

Computational Geometry · Lecture

Line Segment Intersection

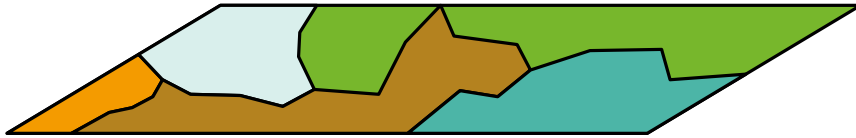
INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Tamara Mchedlidze
25.04.2018



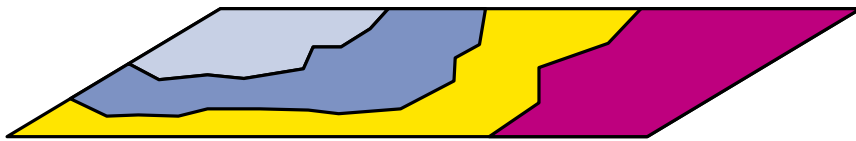
Overlaying Map Layers

Example: Given two different map layers whose intersection is of interest.



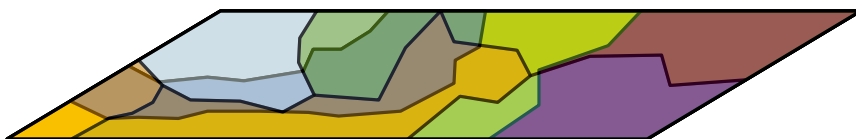
Land use

+



Precipitation

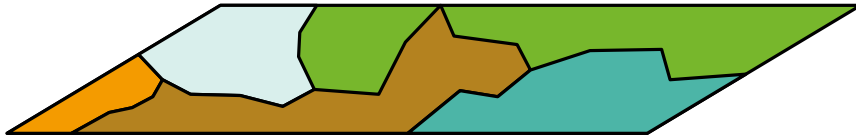
=



Map combining themes

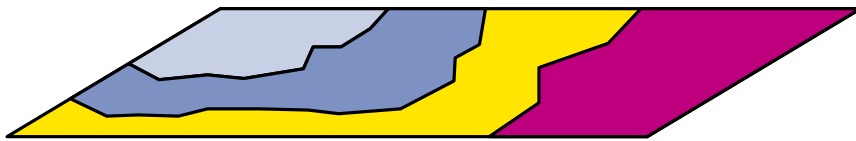
Overlaying Map Layers

Example: Given two different map layers whose intersection is of interest.



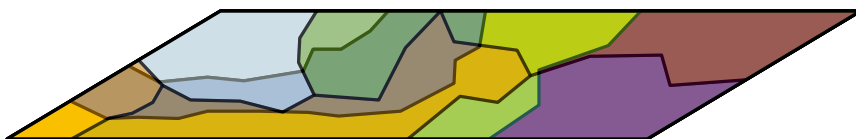
Land use

+



Precipitation

=



Map combining themes

- Regions are polygons
- Polygons are line segments
- **Calculate all line segment intersections**
- Compute regions

Problem Formulation

Given: Set $S = \{s_1, \dots, s_n\}$ of line segments in the plane

Output:

- all intersections of two or more line segments
- for each intersection, the line segments involved.

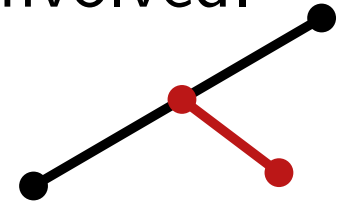
Problem Formulation

Given: Set $S = \{s_1, \dots, s_n\}$ of line segments in the plane

Output:

- all intersections of two or more line segments
- for each intersection, the line segments involved.

Def: Line segments are **closed**



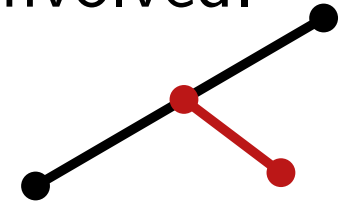
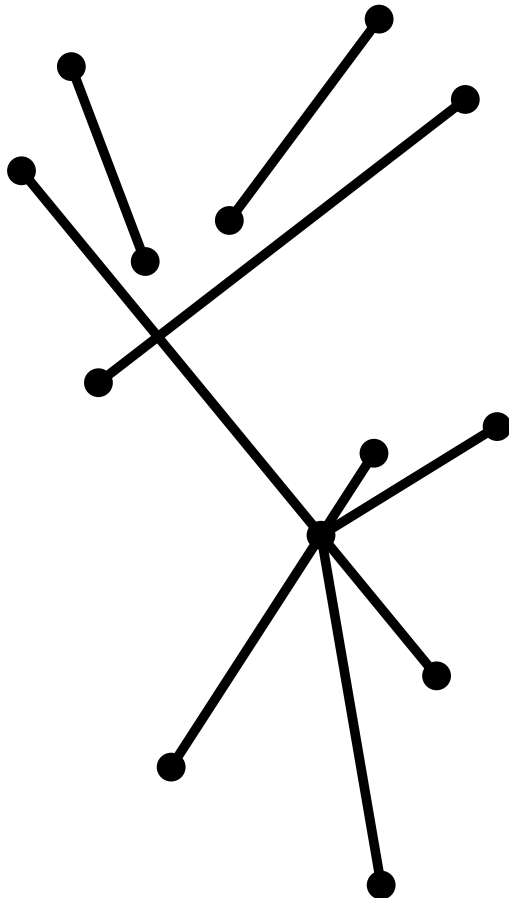
Problem Formulation

Given: Set $S = \{s_1, \dots, s_n\}$ of line segments in the plane

Output:

- all intersections of two or more line segments
- for each intersection, the line segments involved.

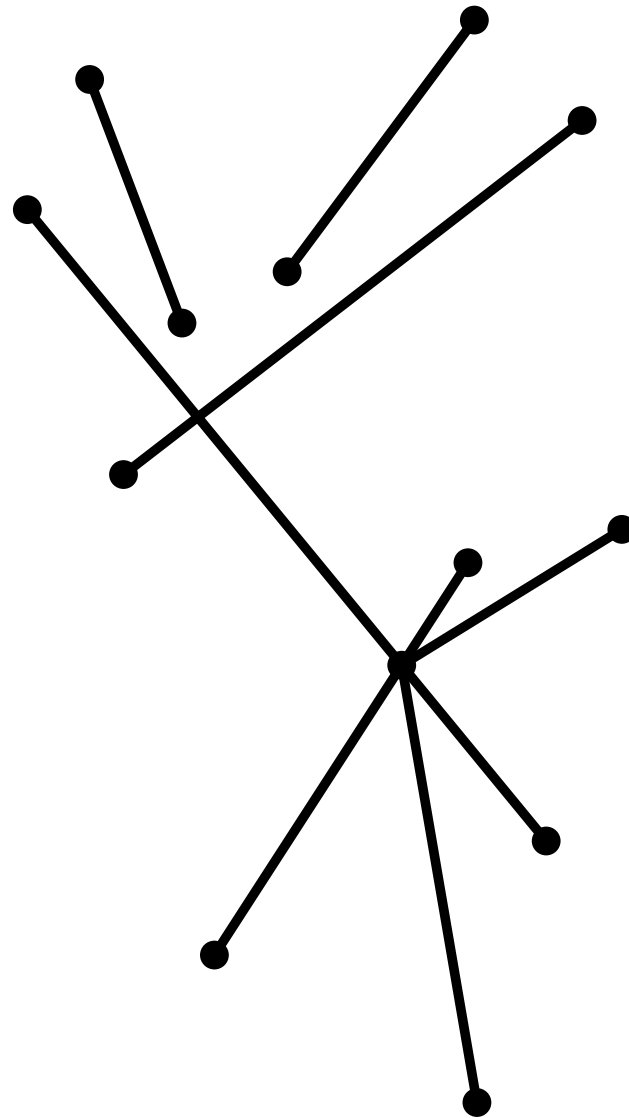
Def: Line segments are **closed**



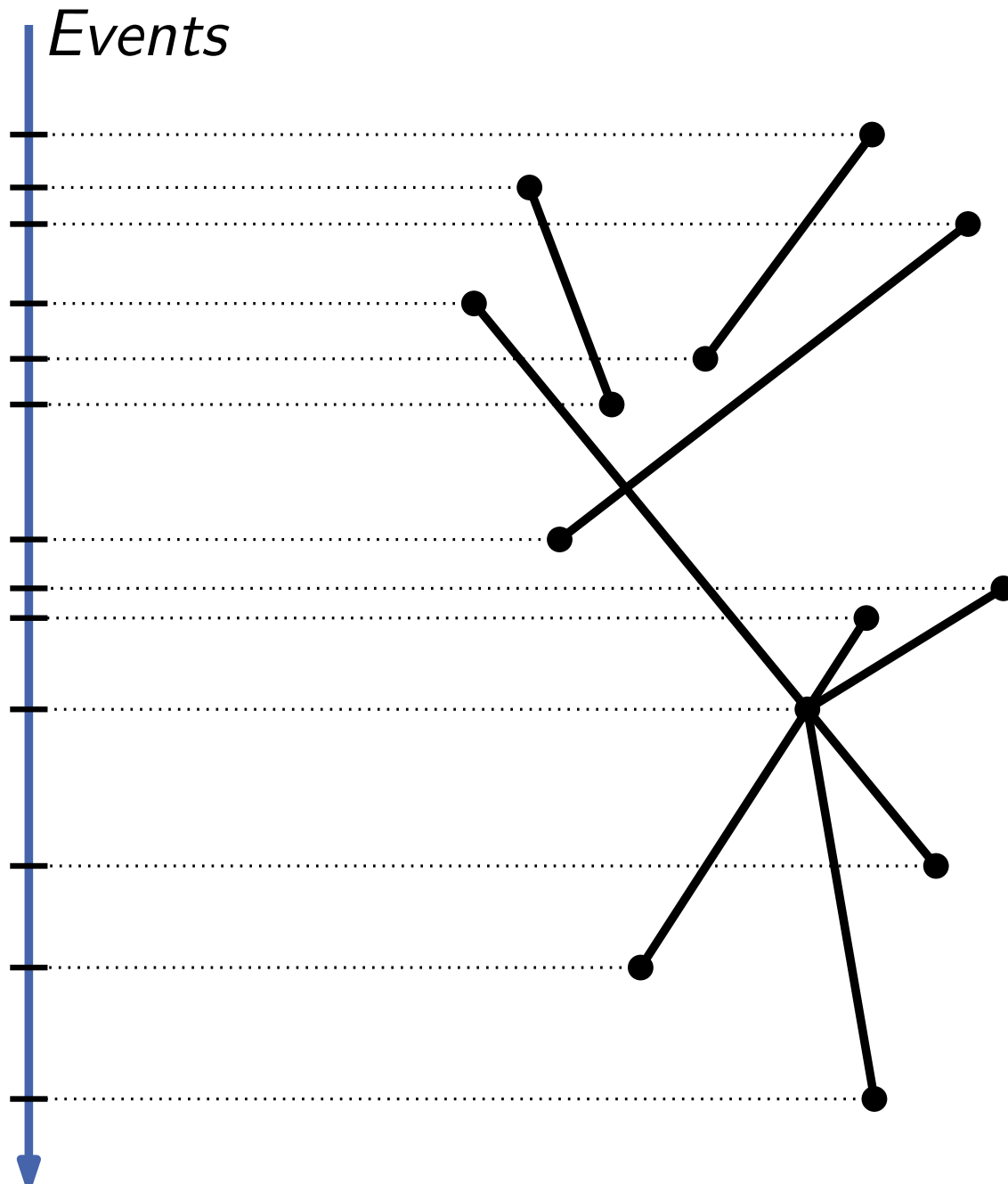
Discussion:

- How can you solve this problem naively?
- Is this already optimal?
- Are there better approaches?

