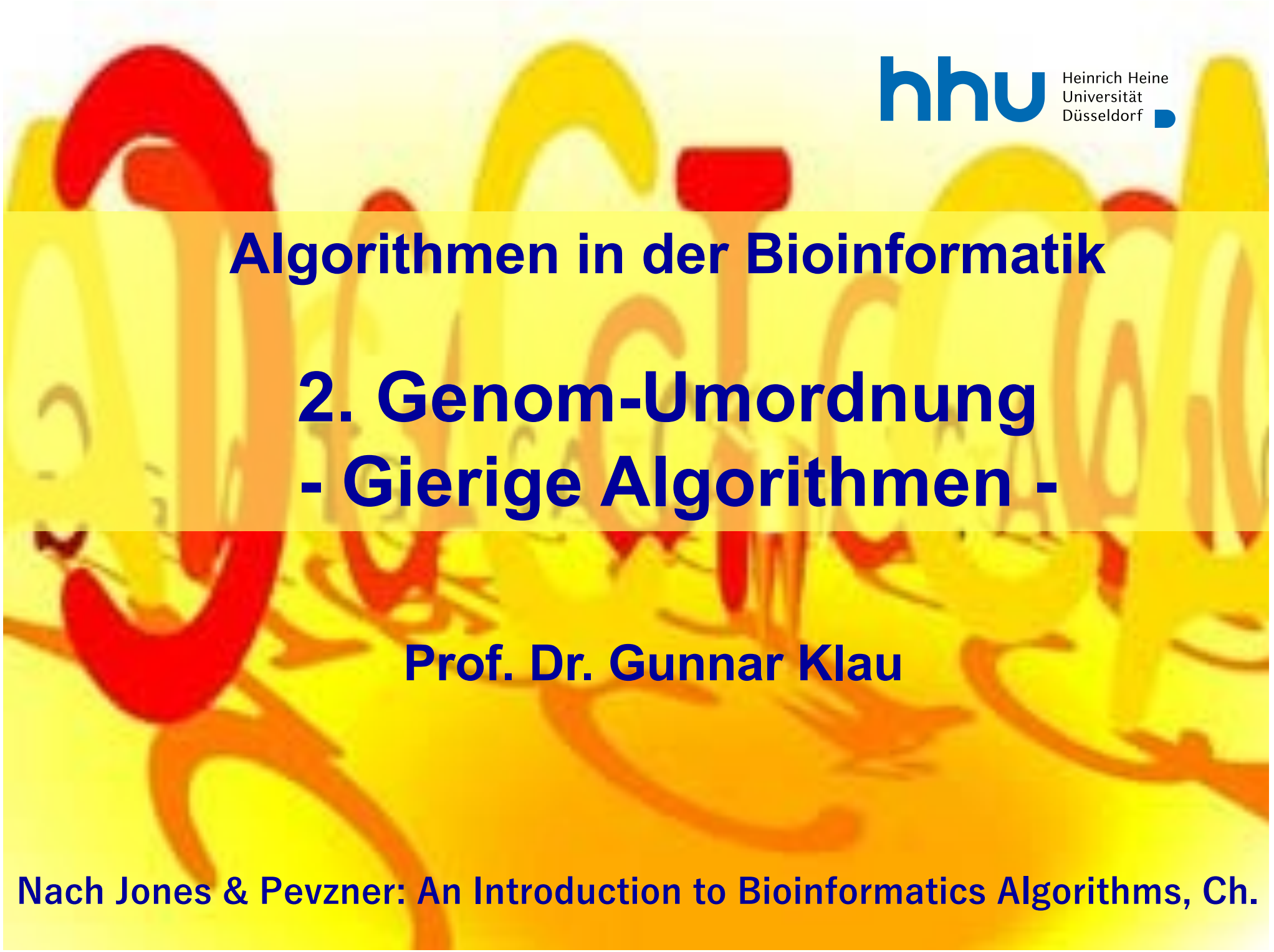
Let's Talk About

BEING GREEDY



Heinrich Heine Universität Düsseldorf

Algorithmen in der Bioinformatik

2. Genom-Umordnung - Gierige Algorithmen -

Prof. Dr. Gunnar Klau

Nach Jones & Pevzner: An Introduction to Bioinformatics Algorithms, Ch.

Hinweise zum Urheberrecht

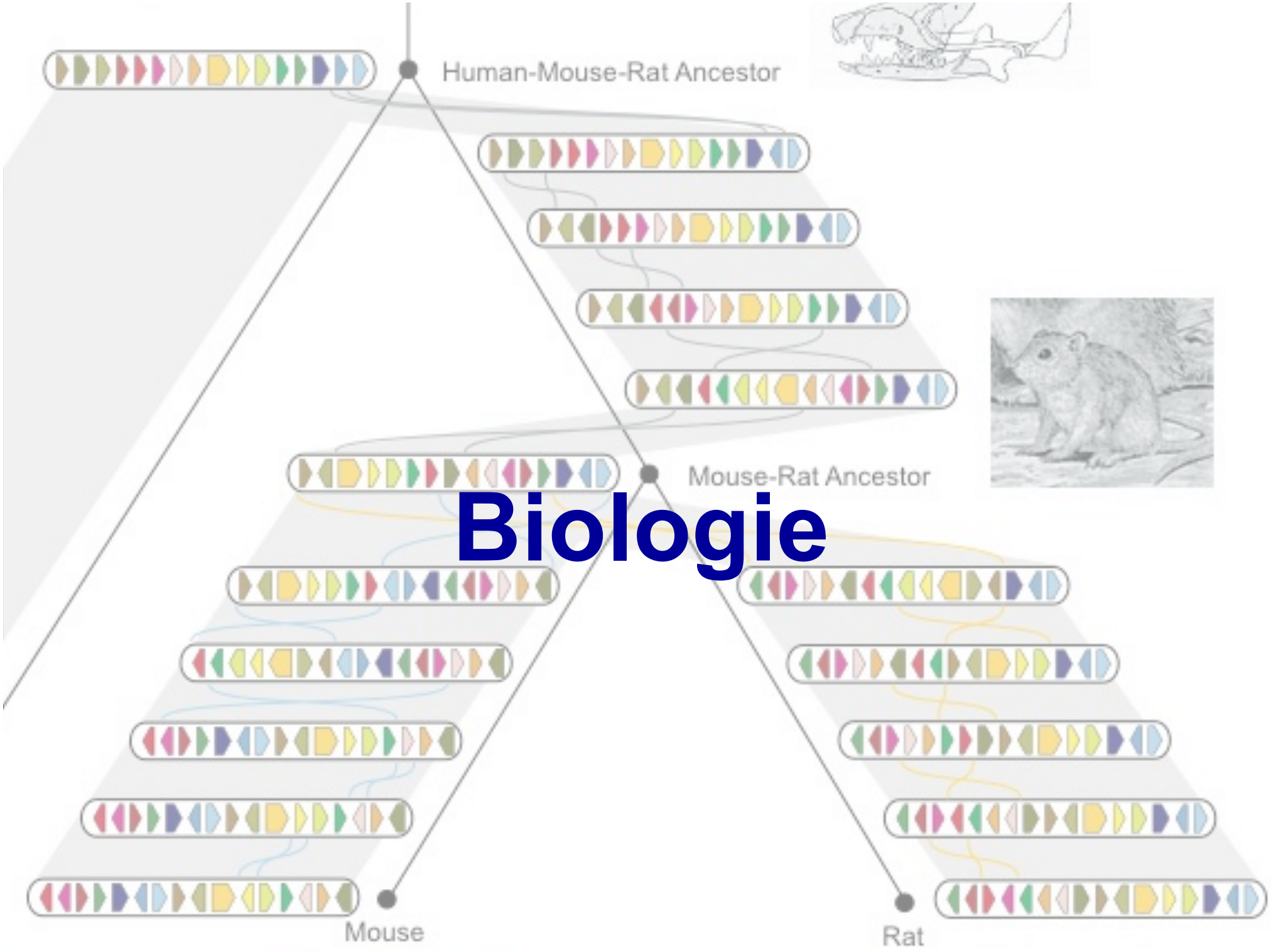
Bitte verzichten Sie in dieser Lehrveranstaltung
auf Bild- und Tonaufzeichnungen.

Weitere Informationen finden Sie im Studierendenportal

Hinweise zum Urheberrecht

Das Veröffentlichen oder Teilen von Bild- und Tonaufzeichnungen dieser Lehrveranstaltung ist nicht gestattet.

