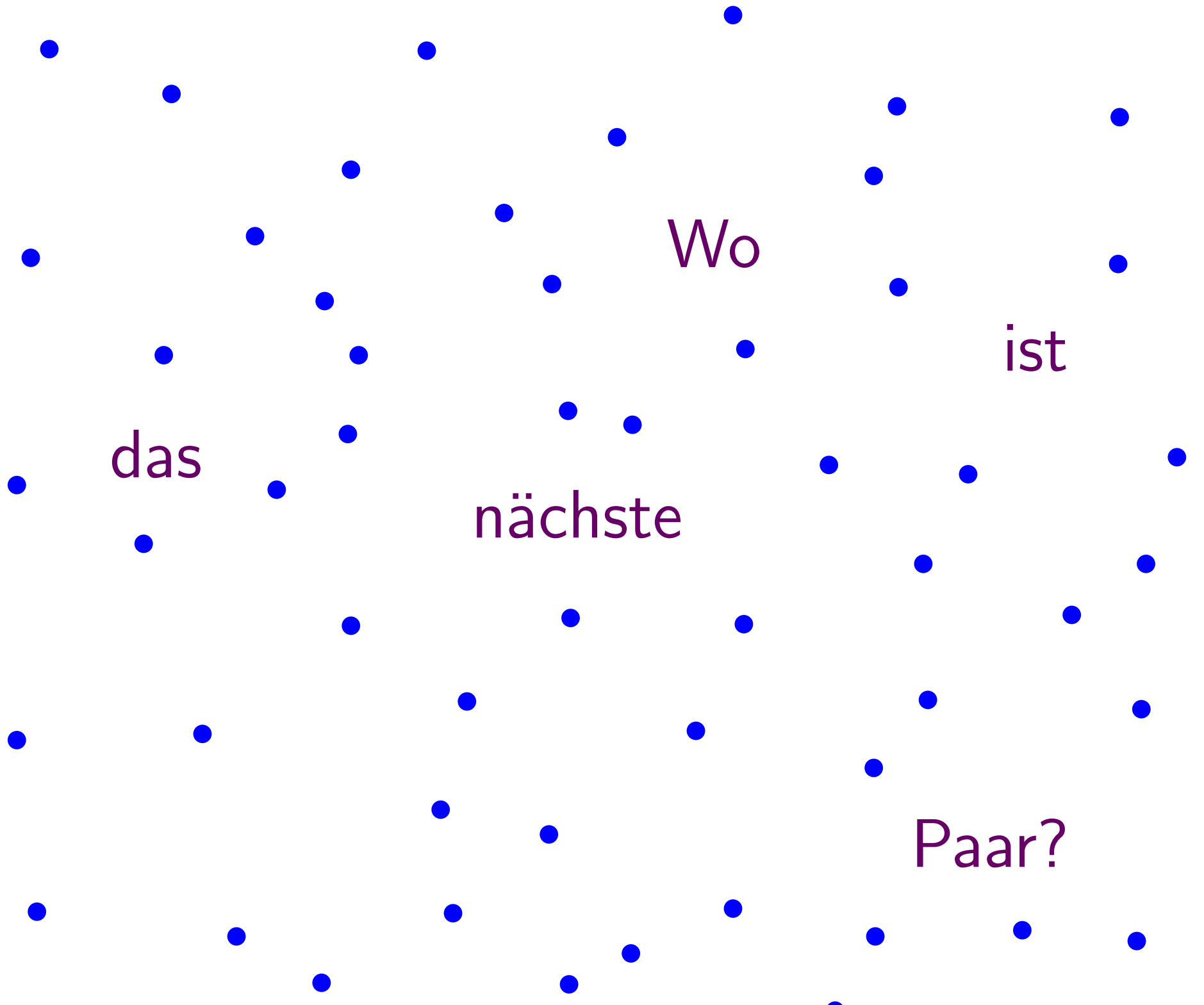


# Algorithmen und Datenstrukturen

Wintersemester 2022/23

16. Vorlesung

Nächstes Paar



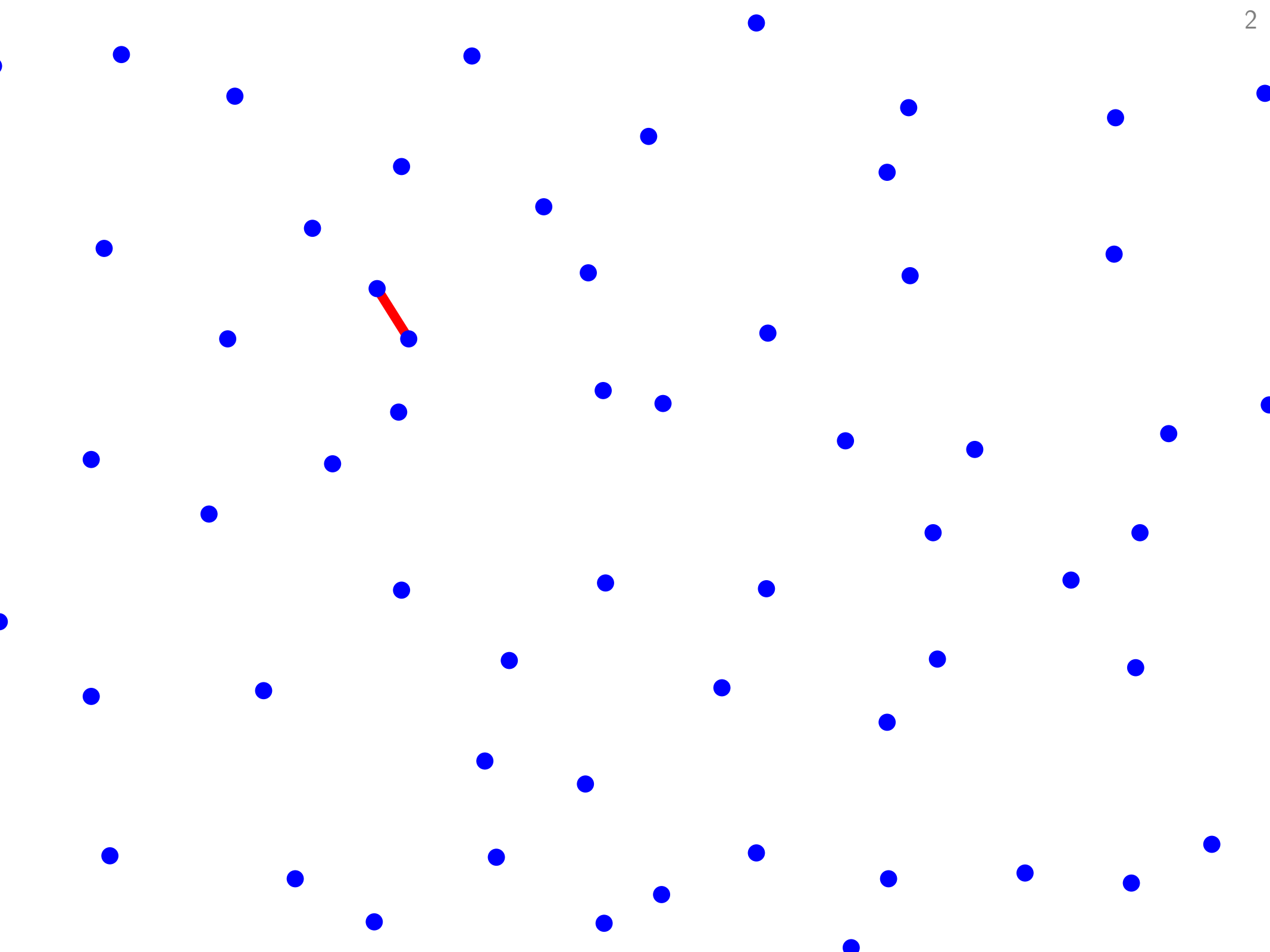
Wo

ist

das

nächste

Paar?



**Problem:**

Gegeben: Menge  $P$  von  $n$  Punkten in der Ebene,  
jeder Punkt  $p \in P$  als  $(x_p, y_p)$ .

**Problem:**

Gegeben: Menge  $P$  von  $n$  Punkten in der Ebene, jeder Punkt  $p \in P$  als  $(x_p, y_p)$ .

Finde: Punktepaar  $\{p, q\} \subseteq P$  mit kleinstem (euklidischen) Abstand.

**Problem:**

Gegeben: Menge  $P$  von  $n$  Punkten in der Ebene, jeder Punkt  $p \in P$  als  $(x_p, y_p)$ .

Finde: Punktepaar  $\{p, q\} \subseteq P$  mit kleinstem (euklidischen) Abstand.

**Def.**

Euklidischer Abstand von  $p$  und  $q$  ist

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}.$$

**Problem:**

Gegeben: Menge  $P$  von  $n$  Punkten in der Ebene, jeder Punkt  $p \in P$  als  $(x_p, y_p)$ .

Finde: Punktepaar  $\{p, q\} \subseteq P$  mit kleinstem (euklidischen) Abstand.

**Def.** Euklidischer Abstand von  $p$  und  $q$  ist

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}.$$

**Lösung:**

## Problem:

Gegeben: Menge  $P$  von  $n$  Punkten in der Ebene, jeder Punkt  $p \in P$  als  $(x_p, y_p)$ .

Finde: Punktepaar  $\{p, q\} \subseteq P$  mit kleinstem (euklidischen) Abstand.

## Def.

Euklidischer Abstand von  $p$  und  $q$  ist

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}.$$

## Lösung:

- Gehe durch alle  $\binom{n}{2}$  Punktepaare und berechne ihren Abstand.
- Gib ein Paar mit kleinstem Abstand zurück.



## Problem:

Gegeben: Menge  $P$  von  $n$  Punkten in der Ebene, jeder Punkt  $p \in P$  als  $(x_p, y_p)$ .

Finde: Punktepaar  $\{p, q\} \subseteq P$  mit kleinstem (euklidischen) Abstand.

## Def.

Euklidischer Abstand von  $p$  und  $q$  ist

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}.$$

## Lösung:

## Laufzeit:

- Gehe durch alle  $\binom{n}{2}$  Punktepaare und berechne ihren Abstand.
- Gib ein Paar mit kleinstem Abstand zurück.

## Problem:

Gegeben: Menge  $P$  von  $n$  Punkten in der Ebene, jeder Punkt  $p \in P$  als  $(x_p, y_p)$ .

Finde: Punktepaar  $\{p, q\} \subseteq P$  mit kleinstem (euklidischen) Abstand.

## Def.

Euklidischer Abstand von  $p$  und  $q$  ist

$$d(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}.$$

## Lösung:

**Laufzeit:**  $\Theta(n^2)$

- Gehe durch alle  $\binom{n}{2}$  Punktepaare und berechne ihren Abstand.
- Gib ein Paar mit kleinstem Abstand zurück.


# Mach's besser!

- Entwurfsparadigma:**
- inkrementell?
  - randomisiert?
  - Teile und Herrsche?



# Mach's besser!

**Entwurfsparadigma:** – inkrementell?  
– randomisiert?  
– Teile und Herrsche?


**Spezialfall:** 

**Lösung:**

# Mach's besser!

**Entwurfsparadigma:**

- inkrementell?
- randomisiert?
- Teile und Herrsche?

**Spezialfall:** 


**Lösung:**

- Sortiere (nach x-Koordinate).

# Mach's besser!

**Entwurfsparadigma:**

- inkrementell?
- randomisiert?
- Teile und Herrsche?


**Spezialfall:** 

**Lösung:**

- Sortiere (nach x-Koordinate).
- Berechne Abstände aller *aufeinanderfolgender* Punktepaare.

# Mach's besser!

**Entwurfsparadigma:** – inkrementell?  
– randomisiert?  
– Teile und Herrsche?

**Spezialfall:** 

**Lösung:**


- Sortiere (nach x-Koordinate).
- Berechne Abstände aller *aufeinanderfolgender* Punktepaare.
- Bestimme das Minimum dieser Abstände.



# Mach's besser!

**Entwurfsparadigma:**

- inkrementell?
- randomisiert?
- Teile und Herrsche?

**Spezialfall:** 


**Lösung:**

- Sortiere (nach x-Koordinate).
- Berechne Abstände *aller aufeinanderfolgender Punktepaare*.
- Bestimme das Minimum dieser Abstände.

**Strukturelle Einsicht:**

# Mach's besser!

**Entwurfsparadigma:** – inkrementell?  
– randomisiert?  
– Teile und Herrsche?

**Spezialfall:** 

- Lösung:**
- Sortiere (nach x-Koordinate).
  - Berechne Abstände *aller aufeinanderfolgender Punktepaare*.
  - Bestimme das Minimum dieser Abstände.


**Strukturelle Einsicht:**

*Kandidatenmenge* der Größe  $n - 1$ ,  
die gesuchtes Objekt enthält.

# Mach's besser!

**Entwurfsparadigma:** – inkrementell?  
– randomisiert?

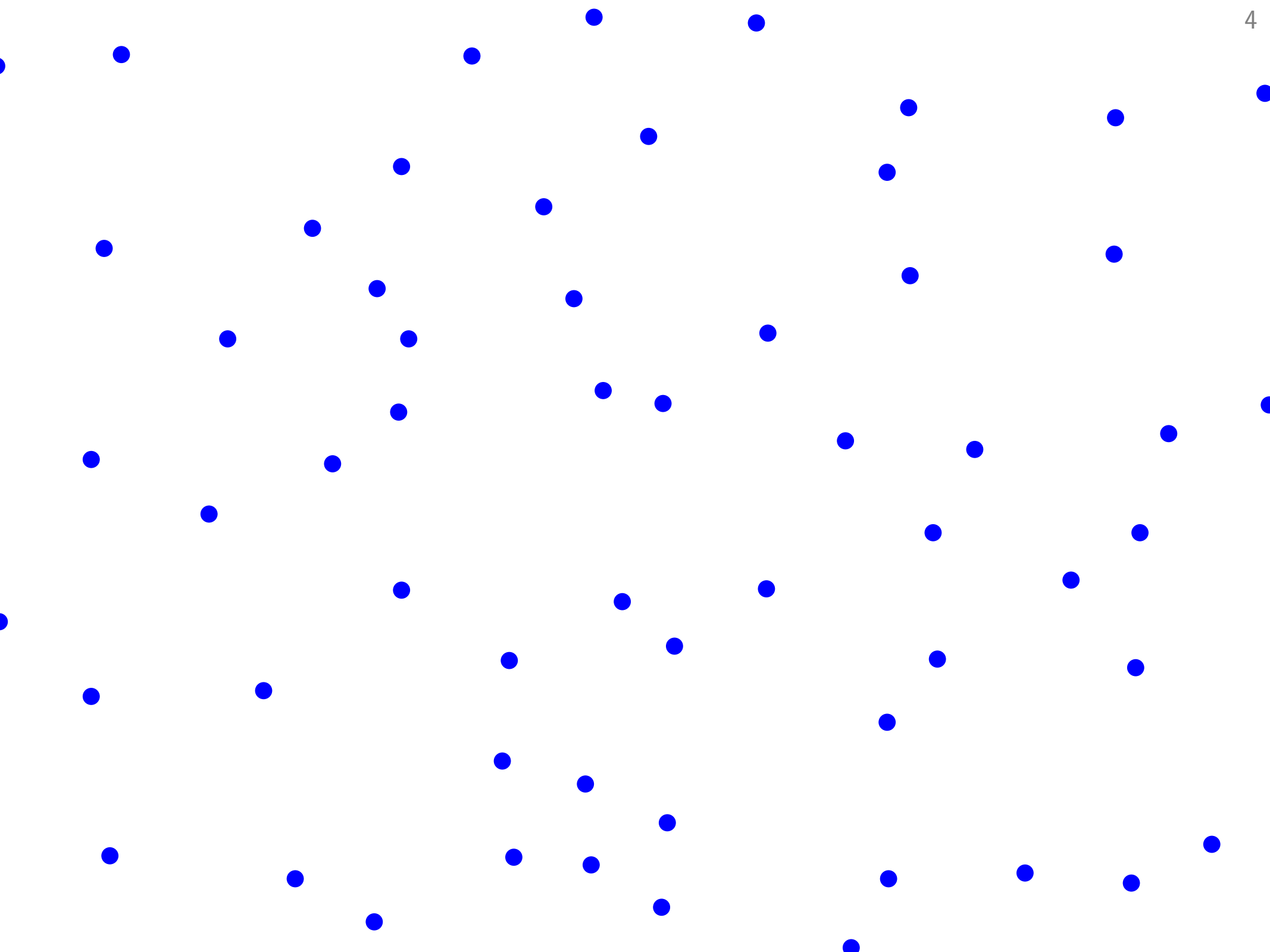
– Teile und Herrsche?!

**Spezialfall:** 

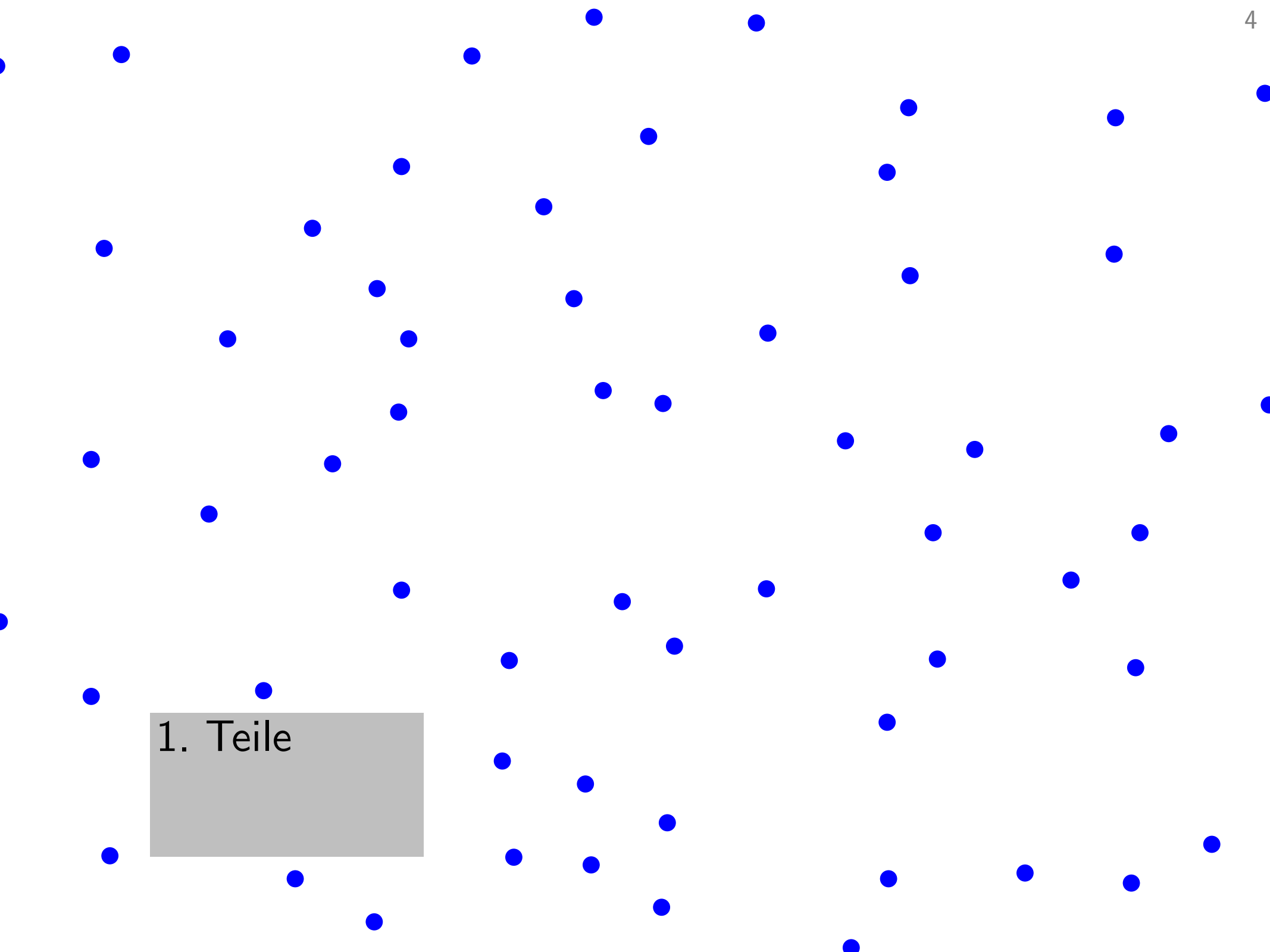
- Lösung:**
- Sortiere (nach x-Koordinate).
  - Berechne Abstände *aller aufeinanderfolgender* Punktepaare.
  - Bestimme das Minimum dieser Abstände.

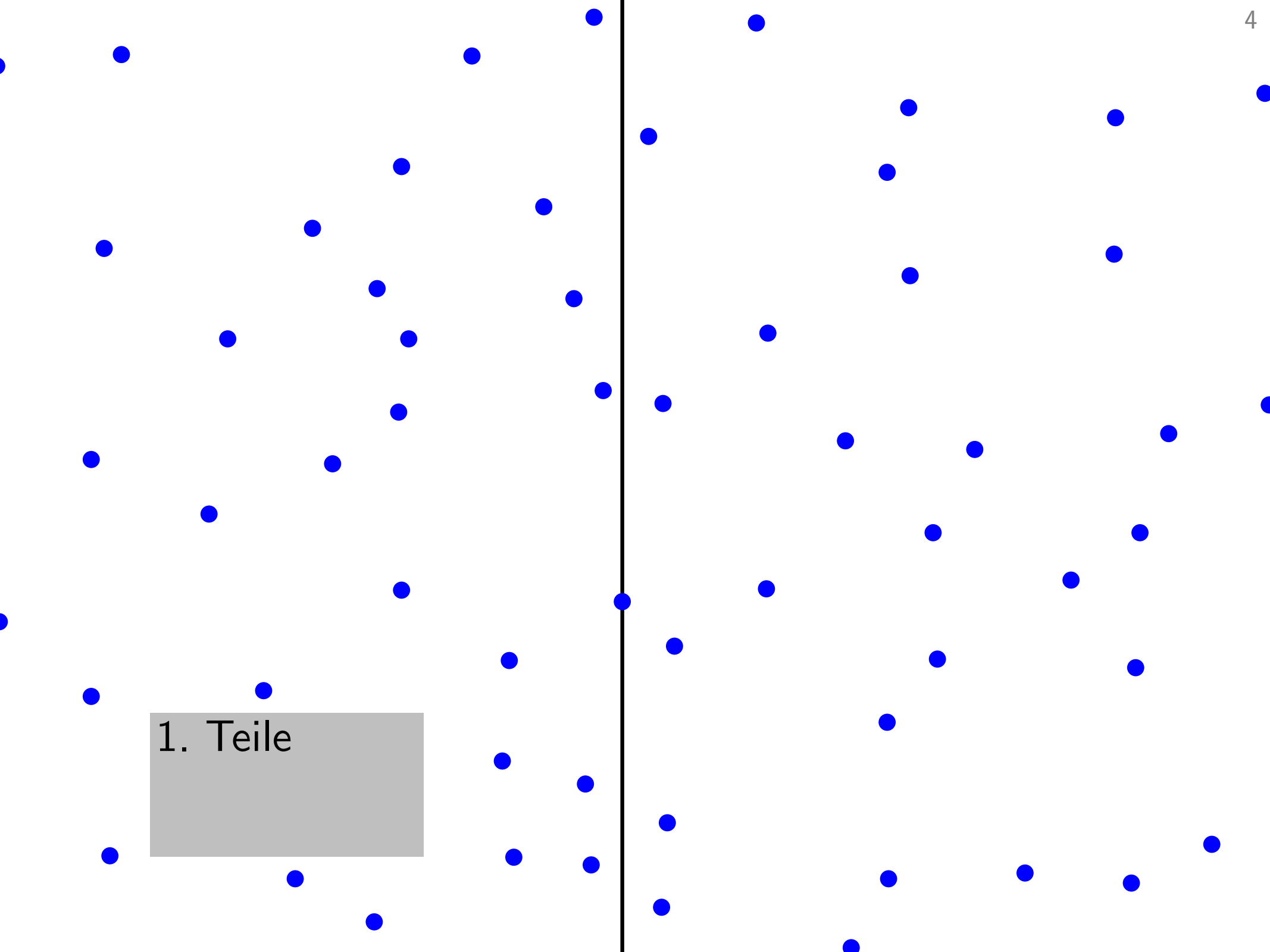
**Strukturelle Einsicht:**

*Kandidatenmenge* der Größe  $n - 1$ ,  
die gesuchtes Objekt enthält.

