Geometric Algorithms

Lecture 1:

- Course Organization
- Introduction
- Line segment intersection for map overlay

This course

instructors:
Kevin Buchin and Herman Haverkort

course website:
`http://www.win.tue.nl/~hermanh/teaching/2IL55/`

content: *algorithmic aspects of spatial data*

- How to store, analyze, create, and manipulate spatial data
- applications in robotics, computer graphics, virtual reality, geographic information systems ...

register on StudyWeb!

This course

learning objectives:

At the end of this course you should be able to ...

- decide which algorithm or data structure to use in order to solve a given basic geometric problem,
- analyze new problems and come up with your own efficient solutions using concepts and techniques from the course.

grading:

- do 3 homework assignments: 20% each
- write research report (in pairs): 40%
- to pass: you need 50% of points in the homework AND a 5 for the research report

This course

before the course you should ...

- have complete Advanced Algorithms (2IL45)
- have experience with the following topics:
  - *basic algorithm design techniques:* divide-and-conquer, greedy algorithms, linear programming ...
  - *basic analysis techniques:* proofs with induction and invariants, O-notation, solving recurrences and summations, basic probability theory ...
  - *basic data structures:* binary search trees, heaps ...

This course

book (compulsory):
[BCKO] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, *Computational Geometry: Algorithms and Applications* (3rd edition). Springer-Verlag, Heidelberg, 2008.

acknowledgement: slides will be based on slides by M. van Kreveld

Introduction

Geometry: points, lines, ...

- Plane (two-dimensional), $\mathbb{R}^2$
- Space (three-dimensional), $\mathbb{R}^3$
- Space (higher-dimensional), $\mathbb{R}^d$

A point in the plane, 3-dimensional space, higher-dimensional space.
$p = (p_x, p_y)$, $p = (p_x, p_y, p_z)$, $p = (p_1, p_2, \ldots, p_d)$

A line in the plane: $y = m \cdot x + c$; representation by m and c

A half-plane in the plane: $y \leq m \cdot x + c$ or $y \geq m \cdot x + c$

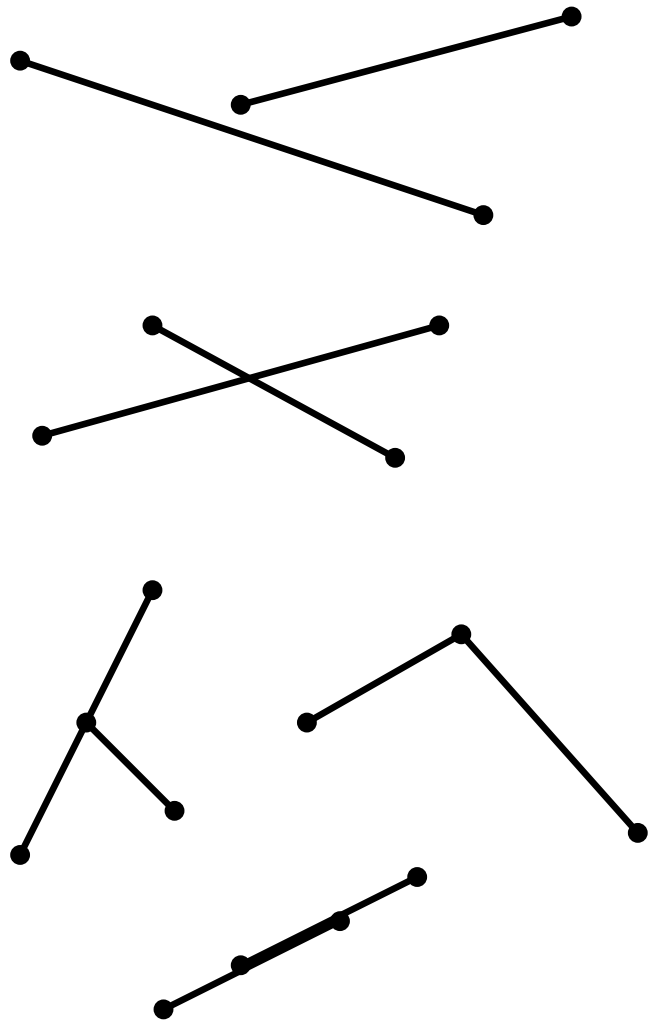
Represent vertical lines? Not by m and c ...

Geometry: line segments

A line segment $\overline{pq}$ is defined by its two endpoints p and q:
$(\lambda \cdot p_x + (1-\lambda) \cdot q_x,\ \lambda \cdot p_y + (1-\lambda) \cdot q_y)$
where $0 \leq \lambda \leq 1$

Line segments are assumed to be closed = with endpoints, not open

Two line segments intersect if they have some point in common. It is a proper intersection if it is exactly one interior point of each line segment



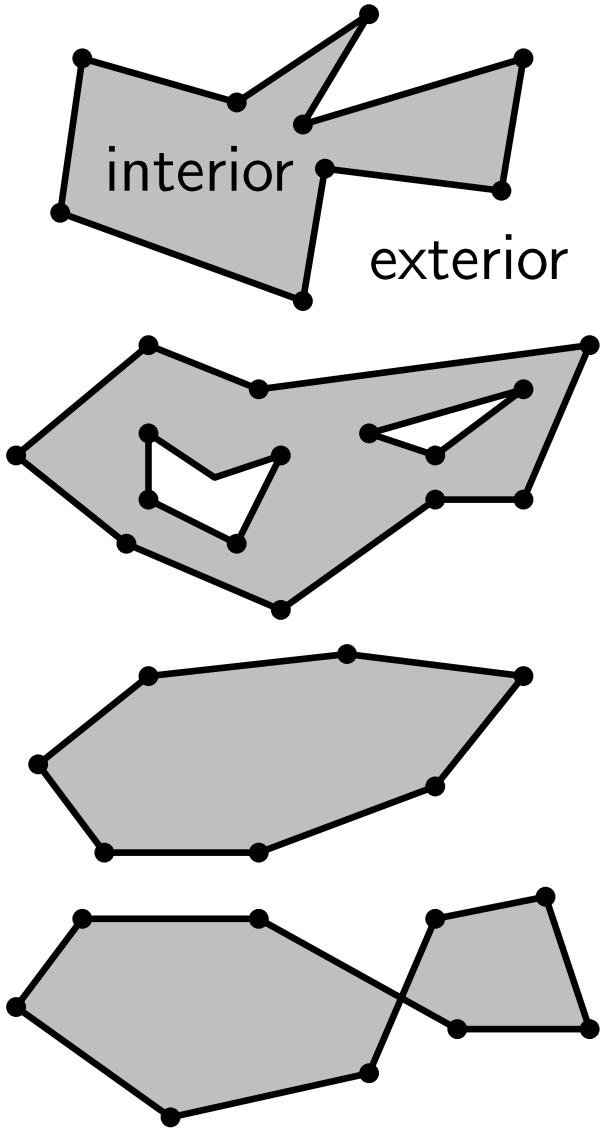
Polygons: simple or not

A polygon is a connected region of the plane bounded by a sequence of line segments

- simple polygon
- polygon with holes
- convex polygon
- non-simple polygon

The line segments of a polygon are called its edges, the endpoints of those edges are the vertices

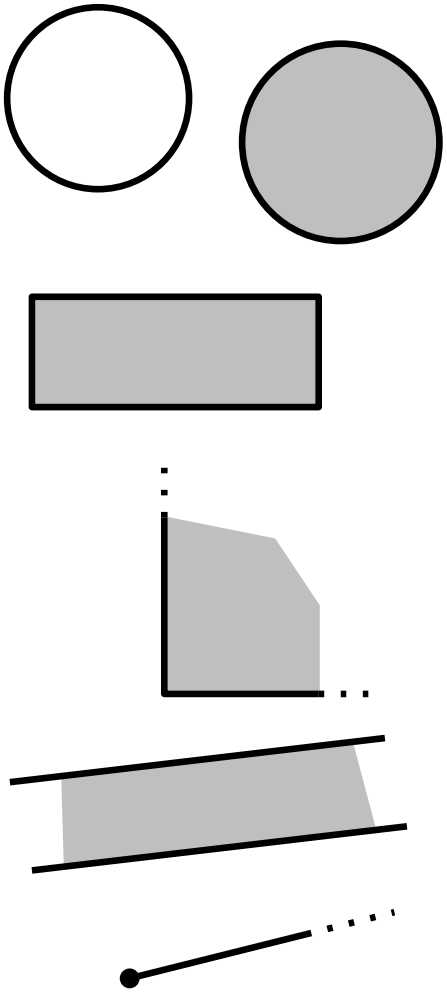
Some abuse: polygon is only boundary



Other shapes: rectangles, circles, disks

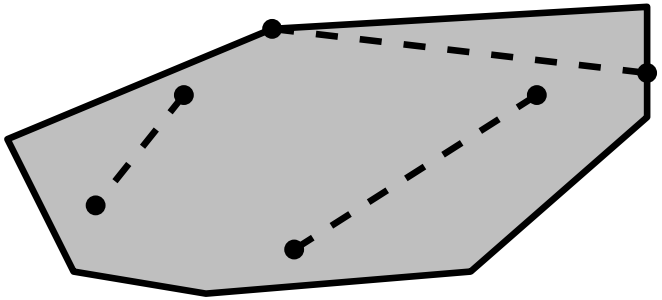
A circle is only the boundary, a disk is the boundary plus the interior

Rectangles, squares, quadrants, slabs, half-lines, wedges, ...

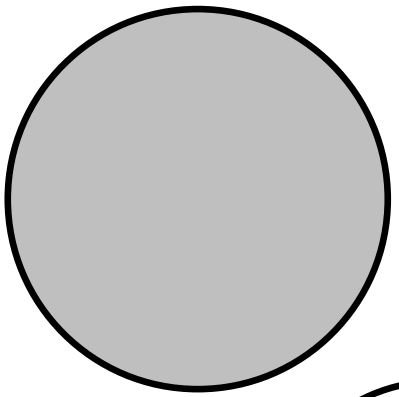


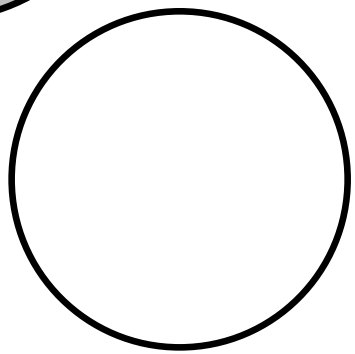
Convexity

A shape or set is convex if for any two points that are part of the shape, the whole connecting line segment is also part of the shape



Question: Which of the following shapes are convex? Point, line segment, line, circle, disk, quadrant?



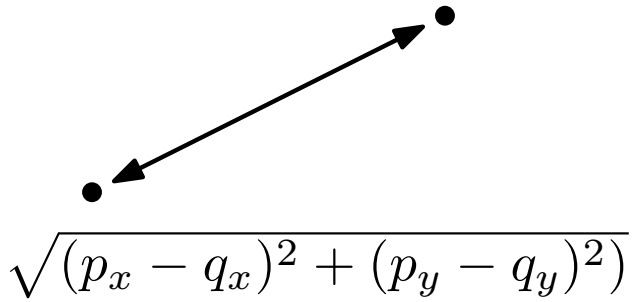


For any subset of the plane (set of points, rectangle, simple polygon), its convex hull is the smallest convex set that contains that subset

Relations: distance, intersection, angle

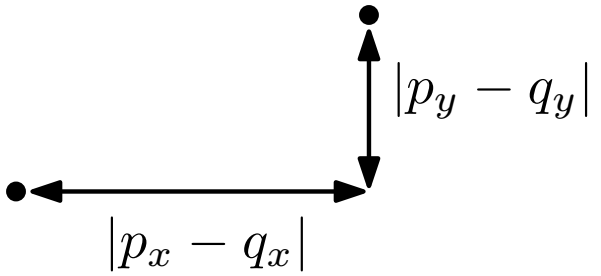
The distance between two points is generally the Euclidean distance:

$$\sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$



Another option: the Manhattan distance:

$$|p_x - q_x| + |p_y - q_y|$$



Question: What is the set of points at equal Manhattan distance to some point?

Relations: distance, intersection, angle

The distance between two geometric objects other than points usually refers to the minimum distance between two points that are part of these objects

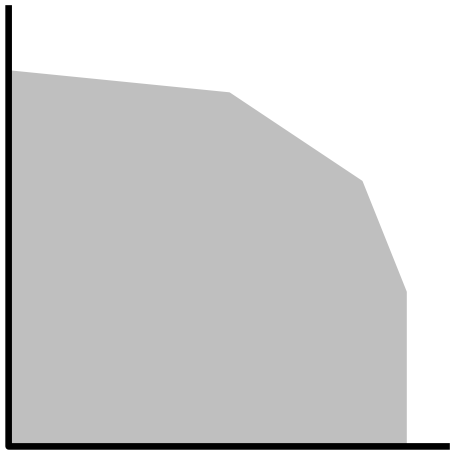
Question: How can the distance between two line segments be realized?

Relations: distance, intersection, angle

The intersection of two geometric objects is the set of points (part of the plane, space) they have in common

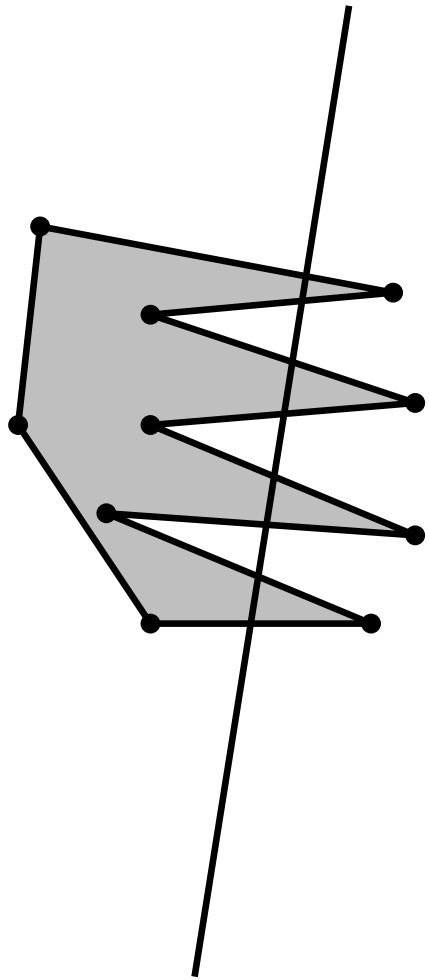
Question 1: How many intersection points can a line and a circle have?

