

# Computational Geometry



- Algorithms for geometric problems.
- Relevant for Games, Computer Graphics, Robotics and GIS.
- We focus on “combinatorial computational geometry”, that is, the objects under study are basic geometrical objects, such as points, lines segments, polygons and polyhedra.

# Computational Geometry



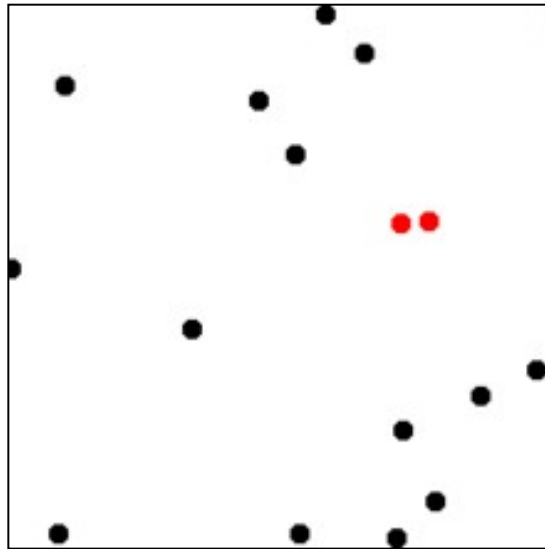
## Readings:

- S. Skiena, M. Revilla, Programming Challenges, Chapter 13
- S. Skiena, The Algorithm Design Manual, Chapter 17
- T. Cormen et al., Introduction to Algorithms, Chapter 33
- David Goldberg. "What Every Computer Scientist Should Know About Floating-Point Arithmetic". ACM Computing Surveys, 23 (1): 5–48, 1991 ([link](#))

# Computational Geometry



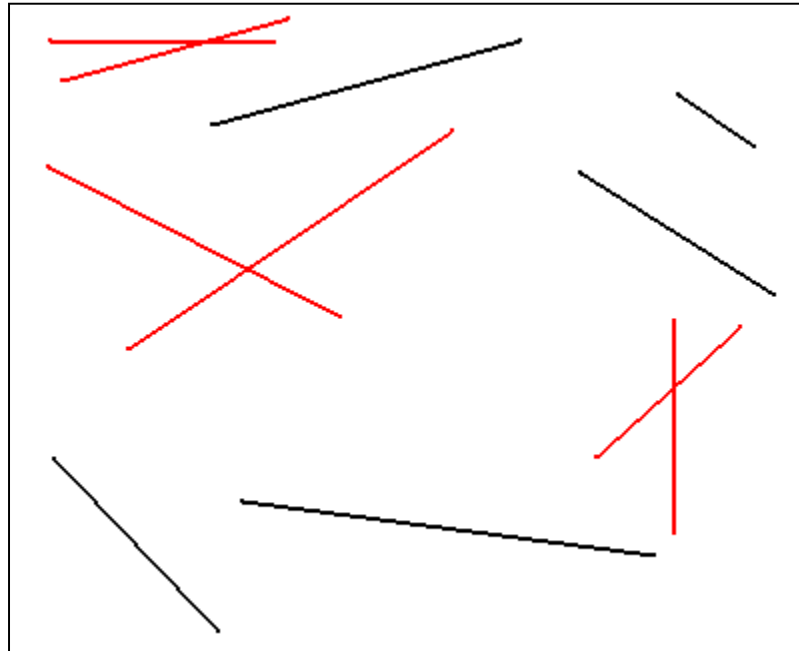
- Closest pair problem



# Computational Geometry



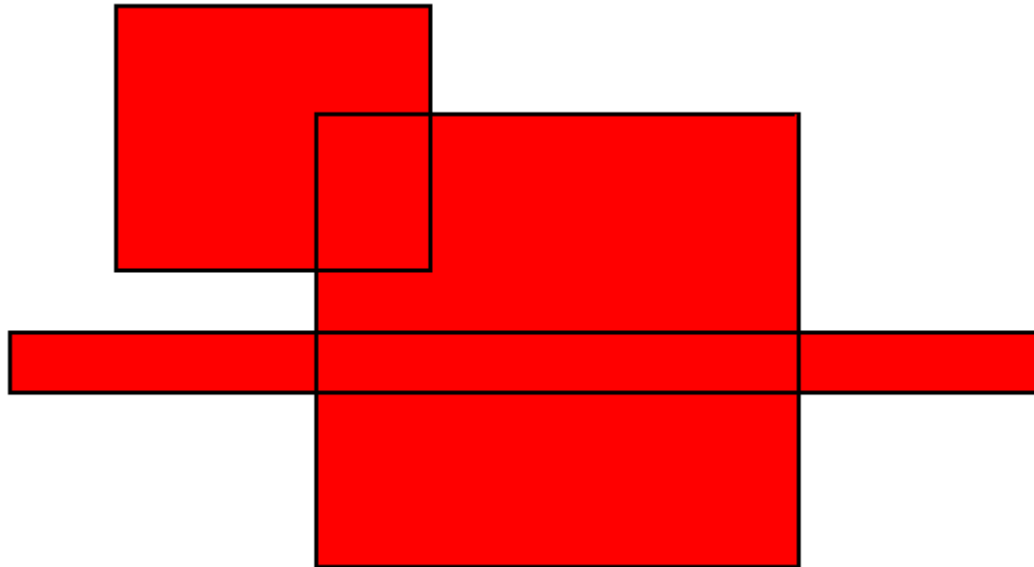
- Line intersection problem



# Computational Geometry



- Area of the union of rectangles problem



# Computational Geometry



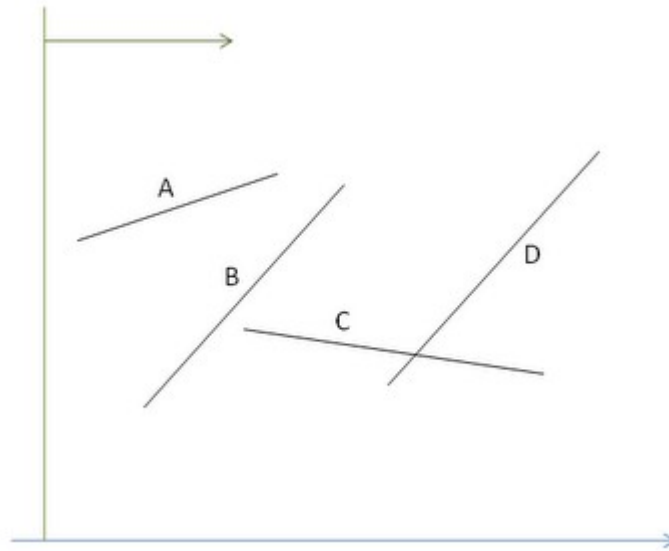
- Special strategies for computational geometry:
  - Line Sweep
    - A line sweeps the plane, stopping at some points.
    - Perform geometric operations at each stop.
    - Assumes that the objects are sorted in one dimension.
    - The solution is obtained once the line passed all objects.
    - For 3D, it is called plane sweep.