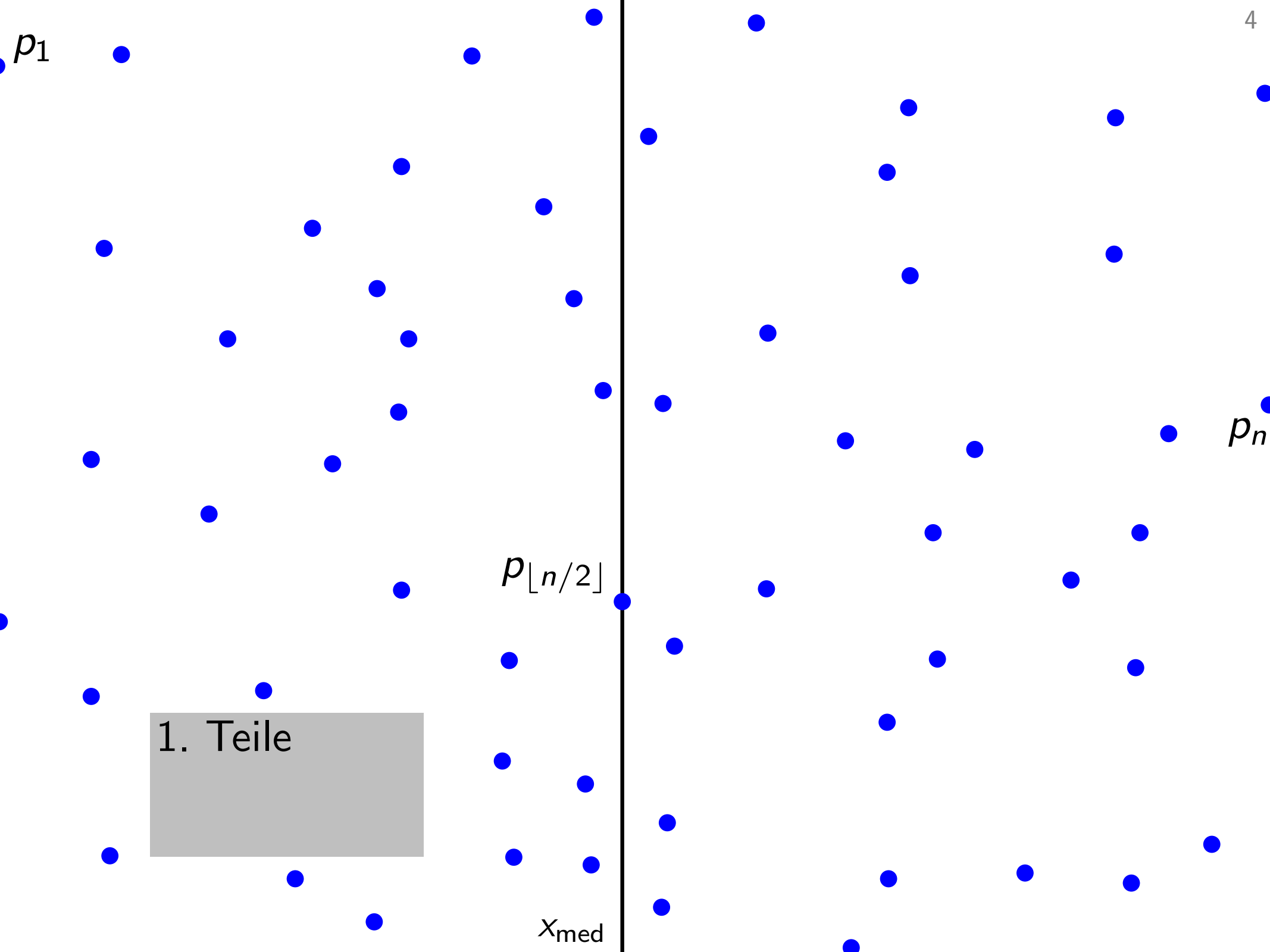


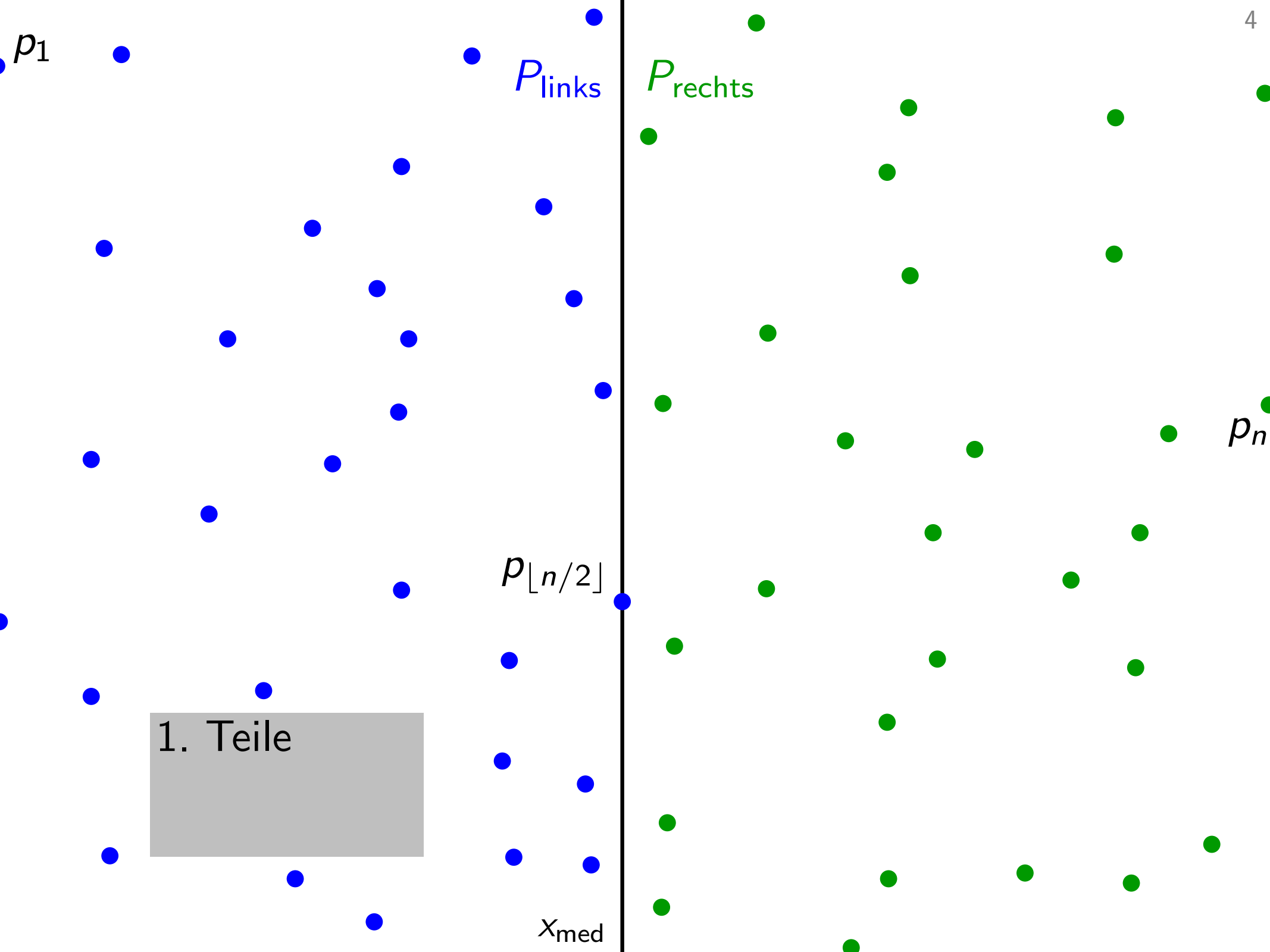
1. Teile



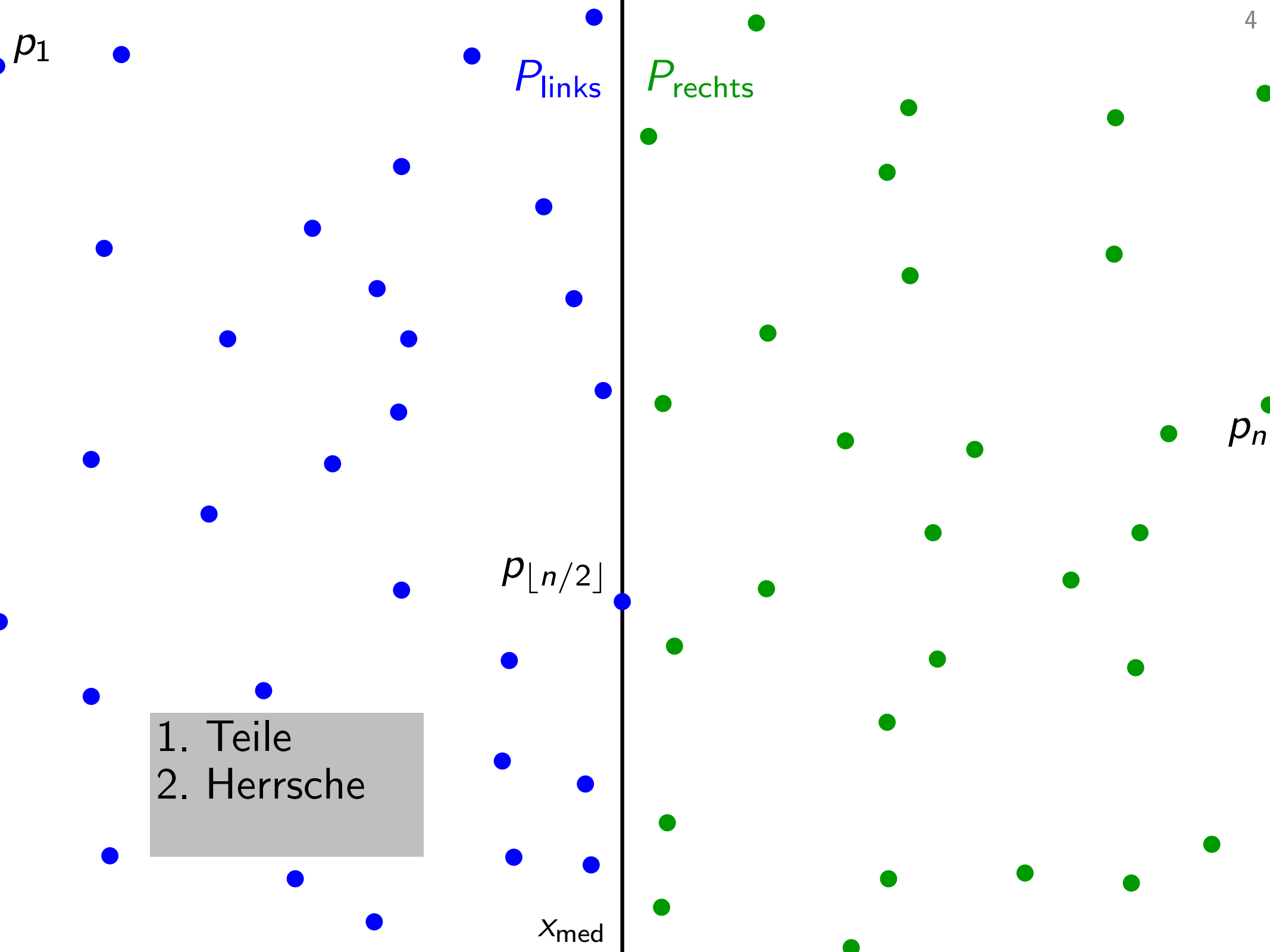


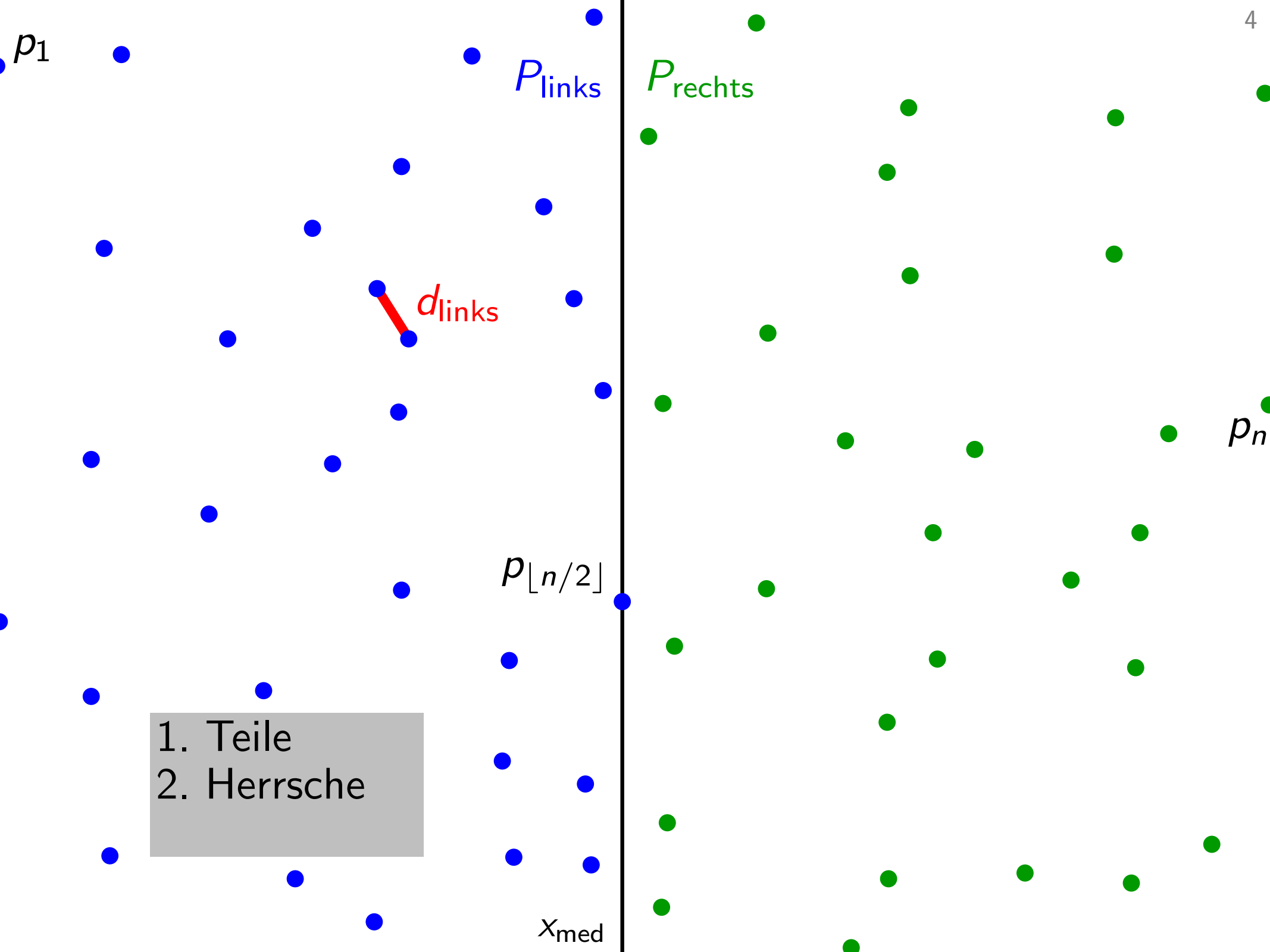
1. Teile

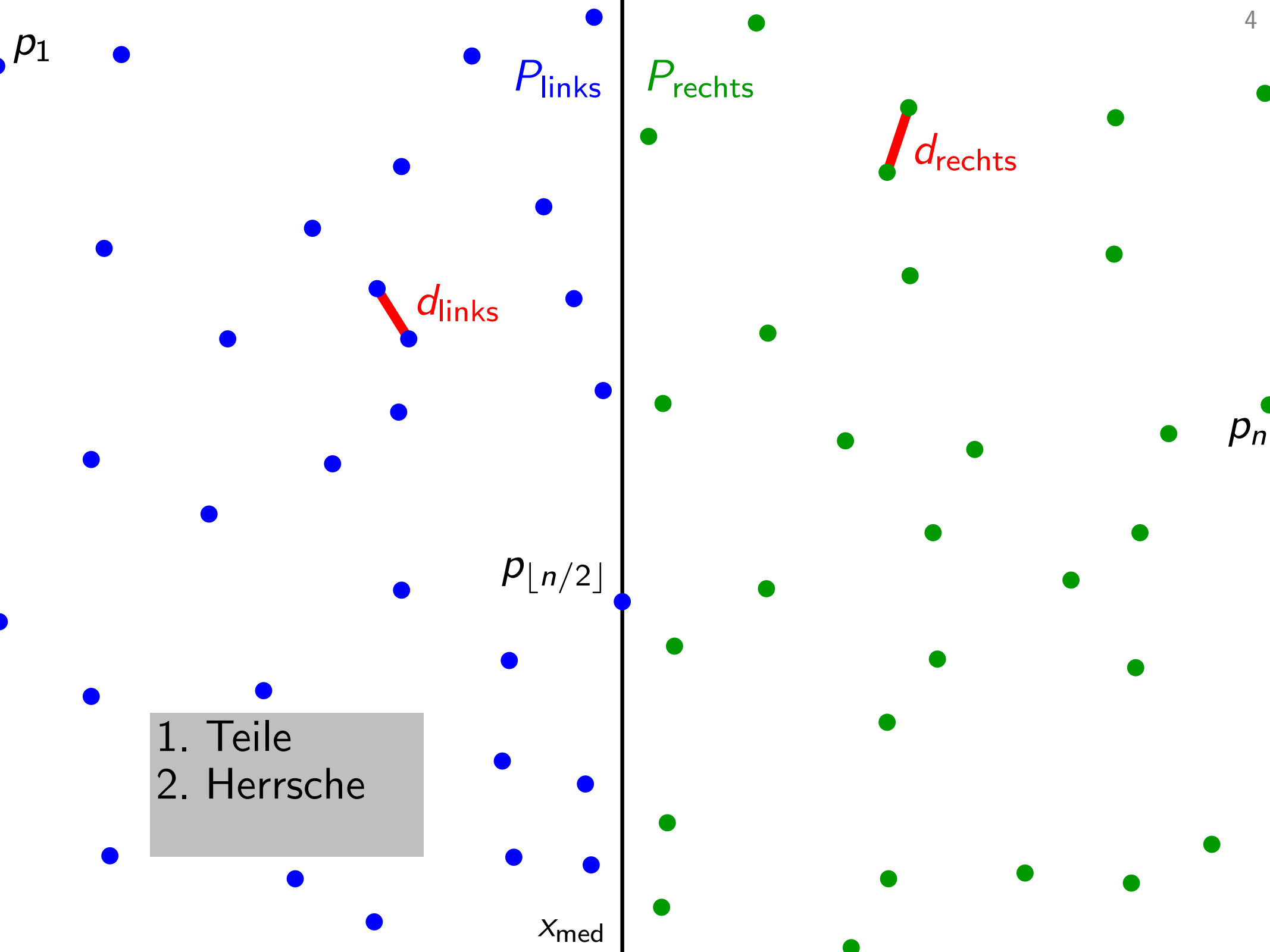


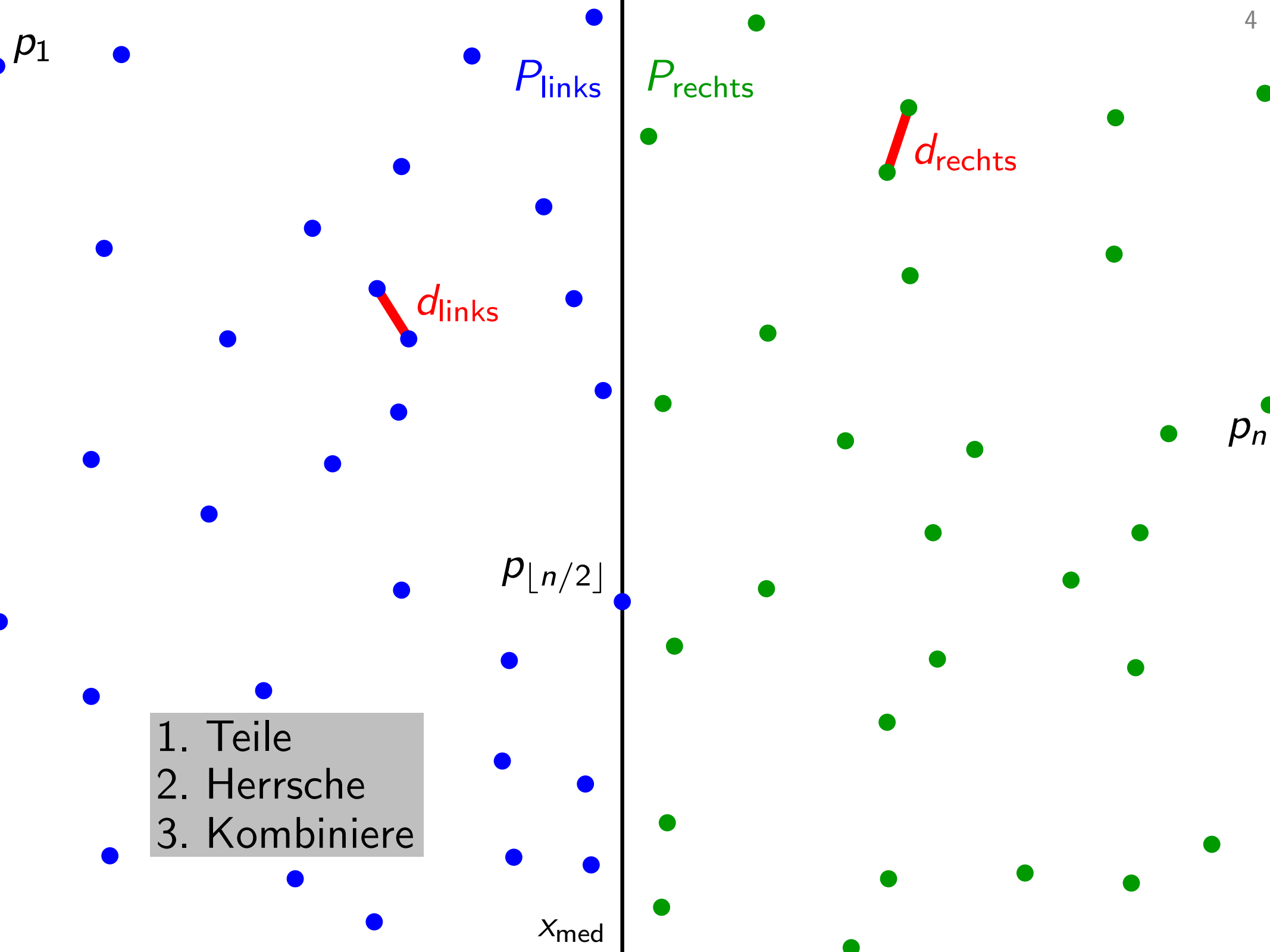


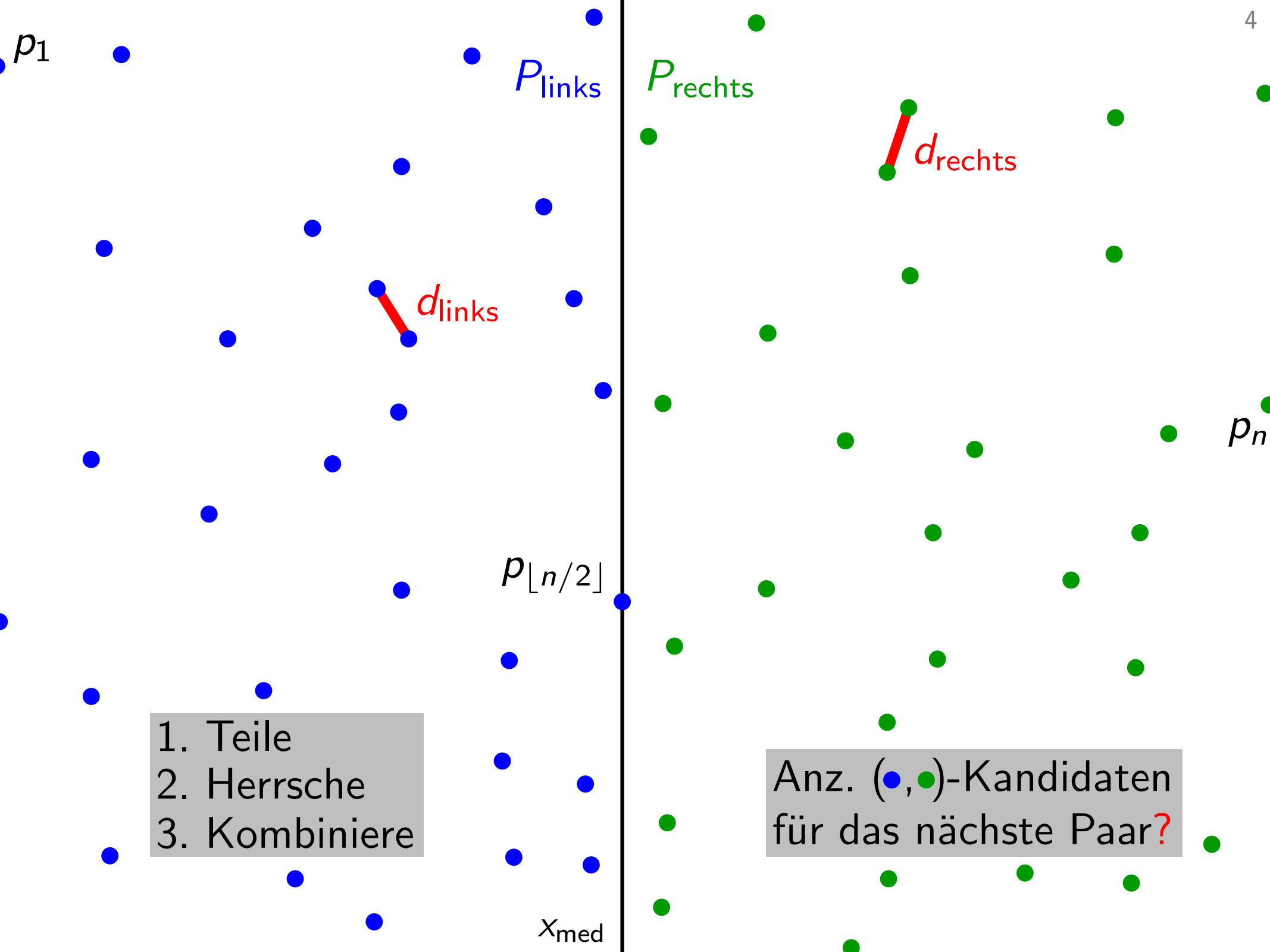












$p_1$

