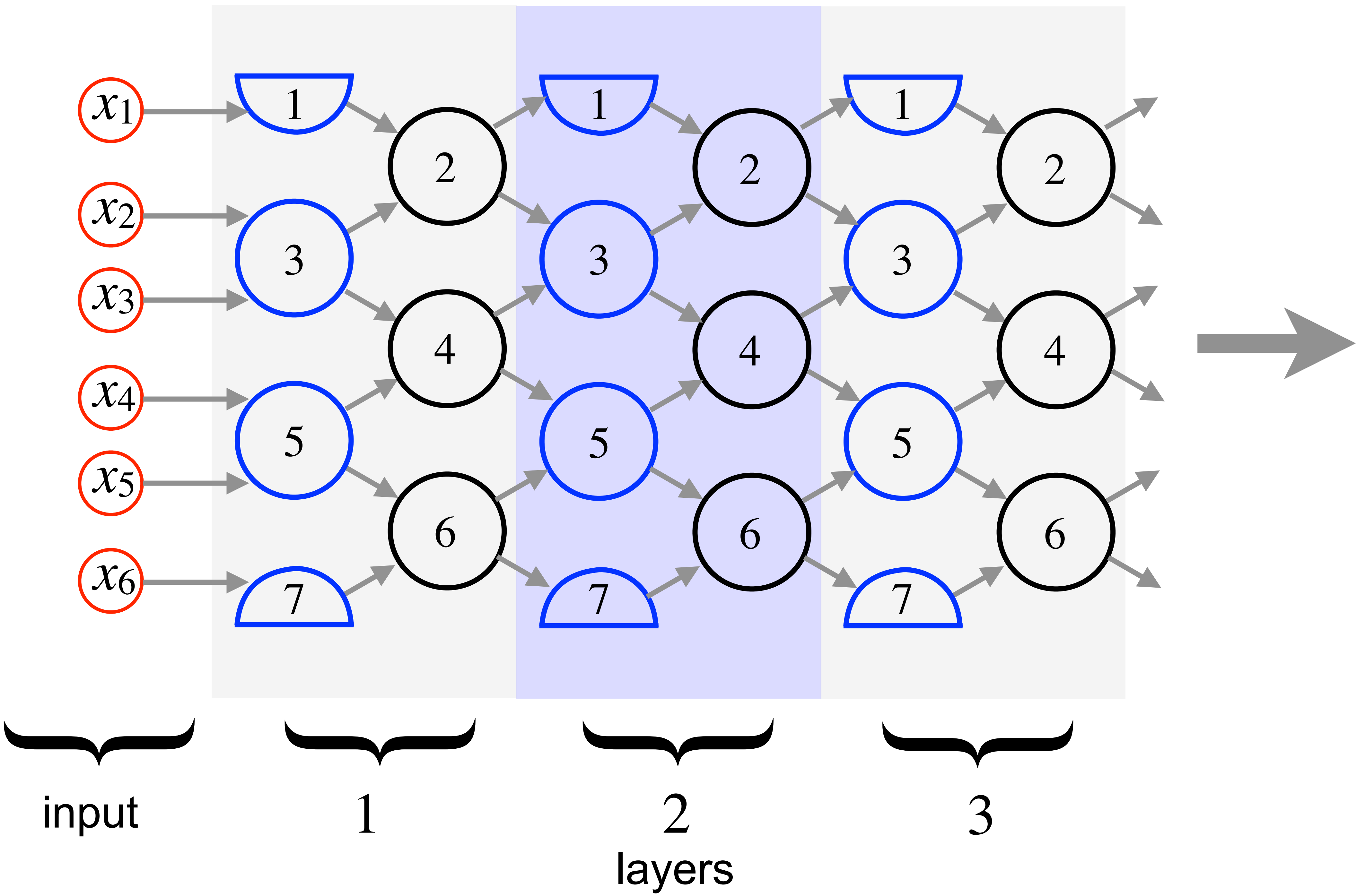


A local Boolean circuit model

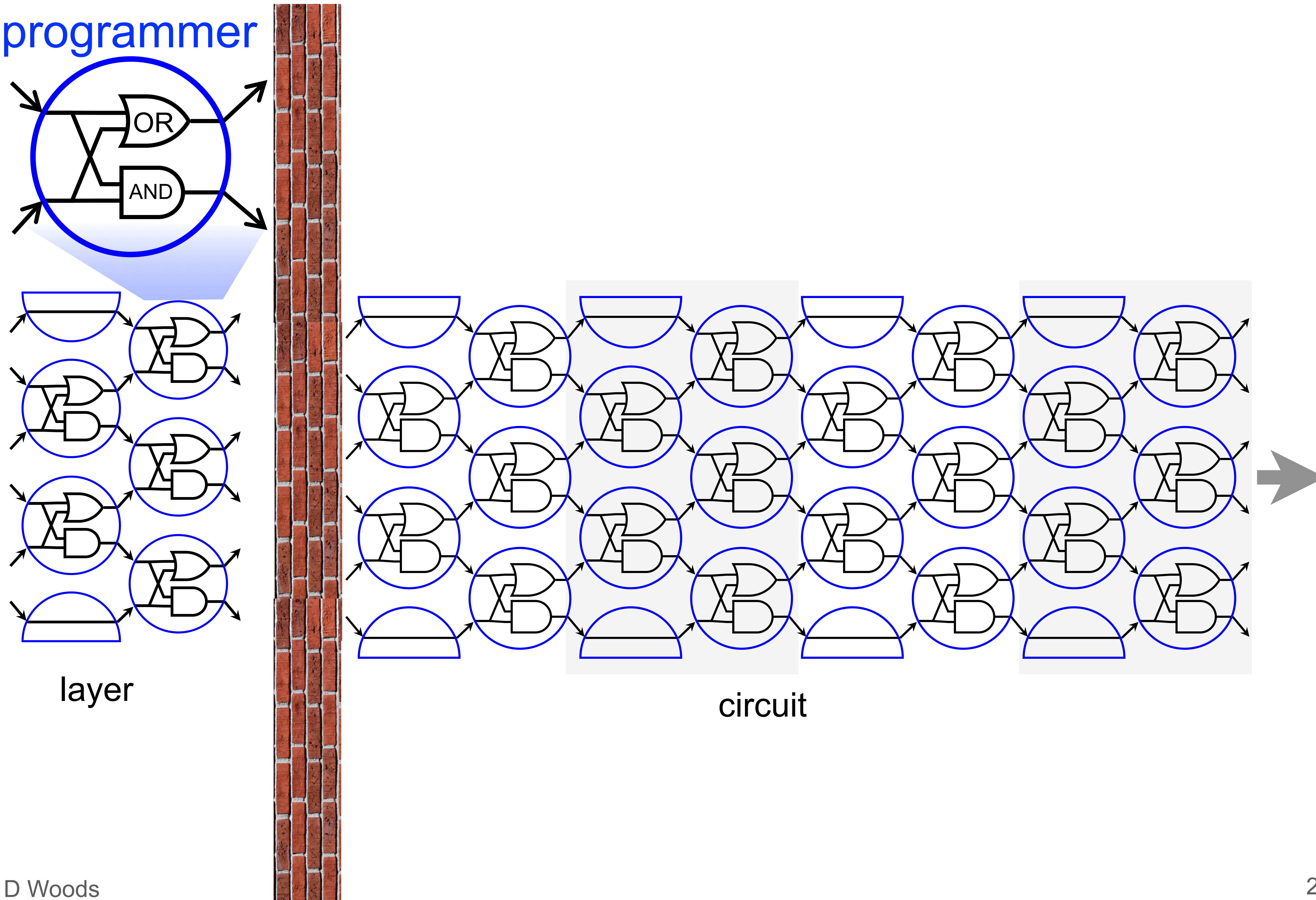
Programmer specifies a layer

User gives *n* input bits $x_k \in \{0,1\}$

Computation flows from input gates to layer 1, layer 2, layer 3 …

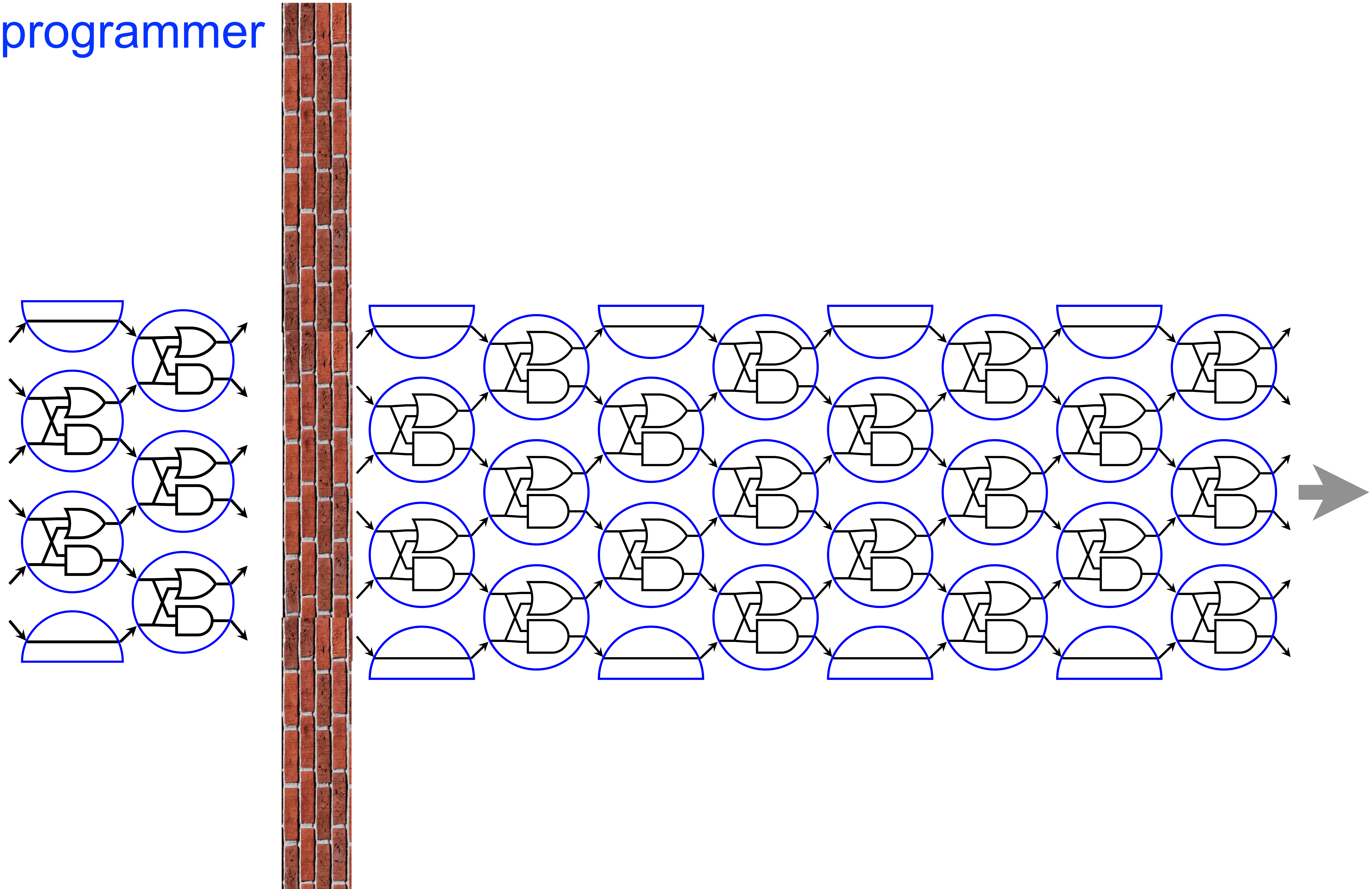


Example circuit



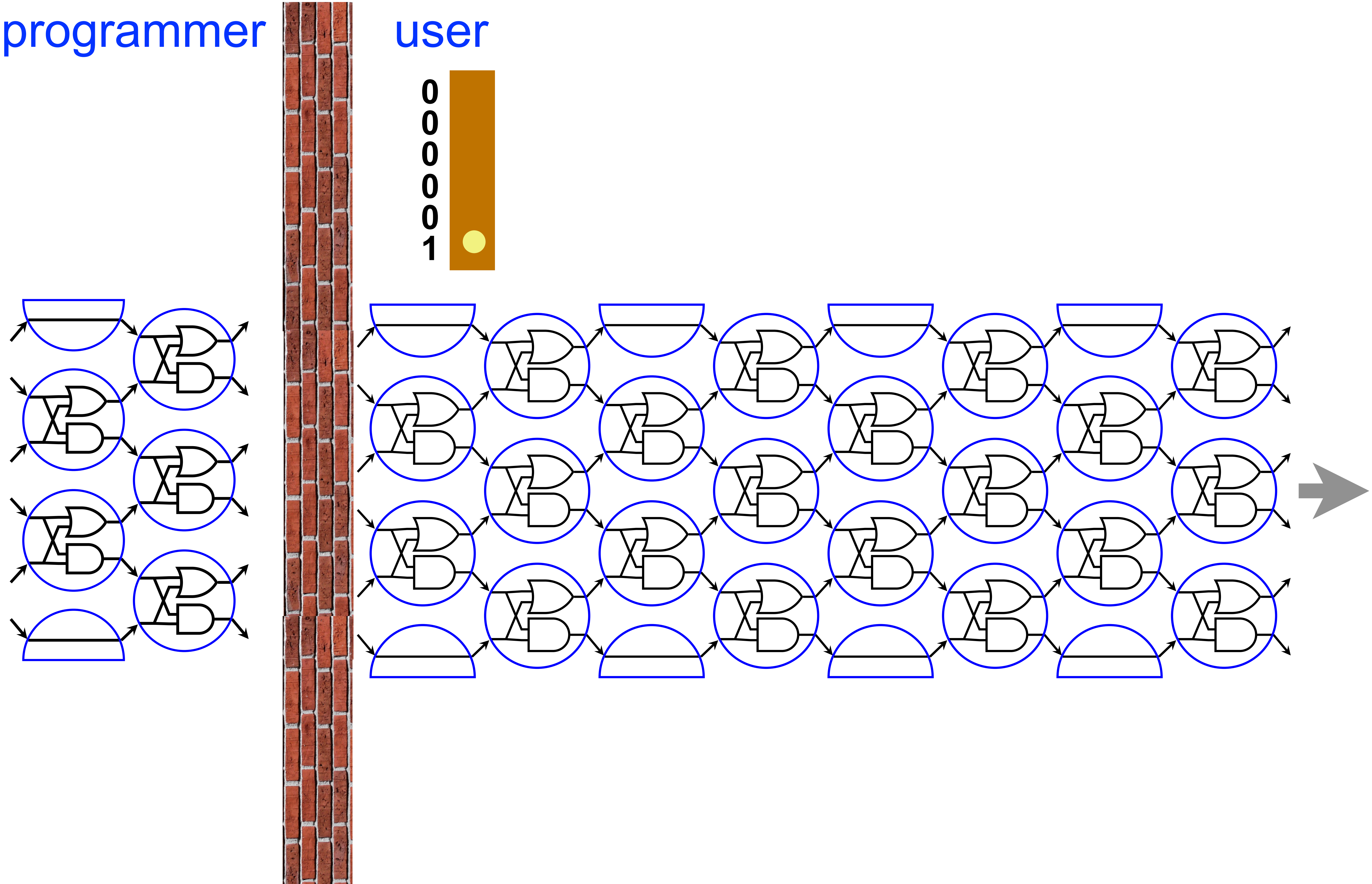
Example circuit

programmer



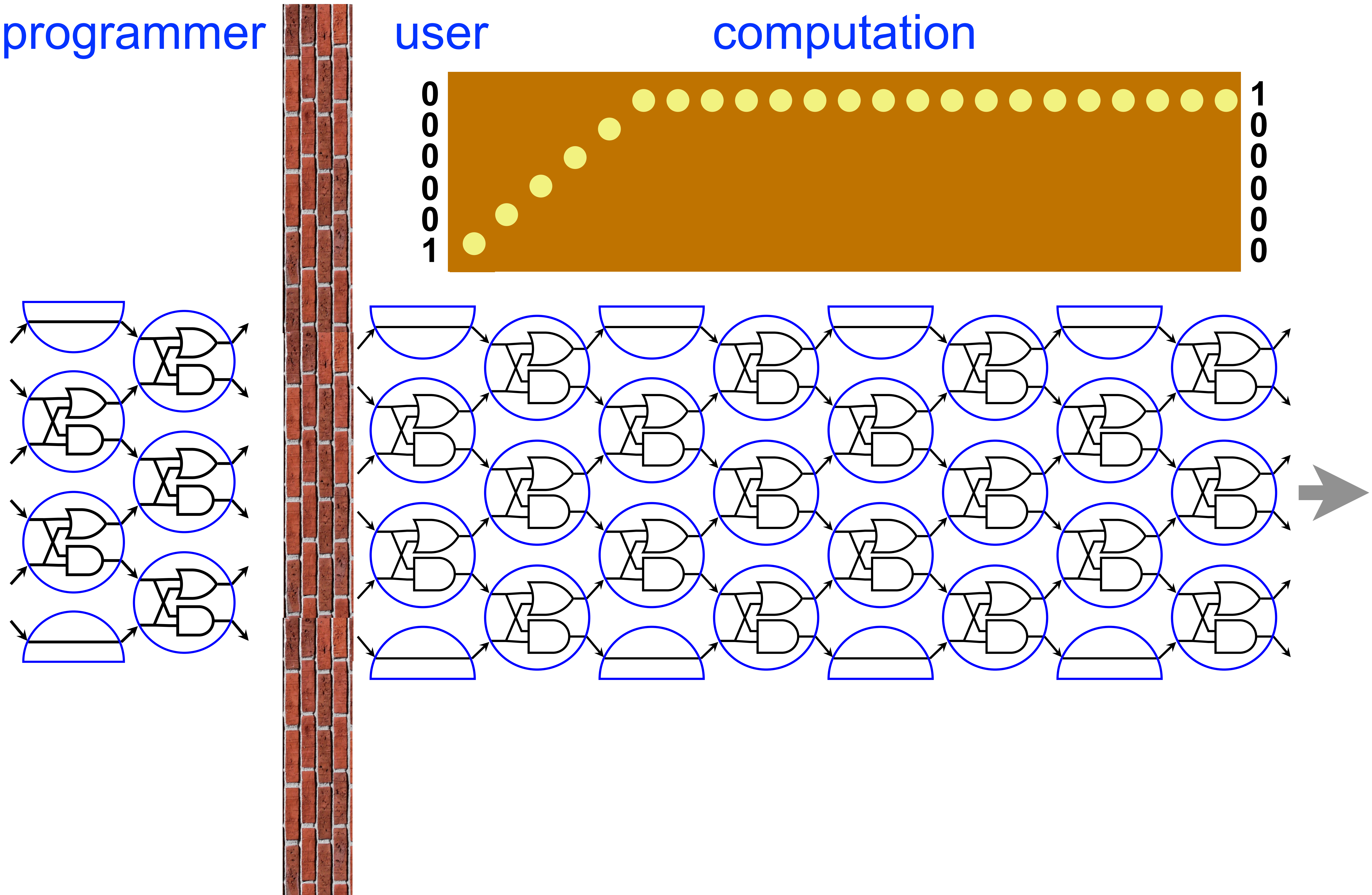
wall credit: Erik Klavins

Example circuit



wall credit: Erik Klavins

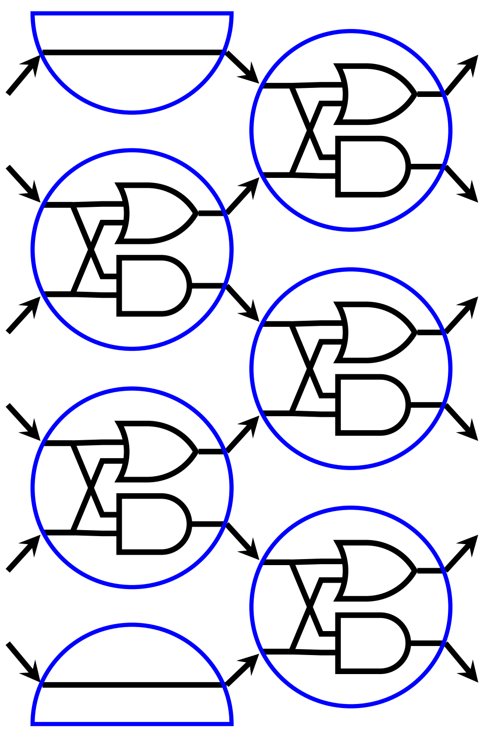
Example circuit



wall credit: Erik Klavins

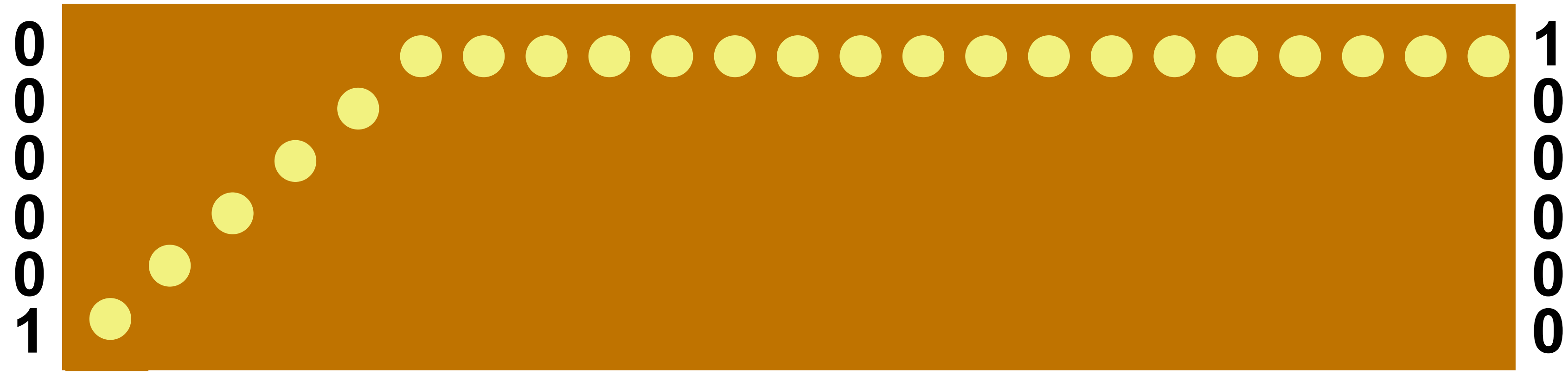
Example circuit

programmer



user

computation



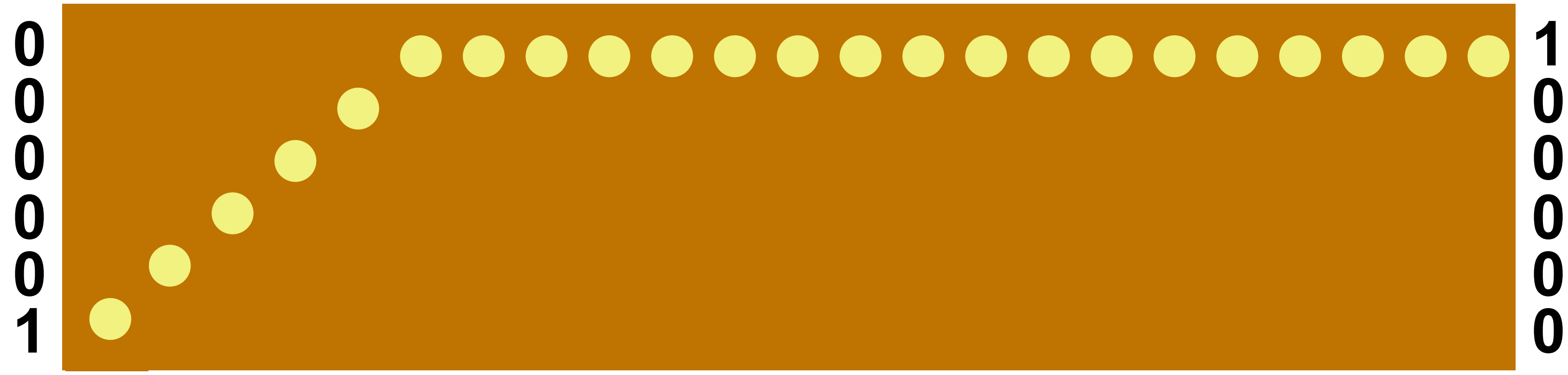
wall credit: Erik Klavins

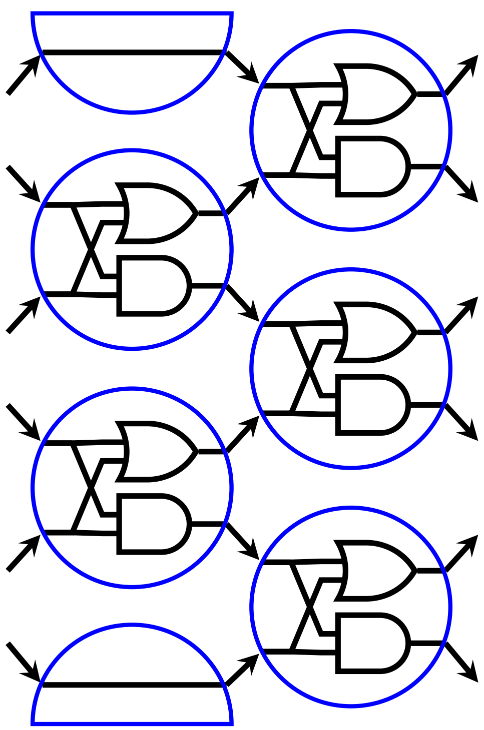
Example circuit: “SORTING”

programmer

user

computation

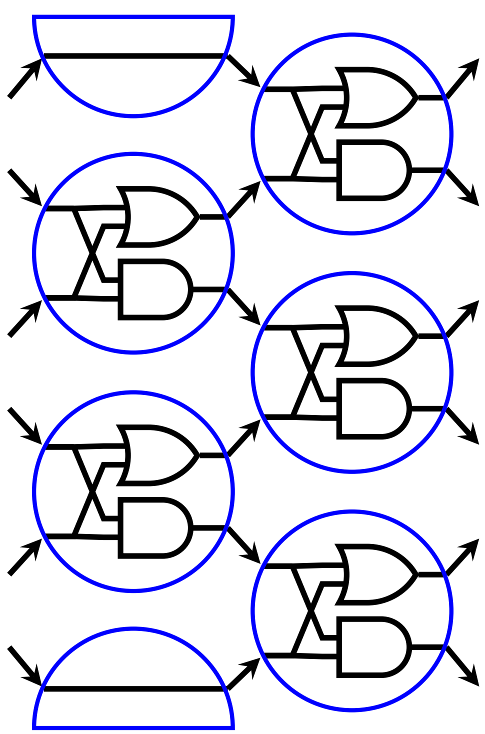




wall credit: Erik Klavins

Example circuit: “SORTING”

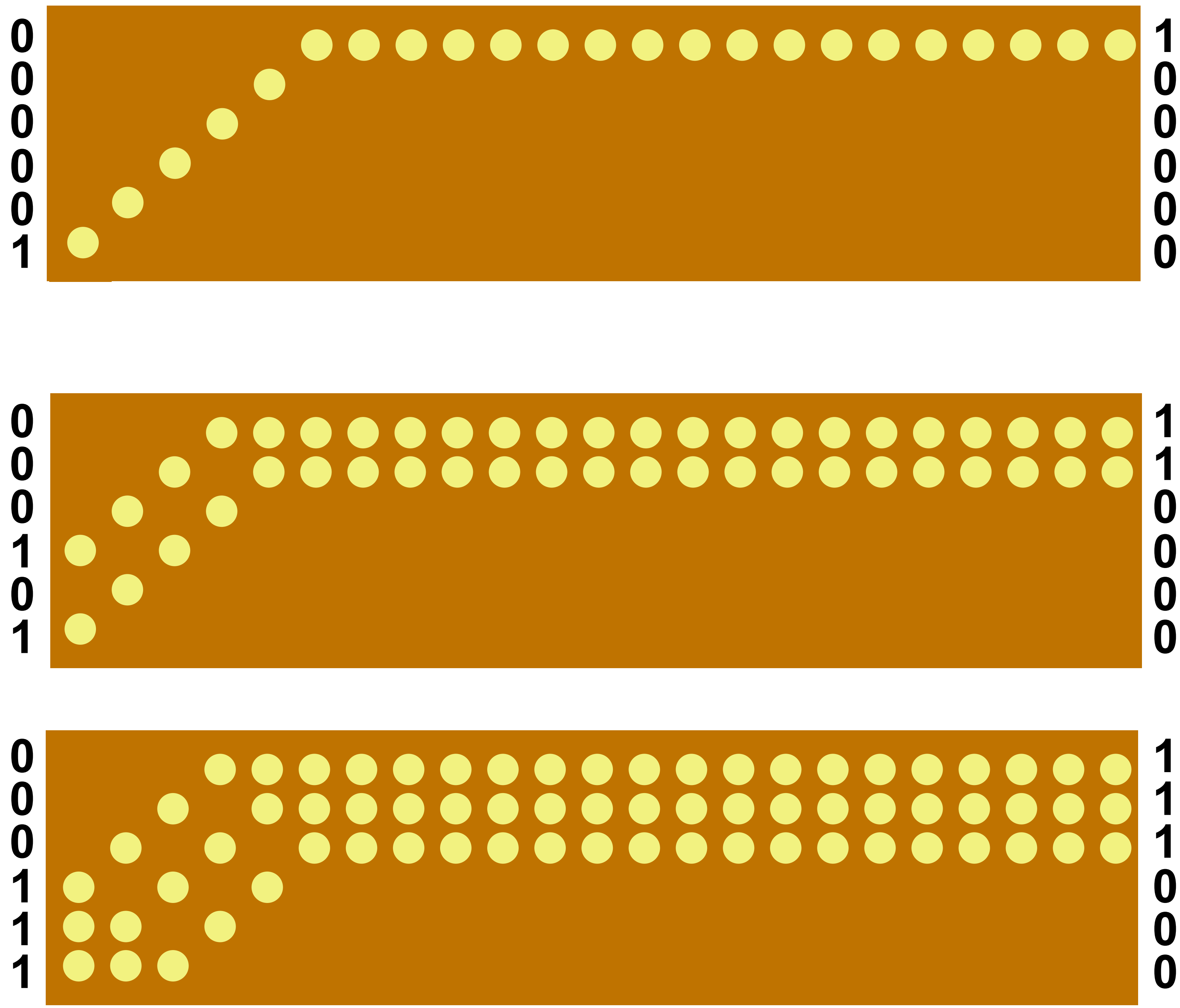
programmer



wall credit: Erik Klavins

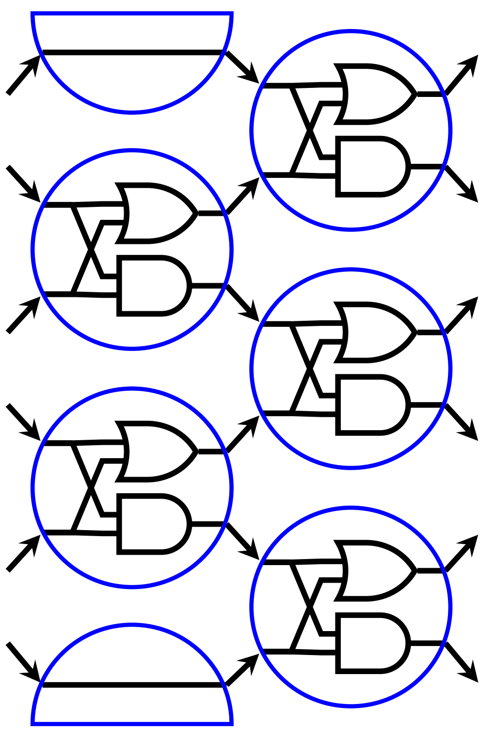
user

computation



Example circuit: "SORTING"

