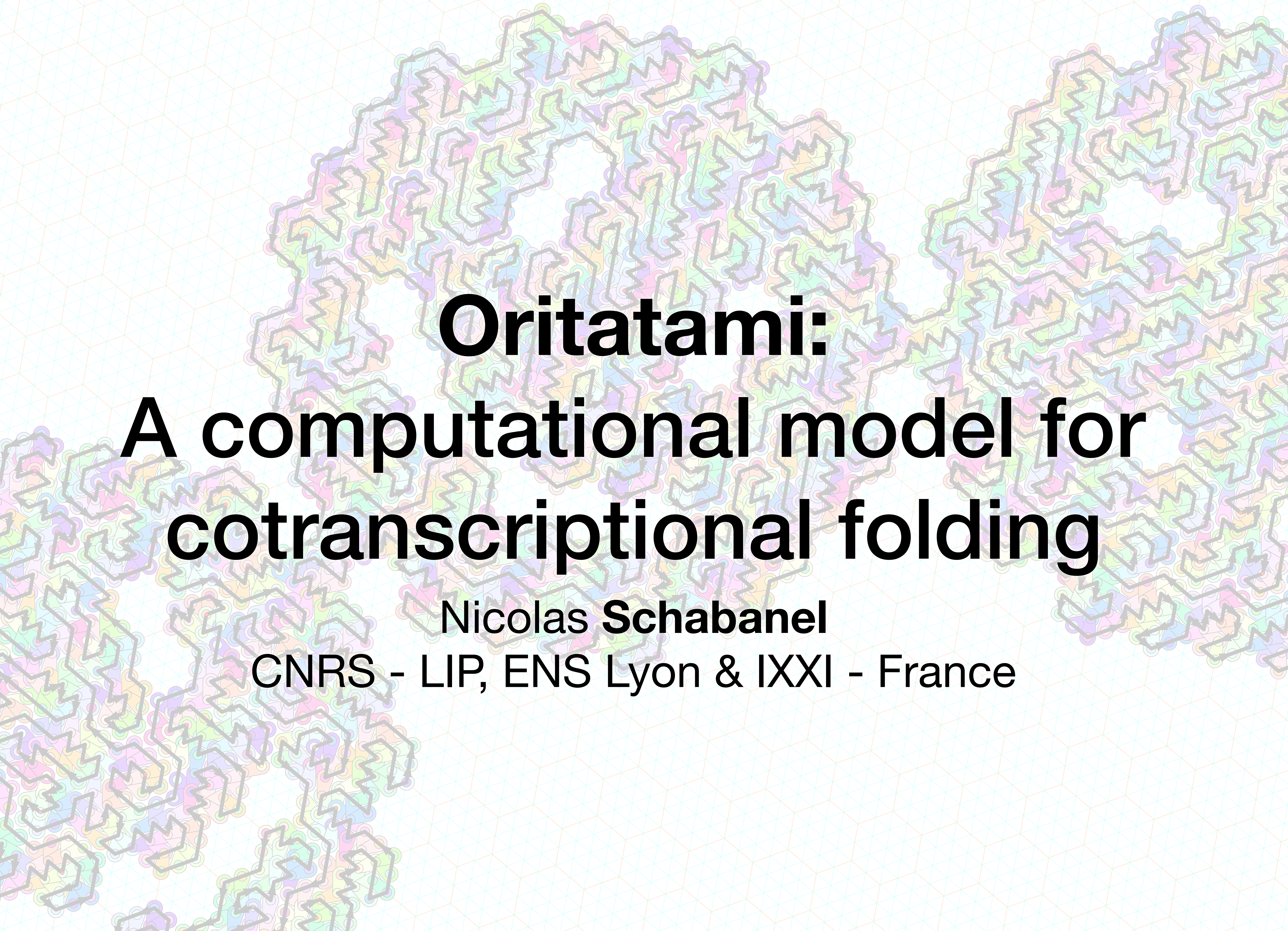
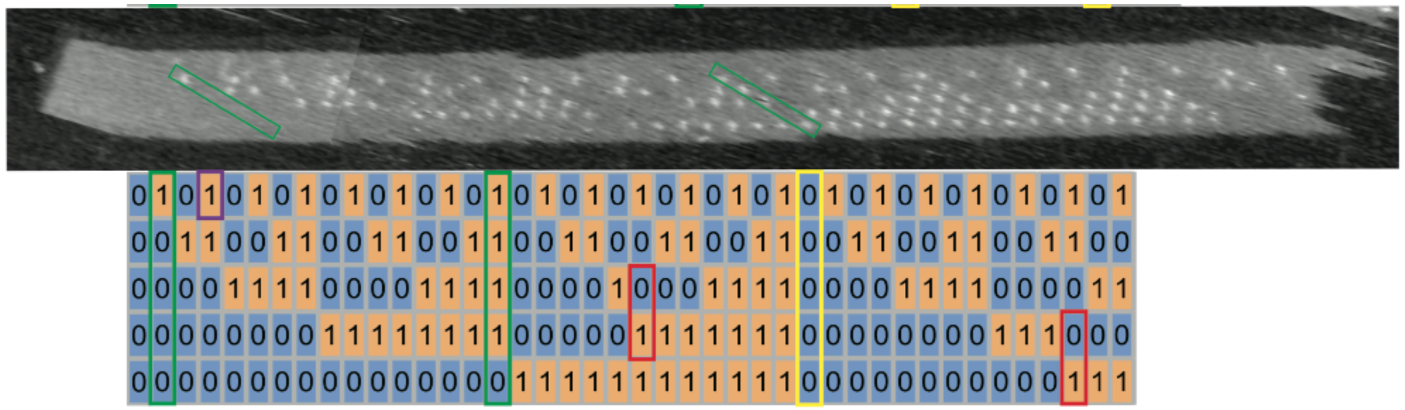


Oritatami:
A computational model for cotranscriptional folding

Nicolas **Schabanel**
CNRS - LIP, ENS Lyon & IXXI - France

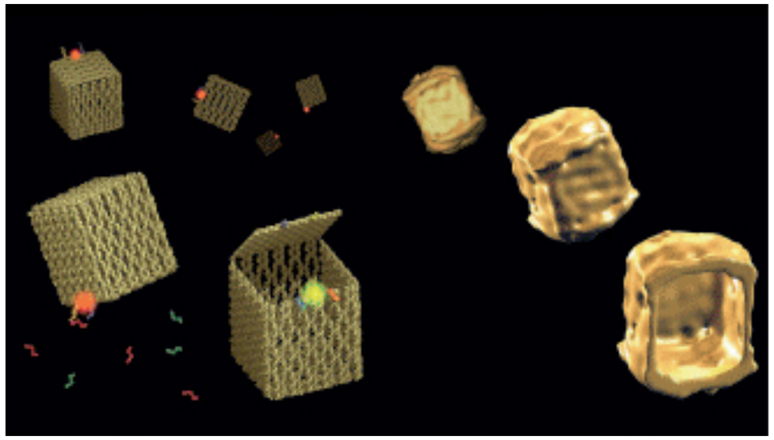
Context: Biomolecular Computing & Engineering

~100 nm

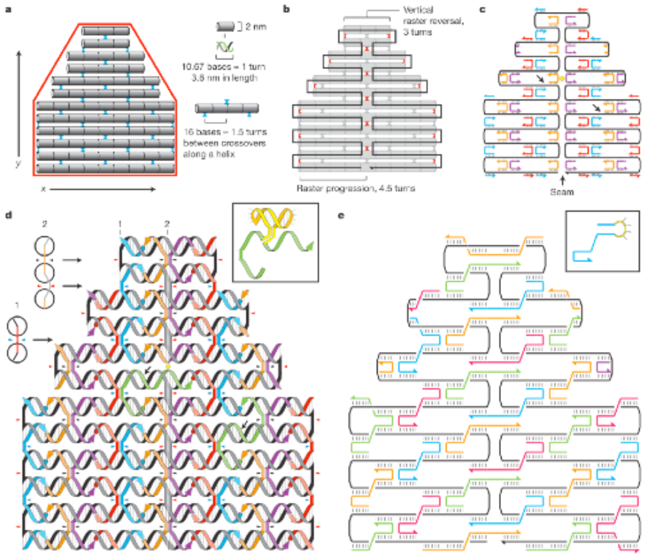


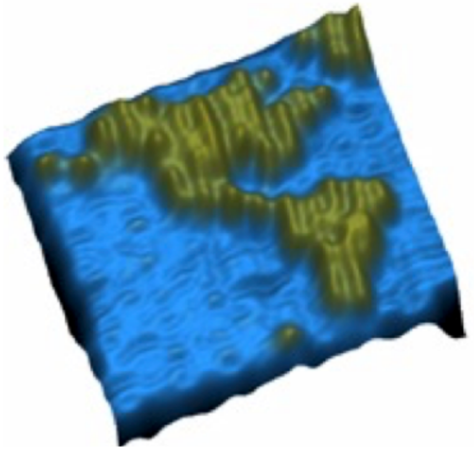
0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1	1	1

Constantine Evans, PhD Thesis, Caltech 2014

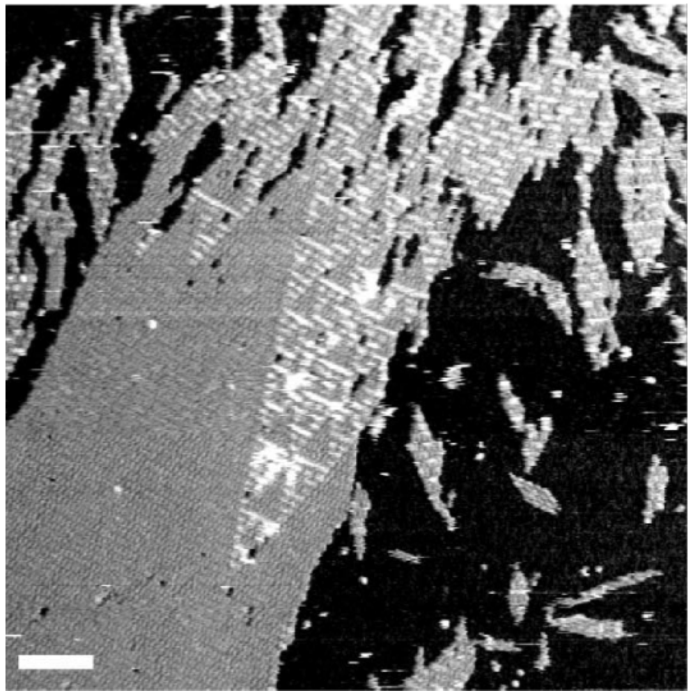


Andersen et al, 2009

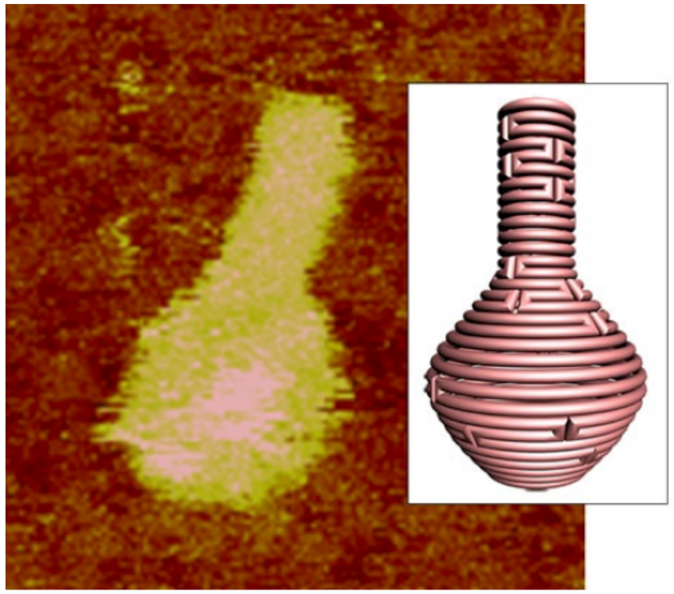




Rothemund, Nature 2006



Fujibayashi et al, 2007

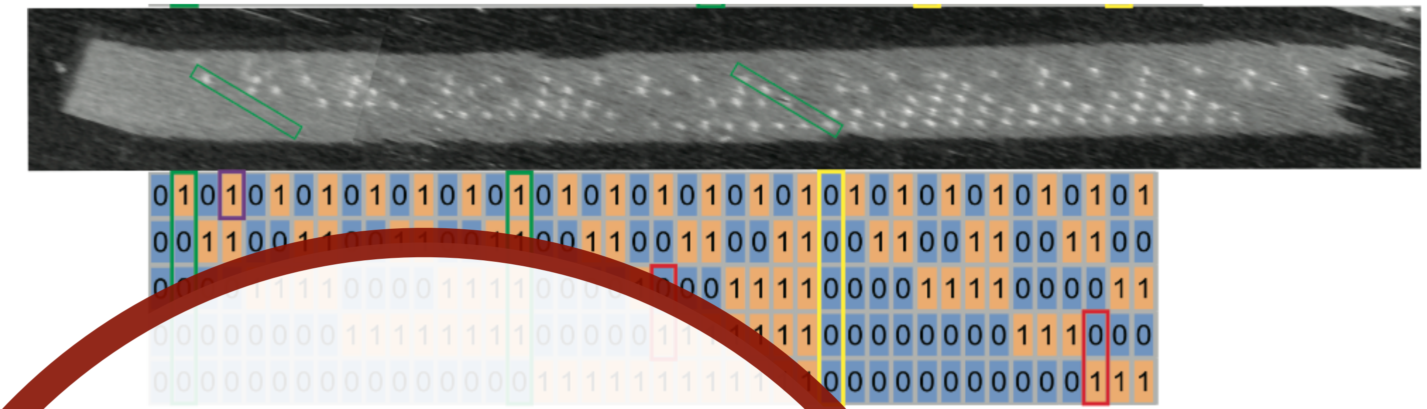


Han et al, Science 2011

Wei, Dai, Yin, Nature 2013

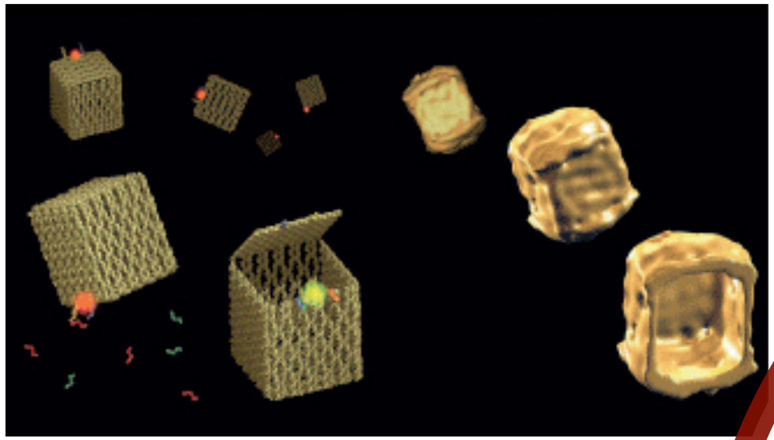
Context: Biomolecular Computing & Engineering

~100 nm



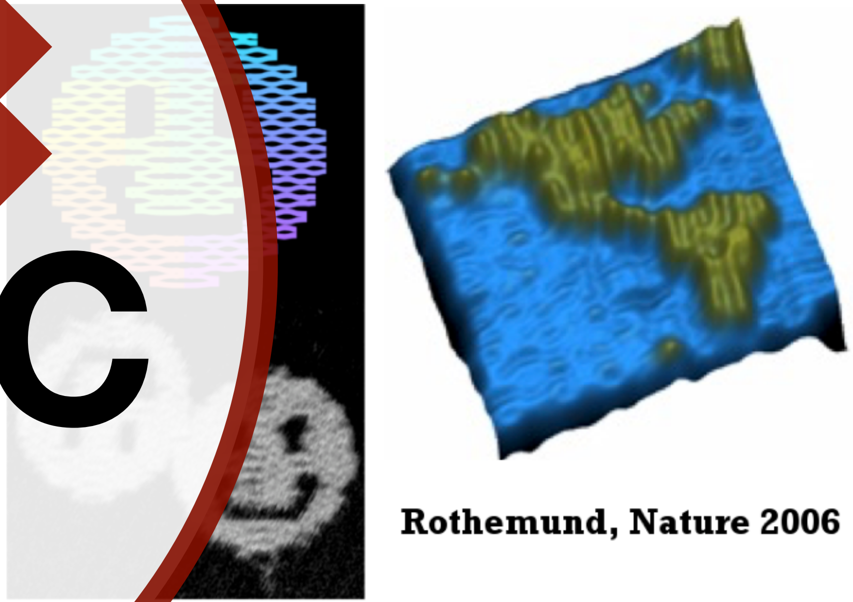
```
0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0
0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 1 1 1
```

Constantine Evans, PhD Thesis, Caltech 2014

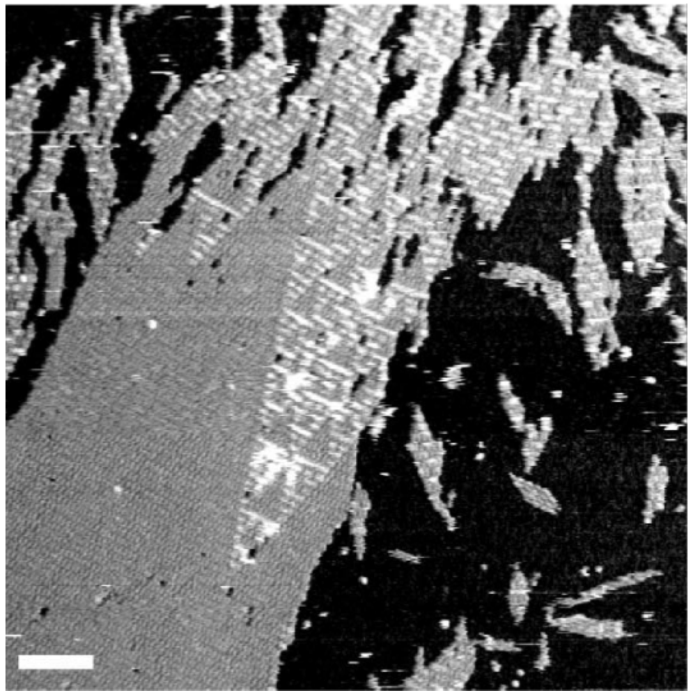


Andersen et al, 200

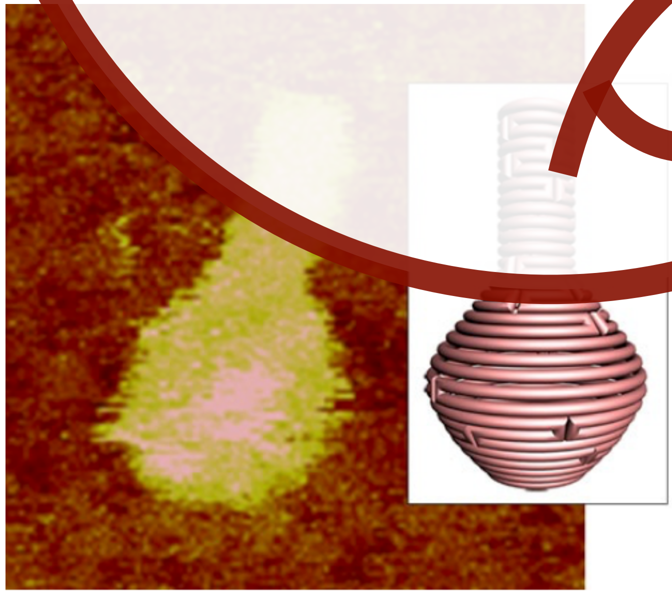
T°≥70°C



Rothemund, Nature 2006



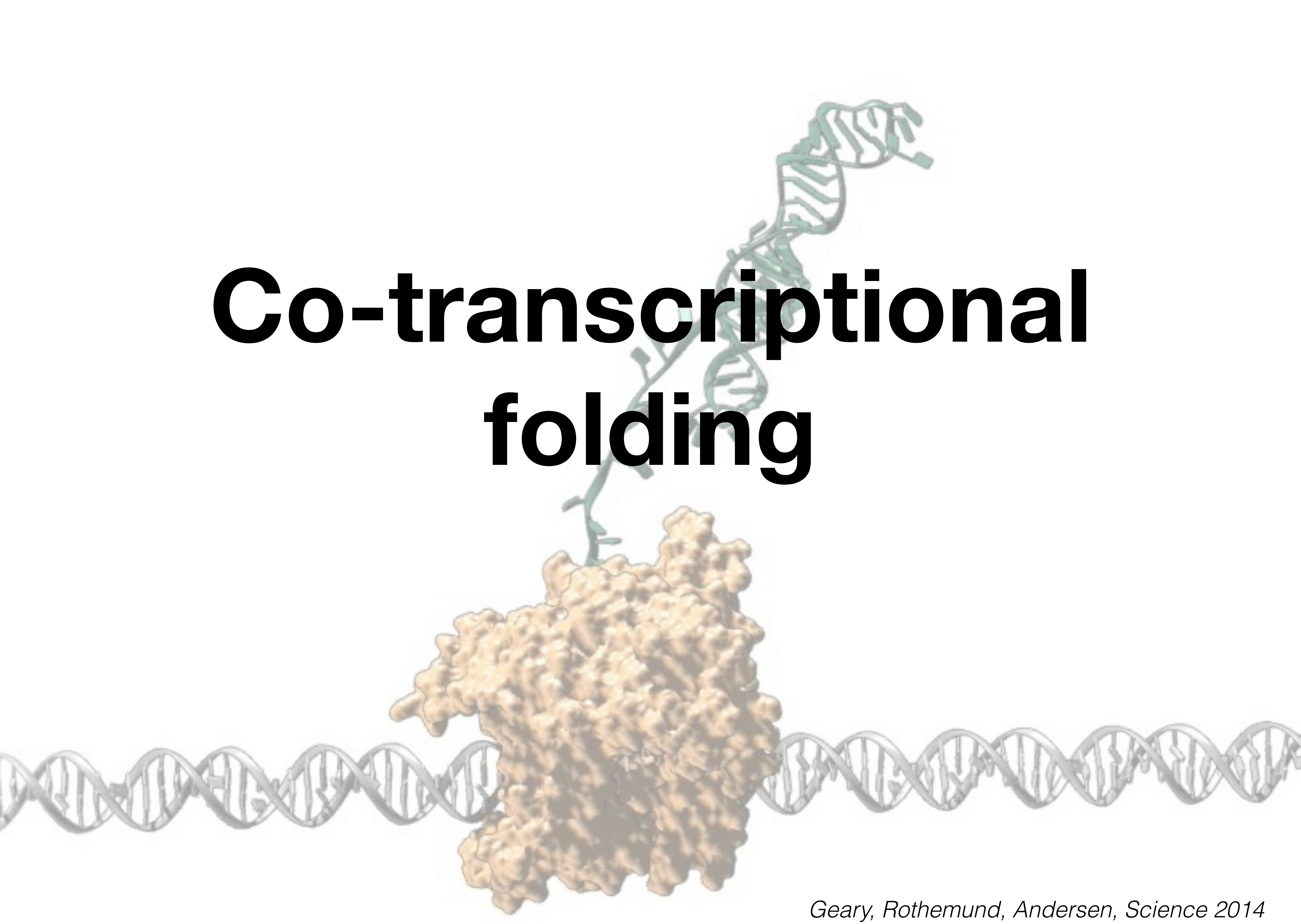
Fujibayashi et al, 2007



Han et al, Science 2011

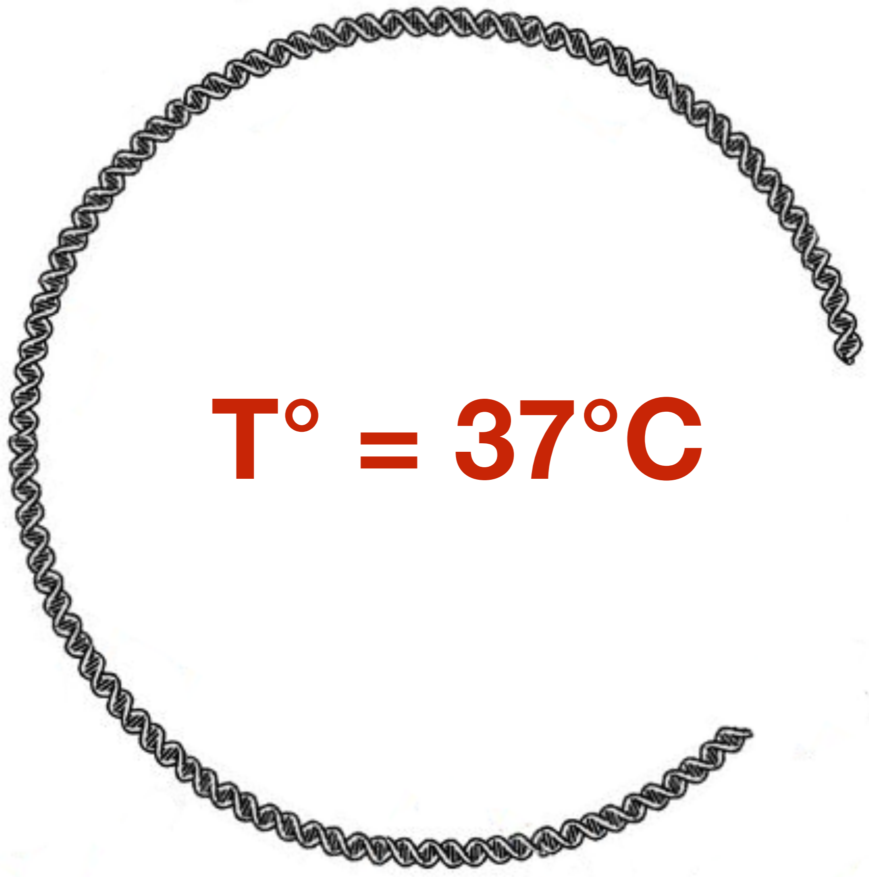
Wei, Dai, Yin, Nature 2013

Co-transcriptional folding

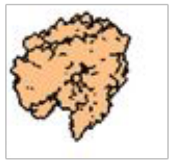


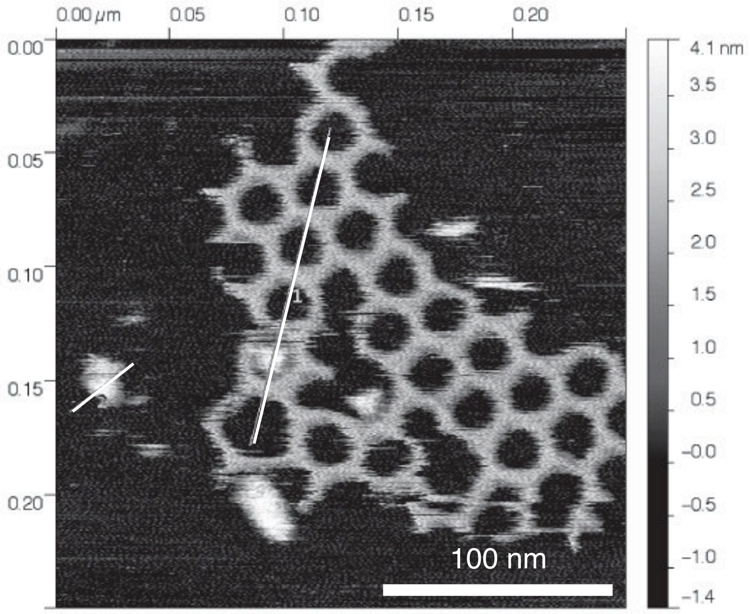
Geary, Rothemund, Andersen, Science 2014

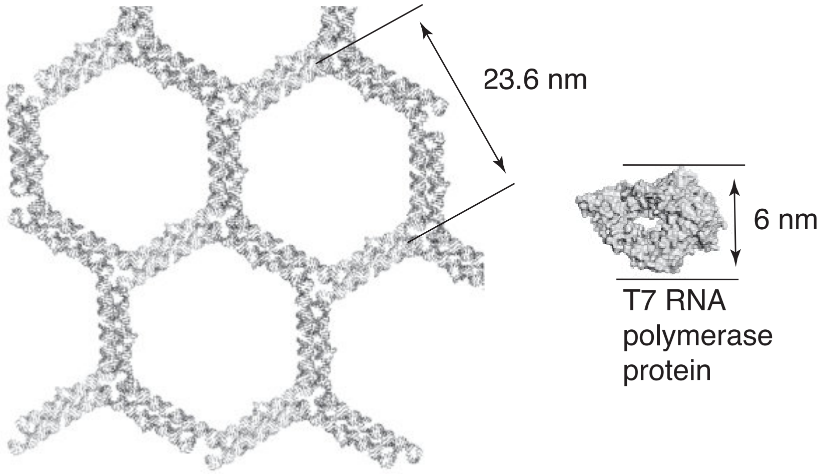
RNA
co-transcriptional folding



T° = 37°C

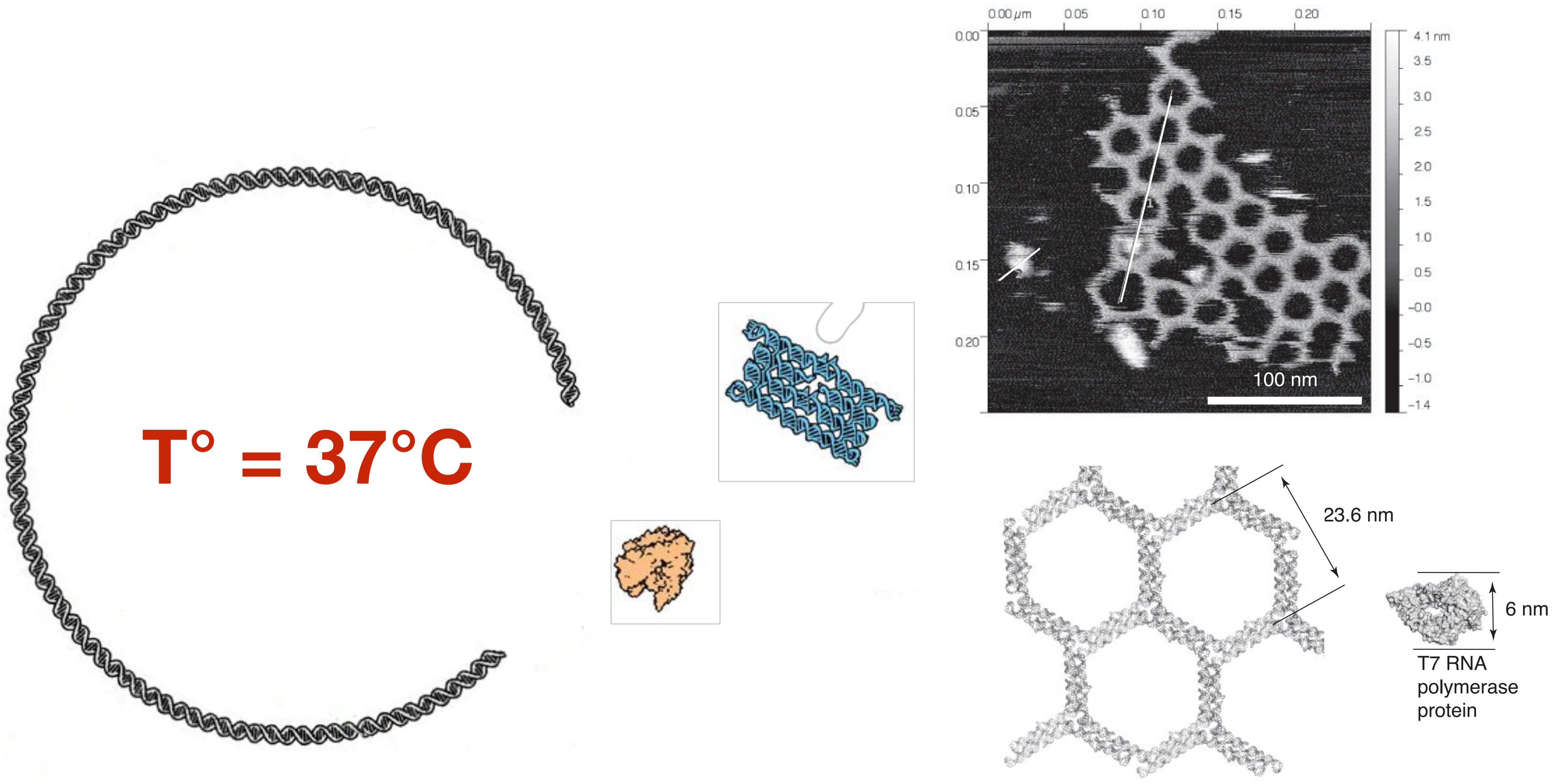






Geary, Rothemund, Andersen, Science 2014

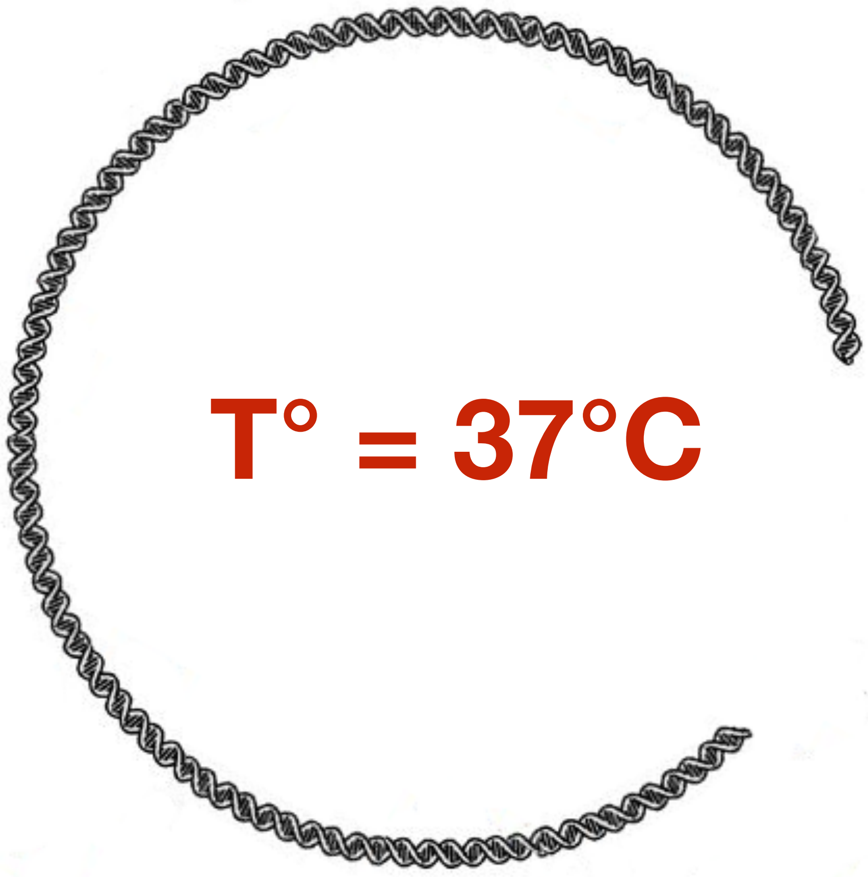
RNA co-transcriptional folding



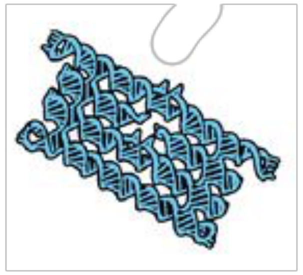
T° = 37°C

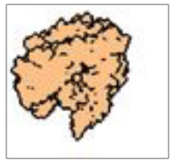
Geary, Rothemund, Andersen, Science 2014

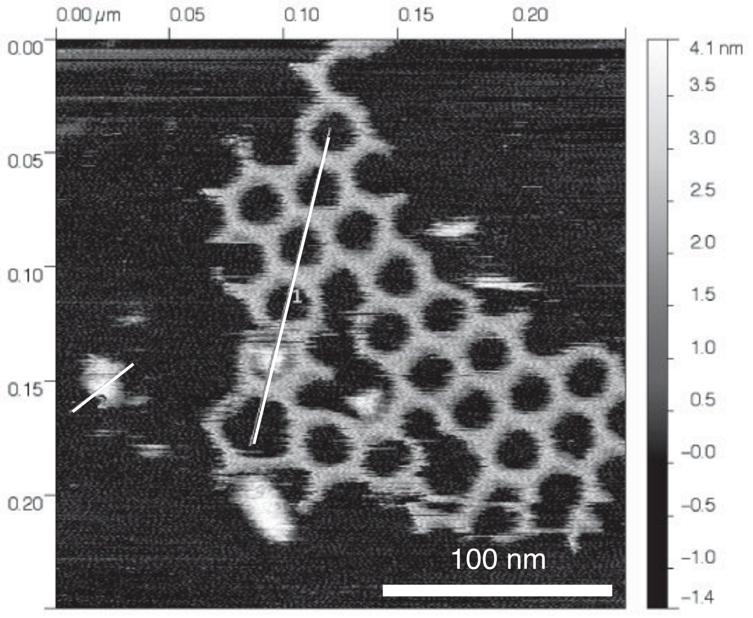
RNA co-transcriptional folding

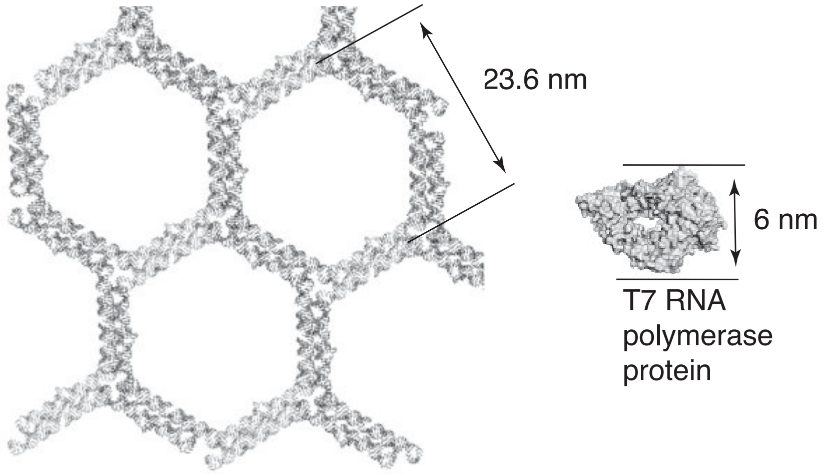


T° = 37°C



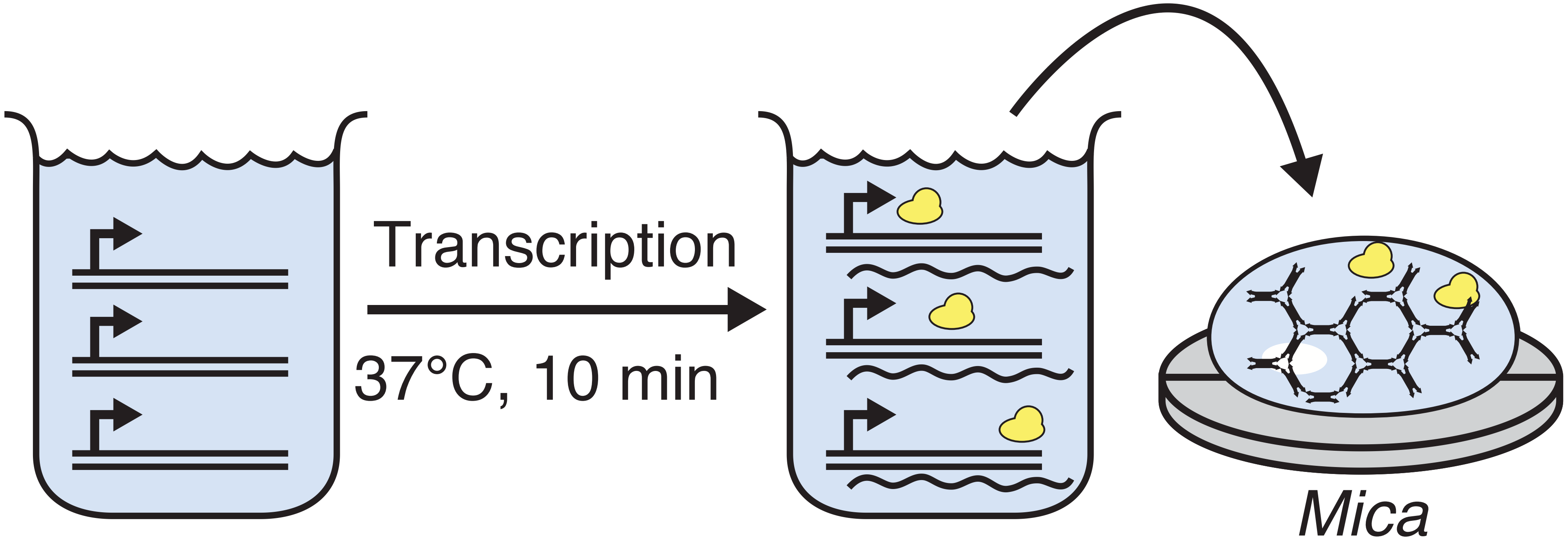




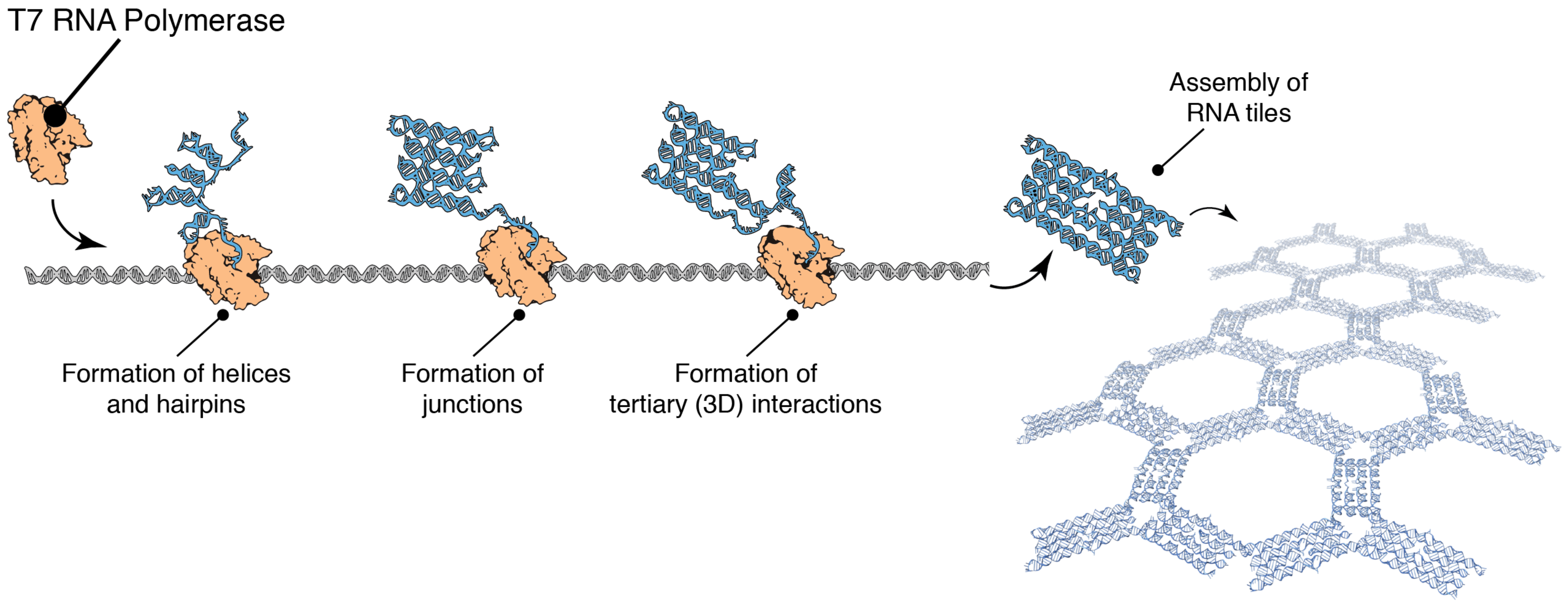


Geary, Rothemund, Andersen, Science 2014

Protocol



RNA Origami in Real Time


T7 RNA Polymerase
Assembly of RNA tiles
Formation of helices and hairpins
Formation of junctions
Formation of tertiary (3D) interactions

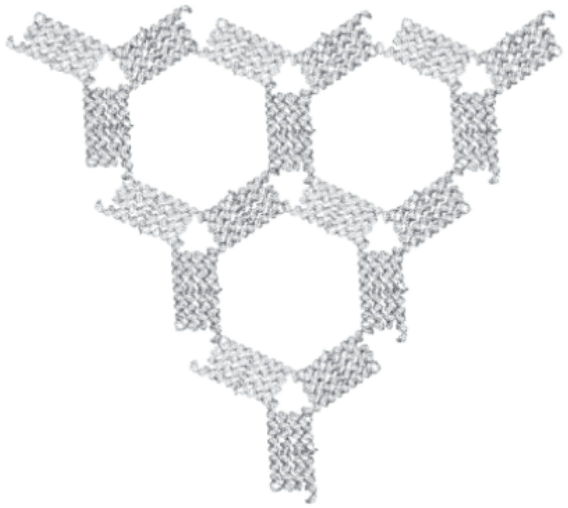
T7 RNA polymerase produces RNA directionally from 5’ to 3’, **at a rate much slower than the RNA folds up (few microseconds).**

The polymerase reads the DNA gene, and becomes an RNA origami production factory, **synthesizing a new RNA origami roughly every 1 second.**

