# Algebras and Languages for Molecular Programming


Luca Cardelli

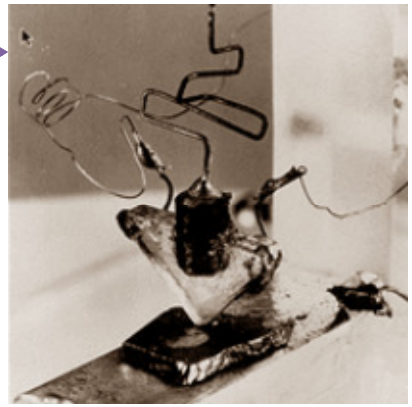
Microsoft Research


Oxford, 2010-02-12

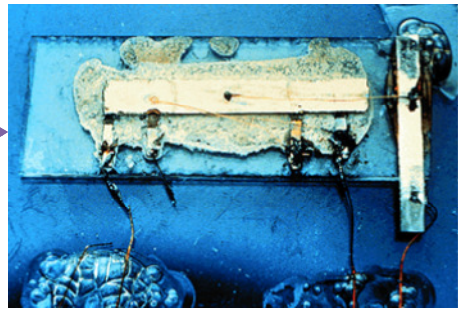
http://lucacardelli.name

# Smaller and Smaller

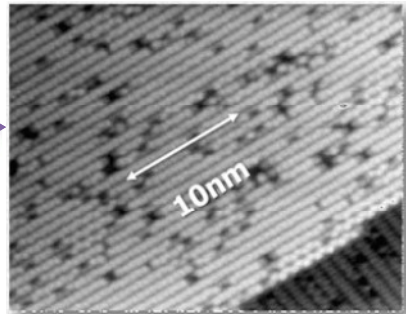
Dec. 23, 1947. John Bardeen and Walter Brattain show the first working transistor.



September 1958. Jack Kilby builds the first integrated circuit.



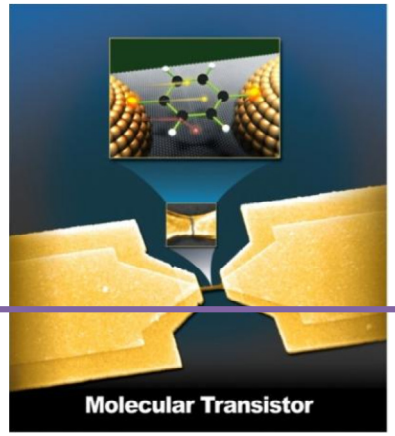
Jan 30, 2010. Intel and Micron announce 25nm NAND flash.



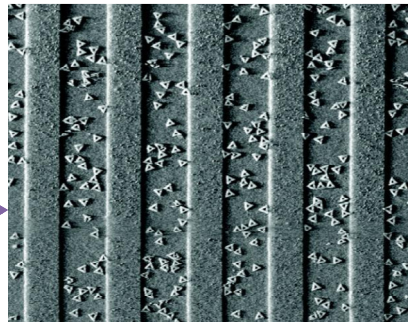

Scanning tunneling microscope image of a silicon surface showing 10nm is ~20 atoms across

Dec. 24, 2009. Working transistor made of a single molecule.

Observation of molecular orbital gating. *Nature*, 2009; 462 (7276): 1039



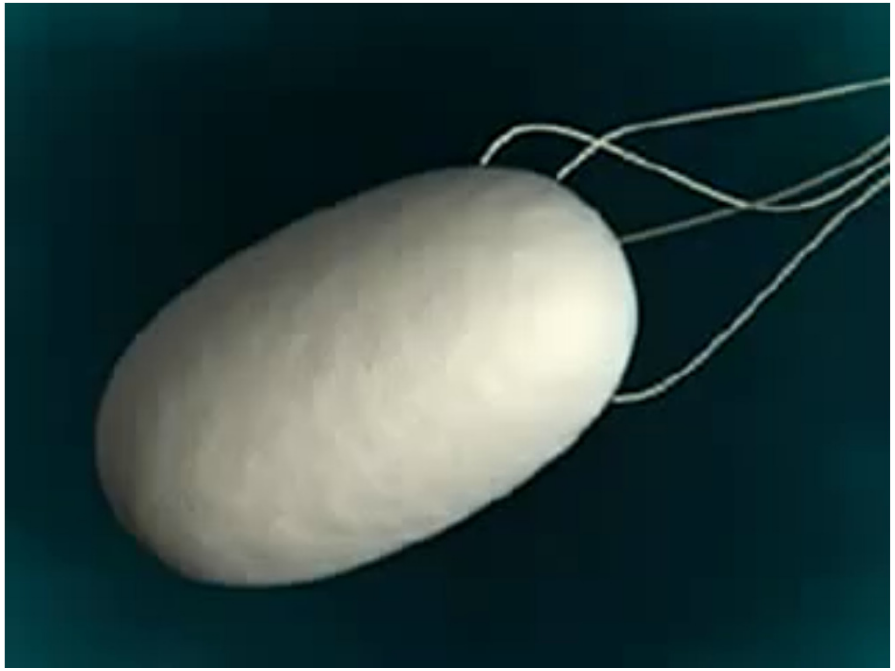

The race is on for *molecular scale integrated circuits*.



Placement and orientation of individual DNA shapes on lithographically patterned surfaces. Nature Nanotechnology 4, 557 - 561 (2009).

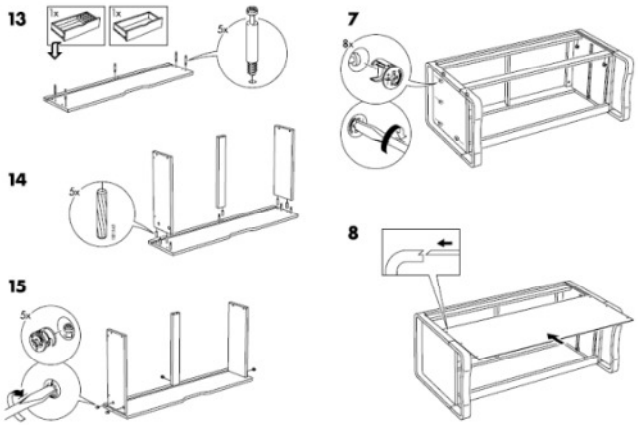
# Building The Small*est* Things

- How do we build structures that are by definition smaller than your tools?
- Basic answer: you can't. Structures (and tools) should build themselves!
- By *programmed self-assembly*.

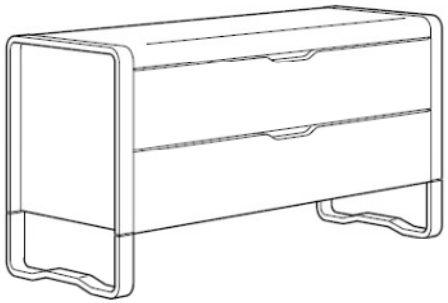


# Molecular IKEA

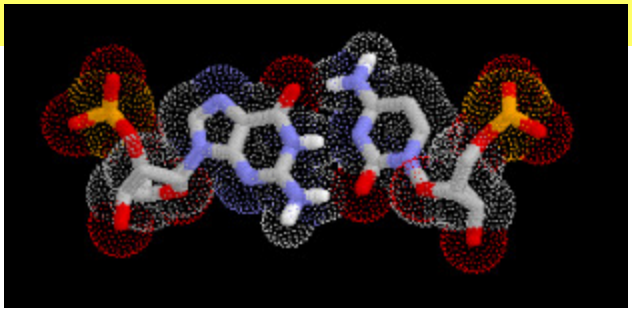
- Nature can self-assemble.
  Can we?
- *"Dear IKEA, please send me a chest of drawers that assembles itself."*
- We need a magical material where the pieces are pre-programmed to fit into to each other.
- At the molecular scale many such materials exist; let's pick one…



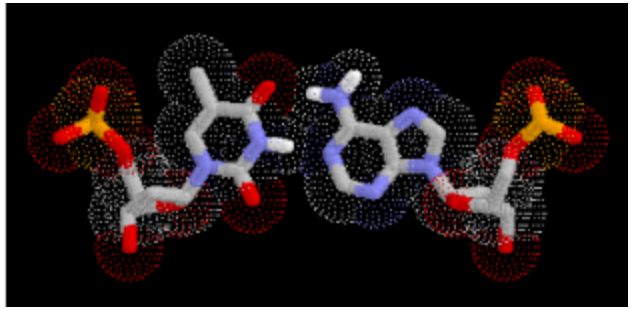

Add water



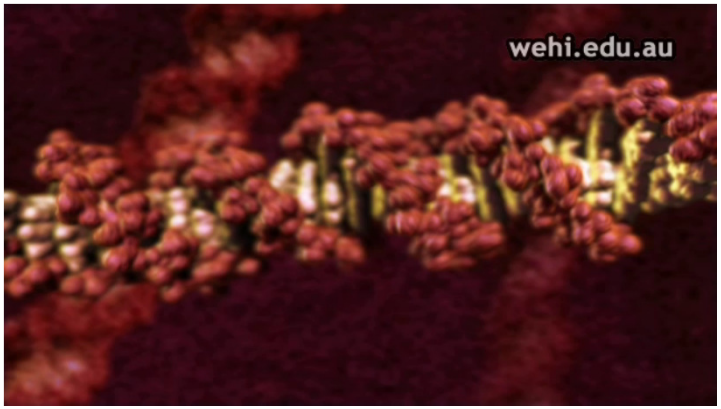
# DNA

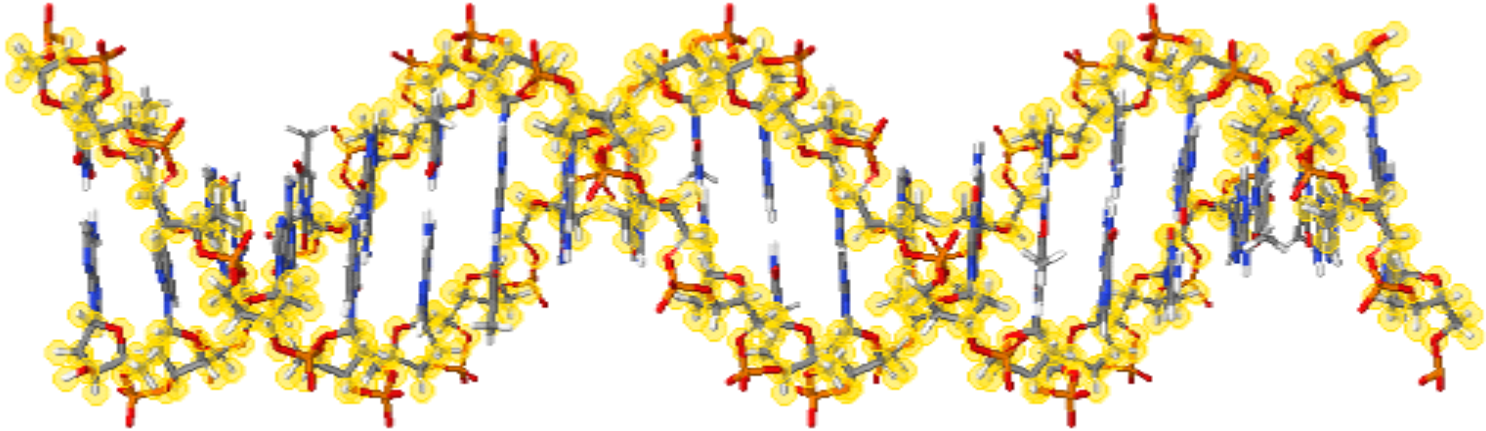


GC Base Pair
Guanine-Cytosine



TA Base Pair
Thymine-Adenine



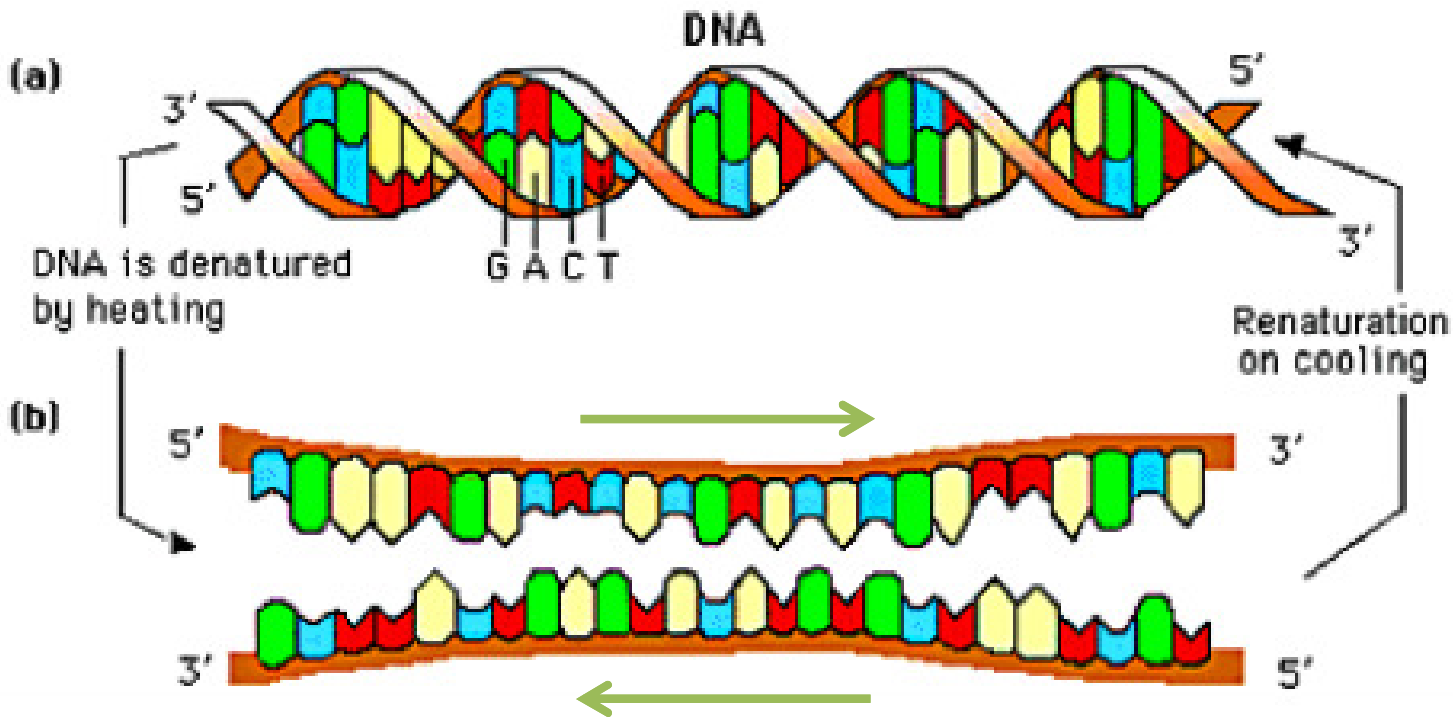



Sequence of Base Pairs (GACT alphabet)

[Interactive DNA Tutorial](http://www.biosciences.bham.ac.uk/labs/minchin/tutorials/dna.html)
(http://www.biosciences.bham.ac.uk/labs/minchin/tutorials/dna.html)

# ssDNA

Double-stranded DNA




Single-stranded DNA has an orientation
Each strand spells a GACT sequence
The two strands have *opposite* orientations

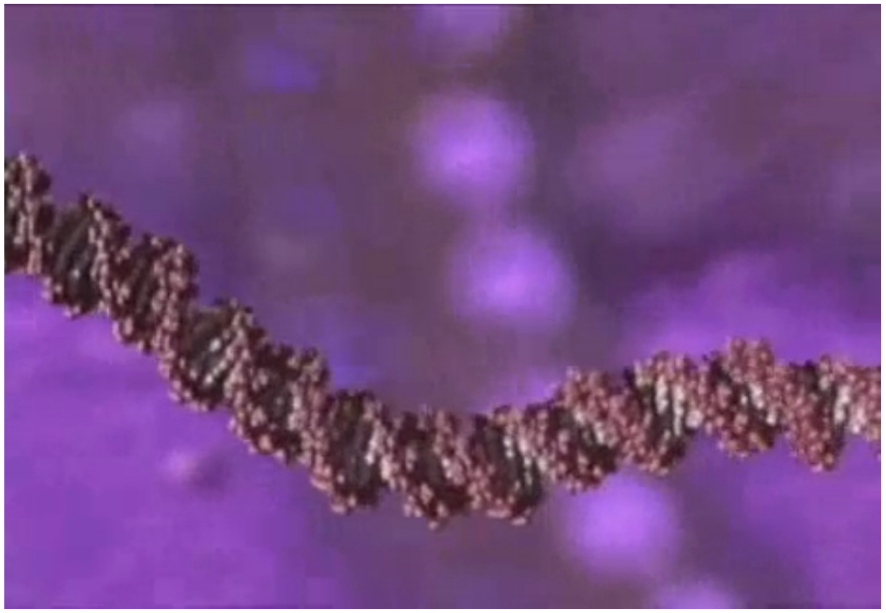
# Robust, and *Long*

- DNA in each human cell:
  - 3 billion base pairs
  - 2 meters long, 2nm thick
  - folded into a 6μm ball
  - 750 MegaBytes

- A huge amount for a cell
  - Every time a cell replicates it has to copy *2 meters of DNA* reliably.
  - To get a feeling for the scale disparity, compute:

- DNA in human body
  - 10 trillion cells
  - 133 Astronomical Units long
  - 7.5 OctaBytes

- DNA in human population
  - 20 million light years long

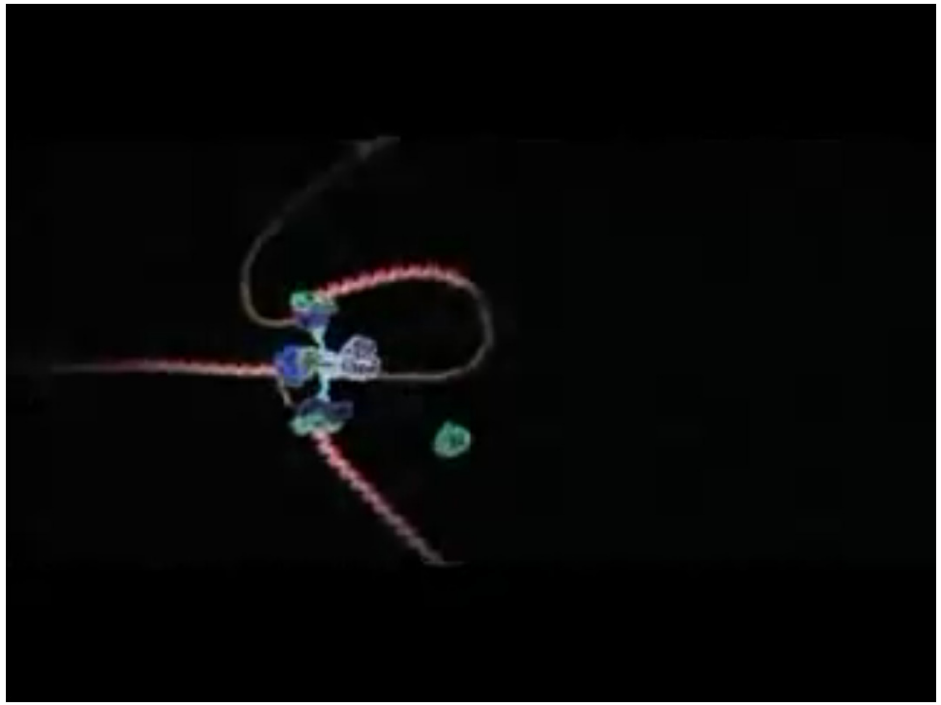


DNA wrapping into chromosomes

Andromeda Galaxy
2.5 million light years

# Zipping Along

- DNA can support structural and computational complexity.



