

# Algorithmen II

## Übung am 17.12.2013

Algorithmische Geometrie

INSTITUT FÜR THEORETISCHE INFORMATIK · PROF. DR. DOROTHEA WAGNER



# Schnitte von Strecken

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

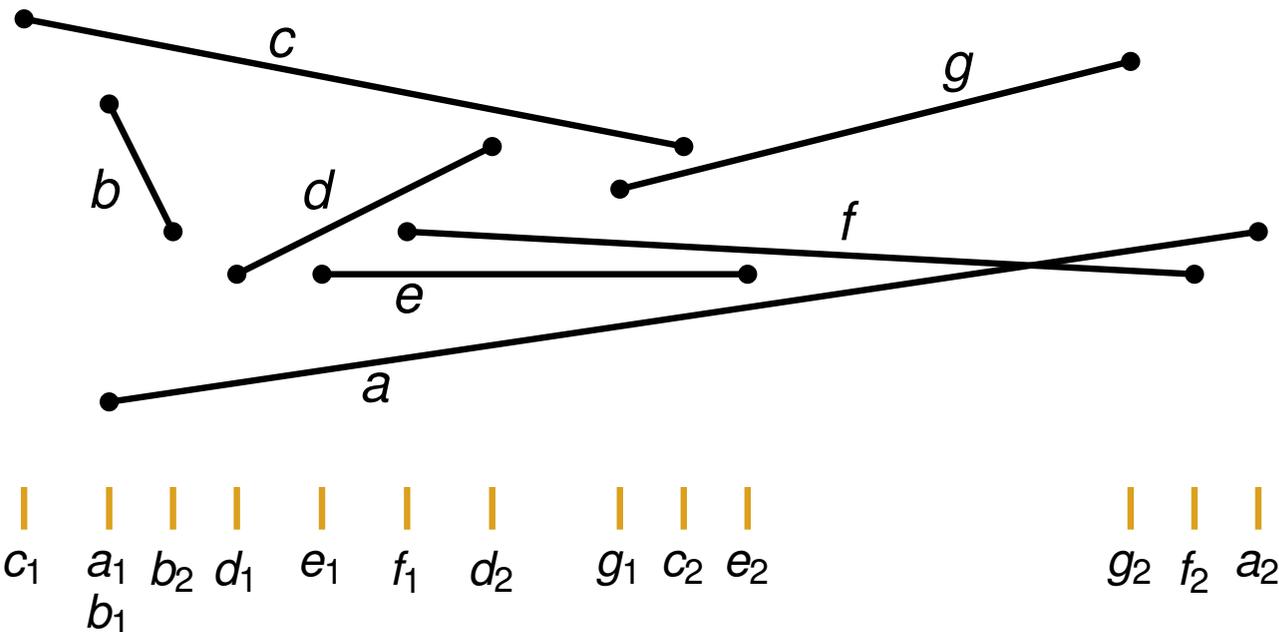
# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

**Beispiel:**

Sweep-Line Zustand



Event-Point Schedule →

# Problem 1(a)

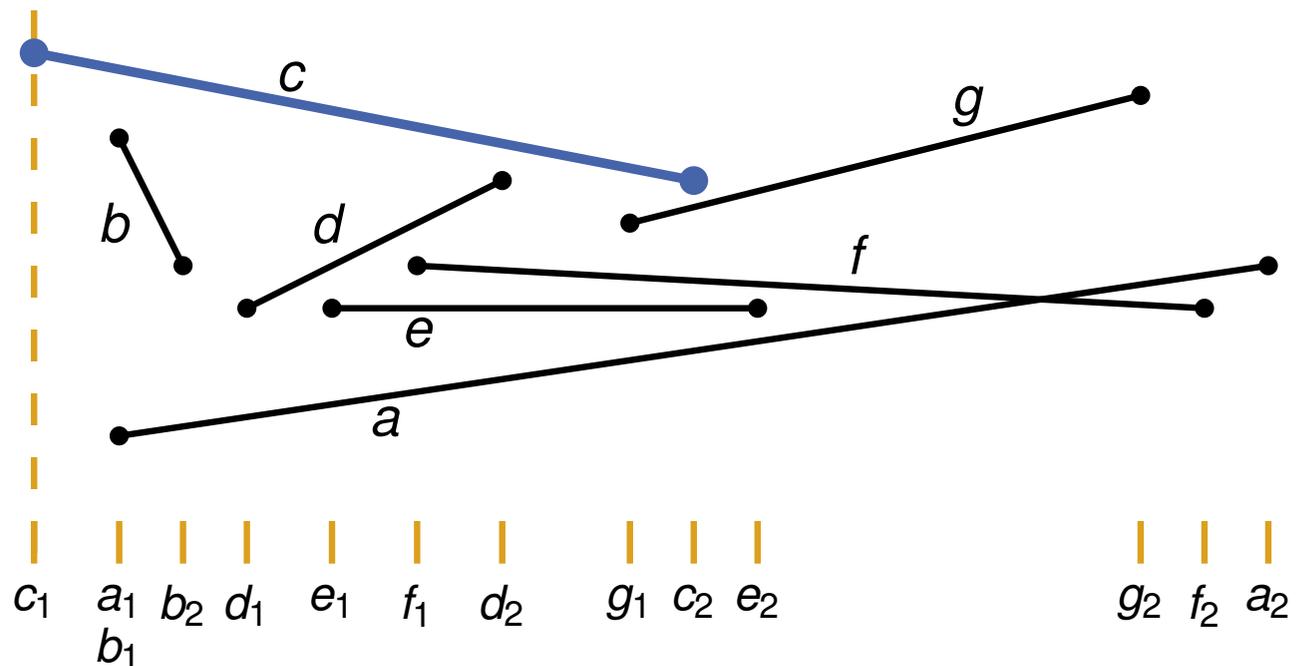
Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

**Beispiel:**

Sweep-Line Zustand

↓  
+  $c$



# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

**Beispiel:**

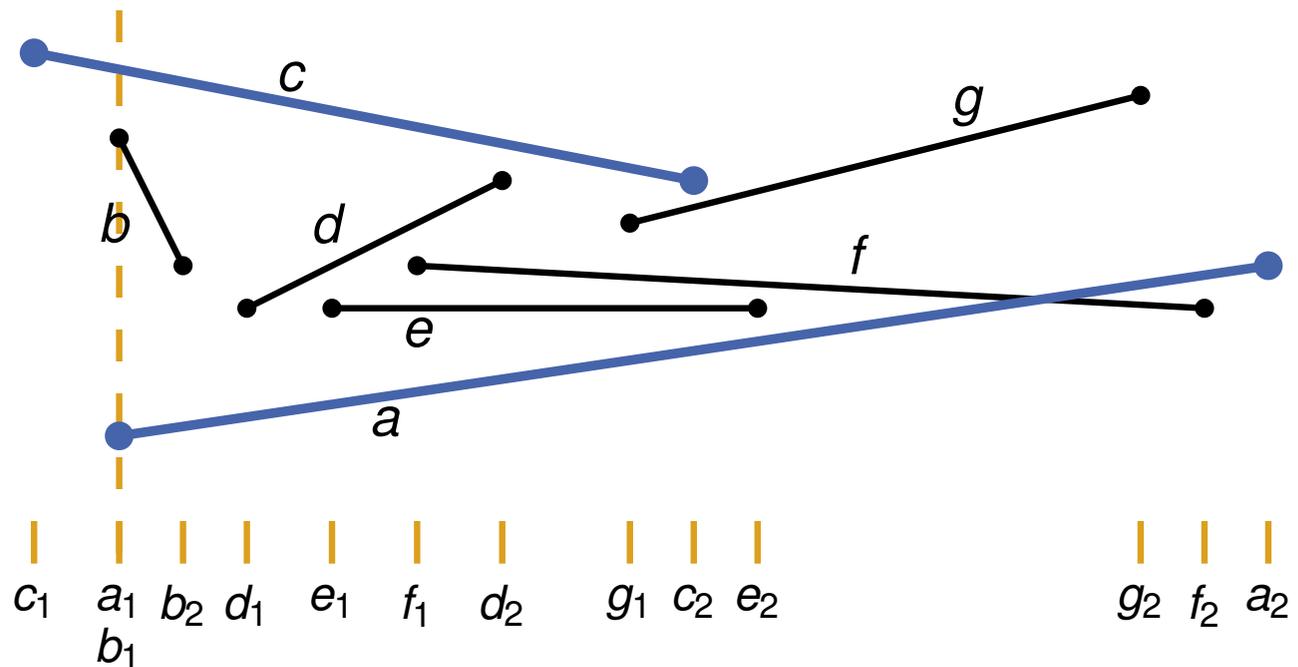
Sweep-Line Zustand



$c$

+  $a$

Event-Point Schedule →



$a$  schneidet  $c$  nicht

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

**Beispiel:**

Sweep-Line Zustand

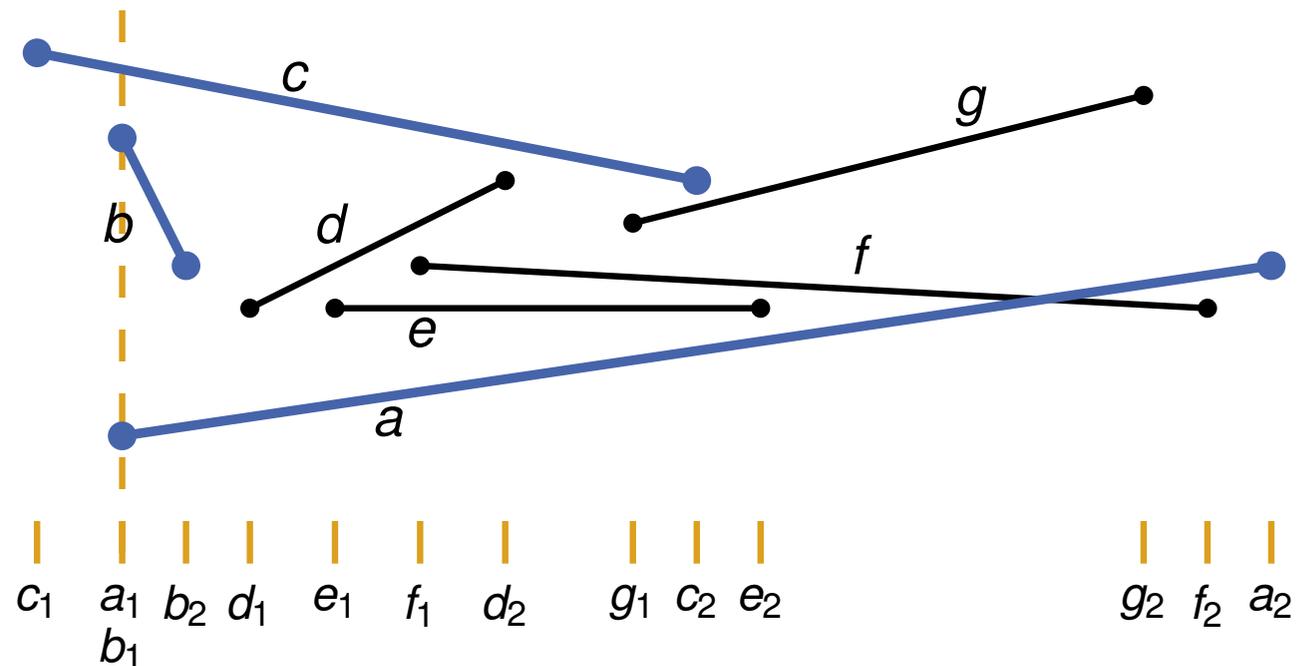


$c$

+  $b$

$a$

Event-Point Schedule →



$b$  schneidet weder  $a$  noch  $c$

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

## Beispiel:

Sweep-Line Zustand

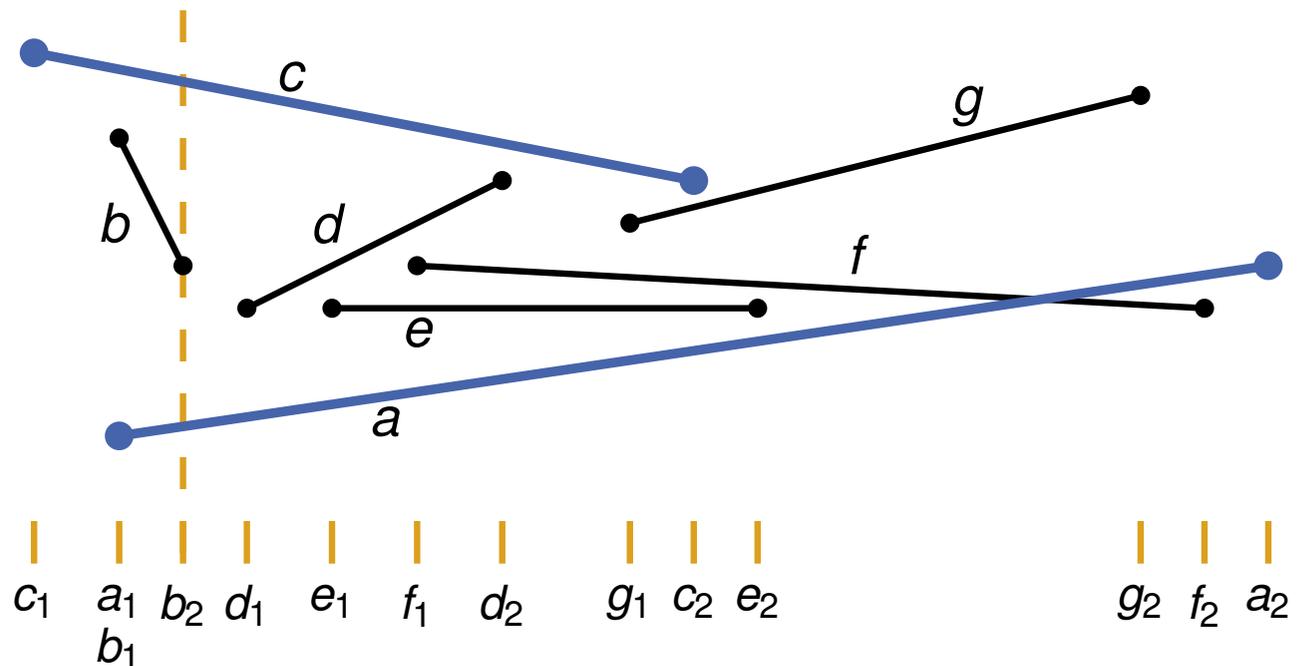


$c$

~~$b$~~

$a$

Event-Point Schedule →



$a$  schneidet  $c$  nicht

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

**Beispiel:**

Sweep-Line Zustand

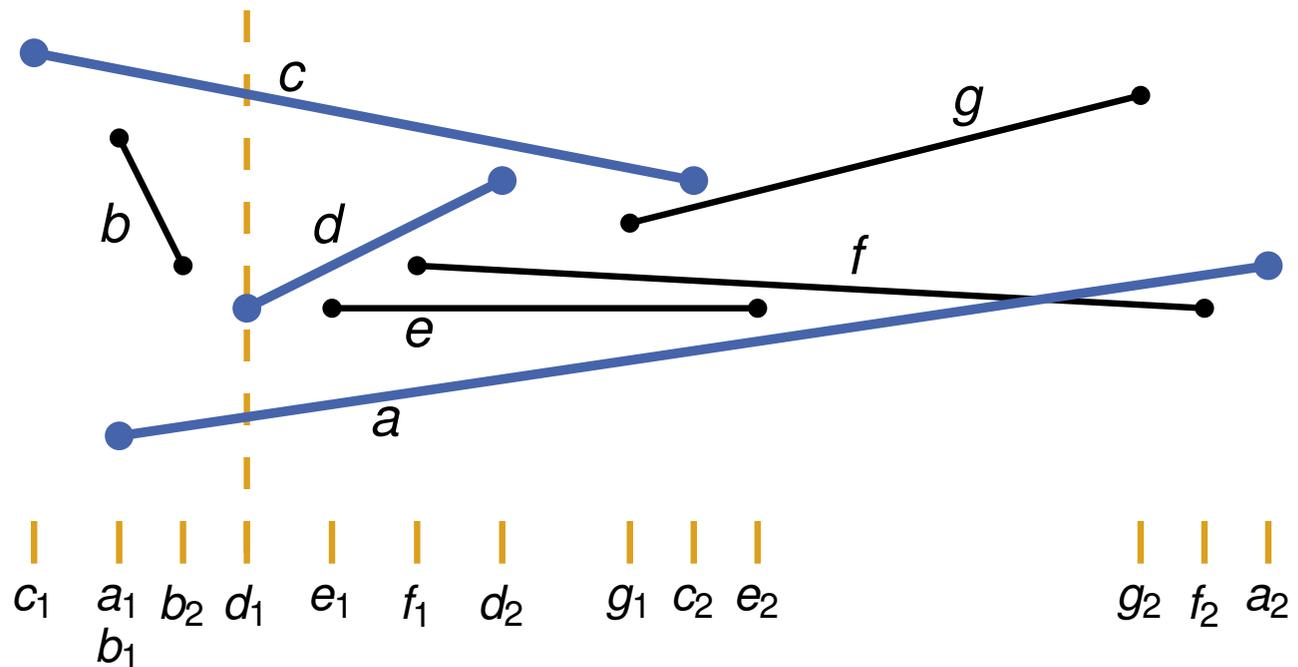


$c$

+  $d$

$a$

Event-Point Schedule →



$d$  schneidet weder  $a$  noch  $c$

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

**Beispiel:**

Sweep-Line Zustand



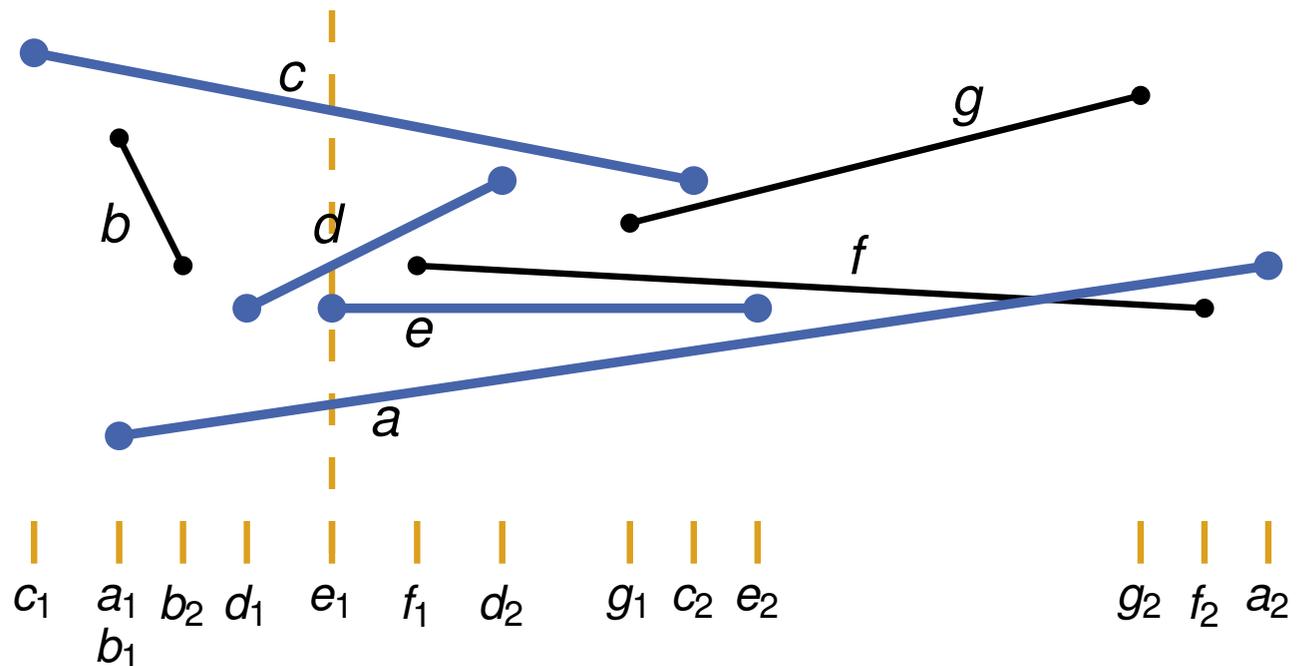
$c$

$d$

+  $e$

$a$

Event-Point Schedule →



$e$  schneidet weder  $d$  noch  $a$

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

## Beispiel:

Sweep-Line Zustand



$c$

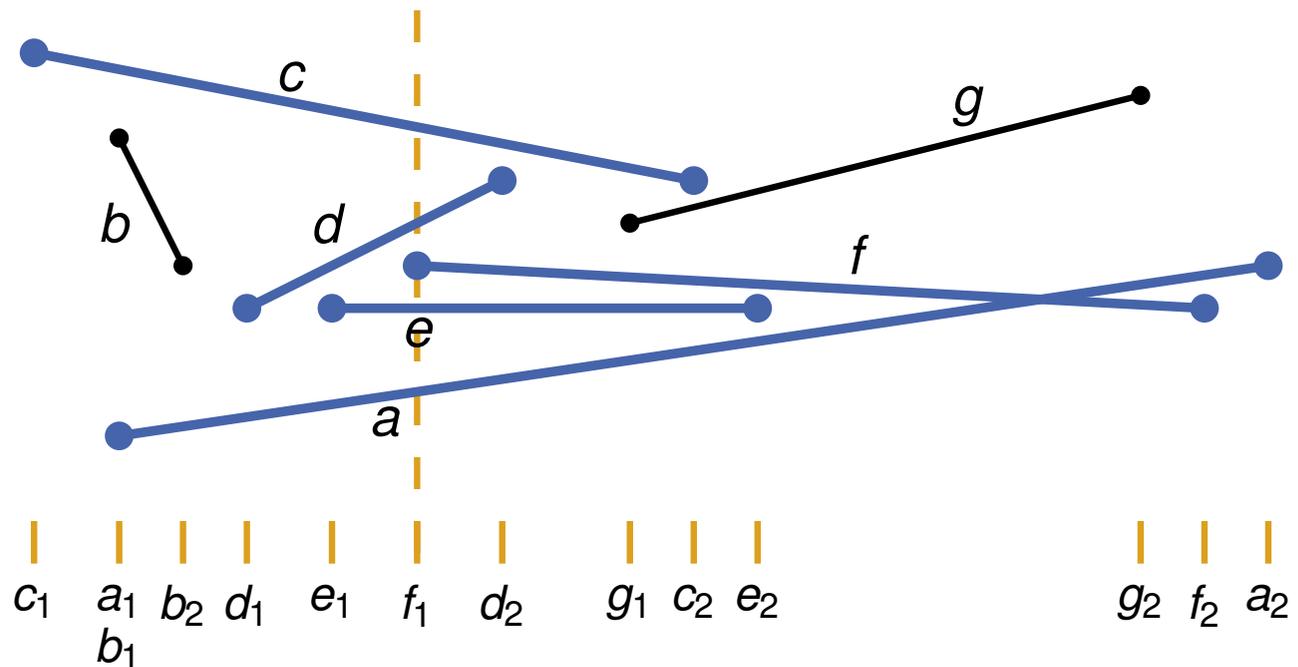
$d$

+  $f$

$e$

$a$

Event-Point Schedule →



$f$  schneidet weder  $d$  noch  $e$

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

## Beispiel:

Sweep-Line Zustand



$c$

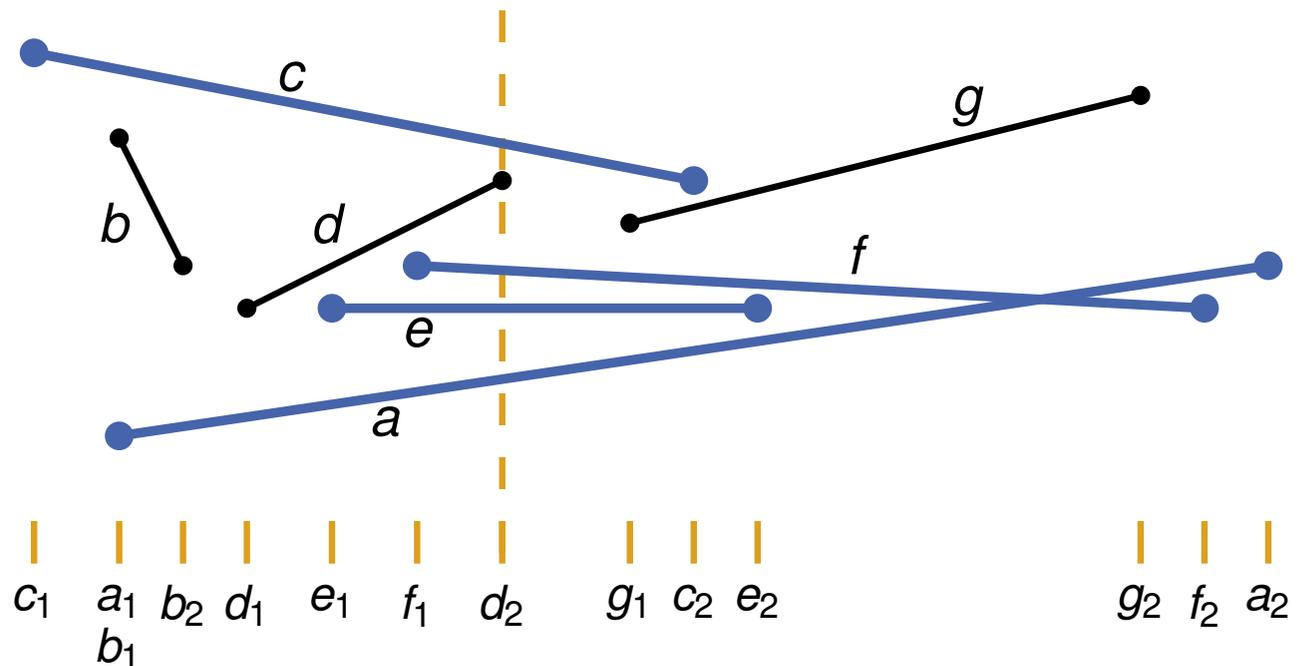
~~$d$~~

$f$

$e$

$a$

Event-Point Schedule →



$f$  schneidet  $c$  nicht

# Problem 1(a)

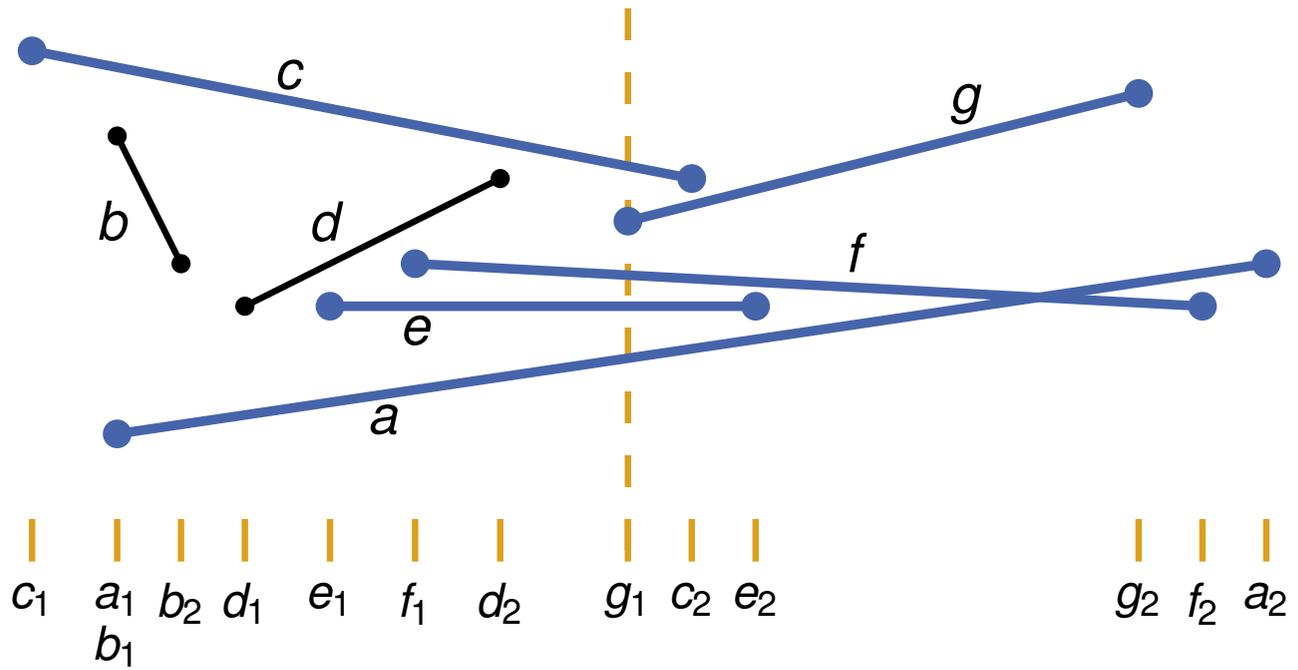
Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

## Beispiel:

Sweep-Line Zustand

↓  
 $c$   
 $+ g$   
 $f$   
 $e$   
 $a$



$g$  schneidet weder  $f$  noch  $c$

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

**Beispiel:**

Sweep-Line Zustand



~~c~~

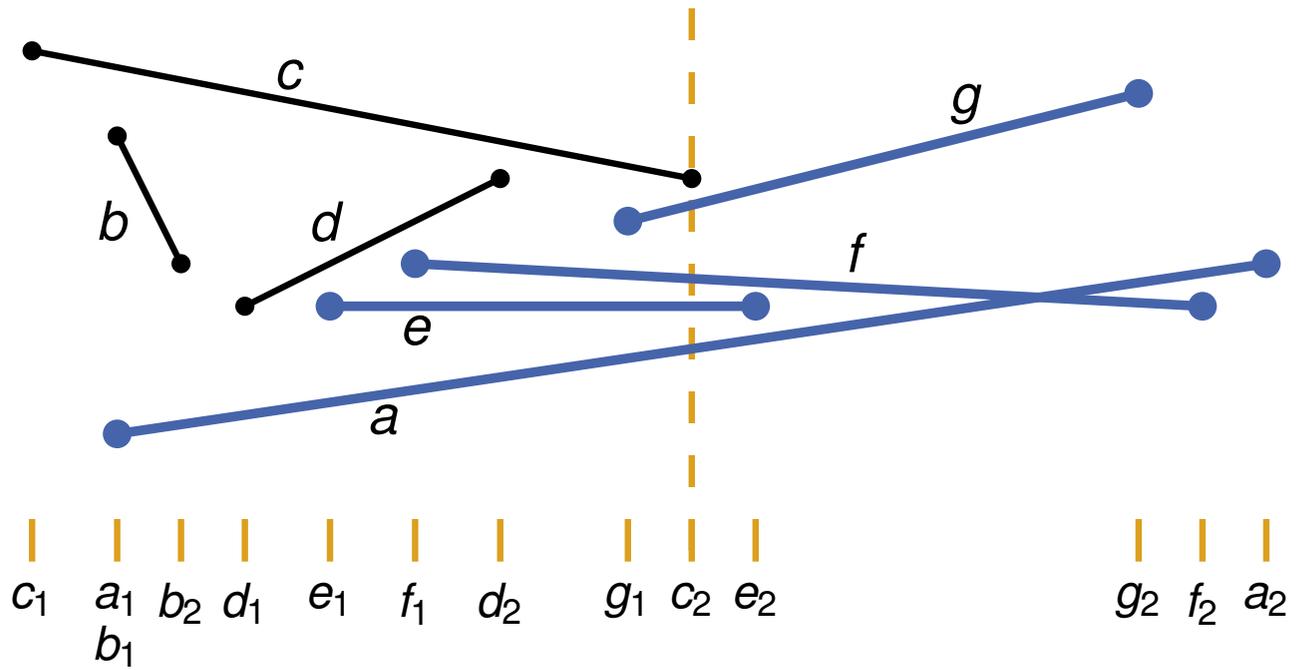
g

f

e

a

Event-Point Schedule →



# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

## Beispiel:

Sweep-Line Zustand

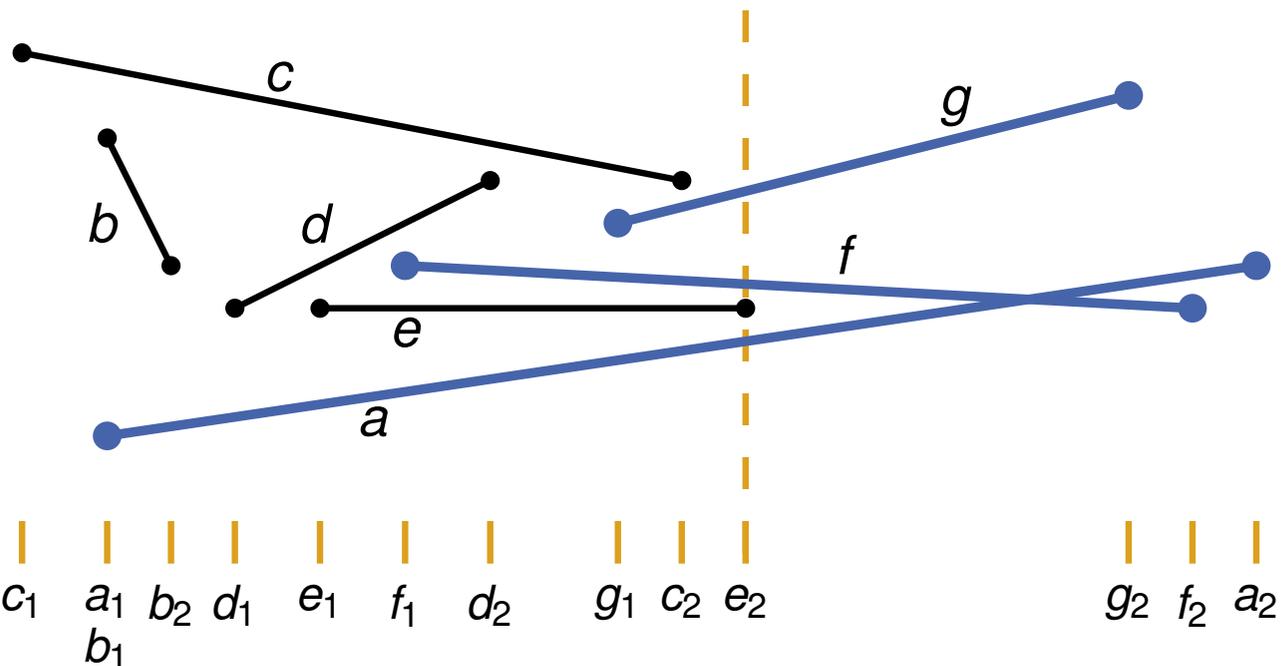


$g$

$f$

~~$e$~~

$a$



Event-Point Schedule →

$f$  schneidet  $a!$   $\Rightarrow$  gib TRUE zurück

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

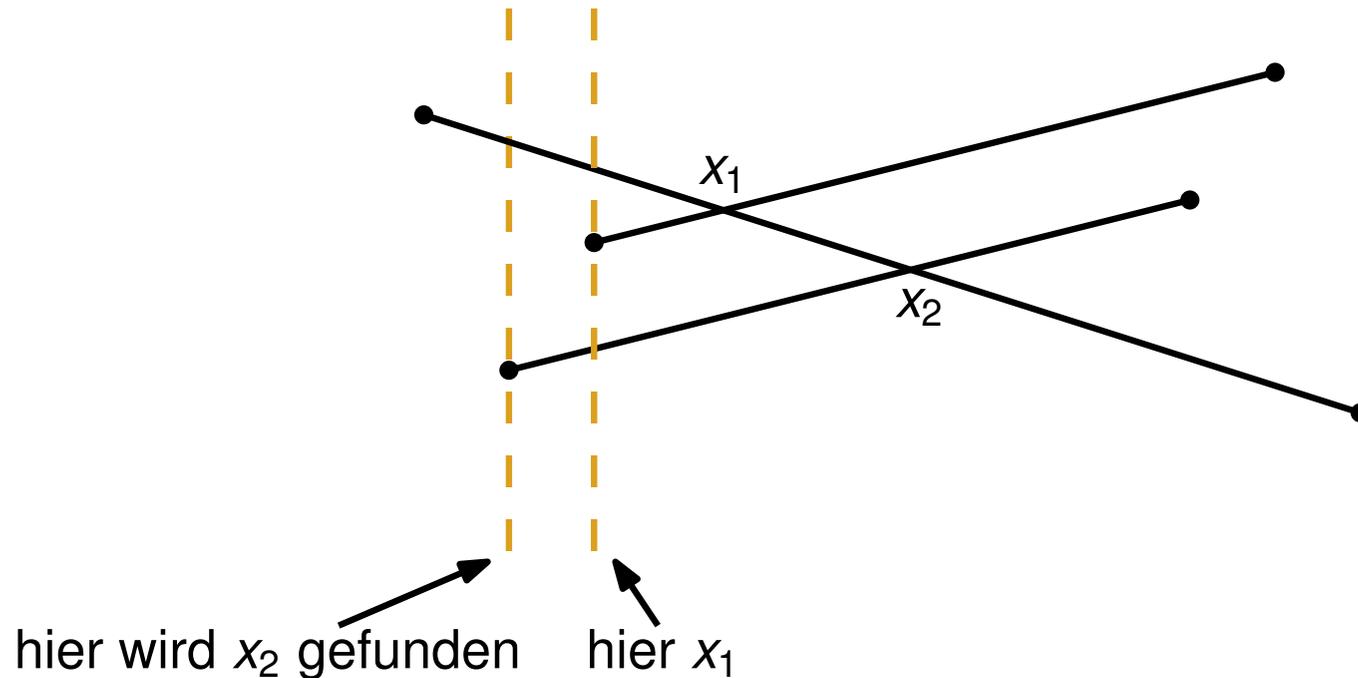
(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

# Problem 1(a)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(a) Zeigen oder widerlegen Sie: Es wird immer der linkeste Schnittpunkt gefunden.

**Nein, denn:**



# Problem 1(b)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.

# Problem 1 (b)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.

## Was ändert sich?

# Problem 1 (b)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.

## Was ändert sich?

- Man darf nicht abbrechen, sobald der erste Schnittpunkte gefunden wurde.

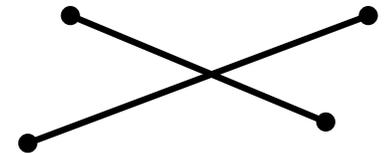
# Problem 1 (b)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.

## Was ändert sich?

- Man darf nicht abbrechen, sobald der erste Schnittpunkte gefunden wurde.
- An Schnittpunkten ändert sich die Reihenfolge der Kanten.



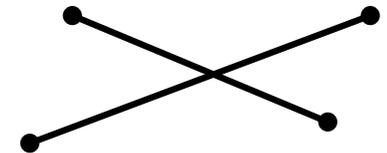
# Problem 1(b)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

(b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.

## Was ändert sich?

- Man darf nicht abbrechen, sobald der erste Schnittpunkte gefunden wurde.
- An Schnittpunkten ändert sich die Reihenfolge der Kanten.
- Gefundene Schnittpunkte müssen als Haltepunkte in den Event-Point Schedule eingefügt werden.



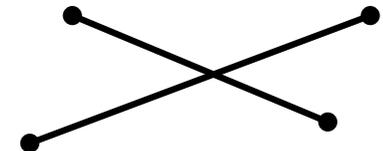
# Problem 1(b)

Sei  $S$  eine Menge von  $n$  Strecken. Der Algorithmus aus der Vorlesung kann in  $O(n \log n)$  Zeit testen, ob es in  $S$  ein sich schneidendes Streckenpaar gibt.

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.

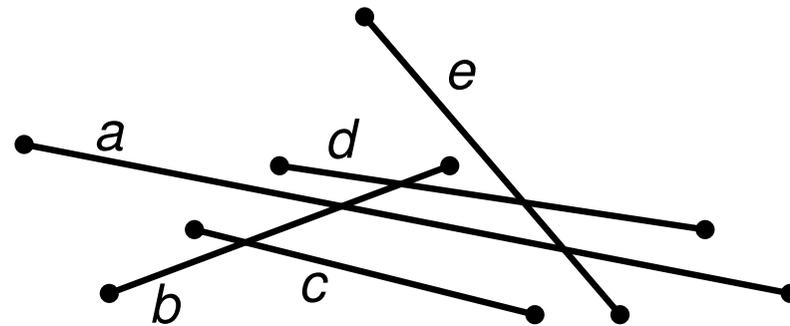
## Was ändert sich?

- Man darf nicht abbrechen, sobald der erste Schnittpunkte gefunden wurde.
  - An Schnittpunkten ändert sich die Reihenfolge der Kanten.
  - Gefundene Schnittpunkte müssen als Haltepunkte in den Event-Point Schedule eingefügt werden.
  - An solchen Haltepunkten muss der Sweep-Line Zustand geändert werden: die beiden für die Kreuzung verantwortlichen Strecken werden vertauscht.
- Achtung:** auch hierbei sind hinterher Streckenpaare benachbart, die vorher nicht benachbart waren.

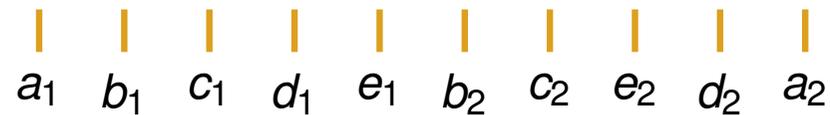


# Problem 1(b) – Beispiel

Sweep-Line Zustand



Event-Point Schedule →



Paare die aktuell auf Schnitt getestet werden müssen:

Schnitt gefunden zwischen:

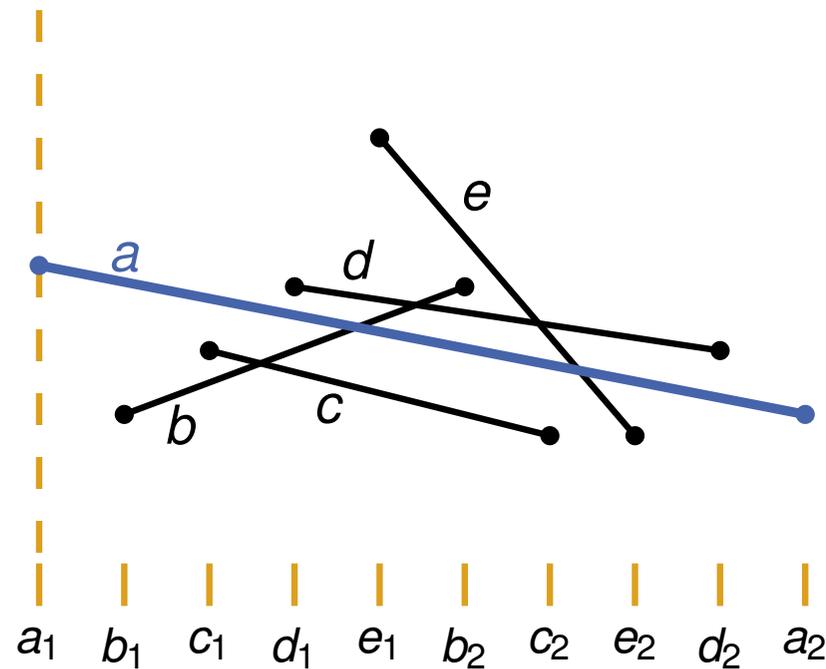
# Problem 1(b) – Beispiel

Sweep-Line Zustand



*a*

Event-Point Schedule →



Paare die aktuell auf Schnitt getestet werden müssen:

Schnitt gefunden zwischen:

# Problem 1(b) – Beispiel

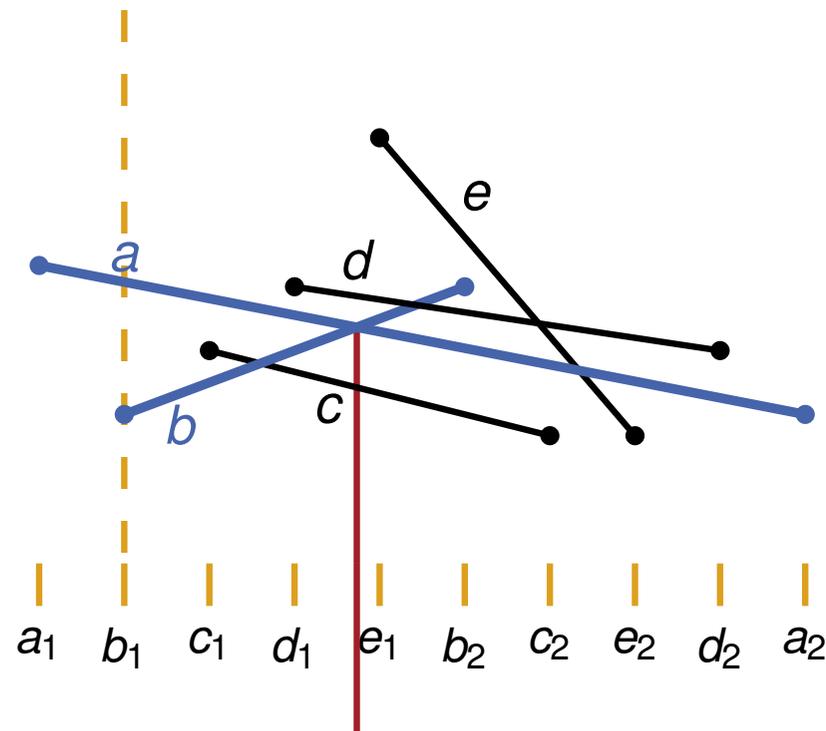
Sweep-Line Zustand



*a*

*b*

Event-Point Schedule →



Paare die aktuell auf Schnitt getestet werden müssen: *a* und *b*

Schnitt gefunden zwischen: *a* und *b*

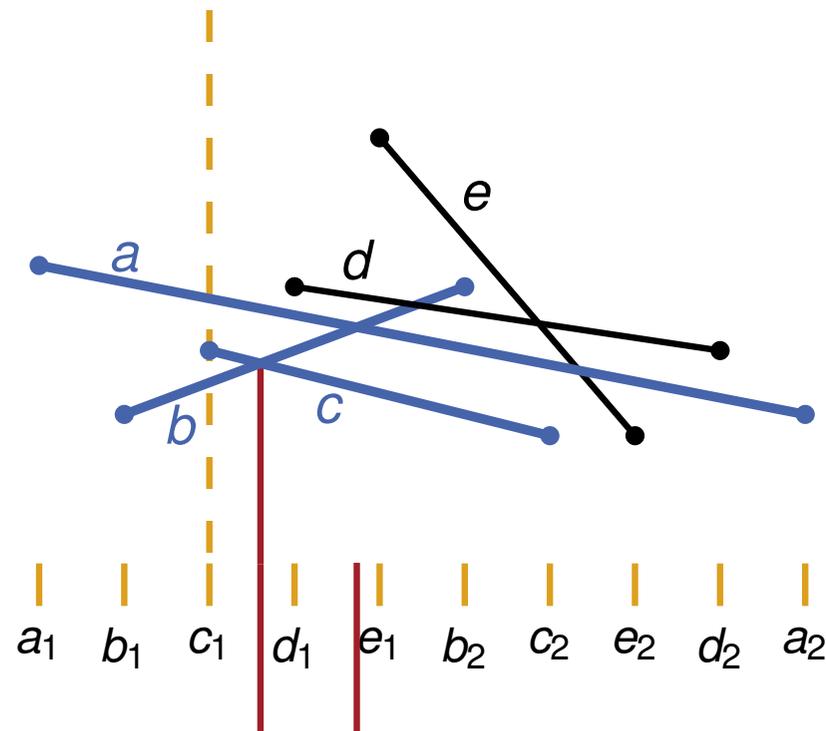
# Problem 1(b) – Beispiel

Sweep-Line Zustand



a  
c  
b

Event-Point Schedule →



Paare die aktuell auf Schnitt getestet werden müssen:  $b$  und  $c$ ,  $c$  und  $a$

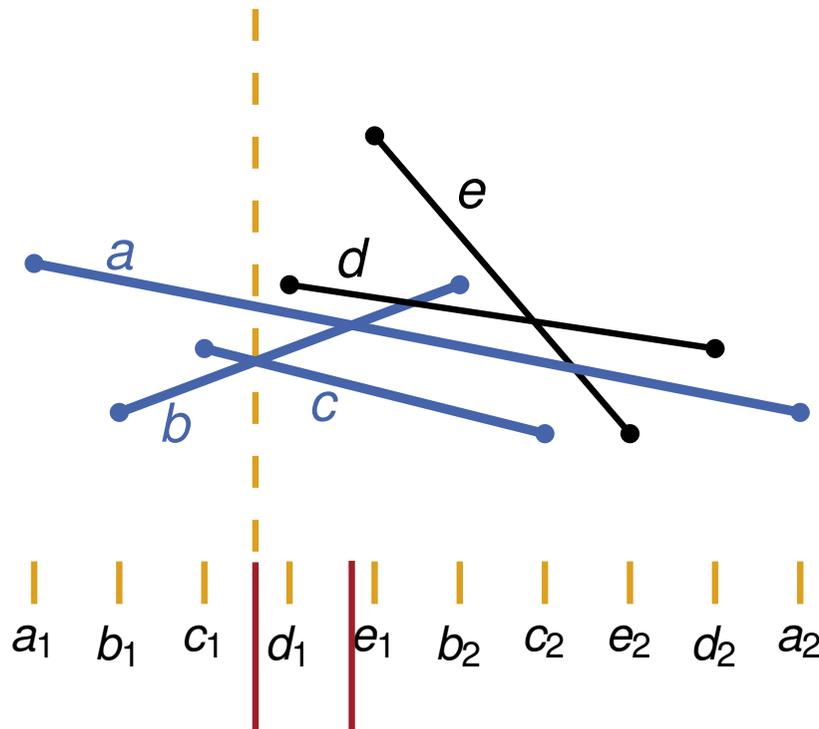
Schnitt gefunden zwischen:  $a$  und  $b$ ,  $b$  und  $c$

# Problem 1(b) – Beispiel

Sweep-Line Zustand



*a*  
*b*  
*c*



Event-Point Schedule →

Paare die aktuell auf Schnitt getestet werden müssen: *b* und *a*

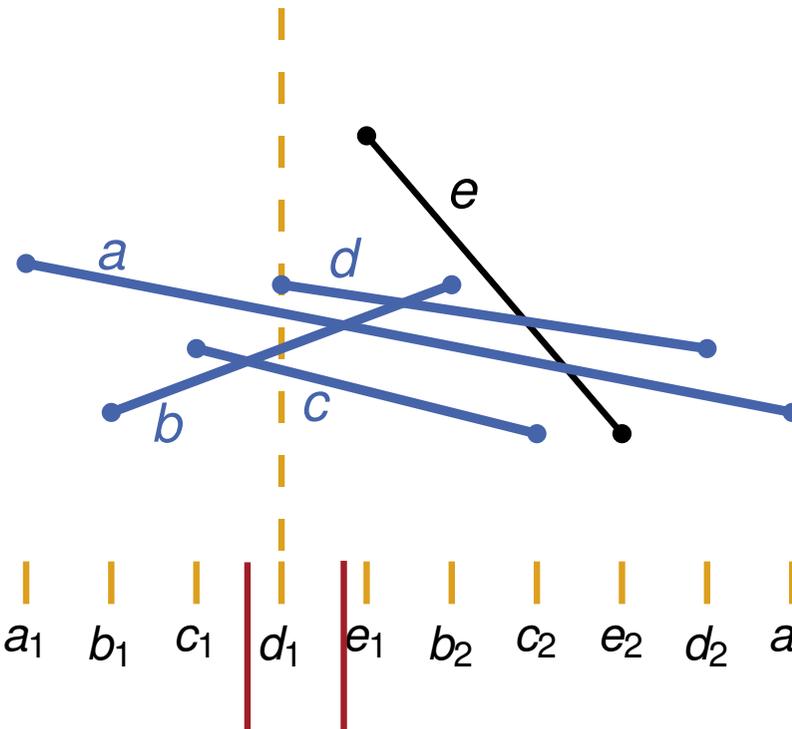
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*

# Problem 1(b) – Beispiel

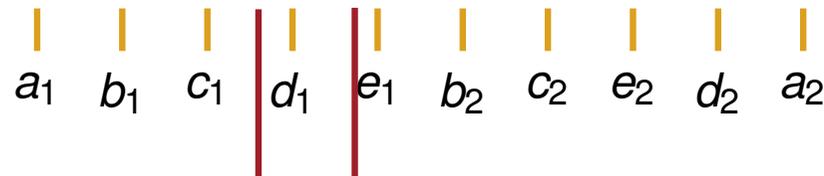
Sweep-Line Zustand



*d*  
*a*  
*b*  
*c*



Event-Point Schedule →



Paare die aktuell auf Schnitt getestet werden müssen: *a* und *d*

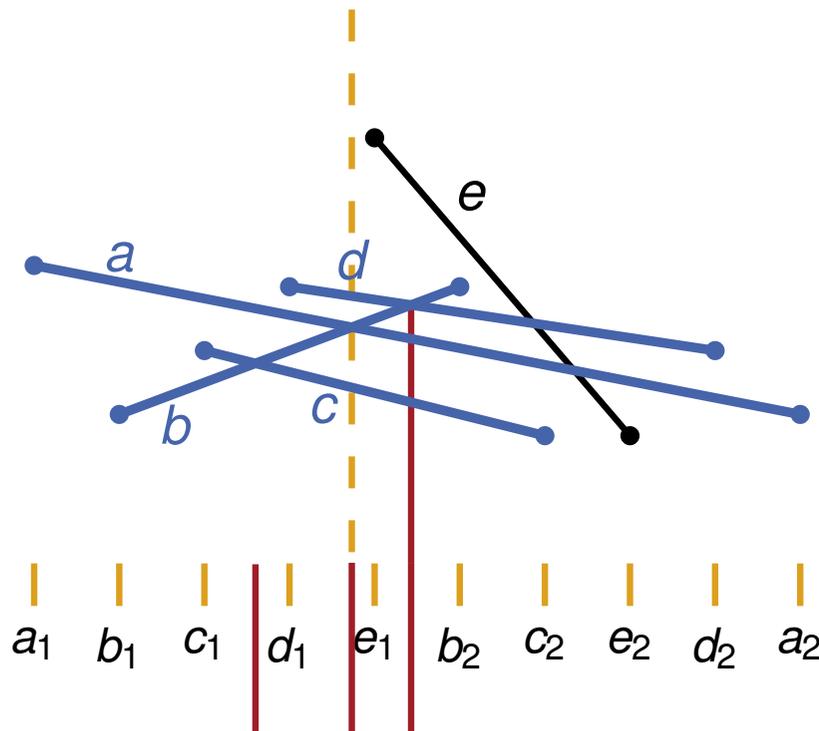
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*

# Problem 1(b) – Beispiel

Sweep-Line Zustand



*d*  
*b*  
*a*  
*c*



Event-Point Schedule →

Paare die aktuell auf Schnitt getestet werden müssen: *c* und *a*, *b* und *d*

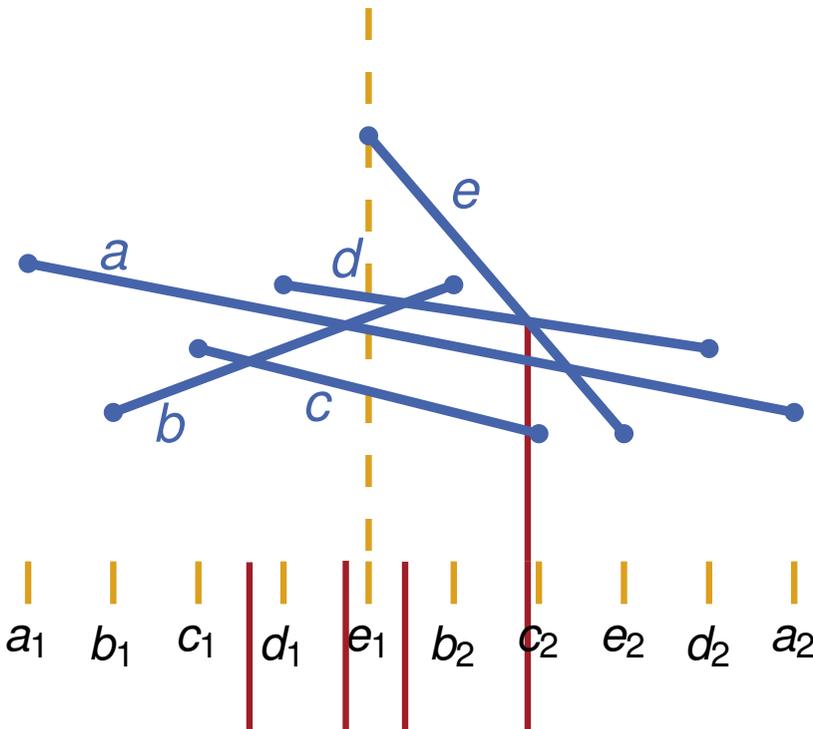
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*

