

DEMONGEOT2017

# Folding Turing is hard but feasible

Nicolas **Schabanel**

*CNRS - U. Paris Diderot (IRIF) - U. Lyon (IXXI) - France*

Joint work with

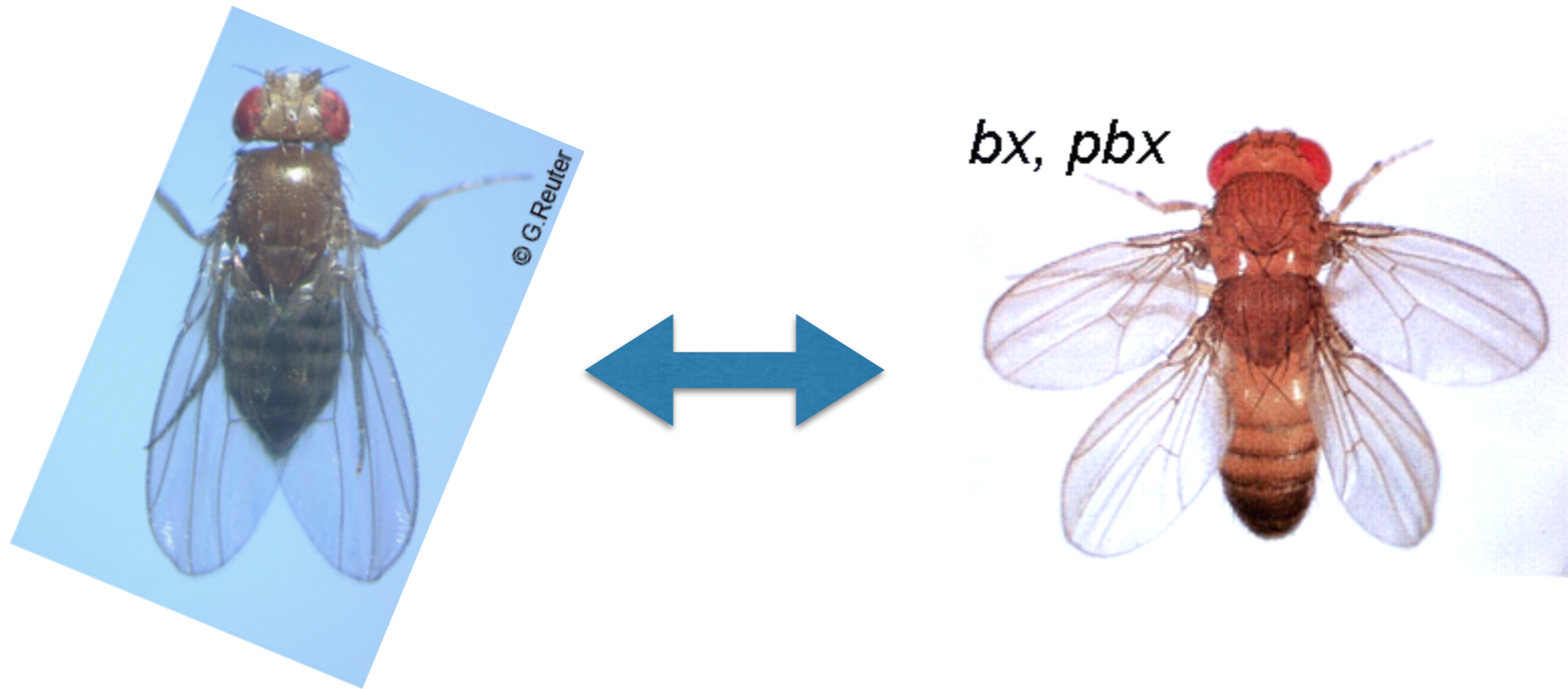
**Cody Geary** (CalTech, USA)

**Pierre-Étienne Meunier** (U. Aalto, Suomi - Finland)

**Shinnosuke Seki** (U. Electro-Communication, 日本 - Japan)

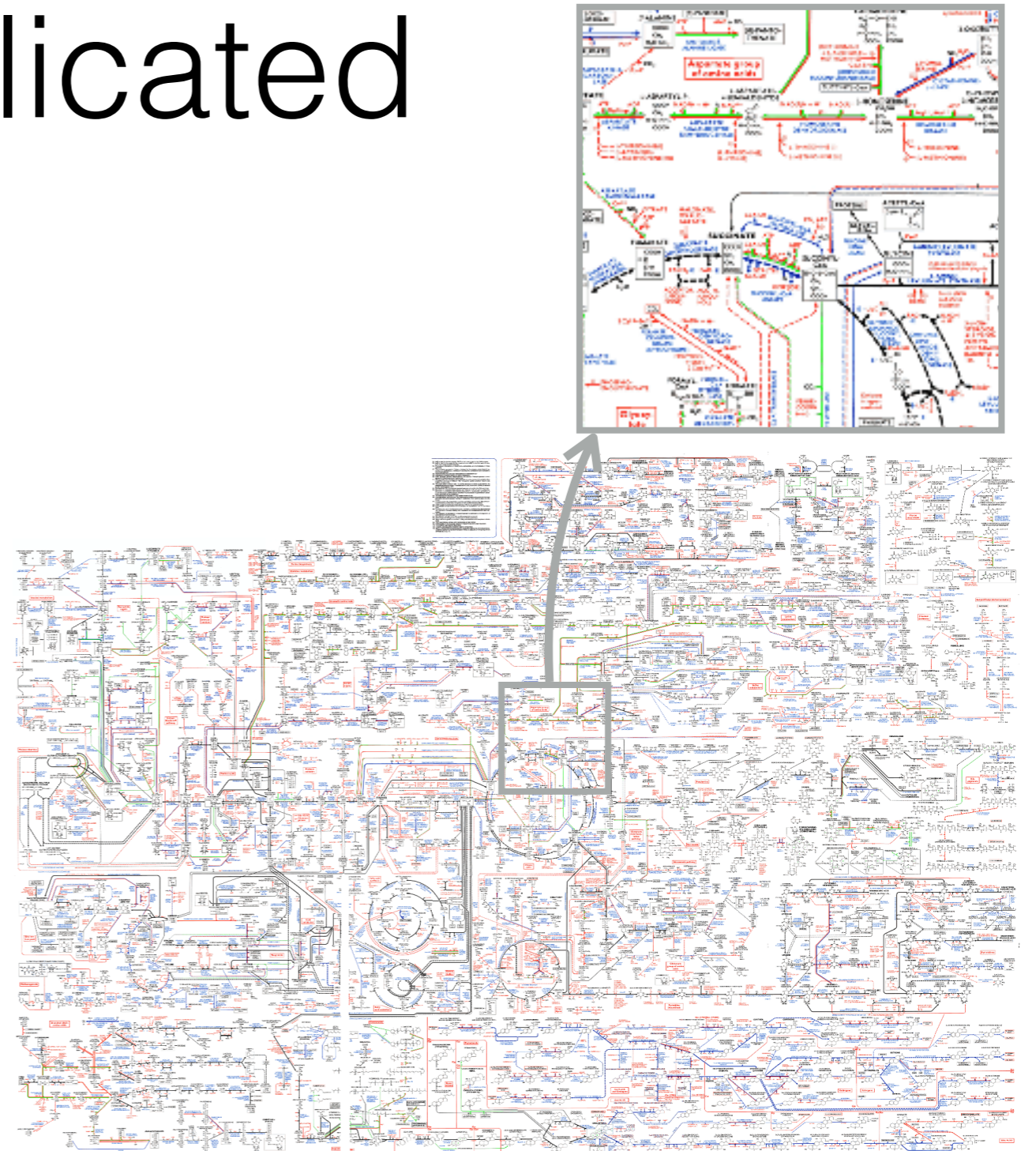


# Genetic code behaves as a program



For instance, small changes in code yields big differences

# Nature is very complicated



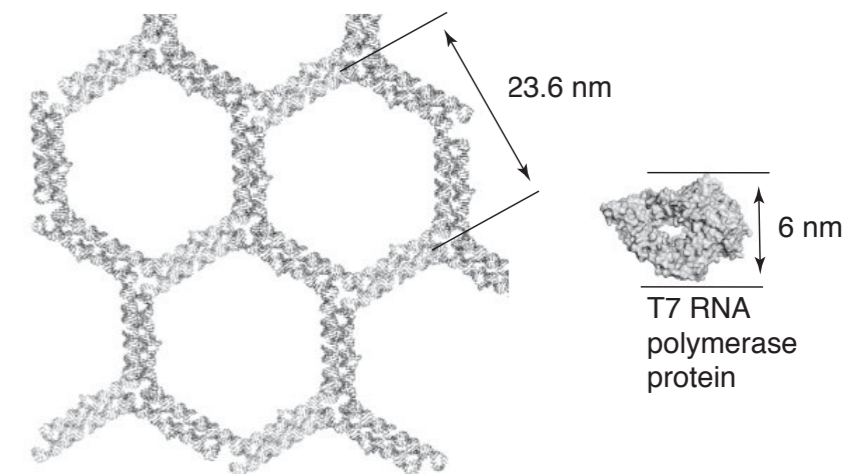
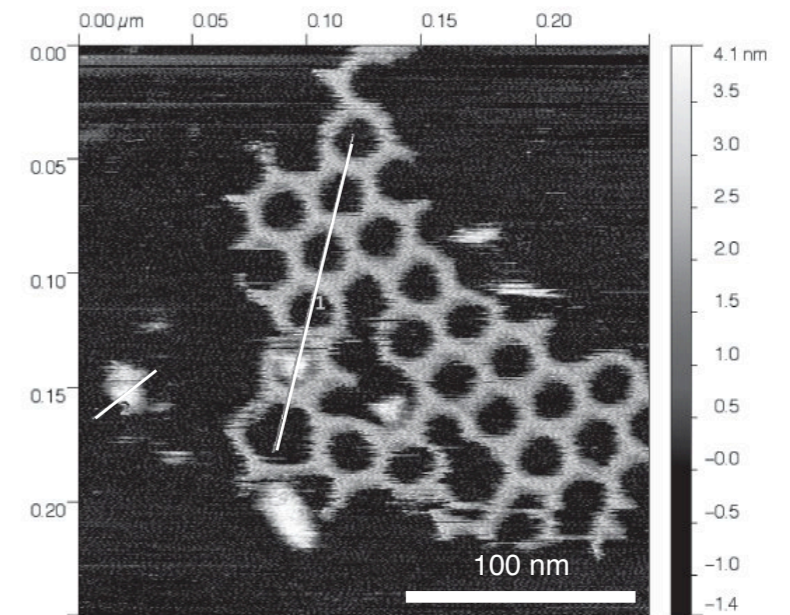
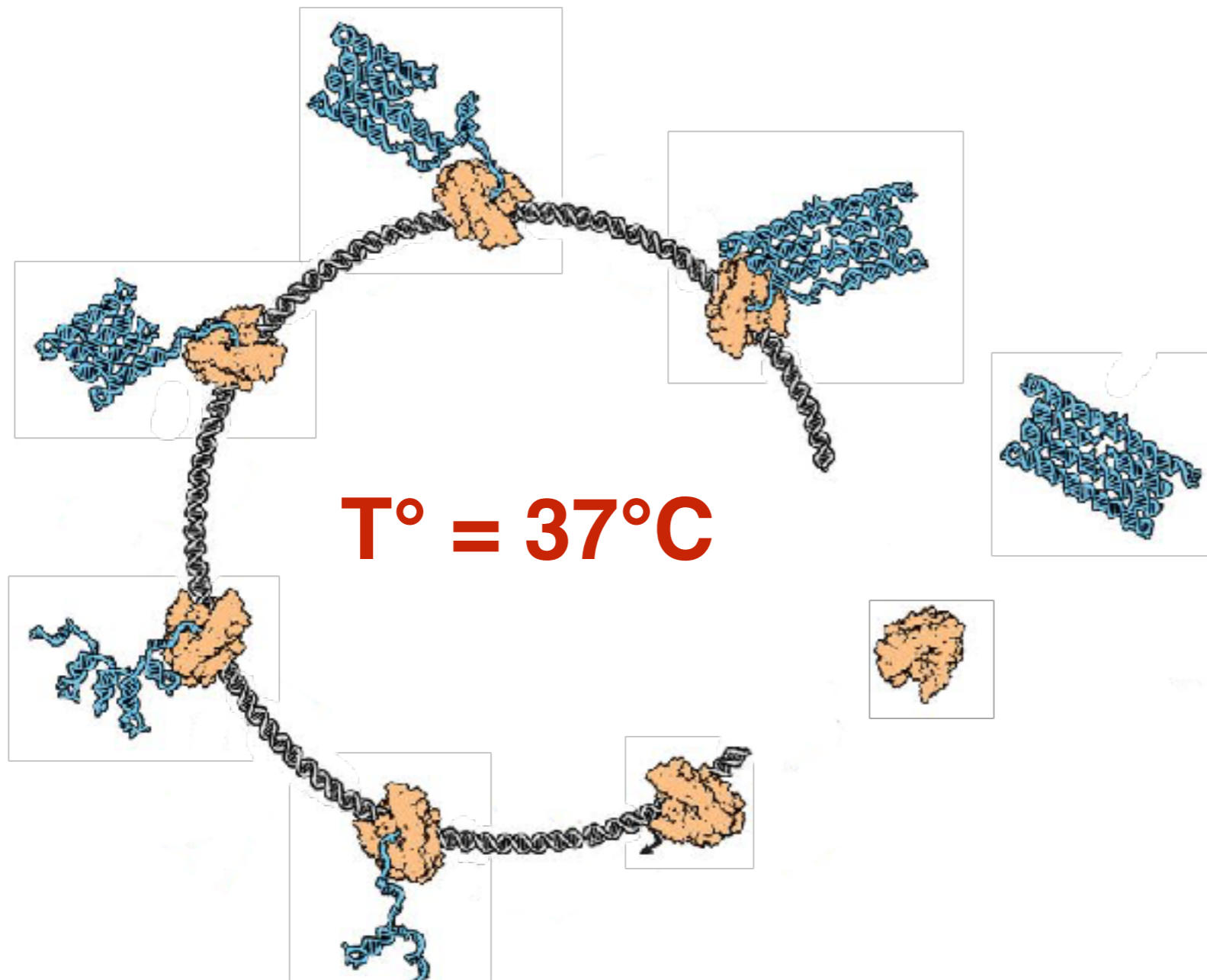
However, we might do  
better differently





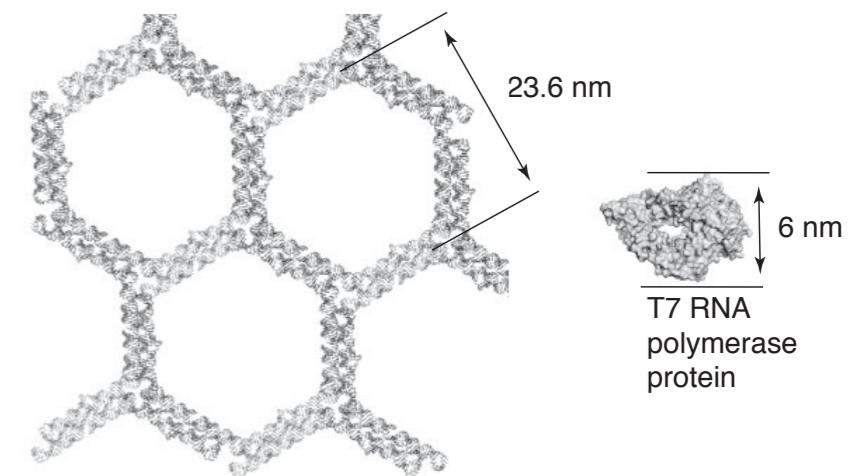
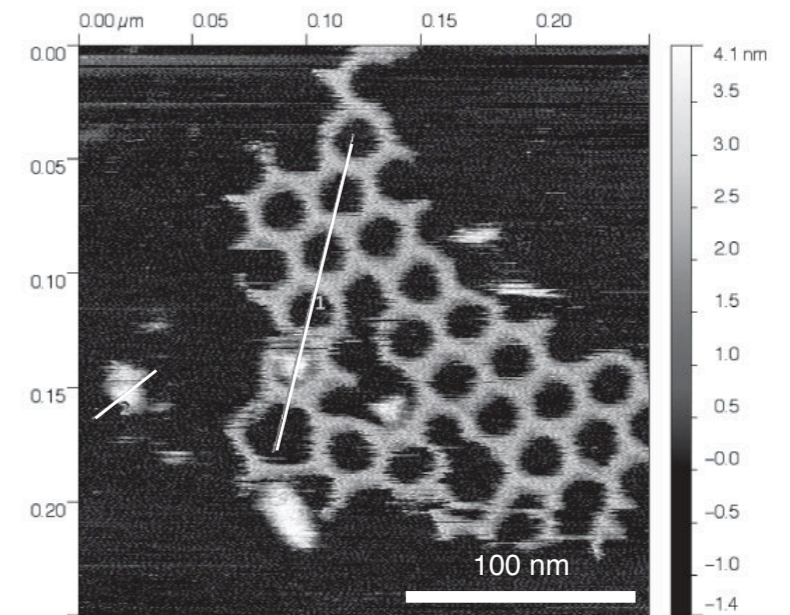
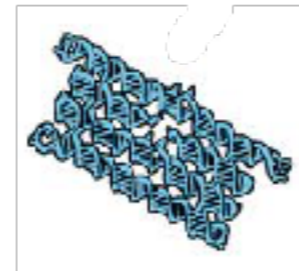
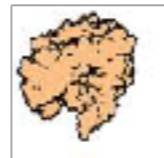
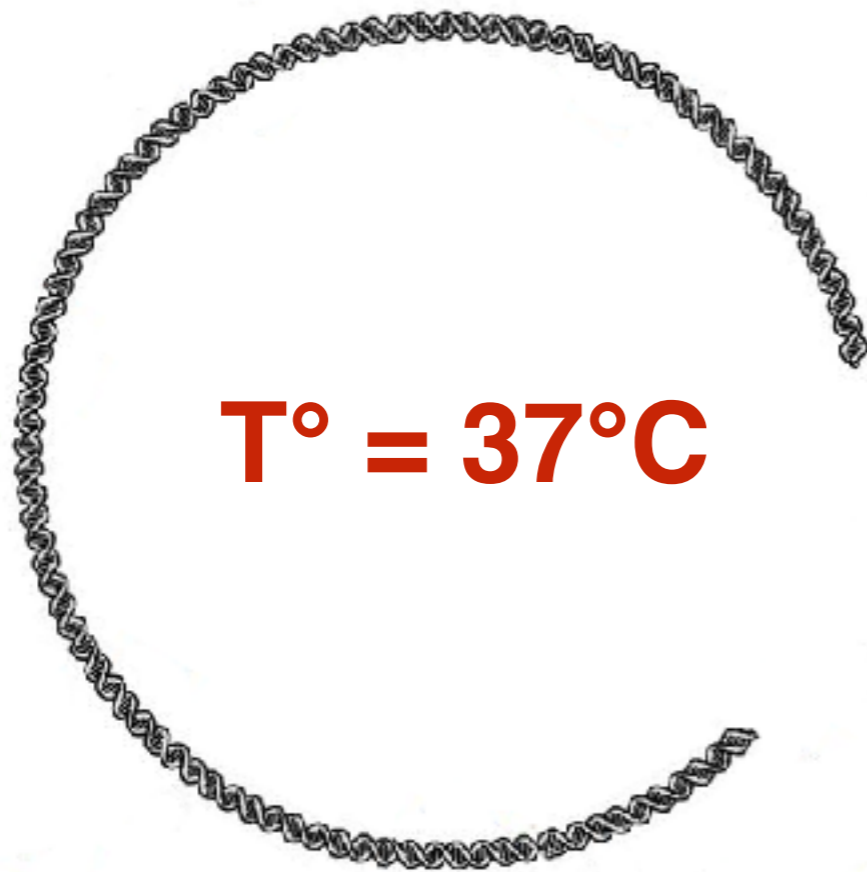
# Co-transcriptional folding