DNA replication in *real time*

In Humans: 50 nucleotides/second
Whole genome in a few hours (with parallel processing)

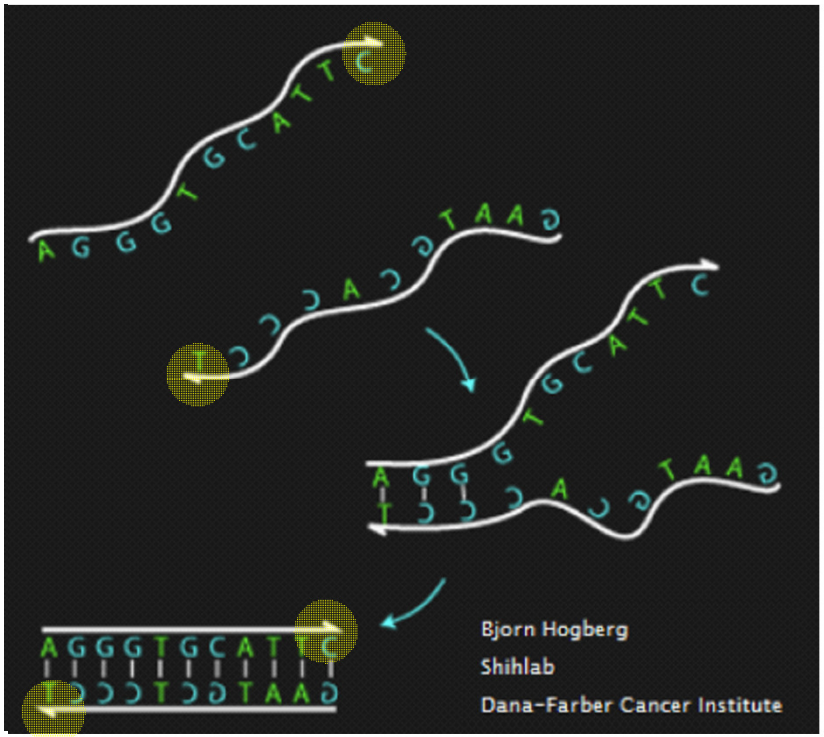
In Bacteria: 1000 nucleotides/second
(higher error rate)

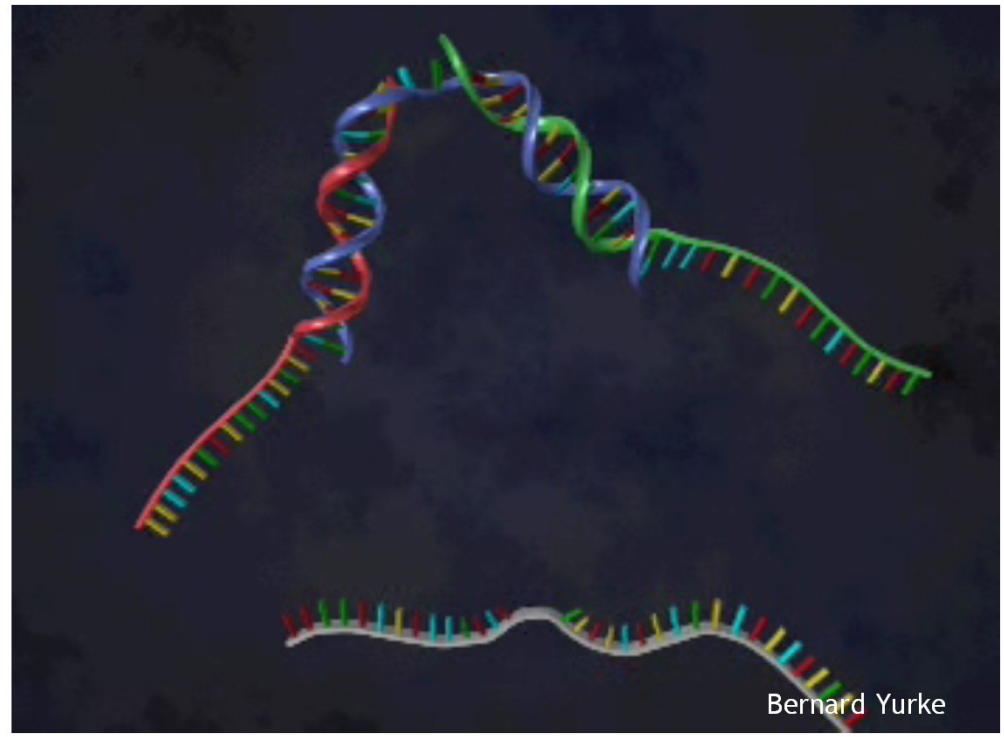
DNA transcription in *real time*

RNA polymerase II: 15-30 base/second

Drew Berry
http://www.wehi.edu.au/wehi-tv

# Hybridization

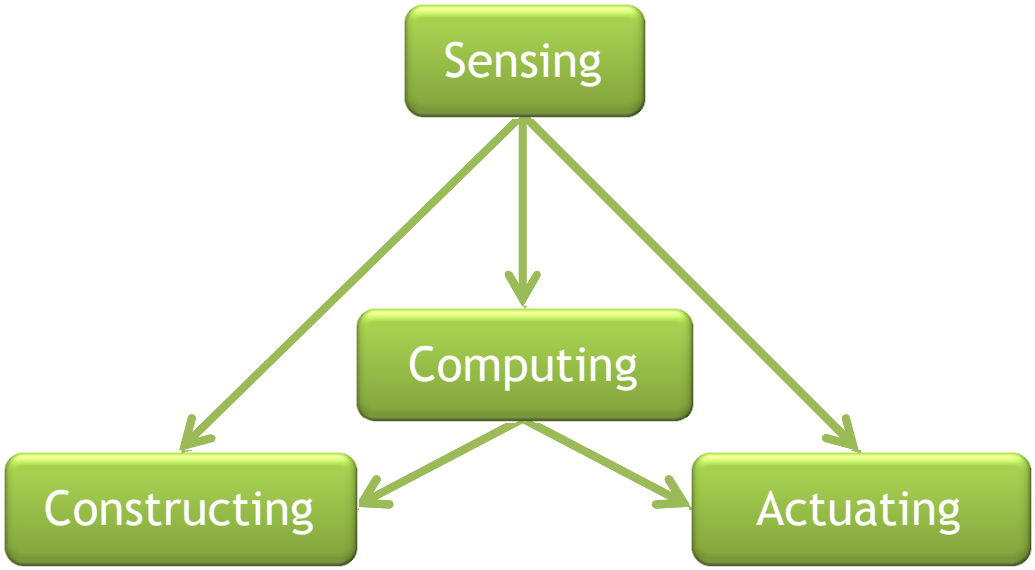



- Strands with opposite orientation and complementary base pairs stick to each other (Watson-Crick duality).
- This is all we are going to use
  - We are not going to exploit DNA replication, transcription, translation, restriction and ligation enzymes, etc., which enable other classes of tricks.

# Nanoscale Engineering

- Sensing
  - Reacting to forces
  - Binding to molecules
- Actuating
  - Releasing molecules
  - Producing forces
- Constructing
  - Chassis
  - Growth
- Computing
  - Signal Processing
  - Decision Making




Nucleic Acids (DNA/RNA) can do all this, and interface to biological structures.

# Compositionality

- Sensors and Actuators at the 'edge' of the system
  - They can use disparate kinds of inputs (sensors) and outputs (actuators)

- The 'kernel' of the system computes
  - Must use uniform inputs and outputs

- Compositionality in the kernel
  - Supporting 'arbitrary' computing complexity
  - The output of each computing components must be the same kind of 'signal' as the input

  - If the inputs are voltages, the outputs must be voltages
  - If the inputs are proteins, the outputs must be proteins
  - If the outputs are photons the inputs must be photons
  - If the inputs are DNA, the outputs must be DNA

- Central design question
  - What should our signals (not components!) be?
  - Design components that manipulate those signals.

# What does DNA Compute?

- Electronics has *electrons*
  - All electrons are the same
  - All you can do is see if you have *few* ('False') or *lots* ('True') of electrons
  - Hence Boolean logic is at the basis of digital circuit design
  - Symbolic and numeric computation has to be encoded above that
  - But mostly we want to compute with symbols and numbers, not with Booleans

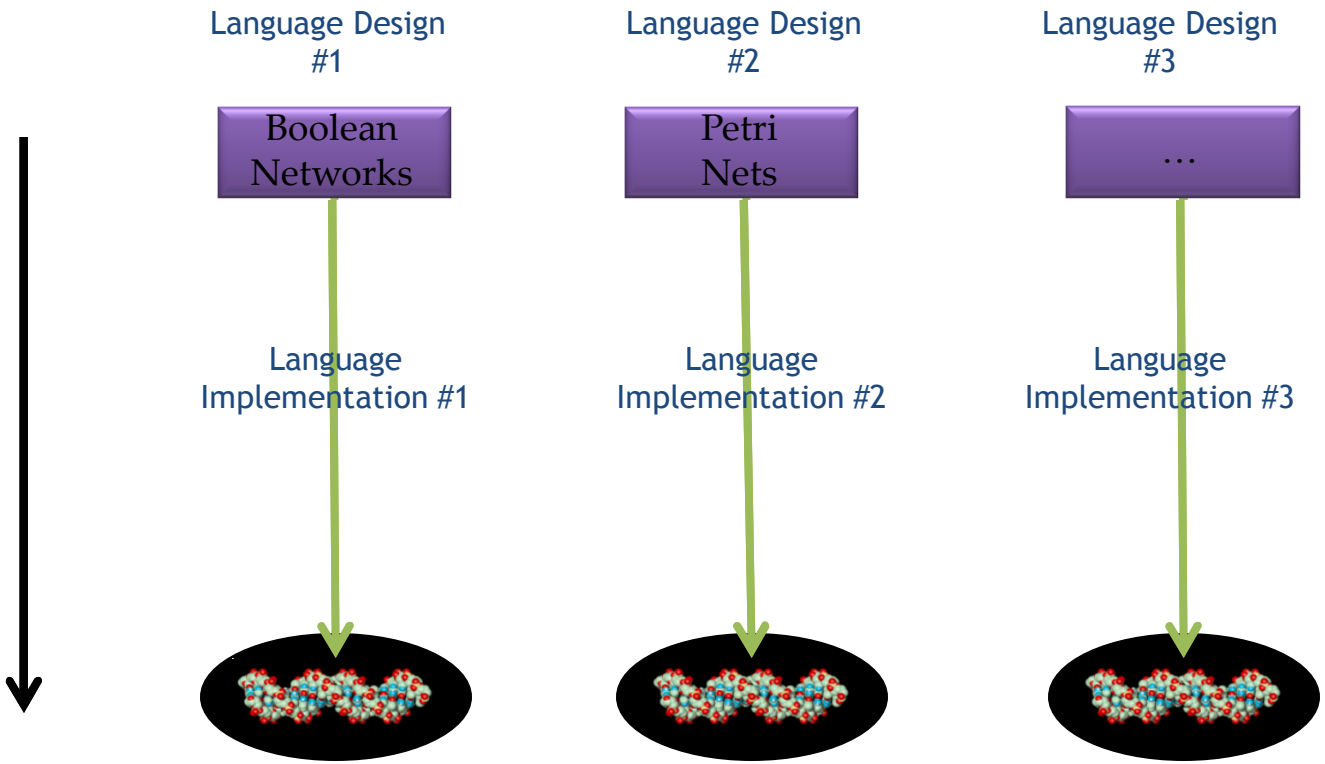
- DNA computing has *symbols* (DNA words)
  - DNA words are not all the same
  - Symbolic computation can be done *directly*
  - We can also directly use molecular concurrency

- Process Algebra as the 'Boolean Algebra' of DNA Computing
  - What are the 'gates' of symbolic concurrent computation?
  - That's what Process Algebra is about
  - (Process Algebra comes from the theory of concurrent systems)

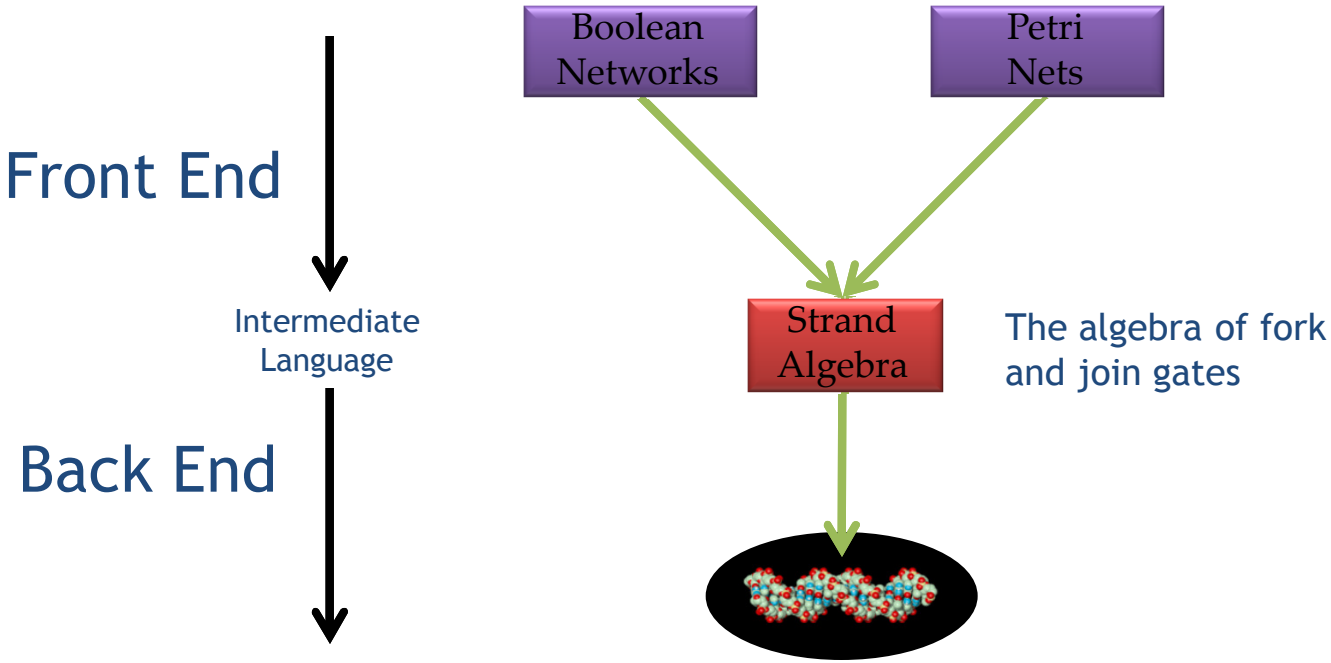
# Implementing "Arbitrary" Computing Functions

# Compilers

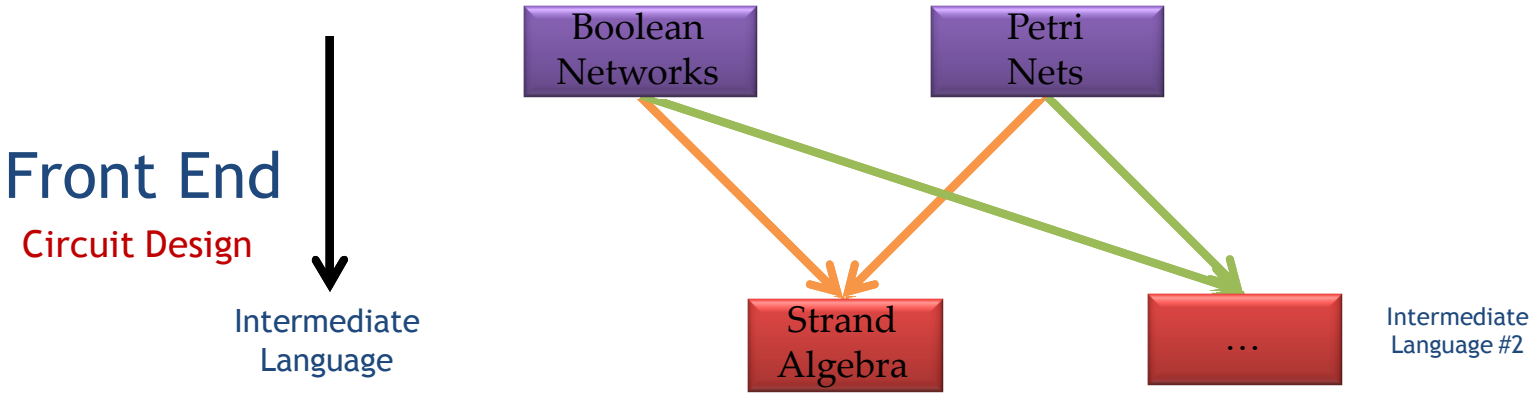
Monolithic Compilers



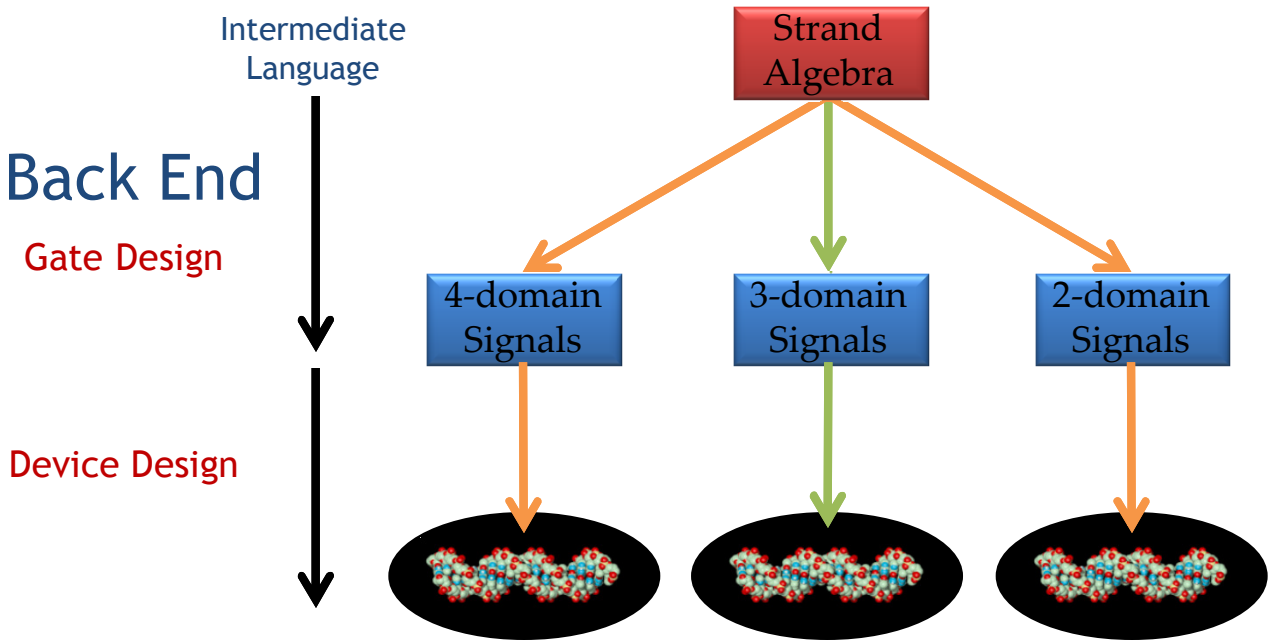
# Intermediate Languages



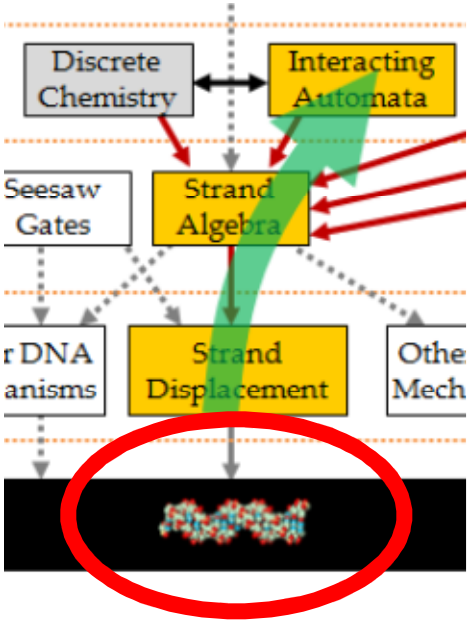
# Front Ends


Front End
Circuit Design
Intermediate Language
Boolean Networks
Petri Nets
Strand Algebra
...
Intermediate Language #2

# Back Ends


Intermediate Language
Back End
Gate Design
Device Design
Strand Algebra
4-domain Signals
3-domain Signals
2-domain Signals

