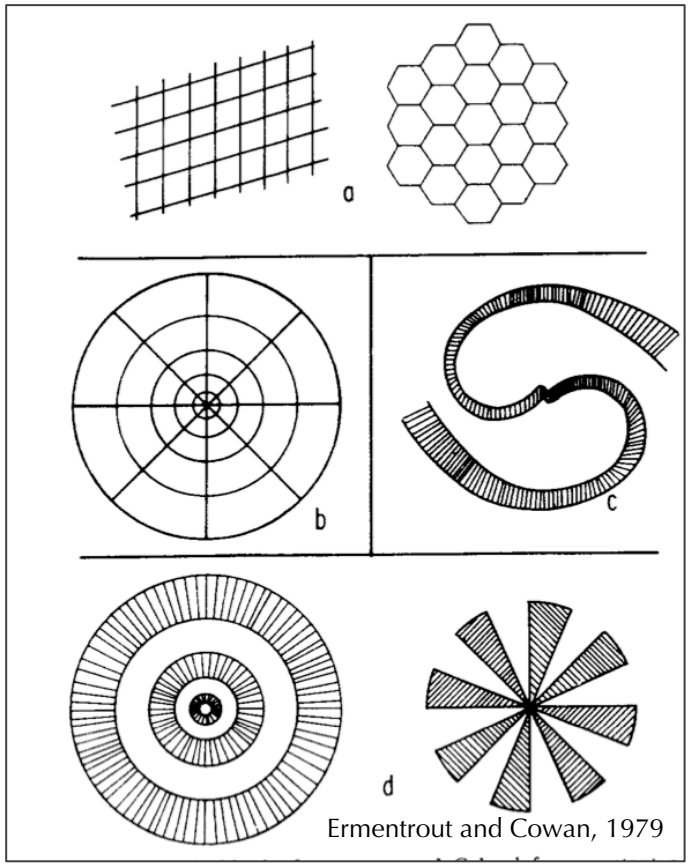
Three Mini Great Ideas

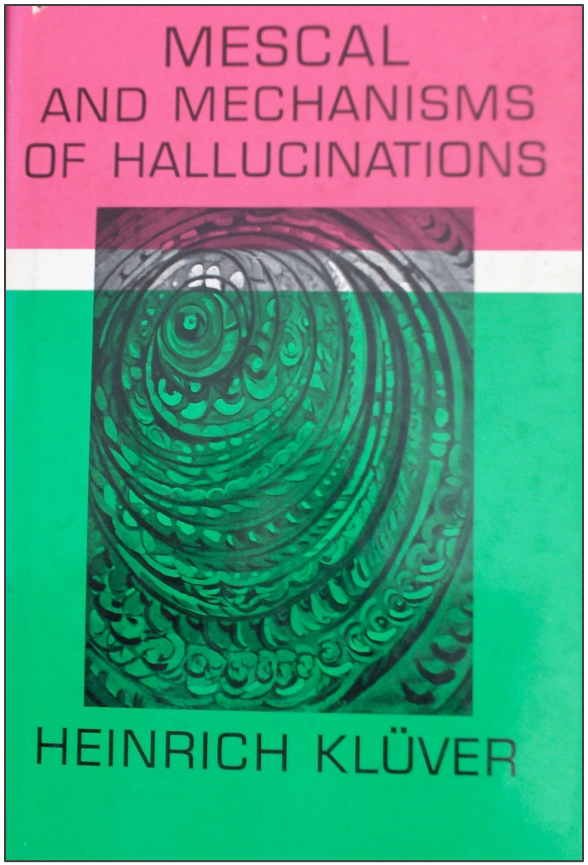
Tripping with Turing, Fragile Genomes, and DNA Computing

Phillip Compeau

IDEA 1: TRIPPING WITH TURING

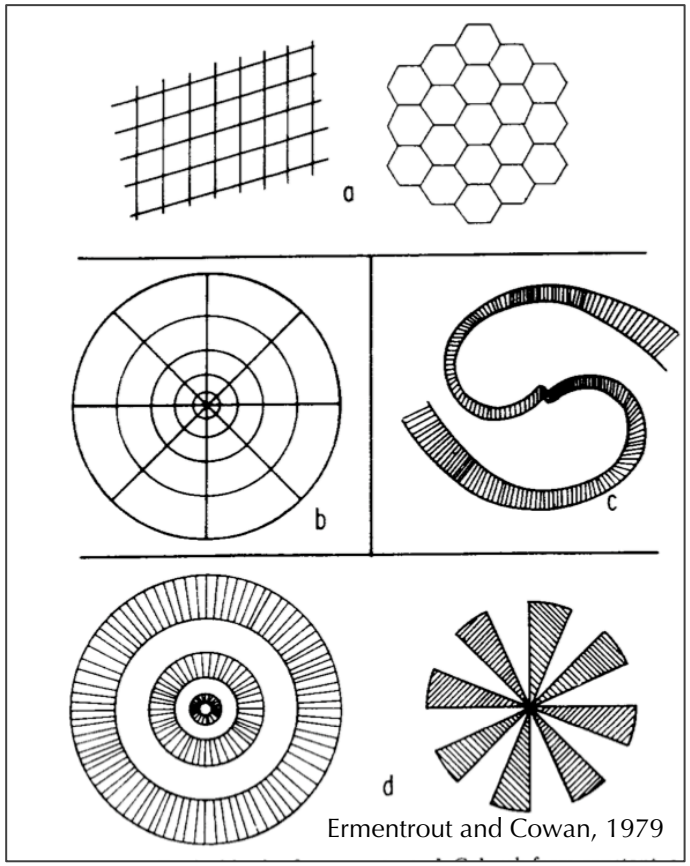
Klüver and "Form Constants"

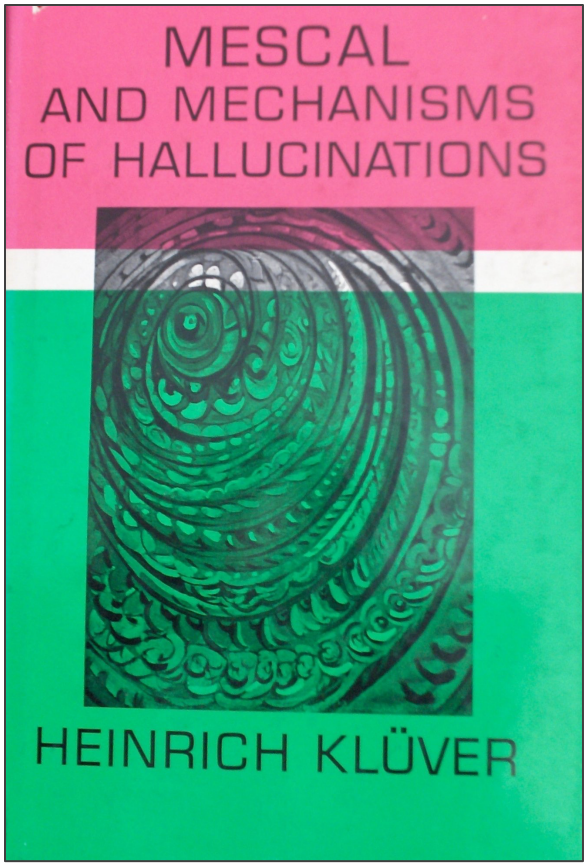




Form constant (Klüver, 1928): a commonly recurring shape in visual hallucinations.

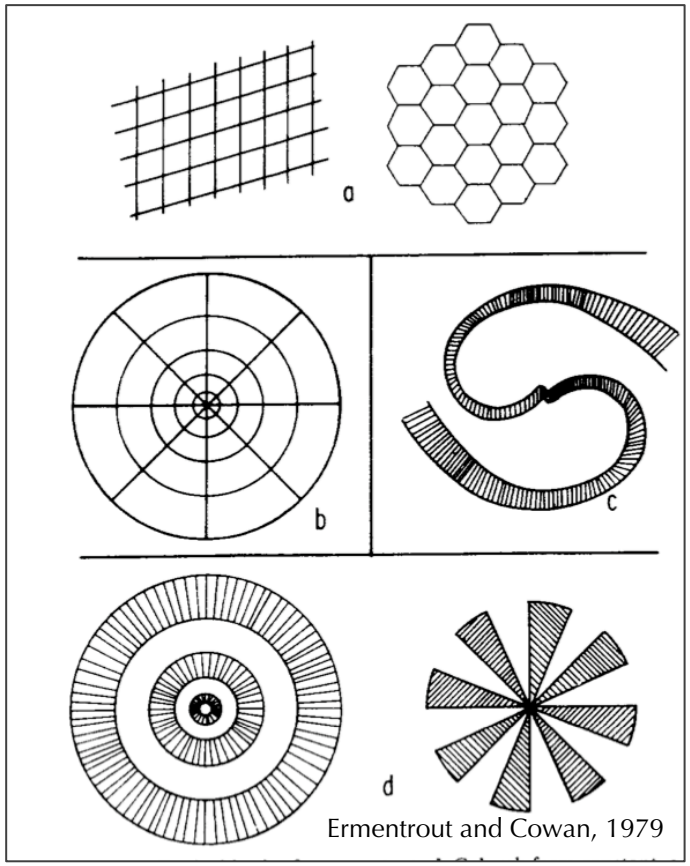
Klüver and "Form Constants"



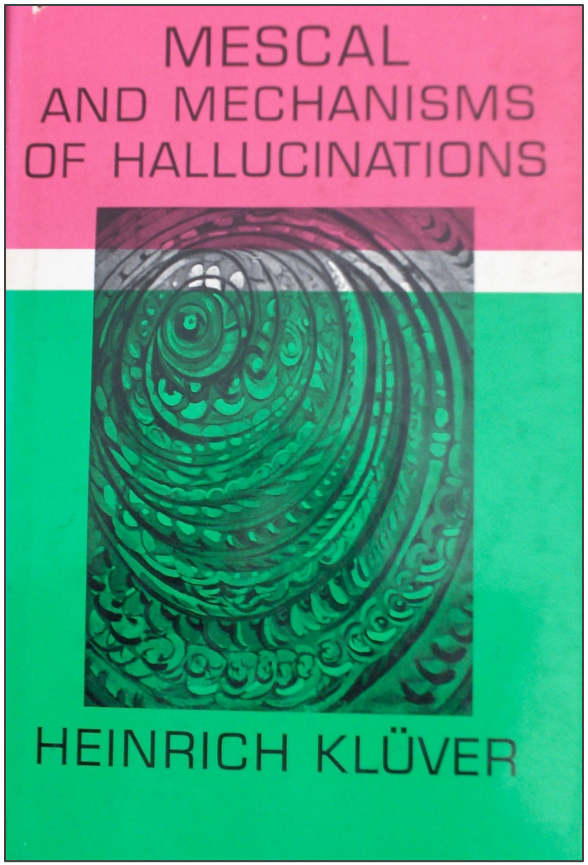


Why would we all see the same shapes, regardless of the cause of the hallucinations?

Klüver and "Form Constants"



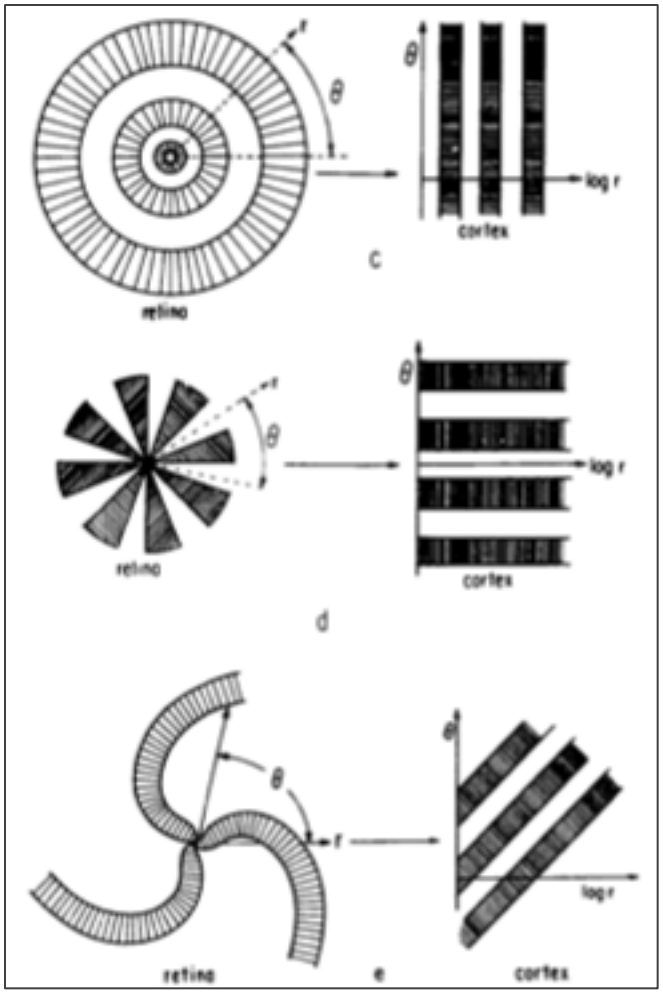
Ermentrout and Cowan, 1979



Key Point: Hallucinations happen in the blind and don't move in visual field, so they originate in brain.

The Brain Encodes Signals from Retina

Cowan 1978: determined details for transformation of retinal coordinates (polar) to cortex (rectangular).

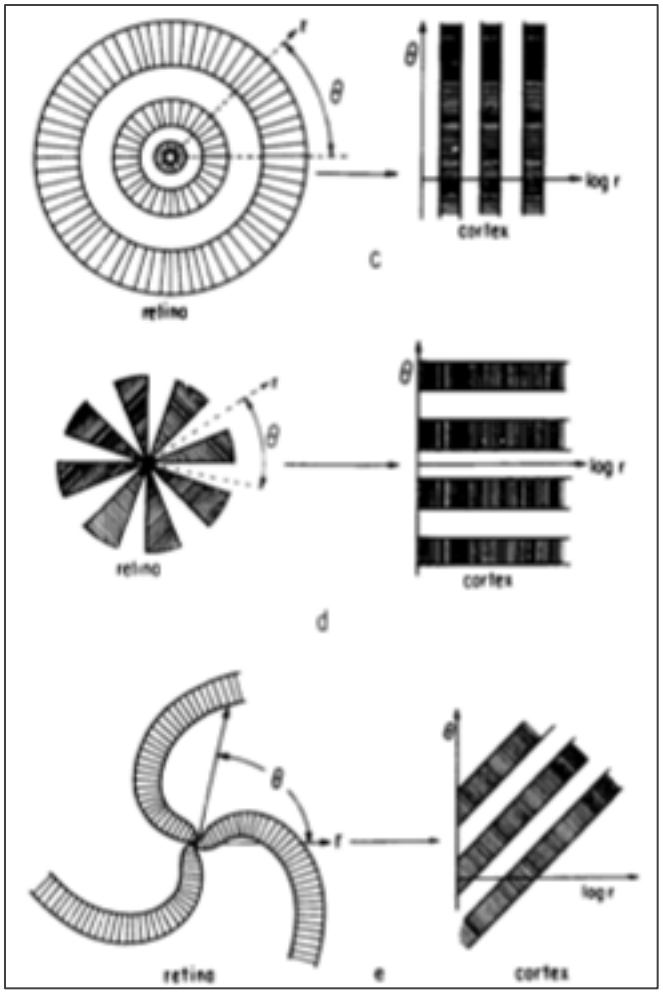


Ermentrout and Cowan, 1979

The Brain Encodes Signals from Retina

Cowan 1978: determined details for transformation of retinal coordinates (polar) to cortex (rectangular).

Key Point: All the form constants reduce to "stripes" in the visual cortex!



Ermentrout and Cowan, 1979

A Seemingly Separate Question: Why Do Animals Have Stripes (and Spots)?


https://i.ebayimg.com/images/g/gy4AAOxylpNTUTCl/s-l300.jpg


https://en.wikipedia.org/wiki/Mbu_pufferfish#/media/File:Giant_Puffer_fish_skin_pattern.JPG


https://lv.wikipedia.org/wiki/Čapmana_zebra#/media/File:Common_zebra_1.jpg


https://en.wikipedia.org/wiki/Leopard#/media/File:Nagarhole_Kabini_Karnataka_India,_Leopard_September_2013.jpg

Answer: Turing!

Reaction-diffusion: a model of a chemical reaction occurring concurrently with diffusion.

Turing patterns: patterns that arise as a result of certain reaction-diffusions.

Answer: Turing!