# RNA co-transcriptional folding

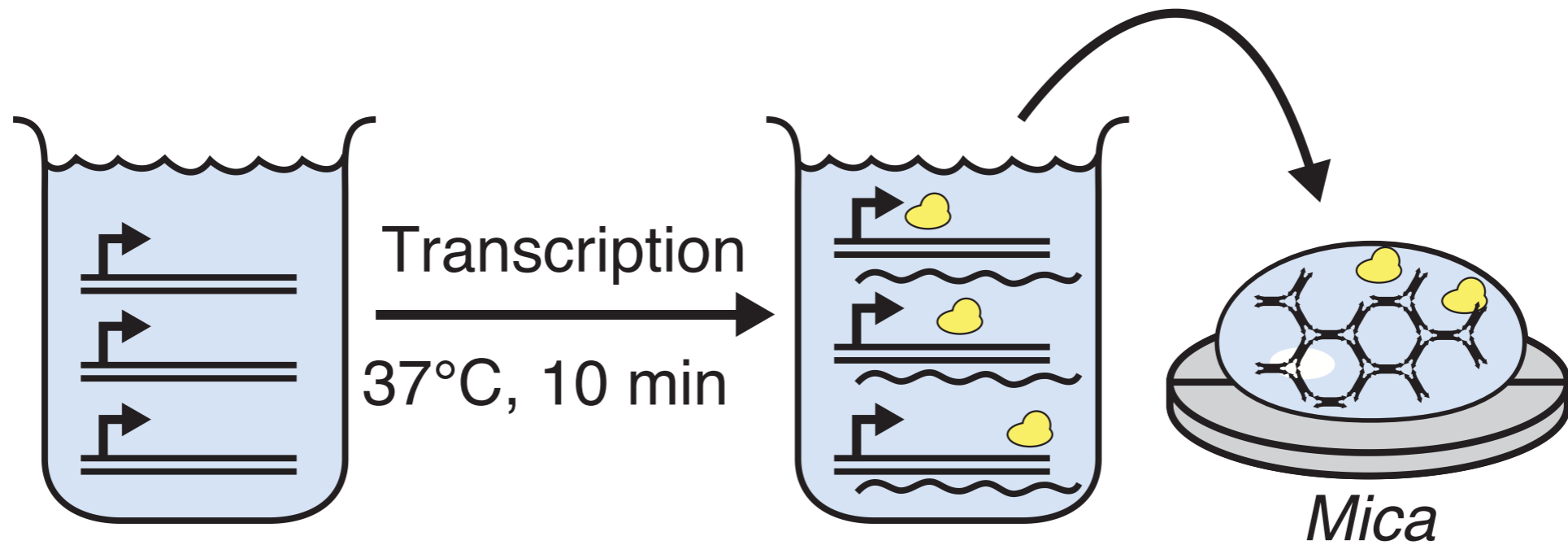




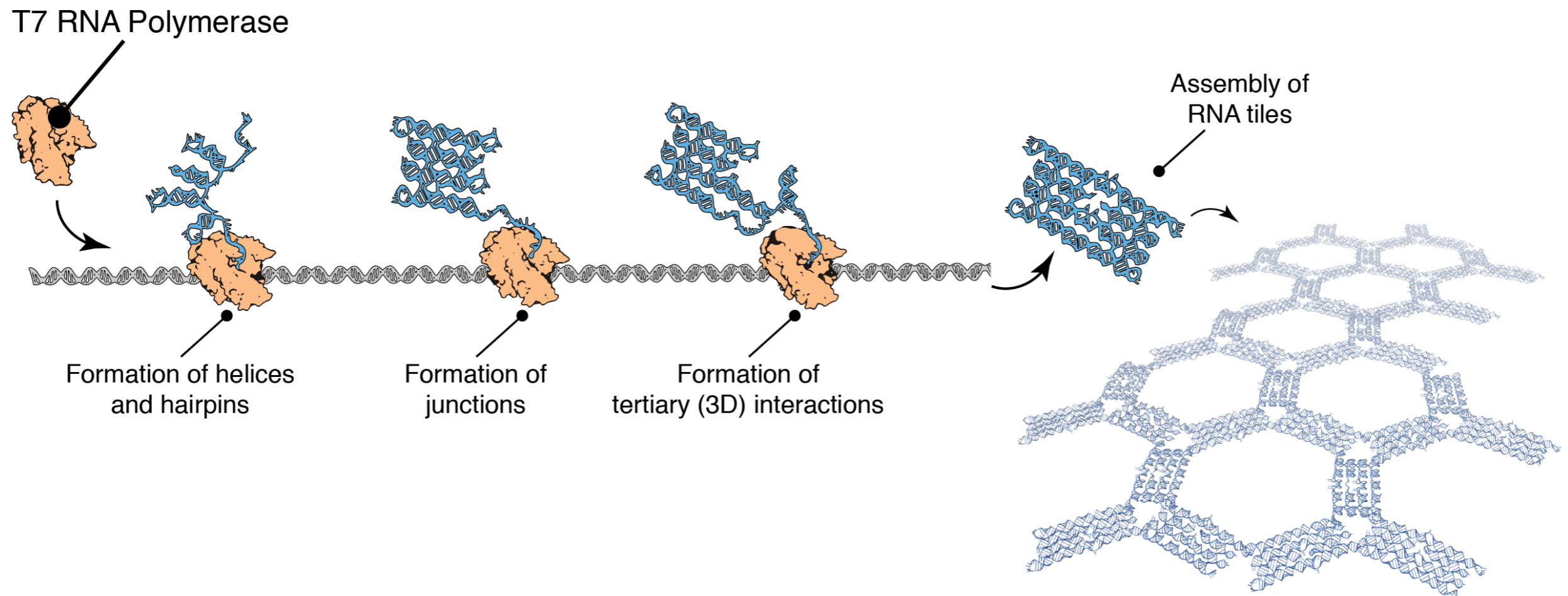
# RNA co-transcriptional folding



# Protocol



# RNA Origami in Real Time



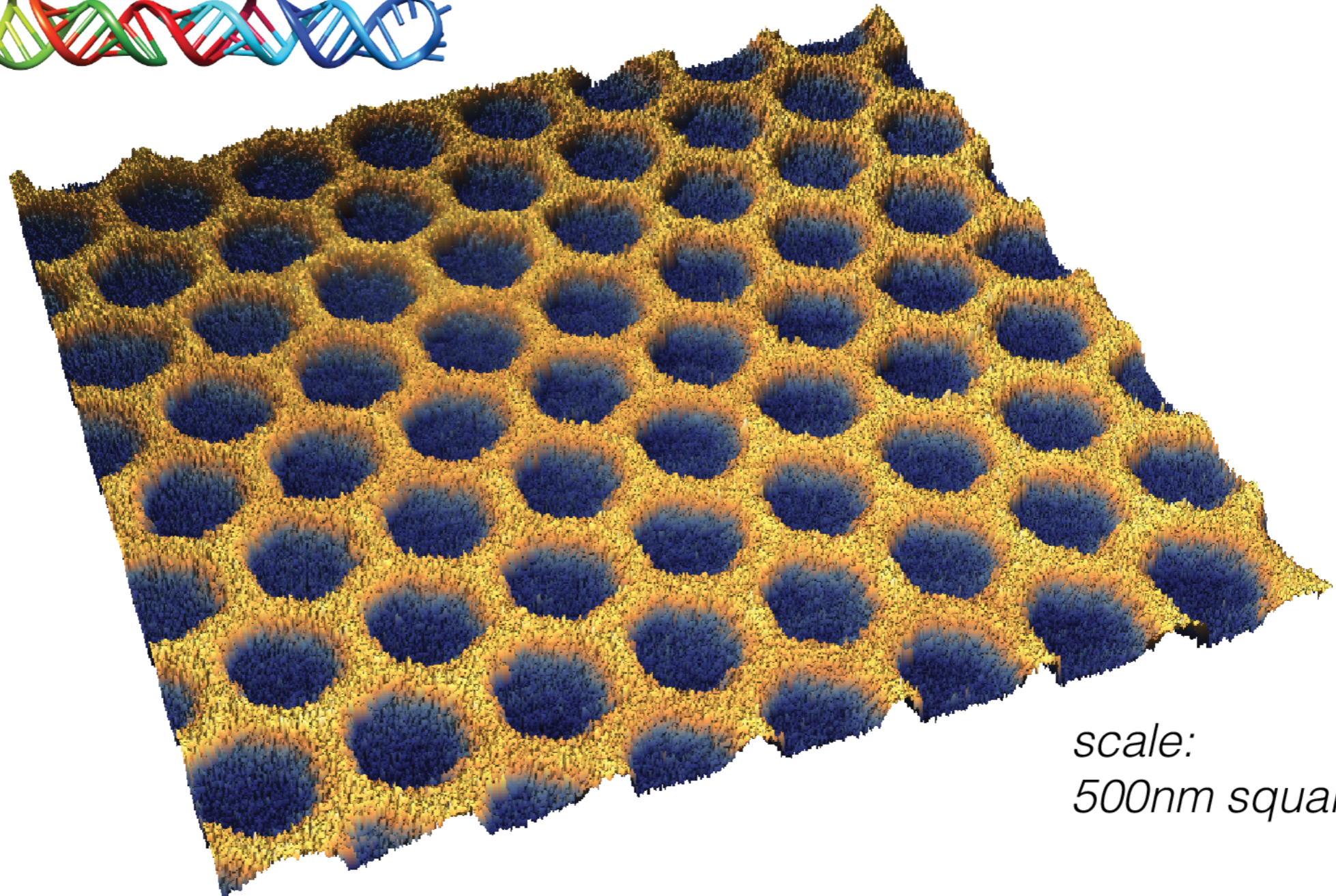
T7 RNA polymerase produces RNA directionally from 5' to 3', **at a rate much slower than the RNA folds up (few microseconds).**

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

# AFM Imaging



2H-AE (176 nucleotides)

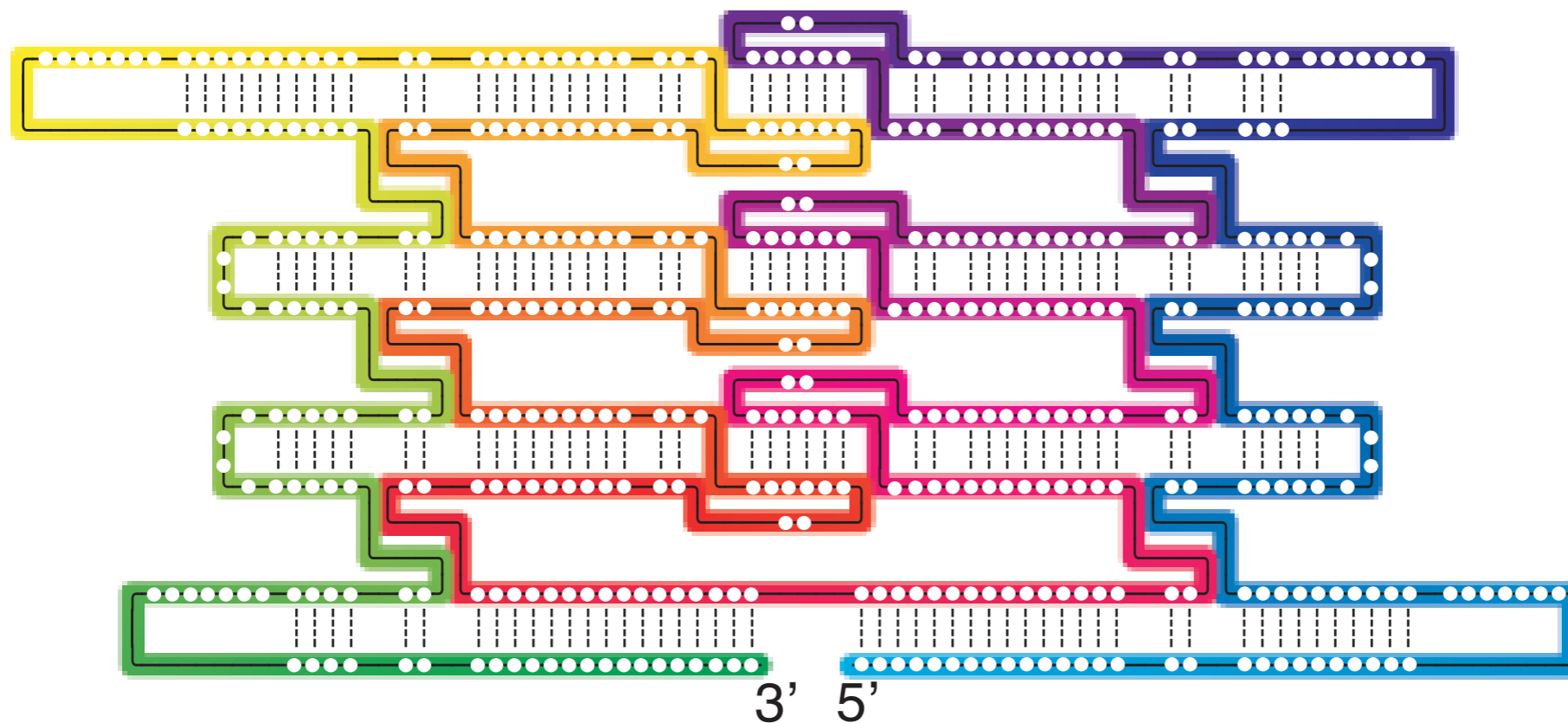


*scale:  
500nm square*



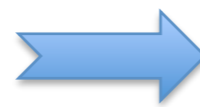
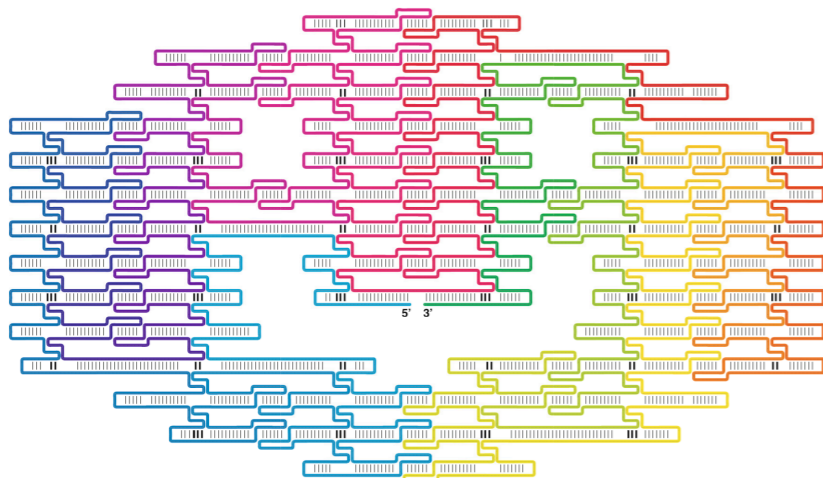
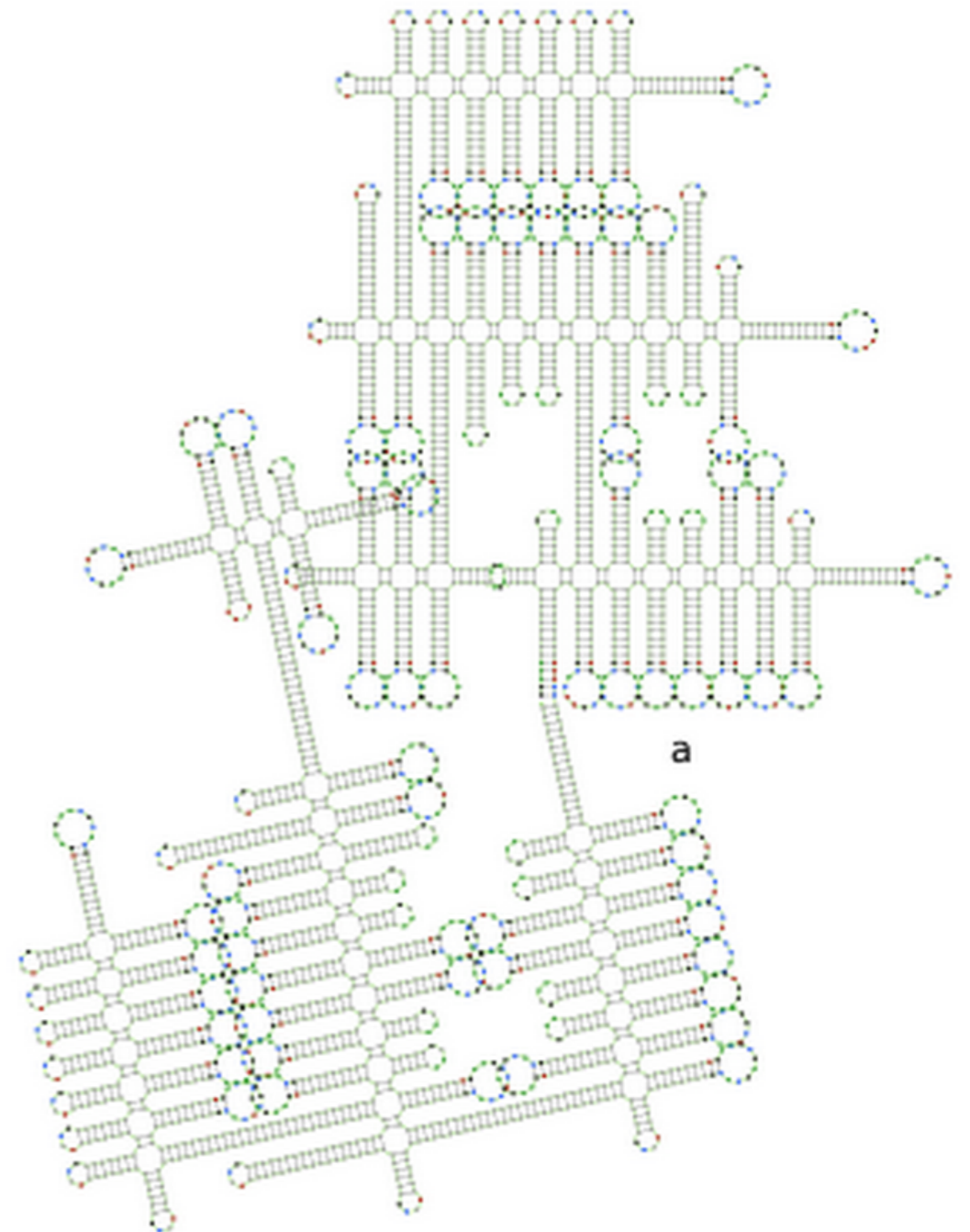
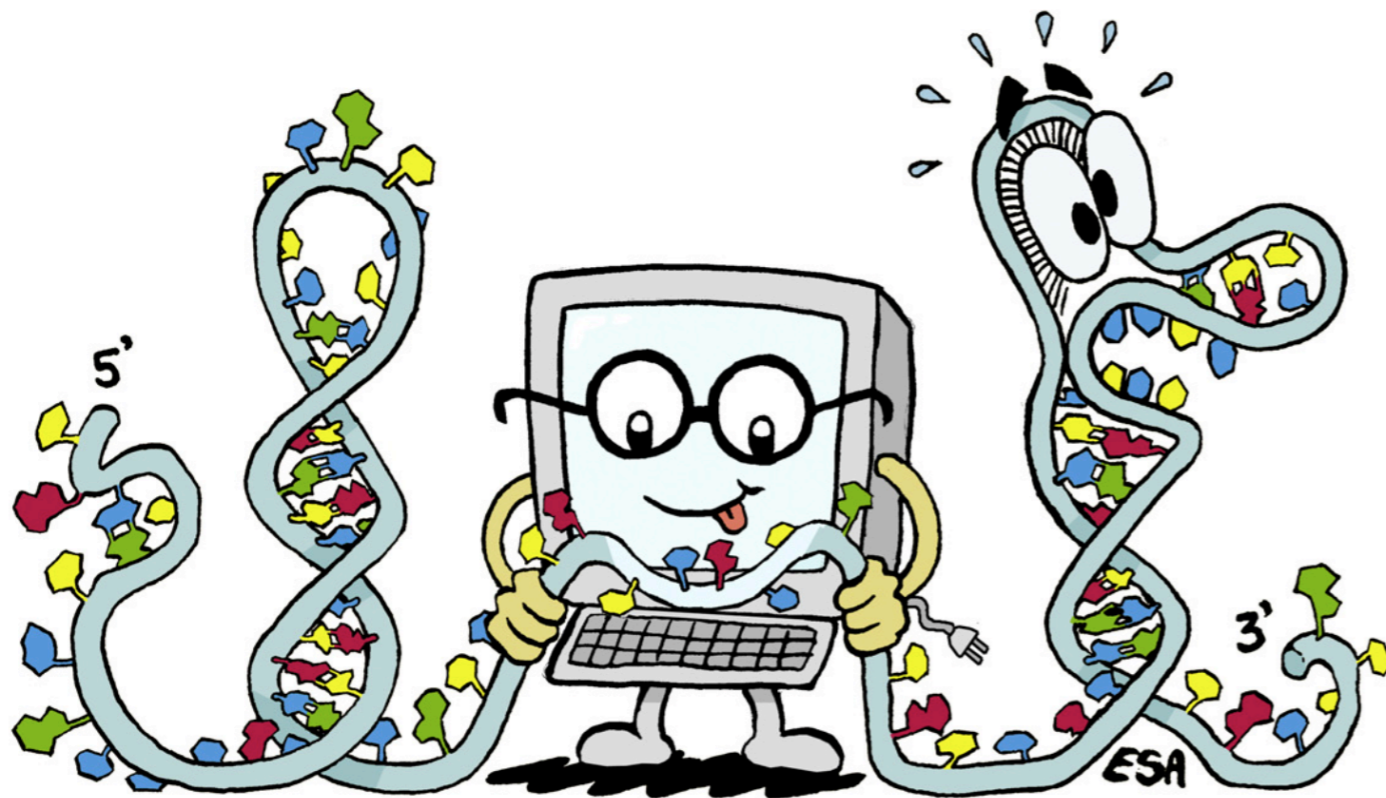
# Strand-path diagram

4H-AE



Shows the order of synthesis from blue (5') to green (3')

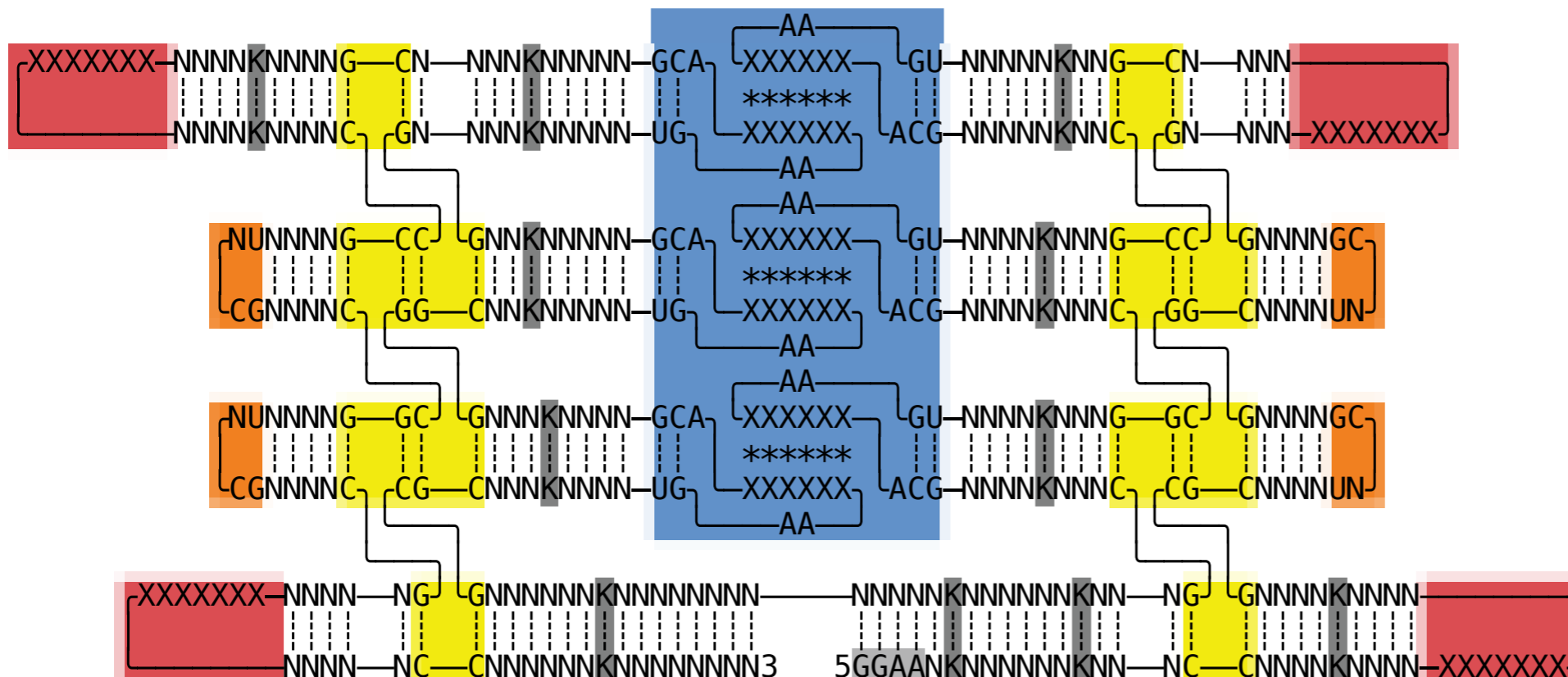
# With some computer & data mining help...








**NUPACK**

# The modular design

4H-AE



RNA Modules	
	180° KL
	120° KL
	Tetraloop
	Dovetail Seam
	Helix
●	5' End

Secondary structure diagrams shows the base pair and sequence constraints used in NUPACK design

