Karlsruhe Institute of Technology

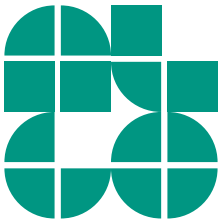
Vorlesung Algorithmische Geometrie

Streckenschnitte

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

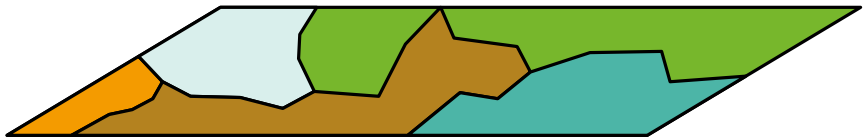
Martin Nöllenburg

19.04.2011



Überlagern von Kartenebenen

Beispiel: Gegeben zwei verschiedene Kartenebenen, bestimme deren Schnitt.

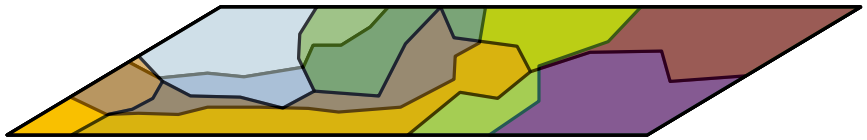


Flächennutzung

+

Niederschlag

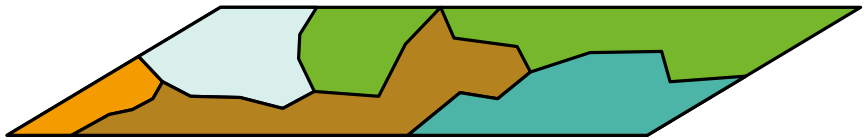
=



Kombination beider Themen

Überlagern von Kartenebenen

Beispiel: Gegeben zwei verschiedene Kartenebenen, bestimme deren Schnitt.

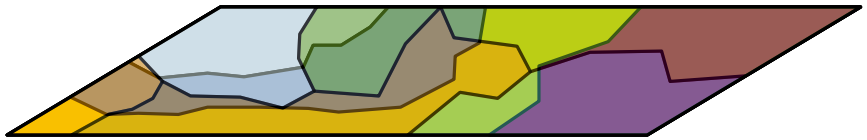


Flächennutzung

+

Niederschlag

=



Kombination beider Themen

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

Problemstellung

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

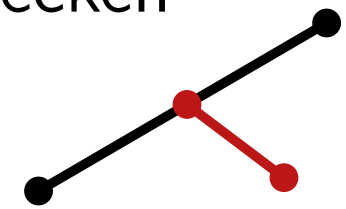
Ges:
- alle Schnittpunkte von zwei oder mehr Strecken
- für jeden Schnittpunkt die beteiligten Strecken

Problemstellung

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

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

Def: Strecken sind **abgeschlossene** Mengen

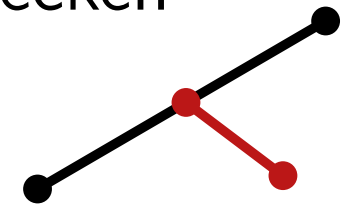


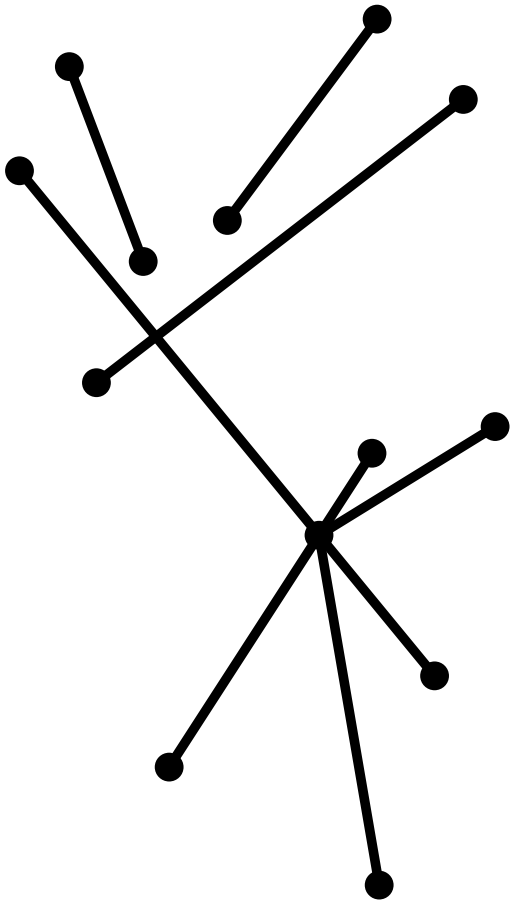
Problemstellung

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

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

Def: Strecken sind **abgeschlossene** Mengen





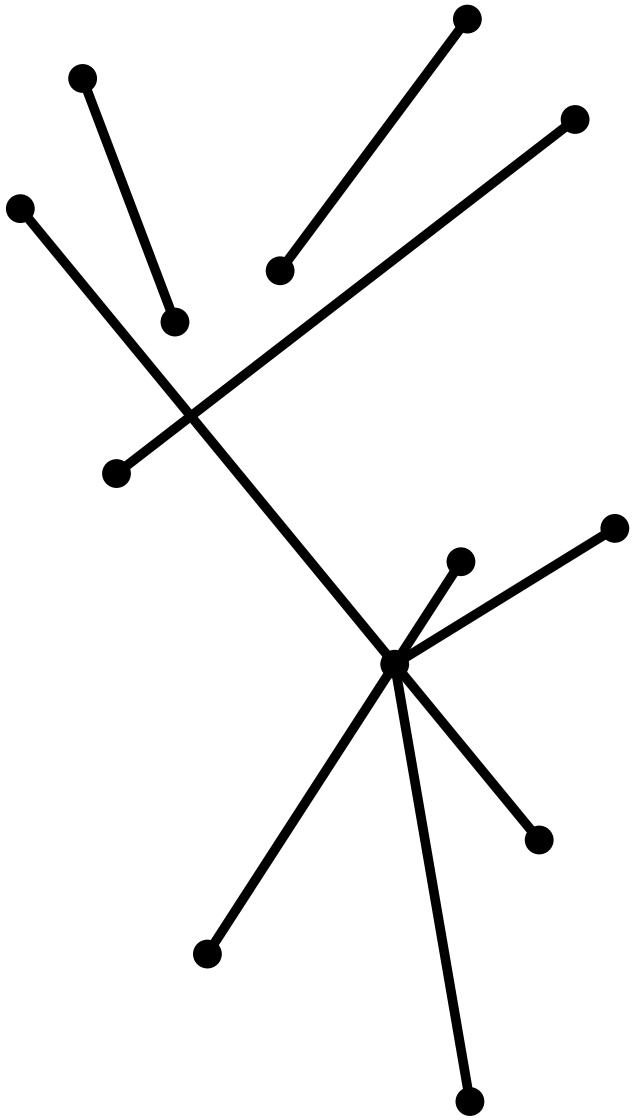
Diskussion:

– Wie kann man das Problem naiv lösen?

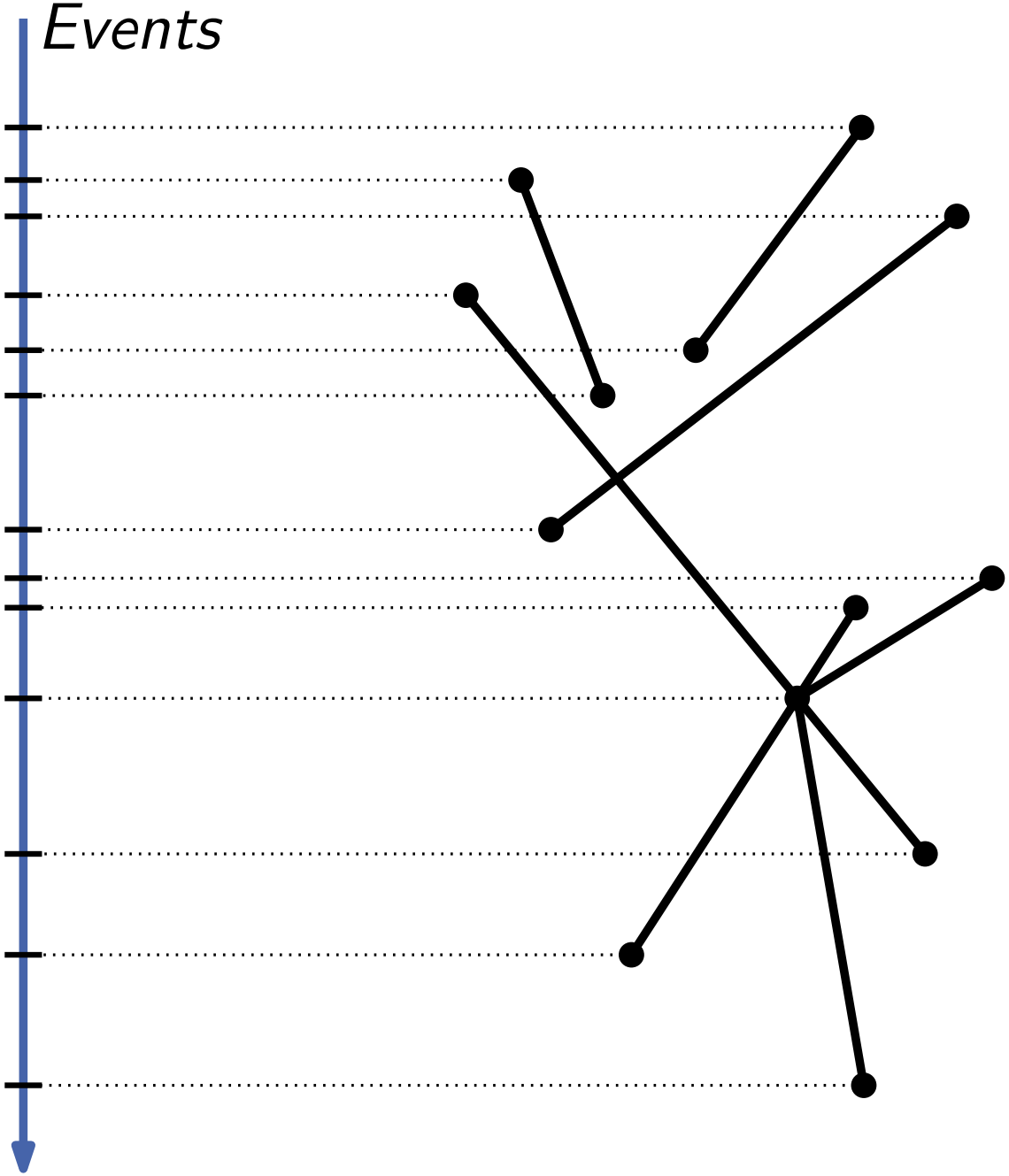
– Ist das evtl. schon optimal?

– Seht ihr Verbesserungsansätze?