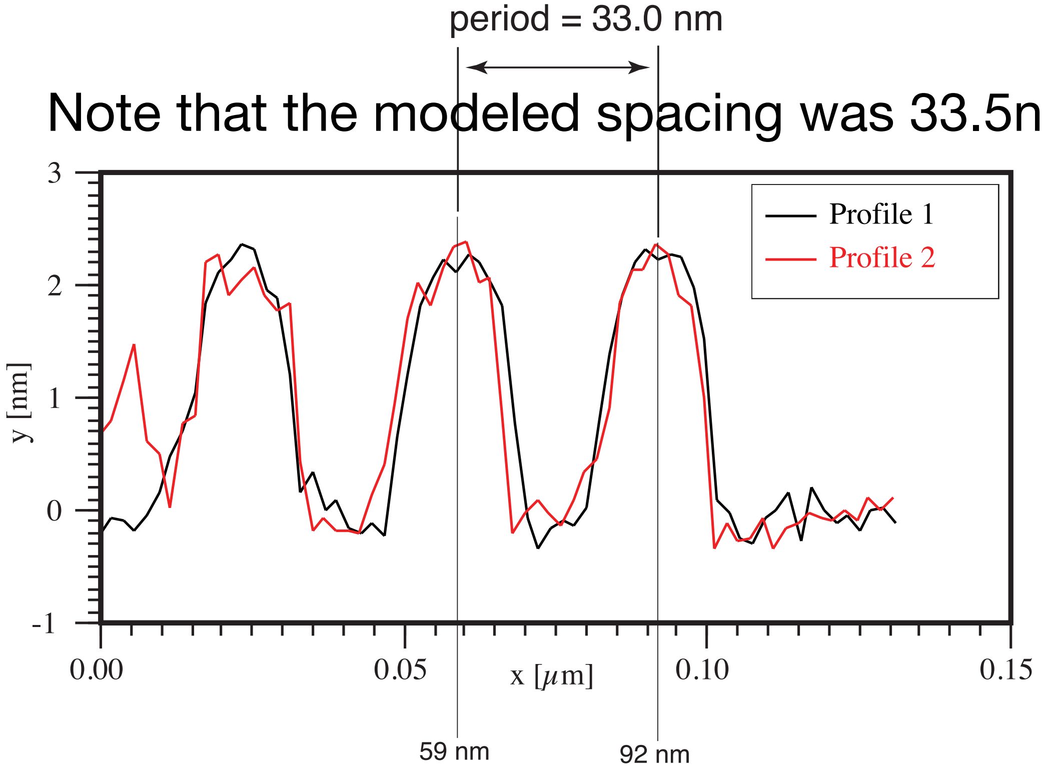
Slide by Cody Geary

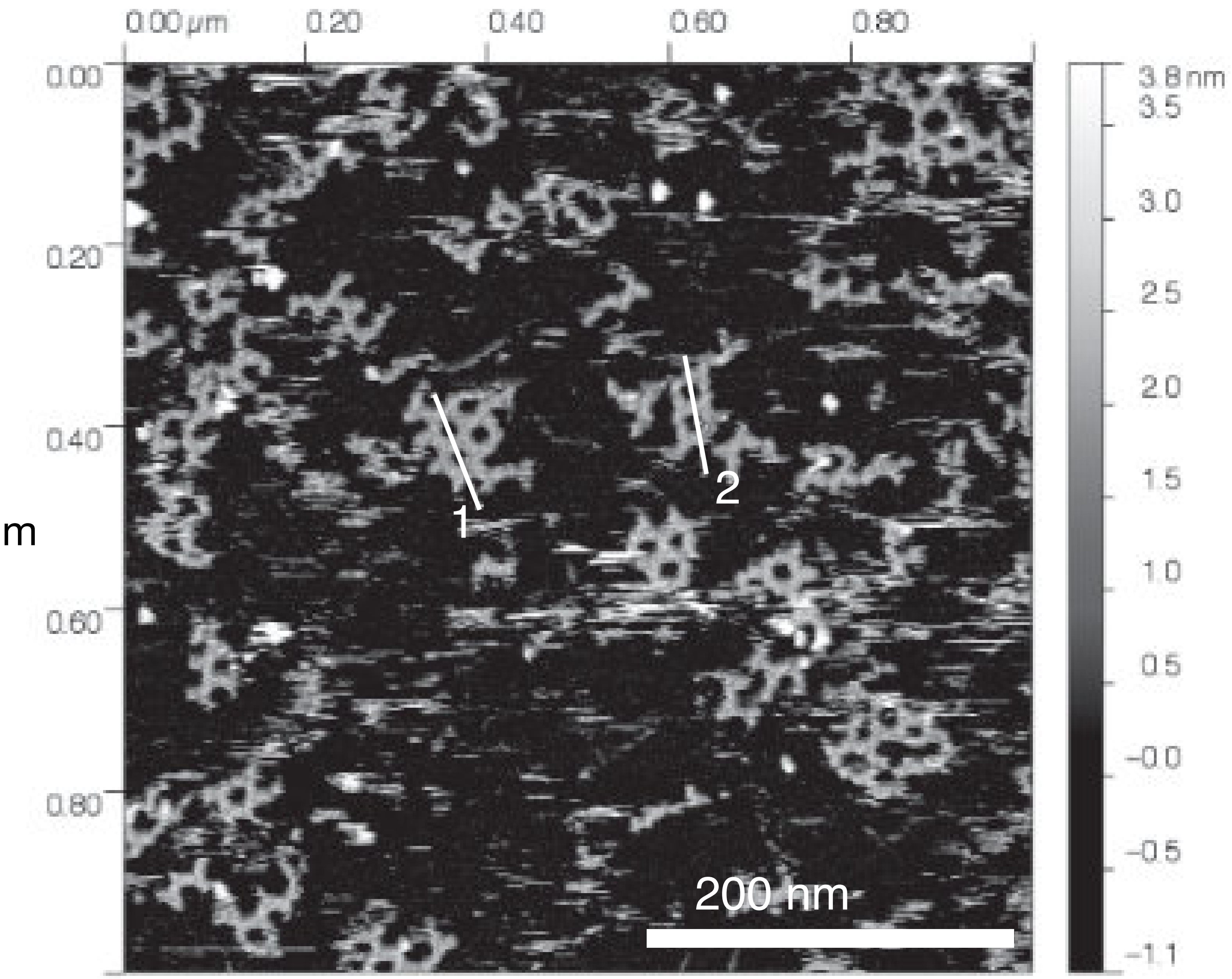
Westhof and Leontis (Science, 2014)

AFM imaging of 4H-AE co-transcriptional assembly



Note that the modeled spacing was 33.5nm

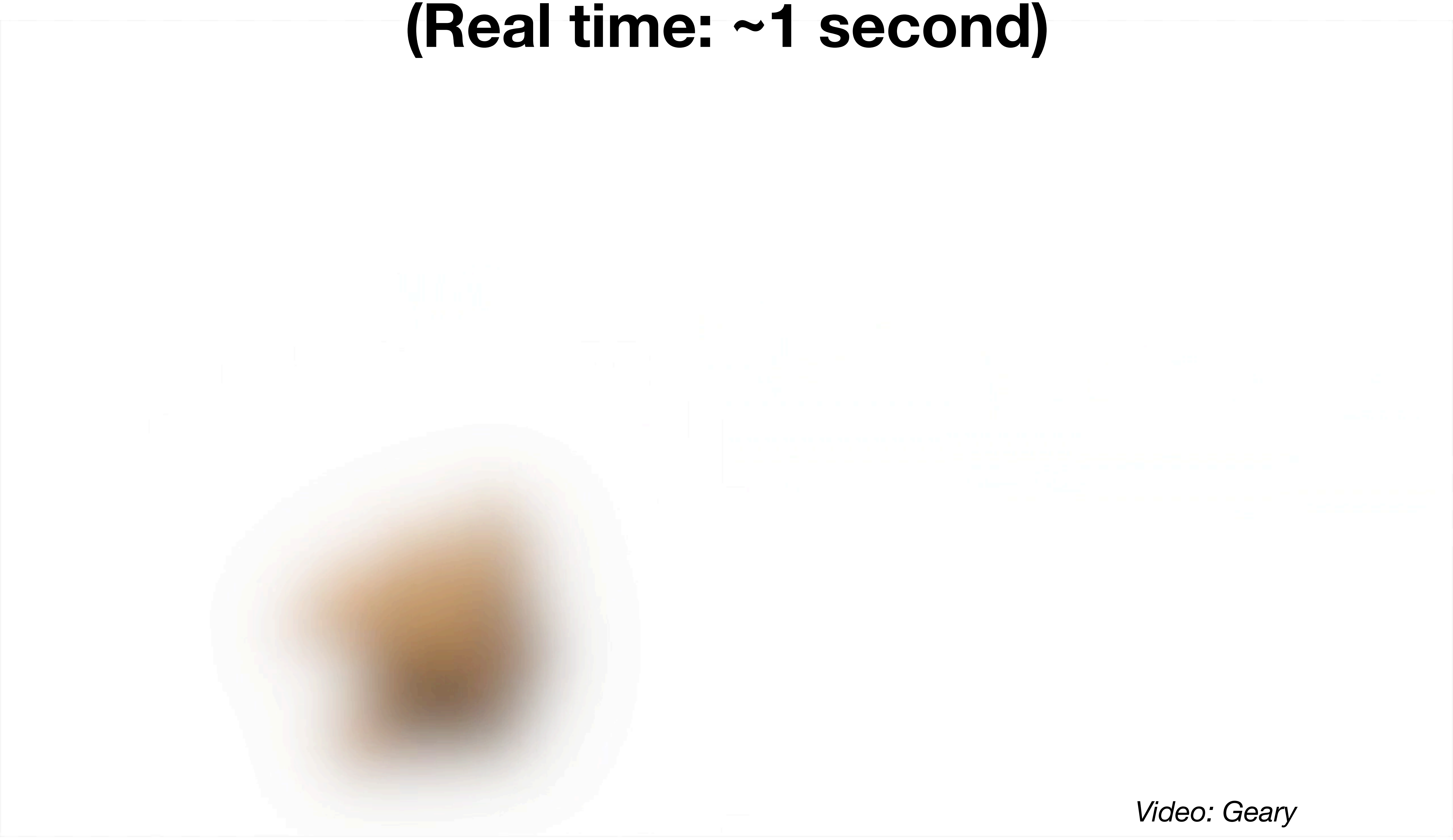




Geary et al (2014) Science

RNA Folding

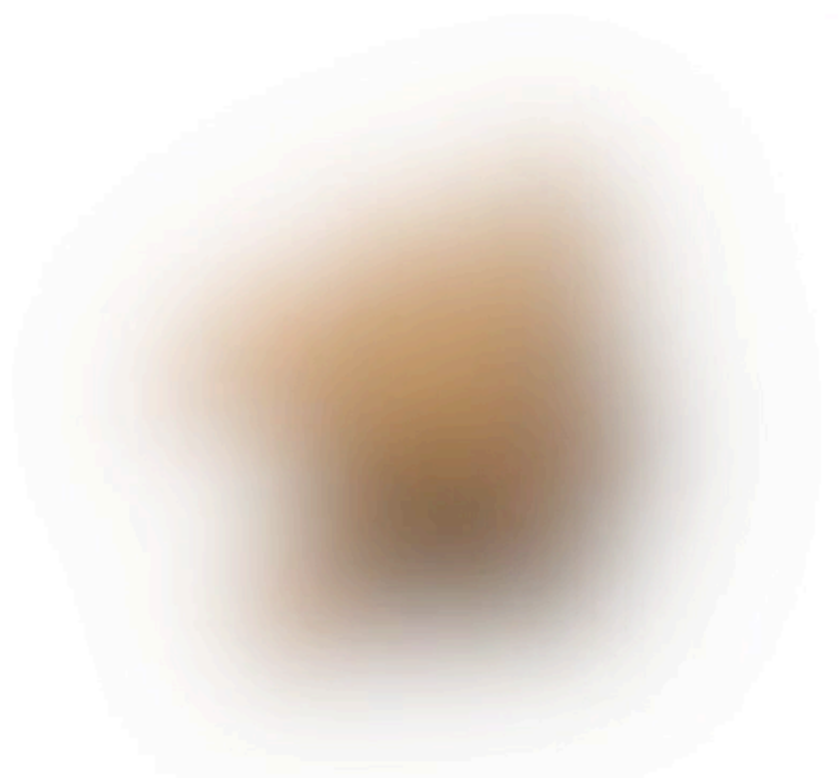
(Real time: ~1 second)



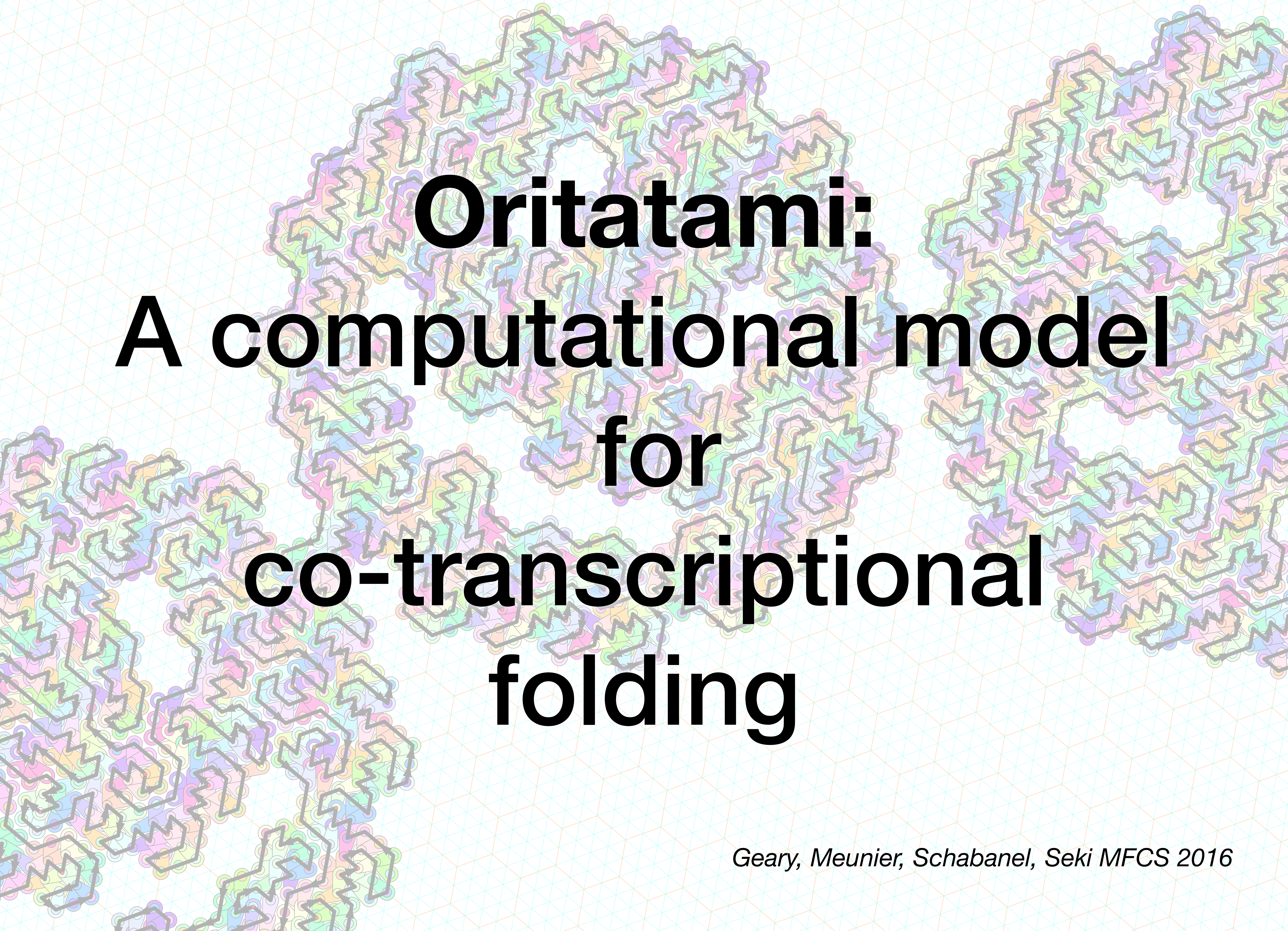
Video: Geary

RNA Folding

(Real time: ~1 second)



Video: Geary



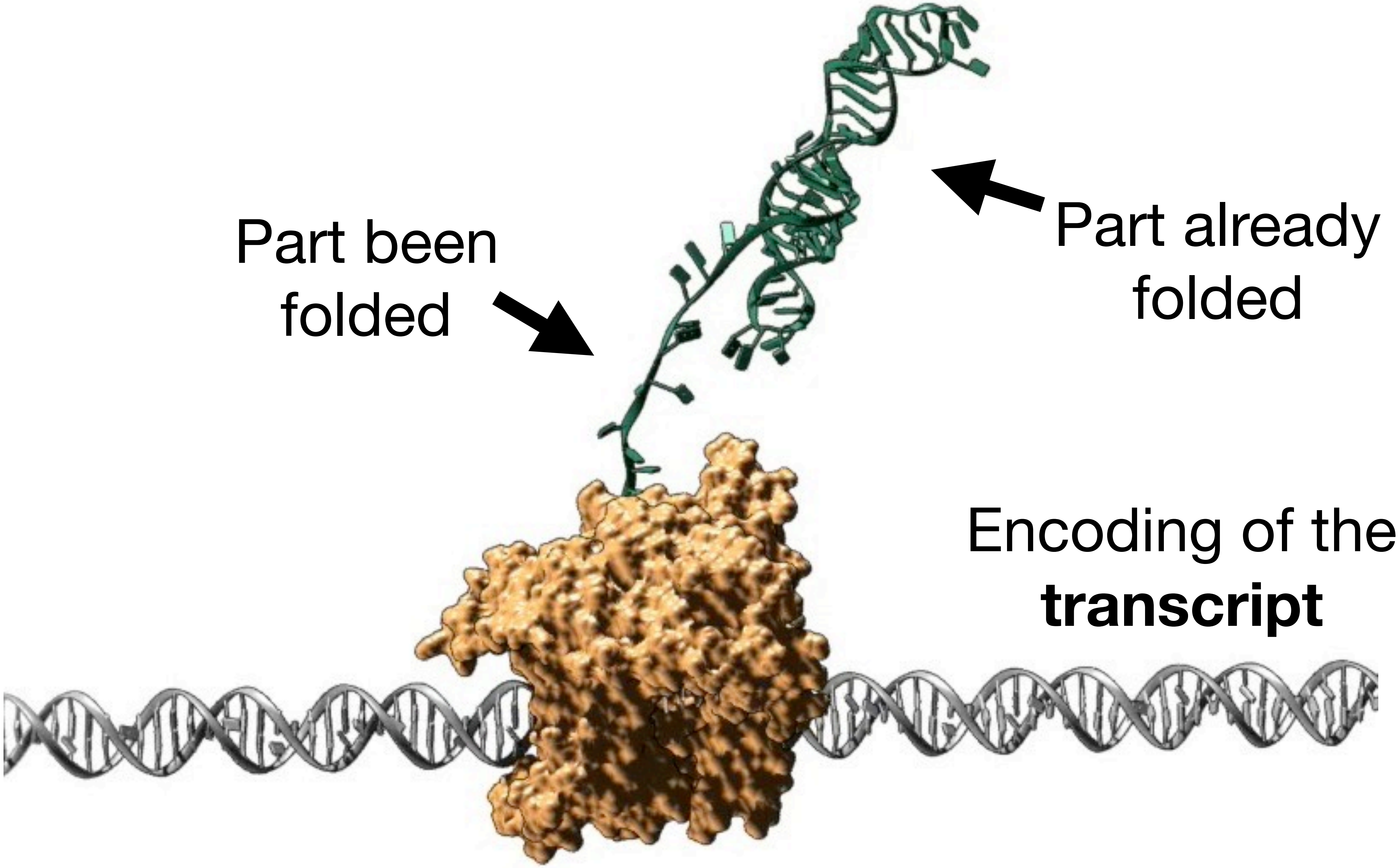
Oritatami:

A computational model for co-transcriptional folding

Geary, Meunier, Schabanel, Seki MFCS 2016

RNA Folding

(Real time: ~1 second)



Part been folded

Part already folded

Encoding of the **transcript**

Video: Geary

Oritatami:
A model for co-transcriptional folding

The program:

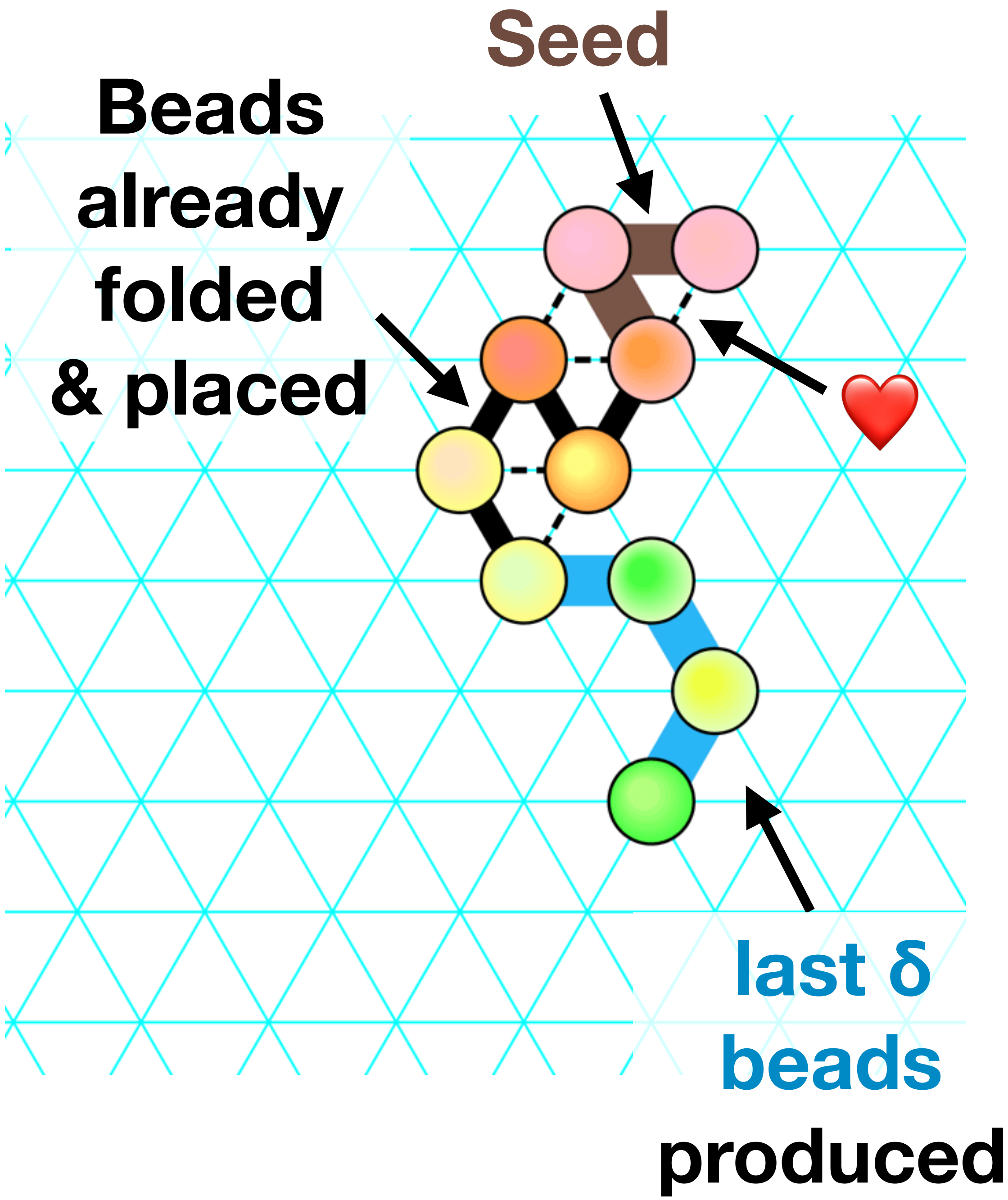
- a **sequence** of **bead types** (the **transcript**)

The instructions:

- the **rule** **a**❤️**b** if bead types **a** and **b** attract each other

The input configuration:

- Some beads placed beforehand (the **seed**)



Seed

Beads already folded & placed

❤️

last δ beads produced

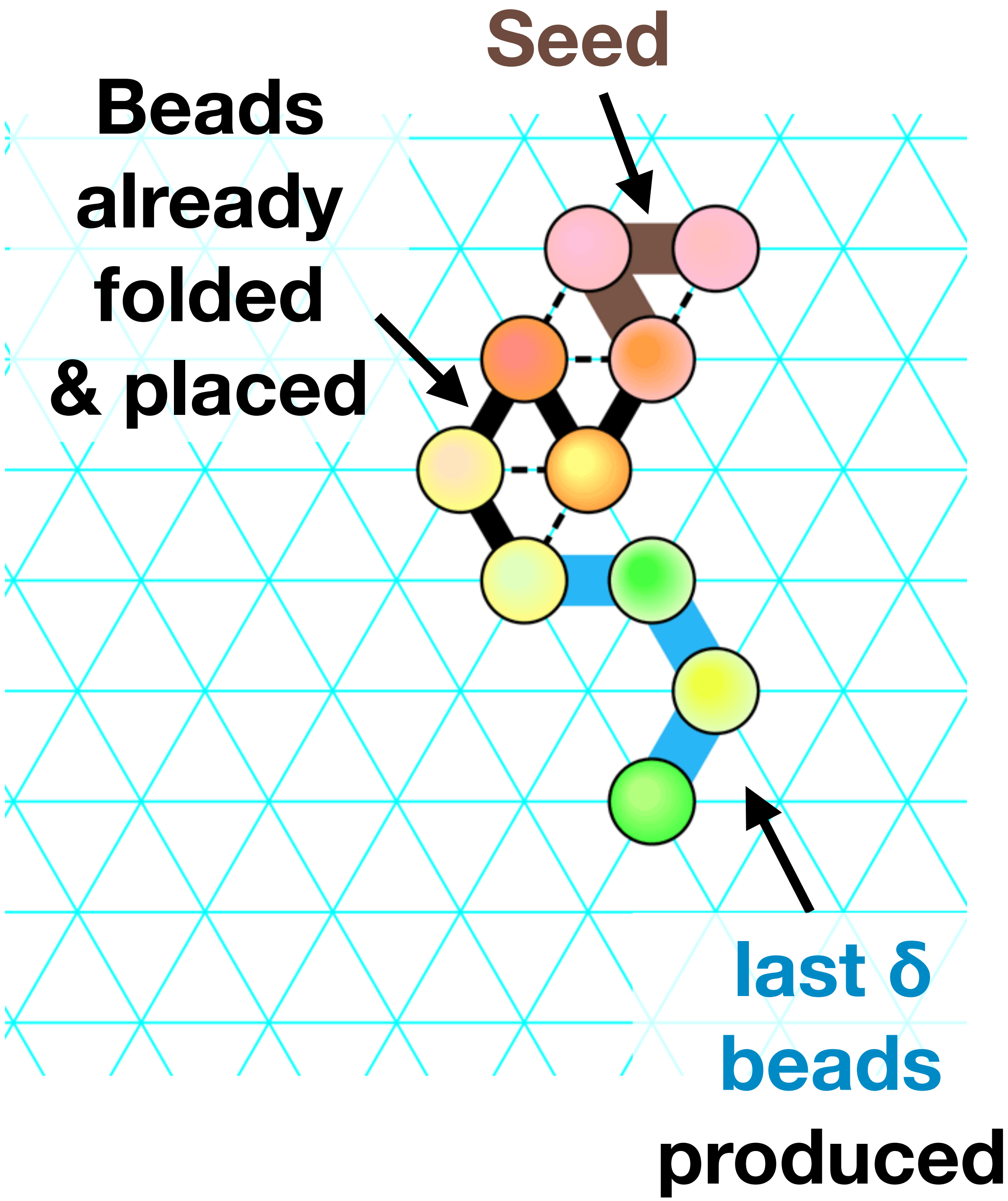
Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations

here, delay **δ = 3**



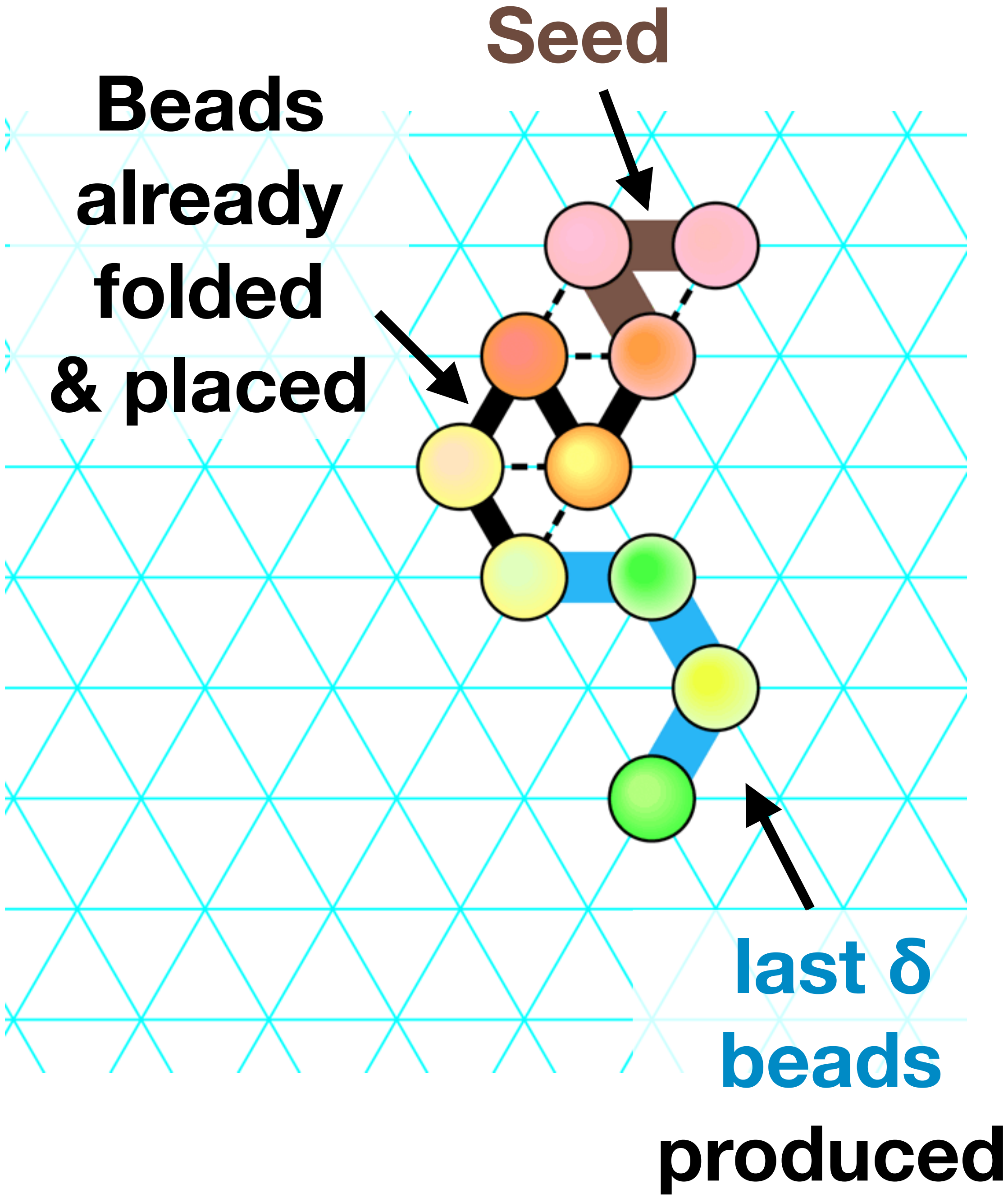
Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations

here, delay **δ = 3**



Seed

Beads already folded & placed

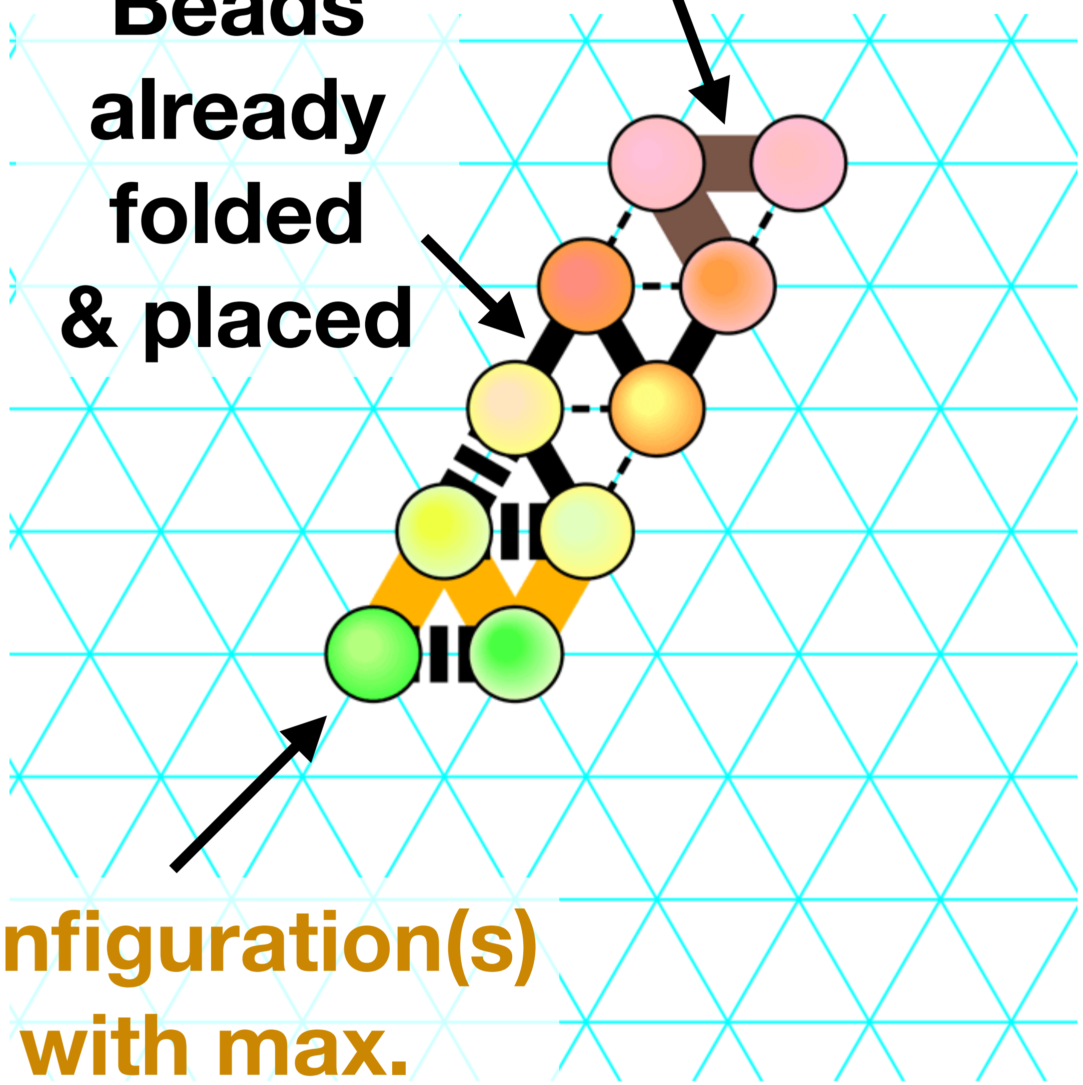
last δ beads produced

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations



Seed

Beads already folded & placed

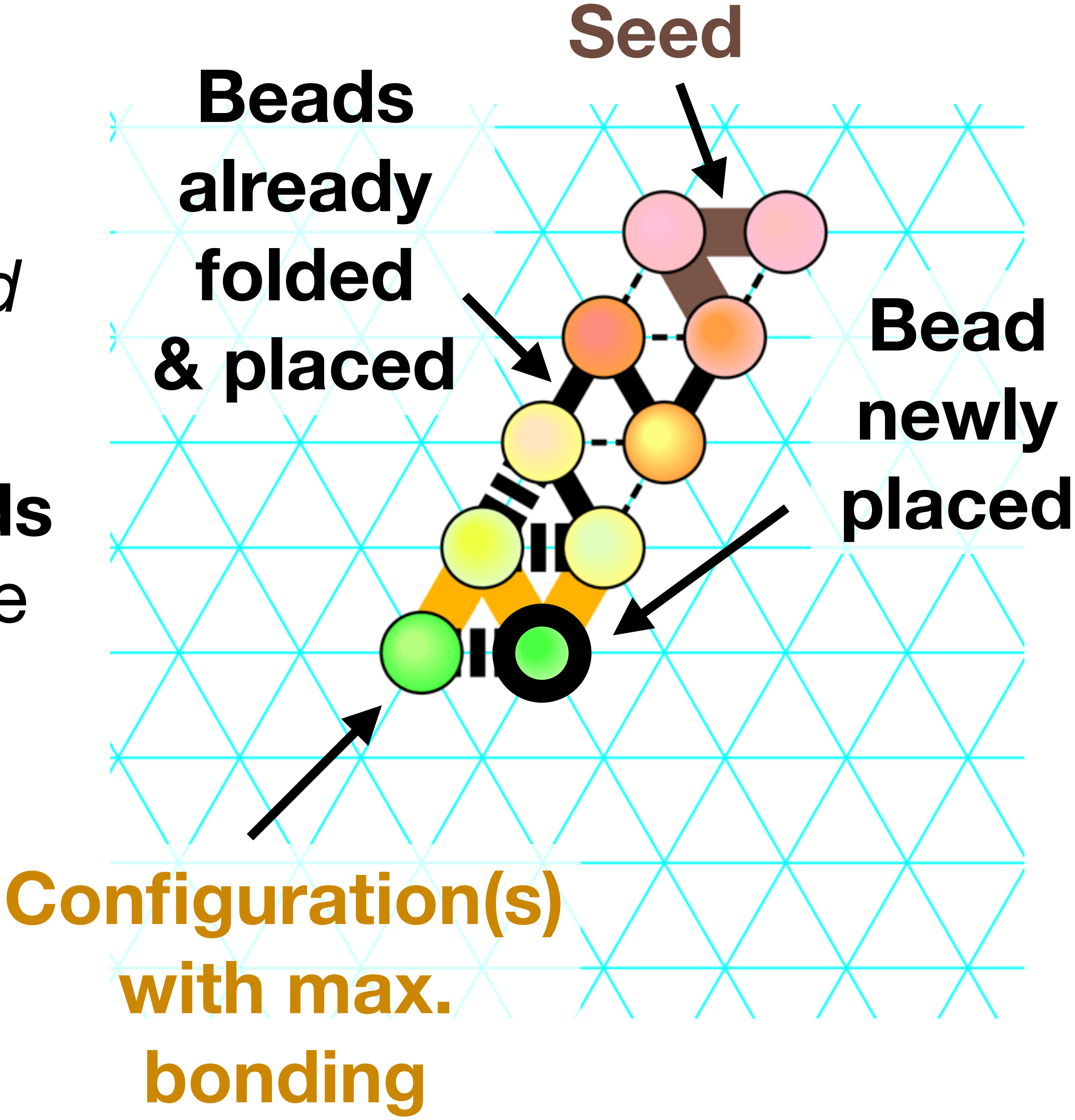
Configuration(s) with max. bonding

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations

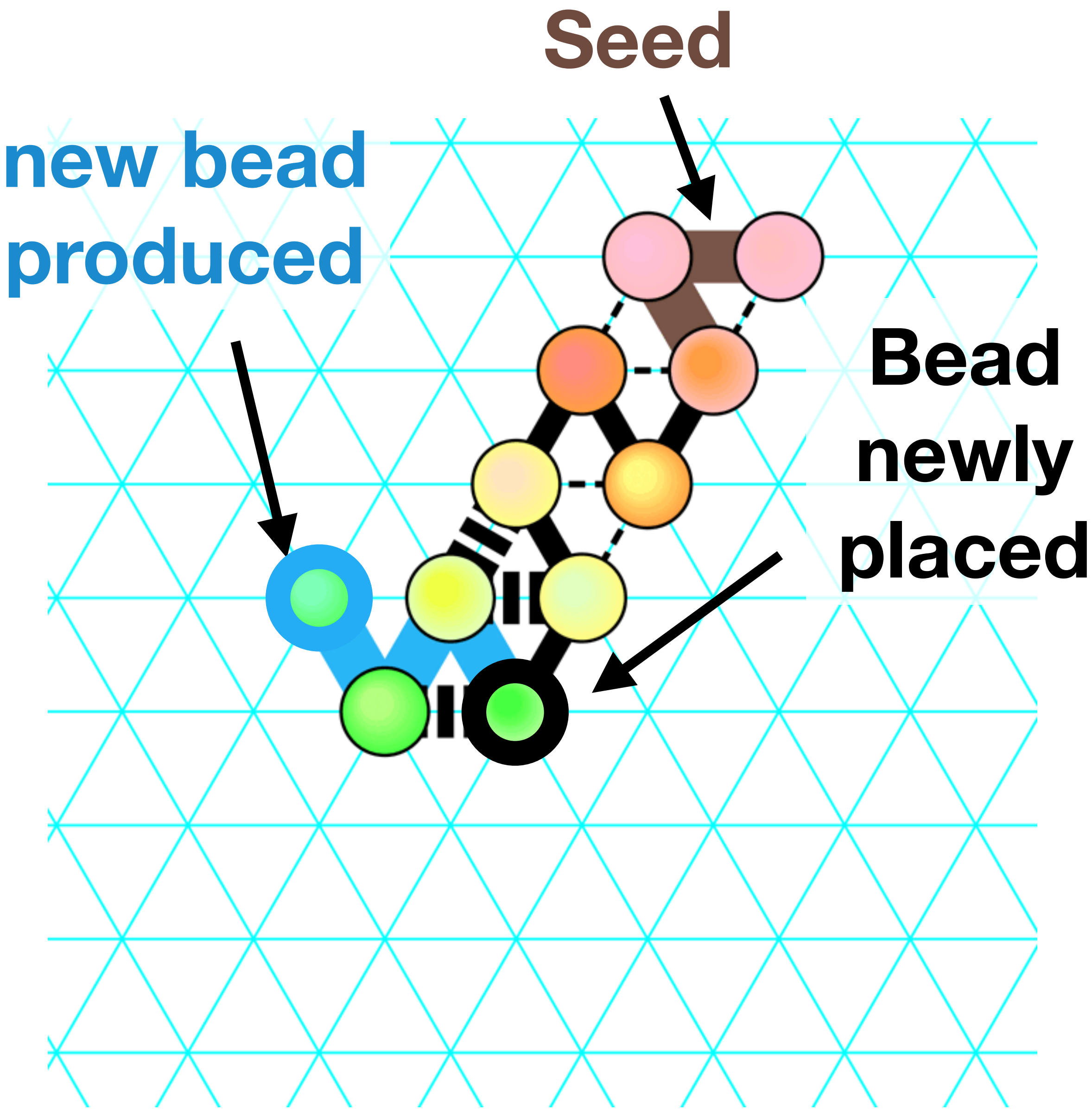

Seed
Beads already folded & placed
Bead newly placed
Configuration(s) with max. bonding

