

# Übung Algorithmische Geometrie

## Lineare Programmierung + Bereichsanfragen

LEHRSTUHL FÜR ALGORITHMIK I · INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Andreas Gemsa  
24.05.2011



Übungsblatt 4

Nachtrag zu Übungsblatt 3

Übungsblatt 5

Werbung

# Aufgabe 1

## Korrektheitsbeweis:

a) Zeige, dass jede mögliche Permutation von  $A$  gleich wahrscheinlich ist.

RandomPermutation( $A$ )

**Input:** Array  $A[1 \dots n]$

**Output:** Array  $A$ , zufällig gleichverteilt permutiert

**for**  $k \leftarrow n$  **to** 2 **do**

$r \leftarrow \text{Random}(k)$   
    tausche  $A[r]$  und  $A[k]$

# Aufgabe 1

## Korrektheitsbeweis:

b) Zeige, dass die Aussage aus a) nicht stimmt, wenn wir in der 2. Zeile  $k$  mit  $n$  ersetzen.

RandomPermutation( $A$ )

**Input:** Array  $A[1 \dots n]$

**Output:** Array  $A$ , zufällig gleichverteilt permutiert

**for**  $k \leftarrow n$  **to** 2 **do**

$r \leftarrow \text{Random}(k)$   
    tausche  $A[r]$  und  $A[k]$

# Aufgabe 1

## Korrektheitsbeweis:

b) Zeige, dass die Aussage aus a) nicht stimmt, wenn wir in der 2. Zeile  $k$  mit  $n$  ersetzen.

RandomPermutation( $A$ )

**Input:** Array  $A[1 \dots n]$

**Output:** Array  $A$ , zufällig gleichverteilt permutiert ?

**for**  $k \leftarrow n$  **to** 2 **do**

$r \leftarrow \text{Random}(n)$   
    tausche  $A[r]$  und  $A[k]$

# Aufgabe 2

## ParanoidMax

**Input:** Endliche Menge  $A \subset \mathbb{R}$

**Output:** Maximum  $\max_{a \in A} a$  der Menge

**if**  $|A| = 1$  **then**

└ **return** einziges Element  $a \in A$

**else**

┌  $a =$  zufällig gewähltes Element aus  $A$

┌  $b =$  ParanoidMax( $A \setminus \{a\}$ )

┌ **if**  $b \geq a$  **then**

└ **return**  $b$

**else**

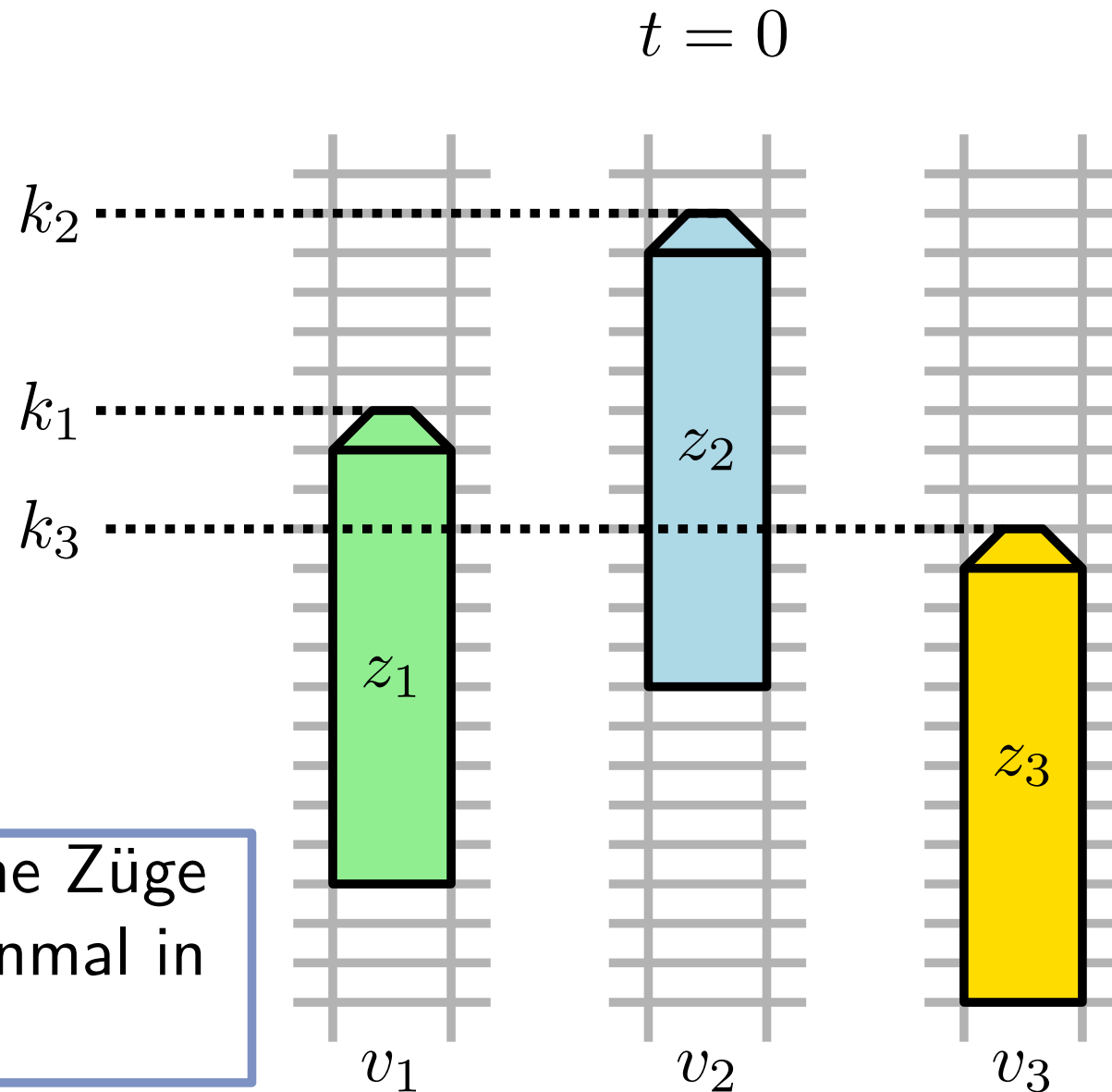
┌ prüfe unnötigerweise jedes Element aus  $A \setminus \{a\}$ , um sicherzugehen, dass  $a$  wirklich größer ist

└ **return**  $a$

a) Asymptotische  
w-c Laufzeit?

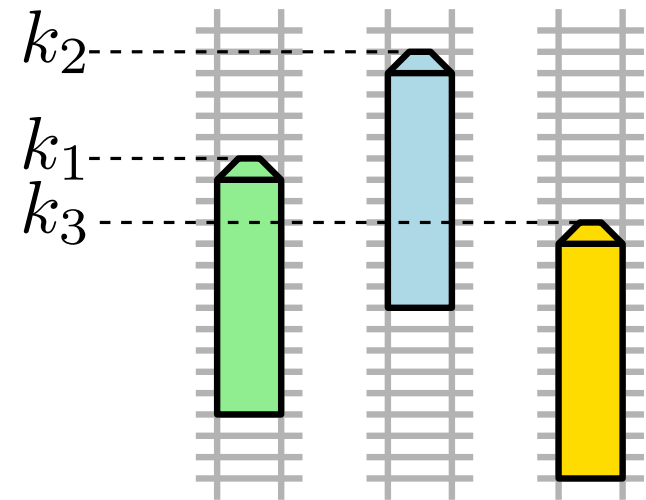
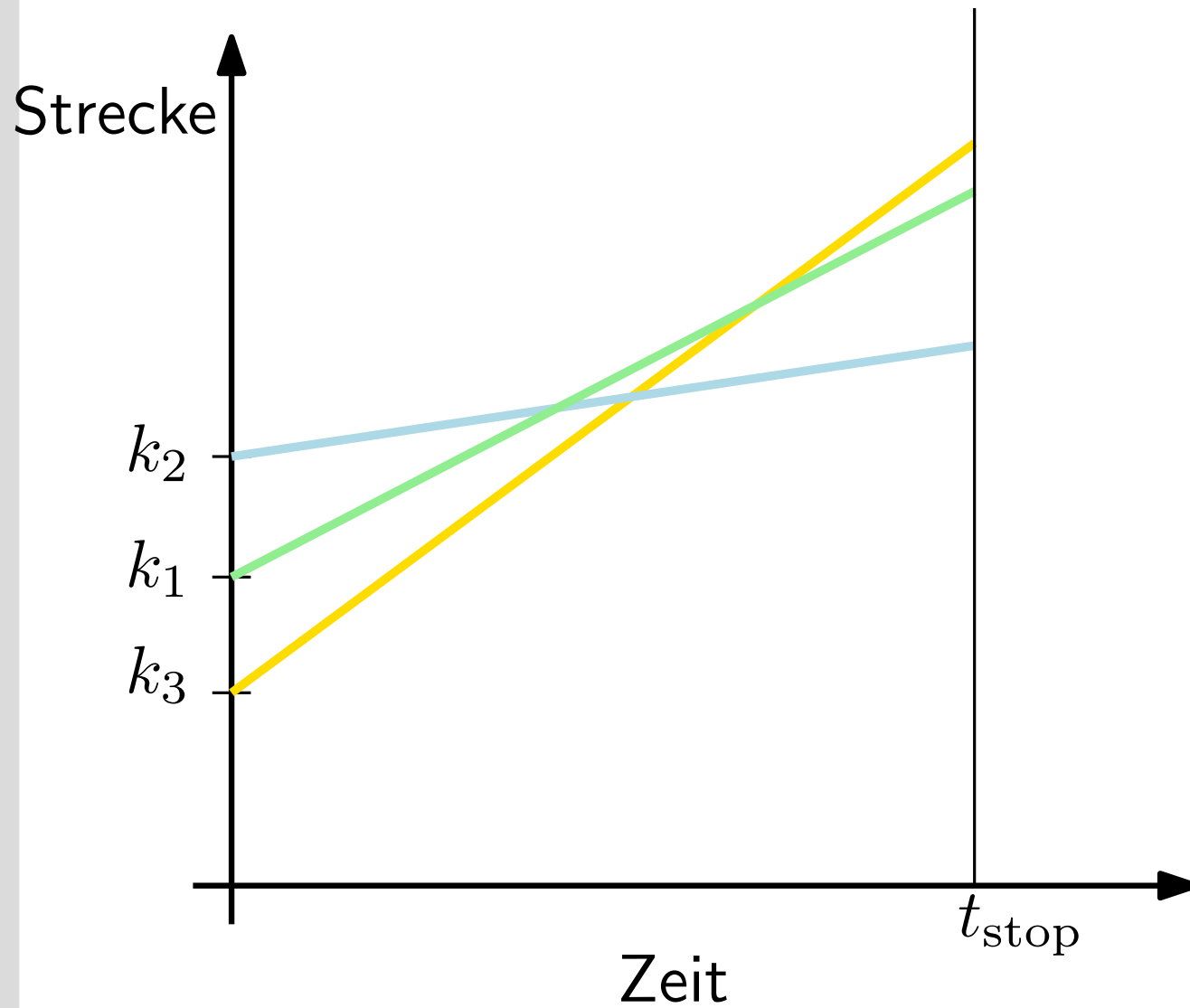
b) erwartete  
Laufzeit echt  
besser als w-c

# Aufgabe 3 : Züge



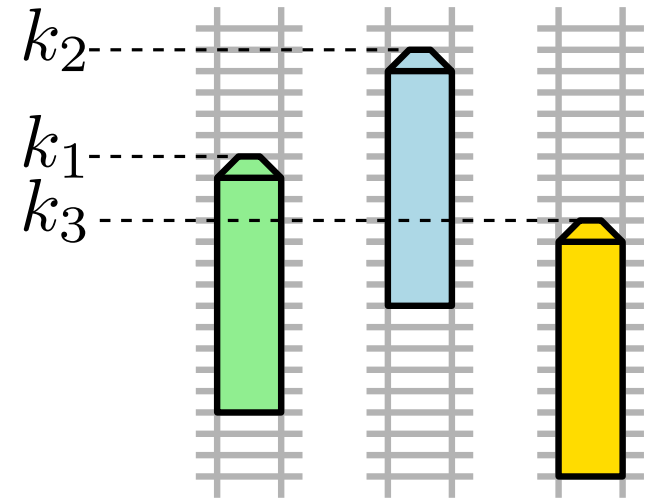
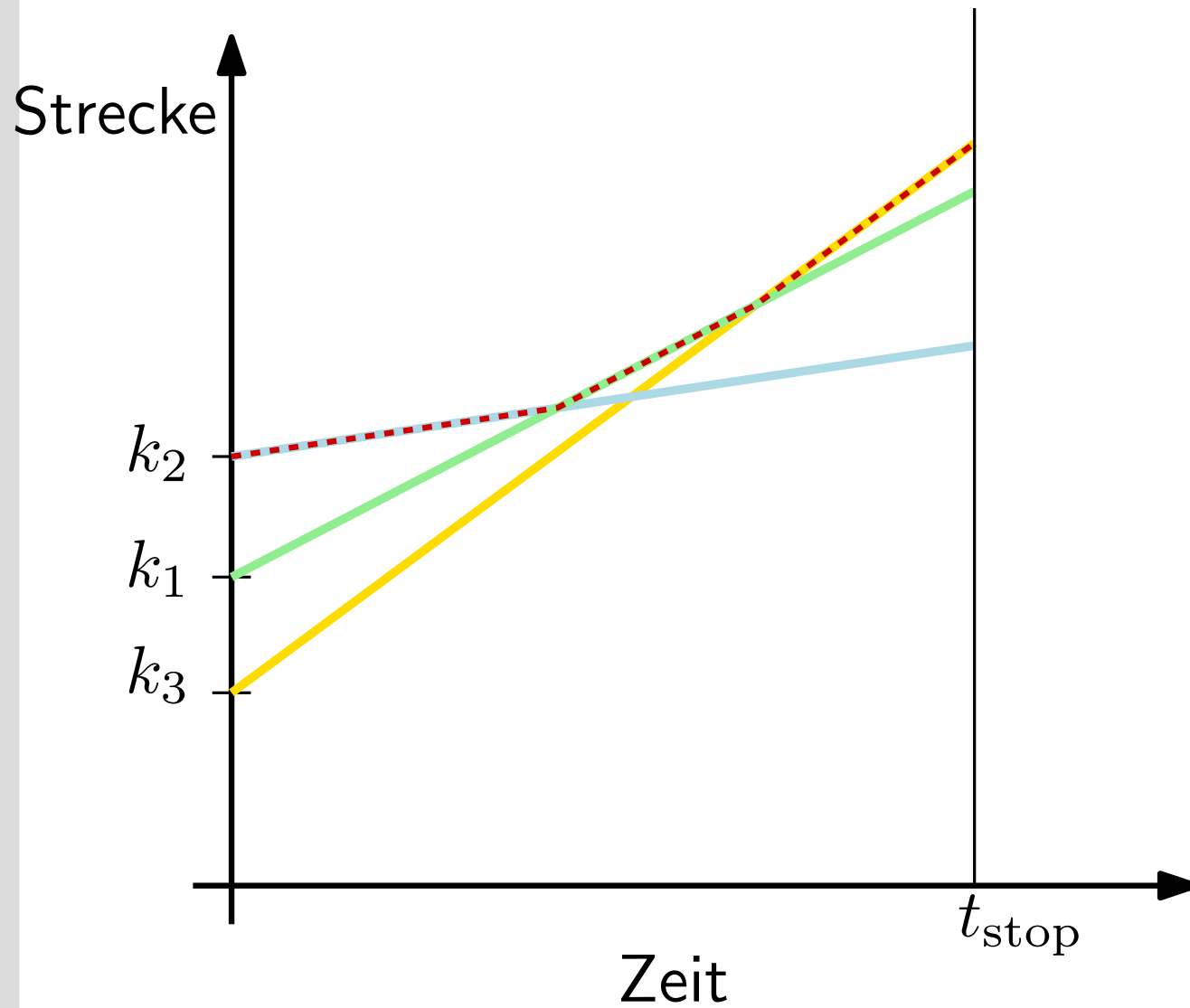
Algorithmus: welche Züge  
bis Zeitpunkt  $t_s$  einmal in  
Führung

# Aufgabe 3 : Züge

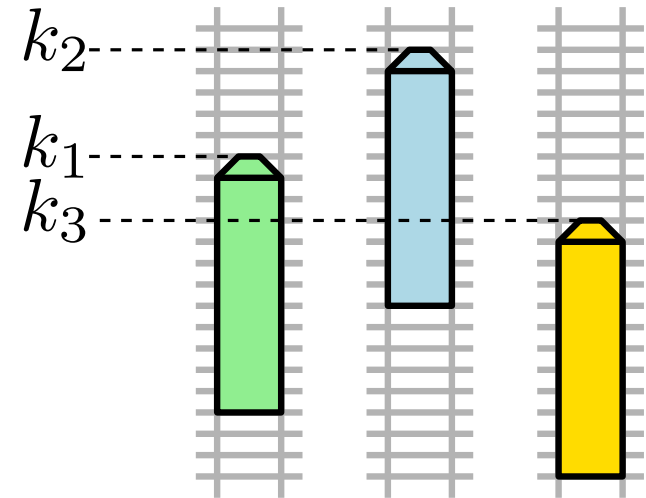
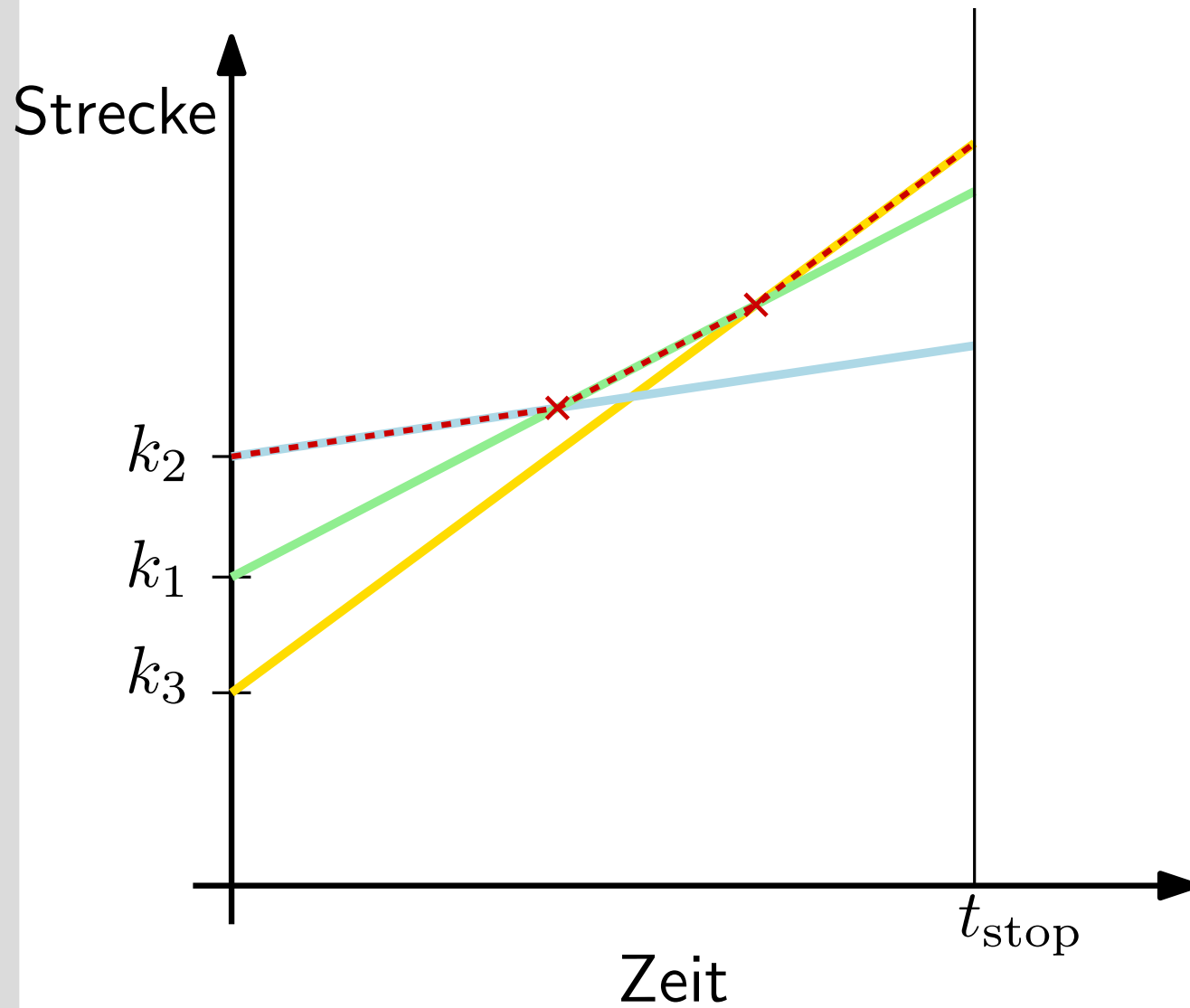




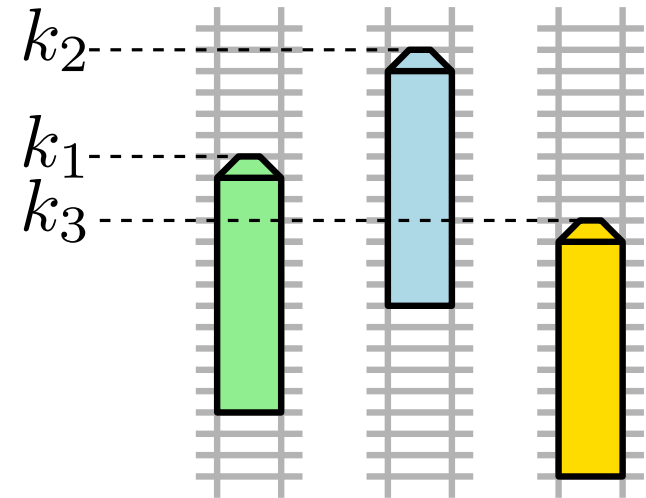
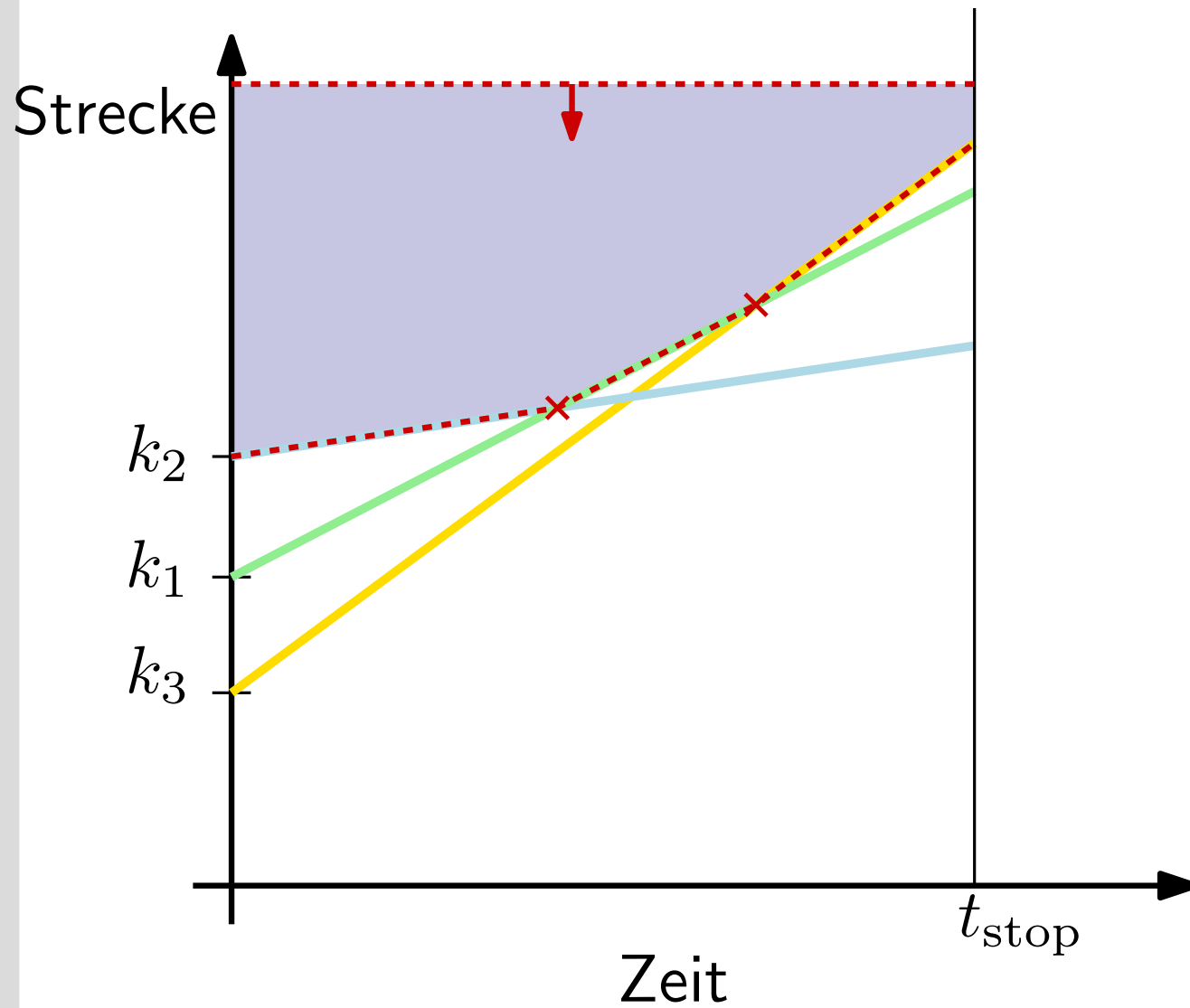
# Aufgabe 3 : Züge



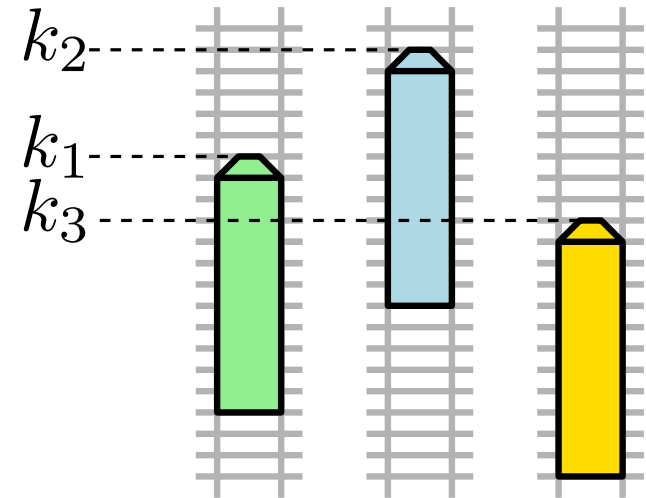
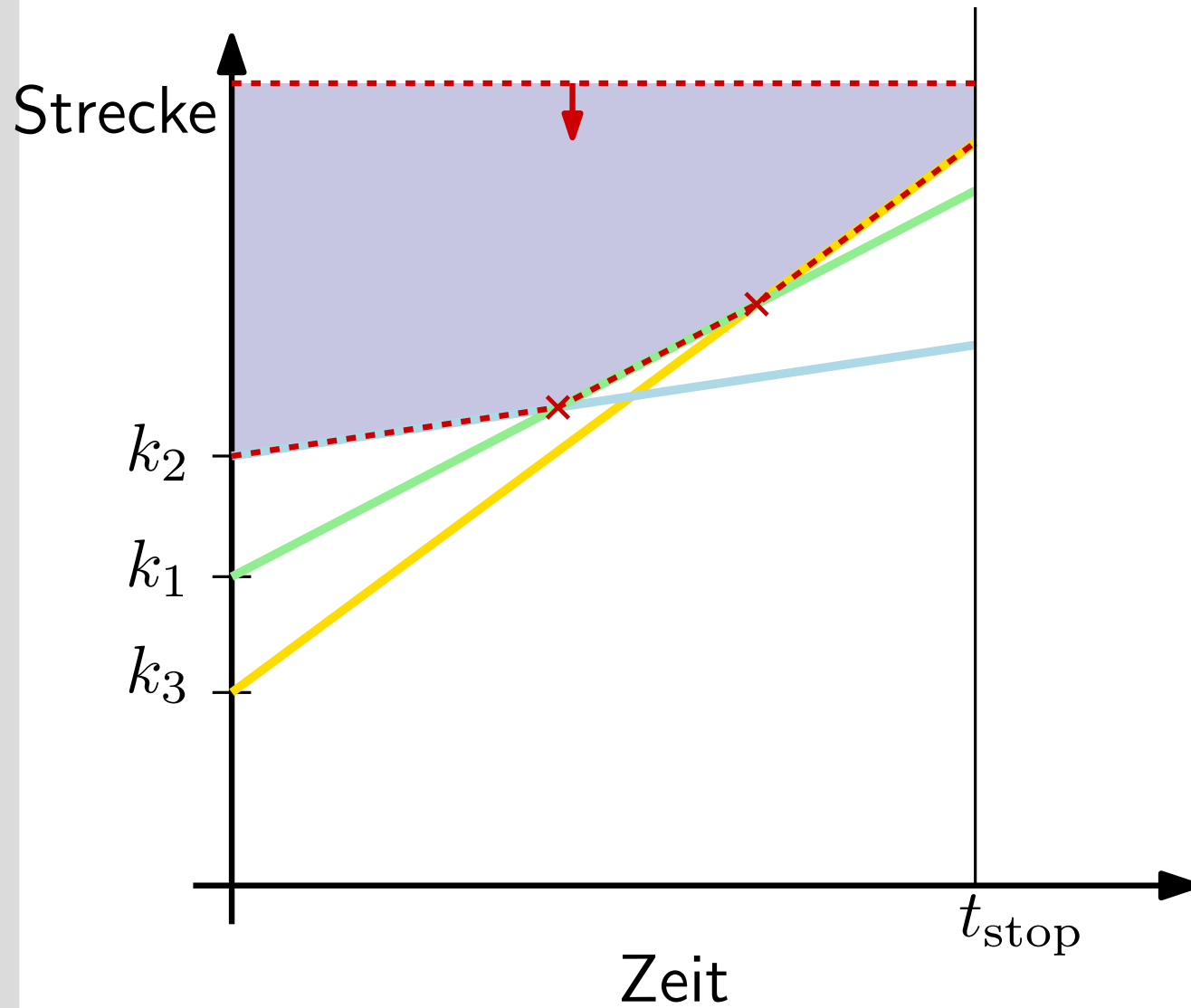
# Aufgabe 3 : Züge



# Aufgabe 3 : Züge



# Aufgabe 3 : Züge



# Aufgabe 3 : Züge

IntersectHalfplanes( $H$ )