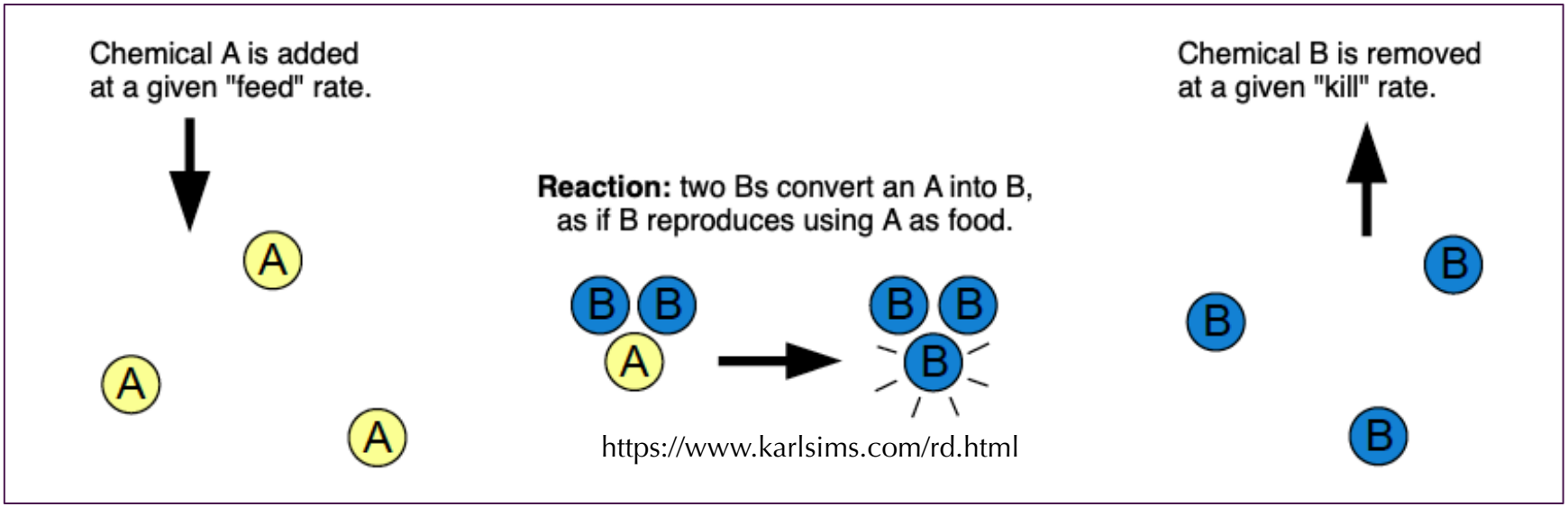
Reaction-diffusion: a model of a chemical reaction occurring concurrently with diffusion.

Well, the stripes are easy. But what about the horse part?

Turing patterns: patterns that arise as a result of certain reaction-diffusions.

Gray-Scott Model for Reaction-Diffusion

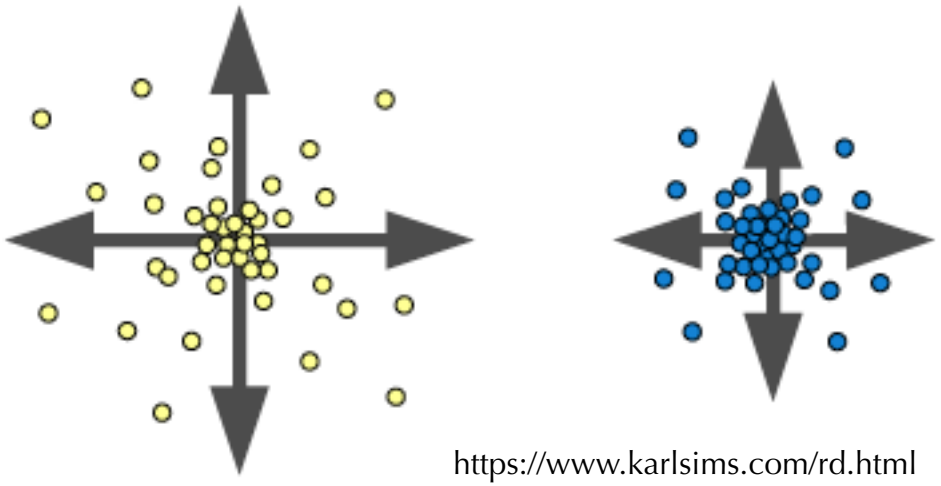
Gray-Scott Model: a reaction-diffusion model often used for generating Turing patterns.



Gray-Scott Model for Reaction-Diffusion

Gray-Scott Model: a reaction-diffusion model often used for generating Turing patterns.

Diffusion: both chemicals diffuse so uneven concentrations spread out across the grid, but A diffuses faster than B.

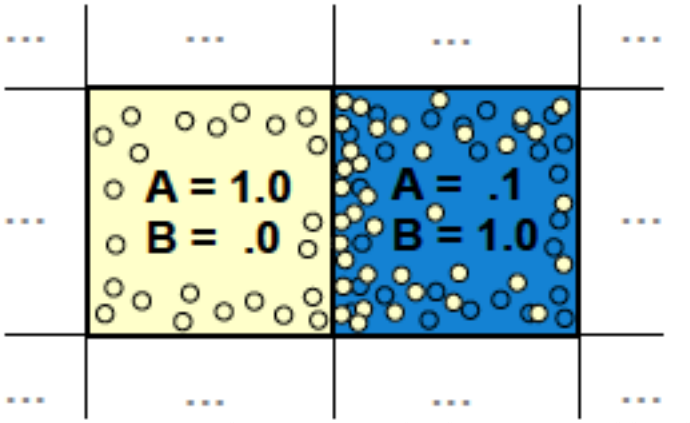


https://www.karlsims.com/rd.html

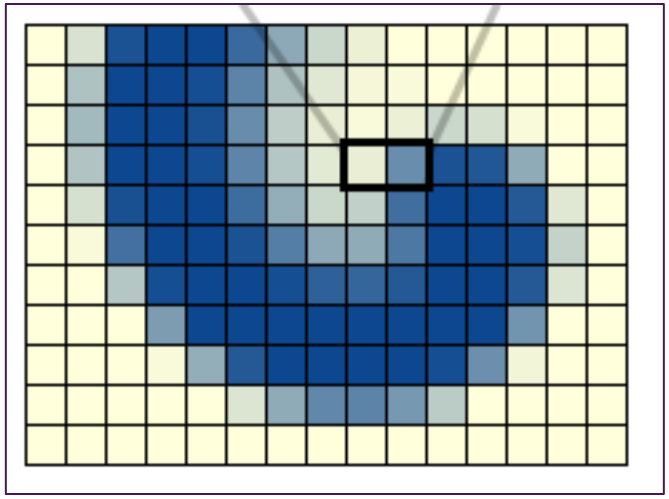
Gray-Scott Model for Reaction-Diffusion

Gray-Scott Model: a reaction-diffusion model often used for generating Turing patterns.

The system is approximated by using two numbers at each grid cell for the local concentrations of A and B. (The particles are not individually simulated.)

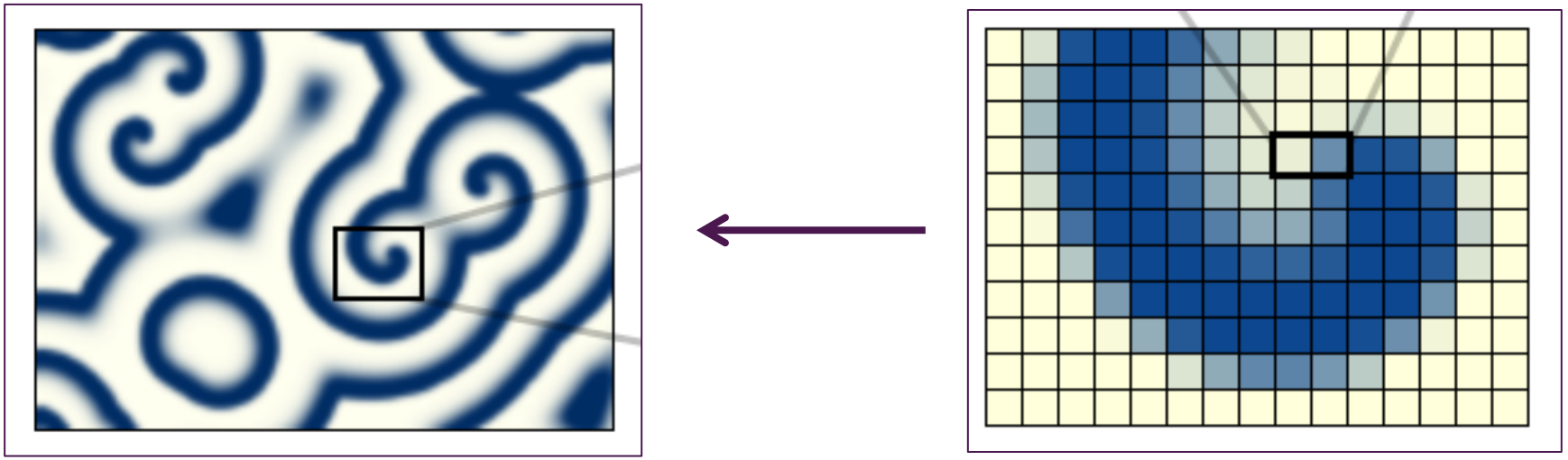


https://www.karlsims.com/rd.html



Gray-Scott Model for Reaction-Diffusion

Stable Turing patterns emerge when the reaction-diffusion is simulated across a field.



https://www.karlsims.com/rd.html

Exploring the Gray-Scott Model

Link: https://pmneila.github.io/jsexp/grayscott/

In this simulation, note:

- Green/red spectrum denotes concentration of A/B
- Default feed rate: produces stripes
- Lower feed rate (0.25): produces spots
- Low feed rate: everything dies out
- High feed rate: feed rate beats diffusion.
- Low death rate: mostly B. High death rate: mostly black

How Does This Relate to Hallucinations?

The visual cortex contains ″activator″ neurons that tend to be connected closer to each other and “inhibitor” neurons with fewer, sparser connections.

How Does This Relate to Hallucinations?

The visual cortex contains ″activator″ neurons that tend to be connected closer to each other and “inhibitor” neurons with fewer, sparser connections.

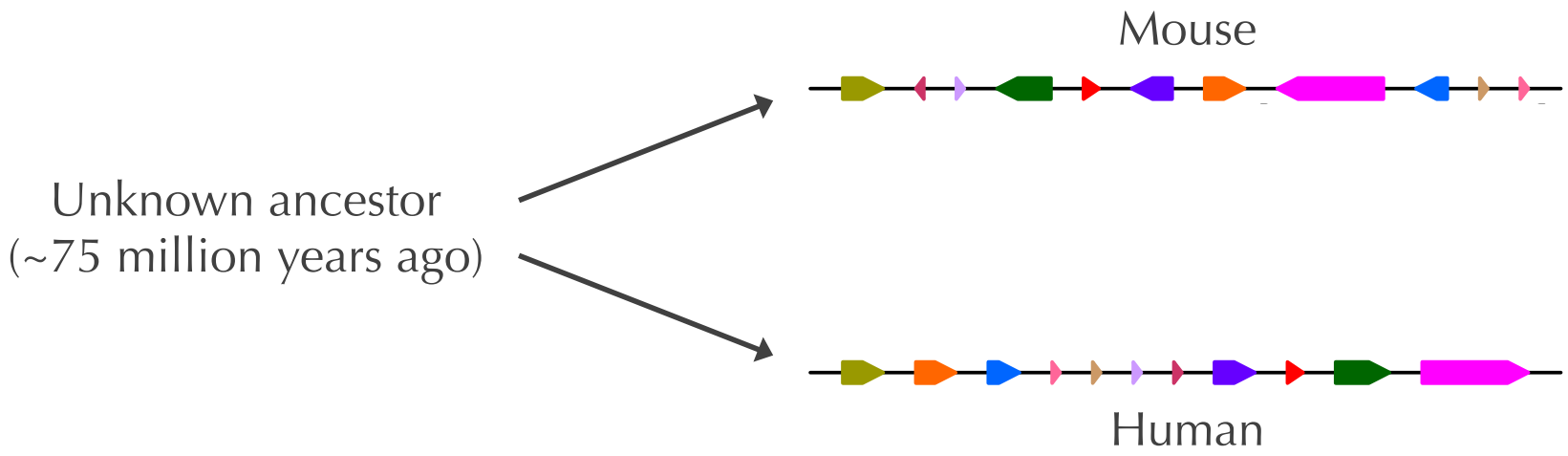
Hypothesis: Hallucinations change the underlying parameters of the system and produce Turing patterns within the visual cortex.

IDEA 2: FRAGILE GENOMES

Comparing Mouse and Human X Chromosomes

Synteny block: a procession of similar genes that appear in the same order in two different genomes.

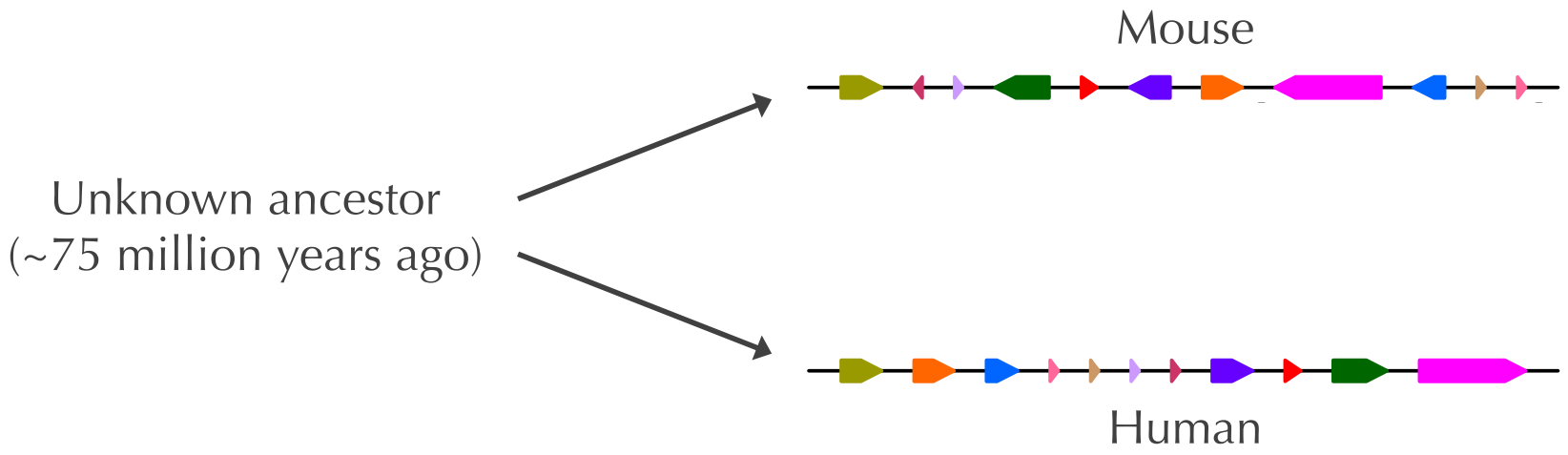
Figure below shows colored synteny blocks for the human and mouse X chromosomes.



Comparing Mouse and Human X Chromosomes

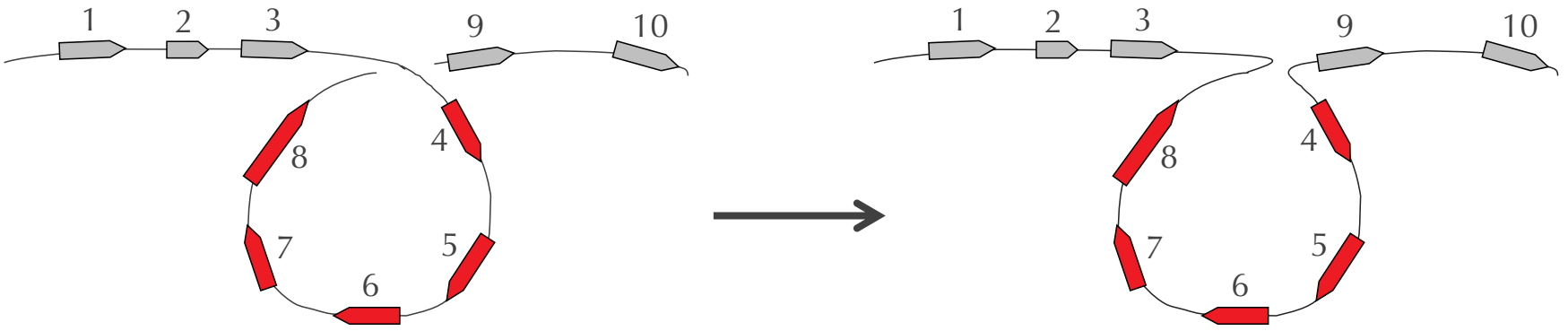
Synteny block: a procession of similar genes that appear in the same order in two different genomes.

Synteny blocks are formed and moved around as the result of large-scale **genome rearrangements**.



Sorting by Reversals

Most common form of genome rearrangement is a **reversal**, which inverts an interval of a chromosome.

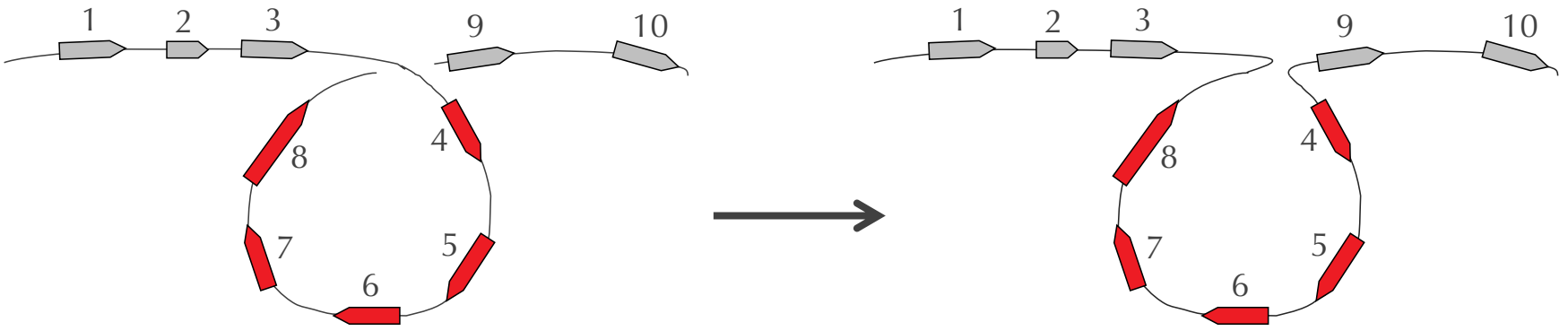


(+1 +2 +3 +4 +5 +6 +7 +8 +9 +10)

(+1 +2 +3 -8 -7 -6 -5 -4 +9 +10)

Sorting by Reversals

Most common form of genome rearrangement is a **reversal**, which inverts an interval of a chromosome.