$P_{\text{links}}$

$P_{\text{rechts}}$

4

$d_{\text{links}}$

$d_{\text{rechts}}$

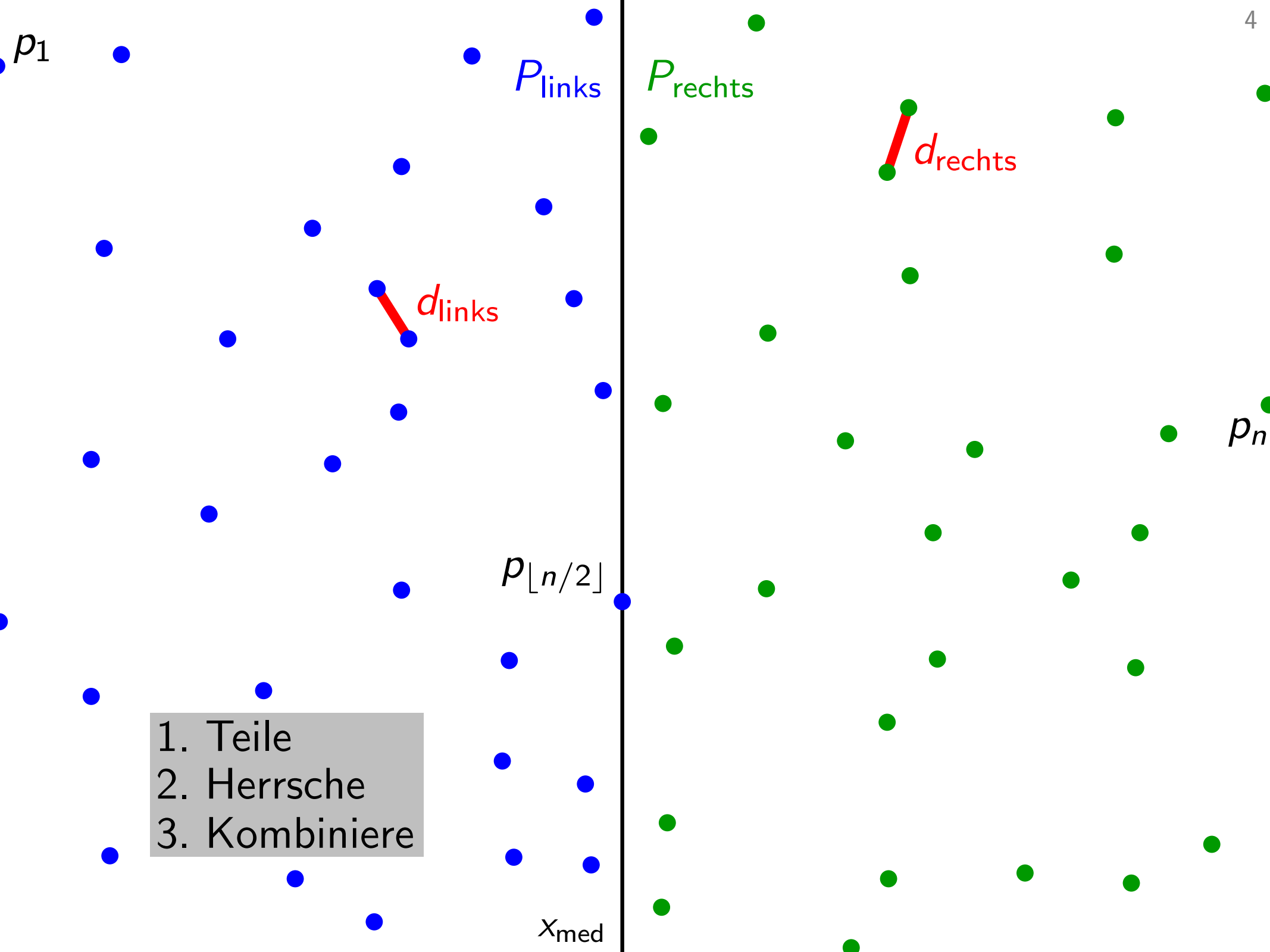
$p_n$

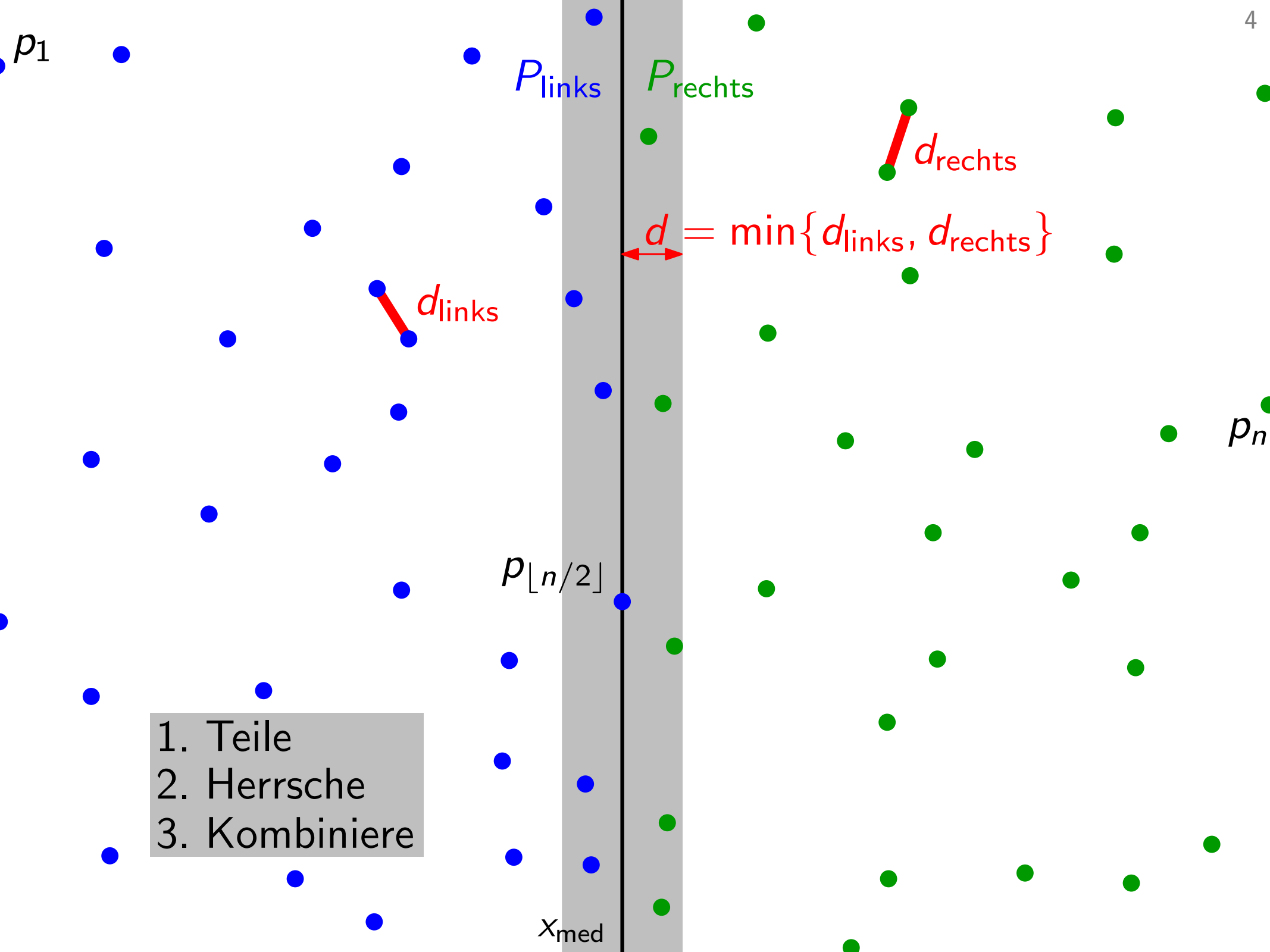
$p_{\lfloor n/2 \rfloor}$

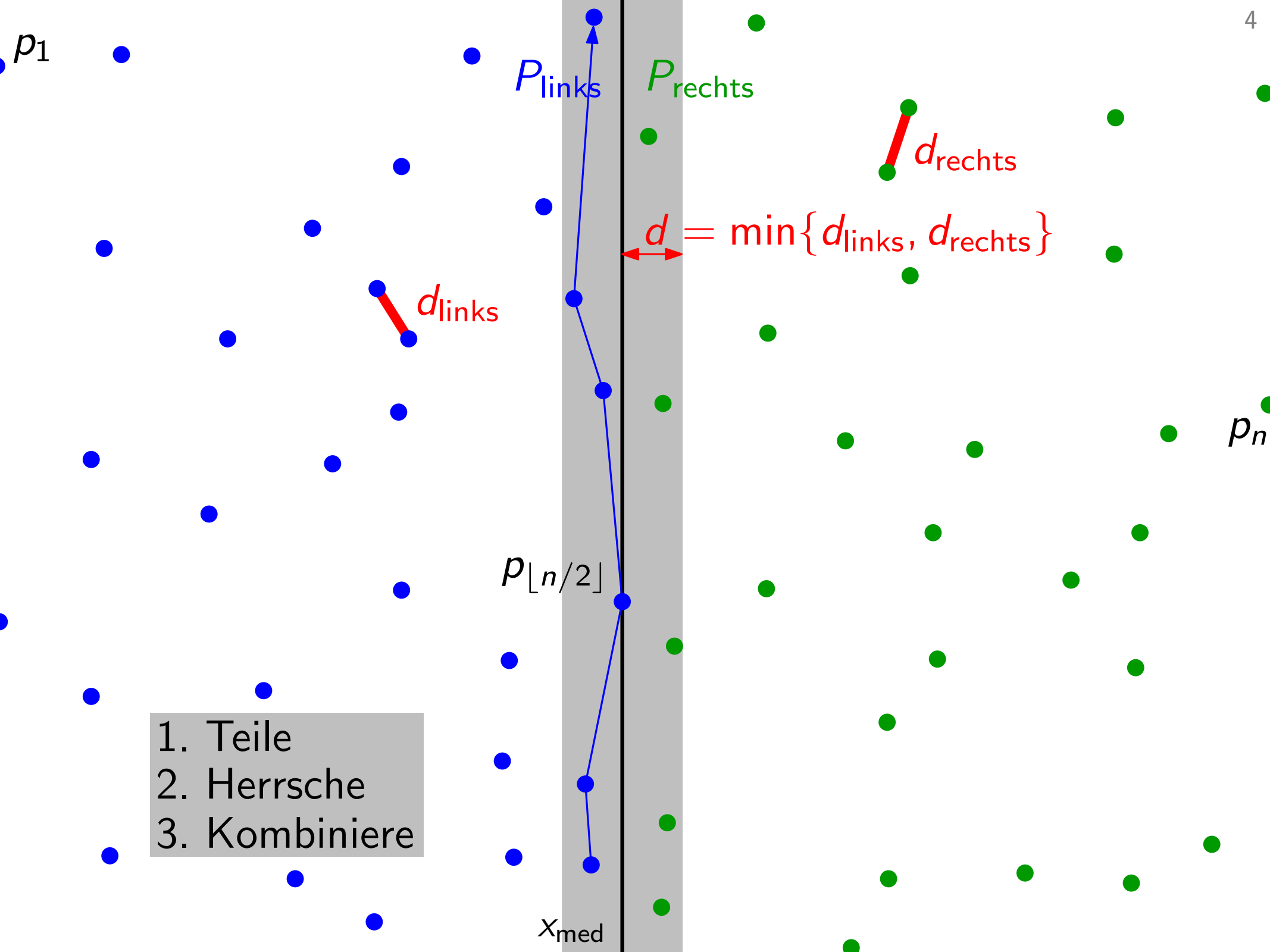
- 1. Teile
- 2. Herrsche
- 3. Kombiniere

Anz.  $(\bullet, \bullet)$ -Kandidaten für das nächste Paar?