**if**  $|H| = 1$  **then**

|  $C \leftarrow H$

**else**

|  $(H_1, H_2) \leftarrow \text{SplitInHalves}(H)$

|  $C_1 \leftarrow \text{IntersectHalfplanes}(H_1)$

|  $C_2 \leftarrow \text{IntersectHalfplanes}(H_2)$

|  $C \leftarrow \text{IntersectConvexRegions}(C_1, C_2)$

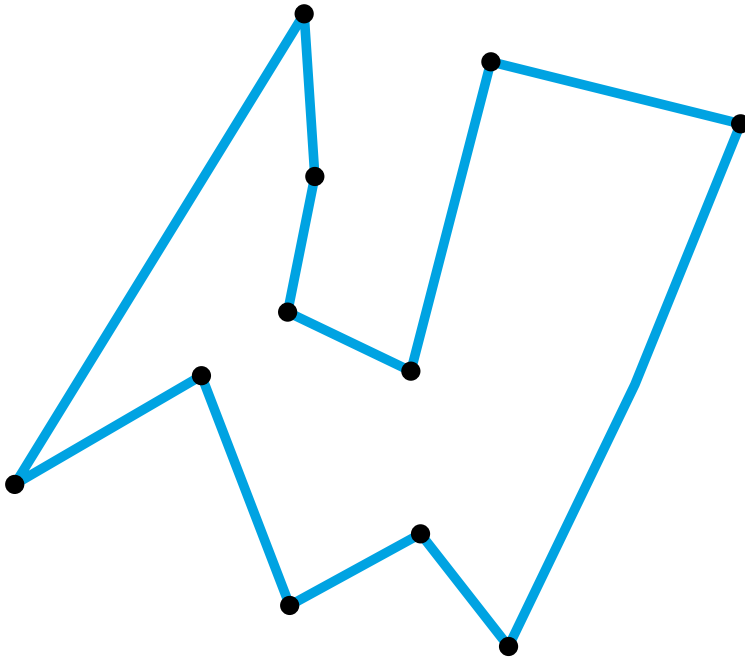
**return**  $C$

# Aufgabe 4 : Polygon Partitionieren

**VL:** ALG zur Zerlegung eines Polygons in  $y$ -mon. Teilstücke

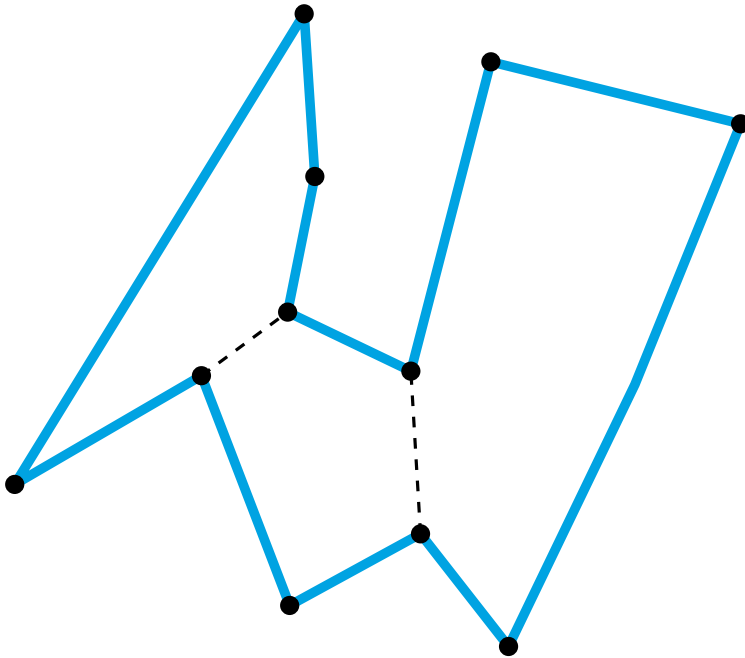
# Aufgabe 4 : Polygon Partitionieren

**VL:** ALG zur Zerlegung eines Polygons in  $y$ -mon. Teilstücke



# Aufgabe 4 : Polygon Partitionieren

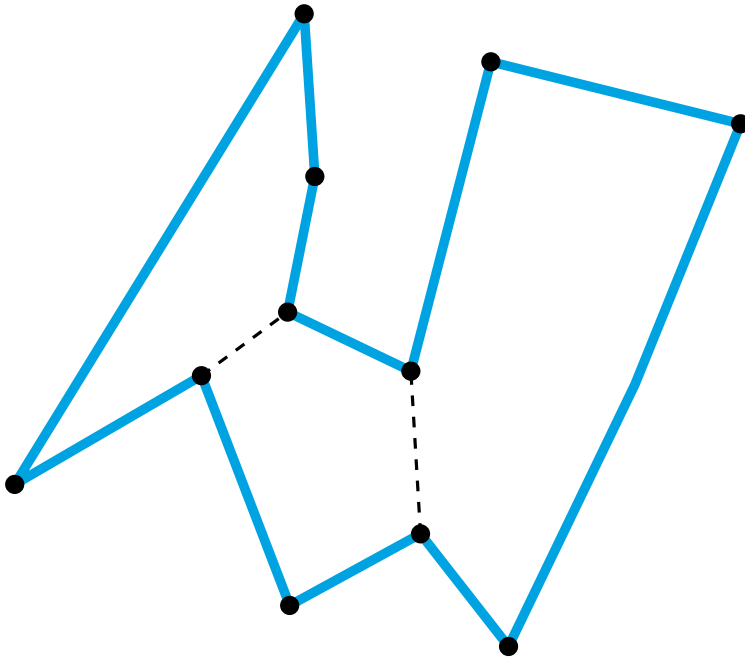
**VL:** ALG zur Zerlegung eines Polygons in  $y$ -mon. Teilstücke





# Aufgabe 4 : Polygon Partitionieren

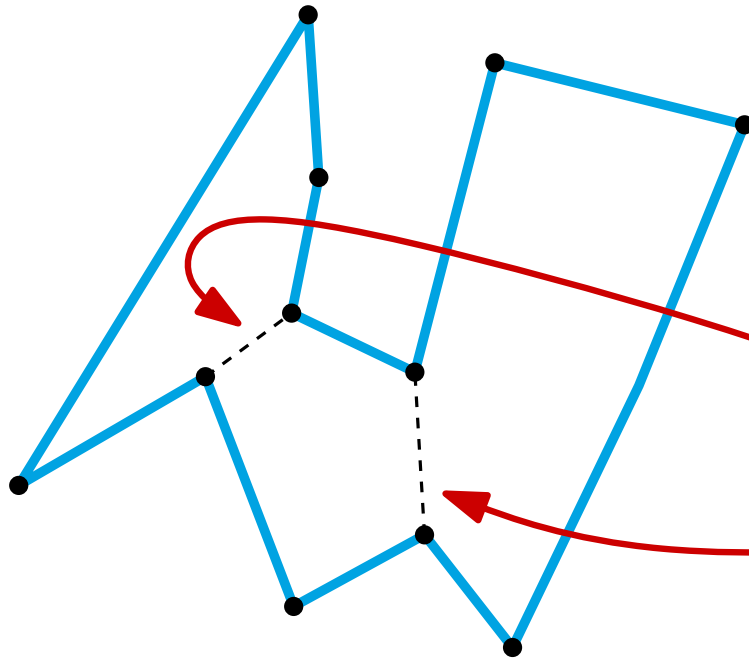
**VL:** ALG zur Zerlegung eines Polygons in  $y$ -mon. Teilstücke



**Datenstruktur:** Doppelt-verkettete Kantenliste (DCEL)

# Aufgabe 4 : Polygon Partitionieren

**VL:** ALG zur Zerlegung eines Polygons in  $y$ -mon. Teilstücke



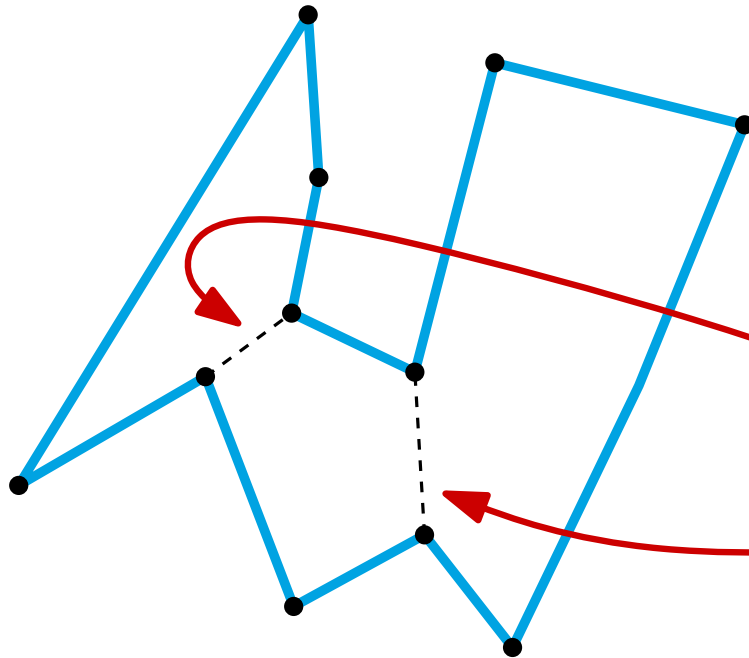
**Behauptung:**

Einfügen von einer  
Diagonalen in  $O(1)$  möglich

**Datenstruktur:** Doppelt-verkettete Kantenliste (DCEL)

# Aufgabe 4 : Polygon Partitionieren

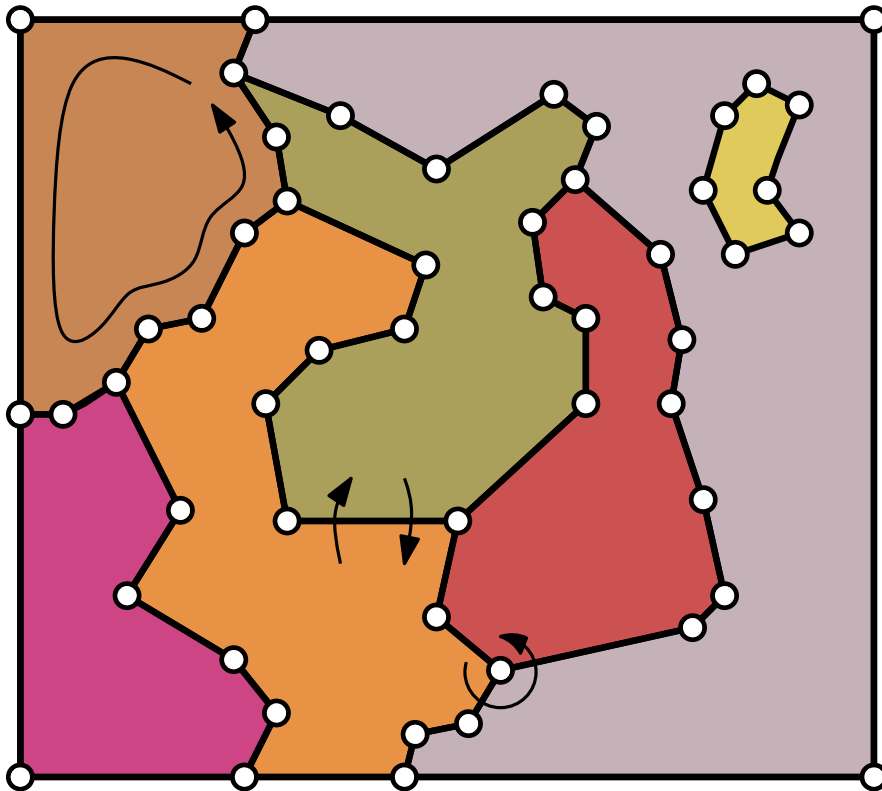
**VL:** ALG zur Zerlegung eines Polygons in  $y$ -mon. Teilstücke



**Behauptung:**

Einfügen von einer  
Diagonalen in  $O(1)$  möglich

**Datenstruktur:** Doppelt-verkettete Kantenliste (DCEL)



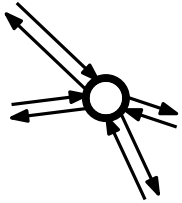
- (politische) Landkarte entspricht Unterteilung der Ebene in Polygone
- Unterteilung entspricht Einbettung eines planaren Graphen mit
  - Knoten
  - Kanten
  - Facetten

Welche Operationen sollte eine Datenstruktur für Unterteilungen der Ebene unterstützen?

- Kanten einer Facette ablaufen
- via Kante von Facette zu Facette wechseln
- Nachbarknoten in zyklischer Reihenfolge besuchen

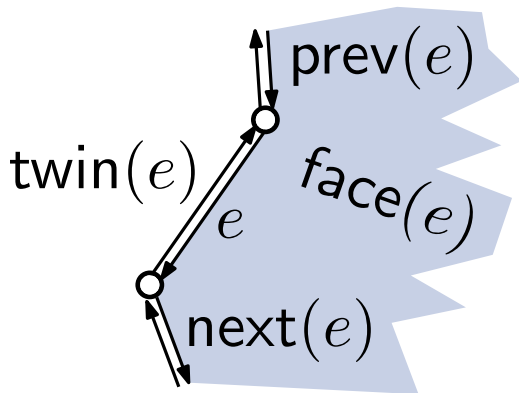
## Zutaten:

### ■ Knoten



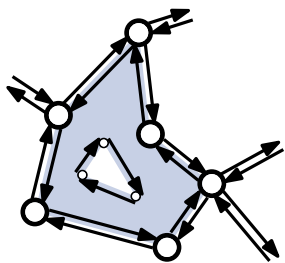
- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$

### ■ Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- inzidente Facette  $\text{face}(e)$

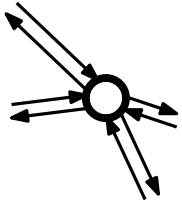
### ■ Facetten



- Randkante  $\text{outer}(f)$
- Kantenliste  $\text{inner}(f)$  für evtl. Löcher

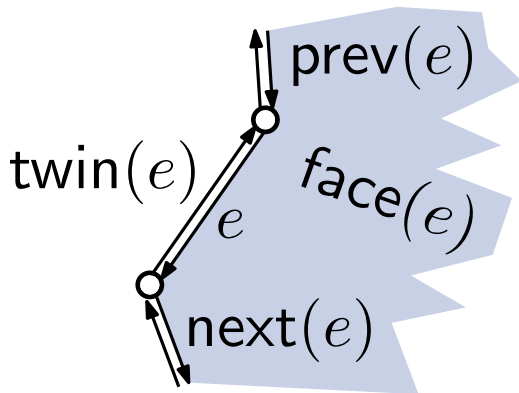
## Zutaten:

### ■ Knoten



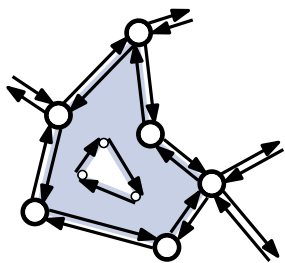
- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$

### ■ Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- inzidente Facette  $\text{face}(e)$

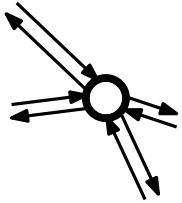
### ■ Facetten



- Randkante  $\text{outer}(f)$
- Kantenliste  $\text{inner}(f)$  für evtl. Löcher

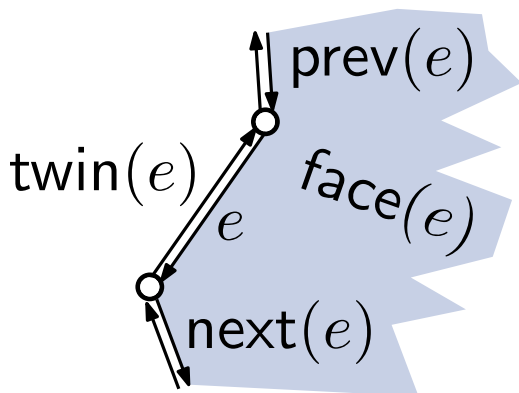
## Zutaten:

- Knoten



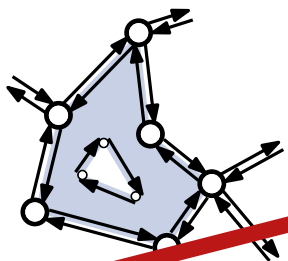
- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$

- Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- inzidente Facette  $\text{face}(e)$

- Facetten



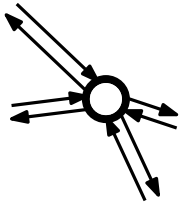
- Randkante  $\text{outer}(f)$
- Kantenliste  $\text{inner}(f)$  für evtl. Löcher

# Doppelt verkettete Kantenliste (DCEL)



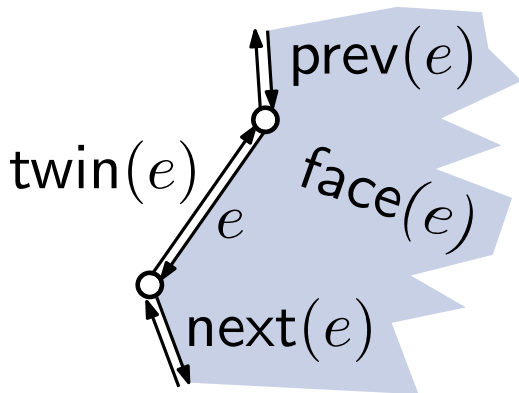
## Zutaten:

### ■ Knoten



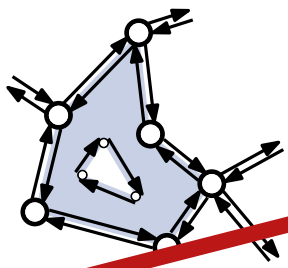
- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$

### ■ Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- ~~■ inzidente Facette  $\text{face}(e)$~~

### ■ Facetten



- ~~■ Randkante  $\text{outer}(f)$~~
- ~~■ Kantenliste  $\text{inner}(f)$  für evtl. Löcher~~

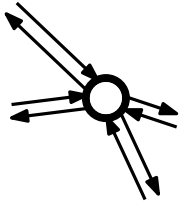


# Doppelt verkettete Kantenliste (DCEL)



## Zutaten:

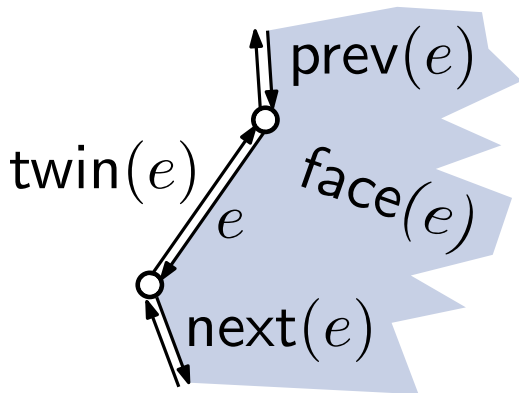
- Knoten



- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$

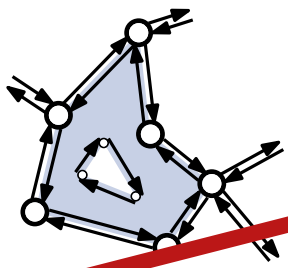


- Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- ~~■ inzidente Facette  $\text{face}(e)$~~

- ~~■ Facetten~~



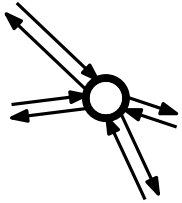
- ~~■ Randkante  $\text{outer}(f)$~~
- ~~■ Kantenliste  $\text{inner}(f)$  für evtl. Löcher~~

# Doppelt verkettete Kantenliste (DCEL)

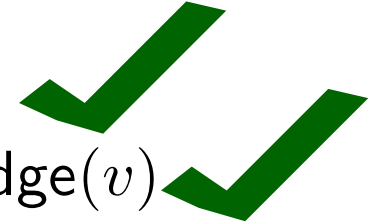


## Zutaten:

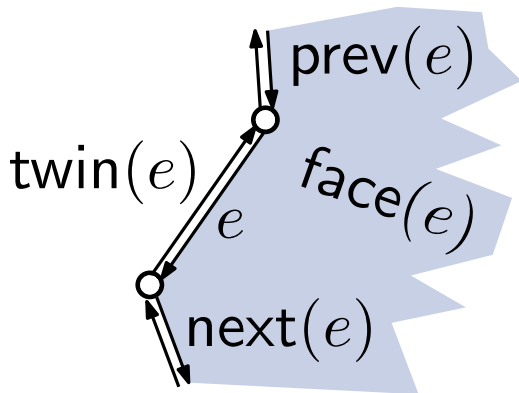
- Knoten



- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$

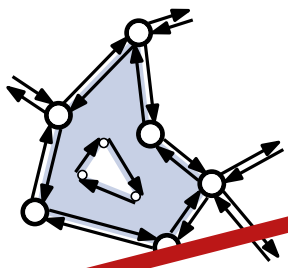


- Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- ~~■ inzidente Facette  $\text{face}(e)$~~

- ~~■ Facetten~~



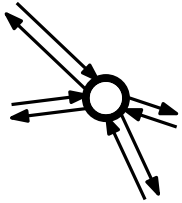
- ~~■ Randkante  $\text{outer}(f)$~~
- ~~■ Kantenliste  $\text{inner}(f)$  für evtl. Löcher~~

# Doppelt verkettete Kantenliste (DCEL)

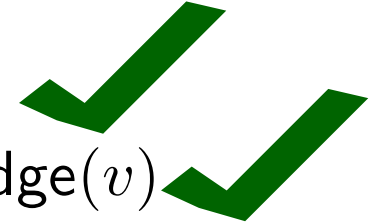


## Zutaten:

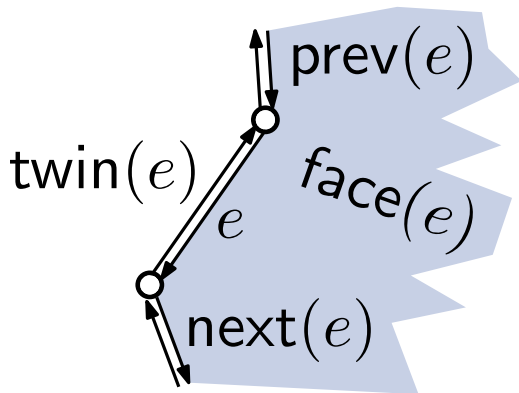
### ■ Knoten



- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$



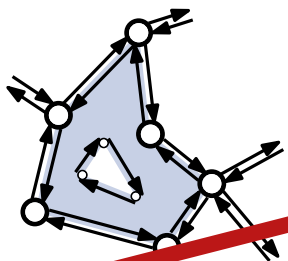
### ■ Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- ~~■ inzidente Facette  $\text{face}(e)$~~



### ■ Facetten



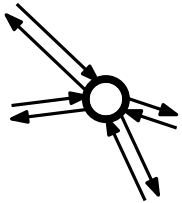
- ~~■ Randkante  $\text{outer}(f)$~~
- ~~■ Kantenliste  $\text{inner}(f)$  für evtl. Löcher~~

# Doppelt verkettete Kantenliste (DCEL)

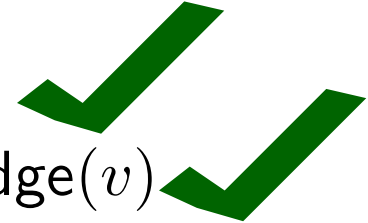


## Zutaten:

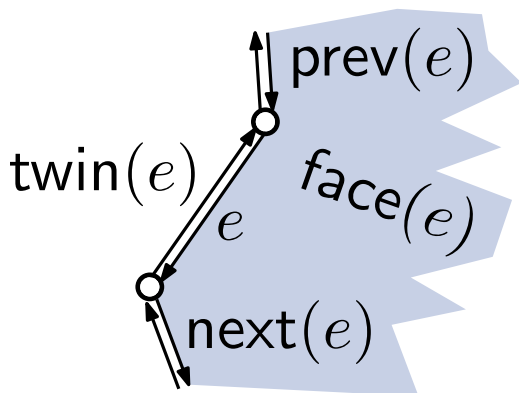
### ■ Knoten



- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$



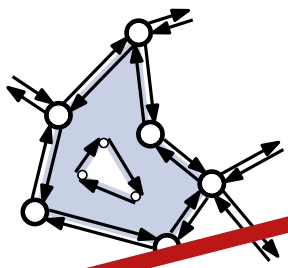
### ■ Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- ~~■ inzidente Facette  $\text{face}(e)$~~



### ■ Facetten



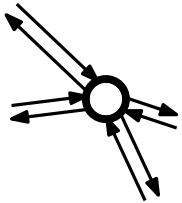
- ~~■ Randkante  $\text{outer}(f)$~~
- ~~■ Kantenliste  $\text{inner}(f)$  für evtl. Löcher~~

# Doppelt verkettete Kantenliste (DCEL)

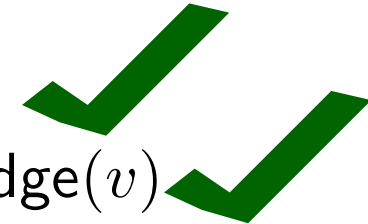


## Zutaten:

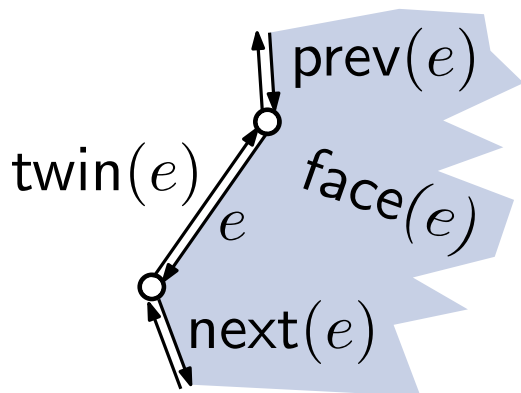
### ■ Knoten



- Koordinaten  $(x(v), y(v))$
- (erste) ausgehende Kante  $\text{edge}(v)$



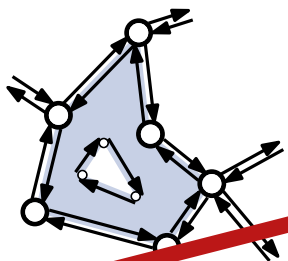
### ■ Kanten = zwei Halbkanten



- Knoten  $\text{origin}(v)$
- Gegenkante  $\text{twin}(e)$
- Vorgänger  $\text{prev}(e)$  & Nachfolger  $\text{next}(e)$
- ~~■ inzidente Facette  $\text{face}(e)$~~



### ■ Facetten



- ~~■ Randkante  $\text{outer}(f)$~~
- ~~■ Kantenliste  $\text{inner}(f)$  für evtl. Löcher~~

**1.** Wie umgeht man das aktualisieren der  $face(e)$  Information?

1. Wie umgeht man das Aktualisieren der  $face(e)$  Information?
  - Facetten spielen keine (große) Rolle

1. Wie umgeht man das Aktualisieren der  $face(e)$  Information?
  - Facetten spielen keine (große) Rolle
  
2. Wie wählen wir in  $O(1)$  die richtigen Kanten für das Anpassen der  $next(e)$  bzw.  $prev(e)$  Einträge?



1. Wie umgeht man das Aktualisieren der  $face(e)$  Information?
  - Facetten spielen keine (große) Rolle
  
2. Wie wählen wir in  $O(1)$  die richtigen Kanten für das Anpassen der  $next(e)$  bzw.  $prev(e)$  Einträge?
  - a) Nur konstant viele Kanten inzident zu jedem Knoten

1. Wie umgeht man das Aktualisieren der  $face(e)$  Information?
  - Facetten spielen keine (große) Rolle
  
2. Wie wählen wir in  $O(1)$  die richtigen Kanten für das Anpassen der  $next(e)$  bzw.  $prev(e)$  Einträge?
  - a) Nur konstant viele Kanten inzident zu jedem Knoten
  - b) Durch passende Sortierung können wir die korrekten Kanten finden

# Algorithmus MakeMonotone( $P$ )

## MakeMonotone(Polygon $P$ )

$\mathcal{D} \leftarrow$  doppelt-verkettete Kantenliste für  $(V(P), E(P))$

$Q \leftarrow$  priority queue für  $V(P)$  lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$  (binärer Suchbaum für Sweep-Line Status)

**while**  $Q \neq \emptyset$  **do**

```
|  $v \leftarrow Q.\text{nextVertex}()$   
|  $Q.\text{deleteVertex}(v)$   
|  $\text{handleVertex}(v)$ 
```

**return**  $\mathcal{D}$

# Algorithmus MakeMonotone(P)

## MakeMonotone(Polygon $P$ )

$\mathcal{D} \leftarrow$  doppelt-verkettete Kantenliste für  $(V(P), E(P))$

$\mathcal{Q} \leftarrow$  priority queue für  $V(P)$  lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$  (binärer Suchbaum für Sweep-Line Status)

**while**  $\mathcal{Q} \neq \emptyset$  **do**

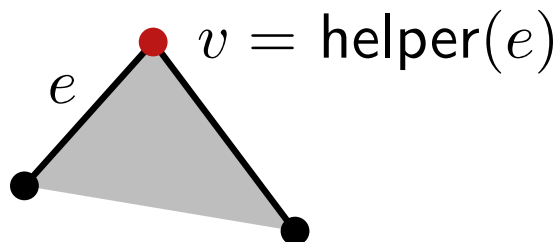
$v \leftarrow \mathcal{Q}.\text{nextVertex}()$   
     $\mathcal{Q}.\text{deleteVertex}(v)$   
     $\text{handleVertex}(v)$

**return**  $\mathcal{D}$

## handleStartVertex(vertex $v$ )

$\mathcal{T} \leftarrow$  füge linke Kante  $e$  ein

helper( $e$ )  $\leftarrow v$



# Algorithmus MakeMonotone(P)

## MakeMonotone(Polygon $P$ )

$\mathcal{D} \leftarrow$  doppelt-verkettete Kantenliste für  $(V(P), E(P))$

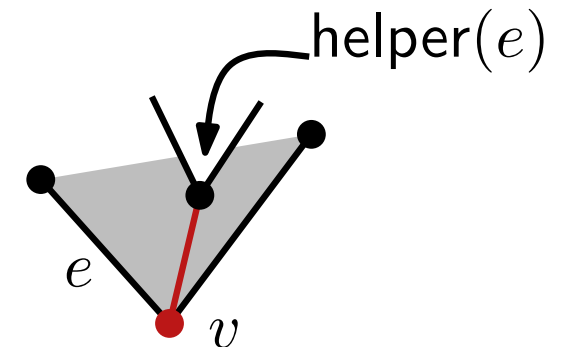
$Q \leftarrow$  priority queue für  $V(P)$  lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$  (binärer Suchbaum für Sweep-Line Status)

**while**  $Q \neq \emptyset$  **do**

$v \leftarrow Q.\text{nextVertex}()$   
     $Q.\text{deleteVertex}(v)$   
     $\text{handleVertex}(v)$

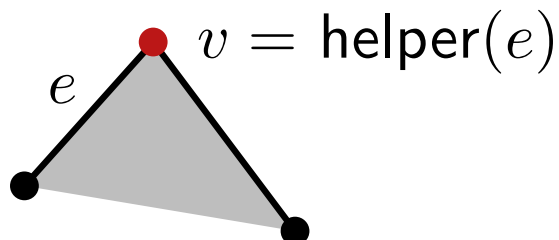
**return**  $\mathcal{D}$



## handleStartVertex(vertex $v$ )

$\mathcal{T} \leftarrow$  füge linke Kante  $e$  ein

$\text{helper}(e) \leftarrow v$



## handleEndVertex(vertex $v$ )

$e \leftarrow$  linke Kante

**if**  $\text{isMergeVertex}(\text{helper}(e))$  **then**

$\mathcal{D} \leftarrow$  füge  $(\text{helper}(e), v)$  ein

lösche  $e$  aus  $\mathcal{T}$

# Algorithmus MakeMonotone(P)

## MakeMonotone(Polygon $P$ )

$\mathcal{D} \leftarrow$  doppelt-verkettete Kantenliste für  $(V(P), E(P))$

$\mathcal{Q} \leftarrow$  priority queue für  $V(P)$  lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$  (binärer Suchbaum für Sweep-Line Status)

**while**  $\mathcal{Q} \neq \emptyset$  **do**

```
┌  $v \leftarrow \mathcal{Q}.\text{nextVertex}()$   
├  $\mathcal{Q}.\text{deleteVertex}(v)$   
└  $\text{handleVertex}(v)$ 
```

**return**  $\mathcal{D}$

## handleSplitVertex(vertex $v$ )

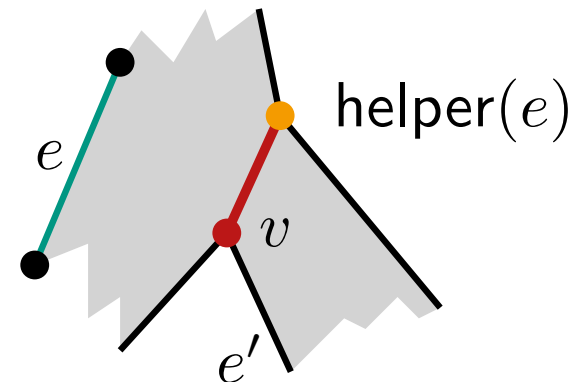
$e \leftarrow$  Kante links von  $v$  in  $\mathcal{T}$