(+1 +2 +3 +4 +5 +6 +7 +8 +9 +10)

(+1 +2 +3 -8 -7 -6 -5 -4 +9 +10)

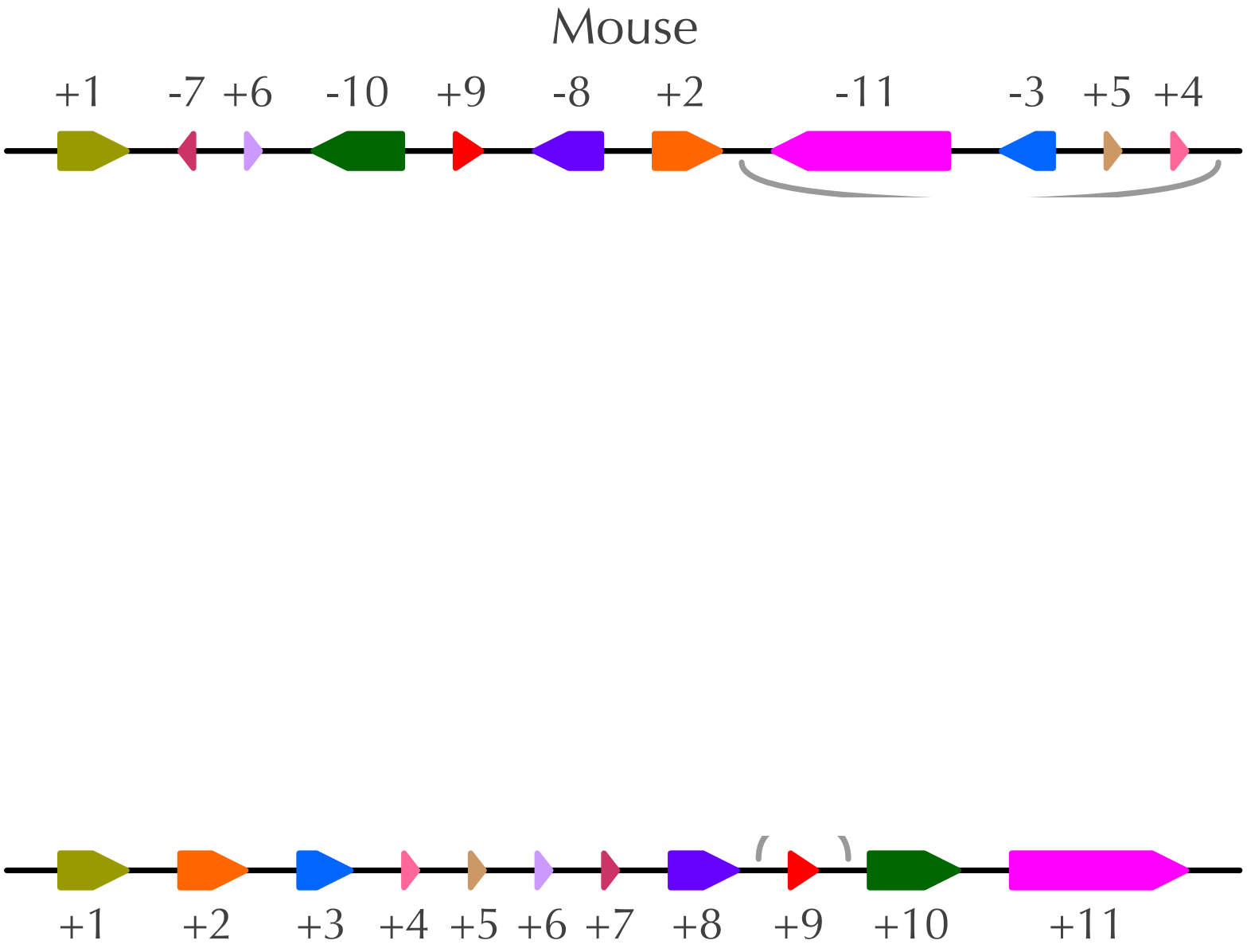
Labeling synteny blocks produces a **signed permutation**.

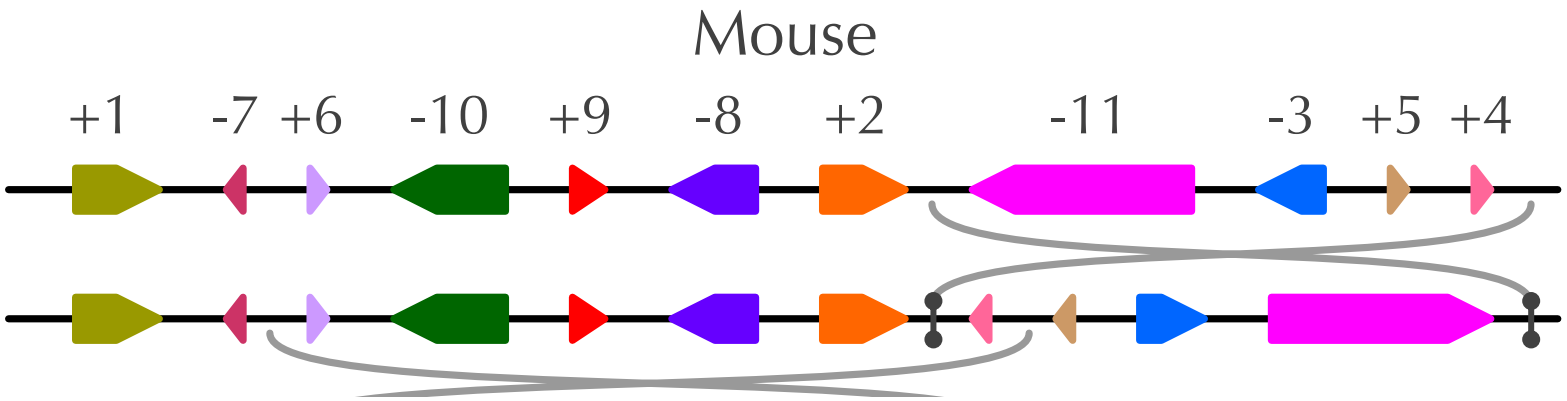
Sorting by Reversals

Sorting by Reversals Problem:

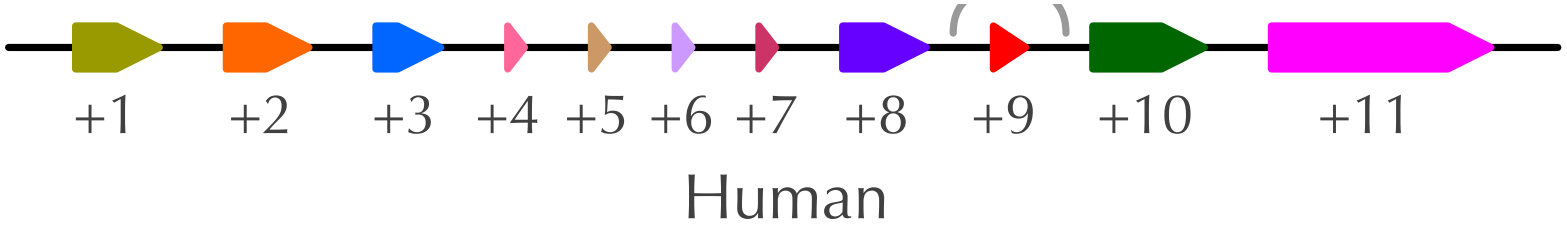
- **Input:** Two signed permutations P and Q.
- **Output:** A minimum length collection of reversals transforming P into Q.

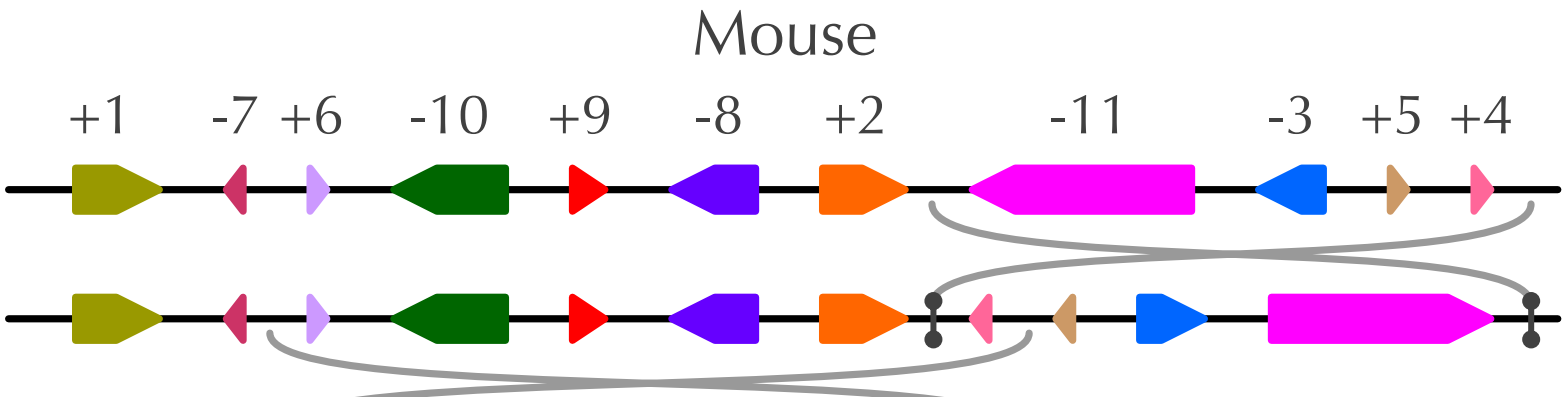
There is a (complicated) polynomial-time algorithm solving this problem, but that is not what we're interested in.





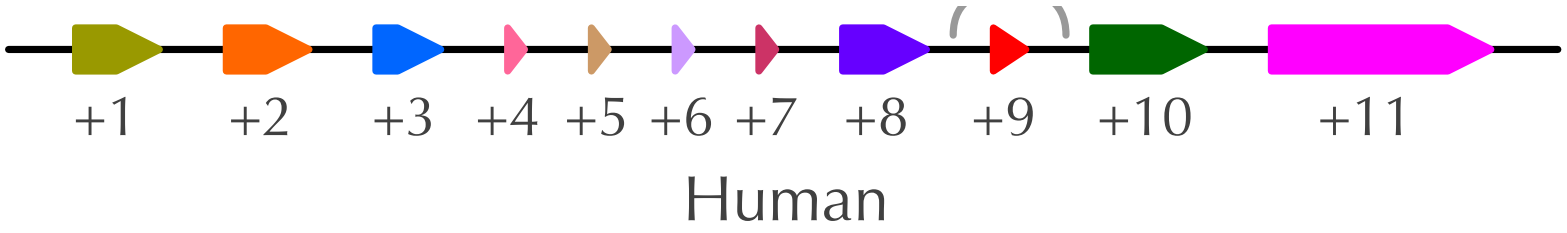
We use the vertical bars to highlight a **breakpoint** in this hypothetical transformation.