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations

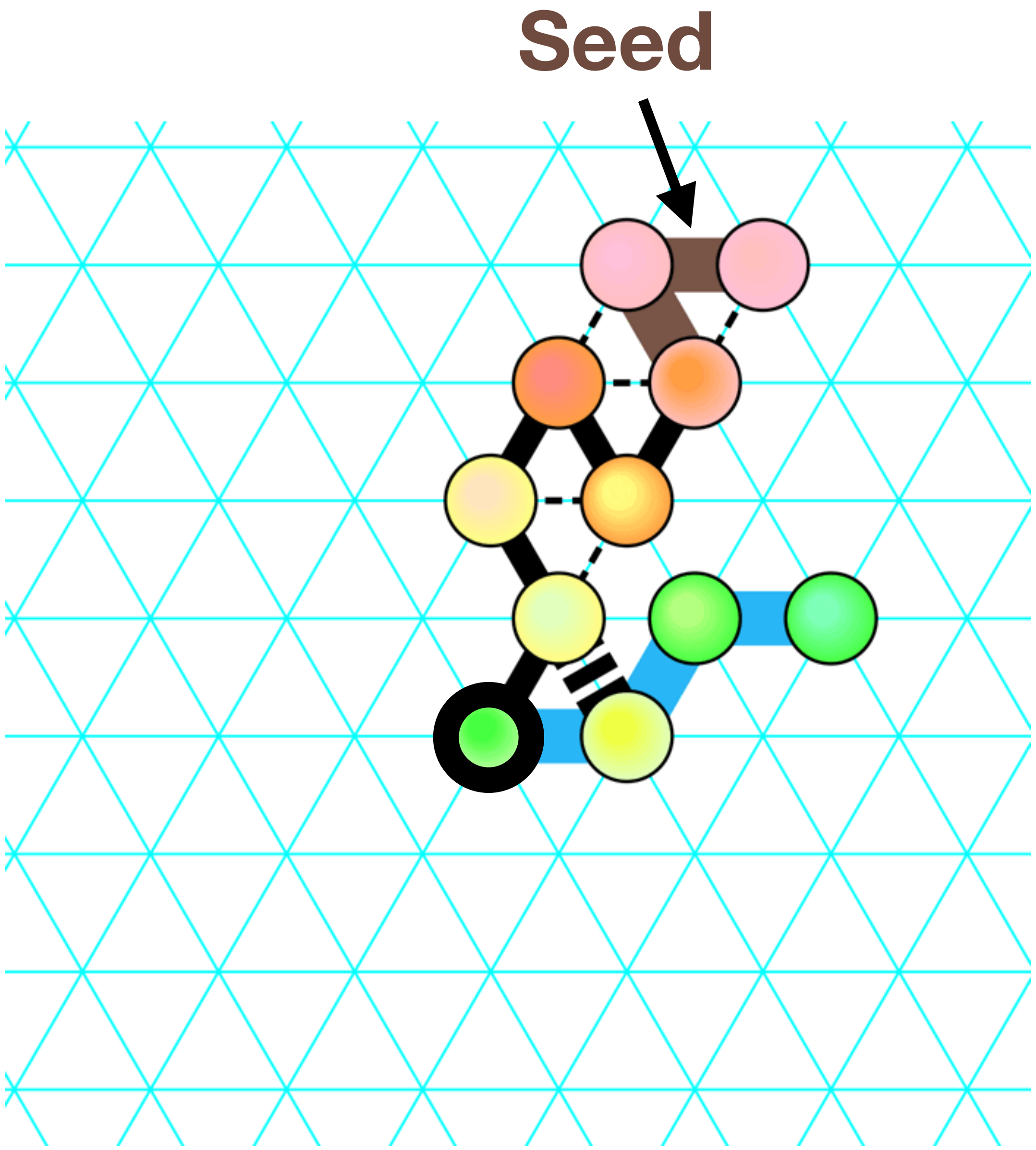


Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations

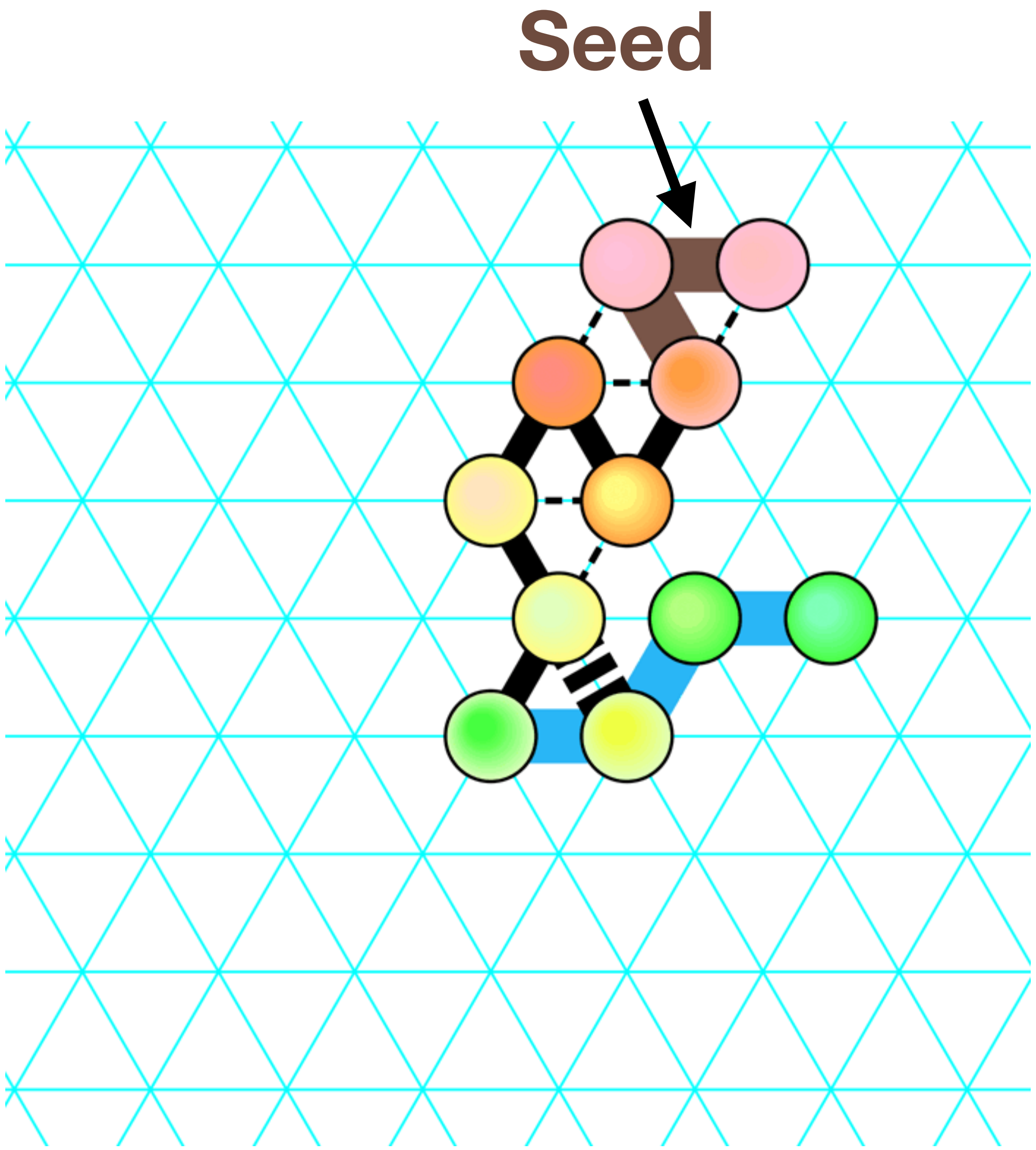


Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations

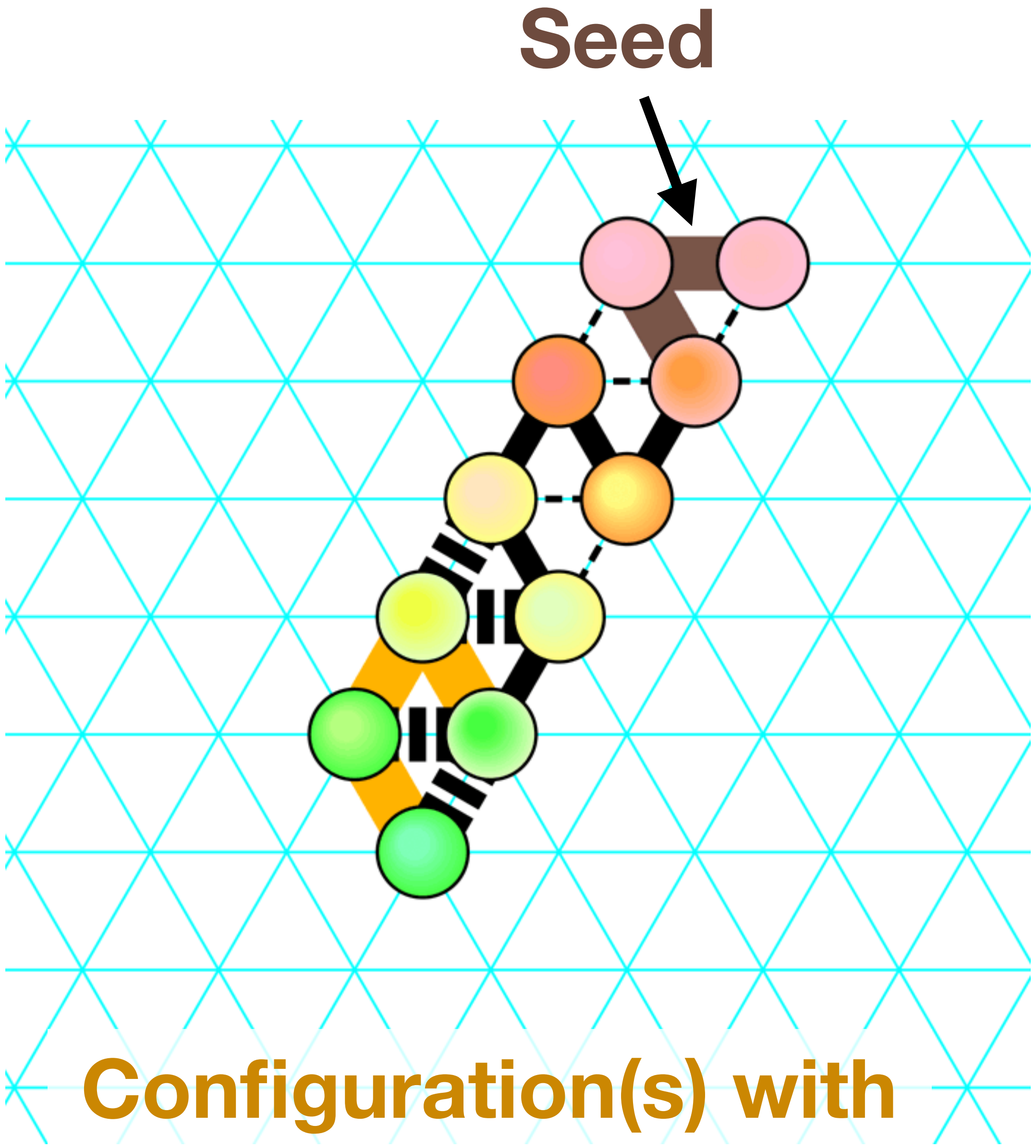


Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations



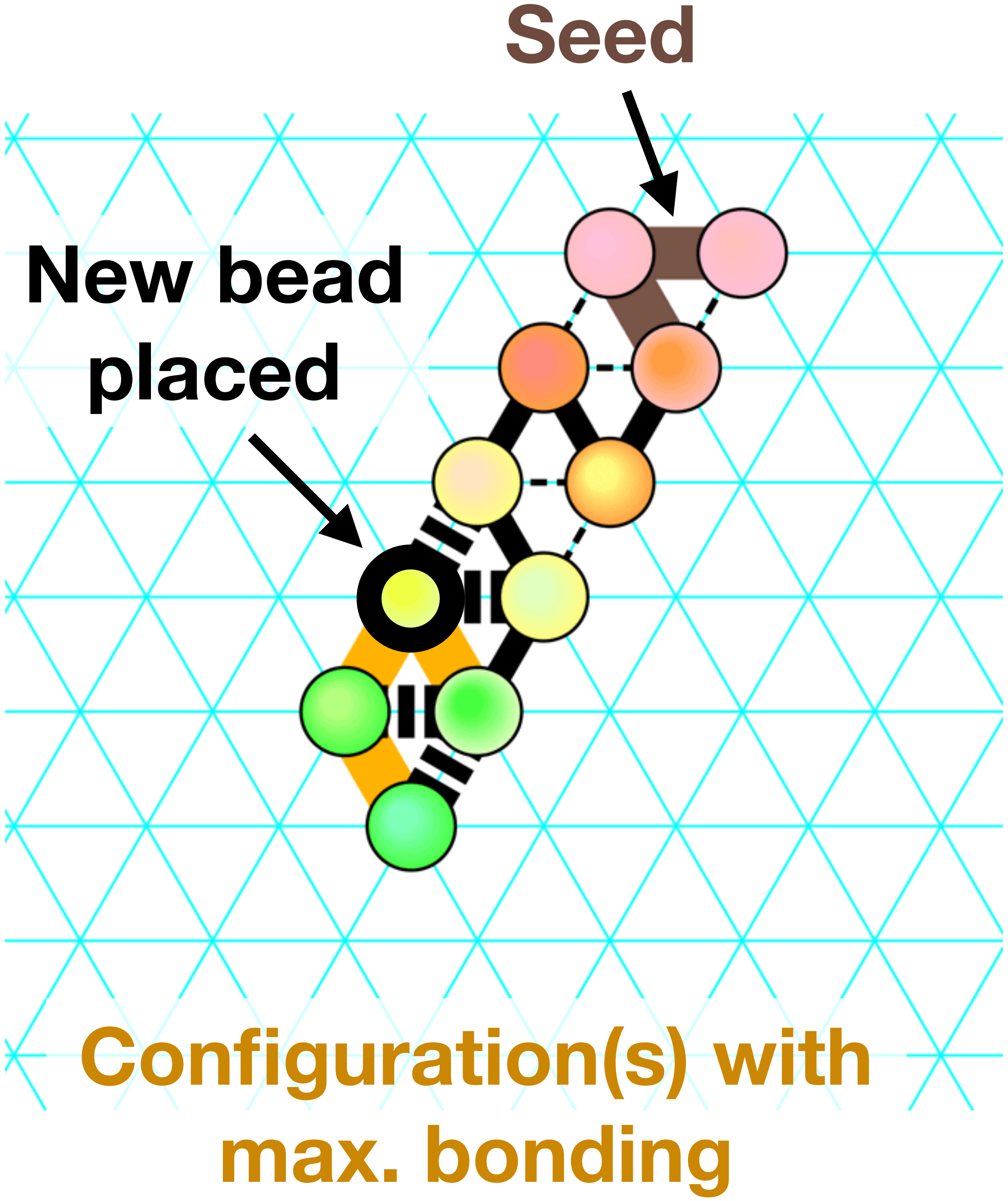
Configuration(s) with max. bonding

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations



Seed

New bead placed

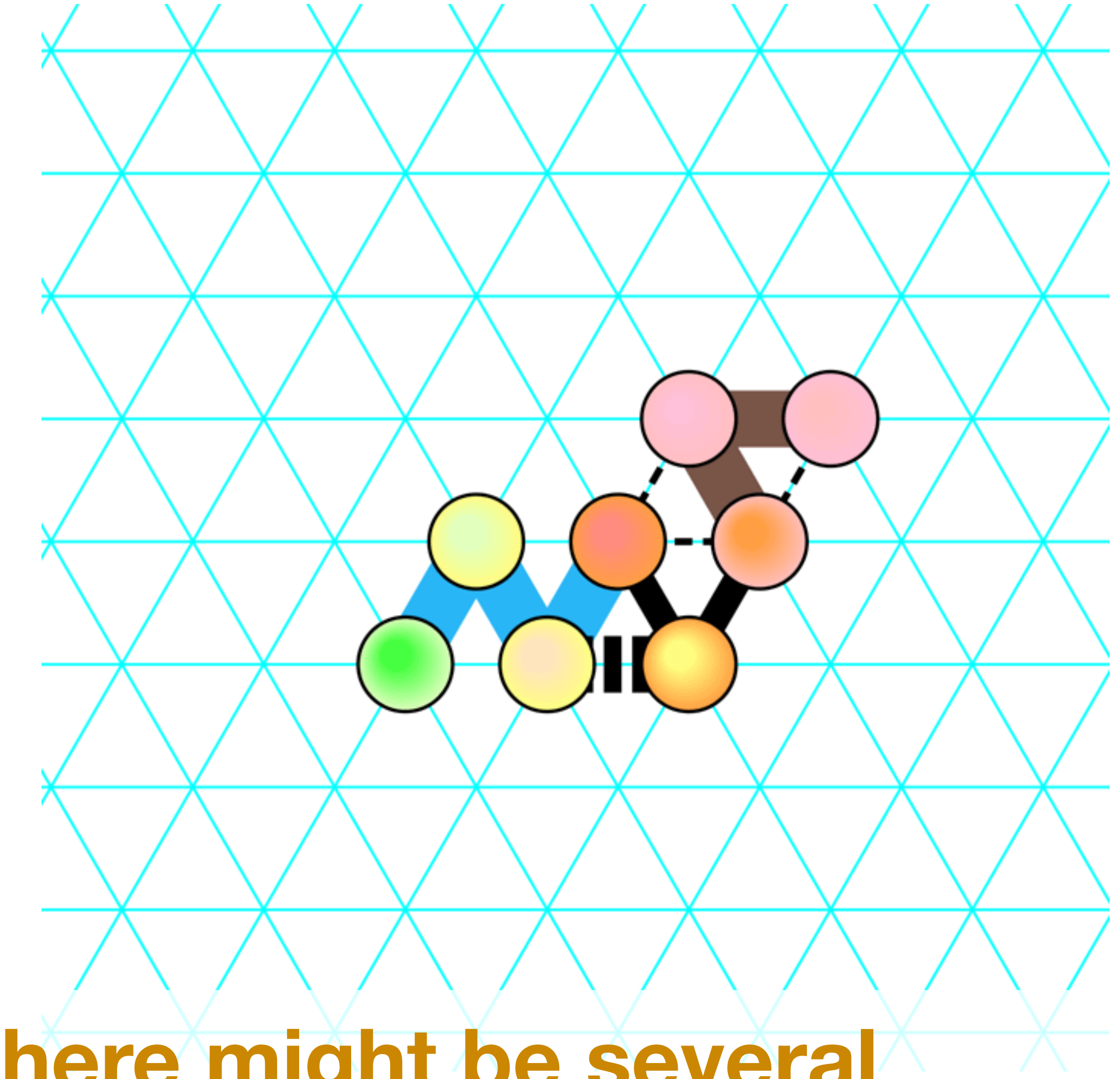
Configuration(s) with max. bonding

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations



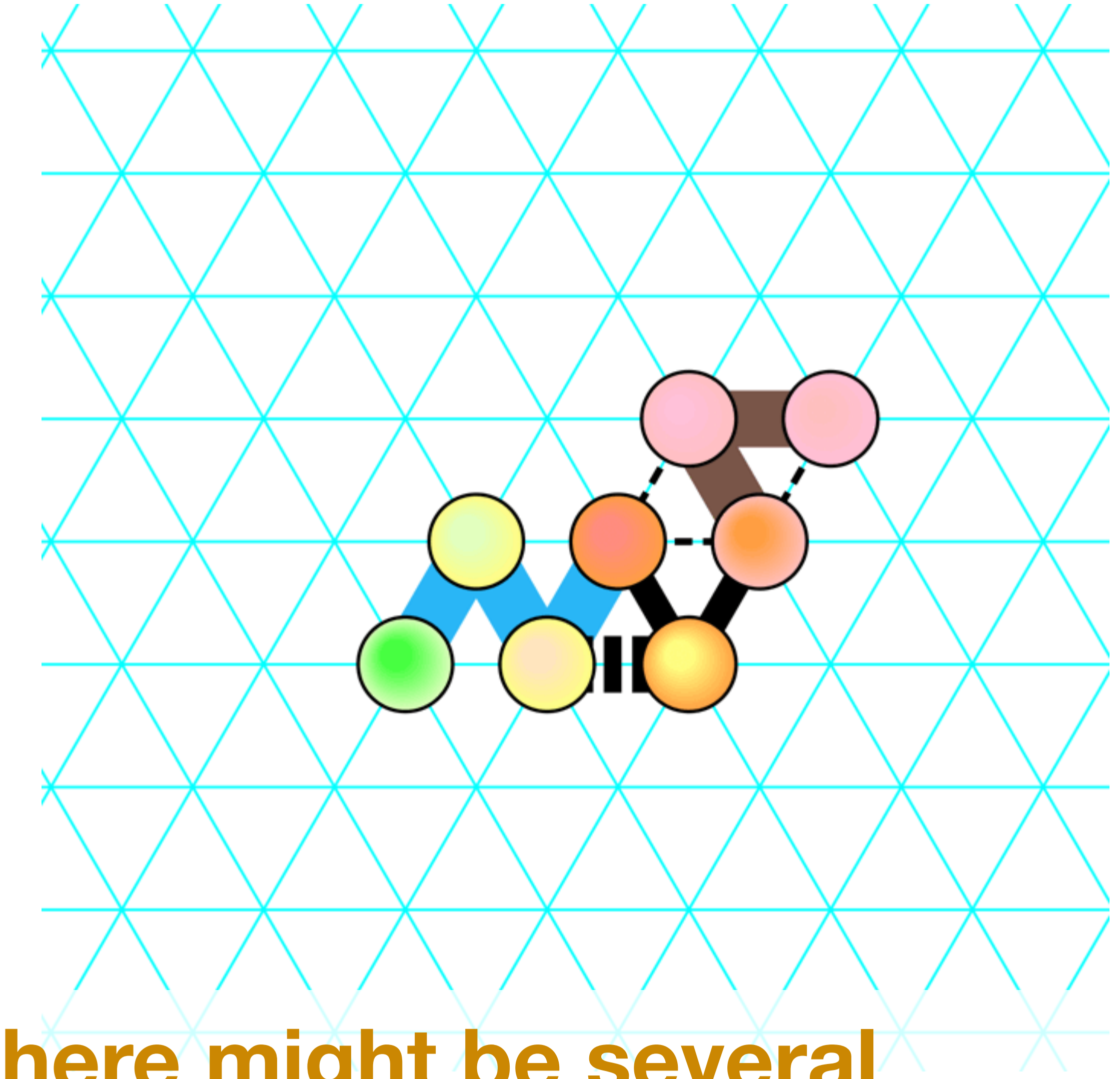
There might be several configurations with max. bonding

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations



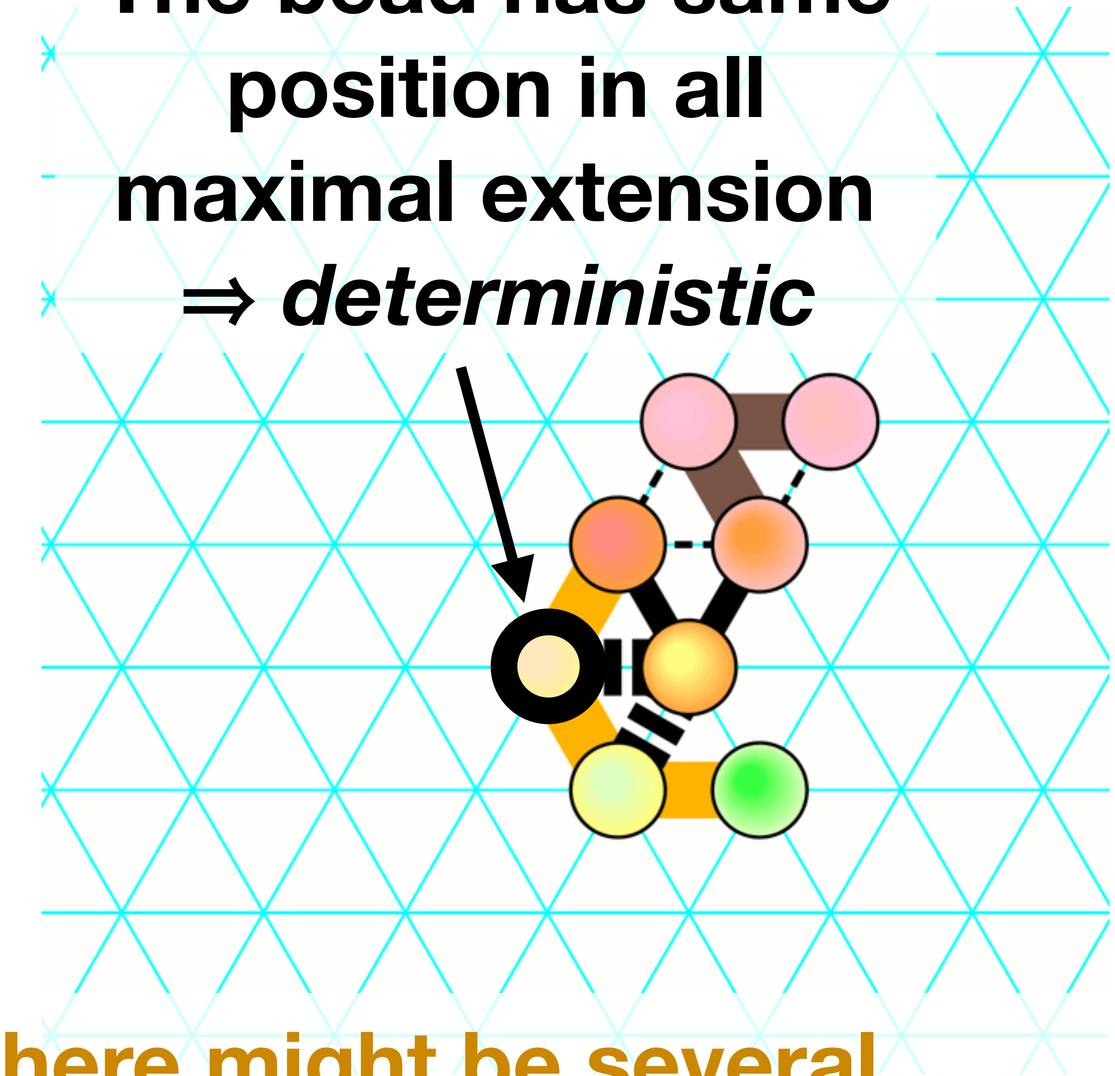
There might be several configurations with max. bonding

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations



The bead has same position in all maximal extension ⇒ *deterministic*

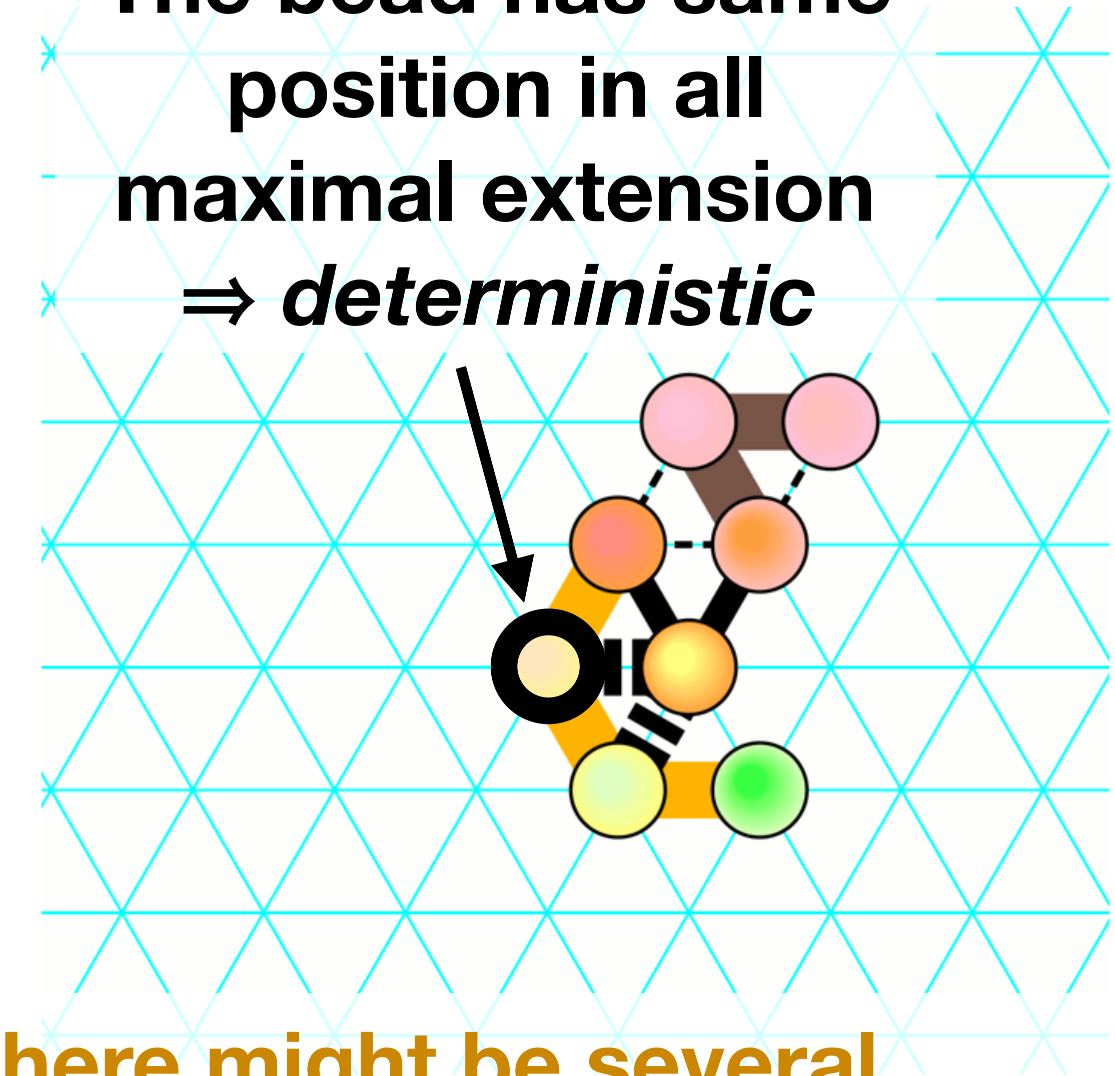
There might be <u>several</u> configurations with max. bonding

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami:
A model for co-transcriptional folding

The dynamics.

- Starting from the seed, the sequence is *produced one bead at a time*
- **Only the δ last produced beads** are free to move and explore the accessible positions to settle in the ones **maximizing the number of bonds**
- All other beads remain in their last locations



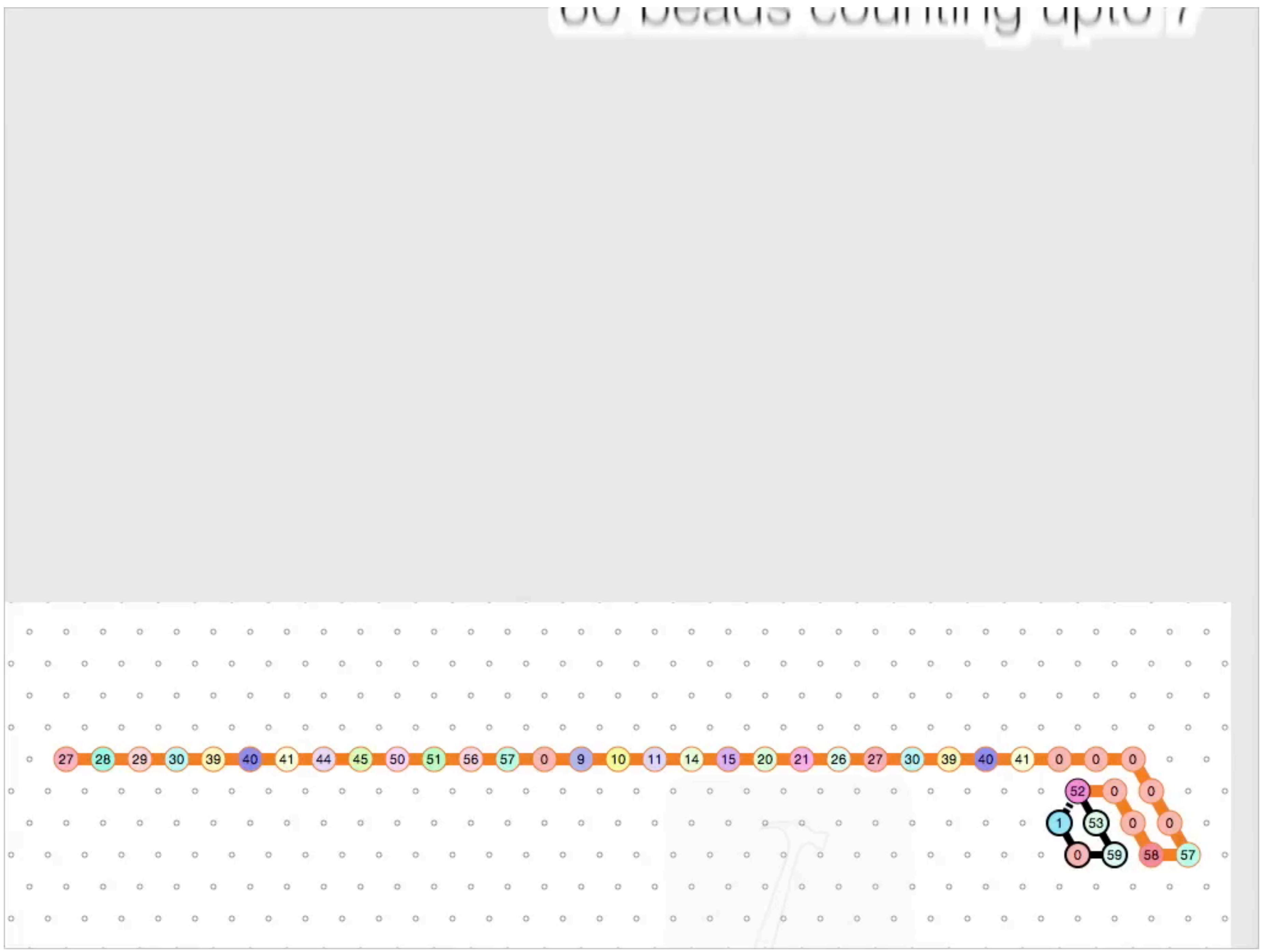
The bead has same position in all maximal extension ⇒ *deterministic*

There might be <u>several</u> configurations with max. bonding

Geary, Meunier, Schabanel, Seki MFCS 2016

Oritatami

A first example

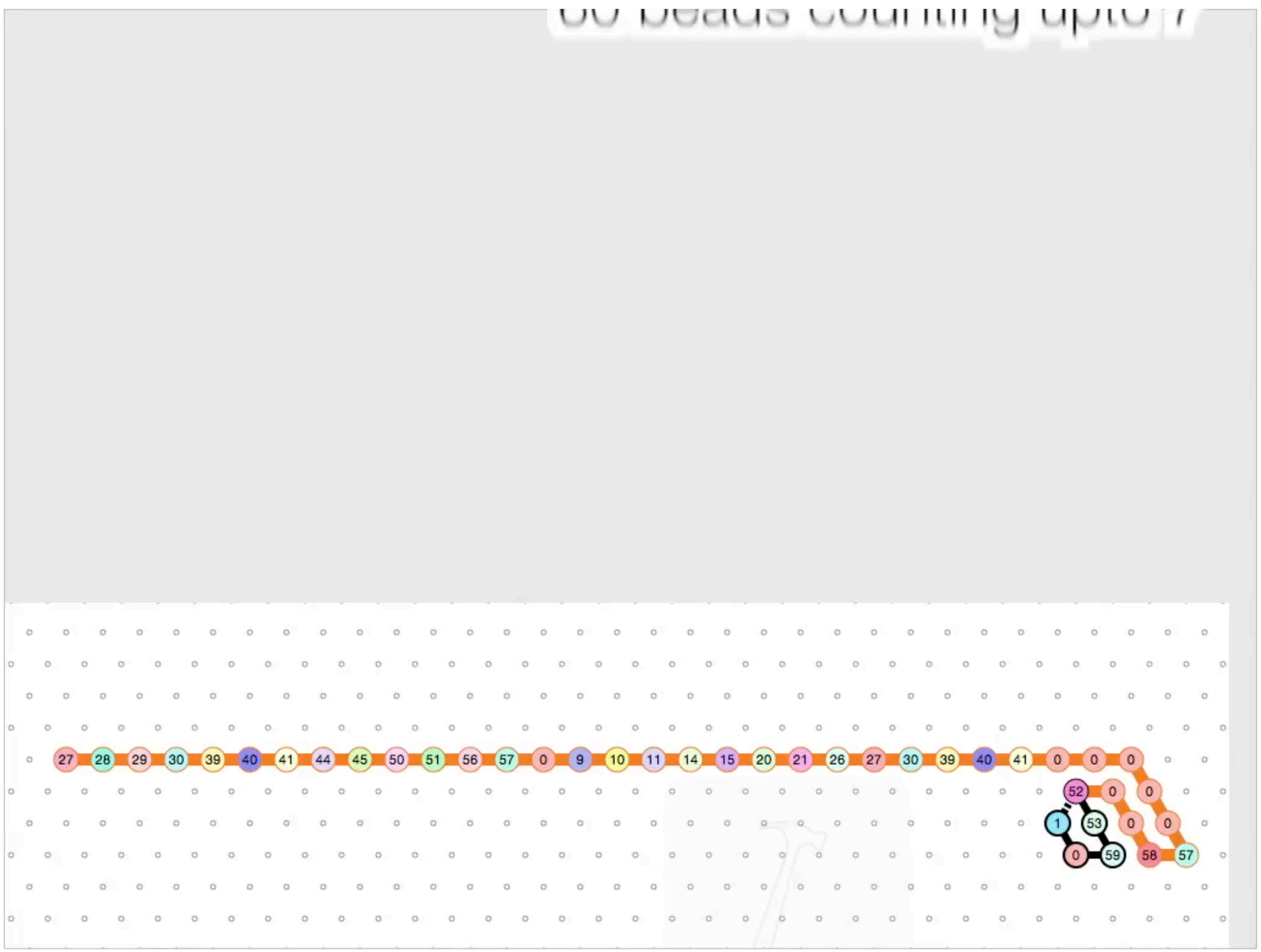


A binary counter made of a **single** 60-beads periodic molecule folding upon itself

Geary Meunier Seki Schabanel 2015

Oritatami

A first example

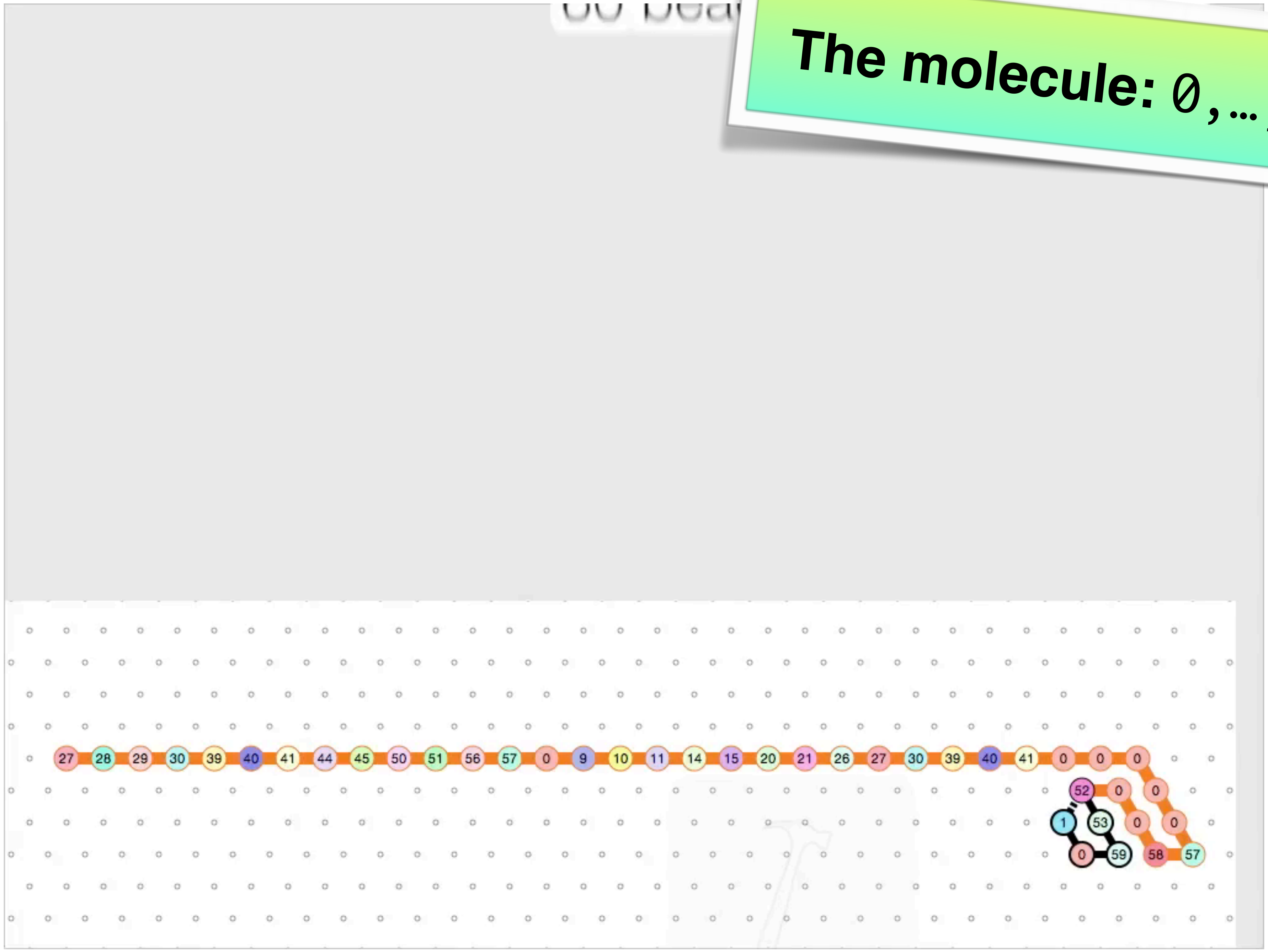


A binary counter made of a **single** 60-beads periodic molecule folding upon itself

