

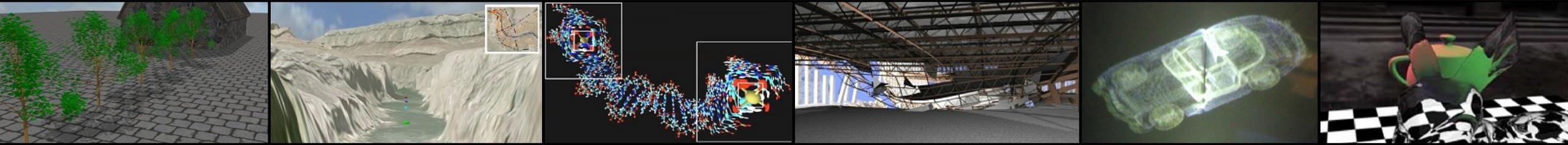
# COT 4521: INTRODUCTION TO COMPUTATIONAL GEOMETRY

---

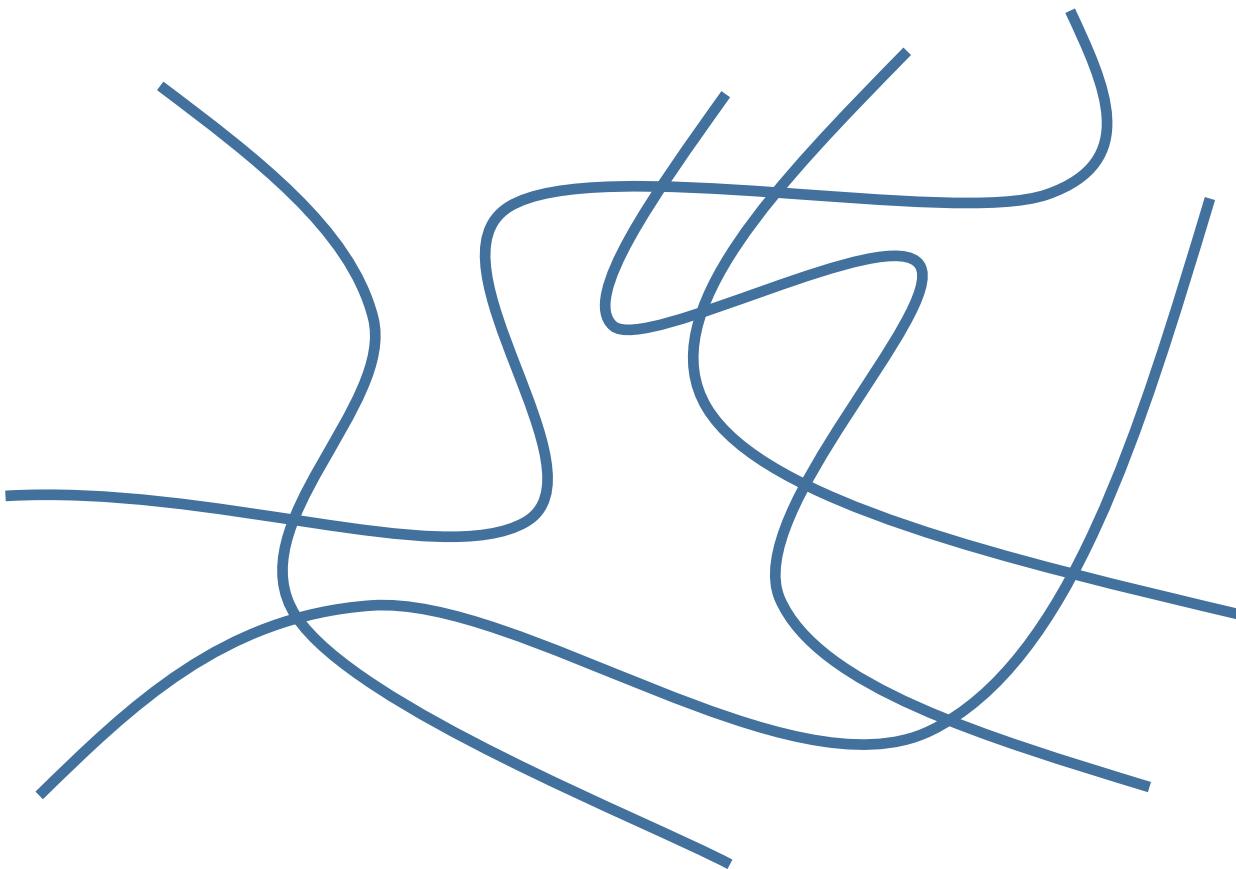


