Computational Geometry

Lecture 2:
Line-Segment Intersection
or
Map Overlay

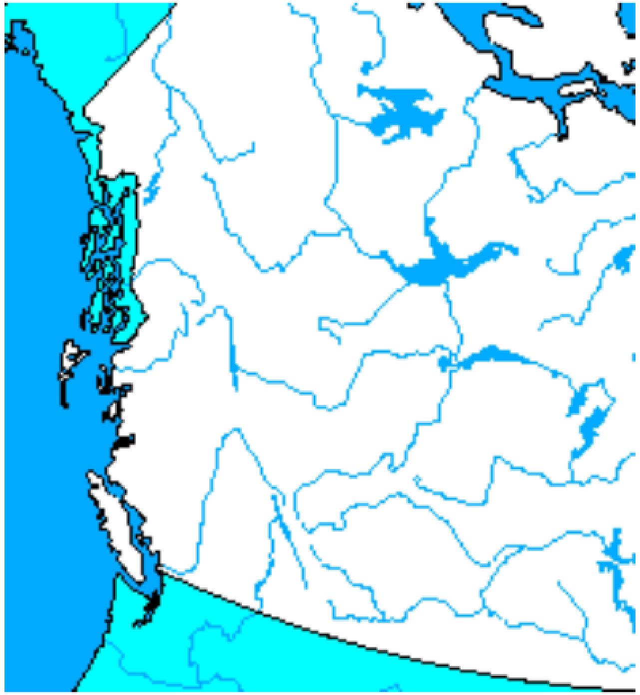
Part I:
Map Overlay

Philipp Kindermann

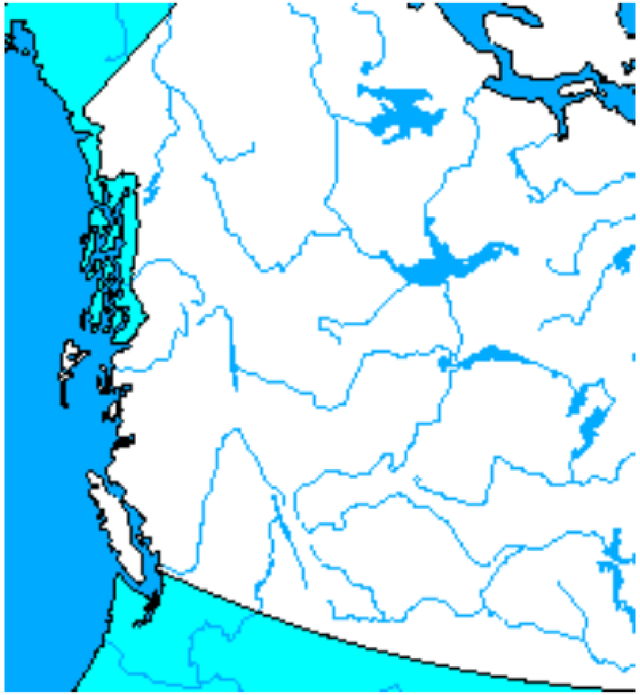
Winter Semester 2020

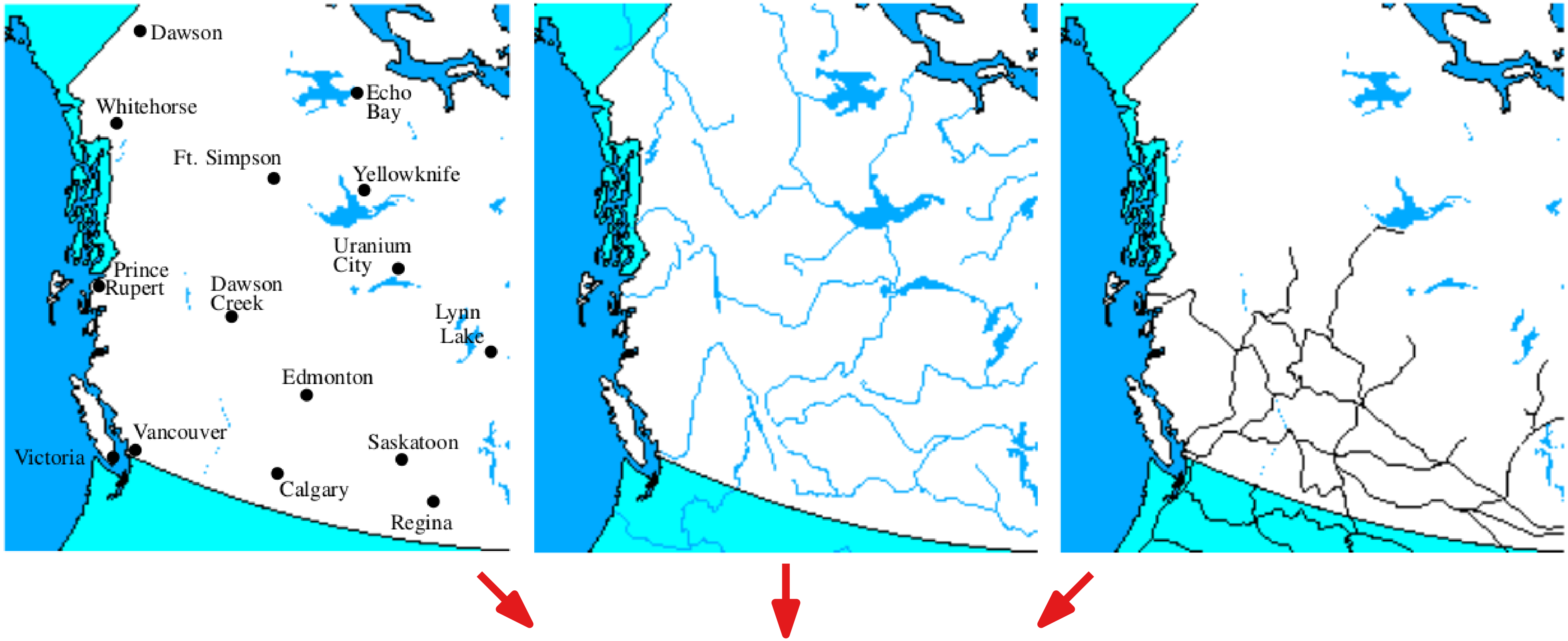

Dawson
Whitehorse
Echo Bay
Ft. Simpson
Yellowknife
Uranium City
Prince Rupert
Dawson Creek
Lynn Lake
Edmonton
Vancouver
Victoria
Saskatoon
Calgary
Regina


Dawson
Echo
Bay
Whitehorse
Ft. Simpson
Yellowknife
Uranium
City
Prince
Rupert
Dawson
Creek
Lynn
Lake
Edmonton
Vancouver
Victoria
Saskatoon
Calgary
Regina

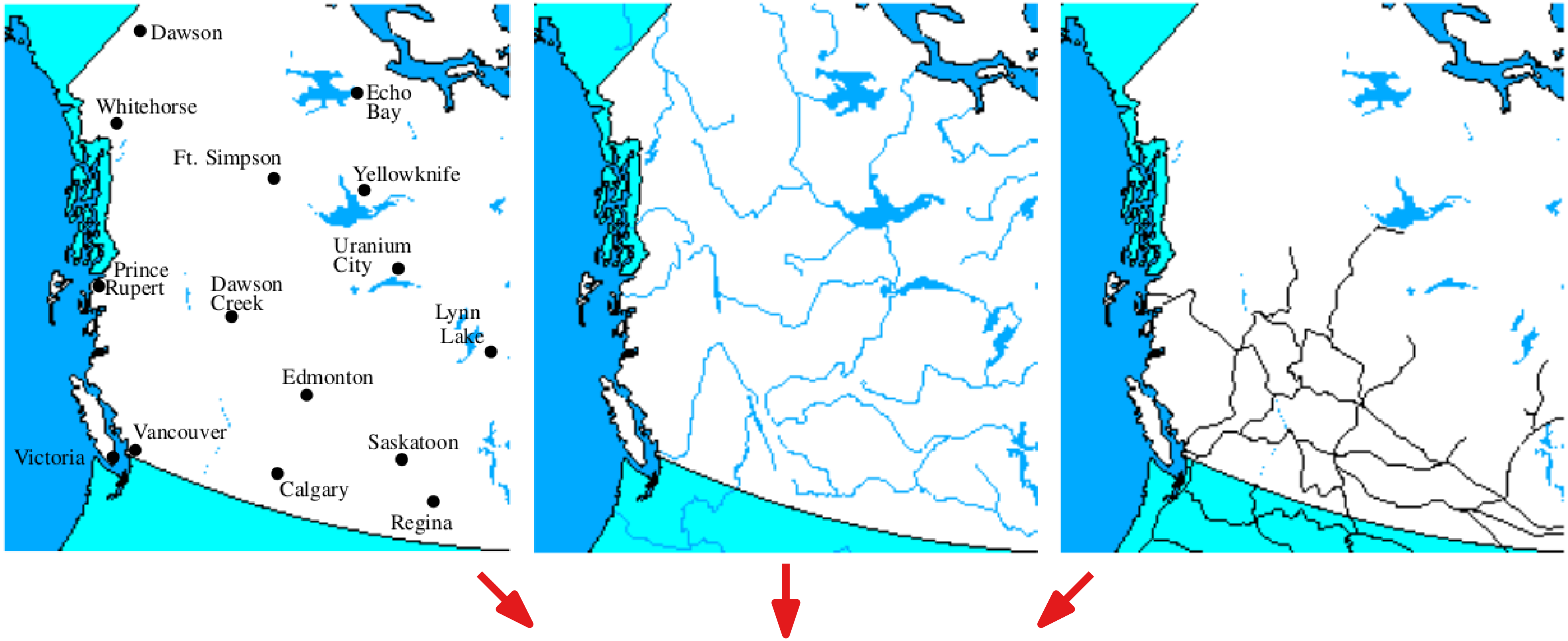



Dawson
Whitehorse
Echo
Bay
Ft. Simpson
Yellowknife
Uranium
City
Prince
Rupert
Dawson
Creek
Lynn
Lake
Edmonton
Vancouver
Victoria
Saskatoon
Calgary
Regina

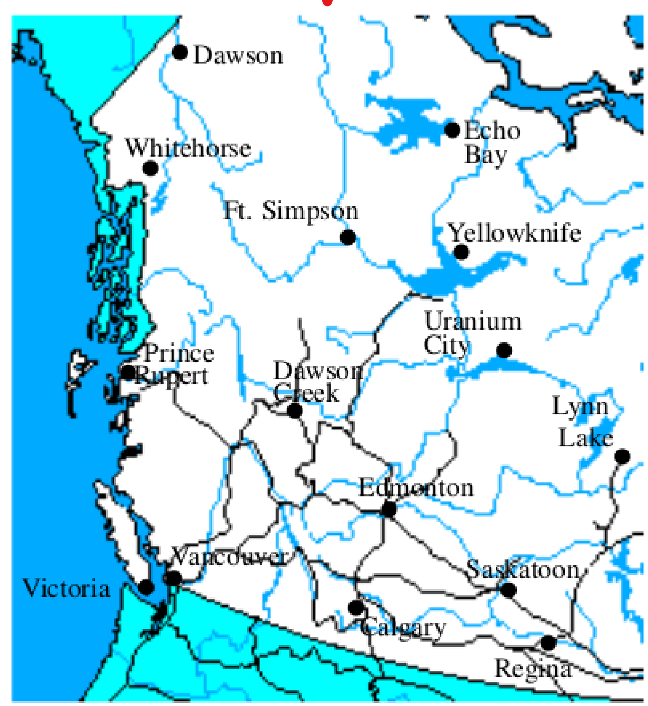


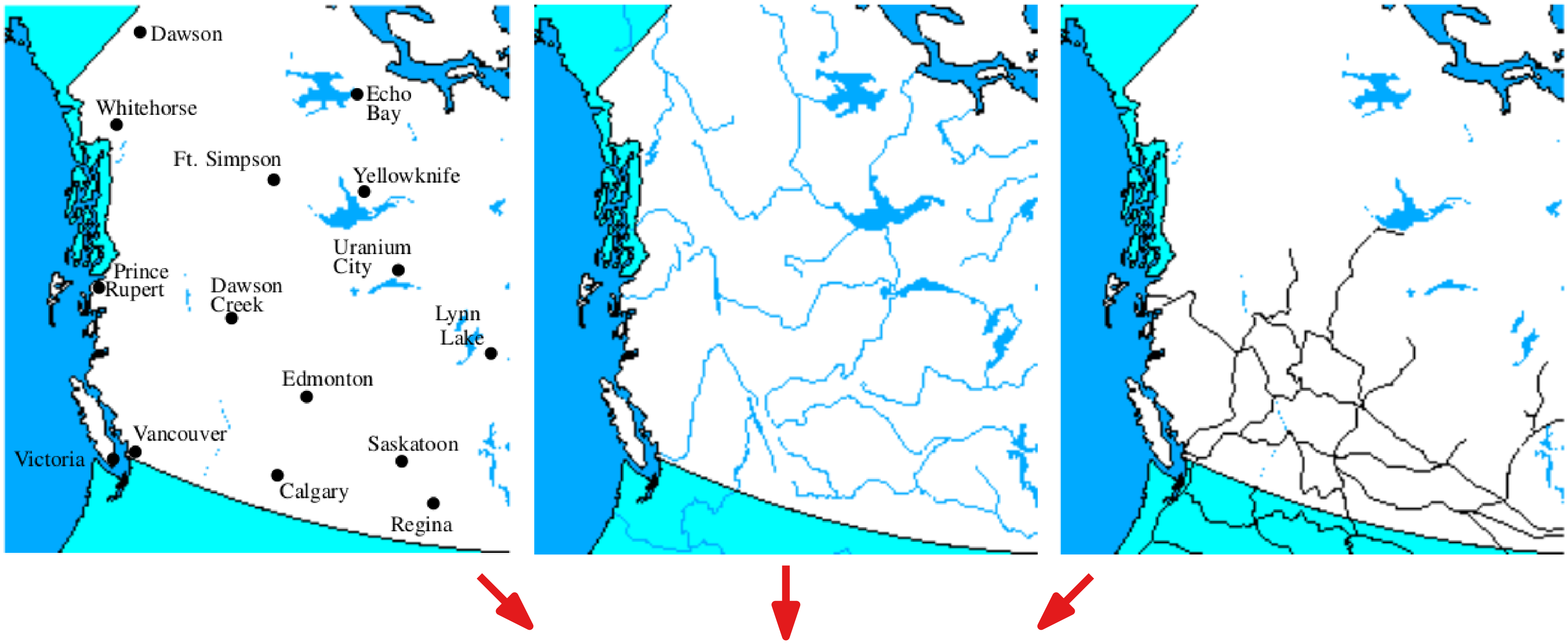


Map Overlay in Geographic Information Systems (GIS)

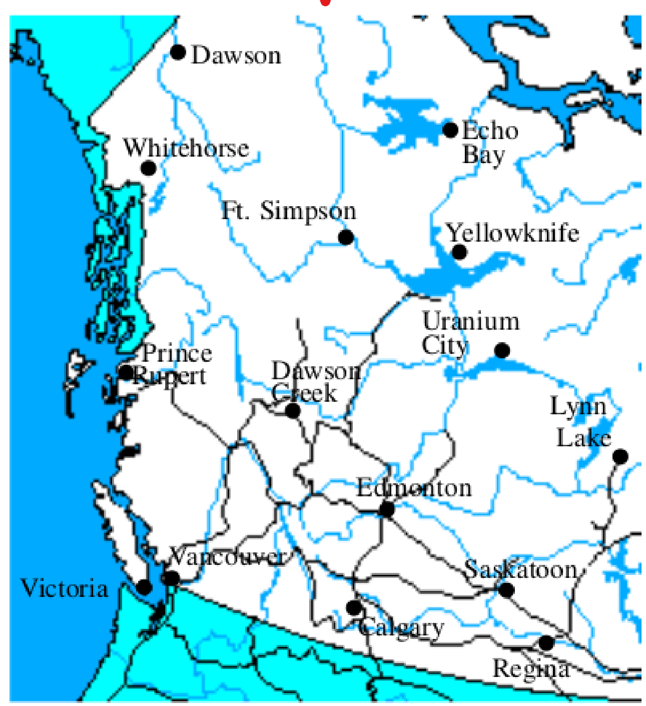


Map Overlay in Geographic Information Systems (GIS)

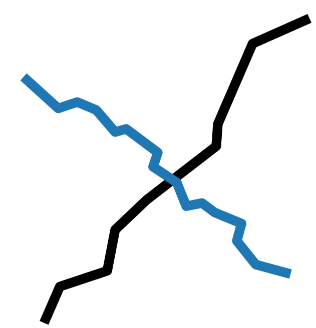


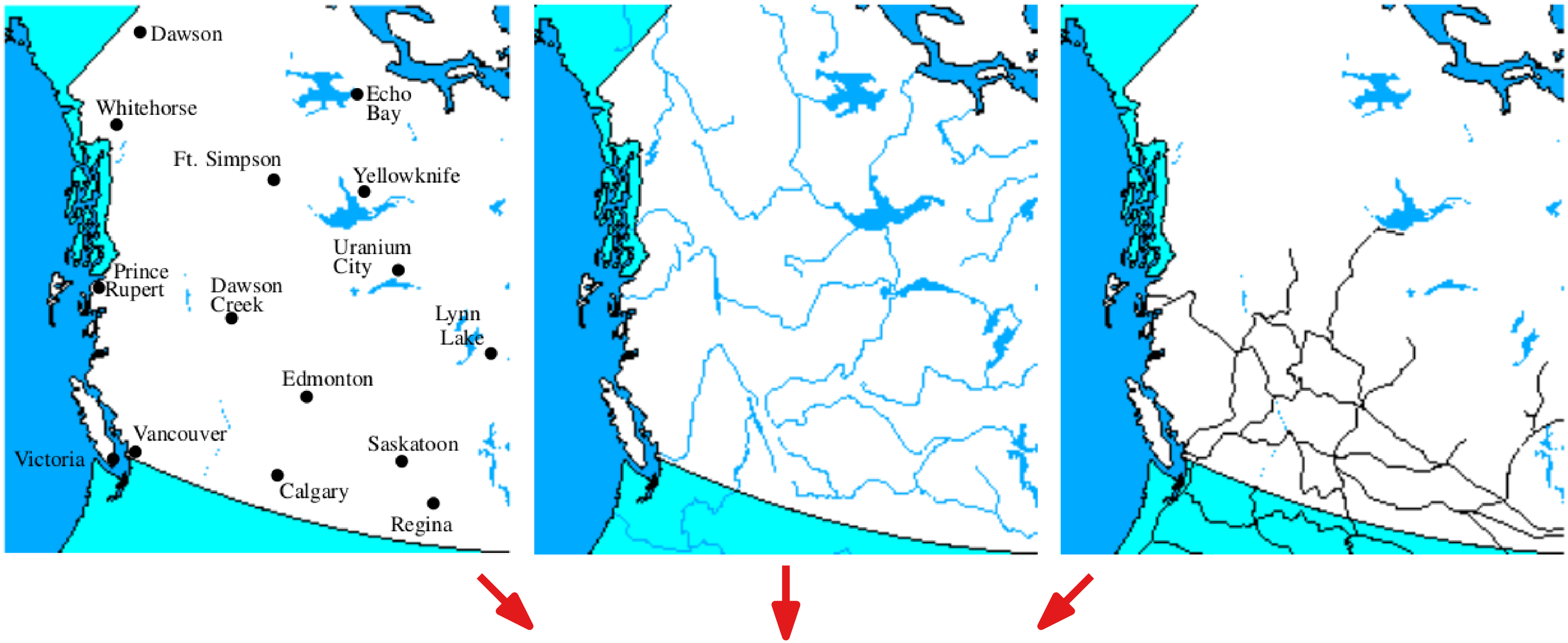


Map Overlay in Geographic Information Systems (GIS)

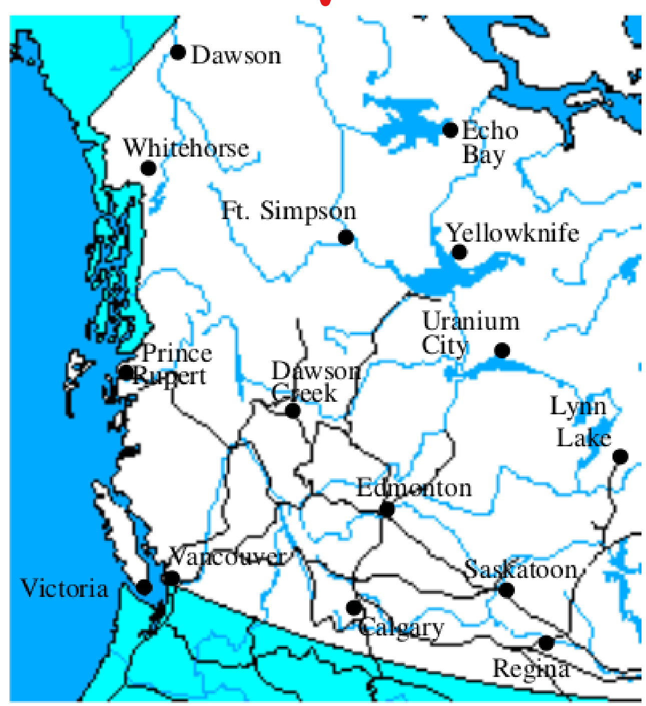


Here:

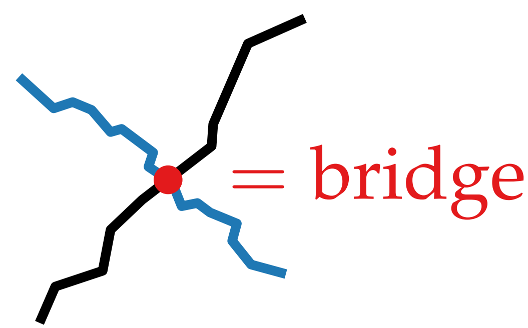




Map Overlay in Geographic Information Systems (GIS)



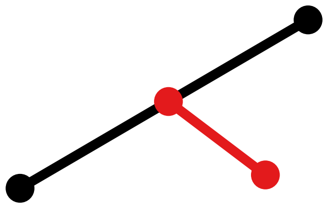
Here:



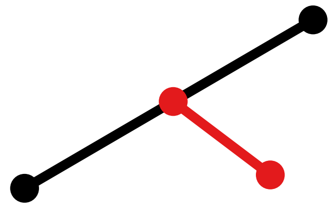
Line-Segment Intersection

Definition:

Line-Segment Intersection

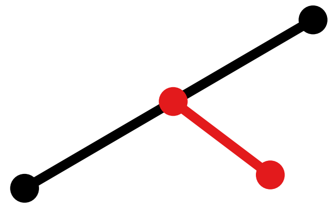
Definition: Is  an intersection?

Line-Segment Intersection

Definition: Is  an intersection?

Answer: Depends. . .

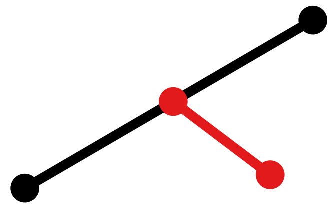
Line-Segment Intersection

Definition: Is  an intersection?

Answer: Depends. . .

Problem: Given a set S of n *closed* non-overlapping line segments in the plane, compute. . .

Line-Segment Intersection

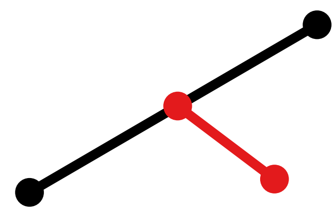
Definition: Is  an intersection?

Answer: Depends. . .

Problem: Given a set S of n *closed* non-overlapping line segments in the plane, compute. . .

- all points where at least two segments intersect and
- for each such point report all segments that contain it.

Line-Segment Intersection

Definition: Is  an intersection?

Answer: Depends. . .

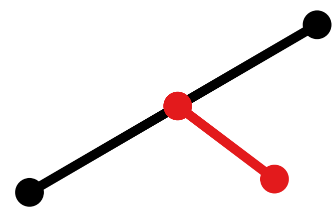
yes!

Problem: Given a set S of n *closed* non-overlapping line segments in the plane, compute. . .

– all points where at least two segments intersect and

– for each such point report all segments that contain it.

Line-Segment Intersection

Definition: Is  an intersection?

Answer: Depends. . .

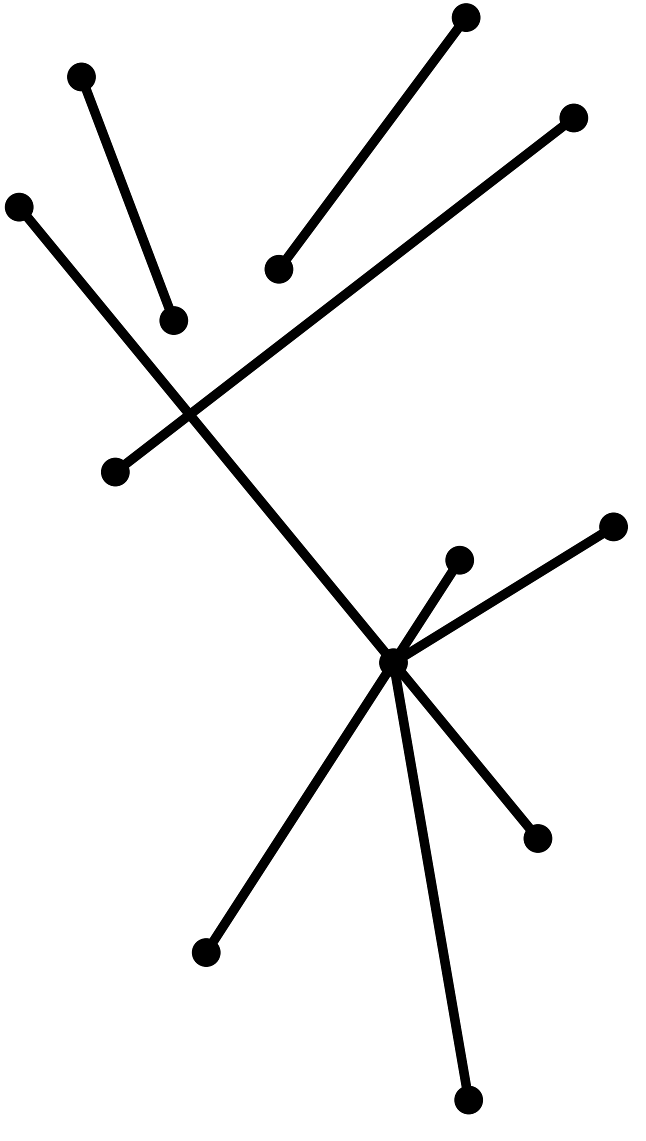
yes!