Geary Meunier Seki Schabanel 2015

Oritatami

A first example



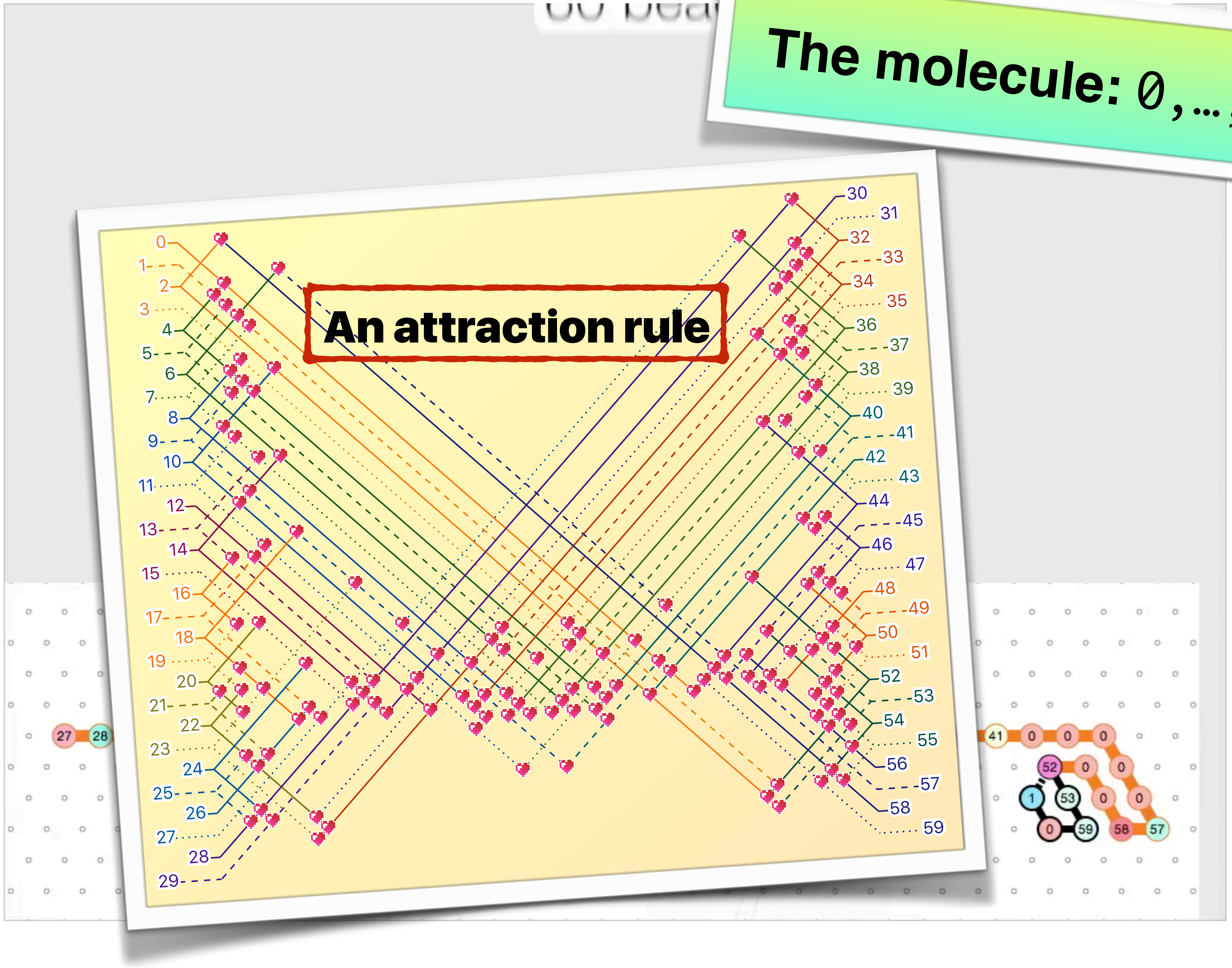
The molecule: `0,…,59,0,…,59,0,…`

of a **single** 60-
beads
periodic
molecule
folding upon
itself

Geary Meunier Seki
Schabanel 2015

Oritatami

A first example



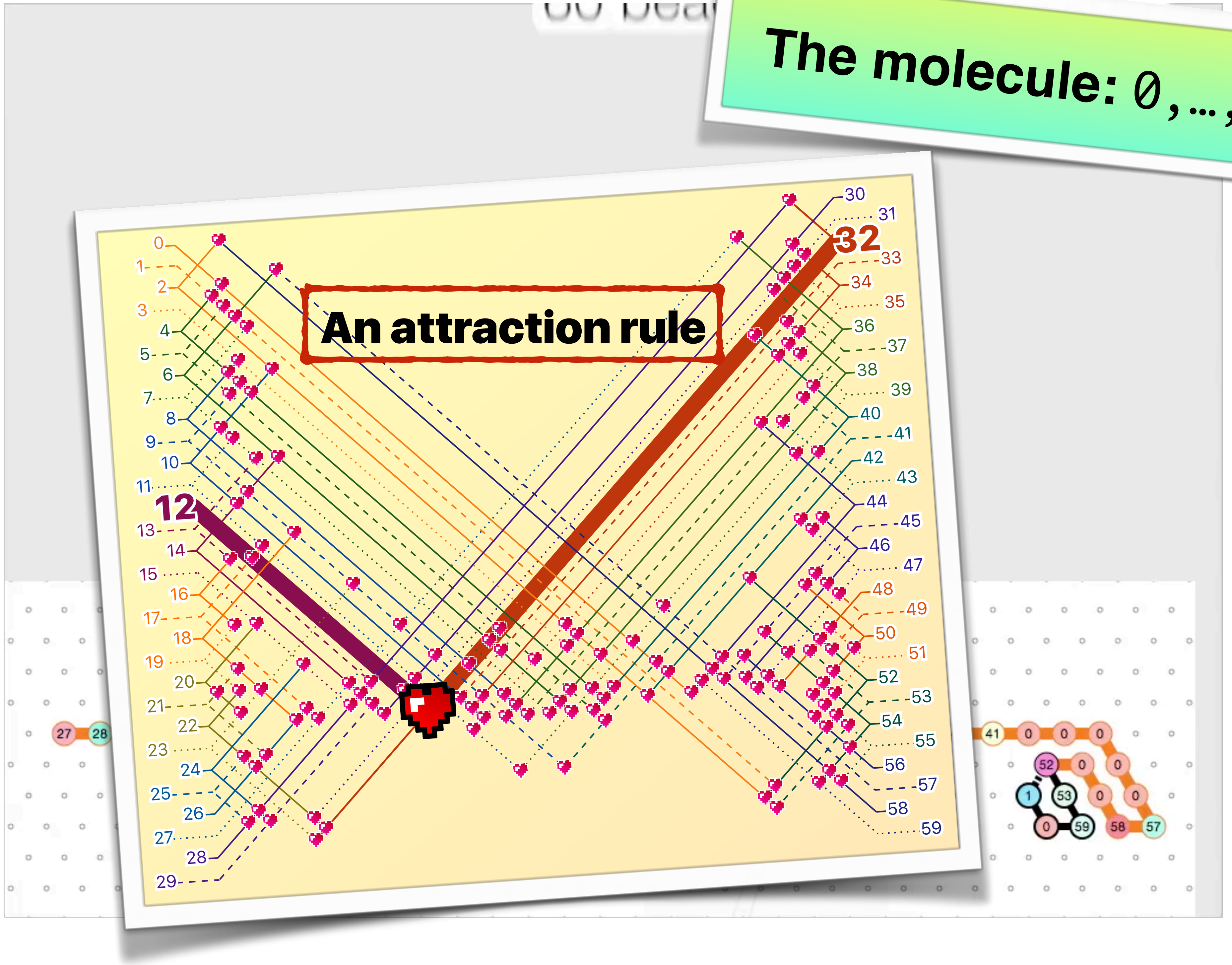
The molecule: `0,…,59,0,…,59,0,…`

of a **single** 60-beads periodic molecule folding upon itself

Geary Meunier Seki Schabanel 2015

Oritatami

A first example



An attraction rule

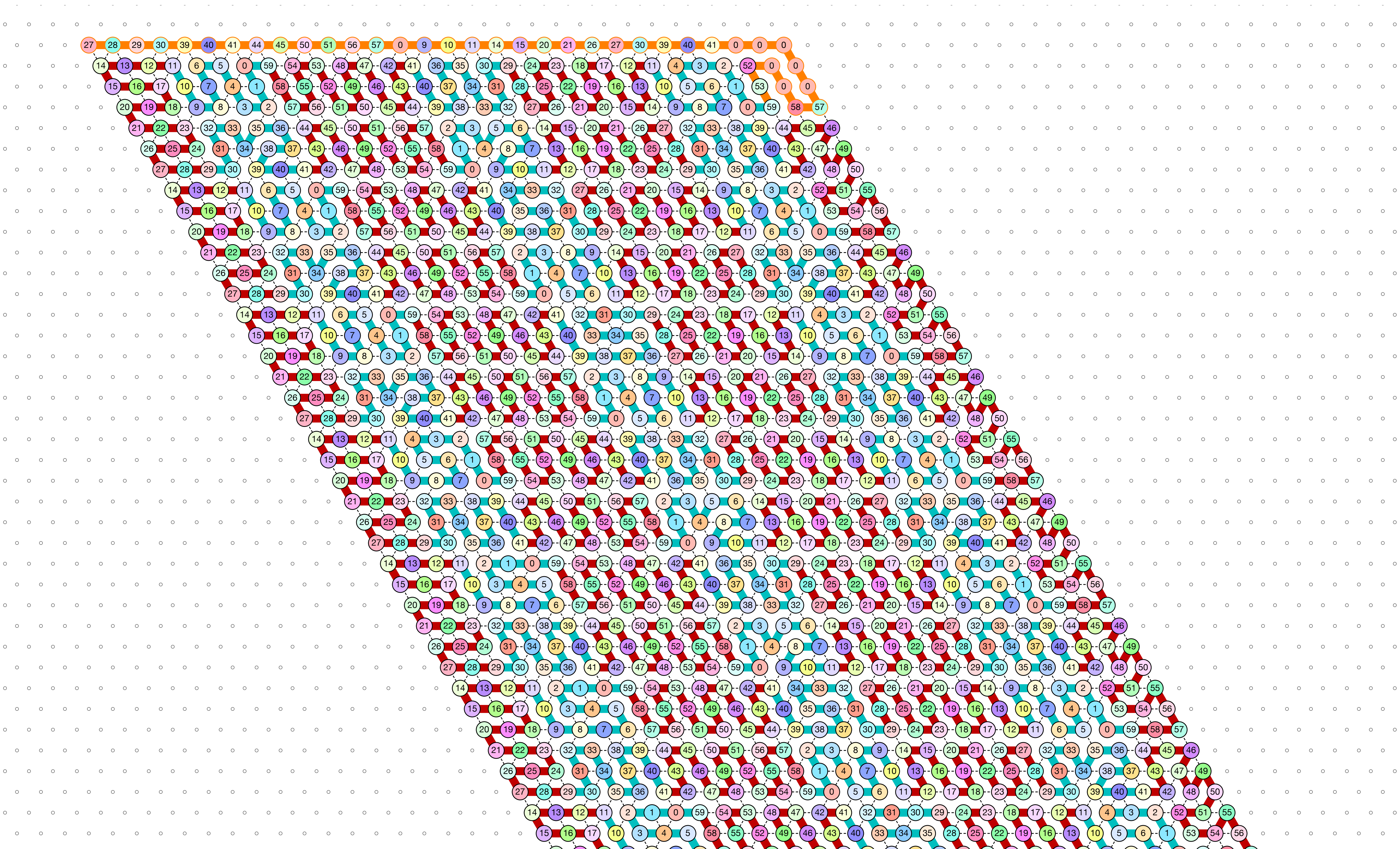
The molecule: `0,…,59,0,…,59,0,…`

of a **single** 60-beads periodic molecule folding upon itself

Geary Meunier Seki Schabanel 2015

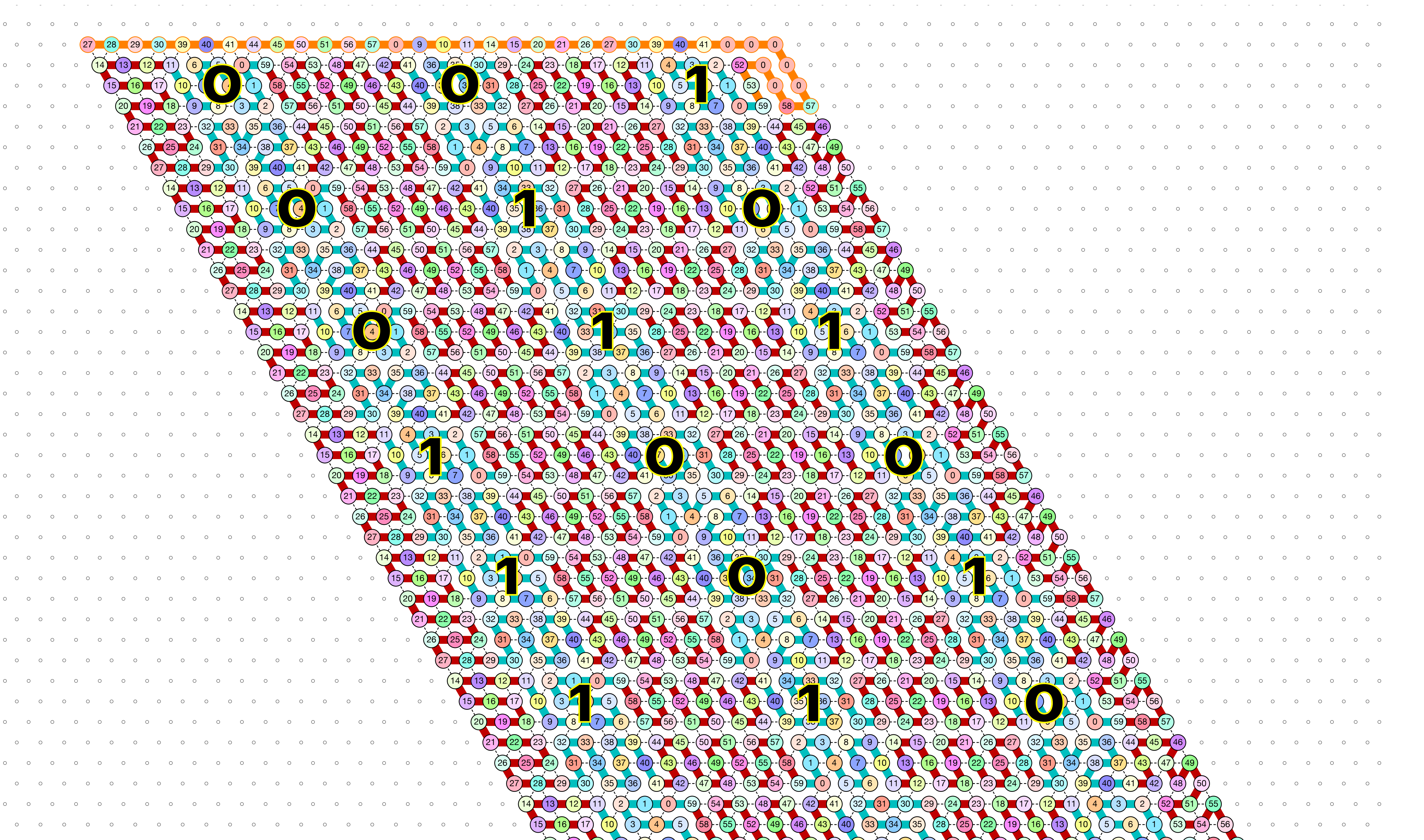
Oritatami. A binary counter

Information is encoded in the geometry



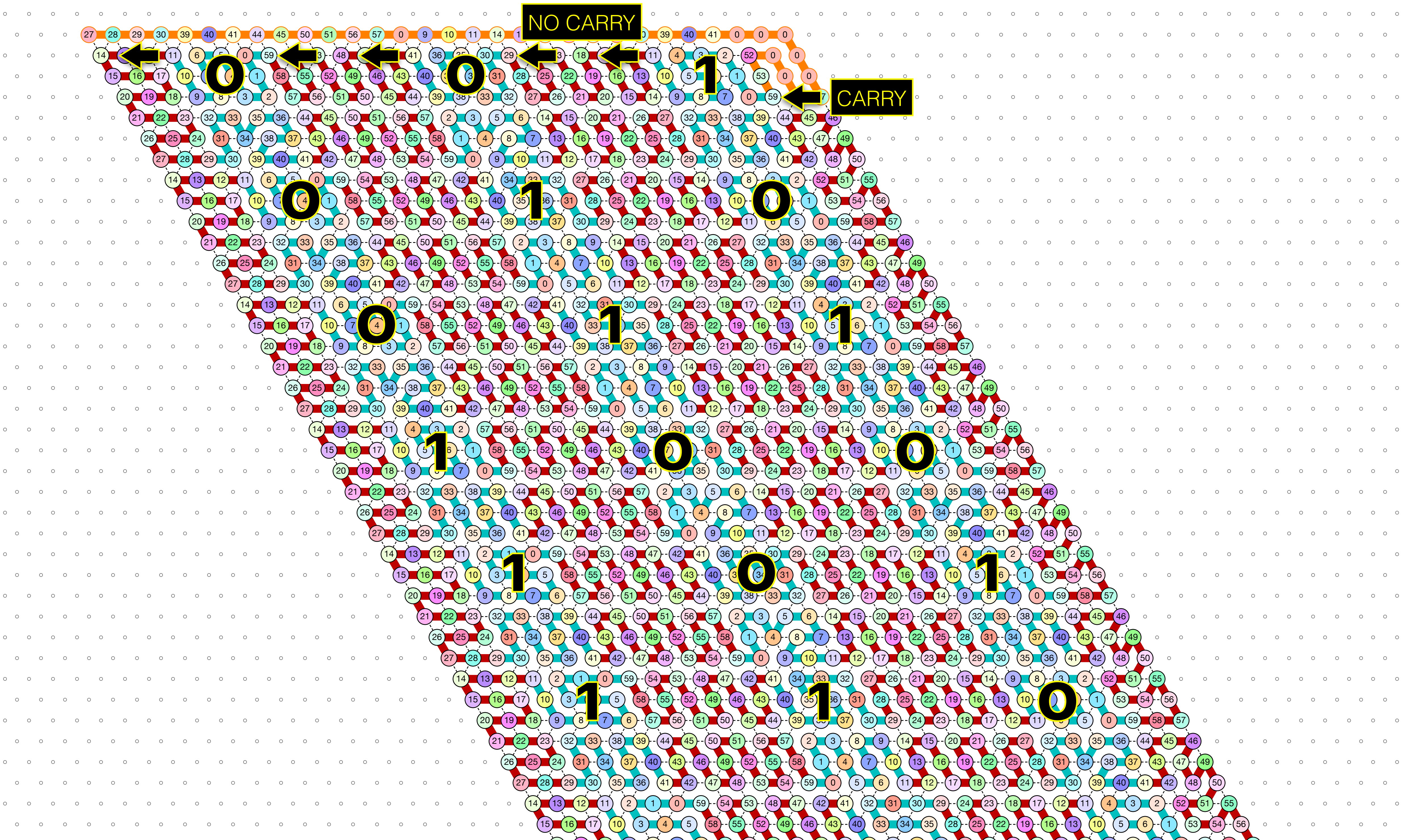
Oritatami. A binary counter

Information is encoded in the geometry



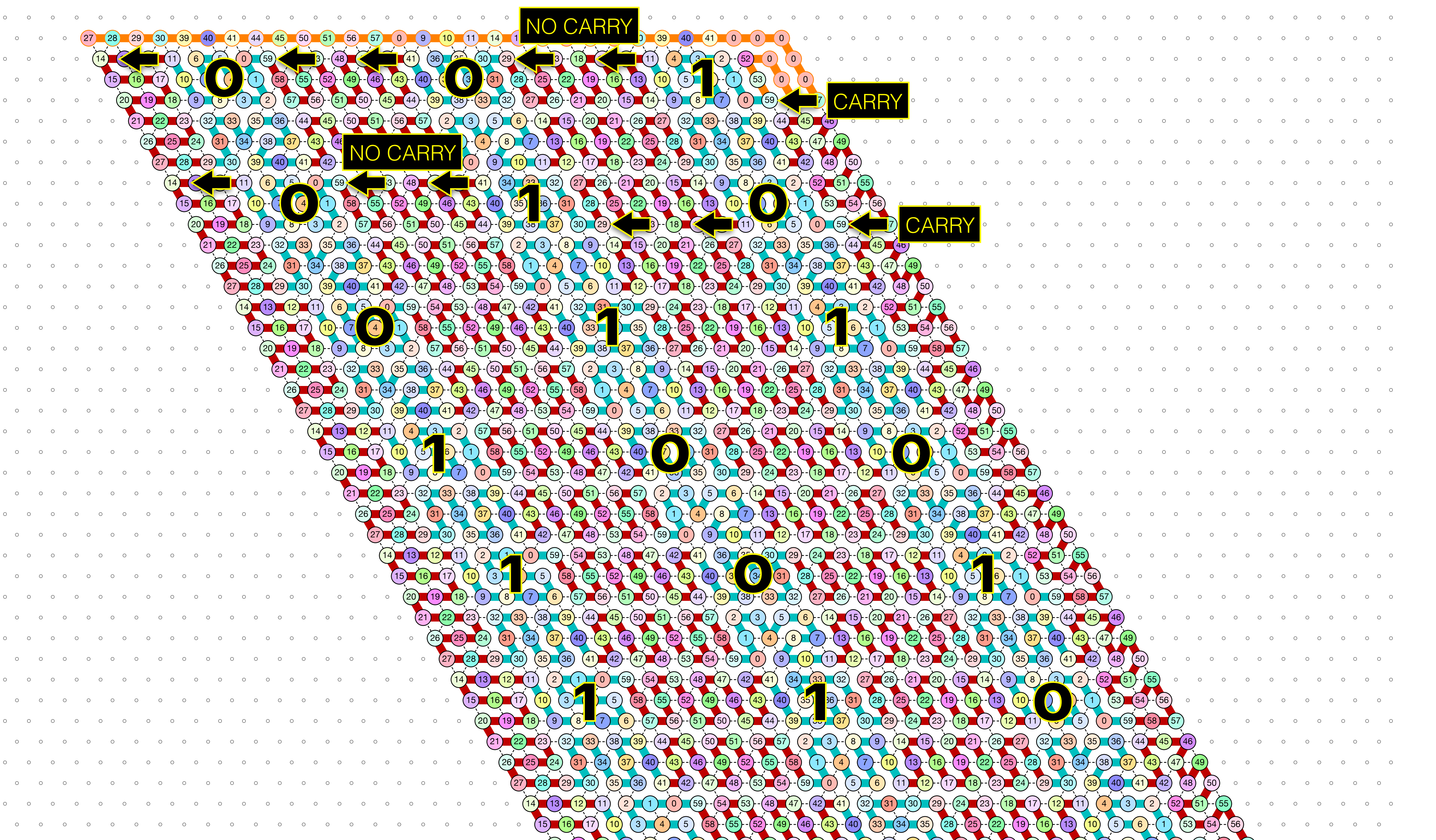
Oritatami. A binary counter

Information is encoded in the geometry



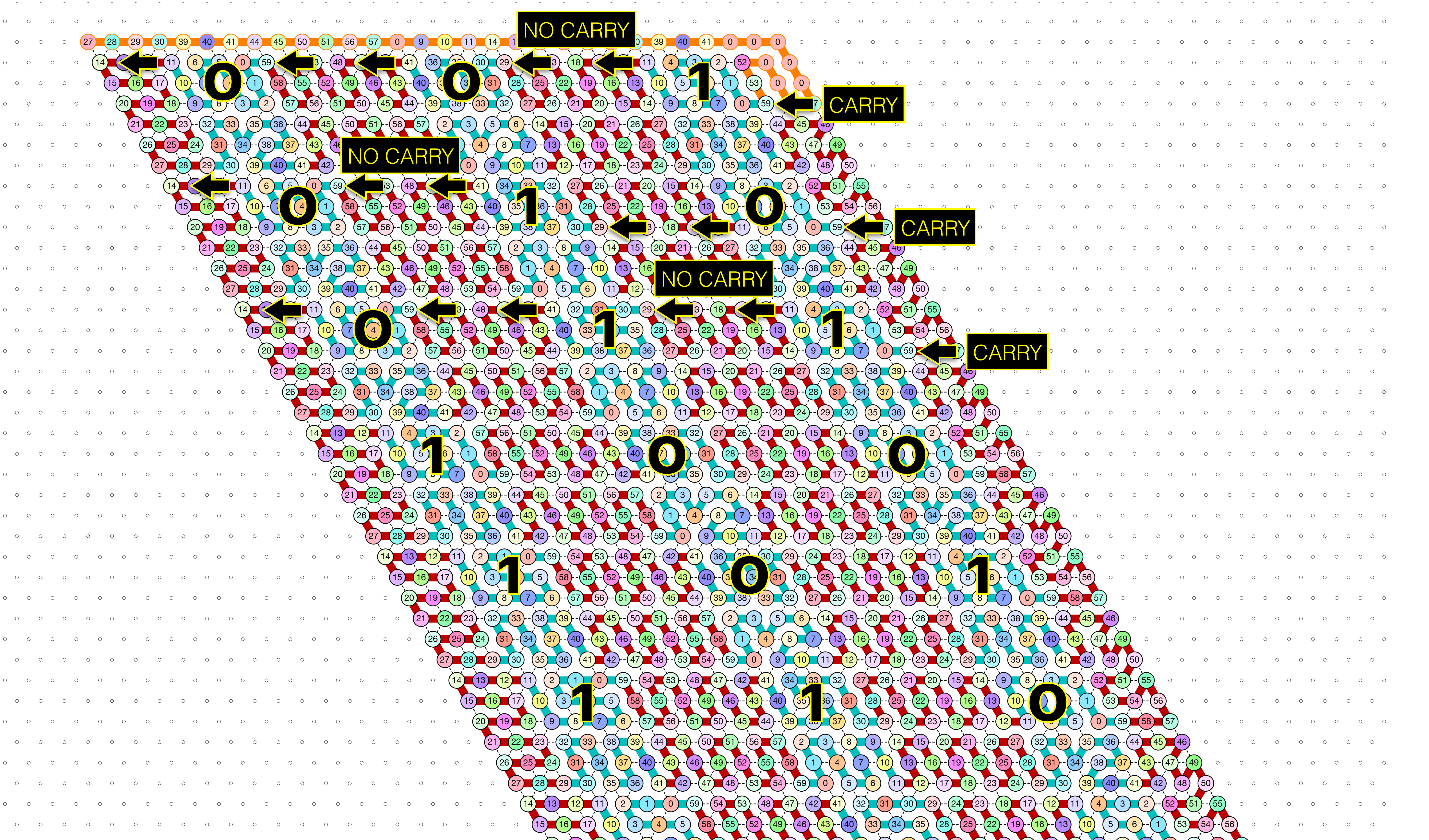
Oritatami. A binary counter

Information is encoded in the geometry



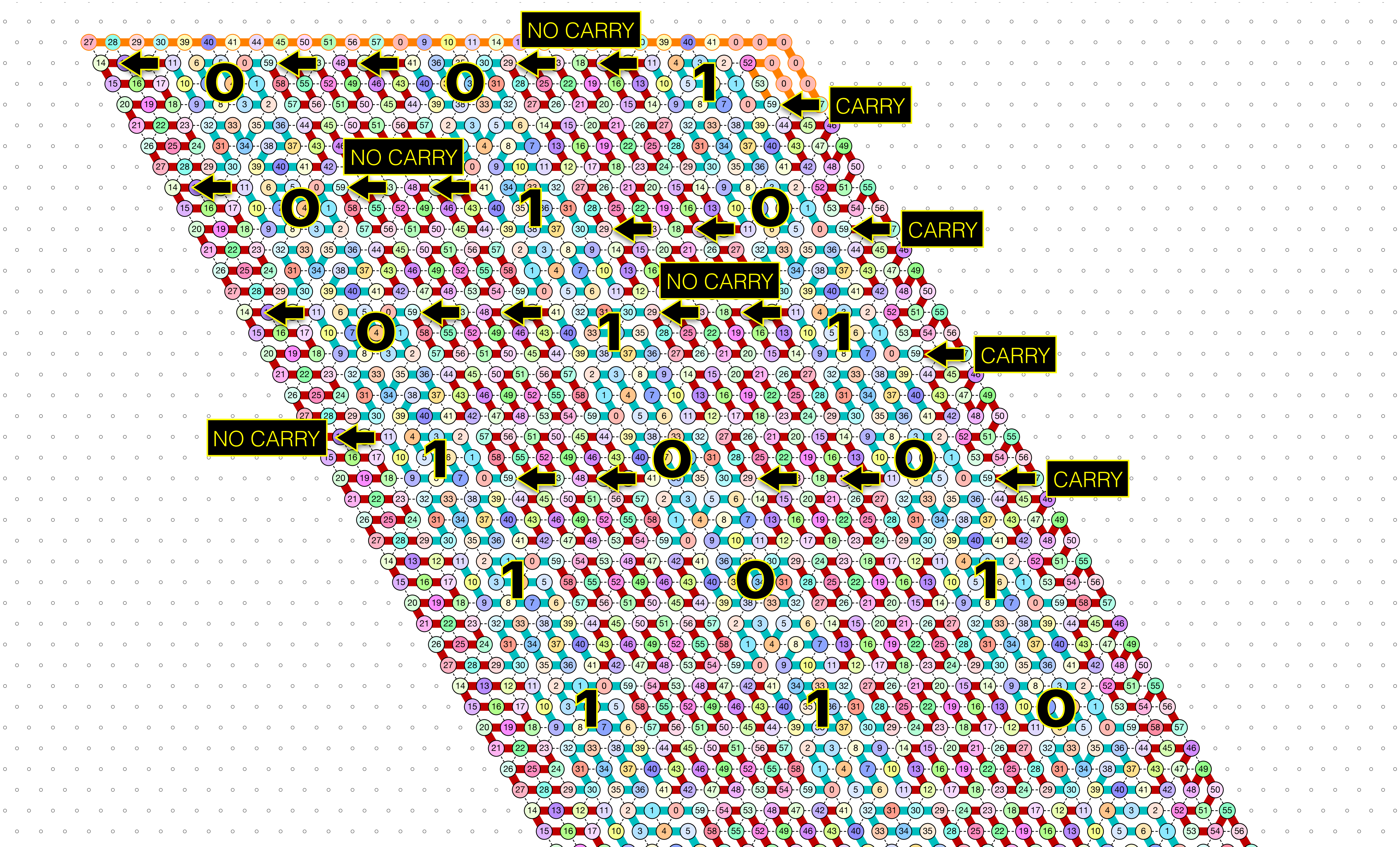
Oritatami. A binary counter

Information is encoded in the geometry



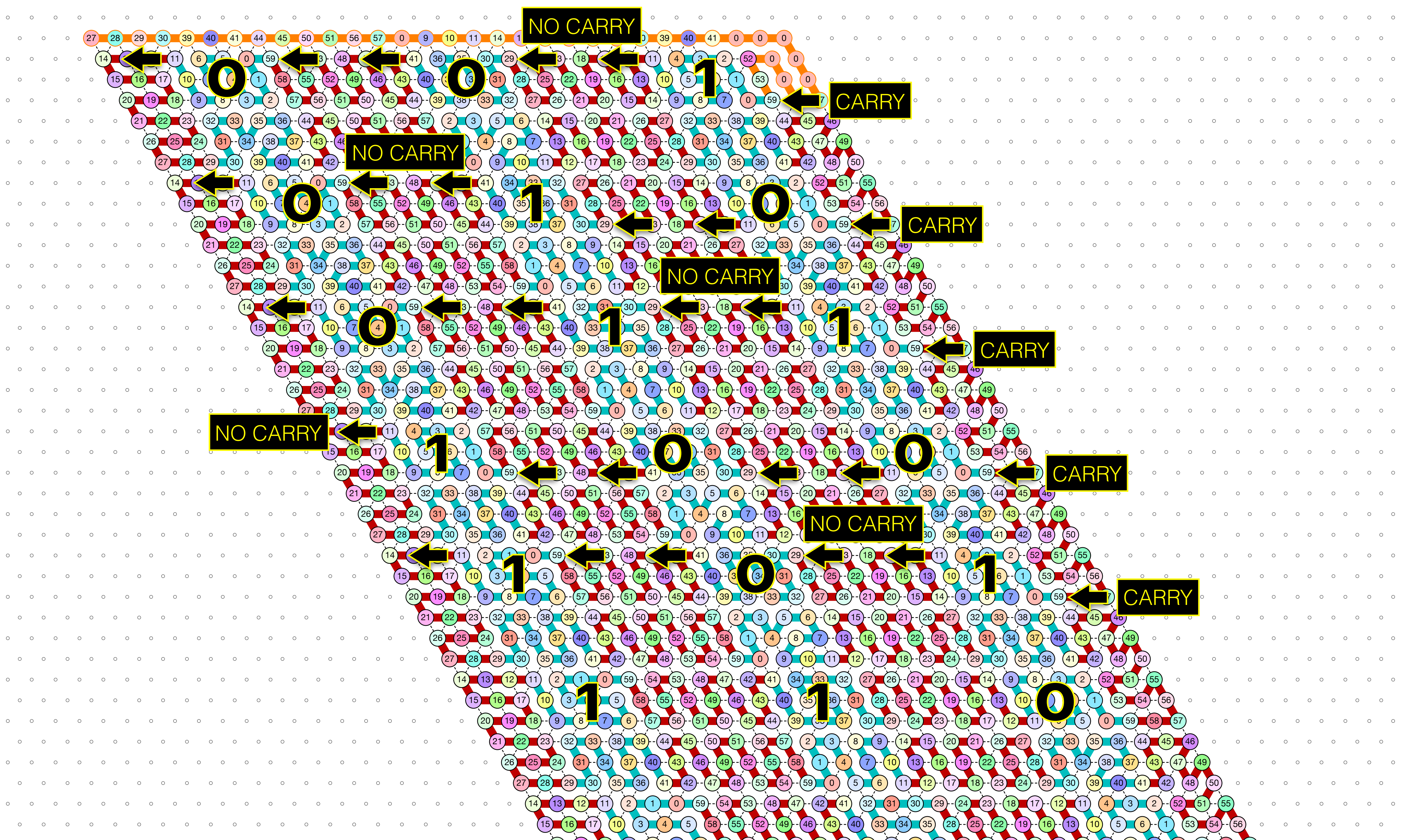
Oritatami. A binary counter

Information is encoded in the geometry



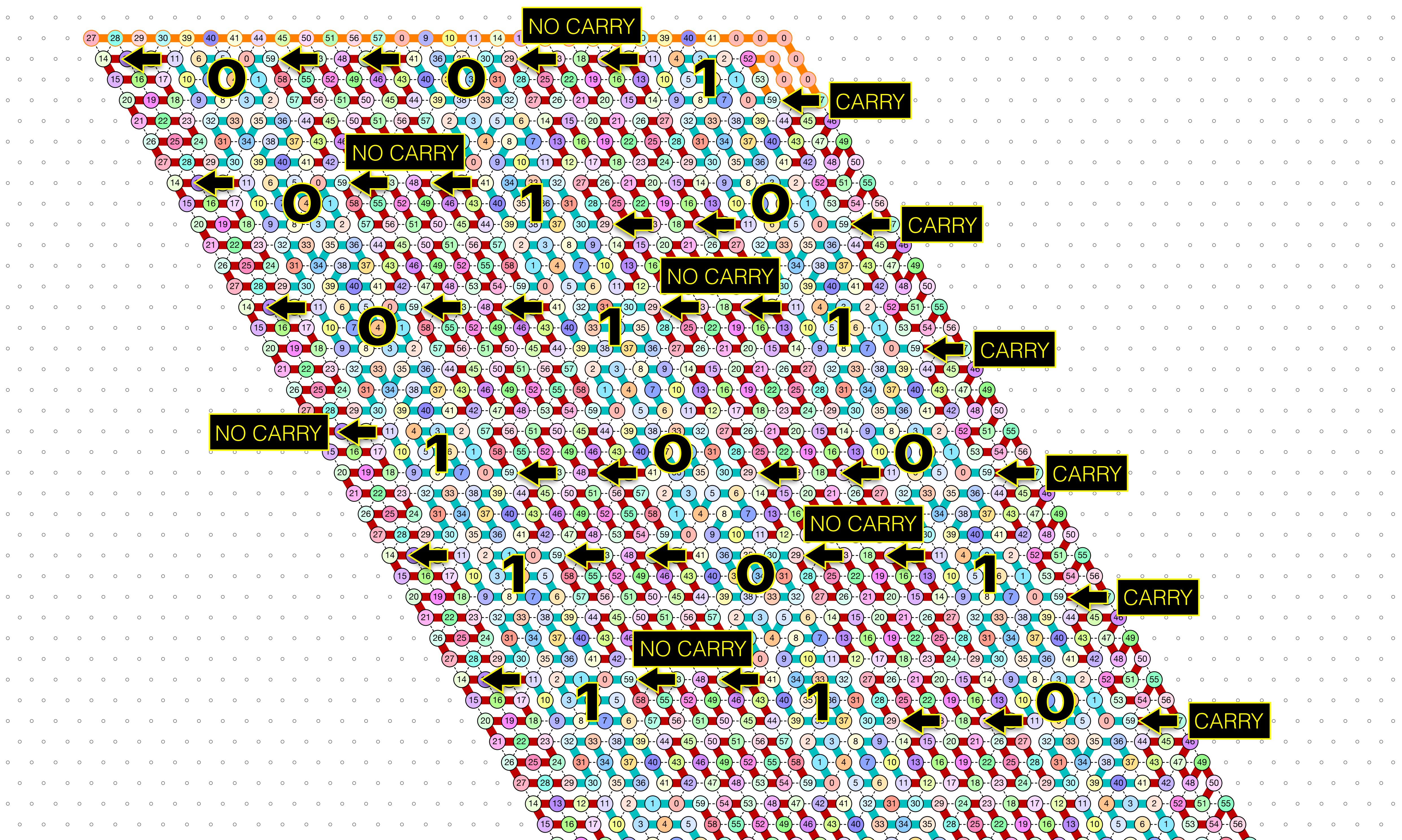
Oritatami. A binary counter

Information is encoded in the geometry



Oritatami. A binary counter

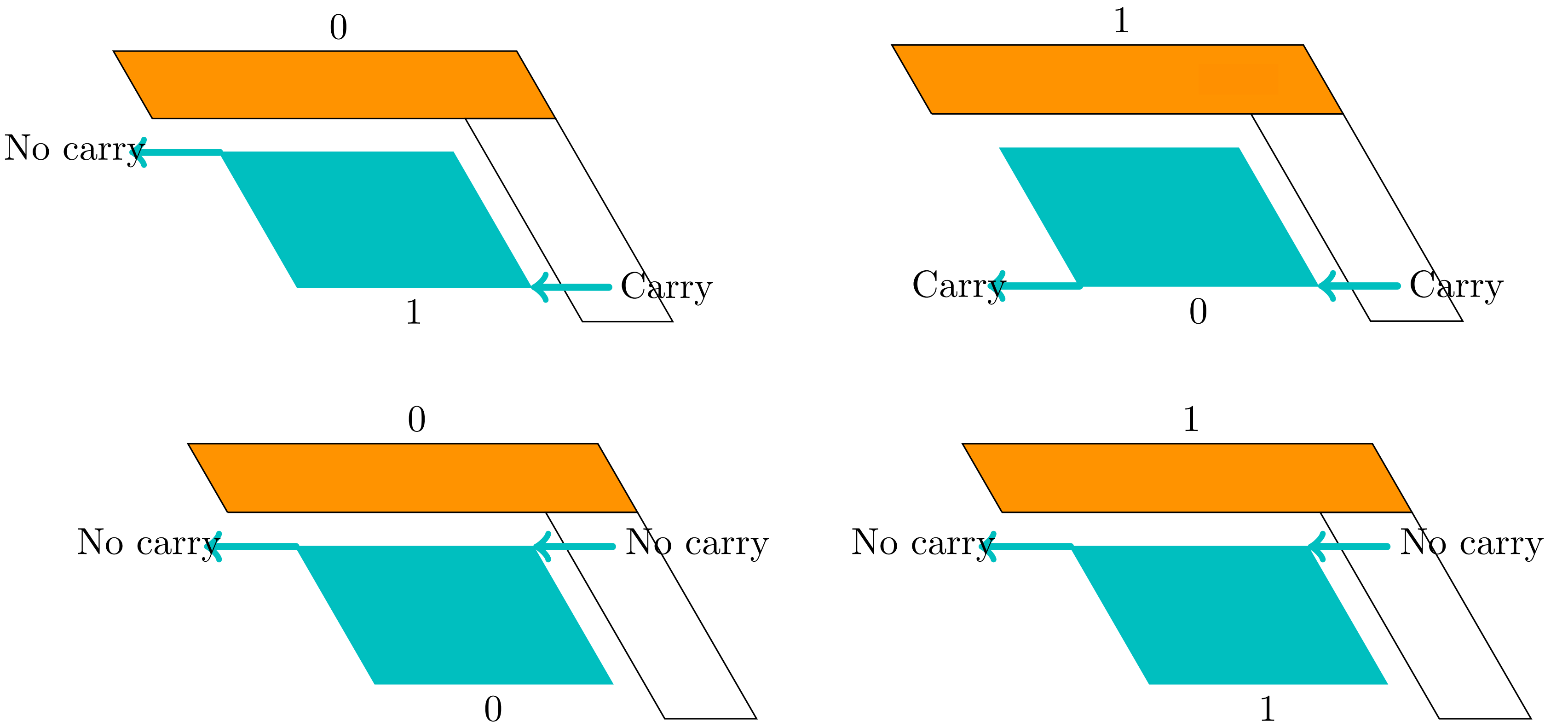
Information is encoded in the geometry



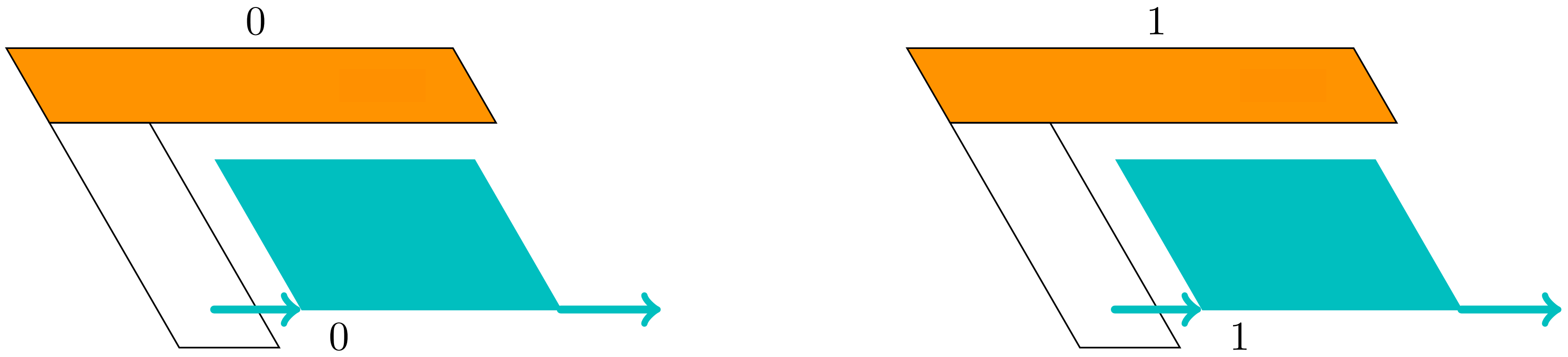
Oritatami. A binary counter

Information is encoded in the geometry

Carry propagation

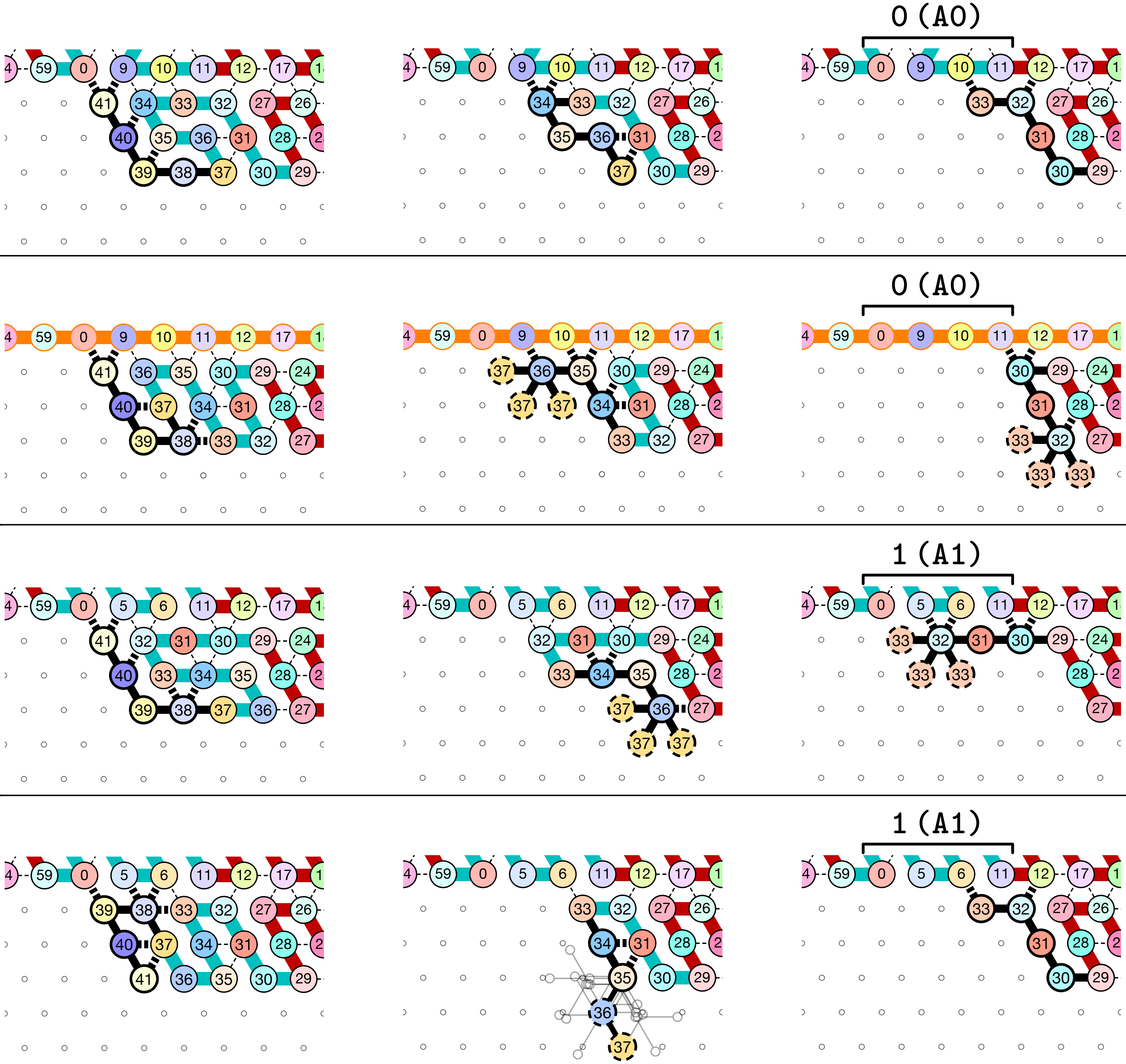


Line feed



How does computation work?

3) Beads 38-41 ← 2) Beads 34-37 ← 1) Beads 30-33



Carry = 0 … Carry = 1

Carry = 0 … Carry = 0

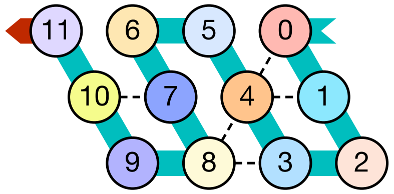
Carry = 0 … Carry = 0

Carry = 1 … Carry = 1

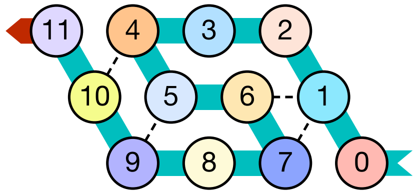
Proving the binary counter:
First, define the *bricks*

- Module A, First Half-Adder (beads 0–11):

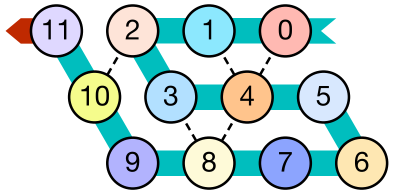
Zig Direction ⟵



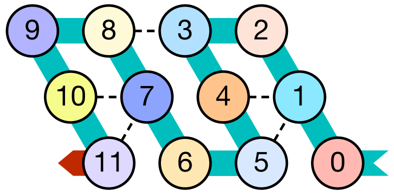
Brick A00



Brick A01

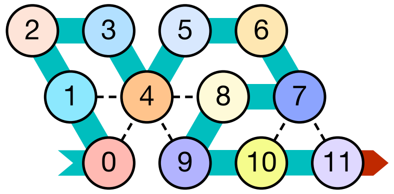


Brick A10

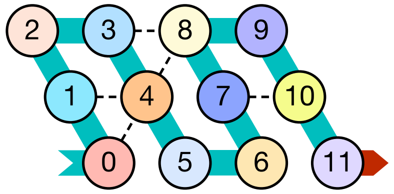


Brick A11

Zag Direction ⟶



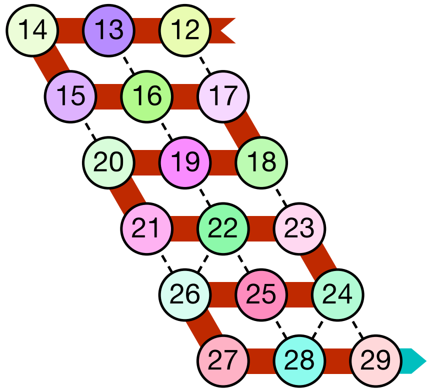
Brick A0



Brick A1

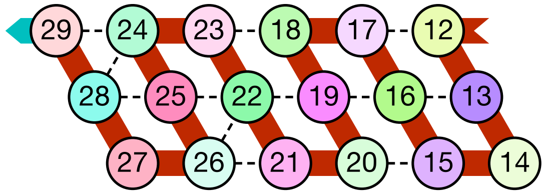
- Module B, Left-Turn module (beads 12–29)

Left Turn ↪ (Zig-to-Zag turn)

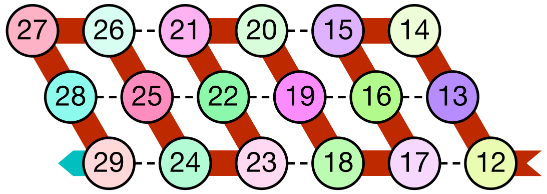


Brick BT

Zig Direction ⟵

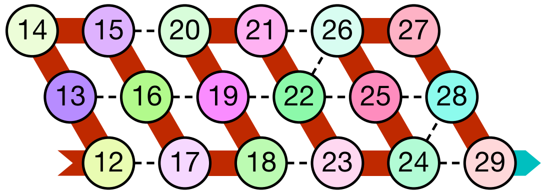


Brick B0



Brick B1

Zag Direction ⟶

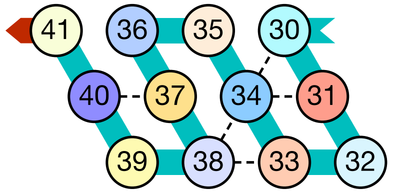


Brick B2

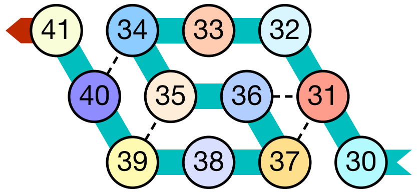
Proving the binary counter:
First, define the *bricks*

- Module C, Second Half-Adder (beads 30–41)

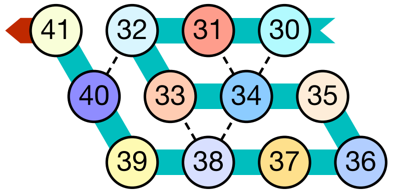
Zig Direction ⟵



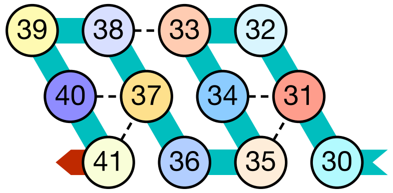
Brick C00



Brick C01

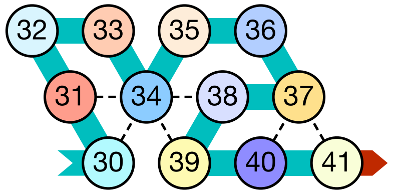


Brick C10

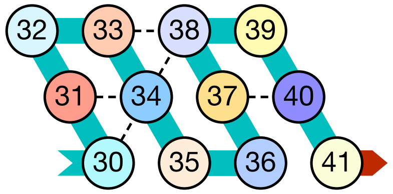


Brick C11

Zag Direction ⟶



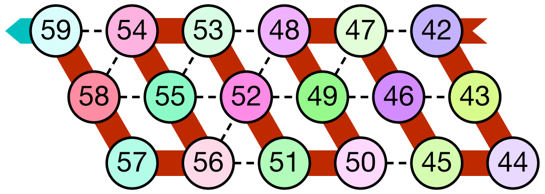
Brick C0



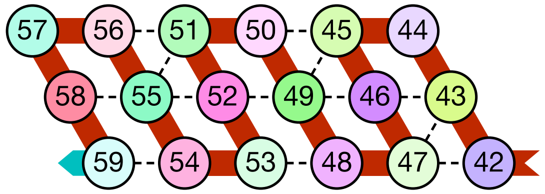
Brick C1

- Module D, Right-Turn module (beads 42–59)