# Problem 1(b) – Beispiel

Sweep-Line Zustand



*e*  
*d*  
*b*  
*a*  
*c*



Event-Point Schedule →

Paare die aktuell auf Schnitt getestet werden müssen: *d* und *e*

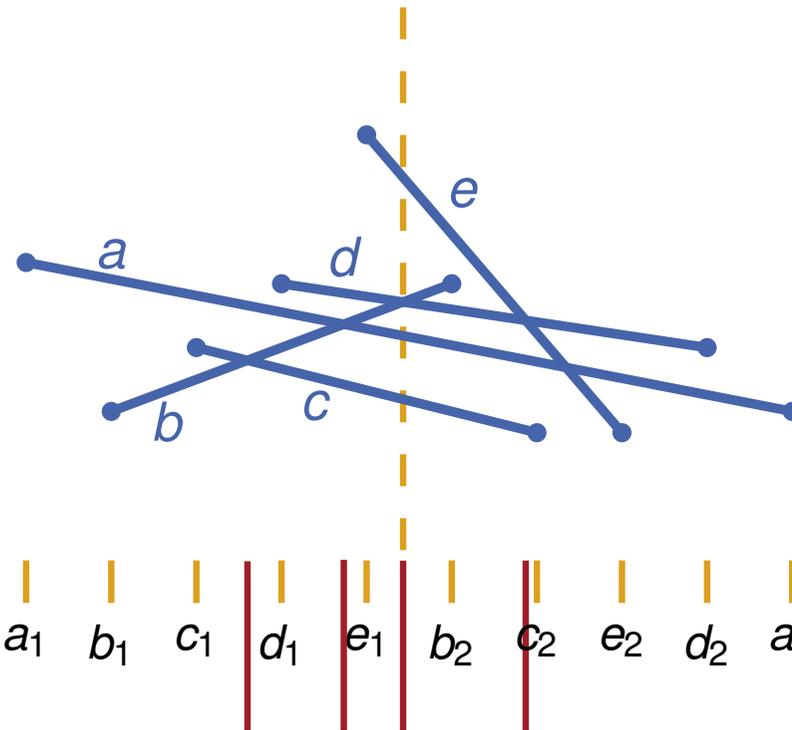
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*, *d* und *e*

# Problem 1(b) – Beispiel

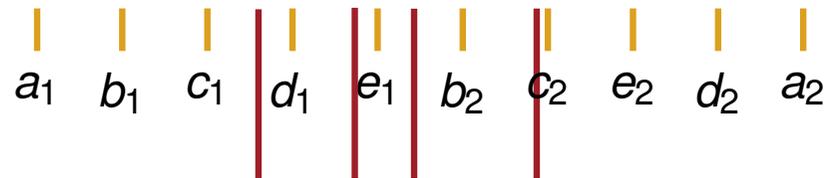
Sweep-Line Zustand



*e*  
*b*  
*d*  
*a*  
*c*



Event-Point Schedule →



Paare die aktuell auf Schnitt getestet werden müssen: *a* und *d*, *b* und *e*

Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*, *d* und *e*

# Problem 1(b) – Beispiel

Sweep-Line Zustand

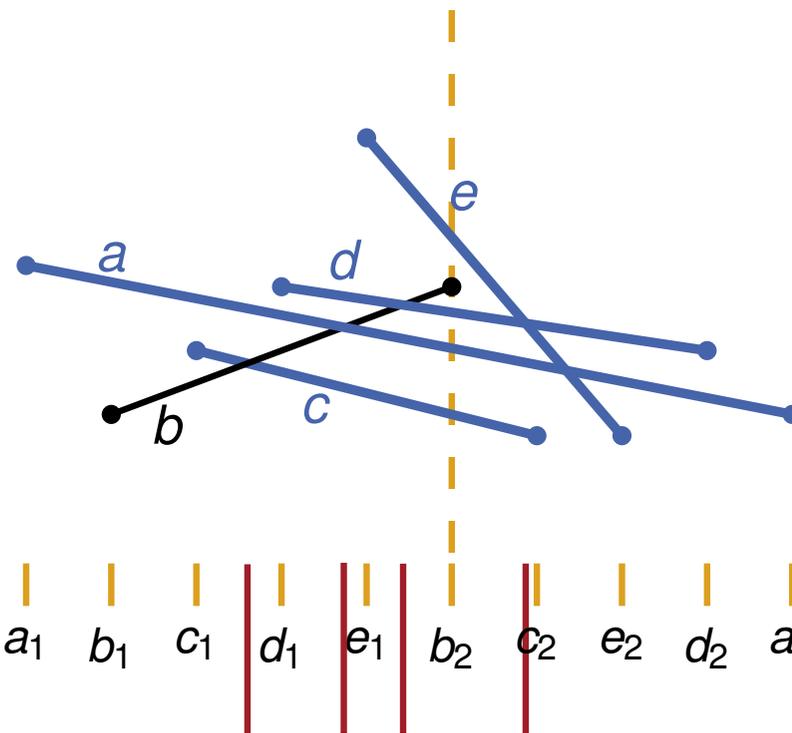


*e*

*d*

*a*

*c*



Paare die aktuell auf Schnitt getestet werden müssen: *d* und *e*

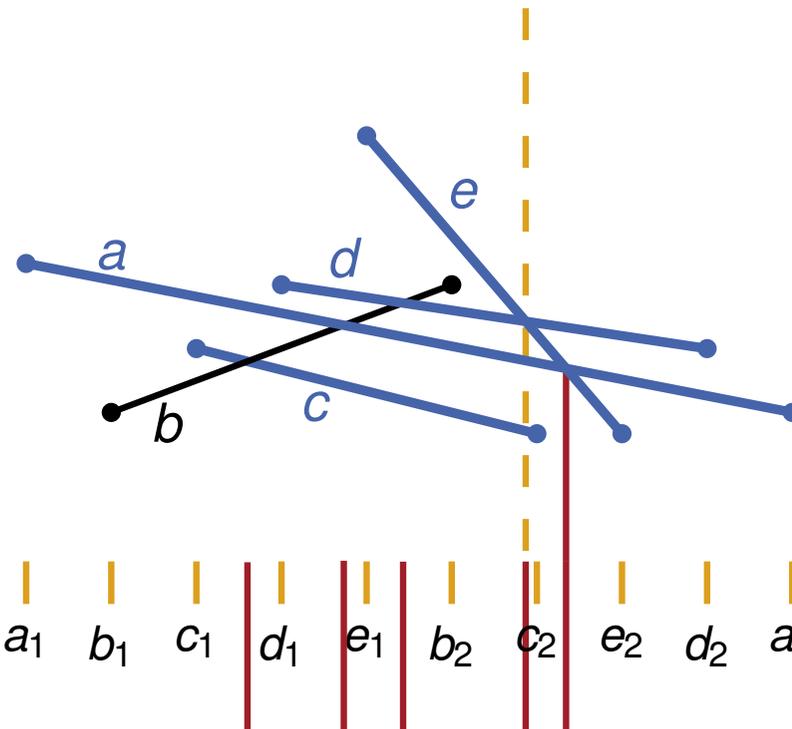
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*, *d* und *e*

# Problem 1(b) – Beispiel

Sweep-Line Zustand



*d*  
*e*  
*a*  
*c*



Event-Point Schedule →

Paare die aktuell auf Schnitt getestet werden müssen: *a* und *e*

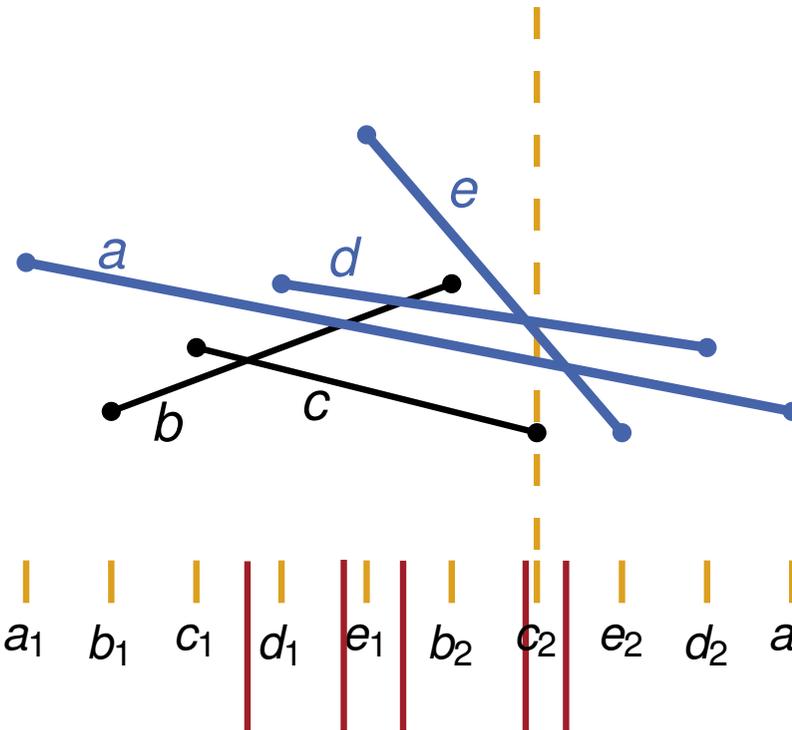
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*, *d* und *e*, *a* und *e*

# Problem 1(b) – Beispiel

Sweep-Line Zustand



*d*  
*e*  
*a*



Event-Point Schedule →

Paare die aktuell auf Schnitt getestet werden müssen:

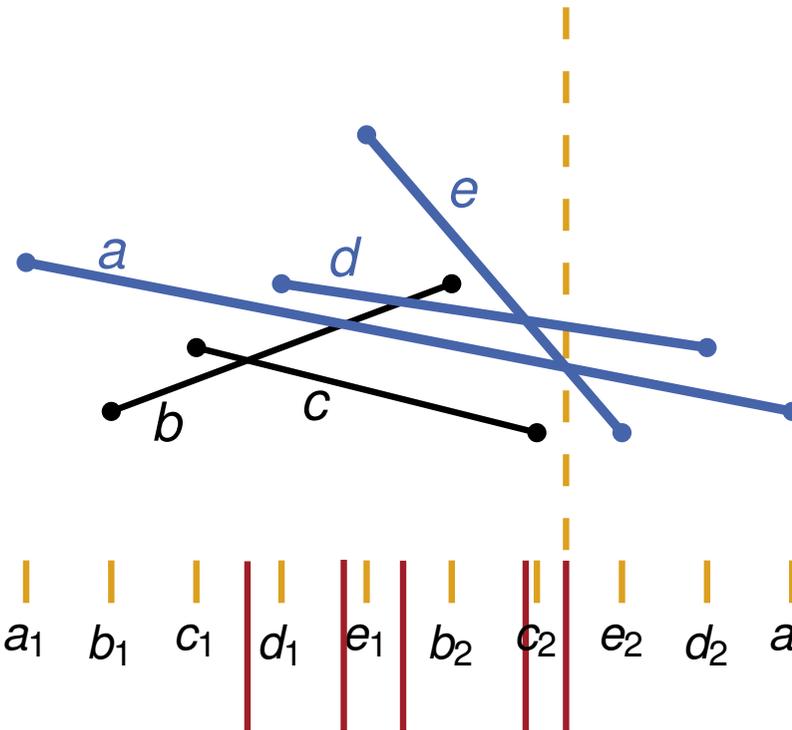
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*, *d* und *e*, *a* und *e*

# Problem 1(b) – Beispiel

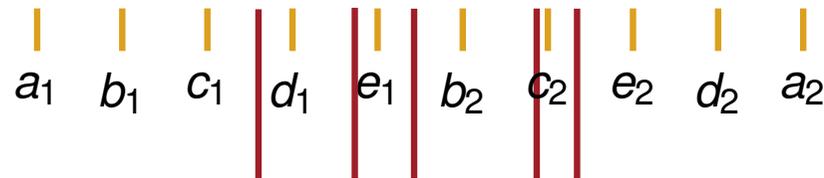
Sweep-Line Zustand



*d*  
*a*  
*e*



Event-Point Schedule →



Paare die aktuell auf Schnitt getestet werden müssen: *a* und *d*

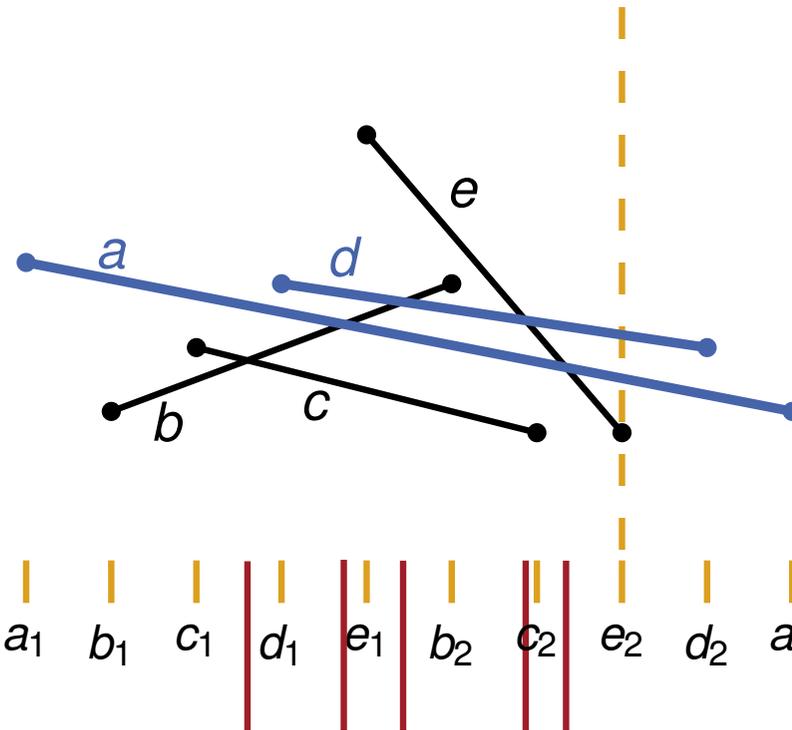
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*, *d* und *e*, *a* und *e*

# Problem 1(b) – Beispiel

Sweep-Line Zustand



*d*  
*a*



Event-Point Schedule →

Paare die aktuell auf Schnitt getestet werden müssen:

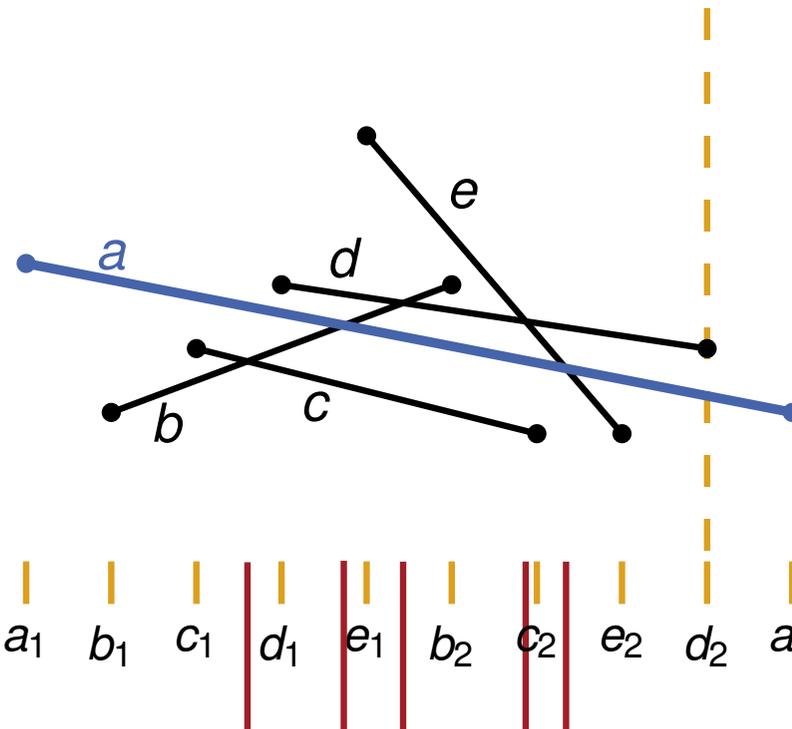
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*, *d* und *e*, *a* und *e*

# Problem 1(b) – Beispiel

Sweep-Line Zustand



*a*



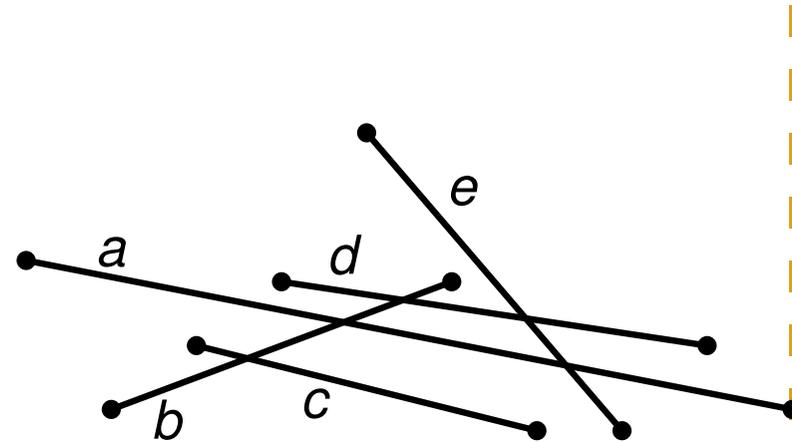
Event-Point Schedule →

Paare die aktuell auf Schnitt getestet werden müssen:

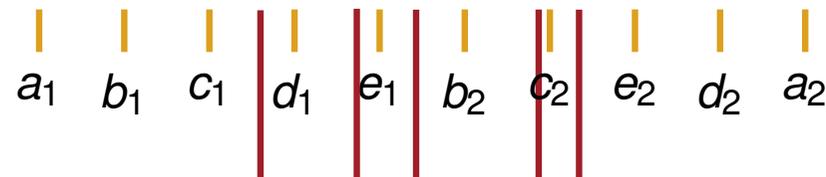
Schnitt gefunden zwischen: *a* und *b*, *b* und *c*, *b* und *d*, *d* und *e*, *a* und *e*

# Problem 1(b) – Beispiel

Sweep-Line Zustand



Event-Point Schedule →



Paare die aktuell auf Schnitt getestet werden müssen:

Schnitt gefunden zwischen:  $a$  und  $b$ ,  $b$  und  $c$ ,  $b$  und  $d$ ,  $d$  und  $e$ ,  $a$  und  $e$

# Problem 1(b)

INTERSECTIONS( $S$ )

$T \leftarrow \emptyset$

$L \leftarrow$  Sortiere Endpunkte der Strecken von links nach rechts

**foreach** Haltepunkt  $p$  in  $L$  **do**

**if**  $p$  linker Endpunkt von Strecke  $s$  **then**

        INSERT( $T, s$ )

        CHECK-FOR-INTERSECTION( $s, \text{ABOVE}(T, s), L$ )

        CHECK-FOR-INTERSECTION( $s, \text{BELOW}(T, s), L$ )

**if**  $p$  rechter Endpunkt von Strecke  $s$  **then**

        CHECK-FOR-INTERSECTION( $\text{ABOVE}(T, s), \text{BELOW}(T, s), L$ )

        DELETE( $T, s$ )

**if**  $p$  Schnittpunkt von Strecken  $s_1$  und  $s_2$  (mit  $s_1 < s_2$  in  $T$ ) **then**

        EXCHANGE( $T, s_1, s_2$ )

        CHECK-FOR-INTERSECTION( $s_1, \text{ABOVE}(T, s), L$ )

        CHECK-FOR-INTERSECTION( $s_2, \text{BELOW}(T, s), L$ )

Erinnerung: ABOVE( $T, s$ ) (BELOW( $T, s$ )) ist die Strecke oberhalb (unterhalb) von  $s$ .

Außerdem: EXCHANGE( $T, s_1, s_2$ ) vertauscht die Reihenfolge von  $s_1$  und  $s_2$  in  $T$ .

CHECK-FOR-INTERSECTION( $s_1, s_2, L$ )

**if**  $s_1$  und  $s_2$  schneiden sich im Punkt  $p$  mit  $p \notin L$  **then**

        Gib aus: „Schnittpunkt zwischen  $s_1$  und  $s_2$ :  $p$ “

        INSERT( $L, p$ )

# Problem 1(b)

INTERSECTIONS( $S$ )

$T \leftarrow \emptyset$

$L \leftarrow$  Sortiere Endpunkte der Strecken von links nach rechts

**foreach** Haltepunkt  $p$  in  $L$  **do**

**if**  $p$  linker Endpunkt von Strecke  $s$  **then**

        INSERT( $T, s$ )

        CHECK-FOR-INTERSECTION( $s, \text{ABOVE}(T, s), L$ )

        CHECK-FOR-INTERSECTION( $s, \text{BELOW}(T, s), L$ )

**if**  $p$  rechter Endpunkt von Strecke  $s$  **then**

        CHECK-FOR-INTERSECTION( $\text{ABOVE}(T, s), \text{BELOW}(T, s), L$ )

        DELETE( $T, s$ )

**if**  $p$  Schnittpunkt von Strecken  $s_1$  und  $s_2$  (mit  $s_1 < s_2$  in  $T$ ) **then**

        EXCHANGE( $T, s_1, s_2$ )

        CHECK-FOR-INTERSECTION( $s_1, \text{ABOVE}(T, s), L$ )

        CHECK-FOR-INTERSECTION( $s_2, \text{BELOW}(T, s), L$ )

Erinnerung: ABOVE( $T, s$ ) (BELOW( $T, s$ )) ist die Strecke oberhalb (unterhalb) von  $s$ .

Außerdem: EXCHANGE( $T, s_1, s_2$ ) vertauscht die Reihenfolge von  $s_1$  und  $s_2$  in  $T$ .

CHECK-FOR-INTERSECTION( $s_1, s_2, L$ )

$O(\log n)$

**if**  $s_1$  und  $s_2$  schneiden sich im Punkt  $p$  mit  $p \notin L$  **then**

        Gib aus: „Schnittpunkt zwischen  $s_1$  und  $s_2$ :  $p$ “

        INSERT( $L, p$ )

# Problem 1(b)

INTERSECTIONS( $S$ )

$T \leftarrow \emptyset$

$L \leftarrow$  Sortiere Endpunkte der Strecken von links nach rechts

**foreach** Haltepunkt  $p$  in  $L$  **do**

<b>if</b> $p$ linker Endpunkt von Strecke $s$ <b>then</b> INSERT( $T, s$ ) CHECK-FOR-INTERSECTION( $s, \text{ABOVE}(T, s), L$ ) CHECK-FOR-INTERSECTION( $s, \text{BELOW}(T, s), L$ )	$O(\log n)$
---	-------------

<b>if</b> $p$ rechter Endpunkt von Strecke $s$ <b>then</b> CHECK-FOR-INTERSECTION( $\text{ABOVE}(T, s), \text{BELOW}(T, s), L$ ) DELETE( $T, s$ )	$O(\log n)$
---	-------------

<b>if</b> $p$ Schnittpunkt von Strecken $s_1$ und $s_2$ (mit $s_1 < s_2$ in $T$ ) <b>then</b> EXCHANGE( $T, s_1, s_2$ ) CHECK-FOR-INTERSECTION( $s_1, \text{ABOVE}(T, s), L$ ) CHECK-FOR-INTERSECTION( $s_2, \text{BELOW}(T, s), L$ )	$O(\log n)$
--	-------------

Erinnerung: ABOVE( $T, s$ ) (BELOW( $T, s$ )) ist die Strecke oberhalb (unterhalb) von  $s$ .  
Außerdem: EXCHANGE( $T, s_1, s_2$ ) vertauscht die Reihenfolge von  $s_1$  und  $s_2$  in  $T$ .

CHECK-FOR-INTERSECTION( $s_1, s_2, L$ ) <b>if</b> $s_1$ und $s_2$ schneiden sich im Punkt $p$ mit $p \notin L$ <b>then</b> Gib aus: „Schnittpunkt zwischen $s_1$ und $s_2$ : $p$ “ INSERT( $L, p$ )	$O(\log n)$
--	-------------

# Problem 1(b)

INTERSECTIONS( $S$ )

$T \leftarrow \emptyset$

$L \leftarrow$  Sortiere Endpunkte der Strecken von links nach rechts  $O(n \log n)$

**foreach** Haltepunkt  $p$  in  $L$  **do**  $O((n + k) \log n)$

**if**  $p$  linker Endpunkt von Strecke  $s$  **then**  $O(\log n)$

    INSERT( $T, s$ )

    CHECK-FOR-INTERSECTION( $s, \text{ABOVE}(T, s), L$ )

    CHECK-FOR-INTERSECTION( $s, \text{BELOW}(T, s), L$ )

**if**  $p$  rechter Endpunkt von Strecke  $s$  **then**  $O(\log n)$

    CHECK-FOR-INTERSECTION( $\text{ABOVE}(T, s), \text{BELOW}(T, s), L$ )

    DELETE( $T, s$ )

**if**  $p$  Schnittpunkt von Strecken  $s_1$  und  $s_2$  (mit  $s_1 < s_2$  in  $T$ ) **then**  $O(\log n)$

    EXCHANGE( $T, s_1, s_2$ )

    CHECK-FOR-INTERSECTION( $s_1, \text{ABOVE}(T, s), L$ )

    CHECK-FOR-INTERSECTION( $s_2, \text{BELOW}(T, s), L$ )

Erinnerung: ABOVE( $T, s$ ) (BELOW( $T, s$ )) ist die Strecke oberhalb (unterhalb) von  $s$ .

Außerdem: EXCHANGE( $T, s_1, s_2$ ) vertauscht die Reihenfolge von  $s_1$  und  $s_2$  in  $T$ .

CHECK-FOR-INTERSECTION( $s_1, s_2, L$ )  $O(\log n)$

**if**  $s_1$  und  $s_2$  schneiden sich im Punkt  $p$  mit  $p \notin L$  **then**

    Gib aus: „Schnittpunkt zwischen  $s_1$  und  $s_2$ :  $p$ “

    INSERT( $L, p$ )

# Problem 1(b)

INTERSECTIONS( $S$ )

$O((n + k) \log n)$

$T \leftarrow \emptyset$

$L \leftarrow$  Sortiere Endpunkte der Strecken von links nach rechts

$O(n \log n)$

**foreach** Haltepunkt  $p$  in  $L$  **do**

$O((n + k) \log n)$

**if**  $p$  linker Endpunkt von Strecke  $s$  **then**

$O(\log n)$

    INSERT( $T, s$ )

    CHECK-FOR-INTERSECTION( $s, \text{ABOVE}(T, s), L$ )

    CHECK-FOR-INTERSECTION( $s, \text{BELOW}(T, s), L$ )

**if**  $p$  rechter Endpunkt von Strecke  $s$  **then**

$O(\log n)$

    CHECK-FOR-INTERSECTION( $\text{ABOVE}(T, s), \text{BELOW}(T, s), L$ )

    DELETE( $T, s$ )

**if**  $p$  Schnittpunkt von Strecken  $s_1$  und  $s_2$  (mit  $s_1 < s_2$  in  $T$ ) **then**

$O(\log n)$

    EXCHANGE( $T, s_1, s_2$ )

    CHECK-FOR-INTERSECTION( $s_1, \text{ABOVE}(T, s), L$ )

    CHECK-FOR-INTERSECTION( $s_2, \text{BELOW}(T, s), L$ )

Erinnerung: ABOVE( $T, s$ ) (BELOW( $T, s$ )) ist die Strecke oberhalb (unterhalb) von  $s$ .

Außerdem: EXCHANGE( $T, s_1, s_2$ ) vertauscht die Reihenfolge von  $s_1$  und  $s_2$  in  $T$ .

CHECK-FOR-INTERSECTION( $s_1, s_2, L$ )

$O(\log n)$

**if**  $s_1$  und  $s_2$  schneiden sich im Punkt  $p$  mit  $p \notin L$  **then**

    Gib aus: „Schnittpunkt zwischen  $s_1$  und  $s_2$ :  $p$ “

    INSERT( $L, p$ )

# Problem 1(c)

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.
- (c) Zeigen Sie, dass Ihr Algorithmus mit  $O(n + k)$  Speicherplatz auskommt. Ist es möglich, den nötigen Speicherplatz auf  $O(n)$  zu reduzieren?

# Problem 1(c)

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.
- (c) Zeigen Sie, dass Ihr Algorithmus mit  $O(n + k)$  Speicherplatz auskommt. Ist es möglich, den nötigen Speicherplatz auf  $O(n)$  zu reduzieren?
- Der Sweep-Line Zustand speichert alle Strecken, die von der aktuellen Sweep-Line geschnitten werden  $\Rightarrow O(n)$  Speicherplatz.

# Problem 1(c)

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.
- (c) Zeigen Sie, dass Ihr Algorithmus mit  $O(n + k)$  Speicherplatz auskommt. Ist es möglich, den nötigen Speicherplatz auf  $O(n)$  zu reduzieren?
- Der Sweep-Line Zustand speichert alle Strecken, die von der aktuellen Sweep-Line geschnitten werden  $\Rightarrow O(n)$  Speicherplatz.
  - Die Haltepunkte im Event-Point Schedule sind entweder Endpunkte von Strecken oder Kreuzungspunkte.  $\Rightarrow O(n + k)$  Speicherplatz.

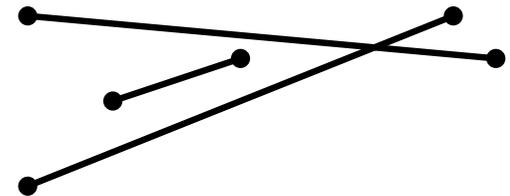
**Idee zur Verbesserung auf  $O(n)$ :**

# Problem 1(c)

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.
- (c) Zeigen Sie, dass Ihr Algorithmus mit  $O(n + k)$  Speicherplatz auskommt. Ist es möglich, den nötigen Speicherplatz auf  $O(n)$  zu reduzieren?
- Der Sweep-Line Zustand speichert alle Strecken, die von der aktuellen Sweep-Line geschnitten werden  $\Rightarrow O(n)$  Speicherplatz.
  - Die Haltepunkte im Event-Point Schedule sind entweder Endpunkte von Strecken oder Kreuzungspunkte.  $\Rightarrow O(n + k)$  Speicherplatz.

## Idee zur Verbesserung auf $O(n)$ :

- Lösche schon berechnete Kreuzungen wieder aus dem Event-Point Schedule, wenn die zugehörigen Strecken nicht mehr benachbart sind.

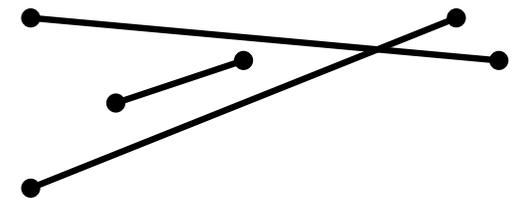


# Problem 1(c)

- (b) Geben Sie einen Algorithmus an, der *alle* Schnitte zwischen Streckenpaaren in  $S$  in  $O((n + k) \cdot \log n)$  Zeit berechnet und ausgibt, wobei  $k$  die Anzahl der Schnittpunkte ist.
- (c) Zeigen Sie, dass Ihr Algorithmus mit  $O(n + k)$  Speicherplatz auskommt. Ist es möglich, den nötigen Speicherplatz auf  $O(n)$  zu reduzieren?
- Der Sweep-Line Zustand speichert alle Strecken, die von der aktuellen Sweep-Line geschnitten werden  $\Rightarrow O(n)$  Speicherplatz.
  - Die Haltepunkte im Event-Point Schedule sind entweder Endpunkte von Strecken oder Kreuzungspunkte.  $\Rightarrow O(n + k)$  Speicherplatz.

## Idee zur Verbesserung auf $O(n)$ :

- Lösche schon berechnete Kreuzungen wieder aus dem Event-Point Schedule, wenn die zugehörigen Strecken nicht mehr benachbart sind.
- Zwei Strecken die sich schneiden aber im aktuellen Sweep-Line Zustand nicht benachbart sind, werden vor dem Schnittpunkt nochmal benachbart sein.

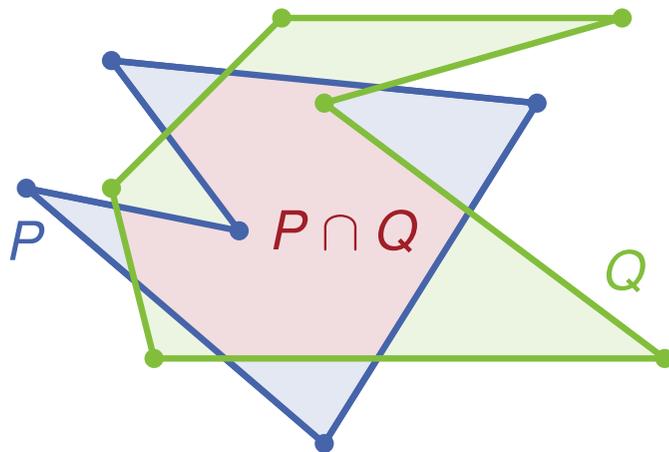


# Schnitte von Polygonen

# Problem 2(a)

Seien  $P$  und  $Q$  zwei einfache Polygone. Als *Schnitt*  $P \cap Q$  von  $P$  und  $Q$  bezeichnen wir die Menge der Punkte, die sowohl im Inneren von  $P$  als auch im Inneren von  $Q$  liegen. Gehen Sie wie folgt vor, um einen Algorithmus mit  $O((n + k) \cdot \log n)$  Laufzeit zur Berechnung von  $P \cap Q$  zu entwerfen, wobei  $k$  die Anzahl der Schnittpunkte zwischen Polygonkanten ist.

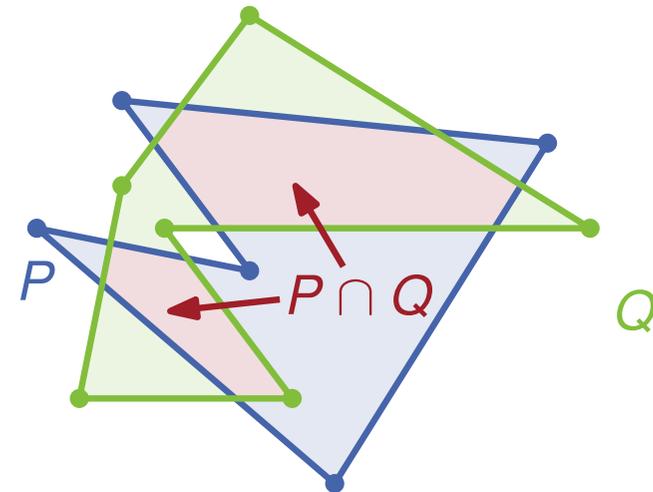
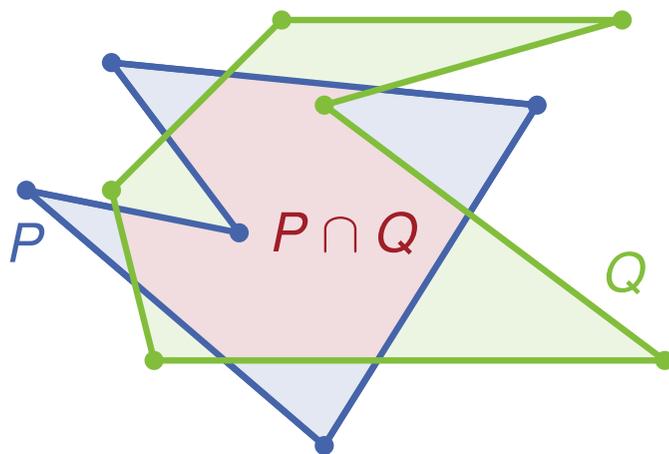
- (a) Machen Sie sich klar, dass  $P \cap Q$  aus dem Inneren mehrerer Polygone bestehen kann.



# Problem 2(a)

Seien  $P$  und  $Q$  zwei einfache Polygone. Als *Schnitt*  $P \cap Q$  von  $P$  und  $Q$  bezeichnen wir die Menge der Punkte, die sowohl im Inneren von  $P$  als auch im Inneren von  $Q$  liegen. Gehen Sie wie folgt vor, um einen Algorithmus mit  $O((n + k) \cdot \log n)$  Laufzeit zur Berechnung von  $P \cap Q$  zu entwerfen, wobei  $k$  die Anzahl der Schnittpunkte zwischen Polygonkanten ist.

- (a) Machen Sie sich klar, dass  $P \cap Q$  aus dem Inneren mehrerer Polygone bestehen kann.



# Problem 2(b)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

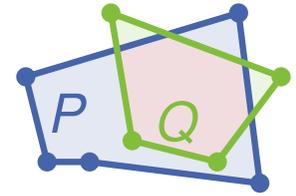
1. Der Schnitt  $P \cap Q$  ist leer.



2.  $P$  ist in  $Q$  enthalten oder umgekehrt.



3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .



# Problem 2(b)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

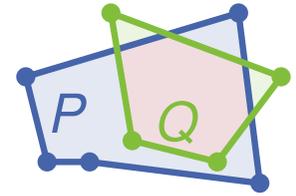
1. Der Schnitt  $P \cap Q$  ist leer.



2.  $P$  ist in  $Q$  enthalten oder umgekehrt.



3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .



Teste zunächst ob **Fall 3** eintritt:

# Problem 2(b)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

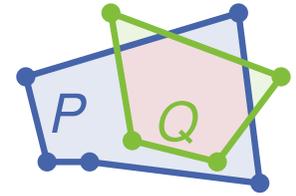
1. Der Schnitt  $P \cap Q$  ist leer.



2.  $P$  ist in  $Q$  enthalten oder umgekehrt.



3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .



Teste zunächst ob **Fall 3** eintritt:

- Tritt genau dann ein, wenn es in  $P$  eine Kante gibt, die eine Kante in  $Q$  schneidet.
- Kann in  $O(n \log n)$  Zeit mit Verfahren aus der Vorlesung getestet werden.

# Problem 2(b)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

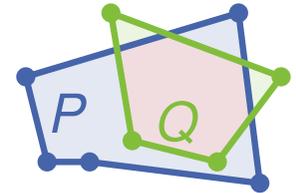
1. Der Schnitt  $P \cap Q$  ist leer.



2.  $P$  ist in  $Q$  enthalten oder umgekehrt.



3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .



Teste zunächst ob **Fall 3** eintritt:

- Tritt genau dann ein, wenn es in  $P$  eine Kante gibt, die eine Kante in  $Q$  schneidet.
- Kann in  $O(n \log n)$  Zeit mit Verfahren aus der Vorlesung getestet werden.

Falls Fall 3 nicht gilt, teste ob **Fall 2** eintritt:

# Problem 2(b)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

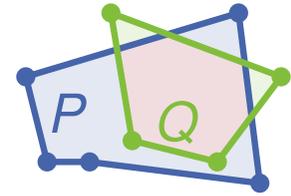
1. Der Schnitt  $P \cap Q$  ist leer.



2.  $P$  ist in  $Q$  enthalten oder umgekehrt.



3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .



Teste zunächst ob **Fall 3** eintritt:

- Tritt genau dann ein, wenn es in  $P$  eine Kante gibt, die eine Kante in  $Q$  schneidet.
- Kann in  $O(n \log n)$  Zeit mit Verfahren aus der Vorlesung getestet werden.

Falls Fall 3 nicht gilt, teste ob **Fall 2** eintritt:

- Tritt genau dann ein, wenn alle Punkte aus  $P$  in  $Q$  liegen oder umgekehrt.

# Problem 2(b)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

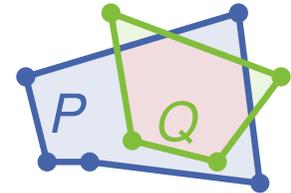
1. Der Schnitt  $P \cap Q$  ist leer.



2.  $P$  ist in  $Q$  enthalten oder umgekehrt.



3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .



Teste zunächst ob **Fall 3** eintritt:

- Tritt genau dann ein, wenn es in  $P$  eine Kante gibt, die eine Kante in  $Q$  schneidet.
- Kann in  $O(n \log n)$  Zeit mit Verfahren aus der Vorlesung getestet werden.

Falls Fall 3 nicht gilt, teste ob **Fall 2** eintritt:

- Tritt genau dann ein, wenn alle Punkte aus  $P$  in  $Q$  liegen oder umgekehrt.
- Da Fall 3 nicht gilt genügt es diese Eigenschaft für einen Knoten von  $P$  (bzw.  $Q$ ) zu testen.

⇒ Wir müssen nur testen, ob ein Punkt in einem Polygon liegt.

# Problem 2(b)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

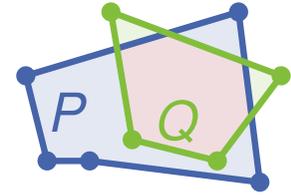
1. Der Schnitt  $P \cap Q$  ist leer.



2.  $P$  ist in  $Q$  enthalten oder umgekehrt.



3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .



Teste zunächst ob **Fall 3** eintritt:

- Tritt genau dann ein, wenn es in  $P$  eine Kante gibt, die eine Kante in  $Q$  schneidet.
- Kann in  $O(n \log n)$  Zeit mit Verfahren aus der Vorlesung getestet werden.

Falls Fall 3 nicht gilt, teste ob **Fall 2** eintritt:

- Tritt genau dann ein, wenn alle Punkte aus  $P$  in  $Q$  liegen oder umgekehrt.
- Da Fall 3 nicht gilt genügt es diese Eigenschaft für einen Knoten von  $P$  (bzw.  $Q$ ) zu testen.

⇒ Wir müssen nur testen, ob ein Punkt in einem Polygon liegt.

**Fall 1** tritt ein, wenn weder Fall 2 noch Fall 3 eintreten.

# Problem 2(b)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

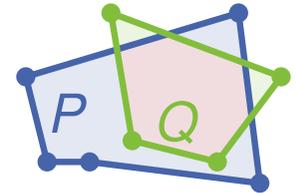
1. Der Schnitt  $P \cap Q$  ist leer.



2.  $P$  ist in  $Q$  enthalten oder umgekehrt.



3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .



Teste zunächst ob **Fall 3** eintritt:

- Tritt genau dann ein, wenn es in  $P$  eine Kante gibt, die eine Kante in  $Q$  schneidet.
- Kann in  $O(n \log n)$  Zeit mit Verfahren aus der Vorlesung getestet werden.

Falls Fall 3 nicht gilt, teste ob **Fall 2** eintritt:

- Tritt genau dann ein, wenn alle Punkte aus  $P$  in  $Q$  liegen oder umgekehrt.
- Da Fall 3 nicht gilt genügt es diese Eigenschaft für einen Knoten von  $P$  (bzw.  $Q$ ) zu testen.

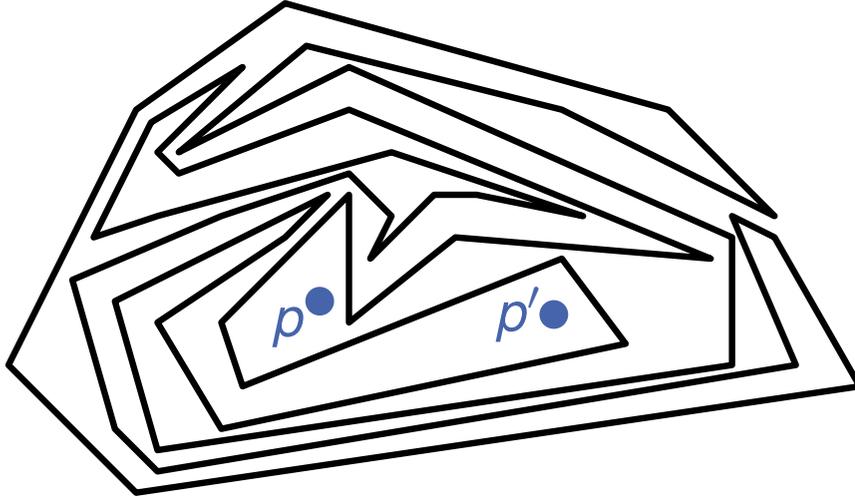
⇒ Wir müssen nur testen, ob ein Punkt in einem Polygon liegt.

Wie geht das?

**Fall 1** tritt ein, wenn weder Fall 2 noch Fall 3 eintreten.

# Problem 2(b)

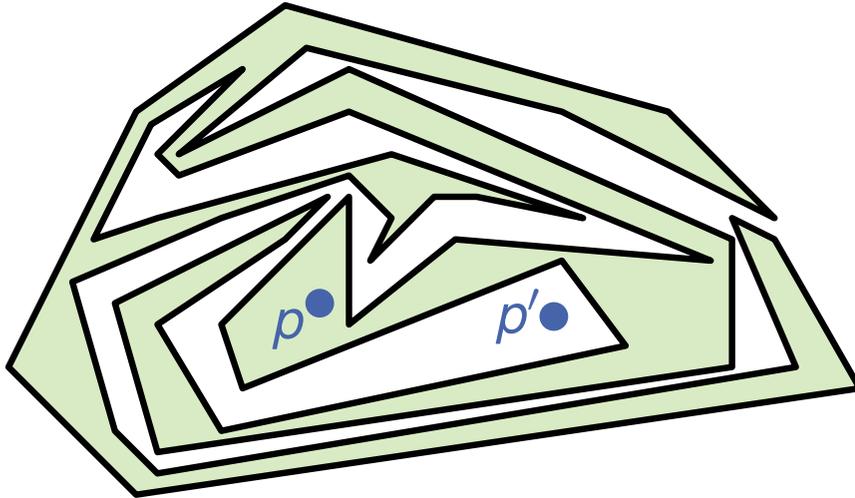
Wie kann man Testen, ob ein Punkt im Inneren eines Polygons  $P$  liegt?



Welcher der Punkte liegt in  $P$  und welcher außerhalb?

## Problem 2(b)

Wie kann man Testen, ob ein Punkt im Inneren eines Polygons  $P$  liegt?



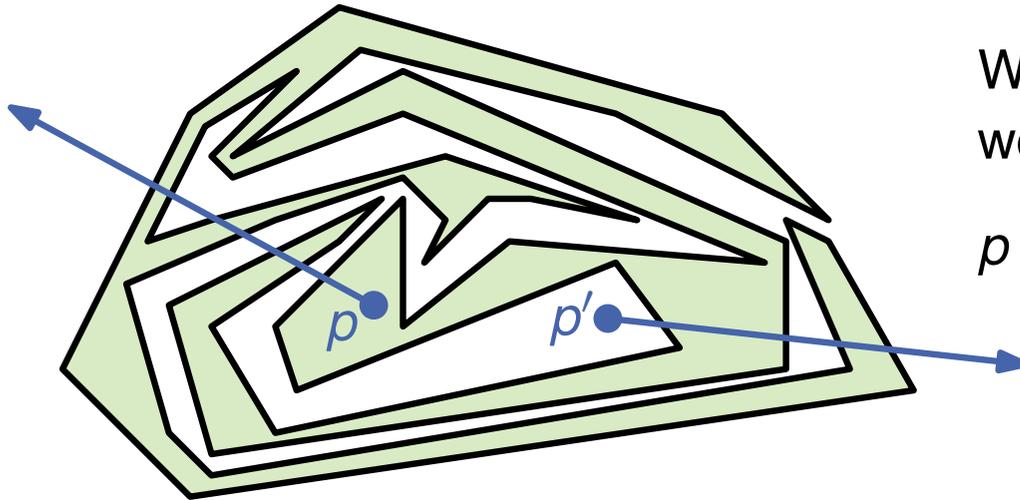
Welcher der Punkte liegt in  $P$  und welcher außerhalb?

$p$  liegt innen,  $p'$  außen!

**Idee für einen Algorithmus:**

## Problem 2(b)

Wie kann man Testen, ob ein Punkt im Inneren eines Polygons  $P$  liegt?



Welcher der Punkte liegt in  $P$  und welcher außerhalb?

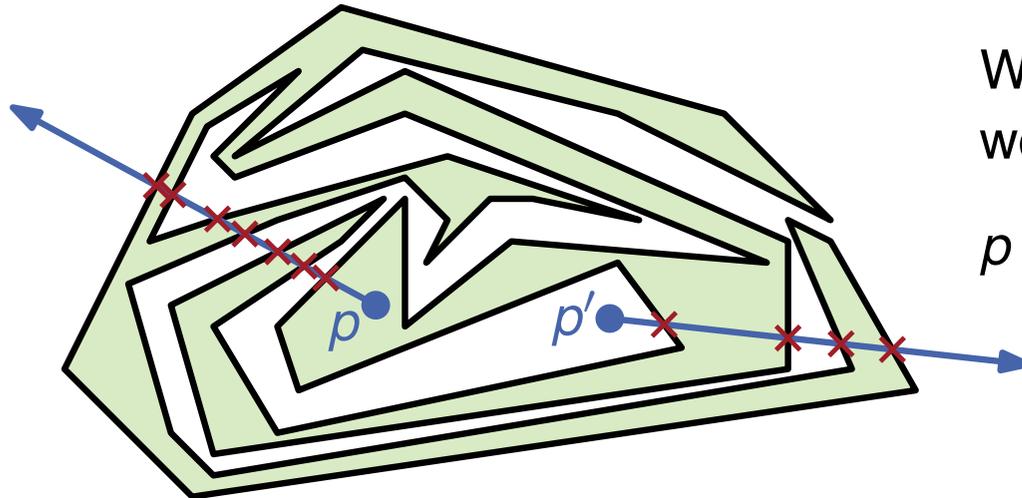
$p$  liegt innen,  $p'$  außen!

**Idee für einen Algorithmus:**

- Schieße Strahl von  $p$  aus in irgendeine Richtung.

## Problem 2(b)

Wie kann man Testen, ob ein Punkt im Inneren eines Polygons  $P$  liegt?



Welcher der Punkte liegt in  $P$  und welcher außerhalb?

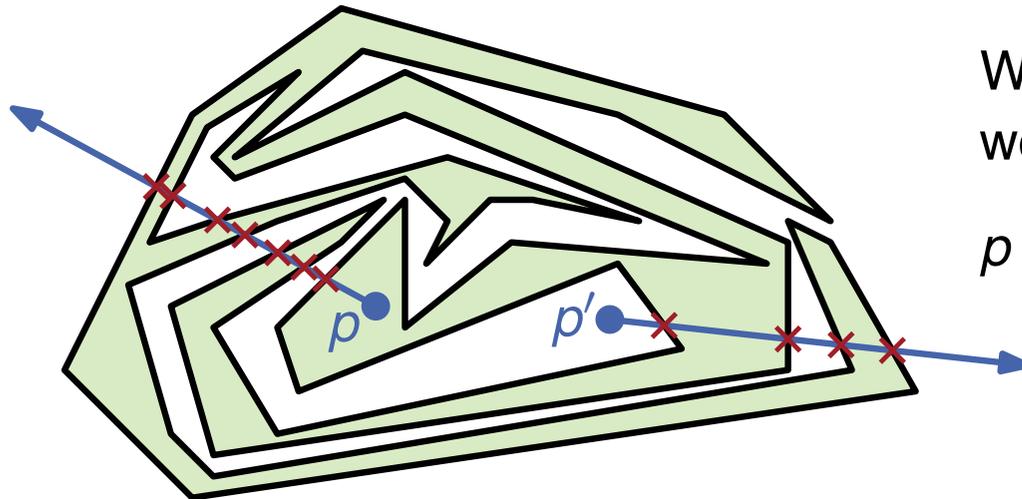
$p$  liegt innen,  $p'$  außen!

### Idee für einen Algorithmus:

- Schieße Strahl von  $p$  aus in irgendeine Richtung.
- Zähle Schnittpunkte mit Polygonkanten:  
bei ungerader Anzahl liegt  $p$  innen, sonst außen.

# Problem 2(b)

Wie kann man Testen, ob ein Punkt im Inneren eines Polygons  $P$  liegt?



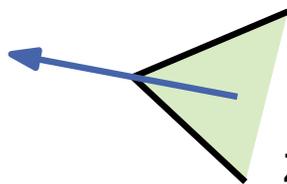
Welcher der Punkte liegt in  $P$  und welcher außerhalb?

$p$  liegt innen,  $p'$  außen!

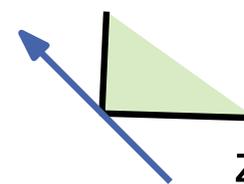
## Idee für einen Algorithmus:

- Schieße Strahl von  $p$  aus in irgendeine Richtung.
- Zähle Schnittpunkte mit Polygonkanten:  
bei ungerader Anzahl liegt  $p$  innen, sonst außen.

**Achtung:** Man muss aufpassen, wenn der Schnittpunkt ein Knoten von  $P$  ist.



zählt als Schnittpunkt



zählt **nicht** als Schnittpunkt

# Problem 2(c)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

1. Der Schnitt  $P \cap Q$  ist leer.
2.  $P$  ist in  $Q$  enthalten oder umgekehrt.
3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .

## Problem 2(c)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

1. Der Schnitt  $P \cap Q$  ist leer.  nichts weiter zu tun
2.  $P$  ist in  $Q$  enthalten oder umgekehrt. 
3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .

(c) Entwickeln Sie im Folgenden einen Sweep-Line Algorithmus, der den Schnitt  $P \cap Q$  für den dritten Fall berechnet. Gehen Sie wie folgt vor.

1. Verwenden Sie als Haltepunkte sowohl die Eckpunkte der beiden Polygone, als auch die Schnittpunkte zwischen Polygonkanten. Welche Fälle können bei Haltepunkten auftreten?

## Problem 2(c)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

1. Der Schnitt  $P \cap Q$  ist leer.  nichts weiter zu tun
2.  $P$  ist in  $Q$  enthalten oder umgekehrt. 
3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .

(c) Entwickeln Sie im Folgenden einen Sweep-Line Algorithmus, der den Schnitt  $P \cap Q$  für den dritten Fall berechnet. Gehen Sie wie folgt vor.

1. Verwenden Sie als Haltepunkte sowohl die Eckpunkte der beiden Polygone, als auch die Schnittpunkte zwischen Polygonkanten. Welche Fälle können bei Haltepunkten auftreten?

**Offensichtliche Unterscheidung:** Ein Haltepunkt  $p$  ist entweder Knoten eines Polygons oder Schnittpunkt zweier Polygonkanten.

## Problem 2(c)

(b) Geben Sie einen Algorithmus mit Laufzeit  $O(n \log n)$  an, der entscheidet welcher der folgenden drei Fälle auftritt.

1. Der Schnitt  $P \cap Q$  ist leer.  nichts weiter zu tun
2.  $P$  ist in  $Q$  enthalten oder umgekehrt. 
3. Der Schnitt  $P \cap Q$  ist nicht leer und weder gleich  $P$  noch gleich  $Q$ .

(c) Entwickeln Sie im Folgenden einen Sweep-Line Algorithmus, der den Schnitt  $P \cap Q$  für den dritten Fall berechnet. Gehen Sie wie folgt vor.

1. Verwenden Sie als Haltepunkte sowohl die Eckpunkte der beiden Polygone, als auch die Schnittpunkte zwischen Polygonkanten. Welche Fälle können bei Haltepunkten auftreten?

**Offensichtliche Unterscheidung:** Ein Haltepunkt  $p$  ist entweder Knoten eines Polygons oder Schnittpunkt zweier Polygonkanten.

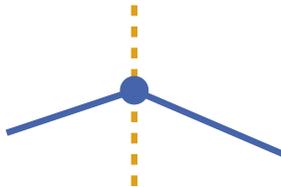
**Aber:** Man kann diese Fälle noch weiter unterteilen.

