Algorithmic self-assembly with DNA tiles
Tutorial

David Doty (UC-Davis)

23rd International Meeting on DNA Computing and Molecular Programming

University of Texas–Austin

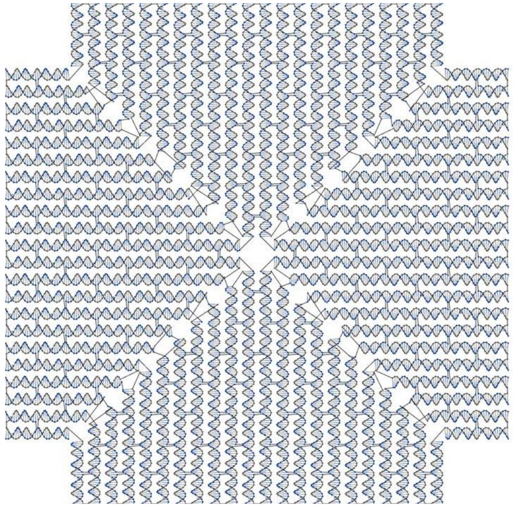
September 2017

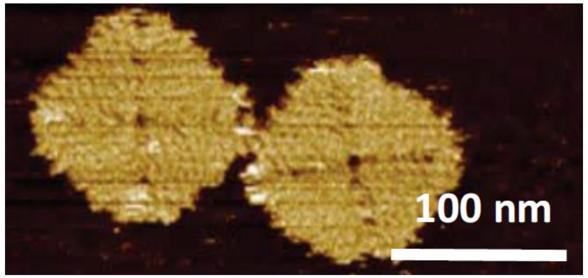
DNA tile self-assembly

DNA tile self-assembly

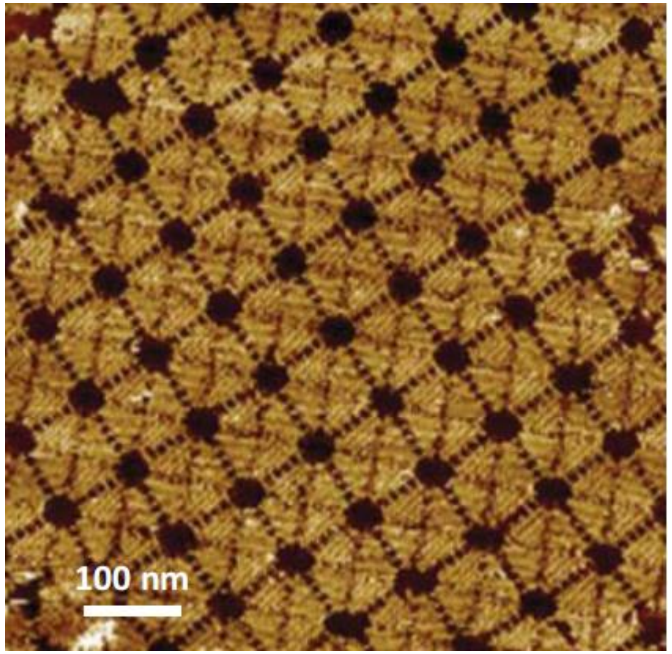
monomers ("tiles" made from DNA) bind into a crystal lattice

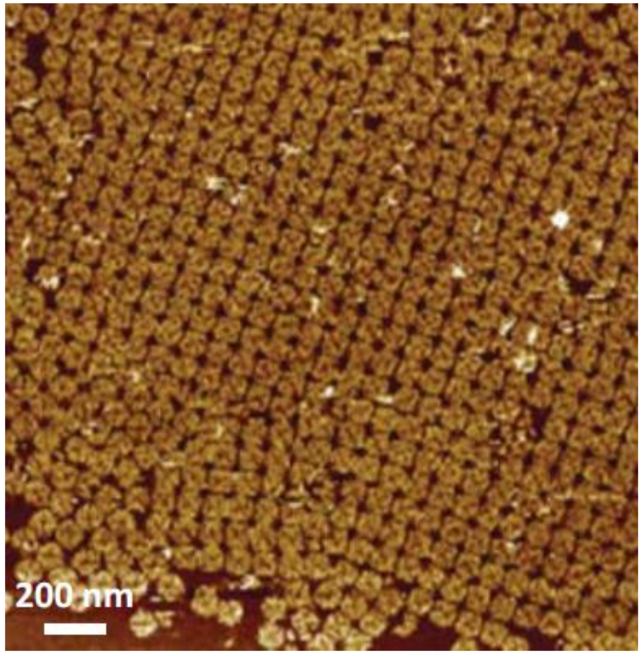
tile





lattice

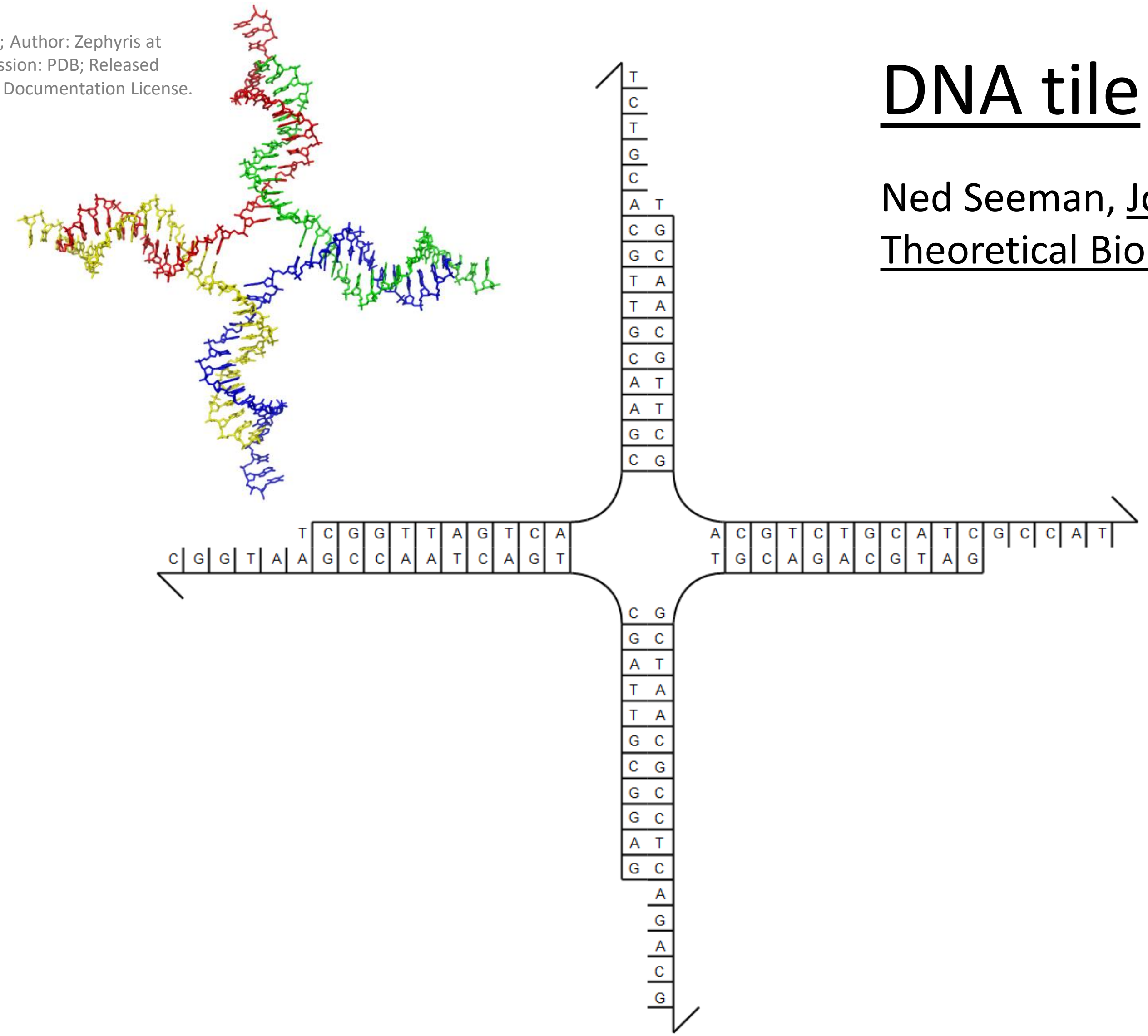




Source: *Programmable disorder in random DNA tilings*. Tikhomirov, Petersen, Qian, Nature Nanotechnology 2017

Practice of DNA tile self-assembly

Source:en.wikipedia; Author: Zephyris at en.wikipedia; Permission: PDB; Released under the GNU Free Documentation License.

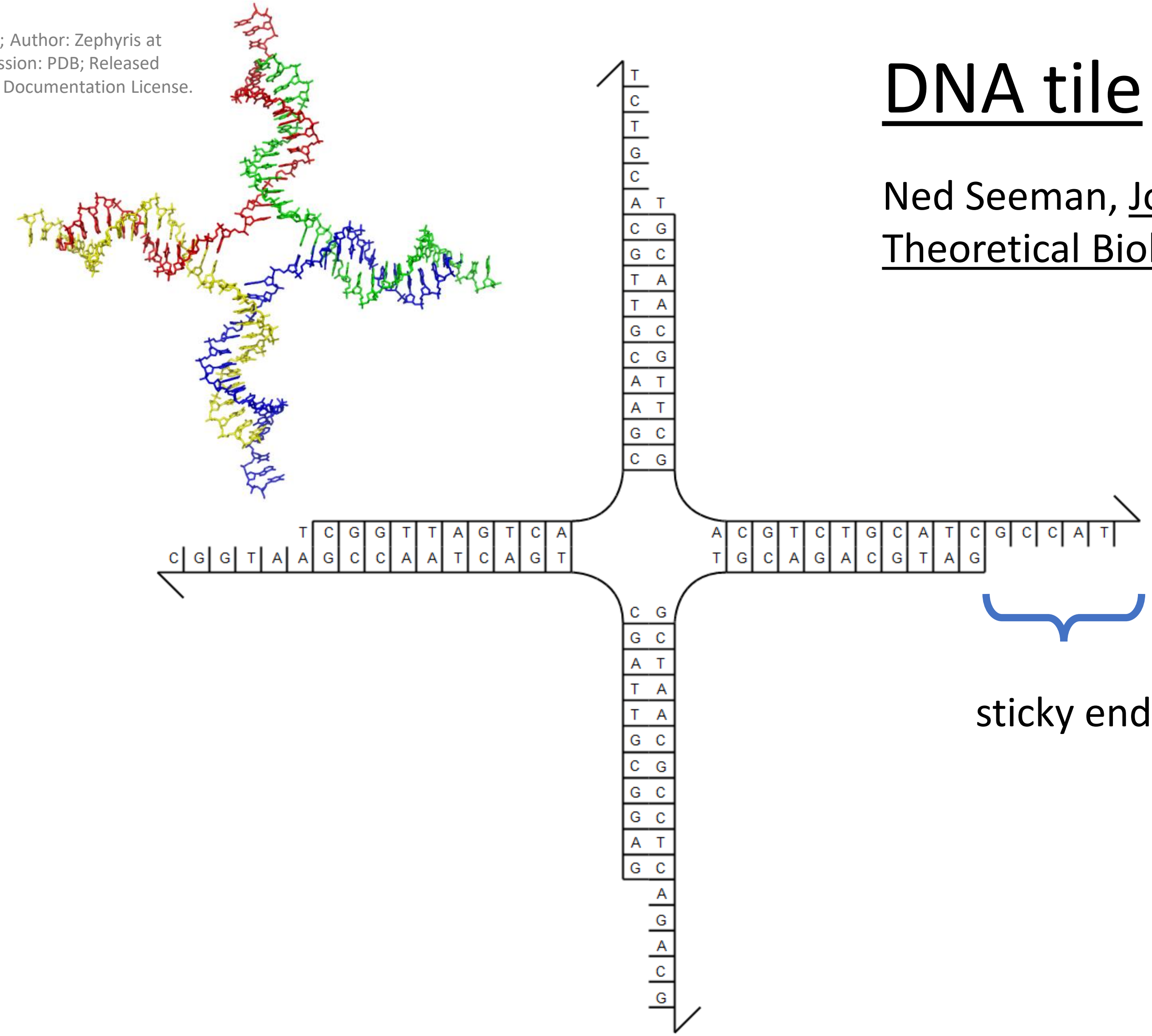


DNA tile

Ned Seeman, Journal of Theoretical Biology 1982

Practice of DNA tile self-assembly

Source:en.wikipedia; Author: Zephyris at en.wikipedia; Permission: PDB; Released under the GNU Free Documentation License.

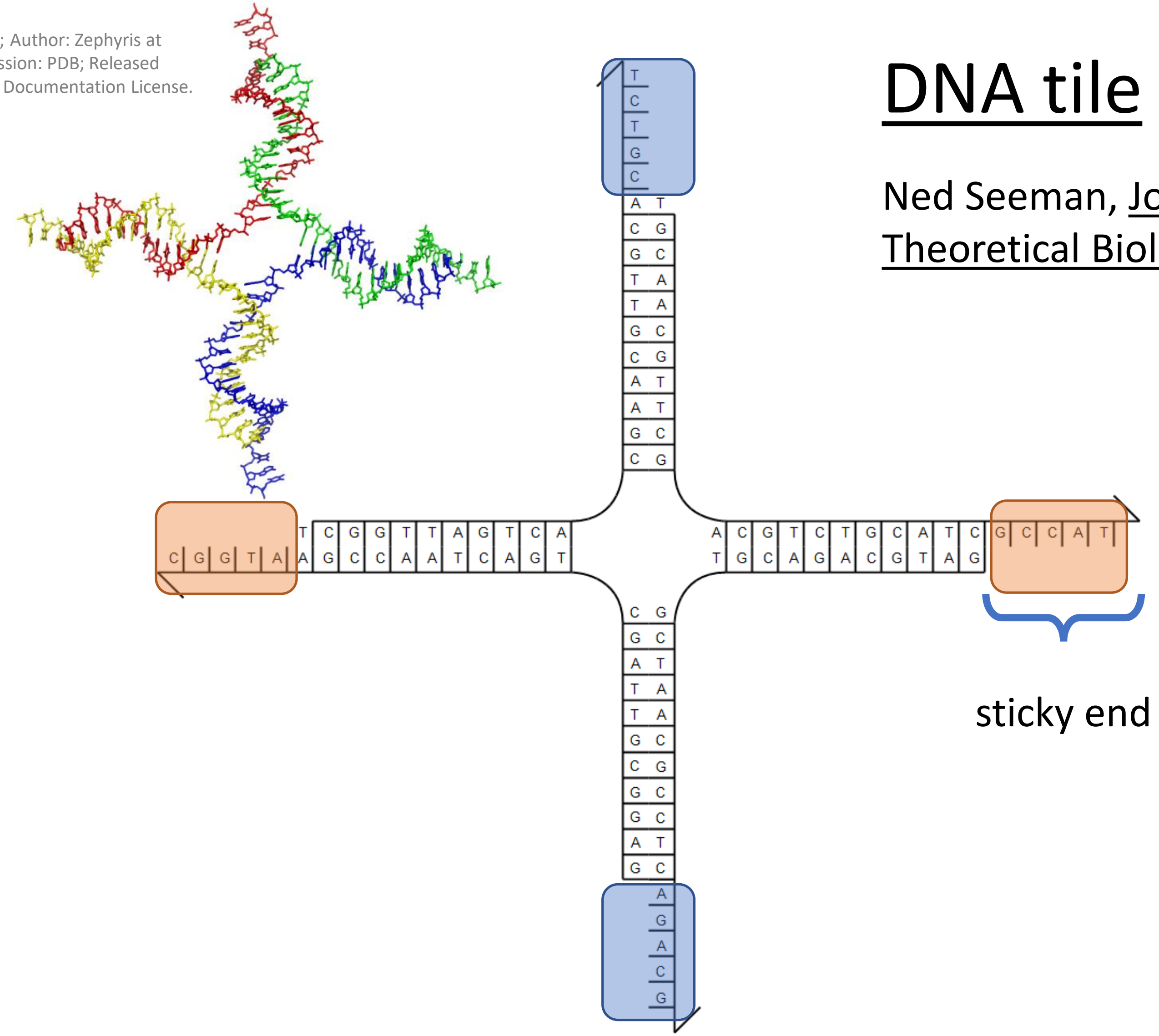


sticky end

DNA tile

Ned Seeman, Journal of Theoretical Biology 1982

Practice of DNA tile self-assembly



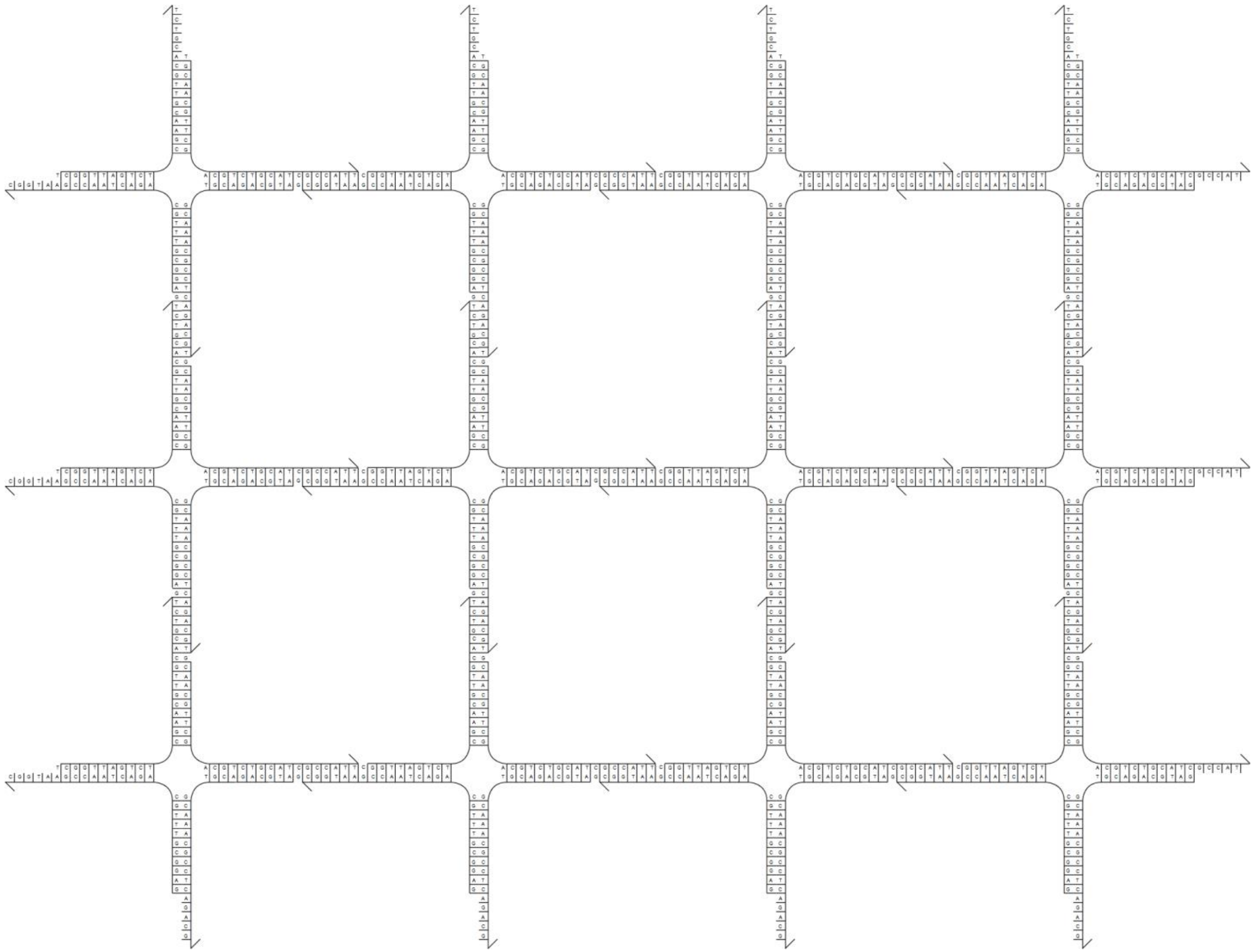
sticky end

DNA tile

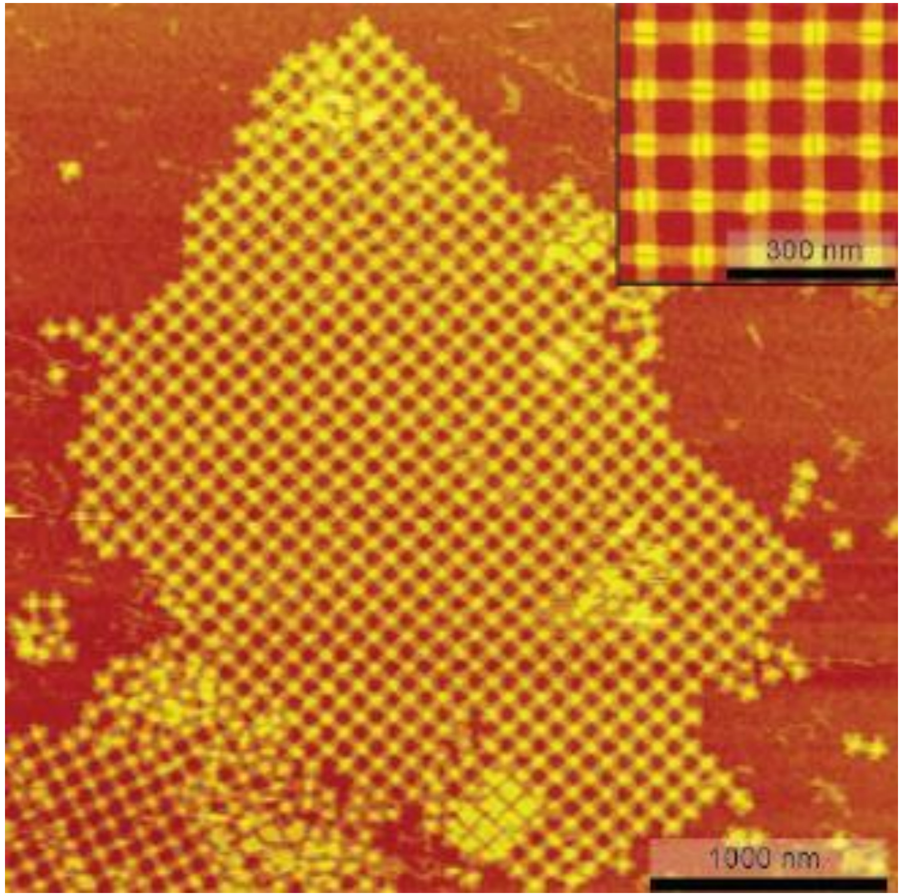
Ned Seeman, Journal of Theoretical Biology 1982

Practice of DNA tile self-assembly

Place many copies of DNA tile in solution…



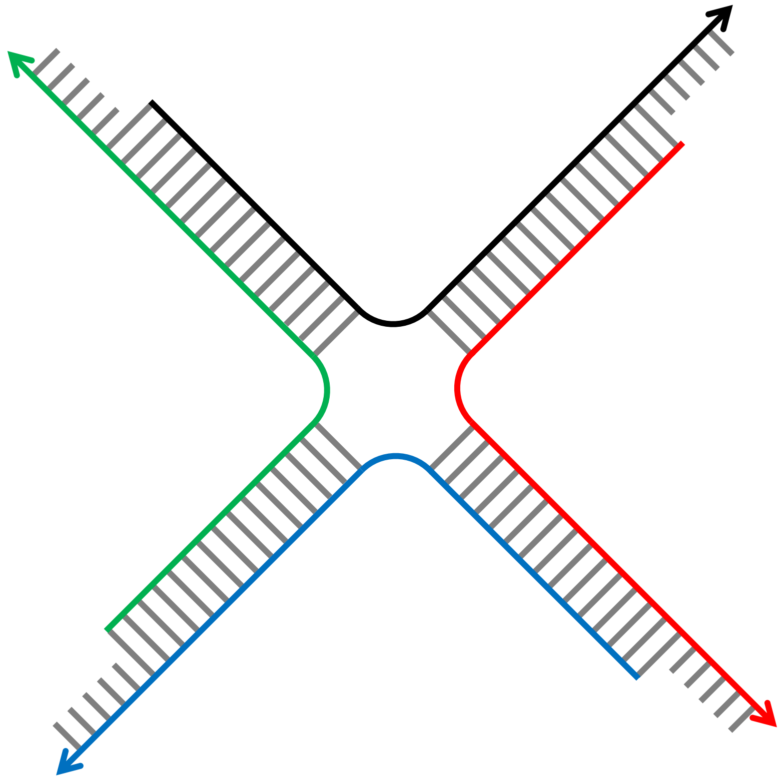
(not the same tile motif in this image)



Liu, Zhong, Wang, Seeman, Angewandte Chemie 2011

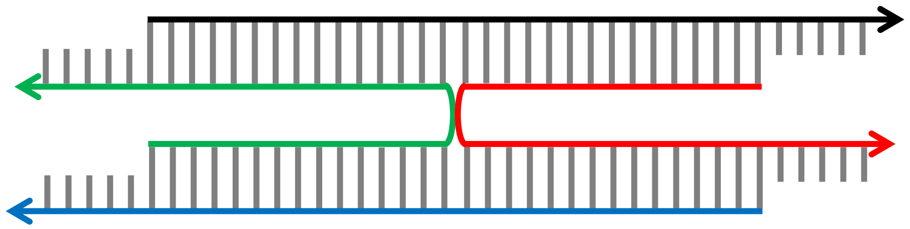
Practice of DNA tile self-assembly

What really happens in practice to Holliday junction ("base stacking")



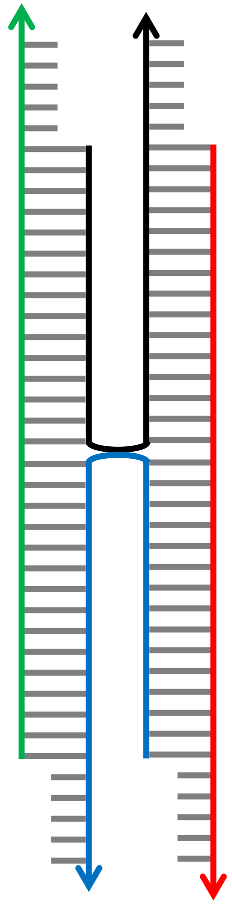
Practice of DNA tile self-assembly

What really happens in practice to Holliday junction ("base stacking")



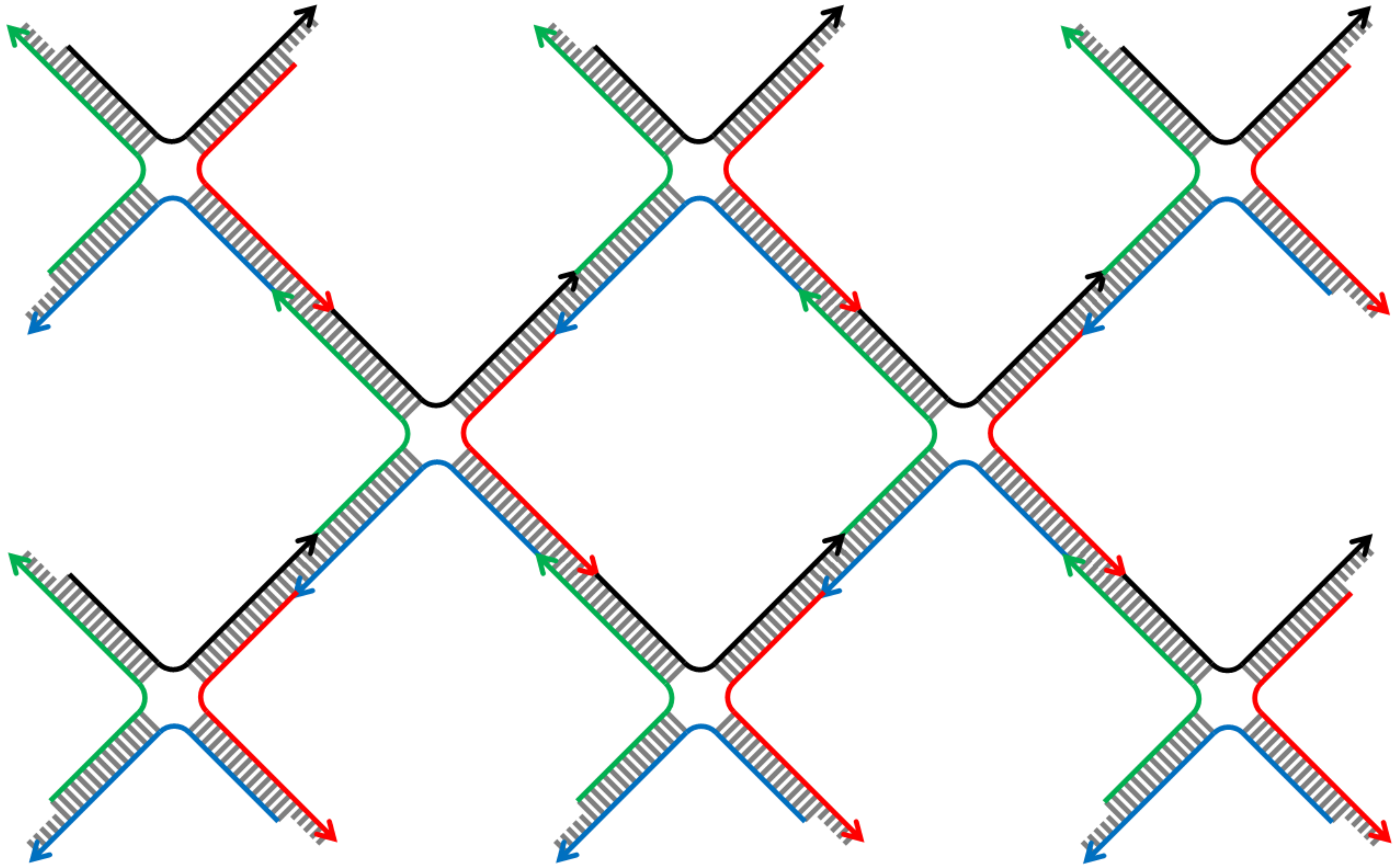
Practice of DNA tile self-assembly

What really happens in practice to Holliday junction ("base stacking")



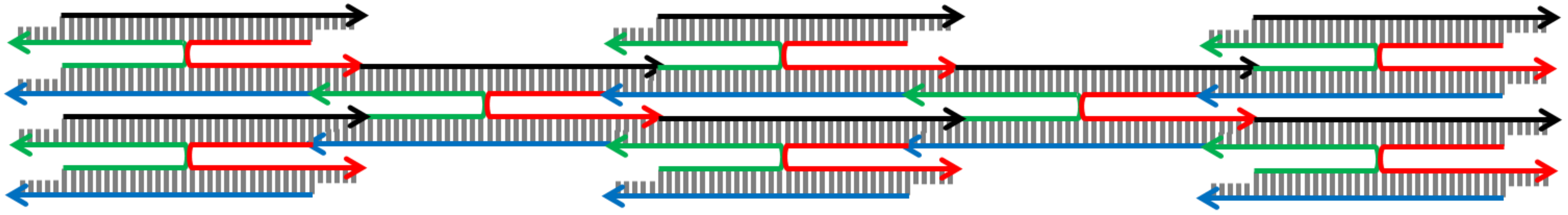
Practice of DNA tile self-assembly

What really happens in practice to Holliday junction ("base stacking")

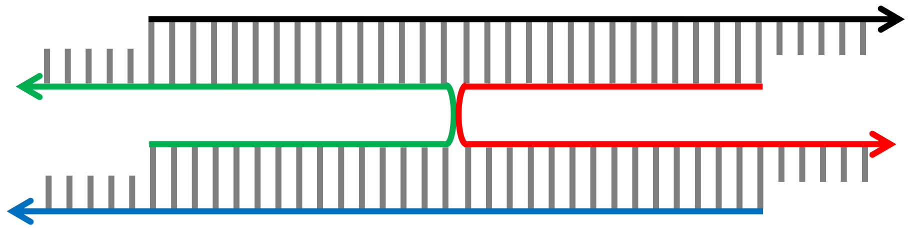


Practice of DNA tile self-assembly

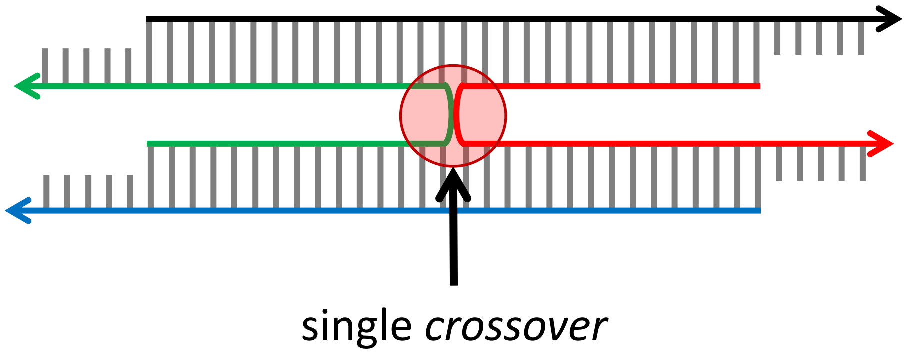
What really happens in practice to Holliday junction ("base stacking")