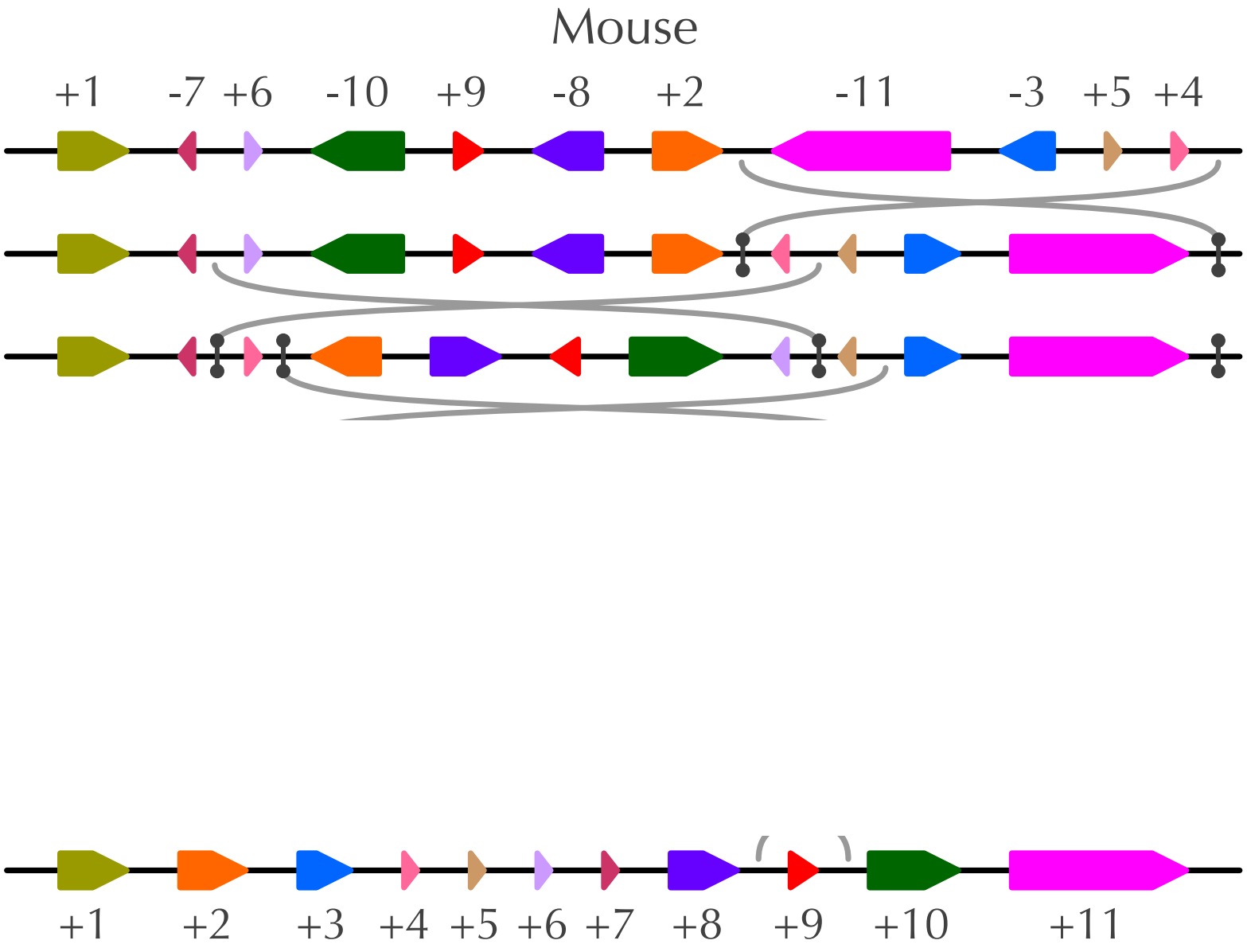


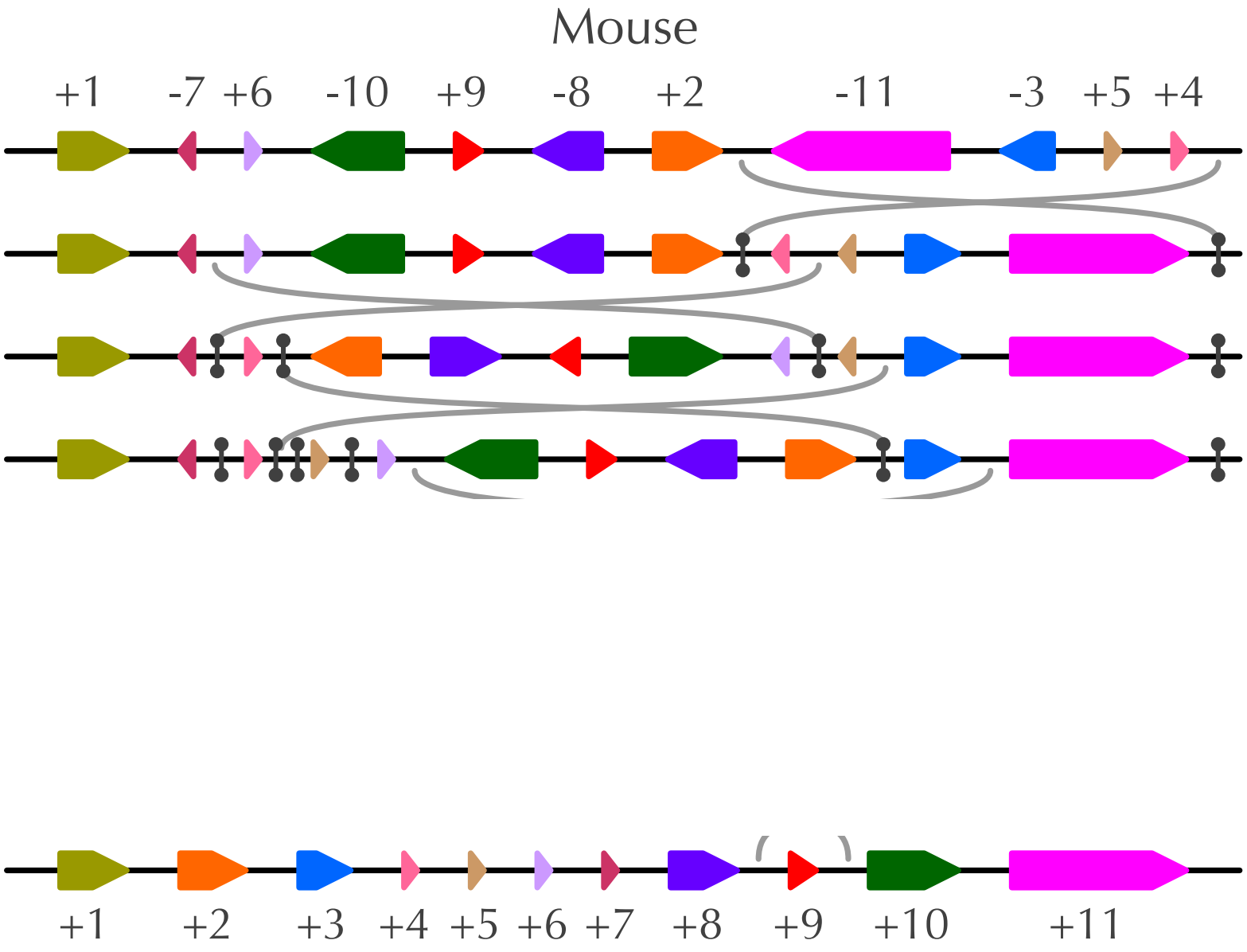


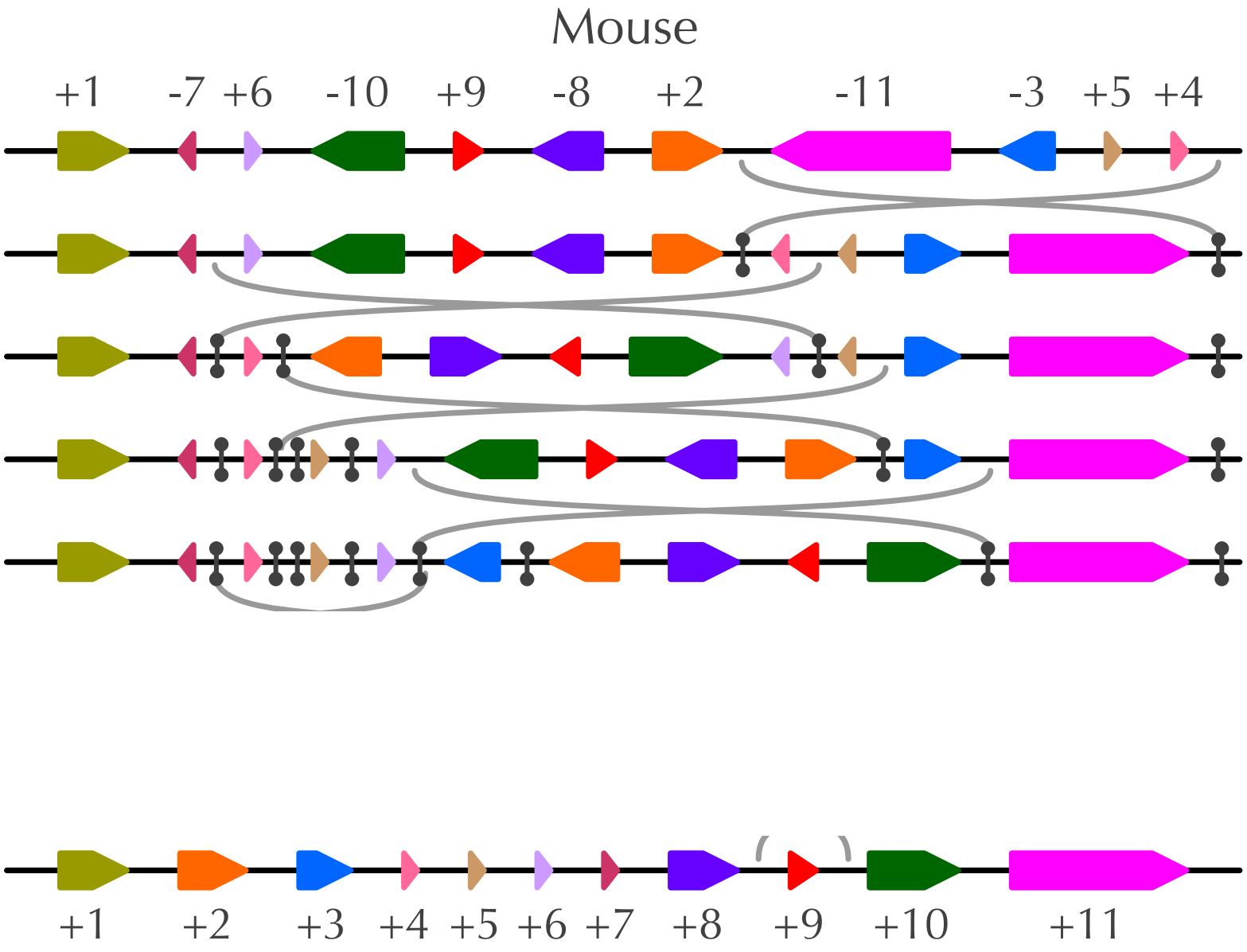
We use the vertical bars to highlight a **breakpoint** in this hypothetical transformation.

Question: Do the breakpoints get reused? That is, are there "fragile regions" or is breakage random?

