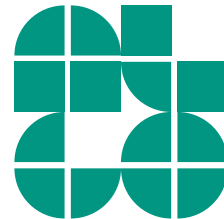


Vorlesung Algorithmische Geometrie

Streckenschnitte

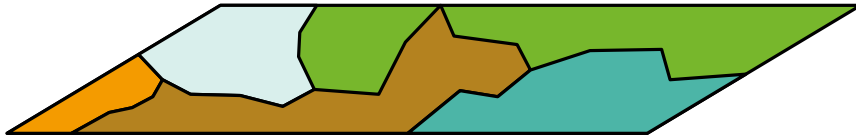
INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Martin Nöllenburg
22.04.2014



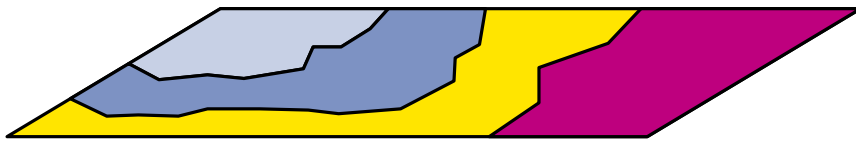
Überlagern von Kartenebenen

Beispiel: Gegeben zwei verschiedene Kartenebenen, bestimme deren Schnitt.



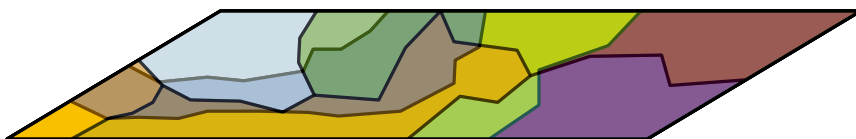
Flächennutzung

+



Niederschlag

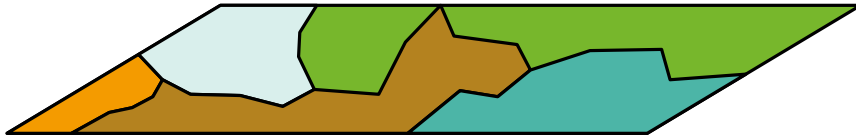
=



Kombination beider Themen

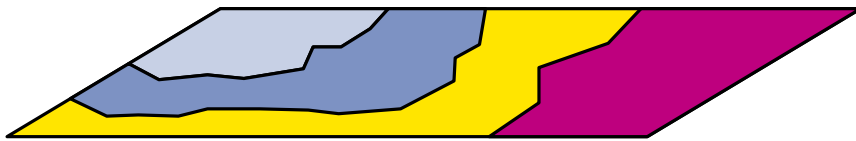
Überlagern von Kartenebenen

Beispiel: Gegeben zwei verschiedene Kartenebenen, bestimme deren Schnitt.



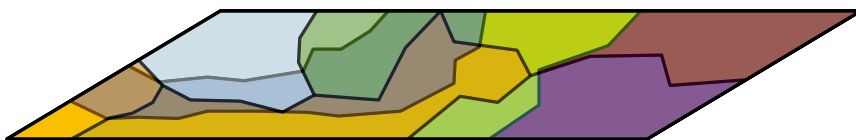
Flächennutzung

+



Niederschlag

=



Kombination beider Themen

- Regionen sind Polygone
- Polygone sind Streckenmengen
- **berechne alle Streckenschnittpunkte**
- berechne induzierte Regionen

Problemstellung

Geg: Menge $S = \{s_1, \dots, s_n\}$ von Strecken in der Ebene

Ges:

- alle Schnittpunkte von zwei oder mehr Strecken
- für jeden Schnittpunkt die beteiligten Strecken

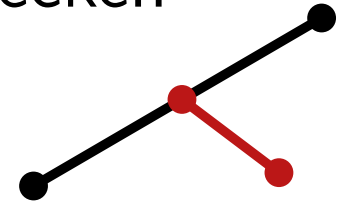
Problemstellung

Geg: Menge $S = \{s_1, \dots, s_n\}$ von Strecken in der Ebene

Ges:

- alle Schnittpunkte von zwei oder mehr Strecken
- für jeden Schnittpunkt die beteiligten Strecken

Def: Strecken sind **abgeschlossene** Mengen



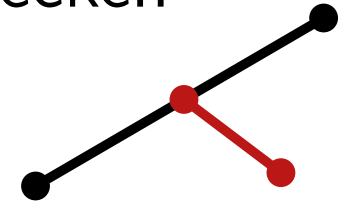
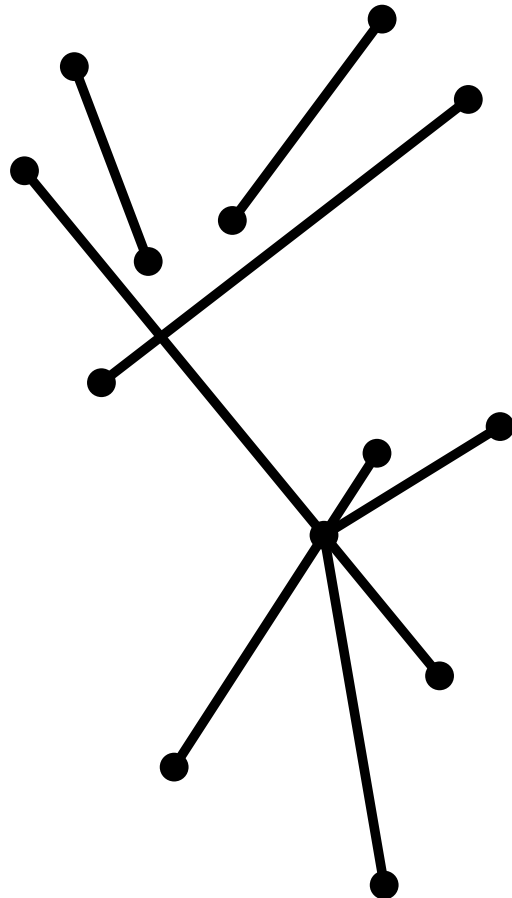
Problemstellung

Geg: Menge $S = \{s_1, \dots, s_n\}$ von Strecken in der Ebene

Ges:

- alle Schnittpunkte von zwei oder mehr Strecken
- für jeden Schnittpunkt die beteiligten Strecken

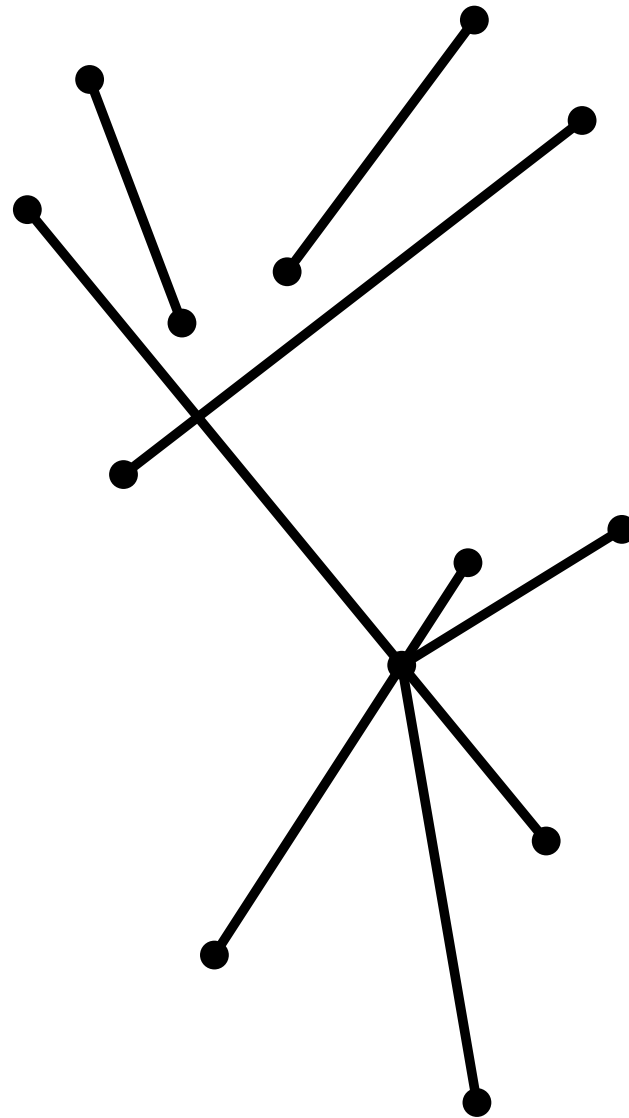
Def: Strecken sind **abgeschlossene** Mengen



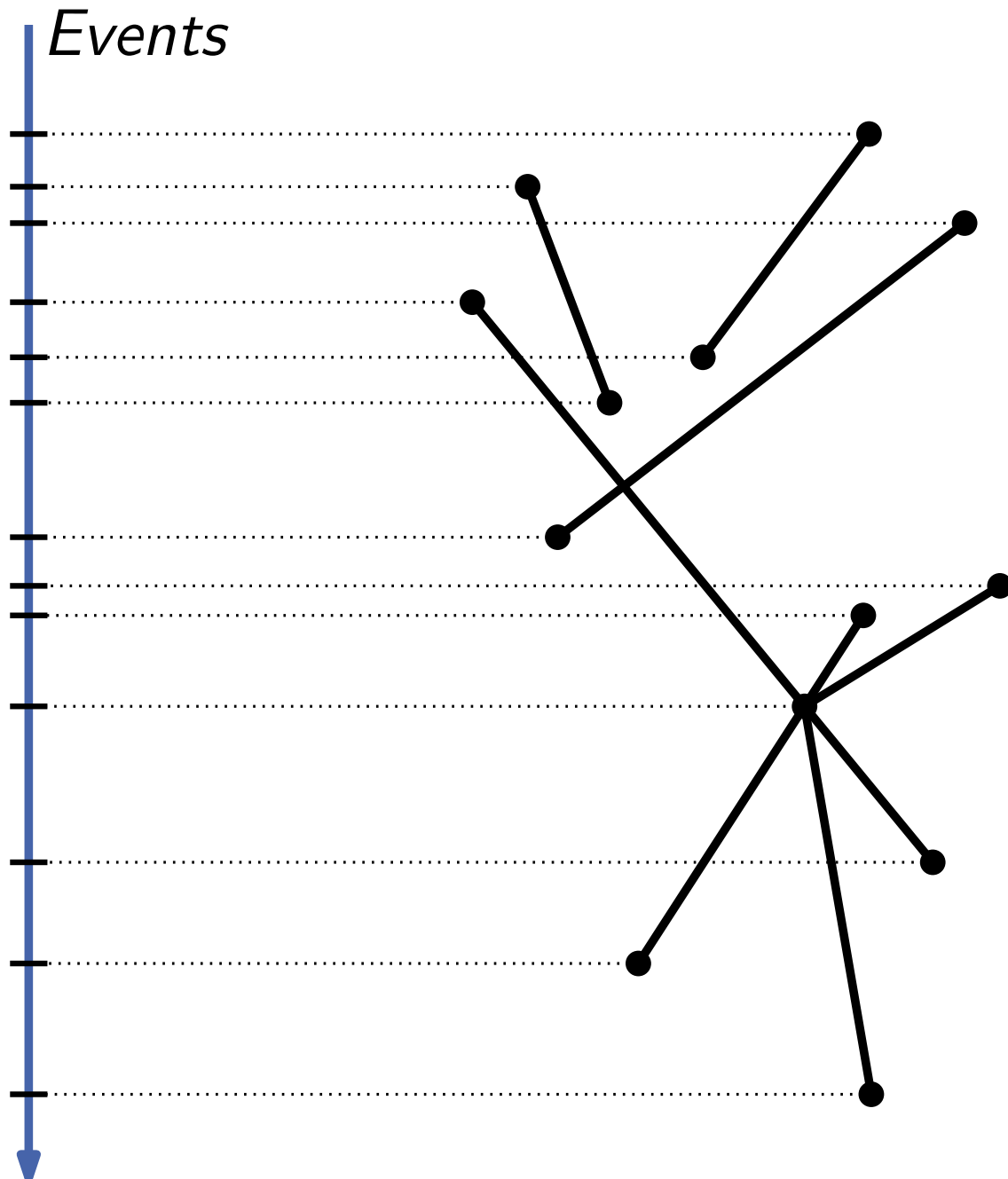
Diskussion:

- Wie kann man das Problem naiv lösen?
- Ist das evtl. schon optimal?
- Sehen Sie Verbesserungsansätze?