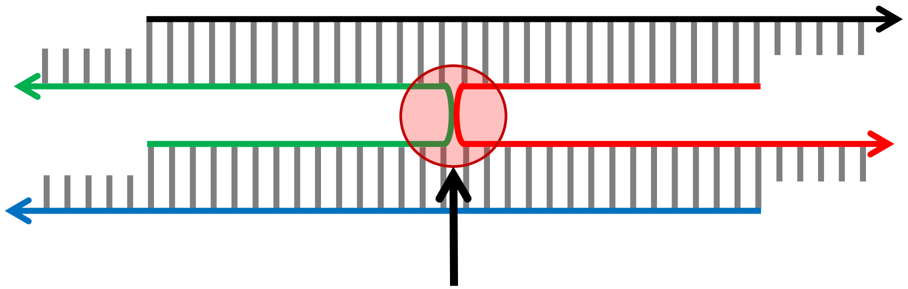
Practice of DNA tile self-assembly



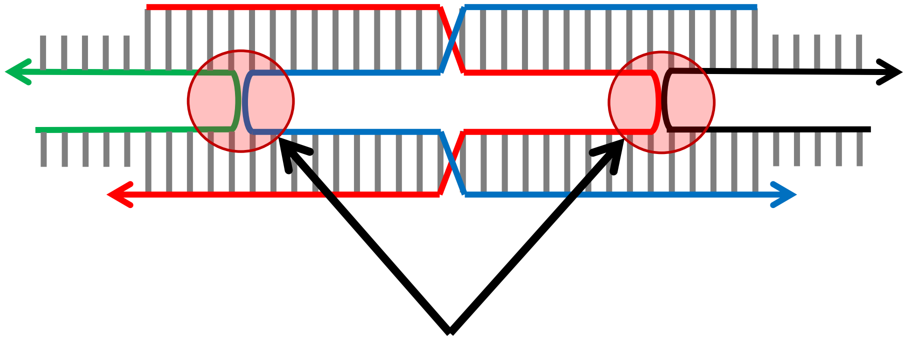
Practice of DNA tile self-assembly



Practice of DNA tile self-assembly



single *crossover*



double crossover

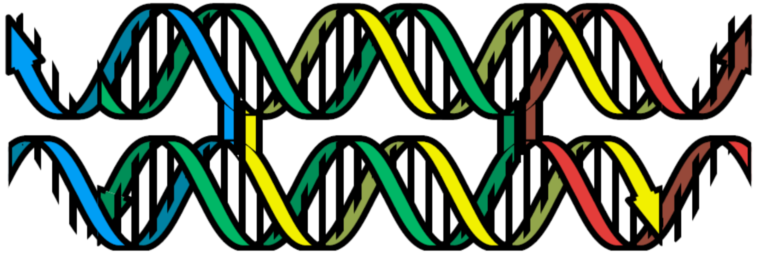
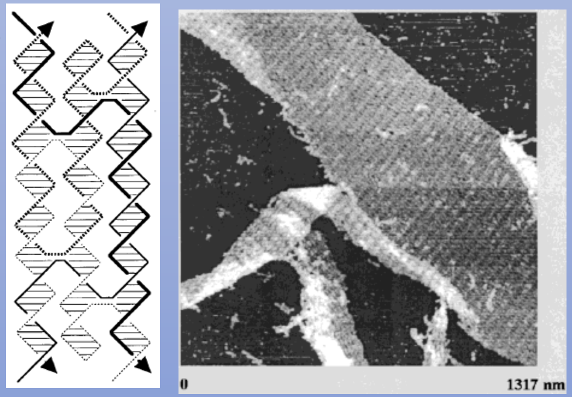


Figure from Schulman, Winfree, *PNAS* 2009

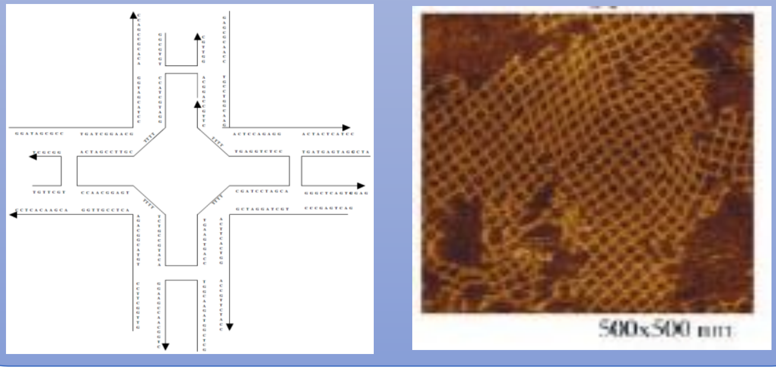
Practice of DNA tile self-assembly

other tile motifs

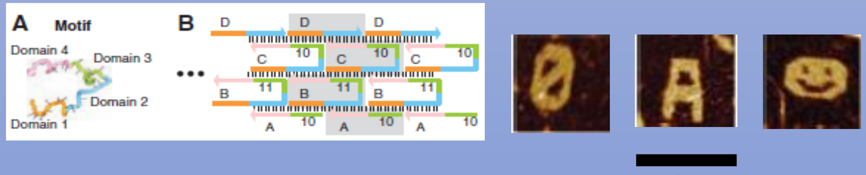
triple-crossover tile (LaBean, Yan, Kopatsch, Liu, Winfree, Reif, Seeman, *JACS* 2000)



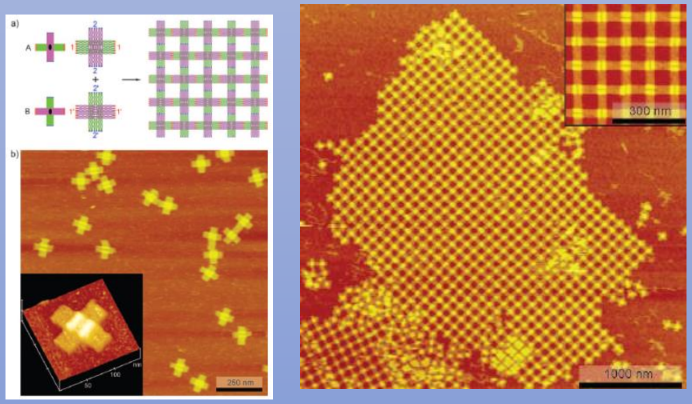
4x4 tile (Yan, Park, Finkelstein, Reif, LaBean, *Science* 2003)



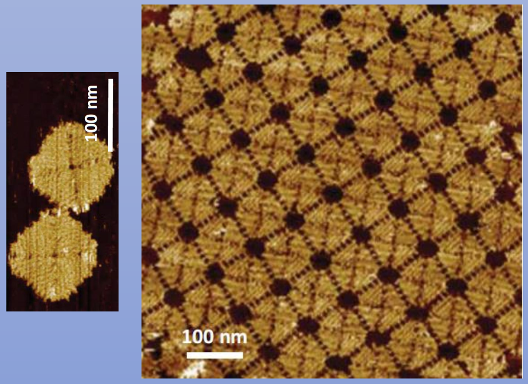
single-stranded tile (Yin, Hariadi, Sahu, Choi, Park, LaBean, Reif, *Science* 2008)



DNA origami tile (Liu, Zhong, Wang, Seeman, *Angewandte Chemie* 2011)



Tikhomirov, Petersen, Qian, Nature Nanotechnology 2017



Theory of *algorithmic* self-assembly

What if…

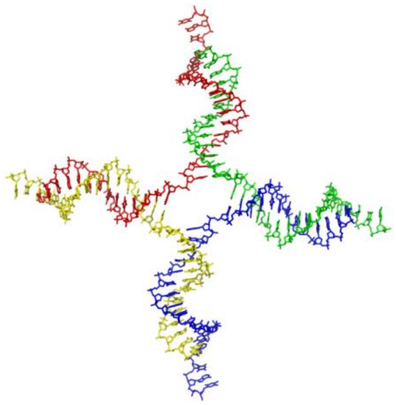
… there is more than one tile type?

… some sticky ends are "weak"?

Erik Winfree

Abstract Tile Assembly Model

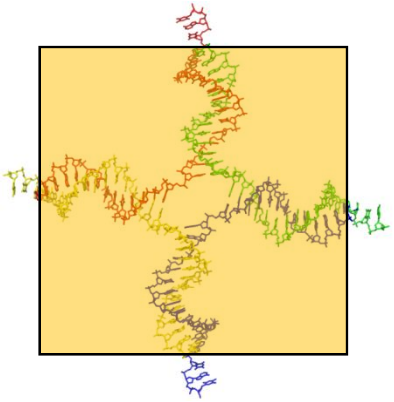
Erik Winfree,
Ph.D. thesis,
Caltech 1998



Abstract Tile Assembly Model

Erik Winfree,
Ph.D. thesis,
Caltech 1998

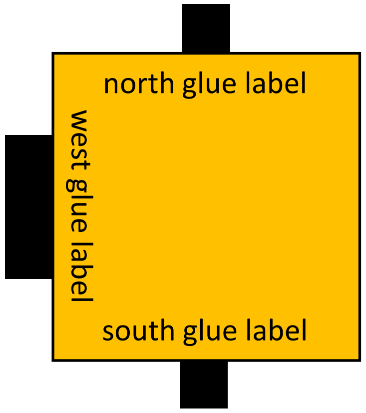
- **tile type** = unit square

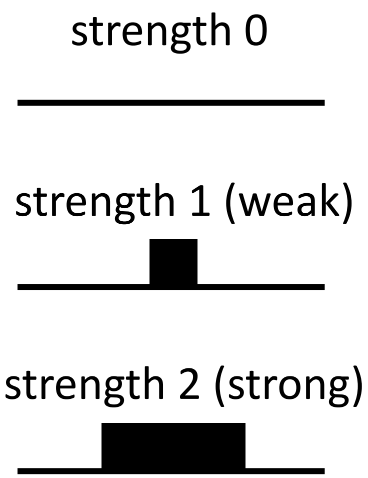


Abstract Tile Assembly Model

Erik Winfree, Ph.D. thesis, Caltech 1998

- **tile type** = unit square
- each side has a **glue** with a **label** and **strength** (0, 1, or 2)

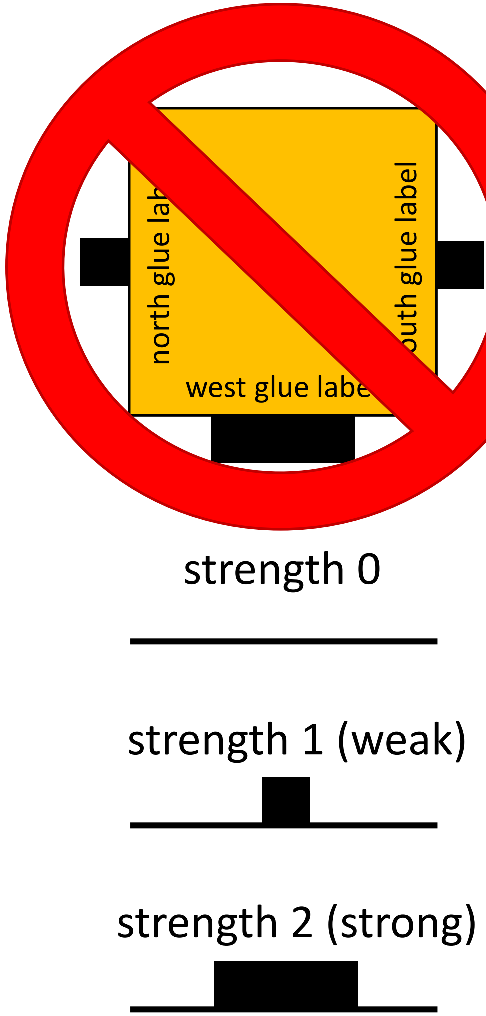




Abstract Tile Assembly Model

Erik Winfree,
Ph.D. thesis,
Caltech 1998

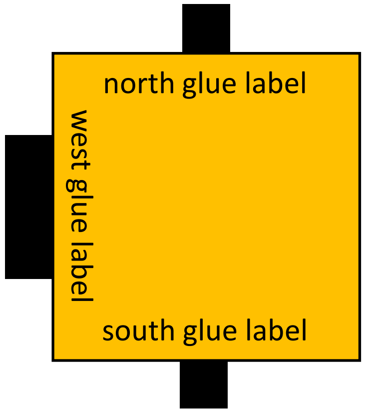
- **tile type** = unit square
- each side has a **glue** with a **label** and **strength** (0, 1, or 2)
- tiles cannot rotate

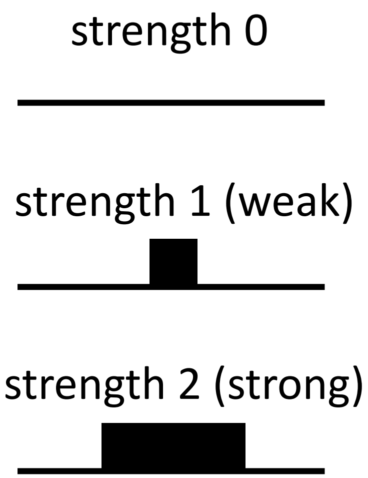


Abstract Tile Assembly Model

Erik Winfree,
Ph.D. thesis,
Caltech 1998

- **tile type** = unit square
- each side has a **glue** with a **label** and **strength** (0, 1, or 2)
- tiles cannot rotate



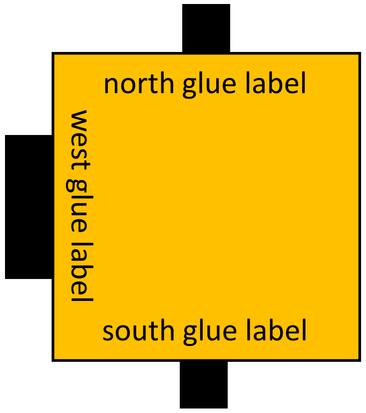


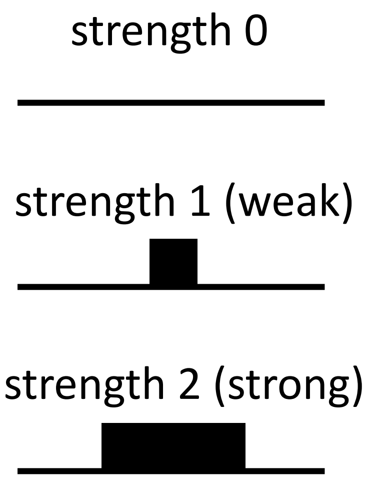
- finitely many tile **types**
- infinitely many **tiles**: copies of each type

Abstract Tile Assembly Model

Erik Winfree,
Ph.D. thesis,
Caltech 1998

- **tile type** = unit square
- each side has a **glue** with a **label** and **strength** (0, 1, or 2)
- tiles cannot rotate



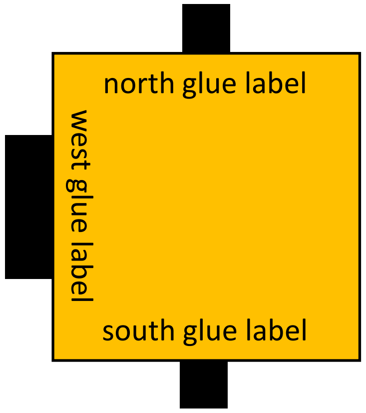


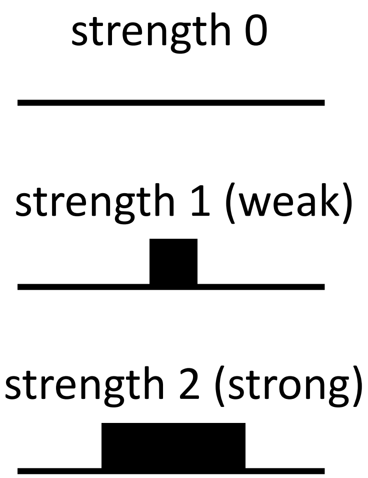
- finitely many tile **types**
- infinitely many **tiles**: copies of each type
- assembly starts as a single copy of a special **seed** tile

Abstract Tile Assembly Model

Erik Winfree,
Ph.D. thesis,
Caltech 1998

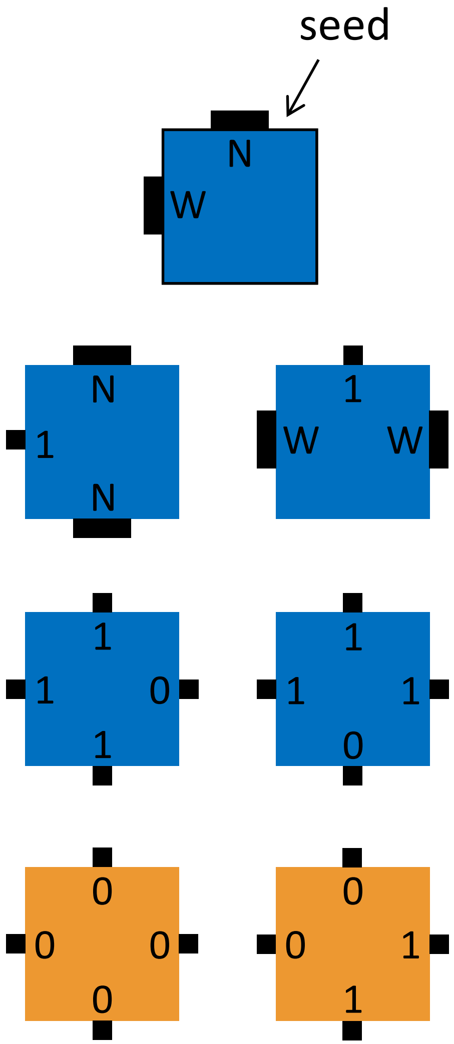
- **tile type** = unit square
- each side has a **glue** with a **label** and **strength** (0, 1, or 2)
- tiles cannot rotate



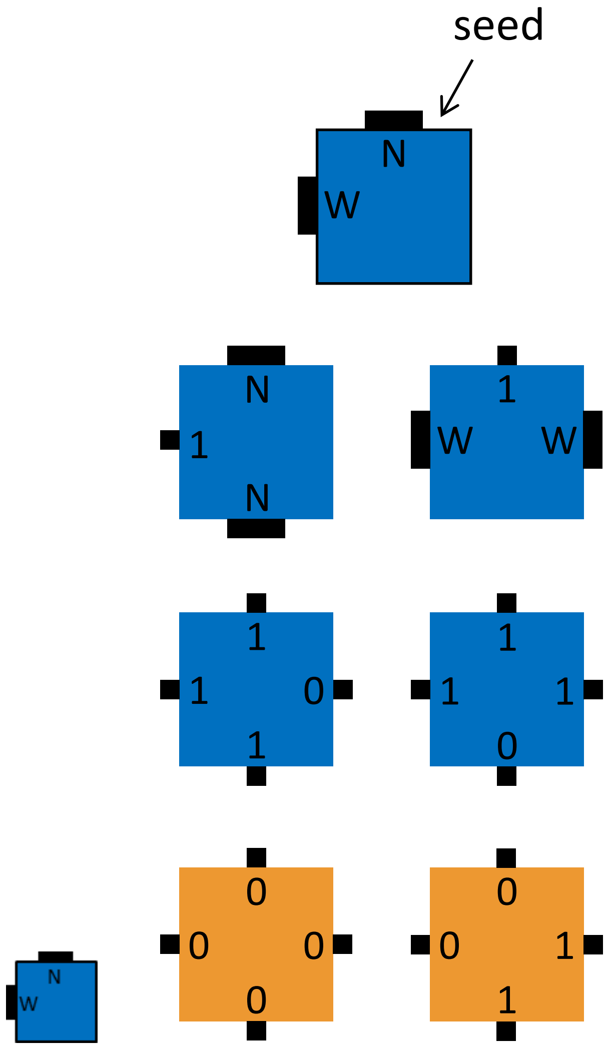


- finitely many tile **types**
- infinitely many **tiles**: copies of each type
- assembly starts as a single copy of a special **seed** tile
- tile can bind to the assembly if total binding strength ≥ 2 (two weak glues or one strong glue)