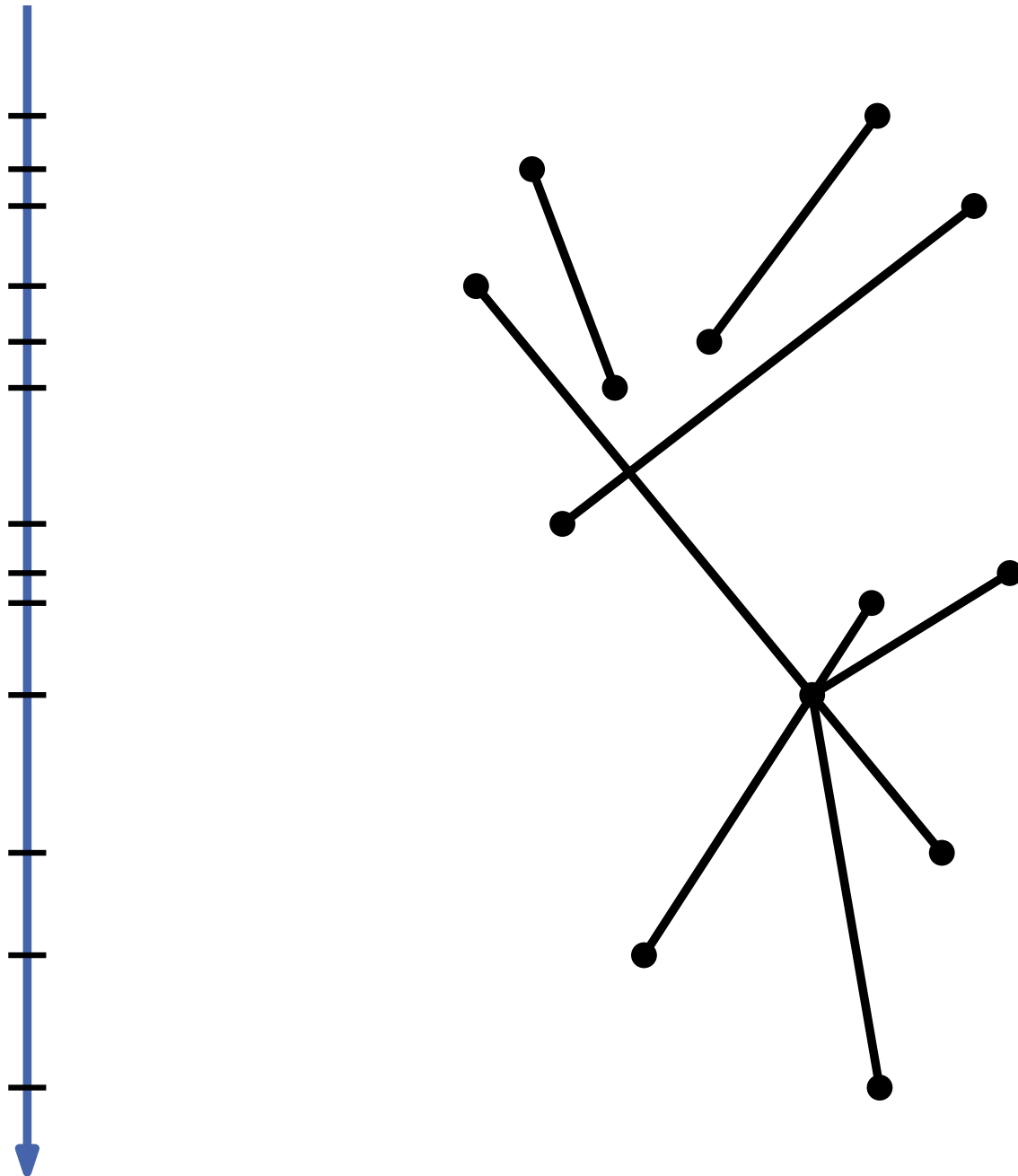
The Sweep-Line Method: An Example



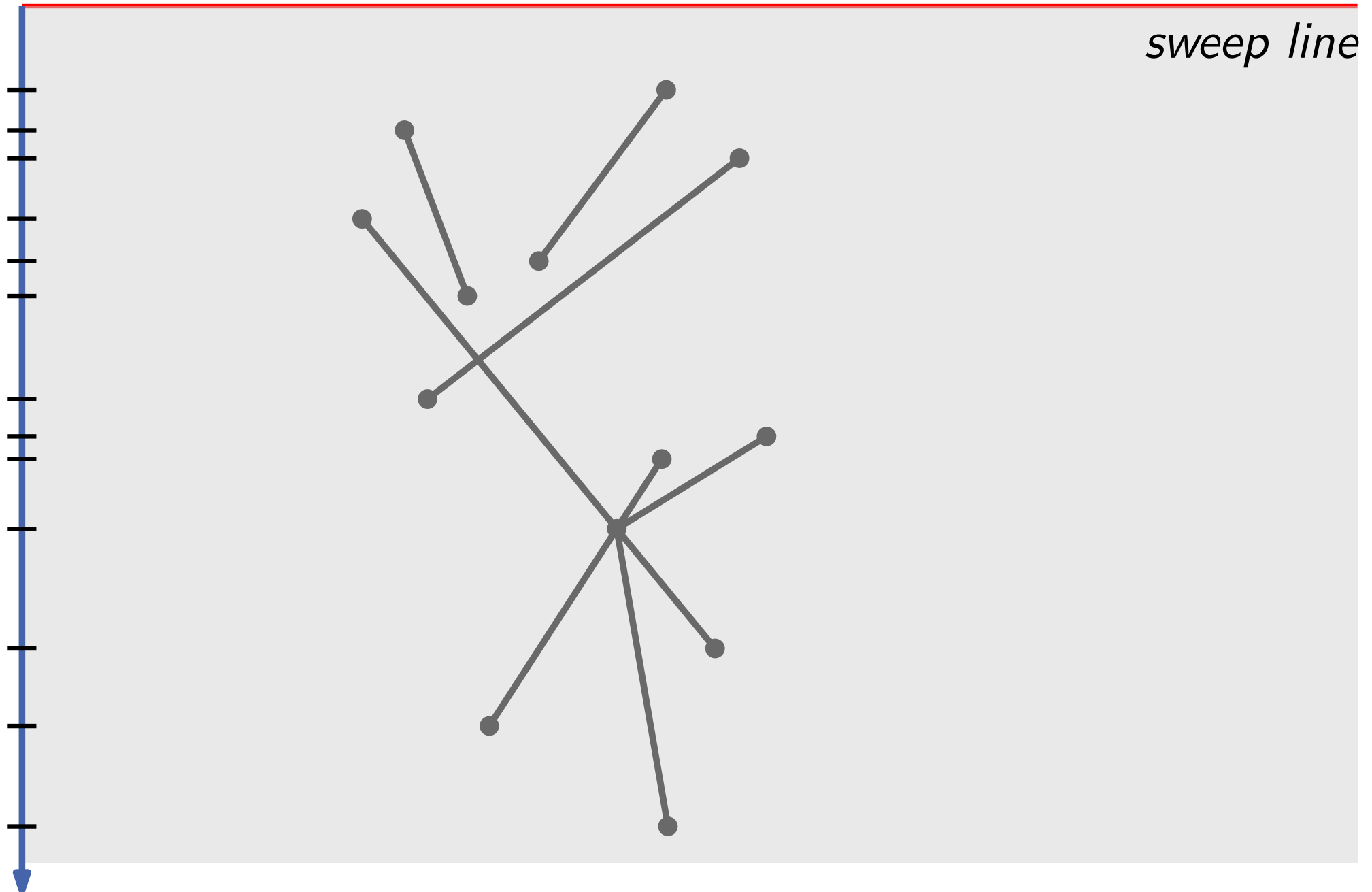
The Sweep-Line Method: An Example



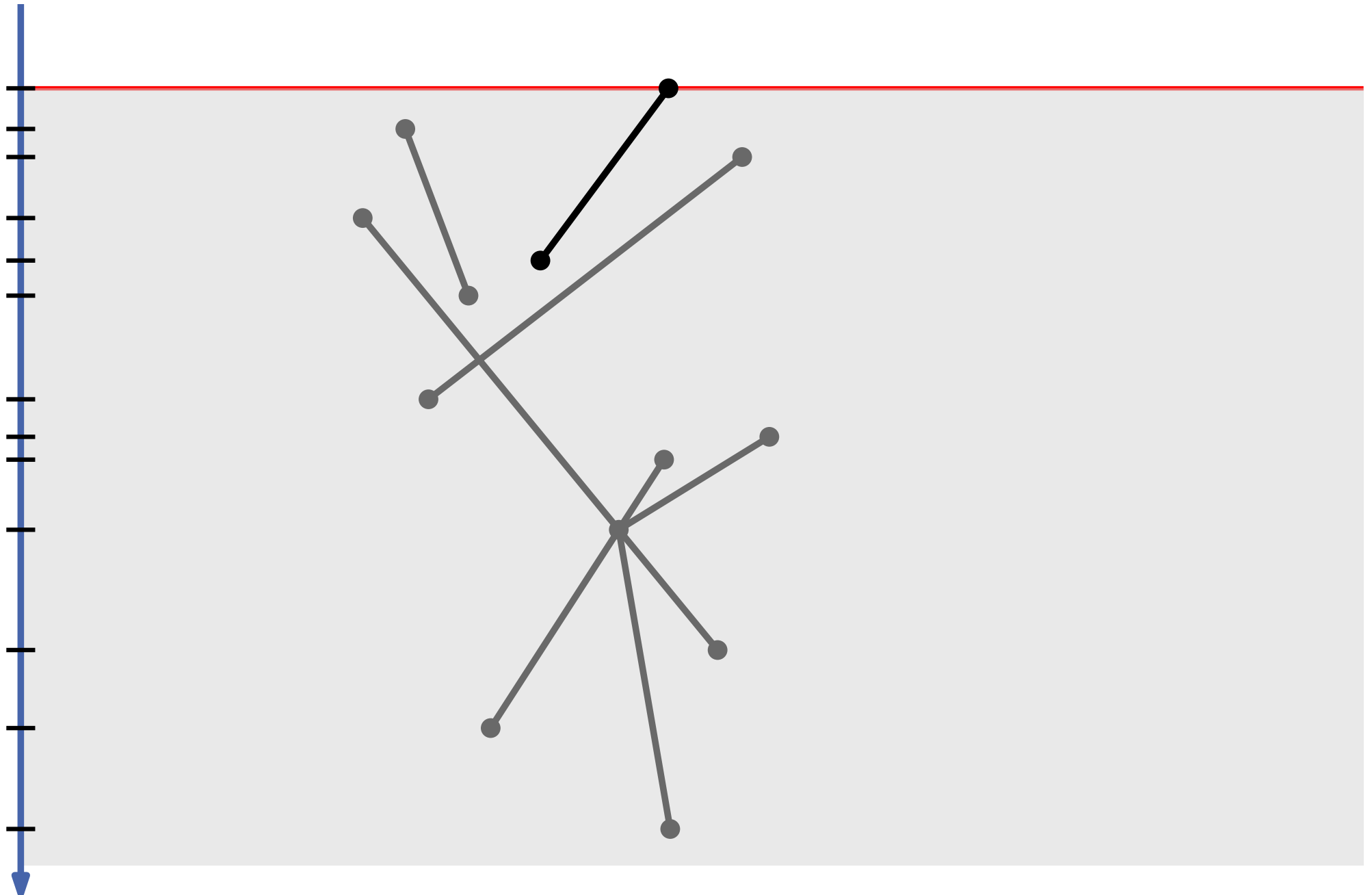
The Sweep-Line Method: An Example



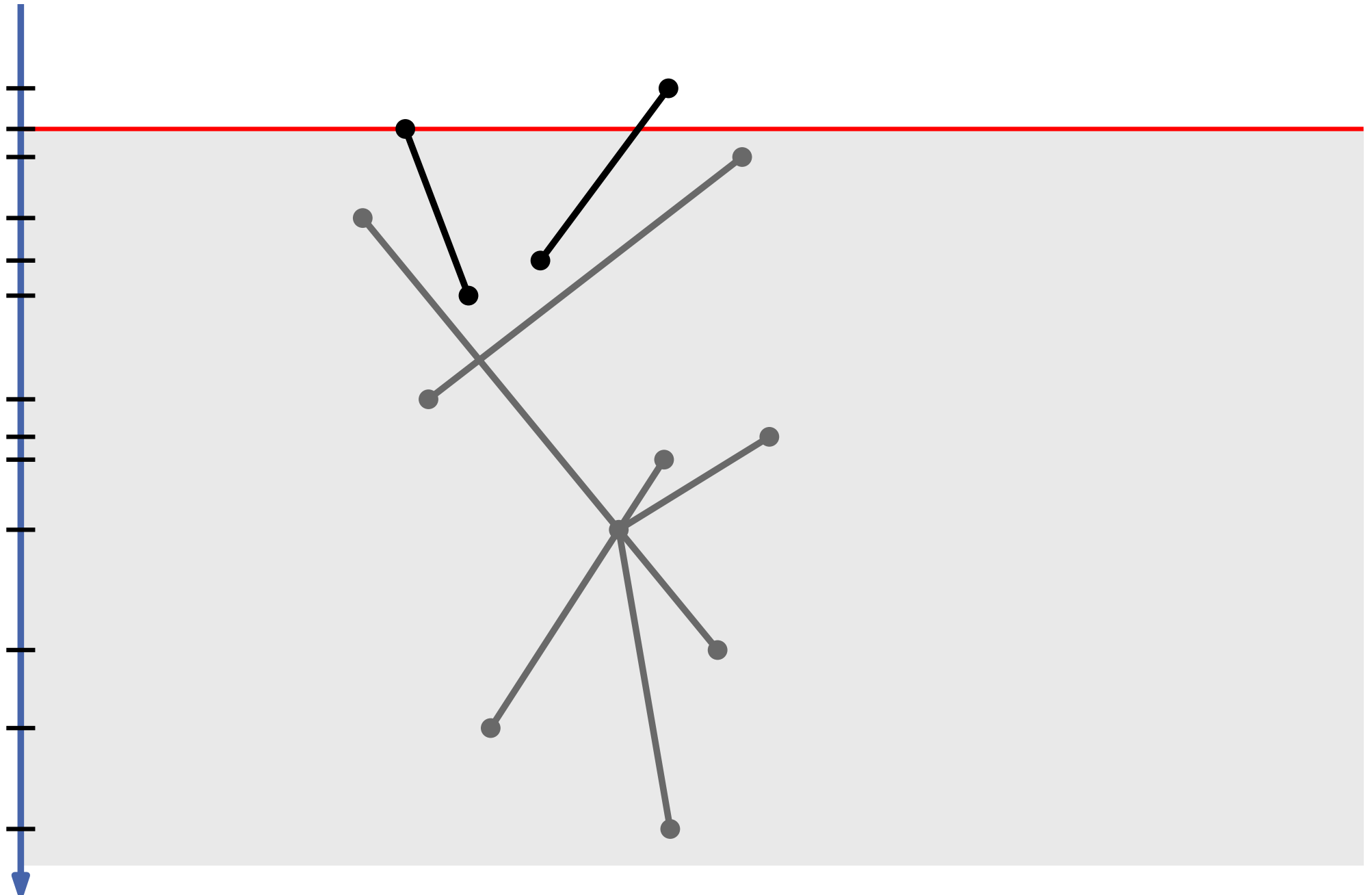
The Sweep-Line Method: An Example



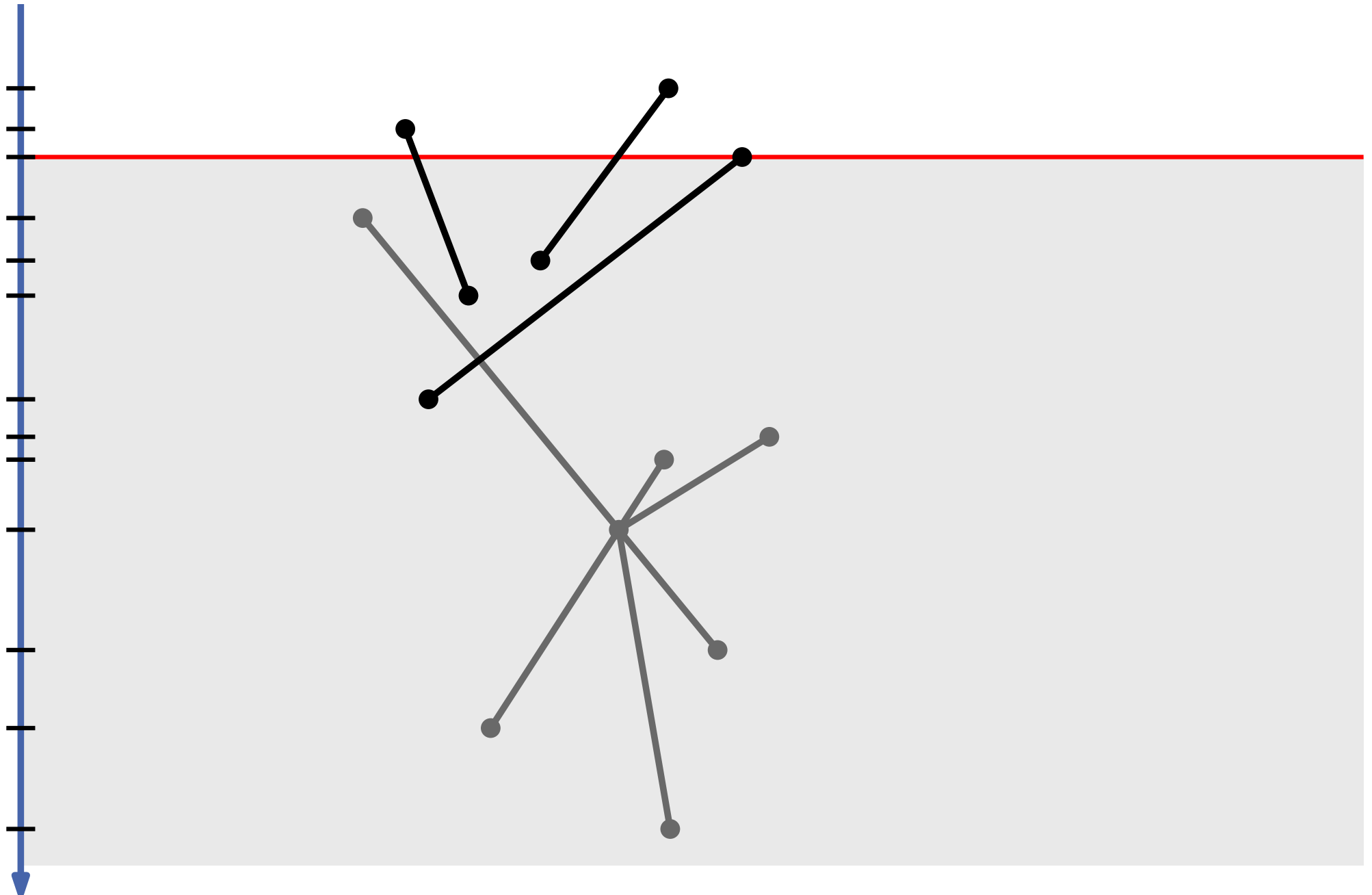
The Sweep-Line Method: An Example



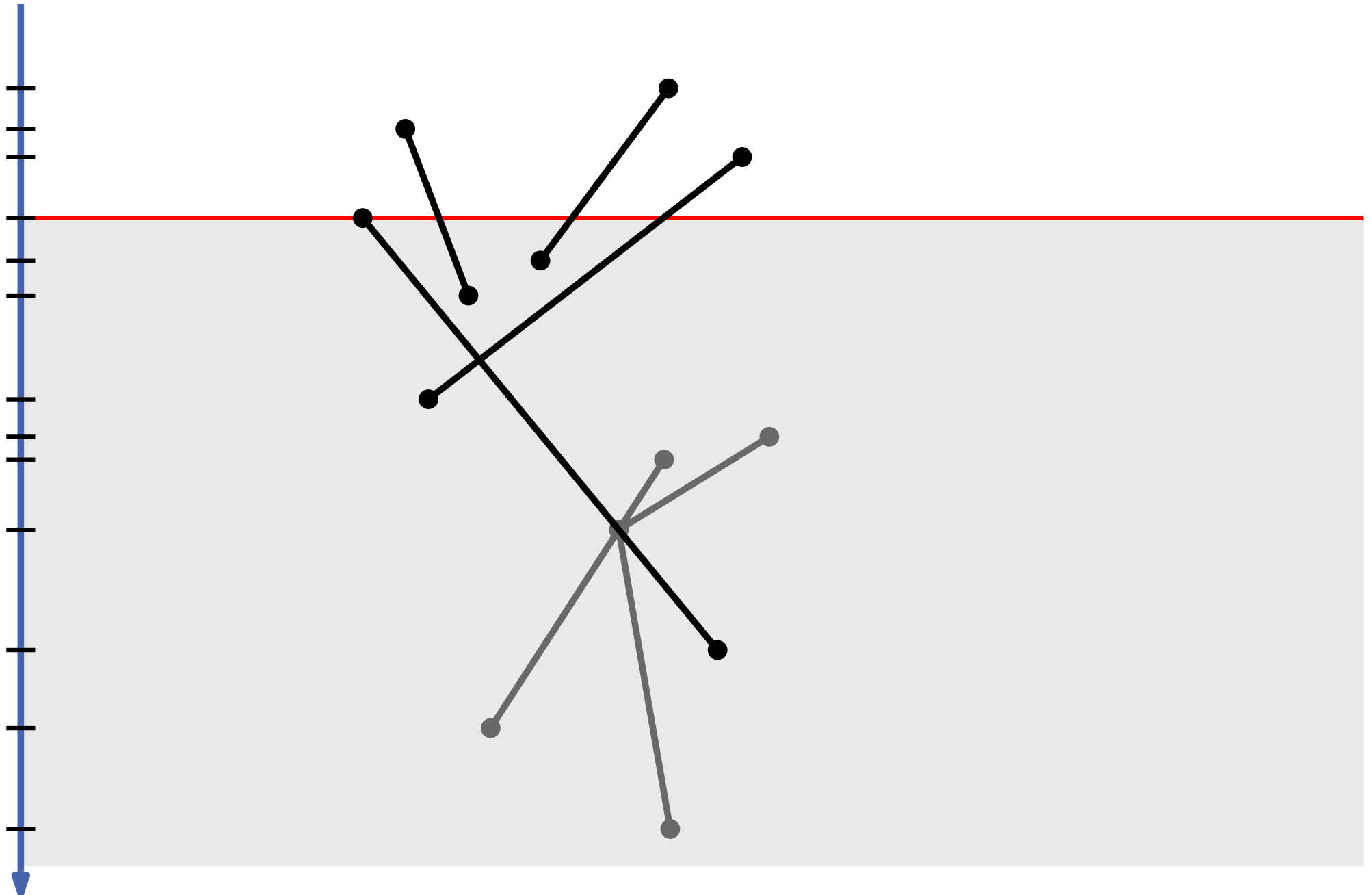
The Sweep-Line Method: An Example



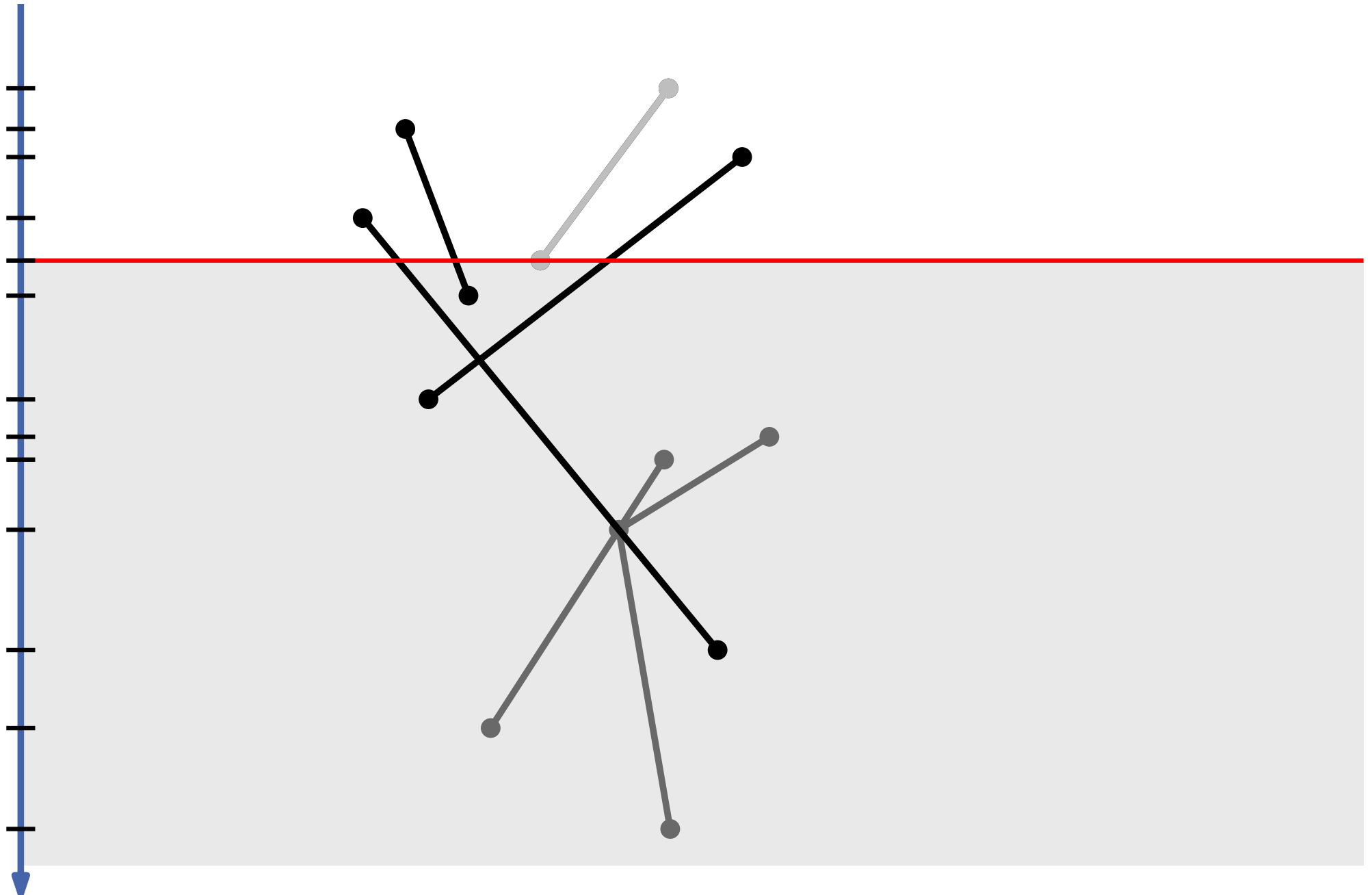
The Sweep-Line Method: An Example



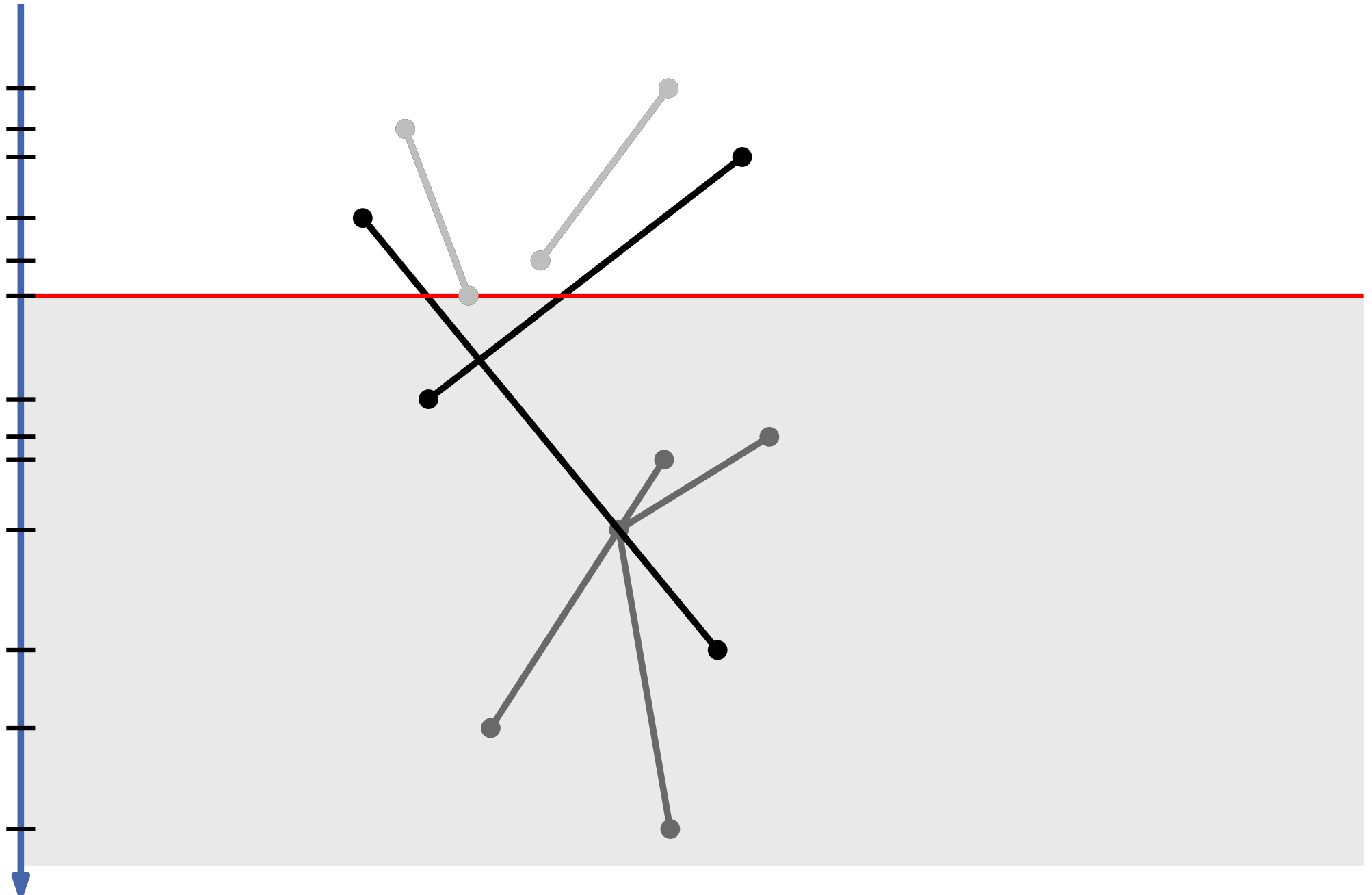
The Sweep-Line Method: An Example



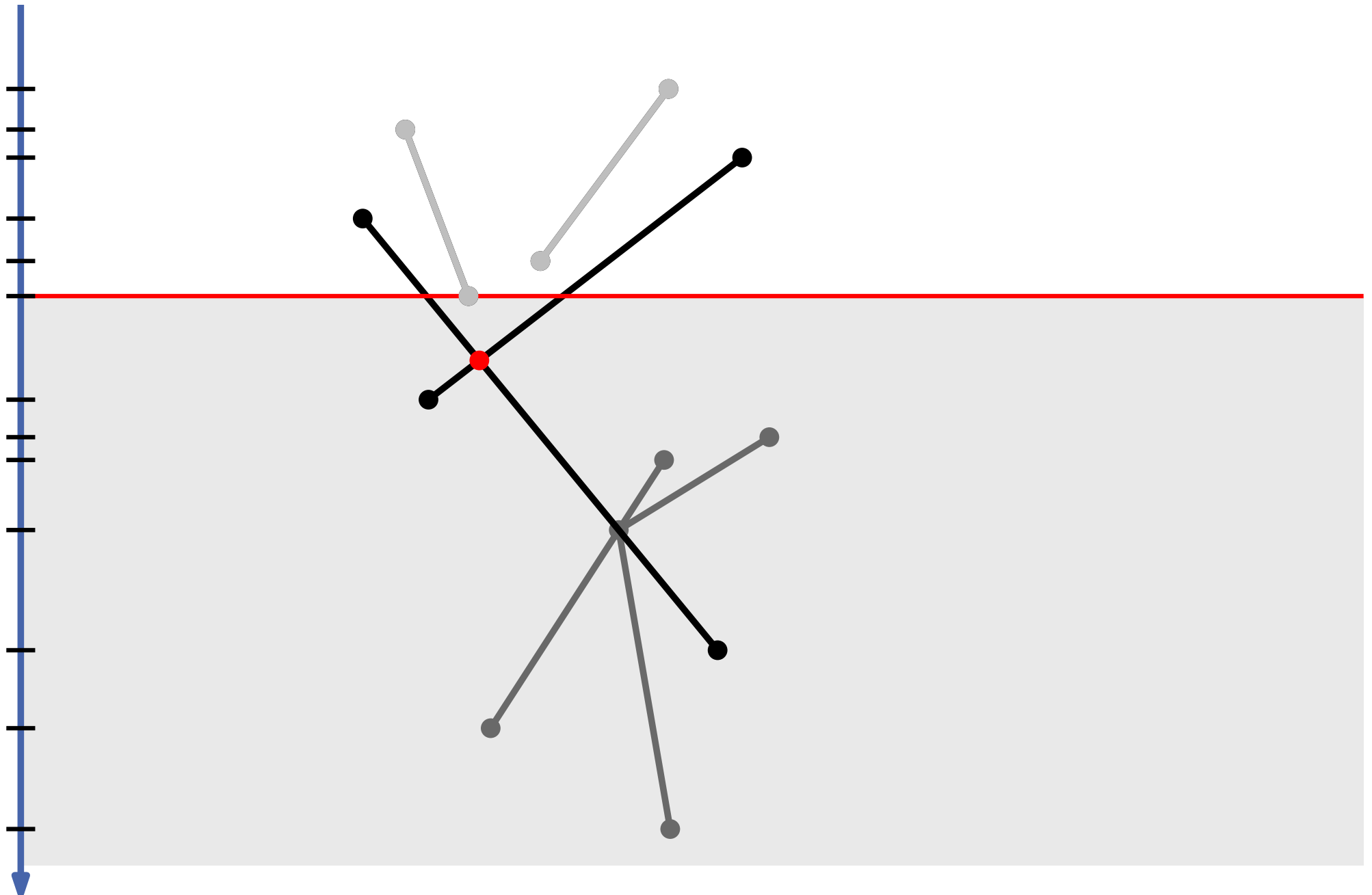
The Sweep-Line Method: An Example



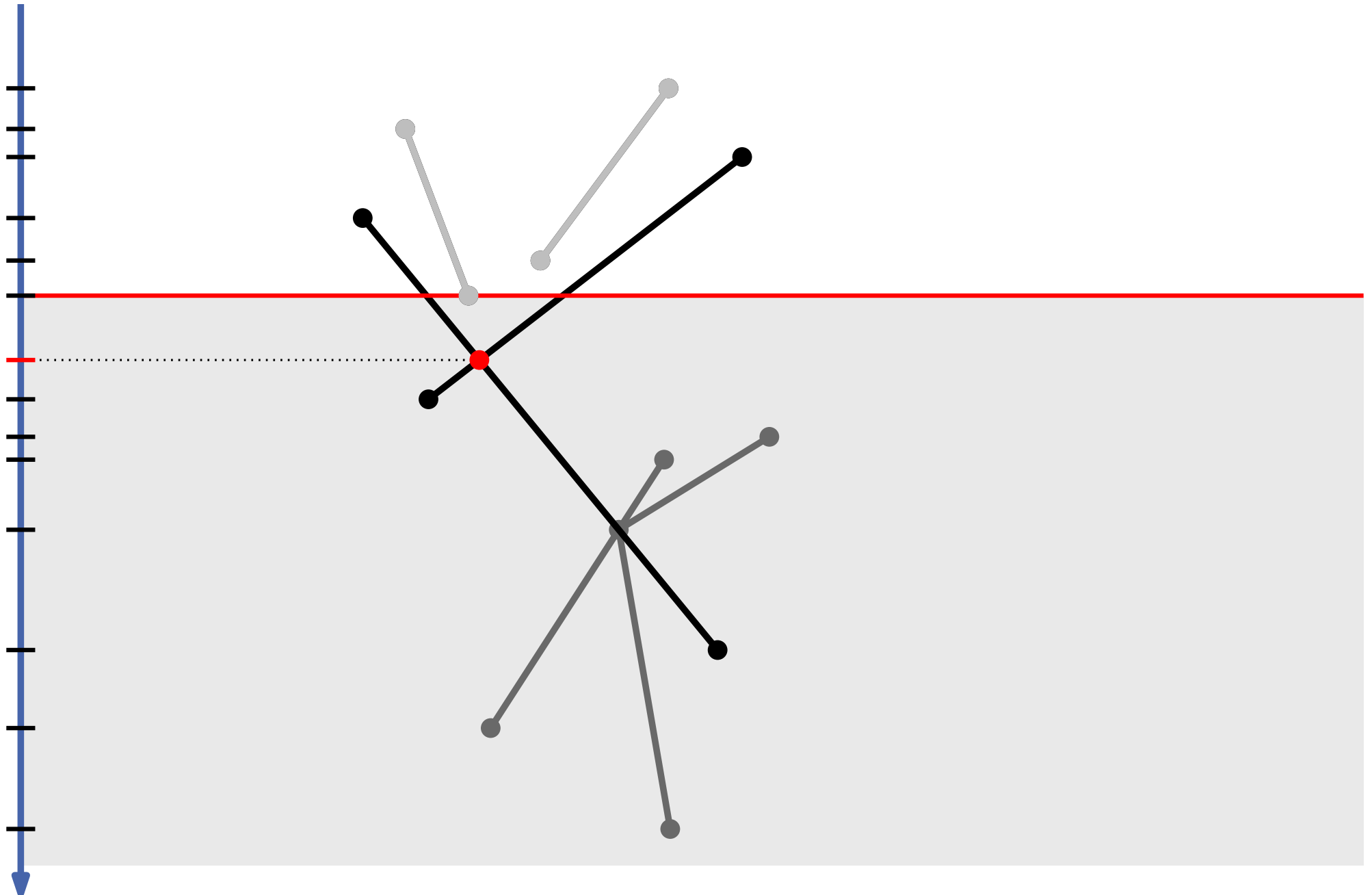
The Sweep-Line Method: An Example



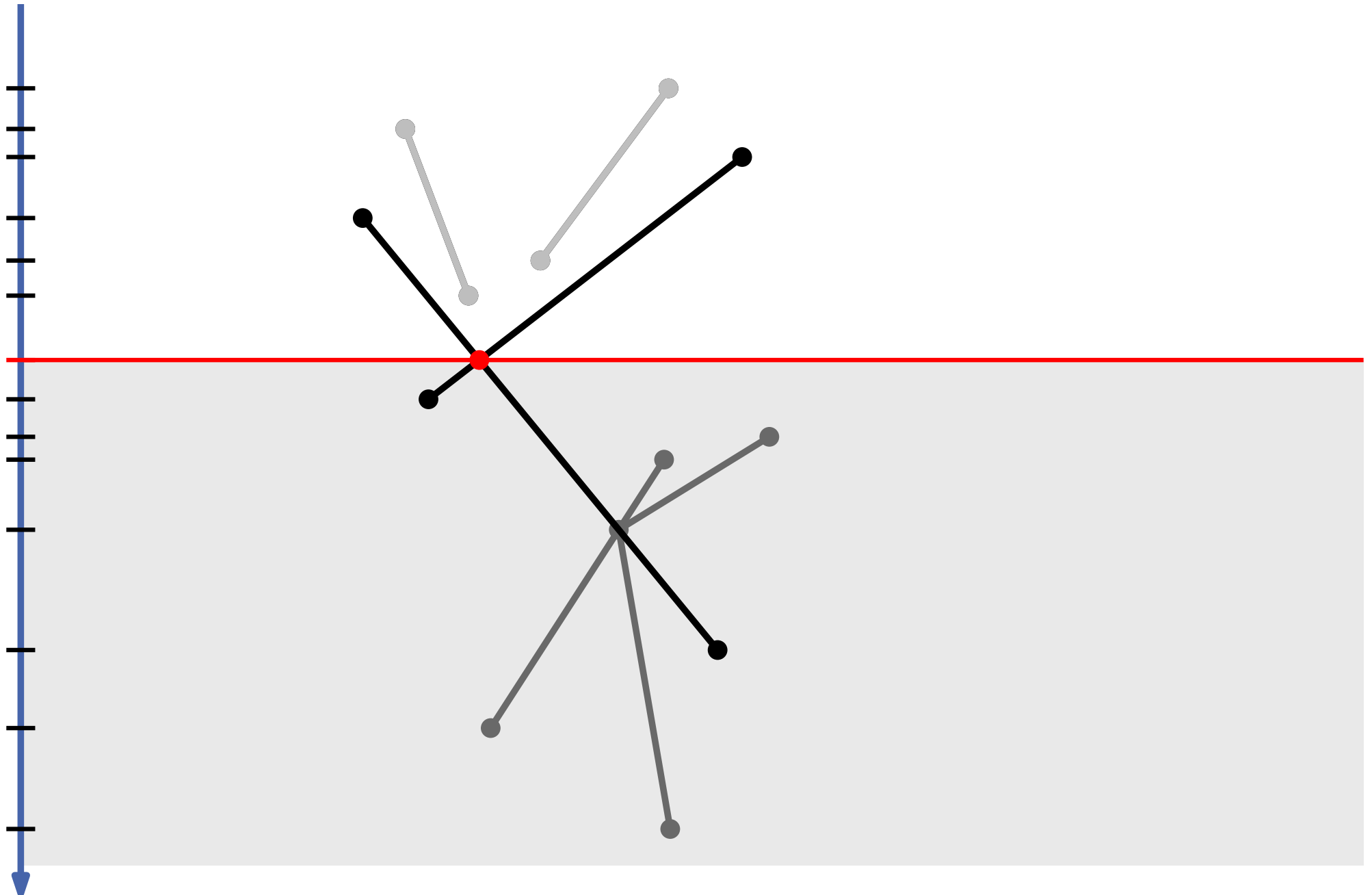
The Sweep-Line Method: An Example



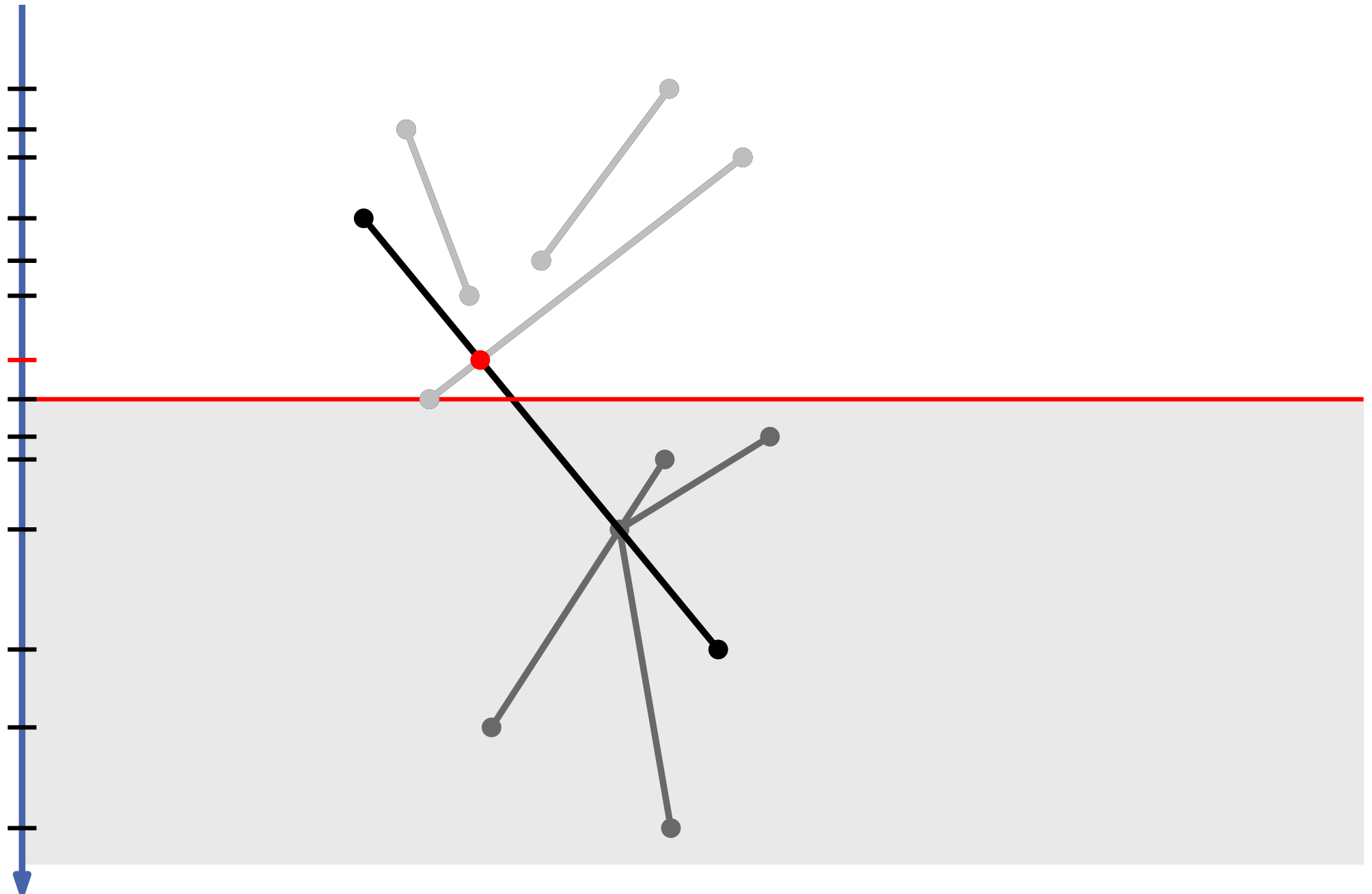
The Sweep-Line Method: An Example



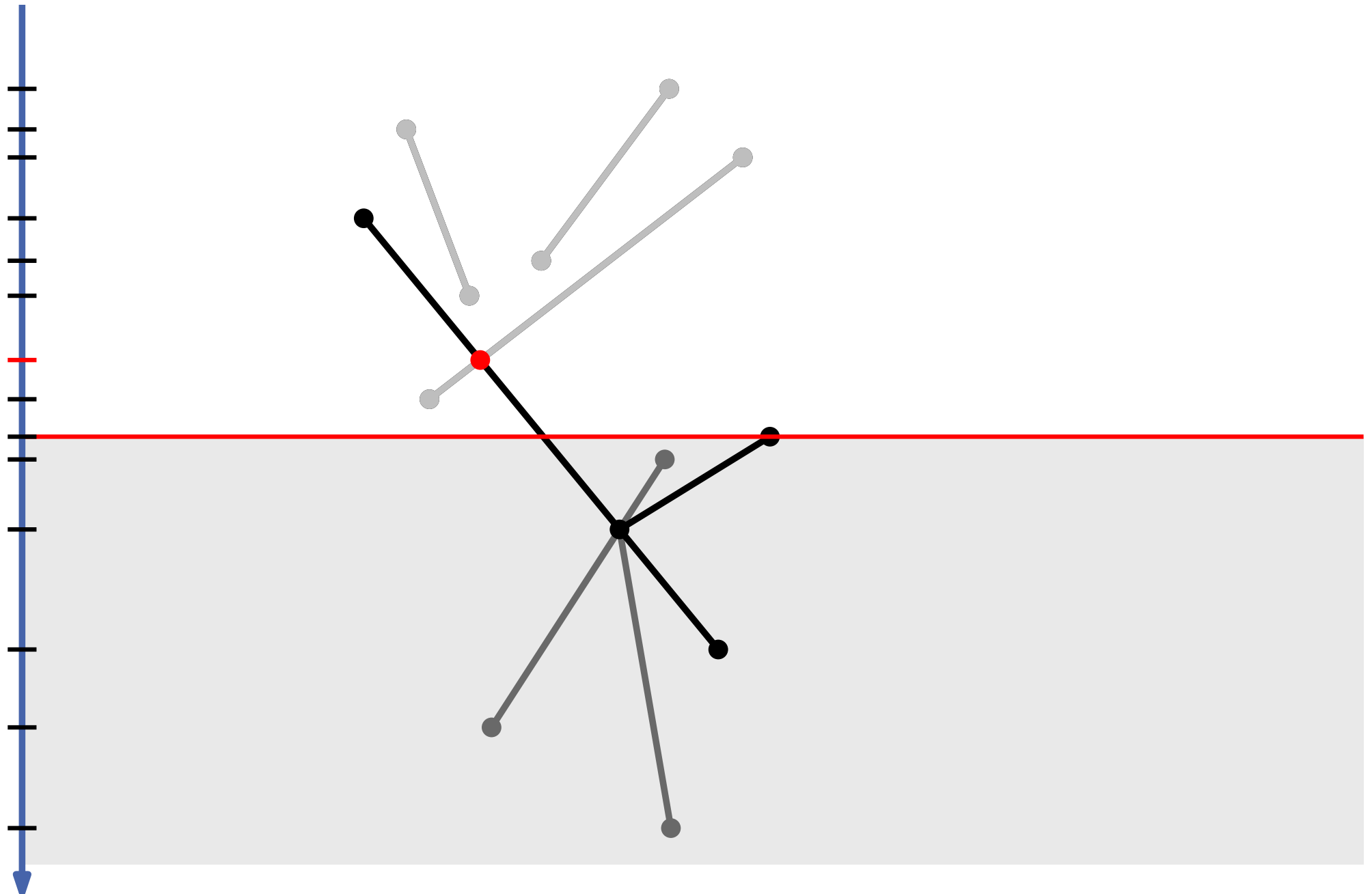
The Sweep-Line Method: An Example



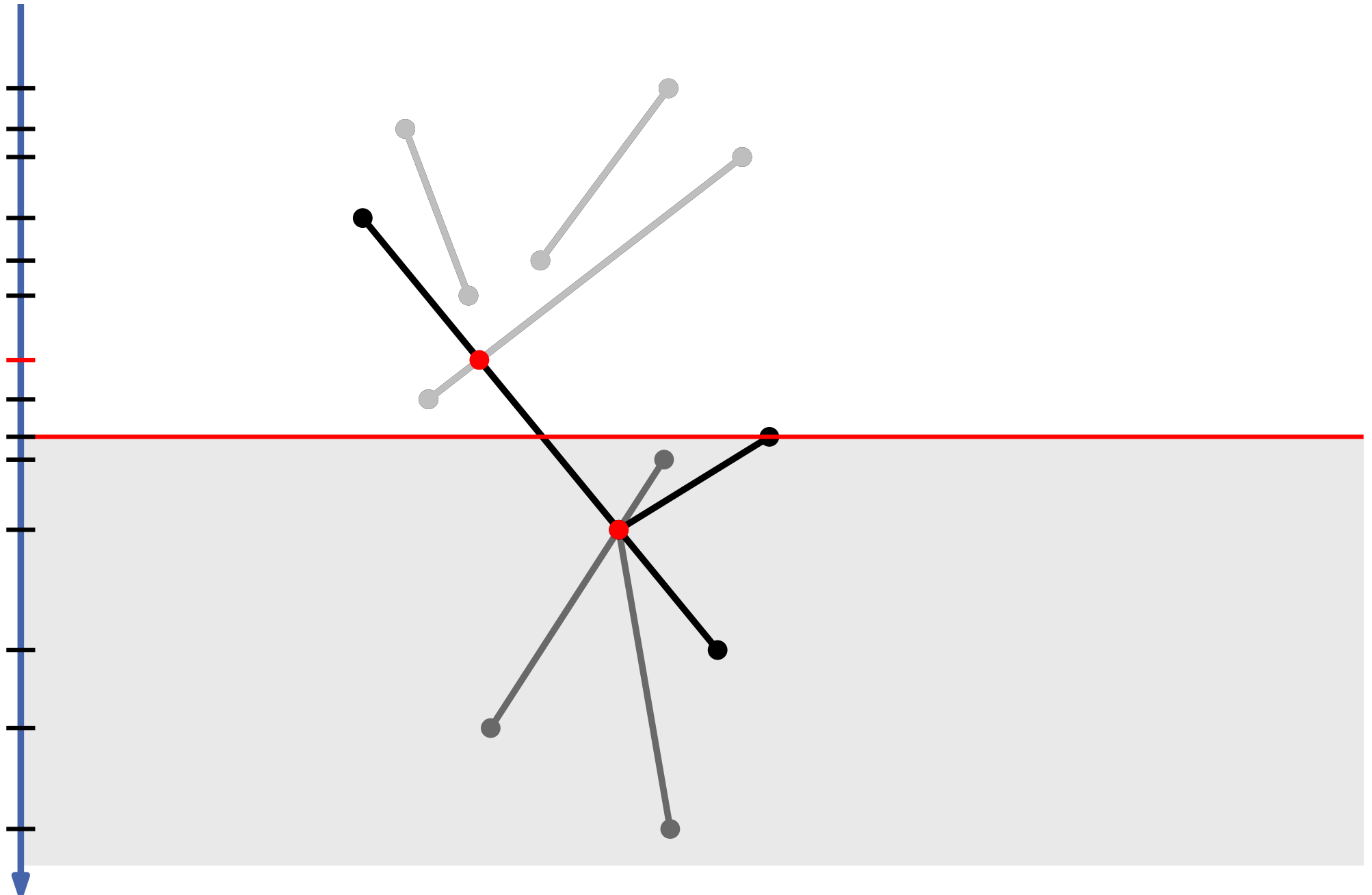
The Sweep-Line Method: An Example



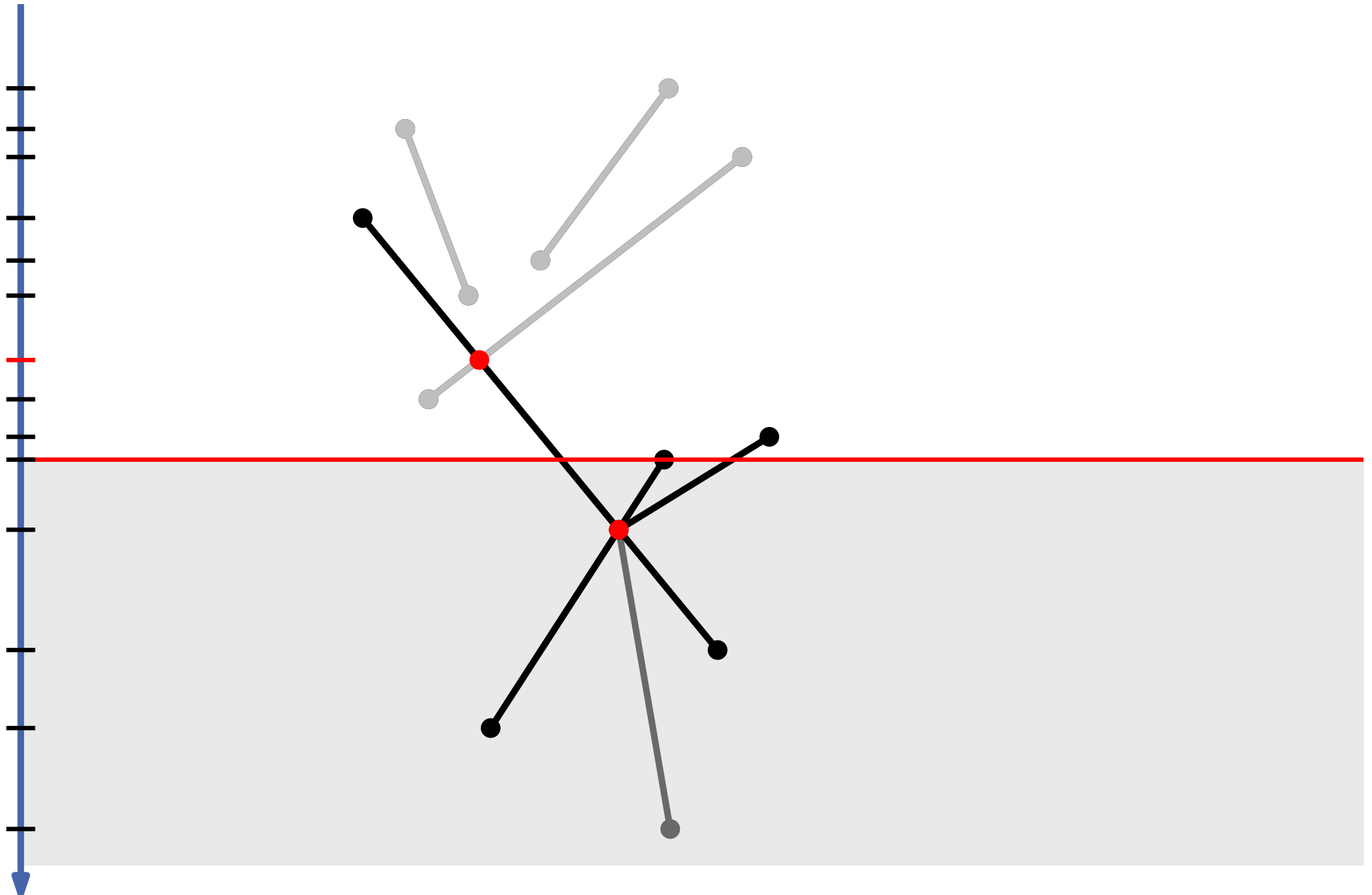
The Sweep-Line Method: An Example



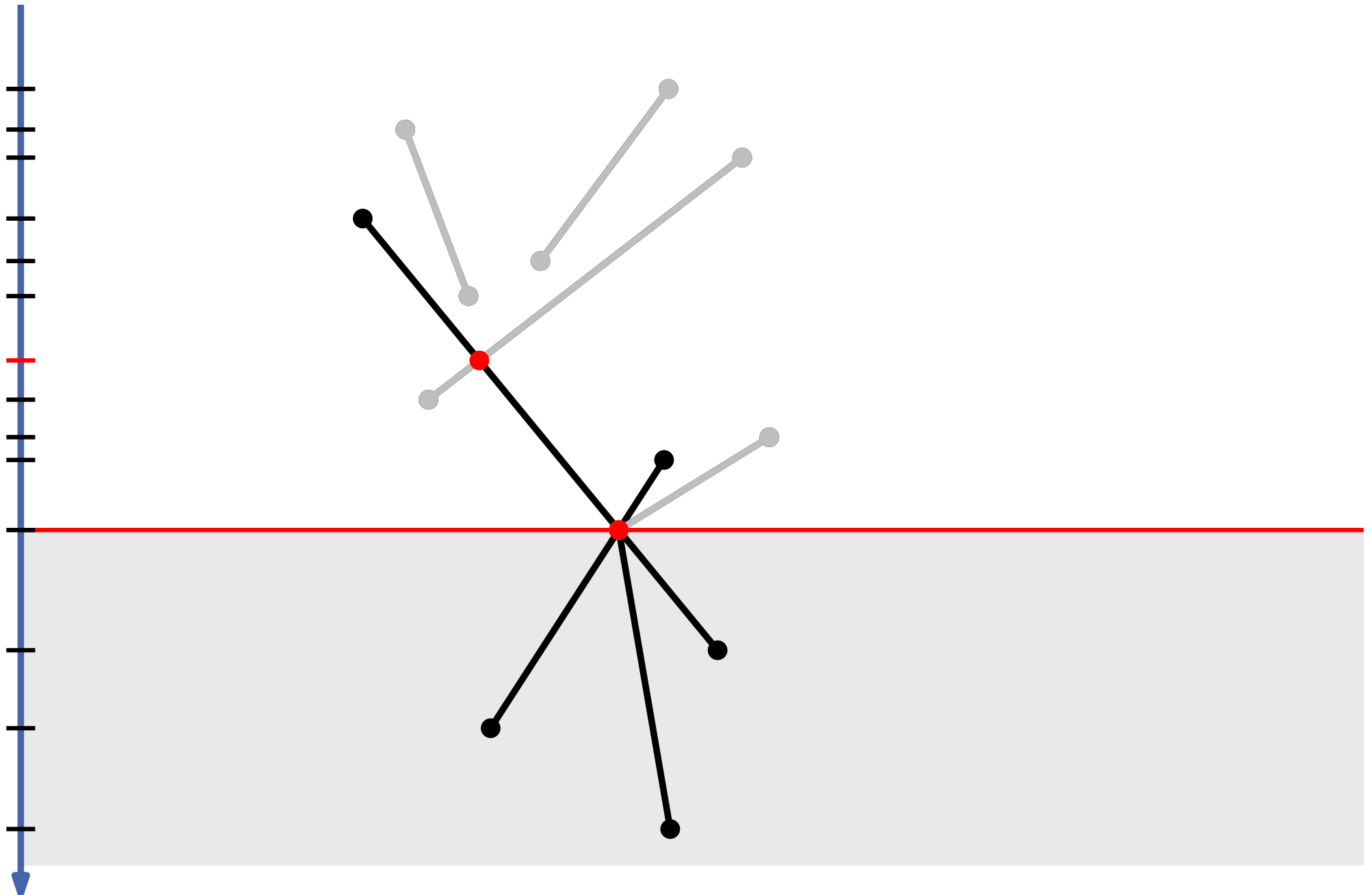
The Sweep-Line Method: An Example



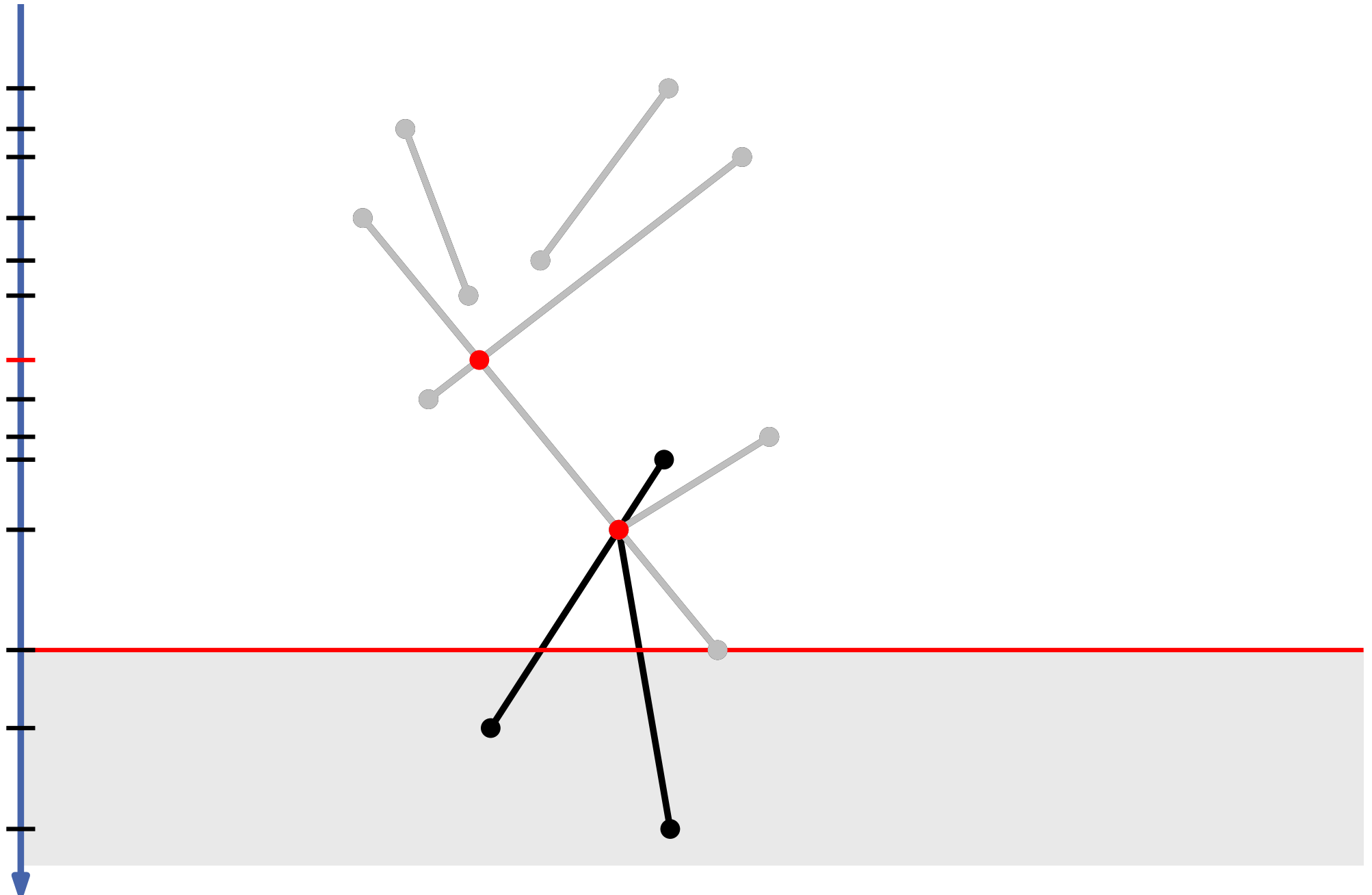
The Sweep-Line Method: An Example



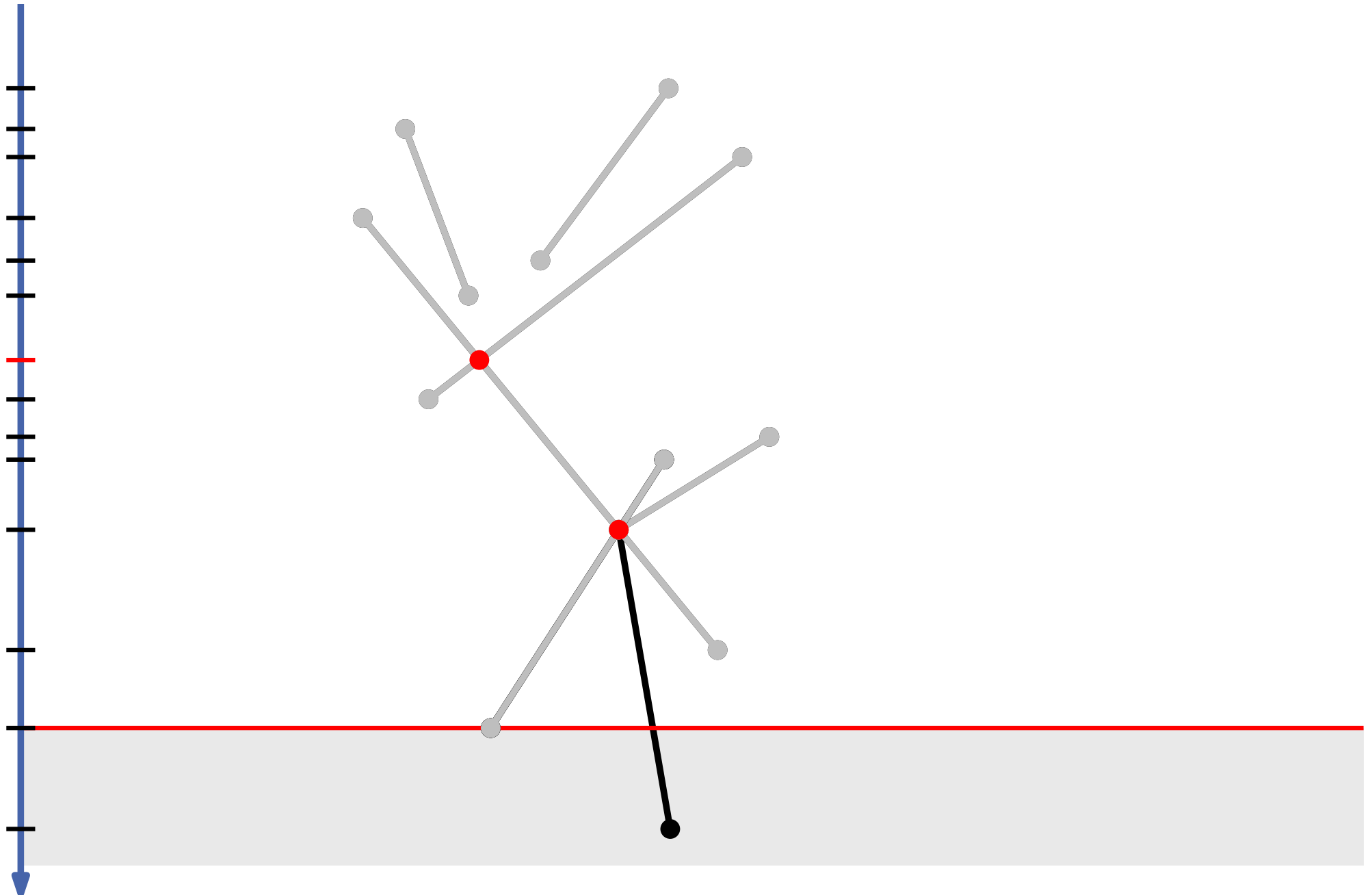
The Sweep-Line Method: An Example



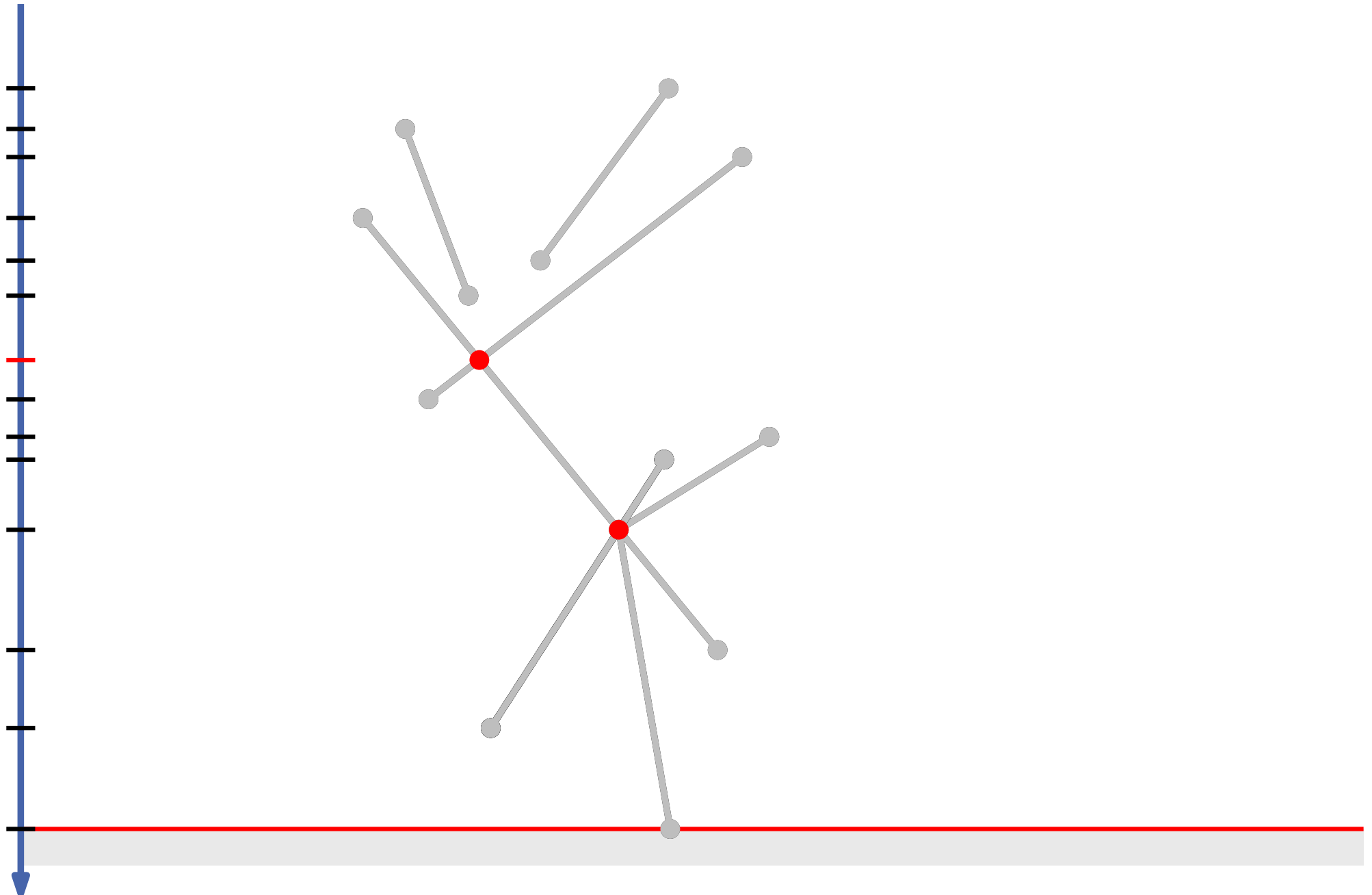
The Sweep-Line Method: An Example



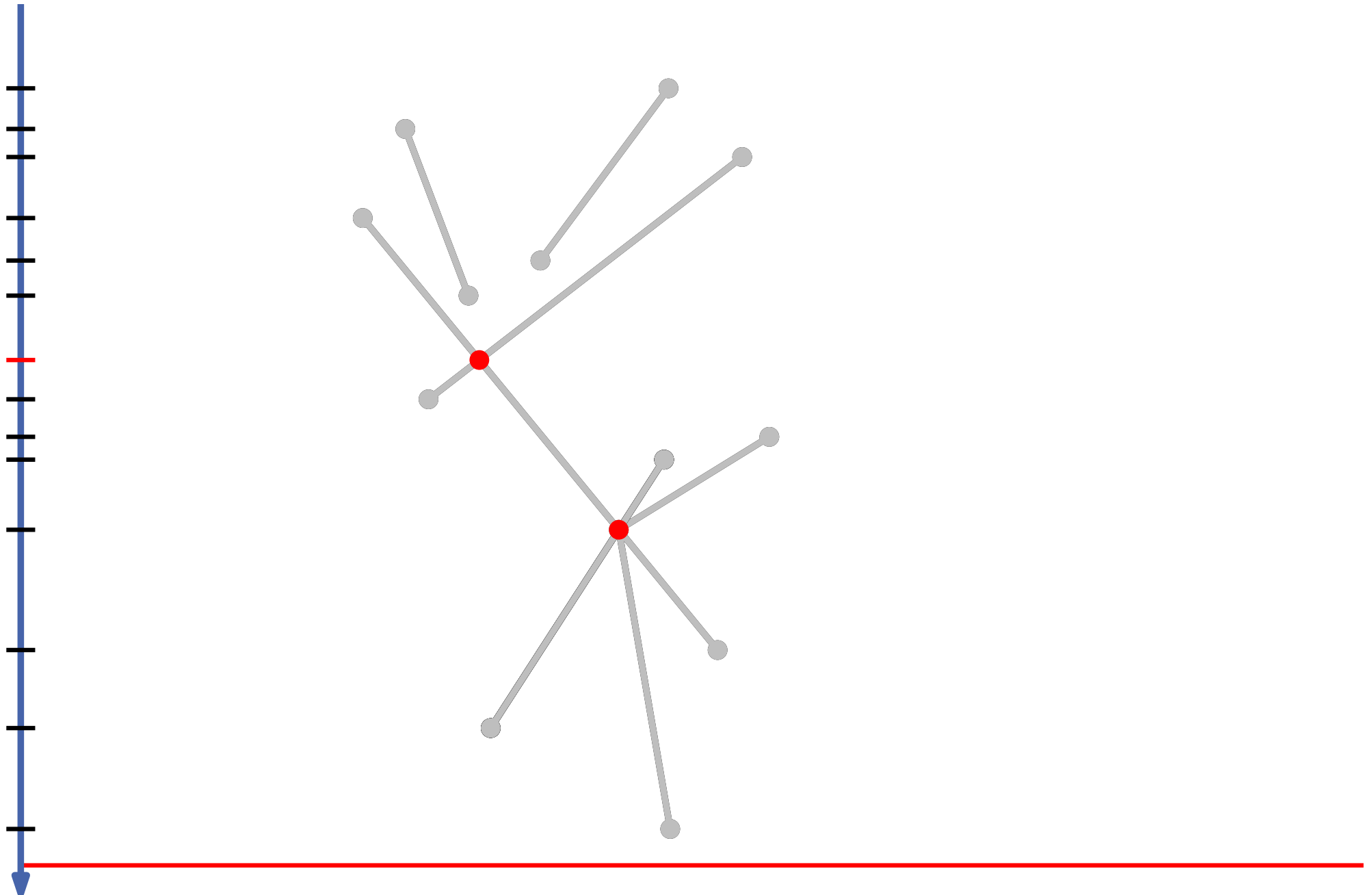
The Sweep-Line Method: An Example