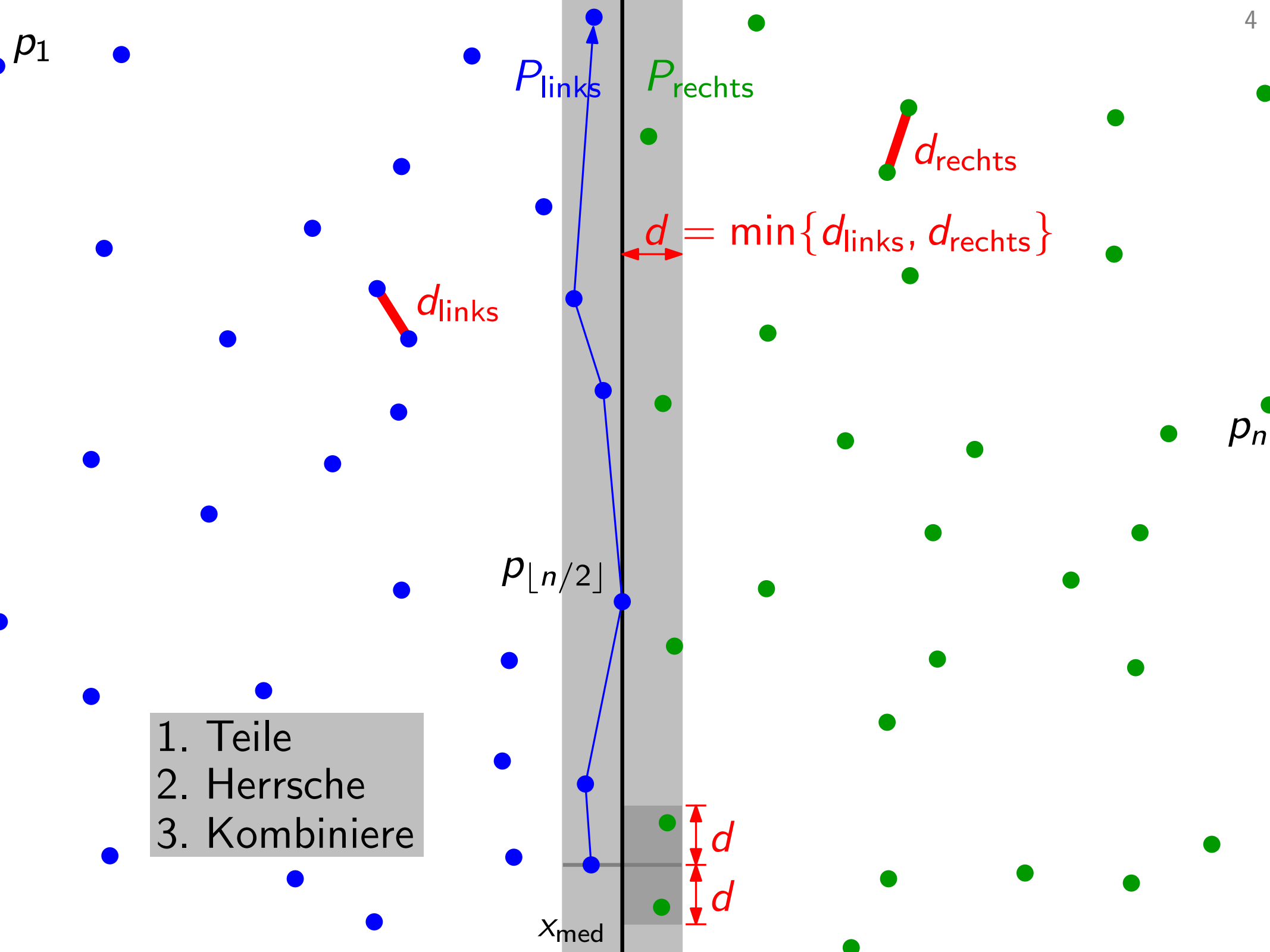
$x_{\text{med}}$

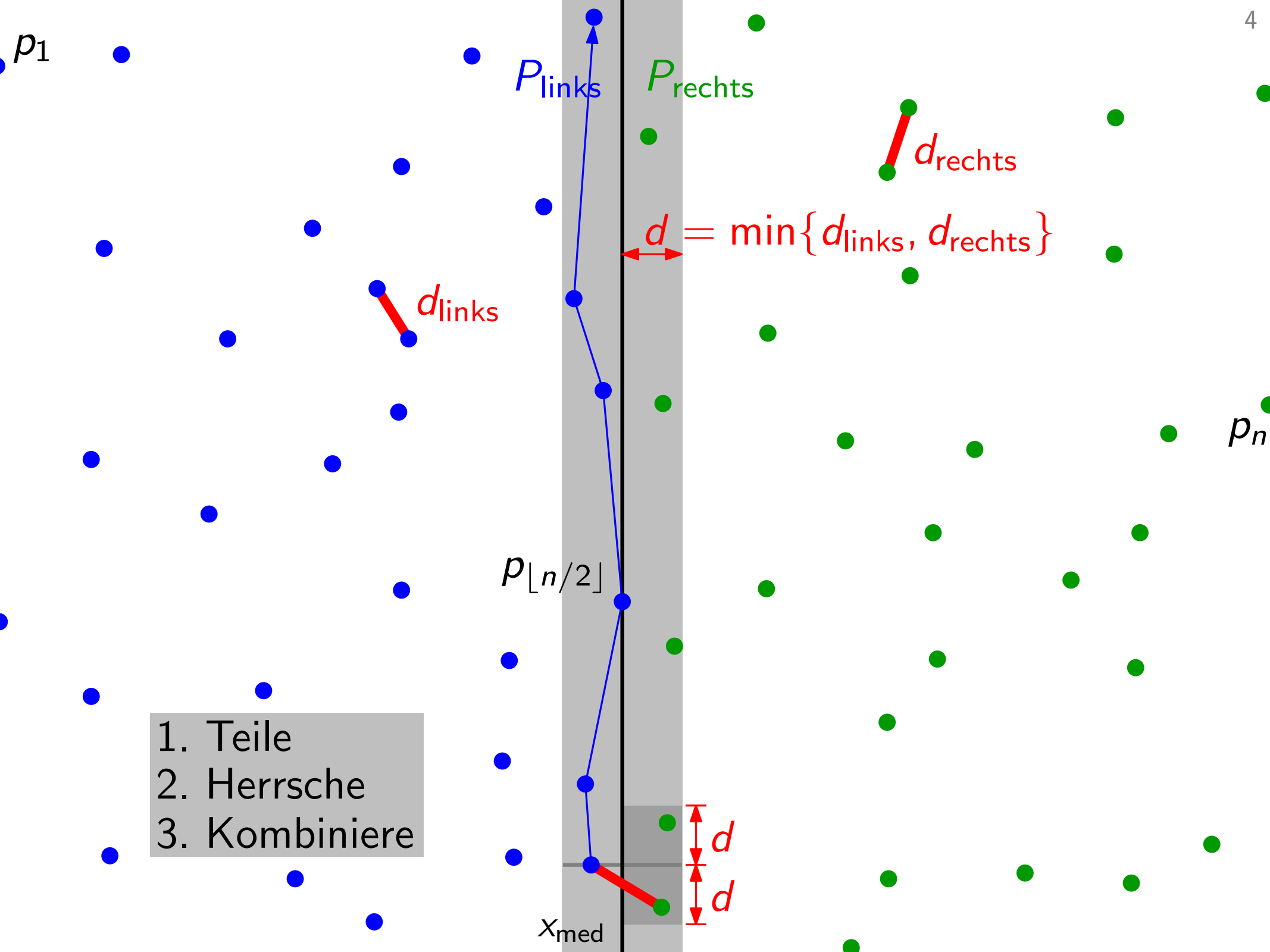


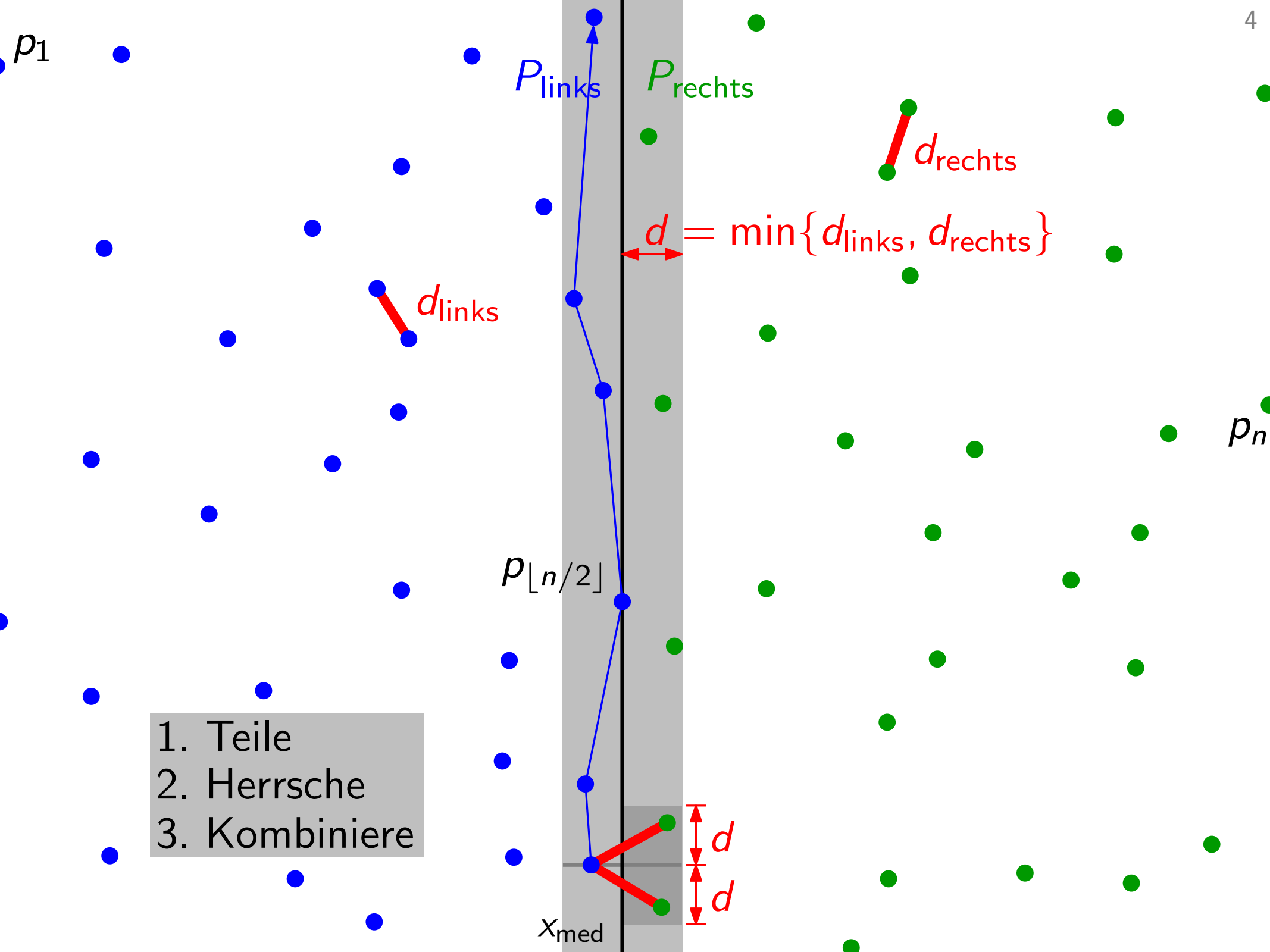


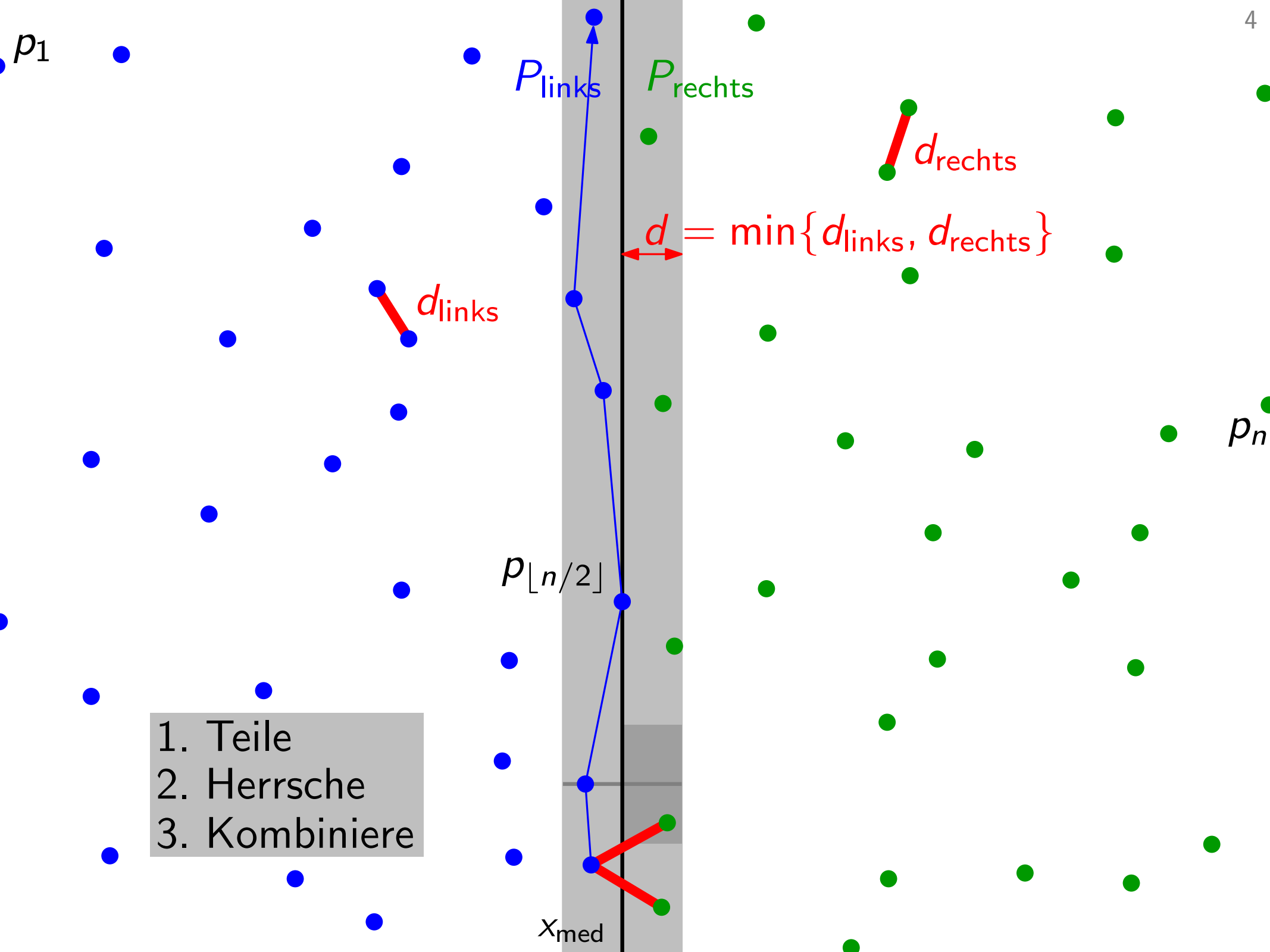


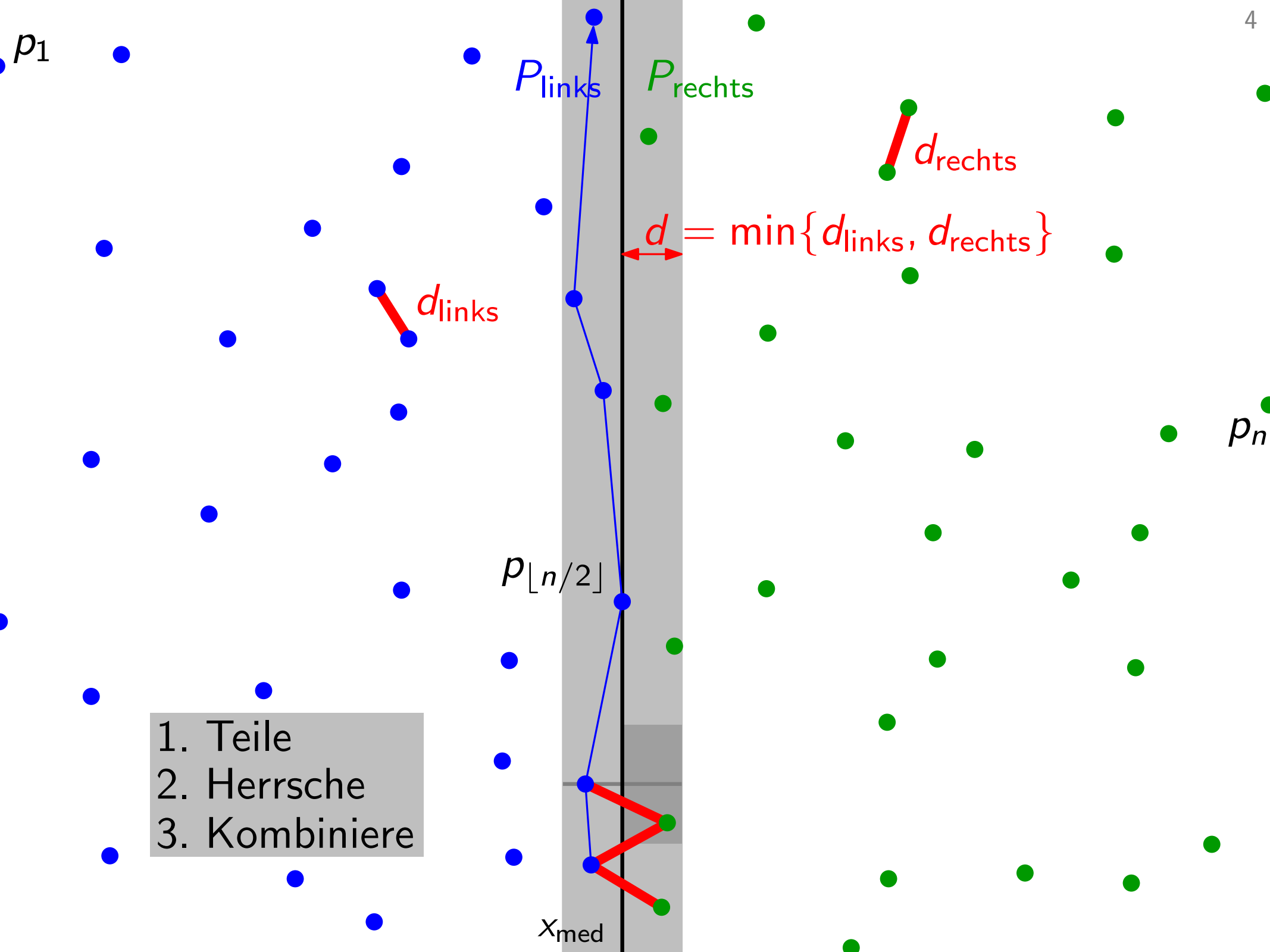


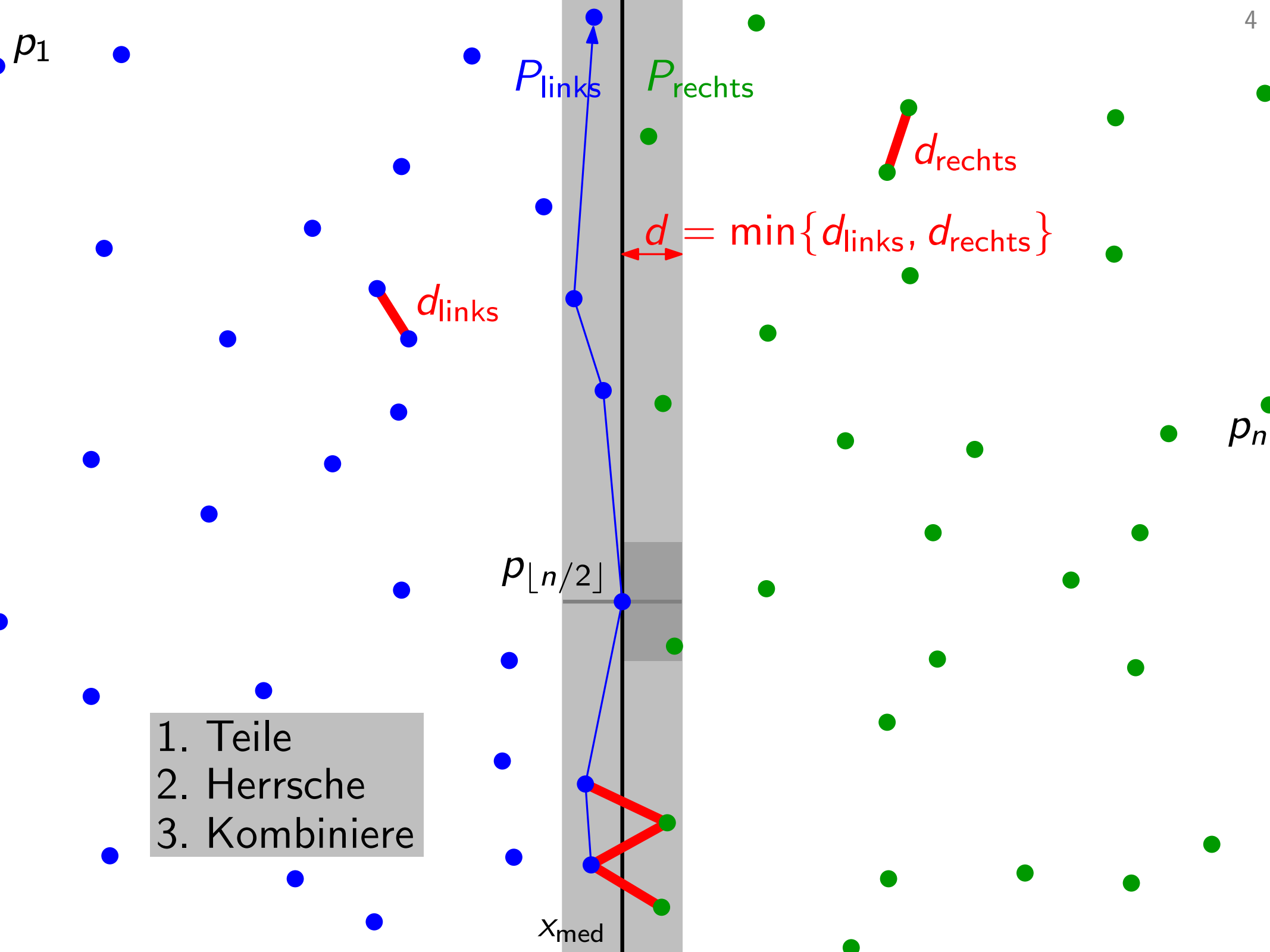


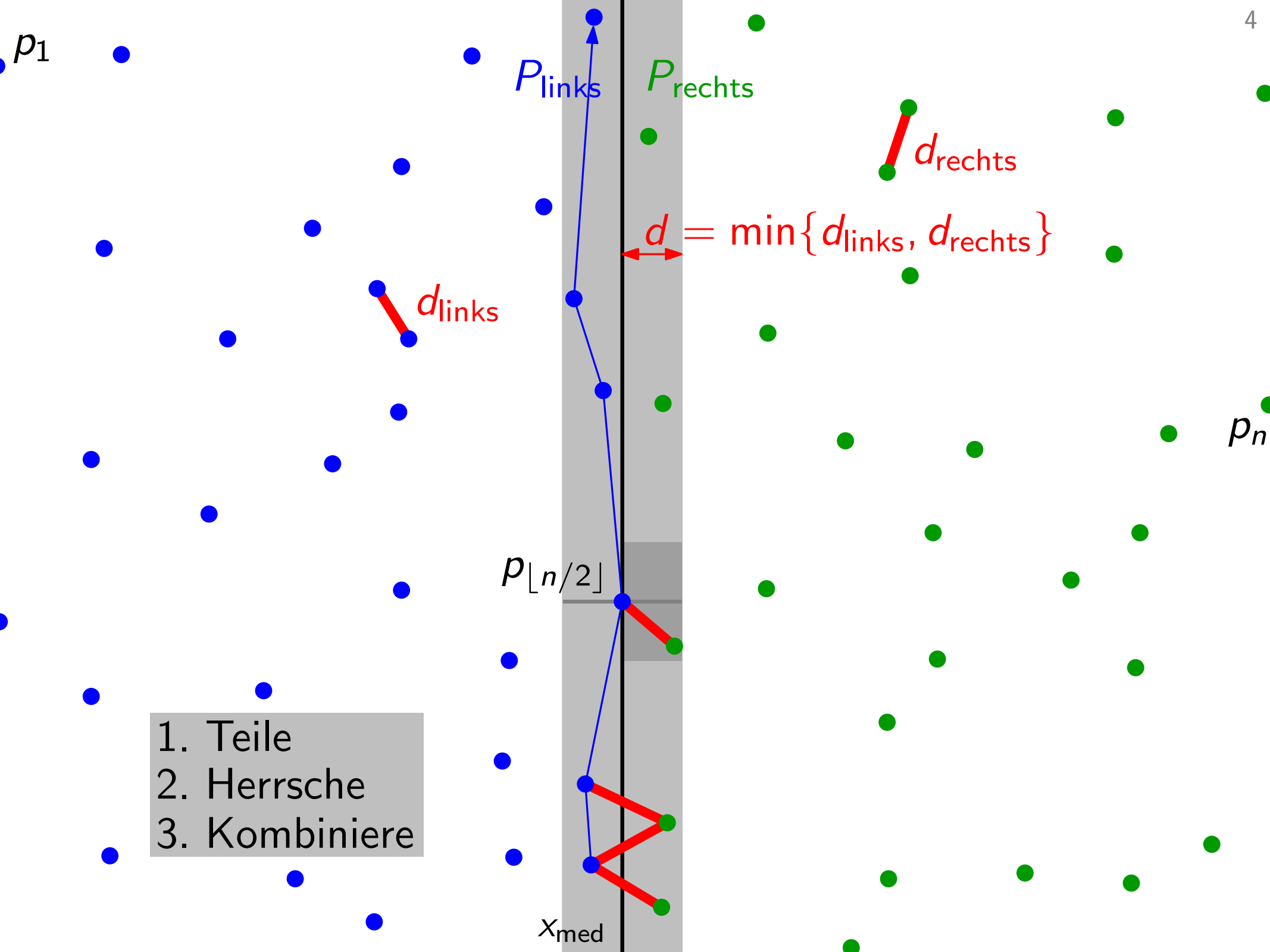


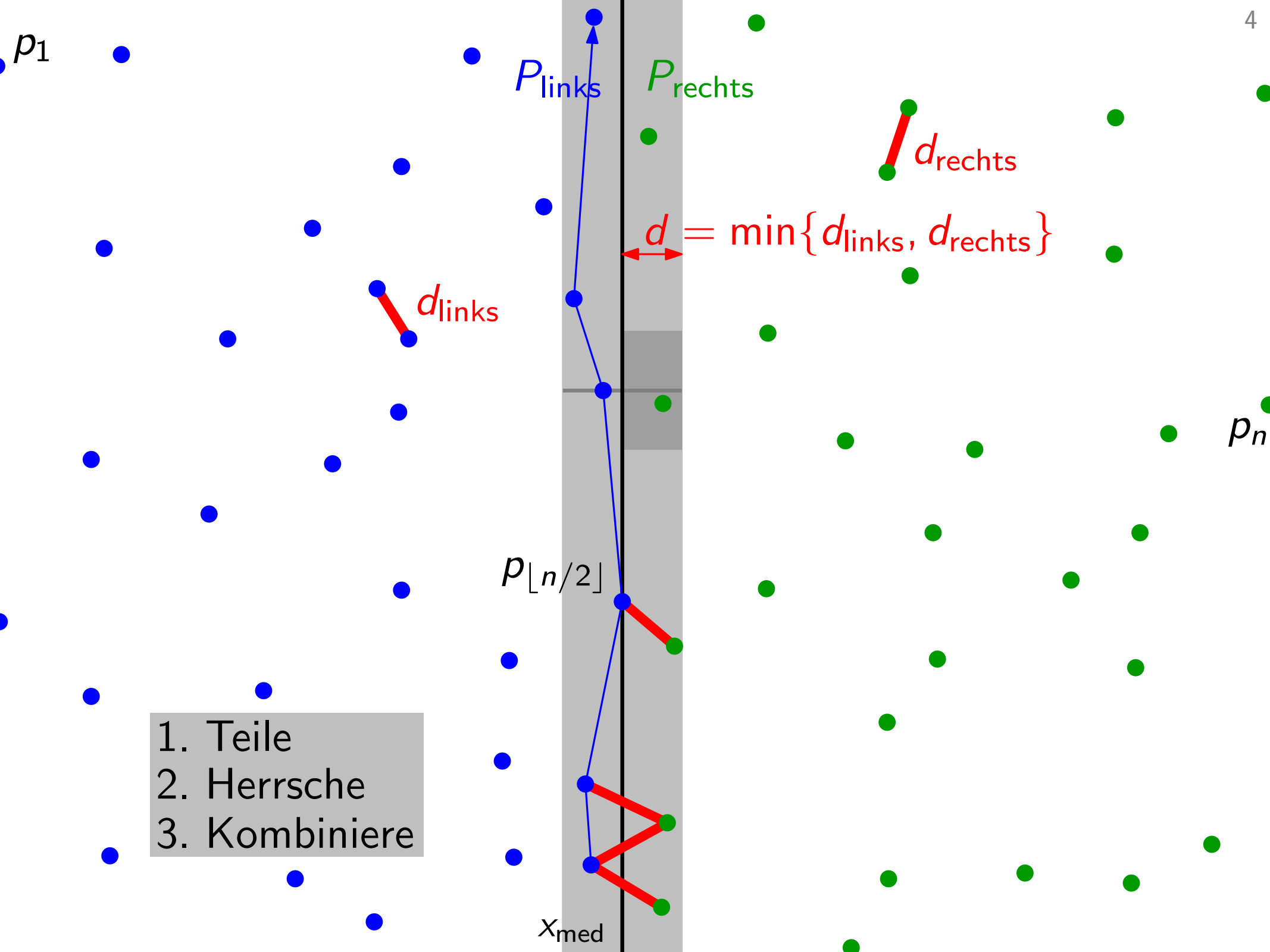






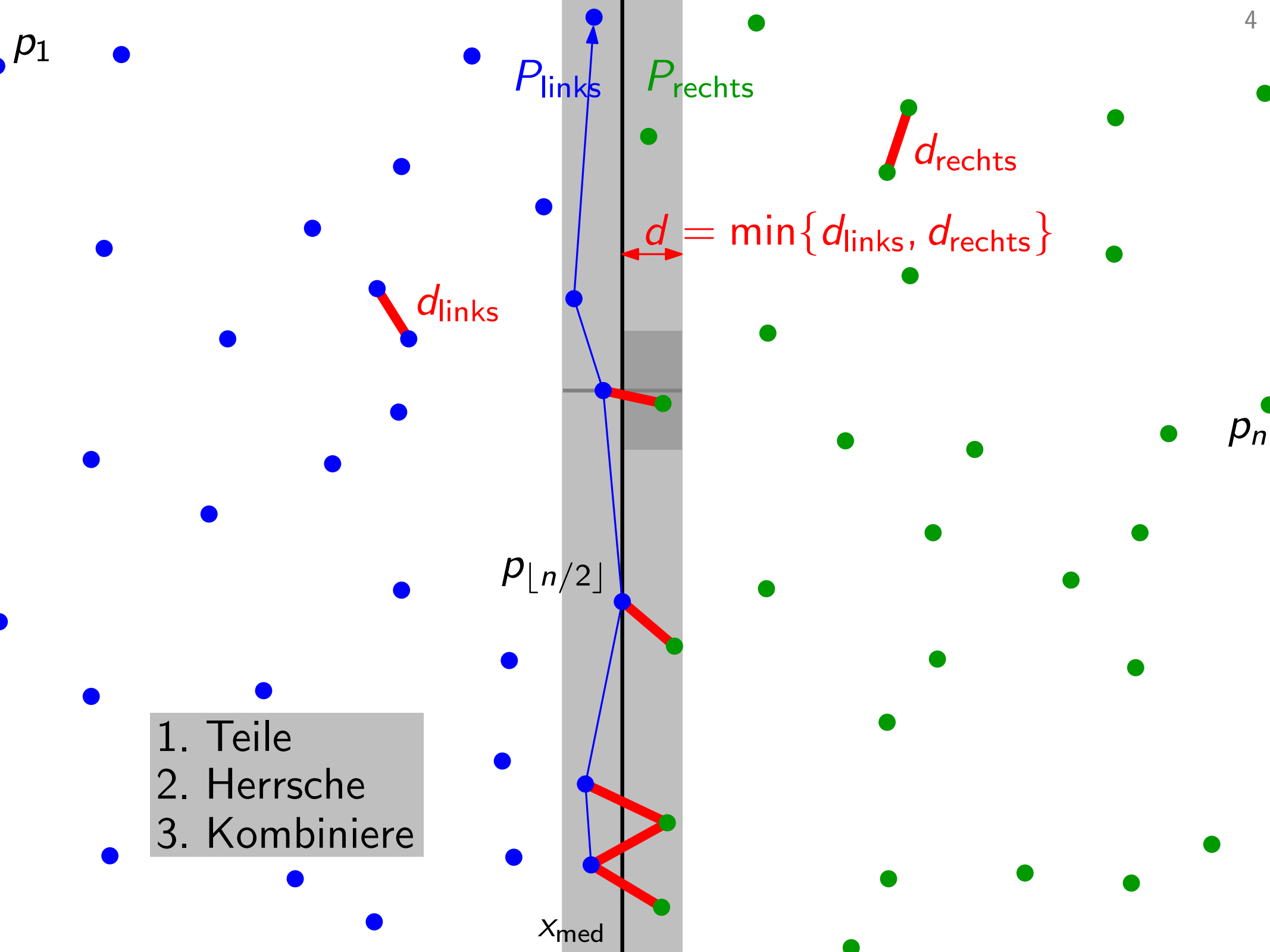




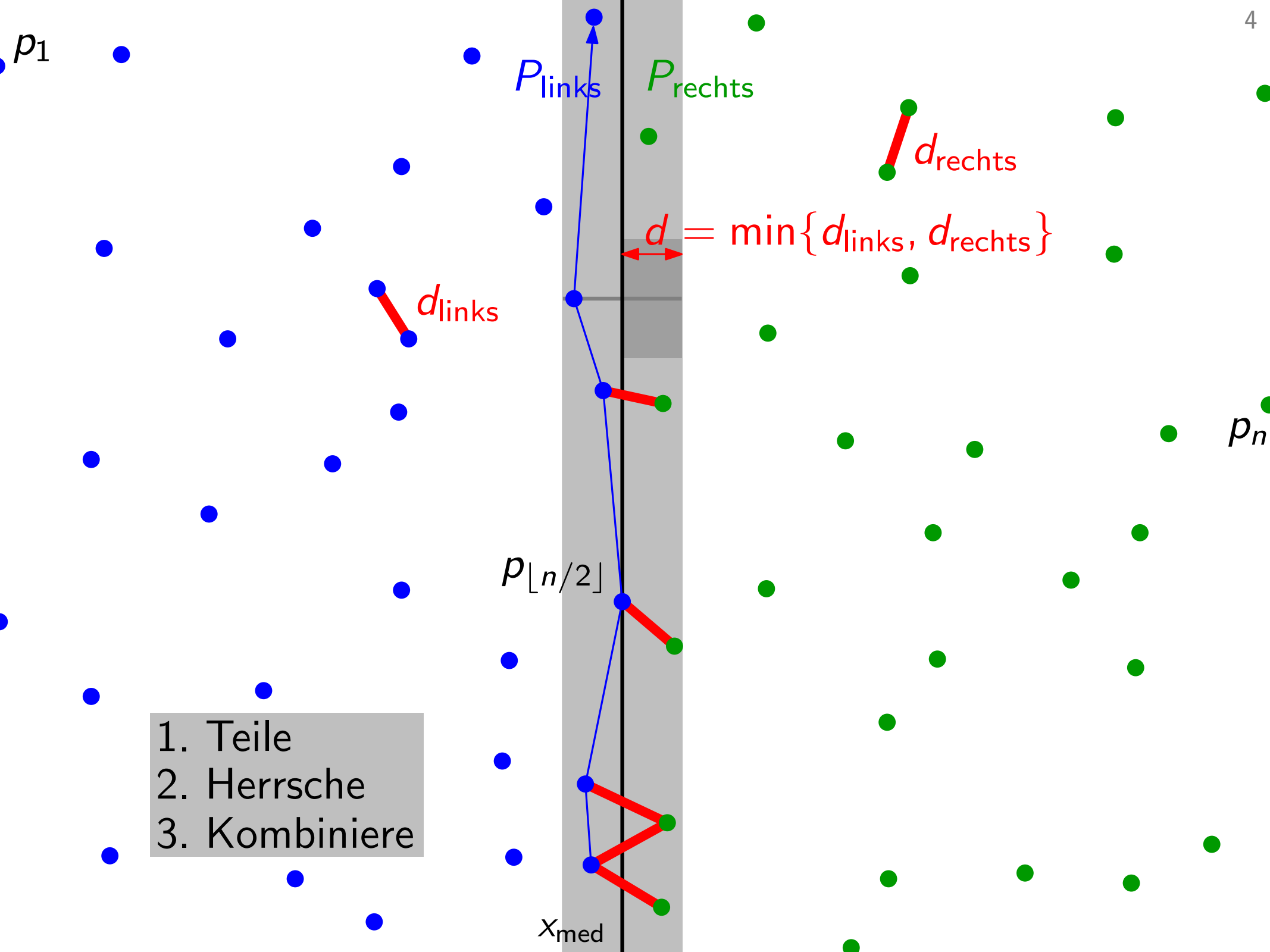