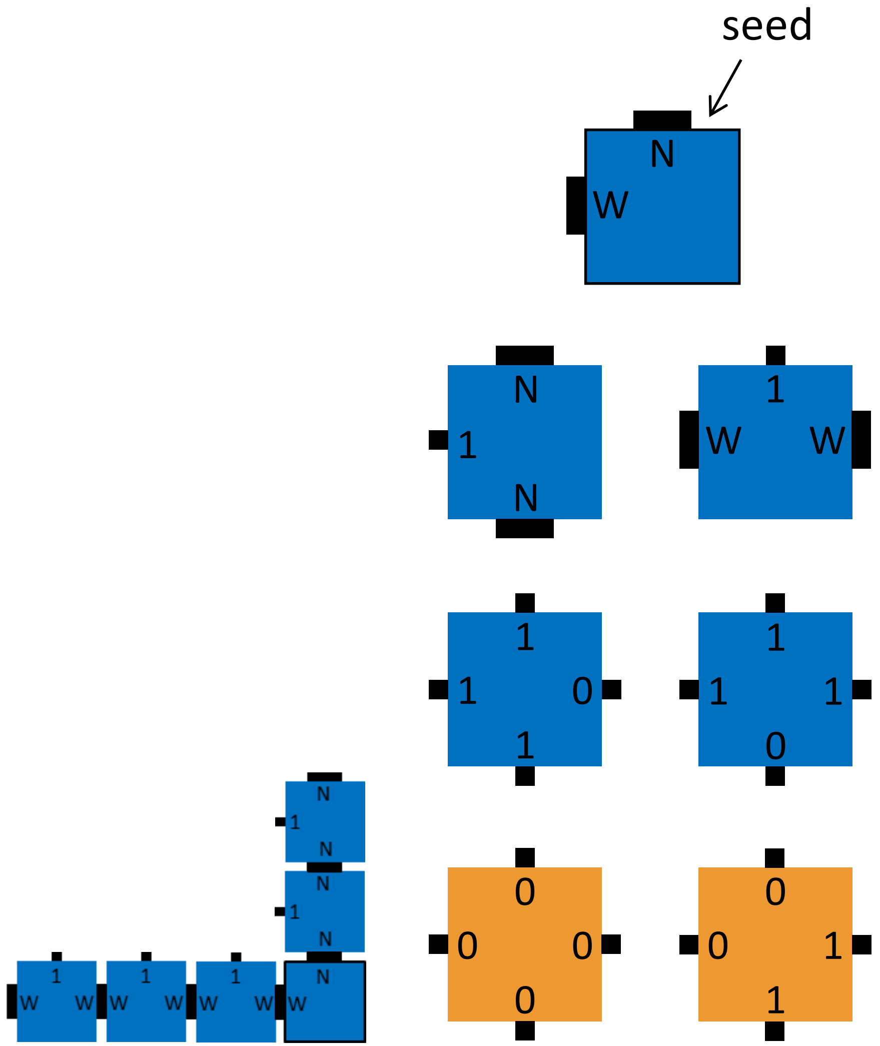
Example tile set



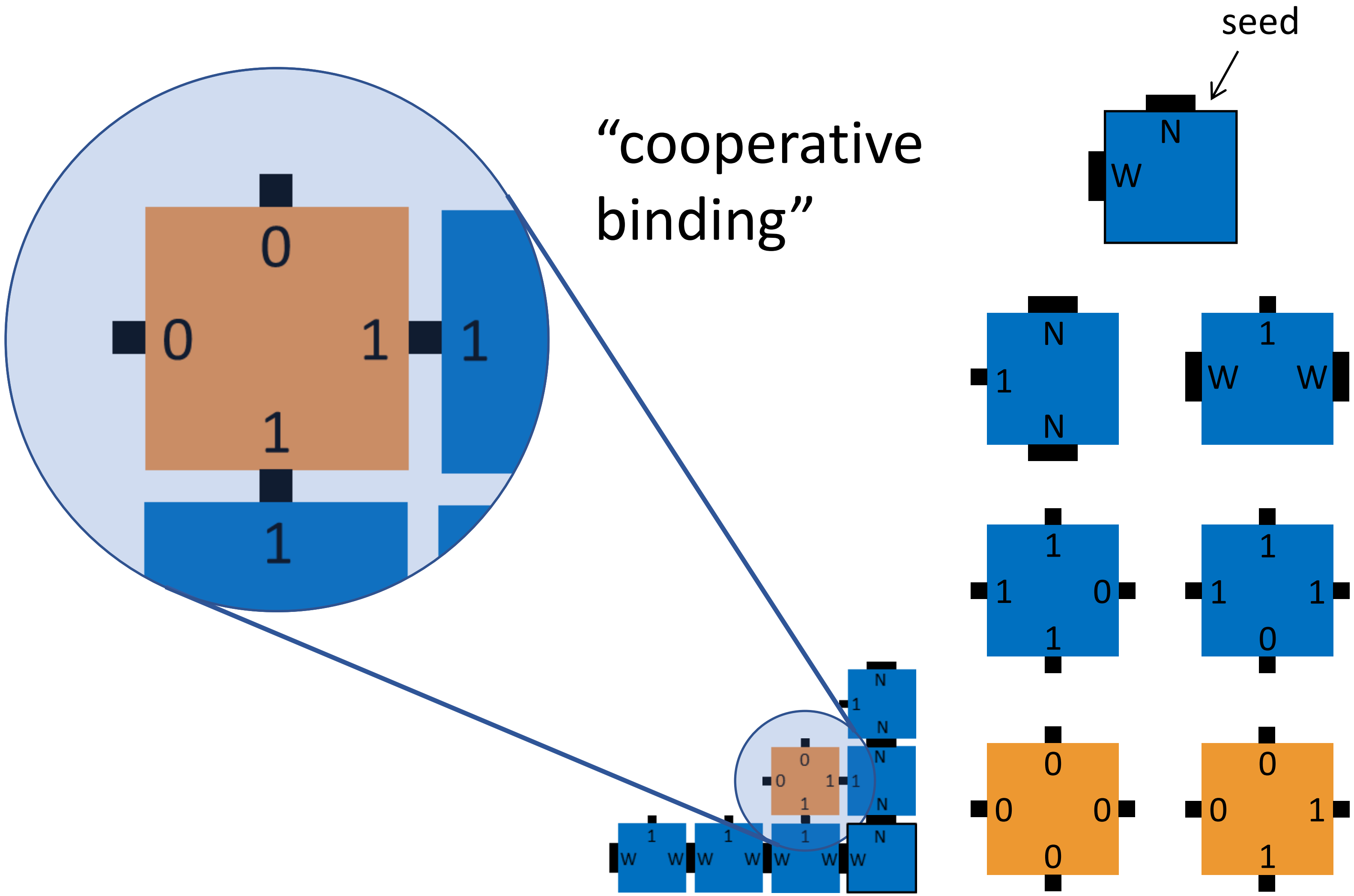
Example tile set



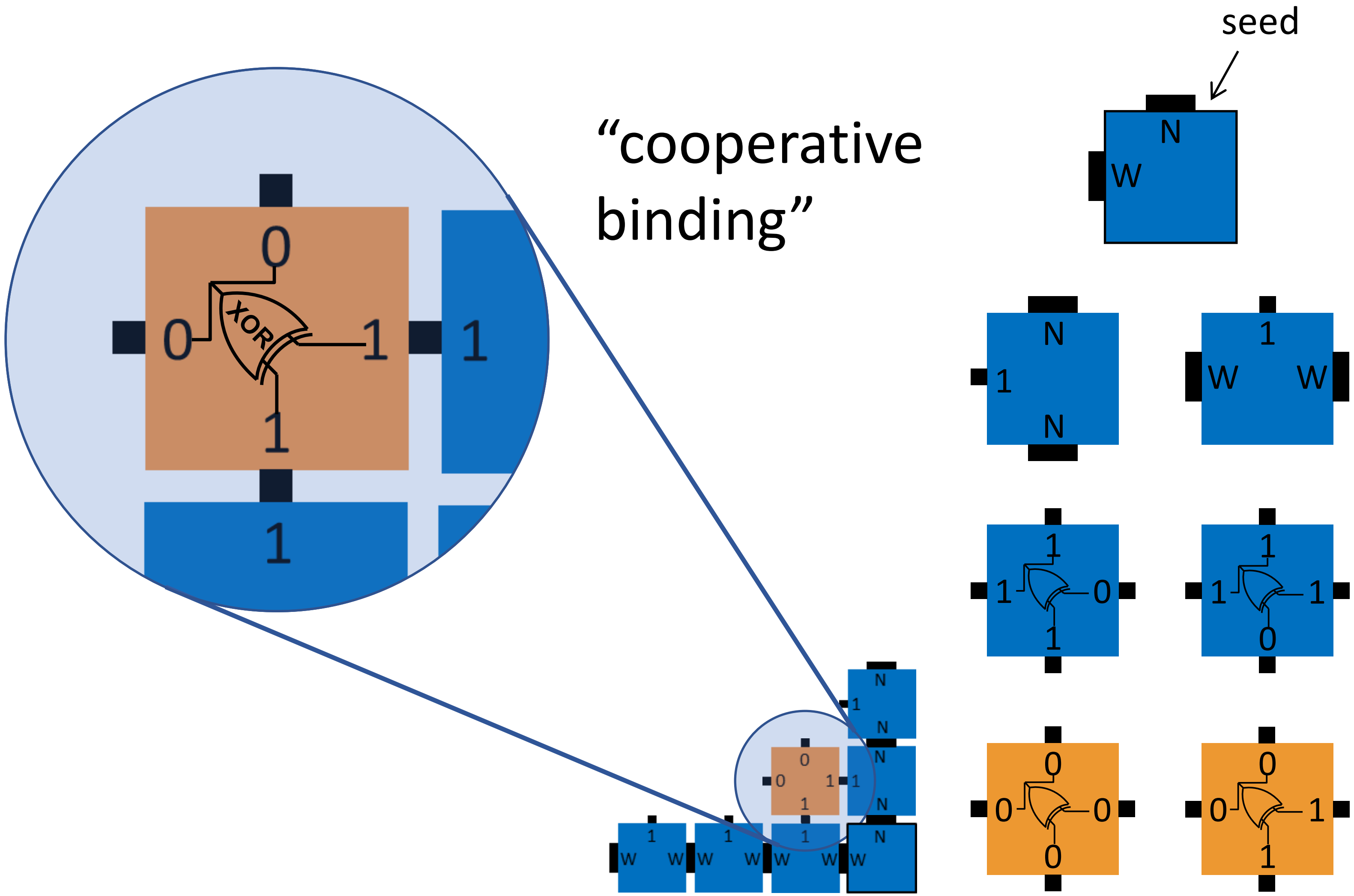
Example tile set



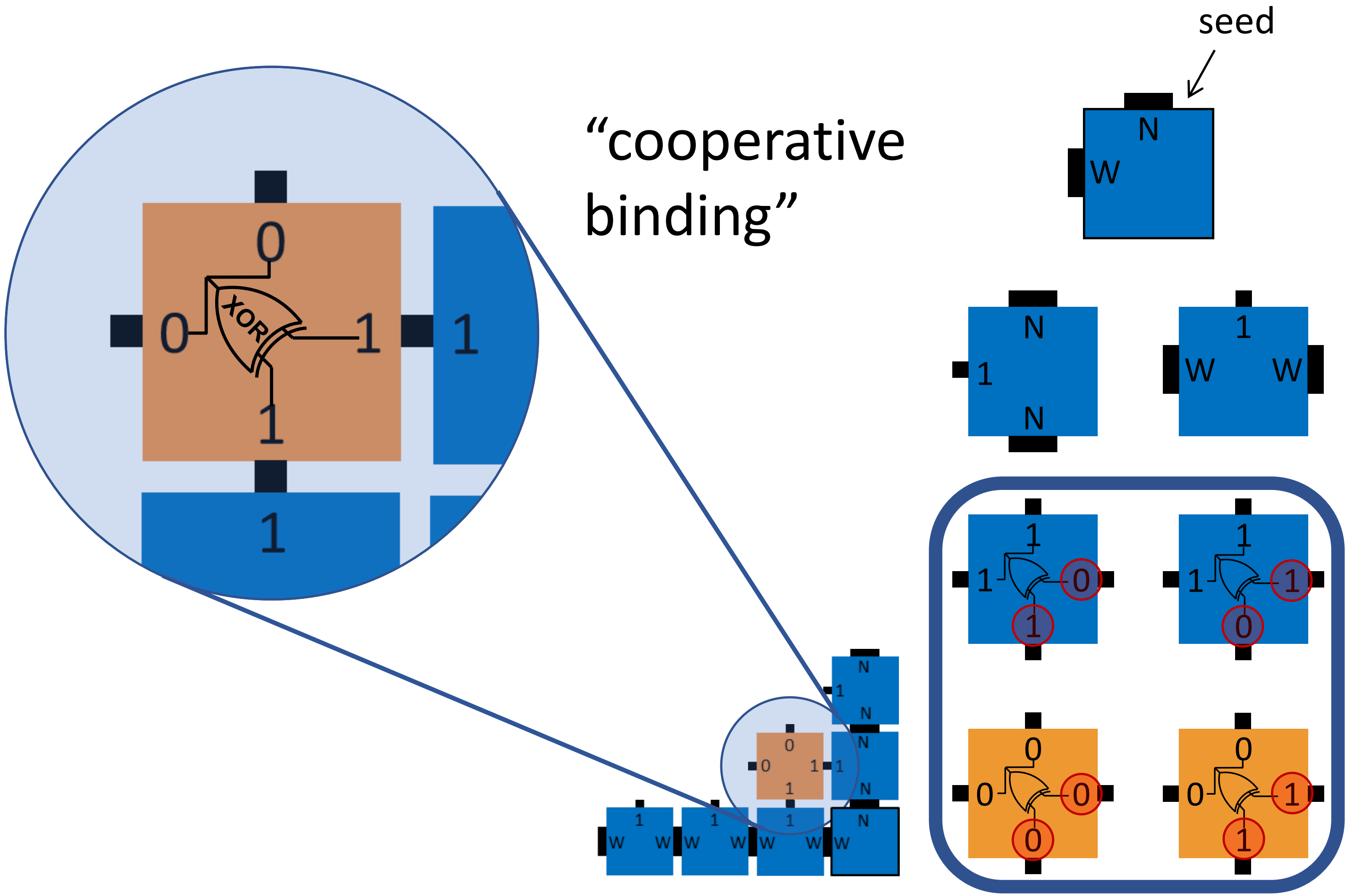
Example tile set



Example tile set

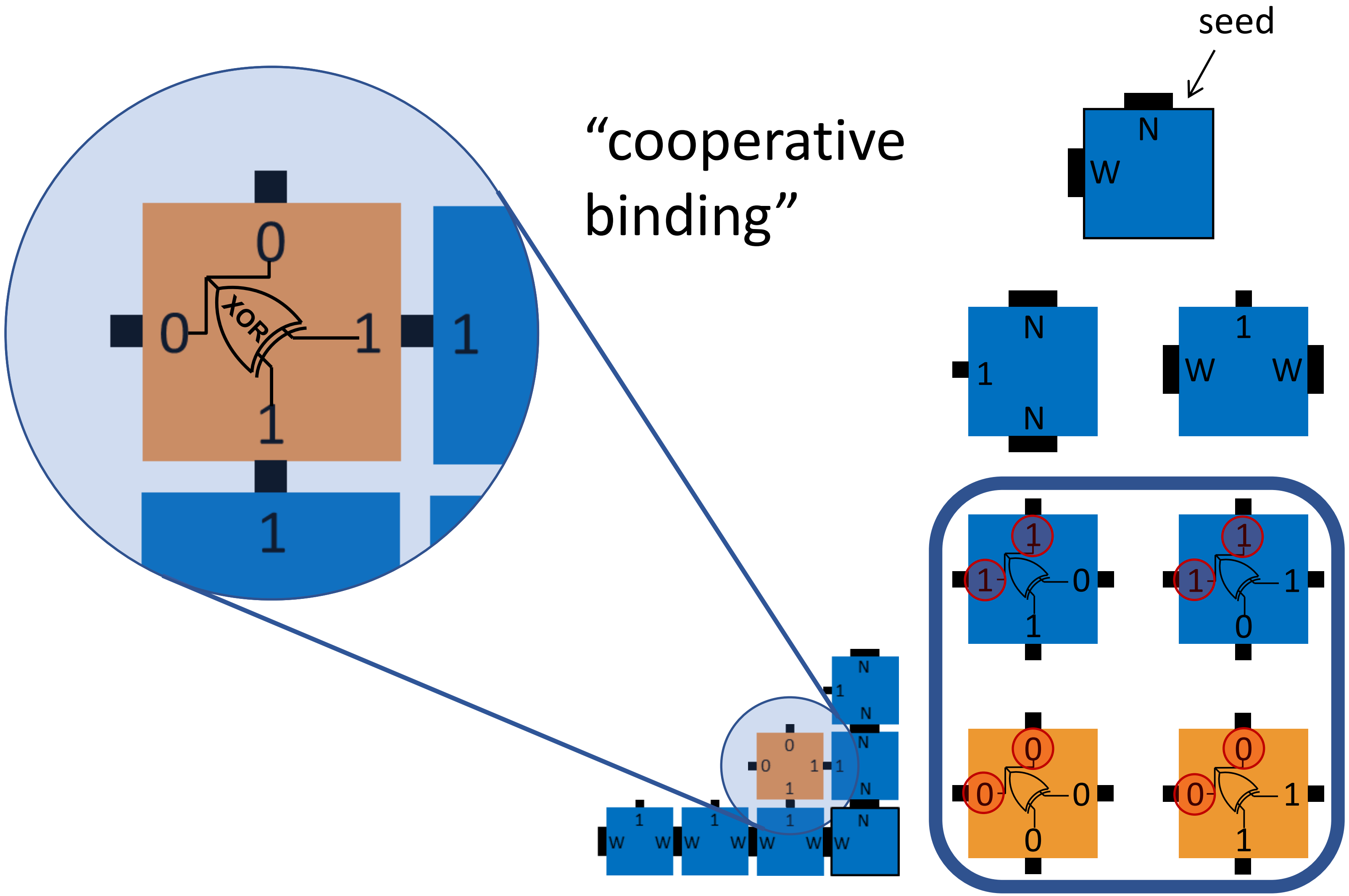


Example tile set

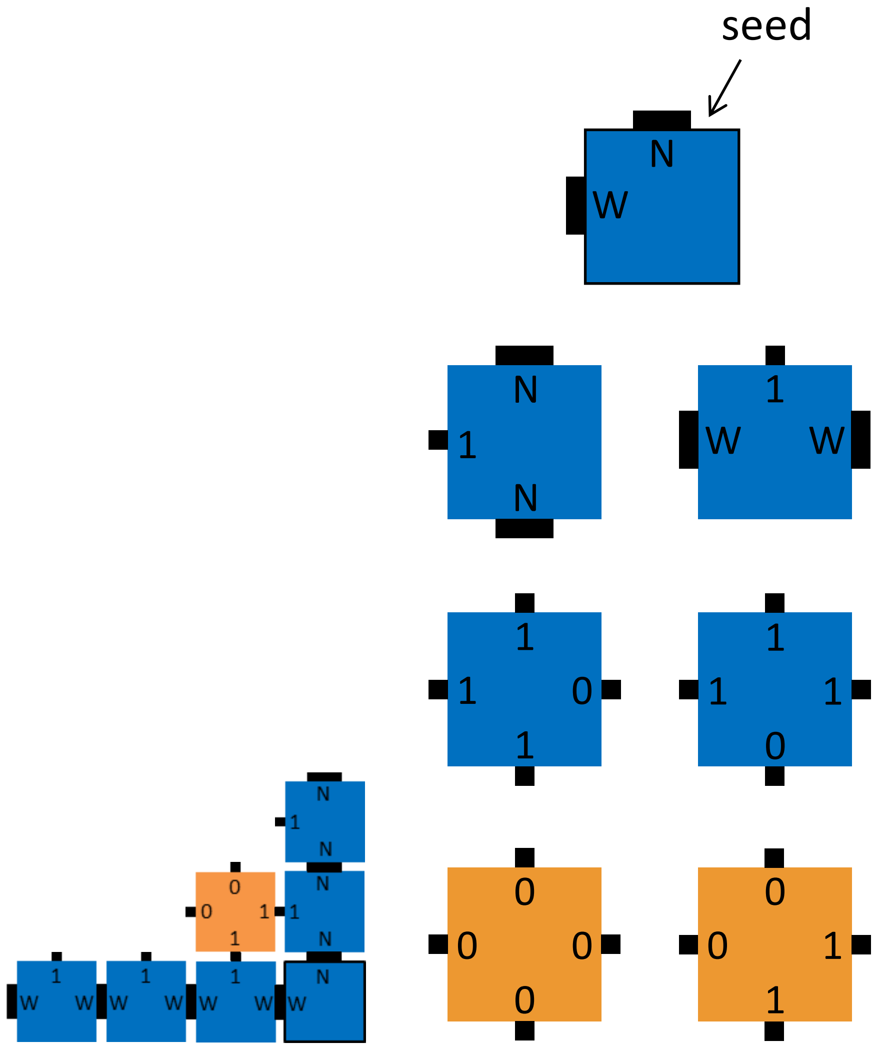


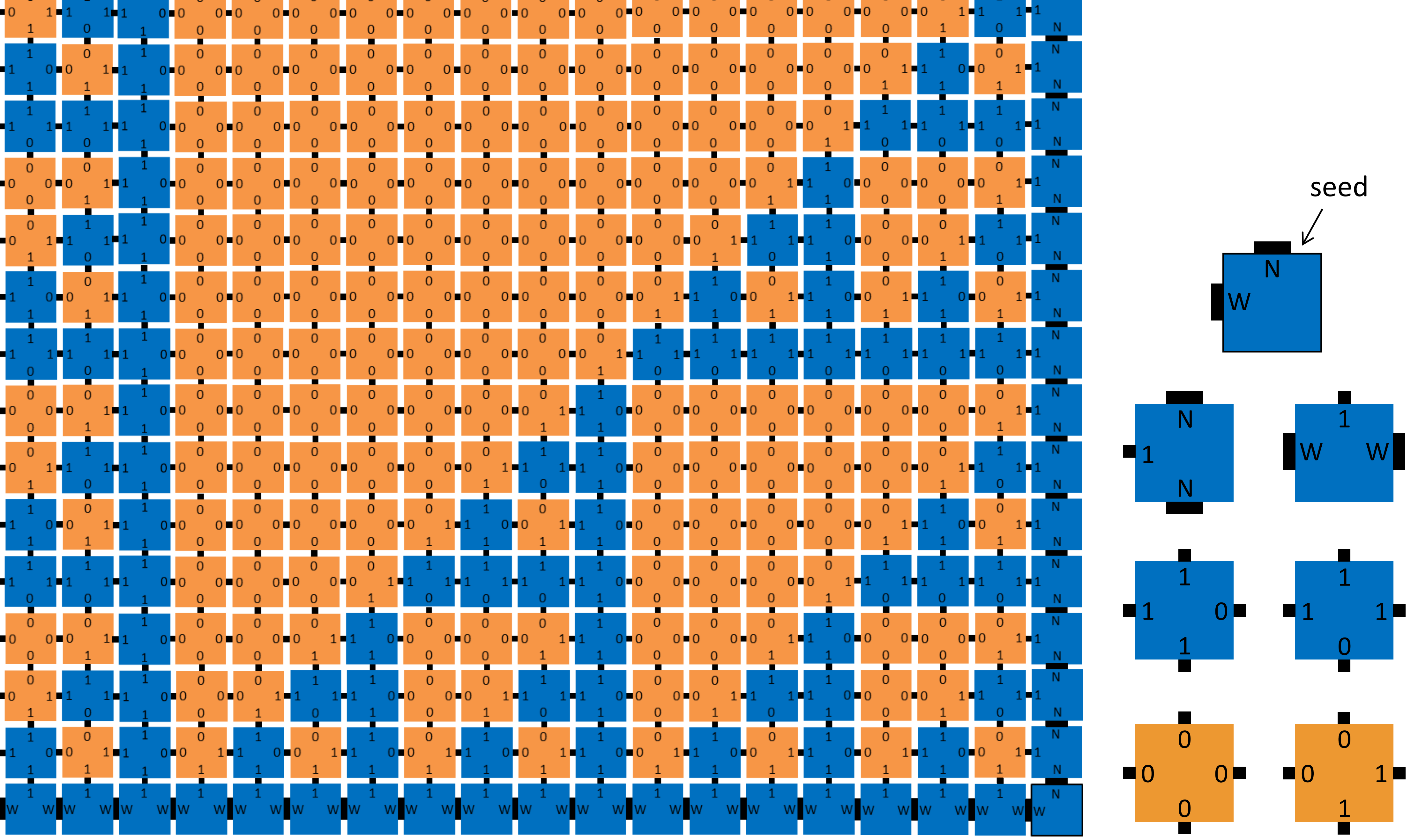
Example tile set

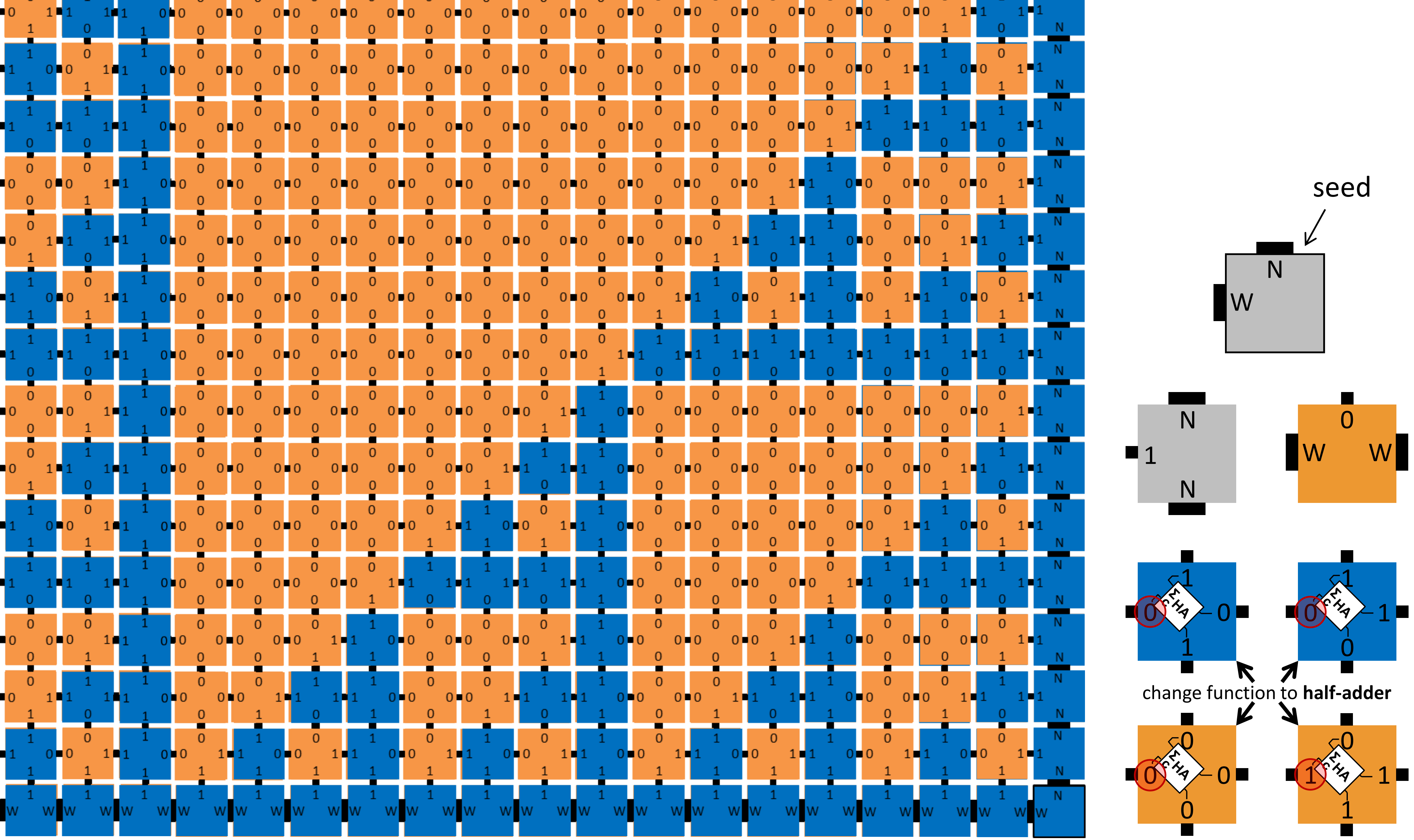
“cooperative binding”

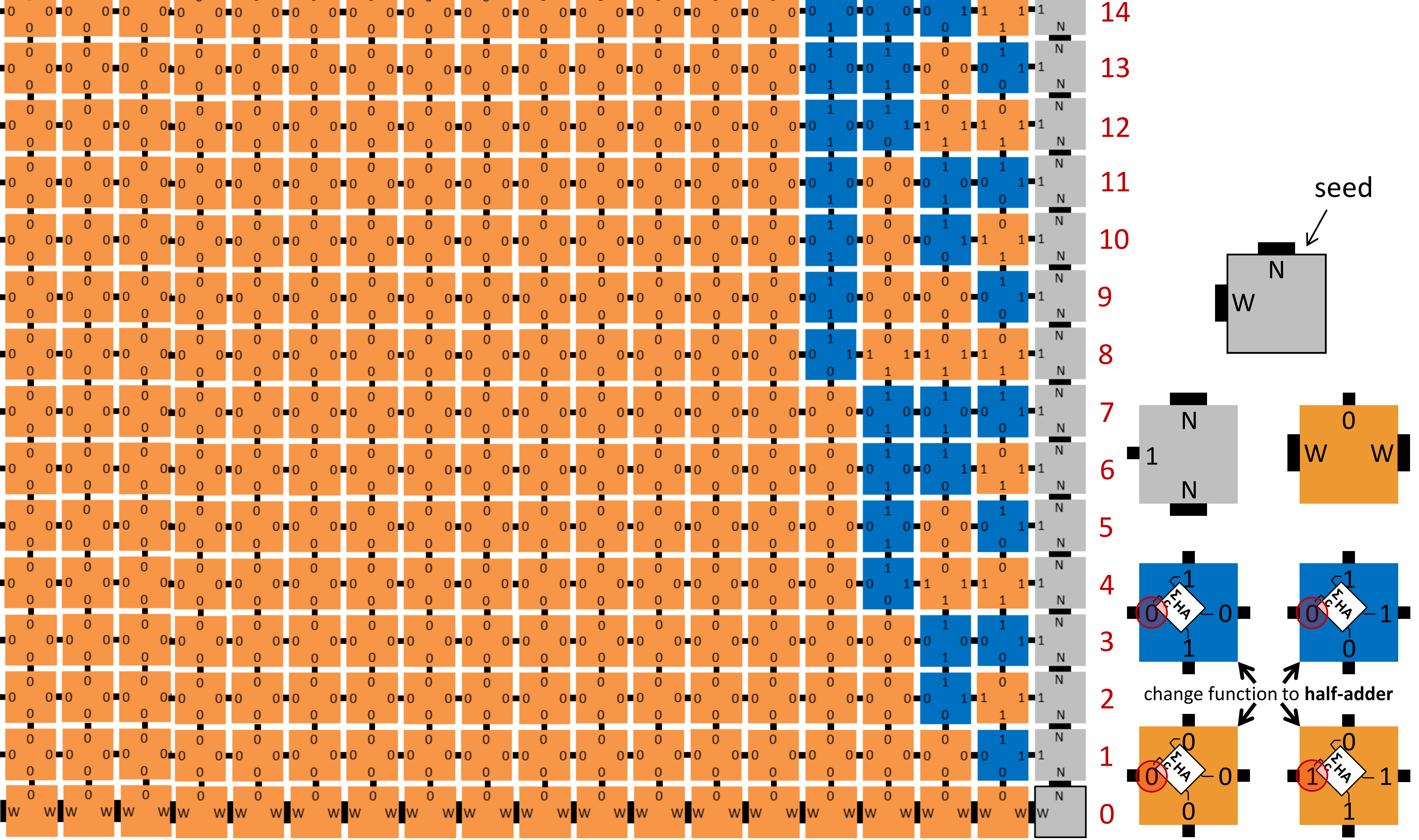


Example tile set



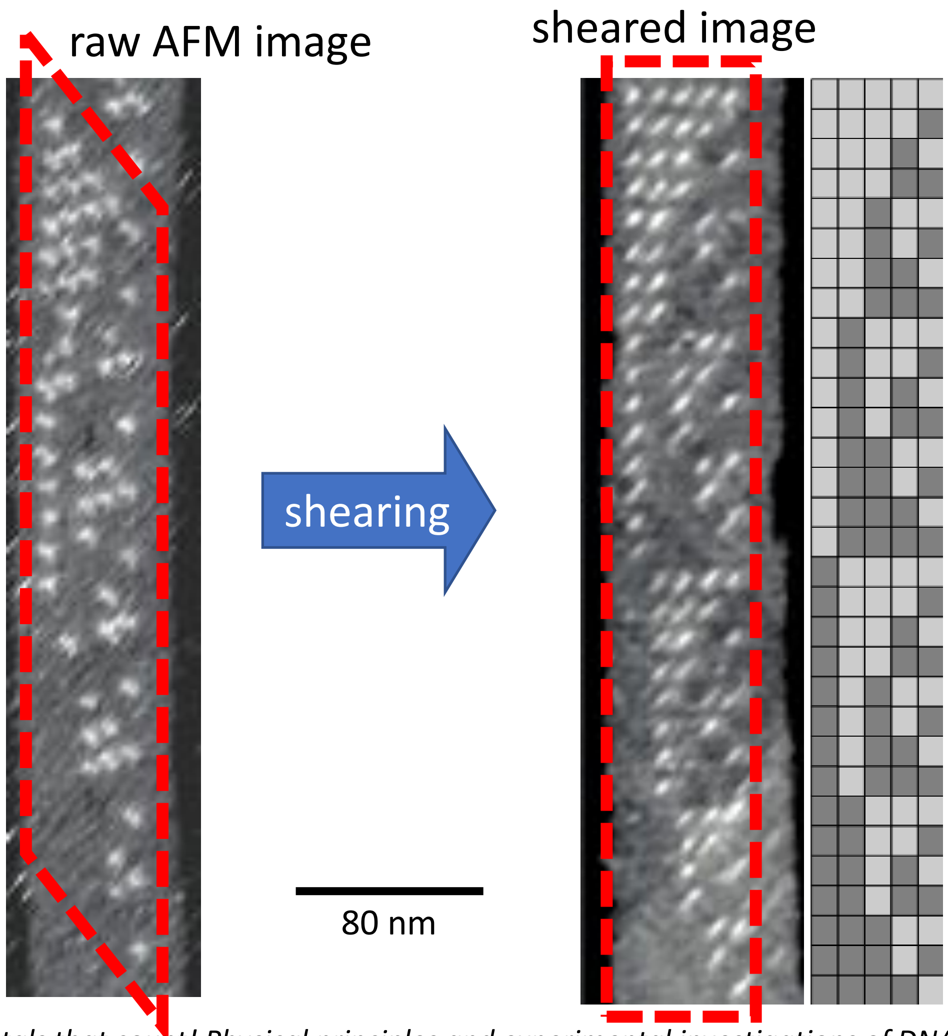

seed


seed
N
W
1
N
0
W
W
change function to half-adder



14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
seed
change function to **half-adder**

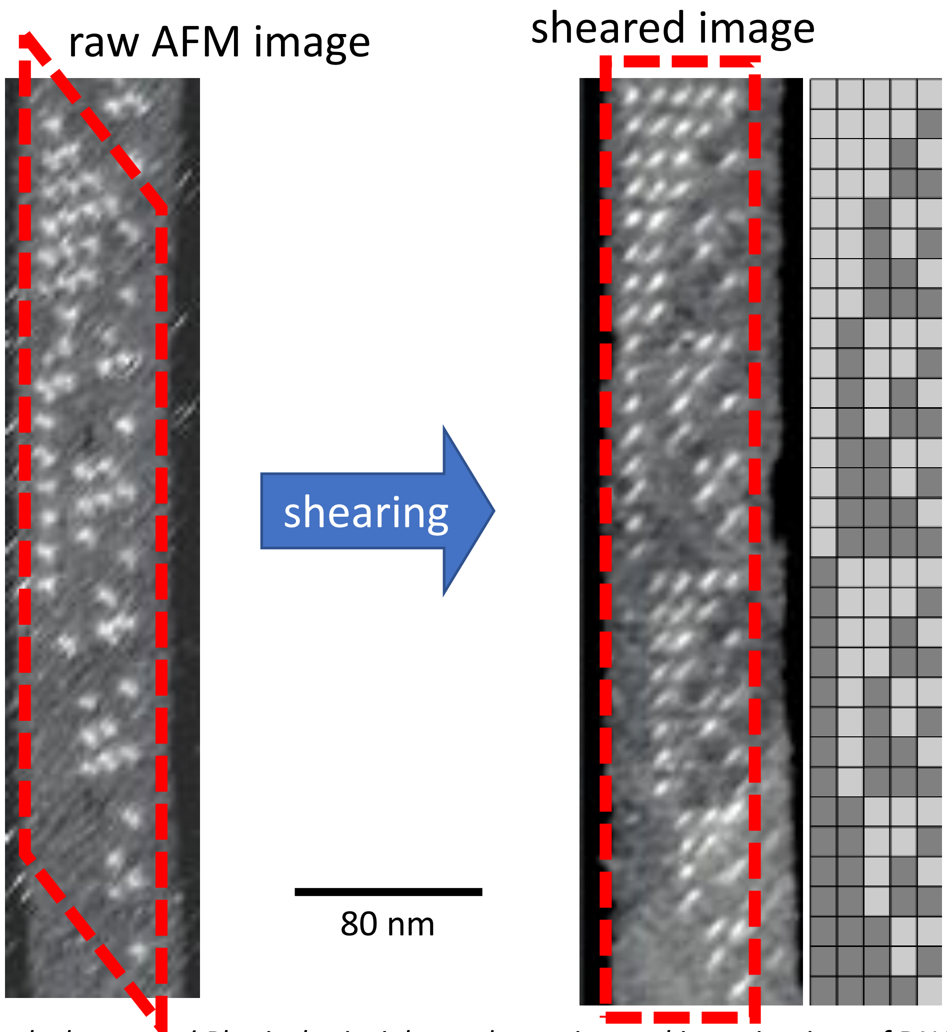
Algorithmic self-assembly in action



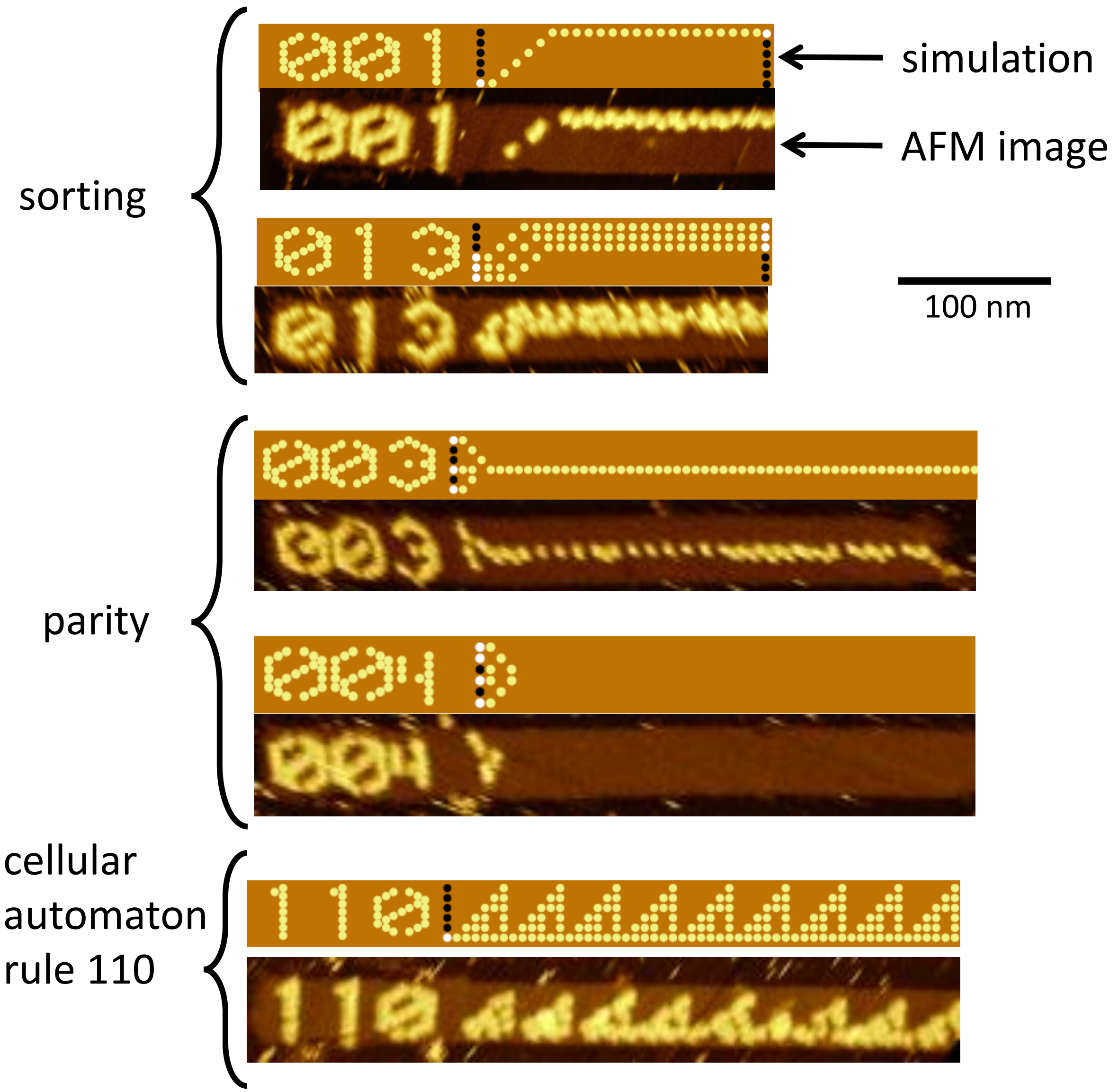
[*Crystals that count! Physical principles and experimental investigations of DNA tile self-assembly*, Constantine Evans, Ph.D. thesis, Caltech, 2014]

Algorithmic self-assembly in action

Track B talk by Damien Woods at 11:30am tomorrow!



[*Crystals that count! Physical principles and experimental investigations of DNA tile self-assembly*, Constantine Evans, Ph.D. thesis, Caltech, 2014]

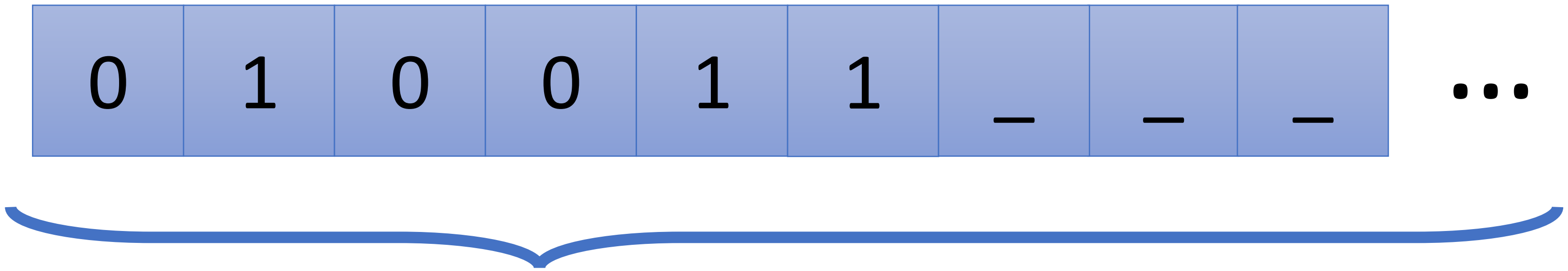


[*Iterated Boolean circuit computation via a programmable DNA tile array*. Woods, Doty, Myhrvold, Hui, Wu, Yin, Winfree, in preparation]

How computationally powerful are self-assembling tiles?