Colored areas indicate conserved tertiary motifs

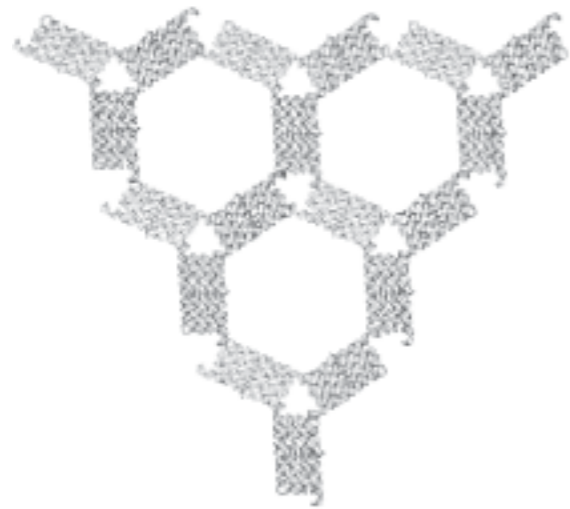




# The final sequence

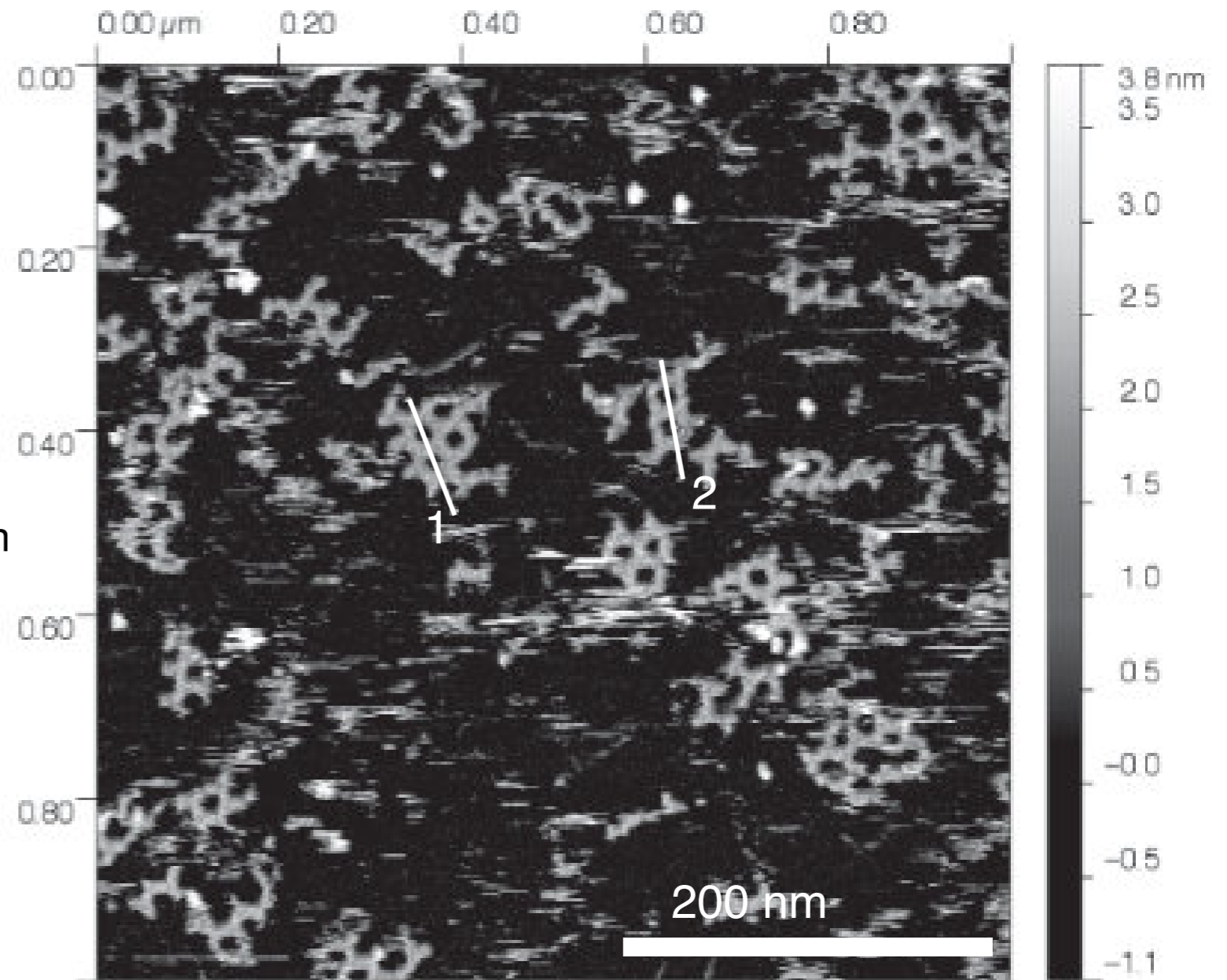
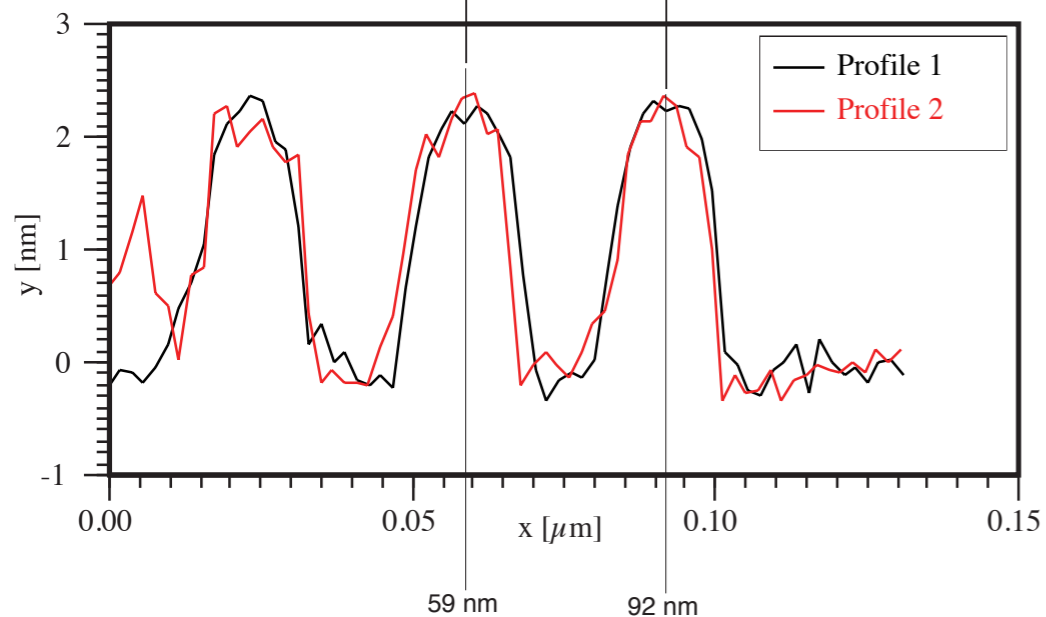
Sequence Name	RNA Sequence
<b>4H-AE-B</b> <b>418</b> <b>nucleotides</b>	GGAAGUGAGUAGUAGUCCACCGUCUACAACCAUCGUAGGCGGUGCGCUCACUUCGGUG AGGGCCAGCUACGGCUGGGACCCGAGACGUGGGUCGACGGAGGCUGAAGCGAGCACGG CCUCUGUCCCGAGCUCUGCUGAAAGGCUCACGGCAGGGCUCCGGCAGGUGGCUGAAGC CUCCACGGCCAUCUGCGACUGCUACUCGCUUCCGCGAAAUGUCAAUACGCGCAGCGAC GGUGAAGGAGGCACGCCGUUGCUGGGCACUGAGGGUGAAGAGCCUACGCCUCGGUGG CGAAGCGACCUGAAGCUCGCACGGGUCGUUUCGCGCAGGGUCAGAGAACGCCUGAUCC UGCCCGAUCCUACGGGAUCCGGAACAUAACGUGUUCGGACCGACCACGACGGUCCCGUA UUGGCAUUUCGC

# AFM imaging of 4H-AE co-transcriptional assembly



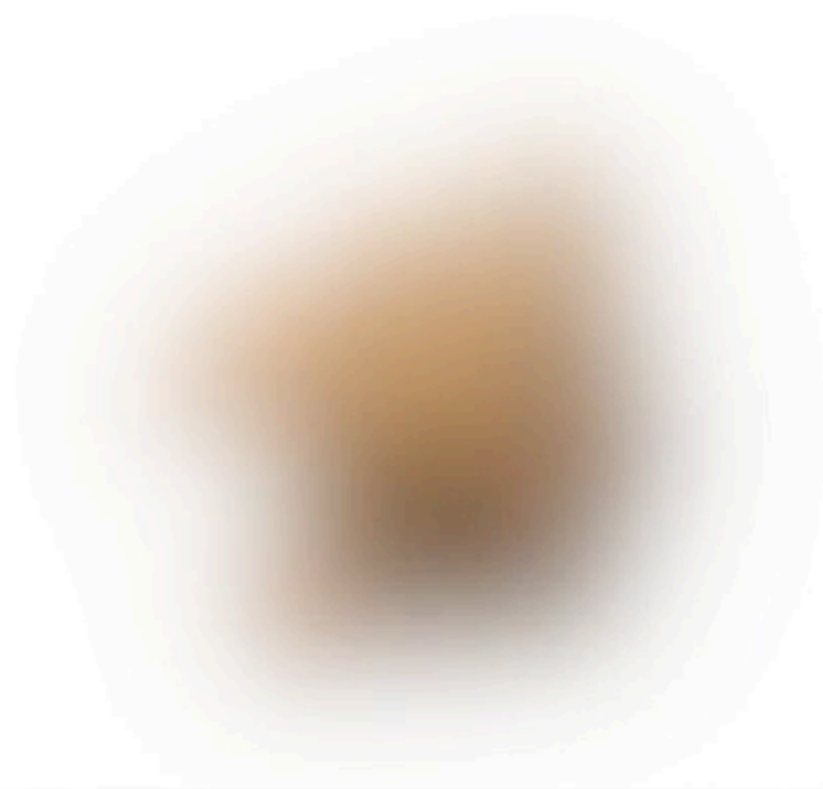
period = 33.0 nm

Note that the modeled spacing was 33.5nm



# RNA Folding

(Real time: ~1 second)





# Oritatami

A model for co-transcriptional folding

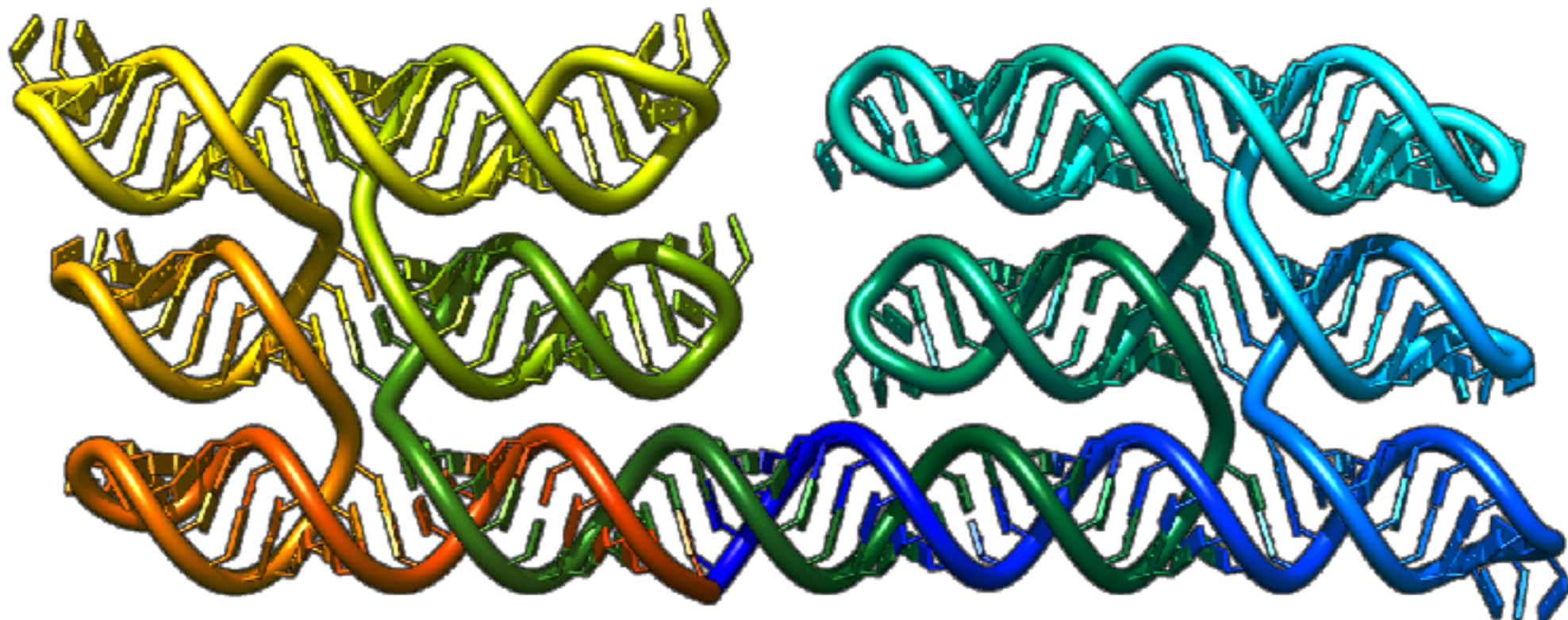
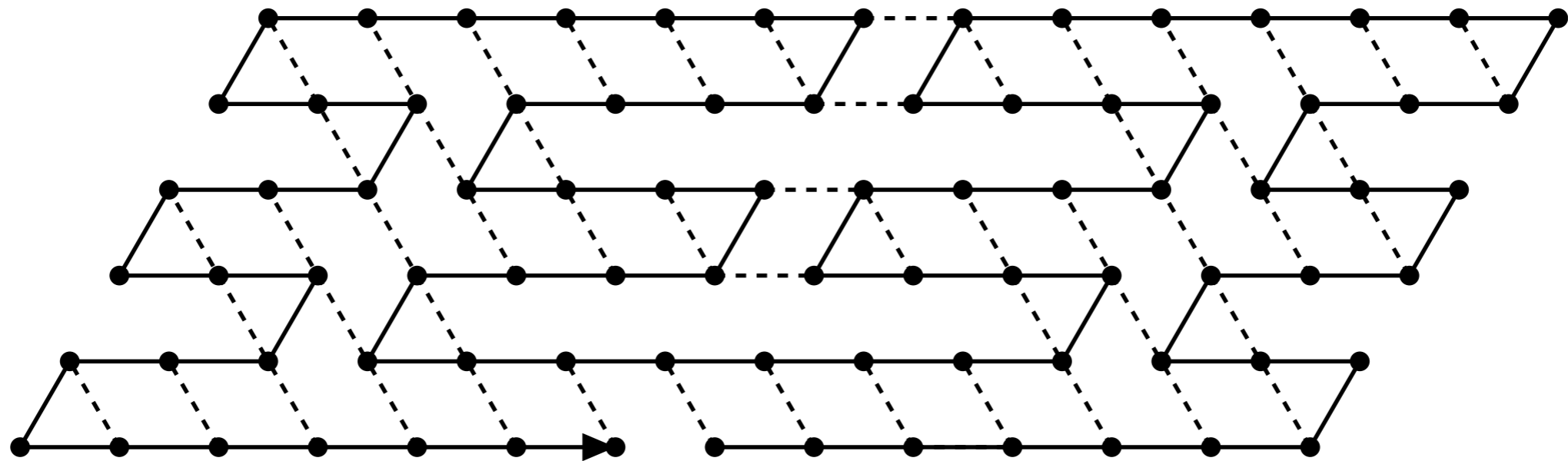
- **The program:** a **periodic sequence** of **beads** (the **primary structure**) onto the triangular grid
- **The instructions:** the **rule** **a♥b** if beads **a** and **b** attract each other
- **The input:** the **seed**, some beads placed beforehand and a starting position

# Oritatami

A model for co-transcriptional folding

- **The dynamics.**
  - The sequence is *produced bead by bead*
  - **Only the  $\delta$  last beads** explore the accessible positions and settle in the ones **maximizing the number of bonds**
  - All other beads remain in their last location

# Example of such molecule



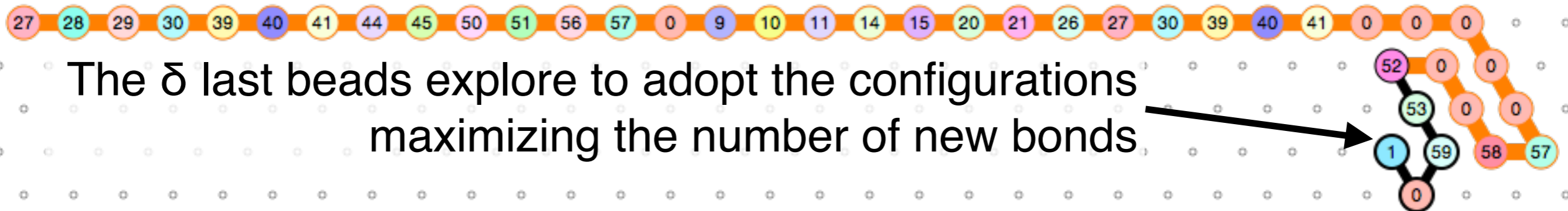


# Oritatami

A model for co-transcriptional folding

A growing molecule to fold given as a **periodic sequence** of **beads** which **attract** each other according to the **rule** ❤️

Starts from a seed

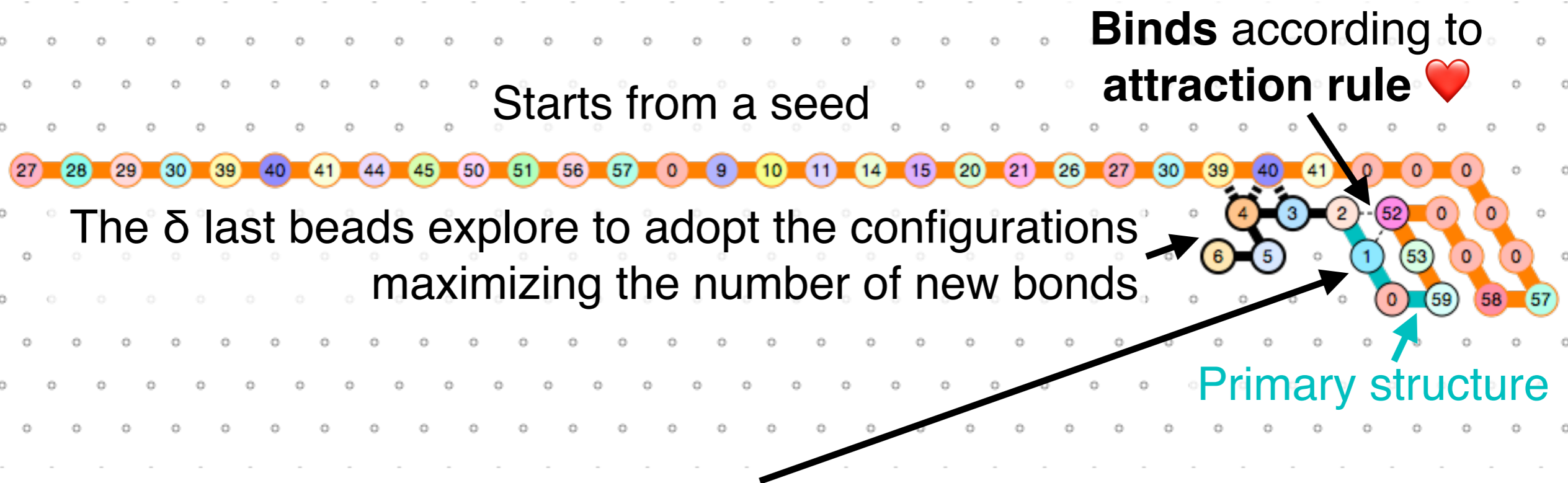


The  $\delta$  last beads explore to adopt the configurations maximizing the number of new bonds

# Oritatami

A model for co-transcriptional folding

A growing molecule to fold given as a **periodic sequence** of **beads** which **attract** each other according to the **rule** ❤️

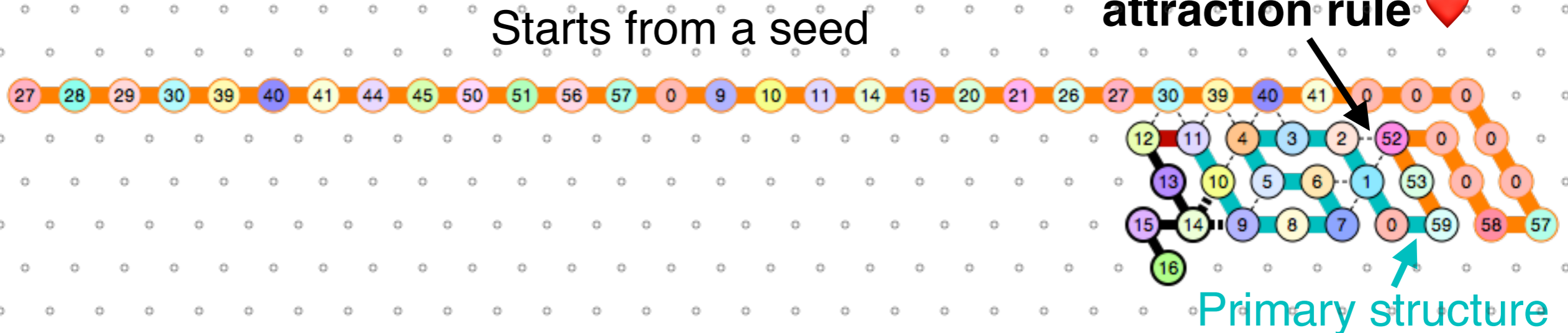


The beads older than  $\delta$  will keep their positions

# Oritatami

A model for co-transcriptional folding

A growing molecule to fold given as a **periodic sequence** of **beads** which **attract** each other according to the **rule** ❤️



The sequence keeps folding upon itself as it grows

# Oritatami

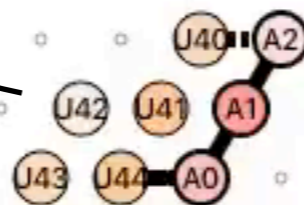
A model for co-transcriptional folding

A growing molecule to fold given as a **periodic sequence**

A0-A1-A2-A3-A4-A5-A6-A7-A8-A9-A10...

The Seed

The growing molecule  
( $\delta=3$ )



# How does computation work?

- **Environment** is the **memory**
- **Entry point** in an area is the **current machine state**
- Depending on the entry point, **different parts** of the molecule will «**read**» **the input** encoded in the environment **by binding to it**



# How does computation work?

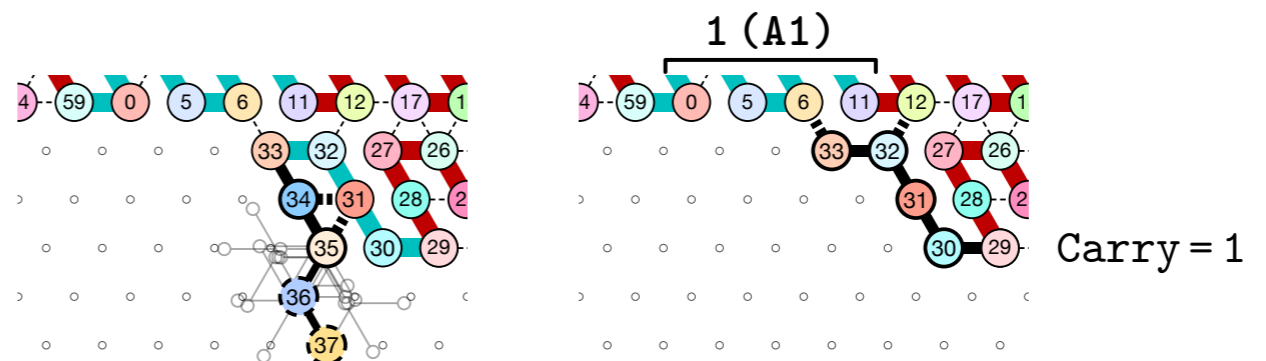
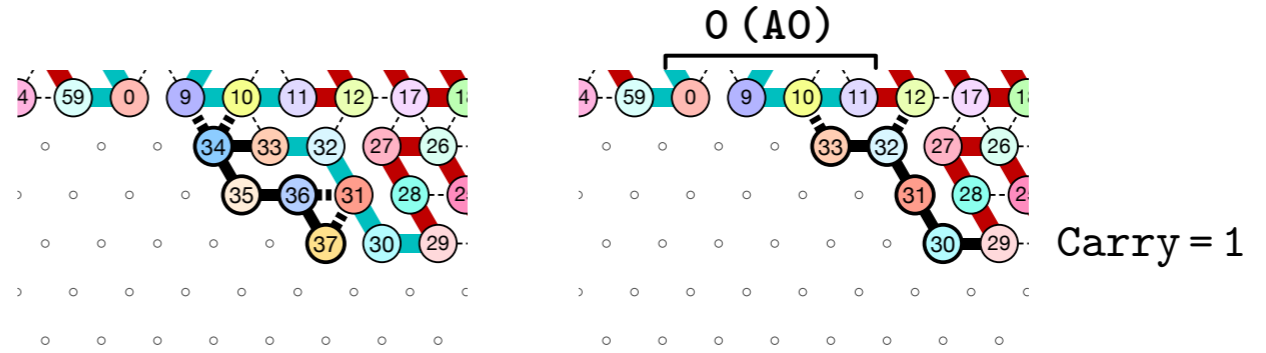
3) Beads 38-41



2) Beads 34-37



1) Beads 30-33



# How does computation work?

3) Beads 38-41

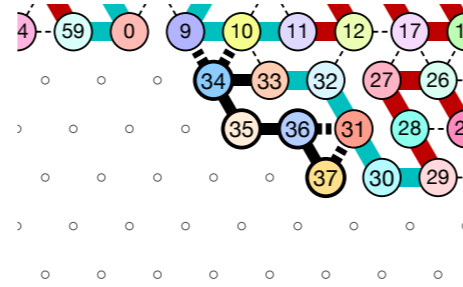
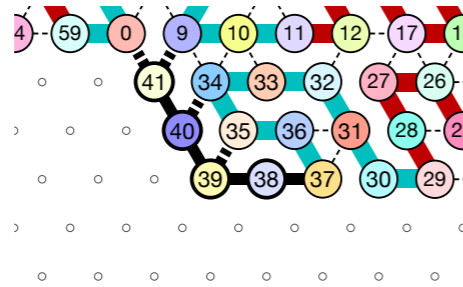


2) Beads 34-37

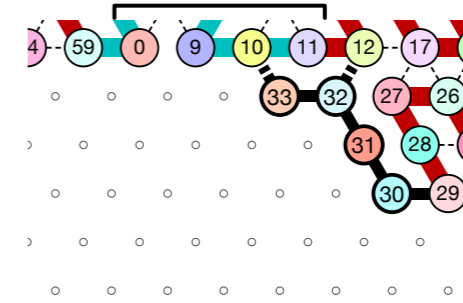


1) Beads 30-33

Carry = 0

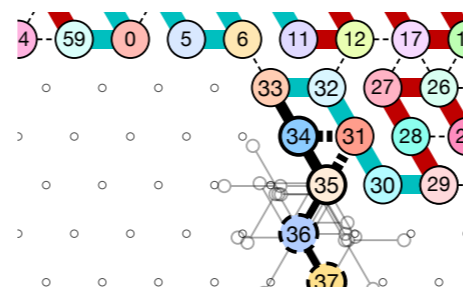
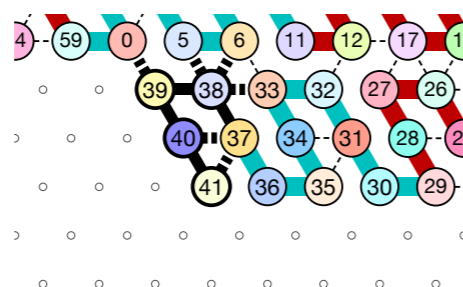


0 (A0)

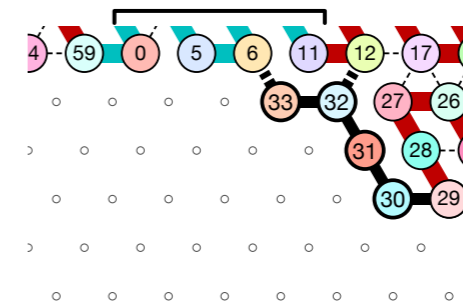


Carry = 1

Carry = 1



1 (A1)



Carry = 1



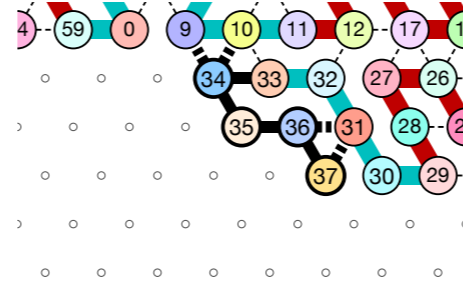
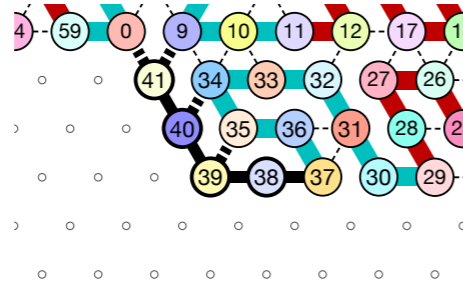
# How does computation work?