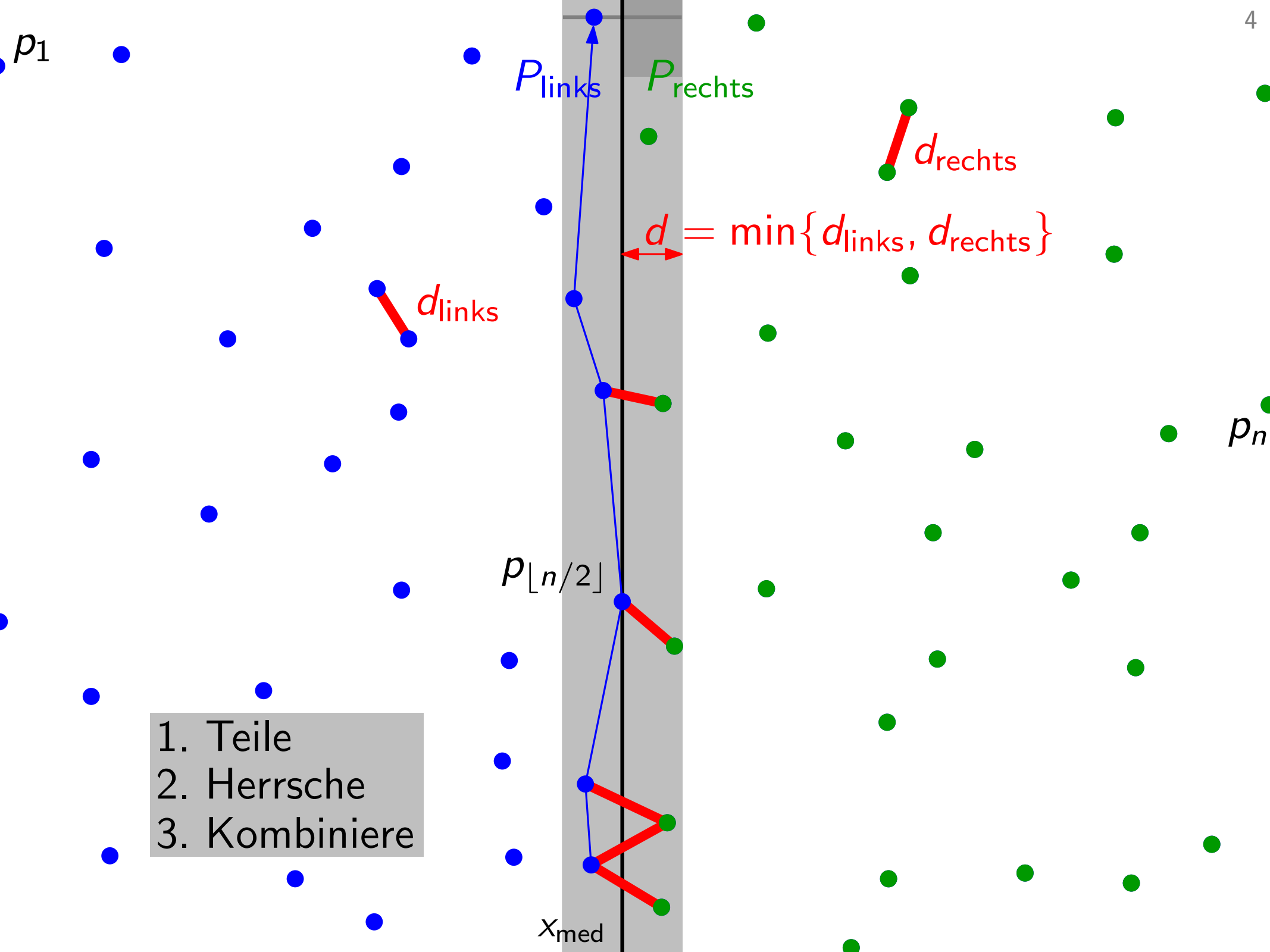
1. Teile
2. Herrsche
3. Kombiniere

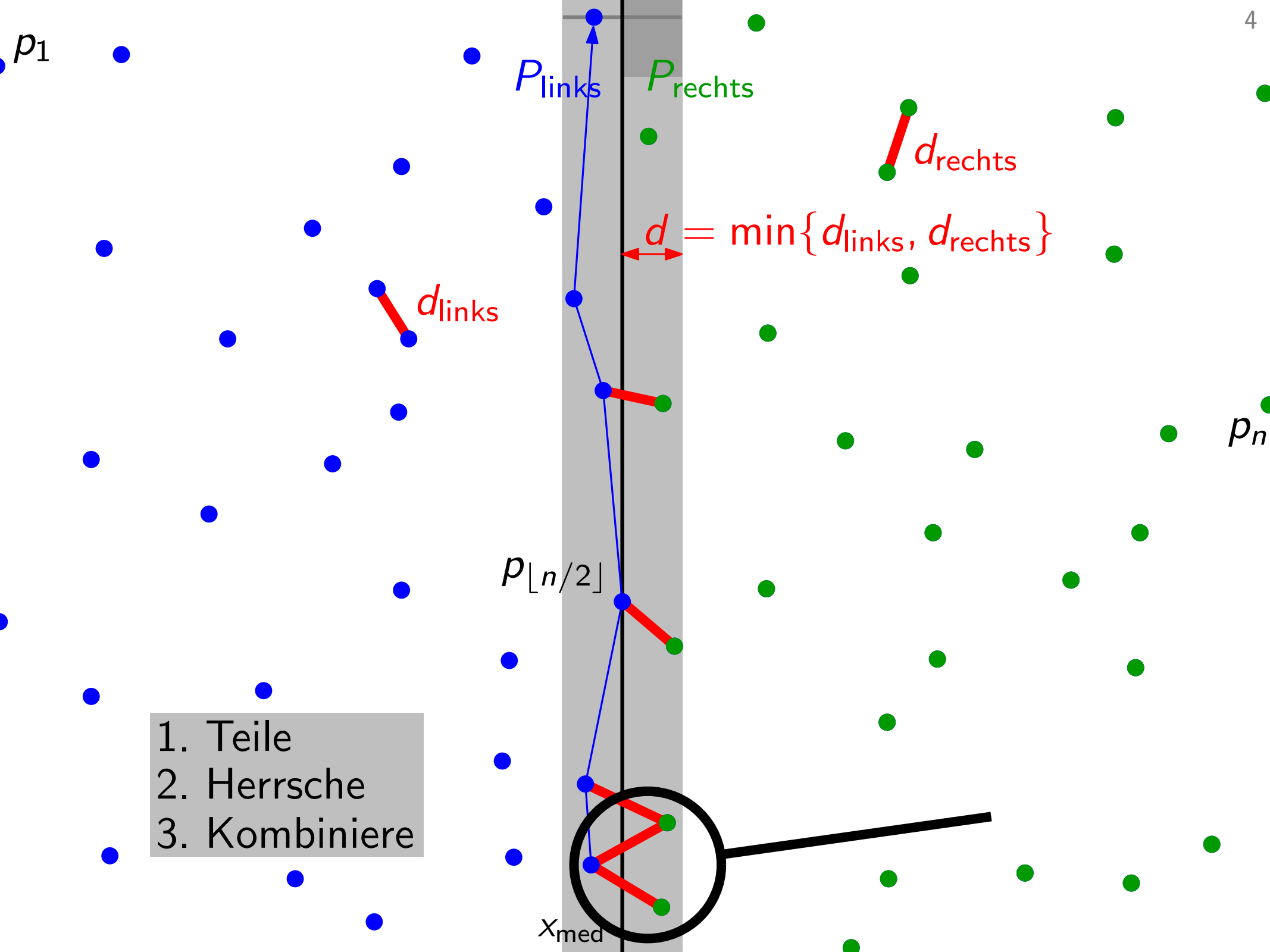


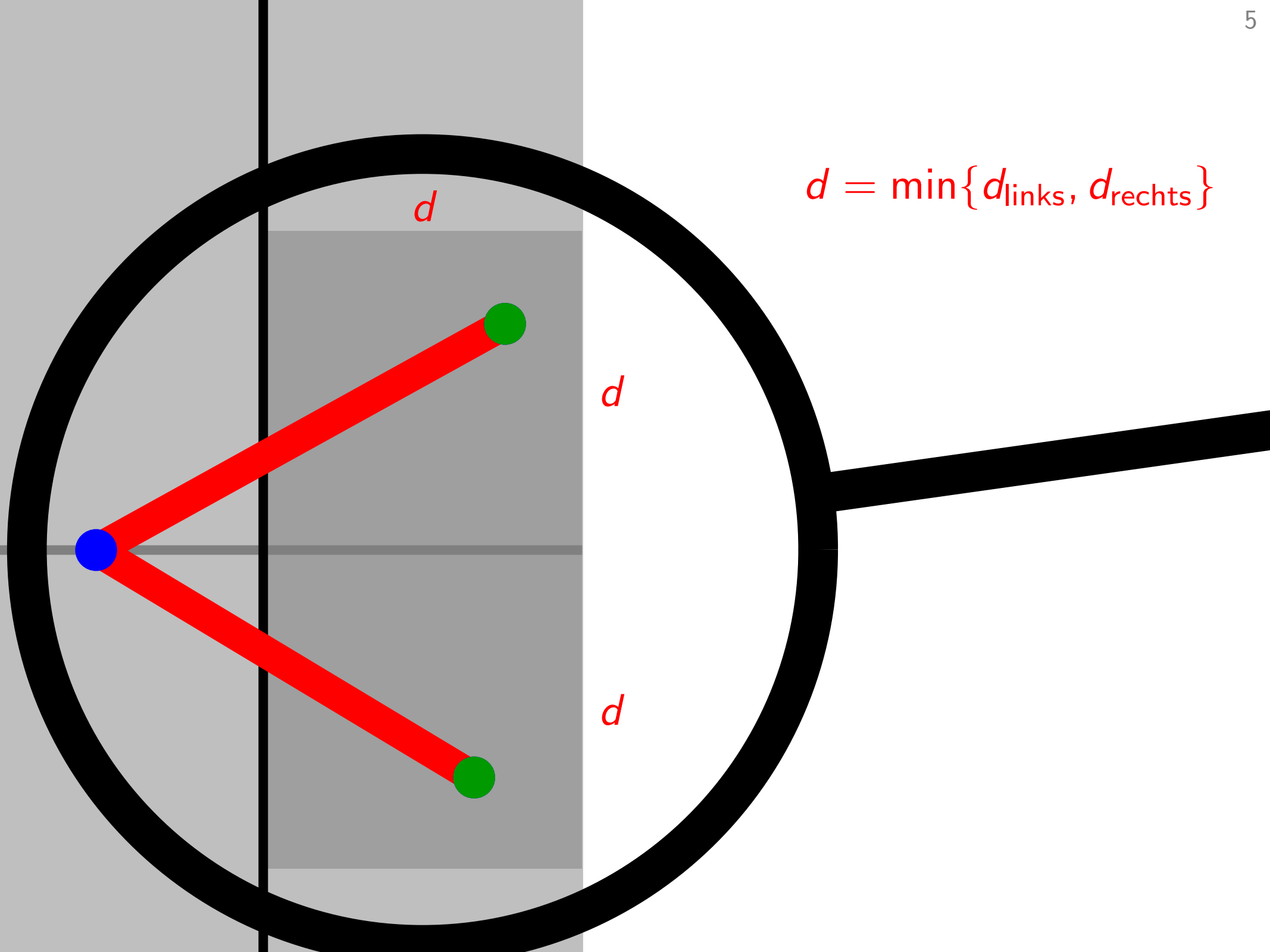


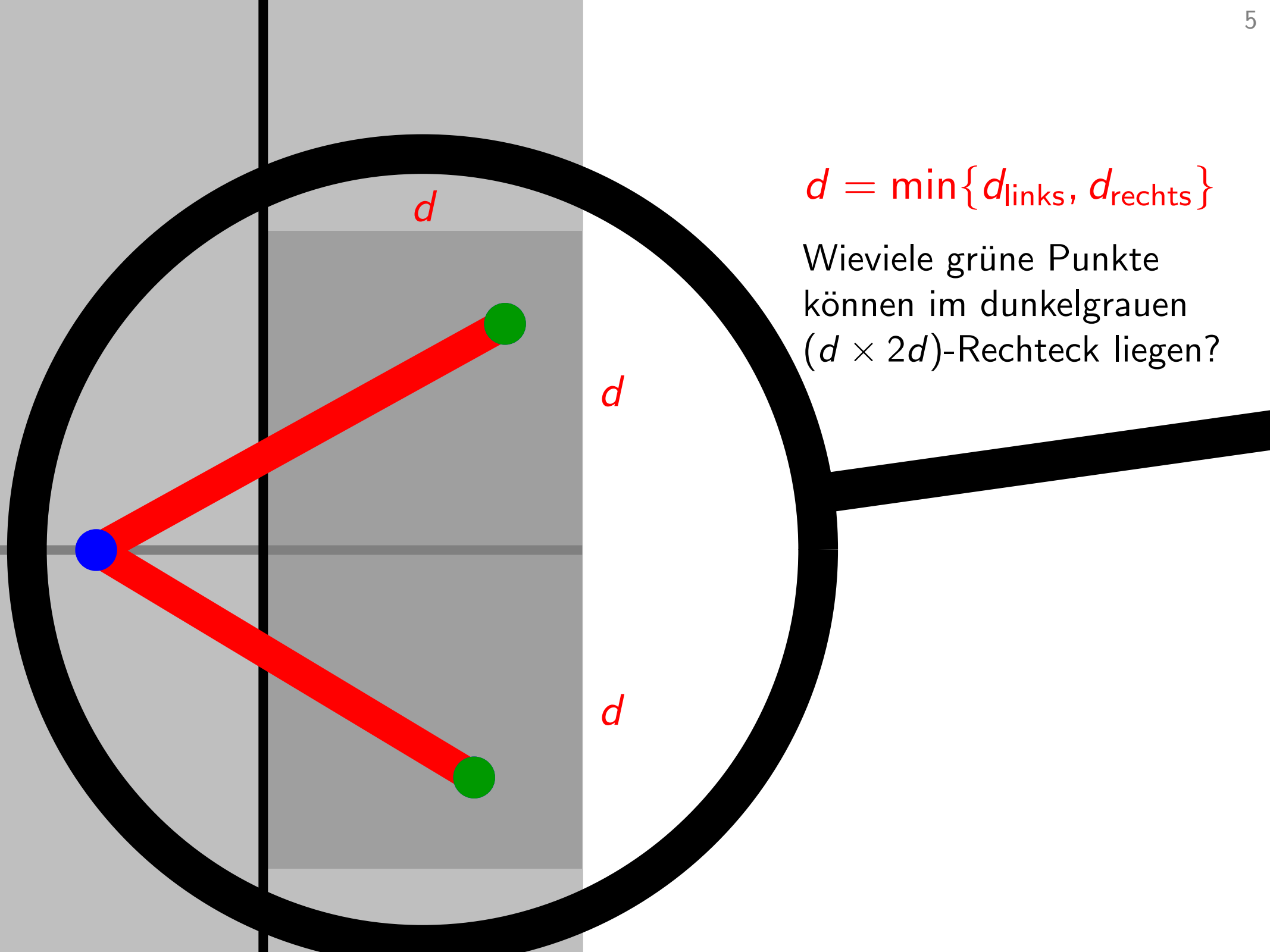
- 1. Teile
- 2. Herrsche
- 3. Kombiniere





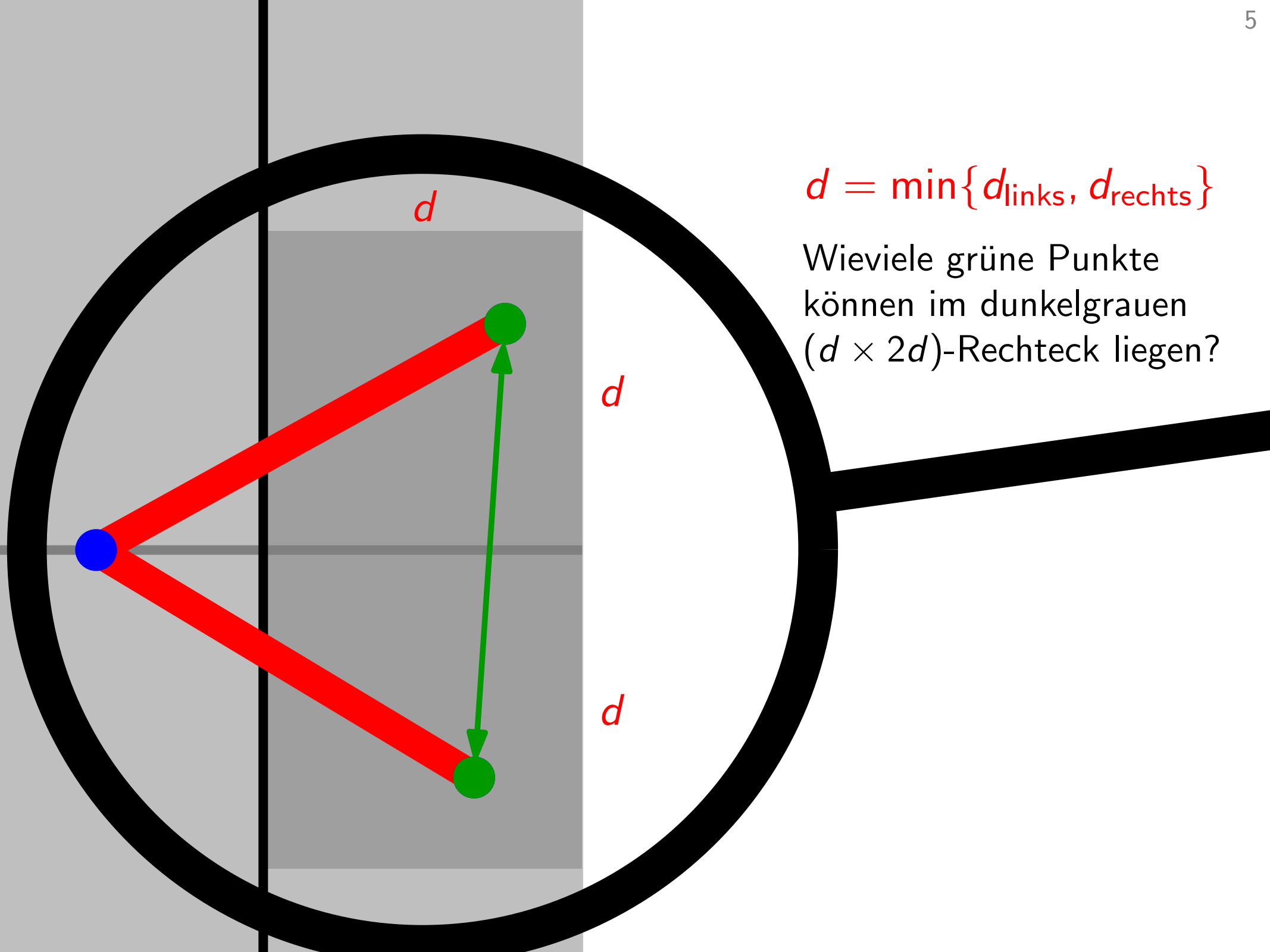






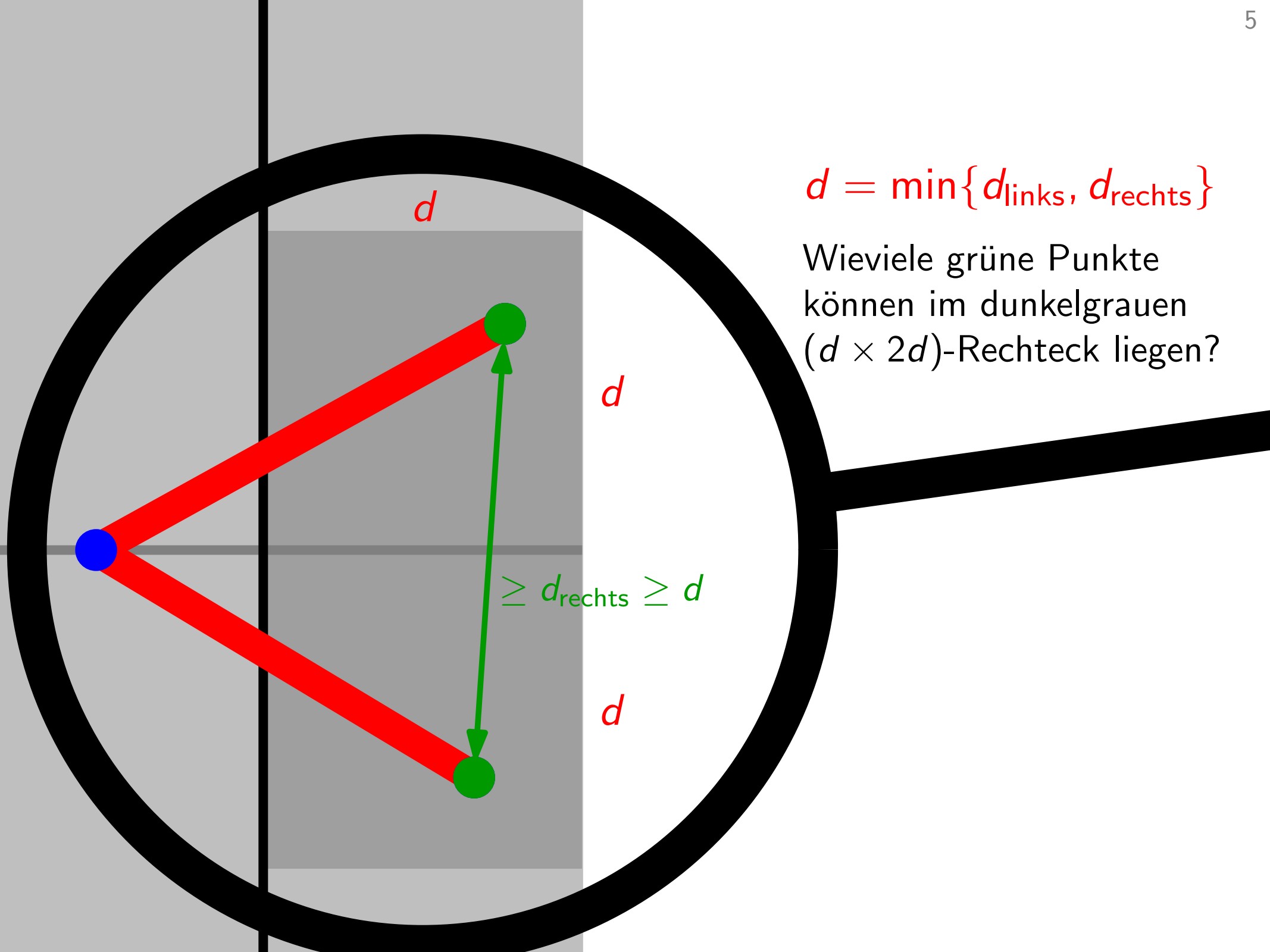
$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte  
können im dunkelgrauen  
( $d \times 2d$ )-Rechteck liegen?