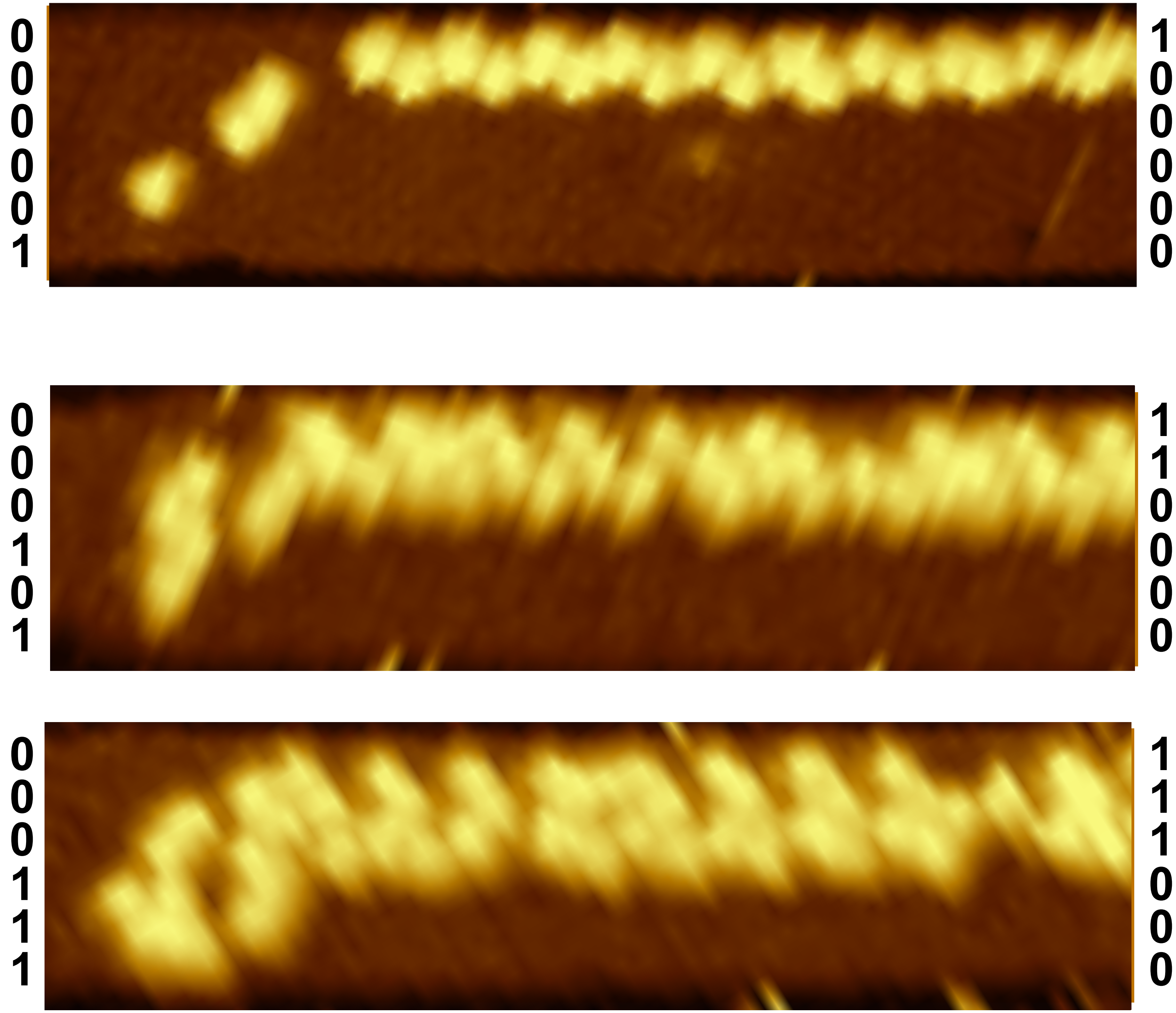
programmer



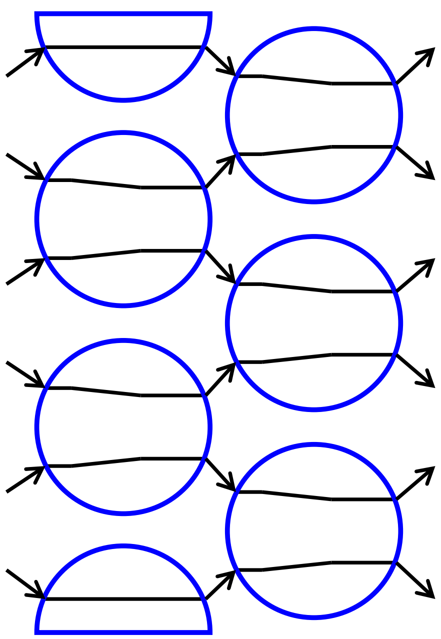
wall credit: Erik Klavins

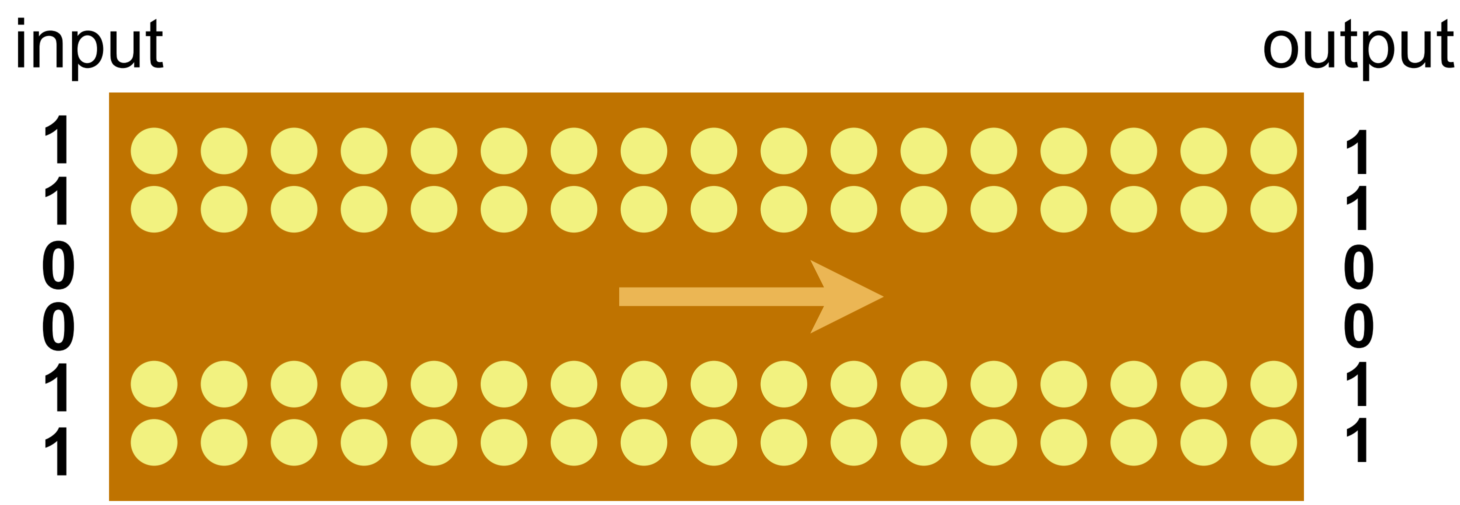
user

computation



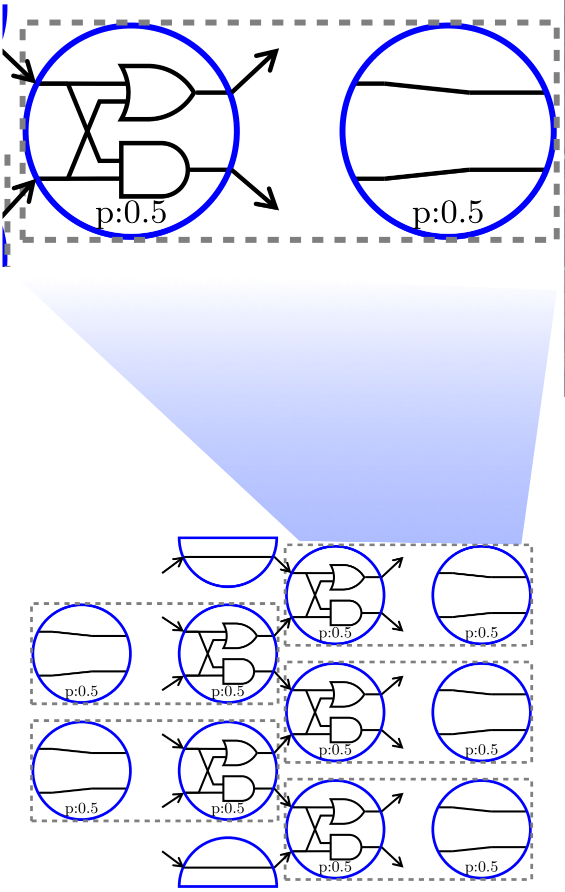
Example circuits: COPY bits to the right





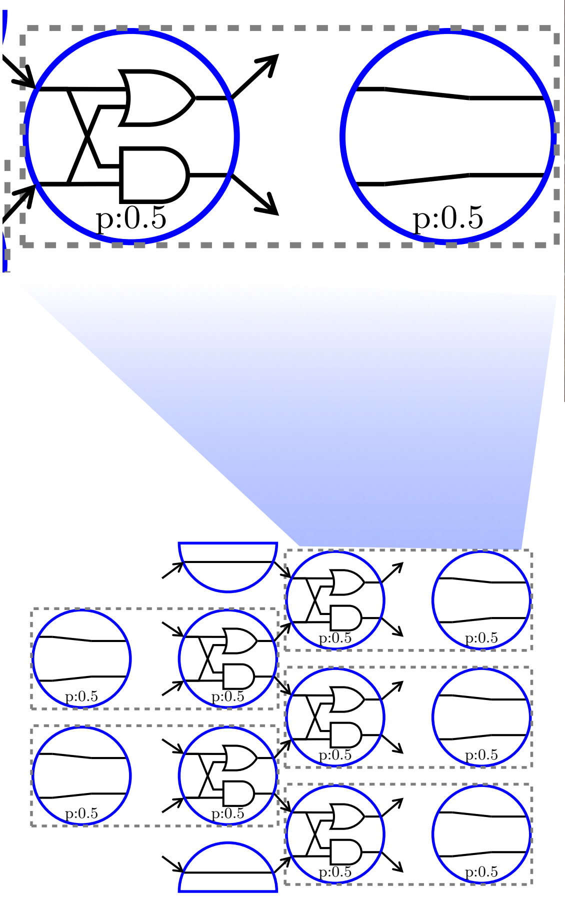
Example circuit: LAZYSORTING

programmer



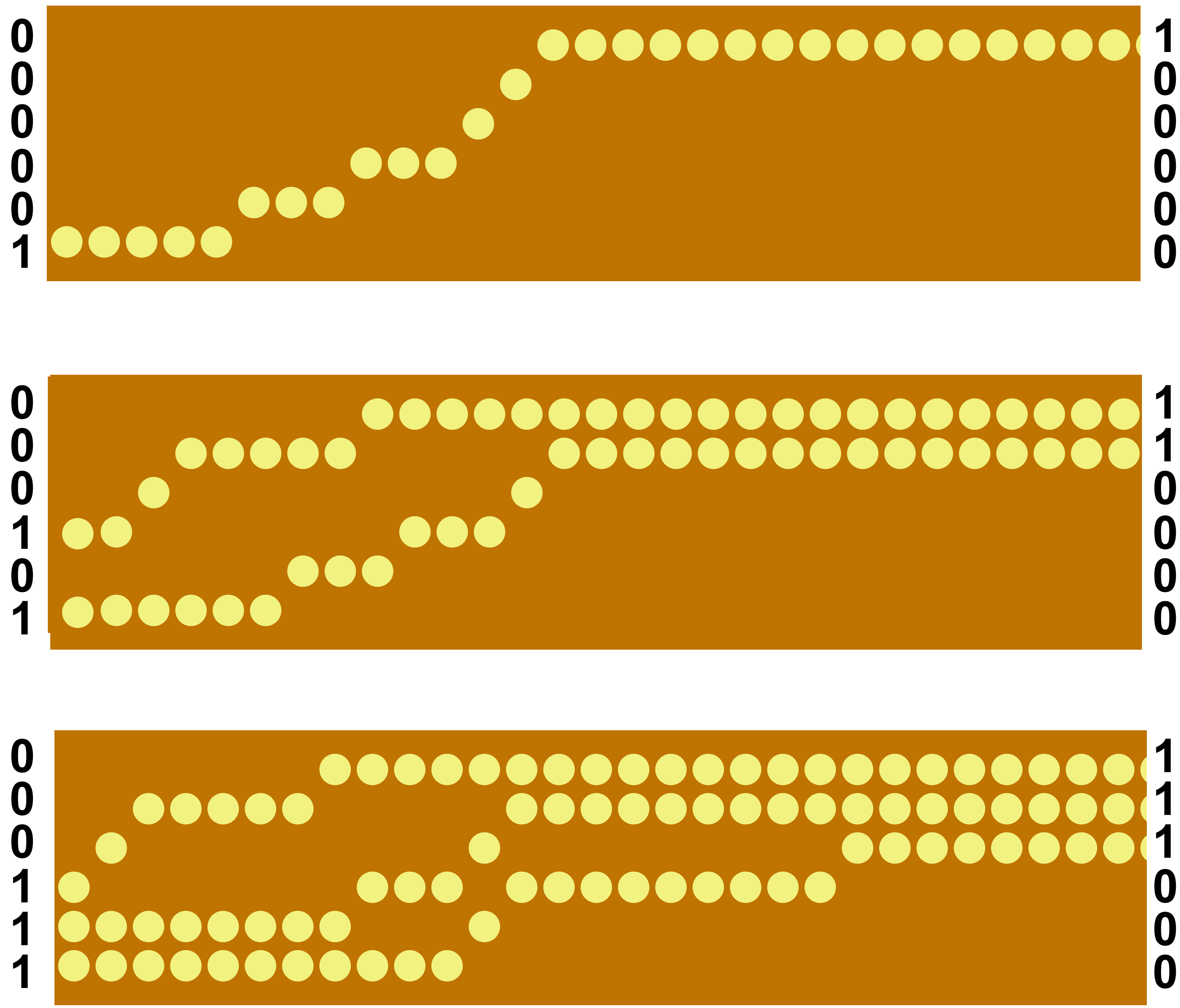
Example circuit: LAZYSORTING

programmer



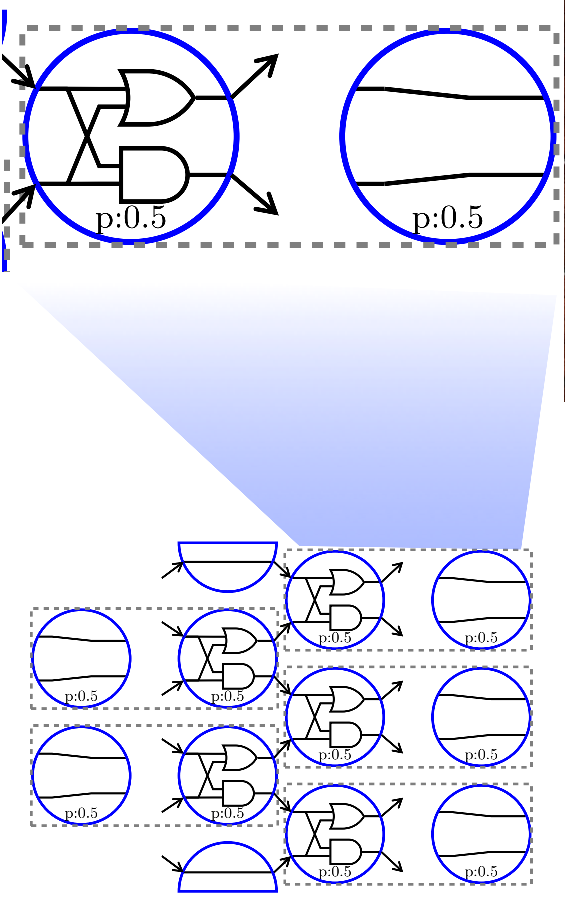
user

computation



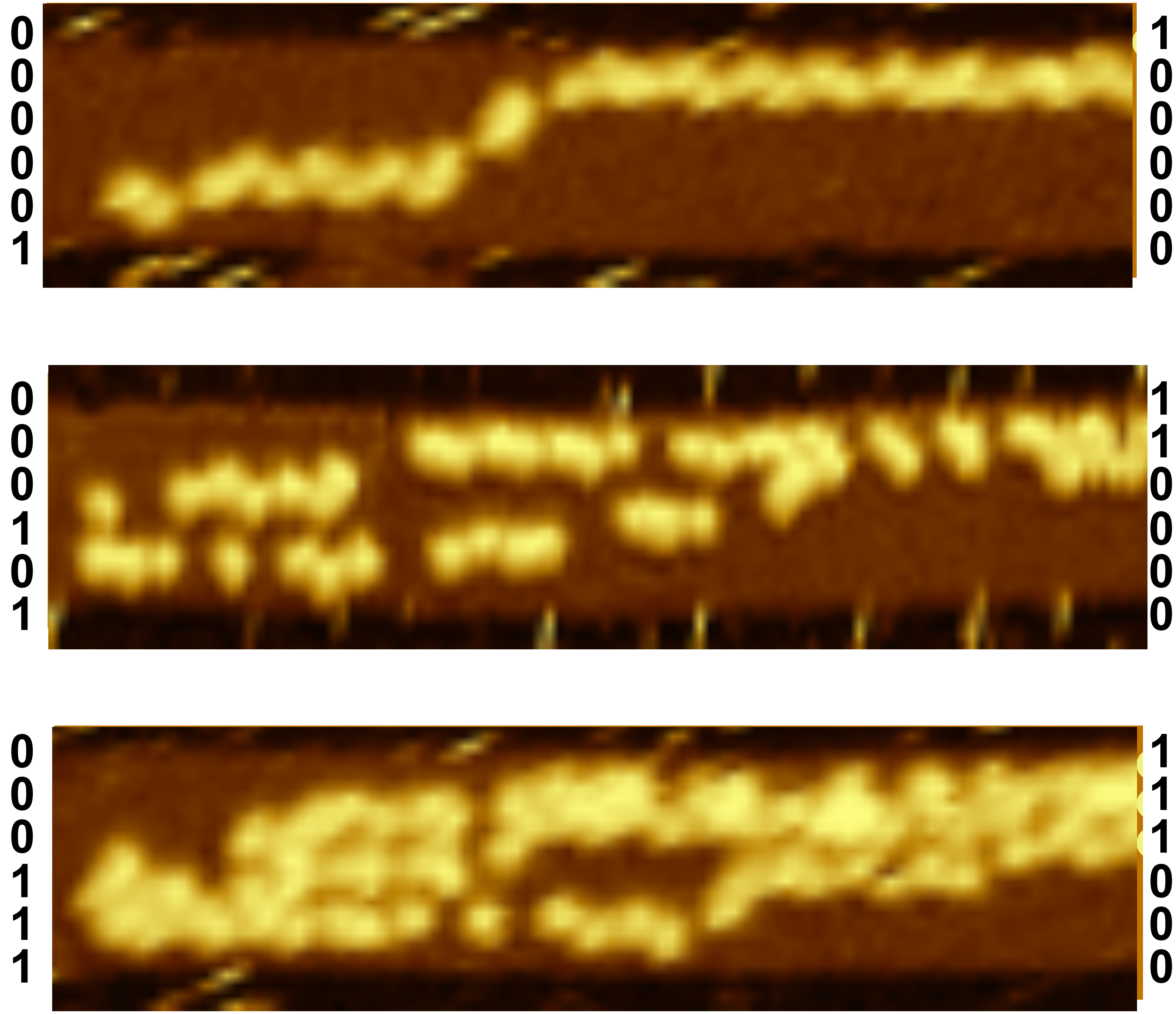
Example circuit: LAZYSORTING

programmer

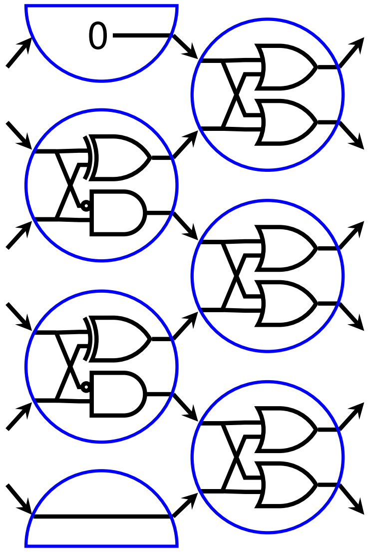


user

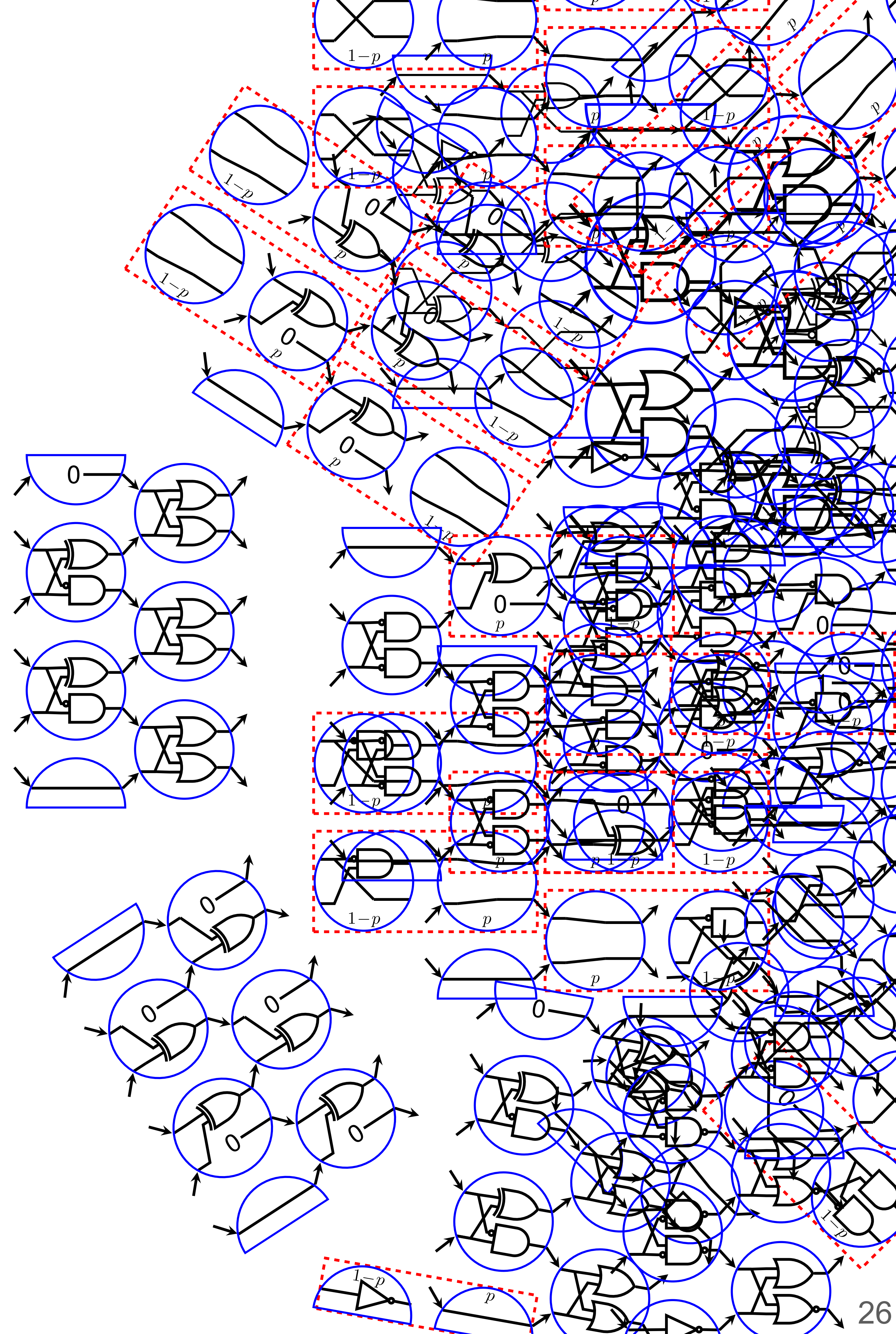
computation



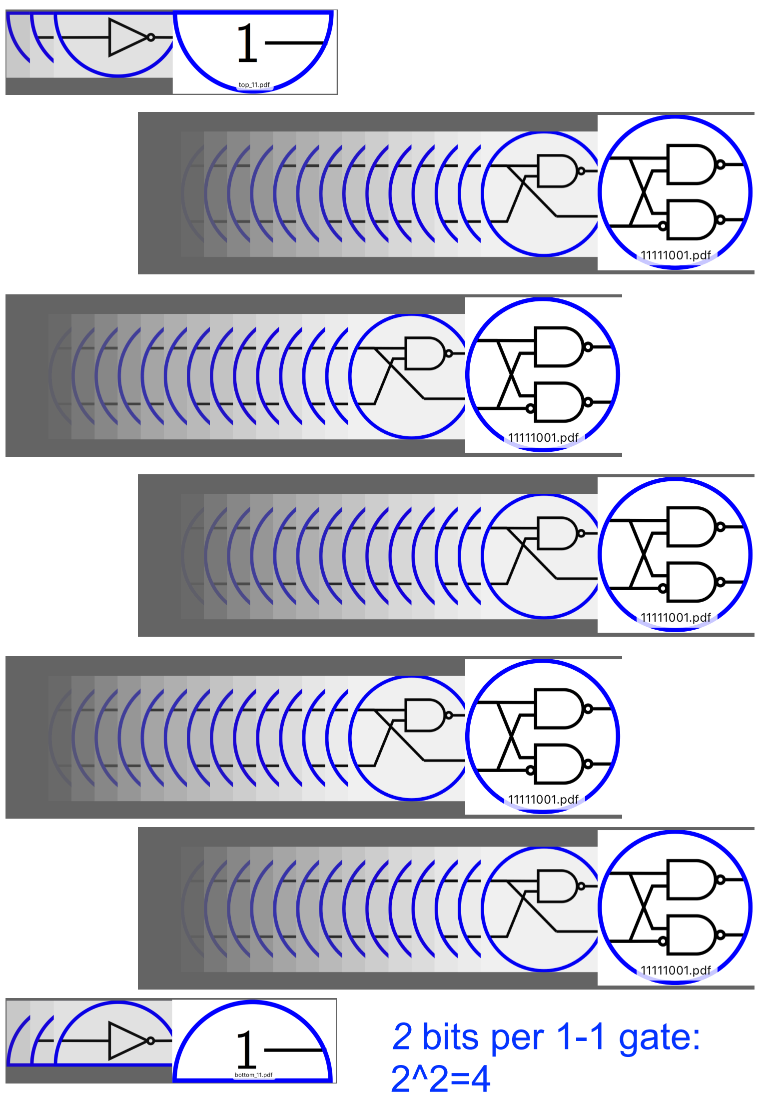
Which circuits to build?



Which circuits to build?



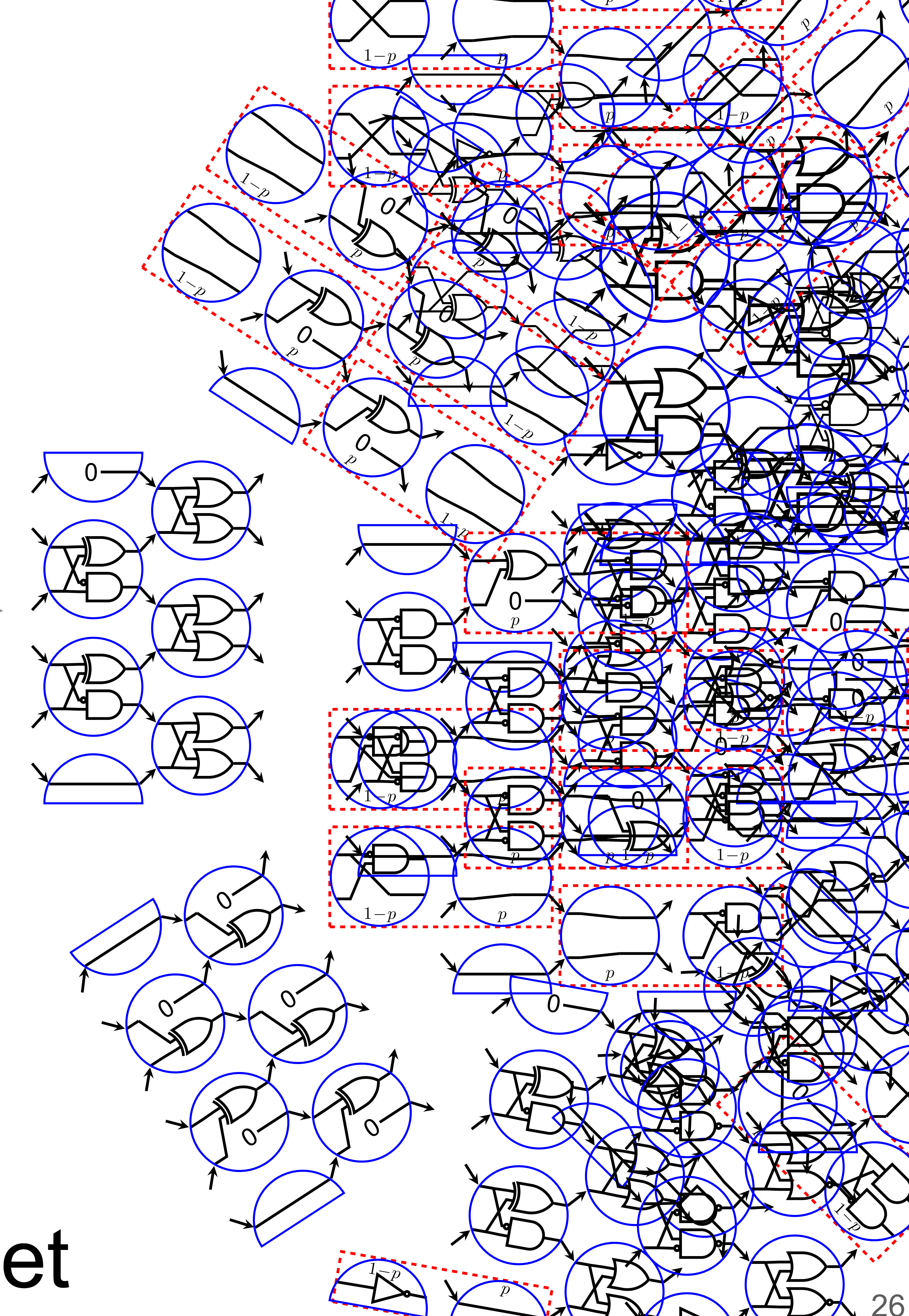
Which circuits to build?



