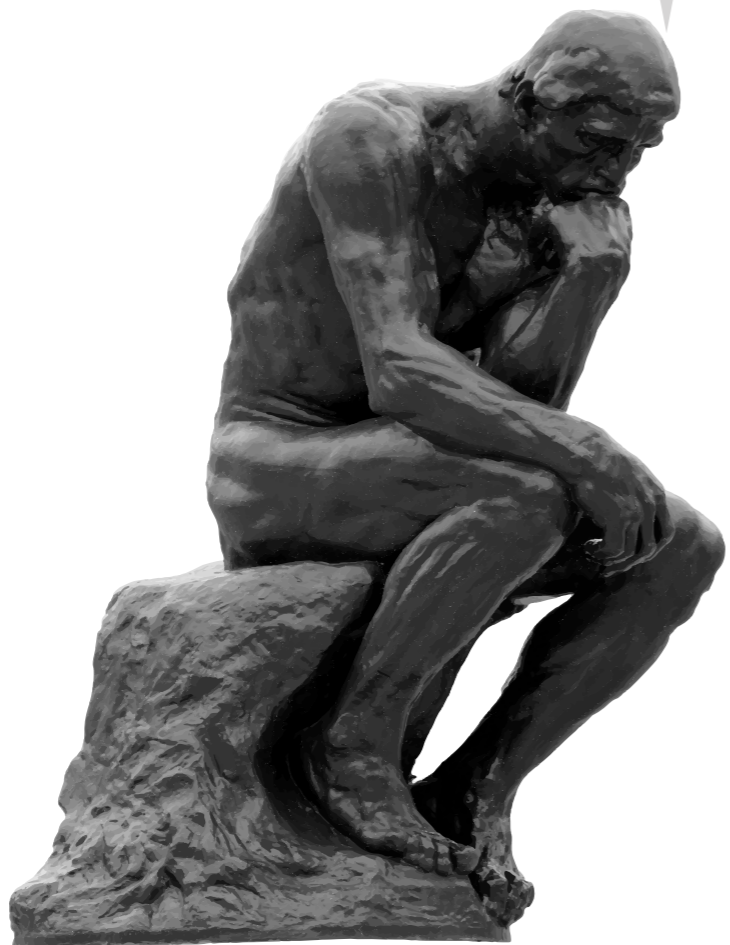
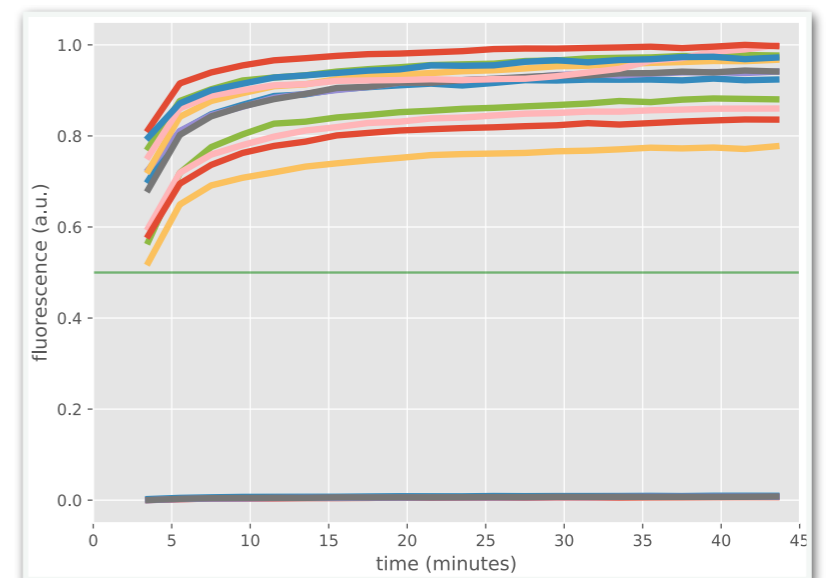
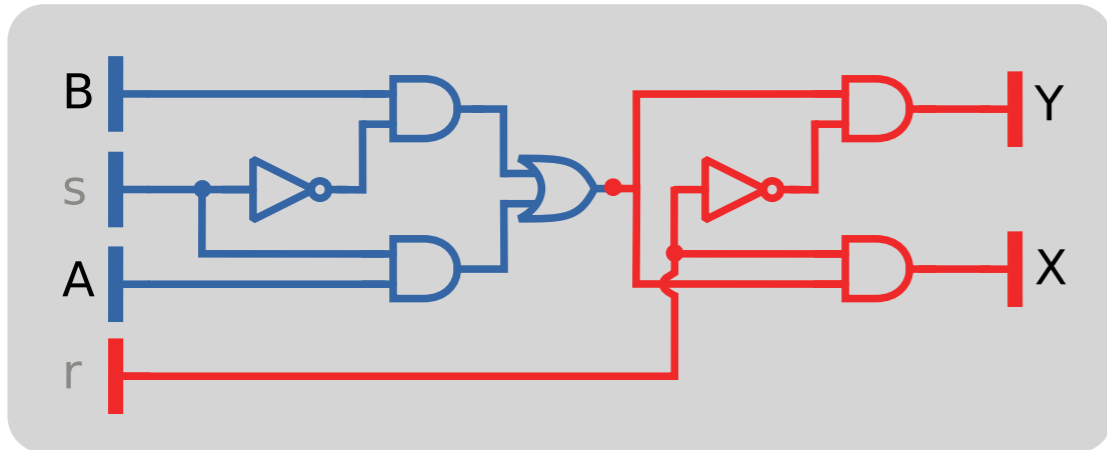


*Theory* & *Practice*  
of DNA strand displacement circuits

*October 8, 2018 @ DNA 24*

Chris Thachuk  
Winfree Lab, California Institute of Technology

# Today's tutorial in a nutshell



# Molecular Circuits

Built upon DNA strand displacement cascades

## Input

5'–ACCACGATCACATTAC–3'

5'–GCAACATACAT–3'

5'–CCCATACATCACCAG–3'

5'–TACCACATGAGCAGCA–3'

## Computation

DNA strand displacement cascades

## Output

5'–GAGCTACATCAC–3'

5'–TAAATCATGATCAG–3'

# Molecular Circuits

Built upon DNA strand displacement cascades

## Input

5'-ACCACGATCACATTAC-3'

5'-GCAACATACAT-3'

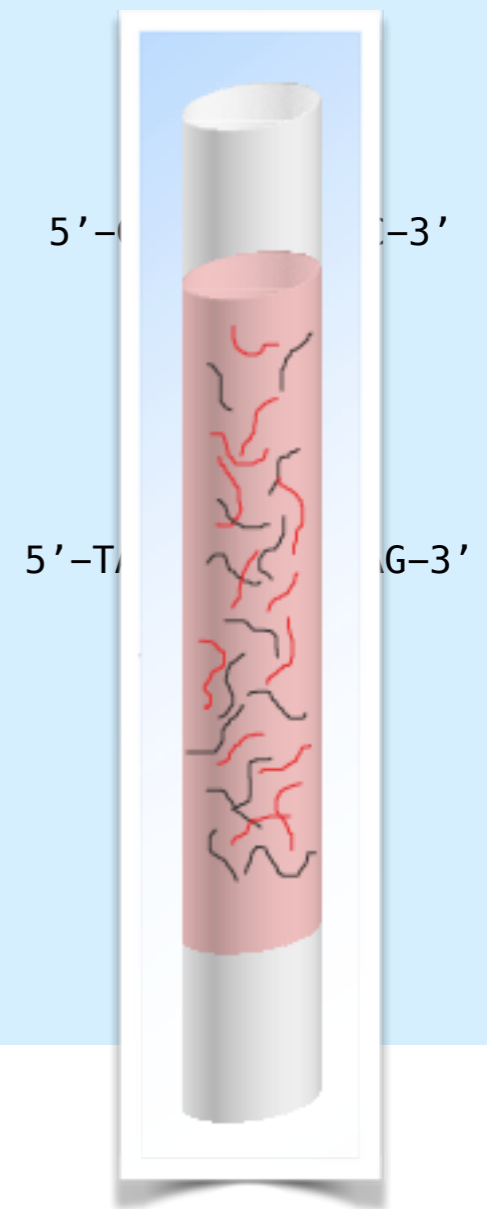
5'-CCCATACATCACCAG-3'

5'-TACCACATGAGCAGCA-3'

## Computation

DNA strand displacement cascades

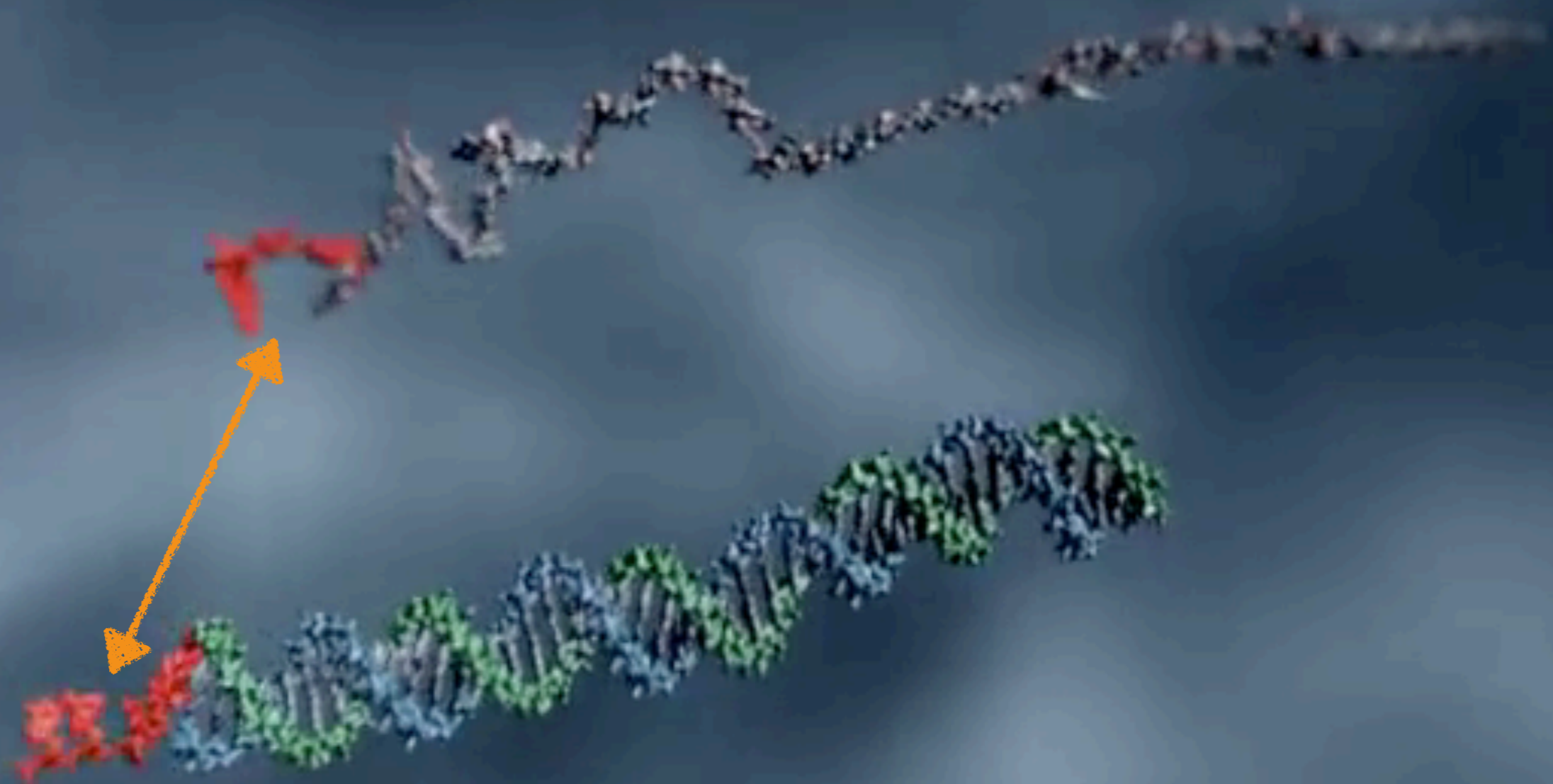
## Output



# Tutorial Outline

- ▶ **Review of strand displacement**
- ▶ Building and composing logic gates
- ▶ Tools for designing and verifying circuits
- ▶ Robustness of strand displacement

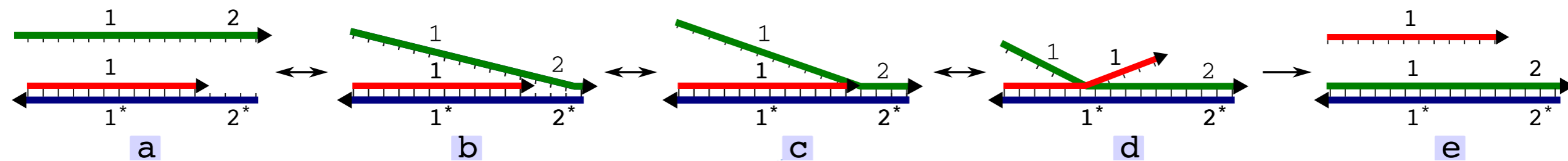
# Review of DNA Strand Displacement (DSD)



B. Yurke, A. J. Turberfield, A. P. Mills Jr., F. C. Simmel, J. L. Neumann, *Nature* 406, 605 (2000).

A. J. Turberfield et al., *Phys. Rev. Lett.* 90, 118102

Adapted from Zhang & Seelig 2011



# Strand Displacement Cascades

=

## Three Rules

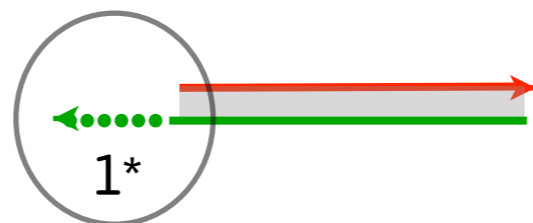
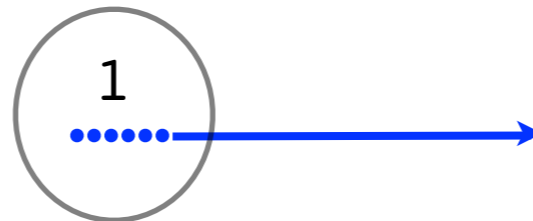




# Domain level rules for DSD

## Rule 1: *Bind*

### Example



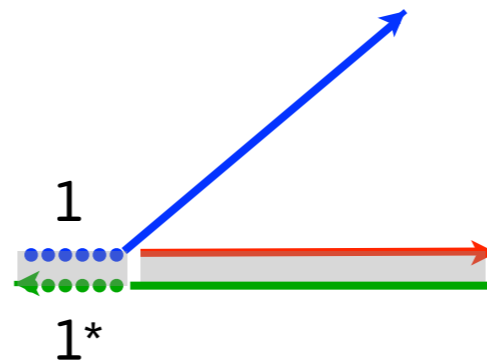
single-stranded  
complementary  
domains

# Domain level rules for DSD

## Rule 1: *Bind*

Two single-stranded complementary domains can bind

Example

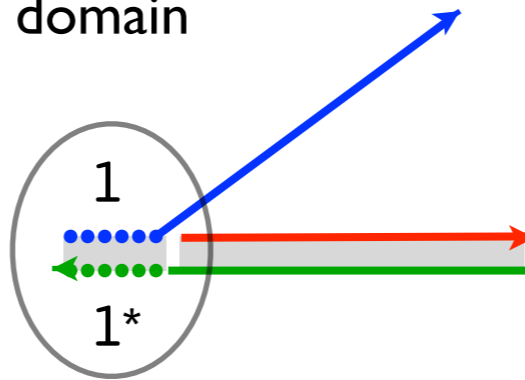


# Domain level rules for DSD

## Rule 2: *Release*

### Example

blue strand bound by only  
a short domain



# Domain level rules for DSD

## Rule 2: *Release*

Any strand bound by only a short domain can **release**

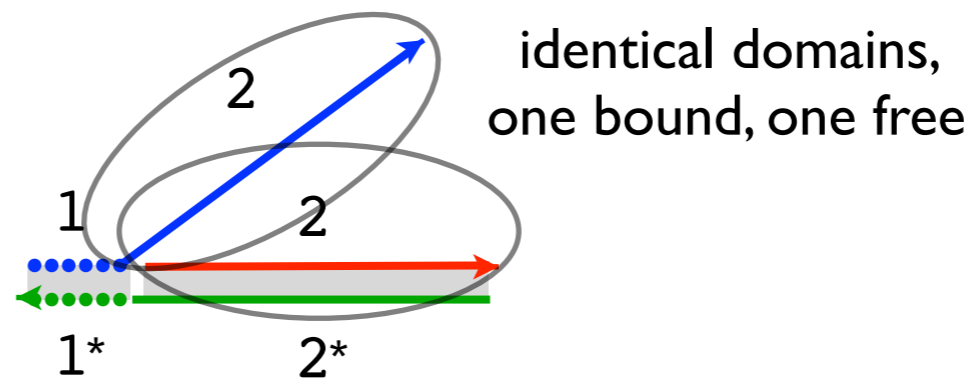
### Example



# Domain level rules for DSD

## Rule 3: *Displace*

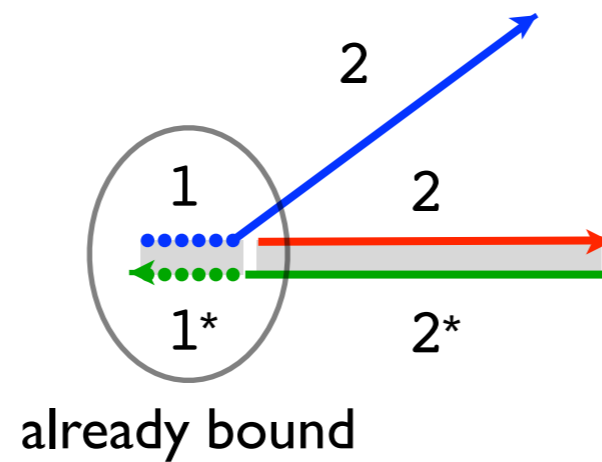
### Example



# Domain level rules for DSD

## Rule 3: *Displace*

### Example

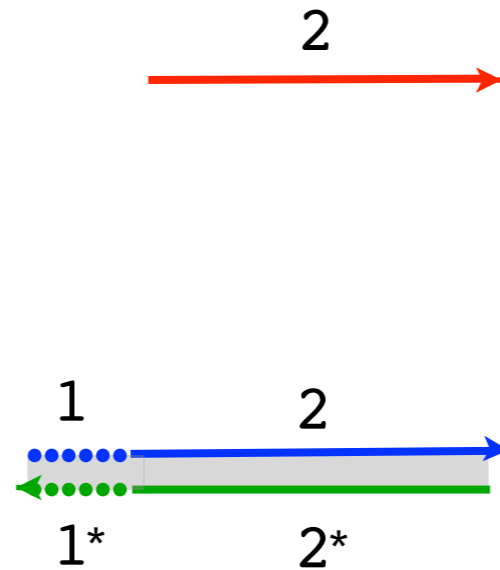


# Domain level rules for DSD

## Rule 3: *Displace*

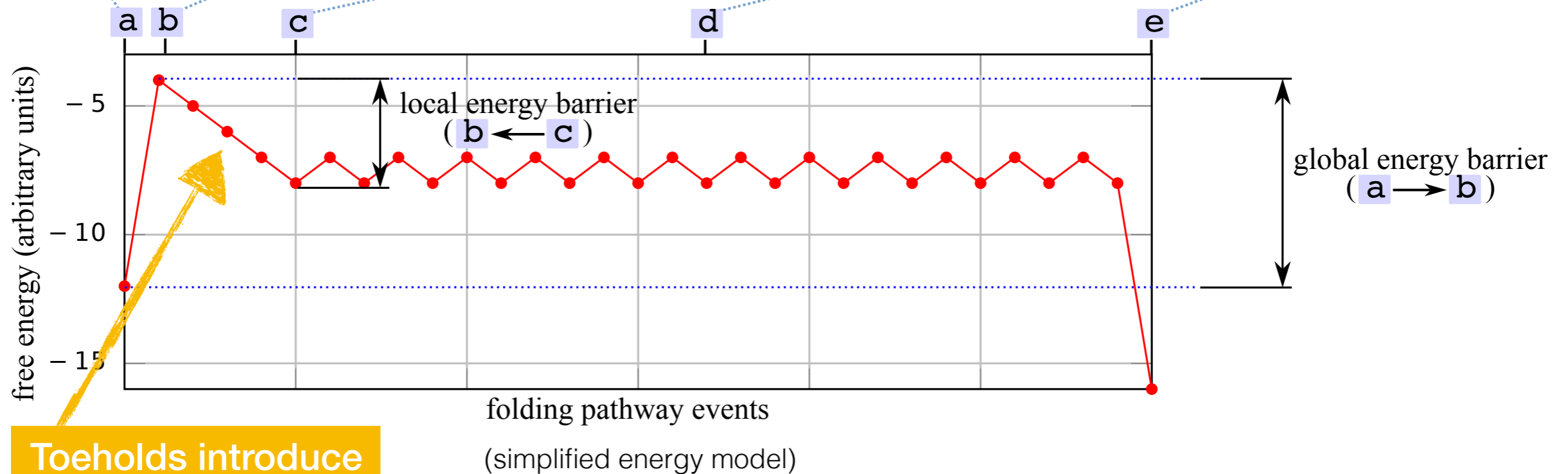
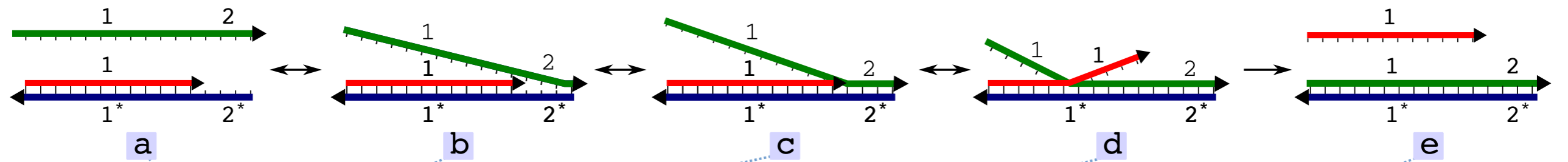
A domain can **displace** an identical domain of another strand, if neighboring domains are already bound

### Example



# Why do we use toeholds?

Adapted from Zhang & Seelig 2011



**Toeholds introduce energy barriers**

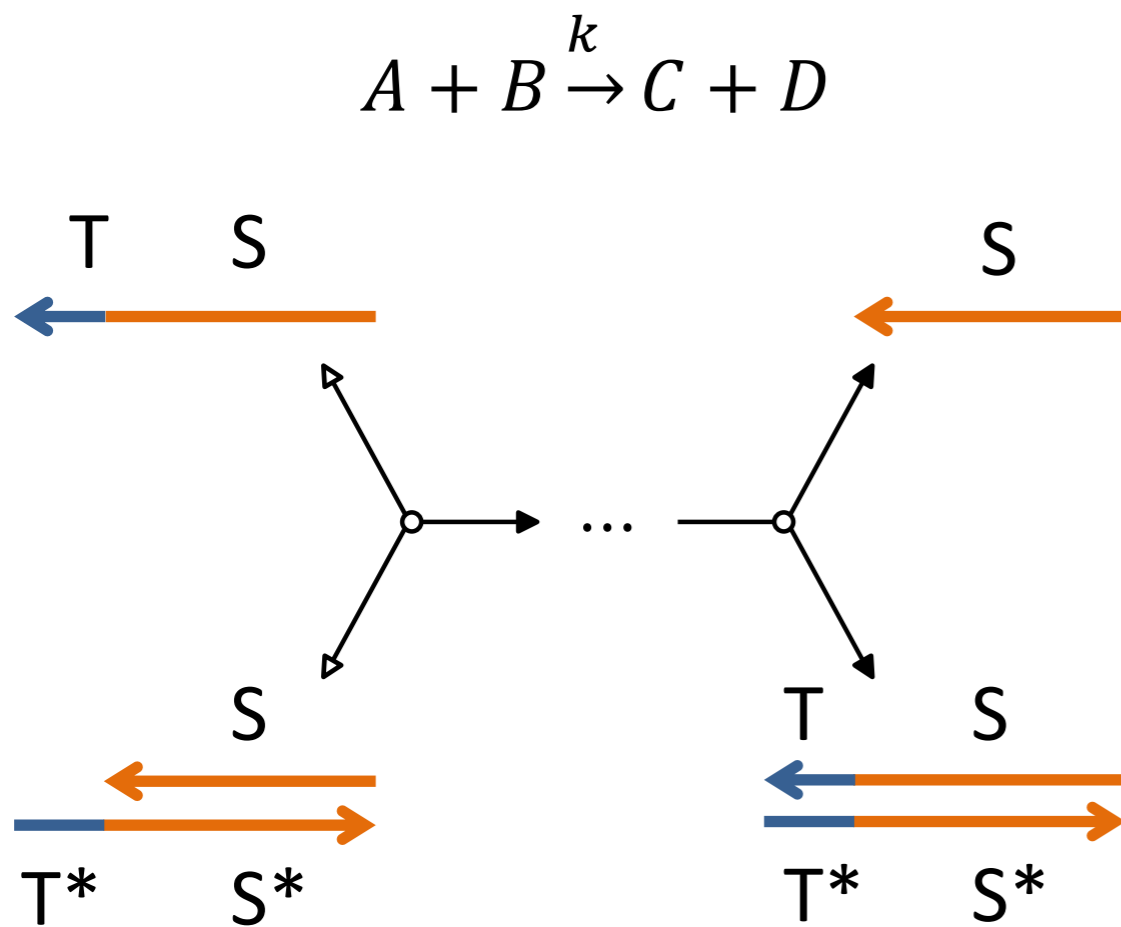
folding pathway events  
(simplified energy model)



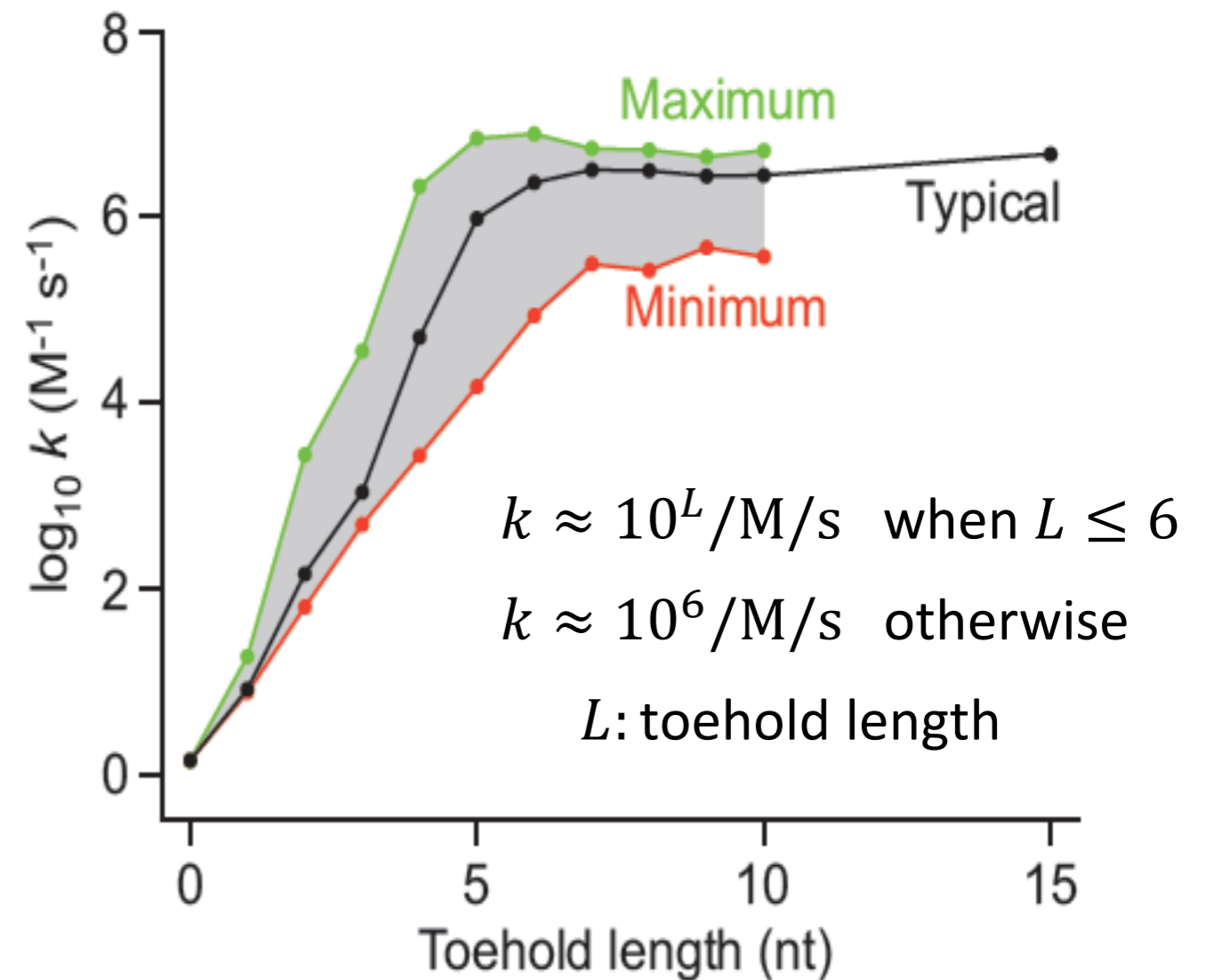
# Toehold-mediated DNA strand displacement

T: toehold domain (typically 3-7 nucleotides)

S: branch migration domain (typically 15-20 nucleotides)

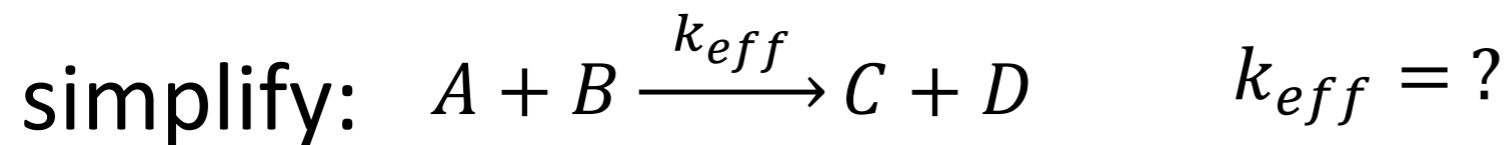
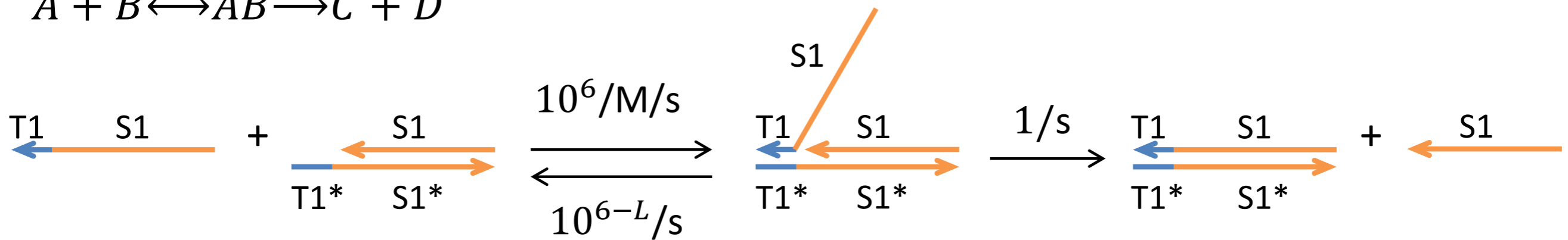


The rate of strand displacement grows exponentially with toehold length for short toeholds.



Zhang and Seelig, *Nature Chemistry* 2011

# Kinetics of toehold-mediated strand displacement



collision rate:  $10^6[A][B]$

collision success probability:  $\frac{1/s}{1/s + 10^{6-L}/s}$

net rate of success:  $10^6 \cdot \frac{1}{1 + 10^{6-L}} [A][B]$

$k_{eff}$

$k_{eff} \approx 10^L/M/s$  when  $L \leq 6$   
otherwise  $k_{eff} \approx 10^6/M/s$

$L$ : toehold length  $|T1|$

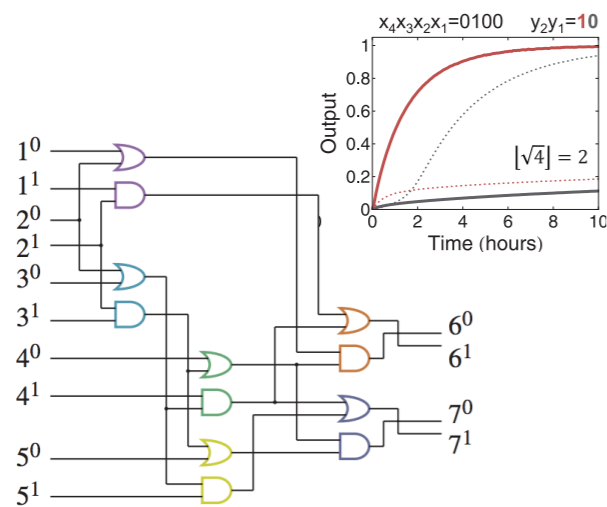
Zhang et al, *JACS* 2009

Srinivas et al, *NAR* 2013

This approximation is valid for low concentrations of A and B (e.g.  $[A]=[B]=100\text{nM}$ ) such that the unimolecular reaction is sufficiently faster than the bimolecular reaction.

# Strand displacement in the lab

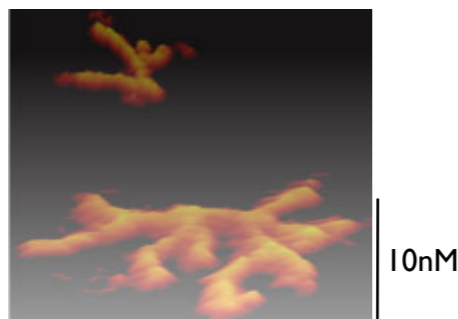
## molecular logic circuits



- Large autonomous biochemical networks built from scratch

Qian, Winfree, *Science* 2011

## controlling assembly of nanoscale structures

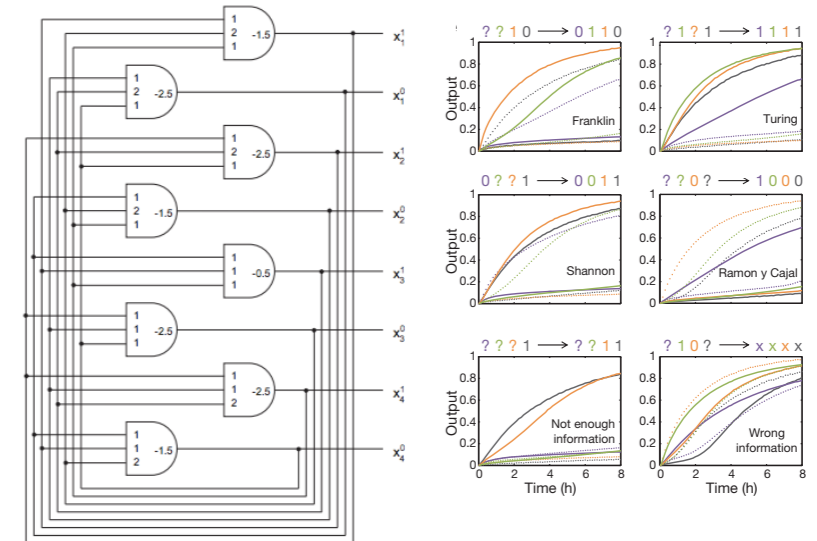


- Prescribed nanoscale structures seen under atomic force microscope

Yin, Choi, Calvert, Yurke, Pierce *Nature* 2008



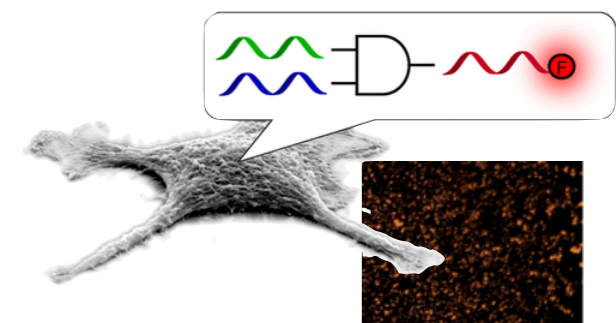
## molecular artificial neural networks



- Biochemical system doing inference

Qian, Winfree, Bruck *Nature* 2011

## strand displacement in vivo



- Logic on biological signals

Hemphill, Deiters *J Am Chem Soc* 2013

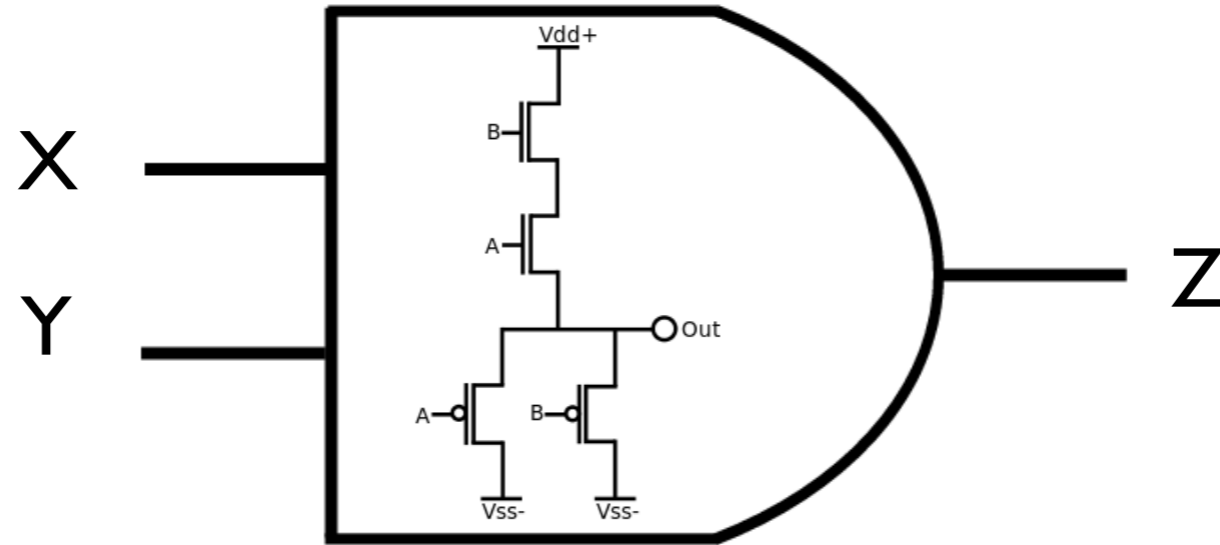
# Tutorial Outline

- ▶ Review of strand displacement
- ▶ **Building and composing logic gates**
- ▶ Tools for designing and verifying circuits
- ▶ Robustness of strand displacement

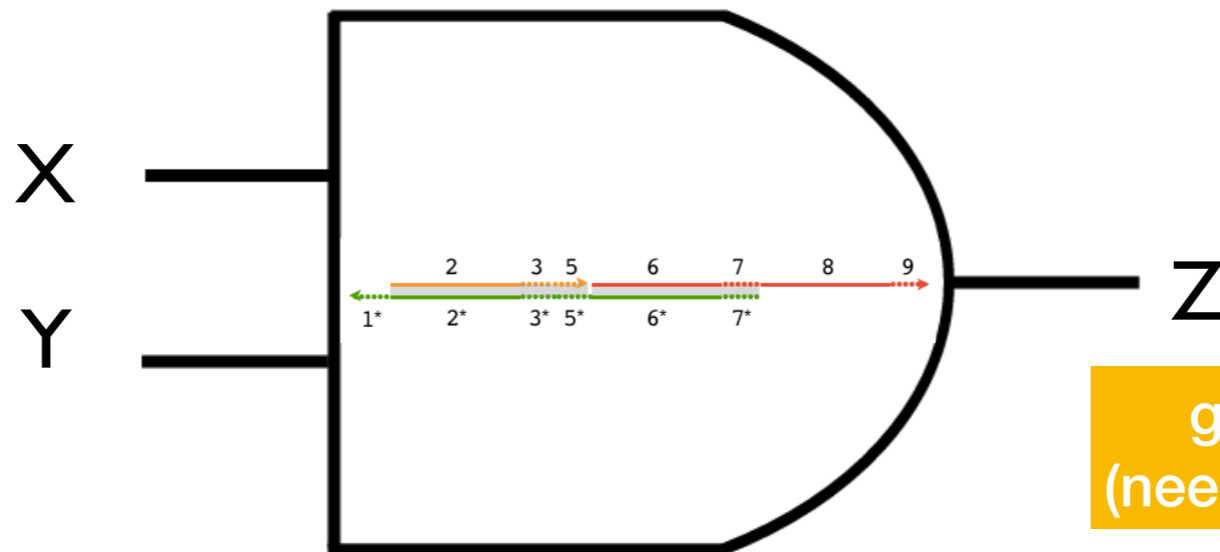
# AND gate

release Z if and only if X and Y are present

voltages



strands

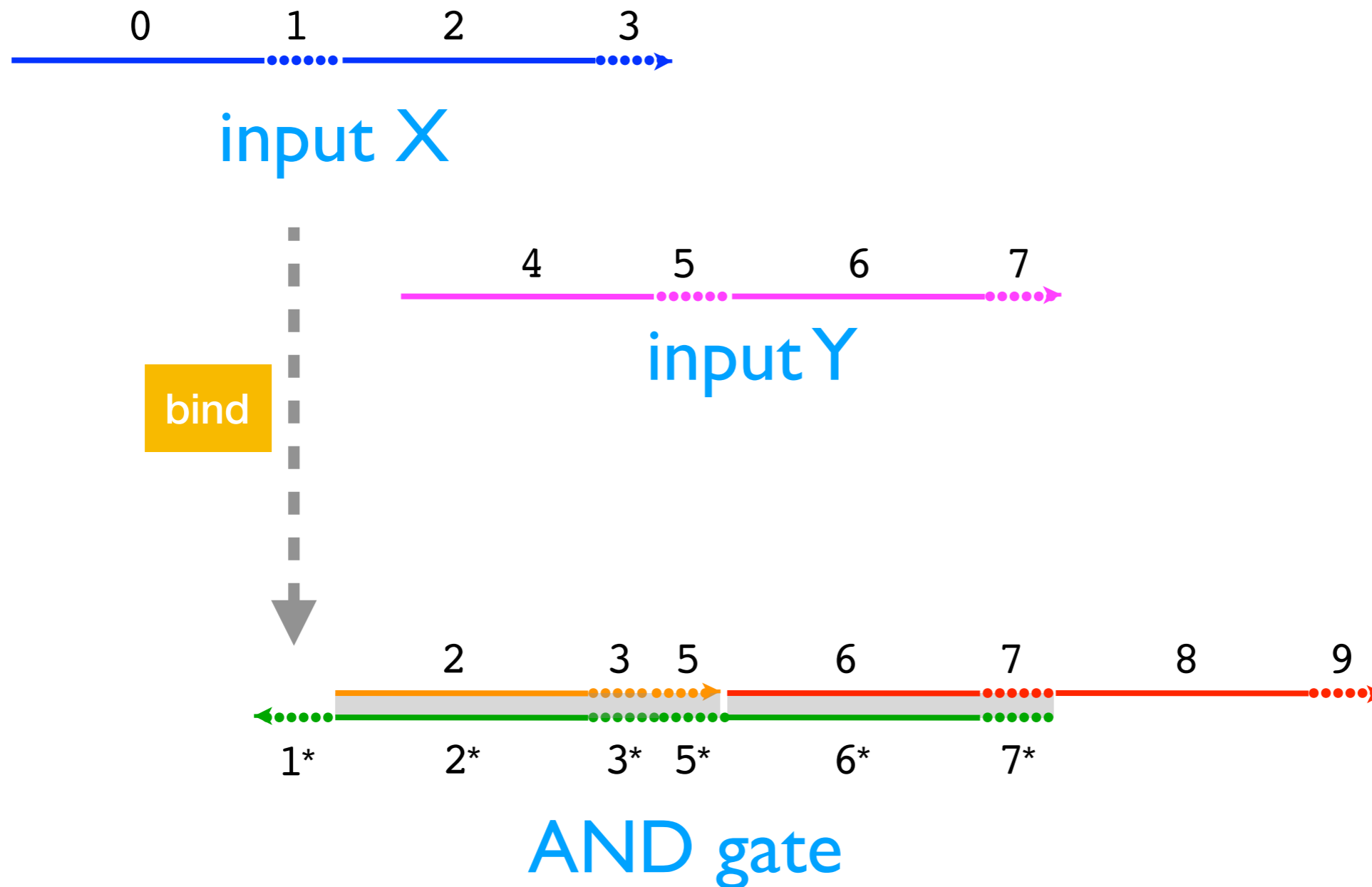


gates get consumed!  
(need to have many copies)

gate=complex

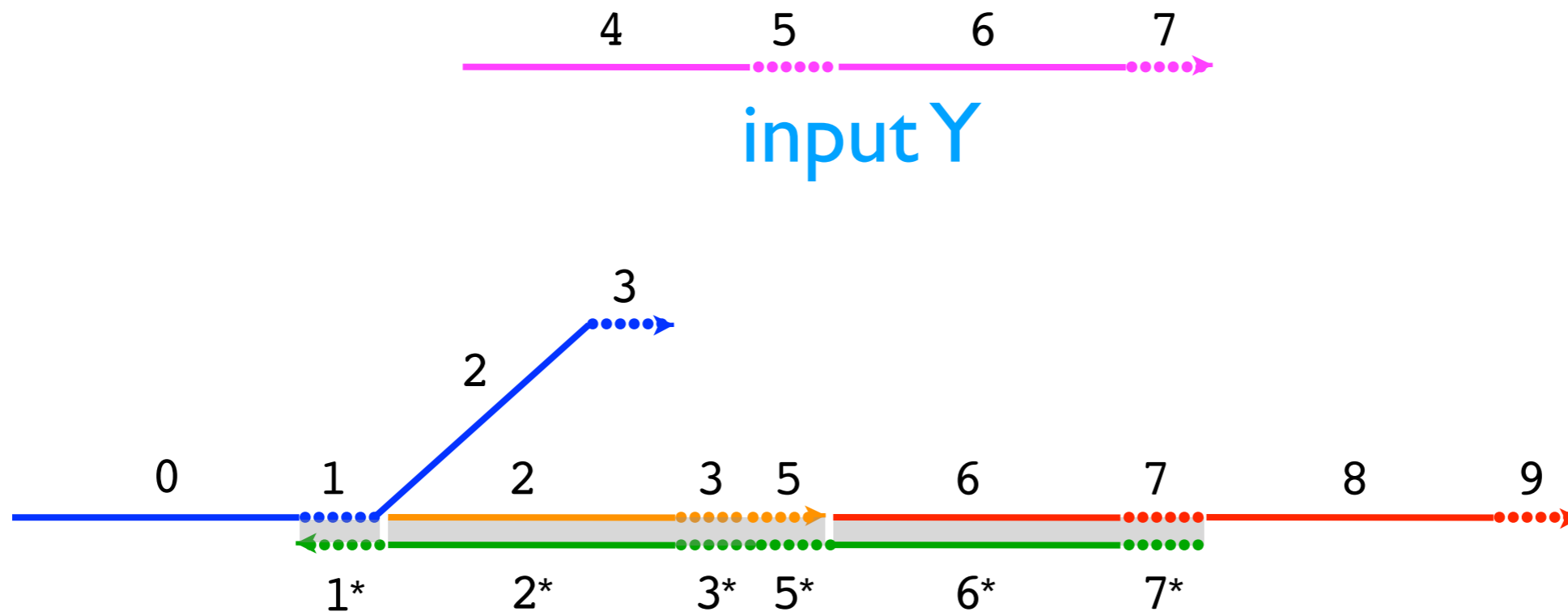
# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



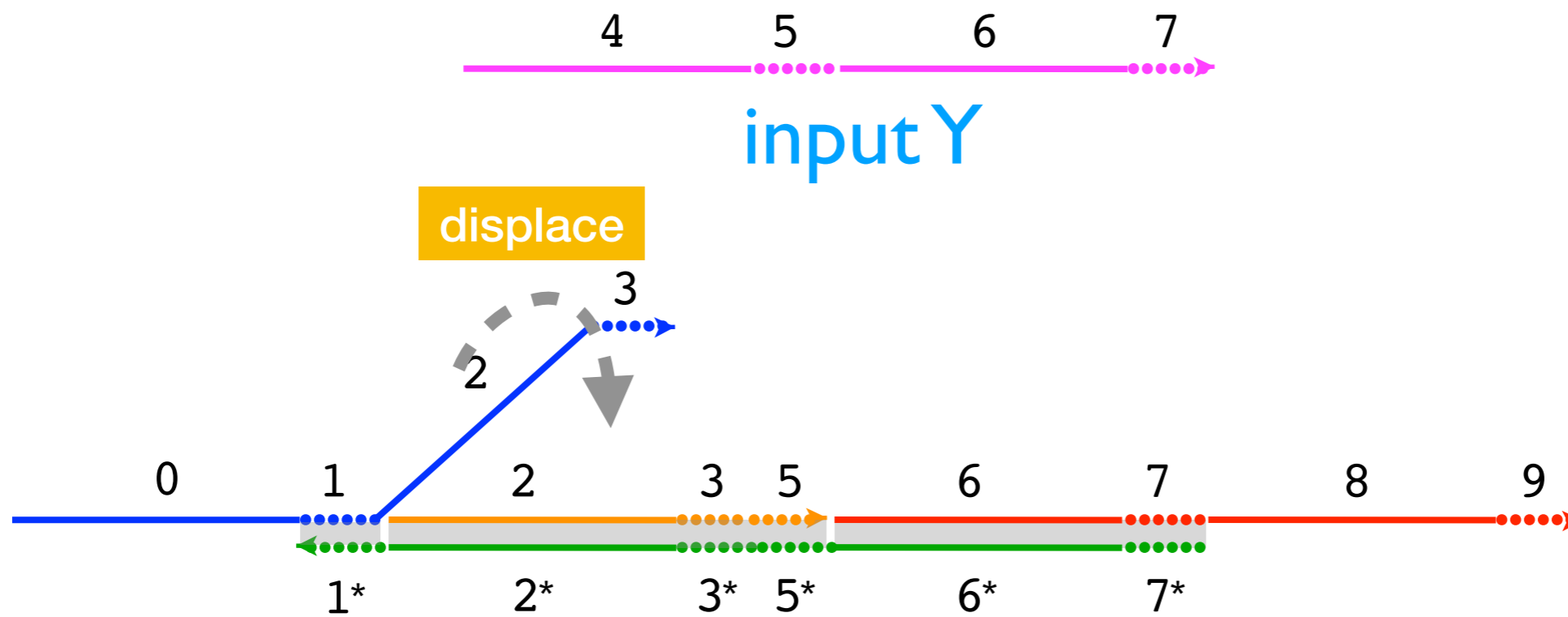
# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