# Toehold Mediated Strand Displacement



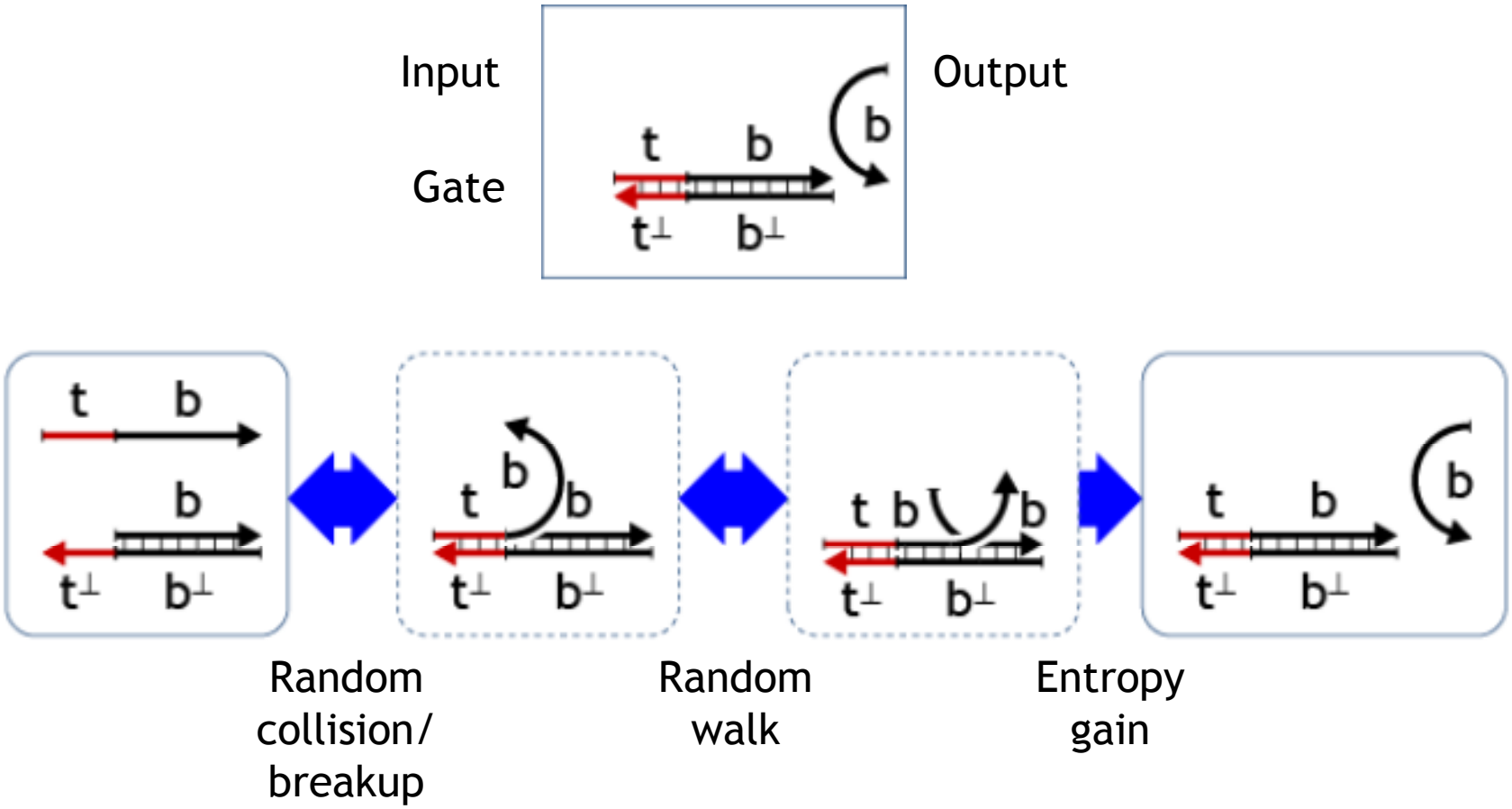
# Rules of the Game

- Short complementary segments hybridize reversibly


- Long complementary se rsibly

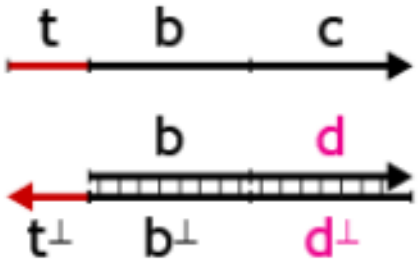
# DNA Strand Displacement

- Short strand (toehold): reversible binding
- Long strand (body): irreversible binding

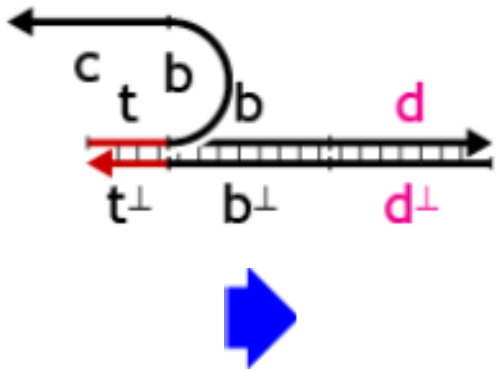


# Failed Strand Displacement

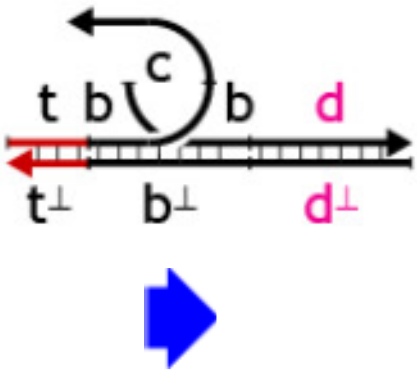
- What if the input does not match the gate?



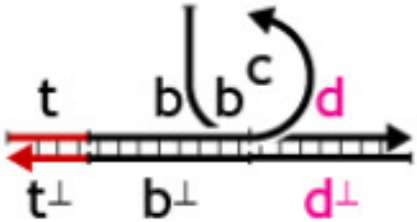
# Failed Strand Displacement



# Failed Strand Displacement

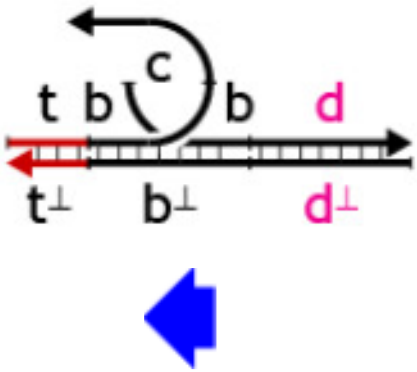


# Failed Strand Displacement

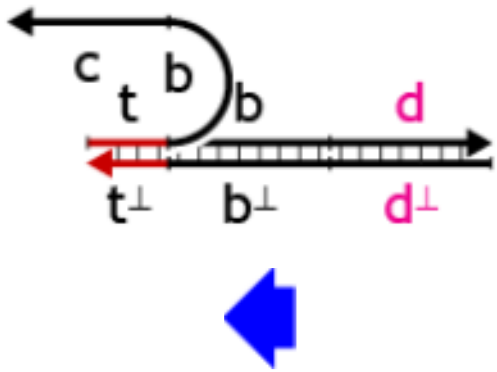



?

# Failed Strand Displacement

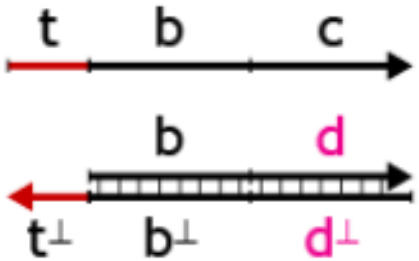


# Failed Strand Displacement



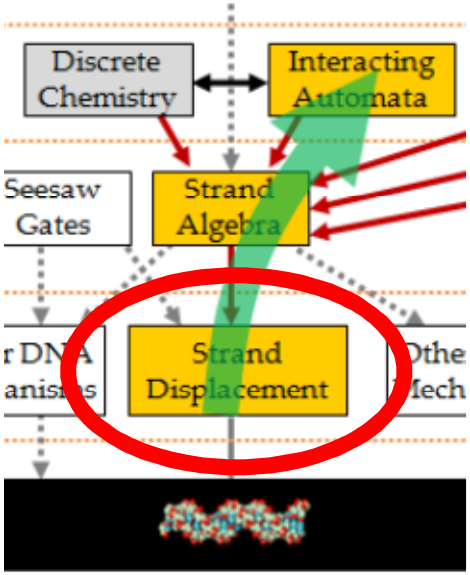
# Failed Strand Displacement

- Hence an incorrect binding will undo
  - That's why toeholds must bind reversibly




- Matching depends on the long segment only
  - Strand displacement succeeds iff the whole long segment matches
  - The address space is determined by the size of the long segment, which is unbounded (not by the size of the toehold)
  - The toehold is just a 'cache' of the address

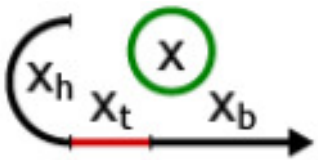
# Strand Displacement Signals and Gates



# Signals

- A signal is the representation of an abstract event
  - E.g. generated by a sensor
  - E.g. accepted by an effector
  - We are not limited to true/false

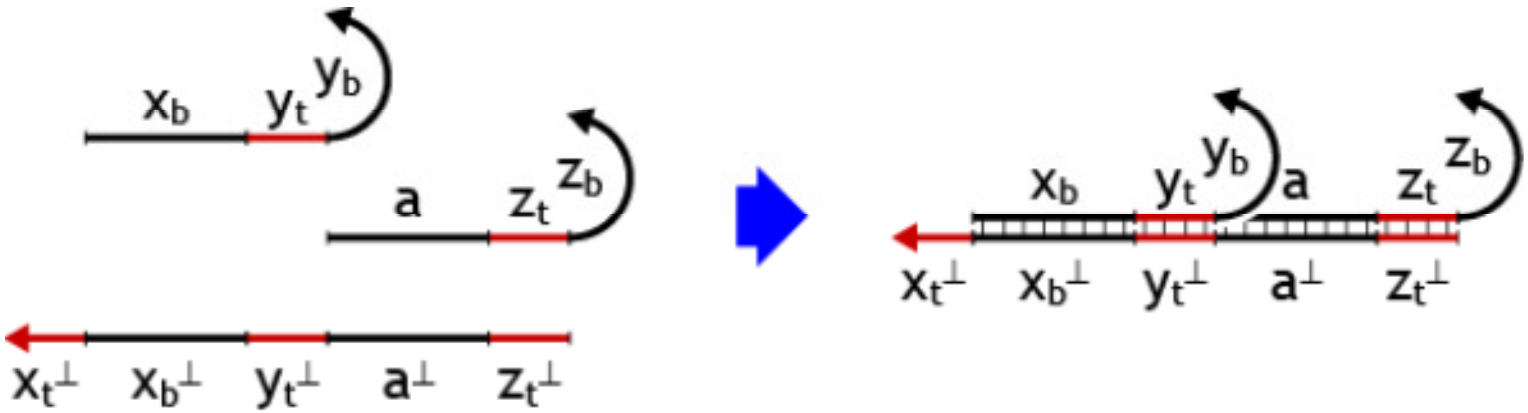
- 3-domain signals
  - $x_h$: hystory (ignore)
  - $x_t$: toehold (binding)
  - $x_b$: body (recognition)




- Signals (single stranded DNA) are prepared by (artificial) DNA synthesis

# Gates

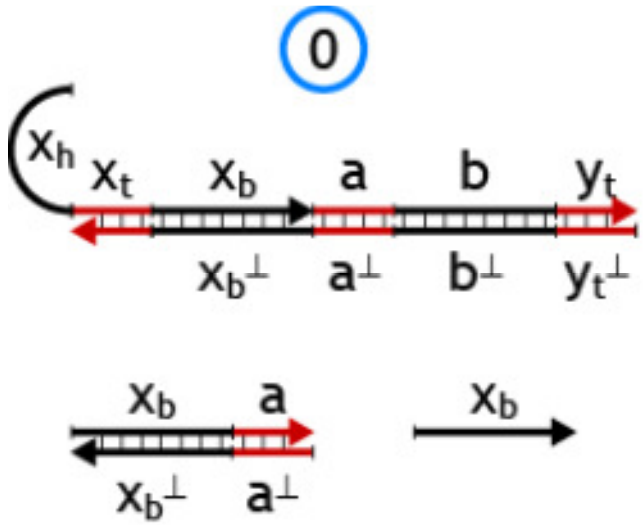
- Double-stranded structures with free toeholds




- Gates are prepared by self-assembly from single-stranded DNA that is synthesized

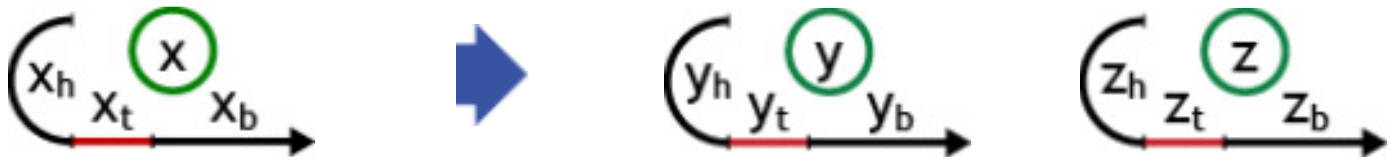
# Waste

A system is considered *inert* (terminated) if it has no free toeholds.



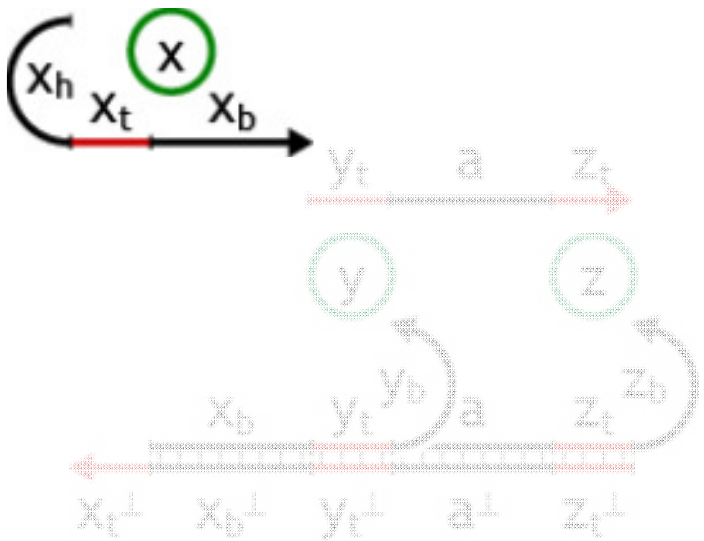
# Fork Gate

- $x \rightarrow y + z$



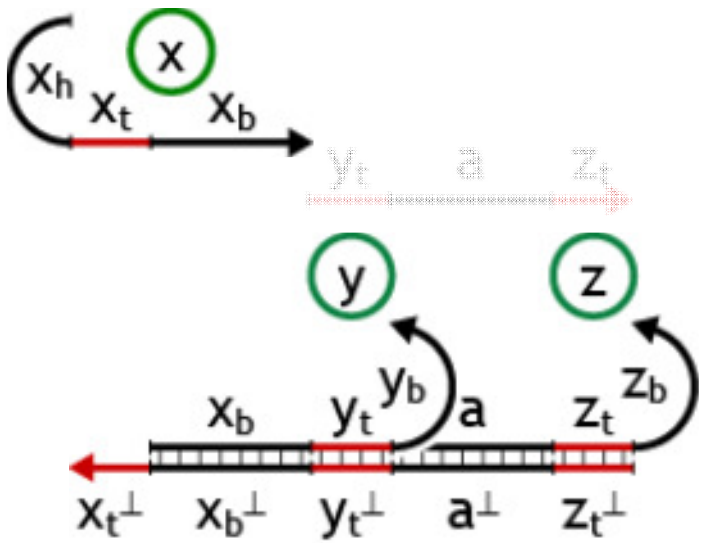

- $x \rightarrow y + 0$ transform x to y (transducer)
- $x \rightarrow x + y$ linear production of y (catalyst)
- $x \rightarrow x + x$ exponential production of x (amplifier)

# Fork Gate

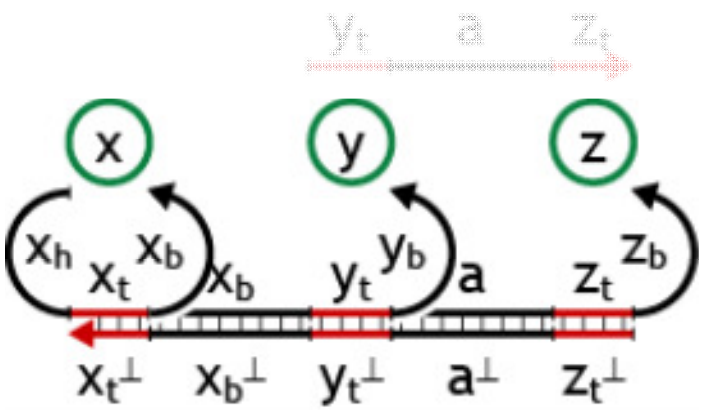



This is the Fork Gate structure

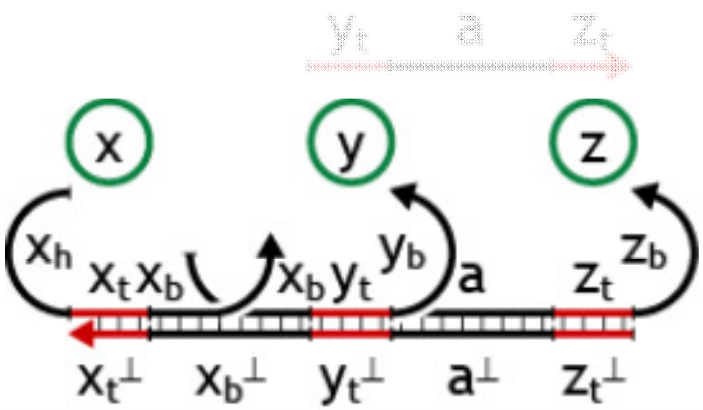
# Fork Gate


$x_h$
$x$
$x_t$
$x_b$
$y_t$
$a$
$z_t$
$y$
$z$
$y_b$
$z_b$
$x_b$
$y_t$
$a$
$z_t$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$a^\perp$
$z_t^\perp$