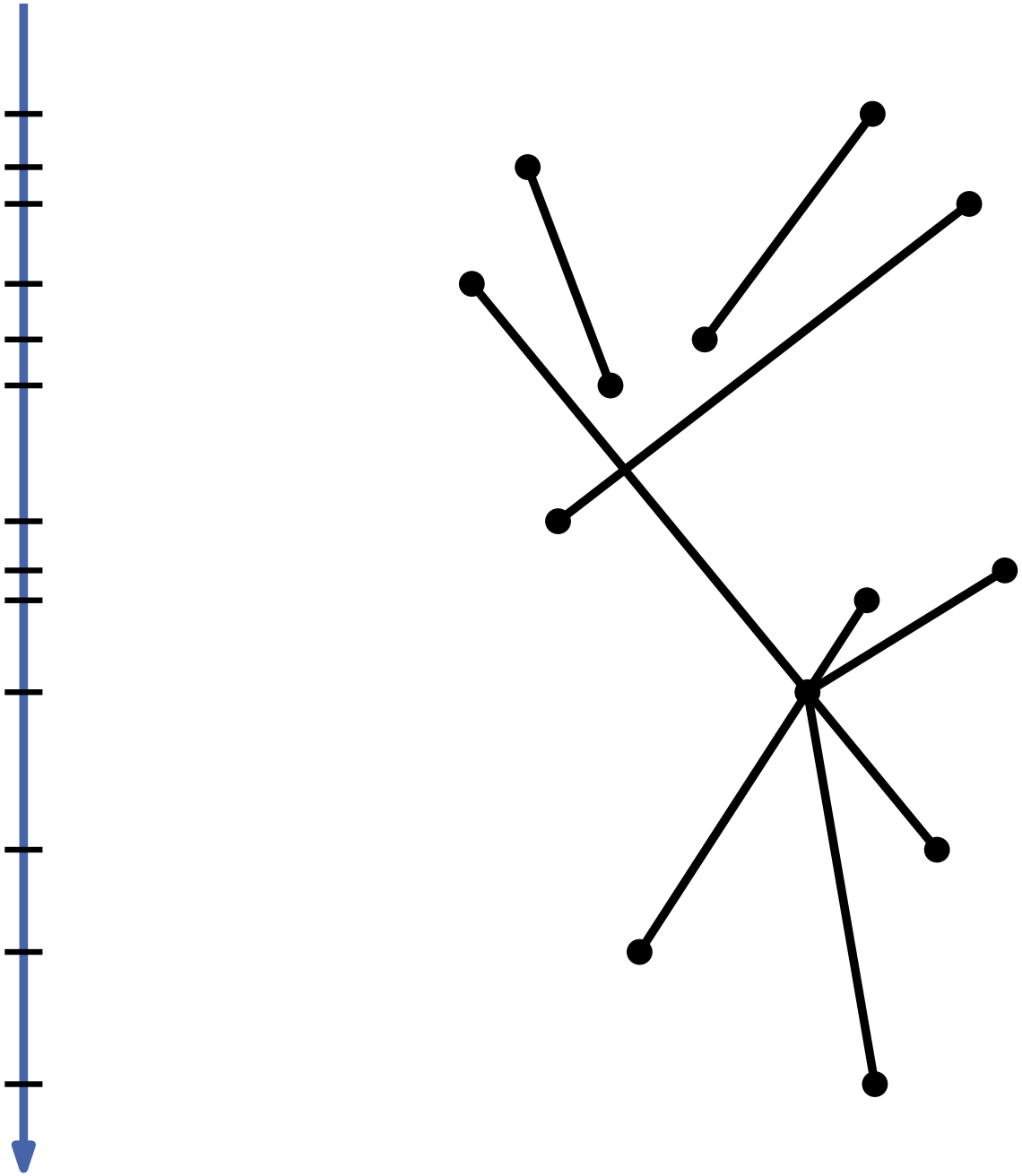
Das Sweep-Line Verfahren: Beispiel



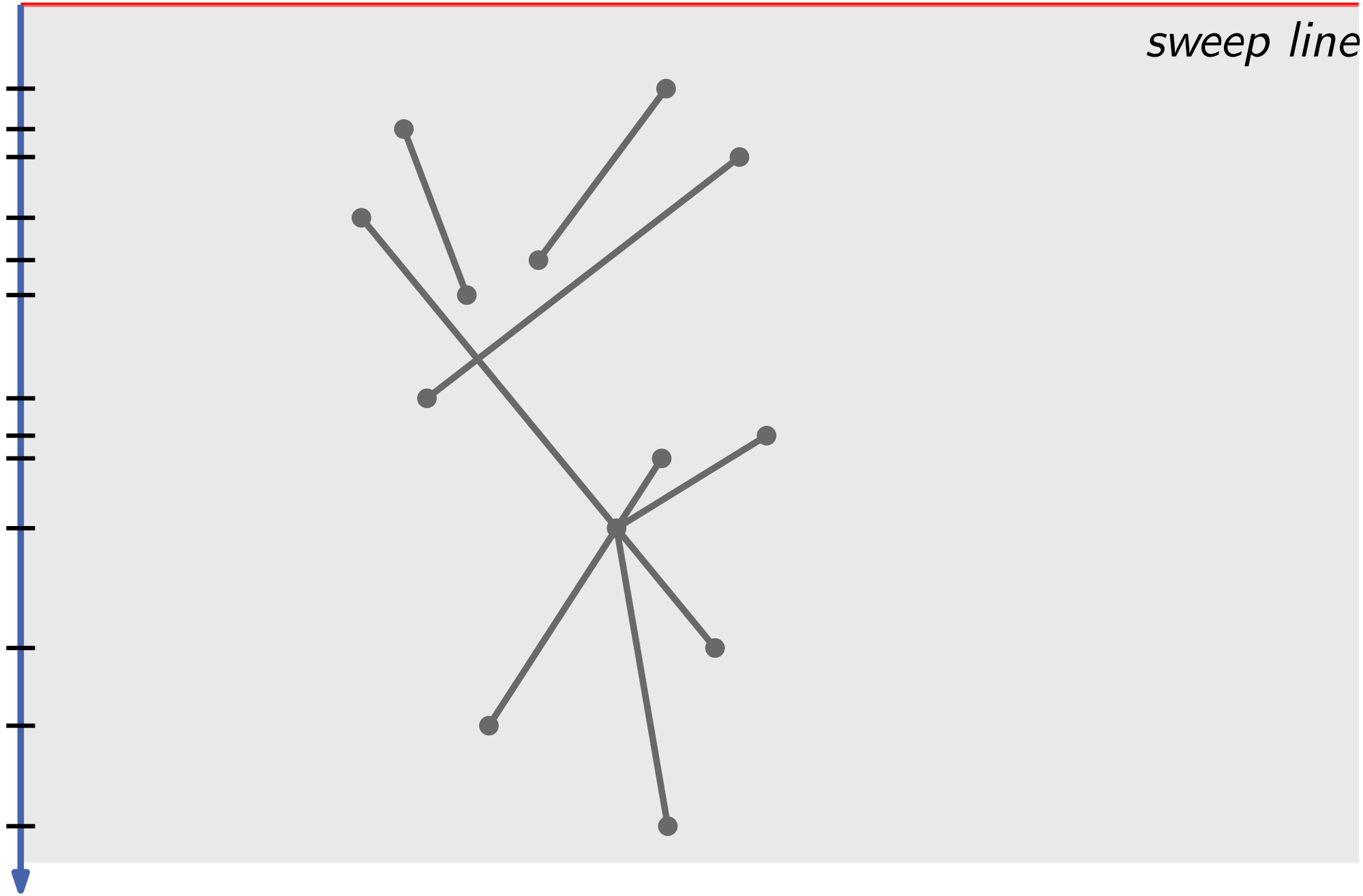
Das Sweep-Line Verfahren: Beispiel



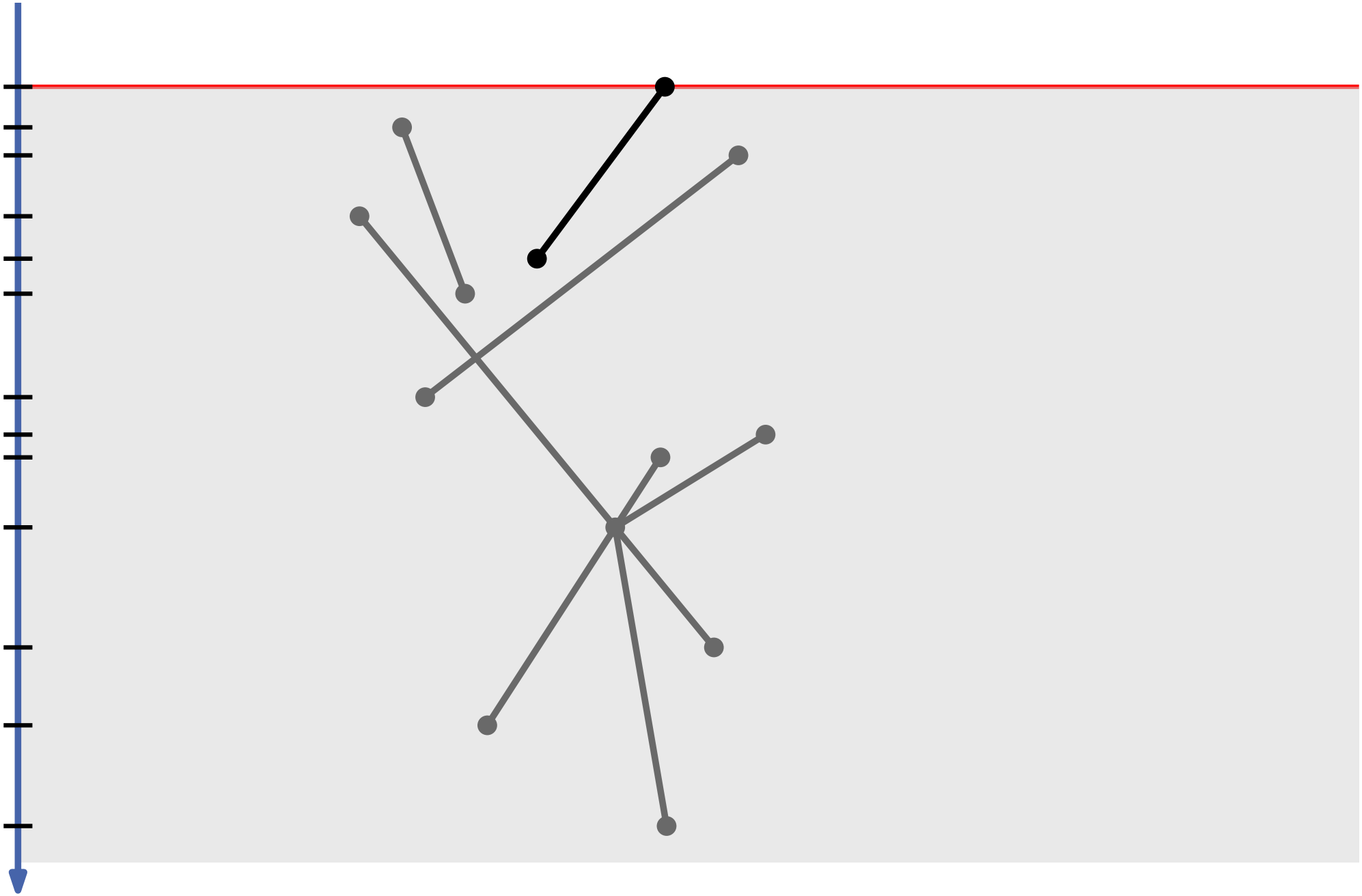
Das Sweep-Line Verfahren: Beispiel



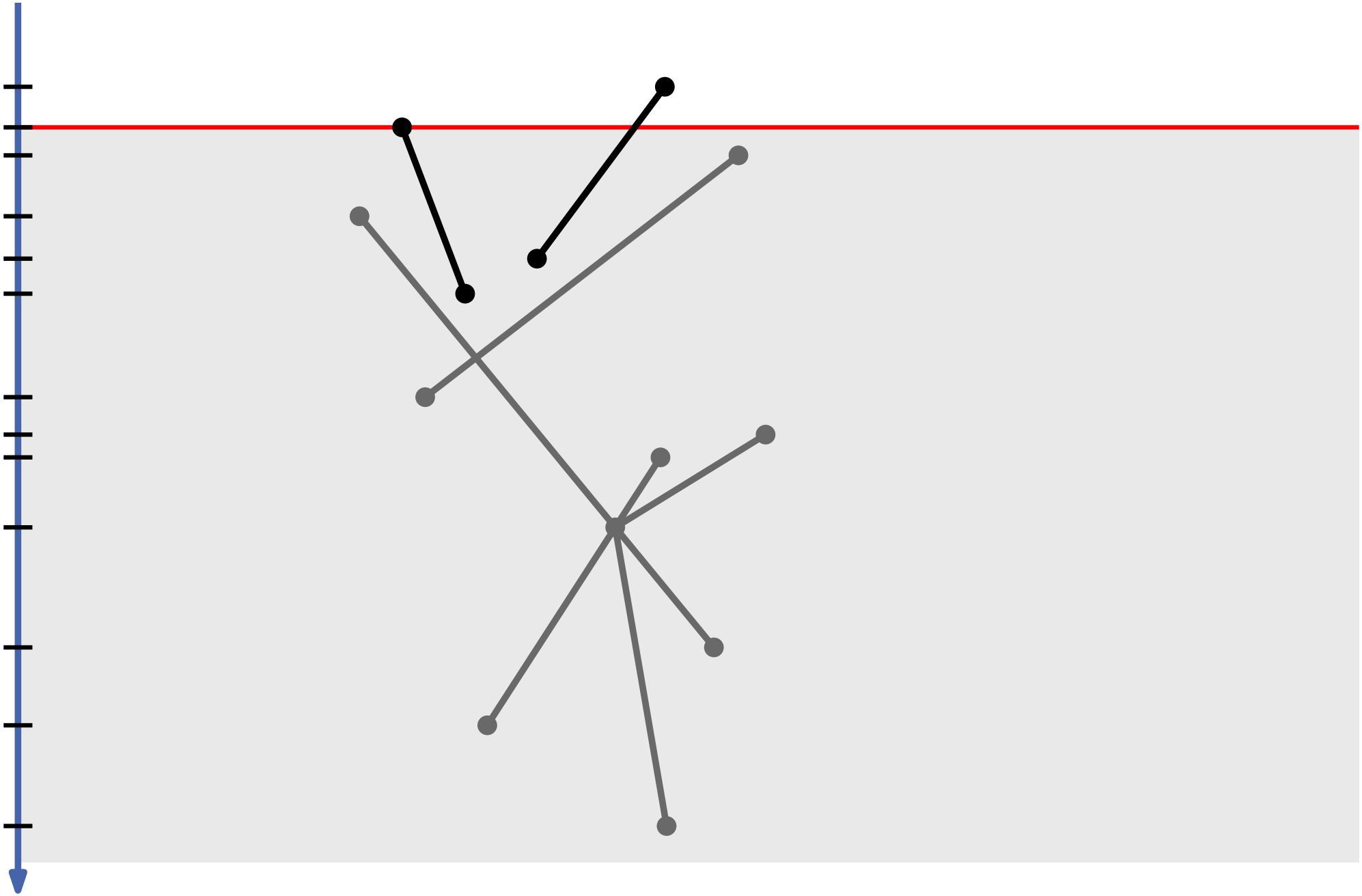
Das Sweep-Line Verfahren: Beispiel



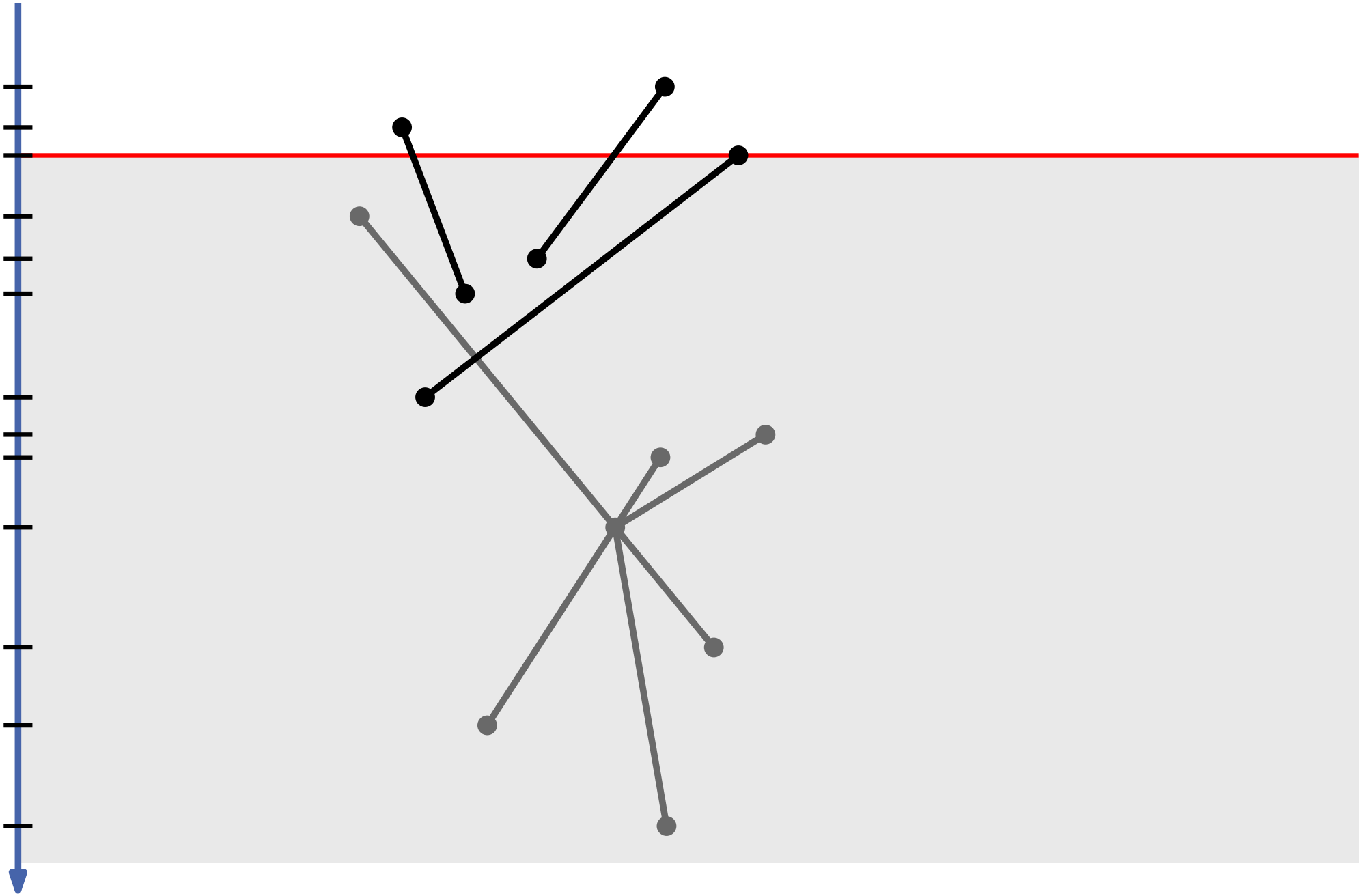
Das Sweep-Line Verfahren: Beispiel



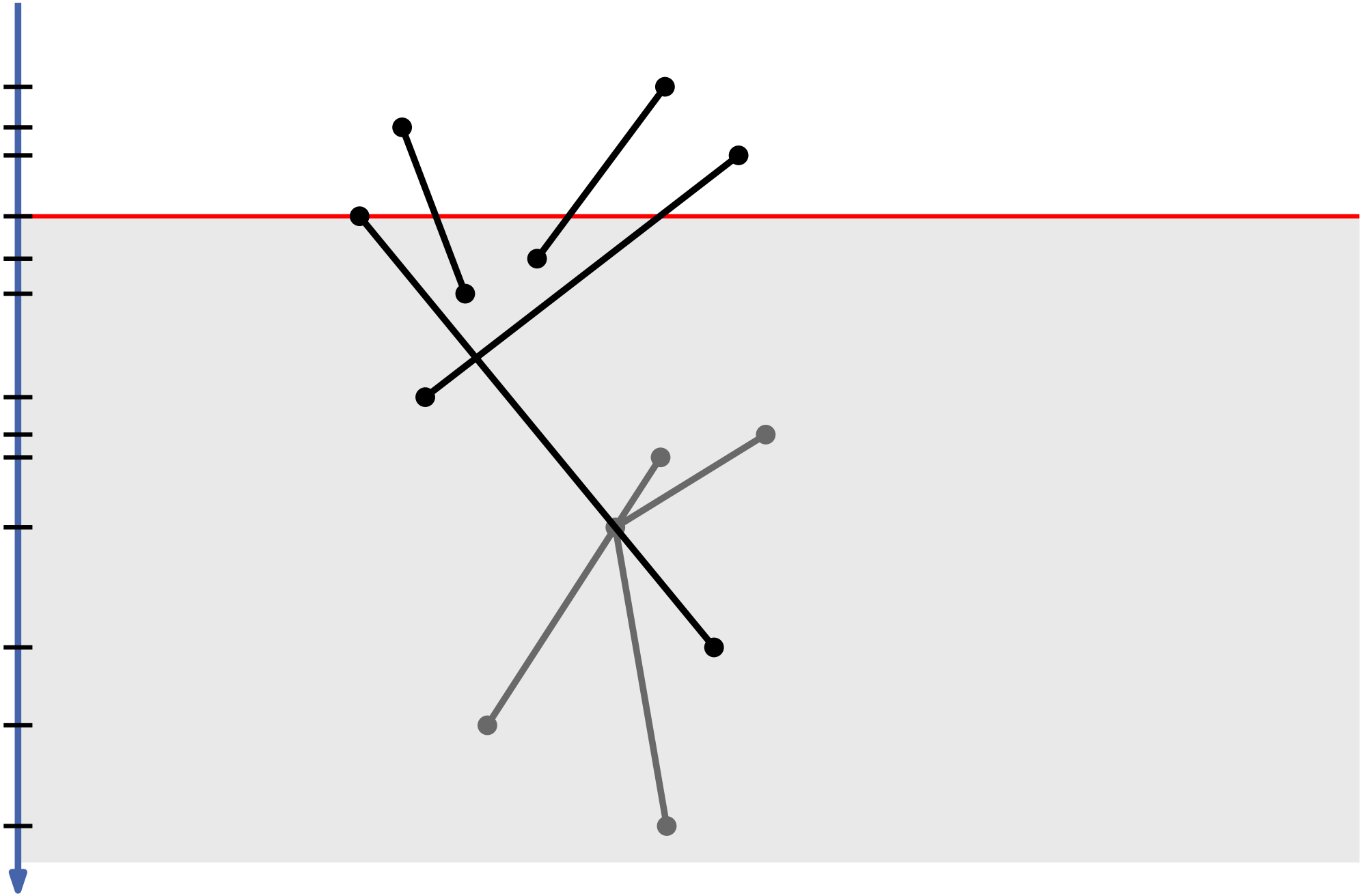
Das Sweep-Line Verfahren: Beispiel



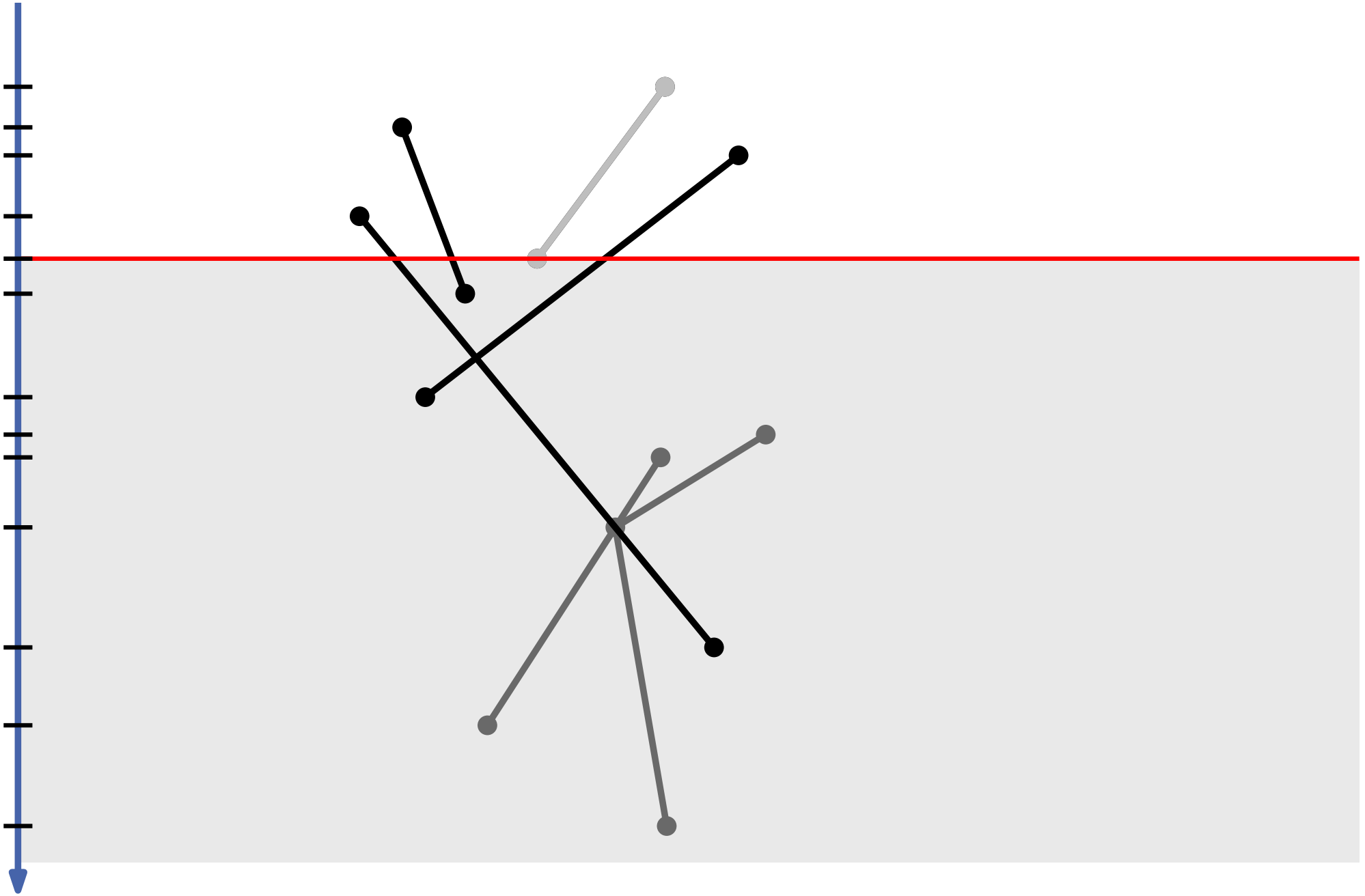
Das Sweep-Line Verfahren: Beispiel



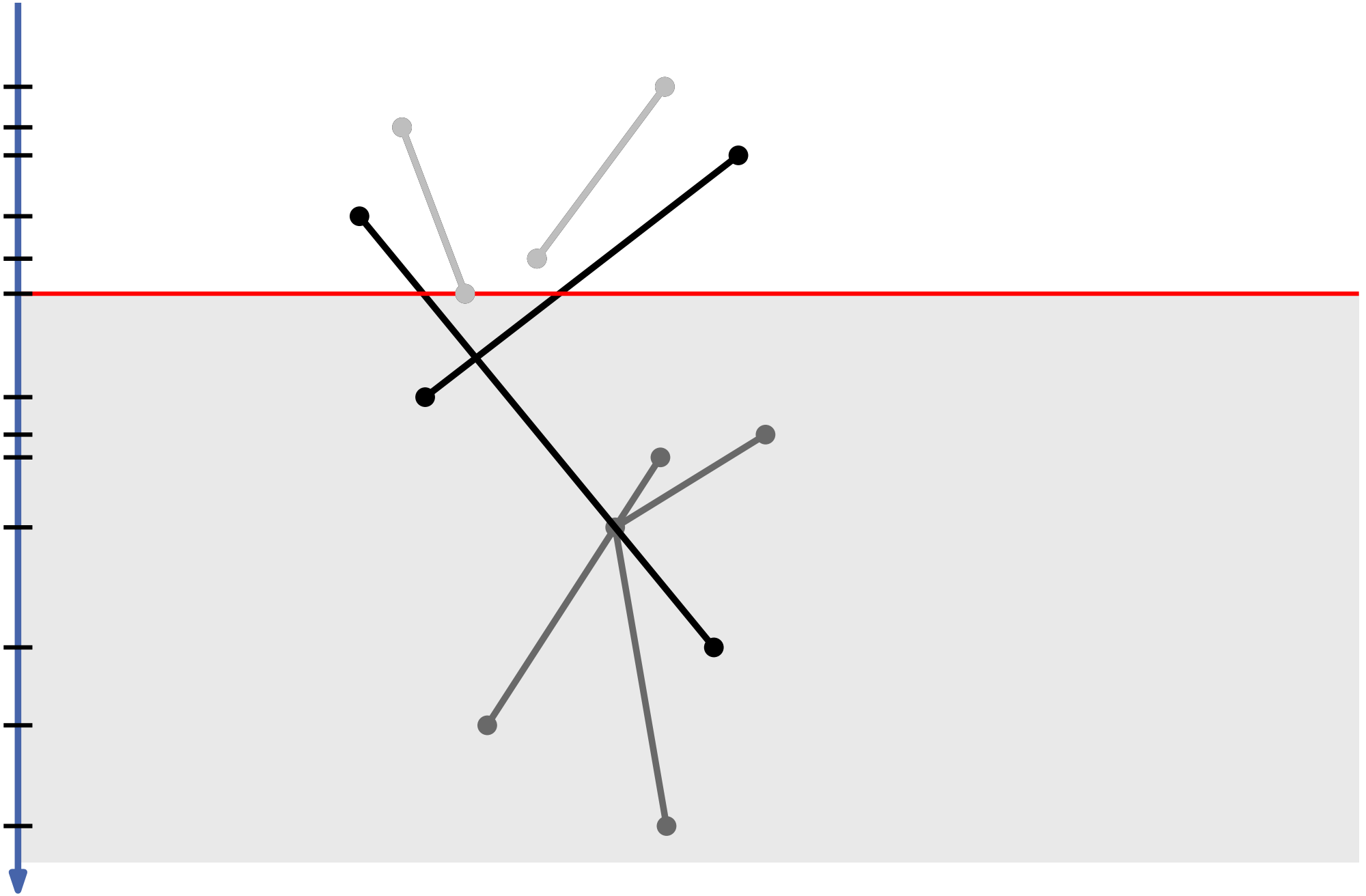
Das Sweep-Line Verfahren: Beispiel



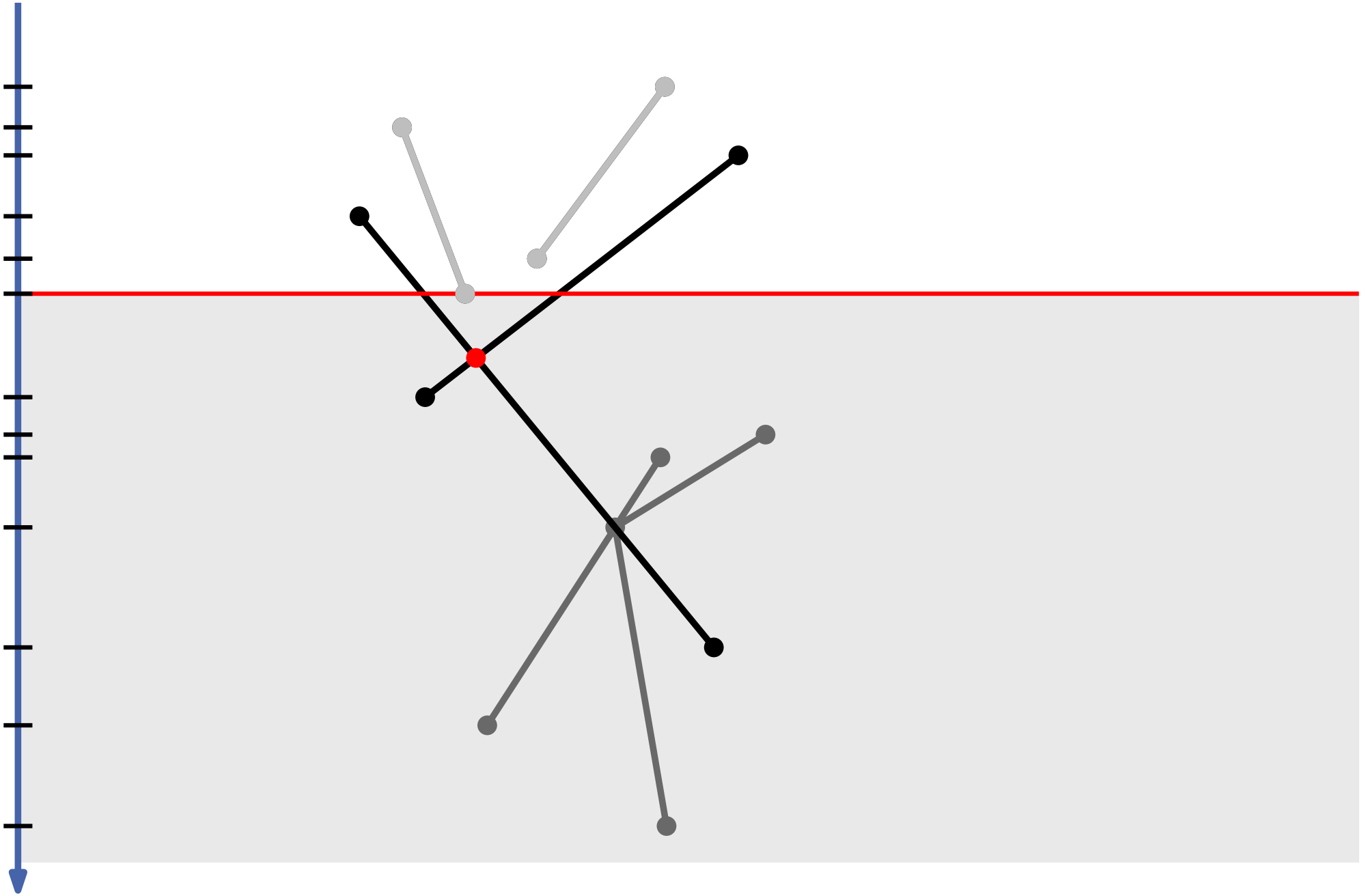
Das Sweep-Line Verfahren: Beispiel



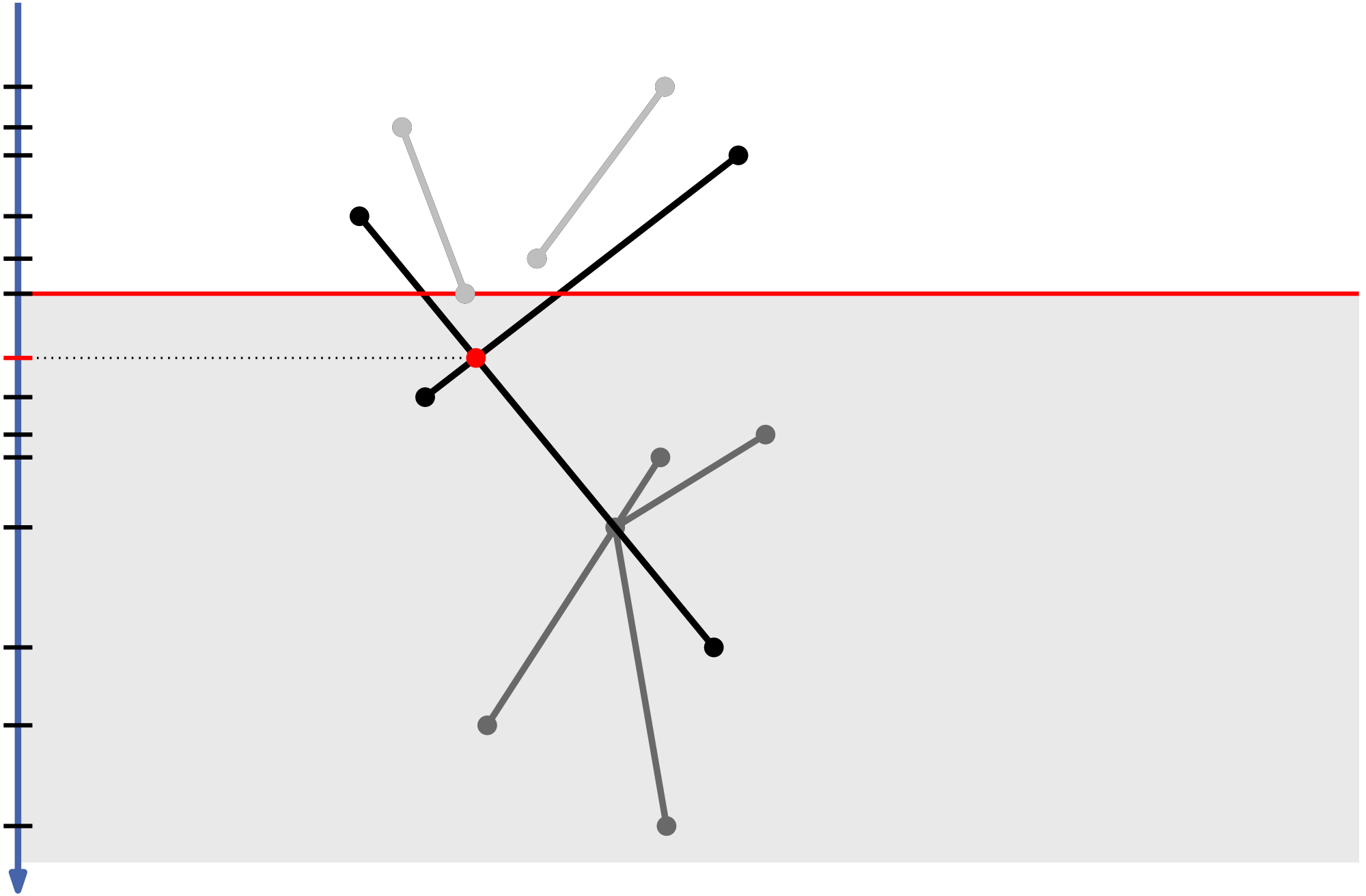
Das Sweep-Line Verfahren: Beispiel



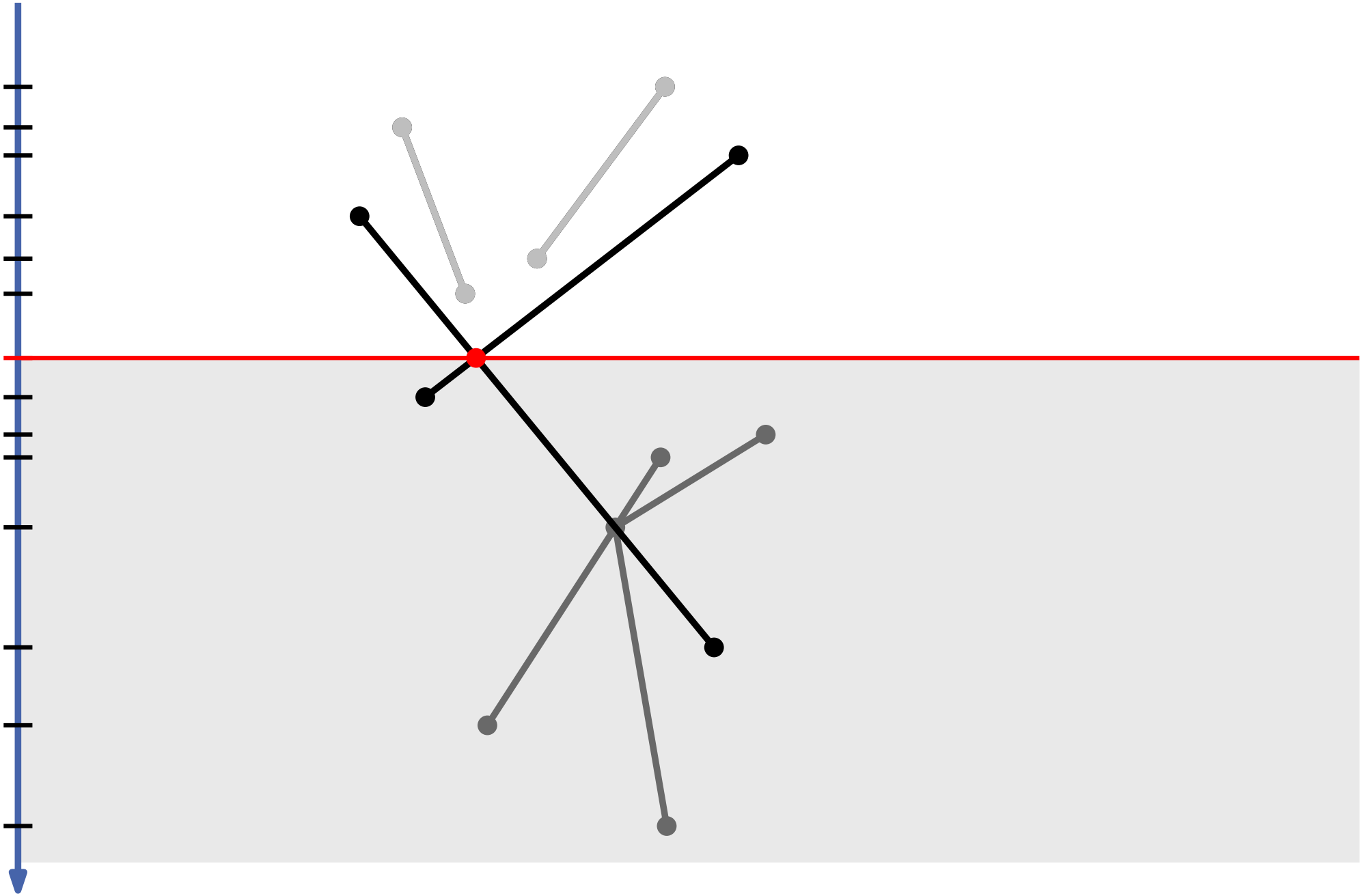
Das Sweep-Line Verfahren: Beispiel



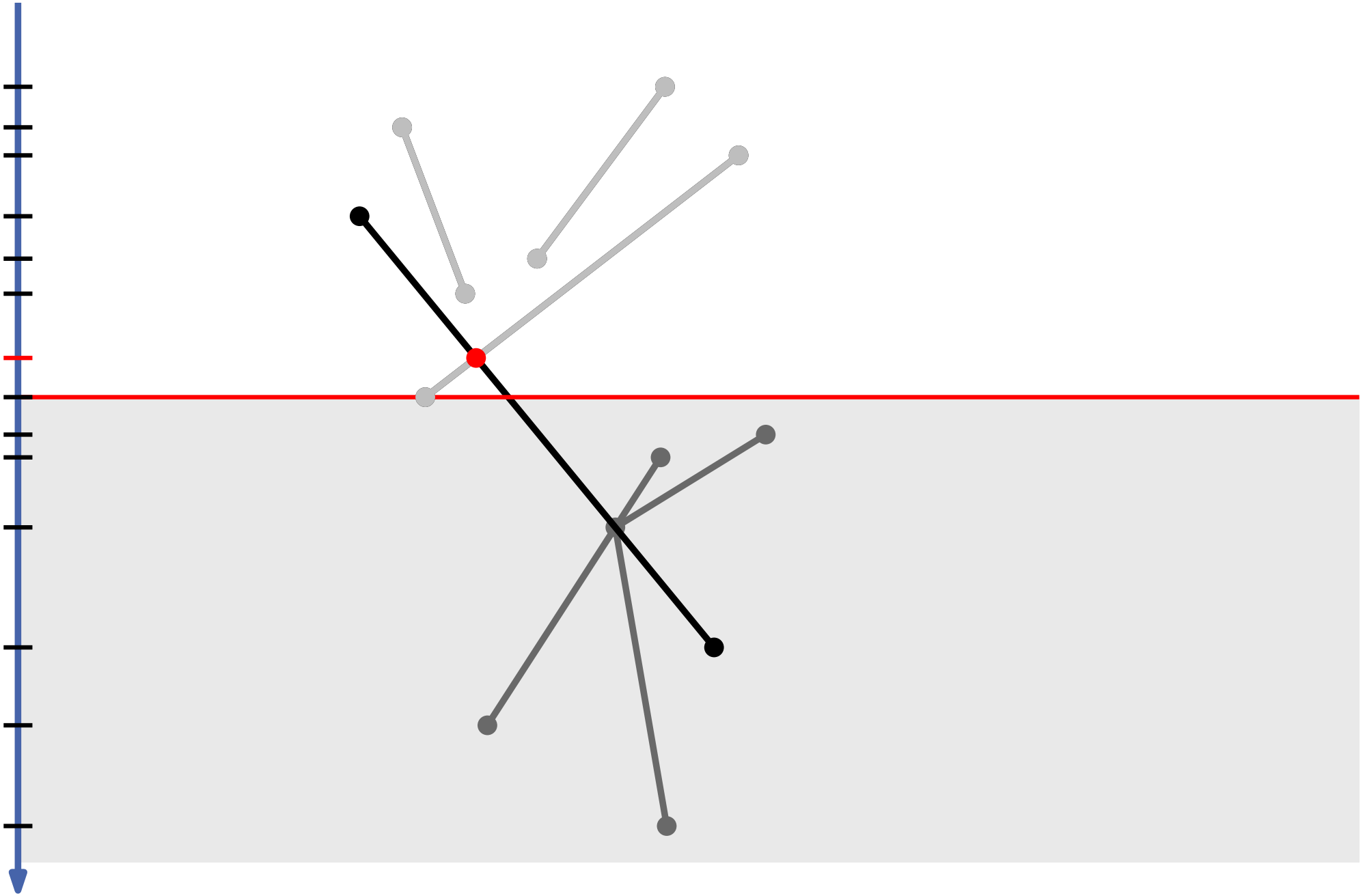
Das Sweep-Line Verfahren: Beispiel



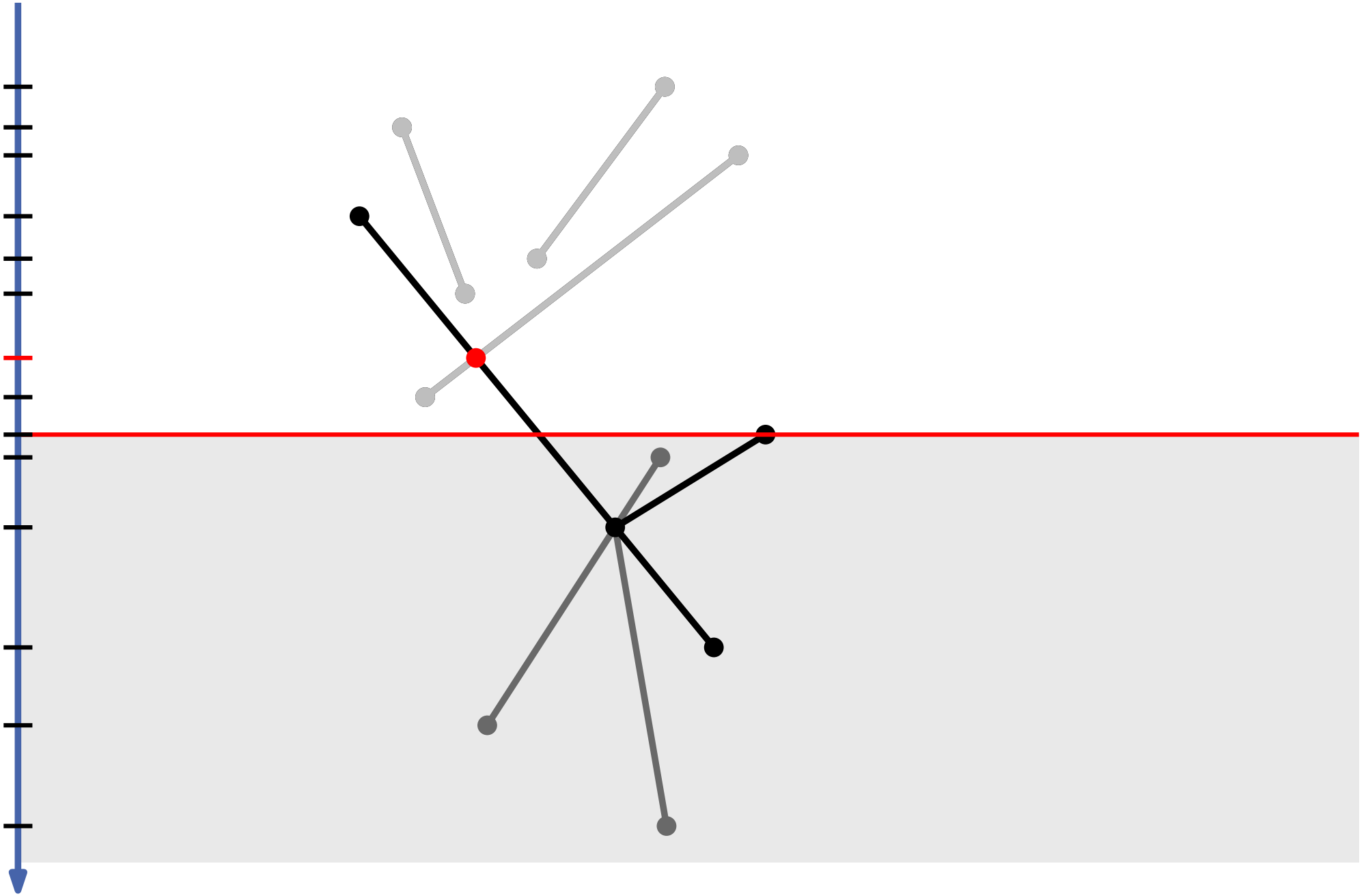
Das Sweep-Line Verfahren: Beispiel