## Segment Intersection

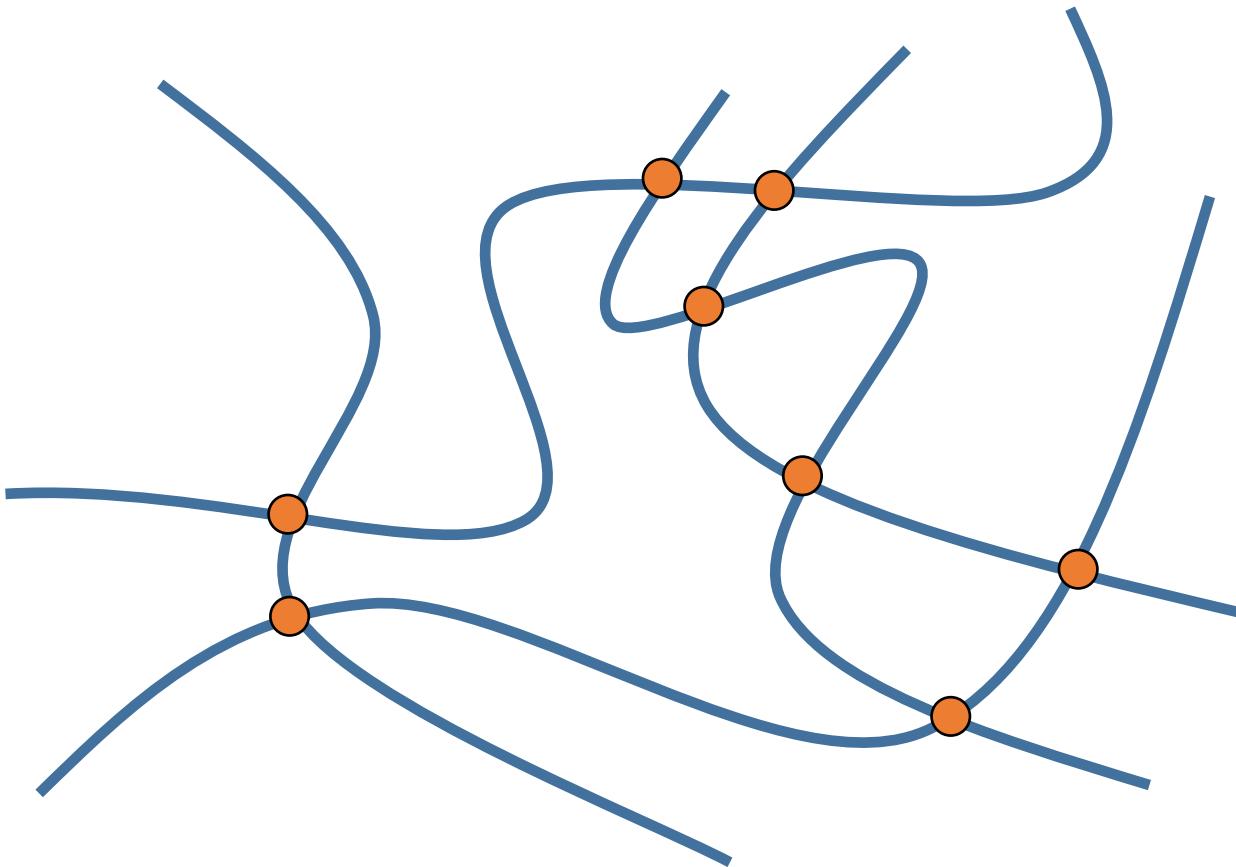
Paul Rosen  
Assistant Professor  
University of South Florida