3) Beads 38-41 ←

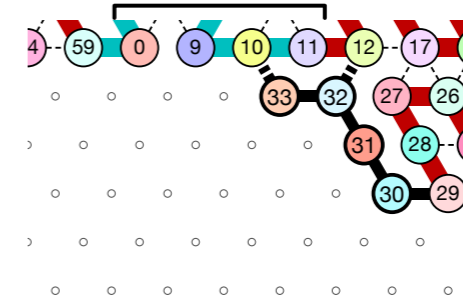
2) Beads 34-37 ←

1) Beads 30-33

Carry = 0

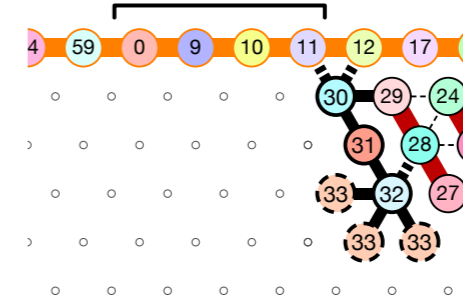


0 (A0)



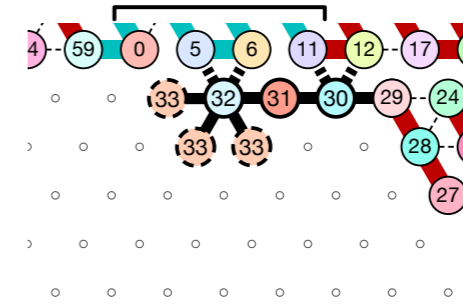
Carry = 1

0 (A0)



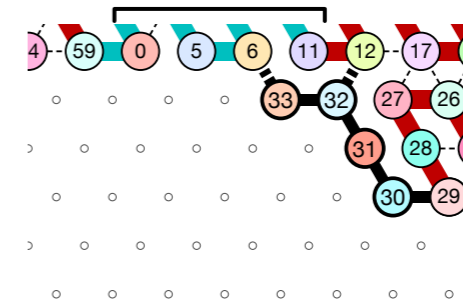
Carry = 0

1 (A1)



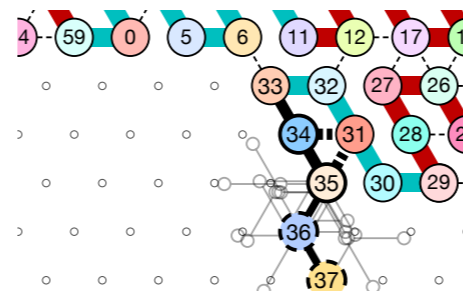
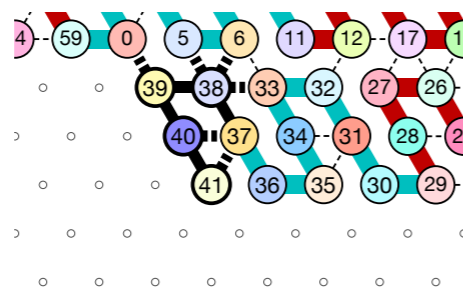
Carry = 0

1 (A1)



Carry = 1

Carry = 1



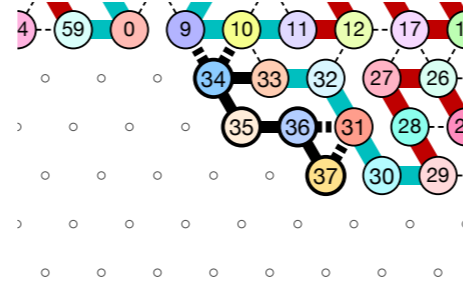
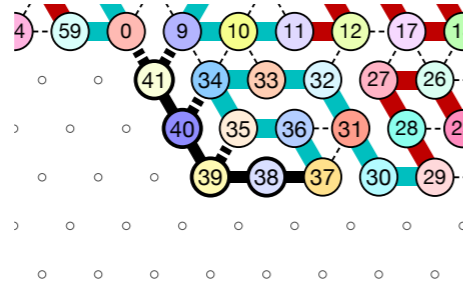
# How does computation work?

3) Beads 38-41 ←

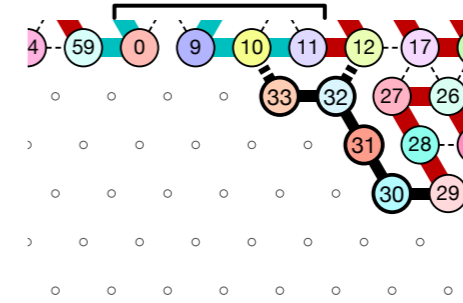
2) Beads 34-37 ←

1) Beads 30-33

Carry = 0

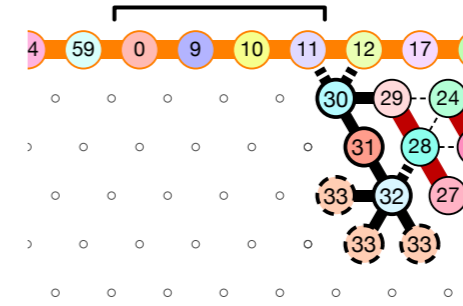
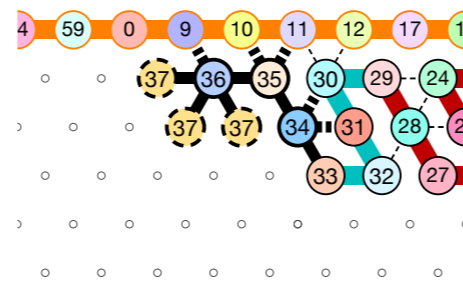


0 (A0)



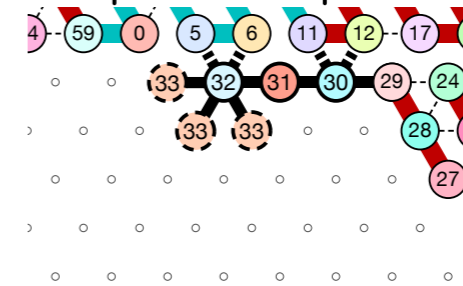
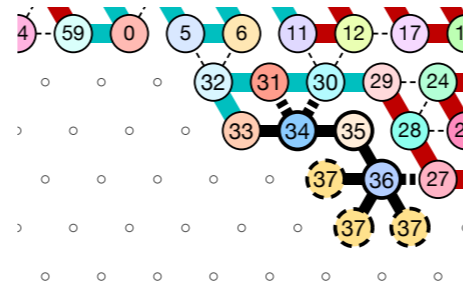
Carry = 1

0 (A0)



Carry = 0

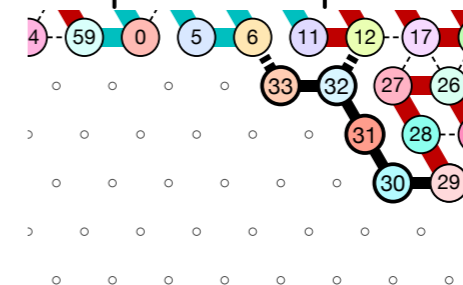
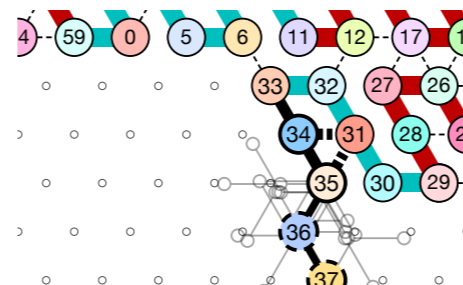
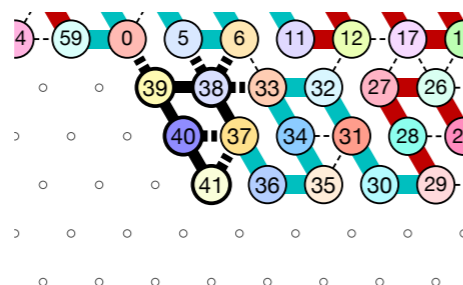
1 (A1)



Carry = 0

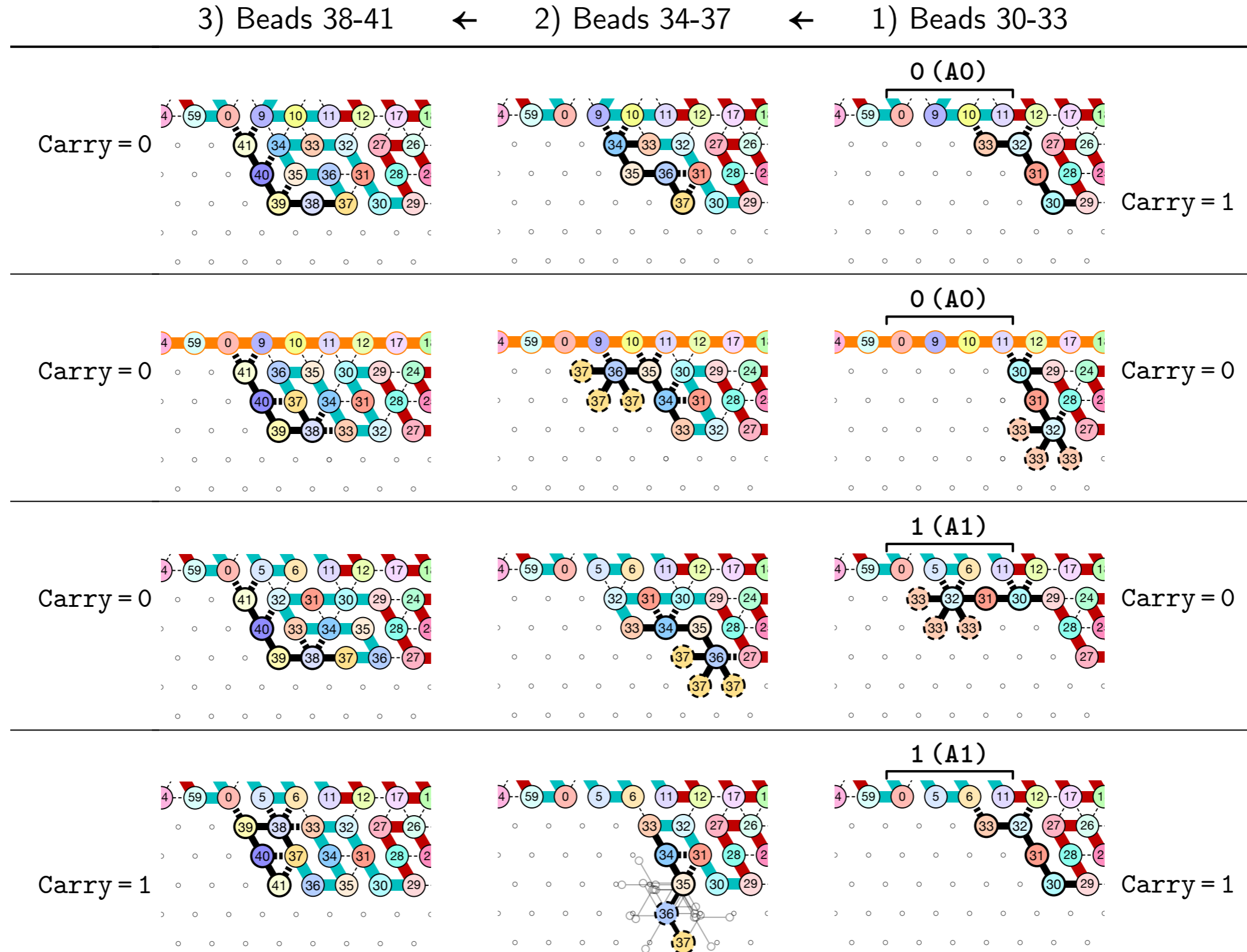
1 (A1)

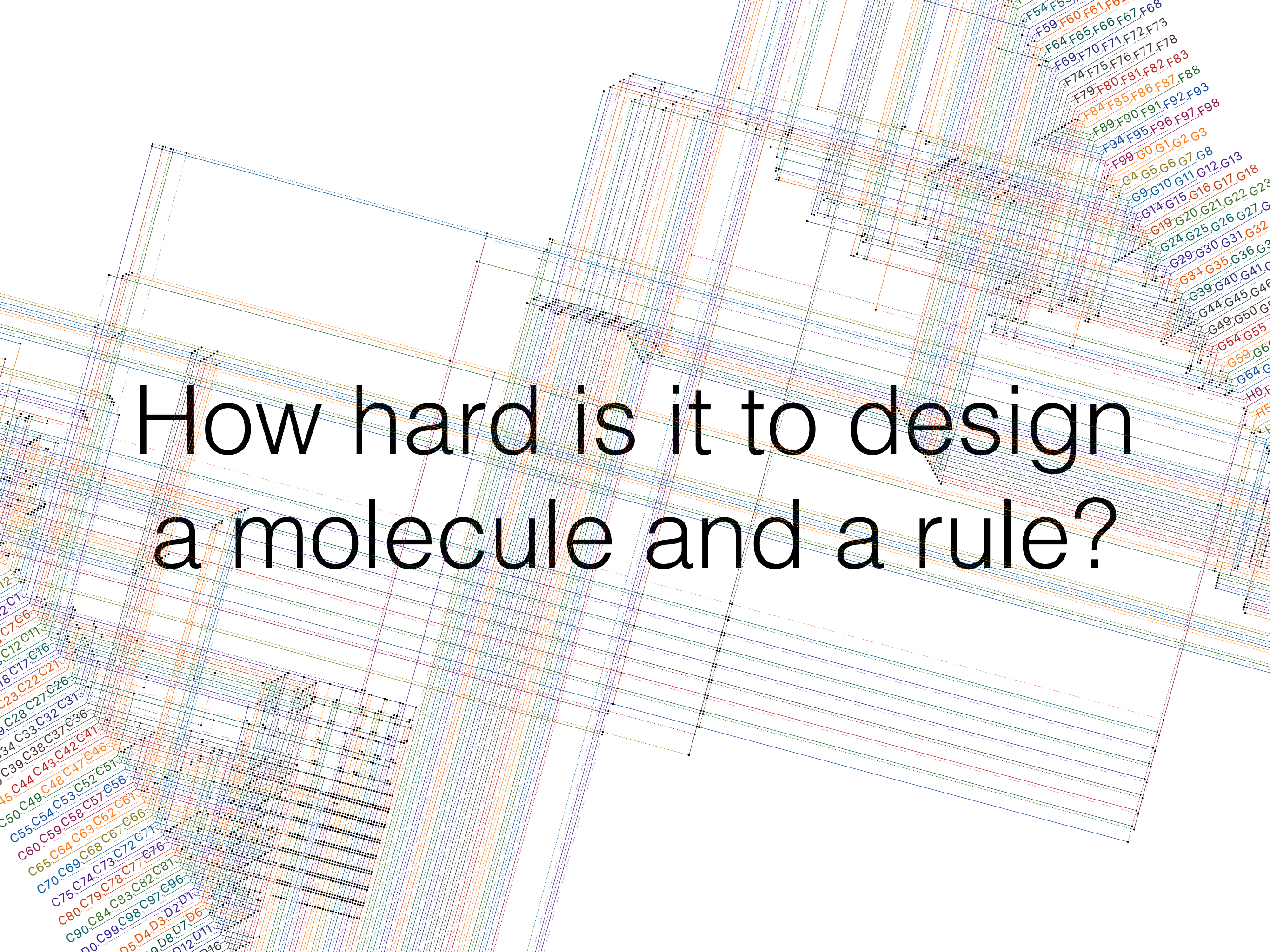
Carry = 1



Carry = 1


# How does computation work?

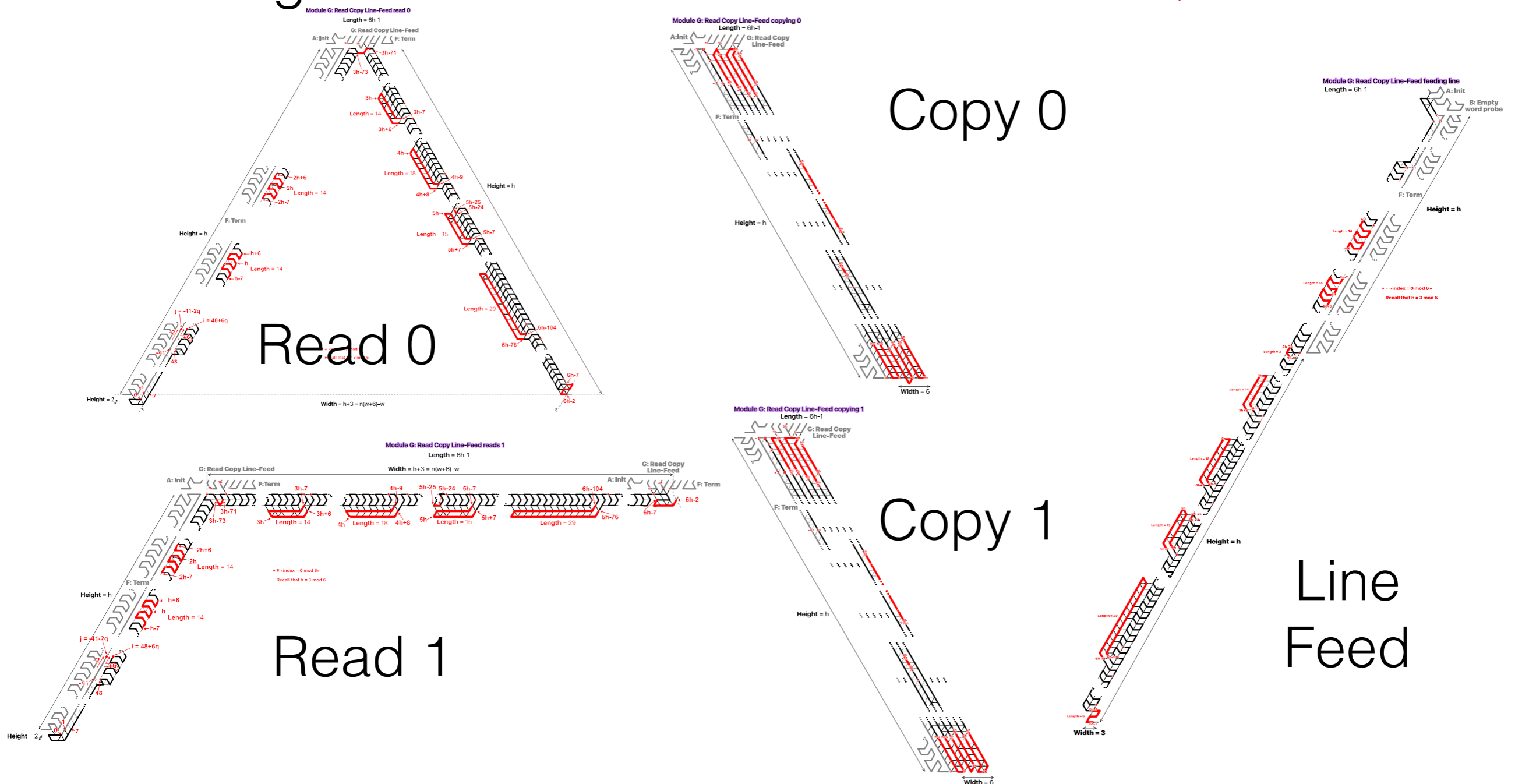


The background of the image is a dense, multi-layered network diagram. It consists of numerous nodes, represented by small black dots, arranged in a grid-like pattern. These nodes are interconnected by a complex web of edges, which are thin lines of various colors (blue, green, orange, purple, red). The network is layered, with some nodes and edges appearing in the foreground and others receding into the background, creating a sense of depth. The overall appearance is that of a highly interconnected and complex system, possibly representing a molecular structure or a computational network. Overlaid on this network is a large, bold, black text question.