# Problem 2(c) – Polygonknoten als Haltepunkt

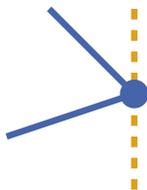
Sei  $p$  ein Knoten aus dem Polygon  $P$  (für  $Q$  analog).

mögliche  
Formen

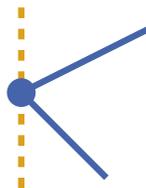
Zwischen-  
knoten



Endknoten



Startknoten



# Problem 2(c) – Polygonknoten als Haltepunkt

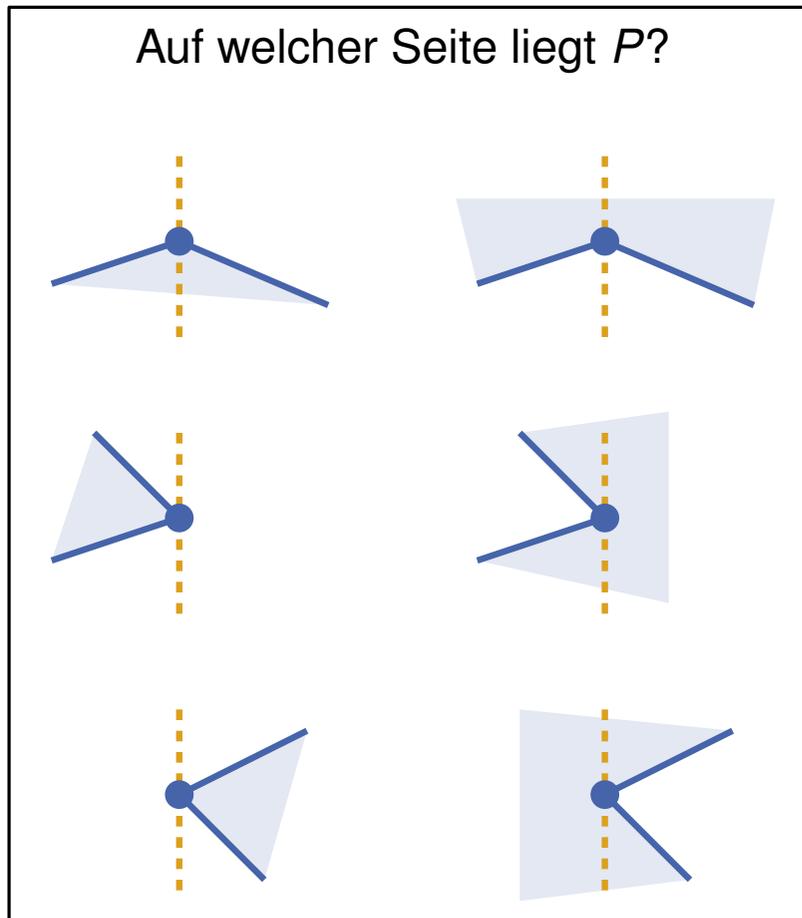
Sei  $p$  ein Knoten aus dem Polygon  $P$  (für  $Q$  analog).

mögliche  
Formen

Zwischen-  
knoten

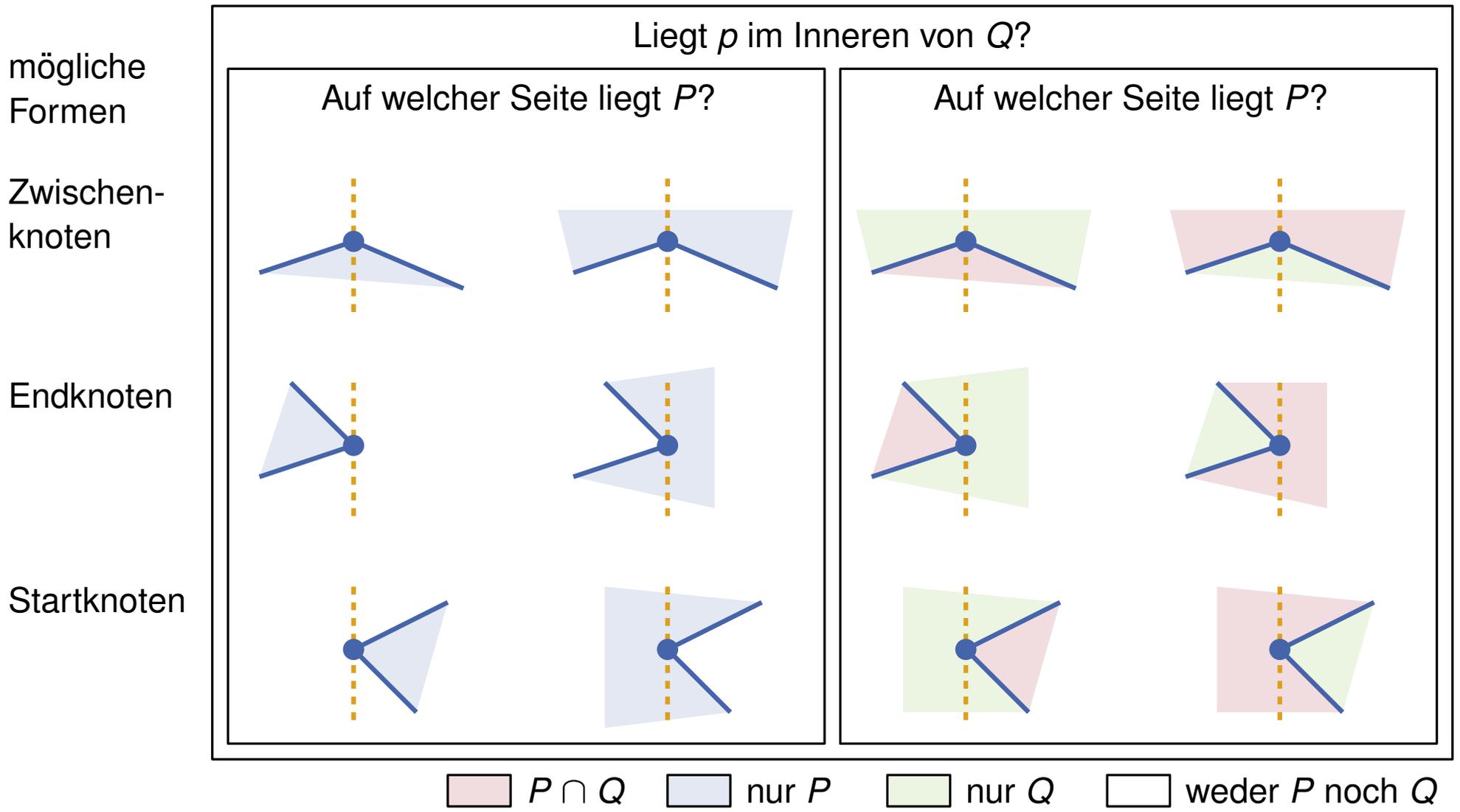
Endknoten

Startknoten



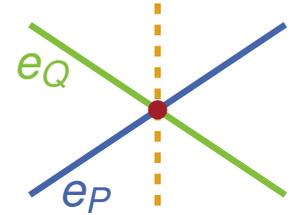
# Problem 2(c) – Polygonknoten als Haltepunkt

Sei  $p$  ein Knoten aus dem Polygon  $P$  (für  $Q$  analog).



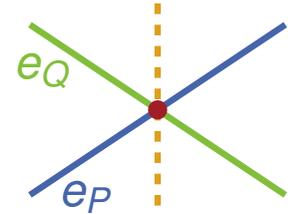
# Problem 2(c) – Schnittpunkt als Haltepunkt

Sei  $p$  Schnittpunkt zwischen Kanten  $e_P$  aus  $P$  und Kante  $e_Q$  aus  $Q$ . Betrachte nur den Fall, dass  $e_P$  vor  $p$  unterhalb von  $e_Q$  liegt (der andere Fall geht analog).



# Problem 2(c) – Schnittpunkt als Haltepunkt

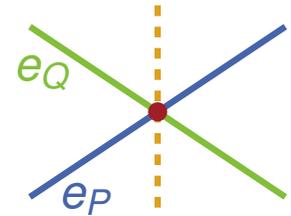
Sei  $p$  Schnittpunkt zwischen Kanten  $e_P$  aus  $P$  und Kante  $e_Q$  aus  $Q$ . Betrachte nur den Fall, dass  $e_P$  vor  $p$  unterhalb von  $e_Q$  liegt (der andere Fall geht analog).



- $P$  kann ober- oder unterhalb von  $e_P$  liegen.
- $Q$  kann ober- oder unterhalb von  $e_Q$  liegen.

# Problem 2(c) – Schnittpunkt als Haltepunkt

Sei  $p$  Schnittpunkt zwischen Kanten  $e_P$  aus  $P$  und Kante  $e_Q$  aus  $Q$ . Betrachte nur den Fall, dass  $e_P$  vor  $p$  unterhalb von  $e_Q$  liegt (der andere Fall geht analog).



- $P$  kann ober- oder unterhalb von  $e_P$  liegen.
- $Q$  kann ober- oder unterhalb von  $e_Q$  liegen.

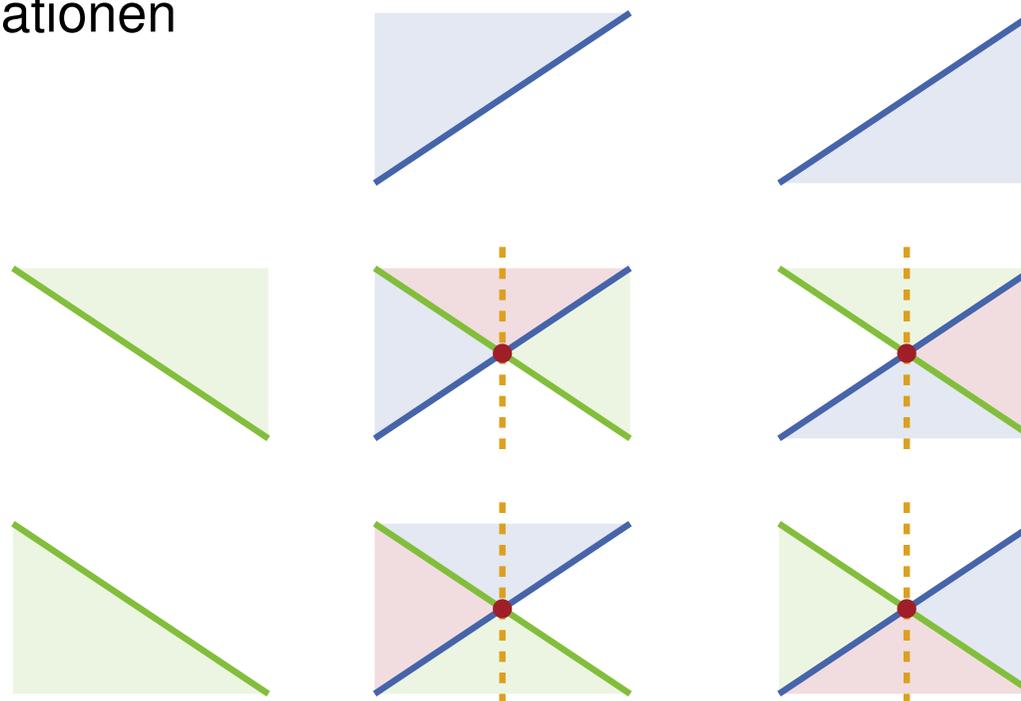
⇒ vier mögliche Kombinationen

  $P \cap Q$

 nur  $P$

 nur  $Q$

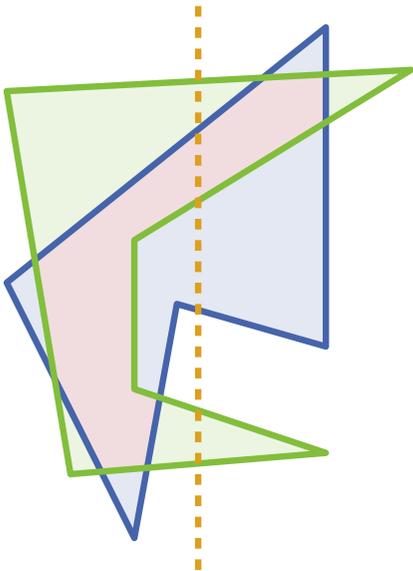
 weder  $P$  noch  $Q$



# Problem 2(c)

(c) Entwickeln Sie im Folgenden einen Sweep-Line Algorithmus, der den Schnitt  $P \cap Q$  für den dritten Fall berechnet. Gehen Sie wie folgt vor.

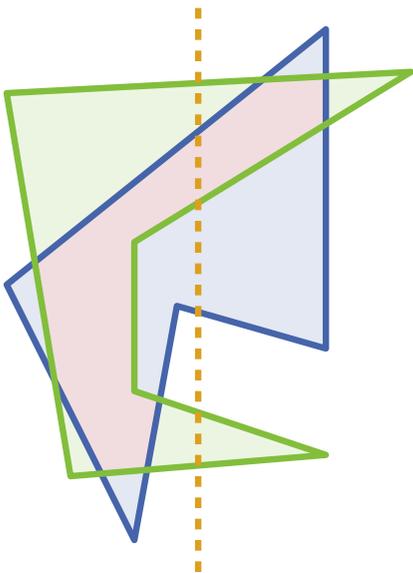
2. Welche Daten sollen als Sweep-Line Zustand gespeichert werden? Dabei sollte zumindest für jeden Punkt auf der aktuellen Sweep-Line bekannt sein, ob er in  $P \cap Q$  liegt oder nicht.



## Problem 2(c)

(c) Entwickeln Sie im Folgenden einen Sweep-Line Algorithmus, der den Schnitt  $P \cap Q$  für den dritten Fall berechnet. Gehen Sie wie folgt vor.

2. Welche Daten sollen als Sweep-Line Zustand gespeichert werden? Dabei sollte zumindest für jeden Punkt auf der aktuellen Sweep-Line bekannt sein, ob er in  $P \cap Q$  liegt oder nicht.

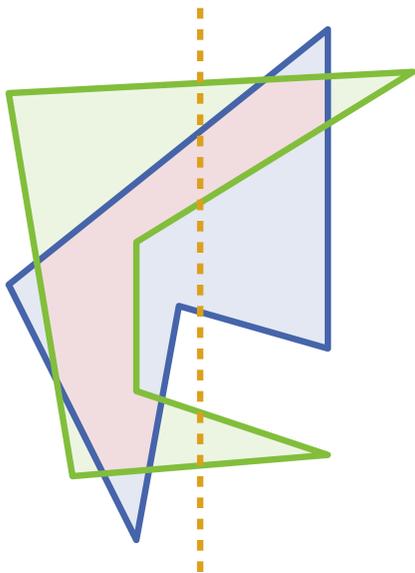


Reihenfolge der Strecken, die die Sweep-Line schneiden.

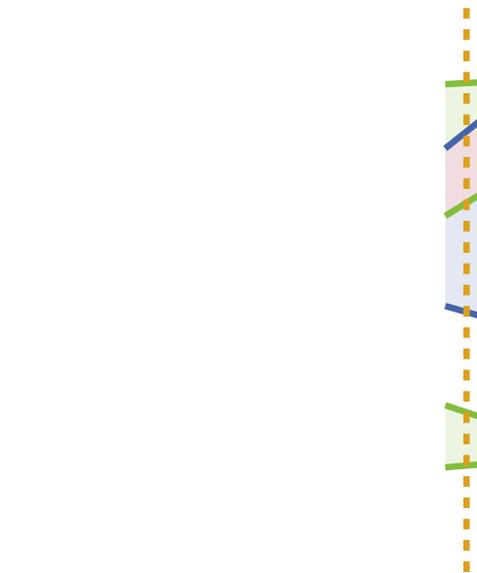
# Problem 2(c)

(c) Entwickeln Sie im Folgenden einen Sweep-Line Algorithmus, der den Schnitt  $P \cap Q$  für den dritten Fall berechnet. Gehen Sie wie folgt vor.

2. Welche Daten sollen als Sweep-Line Zustand gespeichert werden? Dabei sollte zumindest für jeden Punkt auf der aktuellen Sweep-Line bekannt sein, ob er in  $P \cap Q$  liegt oder nicht.



Reihenfolge der Strecken, die die Sweep-Line schneiden.

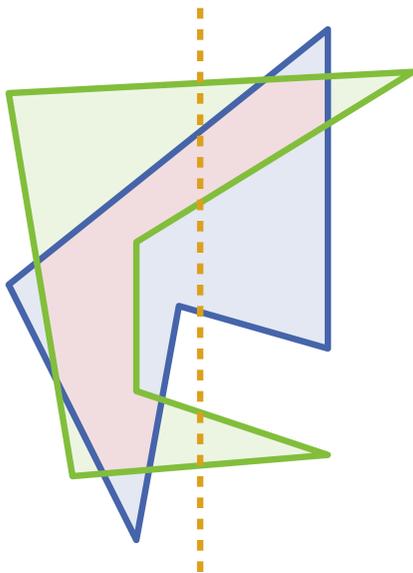


Zerlegung der Sweep-Line in Abschnitte, die in  $P \cap Q$ , nur in  $P$ , nur in  $Q$  oder weder in  $P$  noch in  $Q$  liegen.

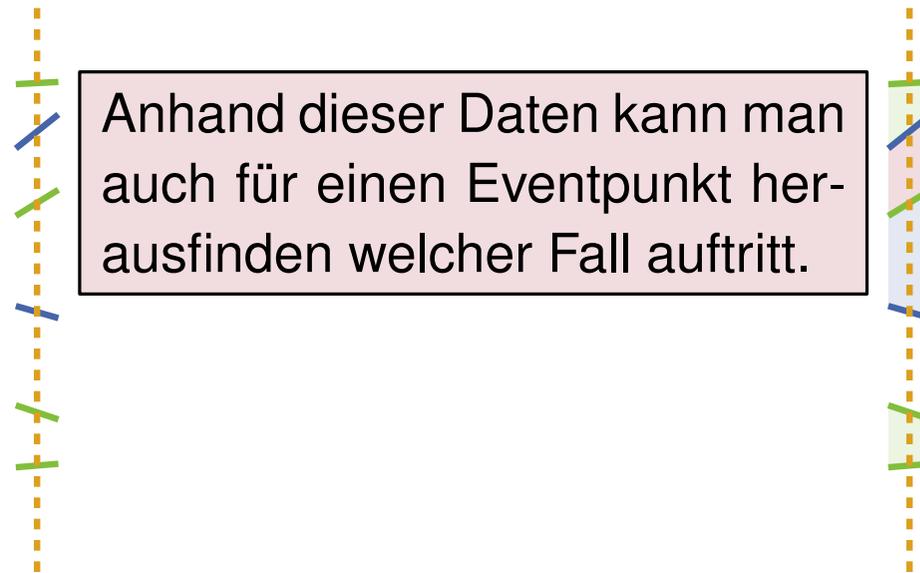
# Problem 2(c)

(c) Entwickeln Sie im Folgenden einen Sweep-Line Algorithmus, der den Schnitt  $P \cap Q$  für den dritten Fall berechnet. Gehen Sie wie folgt vor.

2. Welche Daten sollen als Sweep-Line Zustand gespeichert werden? Dabei sollte zumindest für jeden Punkt auf der aktuellen Sweep-Line bekannt sein, ob er in  $P \cap Q$  liegt oder nicht.



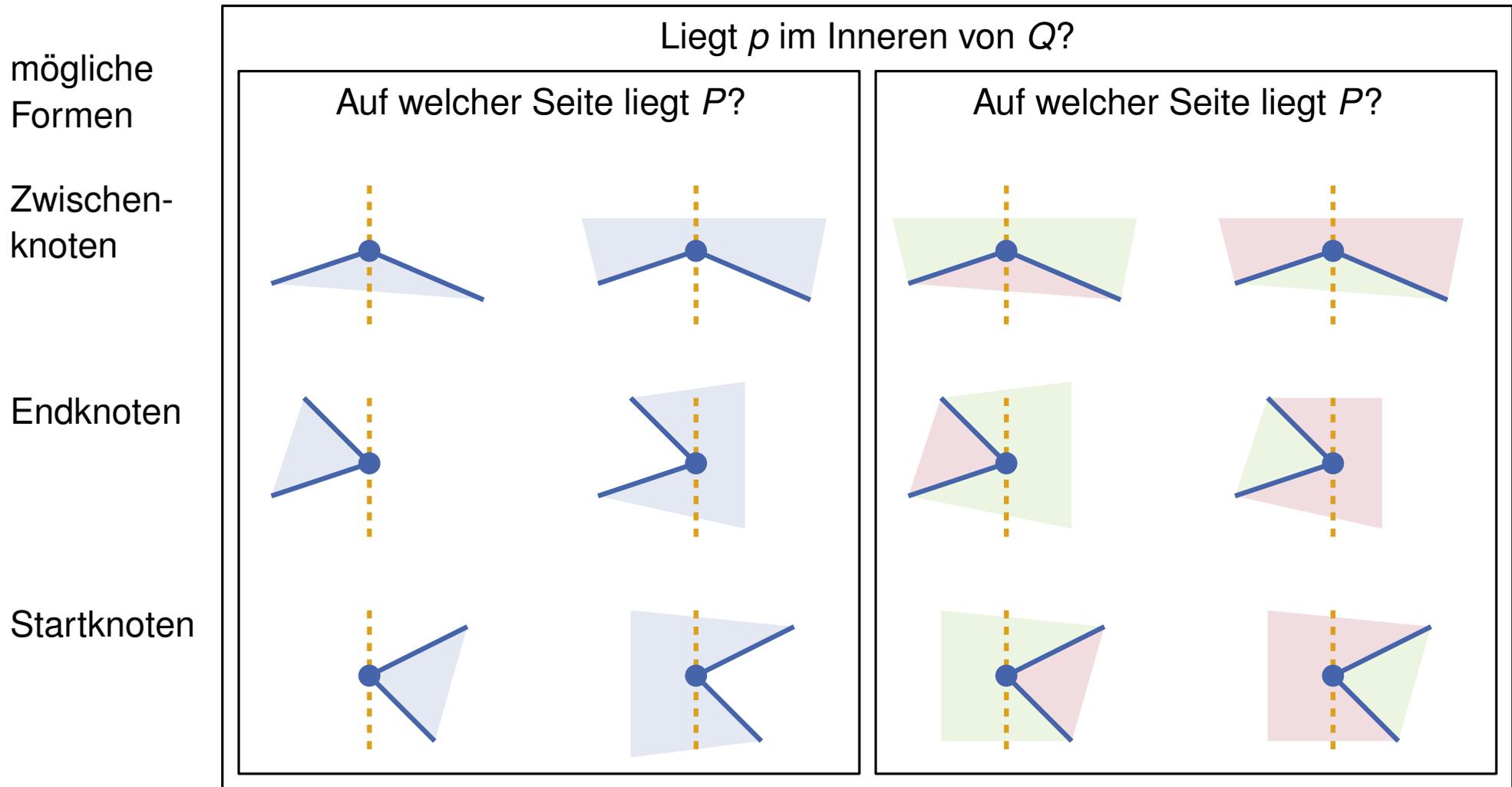
Reihenfolge der Strecken, die die Sweep-Line schneiden.



Zerlegung der Sweep-Line in Abschnitte, die in  $P \cap Q$ , nur in  $P$ , nur in  $Q$  oder weder in  $P$  noch in  $Q$  liegen.

# Problem 2(c) – Polygonknoten als Haltepunkt

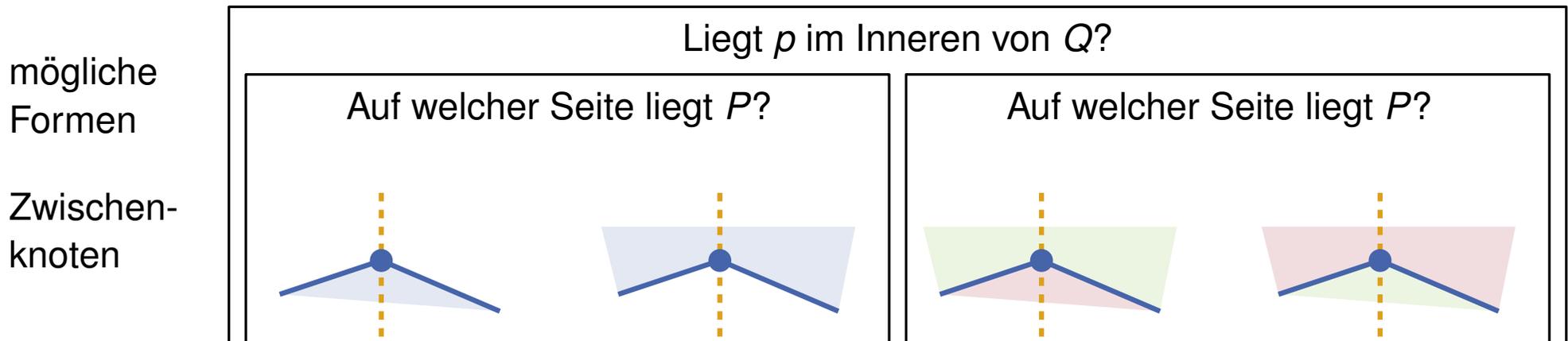
3. Wie ändert sich der Sweep-Line Zustand für die in Punkt 1. identifizierten Fälle.



Welcher Fall eintritt kann anhand des Sweep-Line Zustands festgestellt werden!

# Problem 2(c) – Polygonknoten als Haltepunkt

3. Wie ändert sich der Sweep-Line Zustand für die in Punkt 1. identifizierten Fälle.

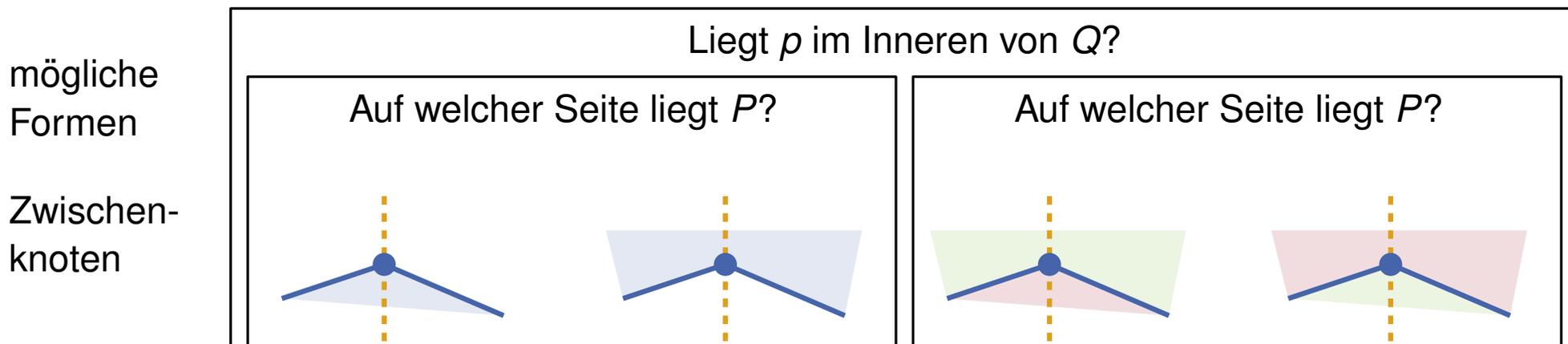


- Bei Zwischenknoten endet eine Kante  $e_1$  und eine neue Kante  $e_2$  startet.
- Im Sweep-Line Zustand wird  $e_1$  durch  $e_2$  ersetzt.
- Zerlegung der Sweep-Line bleibt wie sie ist.

Welcher Fall eintritt kann anhand des Sweep-Line Zustands festgestellt werden!

# Problem 2(c) – Polygonknoten als Haltepunkt

3. Wie ändert sich der Sweep-Line Zustand für die in Punkt 1. identifizierten Fälle.



- Bei Zwischenknoten endet eine Kante  $e_1$  und eine neue Kante  $e_2$  startet.
- Im Sweep-Line Zustand wird  $e_1$  durch  $e_2$  ersetzt.
- Zerlegung der Sweep-Line bleibt wie sie ist.

## Nicht vergessen:

Nach der Ersetzung muss  $e_2$  mit seinen neuen Nachbarn (bezüglich der Reihenfolge im Sweep-Line Zustand) auf Schnitt geprüft werden.

Welcher Fall eintritt kann anhand des Sweep-Line Zustands festgestellt werden!

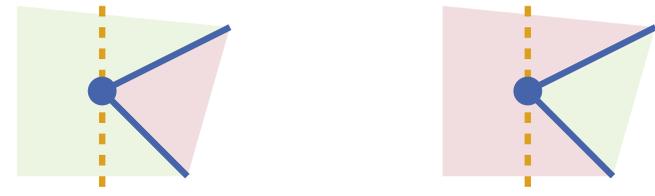
# Problem 2(c) – Polygonknoten als Haltepunkt

- Bei Endknoten enden zwei Kanten  $e_1$  und  $e_2$ , sie werden aus dem Sweep-Line Zustand gelöscht.
- Für die Zerlegung der Sweep-Line gilt: oberhalb von  $e_1$  ist der gleiche Bereich wie unterhalb von  $e_2$ . → Die beiden Abschnitte fallen zusammen.

Endknoten



Startknoten



Welcher Fall eintritt kann anhand des Sweep-Line Zustands festgestellt werden!

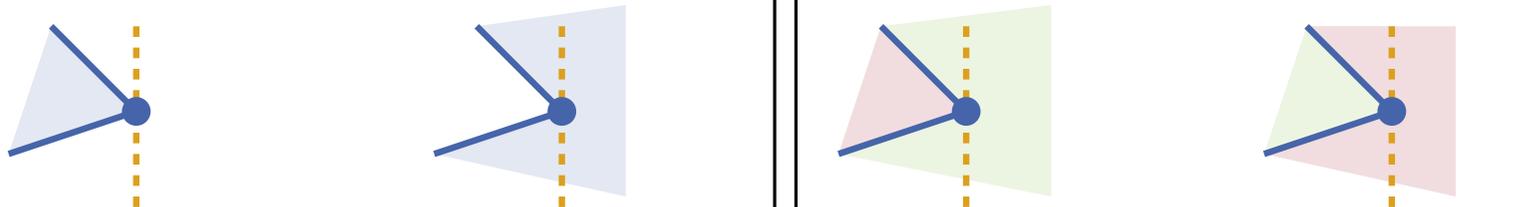
# Problem 2(c) – Polygonknoten als Haltepunkt

- Bei Endknoten enden zwei Kanten  $e_1$  und  $e_2$ , sie werden aus dem Sweep-Line Zustand gelöscht.
- Für die Zerlegung der Sweep-Line gilt: oberhalb von  $e_1$  ist der gleiche Bereich wie unterhalb von  $e_2$ . → Die beiden Abschnitte fallen zusammen.

## Nicht vergessen:

Die Strecken oberhalb von  $e_1$  und unterhalb von  $e_2$  auf Schnitt testen.

Endknoten



Startknoten



Welcher Fall eintritt kann anhand des Sweep-Line Zustands festgestellt werden!

# Problem 2(c) – Polygonknoten als Haltepunkt

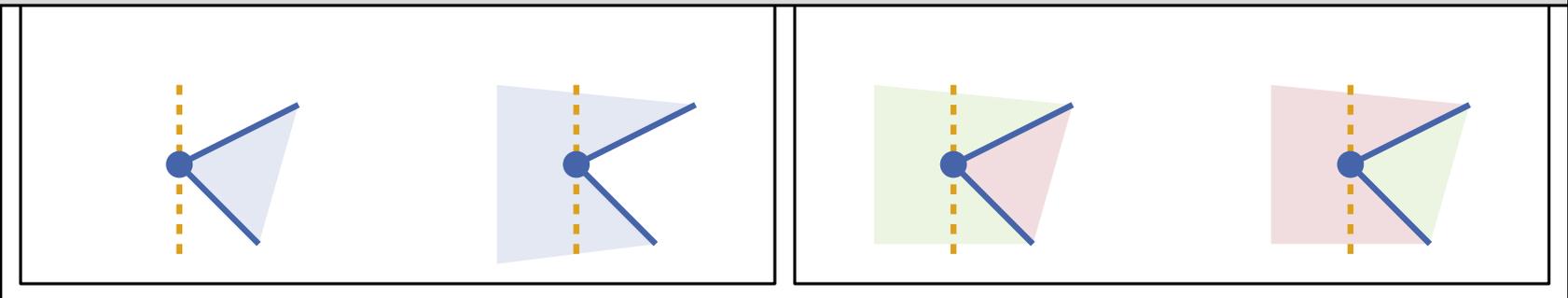
3. Wie ändert sich der Sweep-Line Zustand für die in Punkt 1. identifizierten Fälle.

mögliche

Liegt  $p$  im Inneren von  $Q$ ?

- Bei Startknoten starten zwei neue Kanten  $e_1$  und  $e_2$ .
- Füge sie an der richtigen Stelle im Sweep-Line Zustand ein.
- Der Abschnitt in den sie eingefügt werden wird in zwei Teile gespalten, in den oberhalb von  $e_1$  und den unterhalb von  $e_2$ .
- Zwischen  $e_1$  und  $e_2$  entsteht ein neuer Abschnitt.

Startknoten



Welcher Fall eintritt kann anhand des Sweep-Line Zustands festgestellt werden!

# Problem 2(c) – Polygonknoten als Haltepunkt

3. Wie ändert sich der Sweep-Line Zustand für die in Punkt 1. identifizierten Fälle.

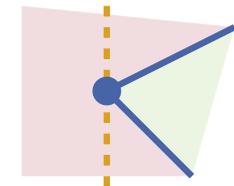
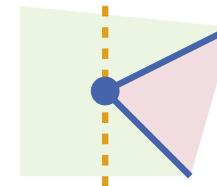
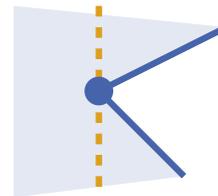
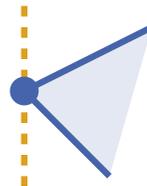
mögliche

Liegt  $p$  im Inneren von  $Q$ ?

- Bei Startknoten starten zwei neue Kanten  $e_1$  und  $e_2$ .
- Füge sie an der richtigen Stelle im Sweep-Line Zustand ein.
- Der Abschnitt in den sie eingefügt werden wird in zwei Teile gespalten, in den oberhalb von  $e_1$  und den unterhalb von  $e_2$ .
- Zwischen  $e_1$  und  $e_2$  entsteht ein neuer Abschnitt.

**Nicht vergessen: ...**

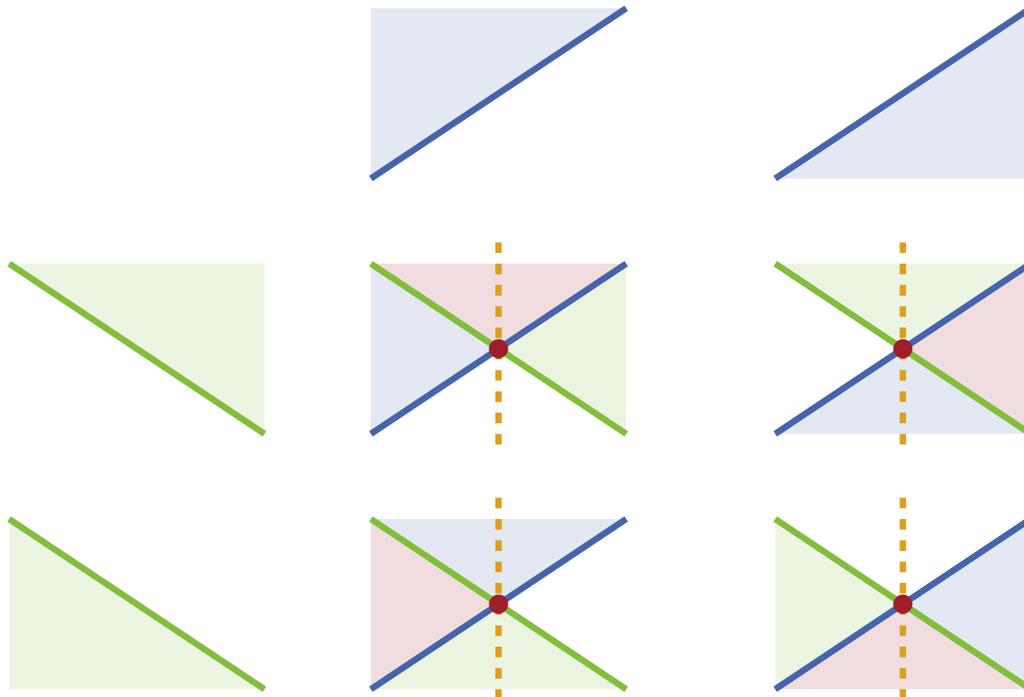
Startknoten



Welcher Fall eintritt kann anhand des Sweep-Line Zustands festgestellt werden!

# Problem 2(c) – Schnittpunkt als Haltepunkt

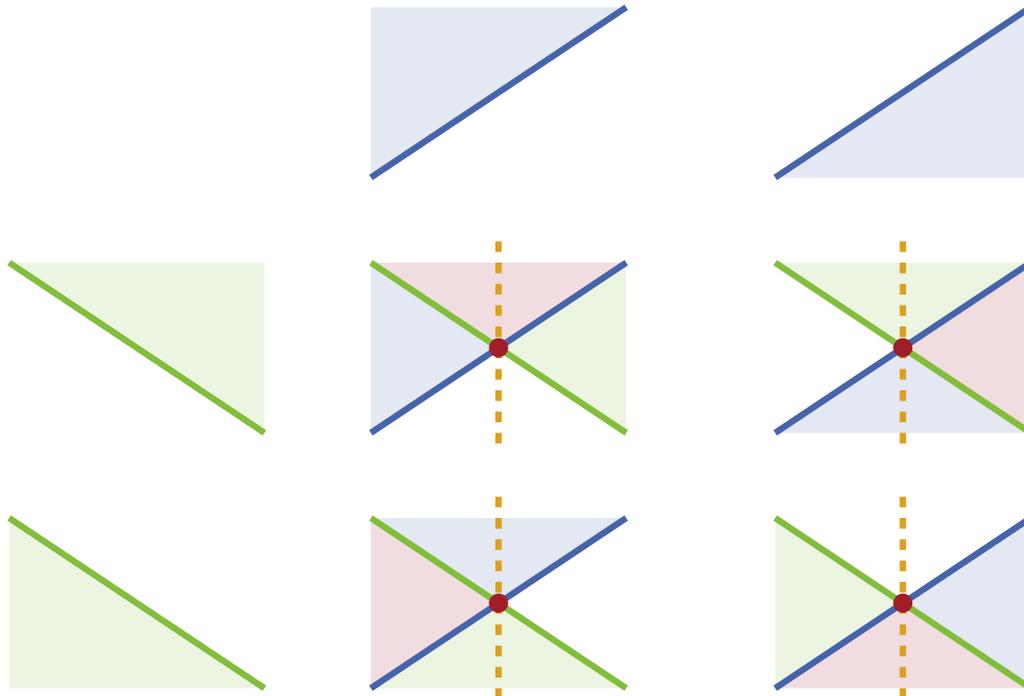
3. Wie ändert sich der Sweep-Line Zustand für die in Punkt 1. identifizierten Fälle.



Seien  $e_P$  und  $e_Q$  die sich kreuzenden Polygonkanten aus  $P$  bzw.  $Q$ .

# Problem 2(c) – Schnittpunkt als Haltepunkt

3. Wie ändert sich der Sweep-Line Zustand für die in Punkt 1. identifizierten Fälle.



Seien  $e_P$  und  $e_Q$  die sich kreuzenden Polygonkanten aus  $P$  bzw.  $Q$ .

- Die Reihenfolge von  $e_P$  und  $e_Q$  im Sweep-Line Zustand wird vertauscht.
- Zu welchem Bereich der Abschnitt der Sweep-Line zwischen  $e_P$  und  $e_Q$  hinterher gehört hängt davon ab, zu welchem er vorher gehört hat.

## Nicht vergessen:

$e_P$  mit der Strecke oberhalb und  $e_Q$  mit der Strecke unterhalb auf Schnitt testen.

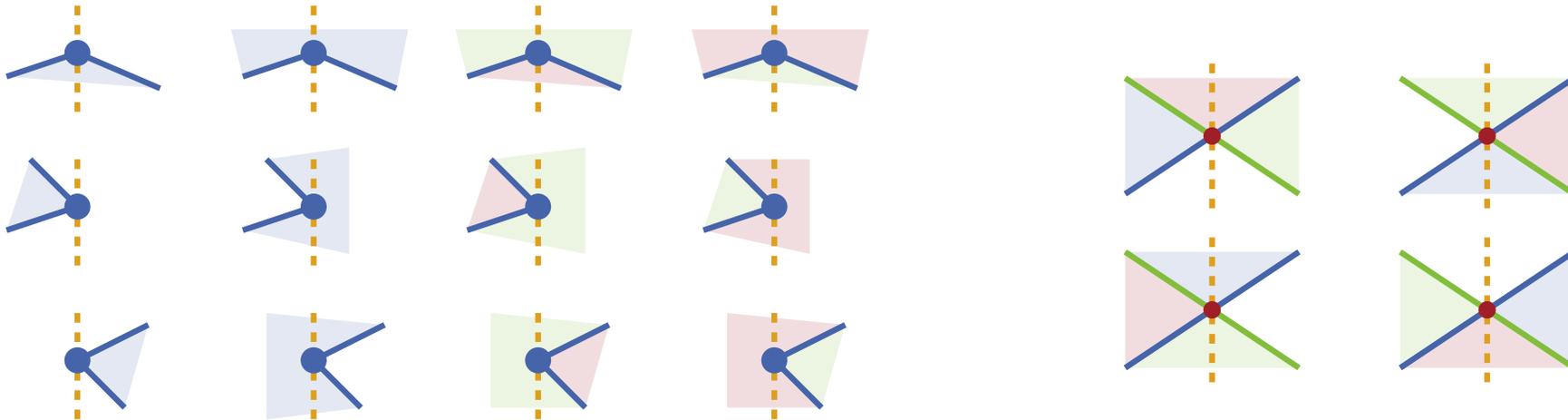
# Problem 2(c) – Zusammenfassung

4. Müssen Sie zusätzliche Informationen mitführen, um am Ende den Schnitt  $P \cap Q$  als Menge von Polygonen auszugeben?

# Problem 2(c) – Zusammenfassung

4. Müssen Sie zusätzliche Informationen mitführen, um am Ende den Schnitt  $P \cap Q$  als Menge von Polygonen auszugeben?

- Wir wissen an jedem Haltepunkt welcher Fall eintritt.

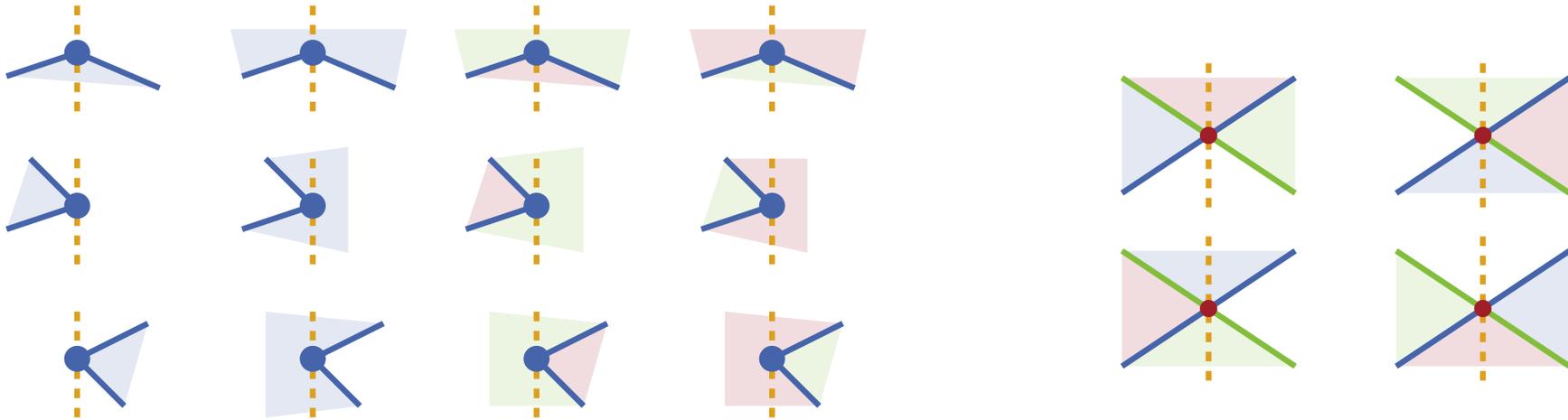


Wie kann man daraus die Polygone berechnen, deren Inneres gerade  $P \cap Q$  ist?

# Problem 2(c) – Zusammenfassung

4. Müssen Sie zusätzliche Informationen mitführen, um am Ende den Schnitt  $P \cap Q$  als Menge von Polygonen auszugeben?

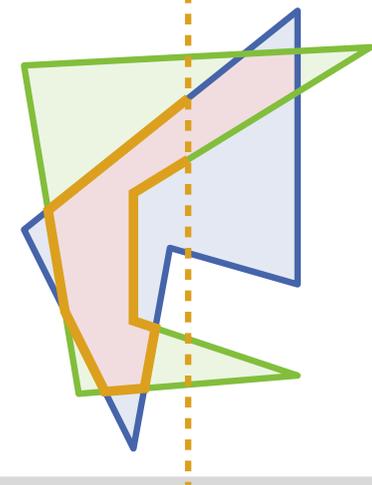
- Wir wissen an jedem Haltepunkt welcher Fall eintritt.



Wie kann man daraus die Polygone berechnen, deren Inneres gerade  $P \cap Q$  ist?

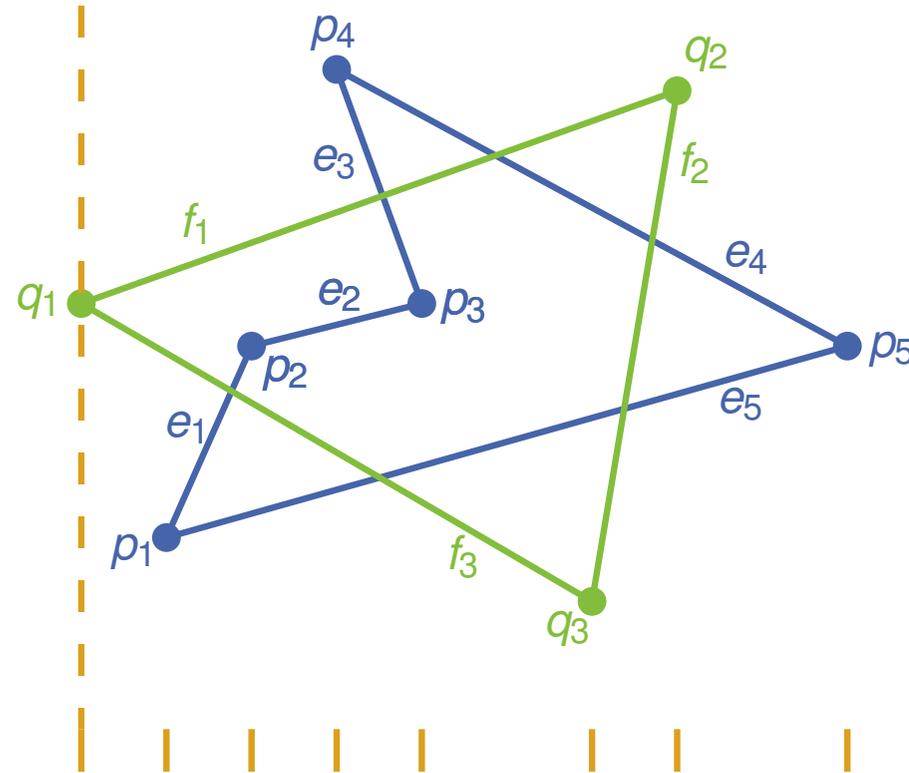
## Idee:

- Speichere für jede Kante, die die aktuelle Sweep-Line kreuzt und  $P \cap Q$  begrenzt den Polygonzug links von der Sweep-Line, der  $P \cap Q$  begrenzt.
- An Haltepunkten werden Polygonzüge fortgesetzt, verschmolzen oder entstehen neu.



# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht  
explizit angegeben)



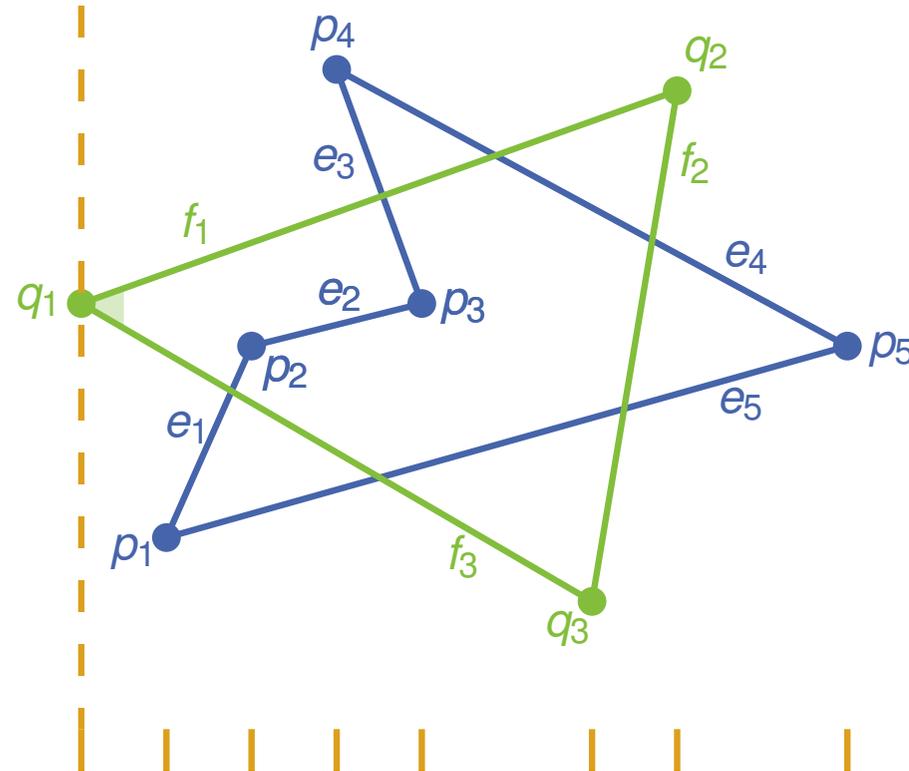
Event-Point Schedule →

bisher berechnete Polygonzüge:

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht  
explizit angegeben)

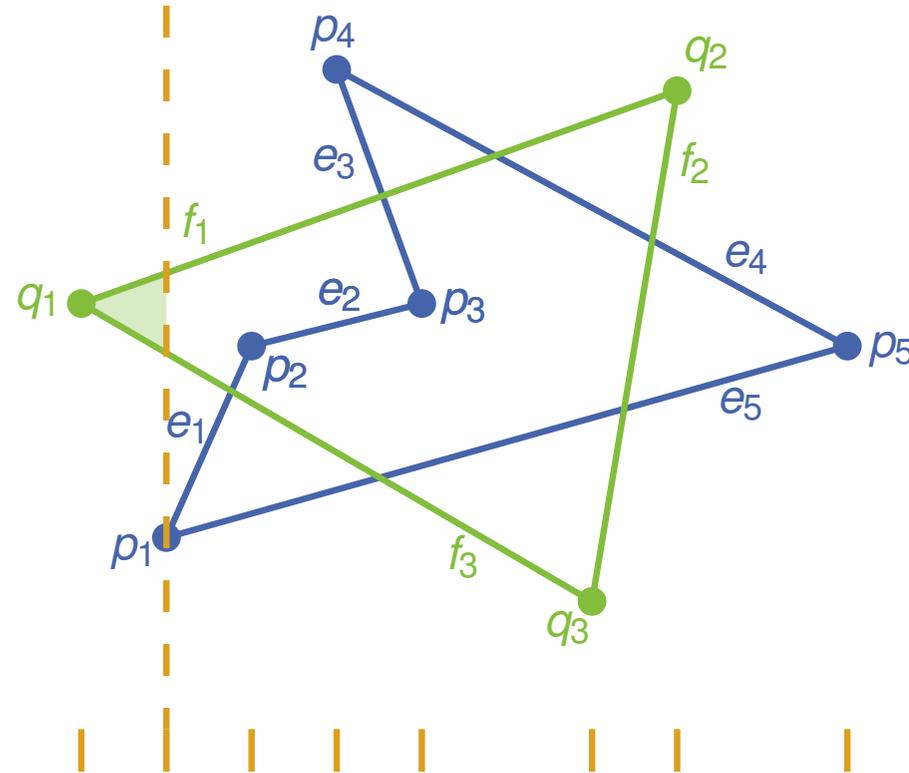
Event-Point Schedule →



bisher berechnete Polygonzüge:

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht  
explizit angegeben)



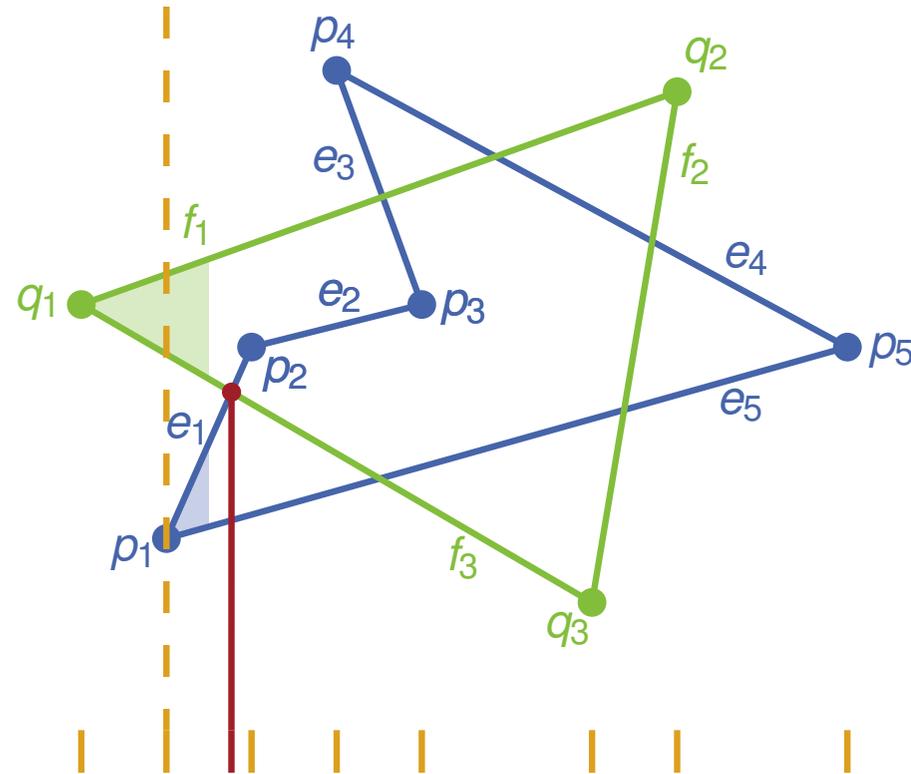
Event-Point Schedule →

bisher berechnete Polygonzüge:

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht  
explizit angegeben)

Event-Point Schedule →

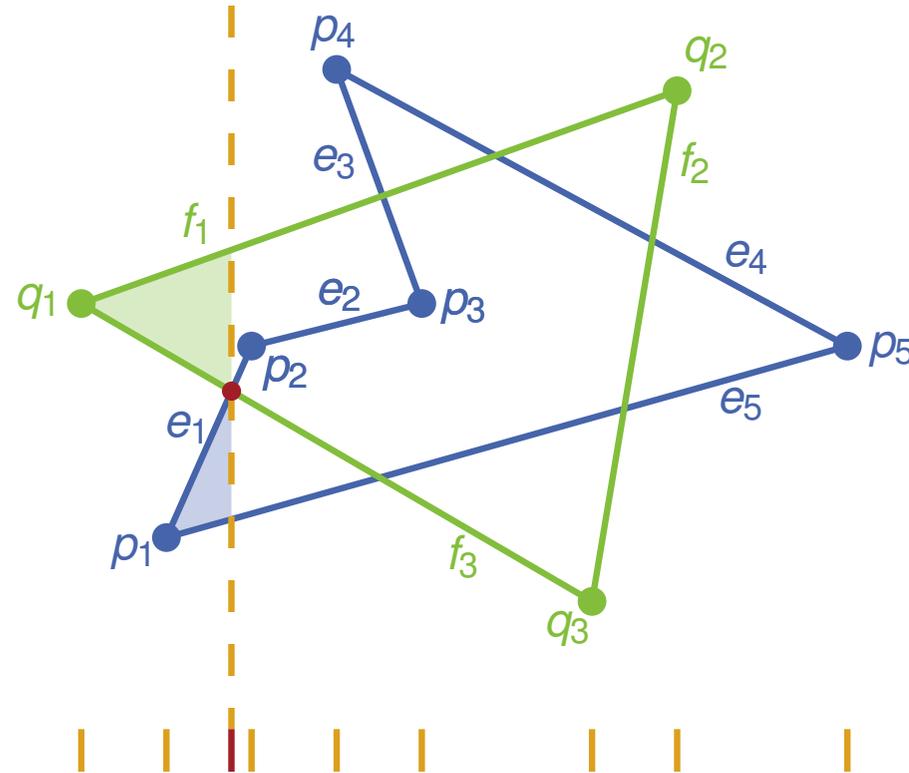


bisher berechnete Polygonzüge:

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht  
explizit angegeben)

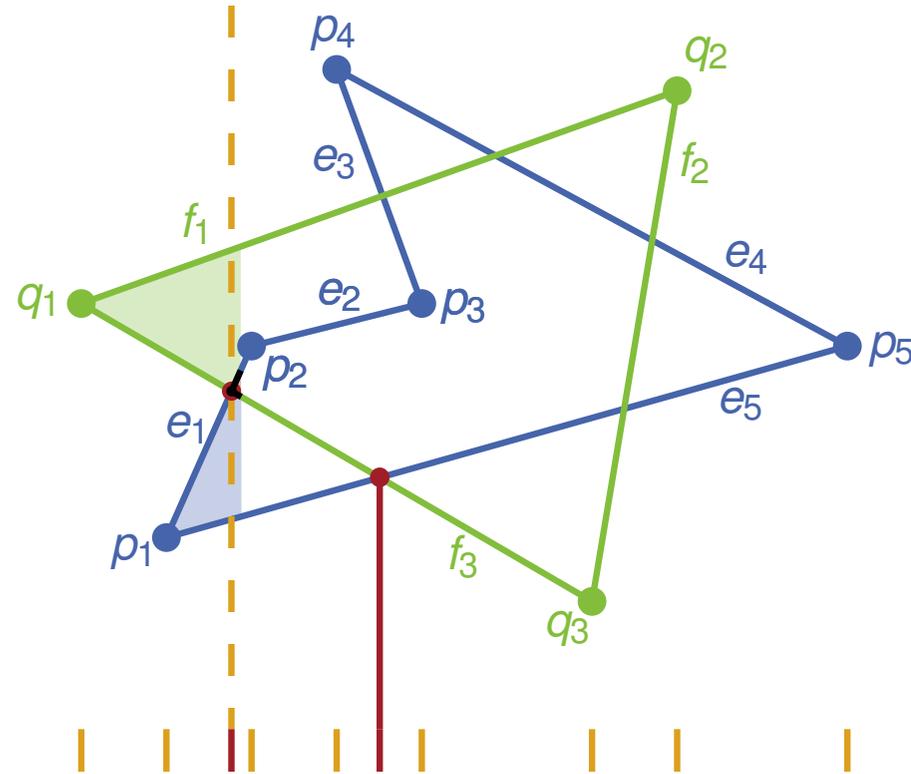
Event-Point Schedule →



bisher berechnete Polygonzüge:

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



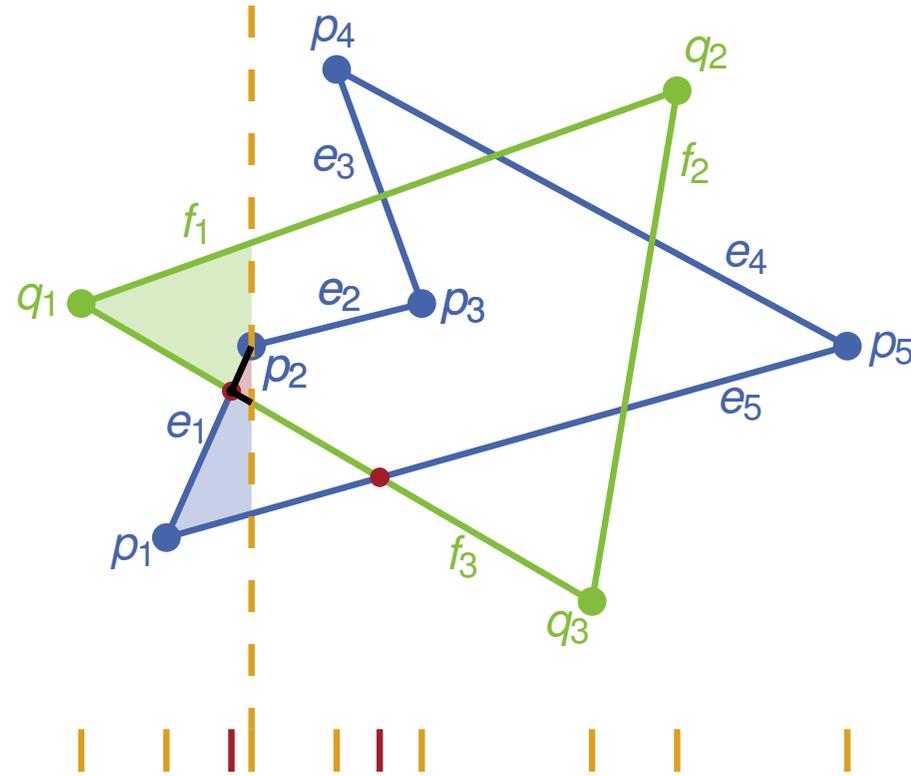
Event-Point Schedule →

bisher berechnete Polygonzüge:

$$(f_3) \longleftrightarrow f_3 \times e_1 \longleftrightarrow (e_1)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