Problem: Given a set S of n *closed* non-overlapping line segments in the plane, compute. . .

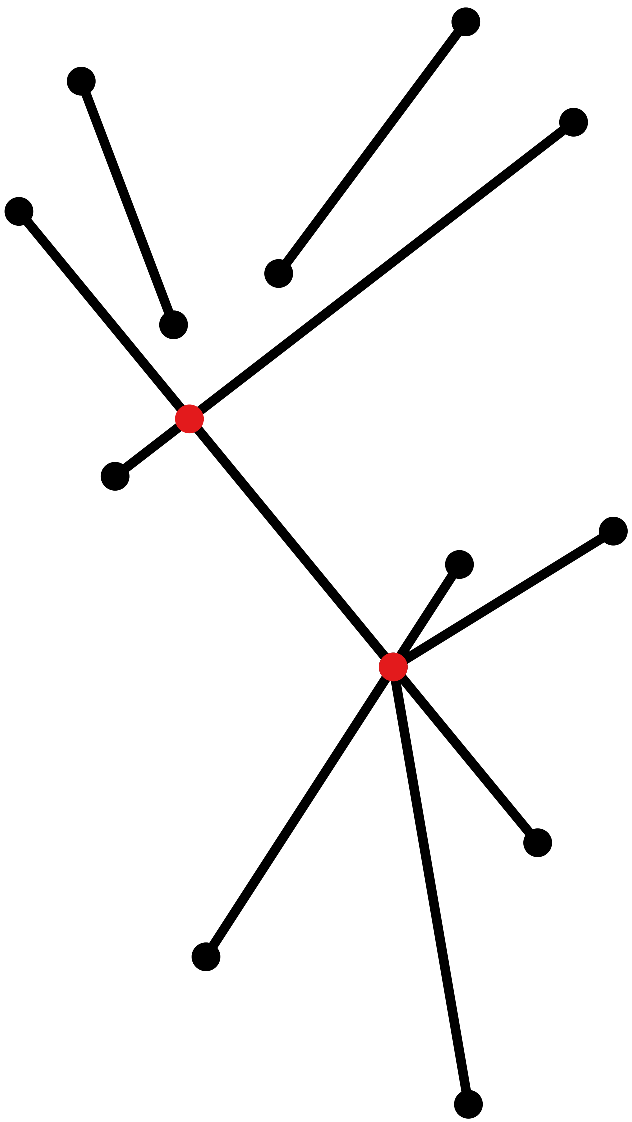
– all points where at least two segments intersect and
– for each such point report all segments that contain it.

Task: How would *you* do it?

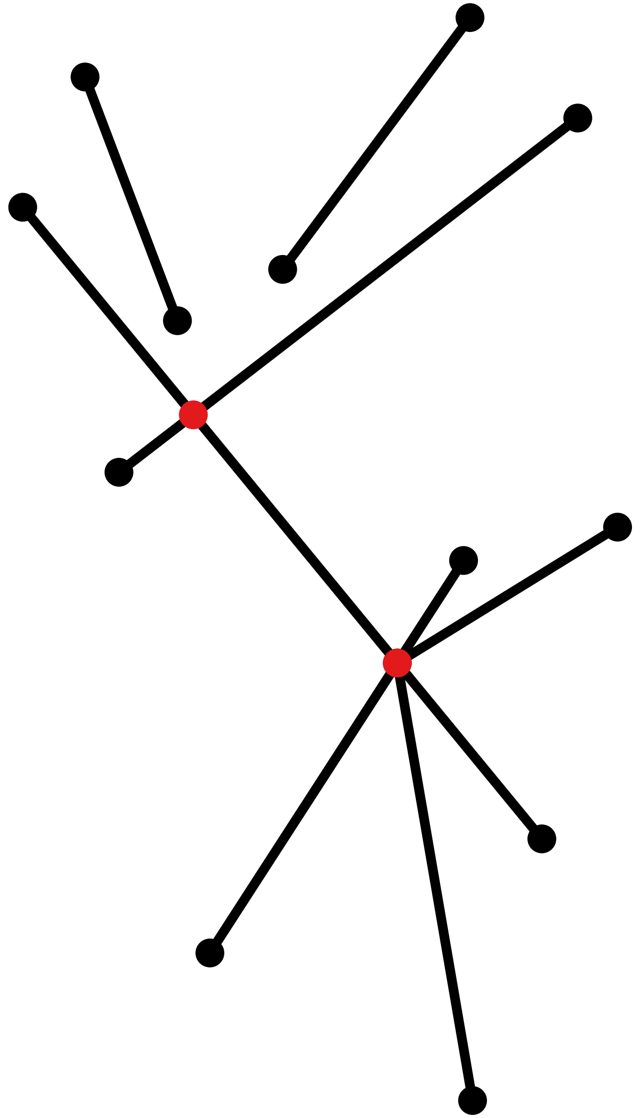
Example



Example



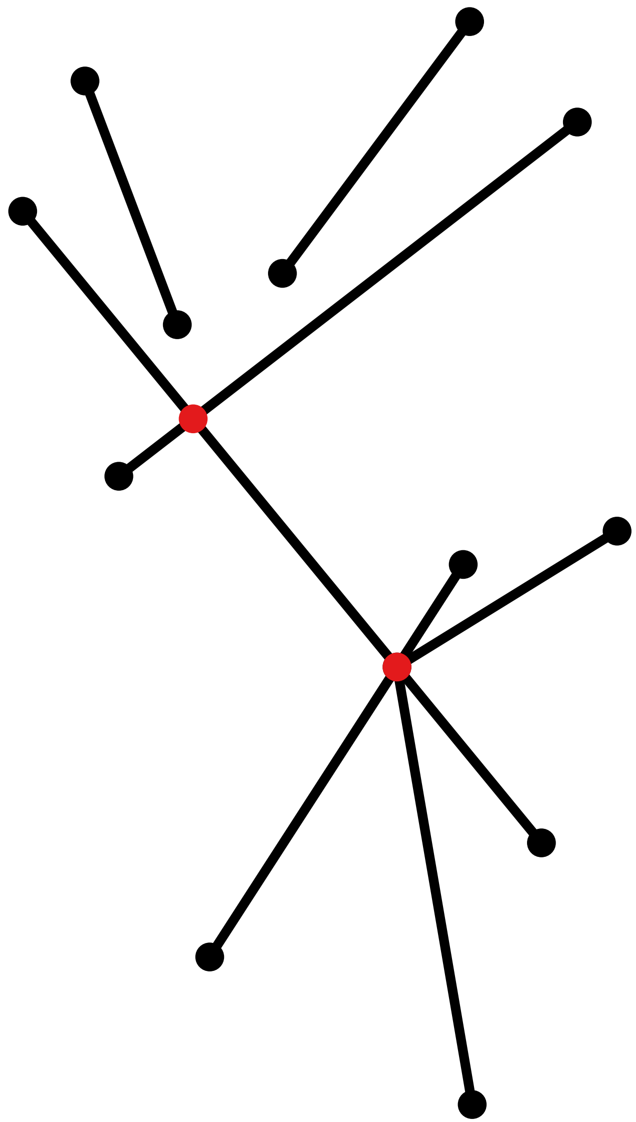
Example



Brute Force?

$O(n^2)$ ... can we do better?

Example



Brute Force?

$O(n^2)$ ... can we do better?

Idea:
Process segments top-to-bottom using a "sweep line".

Computational Geometry

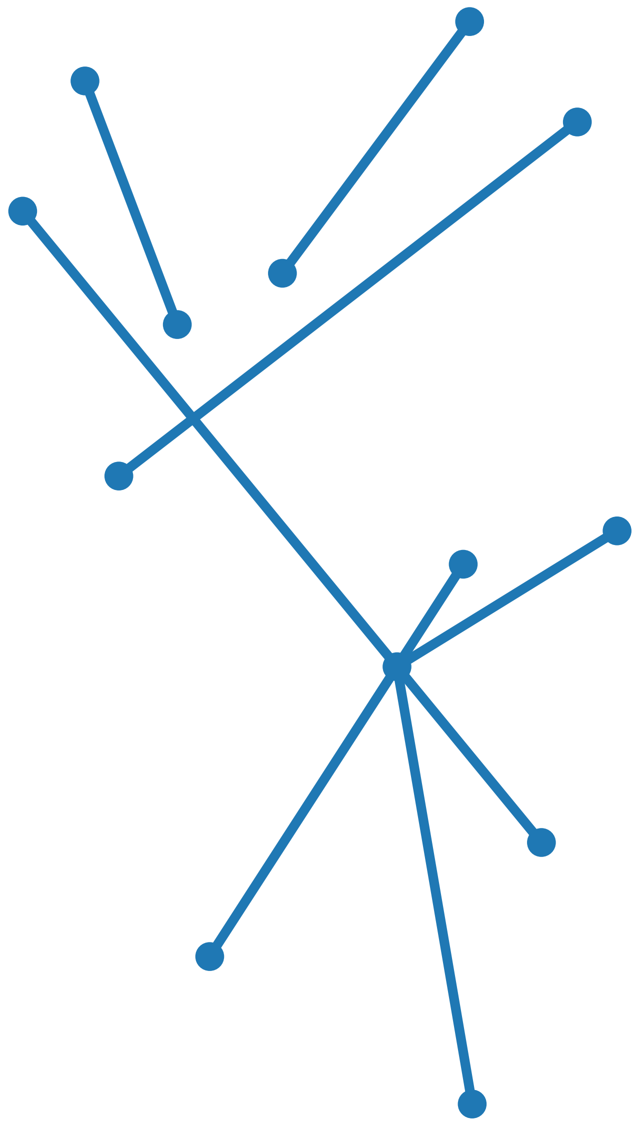
Lecture 2:
Line-Segment Intersection
or
Map Overlay

Part II:
Sweep-Line Algorithm

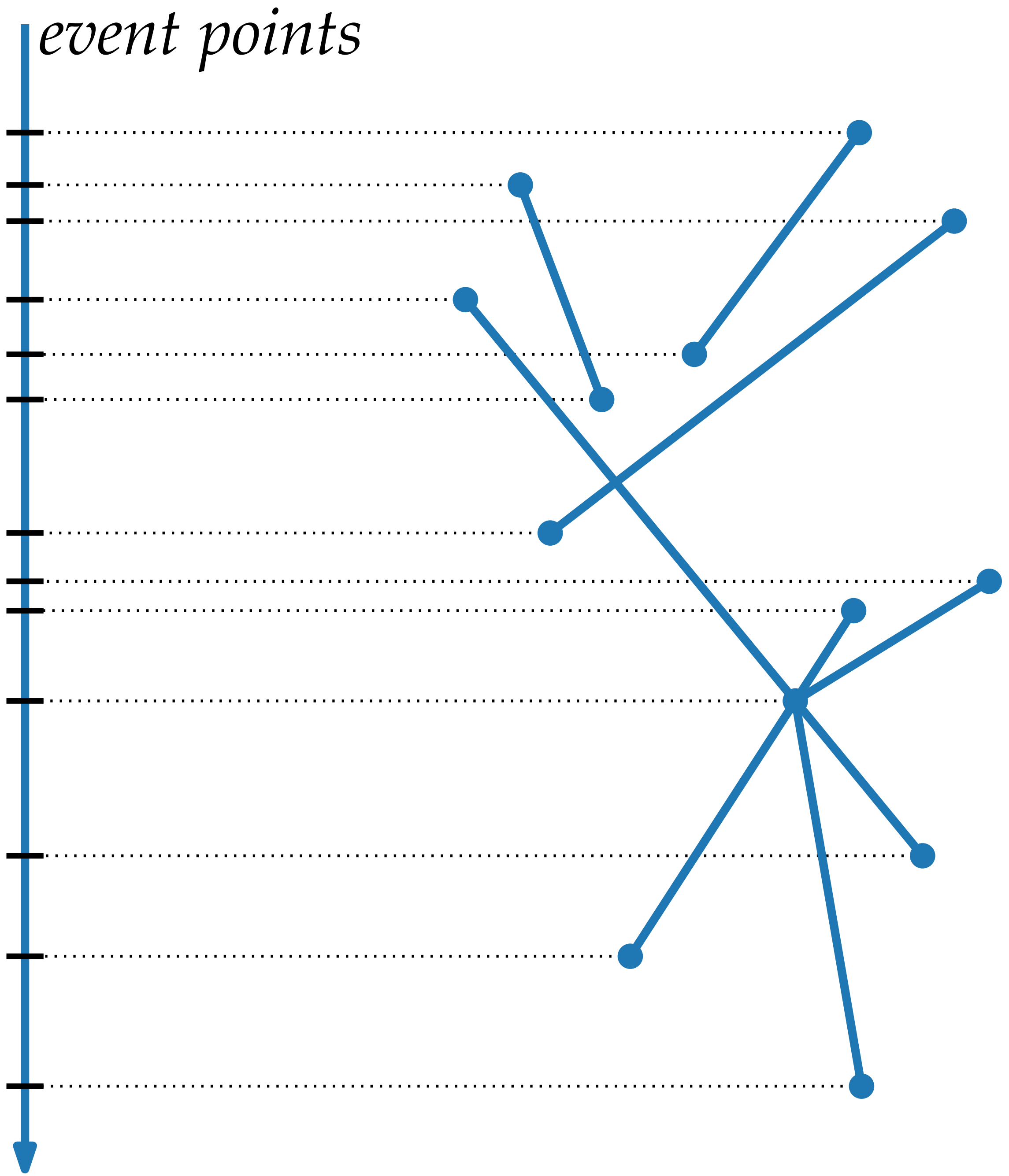
Philipp Kindermann

Winter Semester 2020

Sweep-Line Algorithm

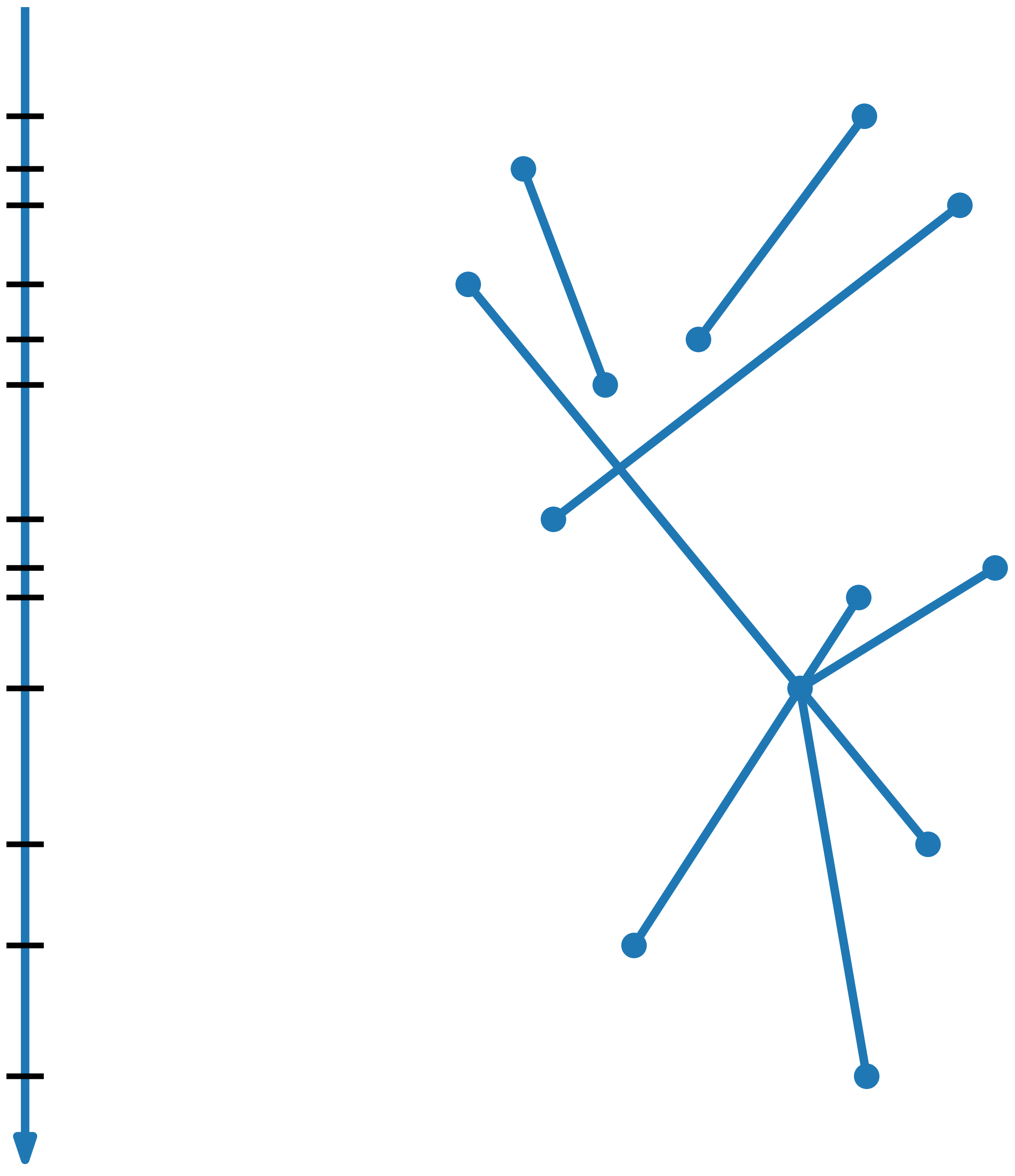


Sweep-Line Algorithm



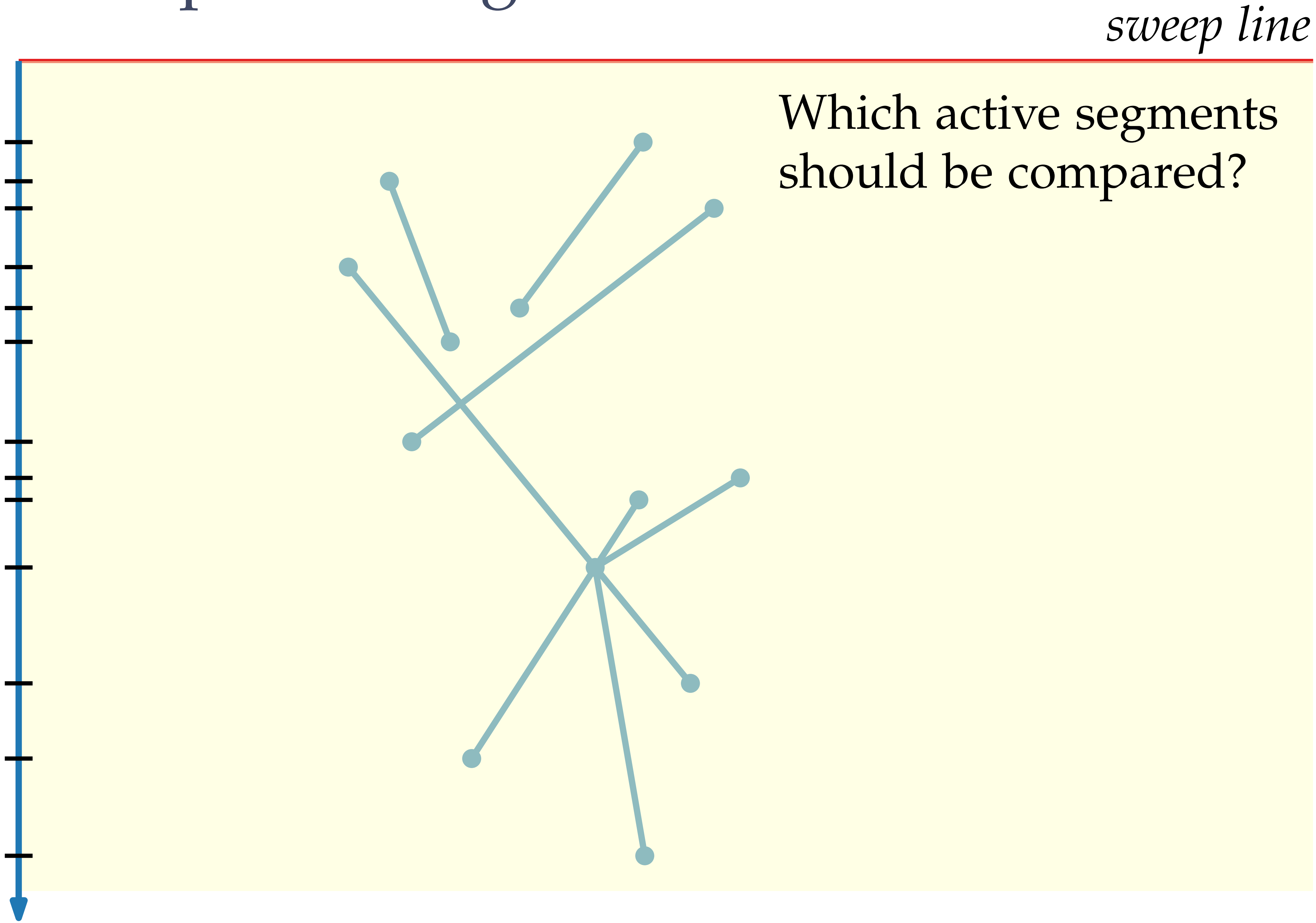
Which active segments should be compared?

Sweep-Line Algorithm



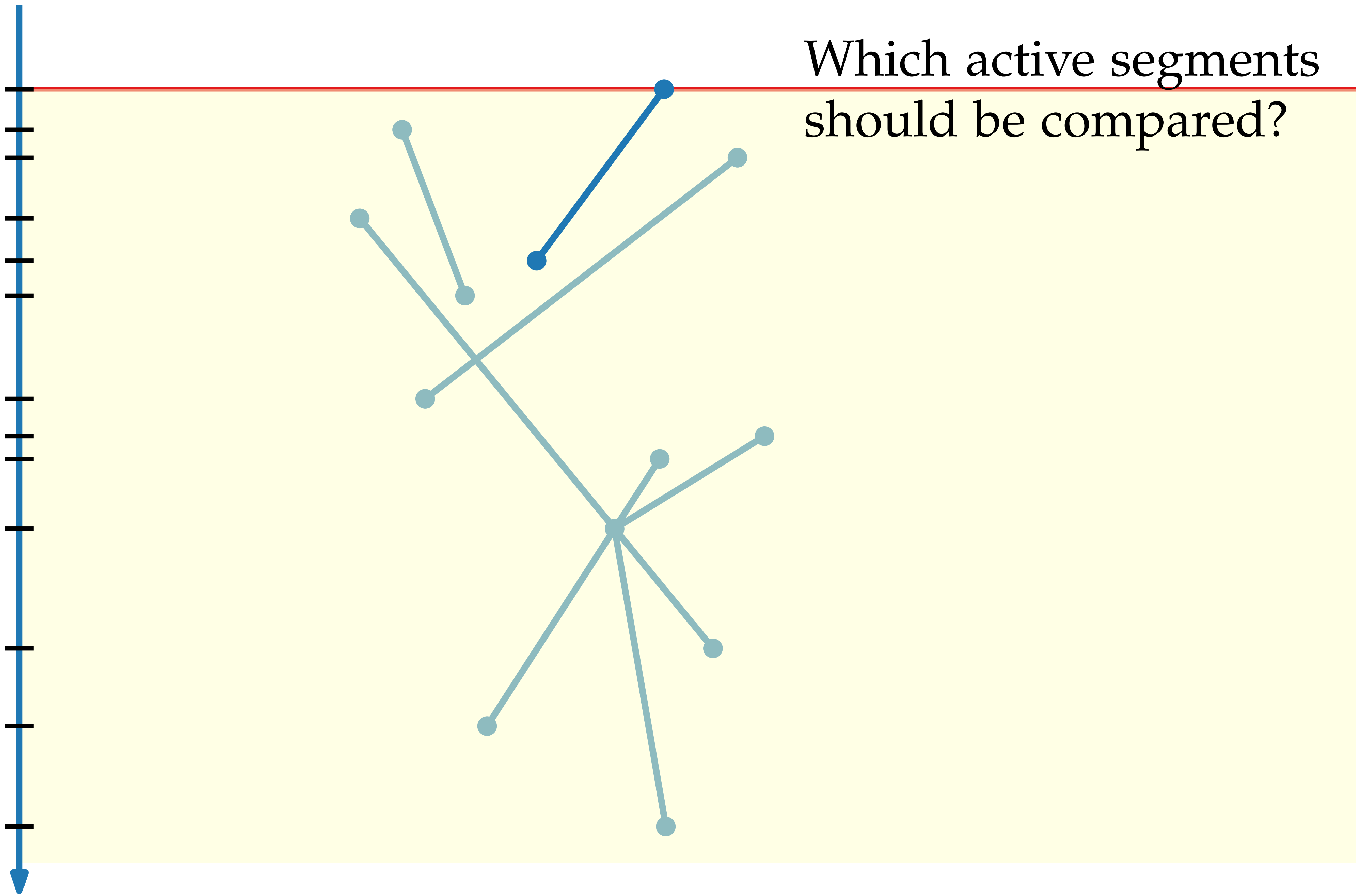
Which active segments should be compared?

Sweep-Line Algorithm



Which active segments should be compared?