Weitere Informationen finden Sie im Studierendenportal



Biologie

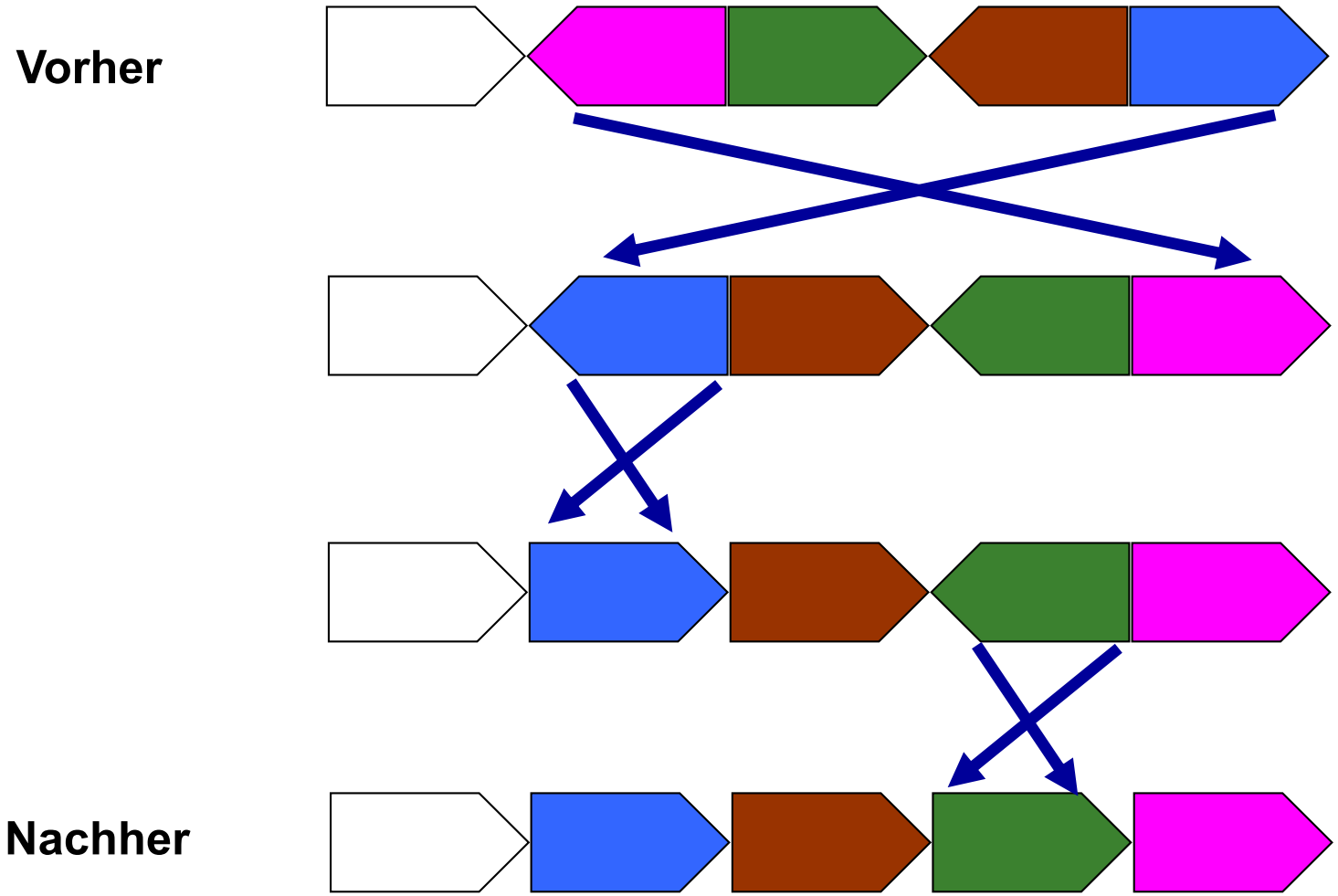
Weißkohl vs. Kohlrabi

Brassica oleracea var. capitata f. alba

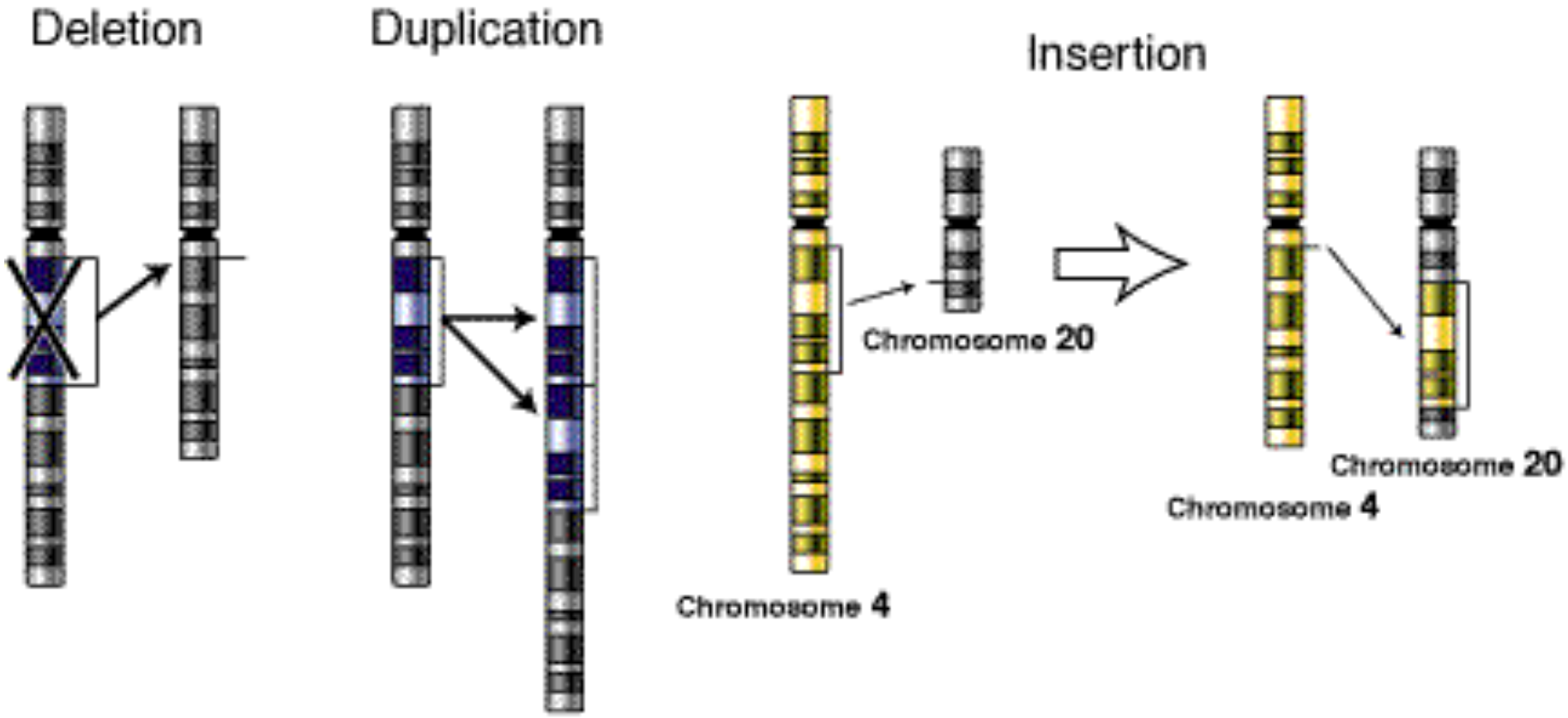
Brassica oleracea gongylodes

- Gemeinsamer Vorfahre
- Unterschiedliches Aussehen und Geschmack
- Mitochondriale DNA ist zu 99% gleich
- Unterschied: Gen-Anordnung!

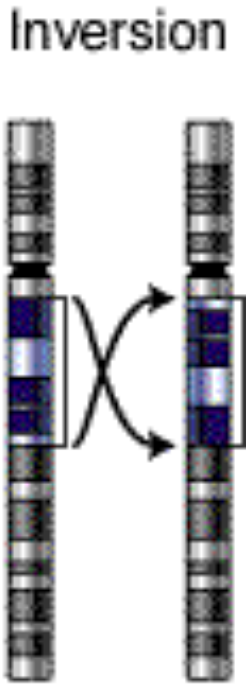
mtDNA: Gen-Ordnung

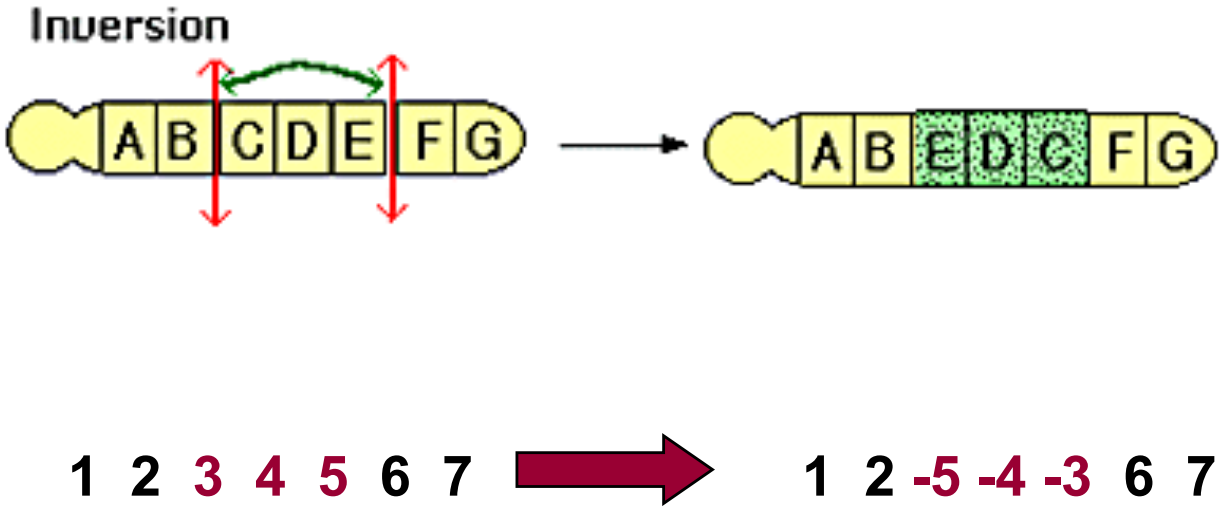


Genom-Umordnung: Mutationen



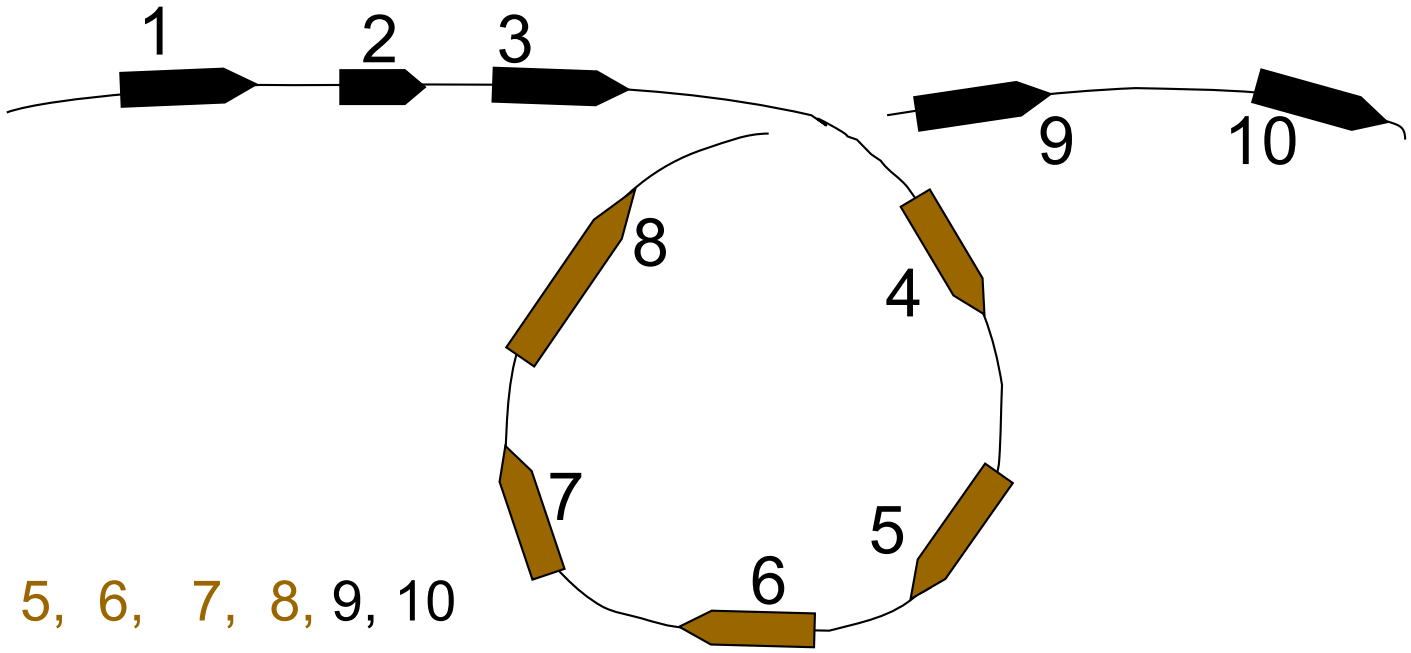
Genom-Umordnung: Mutationen





Inversionen (Umkehrungen) sind die häufigste Ordnungs-Mutation!