Turing machines

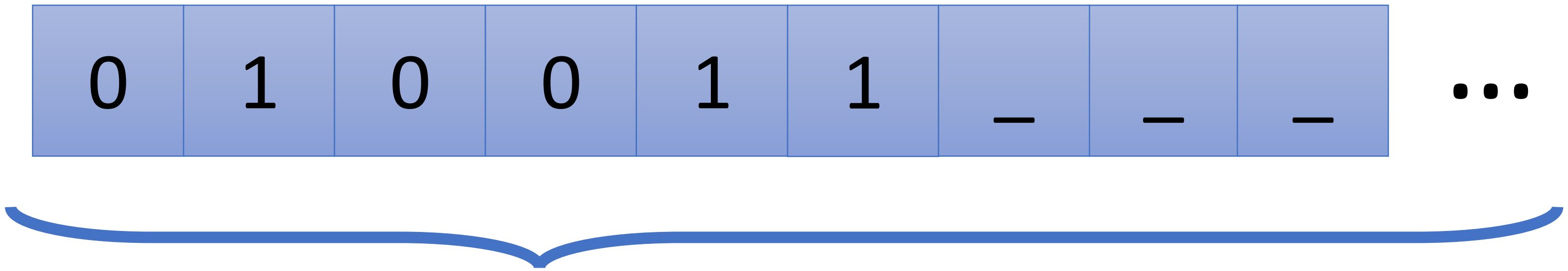
Turing machines



tape ≈ memory

Turing machines

state ≈ line of code

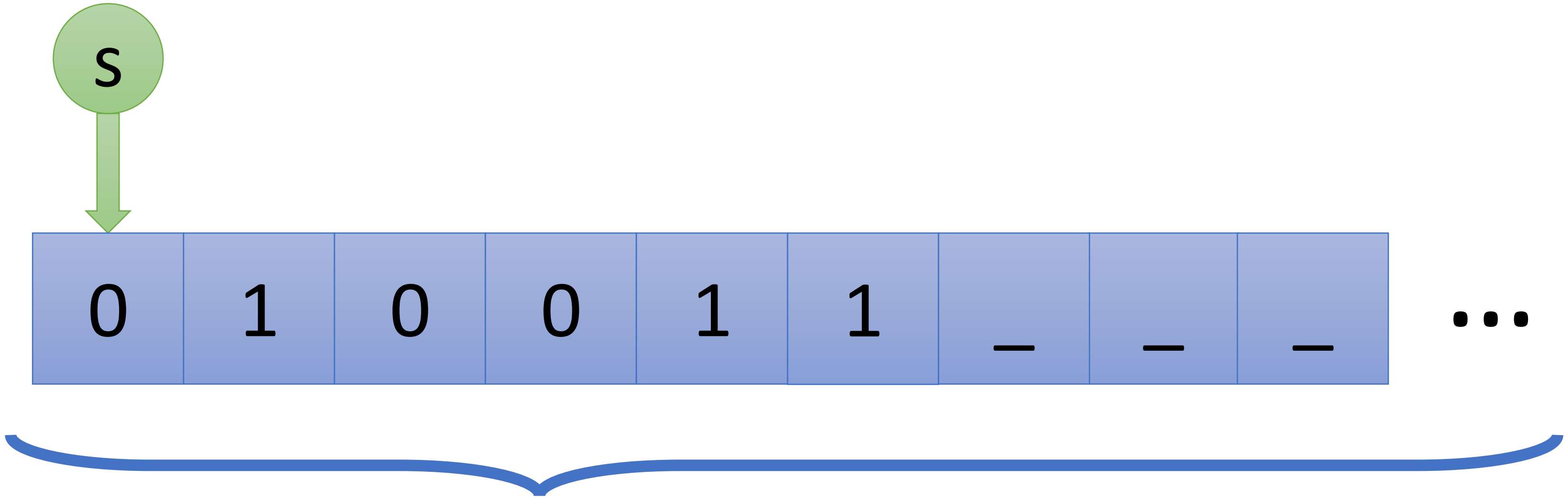


tape ≈ memory

Turing machines

state ≈ line of code

initial state = s



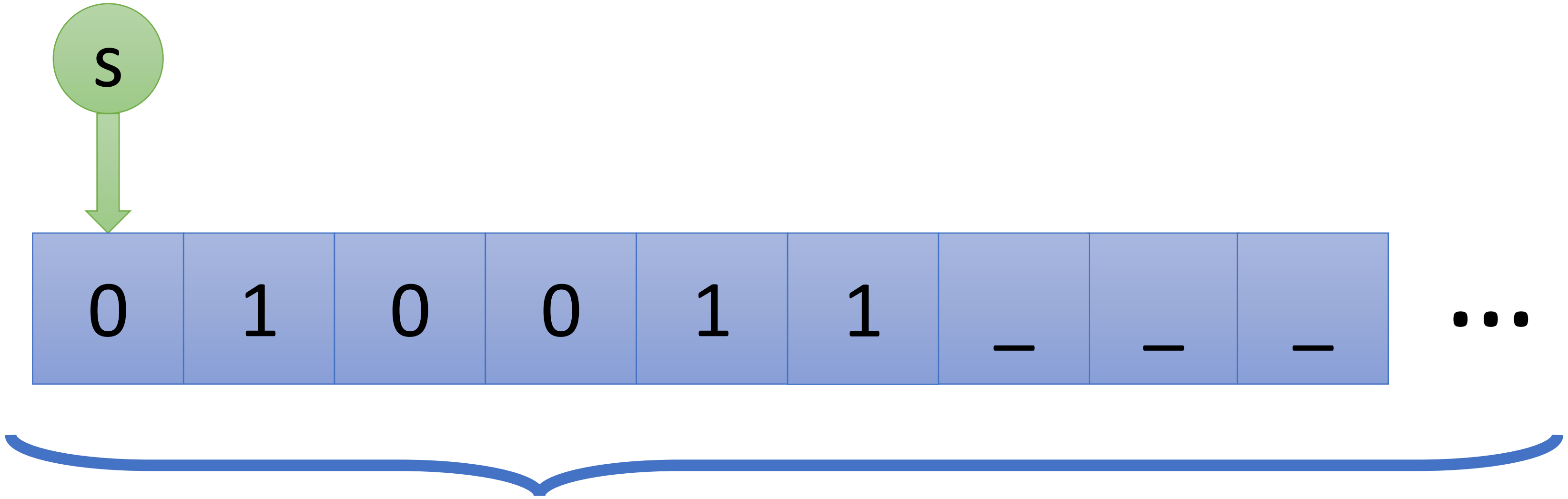
tape ≈ memory

Turing machines

state ≈ line of code

initial state = s

s,0:   q,0,→



transitions
(instructions)

tape ≈ memory

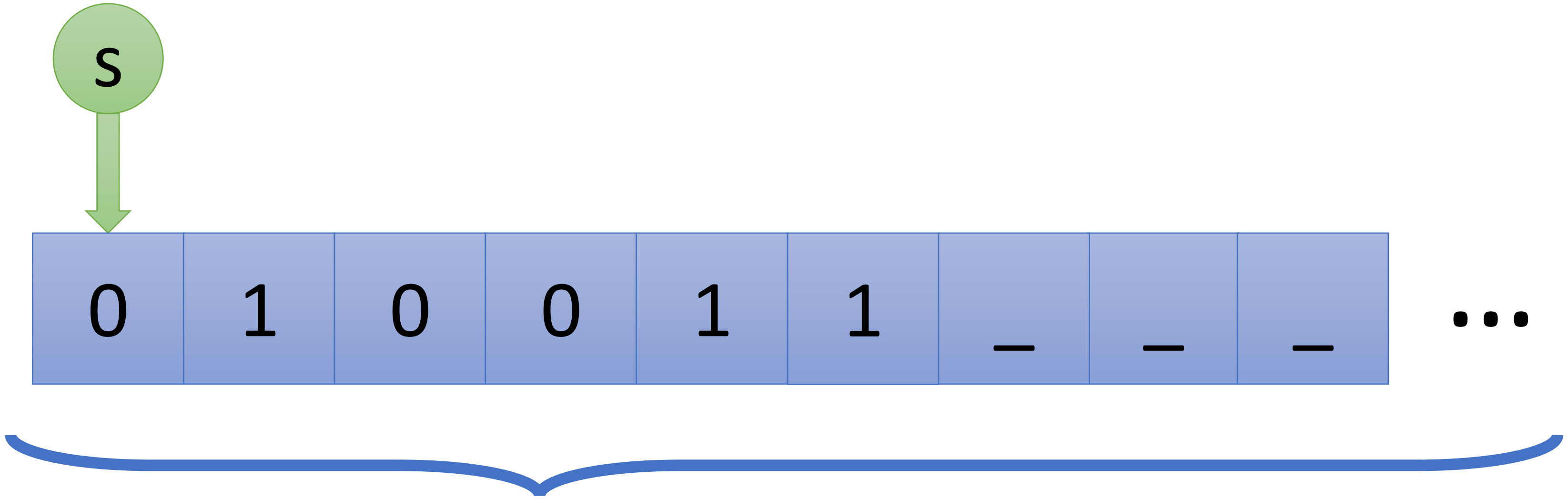
Turing machines

state ≈ line of code

initial state = s

current state

s,0: q,0,→



transitions
(instructions)

tape ≈ memory

Turing machines

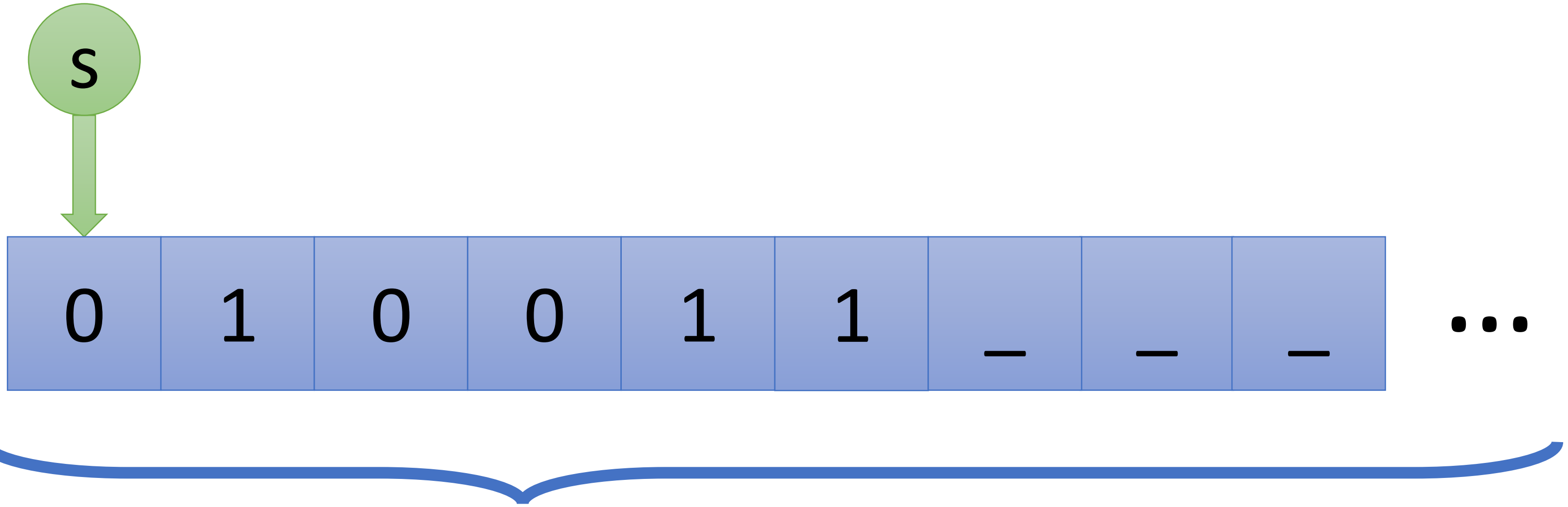
state ≈ line of code

initial state = s

current symbol

current state

s,0: q,0,→



transitions

(instructions)

tape ≈ memory

Turing machines

state ≈ line of code

initial state = s

current symbol

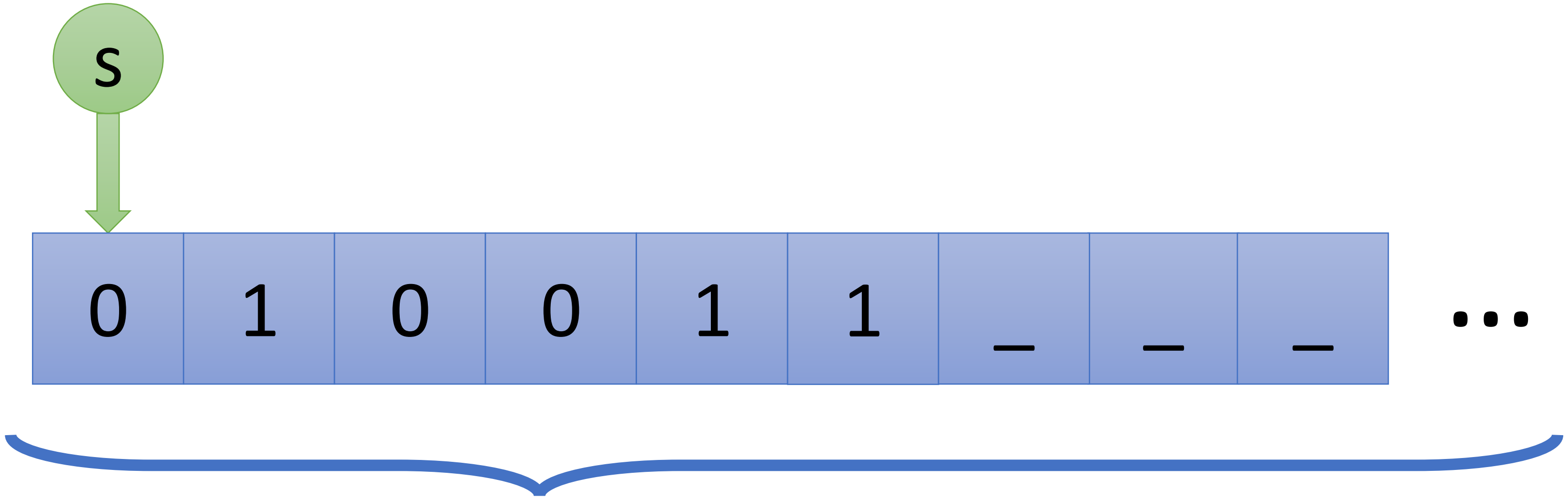
current state

next state

s,0: q,0,→

transitions

(instructions)

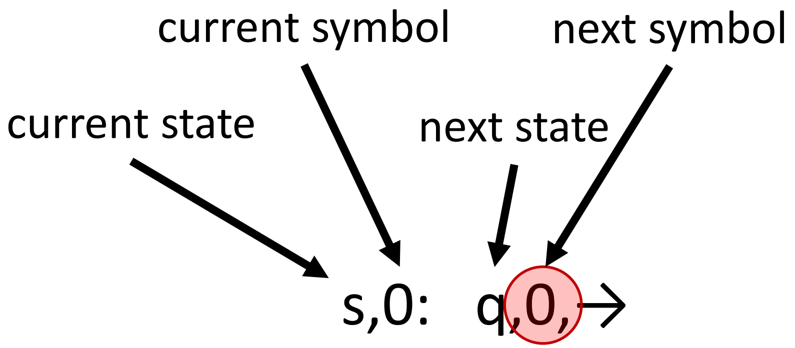


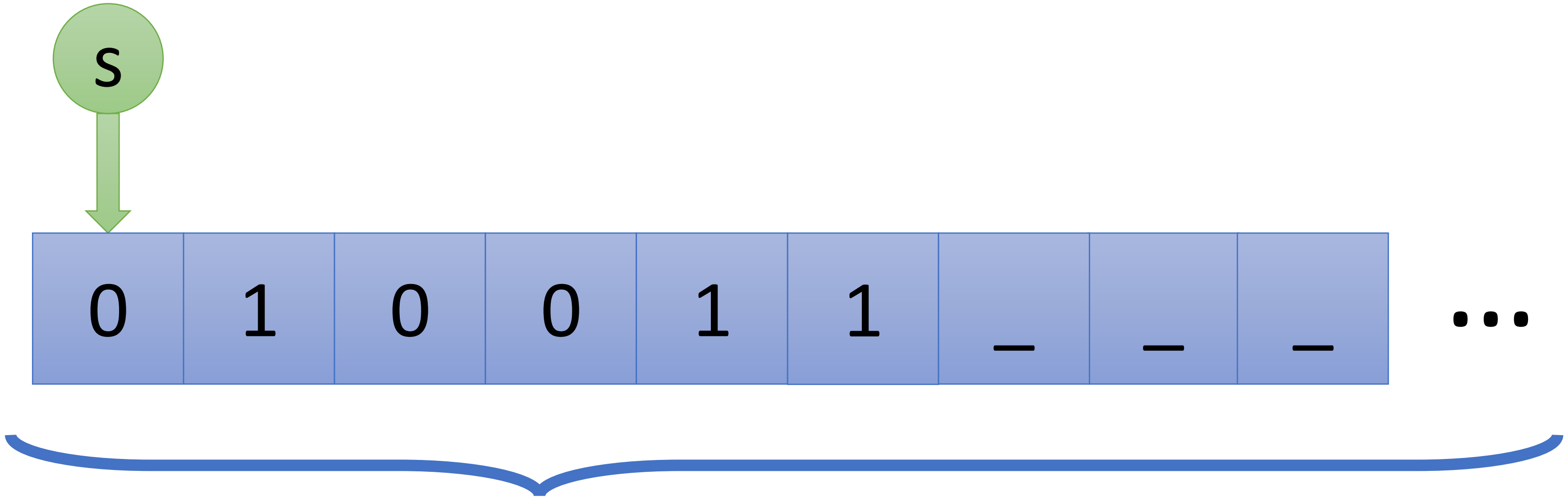
tape ≈ memory

Turing machines

state ≈ line of code

initial state = s





transitions
(instructions)

tape ≈ memory

Turing machines

state ≈ line of code

initial state = s

current symbol

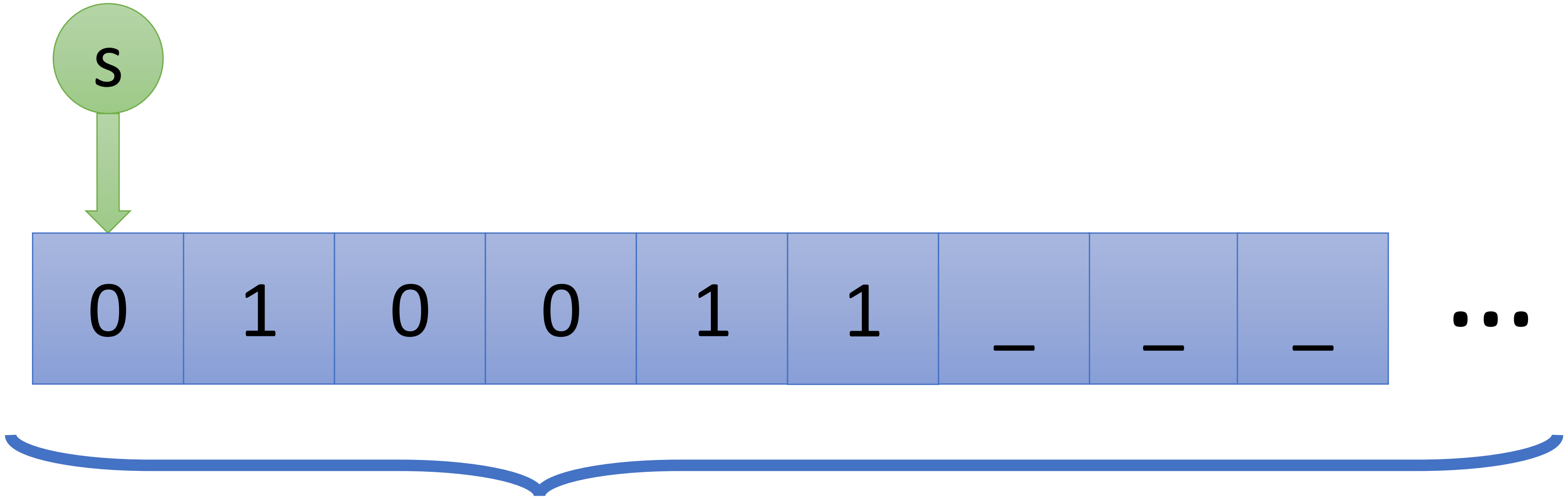
next symbol

current state

next state

next move

s,0: q,0,→



transitions

(instructions)

tape ≈ memory

Turing machines

state ≈ line of code

initial state = s

current symbol   next symbol

current state   next state   next move

s,0:   q,0,→

q,0:   t,1,←

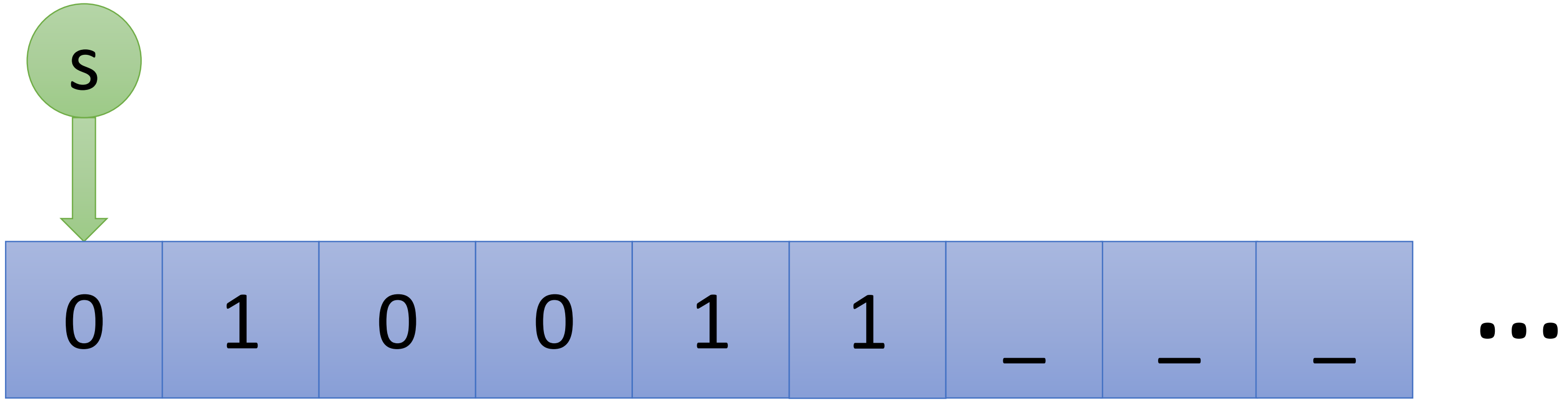
q,1:   s,0,→

t,0:   u,1,→

u,1:   HALT

transitions

(instructions)



tape ≈ memory

Turing machines

current symbol
next symbol
current state
next state
next move

s,0: q,0,→

q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

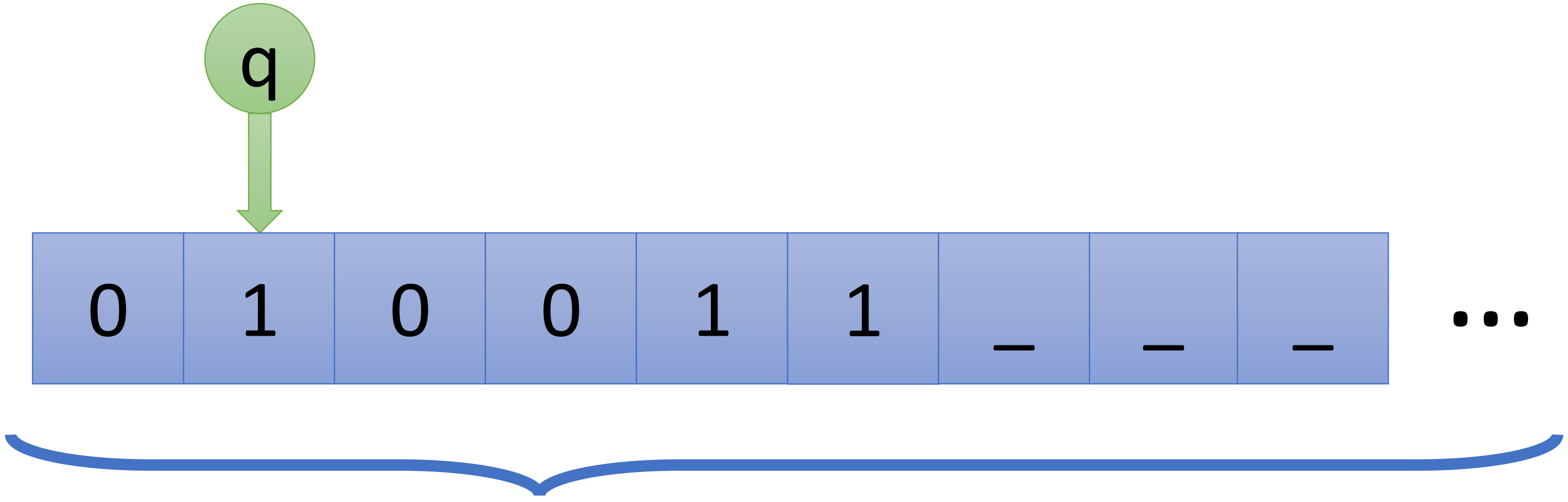
u,1: HALT

transitions

(instructions)

state ≈ line of code

initial state = s



tape ≈ memory

Turing machines

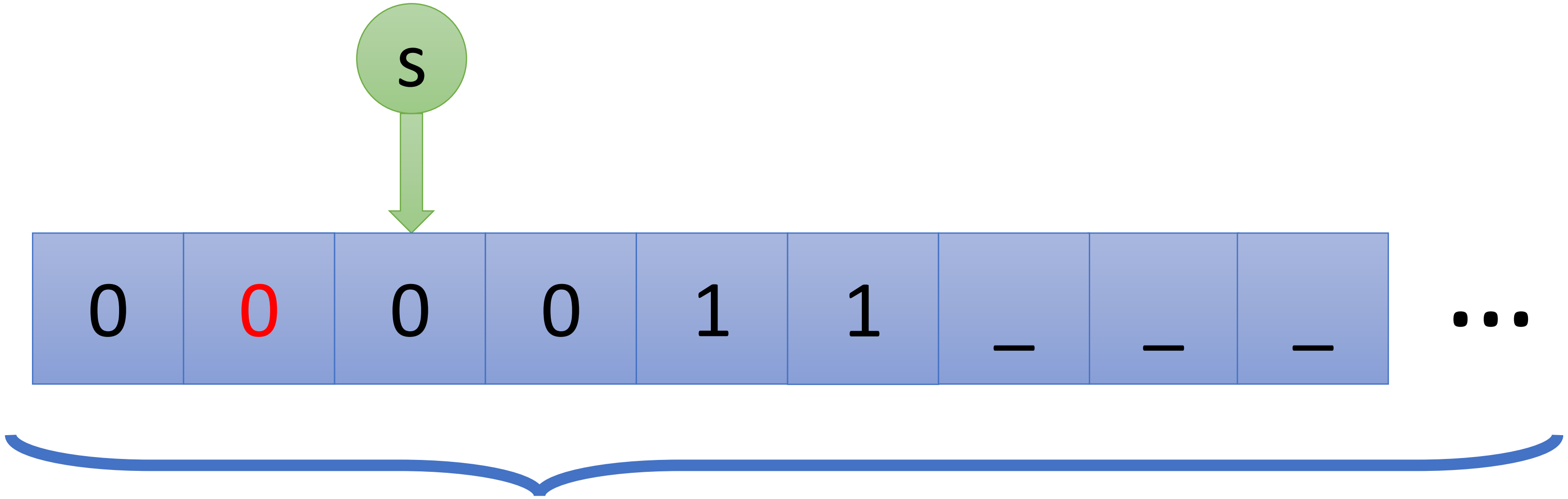
state ≈ line of code

initial state = s

current symbol
next symbol
current state
next state
next move

s,0: q,0,→
q,0: t,1,←
q,1: s,0,→
t,0: u,1,→
u,1: HALT

transitions
(instructions)



tape ≈ memory

Turing machines

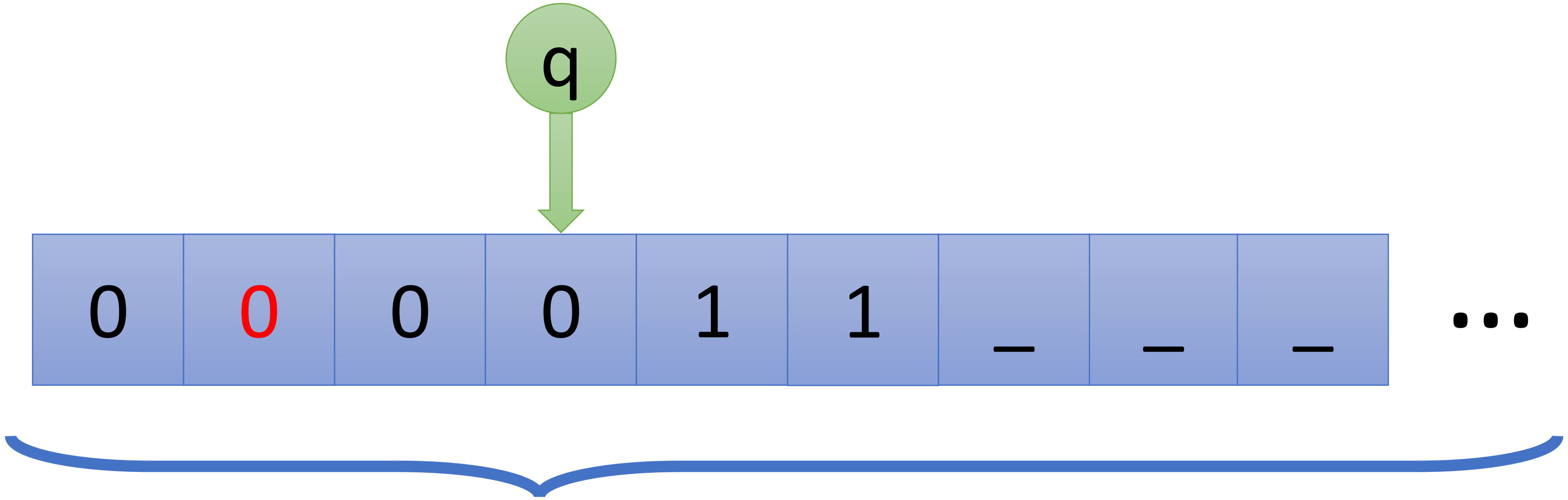
state ≈ line of code

initial state = s

current state, current symbol, next state, next symbol, next move

s,0: q,0,→
q,0: t,1,←
q,1: s,0,→
t,0: u,1,→
u,1: HALT

transitions
(instructions)



tape ≈ memory

Turing machines

current symbol — next symbol

current state — next state — next move

s,0: q,0,→

q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

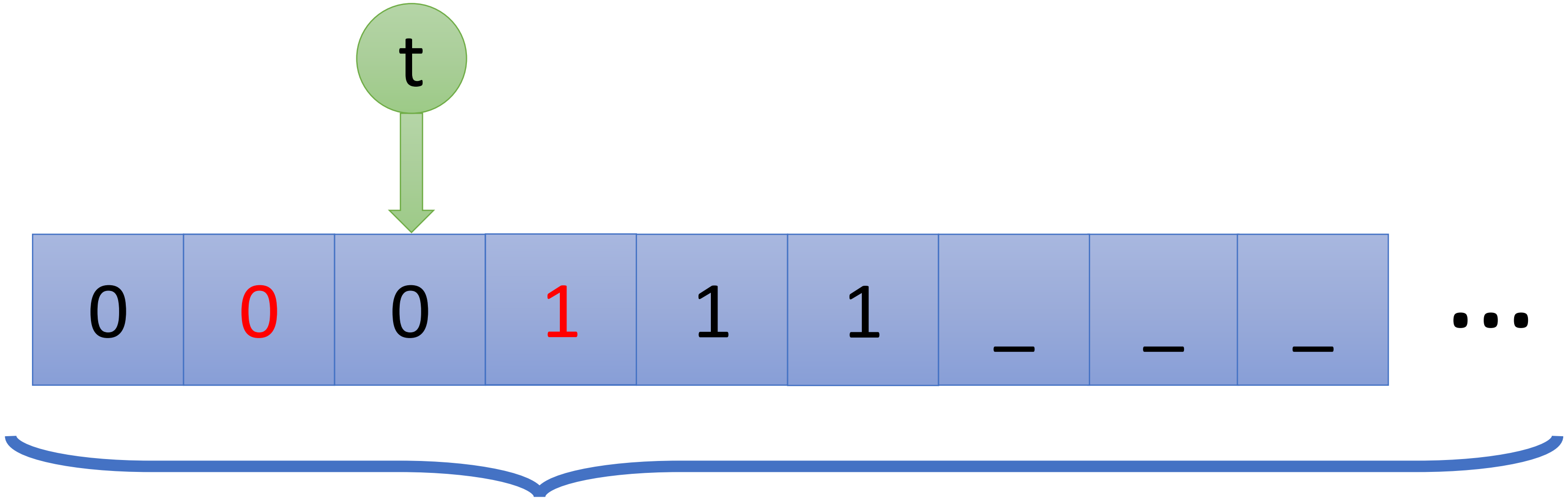
u,1: HALT

transitions

(instructions)

state ≈ line of code

initial state = s

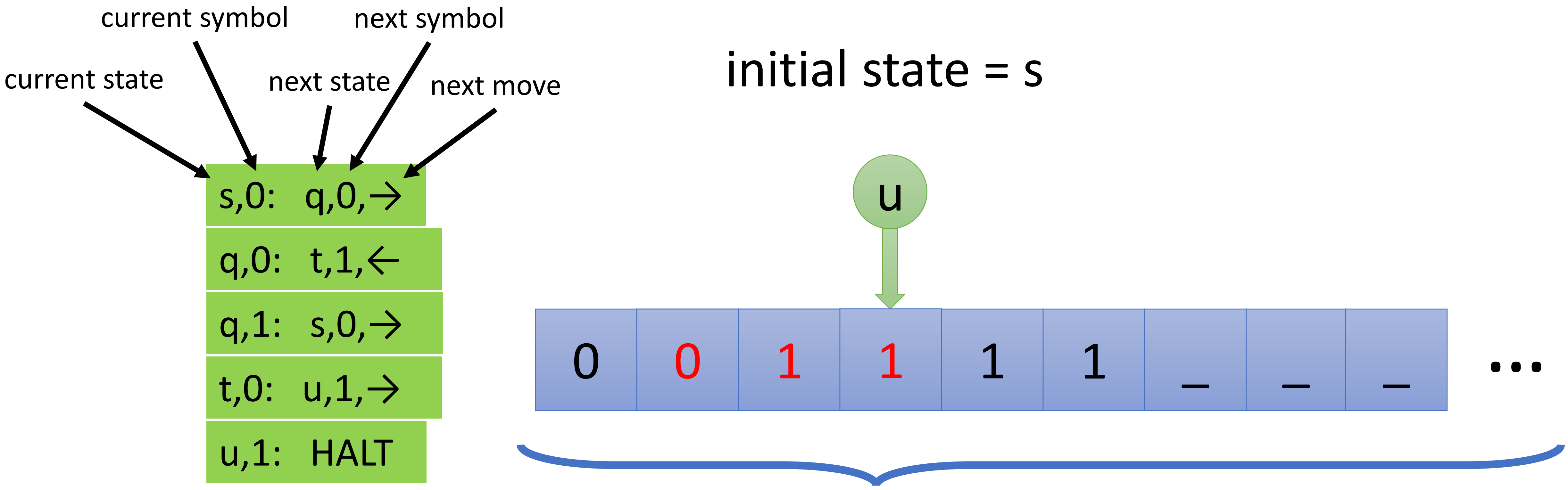


tape ≈ memory

Turing machines

state ≈ line of code

initial state = s



transitions

(instructions)

tape ≈ memory

Tile assembly is Turing-universal

Tile assembly is Turing-universal

s,0: q,0,→

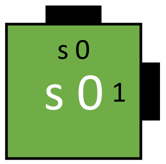
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