Zig Direction ⟵

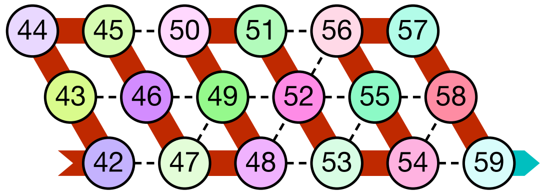


Brick D0



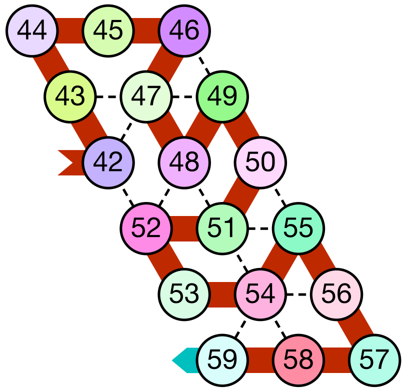
Brick D1

Zag Direction ⟶



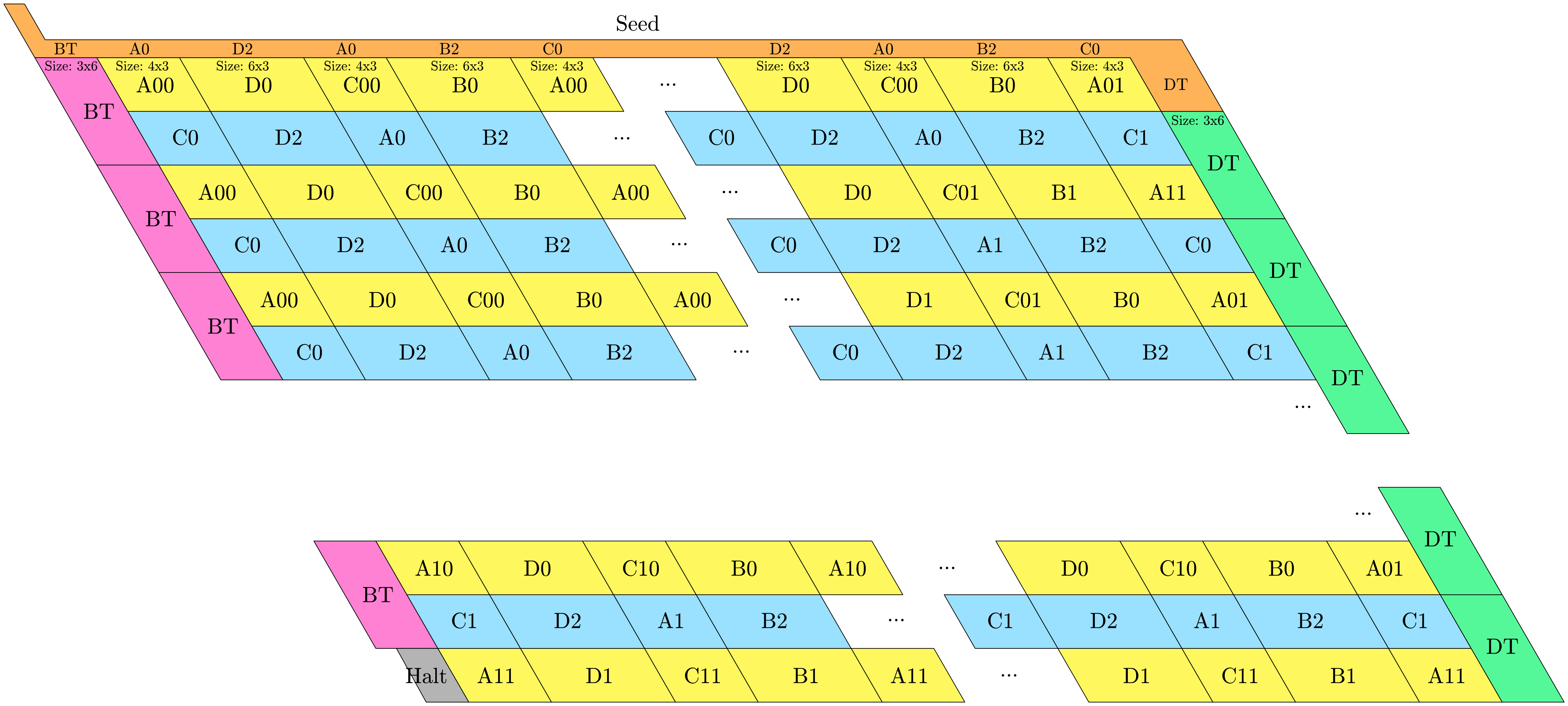
Brick D2

Right Turn ⟷ (Zag-to-Zig turn)



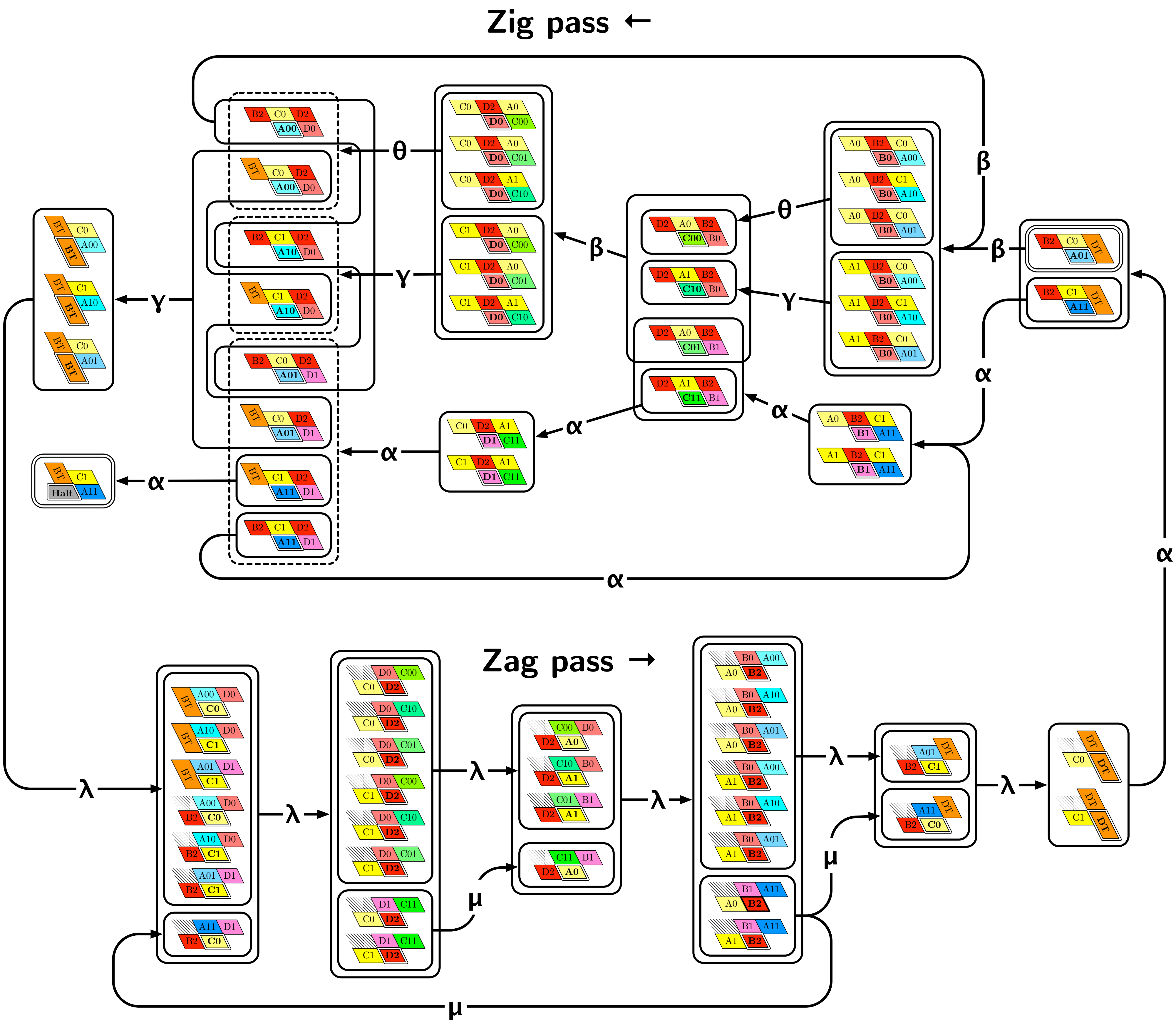
Brick DT

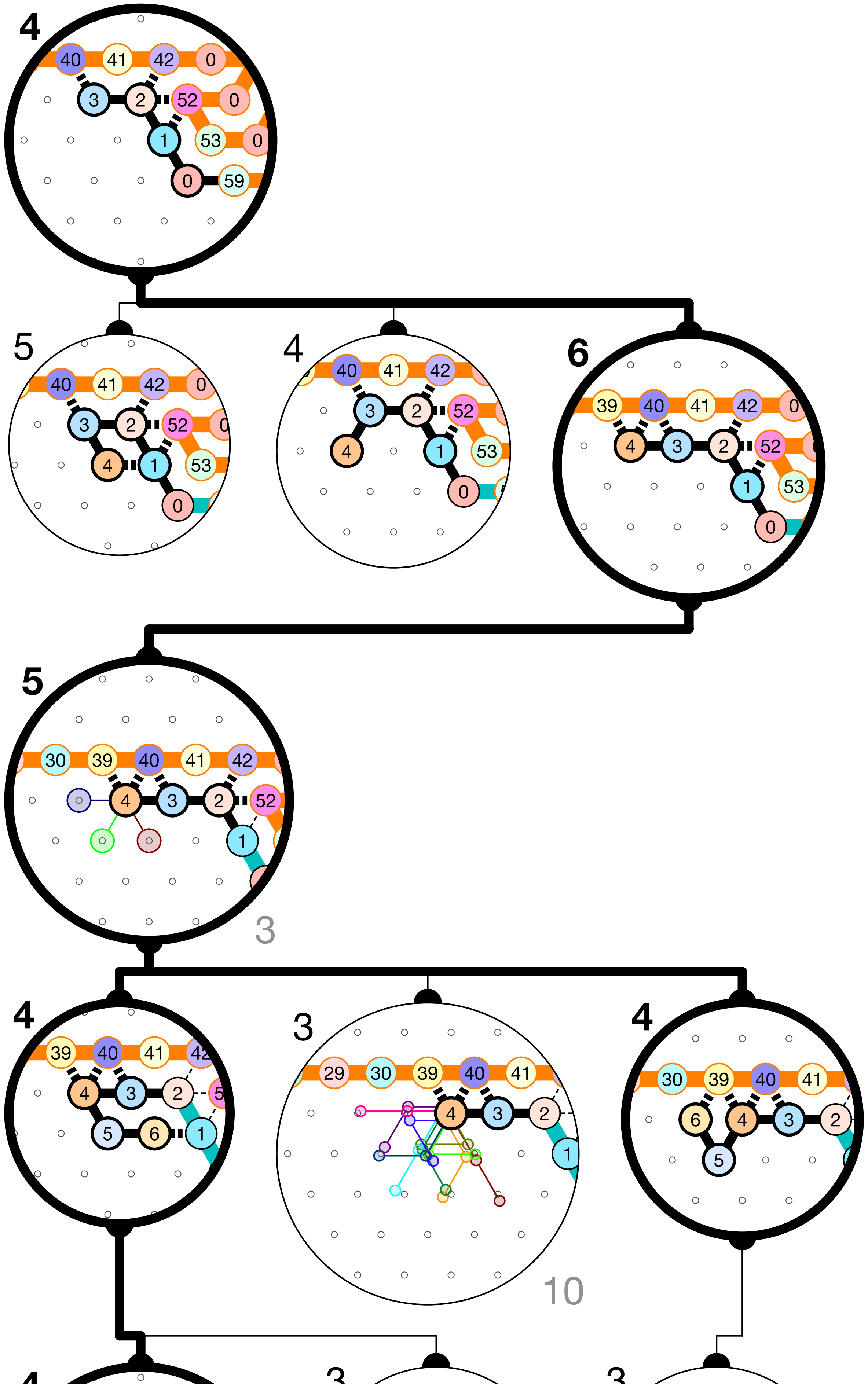
2nd, describe the final folding


Seed
BT A0 D2 A0 B2 C0 D2 A0 B2 C0
Size: 3x6 Size: 4x3 Size: 6x3 Size: 4x3 Size: 6x3 Size: 4x3 Size: 6x3 Size: 4x3 Size: 6x3 Size: 4x3
A00 D0 C00 B0 A00 ··· D0 C00 B0 A01 DT
BT
Size: 3x6
C0 D2 A0 B2 ··· C0 D2 A0 B2 C1
DT
A00 D0 C00 B0 A00 ··· D0 C01 B1 A11
BT
C0 D2 A0 B2 ··· C0 D2 A1 B2 C0
DT
A00 D0 C00 B0 A00 ··· D1 C01 B0 A01
BT
C0 D2 A0 B2 ··· C0 D2 A1 B2 C1
DT
···
···
DT
A10 D0 C10 B0 A10 ··· D0 C10 B0 A01
BT
C1 D2 A1 B2 ··· C1 D2 A1 B2 C1
DT
Halt A11 D1 C11 B1 A11 ··· D1 C11 B1 A11

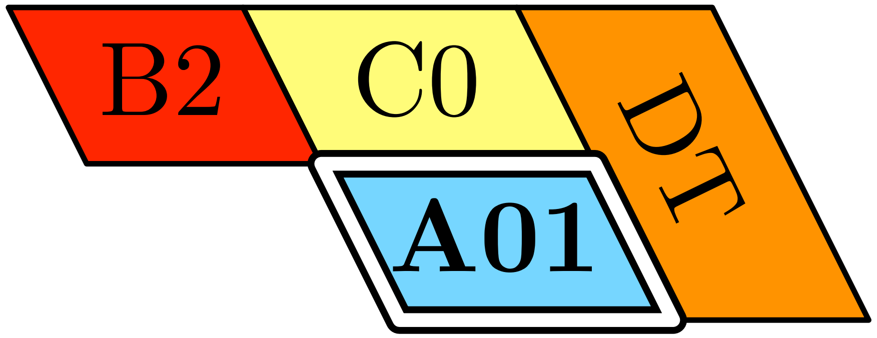
We prove that the molecule folds like this by induction

3rd, enumerate all the environments for each brick

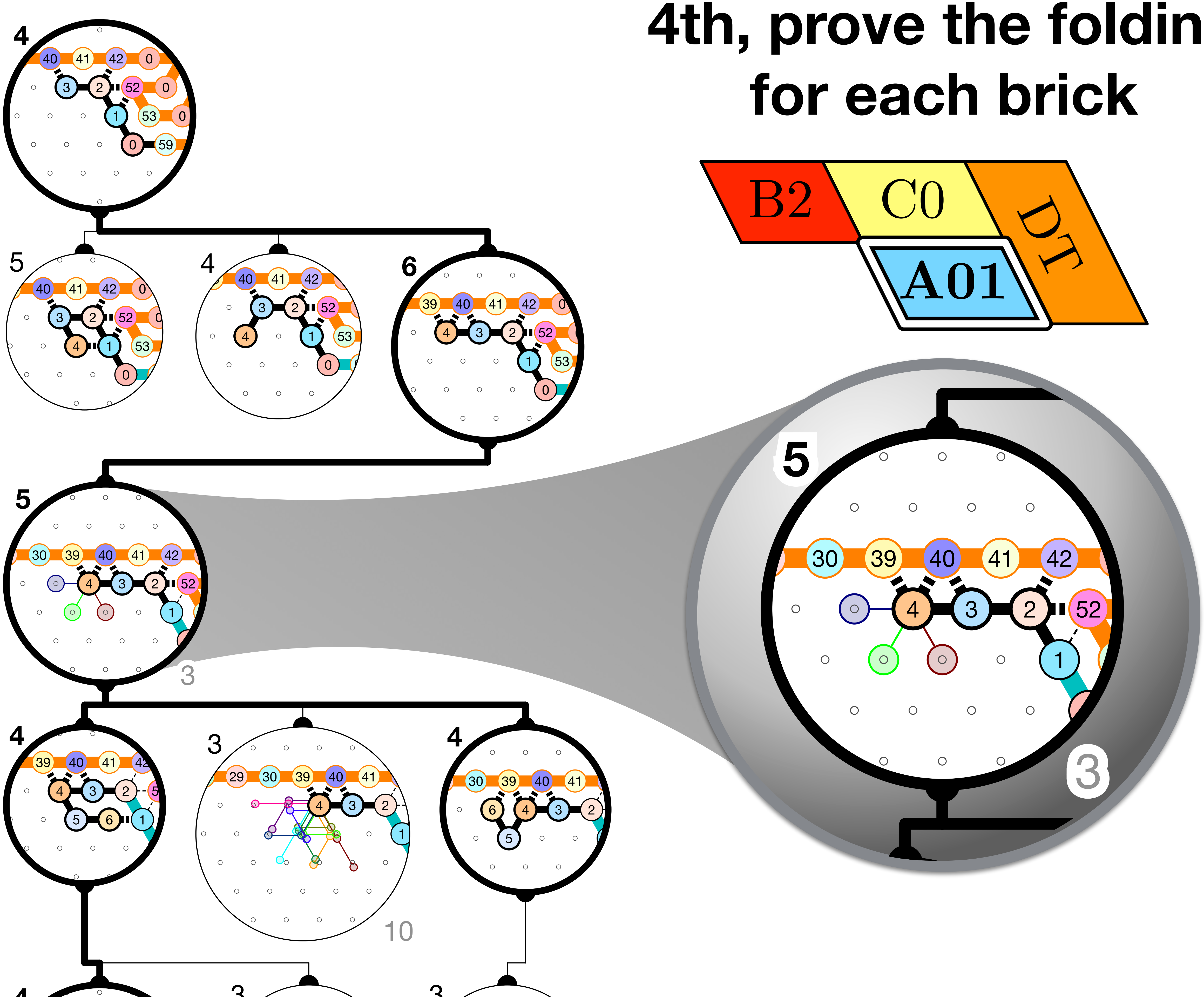


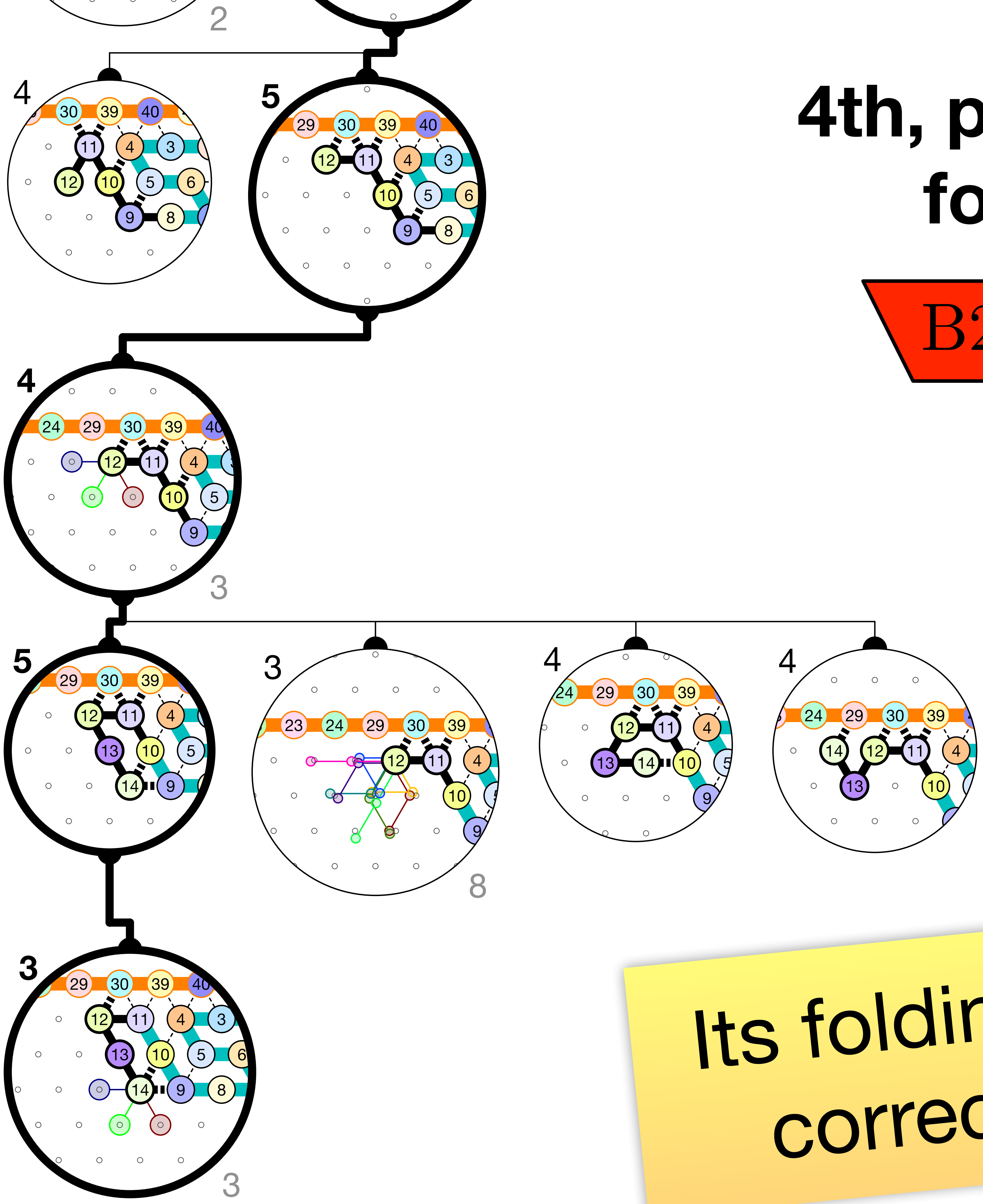


4th, prove the folding for each brick



4th, prove the folding for each brick





4th, prove the folding for each brick

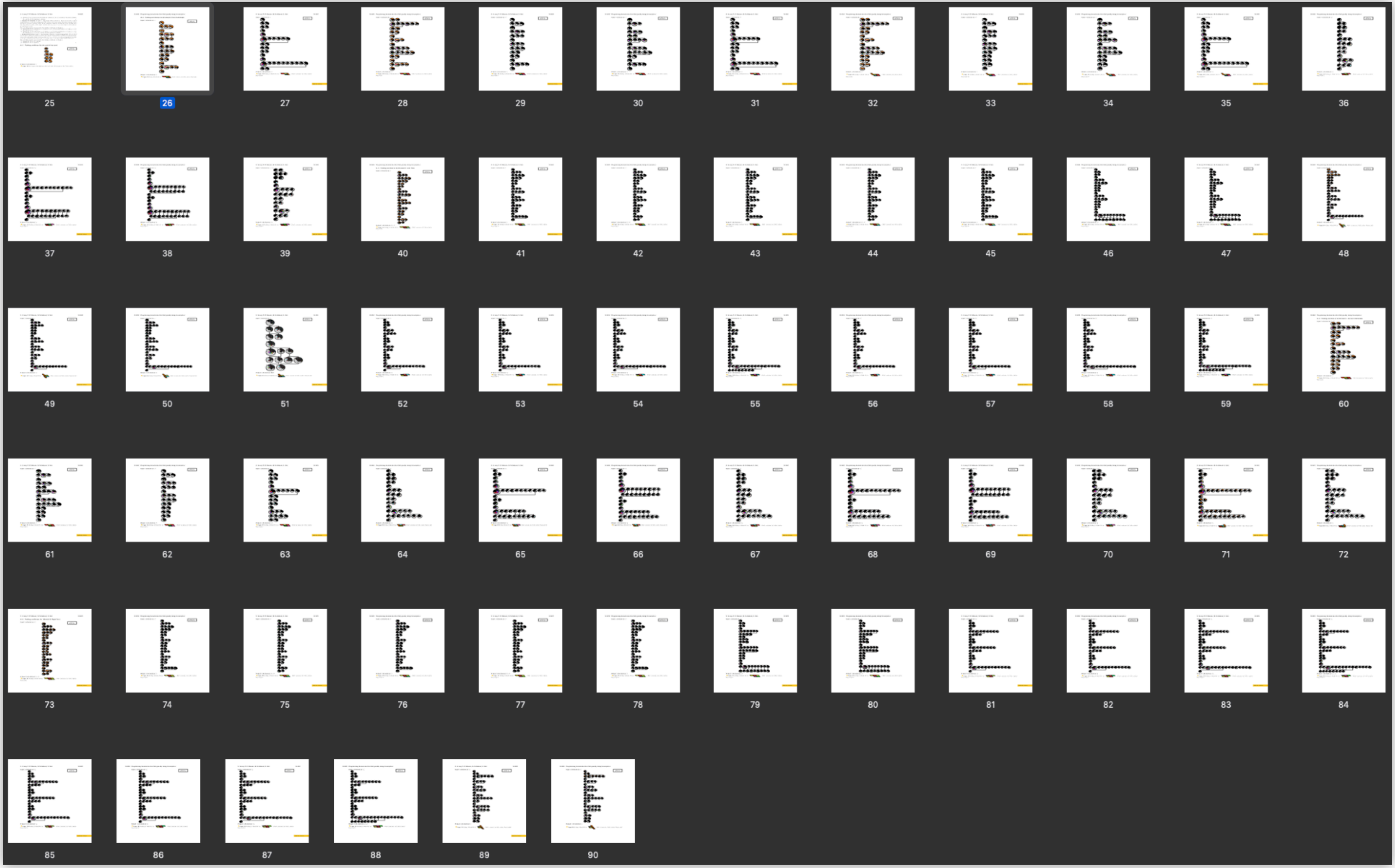
B2 C0 DT

A01

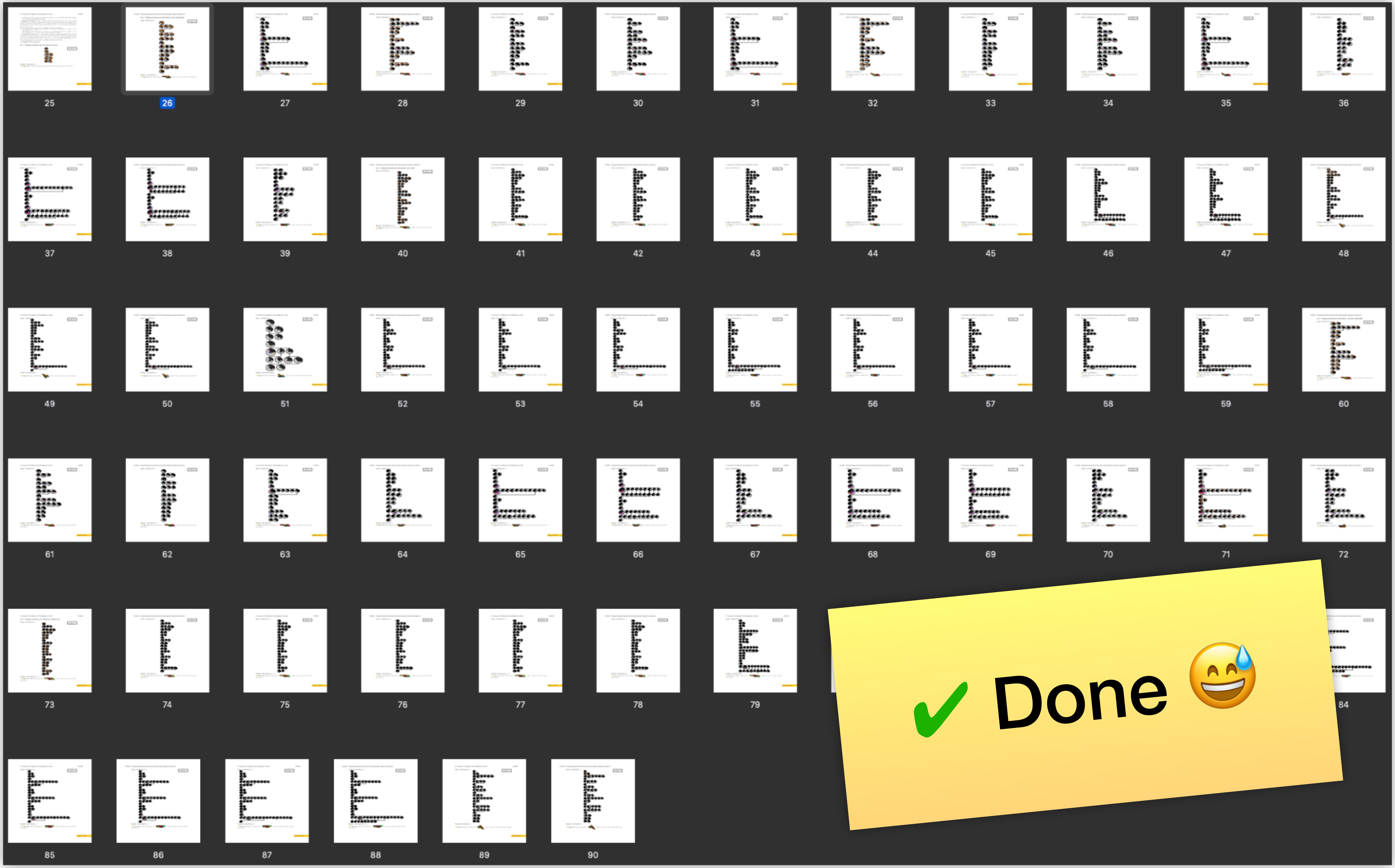
Its folding is correct !

Repeat for each brick in each environment

Repeat for each brick in each environment



Repeat for each brick in each environment



✔ Done 😅

Binary counter: conclusion

- **Theorem.** There is a 60-periodic molecule that simulates a binary counter using 60 bead types and delay 3.

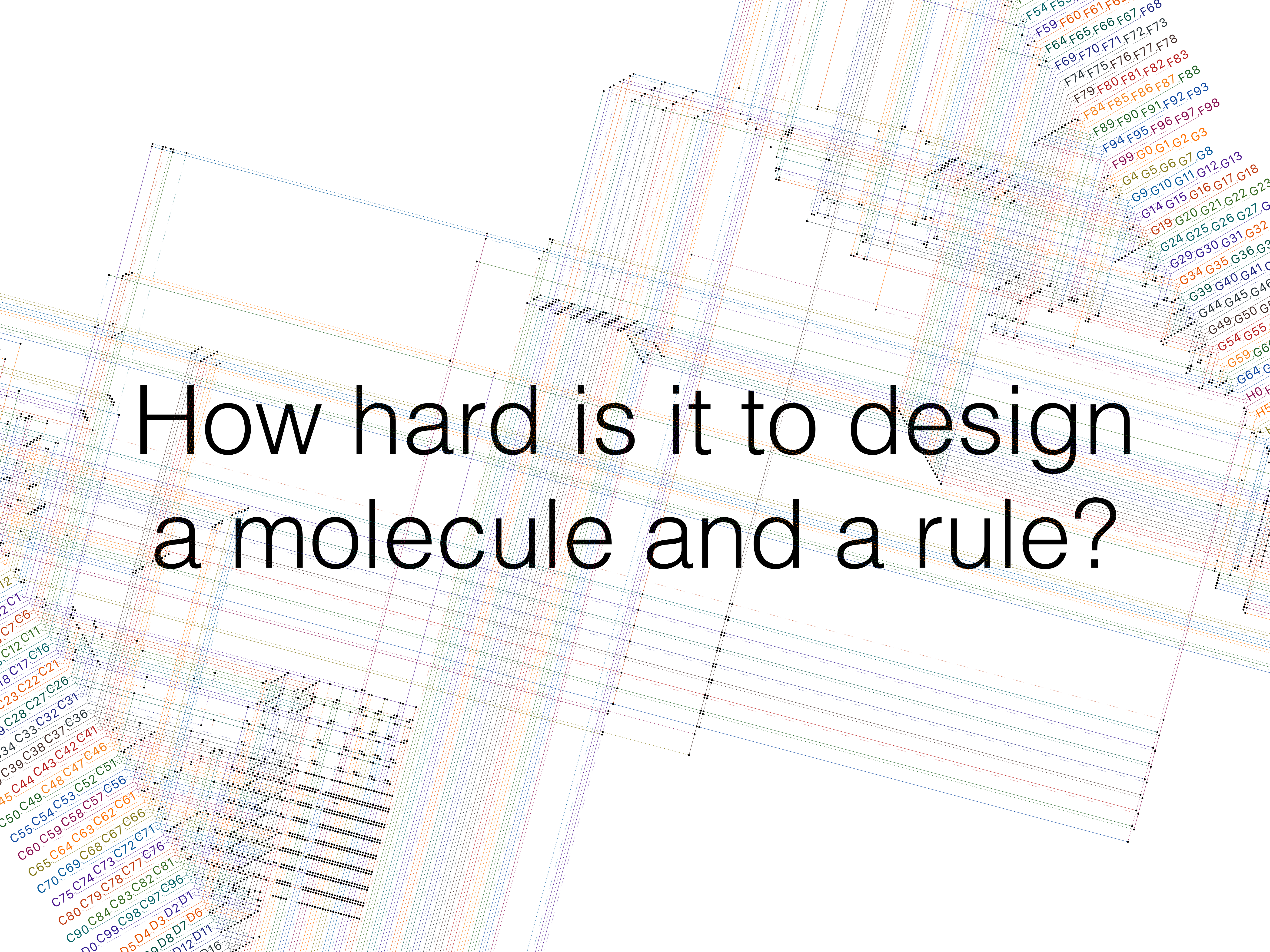
Back to general oritatami

- How hard is it to design a rule?

- What can it compute?

Back to general oritatami

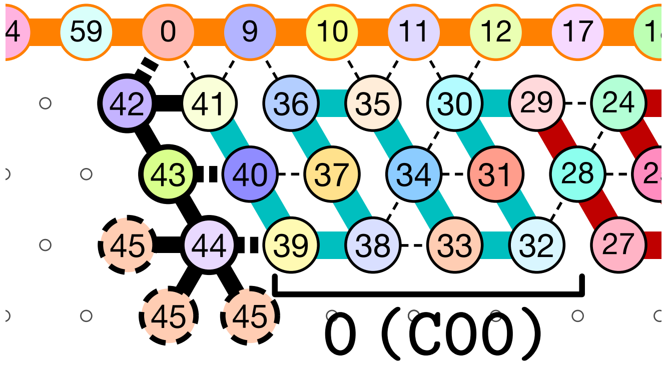
- How hard is it to design a rule?
  - NP-hard… but FPT, thus feasible!
- What can it compute?
  - Simulates any Turing Machine… *efficiently!*



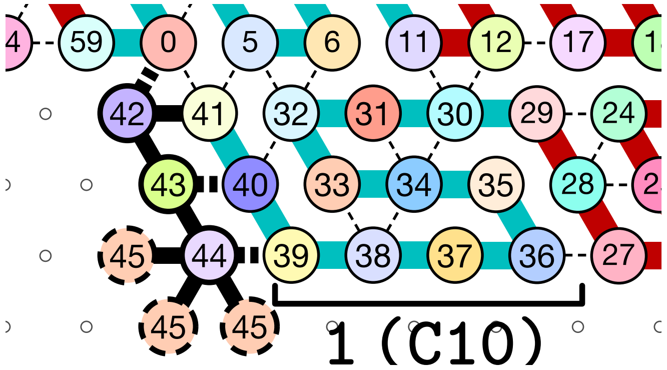
How hard is it to design a molecule and a rule?

The first challenge: Designing the desired shapes

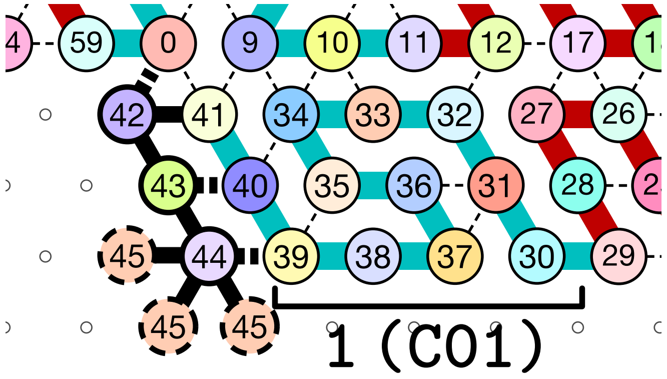
- Design shapes for which a **common** rule ❤️ exists



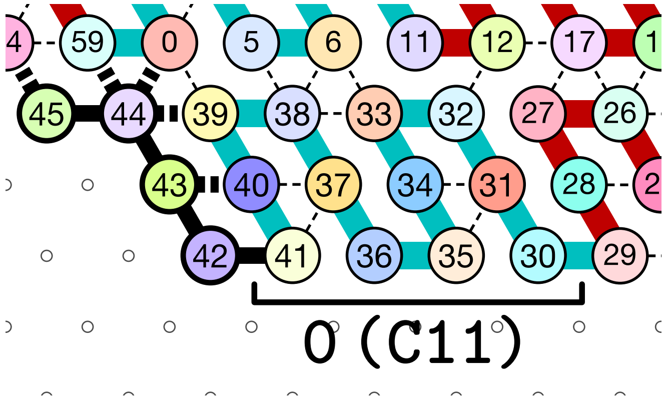
0+0 = 0 + no C



1+0 = 1 + no C



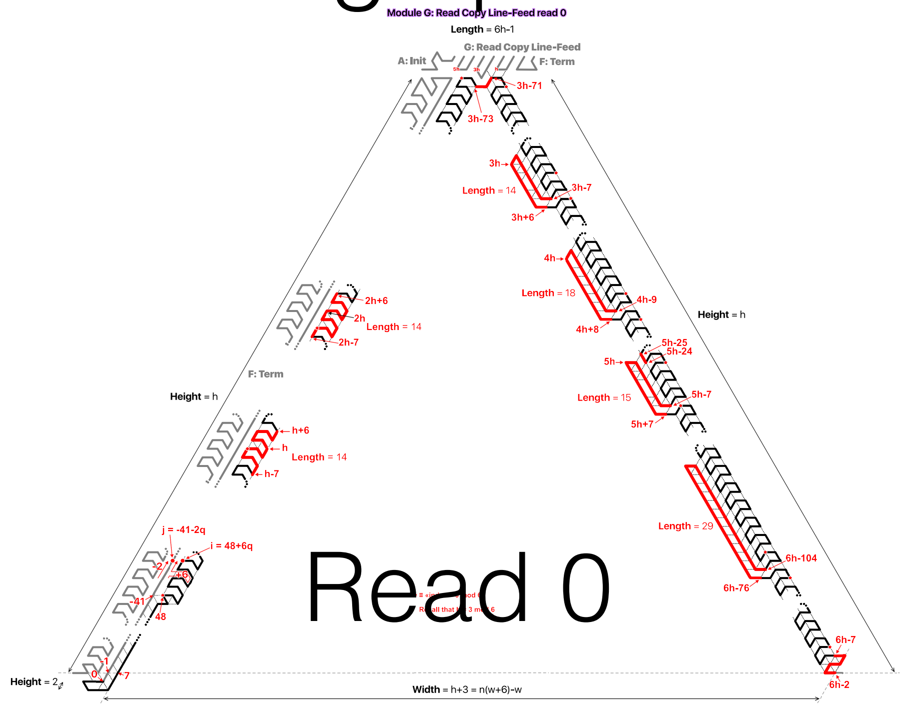
0+1 = 1 + no C



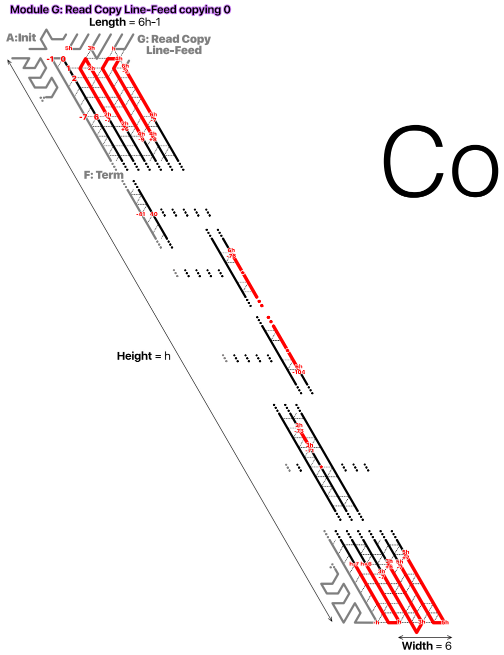
1+1 = 0 + C

The first challenge: Designing the desired paths

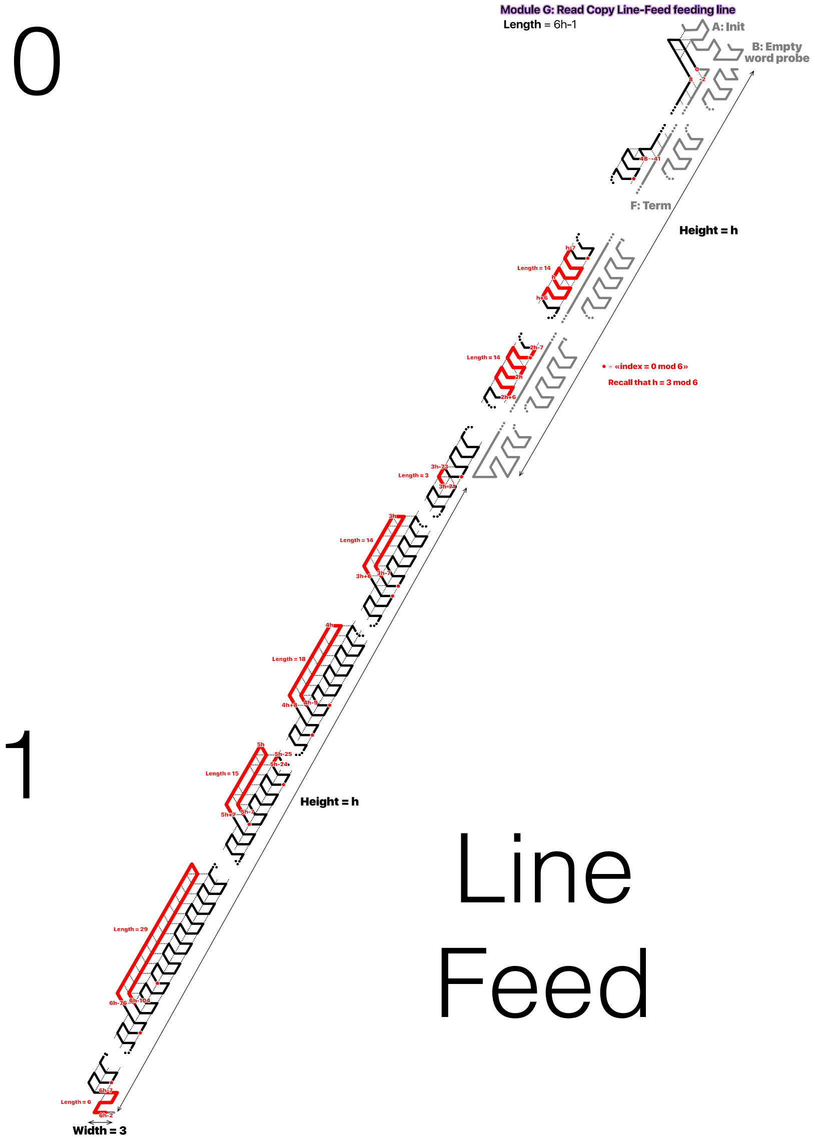
- Design paths for which a **common** rule ❤️ exists



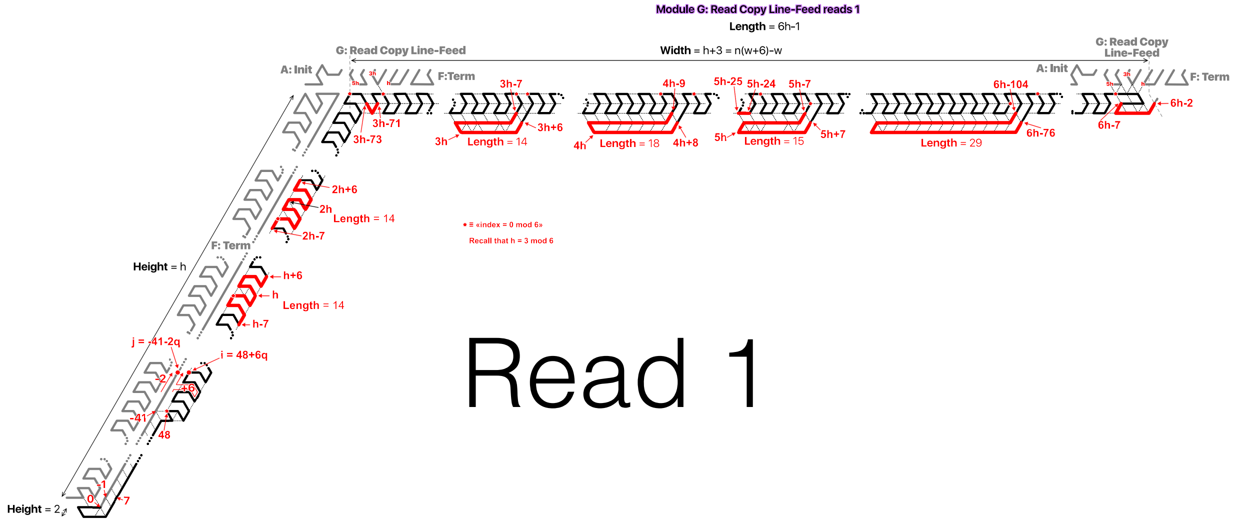
Read 0



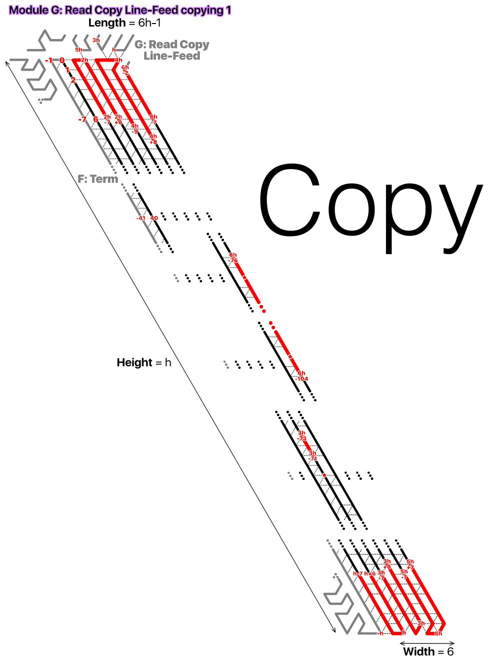
Copy 0



Line Feed



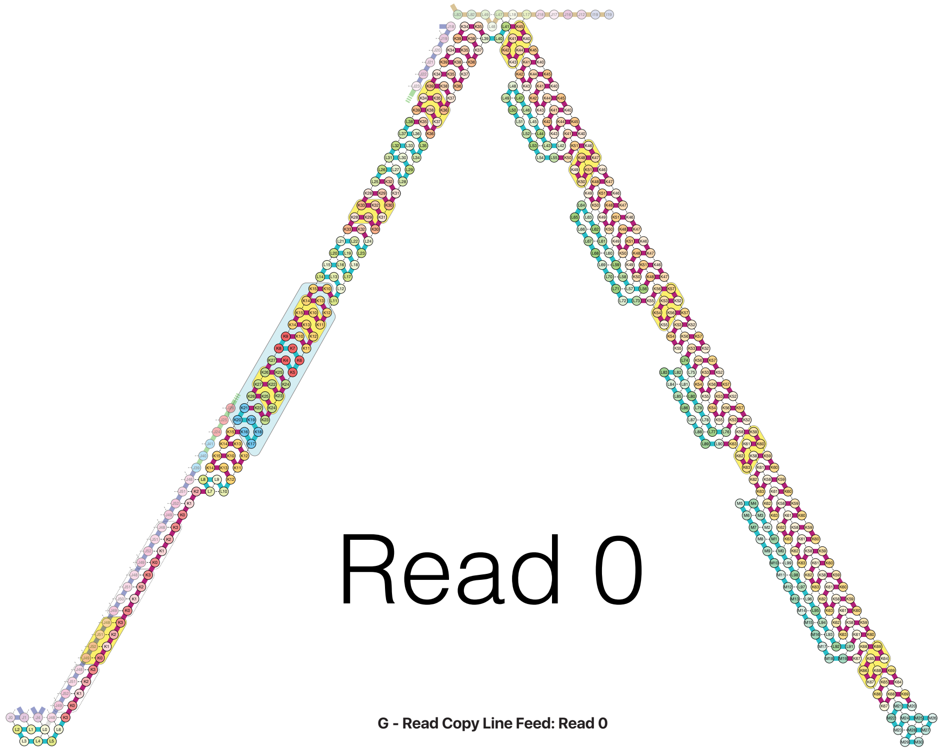
Read 1



Copy 1

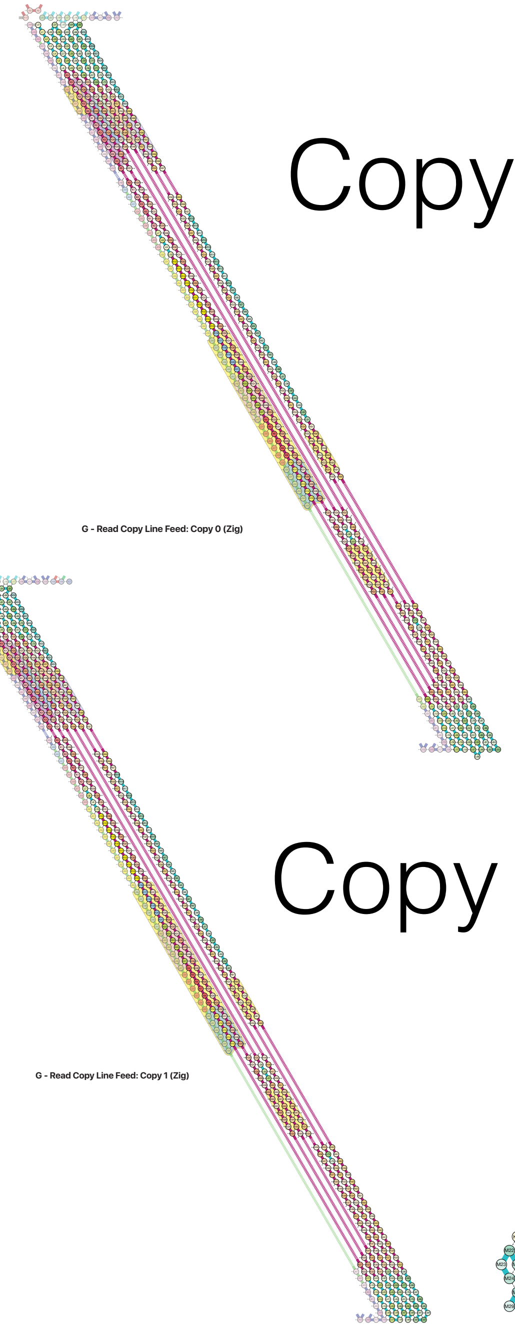
The first challenge: Designing the paths