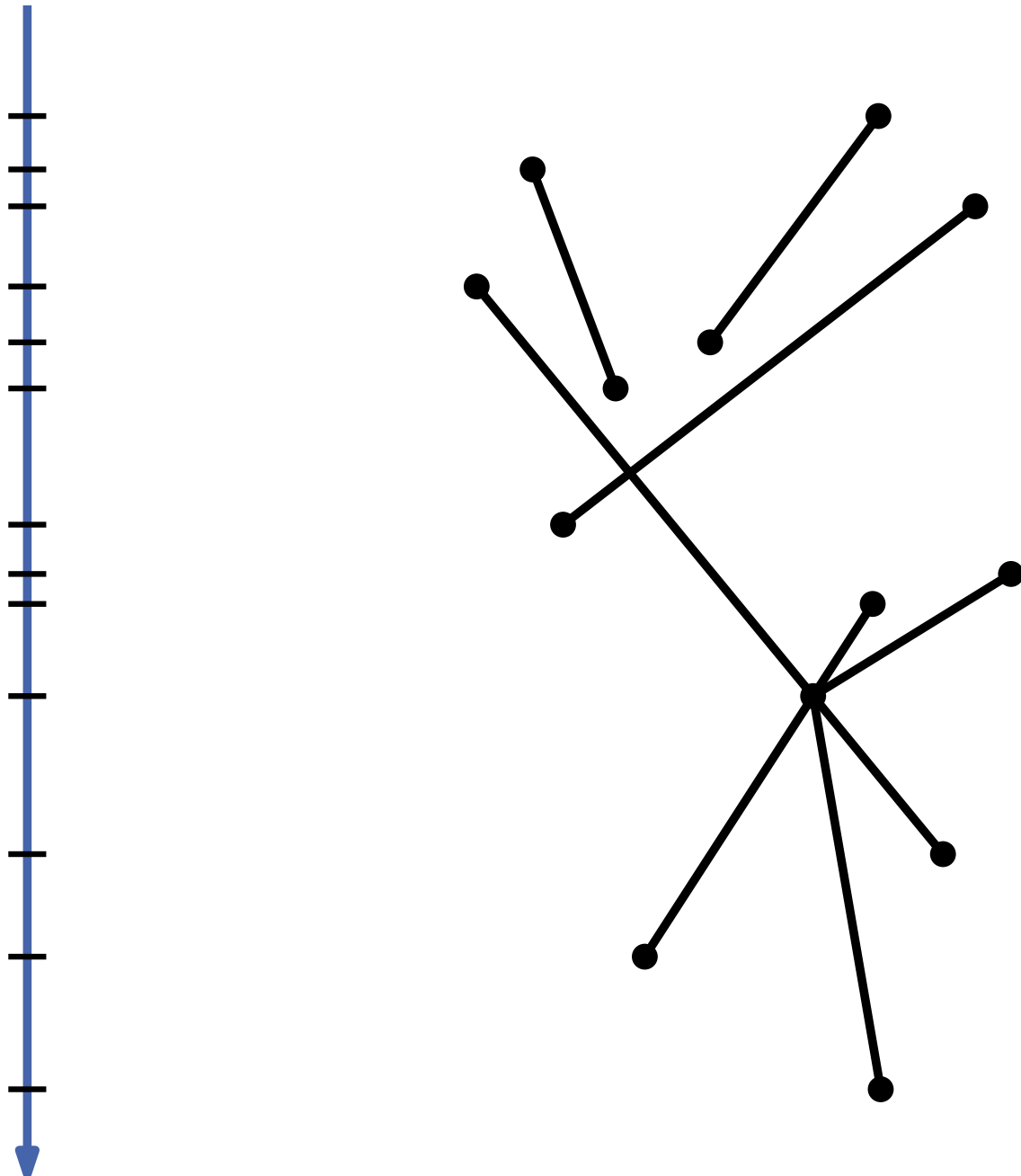
Das Sweep-Line Verfahren: Beispiel



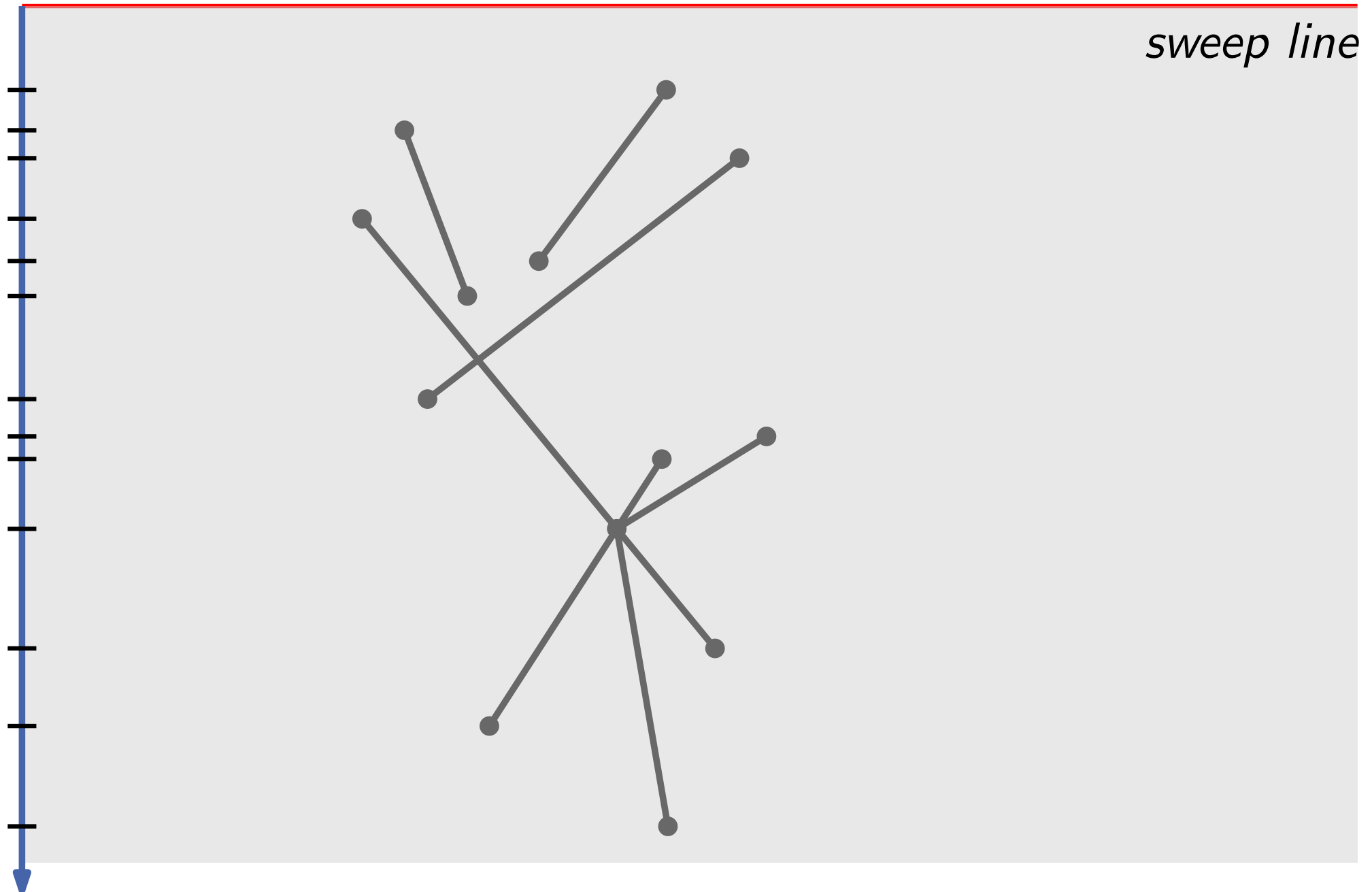
Das Sweep-Line Verfahren: Beispiel



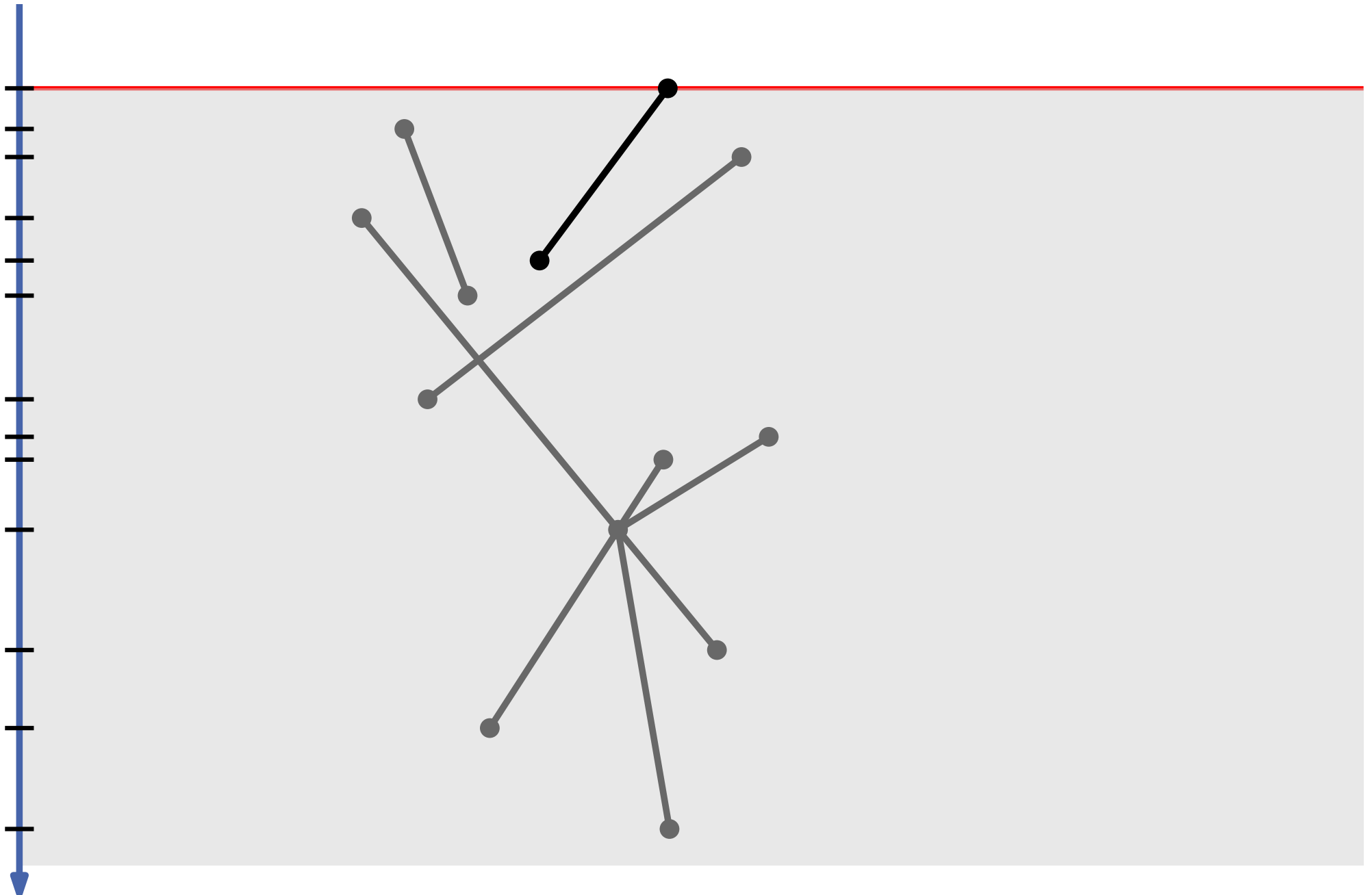
Das Sweep-Line Verfahren: Beispiel



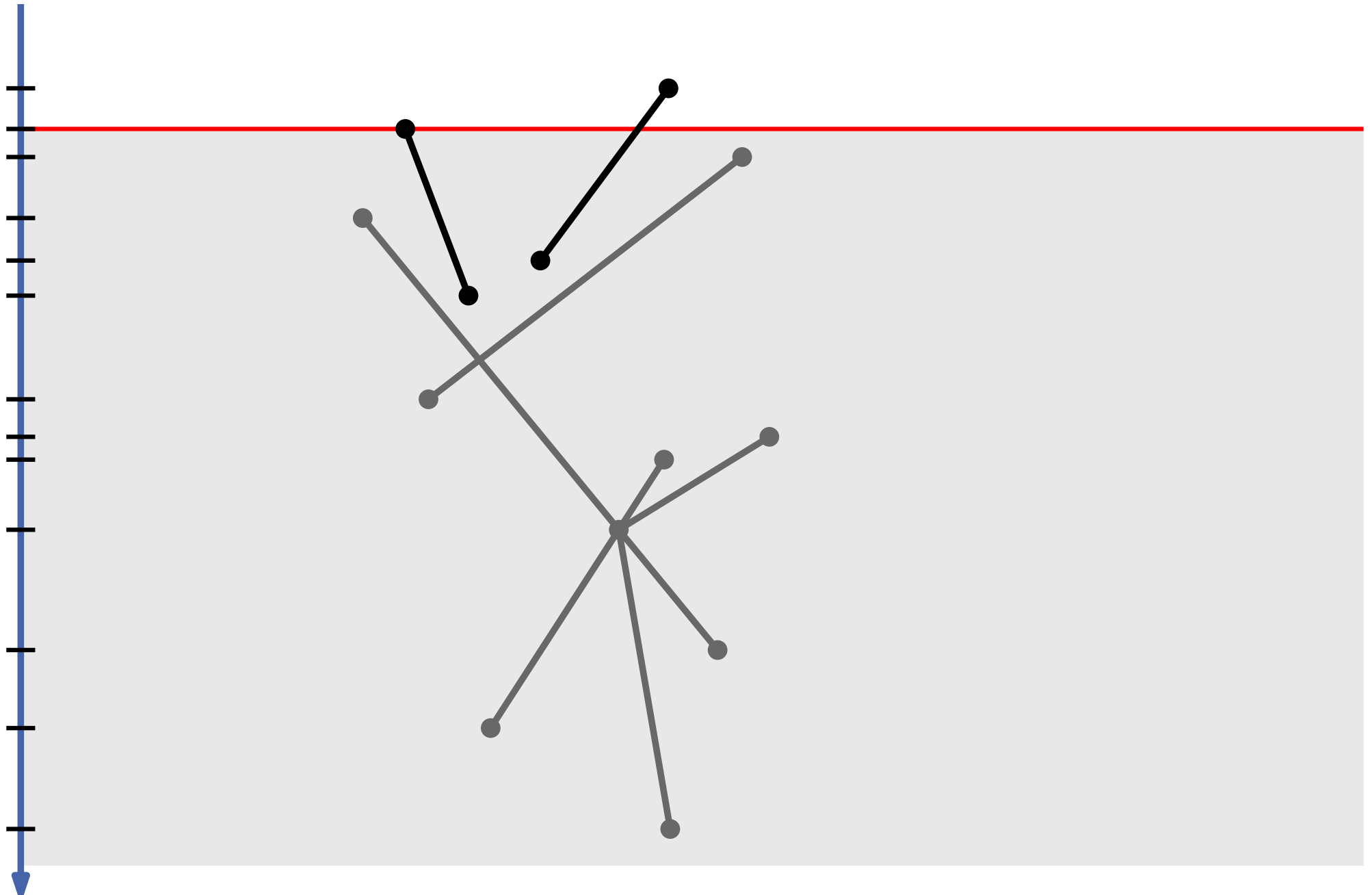
Das Sweep-Line Verfahren: Beispiel



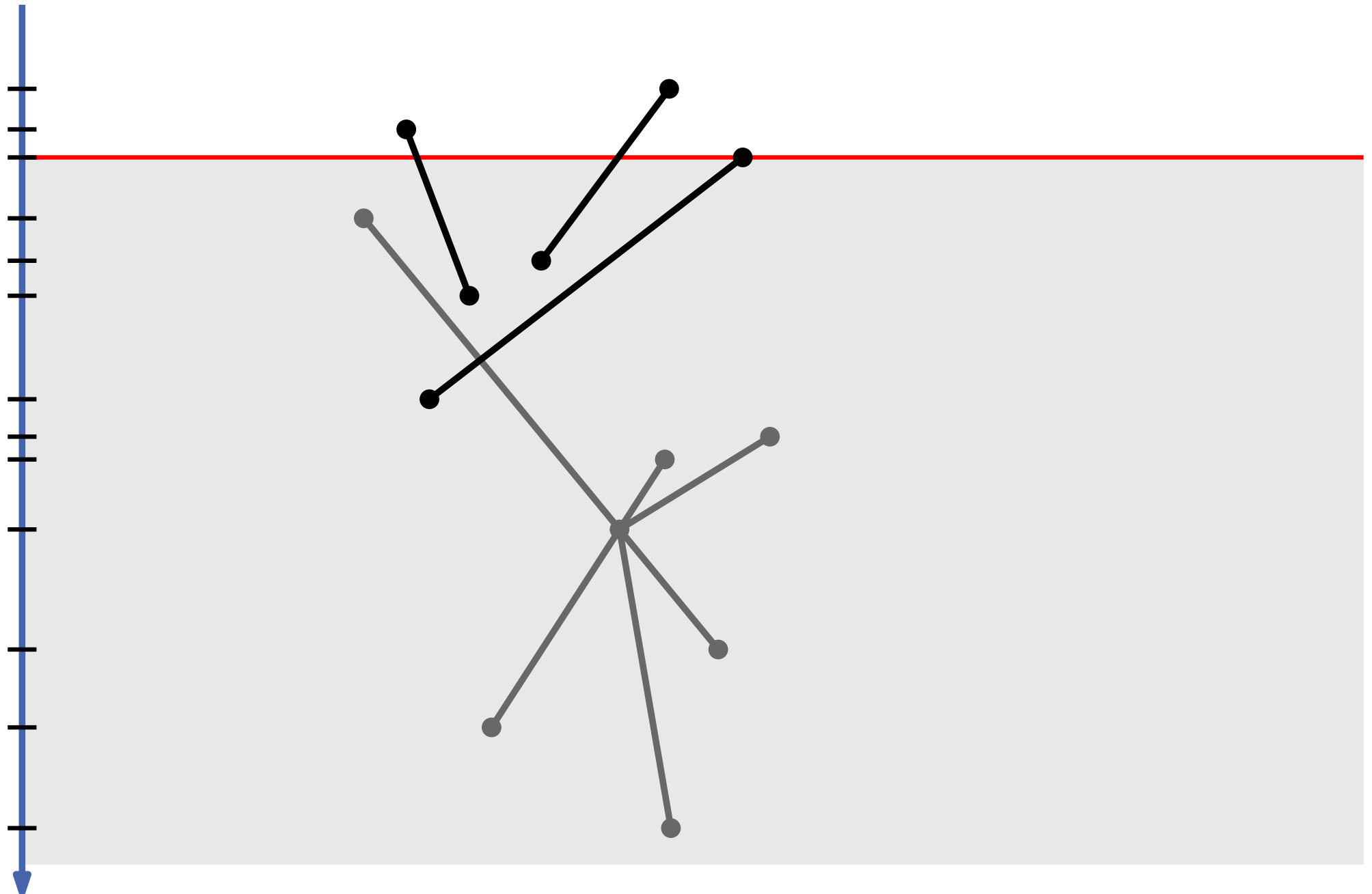
Das Sweep-Line Verfahren: Beispiel



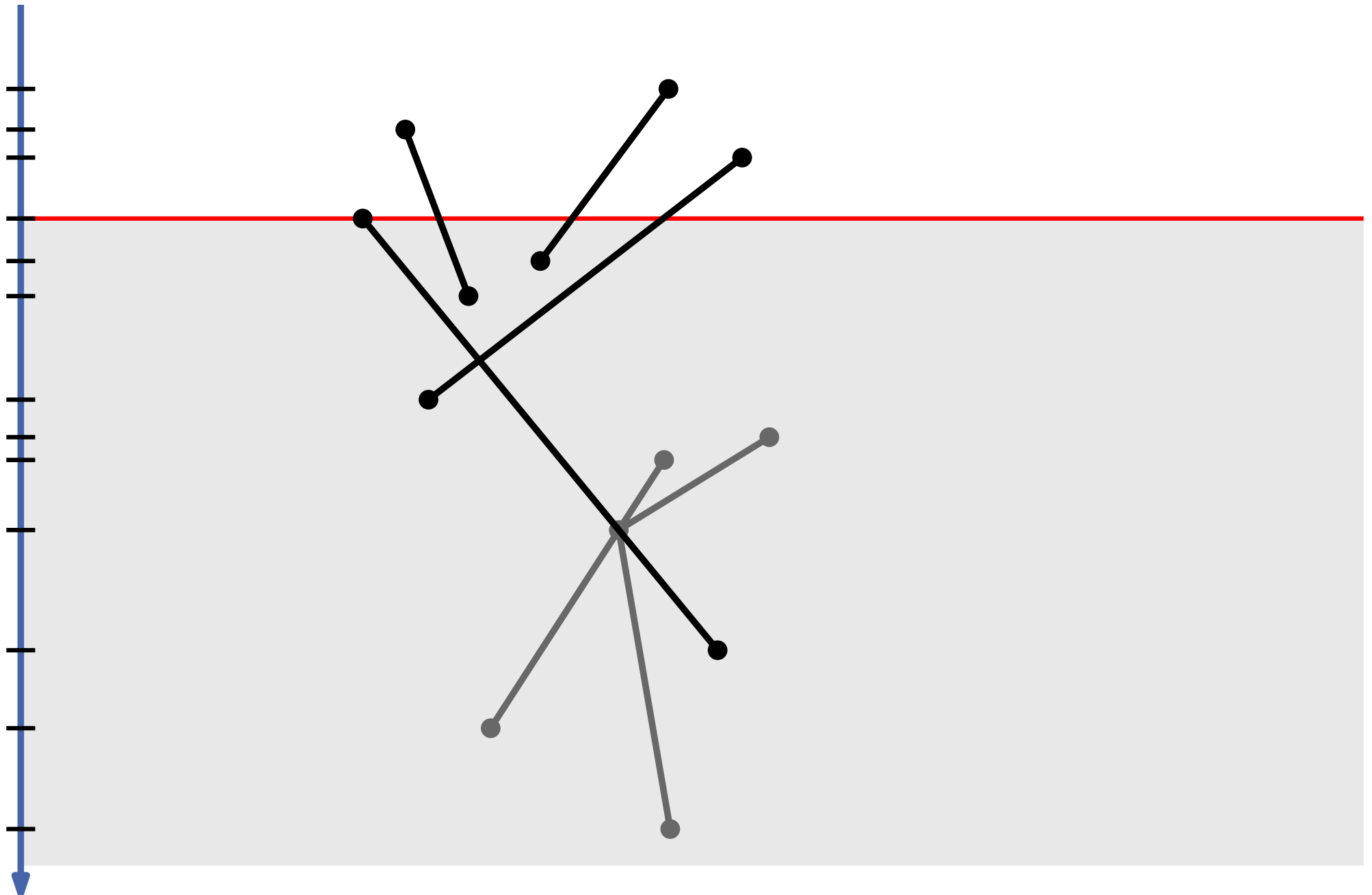
Das Sweep-Line Verfahren: Beispiel



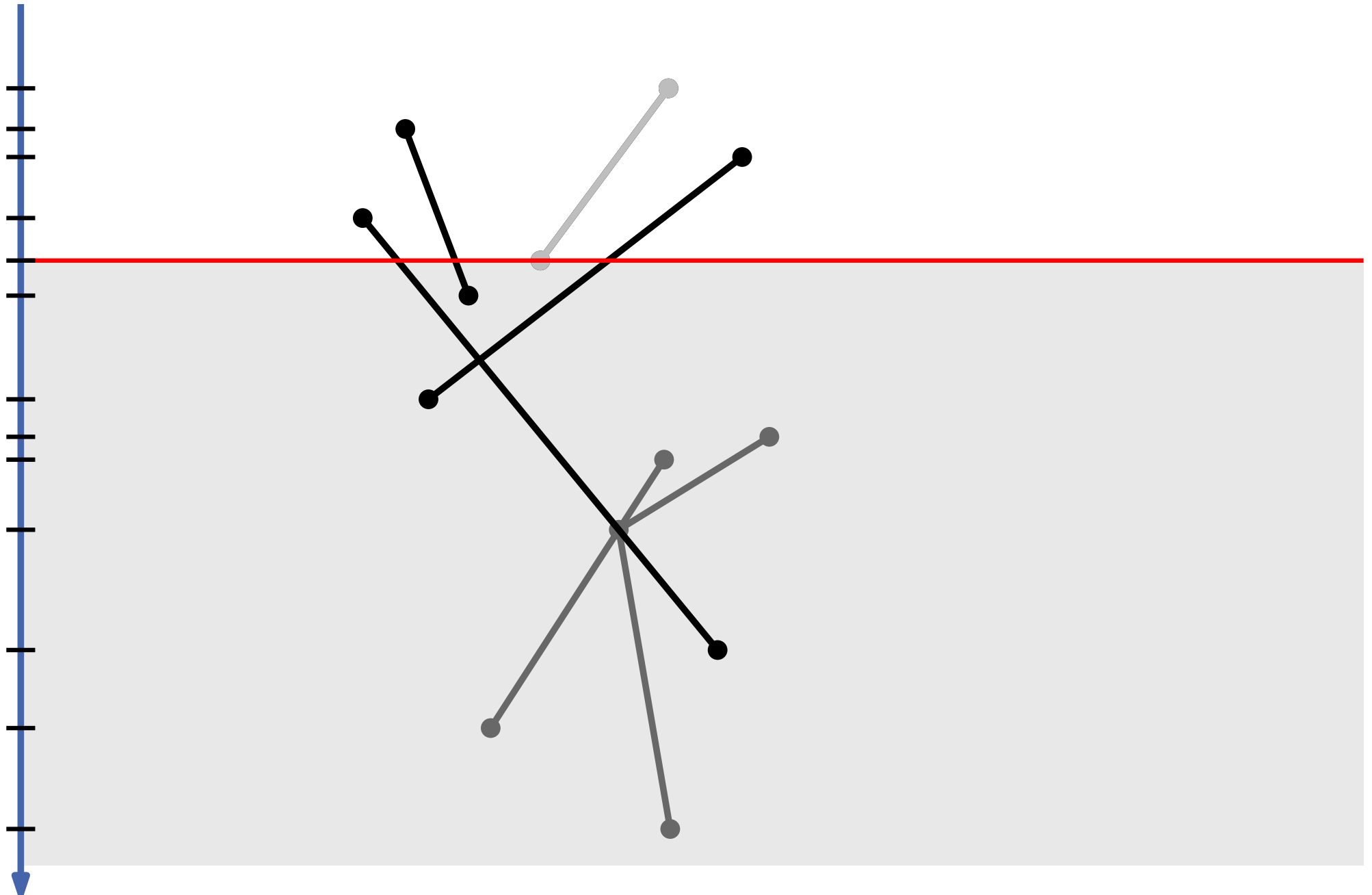
Das Sweep-Line Verfahren: Beispiel



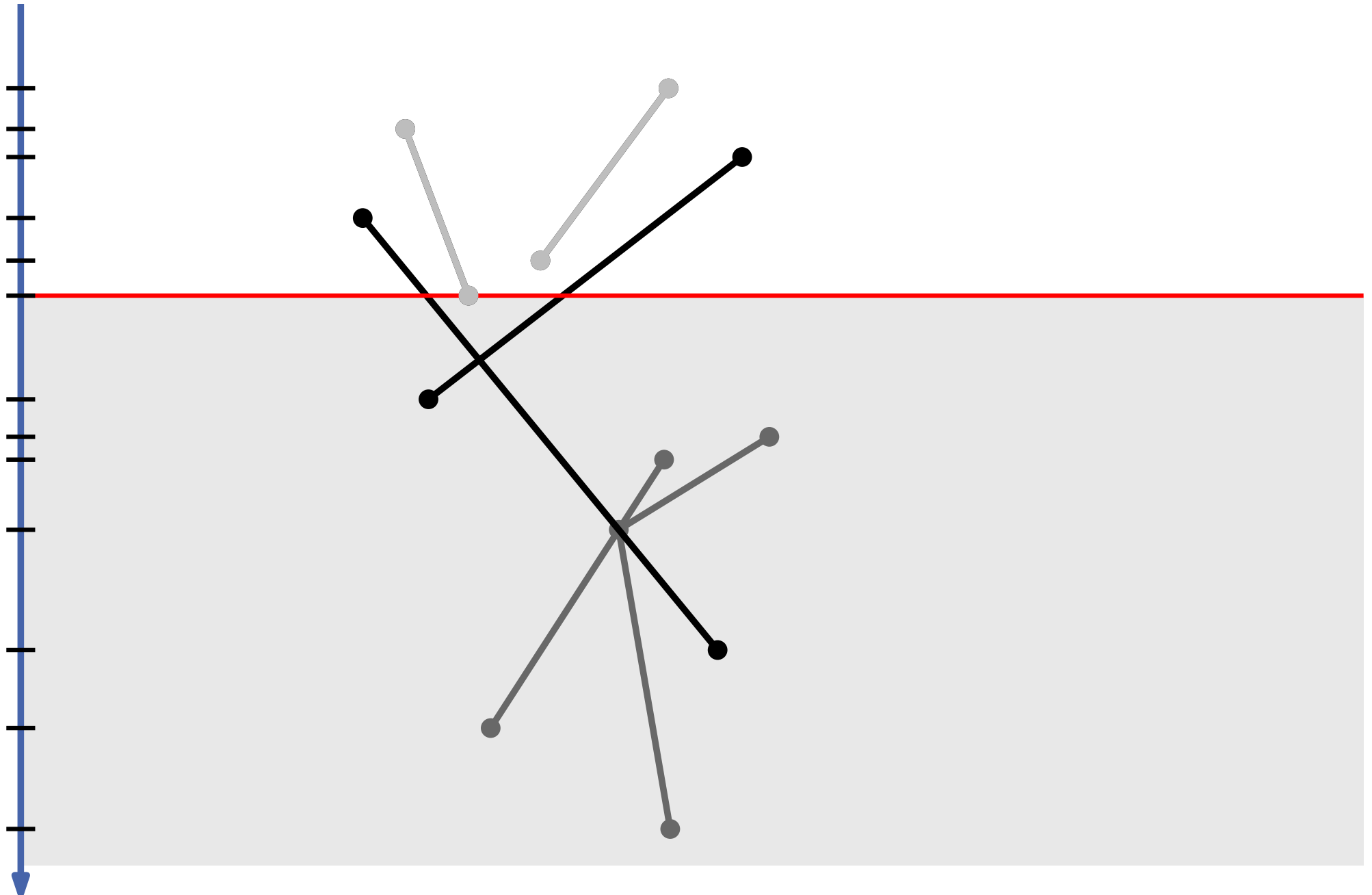
Das Sweep-Line Verfahren: Beispiel



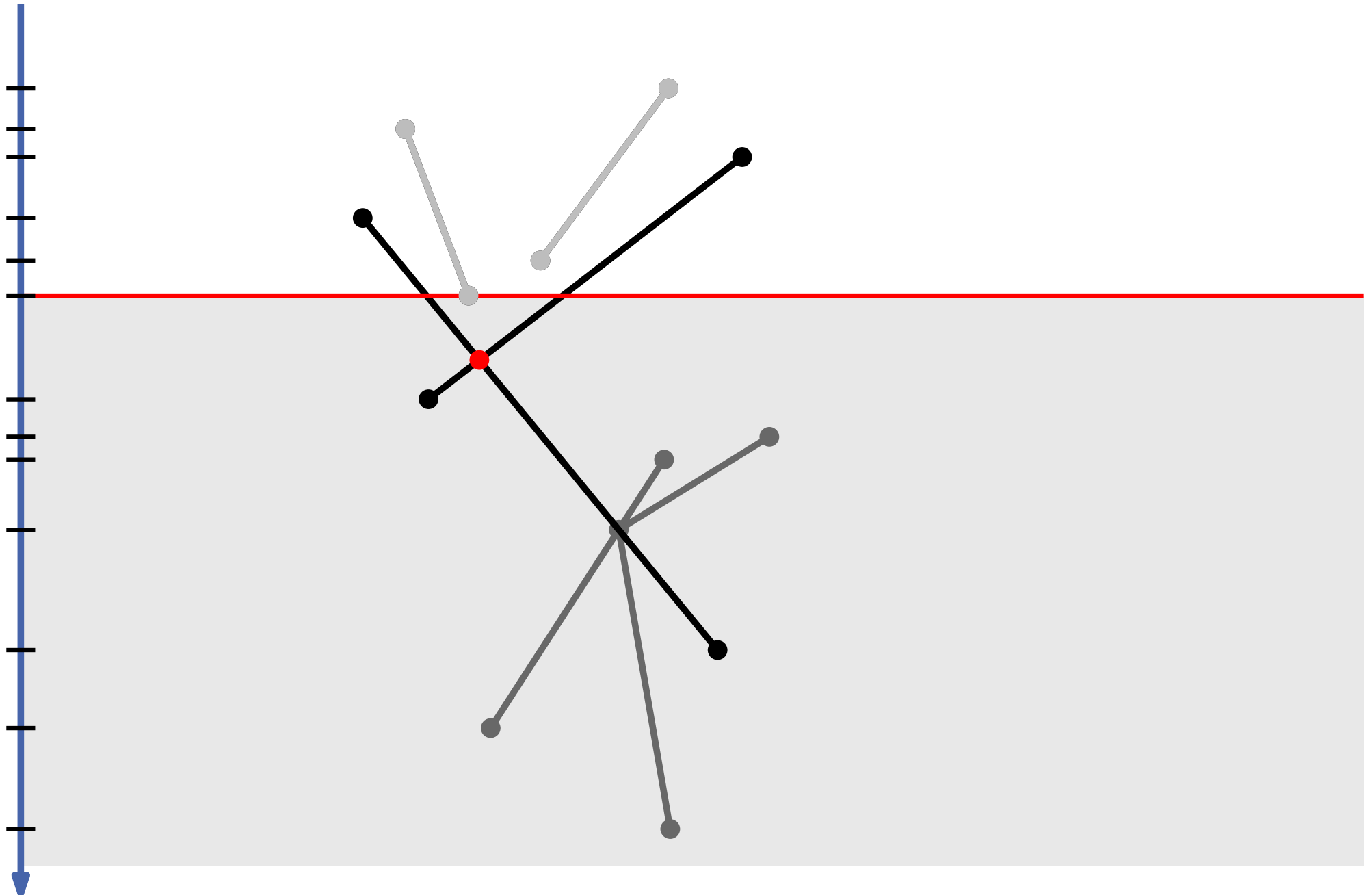
Das Sweep-Line Verfahren: Beispiel



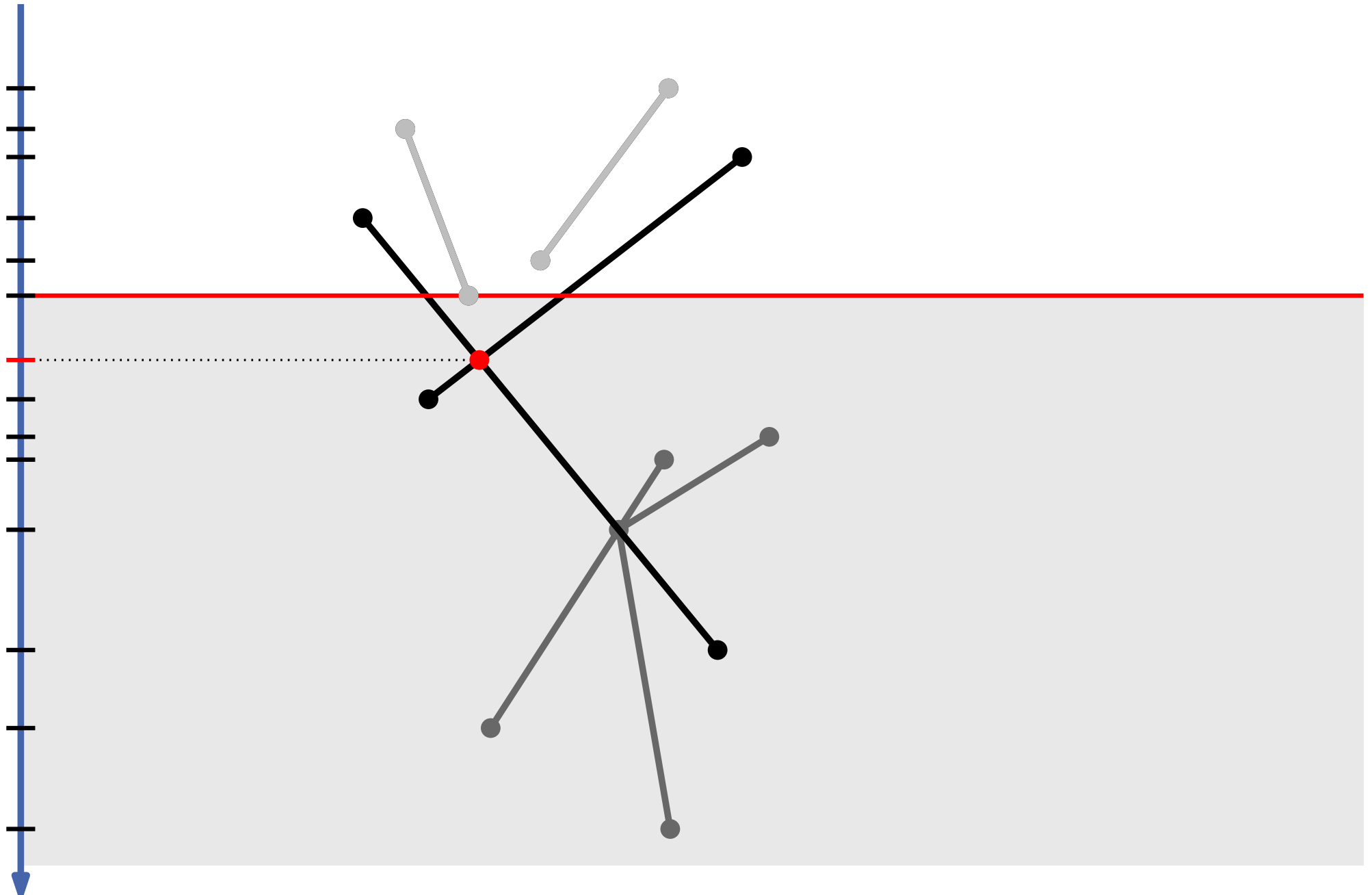
Das Sweep-Line Verfahren: Beispiel



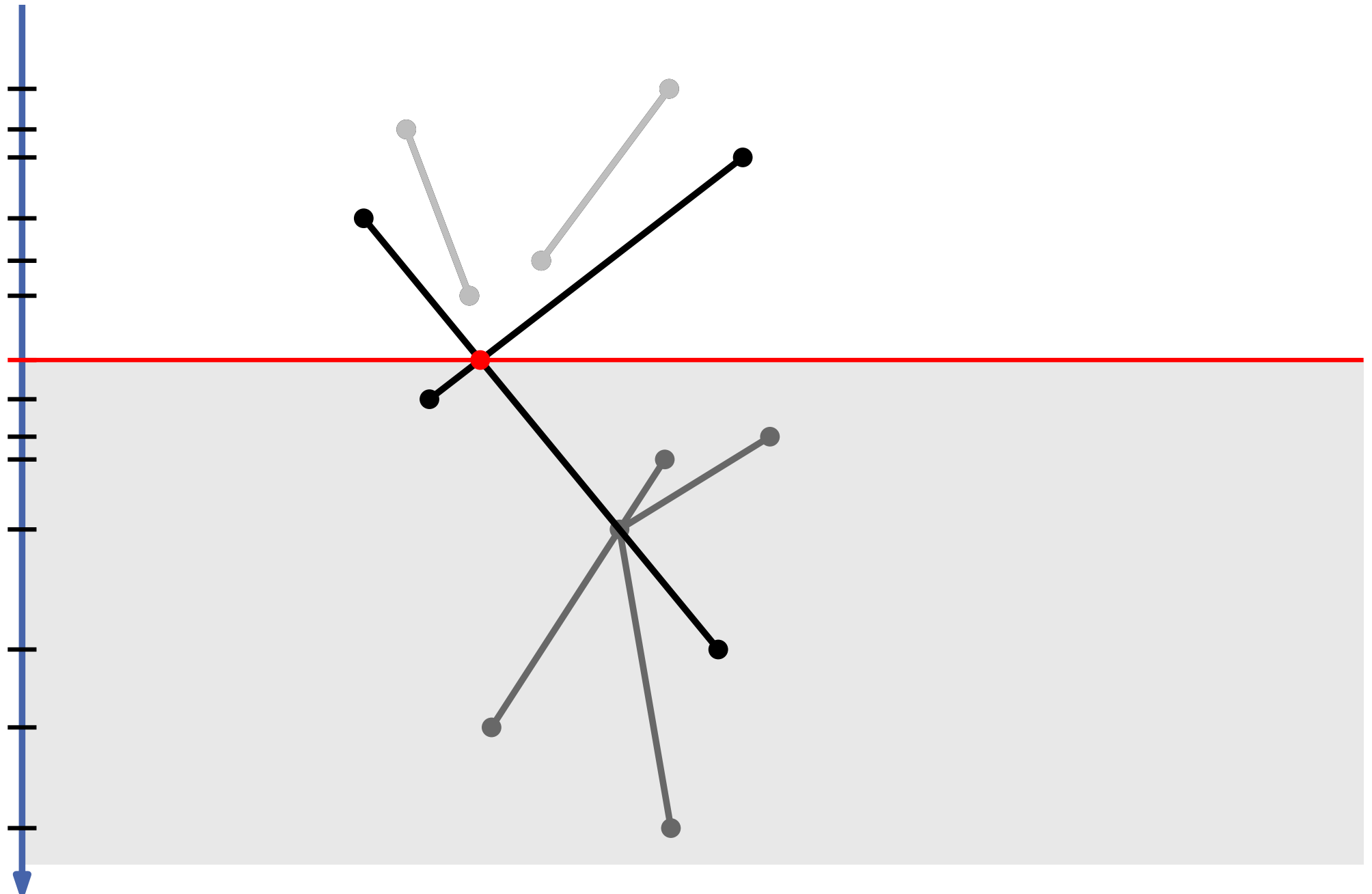
Das Sweep-Line Verfahren: Beispiel



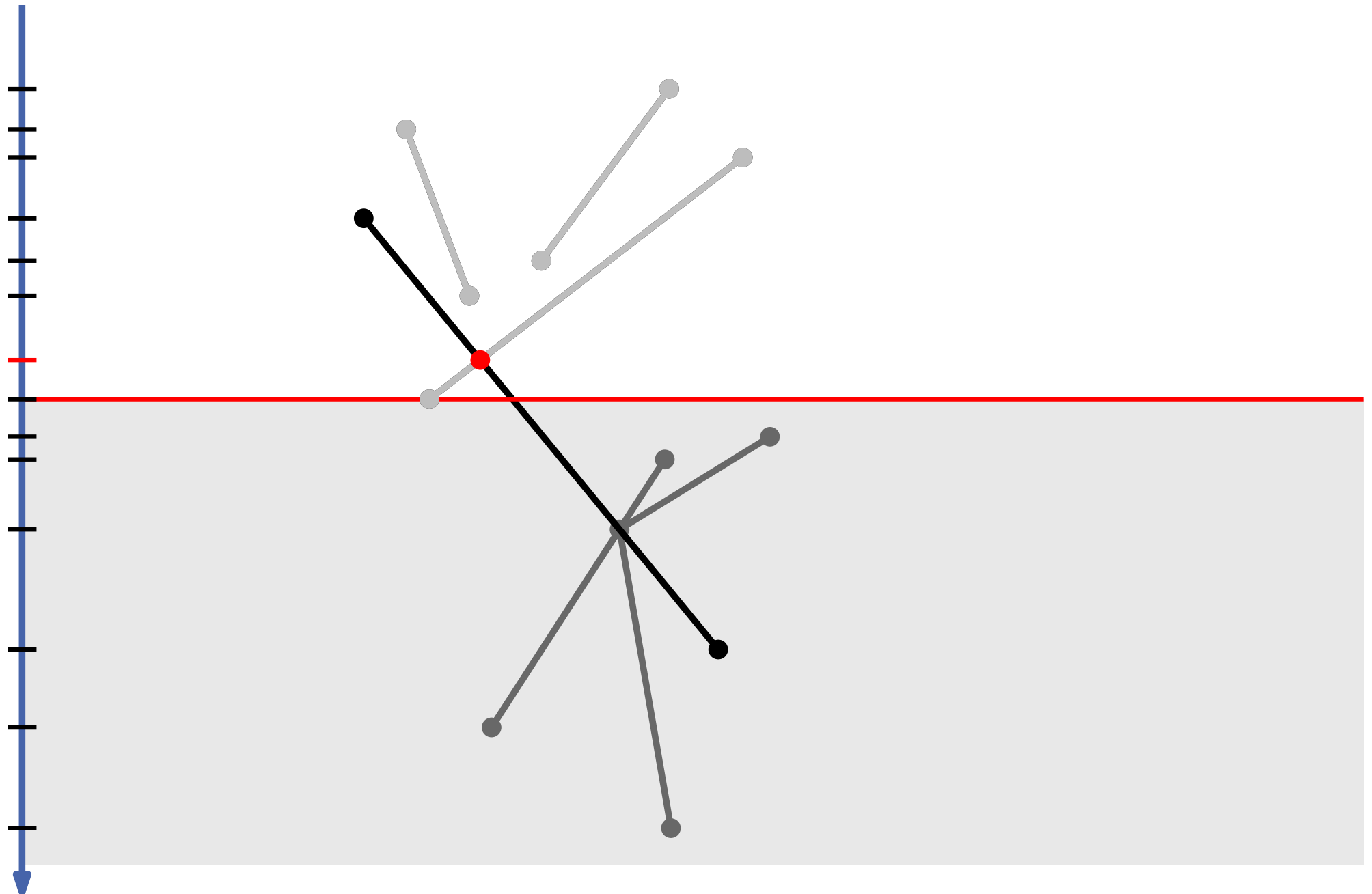
Das Sweep-Line Verfahren: Beispiel



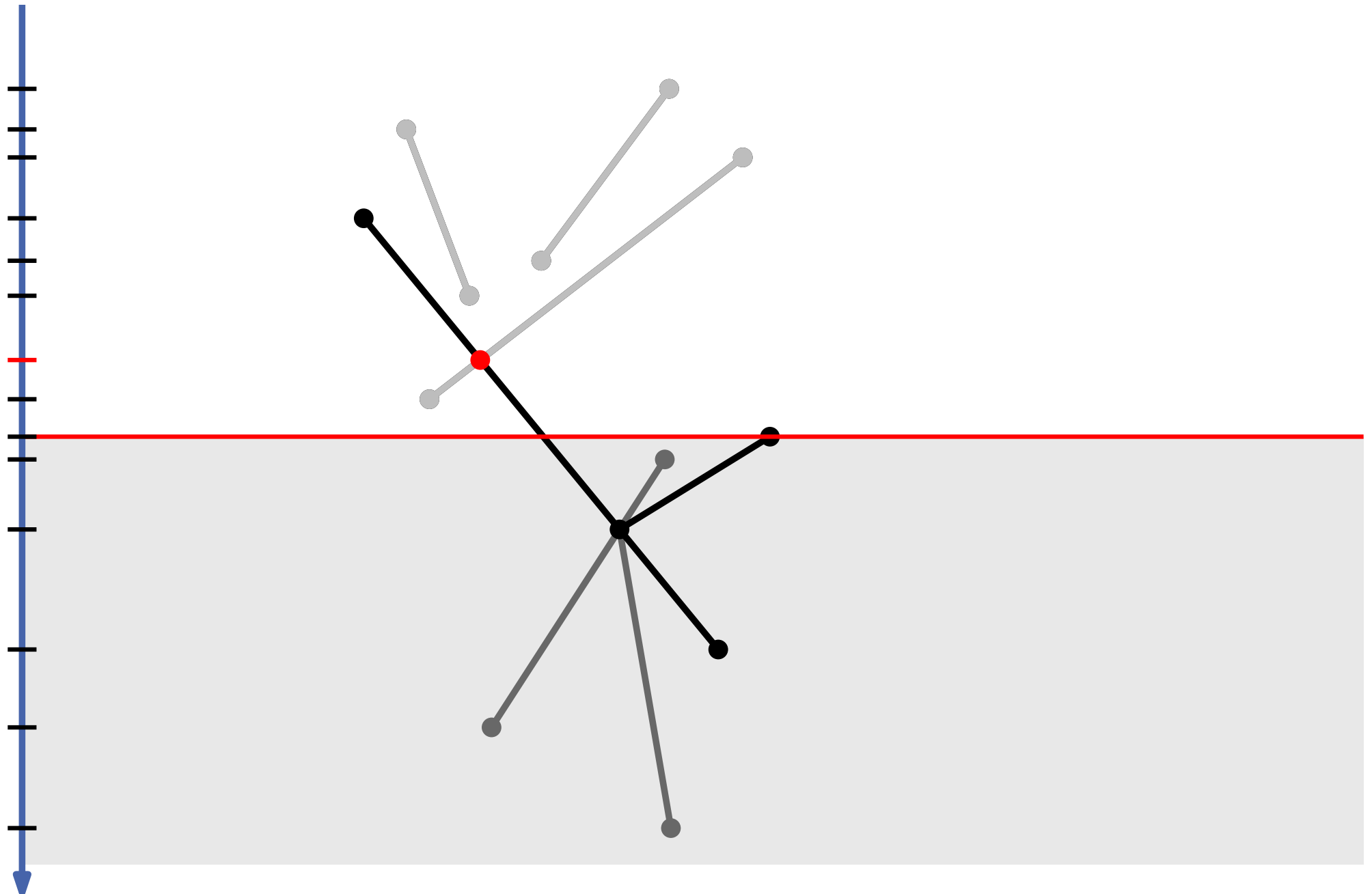
Das Sweep-Line Verfahren: Beispiel



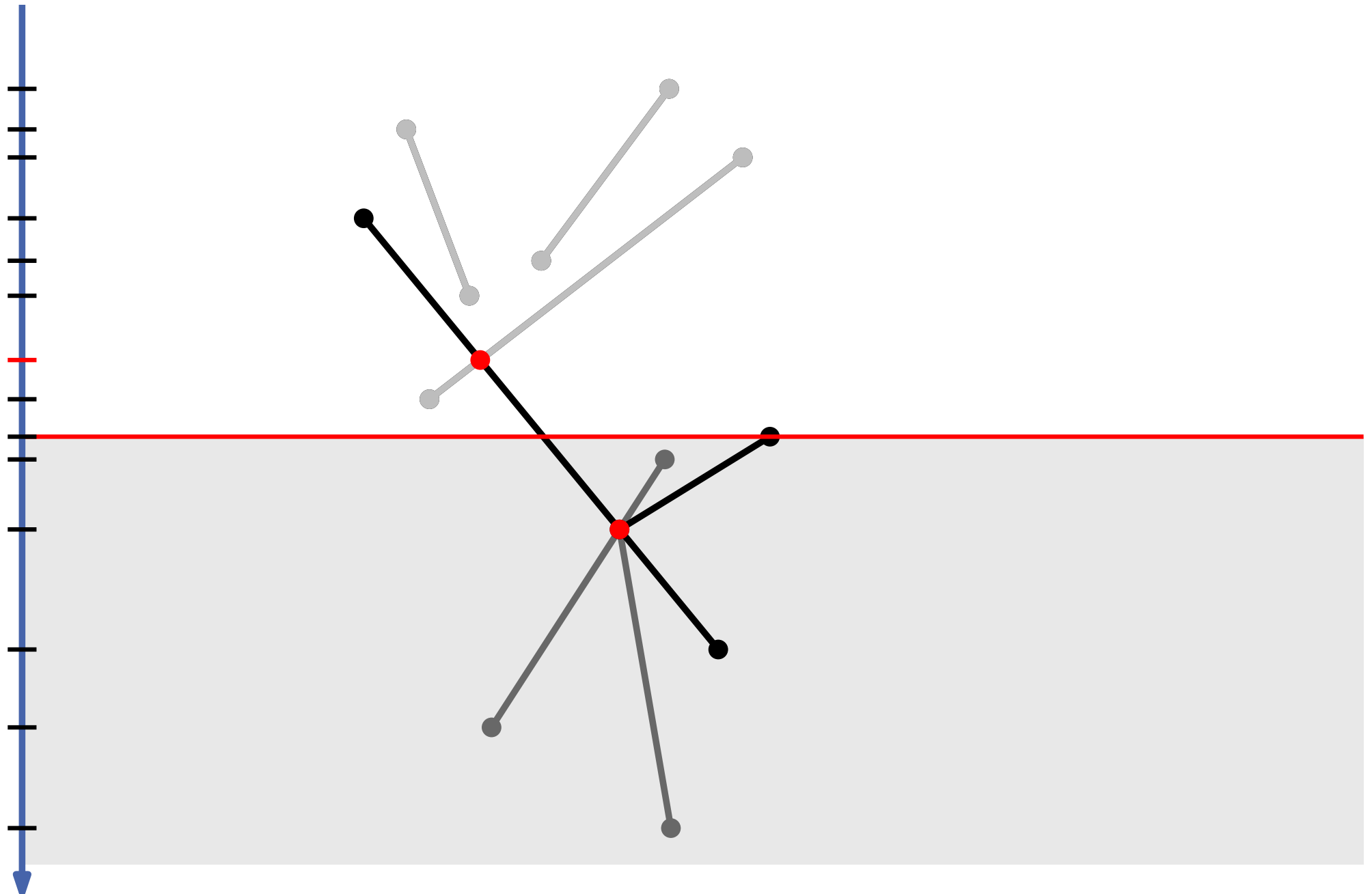
Das Sweep-Line Verfahren: Beispiel



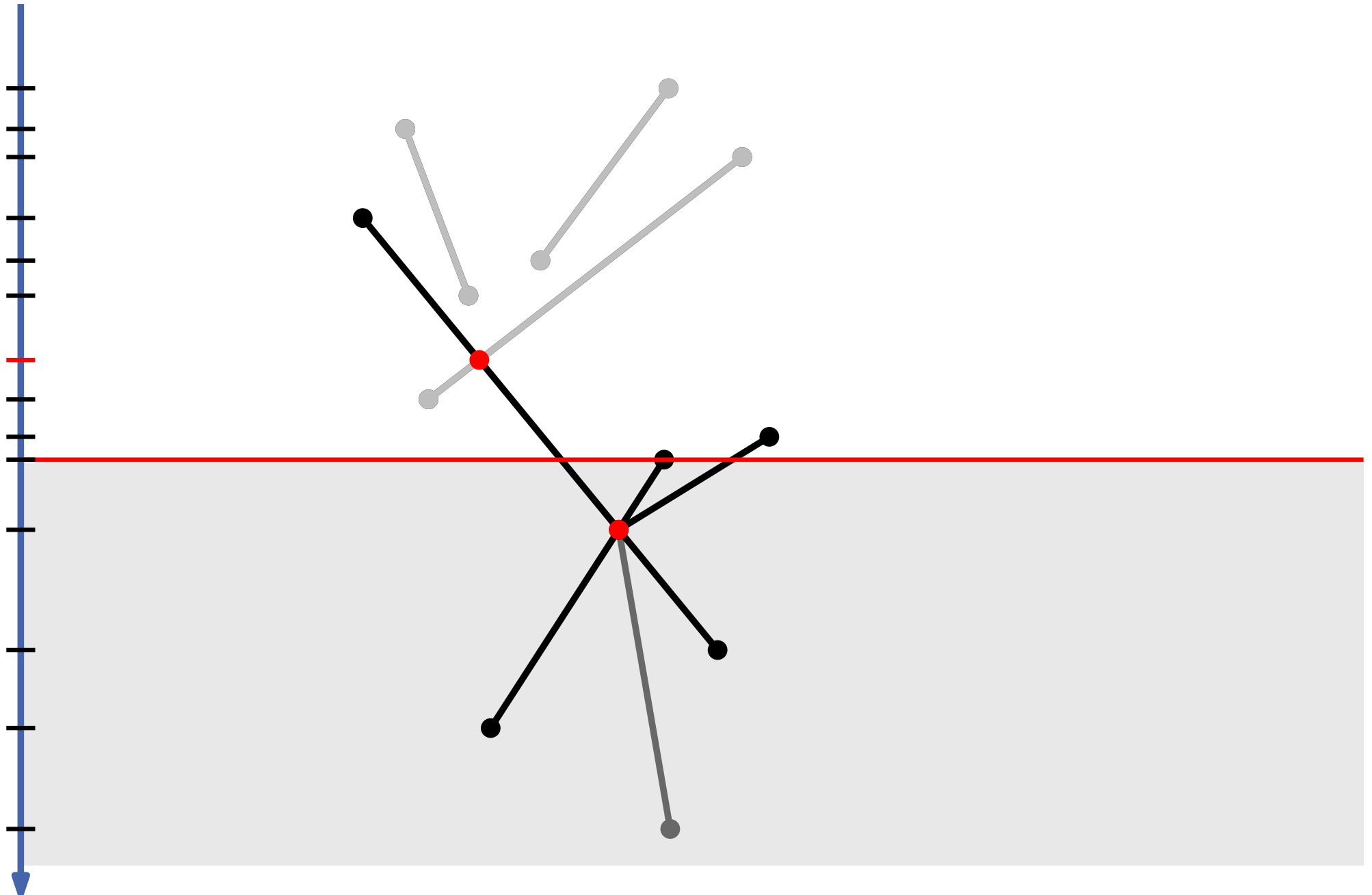
Das Sweep-Line Verfahren: Beispiel



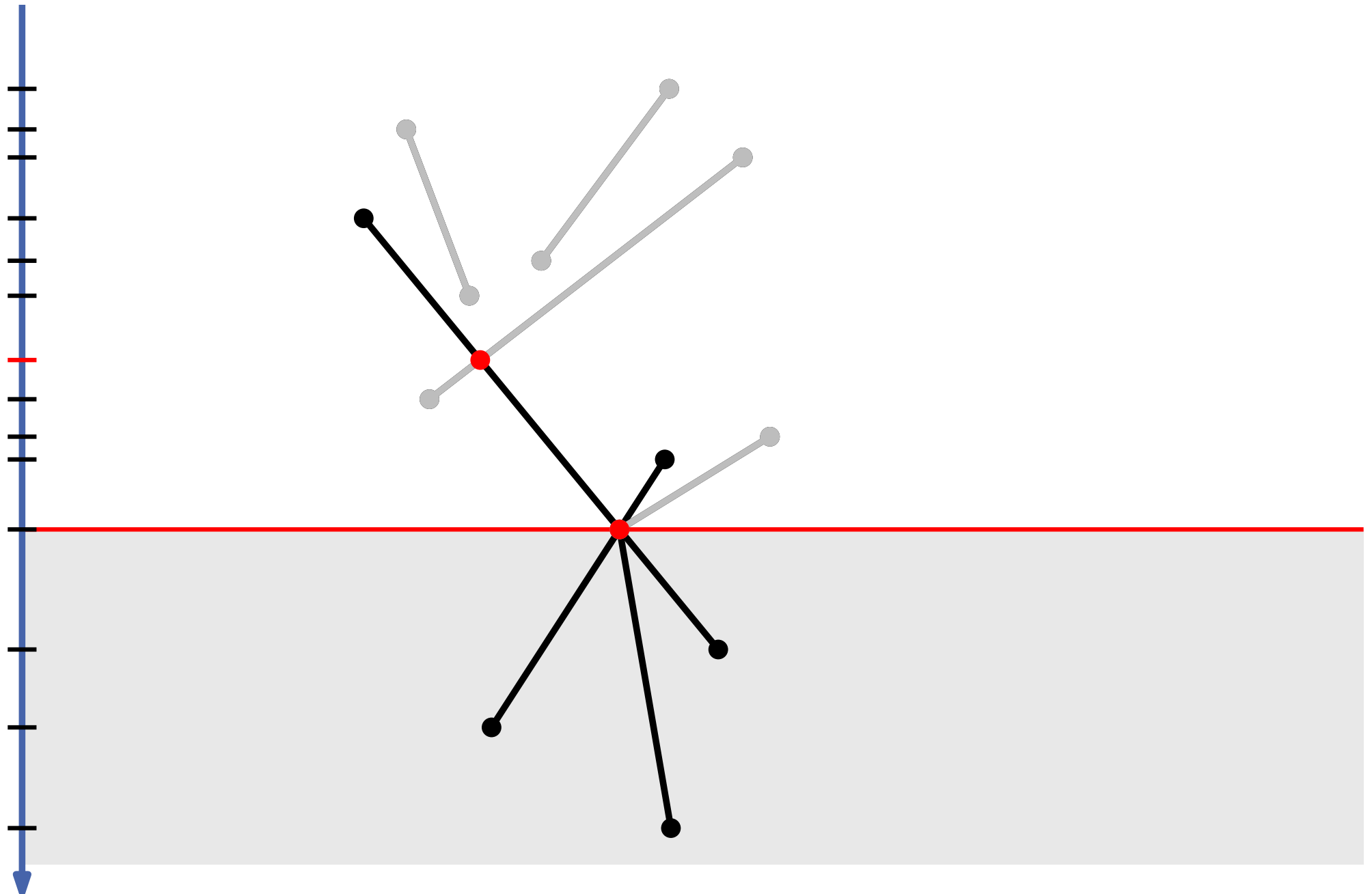
Das Sweep-Line Verfahren: Beispiel



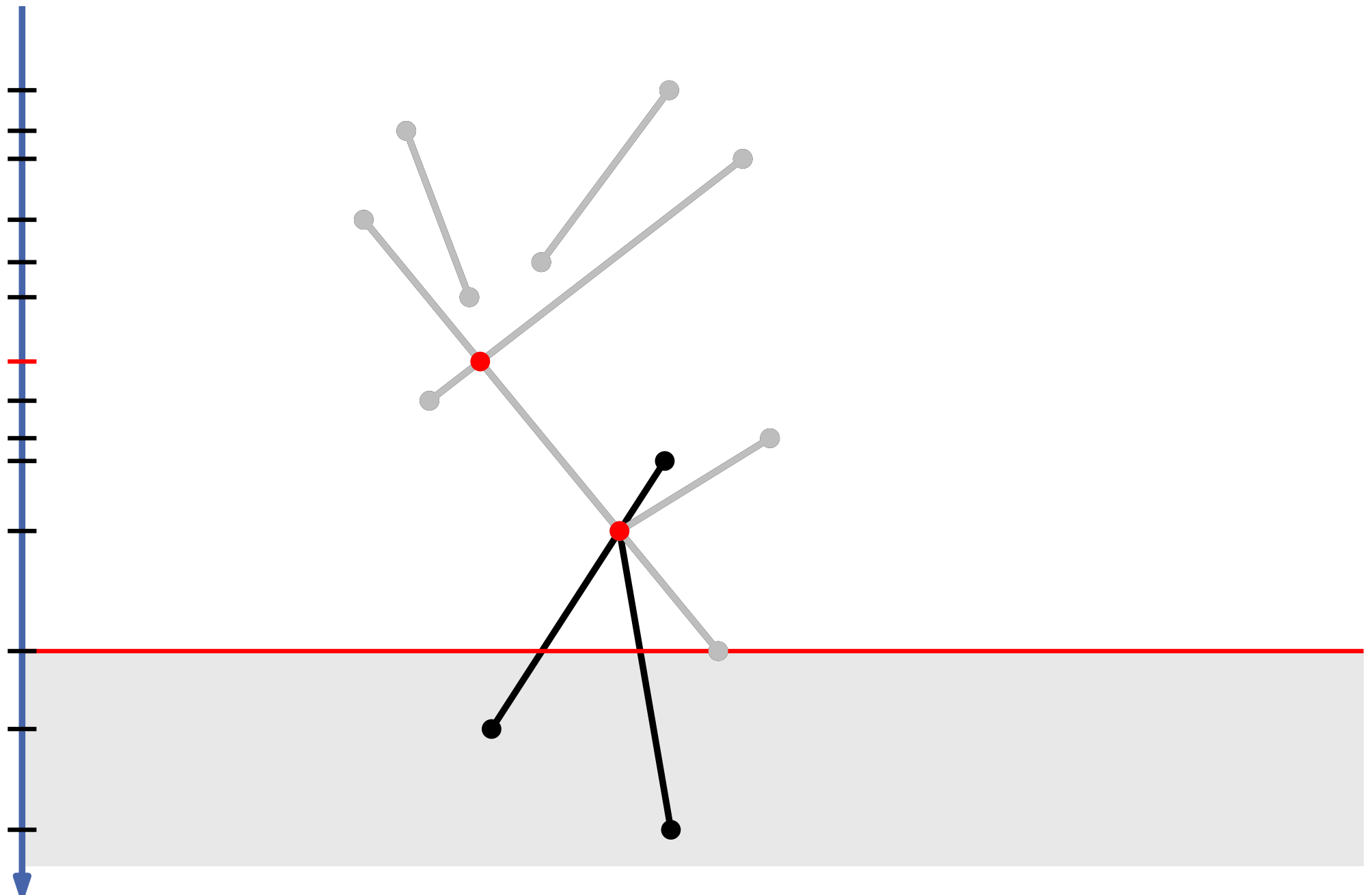
Das Sweep-Line Verfahren: Beispiel



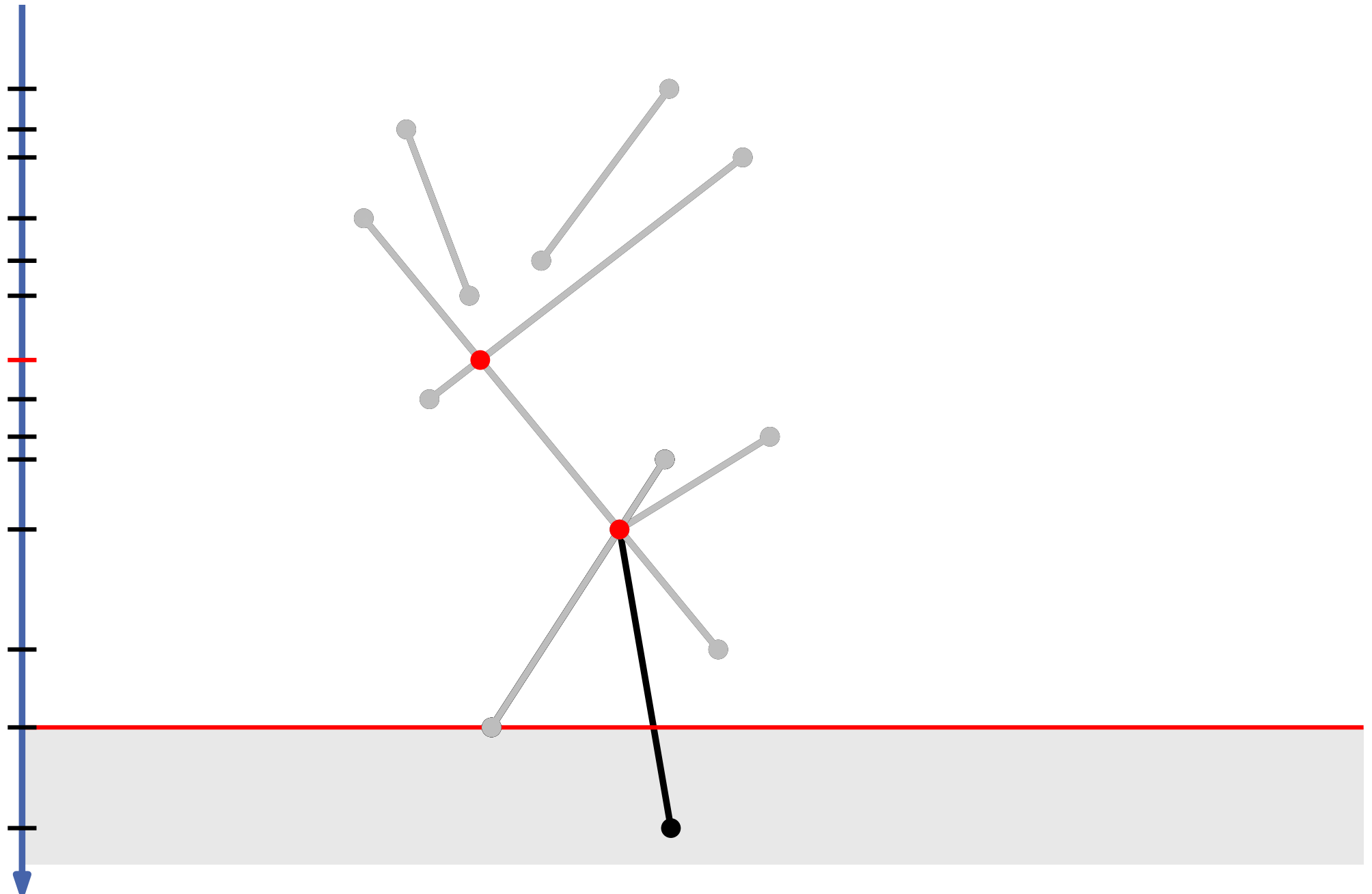
Das Sweep-Line Verfahren: Beispiel



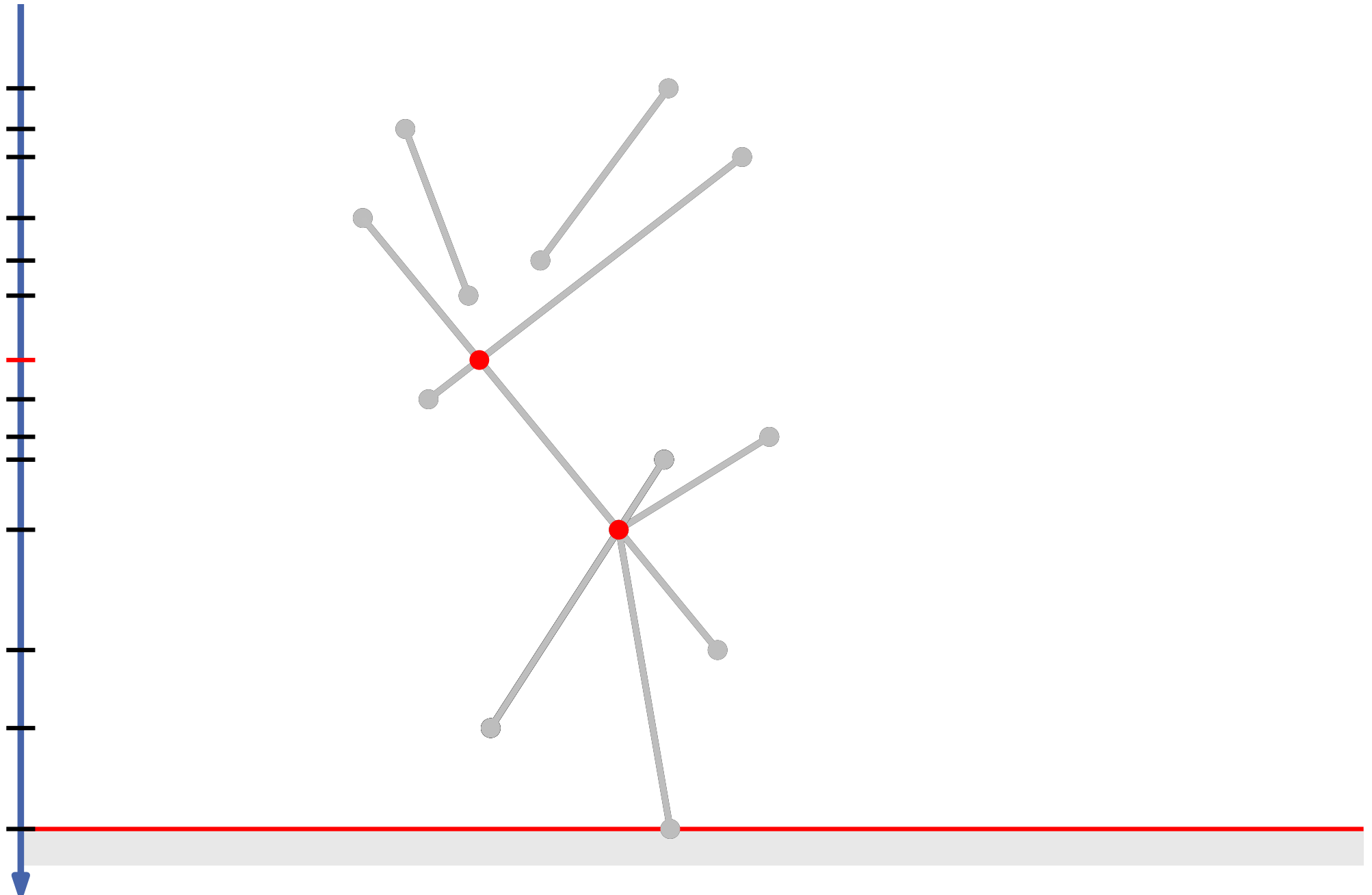
Das Sweep-Line Verfahren: Beispiel



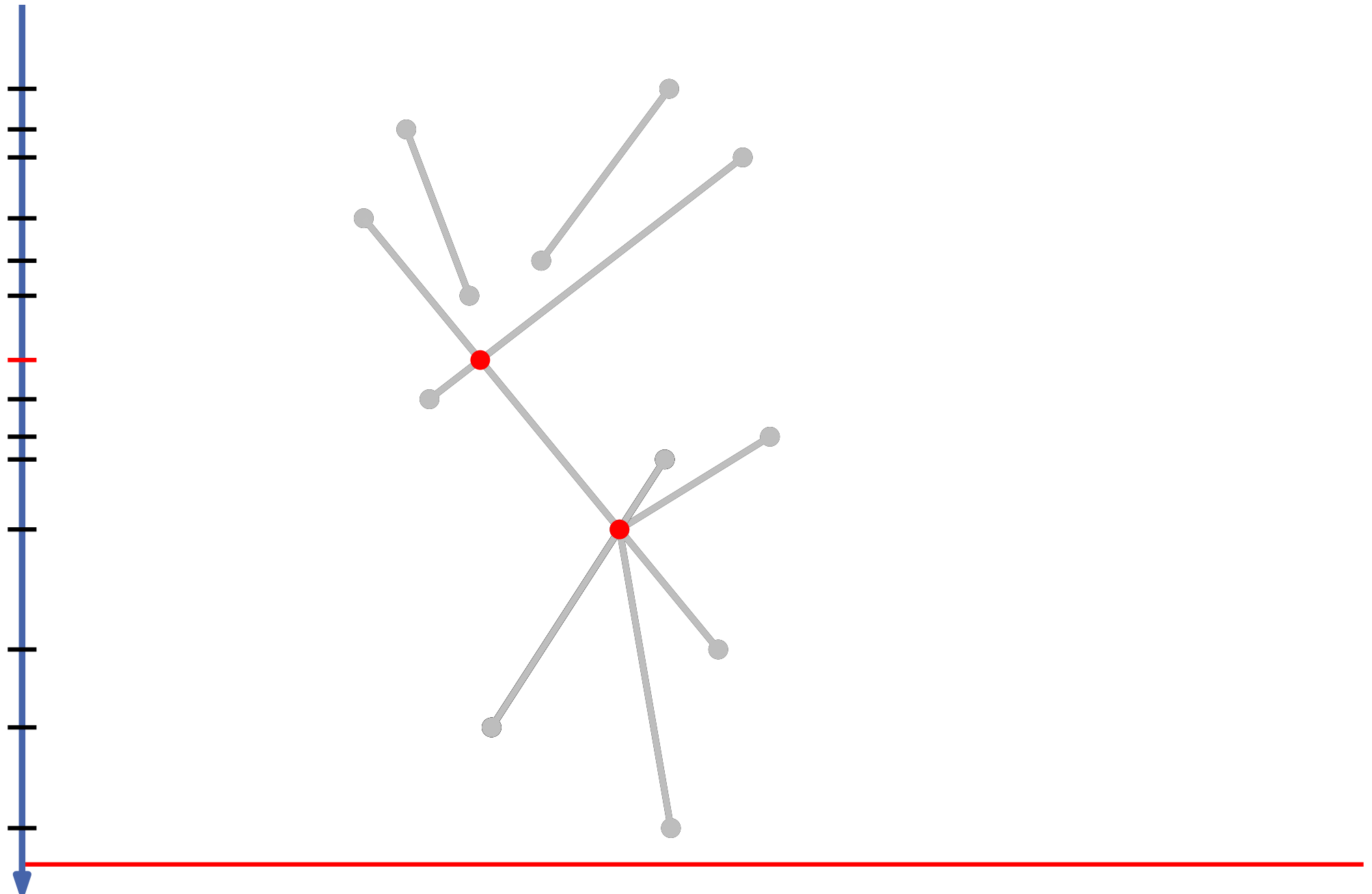
Das Sweep-Line Verfahren: Beispiel