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

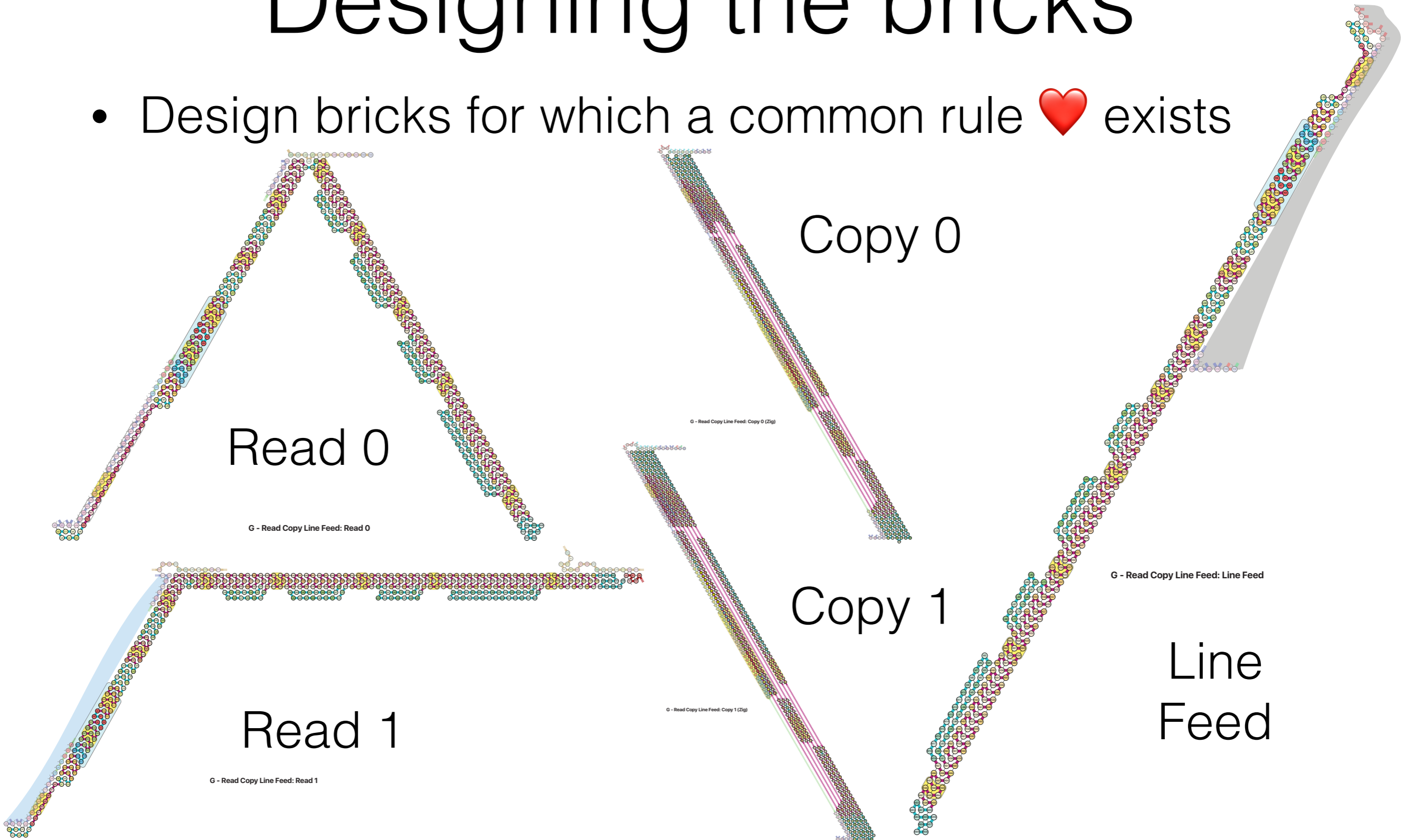
# The first challenge: Designing the bricks

- Design bricks for which a common rule  exists



# The first challenge: Designing the bricks

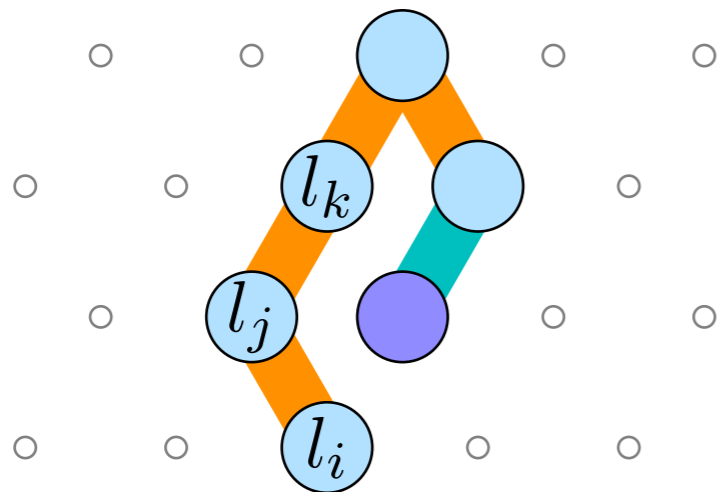
- Design bricks for which a common rule  exists



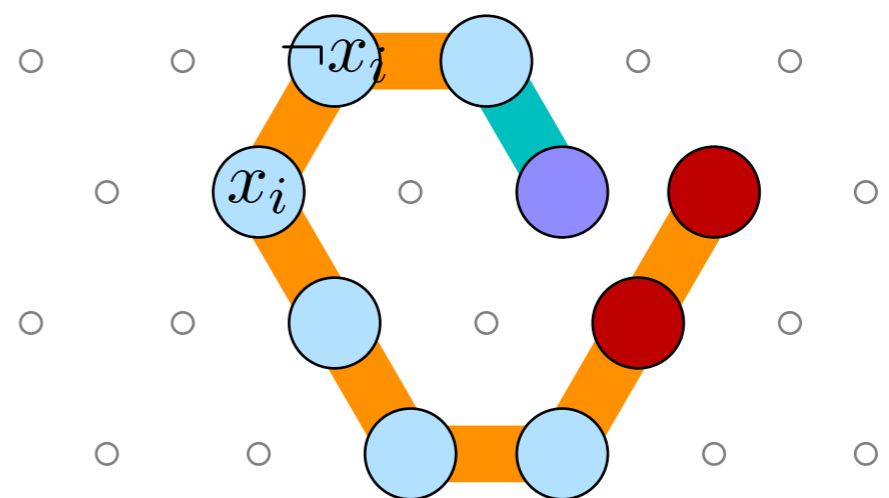
# The second challenge: Designing the rule ♥

**Theorem.** Designing a **rule** ♥ that folds a given **sequence** of length  **$L$**  into  **$k$**  prescribed conformations when folded in  **$k$**  prescribed environments is **NP-hard** in  **$k$** .

*From 3-SAT:  $L=1$  bead,  $k = n+m$  environments,  $\delta$  arbitrary*




Ensures the bead binds to  
at least one literal in  $l_i \wedge l_j \wedge l_k$



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

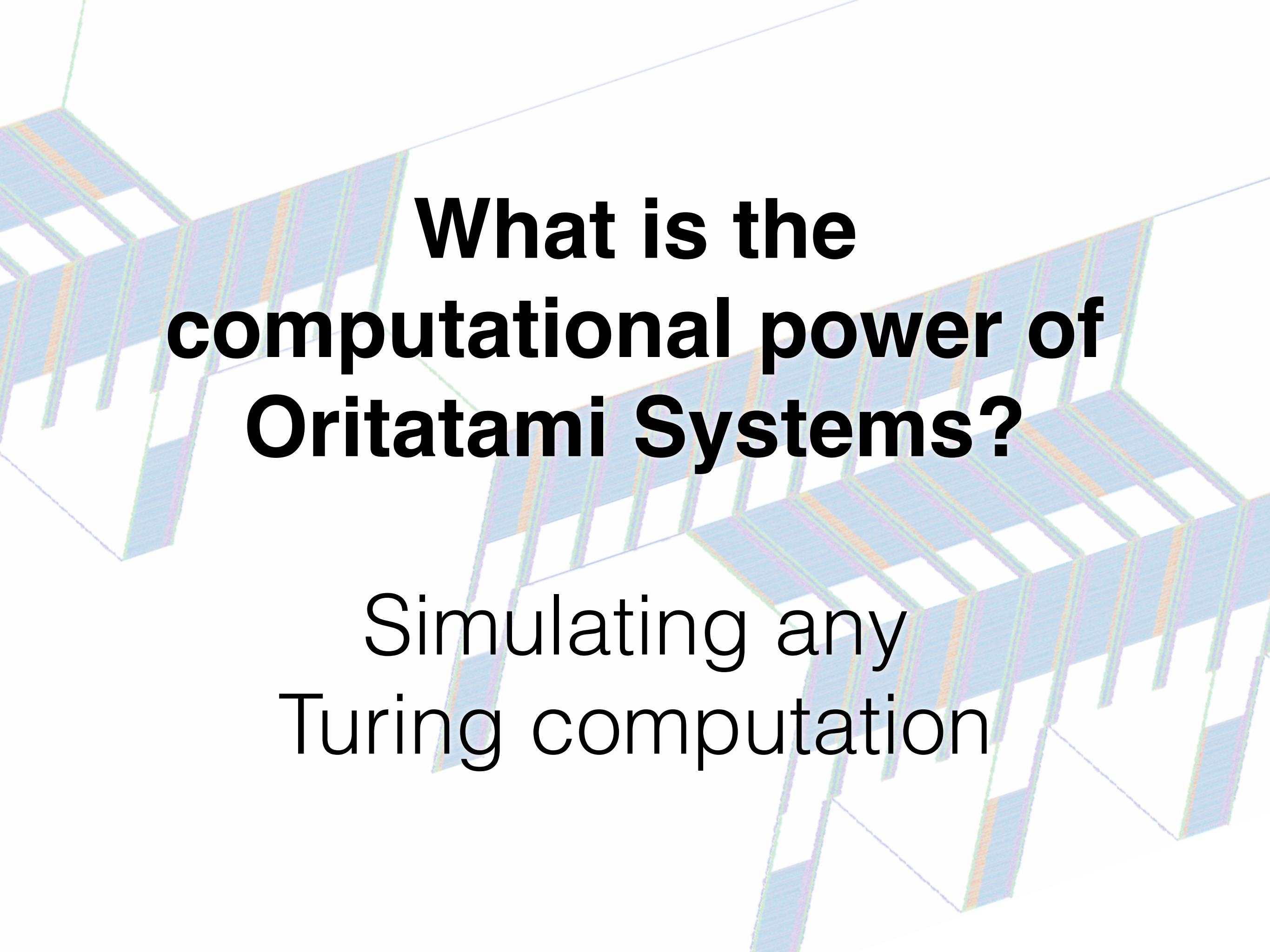
# The second challenge: Designing the rule

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

*Proof.* • **Locality:** each bead only sees a bounded number (exponential in  $\delta$ ) 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.





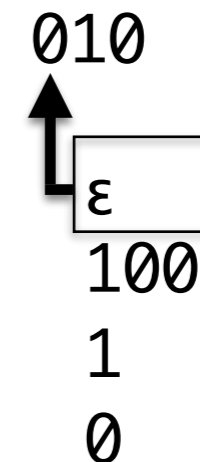
# **What is the computational power of Oritatami Systems?**

Simulating any  
Turing computation

# 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 ( $\varepsilon$ ): 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  $+2$*

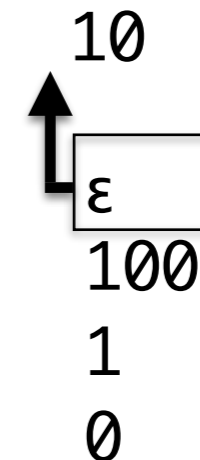
## Example.



# 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 ( $\varepsilon$ ): 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  $+2$*

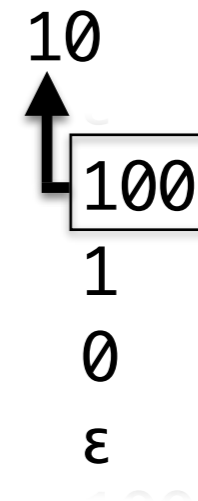
**Example.**



# 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 ( $\varepsilon$ ): 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  $+2$*

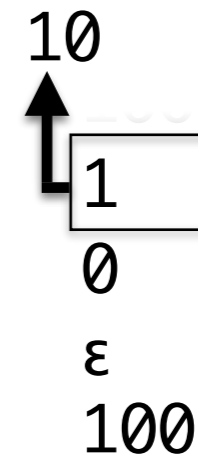
## Example.



# 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 ( $\varepsilon$ ): 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  $+2$*

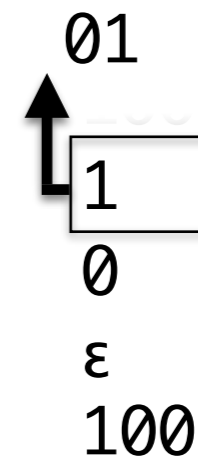
## Example.



# 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 ( $\varepsilon$ ): 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  $+2$*

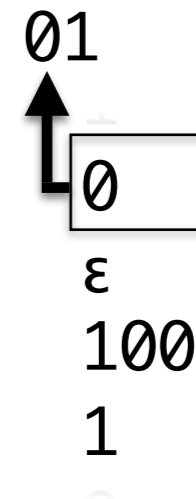
**Example.**



# 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 ( $\varepsilon$ ): 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  $+2$*

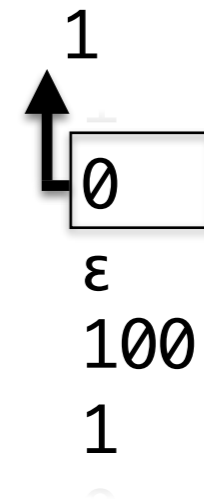
## Example.



# 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 ( $\varepsilon$ ): 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.

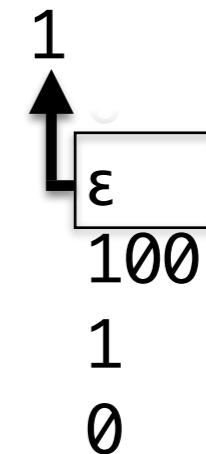




# Skipping Cyclic Tag Systems

- 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 ( $\varepsilon$ ): 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  $+2$*

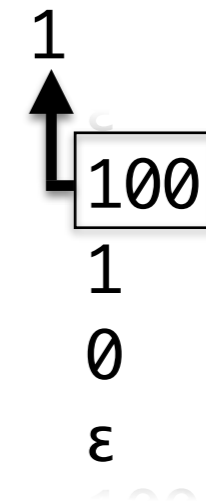
## Example.



# 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 ( $\varepsilon$ ): 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  $+2$*

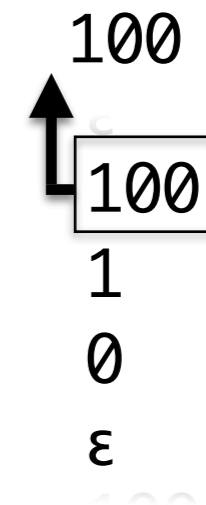
## Example.



# 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 ( $\varepsilon$ ): 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  $+2$*

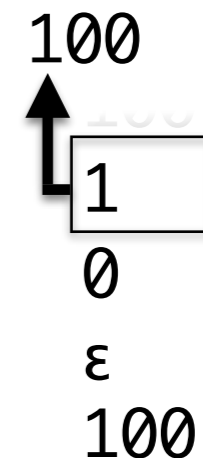
## Example.



# 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 ( $\varepsilon$ ): 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  $+2$*

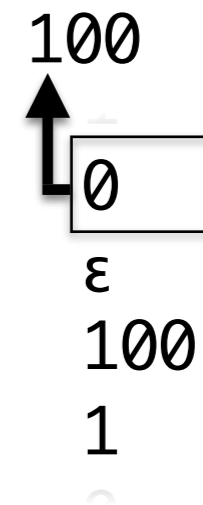
## Example.



# 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 ( $\varepsilon$ ): 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.

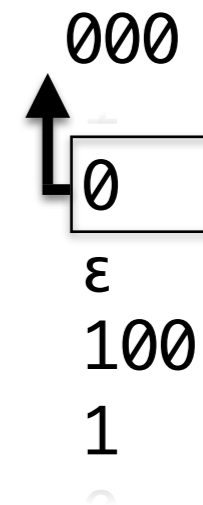


# Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer  $p$  to one of them
- An initial binary **tape word** (the input)

## Example.

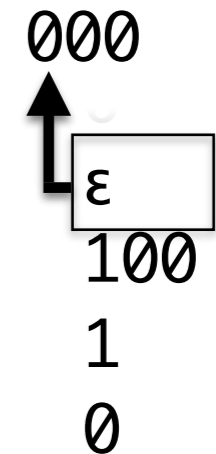
- **Dynamics:**
  - *If the tape word is empty ( $\varepsilon$ ): halt*
  - *If the 1st letter of the tape word is 0: delete the 0 and increment the pointer  $p$*
  - *If the 1st letter of the tape word is 1: delete the 1, append to the tape word the code word at position  $p+1$  and increase  $p$  by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



# 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 ( $\varepsilon$ ): 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  $+2$*

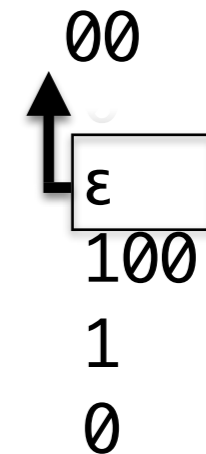
**Example.**



# 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 ( $\varepsilon$ ): 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.**



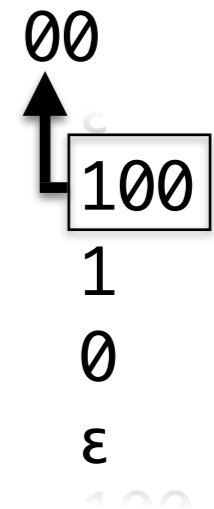


# Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer  $p$  to one of them
- An initial binary **tape word** (the input)

## Example.

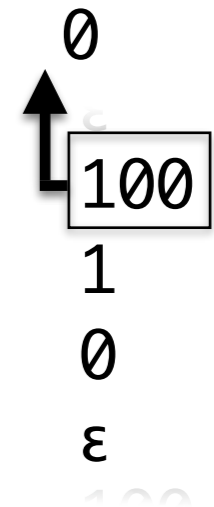
- **Dynamics:**
  - *If the tape word is empty ( $\varepsilon$ ): halt*
  - *If the 1st letter of the tape word is 0: delete the 0 and increment the pointer  $p$*
  - *If the 1st letter of the tape word is 1: delete the 1, append to the tape word the code word at position  $p+1$  and increase  $p$  by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



# 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 ( $\varepsilon$ ): 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  $+2$*

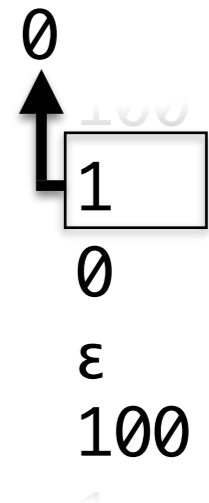
**Example.**



# 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 ( $\varepsilon$ ): 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.**

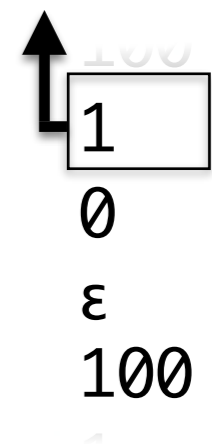


# Skipping Cyclic Tag Systems

- A finite **cyclic sequence** of finite binary **code words** with a pointer  $p$  to one of them
- An initial binary **tape word** (the input)

**Example.**

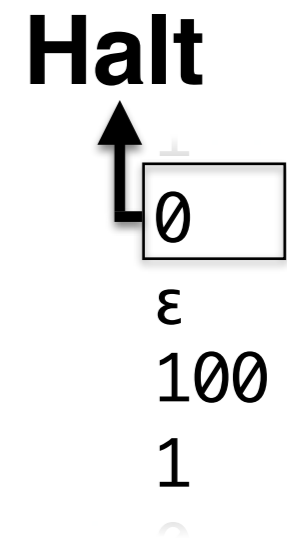
- **Dynamics:**
  - *If the tape word is empty ( $\varepsilon$ ): halt*
  - *If the 1st letter of the tape word is 0: delete the 0 and increment the pointer  $p$*
  - *If the 1st letter of the tape word is 1: delete the 1, append to the tape word the code word at position  $p+1$  and increase  $p$  by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



# 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 ( $\varepsilon$ ): 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  $+2$*

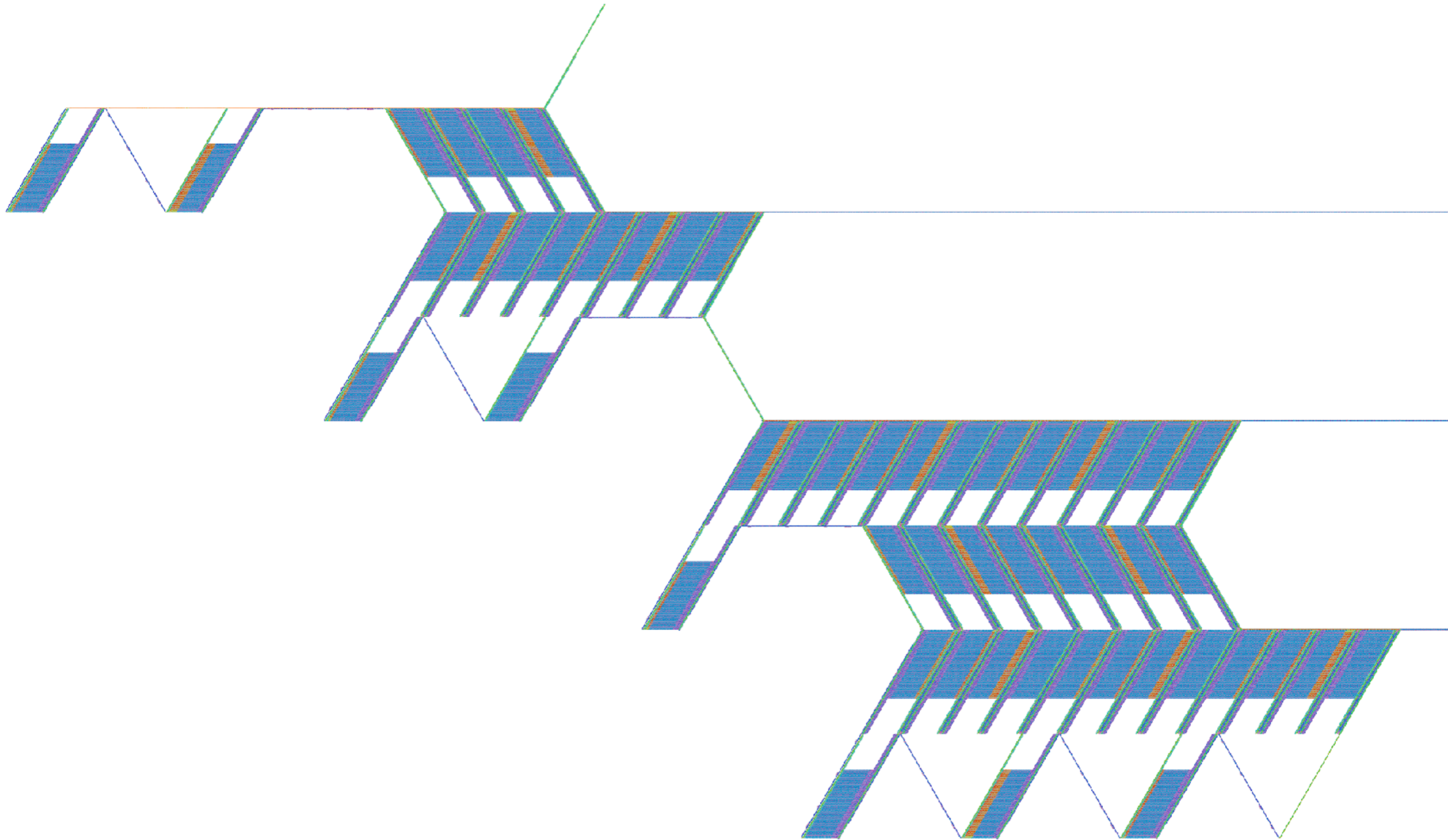
**Example.**



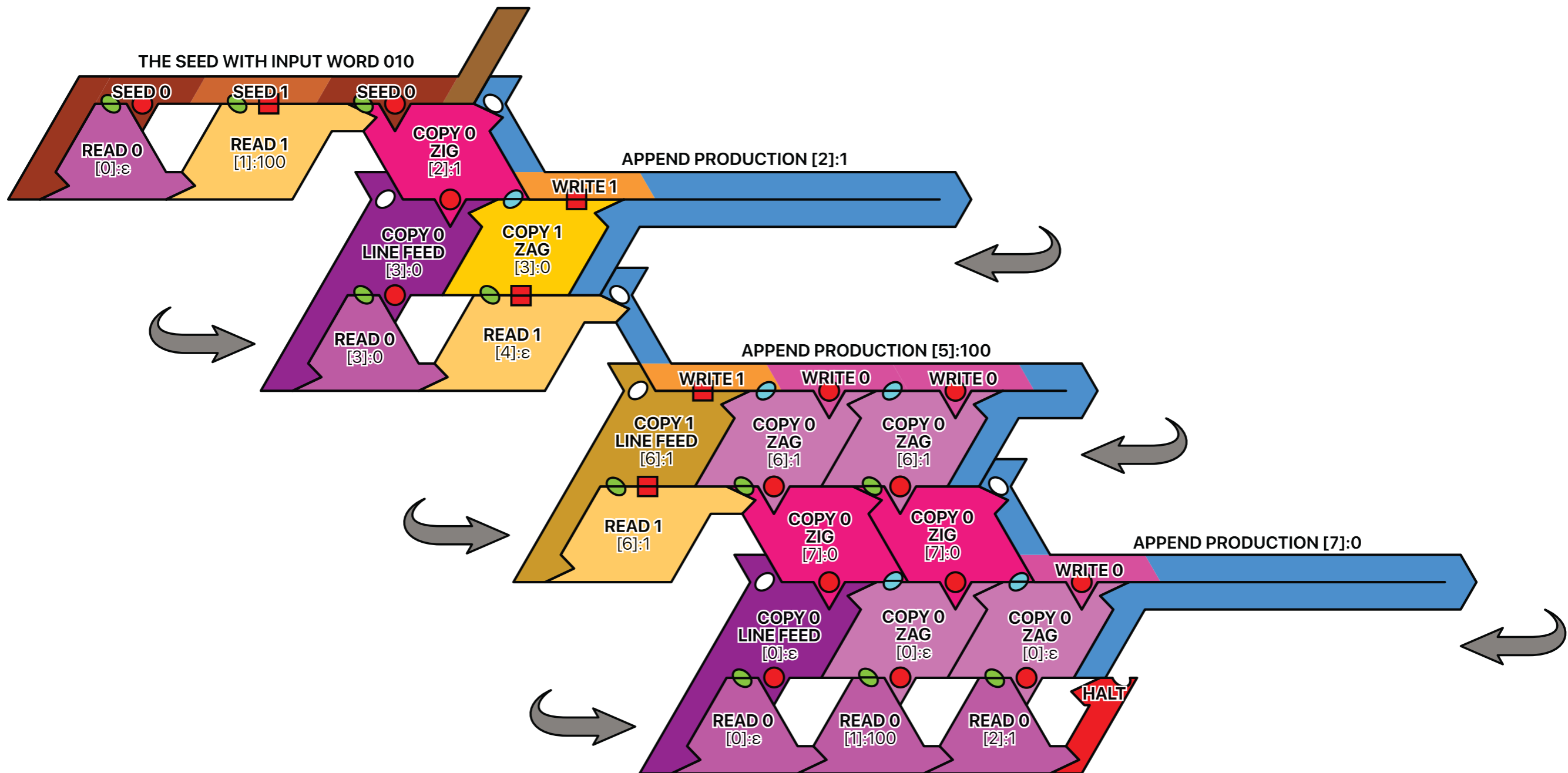
# Skipping Cyclic Tag Systems

- 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 ( $\varepsilon$ ): 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