# Computational Geometry



- Special strategies for computational geometry:

- Line Sweep

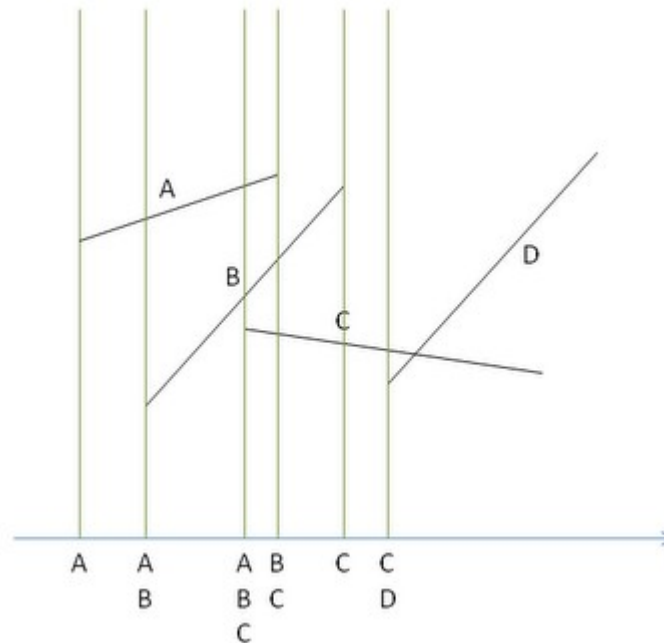


# Computational Geometry



- Special strategies for computational geometry:

- Line Sweep



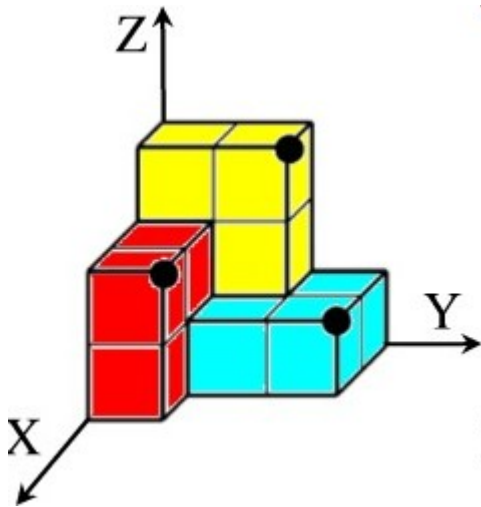


# Computational Geometry



- Special strategies for computational geometry:

- Plane Sweep

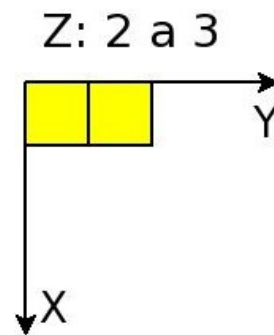
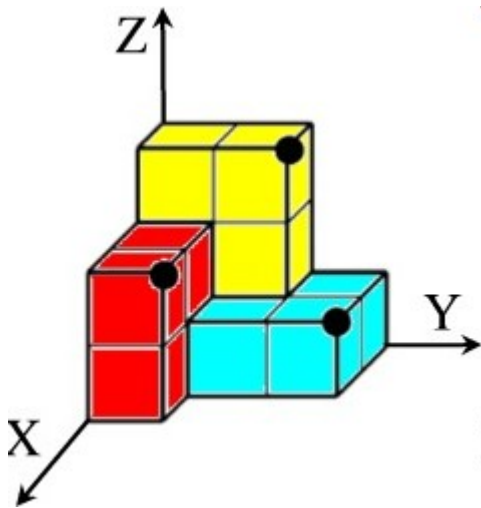


# Computational Geometry



- Special strategies for computational geometry:

- Plane Sweep

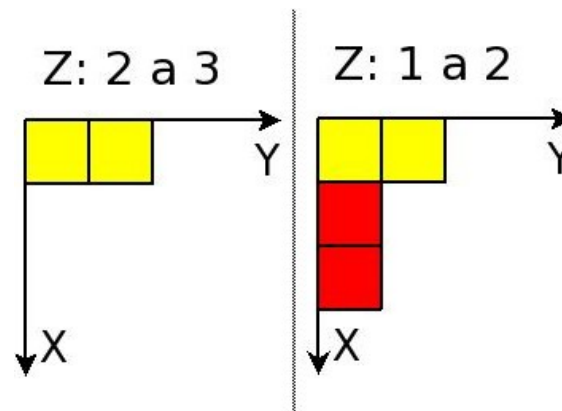
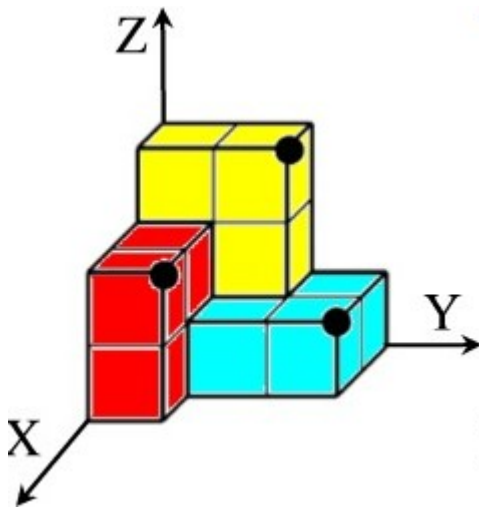


# Computational Geometry



- Special strategies for computational geometry:

- Plane Sweep

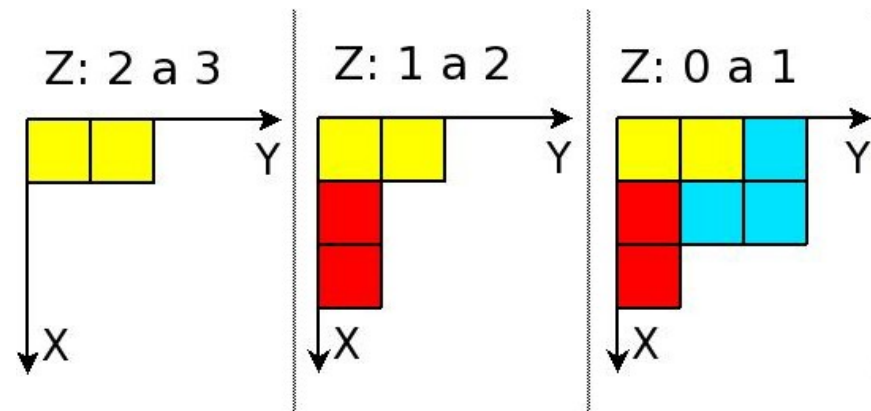
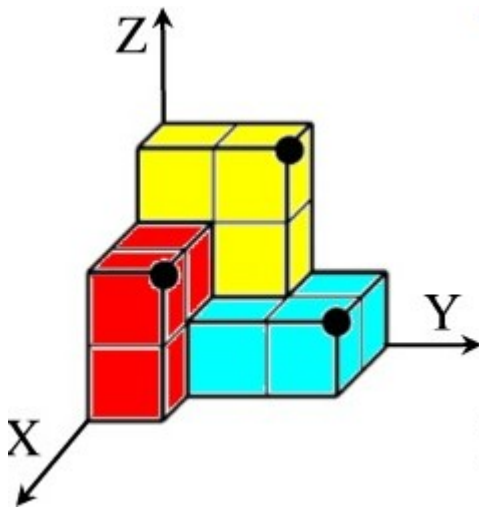


# Computational Geometry



- Special strategies for computational geometry:

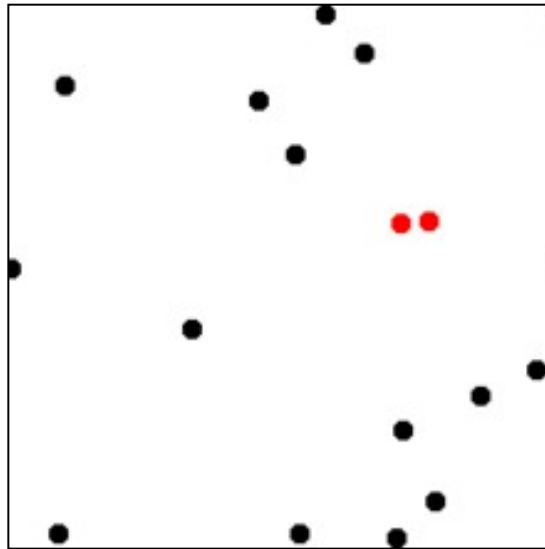
- Plane Sweep



# Computational Geometry



- Closest pair problem (given  $n$  points in 2D)



# Computational Geometry



- Closest pair problem
- Brute force:  $O(n^2)$ 
  - Compute all the distances between the  $n$  points and pick the pair with the smallest distance.

# Computational Geometry



- Closest pair problem
- Divide-and-Conquer:  $O(n \log n)$

Step 1: Sort points along the x-coordinate.

Step 2: Divide: Split the points into 2 equal subsets, and solve recursively, the left and right subsets ( $\delta_1$  and  $\delta_2$ ).

Step 3: Merge: Find the minimum distance between points in the left and in the right subset ( $\delta_c$ ). The minimum distance is  $\min(\delta_1, \delta_2, \delta_c)$ . How to do it efficiently?