Question 2: What are the possible outcomes of the intersection of a rectangle and a quadrant?



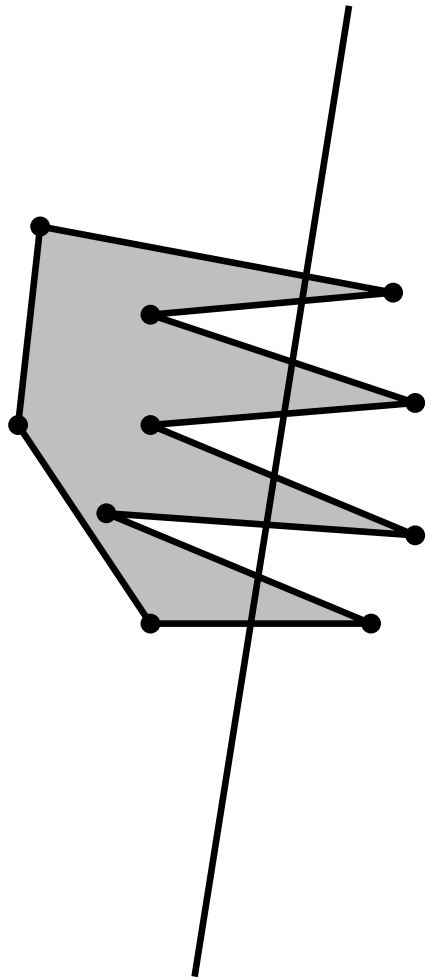
Relations: distance, intersection, angle

Question 3: What is the maximum number of intersection points of a line and a simple polygon with 10 vertices (trick question)?



Relations: distance, intersection, angle

Question 4: What is the maximum number of intersection points of a line and a simple polygon *boundary* with 10 vertices (still a trick question)?



Description size

A point in the plane can be represented using two reals

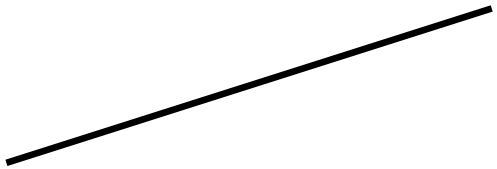
A line in the plane can be represented using two reals and a Boolean (for example)

A line segment can be represented by two points, so four reals

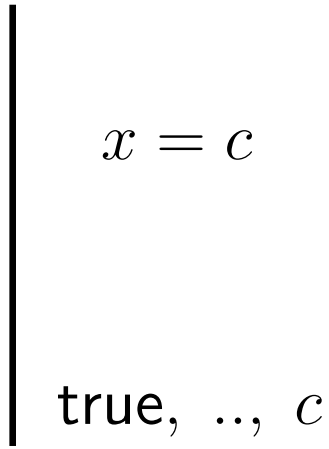
A circle (or disk) requires three reals to store it (center, radius)

A rectangle requires four reals to store it

$y = m \cdot x + c$



false, m, c



true, .., c

Description size

A simple polygon in the plane can be represented using $2n$ reals if it has n vertices (and necessarily, n edges)

A set of n points requires $2n$ reals

A set of n line segments requires $4n$ reals

A point, line, circle, ... requires $O(1)$, or constant, storage.
A simple polygon with n vertices requires $O(n)$, or linear, storage

Computation time

Any computation (distance, intersection) on two objects of $O(1)$ description size takes $O(1)$ time!

Question: Suppose that a simple polygon with n vertices is given; the vertices are given in counterclockwise order along the boundary. Give an efficient algorithm to determine all edges that are intersected by a given line.

How efficient is your algorithm? Why is your algorithm efficient?

Convex hull problem (Advanced Algorithms)

Give an algorithm that computes the convex hull of any given set of n points in the plane efficiently

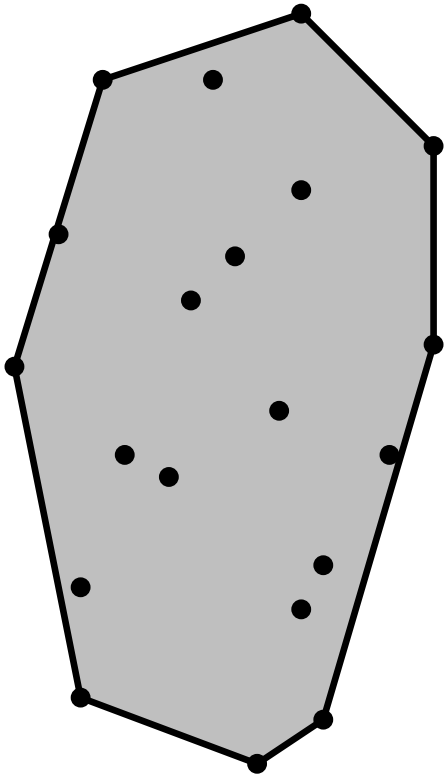
Question 1: What is the input size?

Question 2: Why can't we expect to do any better than $O(n)$ time?

Question 3: Is there any hope of finding an $O(n)$ time algorithm?

Question 4: What's the optimal running time?

Read Chapter 1 of [BCKO] today!



Computational geometry scope

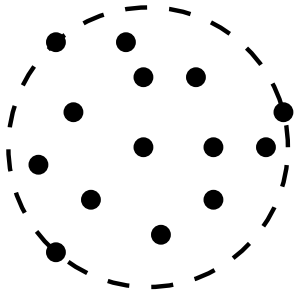
In computational geometry, problems on input with more than constant description size are the ones of interest

Computational geometry (theory): Study of geometric problems on geometric data, and how efficient geometric algorithms that solve them can be

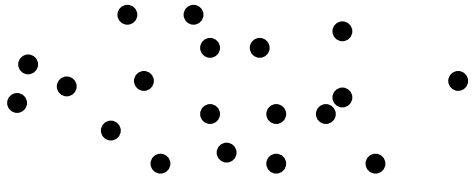
Computational geometry (practice): Study of geometric problems that arise in various applications and how geometric algorithms can help to solve well-defined versions of such problems

Computational geometry theory

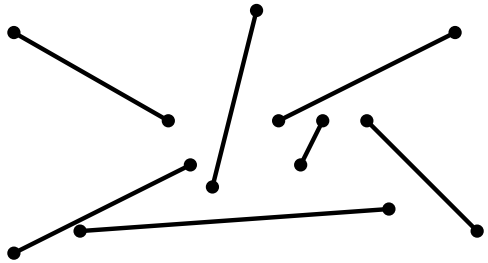
Computational geometry (theory):
Classify abstract geometric problems into classes depending on how efficiently they can be solved



smallest enclosing circle



closest pair



any intersection?

find all intersections

Computational geometry practice

Application areas that require geometric algorithms are computer graphics, motion planning and robotics, geographic information systems, CAD/CAM, statistics, physics simulations, databases, games, multimedia retrieval, ...

- Computing shadows from virtual light sources
- Spatial interpolation from groundwater pollution measurements
- Computing a collision-free path between obstacles
- Computing similarity of two shapes for shape database retrieval

Computational geometry history

Early 70s: First attention for geometric problems from algorithms researchers

1976: First PhD thesis in computational geometry (Michael Shamos)

1985: First Annual ACM Symposium on Computational Geometry. Also: first textbook

1996: CGAL: first serious implementation effort for robust geometric algorithms

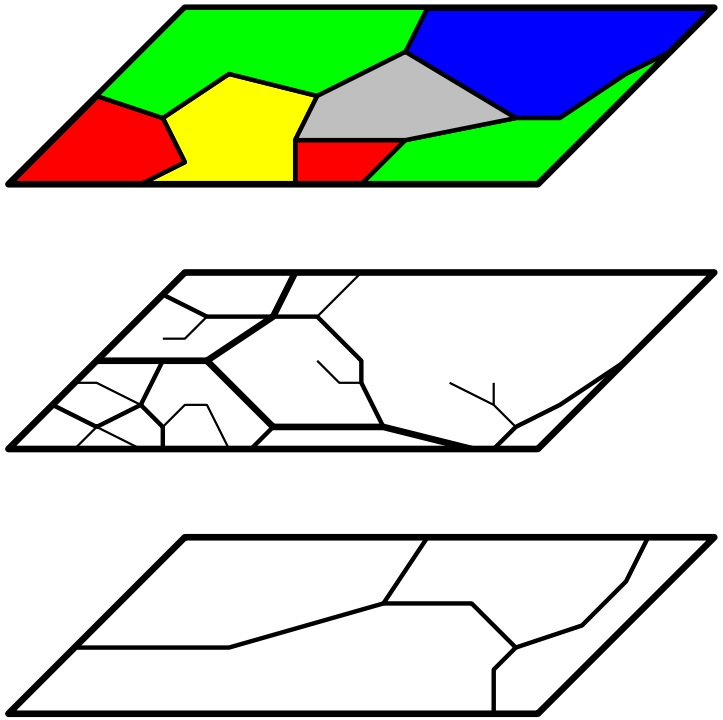
1997: First handbook on computational geometry (second one in 2000)

Line segment intersection

Map layers

In a geographic information system (GIS) data is stored in separate layers

A layer stores the geometric information about some theme, like land cover, road network, municipality boundaries, red fox habitat, ...

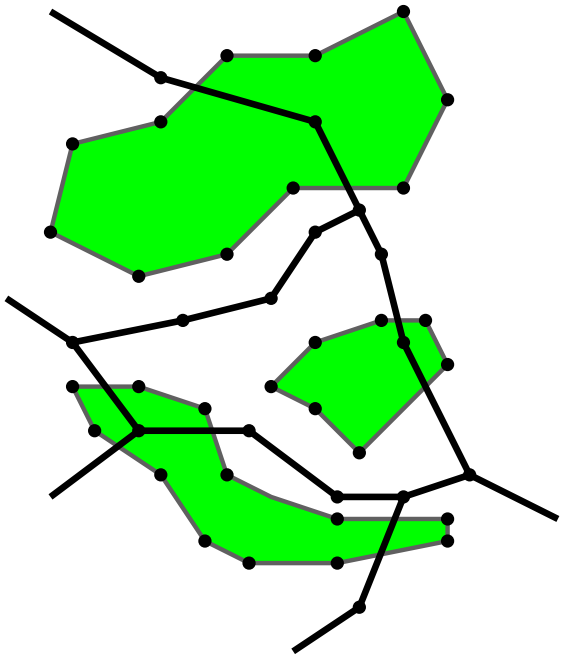


Map overlay

Map overlay is the combination of two (or more) map layers

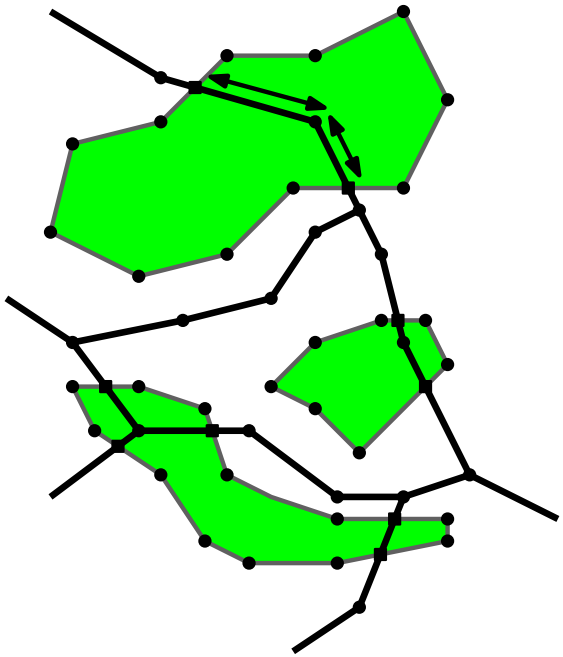
It is needed to answer questions like:

- What is the total length of roads through forests?
- What is the total area of corn fields within $1\,\text{km}$ from a river?



Map overlay

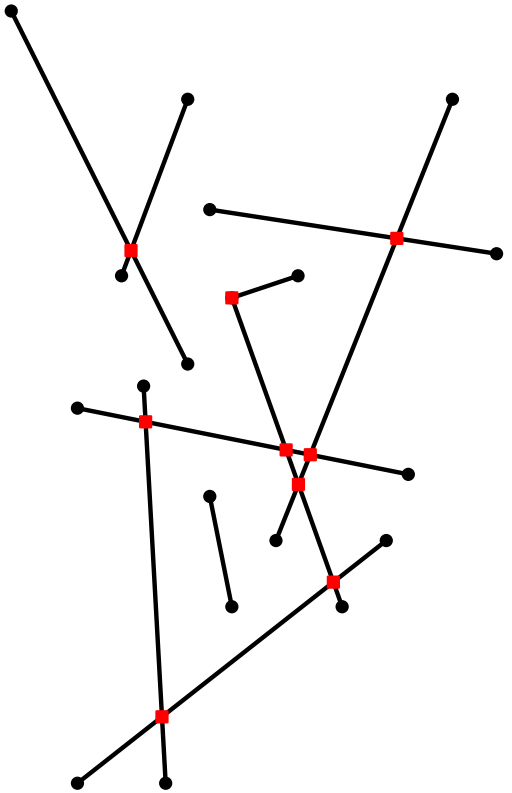
To solve map overlay questions, we need (at the least) intersection points from two sets of line segments (possibly, boundaries of regions)



The (easy) problem

Let's first look at the easiest version of the problem:

Given a set of of n line segments in the plane, find all intersection points efficiently