# The block view





# The shapes & functions of the blocks

**READ**

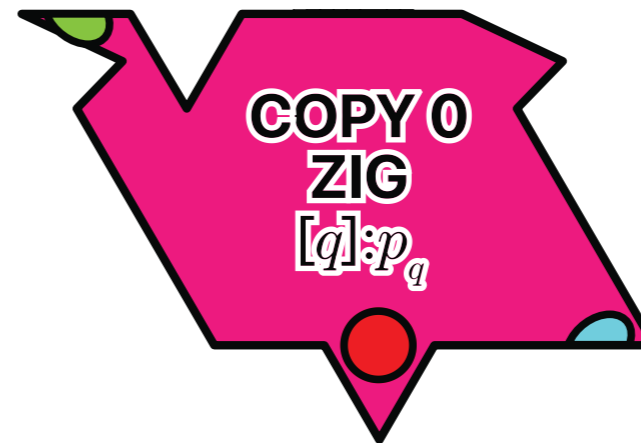
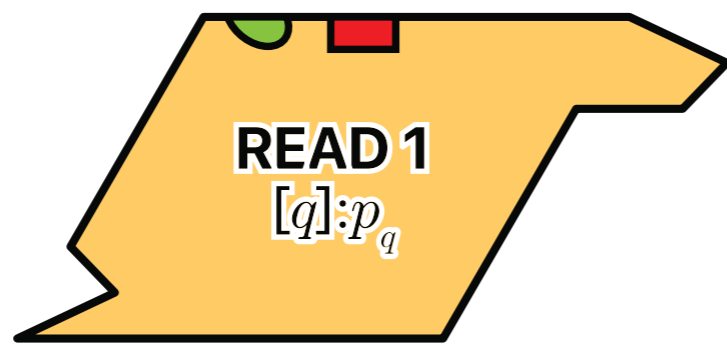
**COPY**

read 0

read 1

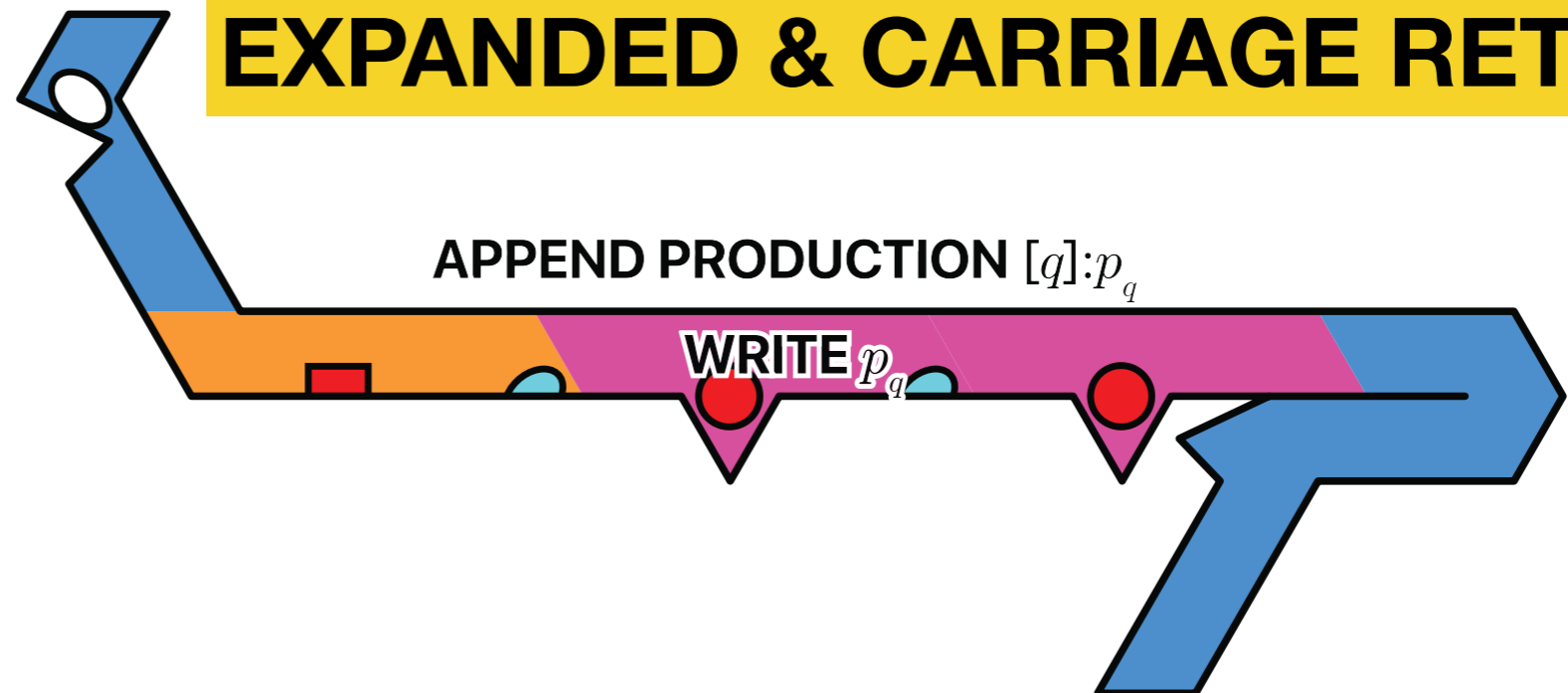
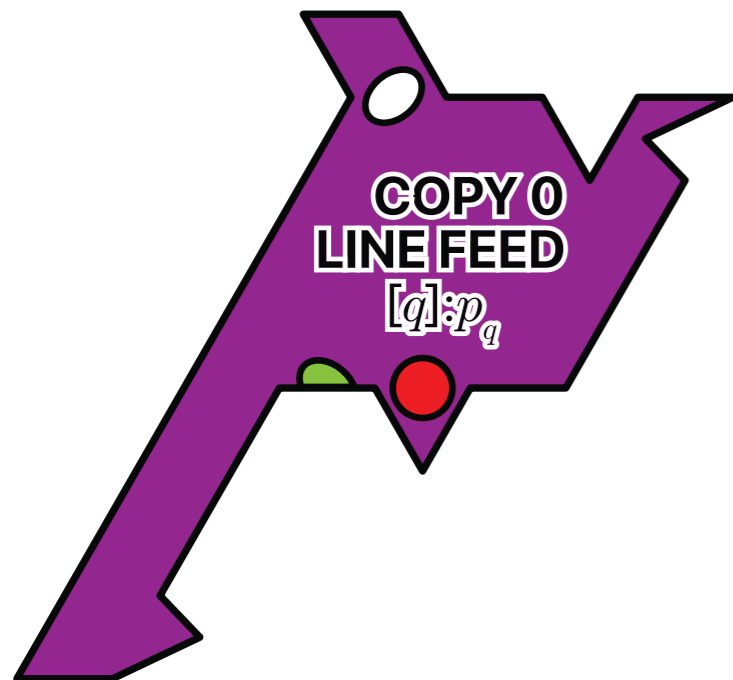
forward →

backward ←



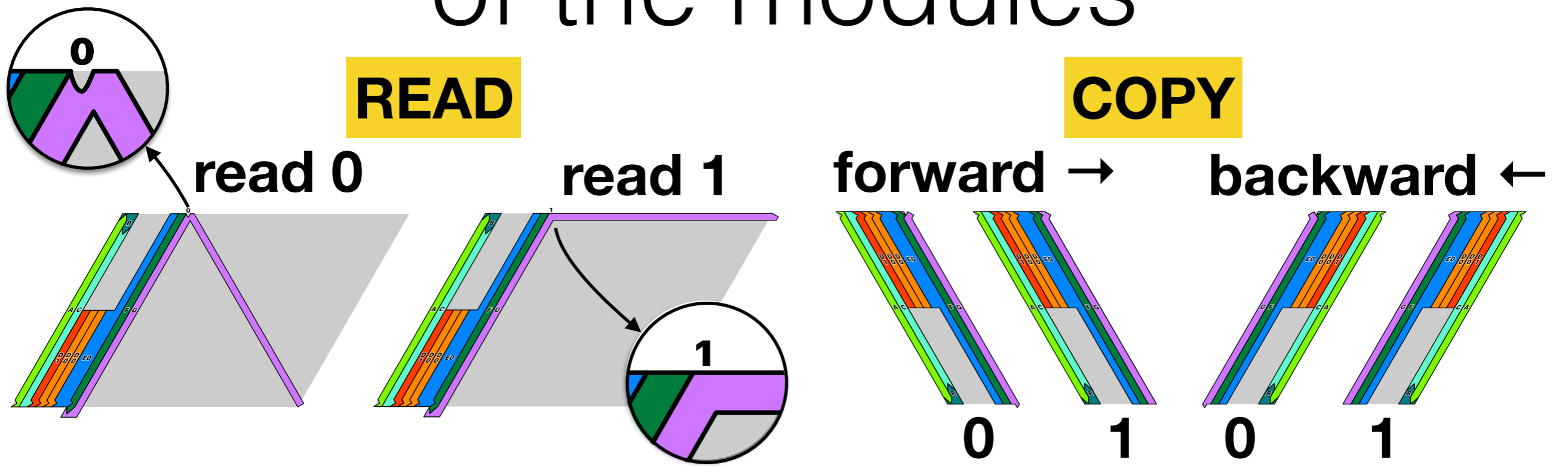
**LINE FEED**

**EXPANDED & CARRIAGE RETURN**

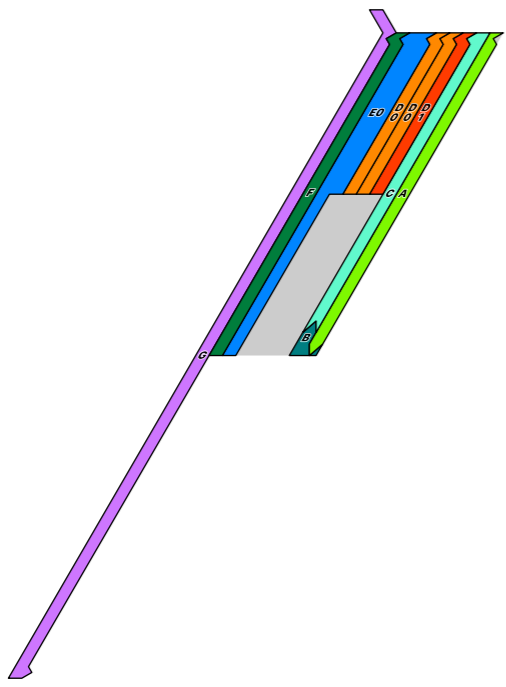




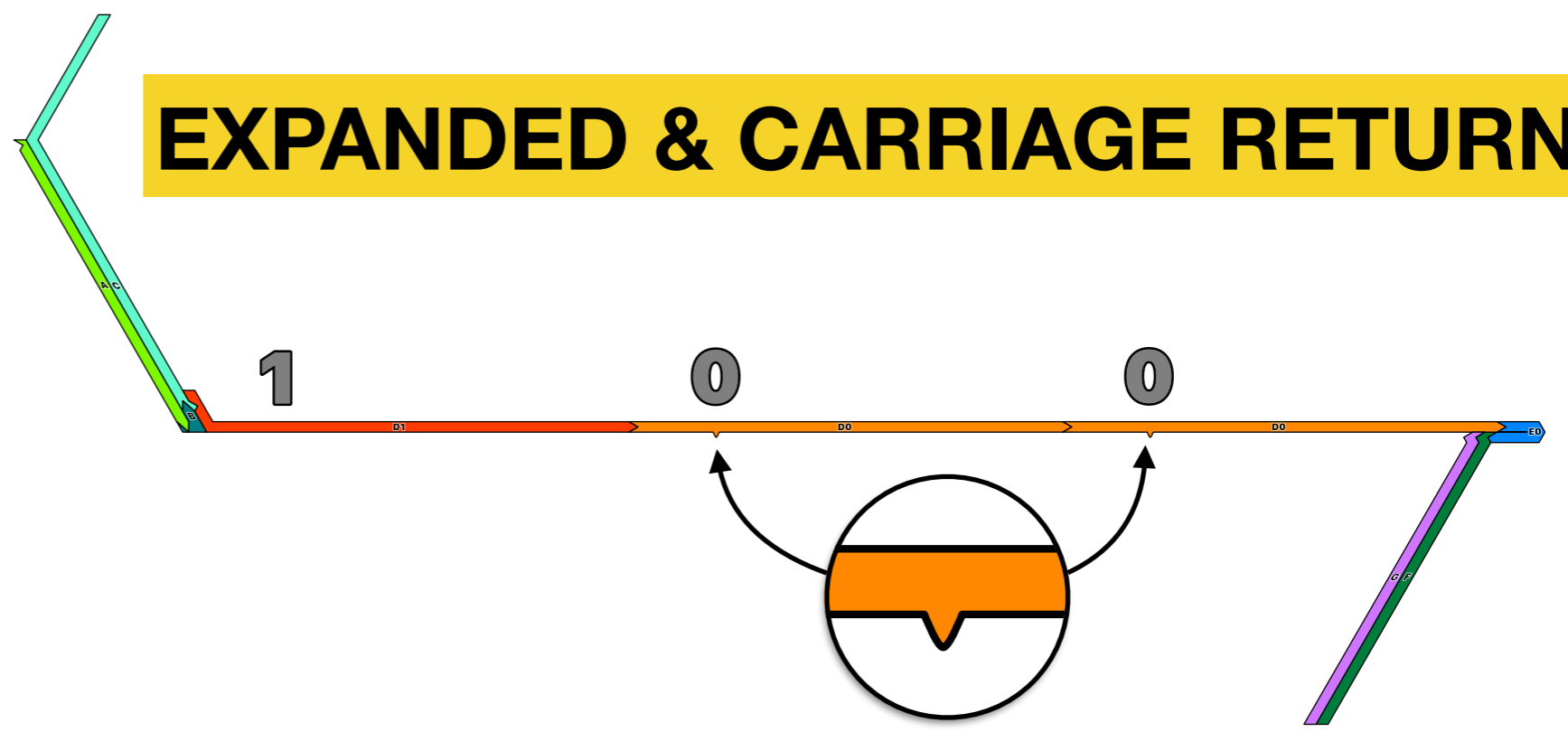
# The shapes & functions of the modules



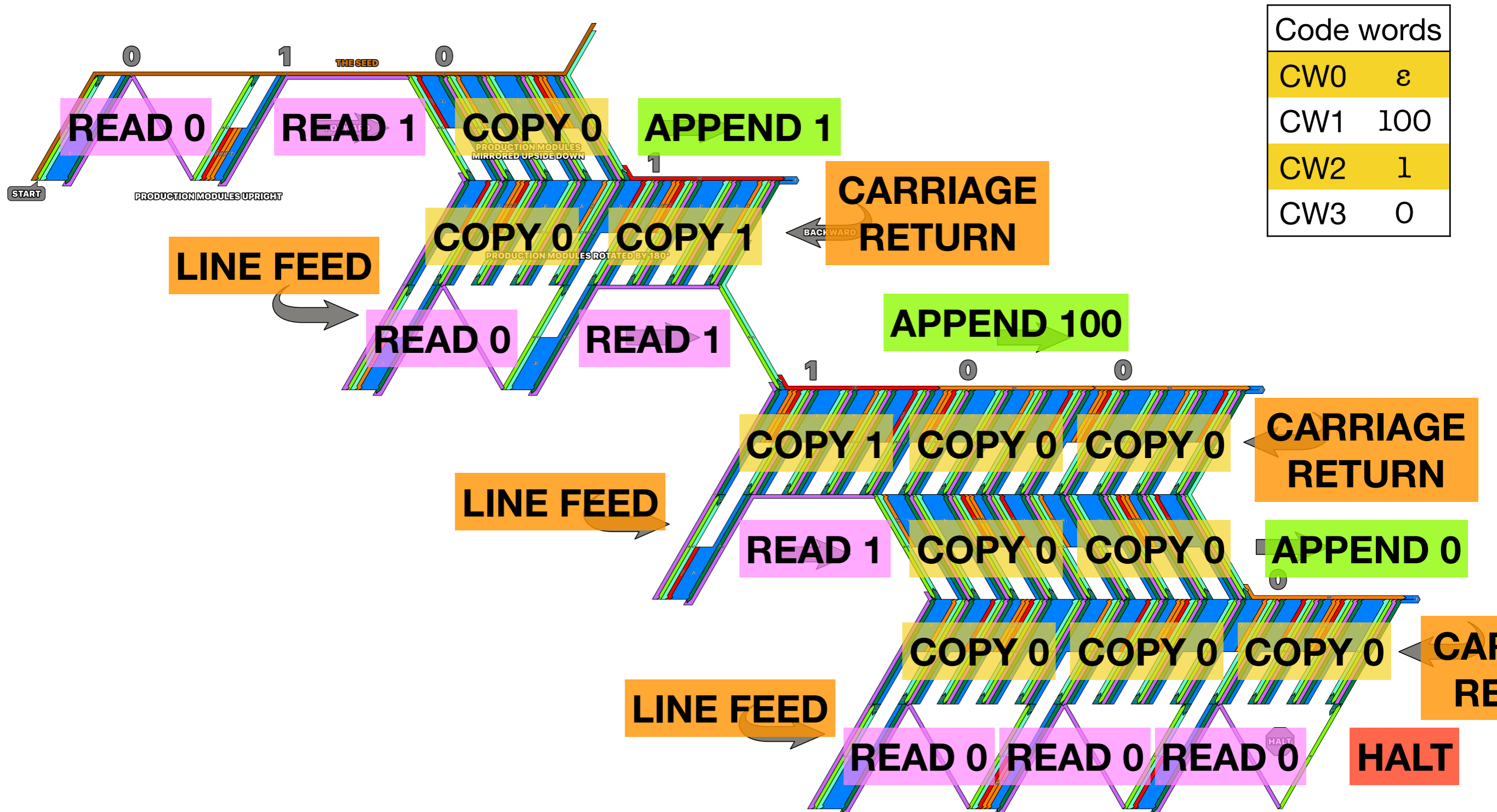
## LINE FEED



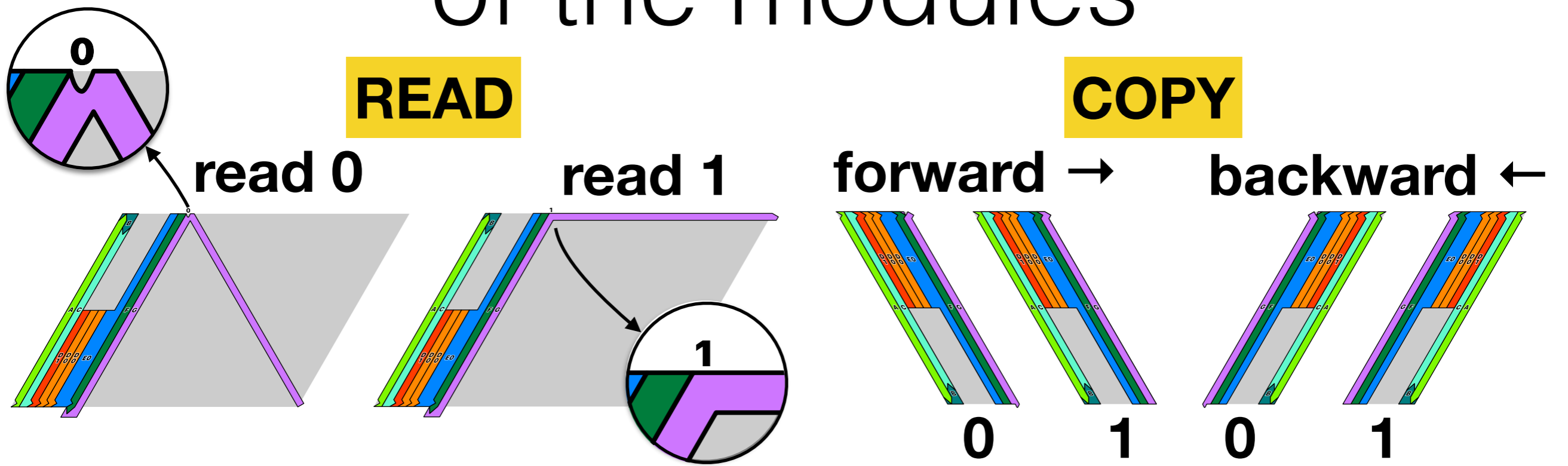
## EXPANDED & CARRIAGE RETURN



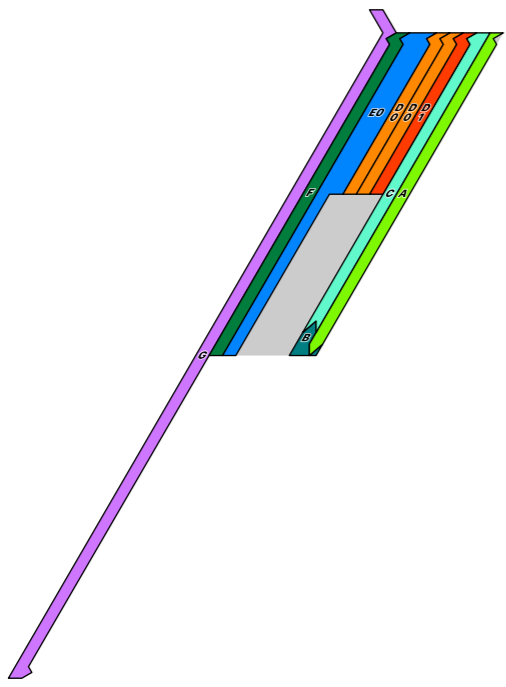
# How does the tag system simulation work?