$\mathcal{D} \leftarrow$  füge  $(\text{helper}(e), v)$  ein

$\text{helper}(e) \leftarrow v$

$\mathcal{T} \leftarrow$  füge rechte Kante  $e'$  von  $v$  ein

$\text{helper}(e') \leftarrow v$



# Algorithmus MakeMonotone(P)

## MakeMonotone(Polygon $P$ )

$\mathcal{D} \leftarrow$  doppelt-verkettete Kantenliste für  $(V(P), E(P))$

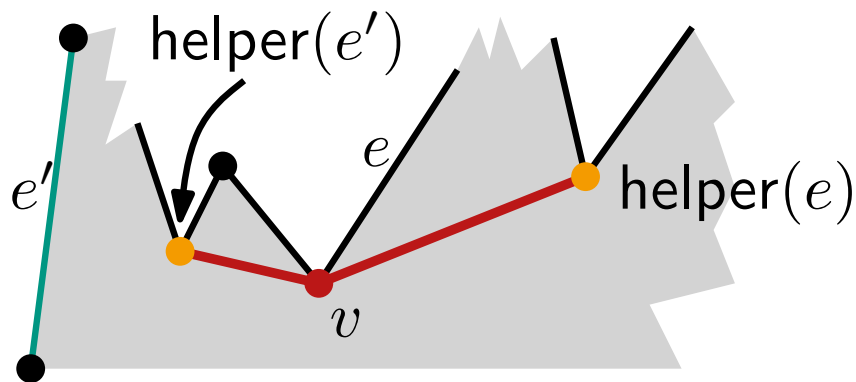
$Q \leftarrow$  priority queue für  $V(P)$  lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$  (binärer Suchbaum für Sweep-Line Status)

**while**  $Q \neq \emptyset$  **do**

$v \leftarrow Q.\text{nextVertex}()$   
     $Q.\text{deleteVertex}(v)$   
    handleVertex( $v$ )

**return**  $\mathcal{D}$



## handleMergeVertex(vertex $v$ )

$e \leftarrow$  rechte Kante

**if** isMergeVertex(helper( $e$ )) **then**

$\mathcal{D} \leftarrow$  füge (helper( $e$ ),  $v$ ) ein

lösche  $e$  aus  $\mathcal{T}$

$e' \leftarrow$  Kante links von  $v$  in  $\mathcal{T}$

**if** isMergeVertex(helper( $e'$ )) **then**

$\mathcal{D} \leftarrow$  füge (helper( $e'$ ),  $v$ ) ein

helper( $e'$ )  $\leftarrow v$

# Algorithmus MakeMonotone( $P$ )

## MakeMonotone(Polygon $P$ )

$\mathcal{D} \leftarrow$  doppelt-verkettete Kantenliste für  $(V(P), E(P))$

$Q \leftarrow$  priority queue für  $V(P)$  lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$  (binärer Suchbaum für Sweep-Line Status)

**while**  $Q \neq \emptyset$  **do**

$v \leftarrow Q.\text{nextVertex}()$   
     $Q.\text{deleteVertex}(v)$   
    handleVertex( $v$ )

**return**  $\mathcal{D}$

**handleRegularVertex**(vertex  $v$ )

**if**  $P$  liegt lokal rechts von  $v$  **then**

$e, e' \leftarrow$  obere, untere Kante

**if** isMergeVertex(helper( $e$ )) **then**

$\mathcal{D} \leftarrow$  füge (helper( $e$ ),  $v$ ) ein

    lösche  $e$  aus  $\mathcal{T}$

$\mathcal{T} \leftarrow$  füge  $e'$  ein; helper( $e'$ )  $\leftarrow v$

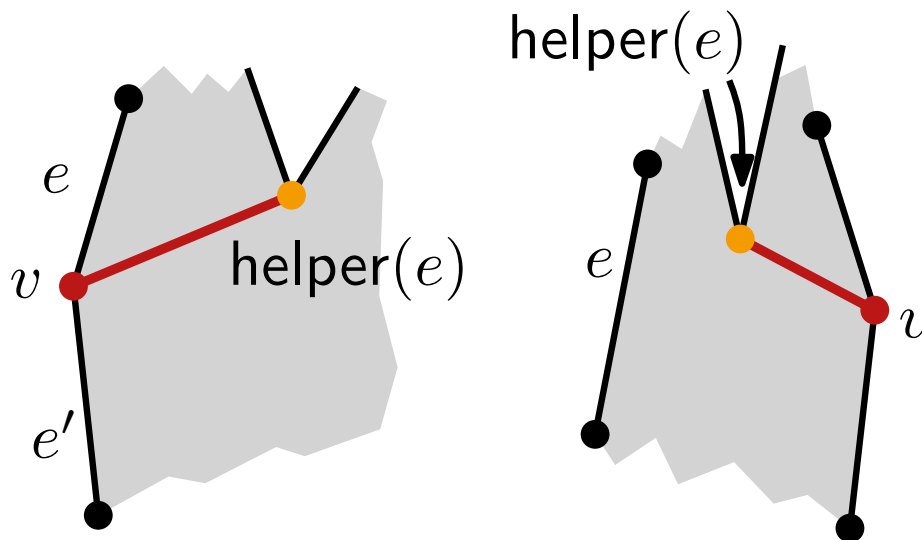
**else**

$e \leftarrow$  Kante links von  $v$  in  $\mathcal{T}$

**if** isMergeVertex(helper( $e$ )) **then**

$\mathcal{D} \leftarrow$  füge (helper( $e$ ),  $v$ ) ein

    helper( $e$ )  $\leftarrow v$





# Algorithmus MakeMonotone( $P$ )

## MakeMonotone(Polygon $P$ )

$\mathcal{D} \leftarrow$  doppelt-verkettete Kantenliste für  $(V(P), E(P))$

$Q \leftarrow$  priority queue für  $V(P)$  lexikographisch sortiert

$\mathcal{T} \leftarrow \emptyset$  (binärer Suchbaum für Sweep-Line Status)

**while**  $Q \neq \emptyset$  **do**

$v \leftarrow Q.\text{nextVertex}()$

$Q.\text{deleteVertex}(v)$

    handleVertex( $v$ )

**return**  $\mathcal{D}$

**handleRegularVertex**(vertex  $v$ )

**if**  $P$  liegt lokal rechts von  $v$  **then**

$e, e' \leftarrow$  obere, untere Kante

**if** isMergeVertex(helper( $e$ )) **then**

$\mathcal{D} \leftarrow$  füge (helper( $e$ ),  $v$ ) ein

    lösche  $e$  aus  $\mathcal{T}$

$\mathcal{T} \leftarrow$  füge  $e'$  ein; helper( $e'$ )  $\leftarrow v$

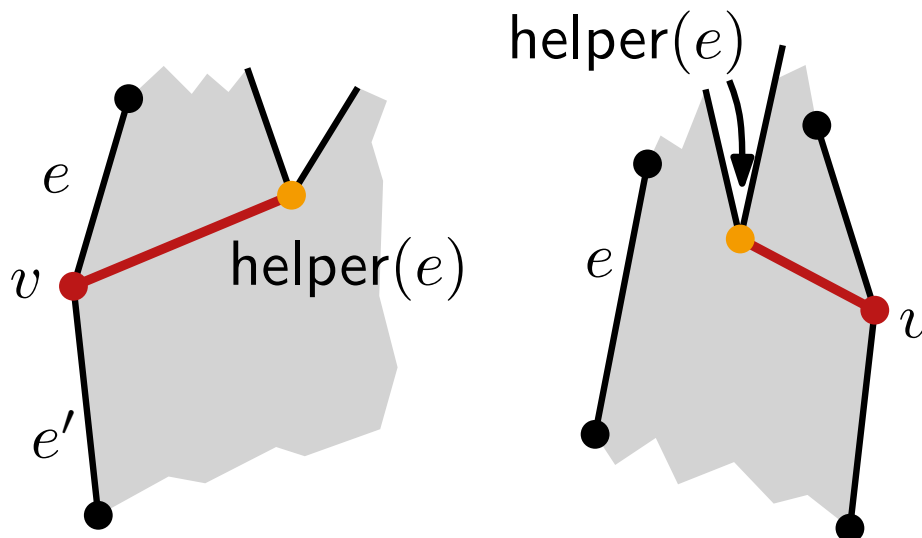
**else**

$e \leftarrow$  Kante links von  $v$  in  $\mathcal{T}$

**if** isMergeVertex(helper( $e$ )) **then**

$\mathcal{D} \leftarrow$  füge (helper( $e$ ),  $v$ ) ein

    helper( $e$ )  $\leftarrow v$



2. Wie wählen wir in  $O(1)$  die richtigen Kanten für das anpassen der  $next(e)$  bzw.  $prev(e)$  Einträge?
- a) Nur konstant viele Kanten inzident zu jedem Knoten
    - Initial hat jeder Knoten Grad 2
    - Jeder Knoten ist höchstens einmal helper: +1
    - Jeder Knoten ist höchstens einmal 'Sonderknoten': +2
  - b) Durch passende Sortierung können wir die korrekten Kanten finden

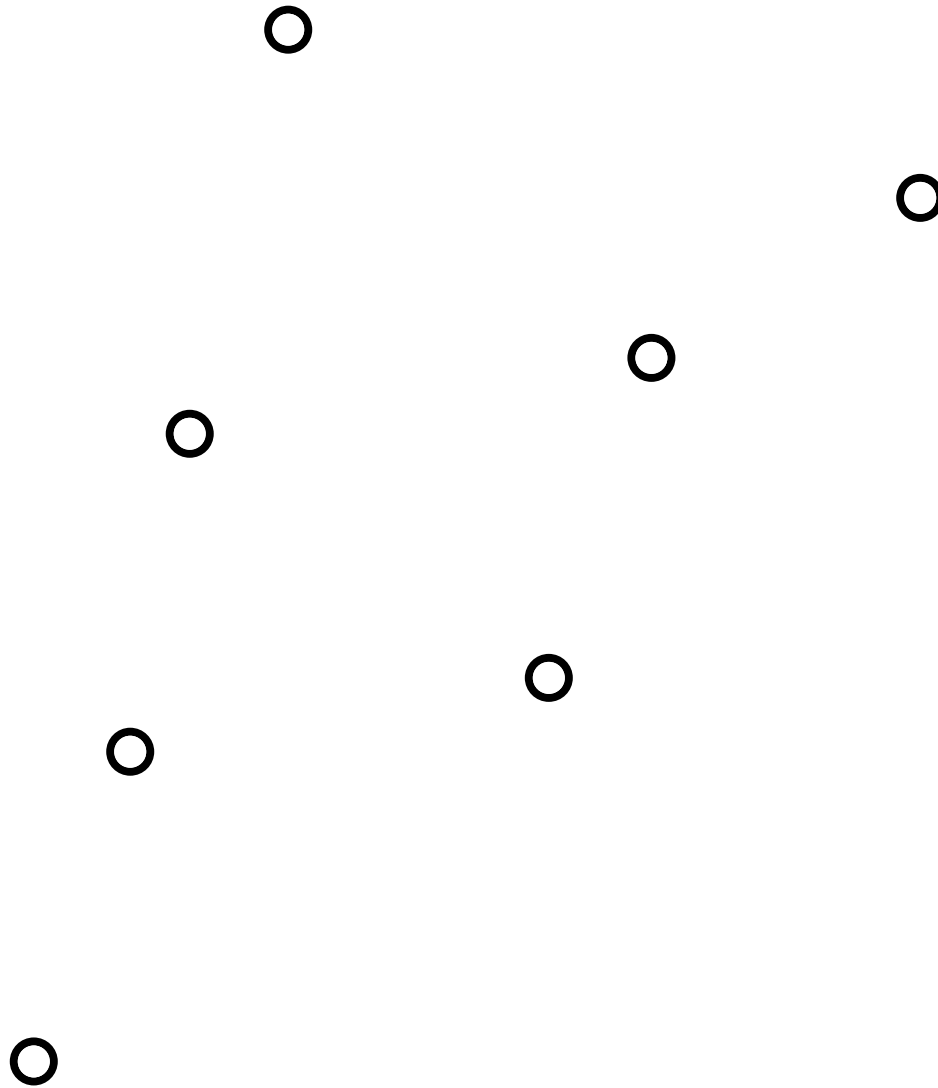
Übungsblatt 4

Nachtrag zu Übungsblatt 3

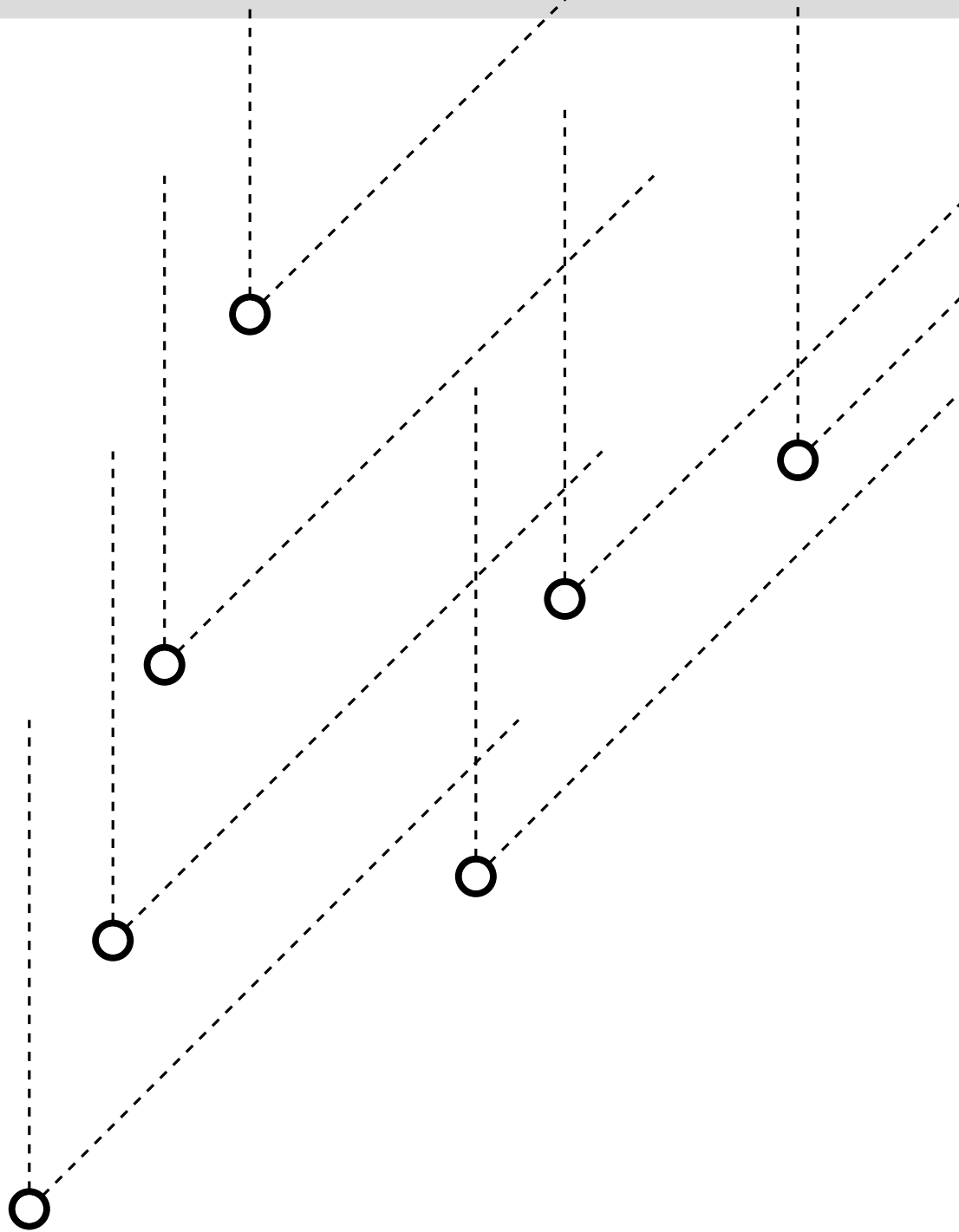
Übungsblatt 5

Werbung

# Nachtrag

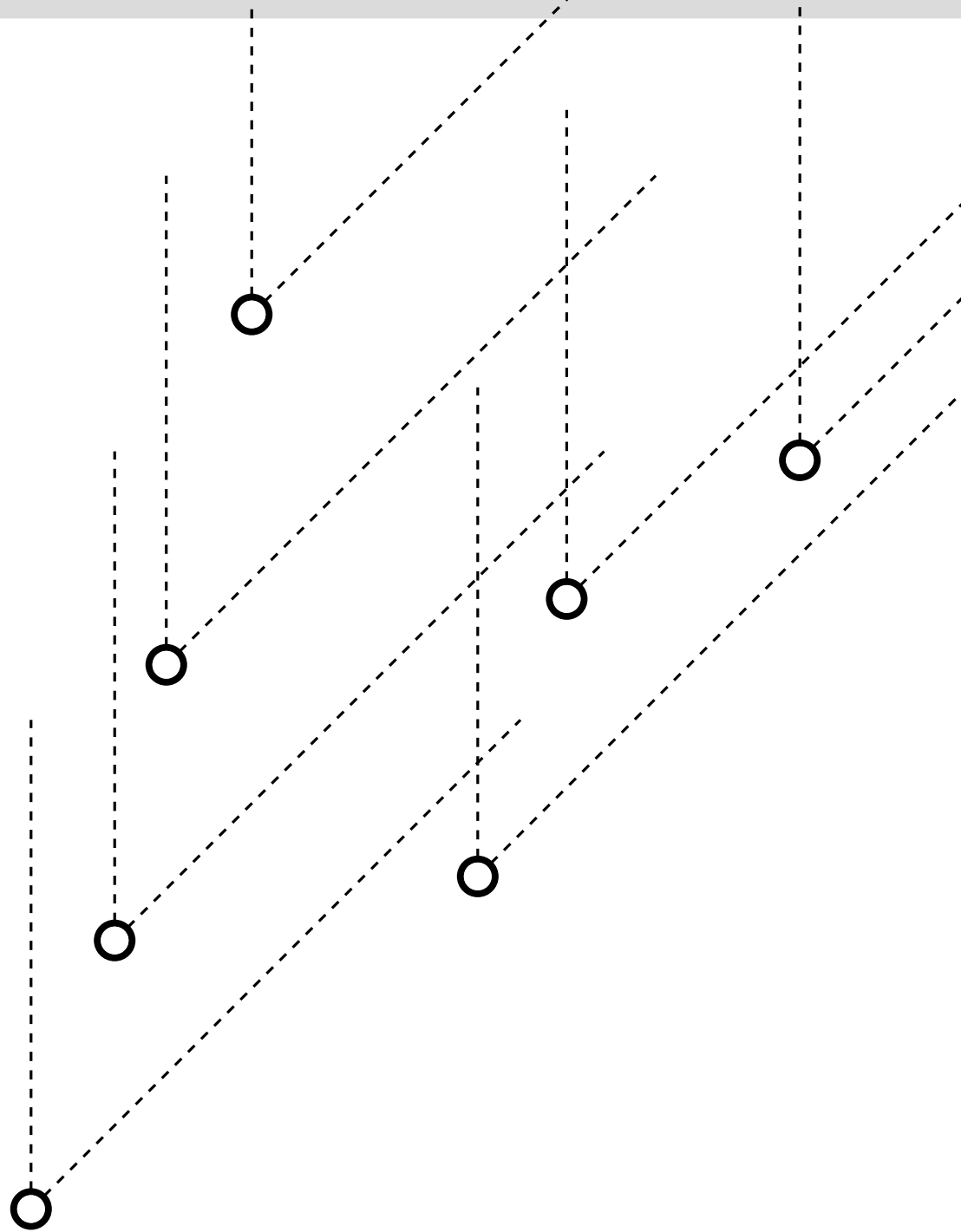


# Nachtrag

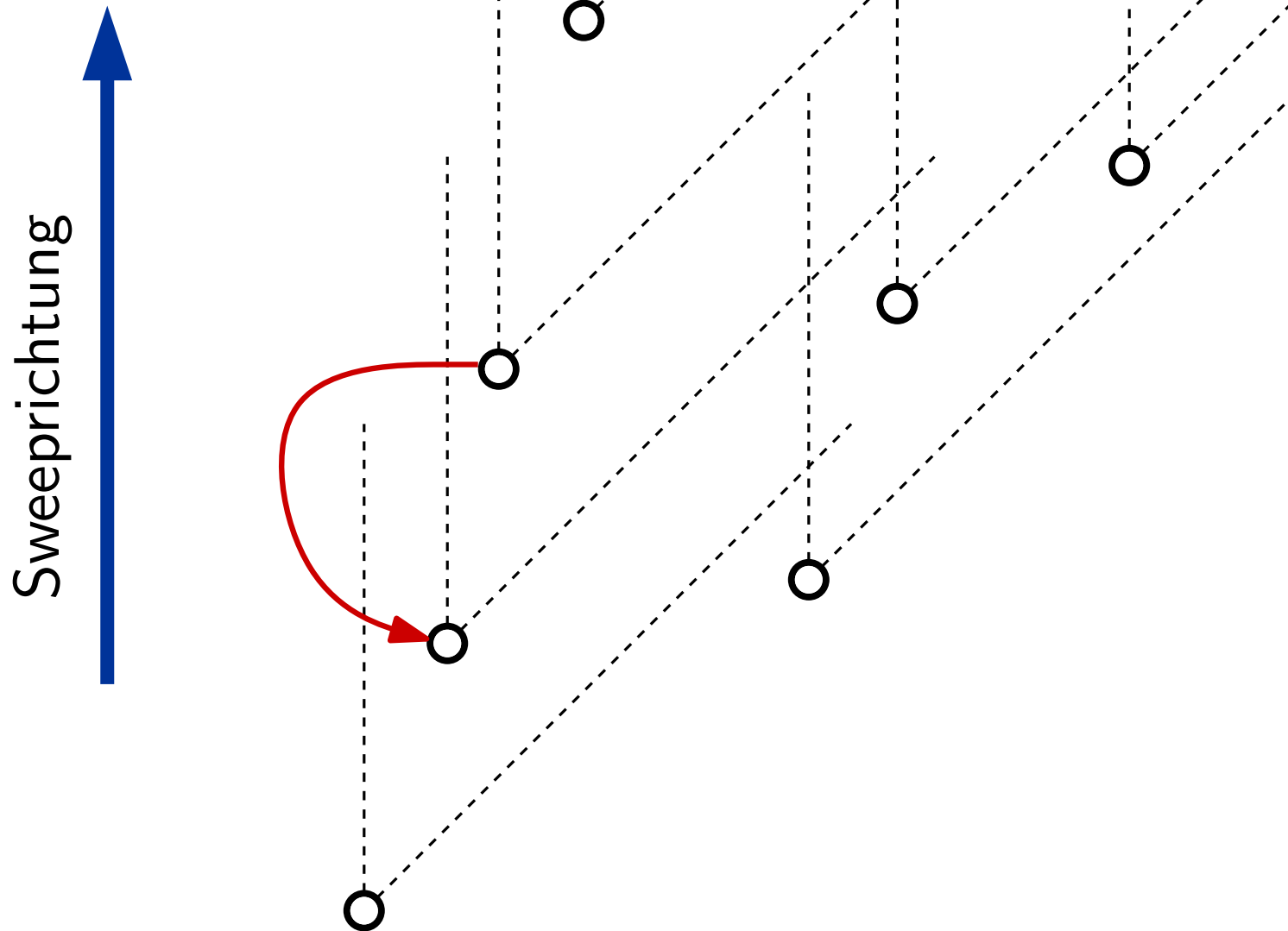




Sweeprichtung  
↑

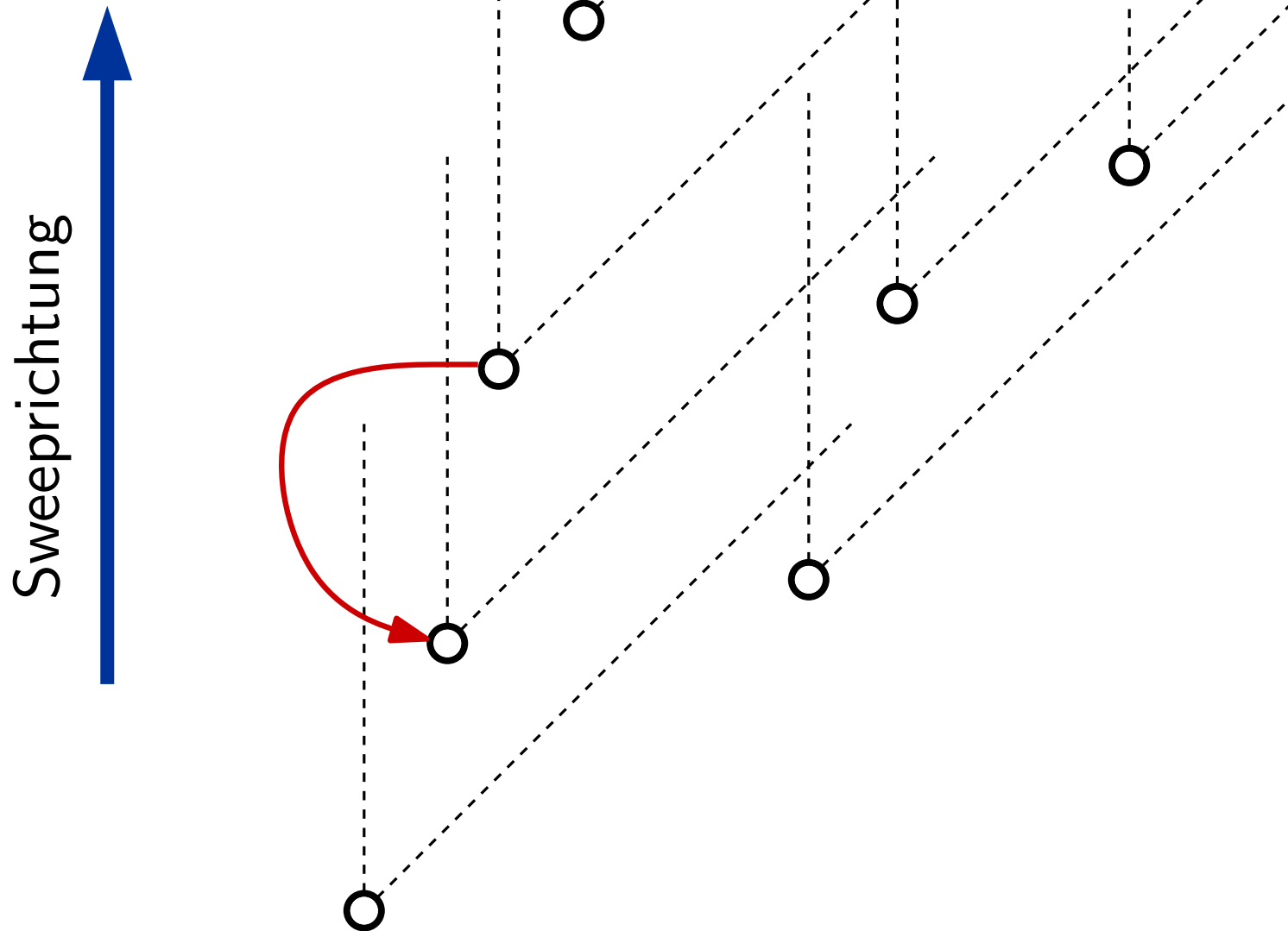


**Idee:** balancierter binärer Suchbaum  $T$



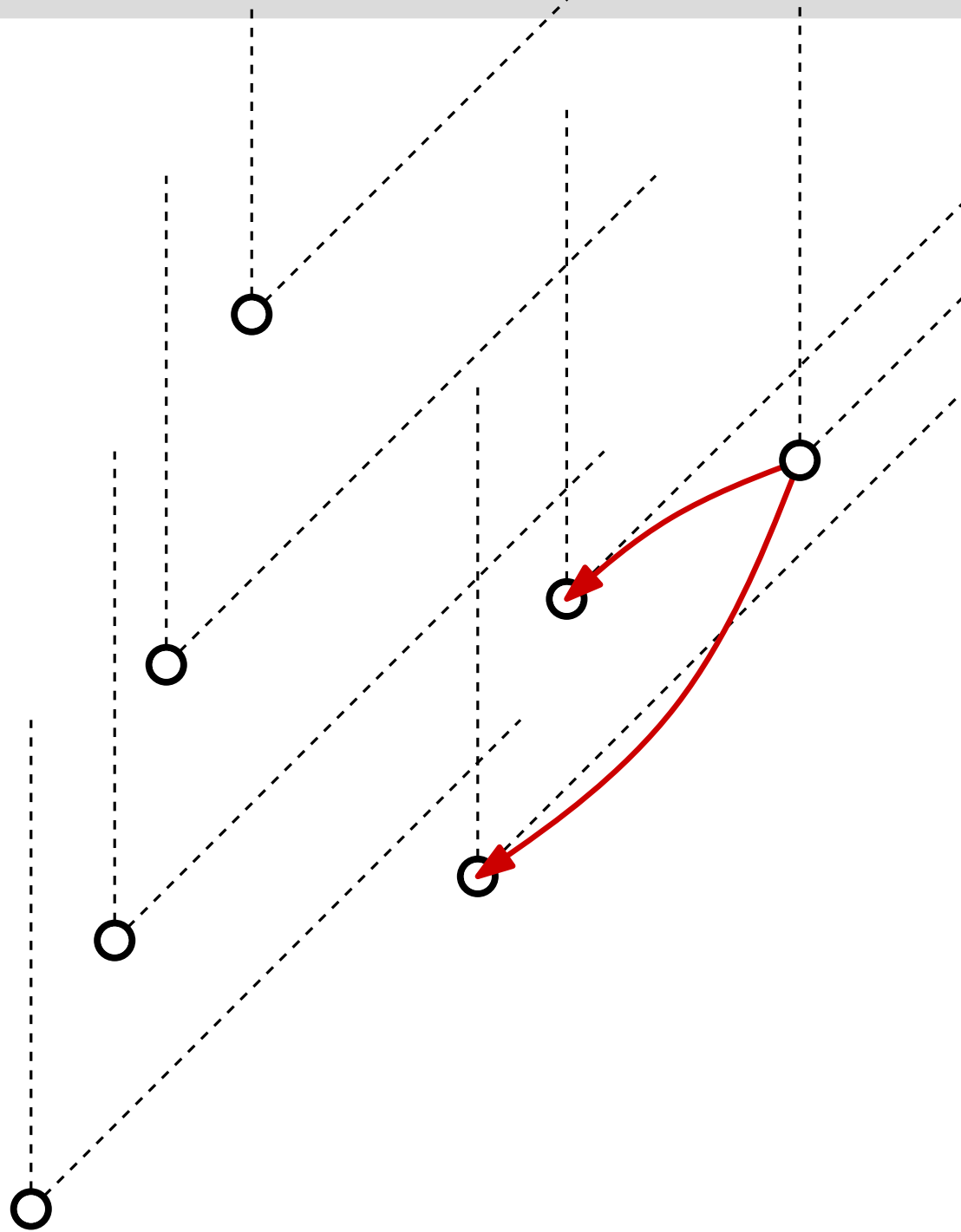
**Idee:** balancierter binärer Suchbaum  $T$





**Idee:** balancierter binärer Suchbaum  $T$

Sweeprichtung  
↑



**Idee:** balancierter binärer Suchbaum  $T$

Übungsblatt 4

Nachtrag zu Übungsblatt 3

Übungsblatt 5

Werbung

# Aufgabe 1

*kd*-Trees – w-c Laufzeit

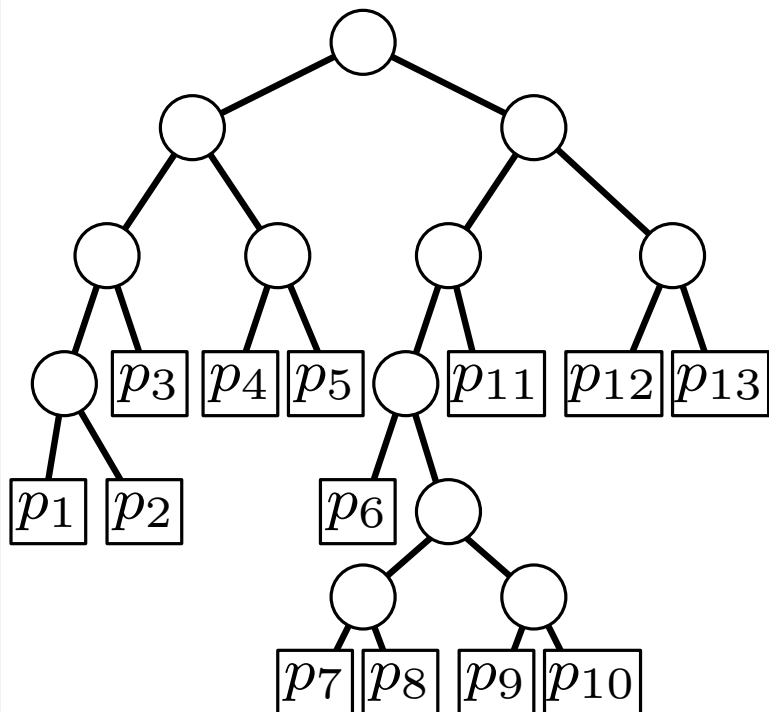
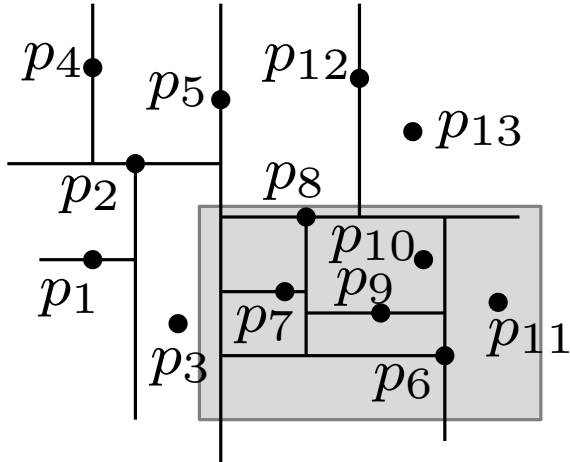
## Anfragen

a) warum?