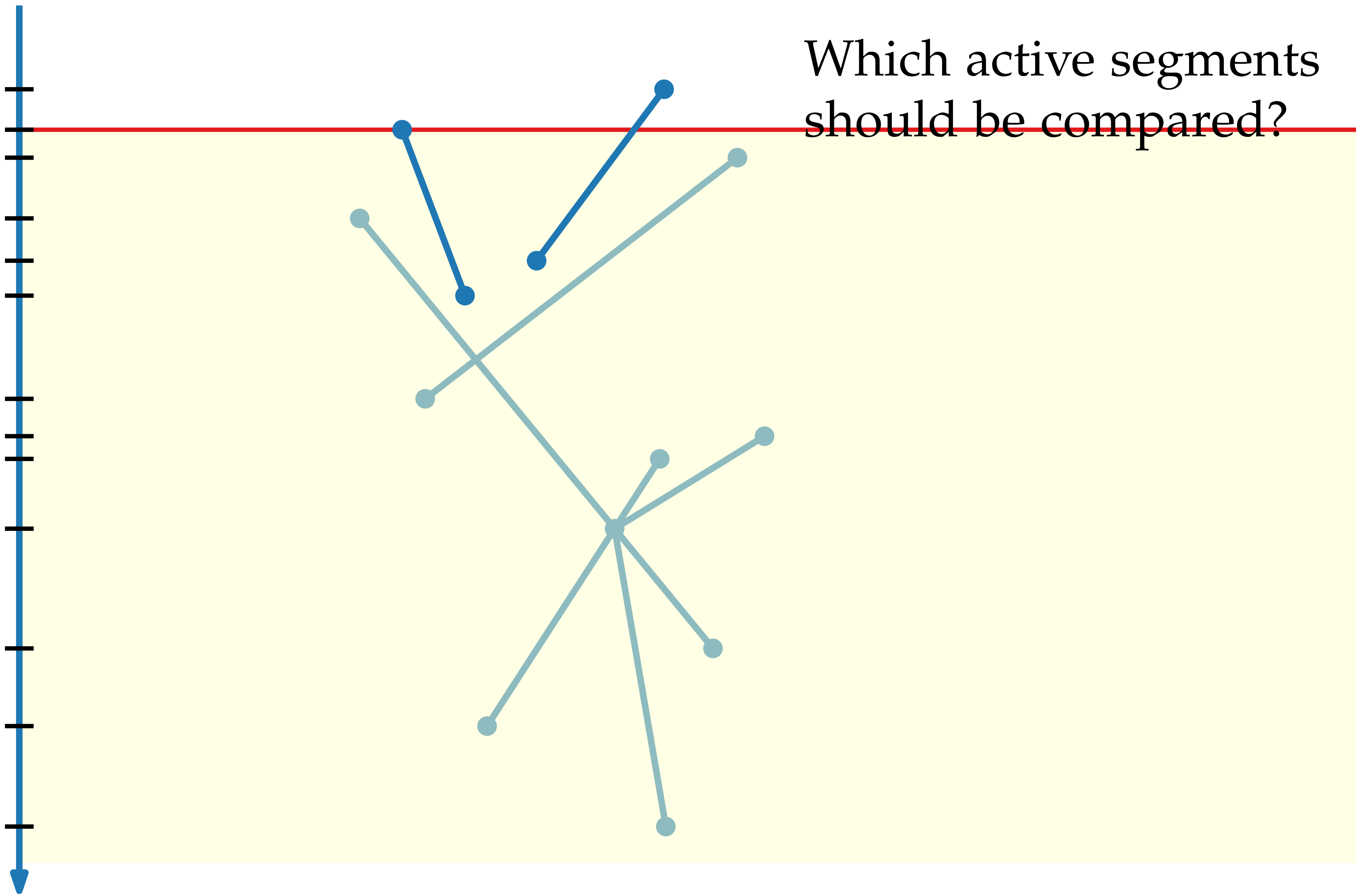
Sweep-Line Algorithm



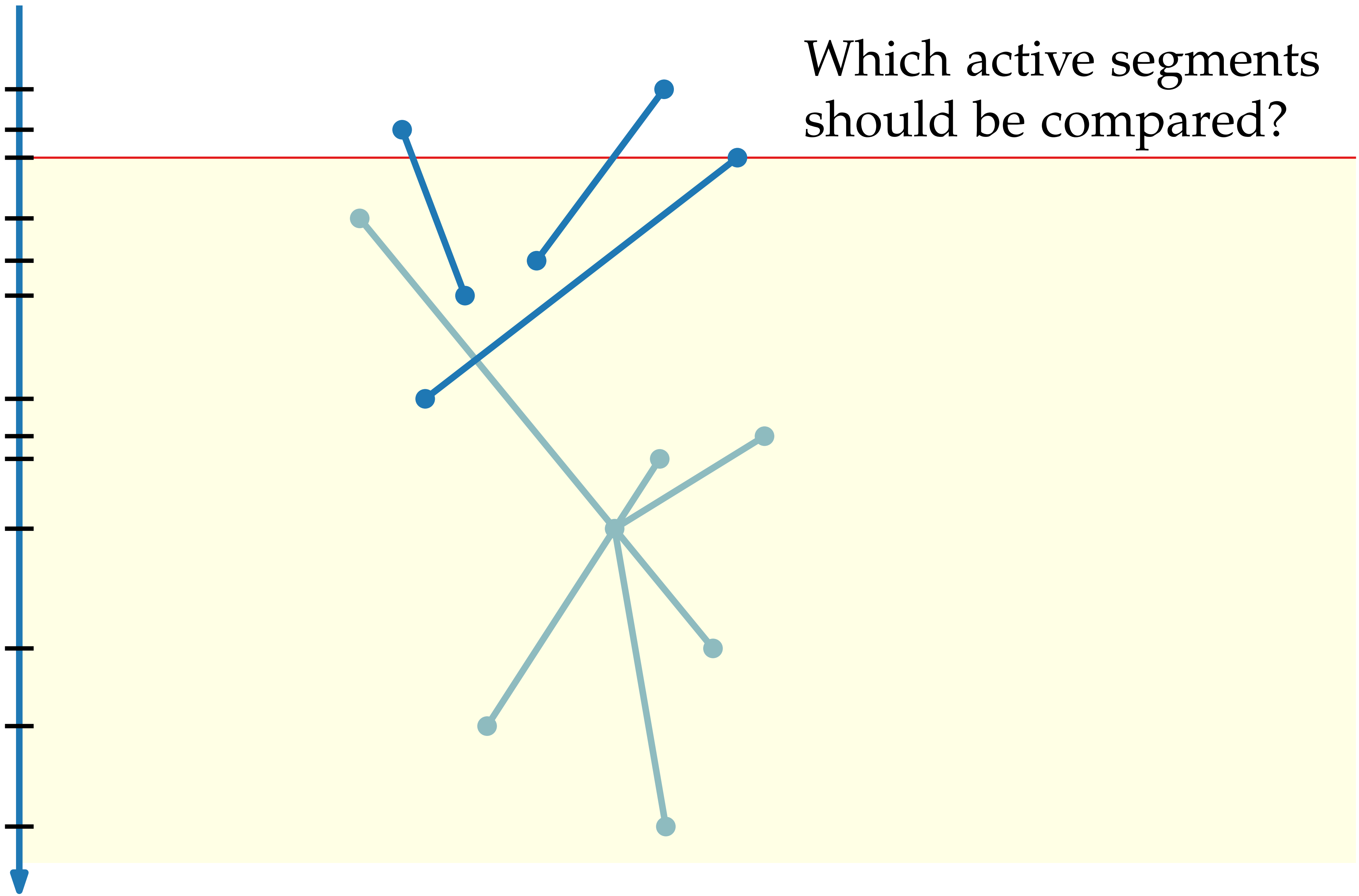
Which active segments should be compared?

Sweep-Line Algorithm

Which active segments should be compared?



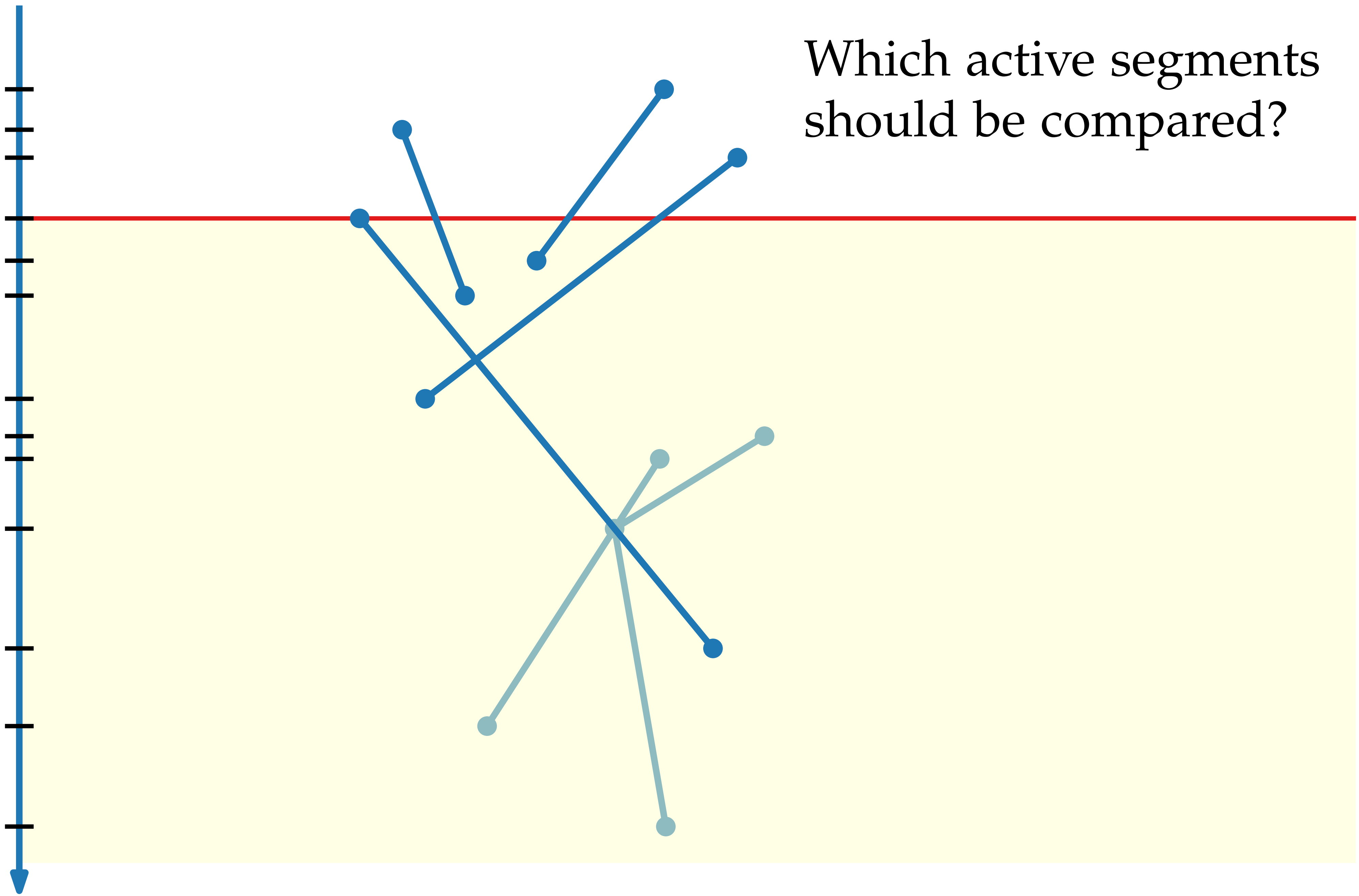
Sweep-Line Algorithm



Which active segments should be compared?

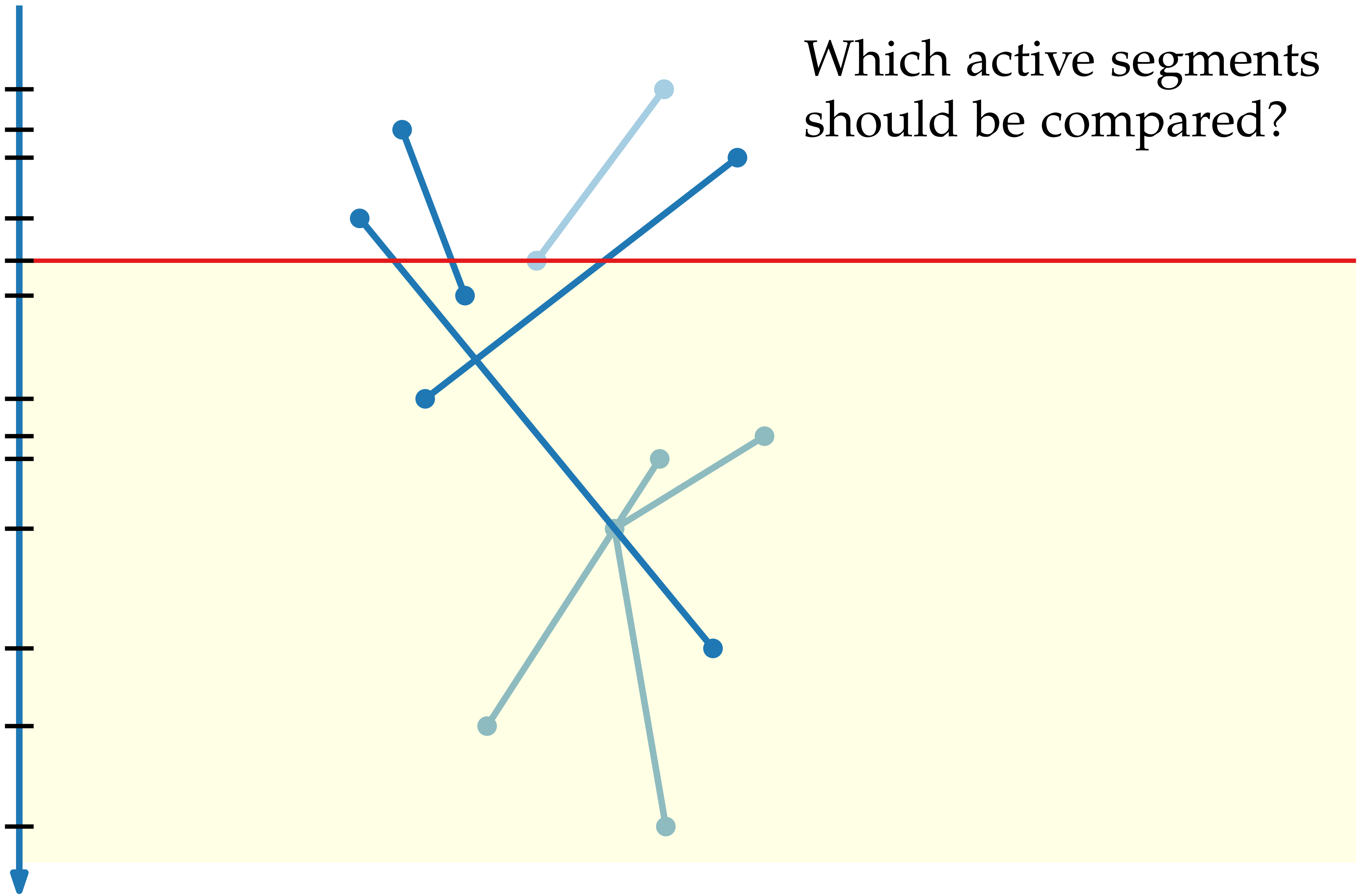
Sweep-Line Algorithm

Which active segments should be compared?



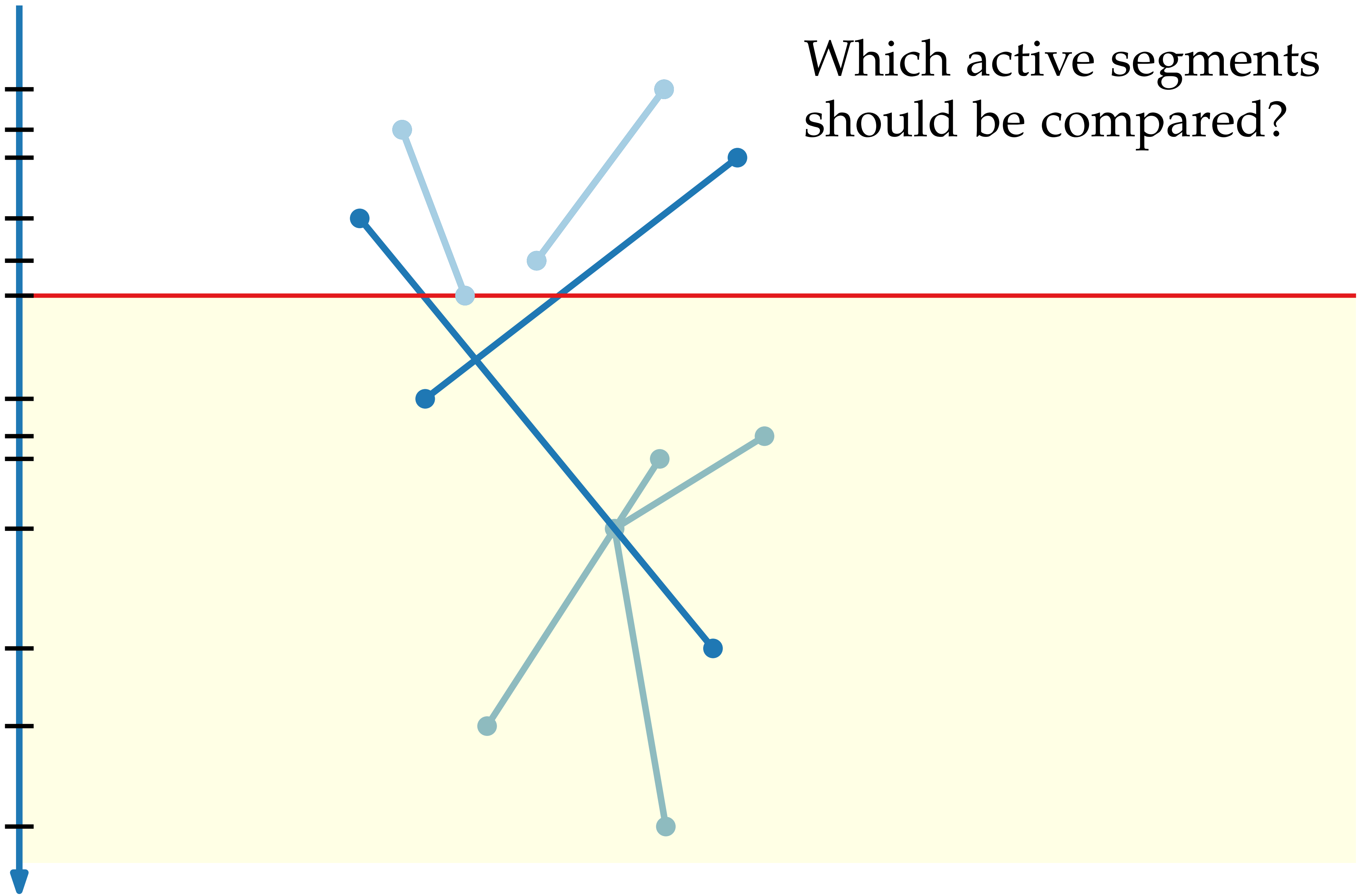
Sweep-Line Algorithm

Which active segments should be compared?

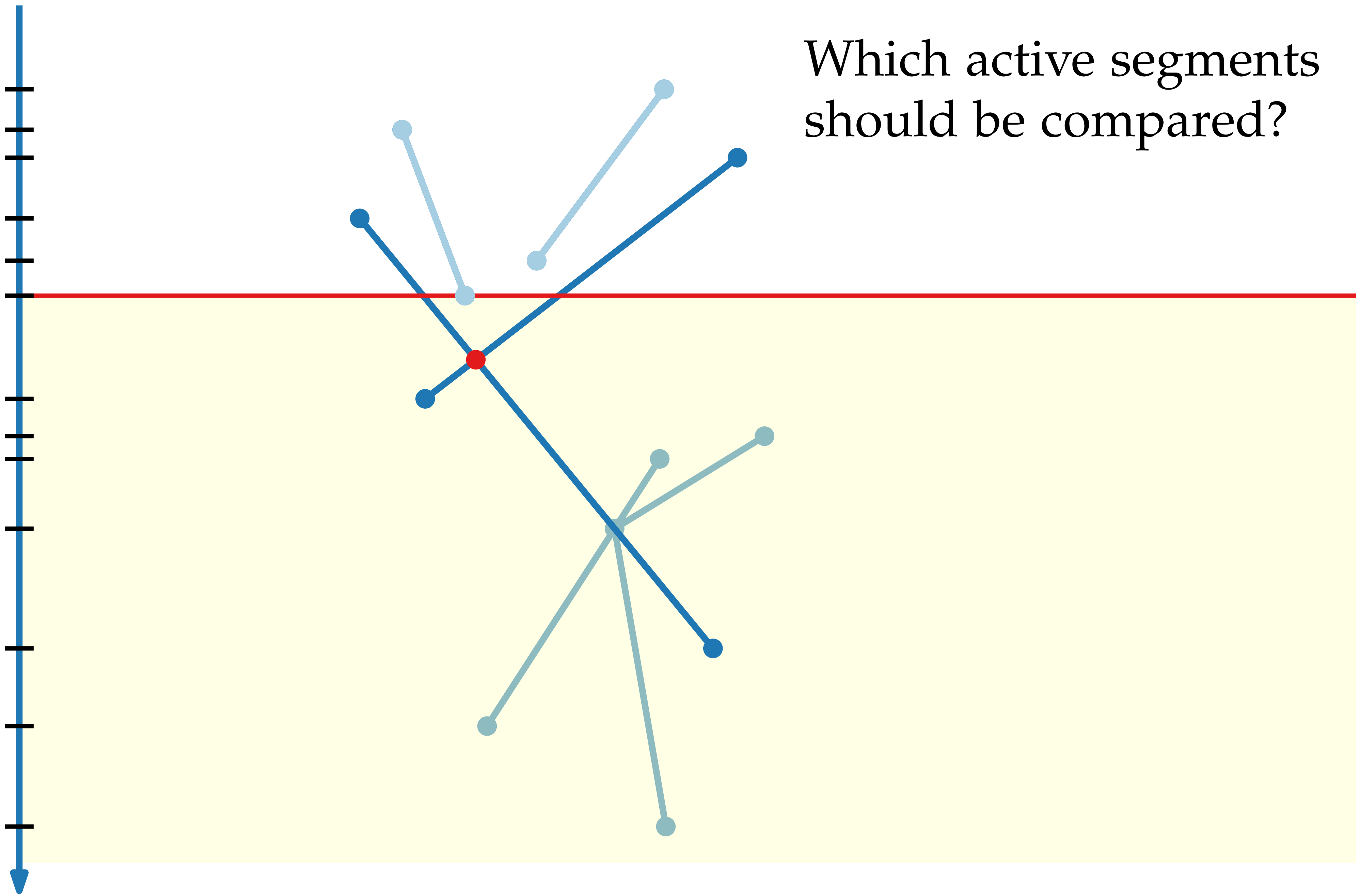


Sweep-Line Algorithm

Which active segments should be compared?

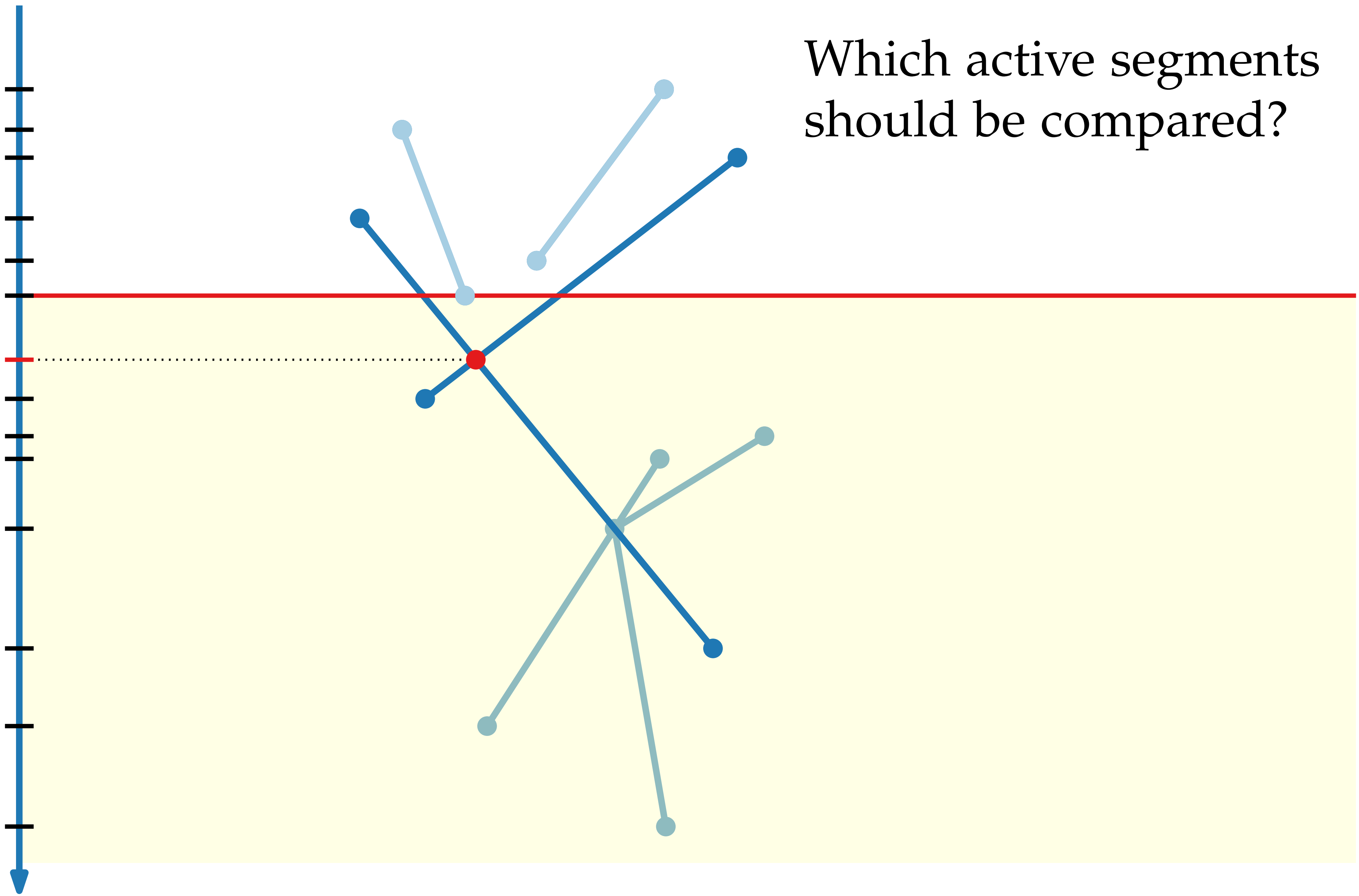


Sweep-Line Algorithm



Which active segments should be compared?