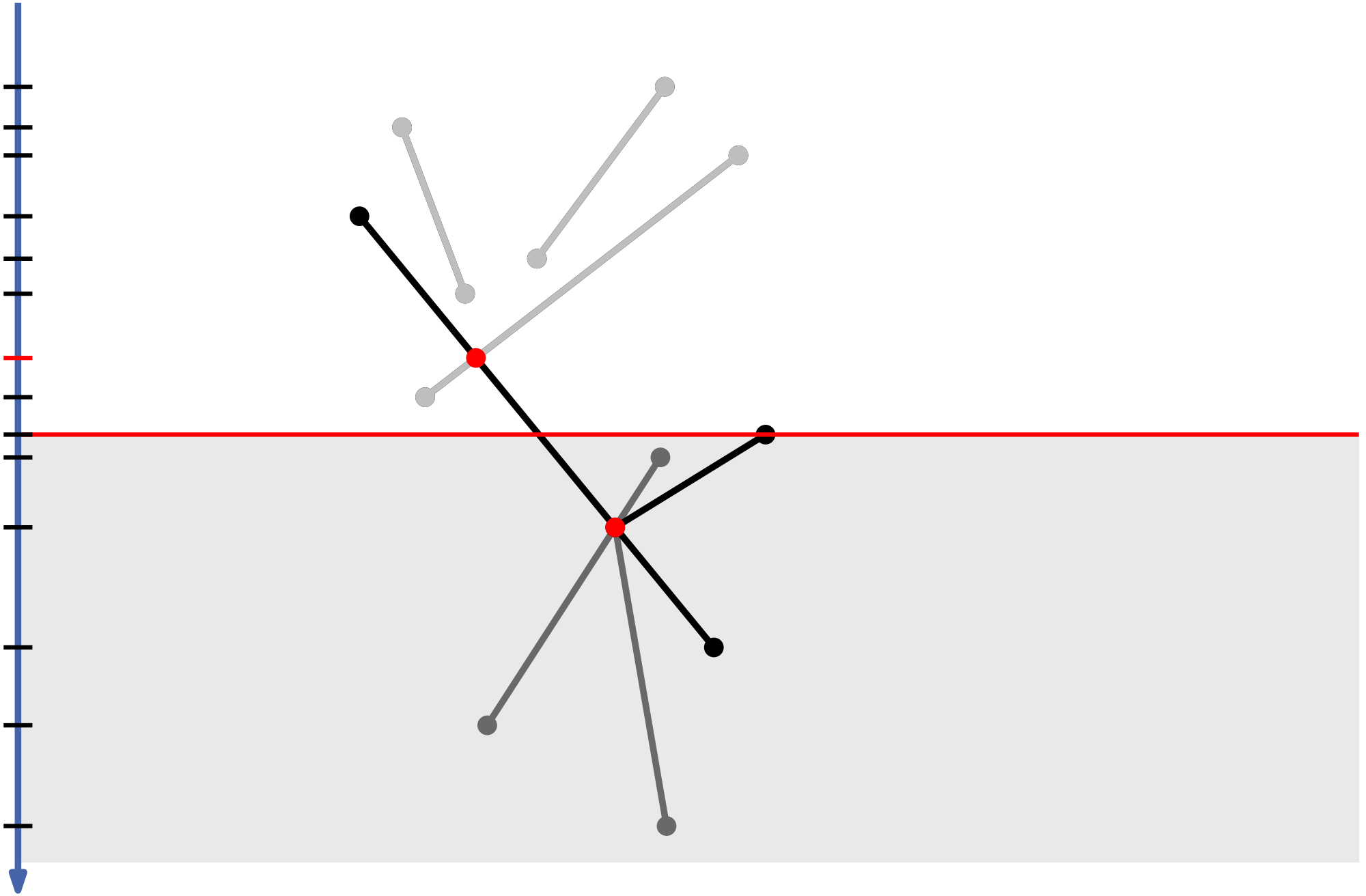
Das Sweep-Line Verfahren: Beispiel



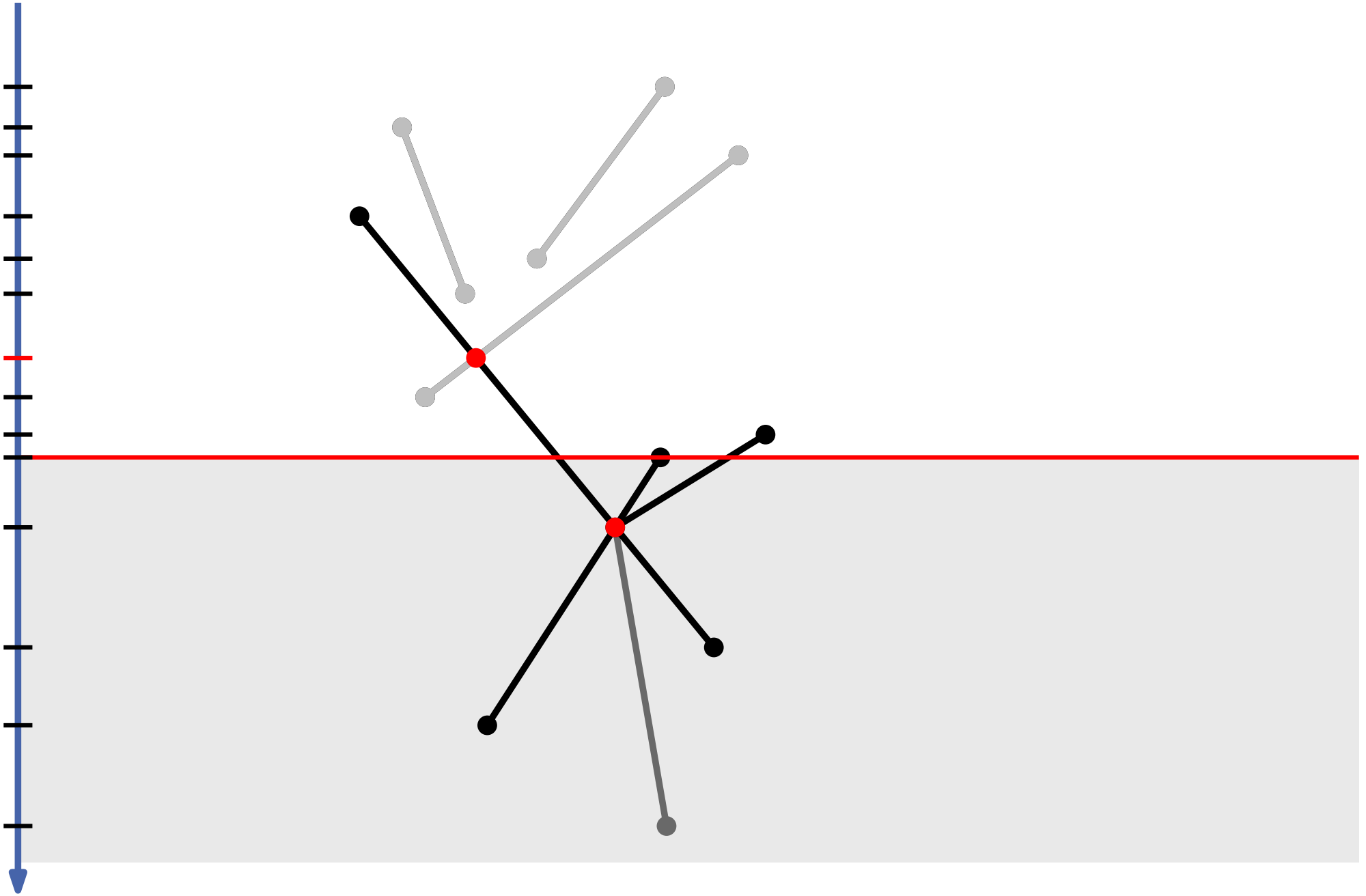
Das Sweep-Line Verfahren: Beispiel



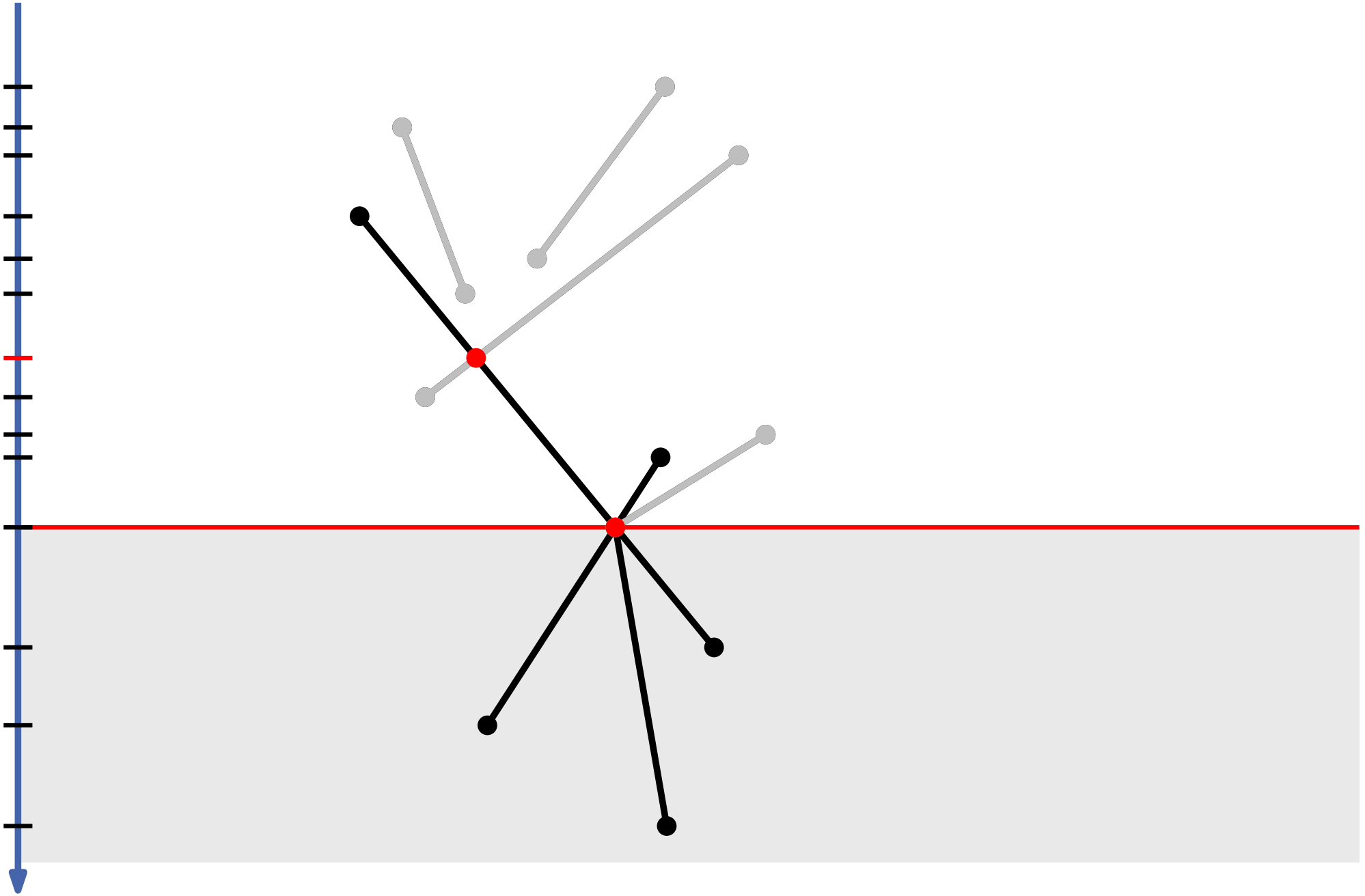
Das Sweep-Line Verfahren: Beispiel



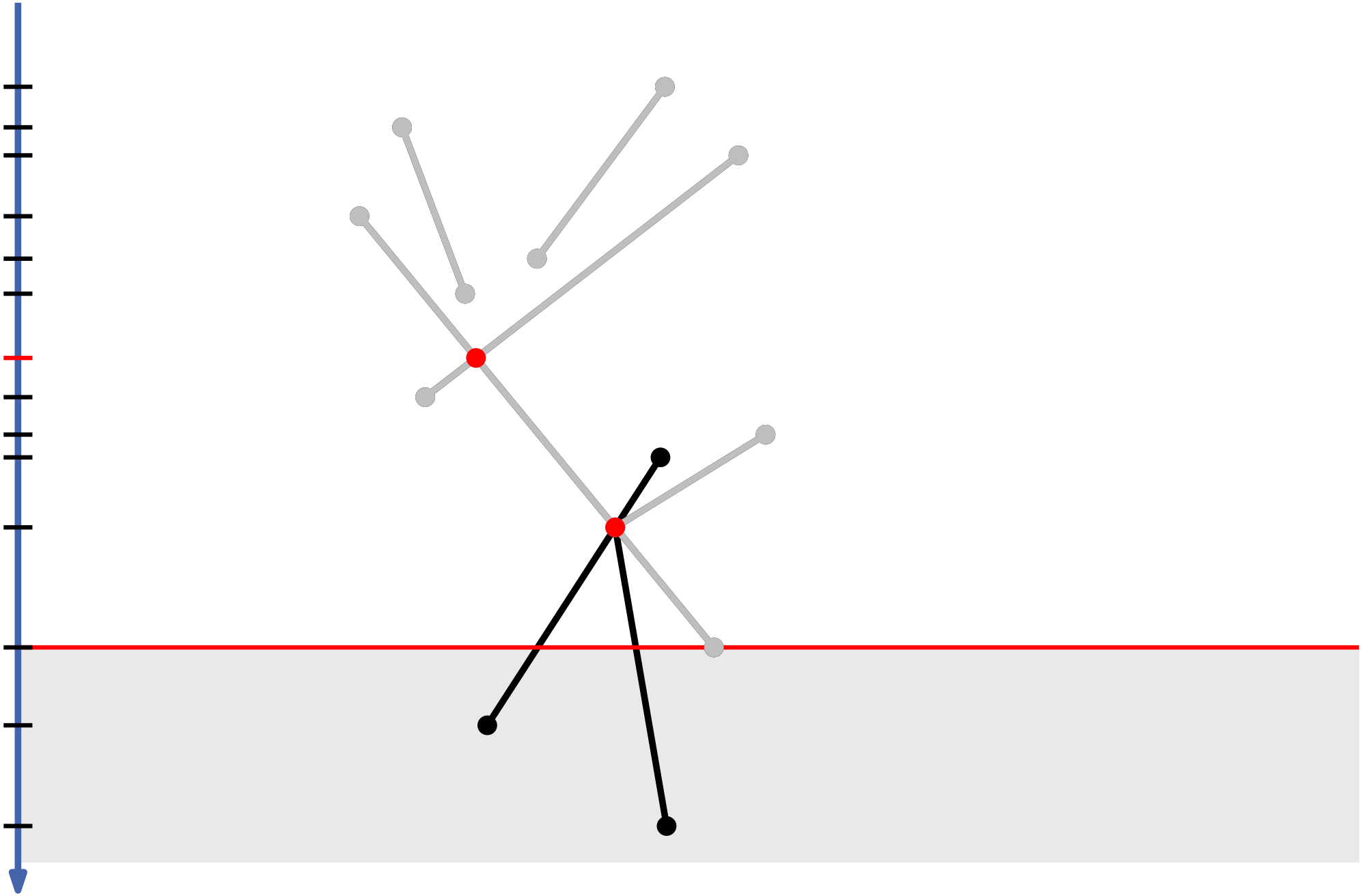
Das Sweep-Line Verfahren: Beispiel



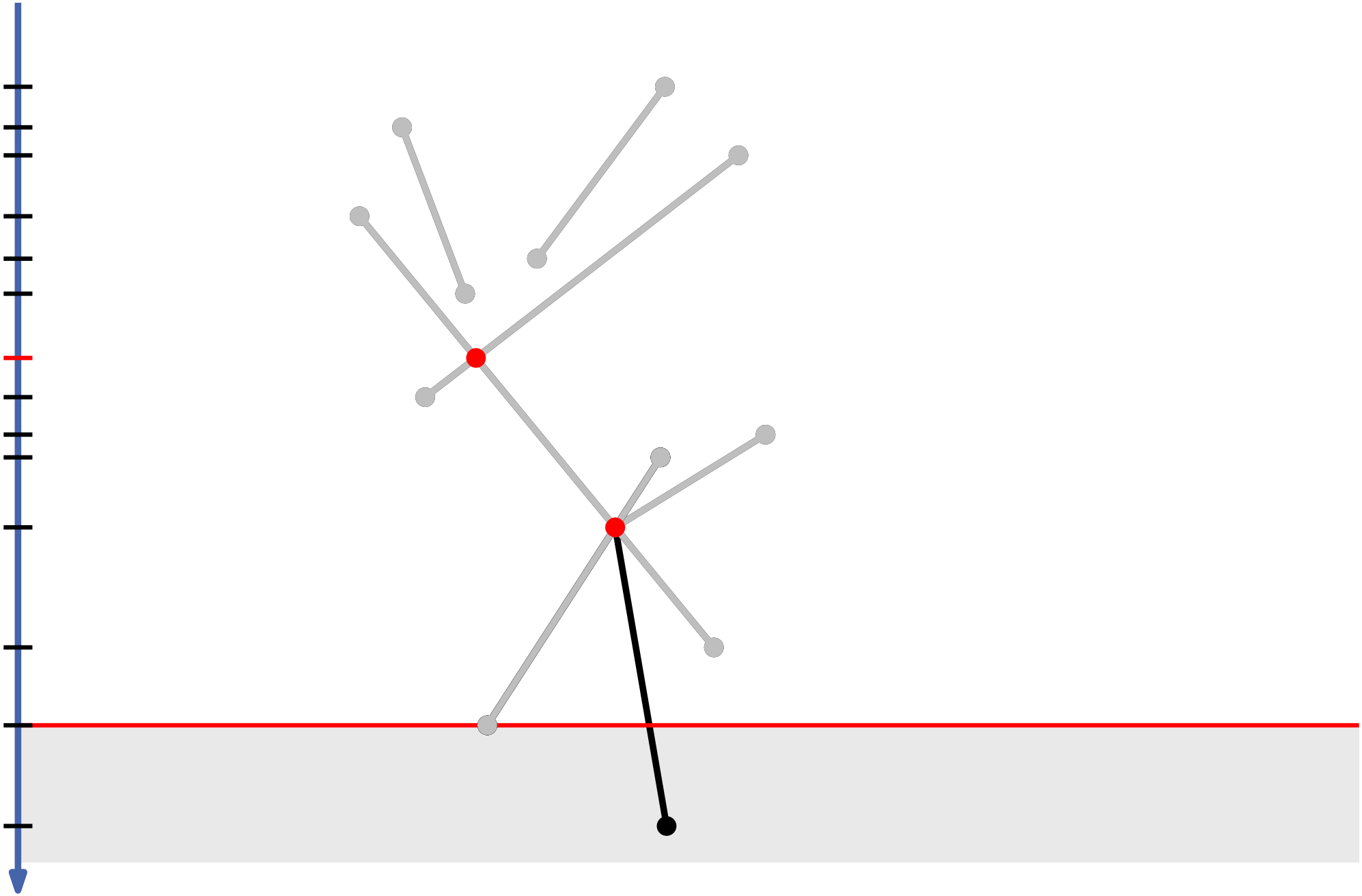
Das Sweep-Line Verfahren: Beispiel



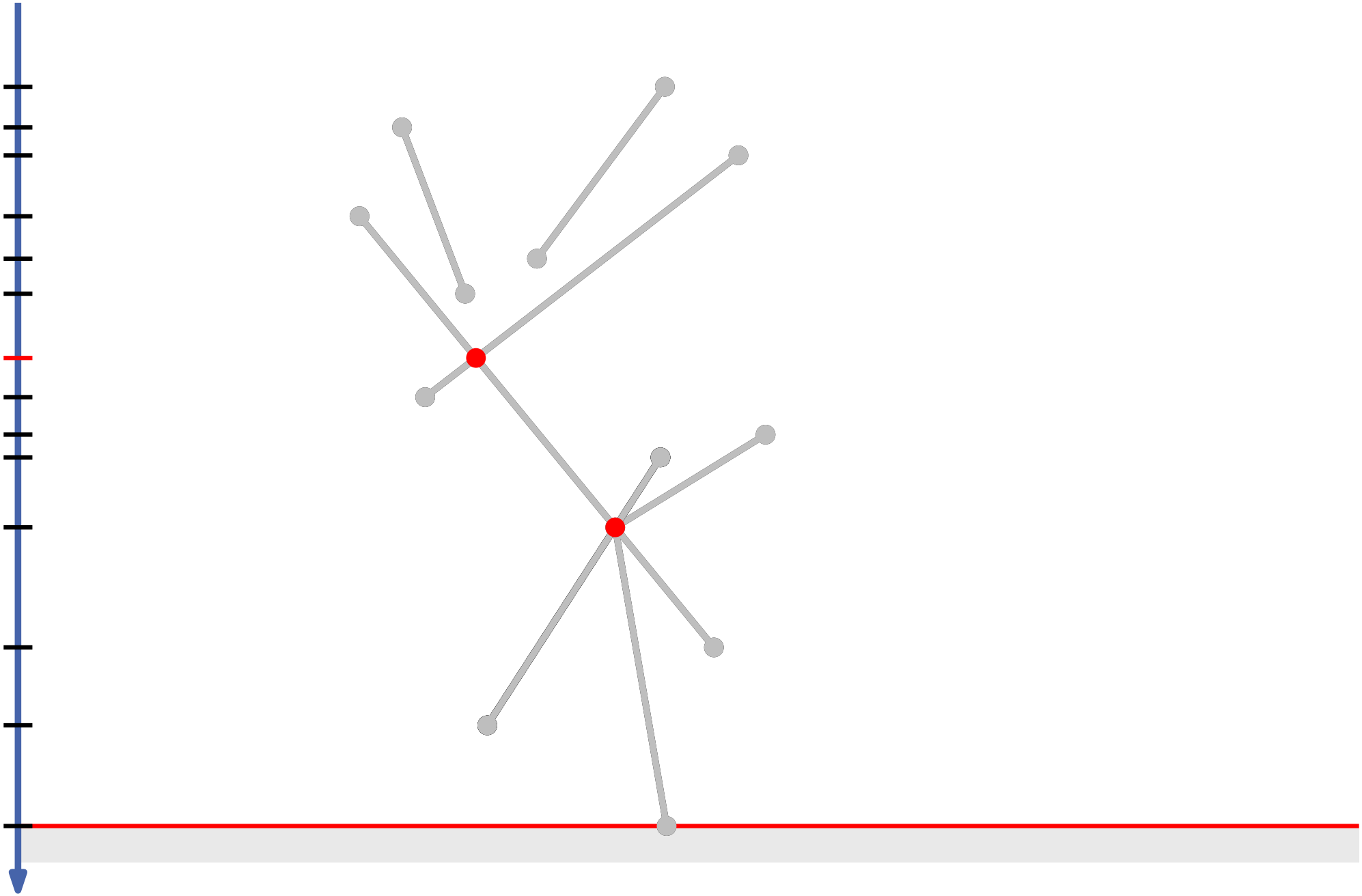
Das Sweep-Line Verfahren: Beispiel



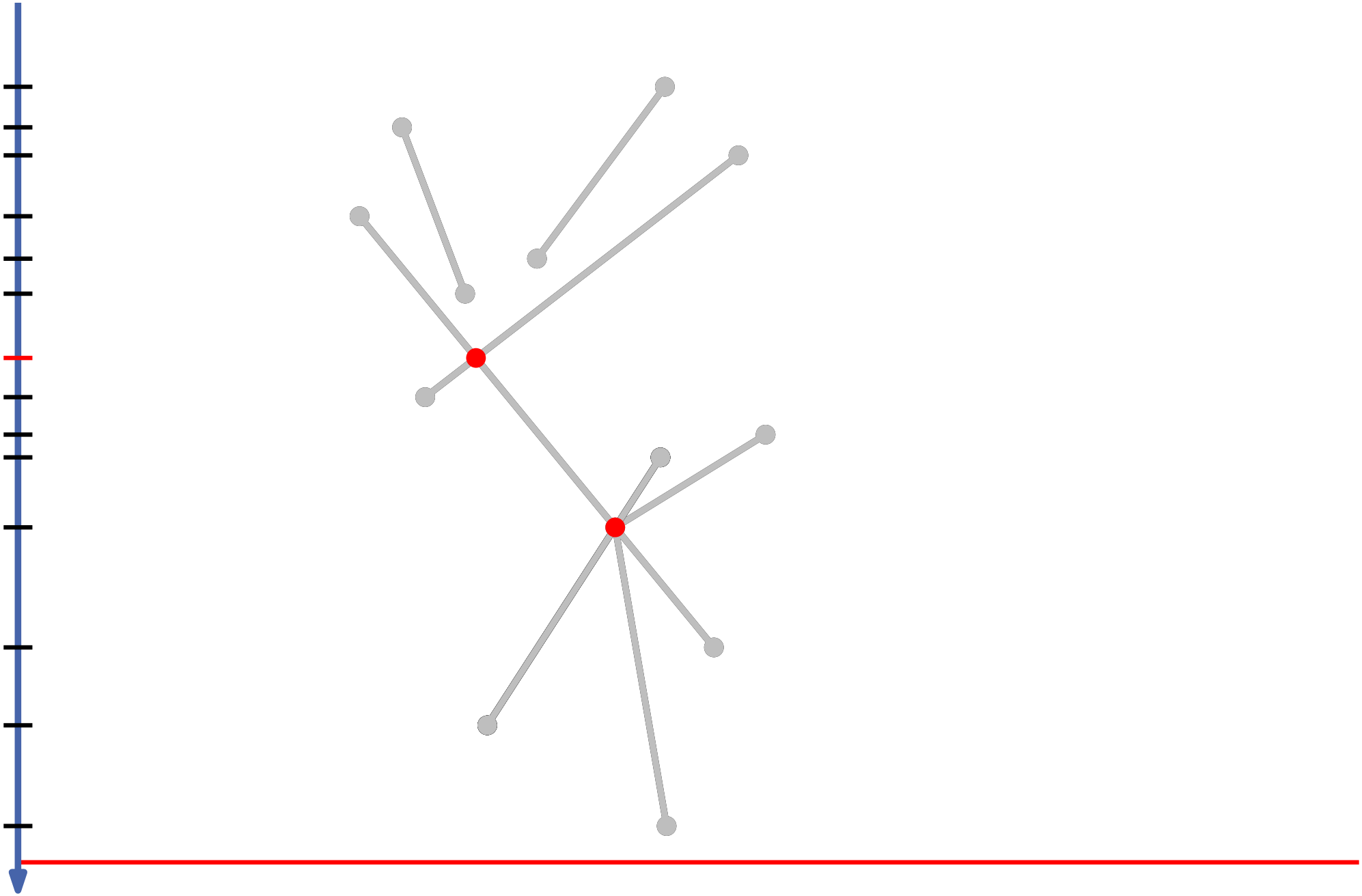
Das Sweep-Line Verfahren: Beispiel



Das Sweep-Line Verfahren: Beispiel

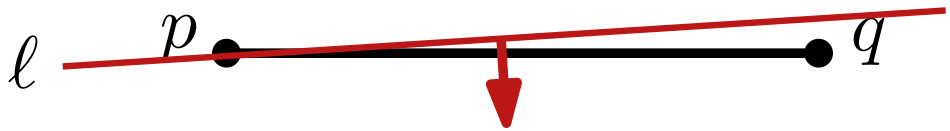


Das Sweep-Line Verfahren: Beispiel



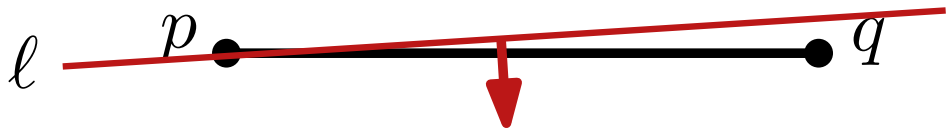
1.) Event Queue $\mathcal{Q}$

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



1.) Event Queue $\mathcal{Q}$

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

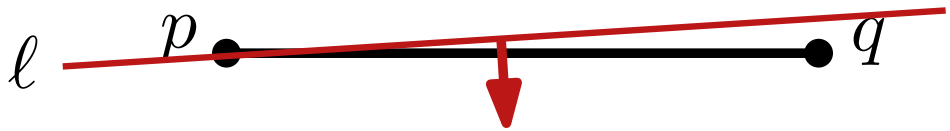


- speichere Events sortiert nach $\prec$ in **balanciertem binärem Suchbaum**

  $\rightarrow$ AVL-Baum, Rot-Schwarz-Baum, ...