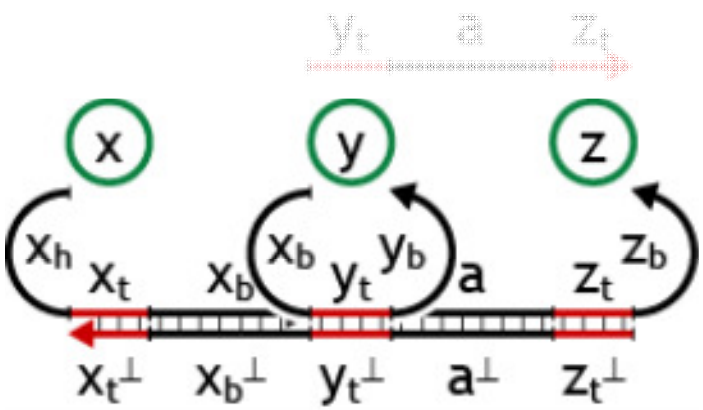
# Fork Gate



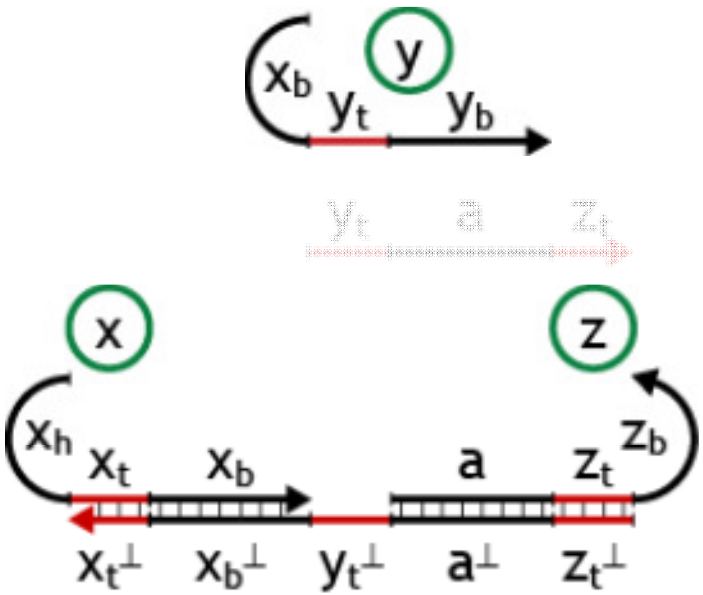
# Fork Gate



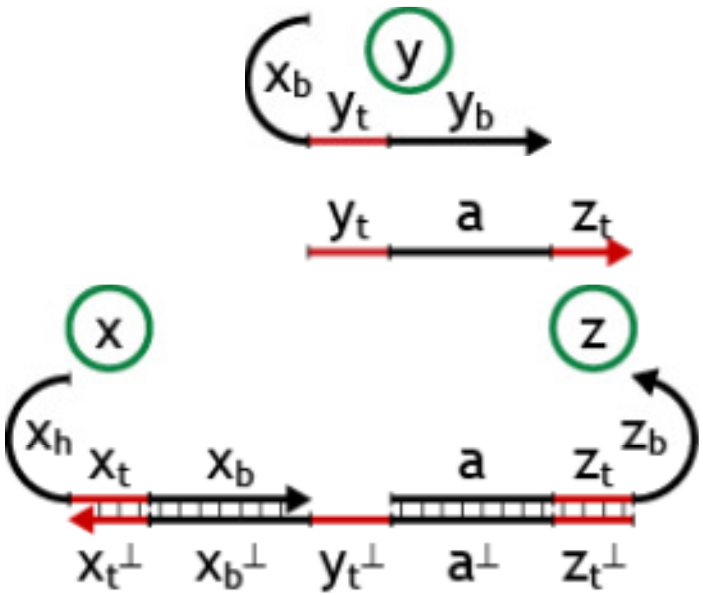
# Fork Gate



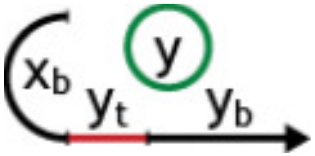
# Fork Gate

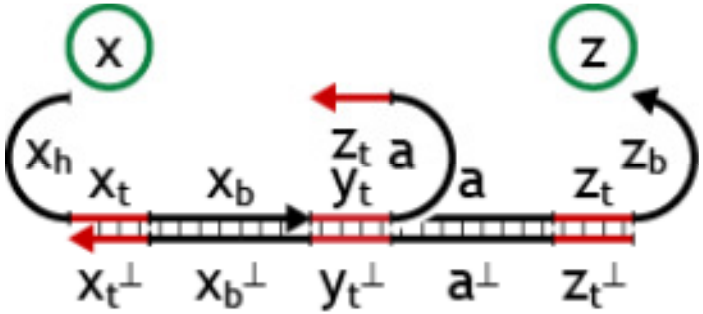

y
$x_b$
$y_t$
$y_b$
$y_t$
a
$z_t$
x
z
$x_h$
$x_t$
$x_b$
a
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$a^\perp$
$z_t^\perp$

# Fork Gate

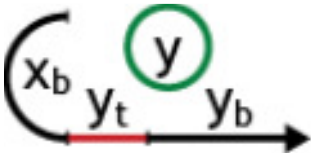

y
$x_b$
$y_t$
$y_b$
$y_t$
a
$z_t$
x
z
$x_h$
$x_t$
$x_b$
a
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$a^\perp$
$z_t^\perp$

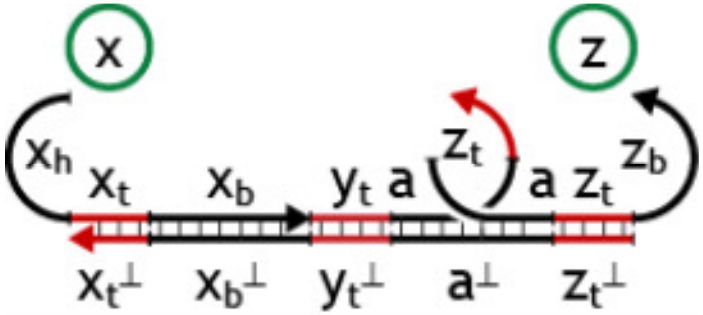
# Fork Gate


y
$x_b$
$y_t$
$y_b$

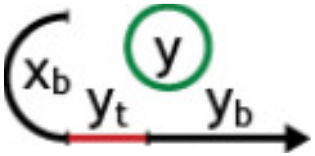

x
z
$x_h$
$x_t$
$x_b$
$z_t$
a
$y_t$
a
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$a^\perp$
$z_t^\perp$

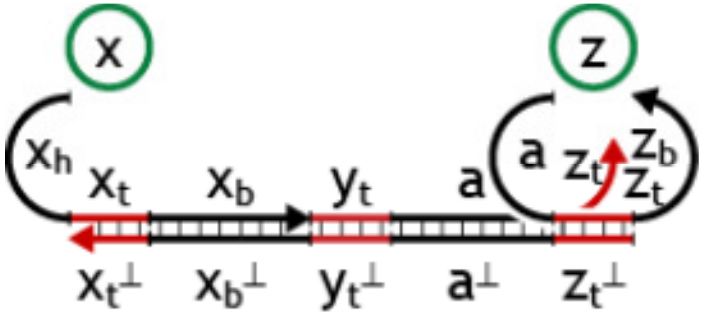
# Fork Gate


y
$x_b$
$y_t$
$y_b$

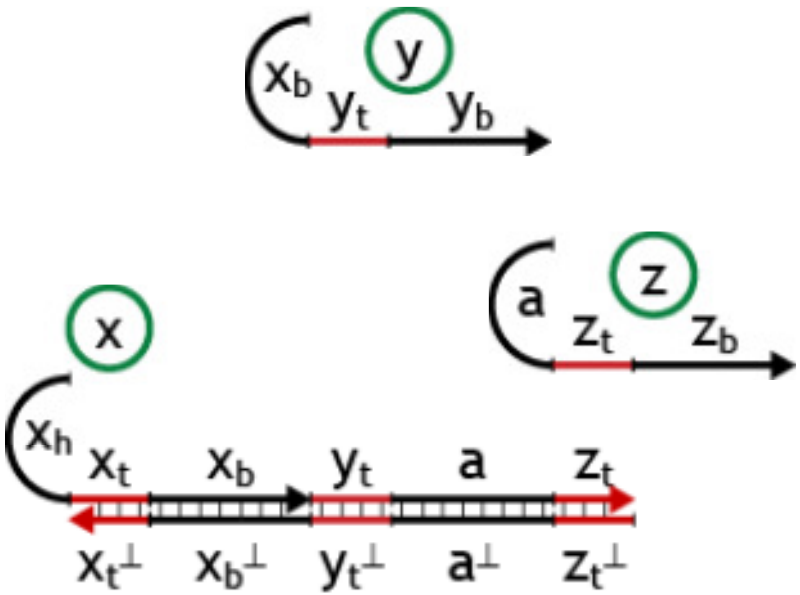

x
z
$x_h$
$x_t$
$x_b$
$y_t$
a
$z_t$
a
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$a^\perp$
$z_t^\perp$

# Fork Gate

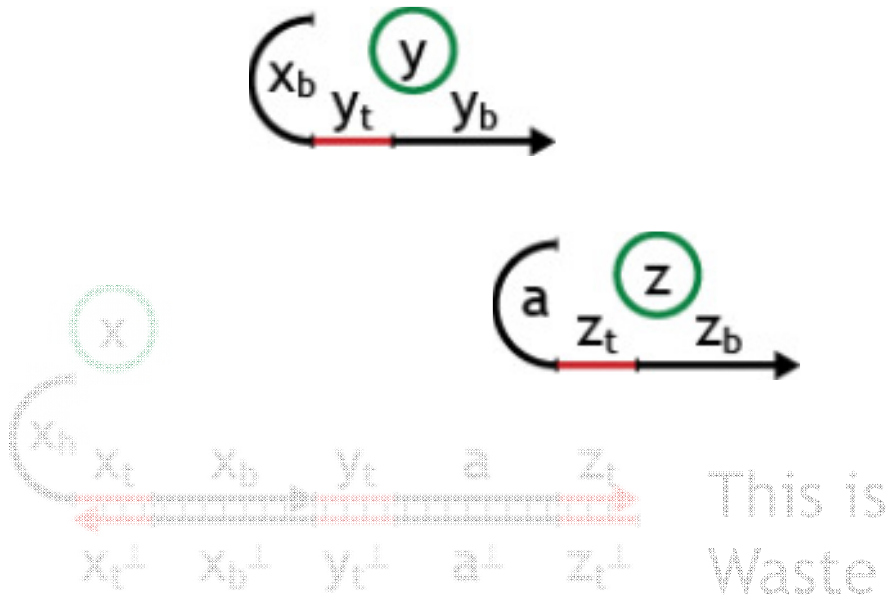

y
$x_b$
$y_t$
$y_b$


x
z
$x_h$
$x_t$
$x_b$
$y_t$
a
a
$z_t$
$z_b$
$z_t$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$a^\perp$
$z_t^\perp$

# Fork Gate

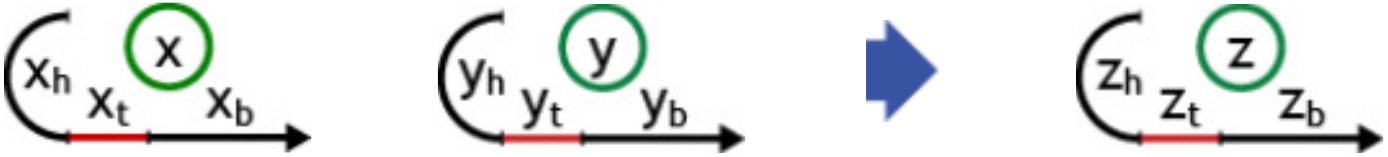

$x_b$
y
$y_t$
$y_b$
a
z
$z_t$
$z_b$
x
$x_h$
$x_t$
$x_b$
$y_t$
a
$z_t$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$a^\perp$
$z_t^\perp$

# Fork Gate

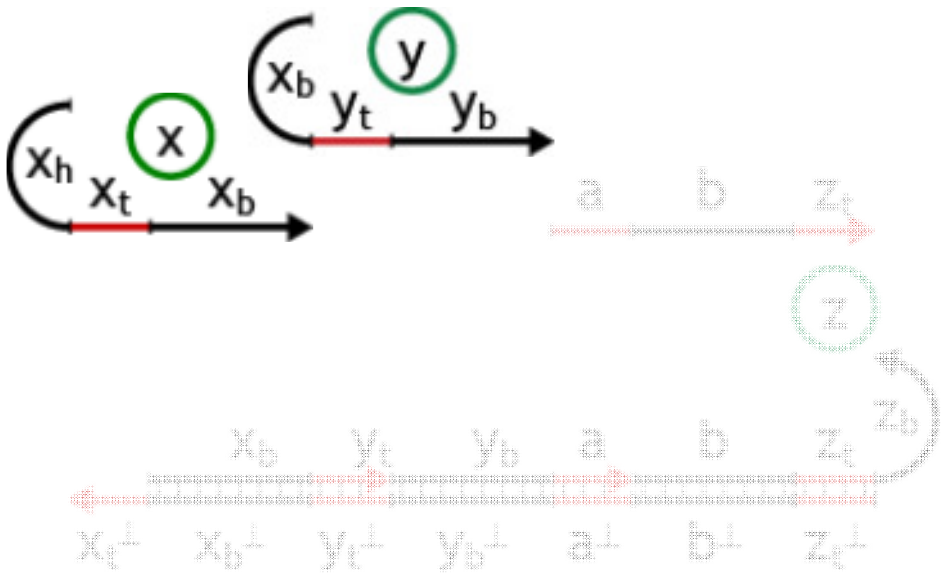

$x_b$
y
$y_t$
$y_b$
a
z
$z_t$
$z_b$
x
$x_b$
$x_t$
$x_b$
$y_t$
a
$z_t$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$a^\perp$
$z_t^\perp$
This is
Waste

# Join Gate

- $x + y \rightarrow z$

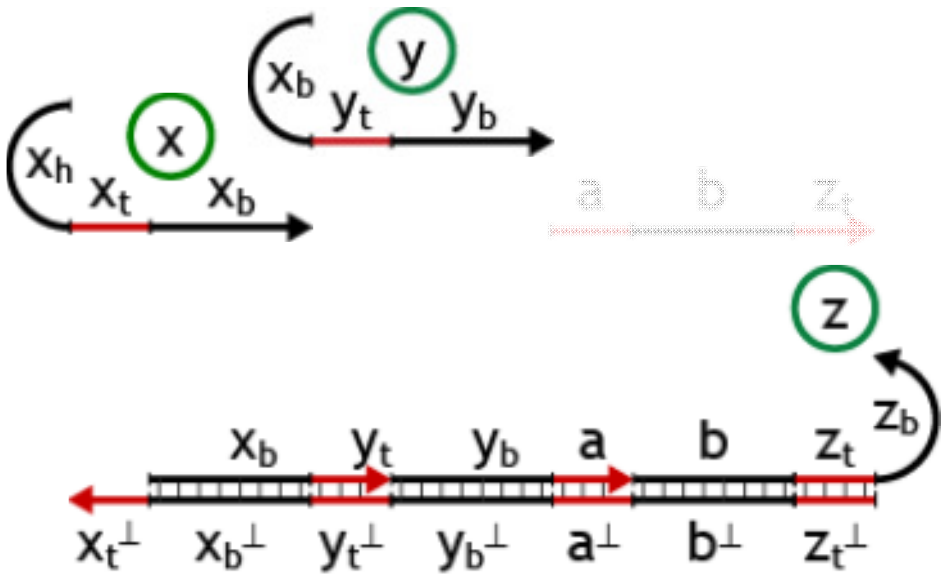


# Join Gate

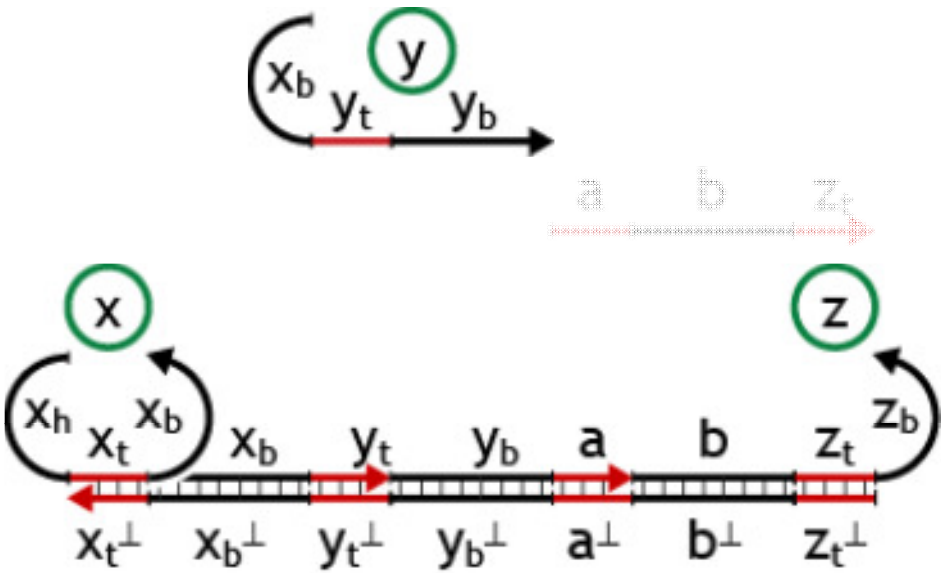



This is the Join Gate structure

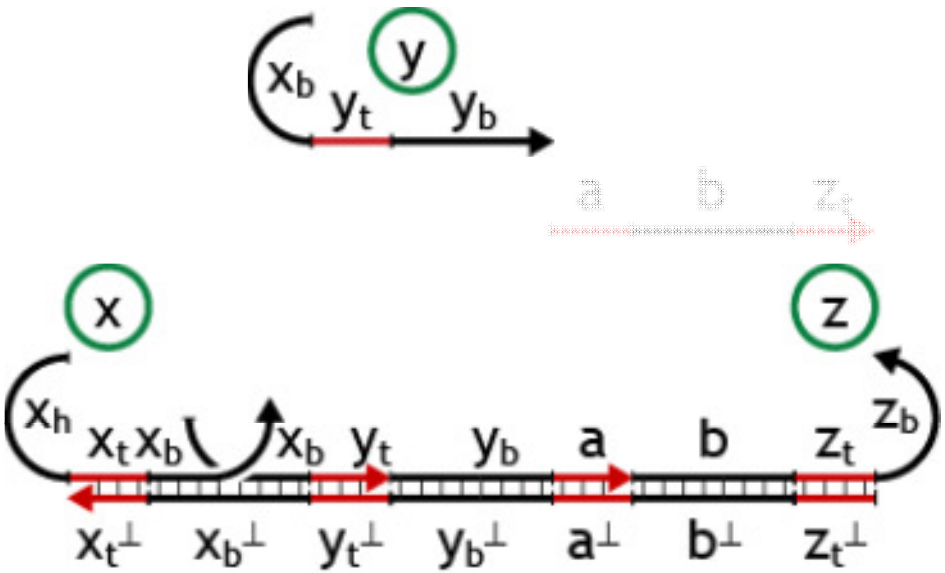
# Join Gate


x
y
z
$x_h$ $x_t$ $x_b$
$x_b$ $y_t$ $y_b$
a b $z_t$
$z_b$
$x_b$ $y_t$ $y_b$ a b $z_t$
$x_t^\perp$ $x_b^\perp$ $y_t^\perp$ $y_b^\perp$ $a^\perp$ $b^\perp$ $z_t^\perp$

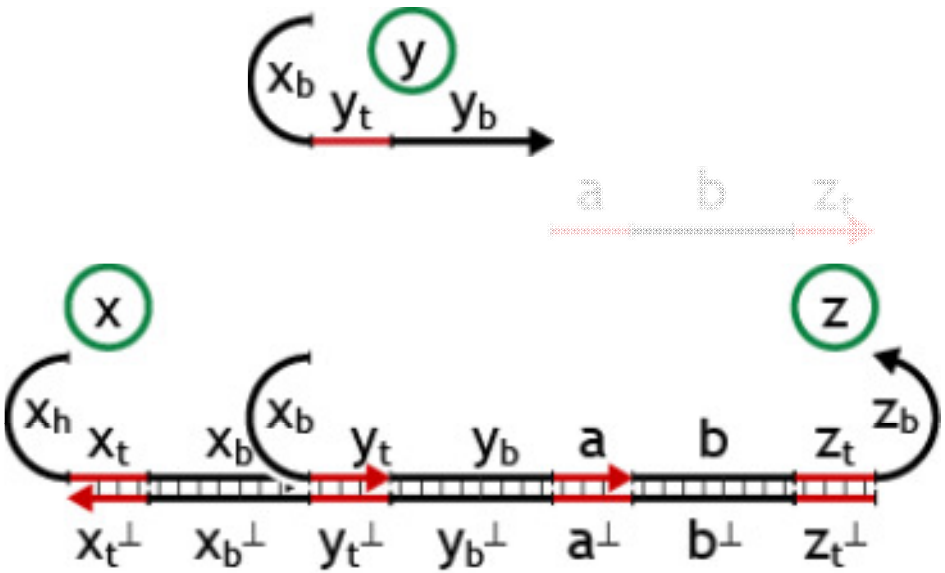
# Join Gate


$x_b$
y
$y_t$
$y_b$
a
b
$z_t$
x
z
$x_h$
$x_t$
$x_b$
$x_b$
$y_t$
$y_b$
a
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$