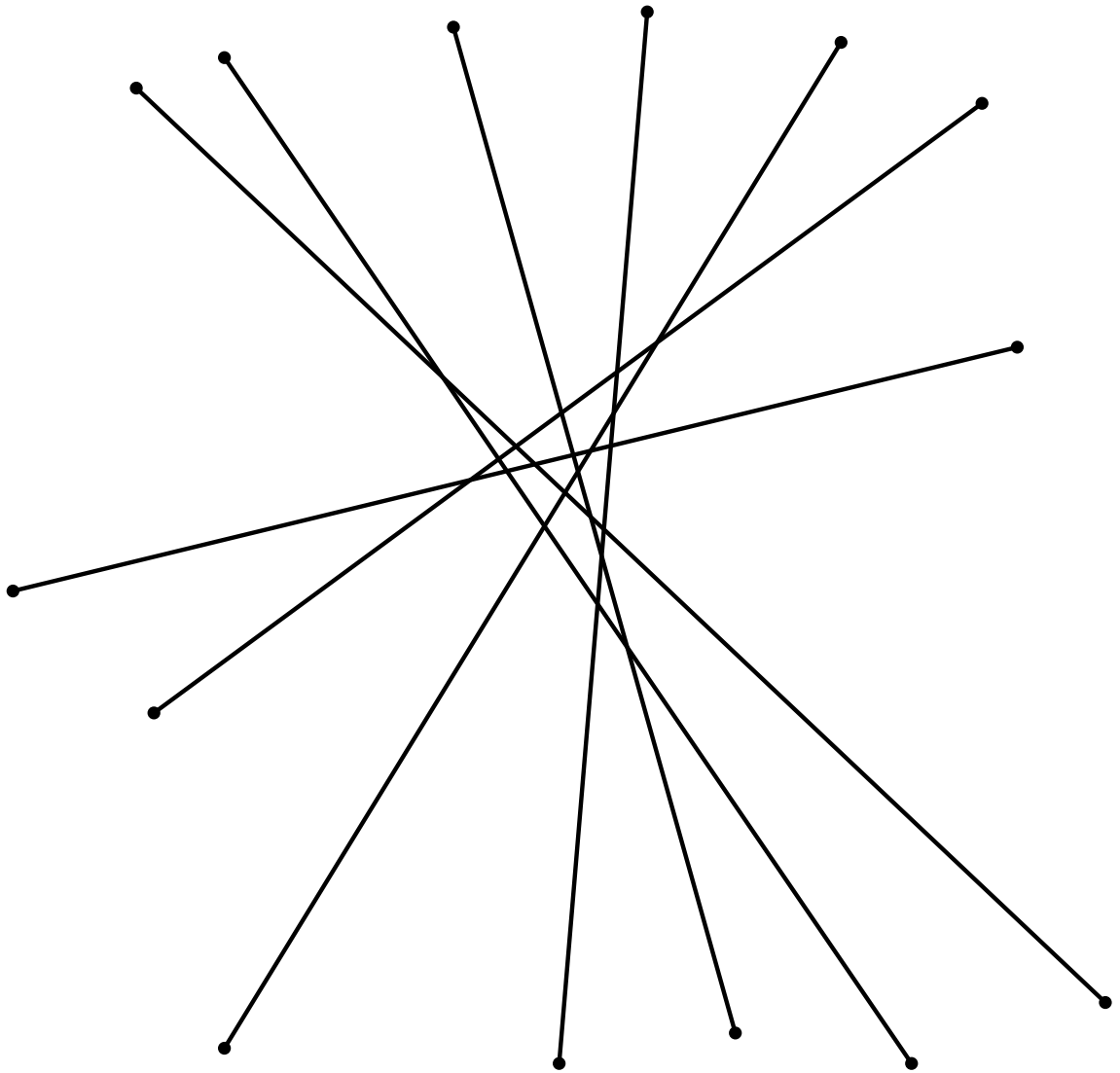


An easy, optimal algorithm?

Algorithm FindIntersections(S)
Input. A set S of line segments in the plane.
Output. The set of intersection points among the segments in S.

```
1.  for each pair of line segments e_i, e_j ∈ S
2.      do if e_i and e_j intersect
3.          then report their intersection point
```

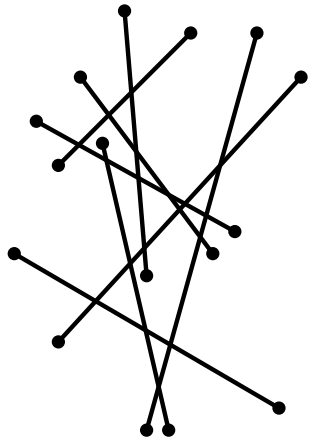
Question: Why can we say that this algorithm is optimal?

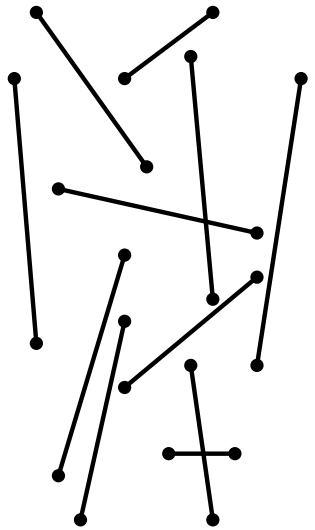


Output-sensitive algorithm

The asymptotic running time of an algorithm is always input-sensitive (depends on n)

We may also want the running time to be output-sensitive: if the output is large, it is fine to spend a lot of time, but if the output is small, we want a fast algorithm

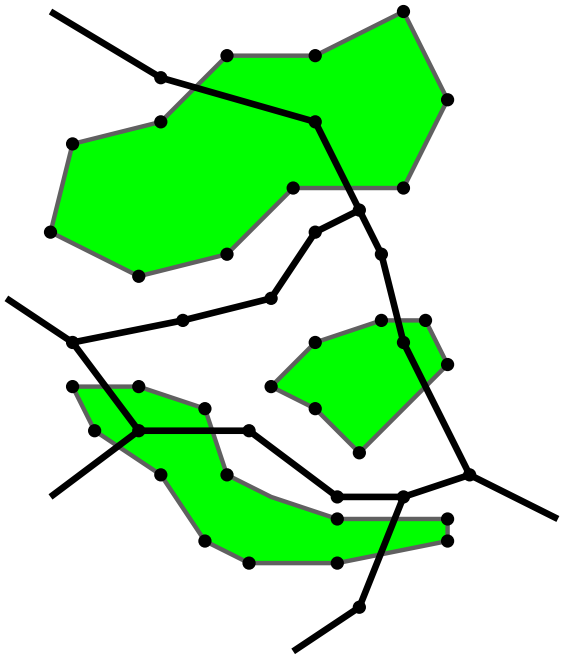




Intersection points in practice

Question: How many intersection points do we typically expect in our application?

If this number is k, and if $k = O(n)$, it would be nice if the algorithm runs in $O(n \log n)$ time

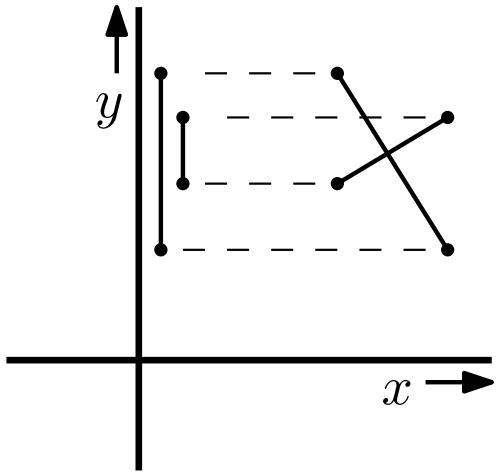


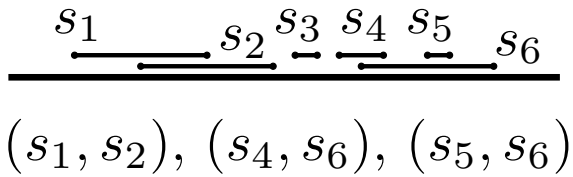
First attempt

Observation: Two line segments can only intersect if their y-spans have an overlap

So, how about only testing pairs of line segments that intersect in the y-projection?

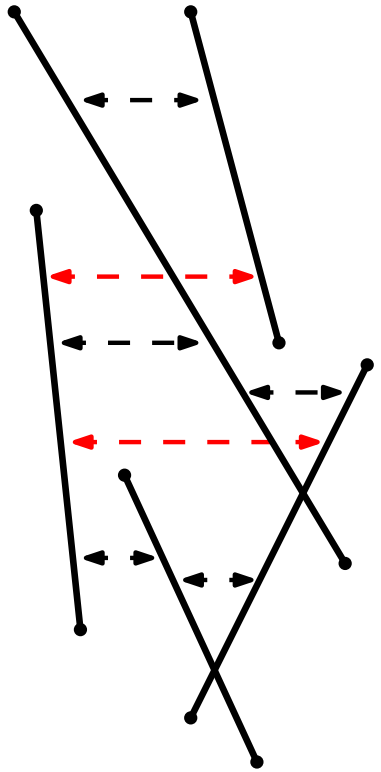
1-D problem: Given a set of intervals on the real line, find all partly overlapping pairs





Second attempt

Refined observation: Two line segments can only intersect if their y-spans have an overlap, and they are adjacent in the x-order at that y-coordinate (they are *horizontal neighbors*)

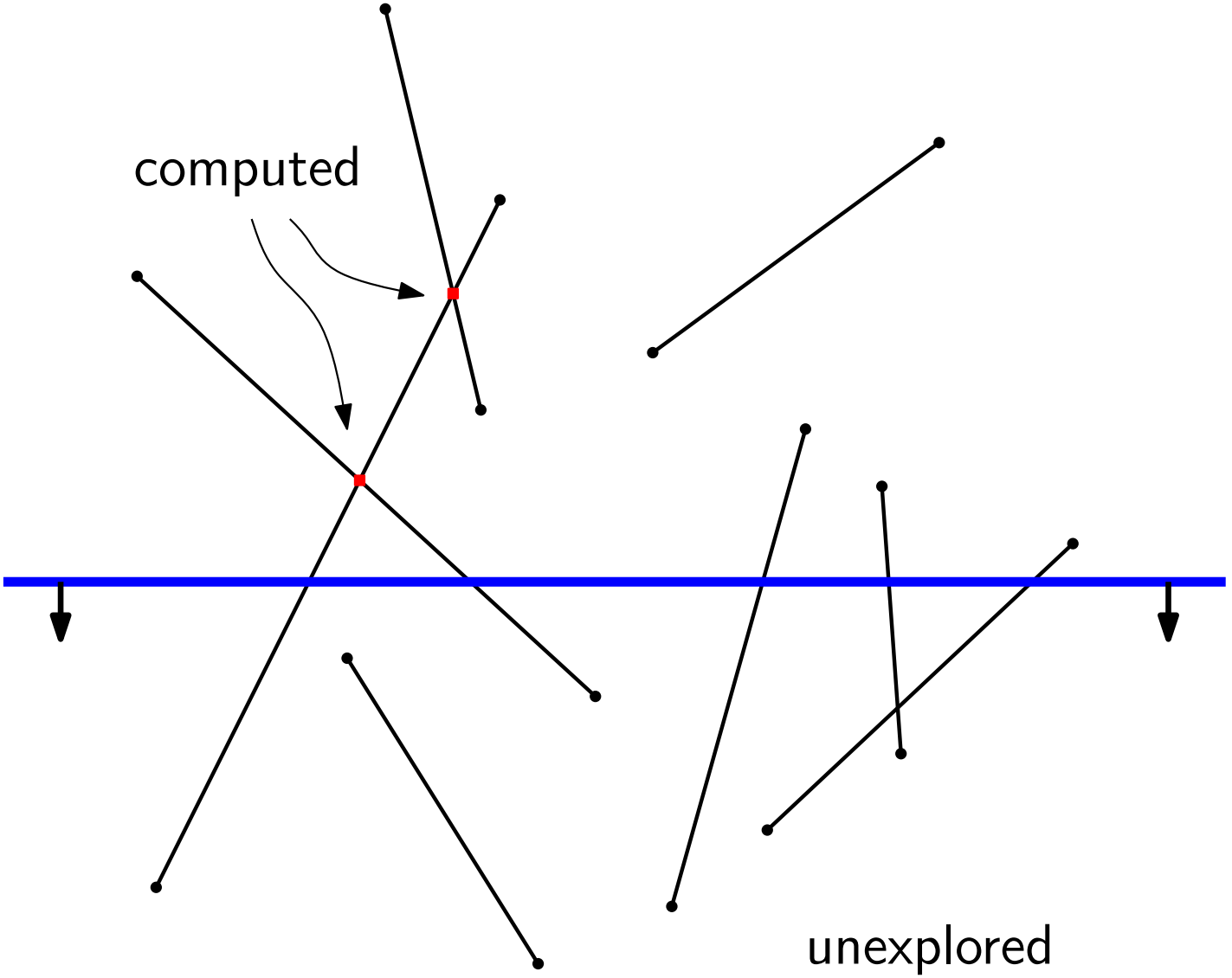


Plane sweep

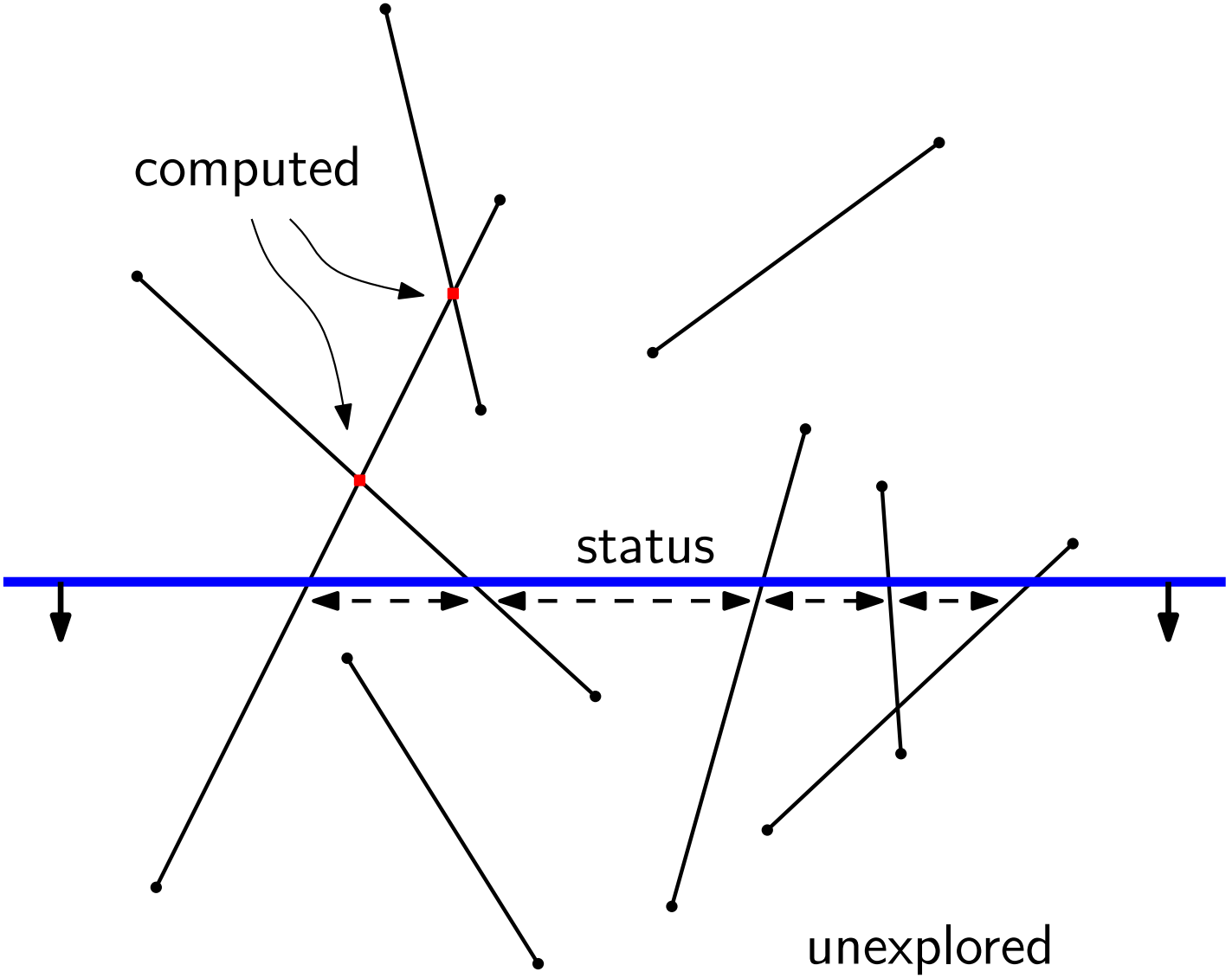
The **plane sweep technique**: Imagine a horizontal line passing over the plane from top to bottom, solving the problem as it moves