# Computational Geometry

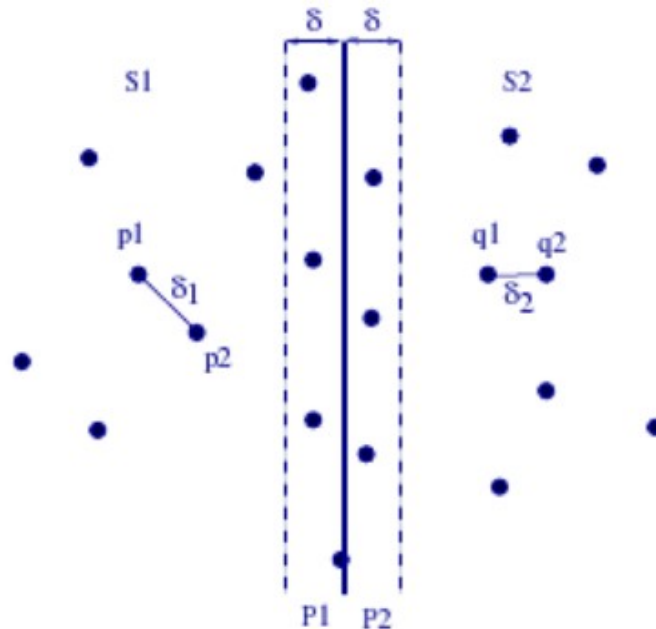


- Closest pair problem
- Divide-and-Conquer:  $O(n \log n)$

Step 3:

How to compute the minimum distance between the points in P1 and P2?

$$\delta = \min(\delta_1, \delta_2)$$





# Computational Geometry

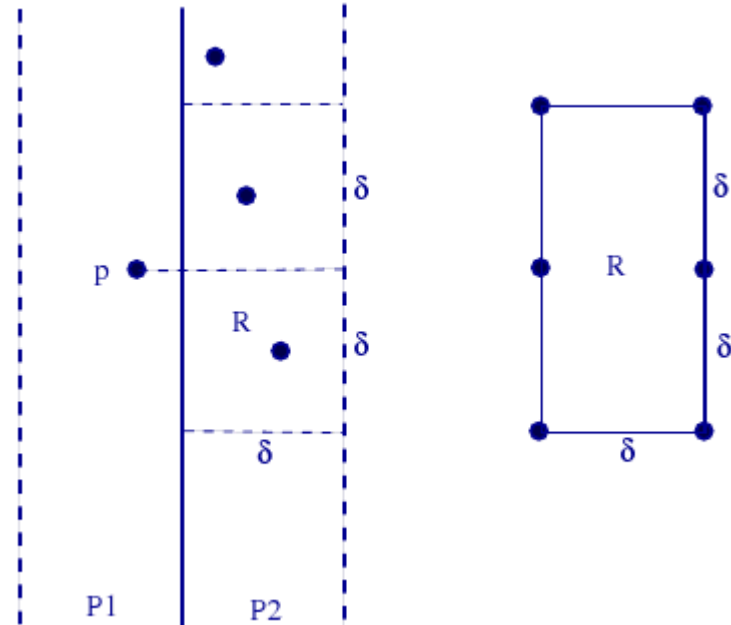


- Closest pair problem
- Divide-and-Conquer:  $O(n \log n)$

## Step 3:

For each point  $p$ , check neighbors at a distance at most  $\delta$  of  $p$

There can be at most 6 neighbors at a distance  $\delta$  of  $p$ .



# Computational Geometry

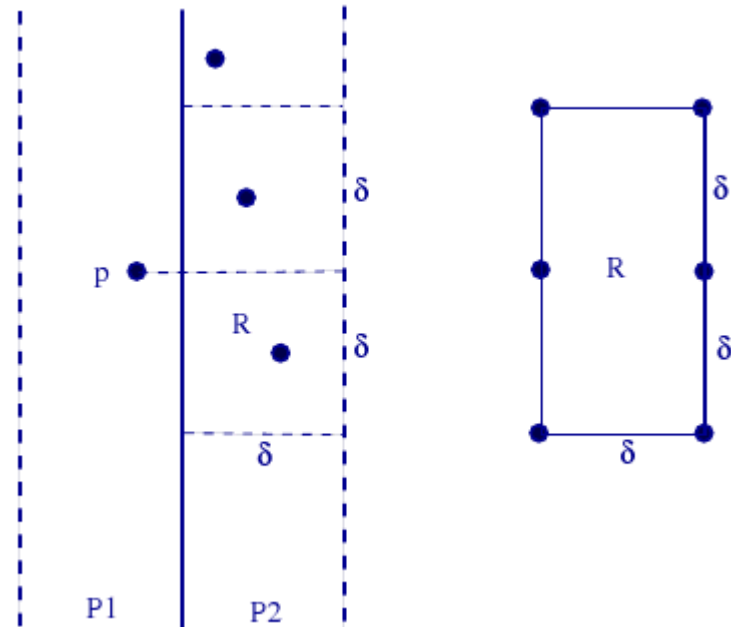


- Closest pair problem
- Divide-and-Conquer:  $O(n \log n)$

Step 3 in  $O(n)$ :

- Project points in P1 and P2 on the vertical line
- For each point, compute the minimum distance from the 6 neighbors

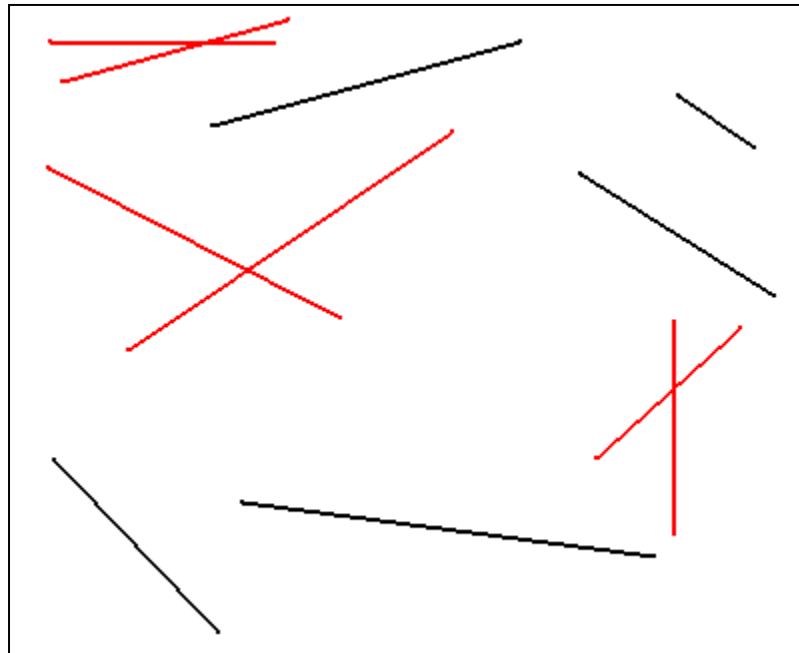
(All points must be sorted in y)



# Computational Geometry



- Line intersection problem (given  $n$  lines and  $k$  intersections)