8 bits per 2-2 gate:
2^8=256

2 bits per 1-1 gate:
2^2=4

1,288 gates that implement **any** 6-bit circuit



“Complete” 6-bit gate set

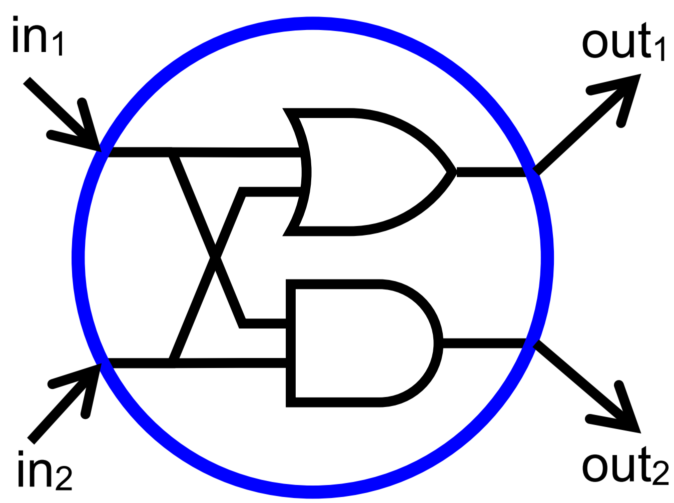
Structure

Theoretical circuit model

How it works: design and implementation

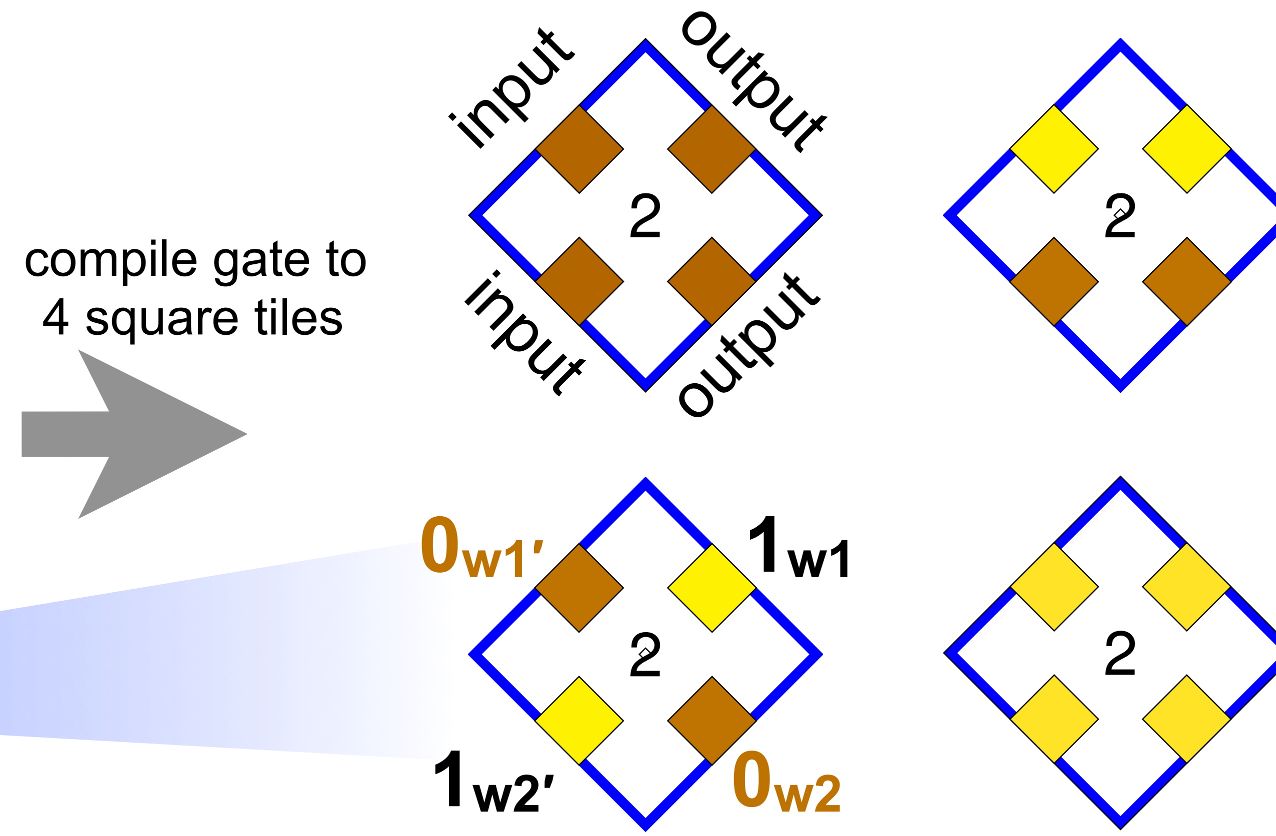
Experimental results

From circuits to square tiles

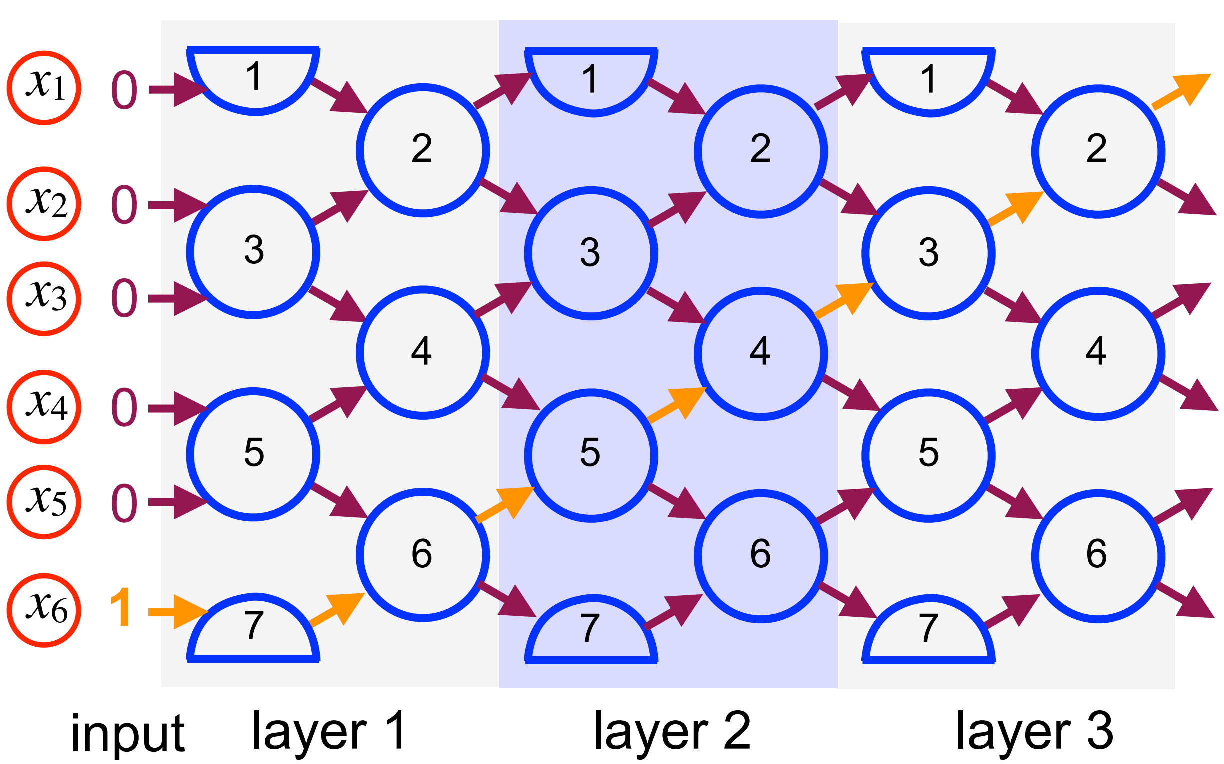


in_1	in_2	out_1	out_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

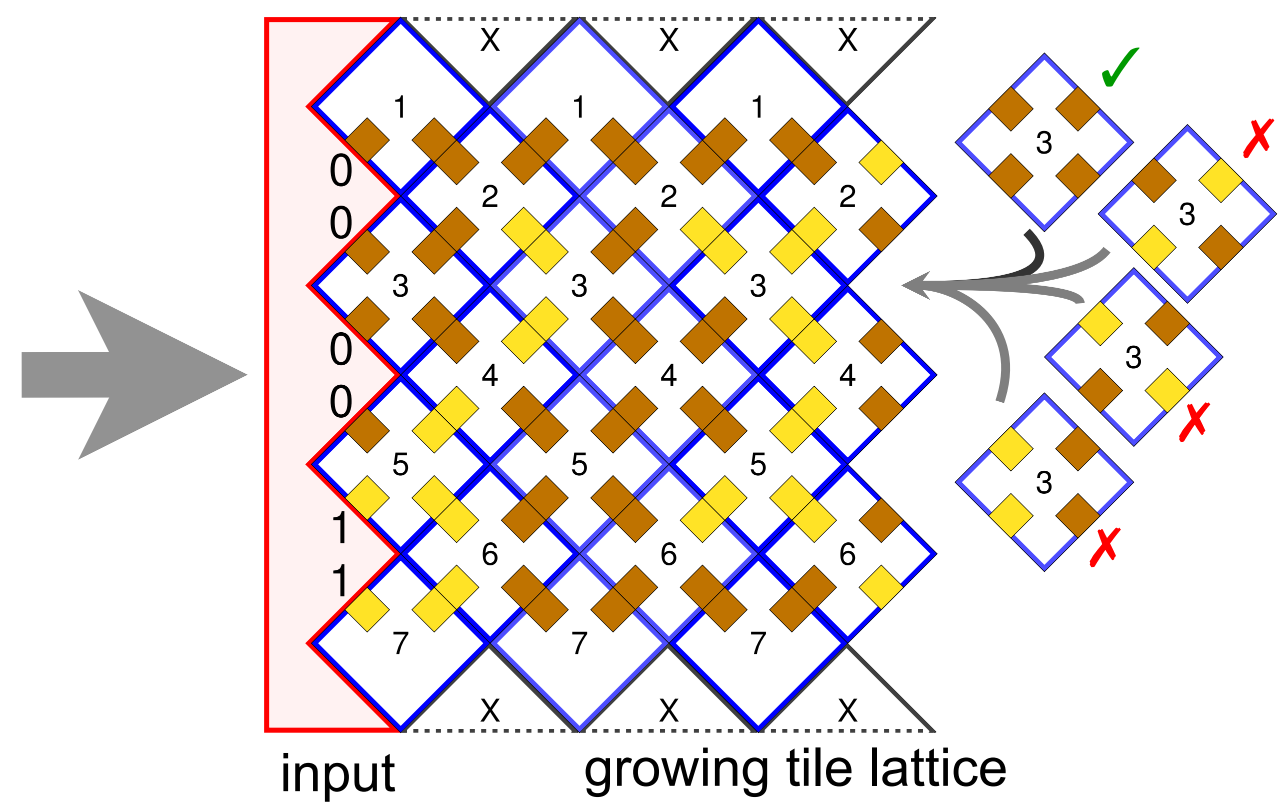
gate truth table



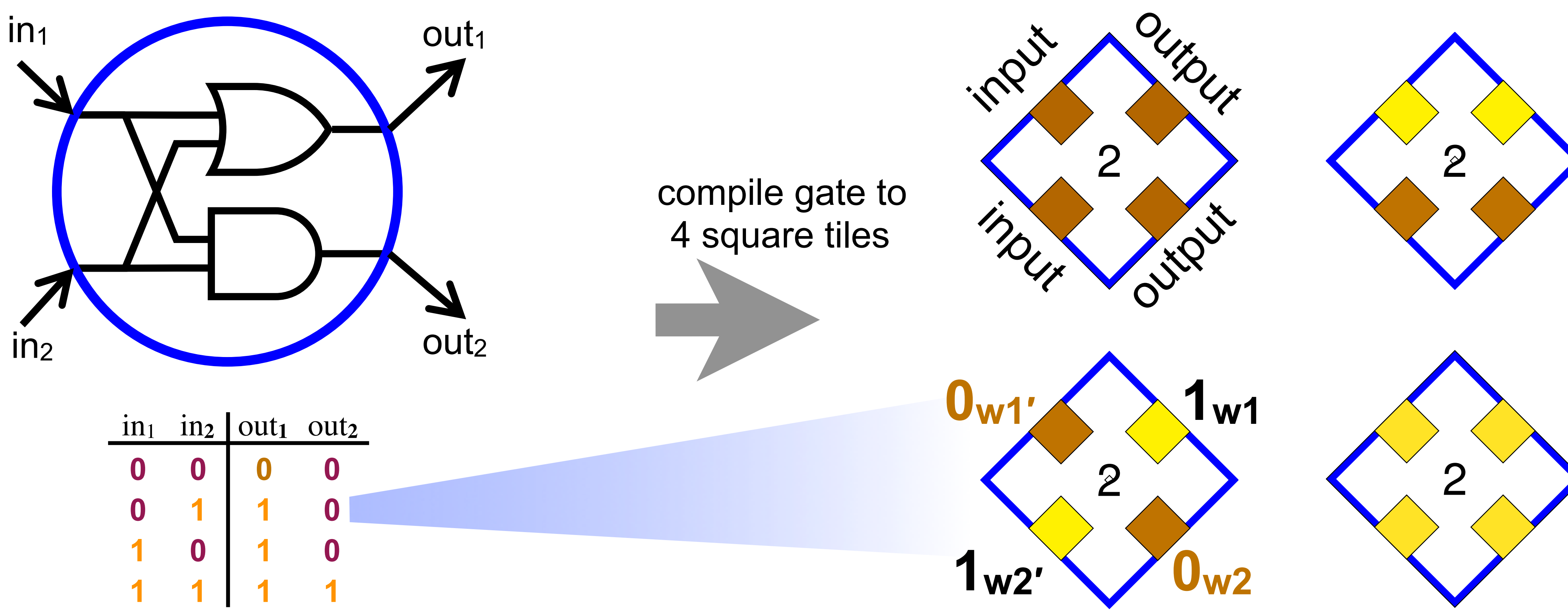
circuit computation



tile self-assembly



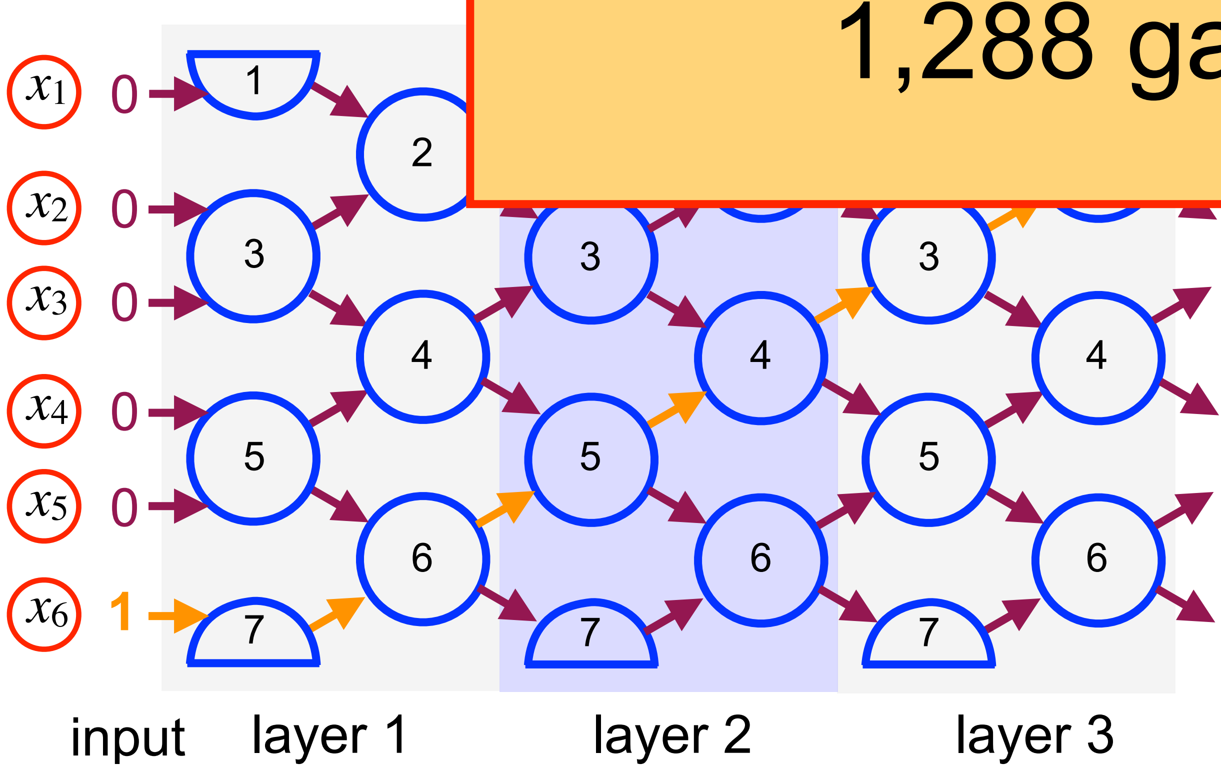
From circuits to square tiles



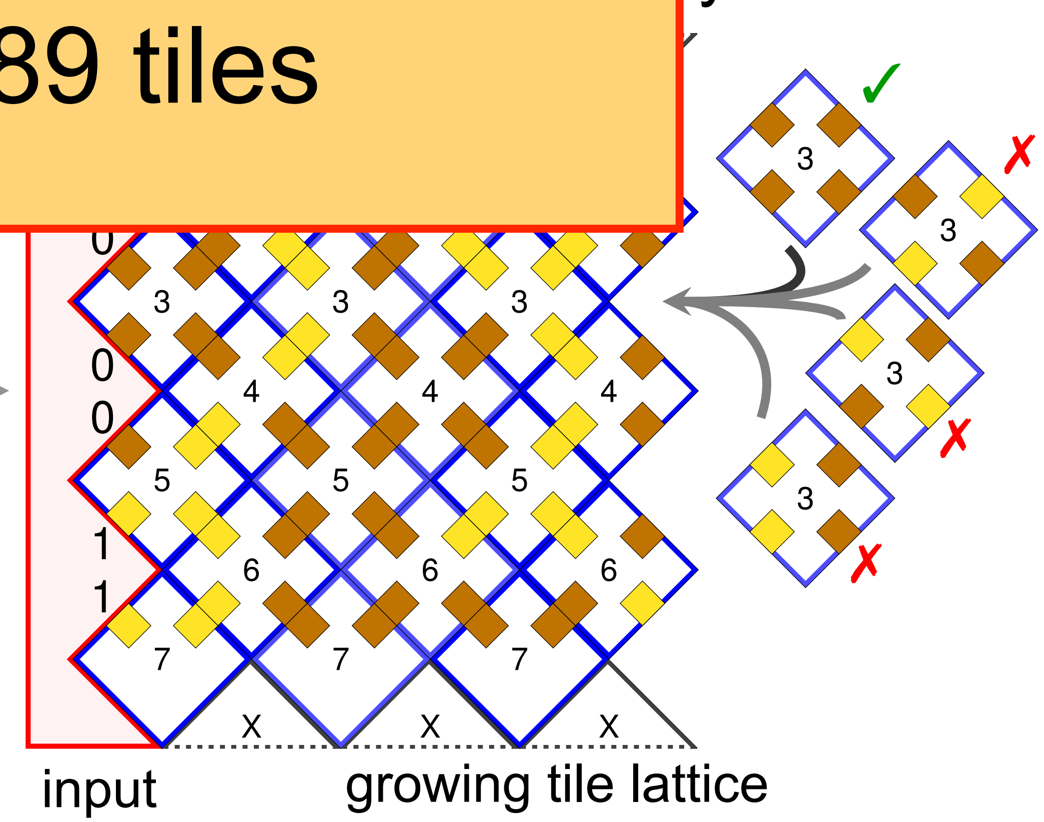
in_1	in_2	out_1	out_2
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

gate truth table

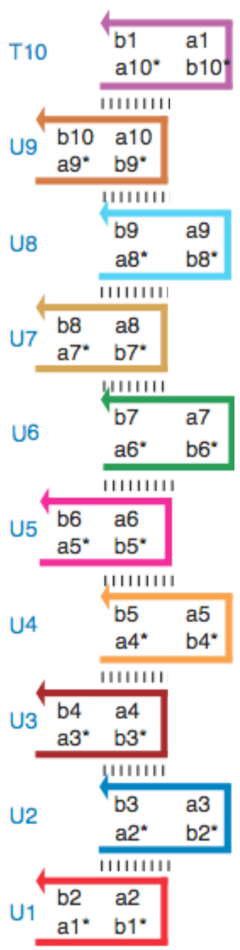
circuit c

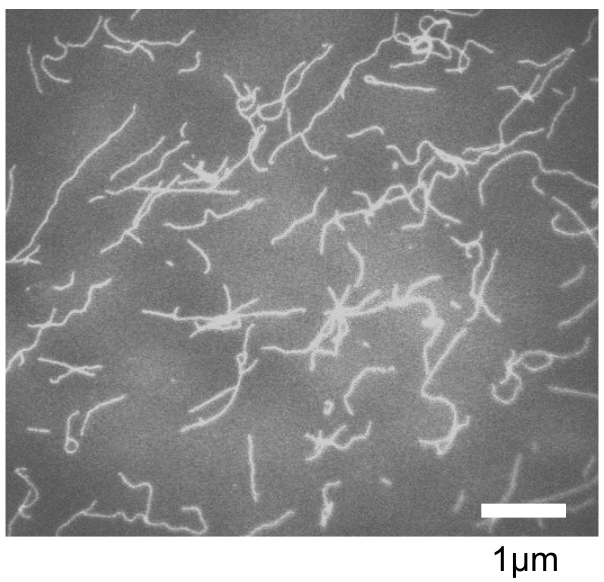


1,288 gates → 89 tiles



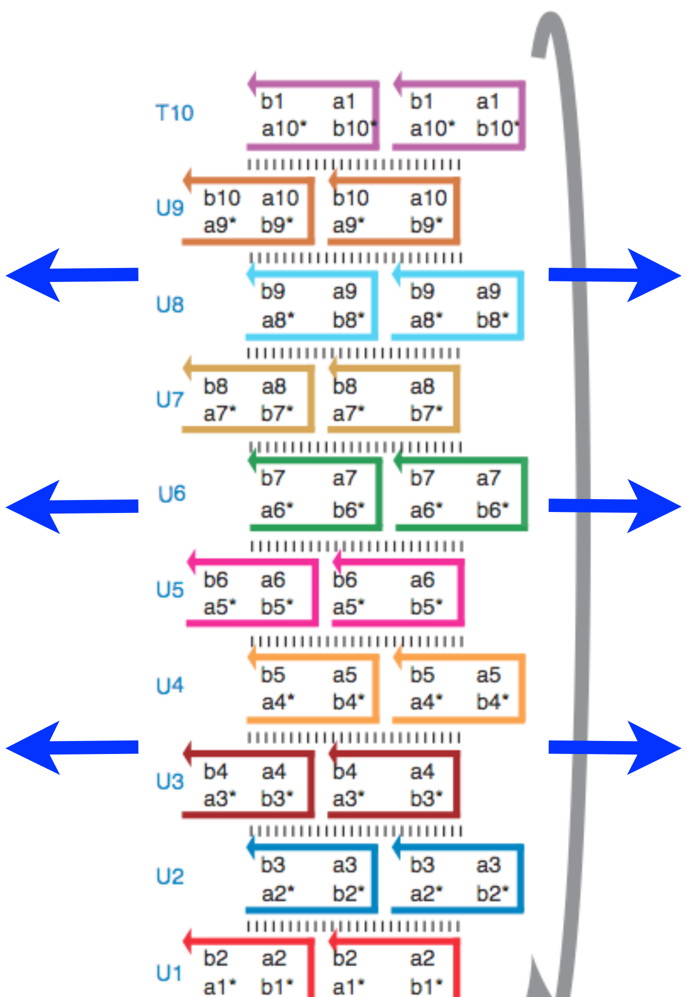
Single-stranded tile motif

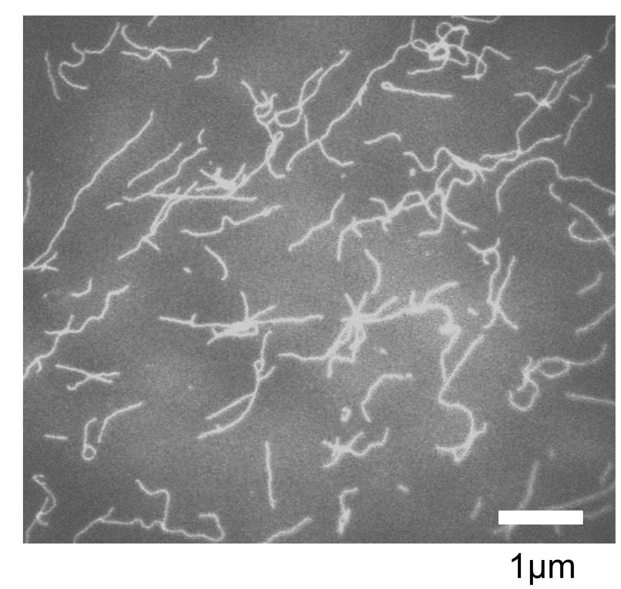




Yin, Hariadi, Sahu, Choi, Park, LaBean, Reif. Science. 2008

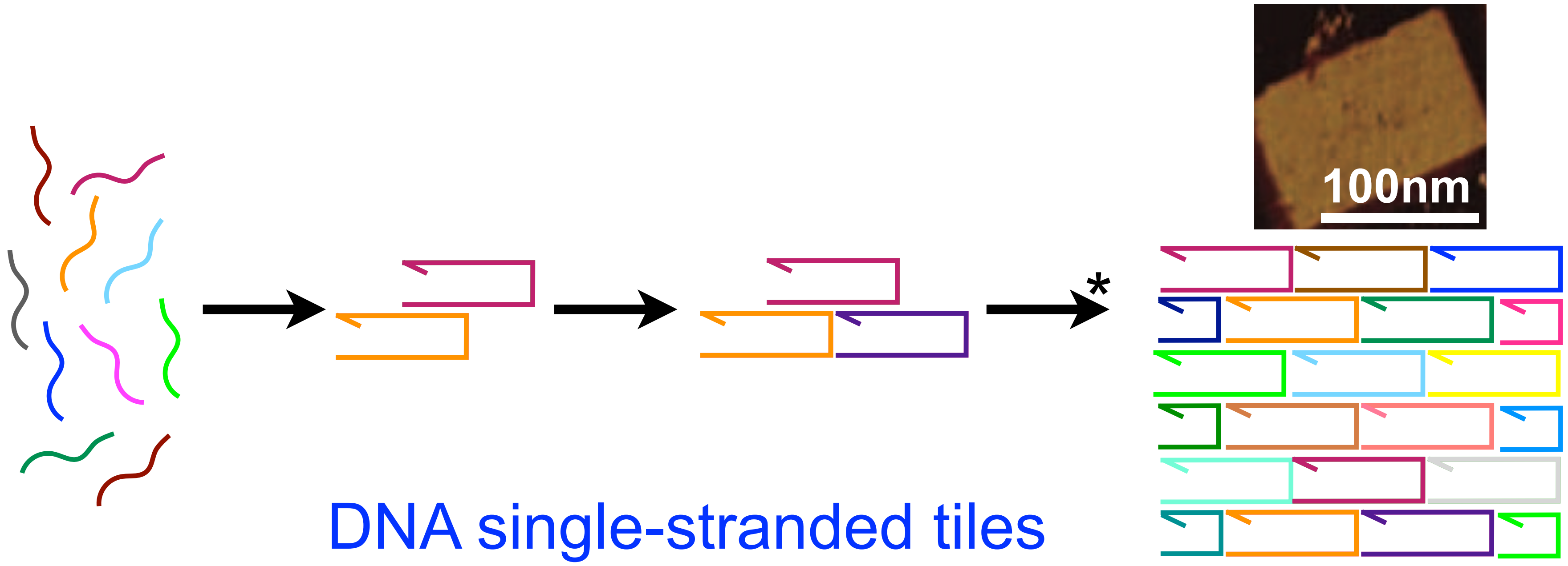
Single-stranded tile motif





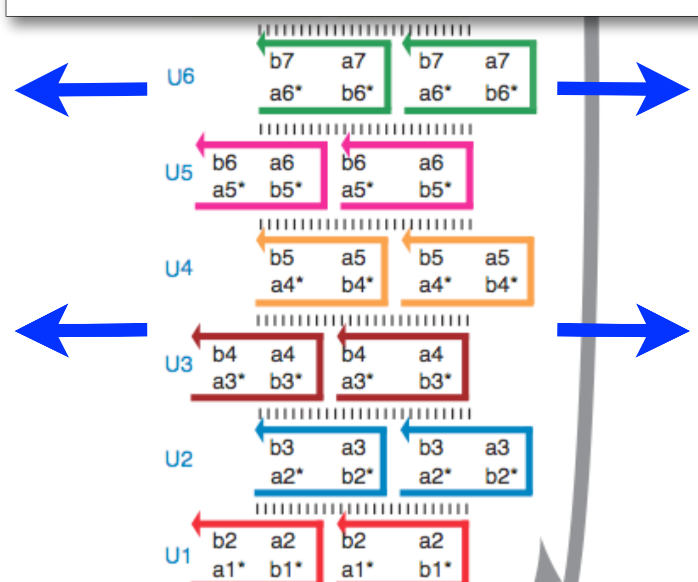
Yin, Hariadi, Sahu, Choi, Park, LaBean, Reif. Science. 2008

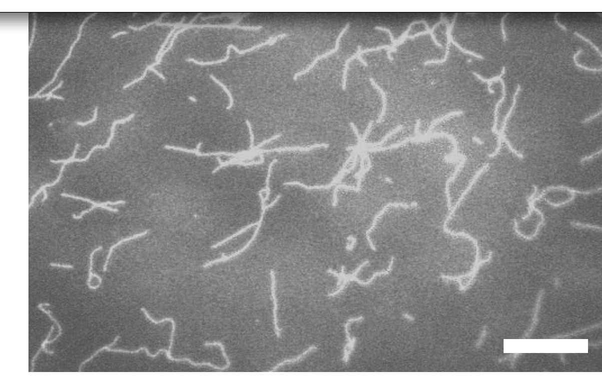
Single-stranded tile motif



DNA single-stranded tiles

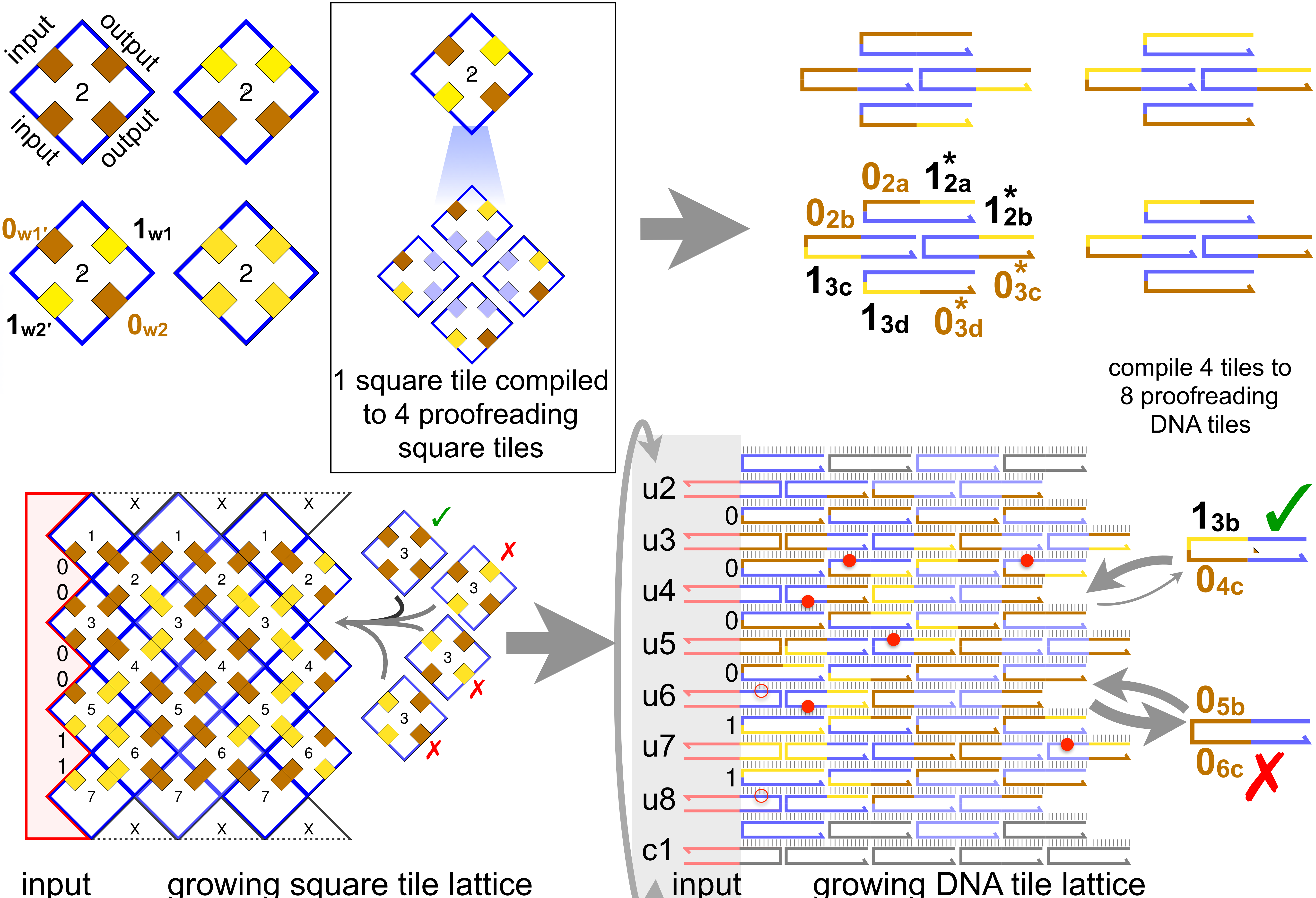
Wei, Dai, Yin. 2013 Nature



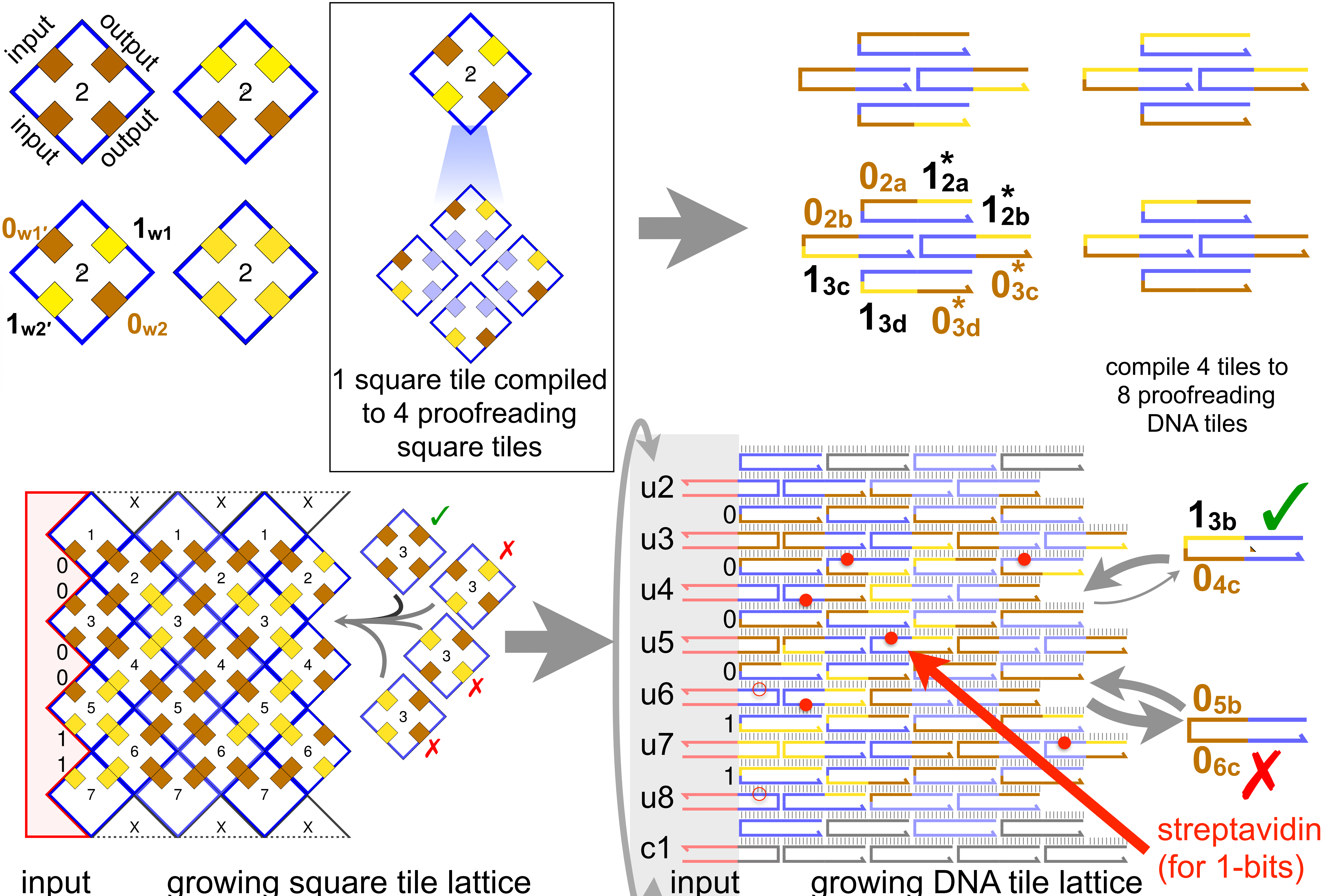


Yin, Hariadi, Sahu, Choi, Park, LaBean, Reif. Science. 2008

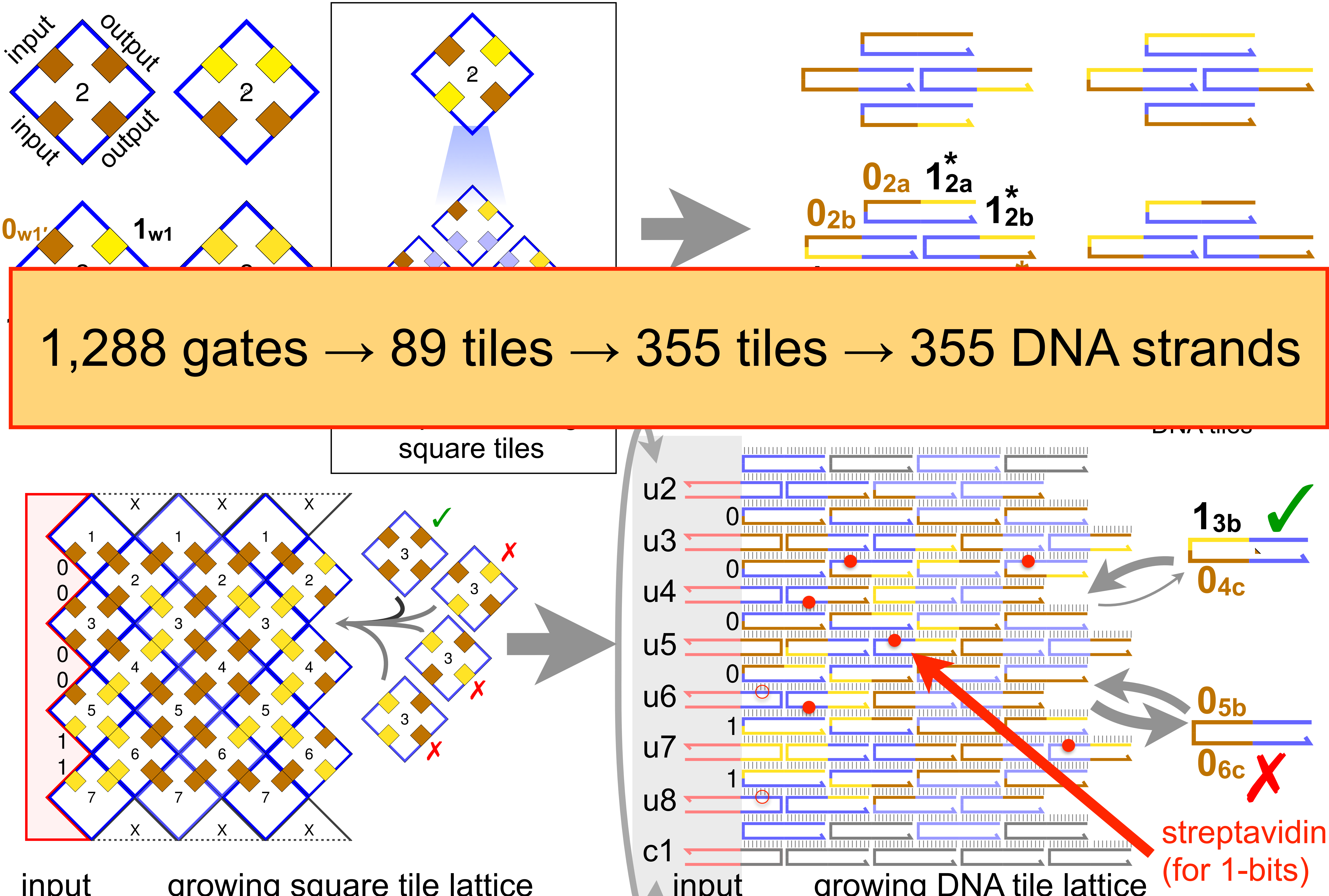
From square tiles to DNA single-stranded tiles