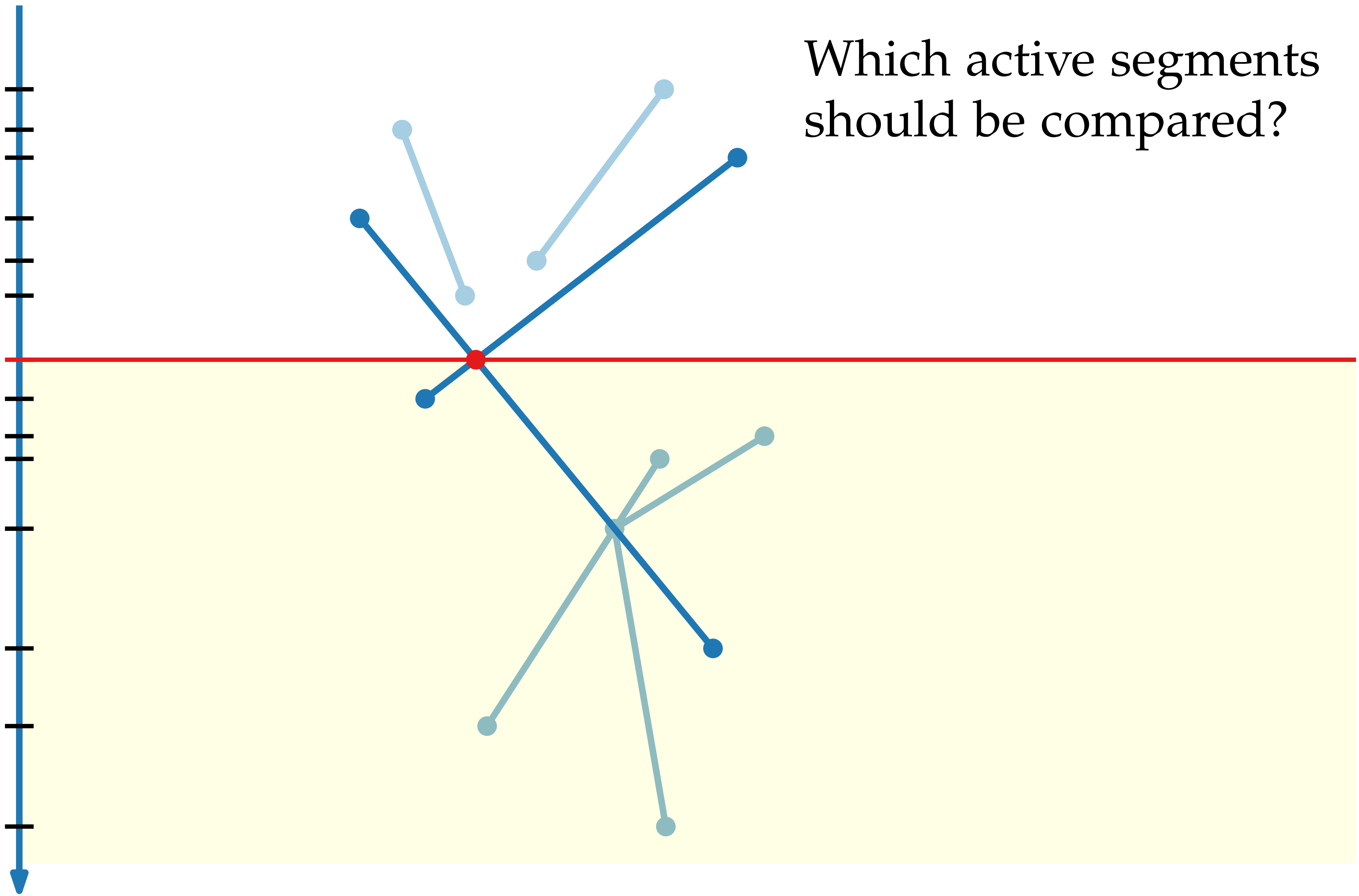
Sweep-Line Algorithm



Which active segments should be compared?

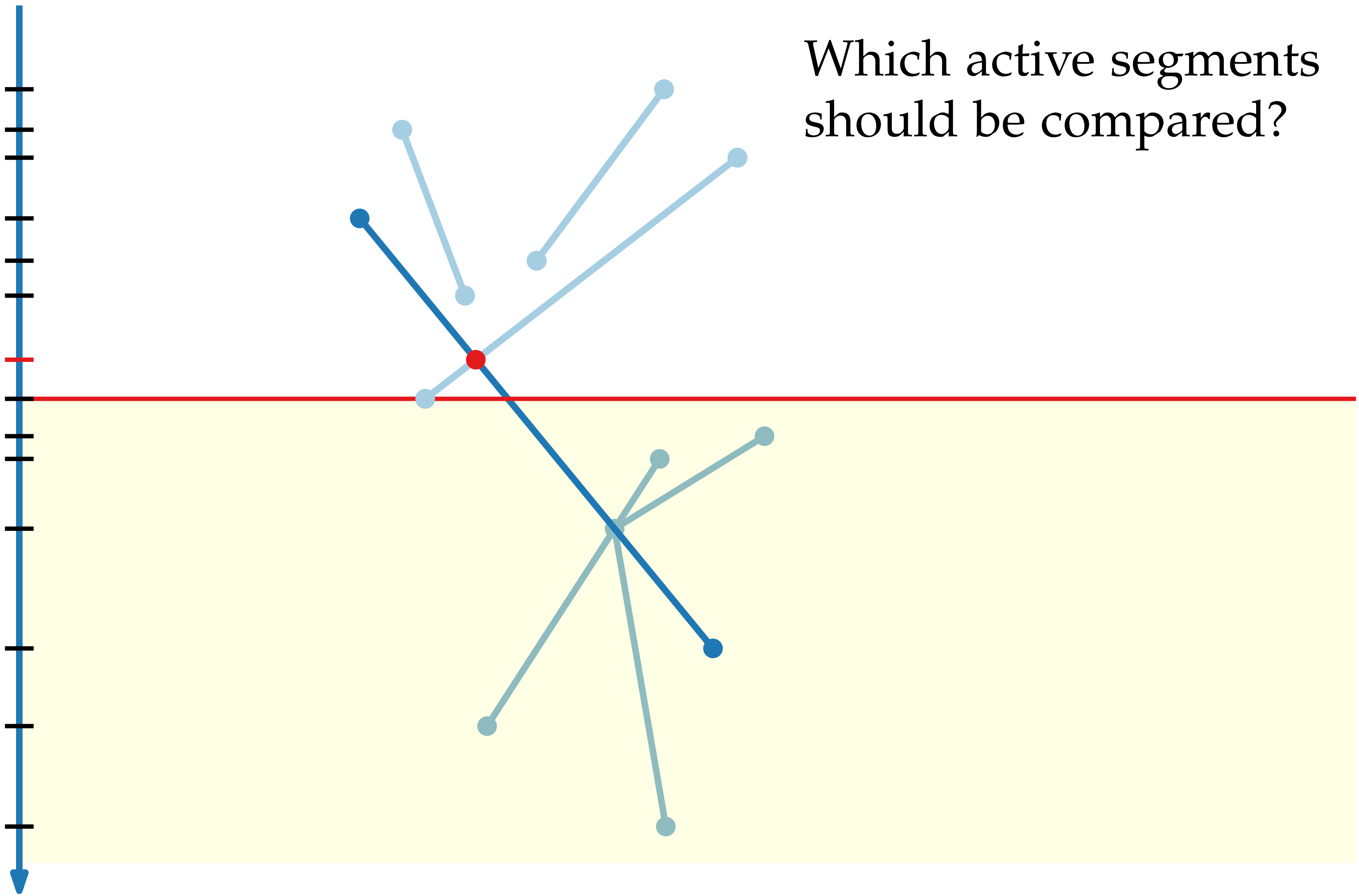
Sweep-Line Algorithm

Which active segments should be compared?



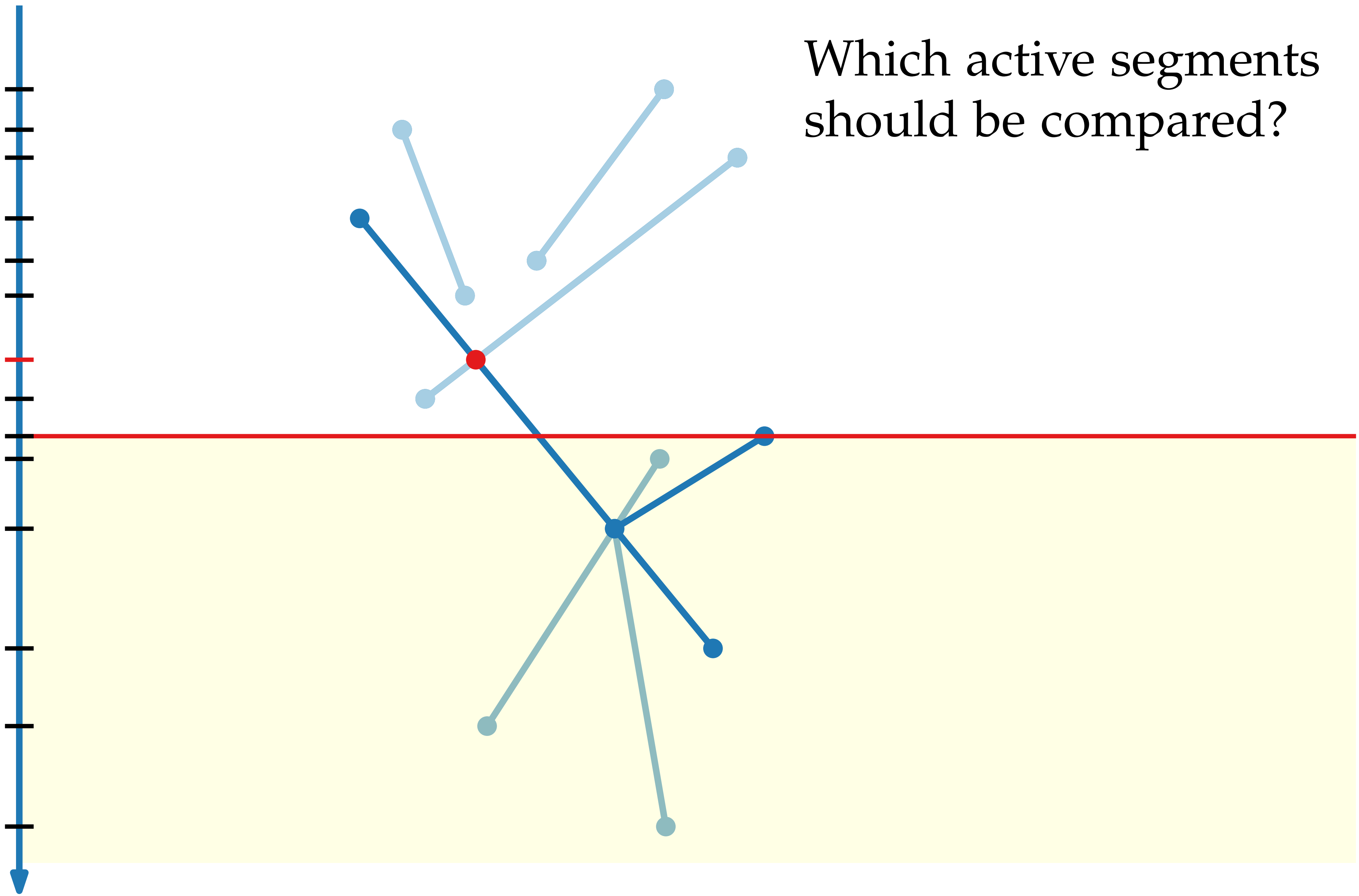
Sweep-Line Algorithm

Which active segments
should be compared?



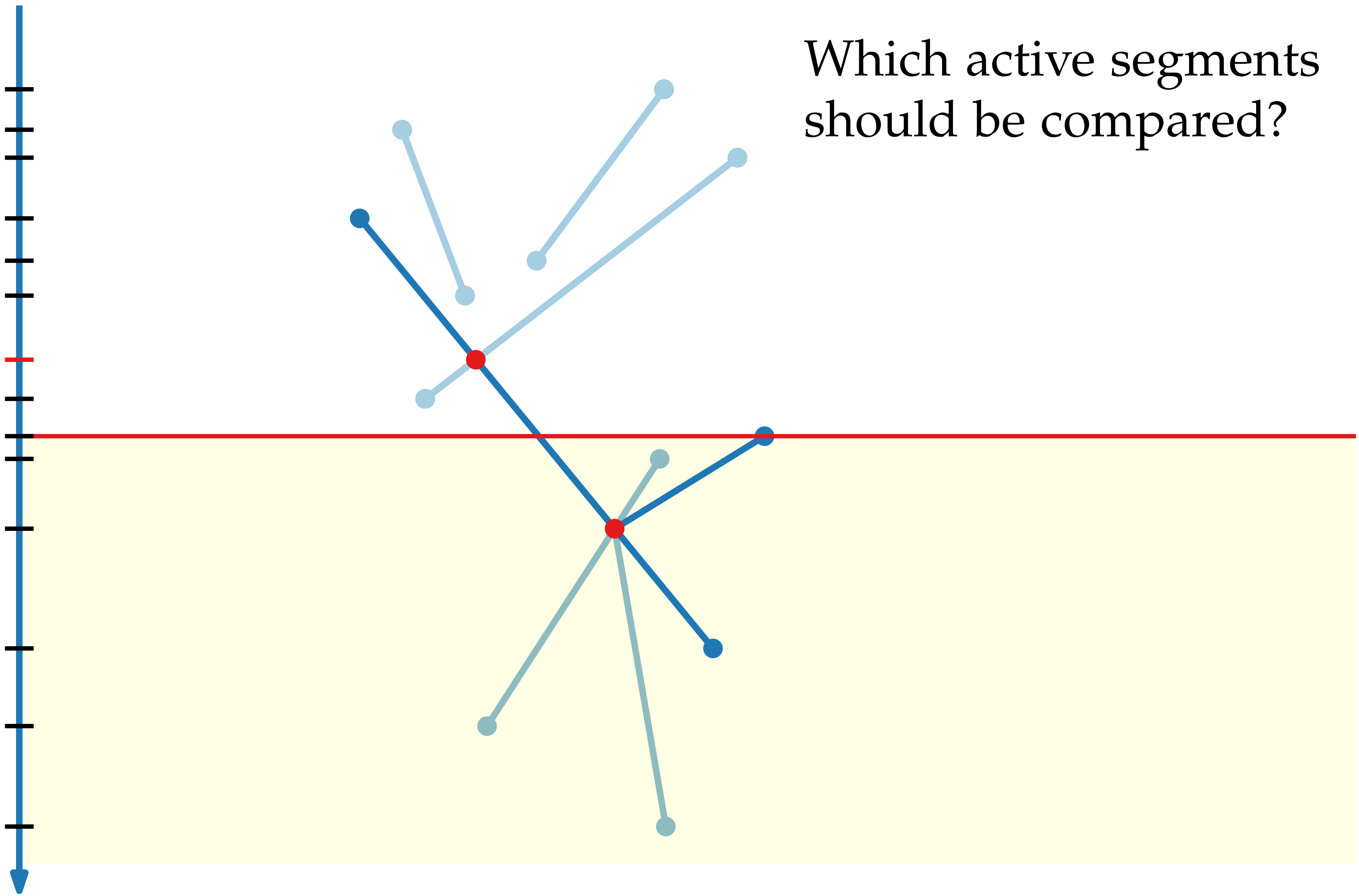
Sweep-Line Algorithm

Which active segments should be compared?



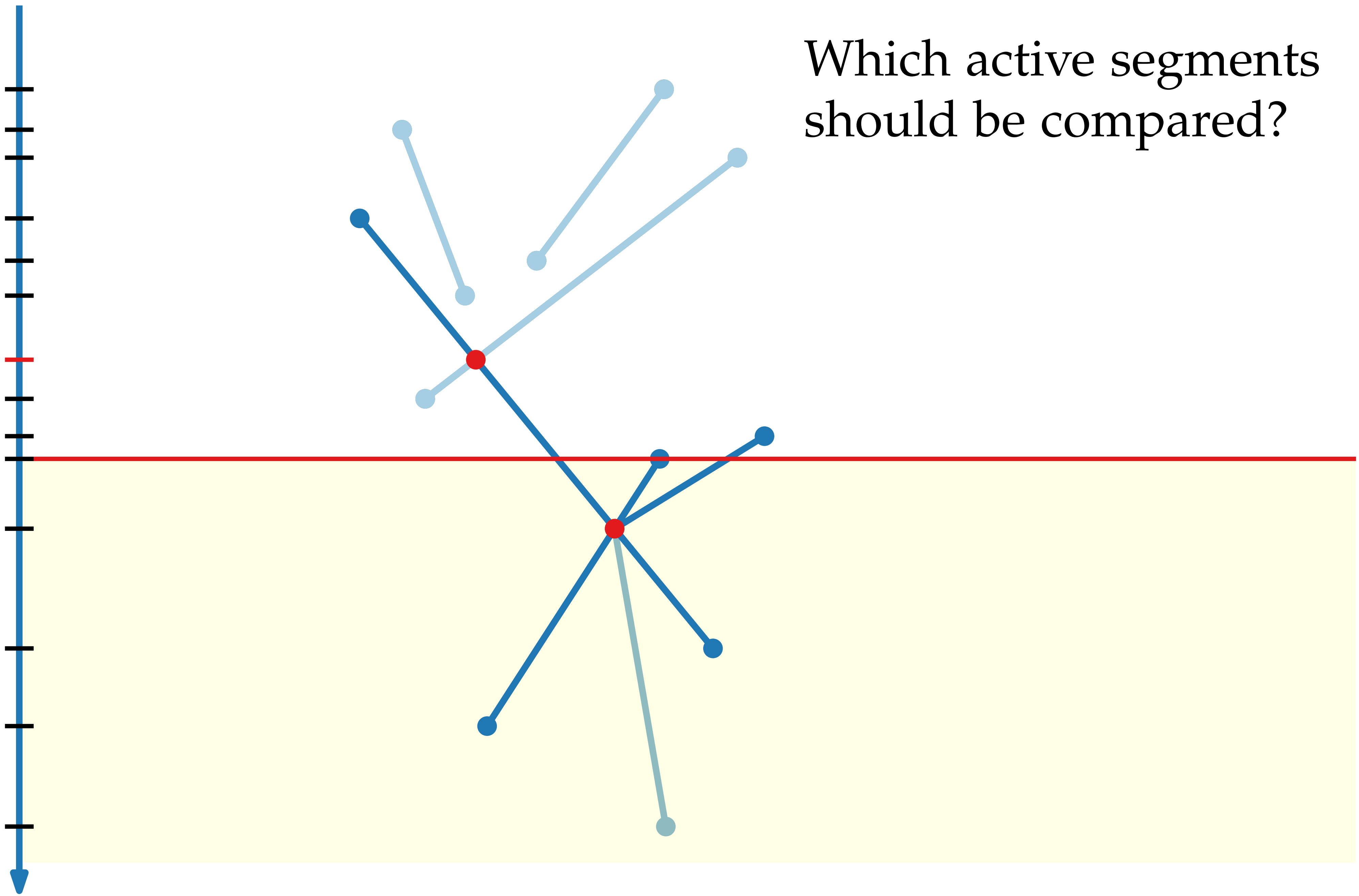
Sweep-Line Algorithm

Which active segments should be compared?



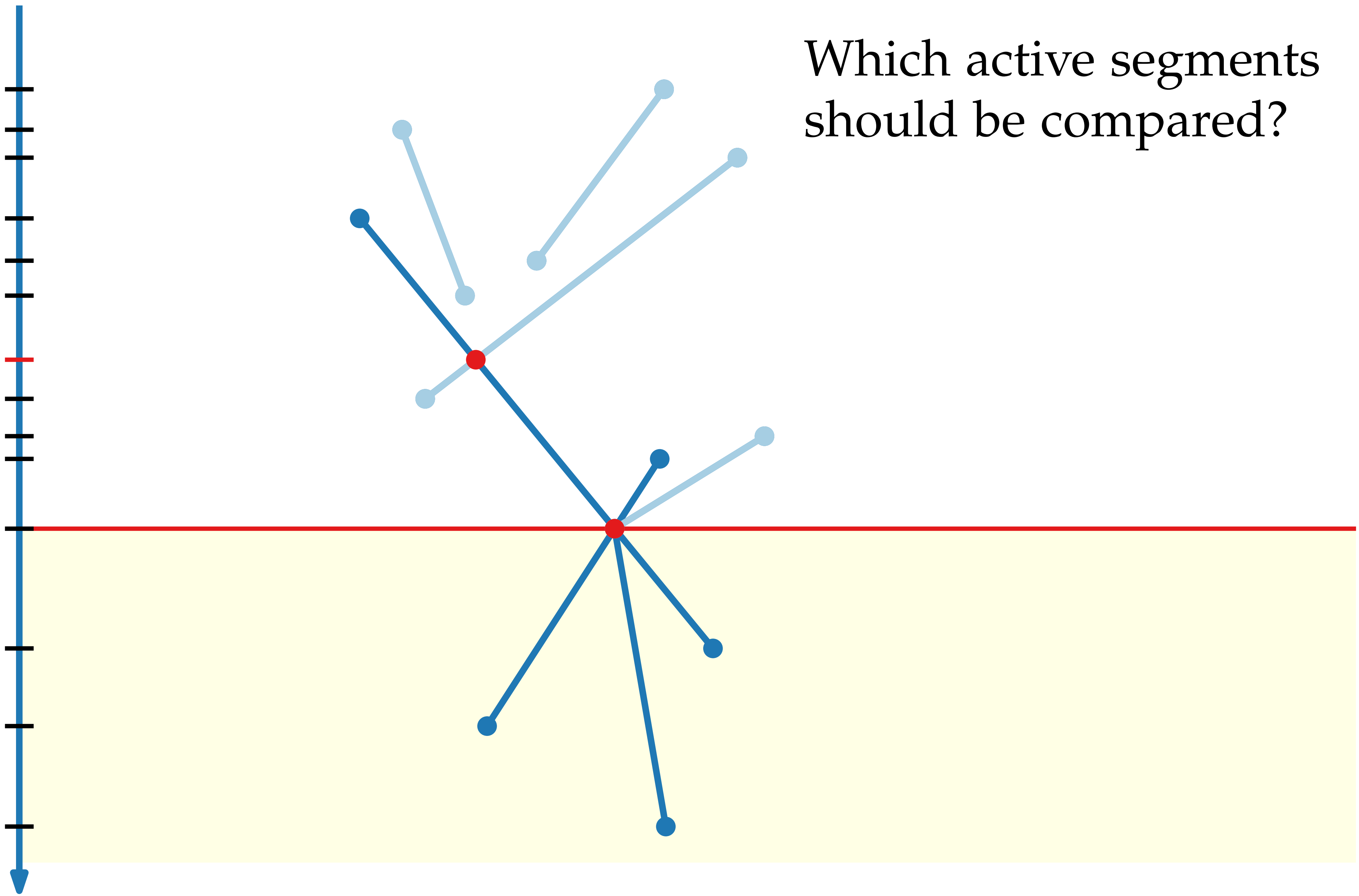
Sweep-Line Algorithm

Which active segments should be compared?



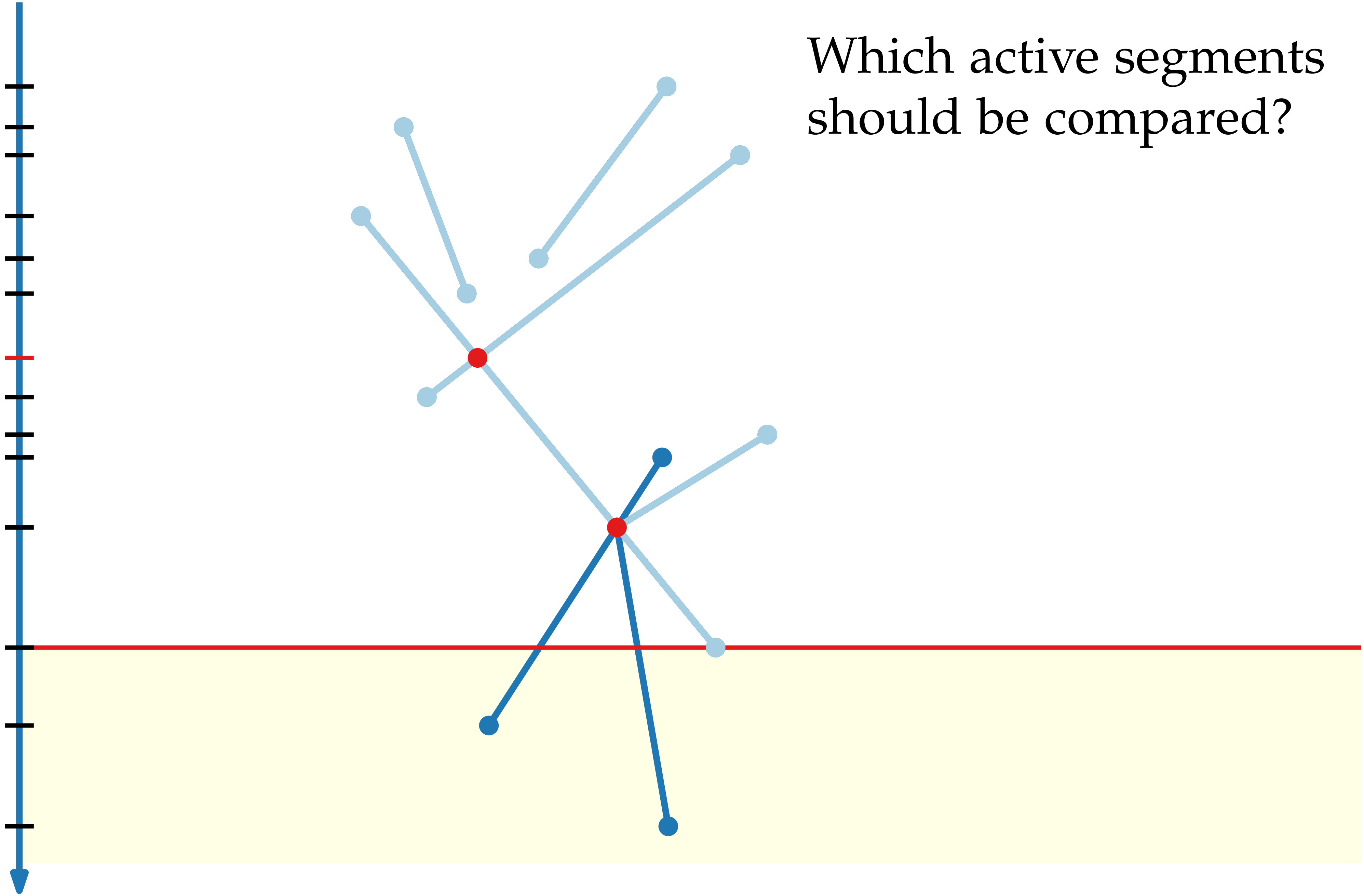
Sweep-Line Algorithm

Which active segments should be compared?