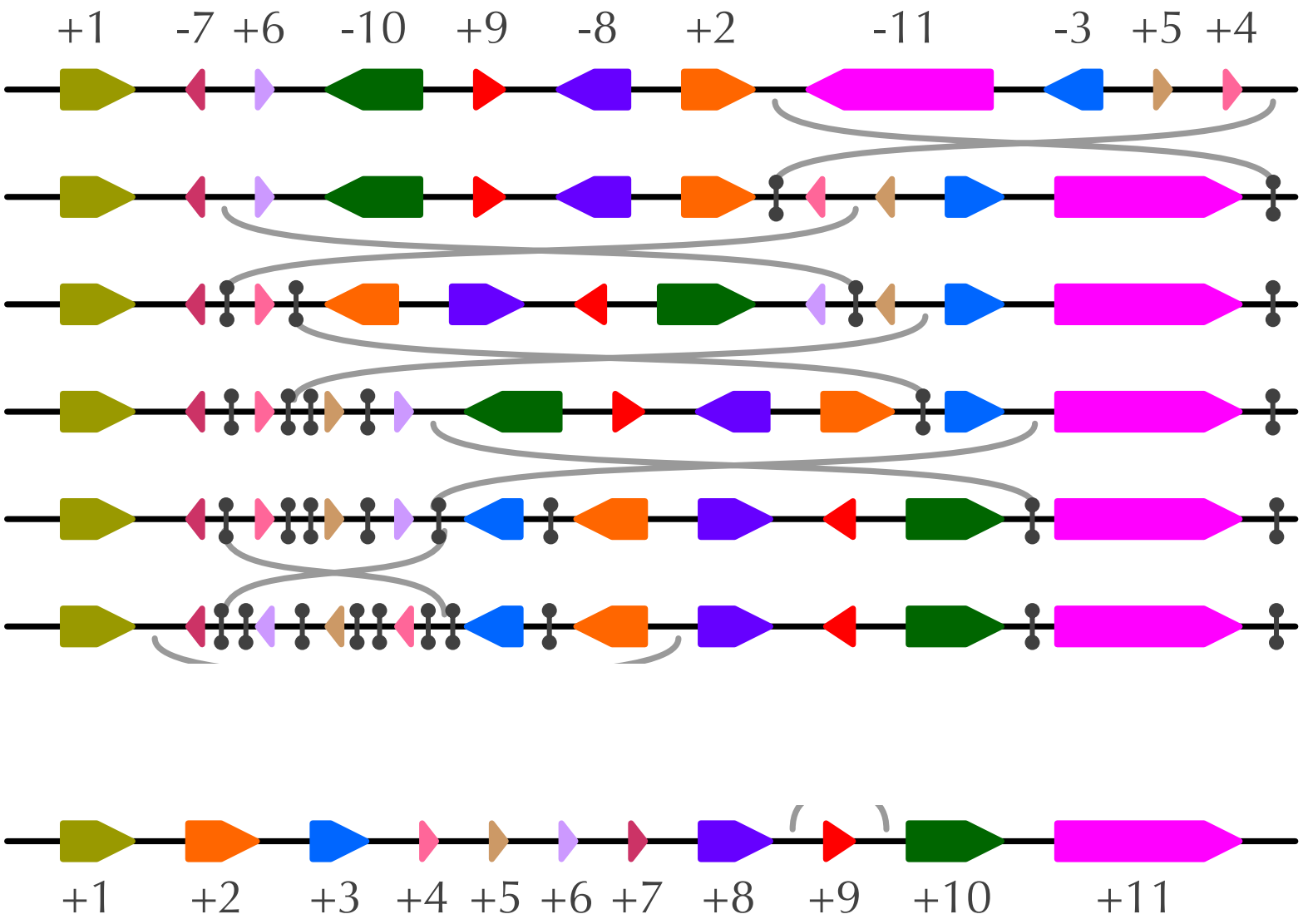


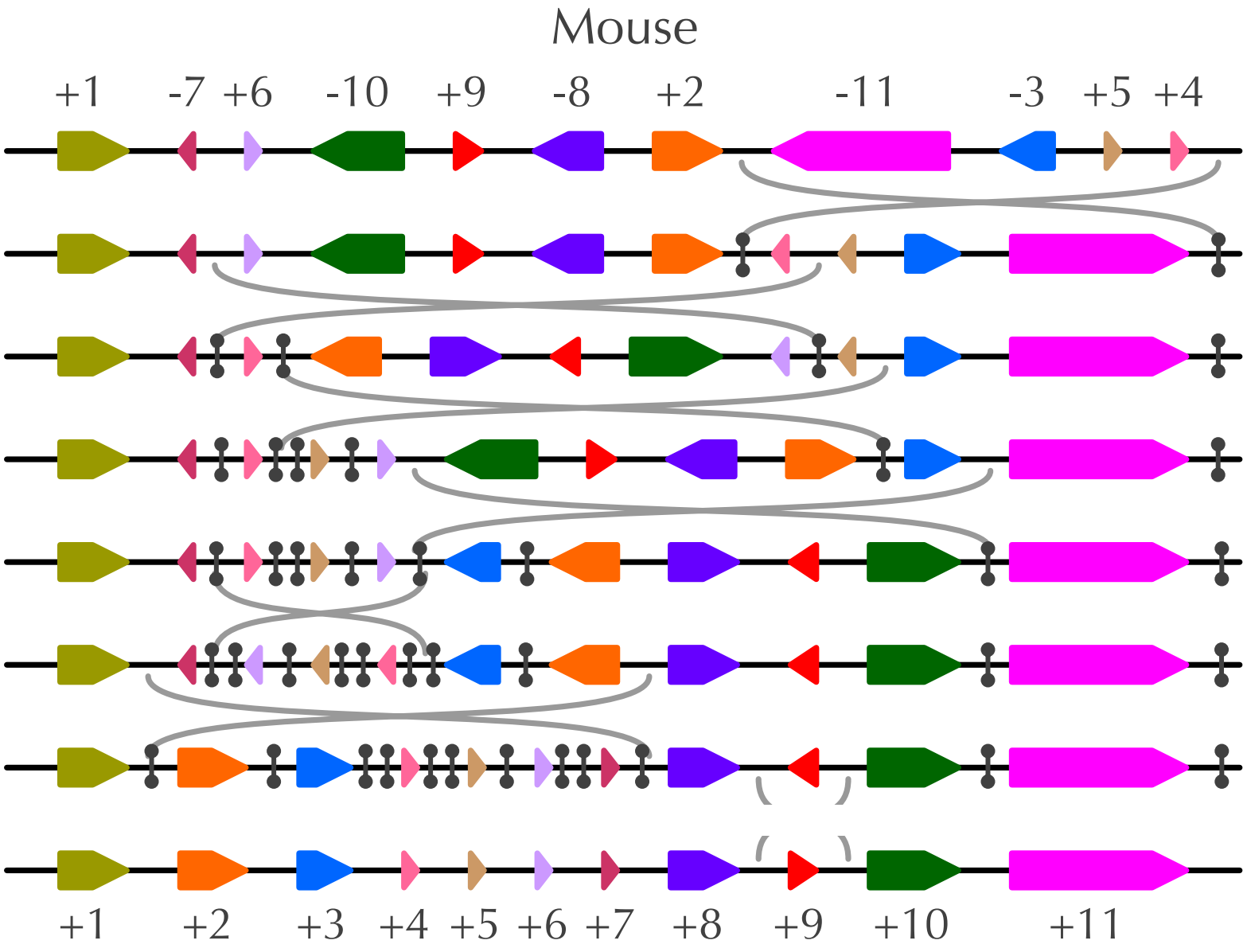


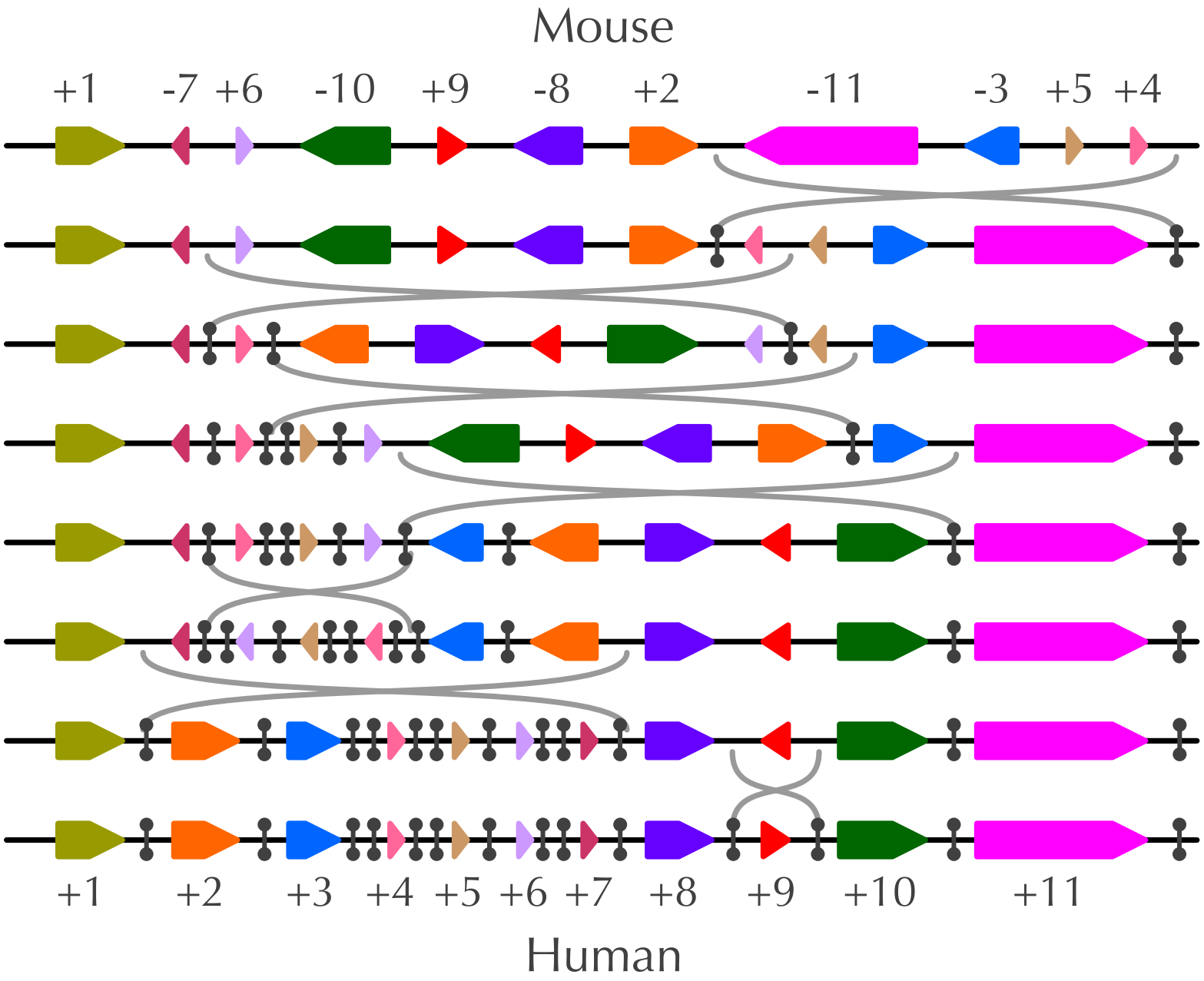




Mouse



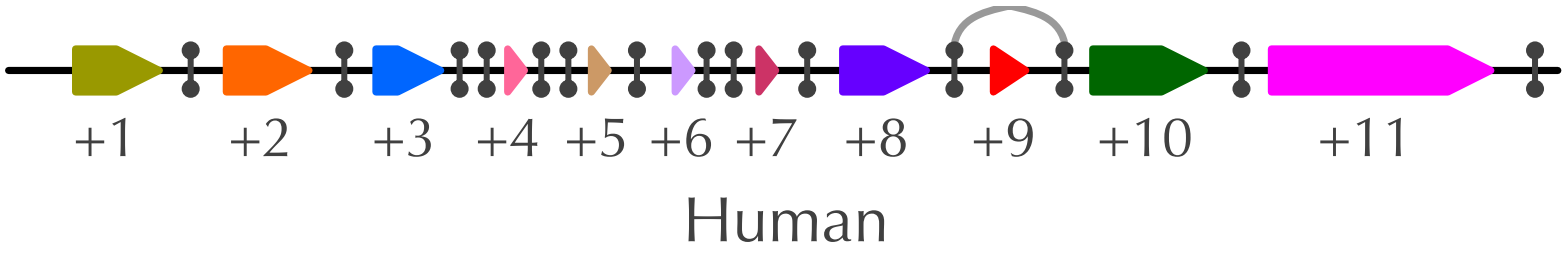




Breakpoint Reuse and Synteny Block Lengths

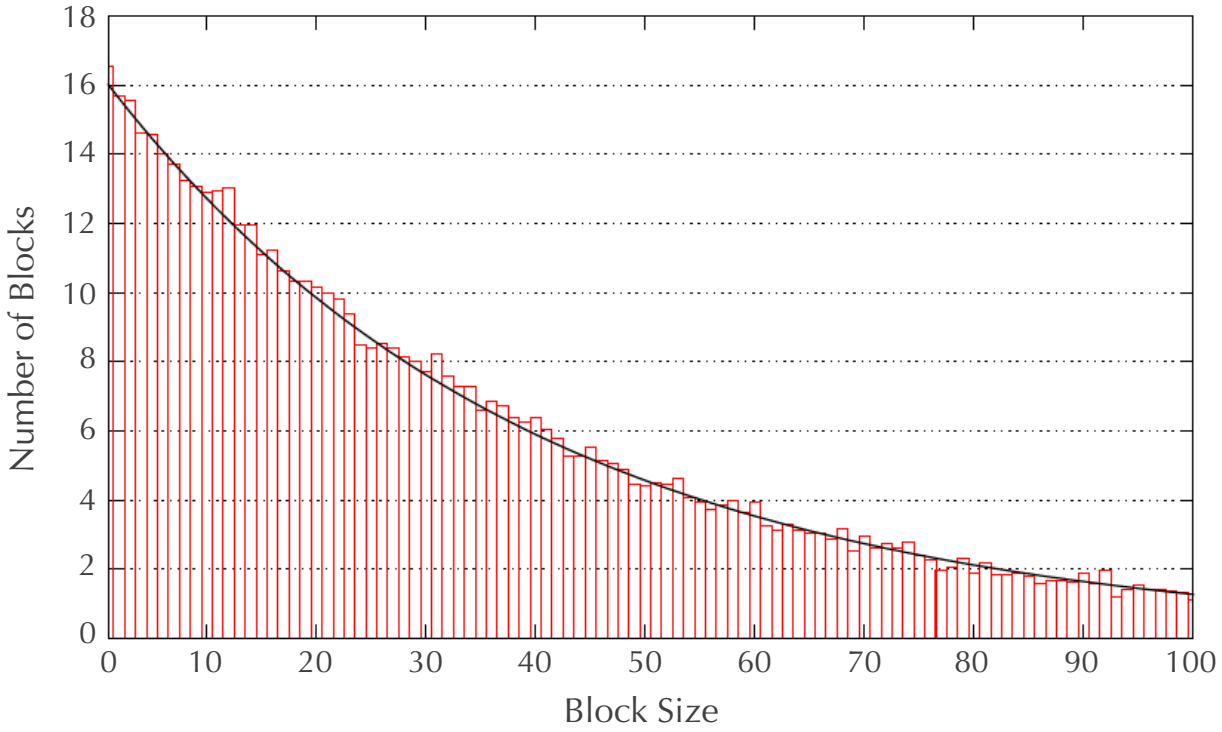
This particular rearrangement scenario had breakpoint reuse, but it's not necessarily optimal...

Checkpoint: If breakage were random, what would the distribution of synteny block lengths look like?



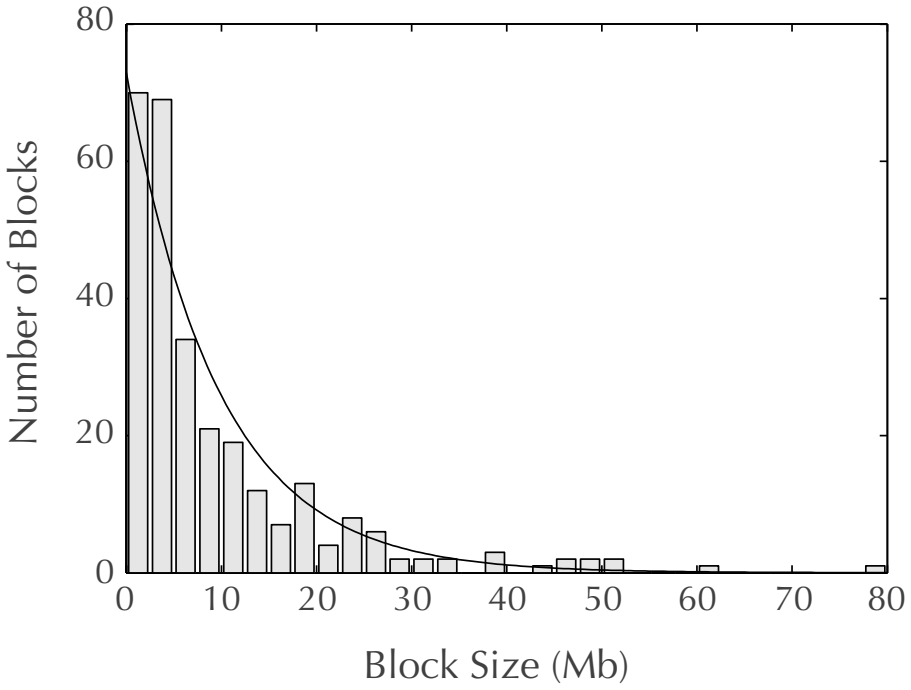
Breakpoint Reuse and Synteny Block Lengths

Answer: The synteny blocks follow the exponential distribution (simulated dataset shown below).



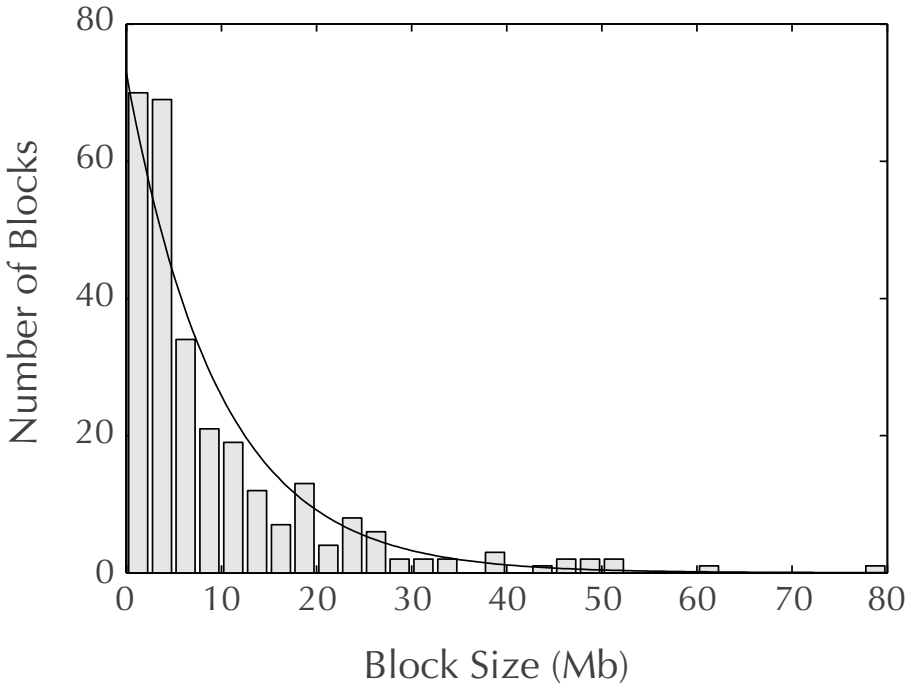
Breakpoint Reuse and Synteny Block Lengths

Checkpoint: Actual human-mouse synteny block lengths are shown below. Are we done?



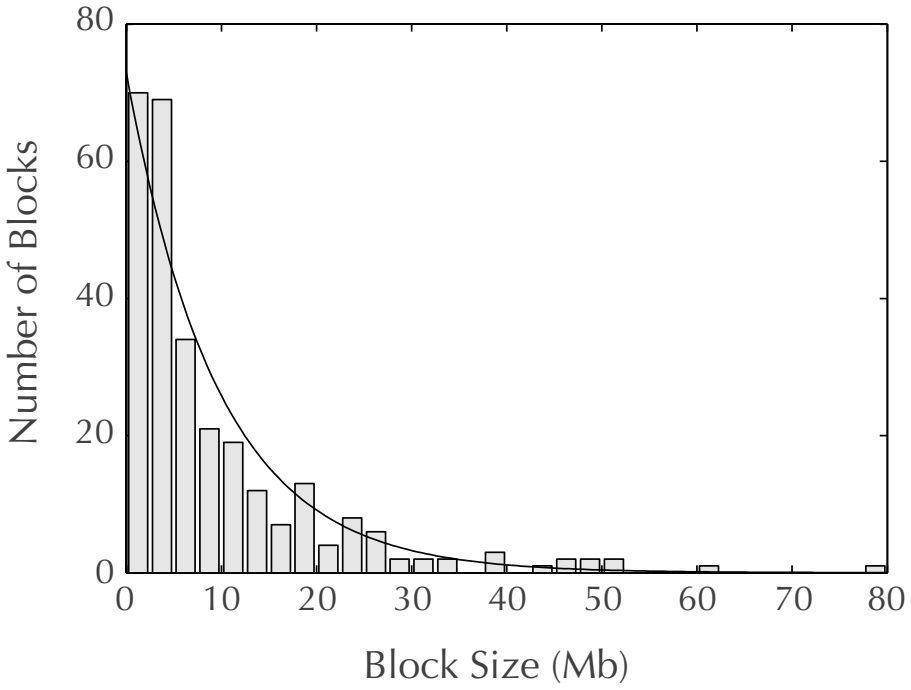
Breakpoint Reuse and Synteny Block Lengths

Answer: We certainly have *evidence* in favor of random breakage, but we don't have *proof*.



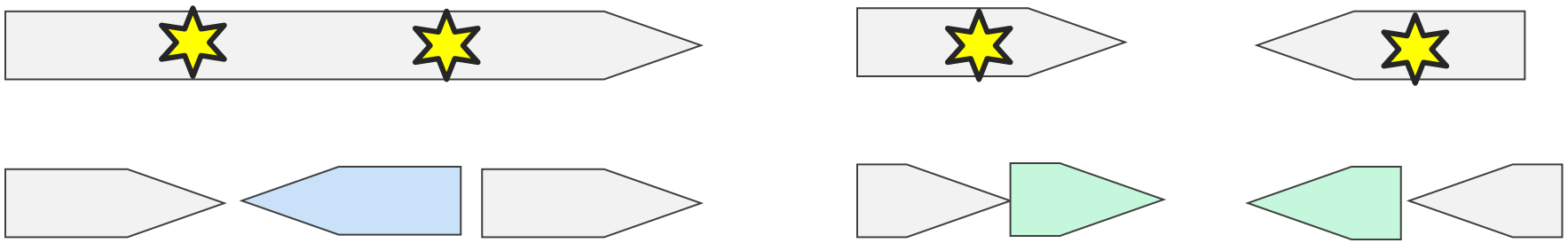
Breakpoint Reuse and Synteny Block Lengths

Checkpoint: If breakpoint use is random, then how many synteny blocks would *N* reversals form?



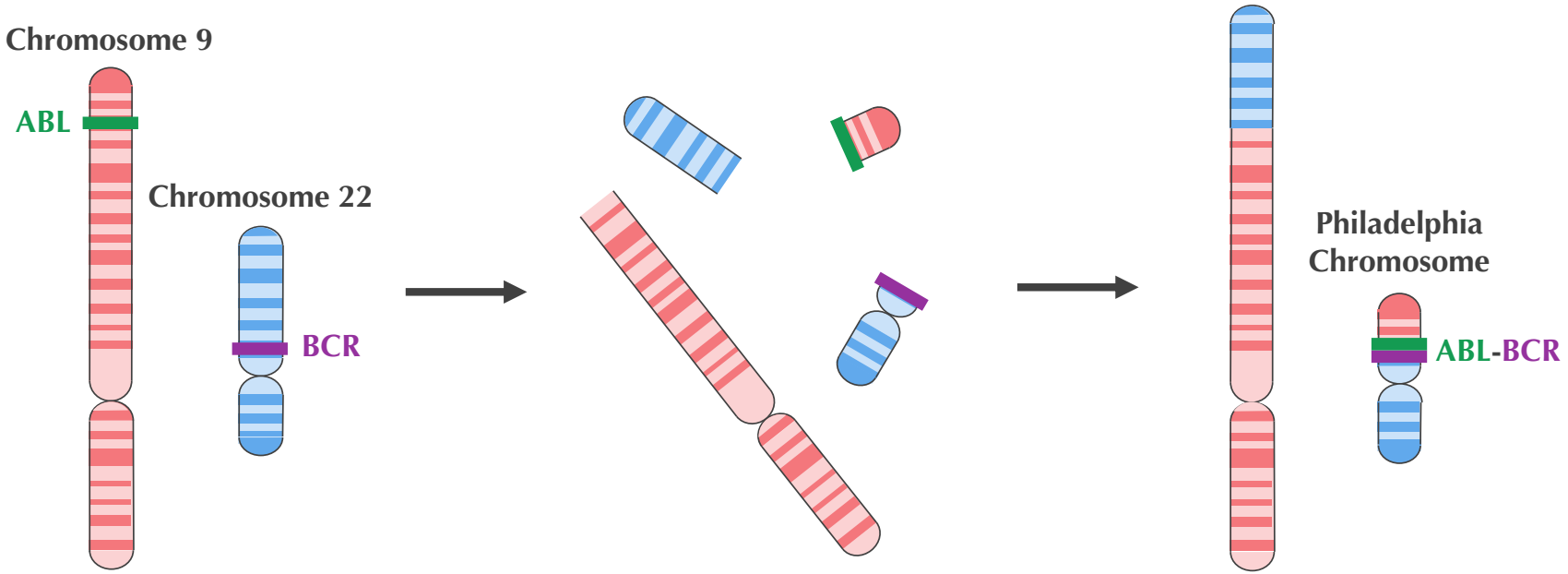
Breakpoint Reuse and Synteny Block Lengths

Answer: $2N$ because each reversal would form two new blocks; either a single block gets divided into three pieces, or two blocks get divided into four pieces.



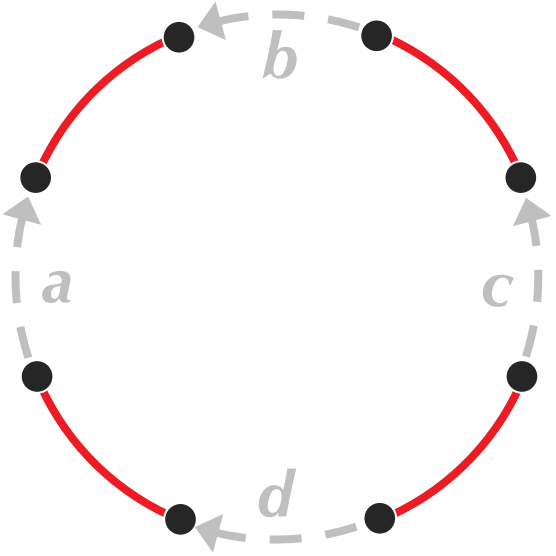
Rearrangements Can Involve Multiple Chromosomes

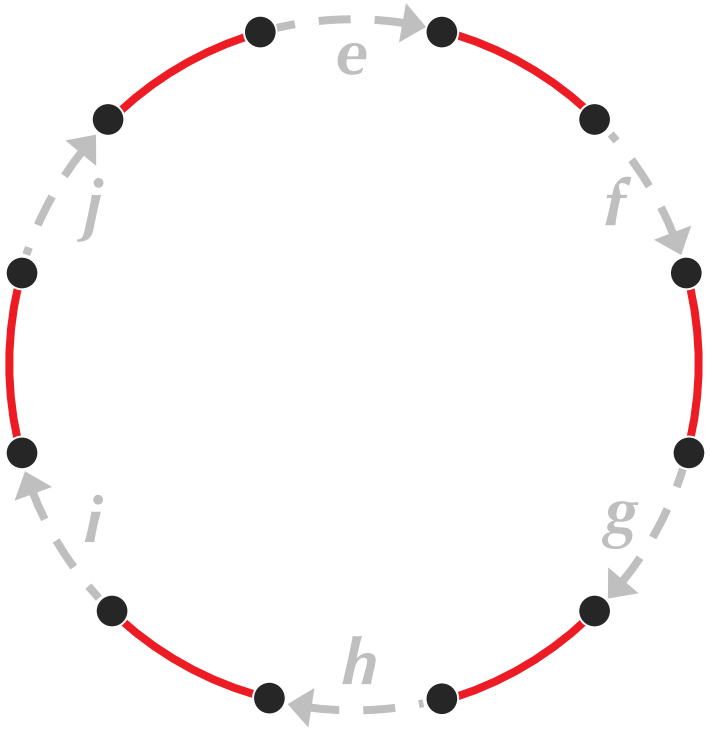
Philadelphia chromosome: leukemia-causing and formed by a **translocation** rearrangement.