Das Sweep-Line Verfahren: Beispiel

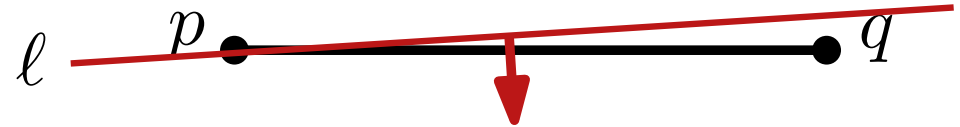


Das Sweep-Line Verfahren: Beispiel



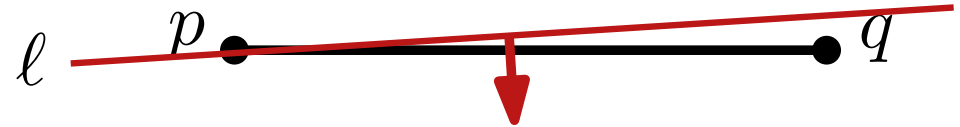
1.) Event Queue Q

- definiere $p \prec q \iff_{\text{def.}} y_p > y_q \vee (y_p = y_q \wedge x_p < x_q)$



1.) Event Queue \mathcal{Q}

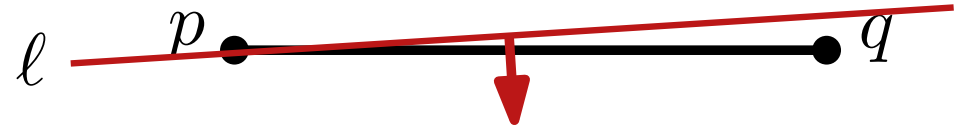
- definiere $p \prec q \iff_{\text{def.}} y_p > y_q \vee (y_p = y_q \wedge x_p < x_q)$



- speichere Events sortiert nach \prec in **balanciertem binärem Suchbaum**
→ AVL-Baum, Rot-Schwarz-Baum, ...

1.) Event Queue \mathcal{Q}

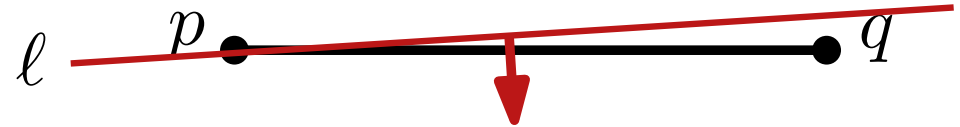
- definiere $p \prec q \iff_{\text{def.}} y_p > y_q \vee (y_p = y_q \wedge x_p < x_q)$



- speichere Events sortiert nach \prec in **balanciertem binärem Suchbaum**
→ AVL-Baum, Rot-Schwarz-Baum, ...
- Operationen insert, delete und nextEvent in $O(\log |\mathcal{Q}|)$ Zeit

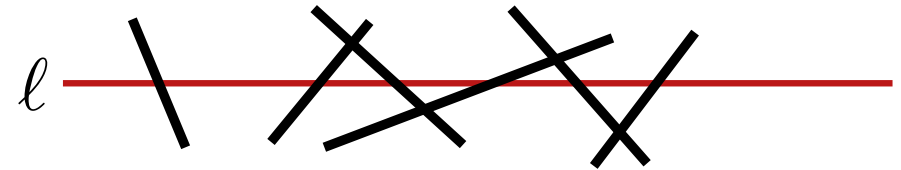
1.) Event Queue \mathcal{Q}

- definiere $p \prec q \iff_{\text{def.}} y_p > y_q \vee (y_p = y_q \wedge x_p < x_q)$



- speichere Events sortiert nach \prec in **balanciertem binärem Suchbaum**
→ AVL-Baum, Rot-Schwarz-Baum, ...
- Operationen insert, delete und nextEvent in $O(\log |\mathcal{Q}|)$ Zeit

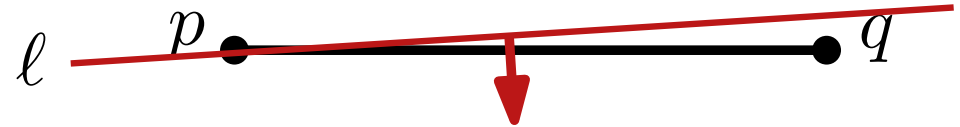
2.) Sweep-Line Status \mathcal{T}



- speichere von ℓ geschnittene Strecken geordnet von links nach rechts

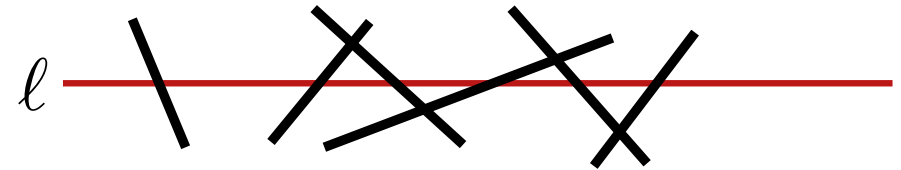
1.) Event Queue \mathcal{Q}

- definiere $p \prec q \iff_{\text{def.}} y_p > y_q \vee (y_p = y_q \wedge x_p < x_q)$



- speichere Events sortiert nach \prec in **balanciertem binärem Suchbaum**