$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$



# Bereichsabfrage in einem $kd$ -Tree

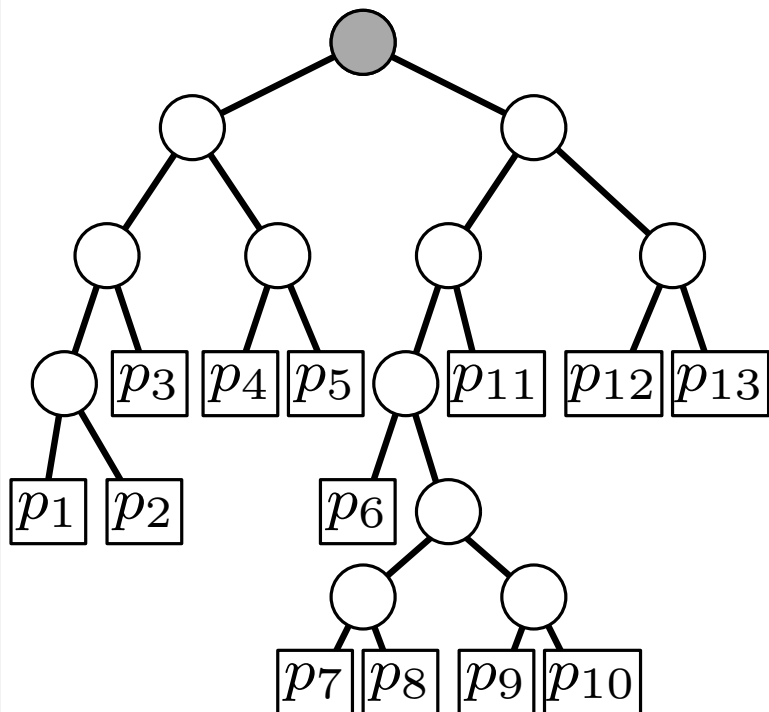
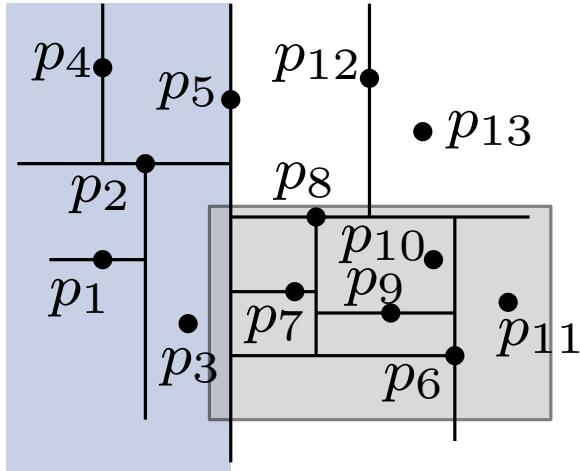


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

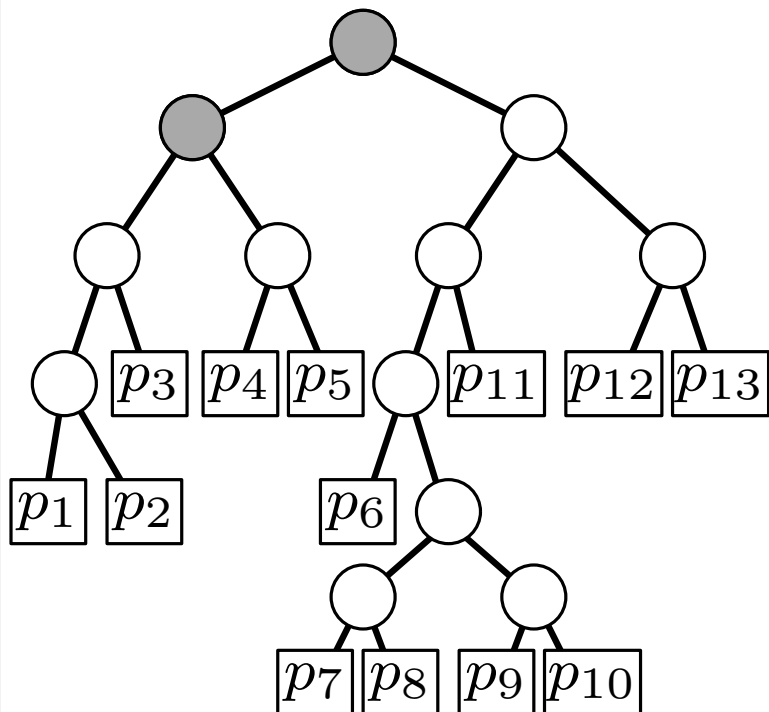
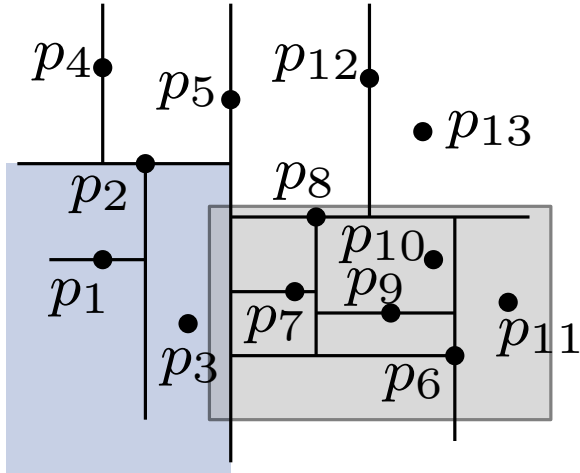


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    | prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    | if region(lc( $v$ ))  $\subseteq R$  then
    | | ReportSubtree(lc( $v$ ))
    | else
    | | if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
    | | | SearchKdTree(lc( $v$ ),  $R$ )
    | if region(rc( $v$ ))  $\subseteq R$  then
    | | ReportSubtree(rc( $v$ ))
    | else
    | | if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
    | | | SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree



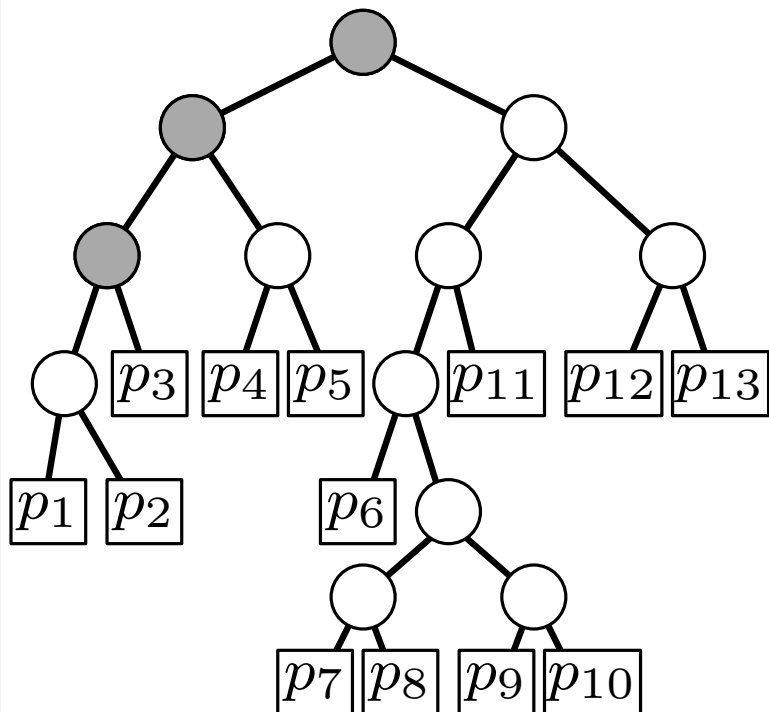
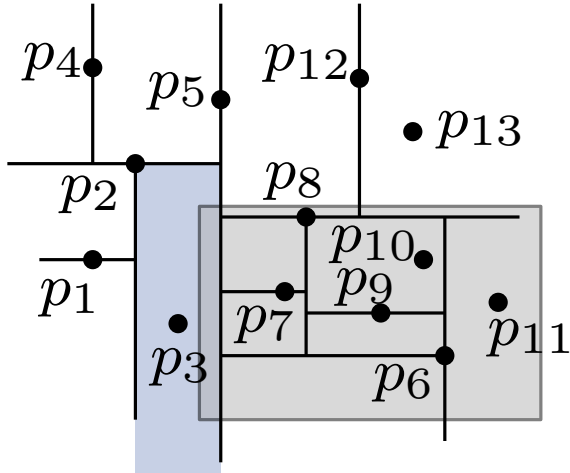
SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    | prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    | if region(lc( $v$ ))  $\subseteq R$  then
    | | ReportSubtree(lc( $v$ ))
    | else
    | | if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
    | | | SearchKdTree(lc( $v$ ),  $R$ )
    | if region(rc( $v$ ))  $\subseteq R$  then
    | | ReportSubtree(rc( $v$ ))
    | else
    | | if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
    | | | SearchKdTree(rc( $v$ ),  $R$ )
    
```



# Bereichsabfrage in einem $kd$ -Tree

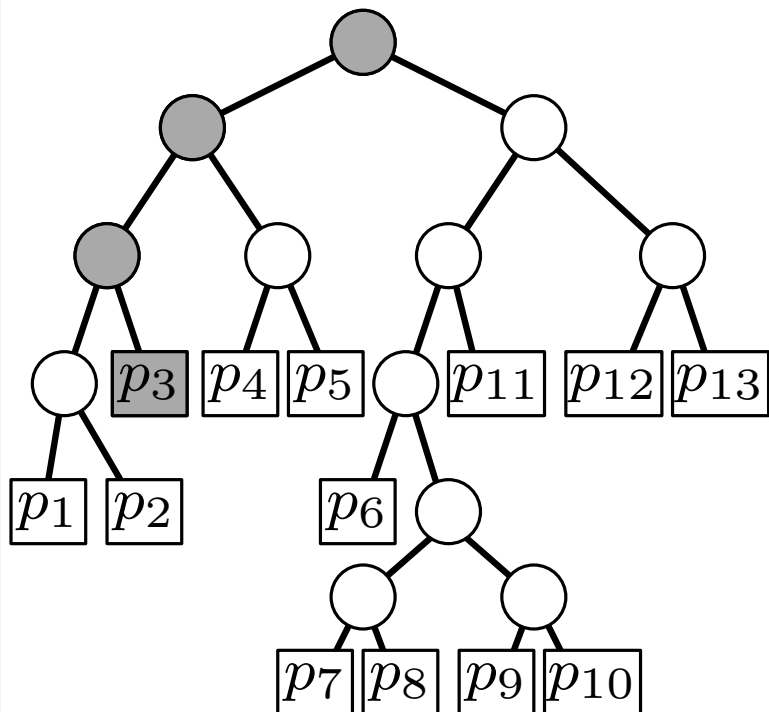
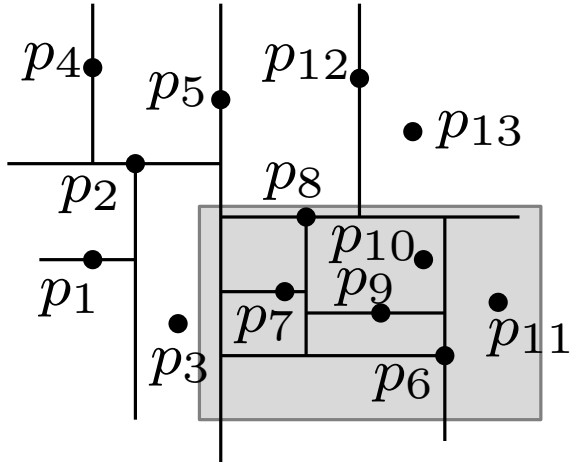


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

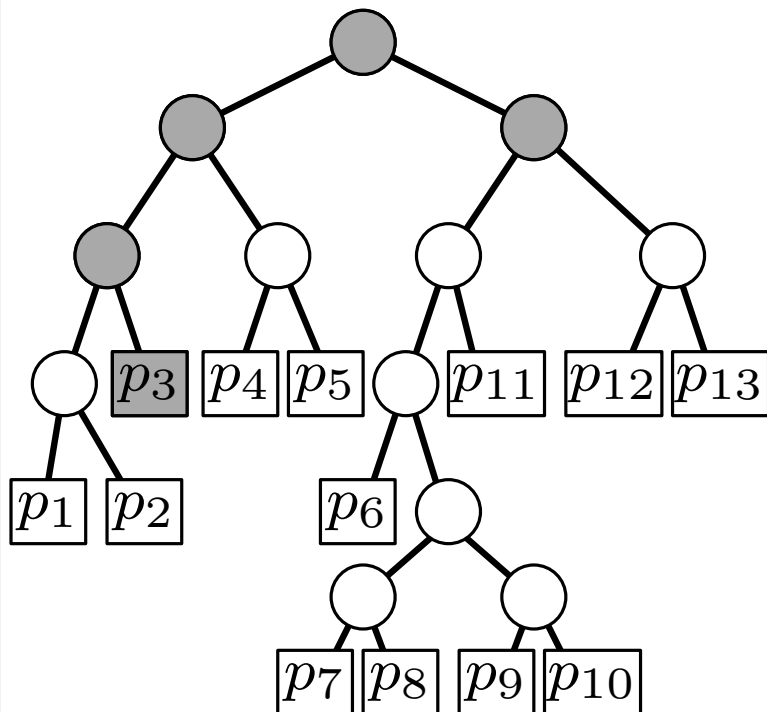
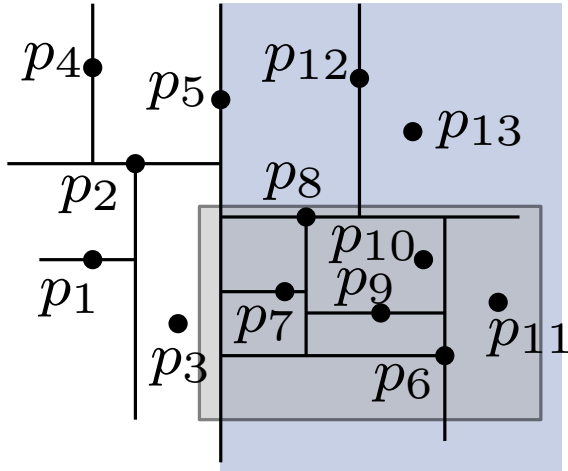


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

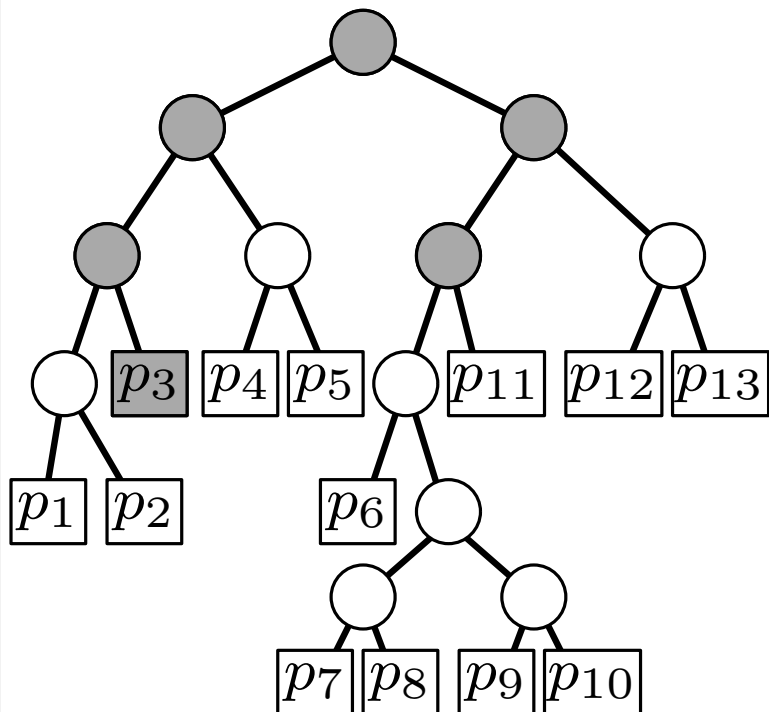
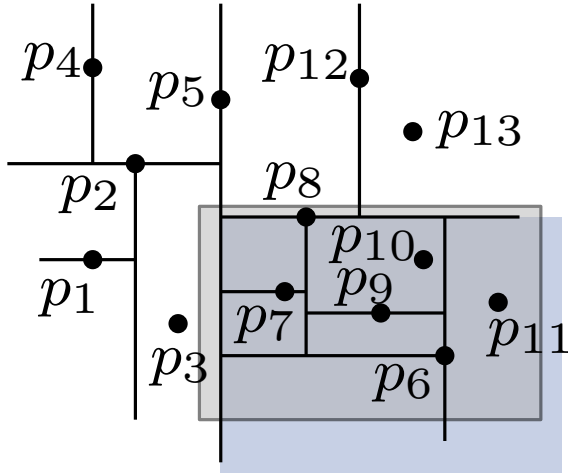


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    | prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    | if region(lc( $v$ ))  $\subseteq R$  then
    | | ReportSubtree(lc( $v$ ))
    | else
    | | if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
    | | | SearchKdTree(lc( $v$ ),  $R$ )
    | if region(rc( $v$ ))  $\subseteq R$  then
    | | ReportSubtree(rc( $v$ ))
    | else
    | | if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
    | | | SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

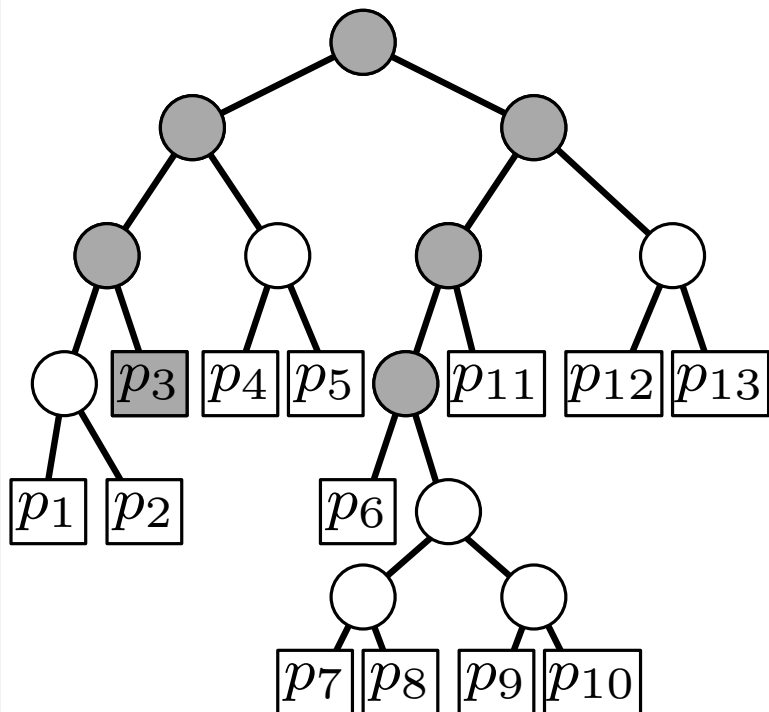
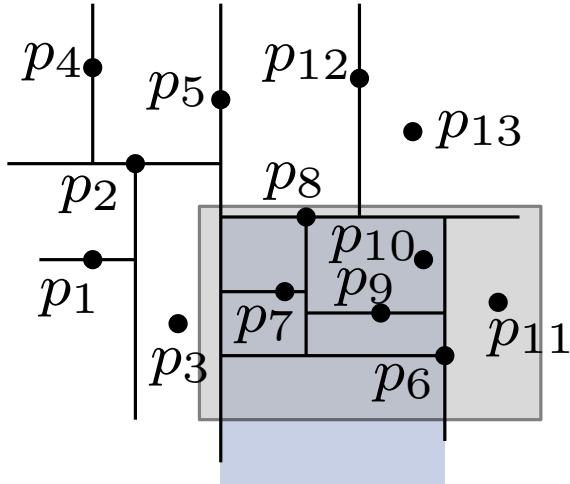


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if  $\text{region}(\text{lc}(v)) \subseteq R$  then
        ReportSubtree( $\text{lc}(v)$ )
    else
        if  $\text{region}(\text{lc}(v)) \cap R \neq \emptyset$  then
            SearchKdTree( $\text{lc}(v), R$ )
    if  $\text{region}(\text{rc}(v)) \subseteq R$  then
        ReportSubtree( $\text{rc}(v)$ )
    else
        if  $\text{region}(\text{rc}(v)) \cap R \neq \emptyset$  then
            SearchKdTree( $\text{rc}(v), R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

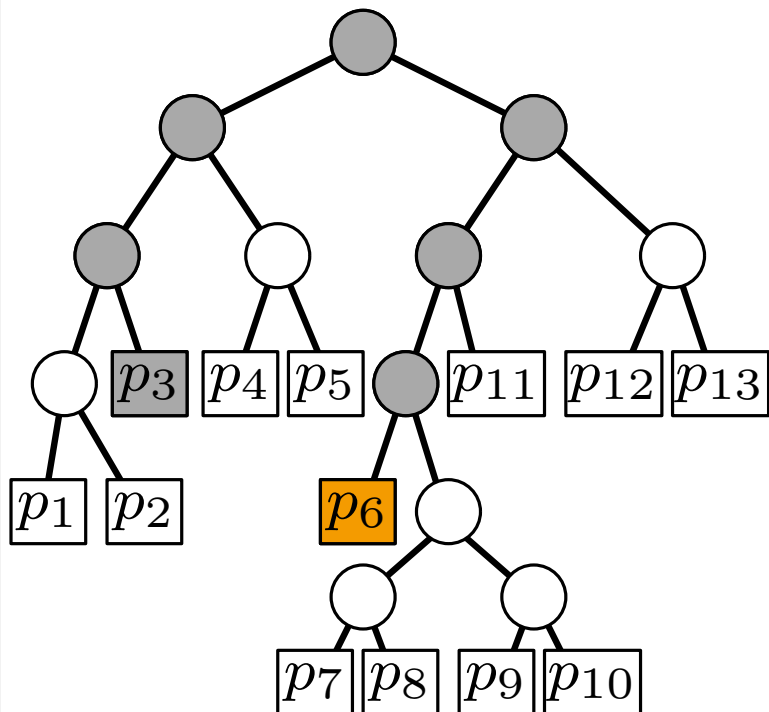
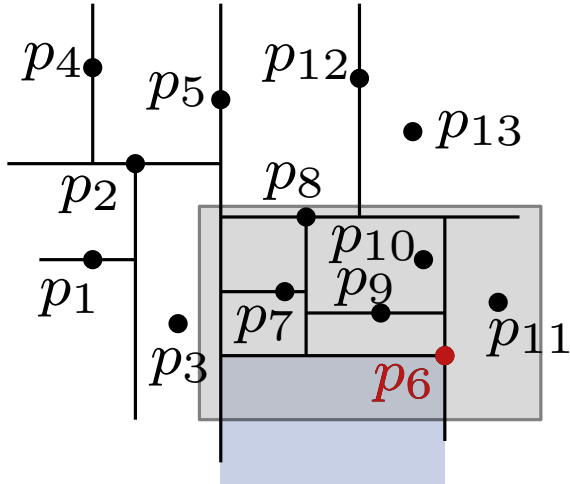


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    | prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    | if  $\text{region}(\text{lc}(v)) \subseteq R$  then
    | | ReportSubtree( $\text{lc}(v)$ )
    | else
    | | if  $\text{region}(\text{lc}(v)) \cap R \neq \emptyset$  then
    | | | SearchKdTree( $\text{lc}(v), R$ )
    | if  $\text{region}(\text{rc}(v)) \subseteq R$  then
    | | ReportSubtree( $\text{rc}(v)$ )
    | else
    | | if  $\text{region}(\text{rc}(v)) \cap R \neq \emptyset$  then
    | | | SearchKdTree( $\text{rc}(v), R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

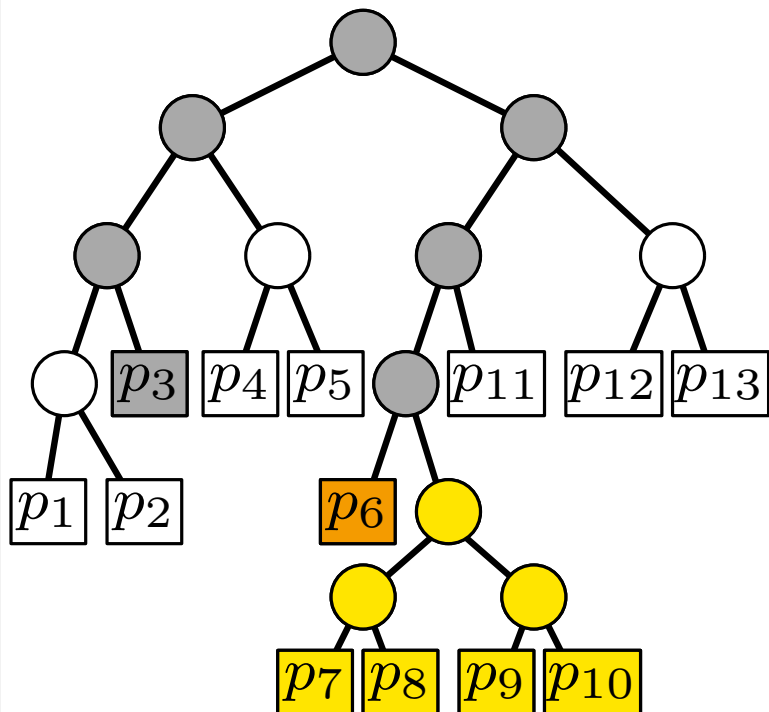
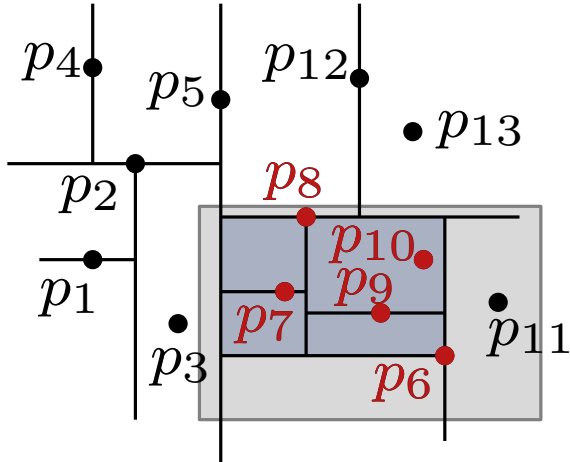


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    | prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    | if  $\text{region}(\text{lc}(v)) \subseteq R$  then
    | | ReportSubtree( $\text{lc}(v)$ )
    | else
    | | if  $\text{region}(\text{lc}(v)) \cap R \neq \emptyset$  then
    | | | SearchKdTree( $\text{lc}(v), R$ )
    | if  $\text{region}(\text{rc}(v)) \subseteq R$  then
    | | ReportSubtree( $\text{rc}(v)$ )
    | else
    | | if  $\text{region}(\text{rc}(v)) \cap R \neq \emptyset$  then
    | | | SearchKdTree( $\text{rc}(v), R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

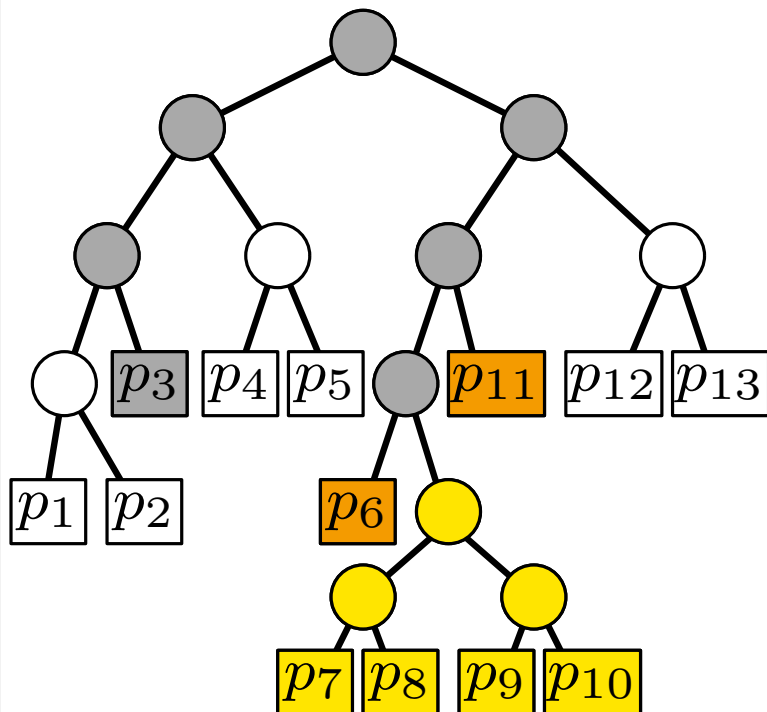
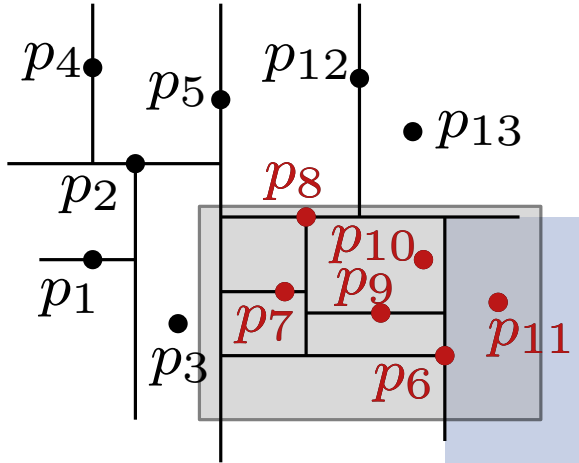


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree



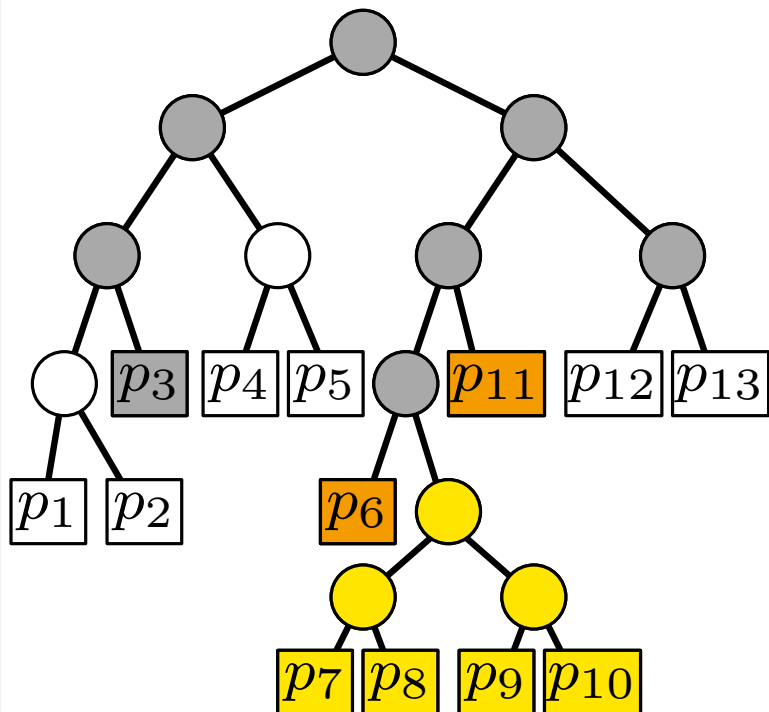
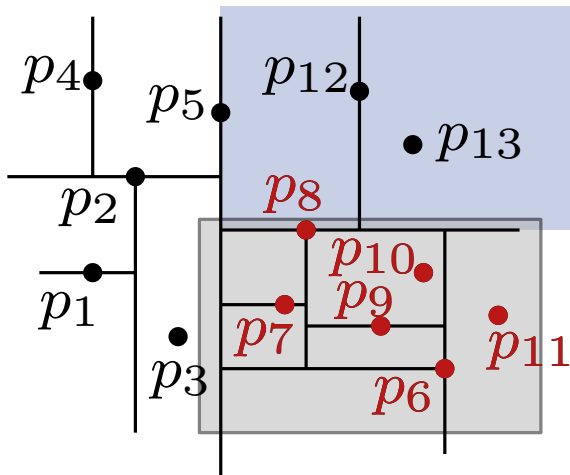
SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```



# Bereichsabfrage in einem $kd$ -Tree

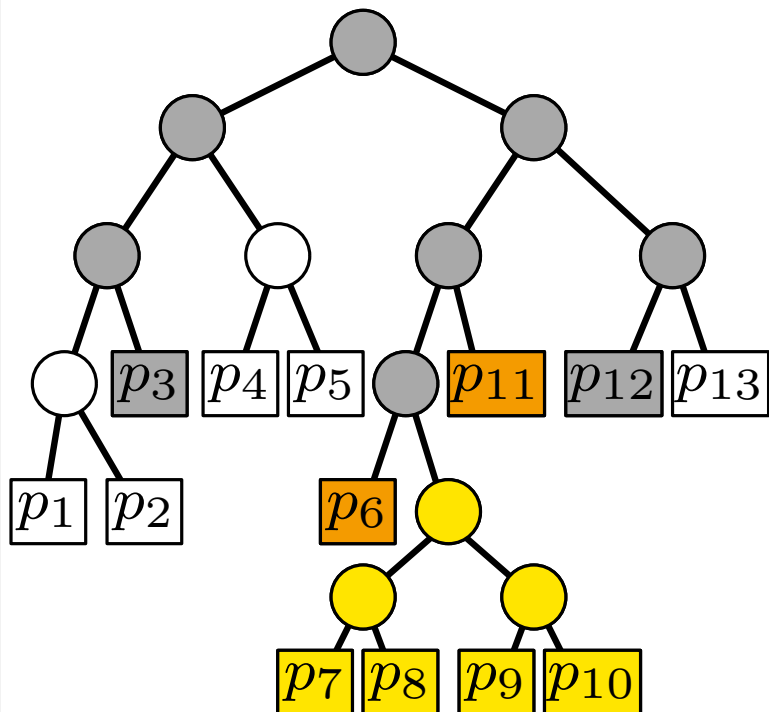
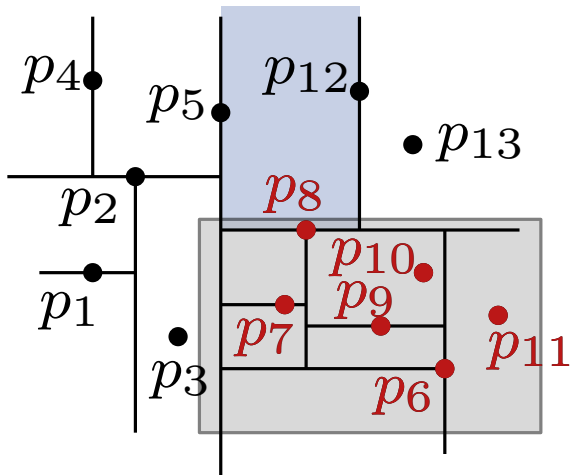


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

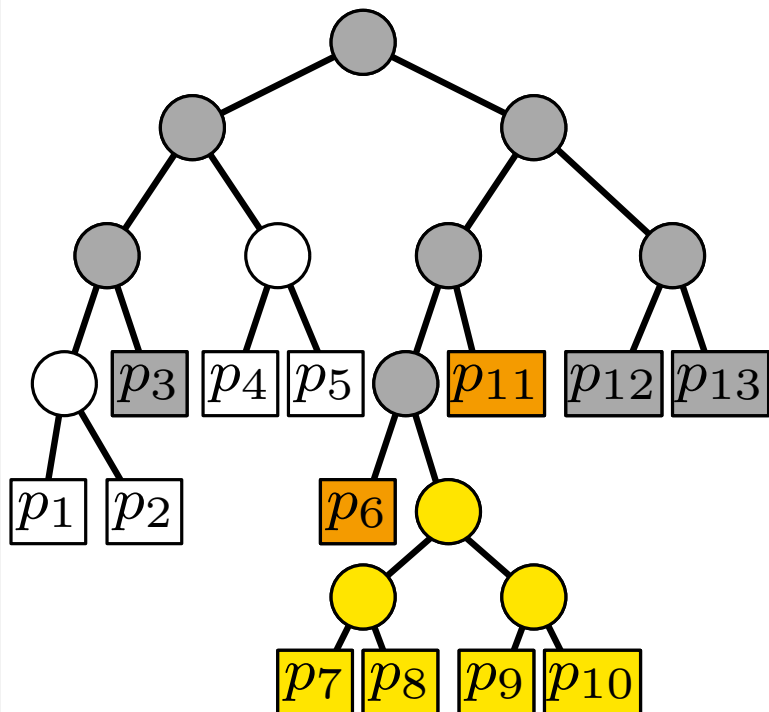
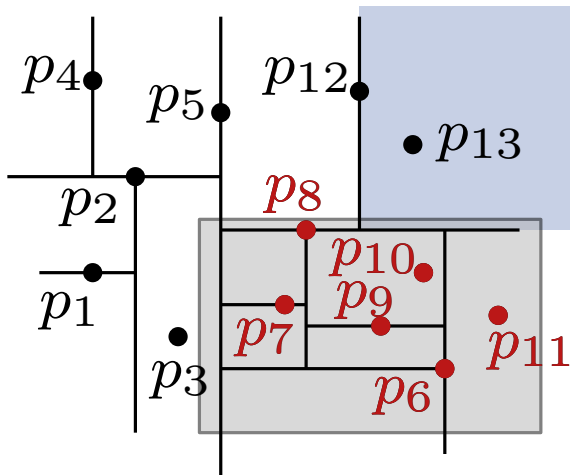


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

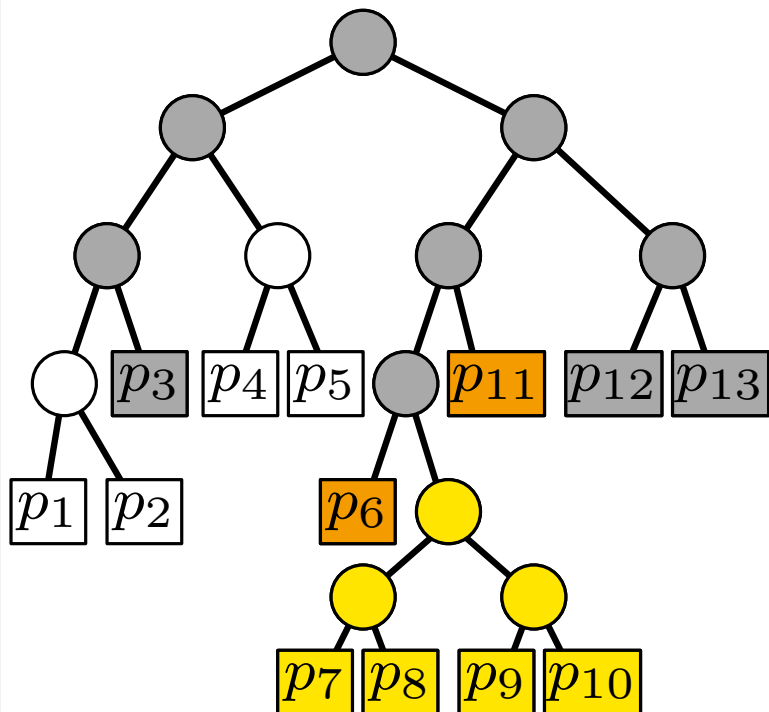
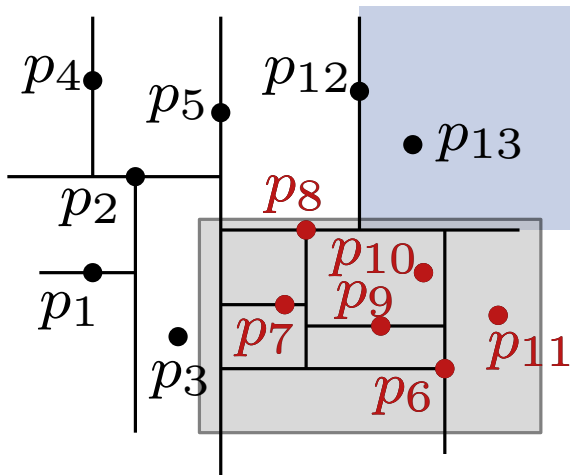


SearchKdTree( $v, R$ )