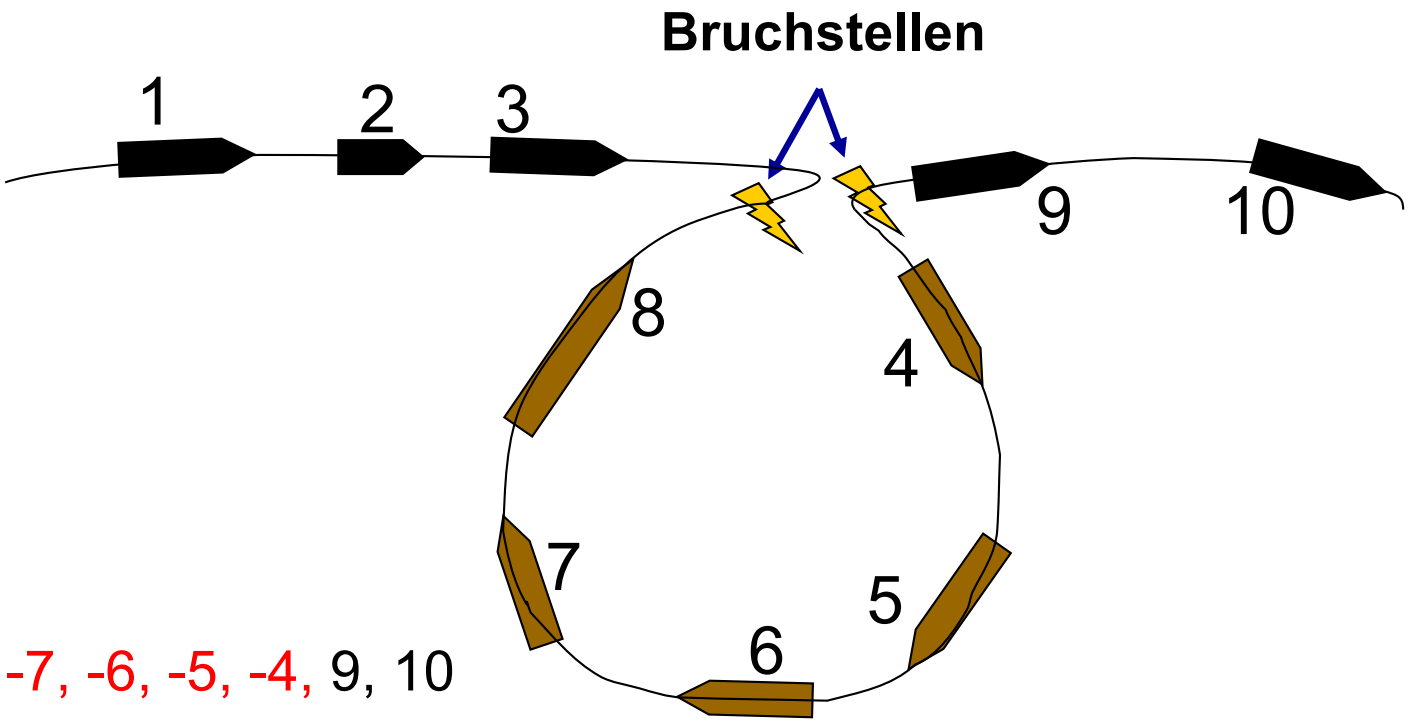
Umkehrungen sind falsch aufgelöste Knoten



1, 2, 3, 4, 5, 6, 7, 8, 9, 10

[Blöcke sind Gene]

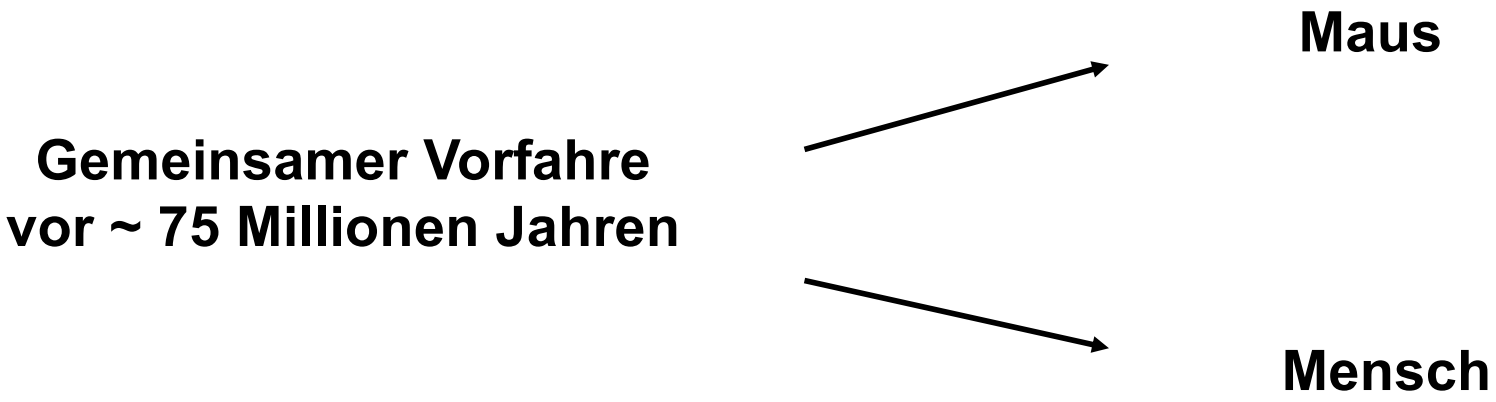
Umkehrungen sind falsch aufgelöste Knoten



1, 2, 3, -8, -7, -6, -5, -4, 9, 10

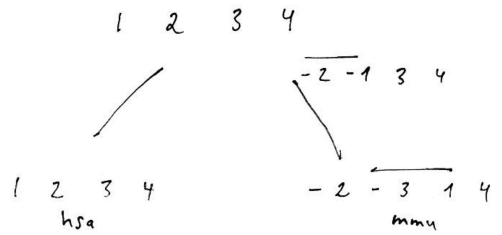
[Blöcke sind Gene]

Mensch-Maus: X-Chromosom

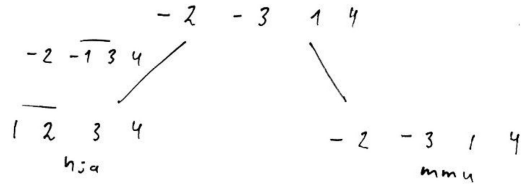


Fragen:

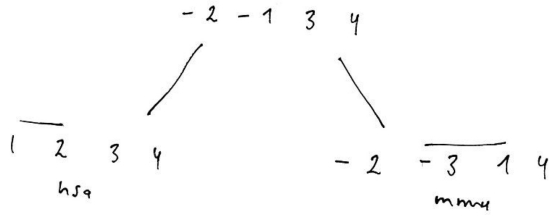
- Wie identifizieren wir Blöcke, die ähnlich (unverändert) geblieben sind?
  →schnelles Alignment (siehe später im Kurs)
- Wie sah das Genom des Vorfahrens aus?
  →nur durch Vergleich mit weiteren Genomen zu beantworten
- Welche Schritte haben zu den unterschiedlichen Anordnungen geführt?



2 op.



also 2 op

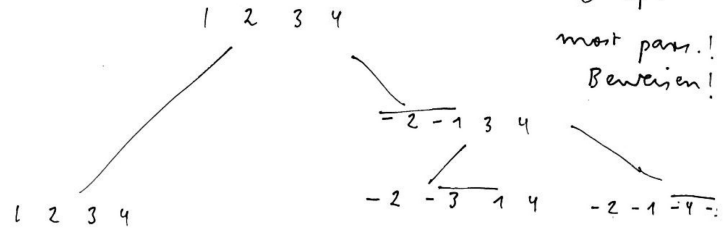


also. 2 op

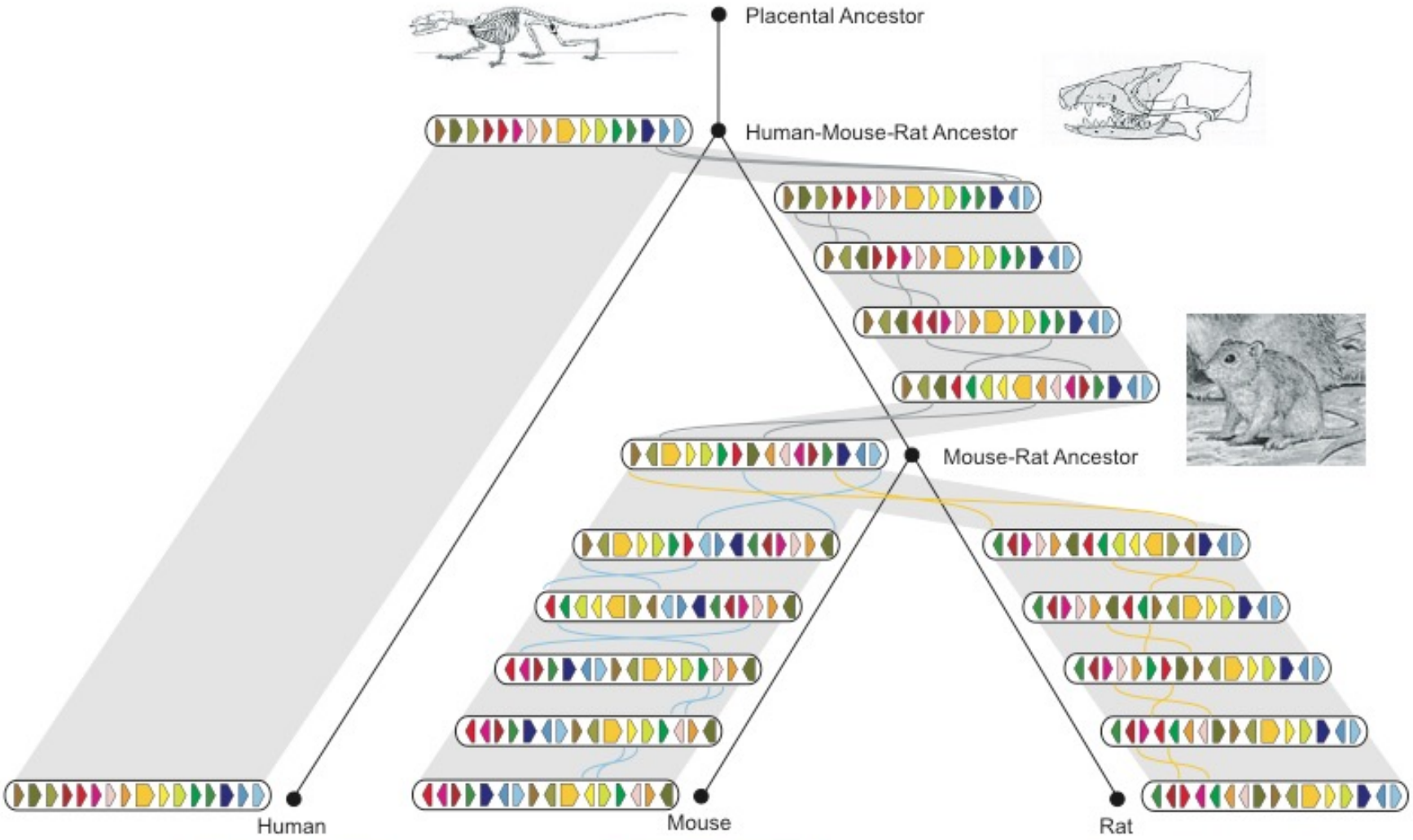
---

3 op.

most parsim.!
Beweisen!