# Strand Displacement Cascades Example: AND gate

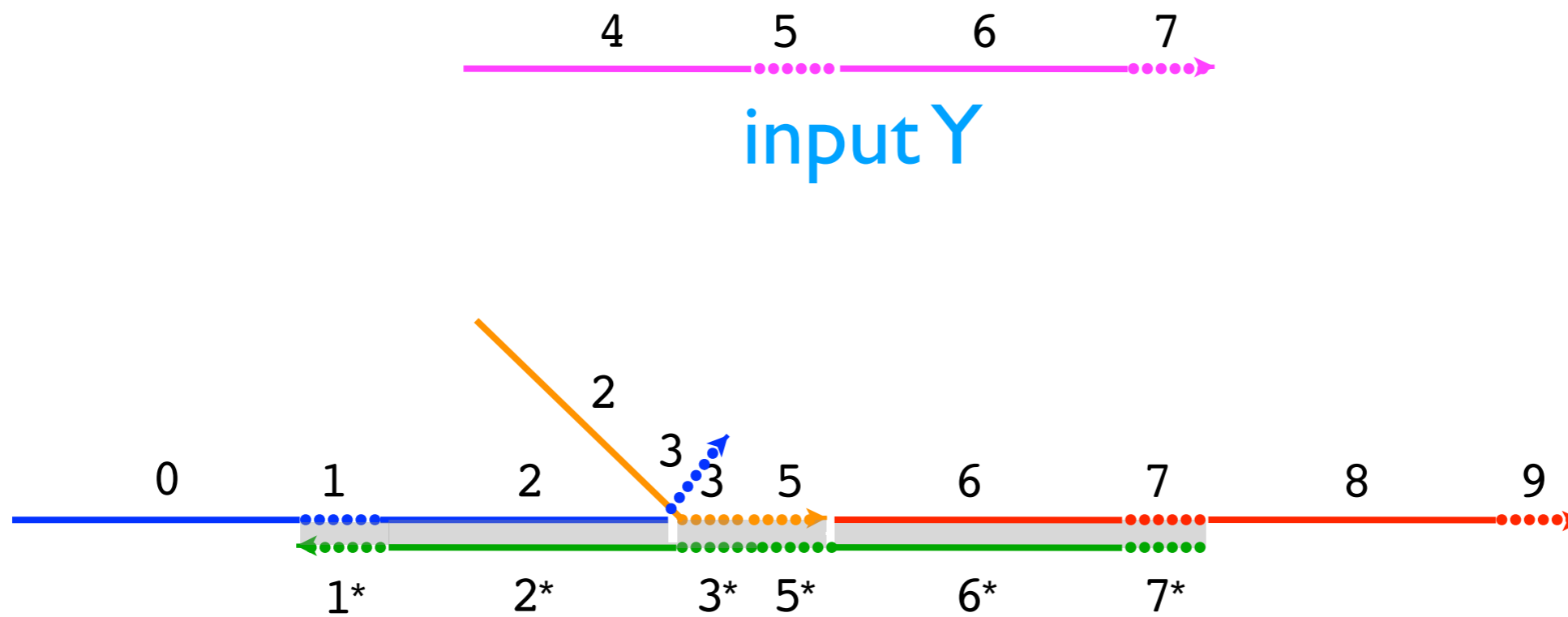
release Z if and only if X and Y are present





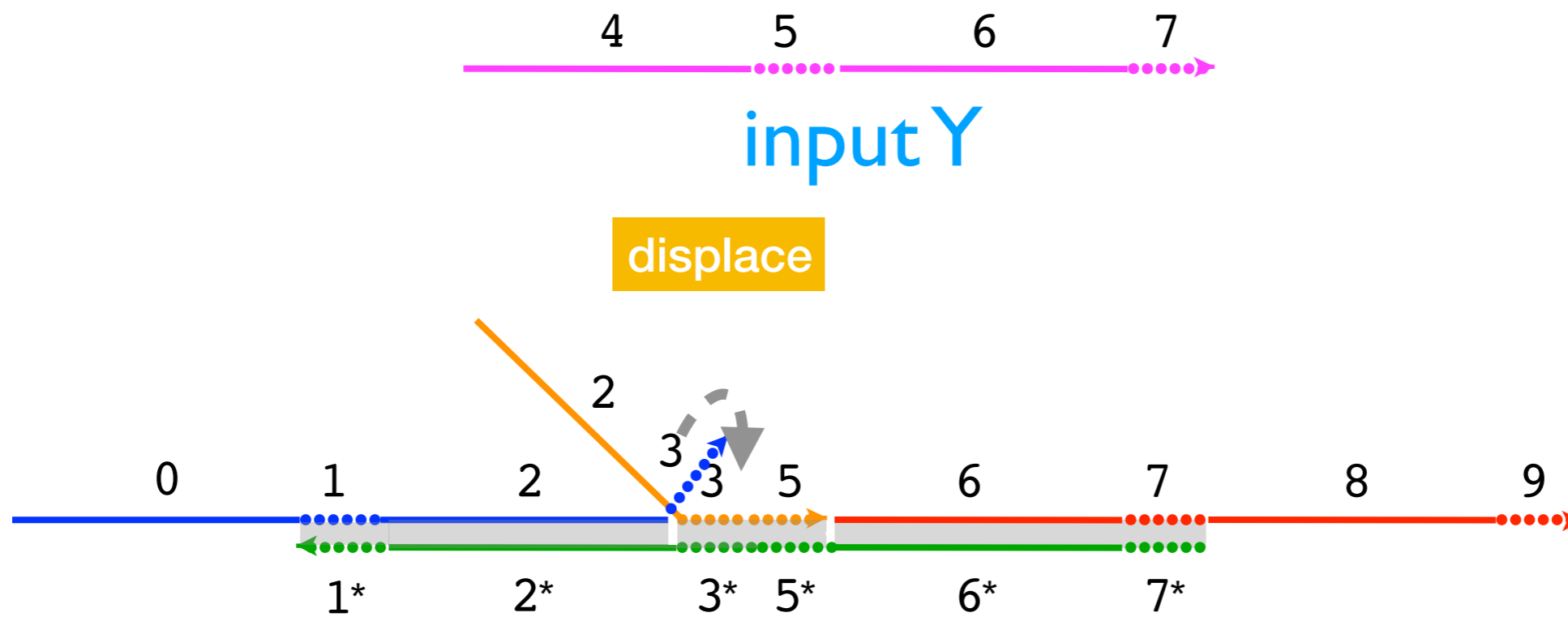
# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



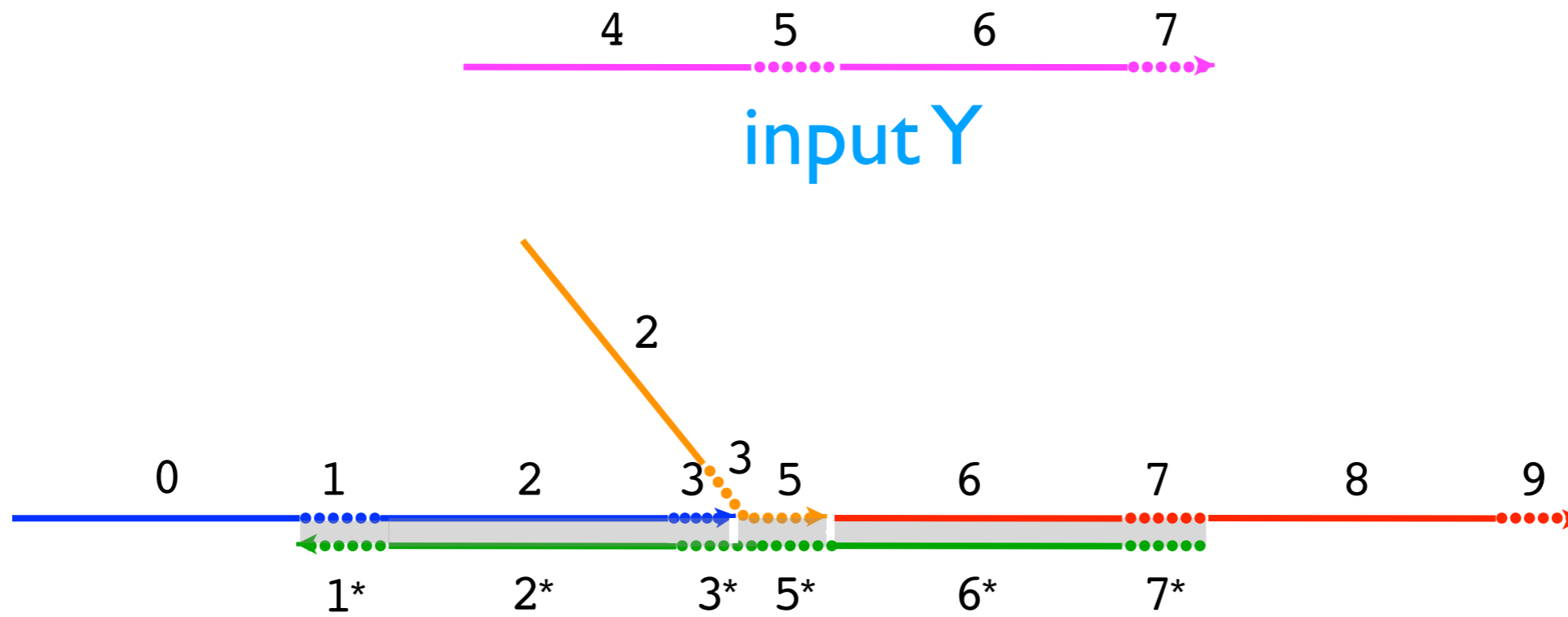
# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



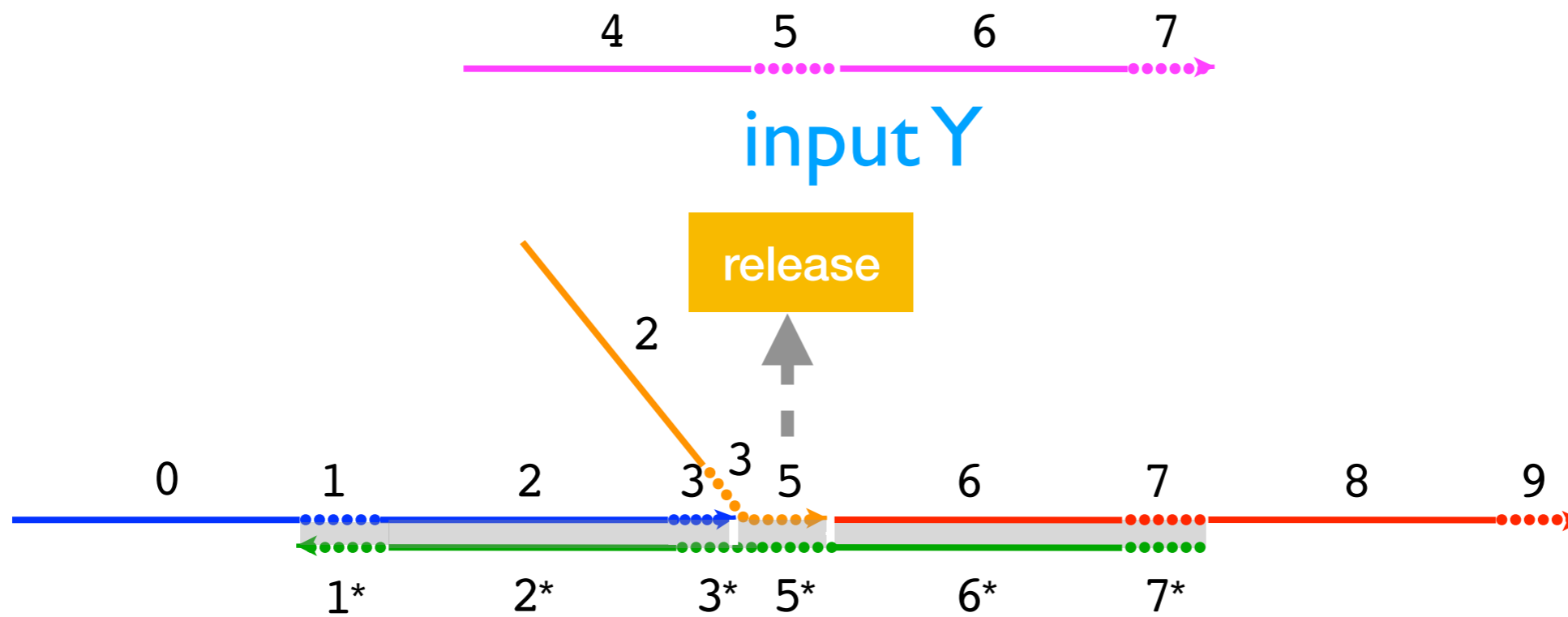
# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



# Strand Displacement Cascades Example: AND gate

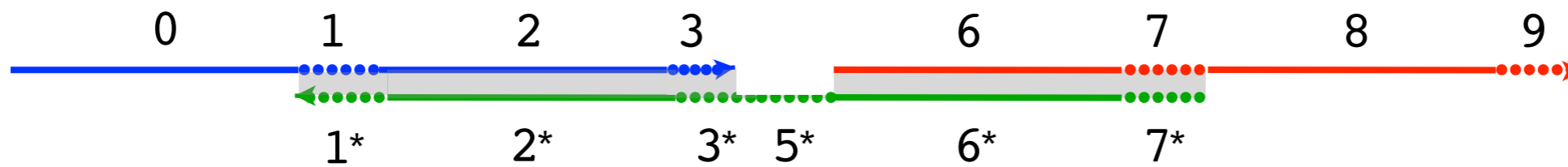
release Z if and only if X and Y are present



waste

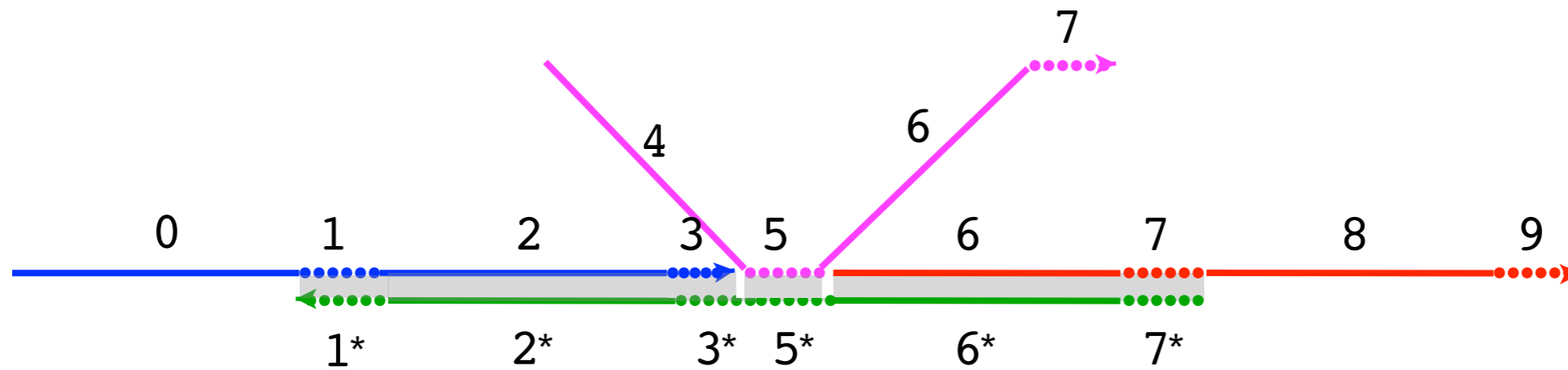


input Y



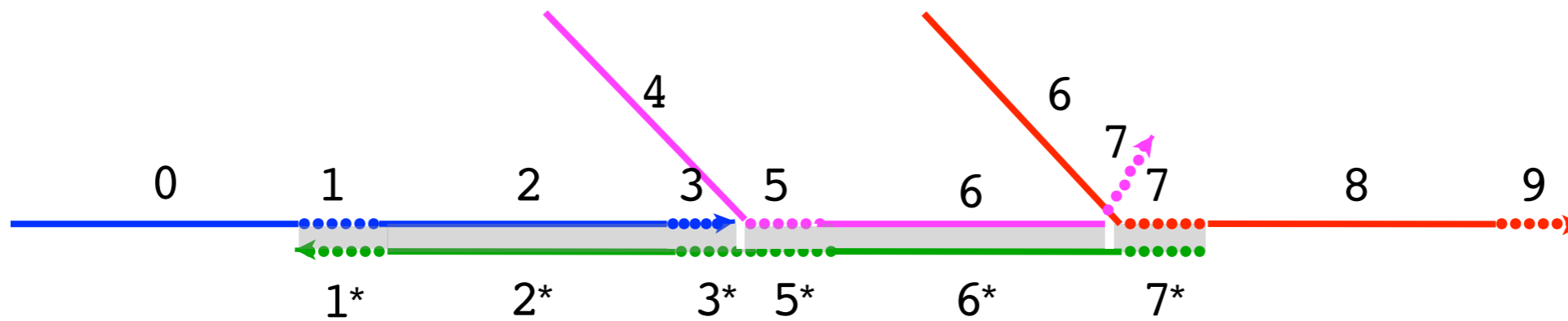
# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



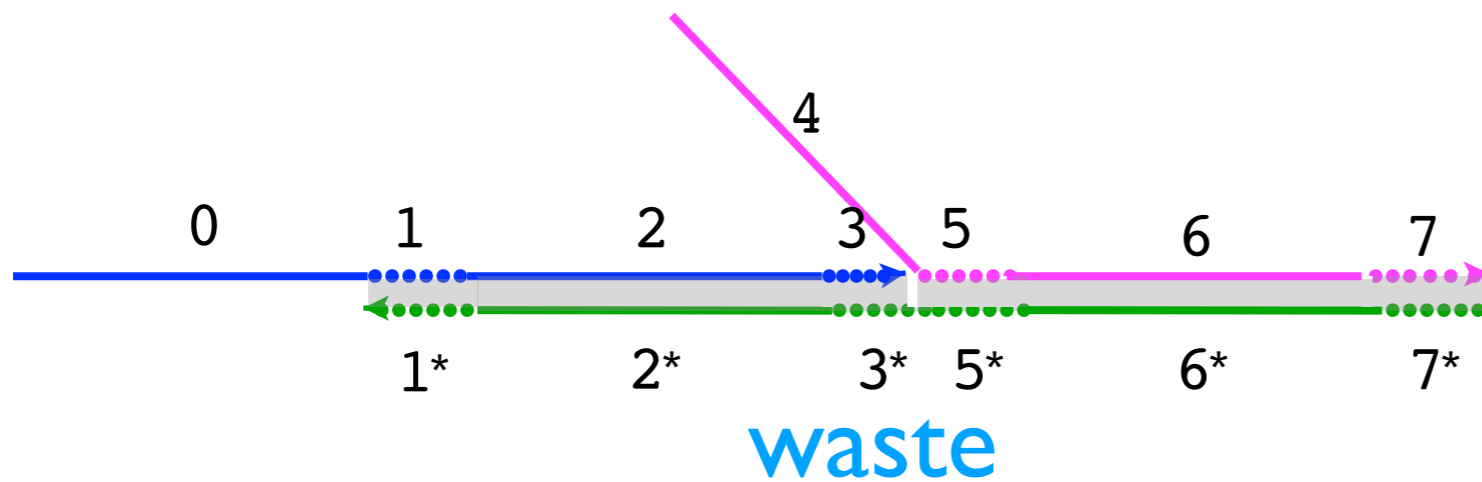
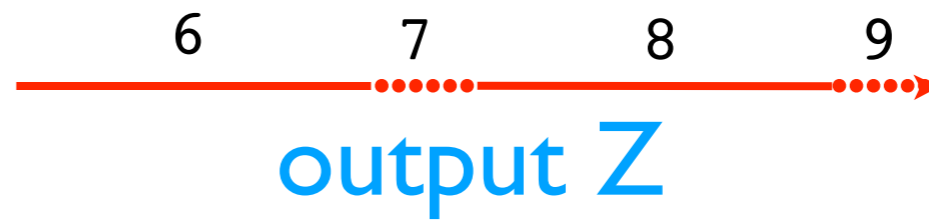
# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



# Strand Displacement Cascades Example: AND gate

release Z if and only if X and Y are present



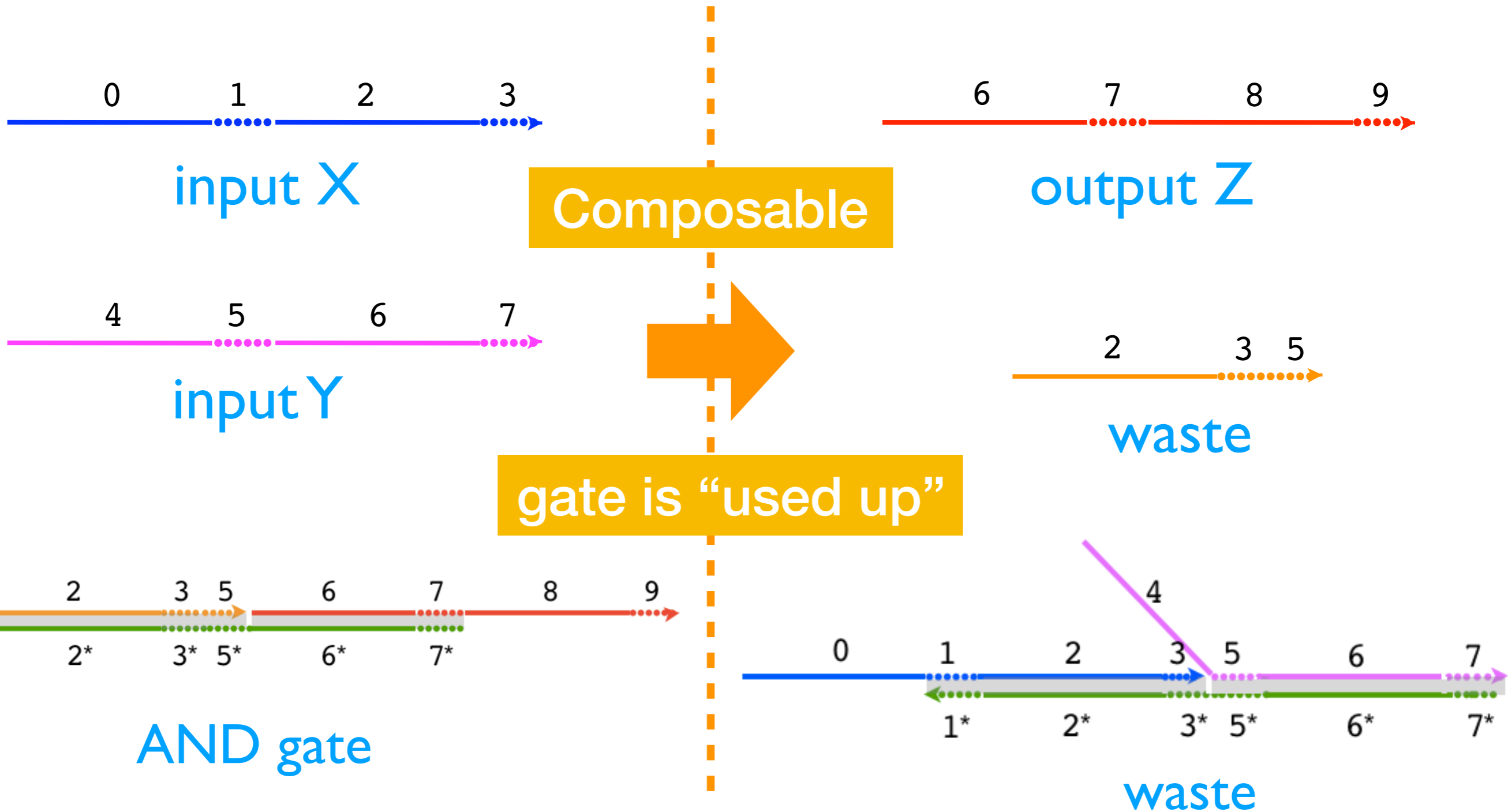


# Strand Displacement Cascades Example: AND gate

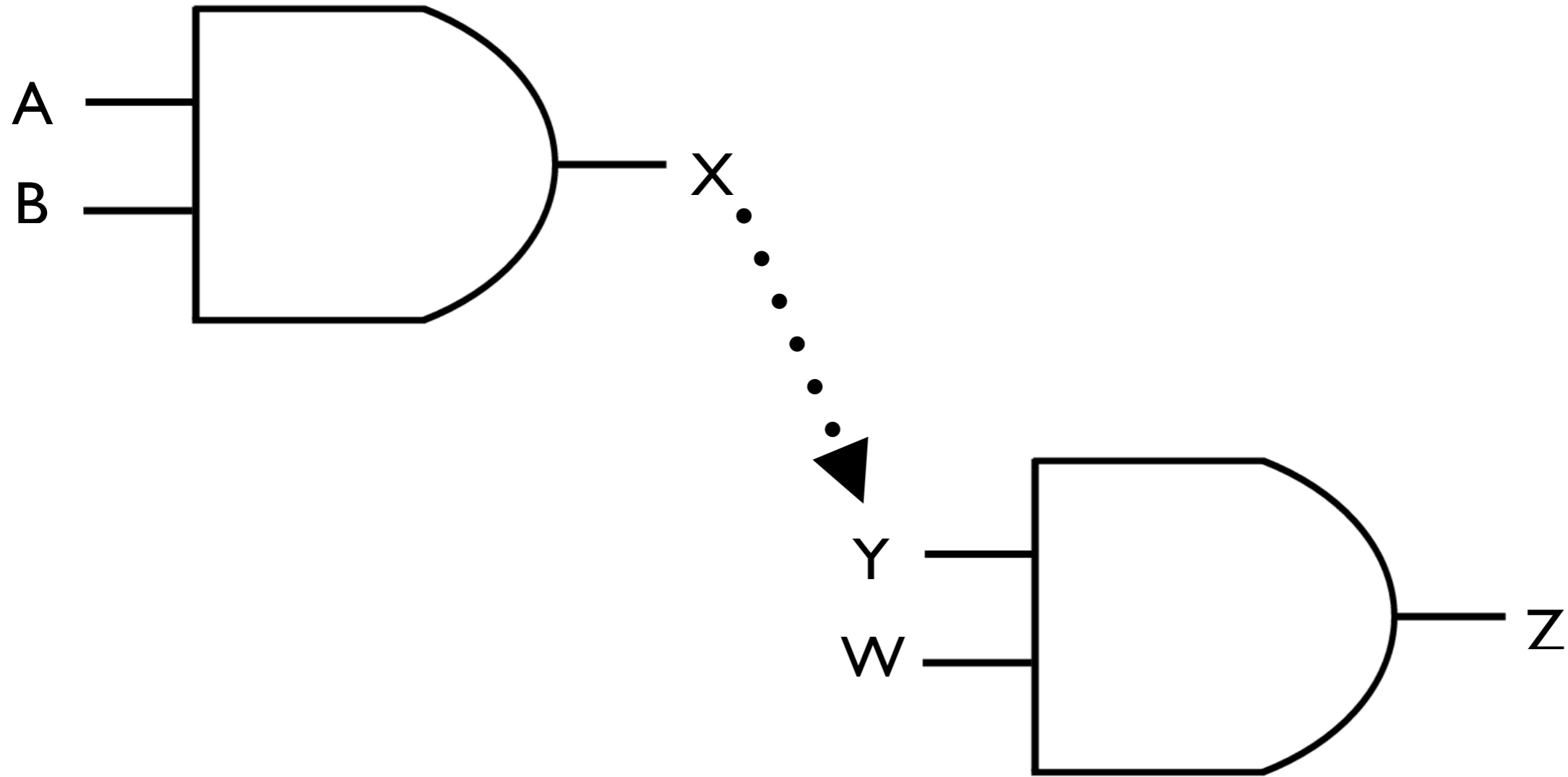
release Z if and only if X and Y are present

before

after



# Composing AND gates



We need a “wire”

# Sequence Independence

Translator (a “wire”):  $X \rightarrow Y$

input  $X$



output  $Y$

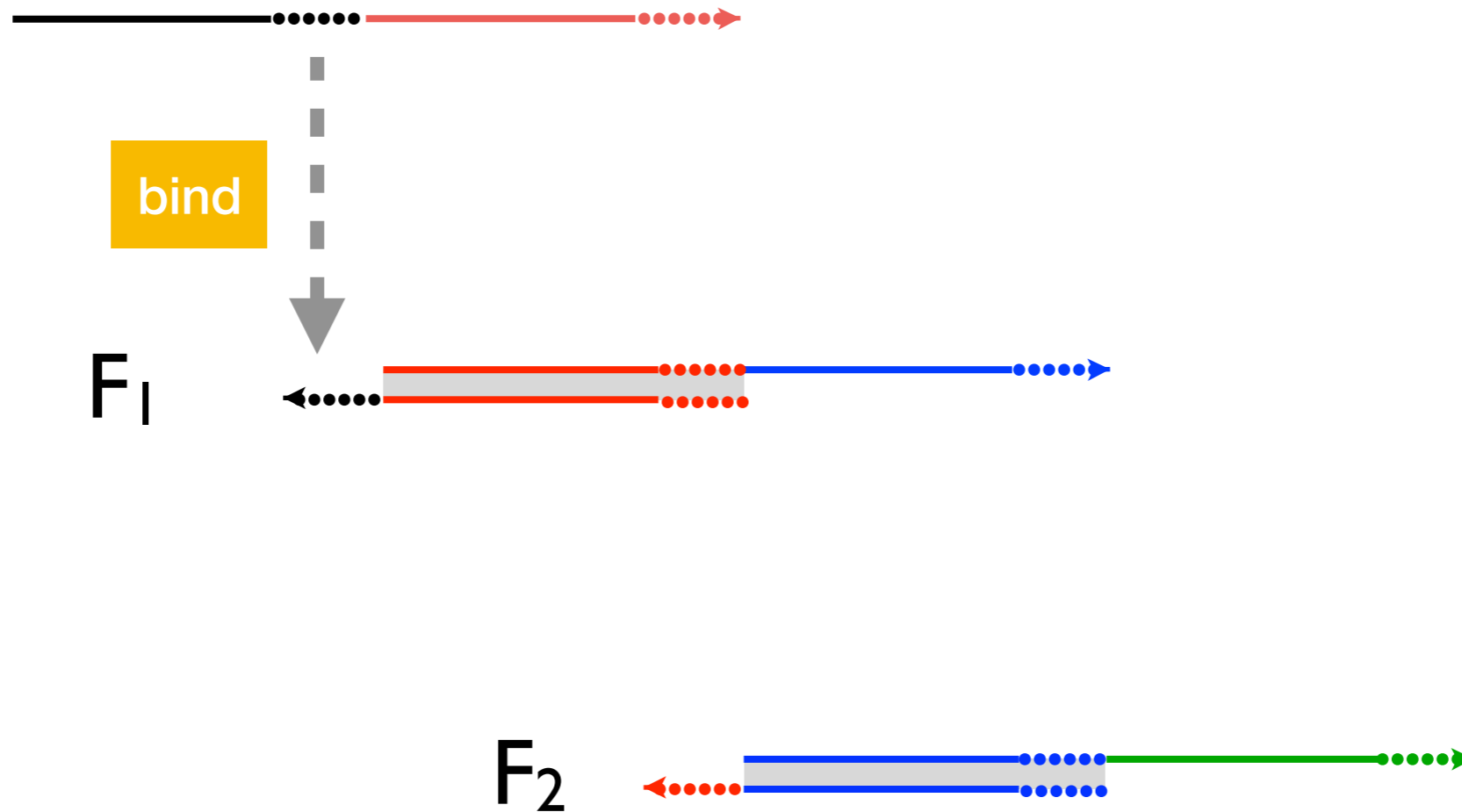


Different coloring scheme to emphasize sequence (in)dependence!

# Sequence Independence

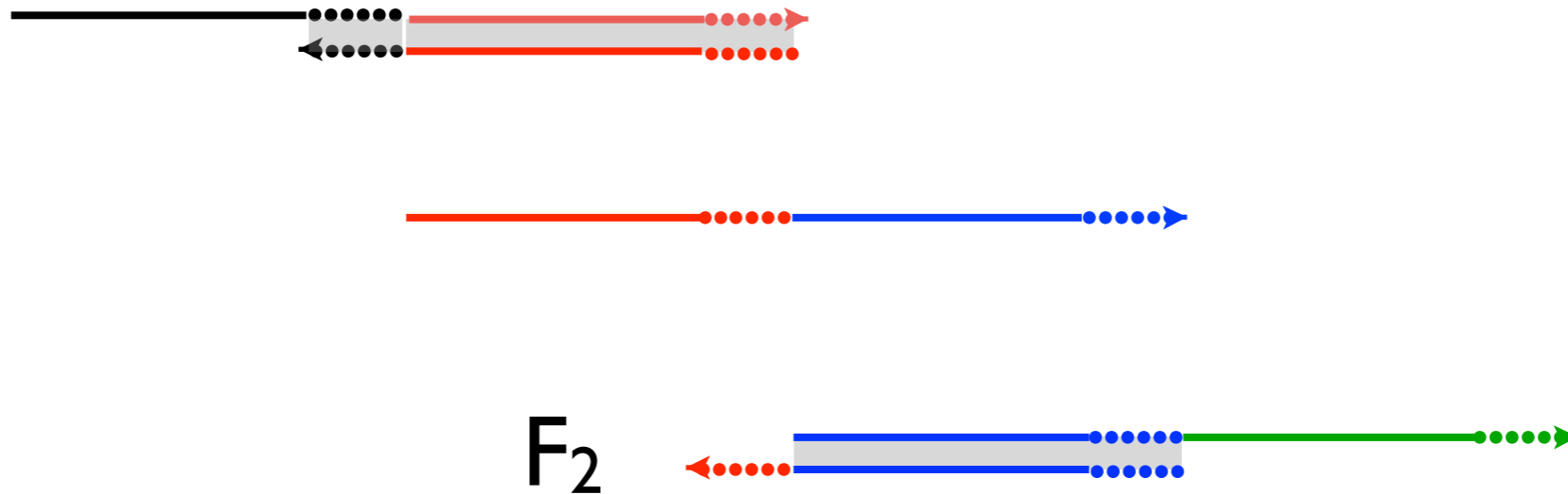
Translator (a “wire”):  $X \rightarrow Y$

input  $X$



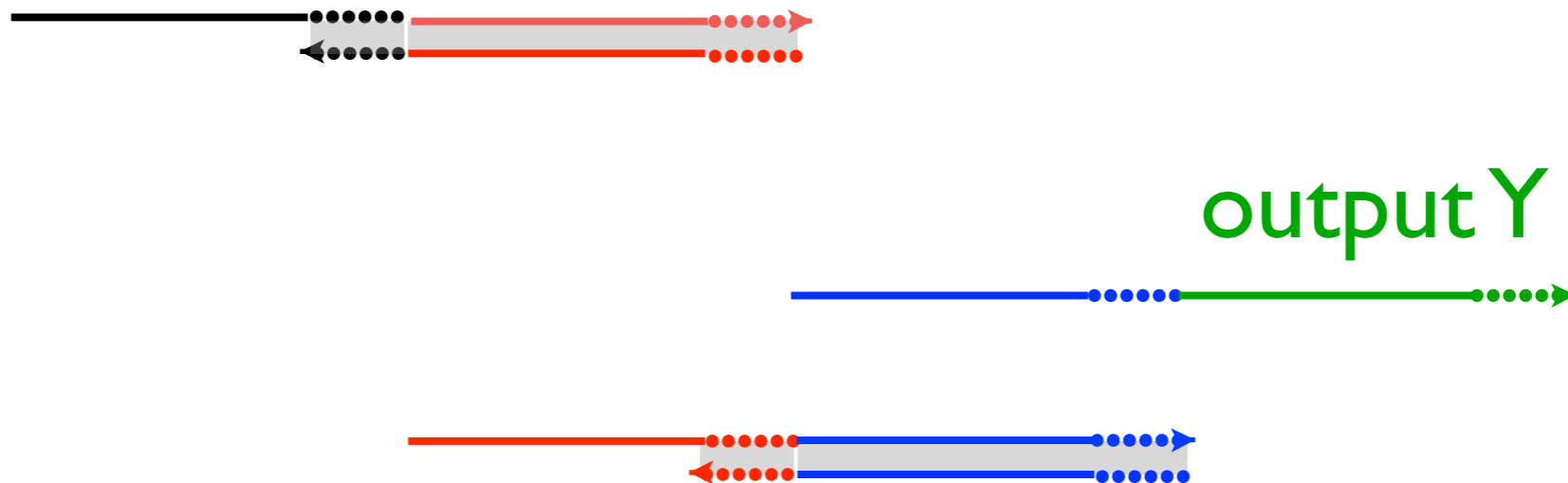
# Sequence Independence

Translator (a “wire”):  $X \rightarrow Y$

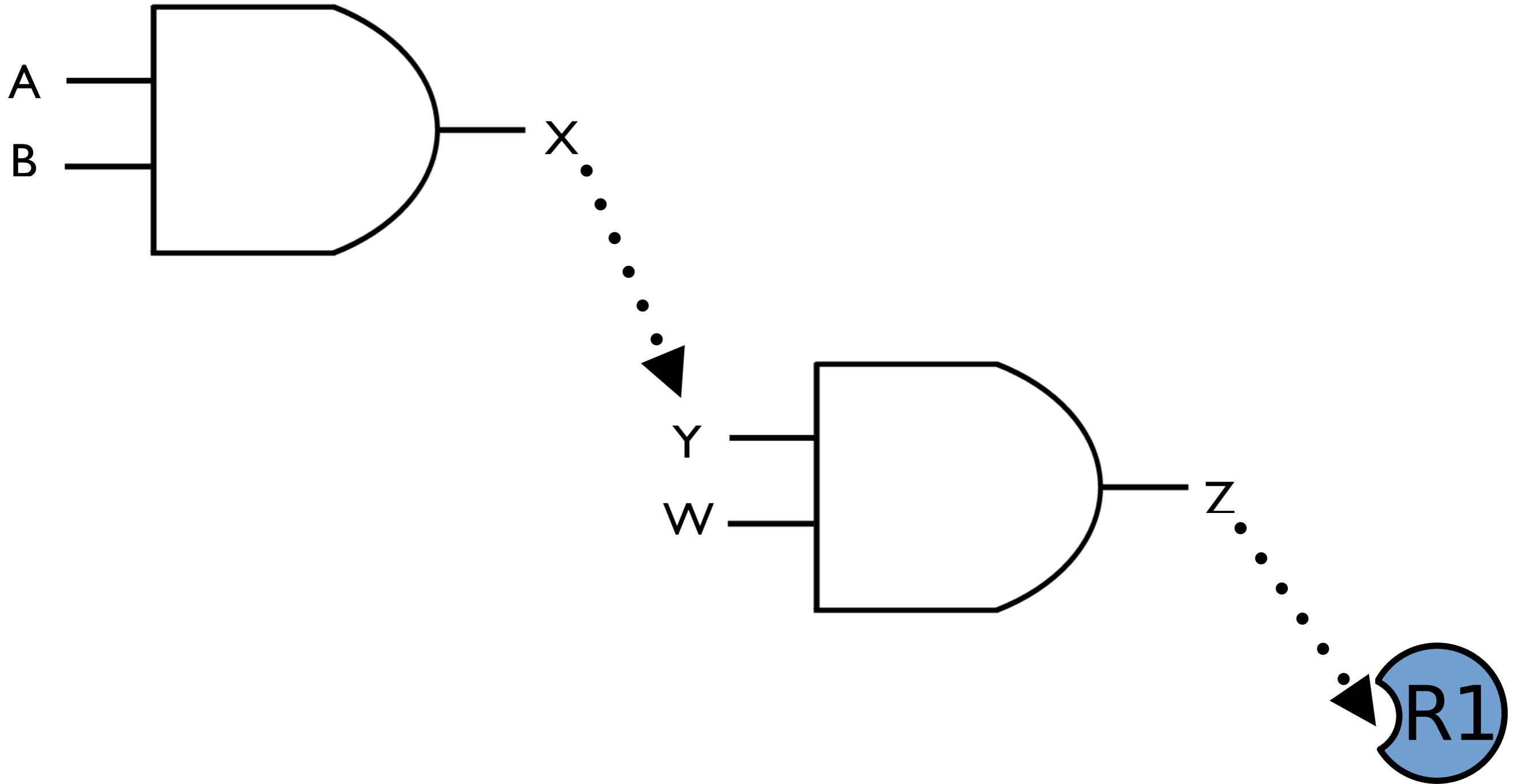


# Sequence Independence

Translator (a “wire”):  $X \rightarrow Y$

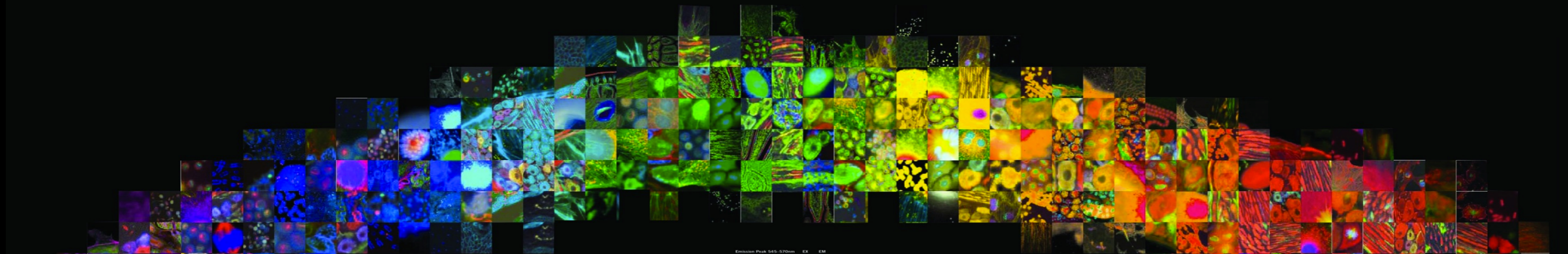


# Reading Output

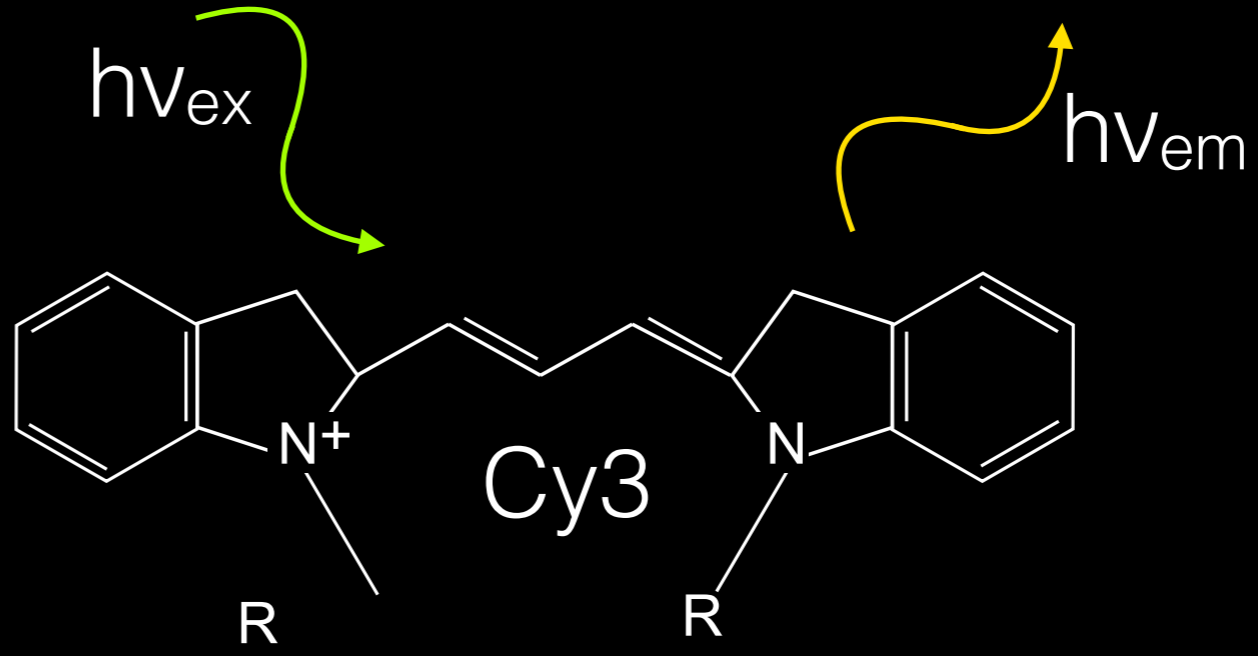


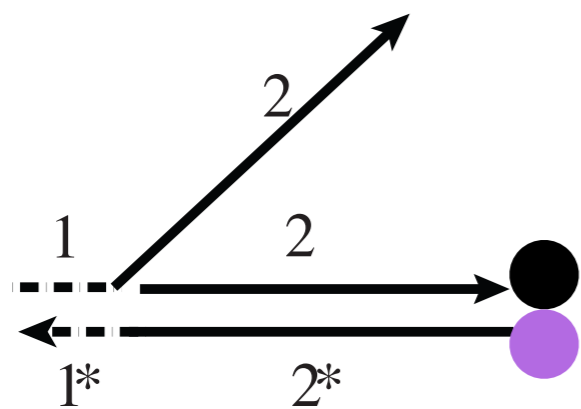


# F L U O R O P H O R E S C E N C E

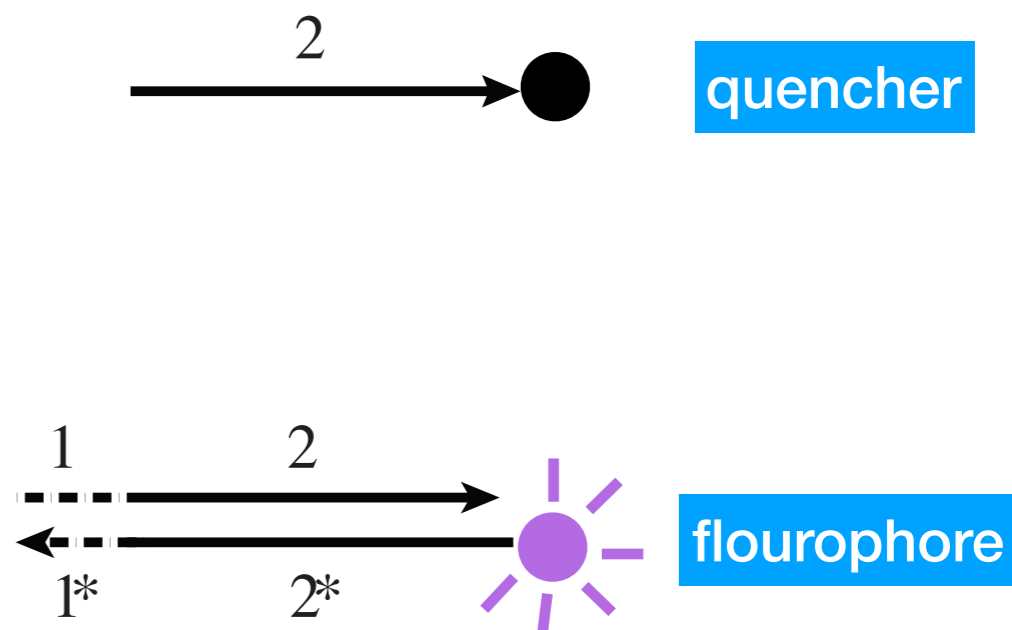


Emission Peak 375-425nm	EX	EM	Emission Peak 490-510nm	EX	EM	Emission Peak 510-545nm	EX	EM	Emission Peak 545-570nm	EX	EM	Emission Peak 570-585nm	EX	EM	Emission Peak 585-595nm	EX	EM	Emission Peak 595-605nm	EX	EM	Emission Peak 605-655nm	EX	EM	Emission Peak 655-800nm	EX	EM			
Calcium Blue	375	420	Acridine orange (+DNA)	492	518	5-Carboxyfluorescein (CFAM)	492	518	Rhodamine GG	525	555	Di-4-ANEPPS	565	573	ATTO-TAG™ FG	486	591	Acridine orange (+RNA)	460	650	5-Carboxyaphthofluorescein	598	668	APC	630	650	APC-Cy7	630	650
Calcium Blue™	400	420	Alexa Fluor® 480	488	505	Alexa Fluor® 488	485	519	ATTO 565	525	555	Rhodamine B	565	590	ATTO 565	565	590	Alexa Fluor® 594	591	618	Alexa Fluor® 647	643	669	APC-Cy5	630	650	APC-Cy7	630	650
DIDS	341	414	BOBYP™ 1	402	481	BOBYP™ 1	402	481	ATTO 565-CRCA	460	560	Rhodamine B™	590	590	ATTO 565	565	590	Alexa Fluor® 647	643	669	Alexa Fluor® 680	679	702	APC-Cy5	630	650	APC-Cy7	630	650
Fast Blue	365	420	BOBYP™ 2	402	481	BOBYP™ 2	402	481	Alexa Fluor® 555	553	568	Rhodamine B™	590	590	ATTO 565	565	590	Alexa Fluor® 647	643	669	Alexa Fluor® 680	679	702	APC-Cy5	630	650	APC-Cy7	630	650
Fluoro-Jade® (low pH)	323	408	BOBYP™ 3	402	481	BOBYP™ 3	402	481	ATTO 565	525	555	Rhodamine B™	590	590	ATTO 565	565	590	Alexa Fluor® 647	643	669	Alexa Fluor® 680	679	702	APC-Cy5	630	650	APC-Cy7	630	650
Indo-1 (high calcium)	330	401	BOBYP™ 4	402	481	BOBYP™ 4	402	481	Alexa Fluor® 555	553	568	Rhodamine B™	590	590	ATTO 565	565	590	Alexa Fluor® 647	643	669	Alexa Fluor® 680	679	702	APC-Cy5	630	650	APC-Cy7	630	650
LysoTracker™ Blue (pH 5)	374	424	BOBYP™ 5	402	481	BOBYP™ 5	402	481	ATTO 565	525	555	Rhodamine B™	590	590	ATTO 565	565	590	Alexa Fluor® 647	643	669	Alexa Fluor® 680	679	702	APC-Cy5	630	650	APC-Cy7	630	650
Mag Indo-1	330	417	BOBYP™ 6	402	481	BOBYP™ 6	402	481	Alexa Fluor® 555	553	568	Rhodamine B™	590	590	ATTO 565	565	590	Alexa Fluor® 647	643	669	Alexa Fluor® 680	679	702	APC-Cy5	630	650	APC-Cy7	630	650
Pyrene	345	378	BOBYP™ 7	402	481	BOBYP™ 7	402	481	ATTO 565	525	555	Rhodamine B™	590	590	ATTO 565	565	590	Alexa Fluor® 647	643	669	Alexa Fluor® 680	679	702	APC-Cy5	630	650	APC-Cy7	630	650