# PROBLEM STATEMENT

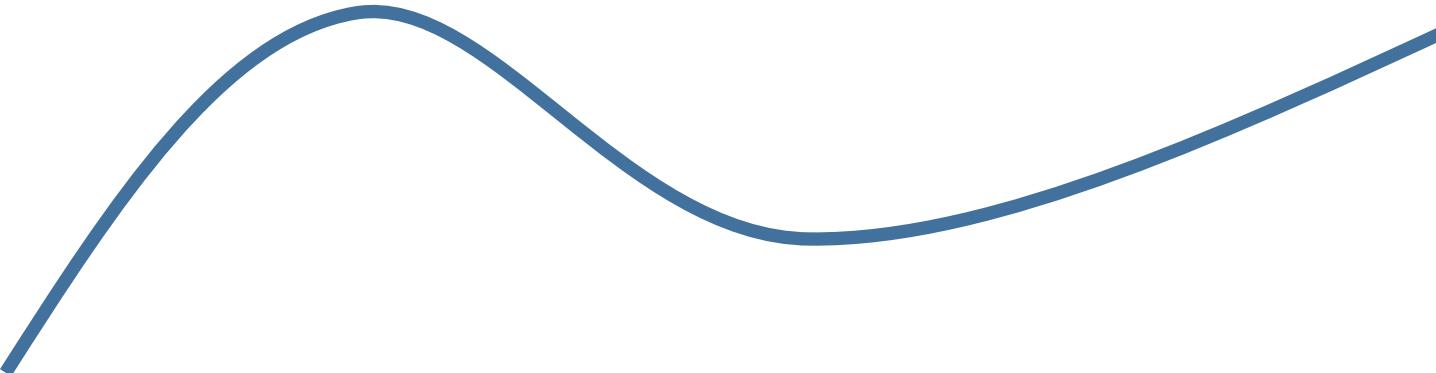


# PROBLEM STATEMENT



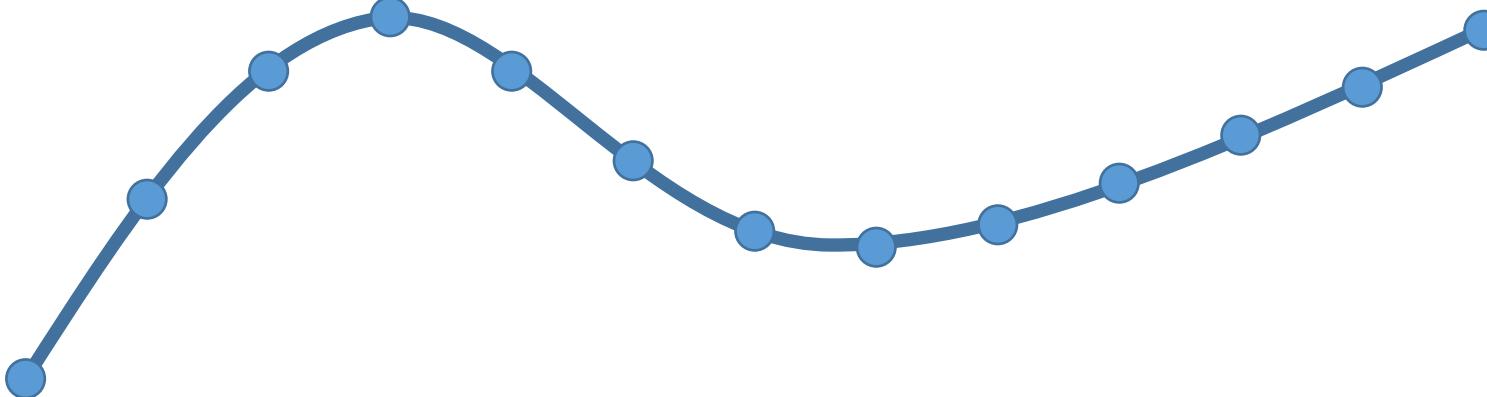
# REPRESENTING CURVES

- CURVES USUALLY REPRESENTED BY A POLYNOMIAL OR POLYNOMIAL SPLINE
  - Bezier, NURBS, etc.
- TESSELATE CURVE INTO MANY SMALL LINE SEGMENTS