From square tiles to DNA single-stranded tiles



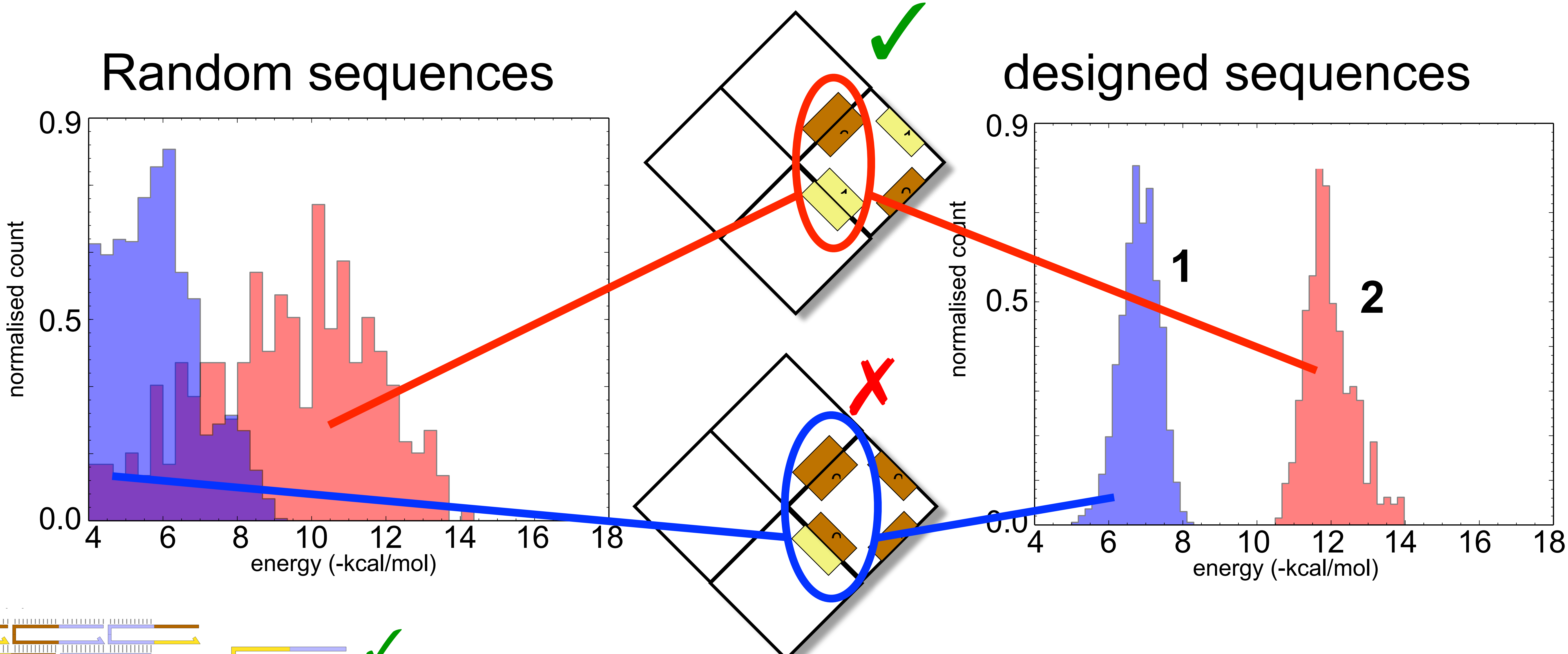
From square tiles to DNA single-stranded tiles

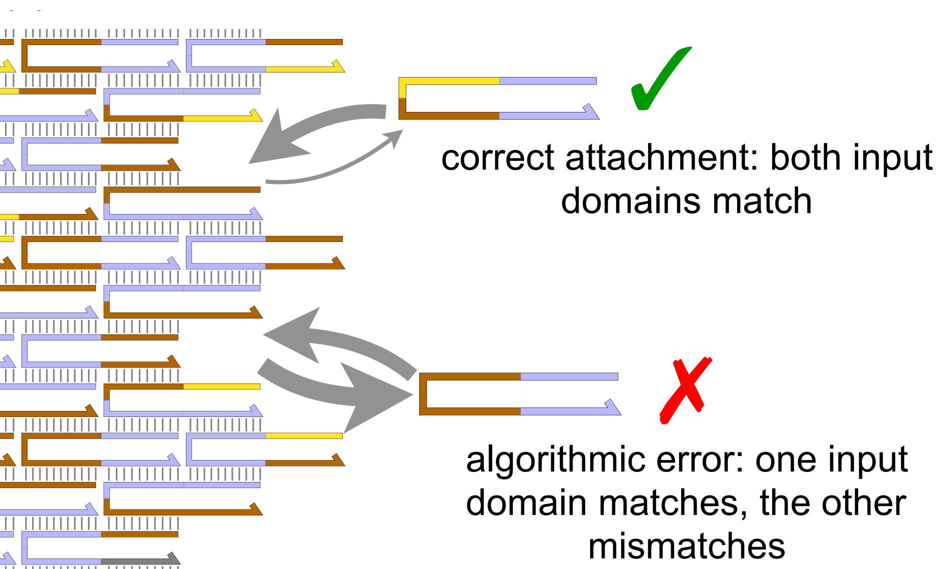


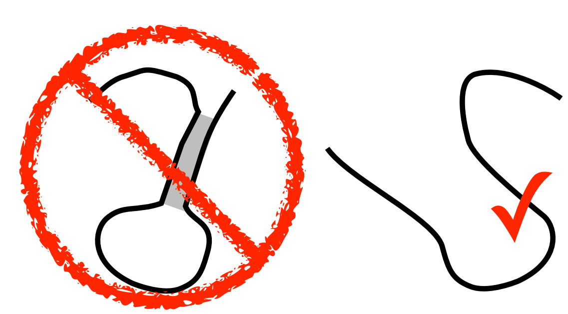
1,288 gates → 89 tiles → 355 tiles → 355 DNA strands

DNA sequence design

- Major challenge: We need to design DNA strands that bind when they should, and to not bind when the shouldn't

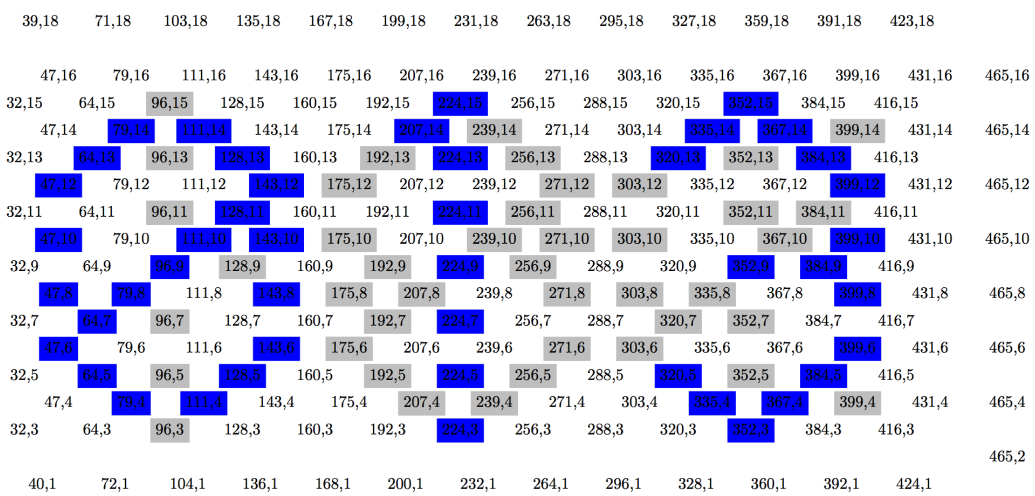






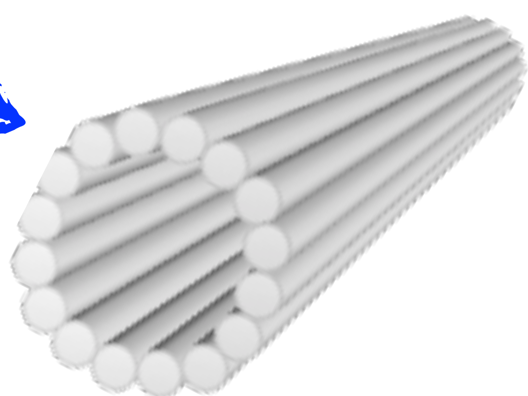
- **3. Strand ss**
- **4. Clean lattice**
- **5. Strand pairs**

Barcoded DNA origami seed

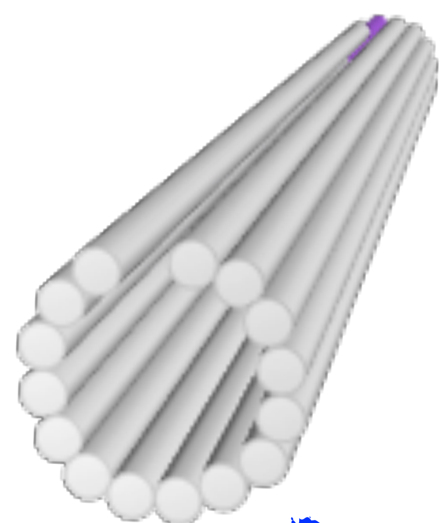


Choose which staples have biotin modifications

Form 16-helix tube



Unzip



add streptadividin & image on mica

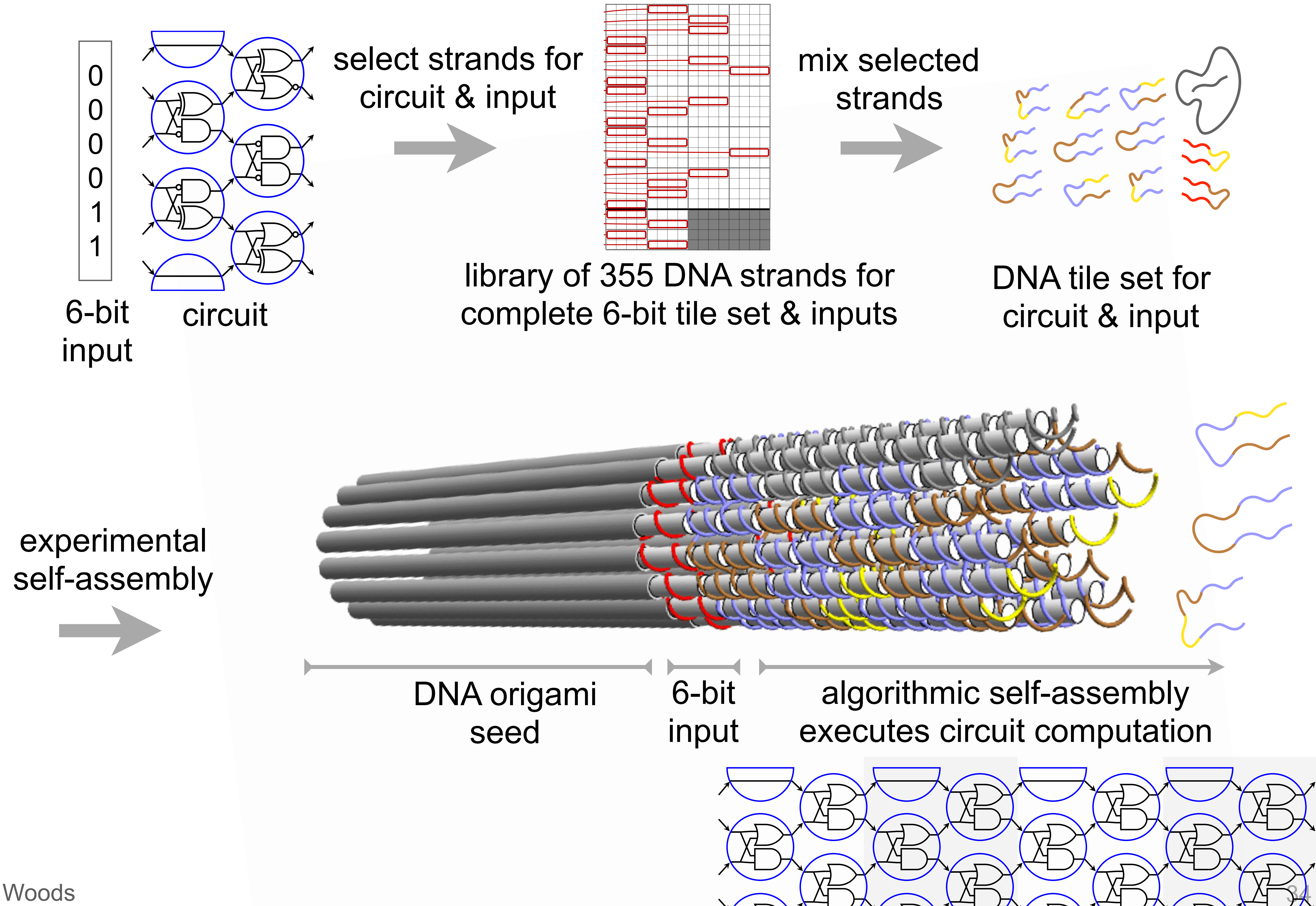
Structure

Theoretical circuit model

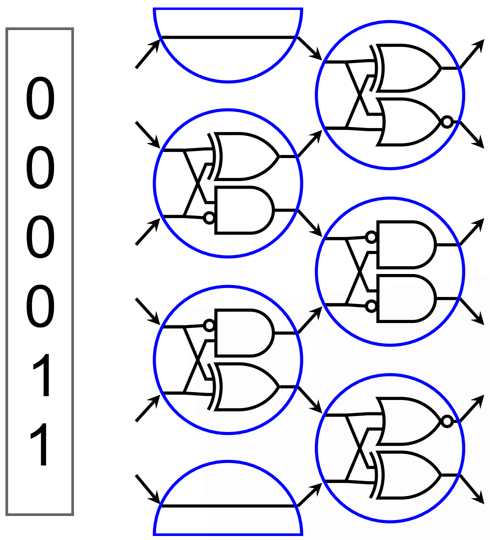
How it works: design and implementation

Experimental results

Schematic


0
0
0
0
1
1
6-bit input
circuit
select strands for circuit & input
library of 355 DNA strands for complete 6-bit tile set & inputs
mix selected strands
DNA tile set for circuit & input
experimental self-assembly
DNA origami seed
6-bit input
algorithmic self-assembly executes circuit computation

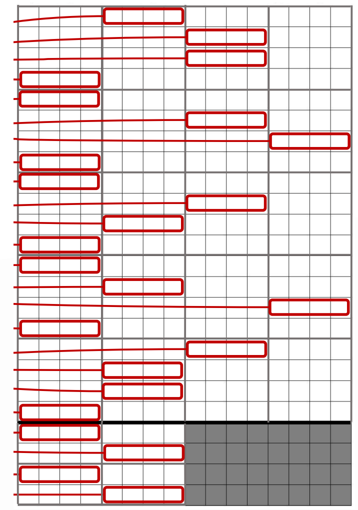
Schematic



6-bit input

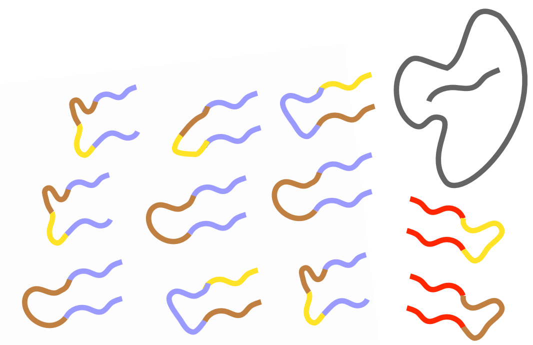
circuit

select strands for circuit & input



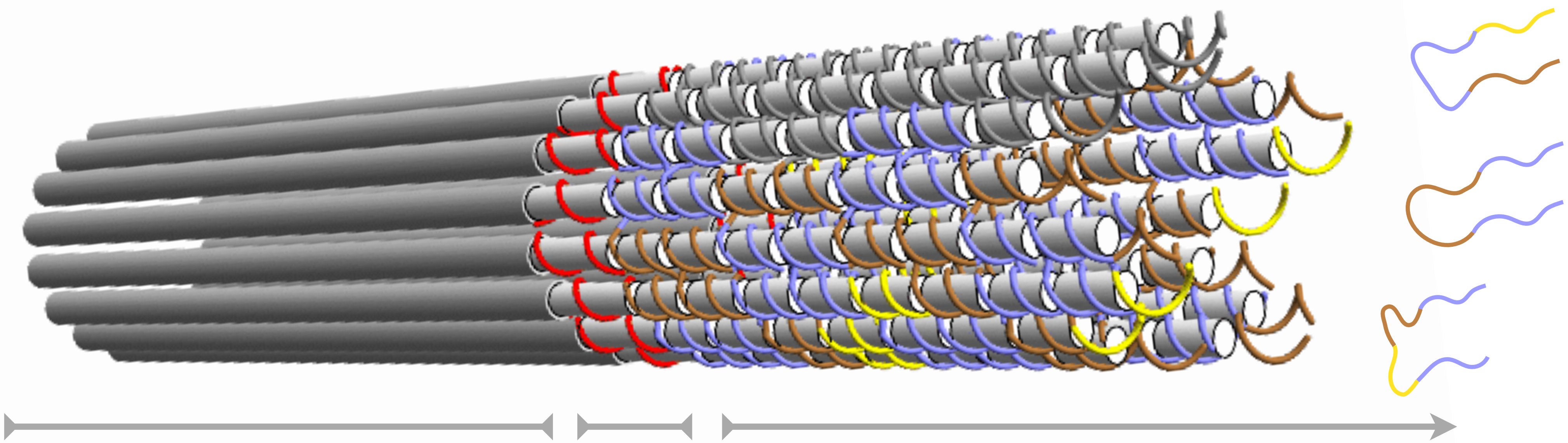
library of 355 DNA strands for complete 6-bit tile set & inputs

mix selected strands



DNA tile set for circuit & input

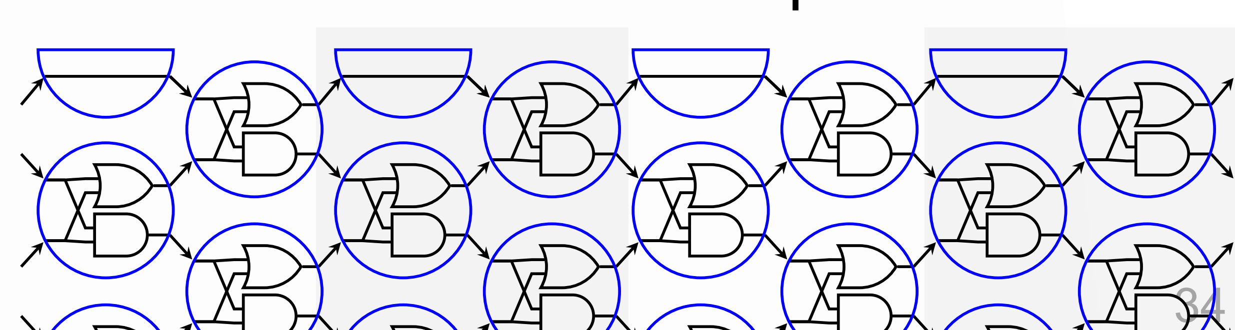
experimental self-assembly



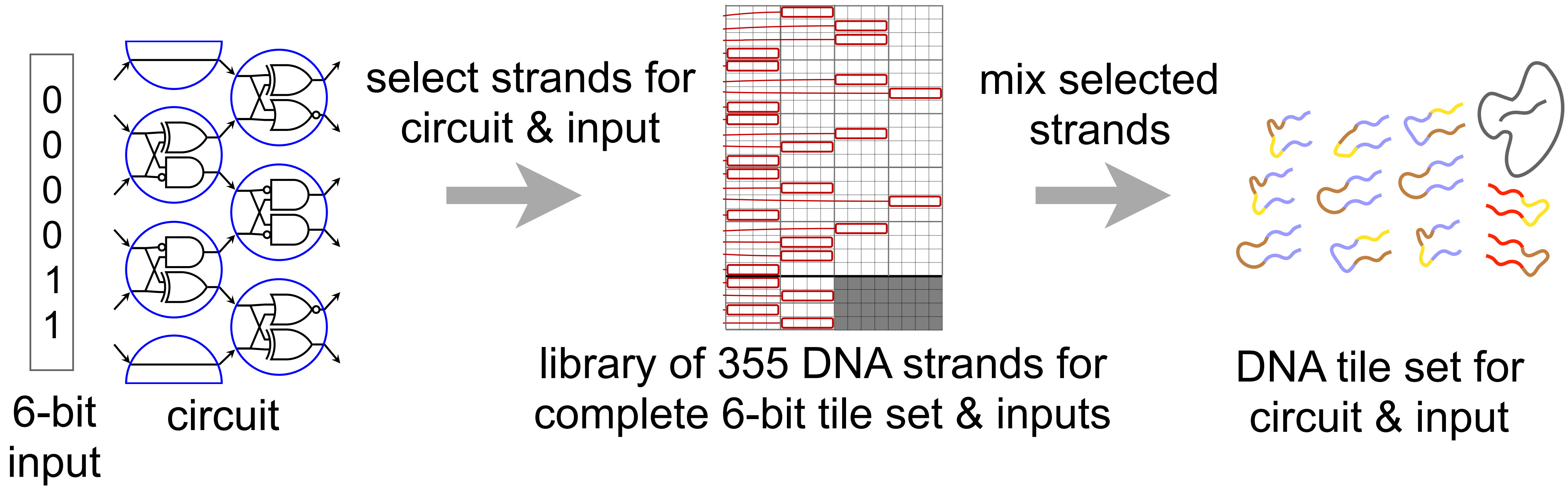
DNA origami seed

6-bit input

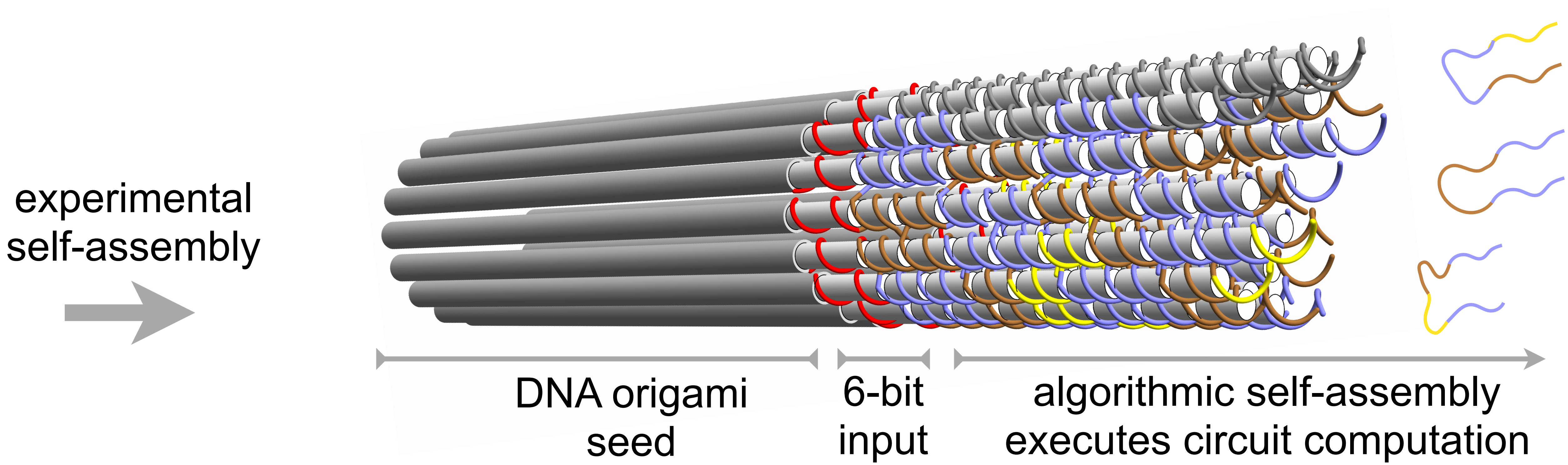
algorithmic self-assembly executes circuit computation



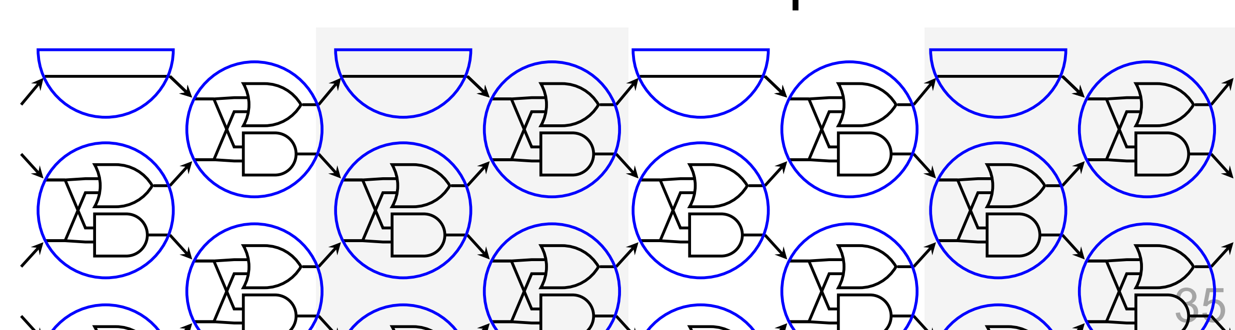
Schematic



0
0
0
0
1
1
6-bit input
circuit
select strands for circuit & input
library of 355 DNA strands for complete 6-bit tile set & inputs
mix selected strands
DNA tile set for circuit & input

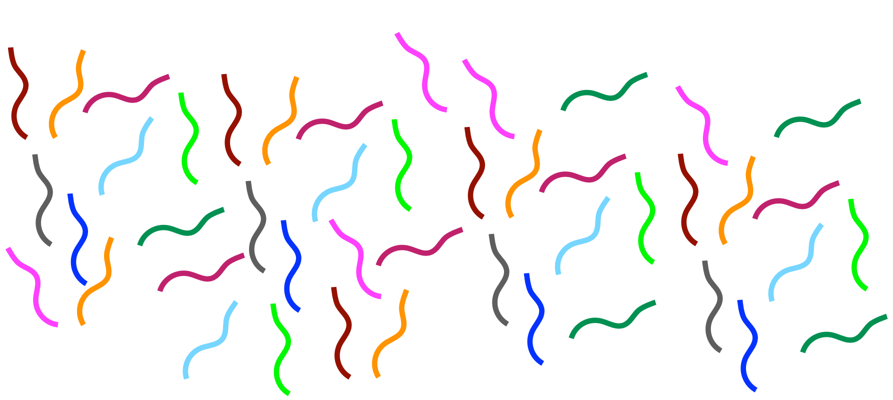


experimental self-assembly
DNA origami seed
6-bit input
algorithmic self-assembly executes circuit computation