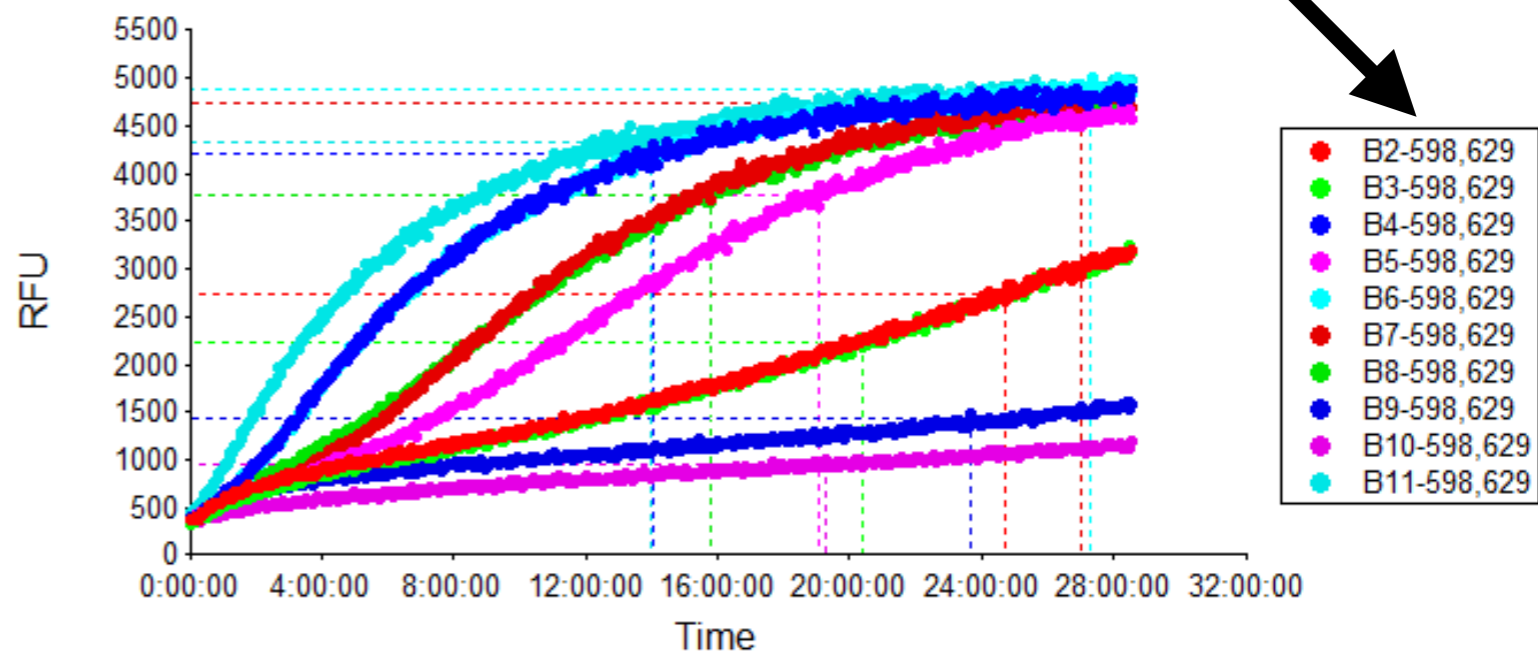




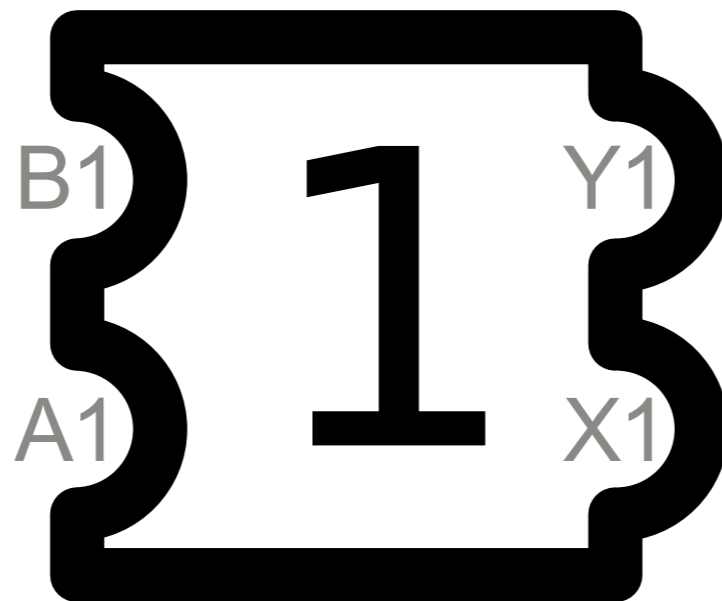
**displace**



**read many different samples**

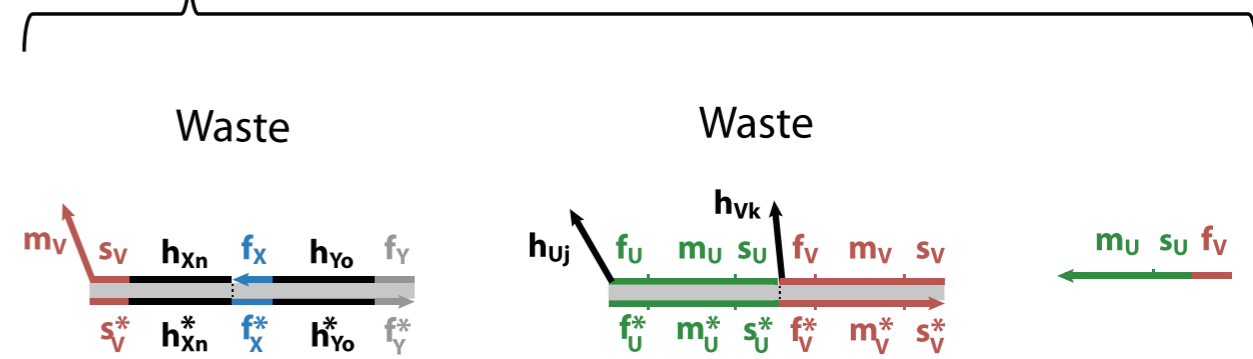
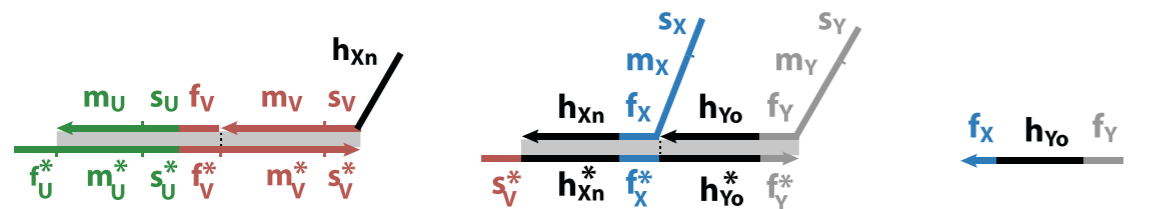


# A reaction gate

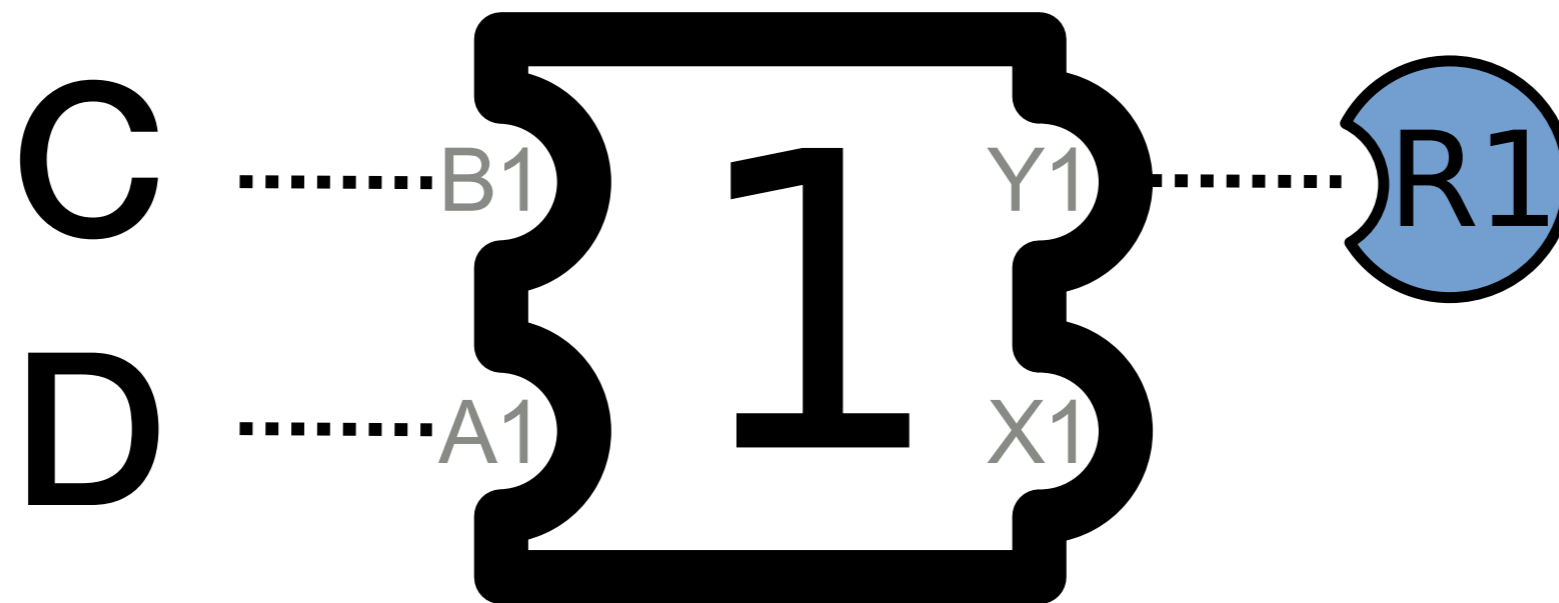


This *universal component* can realize a number of logic gates

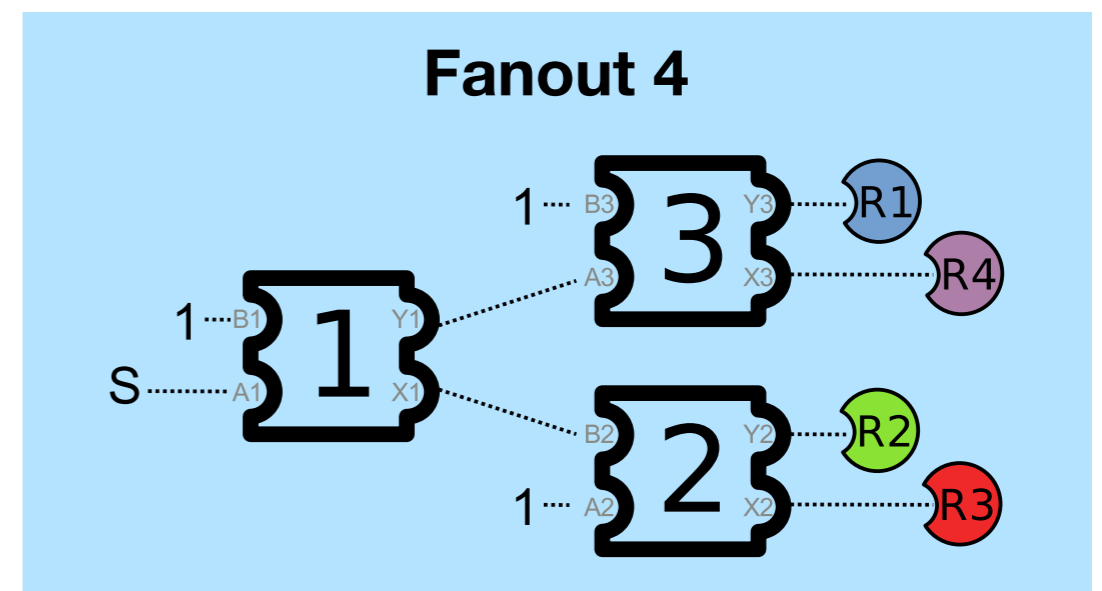
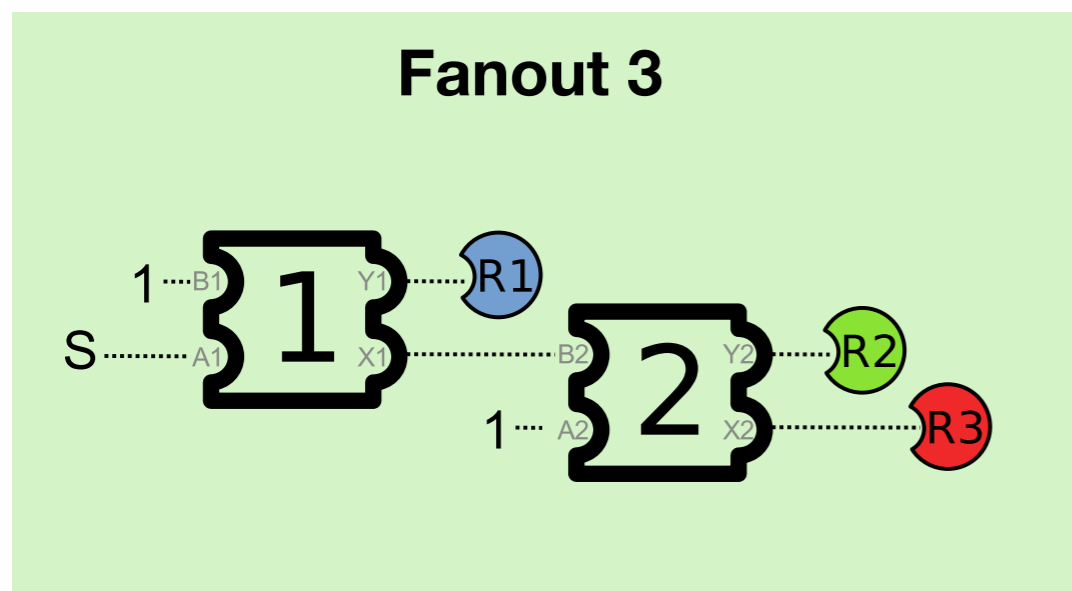
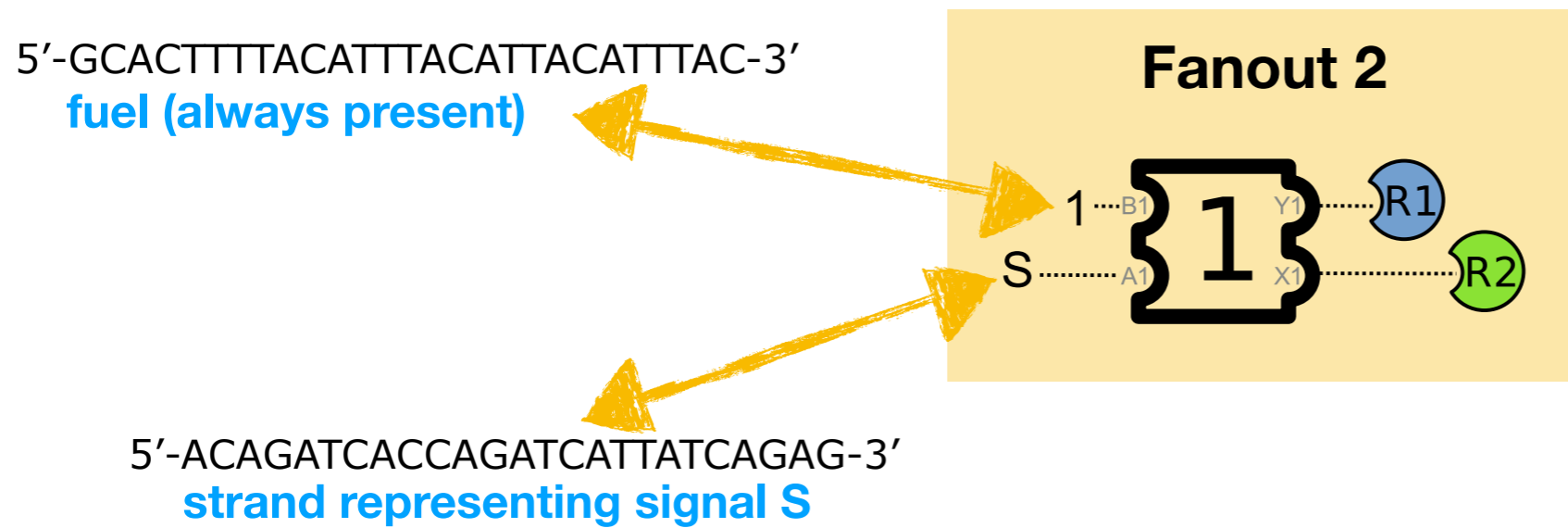
We start with large excess of DNA complexes (fuels) that mediate the reaction:



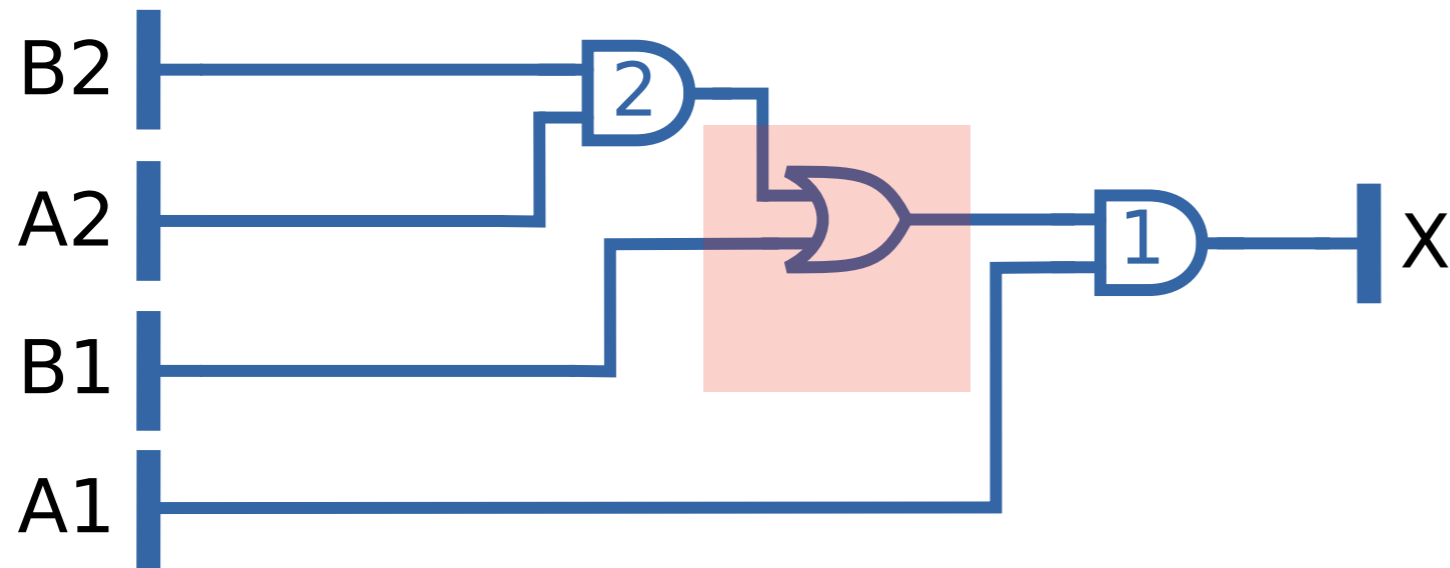
# AND gate



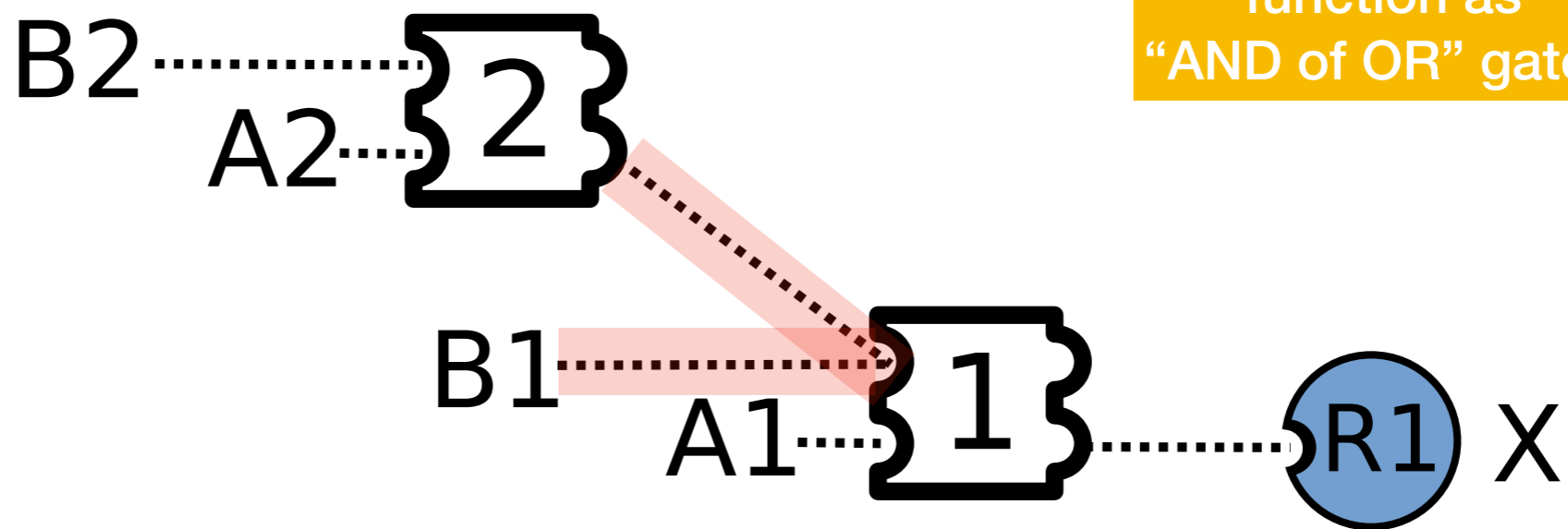
# Signal Fanout Gates



# Handling OR gates

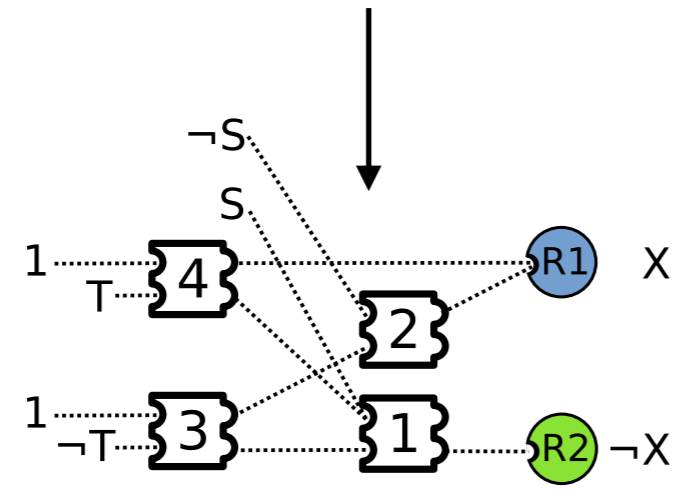
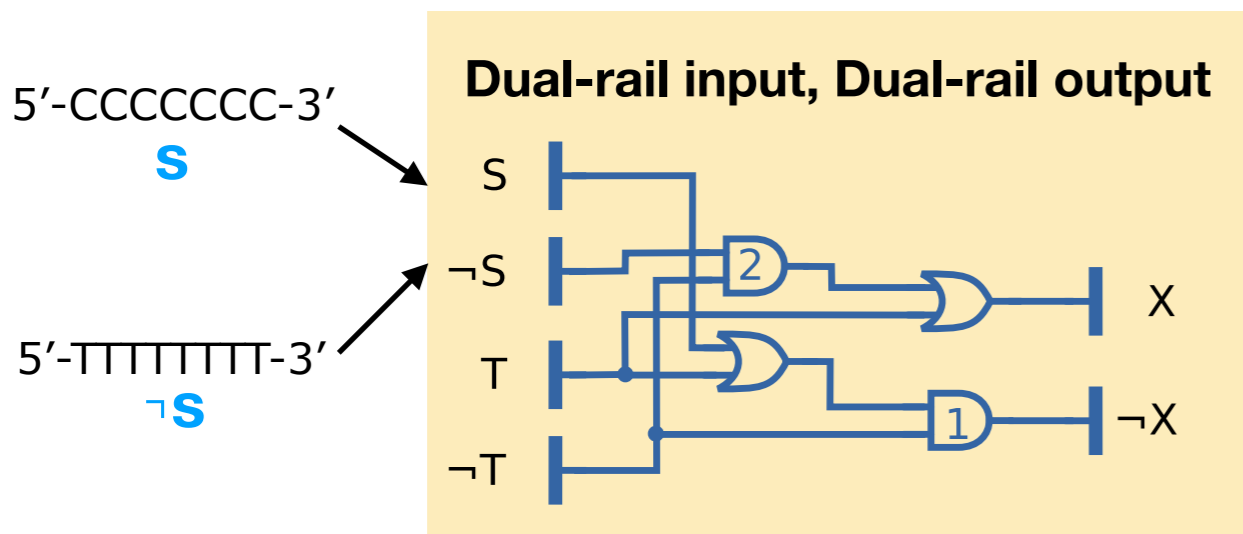
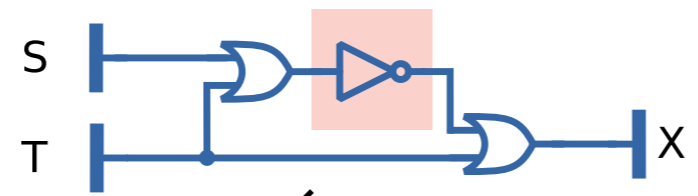


Reaction gates  
function as  
"AND of OR" gates



$$\text{AND}( A1, \text{OR}( B1, \text{AND}( A2, B2 ) ) )$$

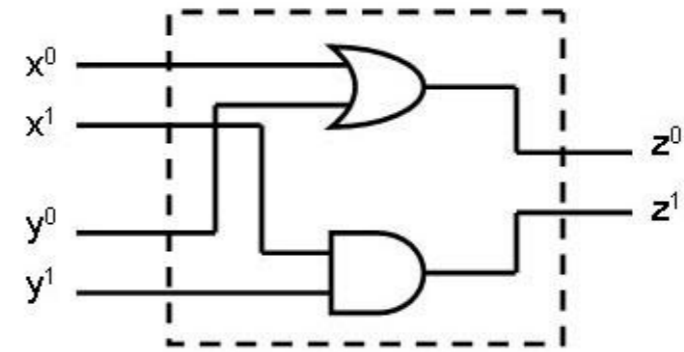
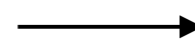
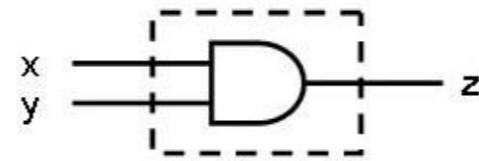
# Handling NOT gates



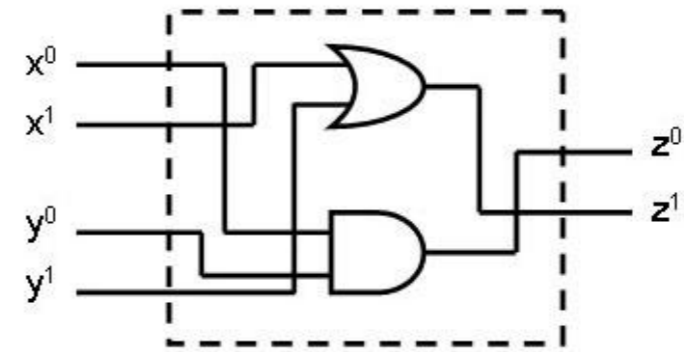
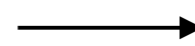
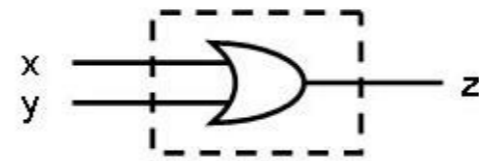


# Dual rail logic

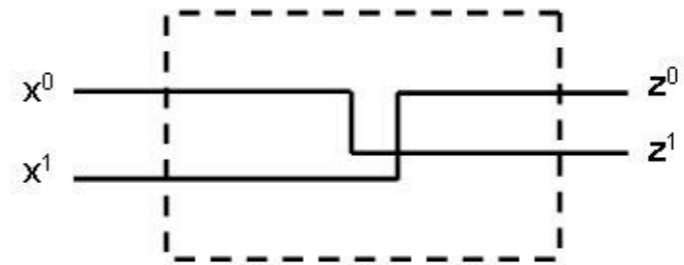
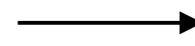
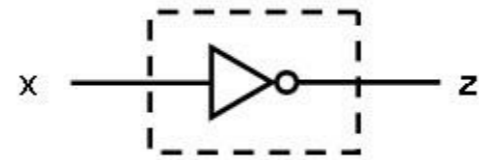
AND



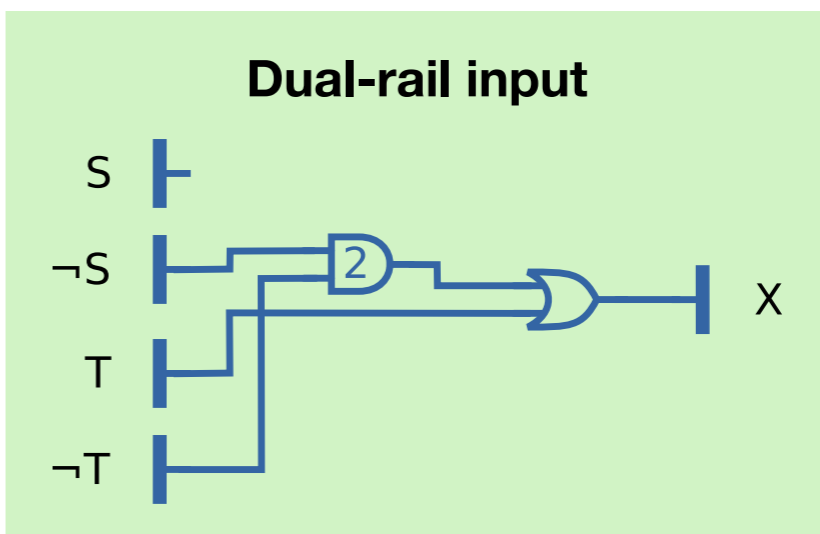
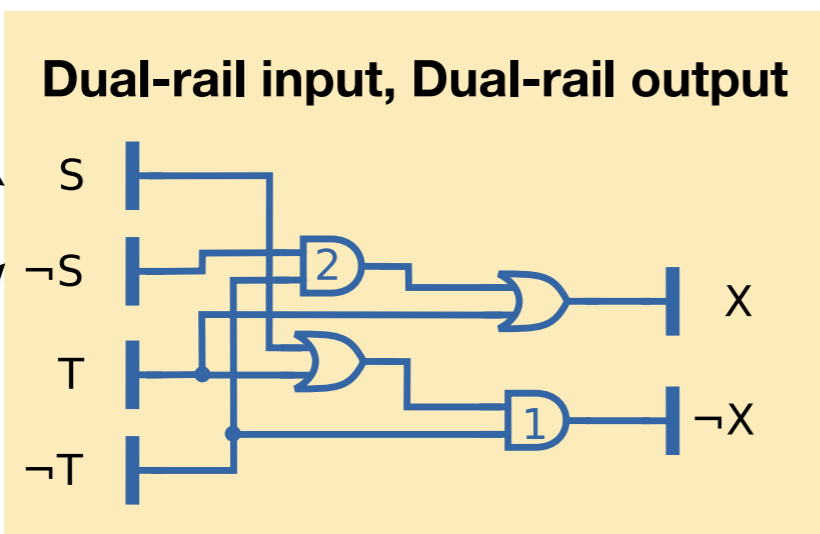
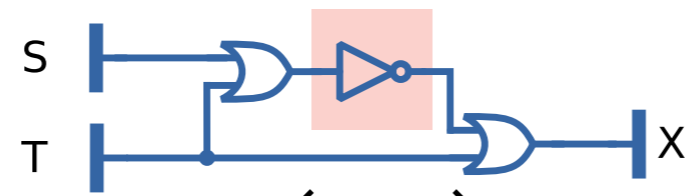
OR



NOT

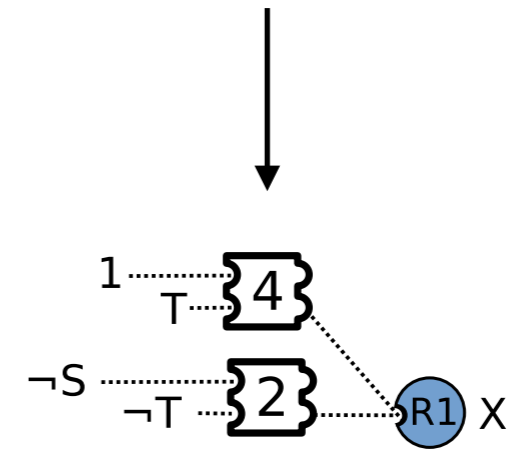
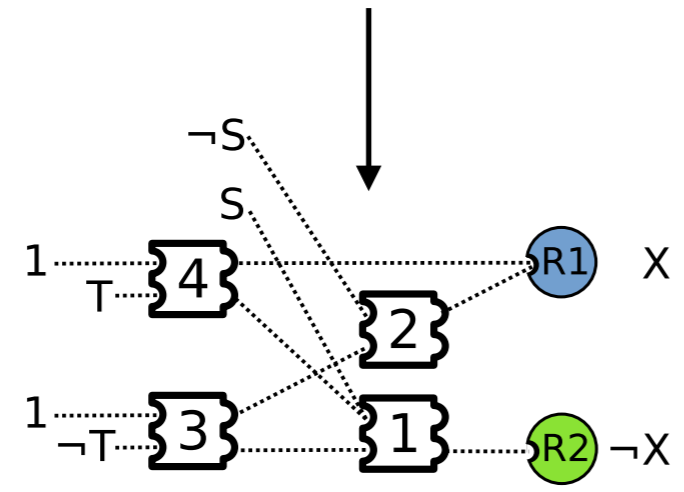


# Handling NOT gates

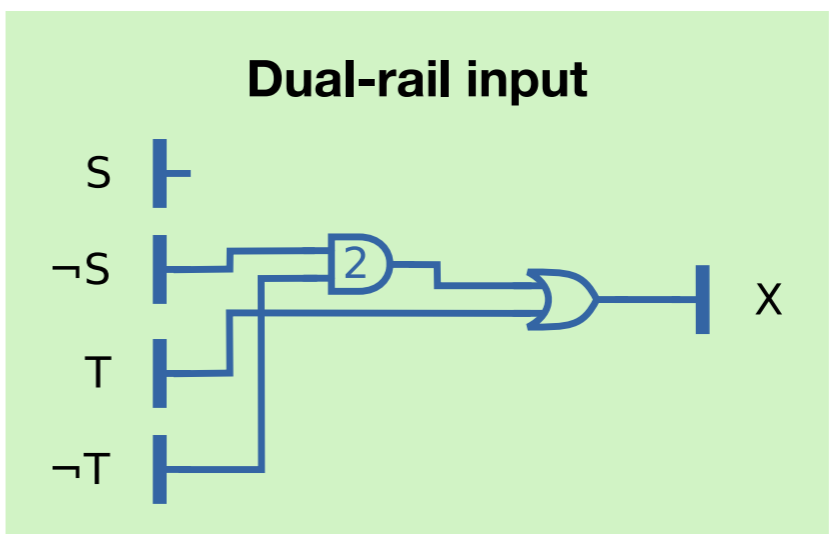
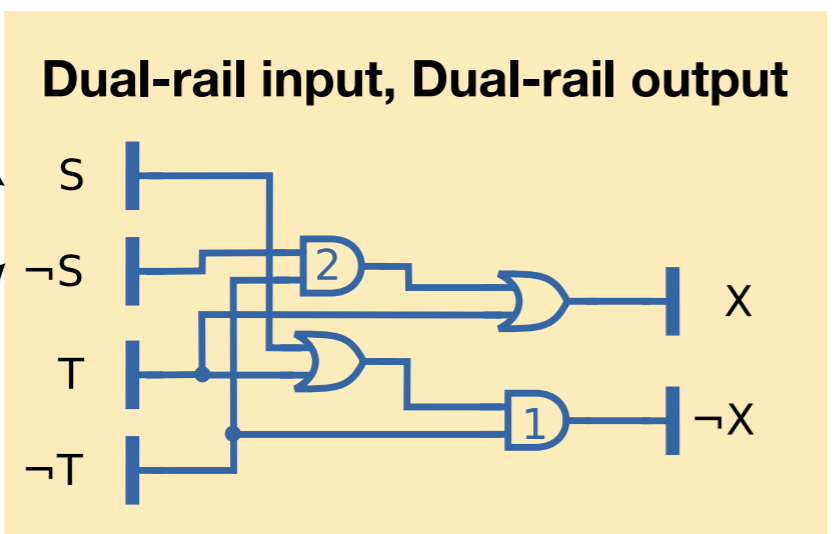
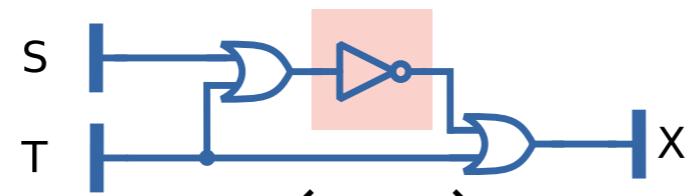


5'-CCCCCCC-3'  
**S**

5'-TTTTTTTT-3'  
**¬S**

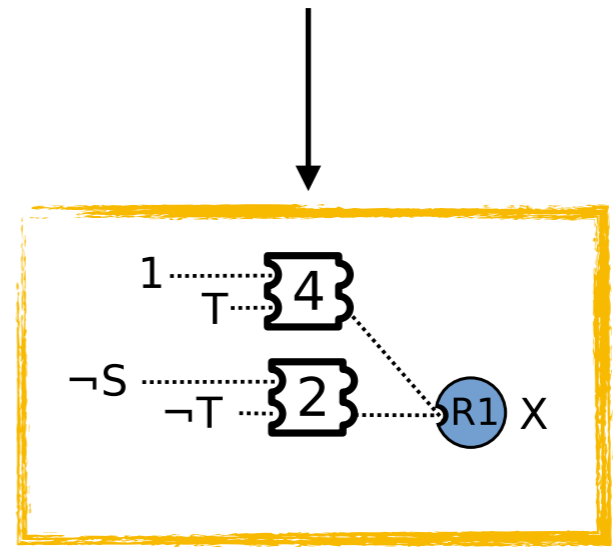
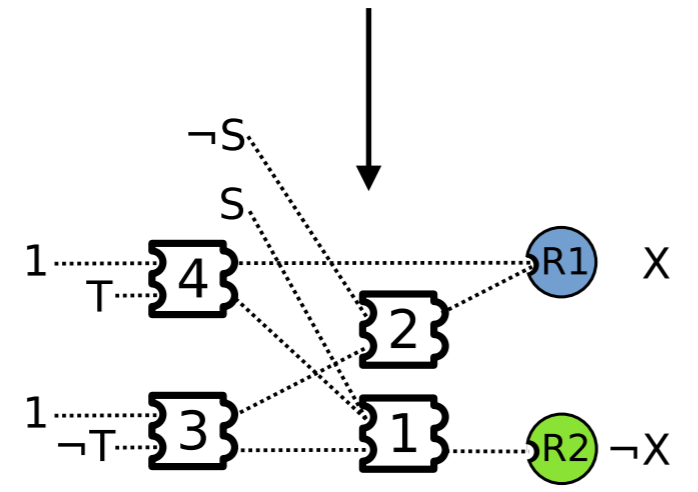


# Handling NOT gates



5'-CCCCCCC-3'  
**S**

5'-TTTTTTTT-3'  
 **$\neg S$**



With reaction gates, wires,  
and dual-rail encoding, we can  
build any combinatorial circuit

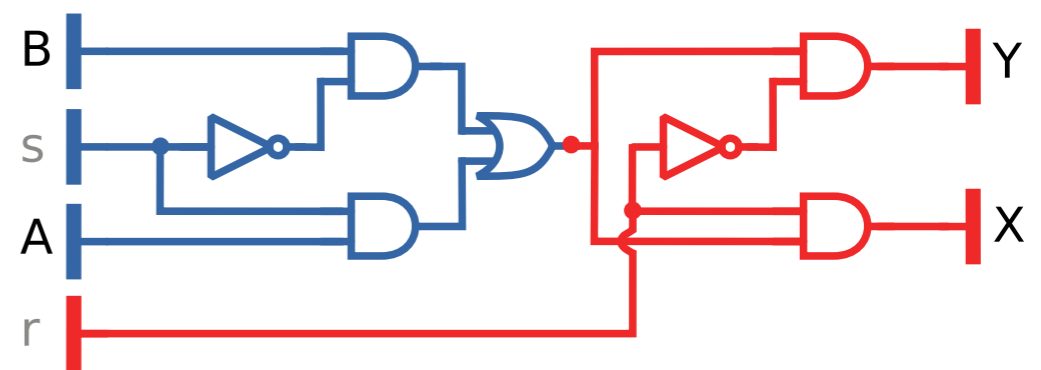
# Tutorial Outline

- ▶ Review of strand displacement
- ▶ Building and composing logic gates
- ▶ **Tools for designing and verifying circuits**
- ▶ Robustness of strand displacement

# How do you design the circuit?

2-input MUX				2-output DEMUX			
A	B	s	out	in	r	X	Y
0	0	0	0	0	0	0	0
0	1	0	1	1	0	0	1
1	0	0	0	0	1	0	0
1	1	0	1	1	1	1	0
0	0	1	0	0	0	1	0
0	1	1	0	0	1	1	0
1	0	1	1	0	0	1	1
1	1	1	1	0	1	1	1

Compile from Verilog, or truth table, into AND-OR-NOT circuit



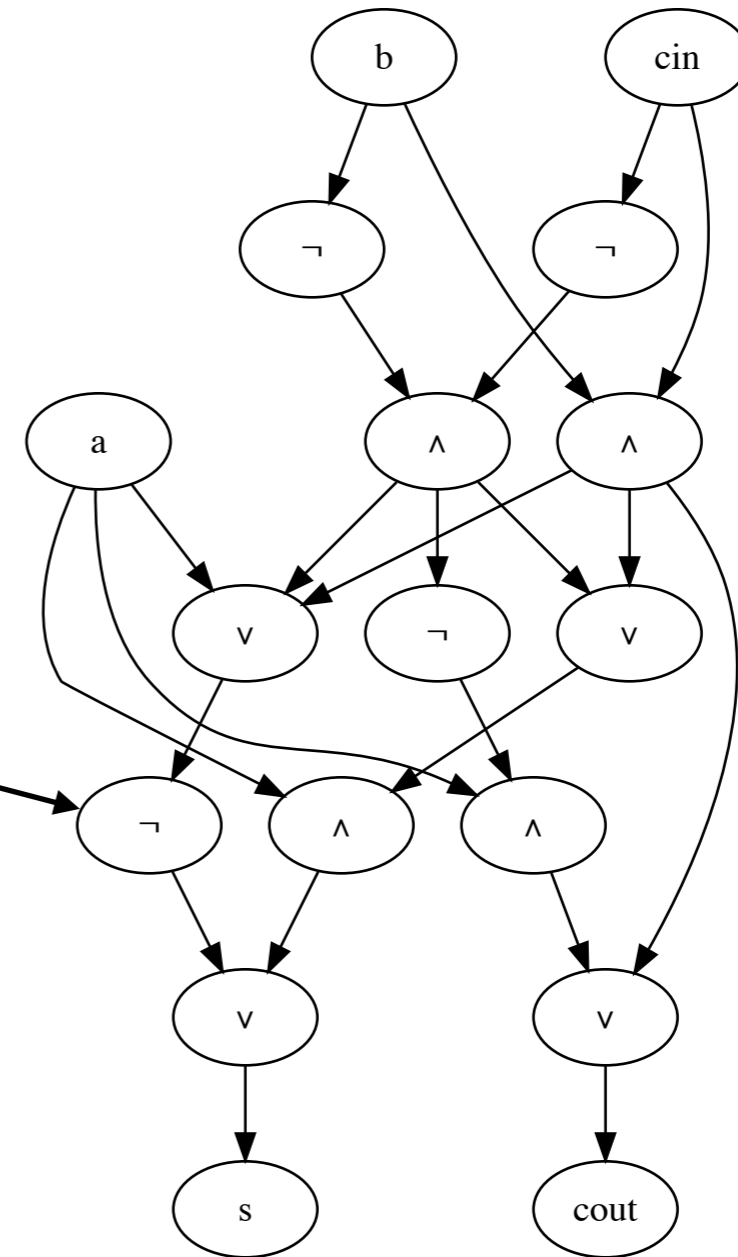
# How do you design the circuit?

Inputs			Outputs	
<b>A</b>	<b>B</b>	<b>C<sub>in</sub></b>	<b>C<sub>out</sub></b>	<b>S</b>
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

# How do you design the circuit?

- ✓ Find minimized AND-OR-NOT circuit using **ABC**

Need to remove explicit NOT gates

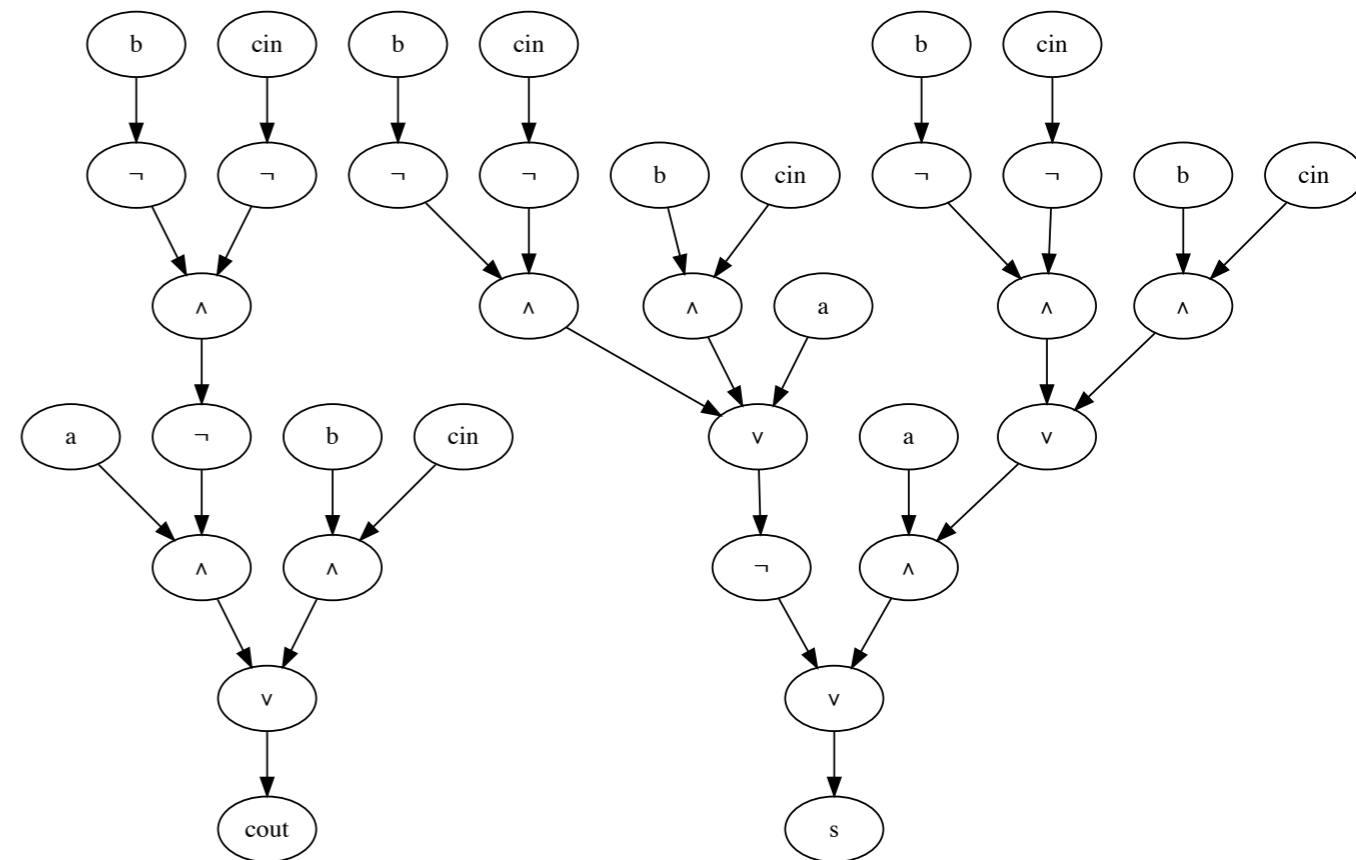




# How do you design the circuit?

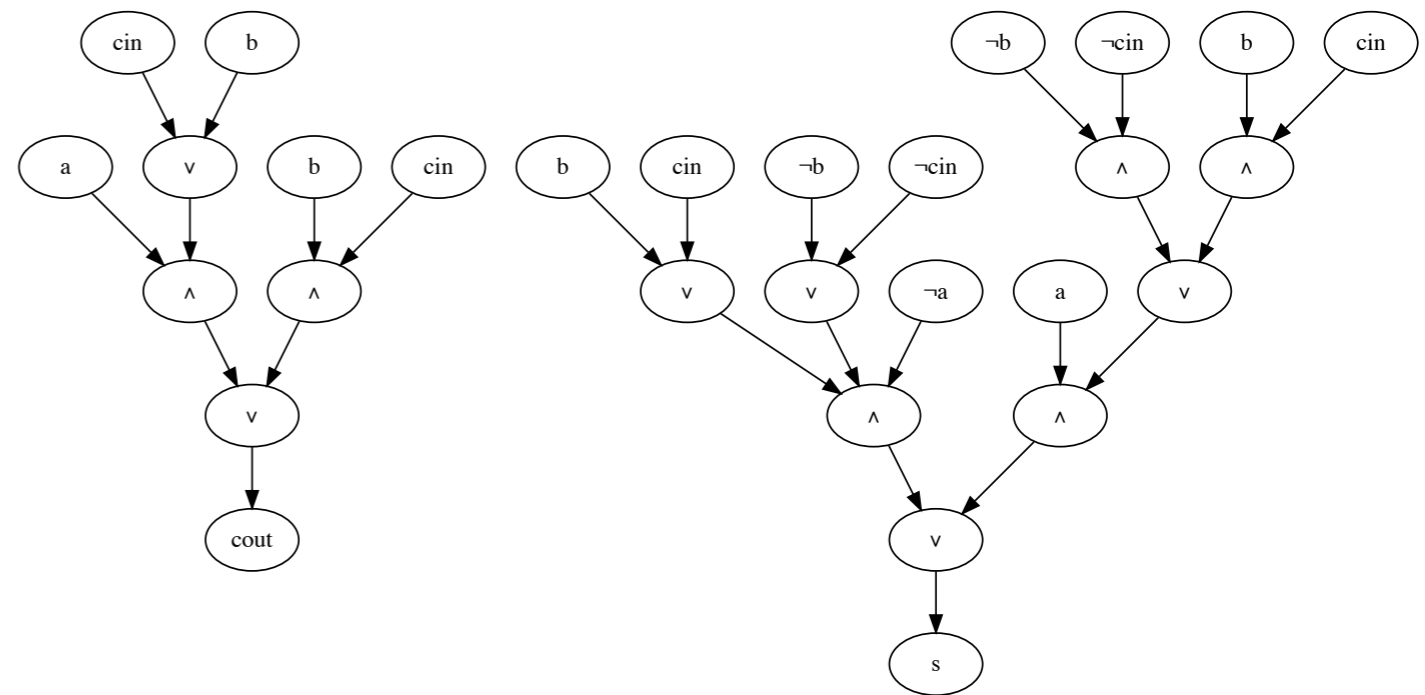
✓ Find minimized AND-OR-NOT circuit using ABC

✓ “Tree-ify” circuit



# How do you design the circuit?

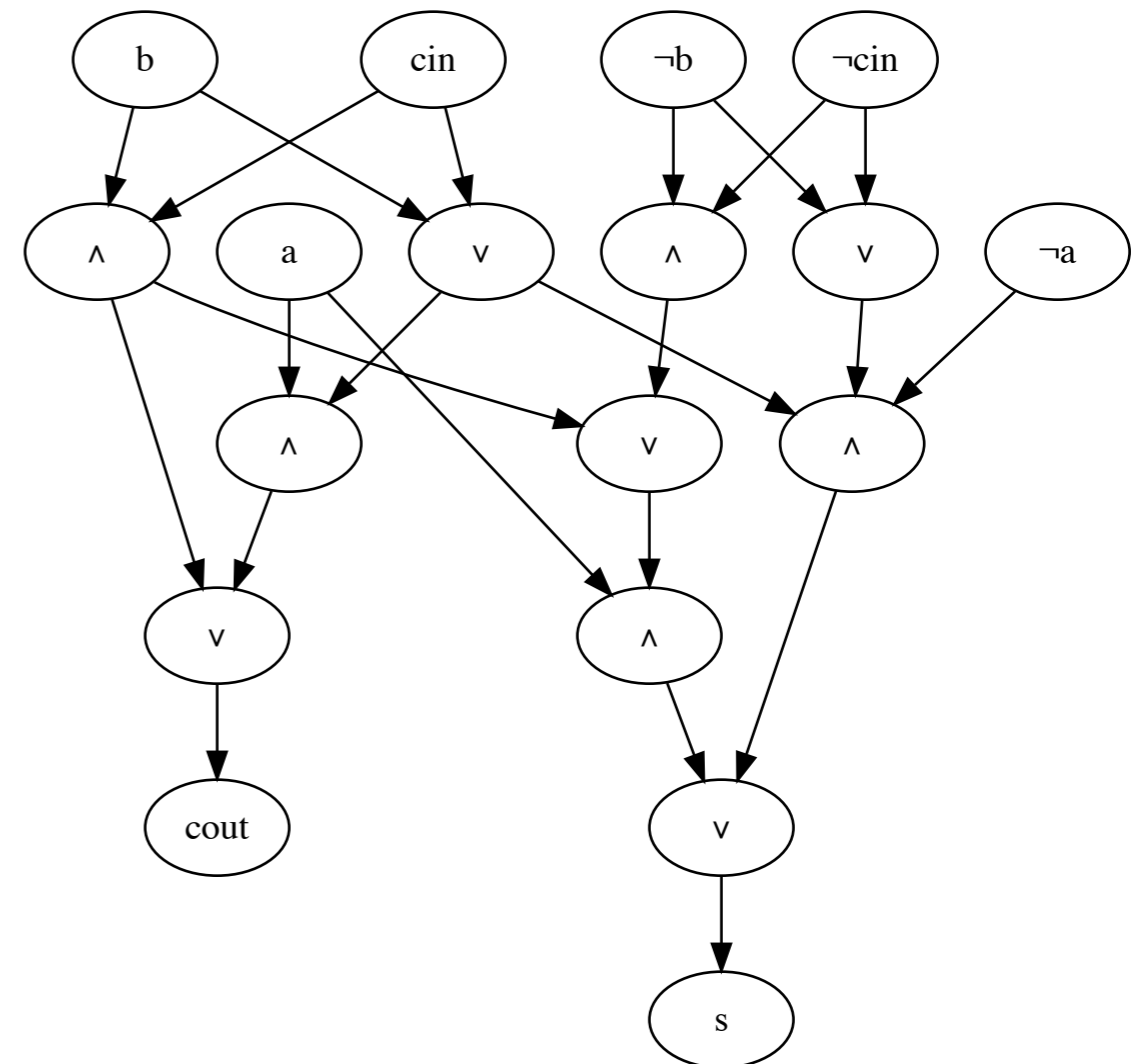
- ✓ Find minimized AND-OR-NOT circuit using ABC
- ✓ “Tree-ify” circuit
- ✓ Push negations to literal level (dual-rail inputs)



# How do you design the circuit?

- ✓ Find minimized AND-OR-NOT circuit using ABC
- ✓ “Tree-ify” circuit
- ✓ Push negations to literal level (dual-rail inputs)
- ✓ Compress circuit

Circuit now using  
dual-rail input



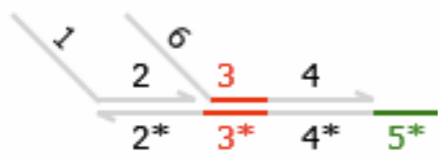
# From circuit to DSD system

DSD: formal language for describing and modeling strand displacement cascades

<http://lepton.research.microsoft.com/webdna/>

$\langle 1 \rangle [2]: \langle 6 \rangle [3^{\wedge} 4]: 5^{\wedge}$

=



The screenshot shows the webDNA interface. On the left is a code editor with the following content:

```

directive sample 200000.0 2000
directive plot <y1 t^ y1 x^>; <y2 t^
def bind = 0.00001 (* /nM/s *)
def unbind = 0.1 (* /s *)
def Excess = 100

new x@ bind,unbind
new t@ bind,unbind

def SpeciesL(N,a,l,a) = N * <a t^ a x
def SpeciesR(N,a,ar) = N * <a t^ ar>
def BinaryLRxRR(N,al,a,b,br,c,cr,d,dr)
  new i
  { constant N * t^:[a x^ b]<i t^ cr t
    | constant N * Excess * x^:[b i]:<c:
  }
def UnaryLxLL(N,al,a,cl,c,dl,d) = (* A
  new i
  { constant N * t^:[a x^]<i cl t^ dl t
    | constant N * Excess * x^:[i]:[cl t^
  }
def UnaryRx(N,a,ar) = (* A ->{N *}
  constant N * [a]:t^

(
  UnaryLxLL(1000,y1,y1,y1,y1,y1,y:
  | BinaryLRxRR(30000,y1,y1,y2,y2r,y:
  | UnaryRx(1000,y2,y2r) (* Y2 -c3->
  | SpeciesL(1000,y1,y1) (* Y1 *)
  | SpeciesR(1000,y2,y2r) (* Y2 *)
)
    
```

On the right is a reaction graph showing various DNA configurations and transitions between them, illustrating the execution of the code. The graph includes nodes representing DNA molecules and arrows representing reactions.