# Computational Geometry

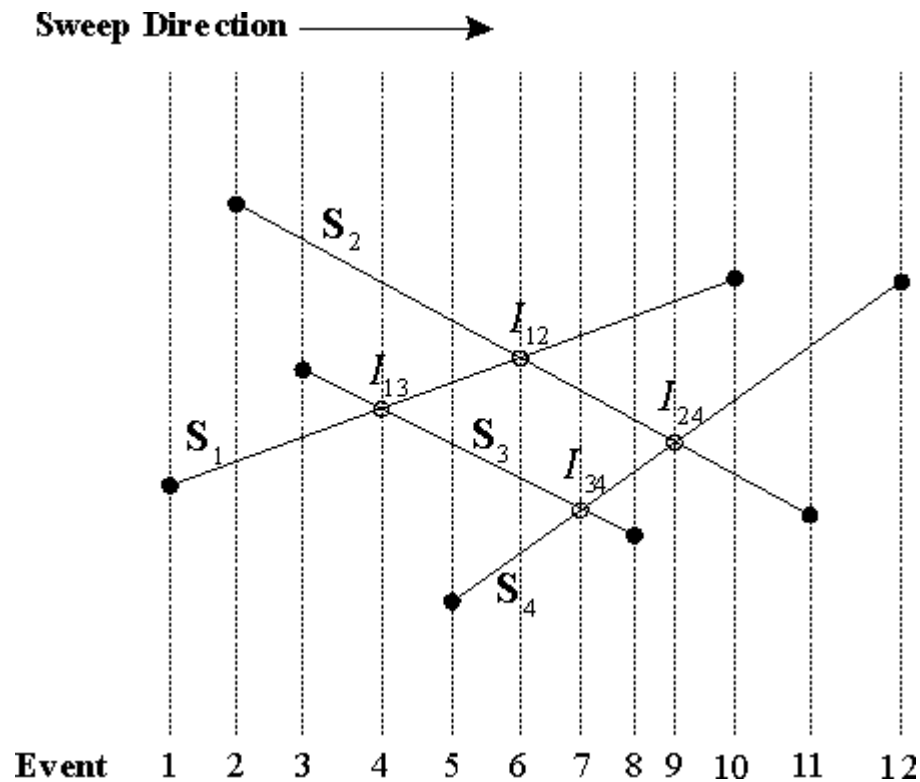


- Line intersection problem
- Brute force:  $O(n^2)$ 
  - Detect intersection between every pair of lines.

# Computational Geometry



- Line intersection problem



Bentley-Ottmann algorithm by sweep-line

# Computational Geometry

- Line intersection problem

Sweep line:  $O((n+k) \log n)$

Step 1: Sort  $2n$  points in x-coordinate and let  $Q = \{p_1, \dots, p_{2n}\}$

Step 2: From left to right, for each point  $p$  in  $Q$ :

Step 2.1: If  $p$  is left-endpoint of  $l$ , add intersection between  $l$  and its neighboring lines to  $Q$ .

Step 2.2: If  $p$  is right-endpoint of  $l$ , add intersection between its neighboring lines to  $Q$  (to the right of  $p$ ).

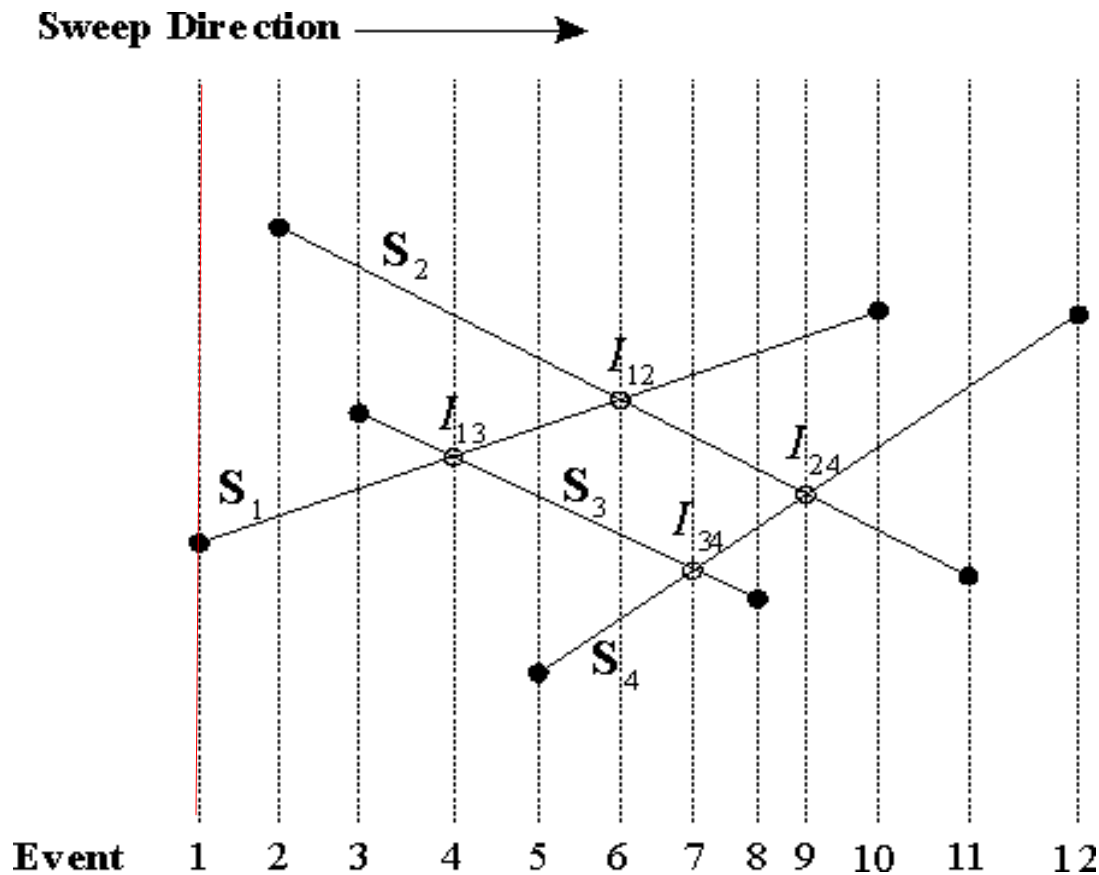
Step 2.3: If  $p$  is an intersection between  $l_1$  and  $l_2$ , add intersection between  $l_1$  and  $l_2$  and their neighboring lines to  $Q$ .

- $Q$  is a priority queue that keeps the line with the minimum x-coordinate
- Use a balanced binary tree to maintain the order of the segments at each step in the y-coordinate