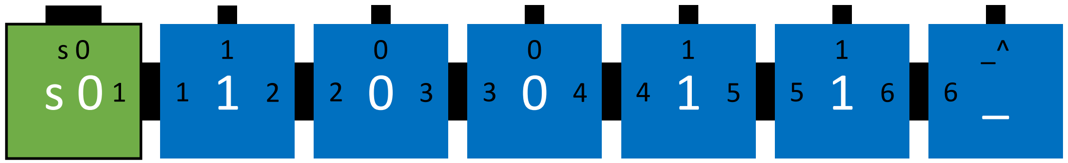
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

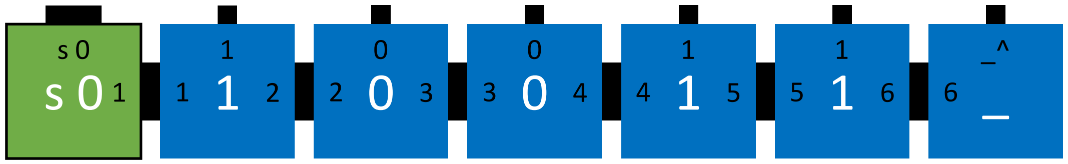
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

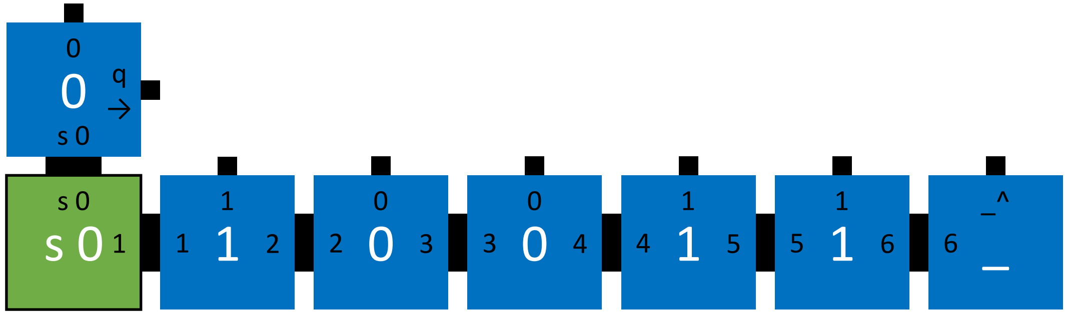
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

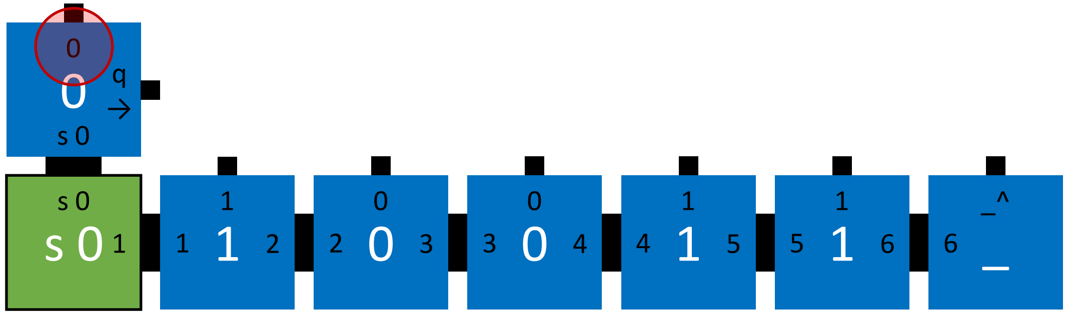
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

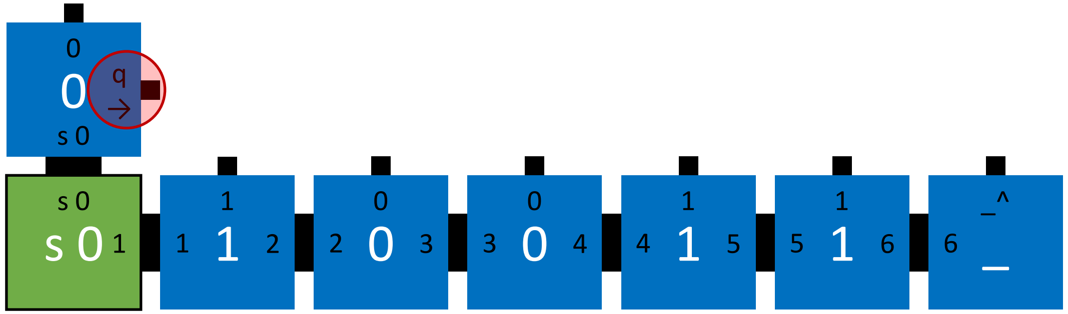
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

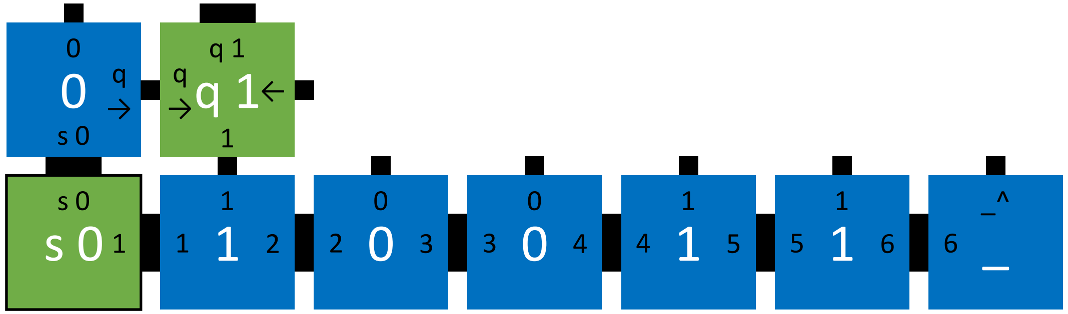
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

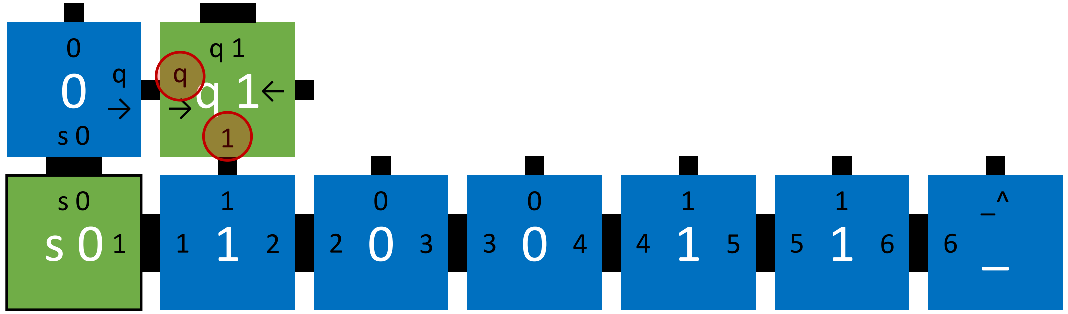
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

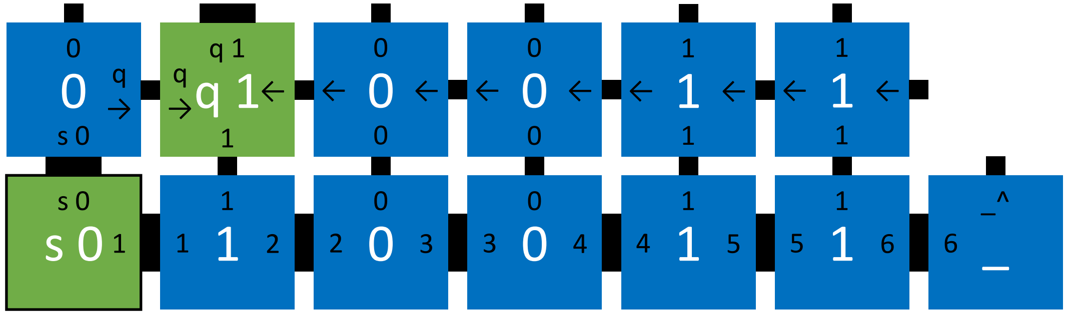
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

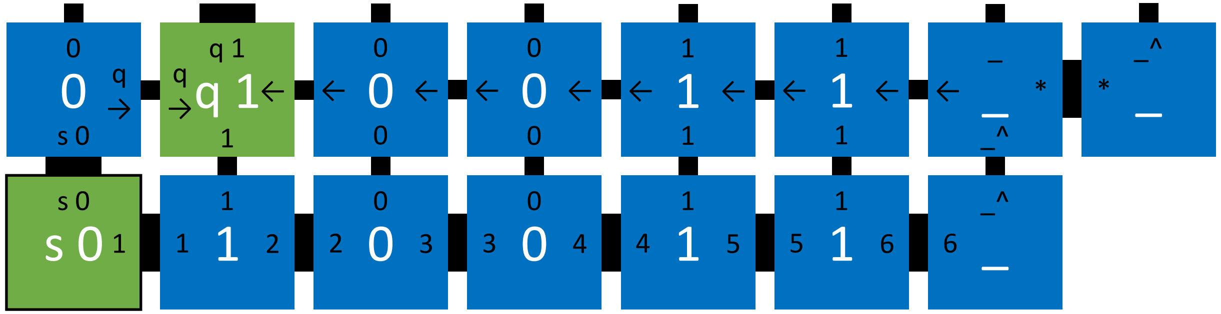
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

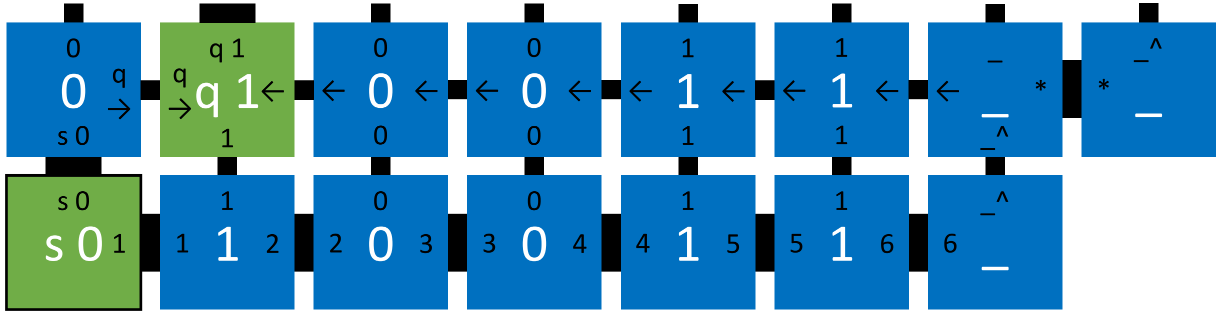
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0:   q,0,→

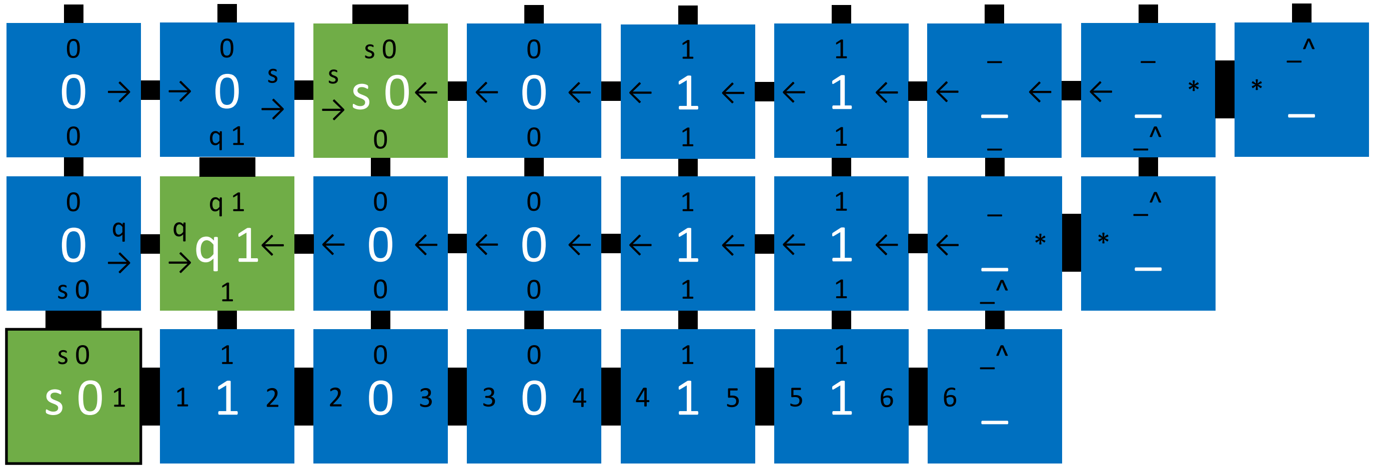
q,0:   t,1,←

q,1:   s,0,→

t,0:   u,1,→

u,1:   HALT

Tile assembly is Turing-universal



s,0: q,0,→

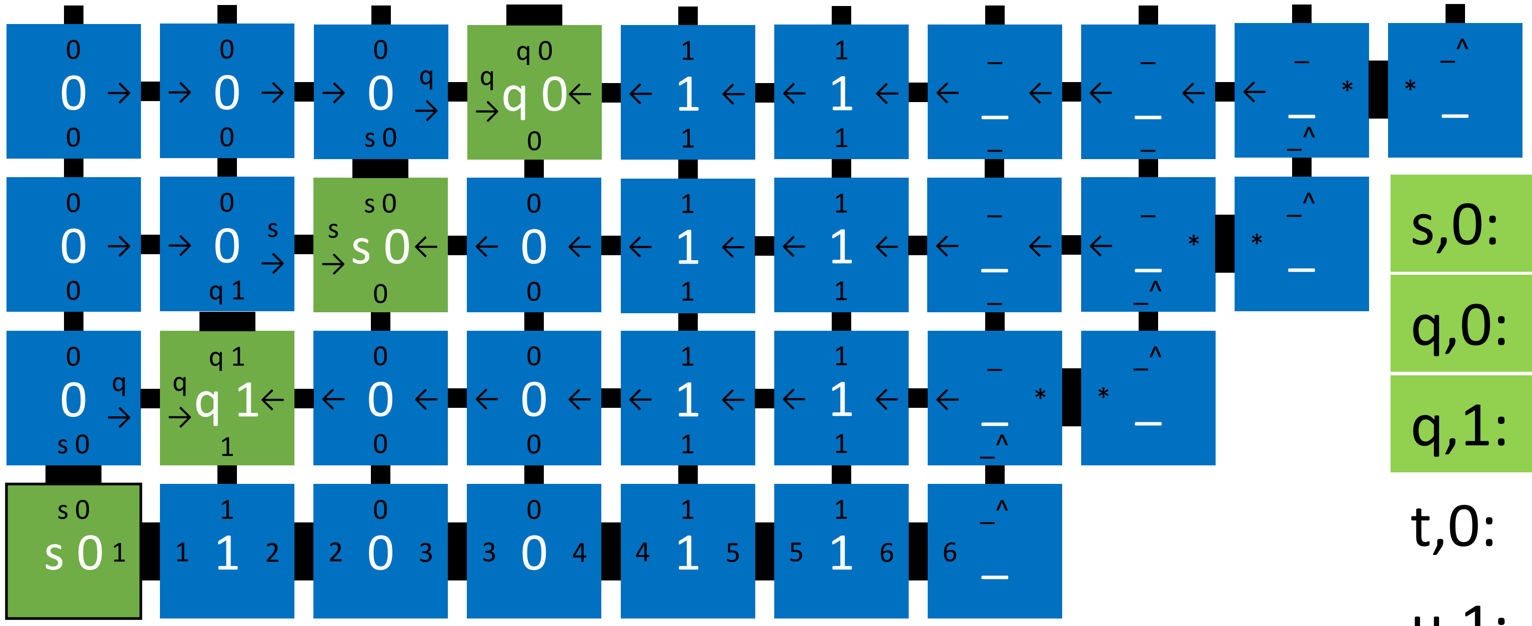
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

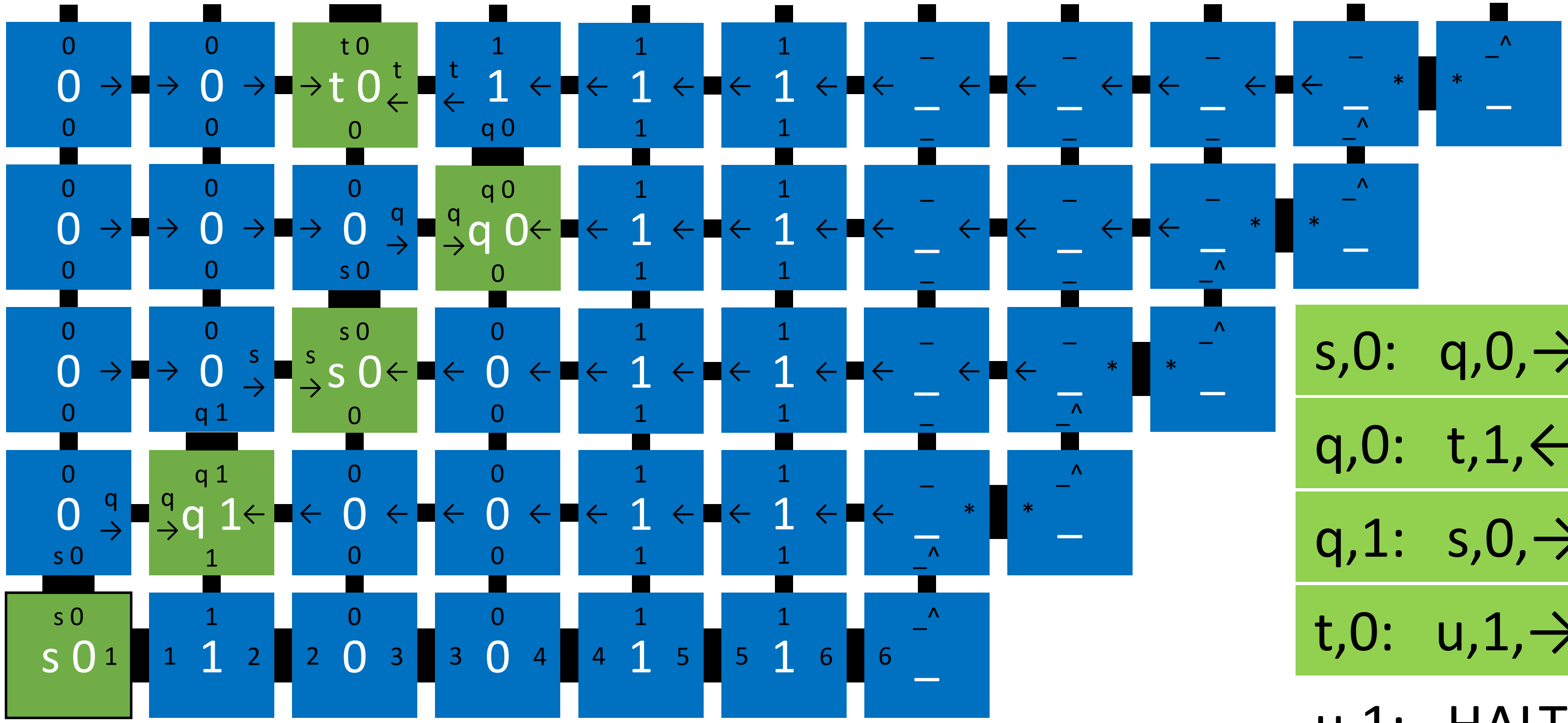
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

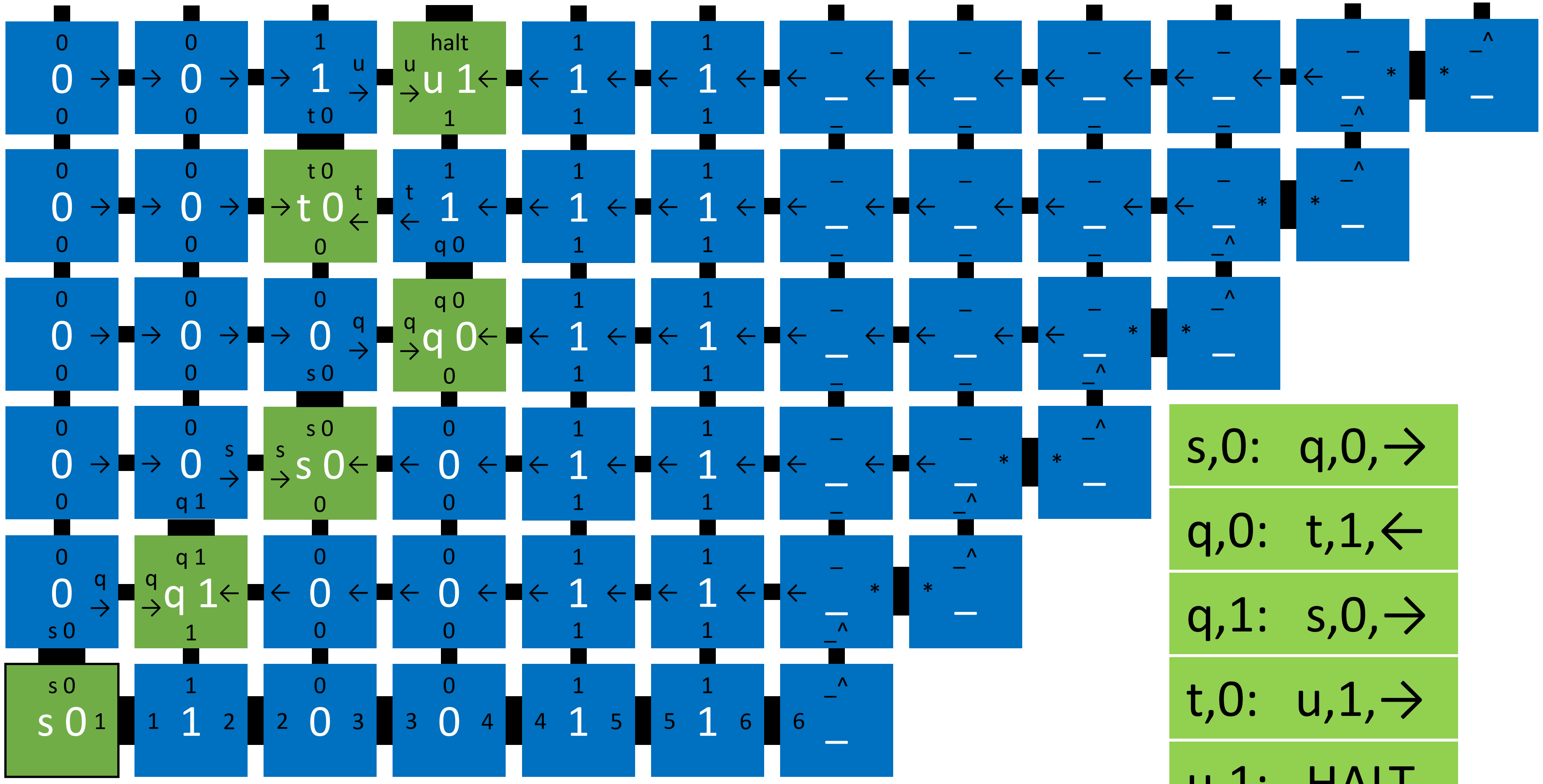
u,1: HALT

Tile assembly is Turing-universal



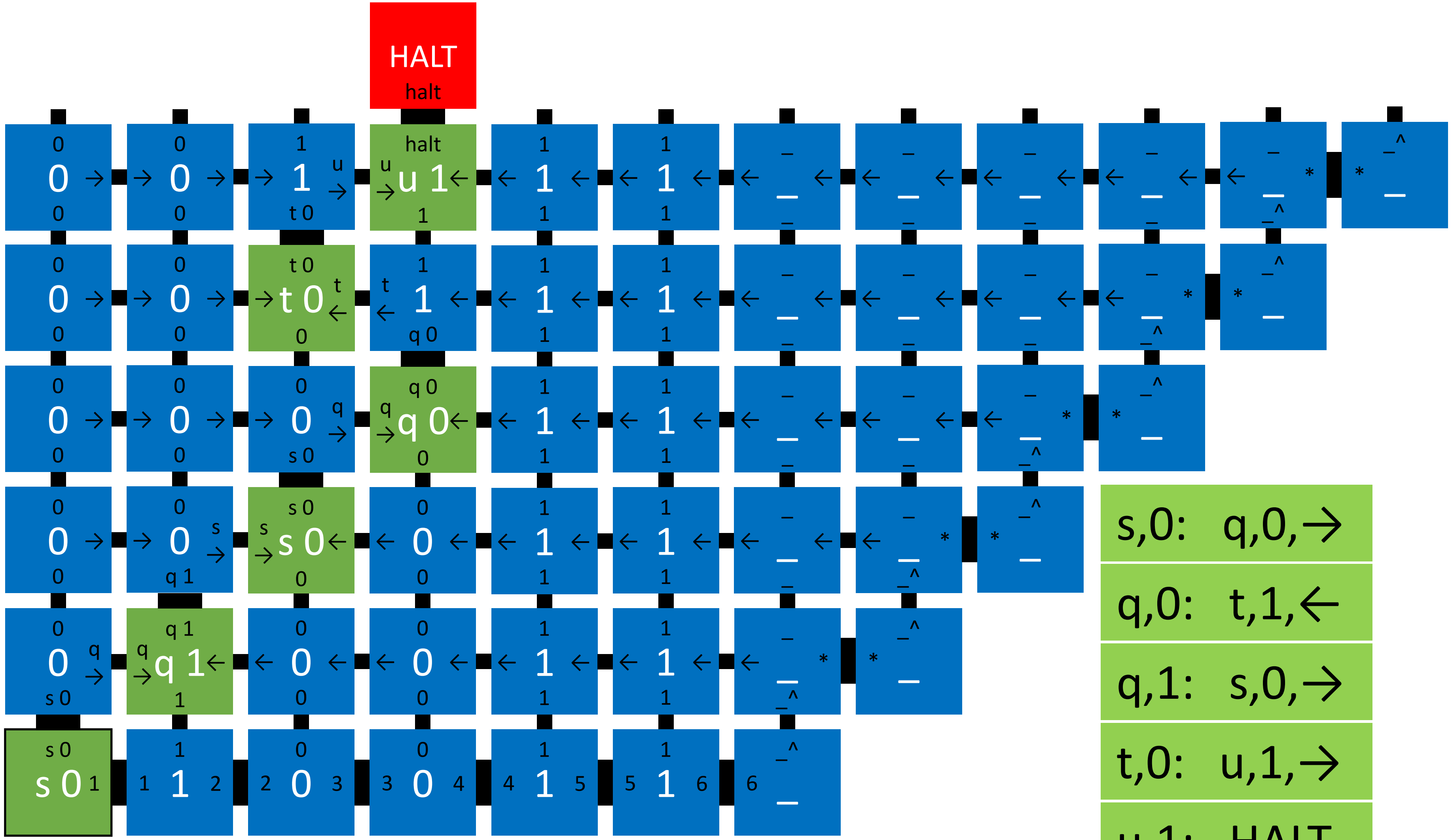
s,0:	q,0,→
q,0:	t,1,←
q,1:	s,0,→
t,0:	u,1,→
u,1:	HALT

Tile assembly is Turing-universal



s,0:	q,0,→
q,0:	t,1,←
q,1:	s,0,→
t,0:	u,1,→
u,1:	HALT

Tile assembly is Turing-universal



s,0: q,0,→

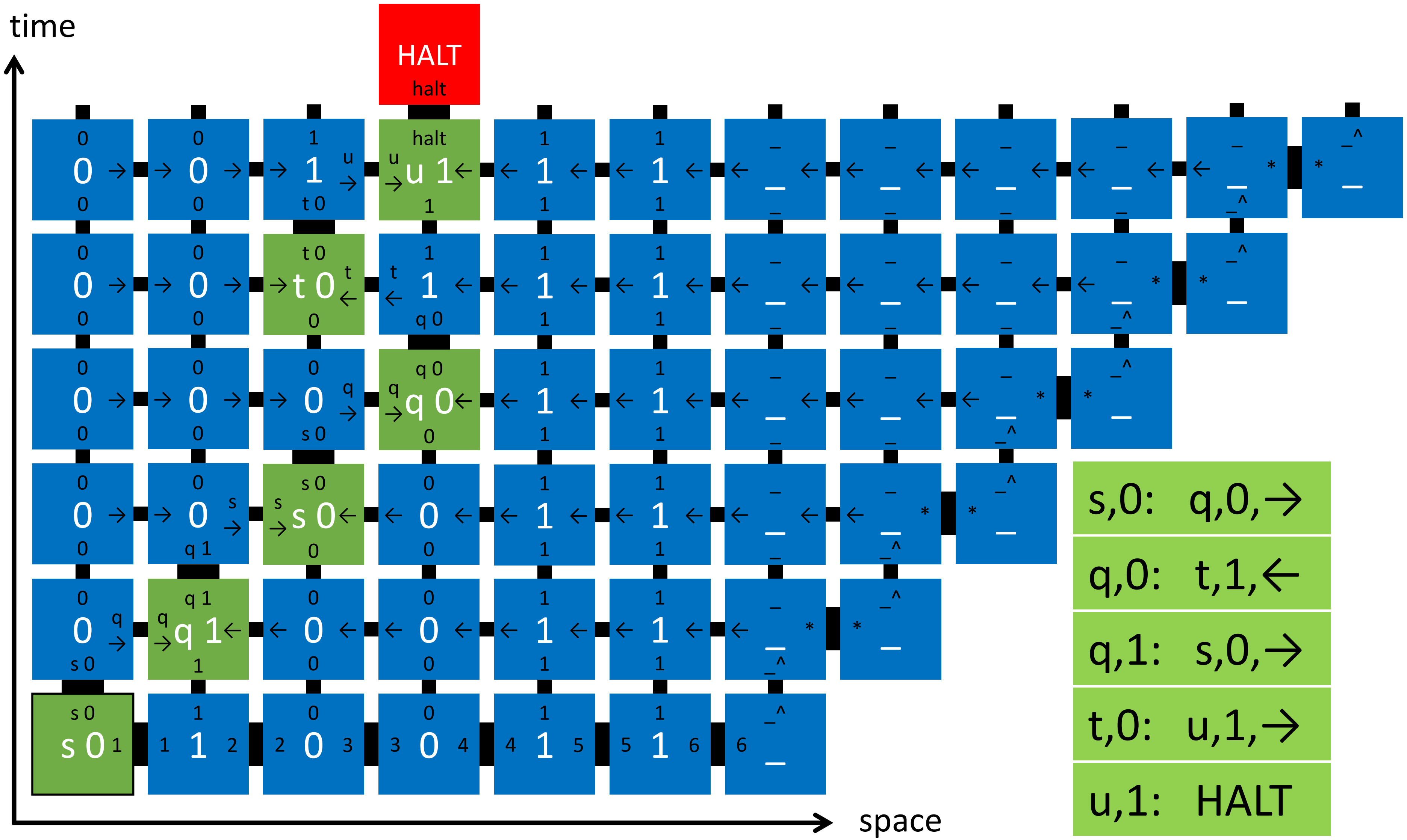
q,0: t,1,←

q,1: s,0,→

t,0: u,1,→

u,1: HALT

Tile assembly is Turing-universal



s,0: q,0,→

q,0: t,1,←

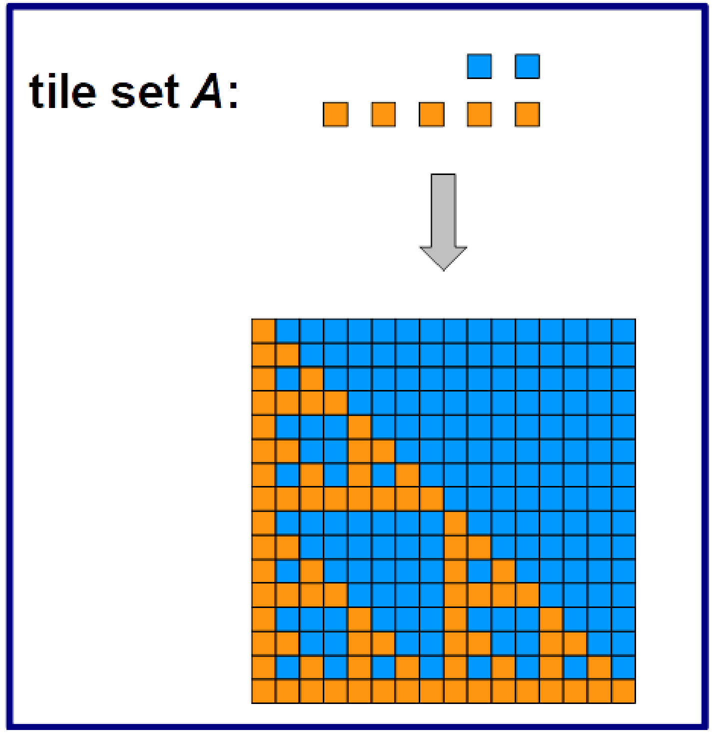
q,1: s,0,→

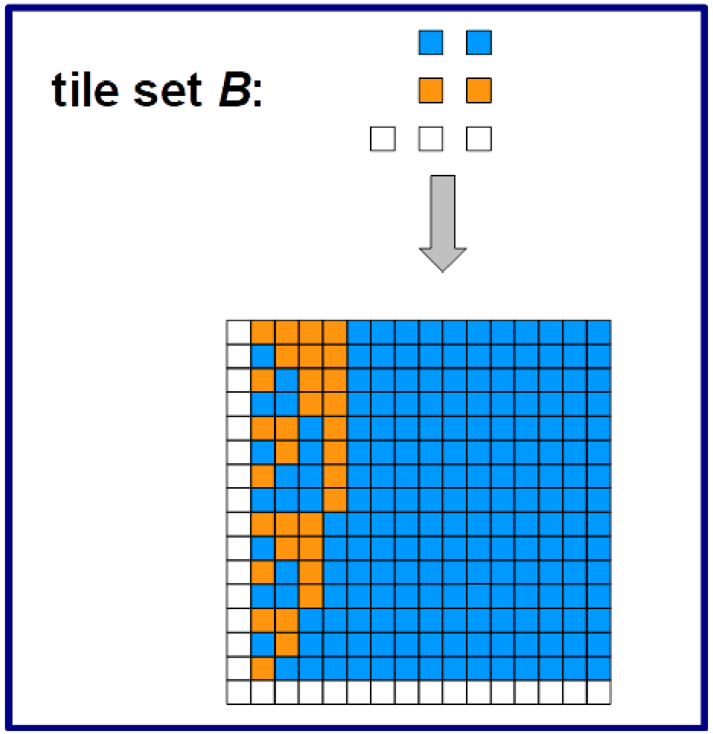
t,0: u,1,→

u,1: HALT

Putting the *algorithm* in *algorithmic* self-assembly

- set of tile types is like a program
- shape it creates, or pattern it paints, is like the output of the program





Putting the *algorithm* in *algorithmic* self-assembly

How is a set of tile types **not** like a program?