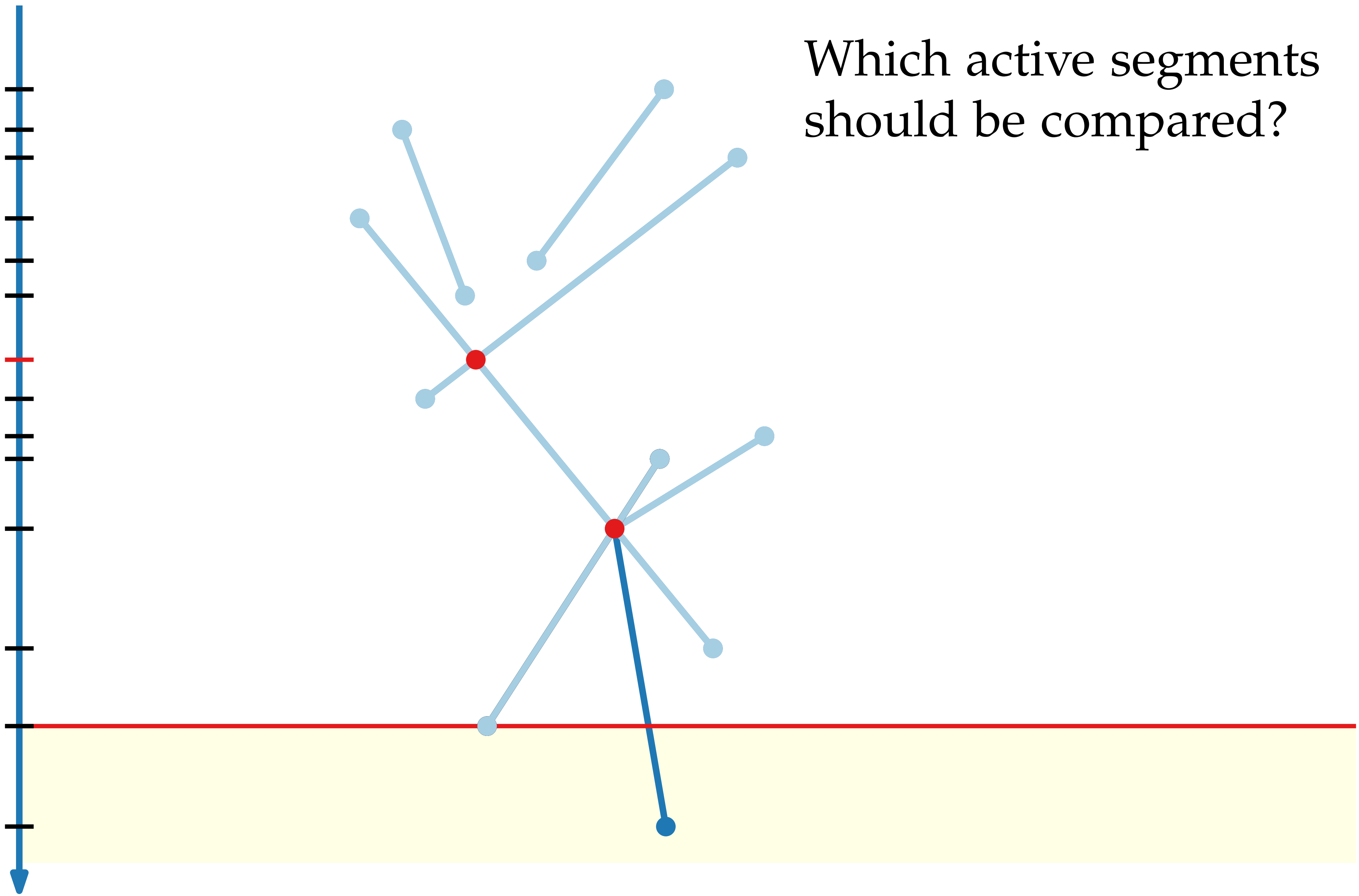


Sweep-Line Algorithm

Which active segments should be compared?

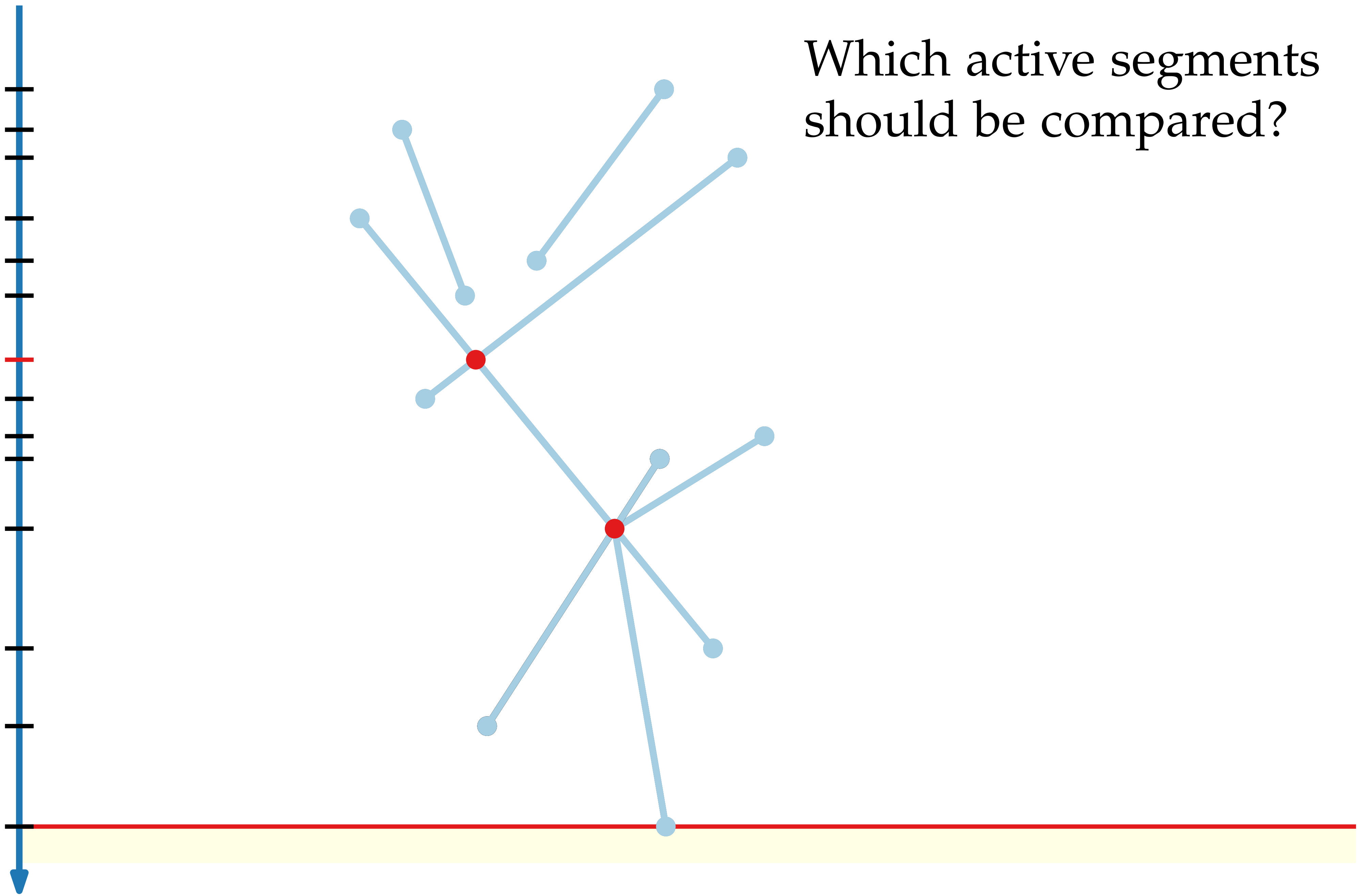


Sweep-Line Algorithm



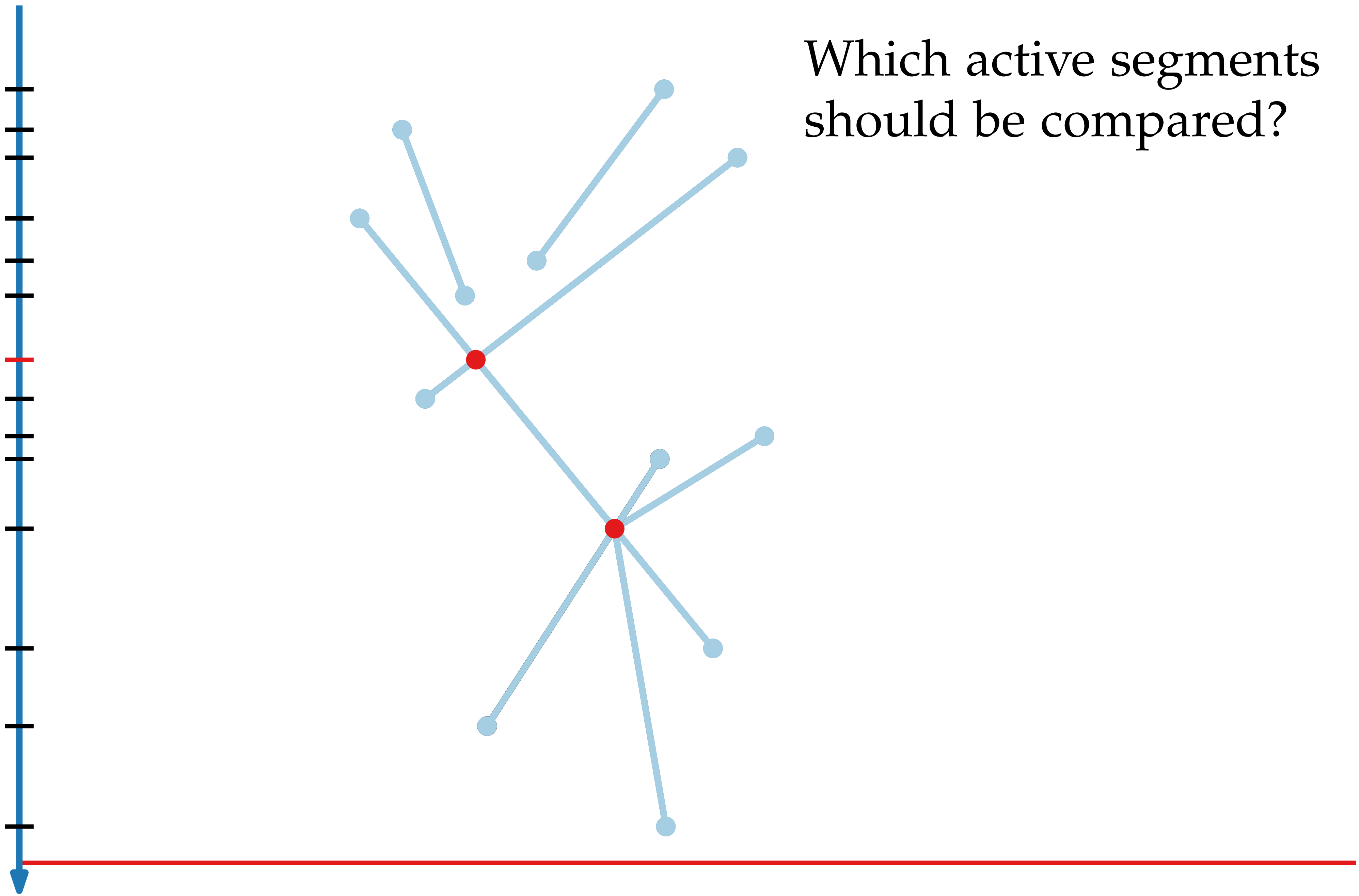
Which active segments should be compared?

Sweep-Line Algorithm



Which active segments should be compared?

Sweep-Line Algorithm



Which active segments should be compared?

Data Structures

1) event (-point) queue $\mathcal{Q}$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}}$$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q$$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q$$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

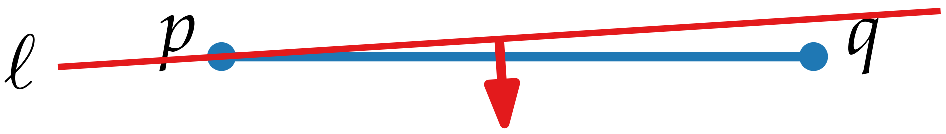
$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

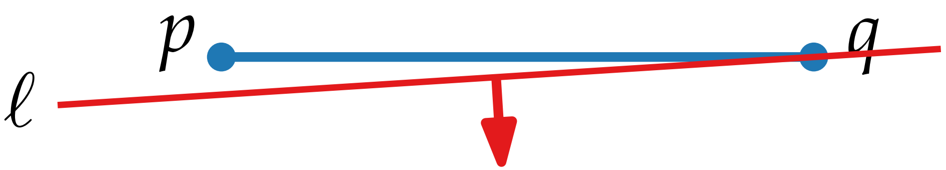


2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$

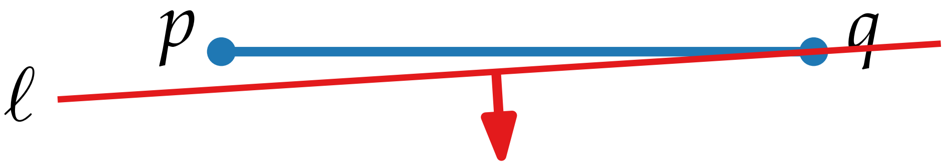


2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



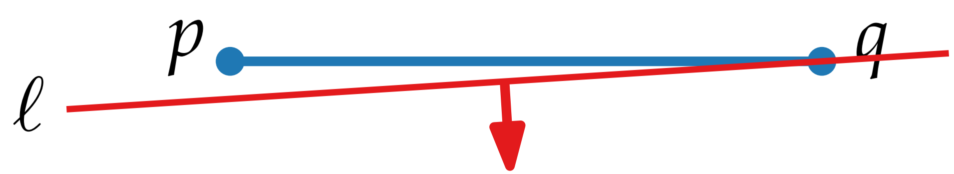
Store event pts in *balanced binary search tree* acc. to $\prec$

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



Store event pts in *balanced binary search tree* acc. to $\prec$

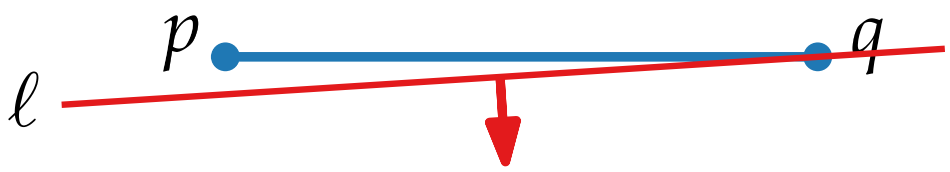
$\Rightarrow$ nextEvent() and del/insEvent() take $O(\log|\mathcal{Q}|)$ time

2) (sweep-line) status $\mathcal{T}$

Data Structures

1) event (-point) queue $\mathcal{Q}$

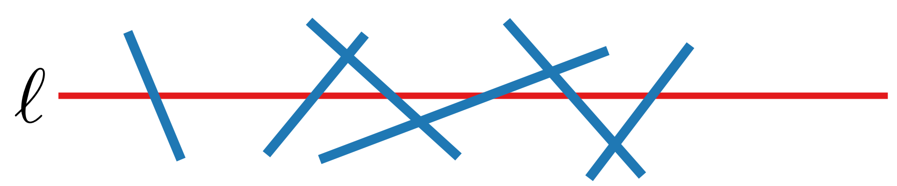
$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



Store event pts in *balanced binary search tree* acc. to $\prec$

$\Rightarrow$ nextEvent() and del/insEvent() take $O(\log |\mathcal{Q}|)$ time

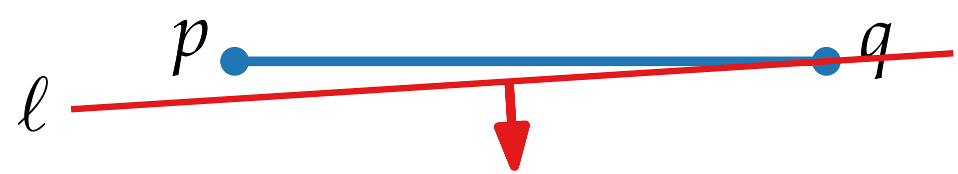
2) (sweep-line) status $\mathcal{T}$



Data Structures

1) event (-point) queue $\mathcal{Q}$

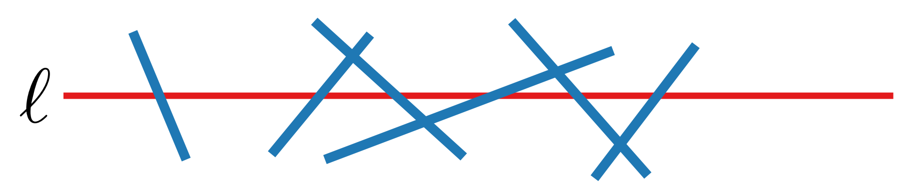
$$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$$



Store event pts in *balanced binary search tree* acc. to $\prec$

$\Rightarrow$ nextEvent() and del/insEvent() take $O(\log |\mathcal{Q}|)$ time

2) (sweep-line) status $\mathcal{T}$

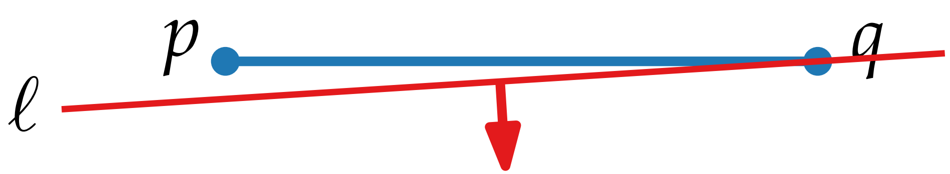


Store the segments intersected by ℓ in left-to-right order.

Data Structures

1) event (-point) queue $\mathcal{Q}$

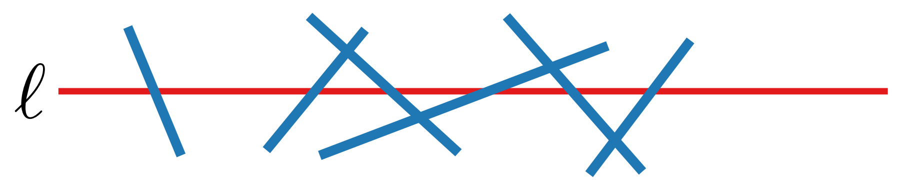
$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$



Store event pts in *balanced binary search tree* acc. to $\prec$

$\Rightarrow$ nextEvent() and del/insEvent() take $O(\log |\mathcal{Q}|)$ time

2) (sweep-line) status $\mathcal{T}$



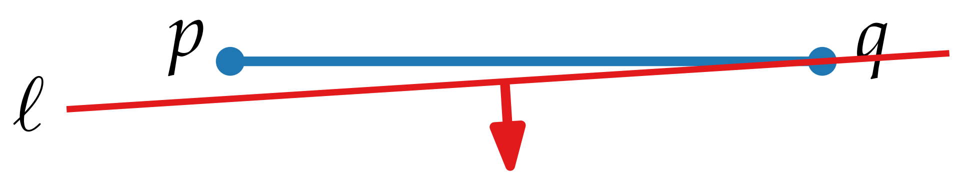
Store the segments intersected by ℓ in left-to-right order.

How?

Data Structures

1) event (-point) queue $\mathcal{Q}$

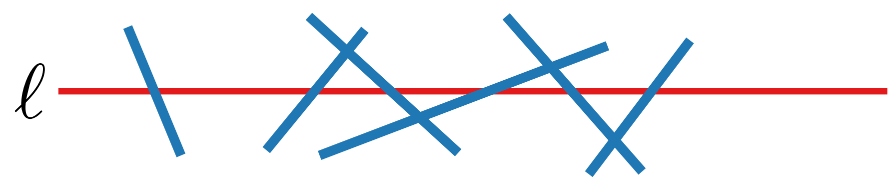
$p \prec q \quad \Leftrightarrow_{\text{def.}} \quad y_p > y_q \quad \text{or} \quad (y_p = y_q \text{ and } x_p < x_q)$



Store event pts in *balanced binary search tree* acc. to $\prec$

$\Rightarrow$ nextEvent() and del/insEvent() take $O(\log |\mathcal{Q}|)$ time

2) (sweep-line) status $\mathcal{T}$



Store the segments intersected by ℓ in left-to-right order.

How? In a balanced binary search tree!

Computational Geometry

Lecture 2:
Line-Segment Intersection
or
Map Overlay

Part III:
Algorithmic Details

Philipp Kindermann

Winter Semester 2020

Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts

Pseudo-code

findIntersections(*S*)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

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

Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels

Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

```
Q ← ∅;  T ← ⟨vertical lines at x = −∞ and x = +∞⟩   // sentinels
foreach s ∈ S do                                   // initialize event queue Q
```

Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

```
Q ← ∅;  T ← ⟨vertical lines at x = −∞ and x = +∞⟩   // sentinels
foreach s ∈ S do                                     // initialize event queue Q
    foreach endpoint p of s do
```

Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output:
- set I of intersection pts
- for each $p \in I$ every $s \in S$ with $p \in s$

```
Q ← ∅;  T ← ⟨ vertical lines at x = −∞ and x = +∞ ⟩   // sentinels
foreach s ∈ S do                                       // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p);
```

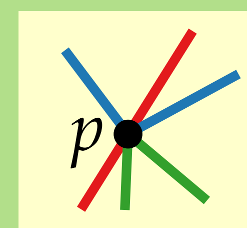
Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels
foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$
  foreach endpoint p of s **do**
    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p);



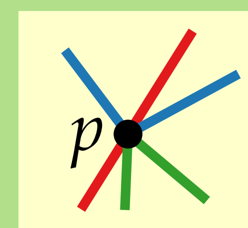
Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels
foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$
  foreach endpoint p of s **do**
    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$



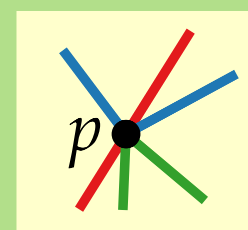
Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels
foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$
  foreach endpoint p of s **do**
    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$
    if p lower endpt of s **then** $L(p)$.append(s)



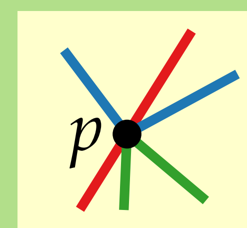
Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels
foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$
  foreach endpoint p of s **do**
    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$
    if p lower endpt of s **then** $L(p)$.append(s)
    if p upper endpt of s **then** $U(p)$.append(s)



Pseudo-code

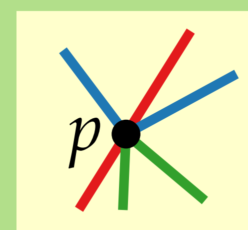
findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

```
Q ← ∅;  T ← ⟨vertical lines at x = −∞ and x = +∞⟩   // sentinels
foreach s ∈ S do                                   // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower  endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)

while Q ≠ ∅ do
```



Pseudo-code

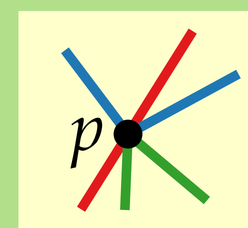
findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

```
Q ← ∅;  T ← ⟨vertical lines at x = −∞ and x = +∞⟩   // sentinels
foreach s ∈ S do                                    // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)

while Q ≠ ∅ do
    p ← Q.nextEvent()
```



Pseudo-code

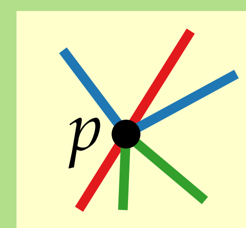
findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

```
Q ← ∅;  T ← ⟨vertical lines at x = −∞ and x = +∞⟩   // sentinels
foreach s ∈ S do                                   // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)

while Q ≠ ∅ do
    p ← Q.nextEvent()
    Q.deleteEvent(p)
```



Pseudo-code

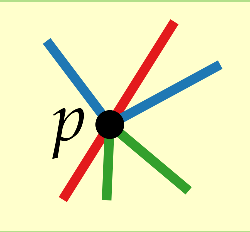
findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output:
– set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

```
Q ← ∅;  T ← ⟨ vertical lines at x = −∞ and x = +∞ ⟩   // sentinels
foreach s ∈ S do                                      // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower  endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)

while Q ≠ ∅ do
    p ← Q.nextEvent()
    Q.deleteEvent(p)
    handleEvent(p)
```



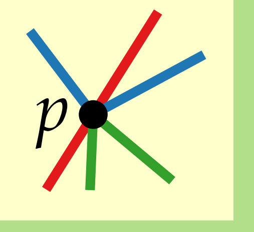
Pseudo-code

findIntersections(S)

Input: set S of n non-overlapping closed line segments

Output: – set I of intersection pts
– for each $p \in I$ every $s \in S$ with $p \in s$

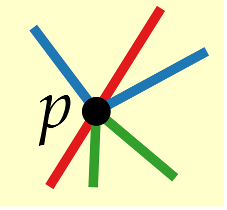
$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels
foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$
  foreach endpoint p of s **do**
    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$
    if p lower endpt of s **then** $L(p)$.append(s)
    if p upper endpt of s **then** $U(p)$.append(s)



while $\mathcal{Q} \neq \emptyset$ **do**
  $p \leftarrow \mathcal{Q}$.nextEvent()
  $\mathcal{Q}$.deleteEvent(p)
  handleEvent(p)

This subroutine does the real work.
How would you implement it?