$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

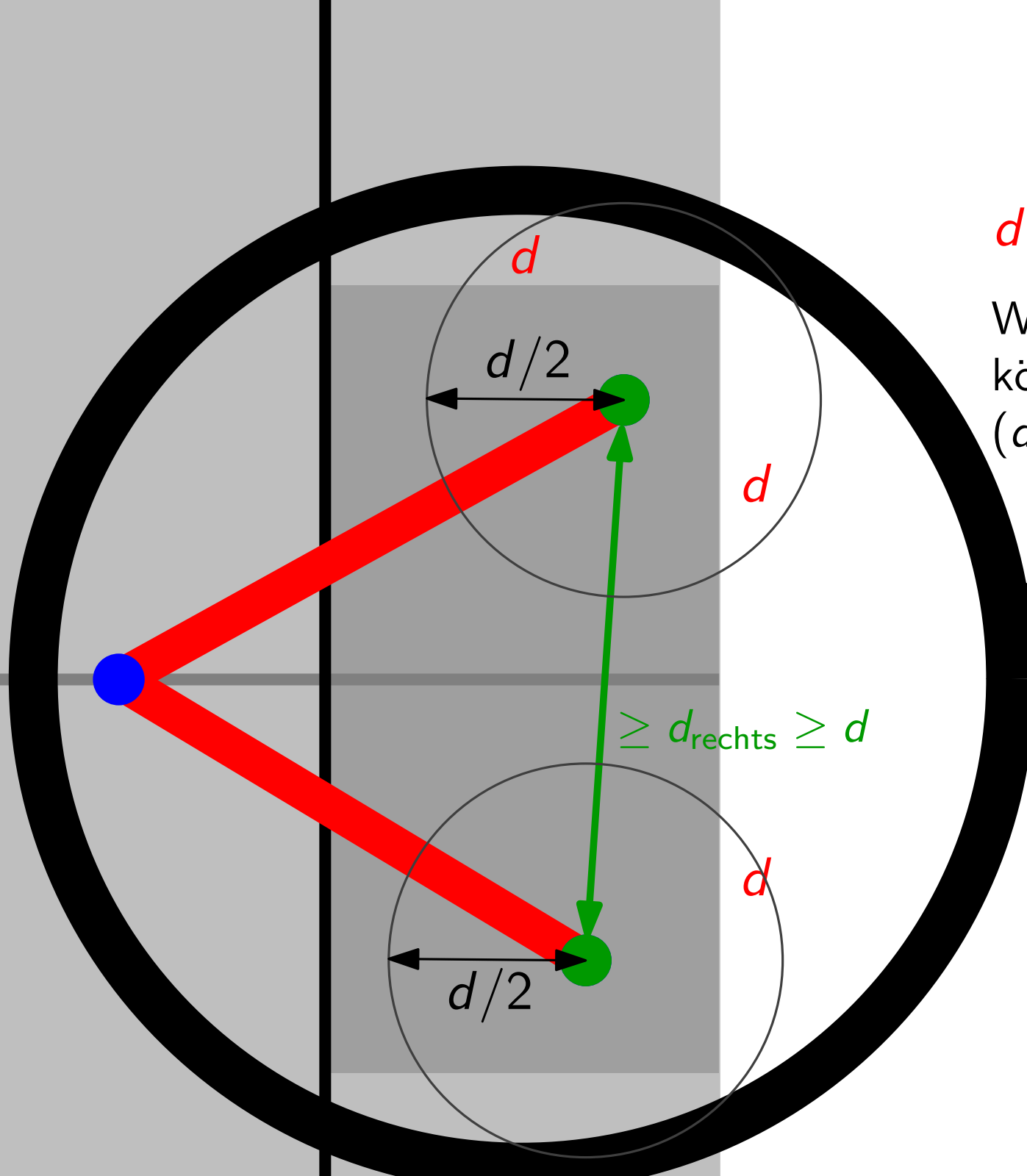
Wieviele grüne Punkte  
können im dunkelgrauen  
( $d \times 2d$ )-Rechteck liegen?



$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte  
können im dunkelgrauen  
( $d \times 2d$ )-Rechteck liegen?

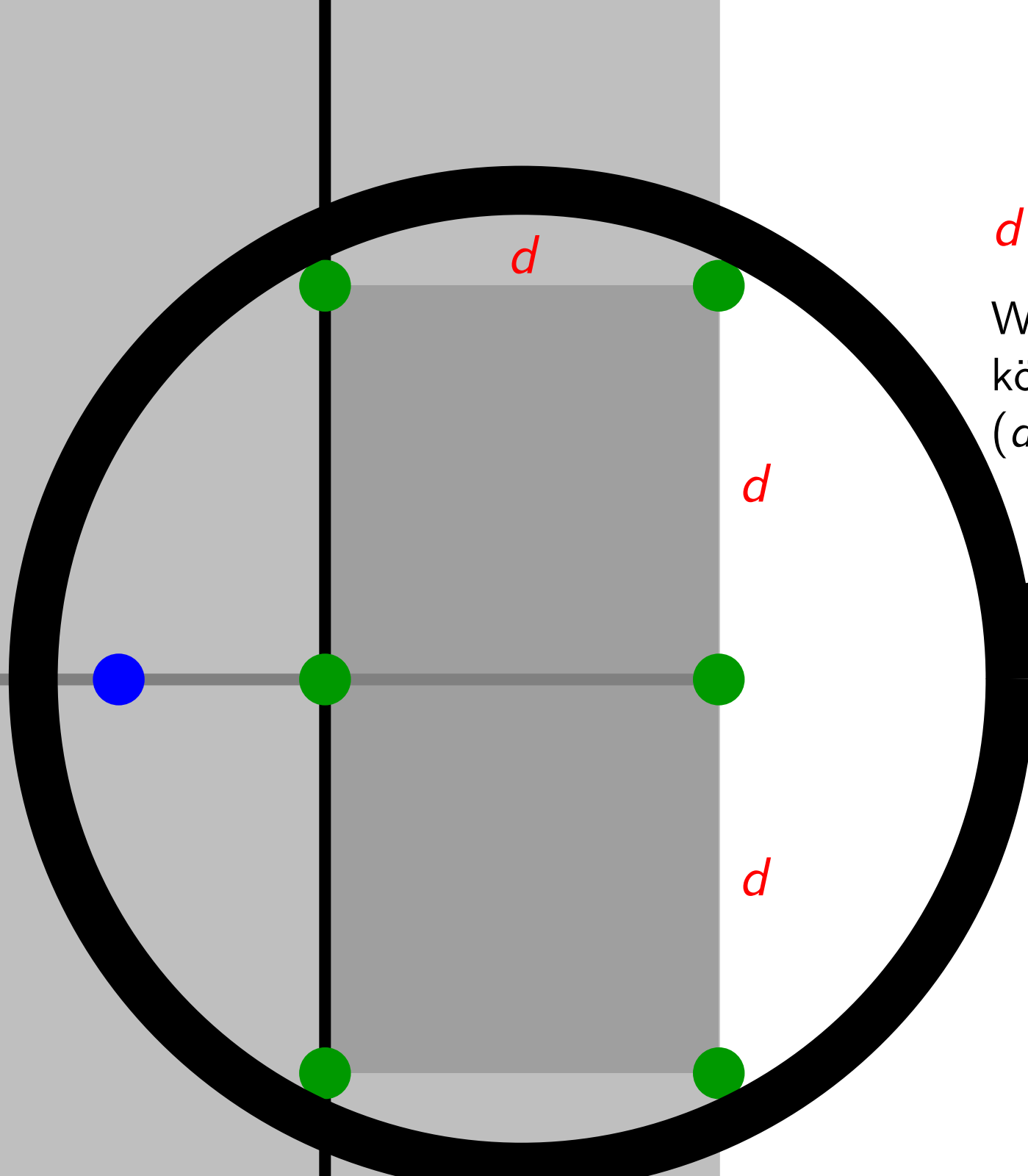




$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte  
können im dunkelgrauen  
( $d \times 2d$ )-Rechteck liegen?

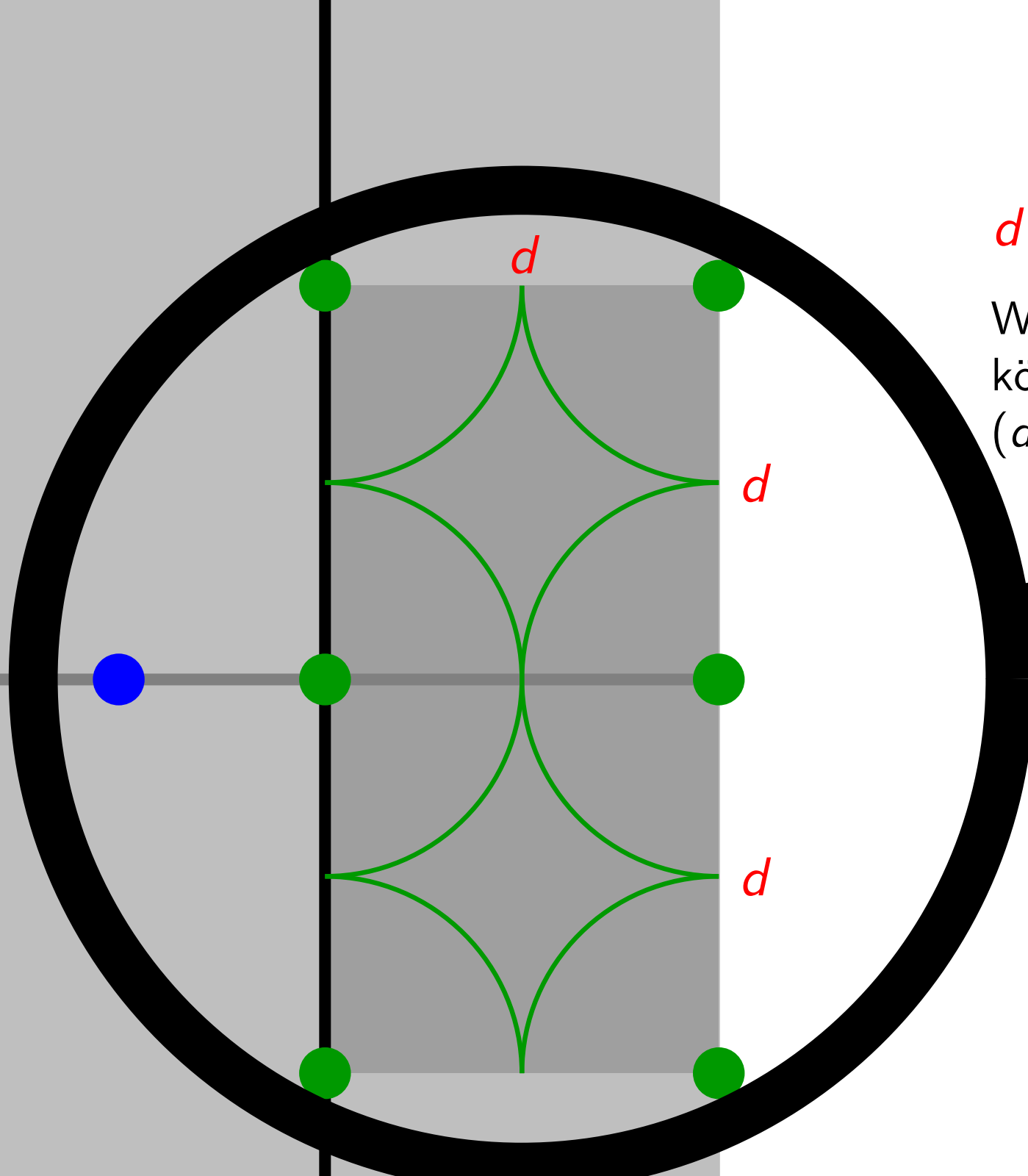
*Packungsargument:*



$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

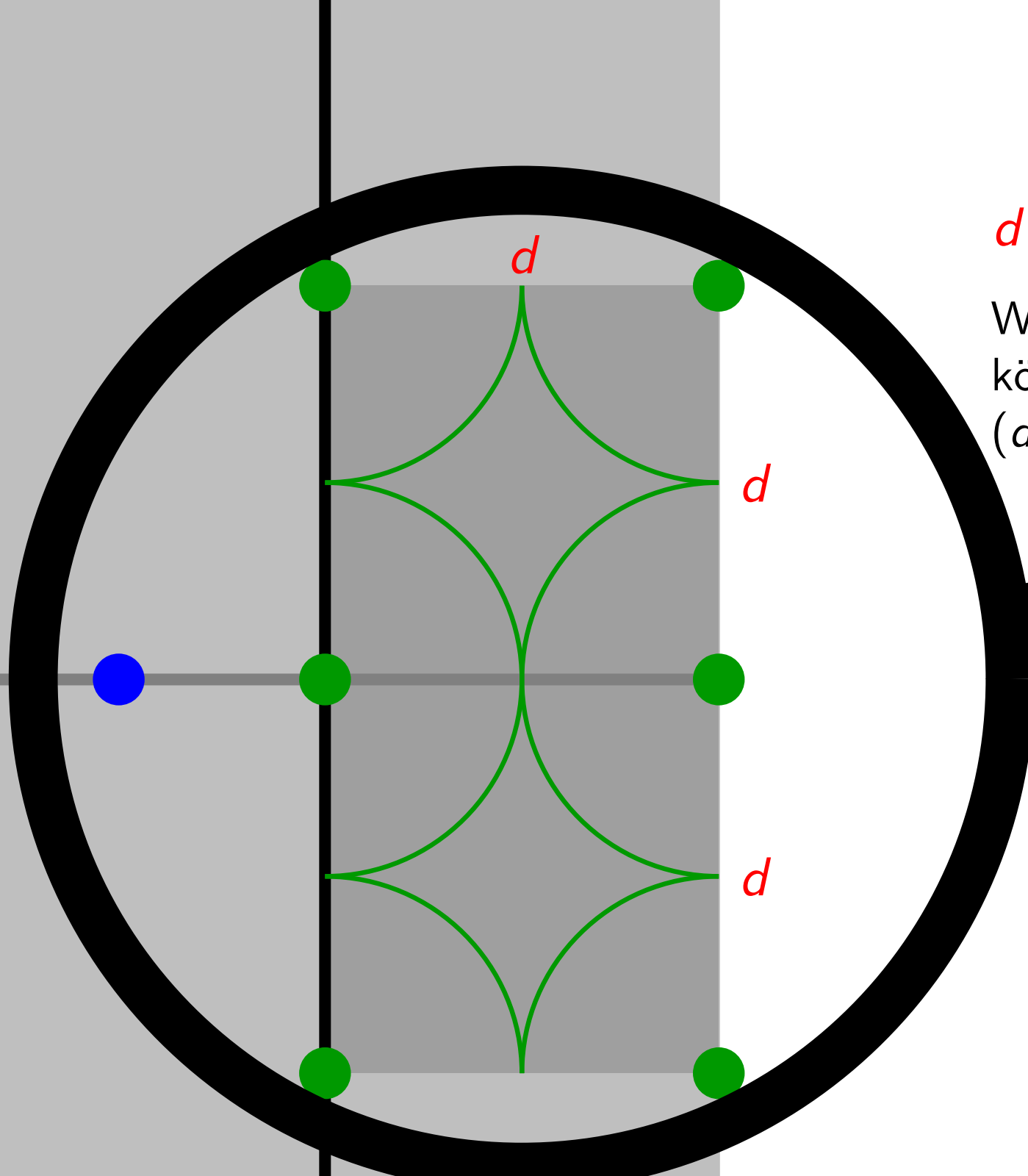
*Packungsargument:*  
maximal 6!



$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!

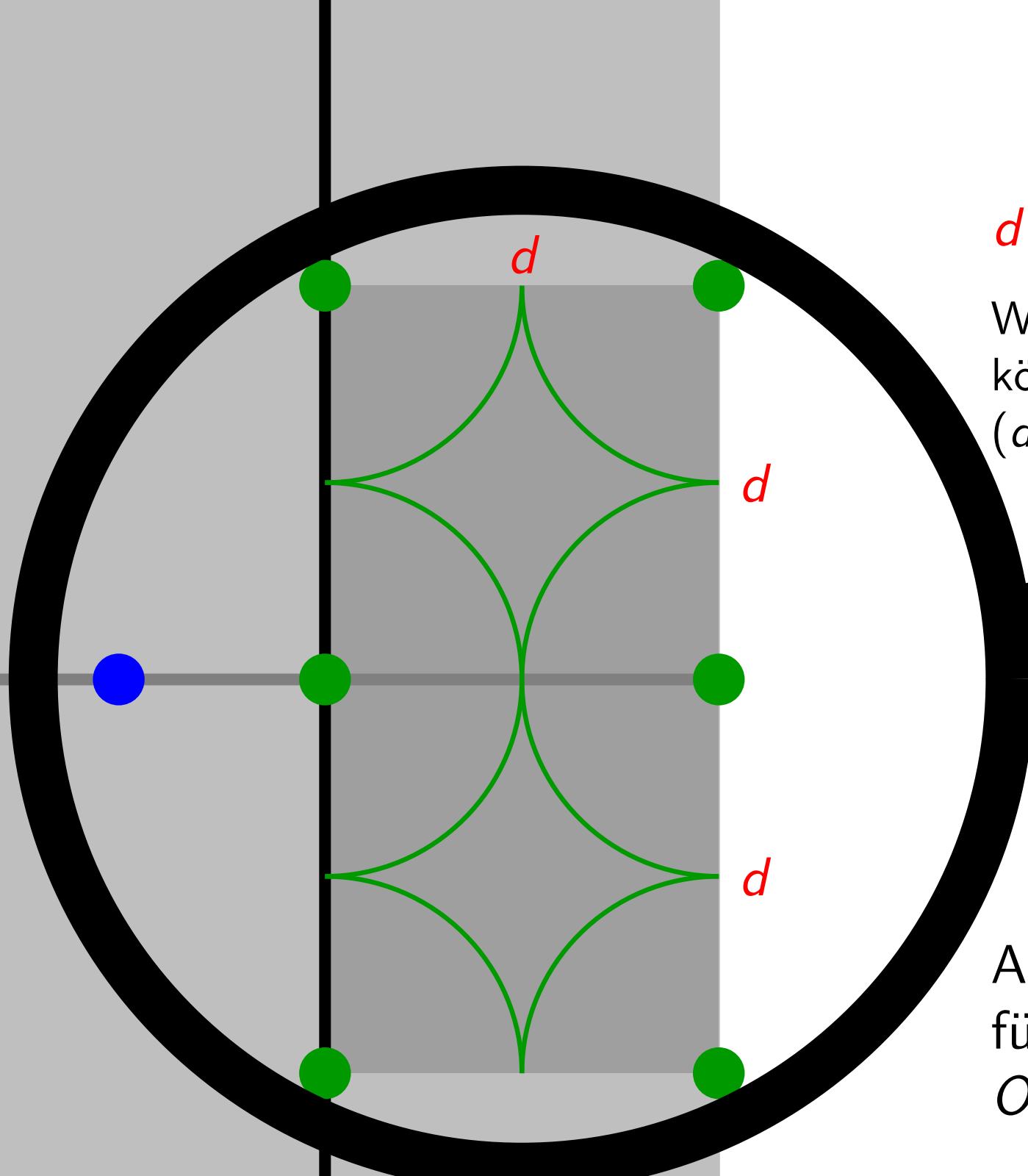


$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!





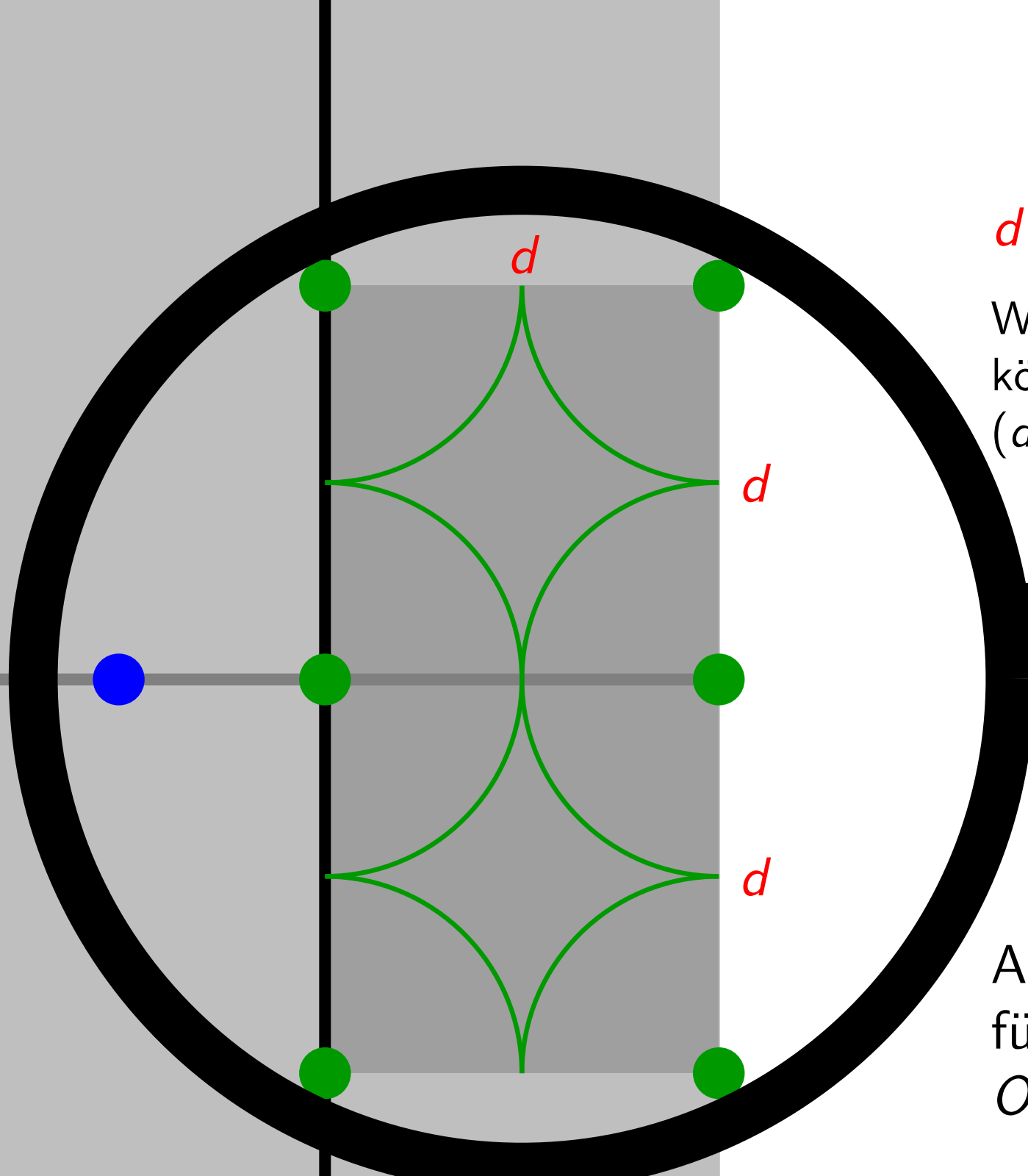
$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!



Anz.  $(\bullet, \bullet)$ -Kandidaten für das nächste Paar:  
 $O(\quad)$ .