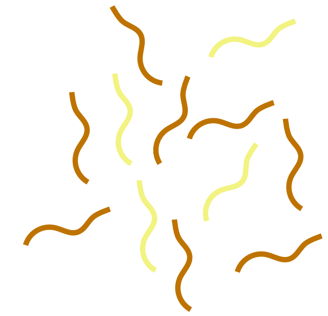


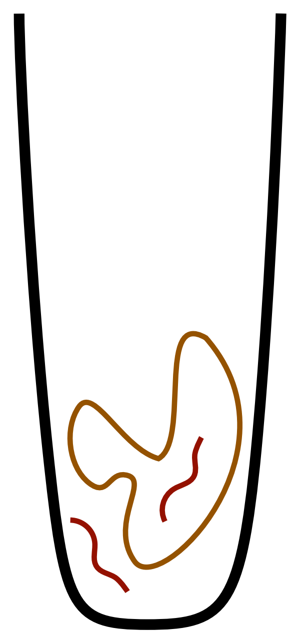
An example experiment: SORTING

355 tiles that implement **any 6-bit circuit**



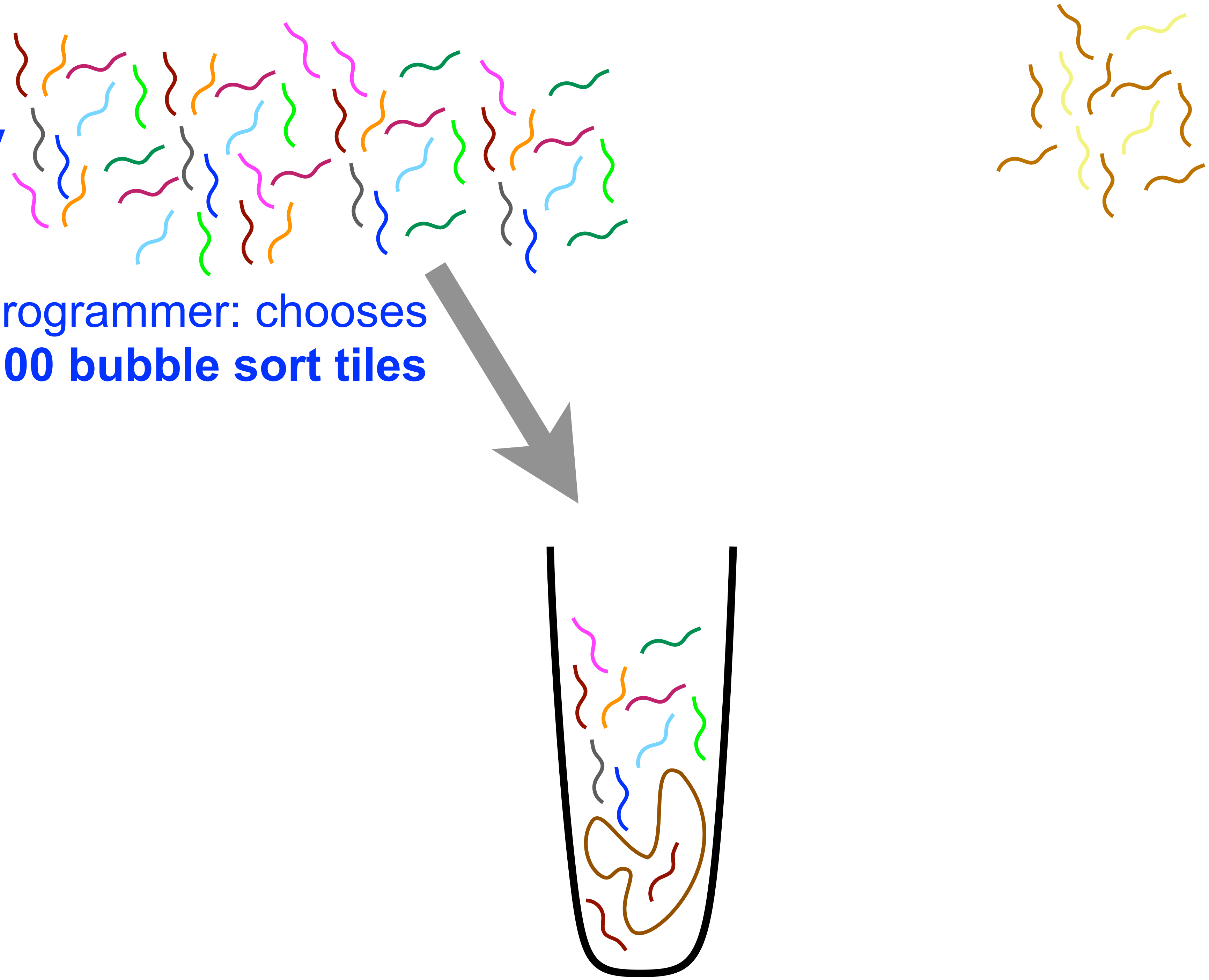
6-bit **input** strands



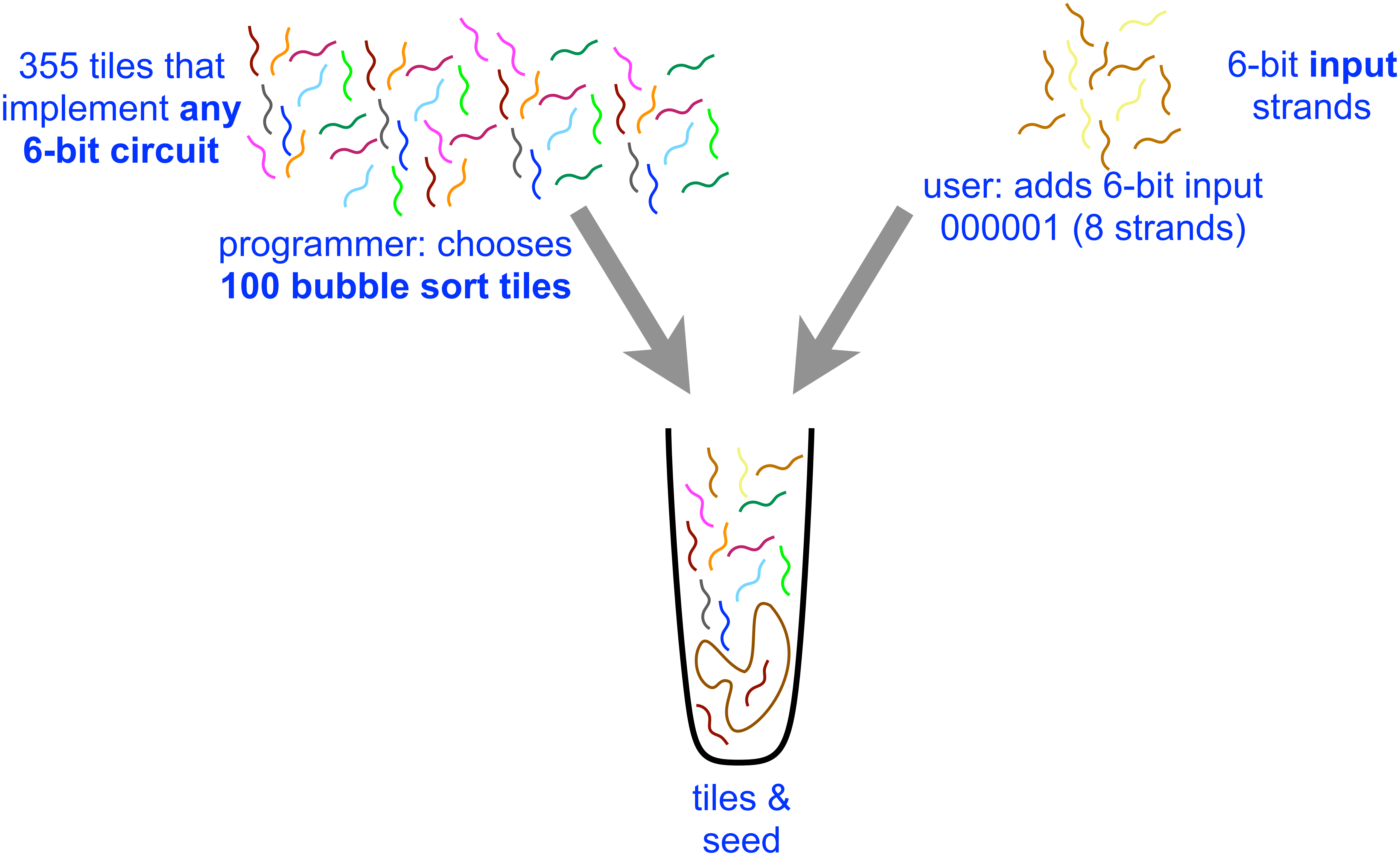


tiles & seed

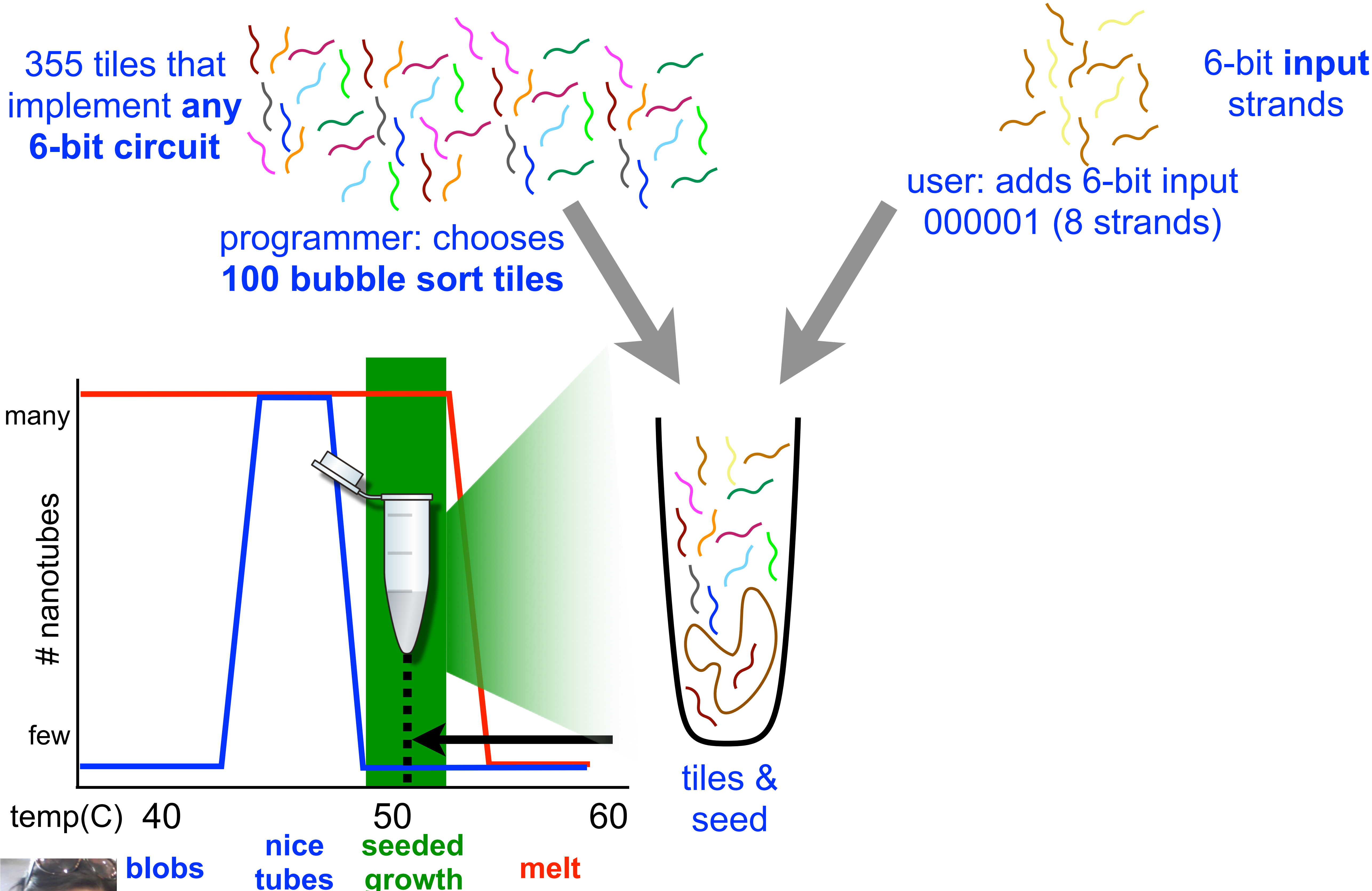
An example experiment: SORTING



An example experiment: SORTING

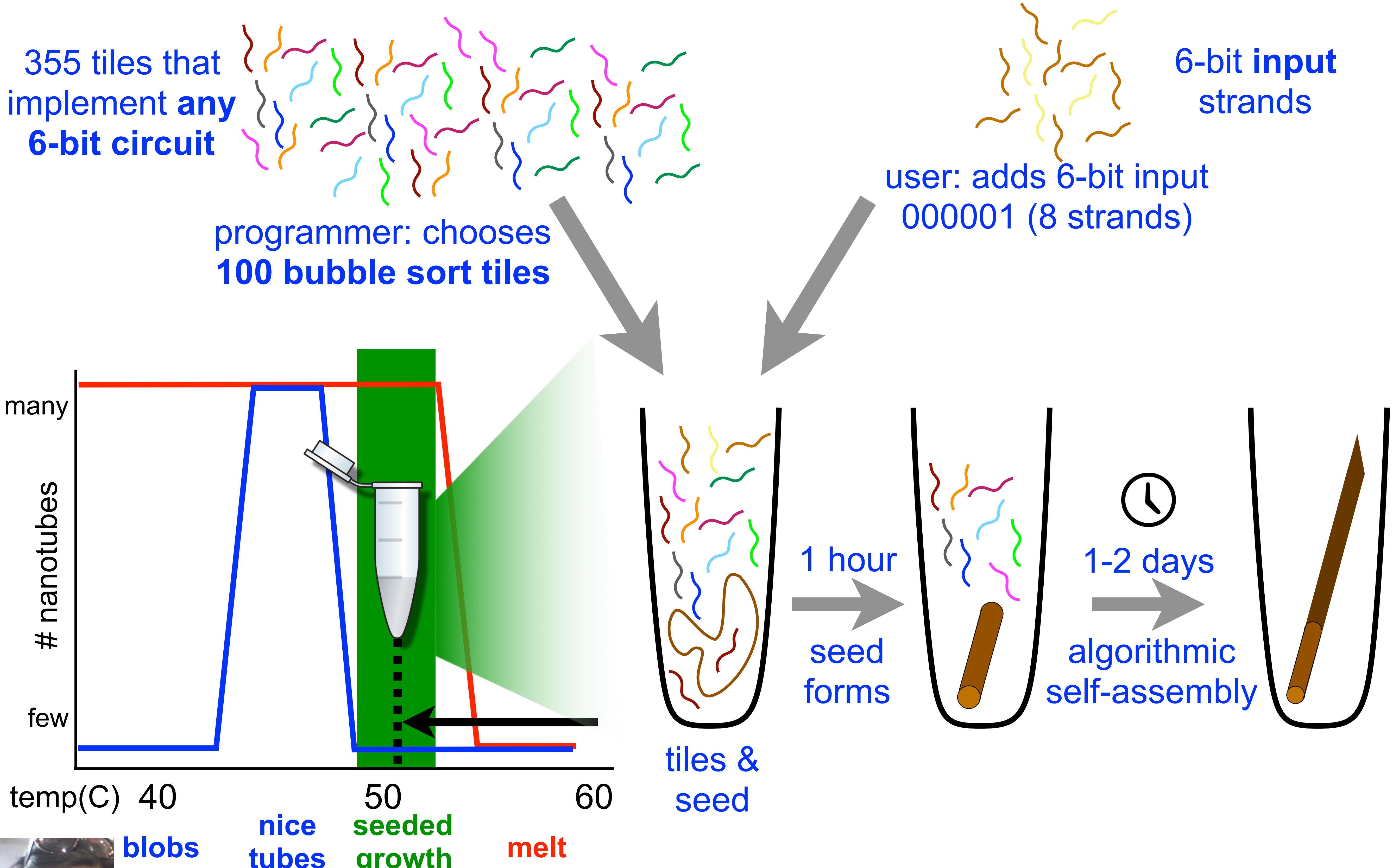


An example experiment: SORTING



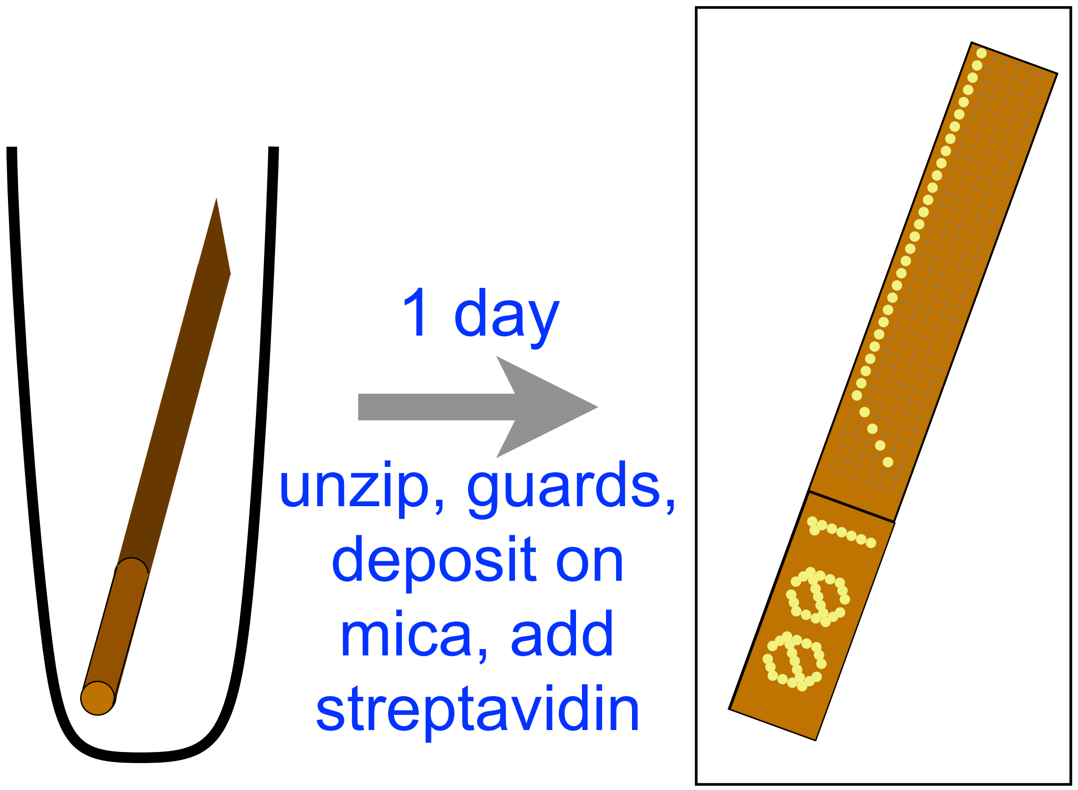
Joy Hui

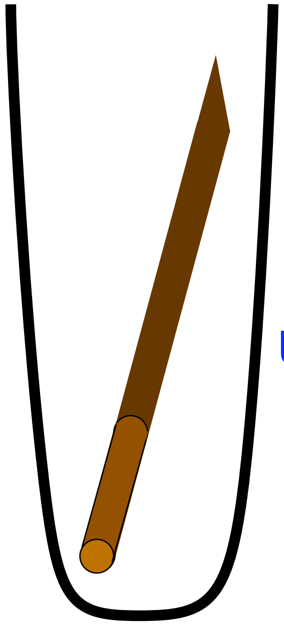
An example experiment: SORTING



Joy Hui

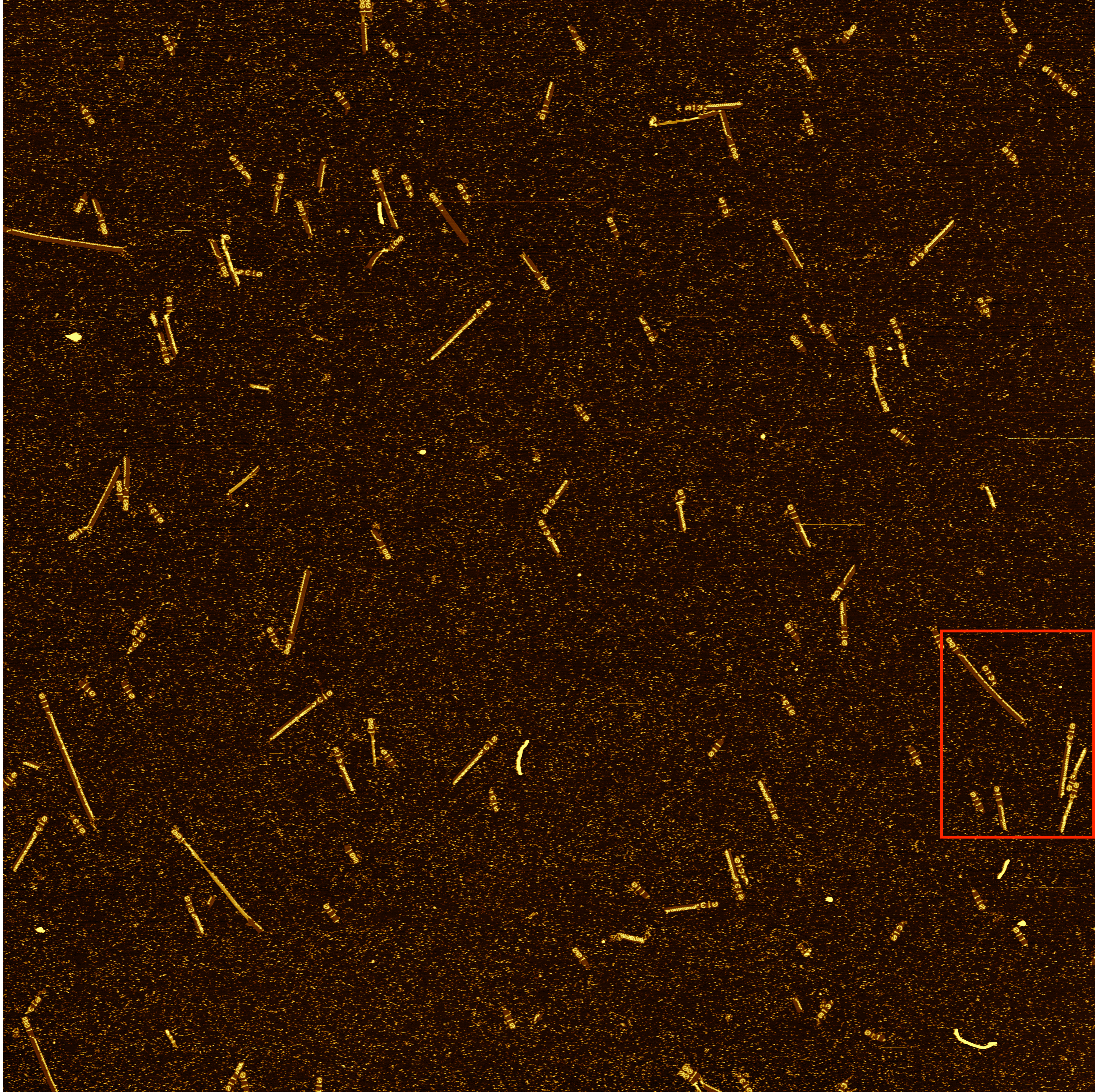
An example experiment: SORTING





1 day

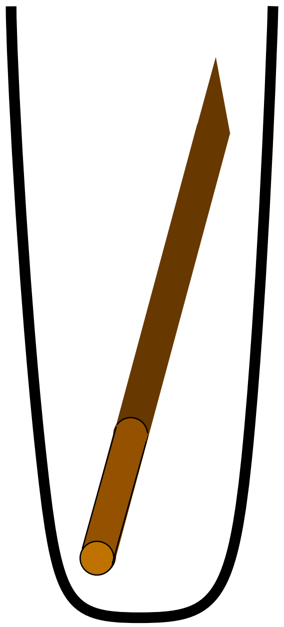
unzip, guards, deposit on mica, add streptavidin



8 µm x 8 µm

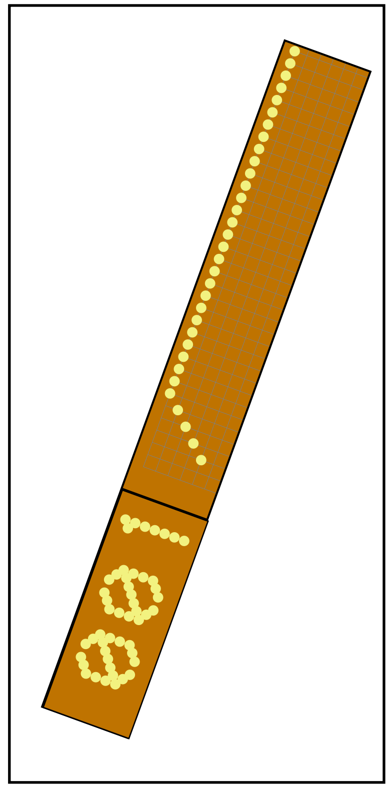
D Woods

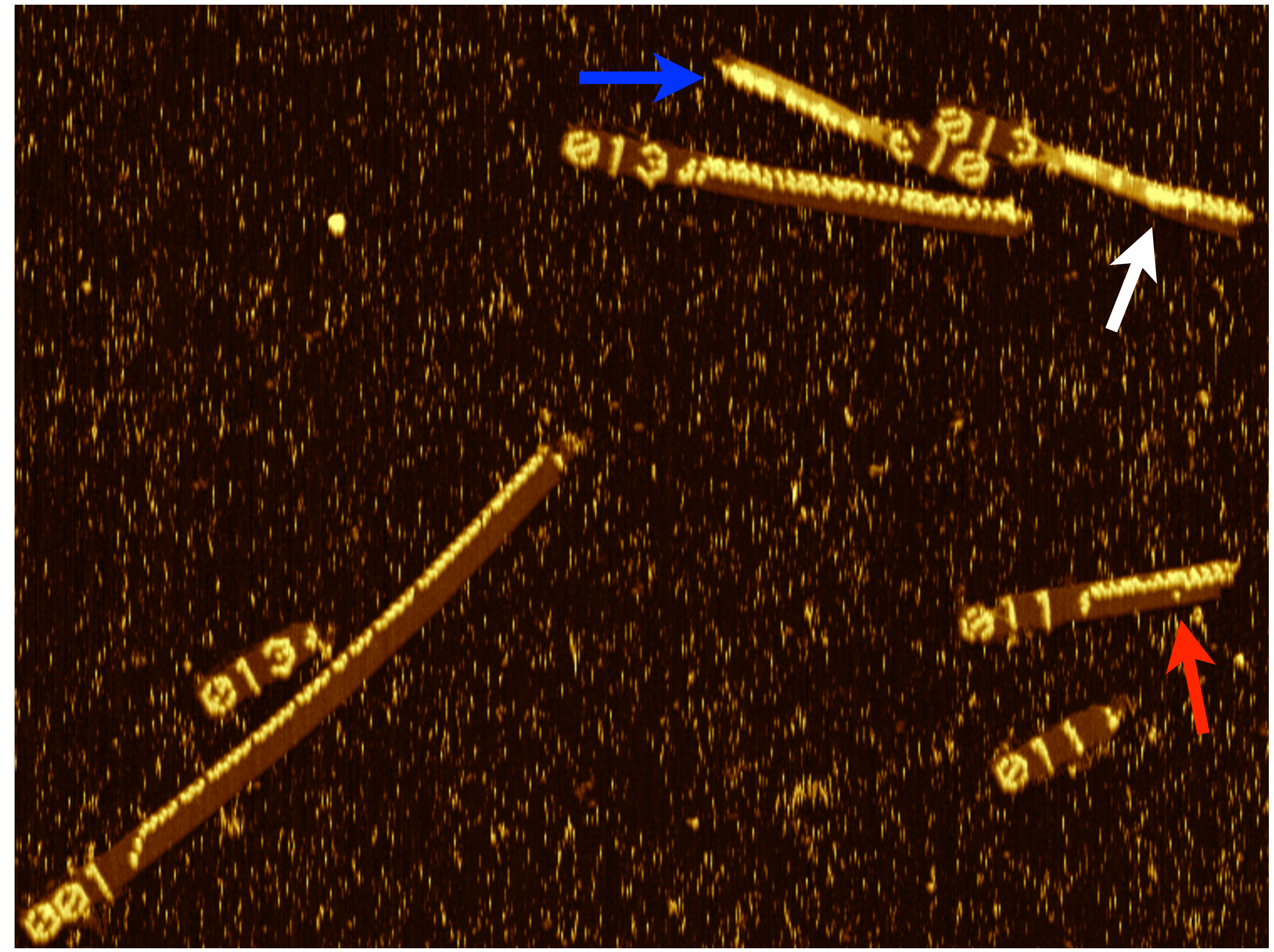
An example experiment: Sorting



1 day

unzip, guards, deposit on mica, add streptavidin

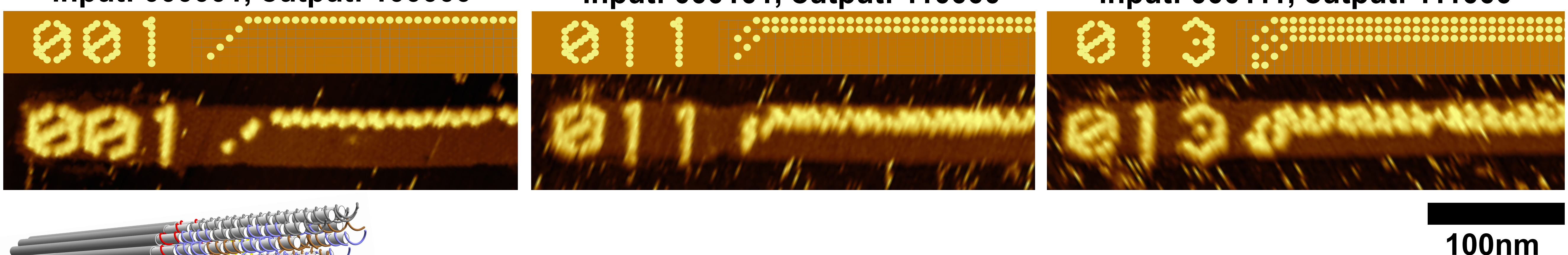


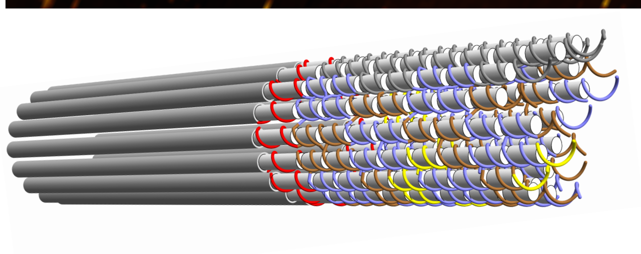


input: 000001, output: 100000

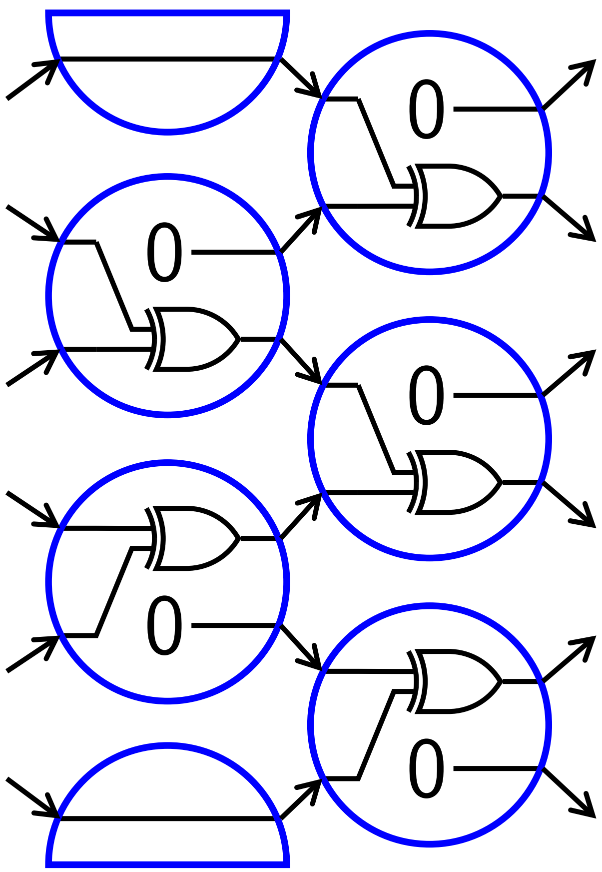
input: 000101, output: 110000

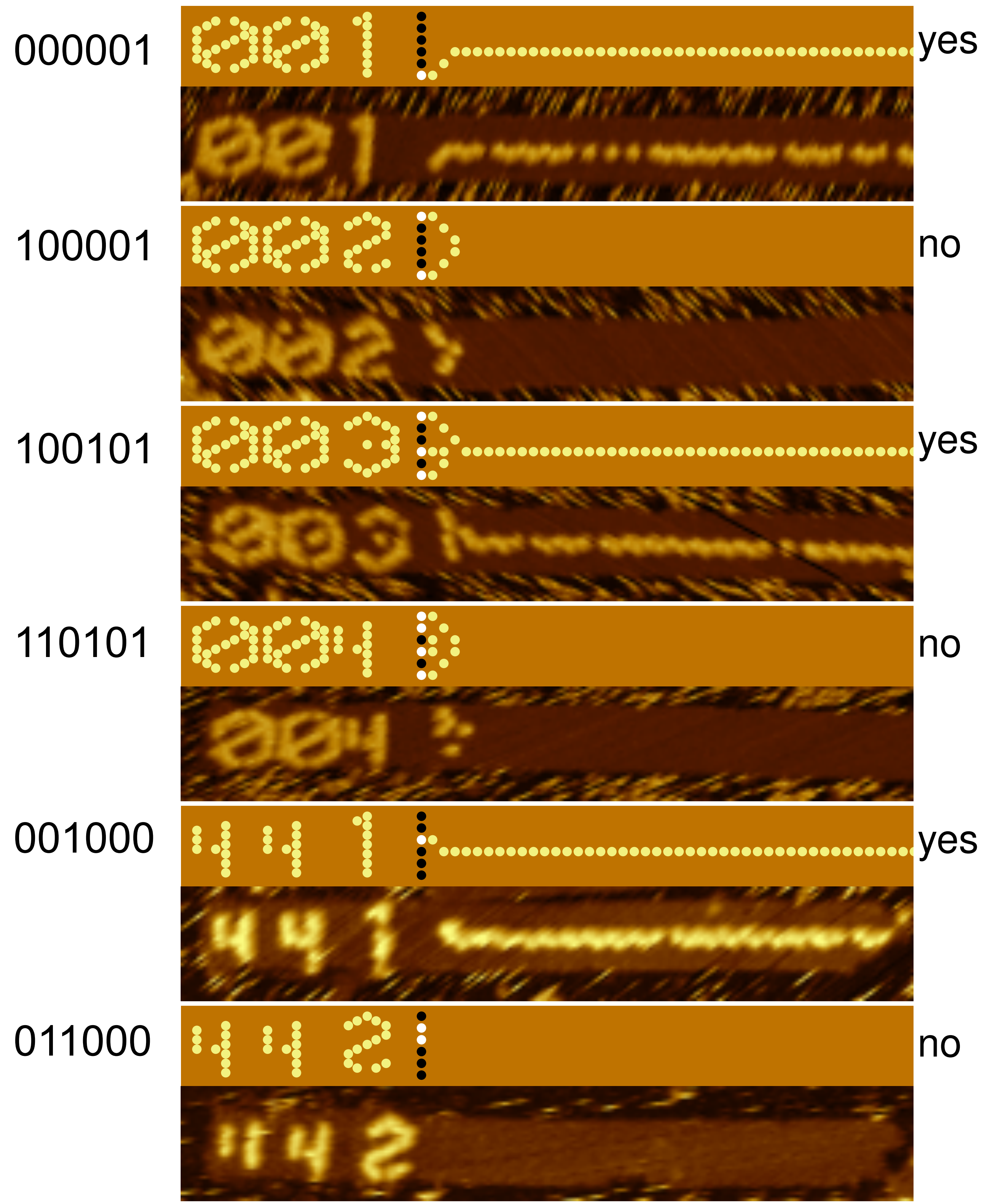
input: 000111, output: 111000



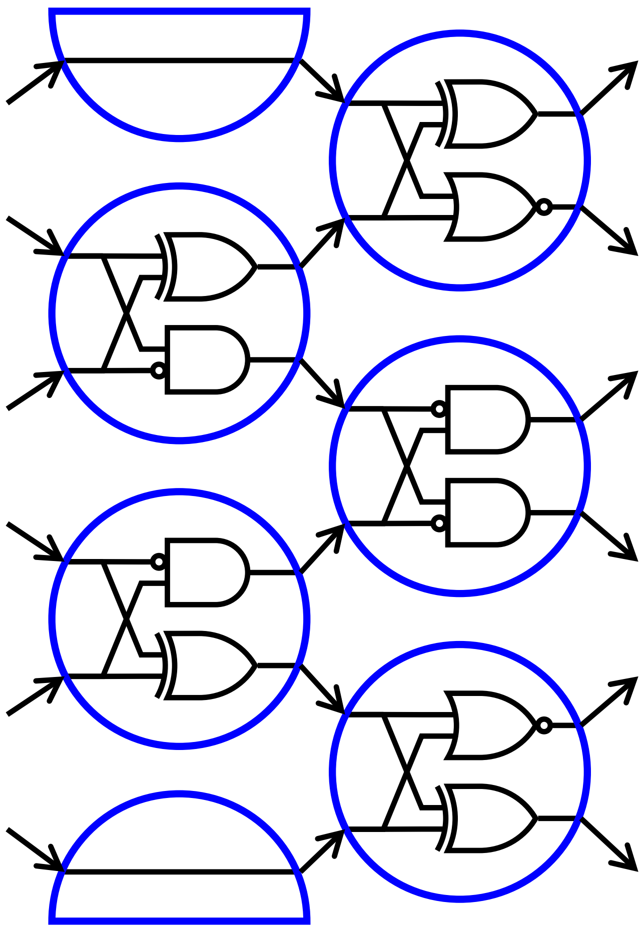


Parity: is the number of 1s odd?