→ AVL-Baum, Rot-Schwarz-Baum, ...
- Operationen insert, delete und nextEvent in $O(\log |Q|)$ Zeit

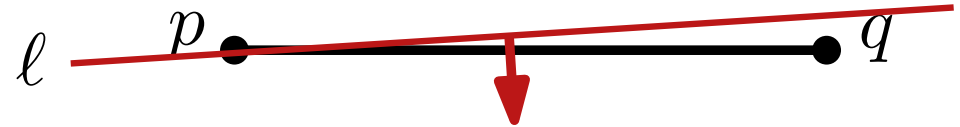
2.) Sweep-Line Status \mathcal{T}



- speichere von ℓ geschnittene Strecken geordnet von links nach rechts
- benötigte Operationen insert, delete, findNeighbor

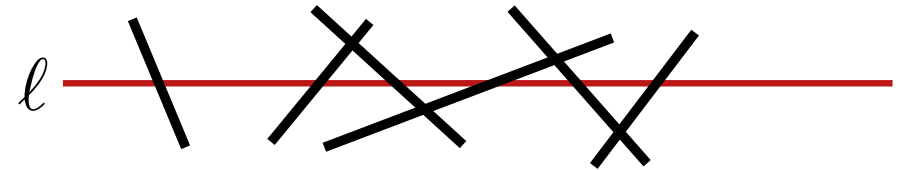
1.) Event Queue \mathcal{Q}

- definiere $p \prec q \iff_{\text{def.}} y_p > y_q \vee (y_p = y_q \wedge x_p < x_q)$



- speichere Events sortiert nach \prec in **balanciertem binärem Suchbaum**
→ AVL-Baum, Rot-Schwarz-Baum, ...
- Operationen insert, delete und nextEvent in $O(\log |Q|)$ Zeit

2.) Sweep-Line Status \mathcal{T}



- speichere von l geschnittene Strecken geordnet von links nach rechts
- benötigte Operationen insert, delete, findNeighbor
- ebenfalls balancierter binärer Suchbaum mit Strecken in den Blättern!

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

$Q \leftarrow \emptyset; \mathcal{T} \leftarrow \emptyset$

foreach $s \in S$ **do**

Q .insert(upperEndPoint(s))
 Q .insert(lowerEndPoint(s))

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

$p \leftarrow Q$.nextEvent()
 Q .deleteEvent(p)
 handleEvent(p)

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

$Q \leftarrow \emptyset; \mathcal{T} \leftarrow \emptyset$

foreach $s \in S$ **do**

$Q.insert(\text{upperEndPoint}(s))$

$Q.insert(\text{lowerEndPoint}(s))$

Was passiert mit
Duplikaten?

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

$p \leftarrow Q.nextEvent()$

$Q.deleteEvent(p)$

$handleEvent(p)$