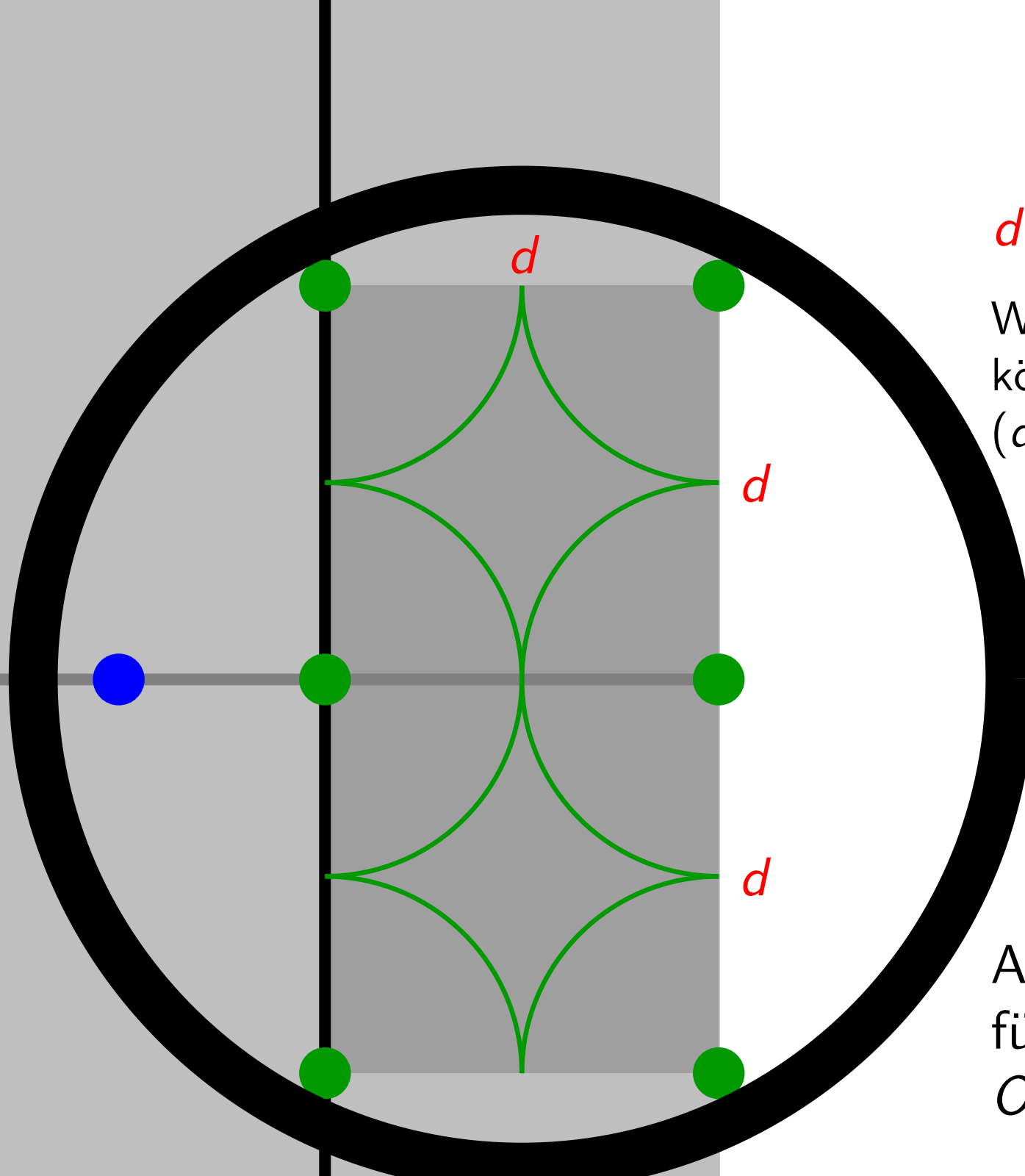
$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!



Anz.  $(\bullet, \bullet)$ -Kandidaten für das nächste Paar:  
 $O(n)$ .



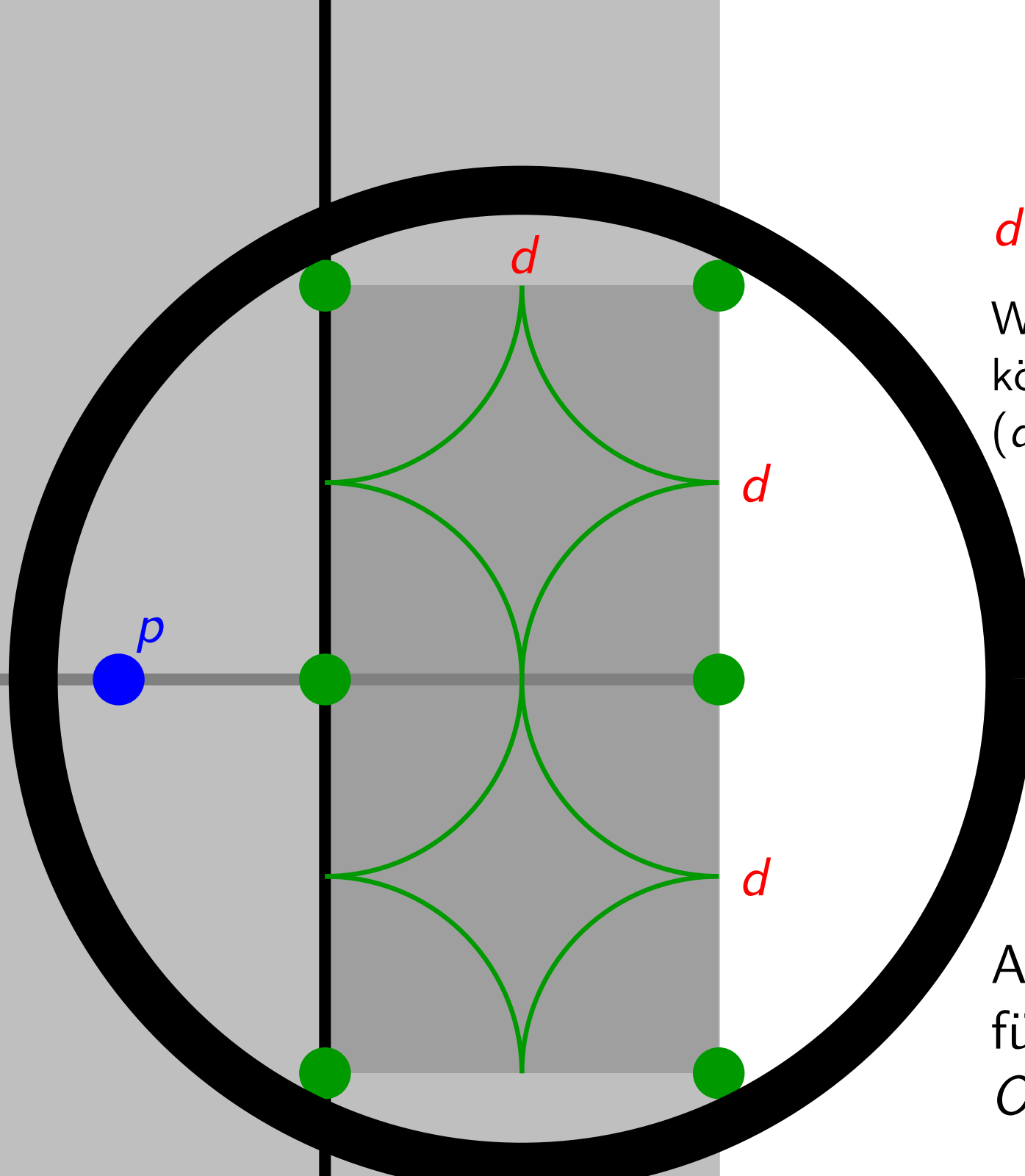
$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!



Anz.  $(\bullet, \bullet)$ -Kandidaten für das nächste Paar:  
 $O(n)$ . *Und finden?*



$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!



Anz.  $(\bullet, \bullet)$ -Kandidaten für das nächste Paar:  
 $O(n)$ . *Und finden?*



$$B_p = [x_{\text{med}}, x_{\text{med}} + d] \times [y_p - d, y_p + d]$$

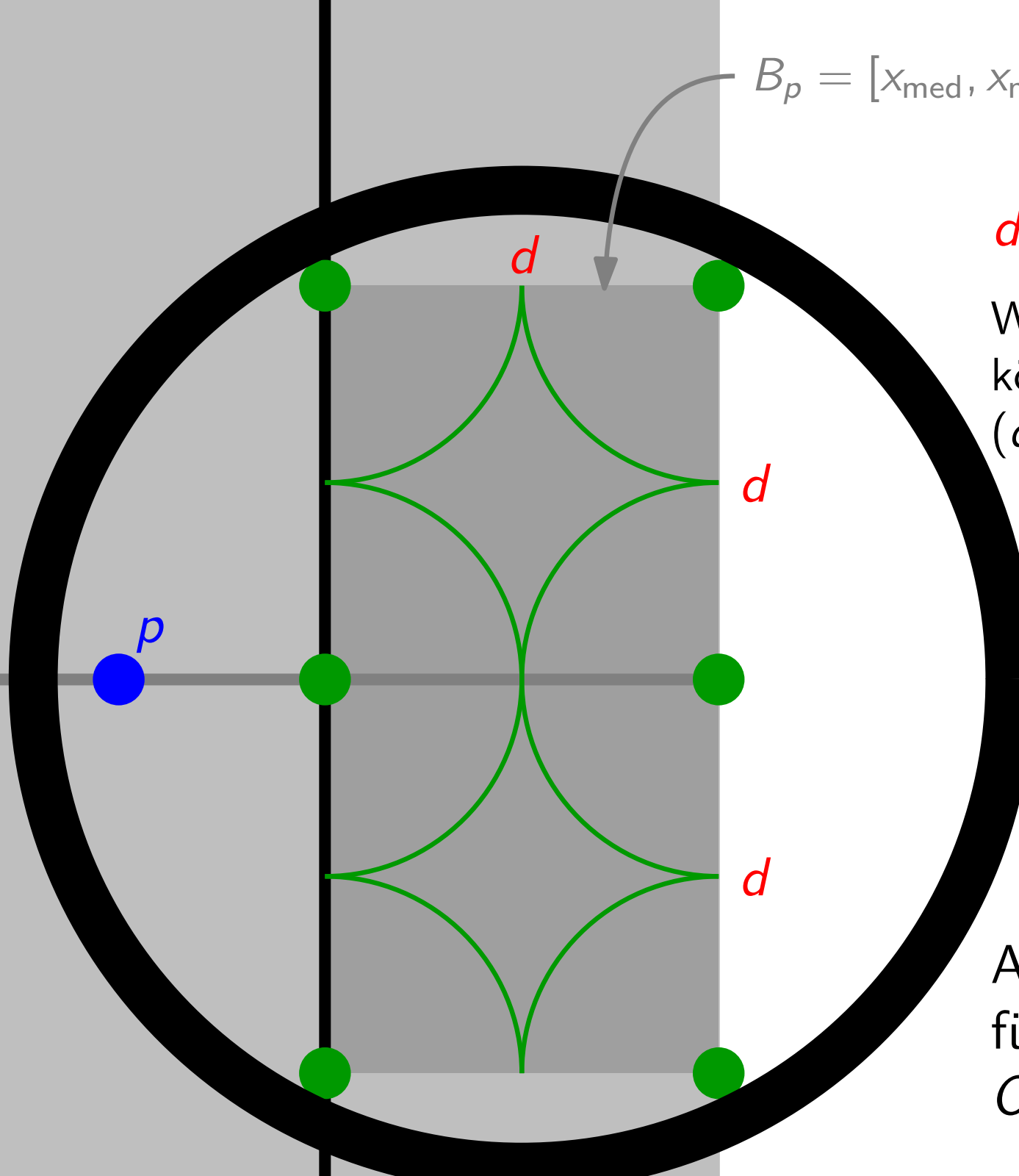
$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

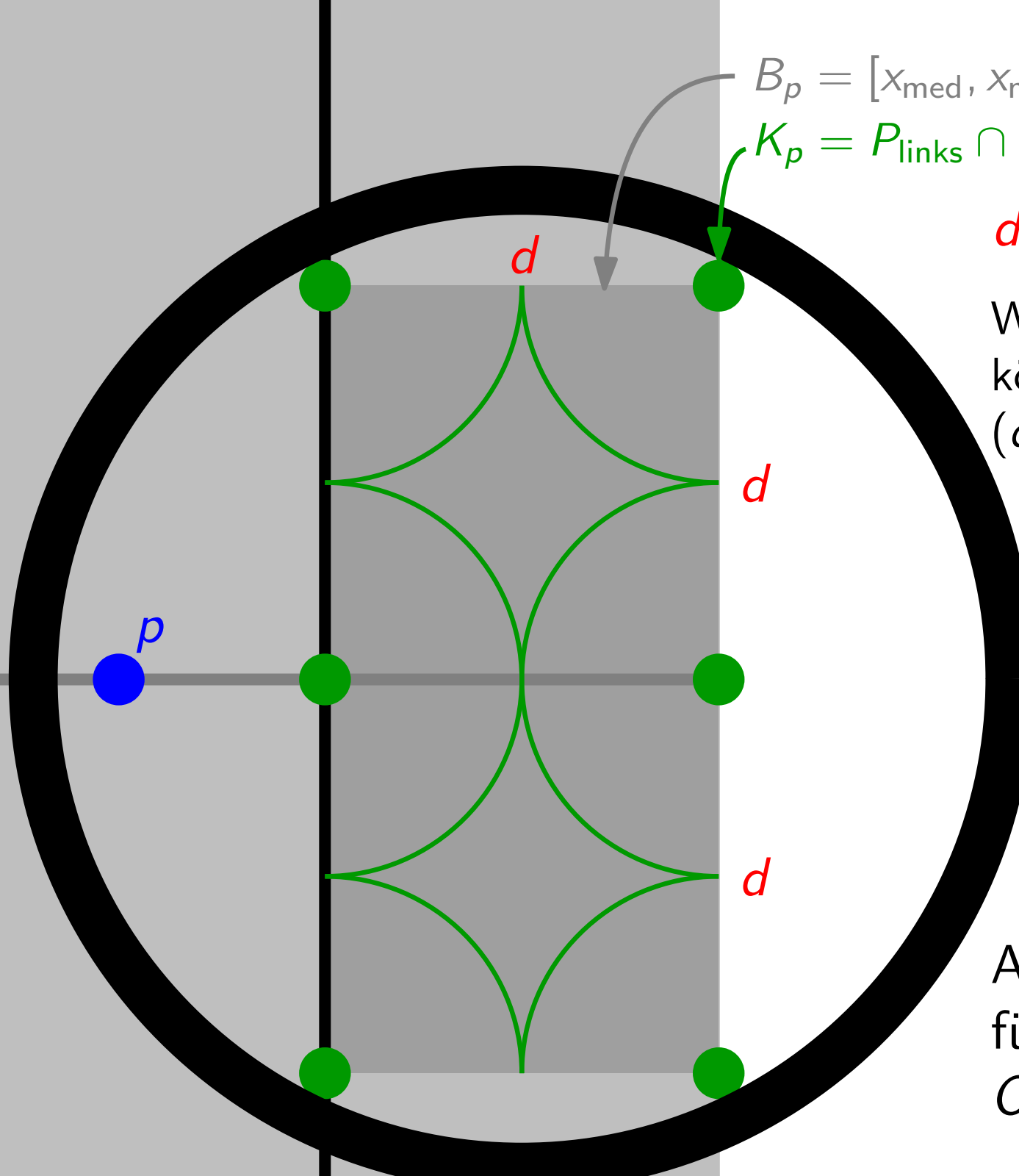
Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!



Anz.  $(\bullet, \bullet)$ -Kandidaten für das nächste Paar:  
 $O(n)$ . *Und finden?*





$$B_p = [x_{\text{med}}, x_{\text{med}} + d] \times [y_p - d, y_p + d]$$

$$K_p = P_{\text{links}} \cap B_p$$

$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte  
können im dunkelgrauen  
( $d \times 2d$ )-Rechteck liegen?

*Packungsargument:*  
maximal 6!



Anz.  $(\bullet, \bullet)$ -Kandidaten  
für das nächste Paar:  
 $O(n)$ . *Und finden?*

$$B_p = [x_{\text{med}}, x_{\text{med}} + d] \times [y_p - d, y_p + d]$$

$$K_p = P_{\text{links}} \cap B_p \Rightarrow |K_p| \leq$$

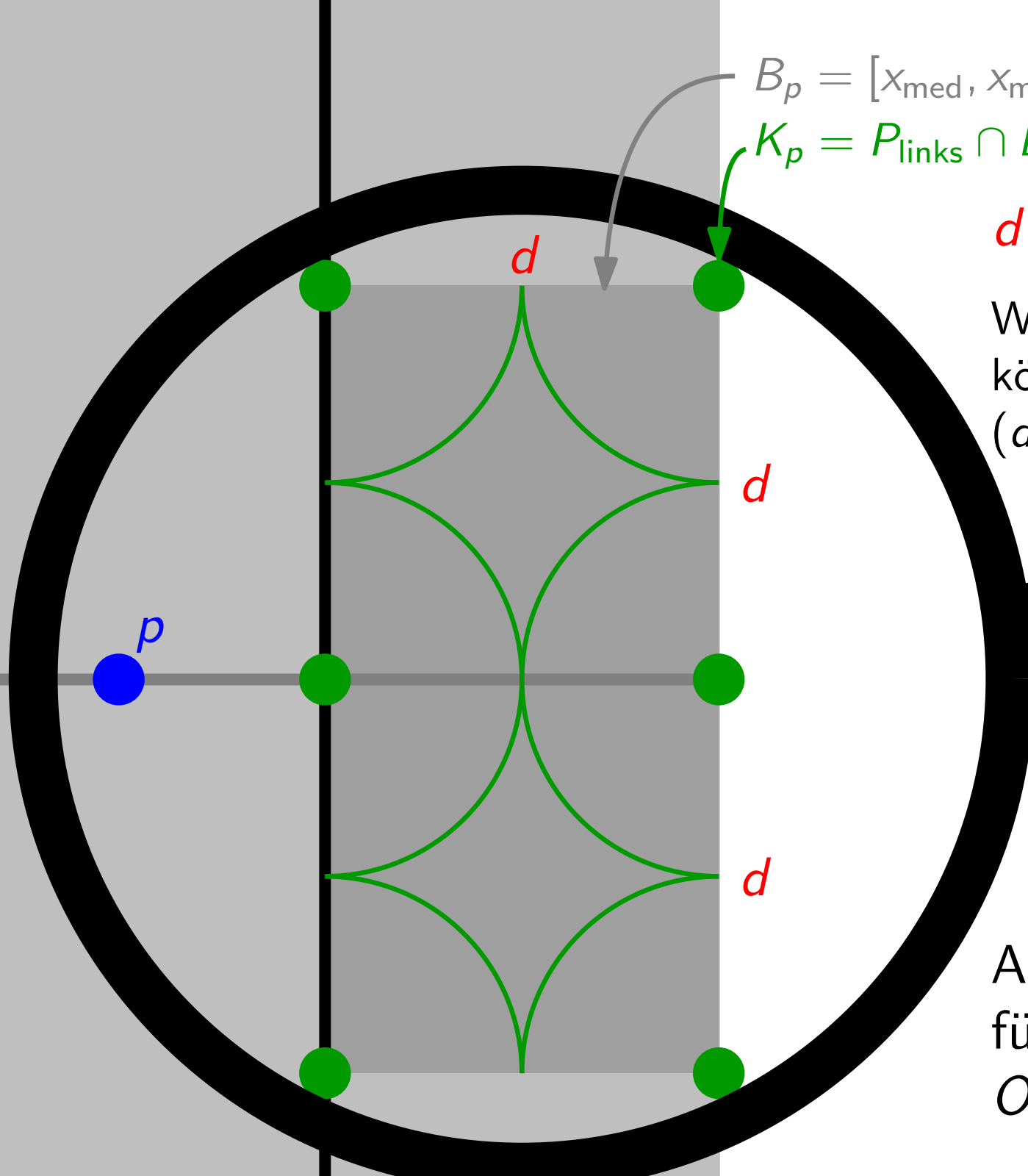
$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!



Anz.  $(\bullet, \bullet)$ -Kandidaten für das nächste Paar:  
 $O(n)$ . *Und finden?*



$$B_p = [x_{\text{med}}, x_{\text{med}} + d] \times [y_p - d, y_p + d]$$

$$K_p = P_{\text{links}} \cap B_p \Rightarrow |K_p| \leq 6$$

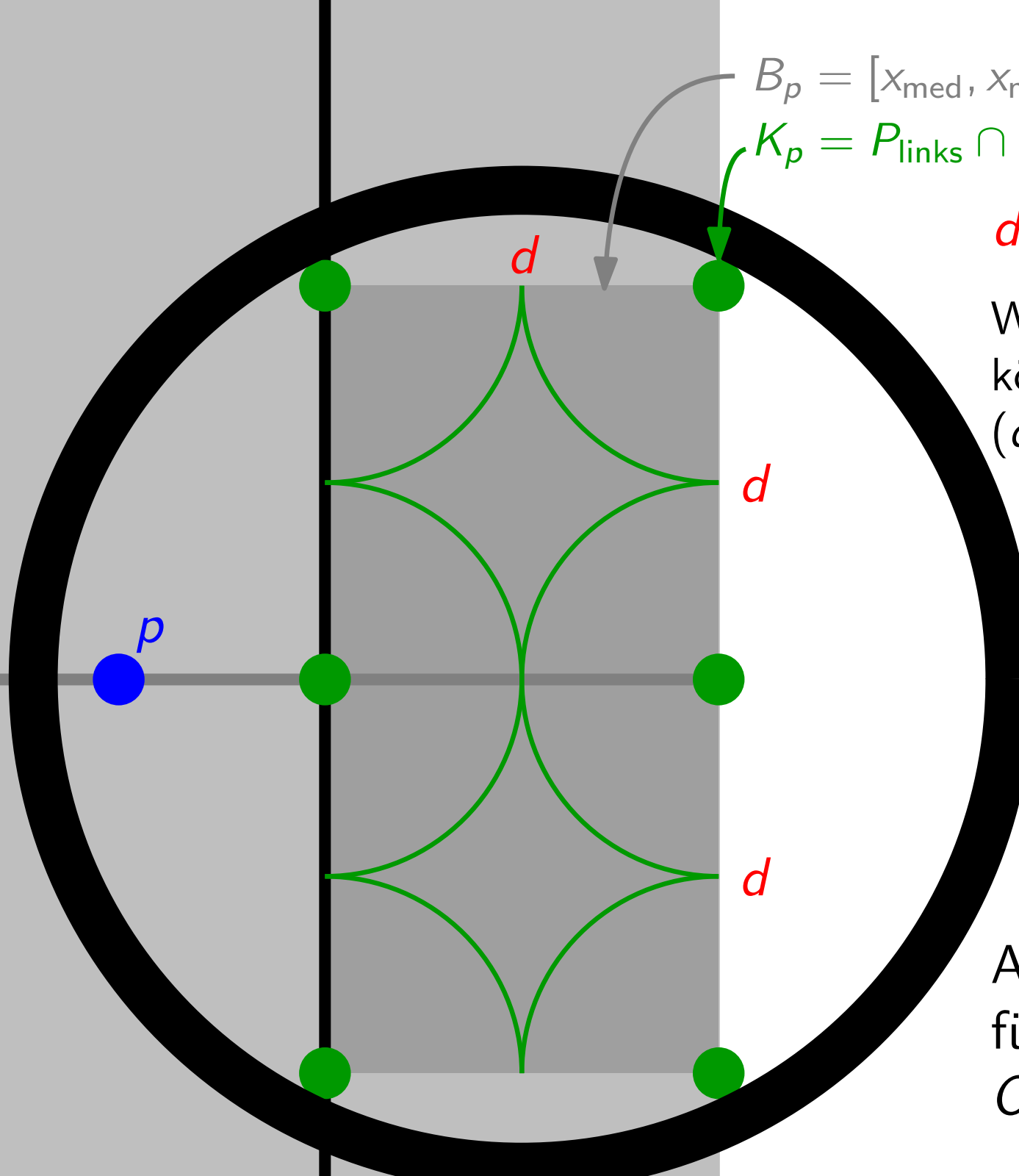
$$d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$$

Wieviele grüne Punkte können im dunkelgrauen  $(d \times 2d)$ -Rechteck liegen?

*Packungsargument:*  
maximal 6!



Anz.  $(\bullet, \bullet)$ -Kandidaten für das nächste Paar:  
 $O(n)$ . *Und finden?*



# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:  
bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$



# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:  
bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:  
bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$
4. Kombiniere:

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:  
bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$
4. Kombiniere:
  - $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:  
bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$
4. Kombiniere:
  - $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
  - Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:  
bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$
4. Kombiniere:
  - $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
  - Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
  - Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entsprech.)