- Design paths for which a common rule ❤️ exists



G - Read Copy Line Feed: Read 0

Read 0

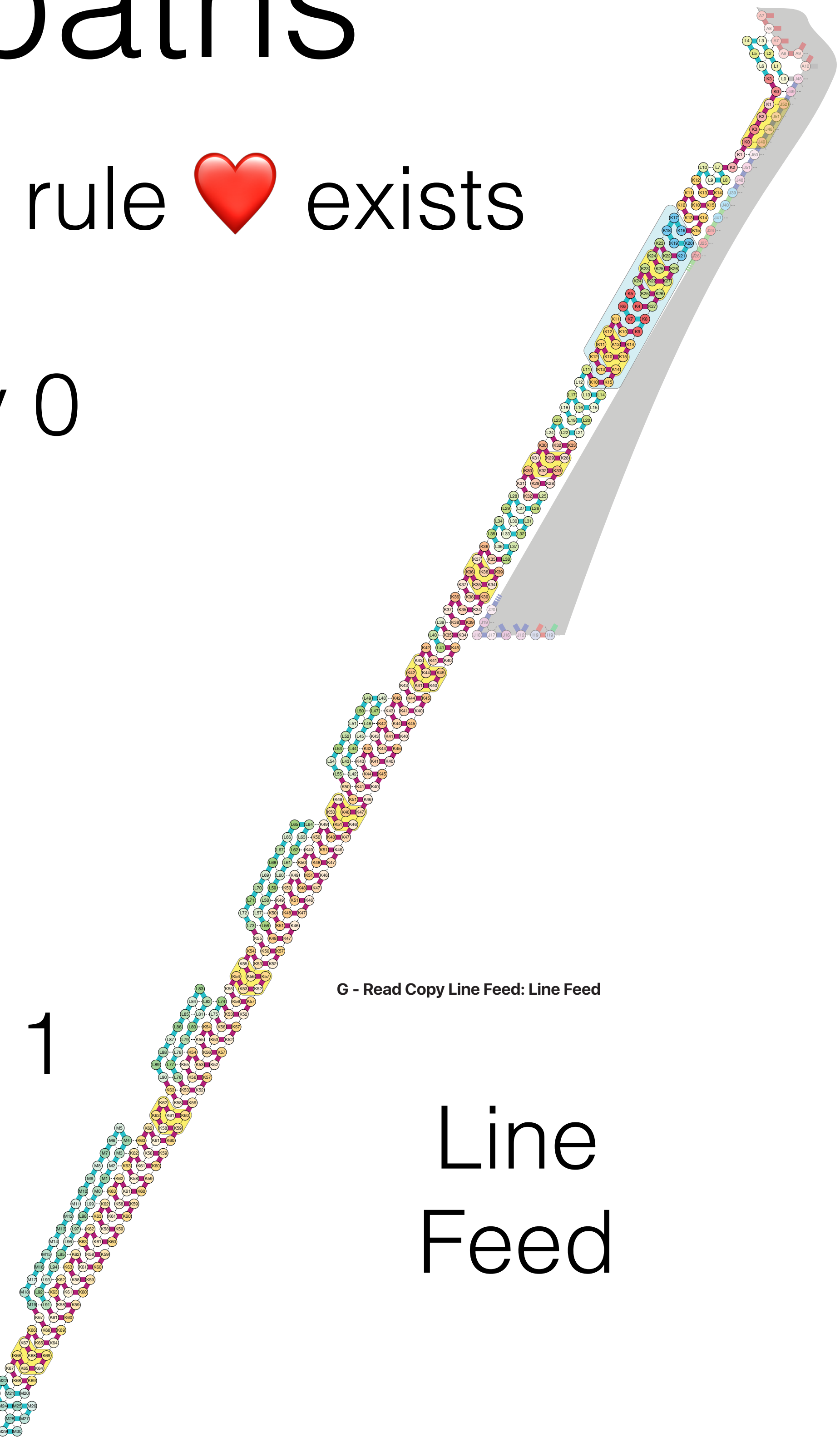


G - Read Copy Line Feed: Copy 0 (Zig)

Copy 0

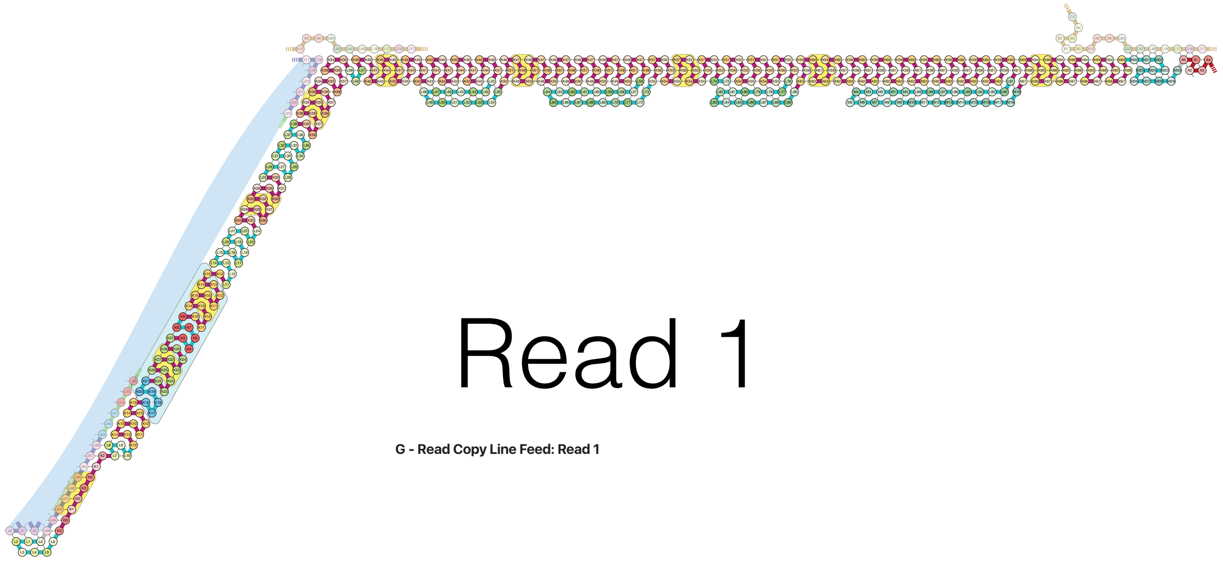
G - Read Copy Line Feed: Copy 1 (Zig)

Copy 1



G - Read Copy Line Feed: Line Feed

Line Feed



G - Read Copy Line Feed: Read 1

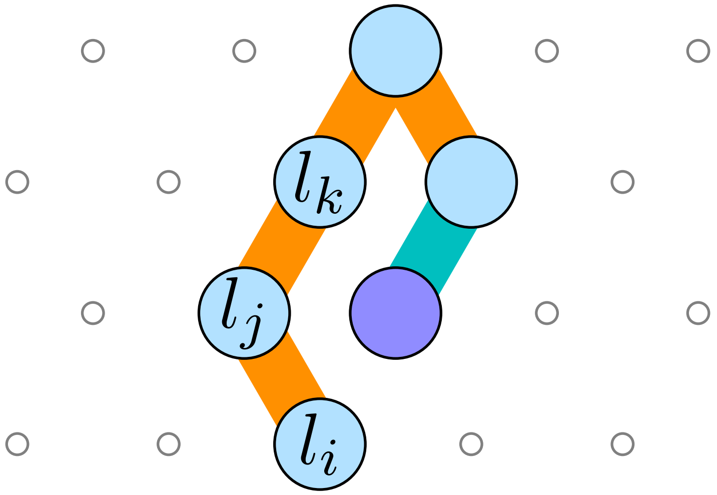
Read 1

Oritatami design is NP-hard

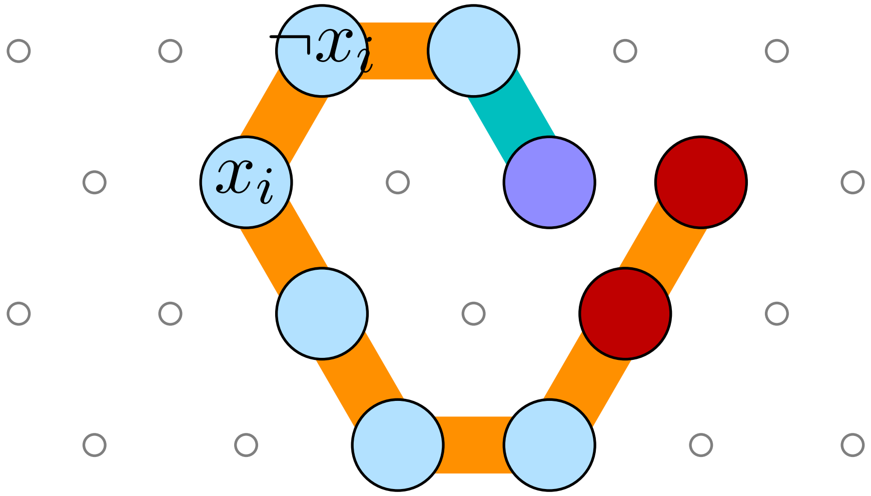
INPUT:	a delay time δ, a list of $n > 0$ seeds $\sigma_1, \sigma_2, \ldots, \sigma_n$, and a list of n conformations $c_1, c_2, \ldots, c_n$ of the same length l
OUTPUT:	an attraction rule $\heartsuit$ such that for all $i \in \{1, 2, \ldots, n\}$, Oritatami system $\mathcal{O}_i = (s, \sigma_i, \heartsuit, \delta)$ deterministically folds into conformation c_i, where s is the sequence of length l such that for all $i \in \{1, 2, \ldots, l\}$, $s_i = i$.

The reduction *(length=1, δ arbitrary)*

Ensures it binds to at least one litteral in $l_i \vee l_j \vee l_k$



Ensures it binds to at most one of x_i and $\neg x_i$



The second challenge: Designing the rule ❤️

Theorem. There is a **FPT algorithm** with respect to ***L*** that designs **in linear time in *L*** (but exponential in ***k*** and ***δ***) a **rule** ❤️ that folds the **sequence *1,…,L* of length *L*** into ***k*** **prescribed conformations** when folded in ***k*** **prescribed environments.**

Proof. • **Locality:** each bead only sees a bounded number (exponential in δ) of other beads when folded.

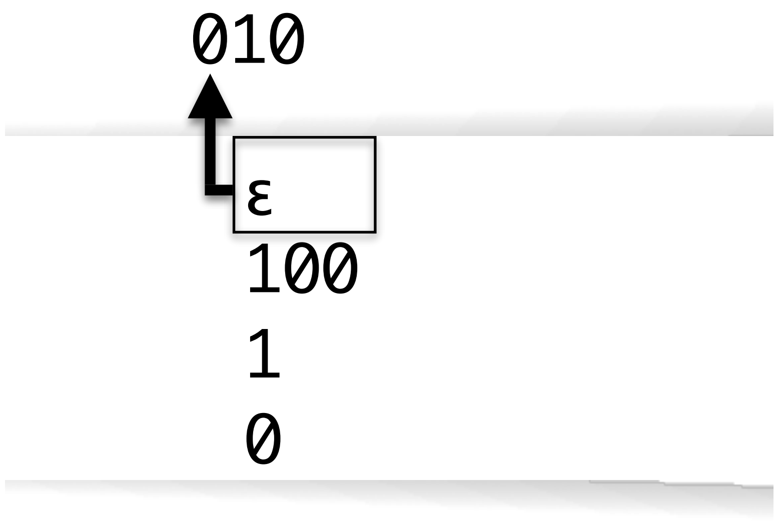
- Then, compute all valid local rules for each of these neighborhoods
- And use dynamic programming to decide whether there is a global rule compatible with at least one of the local rule for each environment.

Oritatami is Turing complete

Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



```
010
ε
100
1
0
```

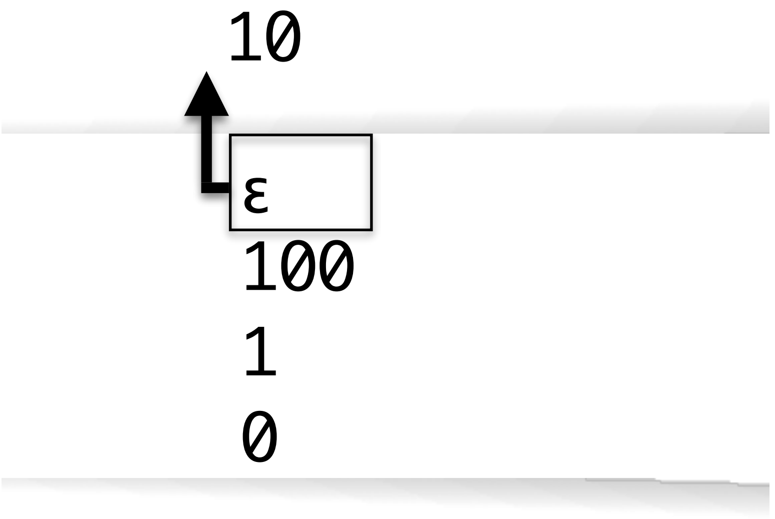
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

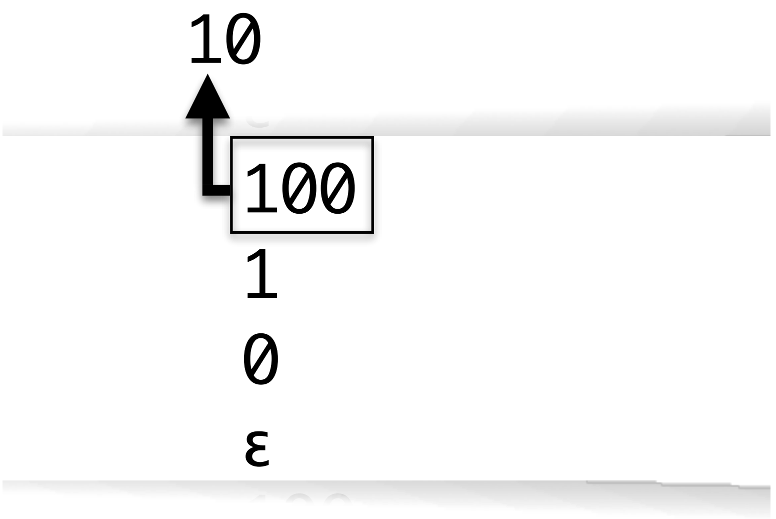
- **Dynamics:**

    - *If the tape word is empty (ε):* halt

    - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p

    - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



10

100

1

0

ε

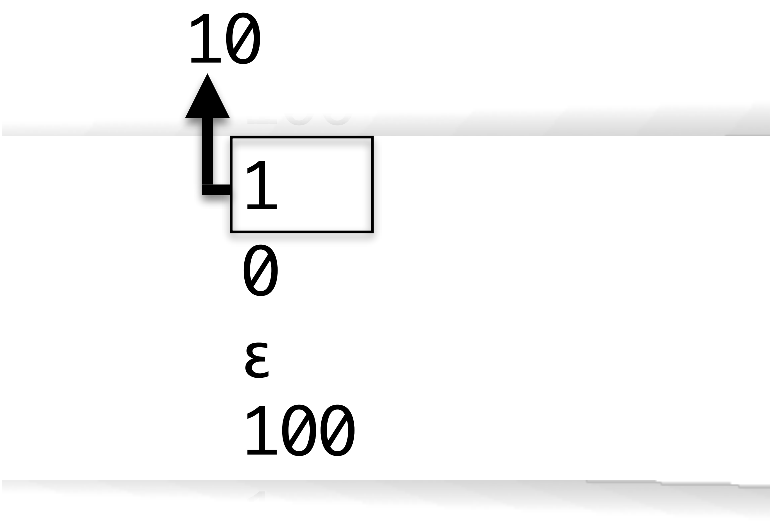
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

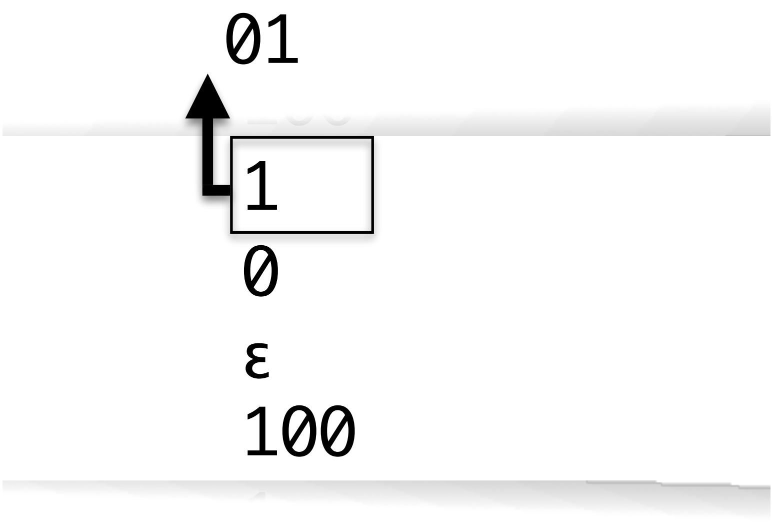
Example.



Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



01

1

0

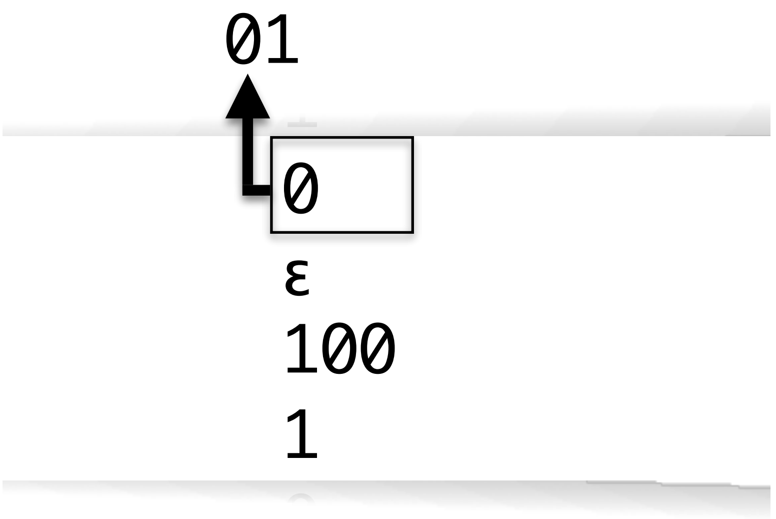
ε

100

Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
    - *If the tape word is empty (ε):* halt
    - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
    - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



01

0

ε

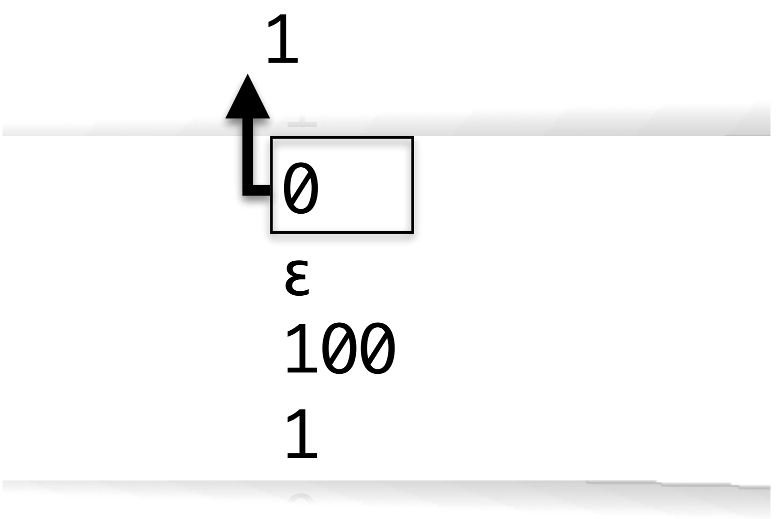
100

1

Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



1

0

ε

100

1

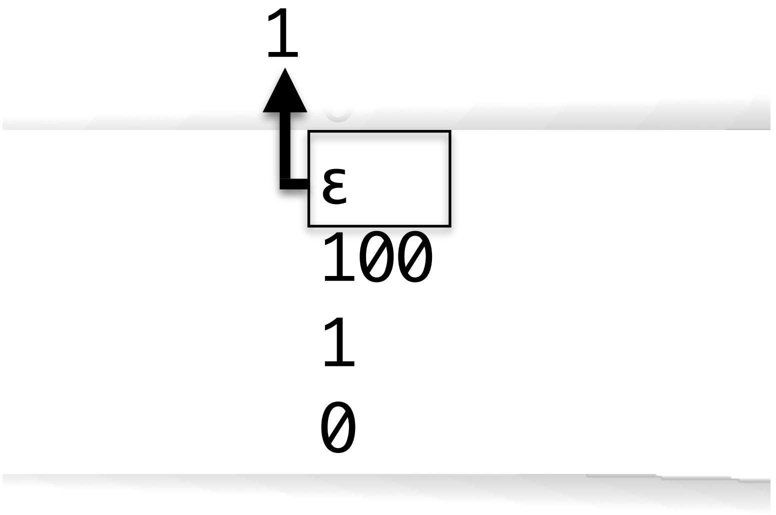
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

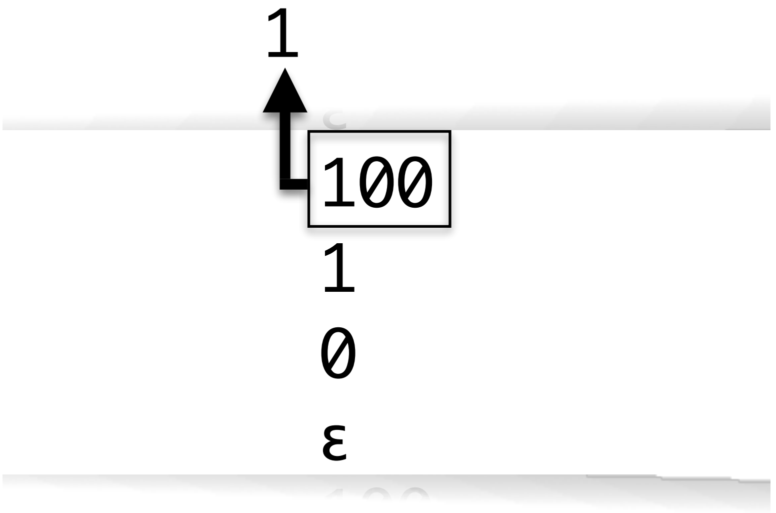
Example.



Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

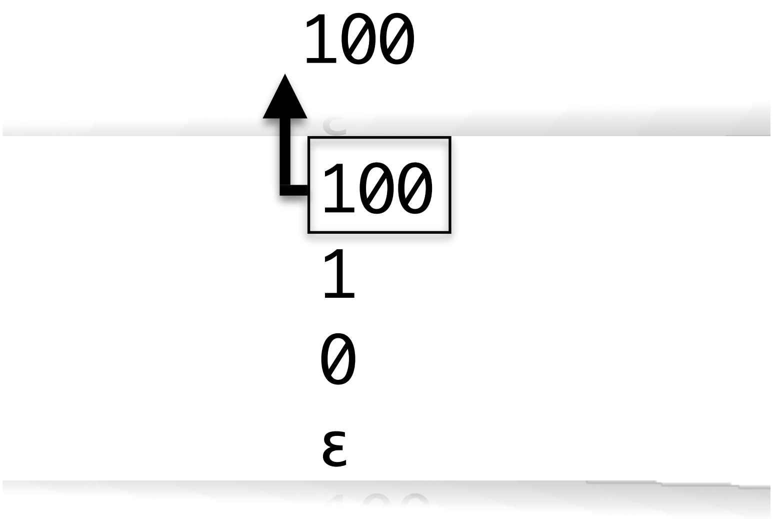
Example.



Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



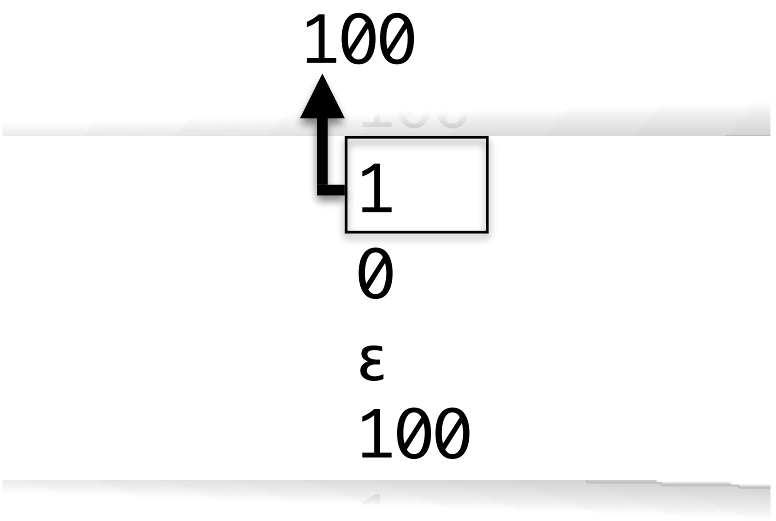
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
    - *If the tape word is empty (ε):* halt
    - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
    - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



100

1

0

ε

100

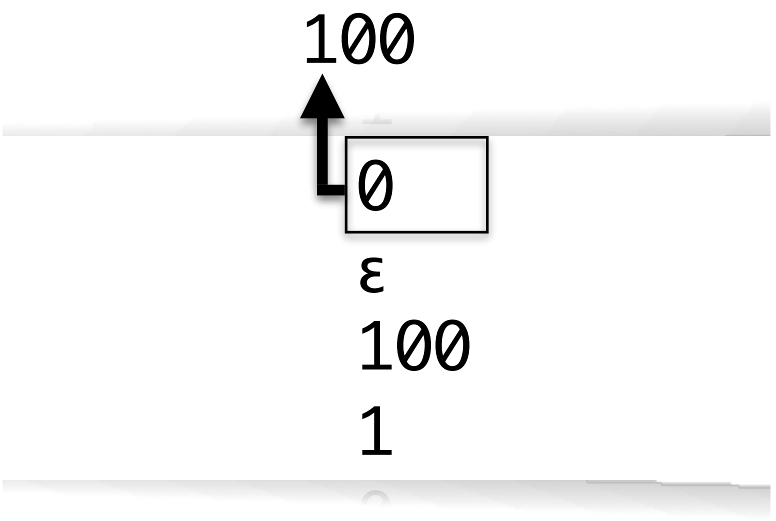
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



100

0

ε

100

1

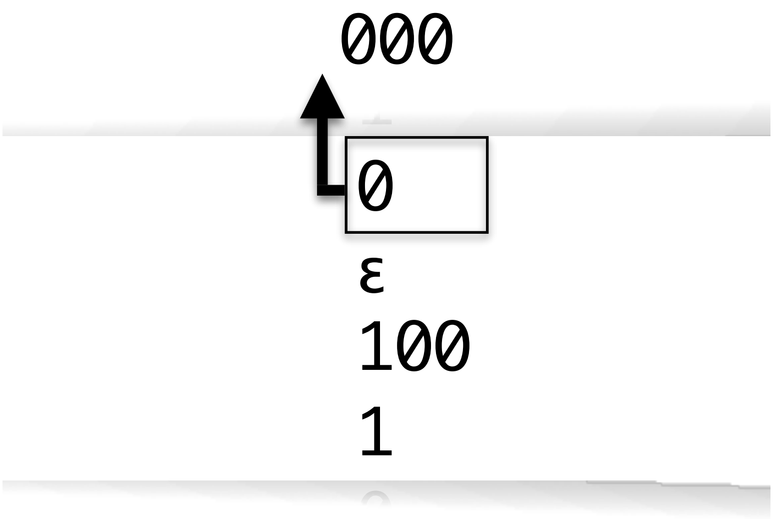
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



000

0

ε

100

1

Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

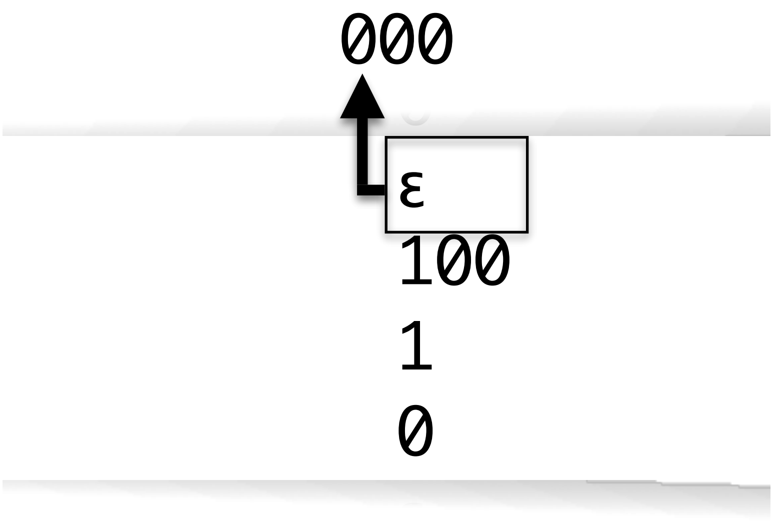
- **Dynamics:**

    - *If the tape word is empty (ε):* halt

    - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p

    - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



000

ε

100

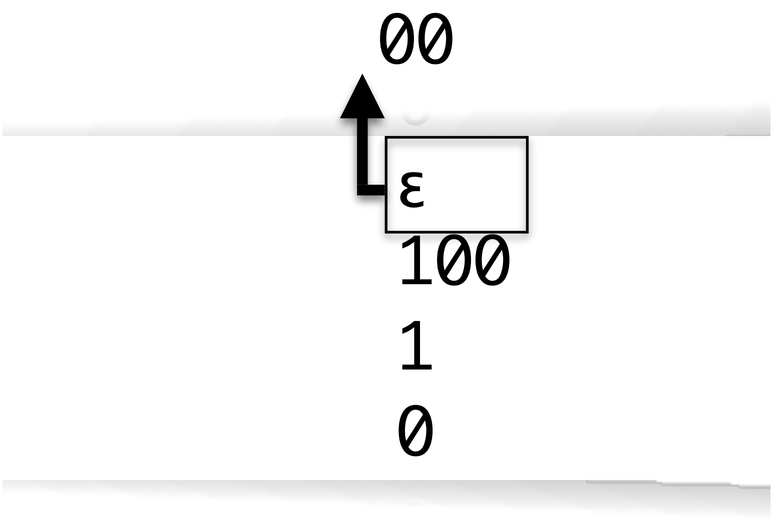
1

0

Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



00

ε

100

1

0

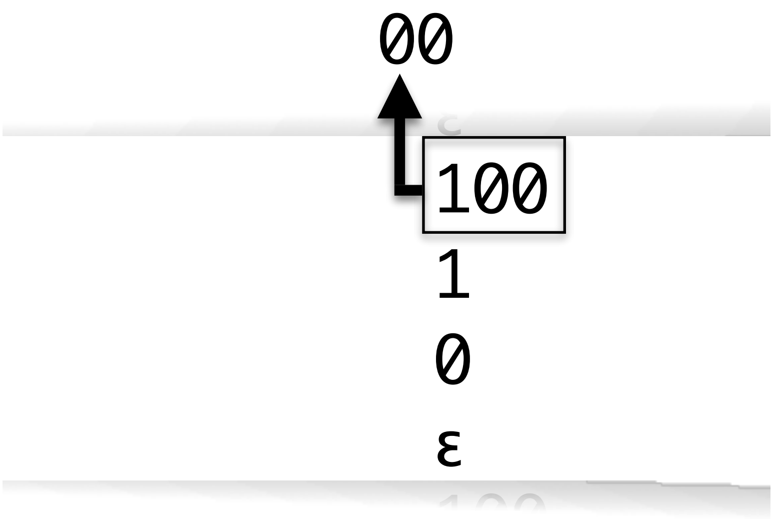
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

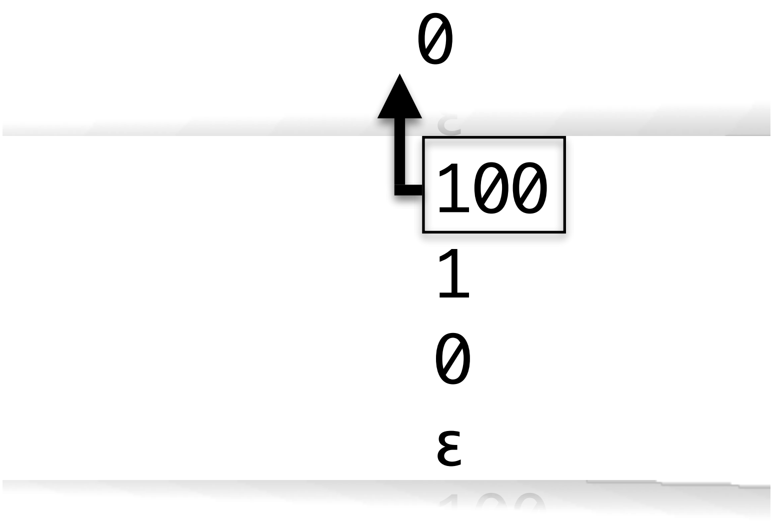
Example.



Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



0

100

1

0

ε

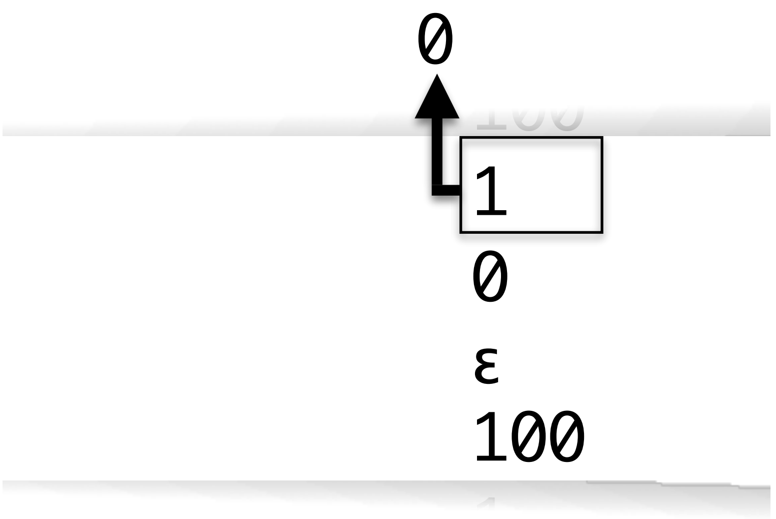
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



0

100

1

0

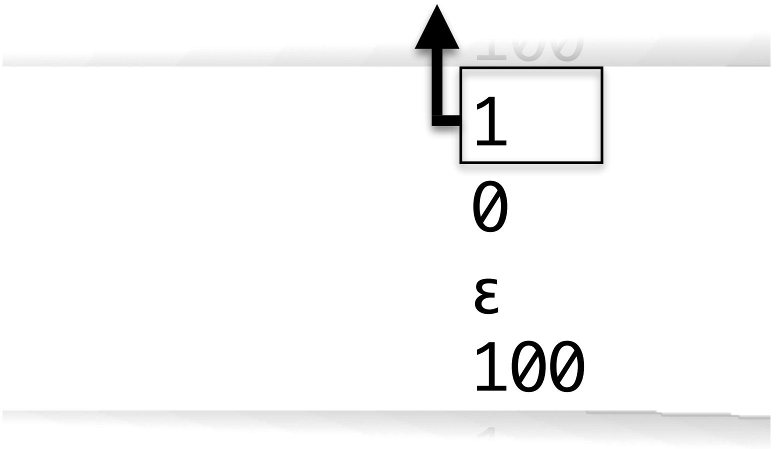
ε

100

Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them
- An initial binary **tape word** (the input)
- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



1

0

ε

100

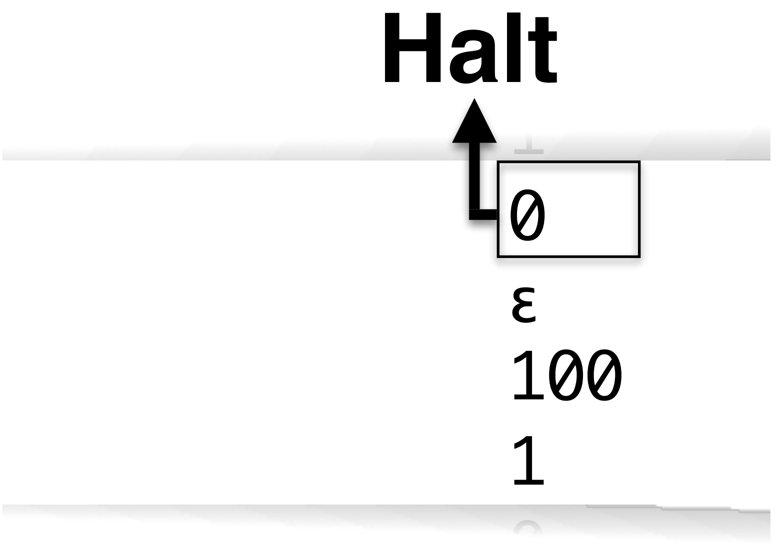
Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

Example.



Skipping Cyclic Tag Systems

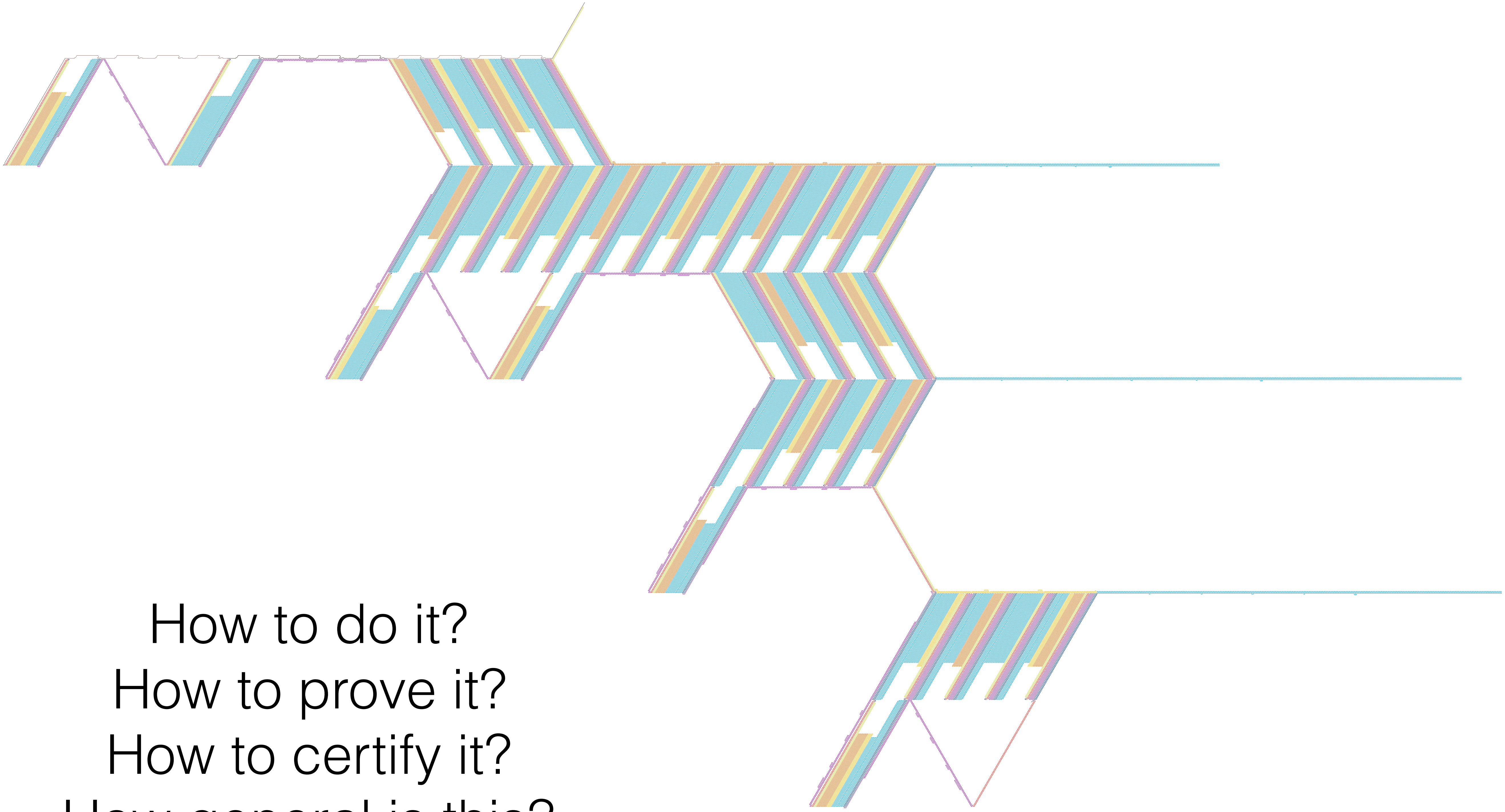
- A finite **cyclic sequence** of finite binary **code words** with a pointer p to one of them

- An initial binary **tape word** (the input)

- **Dynamics:**
  - *If the tape word is empty (ε):* halt
  - *If the 1st letter of the tape word is 0:* delete the 0 and increment the pointer p
  - *If the 1st letter of the tape word is 1:* delete the 1, append to the tape word the code word at position p+1 and increase p by +2

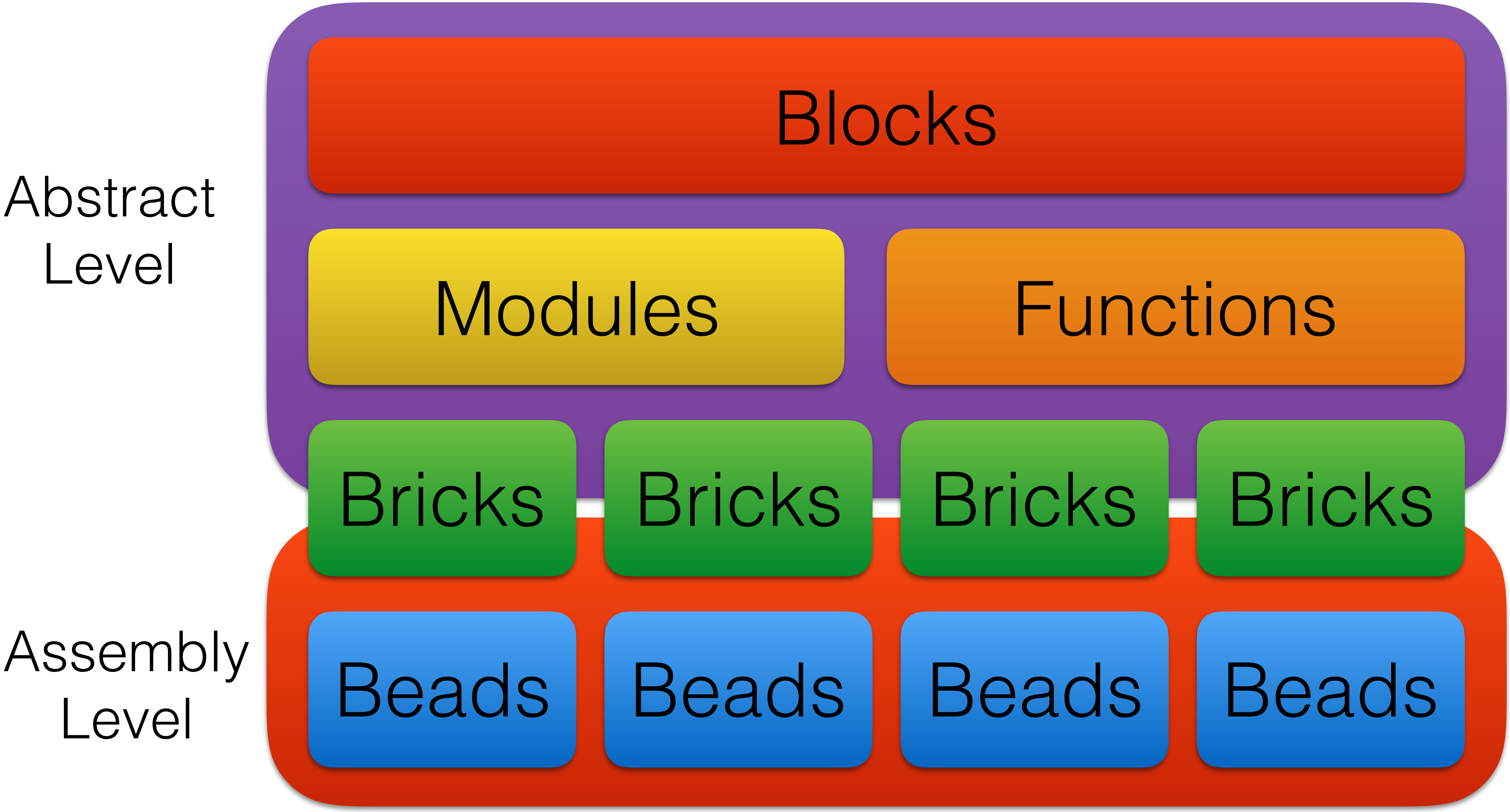
- **Theorem** [Neary, Woods, 2006] Cyclic tag systems simulate any Turing machine with only a **quadratic** slow down

The simulation



How to do it?
How to prove it?
How to certify it?
How general is this?