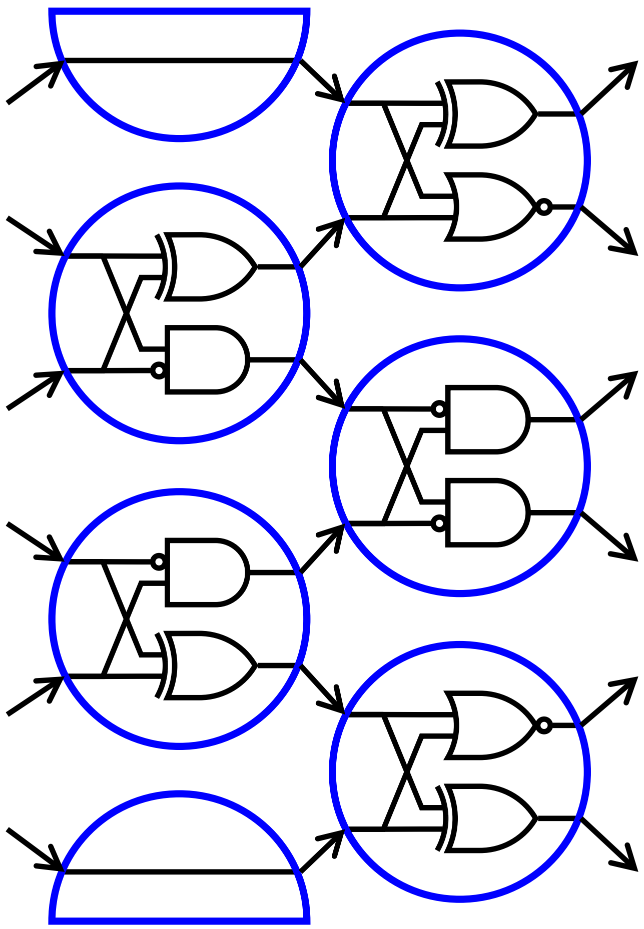


Is the input a multiple of 3?

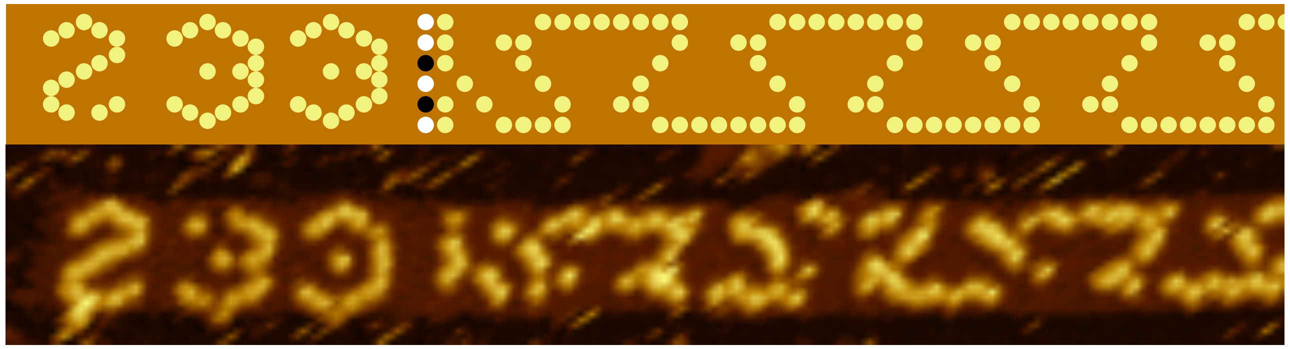


110101

Is the input a multiple of 3?

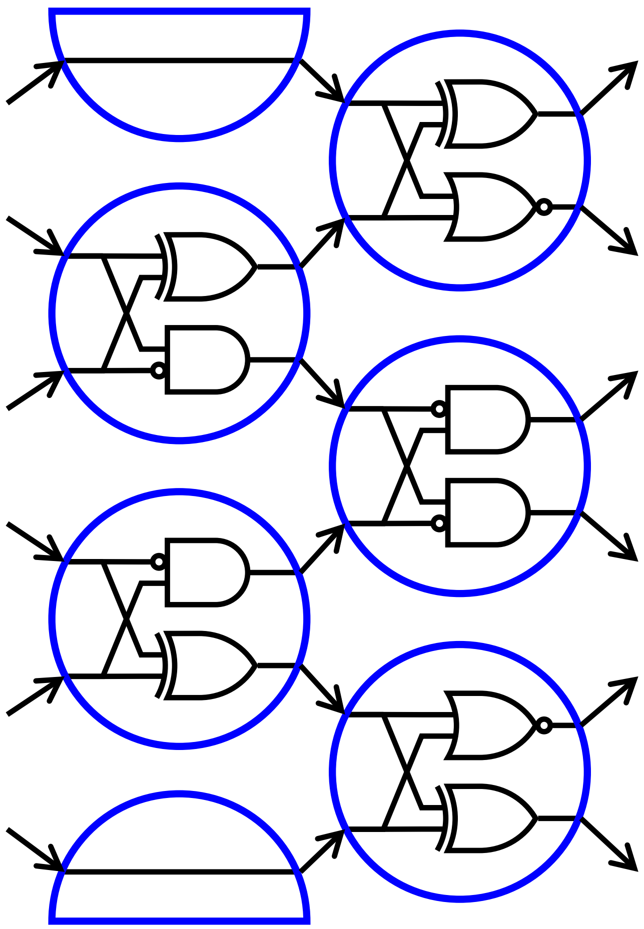


110101
=53



no

Is the input a multiple of 3?



000011
=3

yes

010101
=21

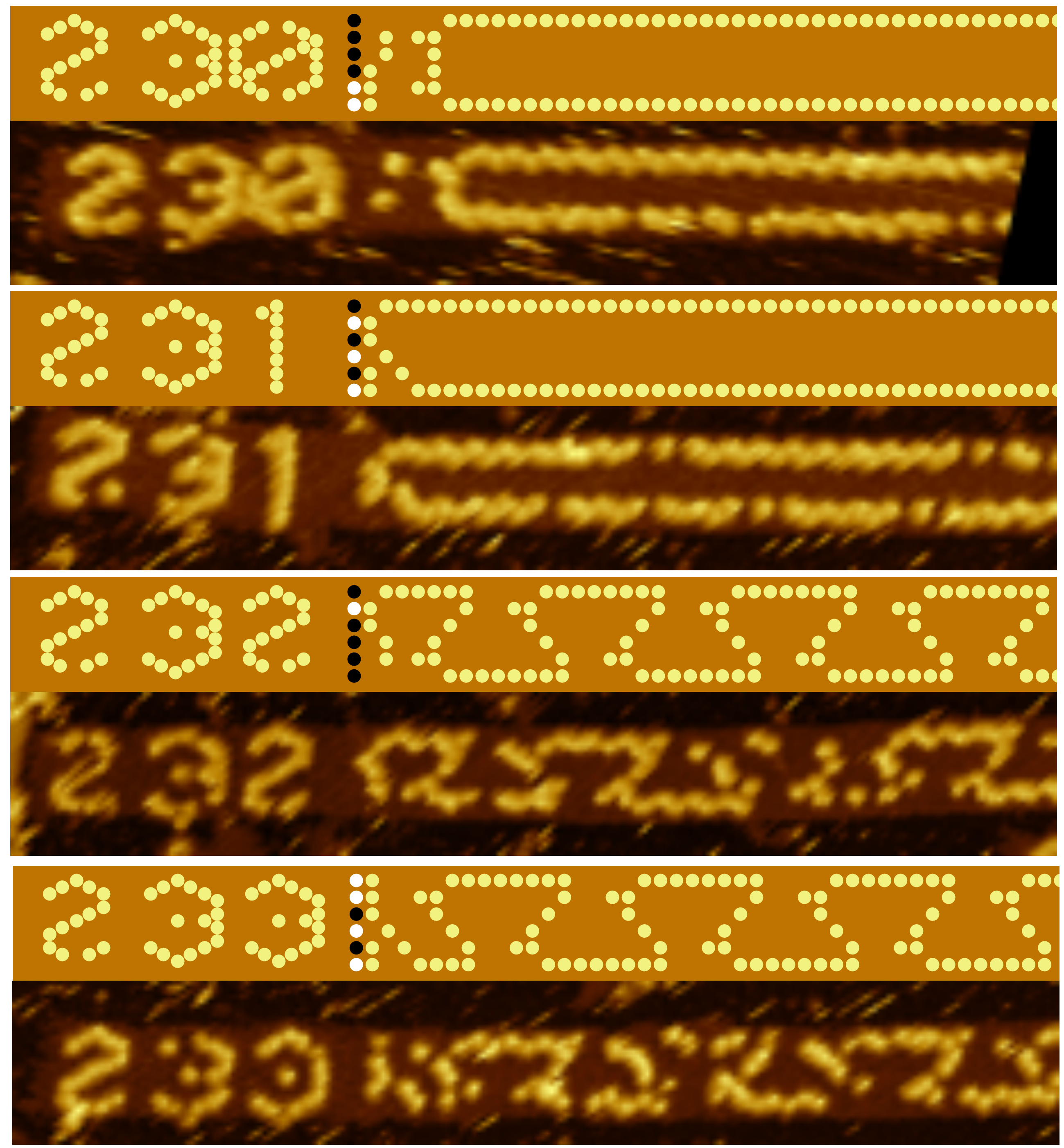
yes

010000
=16

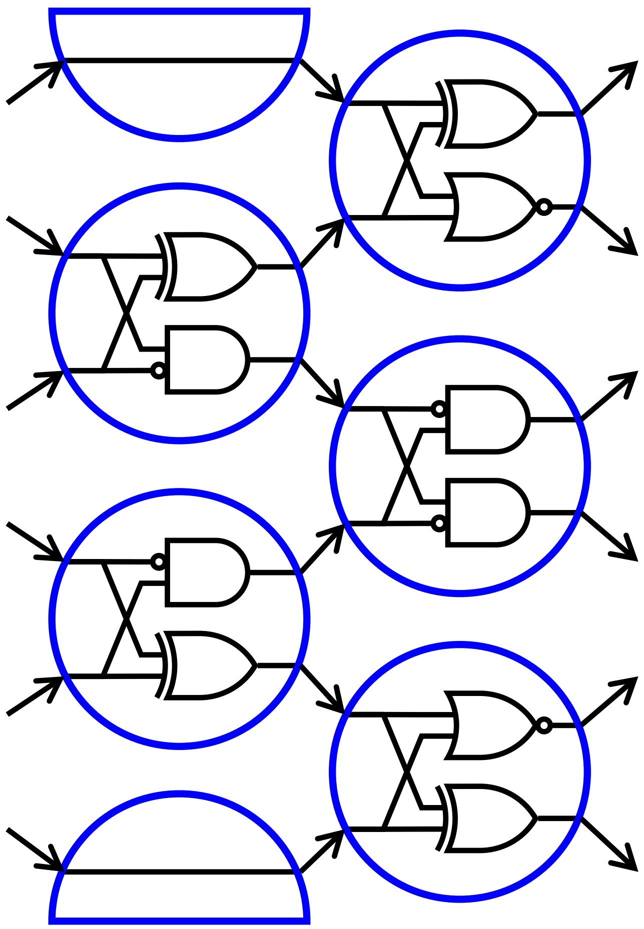
no

110101
=53

no



Is the input a multiple of 3?



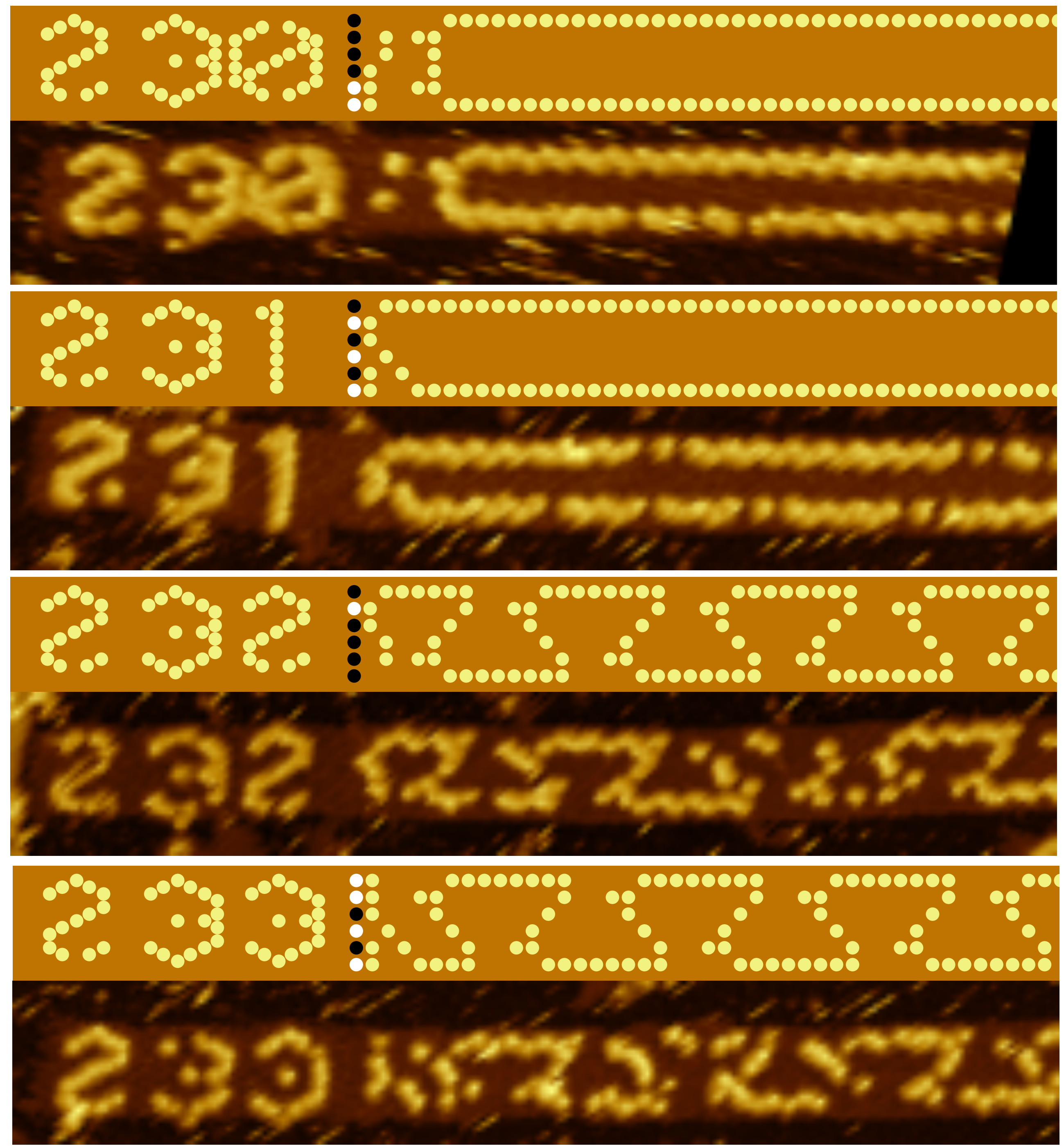
Erik Winfree

000011
=3

010101
=21

010000
=16

110101
=53



yes

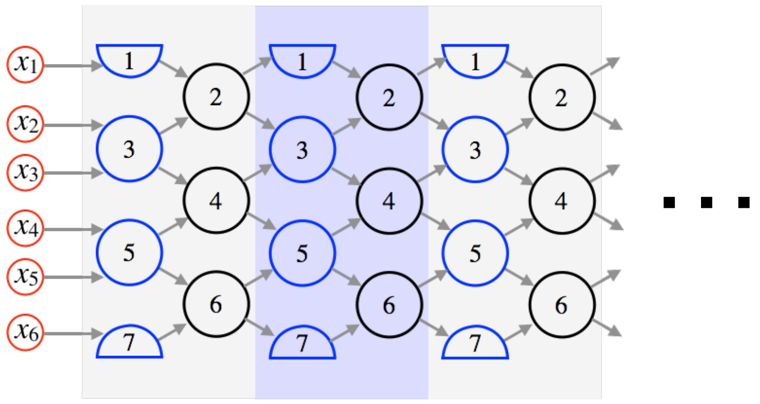
yes

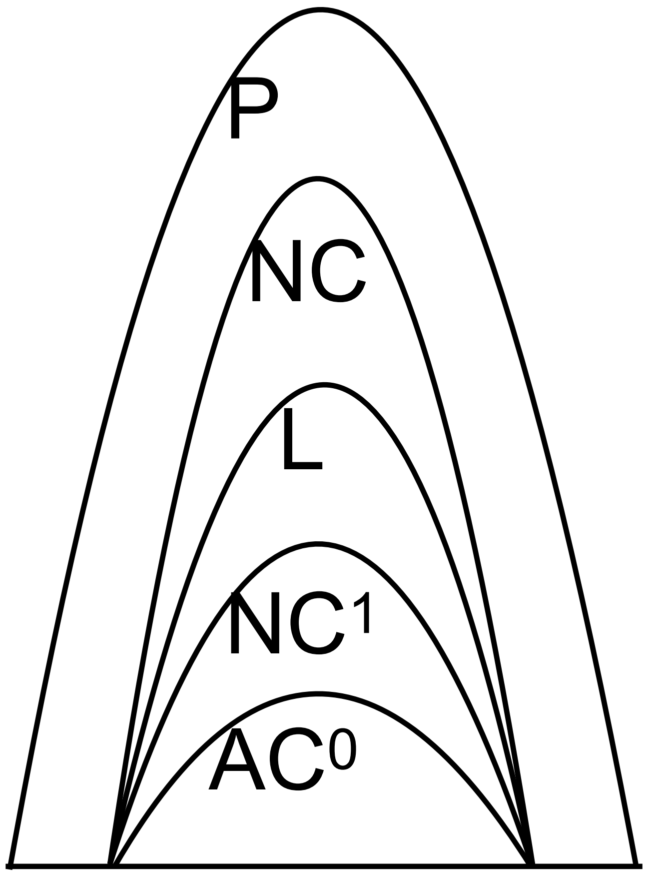
no

no

Computational power of this model?

The model is a rather restricted circuit model: “depth 2 layer”, restricted wiring within layer, repeated-layer, 0/1 signals on the wires. What can it compute?





Classes of problems:

AC^0: constant depth, poly size, Boolean circuits with arbitrary fanin gates

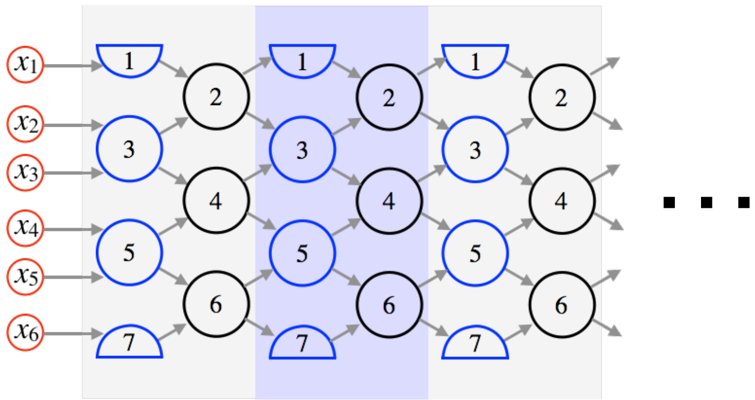
NC^1: log depth, poly size, Boolean circuits with fanin ≤ 2 gates

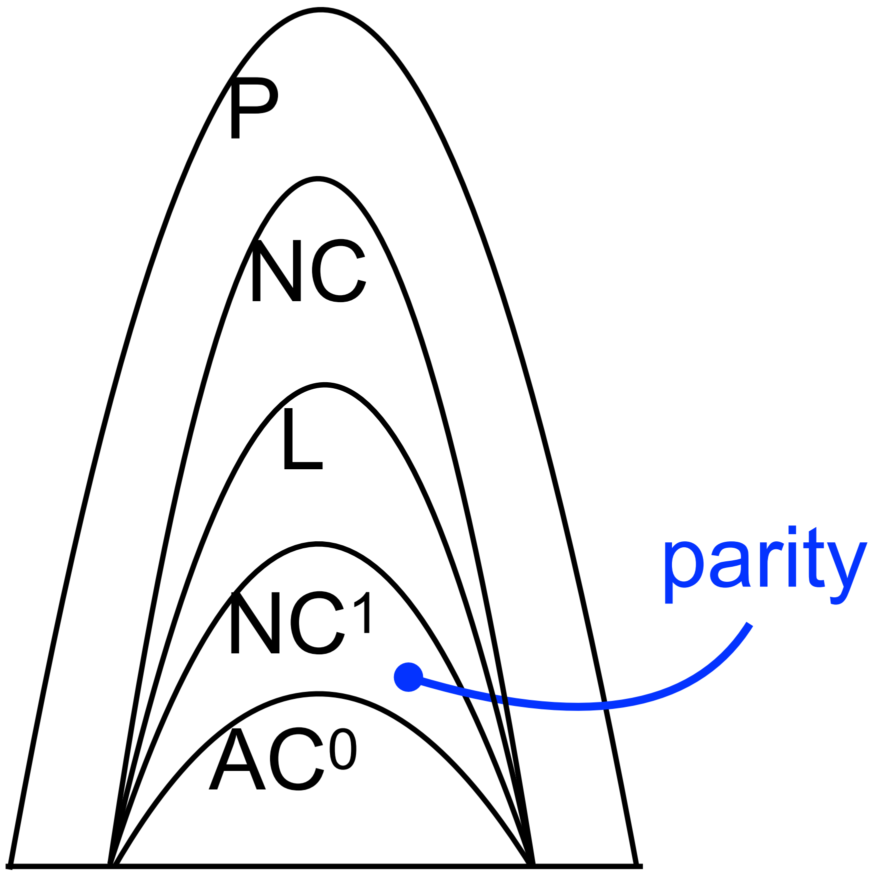
L: deterministic log space Turing machines

P: deterministic polynomial time Turing machines

Computational power of this model?

The model is a rather restricted circuit model: "depth 2 layer", restricted wiring within layer, repeated-layer, 0/1 signals on the wires. What can it compute?





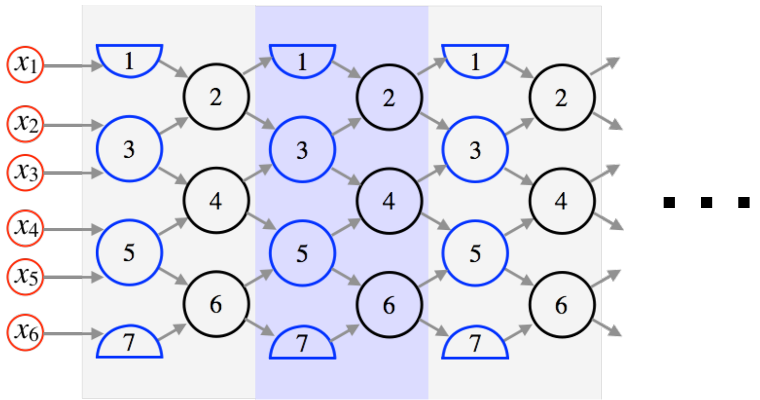
Something outside AC^0 (parity), no more than P (via explicit simulation for *t* layers)

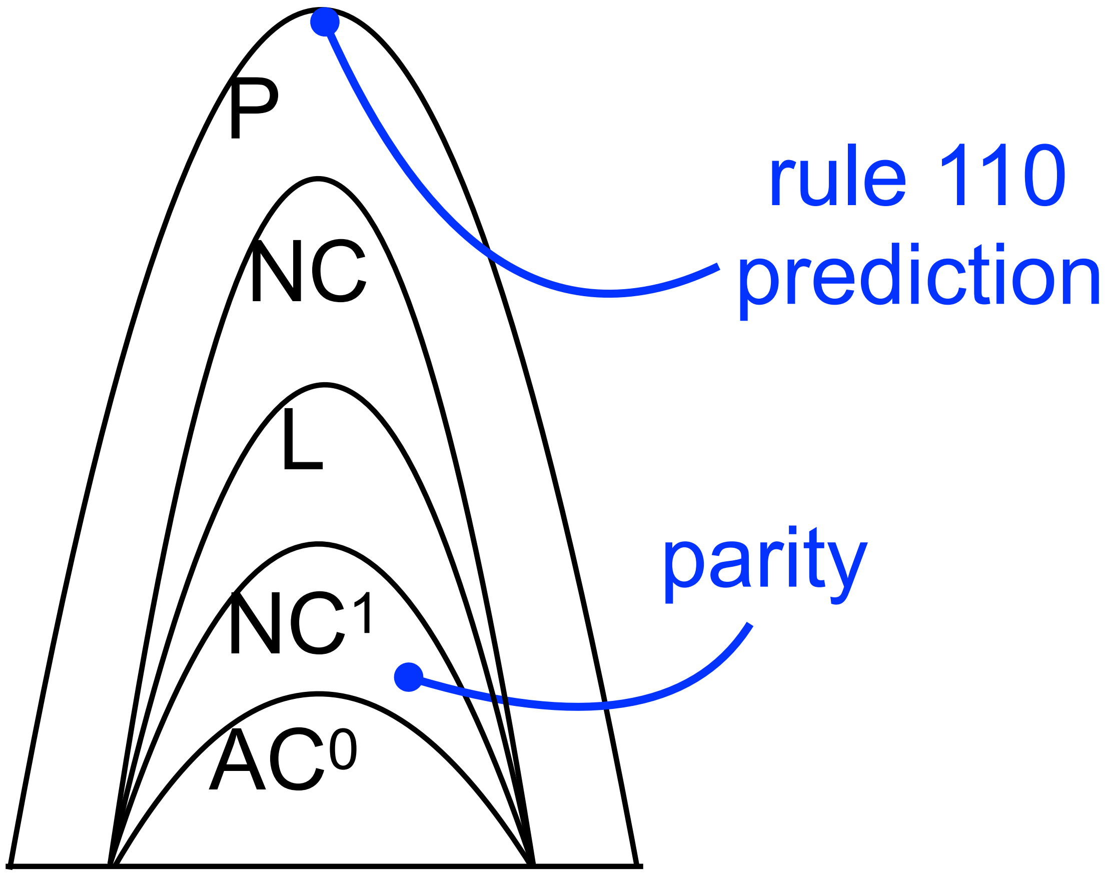
Classes of problems:

- AC^0: constant depth, poly size, Boolean circuits with arbitrary fanin gates
- NC^1: log depth, poly size, Boolean circuits with fanin ≤ 2 gates
- L: deterministic log space Turing machines
- P: deterministic polynomial time Turing machines

Computational power of this model?

The model is a rather restricted circuit model: "depth 2 layer", restricted wiring within layer, repeated-layer, 0/1 signals on the wires. What can it compute?



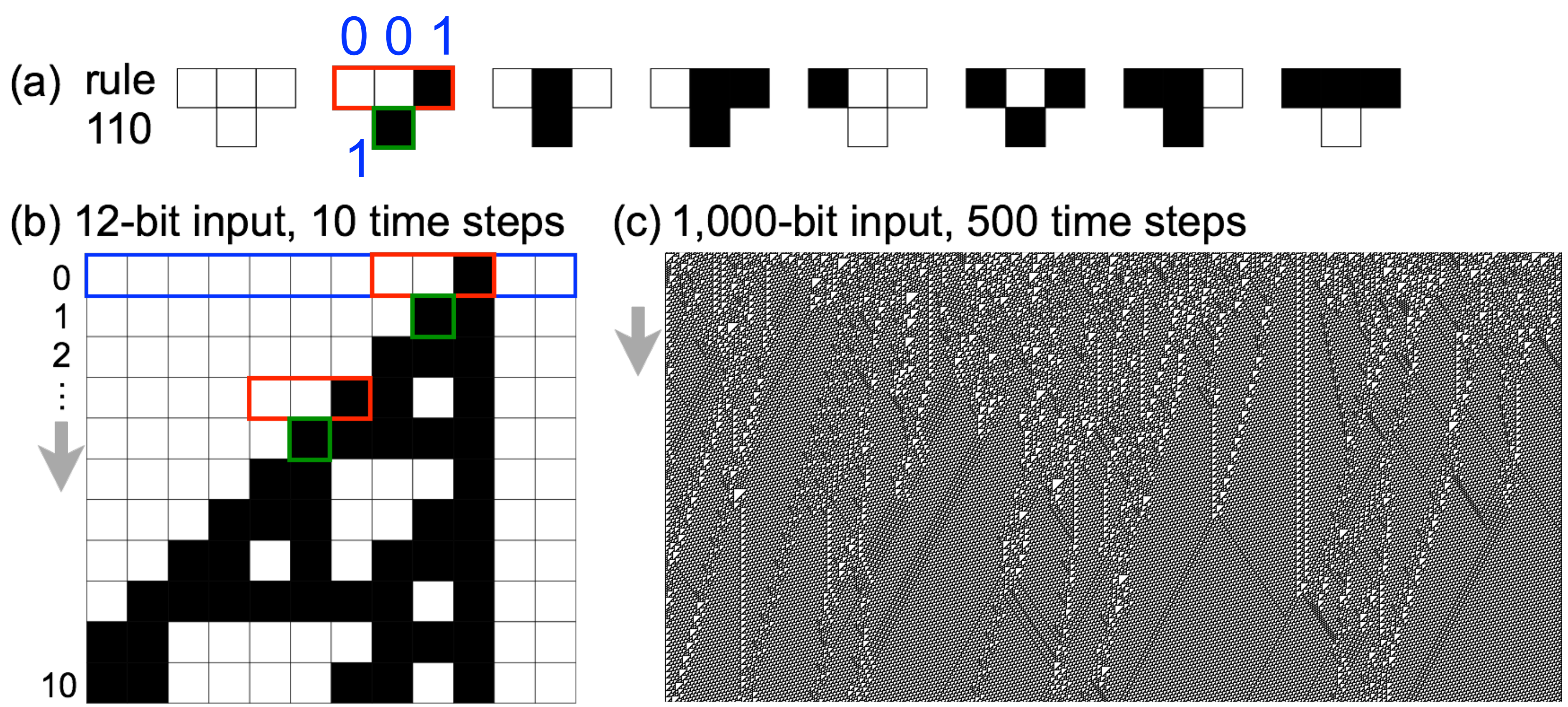


Something outside AC^0 (parity), no more than P (via explicit simulation for *t* layers)

Classes of problems:

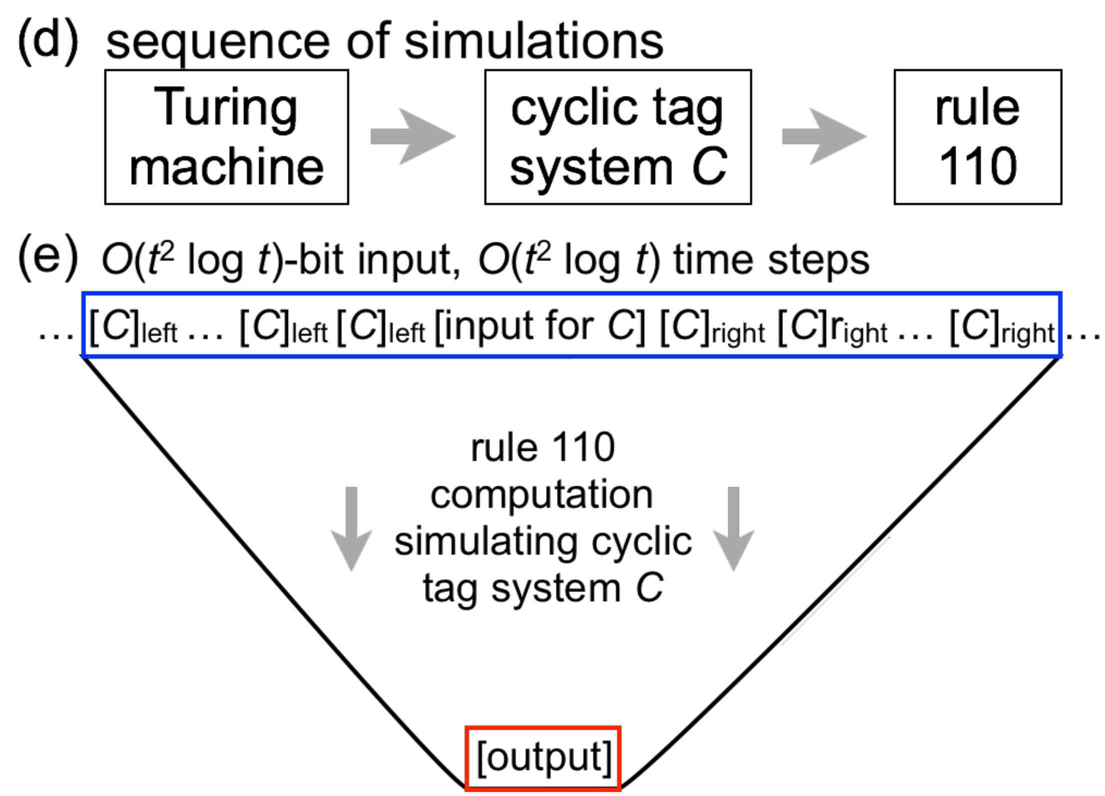
- AC^0: constant depth, poly size, Boolean circuits with arbitrary fanin gates
- NC^1: log depth, poly size, Boolean circuits with fanin ≤ 2 gates
- L: deterministic log space Turing machines
- P: deterministic polynomial time Turing machines