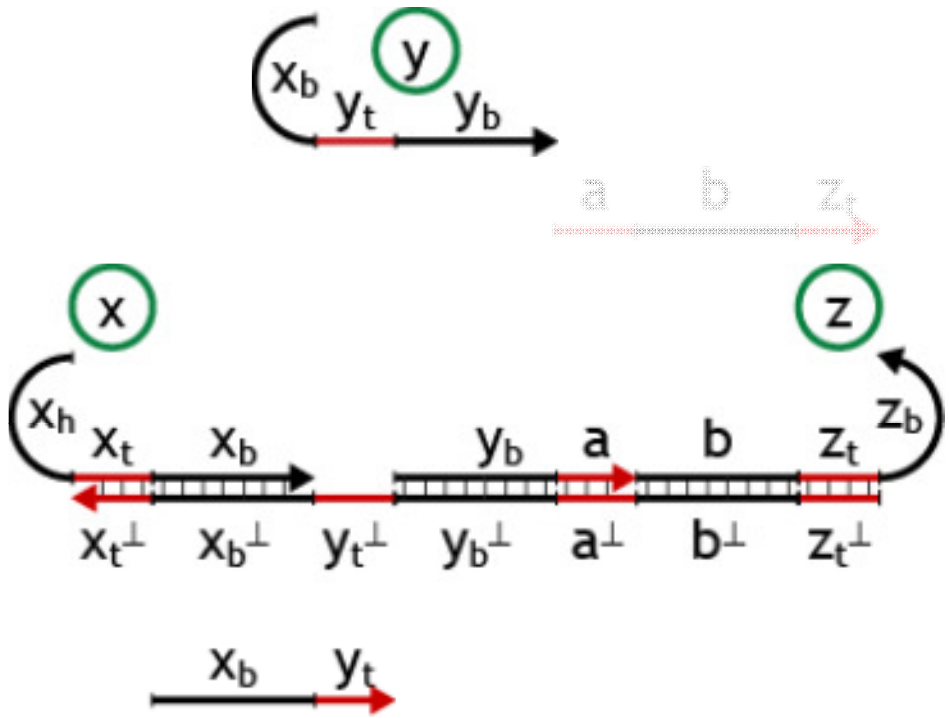
# Join Gate


y
$x_b$
$y_t$
$y_b$
a
b
$z_t$
x
z
$x_h$
$x_t$
$x_b$
$x_b$
$y_t$
$y_b$
a
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$

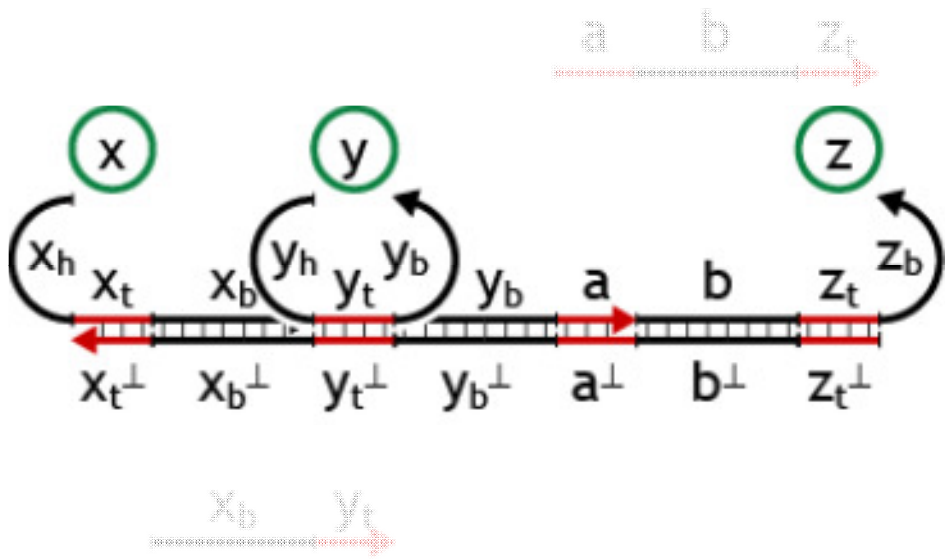
# Join Gate


y
$x_b$
$y_t$
$y_b$
a
b
$z_t$
x
z
$x_h$
$x_t$
$x_b$
$x_b$
$y_t$
$y_b$
a
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$

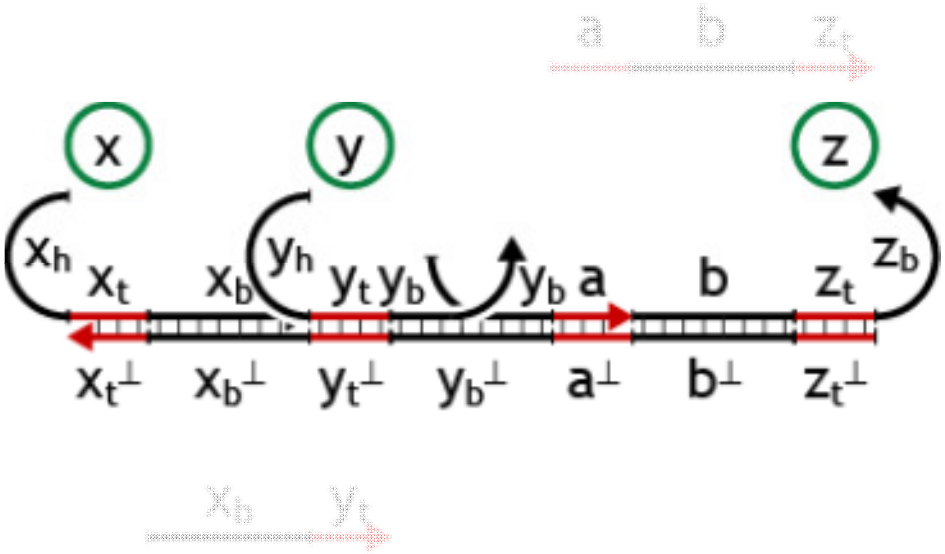
# Join Gate


$x_b$
y
$y_t$
$y_b$
a
b
$z_t$
x
z
$x_h$
$x_t$
$x_b$
$y_b$
a
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$

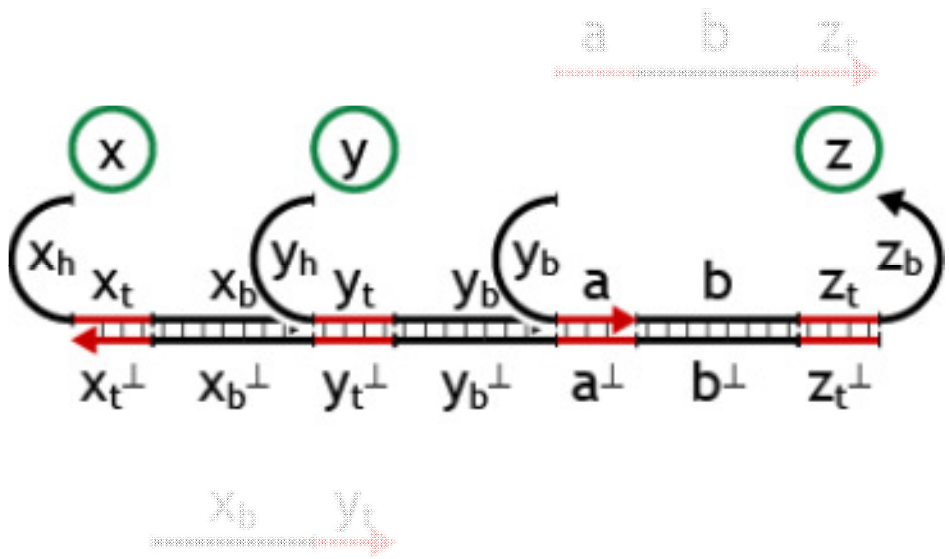
# Join Gate


a
b
$z_t$
x
y
z
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
$y_b$
a
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$

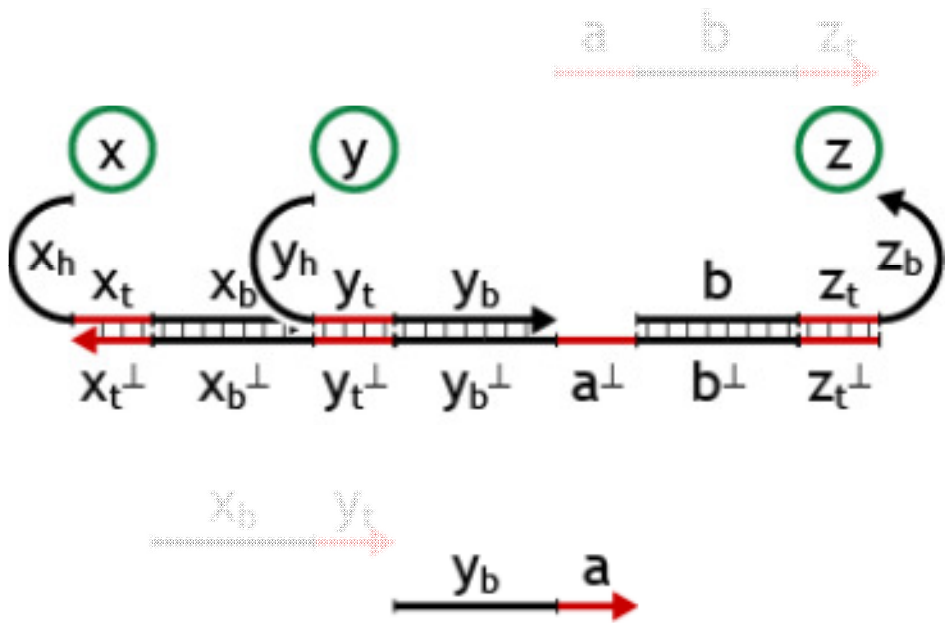
# Join Gate


a
b
$z_t$
x
y
z
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
$y_b$
a
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$

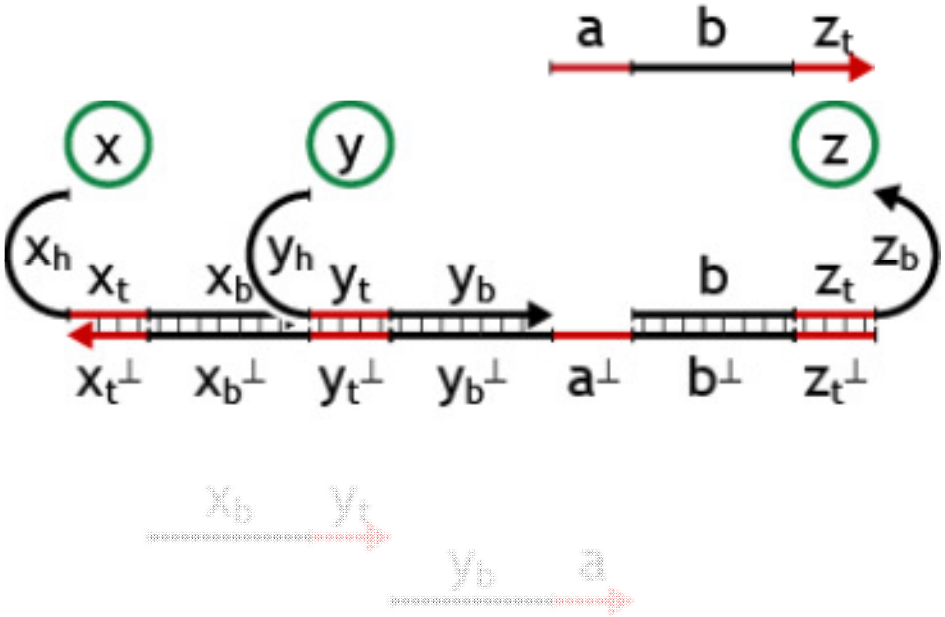
# Join Gate


a
b
z
x
y
z
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
$y_b$
a
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$

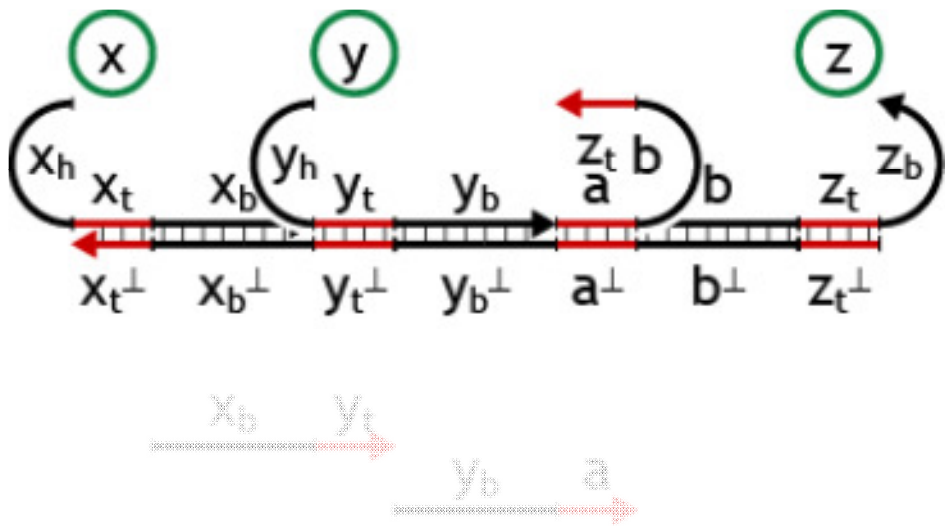
# Join Gate


a
b
$z_t$
x
y
z
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$
$y_b$
a

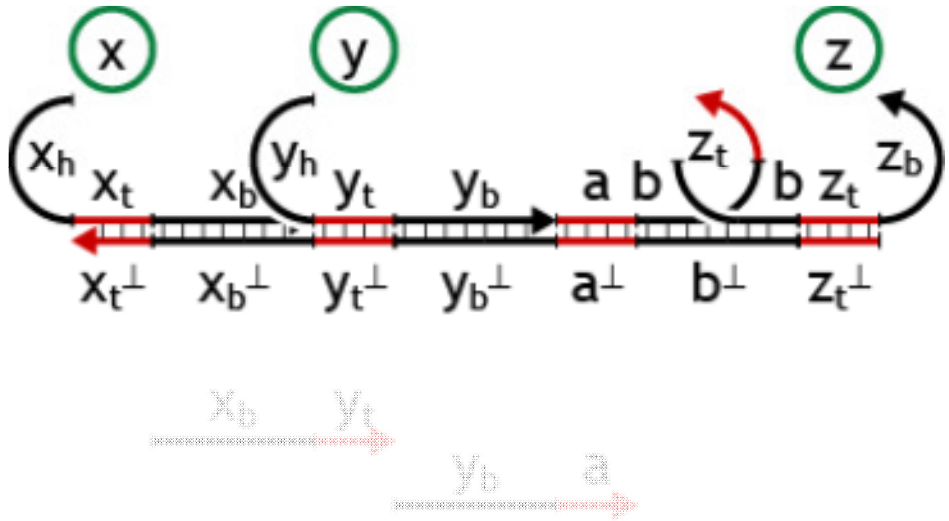
# Join Gate


a
b
$z_t$
x
y
z
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$
$y_b$
a

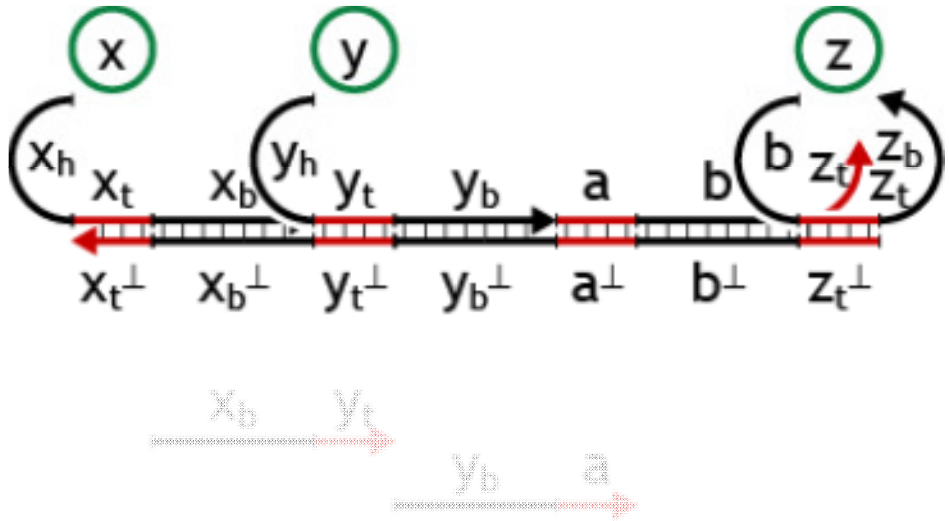
# Join Gate


x
y
z
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
$z_t$
a
b
b
$z_t$
$z_b$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$
$y_b$
a

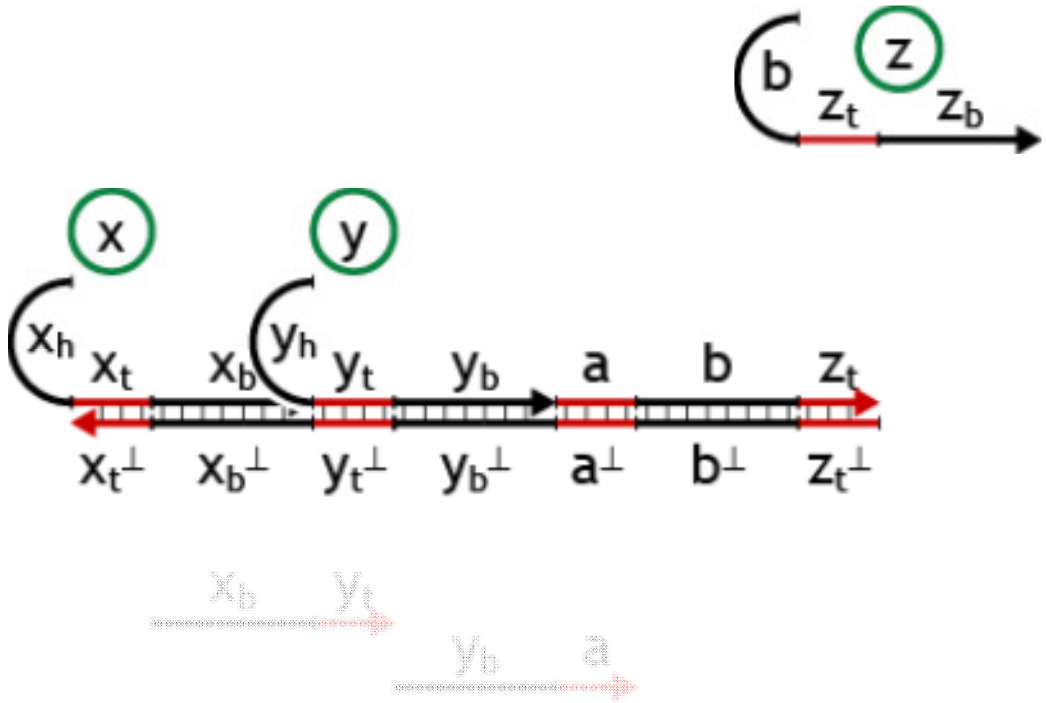
# Join Gate


x
y
z
$x_h$ $x_t$ $x_b$ $y_h$ $y_t$ $y_b$ a b $z_t$ b $z_t$ $z_b$
$x_t^\perp$ $x_b^\perp$ $y_t^\perp$ $y_b^\perp$ $a^\perp$ $b^\perp$ $z_t^\perp$
$x_b$ $y_t$
$y_b$ a