Putting the *algorithm* in *algorithmic* self-assembly

How is a set of tile types **not** like a program?

- Where's the input to the program?

Putting the *algorithm* in *algorithmic* self-assembly

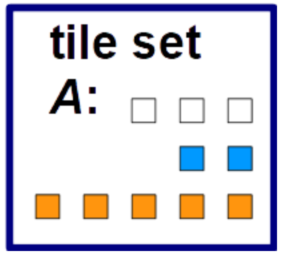
How is a set of tile types **not** like a program?

- Where's the input to the program?
- One perspective: pre-assembled seed encodes the input

Putting the *algorithm* in *algorithmic* self-assembly

How is a set of tile types **not** like a program?

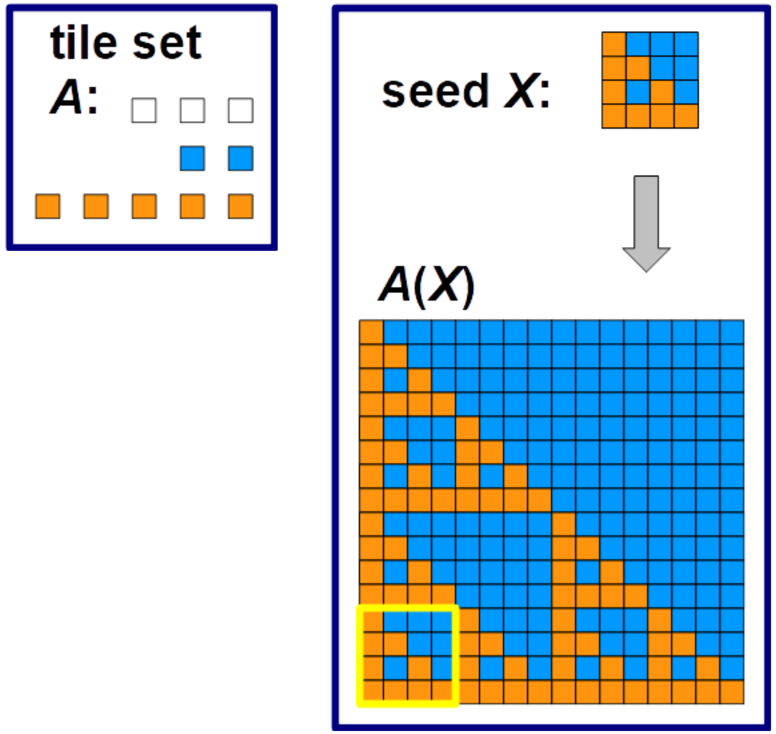
- Where's the input to the program?
- One perspective: pre-assembled seed encodes the input



Putting the *algorithm* in *algorithmic* self-assembly

How is a set of tile types **not** like a program?

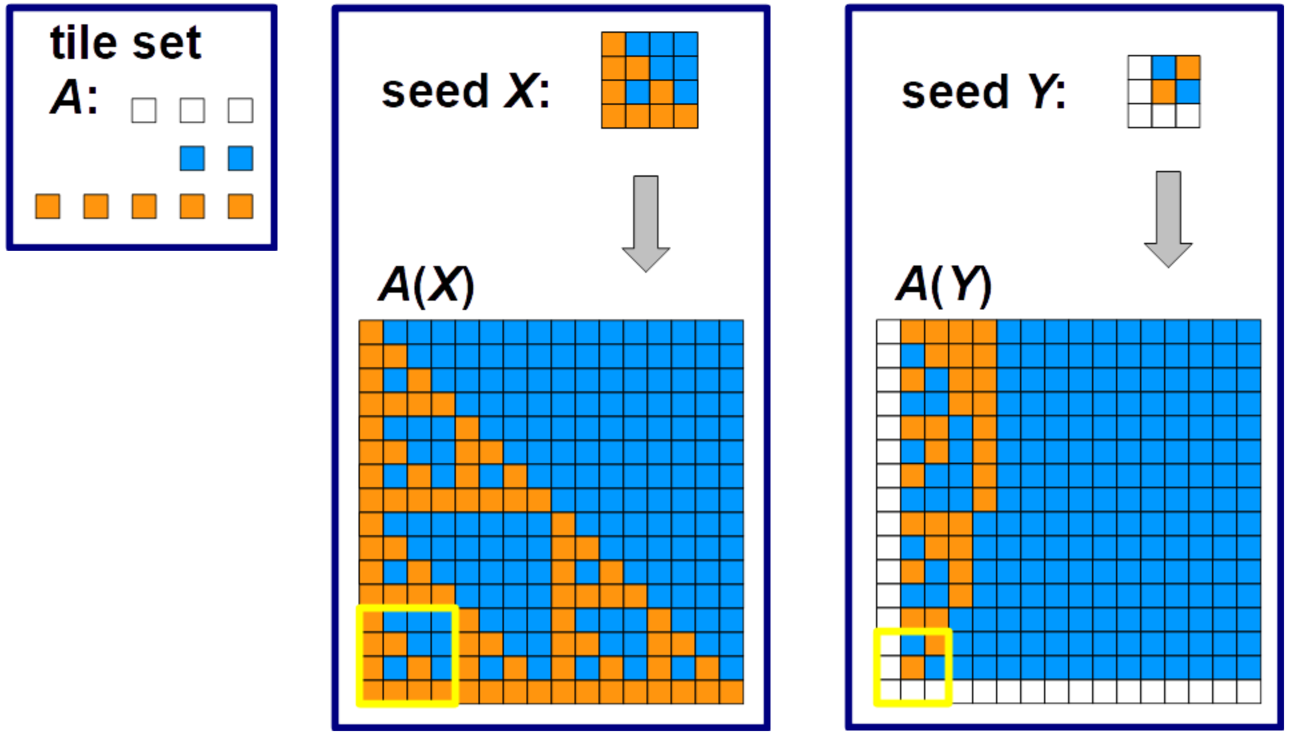
- Where's the input to the program?
- One perspective: pre-assembled seed encodes the input



Putting the *algorithm* in *algorithmic* self-assembly

How is a set of tile types **not** like a program?

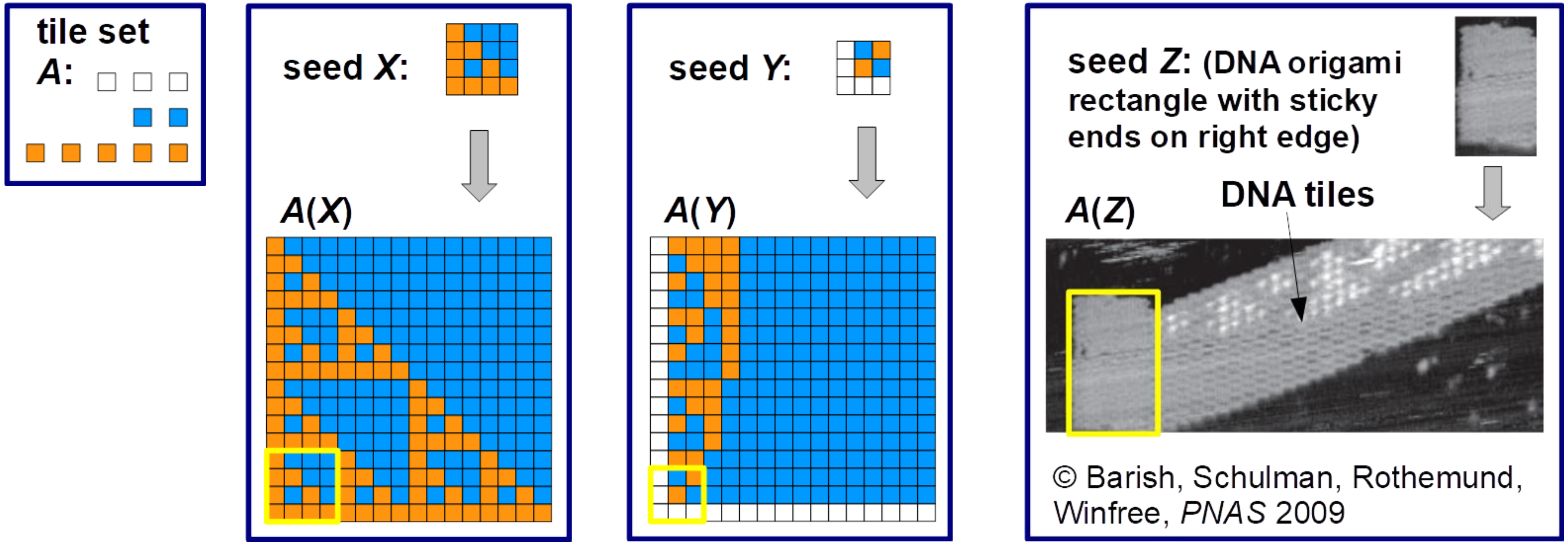
- Where's the input to the program?
- One perspective: pre-assembled seed encodes the input



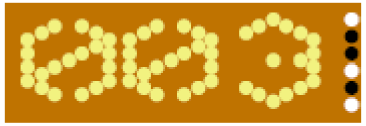
Putting the *algorithm* in *algorithmic* self-assembly

How is a set of tile types **not** like a program?

- Where's the input to the program?
- One perspective: pre-assembled seed encodes the input

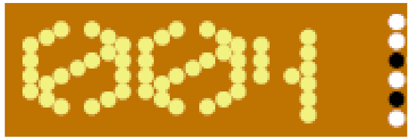


Calculating parity of 6-bit string: 1 algorithm, 2^6 inputs

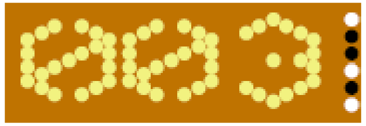


seed encoding **100101**

seed encoding **110101**

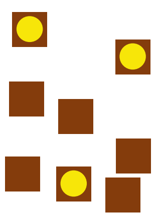


Calculating parity of 6-bit string: 1 algorithm, 2^6 inputs

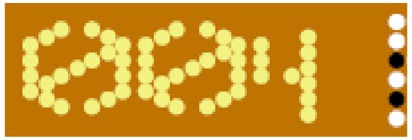


seed encoding **100101**

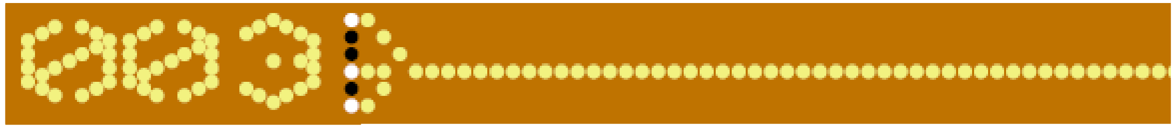
single set of tiles computing parity



seed encoding **110101**

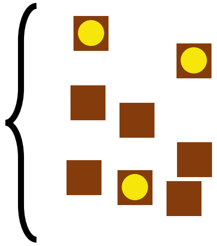


Calculating parity of 6-bit string:
1 algorithm, 2^6 inputs



seed encoding **100101**

single set of tiles computing parity

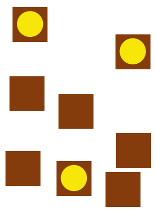


seed encoding **110101**

Calculating parity of 6-bit string: 1 algorithm, 2^6 inputs

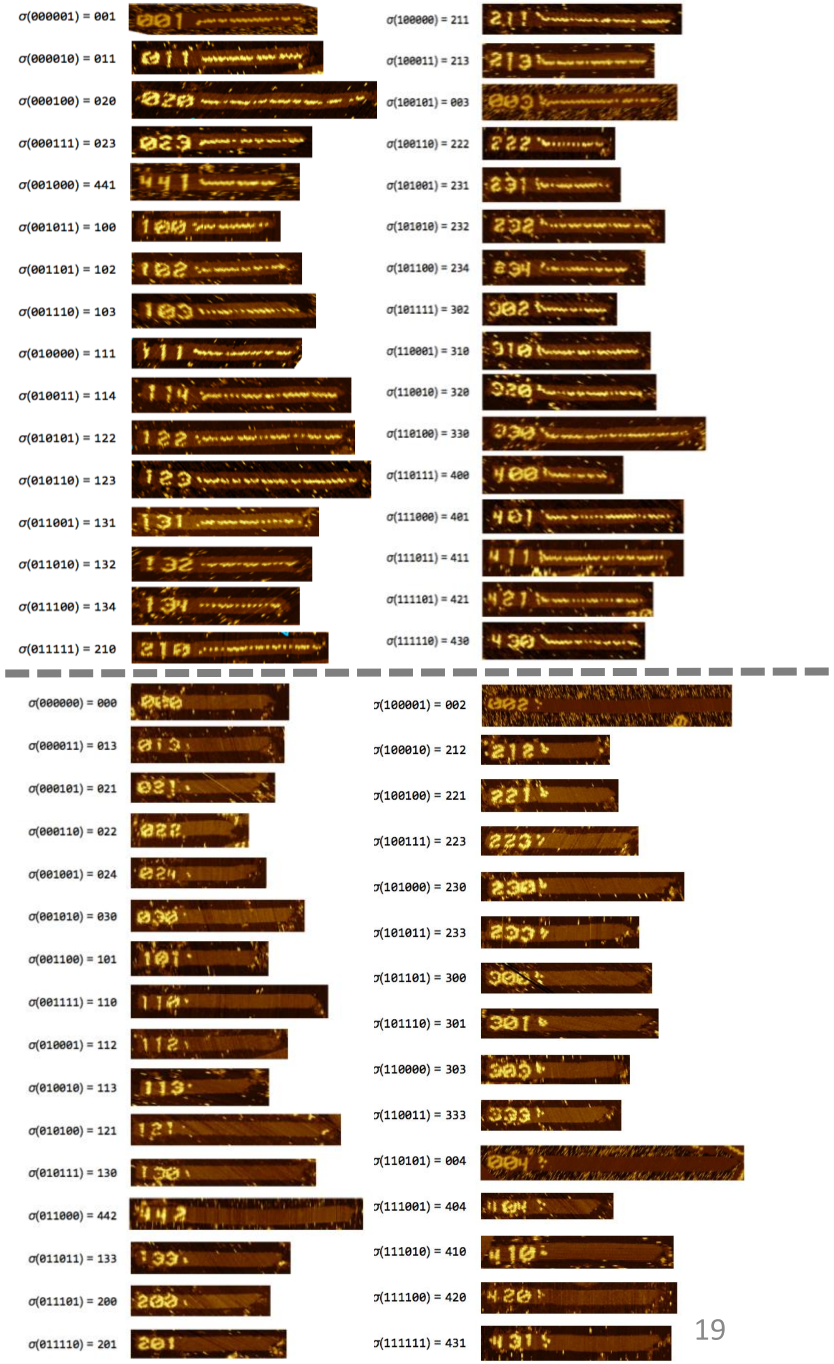
seed encoding **100101**

single set of tiles computing parity



seed encoding **110101**

2^6 seeds:



[*Iterated Boolean circuit computation via a programmable DNA tile array*. Woods, Doty, Myhrvold, Hui, Wu, Yin, Winfree, <u>in preparation</u>, work presented in DNA 23 talk tomrrow by Damien Woods]

So tiles can compute… what’s that good for?

So tiles can compute… what's that good for?

Theorem: There is a <u>single</u> 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.

[*Complexity of Self-Assembled Shapes*. Soloveichik and Winfree, SIAM Journal on Computing 2007]

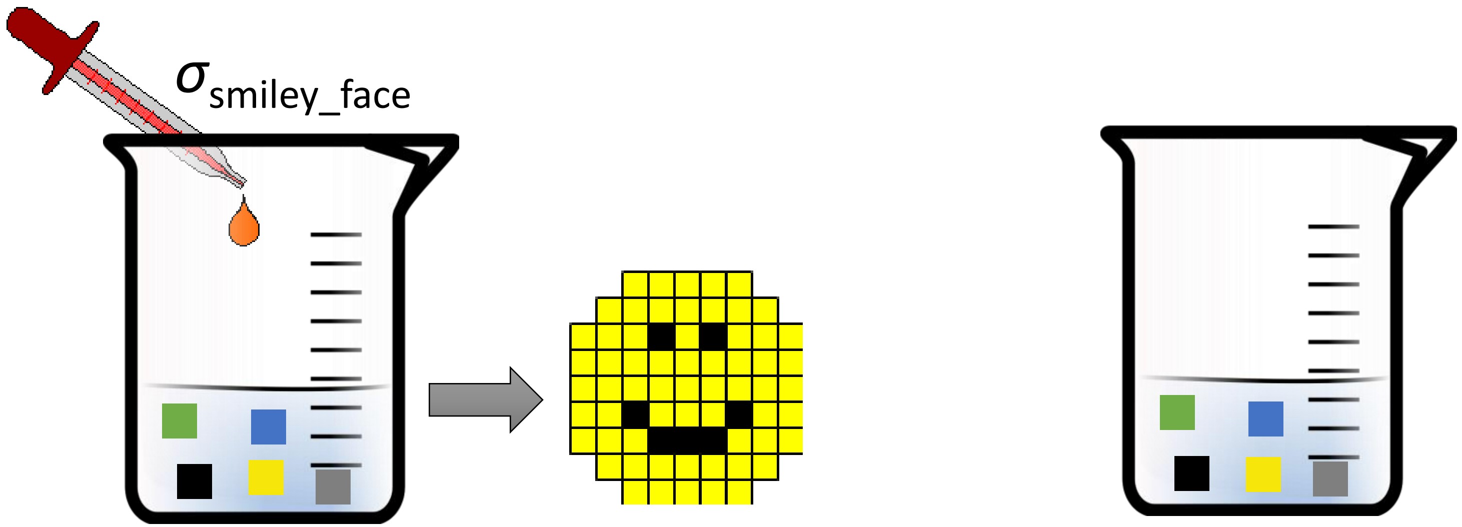
So tiles can compute… what's that good for?

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.

[*Complexity of Self-Assembled Shapes*. Soloveichik and Winfree, SIAM Journal on Computing 2007]

So tiles can compute… what’s that good for?

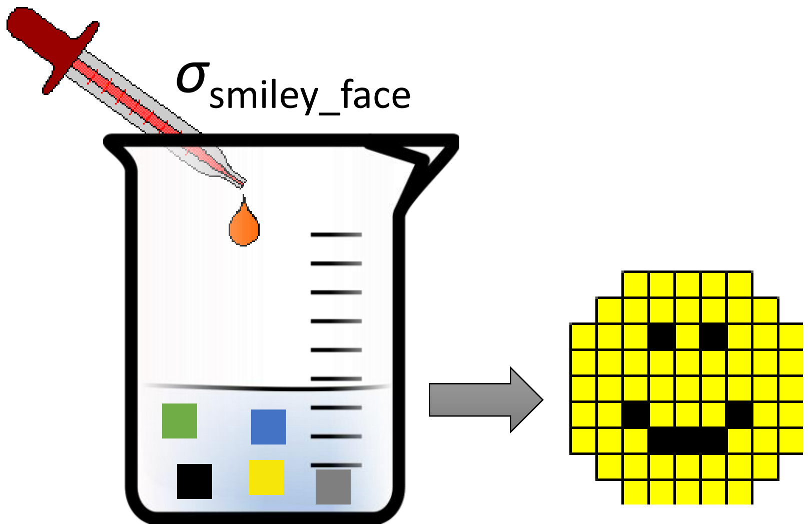
Theorem: There is a single set T of tile types, so that, for any finite shape S, from an appropriately chosen seed σ_S “encoding” S, T self-assembles S.

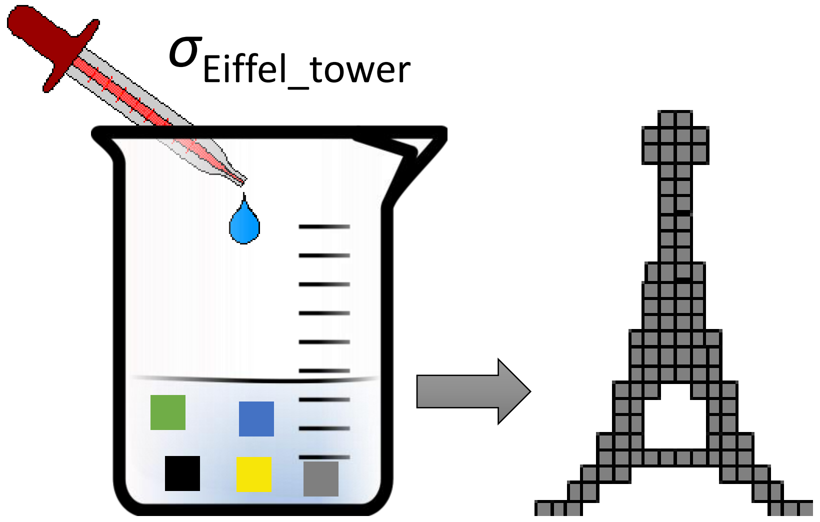


[*Complexity of Self-Assembled Shapes*. Soloveichik and Winfree, SIAM Journal on Computing 2007]

So tiles can compute… what’s that good for?

Theorem: There is a <u>single</u> 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.

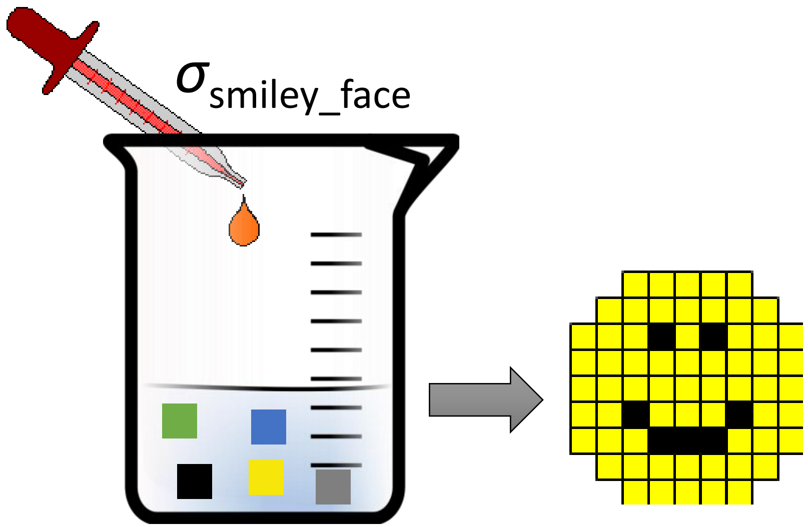


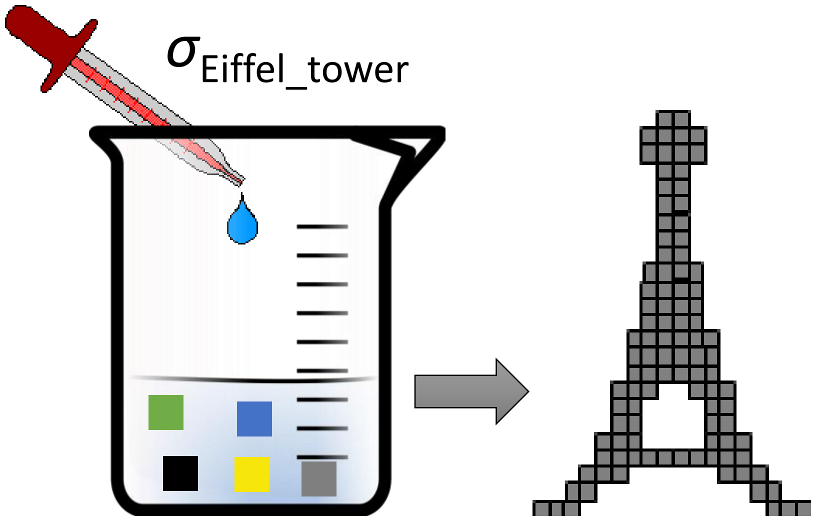


[*Complexity of Self-Assembled Shapes*. Soloveichik and Winfree, <u>SIAM Journal on Computing</u> 2007]

So tiles can compute… what's that good for?

Theorem: There is a <u>single</u> 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.

[*Complexity of Self-Assembled Shapes*. Soloveichik and Winfree, <u>SIAM Journal on Computing</u> 2007]

Active self-assembly

- tiles are *passive*: they bind based on glue identity, and do little else

Active self-assembly

- tiles are *passive*: they bind based on glue identity, and do little else
- active self-assembly: monomers with a "state":

Active self-assembly

- tiles are *passive*: they bind based on glue identity, and do little else
- active self-assembly: monomers with a "state":
  - state can change after binding

Active self-assembly

- tiles are *passive*: they bind based on glue identity, and do little else
- active self-assembly: monomers with a "state":
  - state can change after binding
  - monomer can communicate with neighbors

Active self-assembly

- tiles are *passive*: they bind based on glue identity, and do little else
- active self-assembly: monomers with a “state”:
  - state can change after binding
  - monomer can communicate with neighbors
  - possibly, monomer can move

Active self-assembly at DNA 23

- Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, and Andréa W. Richa. *A Stochastic Approach to Shortcut Bridging in Programmable Matter*
- Yen-Ru Chin, Jui-Ting Tsai and Ho-Lin Chen. *A Minimal Requirement for Self-Assembly of Lines in Polylogarithmic Time*

A Stochastic Approach to Shortcut Bridging in Programmable Matter

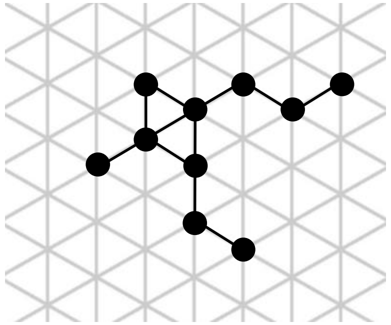
Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, Andréa W. Richa

Abstraction of Programmable Matter: Self-organizing Particle Systems

A collection of simple computational elements that self-organize to solve system-wide problems of movement, configuration, and coordination via fully distributed, local algorithms.

***Geometric* Amoebot Model:** Particles move on the triangular lattice.

- Asynchronous,
- Each has constant-size memory,
- Each can communicate only with neighboring particles,
- There is no common orientation, only common chirality,
- Often desirable that the system stays connected.



A Stochastic Approach to Shortcut Bridging in Programmable Matter

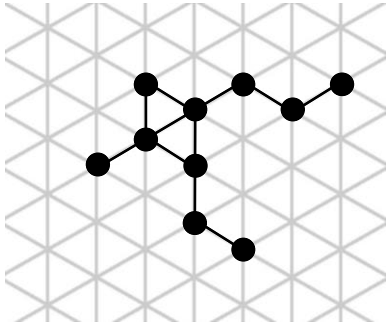
Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, Andréa W. Richa

Abstraction of Programmable Matter: Self-organizing Particle Systems

A collection of simple computational elements that self-organize to solve system-wide problems of movement, configuration, and coordination via fully distributed, local algorithms.

***Geometric* Amoebot Model:** Particles move on the triangular lattice.

- Asynchronous,
- Each has constant-size memory,
- Each can communicate only with neighboring particles,
- There is no common orientation, only common chirality,
- Often desirable that the system stays connected.



(Mostly) deterministic algorithms exist for: Leader election, Shape formation (triangle, hexagon, etc.), Infinite object coating. (See sops.engineering.asu.edu).

A Stochastic Approach to Shortcut Bridging in Programmable Matter

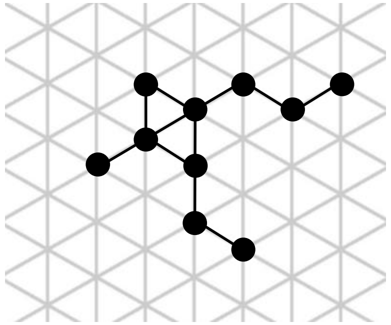
Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, Andréa W. Richa

Abstraction of Programmable Matter: Self-organizing Particle Systems

A collection of simple computational elements that self-organize to solve system-wide problems of movement, configuration, and coordination via fully distributed, local algorithms.

***Geometric* Amoebot Model:** Particles move on the triangular lattice.

- Asynchronous,
- Each has constant-size memory,
- Each can communicate only with neighboring particles,
- There is no common orientation, only common chirality,
- Often desirable that the system stays connected.



(Mostly) deterministic algorithms exist for: Leader election, Shape formation (triangle, hexagon, etc.), Infinite object coating. (See sops.engineering.asu.edu).

Stochastic algorithms exist for:

- **Compression**: gathering a particle system together as tightly as possible.
  - Often found in natural systems.
  - Approach is decentralized, self-stabilizing, and oblivious (no leader necessary).
  - Uses a Markov chain to derive a local algorithm.
- **Shortcut bridging** (DNA23): Generalizes the stochastic approach.

A Stochastic Approach to Shortcut Bridging in Programmable Matter

Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, Andréa W. Richa

A framework for transforming Markov chains into local, asynchronous distributed algorithms:

Markov chain algorithm:

Starting at any configuration, repeat:

1. **Pick a random particle.**
2. Choose a random direction.
3. If certain properties hold, move in that direction with probability **[.....]** .
4. Otherwise, do nothing.

Depends on application

Distributed algorithm:

Each particle continuously executes:

1. Particle proceeds at **its own processing speed** (possibly **variable**).
2. Choose a random direction.
3. If certain properties hold, move in that direction with probability **[.....]** .
4. Otherwise, do nothing.

Poisson clock with individual rate

Depends on application

A Stochastic Approach to Shortcut Bridging in Programmable Matter

Marta Andrés Arroyo, Sarah Cannon, Joshua J. Daymude, Dana Randall, Andréa W. Richa

A framework for transforming Markov chains into local, asynchronous distributed algorithms:

Markov chain algorithm:

Starting at any configuration, repeat:

1. **Pick a random particle.**
2. Choose a random direction.
3. If certain properties hold, move in that direction with probability **[.....]** .
4. Otherwise, do nothing.

Depends on application

Distributed algorithm:

Each particle continuously executes:

1. Particle proceeds at **its own processing speed** (possibly **variable**).
2. Choose a random direction.
3. If certain properties hold, move in that direction with probability **[.....]** .
4. Otherwise, do nothing.

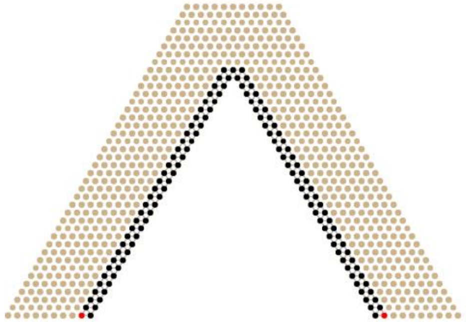
Poisson clock with individual rate

Depends on application

Shortcut Bridging: Bridge across a V-shaped gap in a way that balances minimizing paths between endpoints with cost of bridge.

Chris R. Reid, Matthew J. Lutz, Scott Powell, Albert B. Kao, Iain D. Couzin, and Simon Garnier. Army ants dynamically adjust living bridges in response to a cost-benefit trade-off. *Proceedings of the National Academy of Sciences*, 112(49):15113-15118, 2015.

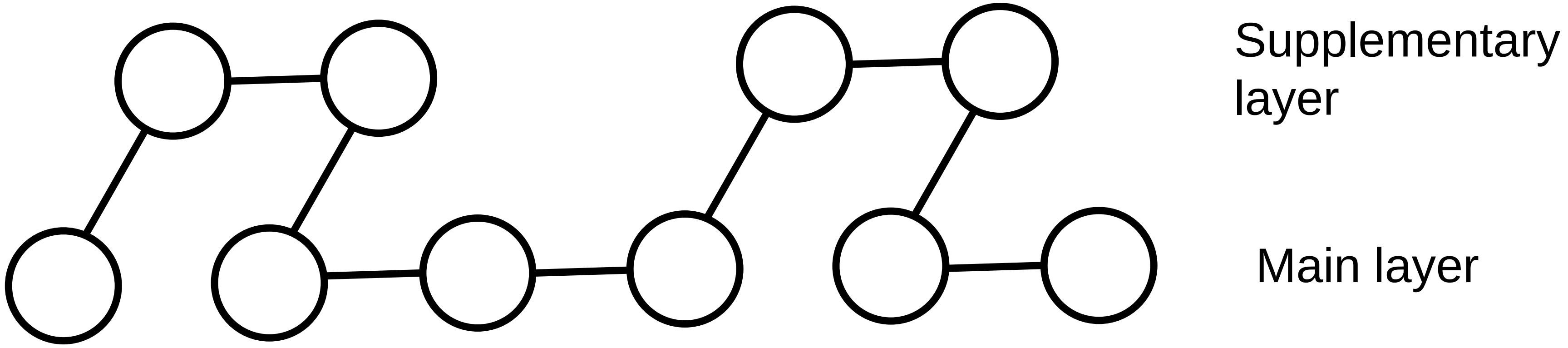
Inspired by ants (Reid et al. 2015)



Our algorithm

Theorem:

With only one supplementary layer,



if each nubot can only perform one state change, then the main layer can only grow linearly.