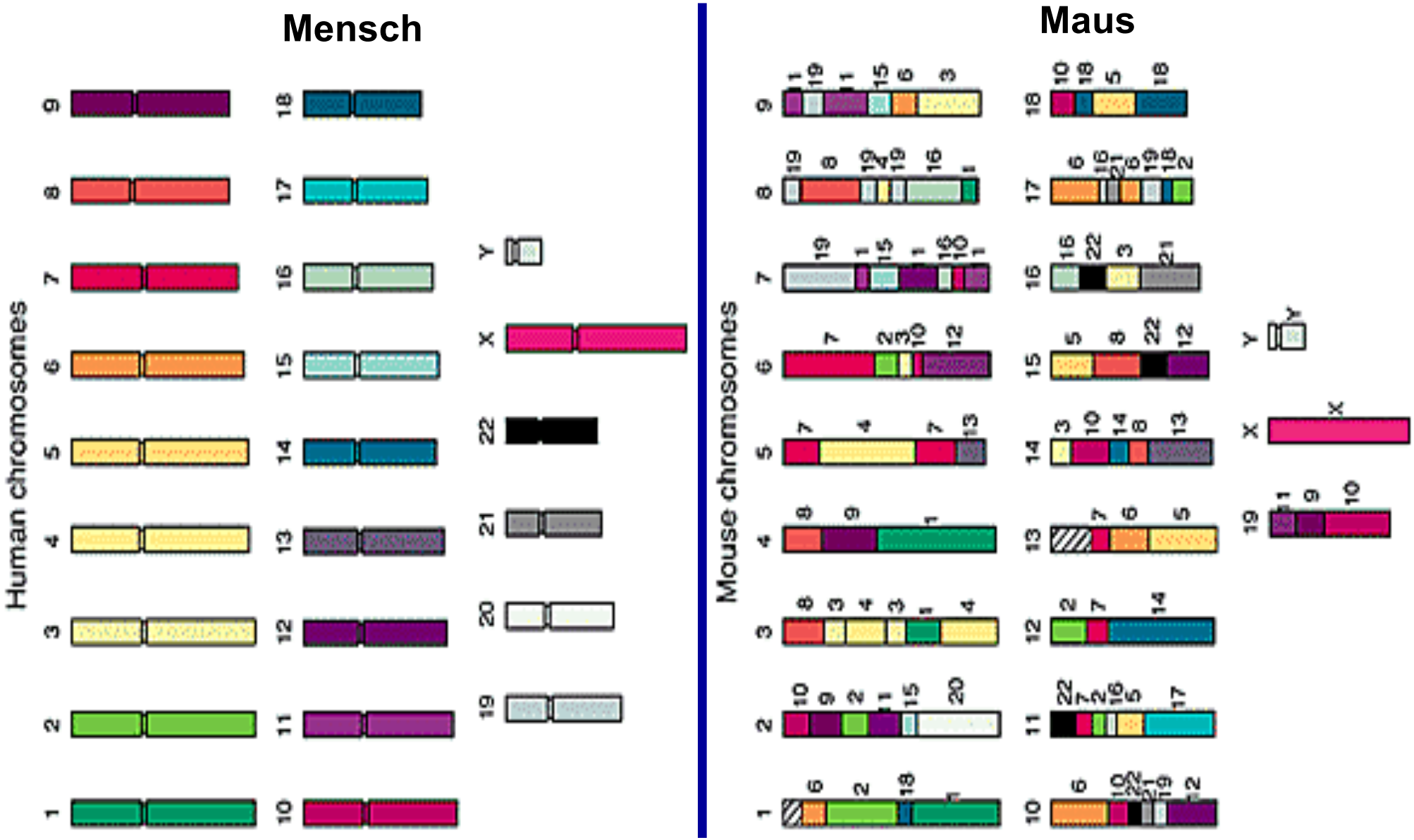
Die Geschichte des X-Chromsoms



Rat Consortium, *Nature*, 2004

Mensch-Maus: Genom-Vergleich

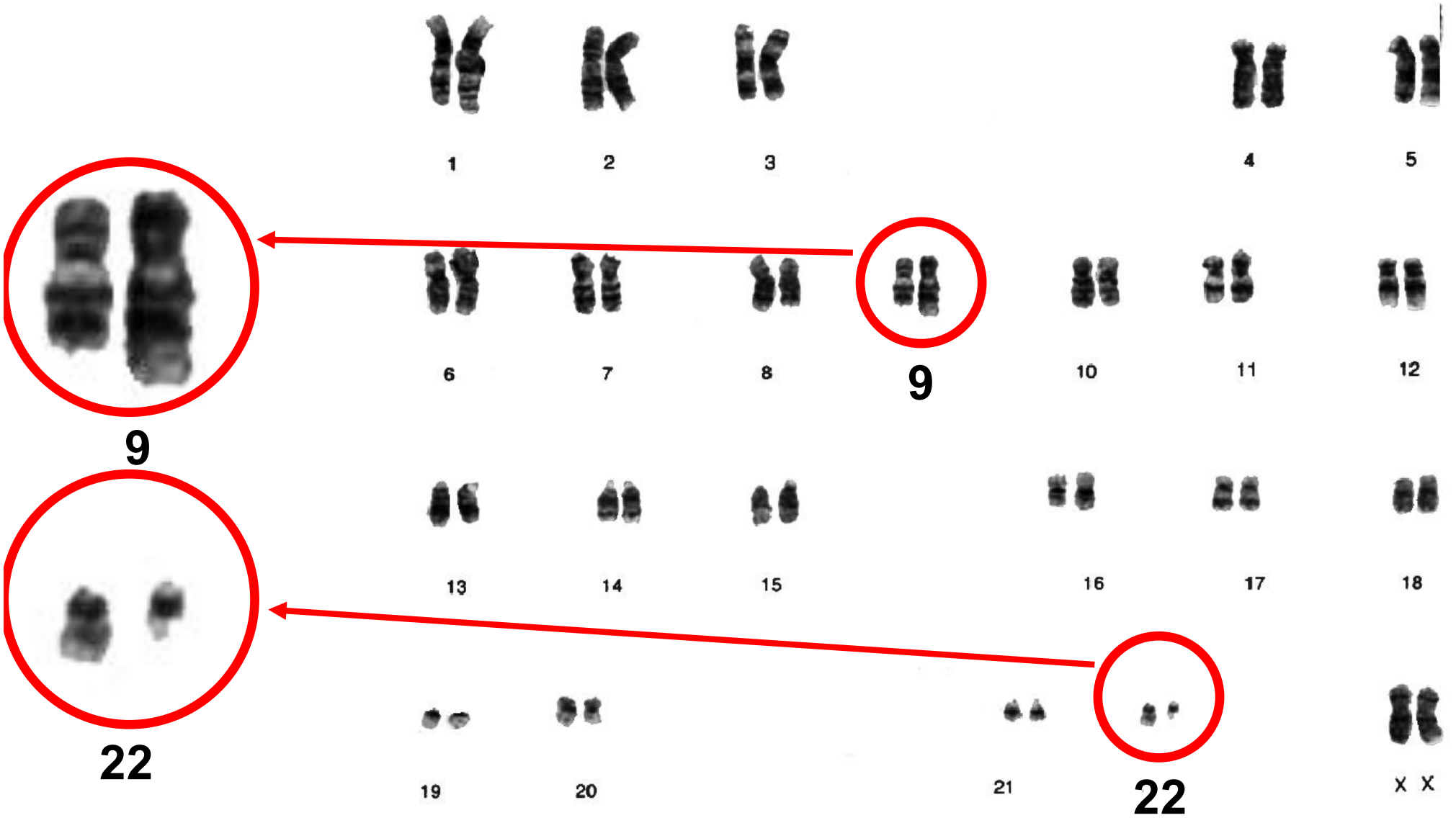
~245 Umordnungs-Mutationen:



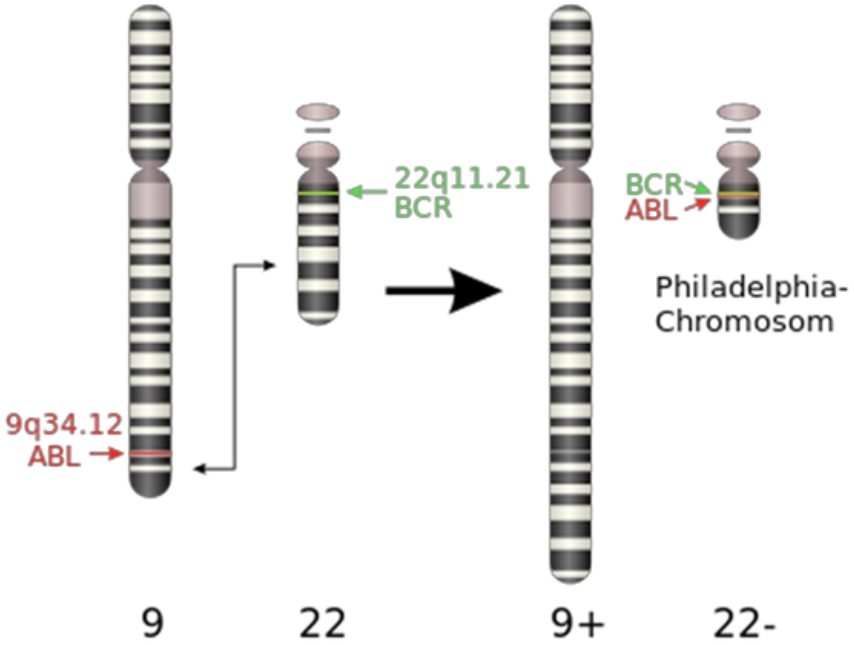
Wen interessiert das?