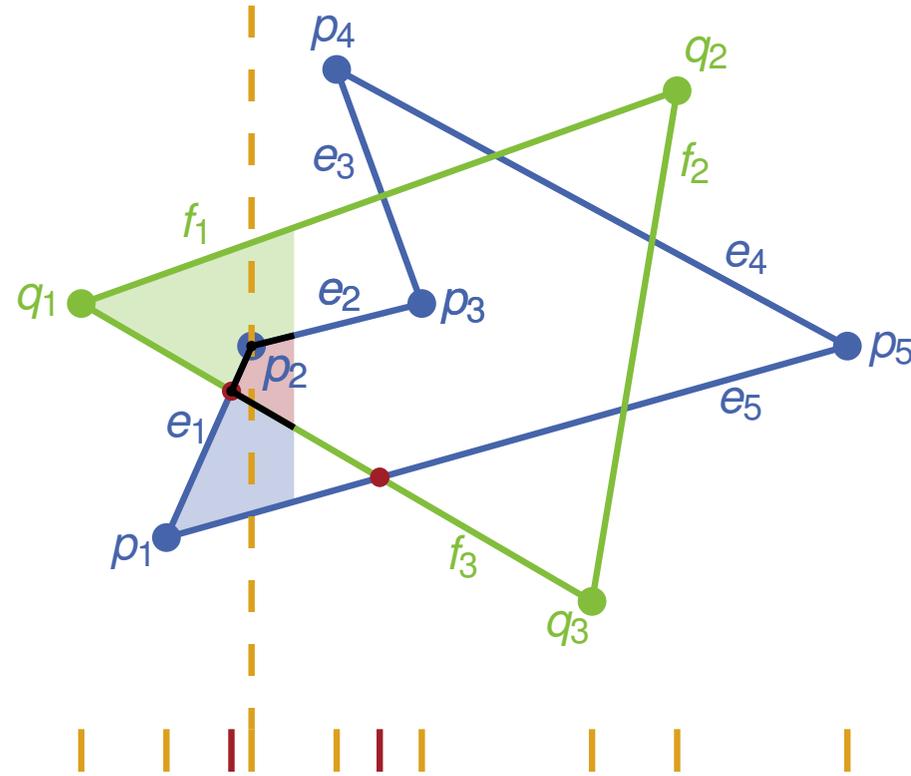
Event-Point Schedule →

bisher berechnete Polygonzüge:

$$(f_3) \leftrightarrow f_3 \times e_1 \leftrightarrow (e_1)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



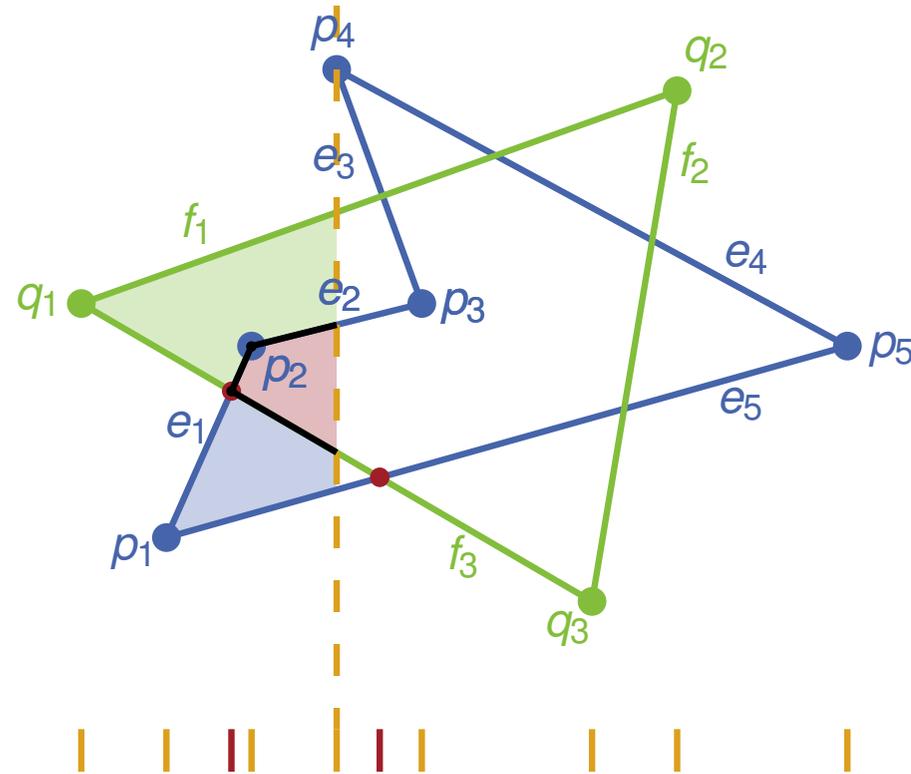
Event-Point Schedule →

bisher berechnete Polygonzüge:

$$(f_3) \leftrightarrow f_3 \times e_1 \leftrightarrow p_2 \leftrightarrow (e_2)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



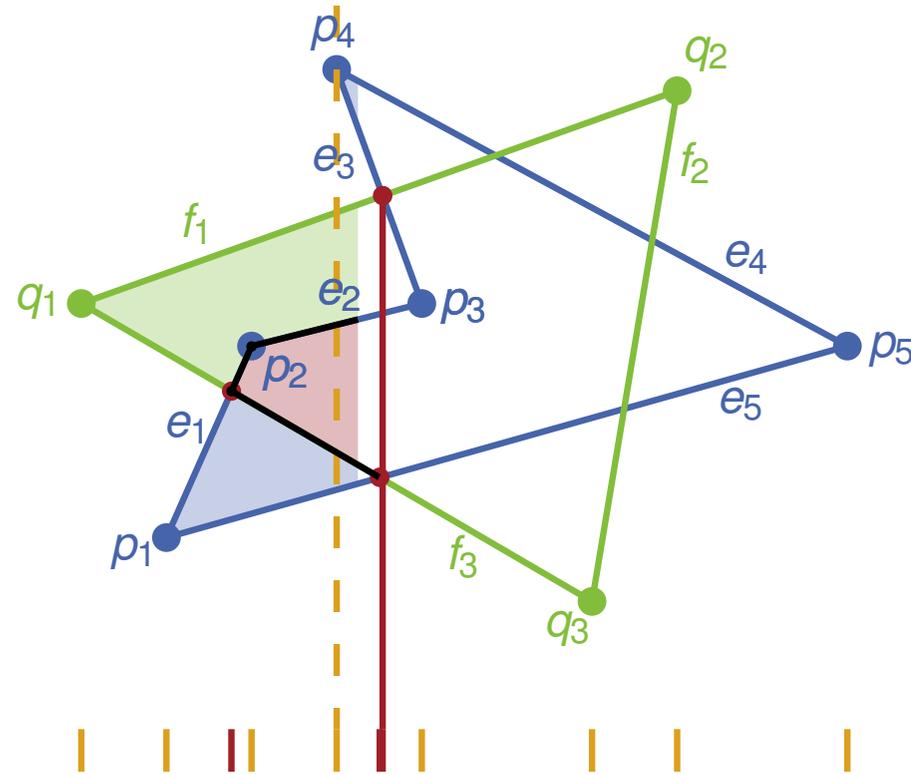
Event-Point Schedule →

bisher berechnete Polygonzüge:

$$(f_3) \leftrightarrow f_3 \times e_1 \leftrightarrow p_2 \leftrightarrow (e_2)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



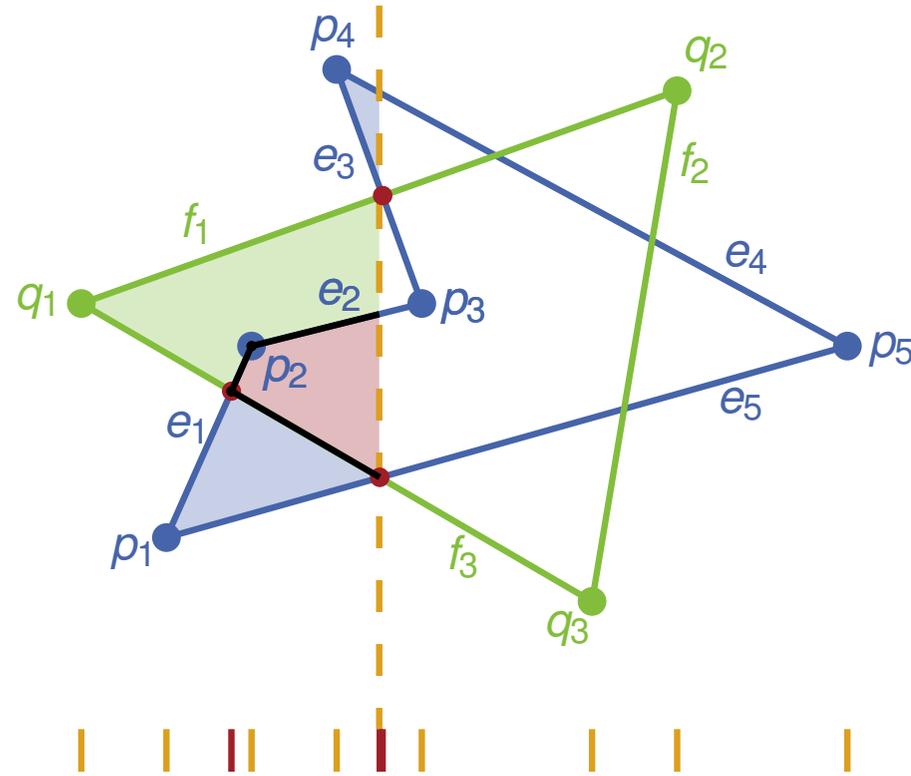
Event-Point Schedule →

bisher berechnete Polygonzüge:

$$(f_3) \leftrightarrow f_3 \times e_1 \leftrightarrow p_2 \leftrightarrow (e_2)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



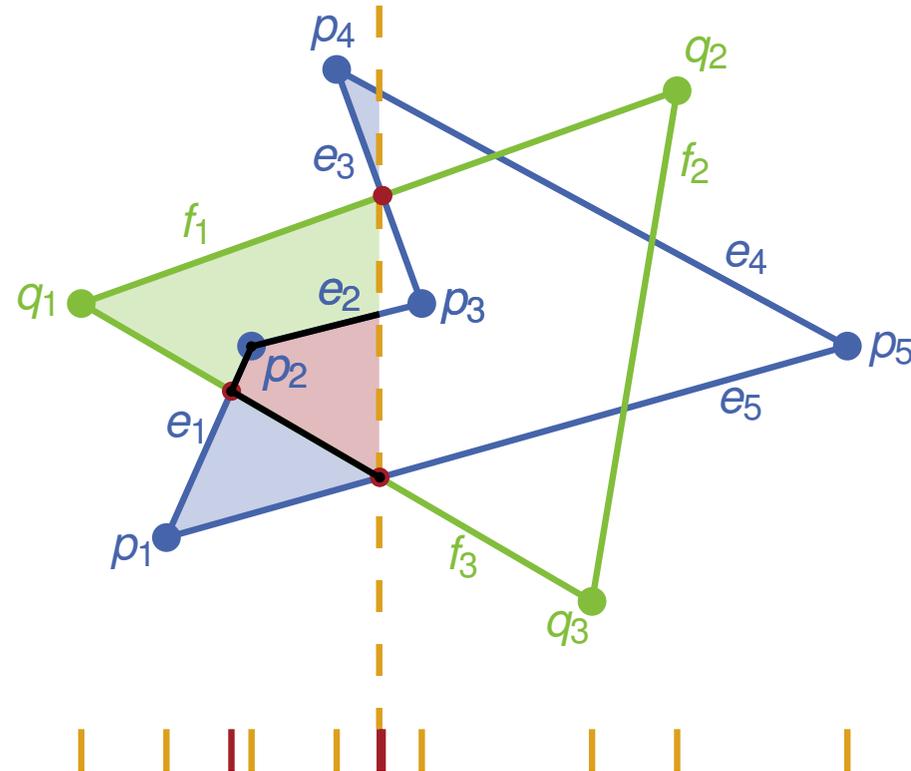
Event-Point Schedule →

bisher berechnete Polygonzüge:

$$(f_3) \leftrightarrow f_3 \times e_1 \leftrightarrow p_2 \leftrightarrow (e_2)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



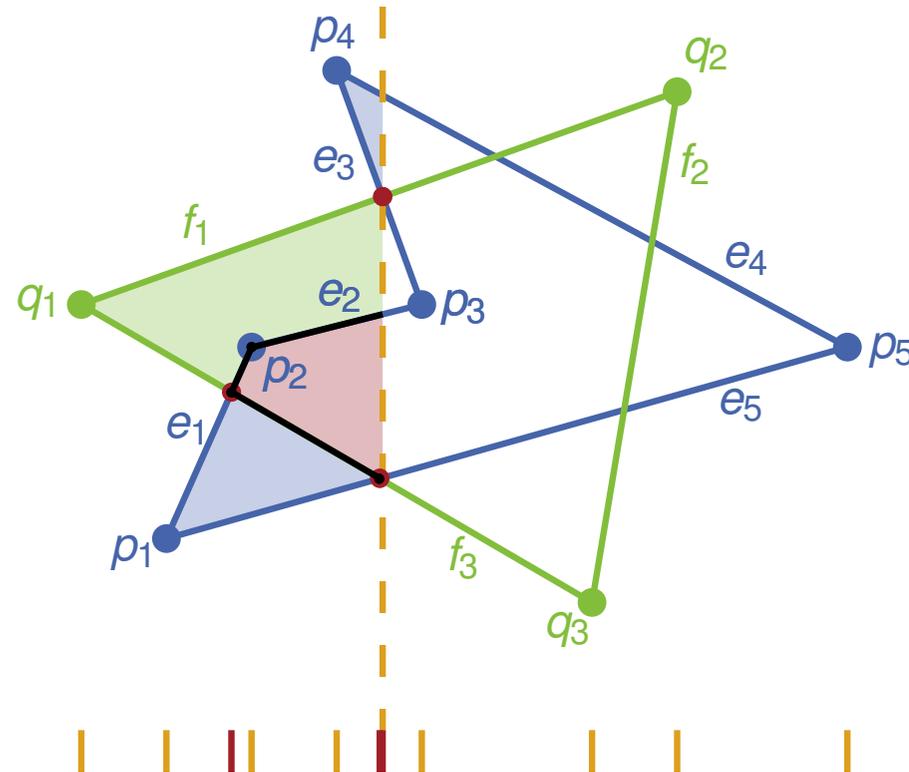
Event-Point Schedule →

bisher berechnete Polygonzüge:

$$(e_5) \leftrightarrow e_5 \times f_3 \leftrightarrow f_3 \times e_1 \leftrightarrow p_2 \leftrightarrow (e_2)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



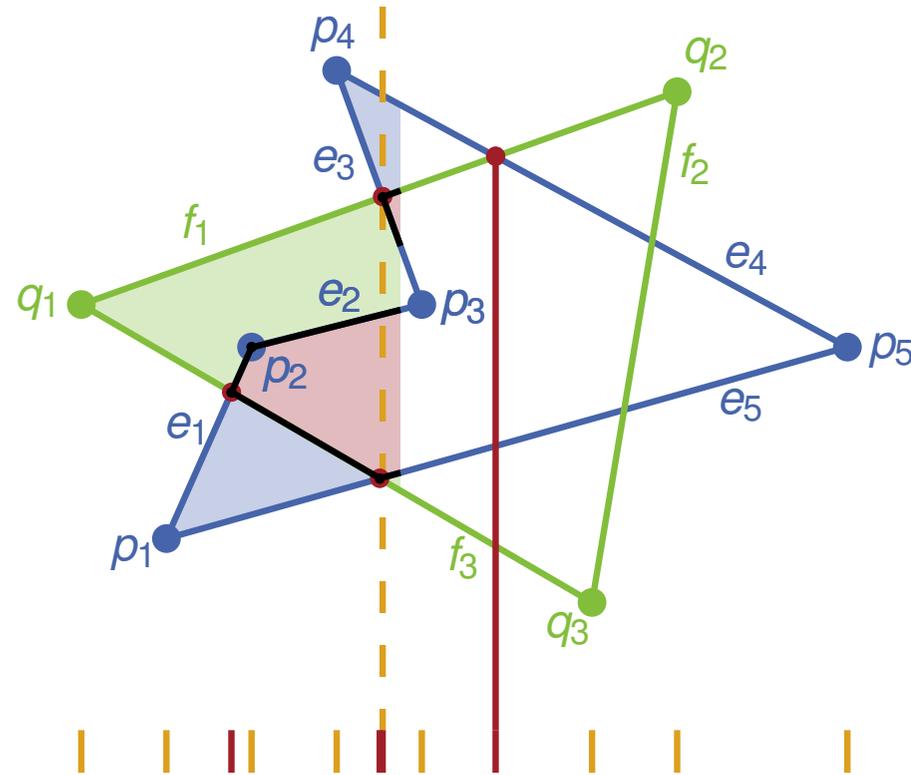
Event-Point Schedule →

bisher berechnete Polygonzüge:

$$(e_5) \longleftrightarrow e_5 \times f_3 \longleftrightarrow f_3 \times e_1 \longleftrightarrow p_2 \longleftrightarrow (e_2)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

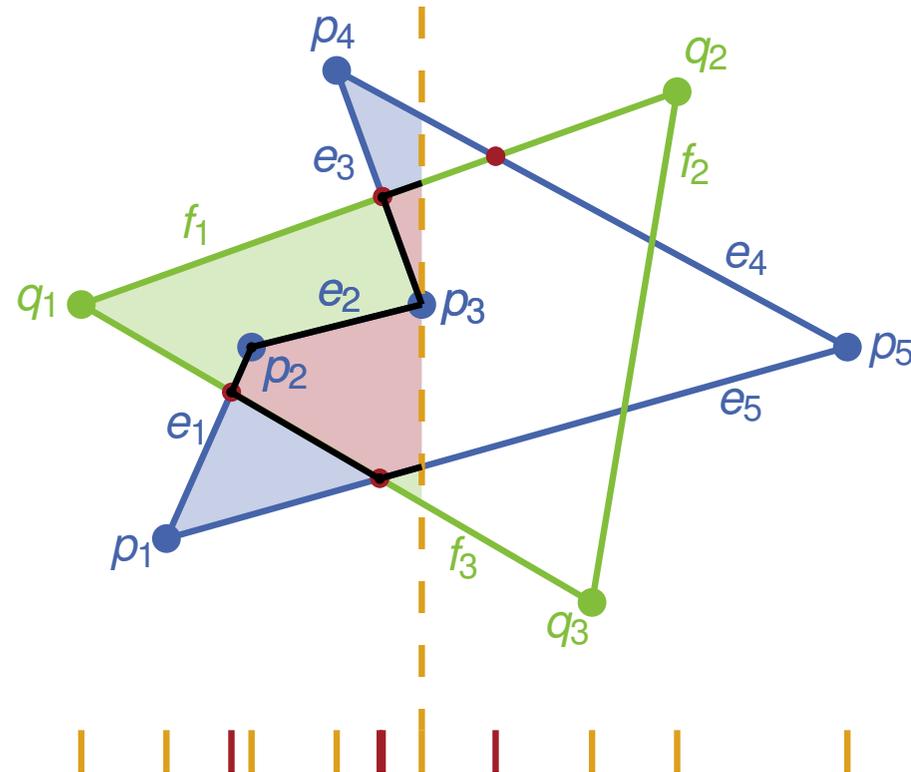
bisher berechnete Polygonzüge:

$$(e_5) \leftrightarrow e_5 \times f_3 \leftrightarrow f_3 \times e_1 \leftrightarrow p_2 \leftrightarrow (e_2)$$

$$(f_1) \leftrightarrow f_1 \times e_3 \leftrightarrow (e_3)$$

# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

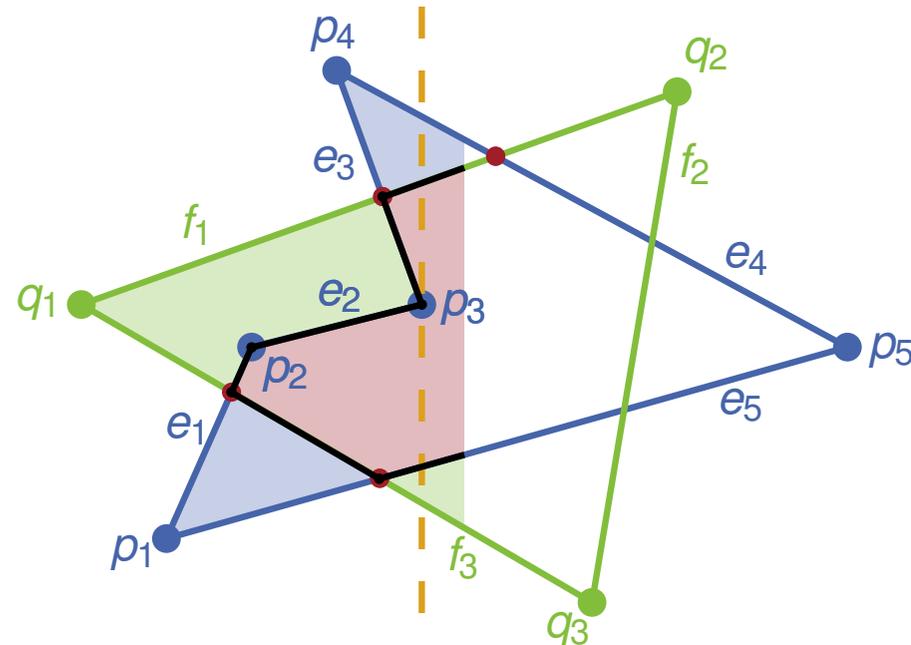
bisher berechnete Polygonzüge:

$$(e_5) \leftrightarrow e_5 \times f_3 \leftrightarrow f_3 \times e_1 \leftrightarrow p_2 \leftrightarrow (e_2)$$

$$(f_1) \leftrightarrow f_1 \times e_3 \leftrightarrow (e_3)$$

# Problem 2 (c) – Beispiel

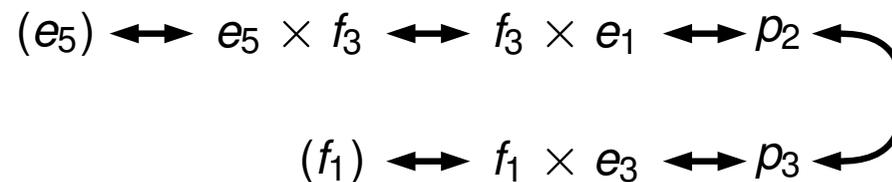
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

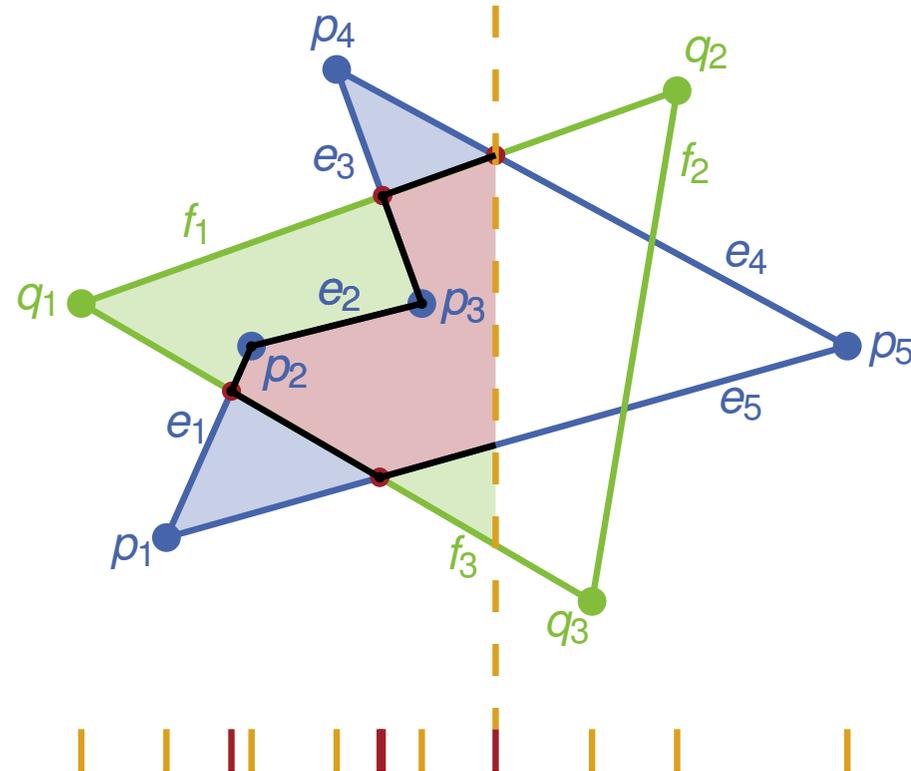


bisher berechnete Polygonzüge:



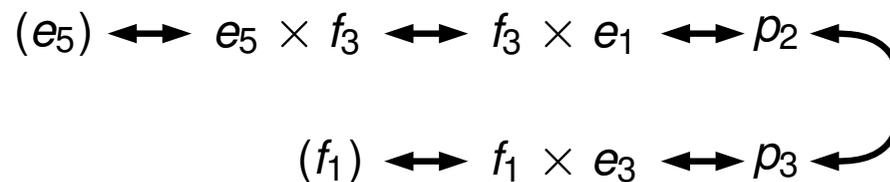
# Problem 2 (c) – Beispiel

(Sweep-Line Zustand nicht explizit angegeben)



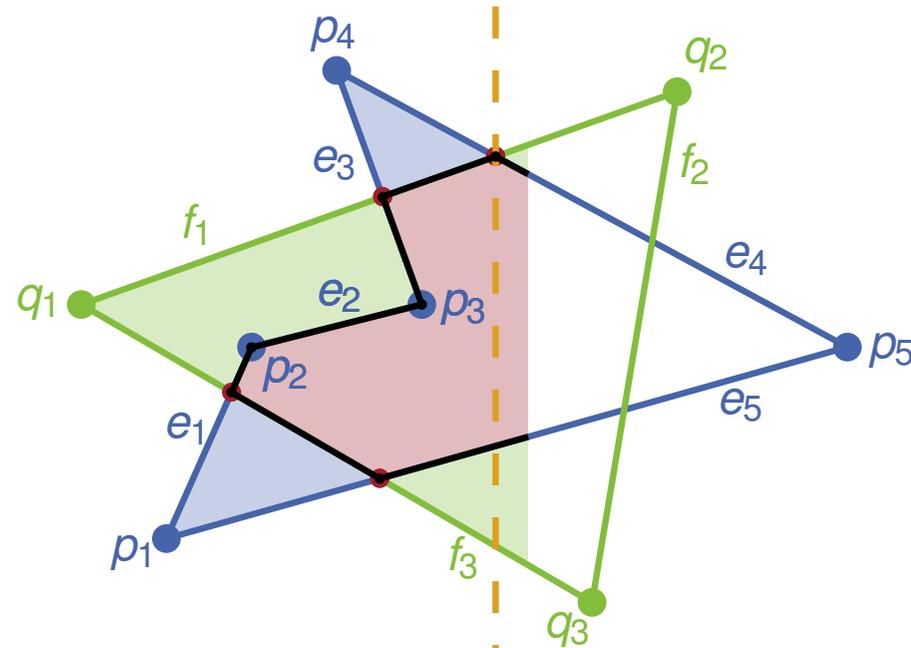
Event-Point Schedule →

bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

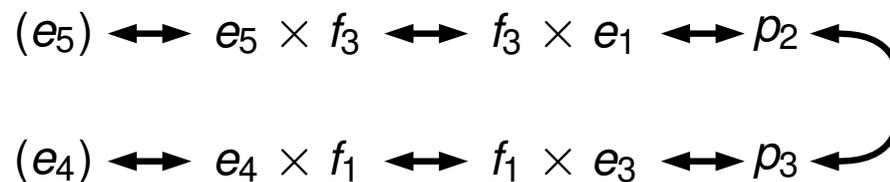
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

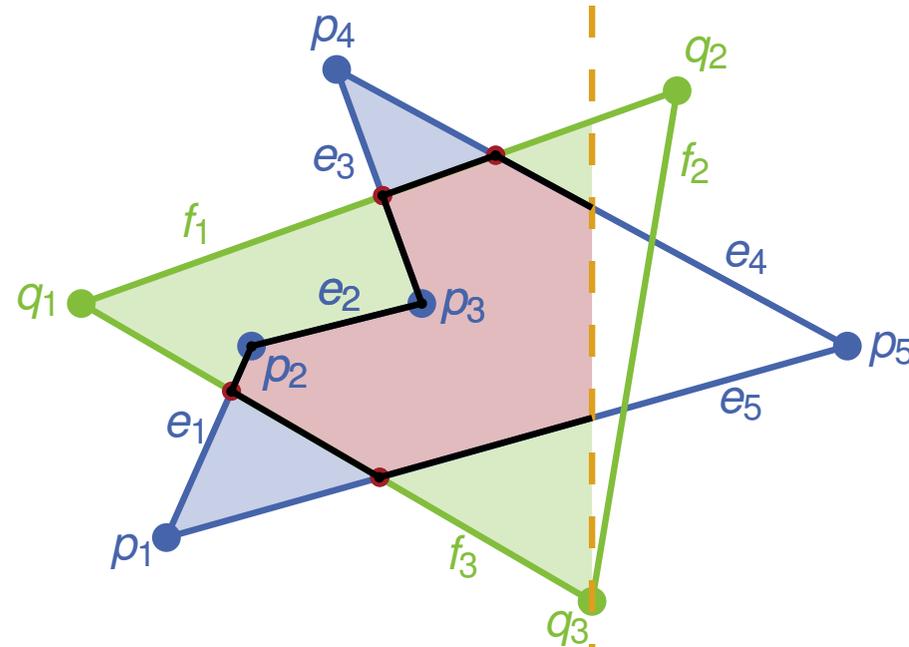


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

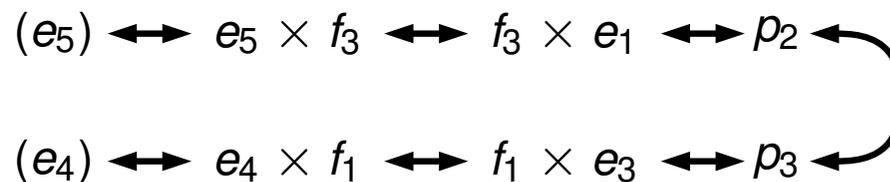
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

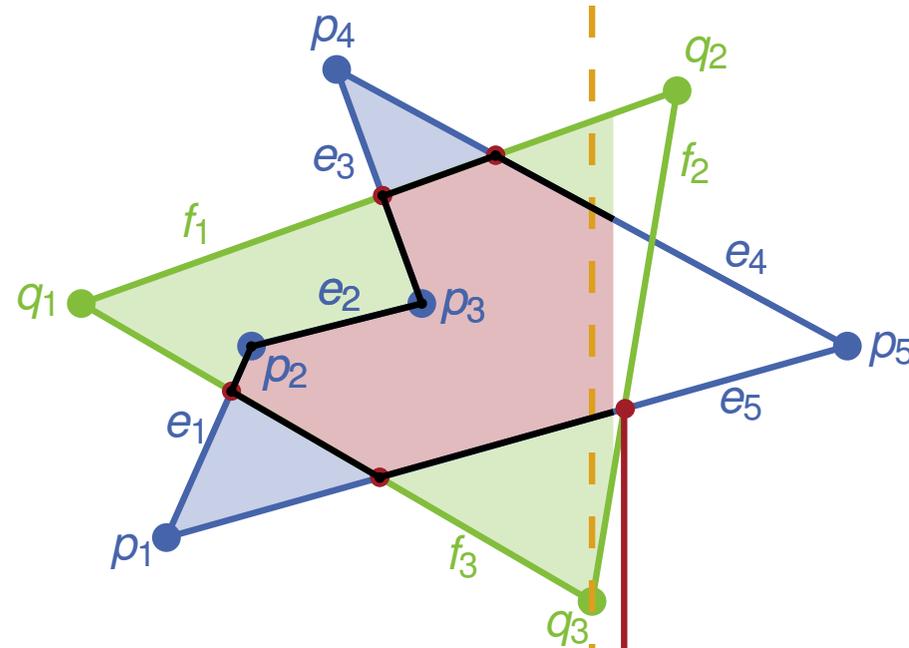


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

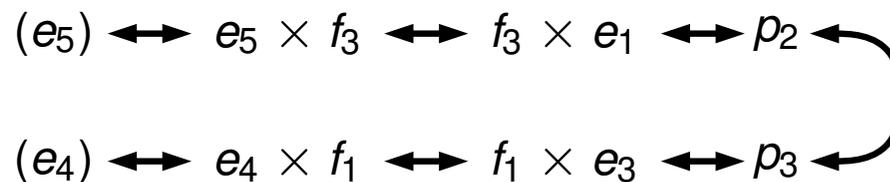
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

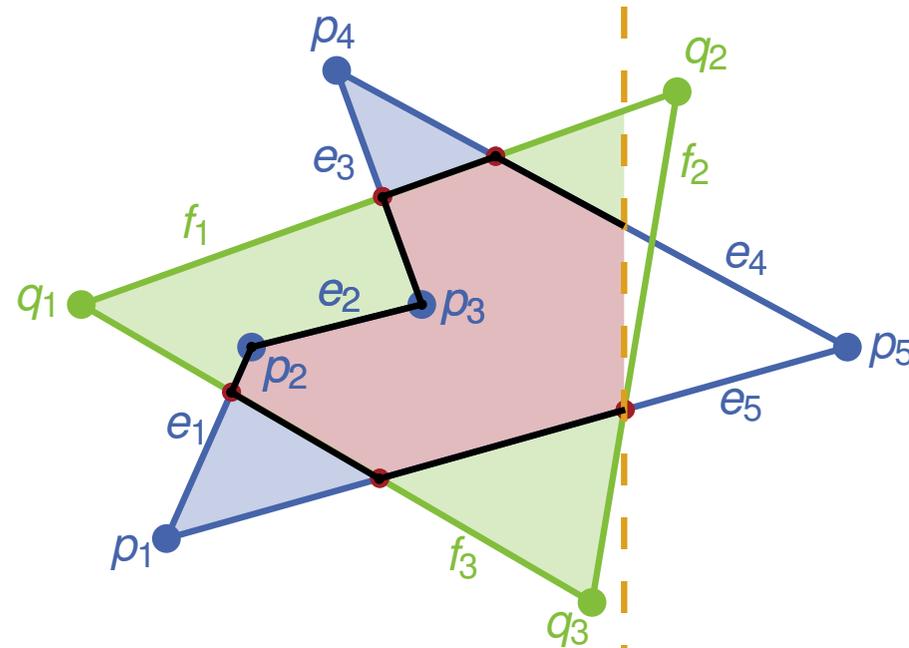


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

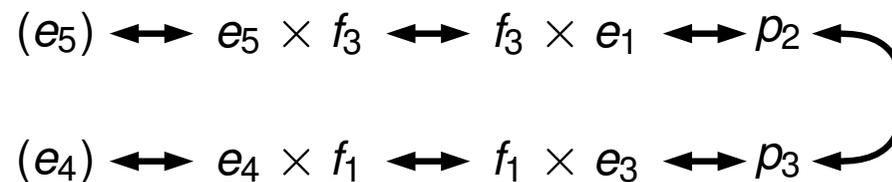
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

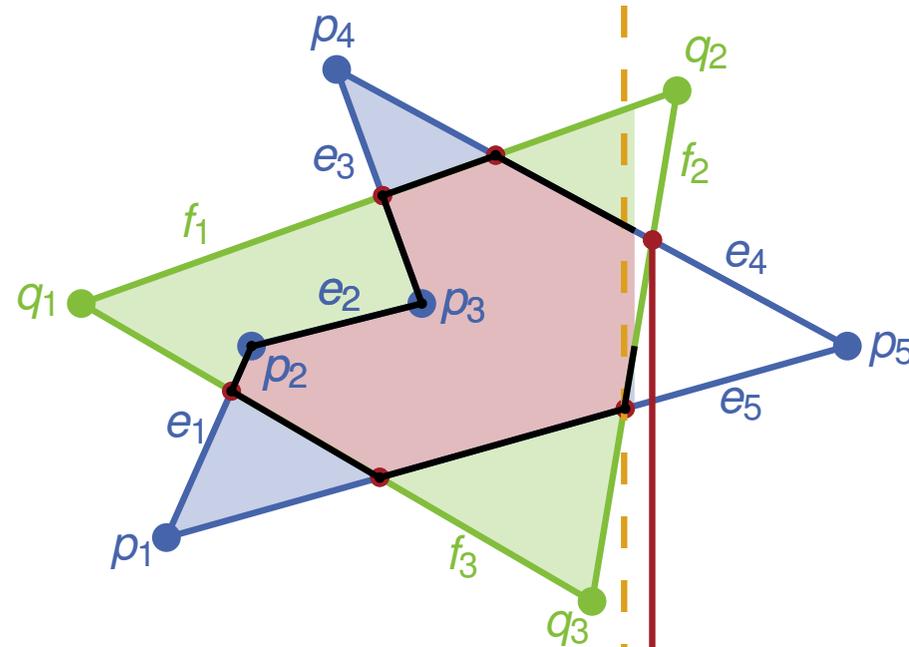


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

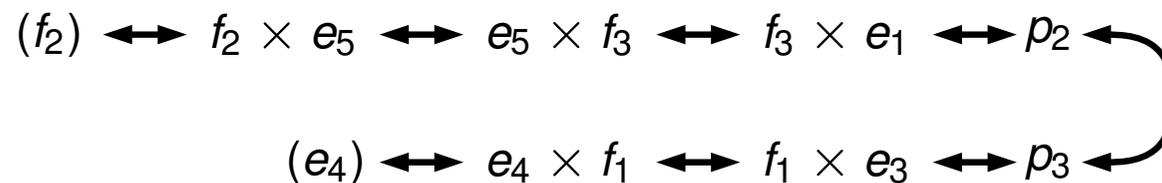
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

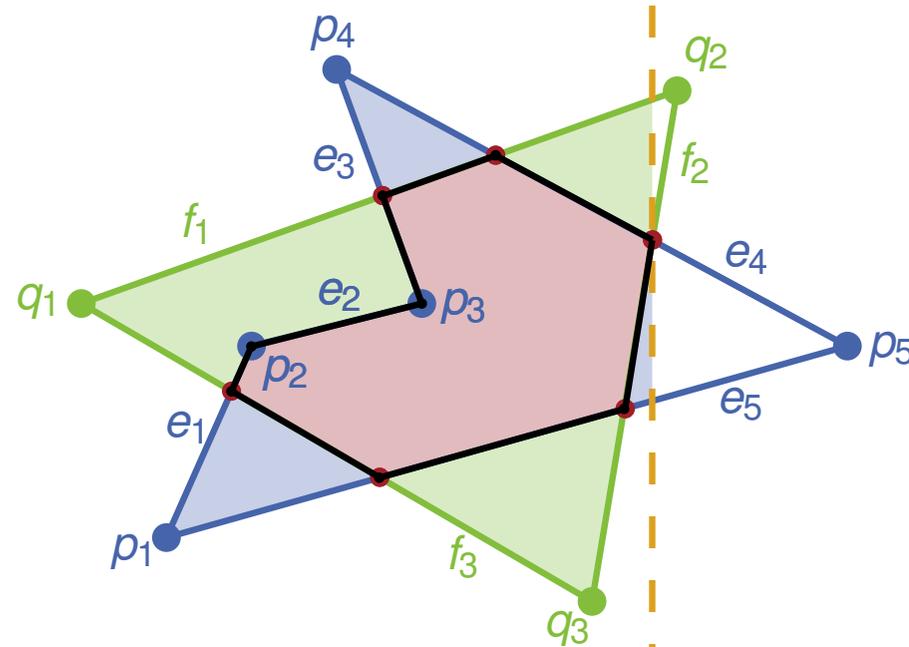


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

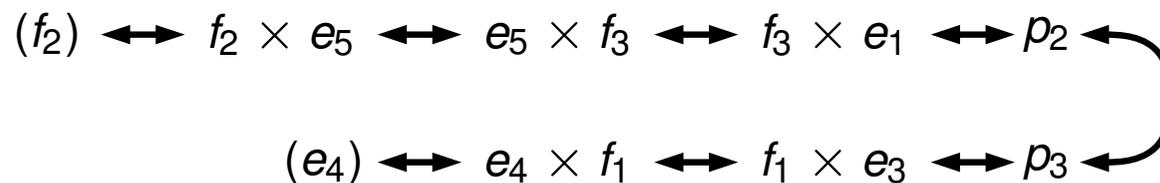
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

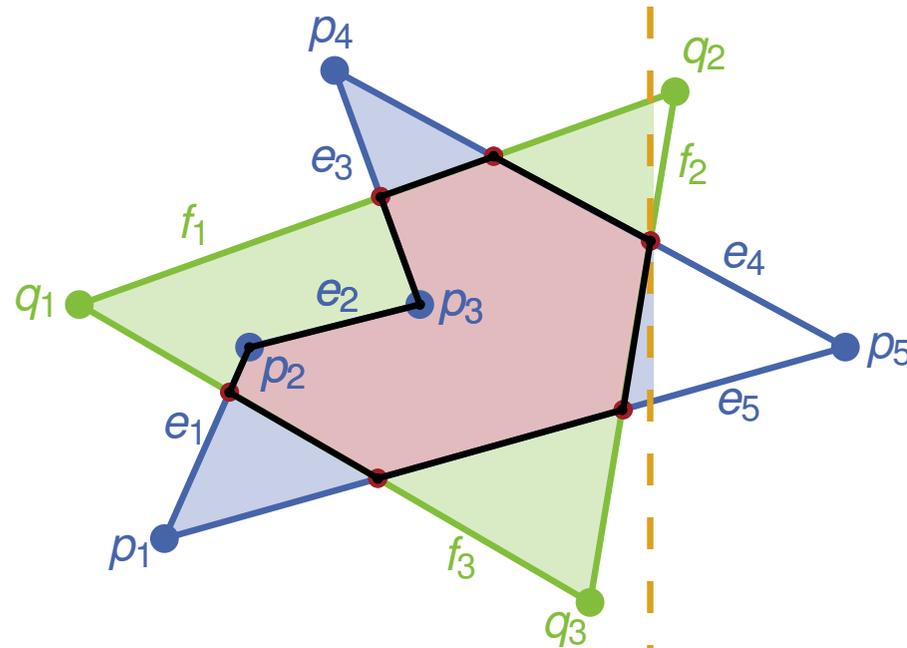


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

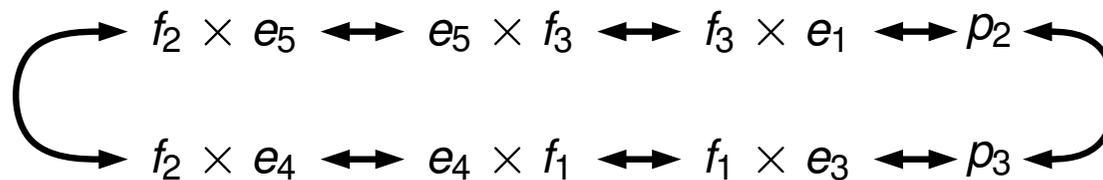
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

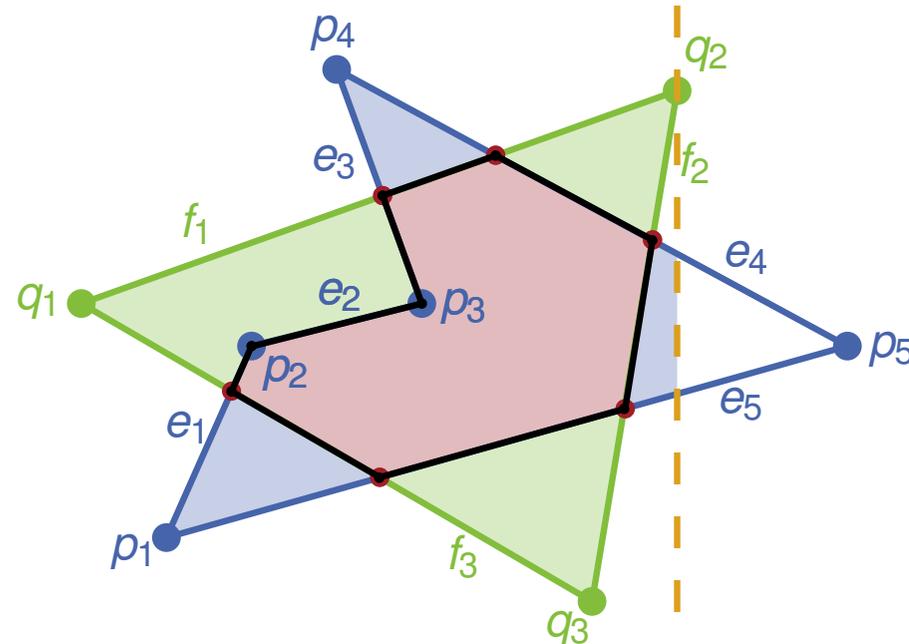


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

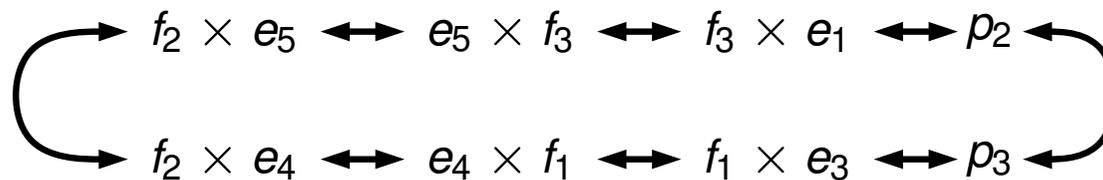
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

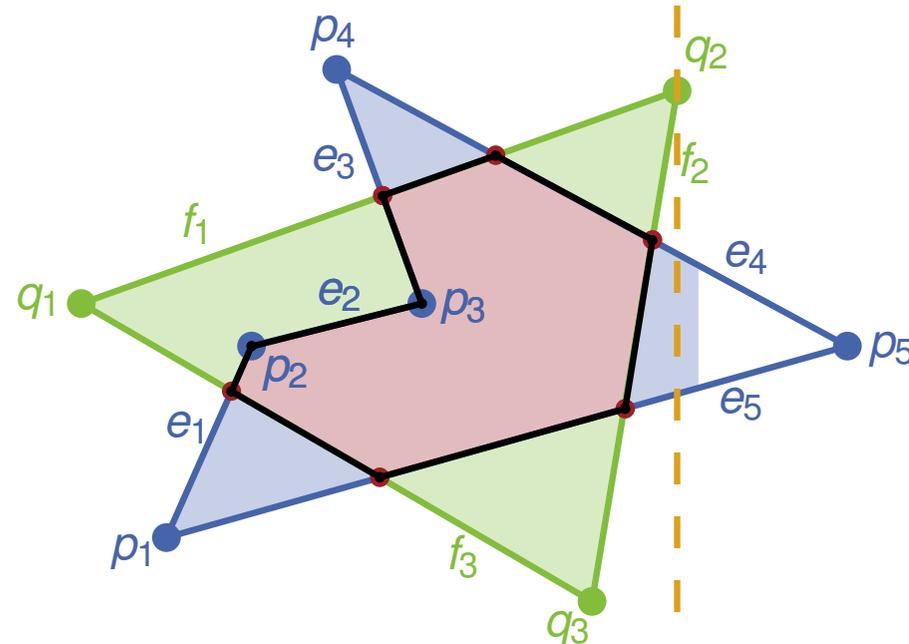


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

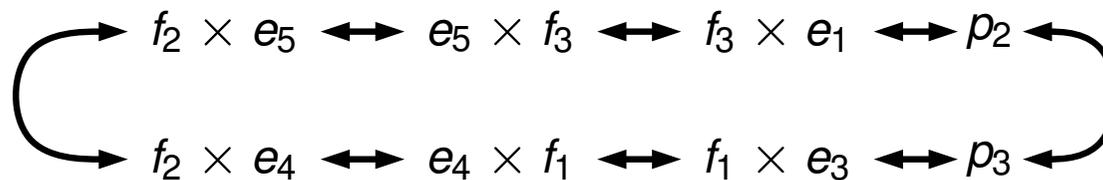
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →

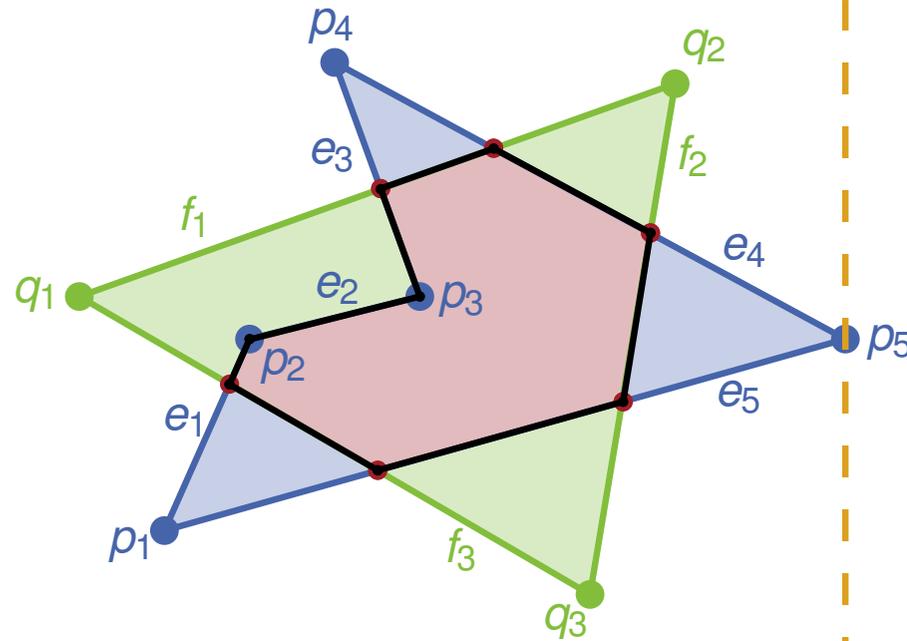


bisher berechnete Polygonzüge:



# Problem 2 (c) – Beispiel

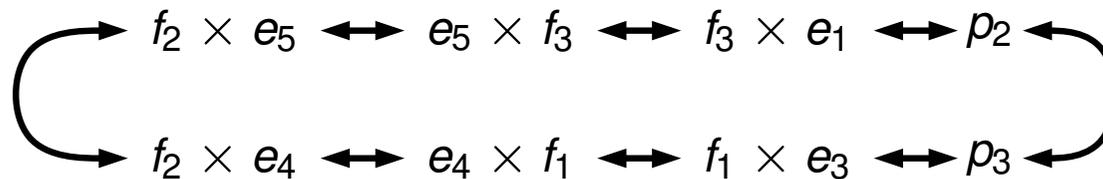
(Sweep-Line Zustand nicht explizit angegeben)



Event-Point Schedule →



bisher berechnete Polygonzüge:



# Problem 2(c) – Anmerkungen

## **Nicht einfacher Schnitt:**

Falls  $P \cap Q$  nicht (wie im Beispiel) aus einem Polygon, sondern aus mehreren besteht, so bleiben am Ende einfach mehrere Listen übrig.

## Nicht einfacher Schnitt:

Falls  $P \cap Q$  nicht (wie im Beispiel) aus einem Polygon, sondern aus mehreren besteht, so bleiben am Ende einfach mehrere Listen übrig.

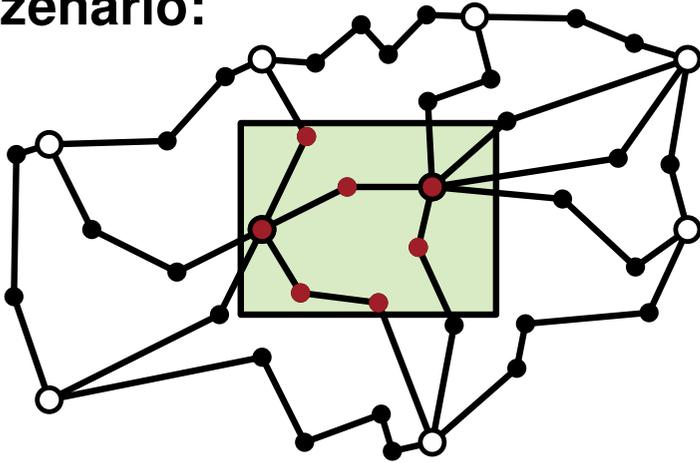
## Laufzeit:

- An jedem Haltepunkt werden konstant viele Aktionen durchgeführt.
- Manche dieser Aktionen brauchen  $O(\log n)$  Zeit.
- Haltepunkte sind genau die Eckpunkte der Polygone und die Schnittpunkte.  
⇒ Laufzeit insgesamt:  $O((n + k) \log n)$

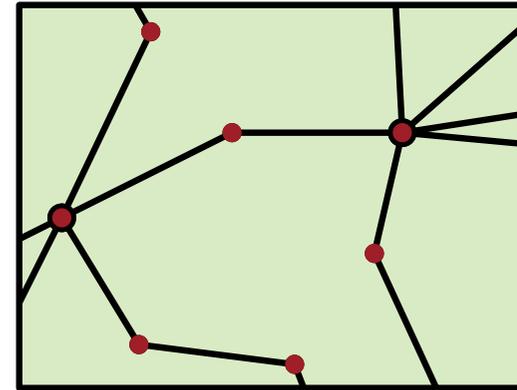
# Quadtree

# Wiederholung: Motivation

Szenario:



großer Straßengraph



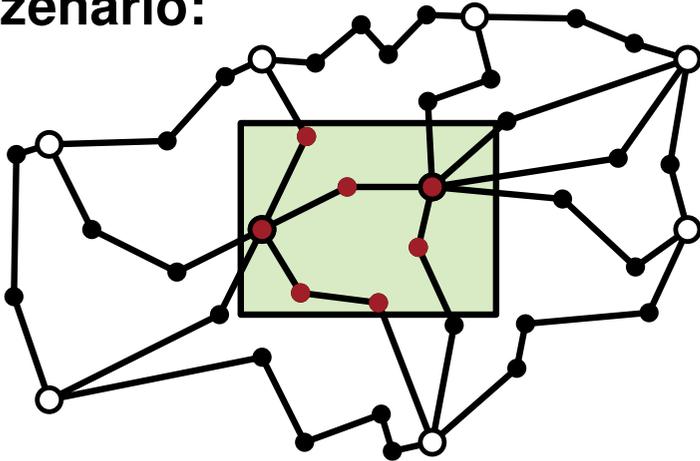
zeige nur einen kleinen Ausschnitt

## Problem: Bereichsanfrage

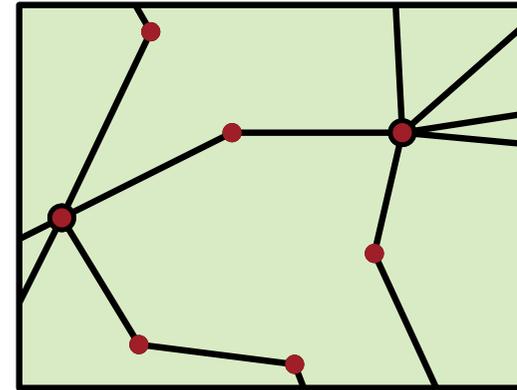
Gegeben eine Punktmenge  $P$  (z.B. die Knoten in einem Straßengraph) sowie ein Rechteck  $R$ , finde **alle Punkte aus  $P$ , die in  $R$  liegen.**

# Wiederholung: Motivation

Szenario:



großer Straßengraph



zeige nur einen kleinen Ausschnitt

## Problem: Bereichsanfrage

Gegeben eine Punktmenge  $P$  (z.B. die Knoten in einem Straßengraph) sowie ein Rechteck  $R$ , finde **alle Punkte aus  $P$ , die in  $R$  liegen.**

## Einfache Lösung:

Teste für jeden Punkt  $p \in P$ , ob  $p$  in  $R$  liegt.