1.) Event Queue $\mathcal{Q}$

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

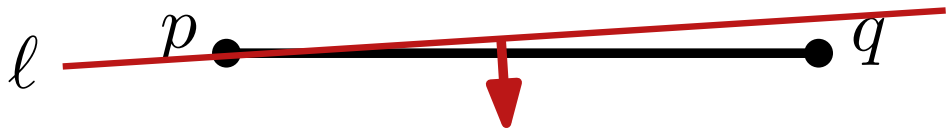


- speichere Events sortiert nach $\prec$ in **balanciertem binärem Suchbaum**
  $\rightarrow$ AVL-Baum, Rot-Schwarz-Baum, ...
- Operationen `insert`, `delete` und `nextEvent` in $O(\log |\mathcal{Q}|)$ Zeit

Datenstrukturen

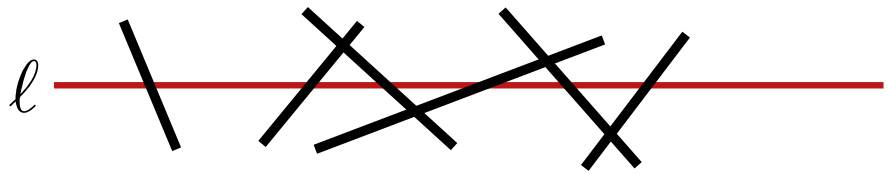
1.) Event Queue $\mathcal{Q}$

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



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

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

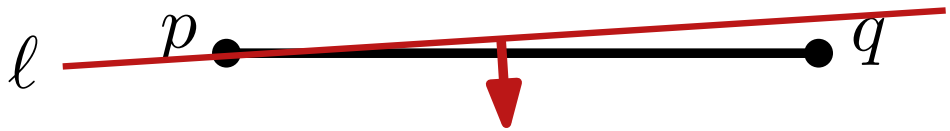


- speichere von ℓ geschnittene Strecken geordnet von links nach rechts

Datenstrukturen

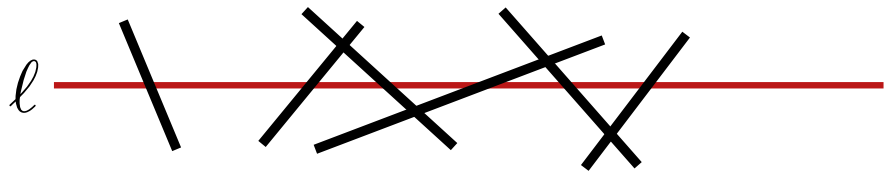
1.) Event Queue $\mathcal{Q}$

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



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

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

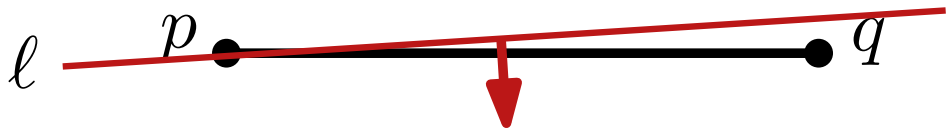


- speichere von ℓ geschnittene Strecken geordnet von links nach rechts
- benötigte Operationen `insert`, `delete`, `findNeighbor`

Datenstrukturen

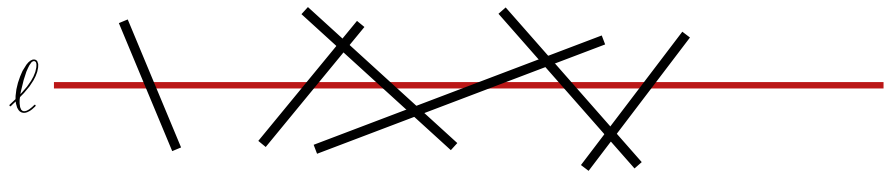
1.) Event Queue $\mathcal{Q}$

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



- speichere Events sortiert nach $\prec$ in **balanciertem binärem Suchbaum**
  $\rightarrow$ 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
- benötigte Operationen `insert`, `delete`, `findNeighbor`
- ebenfalls balancierter binärer Suchbaum mit Strecken in den Blättern!

Algorithmus

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

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

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

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

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

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

    $p \leftarrow \mathcal{Q}$.nextEvent()

    $\mathcal{Q}$.deleteEvent(p)

    handleEvent(p)

Algorithmus

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

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

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

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

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

Was passiert mit Duplikaten?

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

    $p \leftarrow \mathcal{Q}$.nextEvent()

    $\mathcal{Q}$.deleteEvent(p)

    handleEvent(p)

Algorithmus

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

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

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

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

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

Was passiert mit Duplikaten?

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

    $p \leftarrow \mathcal{Q}$.nextEvent()

    $\mathcal{Q}$.deleteEvent(p)

    handleEvent(p)

Hier versteckt sich der Kern des Algorithmus!

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 $\mathcal{T}$
  $\mathcal{Q} \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p
else //s' und s'' linkeste und rechteste Strecke in $U(p) \cup C(p)$
  $\mathcal{Q} \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p
  $\mathcal{Q} \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

Algorithmus

handleEvent(p)

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

mit p in $\mathcal{Q}$ gespeichert

$L(p) \leftarrow$ 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 $\mathcal{T}$

  $\mathcal{Q} \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p

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

  $\mathcal{Q} \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p

  $\mathcal{Q} \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

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

mit p in $\mathcal{Q}$ gespeichert

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 $\mathcal{T}$

  $\mathcal{Q} \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p

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

  $\mathcal{Q} \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p

  $\mathcal{Q} \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

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 $\mathcal{T}$

  $\mathcal{Q} \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p

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

  $\mathcal{Q} \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p

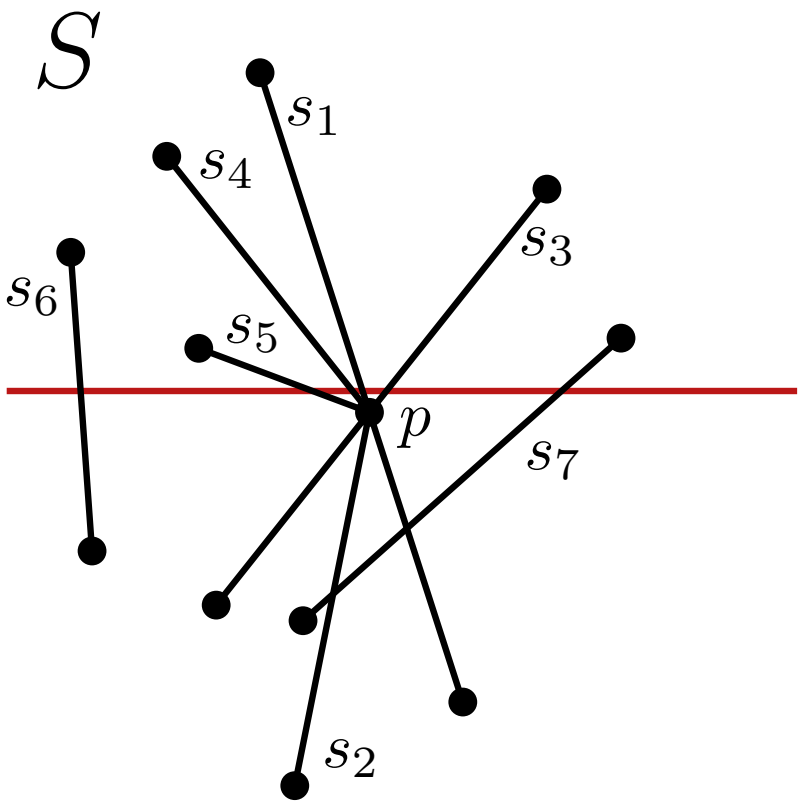
  $\mathcal{Q} \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

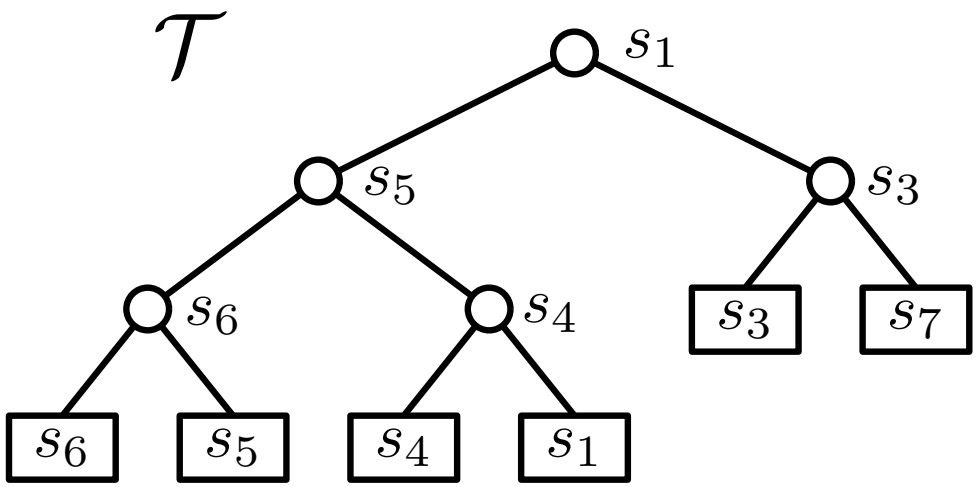
mit p in $\mathcal{Q}$ gespeichert

Nachbarn in $\mathcal{T}$

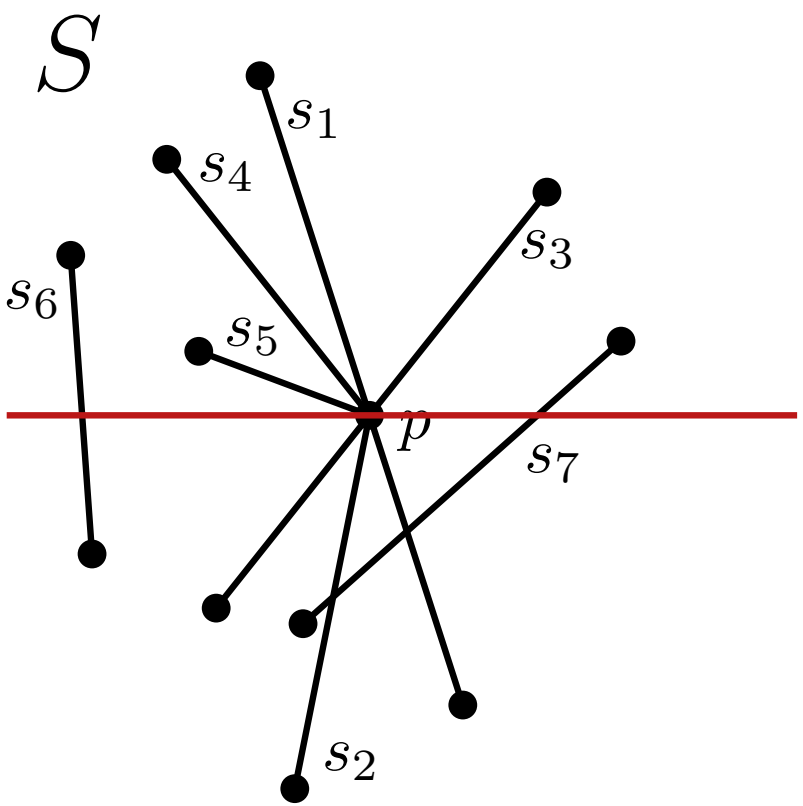
Entfernen und Einfügen dreht $C(p)$ um

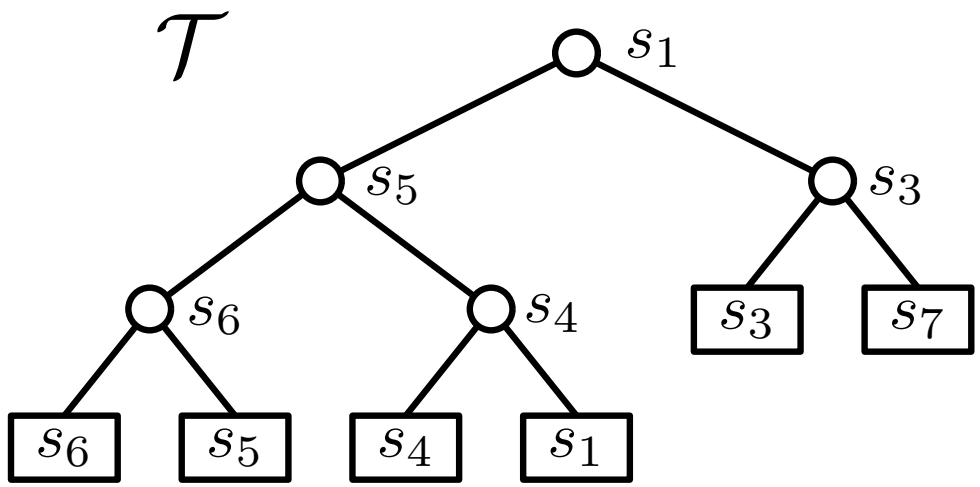
Was passiert genau?





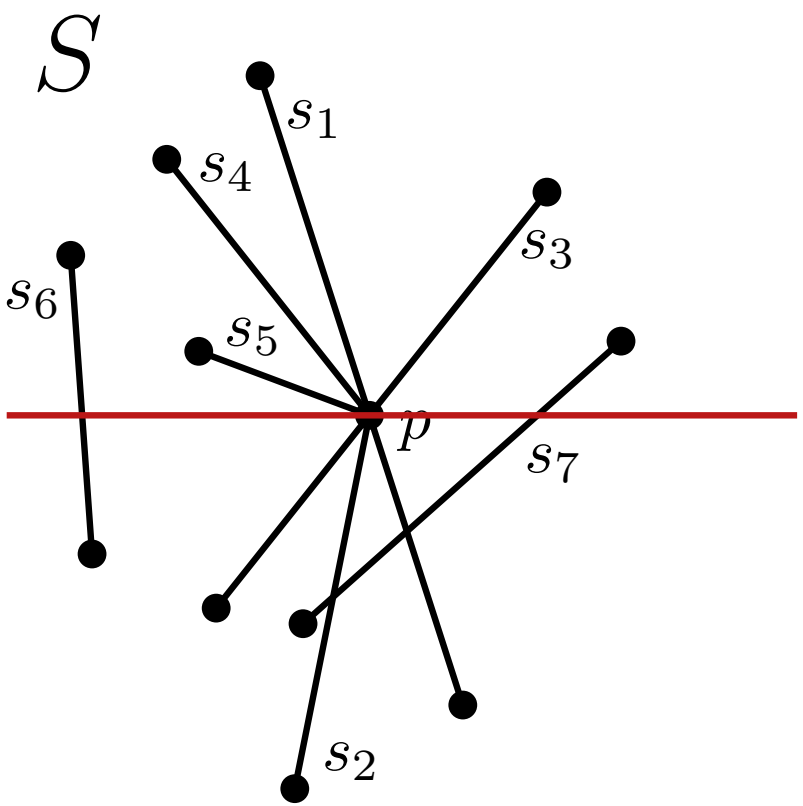
Was passiert genau?

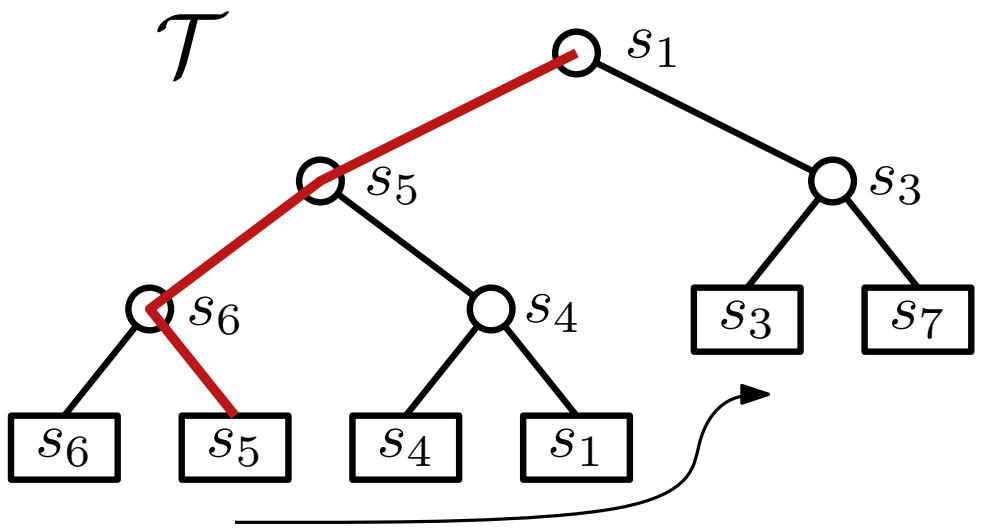




$U(p) = \{s_2\}$
$L(p) =$
$C(p) =$

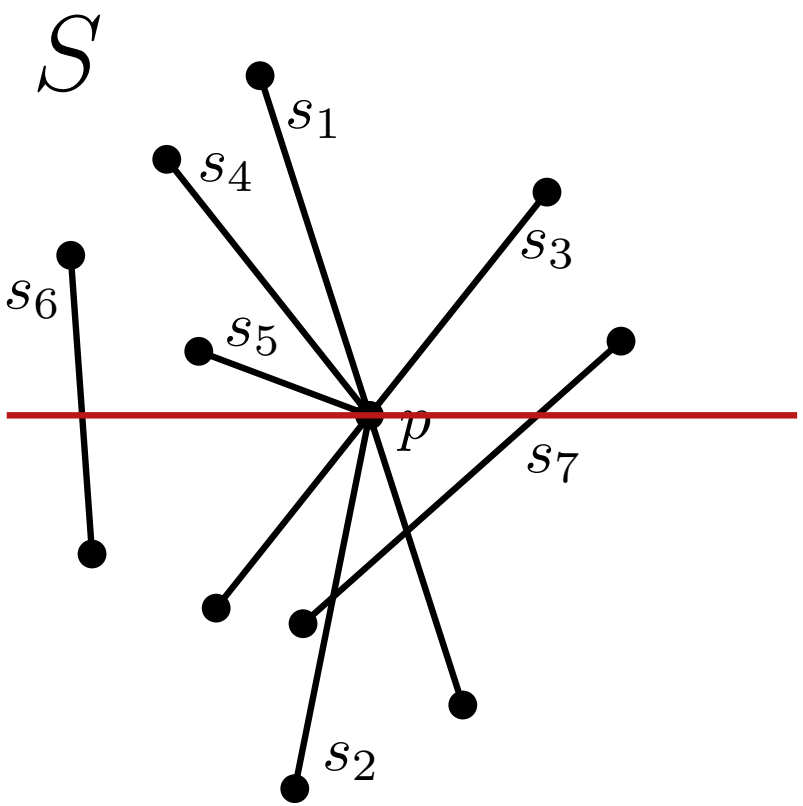
Was passiert genau?

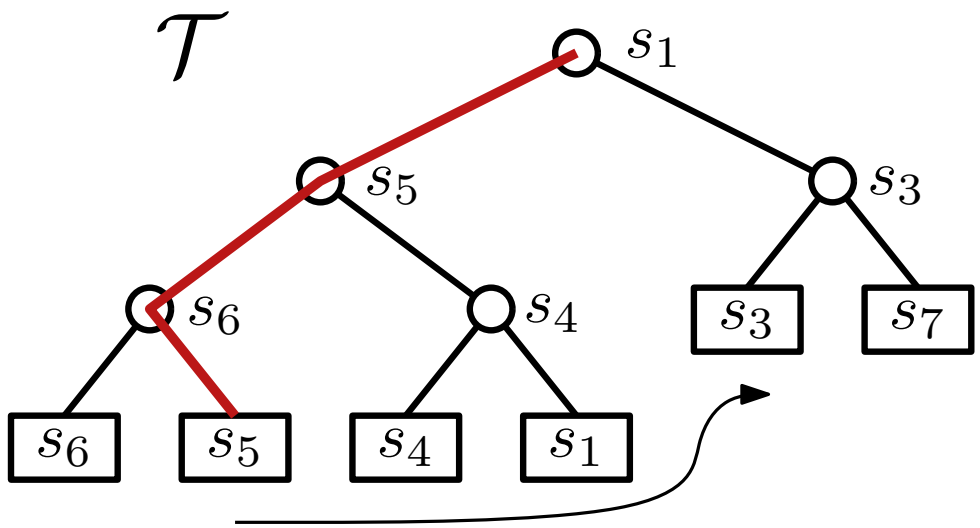




$U(p) = \{s_2\}$
$L(p) =$
$C(p) =$

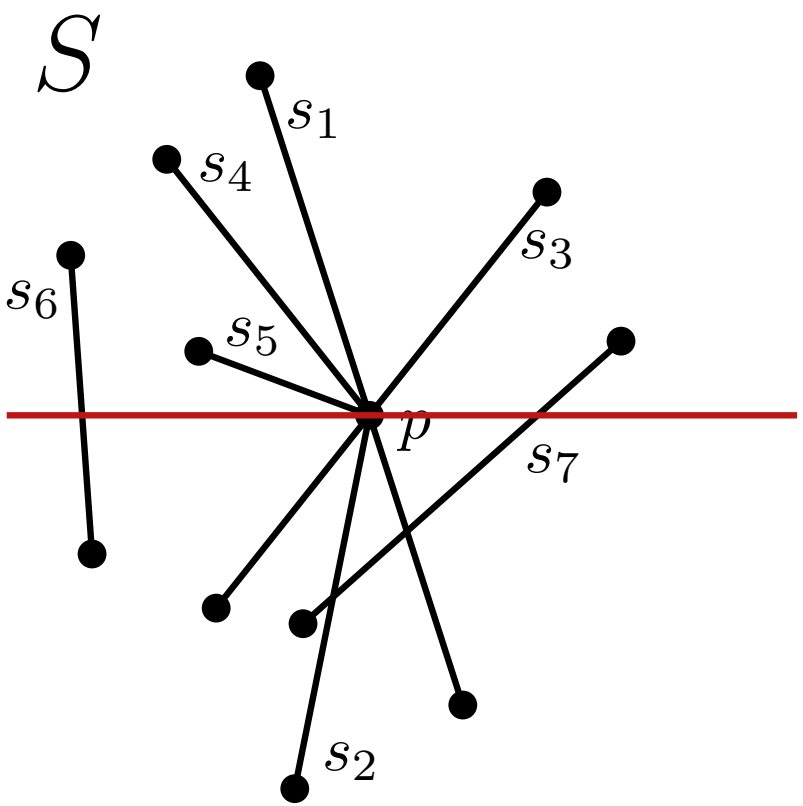
Was passiert genau?

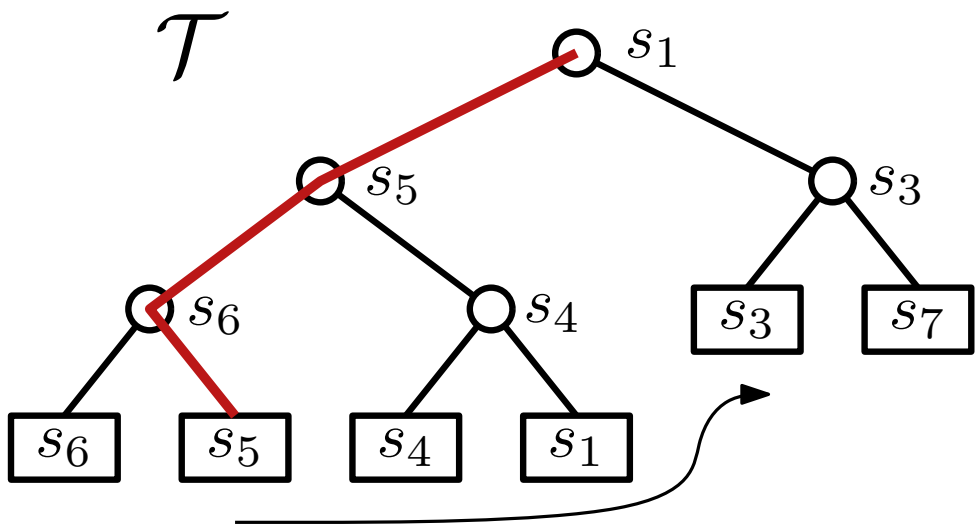




$$U(p) = \{s_2\}$$
$$L(p) = \{s_4, s_5\}$$
$$C(p) = \{s_1, s_3\}$$

Was passiert genau?