The Sweep-Line Method: An Example

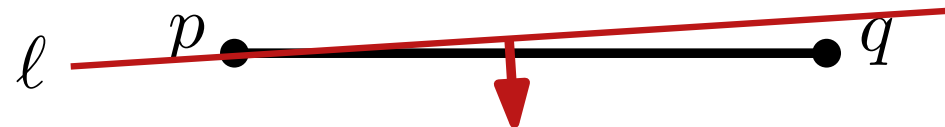


The Sweep-Line Method: An Example



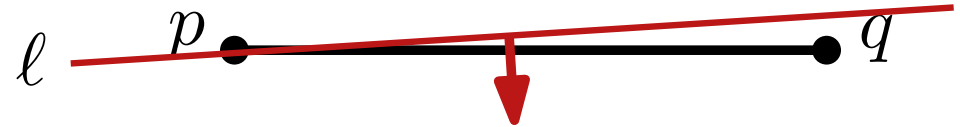
1.) Event Queue Q

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



1.) Event Queue \mathcal{Q}

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

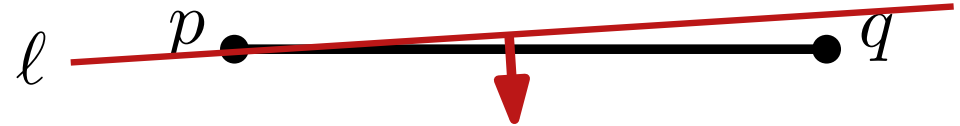


- Store events by \prec in a **balanced binary search tree**

→ e.g., AVL tree

1.) Event Queue Q

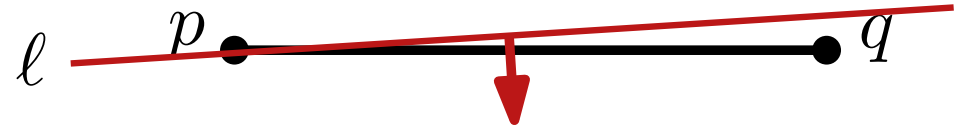
- define $p \prec q \iff_{\text{def.}} y_p > y_q \vee (y_p = y_q \wedge x_p < x_q)$



- Store events by \prec in a **balanced binary search tree**
→ e.g., AVL tree
- Operations insert, delete and nextEvent in $O(\log |Q|)$ time

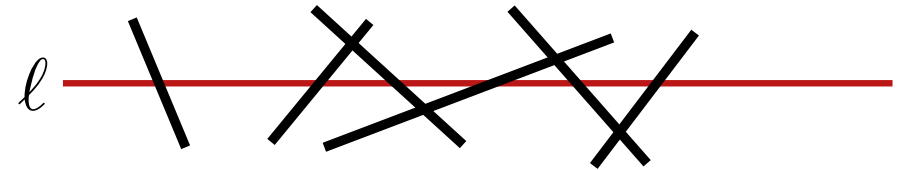
1.) Event Queue \mathcal{Q}

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



- Store events by \prec in a **balanced binary search tree**
→ e.g., AVL tree
- Operations insert, delete and nextEvent in $O(\log |\mathcal{Q}|)$ time

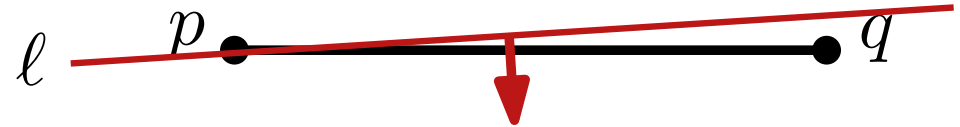
2.) Sweep-Line Status \mathcal{T}



- Stores ℓ cut lines ordered from left to right

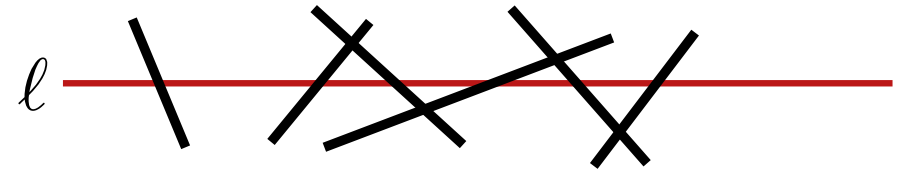
1.) Event Queue \mathcal{Q}

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



- Store events by \prec in a **balanced binary search tree**
→ e.g., AVL tree
- Operations insert, delete and nextEvent in $O(\log |\mathcal{Q}|)$ time

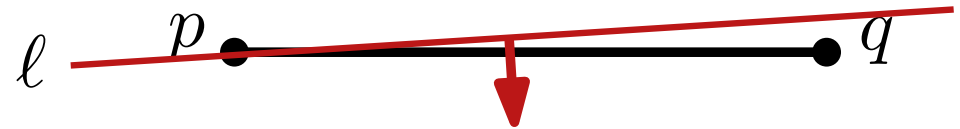
2.) Sweep-Line Status \mathcal{T}



- Stores ℓ cut lines ordered from left to right