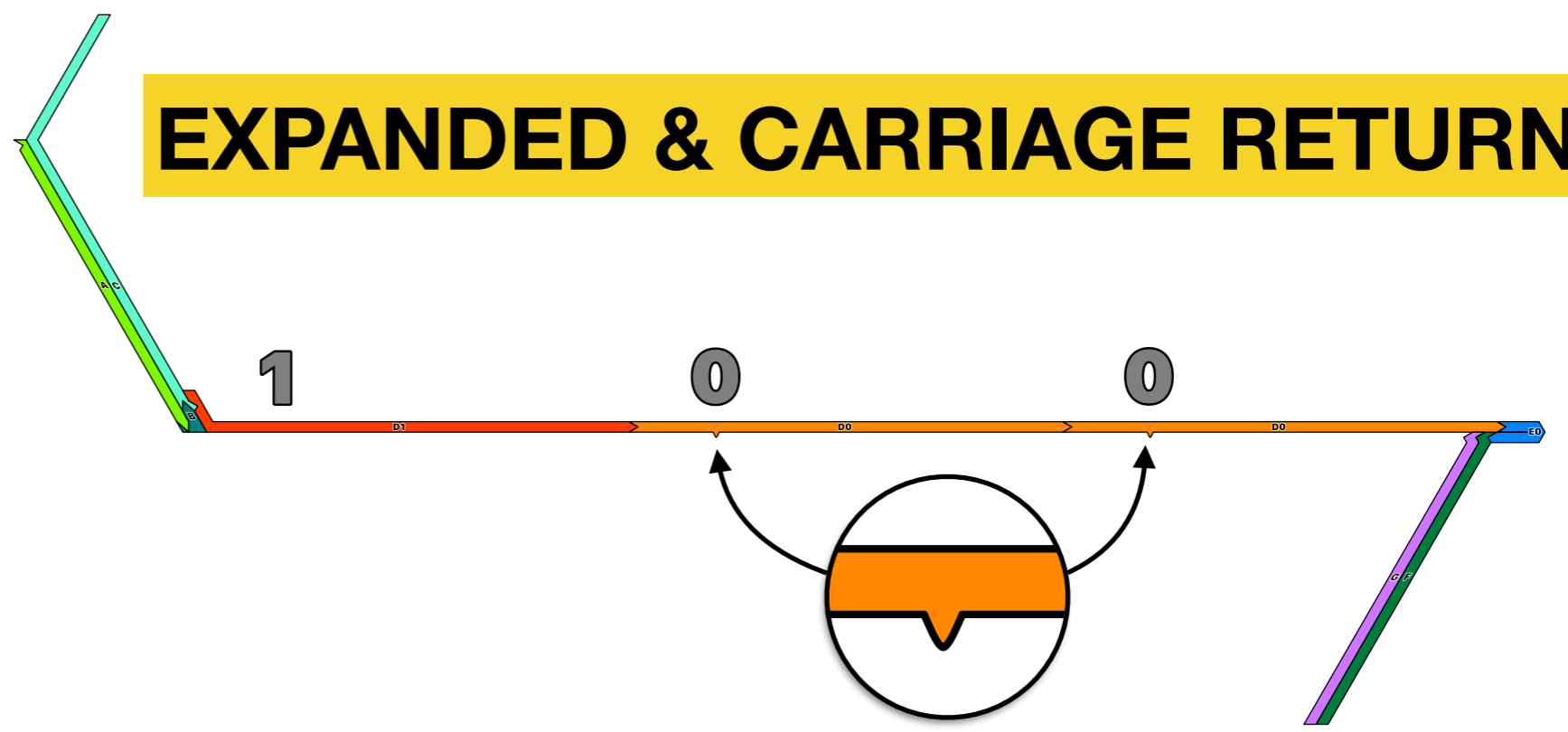
# The shapes & functions of the modules



## LINE FEED



## EXPANDED & CARRIAGE RETURN



# The shapes & functions of the blocks

**READ**

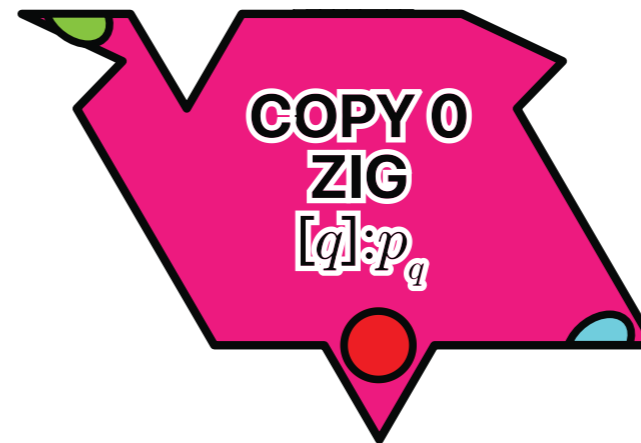
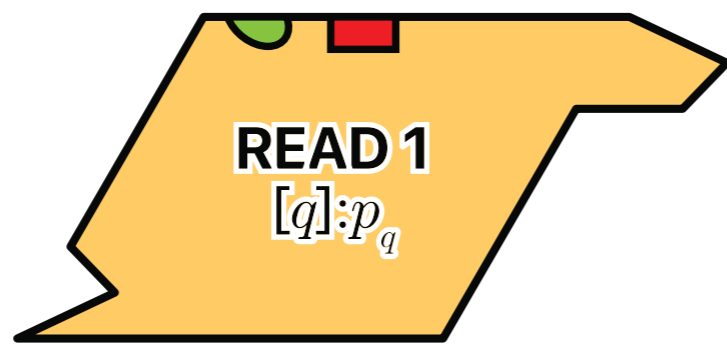
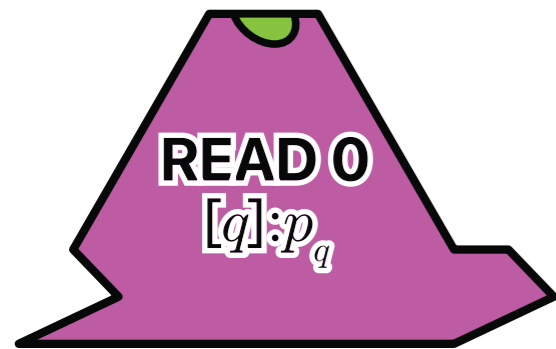
**COPY**

read 0

read 1

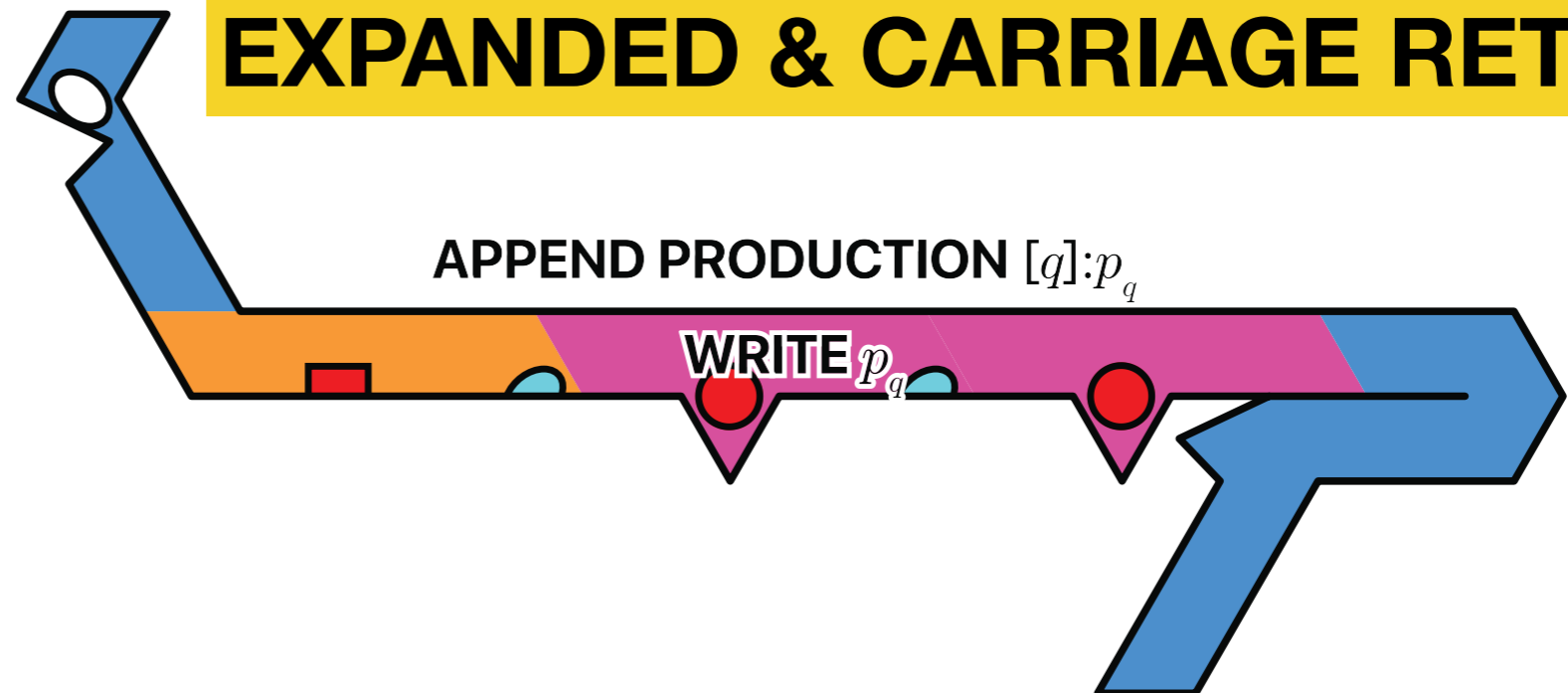
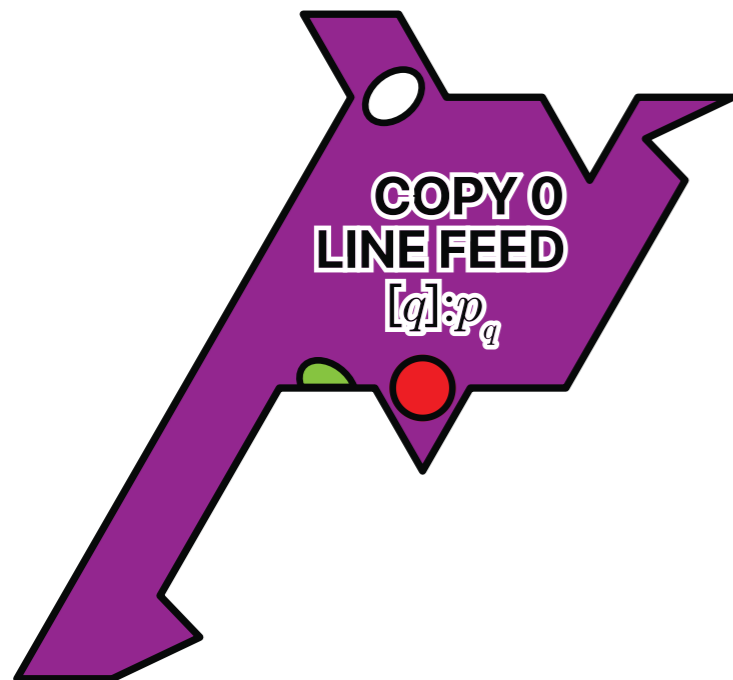
forward →

backward ←

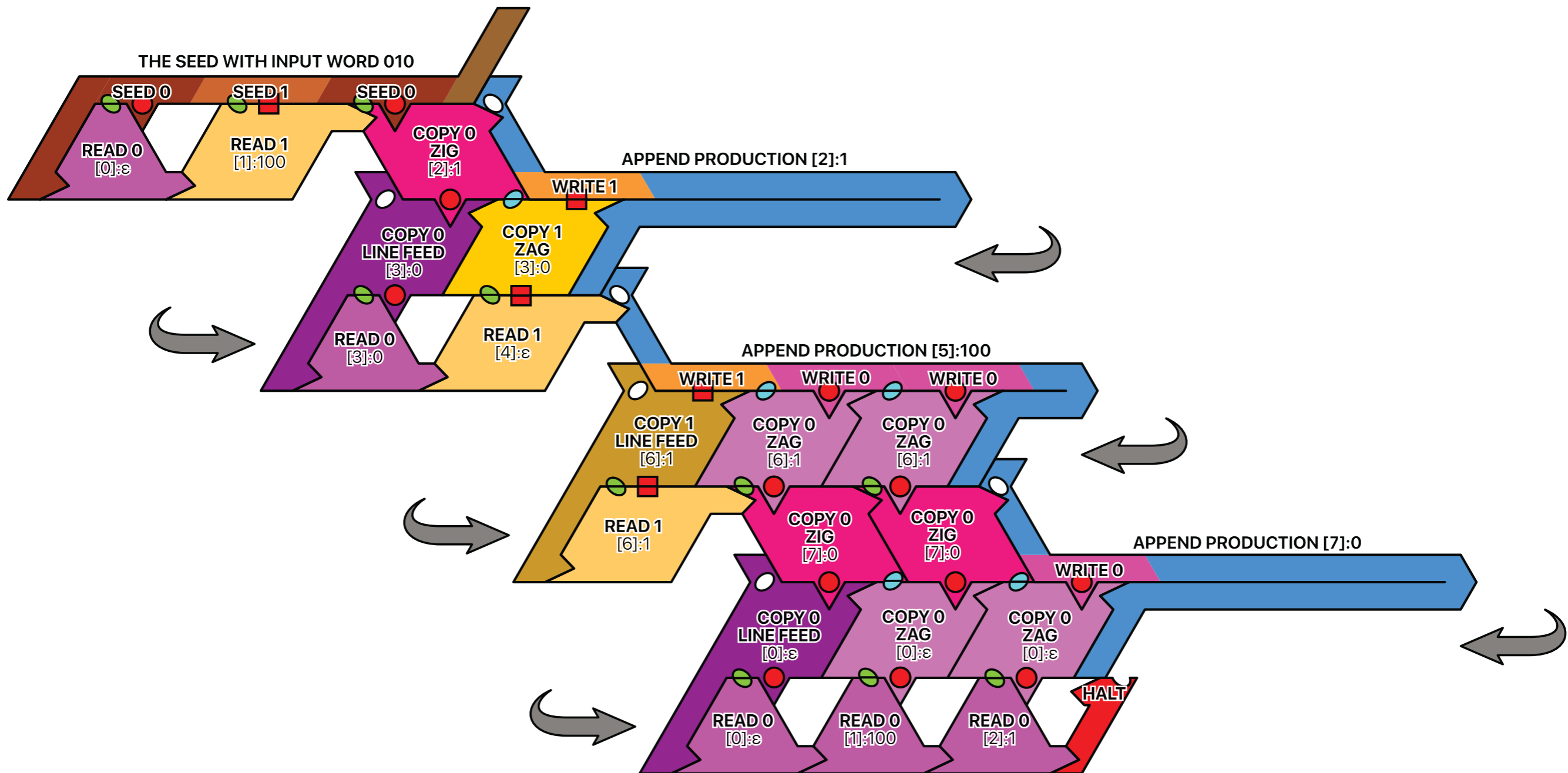


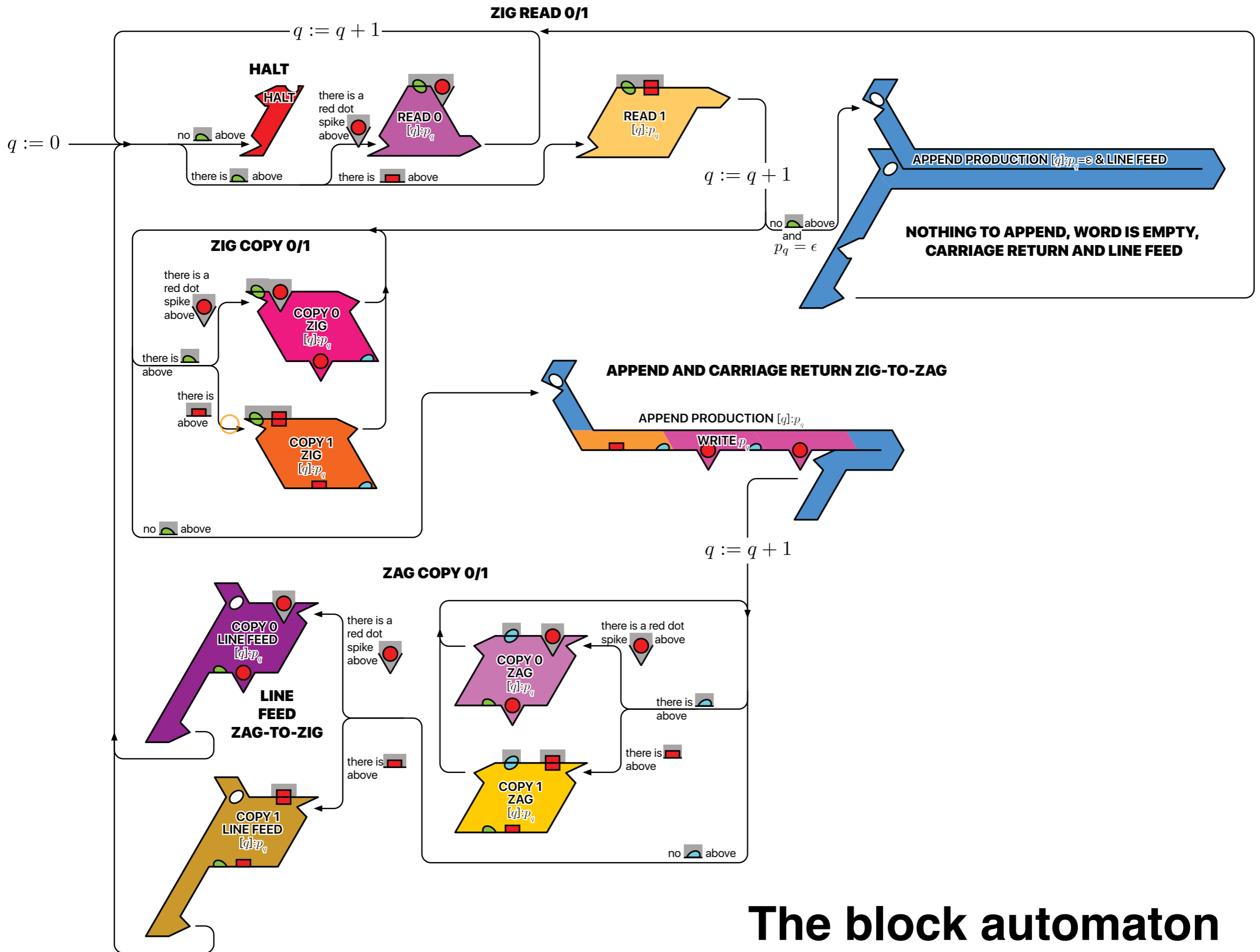
**LINE FEED**

**EXPANDED & CARRIAGE RETURN**



# The block view

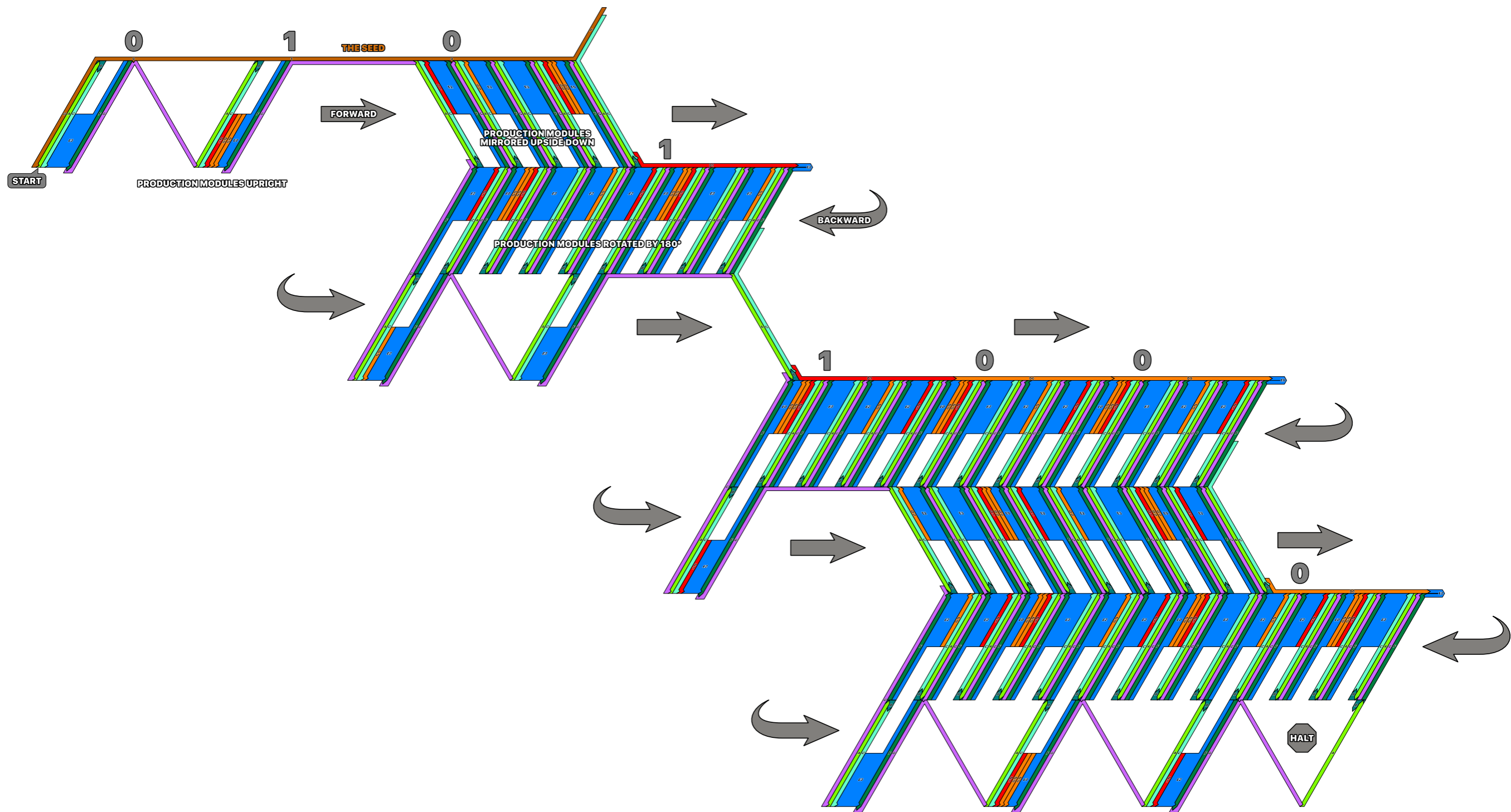




**The block automaton**

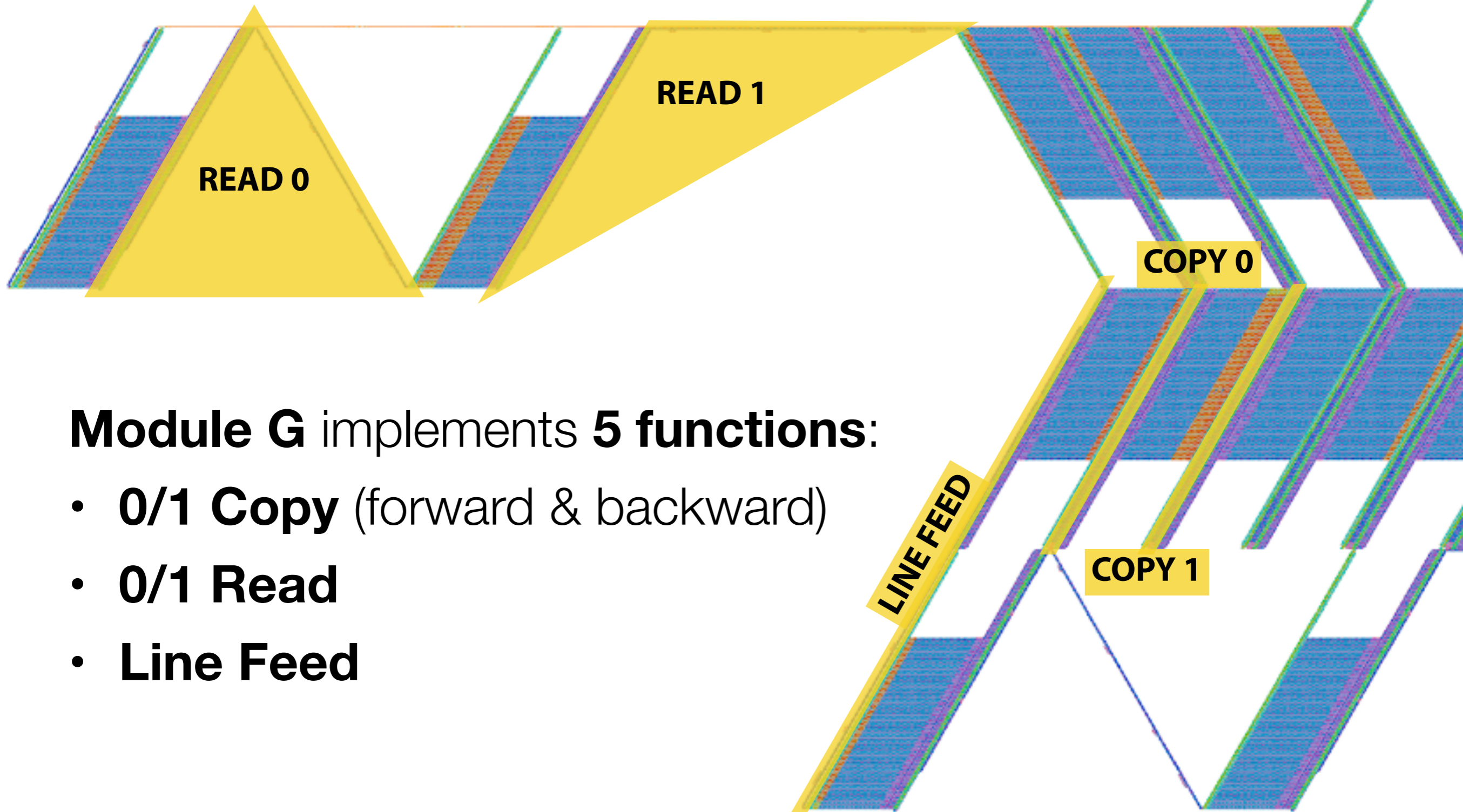


# The module view



How do we implement  
several functions in  
module?