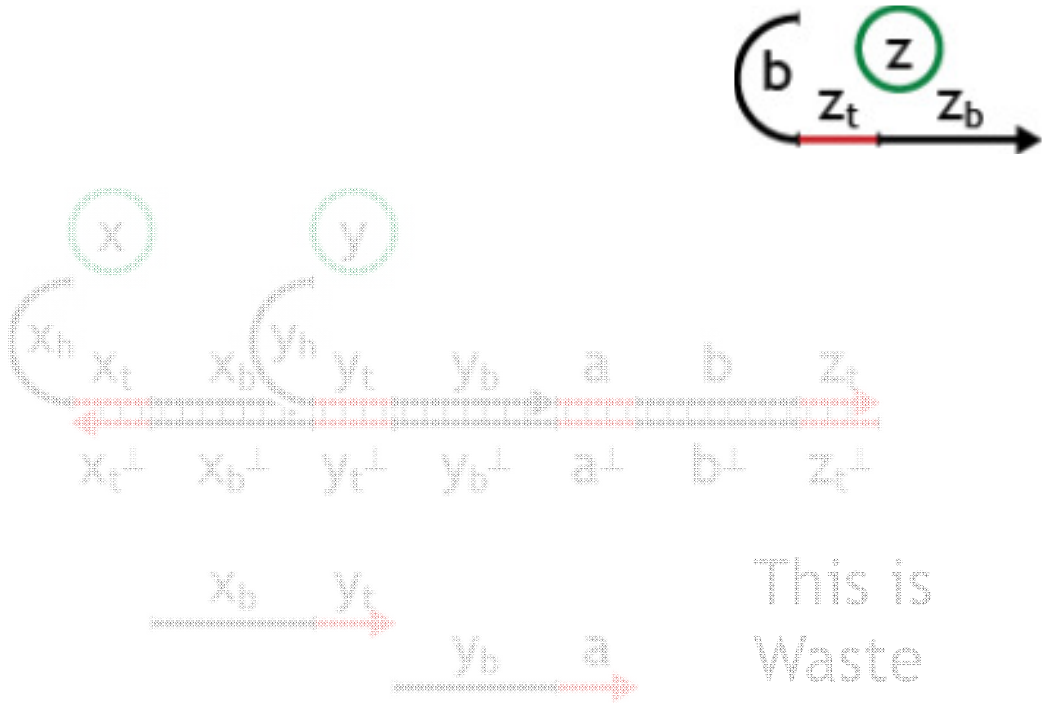
# Join Gate


x
y
z
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
a
b
b
$z_t$
$z_b$
$z_t$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$
$y_b$
a

# Join Gate


b
z
$z_t$
$z_b$
x
y
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
a
b
$z_t$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$
$y_b$
a

# Join Gate


b
z
$z_t$
$z_b$
x
y
$x_h$
$x_t$
$x_b$
$y_h$
$y_t$
$y_b$
a
b
$z_t$
$x_t^\perp$
$x_b^\perp$
$y_t^\perp$
$y_b^\perp$
$a^\perp$
$b^\perp$
$z_t^\perp$
$x_b$
$y_t$
$y_b$
a
This is
Waste

# Gate Design Verification

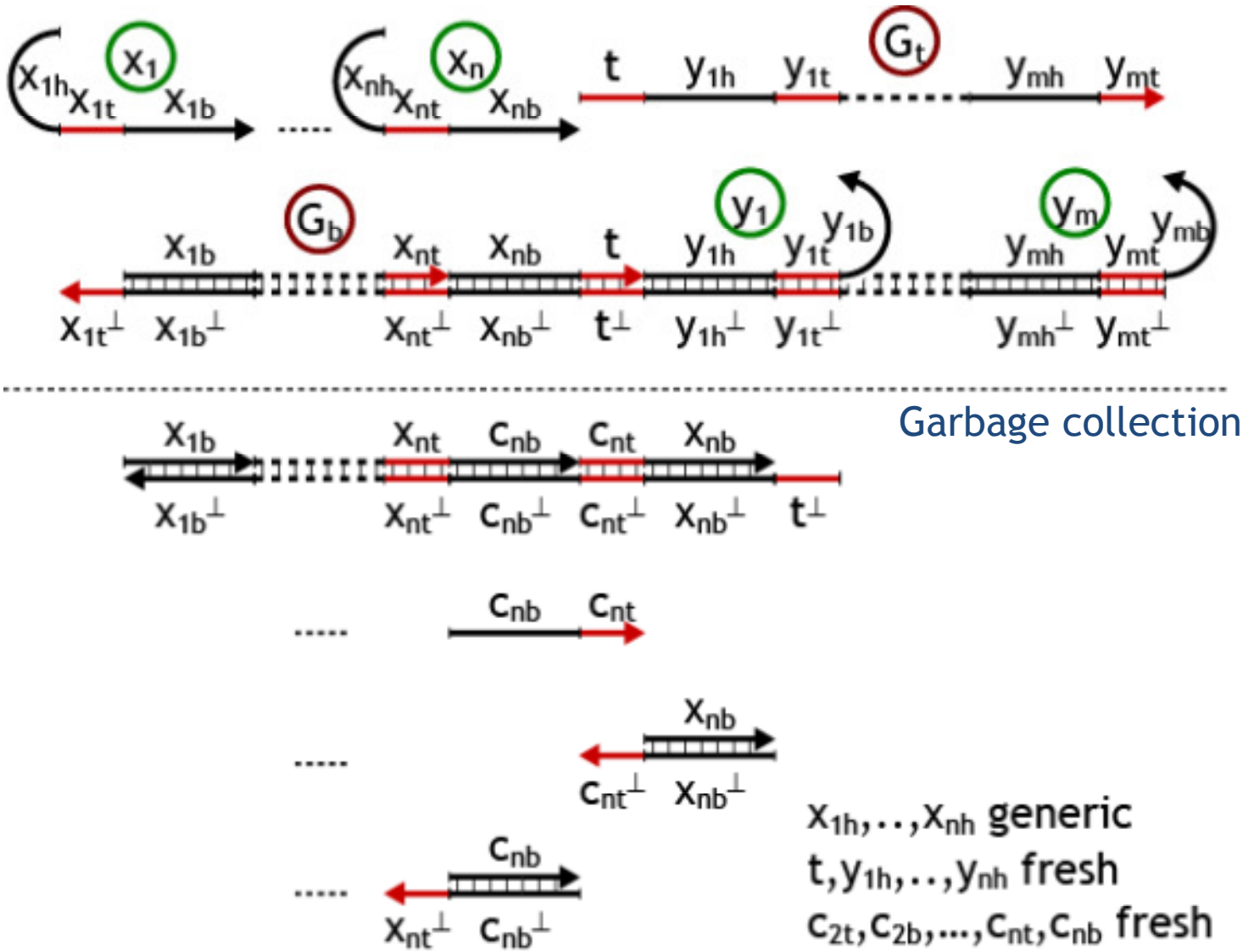
- Active garbage
  - The active join residuals slow down the performance of following joins.
  - → Add a garbage collector to remove the active residuals.

- Interference between gates
  - The join garbage collector interferes with the fork gate.
  - → Modify the fork gate to remove the interference.

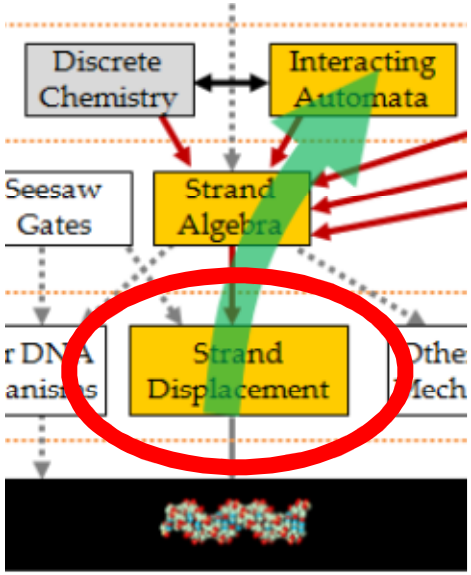
- What else could go wrong?
  - Endless possibilities.
  - → Prove that the fork/join gate structures correctly implement fork/join in all larger circuits.

# $[x_1,..,x_n].[y_1,..,y_m]$ General Join/Fork Gate

$x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[y_1,..,y_m] \rightarrow y_1 \mid .. \mid y_m$



# Strand Displacement Intermediate Language




Matthew Lakin
Simon Youssef
Andrew Phillips

# Syntax

Interface focus · IET · FirstCite e-publishing

*J. R. Soc. Interface*
doi:10.1098/rsif.2009.0072.focus
*Published online*

## A programming language for composable DNA circuits

Andrew Phillips* and Luca Cardelli

*Microsoft Research, Cambridge CB3 0FB, UK*

**A.** Syntax of DNA molecules *D*

| | |
|---|---|
| Upper strand with sequence complementary to S | S<br>`<S>` |
| Molecule with segments $G_1,\ldots,G_K$ | $G_1$ $G_2$ ... $G_K$<br>`G1:G2:...:GK` |
| Parallel molecules $D_1,\ldots,D_K$ | $D_1$ $D_2$ ... $D_K$<br>`D1 \| D2 \| ... \| DK` |
| Molecules *D* with private domains $N_1,\ldots,N_K$ | $(N_1,\ldots,N_K)$ D<br>`new (N1,...,NK) D` |

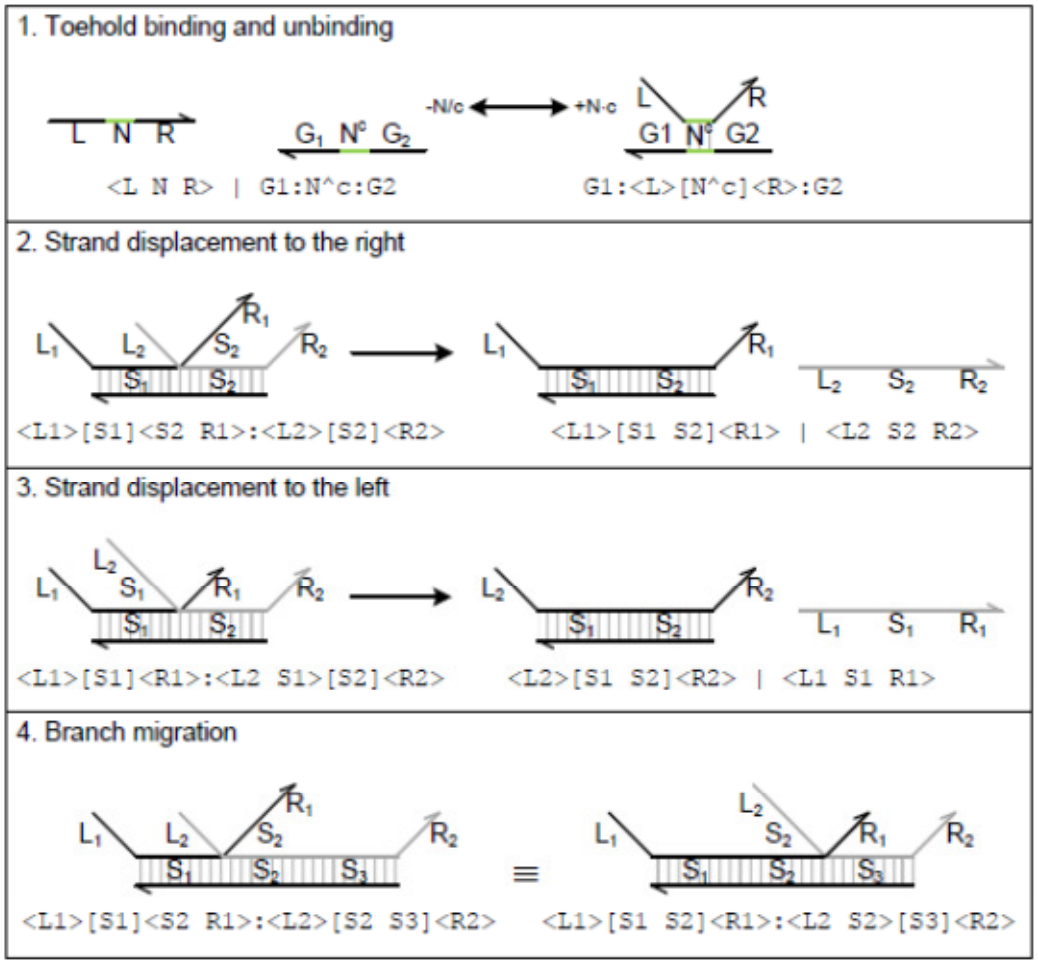
**B.** Syntax of DNA segments *G*

| | |
|---|---|
| Lower strand with toehold $N^c$ | $N^c$<br>`N^c` |
| Double strand with sequence S and overhangs L, R | L R S<br>`<L>[S]<R>` |

**C.** Syntax of DNA sequences *S,L,R*

| | |
|---|---|
| Sequence of domains $O_1,\ldots,O_K$ | $O_1$ $O_2$ ... $O_K$<br>`O1 O2 ... OK` |

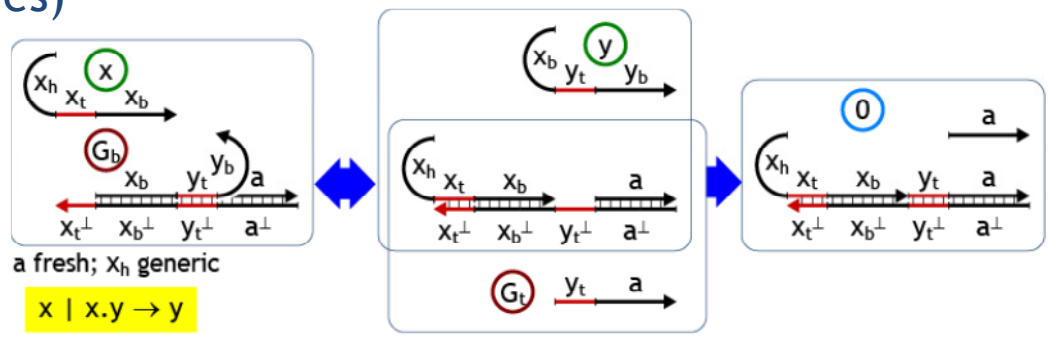
# Dynamics

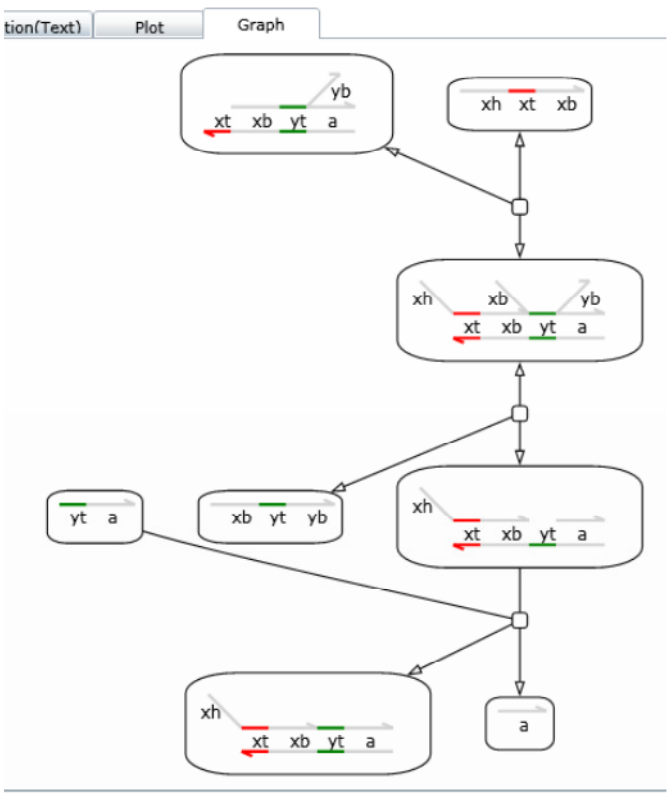

1. Toehold binding and unbinding
L N R
G1 N^c G2
-N/c
+N·c
L R
G1 N^c G2
<L N R> | G1:N^c:G2
G1:<L>[N^c]<R>:G2
2. Strand displacement to the right
L1 L2 S1 S2 R1 R2
<L1>[S1]<S2 R1>:<L2>[S2]<R2>
<L1>[S1 S2]<R1> | <L2 S2 R2>
3. Strand displacement to the left
<L1>[S1]<R1>:<L2 S1>[S2]<R2>
<L2>[S1 S2]<R2> | <L1 S1 R1>
4. Branch migration
L1 L2 S1 S2 S3 R1 R2
<L1>[S1]<S2 R1>:<L2>[S2 S3]<R2>
<L1>[S1 S2]<R1>:<L2 S2>[S3]<R2>

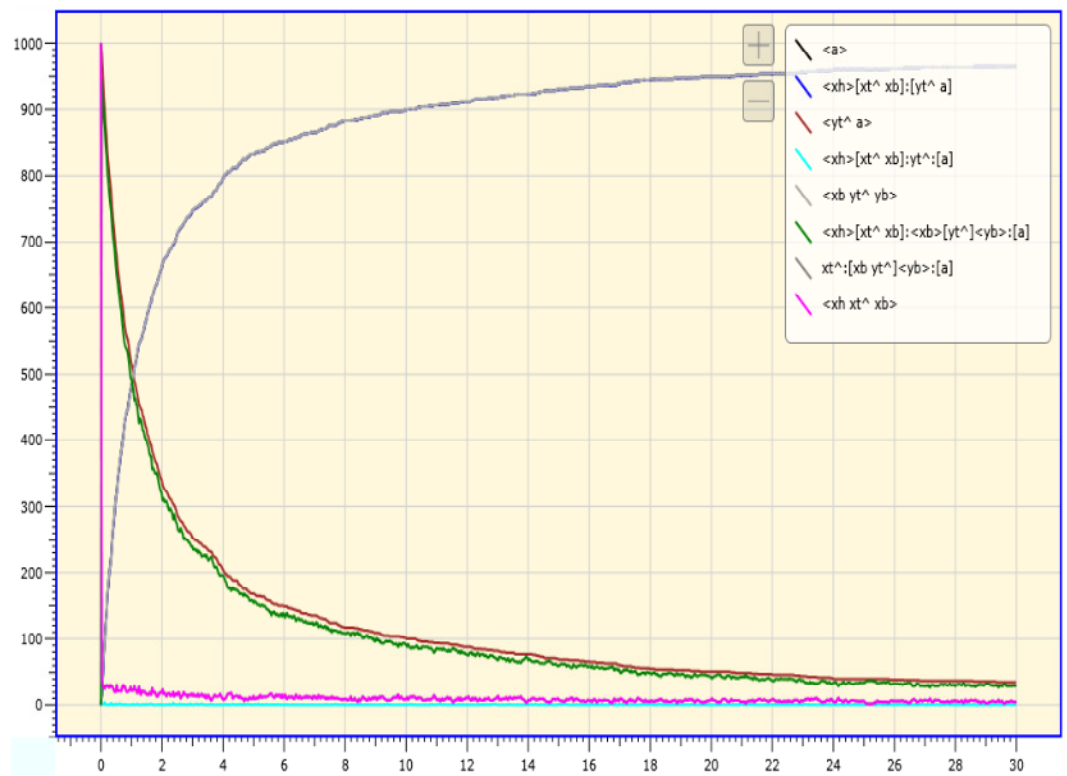
# Strand Displacement Analysis Tool

## 1 Transducer gate x.y (3 initial species)

```
directive sample 30.0 1000
new xt@1.0,1.0
new yt@1.0,1.0
( 1000 * <xh xt^ xb>
| 1000 * xt^:[xb yt^]<yb>:[a]
| 1000 * <yt^ a>
)
```