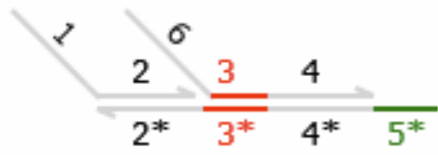
# From circuit to DSD system

DSD: formal language for describing and modeling strand displacement cascades

<http://lepton.research.microsoft.com/webdna/>

$\langle 1 \rangle [2] : \langle 6 \rangle [3^{\wedge} 4] : 5^{\wedge}$

=



The screenshot shows the webdna interface with a code editor on the left and a graphical representation of a strand displacement reaction on the right. The code includes directives for simulation and species definitions. The graphical representation shows a strand with segments y1, t, y1, x and its interaction with another strand.

## formal semantics

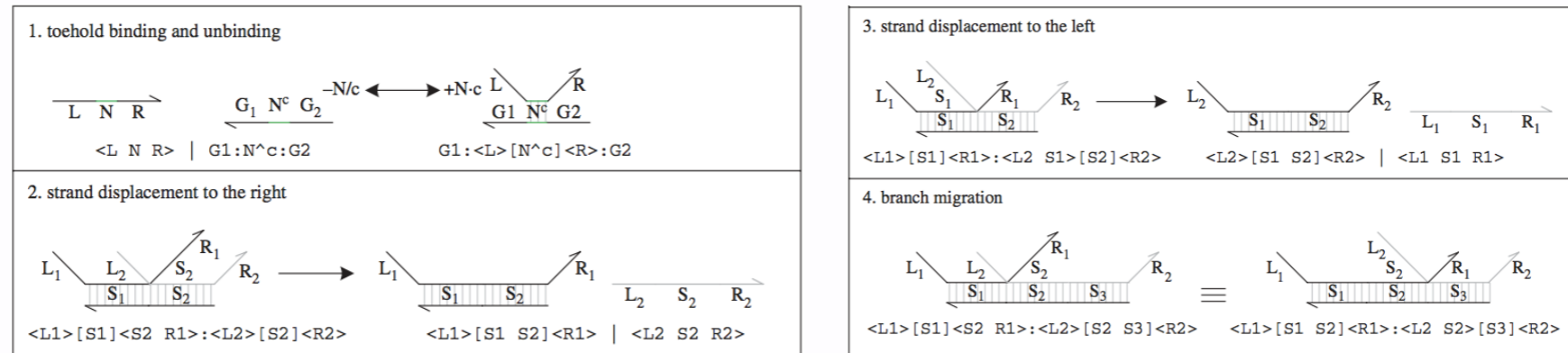
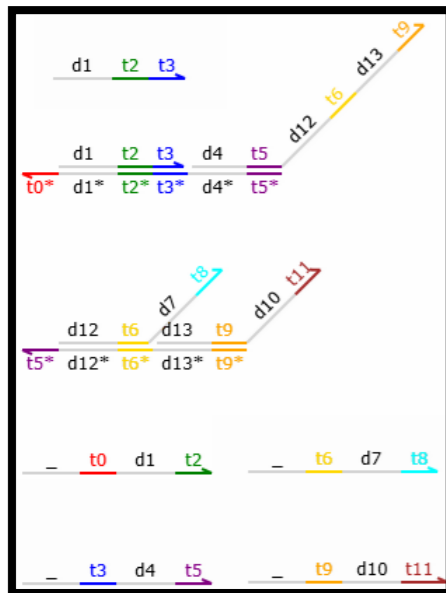


Figure 2. Reduction and branch migration rules of the strand displacement language. For each rule, the graphical representation at the top is equivalent to the program code at the bottom.

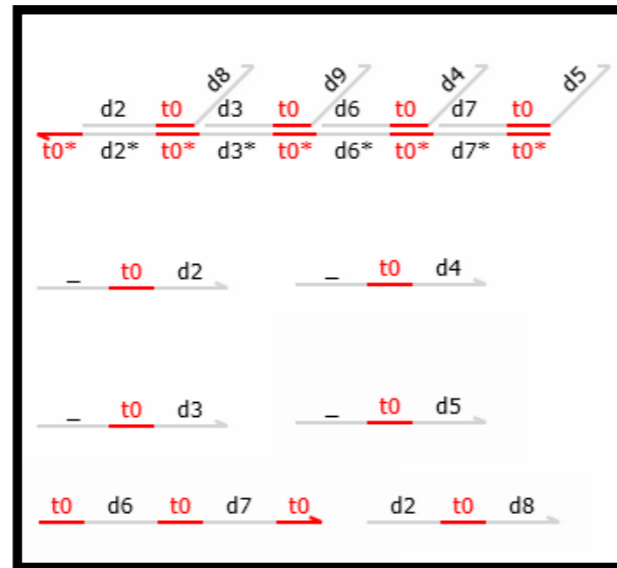
# From circuit to DSD system

$$A + B \rightarrow C + D$$

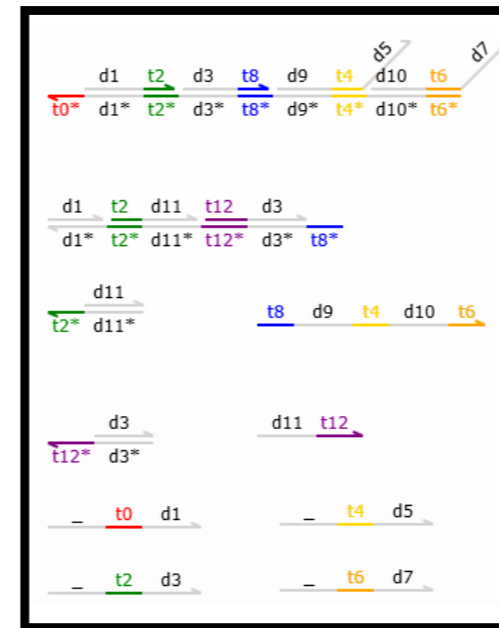
Soloveichik  
et al. (2010)



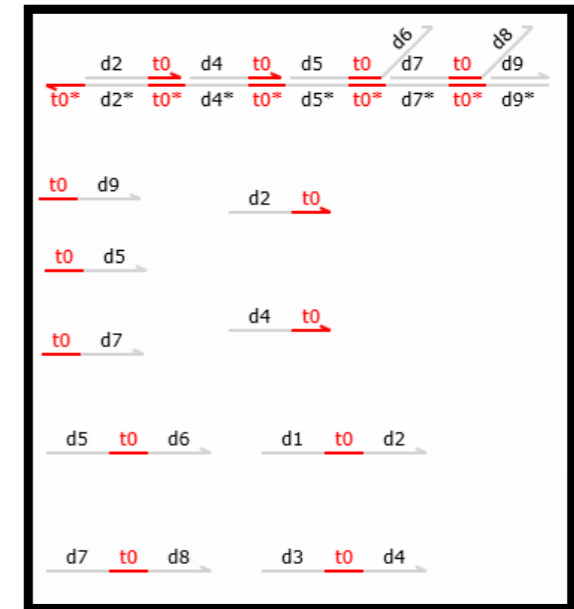
Lakin  
et al. (2012)



Cardelli (2011)



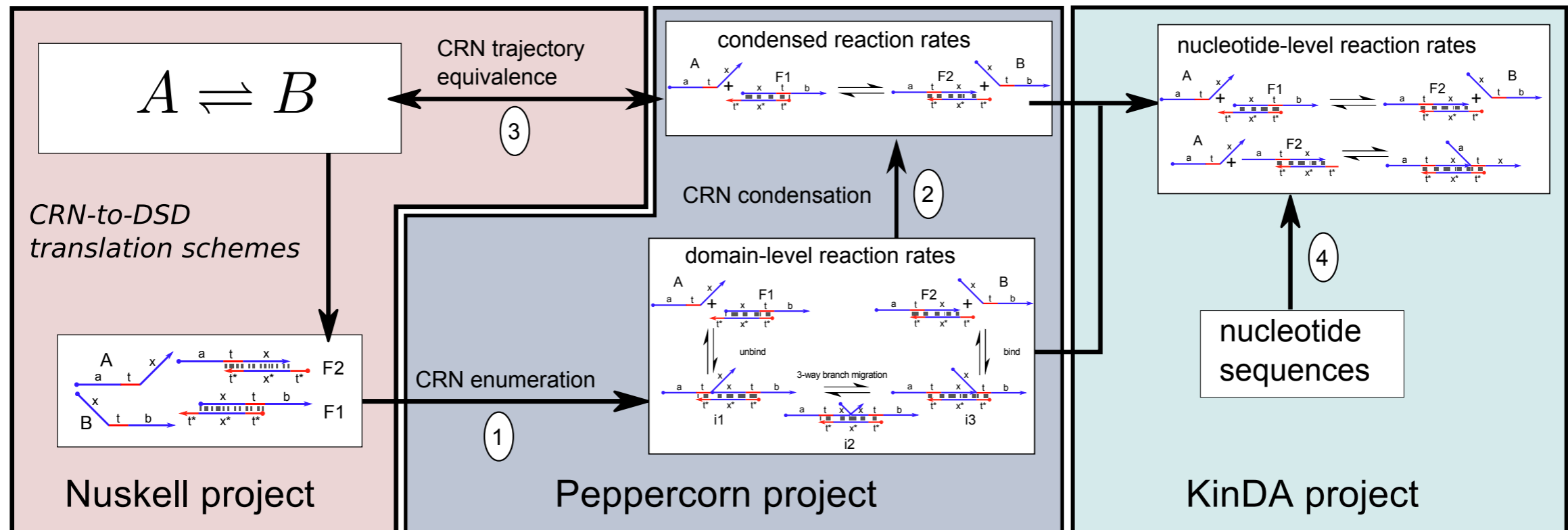
Qian et al. (2011)



Chen et al. (2012), Cardelli (2013), Srinivas (2015), Lakin et al. (2016), ...

Images drawn using VisualDSD, Lakin et al. (2012)

# The Nuskell compiler framework



Badelt et al. (2017) - Nuskell

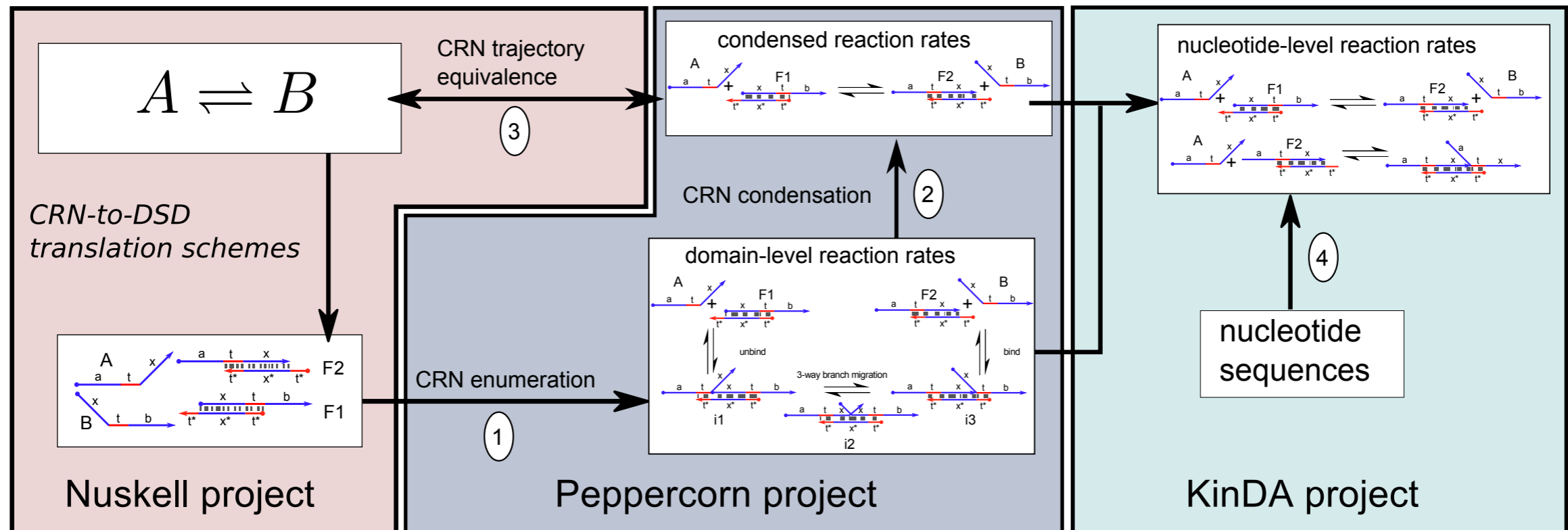
Grun et al. (2014) - Peppercorn

Shin et al. (2017) - CRN pathway decomposition equivalence

Johnson et al. (2018) - CRN bisimulation equivalence

Berleant et al. (submitted) - KinDA

# The Nuskell compiler framework



Badelt et al. (2017) - Nuskell

Grun et al. (2014) - Peppercorn

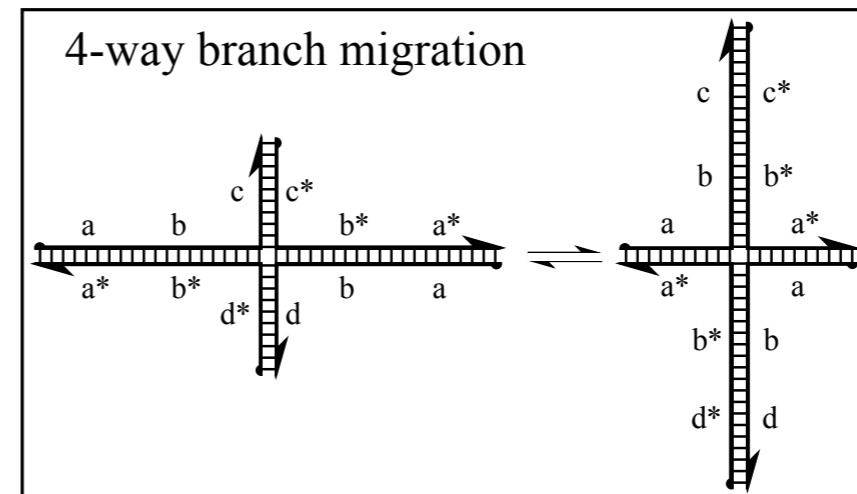
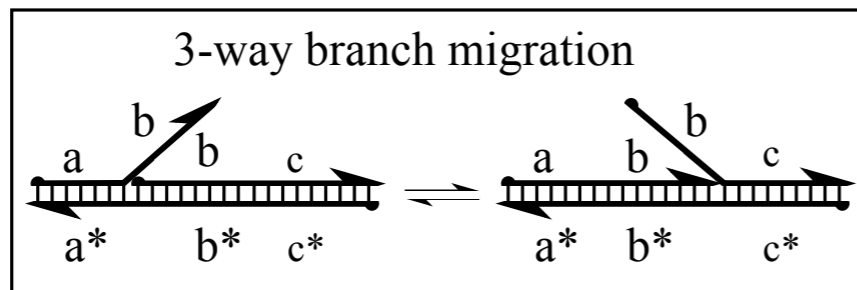
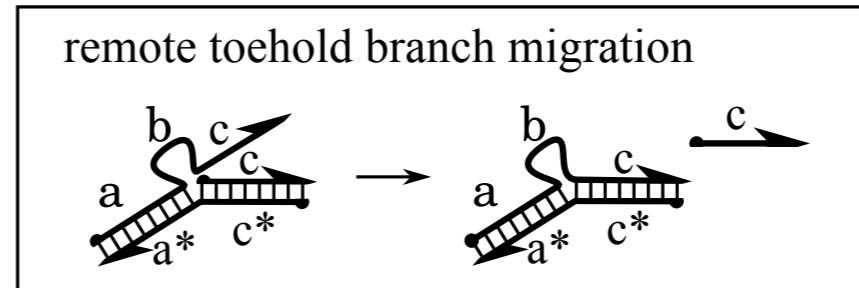
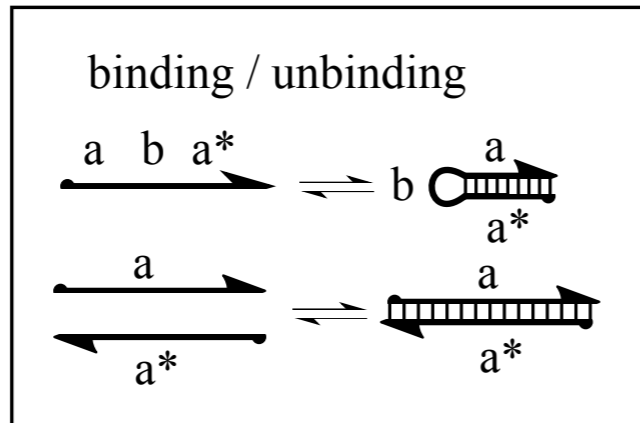
Shin et al. (2017) - CRN pathway decomposition equivalence

Johnson et al. (2018) - CRN bisimulation equivalence

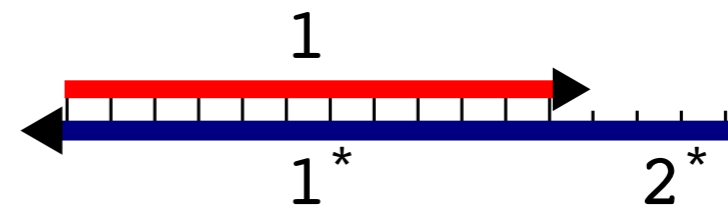
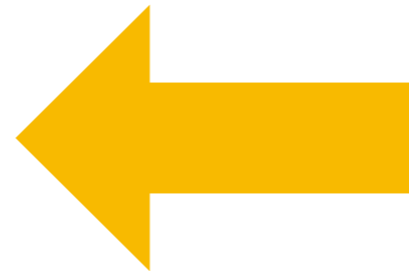
Berleant et al. (submitted) - KinDA



# Reaction Enumeration



# Designing Sequences



# Designing Sequences



## NUPACK

Analysis

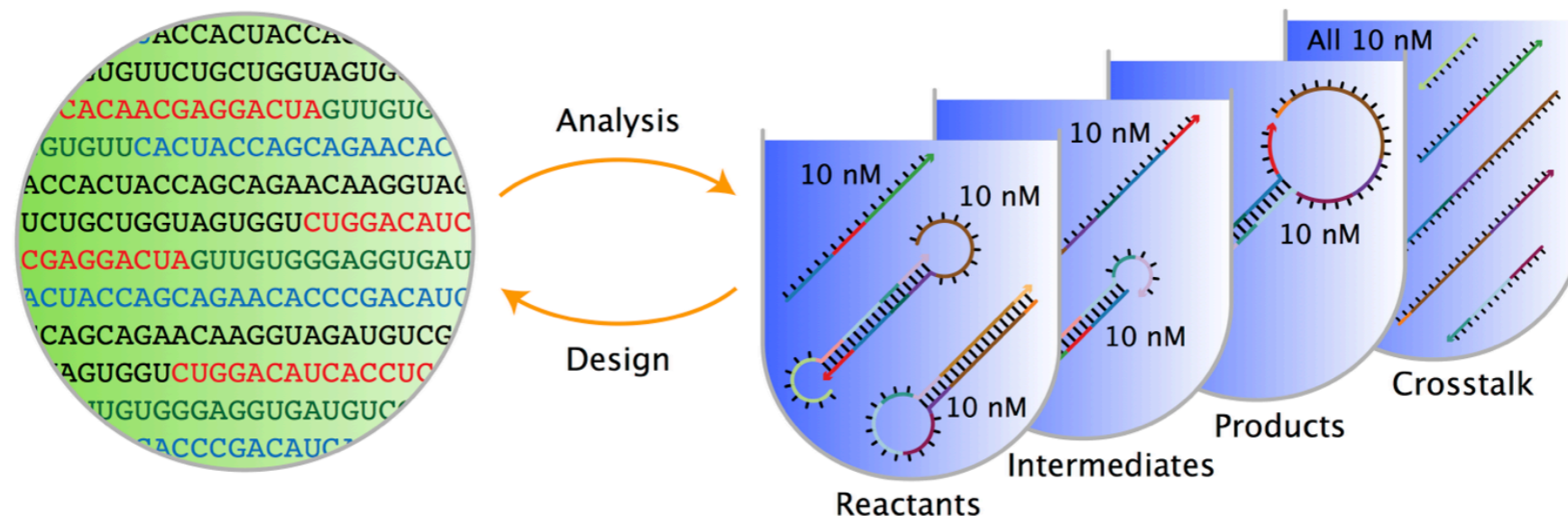
Design

Utilities

Downloads

**NUPACK** is a growing software suite for the analysis and design of nucleic acid structures, devices, and systems.

The NUPACK web application enables analysis and design of the equilibrium base-pairing properties of one or more test tubes of interacting nucleic acid strands:



Please [cite](#) the web application and algorithms appropriately; usage statistics are an important component in helping to secure funding for NUPACK development. We are happy to provide advice and [technical support](#).

— The NUPACK Team

**News:** Constrained multistate test tube design for reaction pathway engineering is now published! ([pdf](#), [supp info](#), [source code](#), [user guide](#))

# Designing Sequences

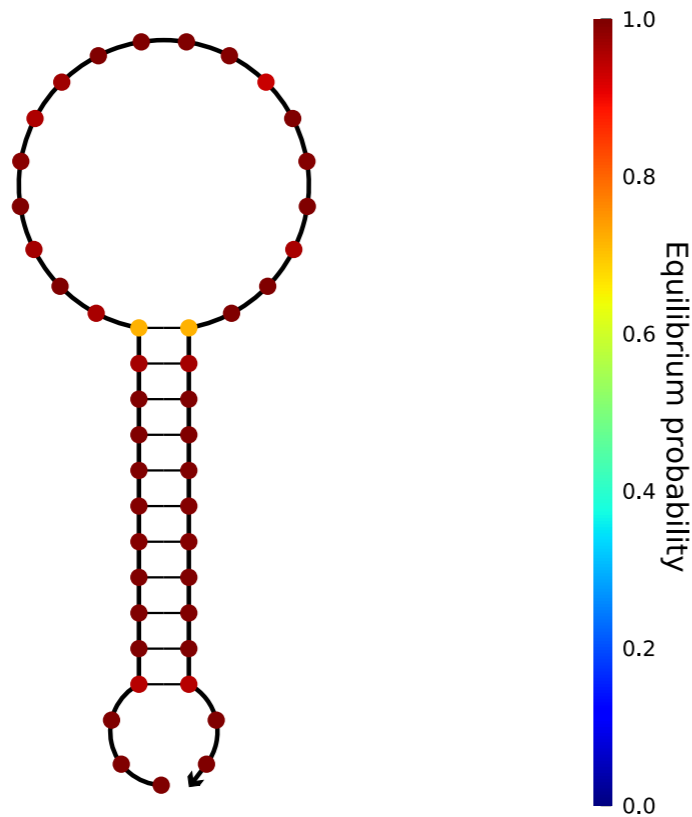


input X

poor sequence  
for a signal strand

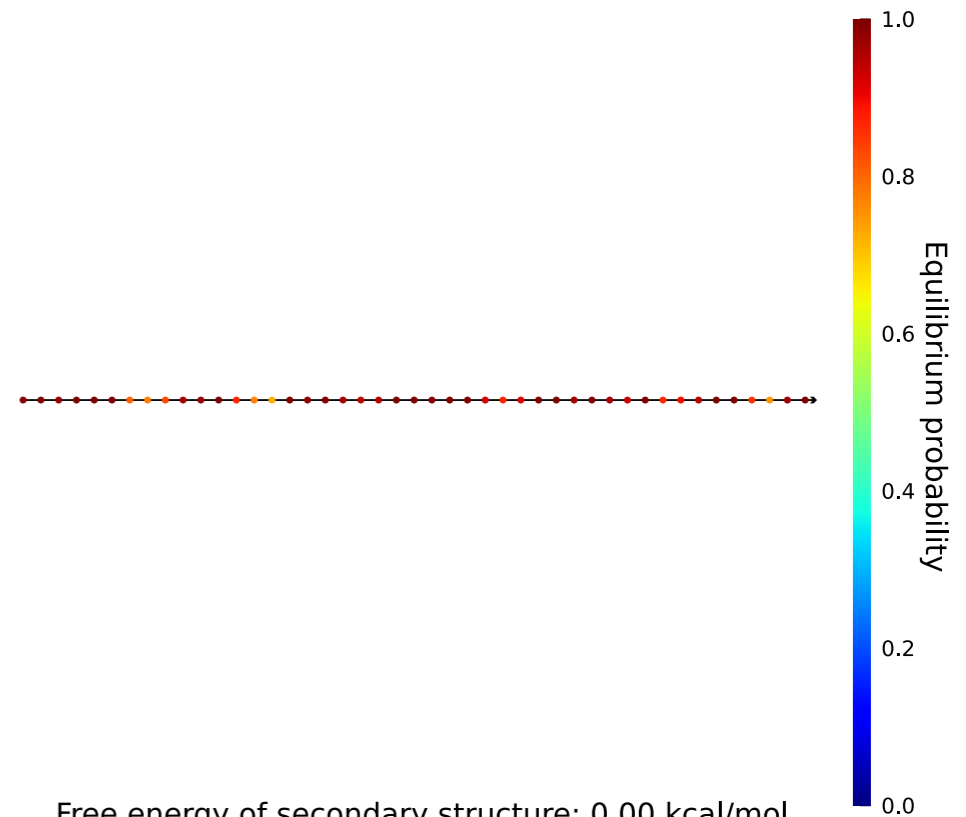
good sequence  
for a signal strand

MFE structure at 25.0 C



Free energy of secondary structure: -10.70 kcal/mol

MFE structure at 25.0 C



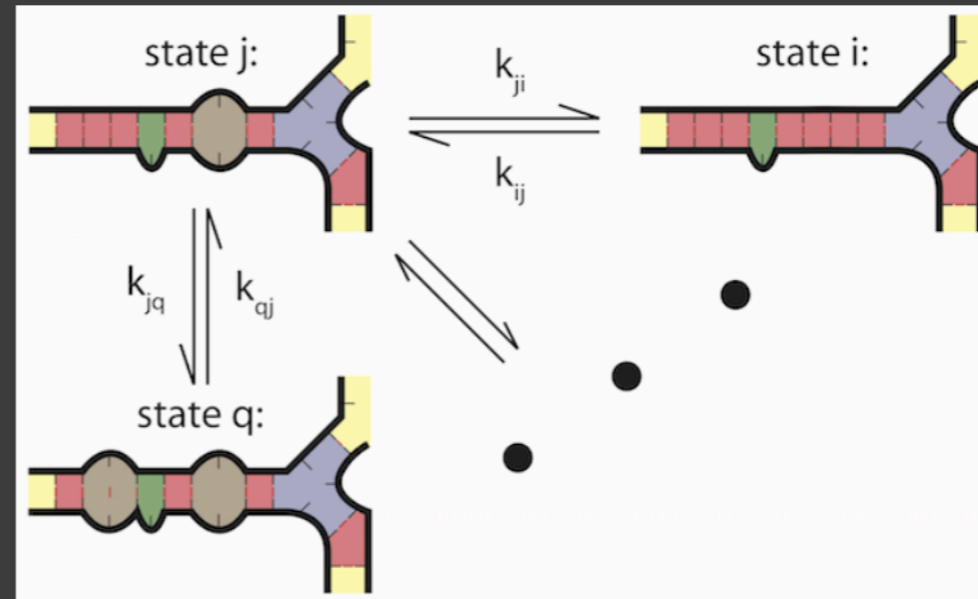
Free energy of secondary structure: 0.00 kcal/mol

# Multistrand.org to determine reaction rates

Multistrand is a software package for simulating the kinetics of multiple interacting nucleic acid strands. It is developed at the Winfree lab at the California Institute of Technology.

› DNA and Natural Algorithms Group @ Caltech

[Live demo](#)



## Key Features

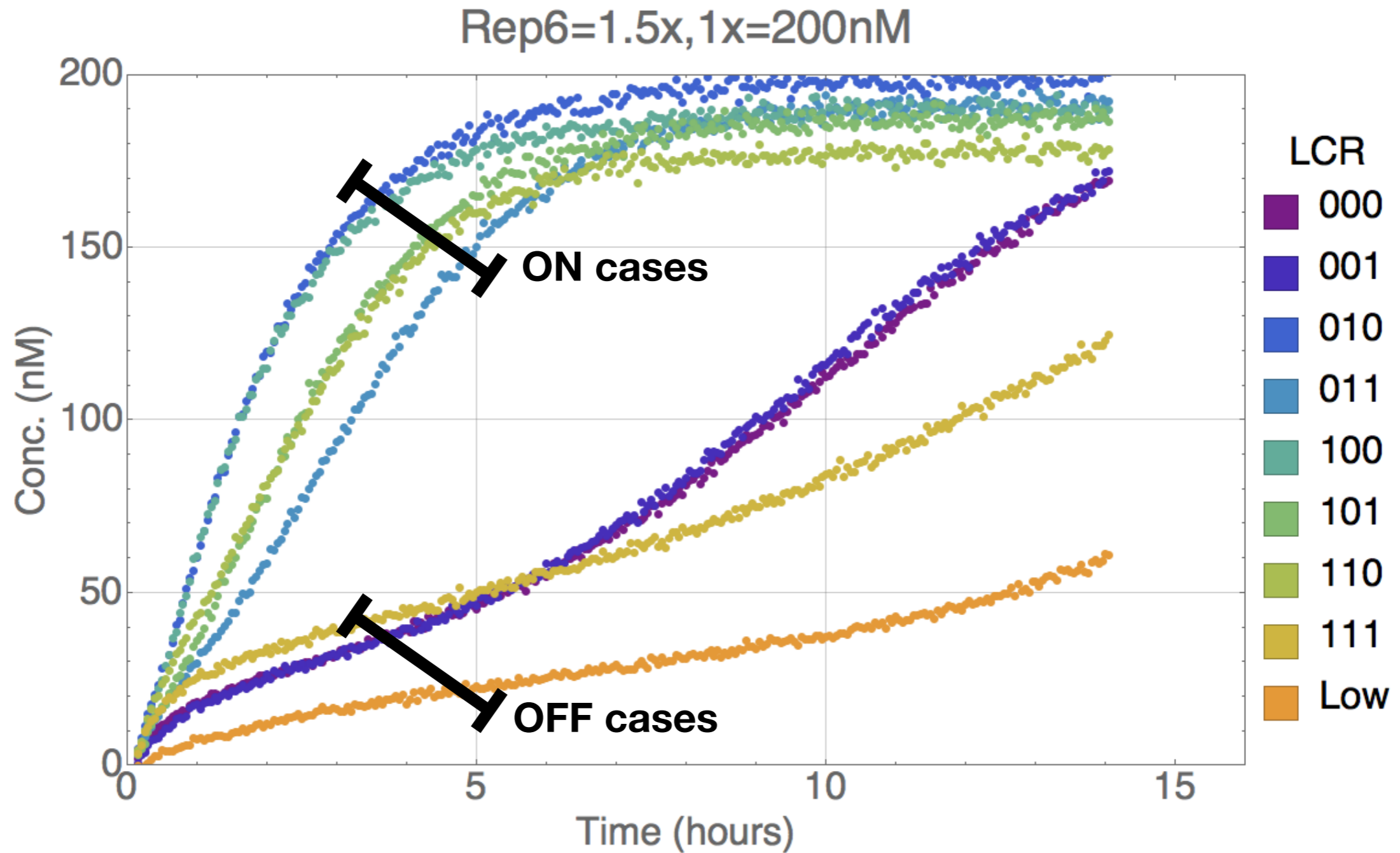
- Kinetic simulations of nucleic acids as random walk on thermodynamic energy model
- Supports multiple interacting strands
- Equilibrium consistent with [NUPACK](#)
- Various usage modes to study kinetic trajectories
- Distributed as a Python package
- MIT License



# Tutorial Outline

- ▶ Review of strand displacement
- ▶ Building and composing logic gates
- ▶ Tools for designing and verifying circuits
- ▶ **Robustness of strand displacement**

# Why is this circuit not *robust*?



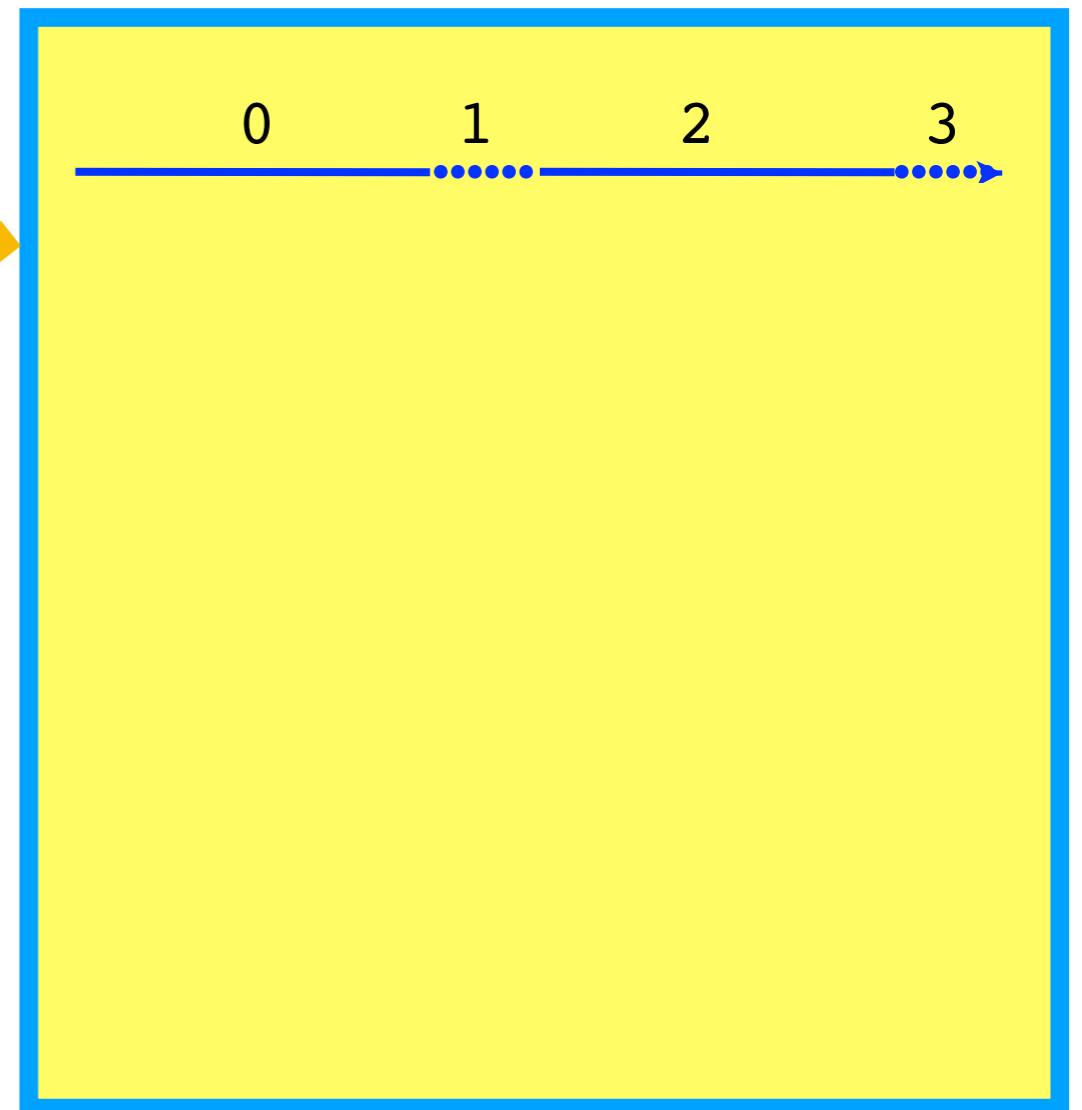
What causes signal *leak*?





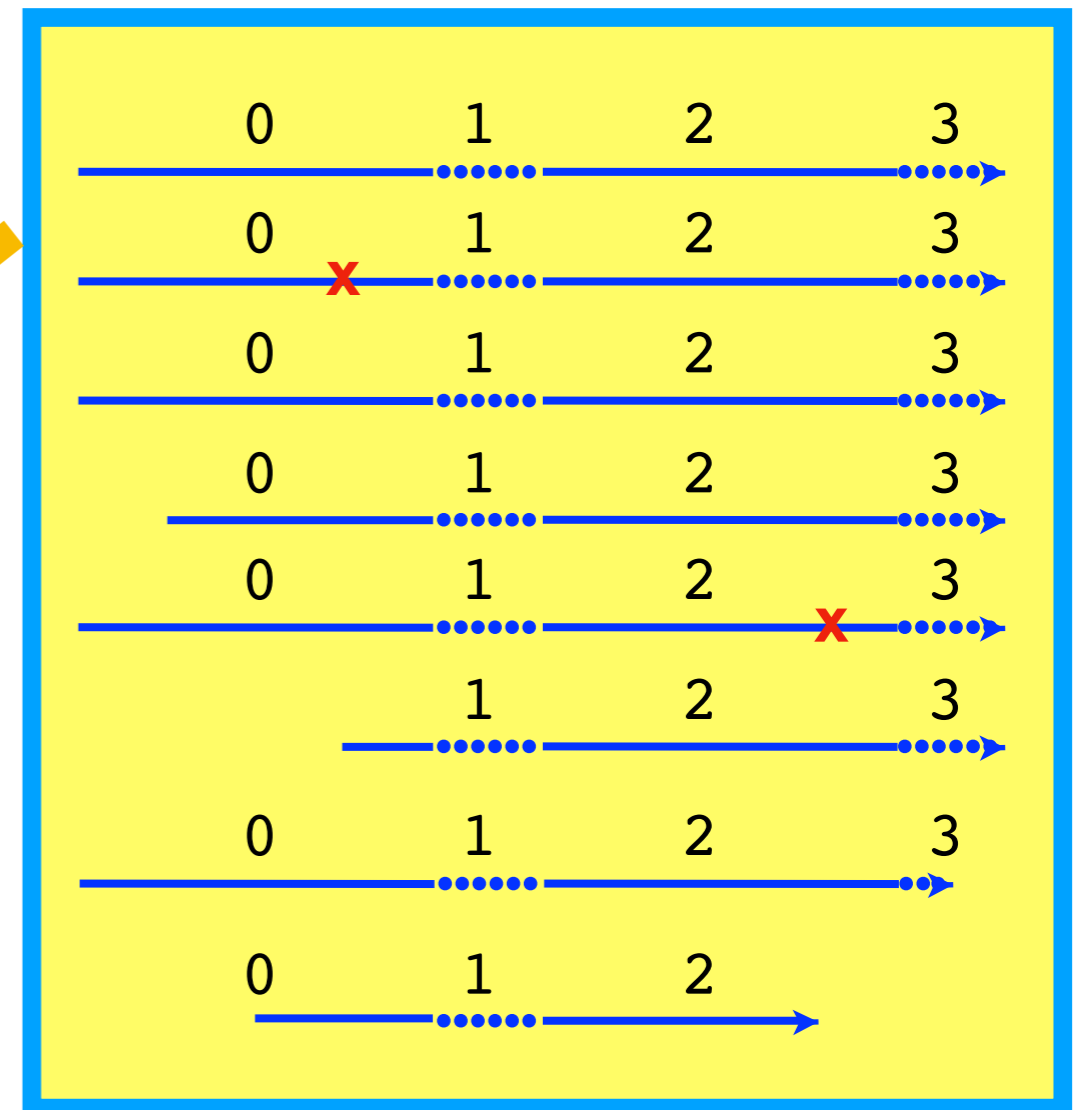
# Problem 1: Molecules are not perfect

**Imperfect strands from imperfect synthesis**



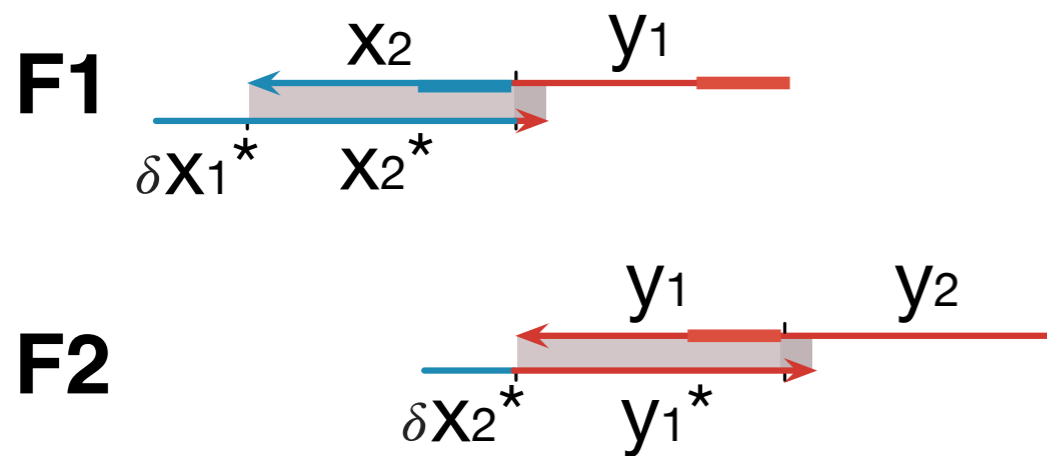
# Problem 1: Molecules are not perfect

## Imperfect strands from imperfect synthesis

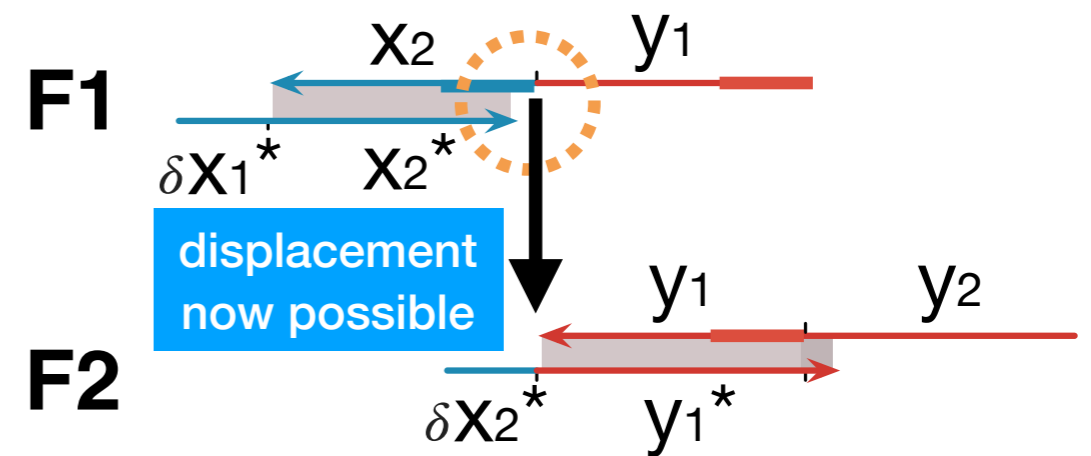


# Problem 1: Molecules are not perfect

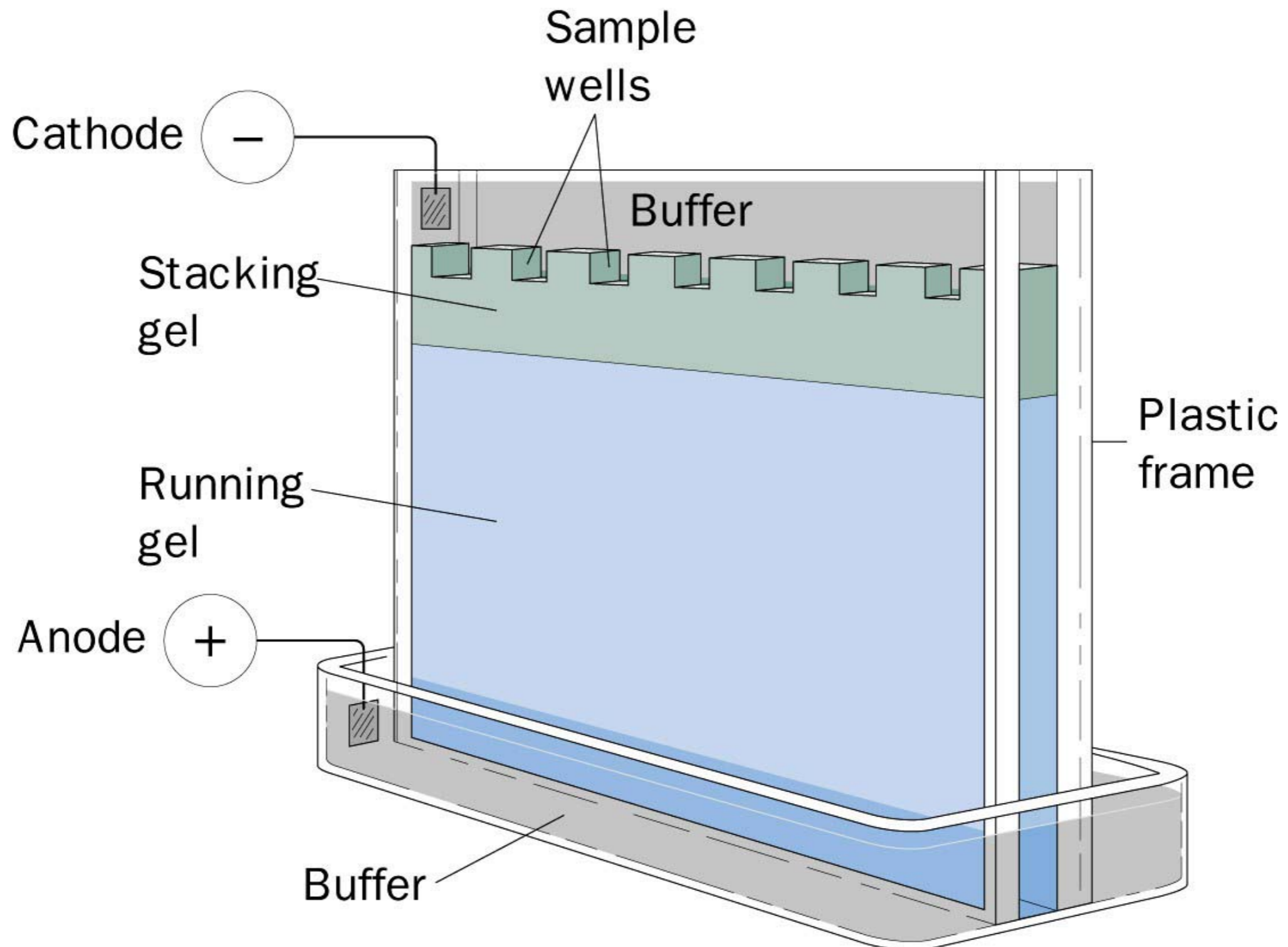
translator cascade  
with perfect molecules



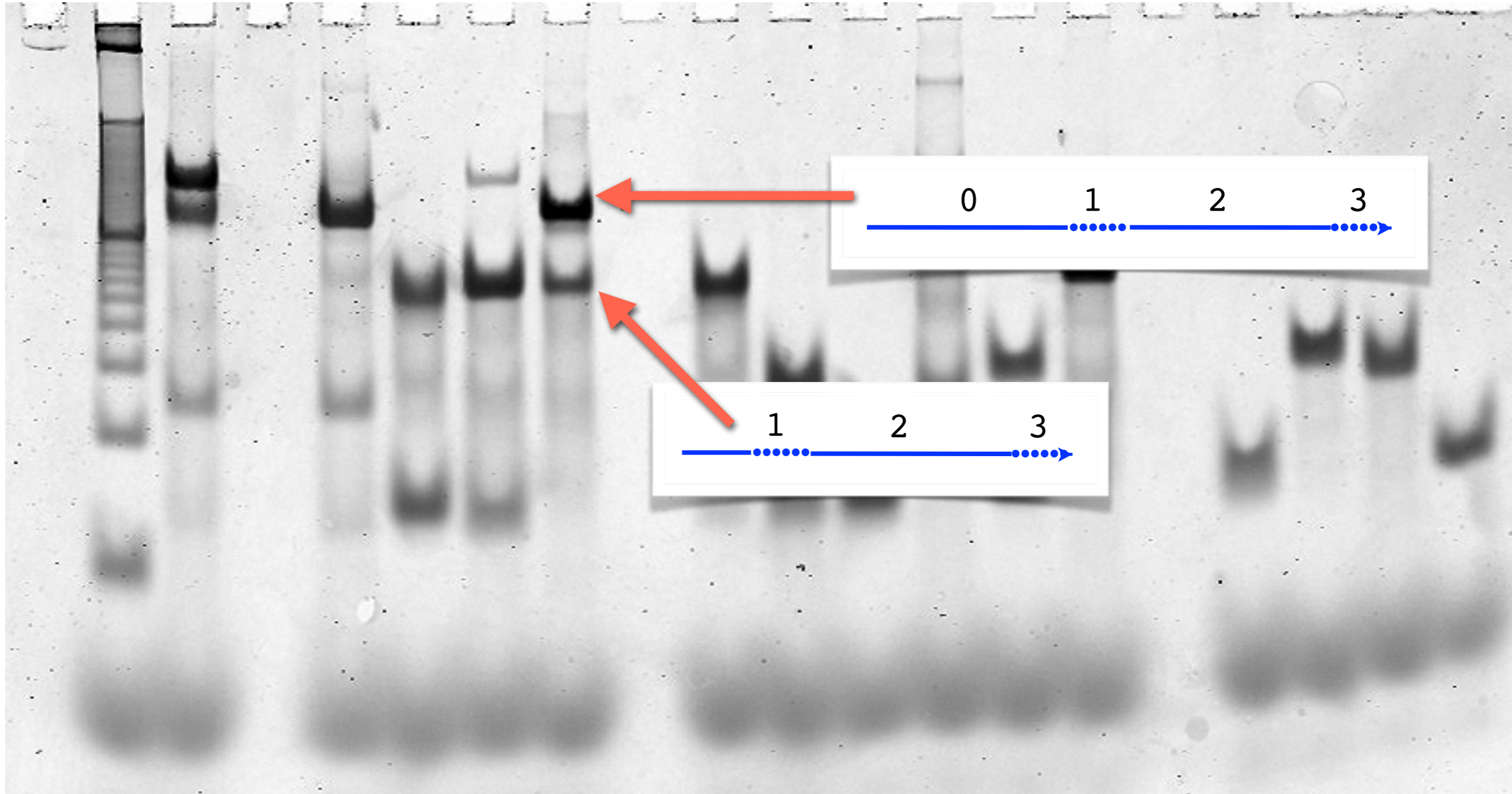
translator cascade  
with imperfect molecules