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

$Q \leftarrow \emptyset; \mathcal{T} \leftarrow \emptyset$

foreach $s \in S$ **do**

$Q.insert(\text{upperEndPoint}(s))$

$Q.insert(\text{lowerEndPoint}(s))$

Was passiert mit
Duplikaten?

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

$p \leftarrow Q.nextEvent()$

$Q.deleteEvent(p)$

$handleEvent(p)$

Hier versteckt sich der Kern
des Algorithmus!

handleEvent(p)

$U(p) \leftarrow$ Strecken mit p oberer Endpunkt

$L(p) \leftarrow$ Strecken mit p unterer Endpunkt

$C(p) \leftarrow$ Strecken mit p innerer Punkt

if $|U(p) \cup L(p) \cup C(p)| \geq 2$ **then**

└ gebe p und $U(p) \cup L(p) \cup C(p)$ aus

entferne $L(p) \cup C(p)$ aus \mathcal{T}

füge $U(p) \cup C(p)$ in \mathcal{T} ein

if $U(p) \cup C(p) = \emptyset$ **then** // s_l und s_r Nachbarn von p in \mathcal{T}

└ $Q \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p

else // s' und s'' linkeste und rechteste Strecke in $U(p) \cup C(p)$

└ $Q \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p

└ $Q \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

handleEvent(p)

$U(p) \leftarrow$ Strecken mit p oberer Endpunkt

$L(p) \leftarrow$ Strecken mit p unterer Endpunkt

$C(p) \leftarrow$ Strecken mit p innerer Punkt

if $|U(p) \cup L(p) \cup C(p)| \geq 2$ **then**

└ gebe p und $U(p) \cup L(p) \cup C(p)$ aus

entferne $L(p) \cup C(p)$ aus \mathcal{T}

füge $U(p) \cup C(p)$ in \mathcal{T} ein

if $U(p) \cup C(p) = \emptyset$ **then**

// s_l und s_r Nachbarn von p in \mathcal{T}

└ $Q \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p