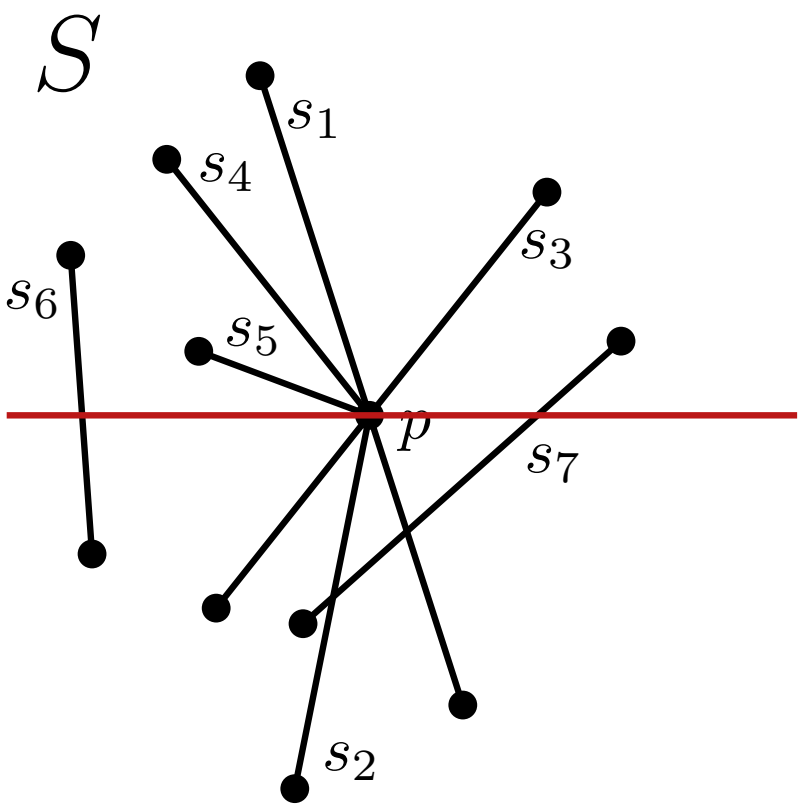


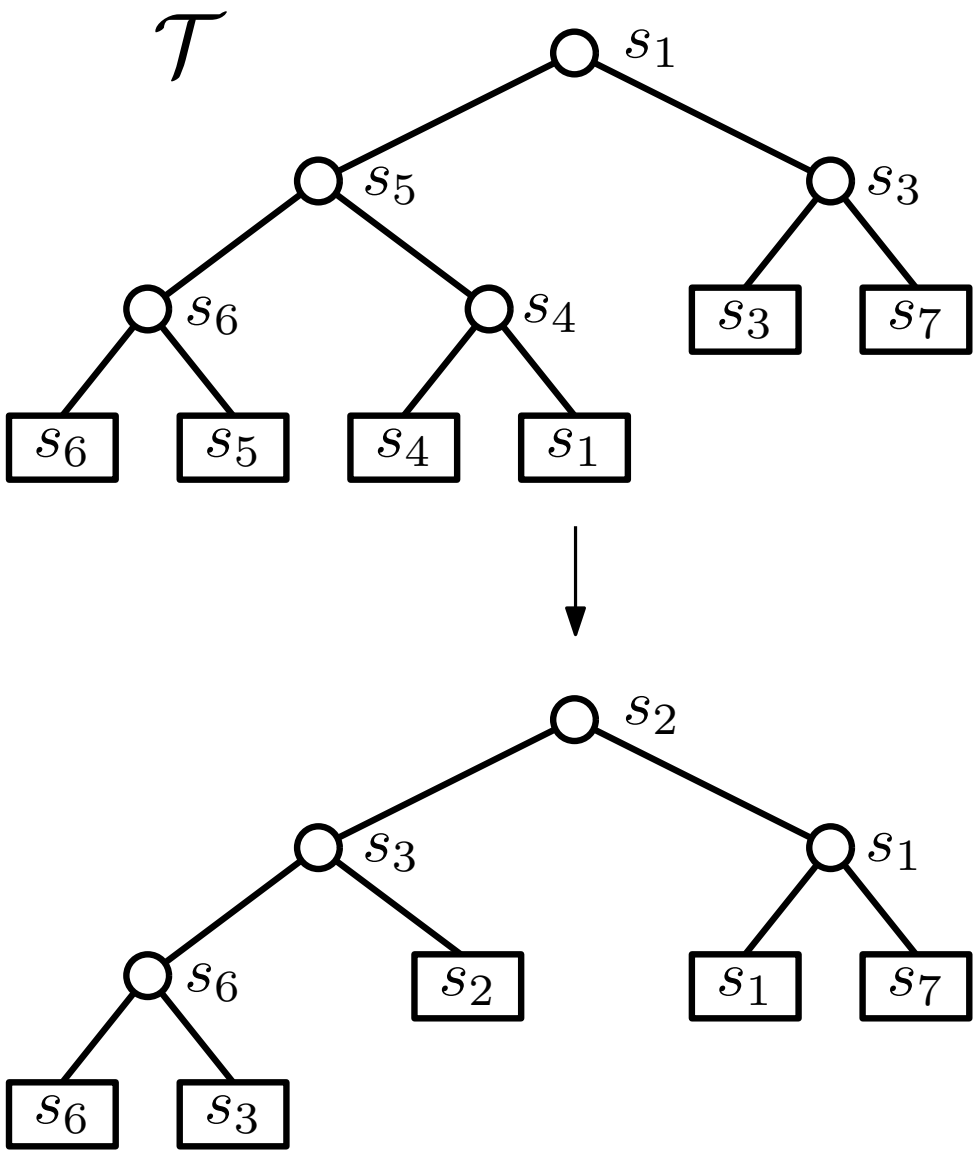


$U(p) = \{s_2\}$
$L(p) = \{s_4, s_5\}$
$C(p) = \{s_1, s_3\}$

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

Was passiert genau?

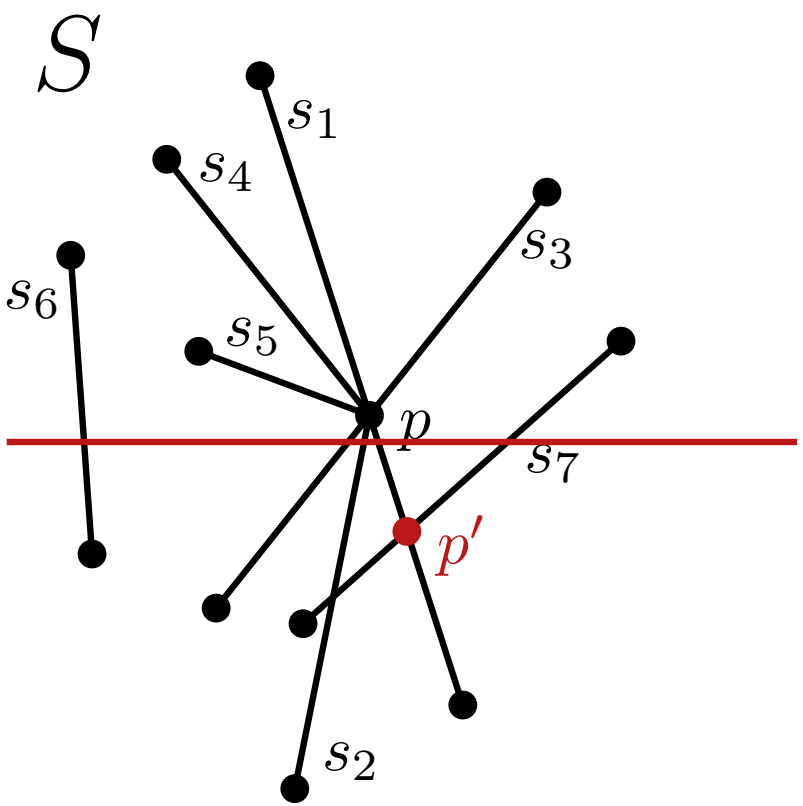


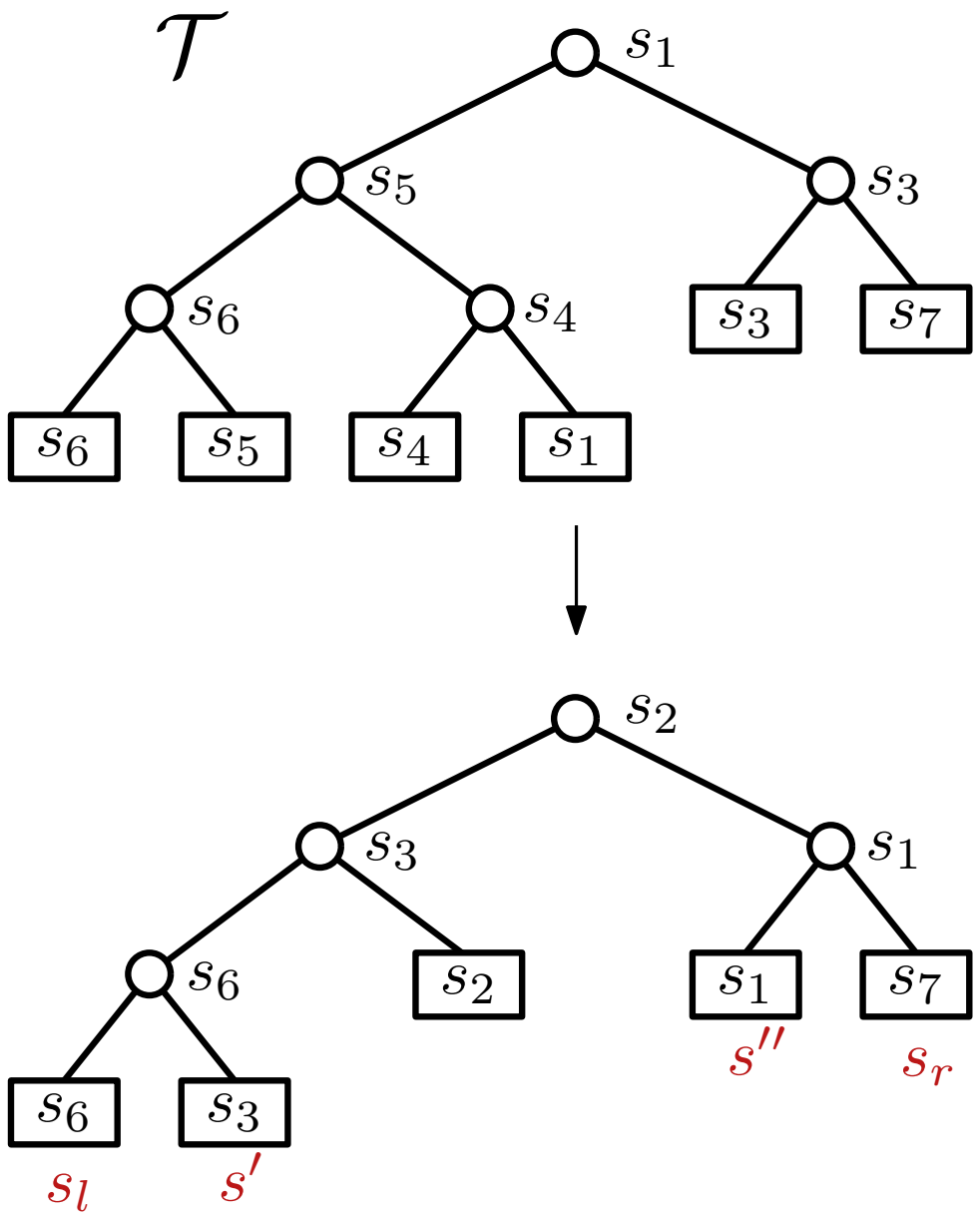


$$U(p) = \{s_2\}$$
$$L(p) = \{s_4, s_5\}$$
$$C(p) = \{s_1, s_3\}$$

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 $\mathcal{Q}$ ein

Korrektheit

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

Korrektheit

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

Korrektheit

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

Fall 2: p ist kein Streckenendpunkt

Überlege warum p in $\mathcal{Q}$ sein muss!

Laufzeitanalyse

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

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

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

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

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

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

  $p \leftarrow \mathcal{Q}$.nextEvent()

  $\mathcal{Q}$.deleteEvent(p)

  handleEvent(p)

Laufzeitanalyse

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

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

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

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

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

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

  $p \leftarrow \mathcal{Q}$.nextEvent()

  $\mathcal{Q}$.deleteEvent(p)

  handleEvent(p)

Laufzeitanalyse

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

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

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

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

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

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

  $p \leftarrow \mathcal{Q}$.nextEvent()

  $\mathcal{Q}$.deleteEvent(p)

  handleEvent(p)

Laufzeitanalyse

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

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

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

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

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

$O(n \log n)$

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

  $p \leftarrow \mathcal{Q}$.nextEvent()

  $\mathcal{Q}$.deleteEvent(p)

$O(\log |\mathcal{Q}|)$

  handleEvent(p)

Laufzeitanalyse

FindIntersections(S)

Input: Menge S von Strecken

Output: Menge aller Schnittpunkte mit zugeh. Strecken

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

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

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

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

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

  $p \leftarrow \mathcal{Q}$.nextEvent()

  $\mathcal{Q}$.deleteEvent(p) $O(\log |\mathcal{Q}|)$

  handleEvent(p) ?

Laufzeitanalyse

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 $\mathcal{T}$
  $\mathcal{Q} \leftarrow$ prüfe s_l und s_r auf Schnitt unterhalb p
else //s' und s'' linkeste und rechteste Strecke in $U(p) \cup C(p)$
  $\mathcal{Q} \leftarrow$ prüfe s_l und s' auf Schnitt unterhalb p
  $\mathcal{Q} \leftarrow$ prüfe s_r und s'' auf Schnitt unterhalb p

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

Zusammenfassung

Satz: 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.

Zusammenfassung

Satz: 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

Zusammenfassung

Satz: 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

Überlege wie viel Platz die Datenstrukturen benötigen!

Zusammenfassung

Satz: 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

Überlege 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

Diskussion

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

Diskussion

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

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

Diskussion

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?

Diskussion

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].

Diskussion

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?

Diskussion

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. Buch)

Diskussion

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].

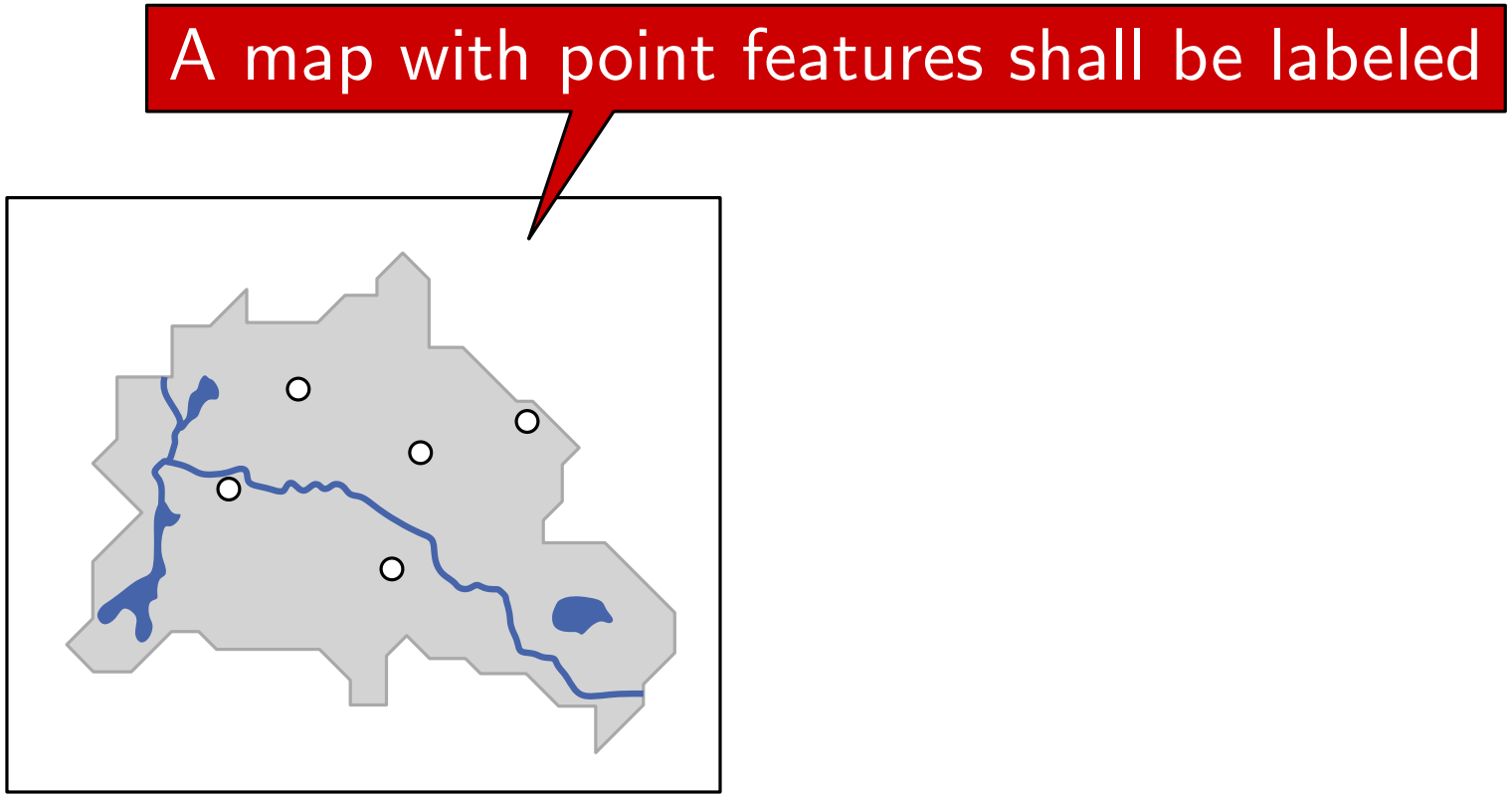
Wie löst man damit das Kartenüberlagerungs-Problem?

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. Buch)

Bonus: Streckenschnitte und dynamische Punktbeschriftung

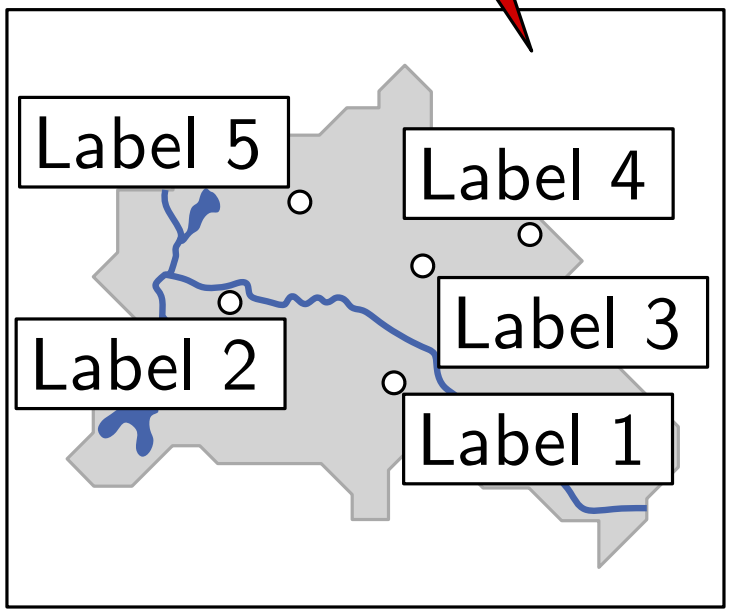
Dynamic One-Sided Boundary Labeling, N., Polishchuk, Sysikaski, 2010

Motivation

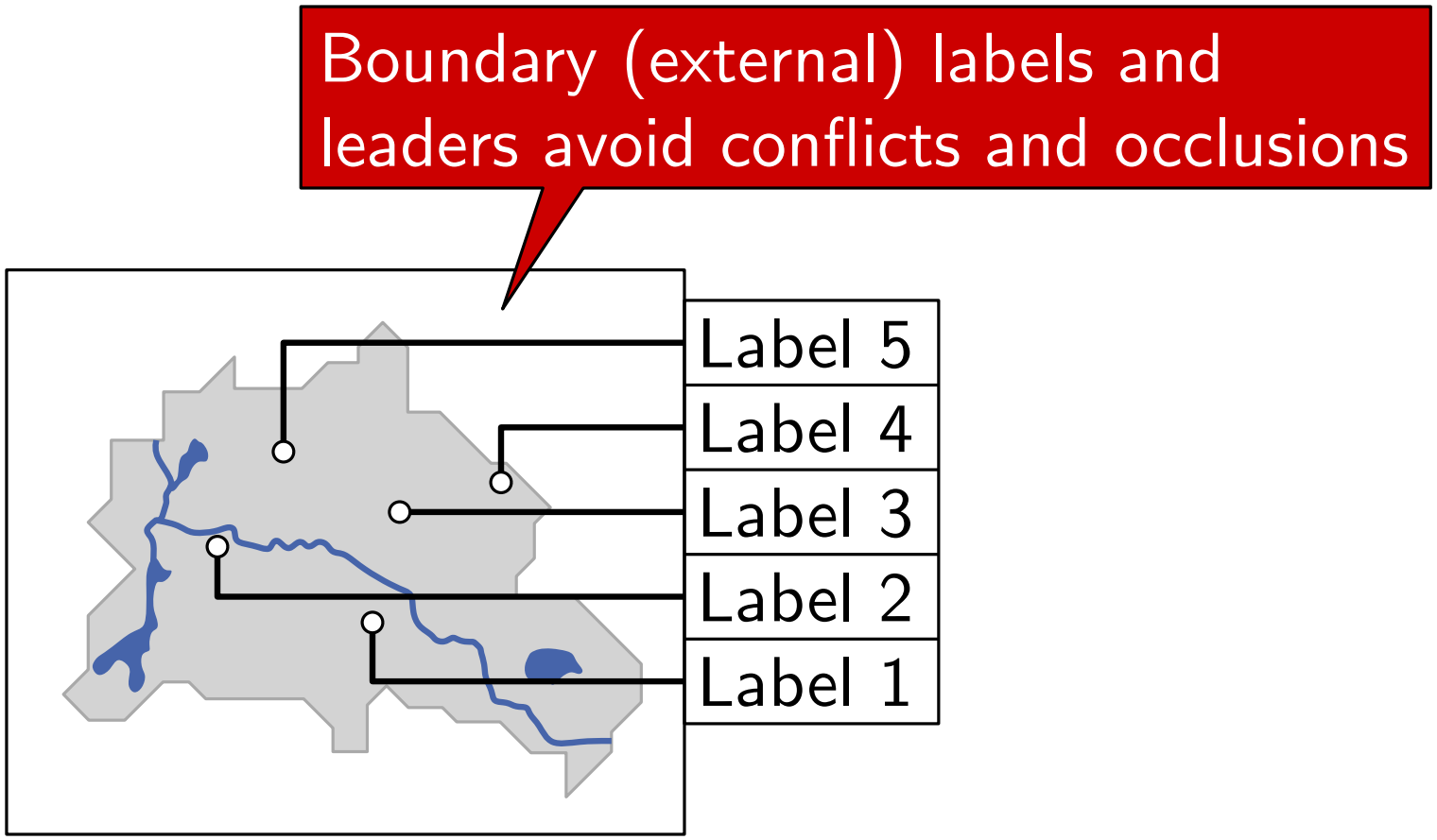


Motivation

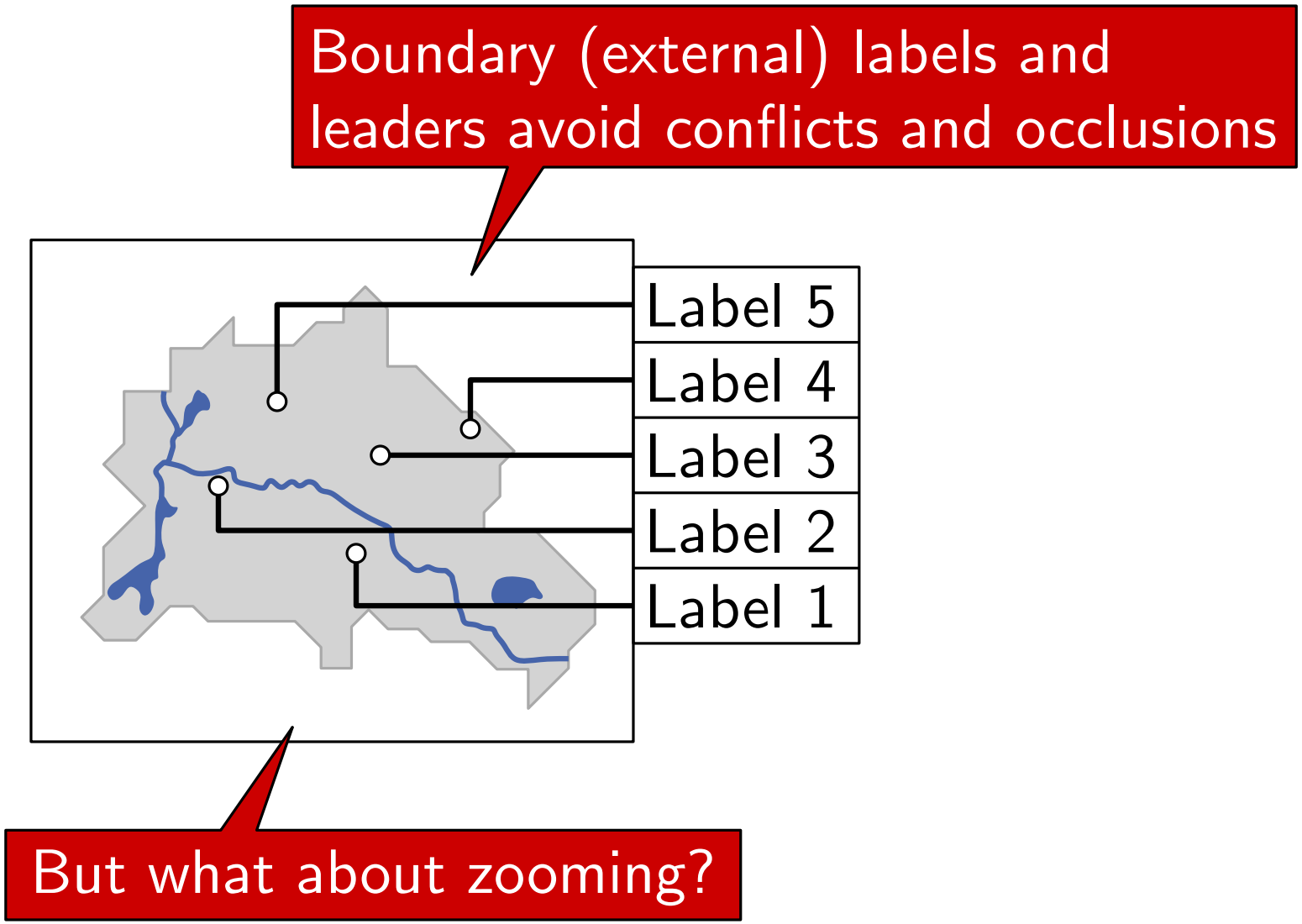
Internal labels may create conflicts and occlusions



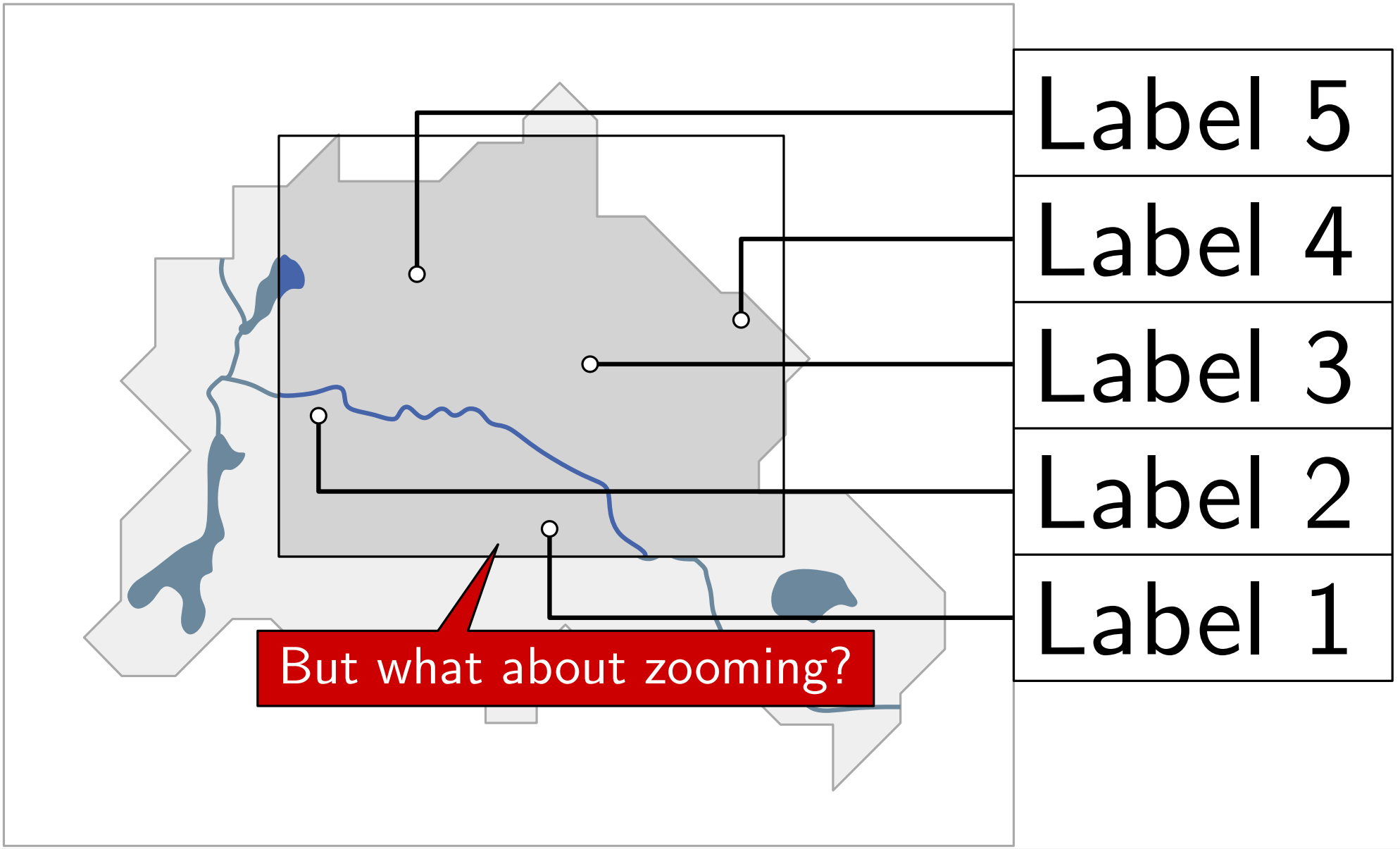
Motivation


Boundary (external) labels and leaders avoid conflicts and occlusions
Label 5
Label 4
Label 3
Label 2
Label 1