```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

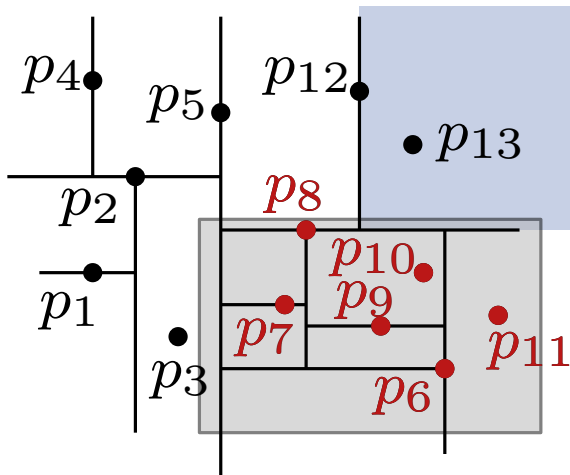


SearchKdTree( $v, R$ )

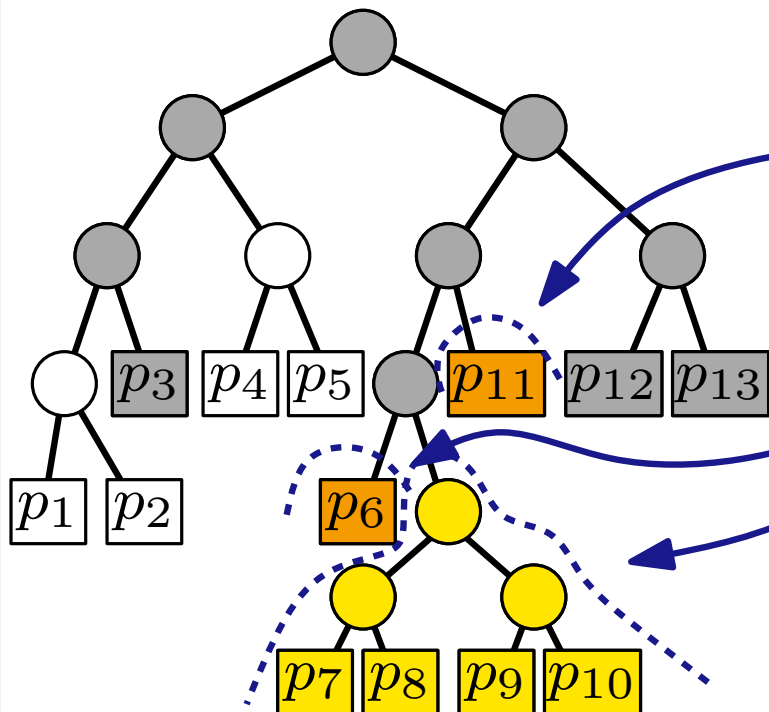
```

if  $v$  Blatt then
    prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    if region(lc( $v$ ))  $\subseteq R$  then
        ReportSubtree(lc( $v$ ))
    else
        if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(lc( $v$ ),  $R$ )
    if region(rc( $v$ ))  $\subseteq R$  then
        ReportSubtree(rc( $v$ ))
    else
        if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
            SearchKdTree(rc( $v$ ),  $R$ )
    
```

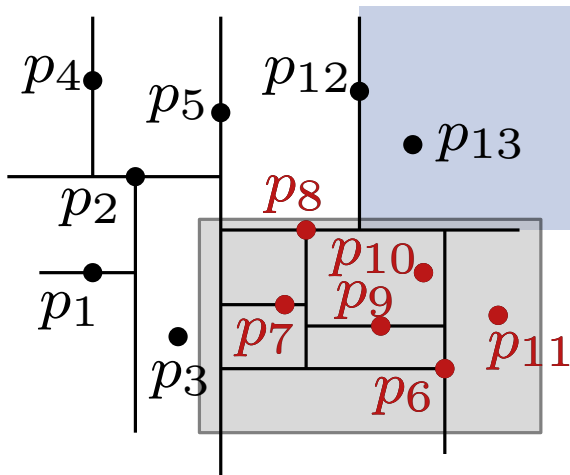
# Bereichsabfrage in einem $kd$ -Tree



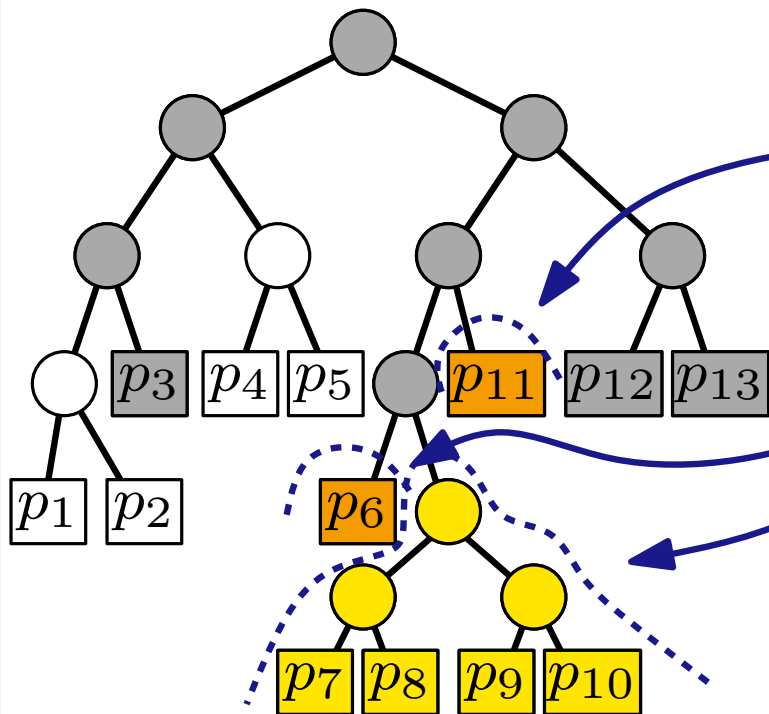
ReportSubtree in  $\mathcal{O}(k)$



# Bereichsabfrage in einem $kd$ -Tree

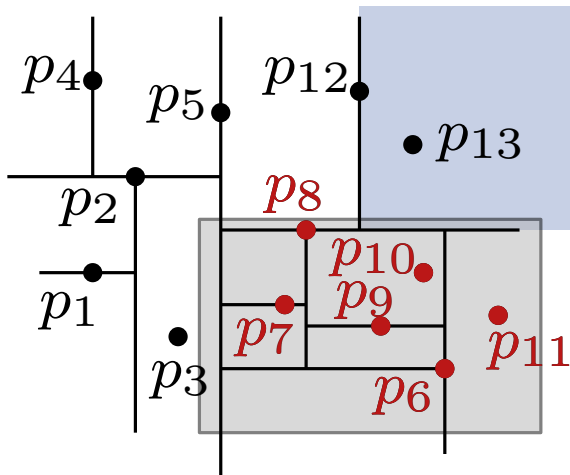


ReportSubtree in  $\mathcal{O}(k)$



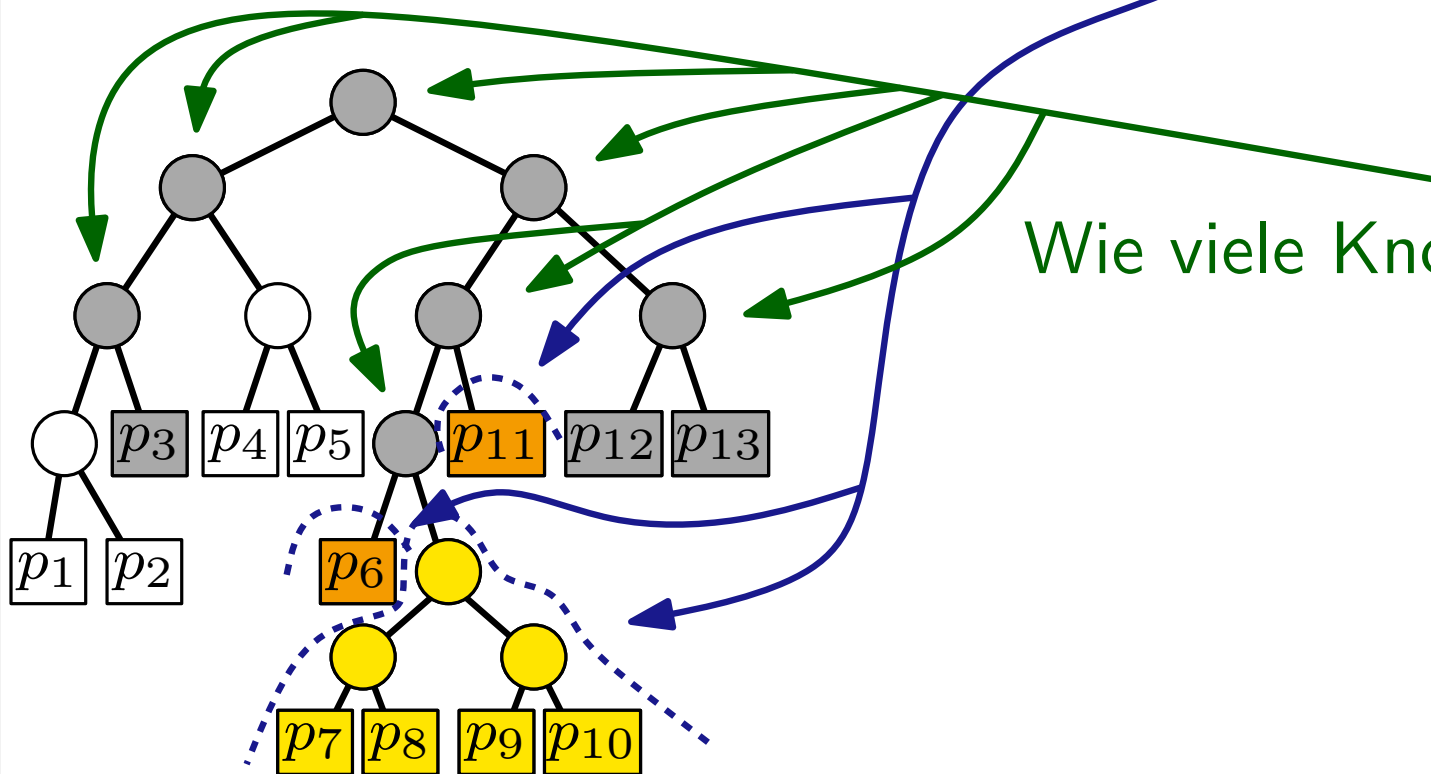
Wie viele Knoten betrachten wir?

# Bereichsabfrage in einem $kd$ -Tree

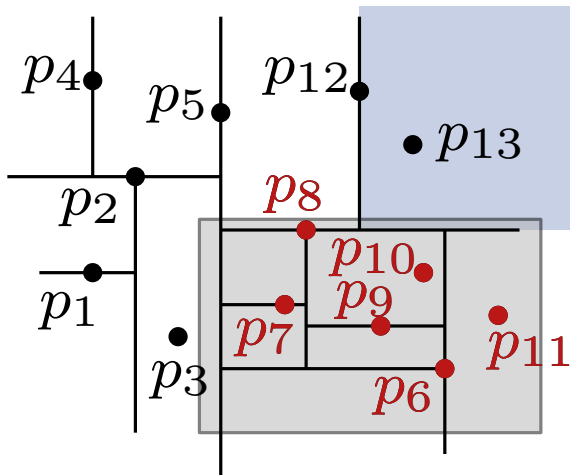


ReportSubtree in  $\mathcal{O}(k)$

Wie viele Knoten betrachten wir?

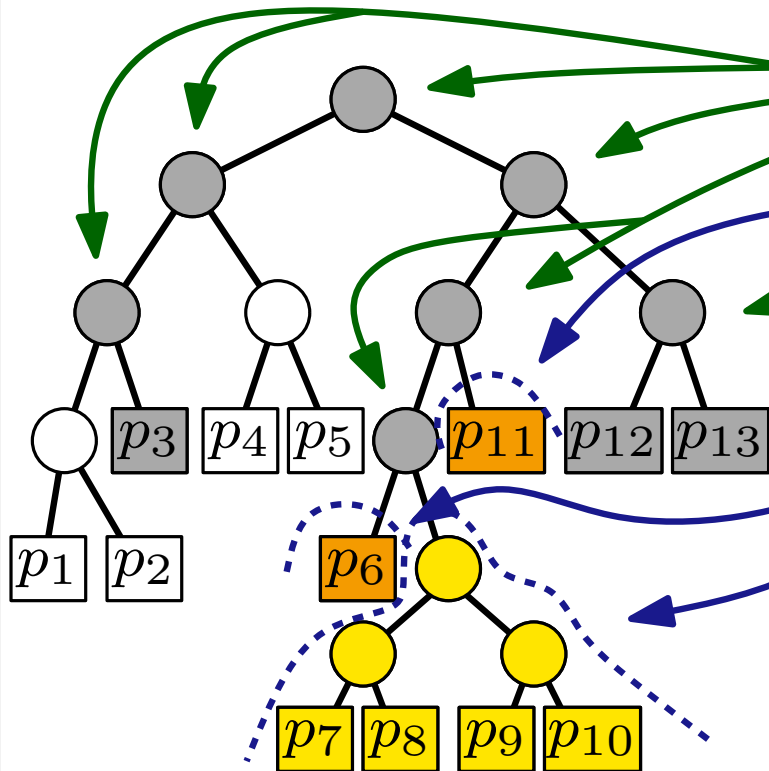


# Bereichsabfrage in einem $kd$ -Tree



ReportSubtree in  $\mathcal{O}(k)$

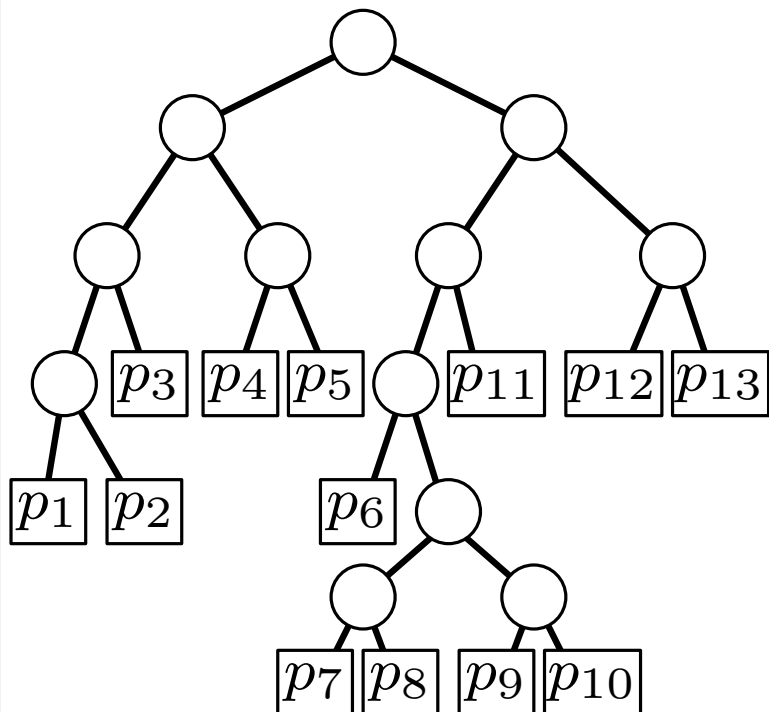
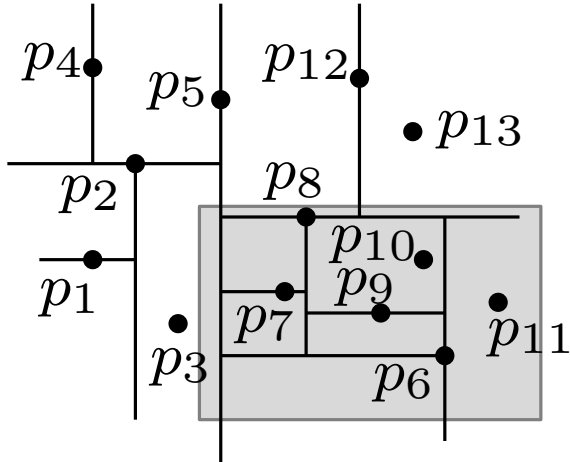
Wie viele Knoten betrachten wir?



jeder Knoten  $\triangleq$  geschnittene Region



# Bereichsabfrage in einem $kd$ -Tree

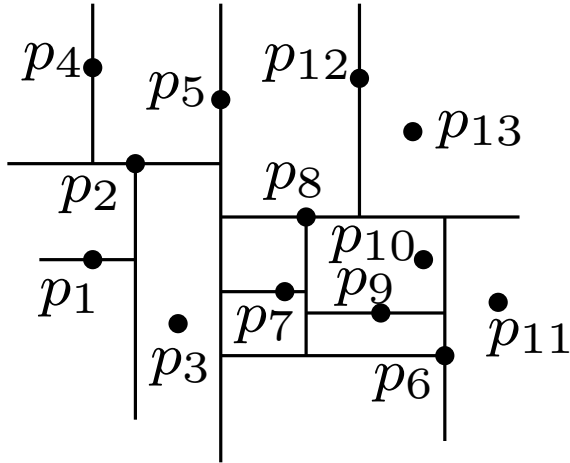


SearchKdTree( $v, R$ )

```

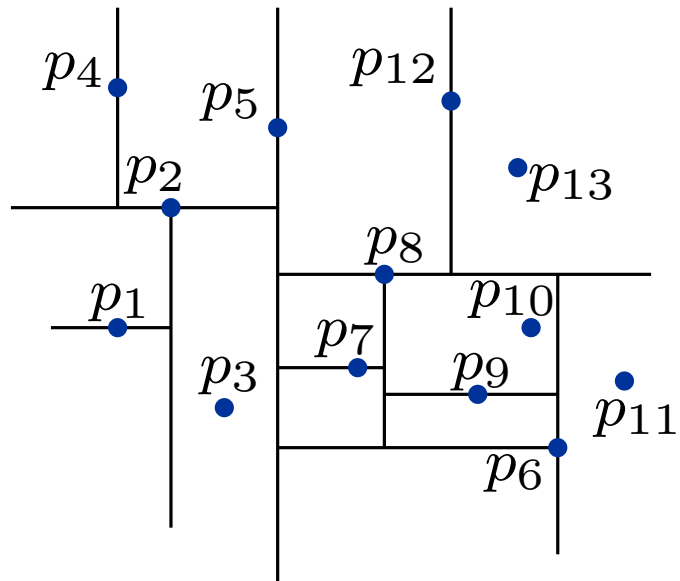
if  $v$  Blatt then
    | prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
    | if region(lc( $v$ ))  $\subseteq R$  then
    | | ReportSubtree(lc( $v$ ))
    | else
    | | if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
    | | | SearchKdTree(lc( $v$ ),  $R$ )
    | if region(rc( $v$ ))  $\subseteq R$  then
    | | ReportSubtree(rc( $v$ ))
    | else
    | | if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
    | | | SearchKdTree(rc( $v$ ),  $R$ )
    
```