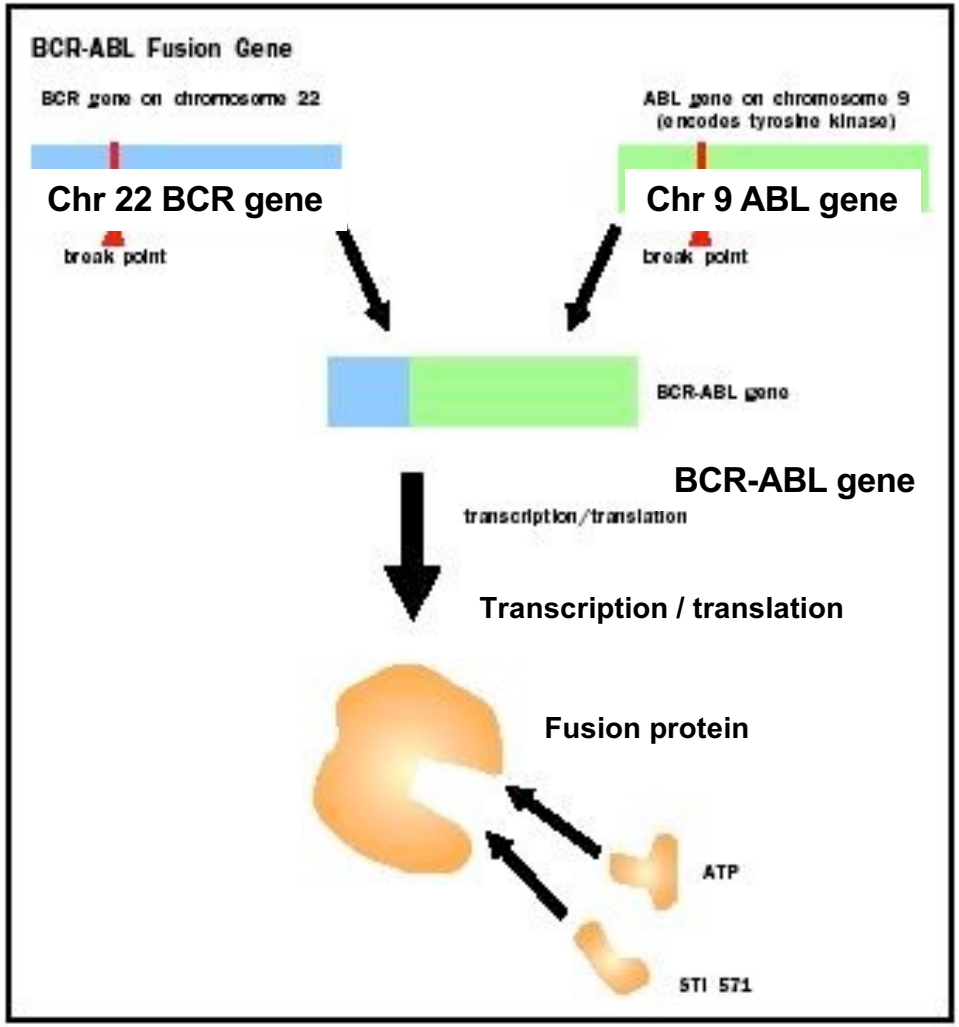
Philadelphia-Chromosom

- Translokation zwischen Chr. 22 → 9
- Haben 90% aller Patienten mit Chronic Myelogenous Leukemia (CML)



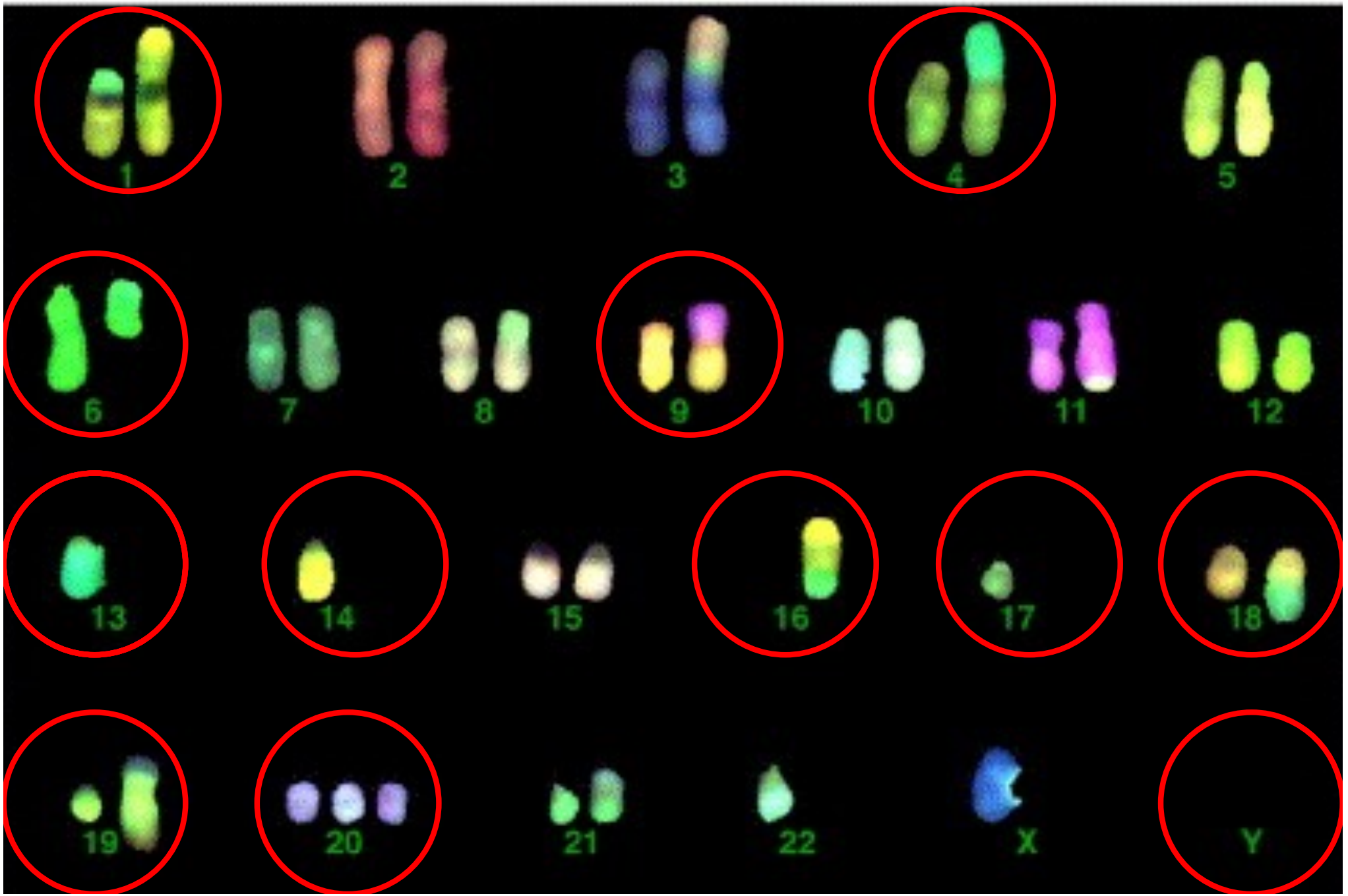
Fusion gene auf dem Philadelphia-Chromosom

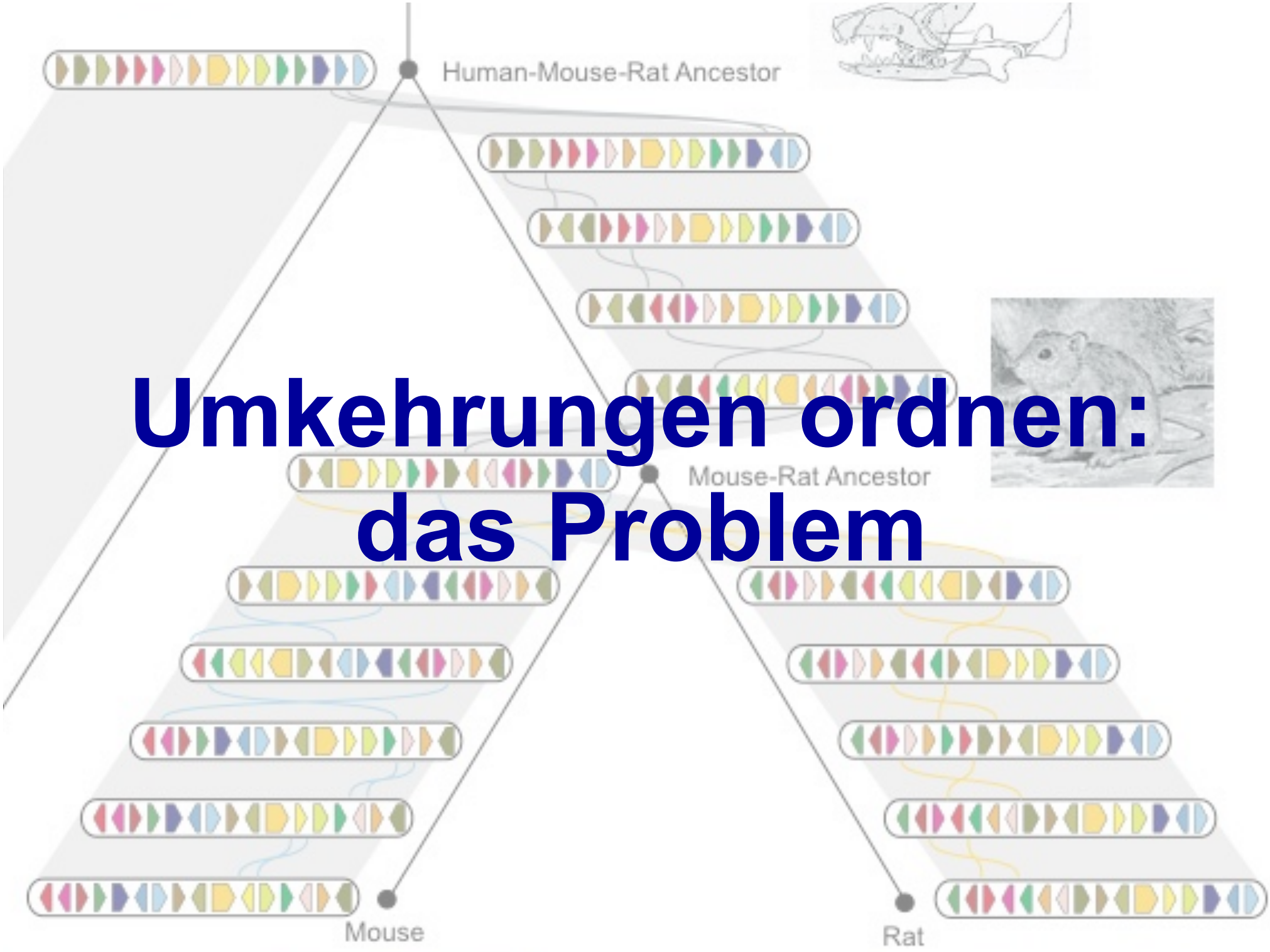




The fusion gene activates signal transduction pathways such as RAS/MAPK, PI-3 kinase, c-CBL and CRKL pathways, JAK-STAT, and the Src pathway.