- Required operations insert, delete, findNeighbor

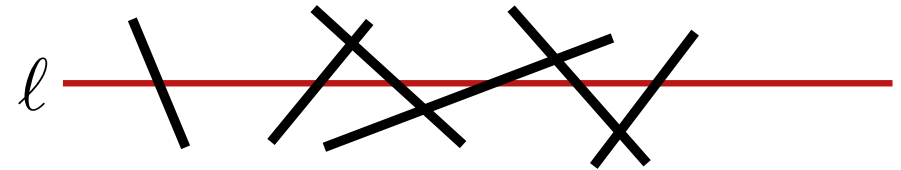
1.) Event Queue \mathcal{Q}

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



- Store events by \prec in a **balanced binary search tree**
→ e.g., AVL tree
- Operations insert, delete and nextEvent in $O(\log |\mathcal{Q}|)$ time

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



- Stores ℓ cut lines ordered from left to right
- Required operations insert, delete, findNeighbor
- This is also a balanced binary search tree with line segments stored in the leaves!

Algorithm

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersection points and the line segments involved

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

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)

Algorithm

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersection points and the line segments involved

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

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

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

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

Store the segment together with its upper end point.

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

$p \leftarrow Q.nextEvent()$

$Q.deleteEvent(p)$

$handleEvent(p)$

Algorithm

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersection points and the line segments involved

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

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