# Bereichsabfrage in einem $kd$ -Tree

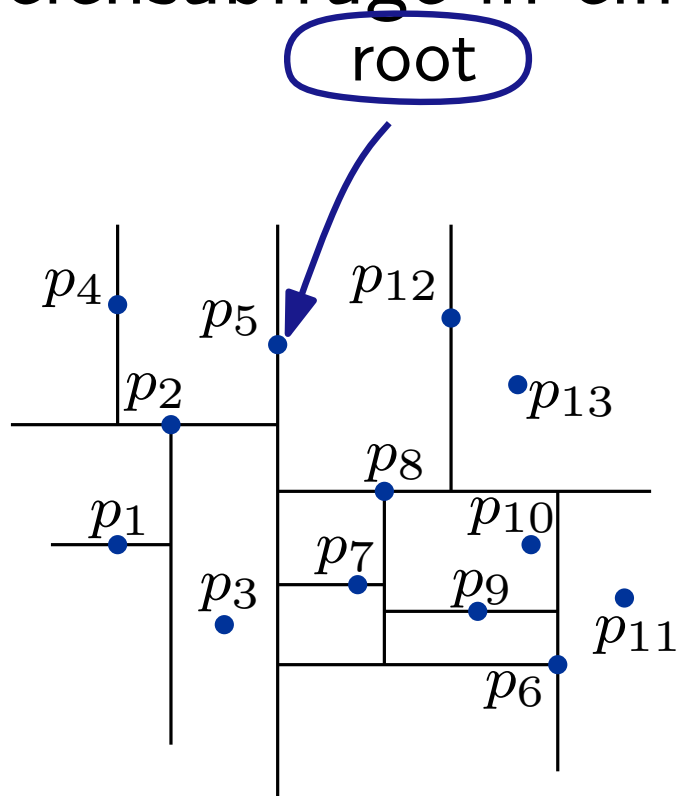


**Wie viele Regionen schneidet eine vertikale Linie?**

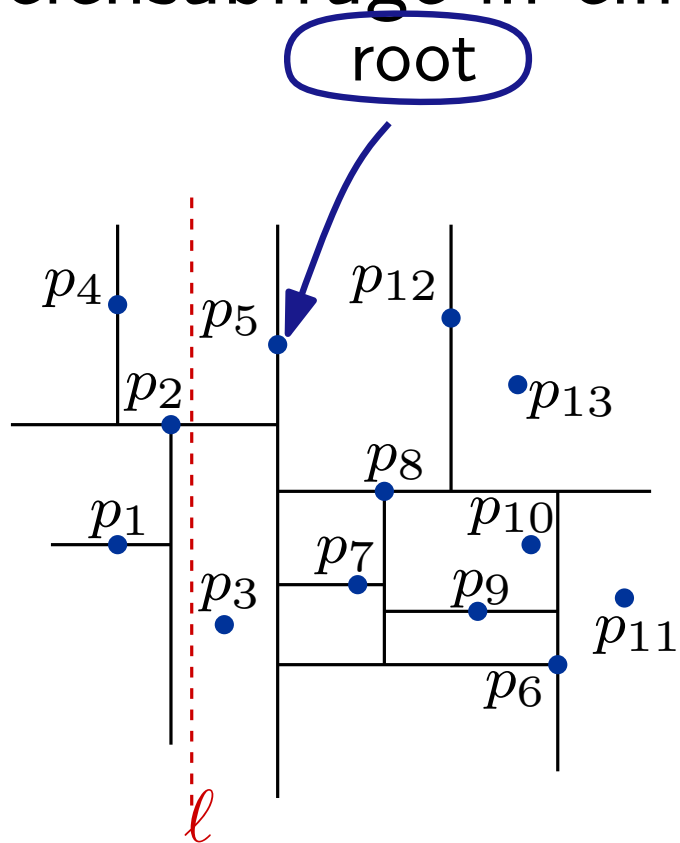
# Bereichsabfrage in einem $kd$ -Tree



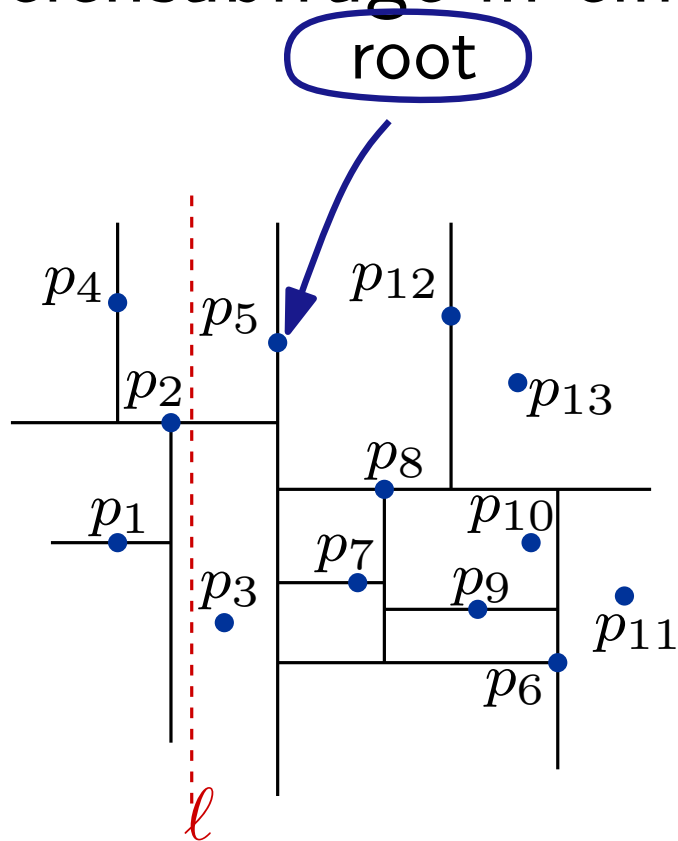
# Bereichsabfrage in einem $kd$ -Tree



# Bereichsabfrage in einem $kd$ -Tree



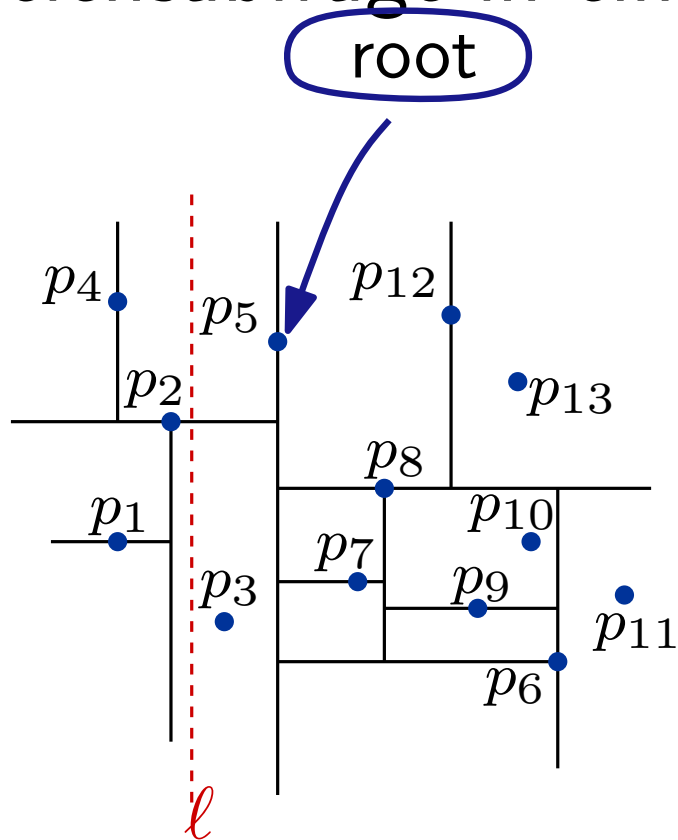
# Bereichsabfrage in einem $kd$ -Tree



**Vermutung:**

$$Q(n) = 1 + Q(n/2)$$

# Bereichsabfrage in einem $kd$ -Tree



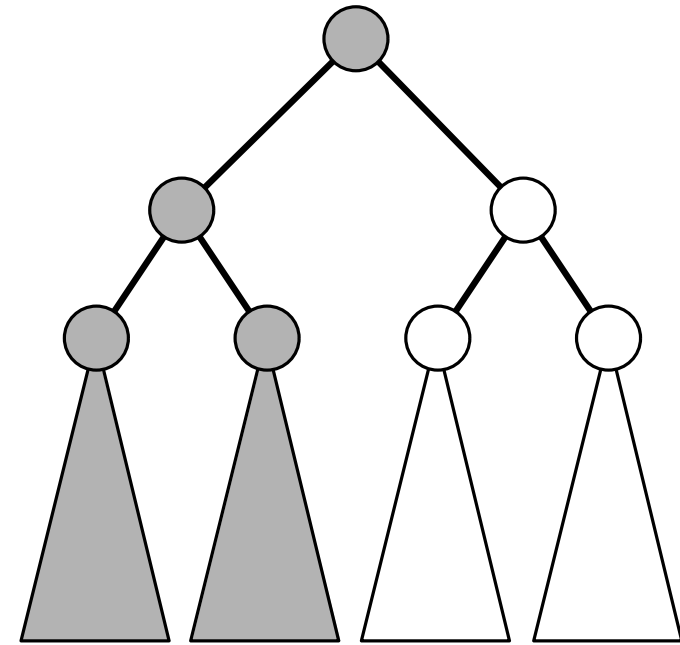
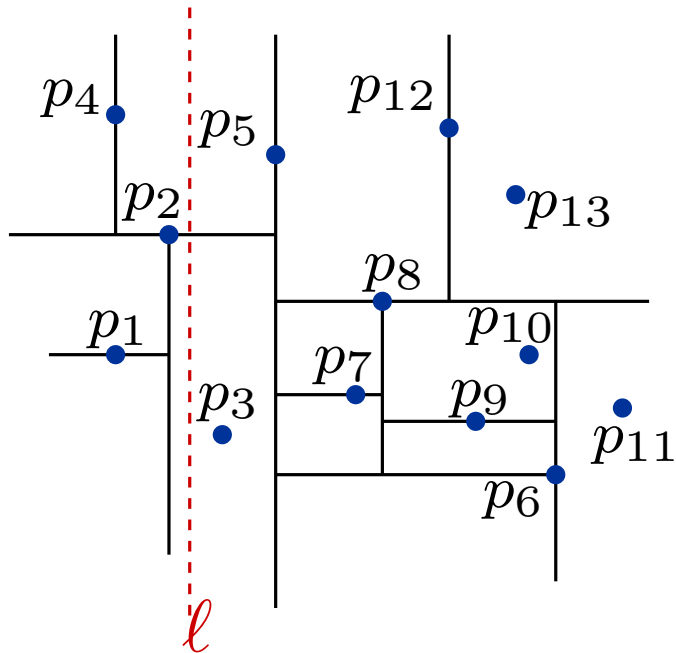
**Vermutung:**

$$Q(n) = 1 + Q(n/2)$$

**Problem?**

$l$  schneidet beide Kinder des linken Kindes von root

# Bereichsabfrage in einem $kd$ -Tree



**Vermutung:**

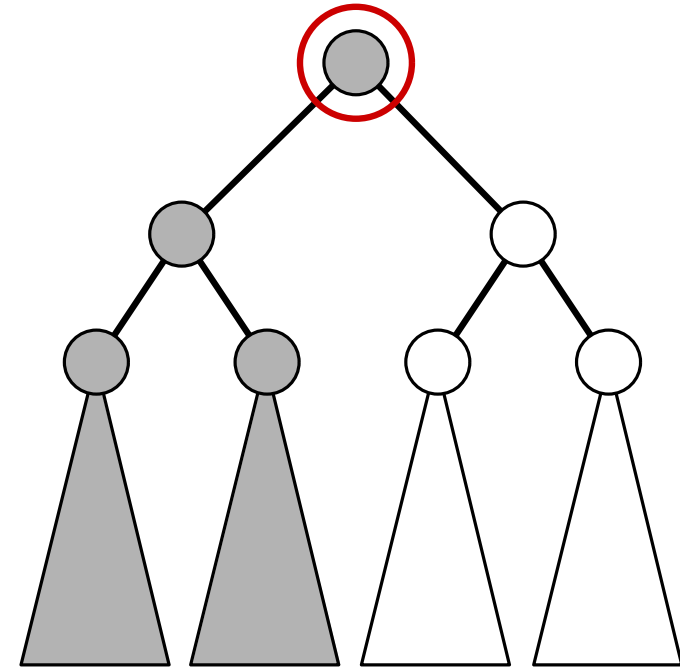
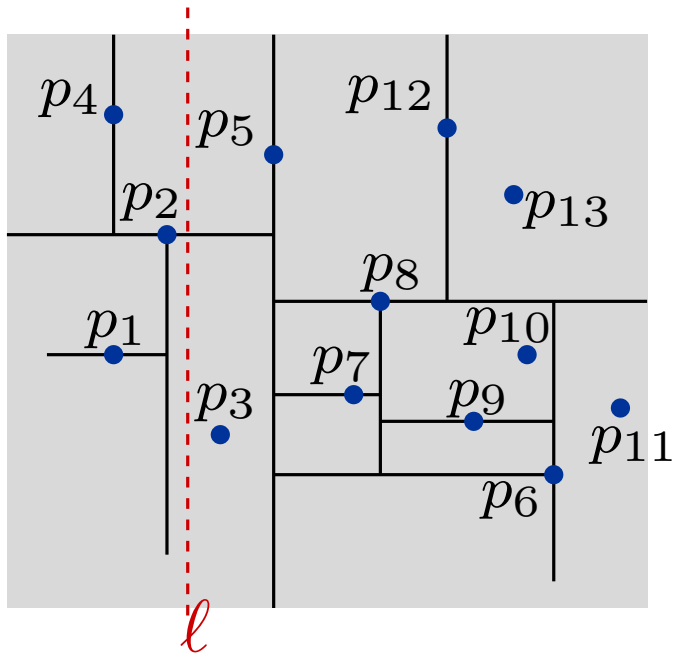
$$Q(n) = 1 + Q(n/2)$$

**Problem?**

$l$  schneidet beide Kinder des linken Kindes von root



# Bereichsabfrage in einem $kd$ -Tree



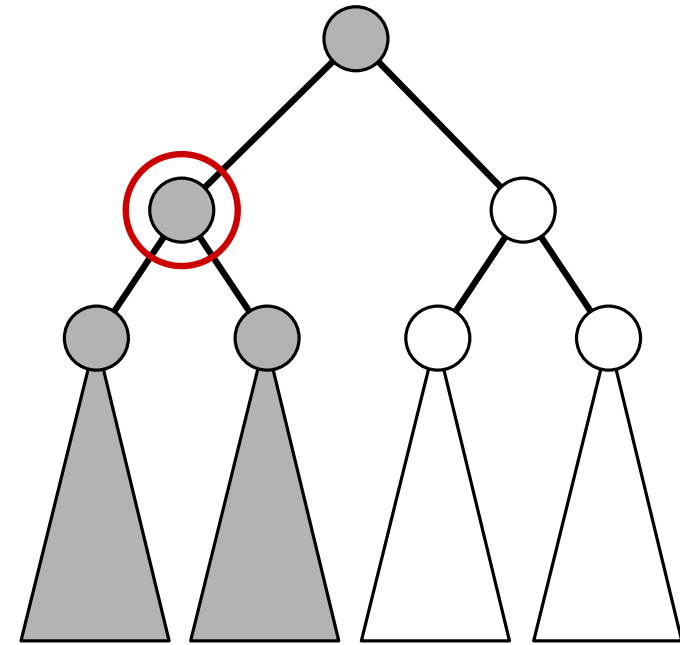
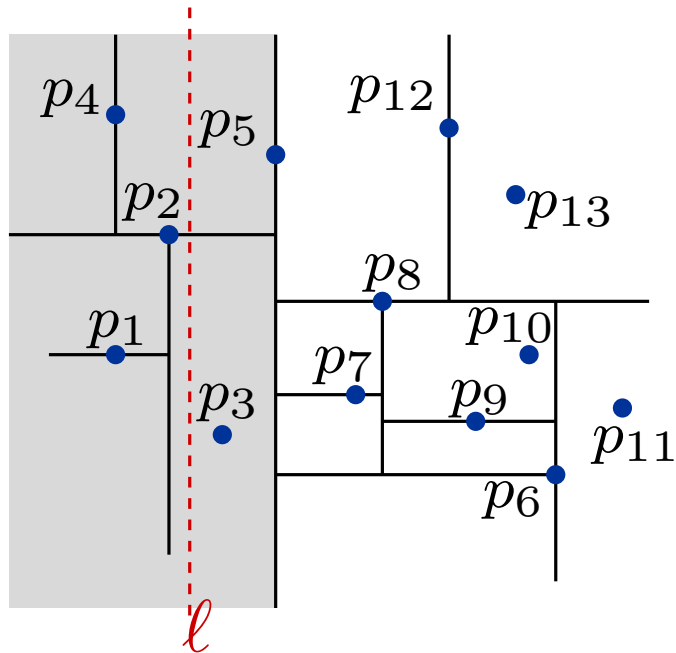
**Vermutung:**

$$Q(n) = 1 + Q(n/2)$$

**Problem?**

$\ell$  schneidet beide Kinder des linken Kindes von root

# Bereichsabfrage in einem $kd$ -Tree



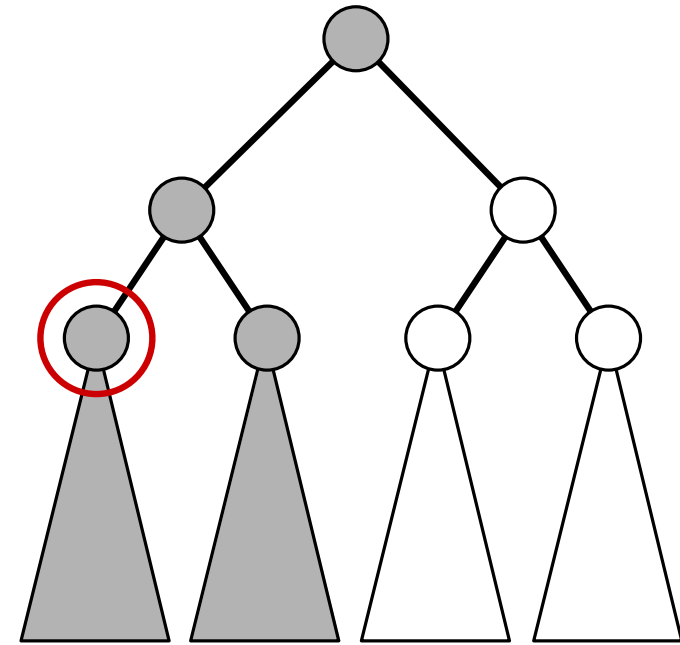
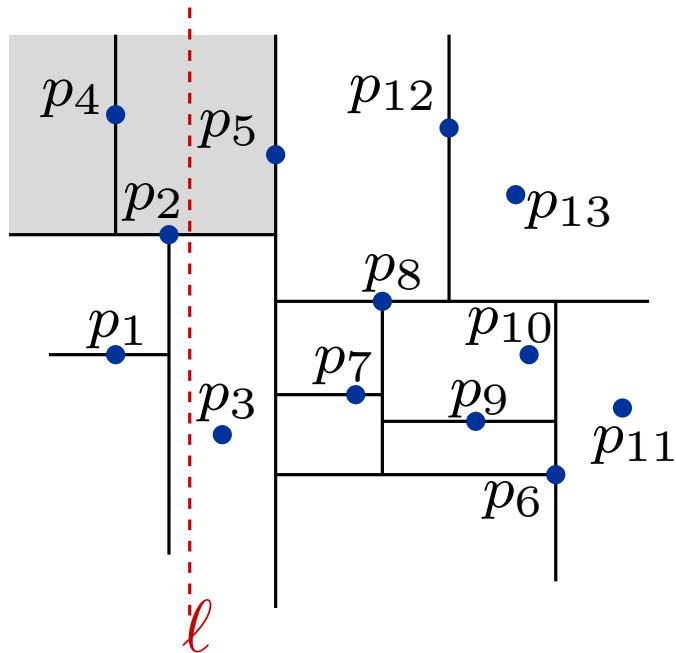
**Vermutung:**

$$Q(n) = 1 + Q(n/2)$$

**Problem?**

$l$  schneidet beide Kinder des linken Kindes von root

# Bereichsabfrage in einem $kd$ -Tree



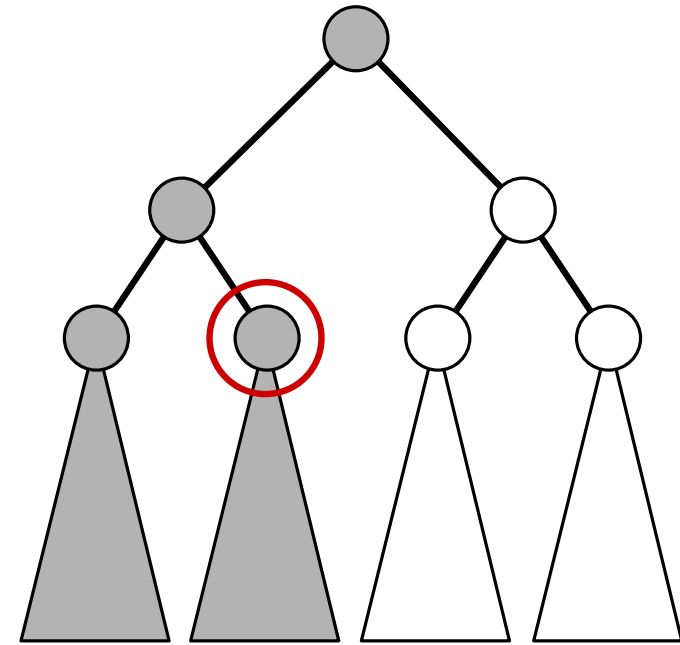
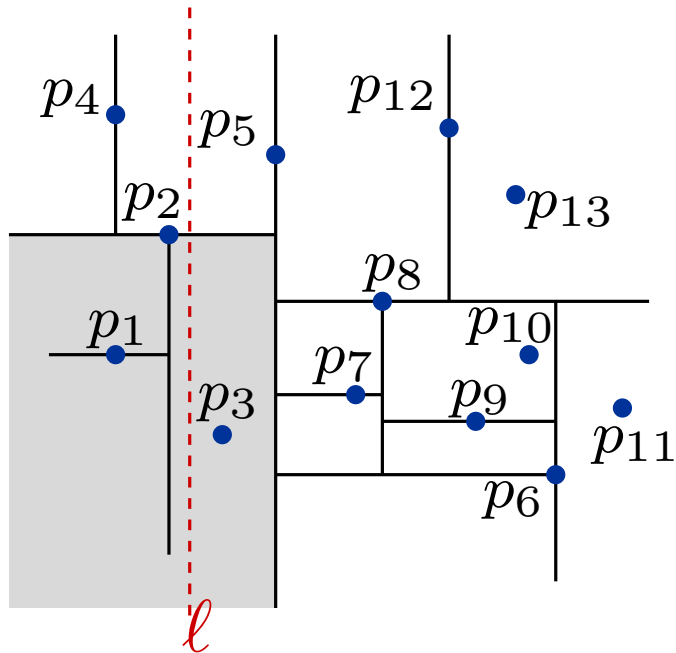
**Vermutung:**

$$Q(n) = 1 + Q(n/2)$$

**Problem?**

$l$  schneidet beide Kinder des linken Kindes von root

# Bereichsabfrage in einem $kd$ -Tree



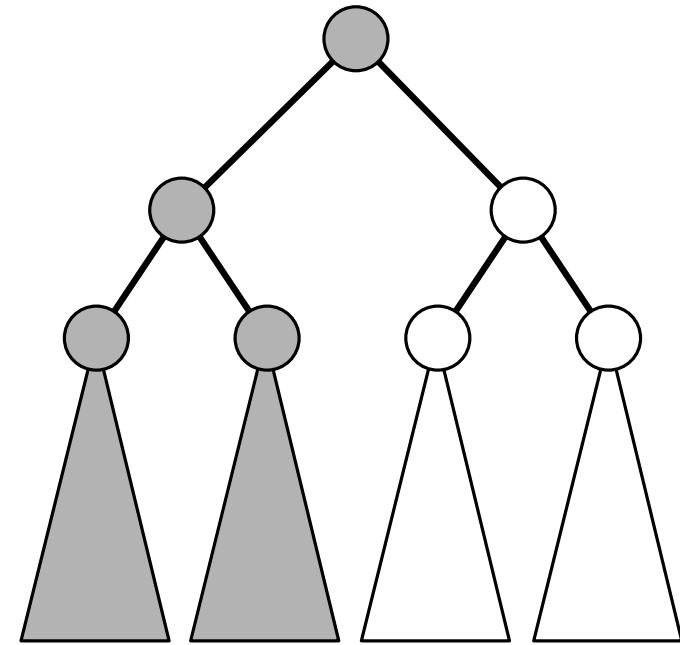
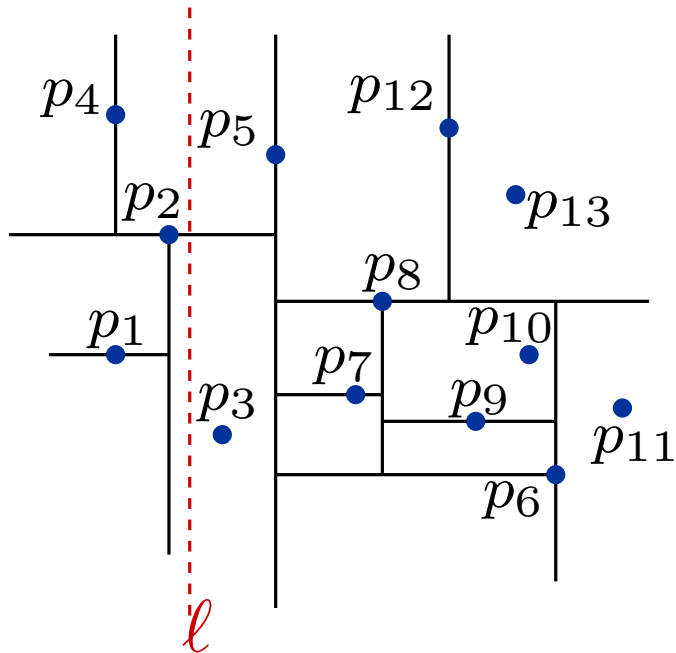
**Vermutung:**

$$Q(n) = 1 + Q(n/2)$$

**Problem?**

$l$  schneidet beide Kinder des linken Kindes von root

# Bereichsabfrage in einem $kd$ -Tree



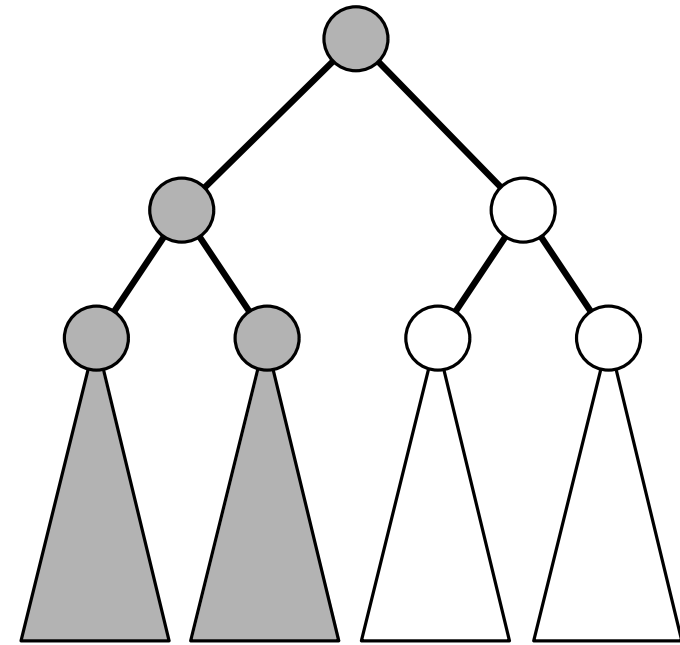
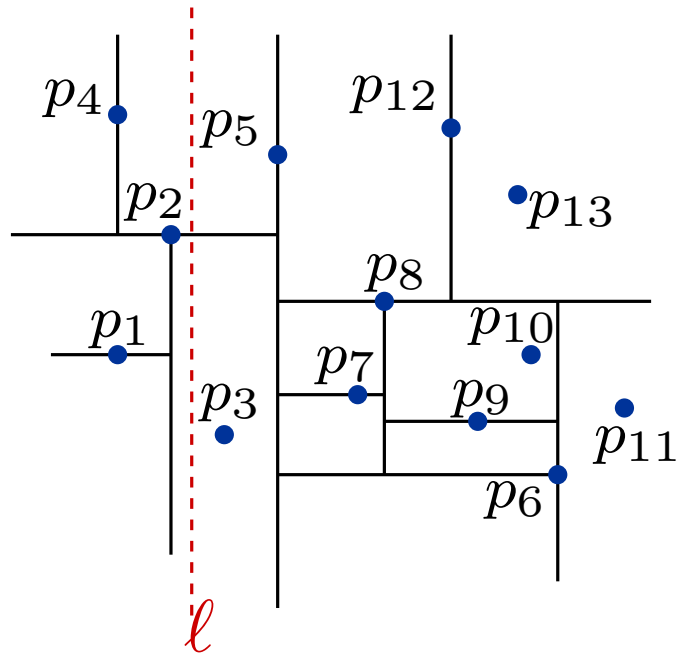
**Vermutung:**

$$Q(n) = 1 + Q(n/2)$$

**Problem?**

$l$  schneidet beide Kinder des linken Kindes von root

# Bereichsabfrage in einem $kd$ -Tree



**Vermutung:**

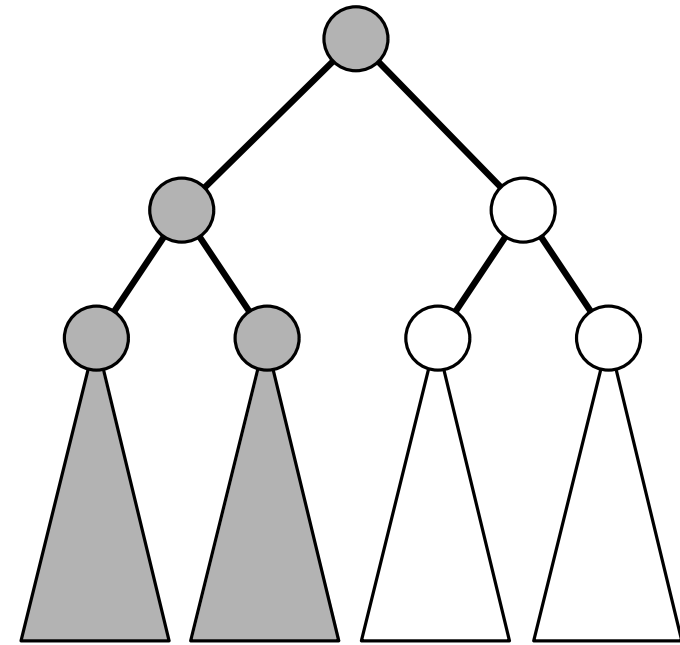
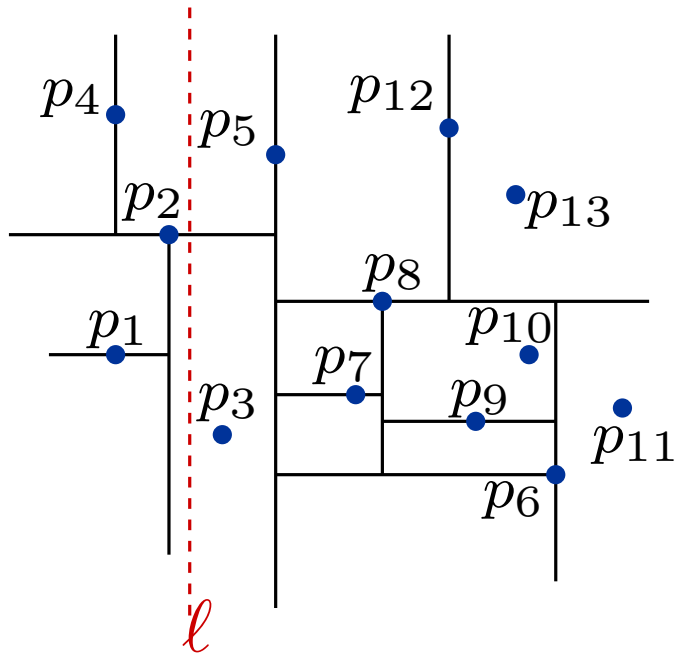
$$Q(n) = 1 + Q(n/2)$$

**Problem?**

$l$  schneidet beide Kinder des linken Kindes von root

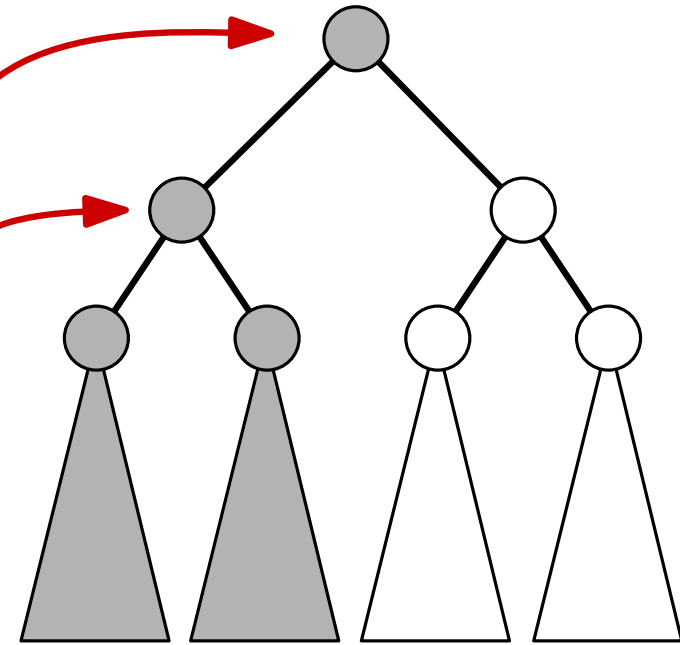
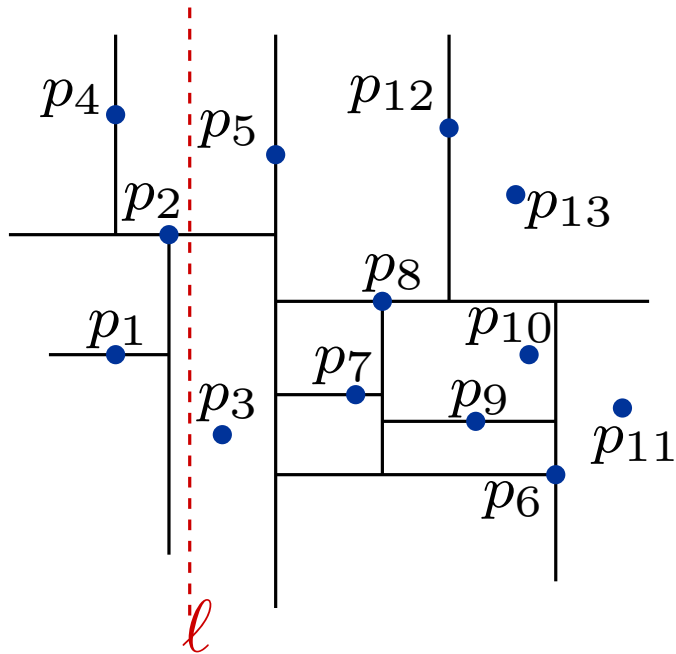
**Gleiche Rekursionsituation**

# Bereichsabfrage in einem $kd$ -Tree



$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$

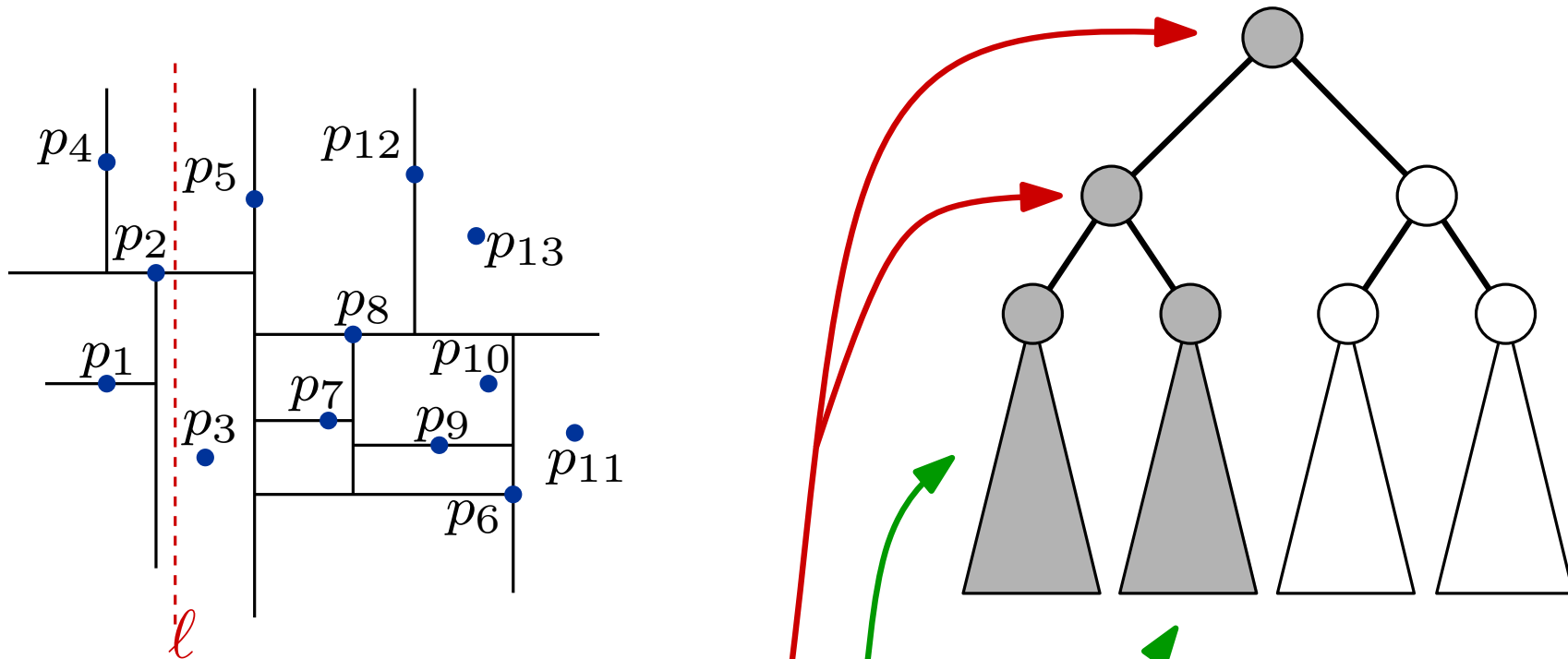
# Bereichsabfrage in einem $kd$ -Tree



$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$



# Bereichsabfrage in einem $kd$ -Tree



$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$

# Aufgabe 1

*kd*-Trees – w-c Laufzeit

## Anfragen

a) warum?