[Yen-Ru Chin, Jui-Ting Tsai and Ho-Lin Chen. A Minimal Requirement for Self-Assembly of Lines in Polylogarithmic Time, *DNA* 23]

Theorem:

Simple extensions allow exponential growth:

- Two supplementary layers

or

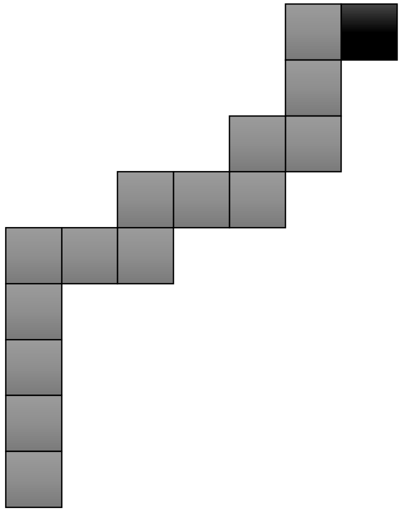
- Disappearance does not require a state change

[Yen-Ru Chin, Jui-Ting Tsai and Ho-Lin Chen. A Minimal Requirement for Self-Assembly of Lines in Polylogarithmic Time, *DNA* 23]

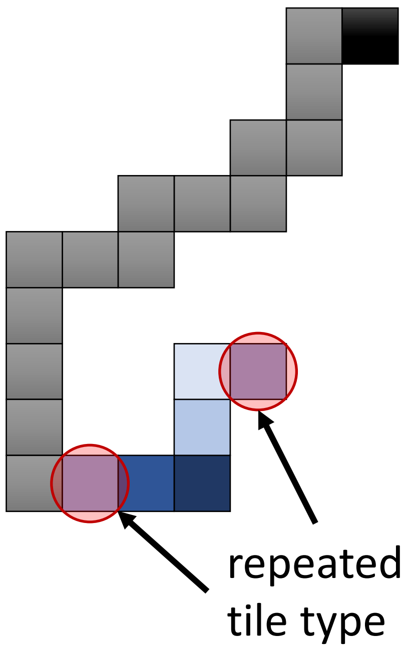
Non-cooperative binding

Non-cooperative binding

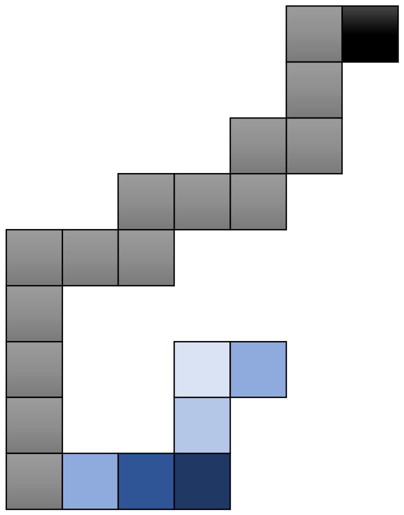
Non-cooperative binding



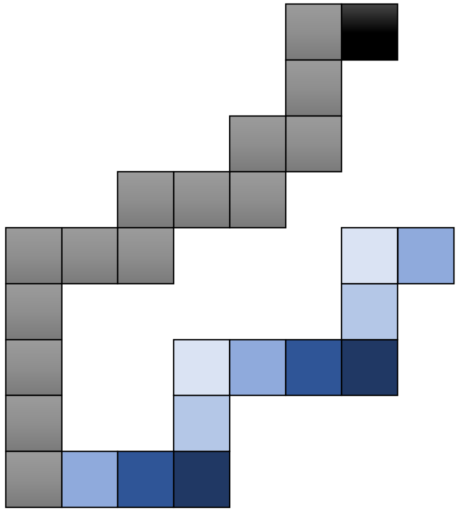
Non-cooperative binding



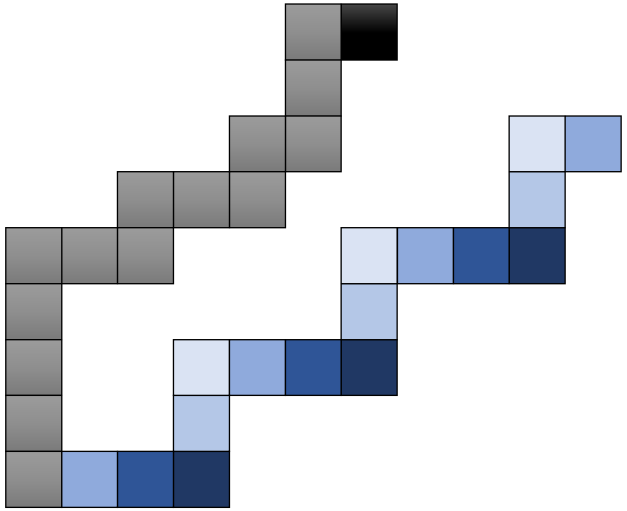
Non-cooperative binding



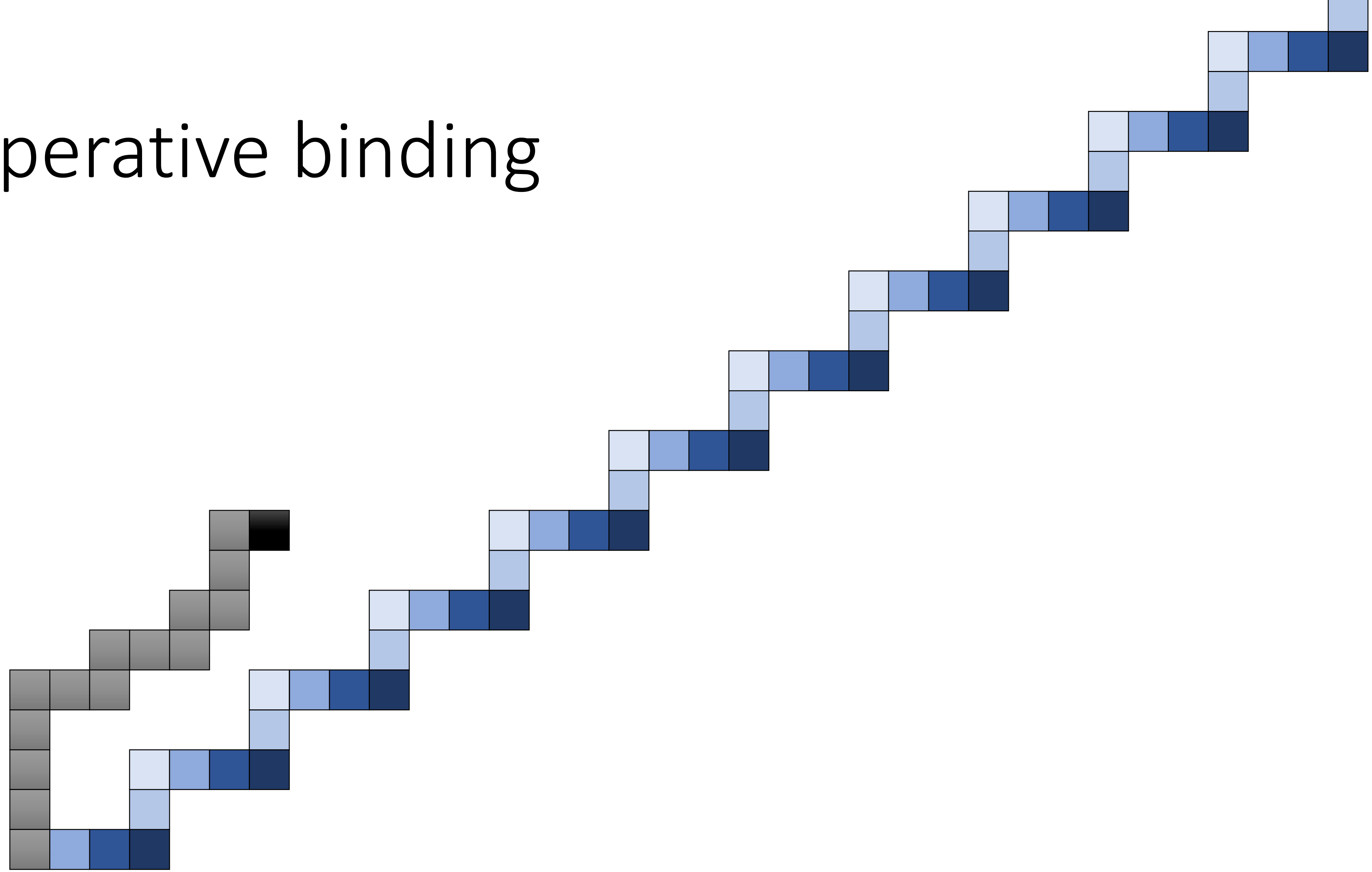
Non-cooperative binding



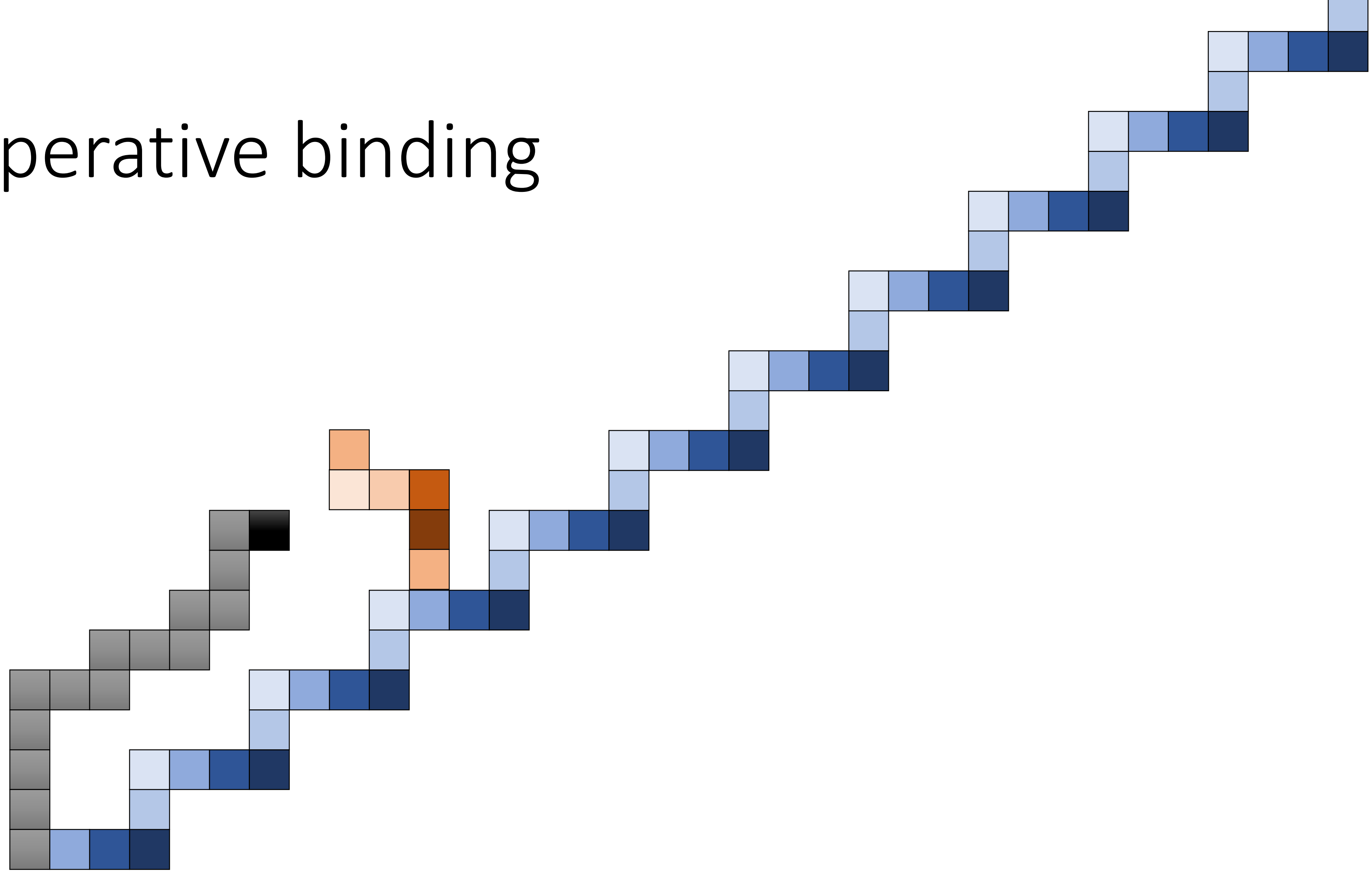
Non-cooperative binding



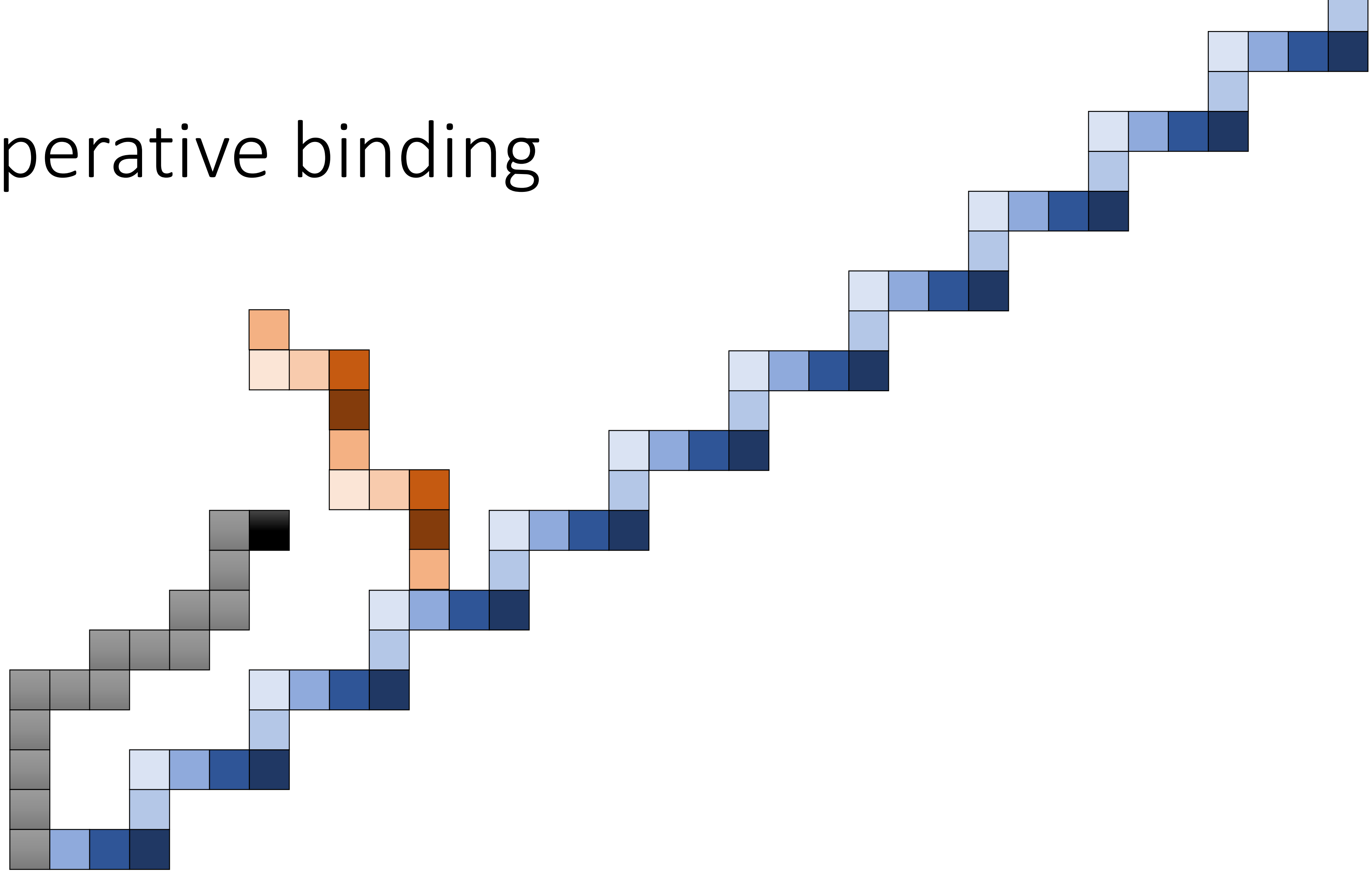
Non-cooperative binding



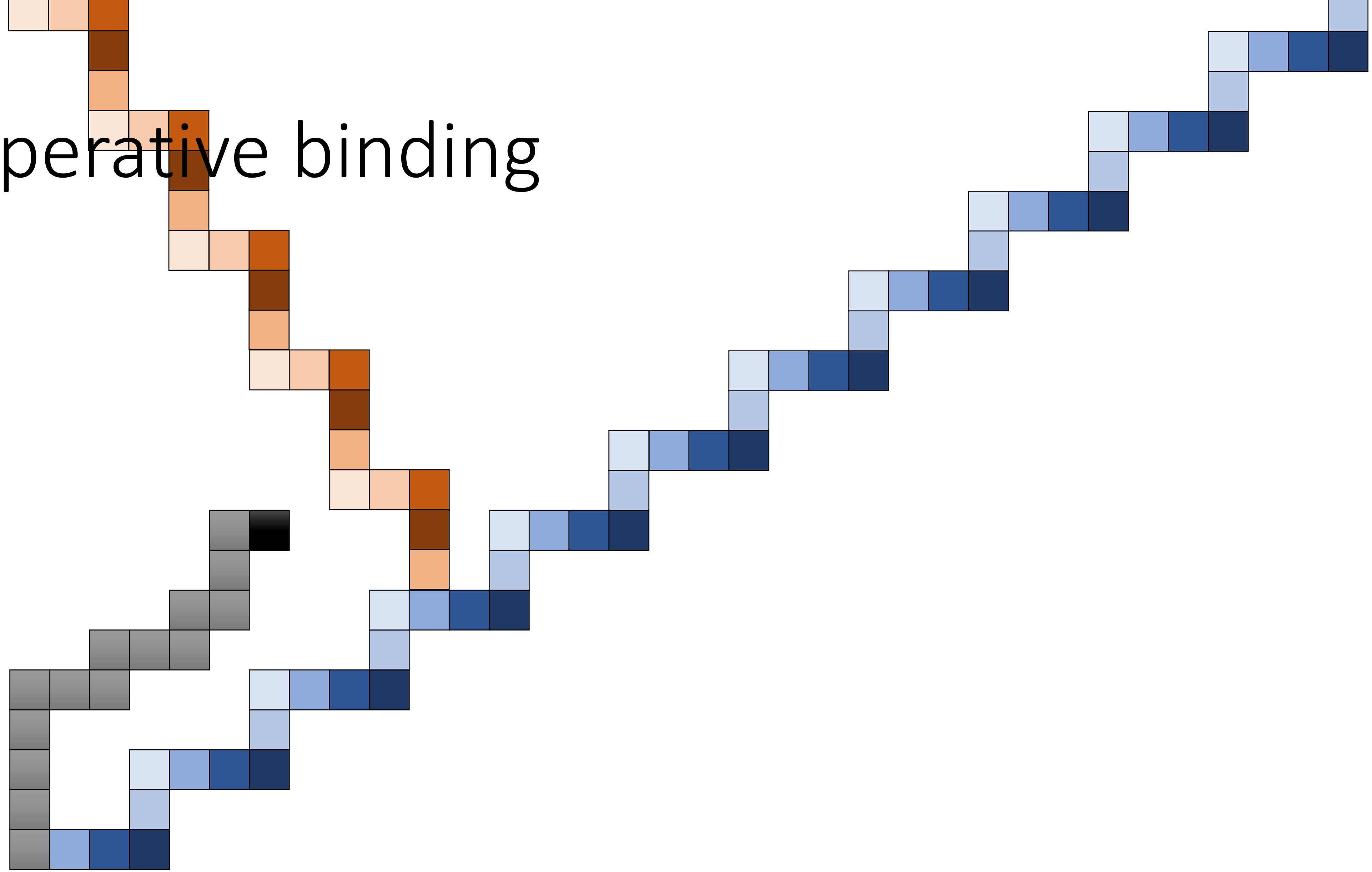
Non-cooperative binding



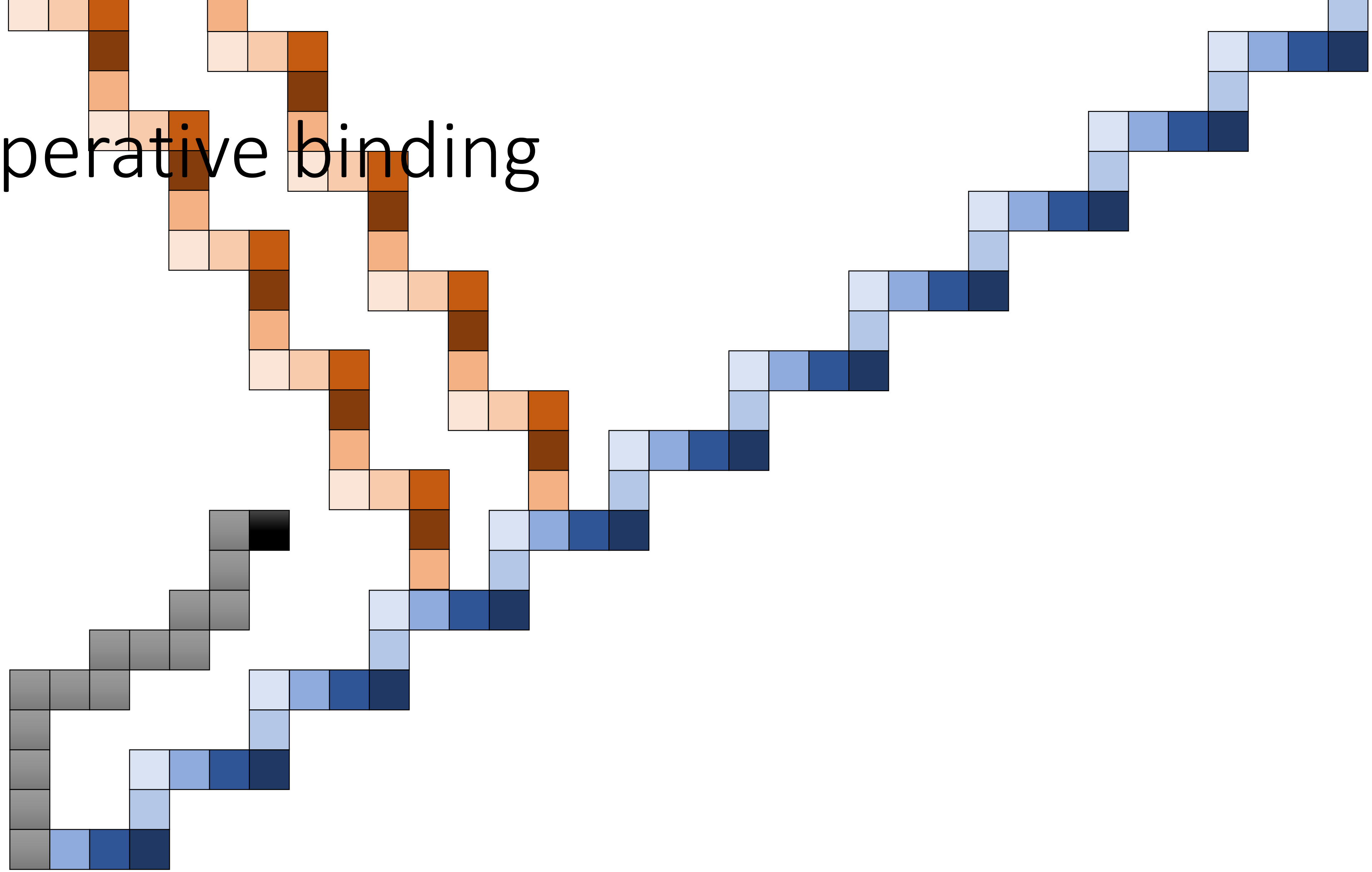
Non-cooperative binding



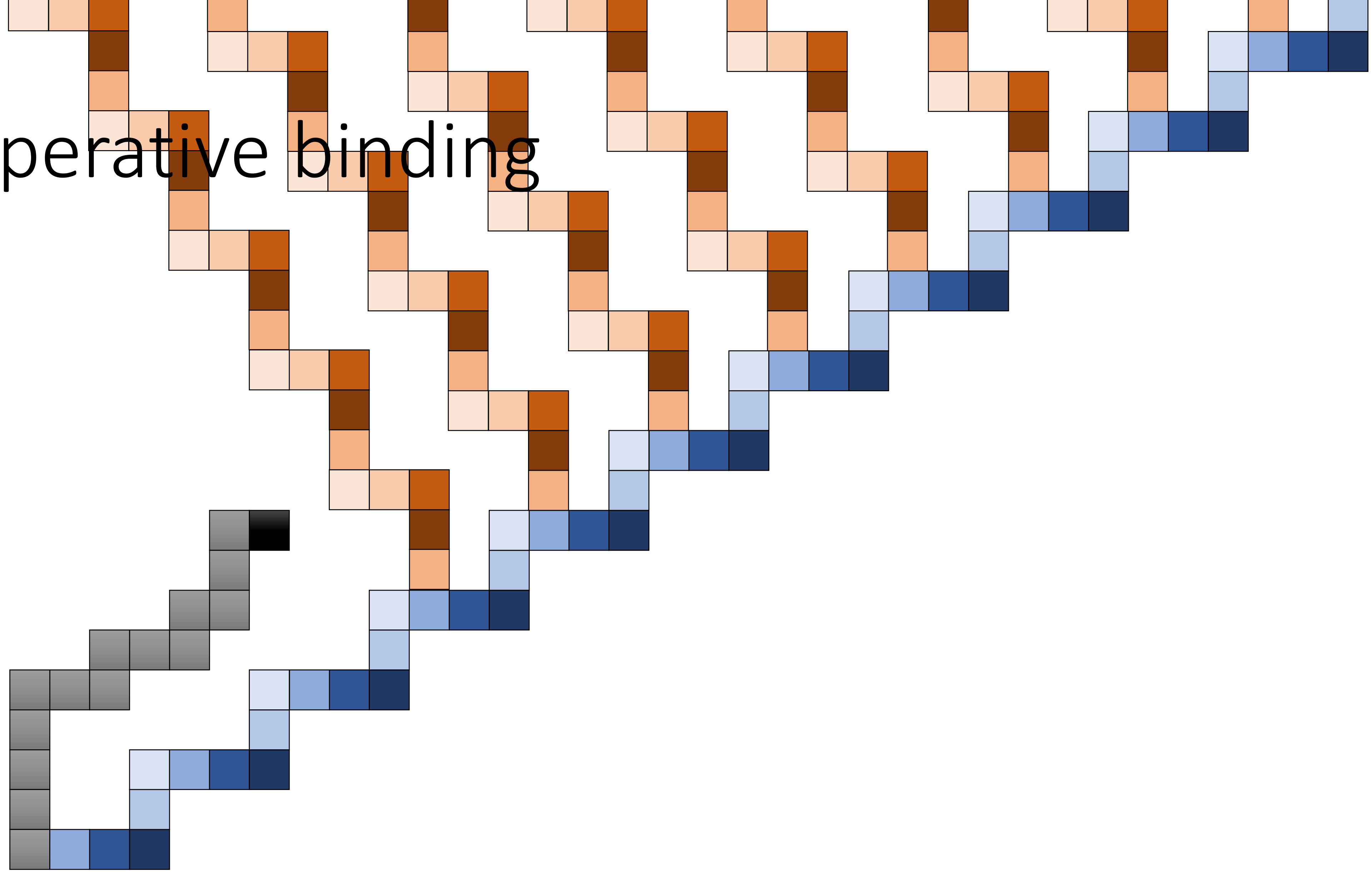
Non-cooperative binding



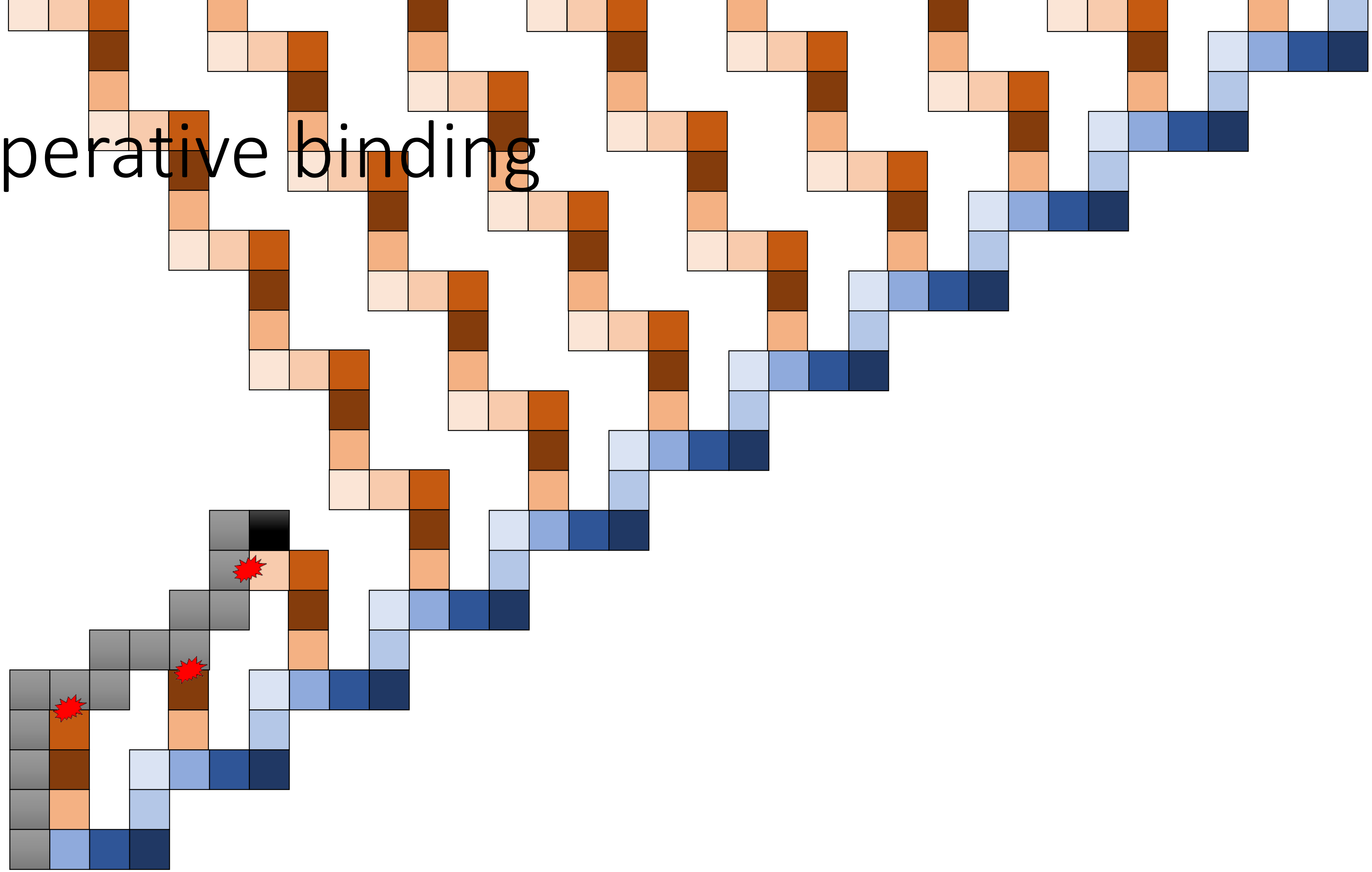
Non-cooperative binding



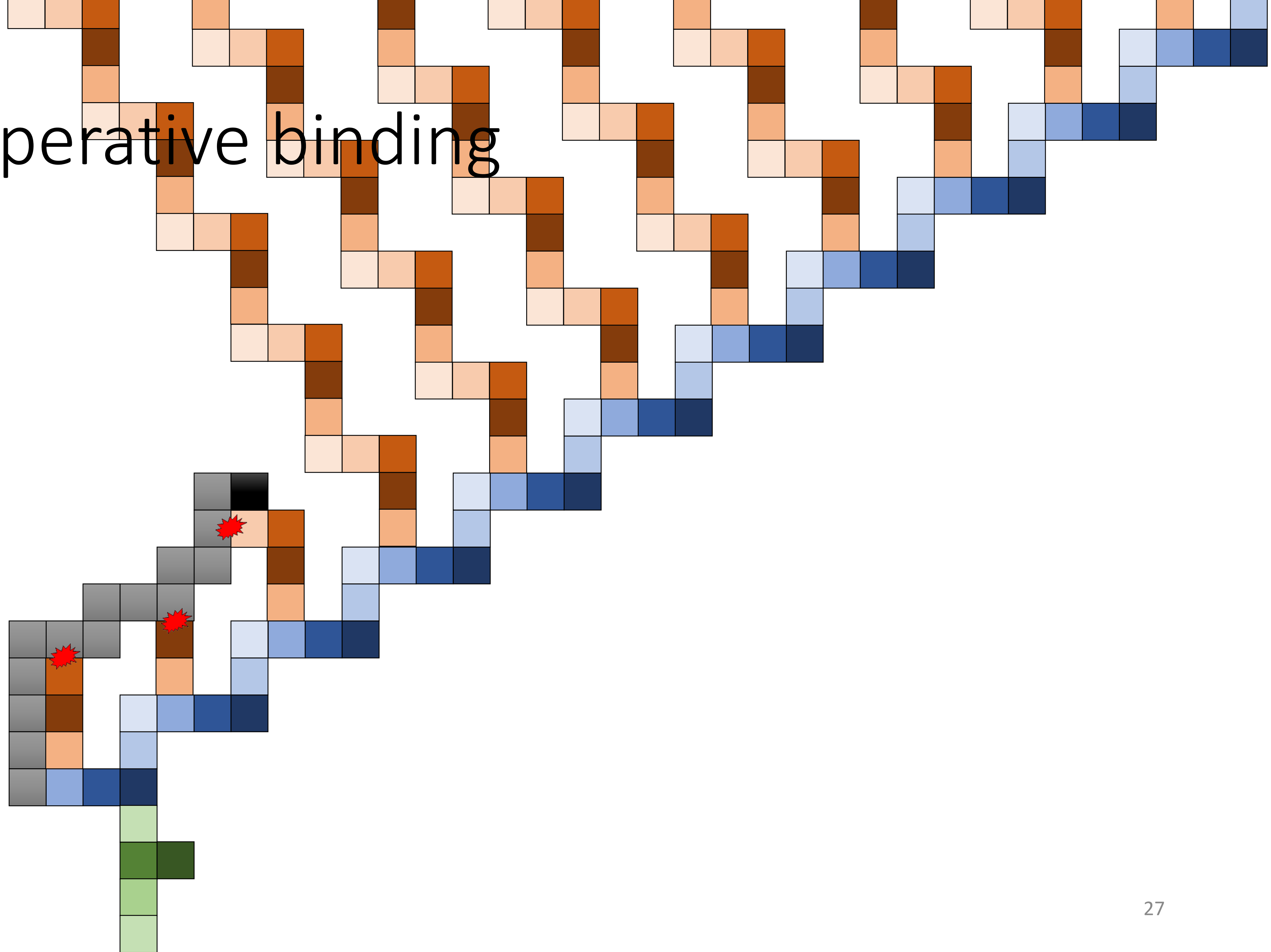
Non-cooperative binding