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

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

What happens with duplicates?

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

$p \leftarrow Q.nextEvent()$

$Q.deleteEvent(p)$

$handleEvent(p)$

Algorithm

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersection points and the line segments involved

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

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

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

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

What happens with duplicates?

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

$p \leftarrow Q.nextEvent()$

$Q.deleteEvent(p)$

$handleEvent(p)$

This is the core of the algorithm!

Algorithm

handleEvent(p)

$U(p) \leftarrow$ Line segments with p as upper endpoint

$L(p) \leftarrow$ Line segments with p as lower endpoint

$C(p) \leftarrow$ Line segments with p as interior point

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

└ report p and $U(p) \cup L(p) \cup C(p)$

remove $L(p) \cup C(p)$ from \mathcal{T}

add $U(p) \cup C(p)$ to \mathcal{T}

if $U(p) \cup C(p) = \emptyset$ **then** // s_l and s_r , neighbors of p in \mathcal{T}

└ $Q \leftarrow$ check if s_l and s_r intersect below p

else // s' and s'' leftmost and rightmost line segment in $U(p) \cup C(p)$

└ $Q \leftarrow$ check if s_l and s' intersect below p

└ $Q \leftarrow$ check if s_r and s'' intersect below p

Algorithm

handleEvent(p)

$U(p) \leftarrow$ Line segments with p as upper endpoint