- The sweep line stops and the algorithm computes at certain positions ⇒ events
- The algorithm stores the relevant situation at the current position of the sweep line ⇒ status
- The algorithm knows everything it needs to know above the sweep line, and found all intersection points

Sweep



Sweep and status

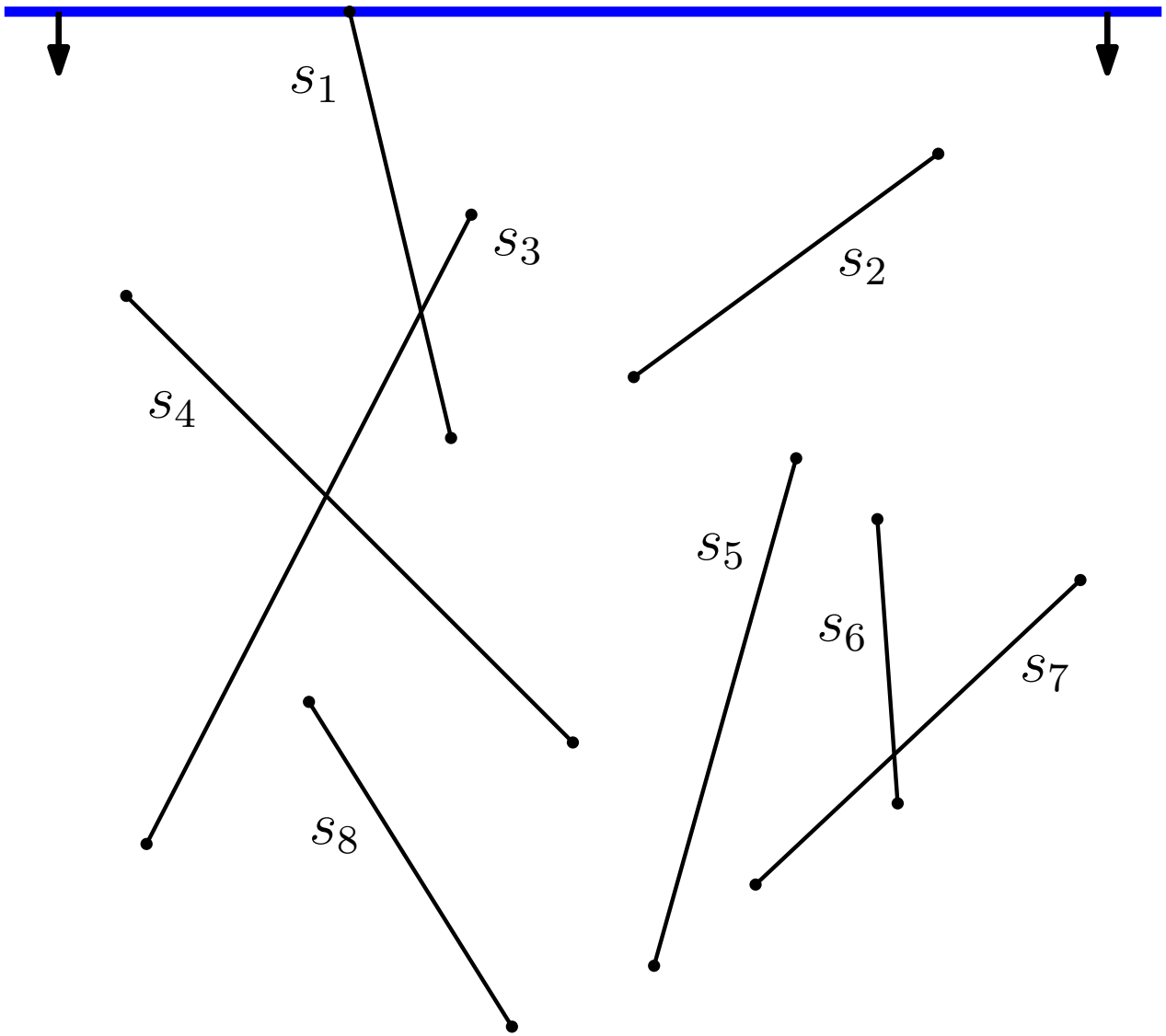


Status and events

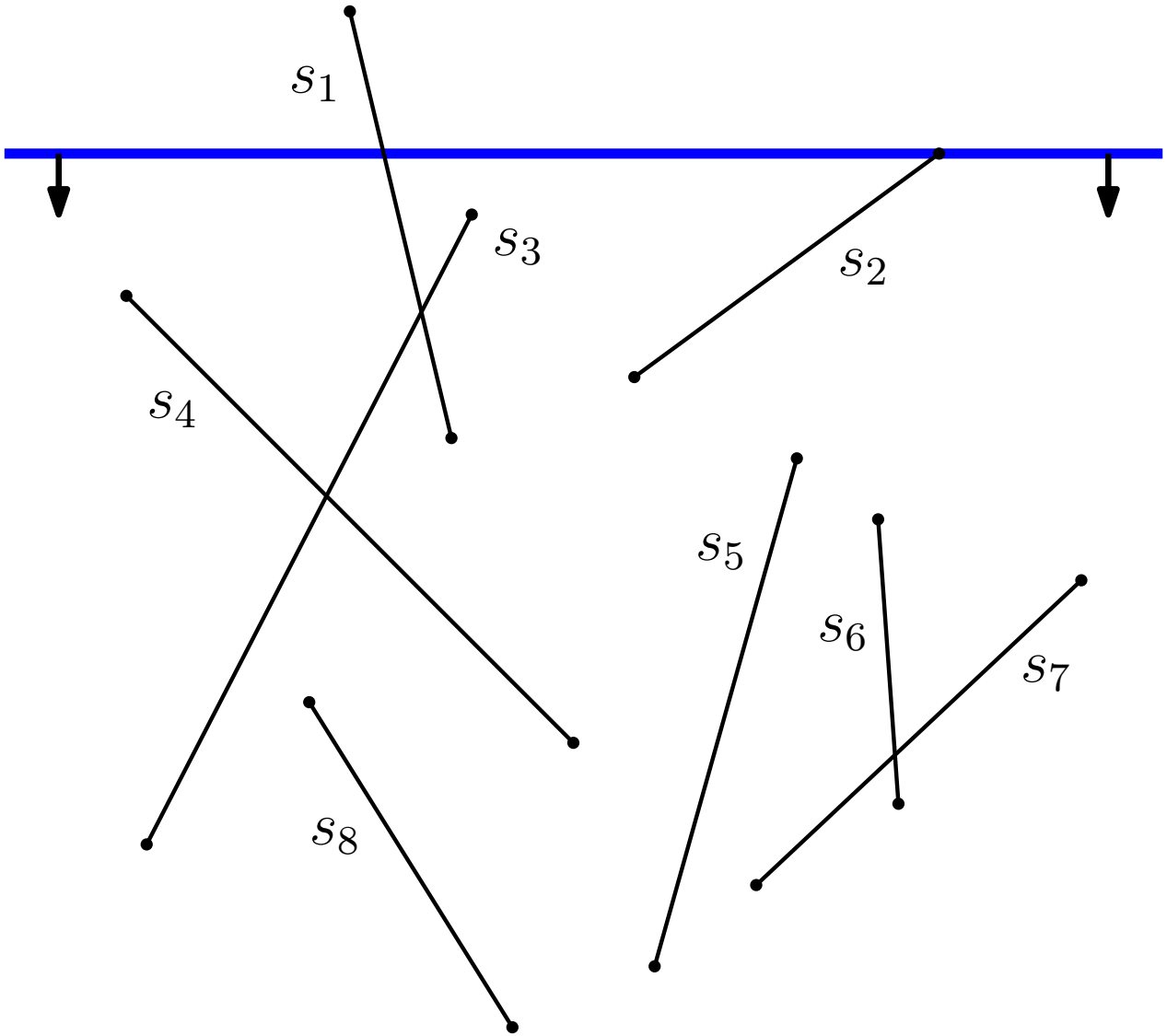
The status of this particular plane sweep algorithm, at the current position of the sweep line, is the set of line segments intersecting the sweep line, ordered from left to right

The events occur when the *status changes*, and when *output is generated*

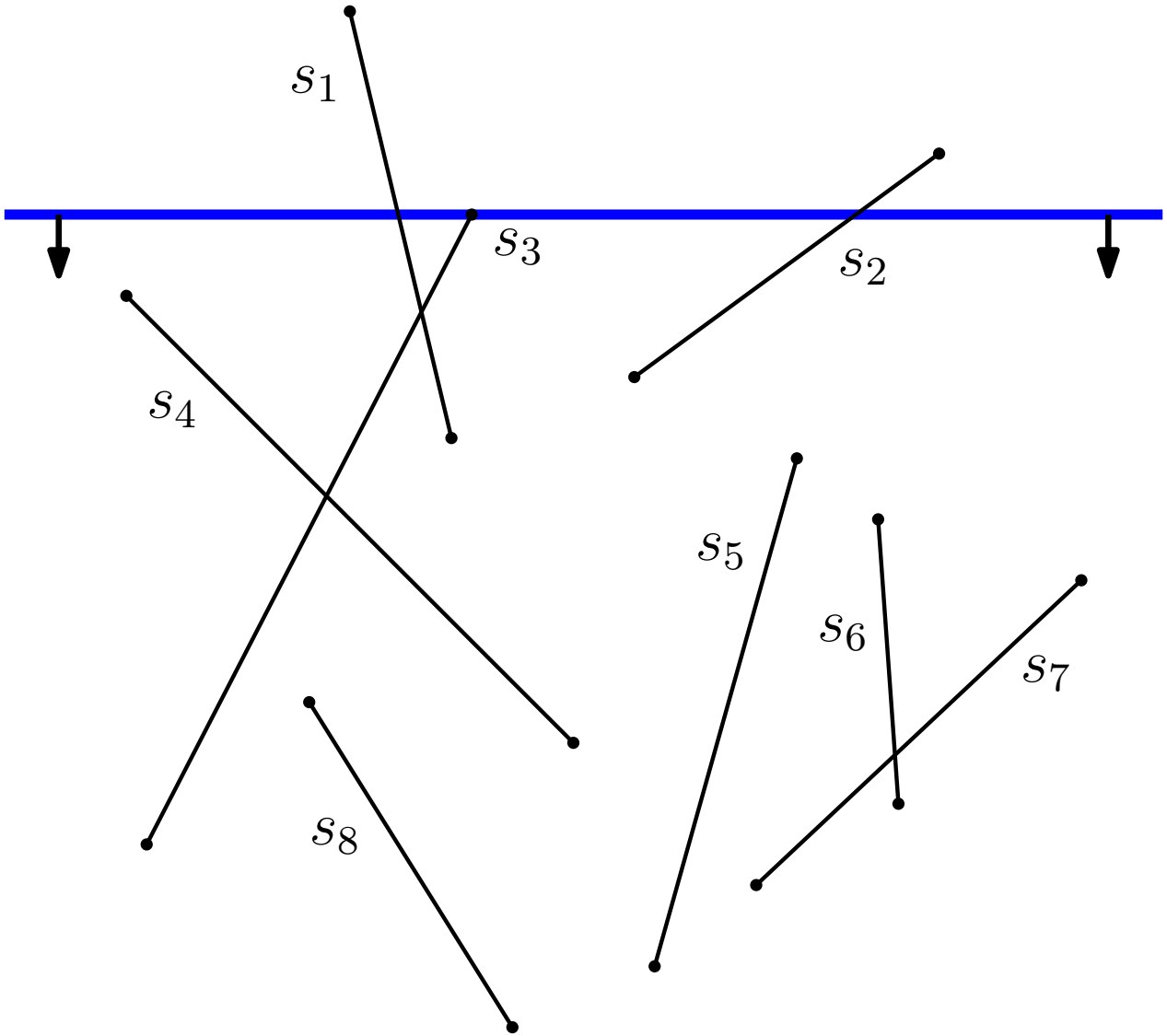
$$\text{event} \approx \text{interesting } y\text{-coordinate}$$