else // s' und s'' linkeste und rechteste Strecke in $U(p) \cup C(p)$

└ $Q \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p

└ $Q \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

mit p in Q gespeichert

handleEvent(p)

$U(p) \leftarrow$ Strecken mit p oberer Endpunkt

mit p in Q gespeichert

$L(p) \leftarrow$ Strecken mit p unterer Endpunkt

$C(p) \leftarrow$ Strecken mit p innerer Punkt

Nachbarn in \mathcal{T}

if $|U(p) \cup L(p) \cup C(p)| \geq 2$ **then**

└ gebe p und $U(p) \cup L(p) \cup C(p)$ aus

entferne $L(p) \cup C(p)$ aus \mathcal{T}

füge $U(p) \cup C(p)$ in \mathcal{T} ein

if $U(p) \cup C(p) = \emptyset$ **then**

// s_l und s_r Nachbarn von p in \mathcal{T}

└ $Q \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p

else // s' und s'' linkeste und rechteste Strecke in $U(p) \cup C(p)$

└ $Q \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p

└ $Q \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

handleEvent(p)

$U(p) \leftarrow$ Strecken mit p oberer Endpunkt

mit p in Q gespeichert

$L(p) \leftarrow$ Strecken mit p unterer Endpunkt

$C(p) \leftarrow$ Strecken mit p innerer Punkt

Nachbarn in \mathcal{T}

if $|U(p) \cup L(p) \cup C(p)| \geq 2$ **then**

┌ gebe p und $U(p) \cup L(p) \cup C(p)$ aus

entferne $L(p) \cup C(p)$ aus \mathcal{T}

füge $U(p) \cup C(p)$ in \mathcal{T} ein

Entfernen und Einfügen
dreht $C(p)$ um

if $U(p) \cup C(p) = \emptyset$ **then**

// s_l und s_r Nachbarn von p in \mathcal{T}

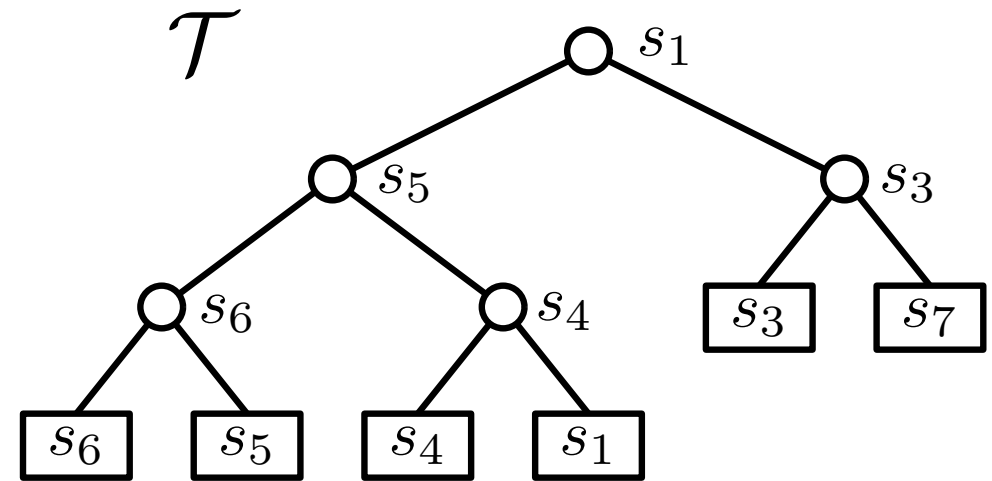
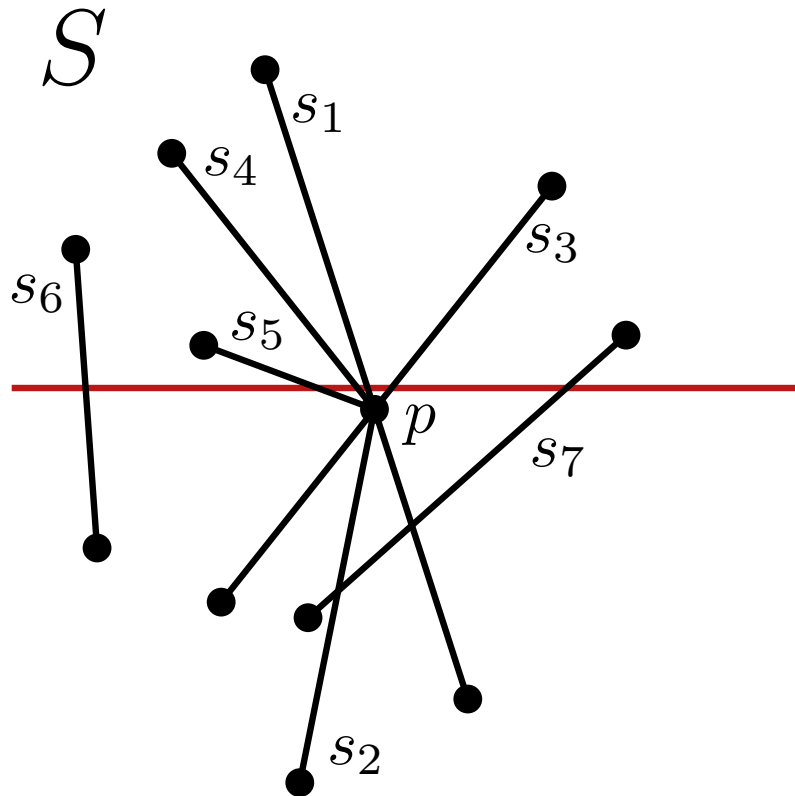
┌ $Q \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p