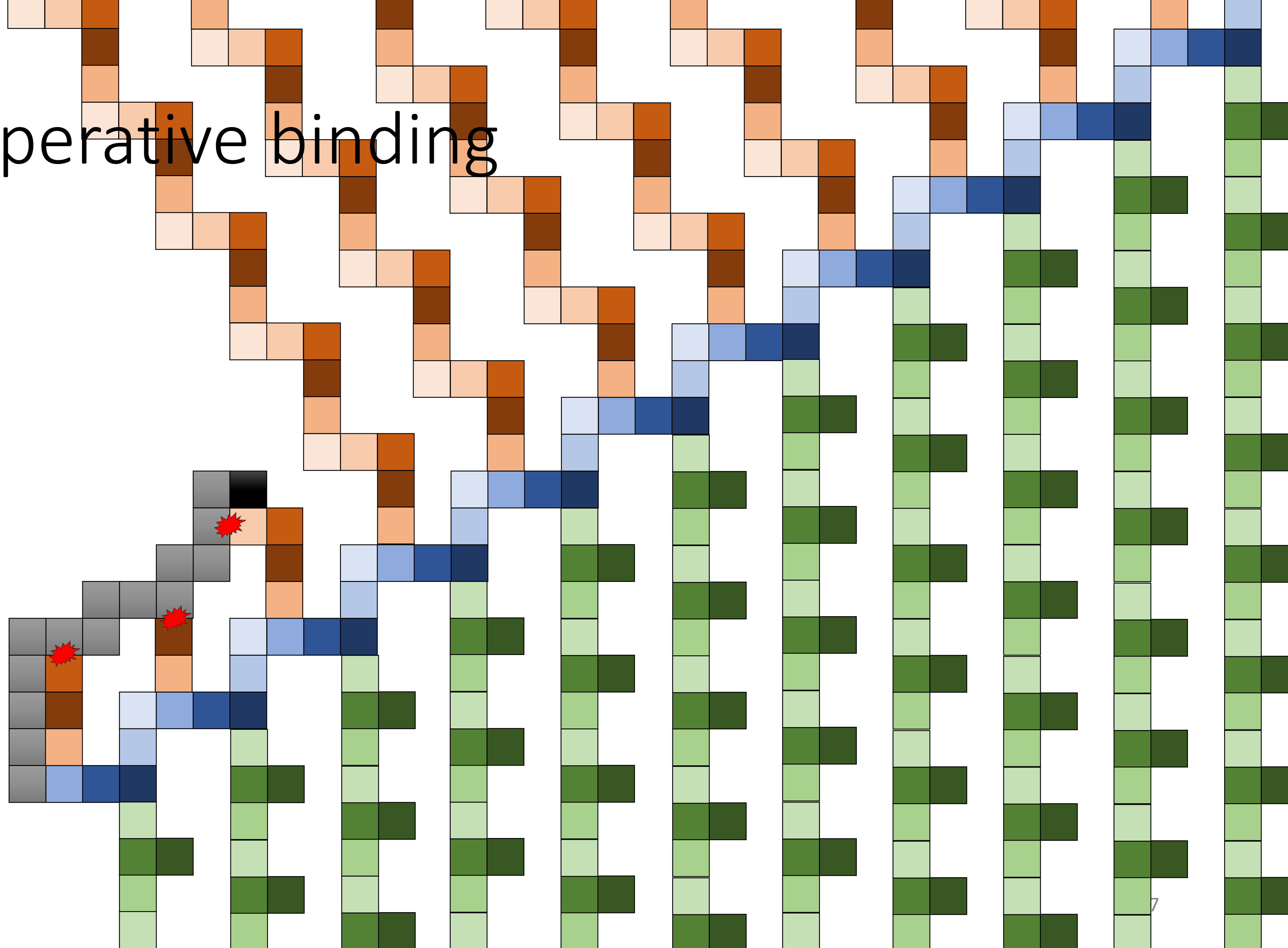
Non-cooperative binding



Non-cooperative binding

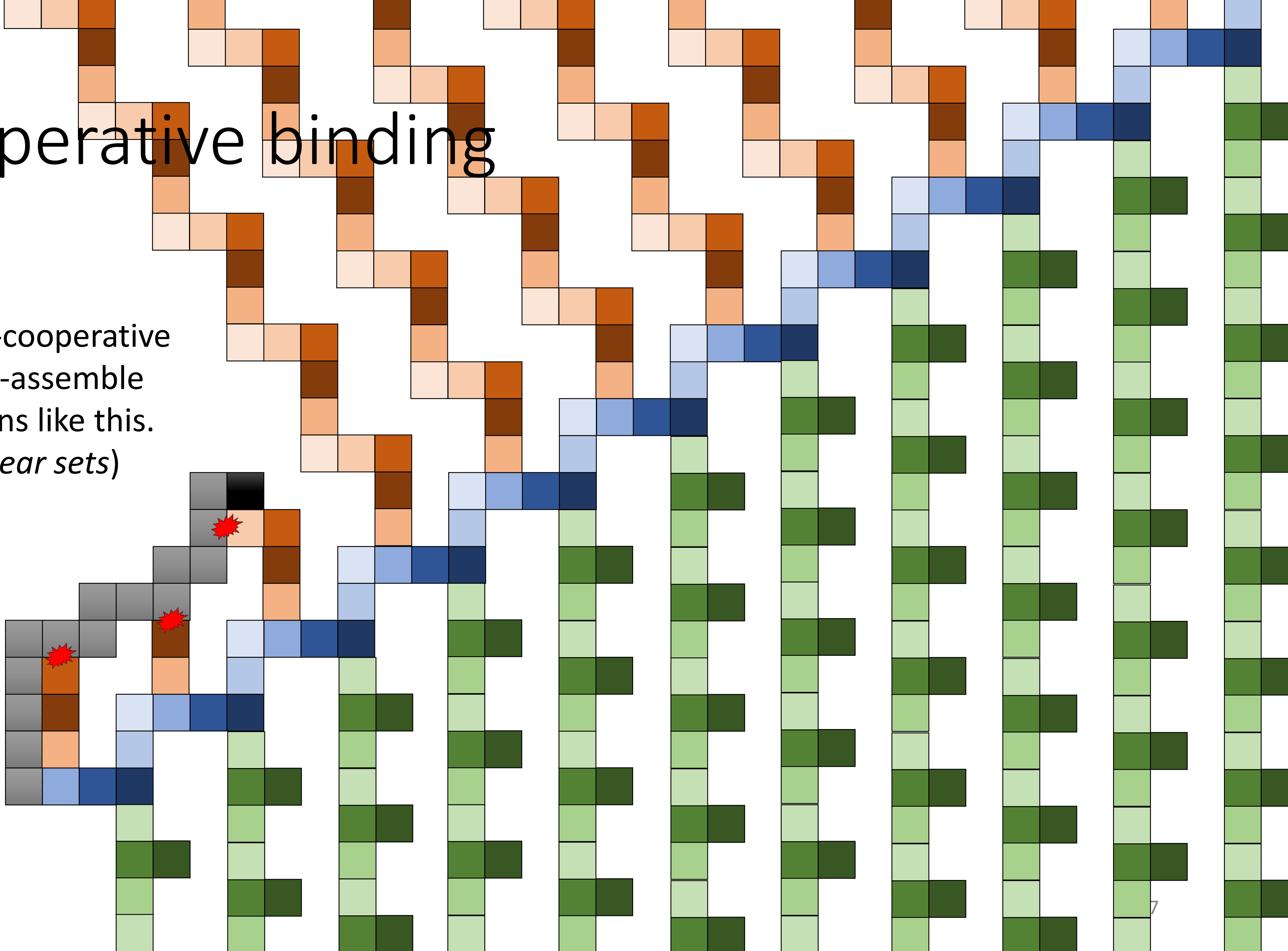


Non-cooperative binding



Non-cooperative binding

Conjecture: Non-cooperative tiles can only self-assemble "periodic" patterns like this. (formally, *semilinear sets*)

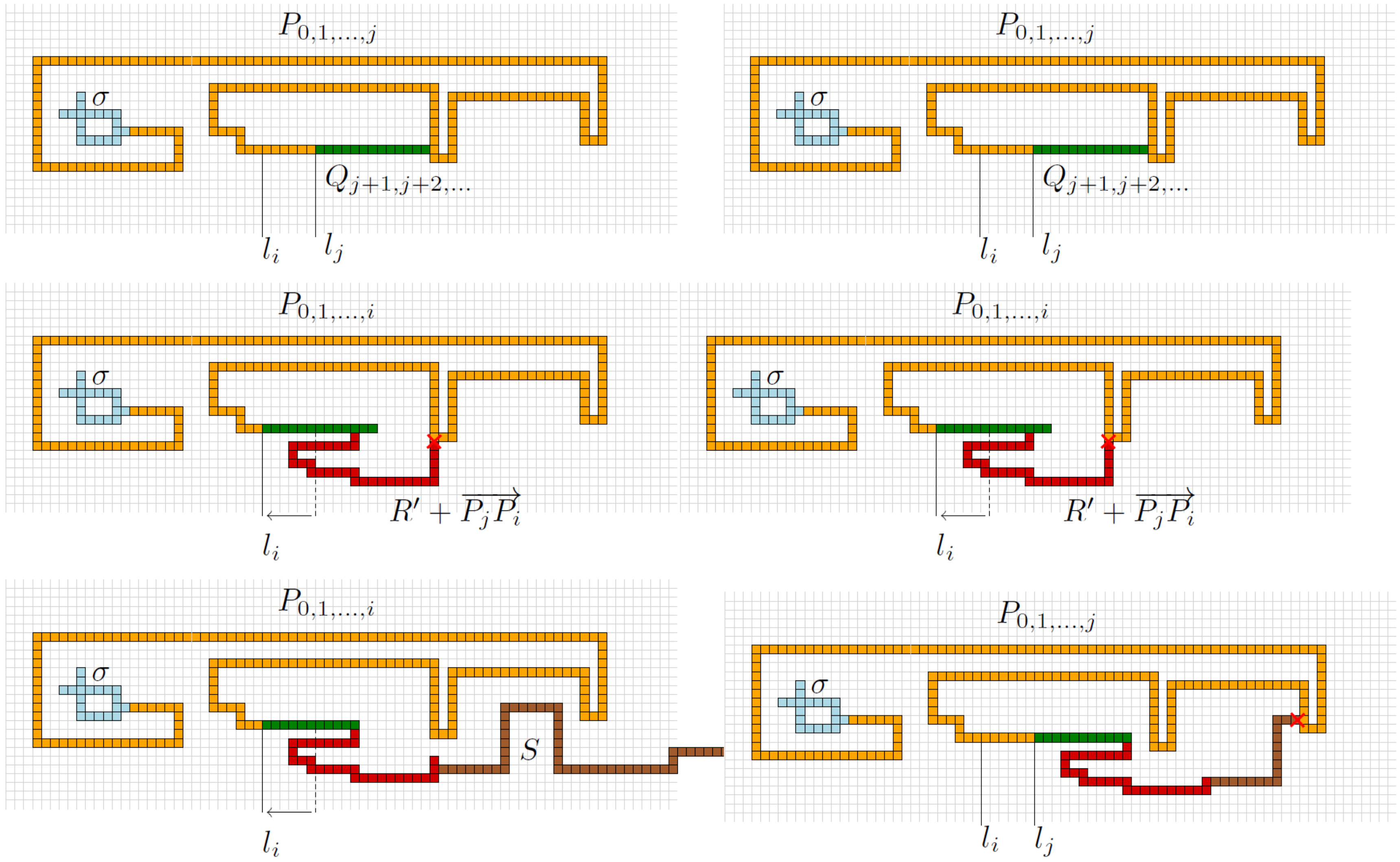


Non-cooperative binding at DNA 23

- Pierre-Étienne Meunier and Damien Woods, *The non-cooperative tile assembly model is not intrinsically universal or capable of bounded Turing machine simulation*

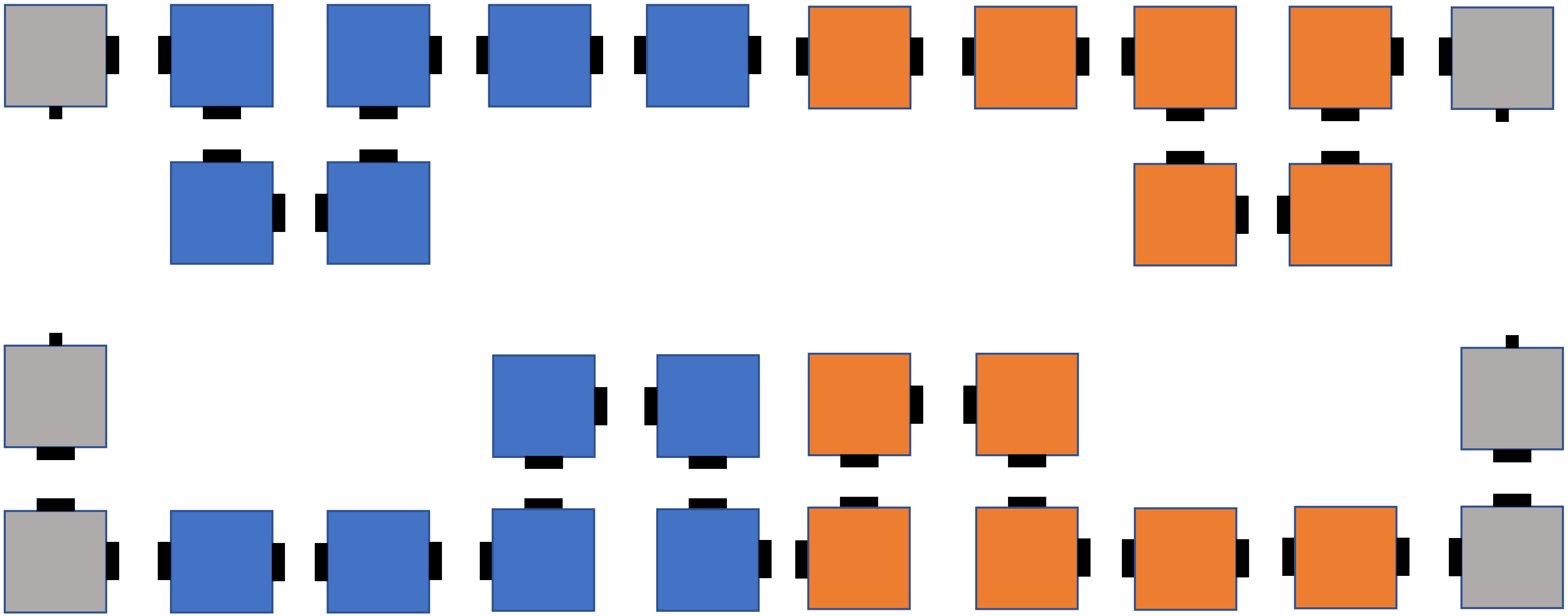
Temperature 1 does not simulate itself

Pierre-Étienne Meunier and Damien Woods

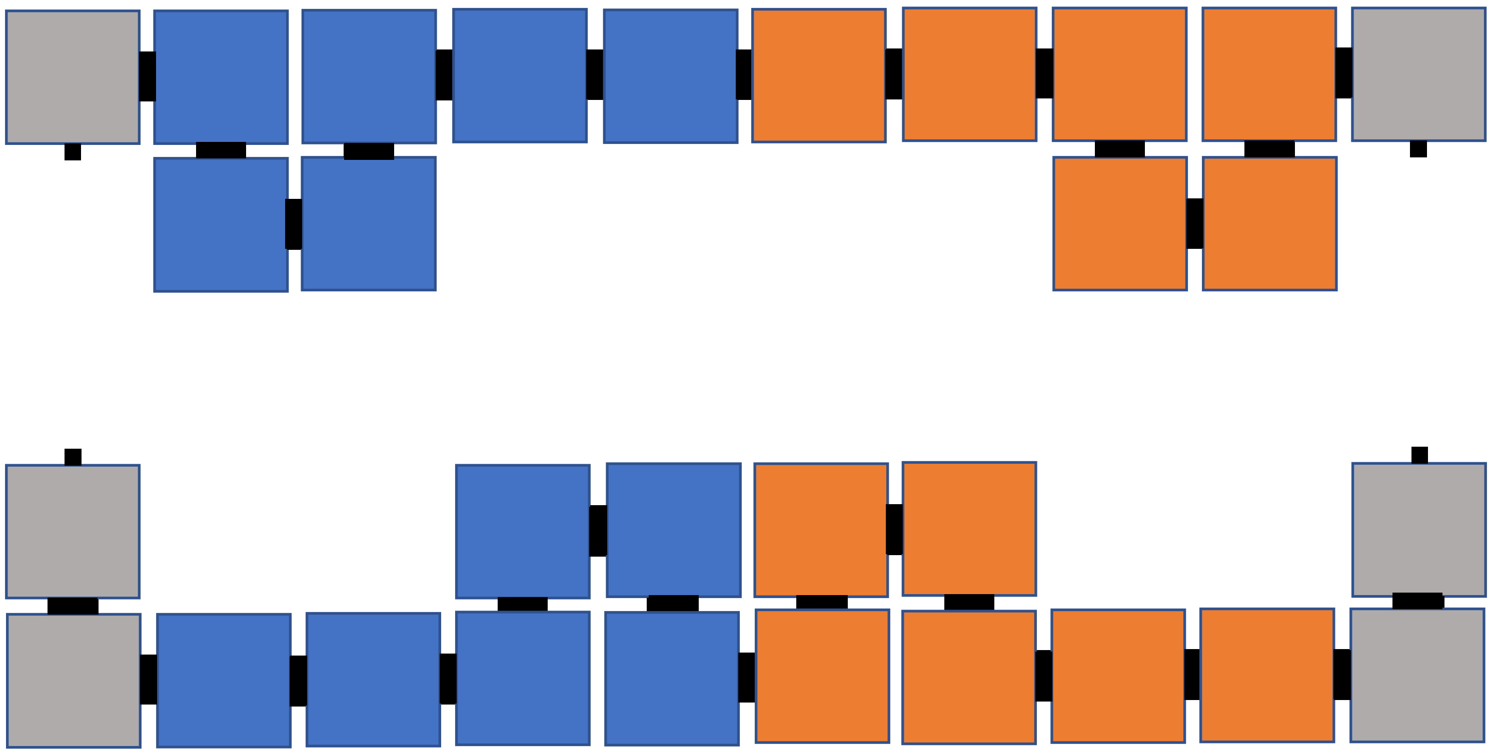


...and much more on our poster!

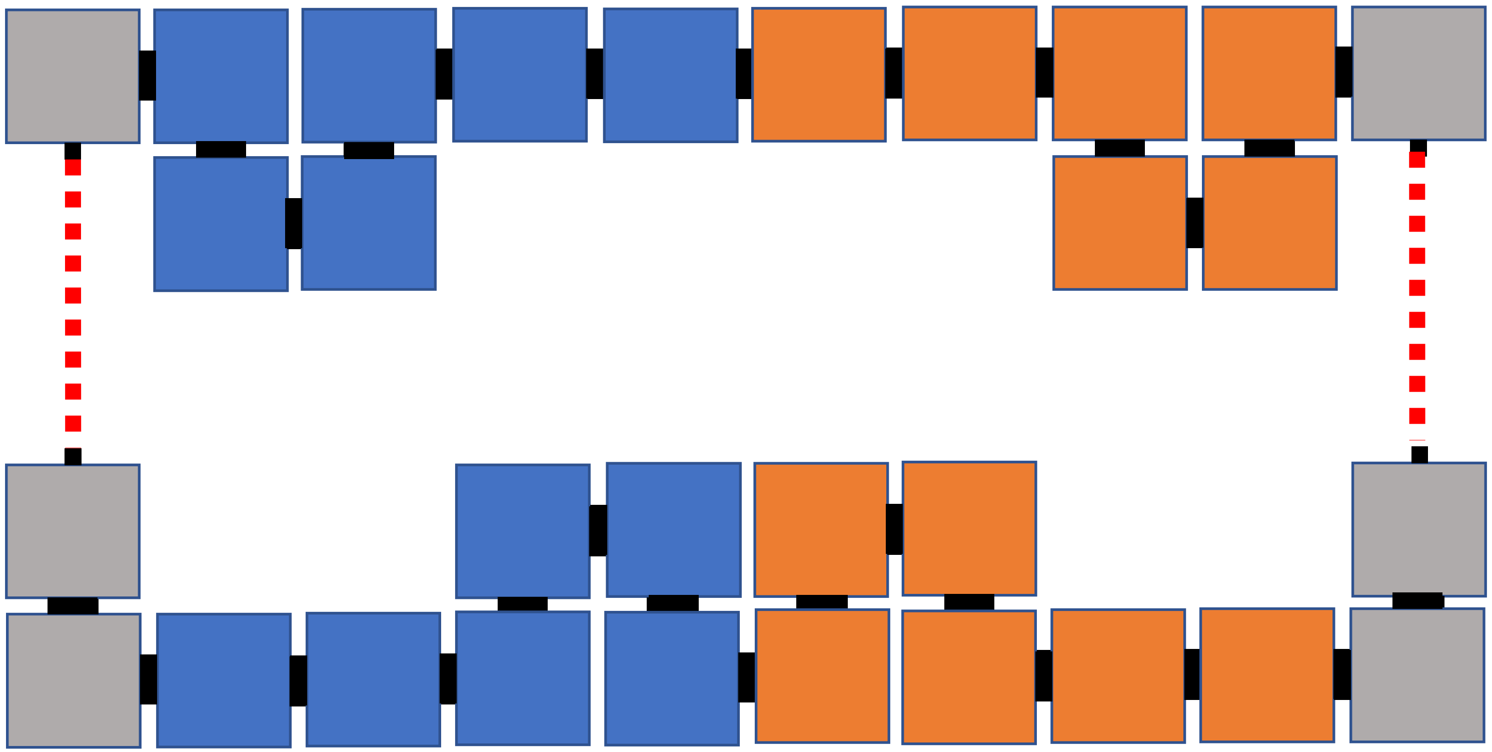
Hierarchical self-assembly



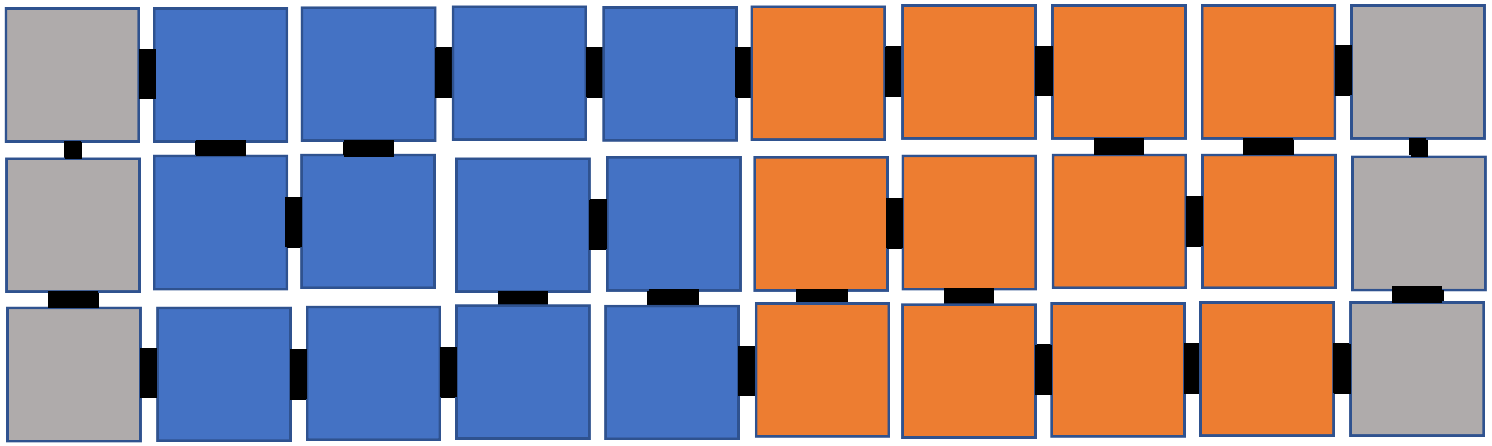
Hierarchical self-assembly



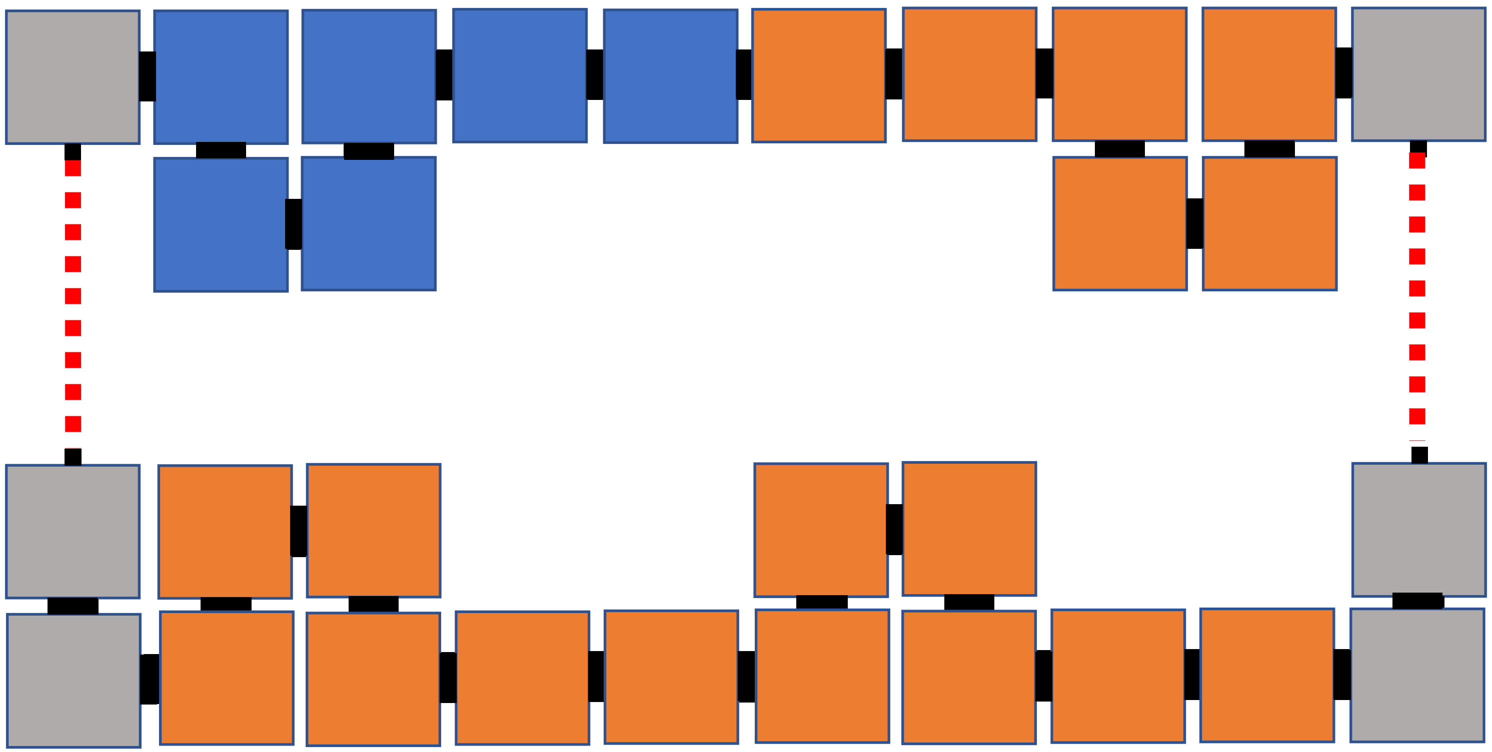
Hierarchical self-assembly



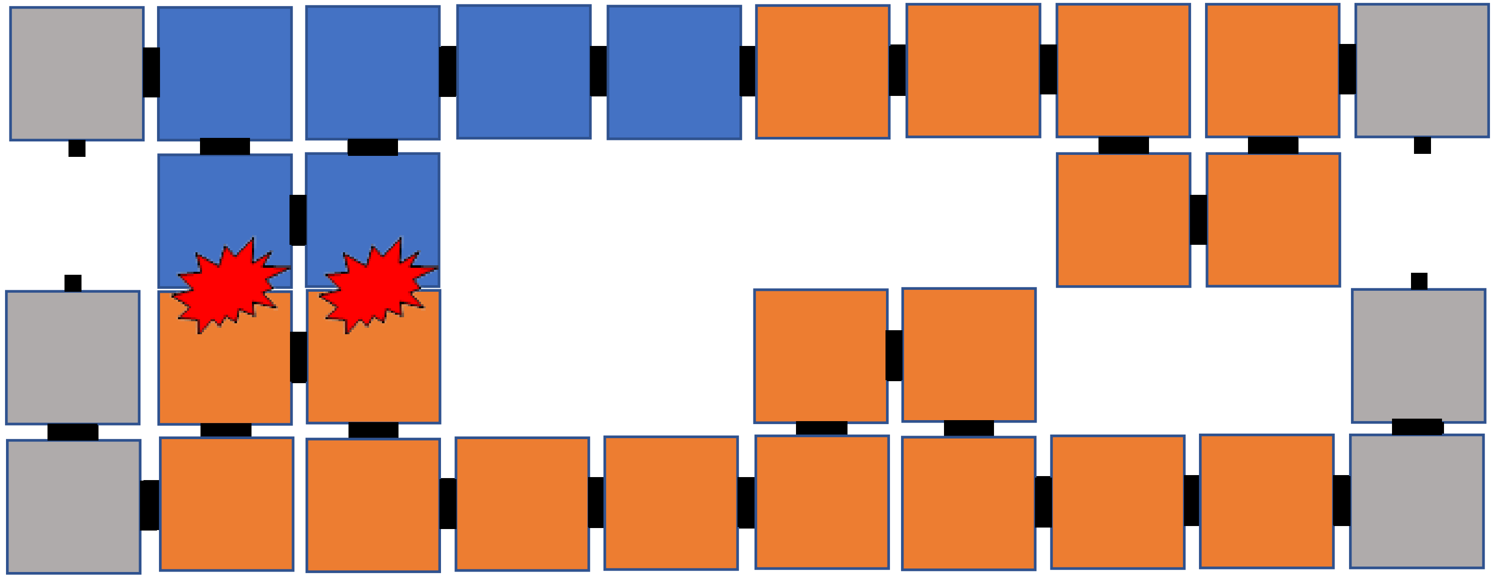
Hierarchical self-assembly



Hierarchical self-assembly



Hierarchical self-assembly



Hierarchical self-assembly at DNA 23

- Robert Schweller, Andrew Winslow, and Tim Wylie, *Complexities for high temperature two-handed tile self-assembly*

Thank you!

Questions?