# (Partial) solution to Problem 1

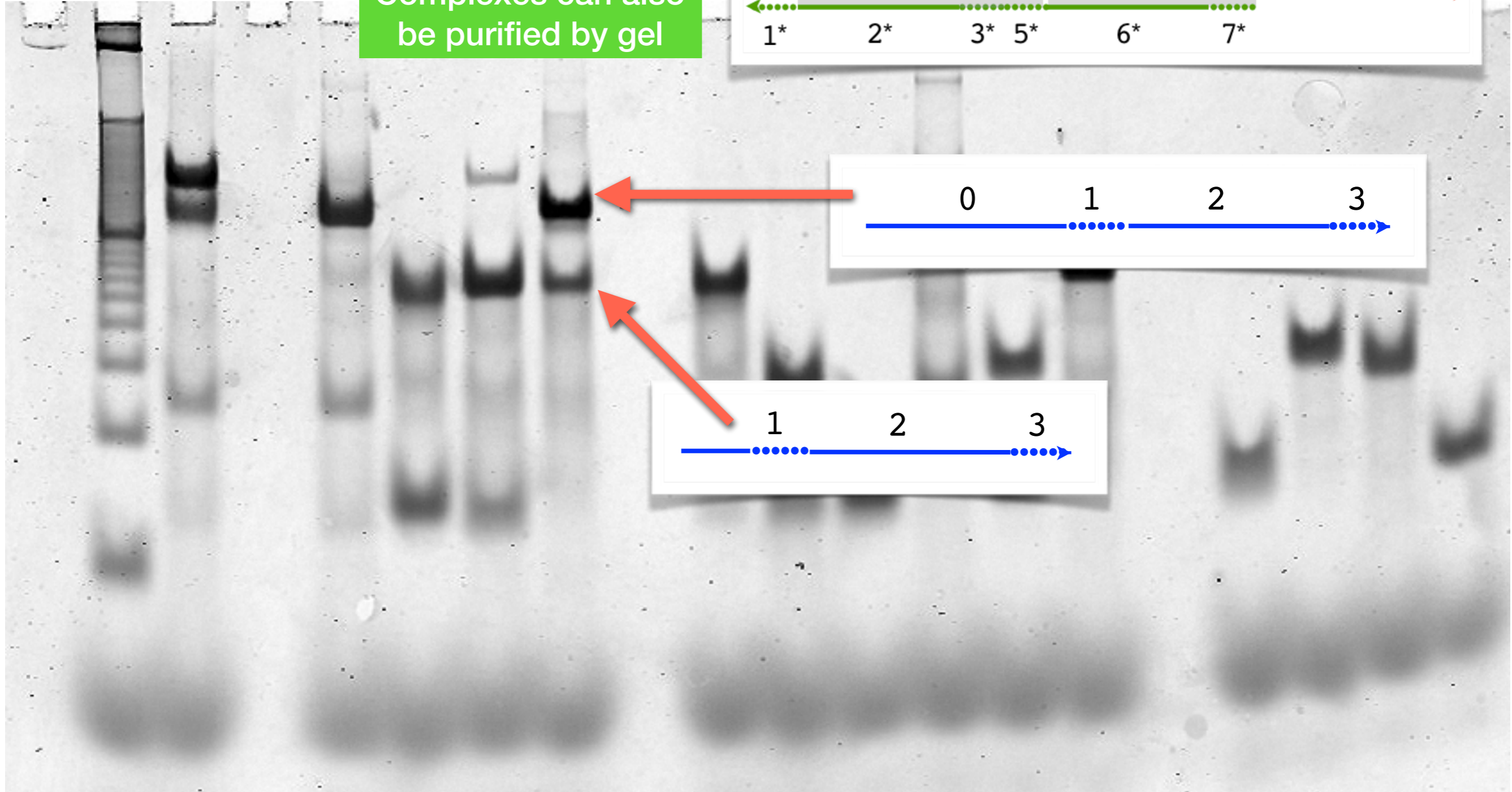
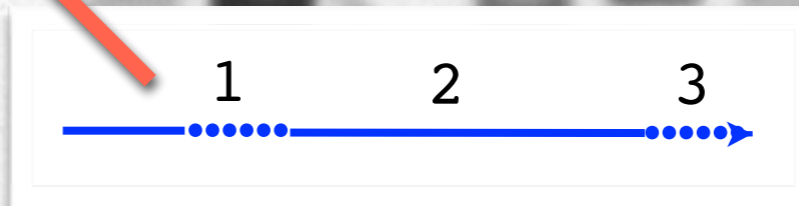
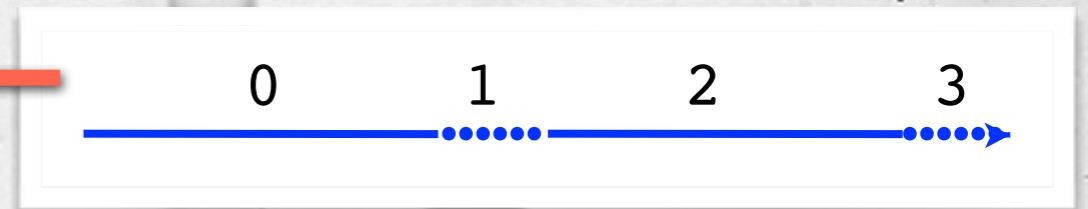


# (Partial) solution to Problem 1

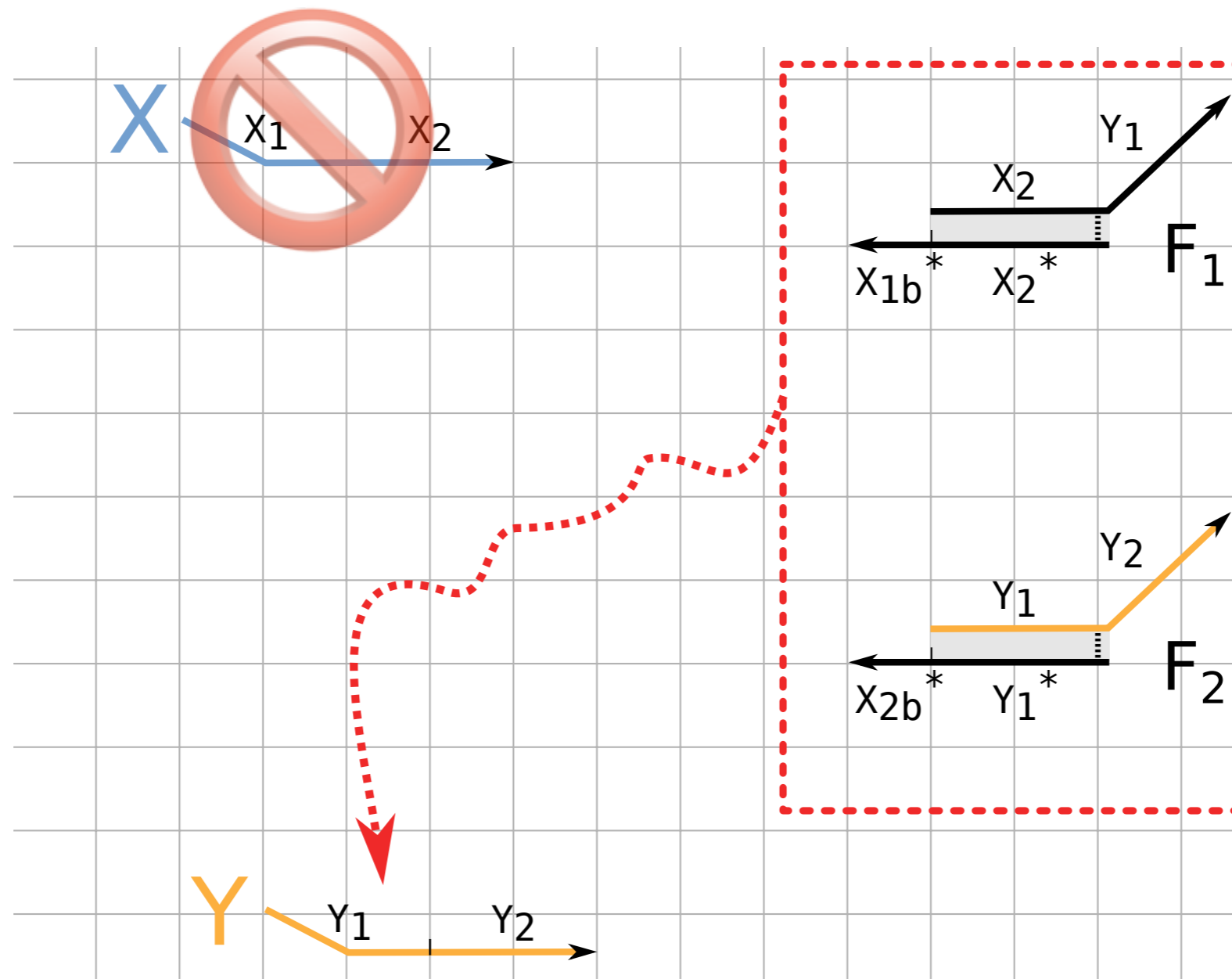
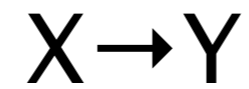


# (Partial) solution to Problem 1

Complexes can also be purified by gel

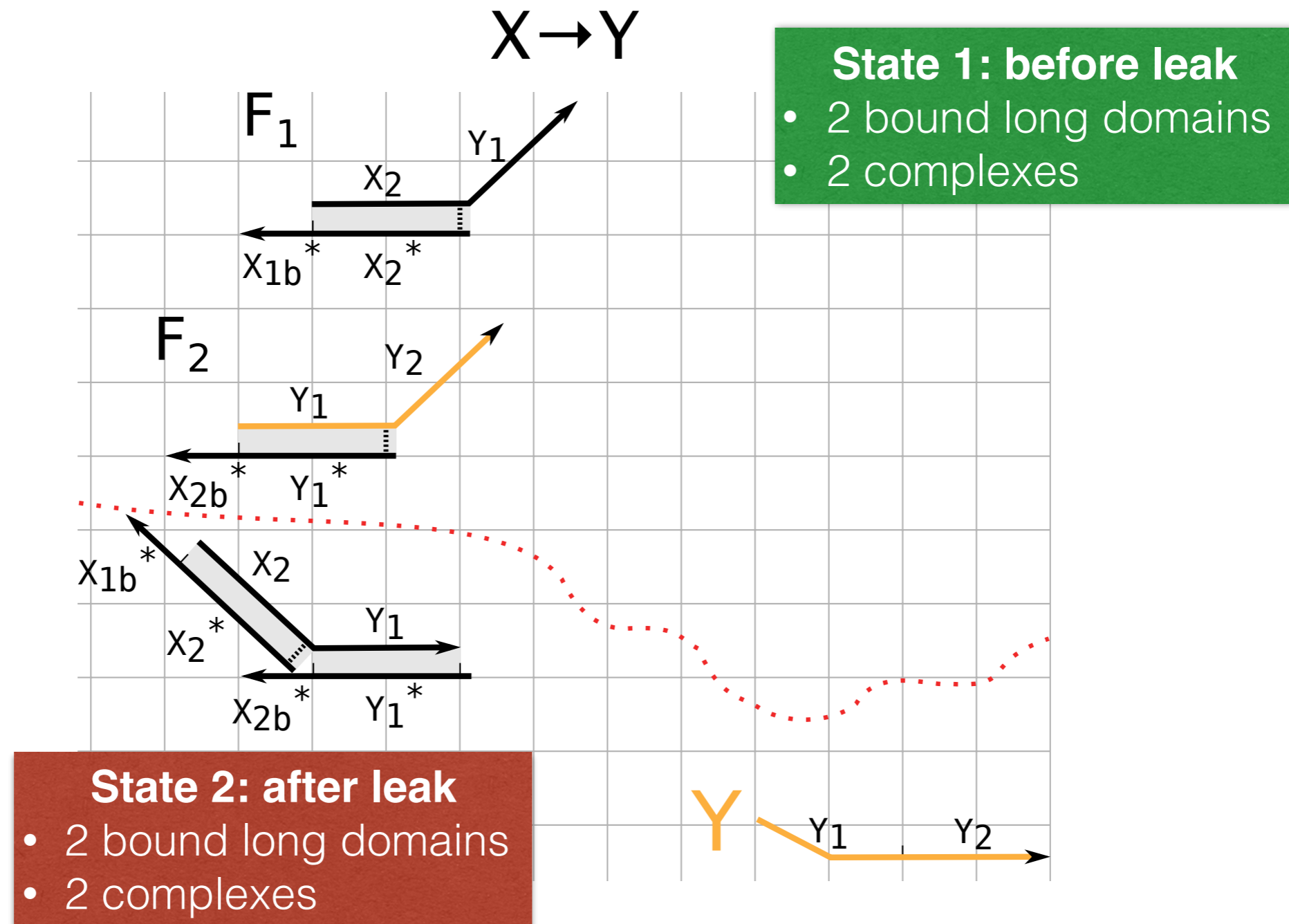


# Problem 2: Spurious reactions occur (even with perfect molecules)



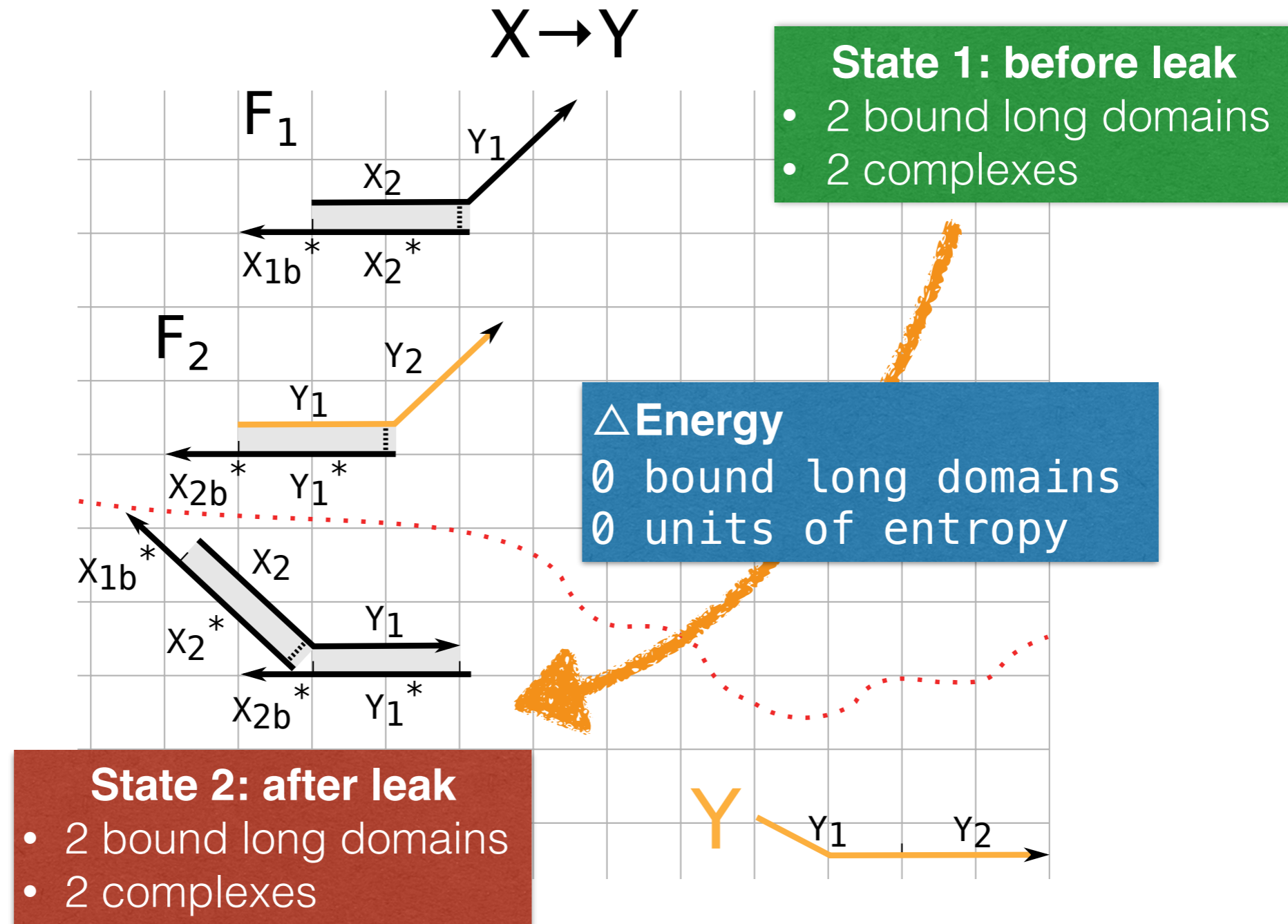
Y has been spuriously “produced”

# Some rough energy accounting





# Some rough energy accounting



# A Motivating Question

Can we rationally design  
*composable, leakless*  
DSD gates?

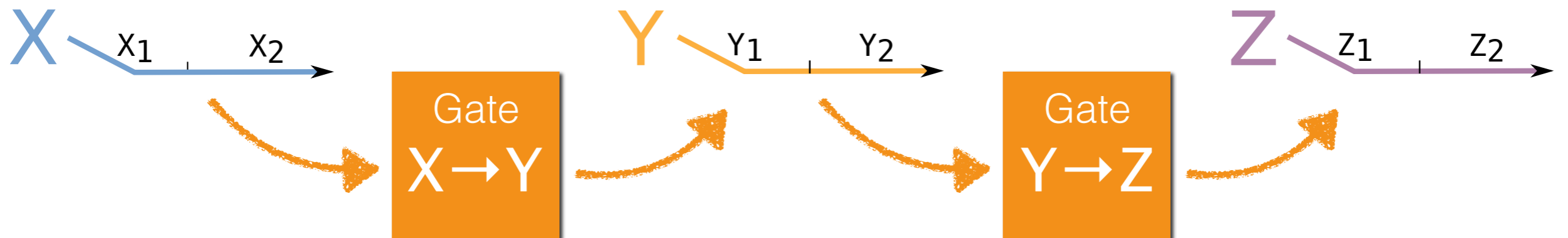
# A Motivating Question

Can we rationally design  
composable, leakless  
DSD gates?

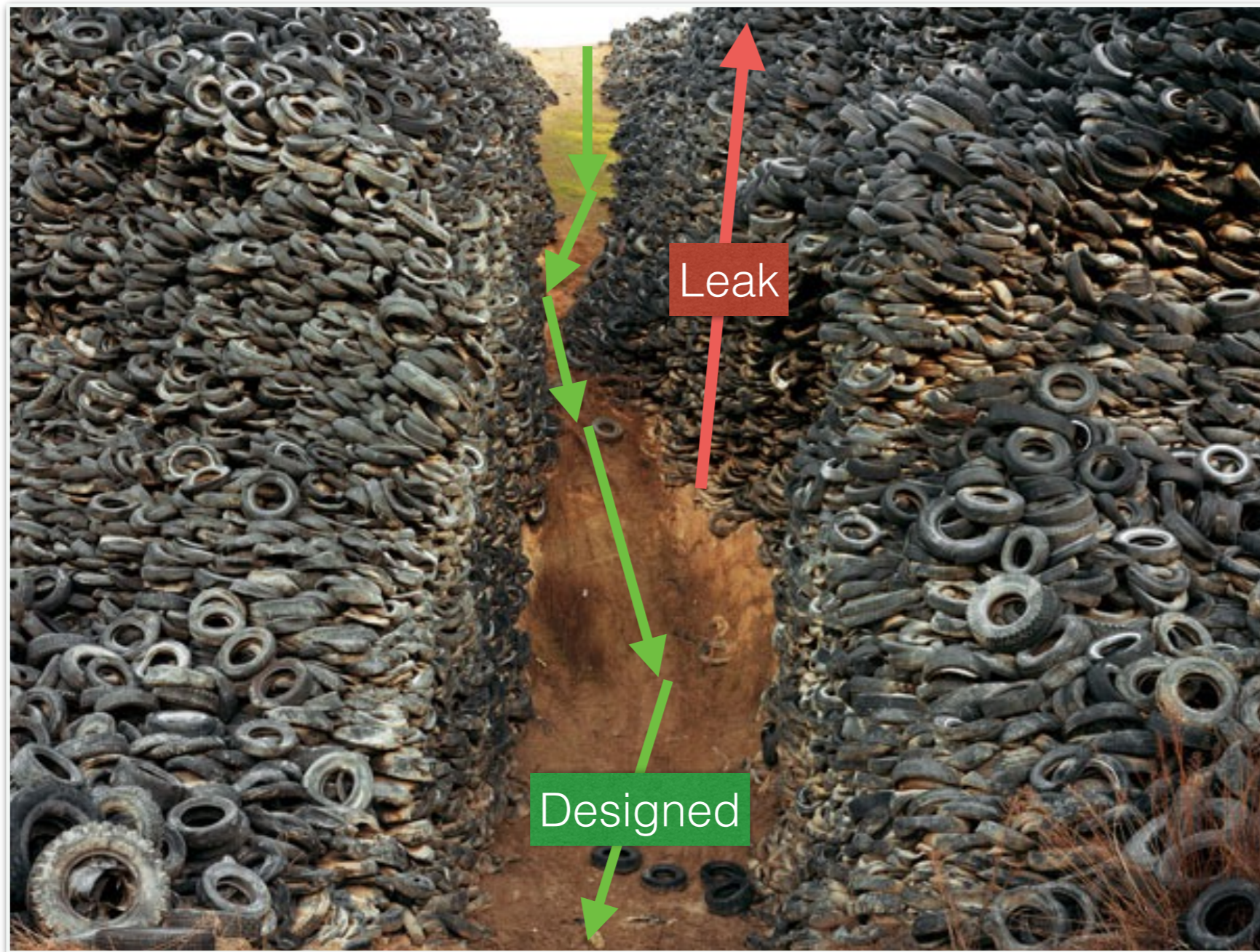


# A Motivating Question

Can we rationally design  
composable, leakless  
DSD gates?

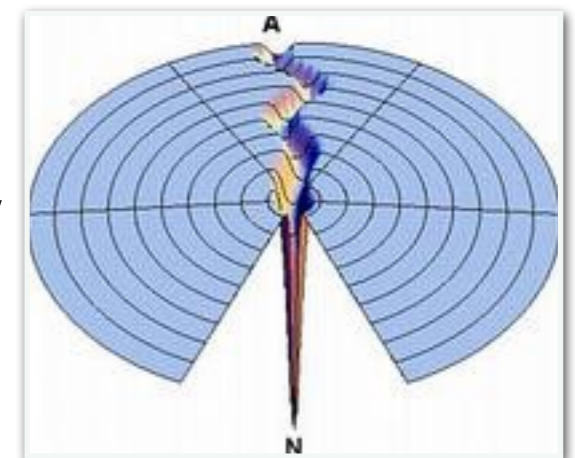


# What do we mean by *leakless*?



**“Golf funnel with deep groove” pathway**

K. Dill & Bromberg (2002). *Molecular Driving Forces*.



# (Partial) solution to Problem 2

For a redundancy parameter  $\mathbf{N}$ , there exist **translator** and **AND** gates using  $\mathbf{N}$  long domains that have the following property:

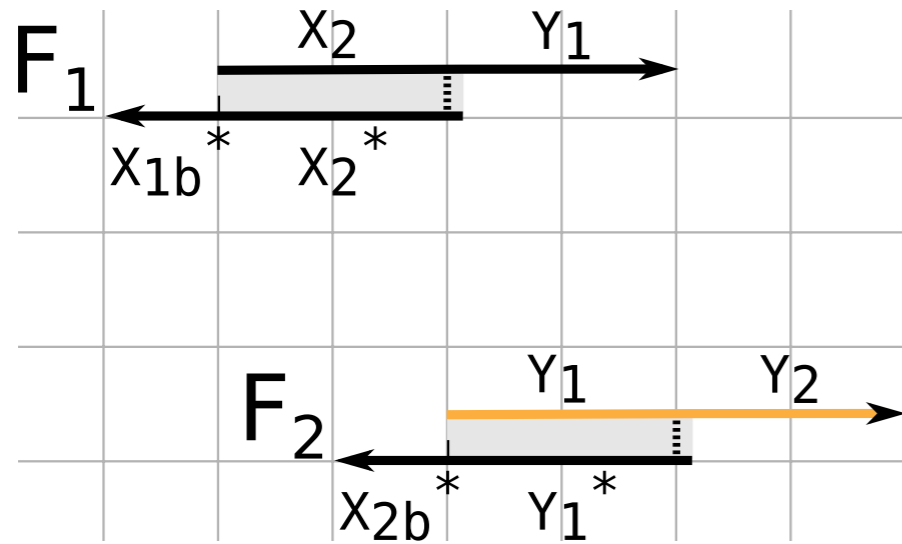
*even at thermodynamic equilibrium,*

the net leak decreases exponentially with  $\mathbf{N}$ .

Thachuk, Winfree, David Soloveichik. (2015)  
*Leakless DNA strand displacement.* DNA 21.



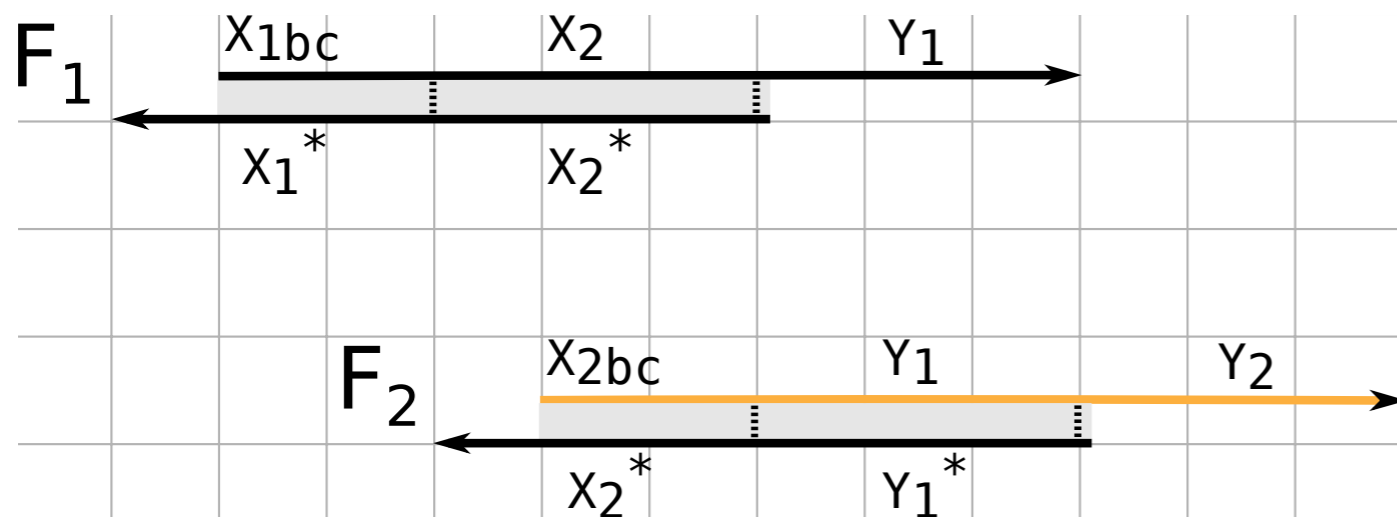
## Typical translator using “Single Long Domain” (SLD)



- Designed pathways: bimolecular
- Leak pathways: bimolecular

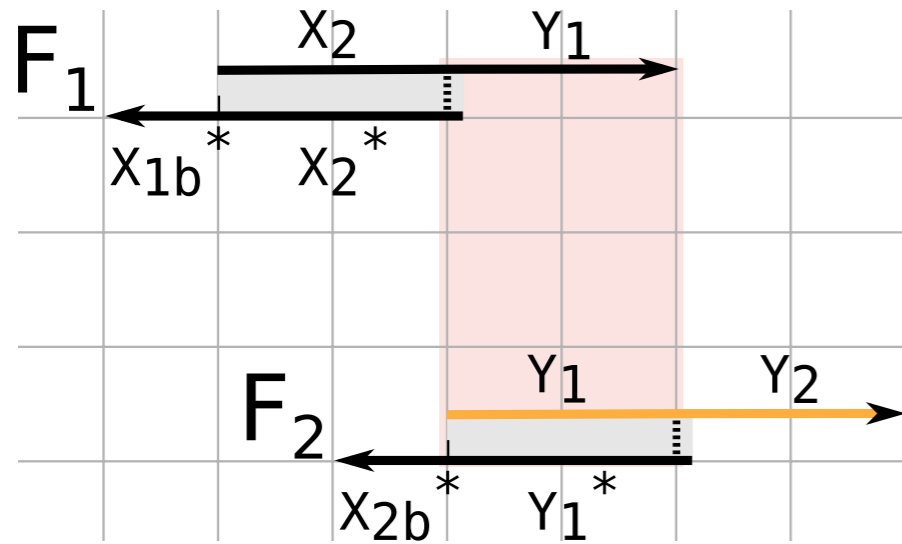
---

## DLD translator using “Double Long Domain” (DLD)



- Designed pathways: bimolecular
- Leak pathways: **trimolecular**

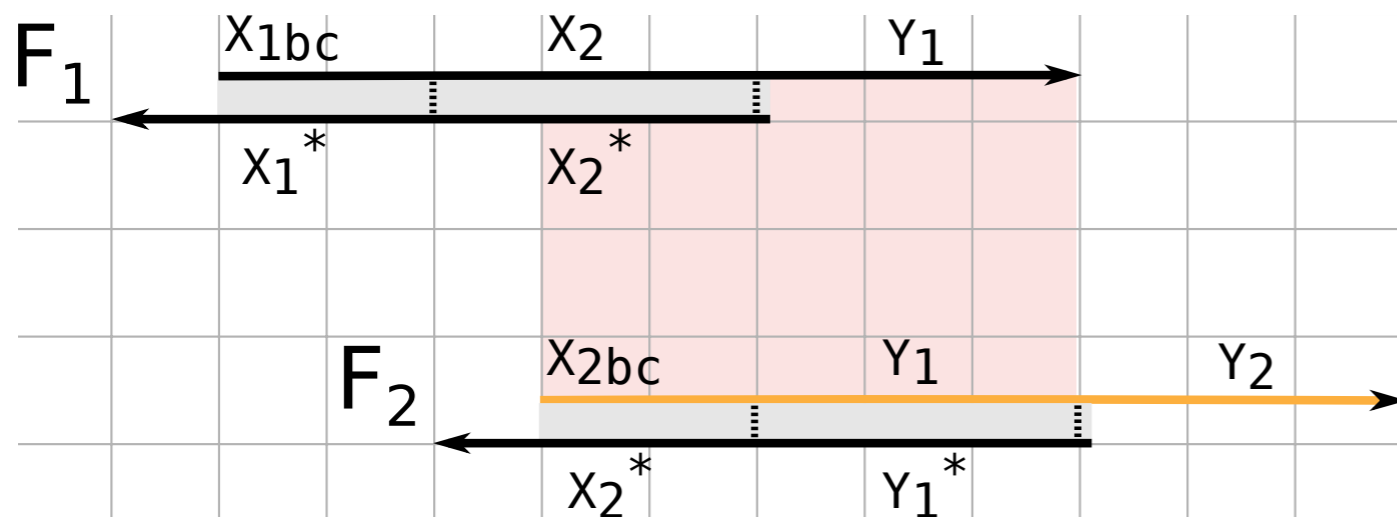
## Typical translator using “Single Long Domain” (SLD)



- Designed pathways: bimolecular
- Leak pathways: bimolecular

---

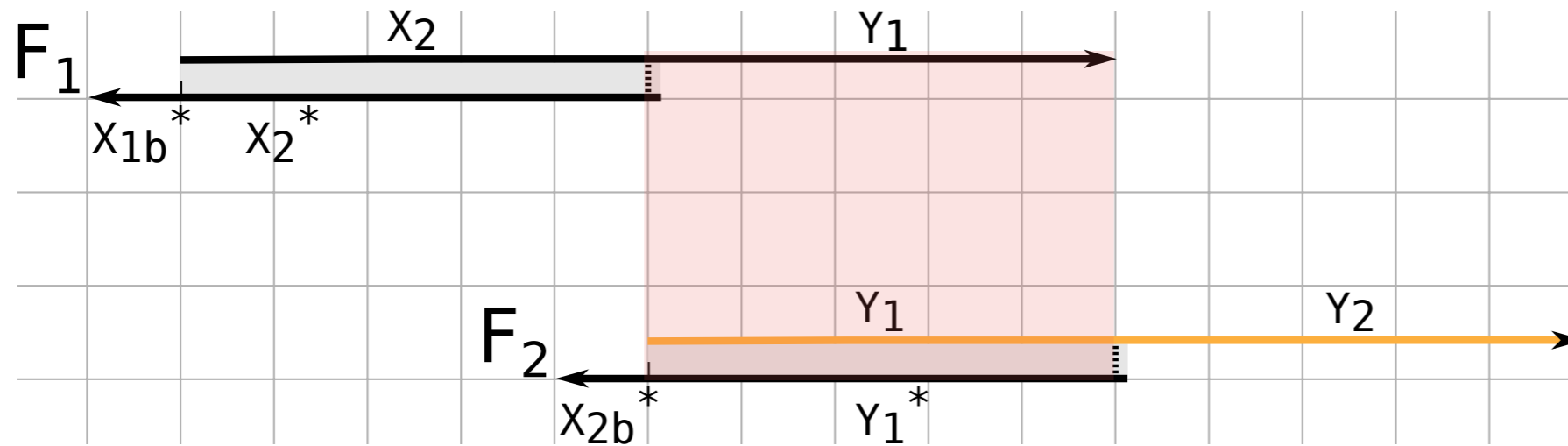
## DLD translator using “Double Long Domain” (DLD)



- Designed pathways: bimolecular
- Leak pathways: **trimolecular**



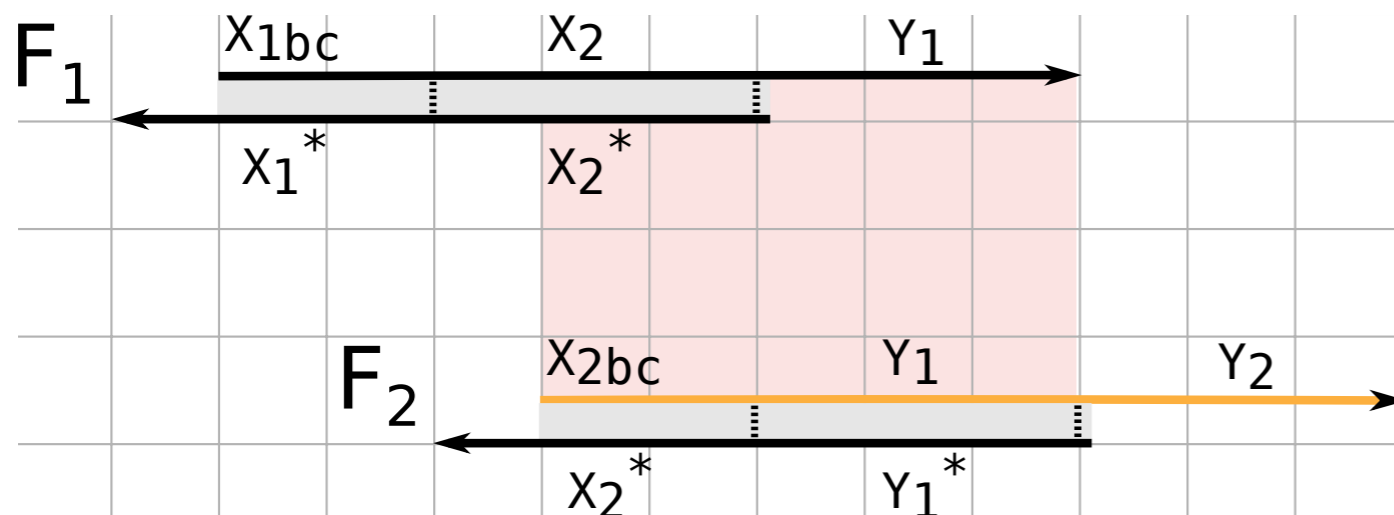
## Typical translator using “Single Long Domain” (SLD)



- Designed pathways: bimolecular
- Leak pathways: bimolecular

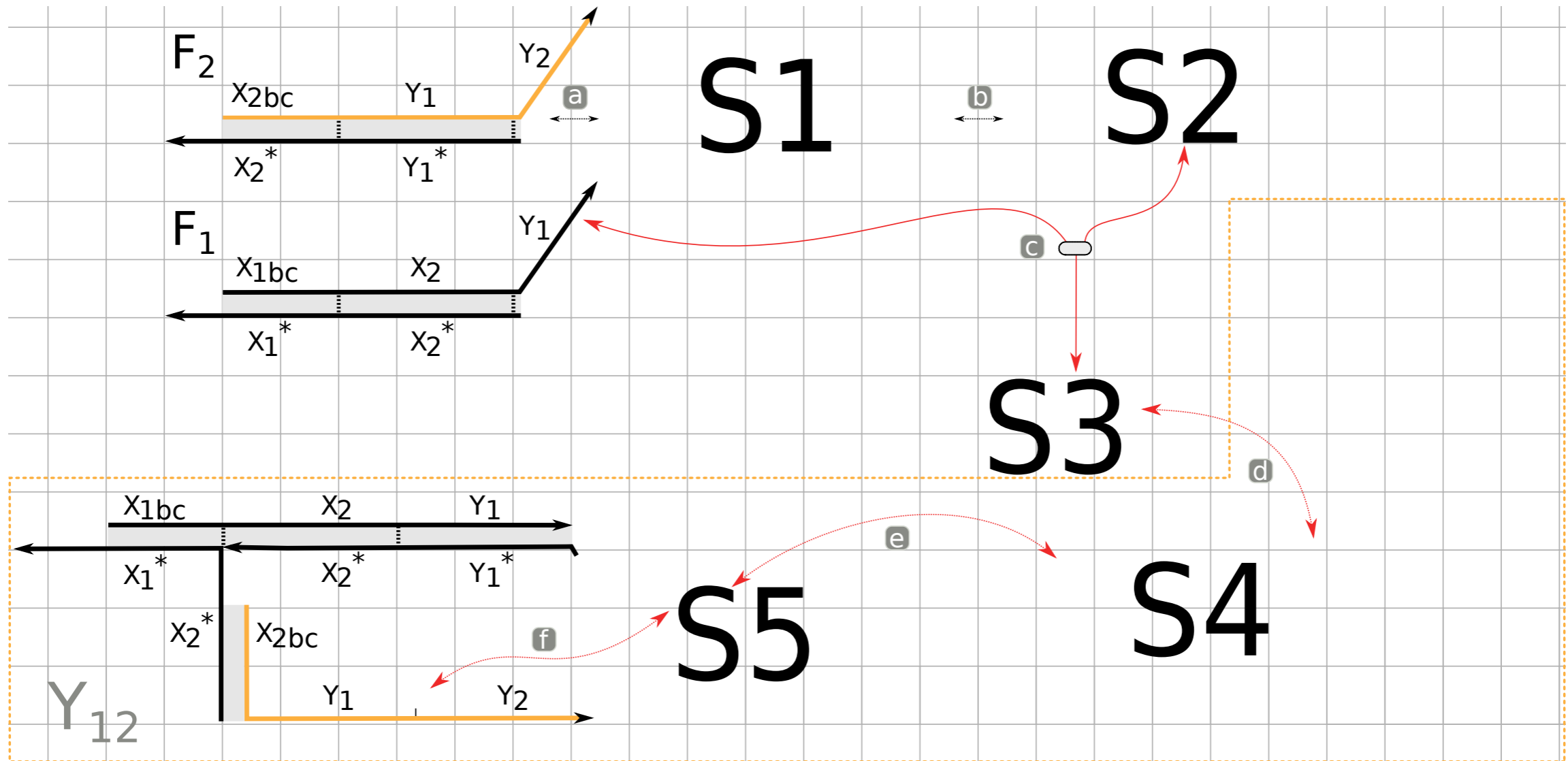
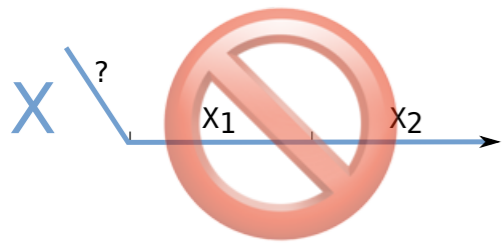
Lengthening recognition domains  
does not help

## DLD translator using “Double Long Domain” (DLD)

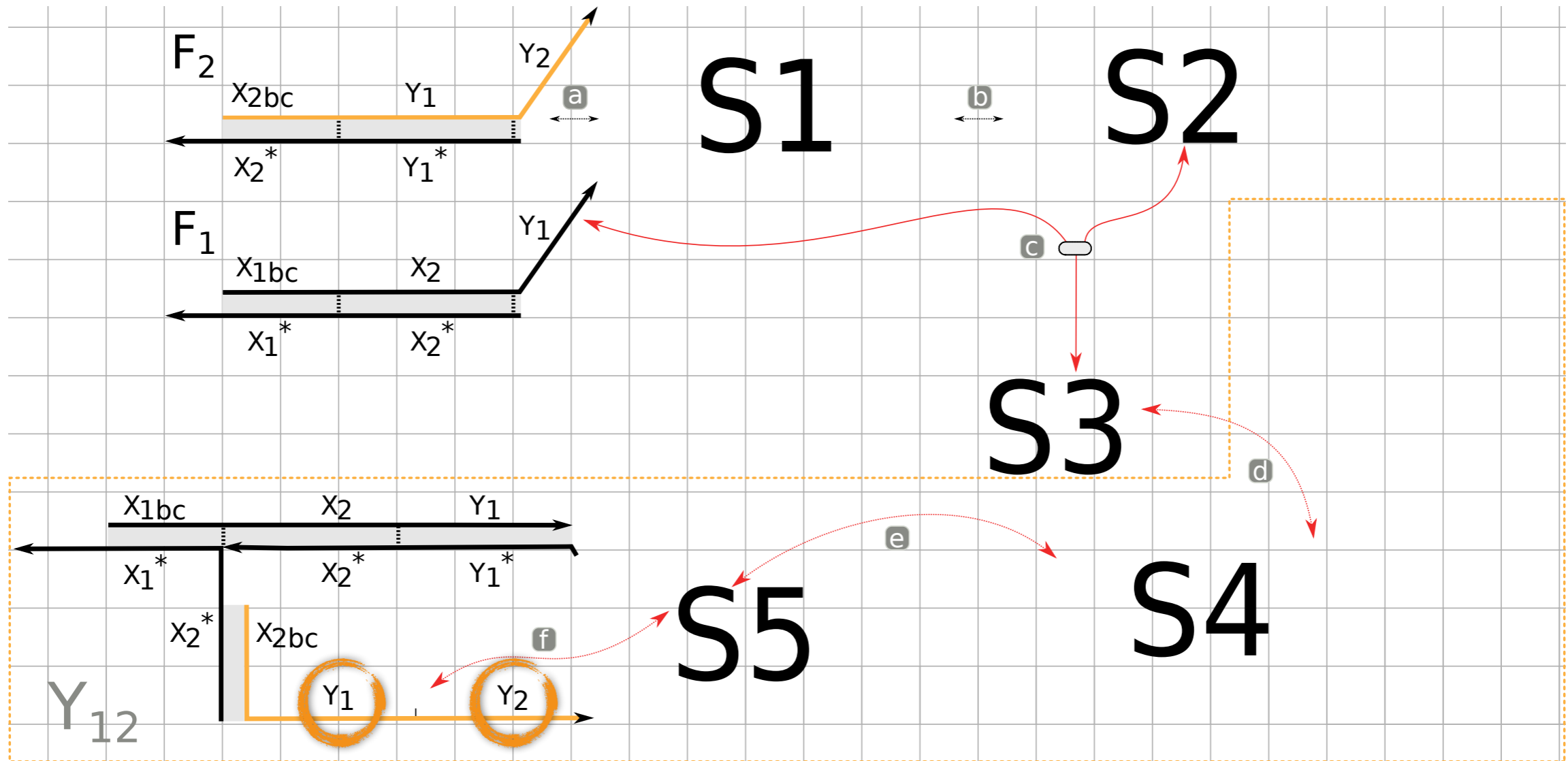
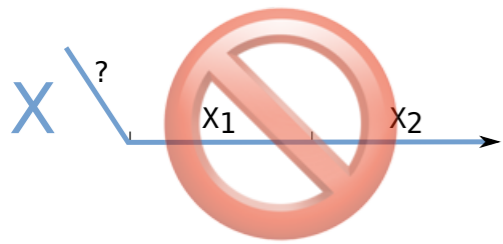


- Designed pathways: bimolecular
- Leak pathways: **trimolecular**

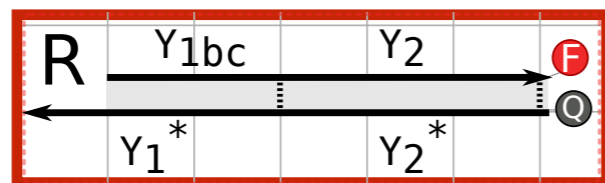
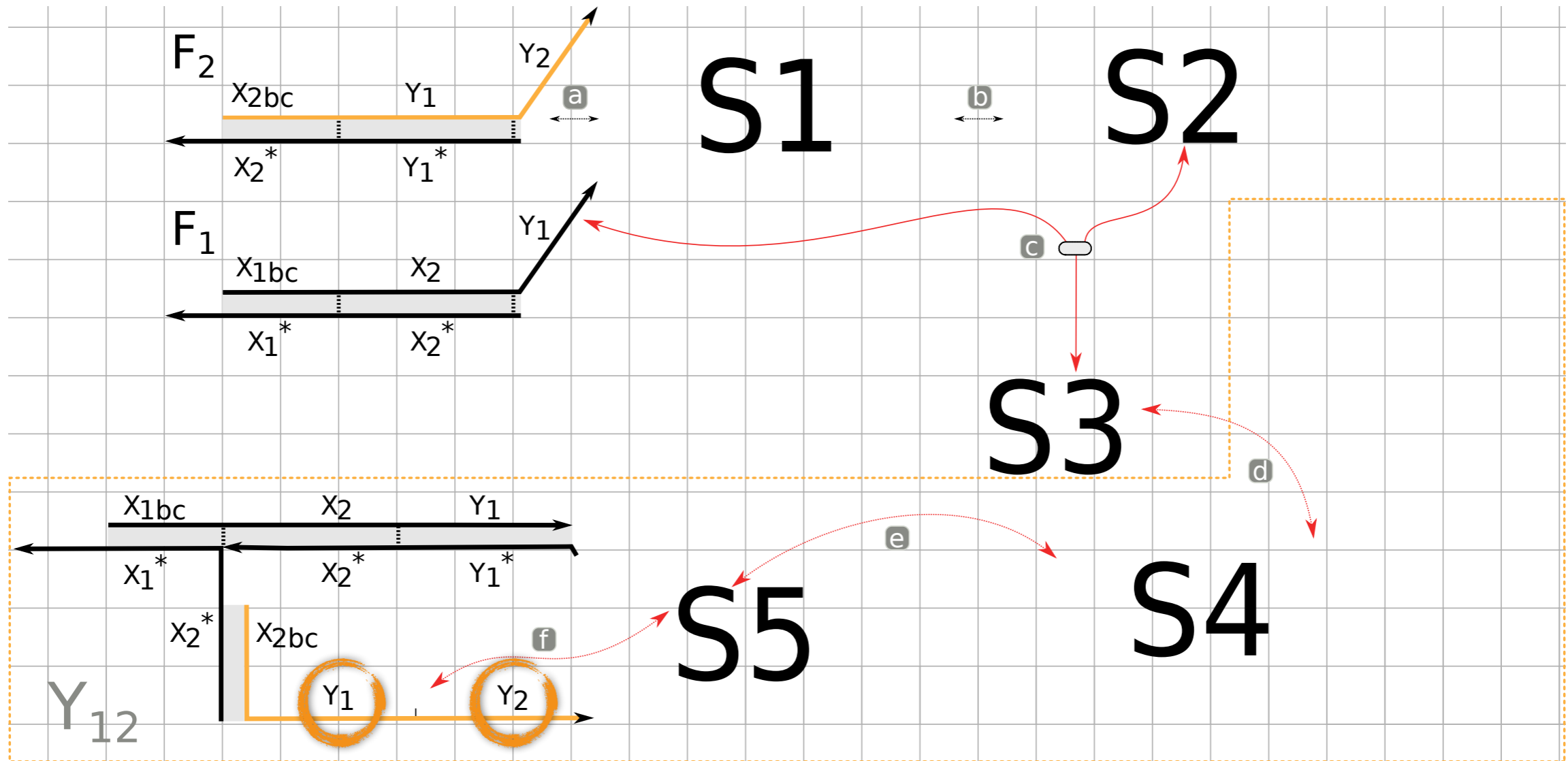
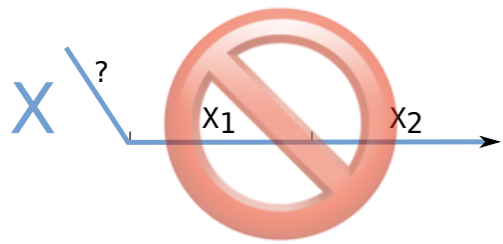
# DLD translators are intrinsically less “leaky”



# DLD translators are intrinsically less “leaky”

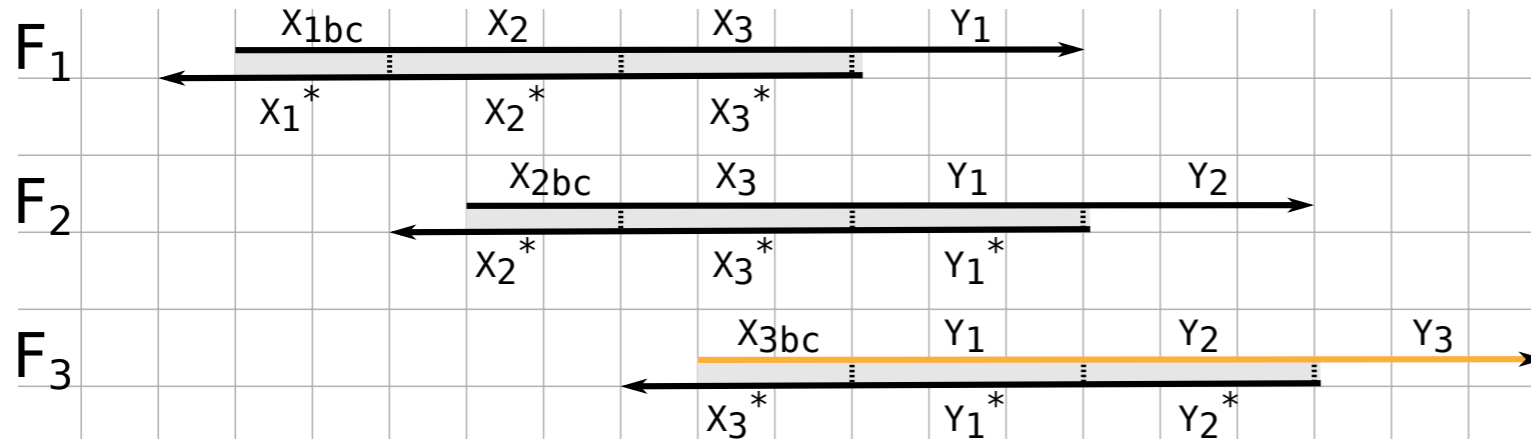


# DLD translators are intrinsically less “leaky”

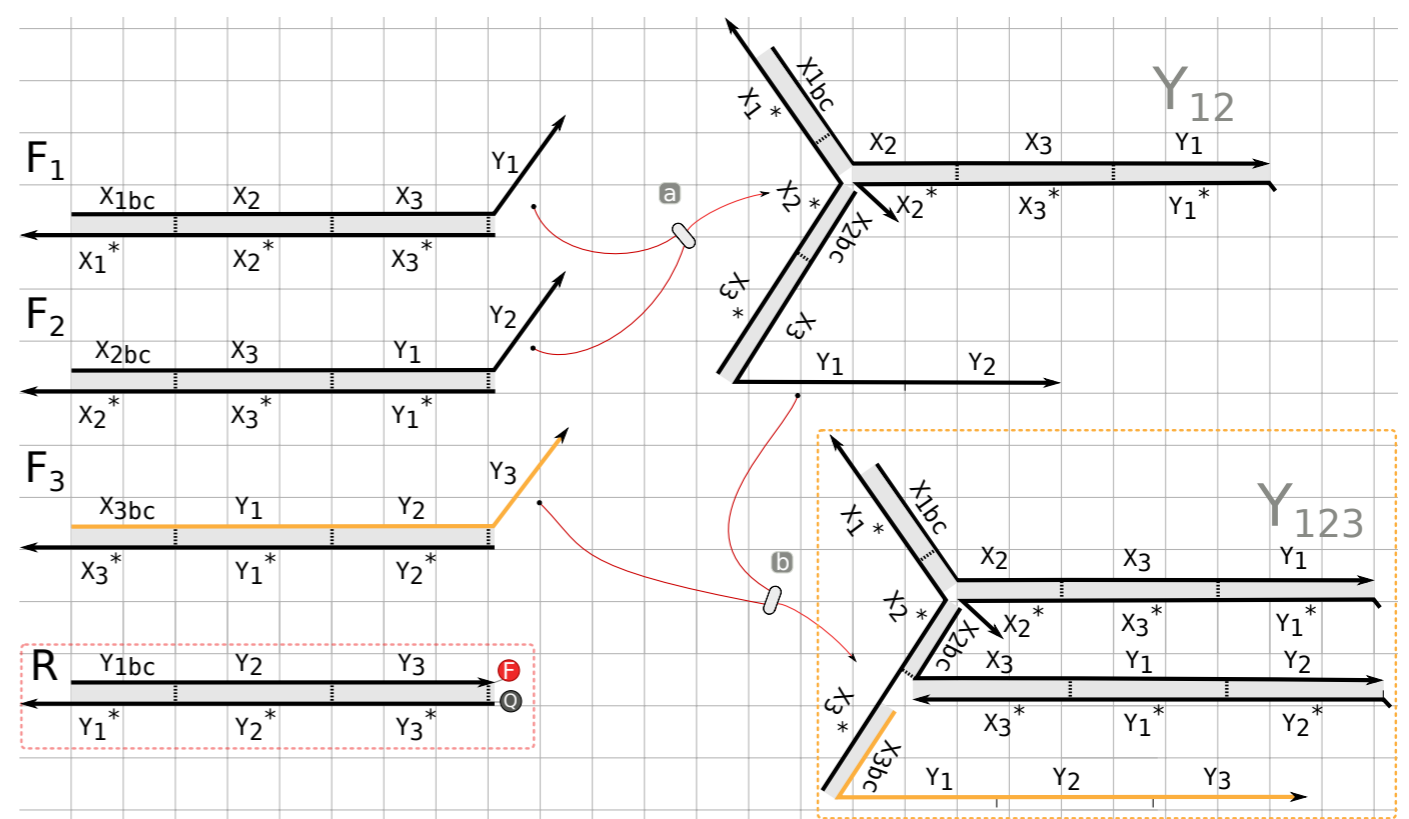


Can we generalize the  
DLD motif?