else // s' und s'' linkeste und rechteste Strecke in $U(p) \cup C(p)$

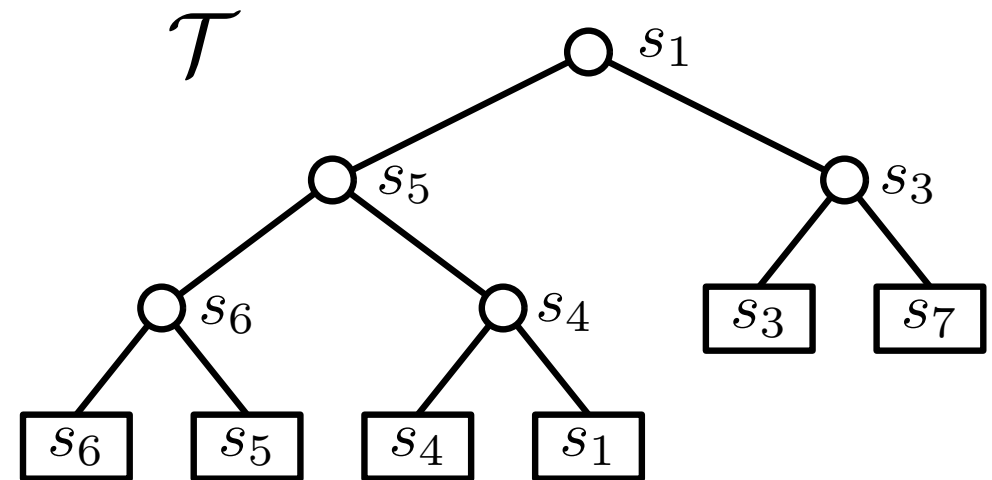
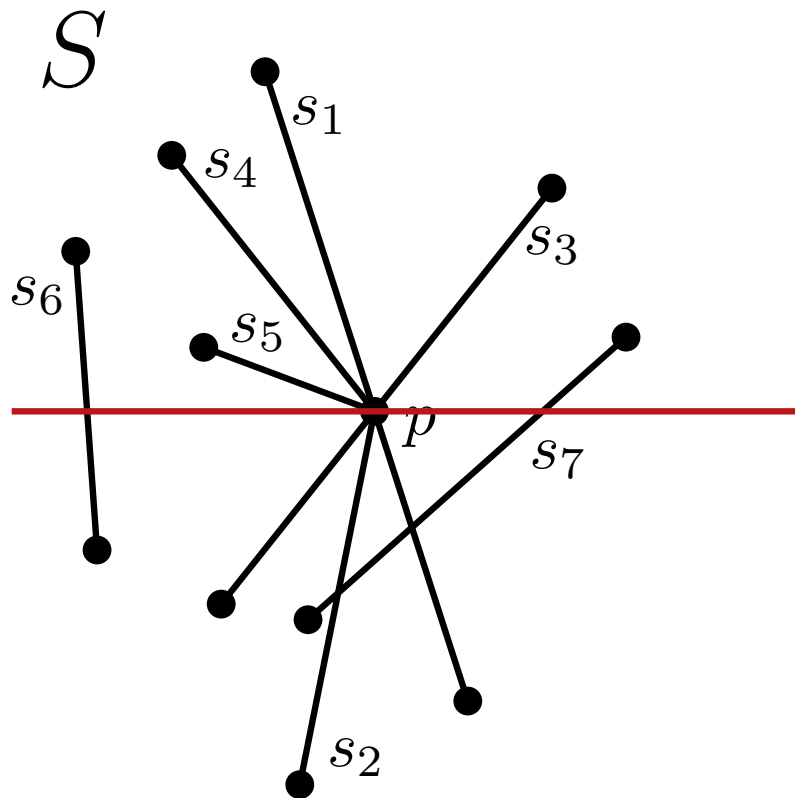
┌ $Q \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p

┌ $Q \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

Was passiert genau?



Was passiert genau?

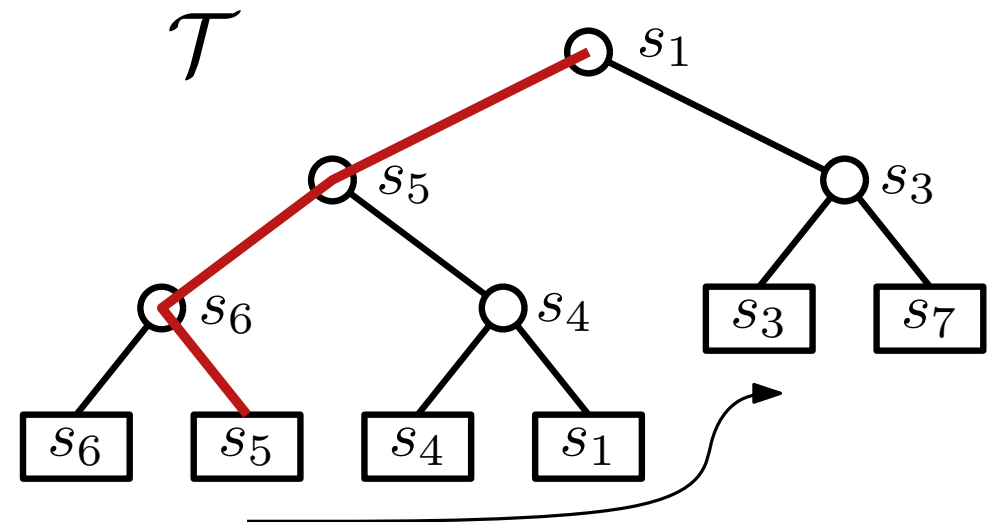
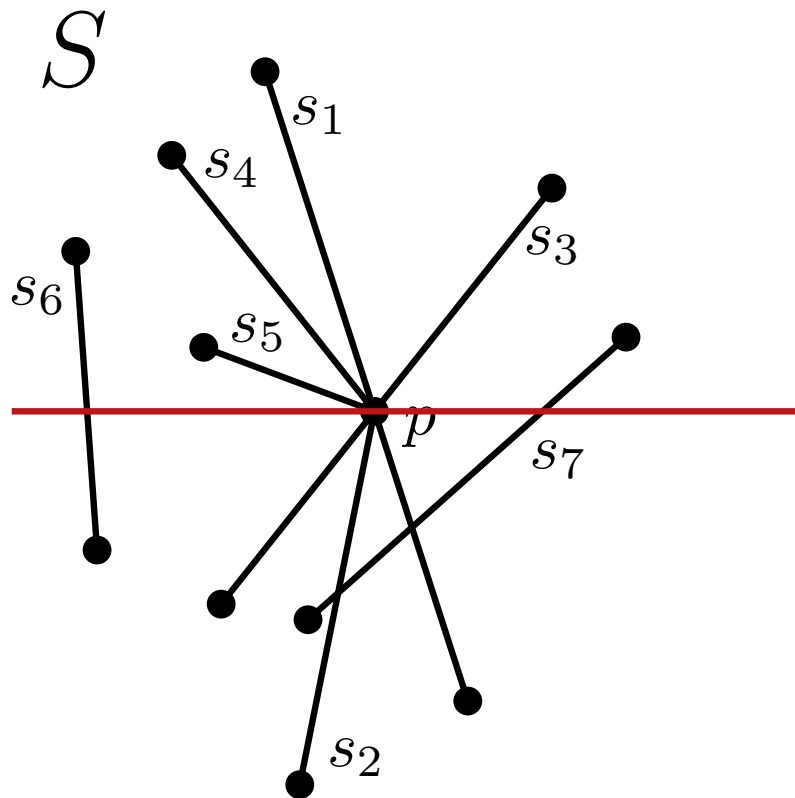


$$U(p) = \{s_2\}$$

$$L(p) =$$

$$C(p) =$$

Was passiert genau?

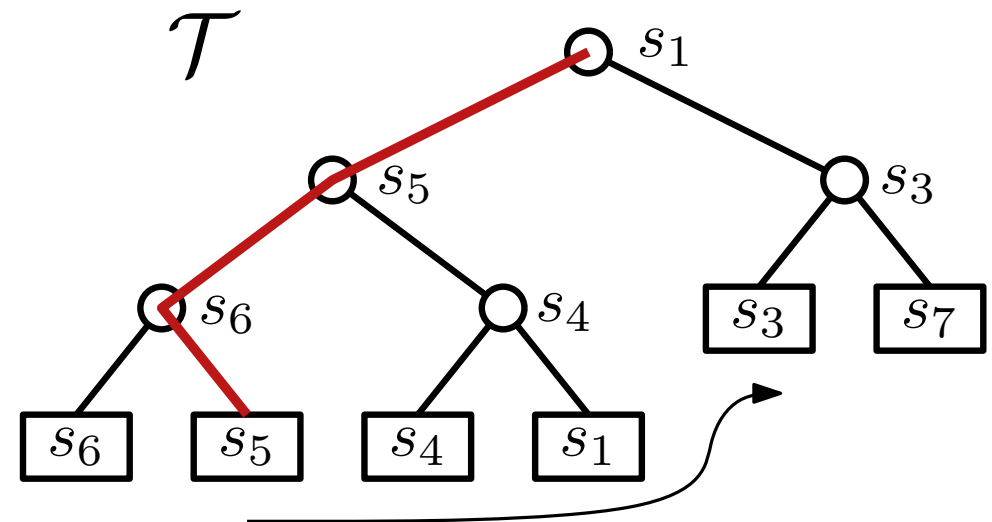
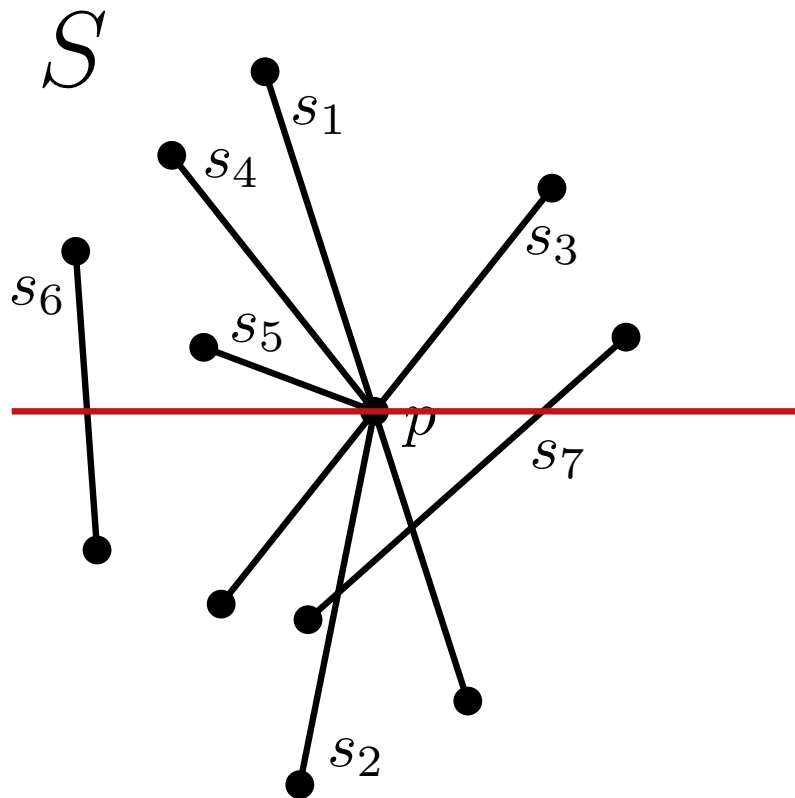


$$U(p) = \{s_2\}$$

$$L(p) =$$

$$C(p) =$$

Was passiert genau?

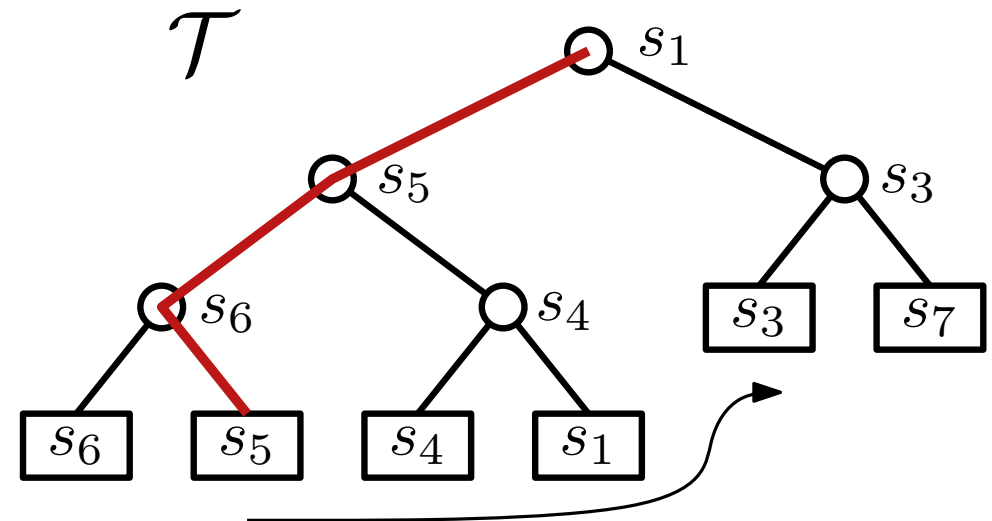
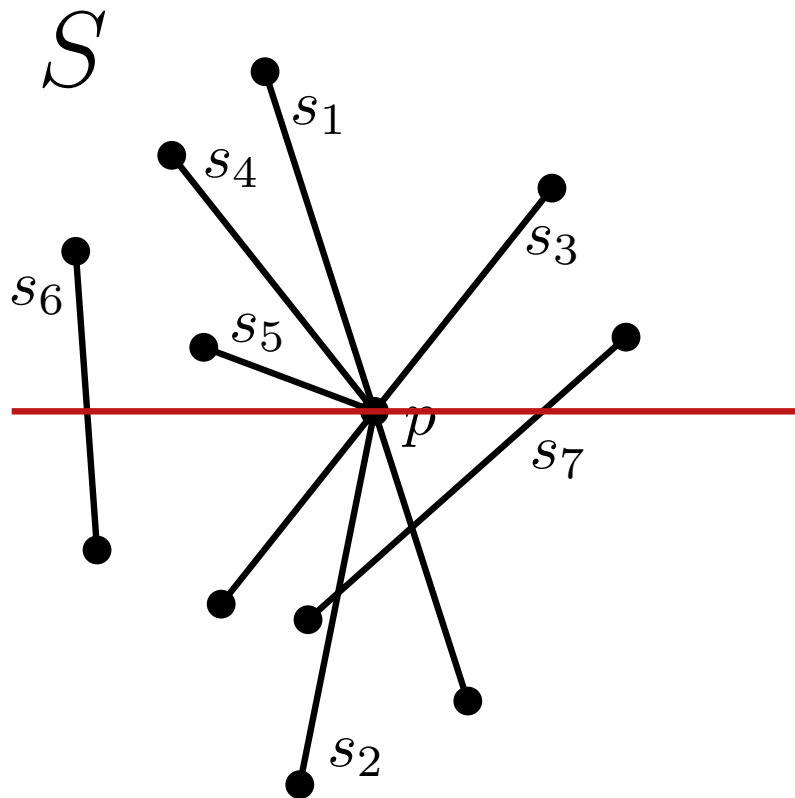


$$U(p) = \{s_2\}$$

$$L(p) = \{s_4, s_5\}$$

$$C(p) = \{s_1, s_3\}$$

Was passiert genau?