Stored with p in \mathcal{Q}

$L(p) \leftarrow$ Line segments with p as lower endpoint

$C(p) \leftarrow$ Line segments with p as interior point

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

┌ report p and $U(p) \cup L(p) \cup C(p)$

remove $L(p) \cup C(p)$ from \mathcal{T}

add $U(p) \cup C(p)$ to \mathcal{T}

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

// s_l and s_r , neighbors of p in \mathcal{T}

┌ $\mathcal{Q} \leftarrow$ check if s_l and s_r intersect below p

else // s' and s'' leftmost and rightmost line segment in $U(p) \cup C(p)$

┌ $\mathcal{Q} \leftarrow$ check if s_l and s' intersect below p

┌ $\mathcal{Q} \leftarrow$ check if s_r and s'' intersect below p

Algorithm

handleEvent(p)

$U(p) \leftarrow$ Line segments with p as upper endpoint

Stored with p in \mathcal{Q}

$L(p) \leftarrow$ Line segments with p as lower endpoint

$C(p) \leftarrow$ Line segments with p as interior point

Neighbors in \mathcal{T}

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

 report p and $U(p) \cup L(p) \cup C(p)$

remove $L(p) \cup C(p)$ from \mathcal{T}

add $U(p) \cup C(p)$ to \mathcal{T}

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

// s_l and s_r , neighbors of p in \mathcal{T}

$\mathcal{Q} \leftarrow$ check if s_l and s_r intersect below p

else // s' and s'' leftmost and rightmost line segment in $U(p) \cup C(p)$

$\mathcal{Q} \leftarrow$ check if s_l and s' intersect below p

$\mathcal{Q} \leftarrow$ check if s_r and s'' intersect below p

Algorithm

handleEvent(p)