Darmkrebs

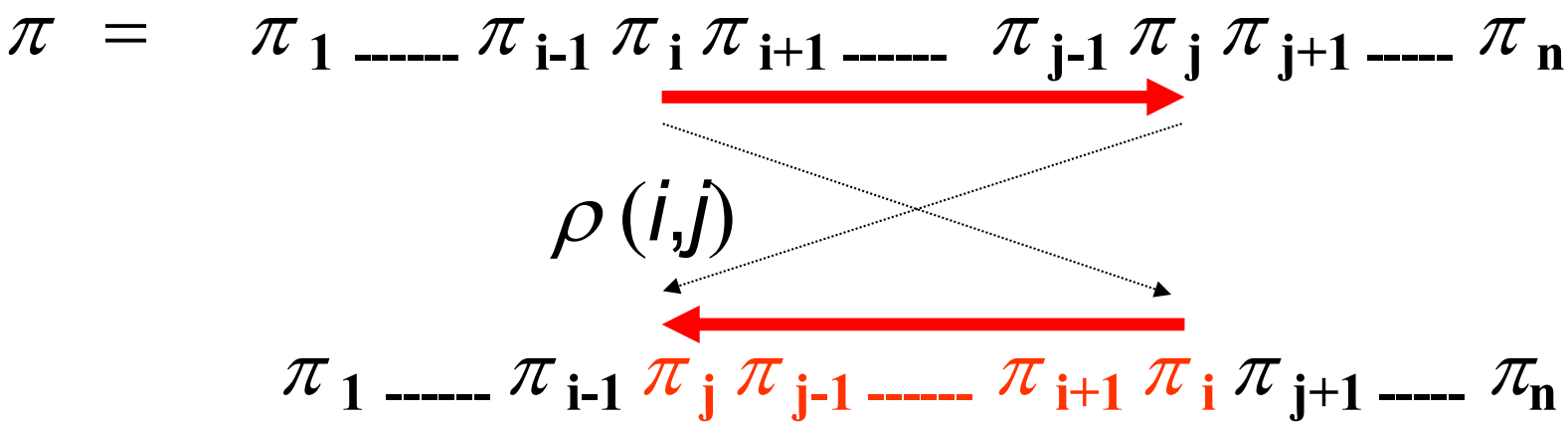




Umkehrungen ordnen: das Problem

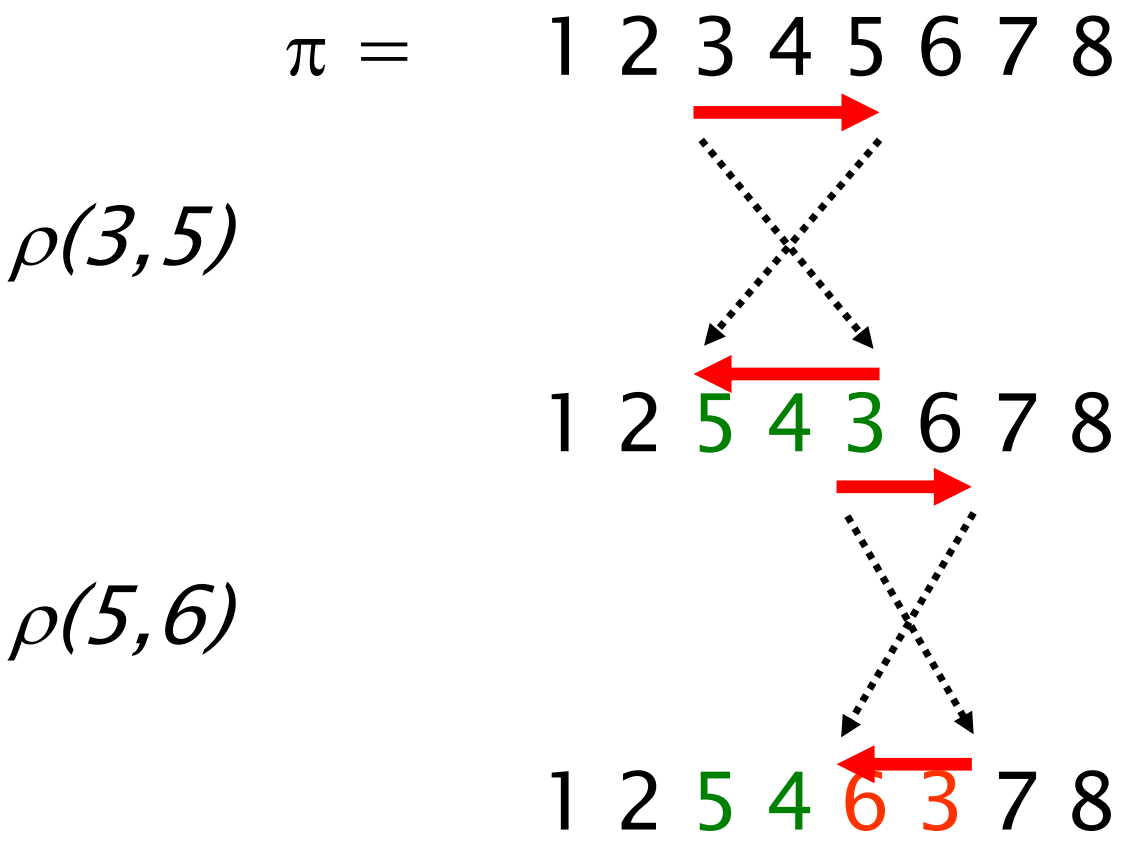
Umkehrungen und Genordnung

Die Genordnung wird durch eine Permutation π beschrieben:



Die Umkehrung ρ (*i, j*) verdreht ein ‘Bruchstück’ von *i* bis *j* in π.

Umkehrungen: Beispiel



Umkehrungsabstand: Das Problem

Ziel: Für zwei vorgegebene Permutationen: finde die kürzeste Abfolge von Umkehrungen, die eine Permutation in die andere überführt.

Eingabe: Permutationen π und σ

Ausgabe: Eine Abfolge von Umkehrungen $\rho_1,\ldots\rho_t$, die π in σ überführt, so dass t minimal ist (für alle möglichen Abfolgen $\rho_1,\ldots\rho_v$)

Definition:

$d(\pi, \sigma)$:= Umkehrungsabstand
:= kleinstmöglicher Wert für t für gegebene π und σ

Sortieren durch Umkehrungen: Das Problem

Ziel: Für **eine** vorgegebene Permutationen: finde die kürzeste Abfolge von Umkehrungen, die sie in die Identitäts-Permutation (1,..,*n*) überführt.

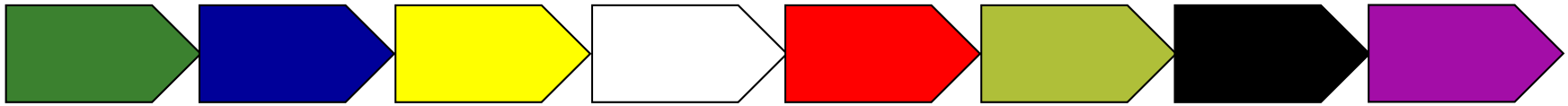
Eingabe: Permutation π

Ausgabe: Eine Abfolge von Umkehrungen $\rho_1,\ldots\rho_t$, die π in die Einheits-Permutation überführt, so dass *t* minimal ist (für alle möglichen Abfolgen $\rho_1,\ldots\rho_v$)

Vorzeichen?

π = (2, -4, -3, 5, -8, 7, -6, 1)

π = (2, 4, 3, 5, 8, 7, 6, 1)



Das Pfannkuchen-Problem

- Der Koch ist nachlässig:
  seine Pfannkuchen haben alle
  unterschiedliche Größen.
- Der Kellner will sie ordnen:
  der Größte unten, der Kleinste oben.
- Dazu nimmt er jeweils eine Gruppe
  von Pfannkuchen hoch und dreht sie um
  (*Flip*)