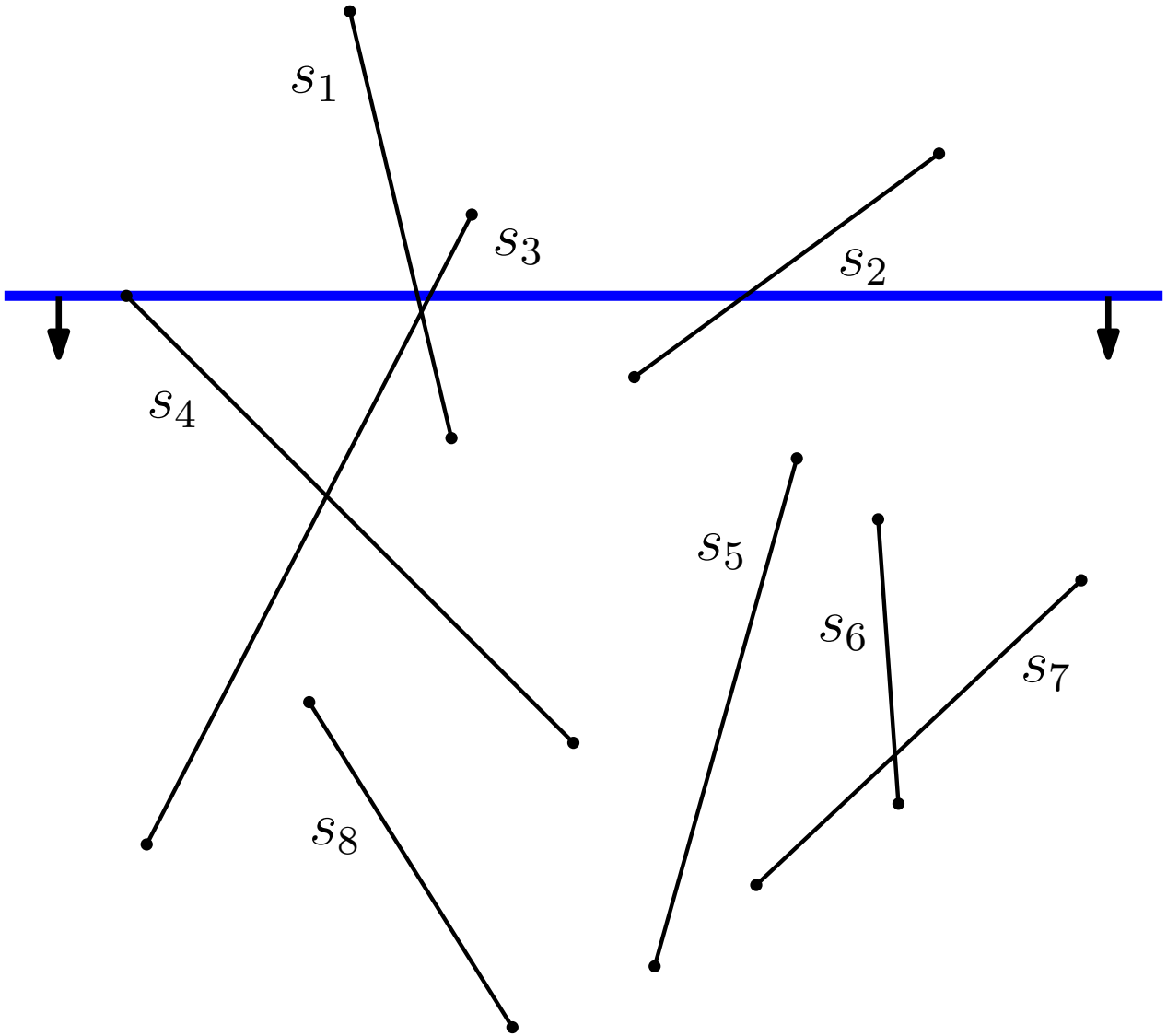
add s_1



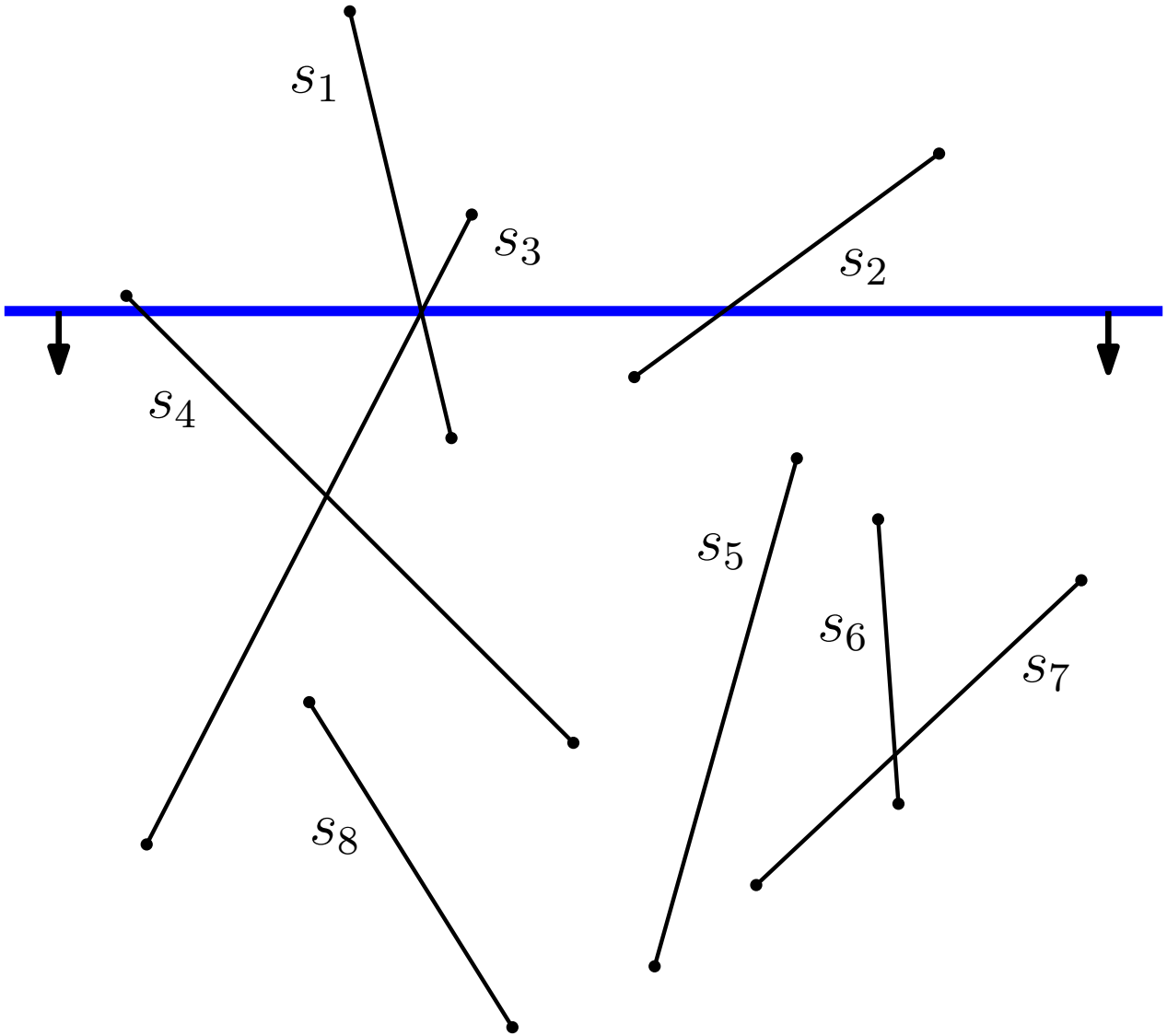
add s_2 after s_1



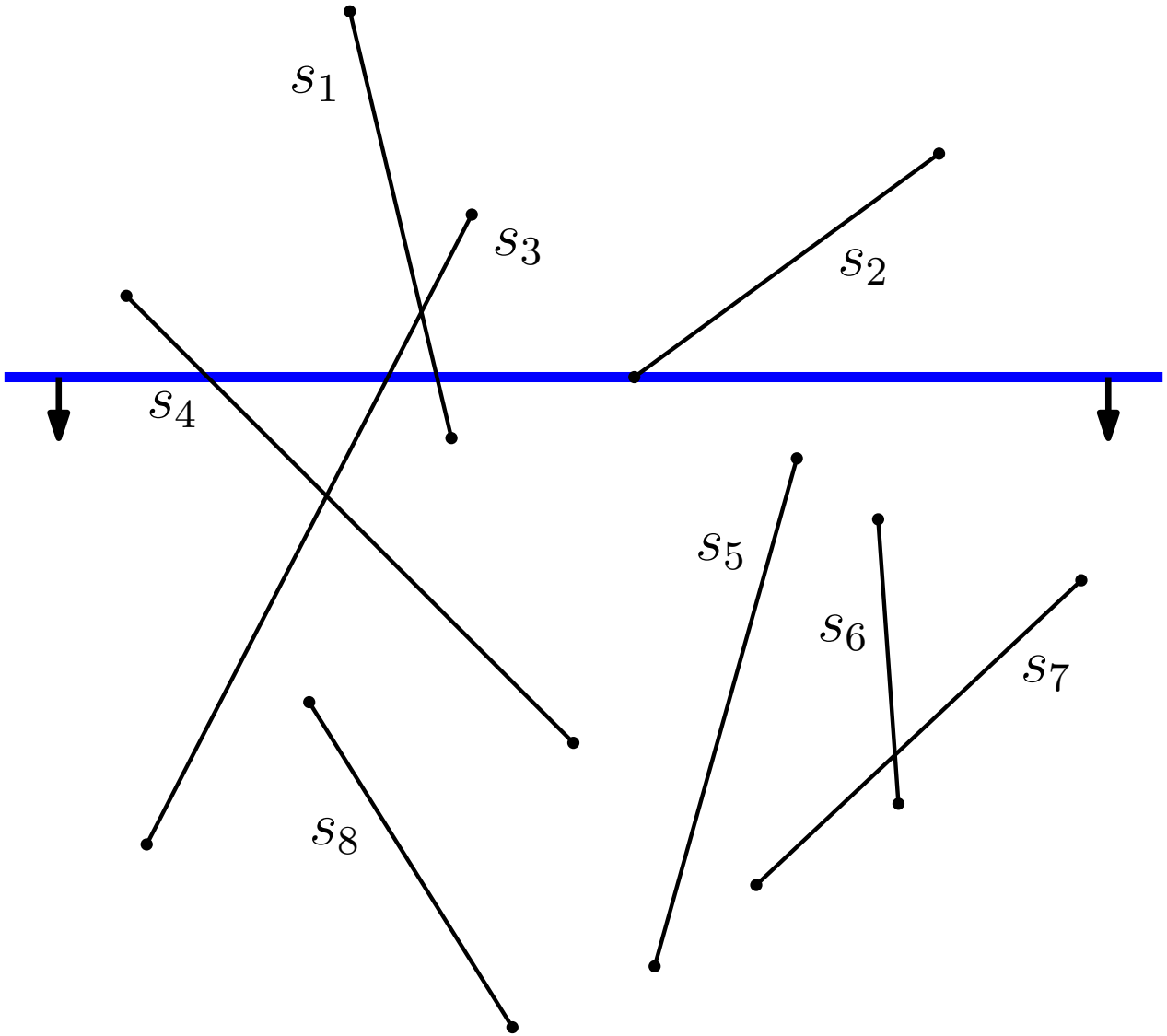
add s_3 between s_1 and s_2



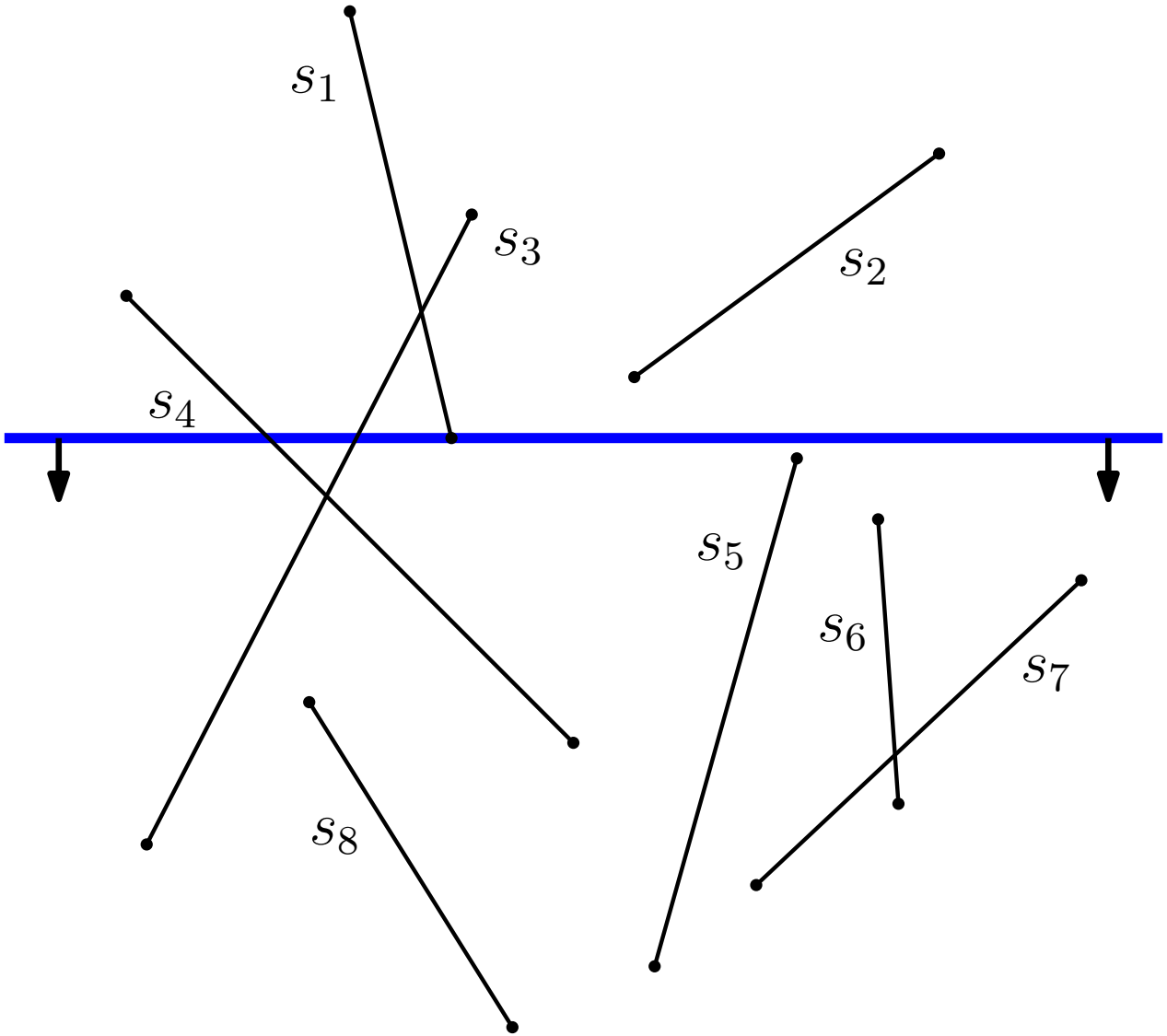
add s_4 before s_1



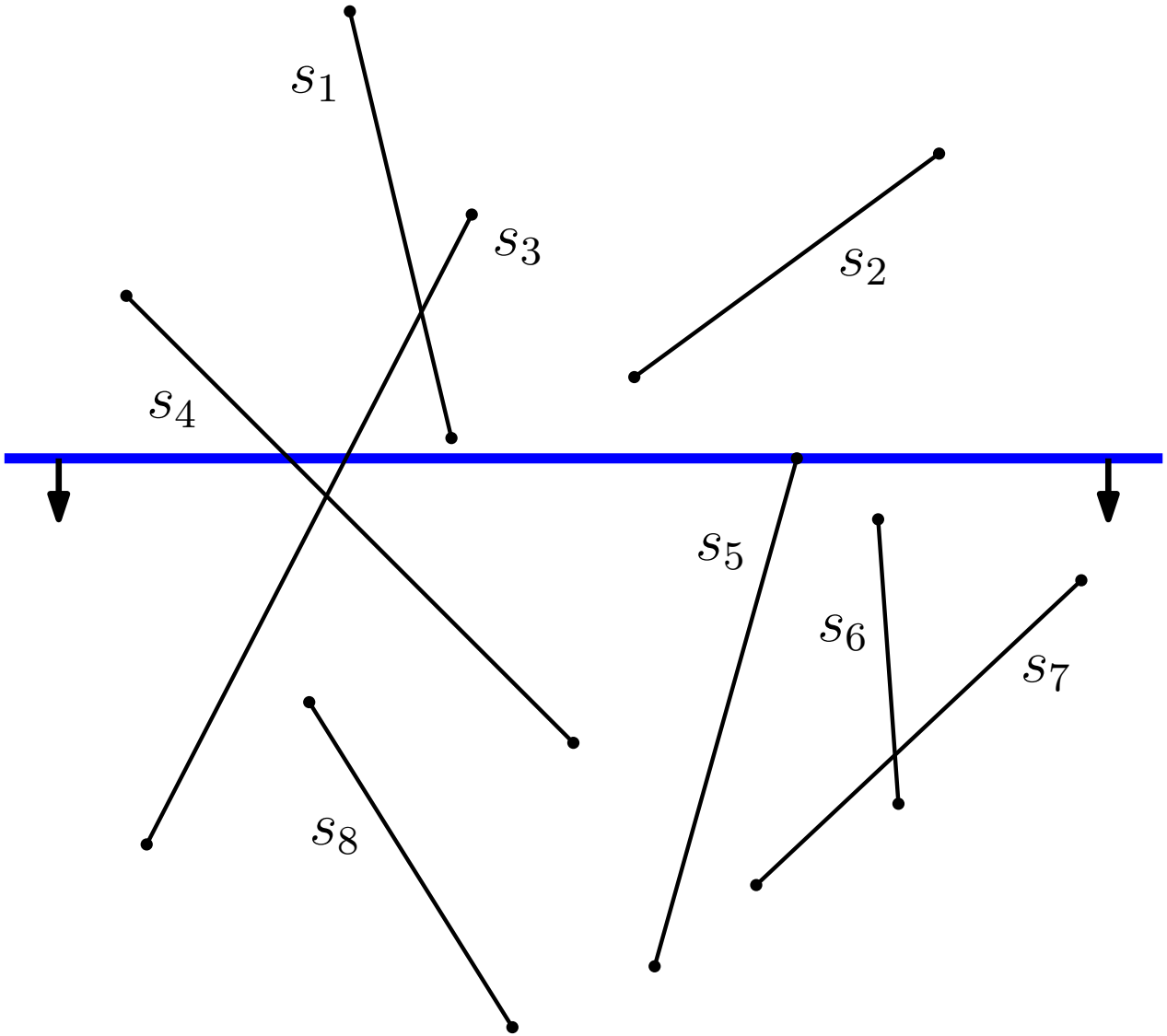
report intersection (s_1, s_2); swap s_1 and s_3



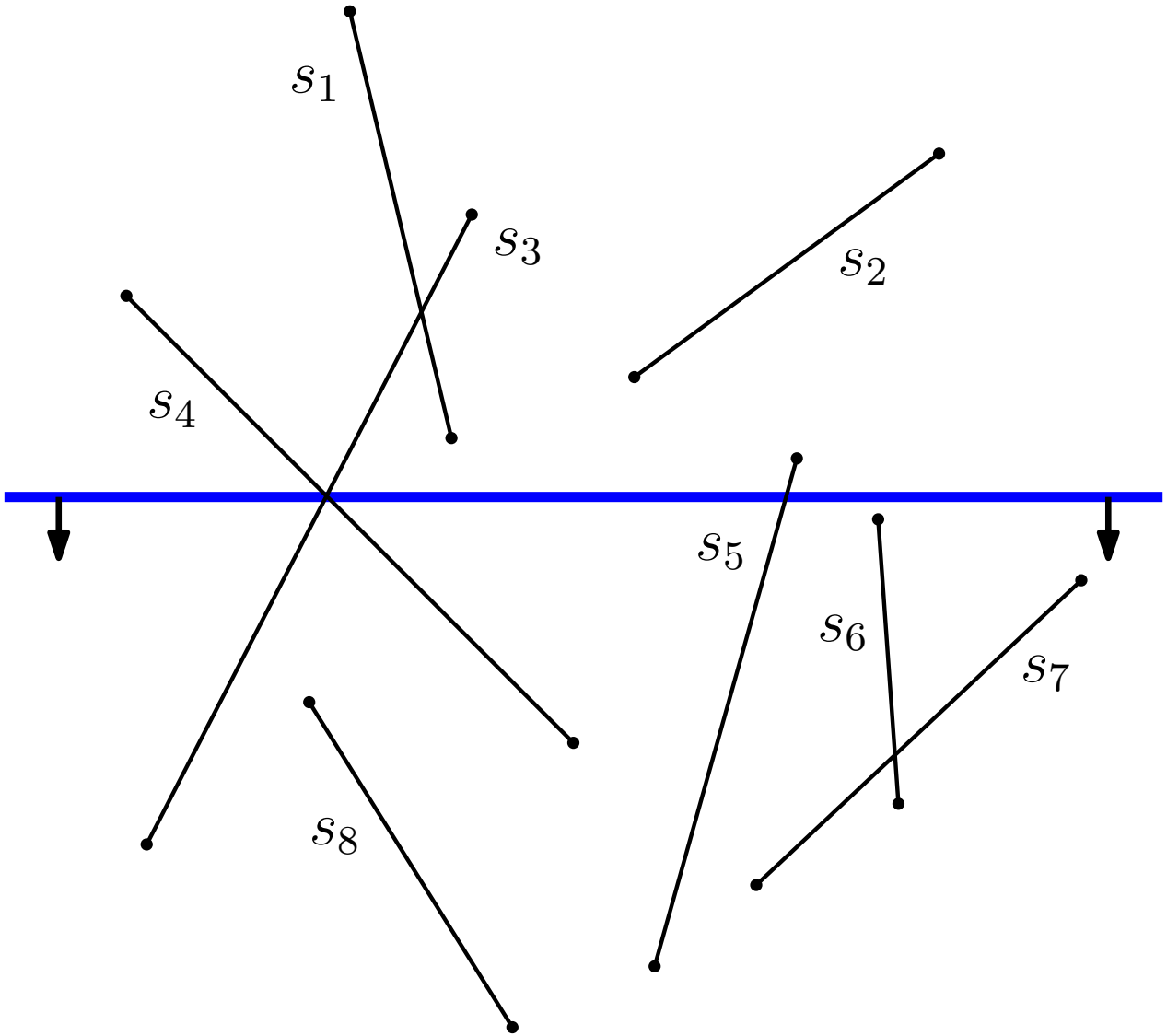
remove s_2



remove s_1



add s_5



report intersection (s_3, s_4); swap s_3 and s_4

... and so on ...

The events

When do the events happen? When the sweep line is

- at an upper endpoint of a line segment
- at a lower endpoint of a line segment
- at an intersection point of a line segment