$U(p) \leftarrow$ Line segments with p as upper endpoint

Stored with p in Q

$L(p) \leftarrow$ Line segments with p as lower endpoint

$C(p) \leftarrow$ Line segments with p as interior point

Neighbors in \mathcal{T}

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

 report p and $U(p) \cup L(p) \cup C(p)$

remove $L(p) \cup C(p)$ from \mathcal{T}

add $U(p) \cup C(p)$ to \mathcal{T}

Remove and insert
reverses order in $C(p)$

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

// s_l and s_r , neighbors of p in \mathcal{T}

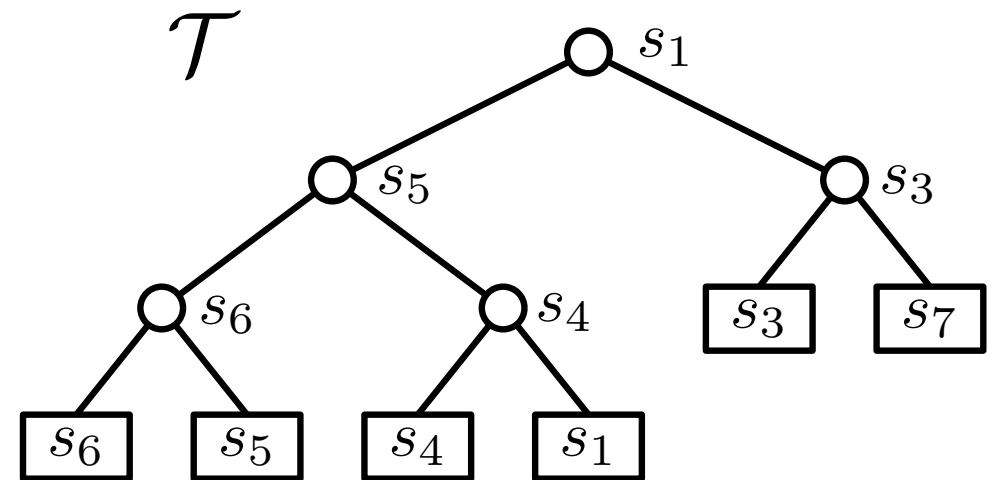
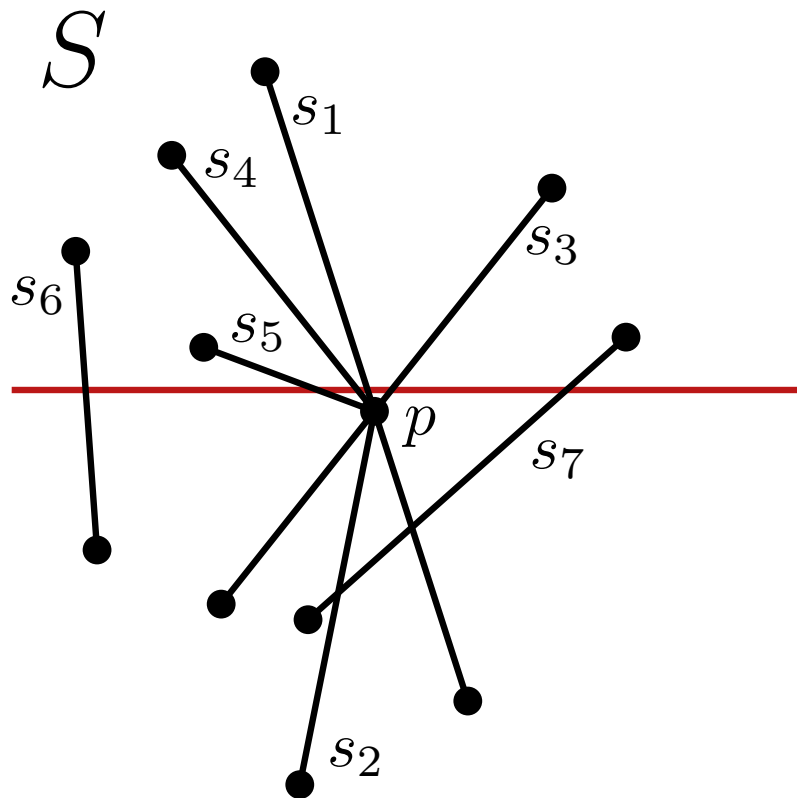
$Q \leftarrow$ check if s_l and s_r intersect below p

else // s' and s'' leftmost and rightmost line segment in $U(p) \cup C(p)$

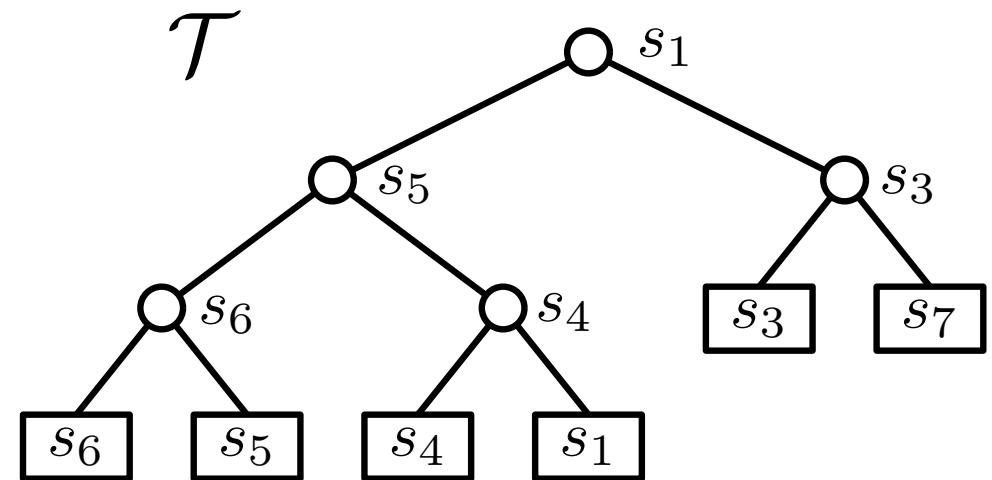
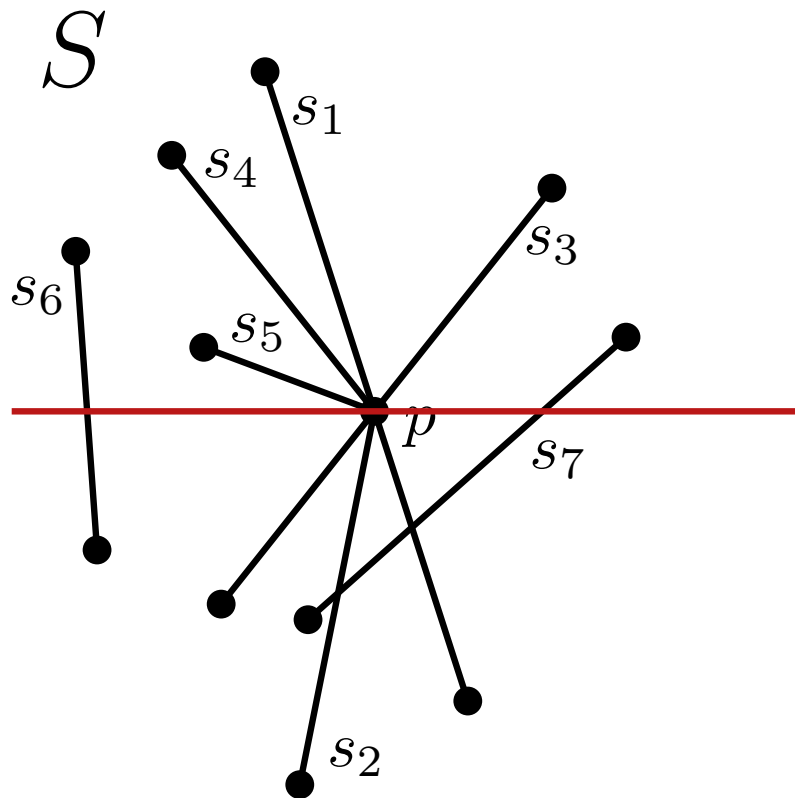
$Q \leftarrow$ check if s_l and s' intersect below p

$Q \leftarrow$ check if s_r and s'' intersect below p

What Happens Exactly?



What Happens Exactly?

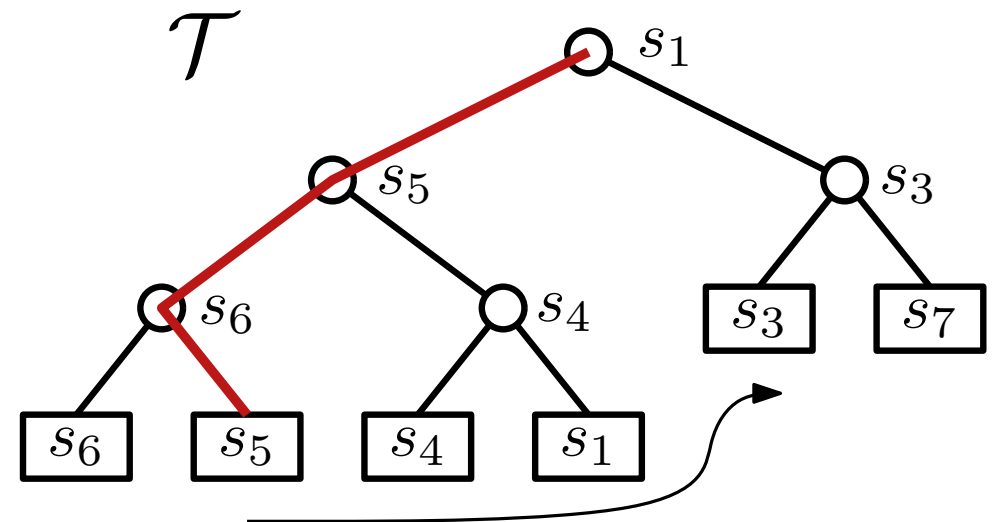
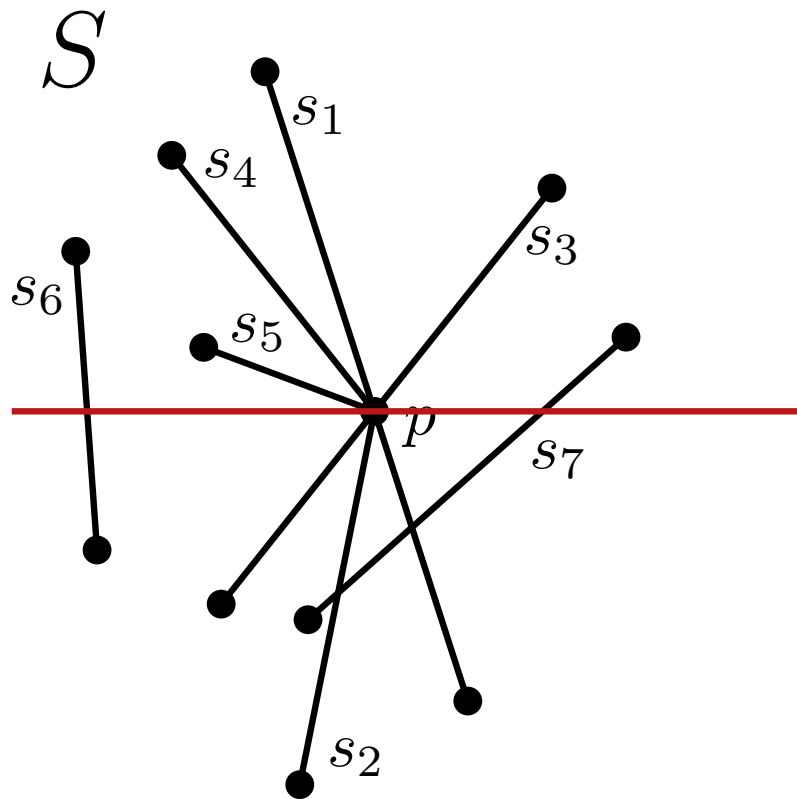


$$U(p) = \{s_2\}$$

$$L(p) =$$

$$C(p) =$$

What Happens Exactly?

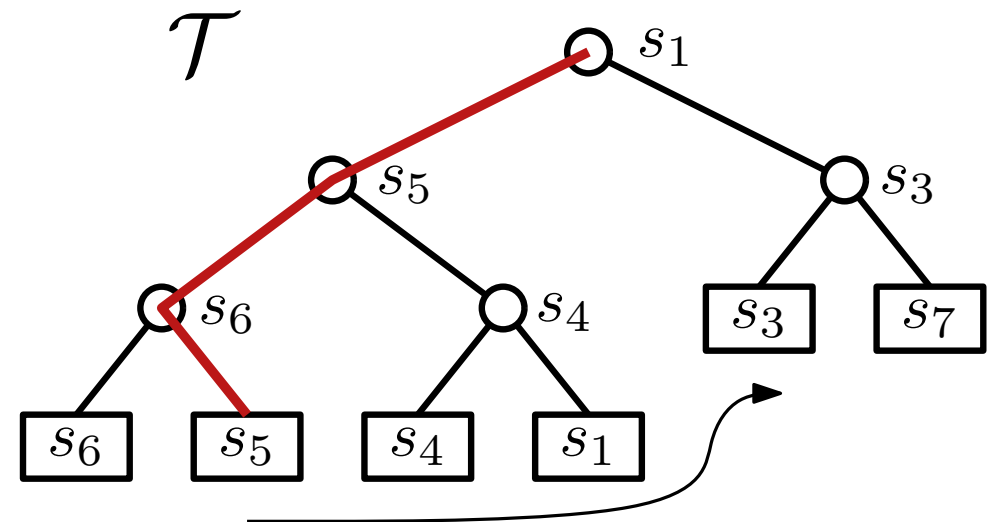
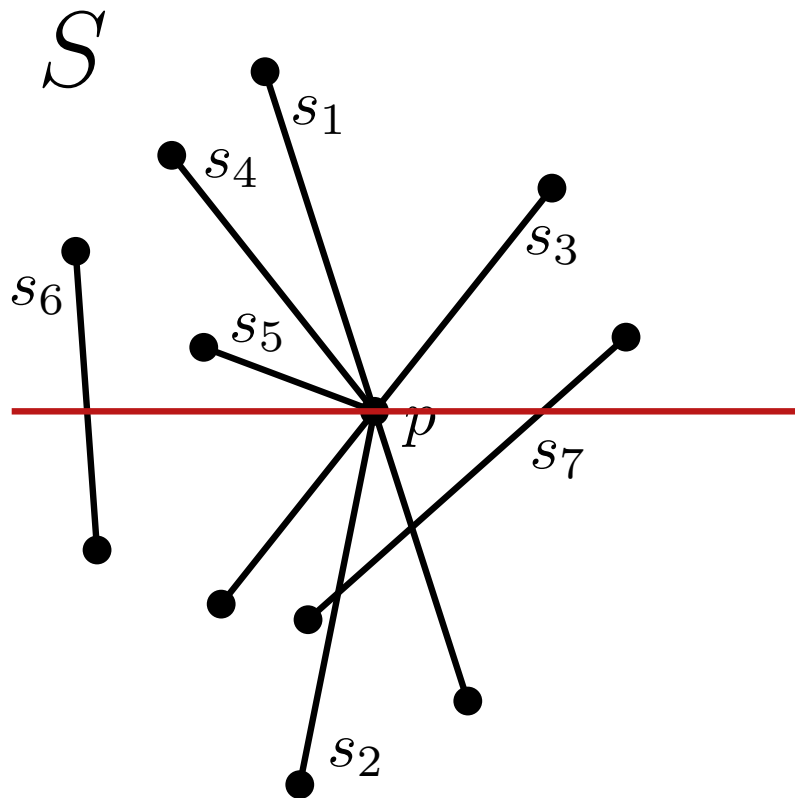


$$U(p) = \{s_2\}$$

$$L(p) =$$

$$C(p) =$$

What Happens Exactly?

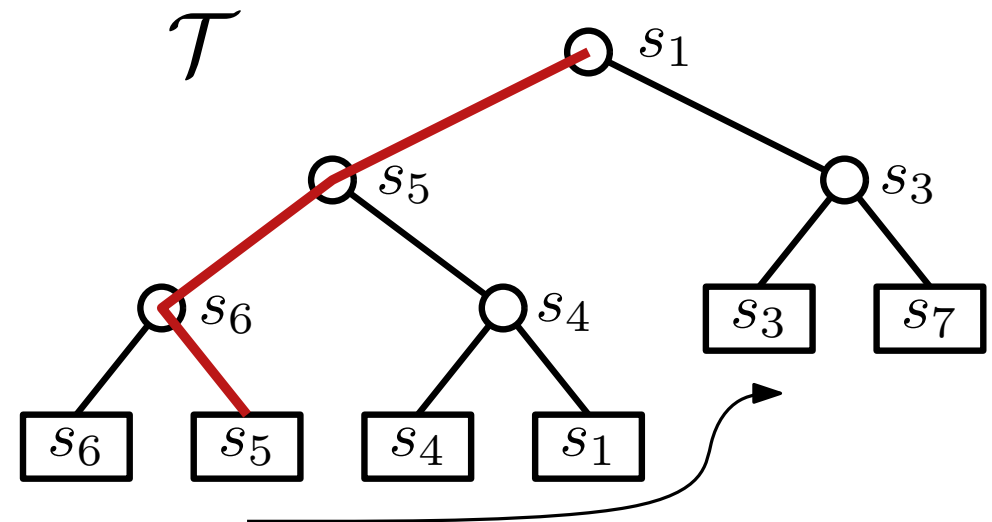
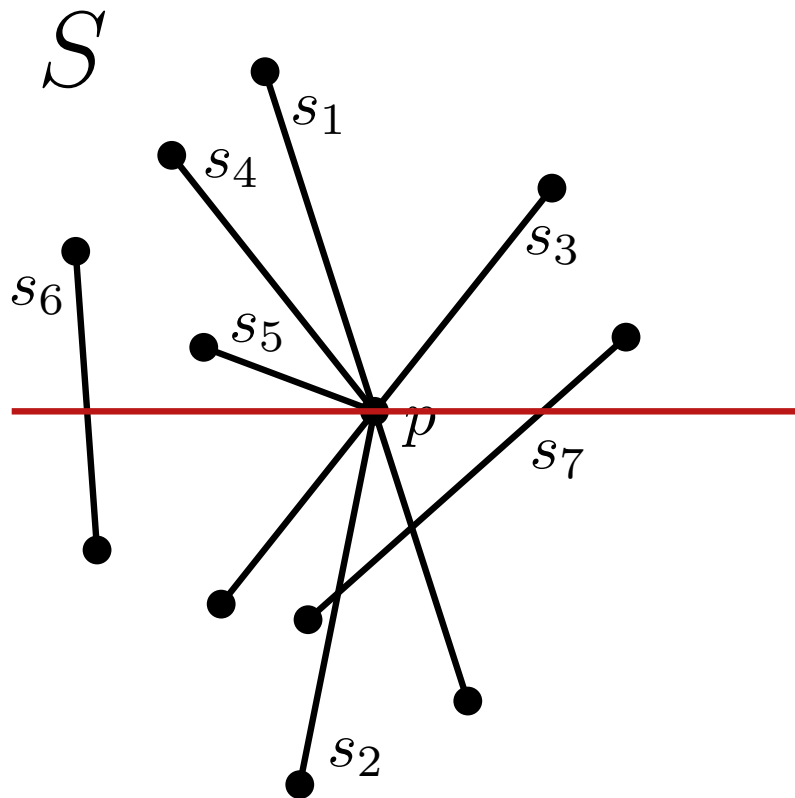


$$U(p) = \{s_2\}$$

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

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

What Happens Exactly?