Rule 110



Theorem: Let M be a Turing machine that runs in time t, rule 110 simulates M in $O(t^2 \log t)$ steps

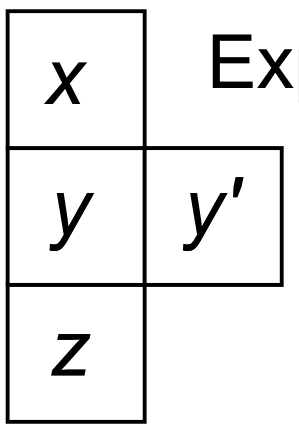
[Cook 2004]
[Neary, Woods, 2006]
[Neary, PhD thesis]



Simulation of rule 110

$x\,y\,z$	$x\,y\,z$
$F(0,0,0)=0$	$F(1,0,0)=0$
$F(0,0,1)=1$	$F(1,0,1)=1$
$F(0,1,0)=1$	$F(1,1,0)=1$
$F(0,1,1)=1$	$F(1,1,1)=0$

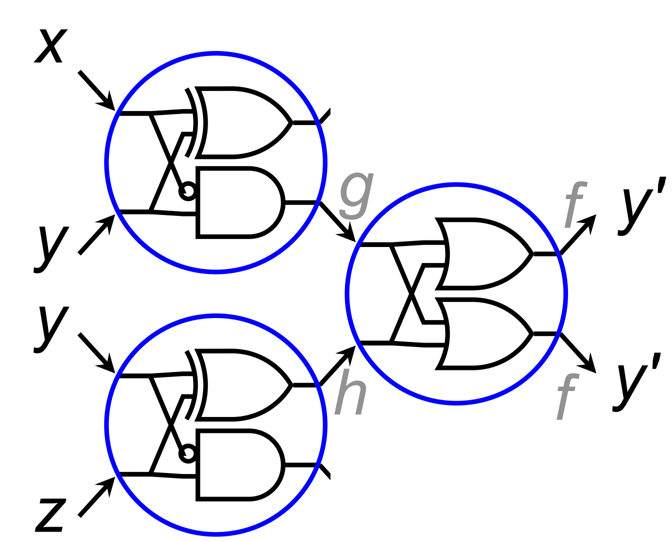
rule 110
truth table



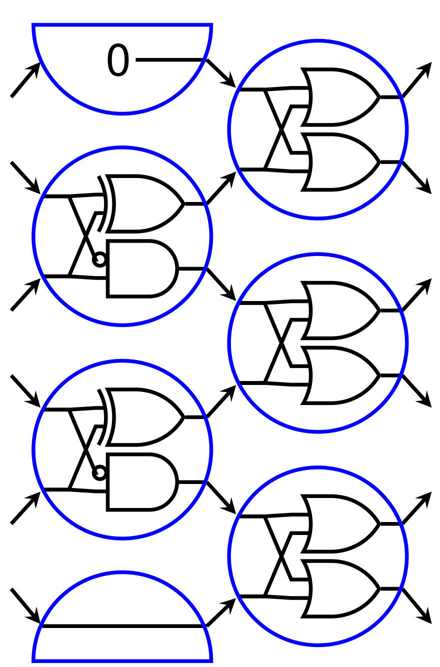
Express the rule as *f*(*g*(*x*,*y*),*h*(*y*,*z*))

$$y' = (\neg x \wedge y) \vee (y \otimes z)$$

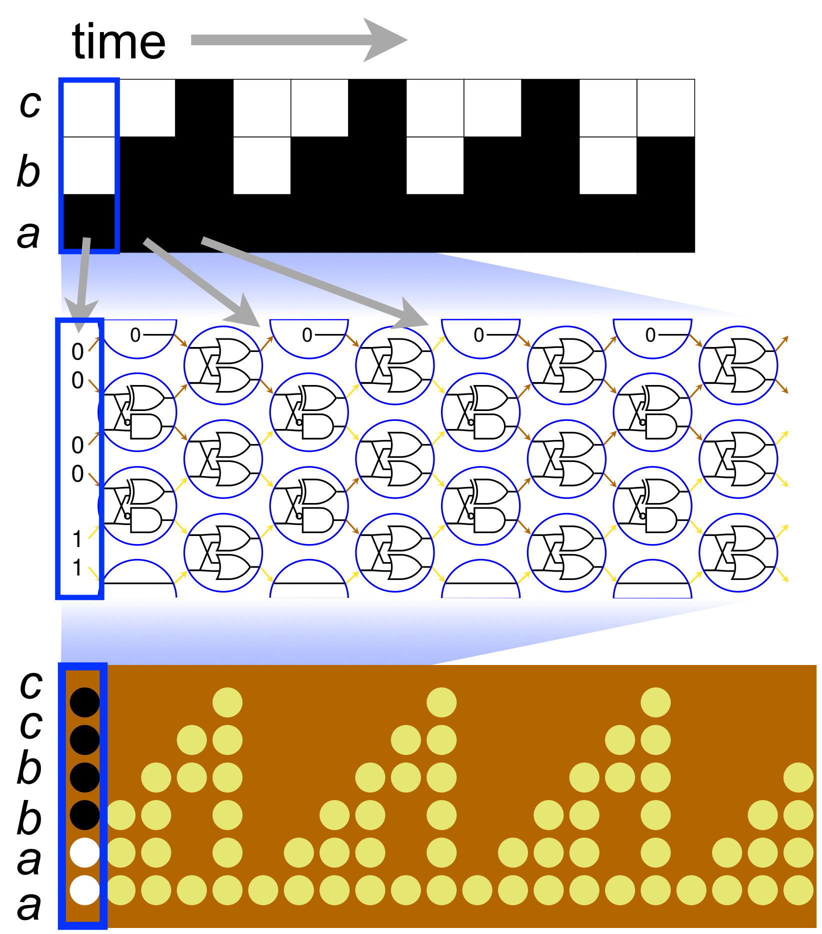
IBC sub-circuit

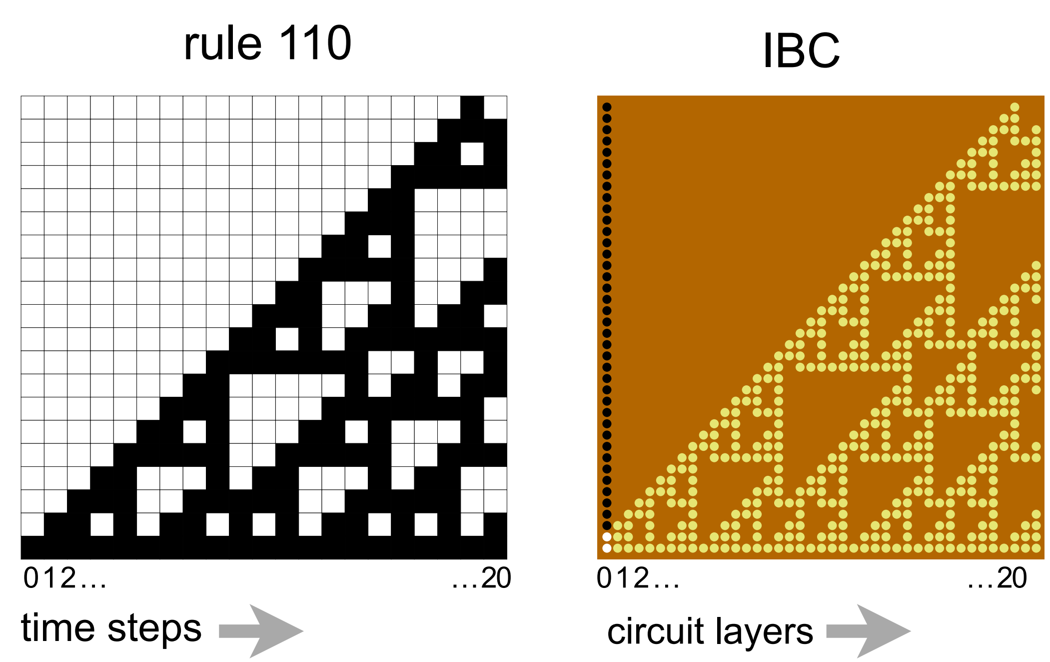


6-bit IBC to simulate
3 bits of rule 110

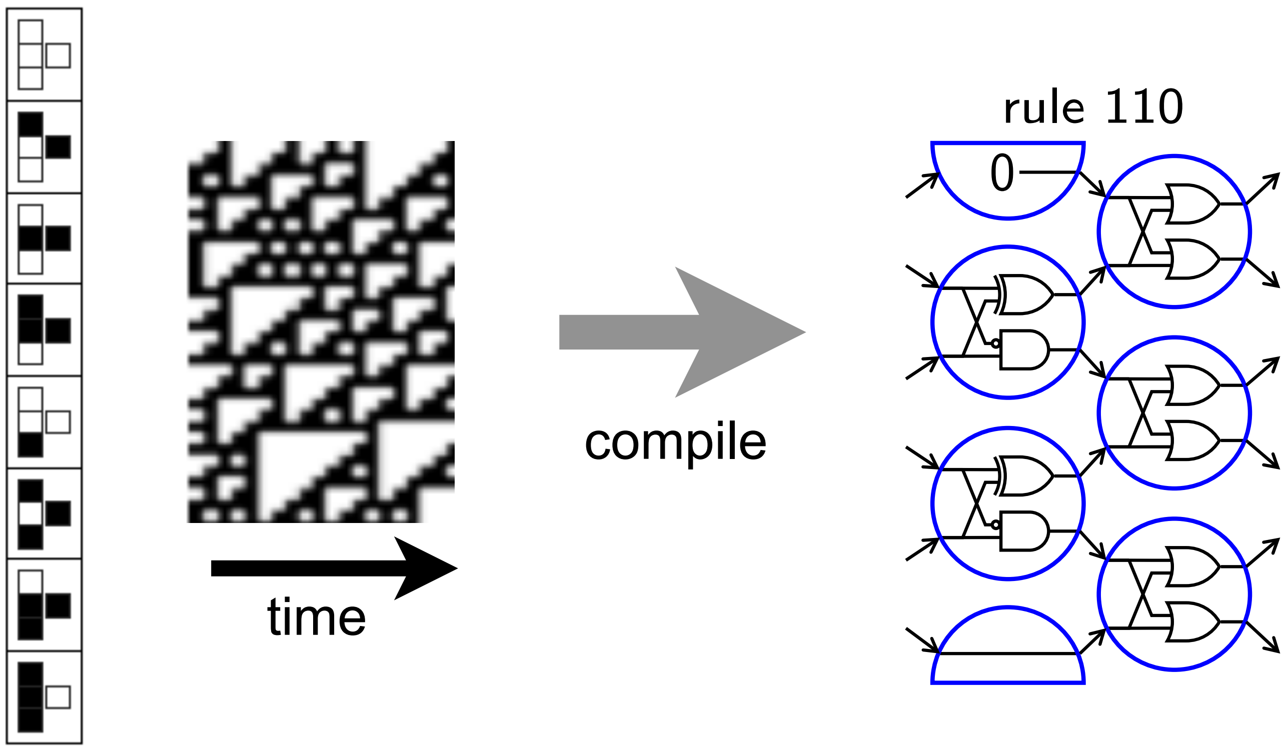


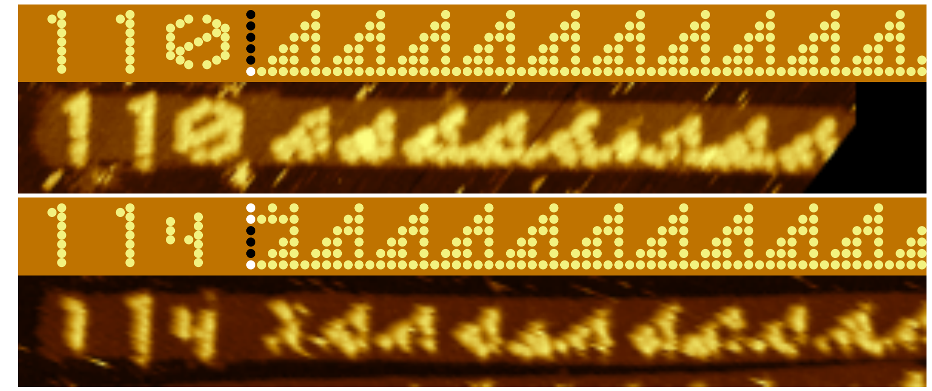
IBC simulation on 3 bits

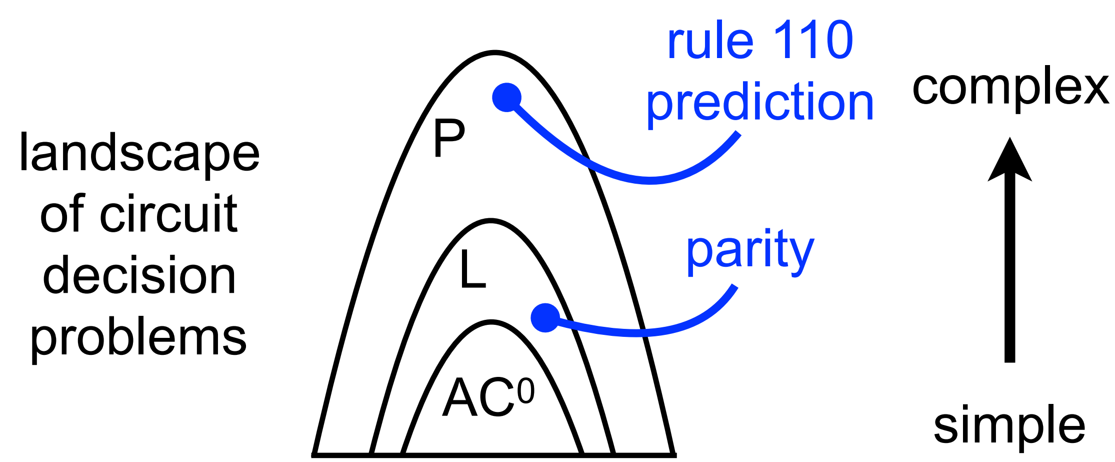




RULE110 circuit: simulation of cellular automata





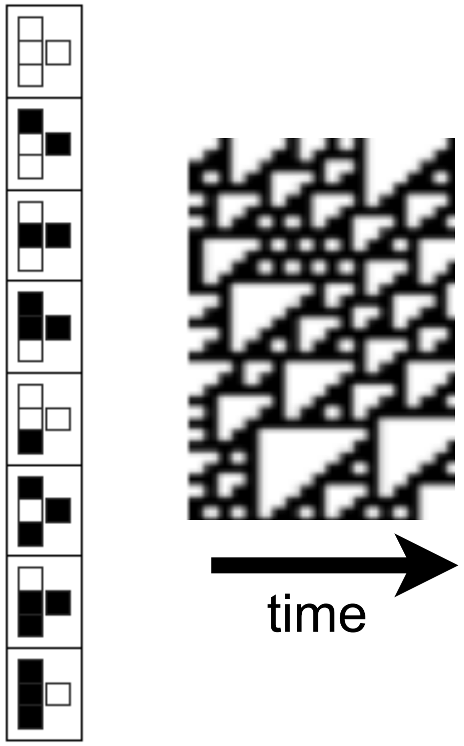


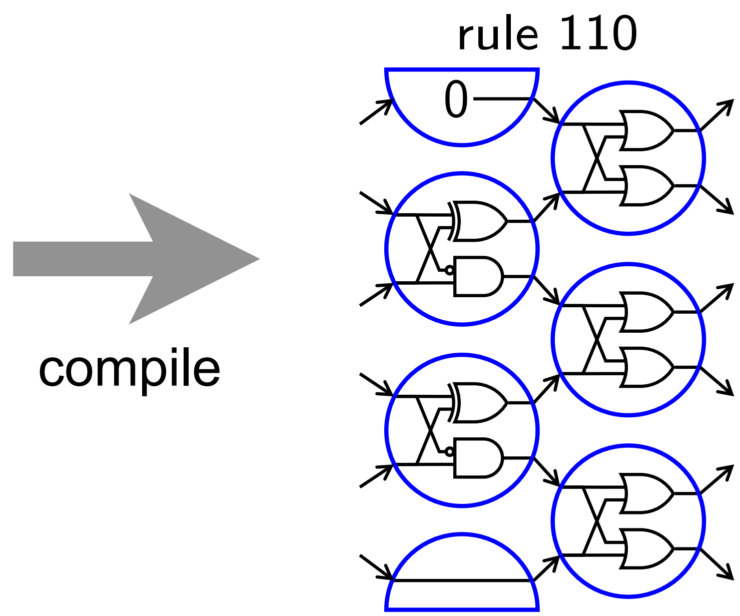
Theorem: Let *M* be a single-tape Turing machine that runs in time *t*, then $O(t^2 \log t)$-bit 1-layer circuits (IBCs) simulate *M*

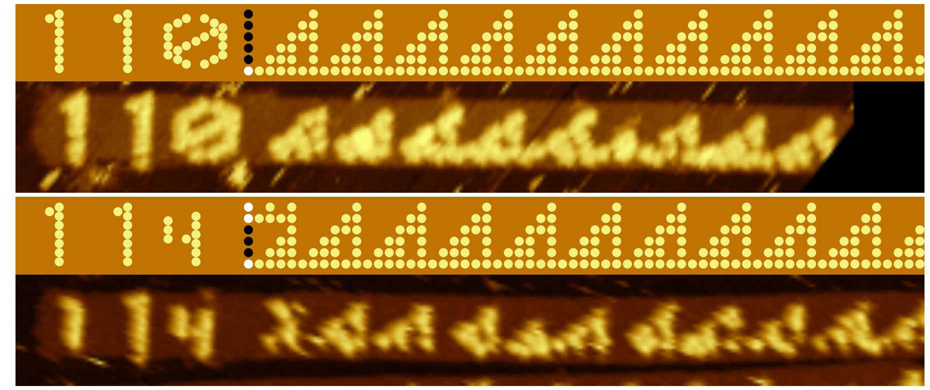
IBCs efficiently simulate any algorithm

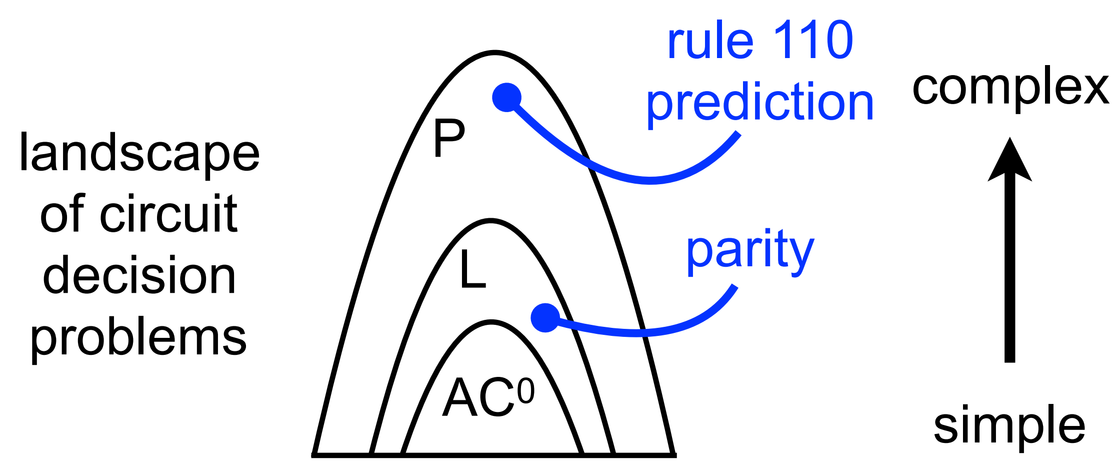
[Cook 2004]
[Neary, Woods, 2006]
[Neary, PhD thesis]

RULE110 circuit: simulation of cellular automata

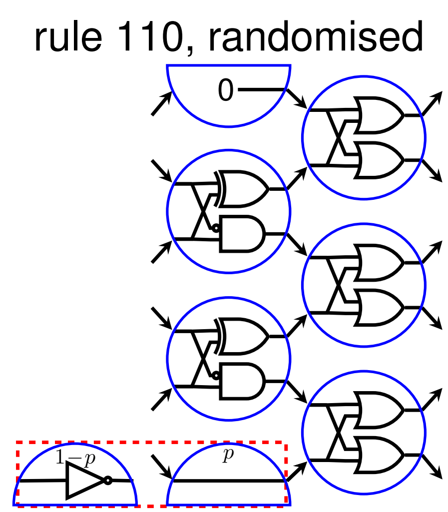


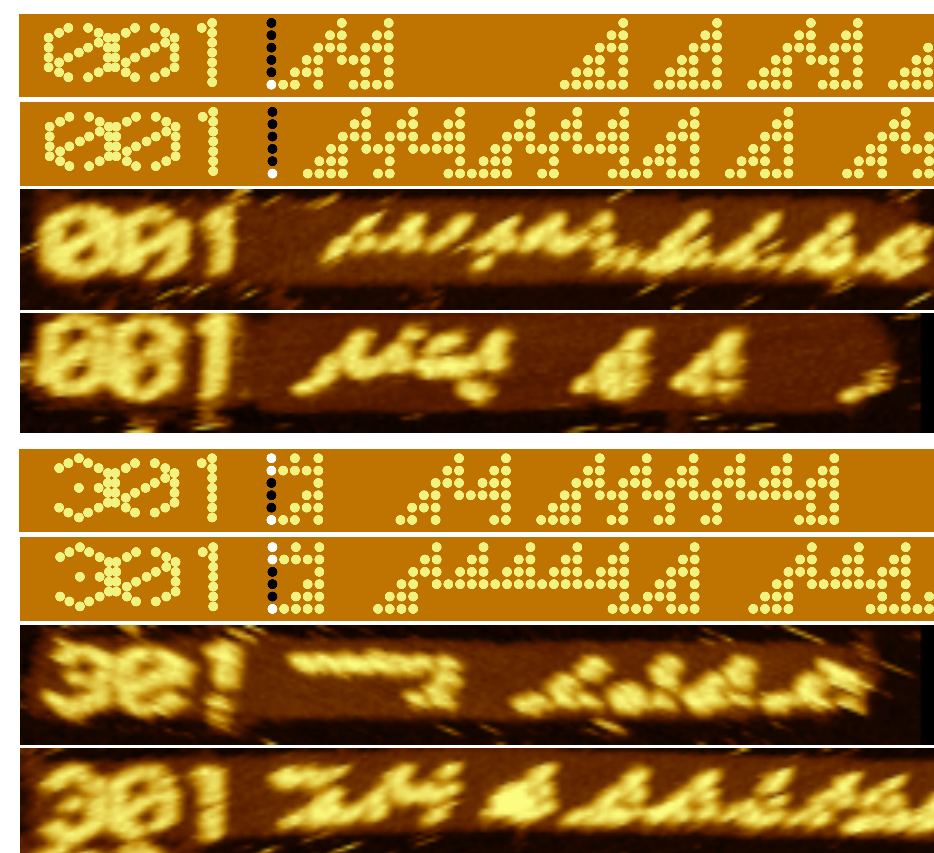






Theorem: Let *M* be a single-tape Turing machine that runs in time *t*, then $O(t^2 \log t)$-bit 1-layer circuits (IBCs) simulate *M*



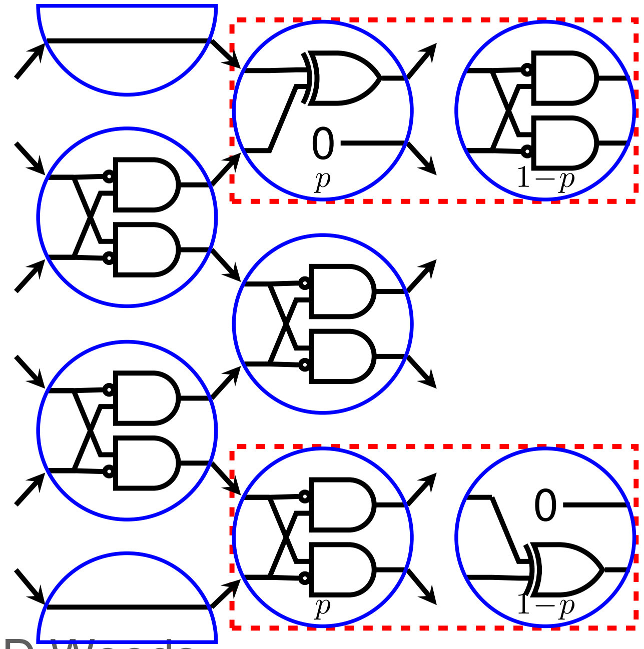


IBCs efficiently simulate any algorithm

[Cook 2004]
[Neary, Woods, 2006]
[Neary, PhD thesis]

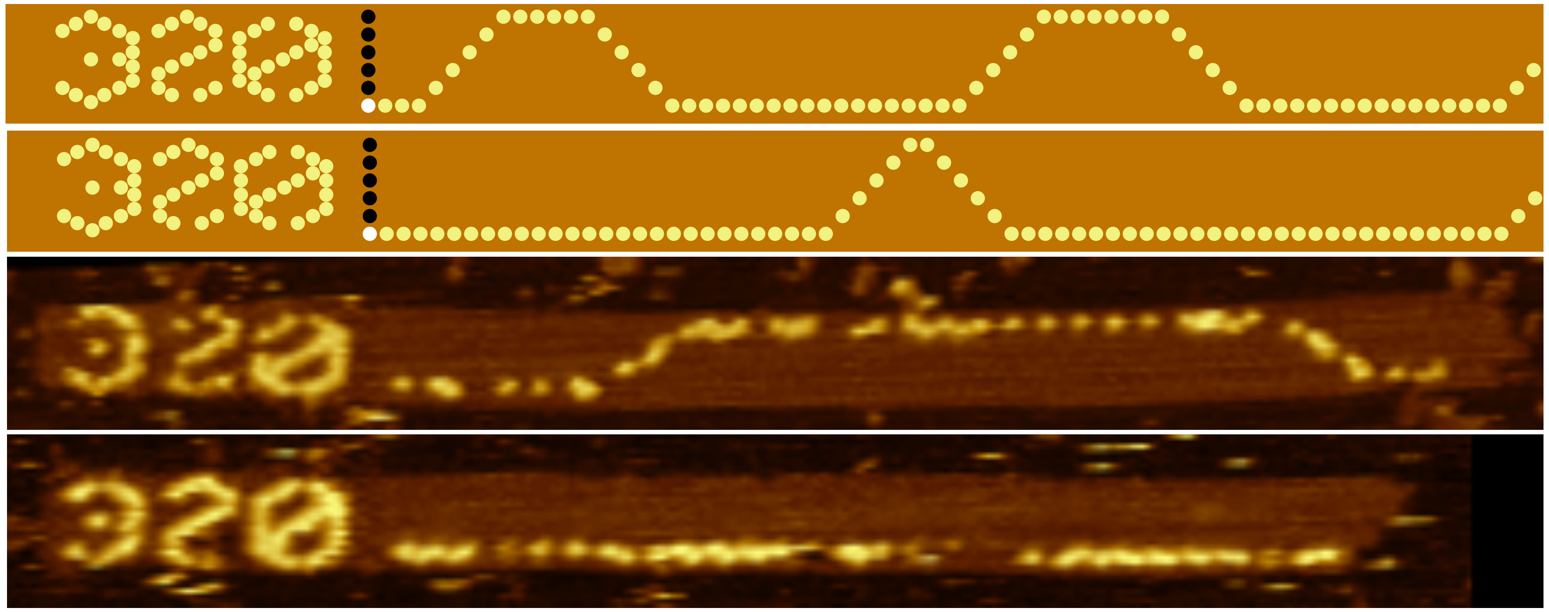
Open: characterise power of randomised model