# Computational Geometry



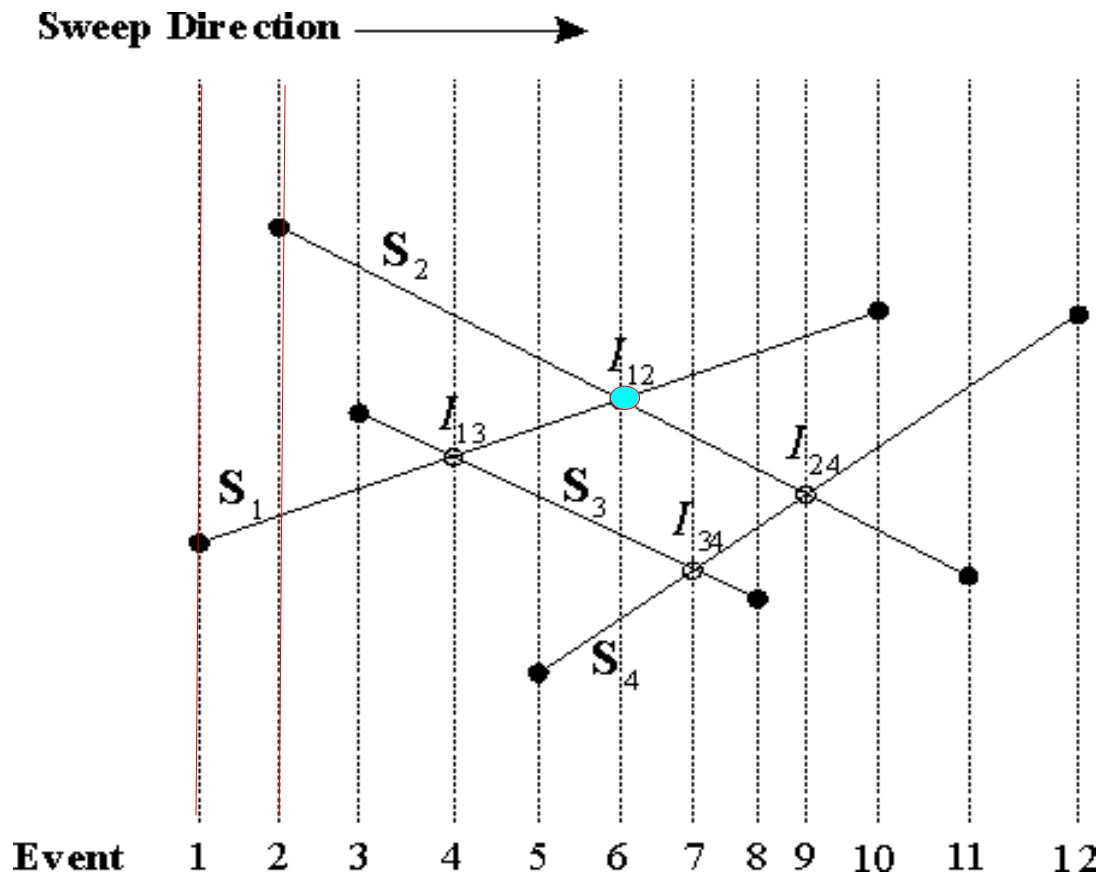
- Line intersection problem



# Computational Geometry



- Line intersection problem



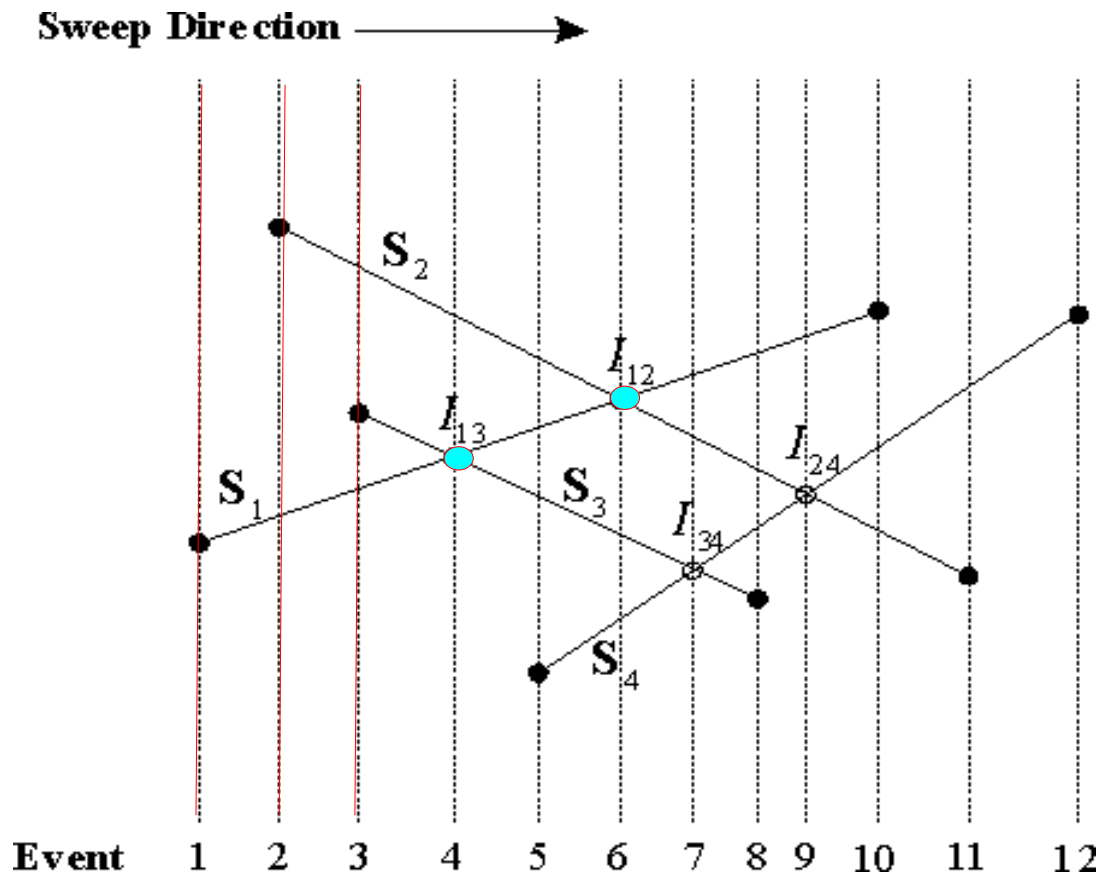
- Check intersection of  $S_1$  and  $S_2$
- Add intersection  $I_{12}$



# Computational Geometry



- Line intersection problem

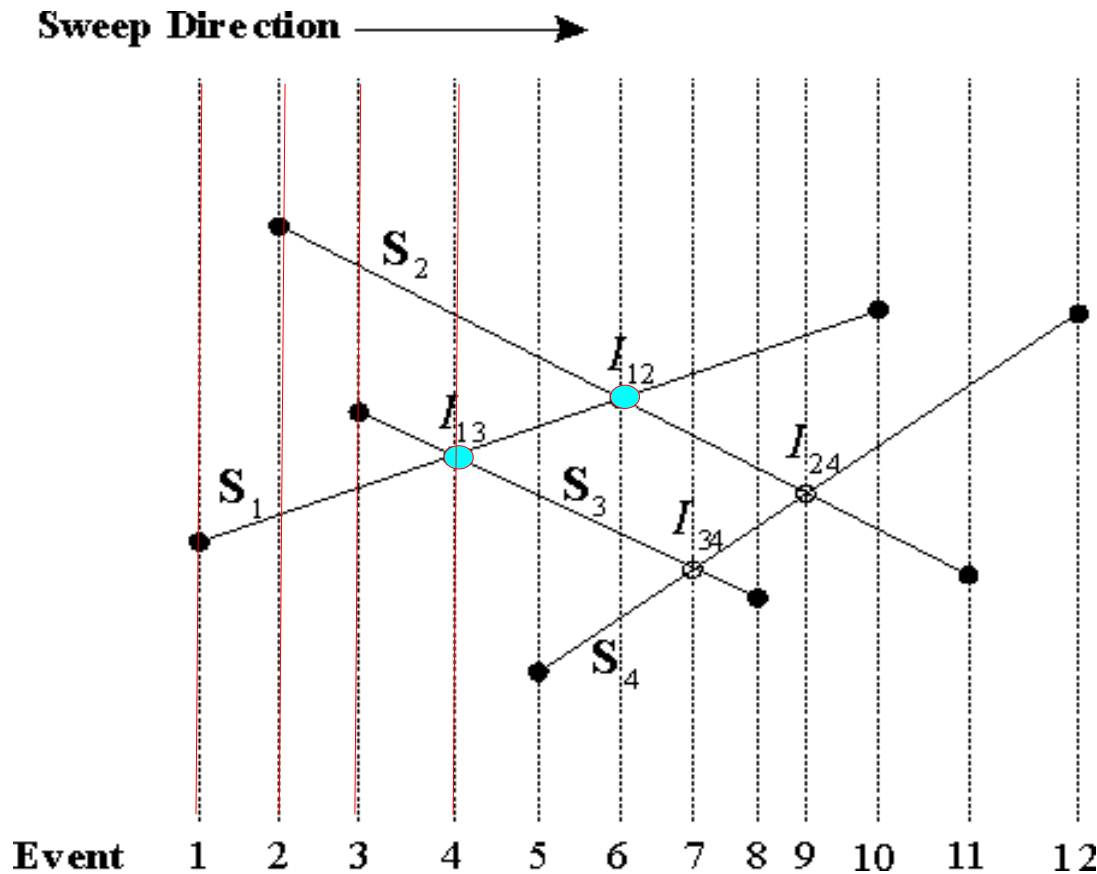


- Check intersection of  $S_3$  and  $S_1$  and of  $S_3$  and  $S_2$
- Add intersection  $I_{13}$