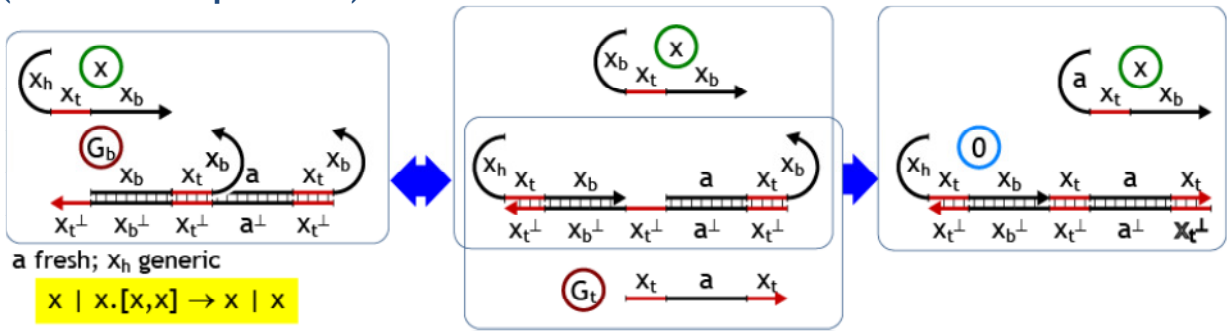



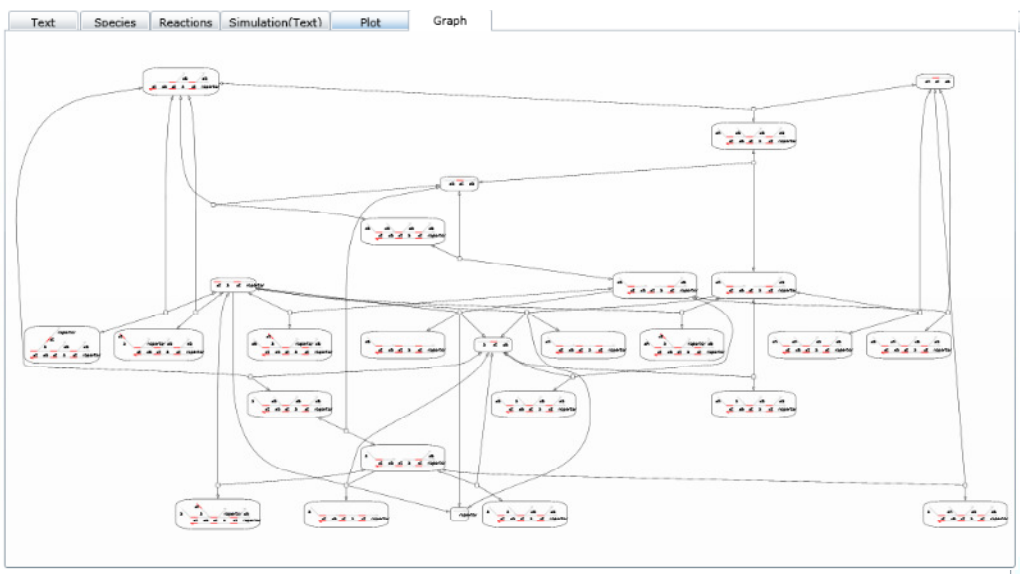
# Strand Displacement Analysis Tool

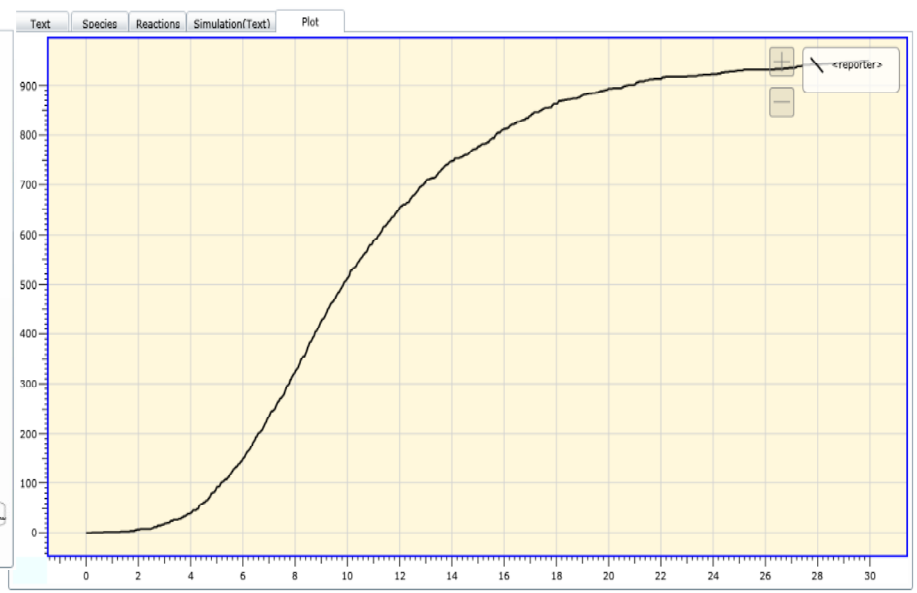
## Fork Chain Reaction x.[x,x] (3 initial species)

```
directive sample 30.0 1000
directive plot "<reporter>"
 new xt@ 1.0 , 1.0
( 1 * <xh xt^ xb>
| 1000 * xt^:[xb xt^]<xb>:[a xt^]<xb>:[reporter]
| 1000 * <xt^ a xt^ reporter>
)
```




## 26 Species, 20 Reactions



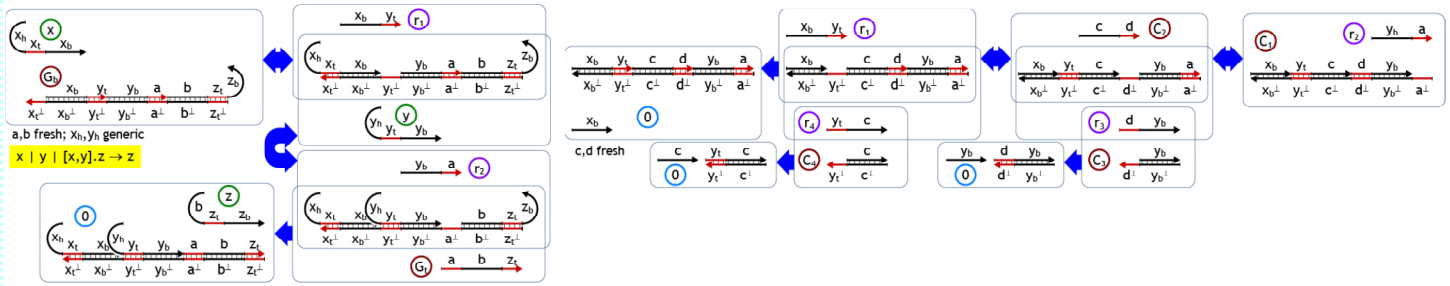



# Strand Displacement Analysis Tool

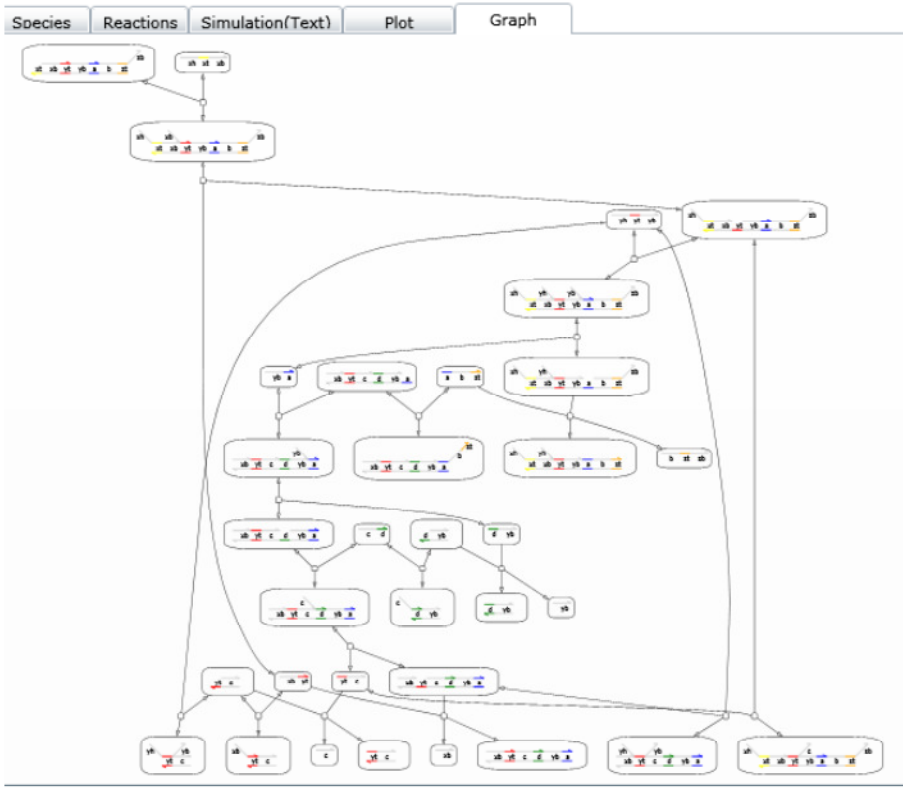
## 1 Join gate with garbage collection [x,y].z (8 initial species)

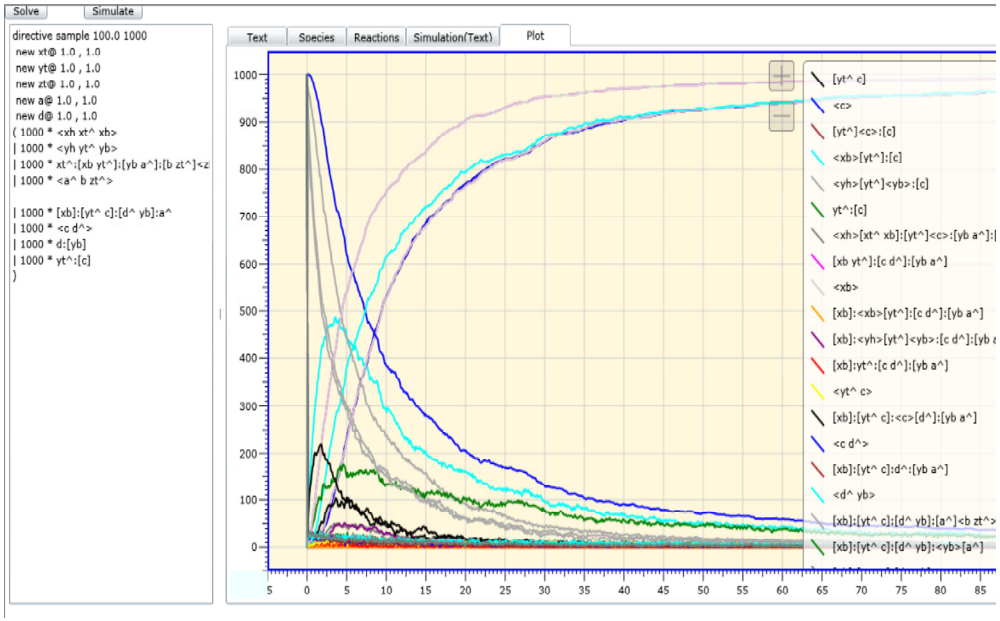
```
directive sample 1000.0 1000
 new xt@ 1.0 , 1.0
 new yt@ 1.0 , 1.0
 new zt@ 1.0 , 1.0
 new a@ 1.0 , 1.0
 new d@ 1.0 , 1.0
 ( 1000 * <xh xt^ xb>
 | 1000 * <yh yt^ yb>
 | 1000 * xt^:[xb yt^]:[yb a^]:[b zt^]<zb>
 | 1000 * <a^ b zt^>

 | 1000 * [xb]:[yt^ c]:[d^ yb]:a^
 | 1000 * <c d^>
 | 1000 * d^:[yb]
 | 1000 * yt^:[c]
 )
```

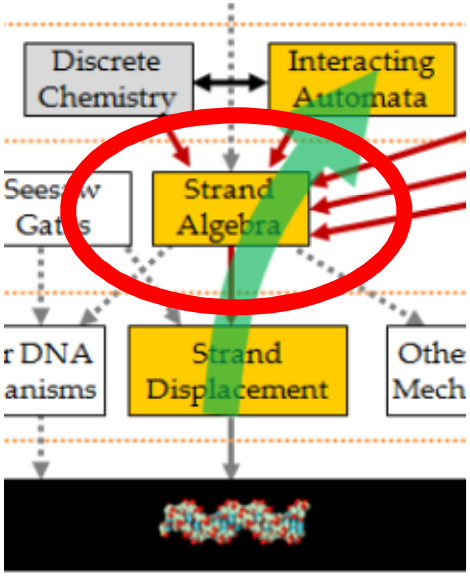



## 34 Species, 18 Reactions



# Strand Algebra



# Strand Algebra

n x m gates

$$P \ ::= \ x \ : \ [x_1,..,x_n].[y_1,..,y_m] \ : \ 0 \ : \ P|P \ : \ P^* \qquad n\geq 1,\ m\geq 0$$

| | |
|---|---|
| $x$ | is a *signal* |
| $[x_1,..,x_n].[y_1,..,y_m]$ | is a *gate* |
| $0$ | is an *inert solution* |
| $P\|P$ | is *parallel composition* of signals and gates |
| $P^*$ | is a *population* (multiset) of signals and gates |

Reaction Rule

$$x_1 \mid .. \mid x_n \mid [x_1,..,x_n].[y_1,..,y_m] \ \rightarrow \ y_1 \mid .. \mid y_m$$

Auxiliary rules (axioms of diluted well-mixed solutions)

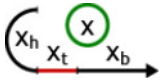
$$P \rightarrow P' \quad \Rightarrow \quad P \mid P'' \rightarrow P' \mid P'' \qquad \text{Dilution}$$

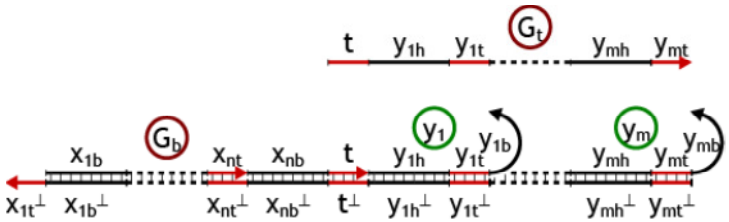
$$P \equiv P_1,\ P_1 \rightarrow P_2,\ P_2 \equiv P' \quad \Rightarrow \quad P \rightarrow P' \qquad \text{Well Mixing}$$

Where $\equiv$ is a congruence relation (syntactical 'chemical mixing')
with $P^* \equiv P \mid P^*$ for unbounded populations.

# Compiling Strand Algebra to DNA

$P ::= x \vdots [x_1,..,x_n].[y_1,..,y_m] \vdots 0 \vdots P|P \vdots P^* \qquad n\geq 1,\ m\geq 0$

- compile(x) = 

- compile($[x_1,..,x_n].[y_1,..,y_m]$) = 

- compile(0) = empty solution
- compile(P | P') = mix(compile(P), compile(P'))
- compile(P*) = population(compile(P))

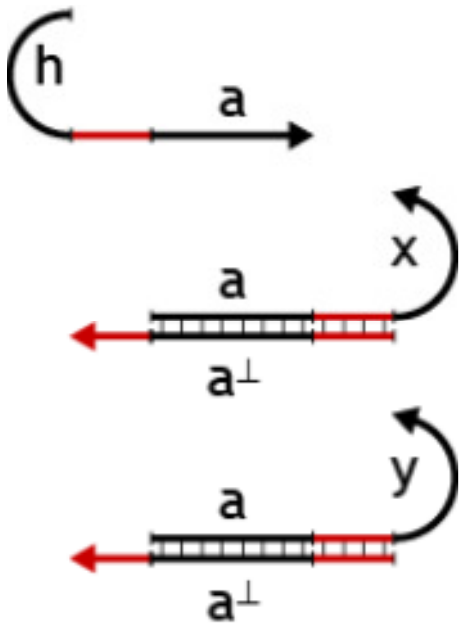
# More in the DNA15 Paper

- Stochastic strand algebra
  - Matches the stochastic semantics of interacting automata
  - Uses a technique for implementing constant buffered populations, to replace P* with finite populations

- Nested strand algebra
  - An higher-level language (with nested expressions)
  - A compilation algorithm into the basic strand algebra

# Other Gates
# Other Algebras

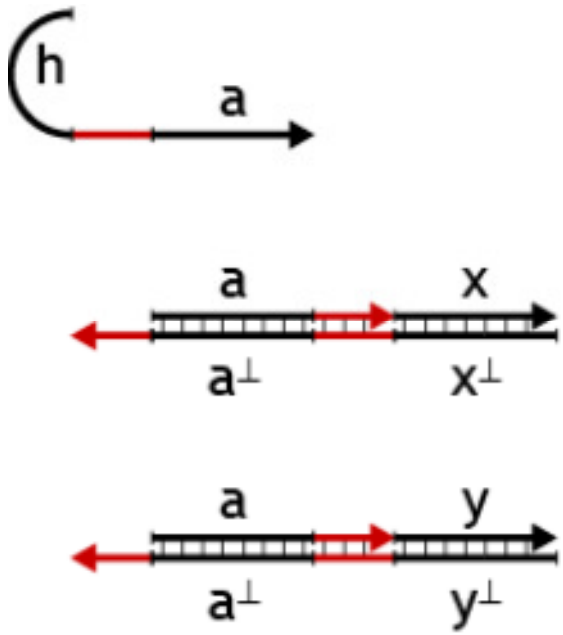
# Ouput Choice Gate




$!x \oplus !y$

Either provide signal x,
or provide signal y,
but not both.

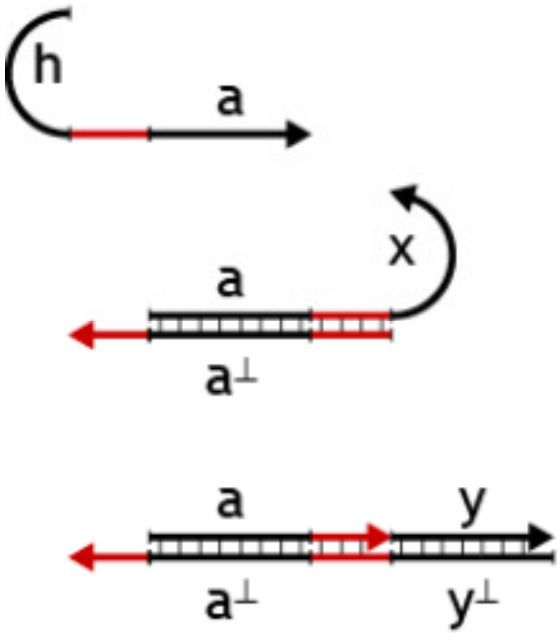
# Input Choice Gate




$?x \oplus ?y$

Either accept signal x,
or accept signal y,
but not both.

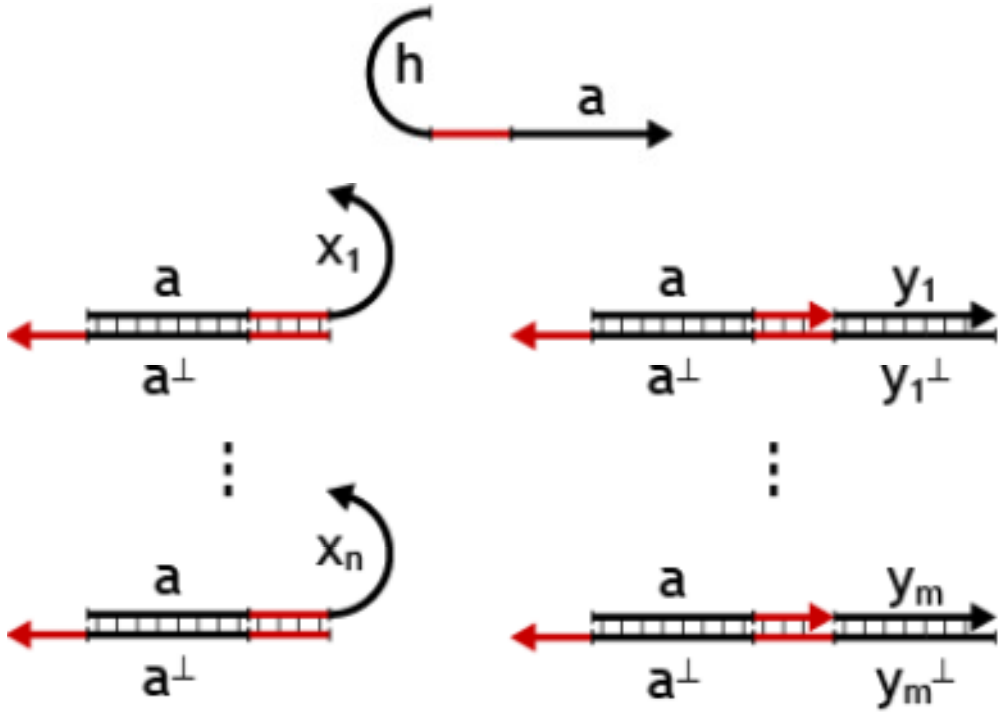
# Mixed Choice Gate




$!x \oplus ?y$

Either provide signal x,
or accept signal y,
but not both.