# Translator using Triple Long Domain (TLD) motif

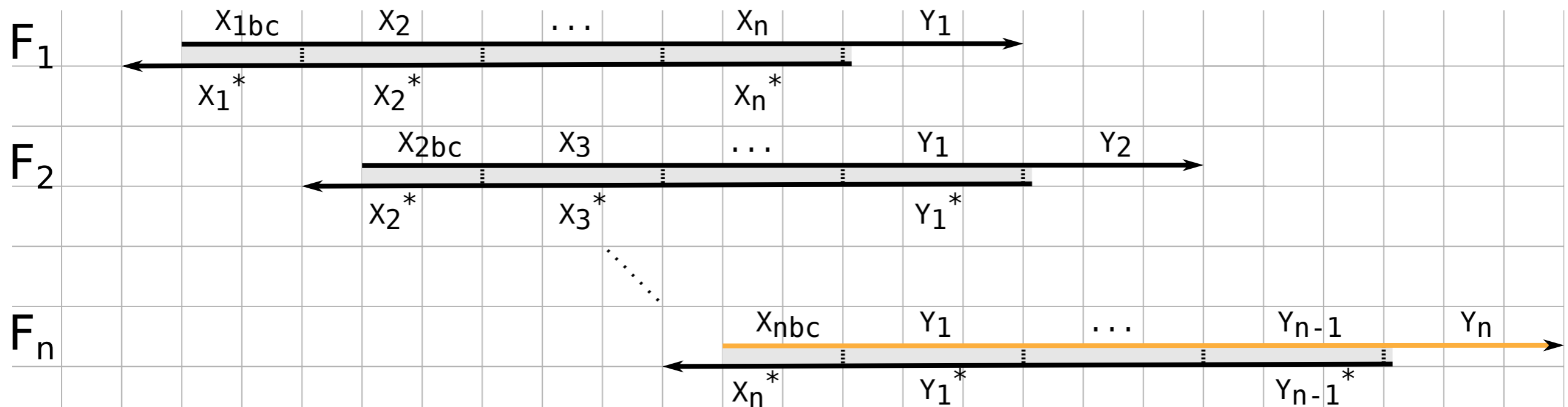


Three fuel complexes must combine to activate output signal.



$\Delta$ Energy  
 $0$  bound long domains  
 $-2$  units of entropy

# Translator using $N$ Long Domain (NLD) motif



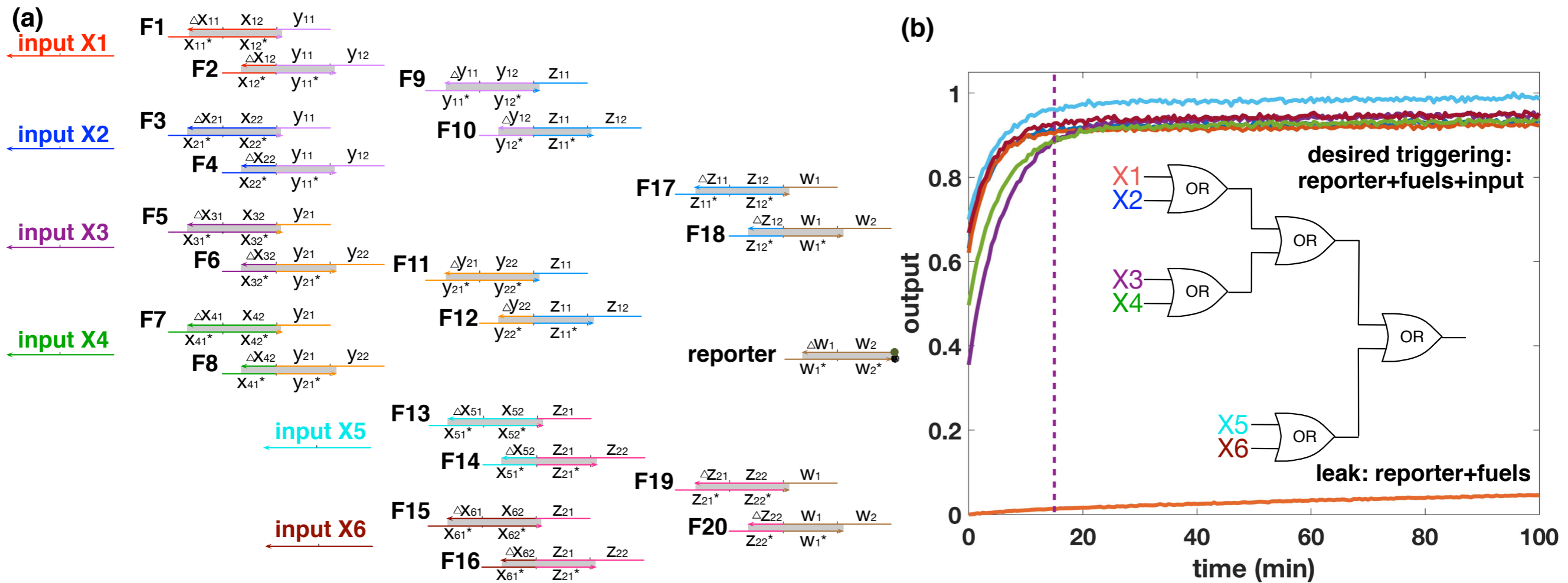
$N$  fuel complexes must combine to activate output signal.

$\Delta$ Energy to leak state

0 bound long domains  
 $-(N-1)$  units of entropy



# Building OR circuits from DLD translators

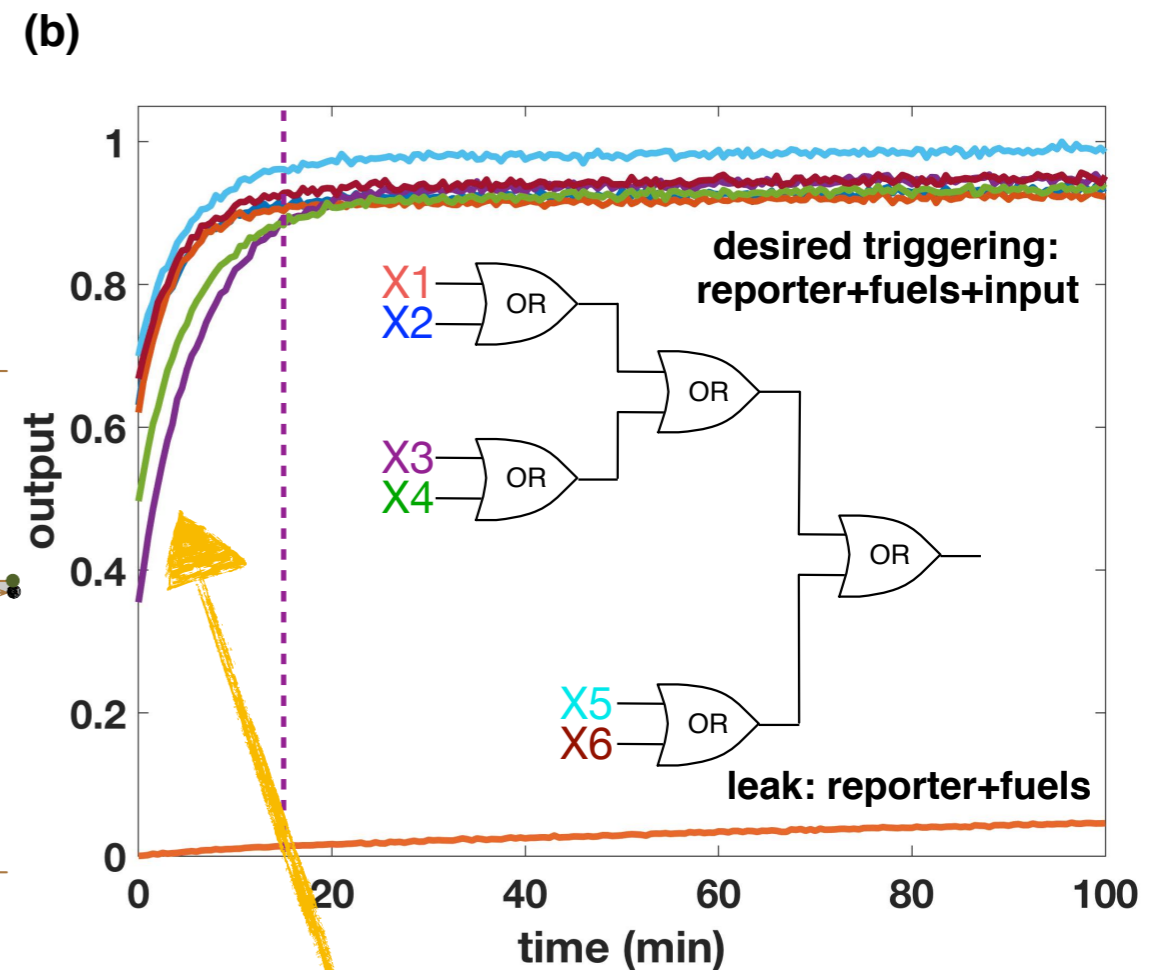
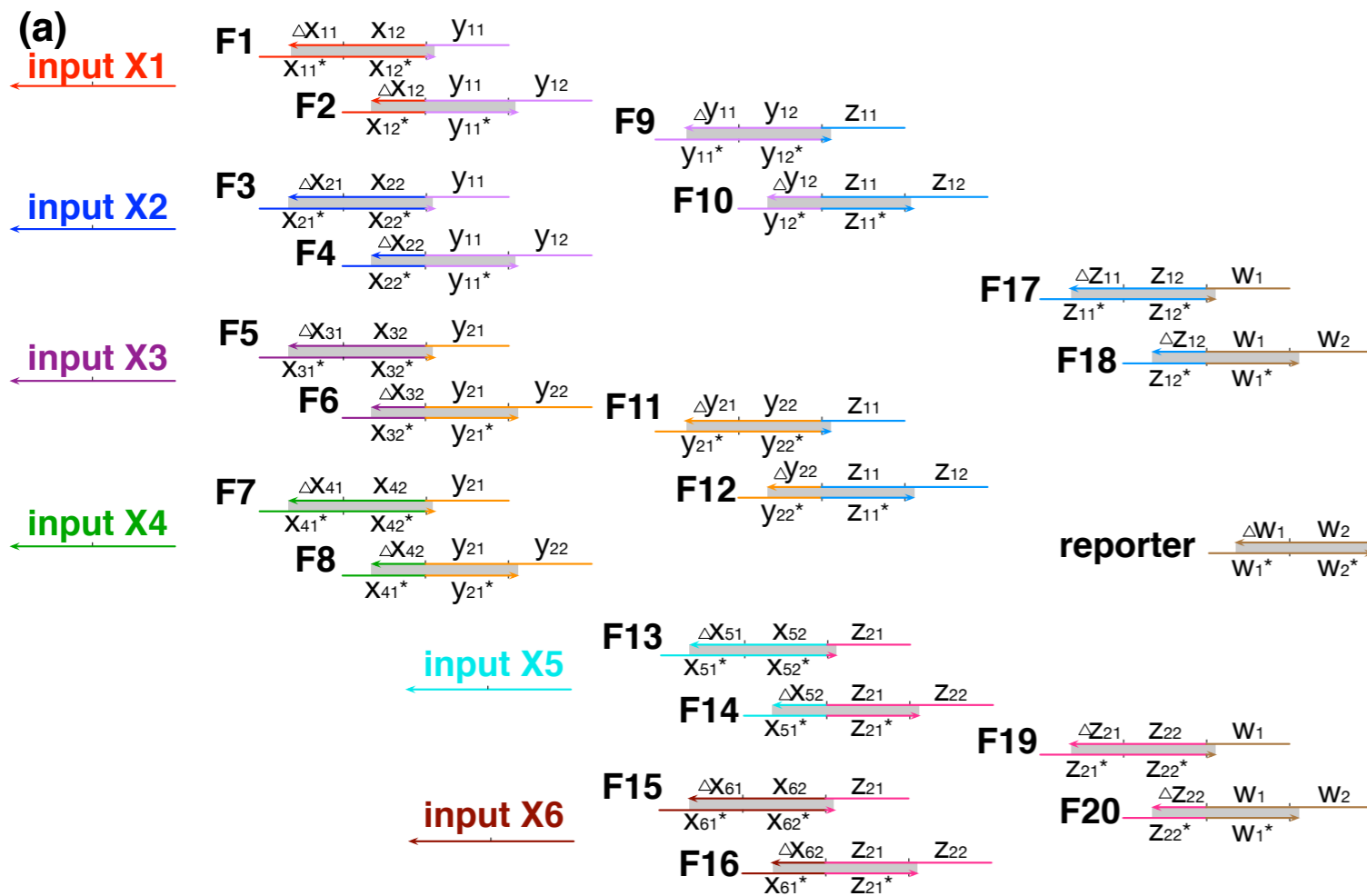


[fuel]=[input]=1000nM  
[reporter]=500nM





# Building OR circuits from DLD translators



Half-time completion ~6 minutes  
No signal restoration used.

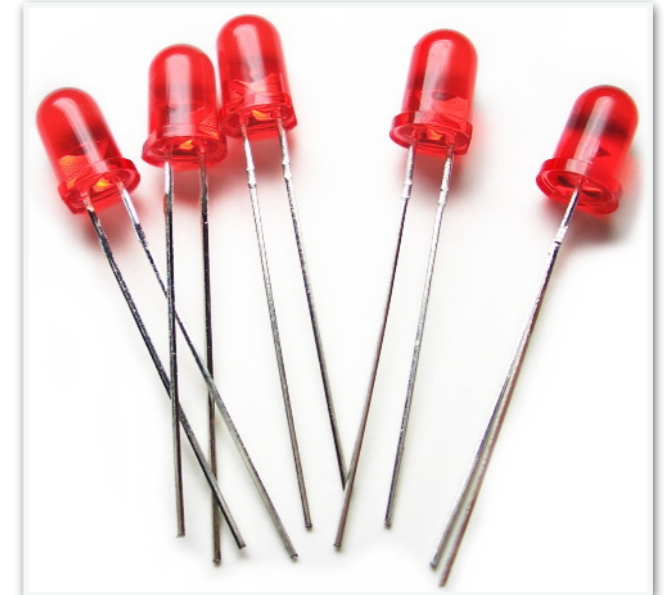
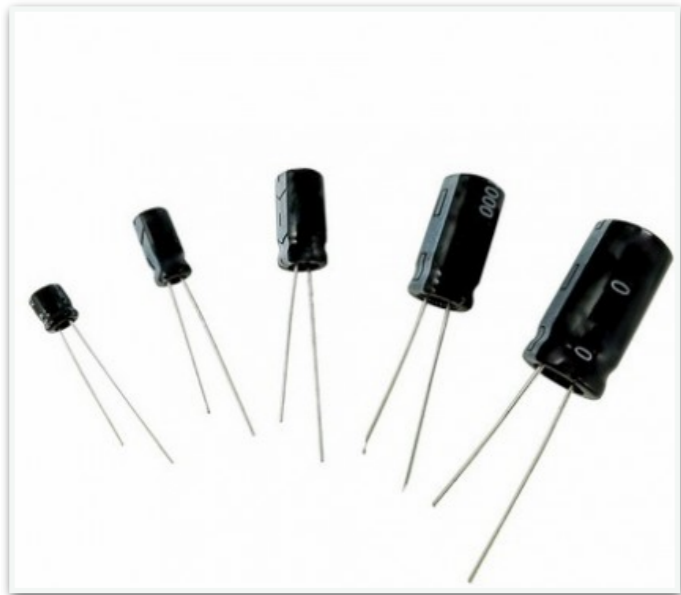
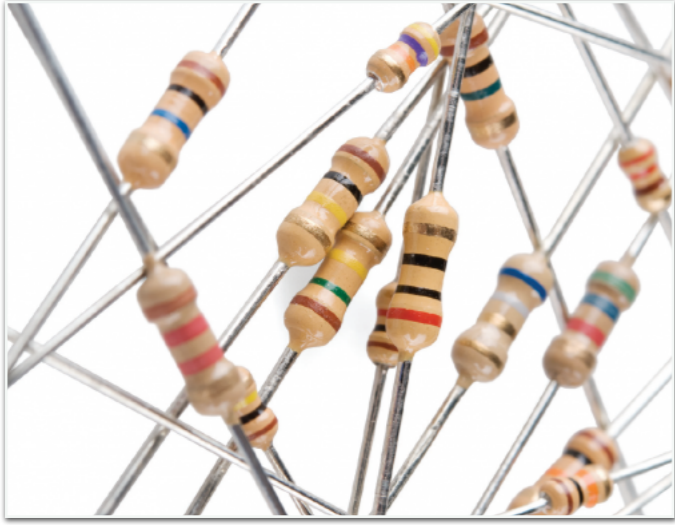
[fuel]=[input]=1000nM  
[reporter]=500nM

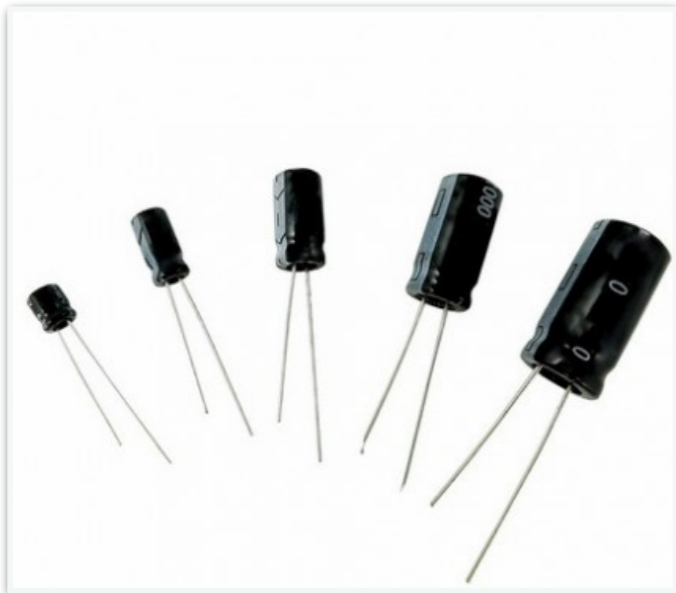
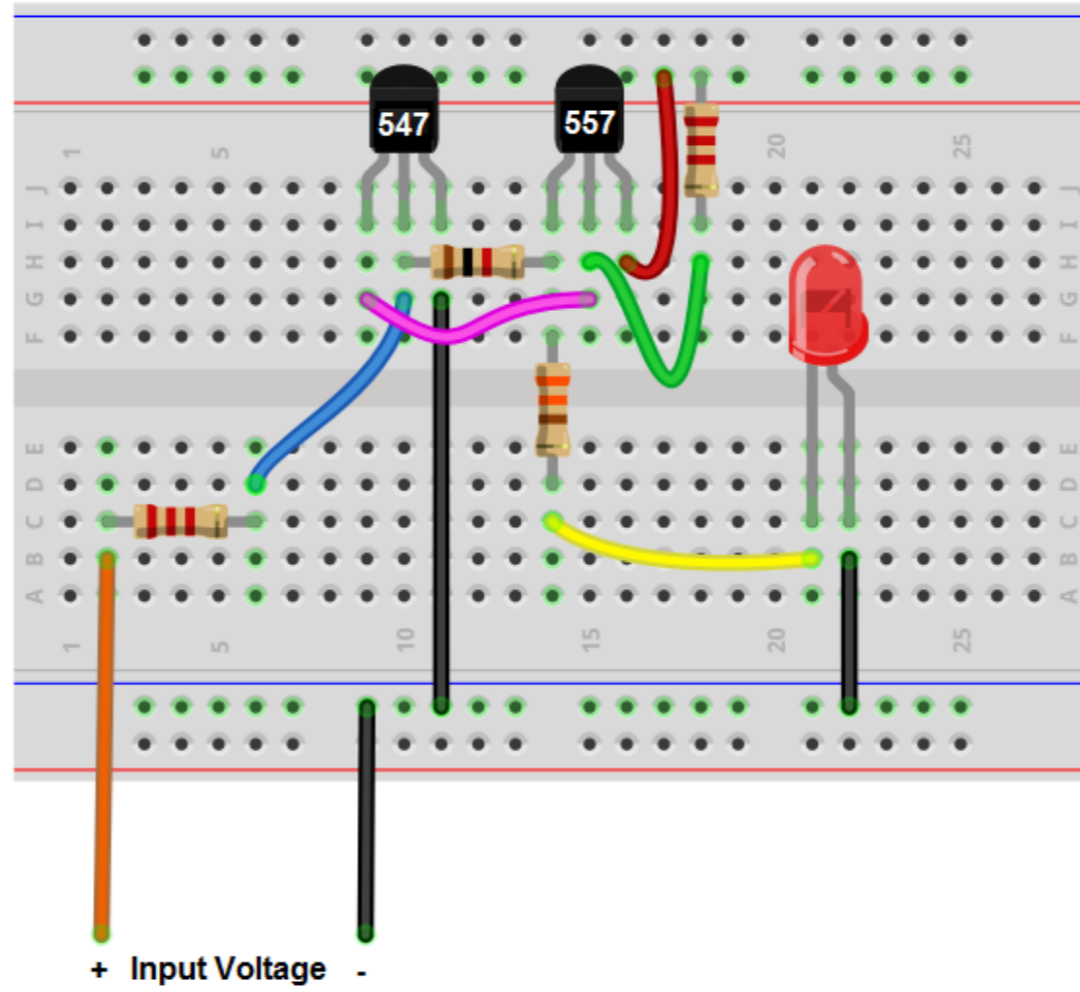
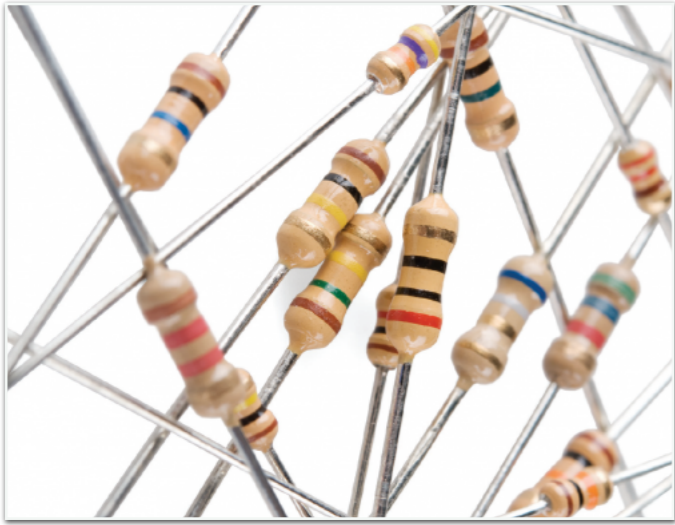
# Tutorial Outline

- ▶ Review of strand displacement
- ▶ Building and composing logic gates
- ▶ Tools for designing and verifying circuits
- ▶ Robustness of strand displacement
- ▶ **(Bonus) DSD circuits the easy way**

Does it need to be this  
difficult to build a circuit?





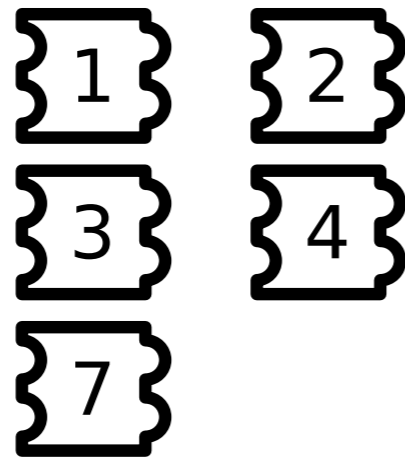


# Molecular breadboard 1.0

input signals

A1 B1  
A2 B2  
A3 B3  
A4 B4  
A7 B7

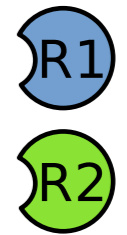
reaction gates



wires

X1.....R1 Y3.....A2  
Y1.....R2 Y3.....B2  
X2.....B1 X4.....B1  
X2.....R2 X4.....B2  
Y2.....R1 Y4.....A2  
X3.....B1 Y4.....R2  
X3.....B4 X7.....A1  
X3.....R2

reporters



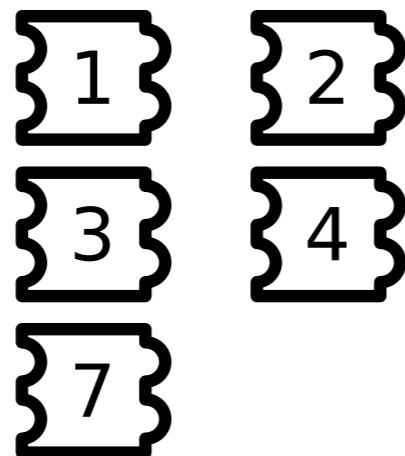
Built using leakless motif

# Molecular breadboard 1.0

input signals

A1 B1  
A2 B2  
A3 B3  
A4 B4  
A7 B7

reaction gates



wires

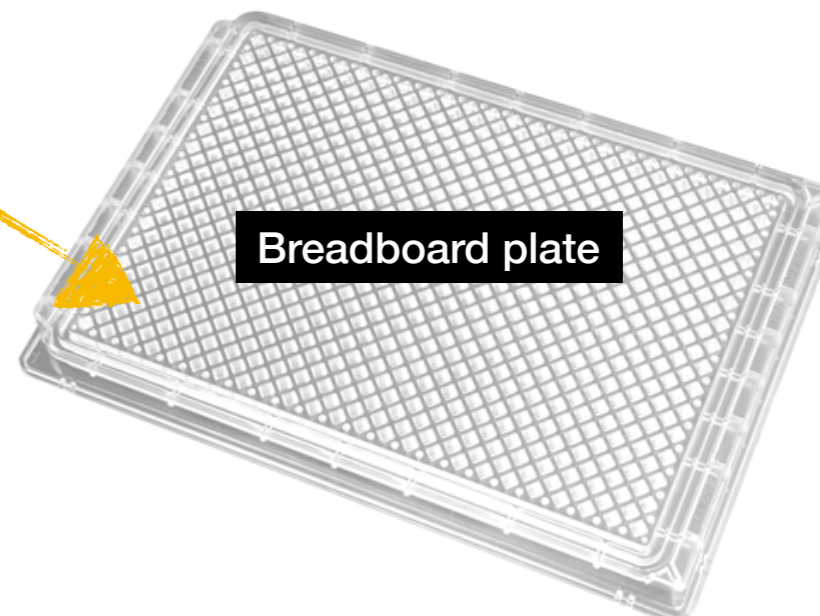
X1.....R1 Y3.....A2  
Y1.....R2 Y3.....B2  
X2.....B1 X4.....B1  
X2.....R2 X4.....B2  
Y2.....R1 Y4.....A2  
X3.....B1 Y4.....R2  
X3.....B4 X7.....A1  
X3.....R2

reporters



Load breadboard components onto 384-well plate

Built using leakless motif



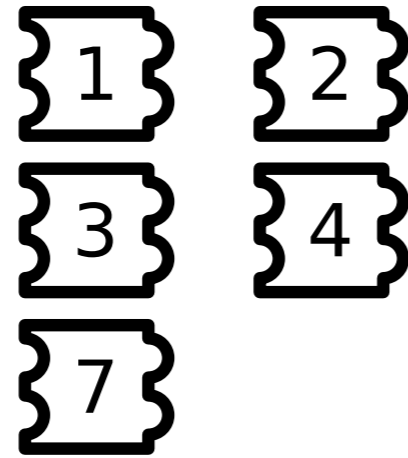


# Molecular breadboard 1.0

input signals

A1 B1  
A2 B2  
A3 B3  
A4 B4  
A7 B7

reaction gates



wires

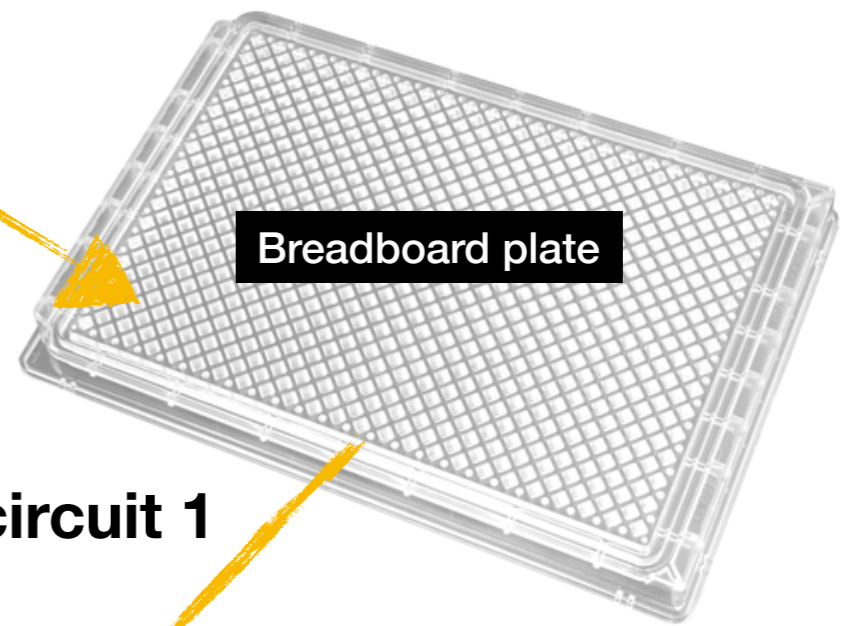
X1.....R1 Y3.....A2  
Y1.....R2 Y3.....B2  
X2.....B1 X4.....B1  
X2.....R2 X4.....B2  
Y2.....R1 Y4.....A2  
X3.....B1 Y4.....R2  
X3.....B4 X7.....A1  
X3.....R2

reporters

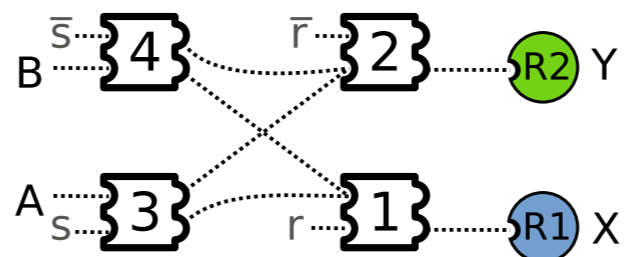


Load breadboard components onto 384-well plate

Built using leakless motif



circuit 1

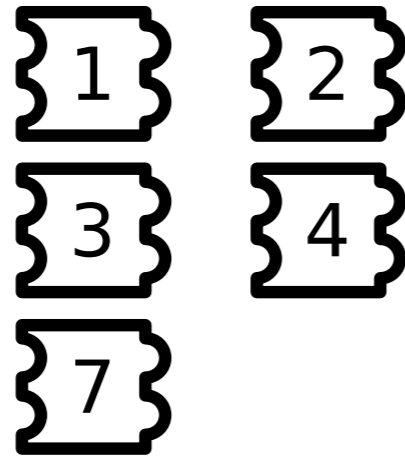


# Molecular breadboard 1.0

input signals

A1 B1  
A2 B2  
A3 B3  
A4 B4  
A7 B7

reaction gates



wires

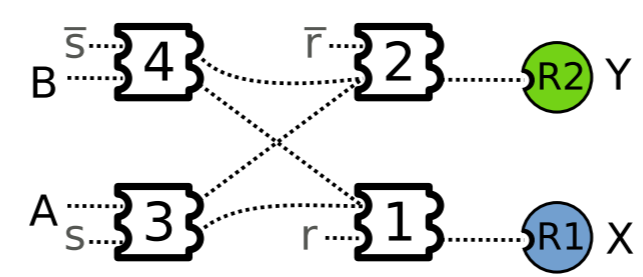
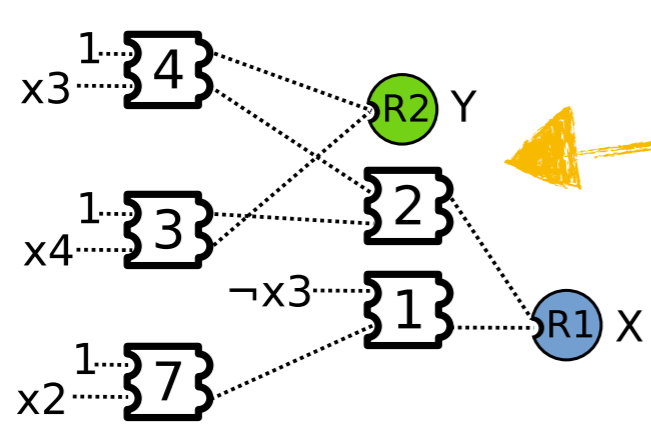
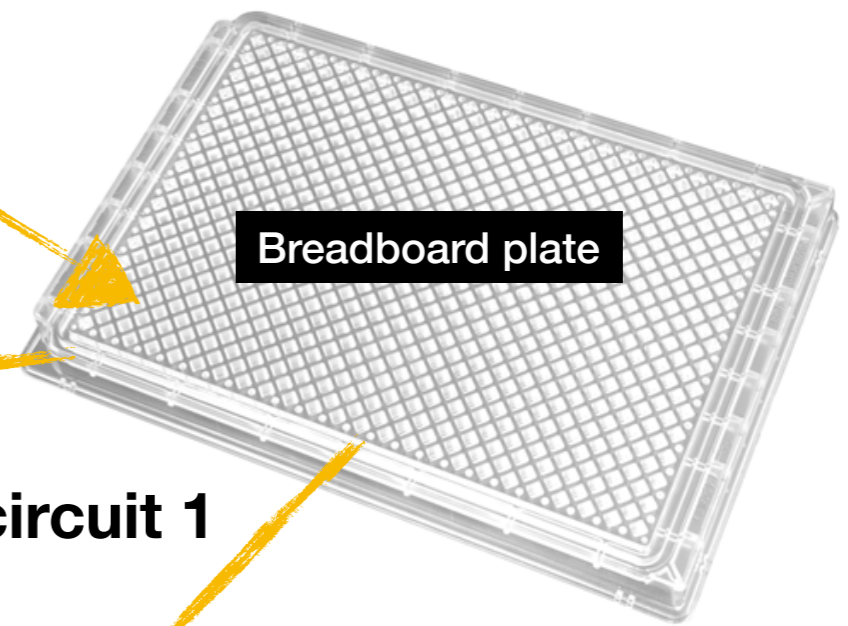
X1.....R1 Y3.....A2  
Y1.....R2 Y3.....B2  
X2.....B1 X4.....B1  
X2.....R2 X4.....B2  
Y2.....R1 Y4.....A2  
X3.....B1 Y4.....R2  
X3.....B4 X7.....A1  
X3.....R2

reporters



Load breadboard components onto 384-well plate

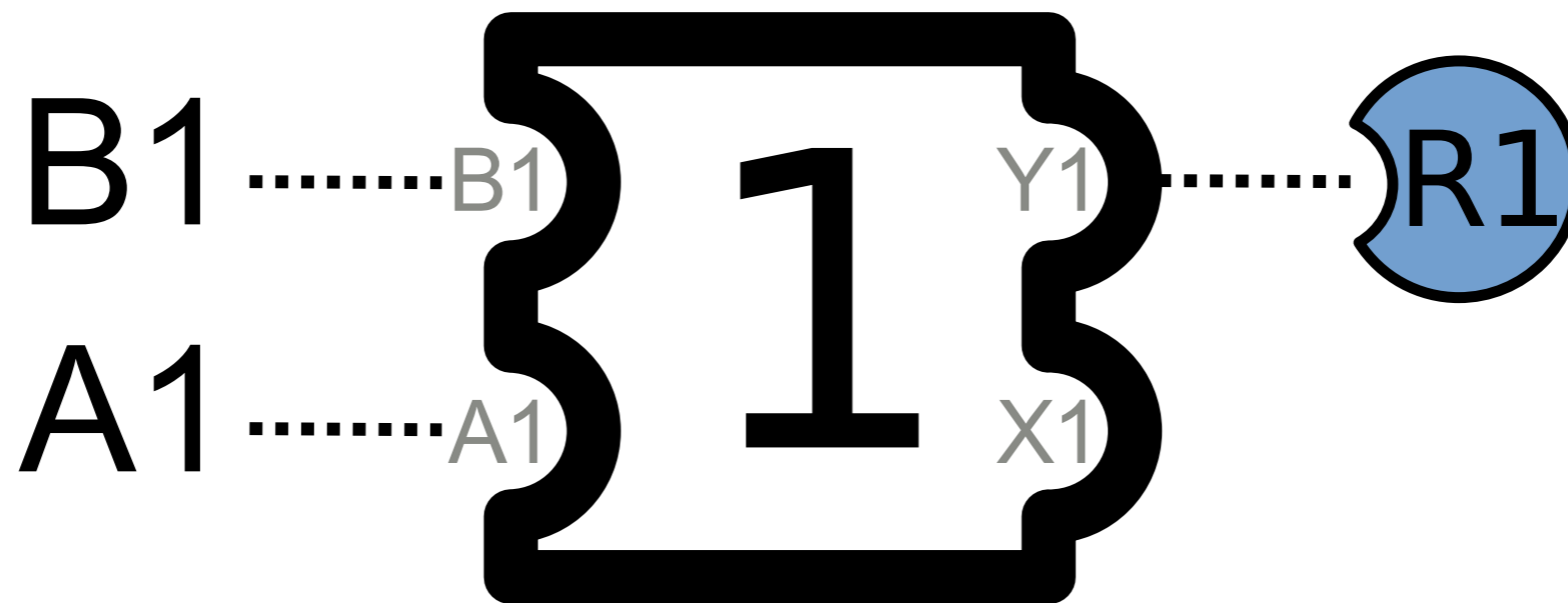
Built using leakless motif



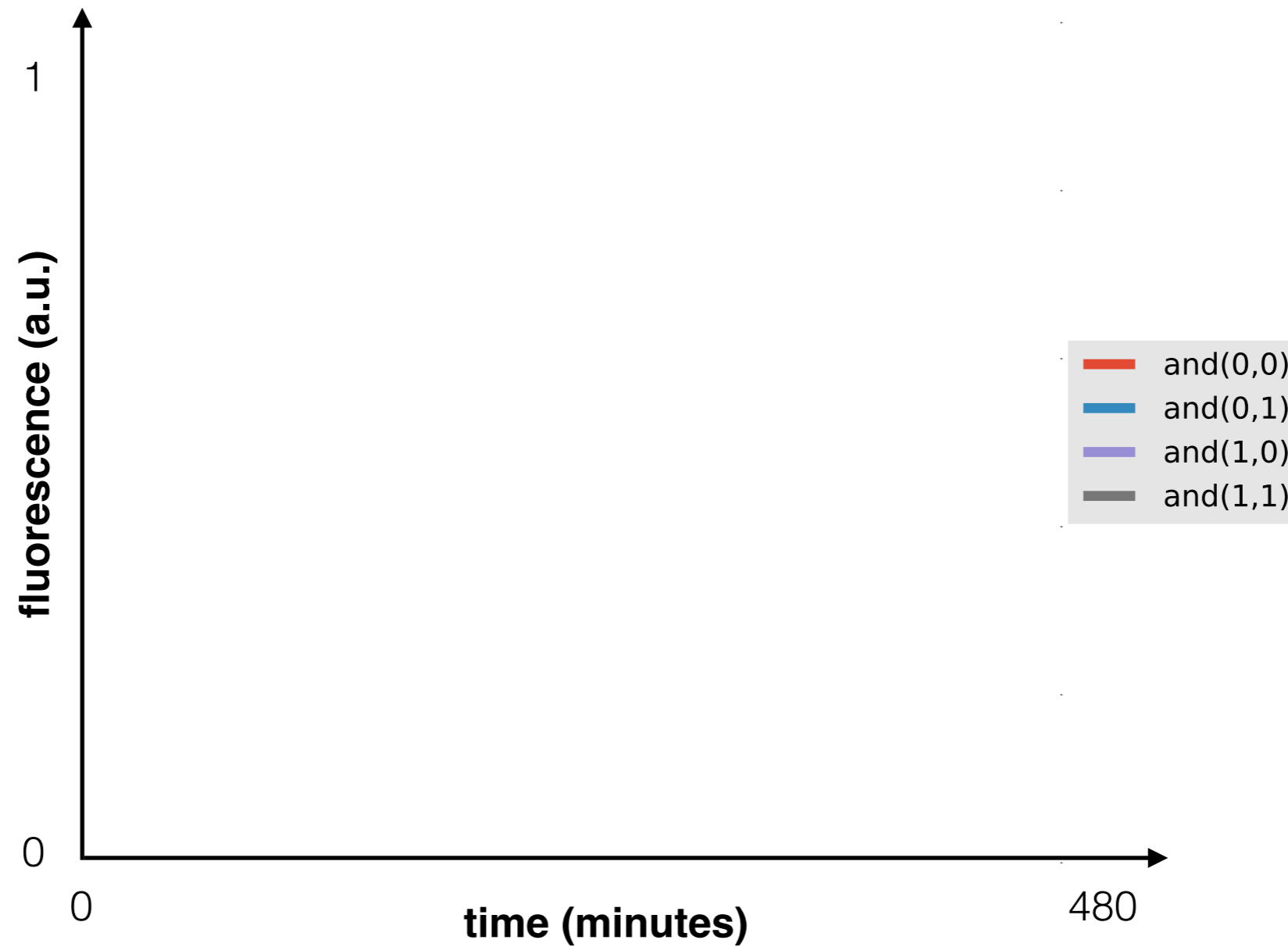
# Testing breadboard components

- ▶ Typical DSD circuits are 50nM - 200nM concentration  
(our circuits can operate at these concentrations)
- ▶ To demonstrate *robustness*, all experiments will be at 2uM  
(~20x higher than typical concentrations)

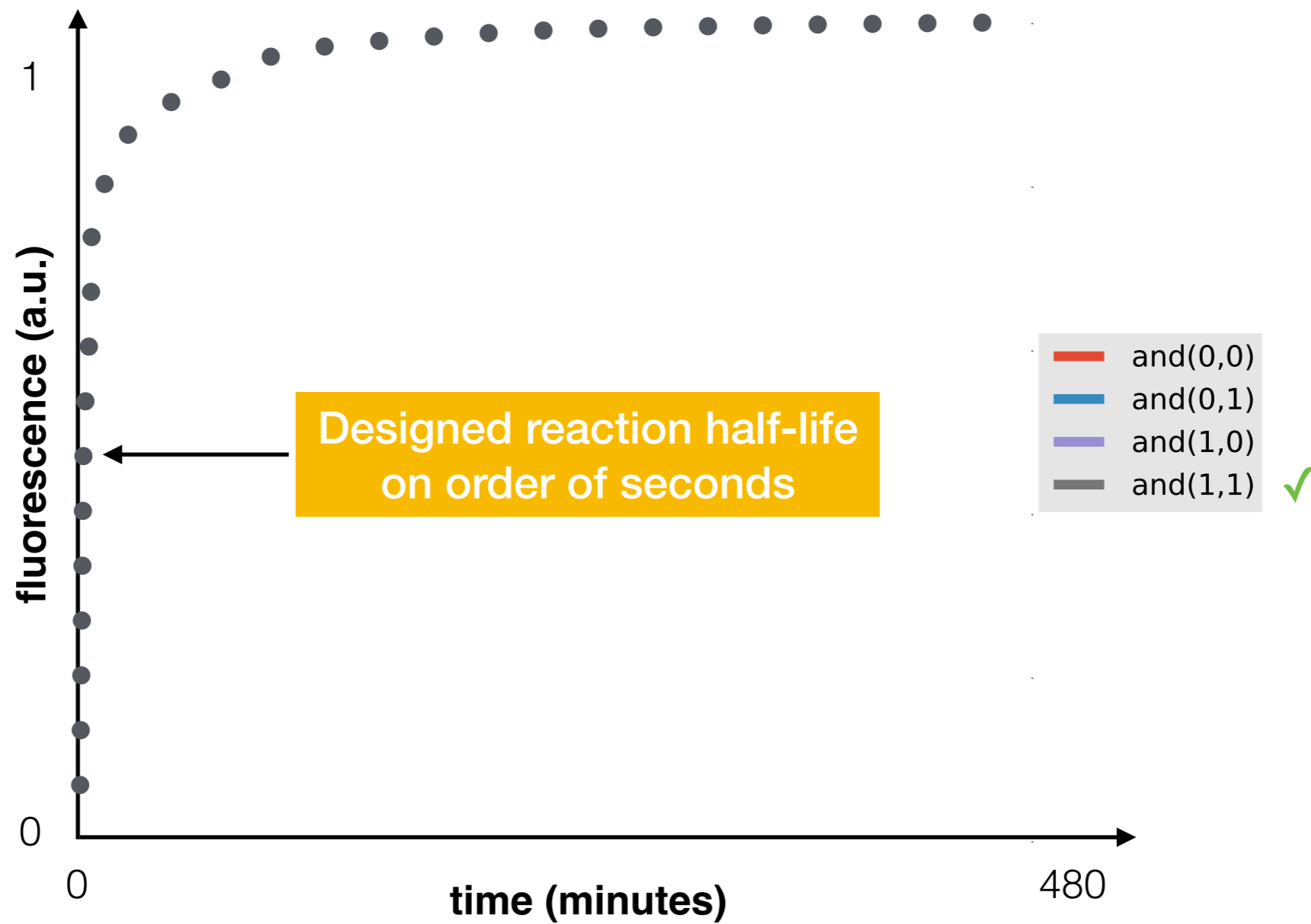
# AND gate



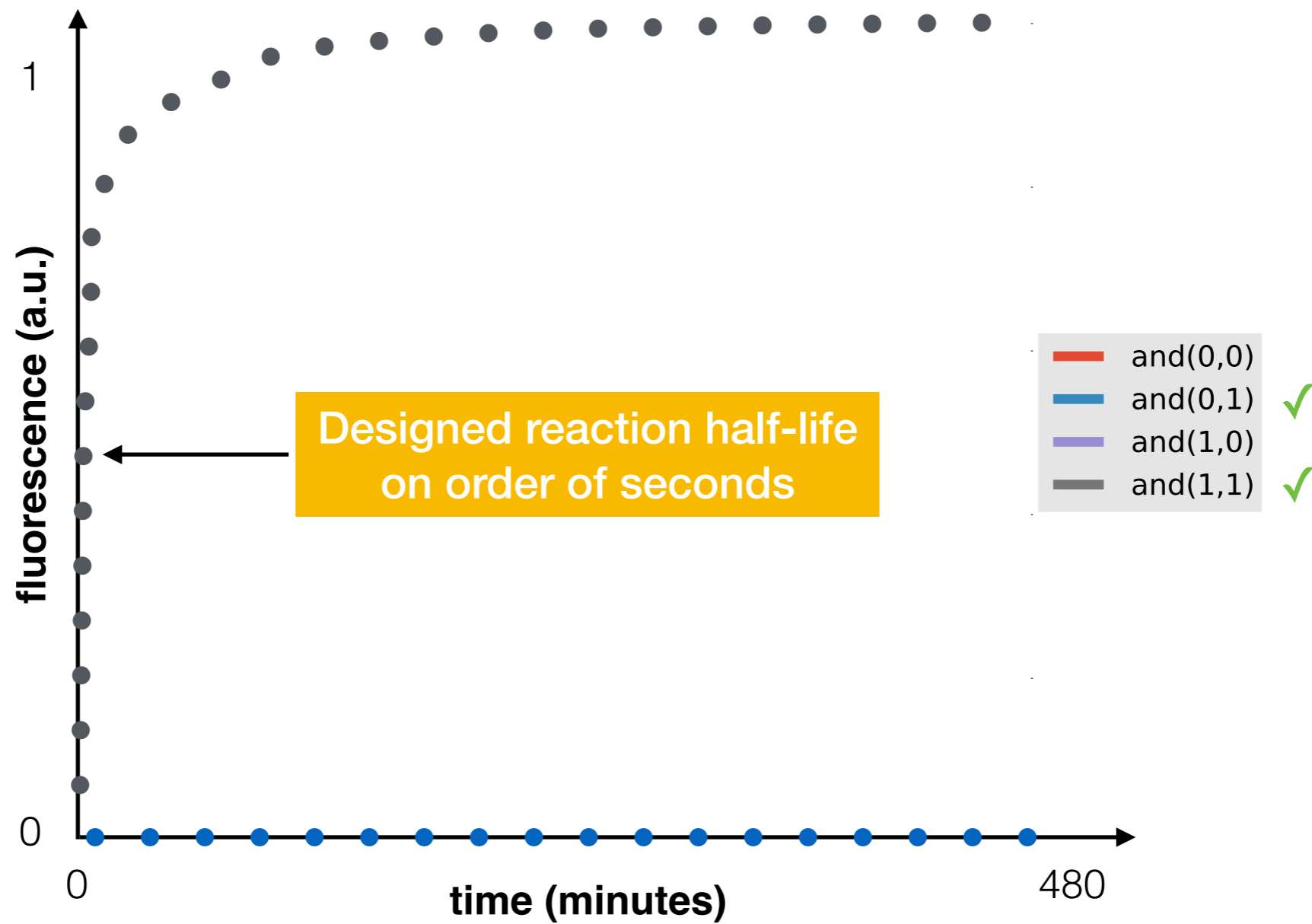
# *Ideal* AND gate simulation @ 2 $\mu$ M