# Computational Geometry



- Line intersection problem

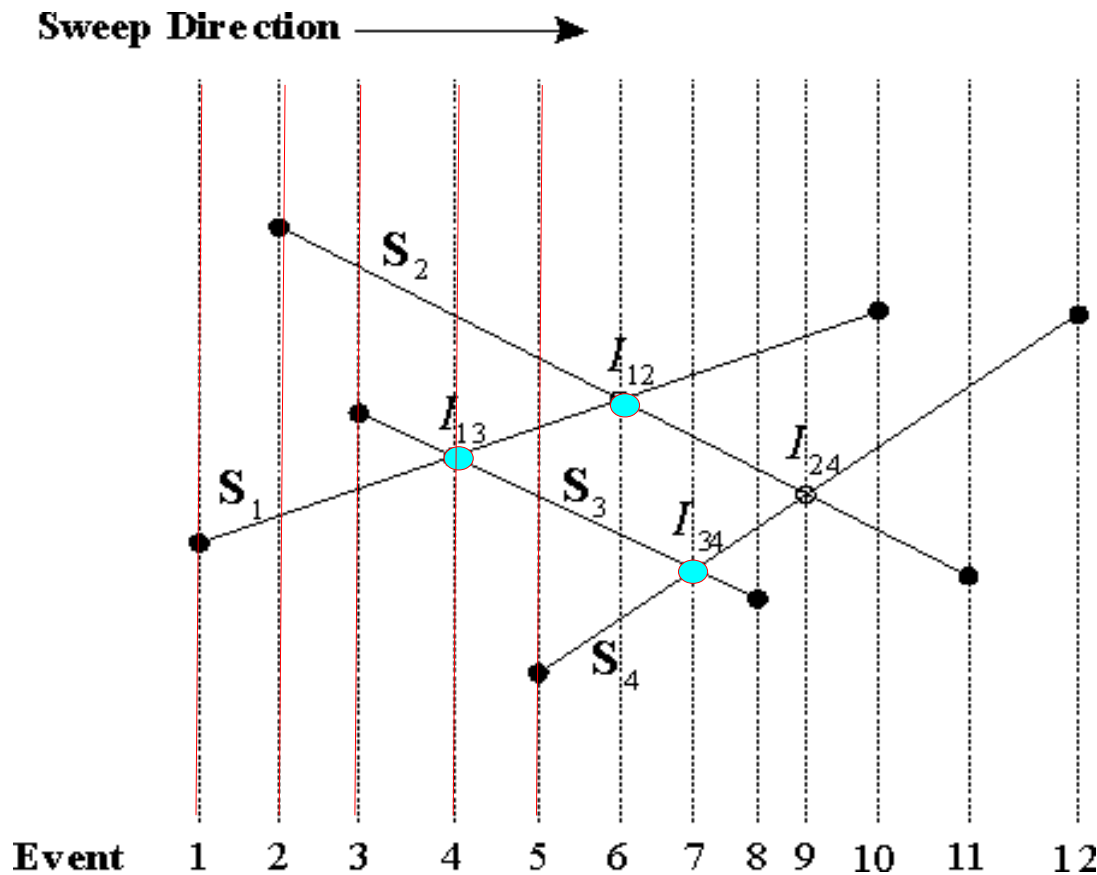


- Swap position of  $S_1$  and  $S_3$
- Check intersection of  $S_1$  and  $S_2$

# Computational Geometry



- Line intersection problem

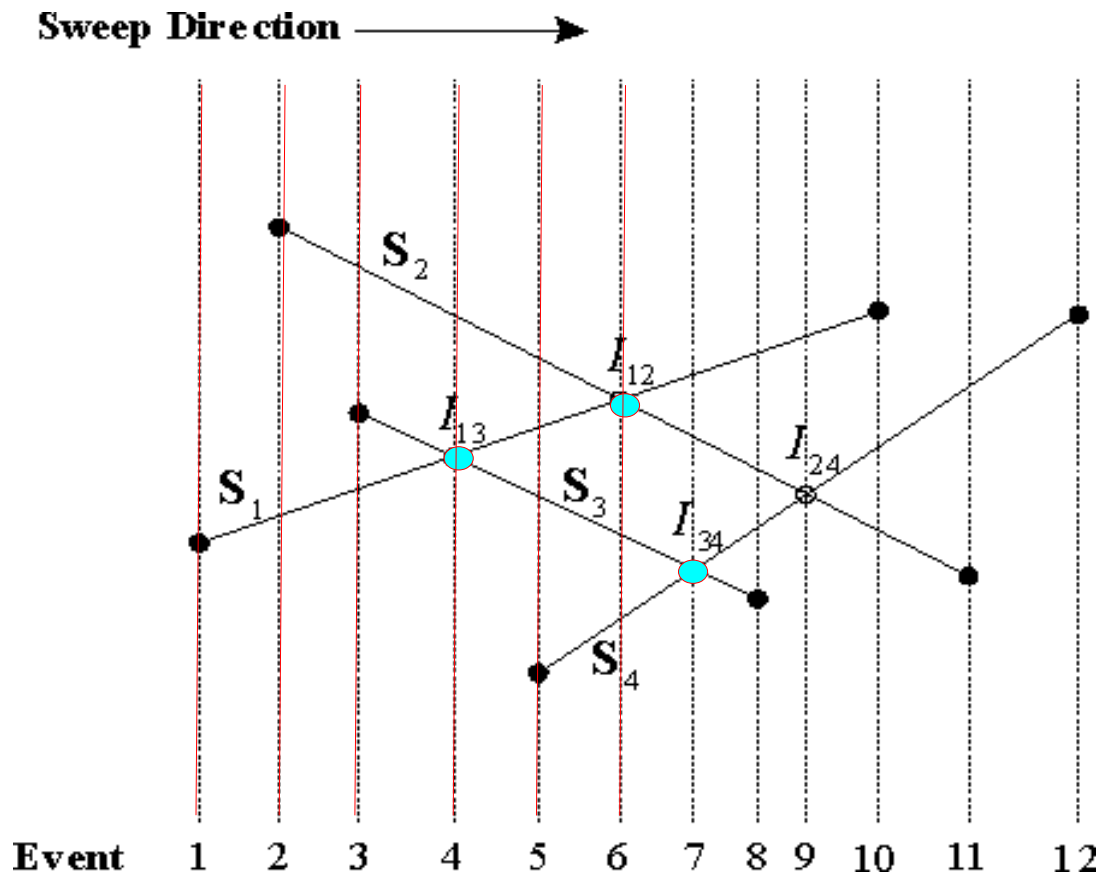


- Check intersection of  $S_4$  and  $S_3$
- Add intersection  $I_{34}$