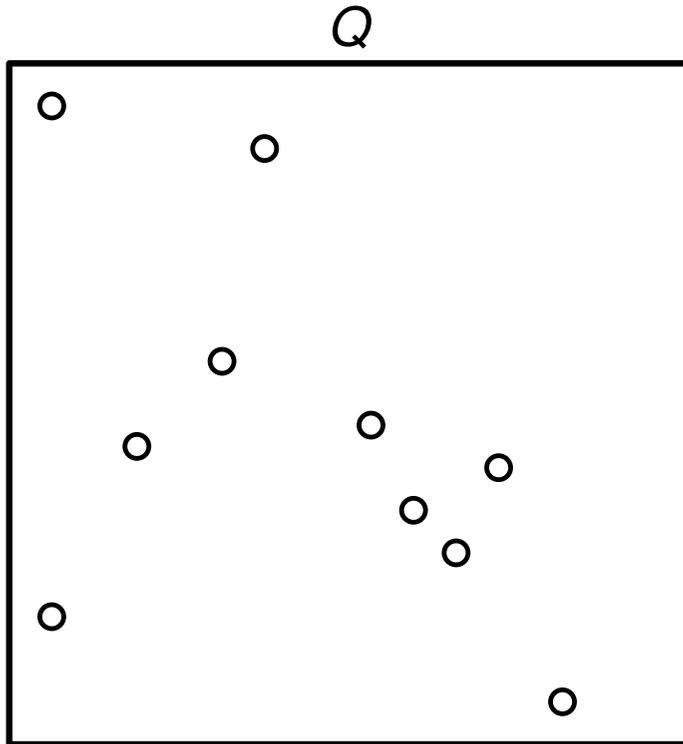
⇒ Laufzeit  $O(|P|)$ , dabei werden ggf. sehr viel weniger Punkte angezeigt.

## Ziel im Folgenden:

Nutze aus, dass nur ein sehr kleiner Anteil der  $|P|$  Punkte angezeigt werden muss.

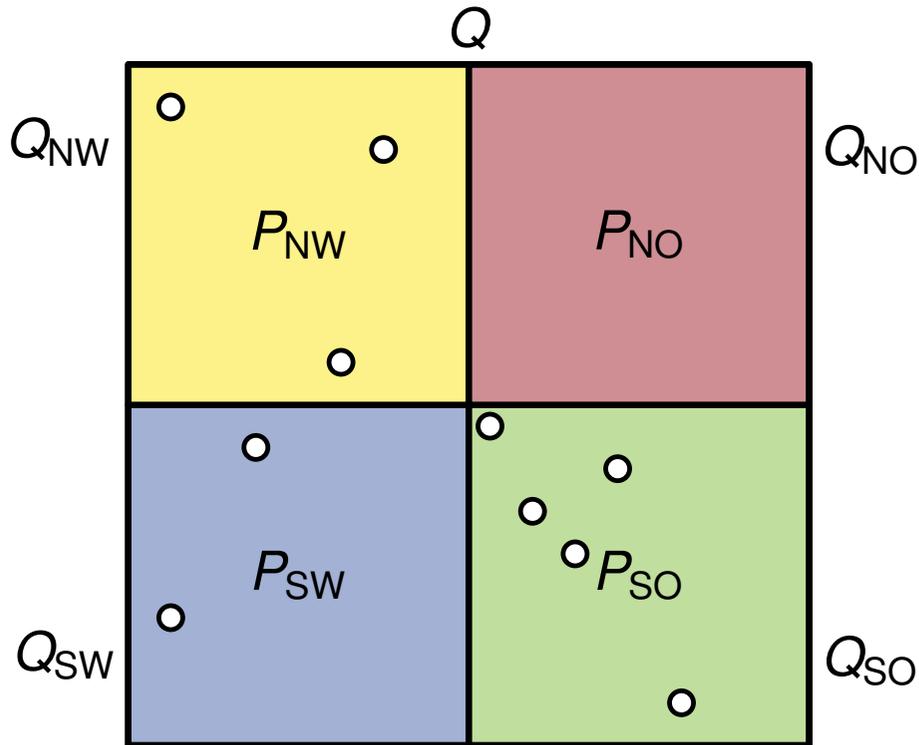
# Wiederholung: Quadtree

Betrachte Punktmenge  $P$  in einem Quadrat  $Q$ .



# Wiederholung: Quadtree

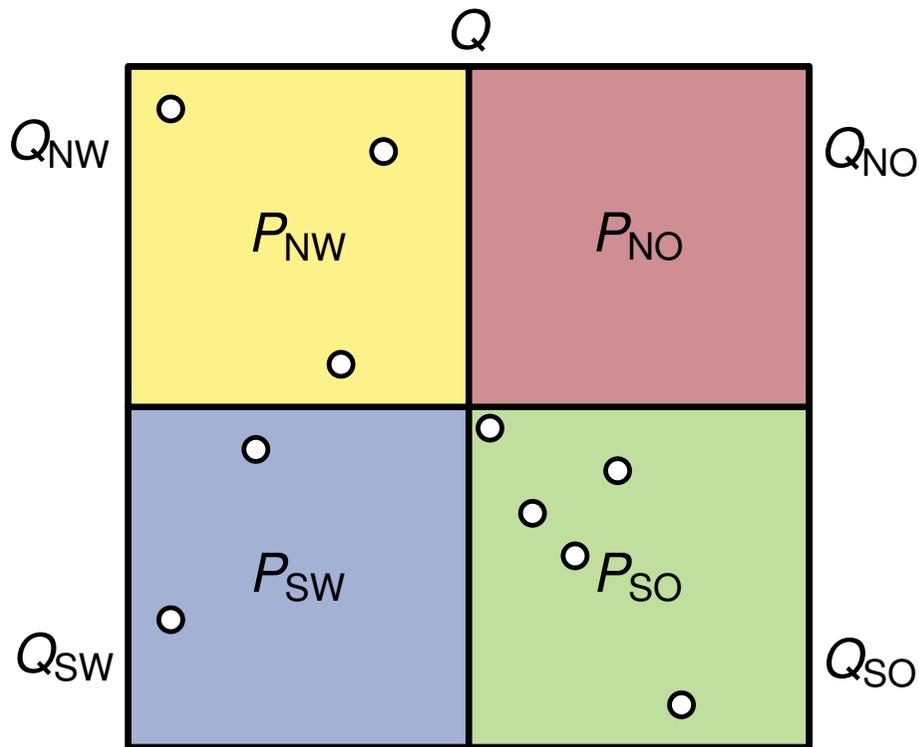
Betrachte Punktmenge  $P$  in einem Quadrat  $Q$ .



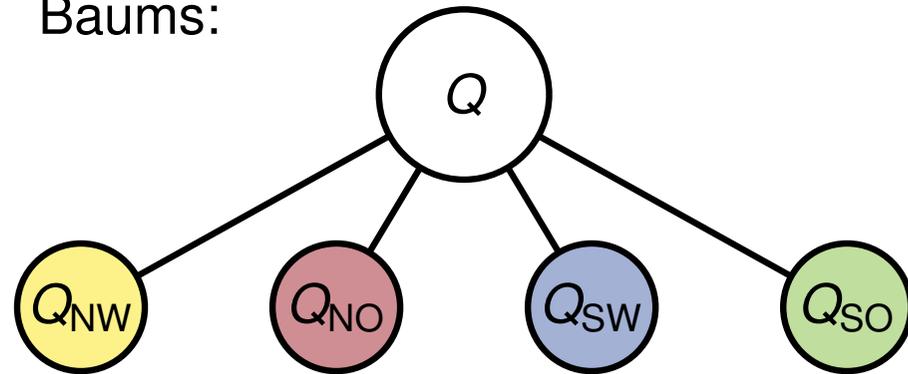
- Wenn  $|P| > 1$ :
- Unterteile  $Q$  in vier Quadrate  $Q_{NW}$ ,  $Q_{NO}$ ,  $Q_{SW}$ ,  $Q_{SO}$ .
  - Zerlege  $P$  entsprechend in  $P_{NW}$ ,  $P_{NO}$ ,  $P_{SW}$ ,  $P_{SO}$ , sodass  $P_i$  komplett in  $Q_i$  liegt (für  $i \in \{NW, NO, SW, SO\}$ ).
  - Fahre rekursiv mit  $P_i$  und  $Q_i$  fort (für  $i \in \{NW, NO, SW, SO\}$ ).

# Wiederholung: Quadtree

Betrachte Punktmenge  $P$  in einem Quadrat  $Q$ .



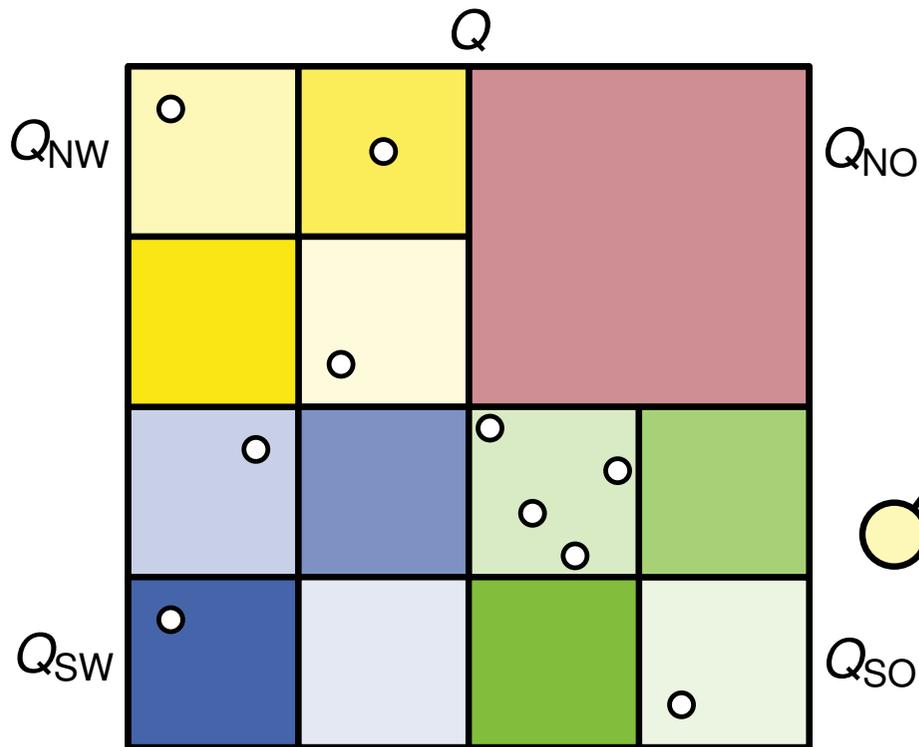
Interpretiere die Quadrate als Knoten eines Baums:



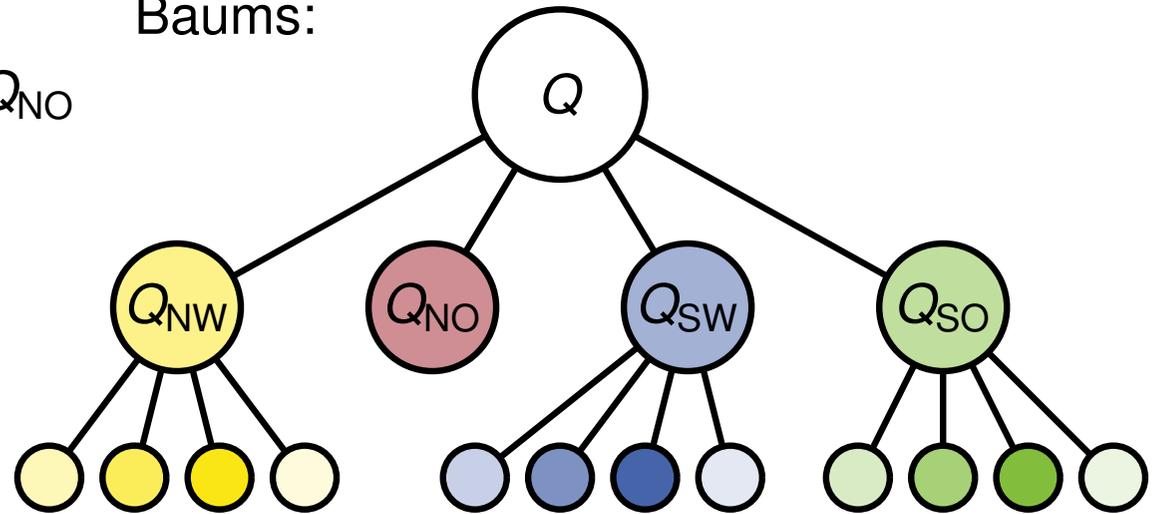
- Wenn  $|P| > 1$ :
- Unterteile  $Q$  in vier Quadrate  $Q_{NW}$ ,  $Q_{NO}$ ,  $Q_{SW}$ ,  $Q_{SO}$ .
  - Zerlege  $P$  entsprechend in  $P_{NW}$ ,  $P_{NO}$ ,  $P_{SW}$ ,  $P_{SO}$ , sodass  $P_i$  komplett in  $Q_i$  liegt (für  $i \in \{NW, NO, SW, SO\}$ ).
  - Fahre rekursiv mit  $P_i$  und  $Q_i$  fort (für  $i \in \{NW, NO, SW, SO\}$ ).

# Wiederholung: Quadtree

Betrachte Punktmenge  $P$  in einem Quadrat  $Q$ .



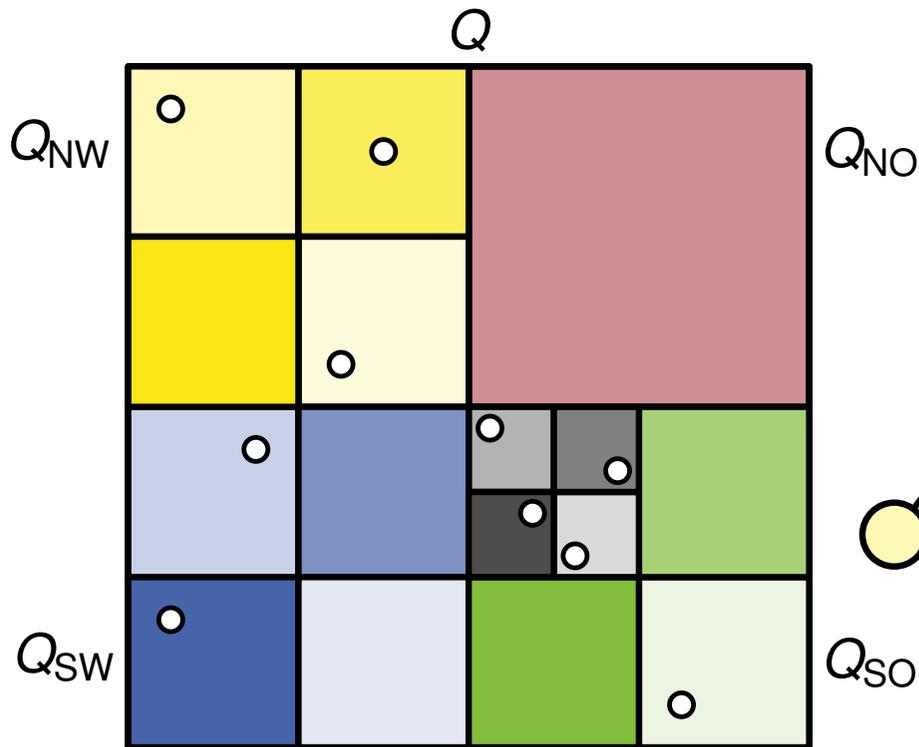
Interpretiere die Quadrate als Knoten eines Baums:



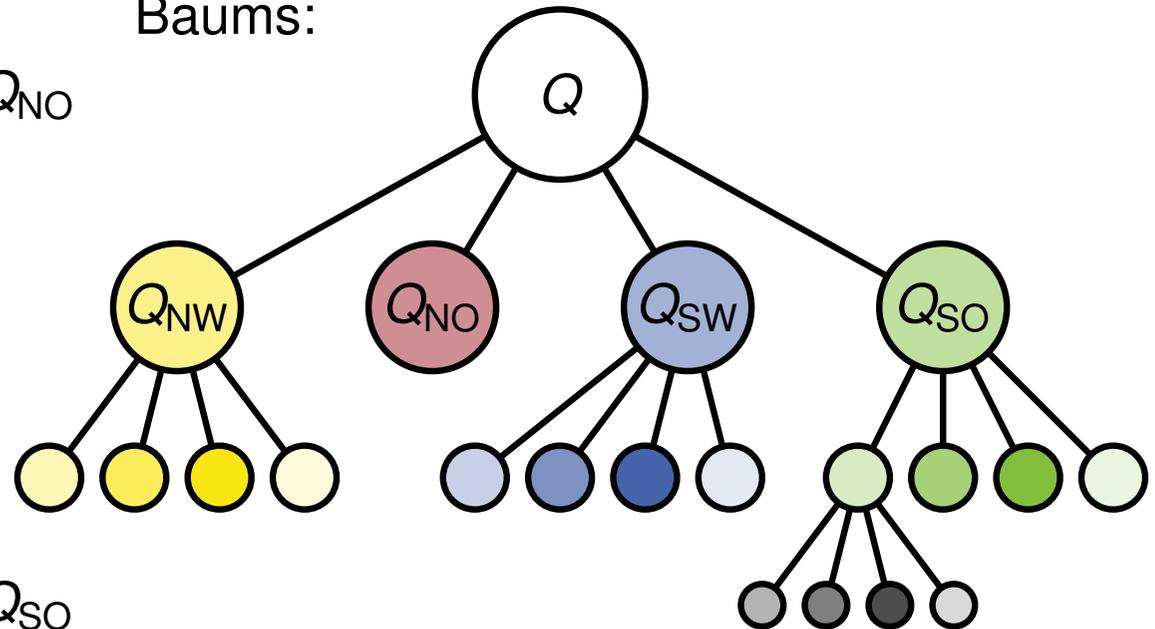
- Wenn  $|P| > 1$ :
- Unterteile  $Q$  in vier Quadrate  $Q_{NW}$ ,  $Q_{NO}$ ,  $Q_{SW}$ ,  $Q_{SO}$ .
  - Zerlege  $P$  entsprechend in  $P_{NW}$ ,  $P_{NO}$ ,  $P_{SW}$ ,  $P_{SO}$ , sodass  $P_i$  komplett in  $Q_i$  liegt (für  $i \in \{NW, NO, SW, SO\}$ ).
  - Fahre rekursiv mit  $P_i$  und  $Q_i$  fort (für  $i \in \{NW, NO, SW, SO\}$ ).

# Wiederholung: Quadtree

Betrachte Punktmenge  $P$  in einem Quadrat  $Q$ .



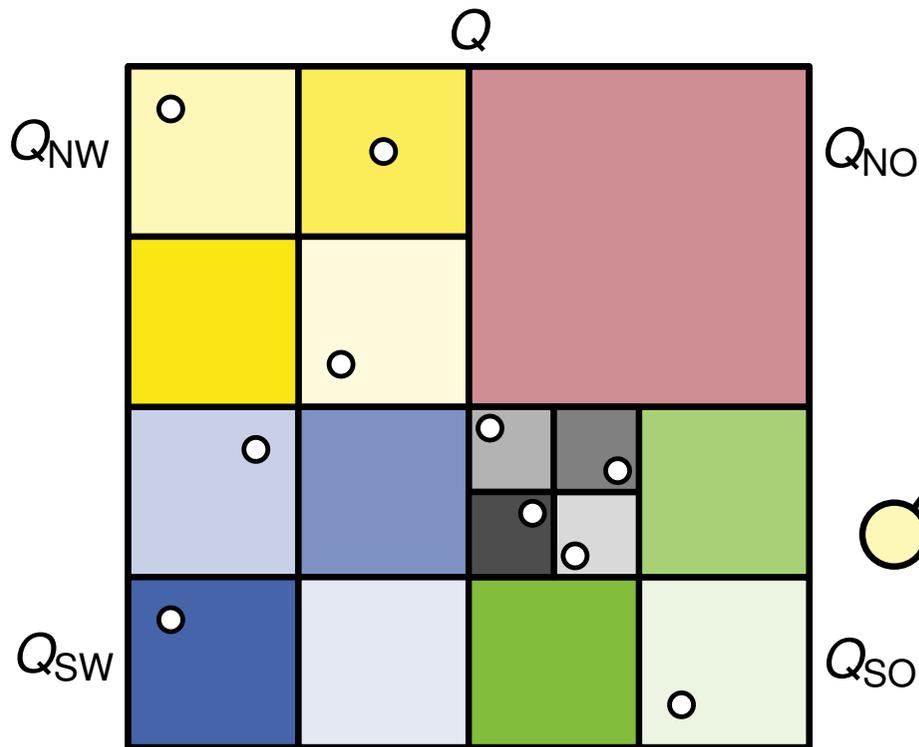
Interpretiere die Quadrate als Knoten eines Baums:



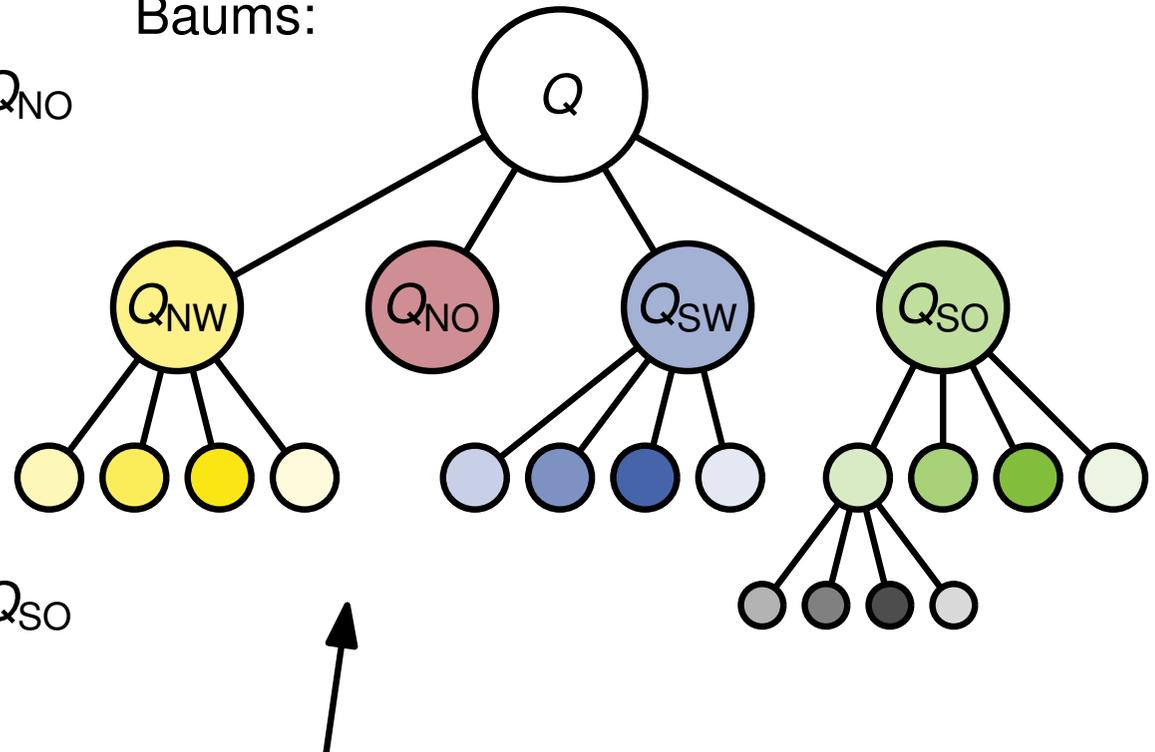
- Wenn  $|P| > 1$ :
- Unterteile  $Q$  in vier Quadrate  $Q_{NW}$ ,  $Q_{NO}$ ,  $Q_{SW}$ ,  $Q_{SO}$ .
  - Zerlege  $P$  entsprechend in  $P_{NW}$ ,  $P_{NO}$ ,  $P_{SW}$ ,  $P_{SO}$ , sodass  $P_i$  komplett in  $Q_i$  liegt (für  $i \in \{NW, NO, SW, SO\}$ ).
  - Fahre rekursiv mit  $P_i$  und  $Q_i$  fort (für  $i \in \{NW, NO, SW, SO\}$ ).

# Wiederholung: Quadtree

Betrachte Punktmenge  $P$  in einem Quadrat  $Q$ .



Interpretiere die Quadrate als Knoten eines Baums:



**Quadtree**  $T(P)$  bezüglich Quadrat  $Q$

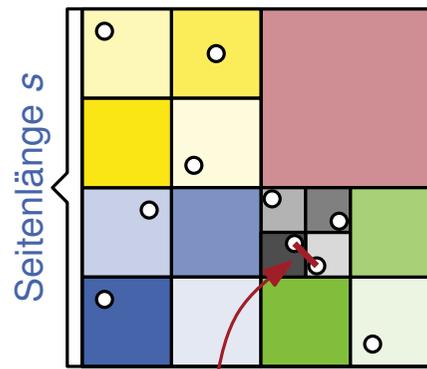
Speichere zusätzlich für jedes Blatt in  $T(P)$  den eventuell enthaltenen Knoten.

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

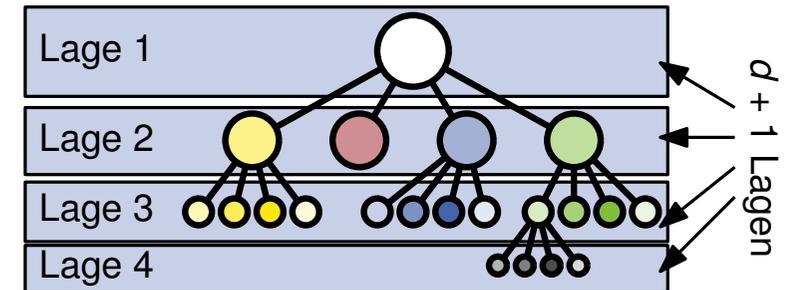
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$

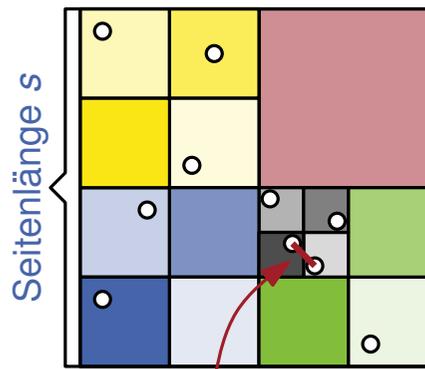


# Problem 3(a)

## Satz: Tiefe des Quadrtrees

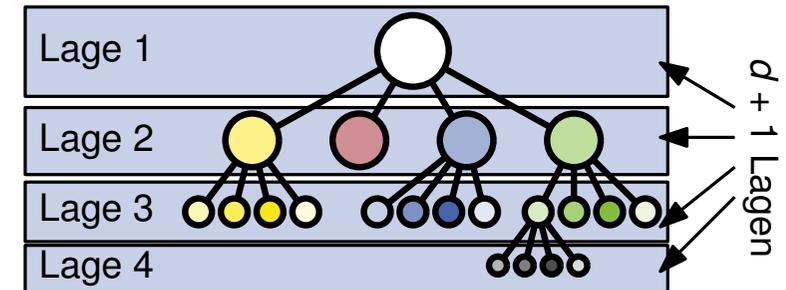
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



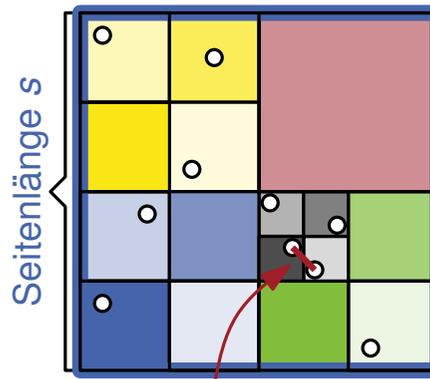
- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left(\frac{1}{2}\right)^{i-1} \cdot s$ .
  - $v_Q$  in Lage 1  $\Rightarrow$   $Q$  hat Seitenlänge  $s$
  - $v_Q$  in Lage 2  $\Rightarrow$   $Q$  hat Seitenlänge  $\frac{1}{2} \cdot s$
  - ...

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

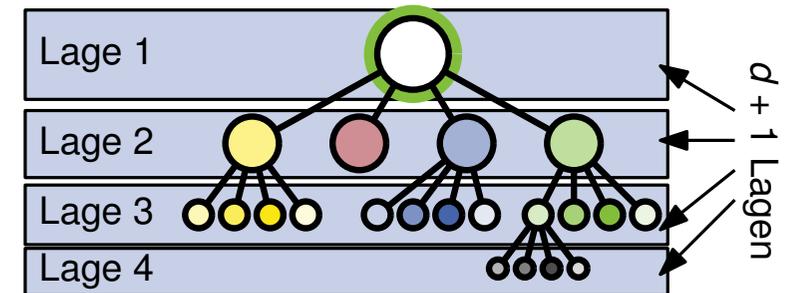
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



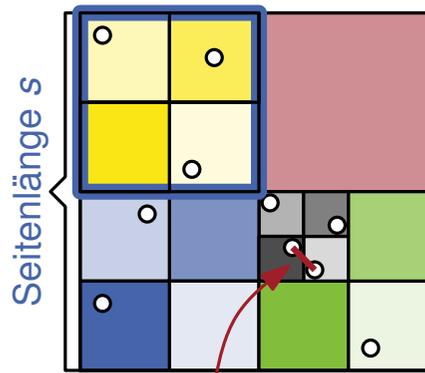
- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left(\frac{1}{2}\right)^{i-1} \cdot s$ .
- $v_Q$  in Lage 1  $\Rightarrow$   $Q$  hat Seitenlänge  $s$
- $v_Q$  in Lage 2  $\Rightarrow$   $Q$  hat Seitenlänge  $\frac{1}{2} \cdot s$
- ...

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

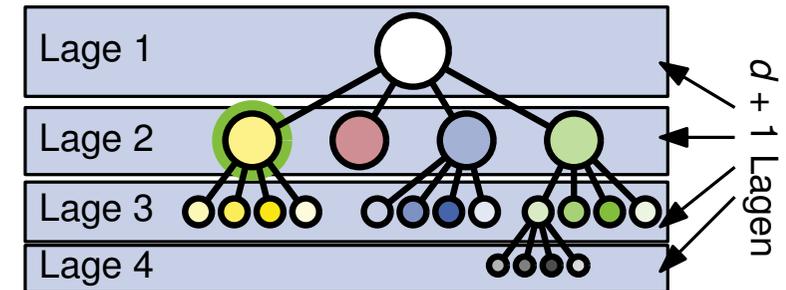
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



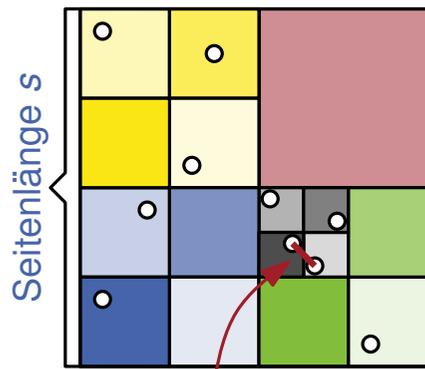
- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left(\frac{1}{2}\right)^{i-1} \cdot s$ .
- $v_Q$  in Lage 1  $\Rightarrow$   $Q$  hat Seitenlänge  $s$
- $v_Q$  in Lage 2  $\Rightarrow$   $Q$  hat Seitenlänge  $\frac{1}{2} \cdot s$
- ...

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

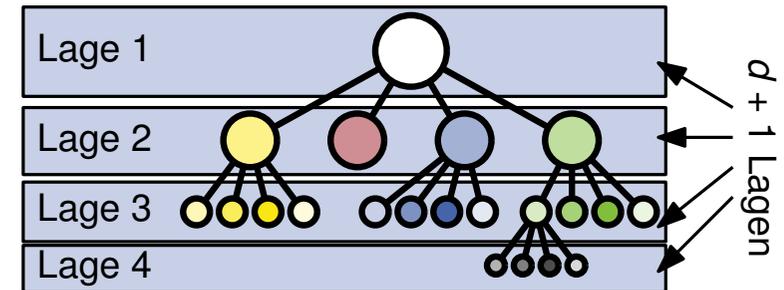
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



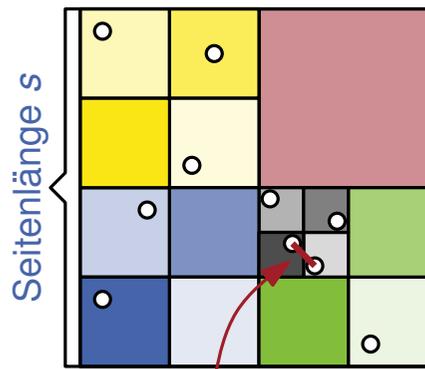
- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left(\frac{1}{2}\right)^{i-1} \cdot s$ .
- Es gibt einen Knoten in Lage  $d + 1$ , sei  $v_Q$  sein Vorgänger in Lage  $d$ .
- $\Rightarrow Q$  hat Seitenlänge  $\left(\frac{1}{2}\right)^{d-1} \cdot s$  und enthält mindestens zwei Punkte.

# Problem 3(a)

## Satz: Tiefe des Quadrees

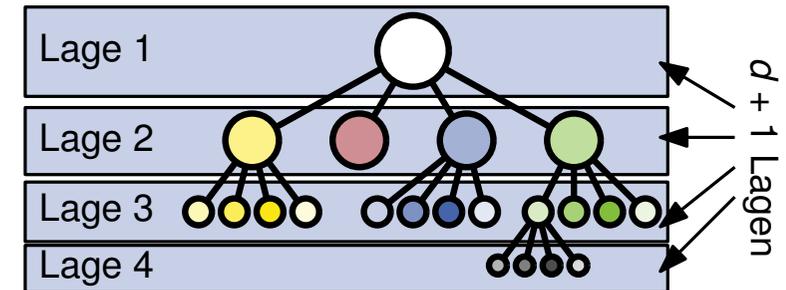
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrees  $T(P)$



- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left(\frac{1}{2}\right)^{i-1} \cdot s$ .
- Es gibt einen Knoten in Lage  $d + 1$ , sei  $v_Q$  sein Vorgänger in Lage  $d$ .
- $\Rightarrow Q$  hat Seitenlänge  $\left(\frac{1}{2}\right)^{d-1} \cdot s$  und enthält mindestens zwei Punkte.

**Annahme:**  $d > \log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$

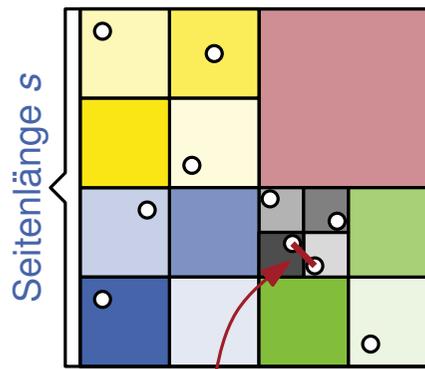
$$\text{Seitenlänge von } Q = \frac{1}{2^{d-1}} \cdot s$$

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

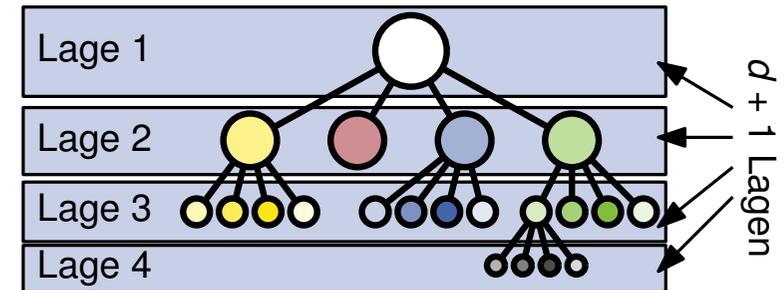
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{i-1} \cdot s$ .
- Es gibt einen Knoten in Lage  $d + 1$ , sei  $v_Q$  sein Vorgänger in Lage  $d$ .
- $\Rightarrow Q$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{d-1} \cdot s$  und enthält mindestens zwei Punkte.

**Annahme:**  $d > \log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$

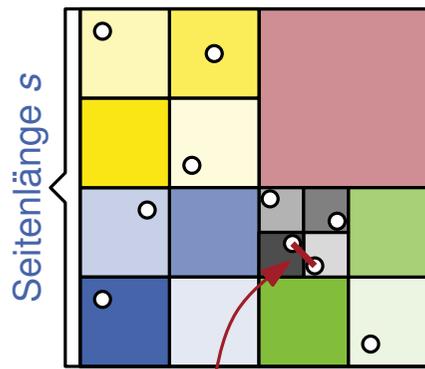
$$\text{Seitenlänge von } Q = \frac{1}{2^{d-1}} \cdot s < \frac{1}{2^{\log_2 \left( \frac{s}{c} \right) + \frac{1}{2}}} \cdot s$$

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

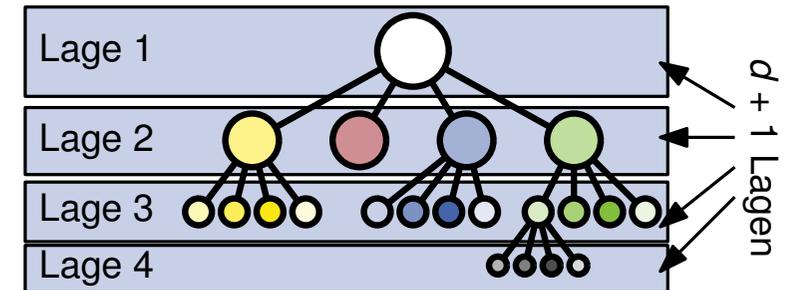
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{i-1} \cdot s$ .
- Es gibt einen Knoten in Lage  $d + 1$ , sei  $v_Q$  sein Vorgänger in Lage  $d$ .
- $\Rightarrow Q$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{d-1} \cdot s$  und enthält mindestens zwei Punkte.

**Annahme:**  $d > \log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$

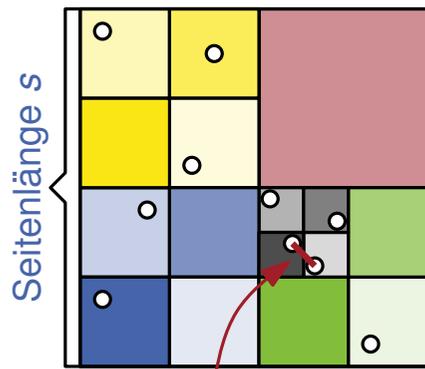
$$\text{Seitenlänge von } Q = \frac{1}{2^{d-1}} \cdot s < \frac{1}{2^{\log_2 \left( \frac{s}{c} \right) + \frac{1}{2}}} \cdot s = \frac{1}{\frac{s}{c}} \cdot \frac{1}{\sqrt{2}} \cdot s$$

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

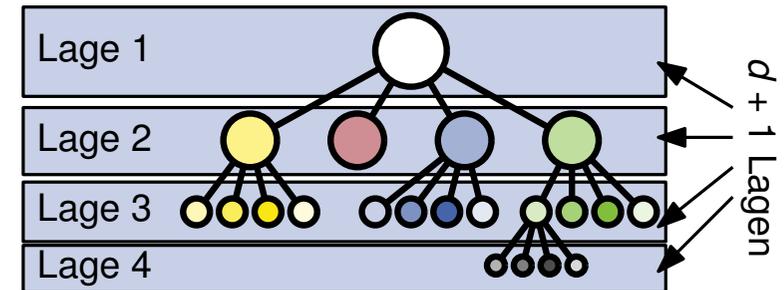
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{i-1} \cdot s$ .
- Es gibt einen Knoten in Lage  $d + 1$ , sei  $v_Q$  sein Vorgänger in Lage  $d$ .
- $\Rightarrow Q$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{d-1} \cdot s$  und enthält mindestens zwei Punkte.

**Annahme:**  $d > \log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$

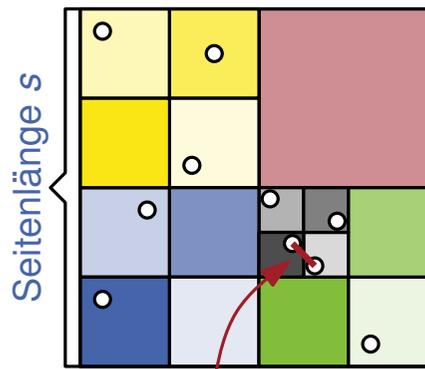
$$\text{Seitenlänge von } Q = \frac{1}{2^{d-1}} \cdot s < \frac{1}{2^{\log_2 \left( \frac{s}{c} \right) + \frac{1}{2}}} \cdot s = \frac{1}{\frac{s}{c}} \cdot \frac{1}{\sqrt{2}} \cdot s = \frac{c}{\sqrt{2}}$$

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

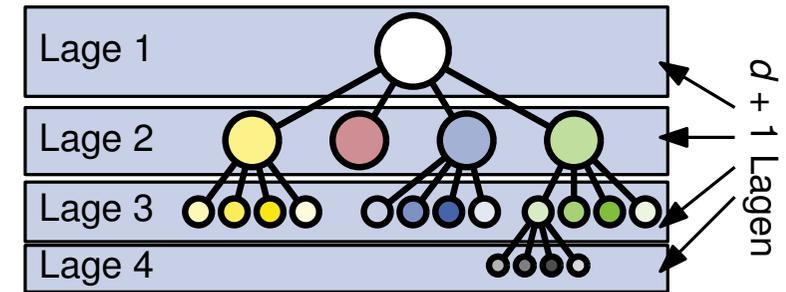
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{i-1} \cdot s$ .
- Es gibt einen Knoten in Lage  $d + 1$ , sei  $v_Q$  sein Vorgänger in Lage  $d$ .
- $\Rightarrow Q$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{d-1} \cdot s$  und enthält mindestens zwei Punkte.

**Annahme:**  $d > \log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$

$$\text{Seitenlänge von } Q = \frac{1}{2^{d-1}} \cdot s < \frac{1}{2^{\log_2 \left( \frac{s}{c} \right) + \frac{1}{2}}} \cdot s = \frac{1}{\frac{s}{c}} \cdot \frac{1}{\sqrt{2}} \cdot s = \frac{c}{\sqrt{2}}$$

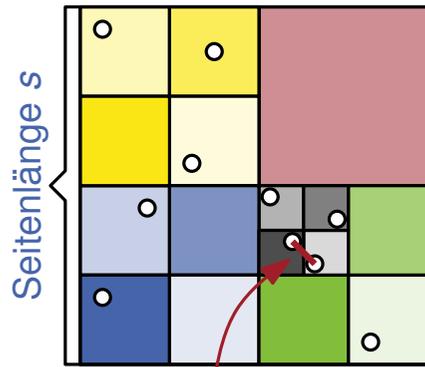
$$\Rightarrow \text{Diagonale von } Q < c$$

# Problem 3(a)

## Satz: Tiefe des Quadrtrees

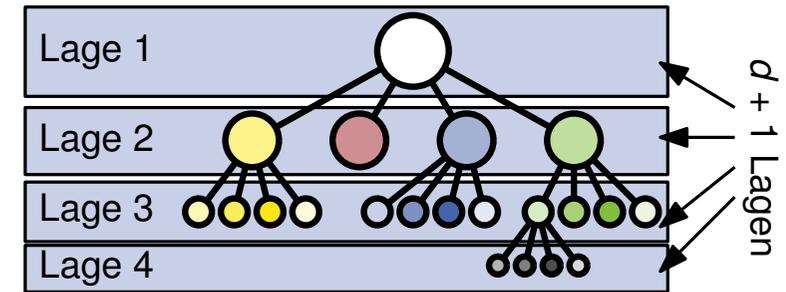
Die Tiefe  $d$  von  $T(P)$  ist höchstens  $\log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$ .

**Beweis:** Problem 3(a)



minimaler Abstand zwischen zwei Punkten  $c$

sei  $d$  die Tiefe des Quadrtrees  $T(P)$



- Quadrat  $Q$  mit zugehörigem Knoten  $v_Q$  in Lage  $i$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{i-1} \cdot s$ .
- Es gibt einen Knoten in Lage  $d + 1$ , sei  $v_Q$  sein Vorgänger in Lage  $d$ .
- $\Rightarrow Q$  hat Seitenlänge  $\left( \frac{1}{2} \right)^{d-1} \cdot s$  und enthält mindestens zwei Punkte.

**Annahme:**  $d > \log_2 \left( \frac{s}{c} \right) + \frac{3}{2}$

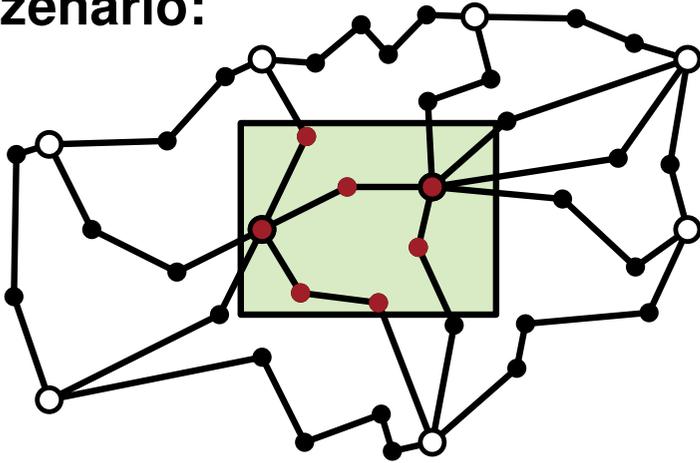
$$\text{Seitenlänge von } Q = \frac{1}{2^{d-1}} \cdot s < \frac{1}{2^{\log_2 \left( \frac{s}{c} \right) + \frac{1}{2}}} \cdot s = \frac{1}{\frac{s}{c}} \cdot \frac{1}{\sqrt{2}} \cdot s = \frac{c}{\sqrt{2}}$$

$$\Rightarrow \text{Diagonale von } Q < c$$

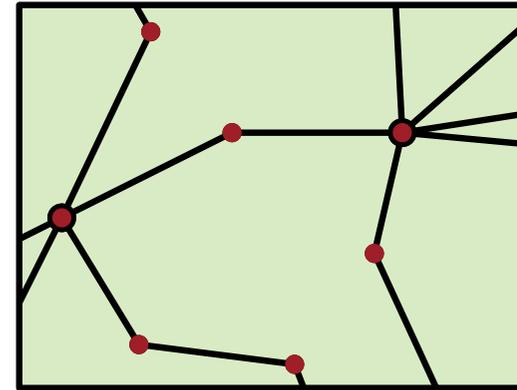
$\Rightarrow Q$  kann keine zwei Punkte mit Abstand mindestens  $c$  enthalten. ⚡

# Wiederholung: Bereichsanfrage

Szenario:



großer Straßengraph



zeige nur einen kleinen Ausschnitt

## Problem: Bereichsanfrage

Gegeben eine Punktmenge  $P$  (die Knoten in dem Straßengraph) sowie ein Rechteck  $R$ , finde **alle Punkte aus  $P$  die in  $R$  liegen**.

## Einfache Lösung:

Teste für jeden Punkt  $p \in P$ , ob  $p$  in  $R$  liegt.

⇒ Laufzeit  $O(|P|)$ , dabei werden ggf. sehr viel weniger Punkte angezeigt.

## Ziel im Folgenden:

Nutze aus, dass nur ein sehr kleiner Anteil der  $|P|$  Punkte angezeigt werden muss.

# Wiederholung: Bereichsanfrage

BEREICHSANFRAGE( $v_Q$ ,  $R$ ) ( $v_Q$  ist ein Knoten von  $T(P)$  (zu Beginn die Wurzel))

**if**  $Q \subseteq R$  **then** vorzeitiges Rekursionsende

| gib alle Punkte in  $Q$  aus

**else if**  $v_Q$  ist ein Blatt **then** Rekursionsende

| **if**  $Q \cap R$  enthält einen Punkt  $p \in P$  **then** gib  $p$  aus

**else** Rekursionsaufruf

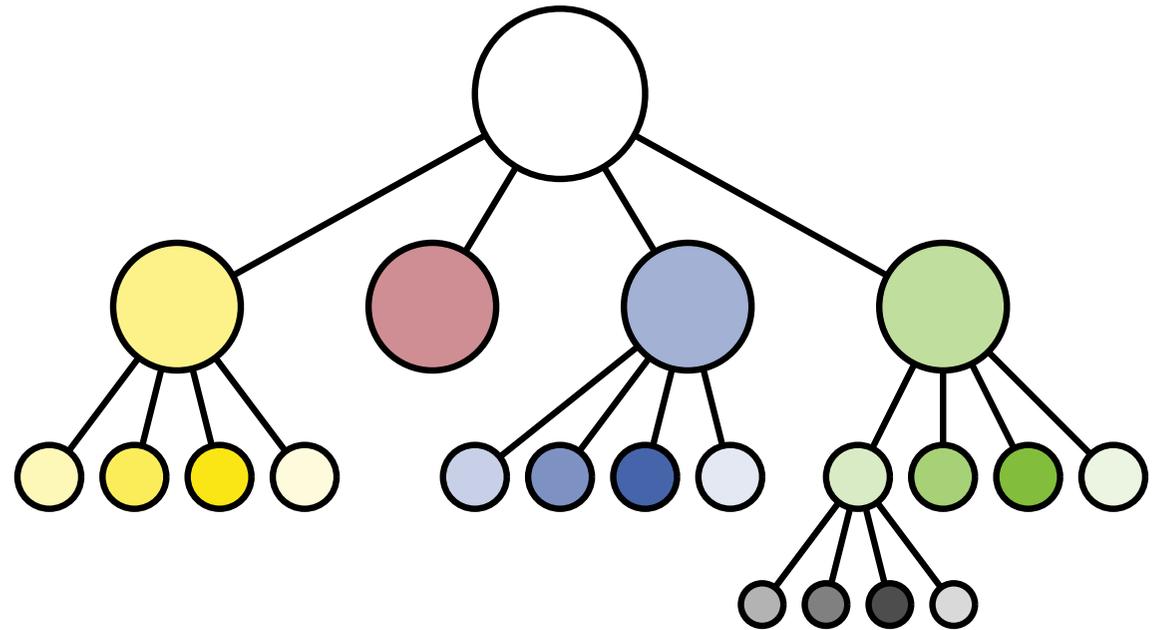
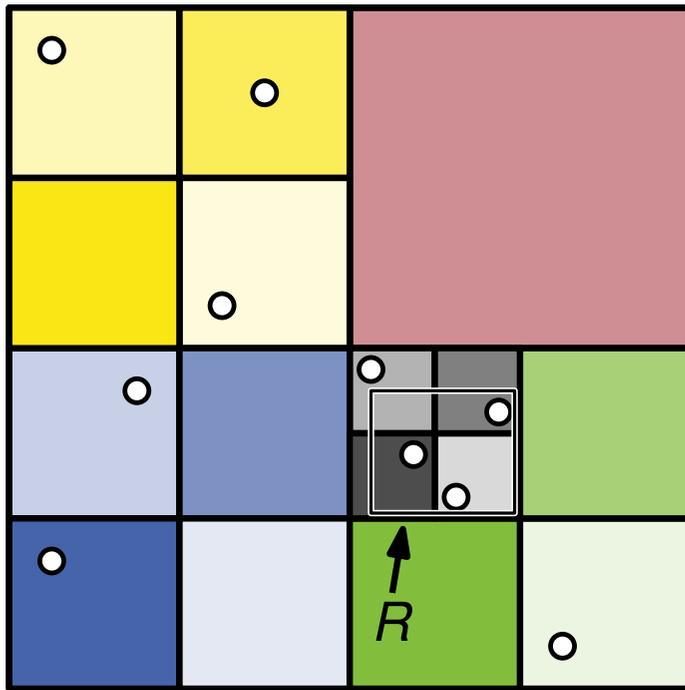
| **foreach** Kind  $v_{Q_i}$  von  $v_Q$  mit  $i \in \{NW, NO, SW, SO\}$  **do**

| | **if**  $Q_i \cap R \neq \emptyset$  **then** BEREICHSANFRAGE( $v_{Q_i}$ ,  $R$ )

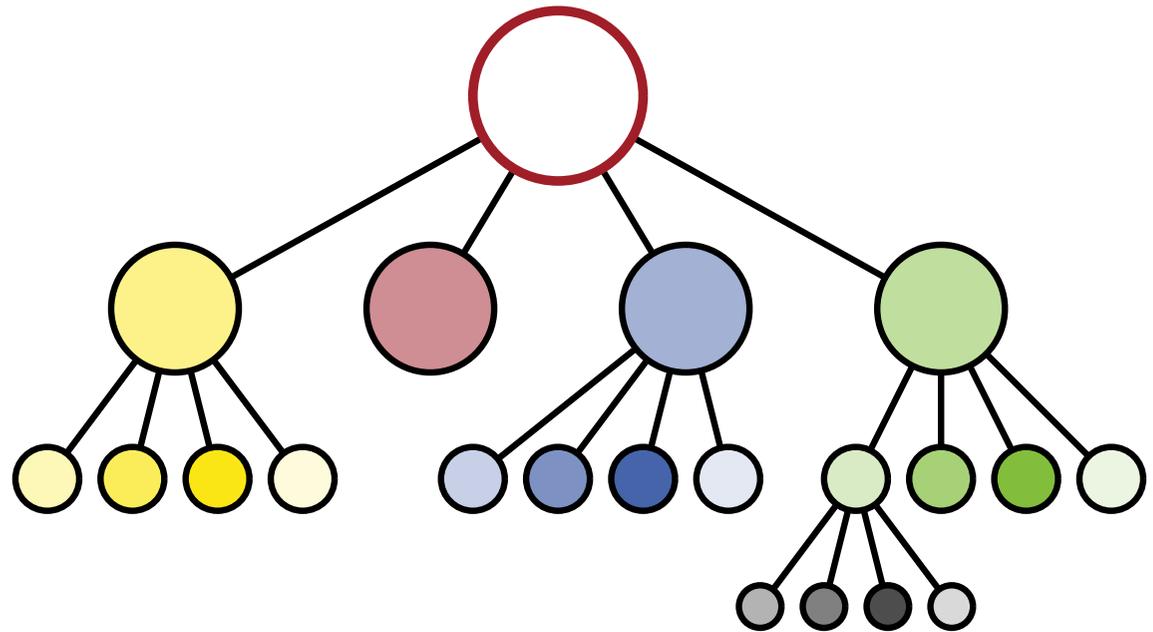
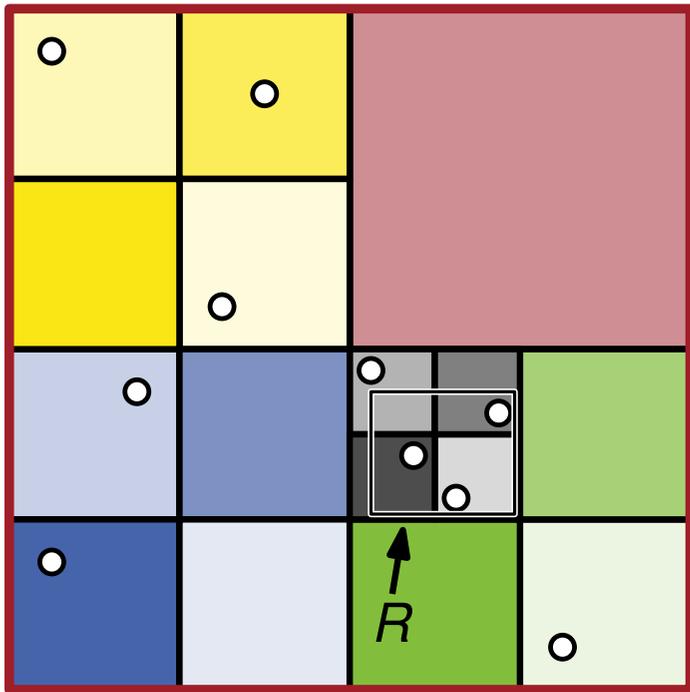
Es kann passieren, dass der gesamte Baum  $T(P)$  traversiert wird, aber:

- Ein Quadrat außerhalb von  $R$  wird im **Rekursionsaufruf** nicht weiter verfolgt.
- Die Traversierung wird **vorzeitig abgebrochen**, wenn ein Quadrat komplett in  $R$  liegt. (Es werden nur noch die enthaltenen Knoten ausgegeben.)