# An example

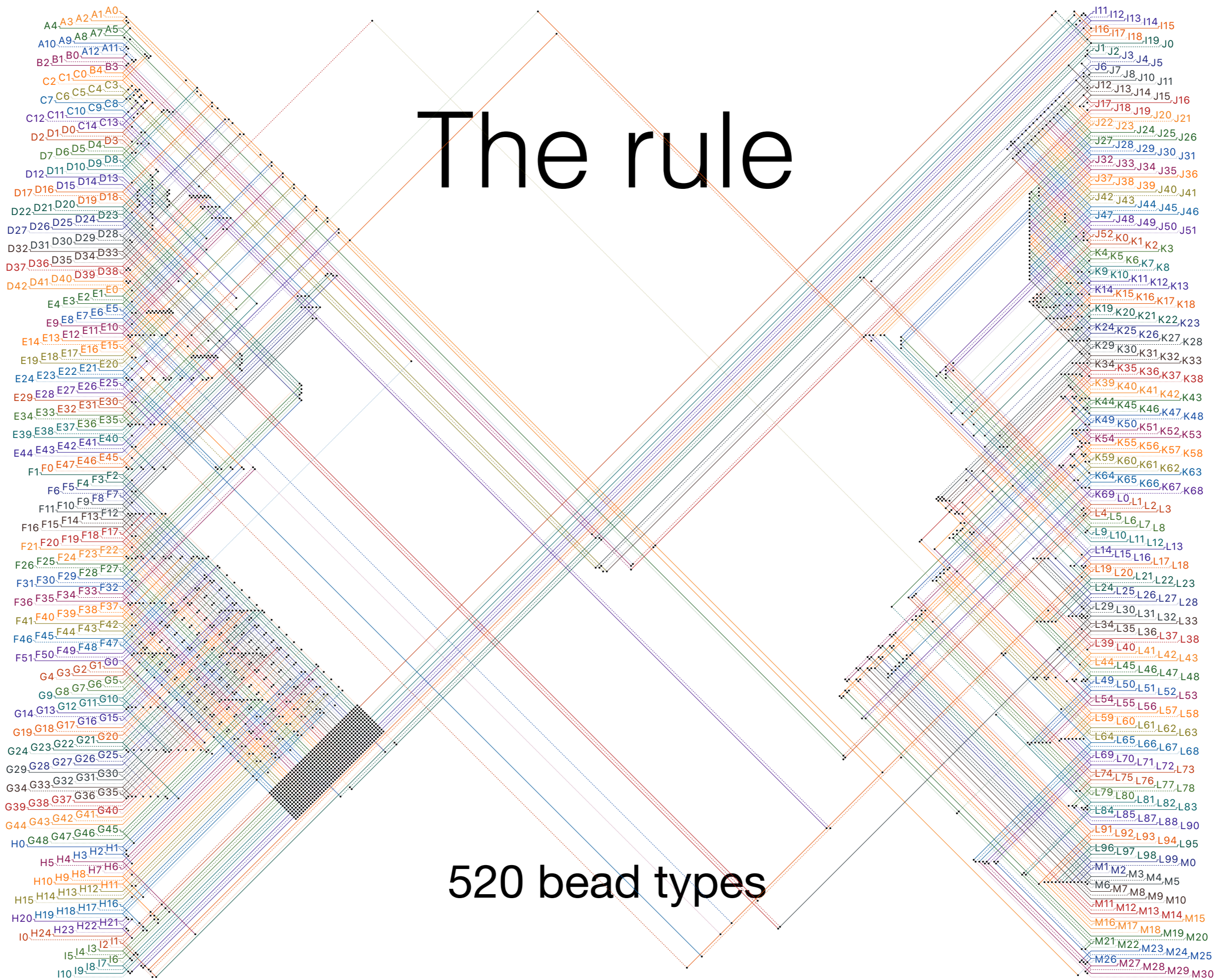


**Module G** implements **5 functions**:

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



# The rule



520 bead types

Proving the correctness  
of the folding

# First, enumerate all the bricks inside the modules and blocks

**READ**

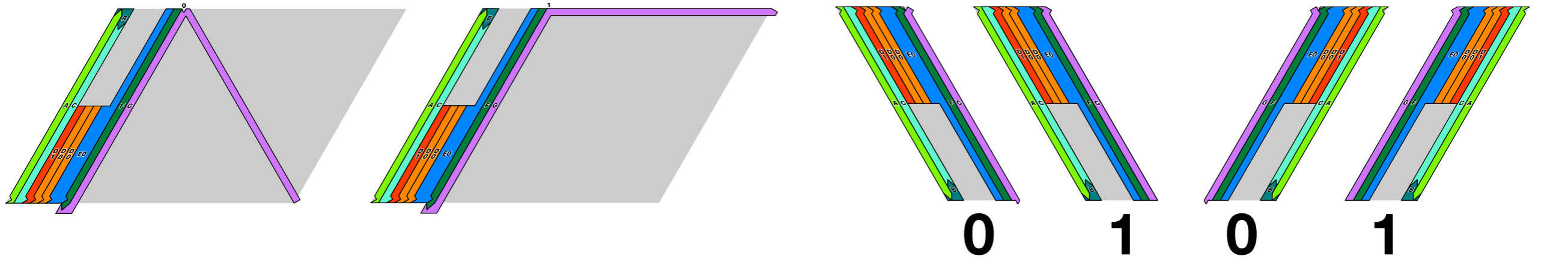
**COPY**

read 0

read 1

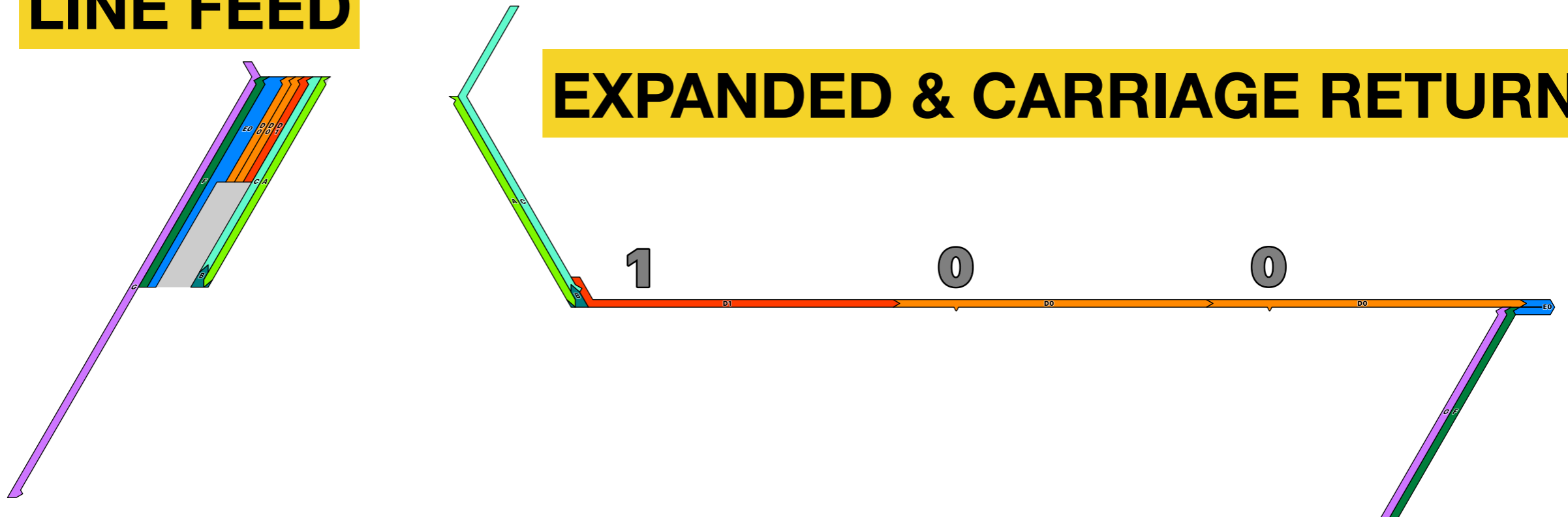
forward →

backward ←

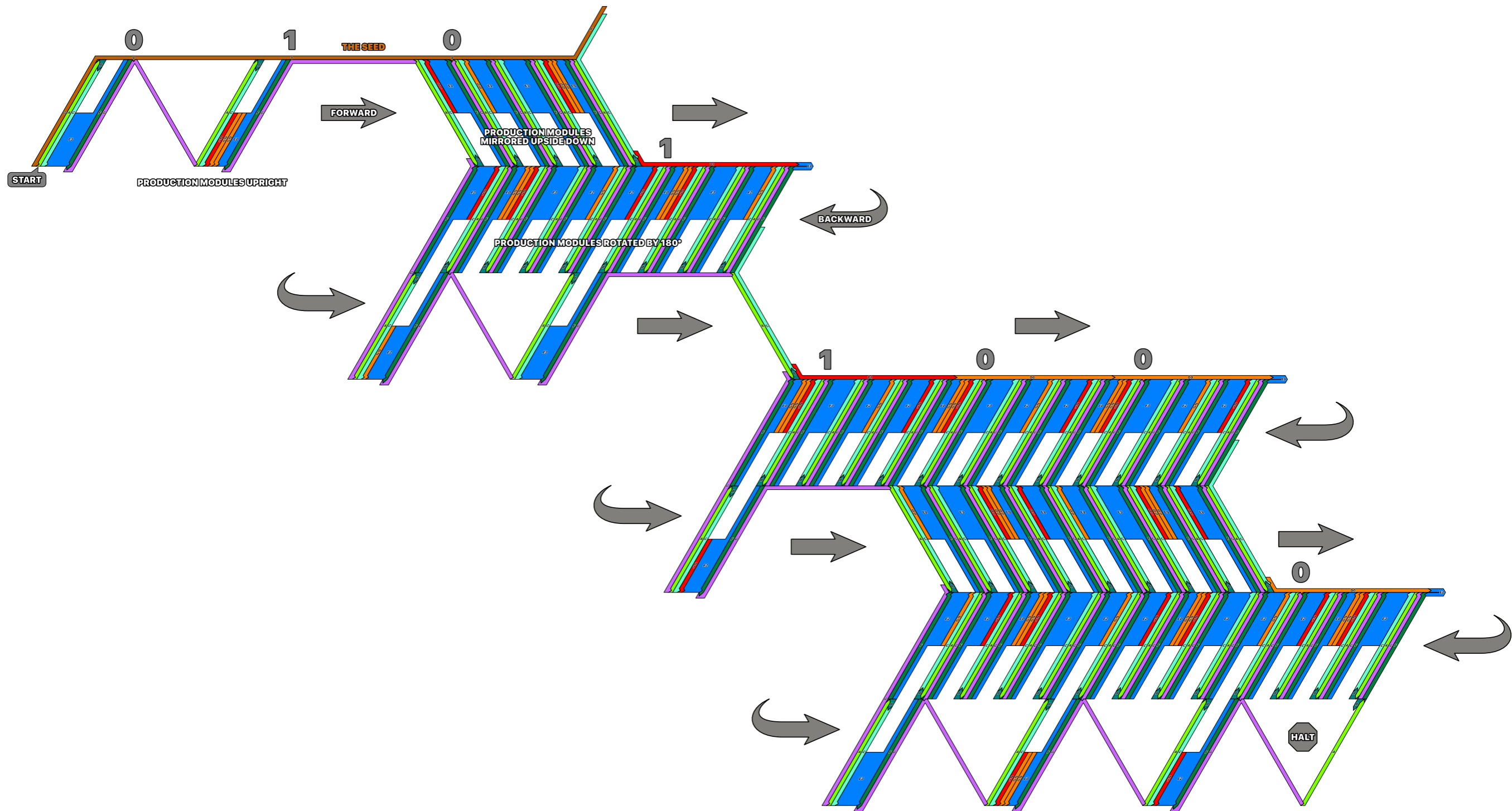


**LINE FEED**

**EXPANDED & CARRIAGE RETURN**

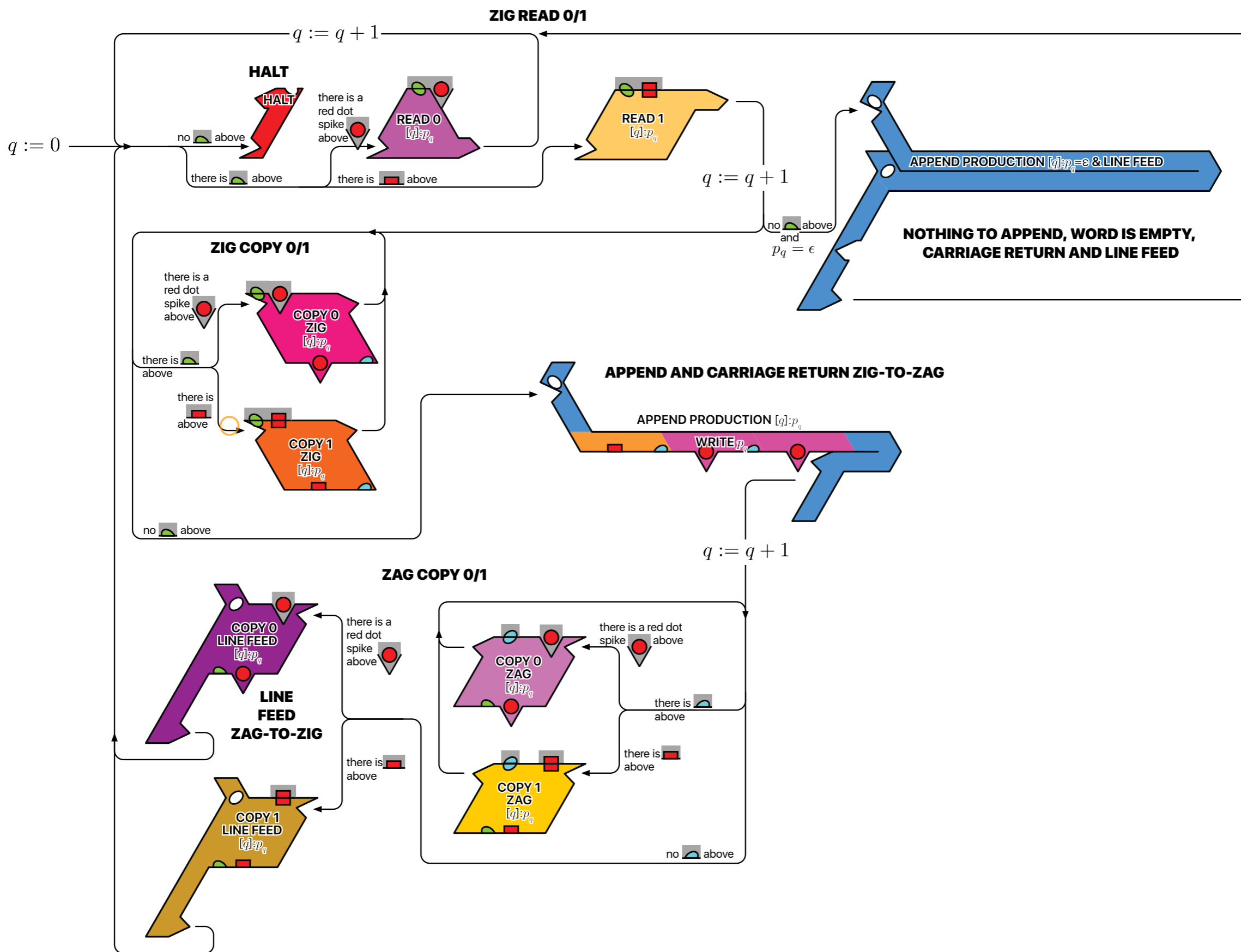


# Second, describe the final conformation

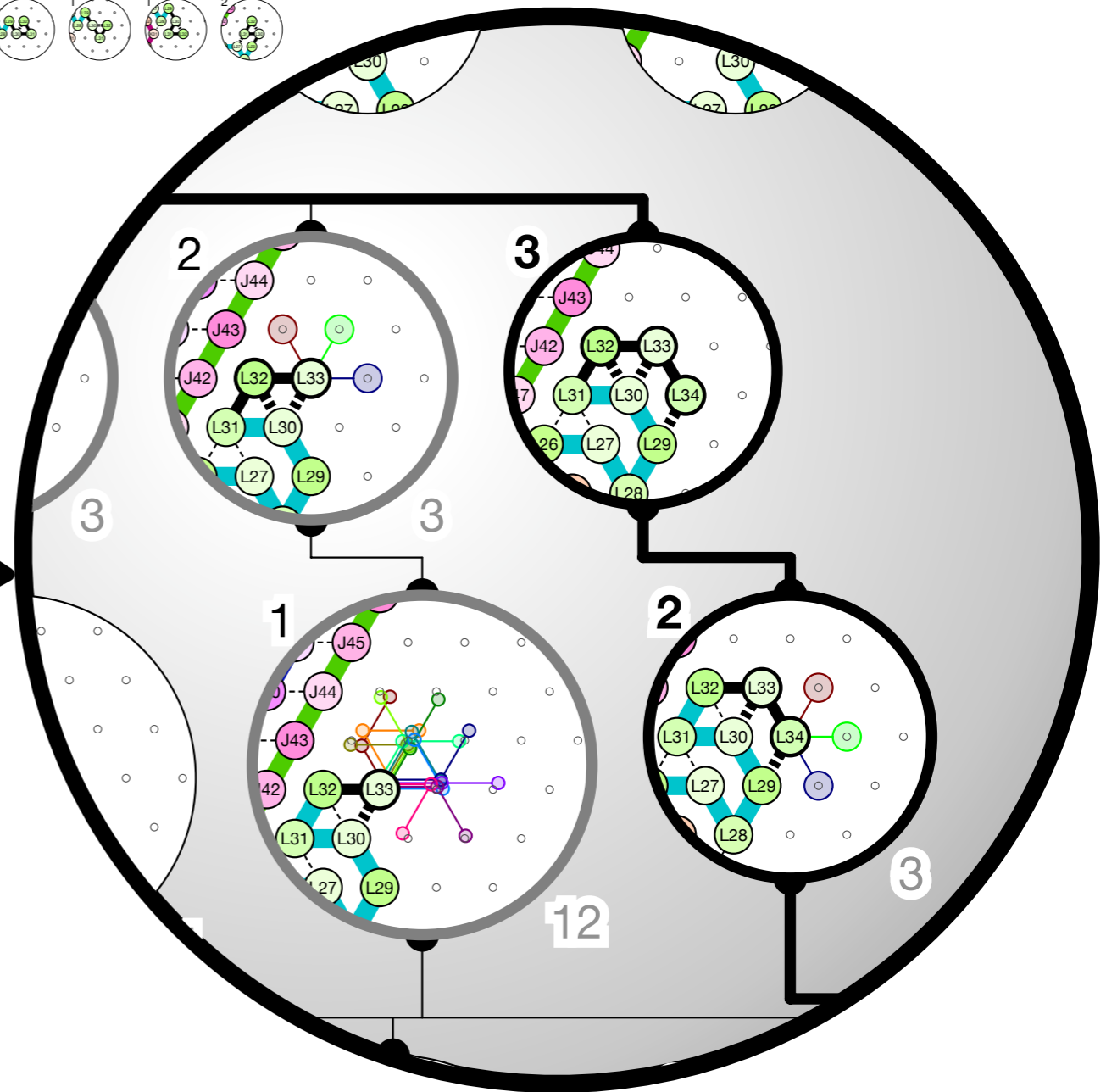
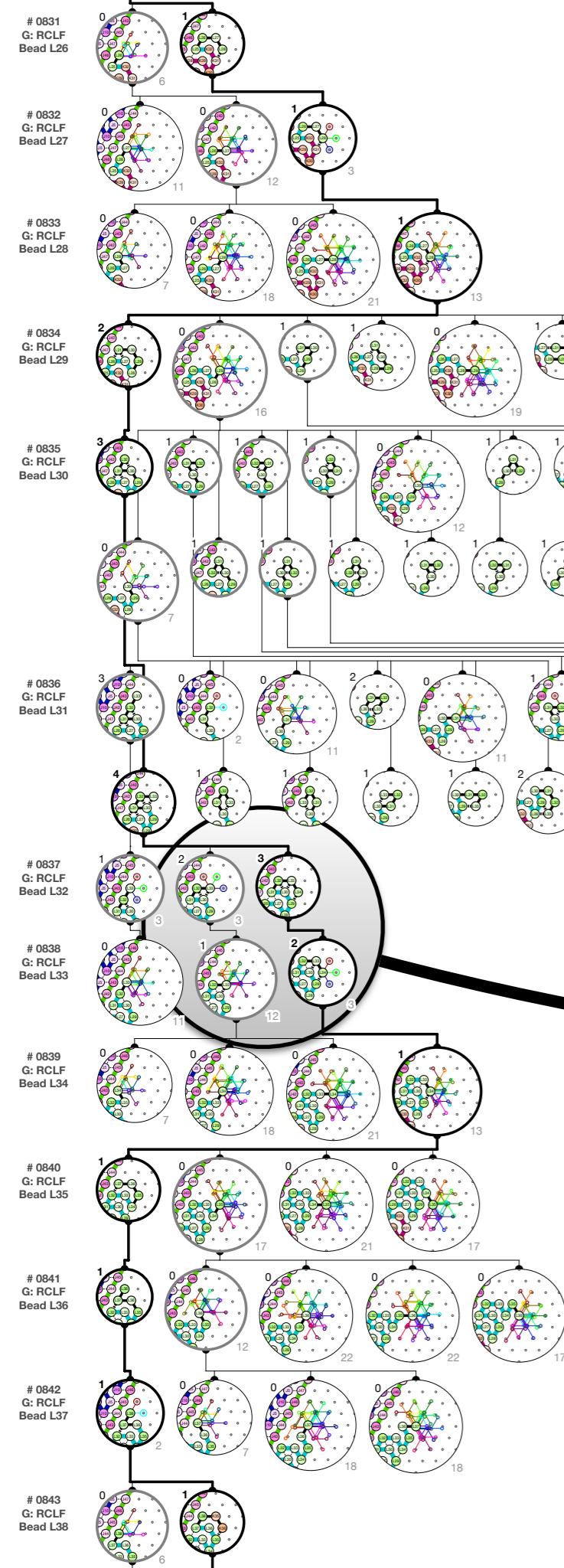




# Third, enumerate all the possible environments



# Finally, prove the folding of each brick in each environment



# *Thank you!*

## *Conclusions*

- Better understanding on how nature might work (Geometry, hidden functions, offsets,...)
- Computational paradigms discovered for biocomputing while folding

## *Perspectives*

- How to handle reconfigurations?
- Universal folding system? (i.e. programming language)



# *Thank you!*

## *Conclusions*

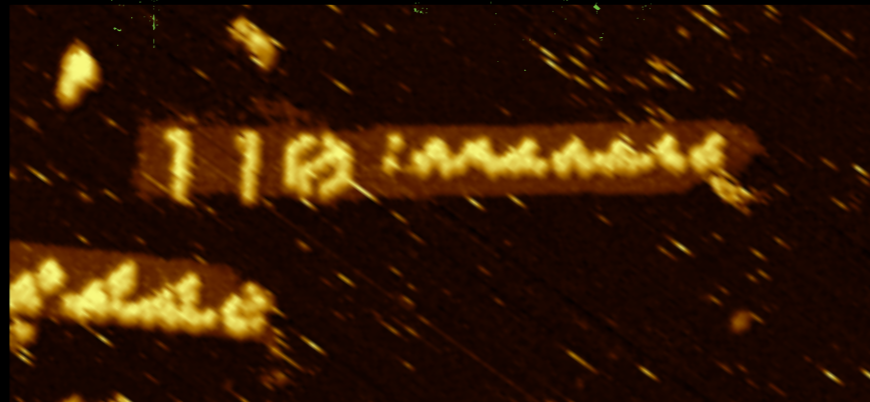
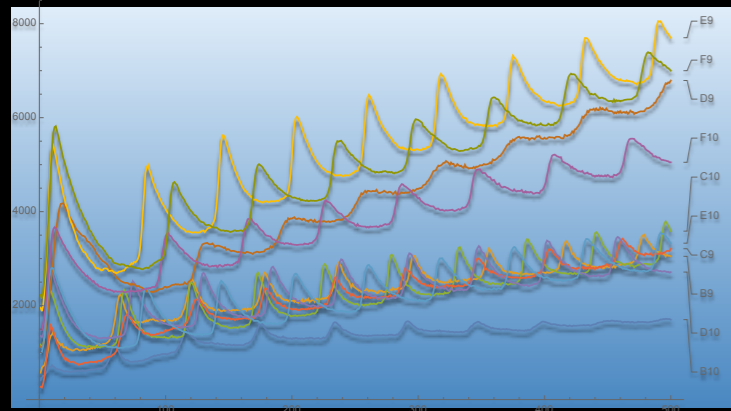
- Better understanding on how nature might work (Geometry, hidden functions, offsets,...)
- Computational paradigms discovered for biocomputing while folding

## *Perspectives*

- How to handle reconfigurations?
- Universal folding system? (i.e. programming language)

# 16-20 Jan. 2017: Research school BioMolecular Computing@ENS Lyon (France)!

## Theory & WET LAB!



Damien Woods, INRIA Paris

Cendrine Moskalenko, ENS Lyon

Ludovic Bellon, ENS Lyon



Yannick Rondelez, ESPCI