# Computational Geometry



- Line intersection problem

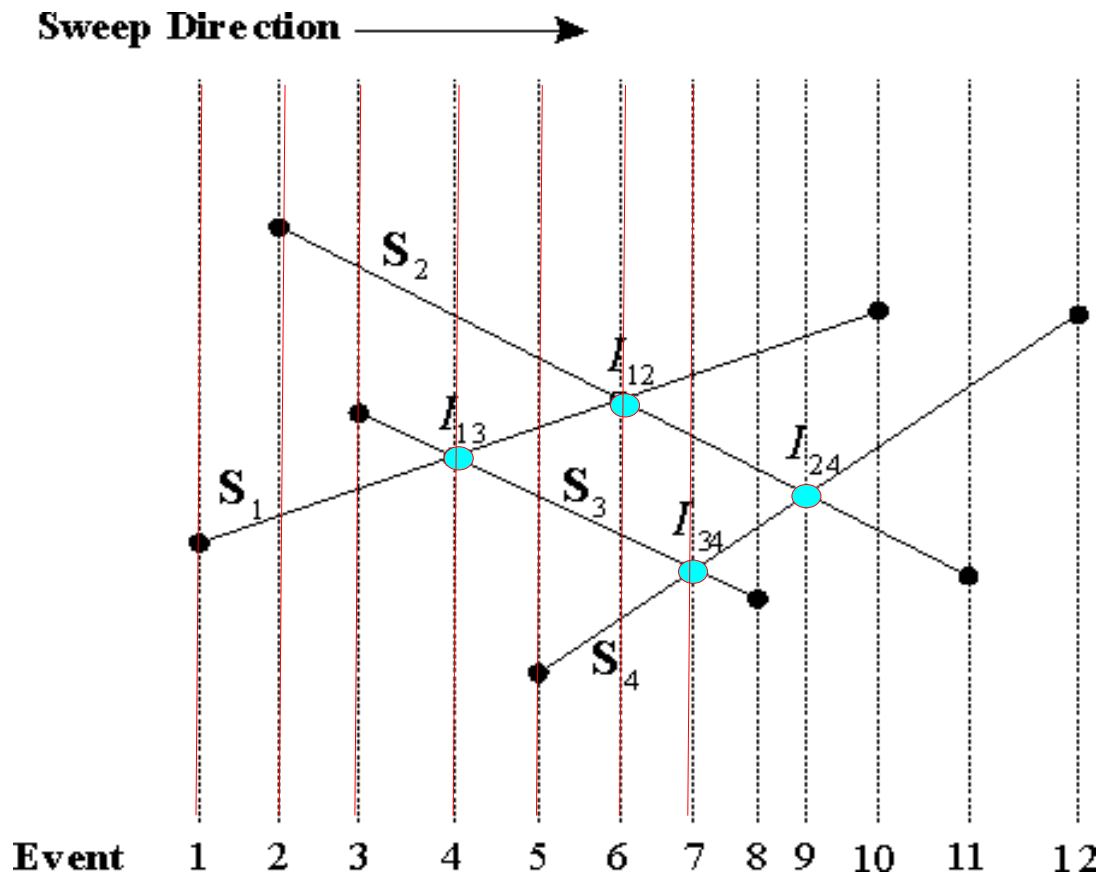


- Swap position of  $S_1$  and  $S_2$
- Check intersection of  $S_2$  and  $S_3$

# Computational Geometry



- Line intersection problem

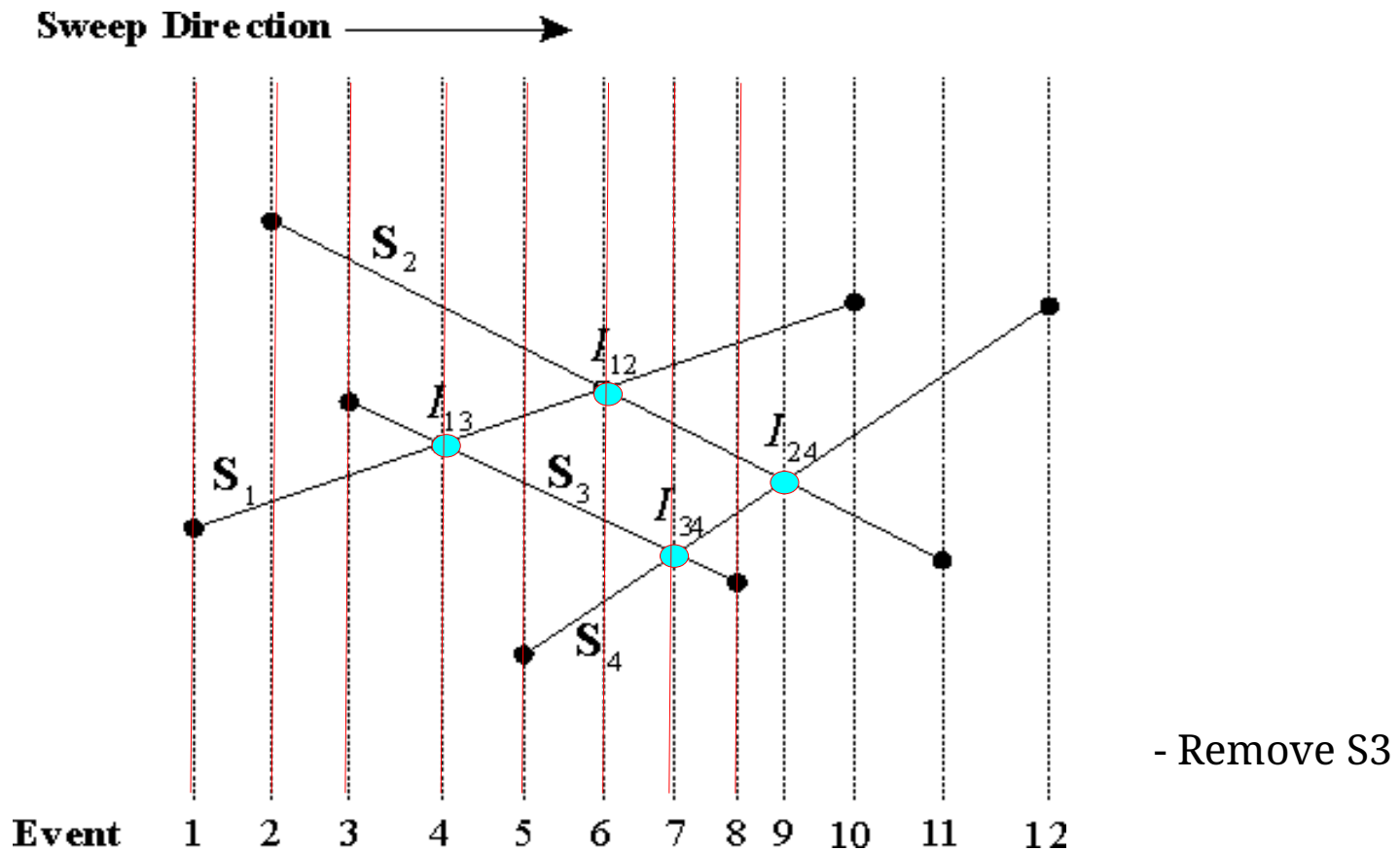


- Swap position of S3 and S4
- Check intersection of S2 and S4
- Add intersection I24