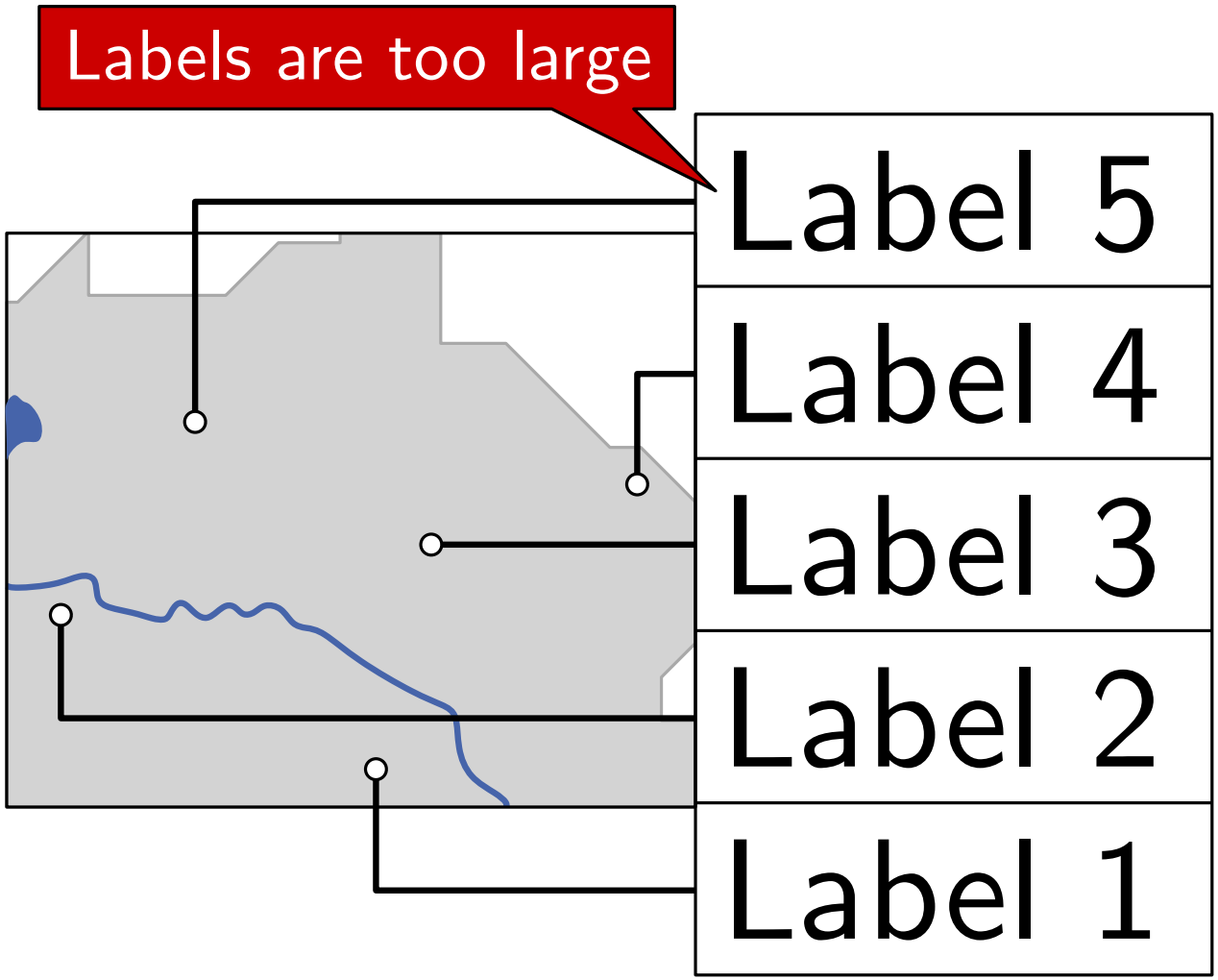
Motivation


Boundary (external) labels and leaders avoid conflicts and occlusions
Label 5
Label 4
Label 3
Label 2
Label 1
But what about zooming?

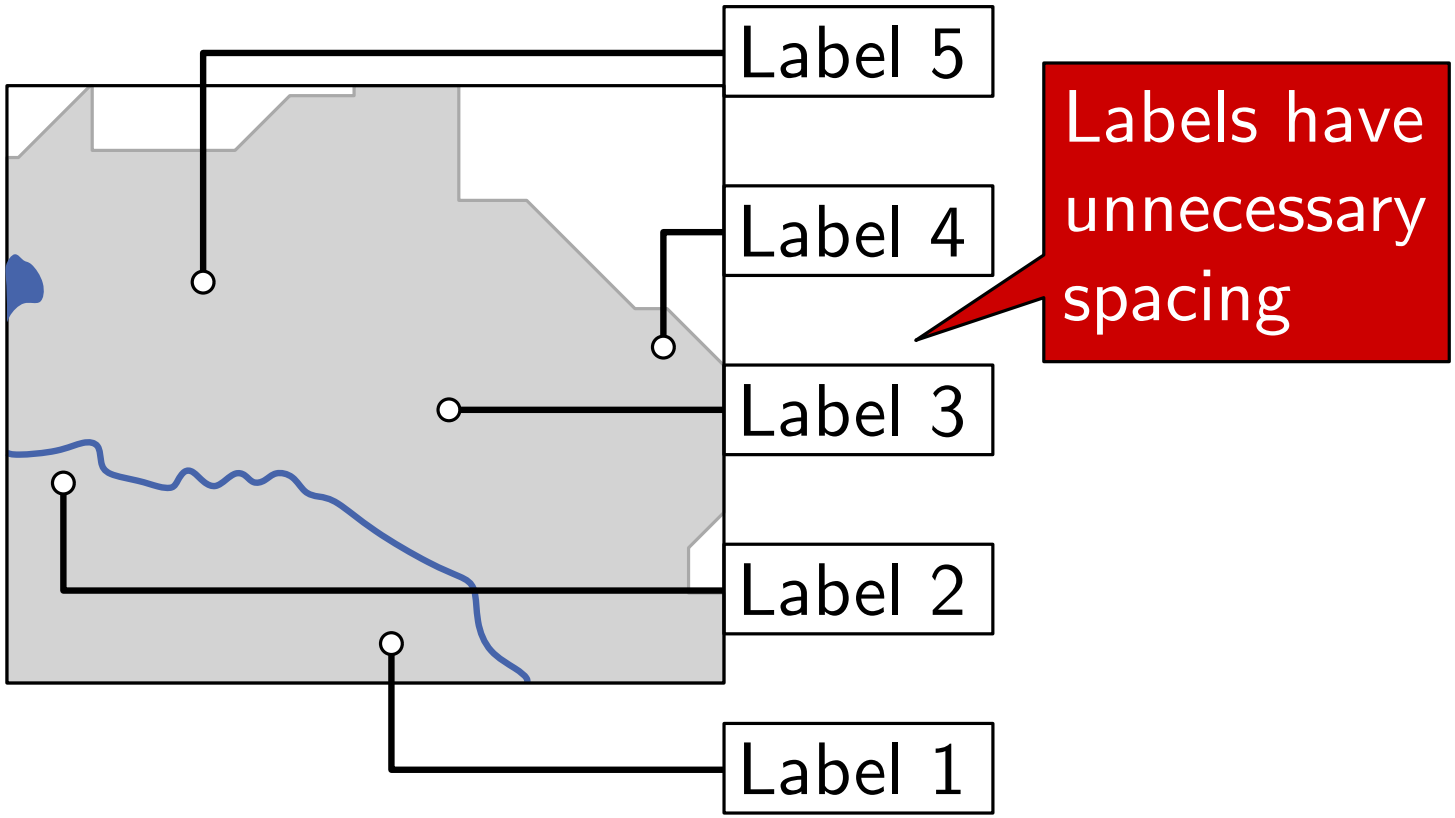
Motivation



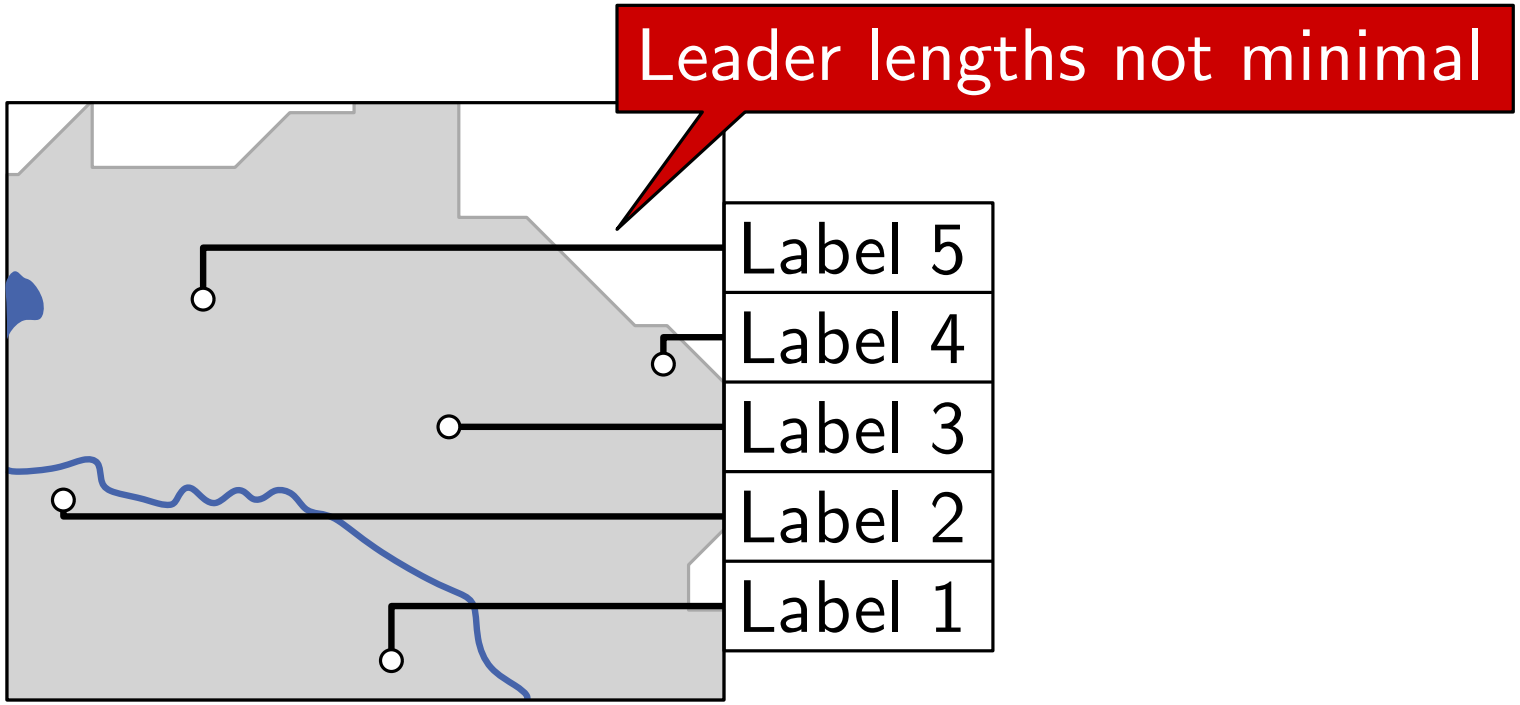
Motivation



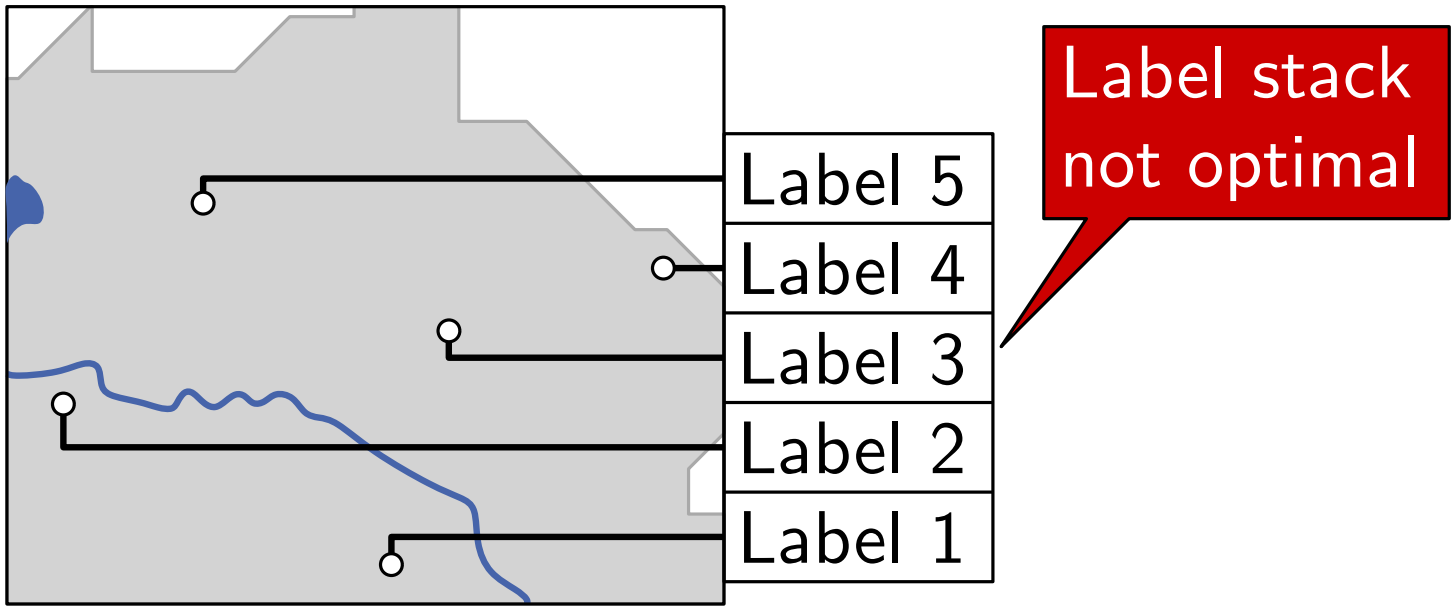
Motivation


Label 5
Label 4
Label 3
Label 2
Label 1
Labels have unnecessary spacing

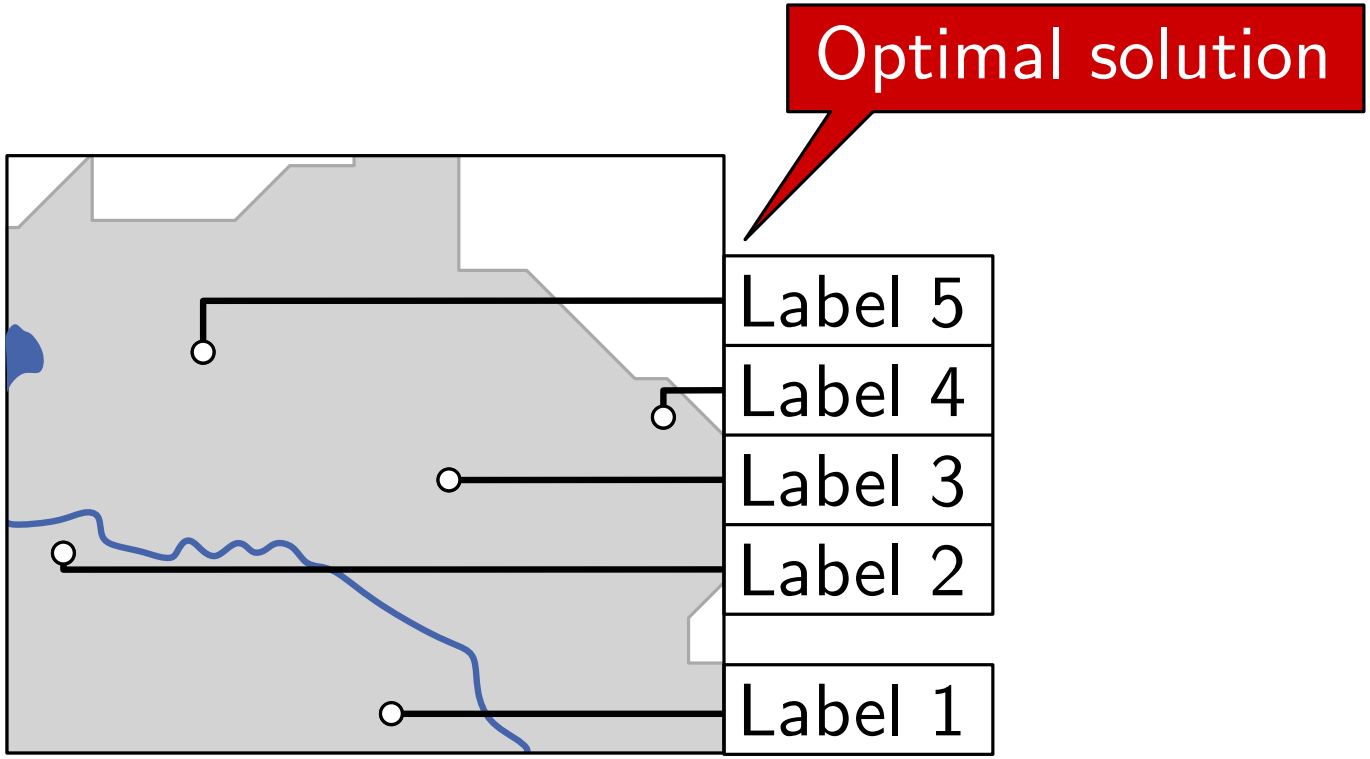
Motivation


Leader lengths not minimal
Label 5
Label 4
Label 3
Label 2
Label 1

Motivation


Label 5
Label 4
Label 3
Label 2
Label 1
Label stack not optimal

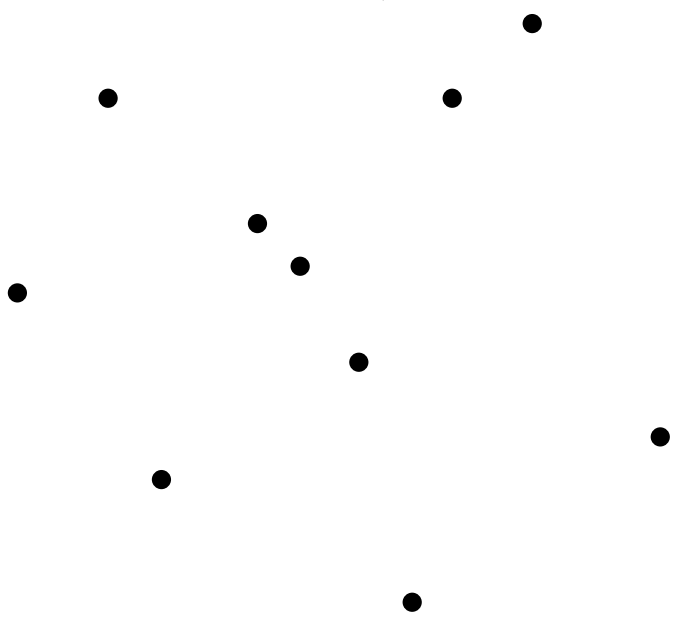
Motivation



Goal: dynamically maintain minimal leader lengths and optimal label stacks during zooming

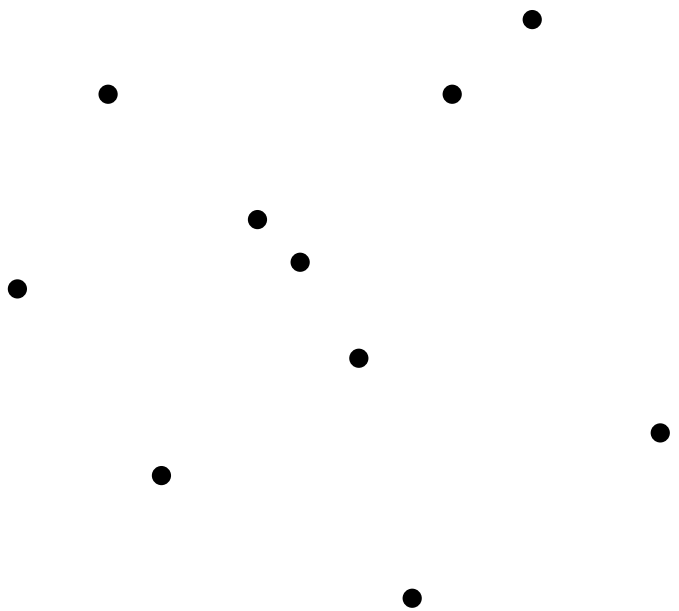
Model

Given a base map with n point sites in the plane



Model

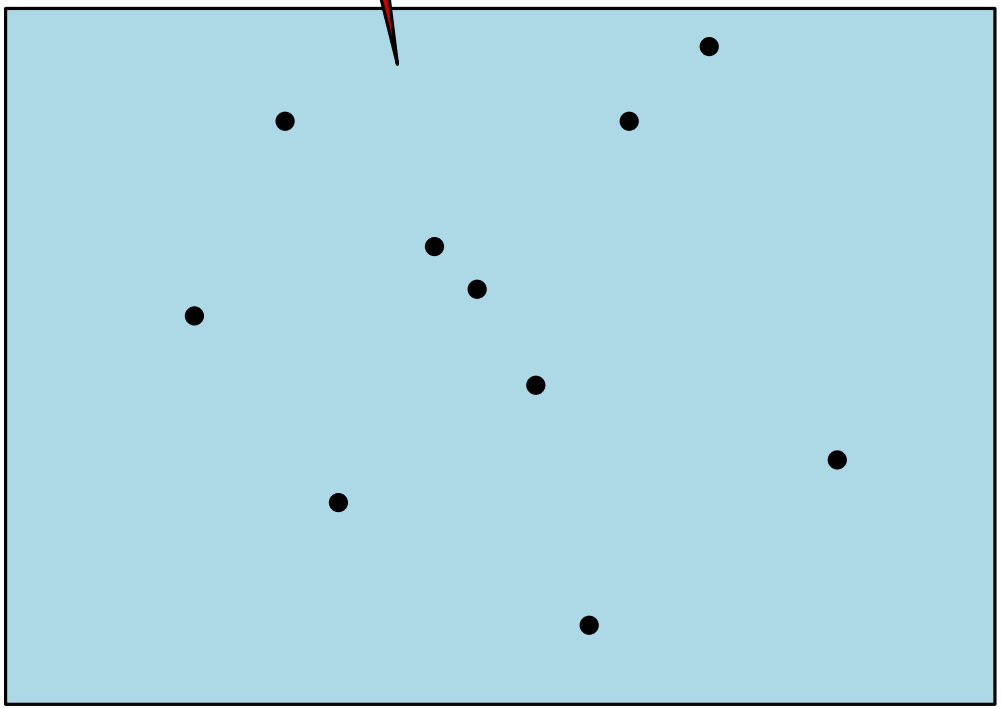
Given a base map with n point sites in the plane



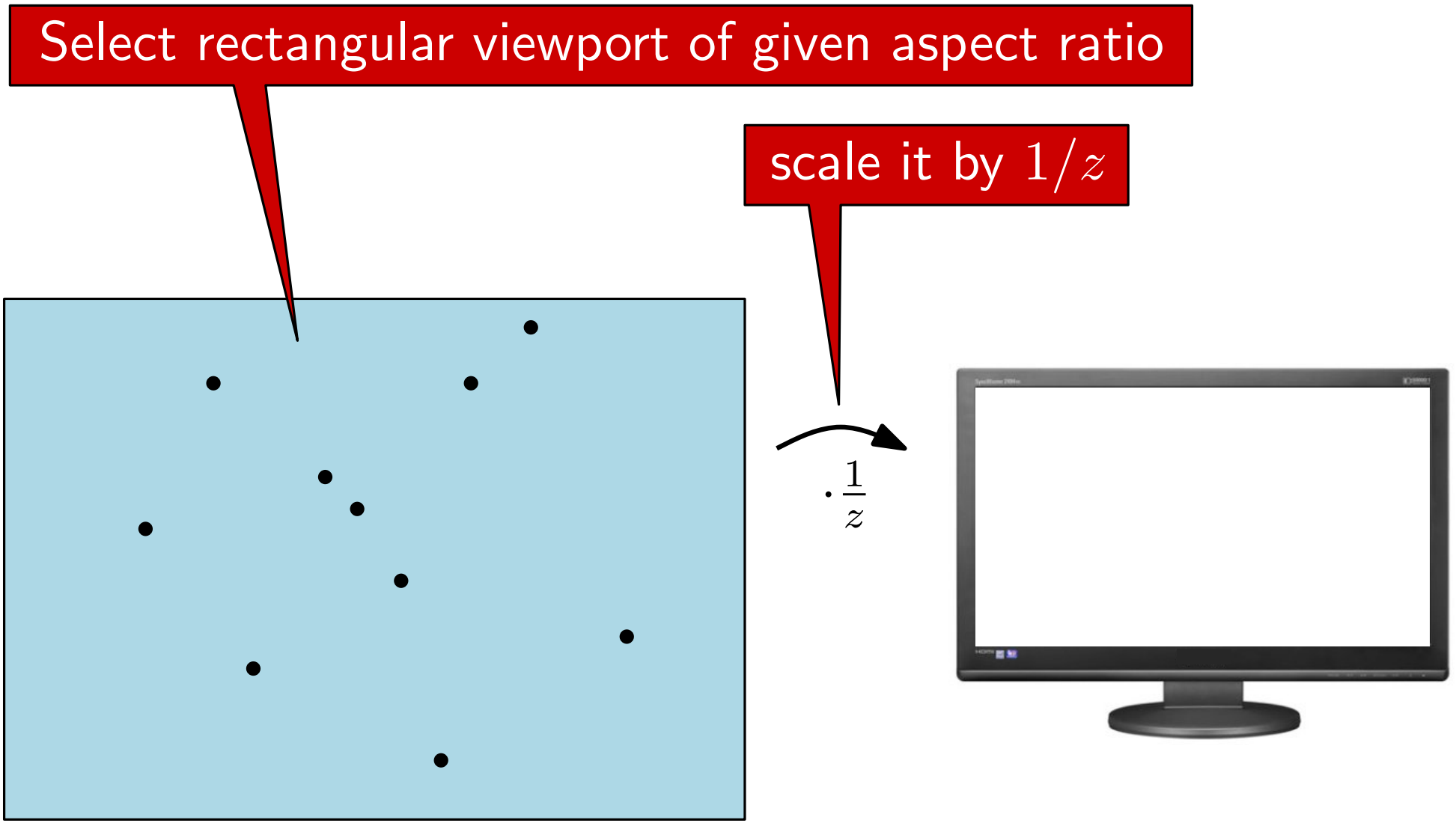
and a screen of fixed aspect ratio

Model

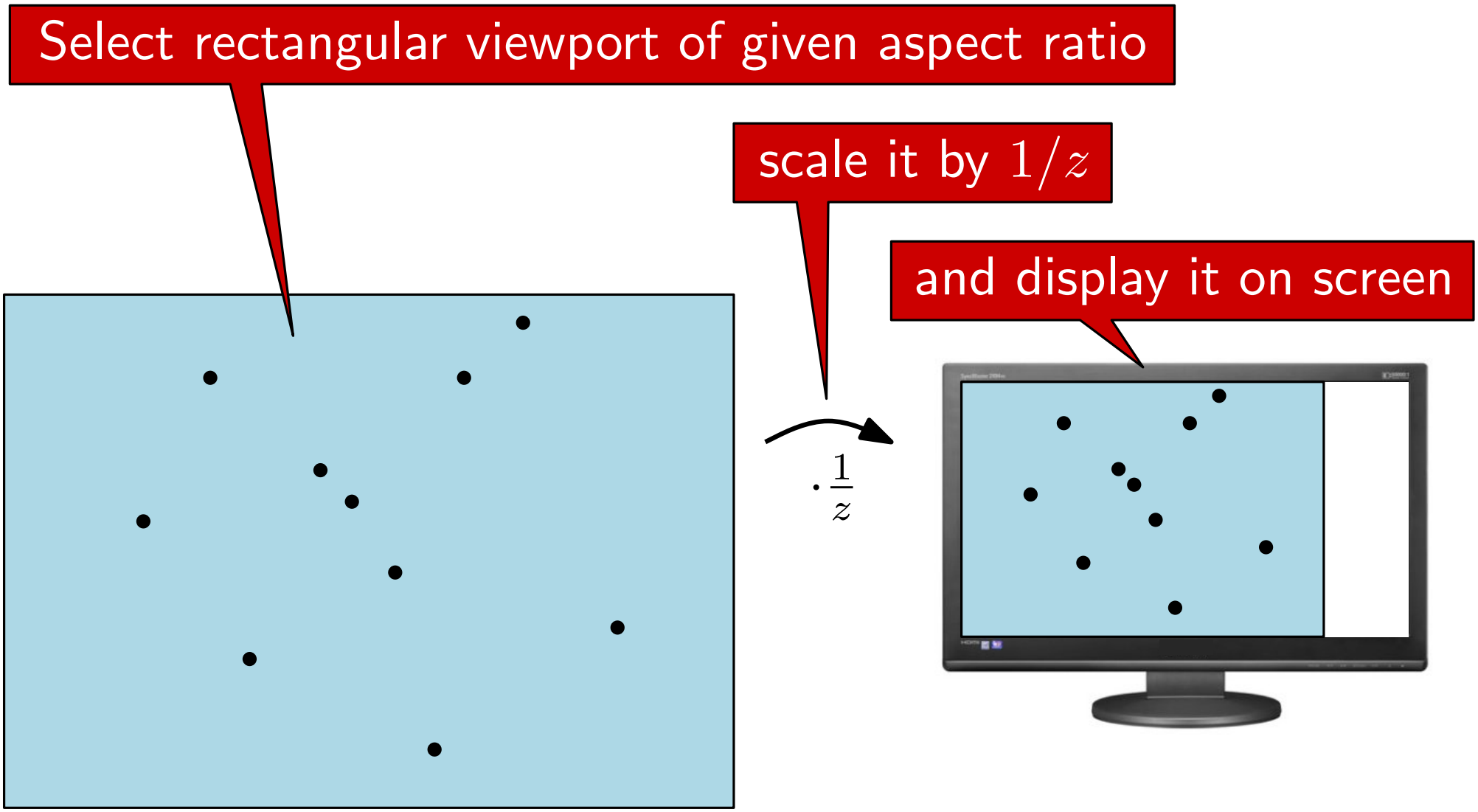
Select rectangular viewport of given aspect ratio