# General Choice Gate

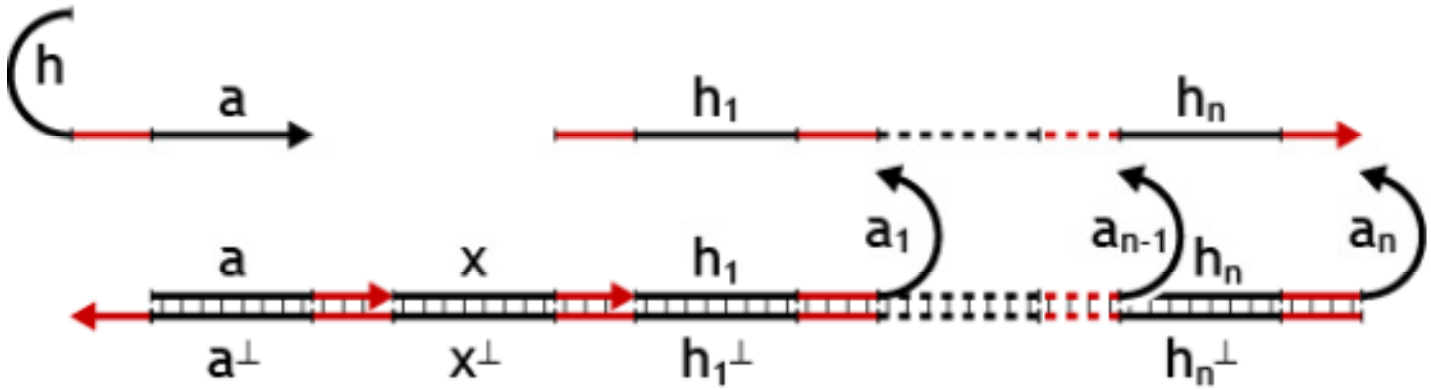



$$!x_1 \oplus \ldots \oplus !x_n \oplus ?y_1 \oplus \ldots \oplus ?y_m$$

Either provide one of $x_1, \ldots, x_n$,
or accept one of $y_1, \ldots, y_m$.

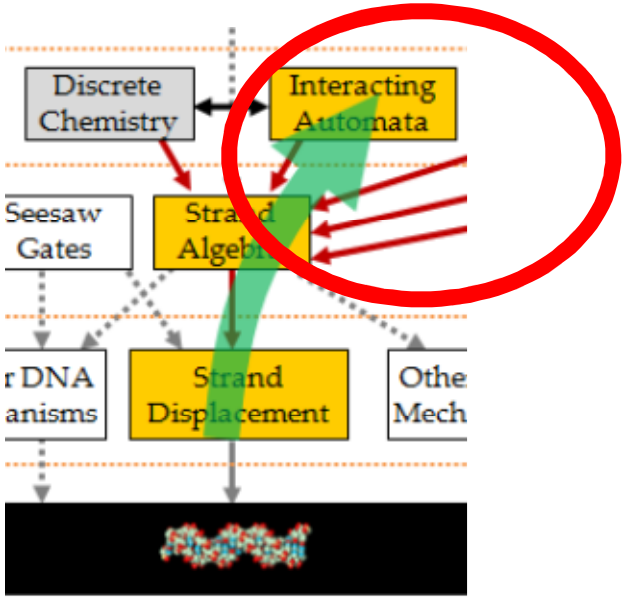
# Moreover…

- Any input choice can trigger the release of number of other signals:




$$?x.\ (a_1 \mid \ldots \mid a_n)$$

# Abstract Machines



# Chemical Reaction Networks

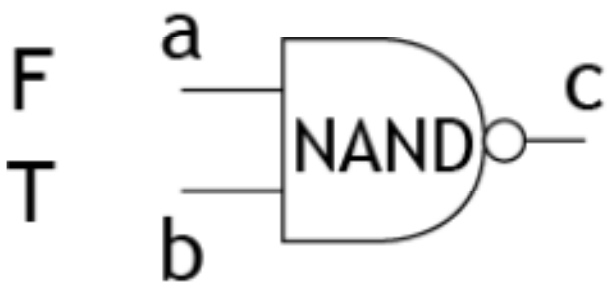
Implementing an arbitrary finite chemical system in DNA
with asymptotically correct kinetics
Soloveichick & al. DNA 15

Species become signals
Reactions become gates

$$A + B \rightarrow C + D \quad \Rightarrow \quad [A,B].[C,D]$$

# Boolean Networks

Boolean Networks to Strand Algebra



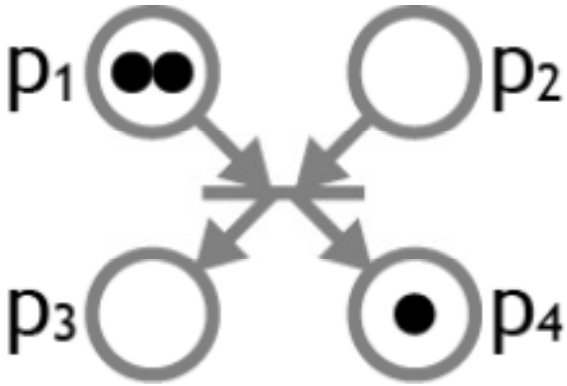

$$
\begin{array}{l}
([a_F,b_F].c_T)^* \mid \\
([a_F,b_T].c_T)^* \mid \\
([a_T,b_F].c_T)^* \mid \\
([a_T,b_T].c_F)^* \mid \\
a_F \mid b_T
\end{array}
$$

This encoding is *compositional*, and can encode *any* Boolean network:
- multi-stage networks can be assembled (combinatorial logic)
- network loops are allowed (sequential logic)

# Petri Nets

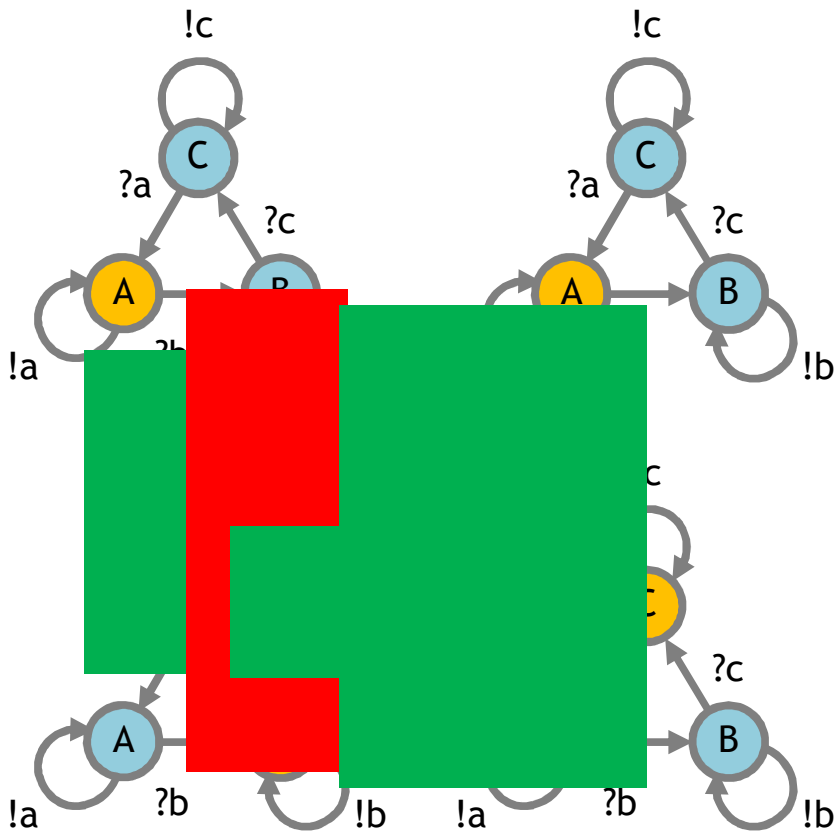
## Petri Nets to Strand Algebra

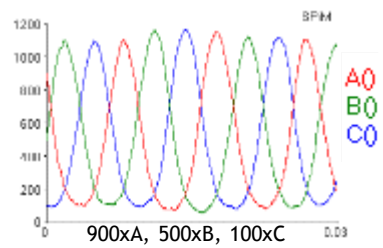
Transitions as Gates
Place markings as Signals




$$([p_1,p_2].[p_3,p_4])^* \mid p_1 \mid p_1 \mid p_4$$

# Interacting Automata

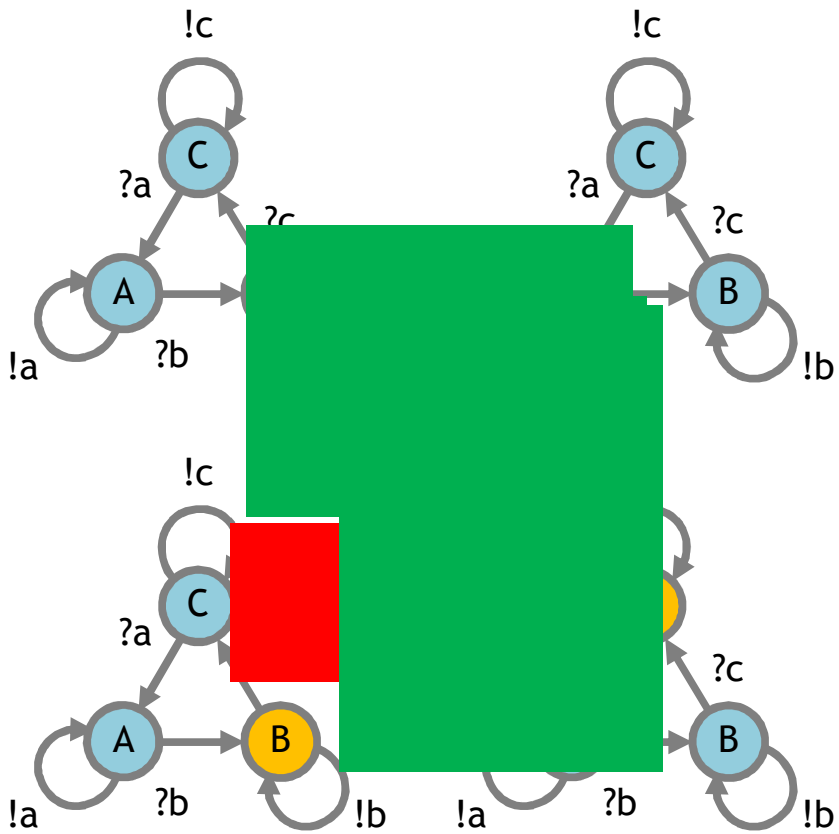


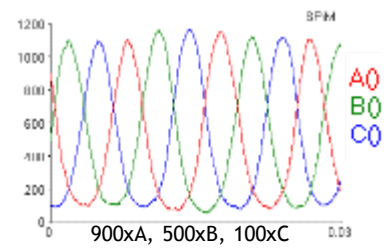




**([A,B].[B,B])* |**
**([B,C].[C,C])* |**
**([C,A].[A,A])* |**
**A | A | B | C**

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

# Interacting Automata

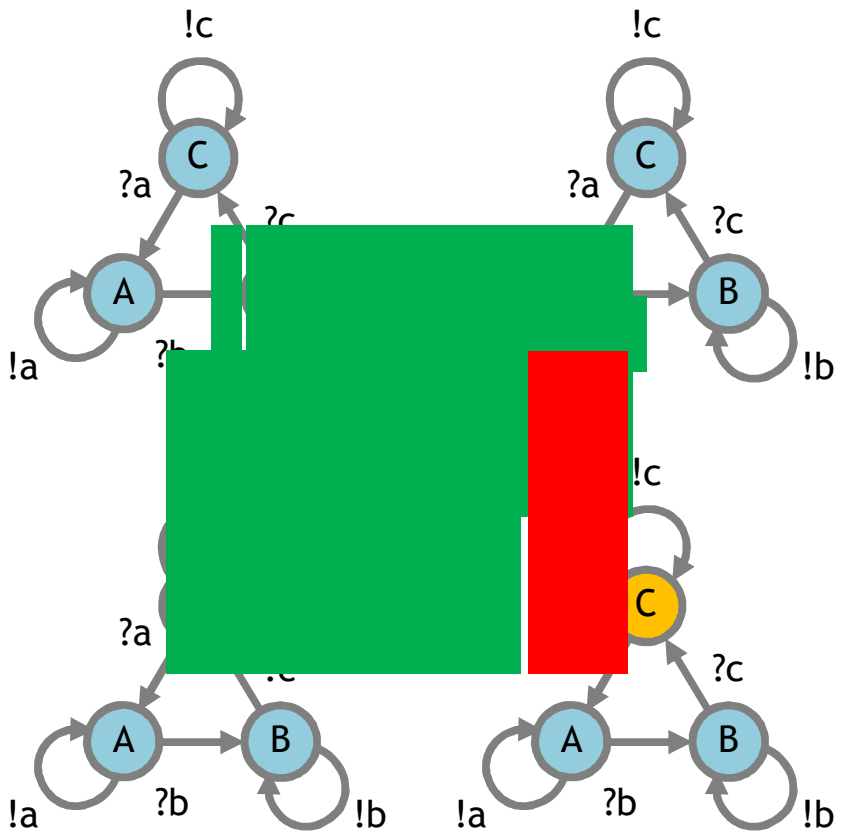


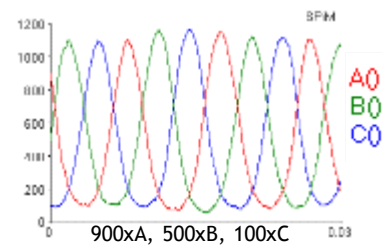




**([A,B].[B,B])* |**
**([B,C].[C,C])* |**
**([C,A].[A,A])* |**
**A | B | B | C**

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

# Interacting Automata

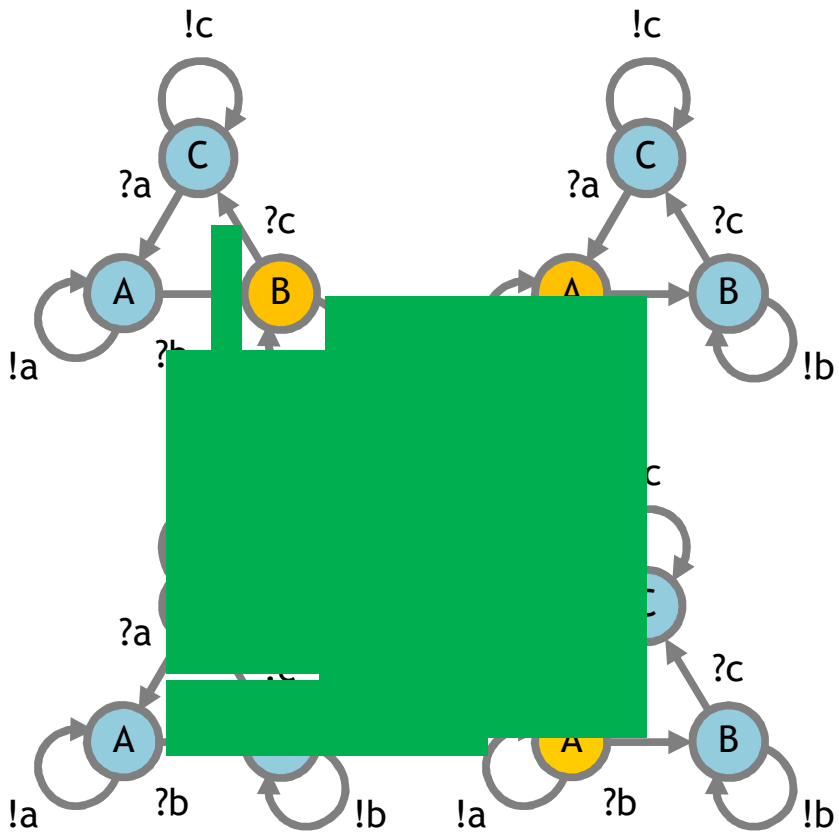


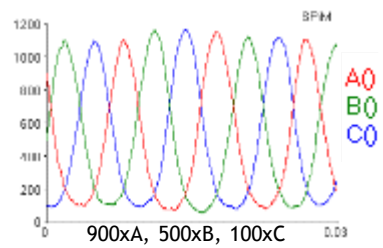




**([A,B].[B,B])* |**
**([B,C].[C,C])* |**
**([C,A].[A,A])* |**
**A | B | C | C**

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

# Interacting Automata

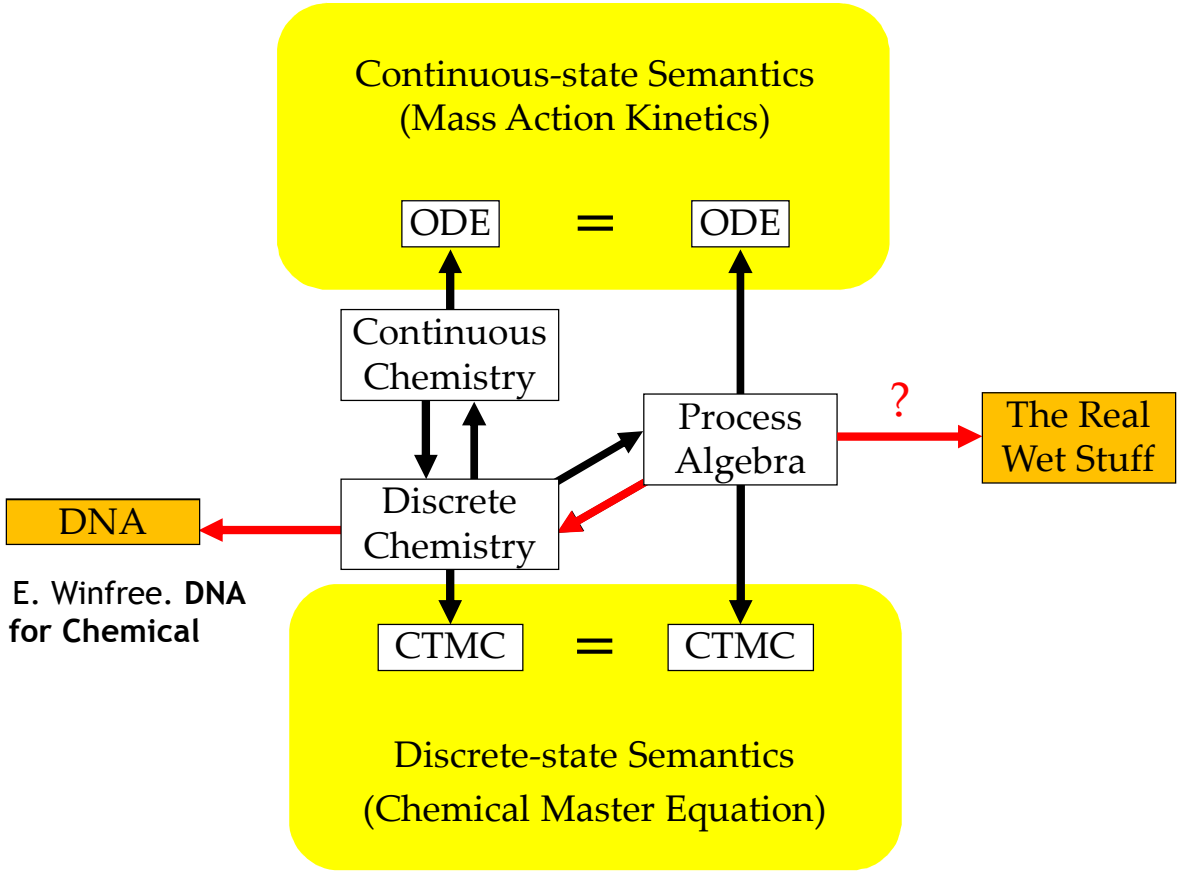



**([A,B].[B,B])* |**
**([B,C].[C,C])* |**
**([C,A].[A,A])* |**
**A | A | B | C**

This is a uniform population of identical automata,
but heterogeneous populations of interacting automata can be similarly handled.

# Molecules as Automata




D. Soloveichik, G. Seelig, E. Winfree. **DNA as a Universal Substrate for Chemical Kinetics.** Proc. DNA14.

L. Cardelli: "On Process Rate Semantics" (TCS)

L. Cardelli: "A Process Algebra Master Equation" (QEST'07)

# Conclusions

# Conclusion

- Nucleic Acids
  - Programmable matter
- DNA Strand Displacement
  - A computational mechanism at the molecular level
- DNA as a Compilation Target for Abstract Machines
  - Abstract Machines (Boolean Networks, Petri Nets, Interacting Automata)
  - Intermediate languages (Strand Algebra, Strand Displacement Language).
  - DNA sequence generation.
- Tools
  - Thermodynamic analysis.
  - Reaction graph generation.
  - Simulation.
  - Verification (not yet).

# Q?

http://lucacardelli.name