A general programming framework



Abstract Level

Blocks

Modules

Functions

Bricks

Bricks

Bricks

Bricks

Assembly Level

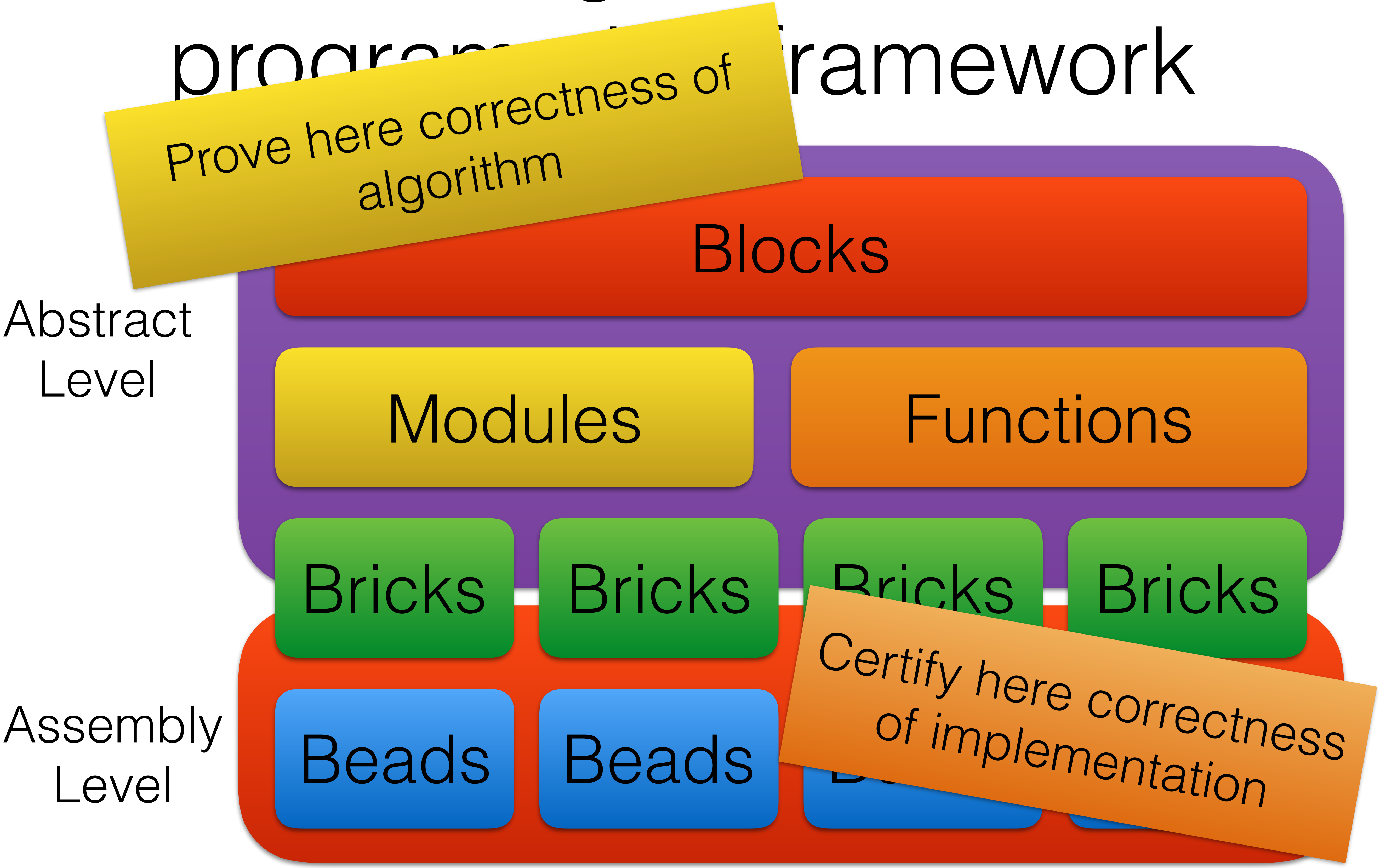
Beads

Beads

Beads

Beads

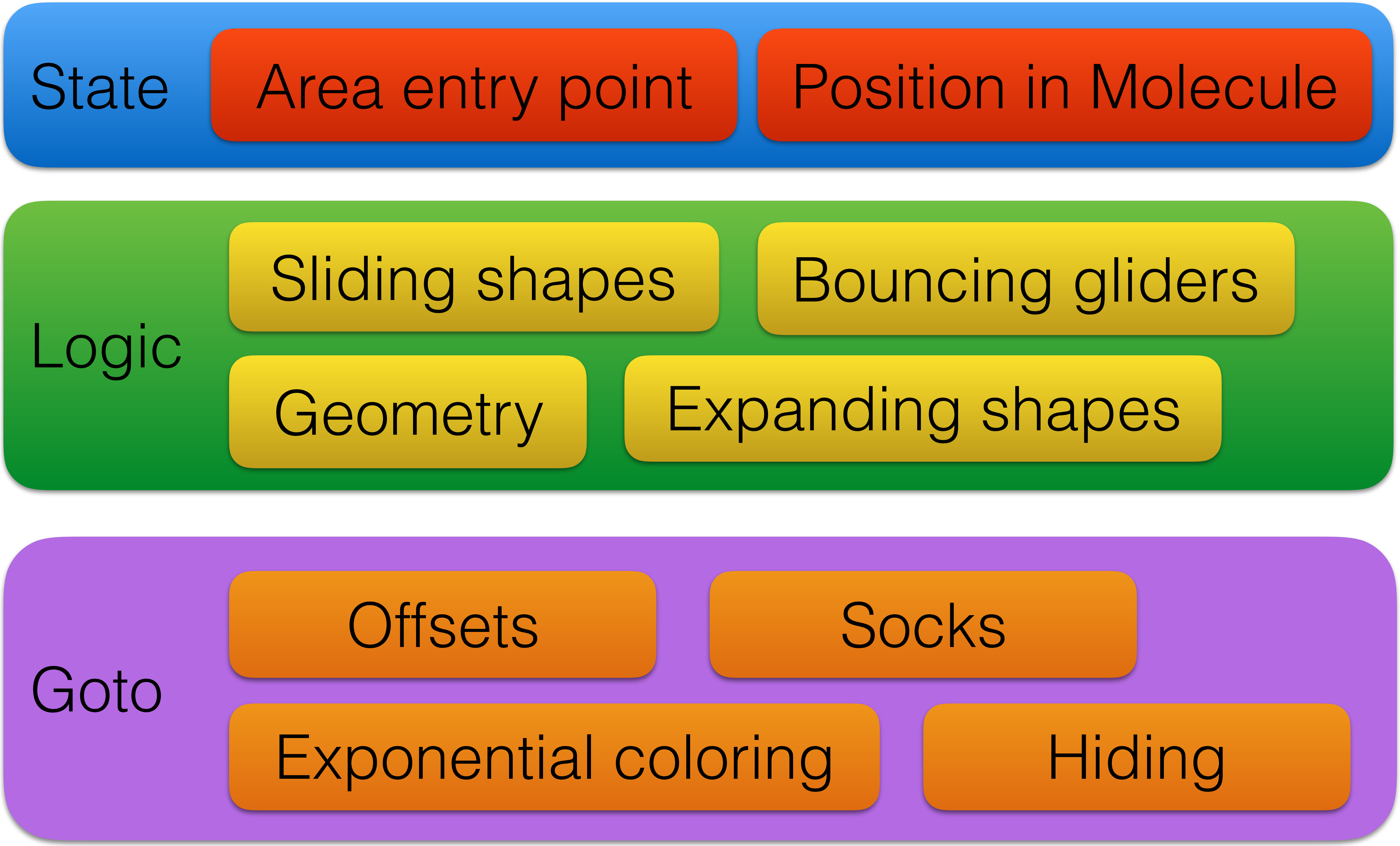
A general programming framework



Prove here correctness of algorithm

Certify here correctness of implementation

General programming tools



State

Area entry point

Position in Molecule

Logic

Sliding shapes

Bouncing gliders

Geometry

Expanding shapes

Goto

Offsets

Socks

Exponential coloring

Hiding

Back to our simulation

Trimmed space-time diagram

Consider the following productions: $p = \langle \overset{[0]}{\texttt{110}}, \overset{[1]}{\epsilon}, \overset{[2]}{\texttt{11}}, \overset{[3]}{\texttt{0}} \rangle$

$$ {}^{[0]}\texttt{010} \to {}^{[1]}\texttt{10} \xrightarrow[{[2]}:\texttt{11}]{\text{Append}} {}^{[3]}\texttt{011} \to {}^{[0]}\texttt{11} \xrightarrow[{[1]}:\epsilon]{\text{Append}} {}^{[2]}\texttt{1} \xrightarrow[{[3]}:\texttt{0}]{\text{Append}} {}^{[0]}\texttt{0} \to {}^{[1]}\ \texttt{Halt} $$

Trimmed space-time diagram

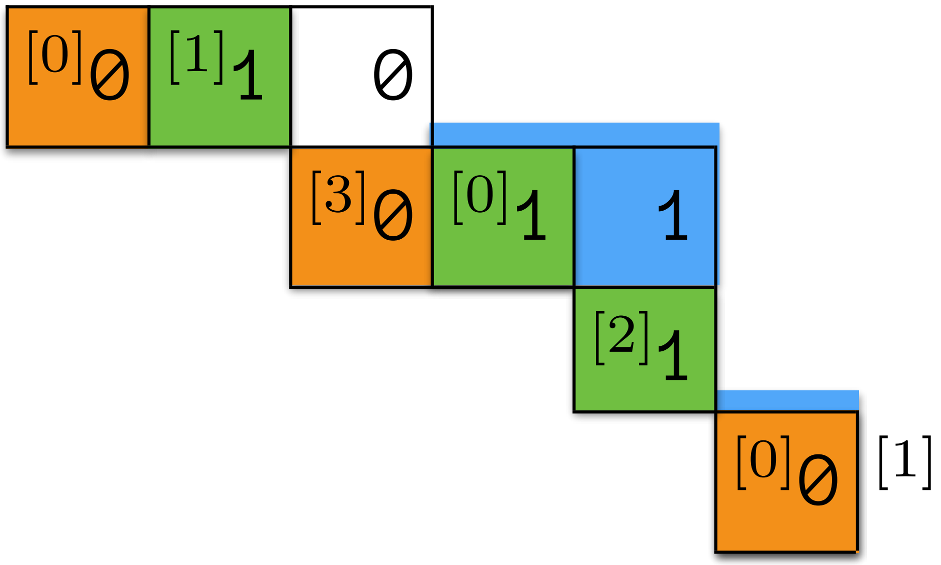
Consider the following productions: $p = \langle \overset{[0]}{\texttt{110}}, \overset{[1]}{\epsilon}, \overset{[2]}{\texttt{11}}, \overset{[3]}{\texttt{0}} \rangle$

$${}^{[0]}\texttt{010} \rightarrow {}^{[1]}\texttt{10} \xrightarrow[{[2]}:\texttt{11}]{\text{Append}} {}^{[3]}\texttt{011} \rightarrow {}^{[0]}\texttt{11} \xrightarrow[{[1]}:\epsilon]{\text{Append}} {}^{[2]}\texttt{1} \xrightarrow[{[3]}:\texttt{0}]{\text{Append}} {}^{[0]}\texttt{0} \rightarrow {}^{[1]}\ \texttt{Halt}$$

Trimmed space-time diagram

Consider the following productions: $p = \langle \overset{[0]}{\texttt{110}}, \overset{[1]}{\epsilon}, \overset{[2]}{\texttt{11}}, \overset{[3]}{\texttt{0}} \rangle$

$${}^{[0]}\texttt{010} \to {}^{[1]}\texttt{10} \xrightarrow[{[2]:\texttt{11}}]{\text{Append}} {}^{[3]}\texttt{011} \to {}^{[0]}\texttt{11} \xrightarrow[{[1]:\epsilon}]{\text{Append}} {}^{[2]}\texttt{1} \xrightarrow[{[3]:\texttt{0}}]{\text{Append}} {}^{[0]}\texttt{0} \to {}^{[1]}\ \texttt{Halt}$$



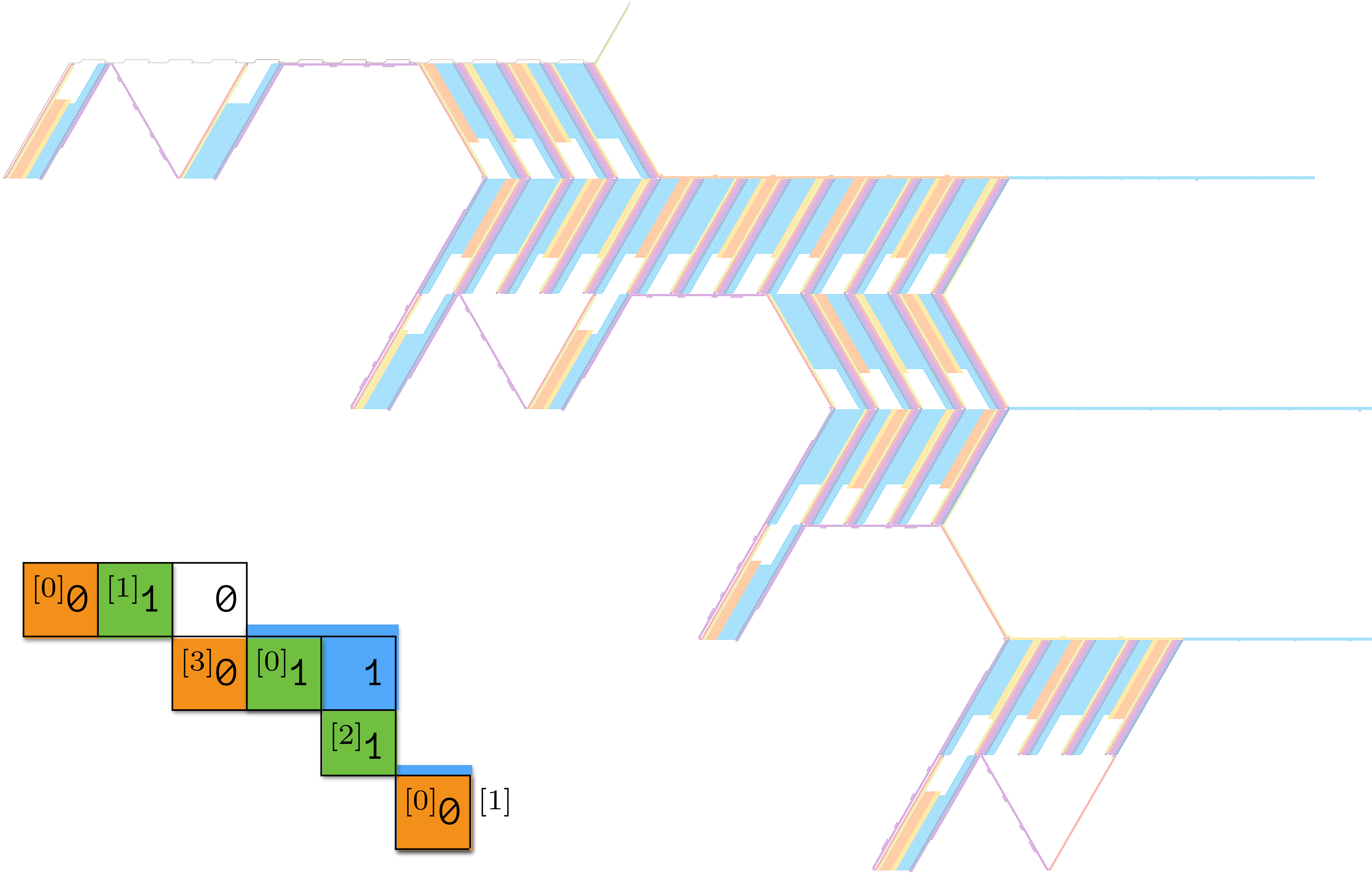
⟶ Append $[2]:\texttt{11}$

⟶ Append $[1]:\epsilon$

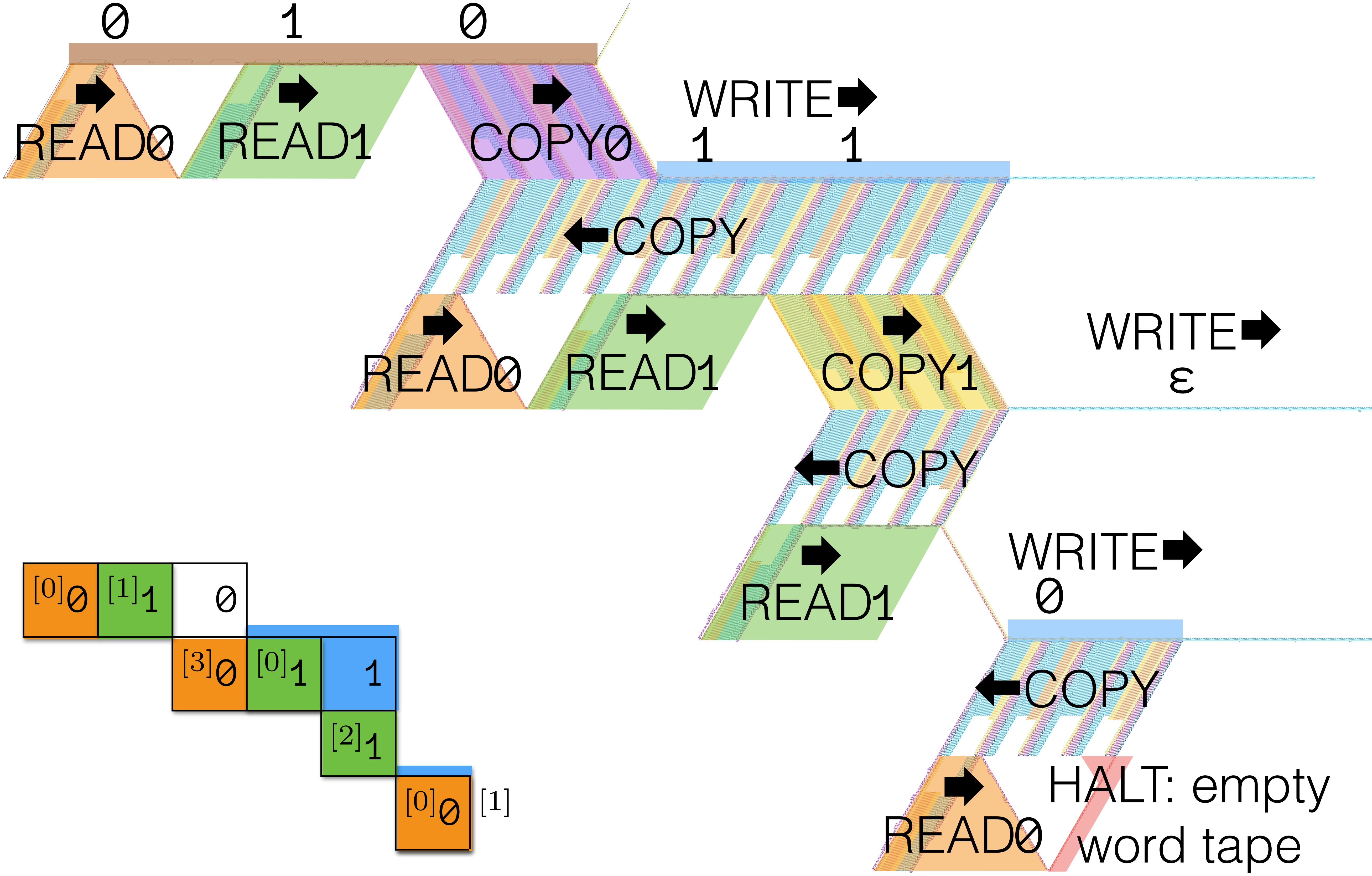
⟶ Append $[3]:\texttt{0}$

⟶ Halt $[1]:\epsilon$

The simulation



The simulation



Initial word tape:

0 1 0

➡ READ0

➡ READ1

➡ COPY0

WRITE➡

1 1

⬅COPY

➡ READ0

➡ READ1

➡ COPY1

WRITE➡

ε

⬅COPY

➡ READ1

WRITE➡

0

⬅COPY

➡ READ0

HALT: empty word tape

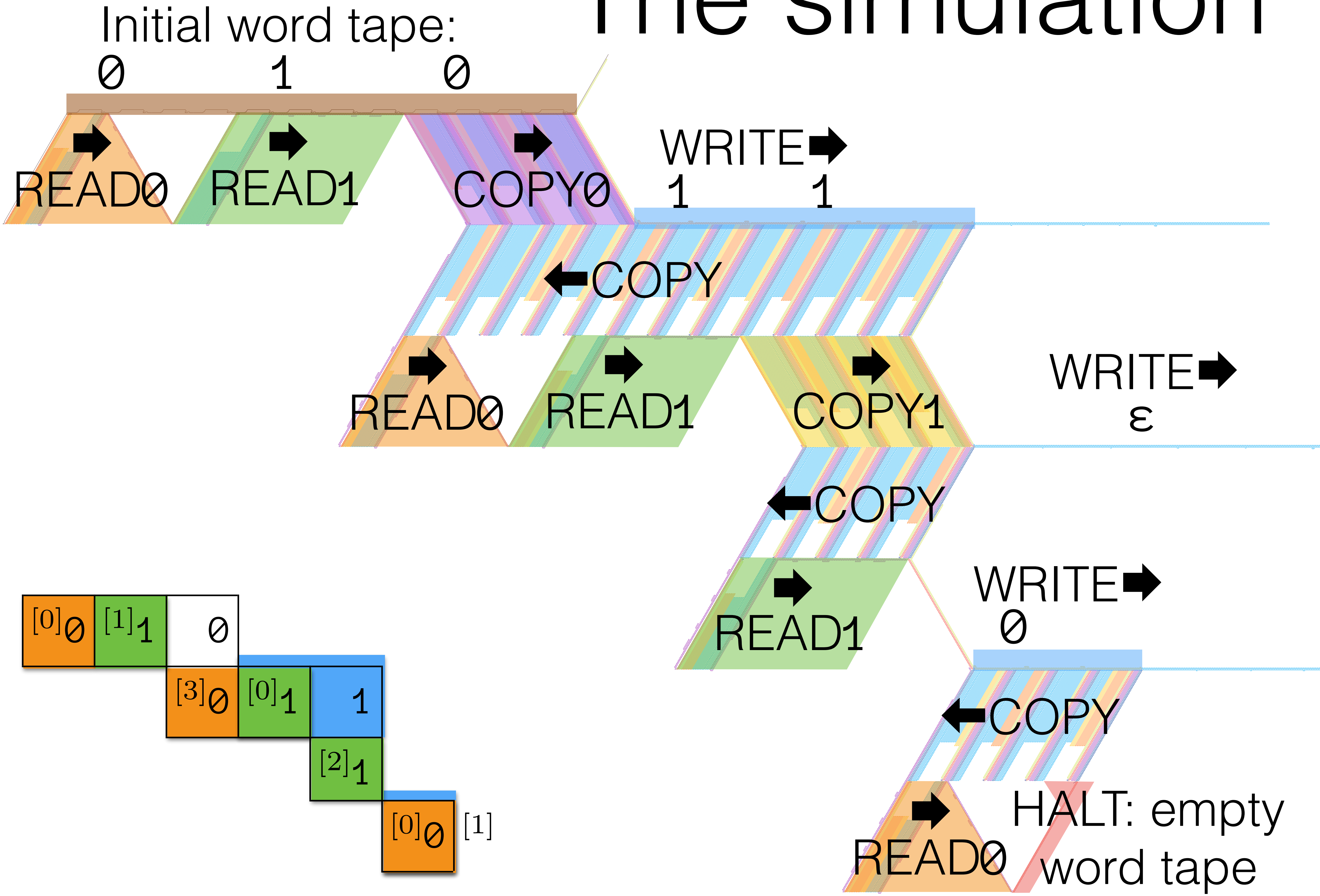
[0]0 [1]1 0

[3]0 [0]1 1

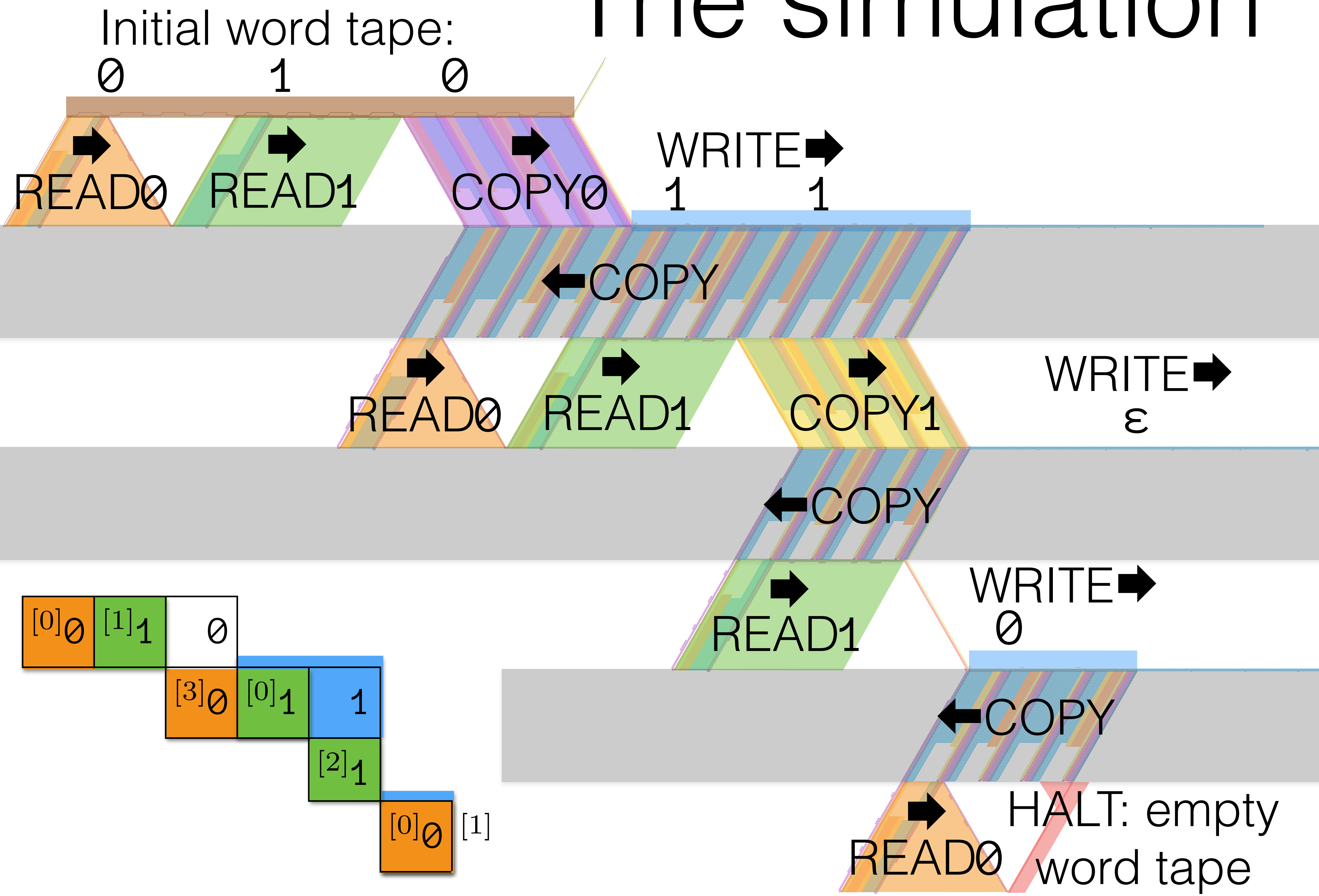
[2]1

[0]0 [1]

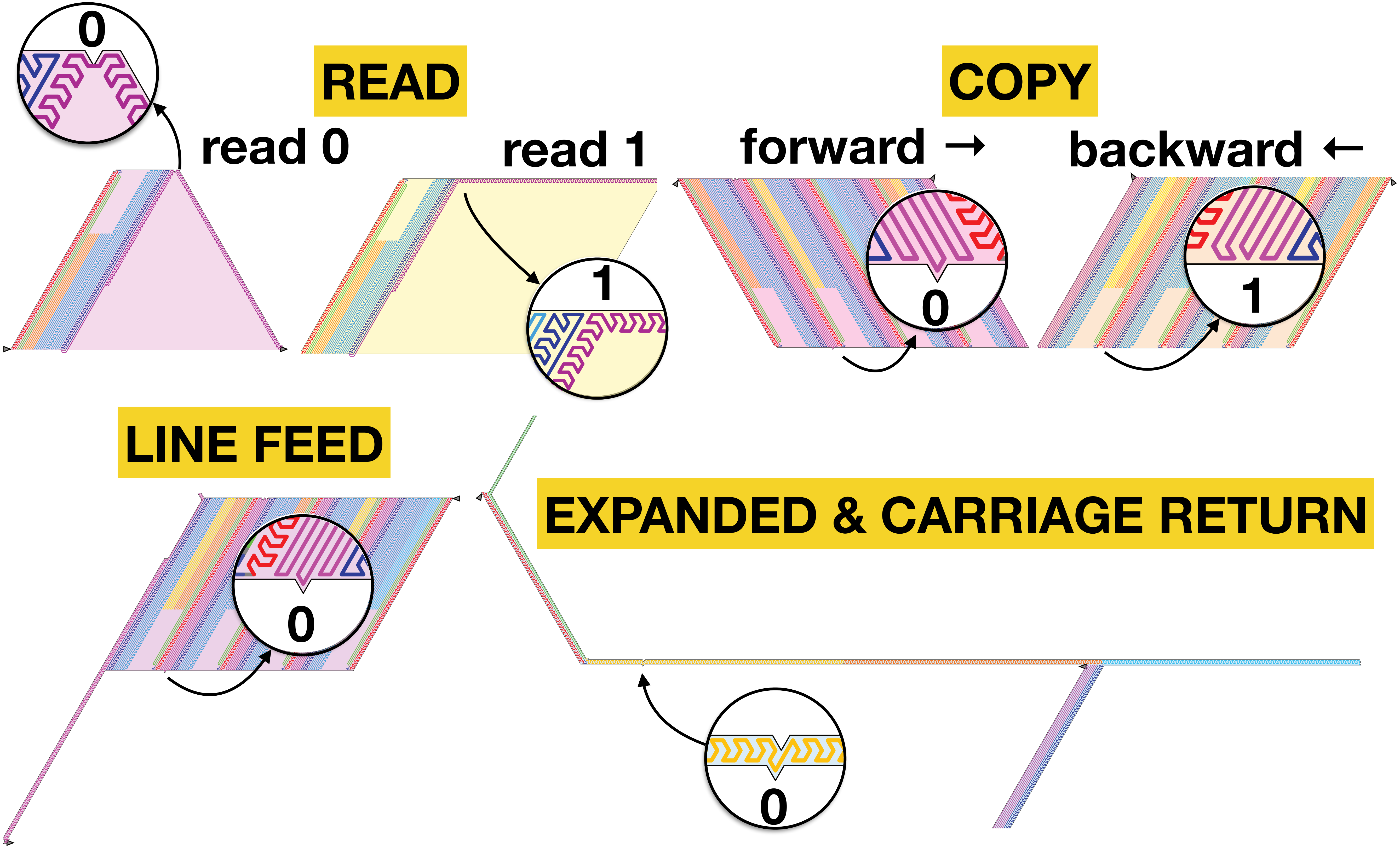
The simulation



The simulation



The blocks



READ

read 0

read 1

COPY

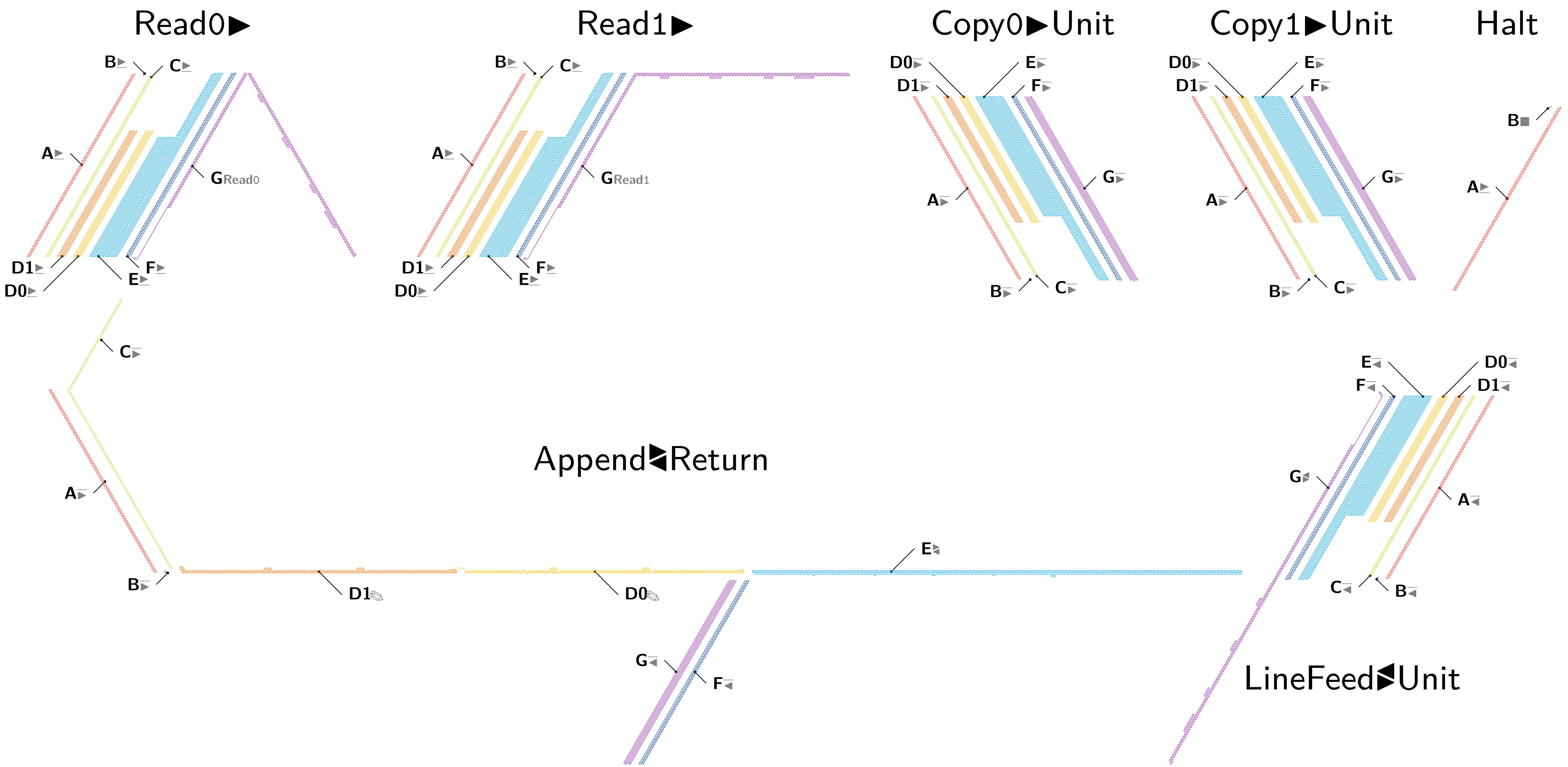
forward →

backward ←

LINE FEED

EXPANDED & CARRIAGE RETURN

The bricks inside each block

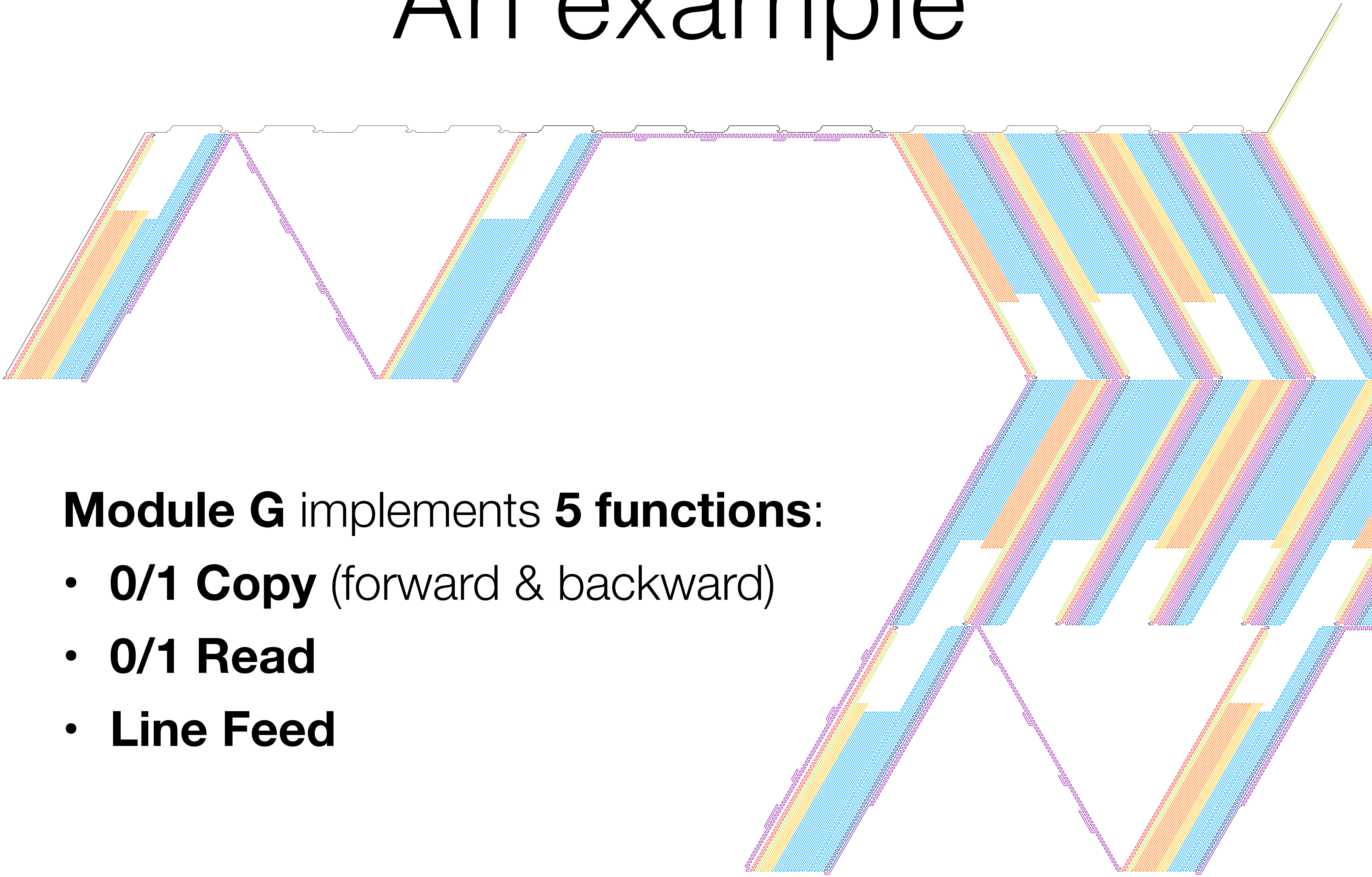


How do we implement several functions

(i.e. folding into different bricks)

in a given module?

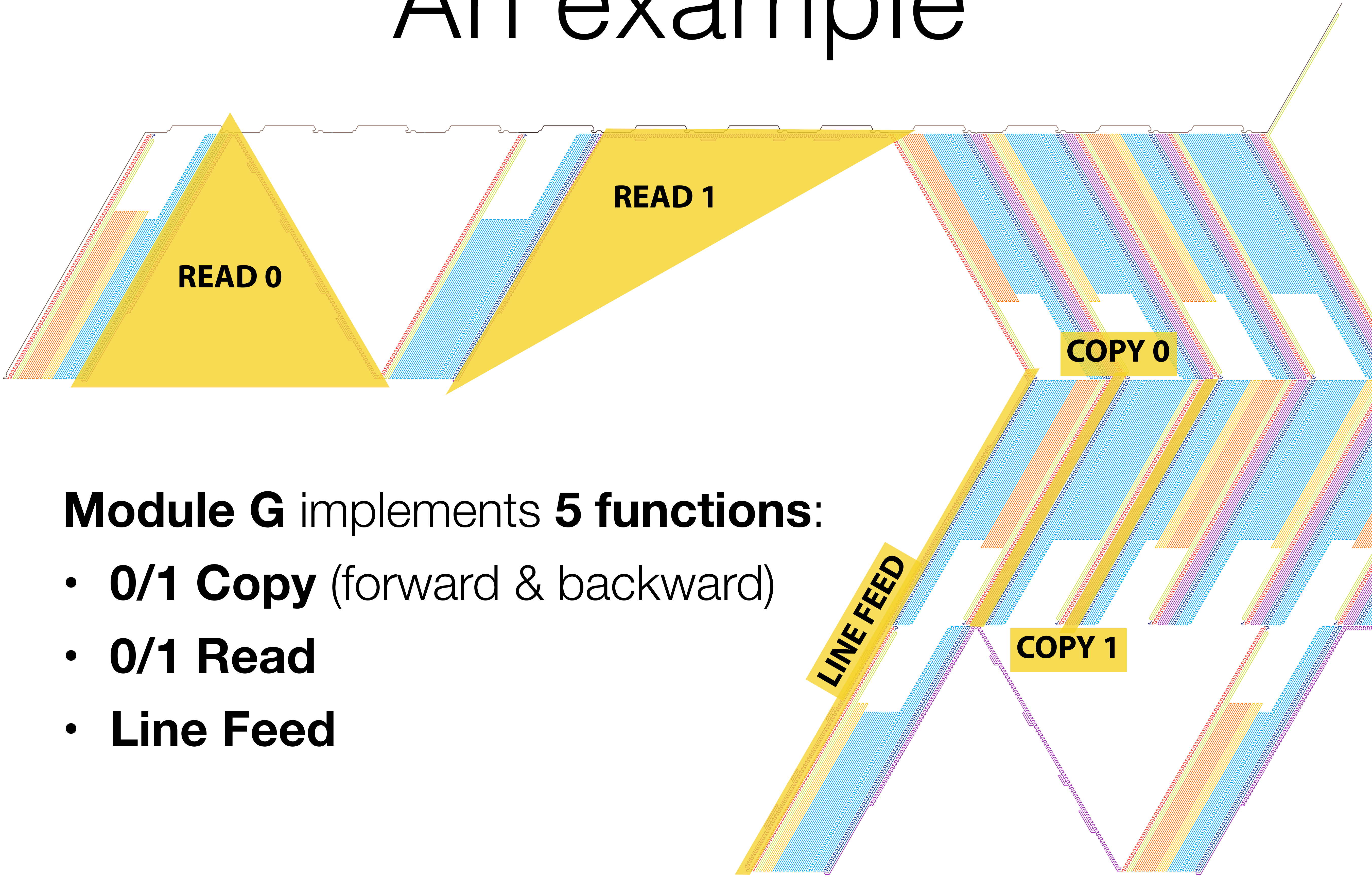
An example



Module G implements **5 functions**:

- **0/1 Copy** (forward & backward)
- **0/1 Read**
- **Line Feed**

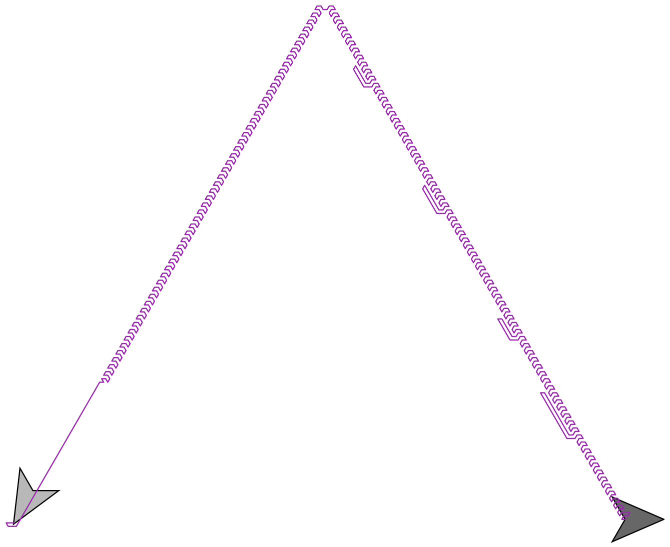
An example



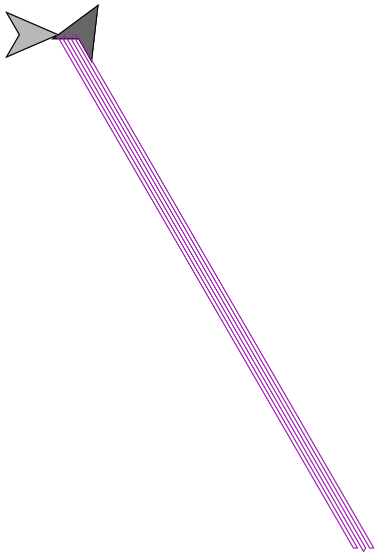
Module G implements **5 functions**:

- **0/1 Copy** (forward & backward)
- **0/1 Read**
- **Line Feed**

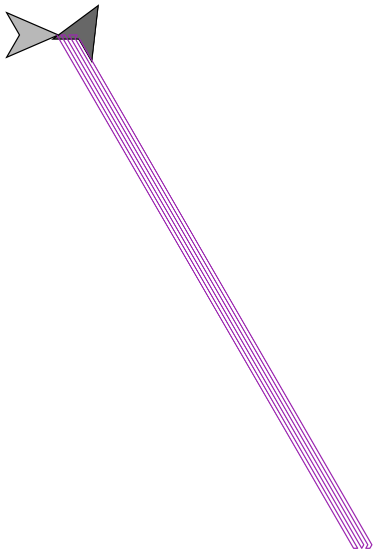
The various bricks for module G



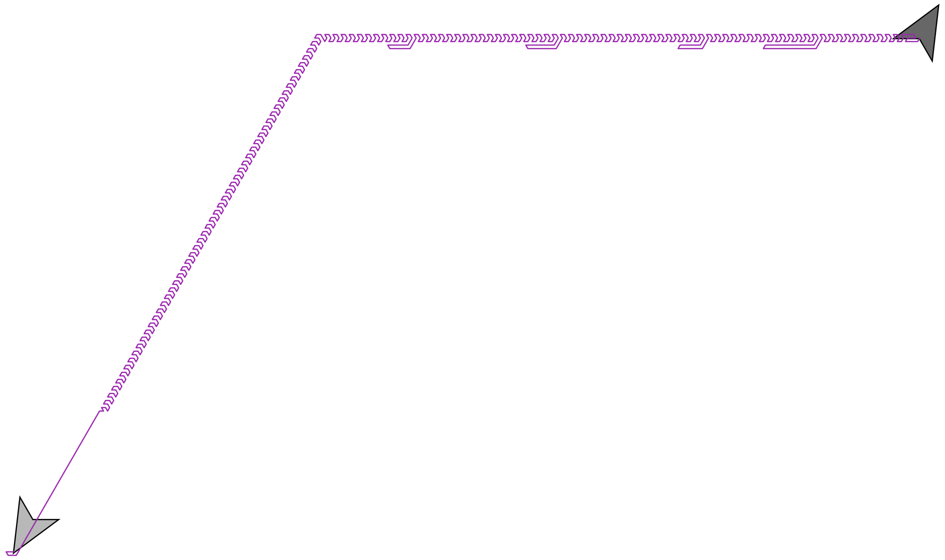
(a) The brick $\boxed{\text{G}\,\underline{\blacktriangleright}\,\textbf{Read0}}$.