From Signed Permutations to a Graph

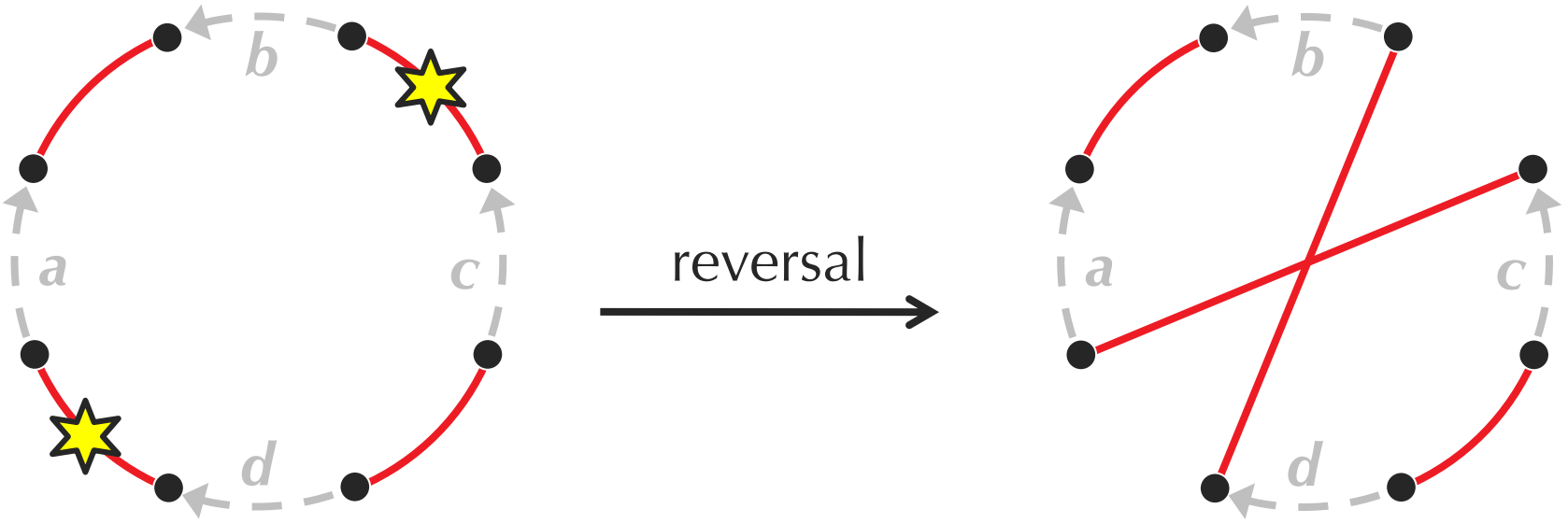
We will assume that every genome is made up of circular chromosomes.





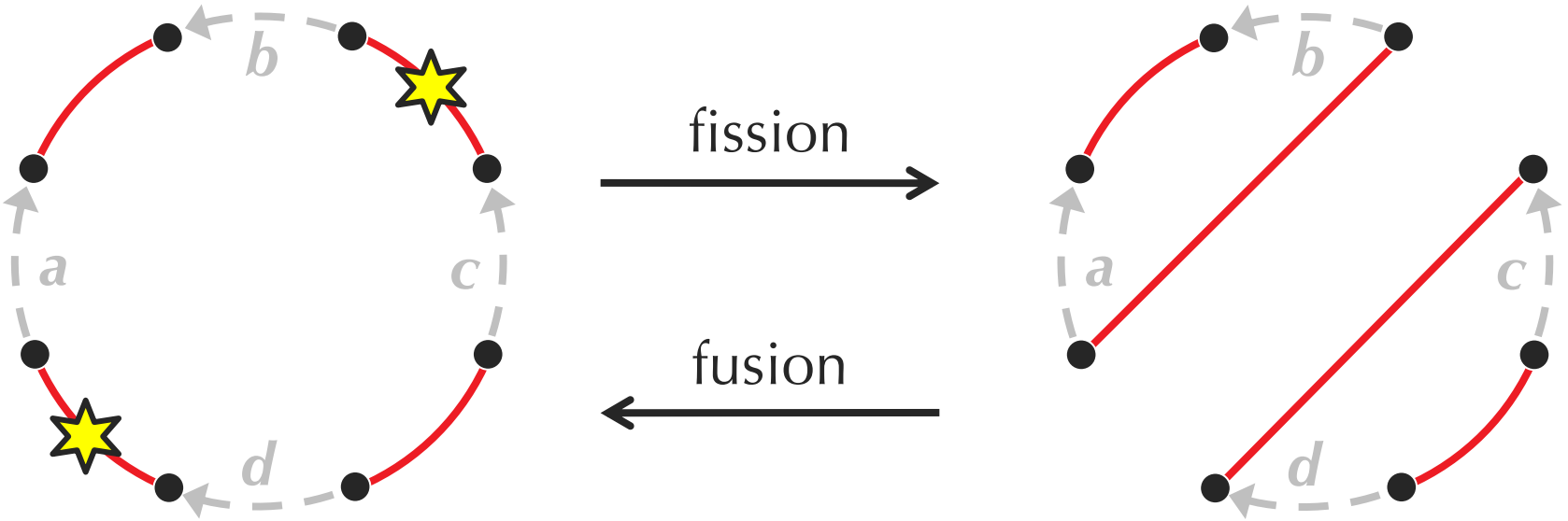
From Signed Permutations to a Graph

A reversal corresponds to breaking two red edges and rejoining them.

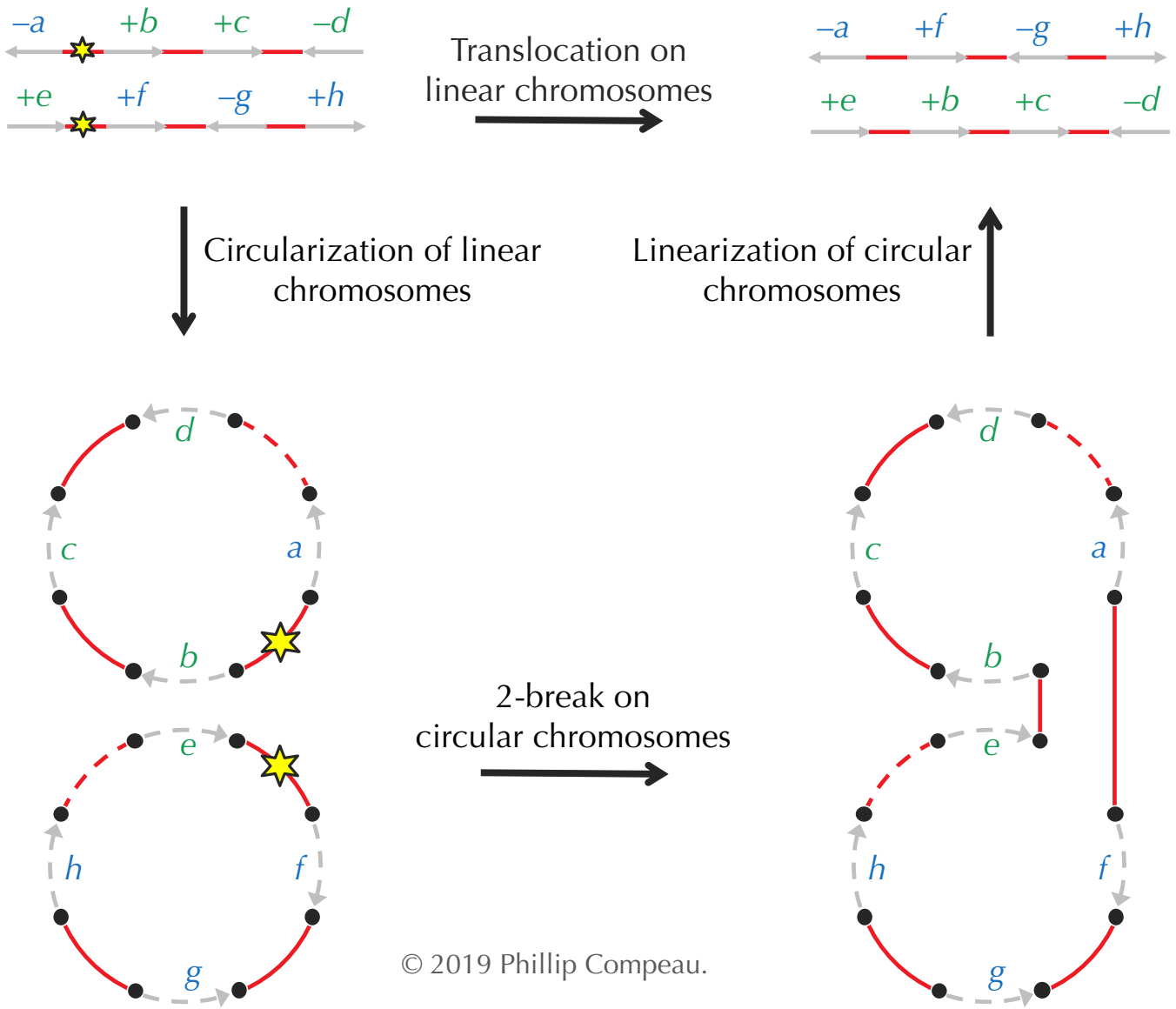


From Signed Permutations to a Graph

A **2-break operation** also allows for **fusion** and **fission** operations.



Even Translocations Look Like 2-Breaks



2-Break Sorting Problem

2-Break Sorting Problem:

- **Input:** Two genomes P and Q.
- **Output:** A minimum collection of 2-breaks transforming P into Q.

2-Break Sorting Problem

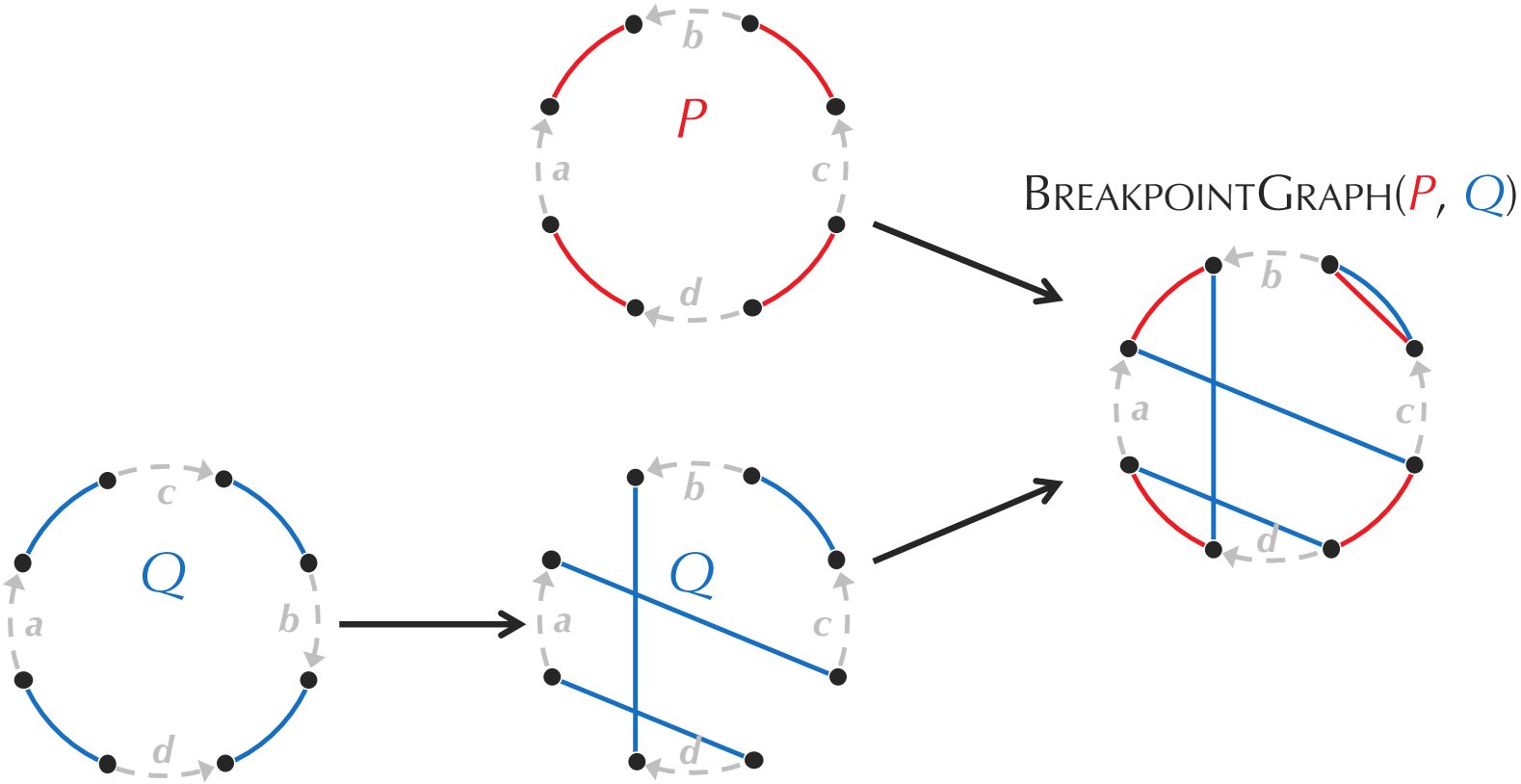
2-Break Sorting Problem:

- **Input:** Two genomes P and Q.
- **Output:** A minimum collection of 2-breaks transforming P into Q.

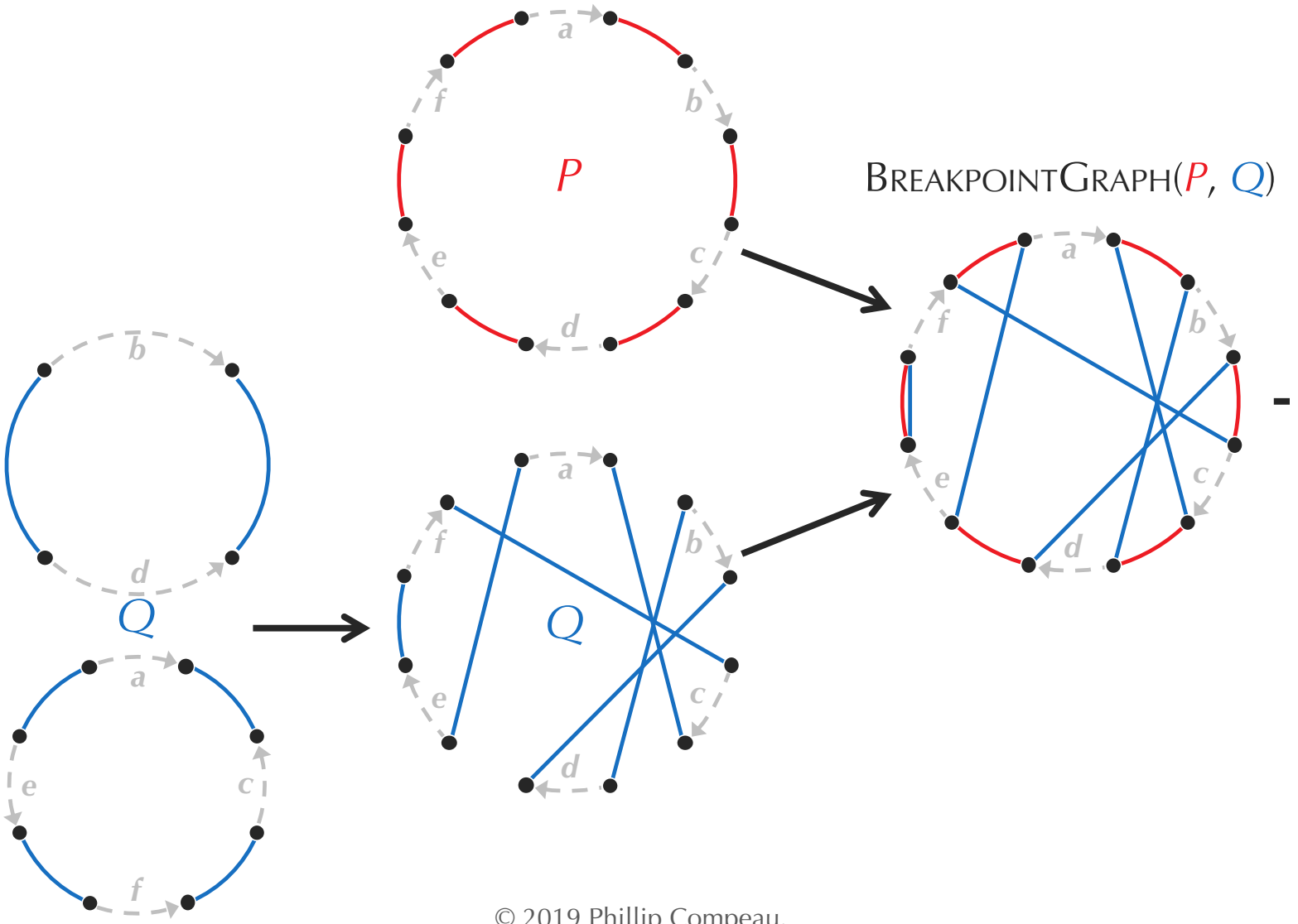
Key Point: the synteny blocks are not important; what matters is the connections between blocks (i.e., the colored edges).

Defining the Breakpoint Graph

Breakpoint Graph: formed by taking the colored edges from the genomes P and Q.

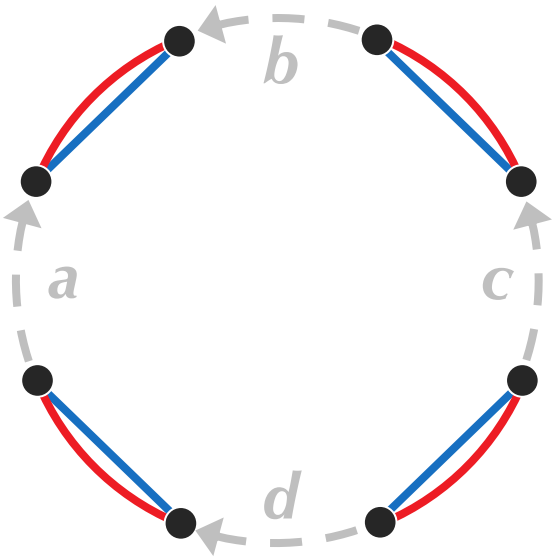


A More Complicated Example



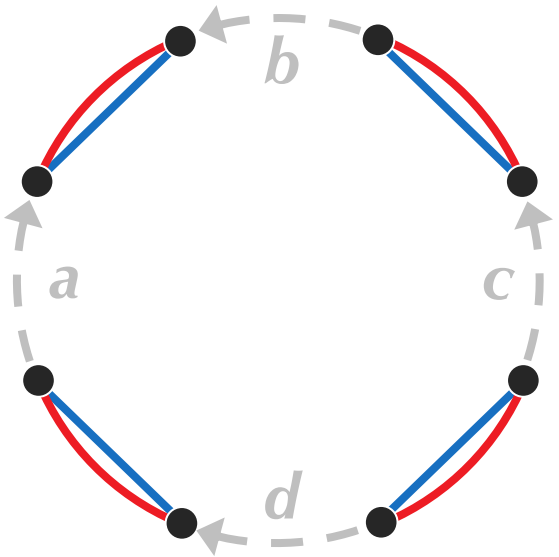
Transforming P into Q Means Increasing Red-Blue Cycles

Key Point: the more similar P and Q are, the more red-blue cycles there are. When P and Q are equal, they have $\#Blocks(P, Q)$ red-blue cycles.