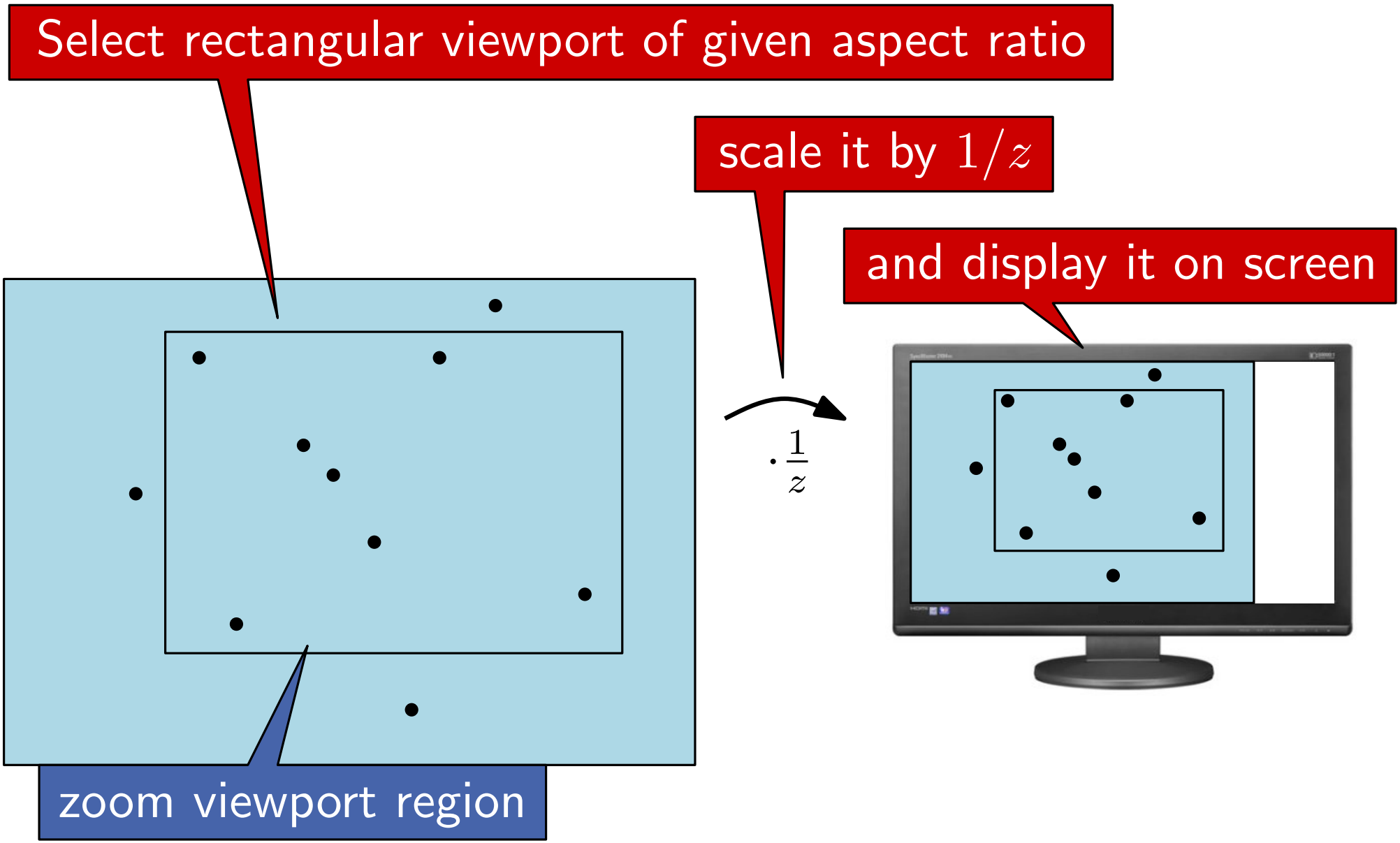
Model


Select rectangular viewport of given aspect ratio
scale it by $1/z$
$\cdot \frac{1}{z}$

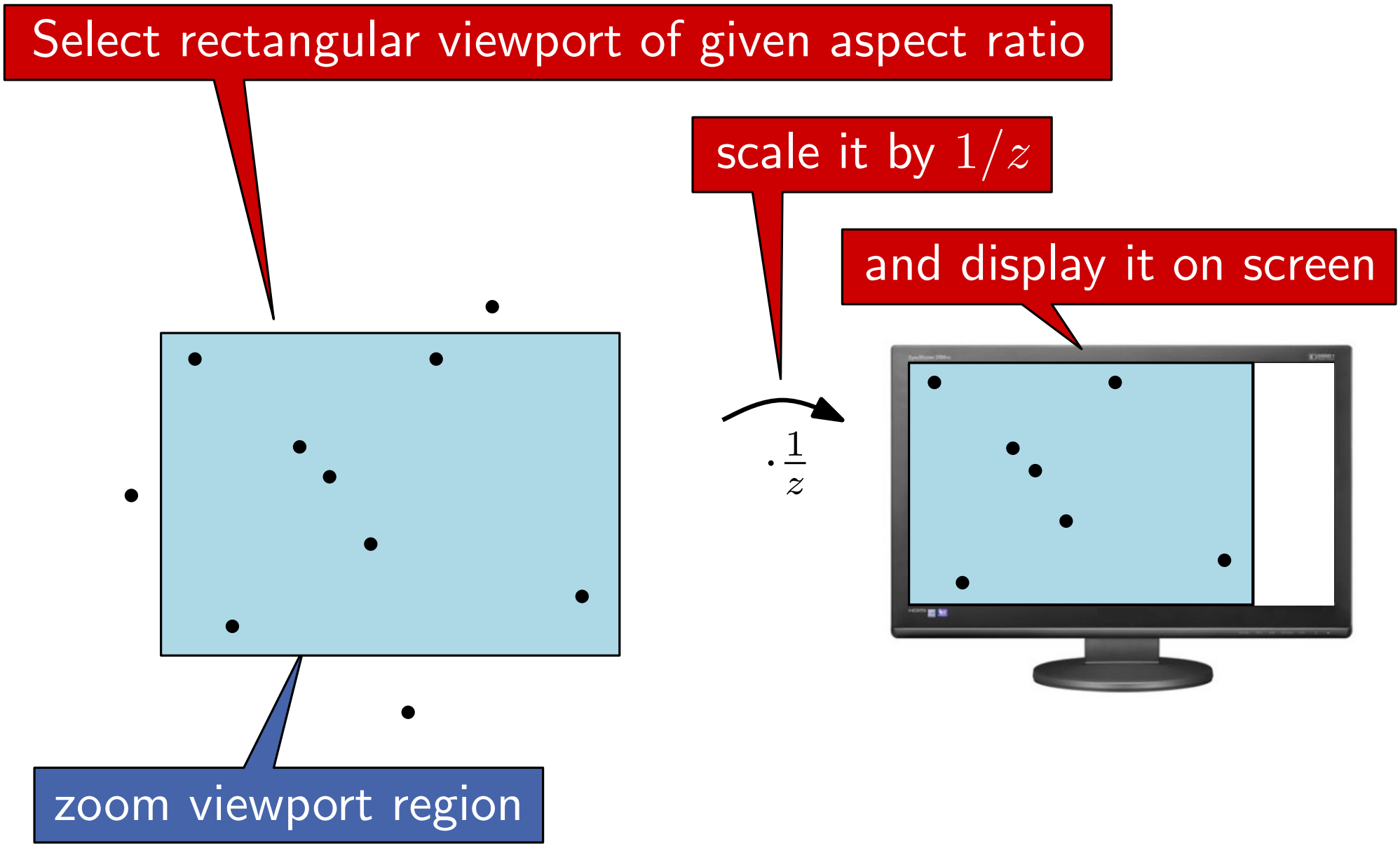
Model


Select rectangular viewport of given aspect ratio
scale it by $1/z$
and display it on screen
$\cdot \frac{1}{z}$

Model


Select rectangular viewport of given aspect ratio
scale it by $1/z$
and display it on screen
$\cdot \frac{1}{z}$
zoom viewport region

Model

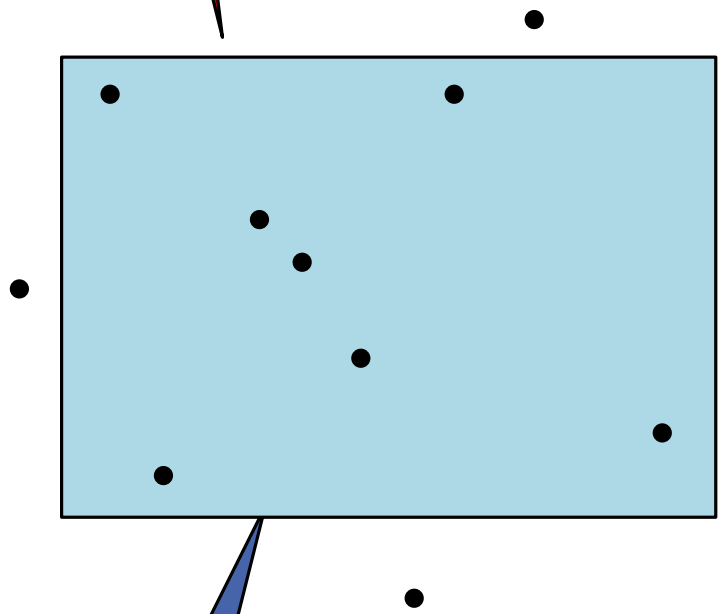

Select rectangular viewport of given aspect ratio
scale it by $1/z$
and display it on screen
$\cdot\frac{1}{z}$
zoom viewport region

Model

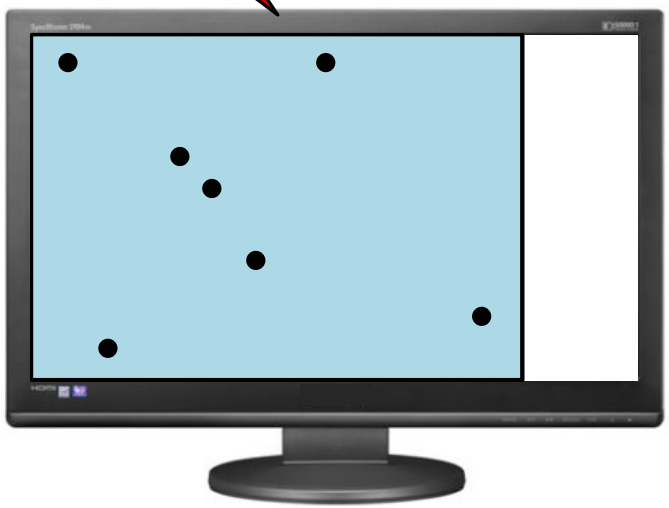
Select rectangular viewport of given aspect ratio

scale it by $1/z$

and display it on screen



$\cdot \frac{1}{z}$



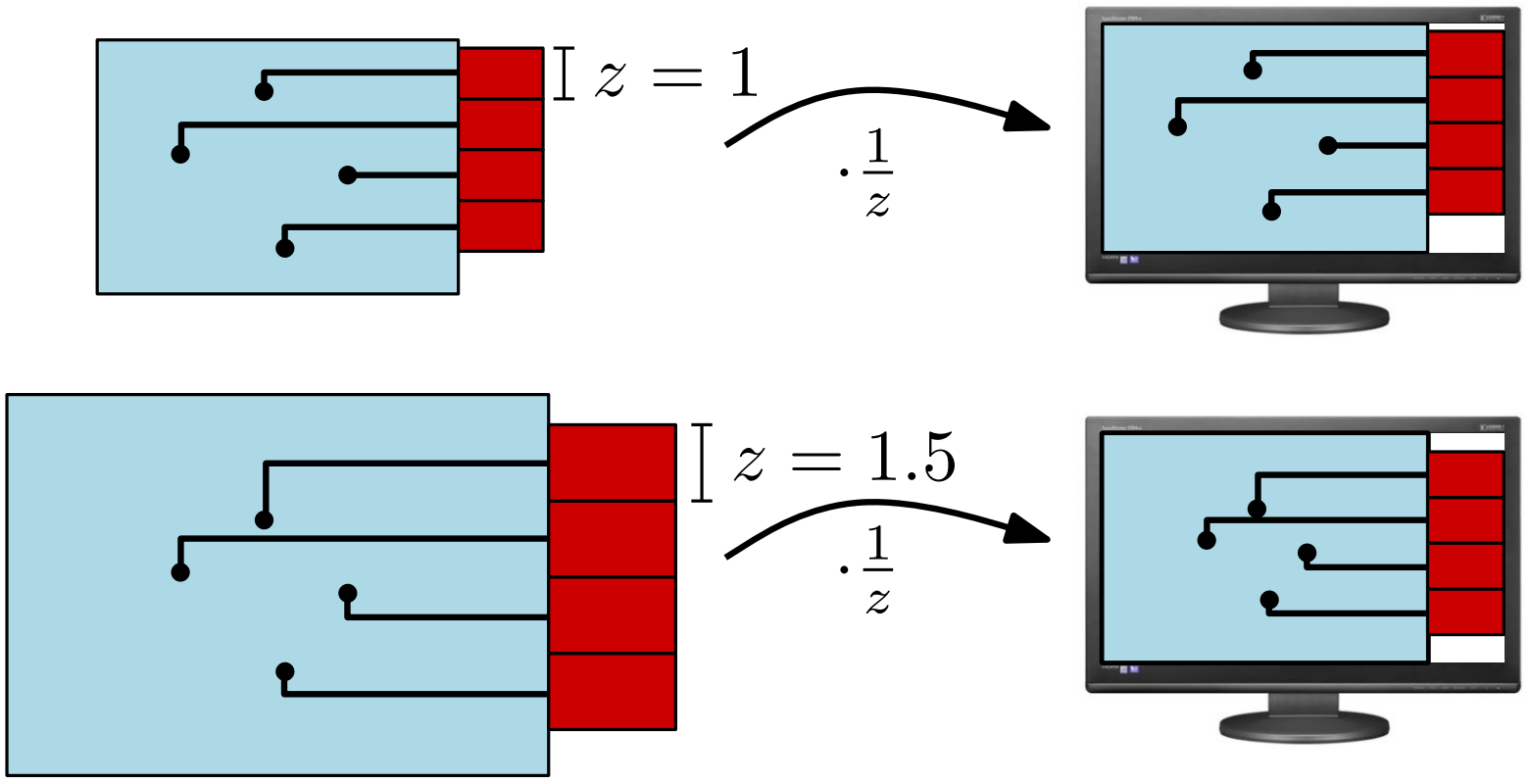
zoom viewport region

$z \in [1, \infty]$ is called zoom level

Label placement and zooming

Objectives

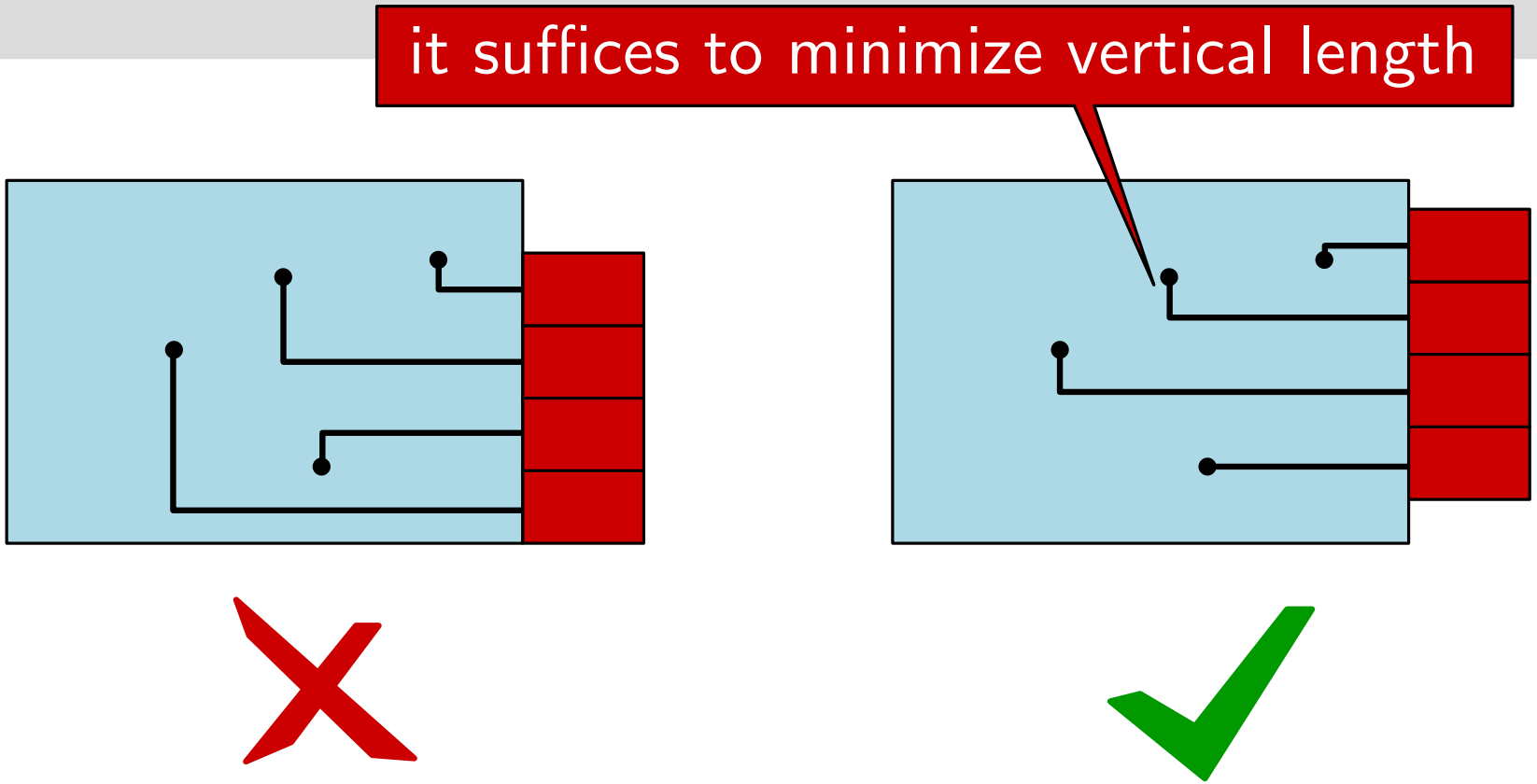
- constant label size on screen
  $\rightarrow$ labels "grow" linearly in z due to scaling with $1/z$



Label placement and zooming

Objectives

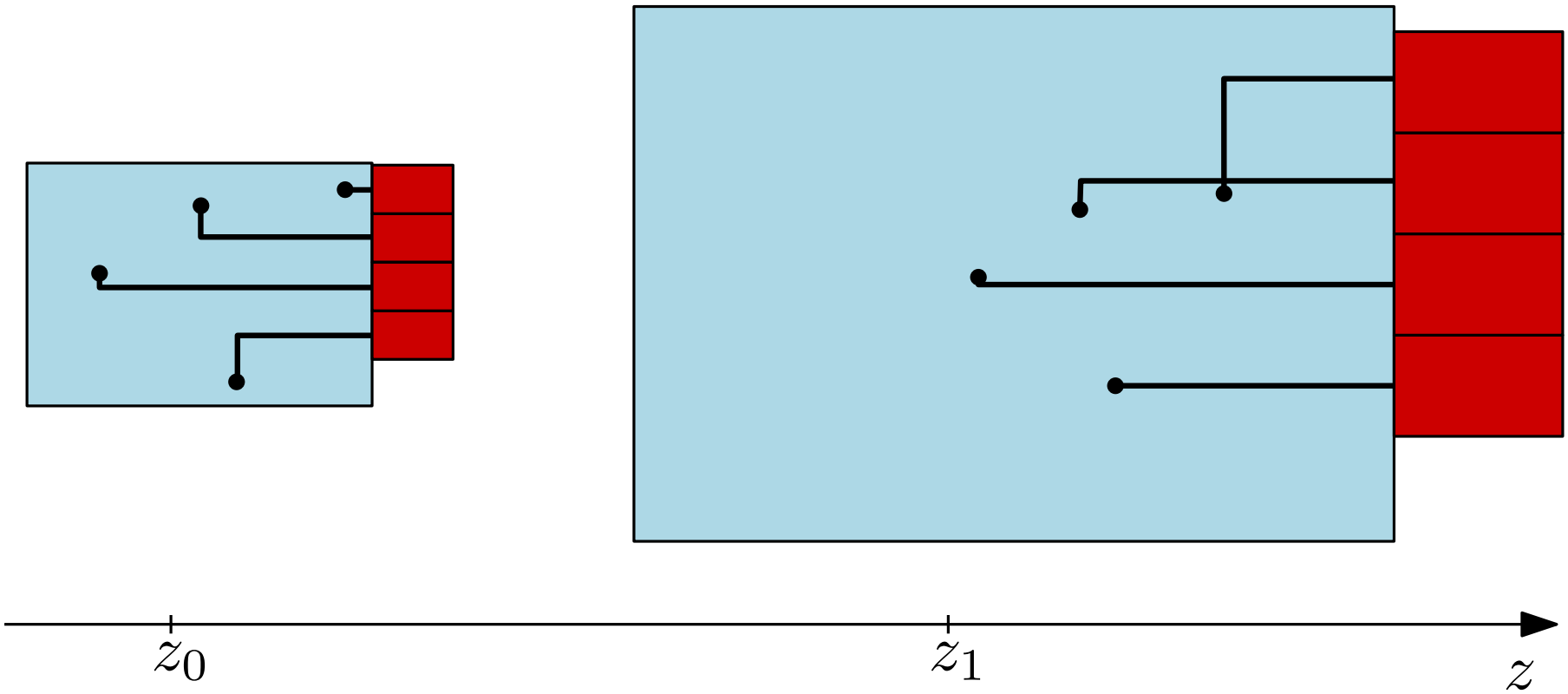
- constant label size on screen
  $\rightarrow$ labels "grow" linearly in z due to scaling with $1/z$
- minimize total leader length



Label placement and zooming

Objectives

- constant label size on screen
  $\rightarrow$ labels "grow" linearly in z due to scaling with $1/z$
- minimize total leader length
- optimal label placement as a function of z



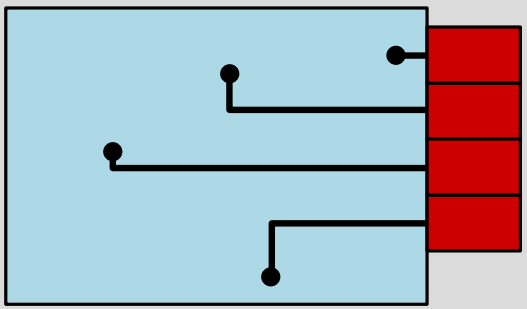
Label placement and zooming

Objectives

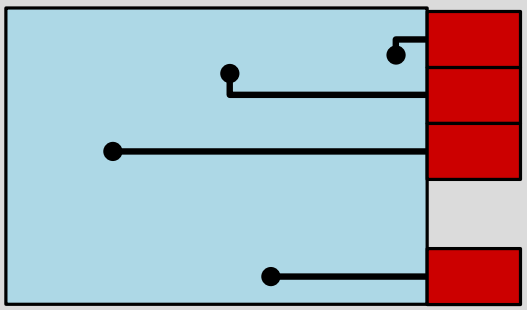
- constant label size on screen
  $\rightarrow$ labels "grow" linearly in z due to scaling with $1/z$
- minimize total leader length
- optimal label placement as a function of z