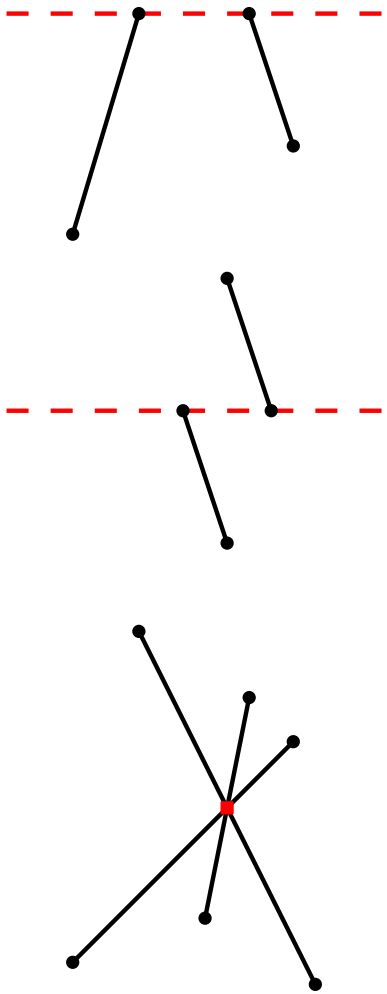
At each type, the **status** changes; at the third type **output** is found too

Assume no degenerate cases

We will at first exclude degenerate cases:

- No two endpoints have the same y-coordinate
- No more than two line segments intersect in a point
- ...

Question: Are there more degenerate cases?

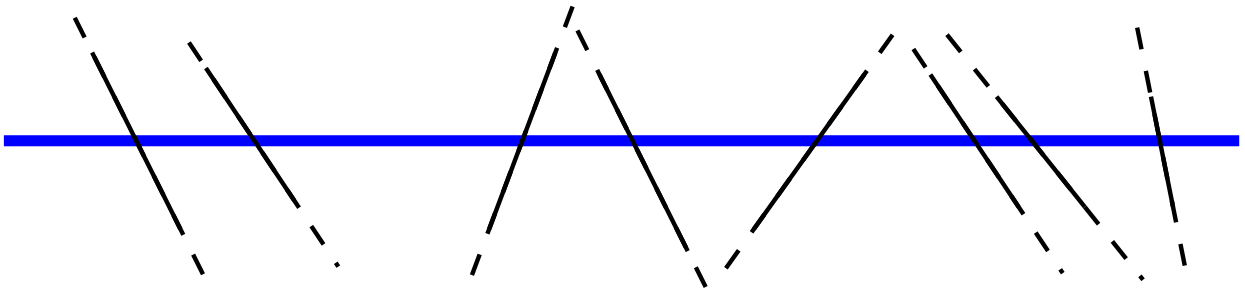


Event list and status structure

The event list is an abstract data structure that stores all events in the order in which they occur

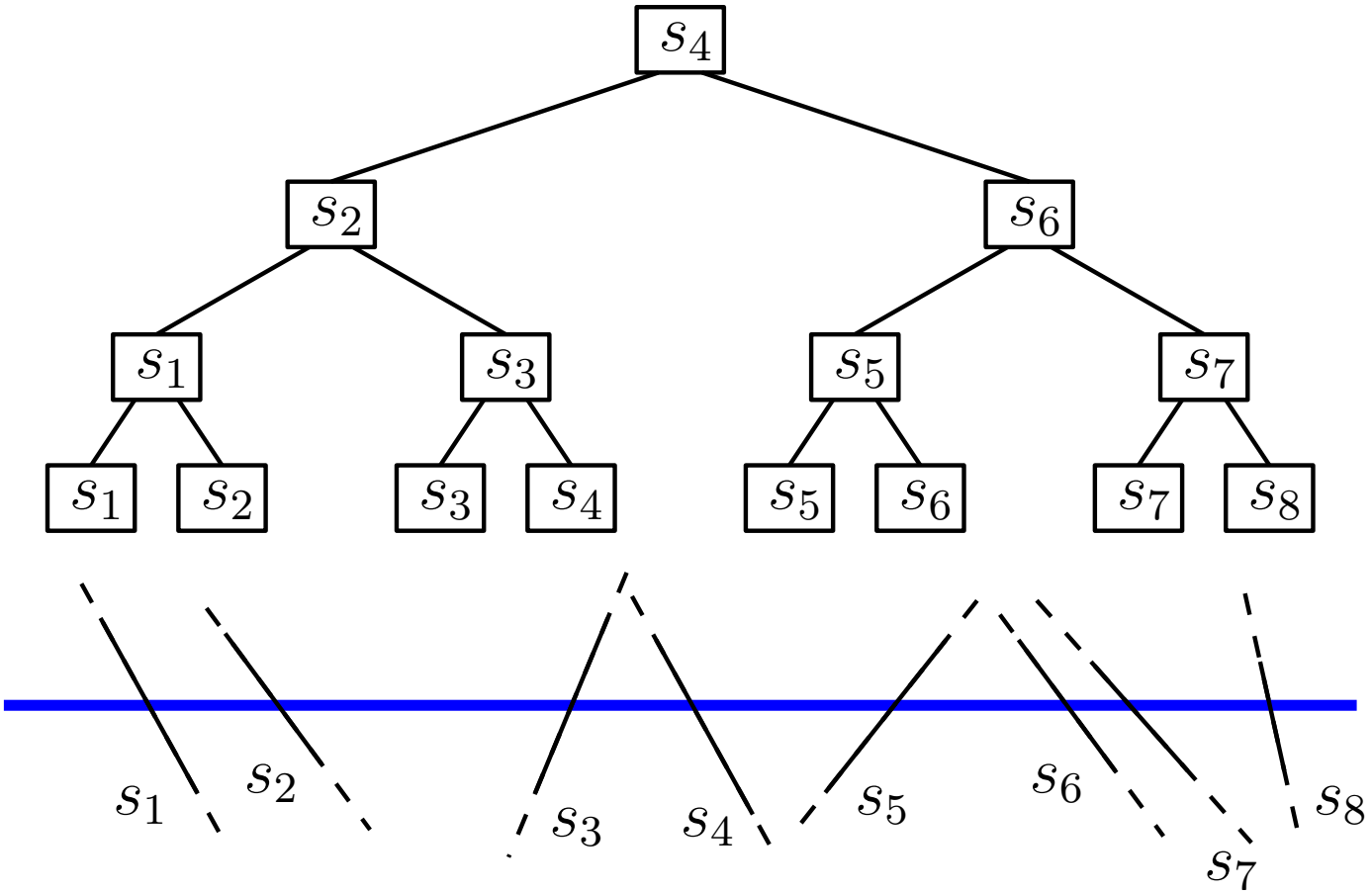
The status structure is an abstract data structure that maintains the current status