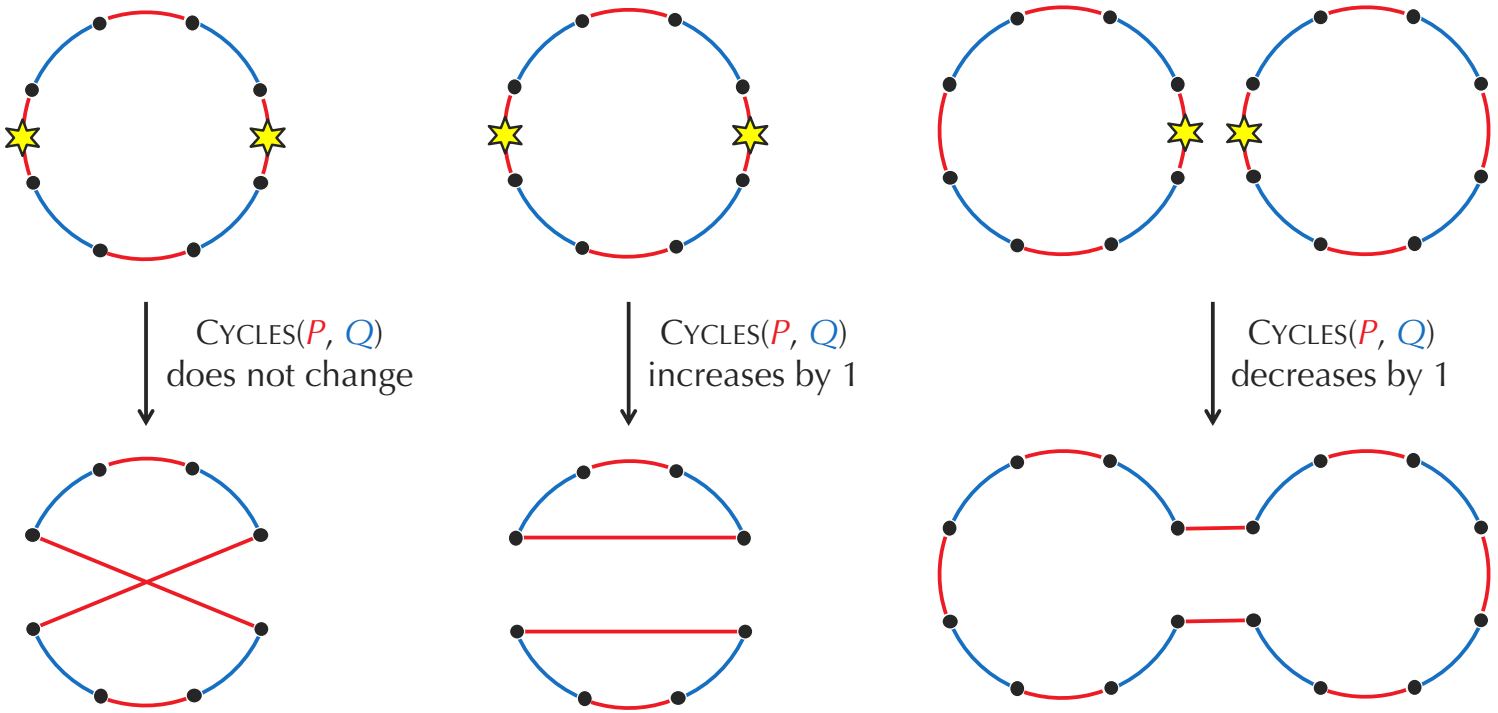
Transforming *P* into *Q* Means Increasing Red-Blue Cycles

So we are looking for a sequence of 2-breaks that *increases* the number of red-blue cycles, denoted *Cycles*(*P*, *Q*).



Transforming P into Q Means Increasing Red-Blue Cycles

Cycle Theorem: Any 2-break applied to P can change $Cycles(P, Q)$ by at most 1.



Transforming P into Q Means Increasing Red-Blue Cycles

2-Break Distance Theorem: The 2-break distance between P and Q is $Blocks(P, Q) - Cycles(P, Q)$.

Transforming P into Q Means Increasing Red-Blue Cycles

2-Break Distance Theorem: The 2-break distance between P and Q is $Blocks(P, Q) - Cycles(P, Q)$.

Proof: Unless $P = Q$, we can always find a 2-break that increases the number of cycles by 1.

Transforming P into Q Means Increasing Red-Blue Cycles

2-Break Distance Theorem: The 2-break distance between P and Q is $Blocks(P, Q) - Cycles(P, Q)$.

Proof: Unless $P = Q$, we can always find a 2-break that increases the number of cycles by 1.

Note: Much like the proof of Euler's theorem, is a "constructive proof".

Returning to Genome Fragility

When we construct the breakpoint graph of the human-mouse genome, we find:

- $Blocks(P, Q) = 280$
- $Cycles(P, Q) = 35$.

Checkpoint: So, what is the 2-break distance between human and mouse? What would it be if breakage were random?

Returning to Genome Fragility

When we construct the breakpoint graph of the human-mouse genome, we find:

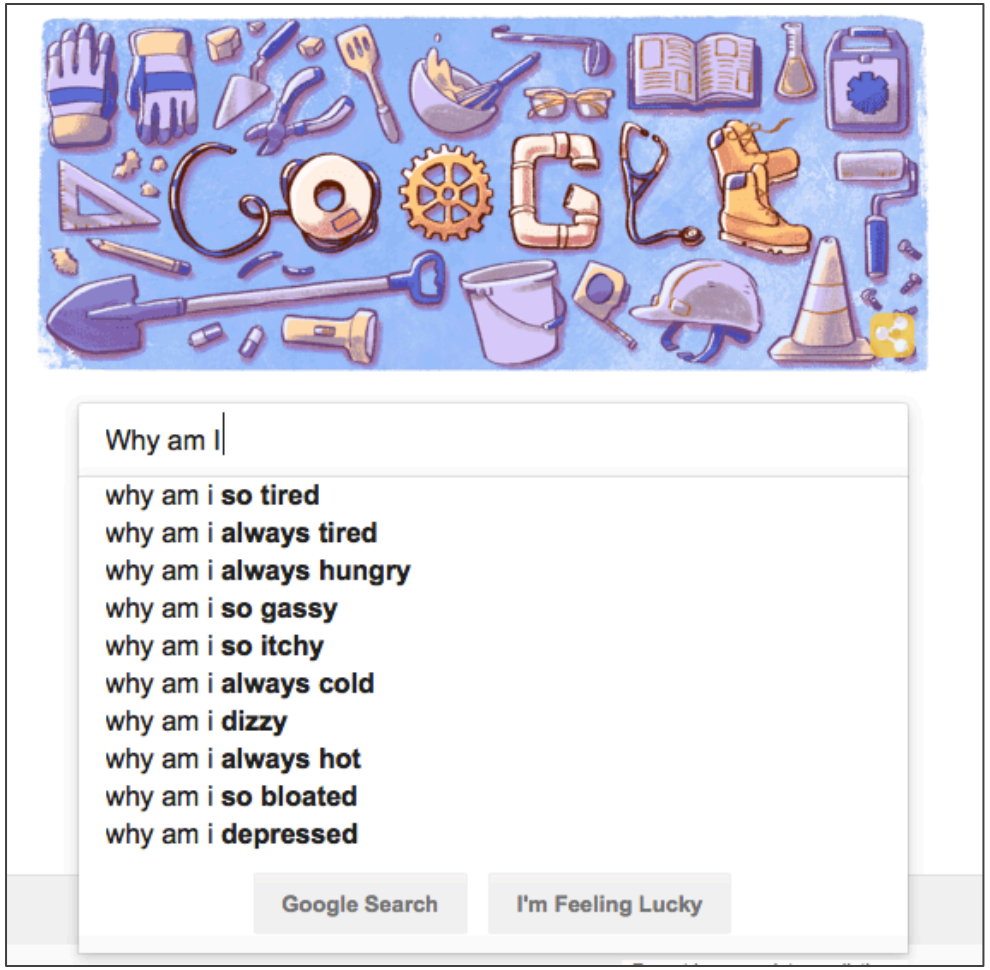
- $Blocks(P, Q) = 280$
- $Cycles(P, Q) = 35$.

Answer: In a random breakage environment, each 2-break would produce two new synteny blocks. So we would only need 140 operations. But the 2-Break Distance Theorem says that this distance is 245. So the genome must have fragile regions!

IDEA 3: DNA COMPUTING

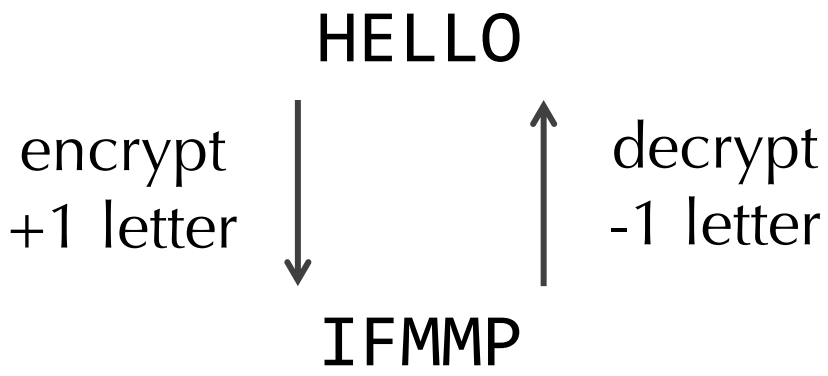
Encryption is Vital to Internet Security

Encryption: transforming a message so that it cannot be read by an eavesdropper but can be **decrypted** by the recipient.



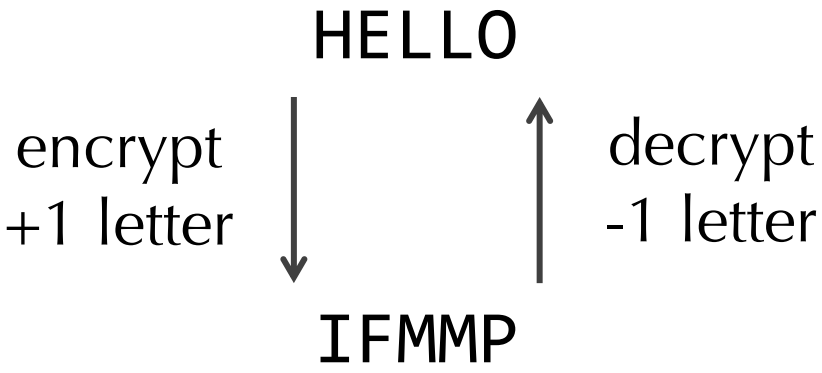
Most Encryption Schemes are Symmetric

A **symmetric** encryption scheme uses the same **key** for encrypting/decrypting a message.

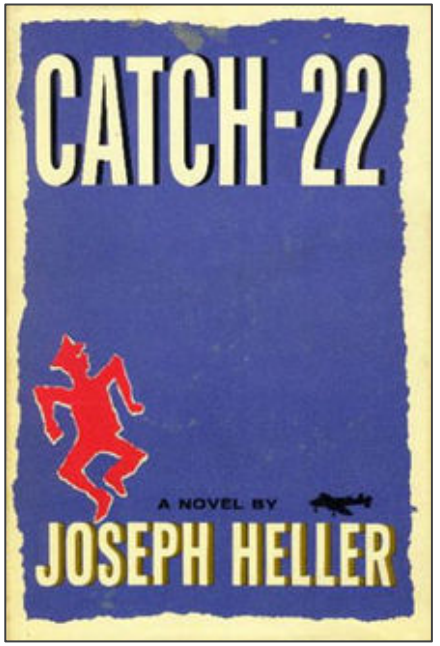


Most Encryption Schemes are Symmetric

A **symmetric** encryption scheme uses the same **key** for encrypting/decrypting a message.



Bigger problem: even if we have a complicated key, it must be kept **private**: the sender and receiver must agree on the key in advance.

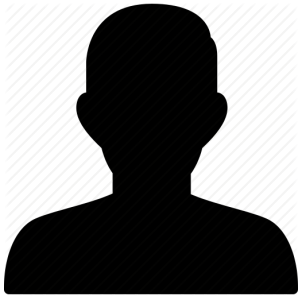


Primes Save the Day

Public key encryption (Rivest, Shamir, Adleman 1978): knowing the key doesn't make it automatically easy to decrypt!

Primes Save the Day

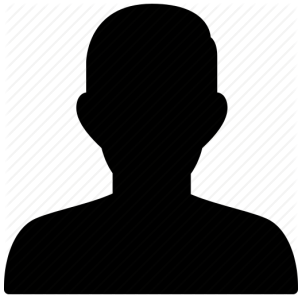
Public key encryption (Rivest, Shamir, Adleman 1978): knowing the key doesn't make it automatically easy to decrypt!



Picks two large primes p and q
(typically ~300 digits long)

Primes Save the Day

Public key encryption (Rivest, Shamir, Adleman 1978): knowing the key doesn't make it automatically easy to decrypt!



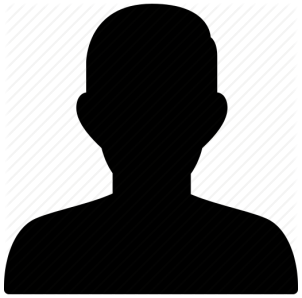
Use n to encrypt

Public Key
$n = p * q$

Picks two large primes p and q
(typically ~300 digits long)

Primes Save the Day

Public key encryption (Rivest, Shamir, Adleman 1978): knowing the key doesn't make it automatically easy to decrypt!



Use n to encrypt

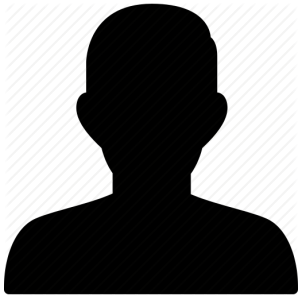
Public Key
$n = p * q$

Picks two large primes p and q
(typically ~300 digits long)

Key Point: The only way to decrypt is by knowing the primes p and q. This makes the key **asymmetric**.

Primes Save the Day

Public key encryption (Rivest, Shamir, Adleman 1978): knowing the key doesn't make it automatically easy to decrypt!



Use n to encrypt

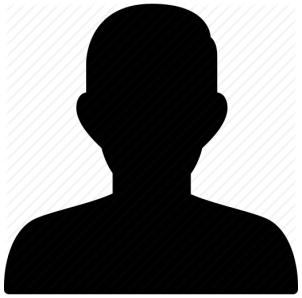
Public Key
$n = p * q$

Picks two large primes p and q
(typically ~300 digits long)

Note 1: This was discovered in 1973 by Clifford Cocks but not declassified by the UK for 25 years.

Primes Save the Day

Public key encryption (Rivest, Shamir, Adleman 1978): knowing the key doesn't make it automatically easy to decrypt!



Use n to encrypt

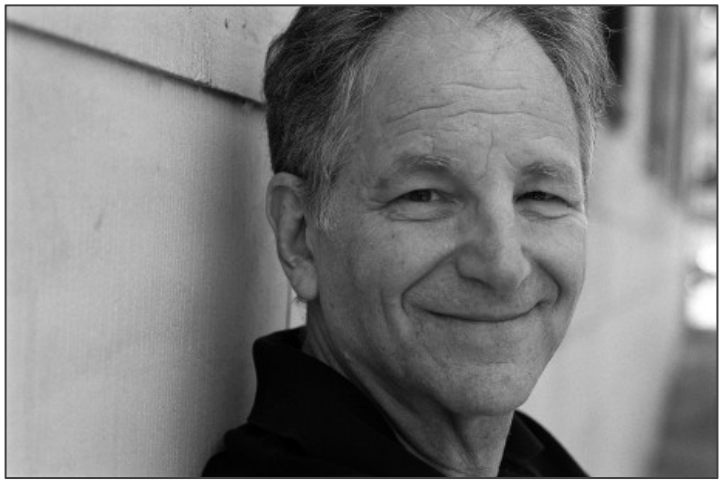
Public Key
$n = p * q$

Picks two large primes p and q
(typically ~300 digits long)

Note 2: "RSA" and its descendants are foundation of almost every secure internet transaction.

Primes Save the Day

Public key encryption (Rivest, Shamir, **Adleman** 1978): knowing the key doesn't make it automatically easy to decrypt!