# Computational Geometry



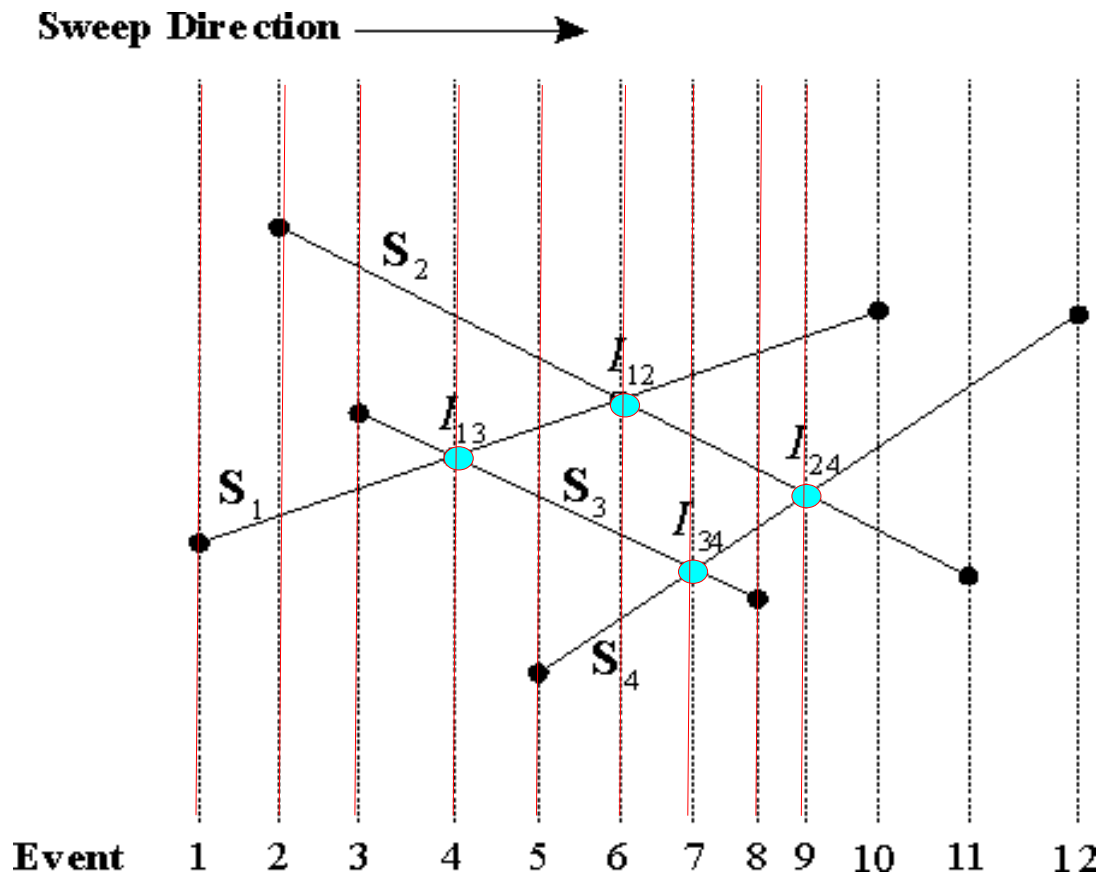
- Line intersection problem



# Computational Geometry



- Line intersection problem

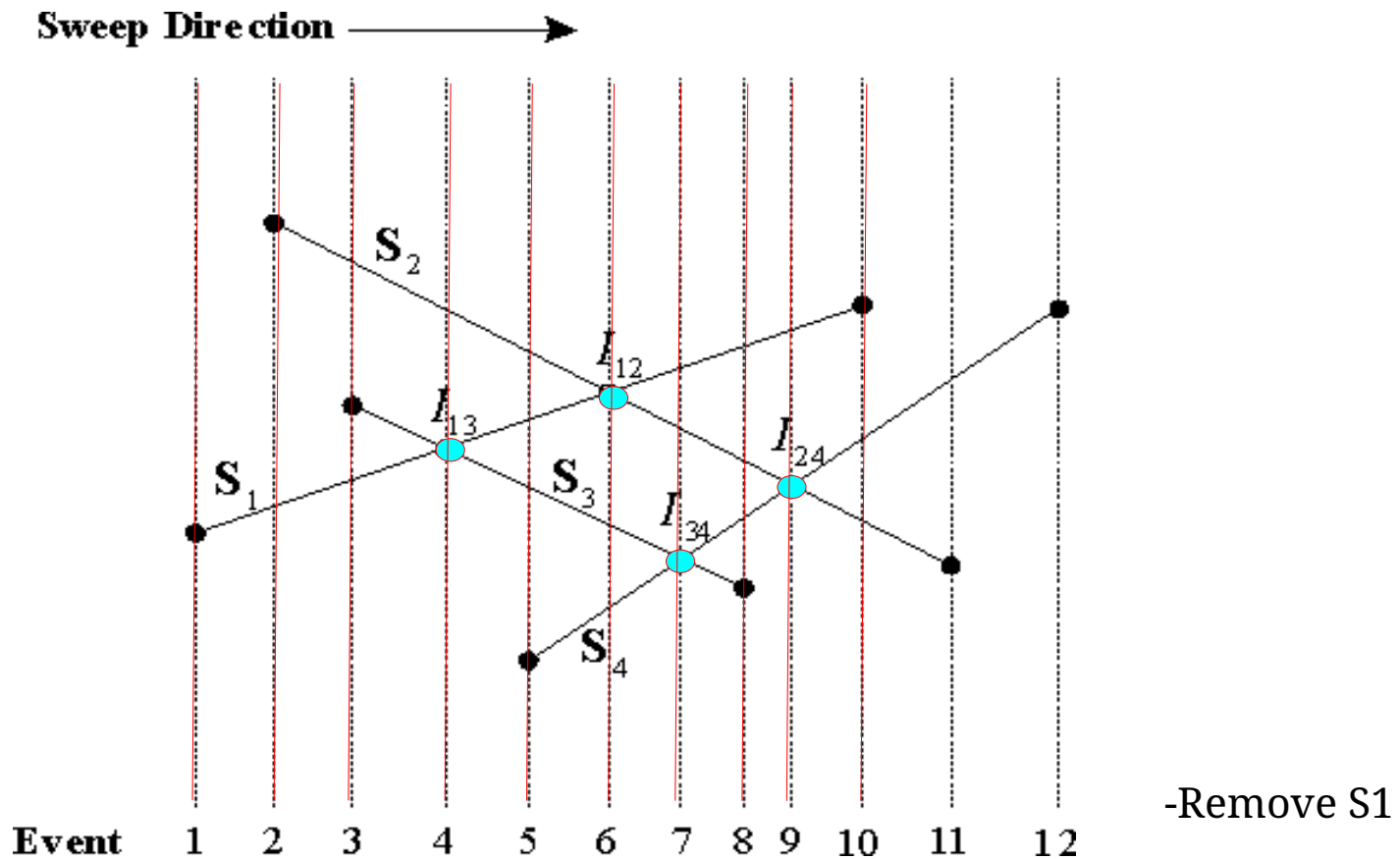


- Swap position of  $S_2$  and  $S_4$
- Check intersection of  $S_1$  and  $S_4$

# Computational Geometry



- Line intersection problem

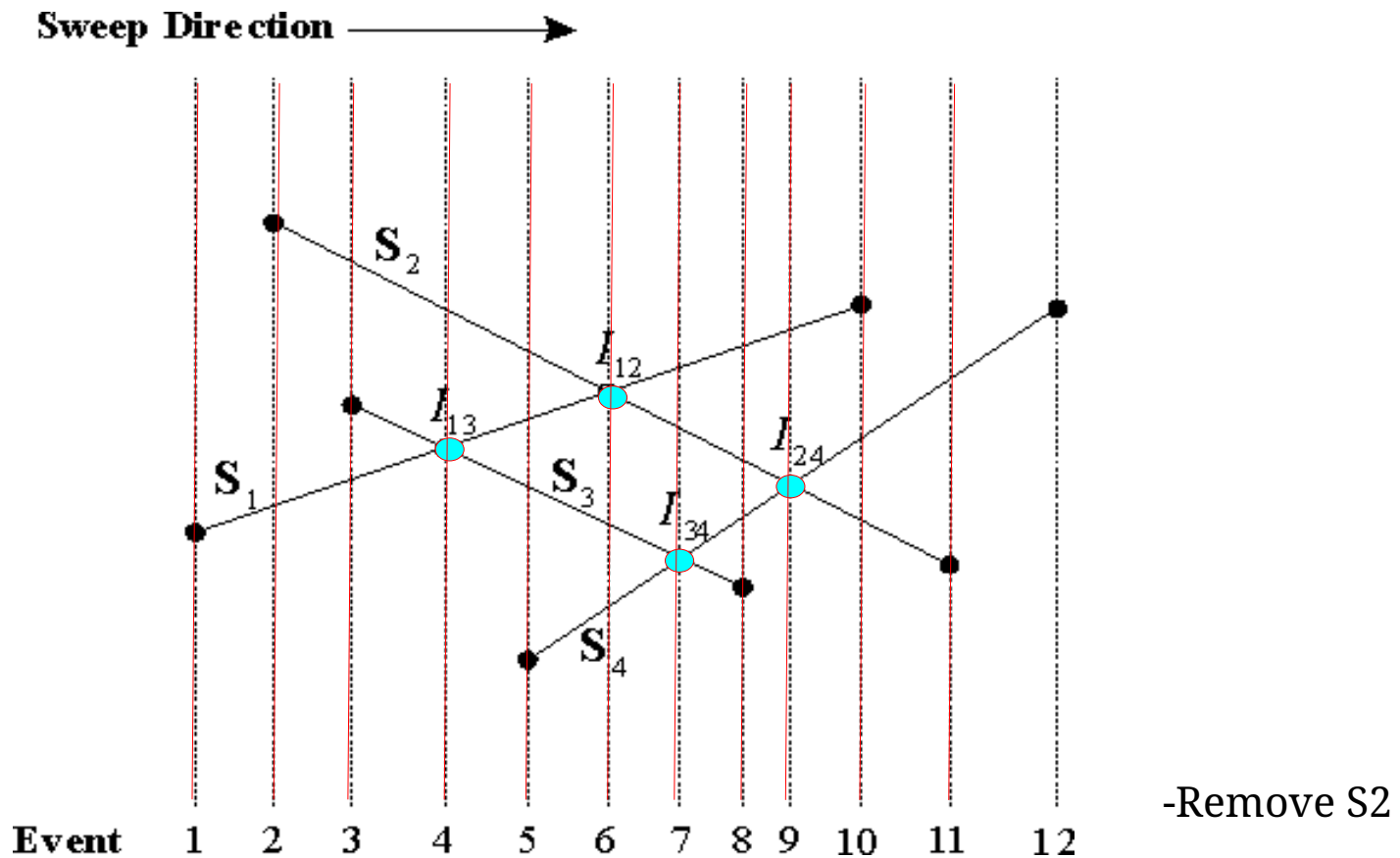




# Computational Geometry



- Line intersection problem



# Computational Geometry



- Line intersection problem