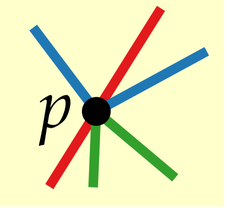
Handling an Event



$C(p), L(p), U(p)$

handleEvent(event p)

Handling an Event

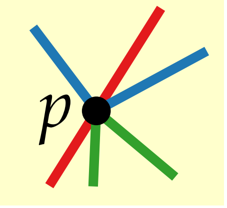


$C(p)$, $L(p)$, $U(p)$

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

Handling an Event



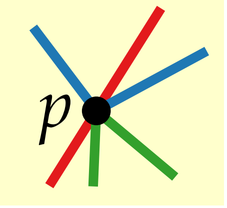
$C(p), L(p), U(p)$

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$

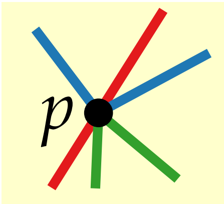
Handling an Event



$C(p), L(p), U(p)$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

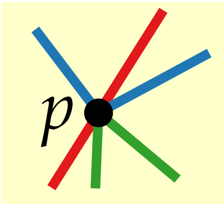
Handling an Event



$C(p)$, $L(p)$, $U(p)$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

Handling an Event

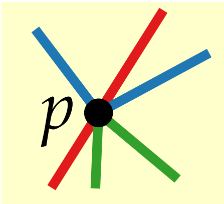


$C(p)$, $L(p)$, $U(p)$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**

else

Handling an Event



$C(p)$, $L(p)$, $U(p)$

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

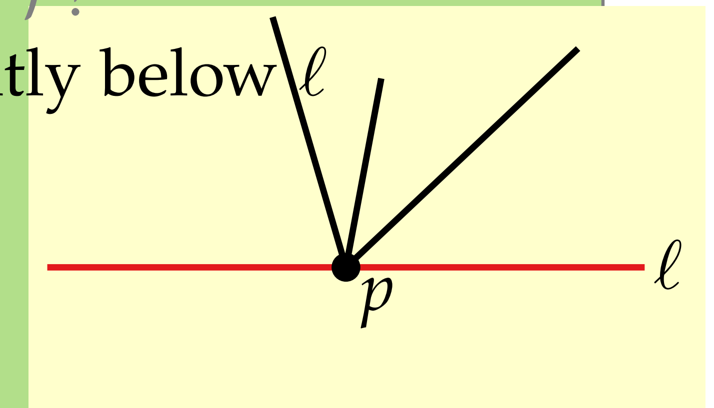
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$

delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

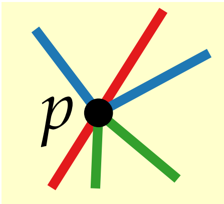
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

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

else



Handling an Event



$C(p)$, $L(p)$, $U(p)$

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$

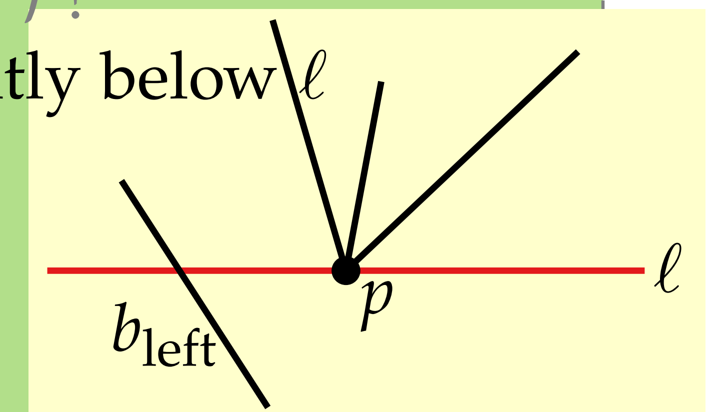
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

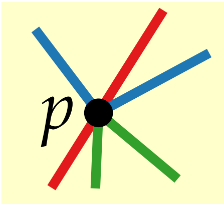
if $U(p) \cup C(p) = \emptyset$ **then**

  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$

else

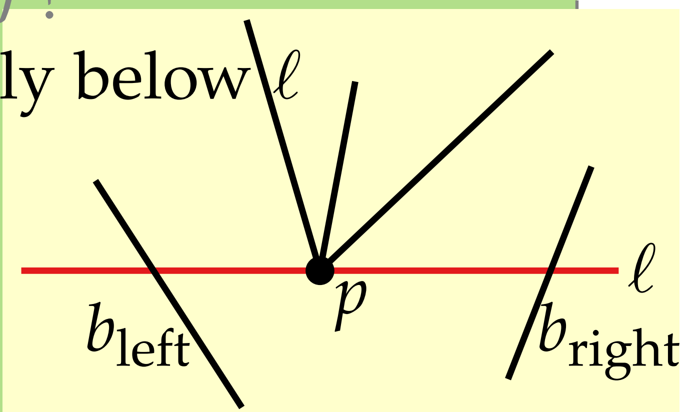


Handling an Event

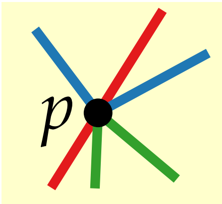


$C(p)$,$L(p)$,$U(p)$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
else

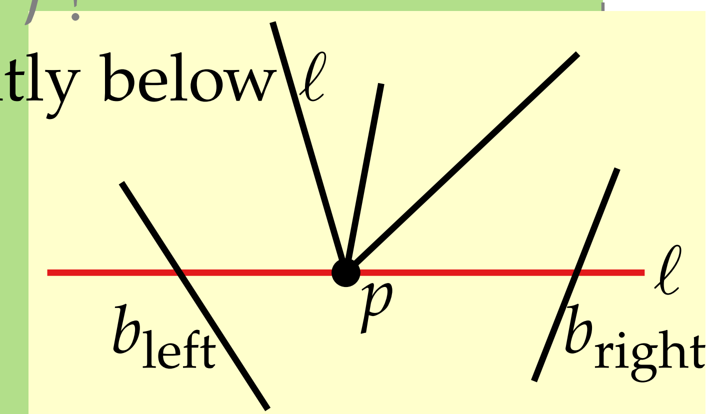


Handling an Event

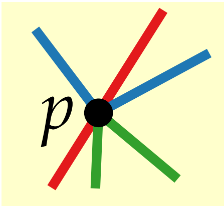


$C(p)$, $L(p)$, $U(p)$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent(b_{left}, b_{right}, p)
else



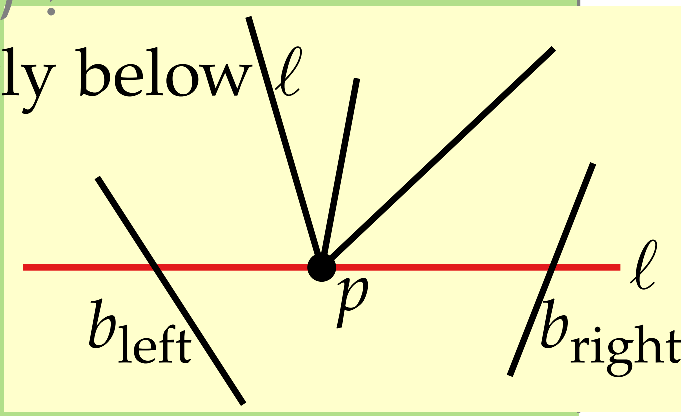
Handling an Event



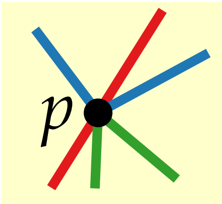
$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent(b_{left}, b_{right}, p)
else



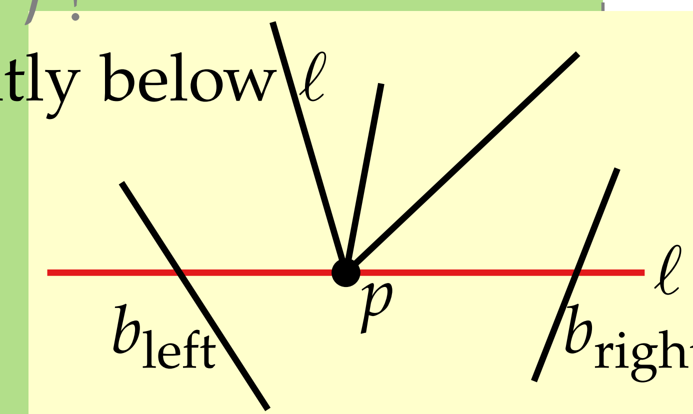
Handling an Event



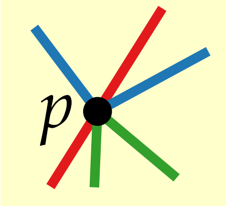
$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
 report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
 $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
 findNewEvent(b_{left}, b_{right}, p)
else



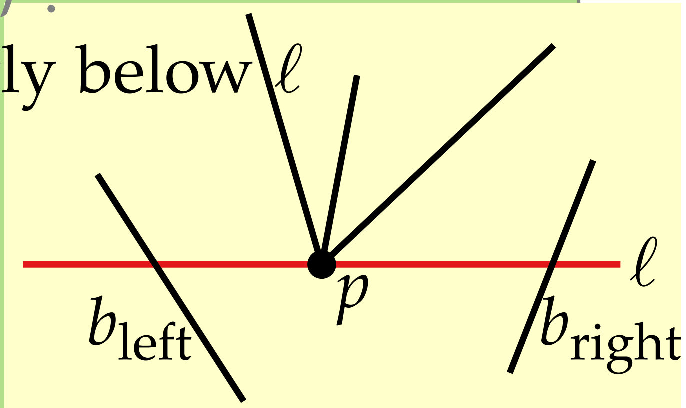
Handling an Event



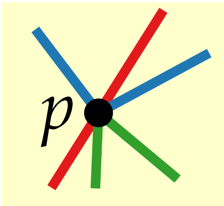
$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}} / b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent(b_{left}, b_{right}, p)
else



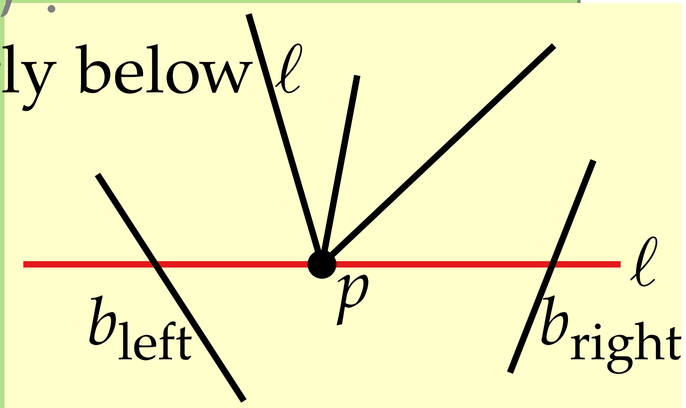
Handling an Event



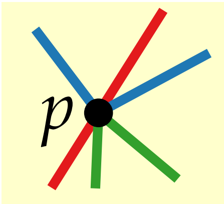
$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent(b_{left}, b_{right}, p)
else



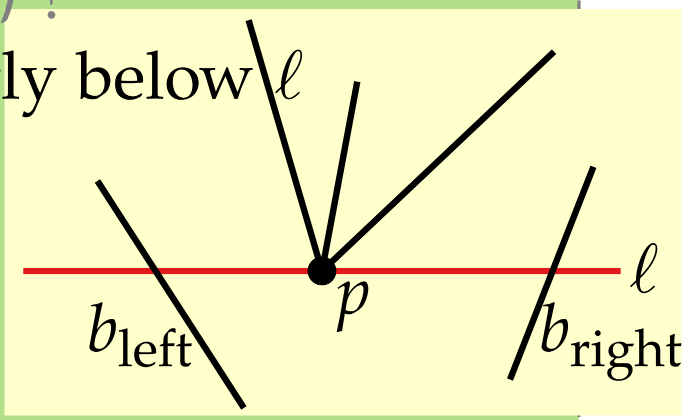
Handling an Event



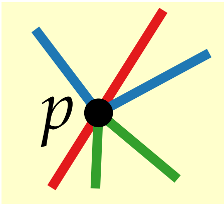
$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else



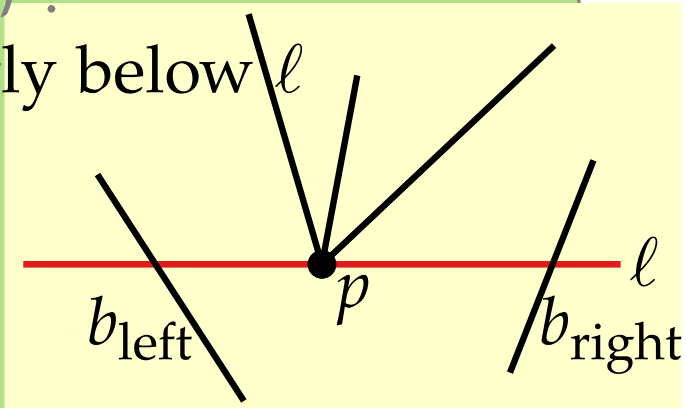
Handling an Event



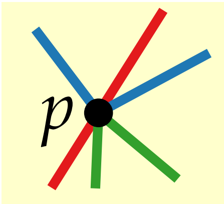
$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else



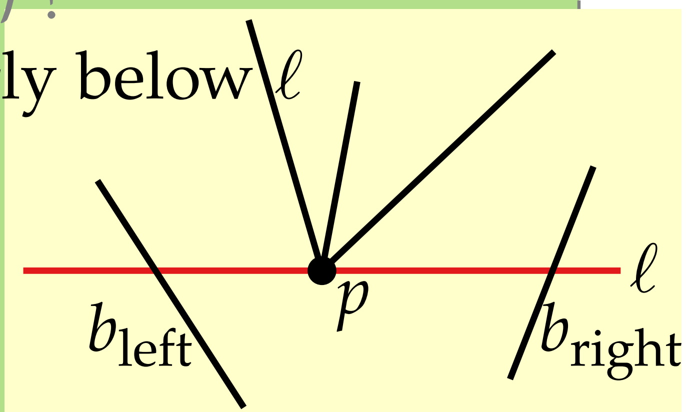
Handling an Event



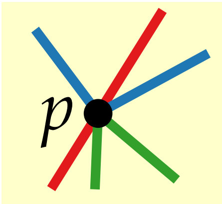
$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else



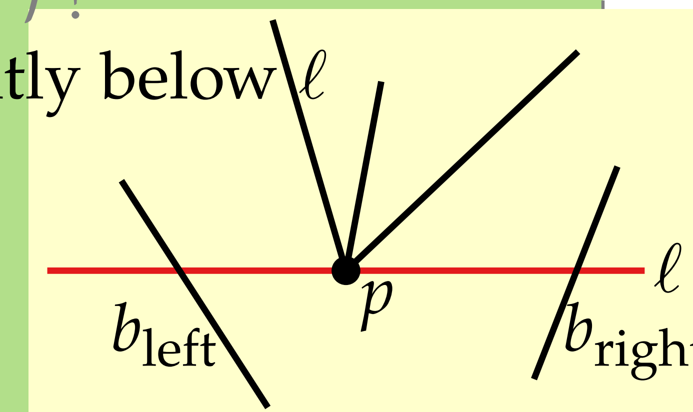
Handling an Event

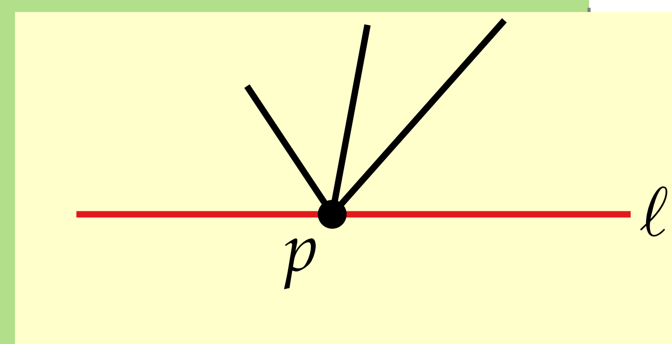


$C(p)$, $L(p)$, $U(p)$

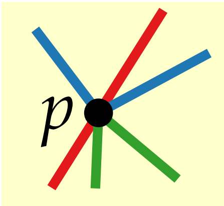
findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}} / b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else





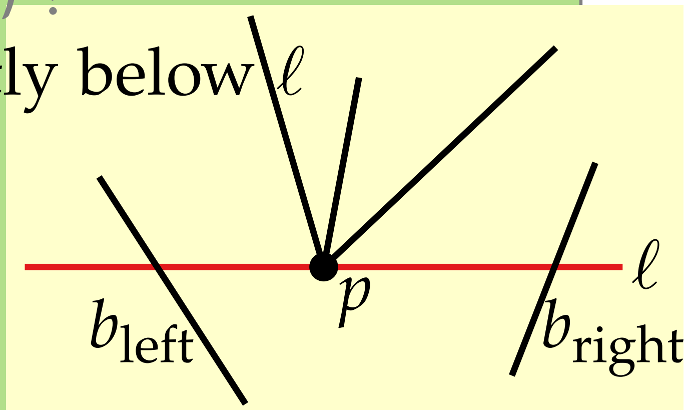
Handling an Event

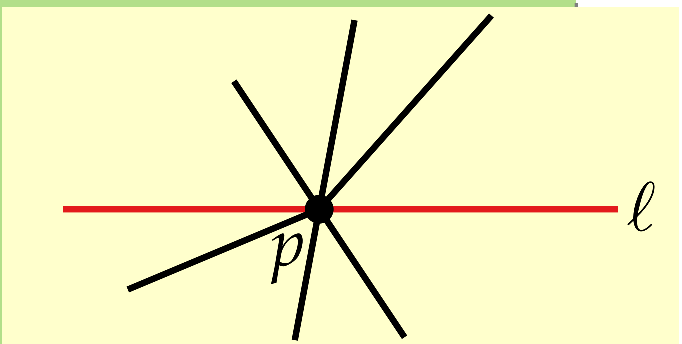


$C(p)$, $L(p)$, $U(p)$

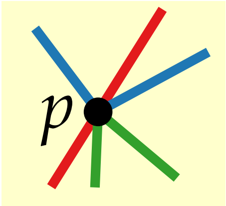
findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}.\text{add}(x)$
  if $x \in \text{rel-int}(s)$ **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in \text{rel-int}(s')$ **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else





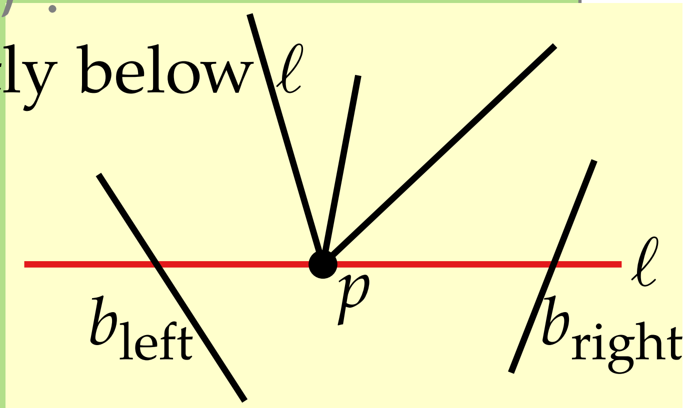
Handling an Event

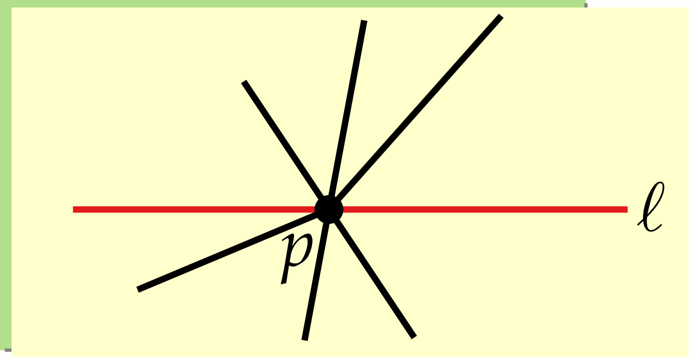


$C(p)$, $L(p)$, $U(p)$

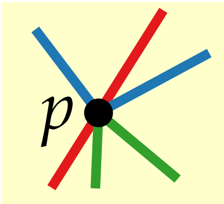
findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else
  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$





Handling an Event



$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)

if $s \cap s' = \emptyset$ **then return**

$\{x\} = s \cap s'$

if x below ℓ or on ℓ to the right of p **then**

- **if** $x \notin \mathcal{Q}$ **then** $\mathcal{Q}.\text{add}(x)$
- **if** $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
- **if** $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

- report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$

delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

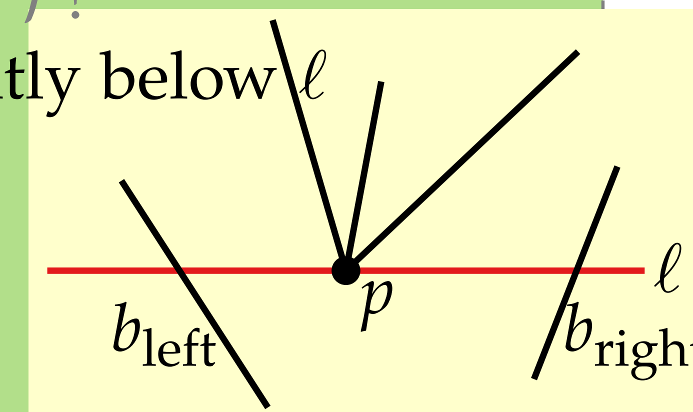
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

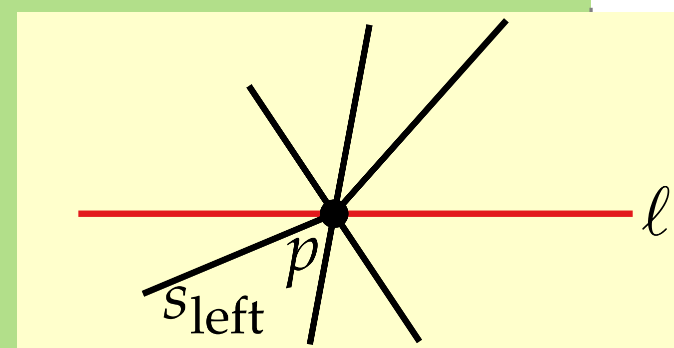
if $U(p) \cup C(p) = \emptyset$ **then**

- $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
- findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$

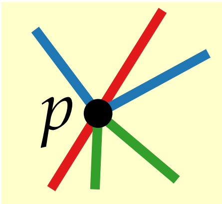
else

- $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$





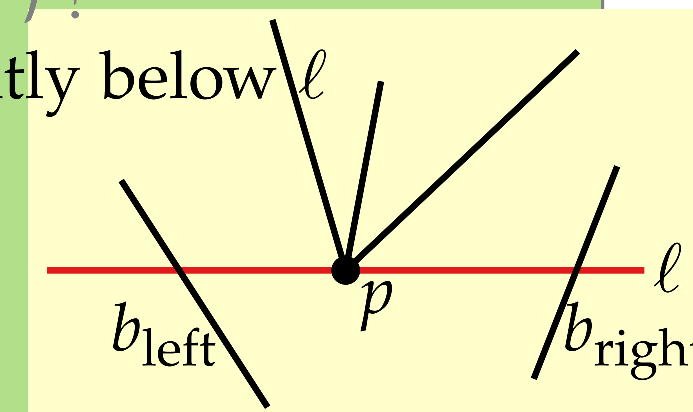
Handling an Event

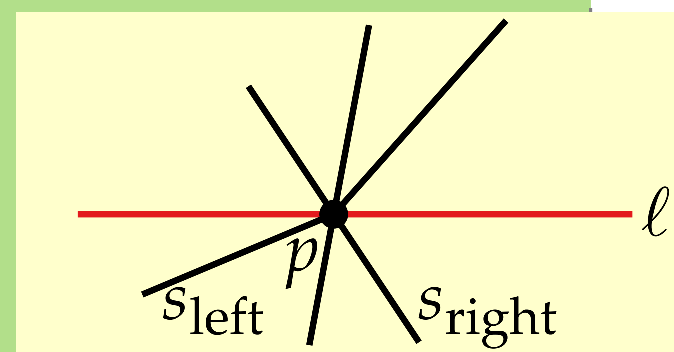


$C(p)$, $L(p)$, $U(p)$

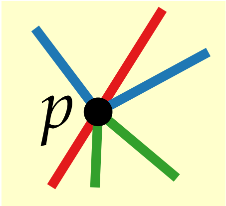
findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else
  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$





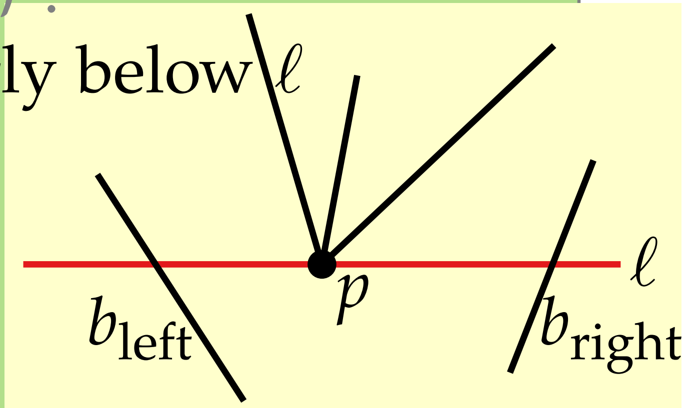
Handling an Event

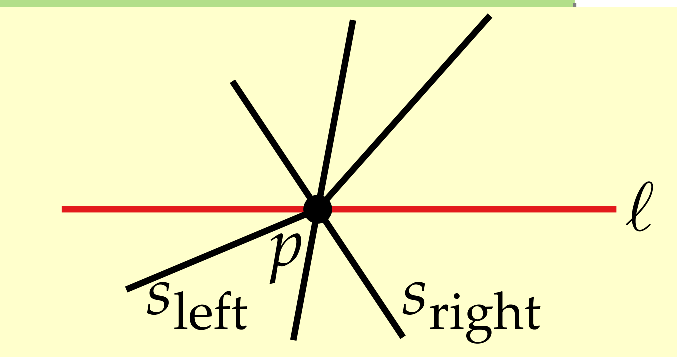


$C(p)$, $L(p)$, $U(p)$

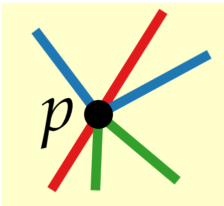
findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else
  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$
  b_{left} = left neighbor of s_{left} in $\mathcal{T}$
  b_{right} = right neighbor of s_{right} in $\mathcal{T}$