$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$

# Aufgabe 1

*kd*-Trees – w-c Laufzeit

## Anfragen

a) warum?

$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$

b) Rekurrenz auflösen:  $Q(n) = \mathcal{O}(\sqrt{n})$ .

# Aufgabe 1

*kd*-Trees – w-c Laufzeit

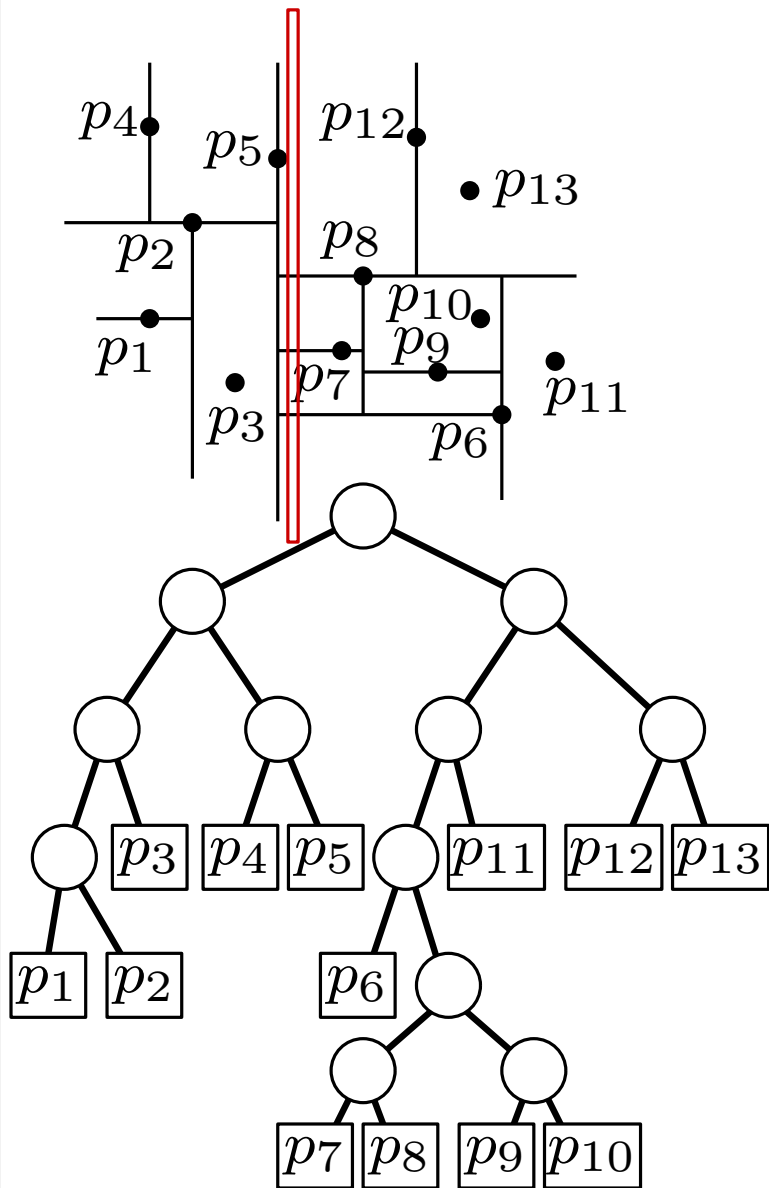
## Anfragen

a) warum?

$$Q(n) = \begin{cases} \mathcal{O}(1) & , \text{ für } n = 1 \\ 2 + 2Q(n/4) & , \text{ für } n > 1 \end{cases}$$

b) Rekurrenz auflösen:  $Q(n) = \mathcal{O}(\sqrt{n})$ .

c)  $\Omega(\sqrt{n})$  untere Schranke für Bereichsanfragen in *kd*-Trees



SearchKdTree( $v, R$ )

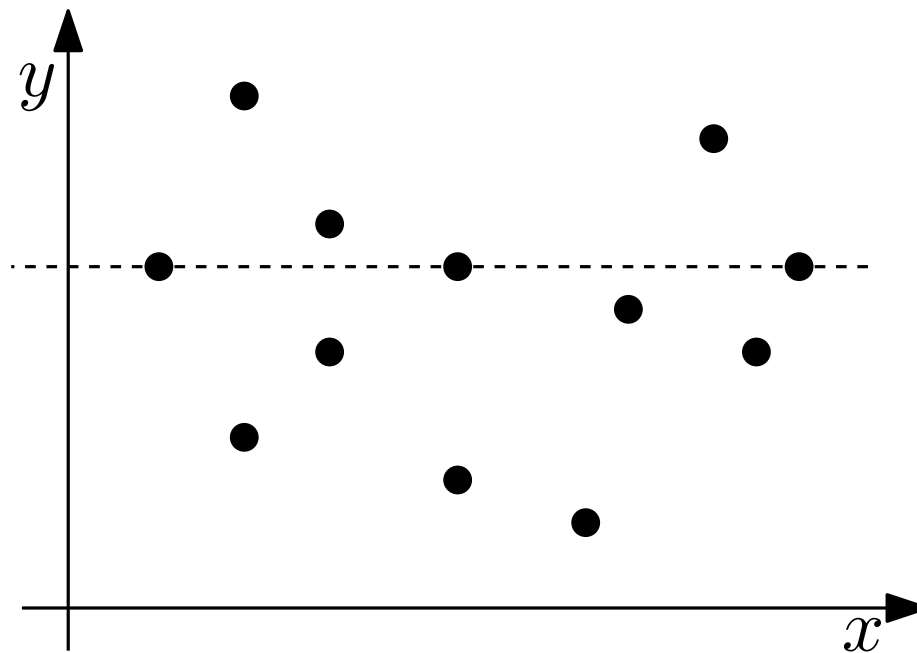
```

if  $v$  Blatt then
  | prüfe Punkt  $p$  in  $v$  auf  $p \in R$ 
else
  | if region(lc( $v$ ))  $\subseteq R$  then
  | | ReportSubtree(lc( $v$ ))
  | else
  | | if region(lc( $v$ ))  $\cap R \neq \emptyset$  then
  | | | SearchKdTree(lc( $v$ ),  $R$ )
  | if region(rc( $v$ ))  $\subseteq R$  then
  | | ReportSubtree(rc( $v$ ))
  | else
  | | if region(rc( $v$ ))  $\cap R \neq \emptyset$  then
  | | | SearchKdTree(rc( $v$ ),  $R$ )
  
```

c)  $\Omega(\sqrt{n})$  untere Schranke für Bereichsanfragen in  $kd$ -Trees

# Aufgabe 2

*'partial match queries'*

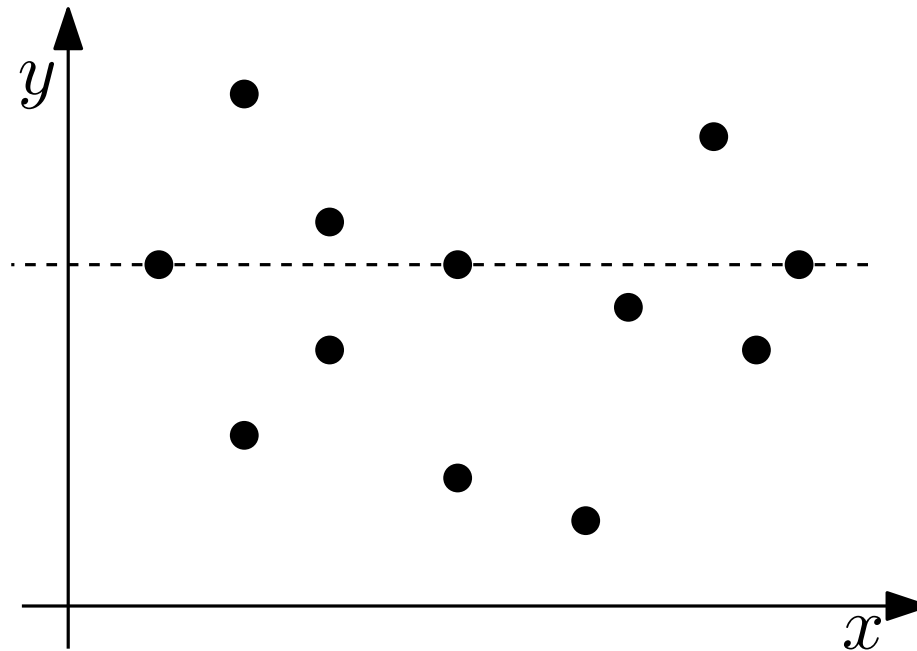


Gebe mir alle Punkte mit  $y = 7$ .

---

# Aufgabe 2

*'partial match queries'*



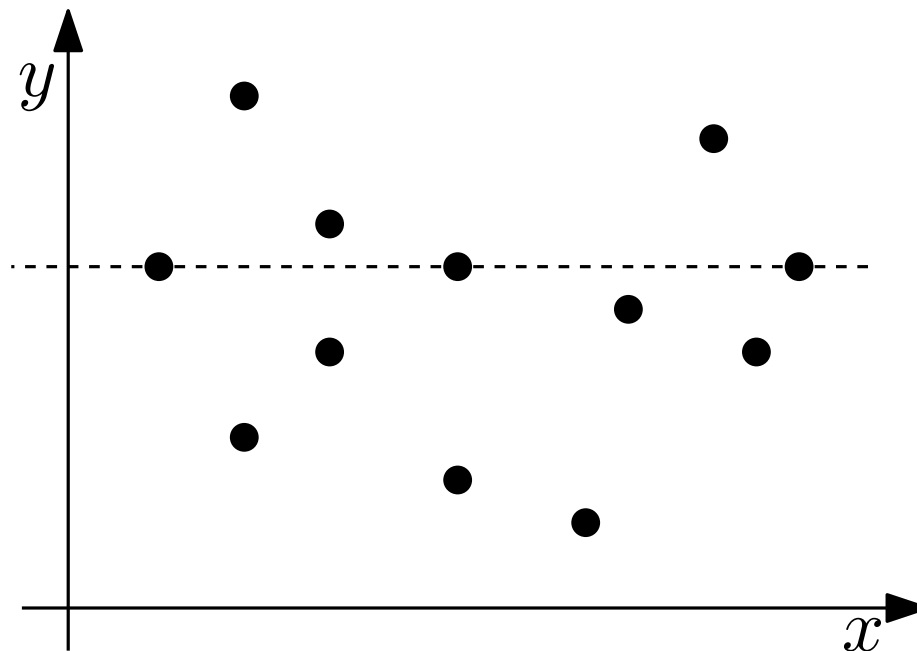
Gebe mir alle Punkte mit  $y = 7$ .

---

a) Wie kann man *kd*-Trees dafür nutzen?

# Aufgabe 2

*'partial match queries'*



Gebe mir alle Punkte mit  $y = 7$ .

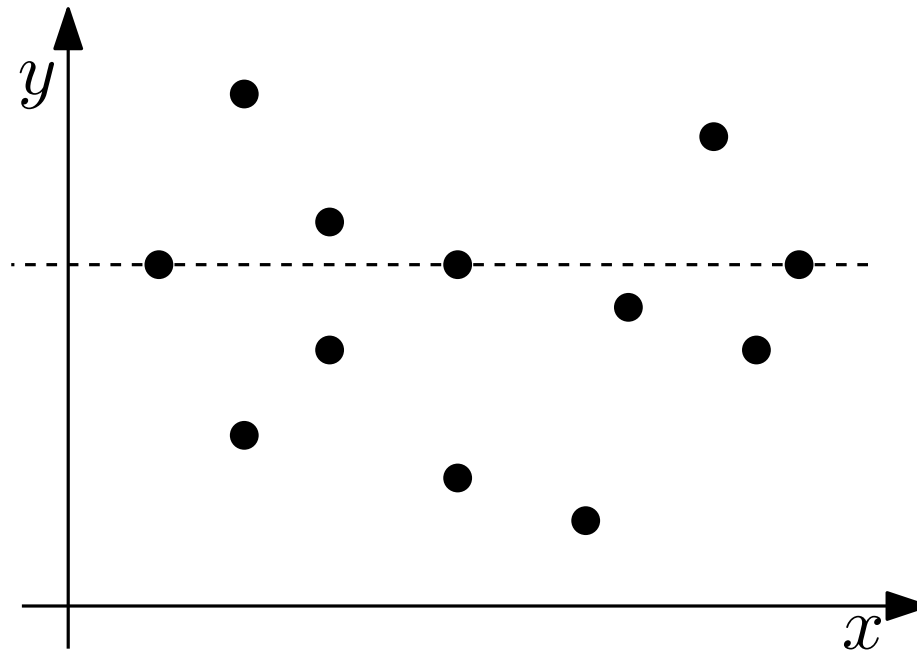
---

b) Wie kann man range-Trees dafür nutzen?



# Aufgabe 2

*'partial match queries'*



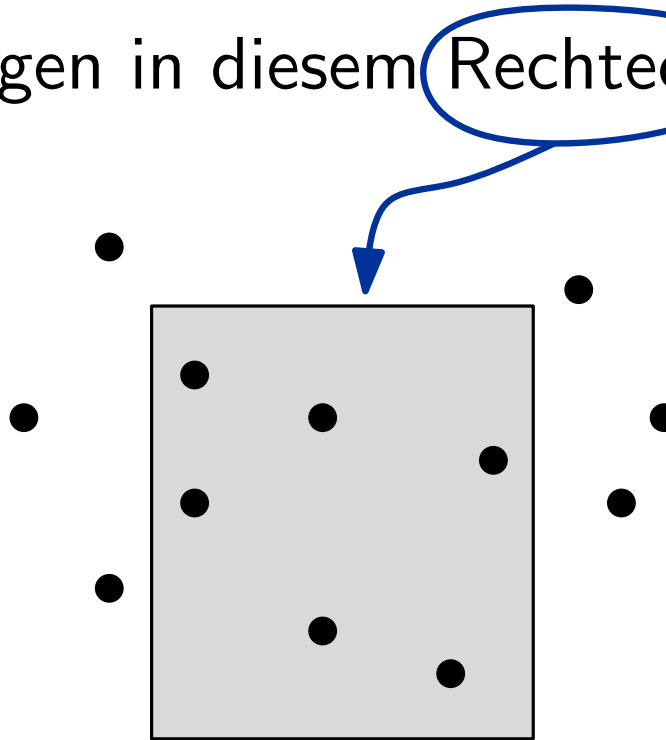
Gebe mir alle Punkte mit  $y = 7$ .

---

c) Gesucht: Datenstruktur, die das Problem in  $\mathcal{O}(\log n + k)$  Zeit und  $\mathcal{O}(n)$  Speicher löst.

# Aufgabe 3

Wie viele Punkte liegen in diesem Rechteck?

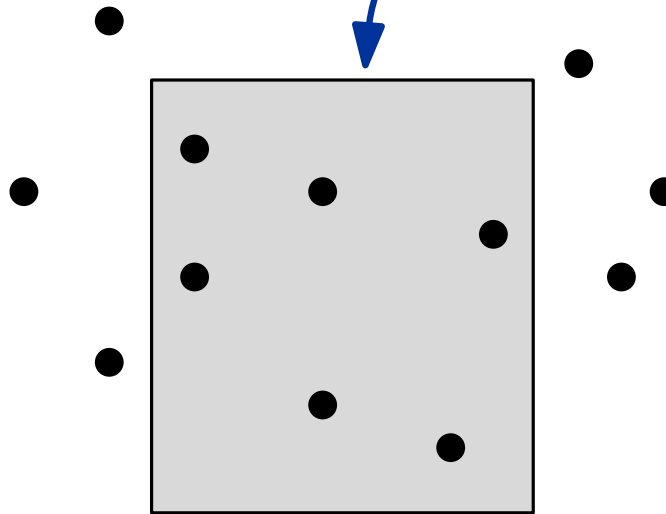


*range counting query*

**Anforderung:** Additive Konstante  $\mathcal{O}(k)$  in Laufzeit vermeiden

# Aufgabe 3

Wie viele Punkte liegen in diesem Rechteck?



*range counting query*

**Anforderung:** Additive Konstante  $\mathcal{O}(k)$  in Laufzeit vermeiden

a) Adaptiere 1-dim Range-Tree um range counting queries in  $\mathcal{O}(\log n)$  machbar ist

# Aufgabe 3

## 1dRangeQuery( $T, x, x'$ )

$v_{\text{split}} \leftarrow \text{FindSplitNode}(T, x, x')$   
**if**  $v_{\text{split}}$  ist Blatt **then** prüfe  $v_{\text{split}}$   
**else**

$v \leftarrow \text{lc}(v_{\text{split}})$

**while**  $v$  kein Blatt **do**

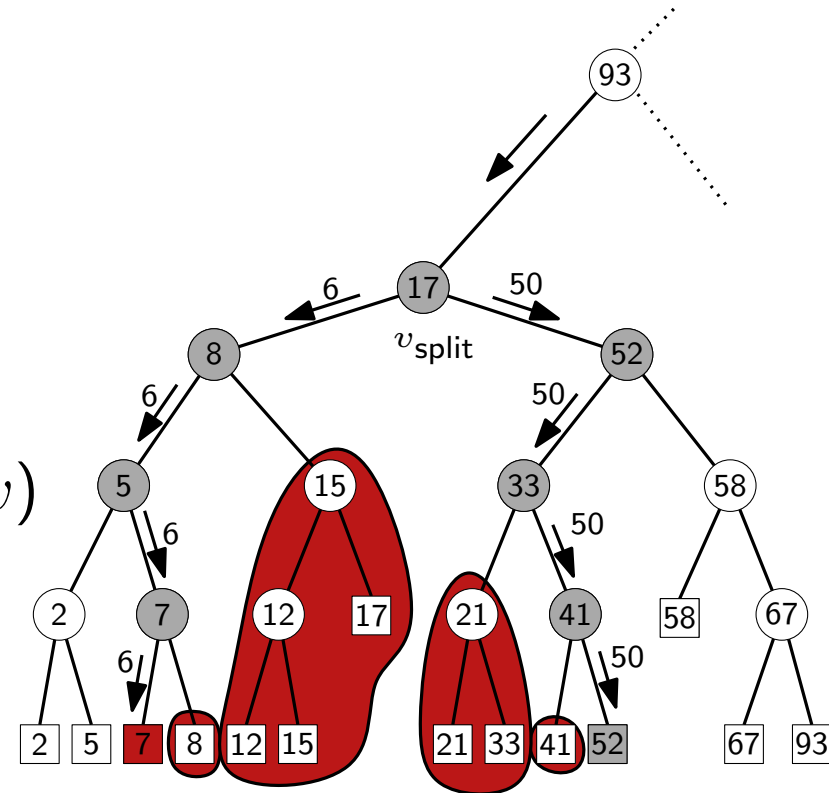
**if**  $x \leq x_v$  **then**

        | ReportSubtree( $\text{rc}(v)$ );  $v \leftarrow \text{lc}(v)$

**else**  $v \leftarrow \text{rc}(v)$

    prüfe  $v$

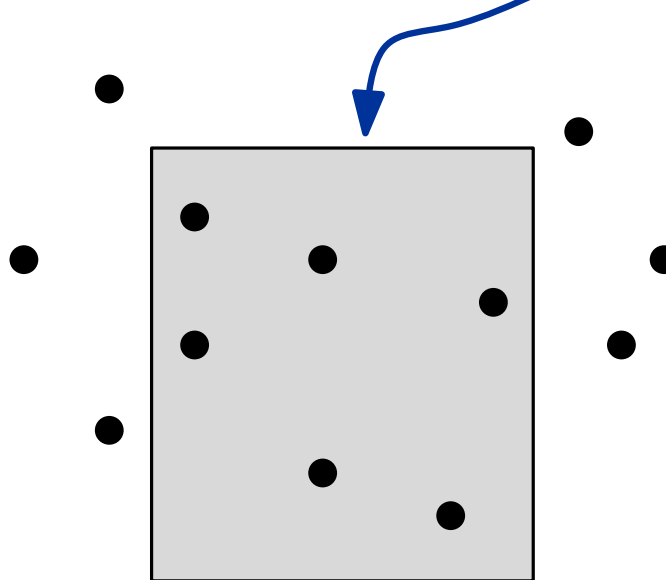
    // analog für  $x'$  und  $\text{rc}(v_{\text{split}})$



a) Adaptiere 1-dim Range-Tree um range counting queries in  $\mathcal{O}(\log n)$  machbar ist

# Aufgabe 3

Wie viele Punkte liegen in diesem Rechteck?

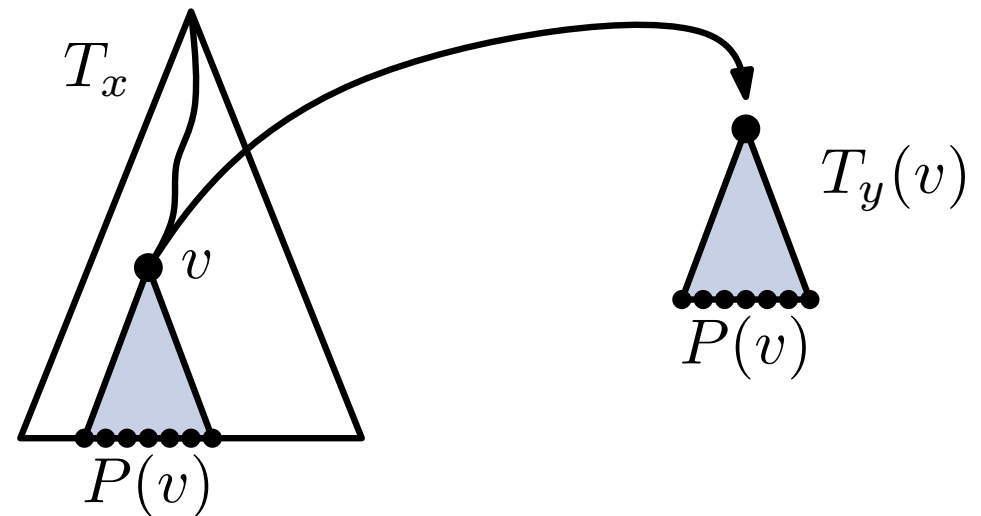


*range counting query*

**Anforderung:** Additive Konstante  $\mathcal{O}(k)$  in Laufzeit vermeiden

b) Benutze Lösung aus a) um das  $d$ -dimensionale Problem zu lösen

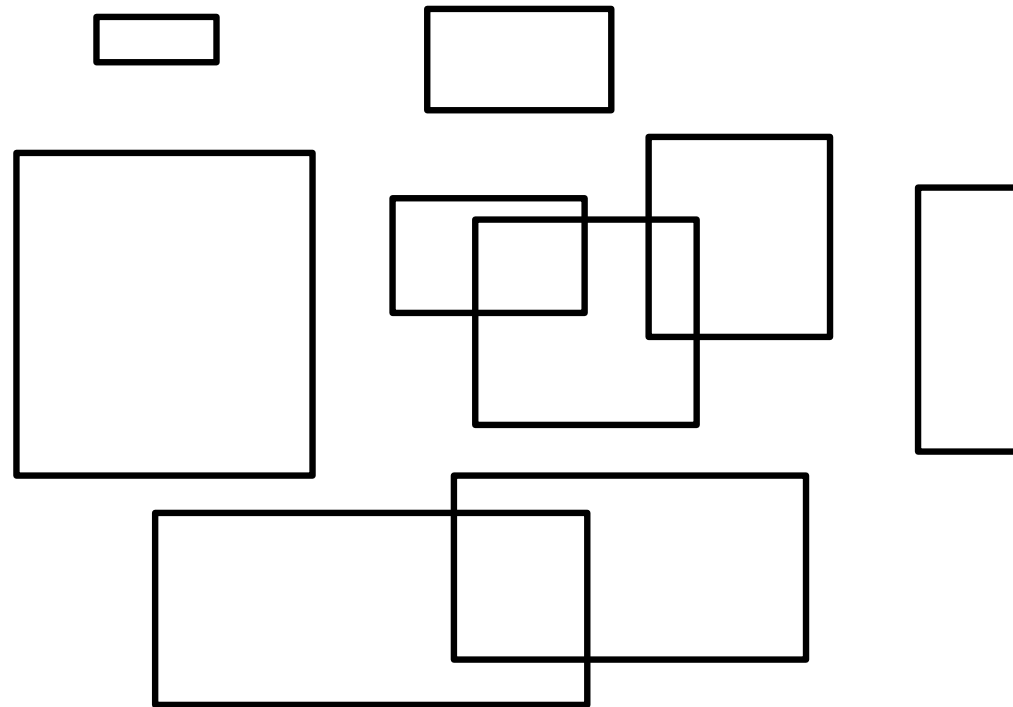
## 2dim Range-Trees:



**Anforderung:** Additive Konstante  $\mathcal{O}(k)$  in Laufzeit vermeiden

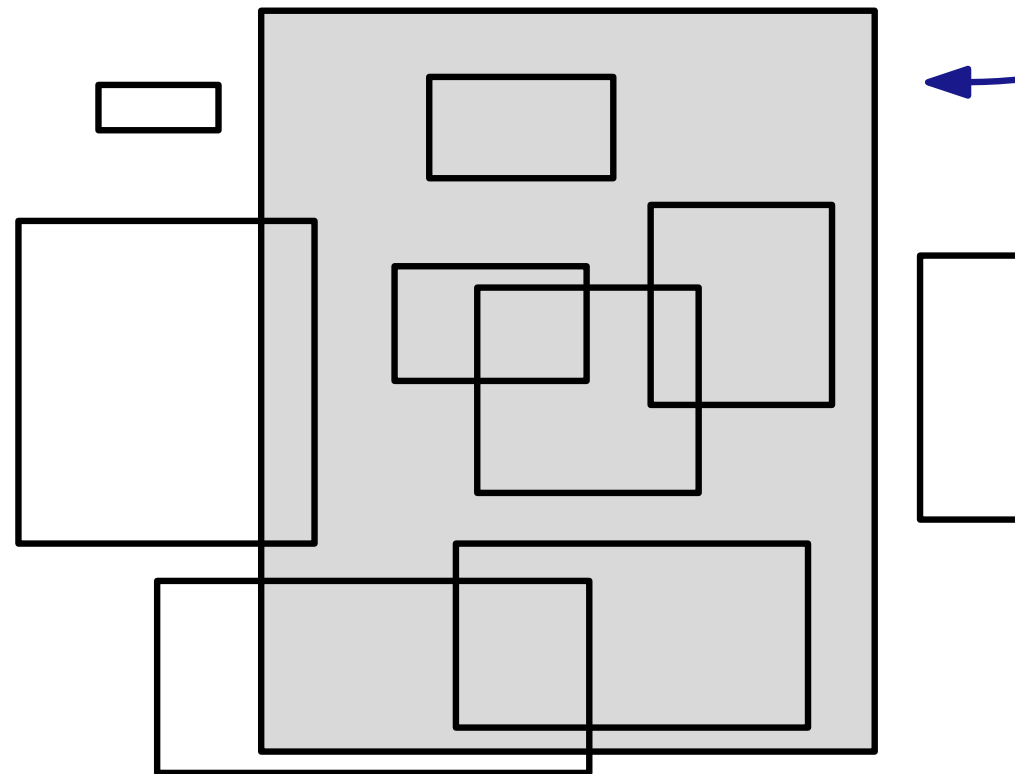
b) Benutze Lösung aus a) um das  $d$ -dimensionale Problem zu lösen

# Aufgabe 4



# Aufgabe 4

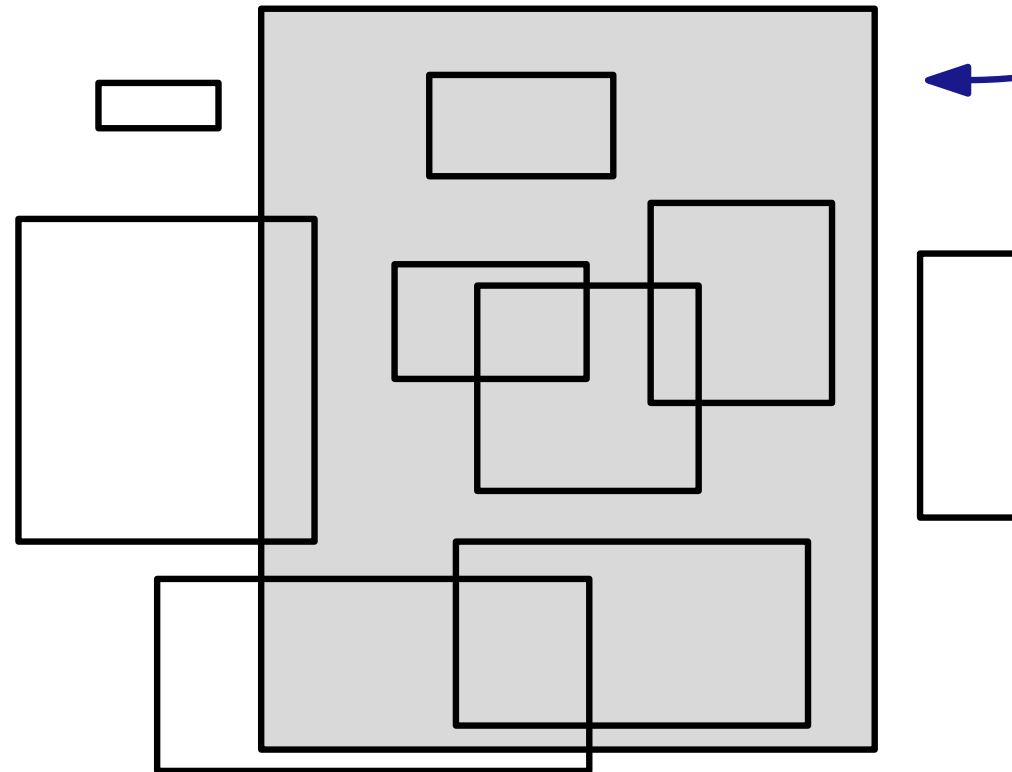
Welche Rechtecke liegen vollständig in diesem Rechteck?