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:  
bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$
4. Kombiniere:
  - $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
  - Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
  - Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entsprach.)  
Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:  
bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$
4. Kombiniere:
  - $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
  - Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
  - Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entsprach.)  
Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
  - Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .

# Algorithmus

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$
2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$
3. Herrsche:
 

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$
4. Kombiniere:
  - $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
  - Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
  - Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
  - Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
  - Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.



Algorithmus  $T(n) =$   $\left\{ \begin{array}{l} \text{Laufzeit des rekursiven Teils,} \\ \text{d.h. ohne Vorverarbeitung (1.)} \end{array} \right.$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

# Algorithmus $T(n) = T(\lfloor n/2 \rfloor)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

# Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

# Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

# Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$   $O(1)$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

# Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$   $O(1)$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

# Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$   $O(1)$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate  $O(n \log n)$
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

# Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$   $O(1)$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate  $O(n \log n)$
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr. h.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.



# Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$   $O(1)$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate  $O(n \log n)$
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr. h.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

$O(n)$

# Algorithmus $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$ ,  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$   $O(1)$
- Sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate  $O(n \log n)$
- Seien  $P_{\text{links}}^=$  die Punkte im grauen Streifen in  $P_{\text{links}}$ . ( $P_{\text{rechts}}^=$  entspr. h.)  
 Für jeden Punkt  $p$  in  $P_{\text{links}}^=$  gehe in  $P_{\text{rechts}}^=$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ ).
- Bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}^=$  und  $q \in K_p$ .
- Gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück.

$O(n)$

# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also  $T(n) \approx 2T(n/2) + O(n \log n)$

# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also  $T(n) \approx 2T(n/2) + O(n \log n)$

Rekursionsgleichung mit Master-Theorem lösen?

# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also  $T(n) \approx 2T(n/2) + O(n \log n)$

Rekursionsgleichung mit Master-Theorem lösen?

Bestimme Parameter für das Theorem:

$$a = b = 2, f(n) = O(n \log n).$$

# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also  $T(n) \approx 2T(n/2) + O(n \log n)$

Rekursionsgleichung mit Master-Theorem lösen?

Bestimme Parameter für das Theorem:

$$a = b = 2, f(n) = O(n \log n).$$

Betrachte  $n^{\log_b a} = n^{\log_2 2} = n^1$ .

# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also  $T(n) \approx 2T(n/2) + O(n \log n)$

Rekursionsgleichung mit Master-Theorem lösen?

Bestimme Parameter für das Theorem:

$$a = b = 2, f(n) = O(n \log n).$$

Betrachte  $n^{\log_b a} = n^{\log_2 2} = n^1$ .

$$\text{Gilt } f \in \left\{ \begin{array}{l} O(n^{1-\varepsilon}) \quad \text{für ein } \varepsilon > 0 \\ \Theta(n^1) \\ \Omega(n^{1+\varepsilon}) \quad \text{für ein } \varepsilon > 0 \end{array} \right\} ?$$



# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also  $T(n) \approx 2T(n/2) + O(n \log n)$

Rekursionsgleichung mit Master-Theorem lösen?

Bestimme Parameter für das Theorem:

$$a = b = 2, f(n) = O(n \log n).$$

Betrachte  $n^{\log_b a} = n^{\log_2 2} = n^1$ .

$$\text{Gilt } f \in \left\{ \begin{array}{l} O(n^{1-\varepsilon}) \quad \text{für ein } \varepsilon > 0 \\ \Theta(n^1) \\ \Omega(n^{1+\varepsilon}) \quad \text{für ein } \varepsilon > 0 \end{array} \right\} ?$$

Nein,  $f: n \mapsto O(n \log n)$  passt in keinen der drei Fälle.



# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also  $T(n) \approx 2T(n/2) + O(n \log n)$

Rekursionsgleichung mit Master-Theorem lösen?

Bestimme Parameter für das Theorem:

$$a = b = 2, f(n) = O(n \log n).$$

Betrachte  $n^{\log_b a} = n^{\log_2 2} = n^1$ .

$$\text{Gilt } f \in \left\{ \begin{array}{l} O(n^{1-\varepsilon}) \quad \text{für ein } \varepsilon > 0 \\ \Theta(n^1) \\ \Omega(n^{1+\varepsilon}) \quad \text{für ein } \varepsilon > 0 \end{array} \right\} ?$$

Nein,  $f: n \mapsto O(n \log n)$  passt in keinen der drei Fälle.



Die Rekursionsbaummethode liefert...

# Laufzeit

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + O(n \log n)$$

Also  $T(n) \approx 2T(n/2) + O(n \log n)$

Rekursionsgleichung mit Master-Theorem lösen?

Bestimme Parameter für das Theorem:

$$a = b = 2, f(n) = O(n \log n).$$

Betrachte  $n^{\log_b a} = n^{\log_2 2} = n^1$ .

$$\text{Gilt } f \in \left\{ \begin{array}{l} O(n^{1-\varepsilon}) \quad \text{für ein } \varepsilon > 0 \\ \Theta(n^1) \\ \Omega(n^{1+\varepsilon}) \quad \text{für ein } \varepsilon > 0 \end{array} \right\} ?$$

Nein,  $f: n \mapsto O(n \log n)$  passt in keinen der drei Fälle.



Die Rekursionsbaummethode liefert...  $T(n) = O(n \log^2 n)$ .

Noch besser?  $T(n) \approx 2T(n/2) + O(n \log n) = O(n \log^2 n)$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P$  in  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$  und  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$

- sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate  $O(n \log n)$

$O(n)$  {

- gehe „gleichzeitig“ durch  $P_{\text{links}}$  und  $P_{\text{rechts}}$ :  
für jeden Punkt  $p$  in  $P_{\text{links}}$  gehe in  $P_{\text{rechts}}$  bis y-Koord.  $y_p + d$ ;  
halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ )
- bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}$  und  $q \in K_p$
- gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück

Noch besser?

$$T(n) \approx 2T(n/2) + O(n \log n) = O(n \log^2 n)$$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P$  in  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$  und  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$

- sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate  $O(n \log n)$

$O(n)$  {

- gehe „gleichzeitig“ durch  $P_{\text{links}}$  und  $P_{\text{rechts}}$ :  
für jeden Punkt  $p$  in  $P_{\text{links}}$  gehe in  $P_{\text{rechts}}$  bis y-Koord.  $y_p + d$ ;  
halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ )
- bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}$  und  $q \in K_p$
- gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück

Noch besser?

$$T(n) \approx 2T(n/2) + O(n \log n) = O(n \log^2 n)$$

1. Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$

2. Teile:  $P$  in  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$  und  $P_{\text{rechts}} = P \setminus P_{\text{links}}$

3. Herrsche:

?! bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

4. Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$

- sortiere  $P_{\text{links}}$  und  $P_{\text{rechts}}$  nach y-Koordinate  $O(n \log n)$

$O(n)$  {

- gehe „gleichzeitig“ durch  $P_{\text{links}}$  und  $P_{\text{rechts}}$ :  
 für jeden Punkt  $p$  in  $P_{\text{links}}$  gehe in  $P_{\text{rechts}}$  bis y-Koord.  $y_p + d$ ;  
 halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ )
- bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P_{\text{links}}$  und  $q \in K_p$
- gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück

Noch besser!

$$T(n) \approx 2T(n/2) + O(n \log n) = O(n \log^2 n)$$

- Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$   
u.  $P' = P$  nach y-Koordinate  $\rightarrow p'_1, \dots, p'_n$  mit  $y'_1 \leq \dots \leq y'_n$

- Teile:  $P$  in  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$  und  $P_{\text{rechts}} = P \setminus P_{\text{links}}$   
 $P'$  in  $P'_{\text{links}}$  und  $P'_{\text{rechts}}$  (sortiert nach y-Koordinate)

- Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

- Kombiniere:

- $O(n)$  {
- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
  - gehe „gleichzeitig“ durch  $P'_{\text{links}}$  und  $P'_{\text{rechts}}$ :  
für jeden Punkt  $p$  in  $P'_{\text{links}}$  gehe in  $P'_{\text{rechts}}$  bis y-Koord.  $y_p + d$ ;  
halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ )
  - bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P'_{\text{links}}$  und  $q \in K_p$
  - gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück

Noch besser!

$$T(n) \approx 2T(n/2) + O(n \log n) = O(n \log n)$$

- Sortiere  $P$  nach x-Koordinate  $\rightarrow p_1, \dots, p_n$  mit  $x_1 \leq \dots \leq x_n$   
u.  $P' = P$  nach y-Koordinate  $\rightarrow p'_1, \dots, p'_n$  mit  $y_1 \leq \dots \leq y_n$

- Teile:  $P$  in  $P_{\text{links}} = \{p_1, \dots, p_{\lfloor n/2 \rfloor}\}$  und  $P_{\text{rechts}} = P \setminus P_{\text{links}}$   
 $P'$  in  $P'_{\text{links}}$  und  $P'_{\text{rechts}}$  (sortiert nach y-Koordinate)

- Herrsche:

bestimme rekursiv kleinsten Abstand  $d_{\text{links}}$  v. Paaren in  $P_{\text{links}}$   
 $d_{\text{rechts}}$   $P_{\text{rechts}}$

- Kombiniere:

- $d = \min\{d_{\text{links}}, d_{\text{rechts}}\}$
- gehe „gleichzeitig“ durch  $P'_{\text{links}}$  und  $P'_{\text{rechts}}$ :  
für jeden Punkt  $p$  in  $P'_{\text{links}}$  gehe in  $P'_{\text{rechts}}$  bis y-Koord.  $y_p + d$ ;  
halte die letzten 6 Punkte im grauen Streifen aufrecht ( $\rightarrow K_p$ )
- bestimme Min.  $d_{\text{mitte}}$  über alle  $d(p, q)$  mit  $p \in P'_{\text{links}}$  und  $q \in K_p$
- gib Min. von  $d_{\text{mitte}}$ ,  $d_{\text{links}}$  und  $d_{\text{rechts}}$  (und entspr. Paar) zurück

$O(n)$



# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)       $O(n \log n)$

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)       $O(n \log n)$

2. Teilen

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$
2. Teilen  $O(n)$

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$
2. Teilen  $O(n)$
3. Herrschen

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)       $O(n \log n)$
2. Teilen                       $O(n)$
3. Herrschen                 $2T(n/2)$

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$
2. Teilen  $O(n)$
3. Herrschen  $2T(n/2)$
4. Kombinieren

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$
2. Teilen  $O(n)$
3. Herrschen  $2T(n/2)$
4. Kombinieren  $O(n)$



# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$

2. Teilen  $O(n)$

3. Herrschen  $2T(n/2)$

4. Kombinieren  $O(n)$

}  $T(n) =$

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$

2. Teilen  $O(n)$

3. Herrschen  $2T(n/2)$

4. Kombinieren  $O(n)$

$$\left. \begin{array}{l} 2. \text{ Teilen } O(n) \\ 3. \text{ Herrschen } 2T(n/2) \\ 4. \text{ Kombinieren } O(n) \end{array} \right\} T(n) = O(n \log n) \quad [\text{MergeSort-Rek.!}]$$

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$

2. Teilen  $O(n)$

3. Herrschen  $2T(n/2)$

4. Kombinieren  $O(n)$

$$\left. \begin{array}{l} 2. \text{ Teilen } O(n) \\ 3. \text{ Herrschen } 2T(n/2) \\ 4. \text{ Kombinieren } O(n) \end{array} \right\} T(n) = O(n \log n) \quad [\text{MergeSort-Rek.!}]$$

---

Gesamtlaufzeit

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$

2. Teilen  $O(n)$

3. Herrschen  $2T(n/2)$

4. Kombinieren  $O(n)$



$T(n) = O(n \log n)$

[MergeSort-Rek.!]

---

Gesamtlaufzeit

$O(n \log n)$



# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$

2. Teilen  $O(n)$

3. Herrschen  $2T(n/2)$

4. Kombinieren  $O(n)$

}

$T(n) = O(n \log n)$

[MergeSort-Rek.!]

---

Gesamtlaufzeit

$O(n \log n)$



*Speicherplatzbedarf?*

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$

2. Teilen  $O(n)$

3. Herrschen  $2T(n/2)$

4. Kombinieren  $O(n)$

}

$T(n) = O(n \log n)$

[MergeSort-Rek.!]

---

Gesamtlaufzeit

$O(n \log n)$



*Speicherplatzbedarf?*

$O(n)$ ,

# Zusammenfassung

1. Vorverarbeitung ( $2 \times$  Sortieren)  $O(n \log n)$

2. Teilen  $O(n)$

3. Herrschen  $2T(n/2)$

4. Kombinieren  $O(n)$

}

$T(n) = O(n \log n)$

[MergeSort-Rek.!]

---

Gesamtlaufzeit

$O(n \log n)$



*Speicherplatzbedarf?*

$O(n)$ , wenn  $P'$  *in situ* in  $P'_{\text{links}}$  und  $P'_{\text{rechts}}$  zerlegt wird.

# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?



# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen, kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller als in  $\Omega(n \log n)$  Zeit gelöst werden – wenn man als Rechenmodell das sogenannte *algebraische Entscheidungsbaummodell* zugrunde legt.

# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
als in  $\Omega(n \log n)$  Zeit gelöst werden –  
wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
als in  $\Omega(n \log n)$  Zeit gelöst werden –  
wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
Zeit lösen –

# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
als in  $\Omega(n \log n)$  Zeit gelöst werden –  
wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
Zeit lösen – dann auch Element Uniqueness! ⚡

# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
als in  $\Omega(n \log n)$  Zeit gelöst werden –  
wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
Zeit lösen – dann auch Element Uniqueness! ⚡

Wie?

# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
 Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
 kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
 als in  $\Omega(n \log n)$  Zeit gelöst werden –  
 wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
 Zeit lösen – dann auch Element Uniqueness! ⚡

Wie? Teste, ob das nächste Paar Abstand 0 hat!

# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
 Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
 kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
 als in  $\Omega(n \log n)$  Zeit gelöst werden –  
 wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
 Zeit lösen – dann auch Element Uniqueness! ⚡

**Wie?** Teste, ob das nächste Paar Abstand 0 hat!

Genaugenommen muss man die Zahlen  $a_1, \dots, a_n$  in eine Menge von (paarweise verschiedenen!)  
 Punkten der Ebene transformieren, aber auch das geht! – Wie?

# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
 Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
 kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

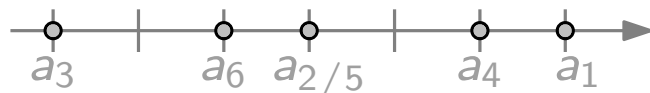
**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
 als in  $\Omega(n \log n)$  Zeit gelöst werden –  
 wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
 Zeit lösen – dann auch Element Uniqueness! ⚡

**Wie?** Teste, ob das nächste Paar Abstand 0 hat!

Genaugenommen muss man die Zahlen  $a_1, \dots, a_n$  in eine Menge von (paarweise verschiedenen!)  
 Punkten der Ebene transformieren, aber auch das geht! – Wie?





# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
 Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
 kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
 als in  $\Omega(n \log n)$  Zeit gelöst werden –  
 wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
 Zeit lösen – dann auch Element Uniqueness! ⚡

**Wie?** Teste, ob das nächste Paar Abstand 0 hat!

Genaugenommen muss man die Zahlen  $a_1, \dots, a_n$  in eine Menge von (paarweise verschiedenen!)  
 Punkten der Ebene transformieren, aber auch das geht! – Wie?



# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
 Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
 kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
 als in  $\Omega(n \log n)$  Zeit gelöst werden –  
 wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
 Zeit lösen – dann auch Element Uniqueness! ⚡

**Wie?** Teste, ob das nächste Paar Abstand 0 hat!

Genaugenommen muss man die Zahlen  $a_1, \dots, a_n$  in eine Menge von (paarweise verschiedenen!)  
 Punkten der Ebene transformieren, aber auch das geht! – Wie?



# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
 Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
 kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
 als in  $\Omega(n \log n)$  Zeit gelöst werden –  
 wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
 Zeit lösen – dann auch Element Uniqueness! ⚡

**Wie?** Teste, ob das nächste Paar Abstand 0 hat!

Genaugenommen muss man die Zahlen  $a_1, \dots, a_n$  in eine Menge von (paarweise verschiedenen!)  
 Punkten der Ebene transformieren, aber auch das geht! – Wie?



# Ist die Laufzeit $O(n \log n)$ optimal?

**Def.** *Element-Uniqueness-Problem (für natürliche Zahlen)*  
 Gegeben eine Folge  $a_1, \dots, a_n$  von  $n$  Zahlen,  
 kommt jede Zahl nur einmal vor, d.h.  $a_i \neq a_j$  für  $i \neq j$ ?

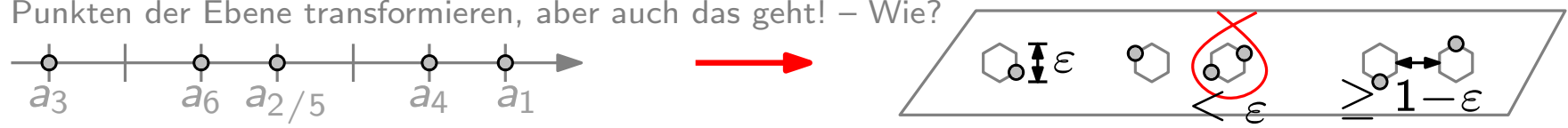
**Satz.** Das Element-Uniqueness-Problem kann nicht schneller  
 als in  $\Omega(n \log n)$  Zeit gelöst werden –  
 wenn man als Rechenmodell das sogenannte  
*algebraische Entscheidungsbaummodell* zugrunde legt.

Was bedeutet das für das Problem *Nächstes Paar*?

Angenommen wir könnten Nächstes Paar in  $o(n \log n)$   
 Zeit lösen – dann auch Element Uniqueness! ⚡

**Wie?** Teste, ob das nächste Paar Abstand 0 hat!

Genaugenommen muss man die Zahlen  $a_1, \dots, a_n$  in eine Menge von (paarweise verschiedenen!)  
 Punkten der Ebene transformieren, aber auch das geht! – Wie?



# Das heißt...

- Satz.** Das Problem Nächstes Paar kann nicht schneller als in  $\Omega(n \log n)$  Zeit gelöst werden, wenn man als Rechenmodell das algebraische Entscheidungsbaummodell zugrunde legt.
- Kor.** Unser  $O(n \log n)$ -Zeit-Algorithmus für das Problem Nächstes Paar ist asymptotisch optimal, wenn man....

Üben, üben, üben.

# Üben, üben, üben.

- Implementieren Sie die einfache Brute-Force-Lösung in Java.

# Üben, üben, üben.

- Implementieren Sie die einfache Brute-Force-Lösung in Java.
- Implementieren Sie einen einfachen Teile-und-Herrsche-Algorithmus, der im Herrsche-Schritt *alle* (quadratisch vielen) (●,●)-Kandidaten testet.



# Üben, üben, üben.

- Implementieren Sie die einfache Brute-Force-Lösung in Java.
- Implementieren Sie einen einfachen Teile-und-Herrsche-Algorithmus, der im Herrsche-Schritt *alle* (quadratisch vielen)  $(\bullet, \bullet)$ -Kandidaten testet. *(Ist der schneller als der Brute-Force-Alg.?)*

# Üben, üben, üben.

- Implementieren Sie die einfache Brute-Force-Lösung in Java.
- Implementieren Sie einen einfachen Teile-und-Herrsche-Algorithmus, der im Herrsche-Schritt *alle* (quadratisch vielen)  $(\bullet, \bullet)$ -Kandidaten testet. *(Ist der schneller als der Brute-Force-Alg.?)*
- Implementieren Sie den hier vorgestellten Teile-und-Herrsche-Algorithmus, der in  $O(n \log^2 n)$  Zeit läuft!

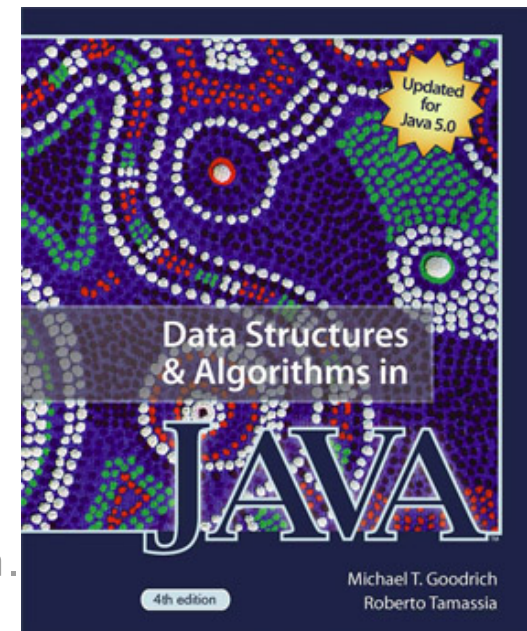
# Üben, üben, üben.

- Implementieren Sie die einfache Brute-Force-Lösung in Java.
- Implementieren Sie einen einfachen Teile-und-Herrsche-Algorithmus, der im Herrsche-Schritt *alle* (quadratisch vielen) (●,●)-Kandidaten testet. *(Ist der schneller als der Brute-Force-Alg.?)*
- Implementieren Sie den hier vorgestellten Teile-und-Herrsche-Algorithmus, der in  $O(n \log^2 n)$  Zeit läuft!
- Implementieren Sie den hier vorgestellten Teile-und-Herrsche-Algorithmus, der in  $O(n \log n)$  Zeit läuft!

# Üben, üben, üben.

- Implementieren Sie die einfache Brute-Force-Lösung in Java.
- Implementieren Sie einen einfachen Teile-und-Herrsche-Algorithmus, der im Herrsche-Schritt *alle* (quadratisch vielen) (•,•)-Kandidaten testet. *(Ist der schneller als der Brute-Force-Alg.?)*
- Implementieren Sie den hier vorgestellten Teile-und-Herrsche-Algorithmus, der in  $O(n \log^2 n)$  Zeit läuft!
- Implementieren Sie den hier vorgestellten Teile-und-Herrsche-Algorithmus, der in  $O(n \log n)$  Zeit läuft!

Goodrich & Tamassia:  
Data Structures & Algorithms in Java.  
Wiley, 4. Aufl., 2005 (5. Aufl., 2010)



# Algorithmen & Datenstrukturen

**Lernziele:** In dieser Veranstaltung haben Sie schon gelernt...

# Algorithmen & Datenstrukturen

**Lernziele:** In dieser Veranstaltung haben Sie schon gelernt...

- die Effizienz von Algorithmen zu **messen** und miteinander zu **vergleichen**,
- grundlegende Algorithmen und Datenstrukturen in Java zu **implementieren**,
- selbst Algorithmen und Datenstrukturen zu **entwerfen** sowie
- deren Korrektheit und Effizienz zu **beweisen**.

# Algorithmen & Datenstrukturen

**Lernziele:** In dieser Veranstaltung haben Sie schon gelernt...

- die Effizienz von Algorithmen zu **messen** und miteinander zu **vergleichen**,
- grundlegende Algorithmen und Datenstrukturen in Java zu **implementieren**,
- selbst Algorithmen und Datenstrukturen zu **entwerfen** sowie
- deren Korrektheit und Effizienz zu **beweisen**.

- Inhalt:**
- Grundlagen und Analysetechniken
  - Sortierverfahren
  - Java
  - Datenstrukturen
  - Graphenalgorithmen
  - Systematisches Probieren

# Algorithmen & Datenstrukturen

**Lernziele:** In dieser Veranstaltung haben Sie schon gelernt...

- die Effizienz von Algorithmen zu **messen** und miteinander zu **vergleichen**,
- grundlegende Algorithmen und Datenstrukturen in Java zu **implementieren**,
- selbst Algorithmen und Datenstrukturen zu **entwerfen** sowie
- deren Korrektheit und Effizienz zu **beweisen**.

**Inhalt:**

- Grundlagen und Analysetechniken
- Sortierverfahren
- Java
- Datenstrukturen

To do

- Graphenalgorithmen
- Systematisches Probieren



# Algorithmen & Datenstrukturen

**Lernziele:** In dieser Veranstaltung haben Sie schon gelernt...

- die Effizienz von Algorithmen zu **messen** und miteinander zu **vergleichen**,
- grundlegende Algorithmen und Datenstrukturen in Java zu **implementieren**,
- selbst Algorithmen und Datenstrukturen zu **entwerfen** sowie
- deren Korrektheit und Effizienz zu **beweisen**.

**Inhalt:**

- Grundlagen und Analysetechniken
- Sortierverfahren
- Java
- Datenstrukturen

To do

- Graphenalgorithmen (kürzeste Wege, min. Spannbäume)
- Systematisches Probieren (dynamisches Progr., Greedy-Alg.)