California surf: WAVES

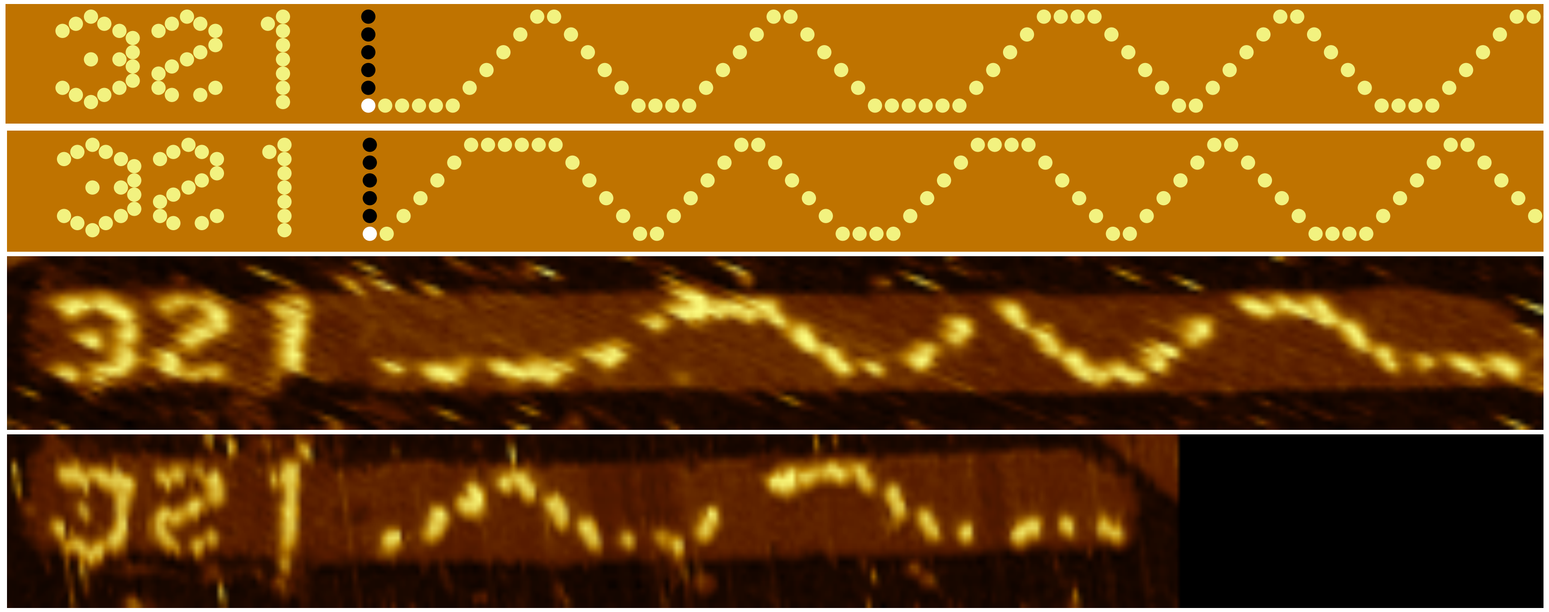


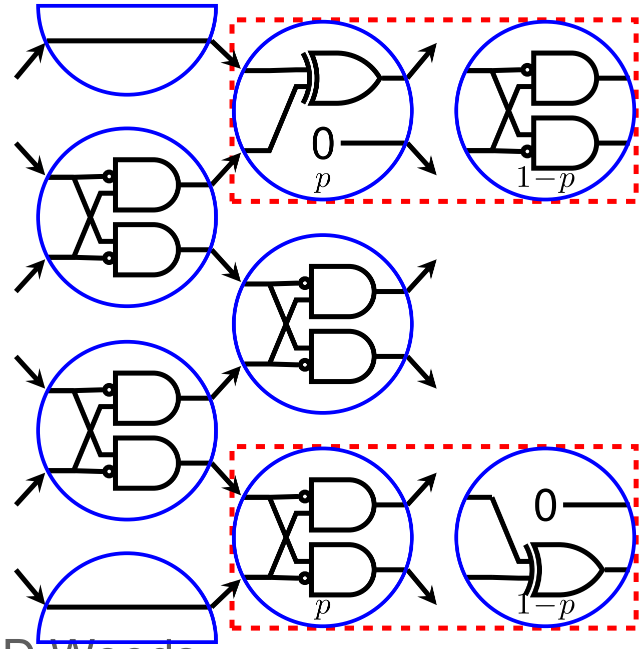
California surf: WAVES

Pr[create wave]=0.1
Pr[crash wave] =0.5

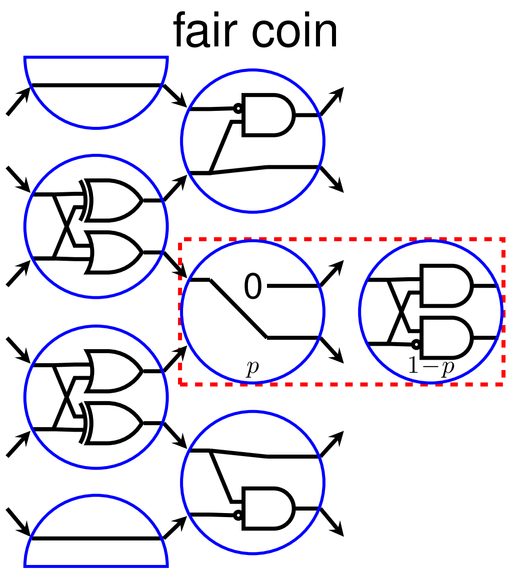


Pr[create wave]=0.5
Pr[crash wave] =0.5





FAIRCOIN: Unbiased bit from biased coin

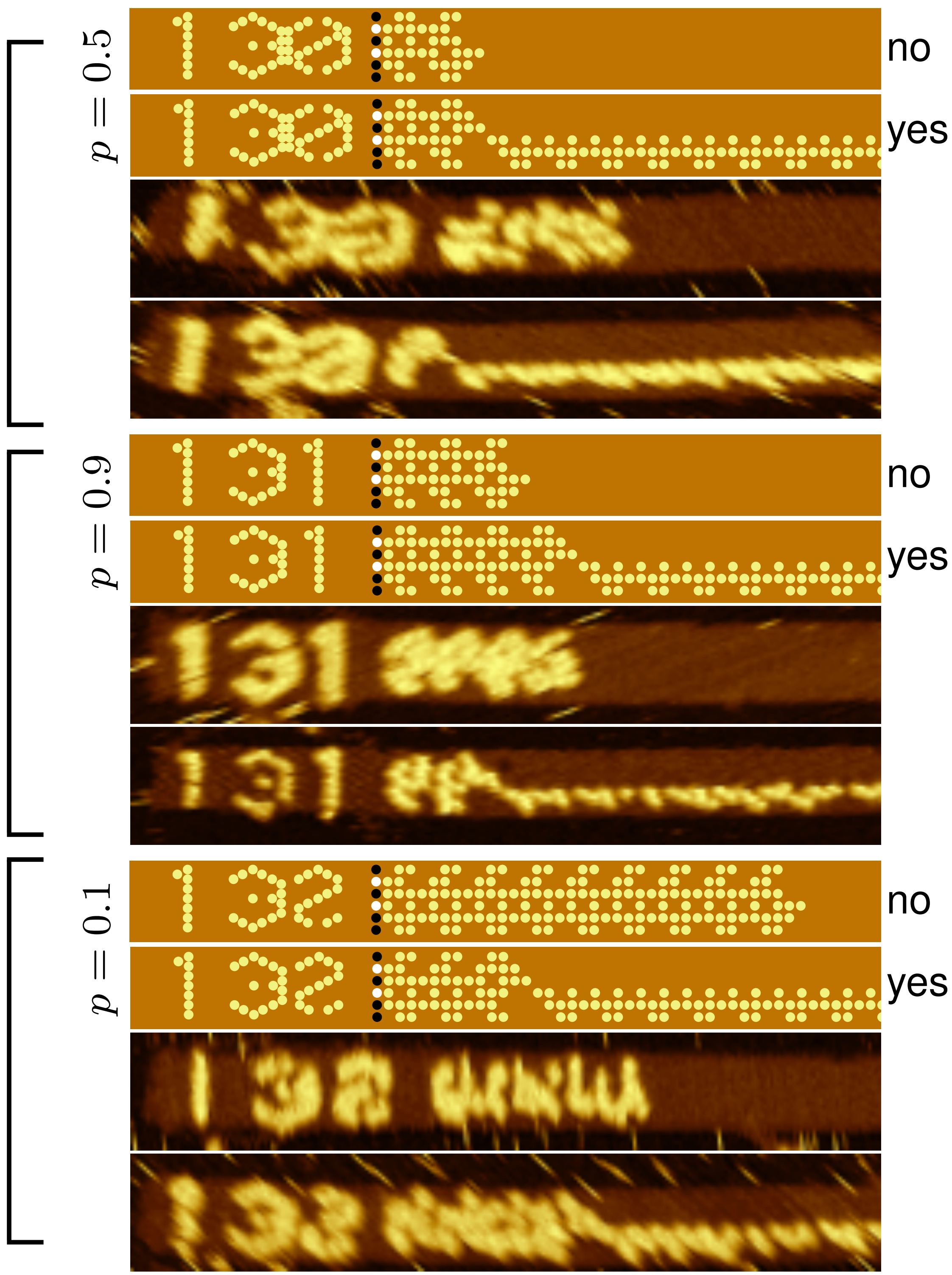


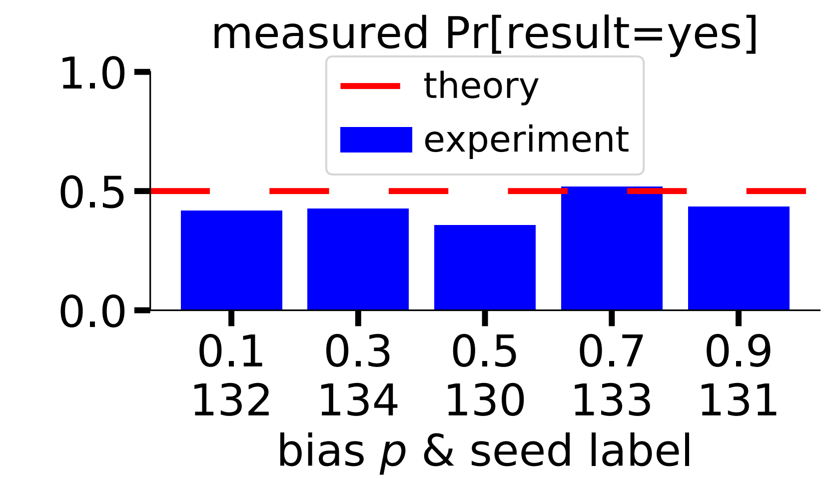
1:1

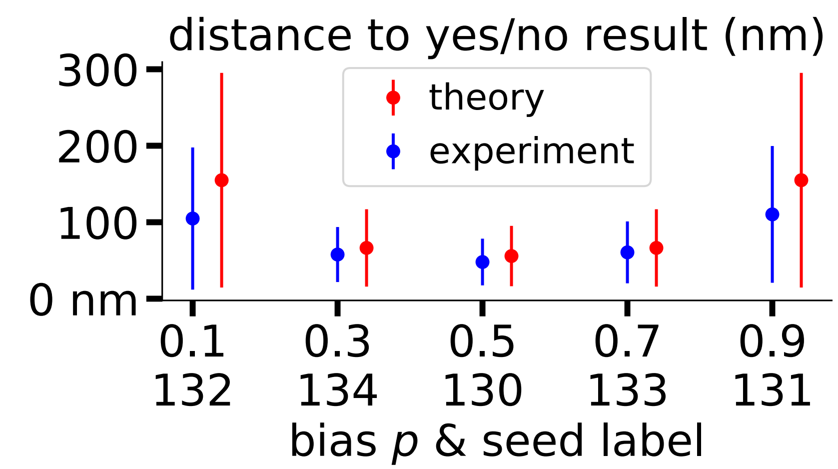
9:1

1:9

heads:tails



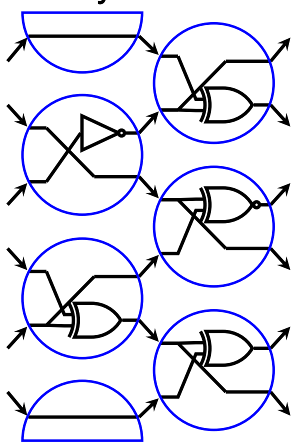




Dave Doty

Counting to 63

cycle 63



Circuit with 63 distinct strings

1 2 3 … …62 63 1 2 …

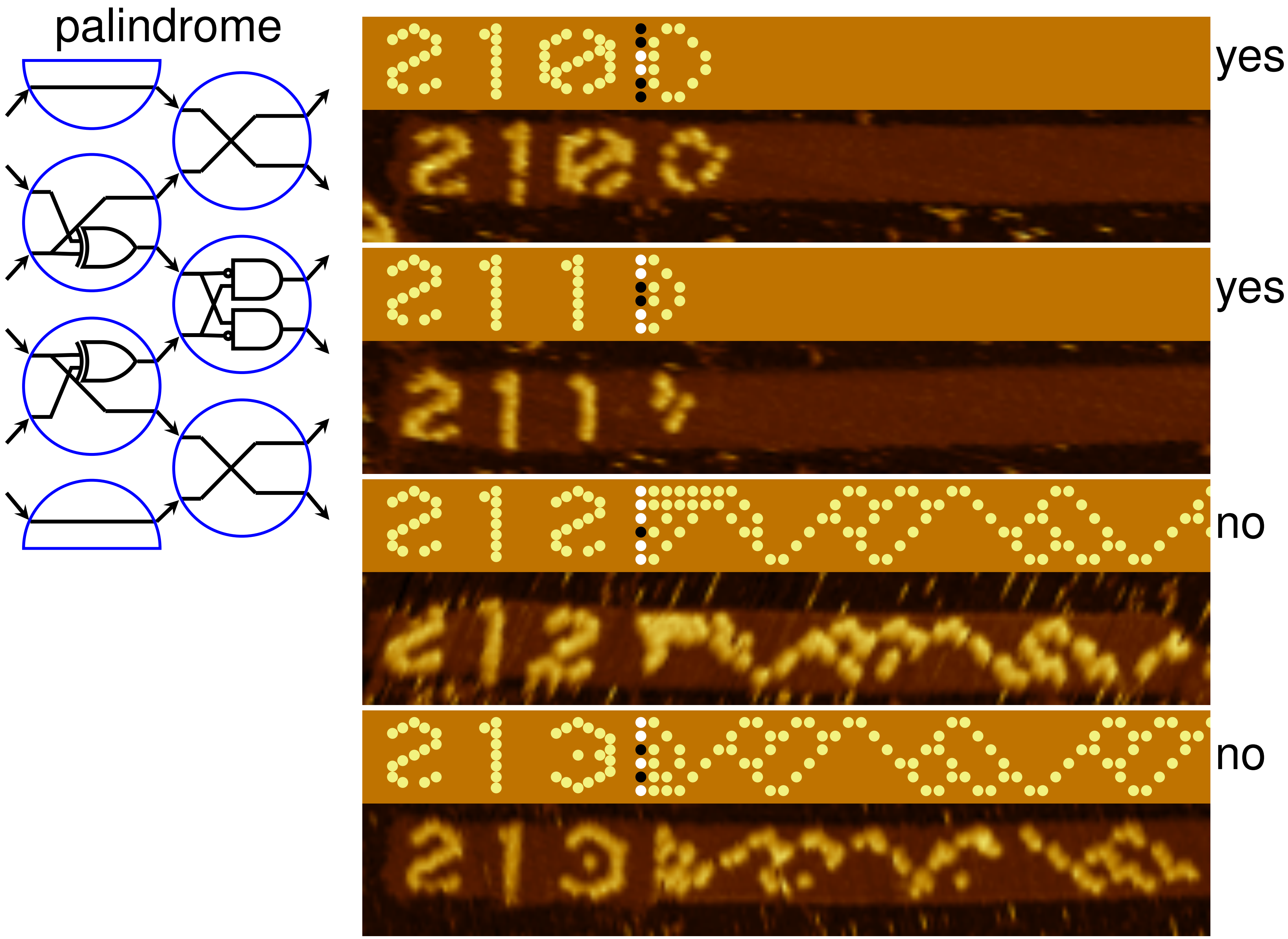
42

Is there a 64-counter?

No!
Proof by Tristan Stérin
Maynooth University

Palindromes: high communication complexity

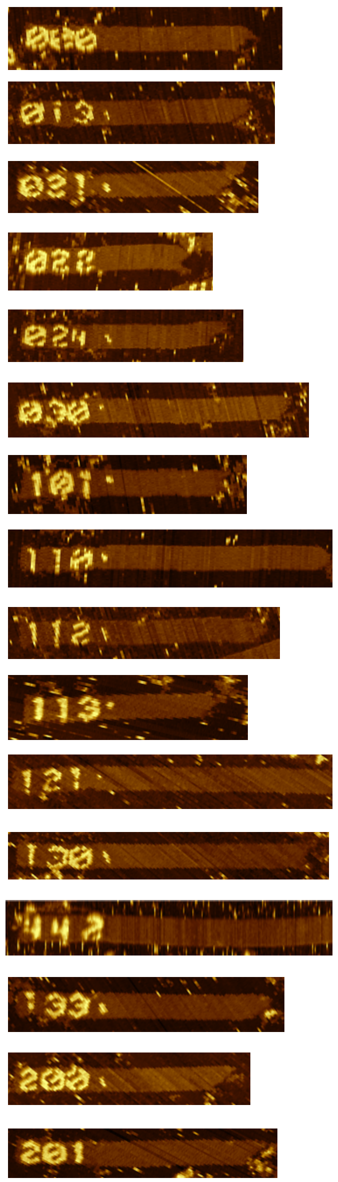
Palindromes: high communication complexity



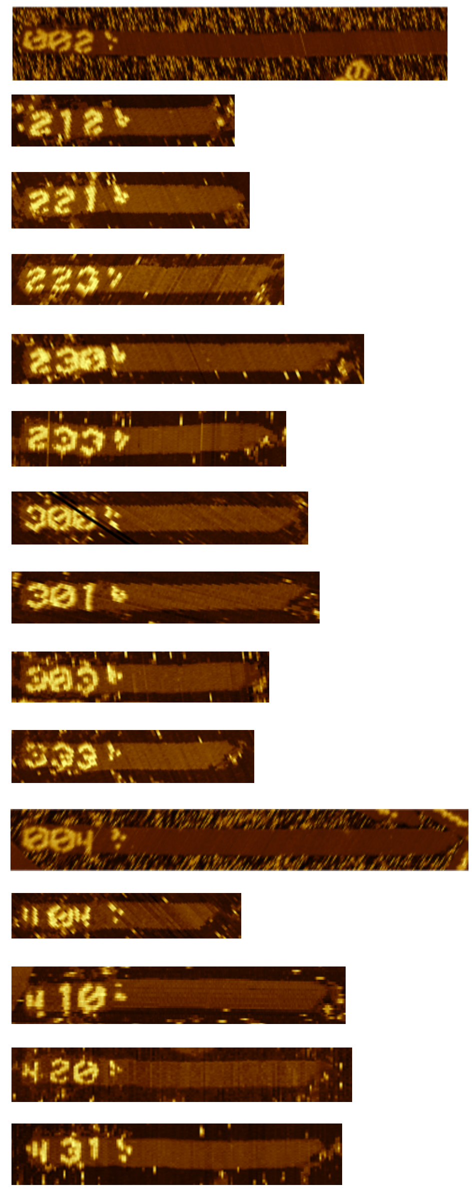
Testing of tile set: PARITY on all 64 inputs

32 x No

σ(000000) = 000
σ(000011) = 013
σ(000101) = 021
σ(000110) = 022
σ(001001) = 024
σ(001010) = 030
σ(001100) = 101
σ(001111) = 110
σ(010001) = 112
σ(010010) = 113
σ(010100) = 121
σ(010111) = 130
σ(011000) = 442
σ(011011) = 133
σ(011101) = 200
σ(011110) = 201

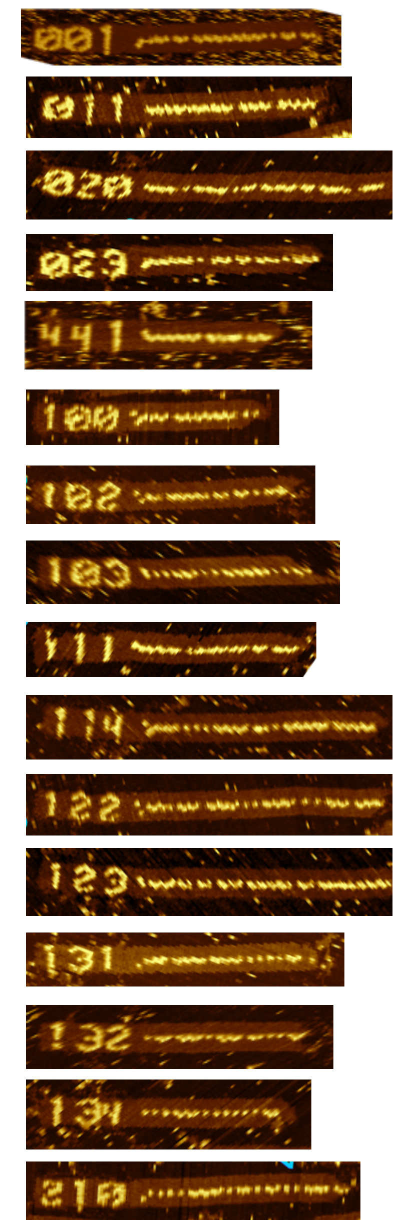


σ(100001) = 002
σ(100010) = 212
σ(100100) = 221
σ(100111) = 223
σ(101000) = 230
σ(101011) = 233
σ(101101) = 300
σ(101110) = 301
σ(110000) = 303
σ(110011) = 333
σ(110101) = 004
σ(111001) = 404
σ(111010) = 410
σ(111100) = 420
σ(111111) = 431

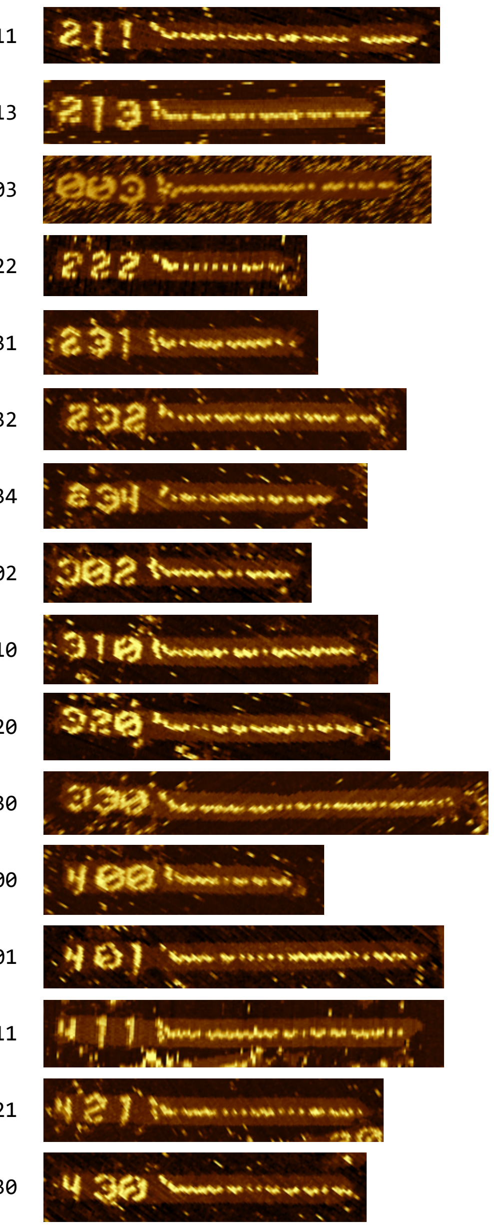


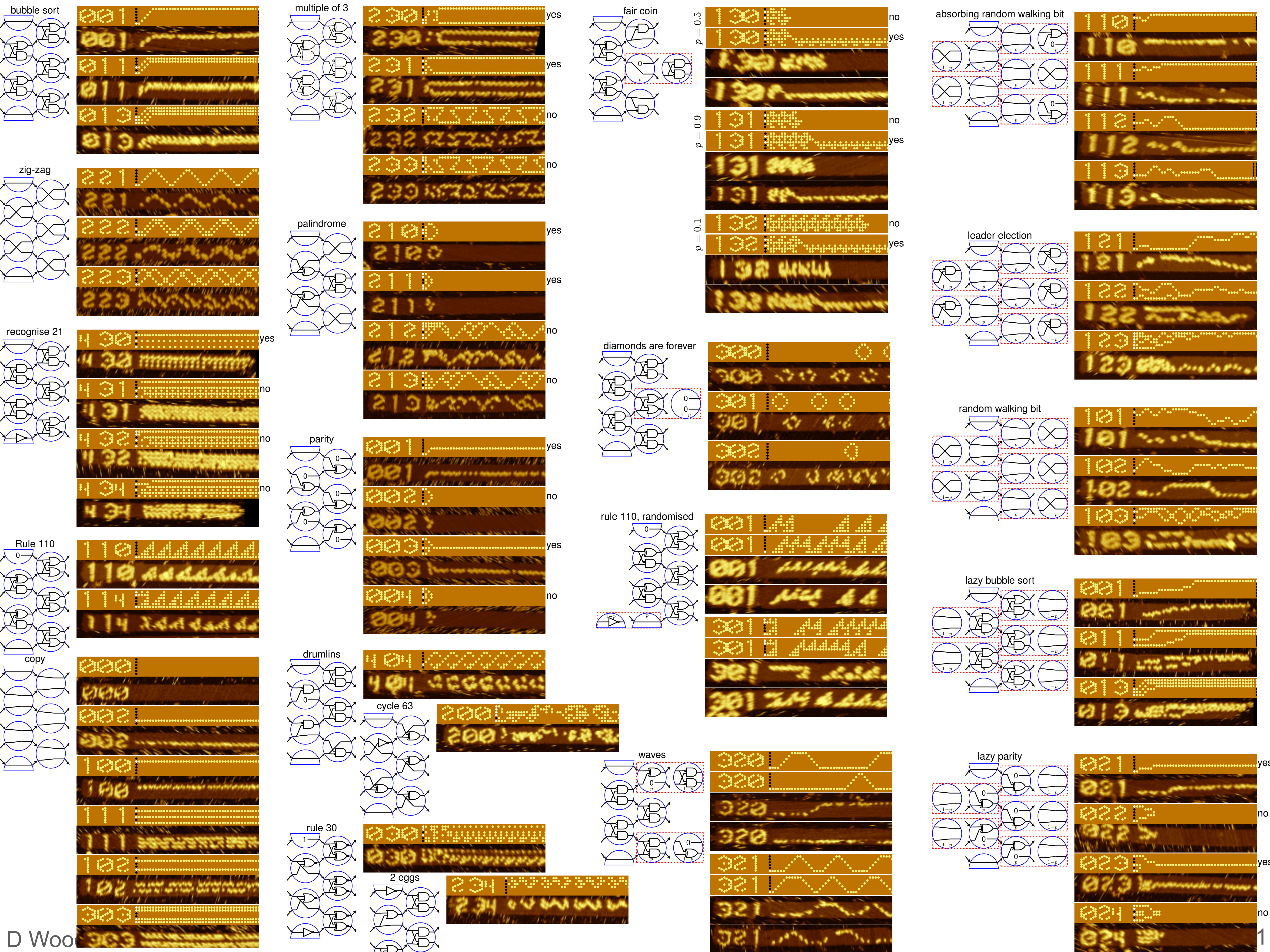
32 x Yes

σ(000001) = 001
σ(000010) = 011
σ(000100) = 020
σ(000111) = 023
σ(001000) = 441
σ(001011) = 100
σ(001101) = 102
σ(001110) = 103
σ(010000) = 111
σ(010011) = 114
σ(010101) = 122
σ(010110) = 123
σ(011001) = 131
σ(011010) = 132
σ(011100) = 134
σ(011111) = 210



σ(100000) = 211
σ(100011) = 213
σ(100101) = 003
σ(100110) = 222
σ(101001) = 231
σ(101010) = 232
σ(101100) = 234
σ(101111) = 302
σ(110001) = 310
σ(110010) = 320
σ(110100) = 330
σ(110111) = 400
σ(111000) = 401
σ(111011) = 411
σ(111101) = 421
σ(111110) = 430

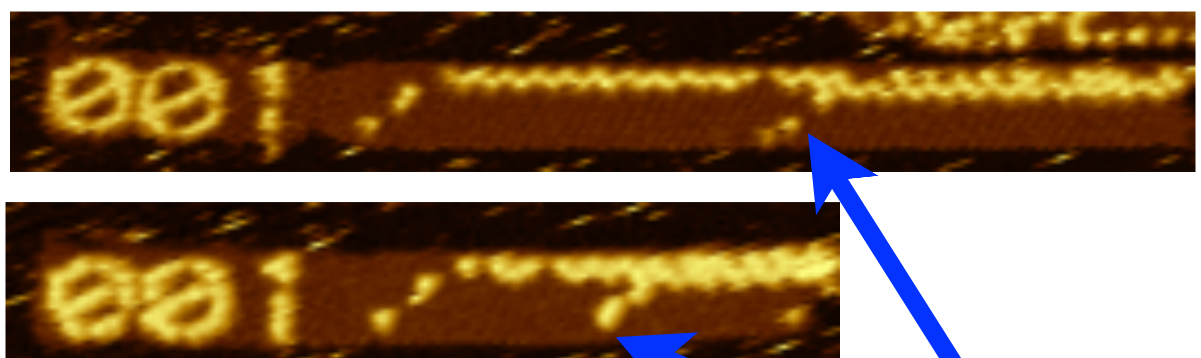


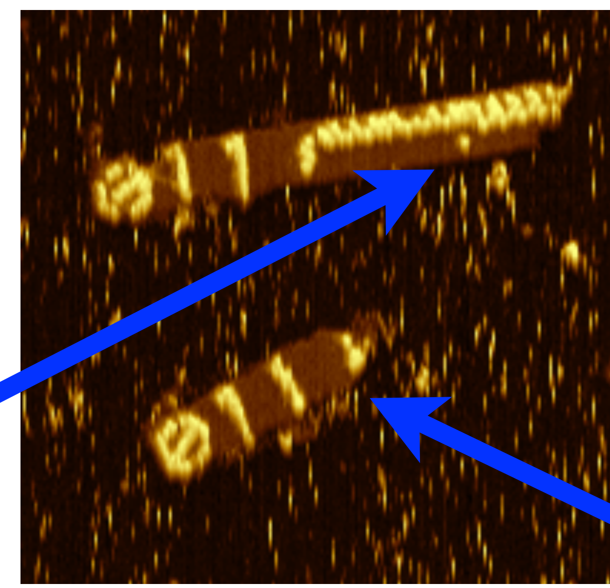

bubble sort
zig-zag
recognise 21
Rule 110
copy
multiple of 3
palindrome
parity
drumlins
rule 30
cycle 63
2 eggs
fair coin
$p = 0.5$
$p = 0.9$
$p = 0.1$
diamonds are forever
rule 110, randomised
waves
absorbing random walking bit
leader election
random walking bit
lazy bubble sort
lazy parity
yes
no

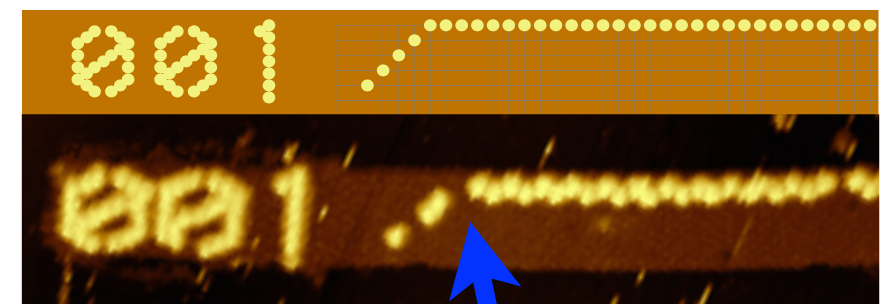
How well did the 21 circuits work?

Extensive testing of all 355 tiles:
- **every tile type** was used in some circuit
- for many circuits **tested all tile types for that circuit**
- ran one circuit on **all 64 inputs**

Analysed ~12k nanotubes with ~5M tile attachments:





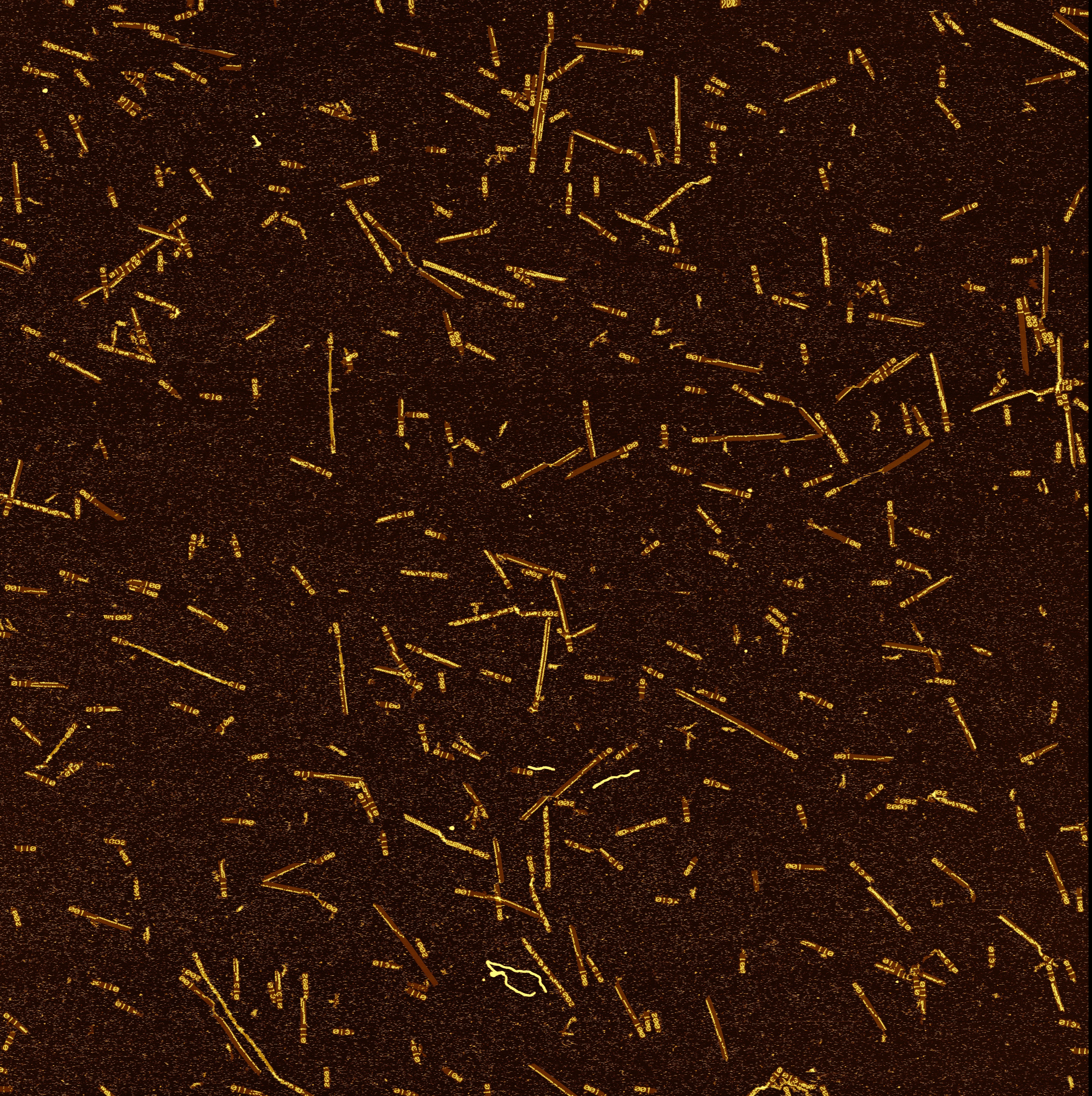


Errors: 0.03% of tile attachments were incorrect (i.e. error rate = 1/3000; 1,419 errors out of 4.6M attachments)

38.9% of seeds had no growth

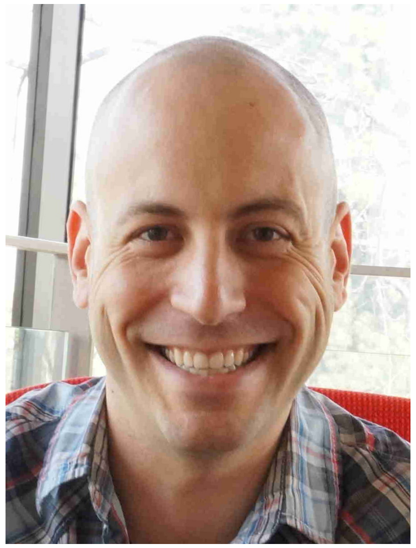
imperfect streptavidin labelling

Reprogrammable: demonstrated many new self-assembly programs
Scaling up: 15x more tile types than previous algorithmic self-assembly systems
Low error: Careful sequence design; Proofreading
Good structure: Nanotube lattice & hardcoded rows
Lots of tile types: Long SST domains



raw data
8µm x 8µm

Acknowledgements

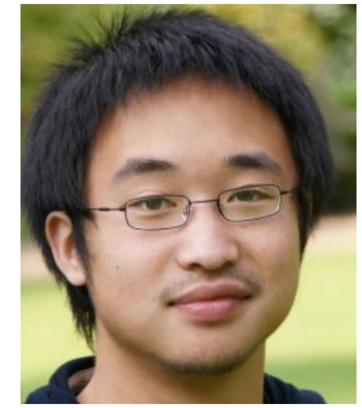


Dave Doty
UC Davis

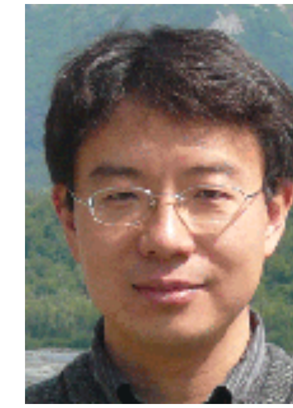
Erik Winfree
Caltech

C Myhrvold
Harvard

Joy Hui
Harvard



Felix Zhou
Oxford



Peng Yin
Harvard

Woods*, Doty*, Myhrvold, Hui, Zhou, Yin, Winfree.
Nature. 567:366-372. 2019
Diverse and robust molecular algorithms using reprogrammable DNA self-assembly

Special thanks to: Constantine Evans, Ashwin Gopinath, Paul Rothemund, Sungwook Woo, Cody Geary, Cris Moore, Chris Thachuk, Rizal Hariadi, Rebecca Schulman

Caltech

Inria

UC Davis

Harvard

NASA

NSF

Maynooth University
National University of Ireland Maynooth

erc

Hamilton Institute

Science Foundation Ireland
sfi For what's next

We are looking for postdocs & PhD students!

Fin

We are looking for postdocs and PhD students!

damien.woods@mu.ie

Science Foundation Ireland For what's next