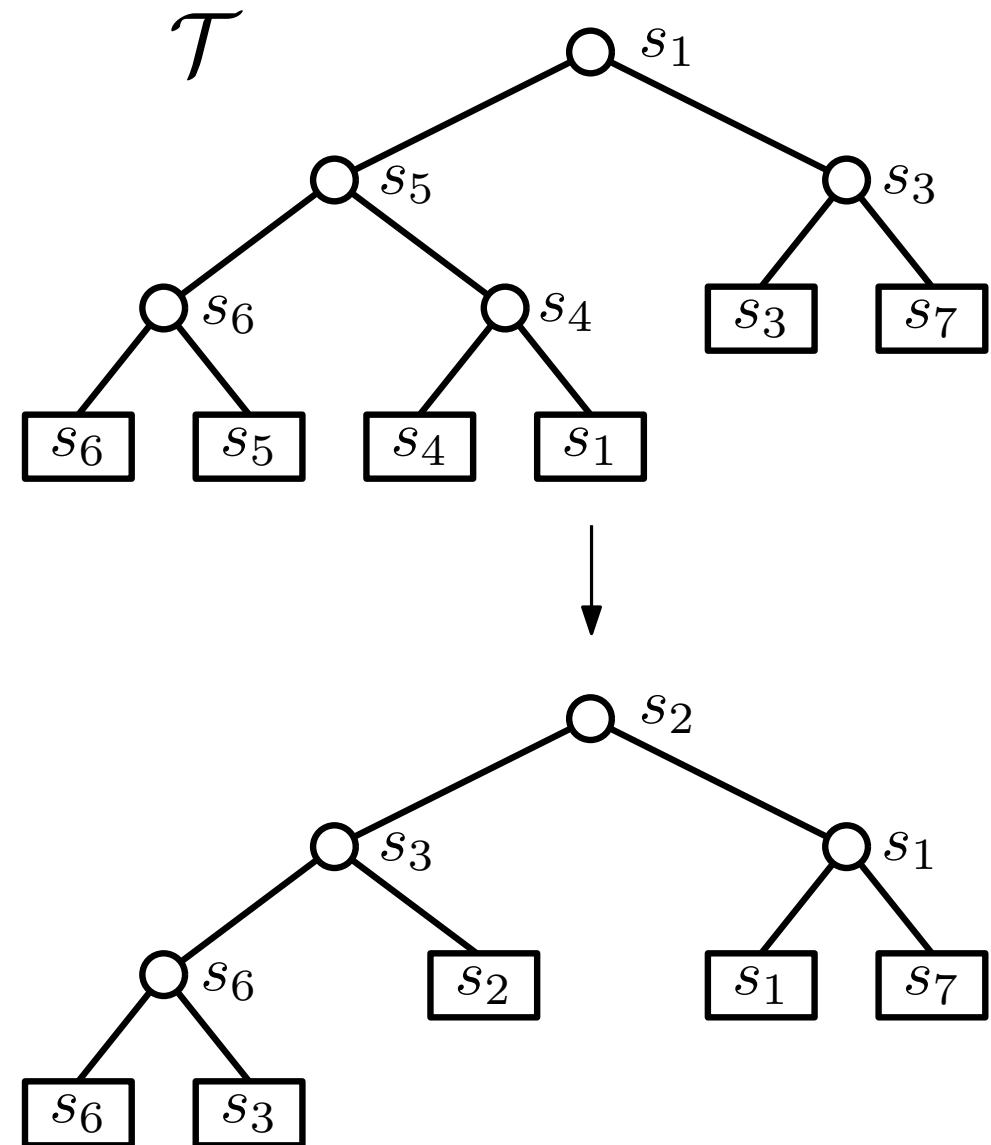
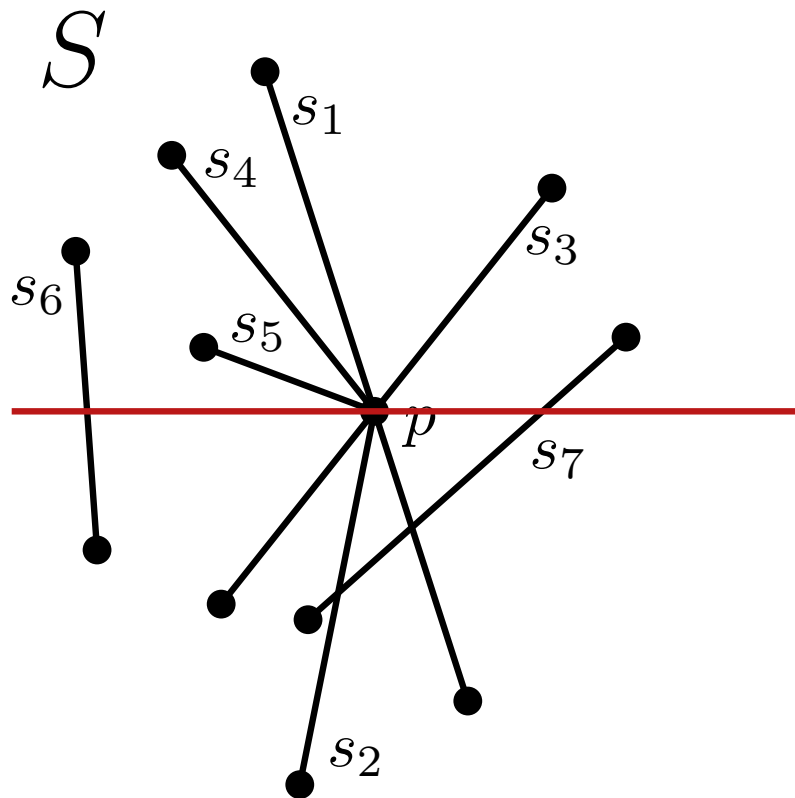
$$U(p) = \{s_2\}$$

$$L(p) = \{s_4, s_5\}$$

$$C(p) = \{s_1, s_3\}$$

Gib $(p, \{s_1, s_2, s_3, s_4, s_5\})$ aus

Was passiert genau?



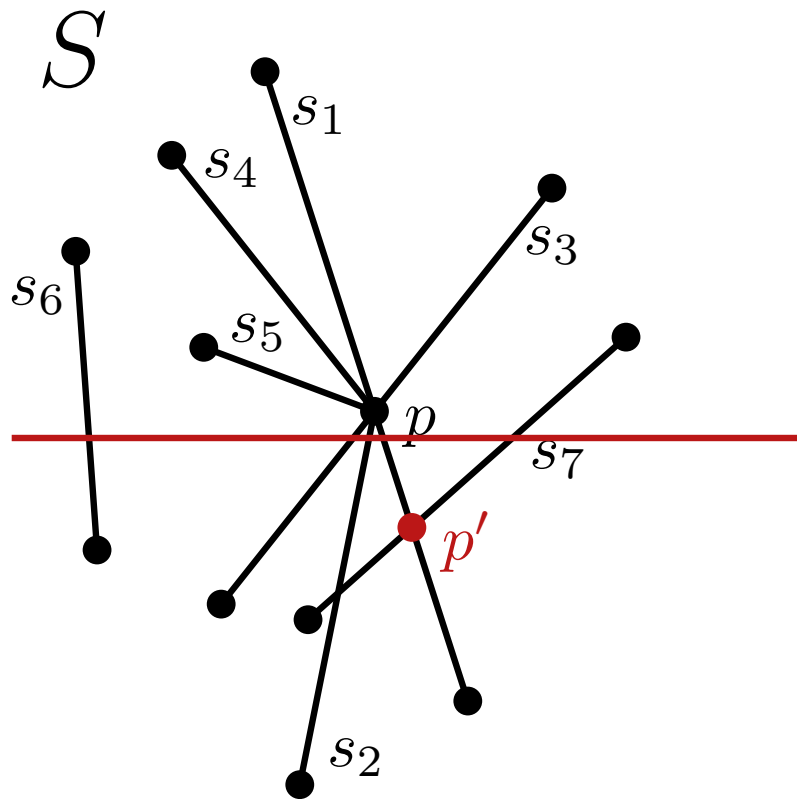
$$U(p) = \{s_2\}$$

$$L(p) = \{s_4, s_5\}$$

$$C(p) = \{s_1, s_3\}$$

Lösche $L(p) \cup C(p)$; füge $U(p) \cup C(p)$ ein

Was passiert genau?

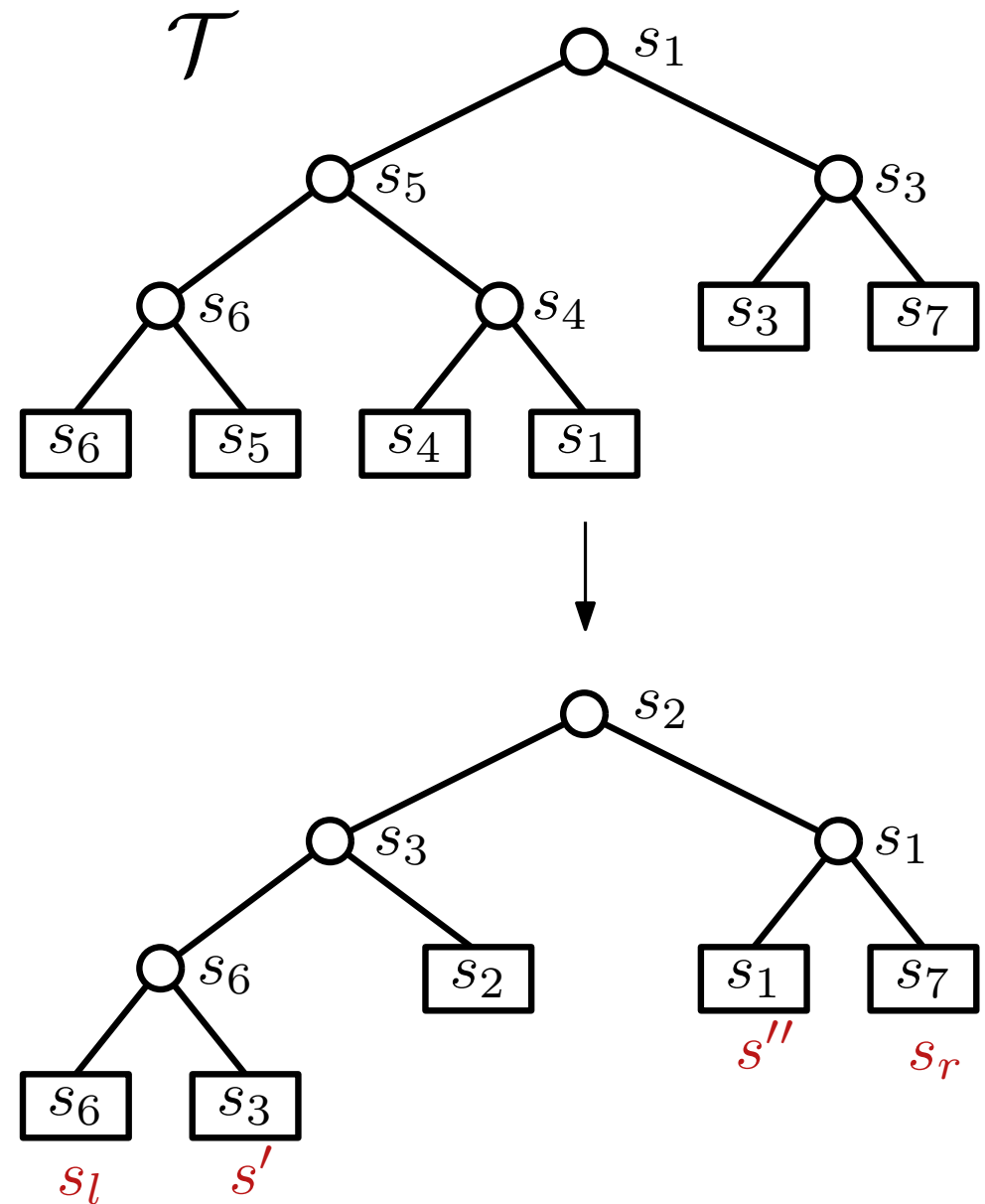


$$U(p) = \{s_2\}$$

$$L(p) = \{s_4, s_5\}$$

$$C(p) = \{s_1, s_3\}$$

Füge Event $p' = s_1 \times s_7$ in Q ein



Lemma 1: Algorithmus FindIntersections findet alle Schnittpunkte und die beteiligten Strecken.

Lemma 1: Algorithmus FindIntersections findet alle Schnittpunkte und die beteiligten Strecken.

Beweis:

Induktion über die Reihenfolge der Events.

Sei p ein Schnittpunkt und alle Schnittpunkte $q \prec p$ seien bereits korrekt berechnet.

Fall 1: p ist Streckenendpunkt

- p wurde am Anfang in Q eingefügt
- $U(p)$ an p gespeichert
- $L(p)$ und $C(p)$ in \mathcal{T} vorhanden

Lemma 1: Algorithmus FindIntersections findet alle Schnittpunkte und die beteiligten Strecken.

Beweis:

Induktion über die Reihenfolge der Events.

Sei p ein Schnittpunkt und alle Schnittpunkte $q \prec p$ seien bereits korrekt berechnet.

Fall 1: p ist Streckenendpunkt

- p wurde am Anfang in Q eingefügt
- $U(p)$ an p gespeichert
- $L(p)$ und $C(p)$ in \mathcal{T} vorhanden

Fall 2: p ist kein Streckenendpunkt

Überlegen Sie warum p in Q sein muss!

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

```
 $Q \leftarrow \emptyset; \mathcal{T} \leftarrow \emptyset$ 
```

```
foreach  $s \in S$  do
```

```
   $Q$ .insert(upperEndPoint( $s$ ))  
   $Q$ .insert(lowerEndPoint( $s$ ))
```

```
while  $Q \neq \emptyset$  do
```

```
   $p \leftarrow Q$ .nextEvent()  
   $Q$ .deleteEvent( $p$ )  
  handleEvent( $p$ )
```

Laufzeitanalyse

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

```
 $Q \leftarrow \emptyset; \mathcal{T} \leftarrow \emptyset$   $O(1)$ 
```

```
foreach  $s \in S$  do
```

```
   $Q$ .insert(upperEndPoint( $s$ ))  
   $Q$ .insert(lowerEndPoint( $s$ ))
```

```
while  $Q \neq \emptyset$  do
```

```
   $p \leftarrow Q$ .nextEvent()  
   $Q$ .deleteEvent( $p$ )  
  handleEvent( $p$ )
```

Laufzeitanalyse

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

$Q \leftarrow \emptyset; \mathcal{T} \leftarrow \emptyset$ $O(1)$

foreach $s \in S$ **do**

$Q.insert(\text{upperEndPoint}(s))$ $O(n \log n)$
 $Q.insert(\text{lowerEndPoint}(s))$

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

$p \leftarrow Q.nextEvent()$
 $Q.deleteEvent(p)$
 $handleEvent(p)$

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

$Q \leftarrow \emptyset; \mathcal{T} \leftarrow \emptyset$ $O(1)$

foreach $s \in S$ **do**

$Q.insert(\text{upperEndPoint}(s))$ $O(n \log n)$
 $Q.insert(\text{lowerEndPoint}(s))$

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

$p \leftarrow Q.nextEvent()$ $O(\log |Q|)$
 $Q.deleteEvent(p)$
 $handleEvent(p)$

Laufzeitanalyse

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

$Q \leftarrow \emptyset; \mathcal{T} \leftarrow \emptyset$ $O(1)$

foreach $s \in S$ **do**

$Q.insert(\text{upperEndPoint}(s))$ $O(n \log n)$
 $Q.insert(\text{lowerEndPoint}(s))$

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

$p \leftarrow Q.nextEvent()$ $O(\log |Q|)$
 $Q.deleteEvent(p)$
 $handleEvent(p)$?

handleEvent(p)

$U(p) \leftarrow$ Strecken mit p oberer Endpunkt

$L(p) \leftarrow$ Strecken mit p unterer Endpunkt

$C(p) \leftarrow$ Strecken mit p innerer Punkt

if $|U(p) \cup L(p) \cup C(p)| \geq 2$ **then**

└ gebe p und $U(p) \cup L(p) \cup C(p)$ aus

entferne $L(p) \cup C(p)$ aus \mathcal{T}

füge $U(p) \cup C(p)$ in \mathcal{T} ein

if $U(p) \cup C(p) = \emptyset$ **then** // s_l und s_r Nachbarn von p in \mathcal{T}

└ $Q \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p

else // s' und s'' linkeste und rechteste Strecke in $U(p) \cup C(p)$

└ $Q \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p

└ $Q \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

Lemma 2: Algorithmus FindIntersections hat eine Laufzeit von $O(n \log n + I \log n)$, wobei I die Anzahl der Schnittpunkte ist.

Satz 1: Sei S eine Menge von n Strecken in der Ebene. Dann können alle Schnittpunkte in S zusammen mit den beteiligten Strecken in $O((n + I) \log n)$ Zeit und $O(?)$ Platz ausgegeben werden.

Satz 1: Sei S eine Menge von n Strecken in der Ebene. Dann können alle Schnittpunkte in S zusammen mit den beteiligten Strecken in $O((n + I) \log n)$ Zeit und $O(?)$ Platz ausgegeben werden.

Beweis:

- Korrektheit ✓
- Laufzeit ✓
- Speicherplatz

Satz 1: Sei S eine Menge von n Strecken in der Ebene. Dann können alle Schnittpunkte in S zusammen mit den beteiligten Strecken in $O((n + I) \log n)$ Zeit und $O(?)$ Platz ausgegeben werden.

Beweis:

- Korrektheit ✓
- Laufzeit ✓
- Speicherplatz

Überlegen Sie wie viel Platz die Datenstrukturen benötigen!

Satz 1: Sei S eine Menge von n Strecken in der Ebene. Dann können alle Schnittpunkte in S zusammen mit den beteiligten Strecken in $O((n + I) \log n)$ Zeit und $O(n)$ Platz ausgegeben werden.

Beweis:

- Korrektheit ✓
- Laufzeit ✓
- Speicherplatz

Überlegen Sie wie viel Platz die Datenstrukturen benötigen!

- \mathcal{T} maximal n Elemente
- \mathcal{Q} maximal $O(n + I)$ Elemente
- Reduktion von \mathcal{Q} auf $O(n)$ Platz: s. Übung

Ist der Sweep-Line Algorithmus immer besser als der naive?

Ist der Sweep-Line Algorithmus immer besser als der naive?

Nein, denn falls $I \in \Omega(n^2)$ hat der Algorithmus eine Laufzeit von $O(n^2 \log n)$.

Ist der Sweep-Line Algorithmus immer besser als der naive?

Nein, denn falls $I \in \Omega(n^2)$ hat der Algorithmus eine Laufzeit von $O(n^2 \log n)$.

Geht es noch besser?

Ist der Sweep-Line Algorithmus immer besser als der naive?

Nein, denn falls $I \in \Omega(n^2)$ hat der Algorithmus eine Laufzeit von $O(n^2 \log n)$.

Geht es noch besser?

Ja, in $\Theta(n \log n + I)$ Zeit und $\Theta(n)$ Platz [Balaban, 1995].

Ist der Sweep-Line Algorithmus immer besser als der naive?

Nein, denn falls $I \in \Omega(n^2)$ hat der Algorithmus eine Laufzeit von $O(n^2 \log n)$.

Geht es noch besser?

Ja, in $\Theta(n \log n + I)$ Zeit und $\Theta(n)$ Platz [Balaban, 1995].

Wie löst man damit das Kartenüberlagerungs-Problem?

Ist der Sweep-Line Algorithmus immer besser als der naive?

Nein, denn falls $I \in \Omega(n^2)$ hat der Algorithmus eine Laufzeit von $O(n^2 \log n)$.

Geht es noch besser?

Ja, in $\Theta(n \log n + I)$ Zeit und $\Theta(n)$ Platz [Balaban, 1995].

Wie löst man damit das Kartenüberlagerungs-Problem?

Unter Verwendung einer geeigneten Datenstruktur (**doppelt-verkettete Kantenliste**) für planare Graphen lässt sich in ebenfalls $O((n + I) \log n)$ Zeit die überlagerte Unterteilung der Karte berechnen.
(Details s. Kap. 2.3 im Buch)