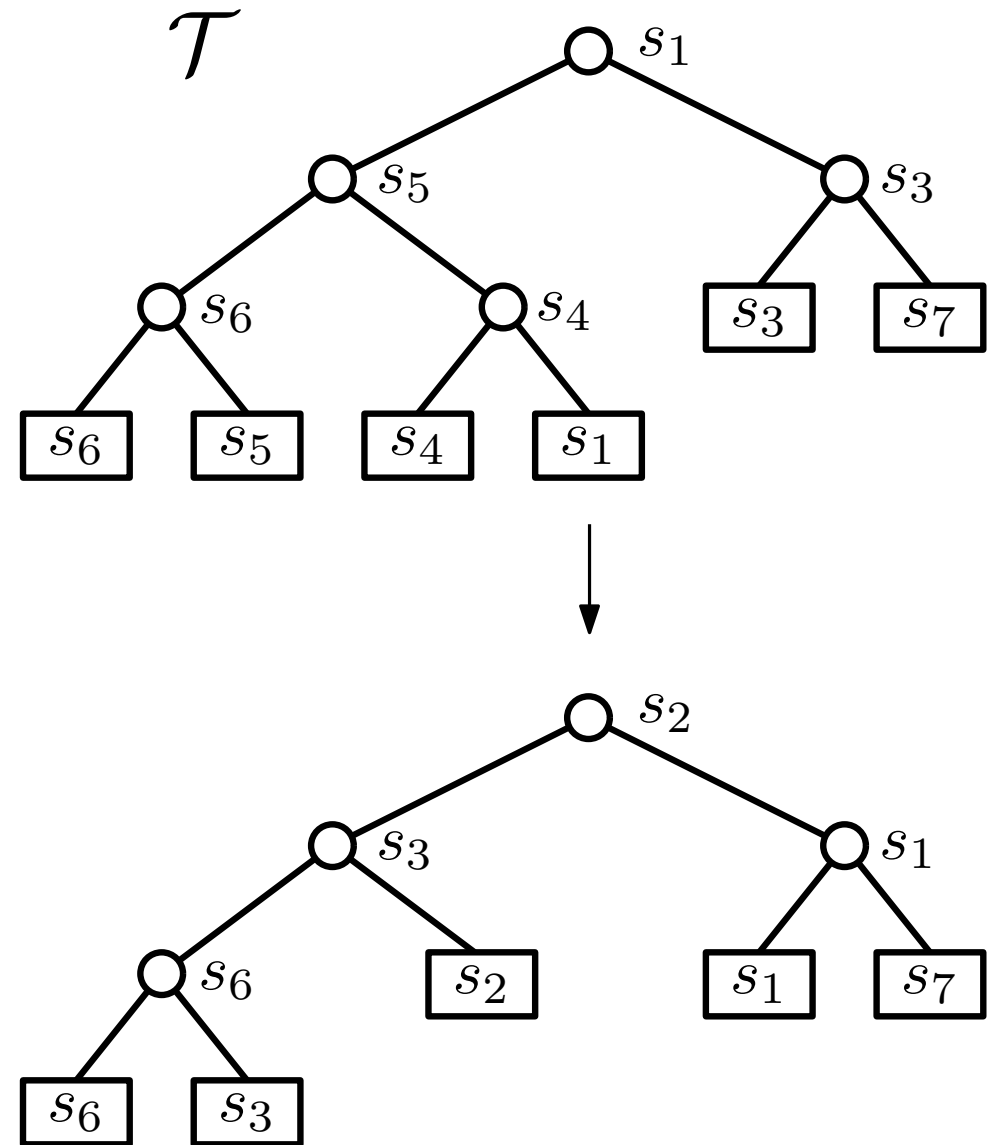
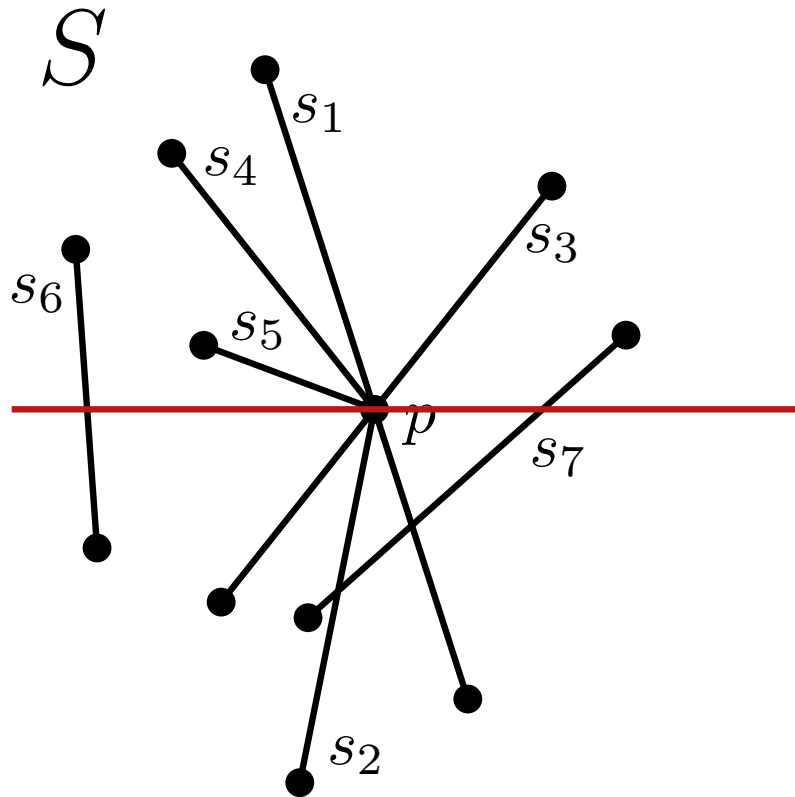
$$U(p) = \{s_2\}$$

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

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

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

What Happens Exactly?



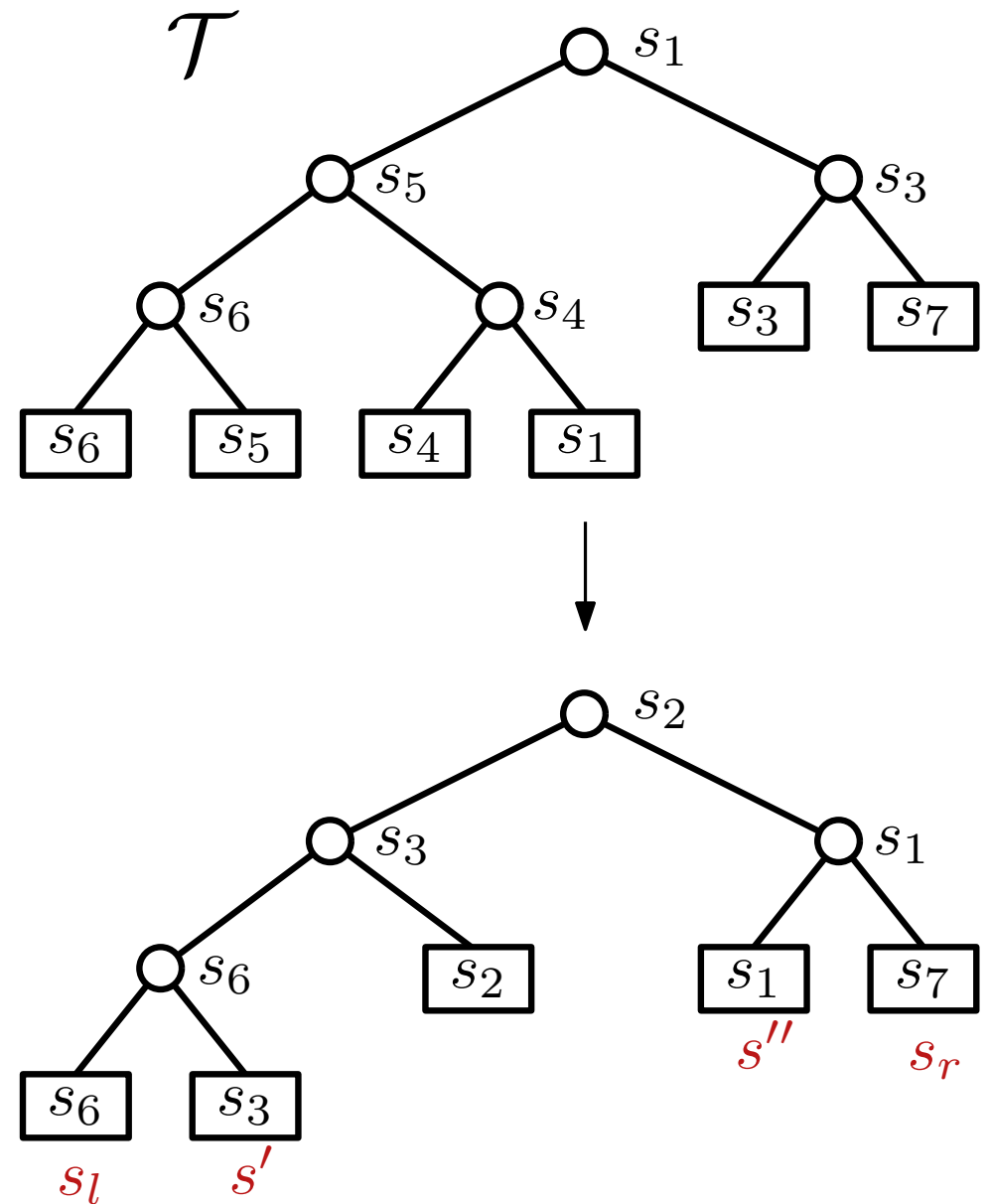
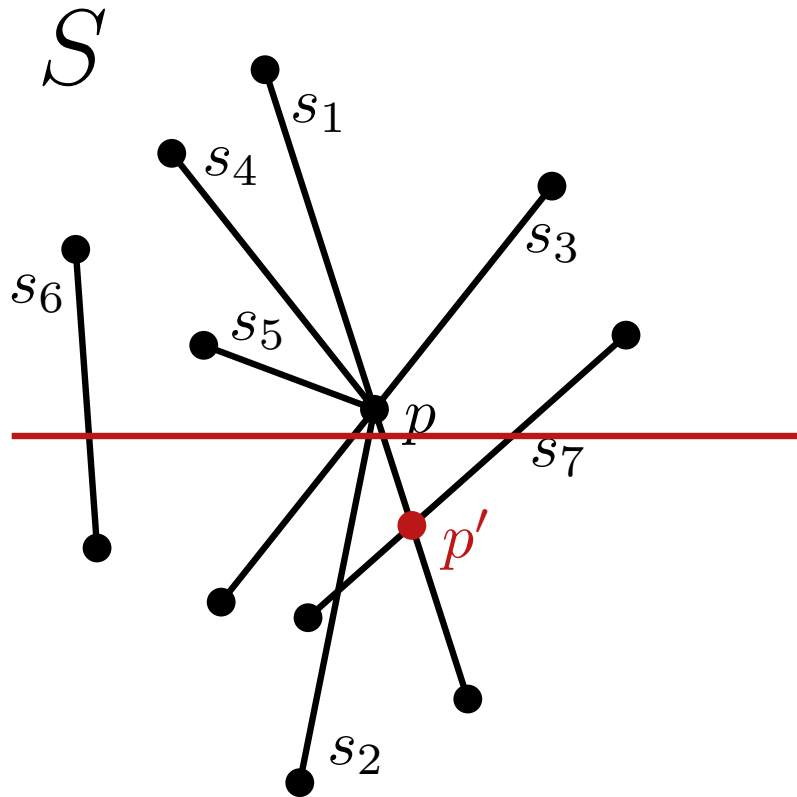
$$U(p) = \{s_2\}$$

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

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

Delete $L(p) \cup C(p)$; add $U(p) \cup C(p)$

What Happens Exactly?



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

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

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

Add event $p' = s_1 \times s_7$ in Q

Correctness

Lemma 1: Algorithm FindIntersections finds all intersection points and the line segments involved

Lemma 1: Algorithm FindIntersections finds all intersection points and the line segments involved

Proof:

Induction on the number of events processed, ordered by their priority.

Let p be an intersection point and all intersection points $q \prec p$ are already correctly computed.

Case 1: p is a line segment endpoint

- p was inserted in Q
- $U(p)$ are stored with p
- $L(p)$ and $C(p)$ are in \mathcal{T}

Lemma 1: Algorithm FindIntersections finds all intersection points and the line segments involved

Proof:

Induction on the number of events processed, ordered by their priority.

Let p be an intersection point and all intersection points $q \prec p$ are already correctly computed.

Case 1: p is a line segment endpoint

- p was inserted in Q
- $U(p)$ are stored with p
- $L(p)$ and $C(p)$ are in \mathcal{T}

Case 2: p is not a line segment endpoint

Consider why p must be in Q !

Running-Time Analysis

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersections with their line segments

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

```
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( $p$ )
```

Running-Time Analysis

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersections with their line segments

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

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)

Running-Time Analysis

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersections with their line segments

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

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

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

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

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

Running-Time Analysis

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersections with their line segments

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

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

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

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

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

Running-Time Analysis

FindIntersections(S)

Input: Set S of line segments

Output: Set of all intersections with their line segments

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

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

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

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

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

Running-Time Analysis

handleEvent(p)

$U(p) \leftarrow$ Line segments with p as upper endpoint

$L(p) \leftarrow$ Line segments with p as lower endpoint

$C(p) \leftarrow$ Line segments with p as interior point

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

 | return p and $U(p) \cup L(p) \cup C(p)$

remove $L(p) \cup C(p)$ from \mathcal{T}

add $U(p) \cup C(p)$ to \mathcal{T}

if $U(p) \cup C(p) = \emptyset$ **then** // s_l and s_r , neighbors of p in \mathcal{T}

 | $Q \leftarrow$ check if s_l and s_r intersect below p

else // s' and s'' leftmost and rightmost line segment in $U(p) \cup C(p)$

 | $Q \leftarrow$ check if s_l and s' intersect below p

 | $Q \leftarrow$ check if s_r and s'' intersect below p

Lemma 2: Algorithm FindIntersections has running time $O(n \log n + I \log n)$, where I is the number of intersection points.

Summary

Thm 1: Let S be a set of n line segments in the plane. Then we can compute intersections in S together with the involved line segments in $O((n + I) \log n)$ time and $O(?)$ space.

Summary

Thm 1: Let S be a set of n line segments in the plane. Then we can compute intersections in S together with the involved line segments in $O((n + I) \log n)$ time and $O(?)$ space.

Proof:

- Correctness ✓
- Running time ✓
- Space

Summary

Thm 1: Let S be a set of n line segments in the plane. Then we can compute intersections in S together with the involved line segments in $O((n + I) \log n)$ time and $O(?)$ space.

Proof:

- Correctness ✓
- Running time ✓
- Space

Consider how much space the data structures need!

Summary

Thm 1: Let S be a set of n line segments in the plane. Then we can compute intersections in S together with the involved line segments in $O((n + I) \log n)$ time and $O(n)$ space.

Proof:

- Correctness ✓
- Running time ✓
- Space

Consider how much space the data structures need!

- \mathcal{T} has at most n elements
- \mathcal{Q} has at most $O(n + I)$ elements
- reduction of \mathcal{Q} to $O(n)$ space: an exercise

Is the Sweep-Line Algorithm always better than the naive one?

Is the Sweep-Line Algorithm always better than the naive one?

No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Discussion

Is the Sweep-Line Algorithm always better than the naive one?

No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Can we do better?

Is the Sweep-Line Algorithm always better than the naive one?

No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Can we do better?

Yes, in $\Theta(n \log n + I)$ time and $\Theta(n)$ space [Balaban, 1995].

Is the Sweep-Line Algorithm always better than the naive one?

No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Can we do better?

Yes, in $\Theta(n \log n + I)$ time and $\Theta(n)$ space [Balaban, 1995].

How does this solve the map overlay problem?

Is the Sweep-Line Algorithm always better than the naive one?

No, because if $I \in \Omega(n^2)$ then the algorithm has running time $O(n^2 \log n)$.

Can we do better?

Yes, in $\Theta(n \log n + I)$ time and $\Theta(n)$ space [Balaban, 1995].

How does this solve the map overlay problem?

Using an appropriate data structure (**doubly-connected edgelist**) for planar graphs we can compute in $O((n + I) \log n)$ time the overlay of two maps.

(Details in Ch. 2.3 of the book)