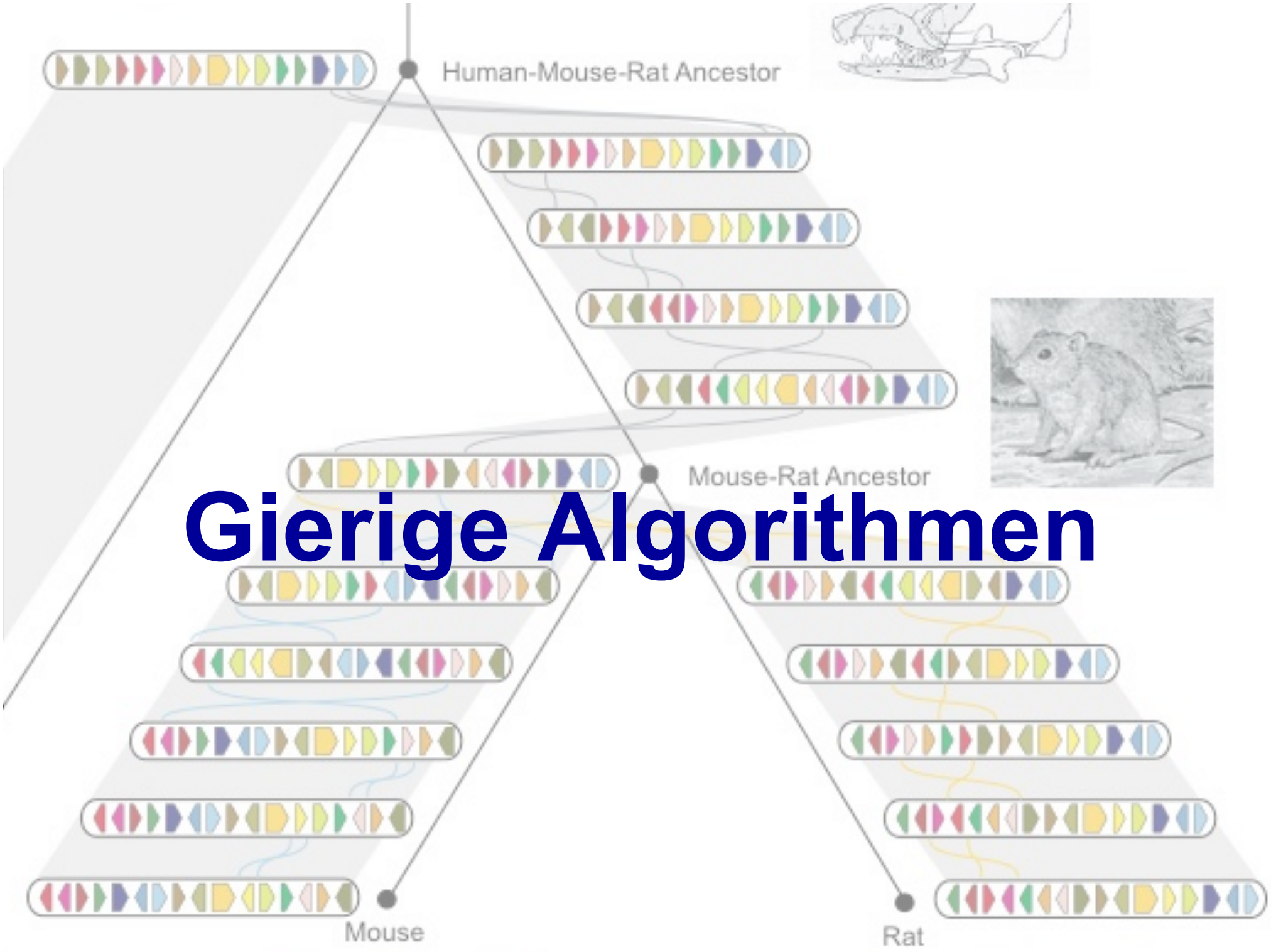
Gegeben ein beliebiger Stapel, löse das mit der minimalen Anzahl von *Flips*.

Das Pfannkuchen-Problem

Ziel: Für einen vorgegebenen Stapel von *n* Pfannkuchen: Was ist die minimale Anzahl von Flips, um den Stapel zu ordnen?

Eingabe: Permutation π

Ausgabe: Eine Abfolge von Präfix-Umkehrungen $\rho_1, \ldots \rho_t$, die π in die Identitäts-Permutation überführen, so dass *t* minimal ist (für alle möglichen $\rho_1, \ldots \rho_v$).



Gierige Algorithmen

Pfannkuchen-Problem: Gieriger Algorithmus

Idee: Sortiere den jeweils größten unsortierten Pfannkuchen ein: zuerst nach oben (1. Flip), dann an seinen Platz (2. Flip)

$\Rightarrow$ max. $2(n-1)$ Flips insgesamt (der letzte Pfannkuchen stimmt automatisch)

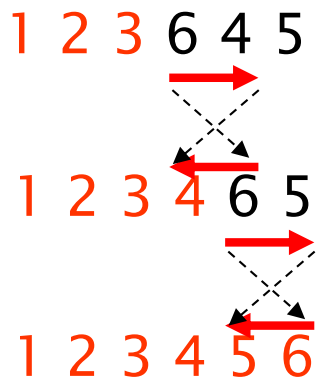
Christos Papadimitriou & William Gates: höchstens 5/3 (n + 1) Flips

Sortieren durch Umkehrungen: Gieriger Ansatz

In π = 1 2 3 6 4 5 sind die ersten drei Elemente sortiert
$\rightarrow$ lassen wir so

Die Länge des bereits sortierten Präfixes nennen wir *prefix*(π) (=3)

Idee: erhöhe *prefix*(π) bei jedem Schritt



$\Rightarrow$ Anzahl von Umkehrungen ist max. (n – 1)

Gieriger Algorithmus: Pseudocode

SimpleReversalSort(π)

1 **for** $i \leftarrow 1$ to $n - 1$

2 $j \leftarrow$ Position des i-ten Elements in π (i.e., $\pi_j = i$)

3 **if** $j \neq i$

4 $\pi \leftarrow \pi * \rho(i, j)$

5 **output** π

6 **if** (π ist Identiät) **return**

Liefert SimpleReversalSort die optimale Lösung?

$$\pi = (6, 1, 2, 3, 4, 5)$$

Approximationsalgorithmen (Minimierung)

Sei OPT(π) ≠ 0 der optimale Lösungswert bei Eingabe π, zum Beispiel die optimale Anzahl Reversals beim Sortieren durch Umkehren oder die minimale Distanz (TotalDistance) beim Median-String-Problem.

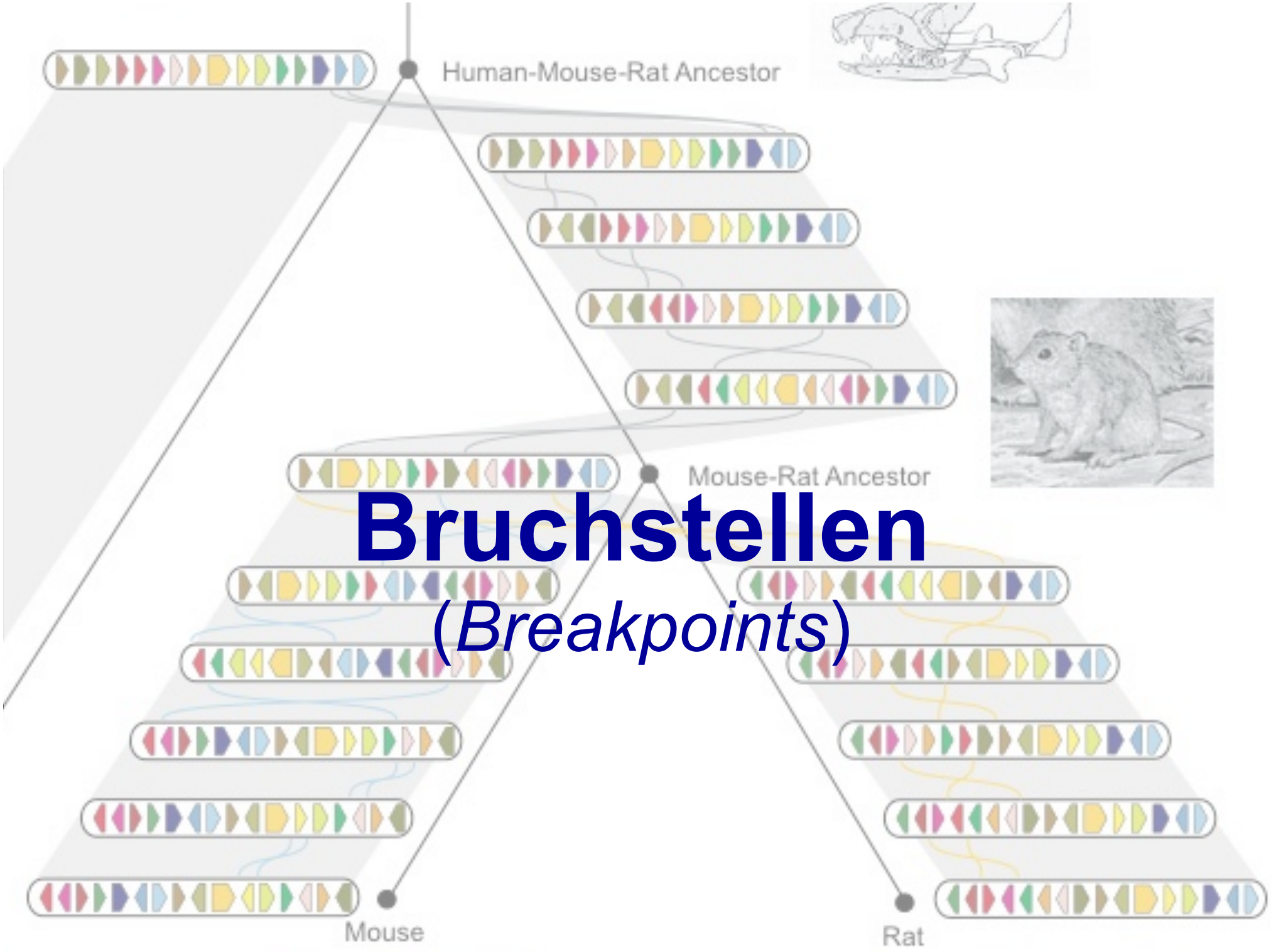
Dann ist der **Approximationsfaktor** eines Algorithmus A **auf Eingabe π**:

$$A(\pi) / OPT(\pi)$$

Approximationsfaktor eines Algorithmus A:
schlechtester Approximationsfaktor über **alle** Eingaben. Bei Minimierungsproblemen:

$$\alpha = \max_{\pi} A(\pi) / OPT(\pi)$$

Bemerkung: bei Maximierungsproblemen gilt $\alpha = \min_{\pi} A(\pi) / OPT(\pi)$



Bruchstellen

(*Breakpoints*)

Nachbarschaften & Bruchstellen

Permutation $\pi = \pi_1\pi_2\pi_3\ldots\pi_{n-1}\pi_n$

Definitionen:

- Zwei Elemente π_i und π_{i+1} sind **benachbart**
  $:\Leftrightarrow \pi_{i+1} = \pi_i \pm 1$

- Zwischen zwei Elementen π_i und π_{i+1} befindet sich eine **Bruchstelle**
  $:\Leftrightarrow \pi_i$ und π_{i+1} sind <u>nicht</u> benachbart

- $b(\pi)$:= # Bruchstellen in π

Beispiel: π = 1 9 3 4 7 8 2 6 5

Erweiterte Permutationen

Damit die Bruchstellen vollständige Auskunft über die Sortierung geben, fügen wir zwei Elemente $\pi_0 = 0$ und $\pi_{n+1} = n+1$ an.

Beispiel:

$$\pi = 1 \mid 9 \mid 3\ \ 4 \mid 7\ \ 8 \mid 2 \mid 6\ \ 5$$

$$\pi = 0\ 1 \mid 9 \mid 3\ \ 4 \mid 7\ \ 8 \mid 2 \mid 6\ \ 5 \mid 10$$

Umkehrungsabstand & Bruchstellen

Jede Umkehrung entfernt maximal zwei Bruchstellen

Beispiel:

π = 0 2 3 1 4 6 5 7

$\Rightarrow$ Umkehrungsabstand ≥ # Bruchstellen / 2

Besser Sortieren durch Bruchstellen?