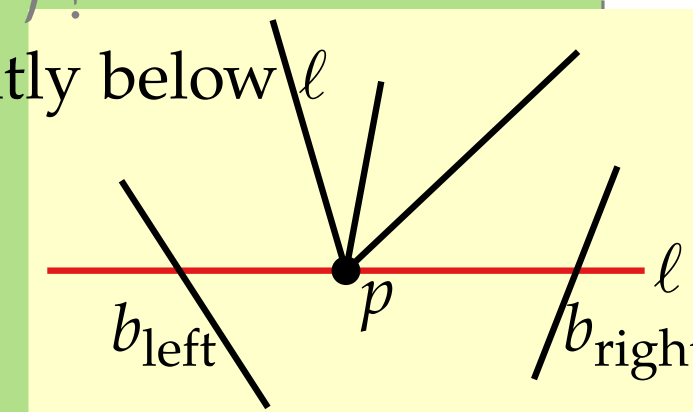
Handling an Event

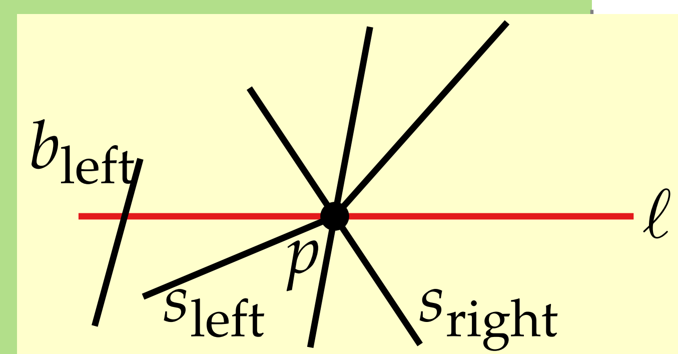


$C(p)$, $L(p)$, $U(p)$

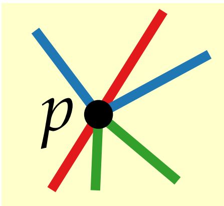
findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else
  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$
  b_{left} = left neighbor of s_{left} in $\mathcal{T}$
  b_{right} = right neighbor of s_{right} in $\mathcal{T}$





Handling an Event



$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)

if $s \cap s' = \emptyset$ **then return**

$\{x\} = s \cap s'$

if x below ℓ or on ℓ to the right of p **then**

- **if** $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
- **if** $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
- **if** $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

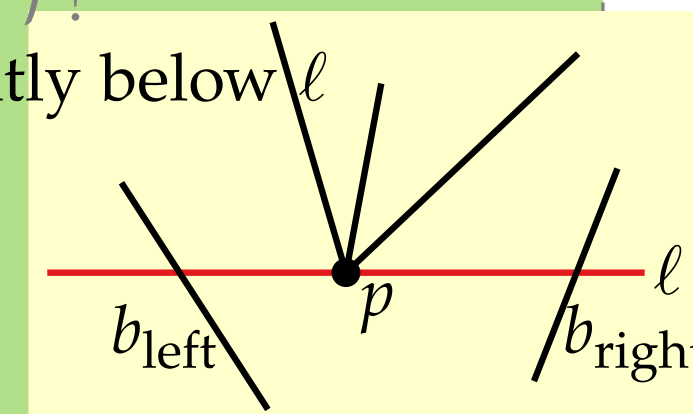
- report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$

delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

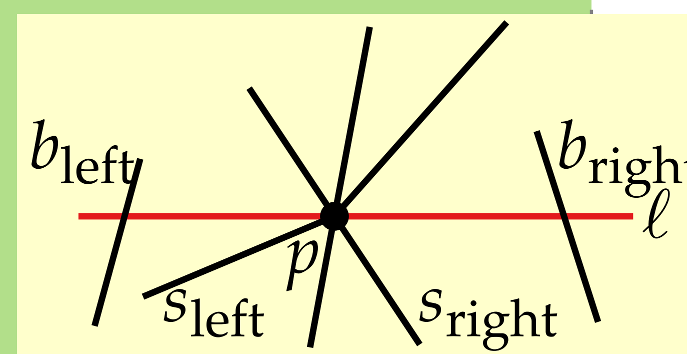
if $U(p) \cup C(p) = \emptyset$ **then**

- $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
- findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$

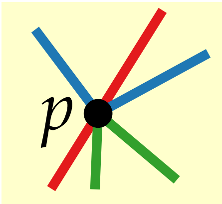


else

- $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$
- b_{left} = left neighbor of s_{left} in $\mathcal{T}$
- b_{right} = right neighbor of s_{right} in $\mathcal{T}$



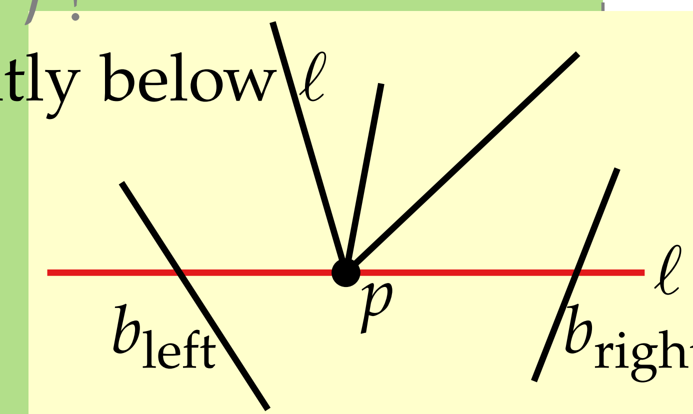
Handling an Event

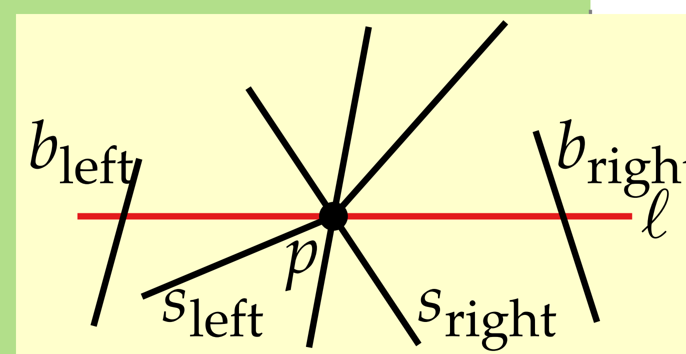


$C(p), L(p), U(p)$

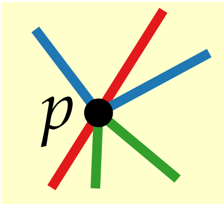
findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}} / b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else
  $s_{\text{left}} / s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$
  b_{left} = left neighbor of s_{left} in $\mathcal{T}$
  b_{right} = right neighbor of s_{right} in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, s_{\text{left}}, p)$





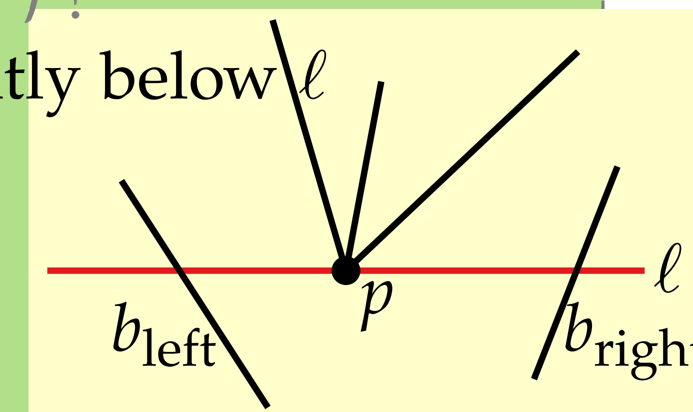
Handling an Event

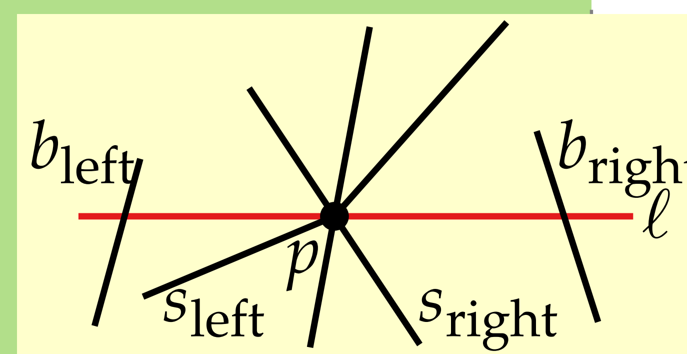


$C(p)$, $L(p)$, $U(p)$

findNewEvent(s, s', p)
if $s \cap s' = \emptyset$ **then return**
$\{x\} = s \cap s'$
if x below ℓ or on ℓ to the right of p **then**
  if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.add(x)
  if $x \in$ rel-int(s) **then** $C(x) \leftarrow C(x) \cup \{s\}$
  if $x \in$ rel-int(s') **then** $C(x) \leftarrow C(x) \cup \{s'\}$

handleEvent(event p)
if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**
  report intersection in p, report segments in $U(p) \cup L(p) \cup C(p)$
delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!
insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ
if $U(p) \cup C(p) = \emptyset$ **then**
  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, b_{\text{right}}, p)$
else
  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$
  b_{left} = left neighbor of s_{left} in $\mathcal{T}$
  b_{right} = right neighbor of s_{right} in $\mathcal{T}$
  findNewEvent$(b_{\text{left}}, s_{\text{left}}, p)$
  findNewEvent$(b_{\text{right}}, s_{\text{right}}, p)$





Computational Geometry

Lecture 2:
Line-Segment Intersection
or
Map Overlay

Part IV:
Correctness

Philipp Kindermann

Winter Semester 2020

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt. Assume:

- Every int. pt $q \prec p$ has been computed correctly.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt. Assume:

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting ℓ in left-to-right order.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting ℓ in left-to-right order.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting ℓ in left-to-right order.

Case I: p is not an interior pt of a segment.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting ℓ in left-to-right order.

Case I: p is not an interior pt of a segment.

$\Rightarrow$ p has been inserted in $\mathcal{Q}$ in the beginning.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting ℓ in left-to-right order.

Case I: p is not an interior pt of a segment.

$\Rightarrow$ p has been inserted in $\mathcal{Q}$ in the beginning.

Segm. in $U(p)$ and $L(p)$ are stored with p in the beginning.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting ℓ in left-to-right order.

Case I: p is not an interior pt of a segment.

$\Rightarrow$ p has been inserted in $\mathcal{Q}$ in the beginning.

Segm. in $U(p)$ and $L(p)$ are stored with p in the beginning.

When p is processed, we output all segm. in $U(p) \cup L(p)$.

Correctness

Lemma. findIntersections(S) correctly computes all intersection points & the segments that contain them.

Proof. Let p be an intersection pt. Assume (by induction):

- Every int. pt $q \prec p$ has been computed correctly.
- $\mathcal{T}$ contains all segments intersecting ℓ in left-to-right order.

Case I: p is not an interior pt of a segment.

$\Rightarrow$ p has been inserted in $\mathcal{Q}$ in the beginning.

Segm. in $U(p)$ and $L(p)$ are stored with p in the beginning.

When p is processed, we output all segm. in $U(p) \cup L(p)$.

$\Rightarrow$ All segments that contain p are reported.

Correctness (Case II)

Case II: p is an int. point of some segment.

Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.

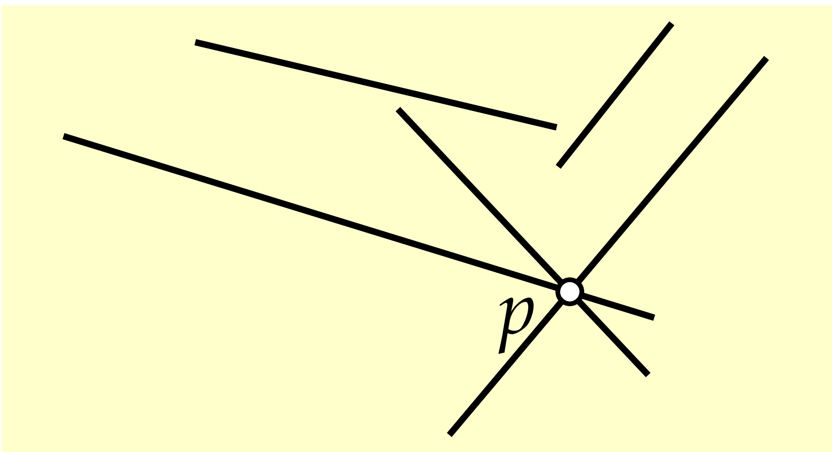
Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.

Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.

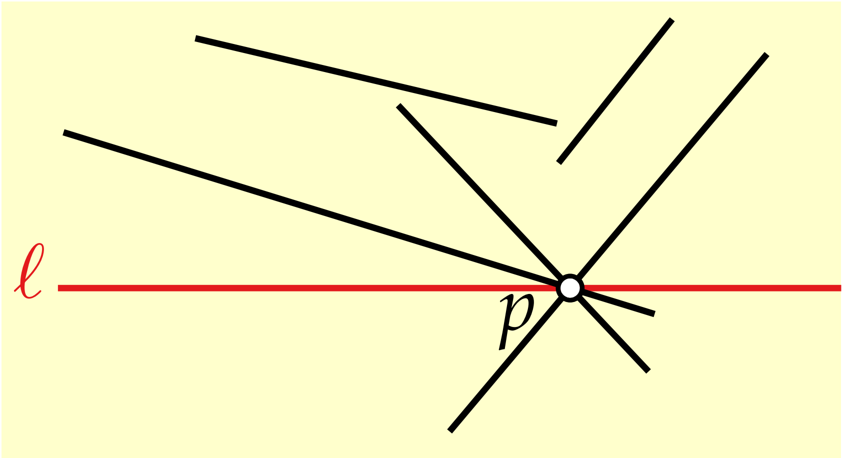
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.



Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.

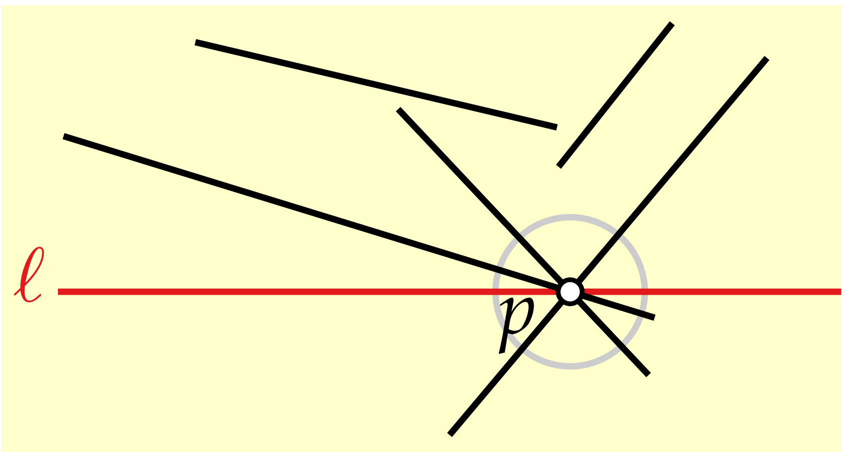
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.



Correctness (Case II)

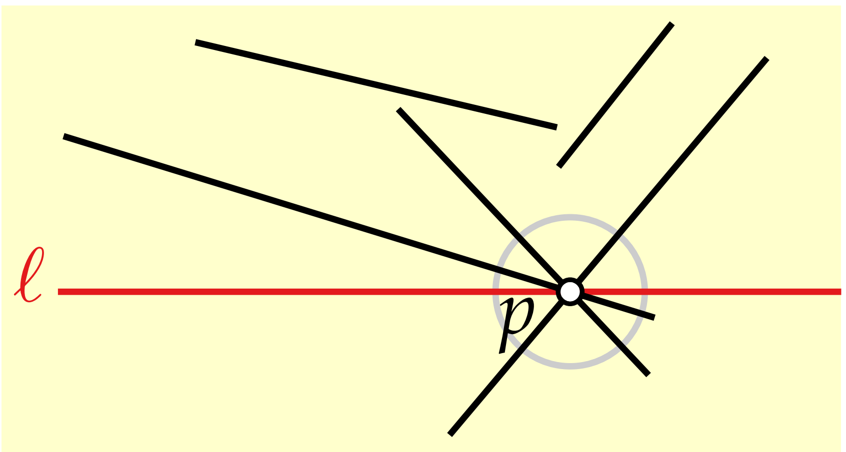
Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.

If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.



Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.

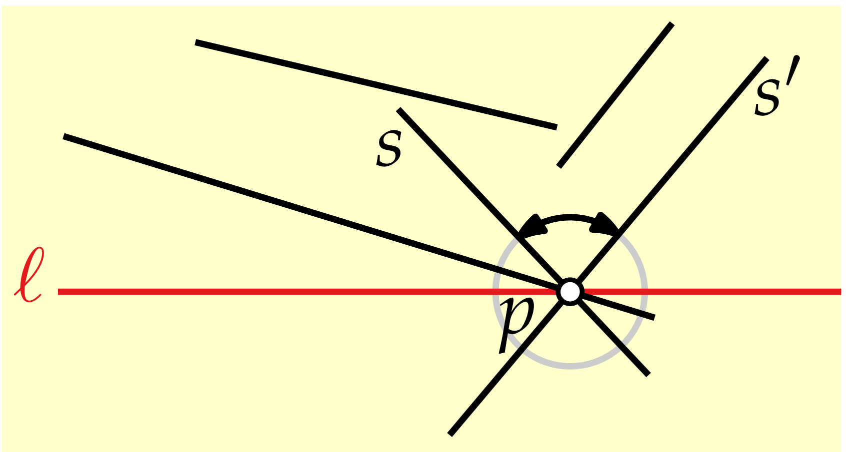


Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around p.

Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.

If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.

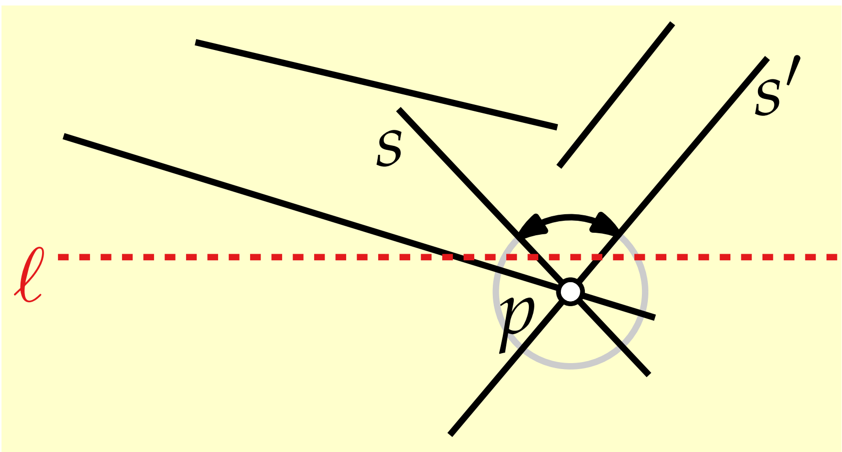


Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around p.

Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.

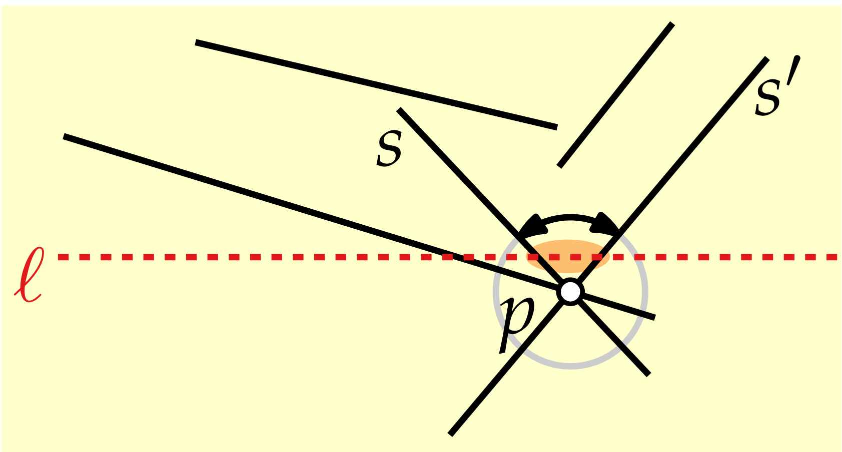
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.



Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around p. Imagine moving ℓ slightly back in time.

Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.

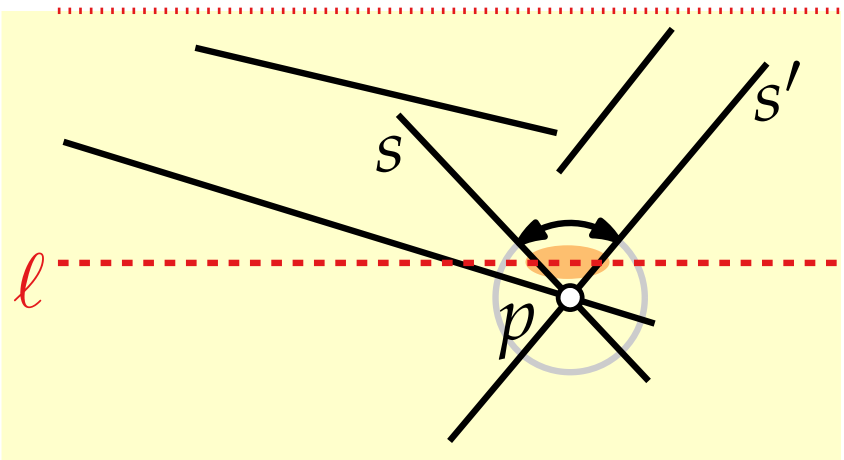


Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around p. Imagine moving ℓ slightly back in time.
Then s, s' were neighbors in the left-to-right order on ℓ (in $\mathcal{T}$).

Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.

If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.



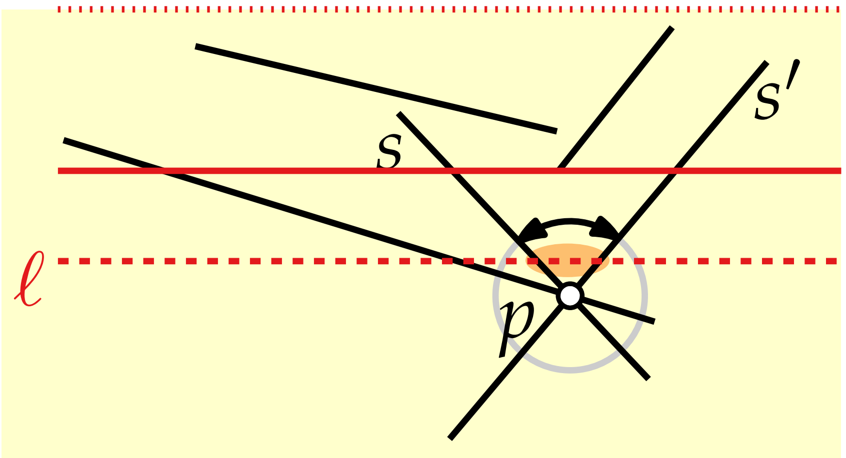
Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around p. Imagine moving ℓ slightly back in time.

Then s, s' were neighbors in the left-to-right order on ℓ (in $\mathcal{T}$).

At the beginning of the alg., they weren't neighbors in $\mathcal{T}$.

Correctness (Case II)

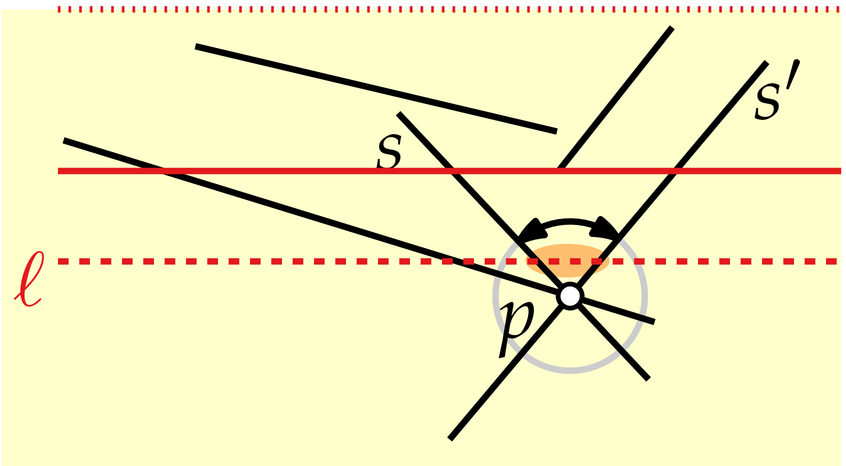
Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.



Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around p. Imagine moving ℓ slightly back in time.
Then s, s' were neighbors in the left-to-right order on ℓ (in $\mathcal{T}$).
At the beginning of the alg., they weren't neighbors in $\mathcal{T}$.
$\Rightarrow$ There was some moment when they became neighbors!

Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.



Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around p. Imagine moving ℓ slightly back in time.

Then s, s' were neighbors in the left-to-right order on ℓ (in $\mathcal{T}$).

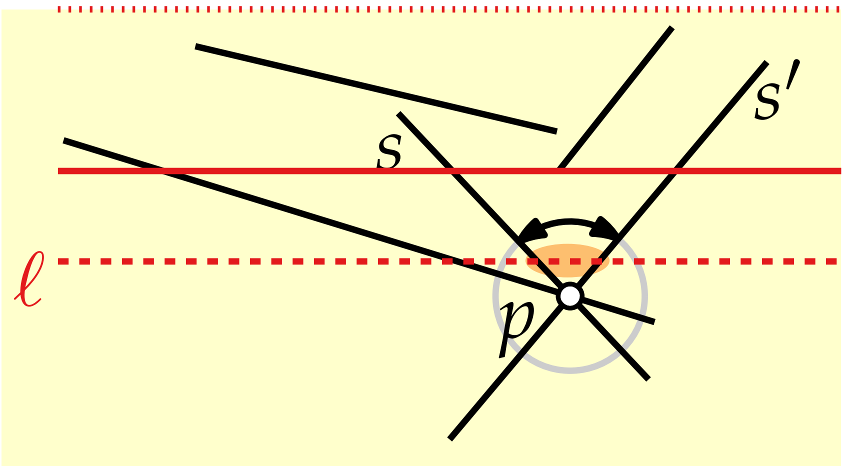
At the beginning of the alg., they weren't neighbors in $\mathcal{T}$.

$\Rightarrow$ There was some moment when they became neighbors!

This is when $\{p\} = s \cap s'$ was inserted into $\mathcal{Q}$.

Correctness (Case II)

Case II: p is an int. point of some segment, i.e., $C(p) \neq \emptyset$.
If p is not an endpt, need that p is inserted into $\mathcal{Q}$ before ℓ reaches p.



We also need that *every* segment with p as an interior point is added to $C(p)$.