Here: The status is the subset of currently intersected line segments in the order of intersection by the sweep line

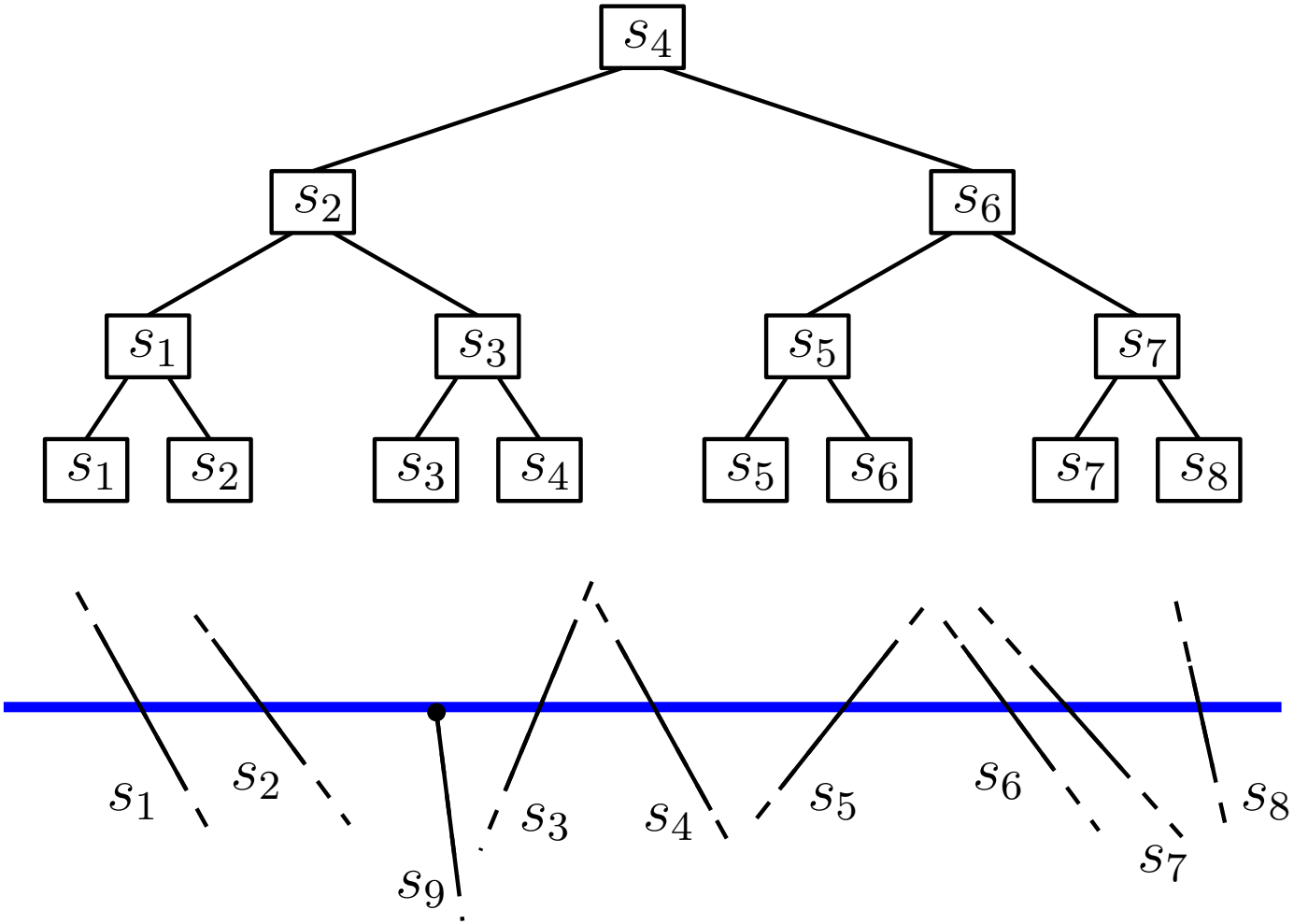


Status structure

We use a balanced binary search tree with the line segments in the leaves as the status structure

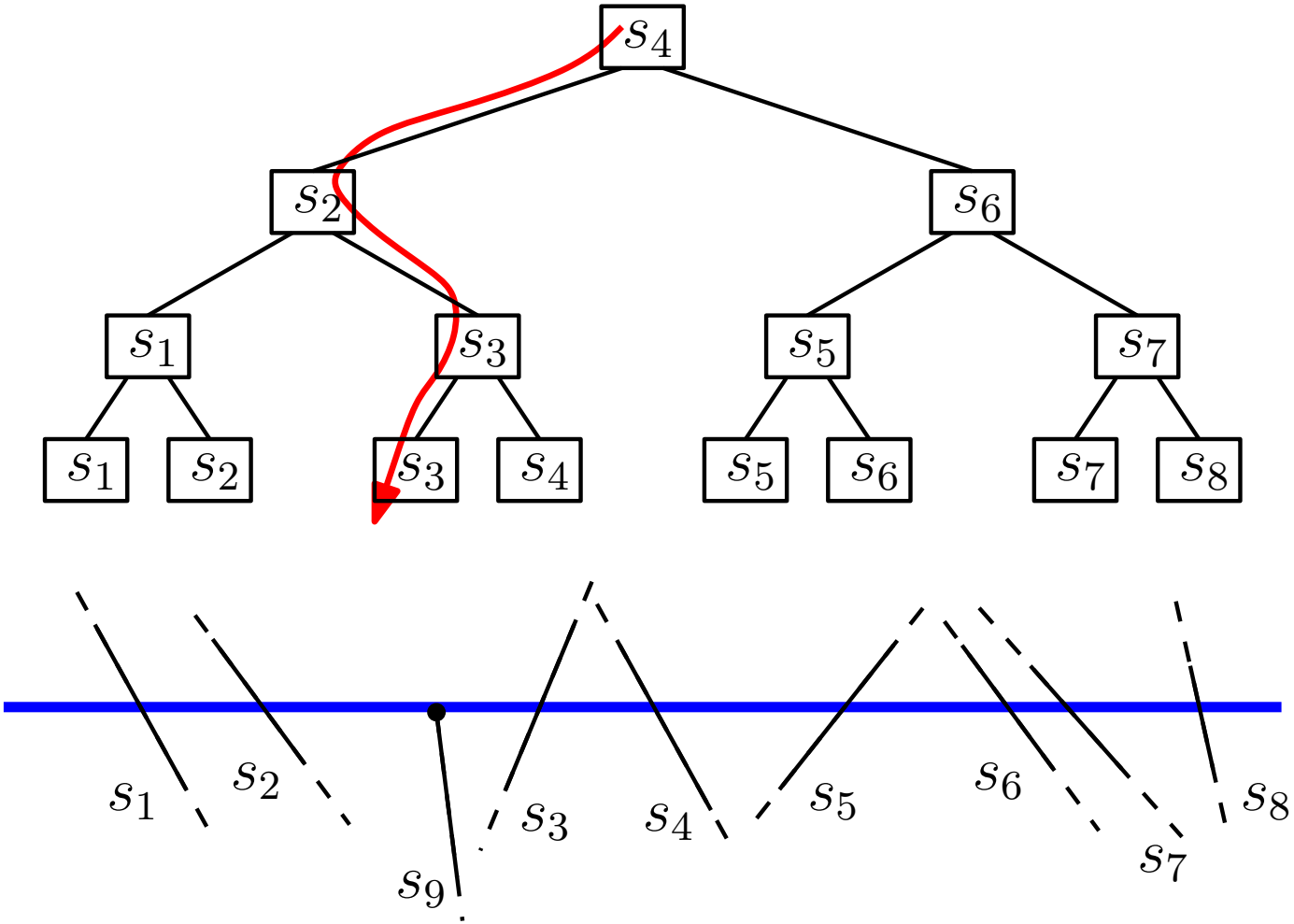


Status structure



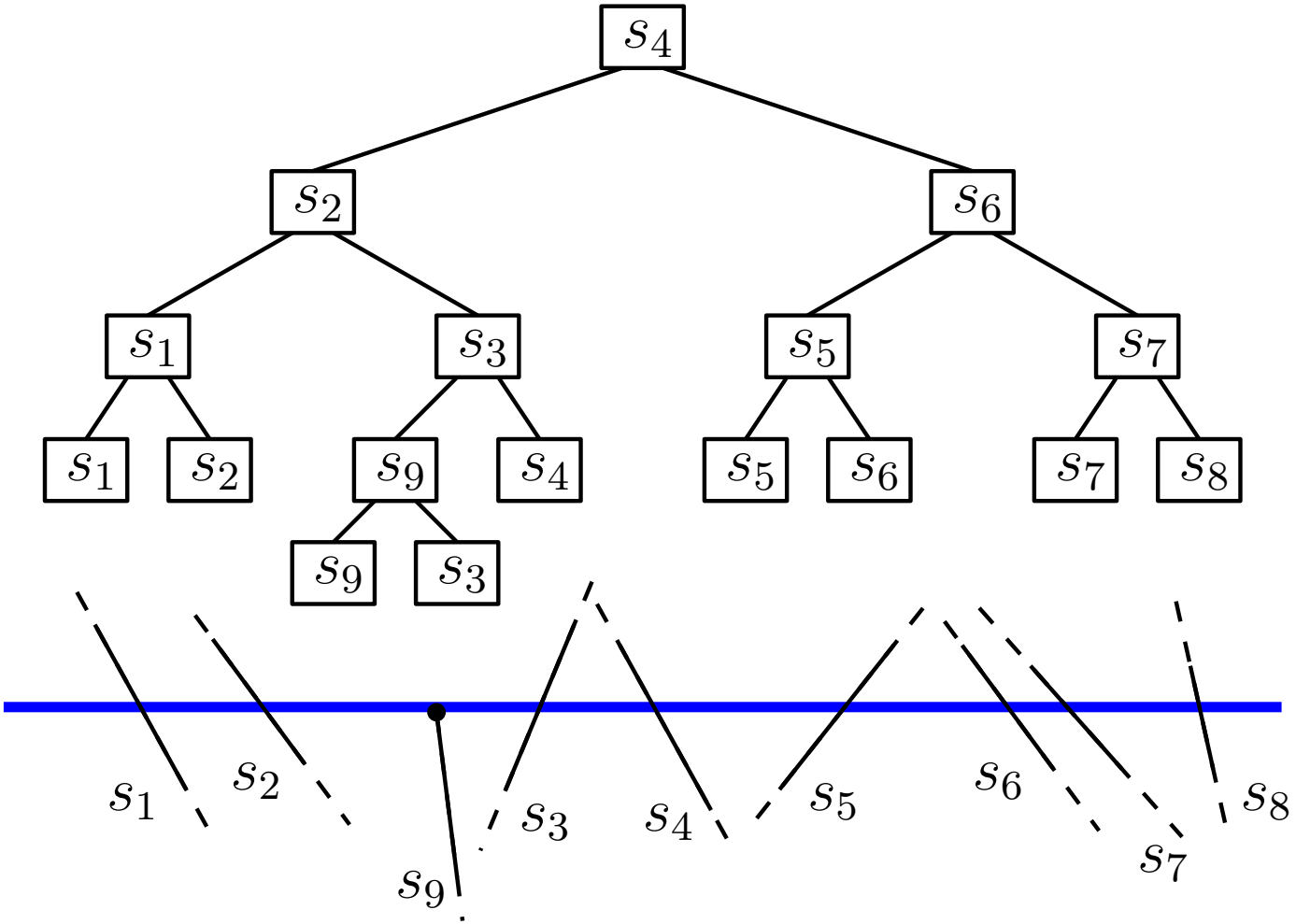
Upper endpoint: search, and insert

Status structure



Upper endpoint: search, and insert

Status structure



Upper endpoint: search, and insert

Status structure

Sweep line reaches lower endpoint of a line segment: delete from the status structure

Sweep line reaches intersection point: swap two leaves in the status structure (and update information on the search paths)

Finding events

Before the sweep algorithm starts, we know all upper endpoint events and all lower endpoint events

But: How do we know intersection point events???
(those we were trying to find ...)

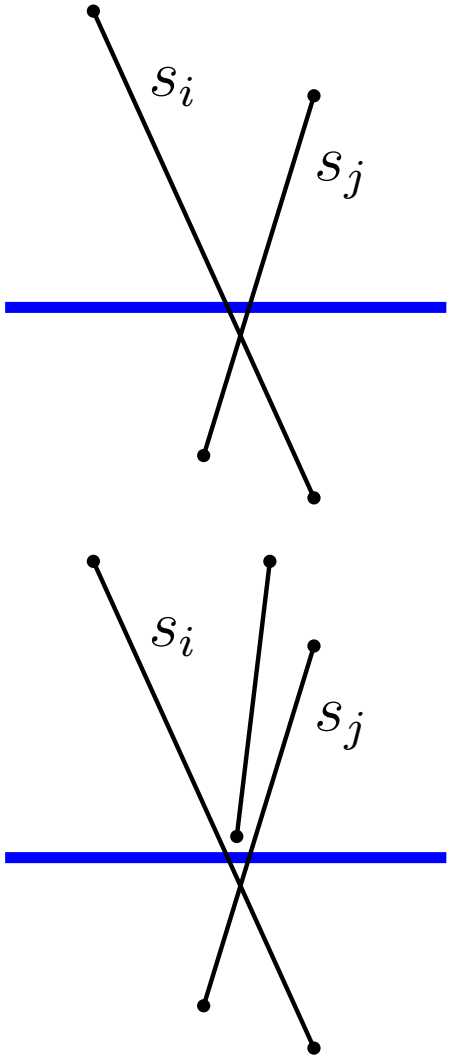
Recall: Two line segments can only intersect if they are horizontal neighbors

Finding events

Lemma: Two line segments s_i and s_j can only intersect after (= below) they have become horizontal neighbors

Proof: Just imagine that the sweep line is ever so slightly above the intersection point of s_i and s_j, but below any other event $\square$

Also: some earlier (= higher) event made s_i and s_j horizontally adjacent!!!