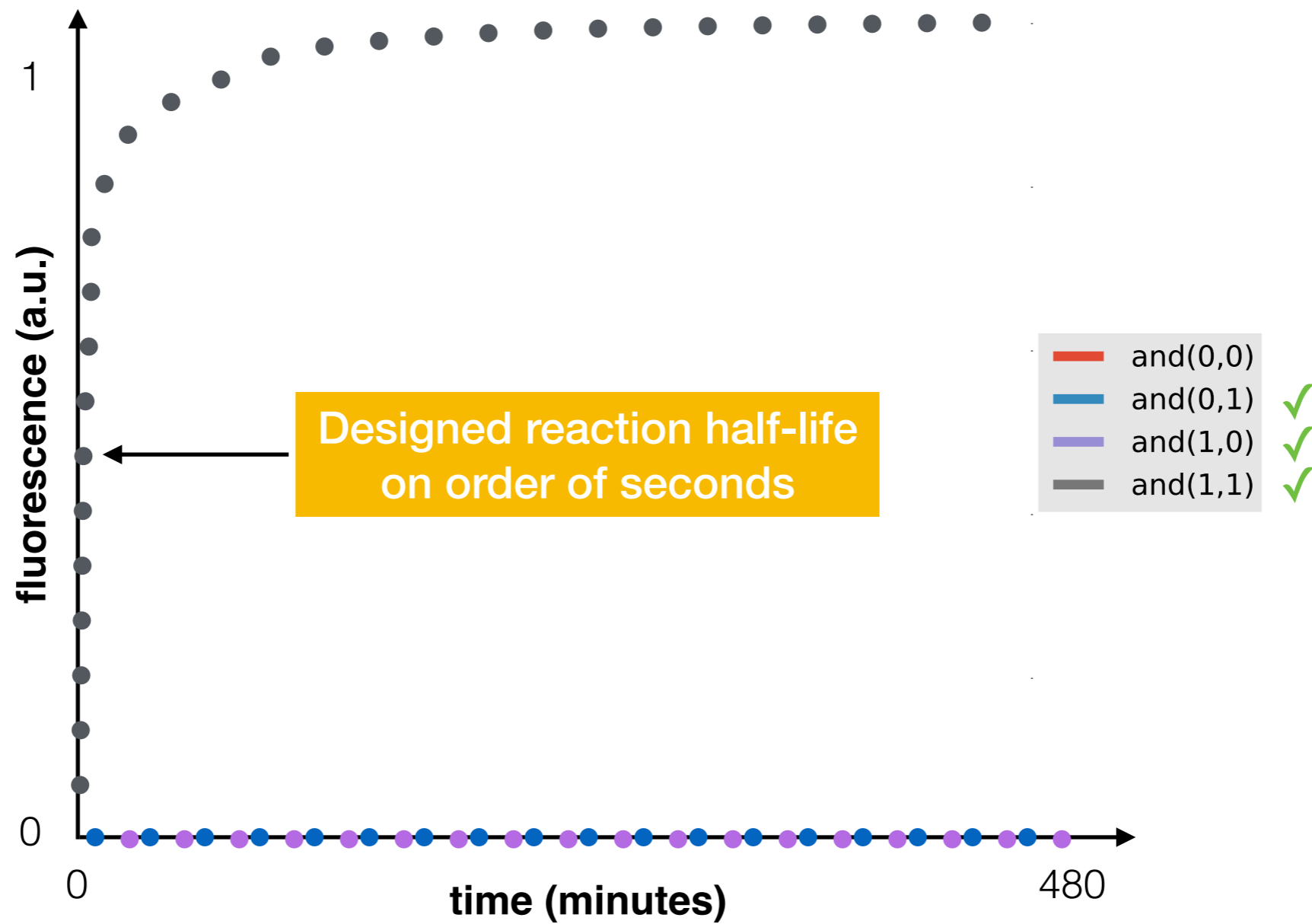
# Ideal AND gate simulation @ 2 $\mu\text{M}$



# Ideal AND gate simulation @ 2 $\mu\text{M}$

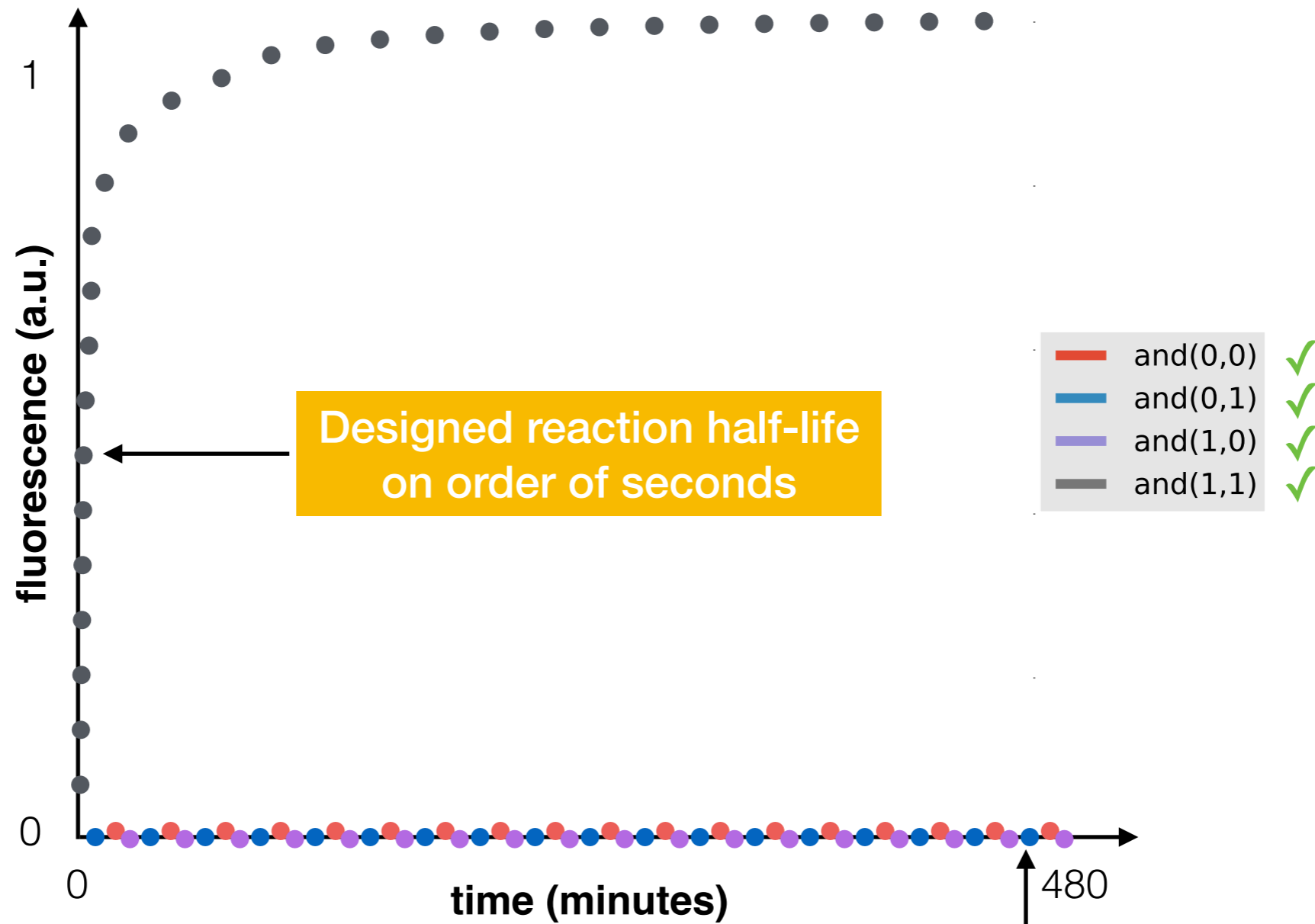


# *Ideal* AND gate simulation @ 2 $\mu\text{M}$



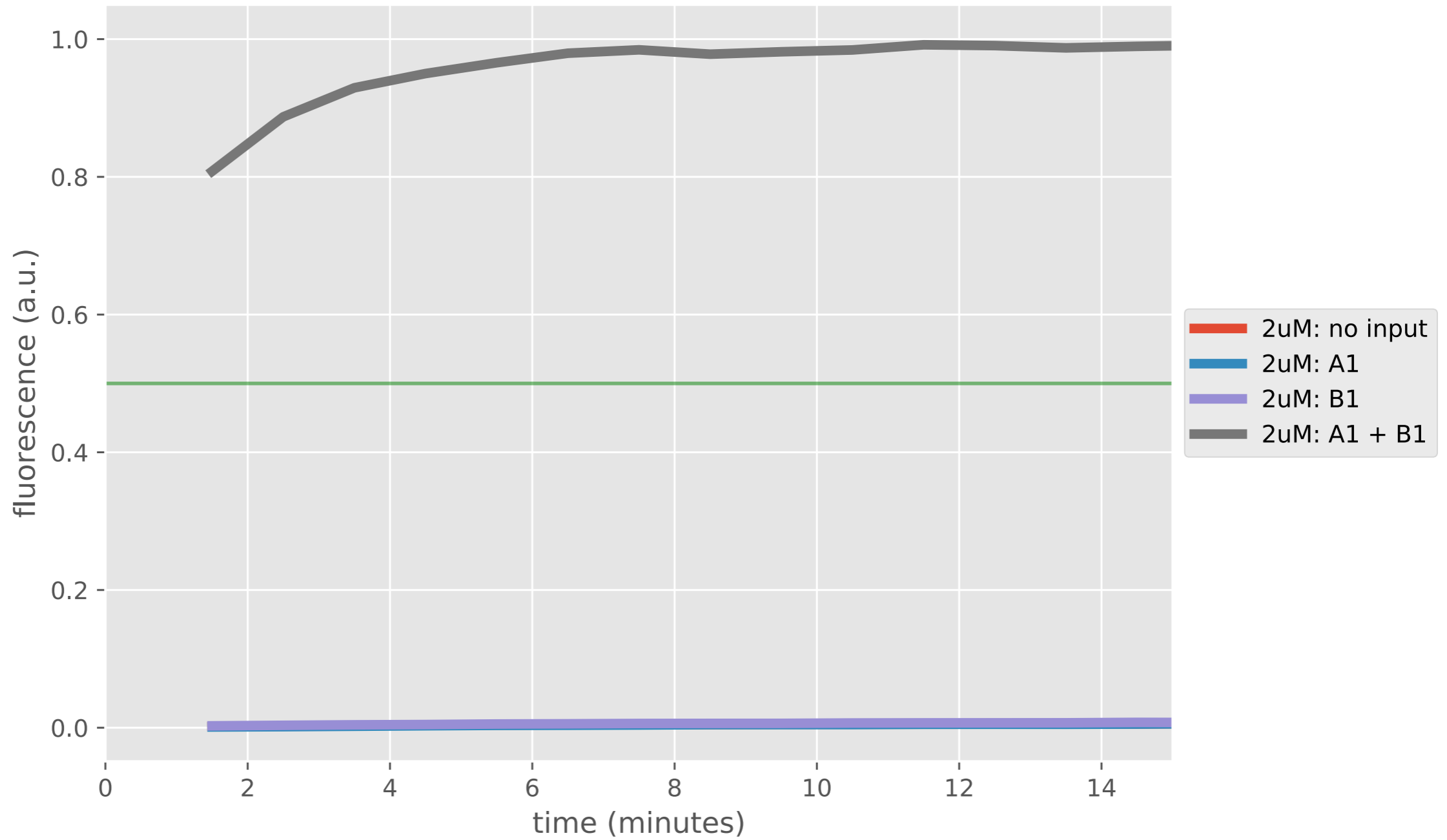
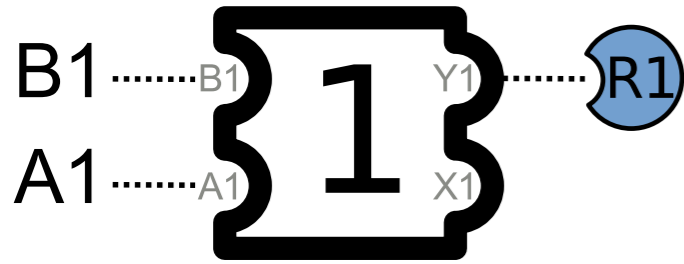


# Ideal AND gate simulation @ 2 $\mu\text{M}$



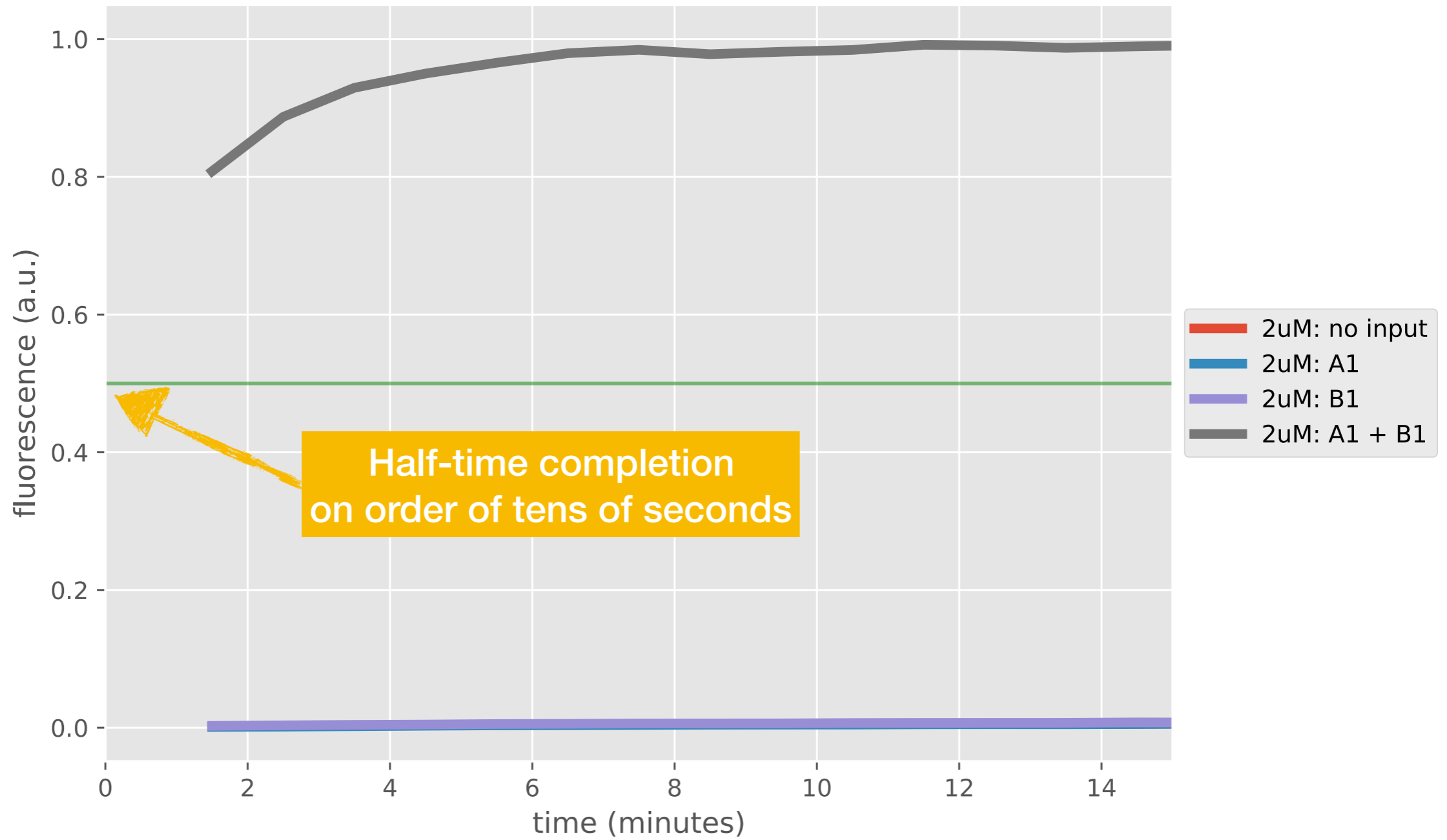
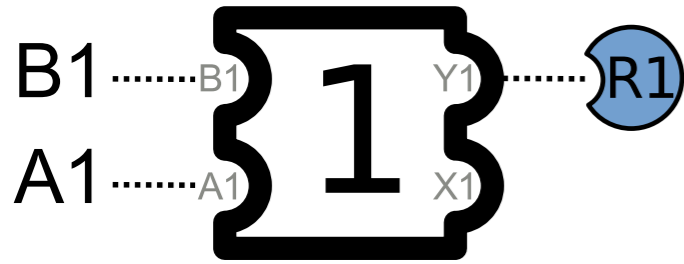
Spurious reactions not observable on order of hours

# AND gate @ 2 $\mu$ M



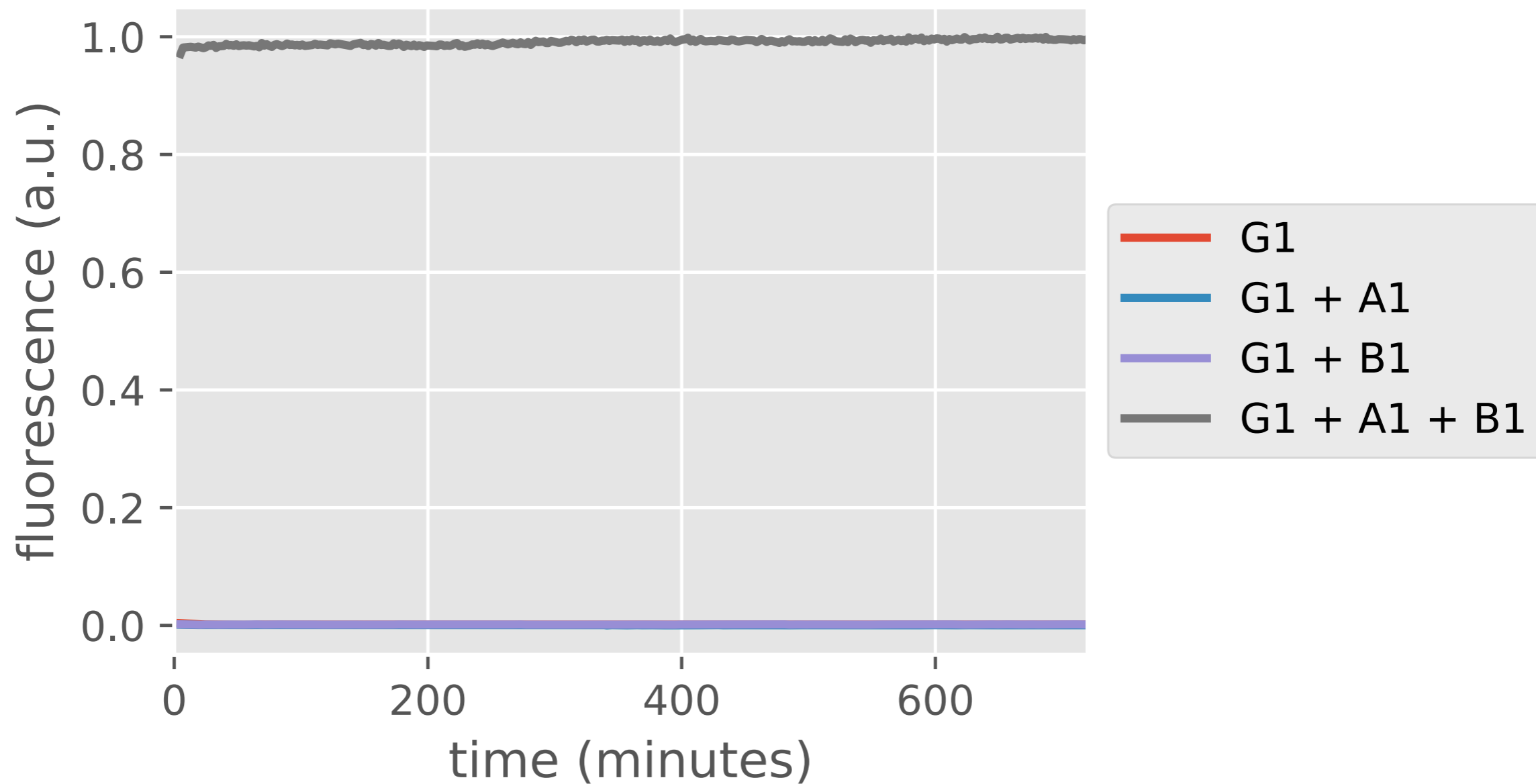
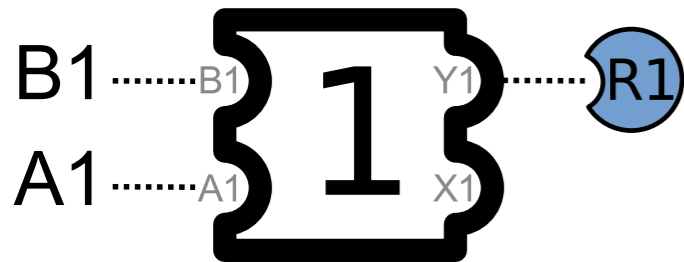
[fuel]=[input]=2uM, [reporter]=1uM

# AND gate @ 2 $\mu$ M



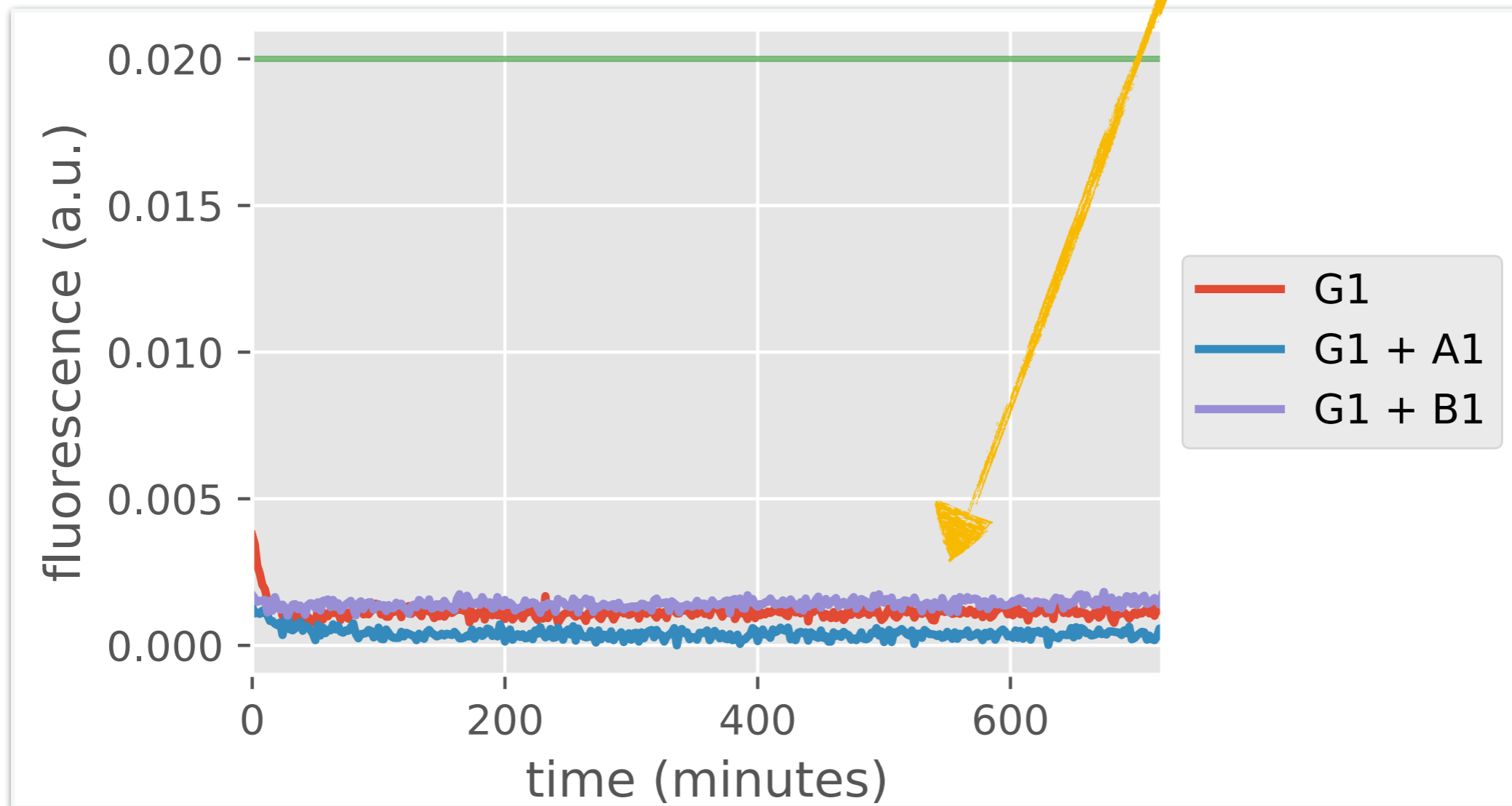
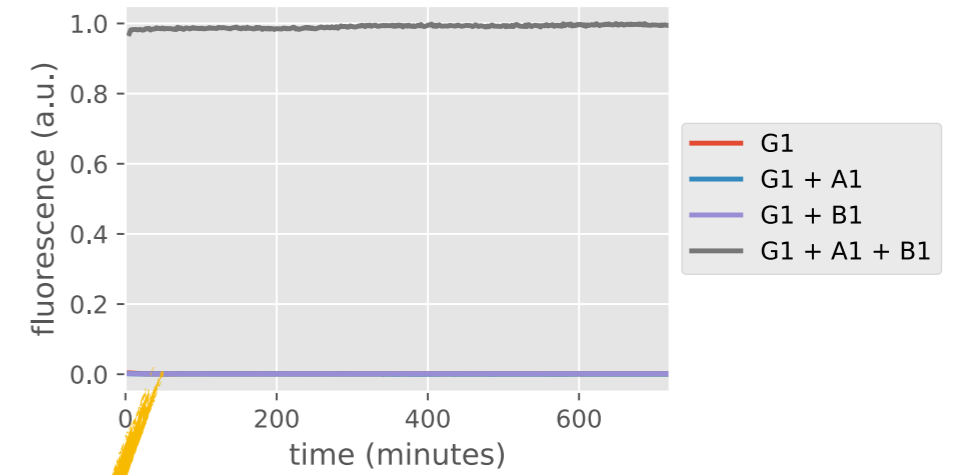
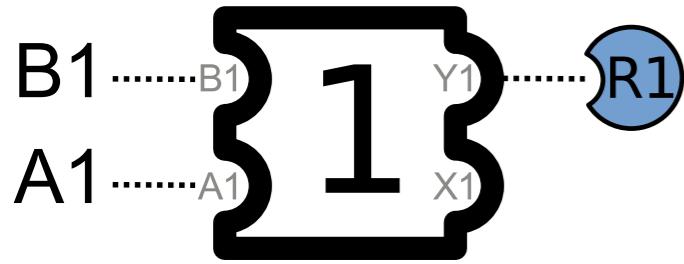
[fuel]=[input]=2uM, [reporter]=1uM

# AND gate @ 2 $\mu$ M (12 hours)



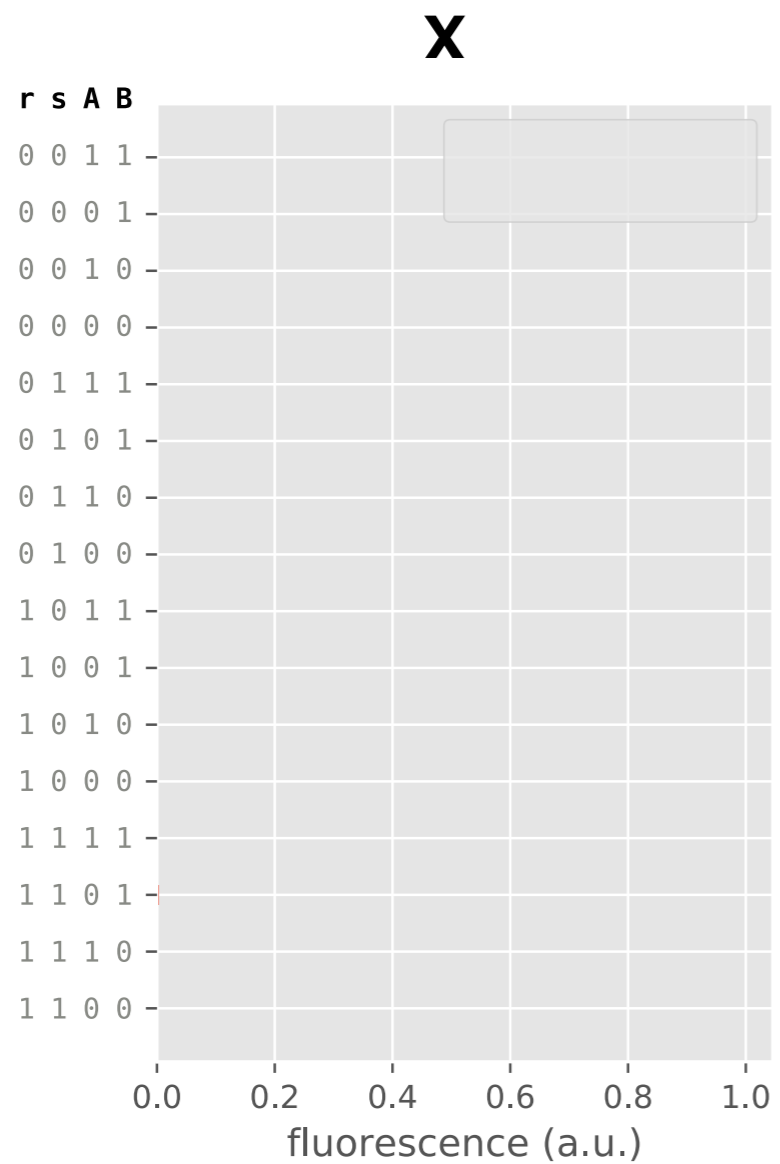
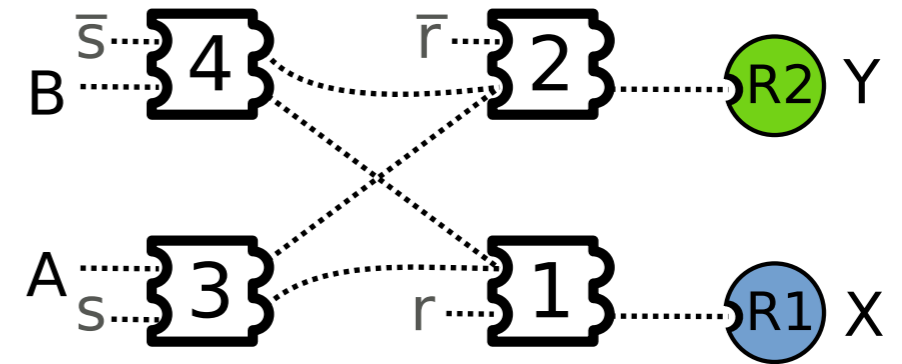
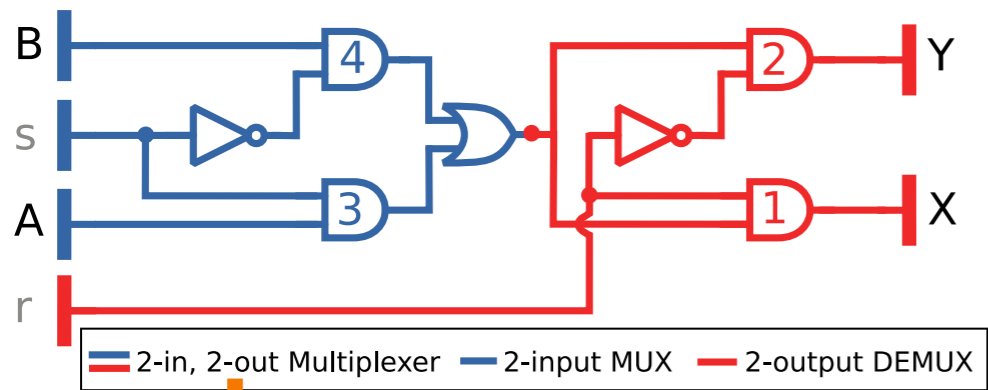
[fuel]=[input]=2 $\mu$ M, [reporter]=2.5 $\mu$ M

# AND gate @ 2 $\mu$ M (12 hours)

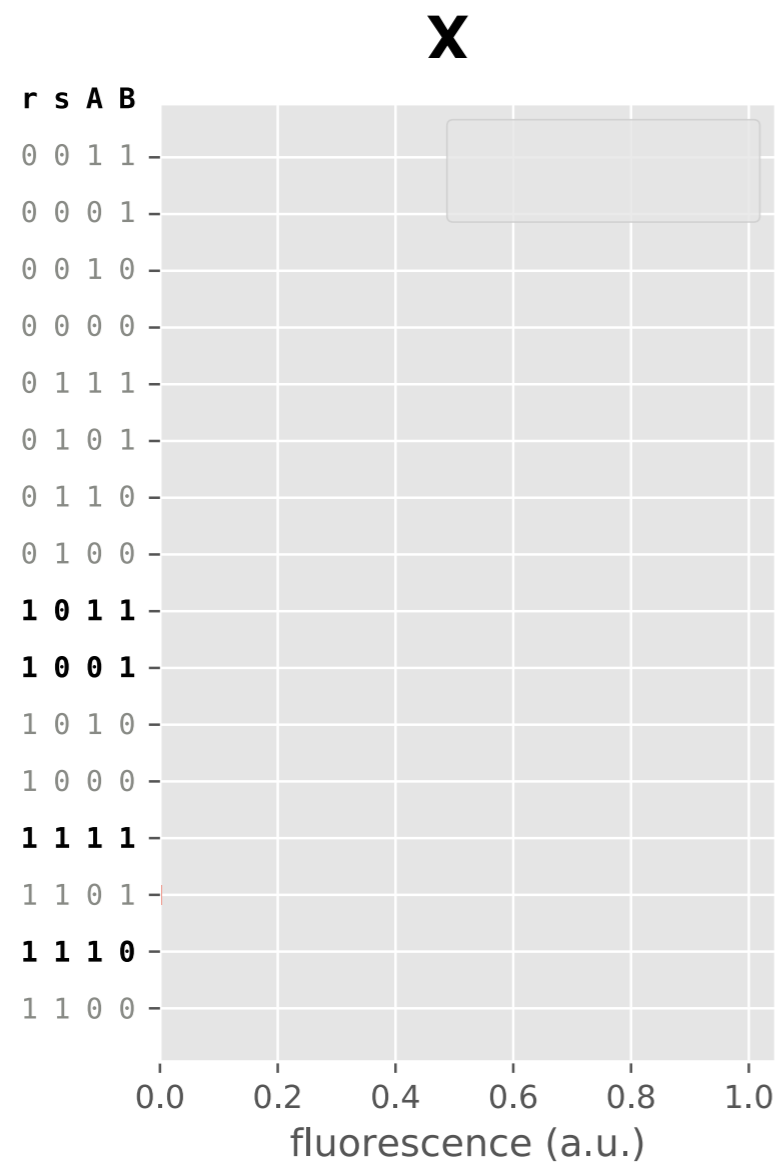
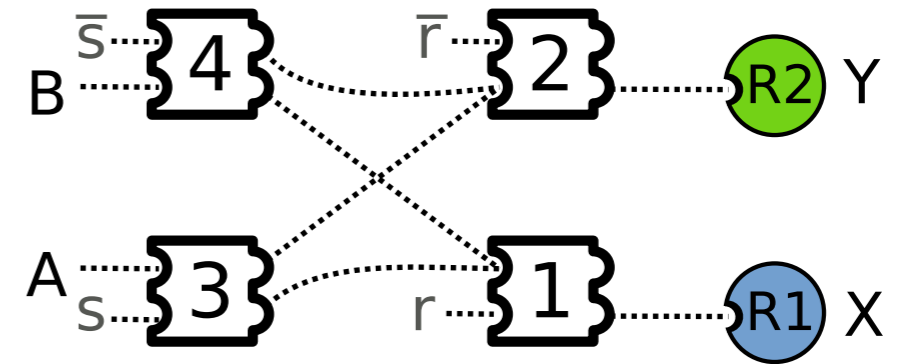
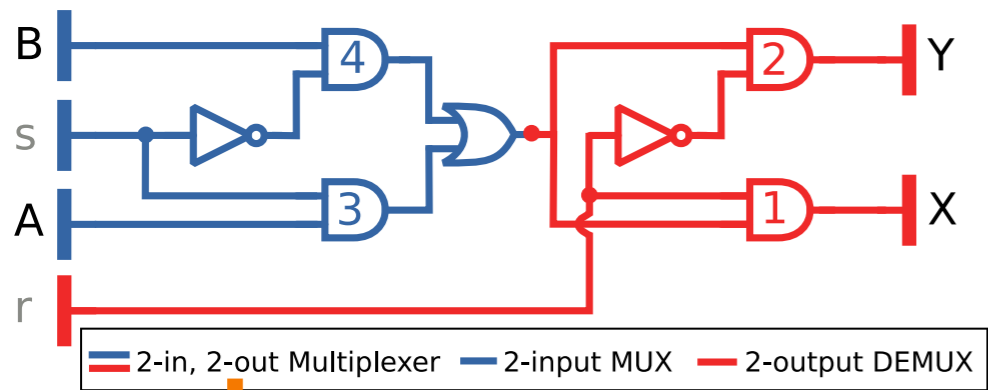


[fuel]=[input]=2 $\mu$ M, [reporter]=2.5 $\mu$ M

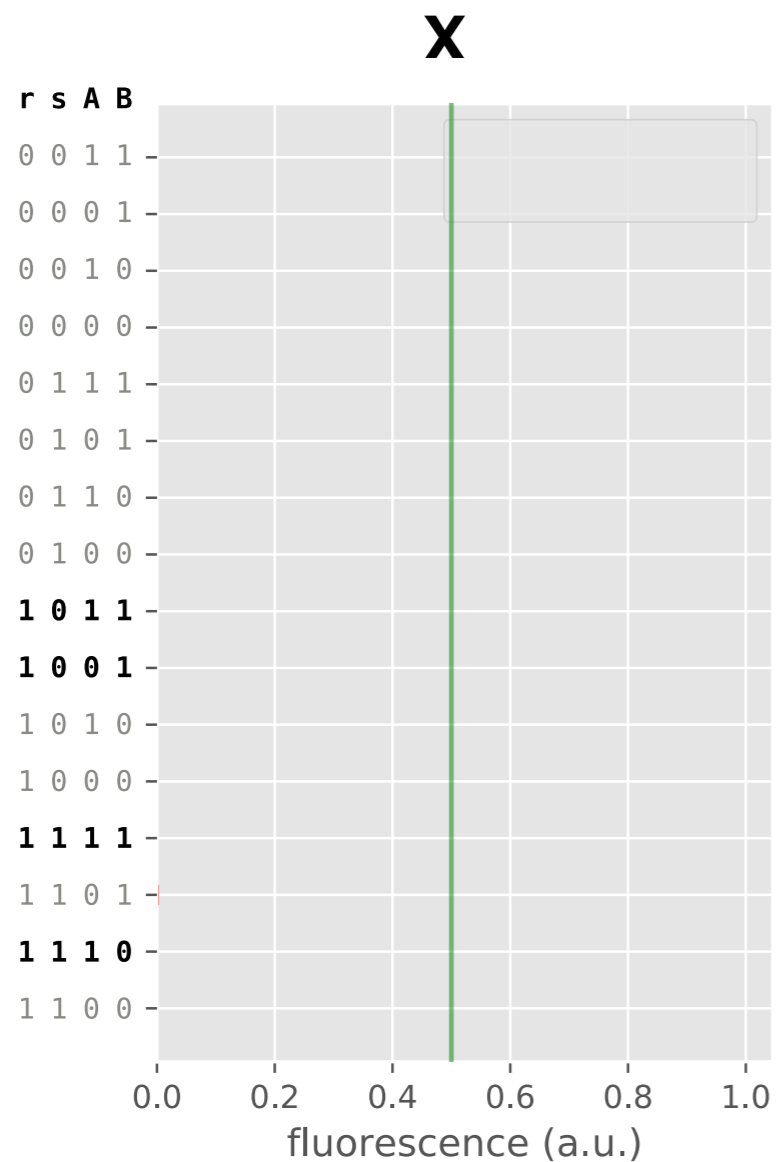
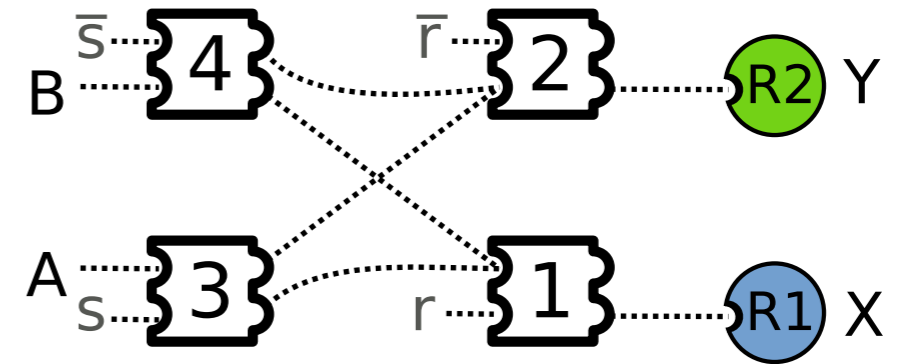
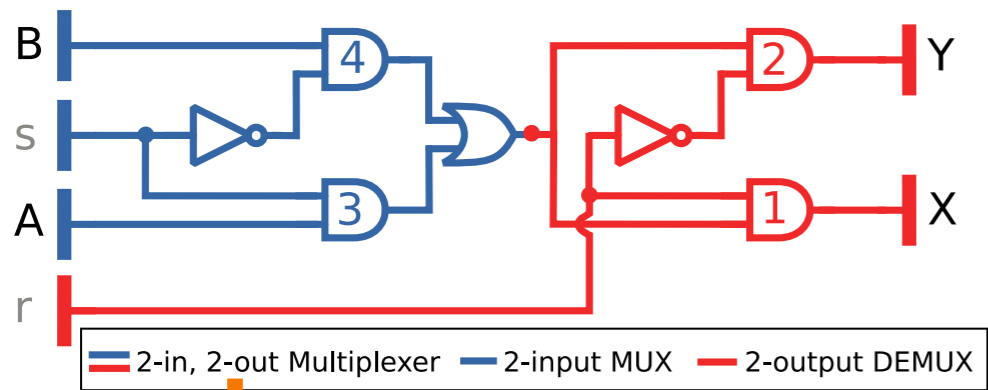
# Multiplexer-Demultiplexer



# Multiplexer-Demultiplexer

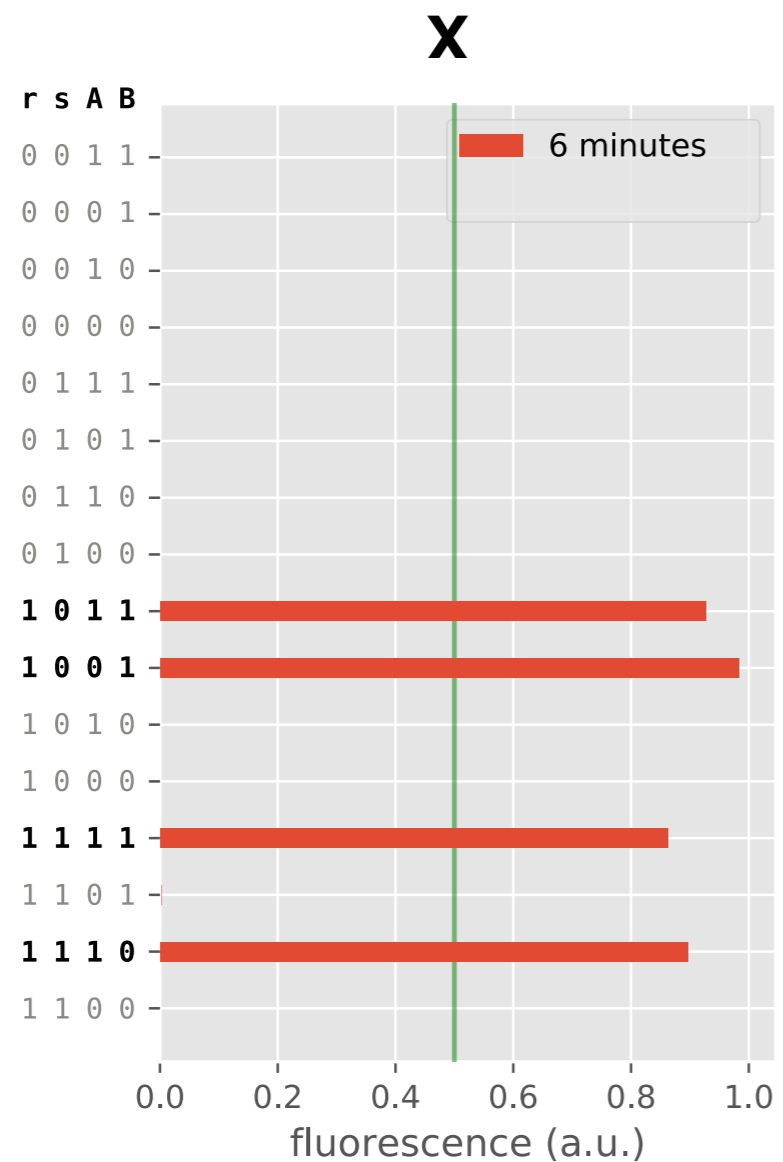
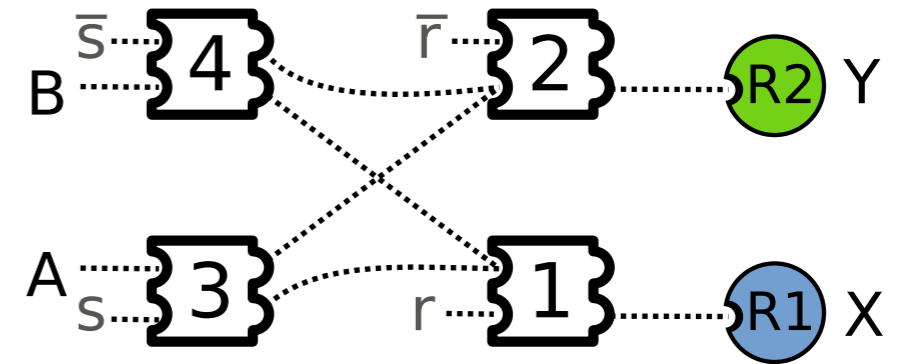
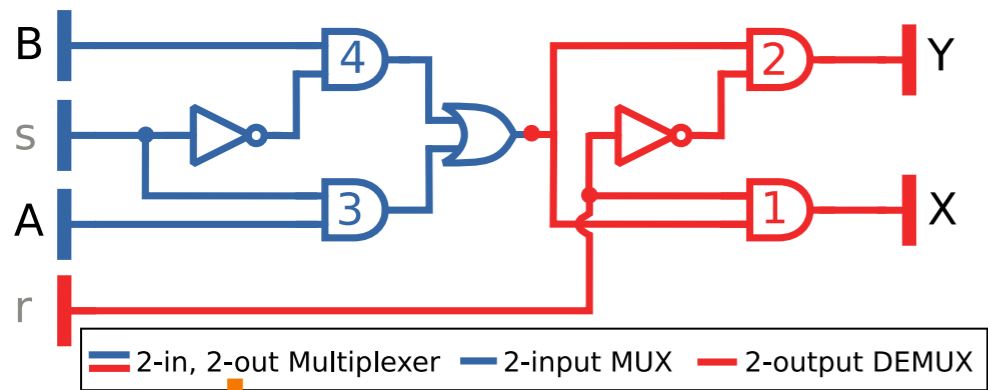


# Multiplexer-Demultiplexer

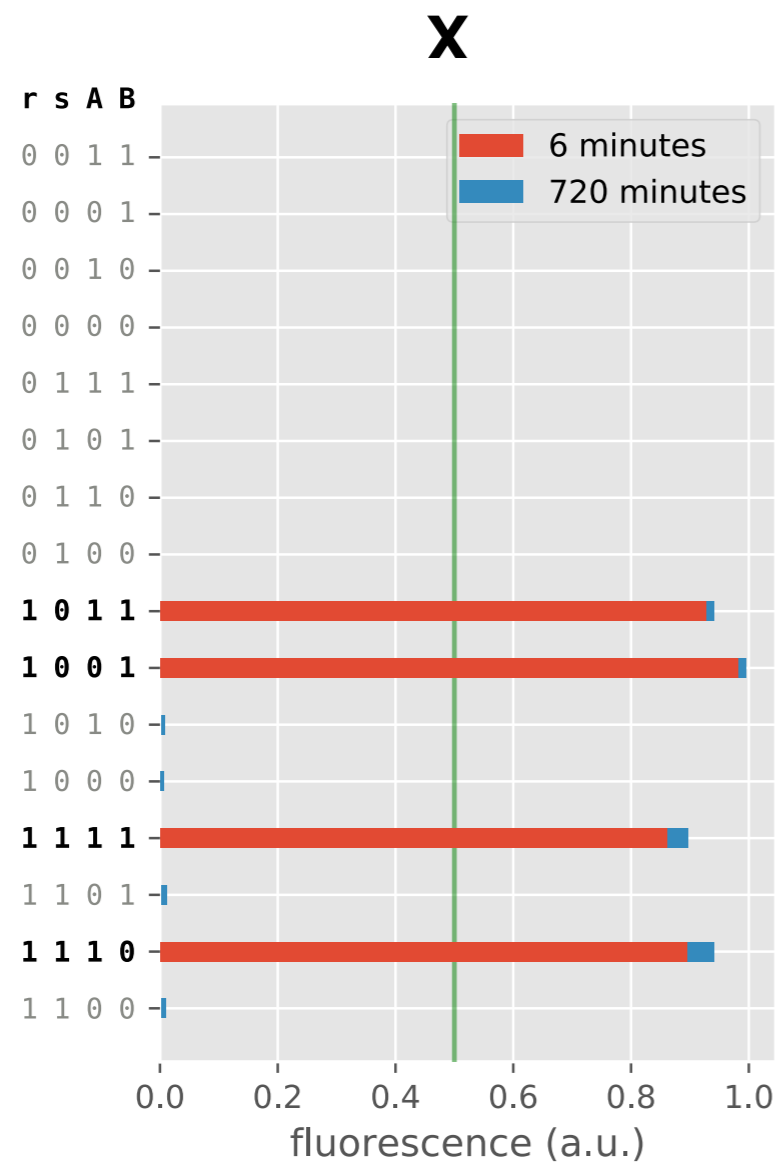
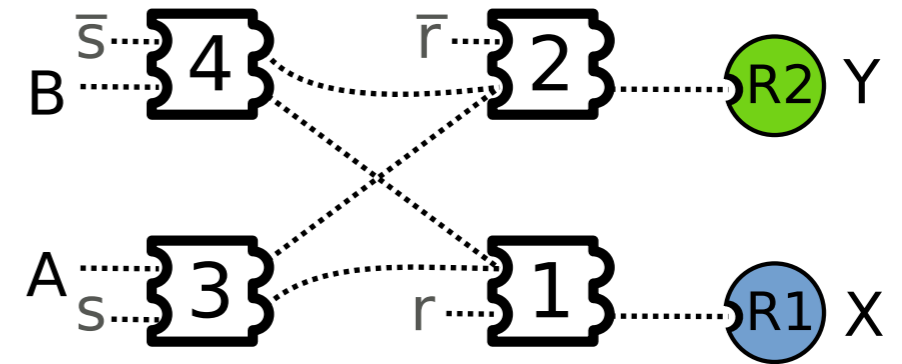
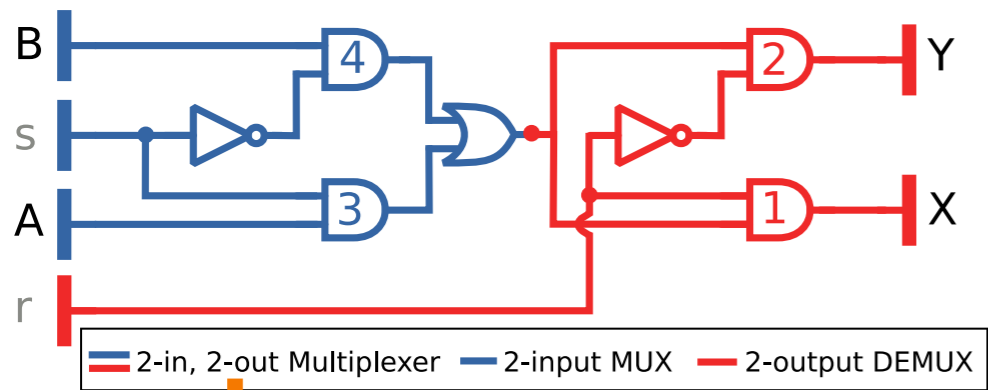