# Wiederholung: Bereichsanfrage

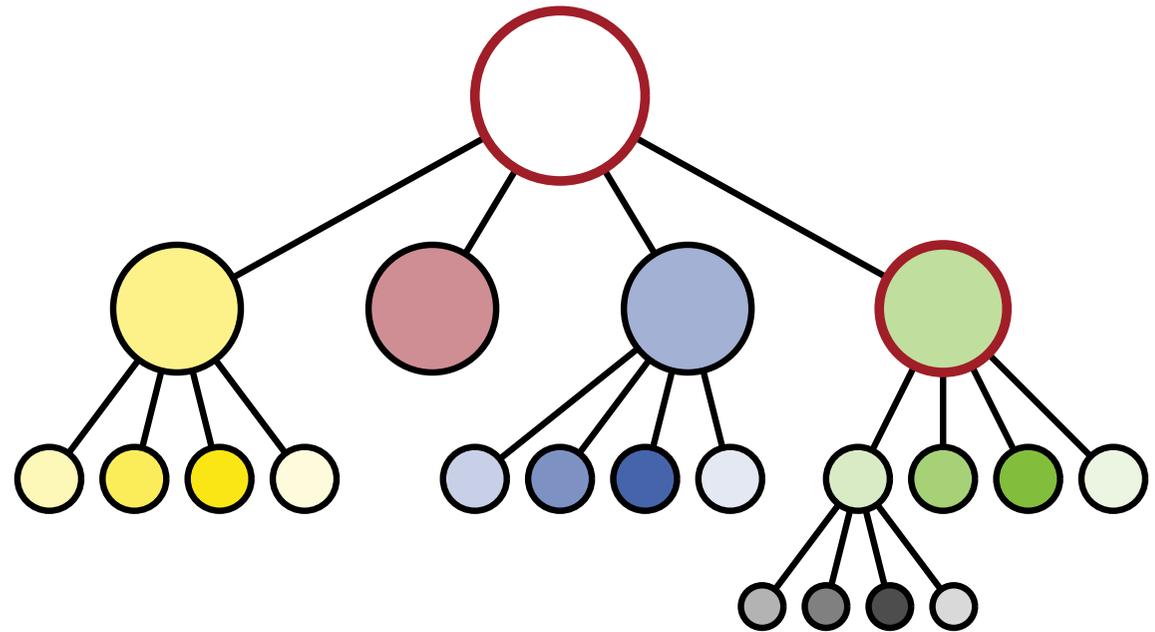
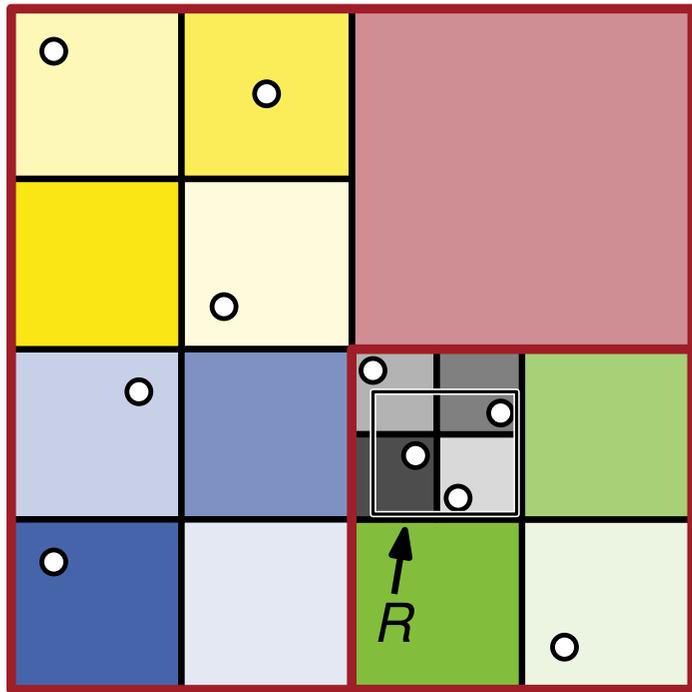


# Wiederholung: Bereichsanfrage



Starte mit der Wurzel

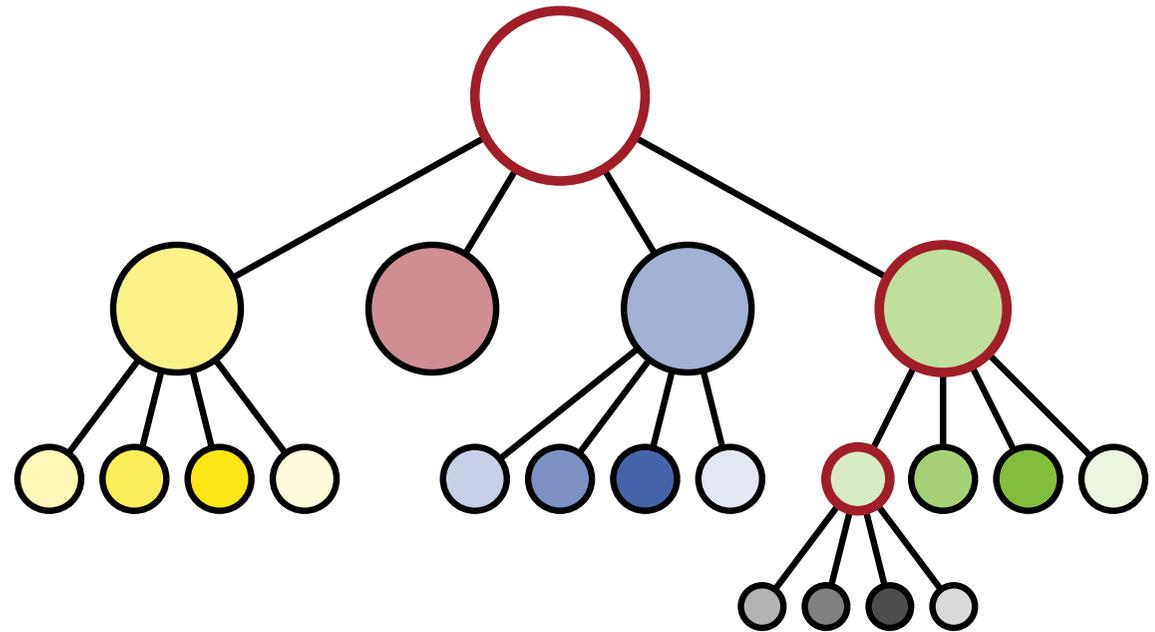
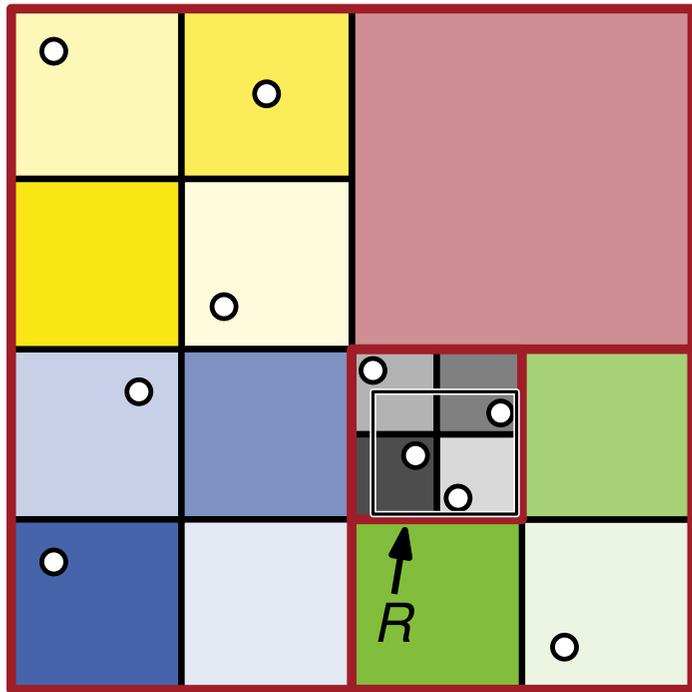
# Wiederholung: Bereichsanfrage



Starte mit der Wurzel

→ Rekursionsaufruf für nur eins der Kinder

# Wiederholung: Bereichsanfrage

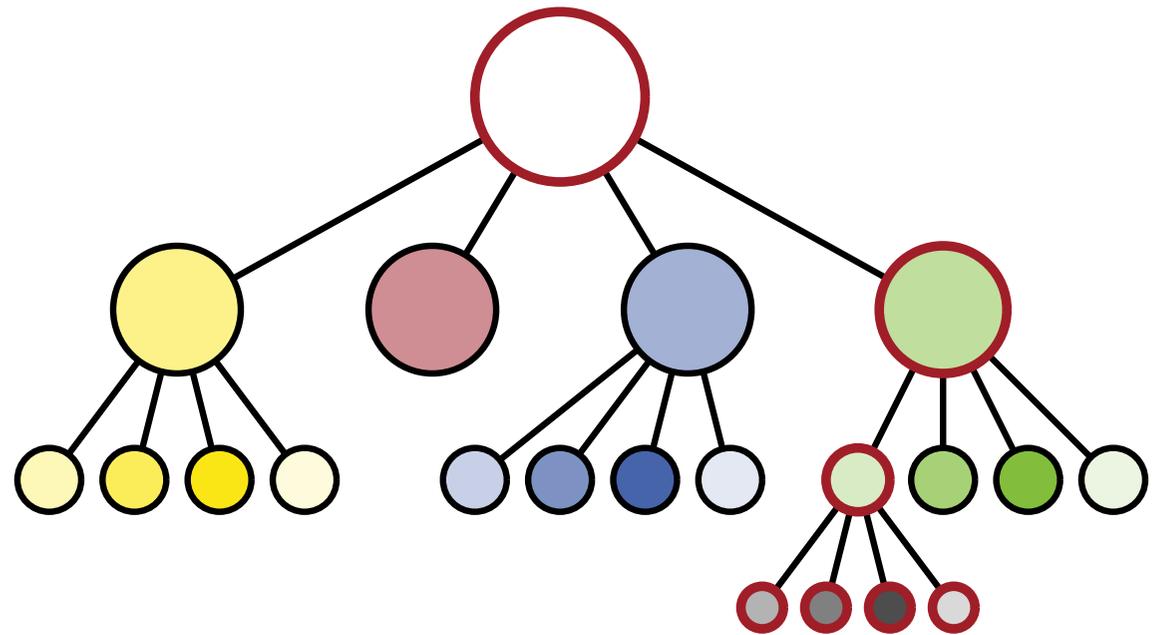
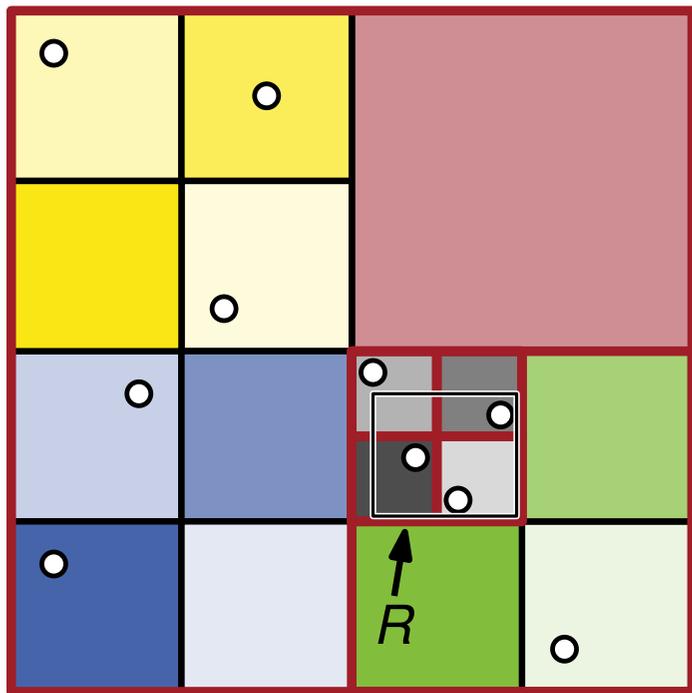


Starte mit der Wurzel

→ Rekursionsaufruf für nur eins der Kinder

→ Rekursionsaufruf für nur eins der Kinder

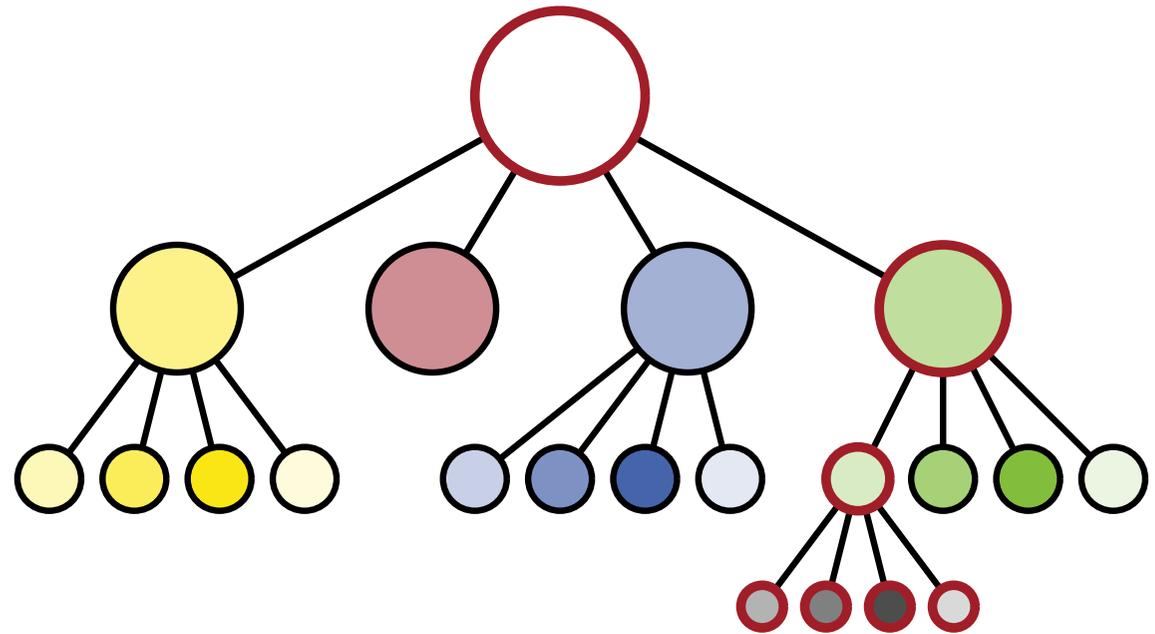
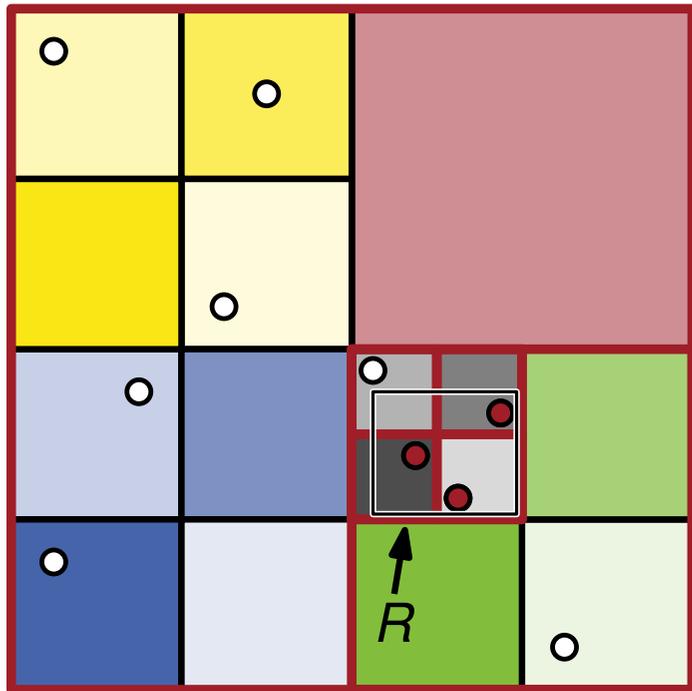
# Wiederholung: Bereichsanfrage



Starte mit der Wurzel

- Rekursionsaufruf für nur eins der Kinder
- Rekursionsaufruf für nur eins der Kinder
- Rekursionsaufruf für alle vier Kinder

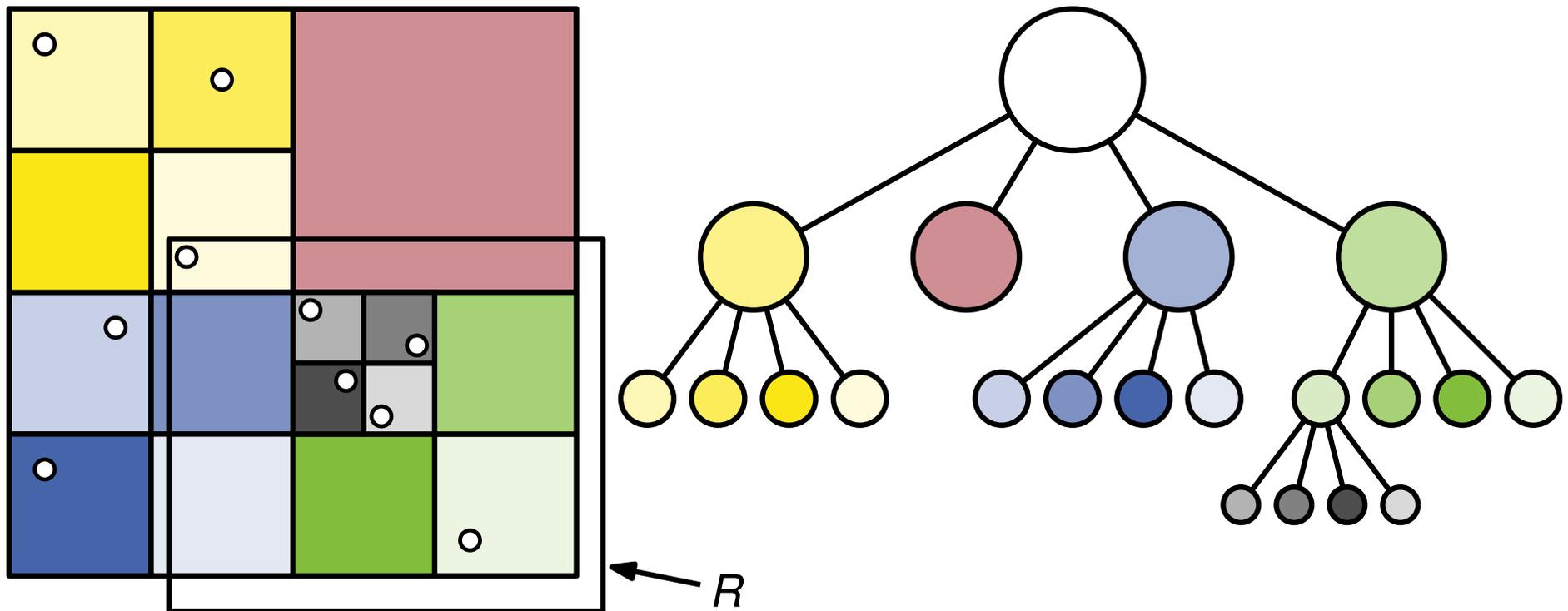
# Wiederholung: Bereichsanfrage



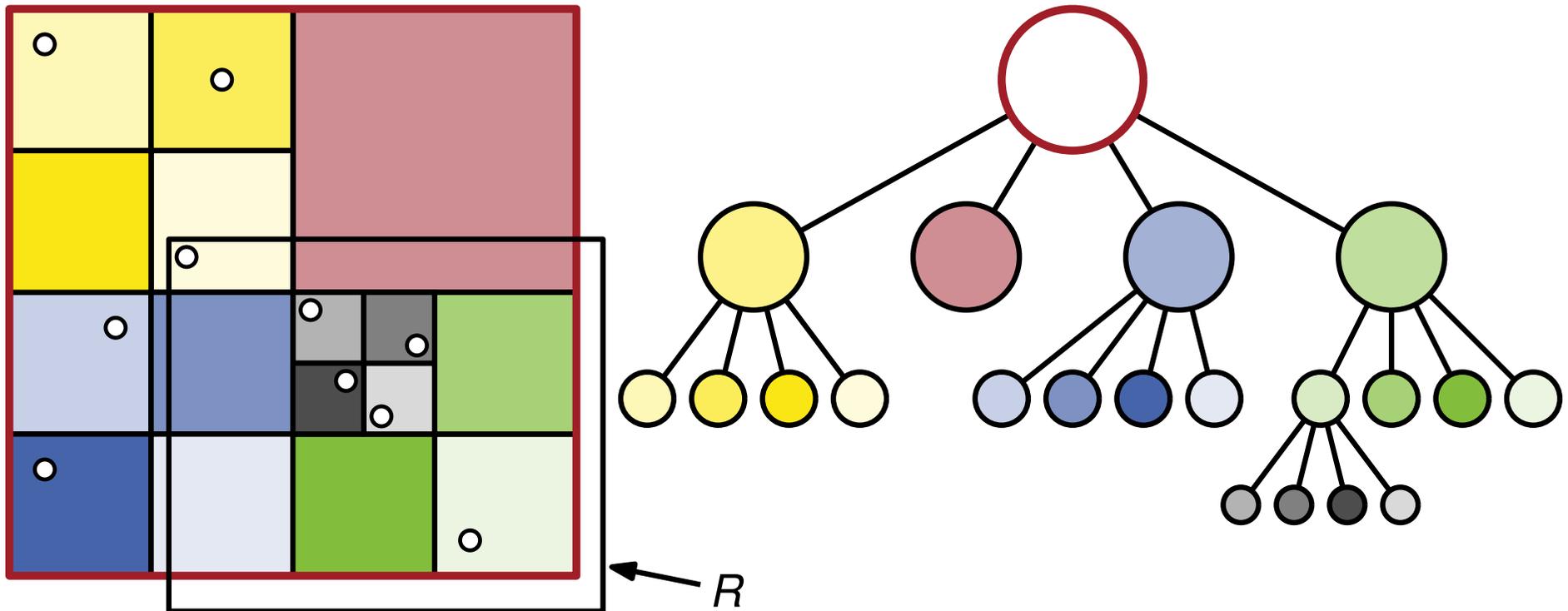
Starte mit der Wurzel

- Rekursionsaufruf für nur eins der Kinder
- Rekursionsaufruf für nur eins der Kinder
- Rekursionsaufruf für alle vier Kinder
- Rekursionsende (Blätter erreicht): gib resultierende Knoten aus

# Wiederholung: Bereichsanfrage

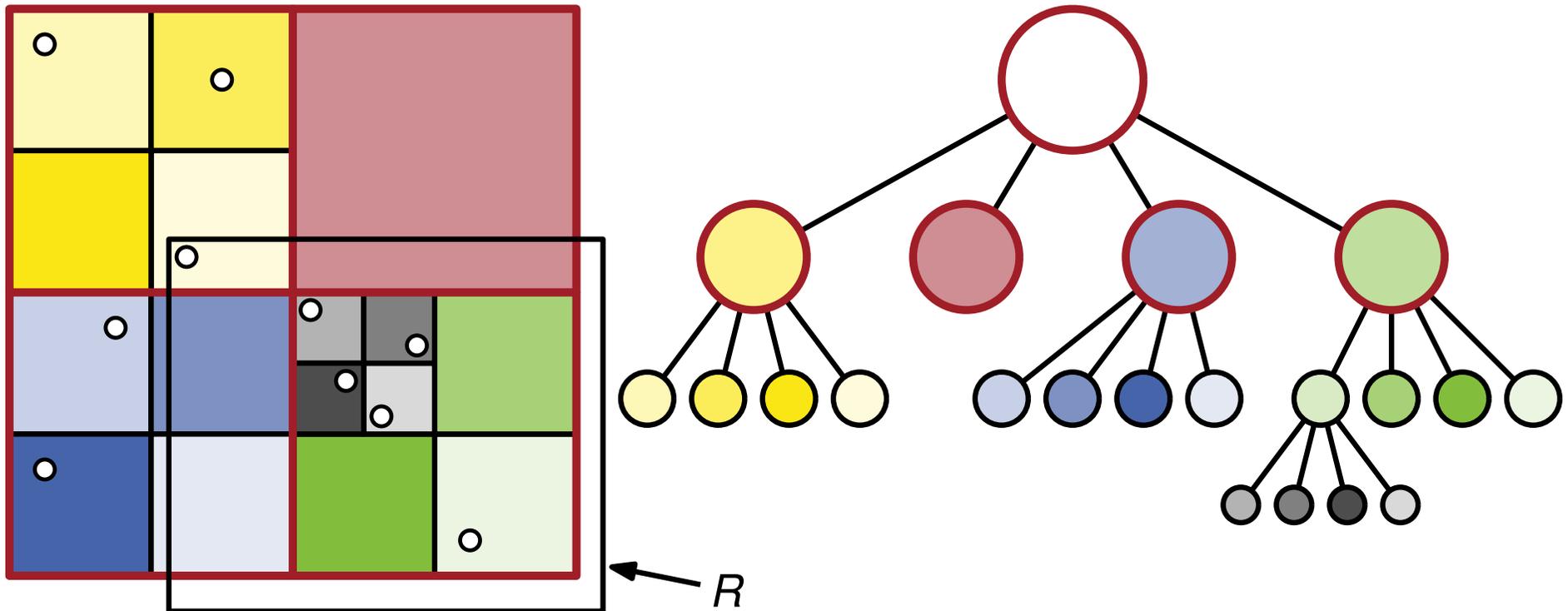


# Wiederholung: Bereichsanfrage



Starte mit der Wurzel

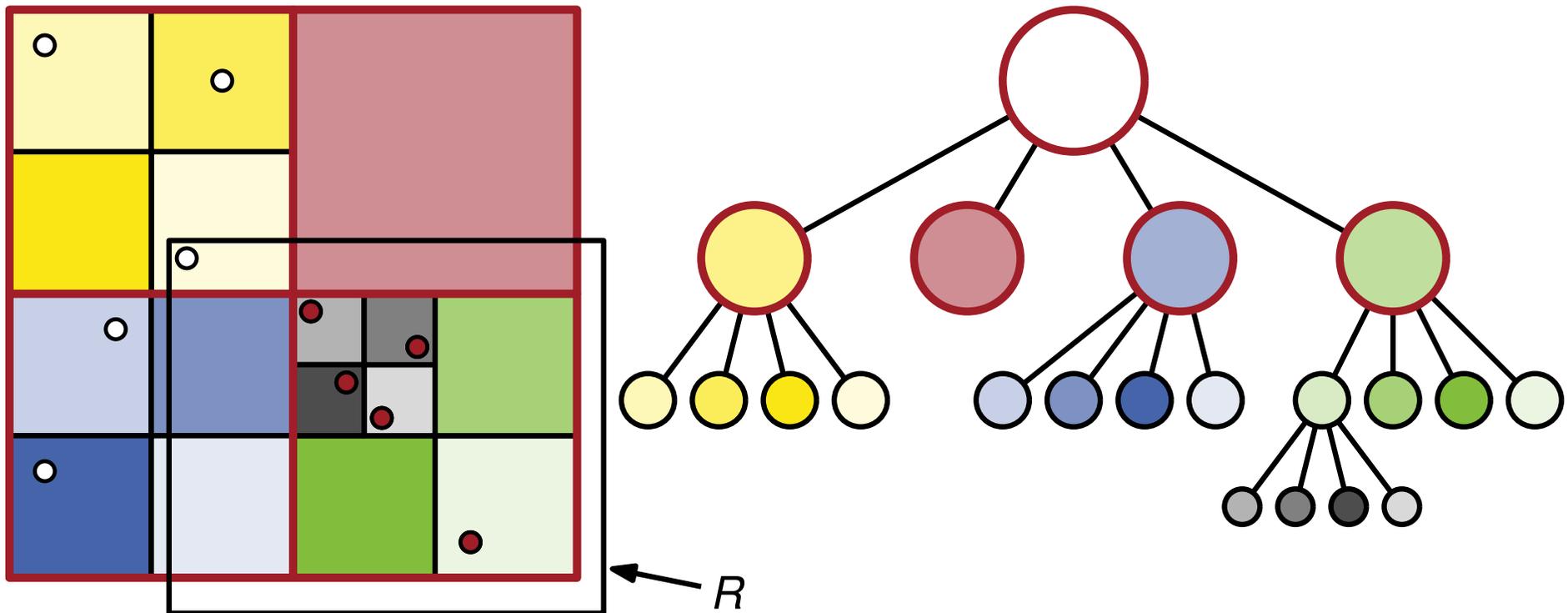
# Wiederholung: Bereichsanfrage



Starte mit der Wurzel

→ Rekursionsaufruf für alle vier Kinder

# Wiederholung: Bereichsanfrage

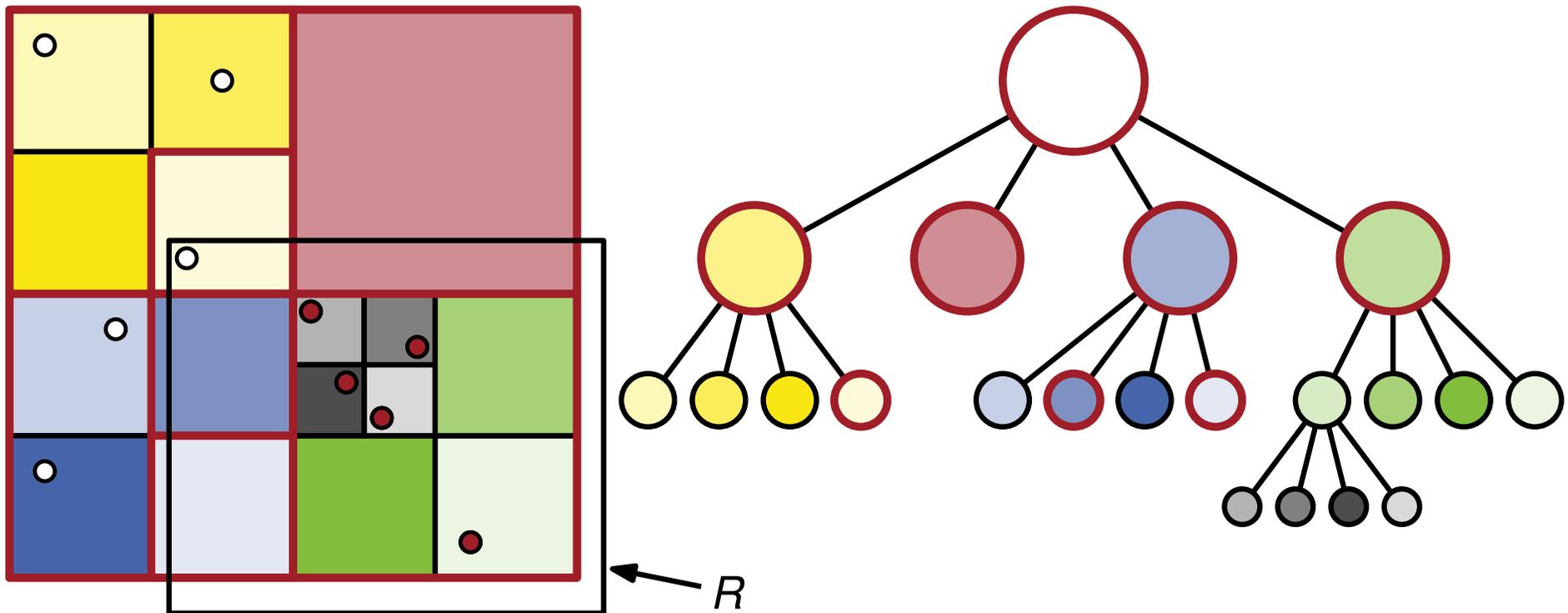


Starte mit der Wurzel

→ Rekursionsaufruf für alle vier Kinder

→ **Süd-Ost-Knoten:** vorzeitiger Abbruch und Ausgabe der enthaltenen Punkte

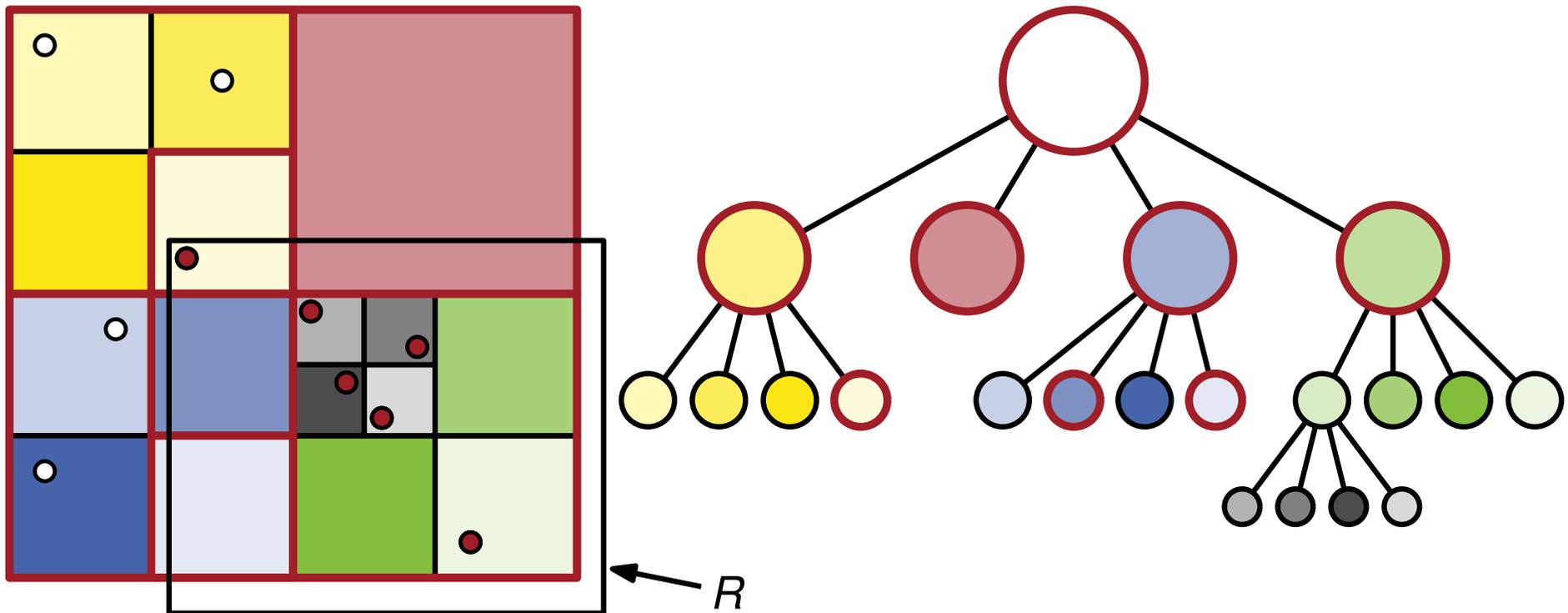
# Wiederholung: Bereichsanfrage



Starte mit der Wurzel

- Rekursionsaufruf für alle vier Kinder
- **Süd-Ost-Knoten:** vorzeitiger Abbruch und Ausgabe der enthaltenen Punkte
- **Süd-West-Knoten:** Rekursionsaufruf für zwei Kinder
- **Nord-West-Knoten:** Rekursionsaufruf für ein Kind

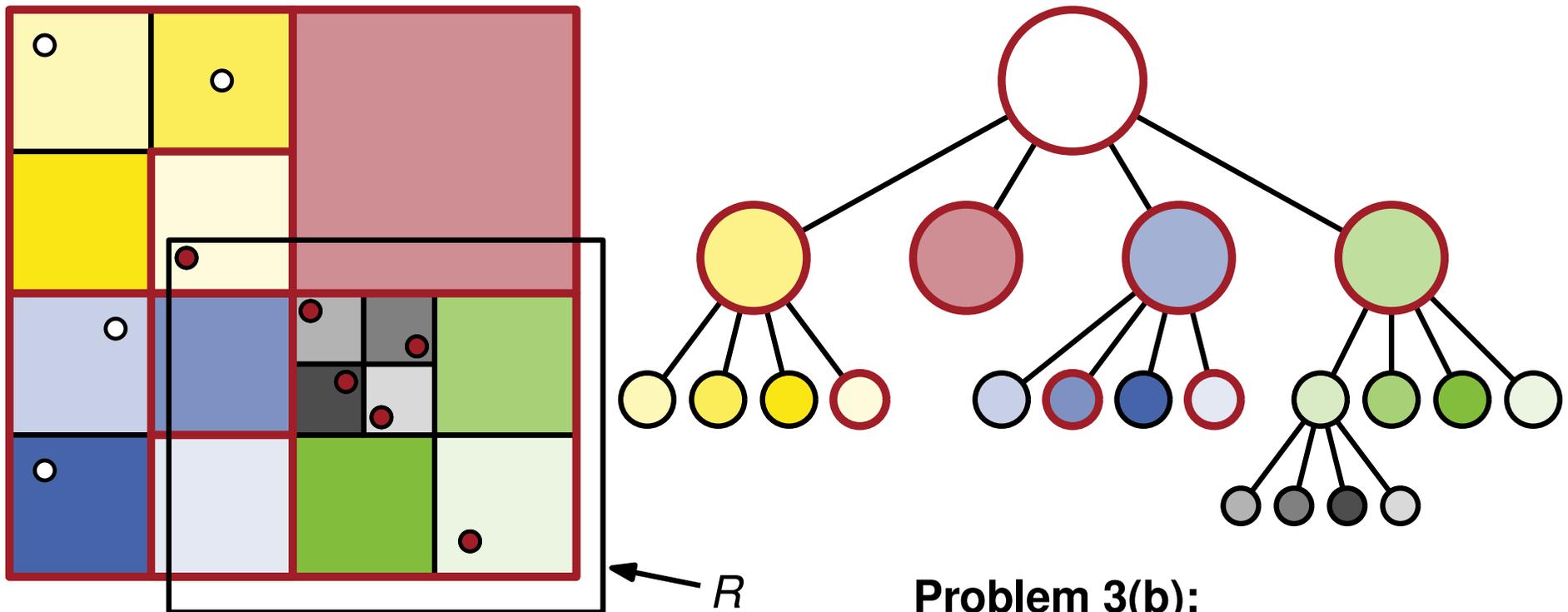
# Wiederholung: Bereichsanfrage



Starte mit der Wurzel

- Rekursionsaufruf für alle vier Kinder
- **Süd-Ost-Knoten:** vorzeitiger Abbruch und Ausgabe der enthaltenen Punkte
- **Süd-West-Knoten:** Rekursionsaufruf für zwei Kinder
- **Nord-West-Knoten:** Rekursionsaufruf für ein Kind
- Rekursionsende (Blätter erreicht): gib resultierende Knoten aus

# Wiederholung: Bereichsanfrage



Starte mit der Wurzel

- Rekursionsaufruf für alle vier Kinder
- **Süd-Ost-Knoten:** vorzeitiger Abbruch und Ausgabe der enthaltenen Punkte
- **Süd-West-Knoten:** Rekursionsaufruf für zwei Kinder
- **Nord-West-Knoten:** Rekursionsaufruf für ein Kind
- Rekursionsende (Blätter erreicht): gib resultierende Knoten aus

**Problem 3(b):  
Wie geht das schnell?**

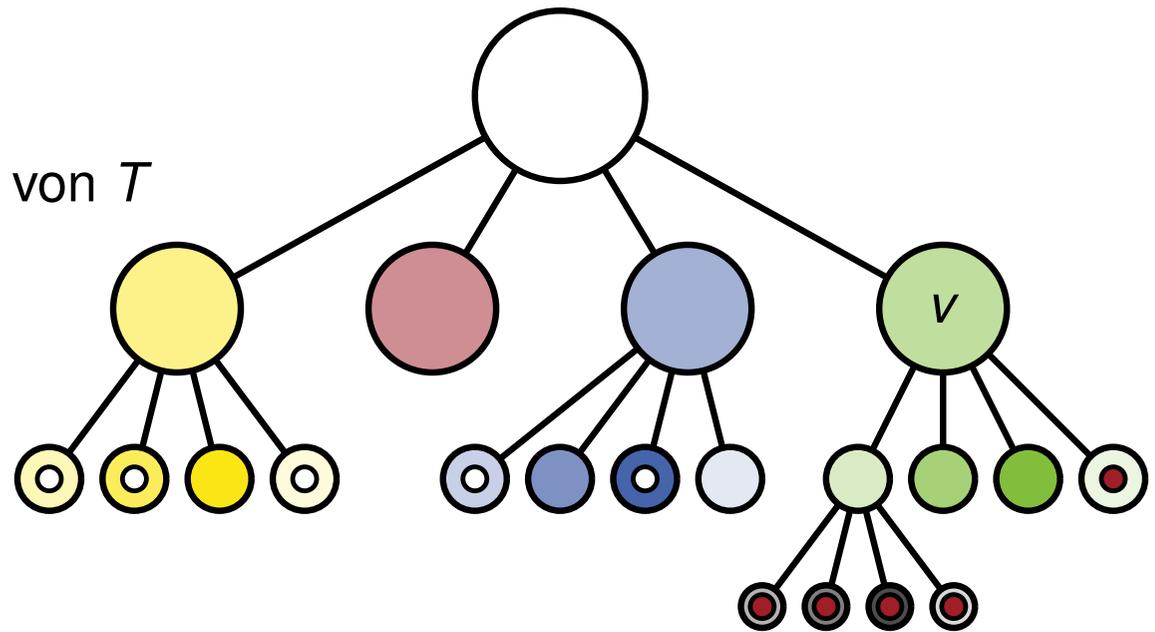


# Problem 3(b)

## Gegeben:

- gewurzelter Baum  $T$
- bis zu ein Punkt in jedem Blatt von  $T$
- ein Knoten  $v$  in  $T$

**Gesucht:** ■ Punkte aller  
Nachfolgern von  $v$

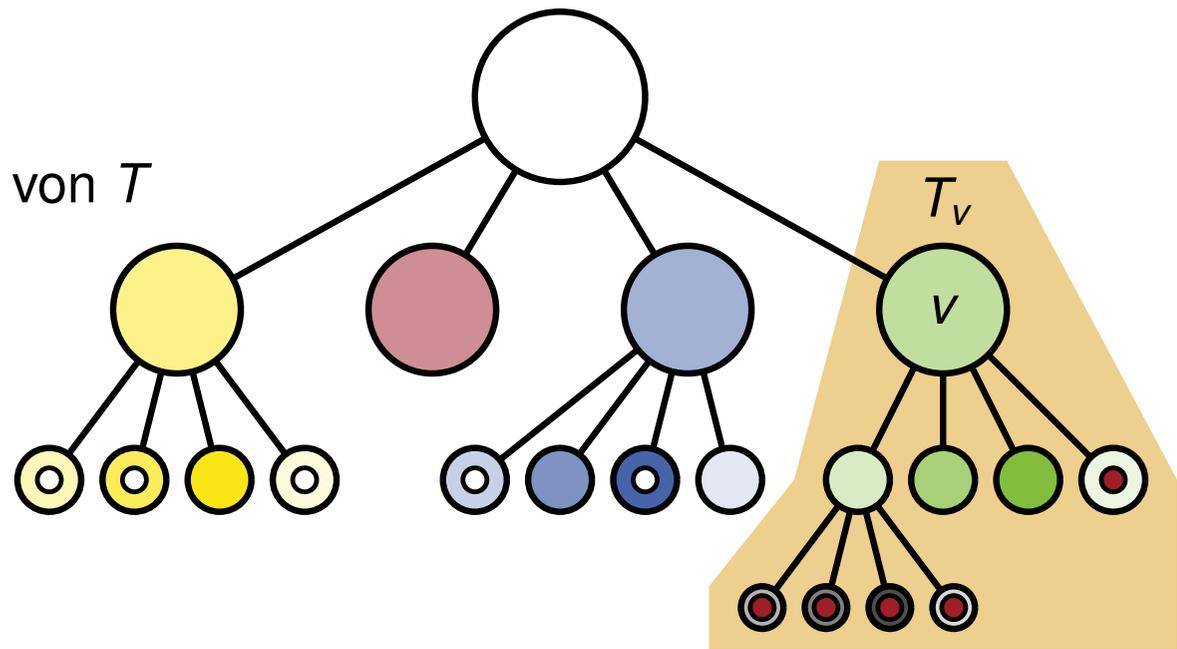


# Problem 3(b)

## Gegeben:

- gewurzelter Baum  $T$
- bis zu ein Punkt in jedem Blatt von  $T$
- ein Knoten  $v$  in  $T$

**Gesucht:** ■ Punkte aller Nachfolgern von  $v$



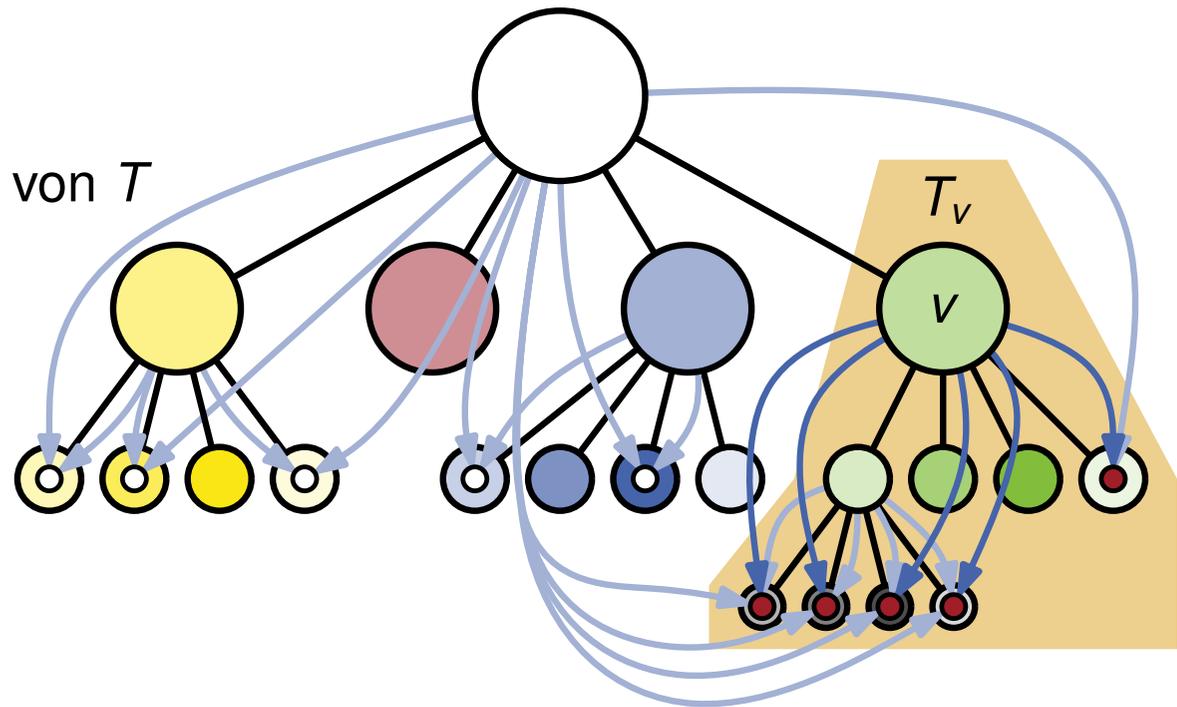
	Ansatz	Laufzeit	Speicher	Vorverarbeitungszeit
traversiere	Teilbaum $T_v$	$O( T_v )$	×	×

# Problem 3(b)

## Gegeben:

- gewurzelter Baum  $T$
- bis zu ein Punkt in jedem Blatt von  $T$
- ein Knoten  $v$  in  $T$

**Gesucht:** ■ Punkte aller Nachfolgern von  $v$



## Ansatz

traversiere Teilbaum  $T_v$   
speichere alle nachfolgenden Punkte

## Laufzeit

$$O(|T_v|)$$

## Speicher

×

## Vorverarbeitungszeit

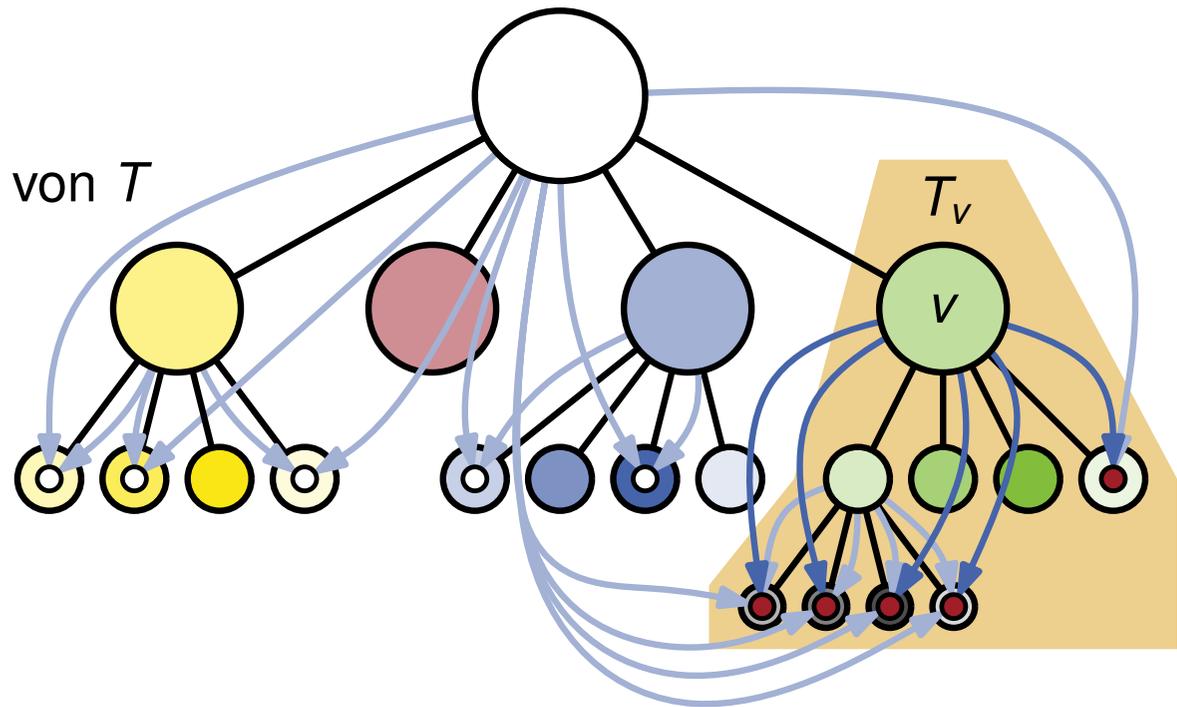
×

# Problem 3(b)

## Gegeben:

- gewurzelter Baum  $T$
- bis zu ein Punkt in jedem Blatt von  $T$
- ein Knoten  $v$  in  $T$

**Gesucht:** ■ Punkte aller Nachfolgern von  $v$



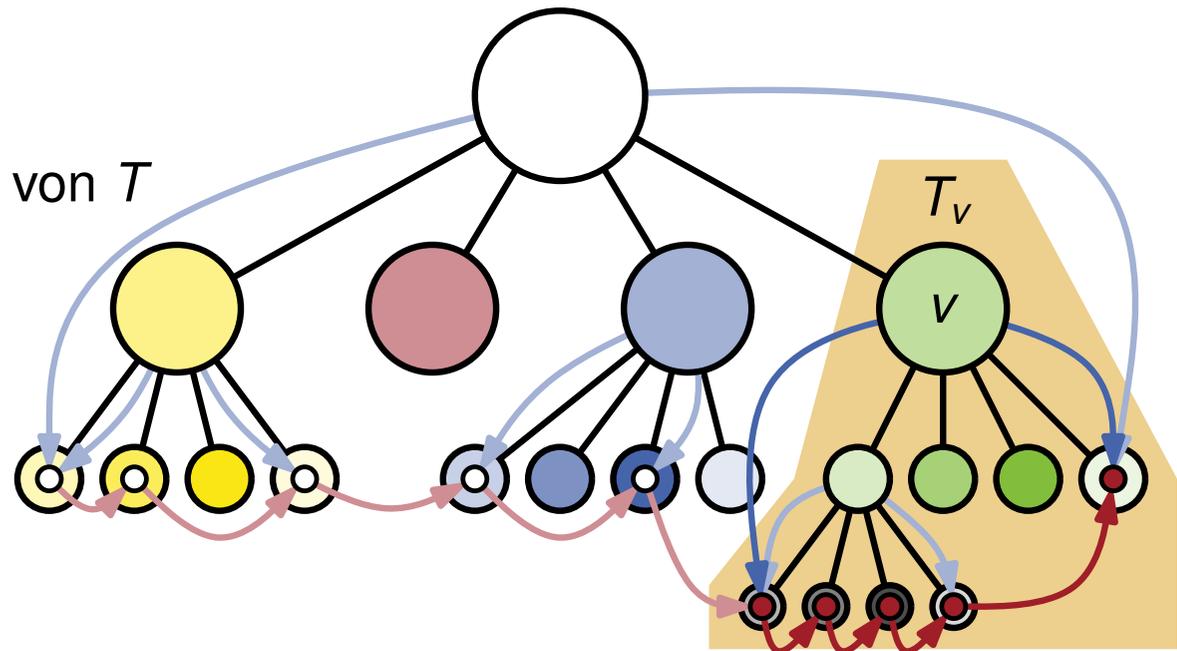
	<b>Ansatz</b>	<b>Laufzeit</b>	<b>Speicher</b>	<b>Vorverarbeitungszeit</b>
traversiere	Teilbaum $T_v$	$O( T_v )$	×	×
speichere alle nachfolgenden Punkte		$O(k)$ ( $k$ = Anzahl gefundener Punkte)	$O(d \cdot  P )$	$O(d \cdot  P )$

# Problem 3(b)

## Gegeben:

- gewurzelter Baum  $T$
- bis zu ein Punkt in jedem Blatt von  $T$
- ein Knoten  $v$  in  $T$

**Gesucht:** ■ Punkte aller Nachfolgern von  $v$



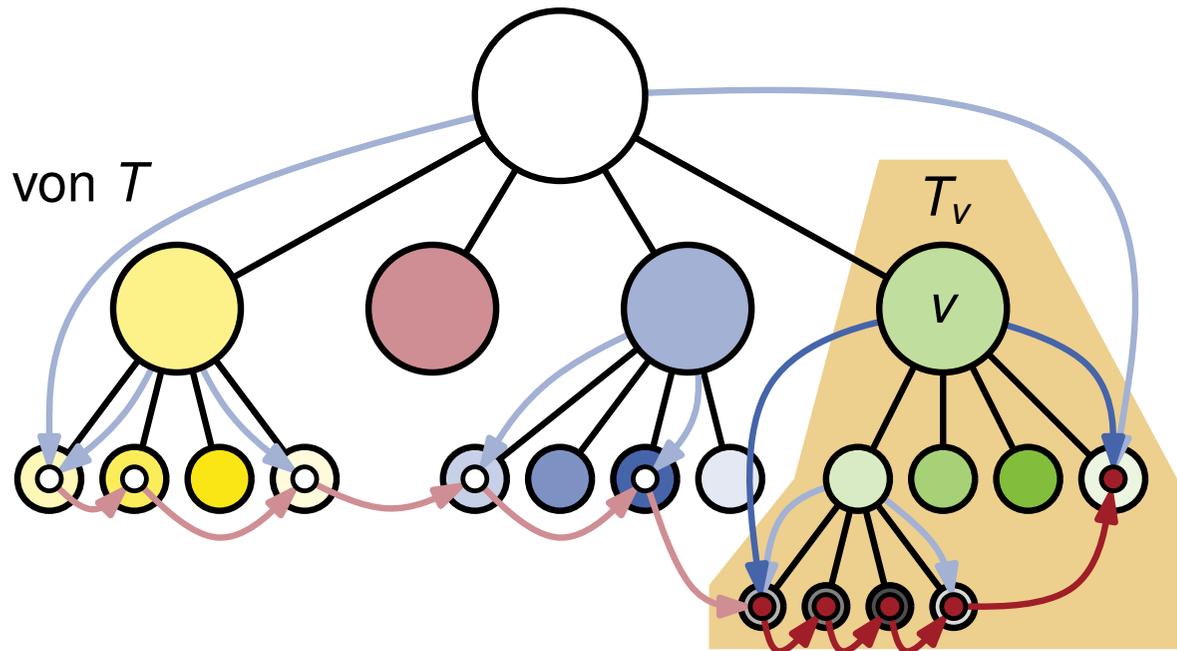
	<b>Ansatz</b>	<b>Laufzeit</b>	<b>Speicher</b>	<b>Vorverarbeitungszeit</b>
traversiere	Teilbaum $T_v$	$O( T_v )$	×	×
speichere	alle nachfolgenden Punkte	$O(k)$ ( $k$ = Anzahl gefundener Punkte)	$O(d \cdot  P )$	$O(d \cdot  P )$
speichere	Start und Ende in Punktliste			

# Problem 3(b)

## Gegeben:

- gewurzelter Baum  $T$
- bis zu ein Punkt in jedem Blatt von  $T$
- ein Knoten  $v$  in  $T$

**Gesucht:** ■ Punkte aller Nachfolgern von  $v$



	<b>Ansatz</b>	<b>Laufzeit</b>	<b>Speicher</b>	<b>Vorverarbeitungszeit</b>
traversiere	Teilbaum $T_v$	$O( T_v )$	×	×
speichere alle nachfolgenden Punkte		$O(k)$ ( $k$ = Anzahl gefundener Punkte)	$O(d \cdot  P )$	$O(d \cdot  P )$
speichere Start und Ende in Punktliste		$O(k)$	$O( T )$	$O( T )$

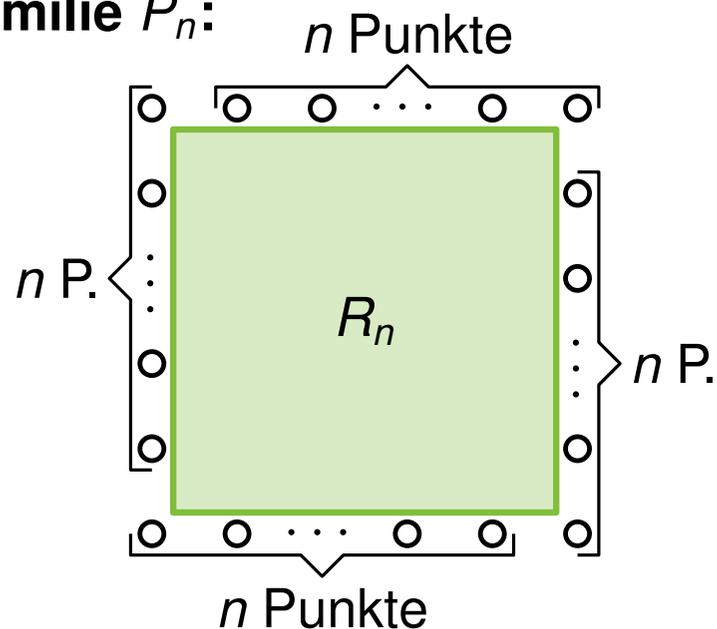
# Problem 3(c)

- (c) Geben Sie eine Familie von Punktfolgen  $P_1, P_2, P_3, \dots$  mit  $|P_n| \in \Theta(n)$ , sowie Rechtecke  $R_1, R_2, R_3, \dots$  an, sodass  $R_n$  keinen Punkt von  $P_n$  enthält, die Bereichsanfrage mit  $R_n$  aber den gesamten Quadtree  $T(P_n)$  traversiert.

# Problem 3(c)

(c) Geben Sie eine Familie von Punktmenge  $P_1, P_2, P_3, \dots$  mit  $|P_n| \in \Theta(n)$ , sowie Rechtecke  $R_1, R_2, R_3, \dots$  an, sodass  $R_n$  keinen Punkt von  $P_n$  enthält, die Bereichsanfrage mit  $R_n$  aber den gesamten Quadtree  $T(P_n)$  traversiert.