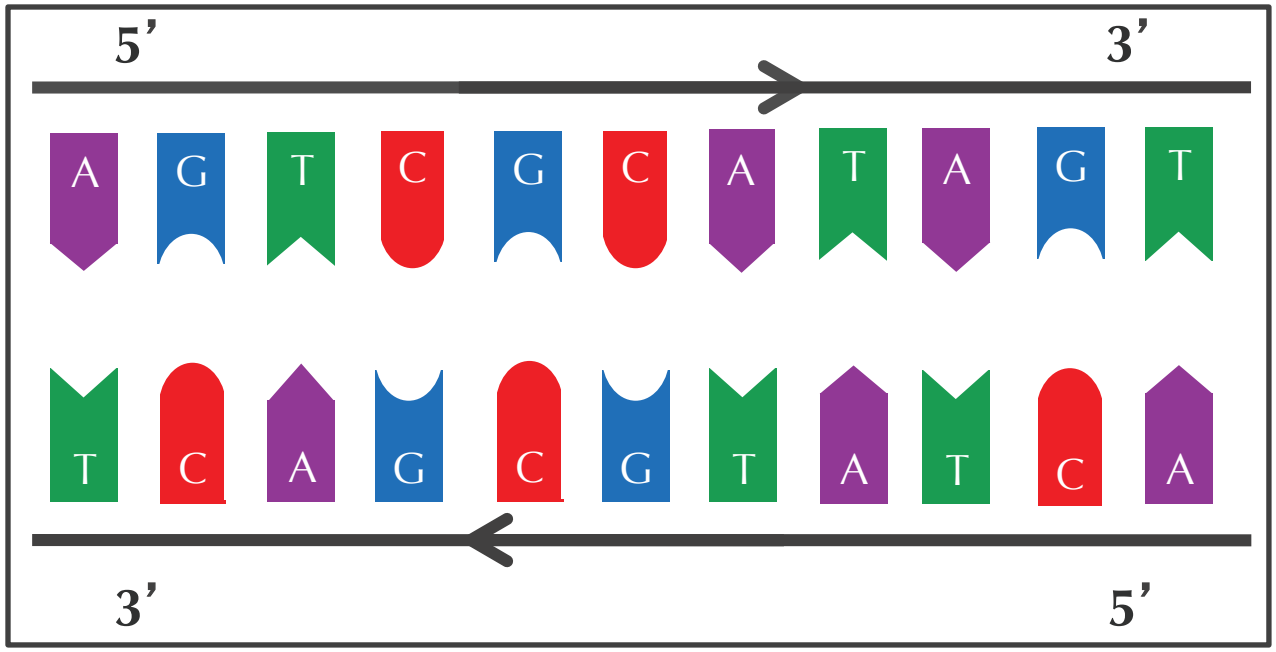
Leonard Adleman

https://www.heidelberg-laureate-forum.org/blog/laureate/leonard-max-adleman/

Note 3: Adleman is famous for two things: RSA and as the founder of "DNA computing".

Overview of DNA Computing

DNA Computing: Using DNA as hardware of computer due to its molecular-scale storage capabilities.



Barriers to DNA Computing

DNA Computing: Using DNA as hardware of computer due to its molecular-scale storage capabilities.

Checkpoint: What barriers do you see for using DNA as a system of storage?

Barriers to DNA Computing

DNA Computing: Using DNA as hardware of computer due to its molecular-scale storage capabilities.

Checkpoint: What barriers do you see for using DNA as a system of storage?

Answer: Reading DNA is expensive, and (until recently) editing DNA has been impossible.

Some DNA Manipulations are Easy

DNA Computing: Using DNA as hardware of computer due to its molecular-scale storage capabilities.

However, there are some things that aren't hard:

- Synthesizing a strand (oligonucleotide) of DNA.

Some DNA Manipulations are Easy

DNA Computing: Using DNA as hardware of computer due to its molecular-scale storage capabilities.

However, there are some things that aren't hard:

- Synthesizing a strand (oligonucleotide) of DNA.
- Forcing a DNA strand to form its complementary strand given a soup of free nucleotides.

Some DNA Manipulations are Easy

DNA Computing: Using DNA as hardware of computer due to its molecular-scale storage capabilities.

However, there are some things that aren't hard:

- Synthesizing a strand (oligonucleotide) of DNA.
- Forcing a DNA strand to form its complementary strand given a soup of free nucleotides.
- Finding all fragments of DNA in a sample of some approximate length.

Some DNA Manipulations are Easy

DNA Computing: Using DNA as hardware of computer due to its molecular-scale storage capabilities.

However, there are some things that aren't hard:

- Synthesizing a strand (oligonucleotide) of DNA.
- Forcing a DNA strand to form its complementary strand given a soup of free nucleotides.
- Finding all fragments of DNA in a sample of some approximate length.
- Amplifying a strand of DNA with given start/end into many copies (PCR, Nobel Prize in 1993).

Recall: Easy and Difficult Problems

Hamiltonian Cycle Problem

NP-Complete

Input: a directed network with *n* nodes.
Output: "Yes" if there is a cycle visiting every ***node*** in the network; "No" otherwise.

Eulerian Cycle Problem

Polynomial

Input: a directed network with *n* nodes.
Output: "Yes" if there is a cycle visiting every ***edge*** in the network; "No" otherwise.

Recall: Easy and Difficult Problems

Hamiltonian Cycle Problem

NP-Complete

Input: a directed network with *n* nodes.
Output: "Yes" if there is a cycle visiting every ***node*** in the network; "No" otherwise.

In particular, all *NP*-Complete problems are equivalent; if we solve the Hamiltonian Cycle Problem, we solve them all.

Recall: Easy and Difficult Problems

Hamiltonian Cycle Problem

NP-Complete

Input: a directed network with *n* nodes.

Output: "Yes" if there is a cycle visiting every ***node*** in the network; "No" otherwise.

Adleman's insight: Rather than trying to use DNA as storage, why not use it to *solve* difficult problems?

Recall: Easy and Difficult Problems

Hamiltonian Cycle Problem

NP-Complete

Input: a directed network with *n* nodes.
Output: "Yes" if there is a cycle visiting every ***node*** in the network; "No" otherwise.

Adleman's insight: Rather than trying to use DNA as storage, why not use it to *solve* difficult problems?

The difficulty here is that it's not clear at all what it means to "program" a DNA computer.

Adleman's Algorithm

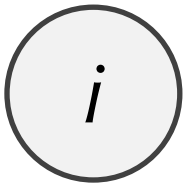
Algorithm for Determining if there is Hamiltonian Path in Graph *G* Connecting v_1 to v_n

1. Generate many "random paths" through *G* to ensure that any Hamiltonian path is present.
2. Keep only those paths that begin with v_1 and end with v_n .
3. Keep only those paths that have *n* nodes.
4. Keep only those paths that enter all the nodes of the graph at least once.
5. If any paths remain, return "Yes"; otherwise, return "No".

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.

Associate every node i of G with a DNA k-mer denoted O_i . Call its reverse complement O'_i .



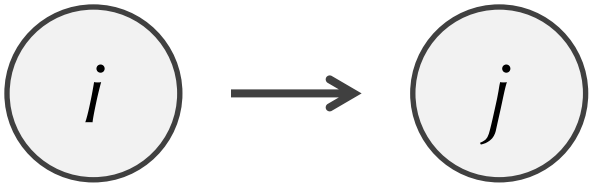
O_i `TGACGC`

O'_i `ACTGCG`

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.

Associate every edge (i, j) with a DNA k-mer $E_{i,j}$ consisting of last $k/2$ symbols of O_i followed by first $k/2$ symbols of O_j . (Preserves edge orientation.)



O_i TGACGC

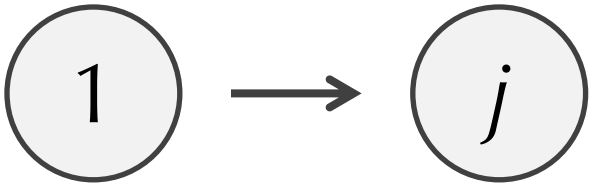
O_j AAGACT

$E_{i,j}$ CGCAAG

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.

Note: If $i = 1$, use all k symbols of O_1. If $j = n$, use all k symbols of O_n .



O_1 `CATTAT`

O_j `AAGACT`

$E_{1,j}$ `CATTATAAG`

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.

Produce many copies of $E_{i,j}$ for every edge (i, j).
Produce many copies of O'_i for every node other than v_1 and v_n .

Checkpoint: What will happen when we combine these DNA strands?

Converting Each Step to Experiment

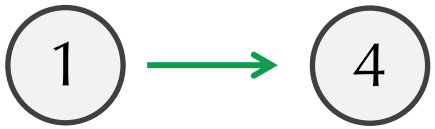
1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.

Produce many copies of $E_{i,j}$ for every edge (i, j).
Produce many copies of O'_i for every node other than v_1 and v_n .

Answer: edge $E_{i,j}$ will hybridize to O'_i and O'_j .
Adjacent edges will therefore join into a path.

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.



$E_{1,4}$

CATTATAAG

TTCTGA

O'_4

O_1 CATTAT

O_2 CGTCCA

O_4 AAGACT

O_7 CTTTAG

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.

$E_{1,4}$ $E_{4,2}$

CATTATAAGAAGCGT

TTCTGA

O'_4

O_1 CATTAT

O_2 CGTCCA

O_4 AAGACT

O_7 CTTTAG

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.

$E_{1,4}$ $E_{4,2}$

CATTATAAGAAGCGT

TTCTGAGCAGGT

O'_4 O'_2

O_1 CATTAT

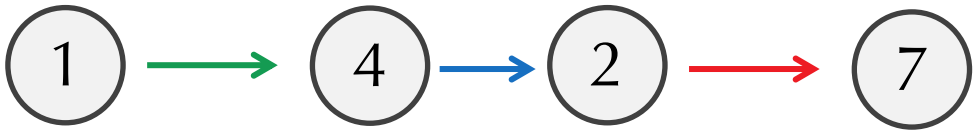
O_2 CGTCCA

O_4 AAGACT

O_7 CTTTAG

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.



$E_{1,4}$ $E_{4,2}$ $E_{2,7}$

CATTATAAGAAGCGTCCACTTTAG

TTCTGAGCAGGT

O'_4 O'_2

O_1	CATTAT
O_2	CGTCCA
O_4	AAGACT
O_7	CTTTAG

Converting Each Step to Experiment

1. Generate many "random paths" through G to ensure that any Hamiltonian path is present.

As a result, every path in the graph will be present as some double-stranded DNA molecule.

Converting Each Step to Experiment

2. Keep only those paths that begin with v_1 and end with v_n .

Use PCR to amplify all pieces of DNA that begin with O_1 and end with O_n .

Converting Each Step to Experiment

3. Keep only those paths that have *n* nodes.

Throw out all fragments of DNA that don't have length approximately equal to *n* * *k* nucleotides.

Converting Each Step to Experiment

4. Keep only those paths that enter all the nodes of the graph at least once.

Convert all DNA to single strands, and hybridize DNA against O'_i for some *i*. Retain only strands that have annealing. Repeat for all O'_i.

`CATTATAAGAAGCGTCCACTTTAG`

O_1 `CATTAT`

O_2 `CGTCCA`

O_3 `GACCGT`

Converting Each Step to Experiment

4. Keep only those paths that enter all the nodes of the graph at least once.

Convert all DNA to single strands, and hybridize DNA against O'_i for some *i*. Retain only strands that have annealing. Repeat for all O'_i.

CATTATAAGAAGCGTCCACTTTAG

GTAATA

O'_1

O_1 CATTAT

O_2 CGTCCA

O_3 GACCGT

Converting Each Step to Experiment

4. Keep only those paths that enter all the nodes of the graph at least once.

Convert all DNA to single strands, and hybridize DNA against O'_i for some *i*. Retain only strands that have annealing. Repeat for all O'_i.

CATTATAAGAAGCGTCCACTTTAG

GCAGGT

O'_2

O_1 CATTAT

O_2 CGTCCA

O_3 GACCGT

Converting Each Step to Experiment

4. Keep only those paths that enter all the nodes of the graph at least once.

Convert all DNA to single strands, and hybridize DNA against O'_i for some *i*. Retain only strands that have annealing. Repeat for all O'_i.

`CATTATAAGAAGCGTCCACTTTAG`

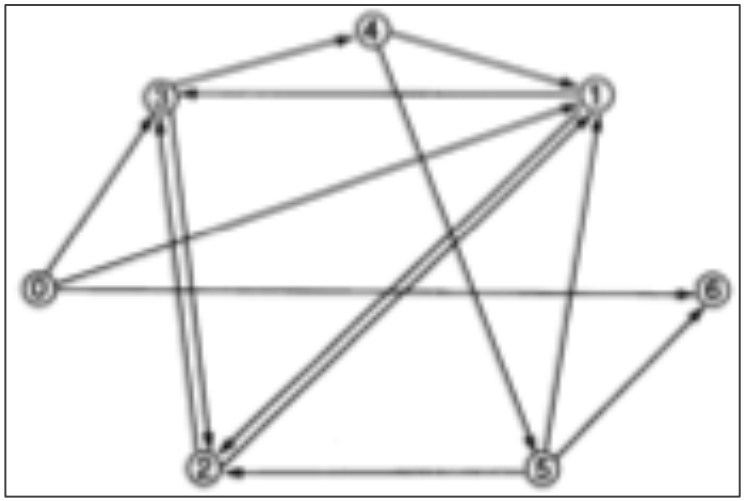
O'_3 = `CTGGCA` doesn't align!

O_1	`CATTAT`
O_2	`CGTCCA`
O_3	`GACCGT`

Converting Each Step to Experiment

5. If any paths remain, return “Yes”; otherwise, return “No”.

If any DNA remains from our experiment, then we know that the answer must be “Yes”! Otherwise, it is “No”.



https://grid.cs.gsu.edu/~wkim/index_files/hpp94.pdf

Adleman’s original graph is shown above.

We've Solved an *NP*-Complete Problem!

Checkpoint: What issues do you see with this approach?

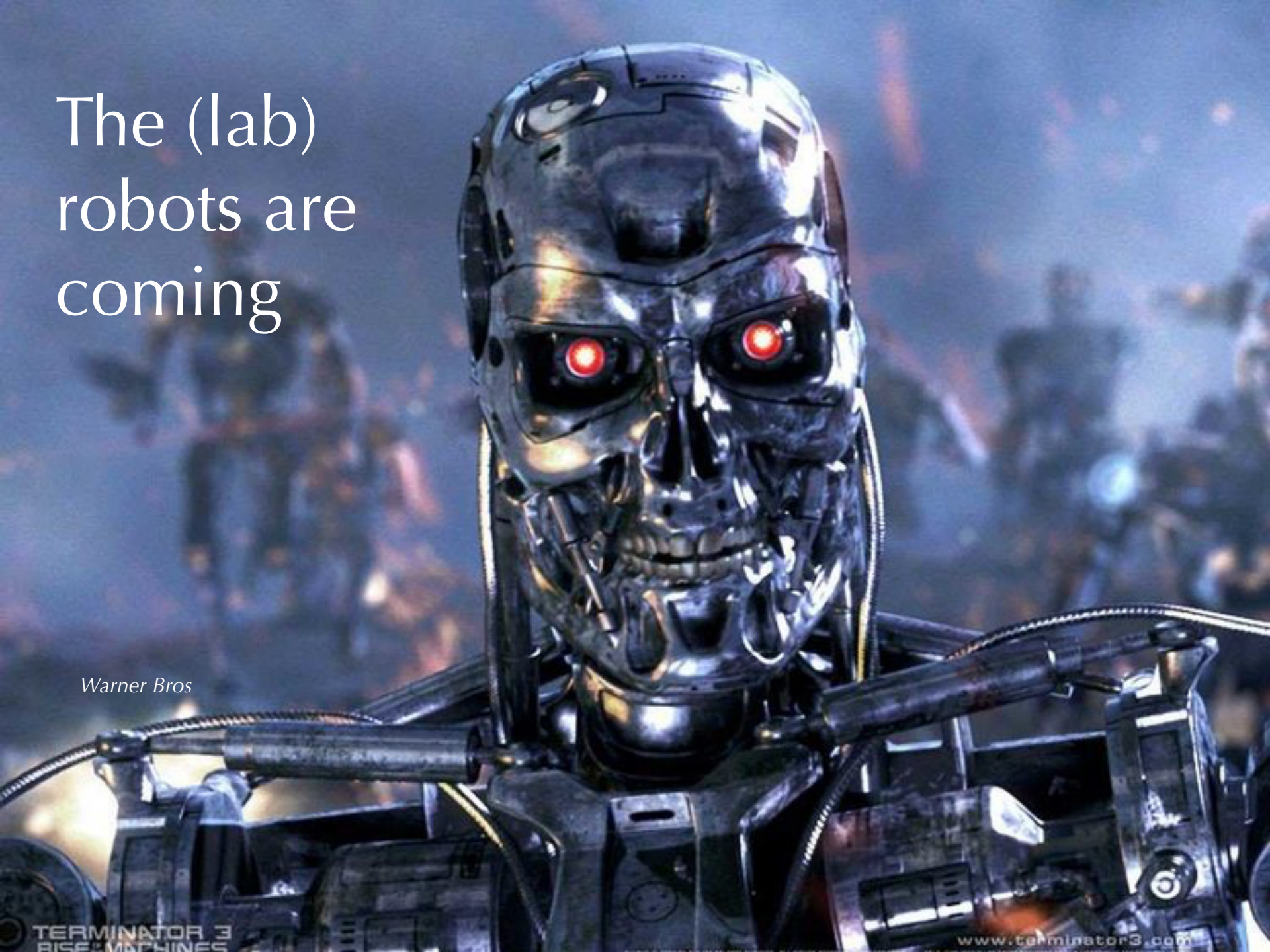
We've Solved an *NP*-Complete Problem!

Checkpoint: What issues do you see with this approach?

Answer: Three immediate barriers:

1. Possibility of errors is high.
2. We still need to generate, at a minimum, $n!$ strands of DNA. So this is impossible for networks with, say, 100 nodes.
3. An enormous amount of lab work needs to be done, with hours of waiting times.

The (lab) robots are coming



Warner Bros

THANK YOU! AND NOW IT'S YOUR TURN ☺