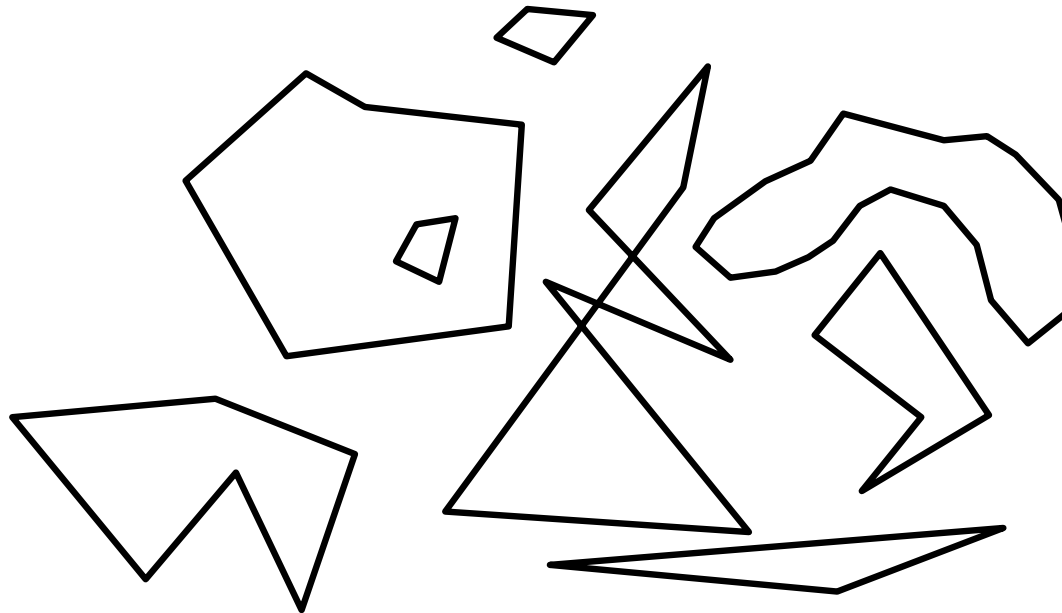
# Aufgabe 4

Welche Rechtecke liegen vollständig in diesem Rechteck?



Datenstruktur mit  $\mathcal{O}(n \log^3 n)$  Speicher und  $\mathcal{O}(\log^4 n + k)$  Anfragezeit.

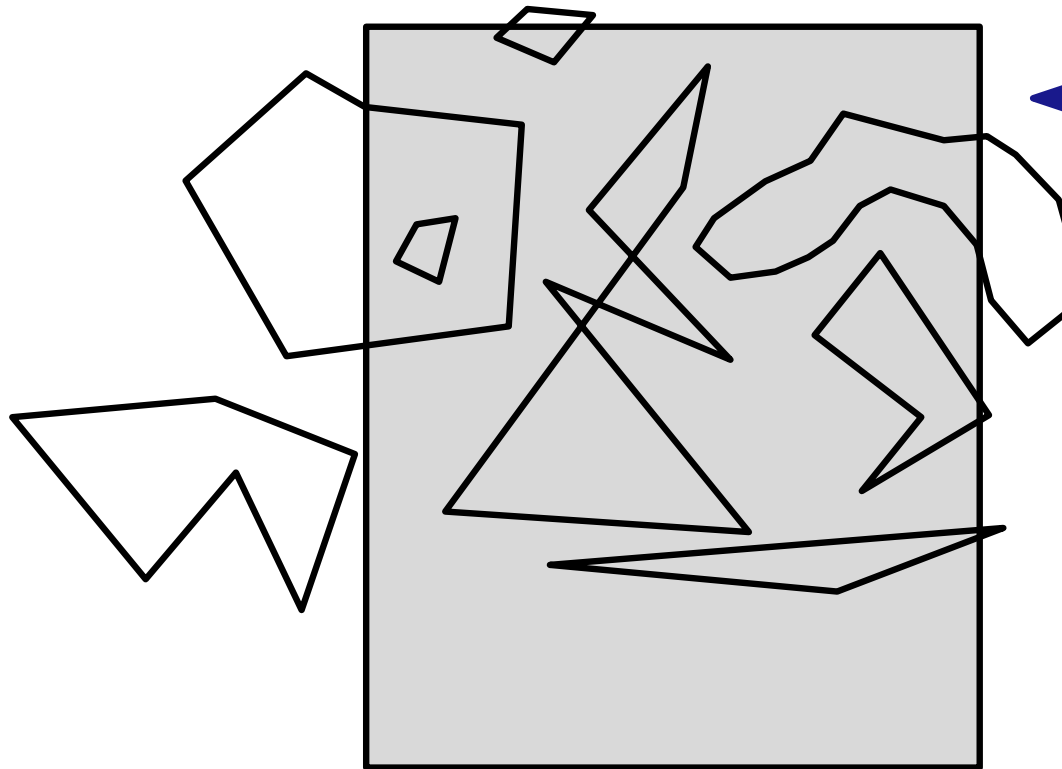
# Aufgabe 4



Datenstruktur mit  $\mathcal{O}(n \log^3 n)$  Speicher und  $\mathcal{O}(\log^4 n + k)$  Anfragezeit.

# Aufgabe 4

Welche Polygone liegen vollständig in diesem Rechteck?



Datenstruktur mit  $\mathcal{O}(n \log^3 n)$  Speicher und  $\mathcal{O}(\log^4 n + k)$  Anfragezeit.

Übungsblatt 4

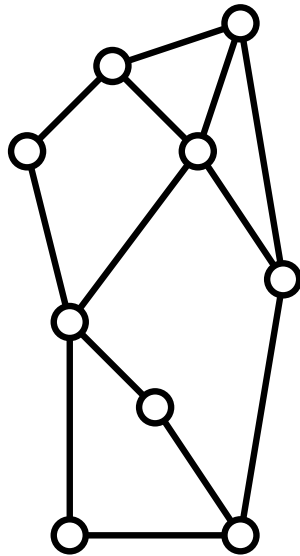
Nachtrag zu Übungsblatt 3

Übungsblatt 5

Werbung

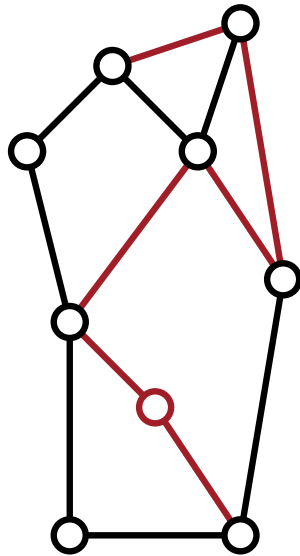
# Visualisierung dynamischer Netzwerke

dynamisches Netzwerk



# Visualisierung dynamischer Netzwerke

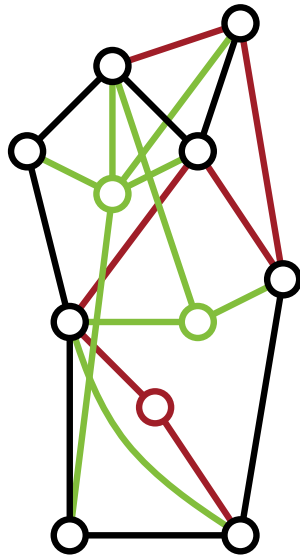
dynamisches Netzwerk



- alte Teile entfallen

# Visualisierung dynamischer Netzwerke

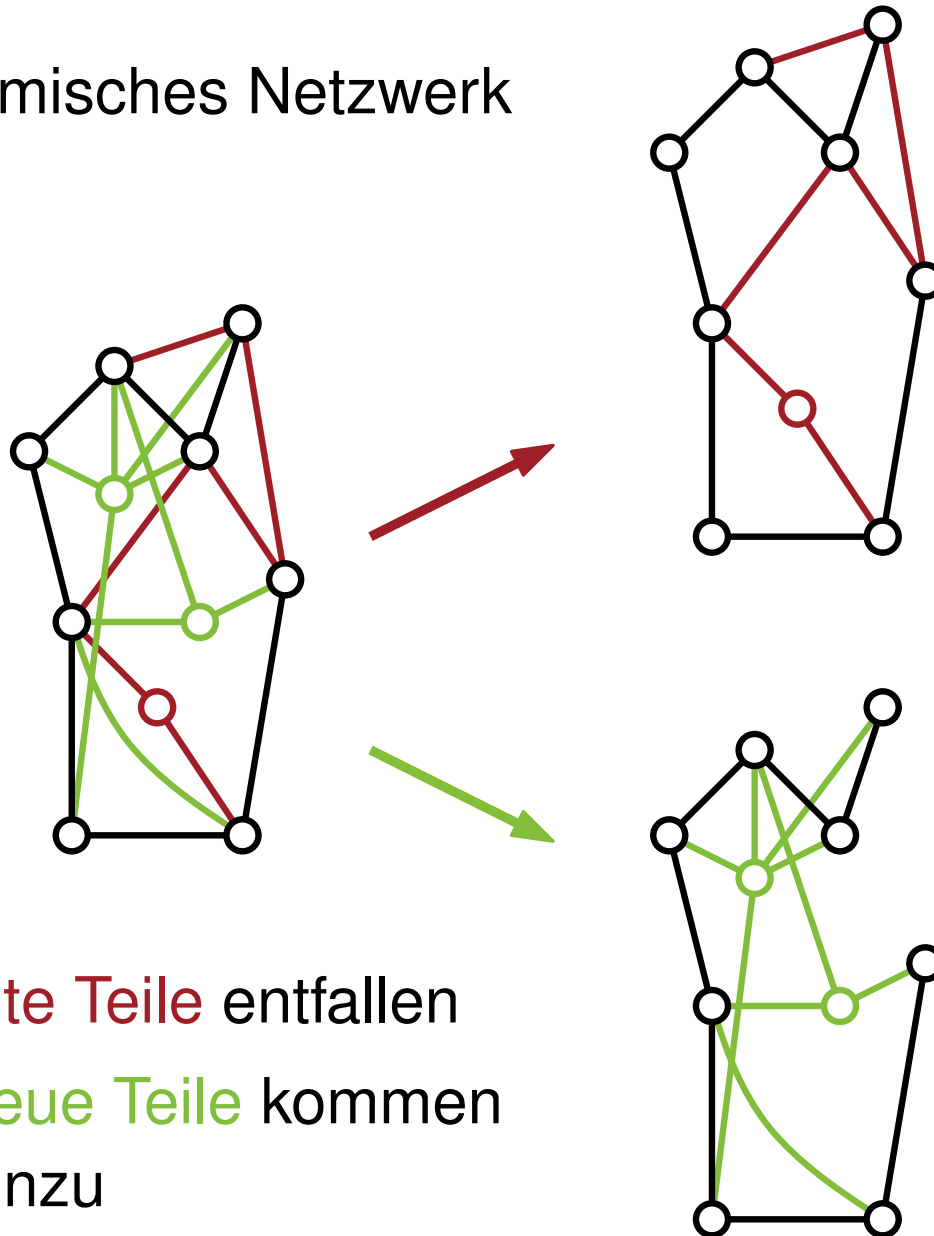
## dynamisches Netzwerk



- alte Teile entfallen
- neue Teile kommen hinzu

# Visualisierung dynamischer Netzwerke

dynamisches Netzwerk

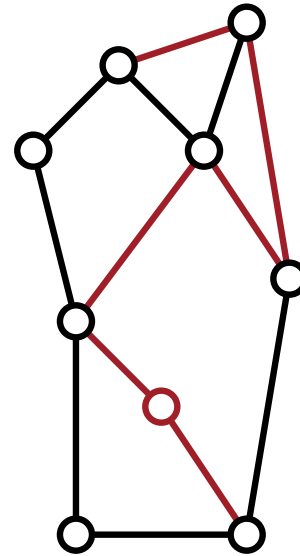
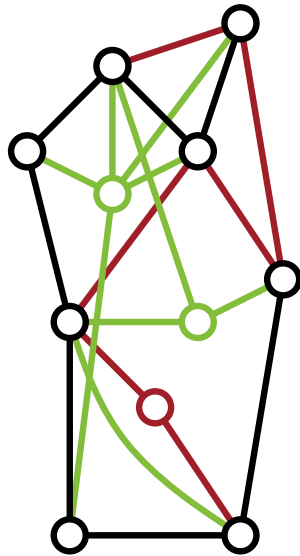


- alte Teile entfallen
- neue Teile kommen hinzu

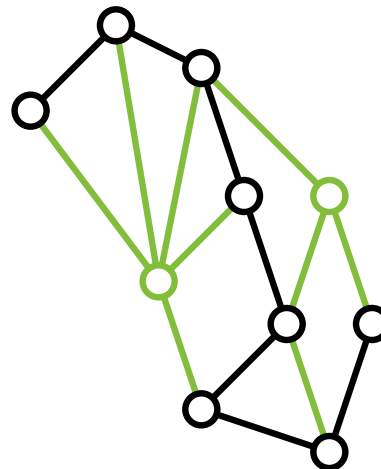


# Visualisierung dynamischer Netzwerke

dynamisches Netzwerk



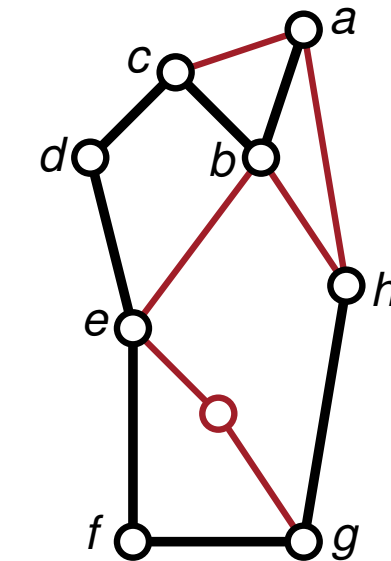
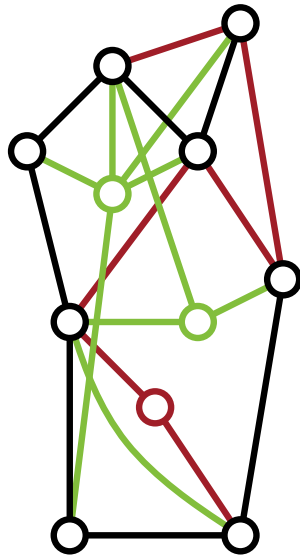
individuelle  
Zeichnungen



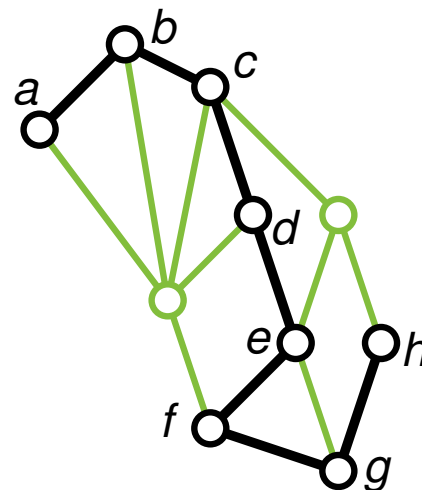
- alte Teile entfallen
- neue Teile kommen hinzu

# Visualisierung dynamischer Netzwerke

dynamisches Netzwerk



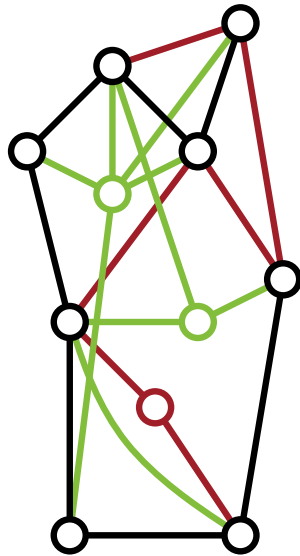
individuelle  
Zeichnungen



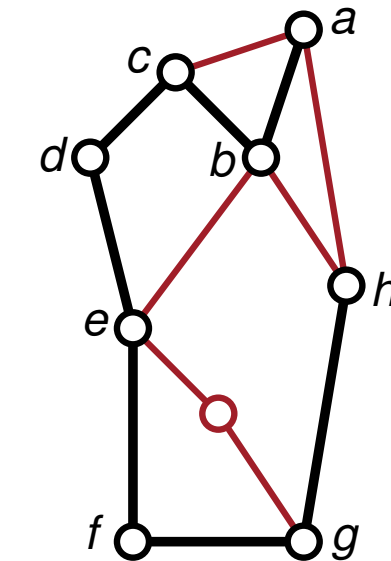
- alte Teile entfallen
- neue Teile kommen hinzu

# Visualisierung dynamischer Netzwerke

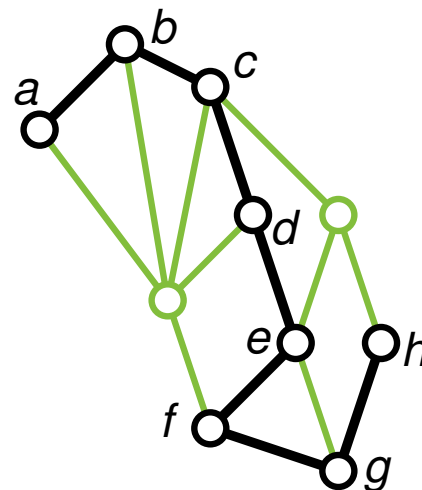
dynamisches Netzwerk



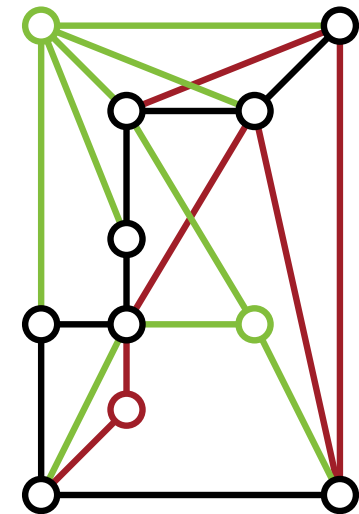
- alte Teile entfallen
- neue Teile kommen hinzu



individuelle  
Zeichnungen



simultane  
Zeichnung



Kompromiss zwischen

- guter Lesbarkeit und
- Ähnlichkeit der Zeichnungen

# Hiwi gesucht

## Aufgabe

Algorithmen zur Visualisierung dynamischer Graphen sollen

- entwickelt,
- implementiert,
- und evaluiert werden.

## Aufgabe

Algorithmen zur Visualisierung dynamischer Graphen sollen

- entwickelt,
- implementiert,
- und evaluiert werden.

## Voraussetzungen

- Kenntnisse in C++ oder Java
- Interesse an algorithmischen Fragestellungen

# Hiwi gesucht

## Aufgabe

Algorithmen zur Visualisierung dynamischer Graphen sollen

- entwickelt,
- implementiert,
- und evaluiert werden.

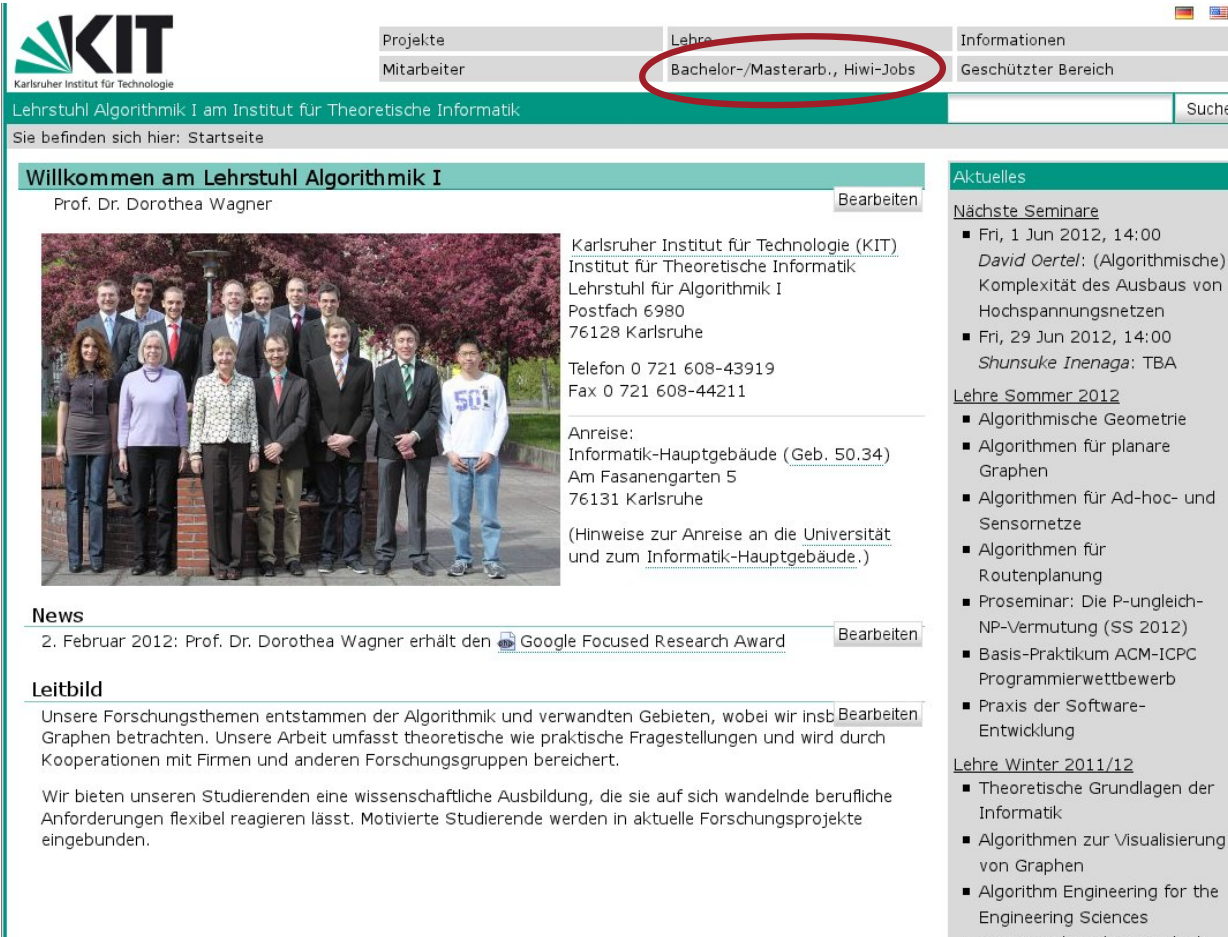
## Voraussetzungen

- Kenntnisse in C++ oder Java
- Interesse an algorithmischen Fragestellungen

## Wie? Wo? Was?

- Thomas Bläsius · Raum 316 · [thomas.blaesius@kit.edu](mailto:thomas.blaesius@kit.edu)

# Hiwi gesucht



The screenshot shows the website of the Chair of Algorithmics I at the Institute of Theoretical Informatics, KIT. The navigation menu includes 'Projekte', 'Mitarbeiter', 'Lehre', and 'Informationen'. The 'Lehre' menu item is circled in red, and its sub-item 'Bachelor-/Masterarb., Hiwi-Jobs' is also circled. Below the navigation, there is a search bar and a 'Sie befinden sich hier: Startseite' breadcrumb. The main content area is divided into several sections: 'Willkommen am Lehrstuhl Algorithmik I' with a photo of the staff and contact information; 'Aktuelles' with a list of seminars and courses; 'News' with a recent announcement; and 'Leitbild' with a description of the research and teaching goals.

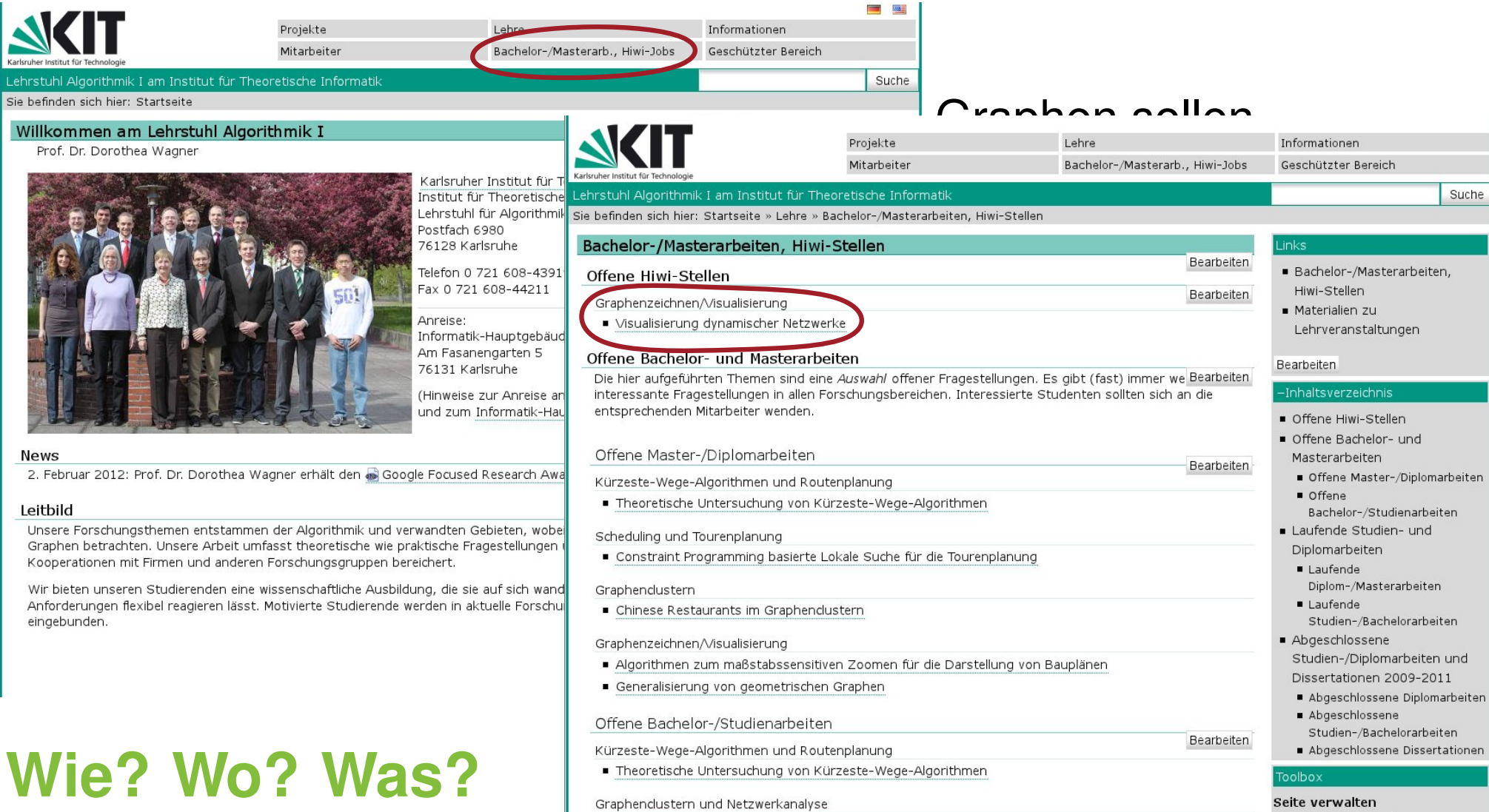
Graphen sollen

ngen

## Wie? Wo? Was?

- Thomas Bläsius · Raum 316 · [thomas.blaesius@kit.edu](mailto:thomas.blaesius@kit.edu)

# Hiwi gesucht



**KIT**  
Karlsruher Institut für Technologie


Projekte | **Bachelor-/Masterarb., Hiwi-Jobs** | Informationen  
Mitarbeiter | Geschützter Bereich

Lehrstuhl Algorithmik I am Institut für Theoretische Informatik

Sie befinden sich hier: Startseite

### Willkommen am Lehrstuhl Algorithmik I

Prof. Dr. Dorothea Wagner



Karlsruher Institut für T  
Institut für Theoretische  
Lehrstuhl für Algorithmik  
Postfach 6980  
76128 Karlsruhe  
Telefon 0 721 608-4391  
Fax 0 721 608-44211  
Anreise:  
Informatik-Hauptgebäude  
Am Fasanengarten 5  
76131 Karlsruhe  
(Hinweise zur Anreise an  
und zum Informatik-Hau

**Bachelor-/Masterarbeiten, Hiwi-Stellen**

**Offene Hiwi-Stellen**

- Graphenzeichnen/Visualisierung
  - Visualisierung dynamischer Netzwerke

**Offene Bachelor- und Masterarbeiten**

Die hier aufgeführten Themen sind eine *Auswahl* offener Fragestellungen. Es gibt (fast) immer we Bearbeiten interessante Fragestellungen in allen Forschungsbereichen. Interessierte Studenten sollten sich an die entsprechenden Mitarbeiter wenden.

**Offene Master-/Diplomarbeiten**

Kürzeste-Wege-Algorithmen und Routenplanung

- Theoretische Untersuchung von Kürzeste-Wege-Algorithmen

Scheduling und Tourenplanung

- Constraint Programming basierte Lokale Suche für die Tourenplanung

Graphendustern

- Chinese Restaurants im Graphendustern

Graphenzeichnen/Visualisierung

- Algorithmen zum maßstabssensitiven Zoomen für die Darstellung von Bauplänen
- Generalisierung von geometrischen Graphen

**Offene Bachelor-/Studienarbeiten**

Kürzeste-Wege-Algorithmen und Routenplanung

- Theoretische Untersuchung von Kürzeste-Wege-Algorithmen

Graphendustern und Netzwerkanalyse

**Links**

- Bachelor-/Masterarbeiten, Hiwi-Stellen
- Materialien zu Lehrveranstaltungen

**-Inhaltsverzeichnis**

- Offene Hiwi-Stellen
- Offene Bachelor- und Masterarbeiten
  - Offene Master-/Diplomarbeiten
  - Offene Bachelor-/Studienarbeiten
- Laufende Studien- und Diplomarbeiten
  - Laufende Diplom-/Masterarbeiten
  - Laufende Studien-/Bachelorarbeiten
- Abgeschlossene Studien-/Diplomarbeiten und Dissertationen 2009-2011
  - Abgeschlossene Diplomarbeiten
  - Abgeschlossene Studien-/Bachelorarbeiten
  - Abgeschlossene Dissertationen

**Toolbox**

Seite verwalten

## Wie? Wo? Was?

■ Thomas Bläsius · Raum 316 · [thomas.blaesius@kit.edu](mailto:thomas.blaesius@kit.edu)



# Das war's!

Nächster Termin:  
**Donnertag, 31.05, 10:15 Uhr**  
Raum 131, Gebäude 50.34