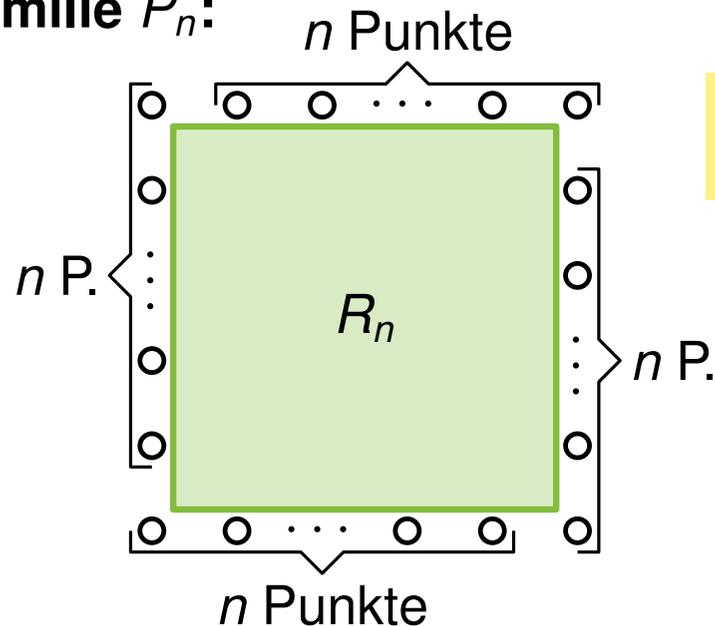
**Familie  $P_n$ :**



# Problem 3(c)

(c) Geben Sie eine Familie von Punktmenge  $P_1, P_2, P_3, \dots$  mit  $|P_n| \in \Theta(n)$ , sowie Rechtecke  $R_1, R_2, R_3, \dots$  an, sodass  $R_n$  keinen Punkt von  $P_n$  enthält, die Bereichsanfrage mit  $R_n$  aber den gesamten Quadtree  $T(P_n)$  traversiert.

**Familie  $P_n$ :**

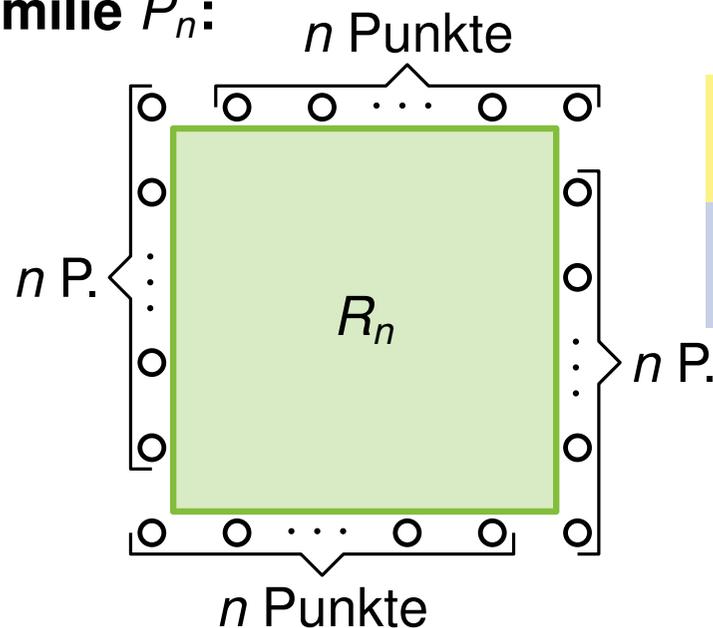


- $P_n$  enthält  $4n \in \Theta(n)$  Punkte

# Problem 3(c)

(c) Geben Sie eine Familie von Punktmenge  $P_1, P_2, P_3, \dots$  mit  $|P_n| \in \Theta(n)$ , sowie Rechtecke  $R_1, R_2, R_3, \dots$  an, sodass  $R_n$  keinen Punkt von  $P_n$  enthält, die Bereichsanfrage mit  $R_n$  aber den gesamten Quadtree  $T(P_n)$  traversiert.

**Familie  $P_n$ :**



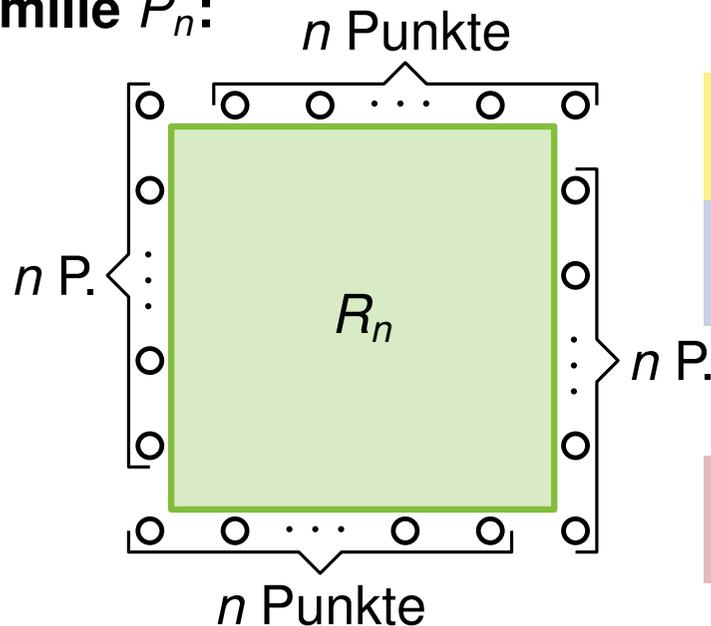
- $P_n$  enthält  $4n \in \Theta(n)$  Punkte

- $R_n$  enthält keinen Punkt von  $P_n$

# Problem 3(c)

(c) Geben Sie eine Familie von Punktmenge  $P_1, P_2, P_3, \dots$  mit  $|P_n| \in \Theta(n)$ , sowie Rechtecke  $R_1, R_2, R_3, \dots$  an, sodass  $R_n$  keinen Punkt von  $P_n$  enthält, die Bereichsanfrage mit  $R_n$  aber den gesamten Quadtree  $T(P_n)$  traversiert.

**Familie  $P_n$ :**



- $P_n$  enthält  $4n \in \Theta(n)$  Punkte

- $R_n$  enthält keinen Punkt von  $P_n$

**Zu zeigen:**

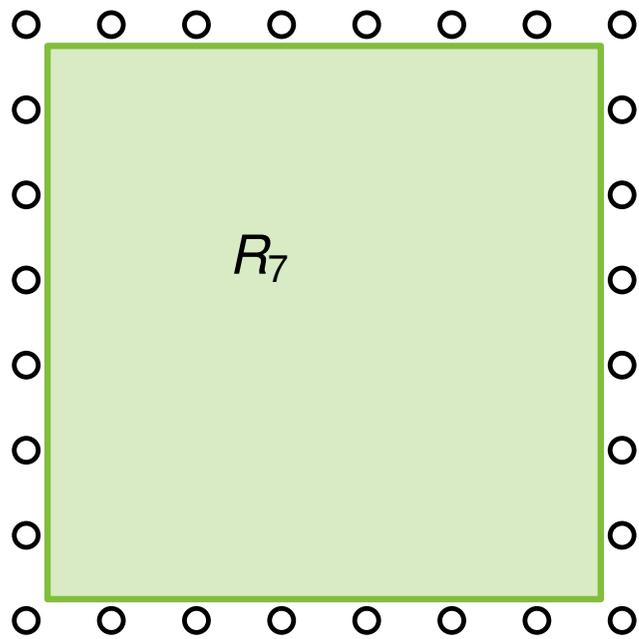
- der gesamte Quadtree  $T(P_n)$  wird traversiert.

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$

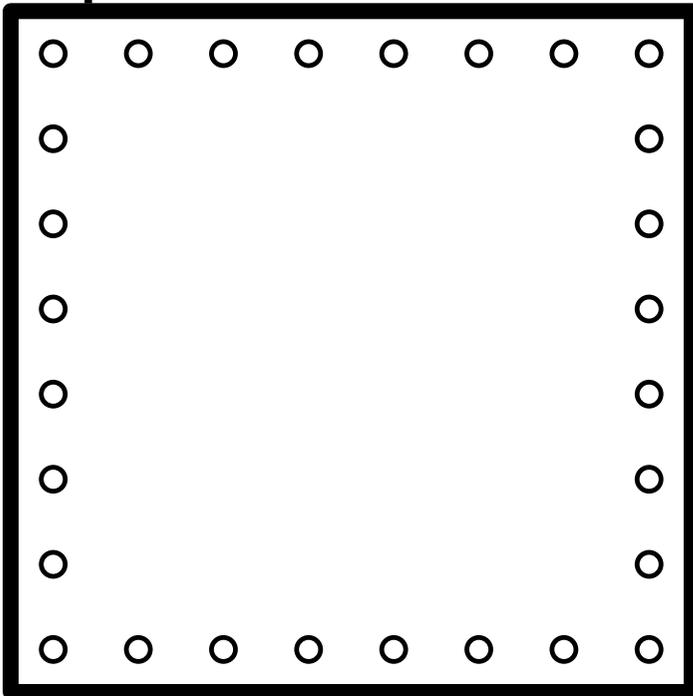


# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

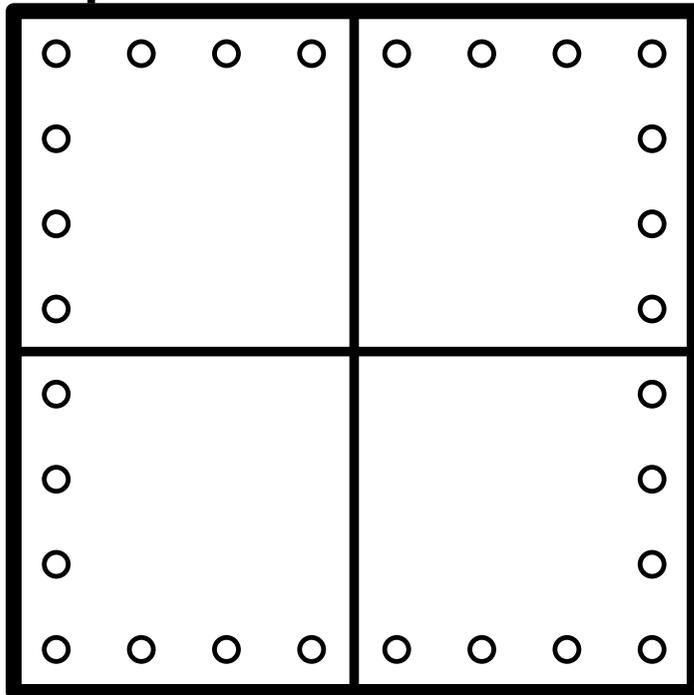
- starte mit Wurzelquadrat

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

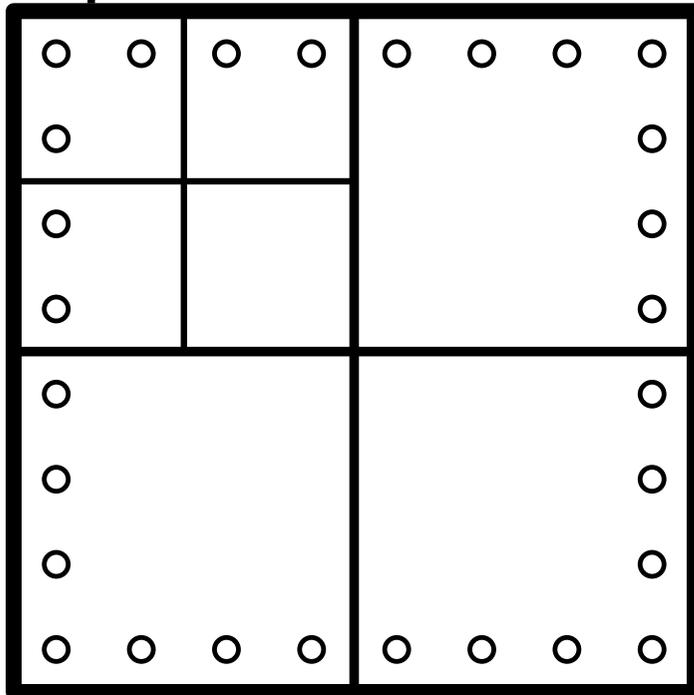
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadrees

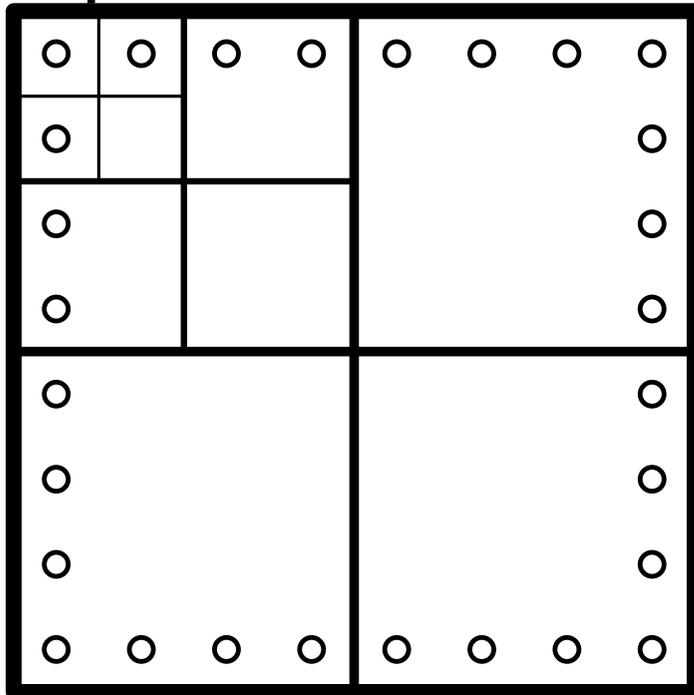
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

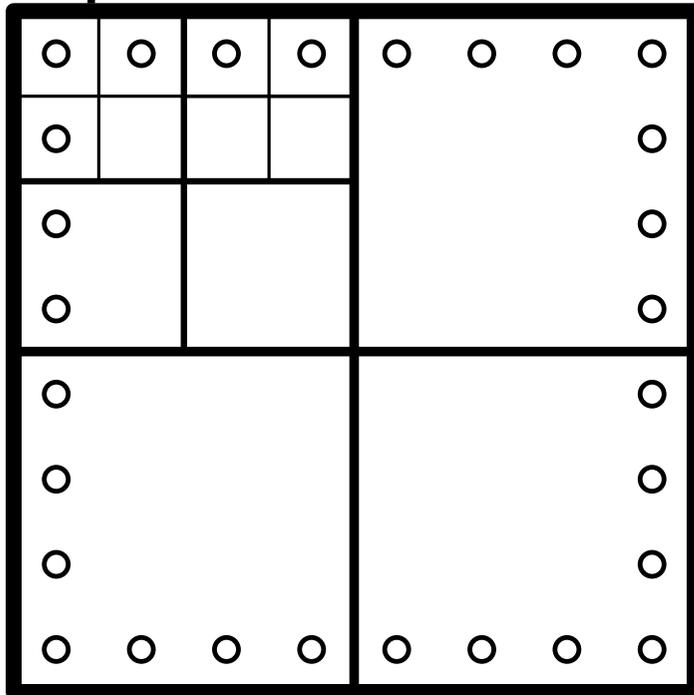
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

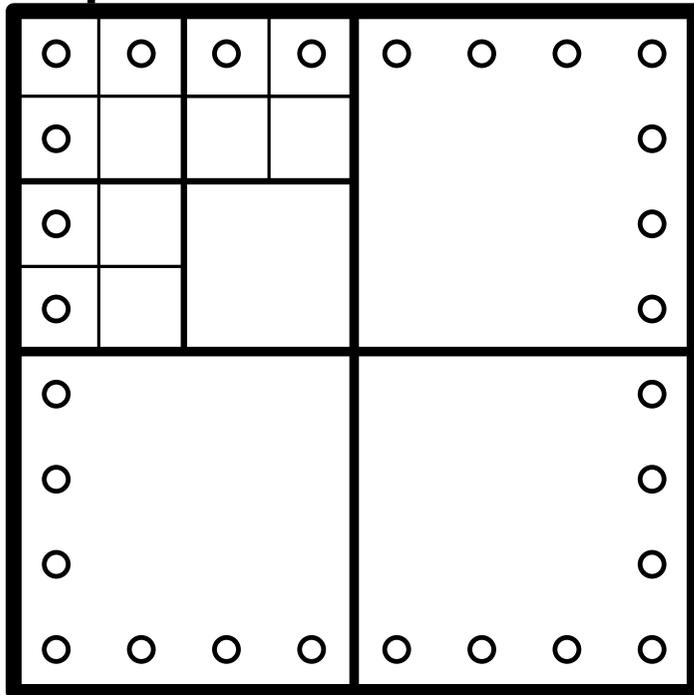
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

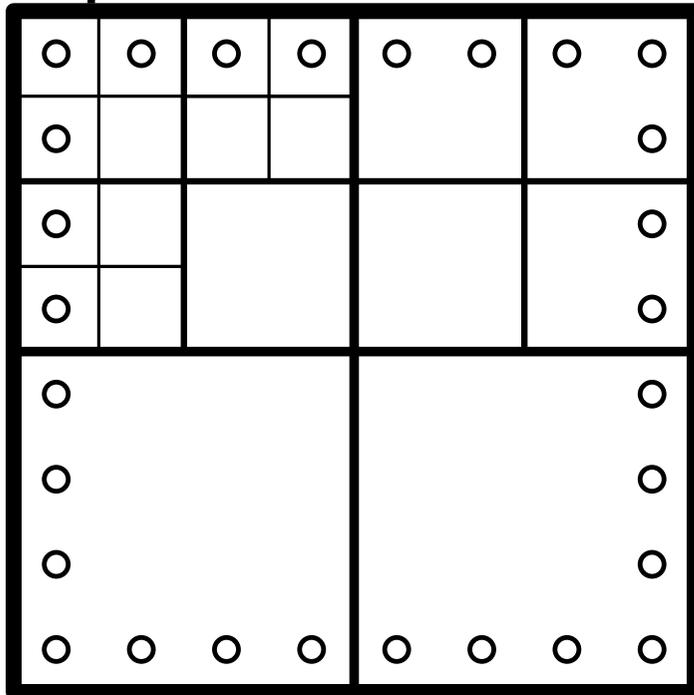
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

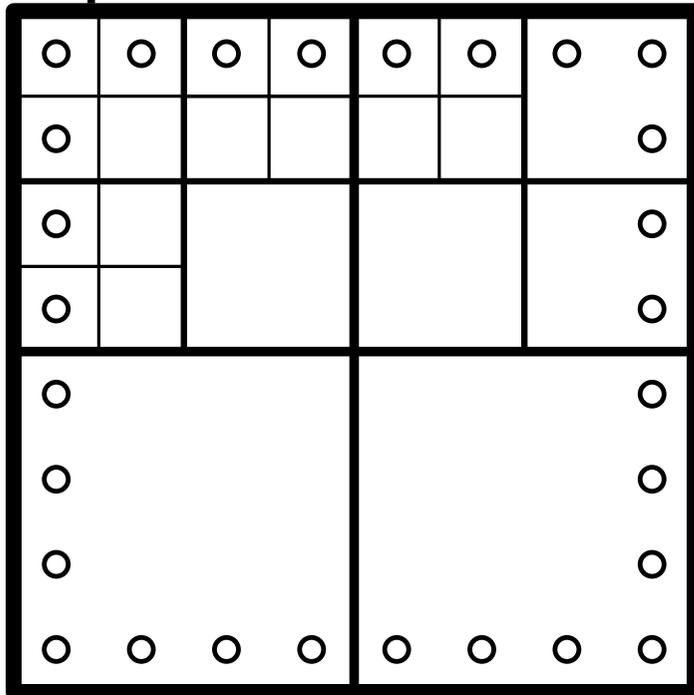
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadrees

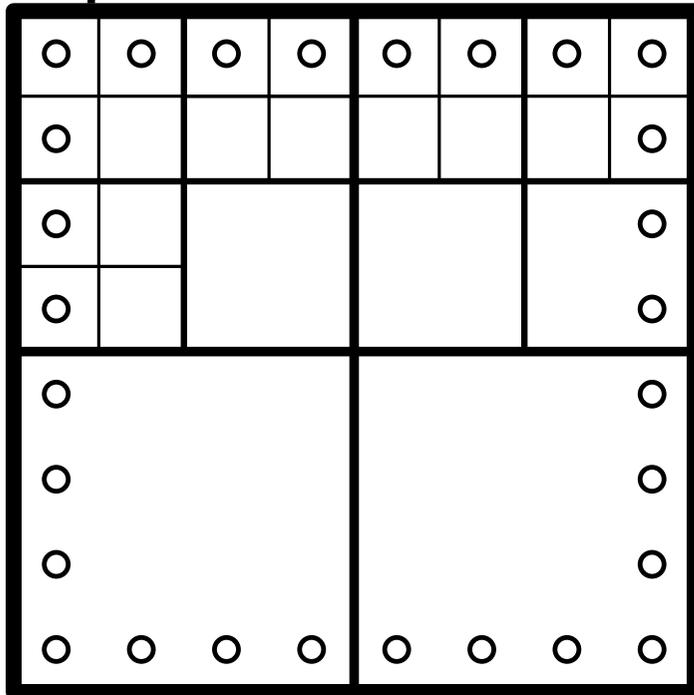
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

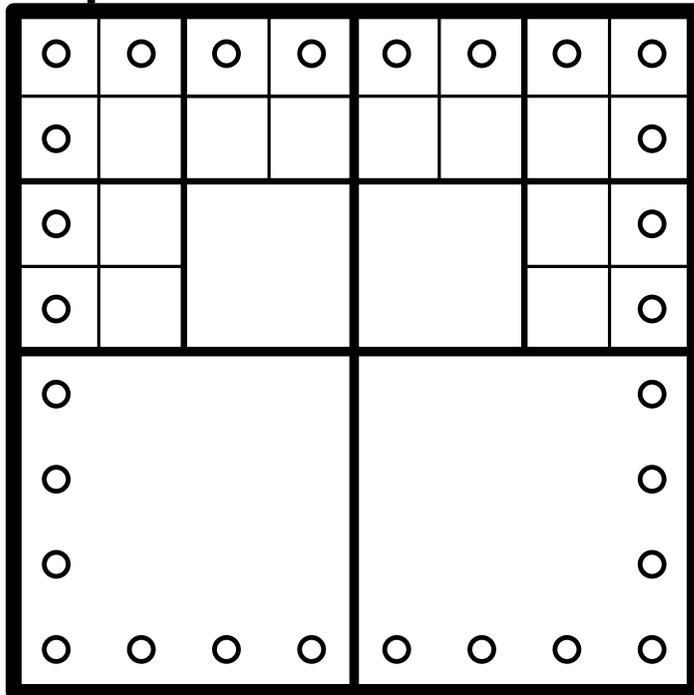
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

**Zu zeigen:**

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

**Beispiel:  $n = 7$**



## Aufbau des Quadrees

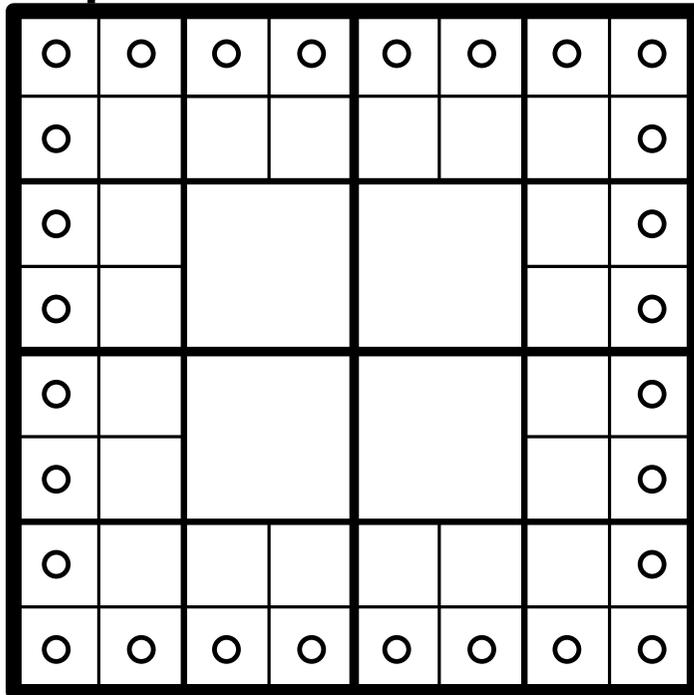
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

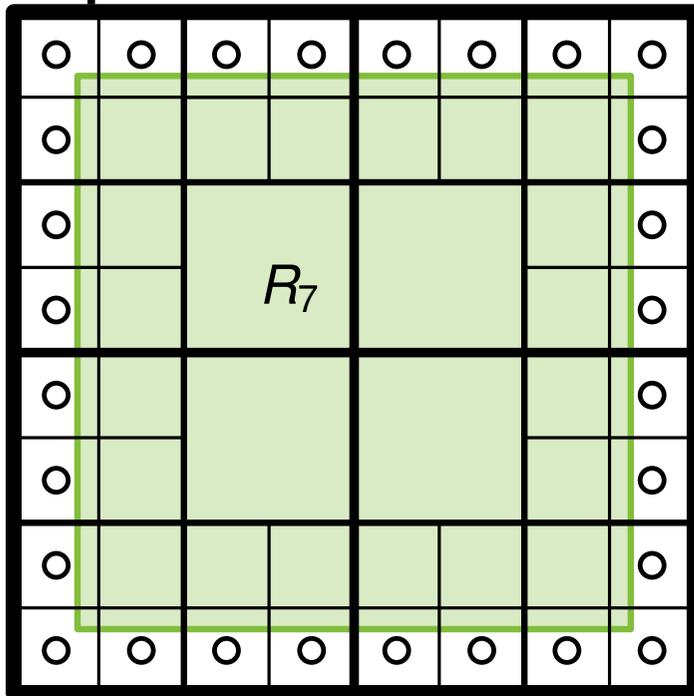
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

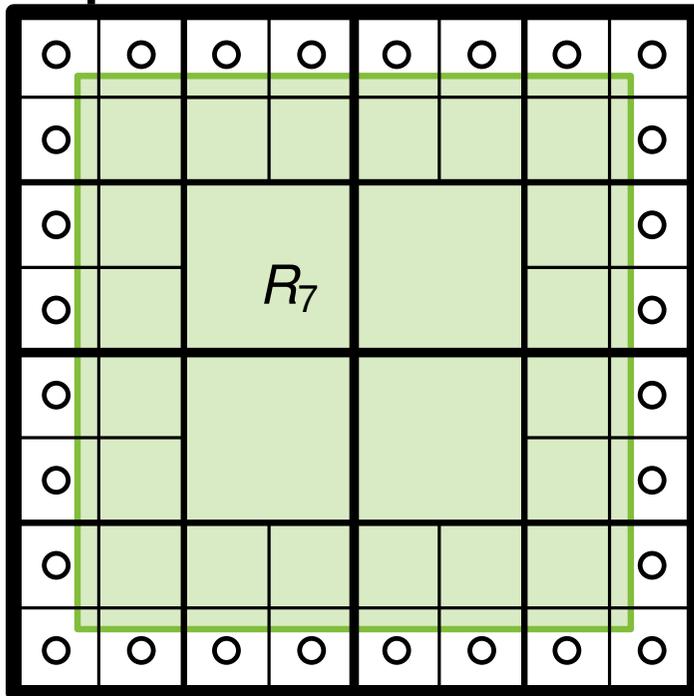
- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadrees

- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

## Bereichsanfrage

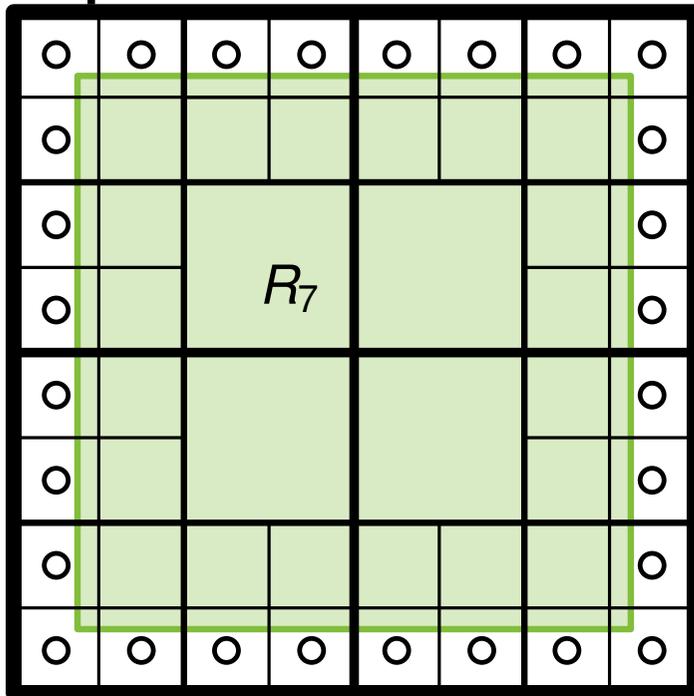
Wird für Knoten  $v_Q$  abgebrochen, wenn ...

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadrees

- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

## Bereichsanfrage

Wird für Knoten  $v_Q$  abgebrochen, wenn ...

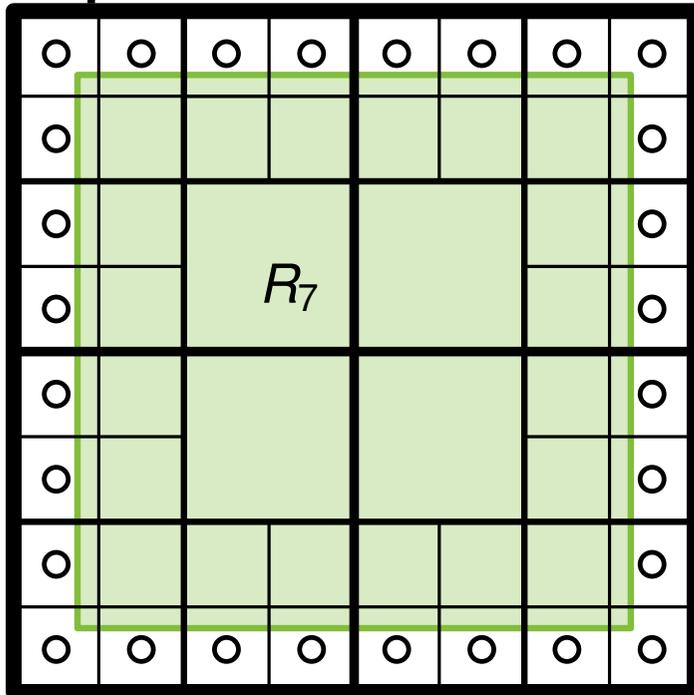
- $Q$  liegt vollständig außerhalb von  $R_7$ .
- $Q$  liegt vollständig innerhalb von  $R_7$ .

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtree

- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

## Bereichsanfrage

Wird für Knoten  $v_Q$  abgebrochen, wenn ...

- $Q$  liegt vollständig außerhalb von  $R_7$ .
- $Q$  liegt vollständig innerhalb von  $R_7$ .

tritt nur auf, wenn  $v_Q$  Blatt ist

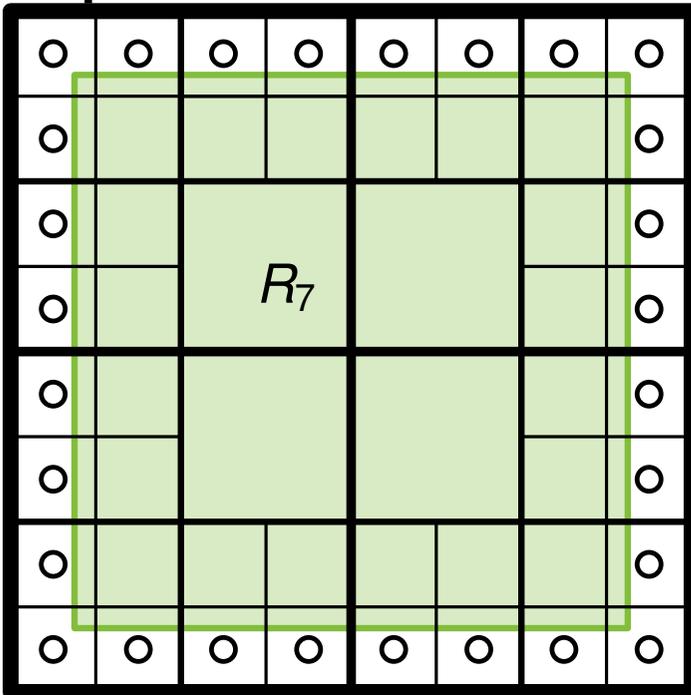
kommt nicht vor

# Problem 3(c)

## Zu zeigen:

- der gesamte Quadtree  $T(P_n)$  wird traversiert.

## Beispiel: $n = 7$



## Aufbau des Quadtrees

- starte mit Wurzelquadrat
- zerlege Quadrate rekursiv, bis sie nur noch einen Knoten enthalten

## Bereichsanfrage

Wird für Knoten  $v_Q$  abgebrochen, wenn ...

- $Q$  liegt vollständig außerhalb von  $R_7$ .
- $Q$  liegt vollständig innerhalb von  $R_7$ .

tritt nur auf, wenn  $v_Q$  Blatt ist

kommt nicht vor

⇒ Der rekursive Abstieg in  $T(P_n)$  wird erst in den Blättern beendet.

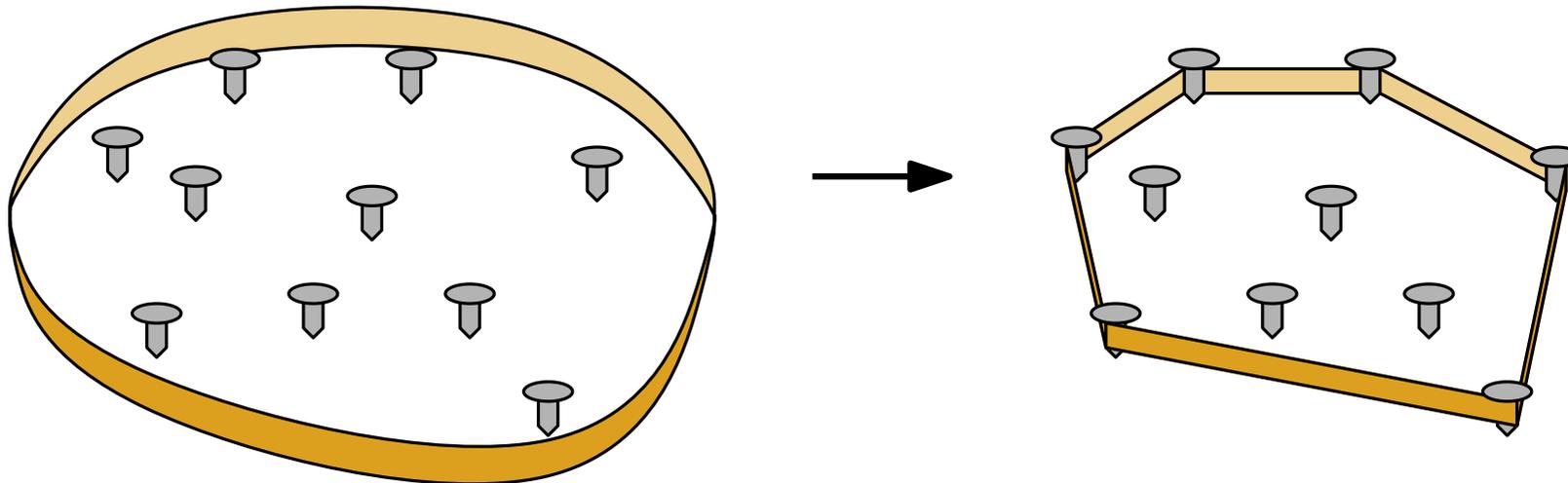
⇒ Der gesamte Quadtree  $T(P_n)$  wird traversiert. (gilt offensichtlich genauso für andere  $n$ )

# Konvexe Hülle

## Problem: Konvexe Hülle

Gegeben sind  $n \geq 3$  Punkte  $Q = \{p_0, \dots, p_{n-1}\}$ . Berechne die *konvexe Hülle*  $H(Q)$ , d.h. das minimale konvexe Polygon, das alle Punkte in  $Q$  im Inneren enthält.

Intuition: Fasse jeden Punkte als Nagel auf, ziehe ein Gummiband über alle Nägel und lasse es los.



## Aus der Vorlesung bekannt:

- Zwei Algorithmen mit Laufzeit  $O(n \log n)$  bzw.  $O(hn)$ , wobei  $h$  die Anzahl der Eckpunkte der konvexen Hülle ist.
- Die Berechnung der konvexen Hülle liegt in  $\Theta(n \log n)$ .

# Problem 4(a)

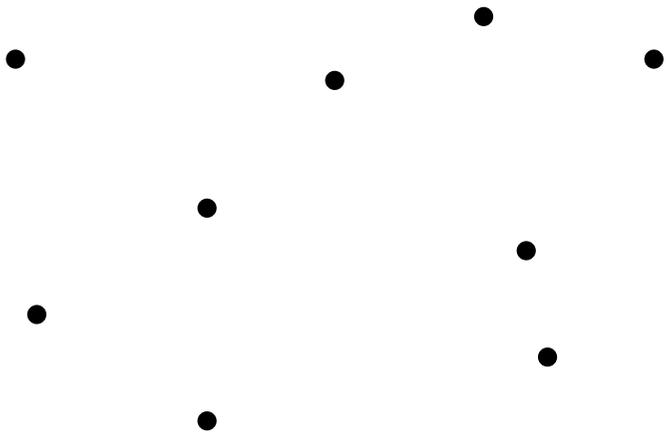
- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



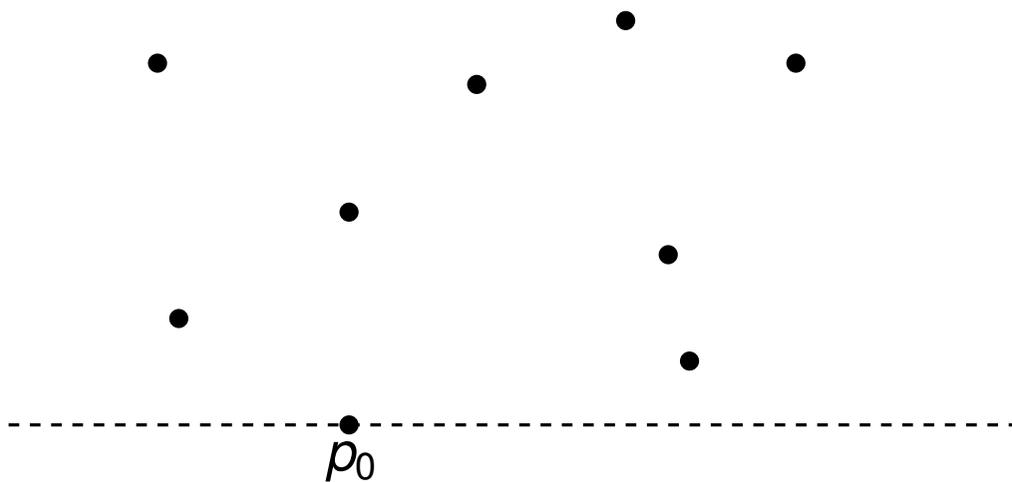
Zwischenergebnis für die konvexe Hülle:

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

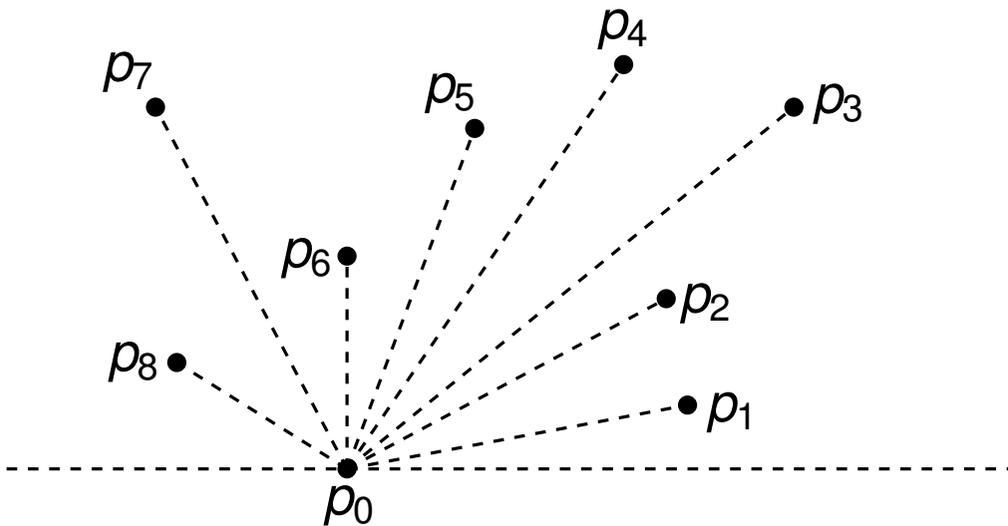
$p_0$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

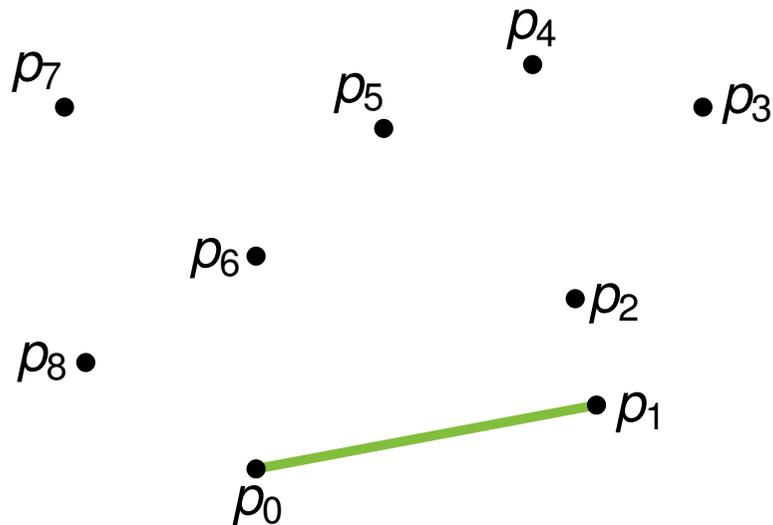
$p_0$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

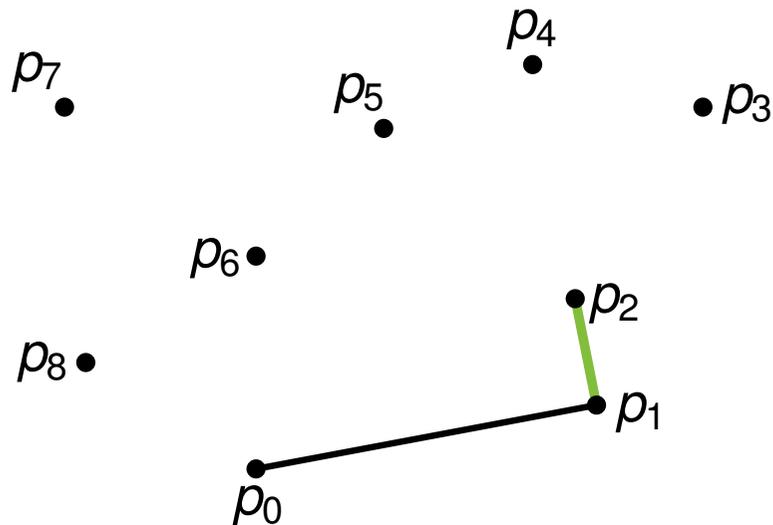
$p_0$   $p_1$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

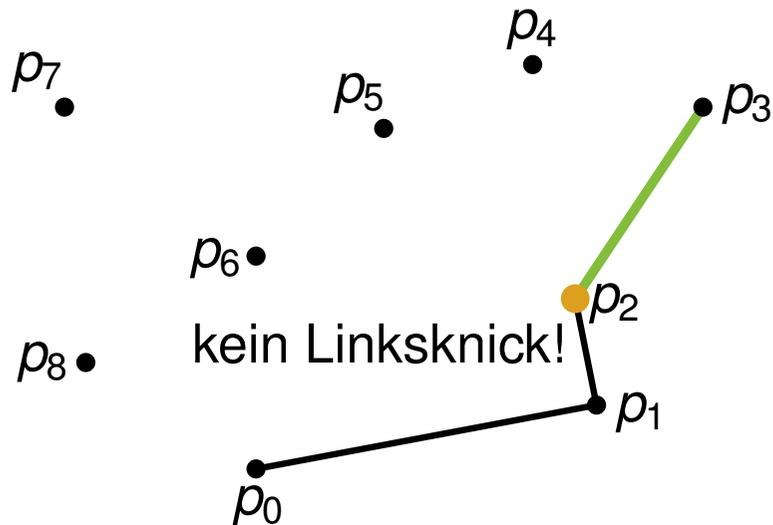
$p_0 \ p_1 \ p_2$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

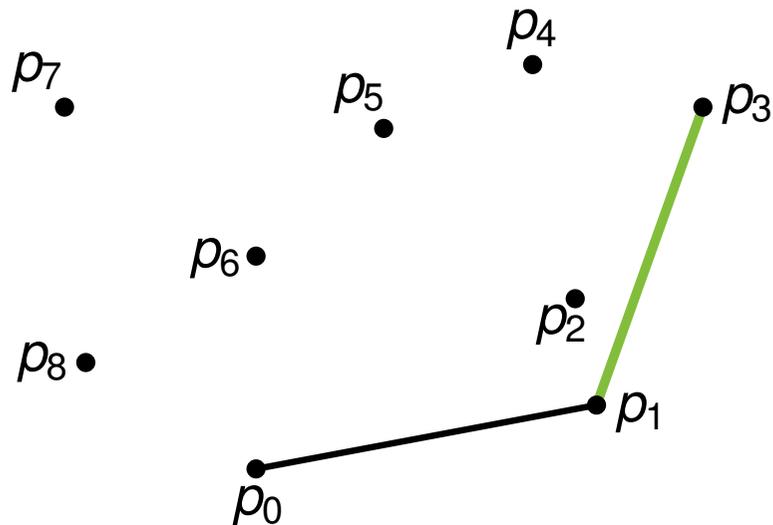
$p_0 \ p_1 \ p_2 \ p_3$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

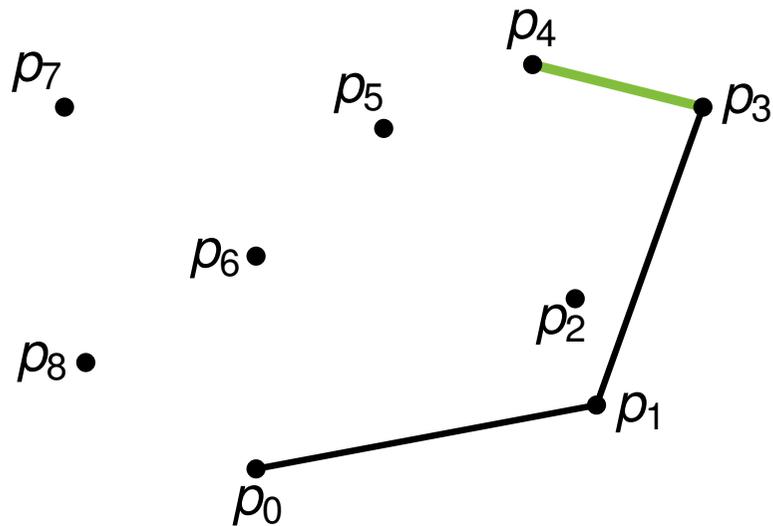
$p_0 \ p_1 \ p_3$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

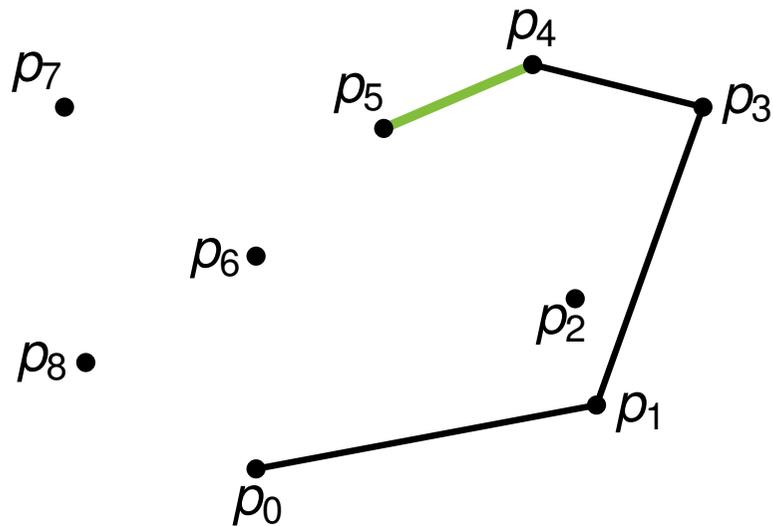
$p_0$   $p_1$   $p_3$   $p_4$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

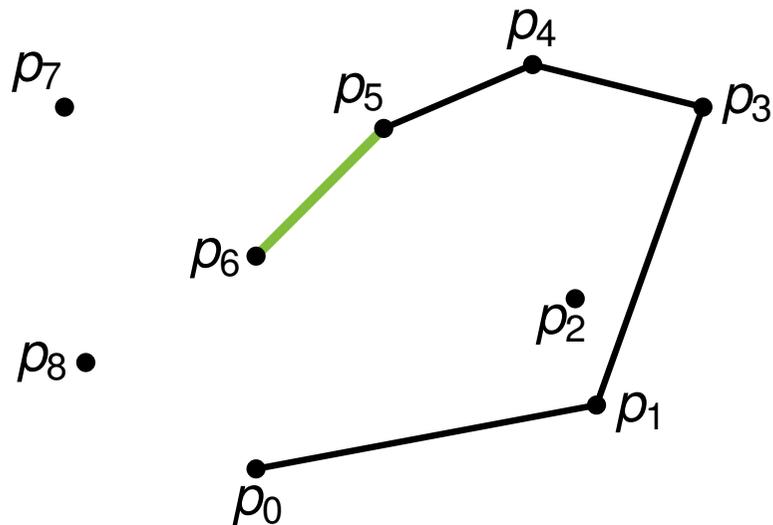
$p_0 \ p_1 \ p_3 \ p_4 \ p_5$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

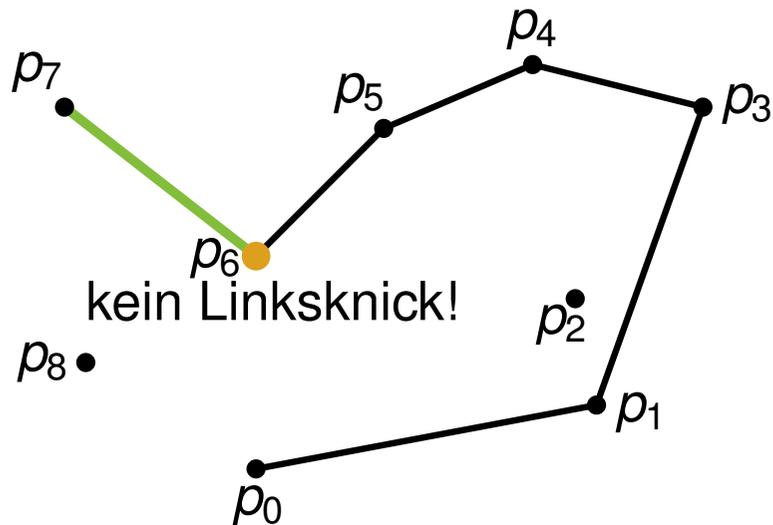
$p_0 \ p_1 \ p_3 \ p_4 \ p_5 \ p_6$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

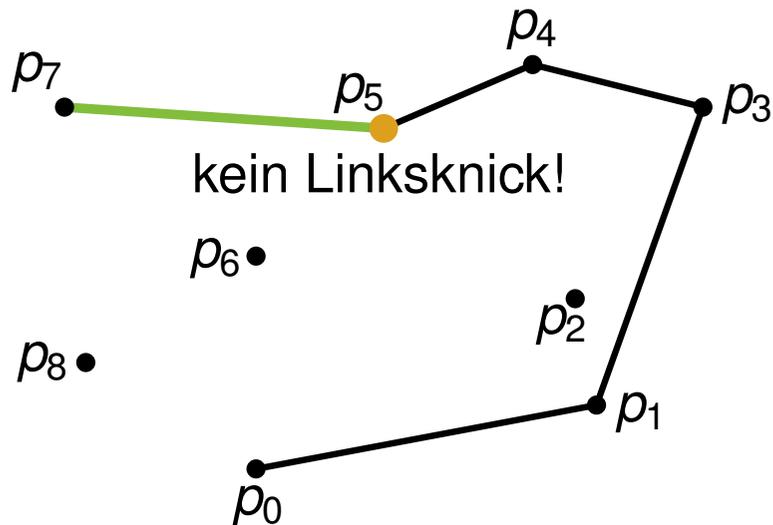
$p_0 \ p_1 \ p_3 \ p_4 \ p_5 \ p_6 \ p_7$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

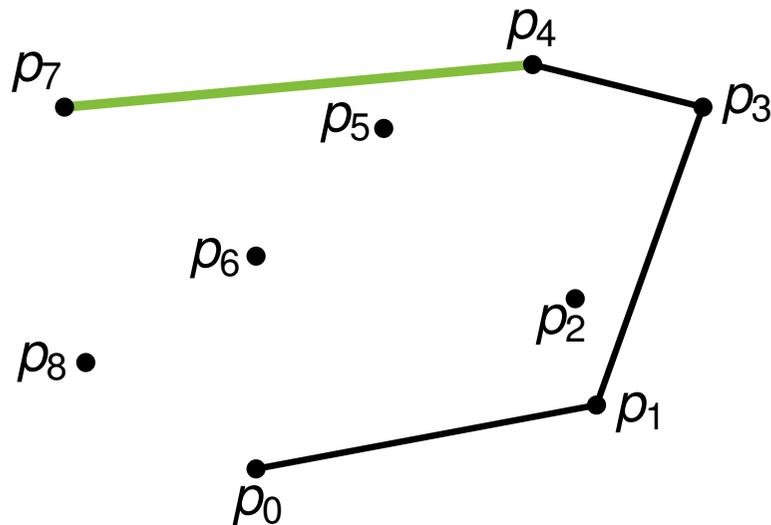
$p_0 \ p_1 \ p_3 \ p_4 \ p_5 \ p_7$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

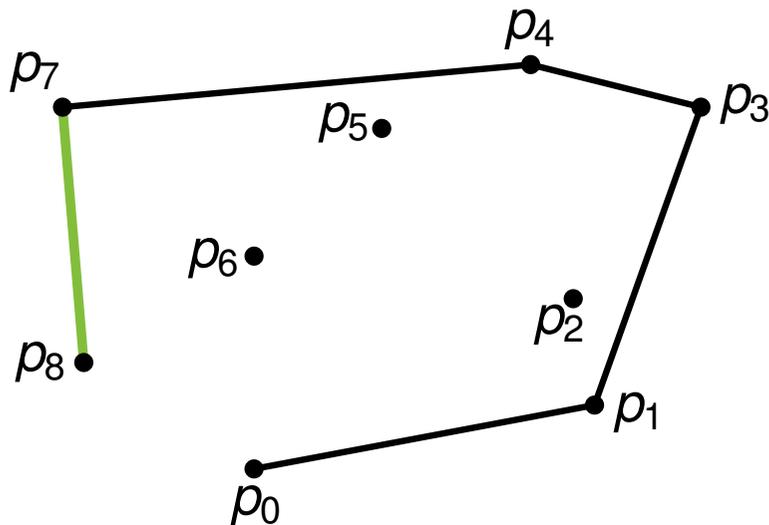
$p_0 \ p_1 \ p_3 \ p_4 \ p_7$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

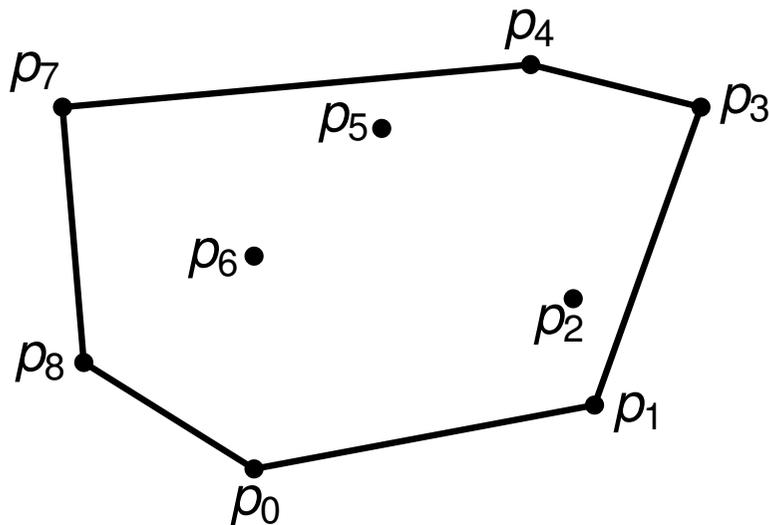
$p_0 \ p_1 \ p_3 \ p_4 \ p_7 \ p_8$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

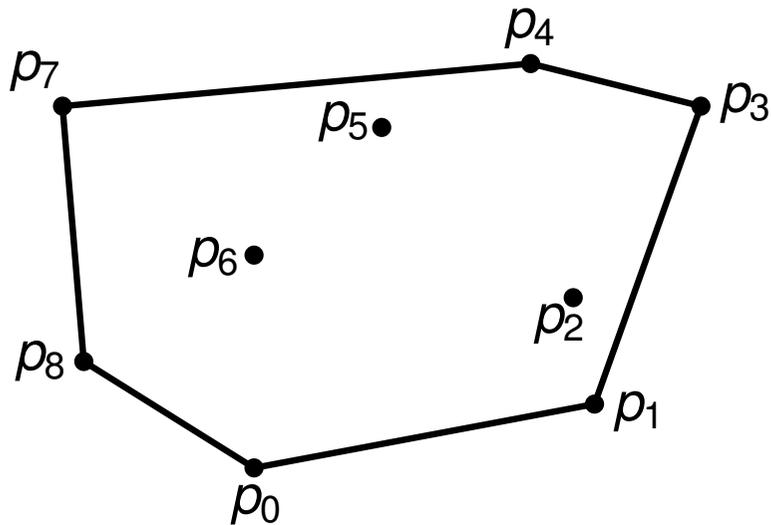
$p_0 \ p_1 \ p_3 \ p_4 \ p_7 \ p_8$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