Let $s, s' \in C(p)$ be neighbors in the circular ordering of $C(p) \cup \{\ell\}$ around p. Imagine moving ℓ slightly back in time. Then s, s' were neighbors in the left-to-right order on ℓ (in $\mathcal{T}$).
At the beginning of the alg., they weren't neighbors in $\mathcal{T}$.
$\Rightarrow$ There was some moment when they became neighbors!
This is when $\{p\} = s \cap s'$ was inserted into $\mathcal{Q}$. $\square$

Computational Geometry

Lecture 2:
Line-Segment Intersection
or
Map Overlay

Part V:
Running Time

Philipp Kindermann

Winter Semester 2020

```
Q ← ∅;  T ← ⟨ vertical lines at x = −∞ and x = +∞ ⟩   // sentinels
foreach s ∈ S do                                     // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower  endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)
while Q ≠ ∅ do
    p ← Q.nextEvent()
    Q.deleteEvent(p)
    handleEvent(p)
```

Running time?

```
Q ← ∅;  T ← ⟨ vertical lines at x = −∞ and x = +∞ ⟩   // sentinels
foreach s ∈ S do                                     // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)
while Q ≠ ∅ do
    p ← Q.nextEvent()
    Q.deleteEvent(p)
    handleEvent(p)
```

```
handleEvent(event p)
if |U(p) ∪ L(p) ∪ C(p)| > 1 then
    report int. in p, report segments in U(p) ∪ L(p) ∪ C(p)
delete L(p) ∪ C(p) from T   // consecutive in T!
insert U(p) ∪ C(p) into T in their order slightly below ℓ
if U(p) ∪ C(p) = ∅ then
    b_left/b_right = left/right neighbor of p in T
    findNewEvent(b_left, b_right, p)
else
    s_left/s_right = leftmost/rightmost segment in U(p) ∪ C(p)
    b_left = left neighbor of s_left in T
    b_right = right neighbor of s_right in T
    findNewEvent(b_left, s_left, p)
    findNewEvent(b_right, s_right, p)
```

Running time?

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels

foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$

    foreach endpoint p of s **do**

        if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$

        if p lower endpt of s **then** $L(p)$.append(s)

        if p upper endpt of s **then** $U(p)$.append(s)

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

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

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

    handleEvent(p)

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

    report int. in p, report segments in $U(p) \cup L(p) \cup C(p)$

delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

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

    $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$

    findNewEvent(b_{left}, b_{right}, p) →

$\{x\} = s \cap s'$

if $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(x)

else

    $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$

    b_{left} = left neighbor of s_{left} in $\mathcal{T}$

    b_{right} = right neighbor of s_{right} in $\mathcal{T}$

    findNewEvent(b_{left}, s_{left}, p)

    findNewEvent(b_{right}, s_{right}, p)

Running time?

```
Q ← ∅;  T ← ⟨ vertical lines at x = −∞ and x = +∞ ⟩   // sentinels
foreach s ∈ S do                                       // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower  endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)
while Q ≠ ∅ do
    p ← Q.nextEvent()
    Q.deleteEvent(p)
    handleEvent(p)
```

```
handleEvent(event p)
if |U(p) ∪ L(p) ∪ C(p)| > 1 then
    report int. in p, report segments in U(p) ∪ L(p) ∪ C(p)
delete L(p) ∪ C(p) from T   // consecutive in T!
insert U(p) ∪ C(p) into T in their order slightly below ℓ
if U(p) ∪ C(p) = ∅ then
    b_left/b_right = left/right neighbor of p in T
    findNewEvent(b_left, b_right, p)  →  {x} = s ∩ s'
                                          if x ∉ Q then Q.insert(x)
else
    s_left/s_right = leftmost/rightmost segment in U(p) ∪ C(p)
    b_left = left neighbor of s_left in T
    b_right = right neighbor of s_right in T
    findNewEvent(b_left, s_left, p)
    findNewEvent(b_right, s_right, p)
```

Running time?

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels

foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$

  foreach endpoint p of s **do**

    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$

    if p lower endpt of s **then** $L(p)$.append(s)

    if p upper endpt of s **then** $U(p)$.append(s)

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

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

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

  handleEvent(p)

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

  report int. in p, report segments in $U(p) \cup L(p) \cup C(p)$

delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

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

  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$

  findNewEvent(b_{left}, b_{right}, p) → $\{x\} = s \cap s'$; **if** $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(x)

else

  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$

  b_{left} = left neighbor of s_{left} in $\mathcal{T}$

  b_{right} = right neighbor of s_{right} in $\mathcal{T}$

  findNewEvent(b_{left}, s_{left}, p)

  findNewEvent(b_{right}, s_{right}, p)

Running time?

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels

foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$

  foreach endpoint p of s **do**

    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$

    if p lower endpt of s **then** $L(p)$.append(s)

    if p upper endpt of s **then** $U(p)$.append(s)

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

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

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

  handleEvent(p)

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

  report int. in p, report segments in $U(p) \cup L(p) \cup C(p)$

delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

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

  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$

  findNewEvent(b_{left}, b_{right}, p) → $\{x\} = s \cap s'$; **if** $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(x)

else

  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$

  b_{left} = left neighbor of s_{left} in $\mathcal{T}$

  b_{right} = right neighbor of s_{right} in $\mathcal{T}$

  findNewEvent(b_{left}, s_{left}, p)

  findNewEvent(b_{right}, s_{right}, p)

Running time?

```
Q ← ∅;  T ← ⟨ vertical lines at x = −∞ and x = +∞ ⟩   // sentinels
foreach s ∈ S do                                        // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)
while Q ≠ ∅ do
    p ← Q.nextEvent()
    Q.deleteEvent(p)
    handleEvent(p)
```

```
handleEvent(event p)
if |U(p) ∪ L(p) ∪ C(p)| > 1 then
    report int. in p, report segments in U(p) ∪ L(p) ∪ C(p)
delete L(p) ∪ C(p) from T   // consecutive in T!
insert U(p) ∪ C(p) into T in their order slightly below ℓ
if U(p) ∪ C(p) = ∅ then
    b_left/b_right = left/right neighbor of p in T
    findNewEvent(b_left, b_right, p)   →   {x} = s ∩ s'
                                           if x ∉ Q then Q.insert(x)
else
    s_left/s_right = leftmost/rightmost segment in U(p) ∪ C(p)
    b_left = left neighbor of s_left in T
    b_right = right neighbor of s_right in T
    findNewEvent(b_left, s_left, p)
    findNewEvent(b_right, s_right, p)
```

Running time?

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels

foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$

  foreach endpoint p of s **do**

    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$

    if p lower endpt of s **then** $L(p)$.append(s)

    if p upper endpt of s **then** $U(p)$.append(s)

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

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

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

  handleEvent(p)

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

  report int. in p, report segments in $U(p) \cup L(p) \cup C(p)$

delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

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

  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$

  findNewEvent(b_{left}, b_{right}, p) → $\{x\} = s \cap s'$; **if** $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(x)

else

  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$

  b_{left} = left neighbor of s_{left} in $\mathcal{T}$

  b_{right} = right neighbor of s_{right} in $\mathcal{T}$

  findNewEvent(b_{left}, s_{left}, p)

  findNewEvent(b_{right}, s_{right}, p)

Running time?

$\mathcal{Q} \leftarrow \emptyset$; $\mathcal{T} \leftarrow \langle$ vertical lines at $x = -\infty$ and $x = +\infty \rangle$ // sentinels

foreach $s \in S$ **do** // initialize event queue $\mathcal{Q}$

  foreach endpoint p of s **do**

    if $p \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(p); $L(p) = U(p) = C(p) = \emptyset$

    if p lower endpt of s **then** $L(p)$.append(s)

    if p upper endpt of s **then** $U(p)$.append(s)

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

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

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

  handleEvent(p)

handleEvent(event p)

if $|U(p) \cup L(p) \cup C(p)| > 1$ **then**

  report int. in p, report segments in $U(p) \cup L(p) \cup C(p)$

delete $L(p) \cup C(p)$ from $\mathcal{T}$ // consecutive in $\mathcal{T}$!

insert $U(p) \cup C(p)$ into $\mathcal{T}$ in their order slightly below ℓ

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

  $b_{\text{left}}/b_{\text{right}}$ = left/right neighbor of p in $\mathcal{T}$

  findNewEvent(b_{left}, b_{right}, p) → $\{x\} = s \cap s'$; **if** $x \notin \mathcal{Q}$ **then** $\mathcal{Q}$.insert(x)

else

  $s_{\text{left}}/s_{\text{right}}$ = leftmost/rightmost segment in $U(p) \cup C(p)$

  b_{left} = left neighbor of s_{left} in $\mathcal{T}$

  b_{right} = right neighbor of s_{right} in $\mathcal{T}$

  findNewEvent(b_{left}, s_{left}, p)

  findNewEvent(b_{right}, s_{right}, p)

Running time?

```
Q ← ∅;  T ← ⟨ vertical lines at x = −∞ and x = +∞ ⟩   // sentinels
foreach s ∈ S do                                    // initialize event queue Q
    foreach endpoint p of s do
        if p ∉ Q then Q.insert(p); L(p) = U(p) = C(p) = ∅
        if p lower endpt of s then L(p).append(s)
        if p upper endpt of s then U(p).append(s)
while Q ≠ ∅ do
    p ← Q.nextEvent()
    Q.deleteEvent(p)
    handleEvent(p)
```

```
handleEvent(event p)
if |U(p) ∪ L(p) ∪ C(p)| > 1 then
    report int. in p, report segments in U(p) ∪ L(p) ∪ C(p)
delete L(p) ∪ C(p) from T   // consecutive in T!
insert U(p) ∪ C(p) into T in their order slightly below ℓ
if U(p) ∪ C(p) = ∅ then
    b_left/b_right = left/right neighbor of p in T
    findNewEvent(b_left, b_right, p)
else
    s_left/s_right = leftmost/rightmost segment in U(p) ∪ C(p)
    b_left = left neighbor of s_left in T
    b_right = right neighbor of s_right in T
    findNewEvent(b_left, s_left, p)
    findNewEvent(b_right, s_right, p)
```

findNewEvent →

```
{x} = s ∩ s'
if x ∉ Q then Q.insert(x)
```

Running time?

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n + I)\log n)$ time.

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n + I) \log n)$ time.

Proof. Let p be an event pt,

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.

Then it's clear that the runtime is $O((m+n)\log n)$.

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n + I) \log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.

Then it's clear that the runtime is $O((m + n) \log n)$.
We show that $m \in O(n + I)$.

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.

Then it's clear that the runtime is $O((m+n)\log n)$.

We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)

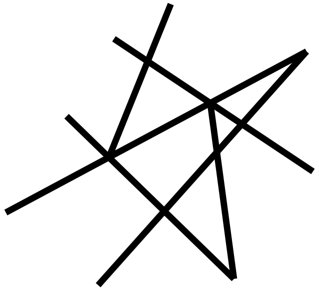
Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.

Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\}$

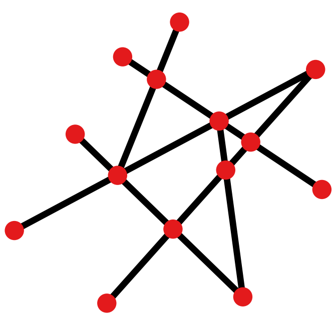


Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.
Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\}$

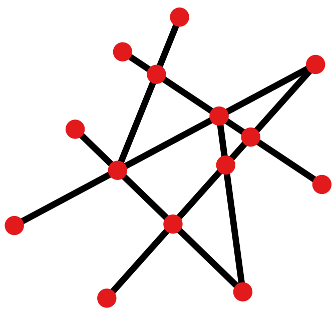


Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.



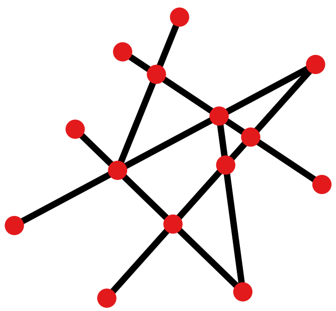
Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\} \Rightarrow |V| \leq 2n + I$.

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.



Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\} \Rightarrow |V| \le 2n + I$.
For any $p \in V$: $m(p) = \deg(p)$.

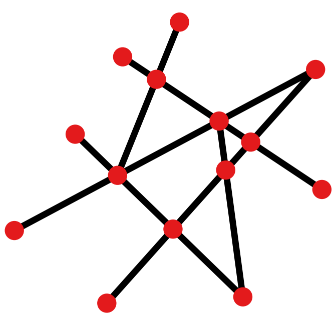
Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.

Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\} \Rightarrow |V| \leq 2n + I$.
For any $p \in V$: $m(p) = \deg(p)$.
$\Rightarrow m = \sum_p \deg(p) = 2|E|$

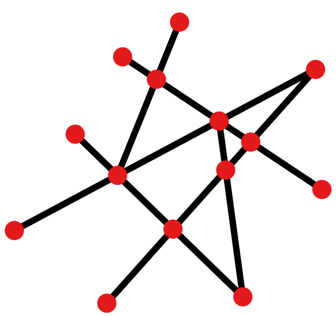


Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.



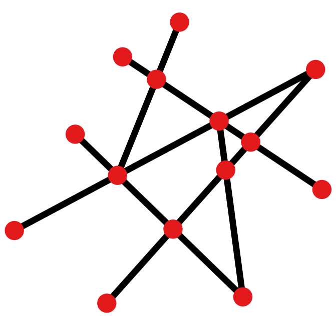
Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\} \Rightarrow |V| \leq 2n + I$.
For any $p \in V$: $m(p) = \deg(p)$.
$\Rightarrow m = \sum_p \deg(p) = 2|E| \leq$

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.



Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\} \Rightarrow |V| \leq 2n + I$.
For any $p \in V$: $m(p) = \deg(p)$.
$\Rightarrow m = \sum_p \deg(p) = 2|E| \leq$

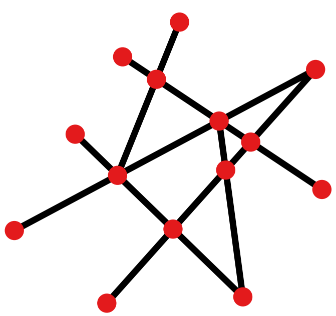
Euler (G is planar!!)

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.



Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\} \Rightarrow |V| \le 2n + I$.
For any $p \in V$: $m(p) = \deg(p)$.
$\Rightarrow m = \sum_p \deg(p) = 2|E| \le 2 \cdot (3|V| - 6)$

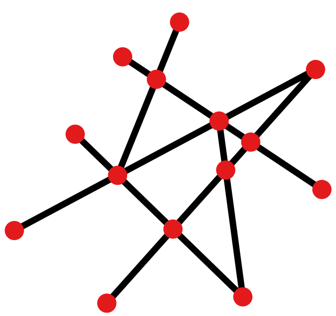
Euler (G is planar!!)

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.



Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\} \Rightarrow |V| \leq 2n + I$.
For any $p \in V$: $m(p) = \deg(p)$.
$\Rightarrow m = \sum_p \deg(p) = 2|E| \leq 2 \cdot (3|V| - 6)$
$\in O(\quad)$

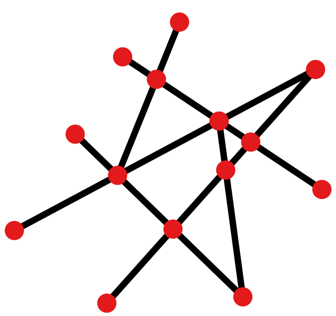
Euler (G is planar!!)

Running Time

Check your knowledge about planar graphs!

Lemma. findIntersections() finds I intersection points among n non-overlapping line segments in $O((n+I)\log n)$ time.

Proof. Let p be an event pt,
$m(p) = |L(p) \cup C(p)| + |U(p) \cup C(p)|$
and $m = \sum_p m(p)$.



Then it's clear that the runtime is $O((m+n)\log n)$.
We show that $m \in O(n+I)$. ($\Rightarrow$ lemma)
Define (geometric) graph $G = (V, E)$ with
$V = \{$ endpts, intersection pts $\} \Rightarrow |V| \le 2n + I$.
For any $p \in V$: $m(p) = \deg(p)$.
$\Rightarrow m = \sum_p \deg(p) = 2|E| \le 2 \cdot (3|V| - 6)$

$\in O(n+I)$ $\square$

Euler (G is planar!!)

Today's Main Result

Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Today's Main Result

Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Today's Main Result

Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure? The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in$

Today's Main Result

Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure?

The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Today's Main Result

Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure? The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Can we do better?

Today's Main Result

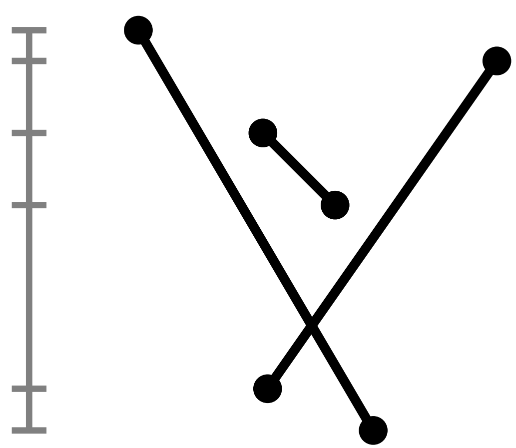
Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

Sure? The event-point queue Q contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n + I)$:-(

Can we do better?



Today's Main Result

Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

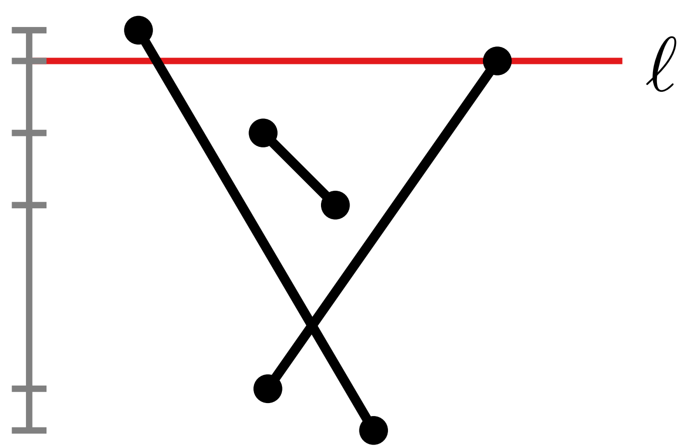
Sure?

The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Can we do better?



Today's Main Result

Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n + I) \log n)$ time and $O(n)$ space.

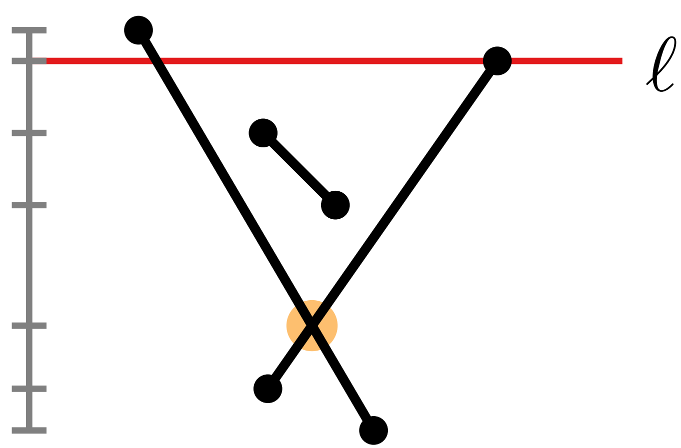
Sure?

The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n + I)$:-(

Can we do better?



Today's Main Result

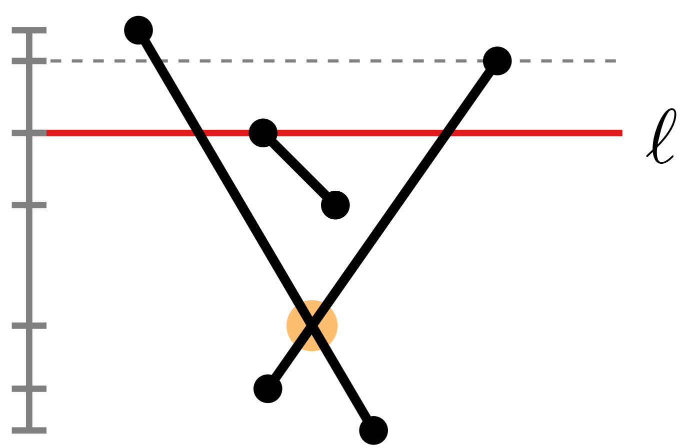
Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure? The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Can we do better?



– insert $s \cap s'$ into $\mathcal{Q}$

Today's Main Result

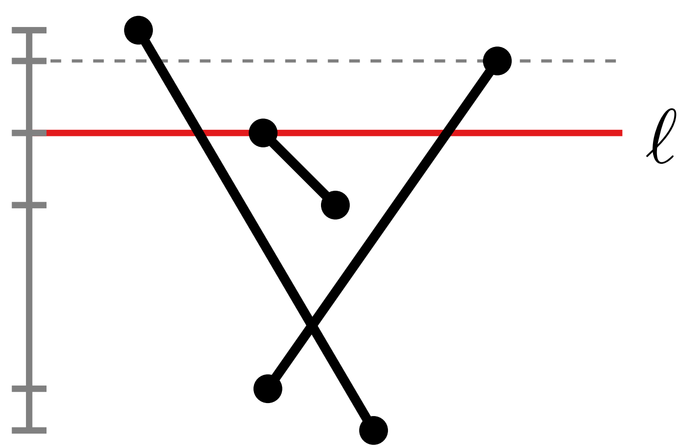
Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure? The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Can we do better?



– insert $s \cap s'$ into $\mathcal{Q}$
– remove $s \cap s'$ from $\mathcal{Q}$

Today's Main Result

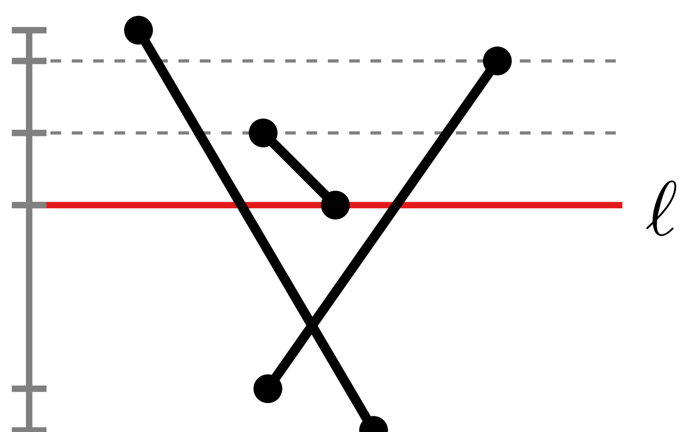
Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure? The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Can we do better?



Today's Main Result

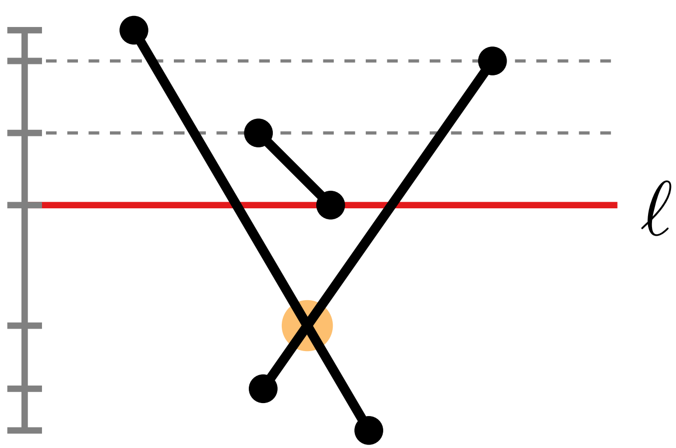
Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure? The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Can we do better?



– insert $s \cap s'$ into $\mathcal{Q}$
– remove $s \cap s'$ from $\mathcal{Q}$
– re-insert $s \cap s'$ into $\mathcal{Q}$

Today's Main Result

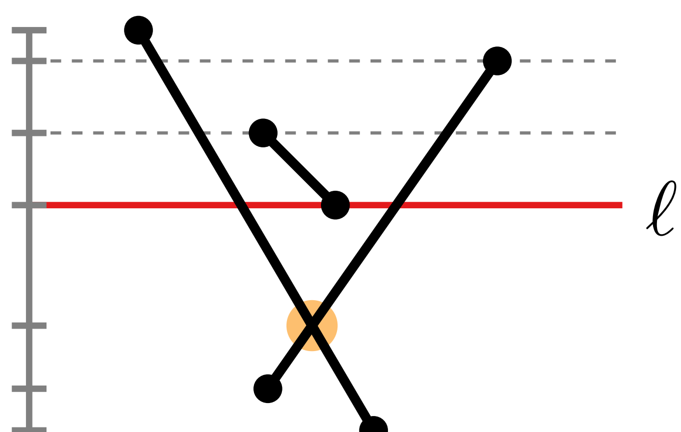
Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure? The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Can we do better?



– insert $s \cap s'$ into $\mathcal{Q}$
– remove $s \cap s'$ from $\mathcal{Q}$
– re-insert $s \cap s'$ into $\mathcal{Q}$

$\Rightarrow$ need just $O(n)$ space;

Today's Main Result

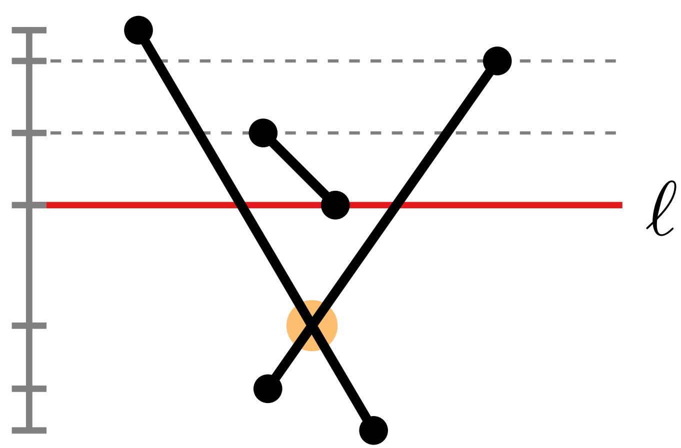
Theorem. We can report all I intersection points among n non-overlapping line segments in the plane and report the segments involved in the intersections in $O((n+I)\log n)$ time and $O(n)$ space.

Sure? The event-point queue $\mathcal{Q}$ contains

- all segment end pts below the sweep line
- all intersection pts below the sweep line

$\Rightarrow$ (worst-case) space consumption $\in \Theta(n+I)$:-(

Can we do better?



– insert $s \cap s'$ into $\mathcal{Q}$

– remove $s \cap s'$ from $\mathcal{Q}$

– re-insert $s \cap s'$ into $\mathcal{Q}$

$\Rightarrow$ need just $O(n)$ space; (asymptotic) running time doesn't change $\square$