Two variants

- single-stack labeling

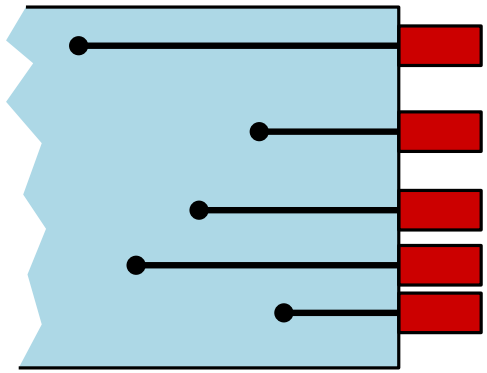


- clustered labeling

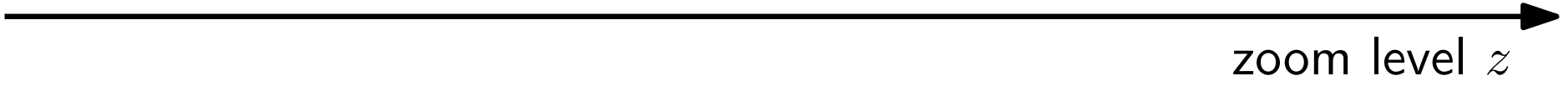


Dynamic clustering

clusters form as z increases

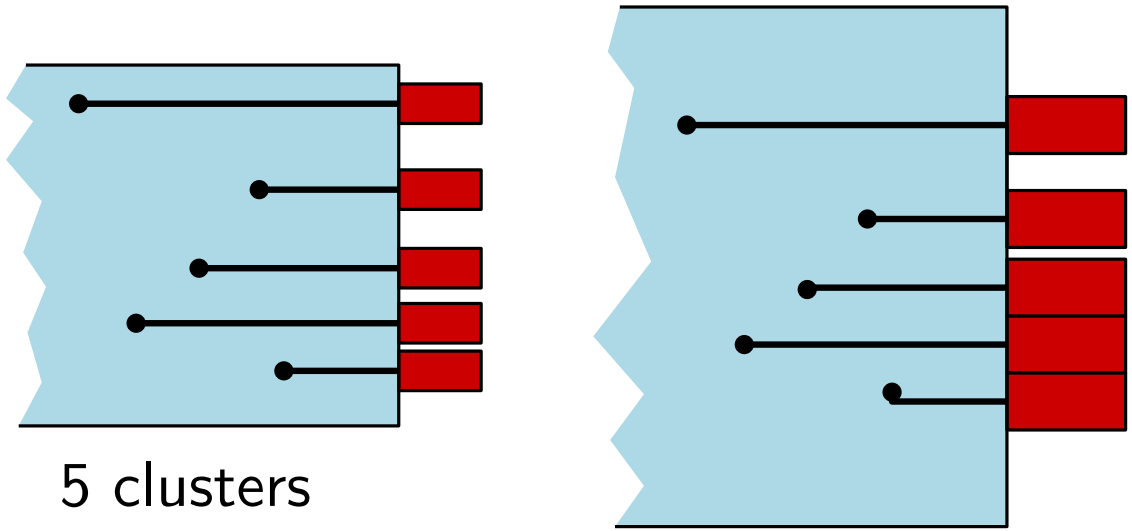


5 clusters



Dynamic clustering

clusters form as z increases



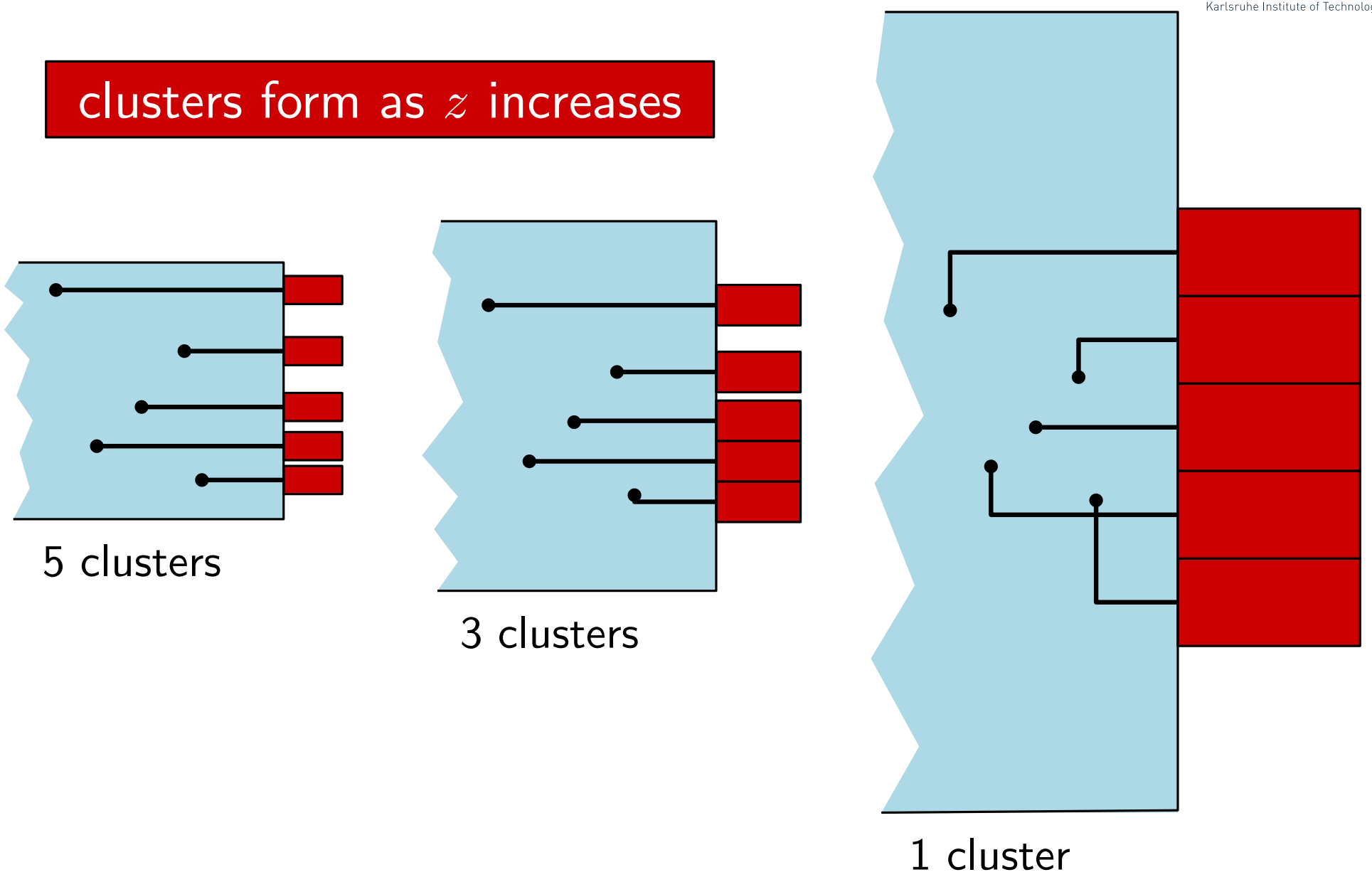
5 clusters

3 clusters

zoom level z

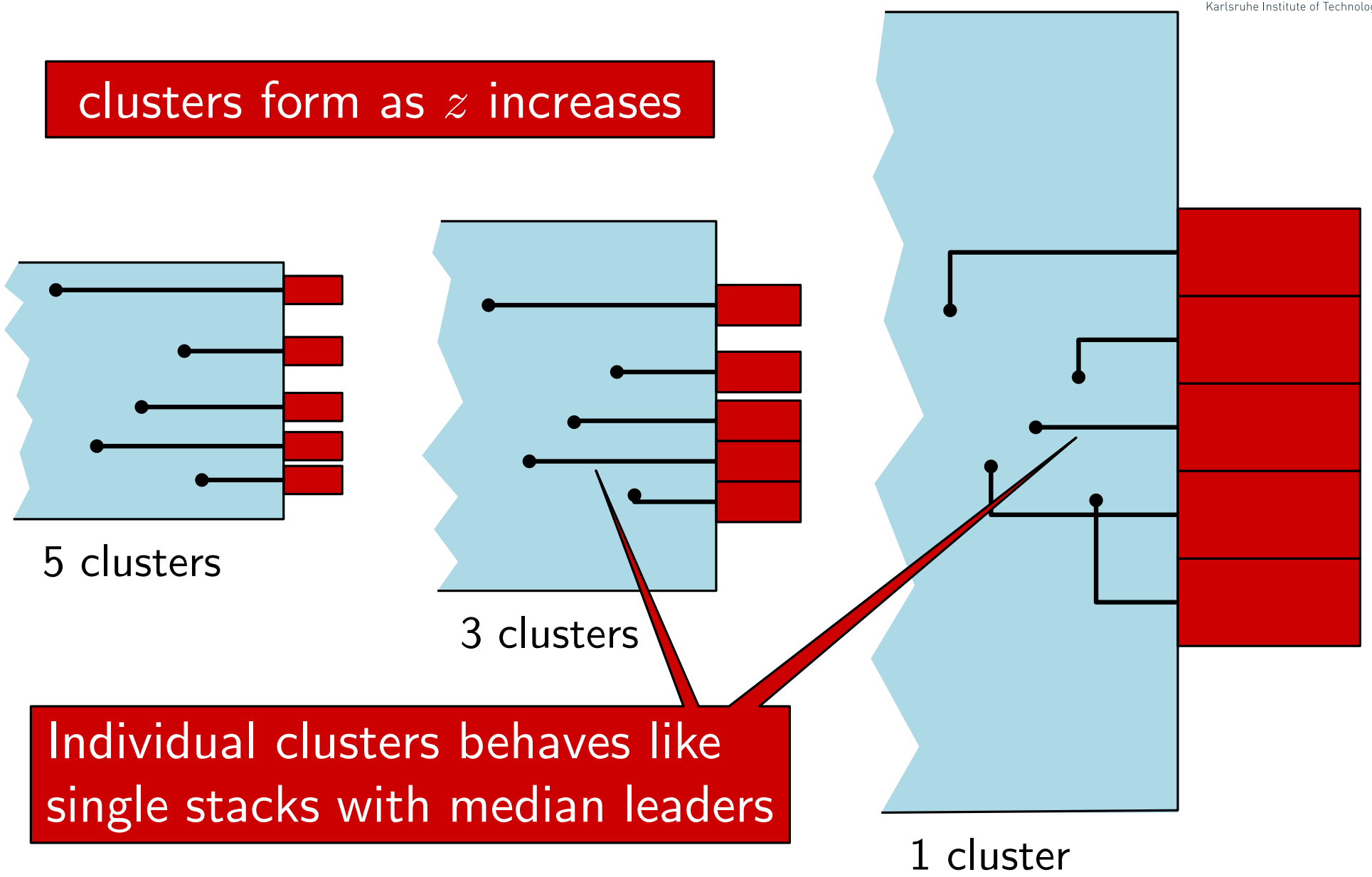
Dynamic clustering

clusters form as z increases



Dynamic clustering

clusters form as z increases

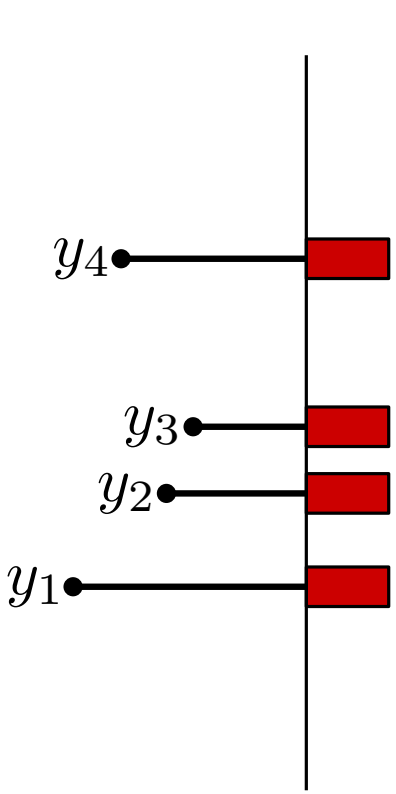


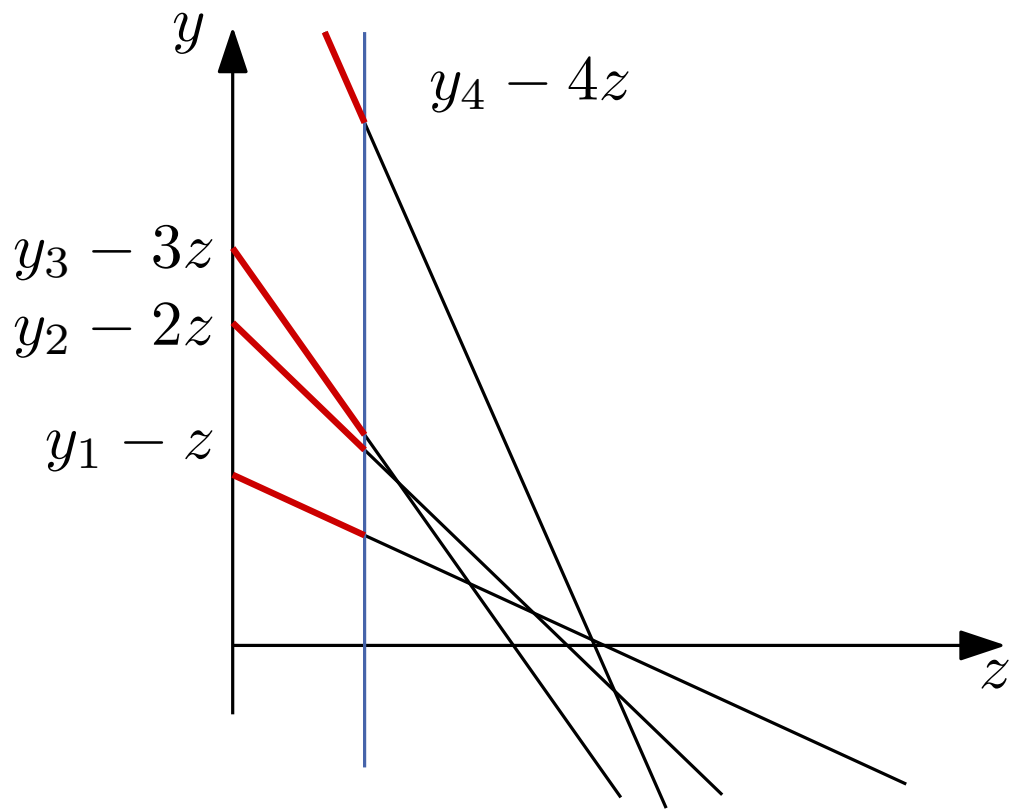
Cluster merging

Observation

Two adjacent clusters collide when the distance between the two medians $y_j - y_i$ equals the height $(j-i) \cdot z$ of the labels between them.

This is the intersection point of the lines $y = y_i - iz$ and $y = y_j - jz$!



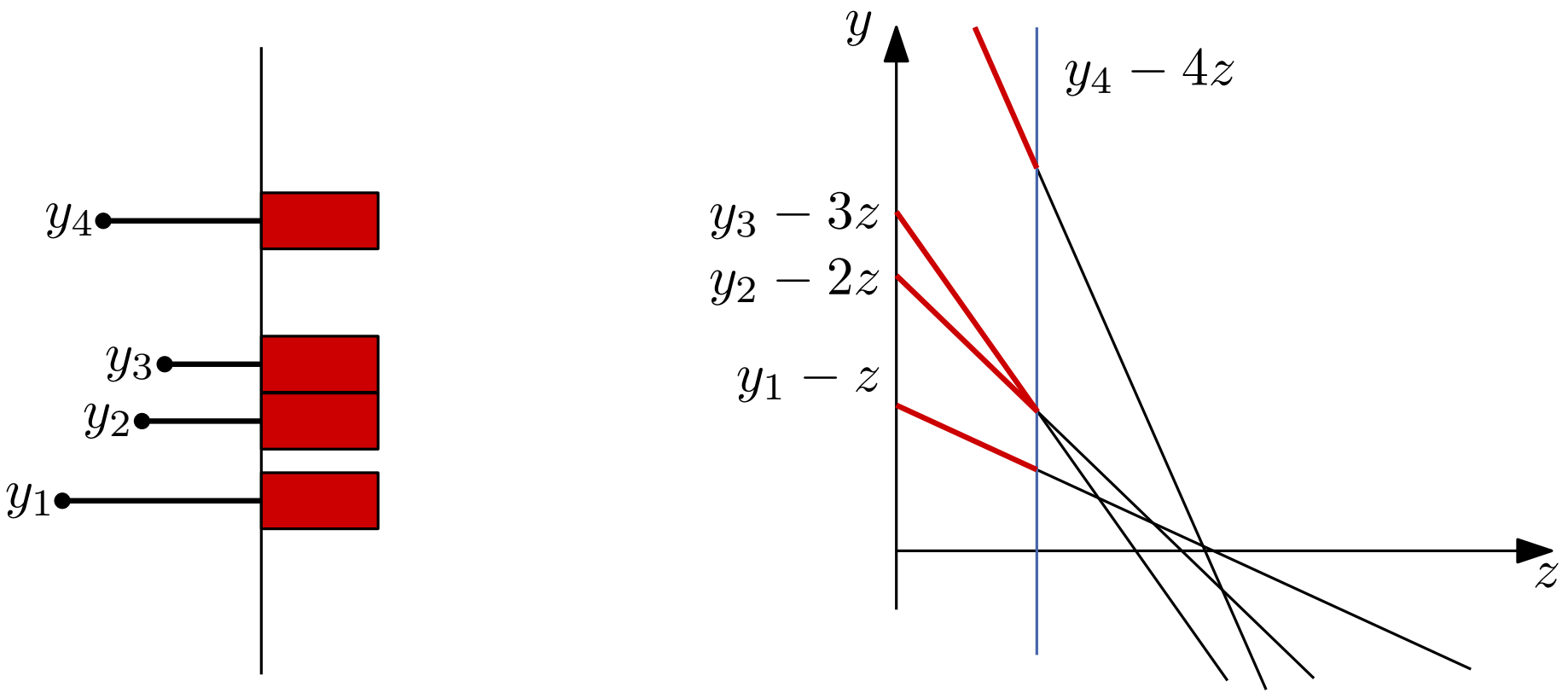


Cluster merging

Observation

Two adjacent clusters collide when the distance between the two medians $y_j - y_i$ equals the height $(j-i) \cdot z$ of the labels between them.

This is the intersection point of the lines $y = y_i - iz$ and $y = y_j - jz$!

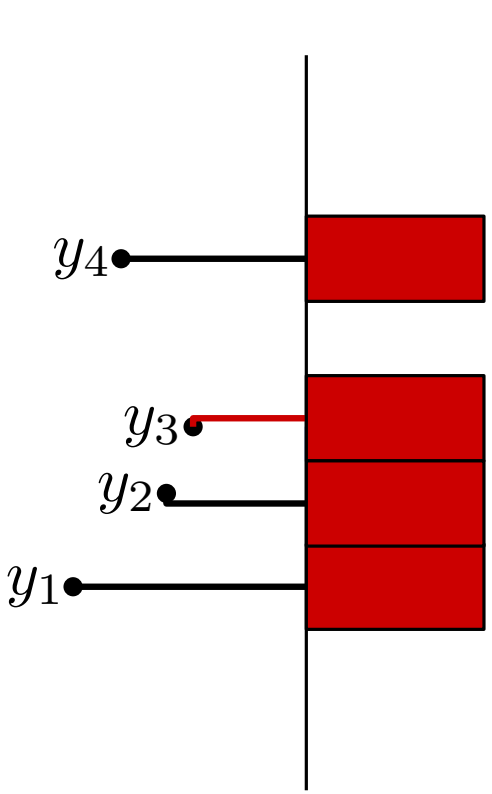


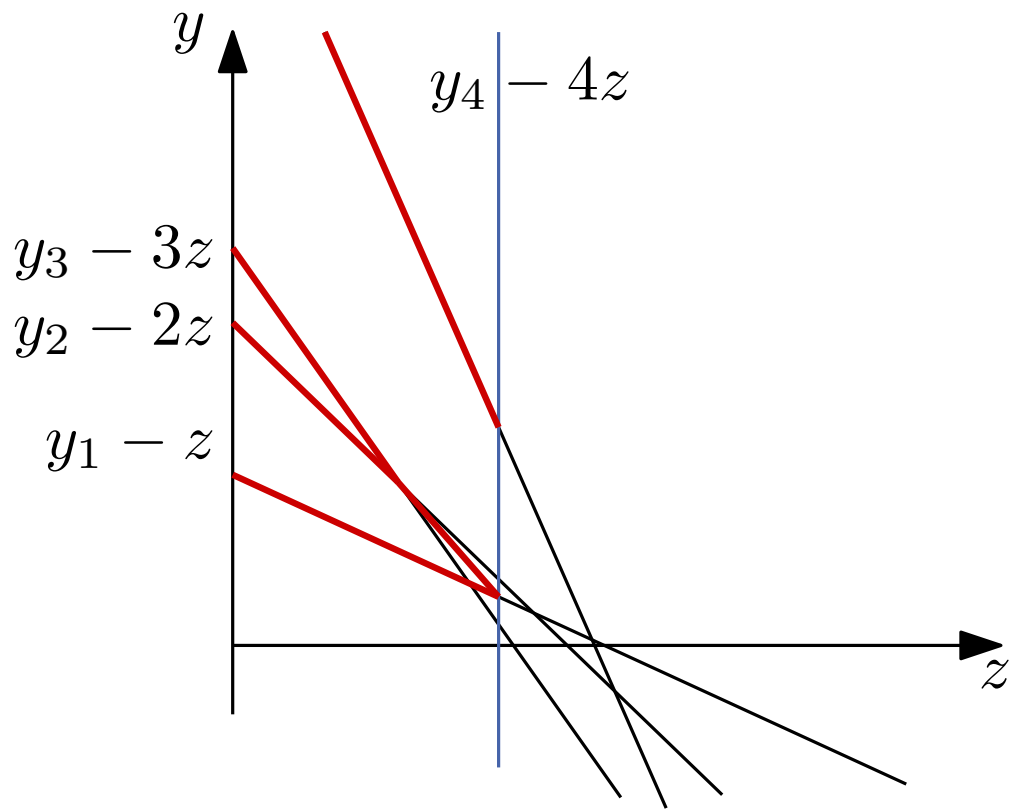
Cluster merging

Observation

Two adjacent clusters collide when the distance between the two medians $y_j - y_i$ equals the height $(j-i) \cdot z$ of the labels between them.

This is the intersection point of the lines $y = y_i - iz$ and $y = y_j - jz$!



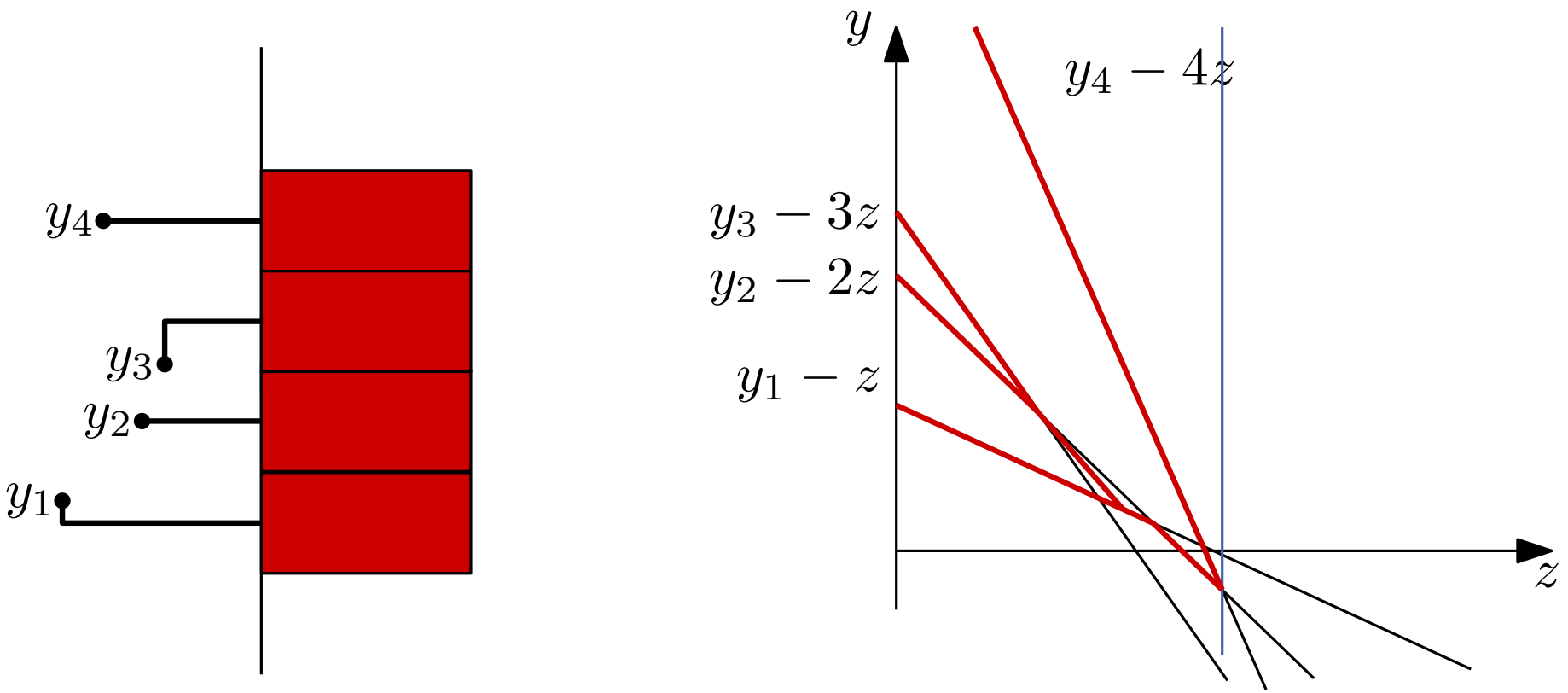


Cluster merging

Observation

Two adjacent clusters collide when the distance between the two medians $y_j - y_i$ equals the height $(j-i) \cdot z$ of the labels between them.

This is the intersection point of the lines $y = y_i - iz$ and $y = y_j - jz$!

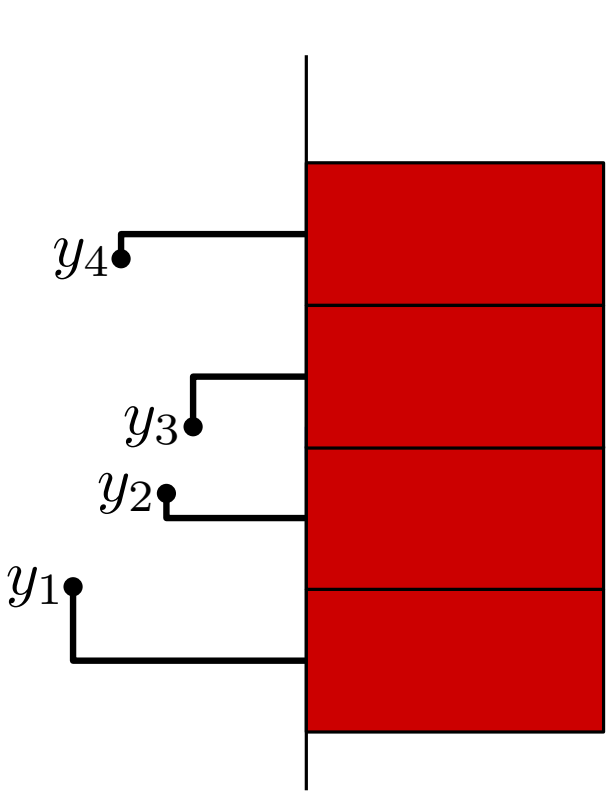


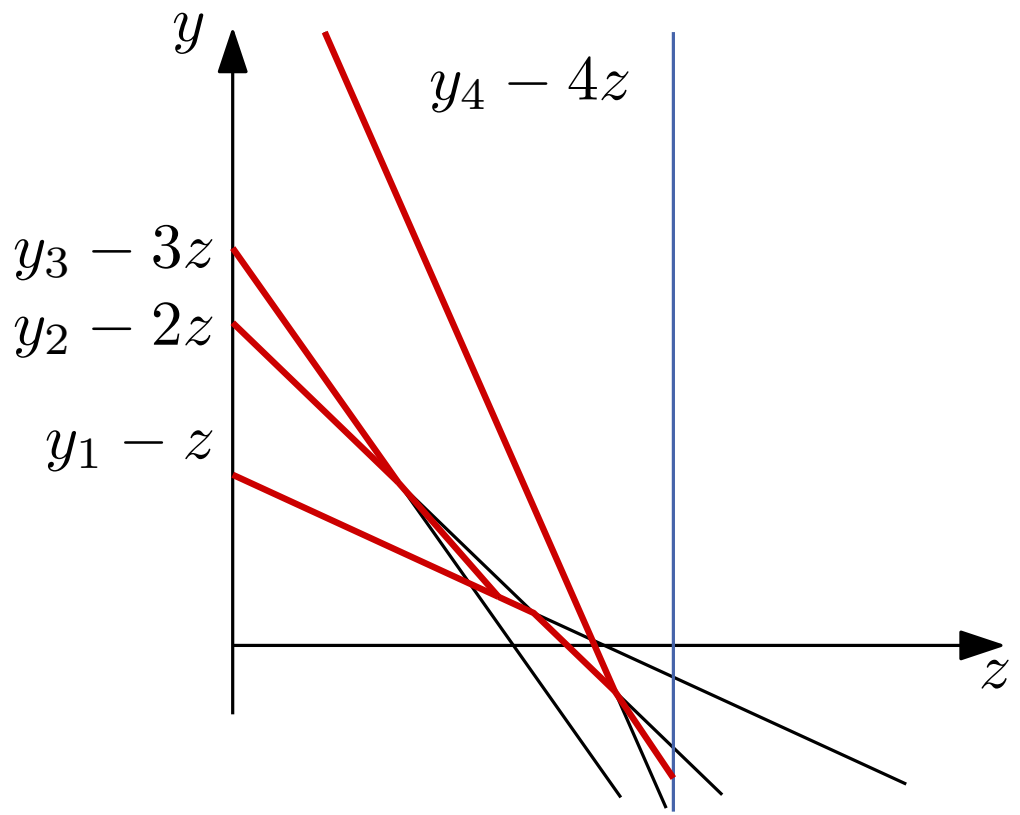
Cluster merging

Observation

Two adjacent clusters collide when the distance between the two medians $y_j - y_i$ equals the height $(j-i) \cdot z$ of the labels between them.

This is the intersection point of the lines $y = y_i - iz$ and $y = y_j - jz$!





Cluster merging

Observation

Two adjacent clusters collide when the distance between the two medians $y_j - y_i$ equals the height $(j - i) \cdot z$ of the labels between them.

This is the intersection point of the lines $y = y_i - iz$ and $y = y_j - jz$!

kurze Demo

http://www.cs.helsinki.fi/group/compgeom/boundarylabeling