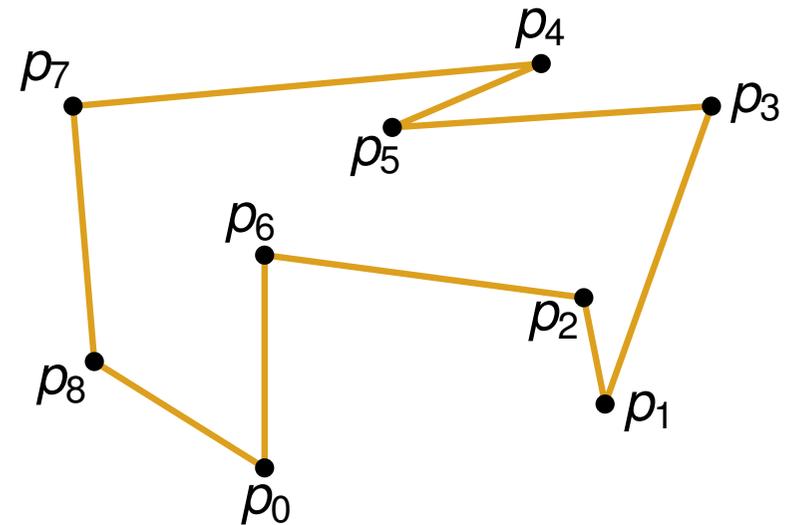
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 \ p_1 \ p_3 \ p_4 \ p_7 \ p_8$

**Beispiel – Graham Scan für Polygon:**



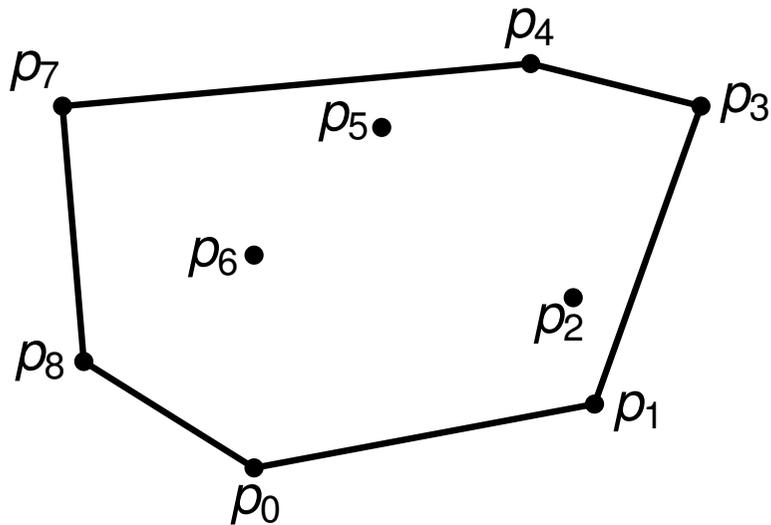
Zwischenergebnis für die konvexe Hülle:

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

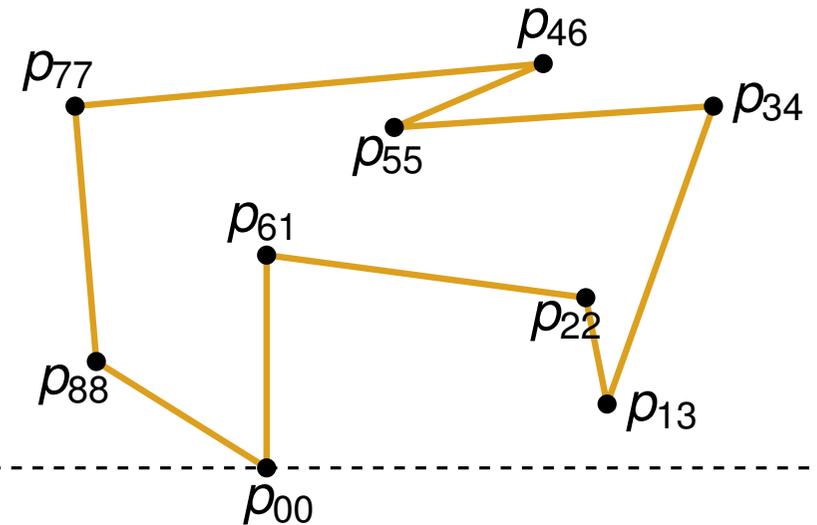
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0$   $p_1$   $p_3$   $p_4$   $p_7$   $p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

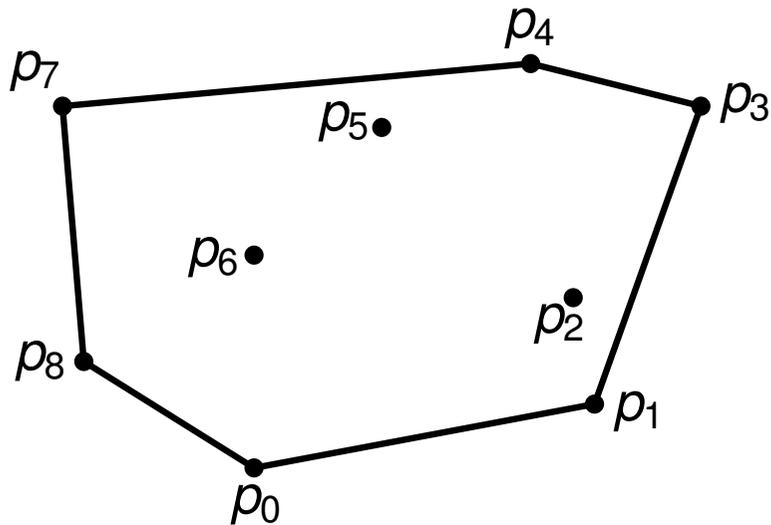
$p_{00}$

# Problem 4(a)

(a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

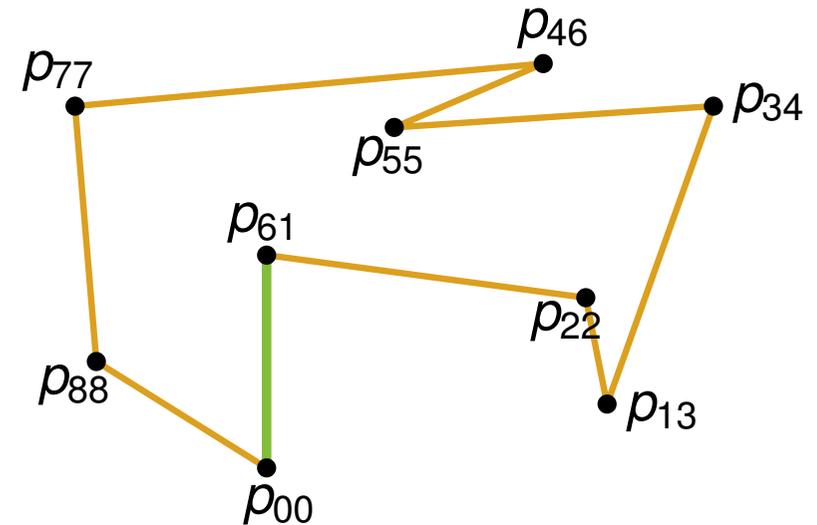
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

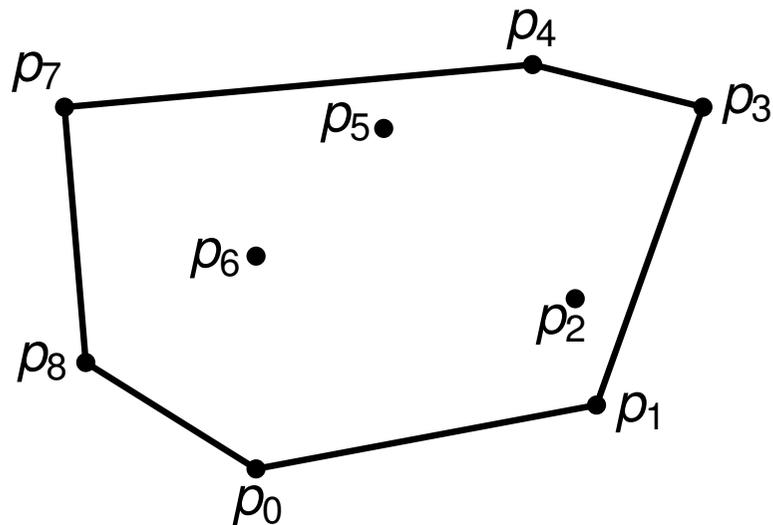
$p_{00} p_{61}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

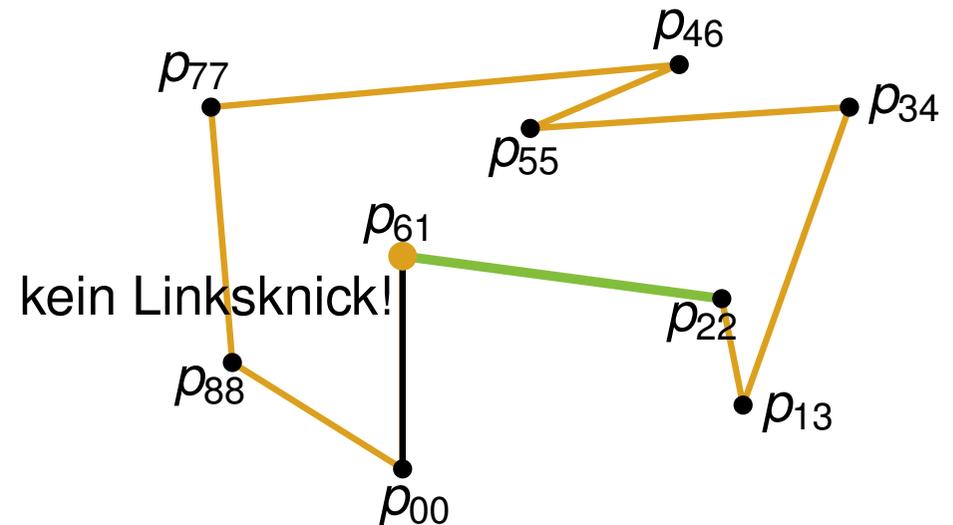
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

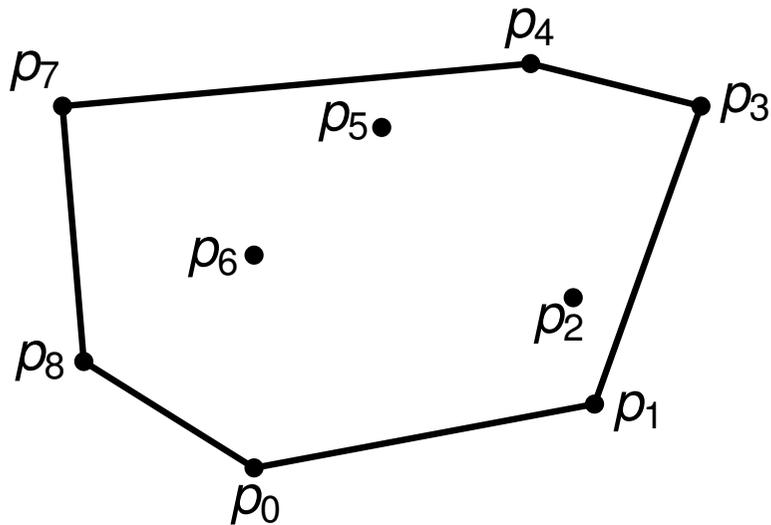
$p_{00} p_{61} p_{22}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

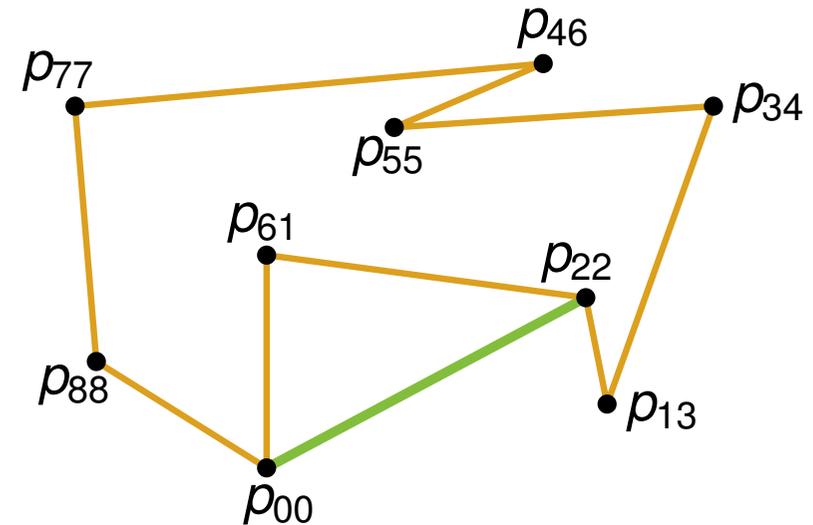
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

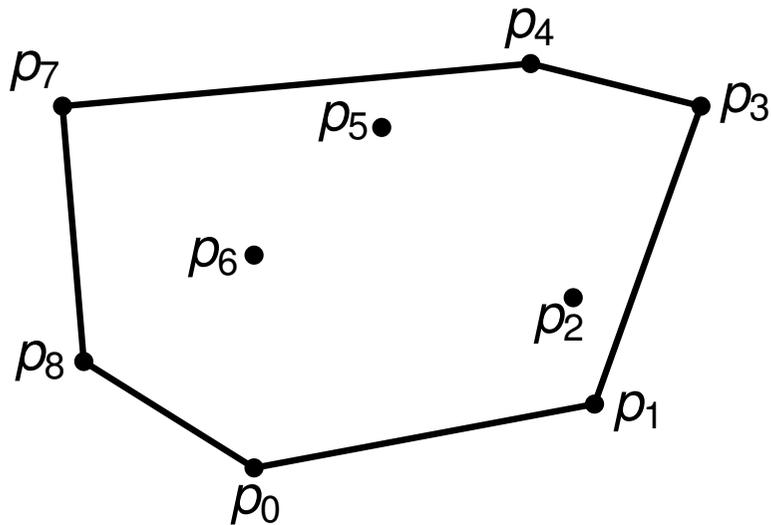
$p_{00} p_{22}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

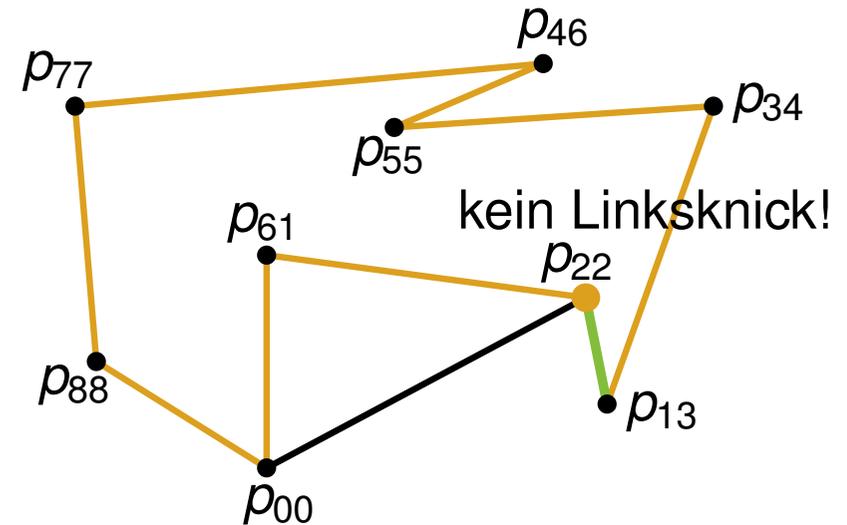
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

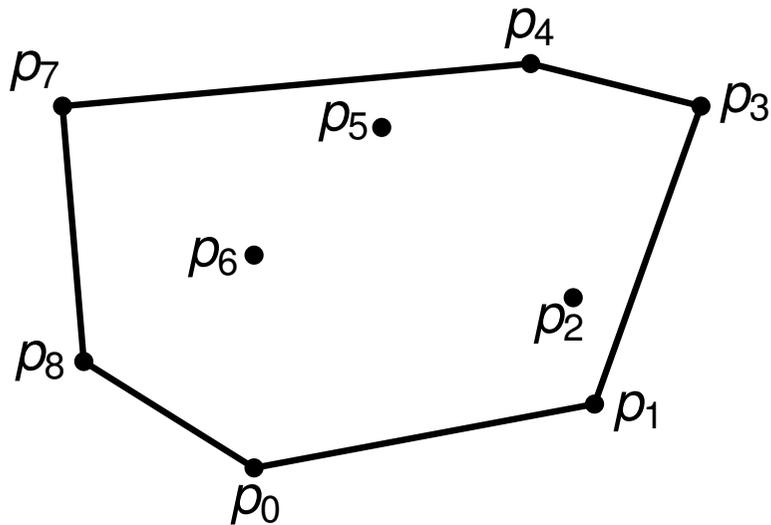
$p_{00} p_{22} p_{13}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

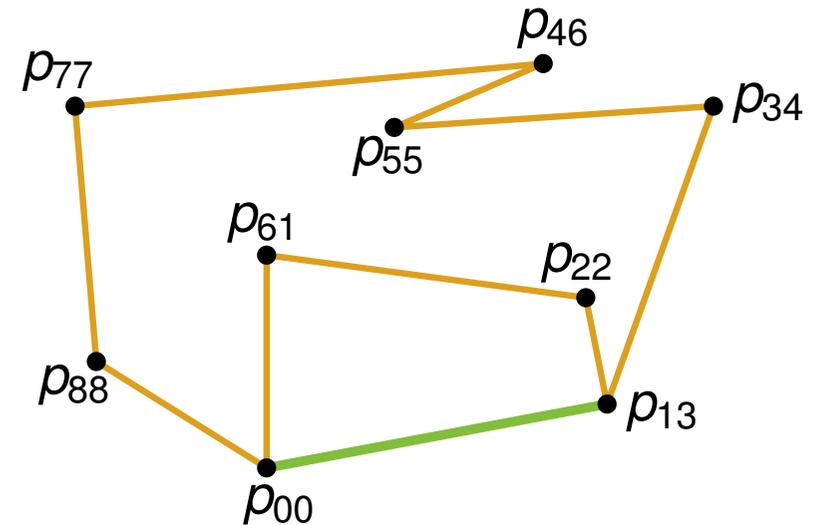
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

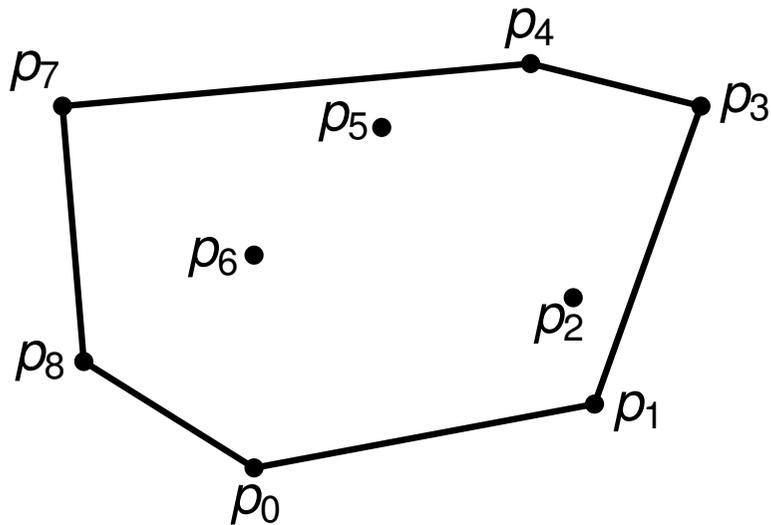
$p_{00} p_{13}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

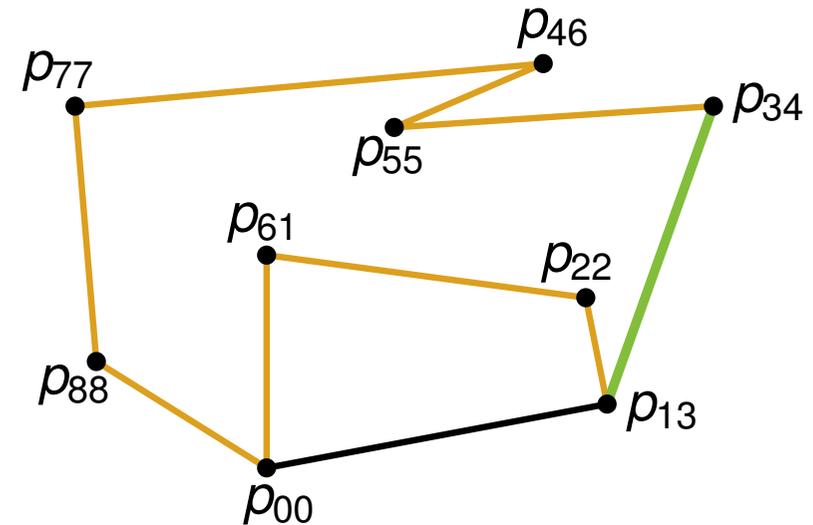
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

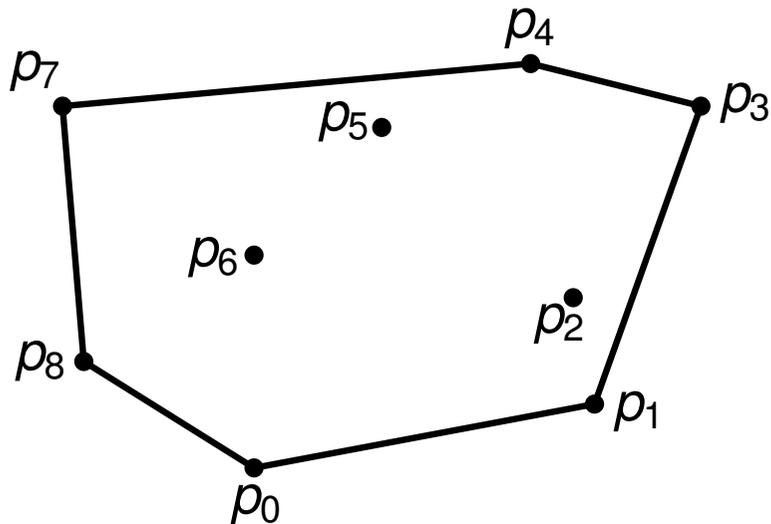
$p_{00} p_{13} p_{34}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

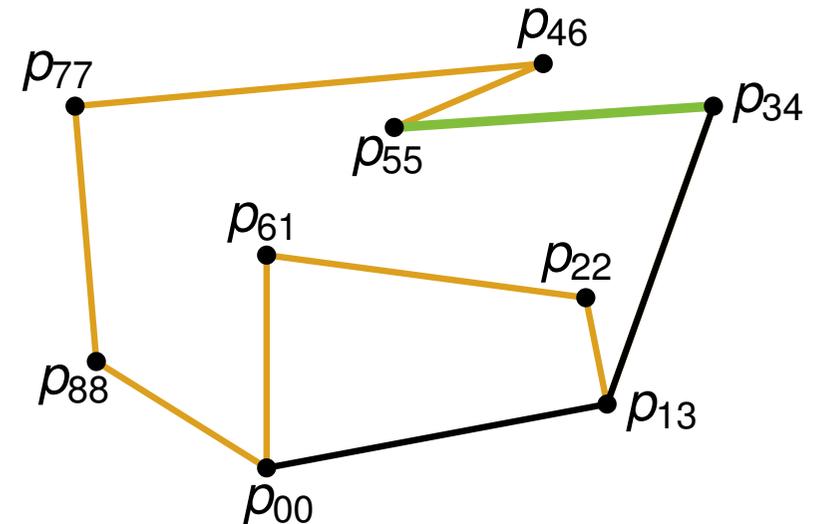
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

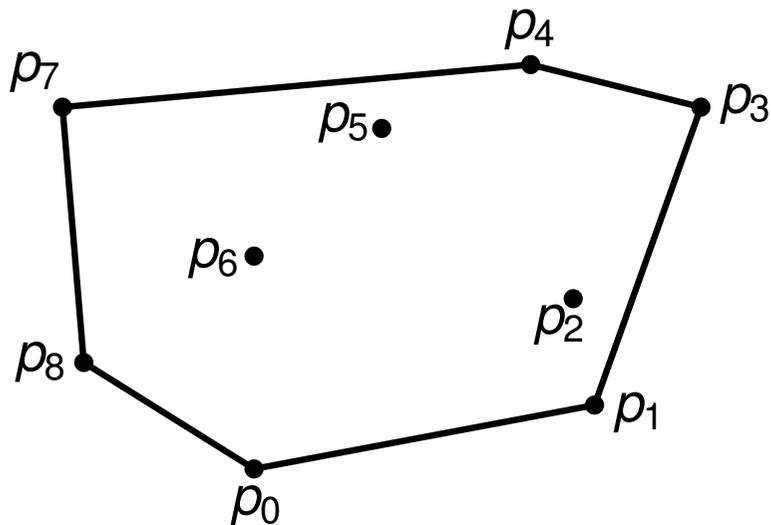
$p_{00} p_{13} p_{34} p_{55}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

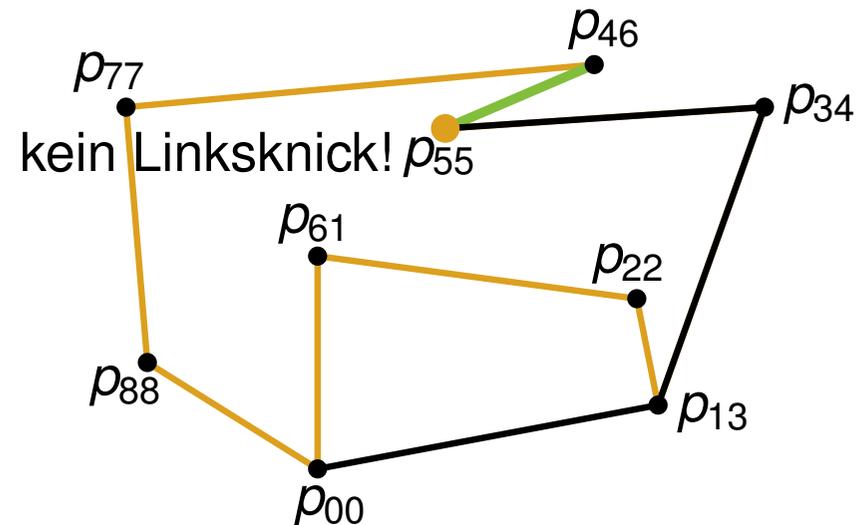
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

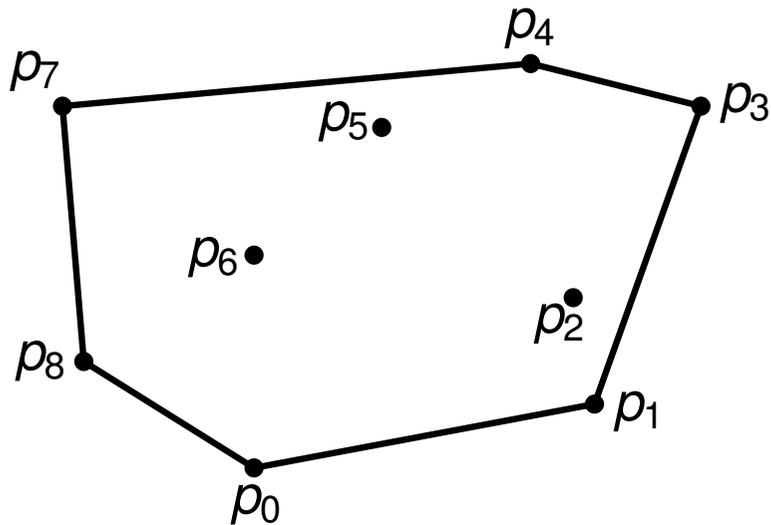
$p_{00} p_{13} p_{34} p_{55} p_{46}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

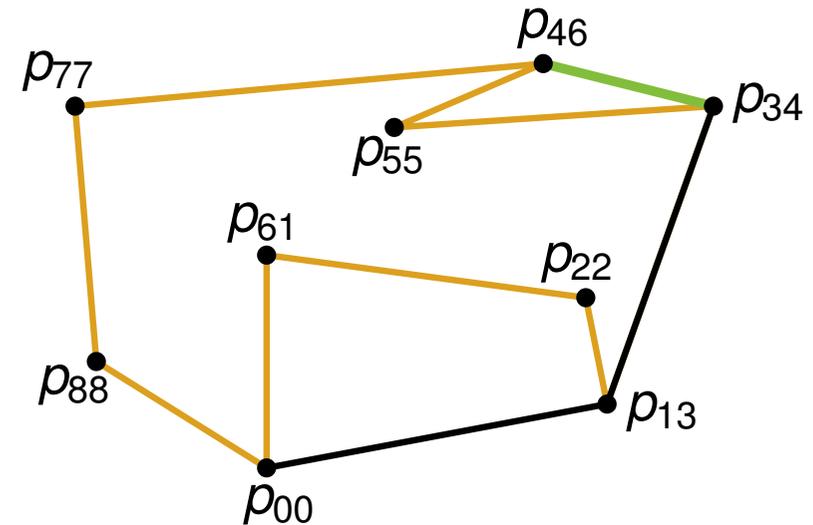
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

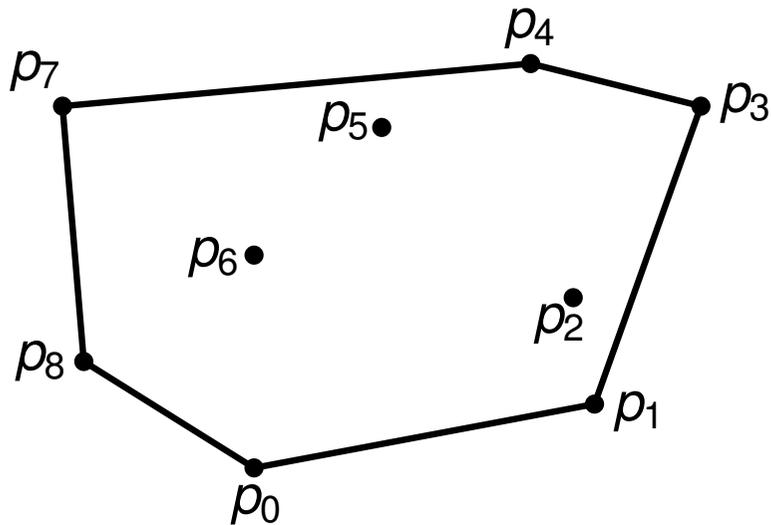
$p_{00} p_{13} p_{34} p_{46}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

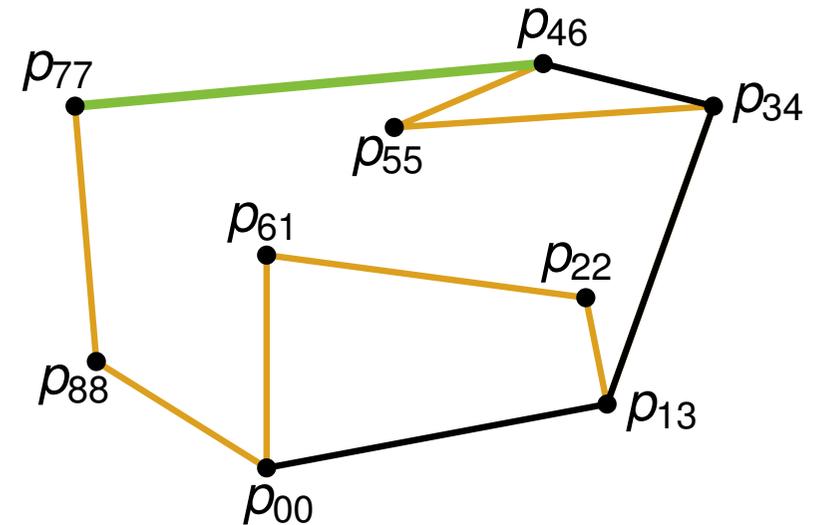
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

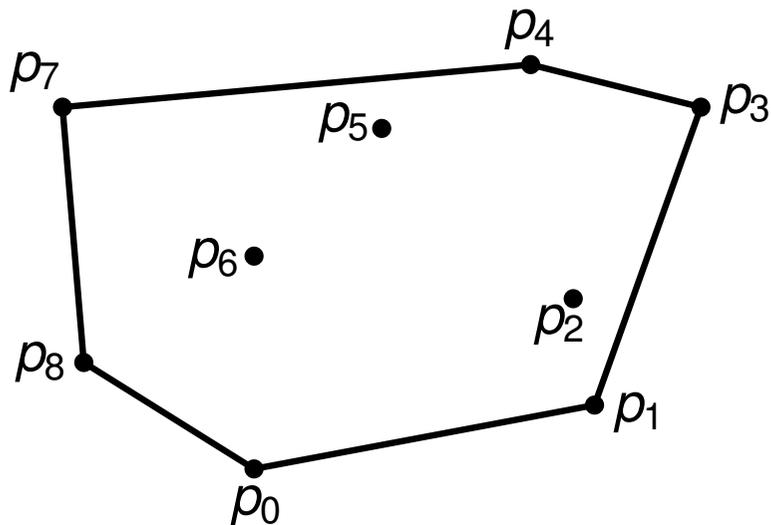
$p_{00} p_{13} p_{34} p_{46} p_{77}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

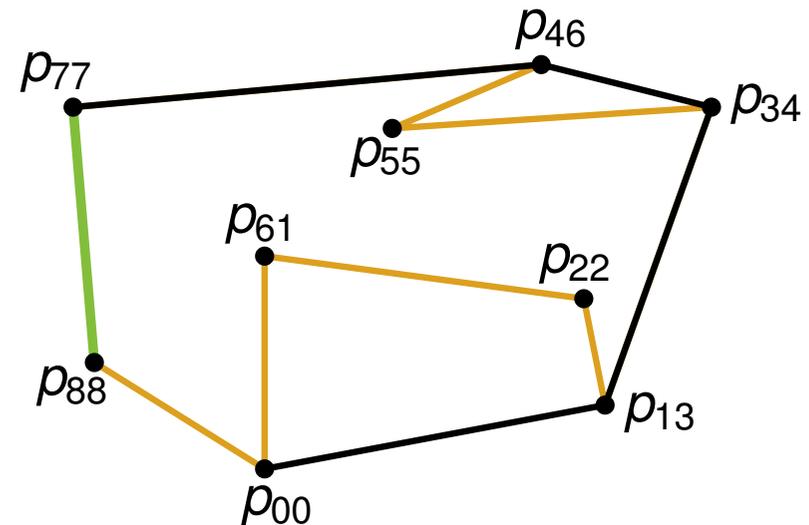
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

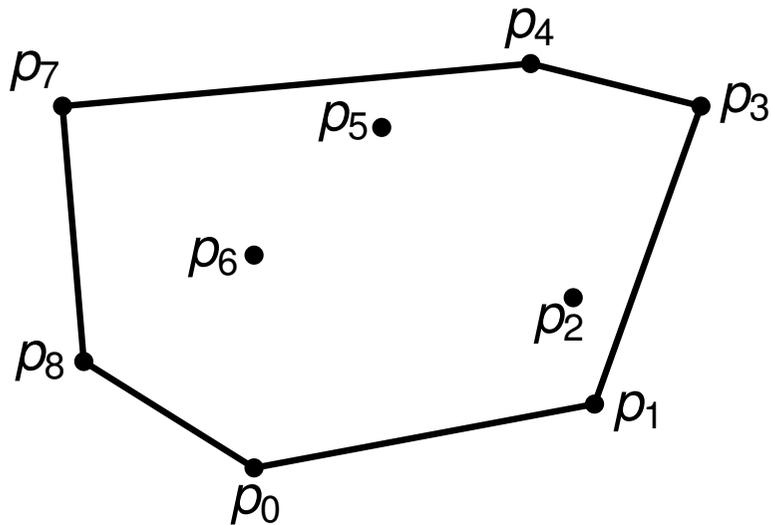
$p_{00} p_{13} p_{34} p_{46} p_{77} p_{88}$

# Problem 4(a)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.

**Idee:** Benutze Graham Scan, nur mit anderer Knotenreihenfolge.

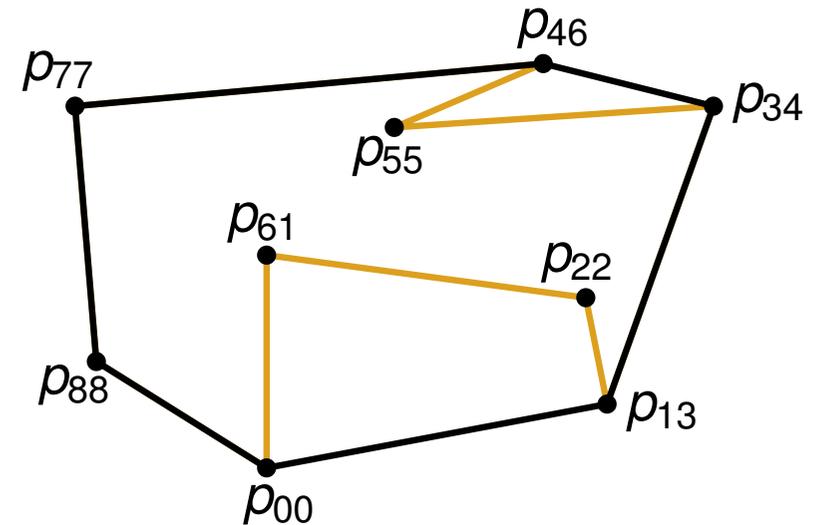
**Beispiel – Graham Scan:**



Zwischenergebnis für die konvexe Hülle:

$p_0 p_1 p_3 p_4 p_7 p_8$

**Beispiel – Graham Scan für Polygon:**



Zwischenergebnis für die konvexe Hülle:

$p_{00} p_{13} p_{34} p_{46} p_{77} p_{88}$

# Problem 4(a)

**Warum funktioniert das?**

# Problem 4(a)

## Warum funktioniert das?

Zeige wie beim Graham Scan die folgenden beiden Invarianten:

- Wird ein Punkt entfernt, so ist er nicht Eckpunkt von der konvexen Hülle.
- Das bisher berechnete Polygon bestimmt stets ein konvexes Polygon.

# Problem 4(a)

## Warum funktioniert das?

Zeige wie beim Graham Scan die folgenden beiden Invarianten:

- Wird ein Punkt entfernt, so ist er nicht Eckpunkt von der konvexen Hülle.
- Das bisher berechnete Polygon bestimmt stets ein konvexes Polygon.

## Wie ist die Laufzeit?

# Problem 4(a)

## Warum funktioniert das?

Zeige wie beim Graham Scan die folgenden beiden Invarianten:

- Wird ein Punkt entfernt, so ist er nicht Eckpunkt von der konvexen Hülle.
- Das bisher berechnete Polygon bestimmt stets ein konvexes Polygon.

## Wie ist die Laufzeit?

- Untersten Punkt finden:  $O(n)$
  - Reihenfolge der Punkte bestimmen:  $O(n)$
  - Jeder Knoten wird nur einmal in das bisher berechnete Polygon eingefügt:  
amortisiert  $O(1)$  pro Schritt
- ⇒ Gesamtlaufzeit:  $O(n)$

# Problem 4(b)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.
- (b) In der Vorlesung wurde gezeigt, dass die Laufzeit zur Berechnung der konvexen Hülle einer Punktmenge mit  $n$  Punkten in  $\Theta(n \log n)$  liegt. Es gibt also keinen Algorithmus, der asymptotisch echt weniger als  $n \log n$  Schritte benötigt. Warum ist das kein Widerspruch zu Teilaufgabe (a)?

# Problem 4(b)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.
- (b) In der Vorlesung wurde gezeigt, dass die Laufzeit zur Berechnung der konvexen Hülle einer Punktmenge mit  $n$  Punkten in  $\Theta(n \log n)$  liegt. Es gibt also keinen Algorithmus, der asymptotisch echt weniger als  $n \log n$  Schritte benötigt. Warum ist das kein Widerspruch zu Teilaufgabe (a)?
- Der Beweis aus der Vorlesung reduziert das Problem Zahlen zu sortieren auf das Problem eine konvexe Hülle zu berechnen.
  - Vergleichsbasiertes Sortieren liegt in  $\Theta(n \log n)$ .

# Problem 4(b)

- (a) Sei  $P$  ein einfaches (nicht notwendigerweise konvexes) Polygon mit  $n$  Eckpunkten. Geben Sie einen Algorithmus mit Laufzeit  $O(n)$  zur Berechnung der konvexen Hülle von  $P$  an.
- (b) In der Vorlesung wurde gezeigt, dass die Laufzeit zur Berechnung der konvexen Hülle einer Punktmenge mit  $n$  Punkten in  $\Theta(n \log n)$  liegt. Es gibt also keinen Algorithmus, der asymptotisch echt weniger als  $n \log n$  Schritte benötigt. Warum ist das kein Widerspruch zu Teilaufgabe (a)?
- Der Beweis aus der Vorlesung reduziert das Problem Zahlen zu sortieren auf das Problem eine konvexe Hülle zu berechnen.
  - Vergleichsbasiertes Sortieren liegt in  $\Theta(n \log n)$ .
  - Wir haben hier nicht nur die Punktmenge sondern auch ein einfaches Polygon gegeben.
  - Sortierte Reihenfolge der Zahlen ist schon durch Reihenfolge der Knoten im Polygon gegeben.

# Problem 5(a)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

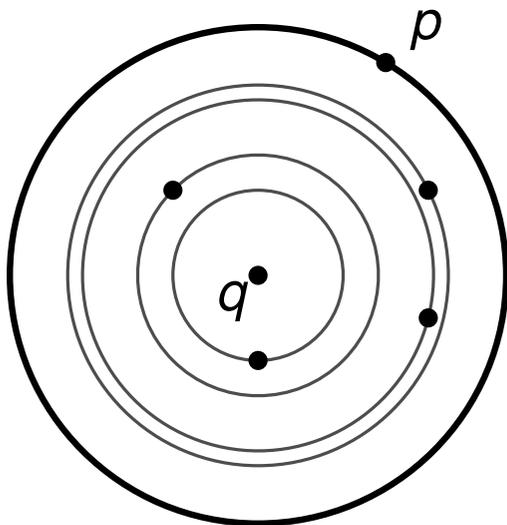
- (a) Seien  $p, q \in Q$  zwei Punkte, sodass  $p$  der am weitesten von  $q$  entfernte Punkt ist. Dann ist  $p$  ein Eckpunkt der konvexen Hülle  $H(Q)$ .
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

# Problem 5(a)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

- (a) Seien  $p, q \in Q$  zwei Punkte, sodass  $p$  der am weitesten von  $q$  entfernte Punkt ist. Dann ist  $p$  ein Eckpunkt der konvexen Hülle  $H(Q)$ .
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

(a)

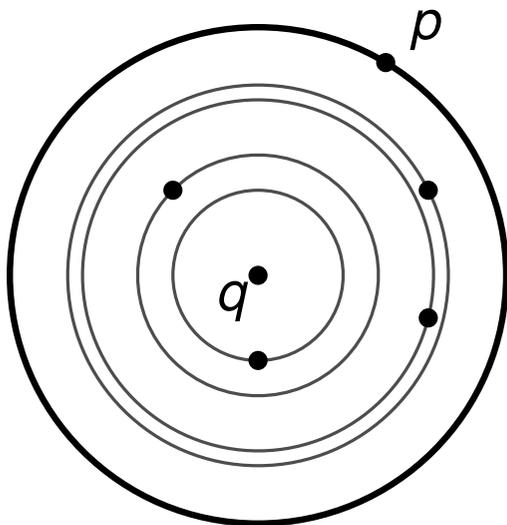


# Problem 5(a)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

- (a) Seien  $p, q \in Q$  zwei Punkte, sodass  $p$  der am weitesten von  $q$  entfernte Punkt ist. Dann ist  $p$  ein Eckpunkt der konvexen Hülle  $H(Q)$ .
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

(a)



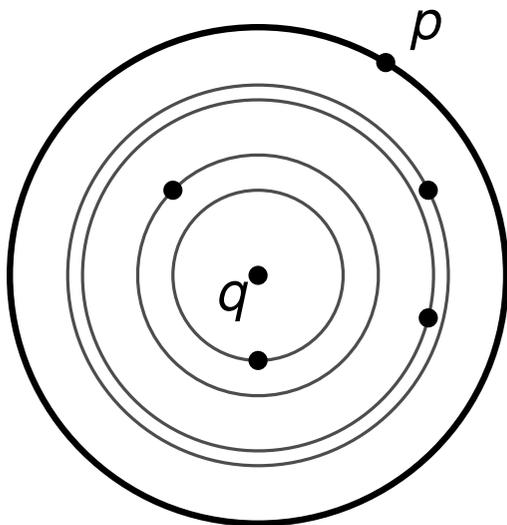
- Der Kreis  $K$  um  $q$  mit Radius  $|\overline{qp}|$  enthält alle anderen Punkte in  $Q$ .

# Problem 5(a)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

- (a) Seien  $p, q \in Q$  zwei Punkte, sodass  $p$  der am weitesten von  $q$  entfernte Punkt ist. Dann ist  $p$  ein Eckpunkt der konvexen Hülle  $H(Q)$ .
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

(a)



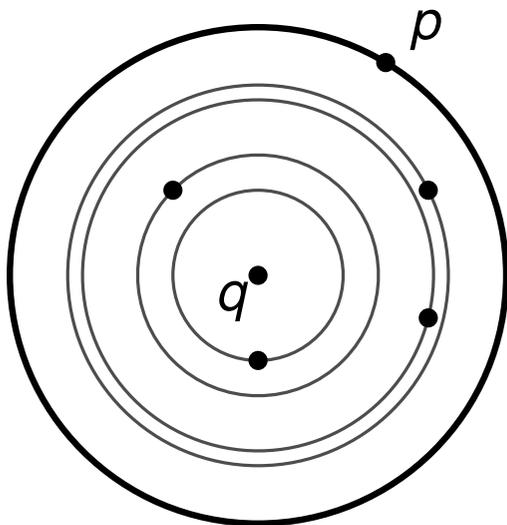
- Der Kreis  $K$  um  $q$  mit Radius  $|\overline{qp}|$  enthält alle anderen Punkte in  $Q$ .
- Außerdem ist ein Kreis konvex.

# Problem 5(a)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

- (a) Seien  $p, q \in Q$  zwei Punkte, sodass  $p$  der am weitesten von  $q$  entfernte Punkt ist. Dann ist  $p$  ein Eckpunkt der konvexen Hülle  $H(Q)$ .
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

(a)



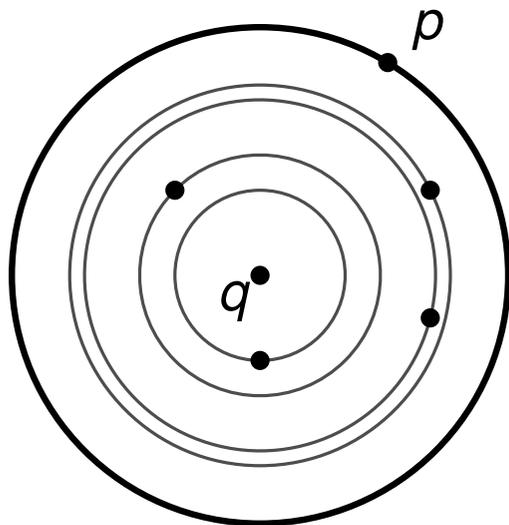
- Der Kreis  $K$  um  $q$  mit Radius  $|\overline{qp}|$  enthält alle anderen Punkte in  $Q$ .
  - Außerdem ist ein Kreis konvex.
- ⇒ die konvexe Hülle  $H(Q)$  ist im Kreis  $K$  enthalten.

# Problem 5(a)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

- (a) Seien  $p, q \in Q$  zwei Punkte, sodass  $p$  der am weitesten von  $q$  entfernte Punkt ist. Dann ist  $p$  ein Eckpunkt der konvexen Hülle  $H(Q)$ .
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

(a)



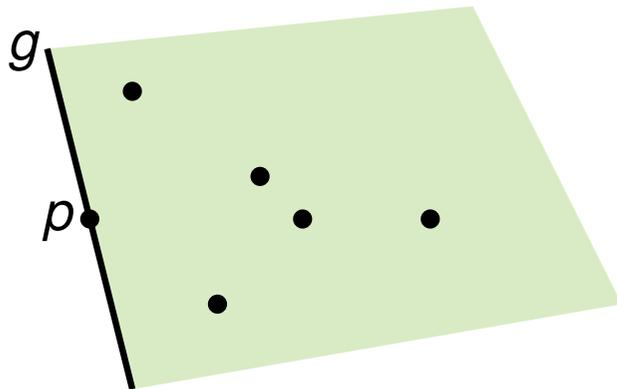
- Der Kreis  $K$  um  $q$  mit Radius  $|\overline{qp}|$  enthält alle anderen Punkte in  $Q$ .
  - Außerdem ist ein Kreis konvex.
- $\Rightarrow$  die konvexe Hülle  $H(Q)$  ist im Kreis  $K$  enthalten.
- $\Rightarrow$  der Randpunkt  $p$  von  $K$  muss auch Randpunkt von  $H(Q)$  sein.

# Problem 5(b)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

**Richtung 1:** Es gibt Gerade  $g \Rightarrow p$  ist Eckpunkt von  $H(Q)$ .

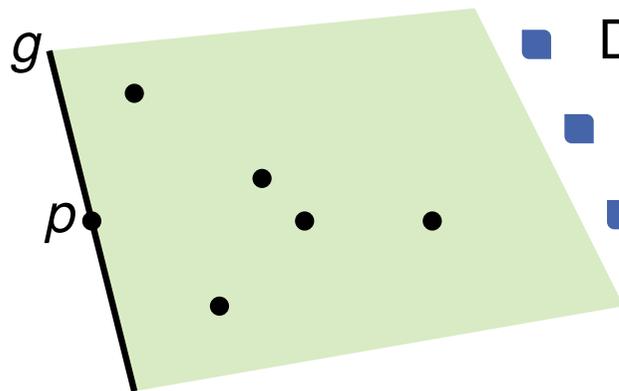


# Problem 5(b)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

**Richtung 1:** Es gibt Gerade  $g \Rightarrow p$  ist Eckpunkt von  $H(Q)$ .



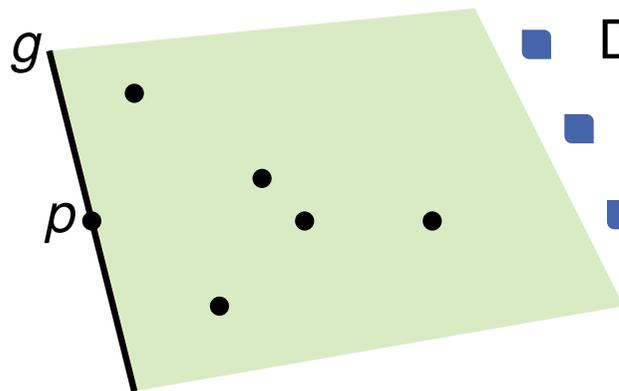
- Durch  $g$  begrenzte Halbebenen enthält alle Punkte in  $Q$ .
- Halbebenen sind konvex.
- Gleiches Argument wie beim Kreis zeigt:  $g$  ist Eckpunkt von  $H(Q)$ .

# Problem 5(b)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

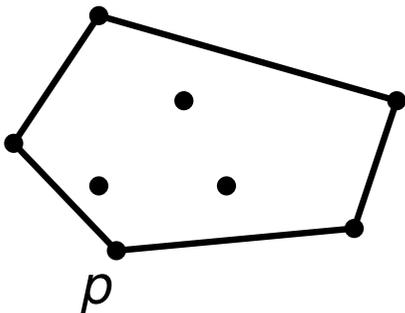
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

**Richtung 1:** Es gibt Gerade  $g \Rightarrow p$  ist Eckpunkt von  $H(Q)$ .



- Durch  $g$  begrenzte Halbebenen enthält alle Punkte in  $Q$ .
- Halbebenen sind konvex.
- Gleiches Argument wie beim Kreis zeigt:  $g$  ist Eckpunkt von  $H(Q)$ .

**Richtung 2:**  $p$  ist Eckpunkt von  $H(Q) \Rightarrow$  es gibt Gerade  $g$ .

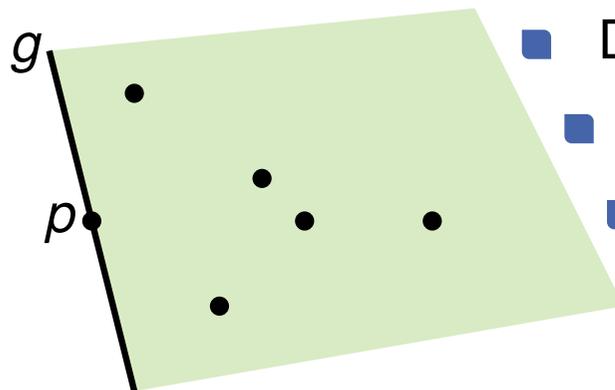


# Problem 5(b)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

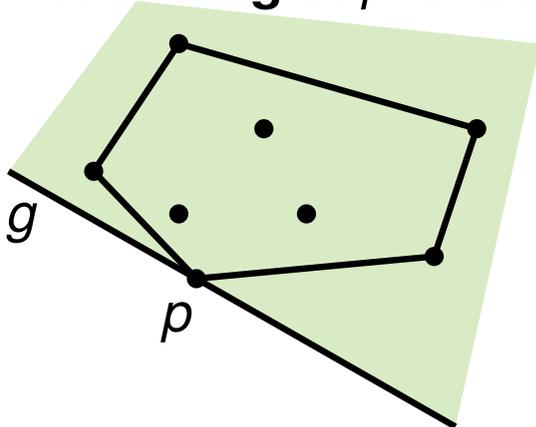
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

**Richtung 1:** Es gibt Gerade  $g \Rightarrow p$  ist Eckpunkt von  $H(Q)$ .



- Durch  $g$  begrenzte Halbebenen enthält alle Punkte in  $Q$ .
- Halbebenen sind konvex.
- Gleiches Argument wie beim Kreis zeigt:  $g$  ist Eckpunkt von  $H(Q)$ .

**Richtung 2:**  $p$  ist Eckpunkt von  $H(Q) \Rightarrow$  es gibt Gerade  $g$ .



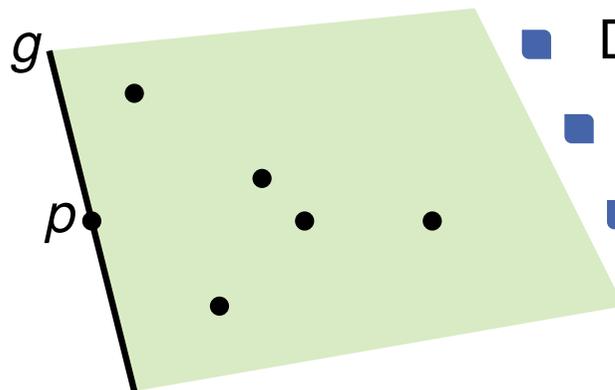
- Sei  $g$  eine Gerade durch  $p$ , die beide Nachbarn auf dem Rand von  $H(Q)$  auf der Gleichen Seite hat.

# Problem 5(b)

Gegeben sei eine Punktmenge  $Q$ . Zeigen Sie folgende Aussagen.

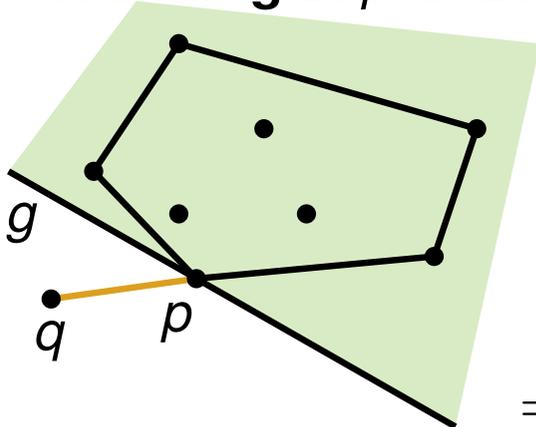
- (b) Ein Punkt  $p \in Q$  ist ein Eckpunkt der Konvexen Hülle  $H(Q)$  genau dann, wenn es eine Gerade  $g$  durch  $p$  gibt, sodass alle Punkte  $Q \setminus \{p\}$  auf der gleichen Seite von  $g$  liegen.

**Richtung 1:** Es gibt Gerade  $g \Rightarrow p$  ist Eckpunkt von  $H(Q)$ .



- Durch  $g$  begrenzte Halbebenen enthält alle Punkte in  $Q$ .
- Halbebenen sind konvex.
- Gleiches Argument wie beim Kreis zeigt:  $g$  ist Eckpunkt von  $H(Q)$ .

**Richtung 2:**  $p$  ist Eckpunkt von  $H(Q) \Rightarrow$  es gibt Gerade  $g$ .



- Sei  $g$  eine Gerade durch  $p$ , die beide Nachbarn auf dem Rand von  $H(Q)$  auf der Gleichen Seite hat.
- Für einen Punkt  $q$  auf der anderen Seite von  $g$  würde  $\overline{pq}$  nicht in  $H(Q)$  liegen.  $\Rightarrow$  Widerspruch zur Konvexität.

$\Rightarrow g$  tut das gewünschte.