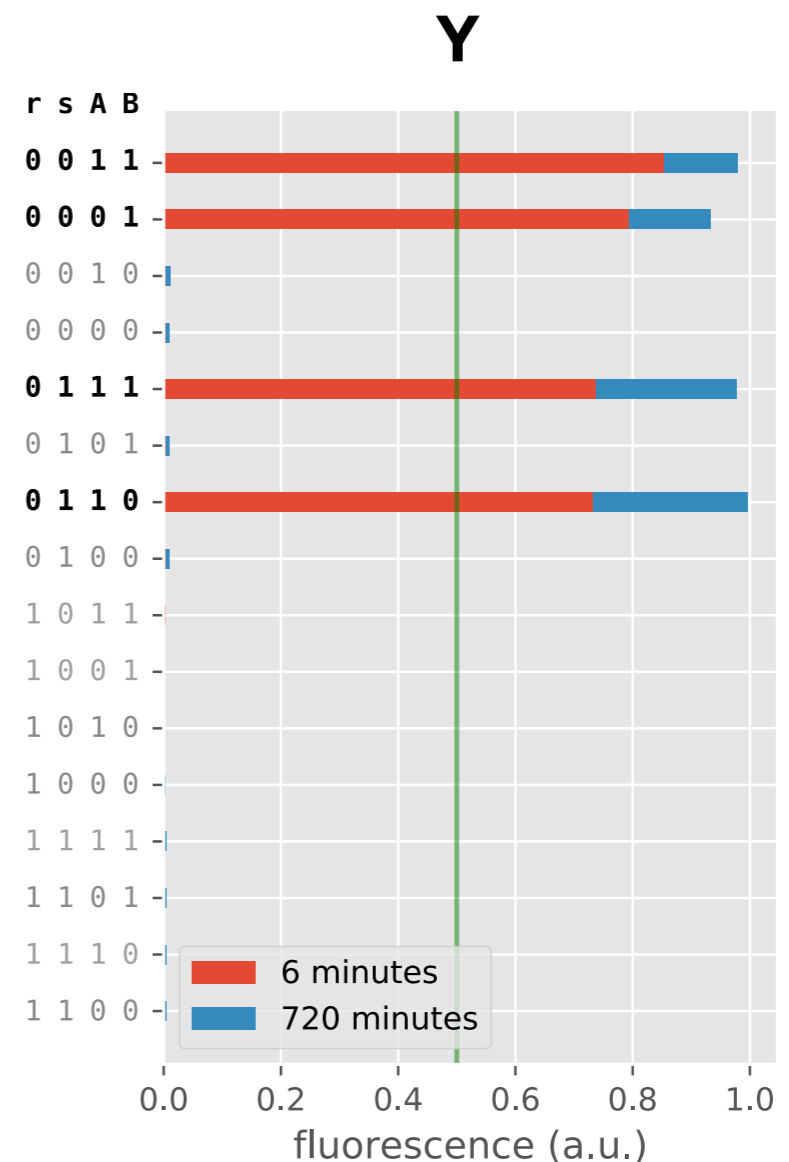
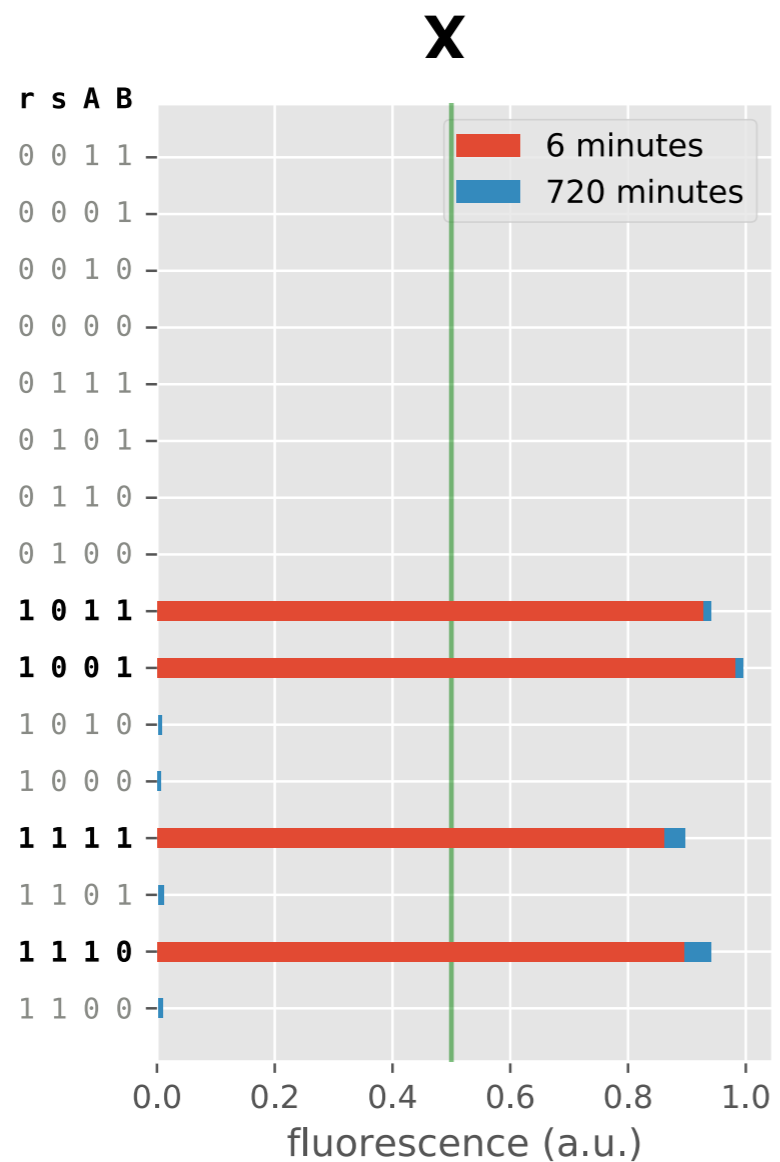
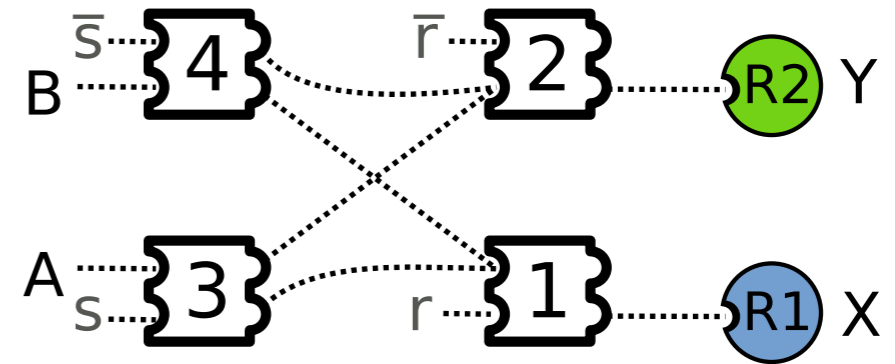
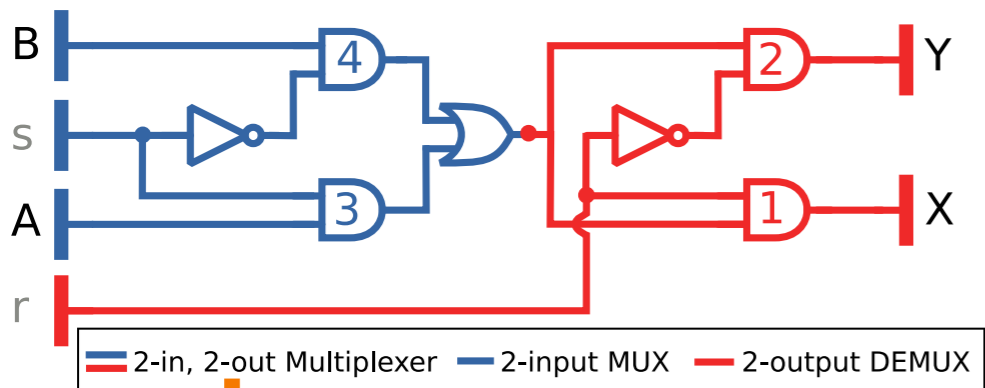
# Multiplexer-Demultiplexer



# Multiplexer-Demultiplexer

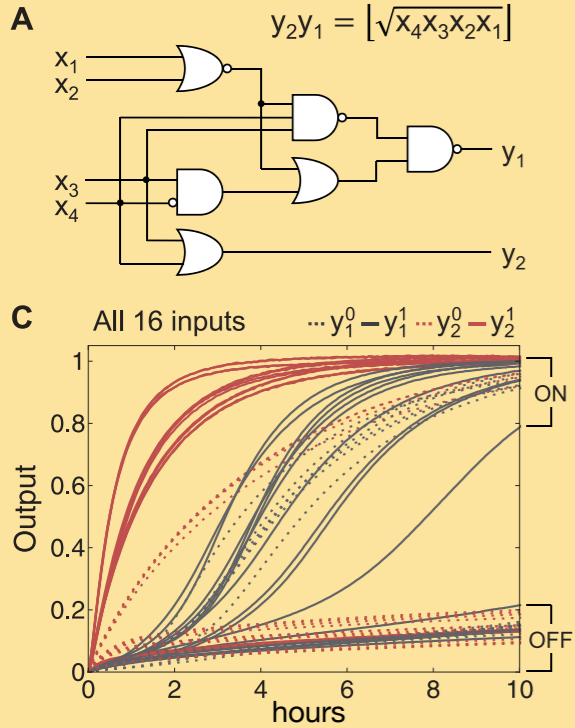


# Multiplexer-Demultiplexer



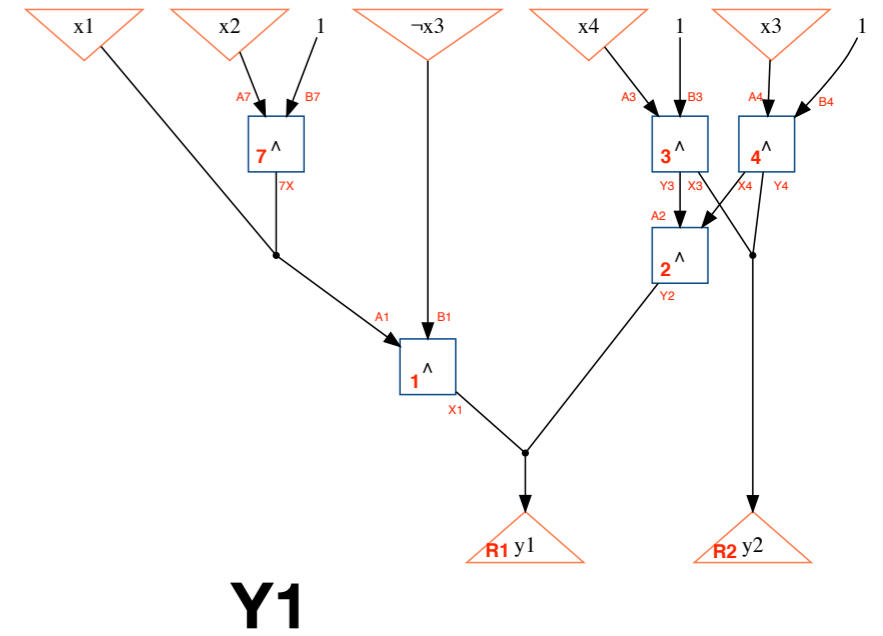
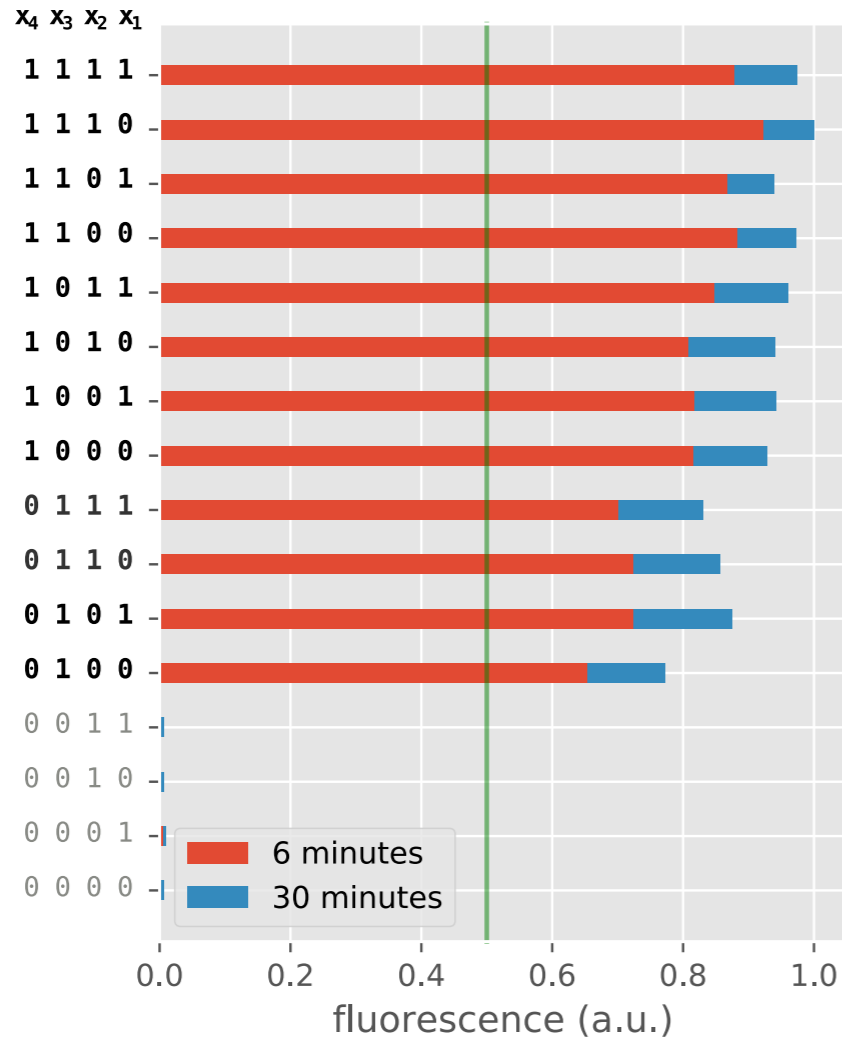
# Large circuits that are fast

Qian & Winfree 2011

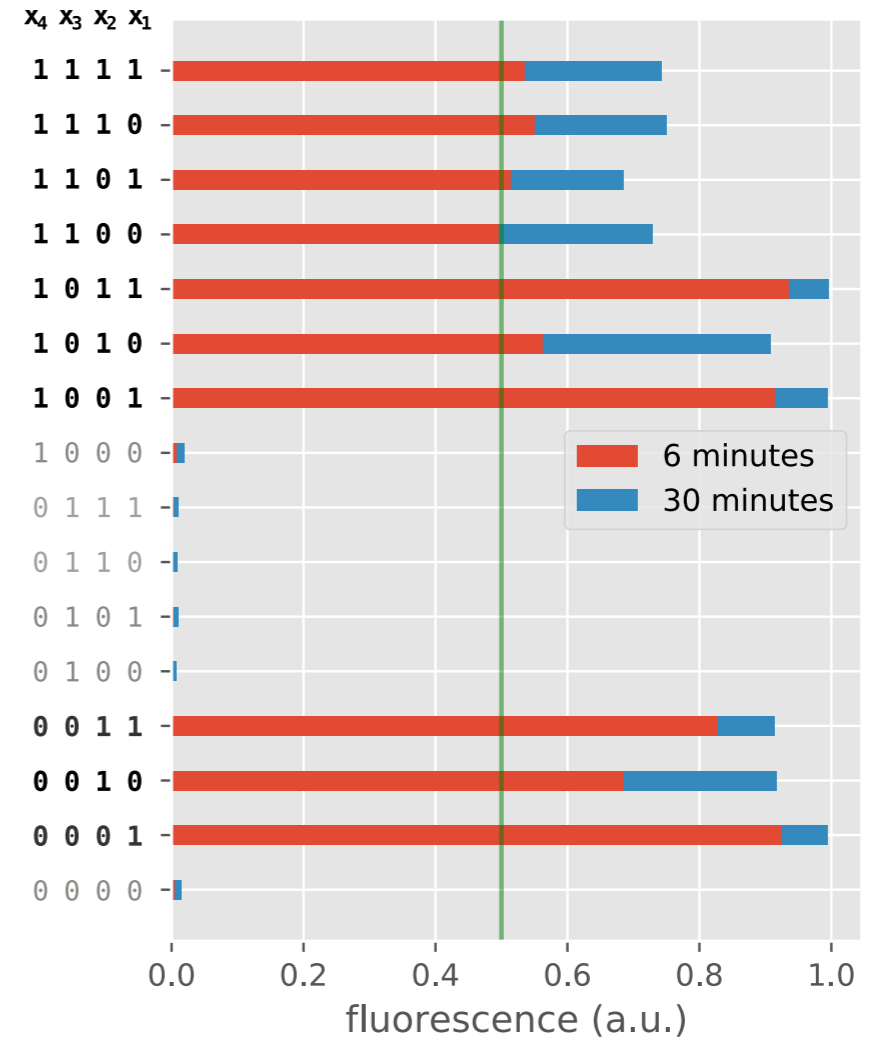


Reaction half-time improved from ~6 hours to < 6 minutes

**Y2**



**Y1**

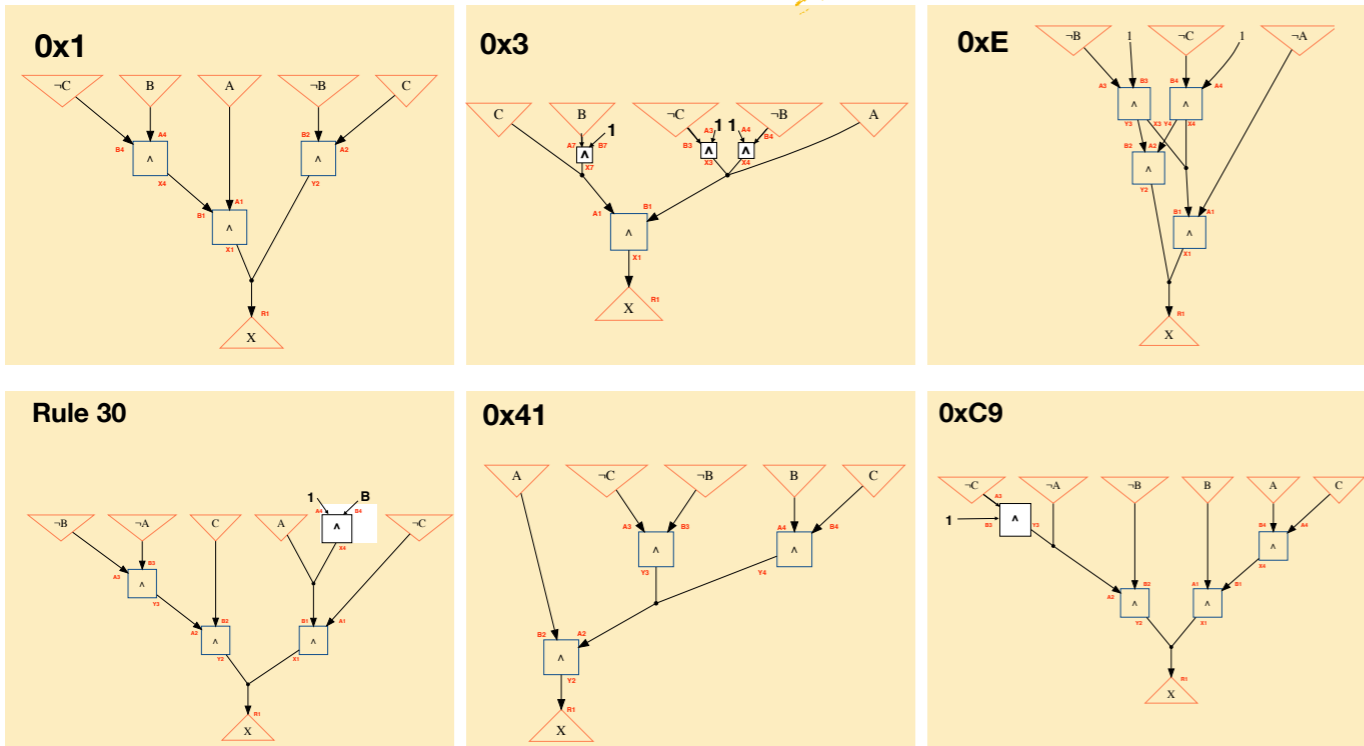


Breadboard plate

Acoustic  
Liquid  
Handler

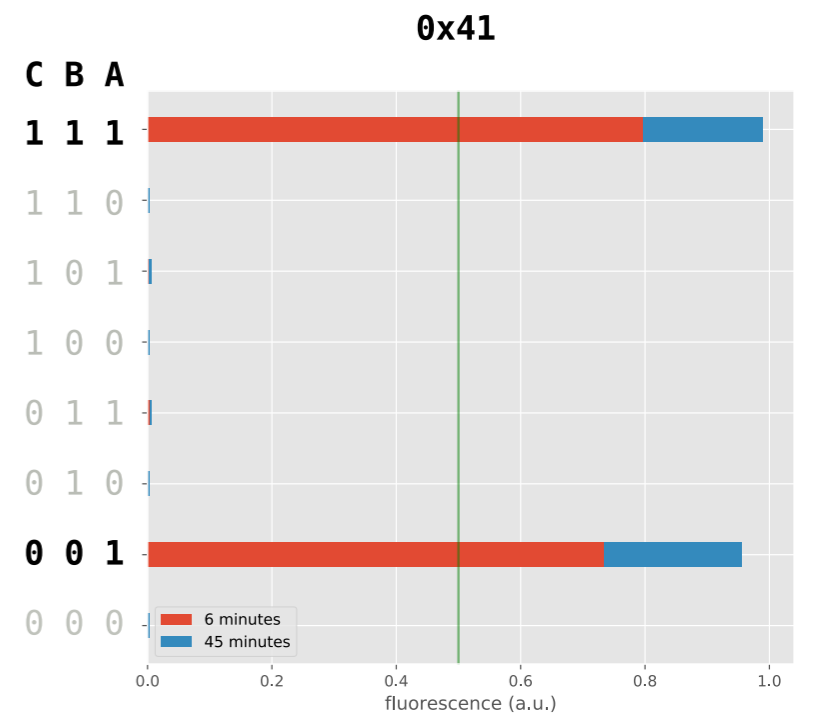
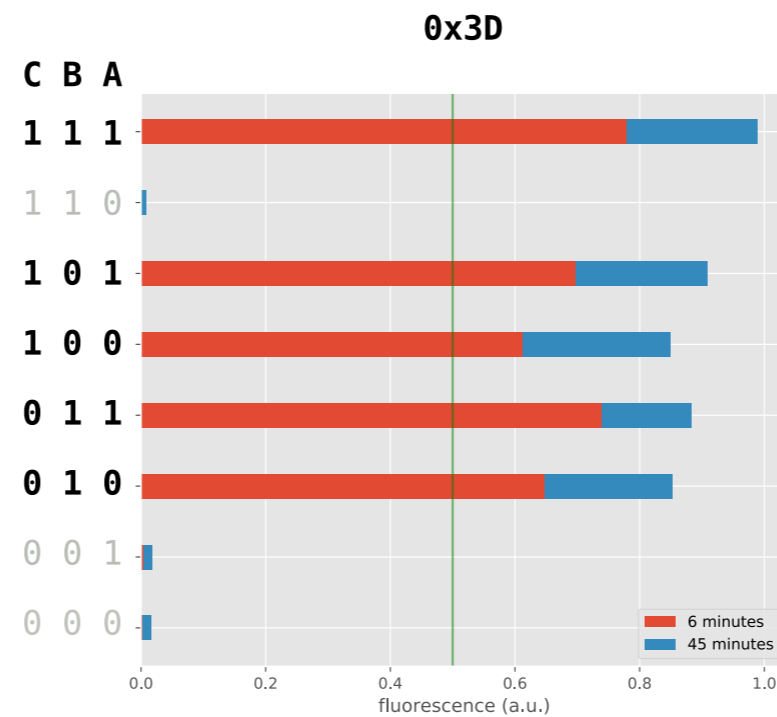
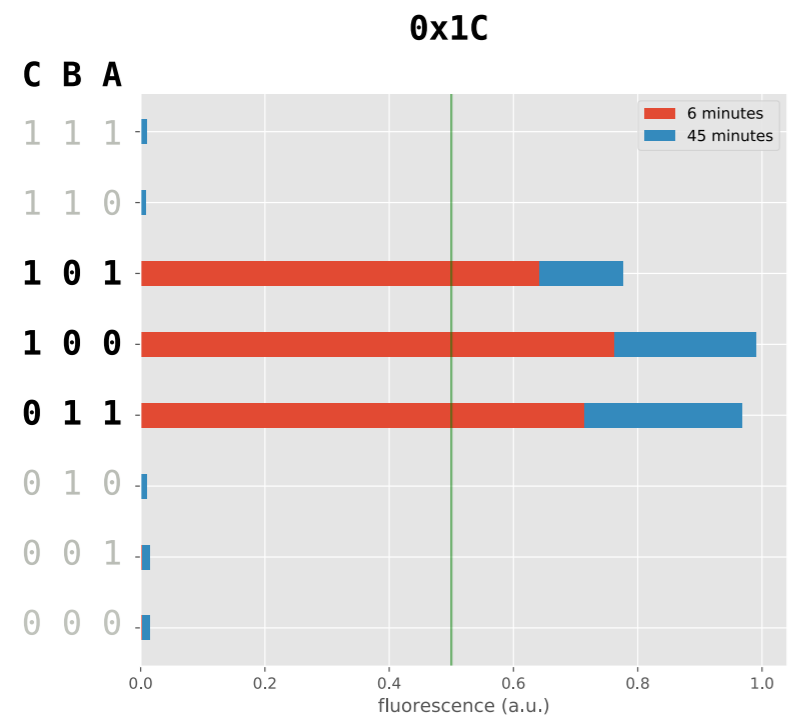
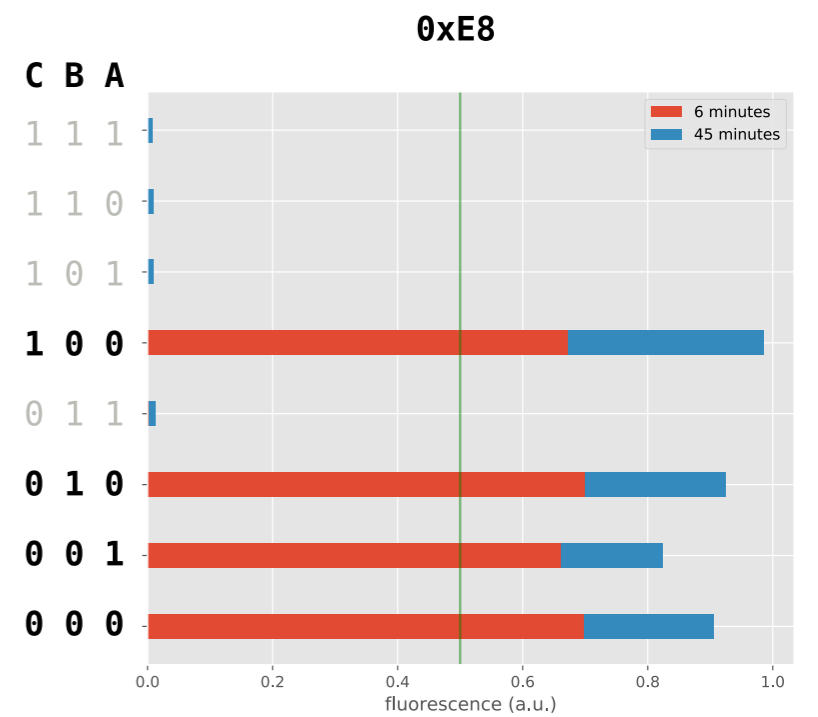
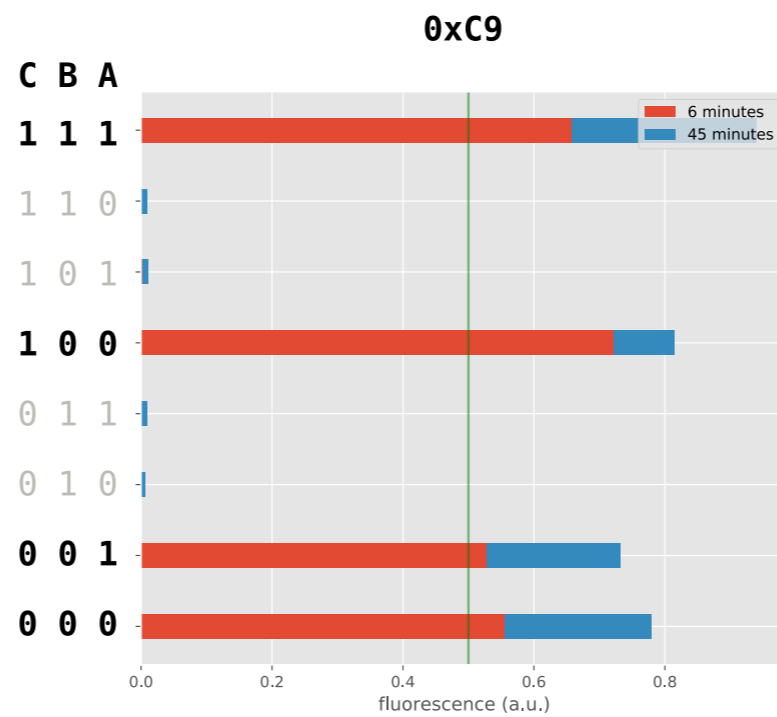
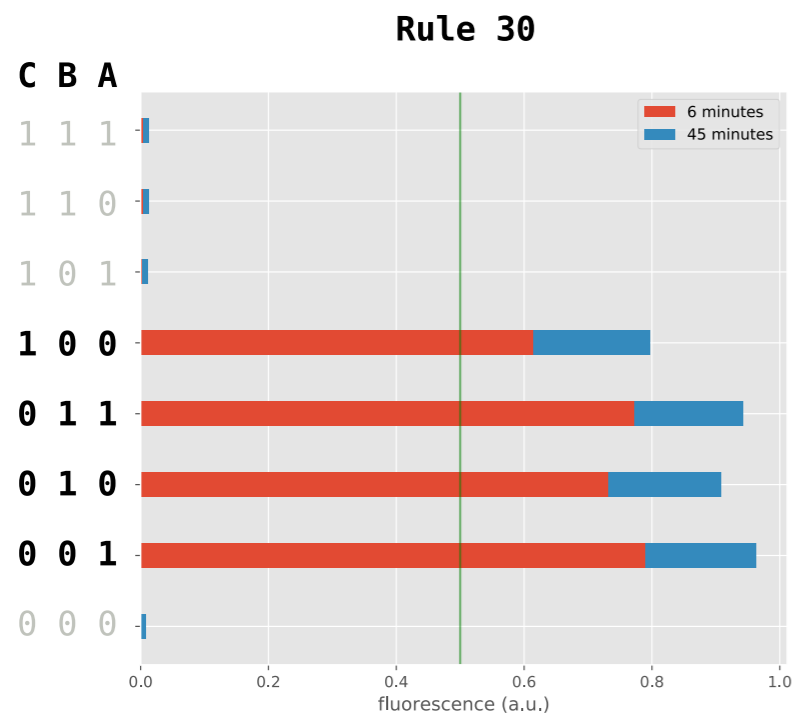
Breadboard compiler  
produces  
a mixing protocol

All 8 input combinations for 6 circuits



Destination plate

# First measurement 6 minutes after mixing start time

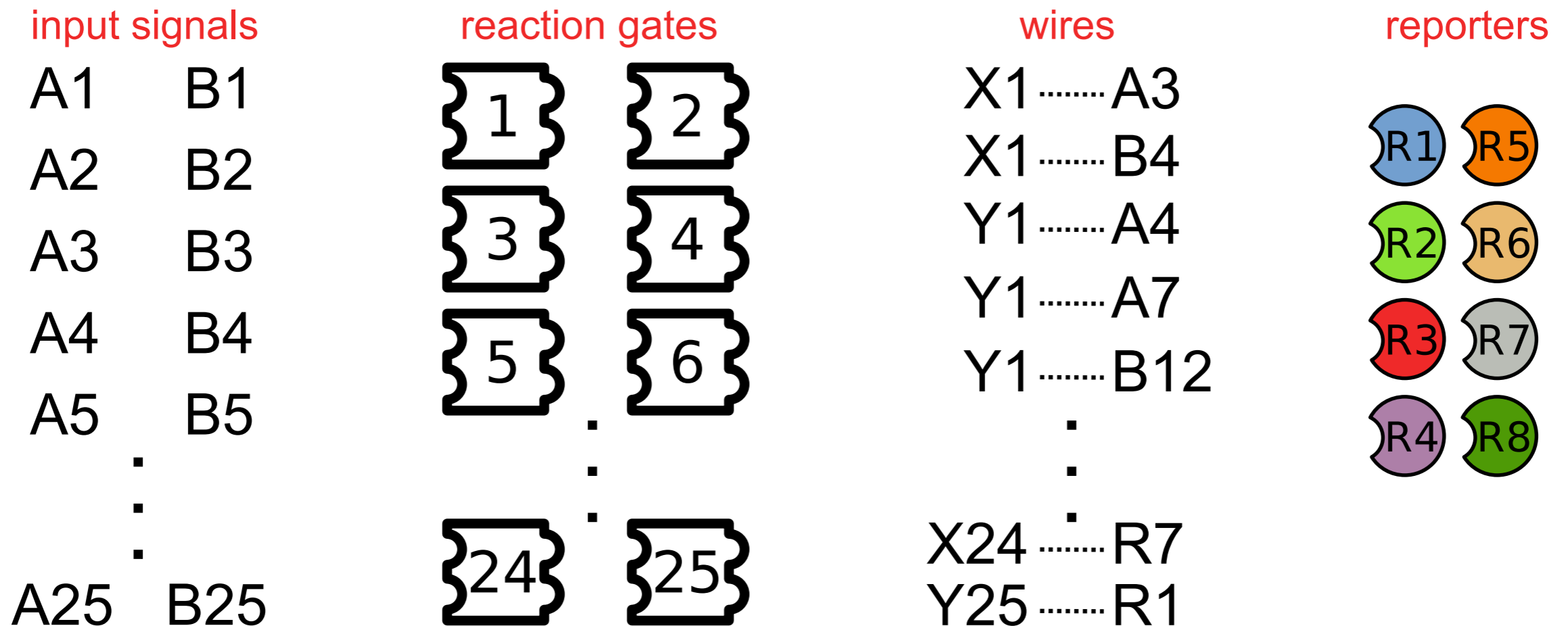


# Molecular Circuit Breadboard

## Roadmap

# Molecular Breadboard 2.0:

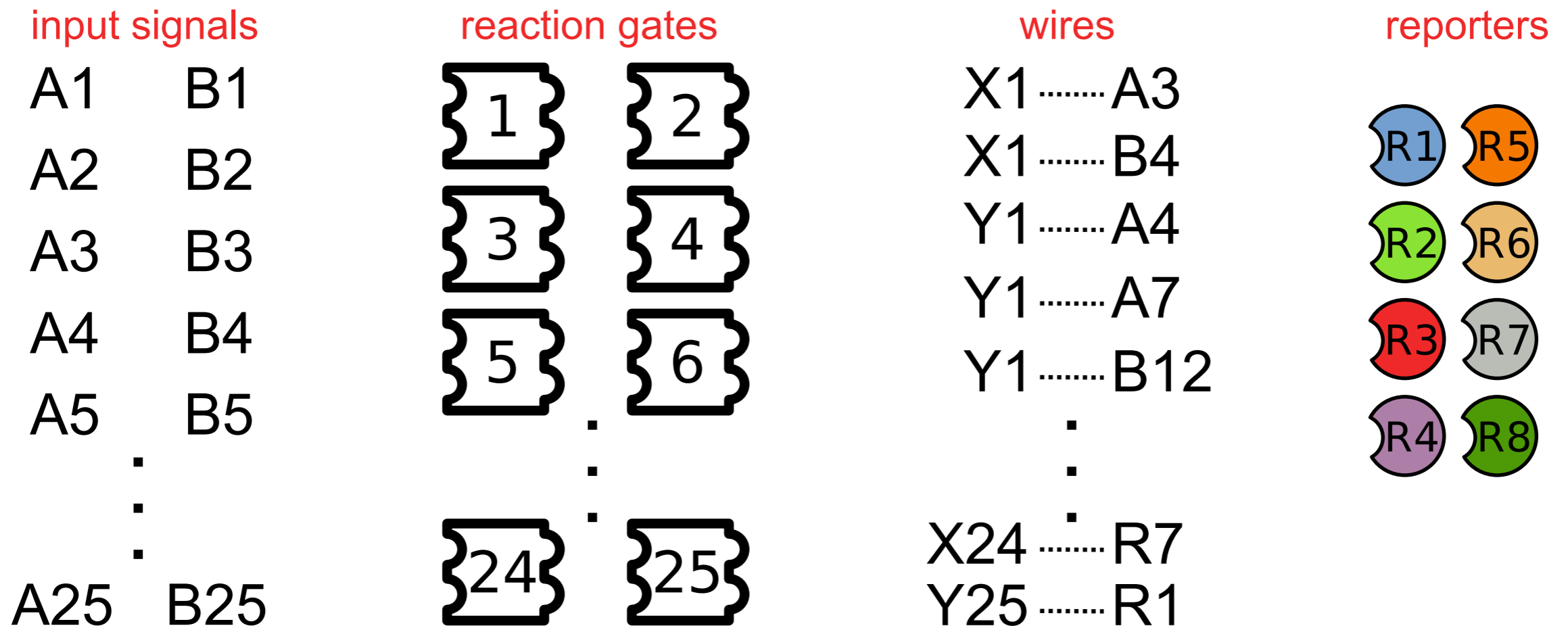
More components





# Molecular Breadboard 2.0:

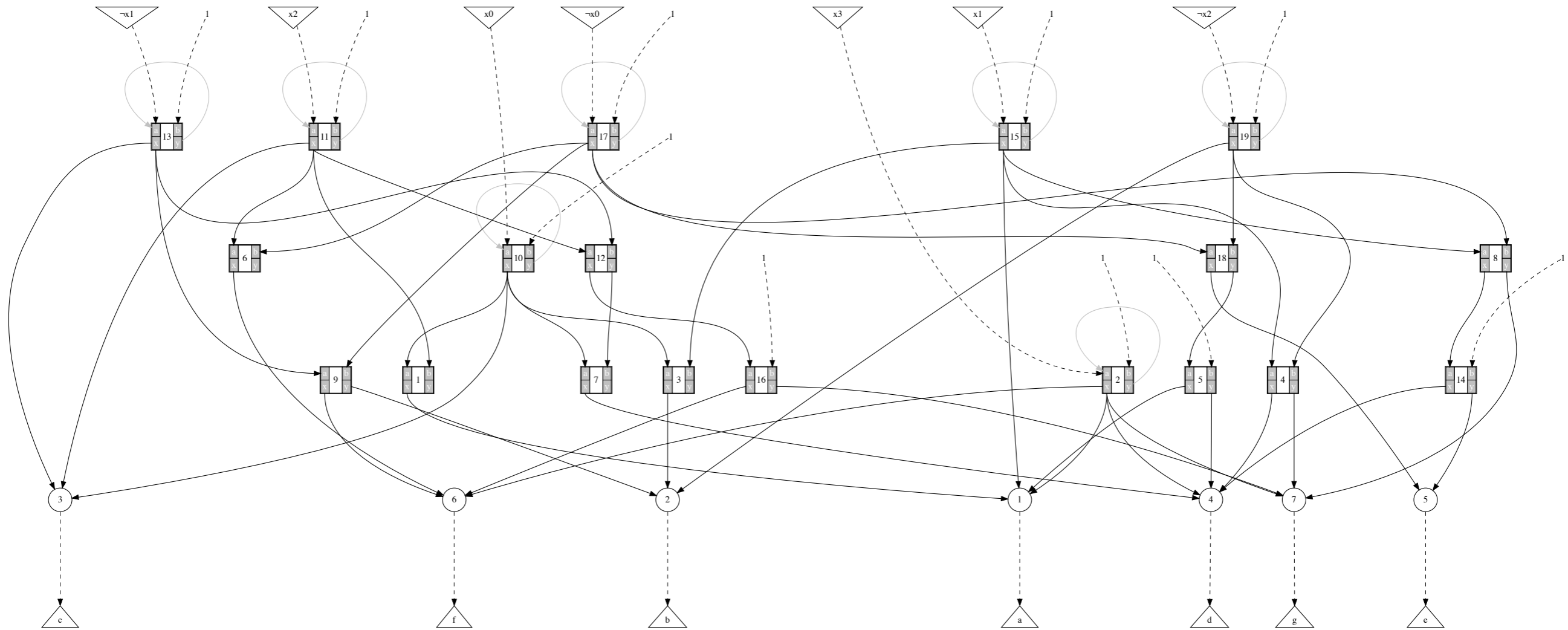
More circuits



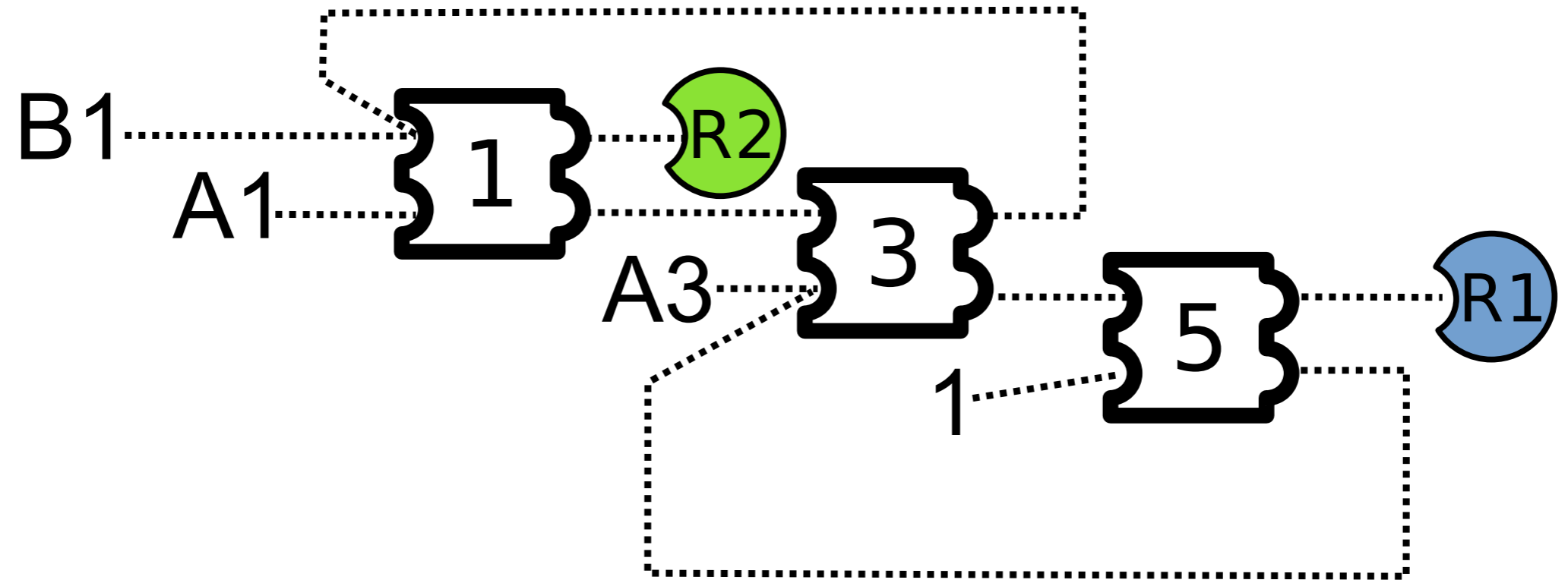
Breadboard 2.0 can  
realize > 130 K circuits

# Molecular Breadboard 2.0:

Larger circuits



# Building circuits with feedback loops

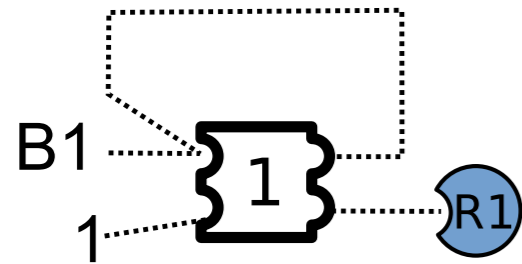


**Chemical Reaction Networks**

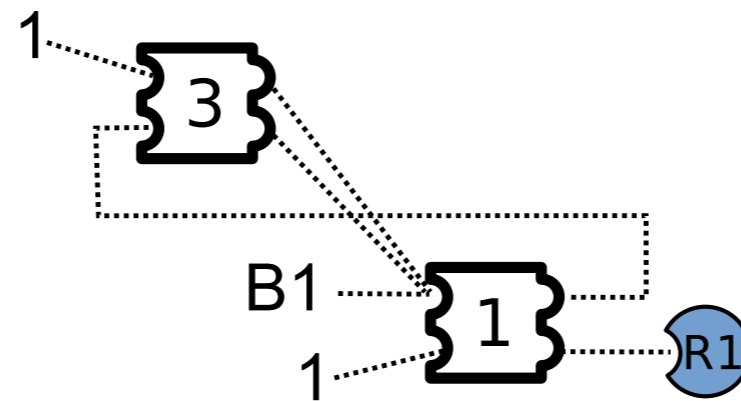
**Asynchronous Sequential Logic Circuits**

└─ **Finite state machines**

# Providing input amplifiers & output signal restoration



**Linear input amplifier**



**Exponential input amplifier**

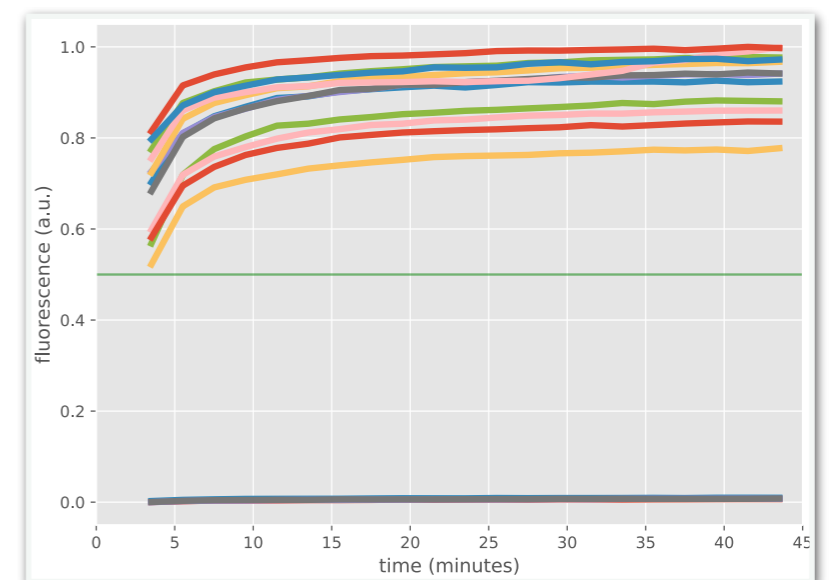
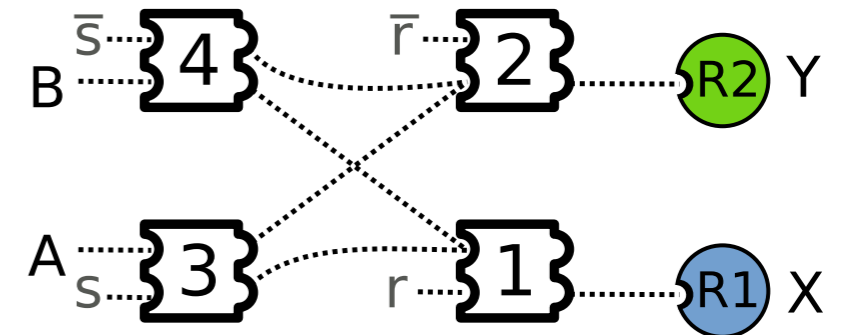
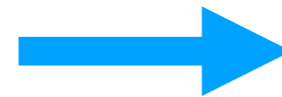
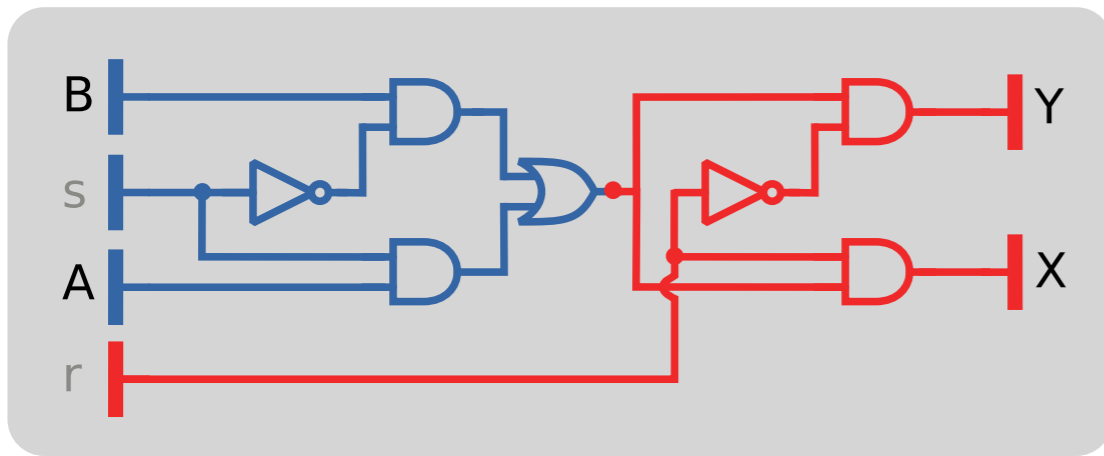


**Output signal restoration**

Increased speed

Robustness to error

Automation



# Related talks & posters @ DNA 24

Dominic Scalise, Nisita Dutta and [Rebecca Schulman](#)  
DNA strand-displacement buffers

Si-Ping Han, Lisa Scherer, Matt Gethers, Marwa Ben Hadj Salah, Rebecca Mancusi, Sahil Sagar, Robin Hu, Julia Derogatis, Ya-Huei Kuo, Guido Marcucci, John Rossi and William A. Goddard III  
Development and optimization of strand displacement based conditional small interfering RNAs for operation inside mammalian cells

[Eyal Nir](#), Yaron Berger and Miran Liber  
Computer Controlled DNA Bipedal Walker that Perform Several Steps a Minute

Abhinav Singh and [Manoj Gopalkrishnan](#)  
EM Algorithm with DNA Molecules

Wooli Bae, [Thomas Ouldridge](#) and [Guy-Bart Stan](#)  
Autonomous generation of multi-stranded RNA complexes for synthetic molecular circuits

Yan Shan Ang and Lin-Yue Lanry Yung  
Design of Split Proximity Circuit as a Plug-and-Play Translator for Discriminating Single Nucleotide Mutation

Yan Shan Ang and Lin-Yue Lanry Yung  
Dynamically Elongated Association Toehold for Tuning Circuit Kinetics and Thermodynamics

Patrick Irmisch and Ralf Seidel  
Modelling DNA-strand displacement reactions in the presence of base-pair mismatches

Boya Wang and [David Soloveichik](#)  
Experimentally characterizing the design space of strand displacement translators with toehold-size clamps

Allison Tai and Anne Condon  
Error-free stable computation with stack-supplemented chemical reaction networks

Kevin Cherry, Gokul Gowri and Lulu Qian  
DNA-based neural networks that learn from their molecular environment

Robert F. Johnson and [Erik Winfree](#)  
Using Bisimulation for Verification of Polymer Reaction Networks

# Acknowledgments

- **Winfree lab** (Caltech)
- **Soloveichik lab** (University of Texas at Austin)
- **Qian lab** (Caltech)
- **Murray lab** (Caltech)
- Thanks to DNA 24 organizers for the invitation



# Tools discussed in tutorial

## **ABC: logic synthesis and verification**

<https://people.eecs.berkeley.edu/~alanmi/abc>

## **VisualDSD**

<https://lepton.research.microsoft.com/webdna>

## **Nuskell compiler framework**

<https://github.com/DNA-and-Natural-Algorithms-Group>

## **DSD breadboard**

<http://dsdbreadboard.org> *(online later this year)*