Event list

The event list must be a balanced binary search tree, because during the sweep, we discover new events that will happen later

We know upper endpoint events and lower endpoint events beforehand; we find intersection point events when the involved line segments become horizontal neighbors

Structure of sweep algorithm

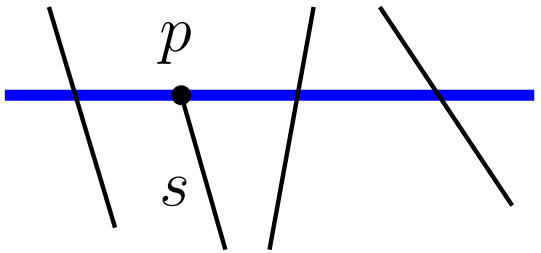
Algorithm FindIntersections(S)
Input. A set S of line segments in the plane.
Output. The intersection points of the segments in S, with for each intersection point the segments that contain it.

1. Initialize an empty event queue Q. Next, insert the segment endpoints into Q; when an upper endpoint is inserted, the corresponding segment should be stored with it
2. Initialize an empty status structure T
3. **while** Q is not empty
4. **do** Determine next event point p in Q and delete it
5. HandleEventPoint(p)

Event handling

If the event is an upper endpoint event, and s is the line segment that starts at p:

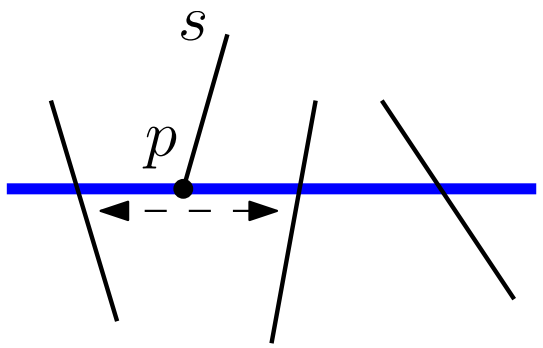
1. Search with p in T, and insert s
2. If s intersects its left neighbor in T, then determine the intersection point and insert it Q
3. If s intersects its right neighbor in T, then determine the intersection point and insert it Q



Event handling

If the event is a lower endpoint event, and s is the line segment that ends at p:

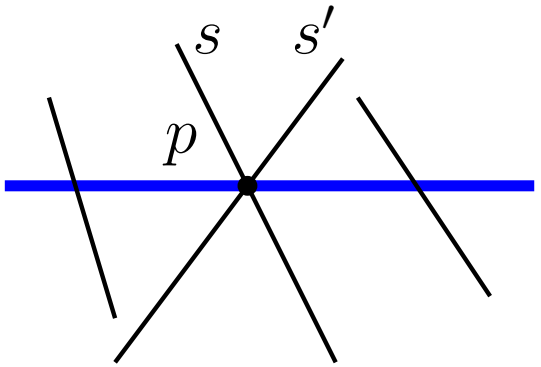
1. Search with p in T, and delete s
2. Let s_l and s_r be the left and right neighbors of s in T (before deletion). If they intersect *below the sweep line*, then insert their intersection point as an event in Q



Event handling

If the event is an intersection point event where s and s' intersect at p:

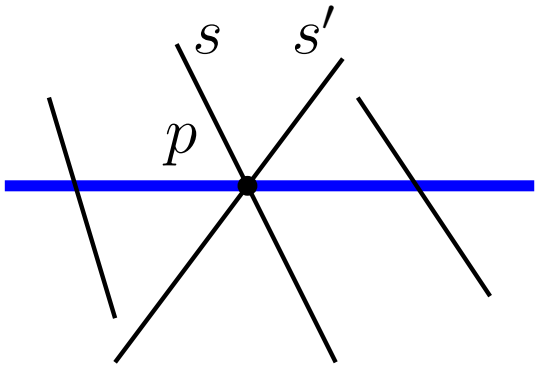
1. ...
2. ...
3. ...
4. ...



Event handling

If the event is an intersection point event where s and s' intersect at p:

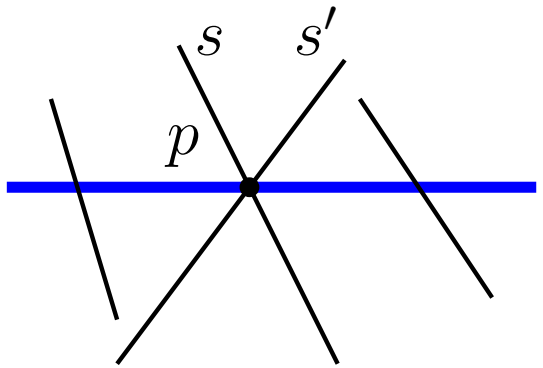
1. Exchange s and s' in T
2. ...
3. ...
4. ...



Event handling

If the event is an intersection point event where s and s' intersect at p:

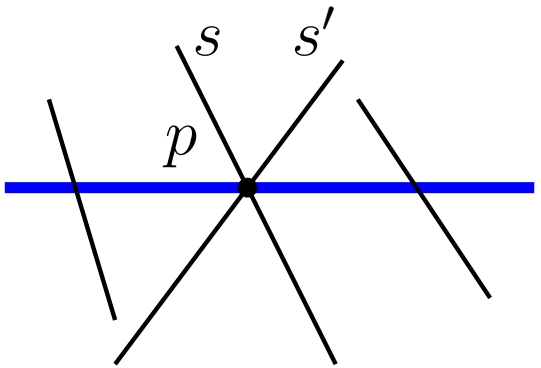
1. Exchange s and s' in T
2. If s' and its new left neighbor in T intersect below the sweep line, then insert this intersection point in Q
3. ...
4. ...



Event handling

If the event is an intersection point event where s and s' intersect at p:

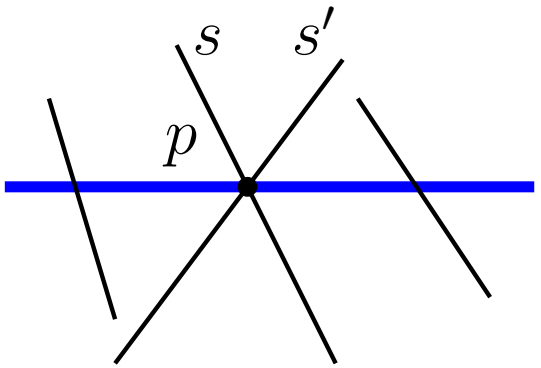
1. Exchange s and s' in T
2. If s' and its new left neighbor in T intersect below the sweep line, then insert this intersection point in Q
3. If s and its new right neighbor in T intersect below the sweep line, then insert this intersection point in Q
4. ...



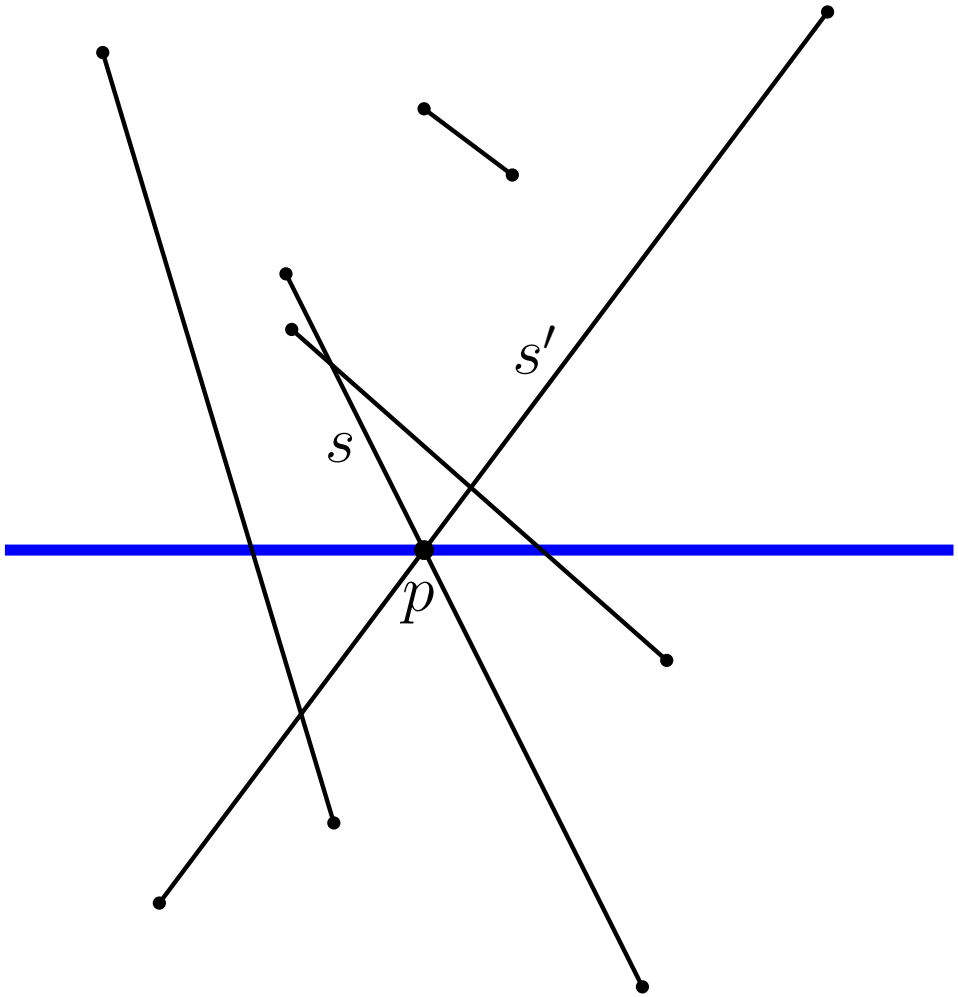
Event handling

If the event is an intersection point event where s and s' intersect at p:

1. Exchange s and s' in T
2. If s' and its new left neighbor in T intersect below the sweep line, then insert this intersection point in Q
3. If s and its new right neighbor in T intersect below the sweep line, then insert this intersection point in Q
4. Report the intersection point



Event handling



Can it be that new horizontal neighbors already intersected above the sweep line?

Can it be that we insert a newly detected intersection point event, but it already occurs in Q?

Efficiency

How much time to handle an event?

At most one search in T and/or one insertion, deletion, or swap

At most twice finding a neighbor in T

At most one deletion from and two insertions in Q

Since T and Q are balanced binary search trees, handling an event takes only $O(\log n)$ time

Efficiency

How many events?

- $2n$ for the upper and lower endpoints
- k for the intersection points, if there are k of them

In total: $O(n+k)$ events

Efficiency

Initialization takes $O(n \log n)$ time (to put all upper and lower endpoint events in Q)

Each of the $O(n+k)$ events takes $O(\log n)$ time

The algorithm takes $O(n \log n + k \log n)$ time

If $k = O(n)$, then this is $O(n \log n)$

Note that if k is really large, the brute force $O(n^2)$ time algorithm is more efficient

Efficiency

Question: How much storage does the algorithm take?

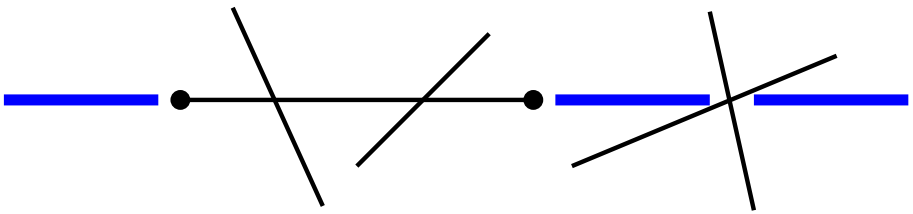
Efficiency

Question: Given that the event list is a binary tree that may store $O(k) = O(n^2)$ events, is the efficiency in jeopardy?

Degenerate cases

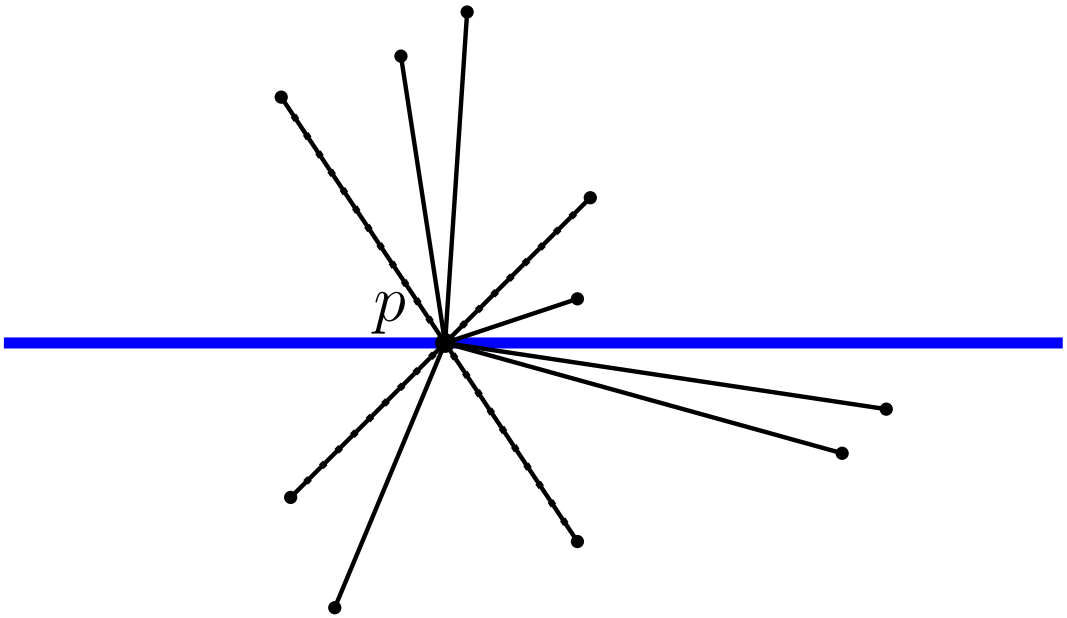
How do we deal with degenerate cases?

For two different events with the same y-coordinate, we treat them from left to right $\Rightarrow$ the "upper" endpoint of a horizontal line segment is its left endpoint



Degenerate cases

How about multiply coinciding event points?



Let $U(p)$ and $L(p)$ be the line segments that have p as upper and lower endpoint, and $C(p)$ the ones that contain p

Question: How do we handle this multi-event?

Conclusion

For every sweep algorithm:

- Define the status
- Choose the status structure and the event list
- Figure out how events must be handled (with sketches!)
- To analyze, determine the number of events and how much time they take

Then deal with degeneracies