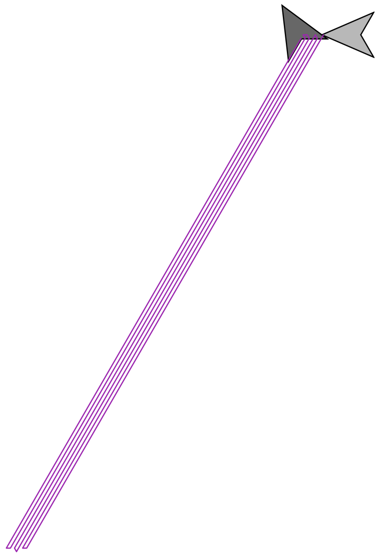
(a) The brick $\boxed{\text{G}\,\overline{\blacktriangleright}\,\textbf{Copy0}}$.



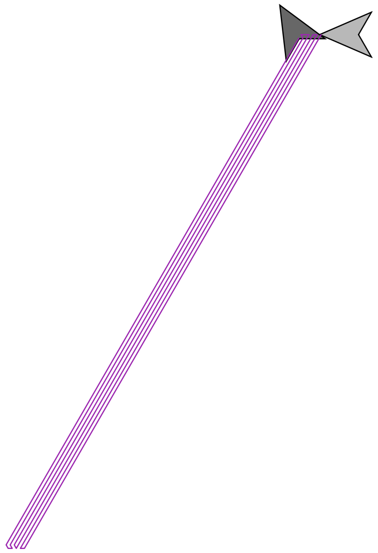
(b) The brick $\boxed{\text{G}\,\overline{\blacktriangleright}\,\textbf{Copy1}}$.



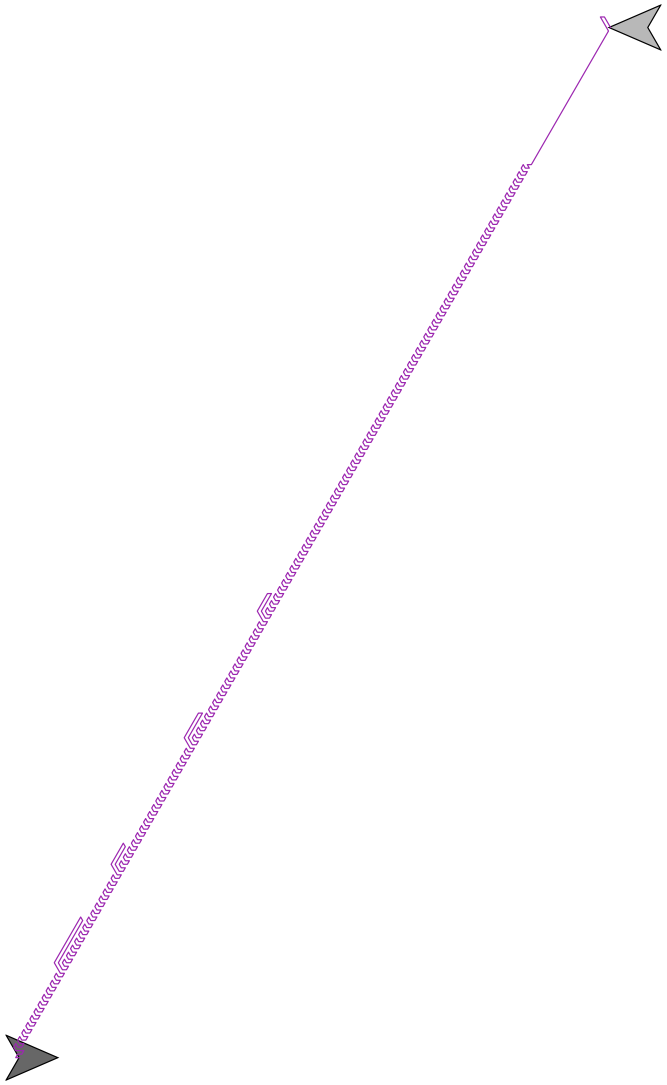
(b) The brick $\boxed{\text{G}\,\underline{\blacktriangleright}\,\textbf{Read1}}$.



(c) The brick $\boxed{\text{G}\,\overline{\blacktriangleleft}\,\textbf{Copy0}}$.

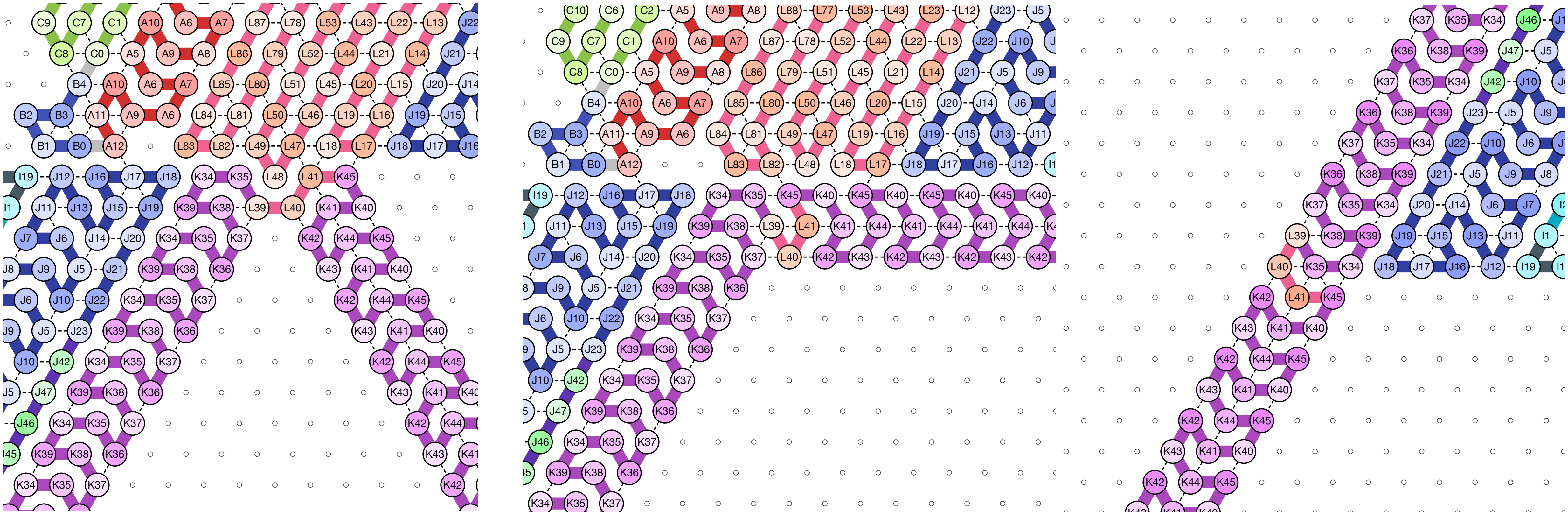


(d) The brick $\boxed{\text{G}\,\overline{\blacktriangleright}\,\textbf{Copy1}}$.



(a) The brick $\boxed{\text{G}\,⧓\,\textbf{LineFeed}}$.

Glider turns



(a) G bounces southeastward in presence of a spike encoding a 0 and folds into G ⊵ Read0.

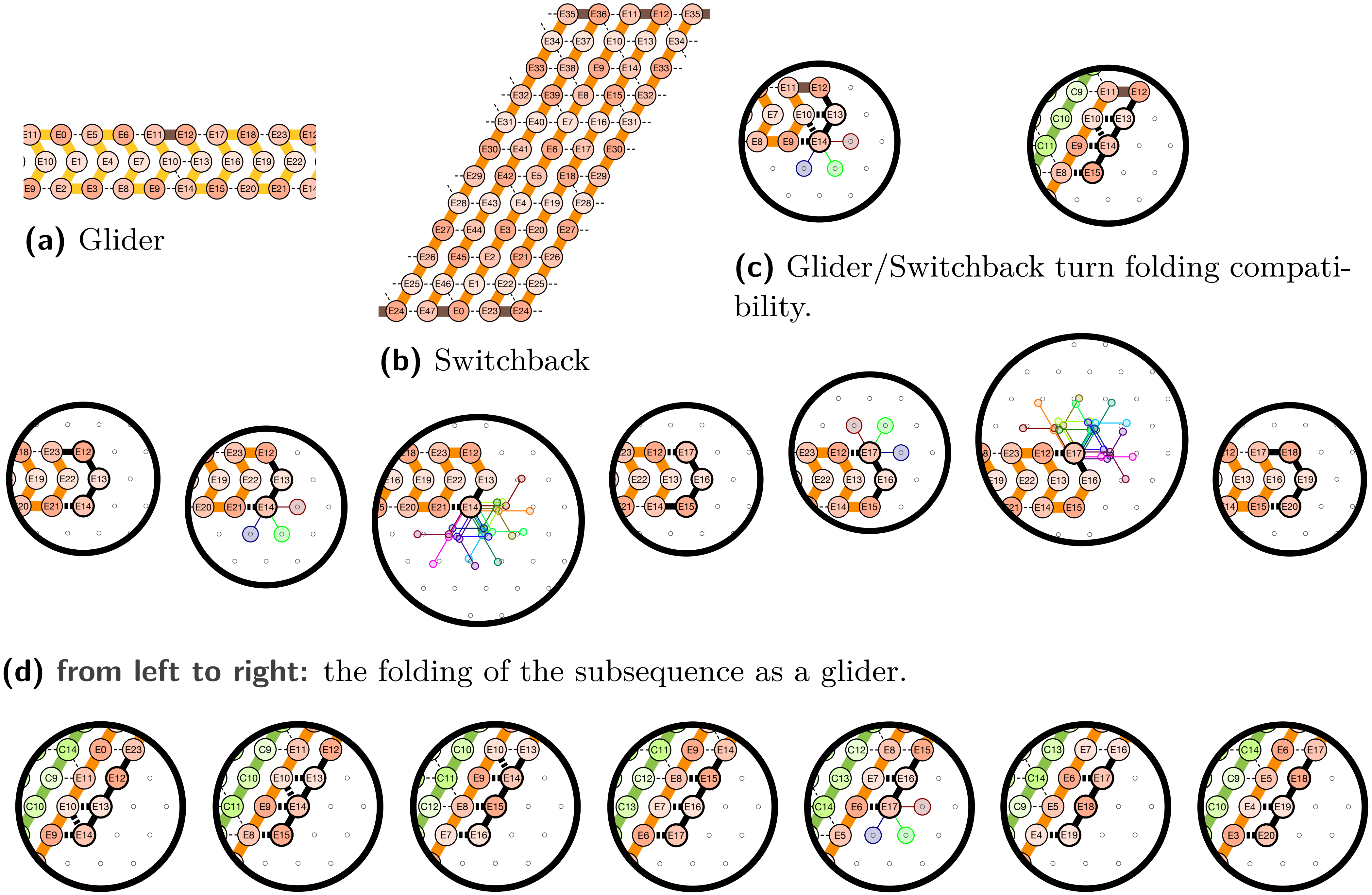
(b) G bounces eastward on a flat surface encoding a 1, and folds into G ⊵ Read1.

(c) G goes straight southwestward in absence of obstacle, and folds into G ⋬ LineFeed.

Some programming paradigms

- Switchback expanding in Gliders
- Offset
- Exponential coloring
- Socks

Switchback expanding into gliders



(a) Glider

(b) Switchback

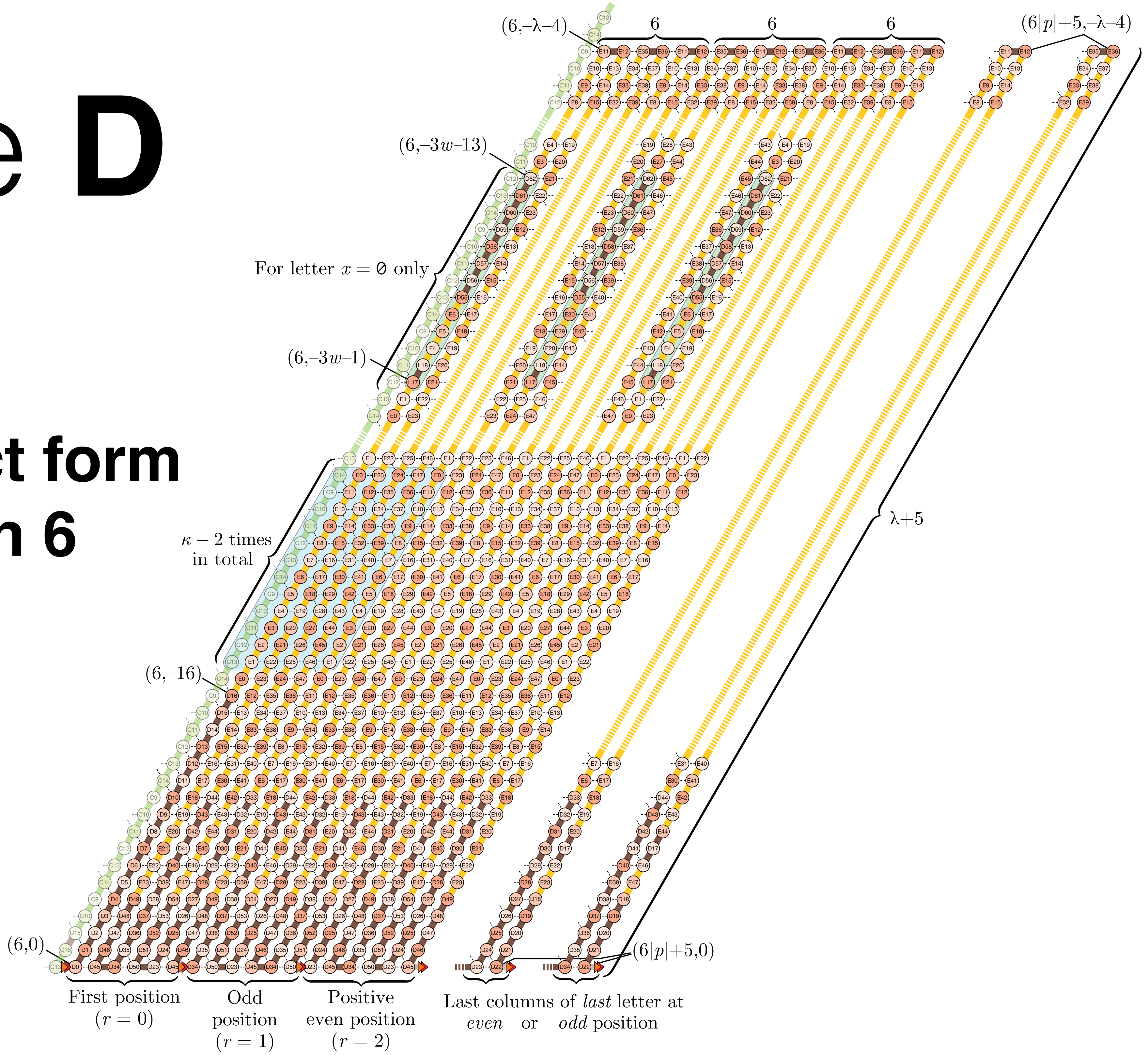
(c) Glider/Switchback turn folding compatibility.

(d) **from left to right:** the folding of the subsequence as a glider.

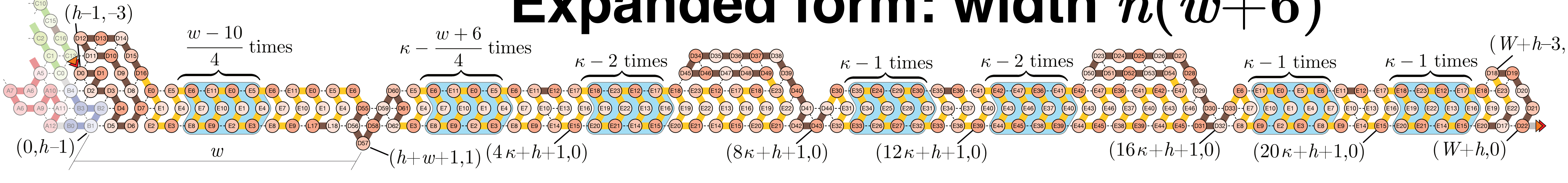
(e) **from left to right:** the folding of the subsequence as a switchback.

Module **D**

Compact form width 6



Expanded form: width $n(w+6)$



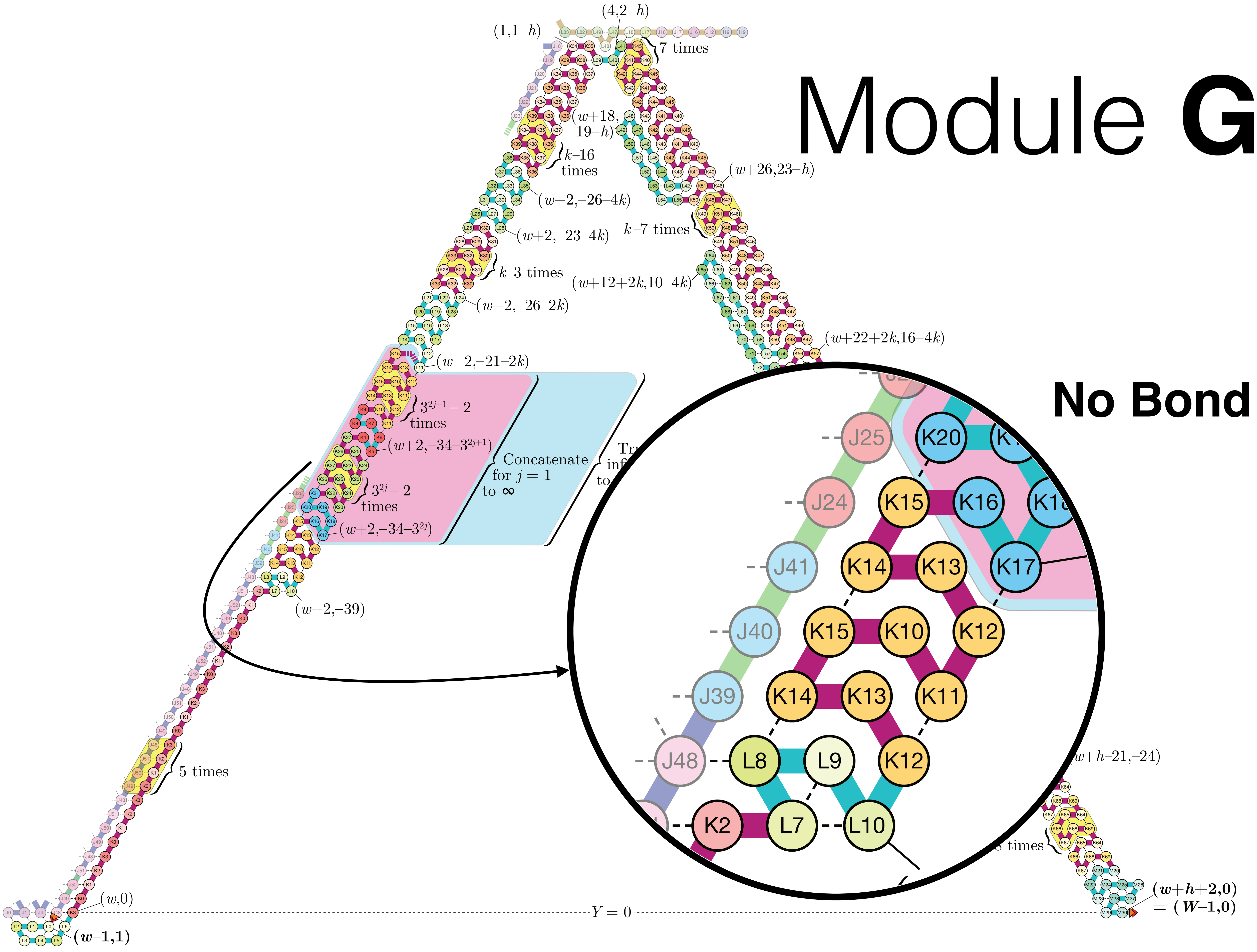
Exponential coloring

- Bonds everywhere if unshifted and then adopt switchback form

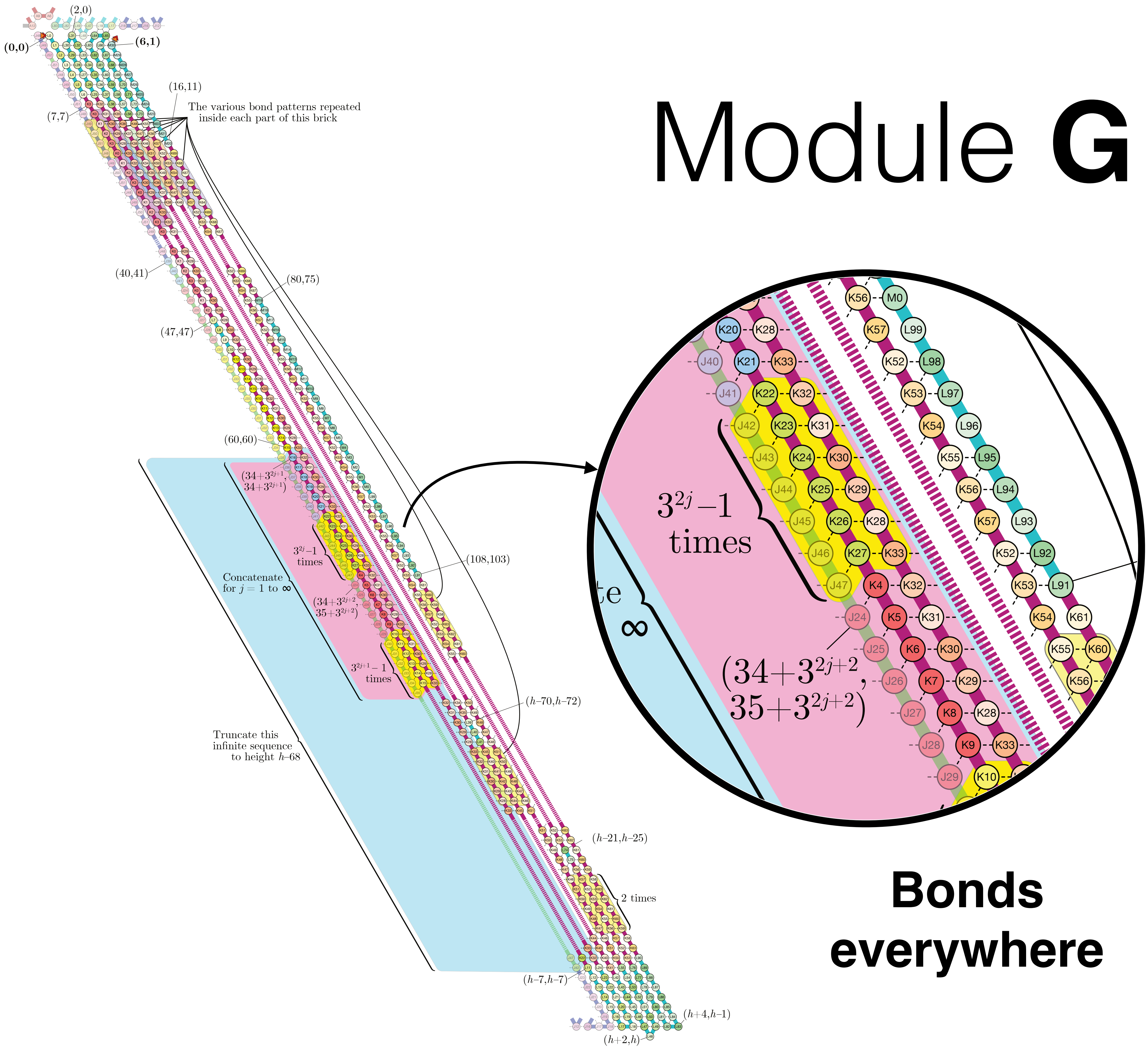
- Bonds nowhere if shifter and then adopt glider form

Module **G**

No Bond



Module **G**

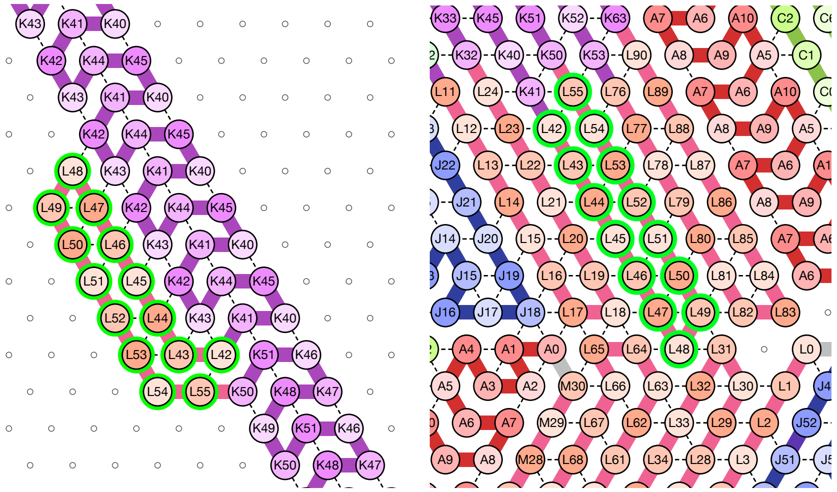

The various bond patterns repeated inside each part of this brick

Concatenate for $j = 1$ to ∞

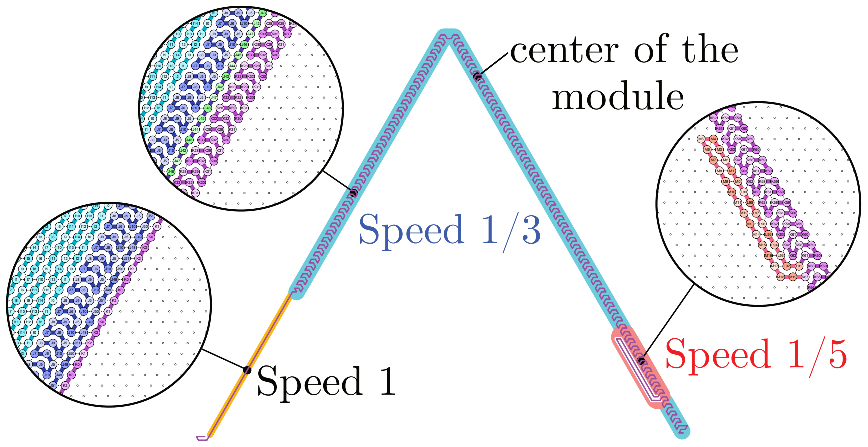
Truncate this infinite sequence to height h–68

Bonds everywhere

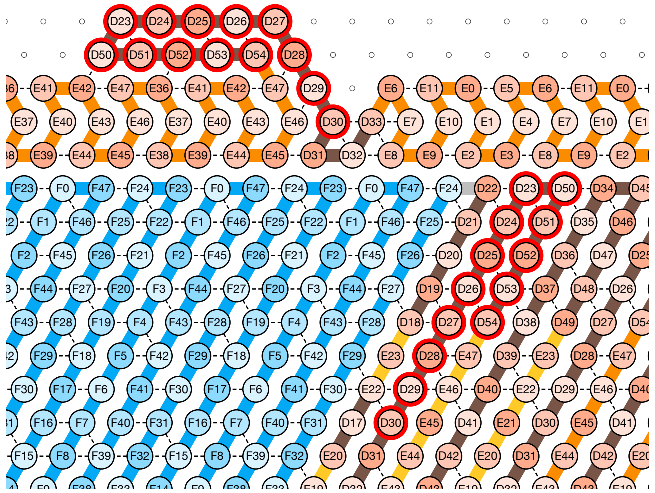
Socks



(a) Easier bond design



(b) Delaying gliders



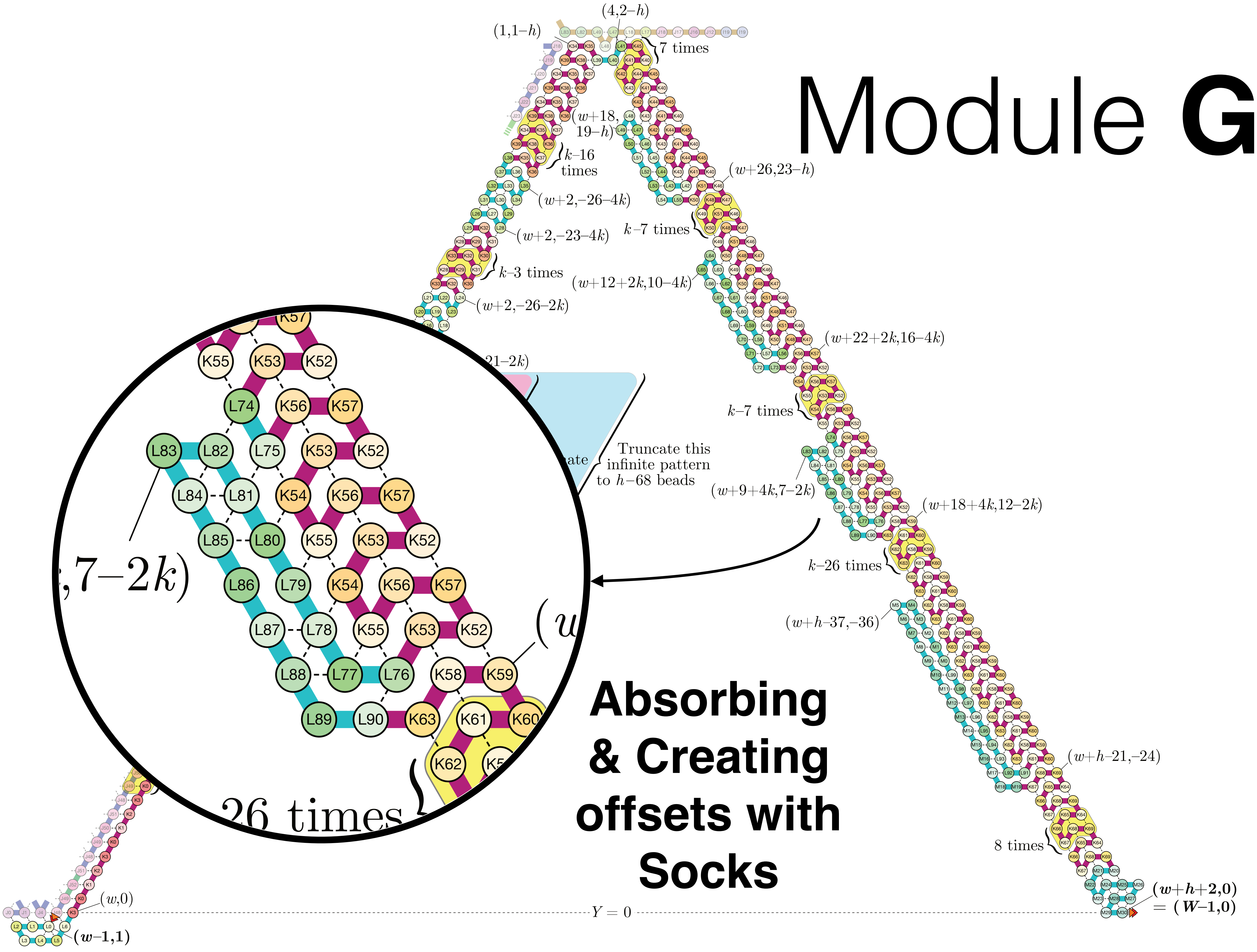
(c) Confinement

(a) Let fold parts in their natural forms, simplify the design

(b) Delaying and shifting to space out various functions

(c) Confinement to prevent unwanted interactions

Module **G**



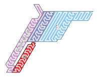
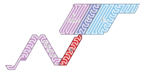
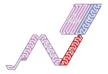
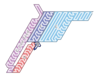
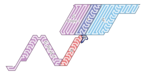
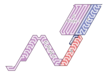
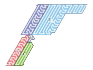
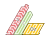
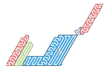
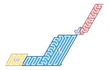
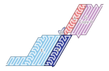
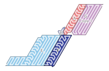
Absorbing & Creating offsets with Socks

Proof of correctness

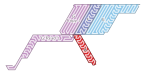
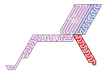
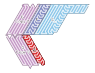
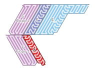
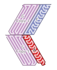
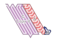
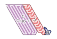
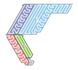
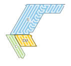
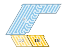
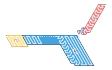
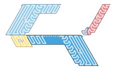
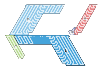
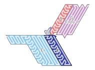
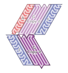
- Enumerate all possible environments for each brick
- Compute proof trees for each brick in all of its fixed environments
- Deal separately with the only three bricks having variable environments

Listing all environments

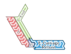
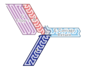
ZIG-UP

A	 #0-98	 #1286-1312	 #-	 #4995-4997
B	 #99-103	 #1313-	 #-	 #4998-5000
C	 #104-159 #1314-1315			
D_x	 #160-339	 #340-384		
E	 #1316-1392	 #385-749		
F	 #750-856	 #1393-1401		
G	 #857-1285	 #1402-1853		

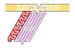
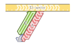
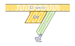
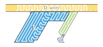
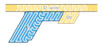
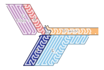
ZIG-DOWN

A	 #1854-1874	 #4745-4752	 #2382-2578	 #2745-2755	 #2790-2797
B	 #1875-1878	 #2579-2580	 #2756-		
C	 #1879-1889 #2757-2758	 #2798-2838 #4701-4702			
D_x	 #1890-1913 #2581-2599 #2600-2602	 #1914-1932	 same as previous ones		
E	 #1933-2011	 #2603-2632	 #2759-2789		
F	 #2012-2041	 #2633-2643			
G	 #2042-2381	 #2644-2744			

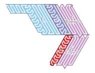
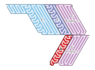
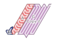
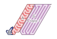
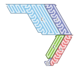
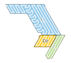
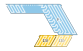
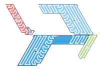
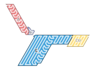
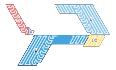
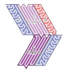
WRITE

D_x	 #2839-2999	 #4753-4786
E	 #4703-4733	 #3000-3749 #4787-4945
F	 #3750-3781 #4946-4959	 #4734-4744

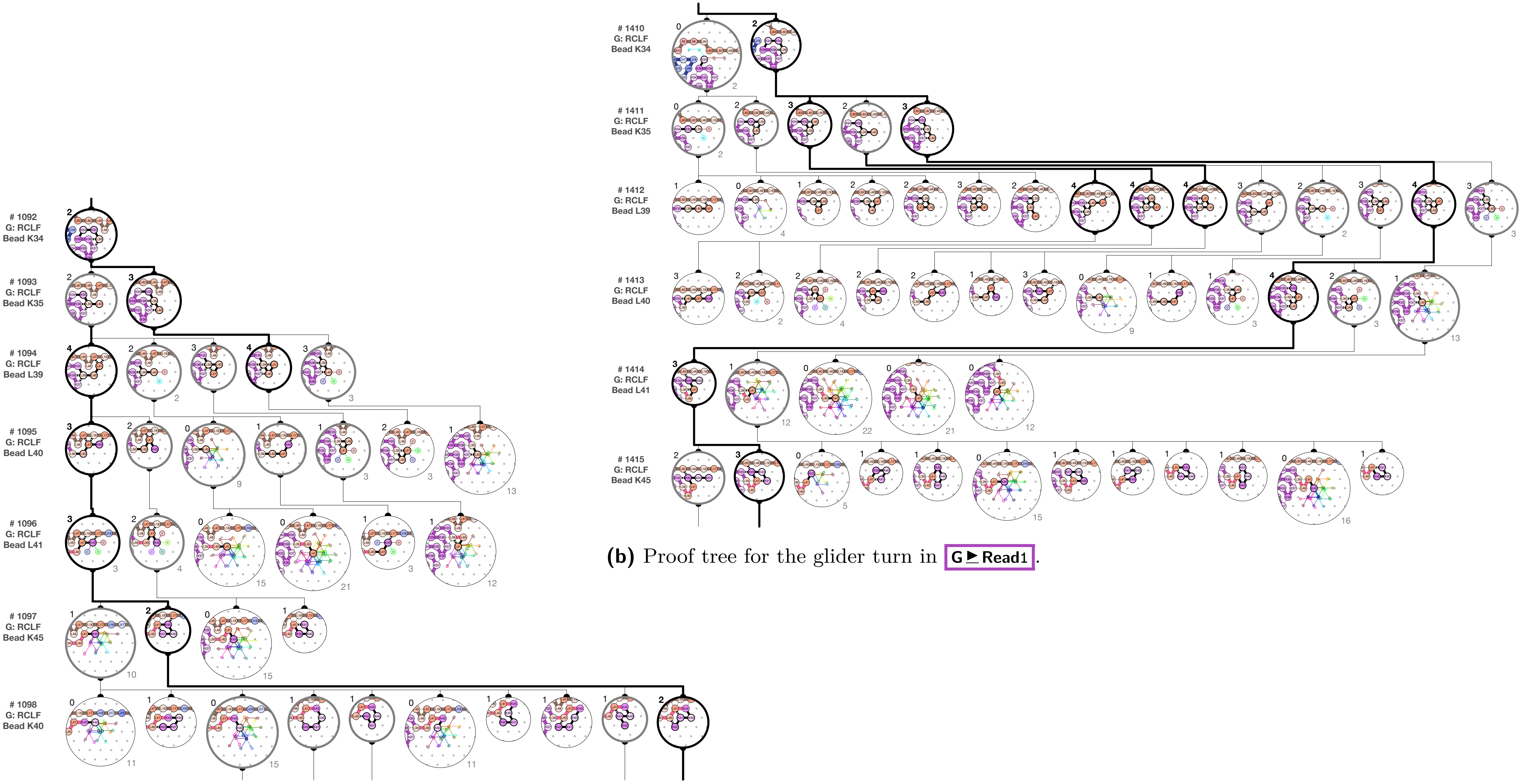
ZAG-WRITE

A	 #4994-	 #3806-3816 , #4059-4069 , #4215-4225 #4310-4320	
C	 #3817-3827 , #4070-4080 , #4226-4236 #4321-4331 , #4459-4460 , #4475-4476 #4550-4551 , #4605-4606		
D_x	 #3828-3851 #4237-4263 #4332-4355	 #3939-3941 , #4434-4439 , #4525-4527 #4571-4573 , #4587-4589 , #4595-4597 #4618-4620	
E	 #4081-4176 , #4461-4474 #4552-4570 , #4607-4617	 #3852-3924 , #3942-4020 , #4264-4271 #4356-4428 , #4440-4458 , #4504-4519 #4528-4549 , #4574-4581 , #4590-4594 #4598-4604 , #4621-4644	
F	 #3925-3938 , #4021-4034 , #4177-4190 #4272-4285 , #4429-4433 , #4520-4524 #4582-4586	 #4645-4653	 #4960-4967
G	 #4968-4993	 #3782-3805 , #4035-4058 , #4191-4214 #4286-4309	

ZAG

A	 see Zig-Down	 see Zig-Down	
B	 see Zig-Down	 see Zig-Down	
C	 see Zig-Down		
D_x	 see Zig-Down	 see Zig-Down	 see Zig-Down
E	 see Zig-Down	 see Zig-Down	 see Zig-Down
F	 see Zig-Down	 see Zig-Down	 see Zig-Down
G	 #4654-4700	 see Zig-Down	 see Zig-Down

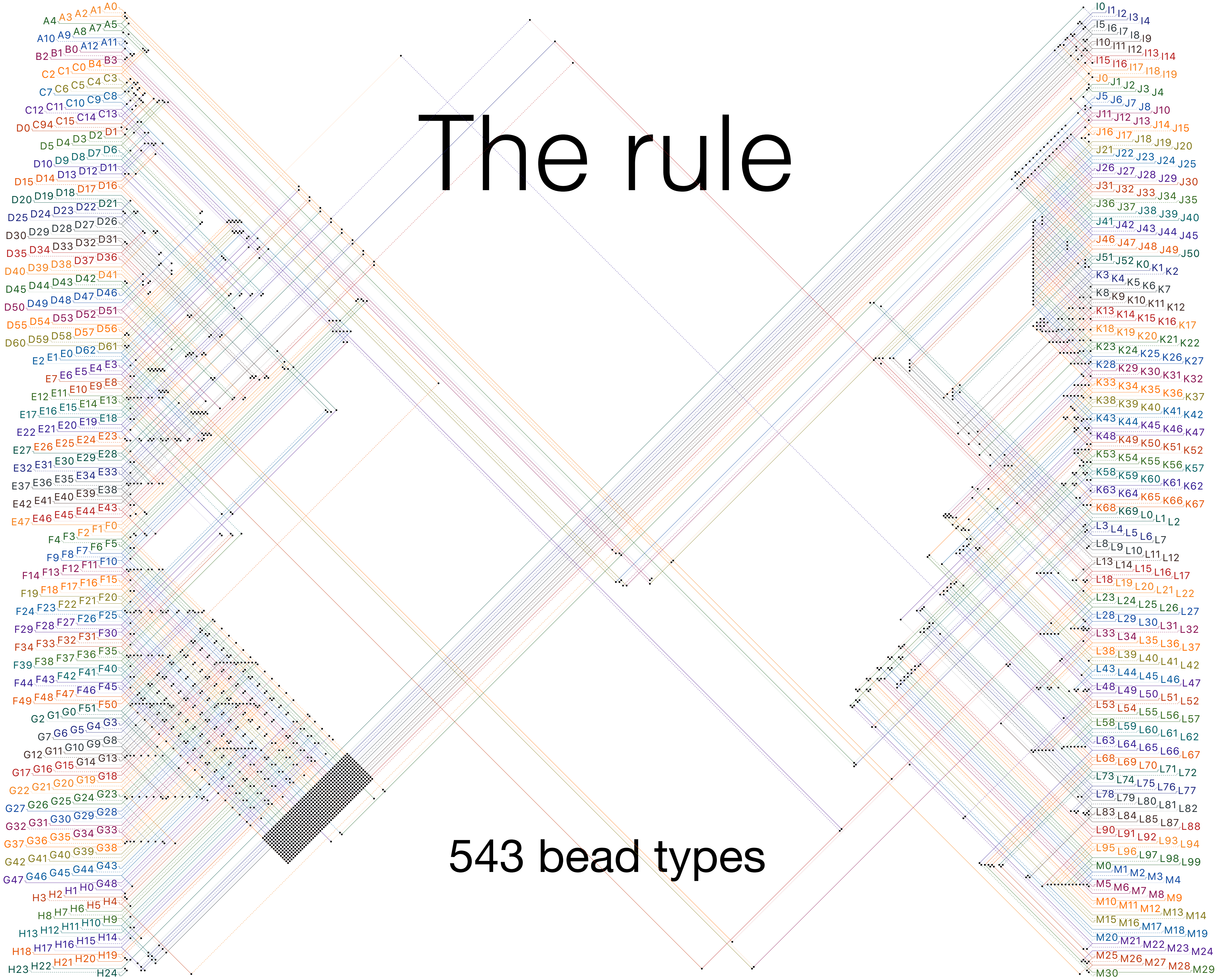
Example: Reading 0/1



(a) Proof tree for the glider turn in G ▶ Read0.

(b) Proof tree for the glider turn in G ▶ Read1.

The rule



543 bead types