BreakPointReversalSort?(π)

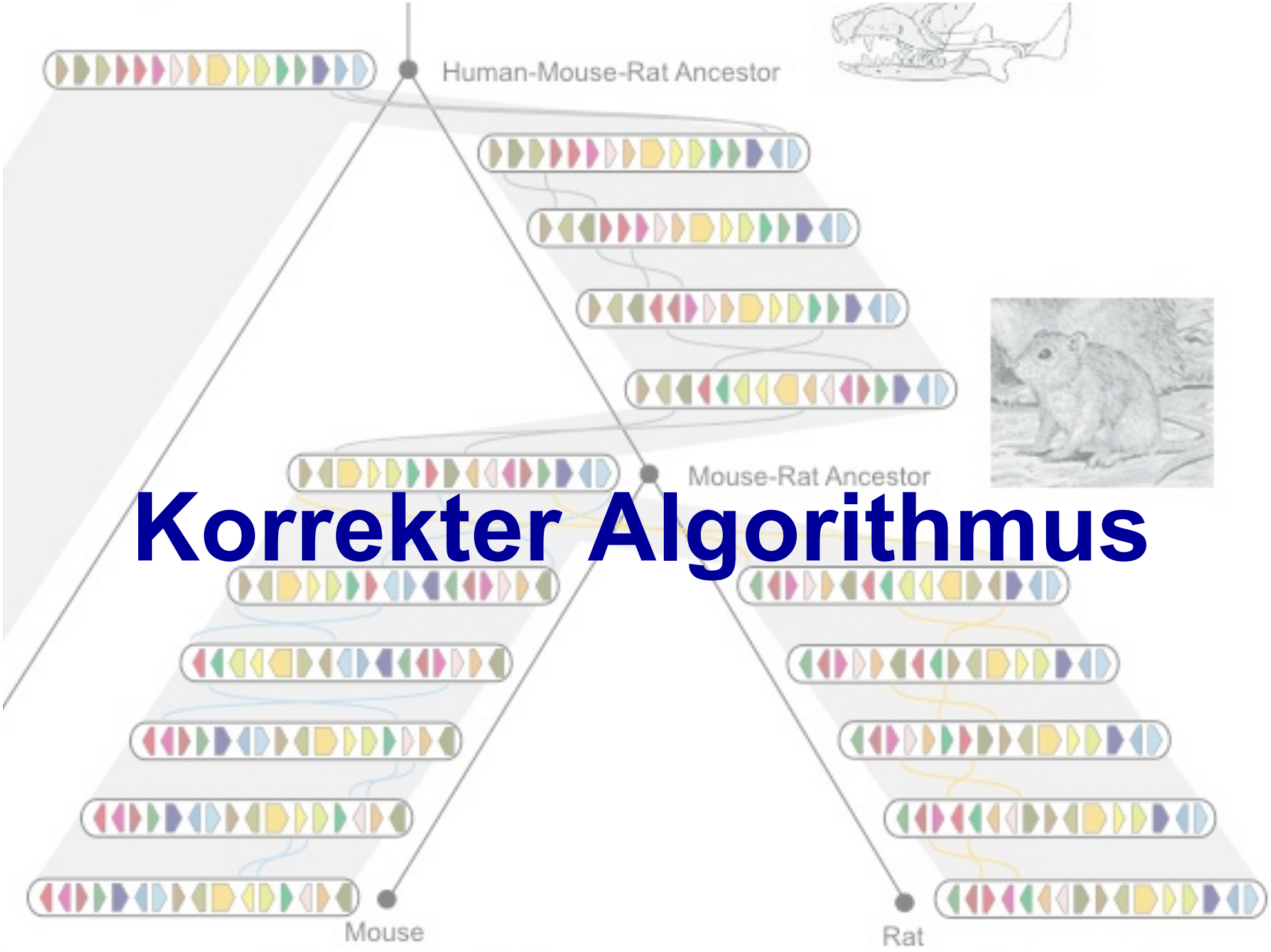
1 **while** $b(\pi) > 0$

2 Finde die Umkehrung $\rho(i,j)$, die $b(\pi \bullet \rho(i,j))$ minimiert

3 $\pi \leftarrow \pi \bullet \rho(i, j)$

4 **output** π

5 **return**



Korrekter Algorithmus

Strips

Definitionen:

- Streifen: ein Intervall zwischen zwei aufeinanderfolgenden Bruchstellen
- Abnehmender Streifen: Elemente in abnehmender Folge
  (z. B. 6 5 oder 4 3 2)
- Zunehmender Streifen: Elemente in zunehmender Folge
  (z. B. 5 6 oder 2 3 4)
- 1-elementige Streifen können zu- oder abnehmend sein.
  Wir definieren sie als abnehmend (mit Ausnahme von 0 und $n + 1$)

Beispiel:

Reduzierung der Bruchstellen

Theorem:

Wenn die Permutation π wenigstens einen abnehmenden Streifen hat, dann existiert eine Umkehrung ρ, die die Anzahl der Breakpoints vermindert (d.h.: $\exists \rho b(\pi \bullet \rho) < b(\pi)$).

Beweis:

Wähle denjenigen abnehmenden Streifen in π mit dem kleinsten Element *k*. Dann ist *k* das letzte Element in diesem Streifen. Dann muss *k*-1 das letzte Element in einem zunehmenden Streifen sein. Umkehrung des Intervalls zwischen *k*-1 und *k* einschließlich *k* (wenn *k*-1 links von *k*; sonst zwischen *k* und *k*-1 einschließlich *k*-1) bringt die beiden zusammen und beseitigt damit mind. eine Bruchstelle.$_{\text{QED}}$

Was, wenn alle Streifen zunehmend sind?

Dann drehen wir einfach einen um!

Korrektes Sortieren durch Umkehrungen!

CorrectBreakpointReversalSort(π)

```
1    while b(π) > 0
2        if π hat einen abnehmenden Streifen
3            Wähle Umkehrung ρ, die b(π • ρ) minimiert
4        else
5            Wähle Umkehrung ρ für zunehmenden Streifen in π
6        π ← π • ρ
7        output π
8    return
```

Performance-Garantie

$$= \frac{\max(\text{Schritte in Approximation})}{\min(\text{Schritte in optimaler Lösung})}$$

Approximation durch *CorrectBreakpointReversalSort*():

Mindestens jeder 2. Schritt eliminiert mindestens eine Bruchstelle

$\Rightarrow$ max. 2 $b(\pi)$ Schritte

Bestmögliche Lösung:

Jeder Schritt eliminiert höchstens 2 Bruchstellen

$\Rightarrow$ min. $b(\pi)/2$ Schritte

- Performance-Garantie: $A = \frac{2b(\pi)}{\frac{b(\pi)}{2}} = 4$

Gierige Motif-Suche

Take home messages

- Verständnis von Genomumordnungen ist wichtig
- Eine Version (nicht die relevanteste)
  - Sortieren von Permutationen ohne Vorzeichen
- Approximationsalgorithmen
- Gierige Algorithmen
- Sortieren über Bruchstellen ergibt Faktor 4, viel besser als SimpleReversalSort

Das Motivsucheproblem

Ziel: Finde genau ein ℓ–mer für jede DNA-Sequenz einer vorgebenen Menge, so dass der Konsensusscore über die ℓ–mere maximal ist.

Eingabe: Eine ***t*** x ***n*** Matrix ***DNA*** sowie ℓ, (die Länge des gesuchten Motivs)

Ausgabe: Ein Vektor mit ***t*** Startpositionen **s** = (s_1, s_2, … s_t), so dass *Score*(**s**, *t*, ***DNA***) maximal ist.

Brachialmethode: $O(\ell\, t\, n^t)$, etwas eleganter $O(\ell\, t\, n4^\ell)$: [letzte Vorlesung]

- zu langsam für realistische Probleme

Gierige Motivsuche: CONSENSUS

Idee: Suche die beste Lösung für Sequenz *i* unter der Annahme, dass die beste Lösung für 1...*i*-1 schon gefunden ist!

Starte mit 2 der *t* Sequenzen

- Finde die 2 ähnlichsten *l*-mere darin, d.h.:
  finde **s**=(s_1, s_2) mit maximalem Score(**s**, 2, DNA)
- Iterativ für Sequenz *i* = 3, ..., *t:*
  Finde das dazu ähnlichste *l*-mer in Sequenz *i,* d.h.:
  finde s_i mit maximalem Score(**s**, *i*, DNA) für **s**=(s_1,…,s_{i-1},s_i,…)

Score berechnet für 2 Sequenzen

Score berechnet für *i* Sequenzen

CONSENSUS: Permutiere Reihenfolge **1000** mal und merke bestes Motiv.

Resultat:

- Nicht exakt (evtl. gibt es bessere Motive)
- Schlechter Approximationsfaktor (Übung)
- **Sehr schnell (die meiste Zeit für die 2 ersten *l*-mere)**
- **Ergebnis häufig sehr gut**

GreedyMotifSearch()

GreedyMotifSearch(*DNA, t, n, l*)

```
1.    bestMotif ← (1, 1, …, 1)
2.    s ← (1, 1, …, 1)
3.    for s1 ← 1 to n – l + 1
4.        for s2 ← 1 to n – l + 1
5.            if Score(s, 2, DNA) > Score(bestMotif, 2, DNA)
6.                bestMotif1 ← s1
7.                bestMotif2 ← s2
8.    s1 ← bestMotif1
9.    s2 ← bestMotif2
10.   for i ← 3 to t
11.       for si ← 2 to n – l + 
12.           if Score(s, i, DNA) > Score(bestMotif, i, DNA)
13.               bestMotifi ← si
14.       si ← bestMotifi
15.   return bestMotif
```

Bisher bester Startvektor

Aktueller Startvektor

Score berechnet für *t'*=2

Score berechnet für *t'*=*i*

Programme zur Motivsuche

- **CONSENSUS**
  Hertz, Stormo (1989-1999)
- **GibbsDNA**
  Lawrence et al (1993)
- **MITRA**
  Eskin, Pevzner (2002)
- **Pattern Branching**
  Price, Pevzner (2003)
- **Grisotto**
  Carvallo, Olivera (2011)
- **MEME**
  *Bailey et al. (*1995-2011)
- **…**

Take home messages

- Verständnis von Genomumordnungen ist wichtig
- Eine Version (nicht die relevanteste)
  - Sortieren von Permutationen ohne Vorzeichen
- Approximationsalgorithmen
- Gierige Algorithmen
- Sortieren über Bruchstellen ergibt Faktor 4, viel besser als SimpleReversalSort

- Motivsuche geht viel schneller (aber nicht mehr exakt) mit gierigen Algorithmen.