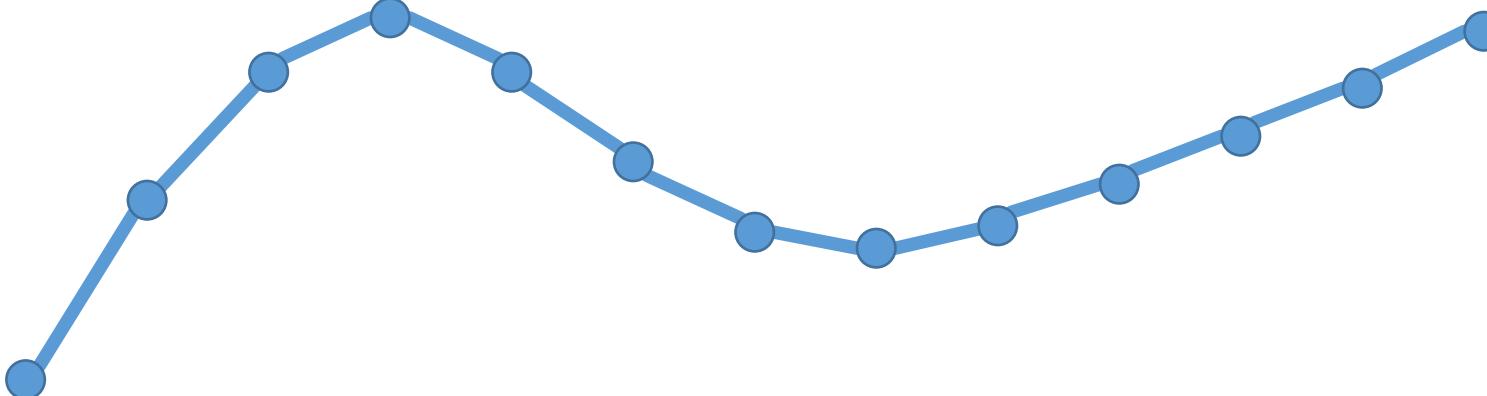
# REPRESENTING CURVES

- CURVES USUALLY REPRESENTED BY A POLYNOMIAL OR POLYNOMIAL SPLINE
  - Bezier, NURBS, etc.
- TESSELATE CURVE INTO MANY SMALL LINE SEGMENTS



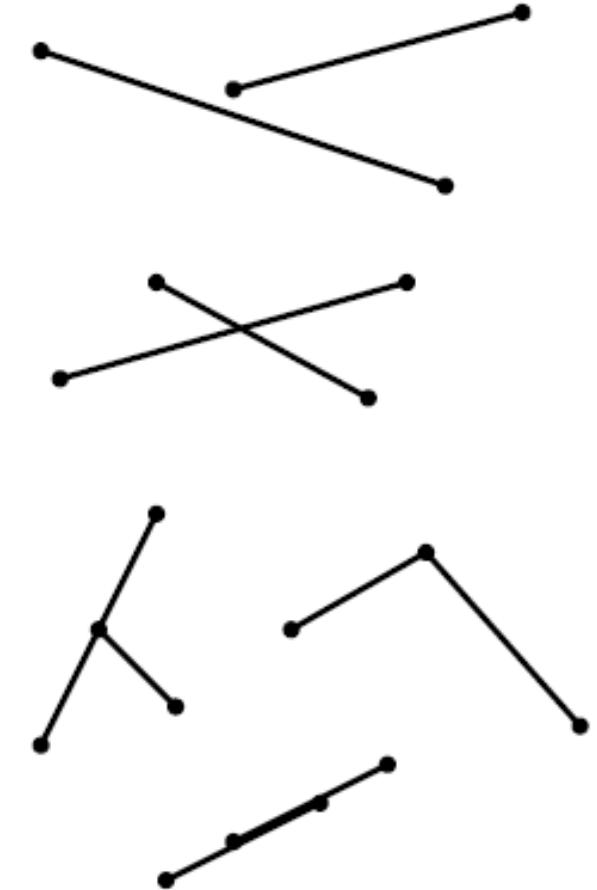
# REPRESENTING CURVES

- CURVES USUALLY REPRESENTED BY A POLYNOMIAL OR POLYNOMIAL SPLINE
  - Bezier, NURBS, etc.
- TESSELATE CURVE INTO MANY SMALL LINE SEGMENTS



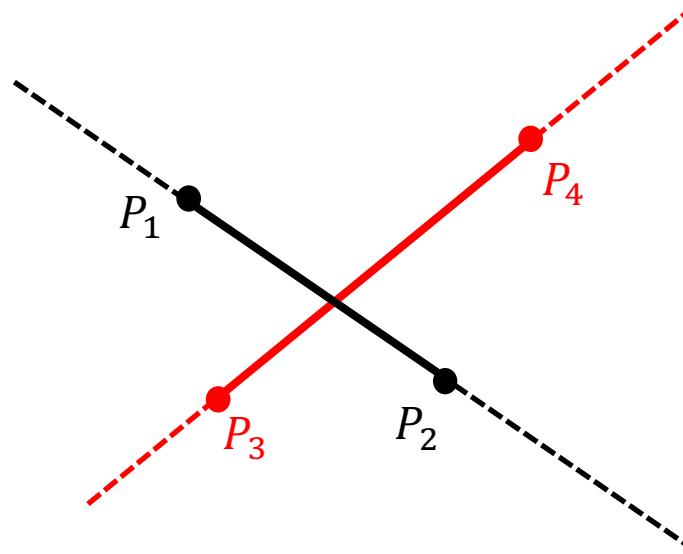
# SEGMENT-SEGMENT INTERSECTION

- A LINE SEGMENT  $\overline{pq}$  IS DENOTED BY ITS TWO ENDPOINTS P AND Q:
  - $\alpha p_x + (1 - \alpha)q_x$
  - $\alpha p_y + (1 - \alpha) q_y$ )
  - where  $0 \leq \alpha \leq 1$
- 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



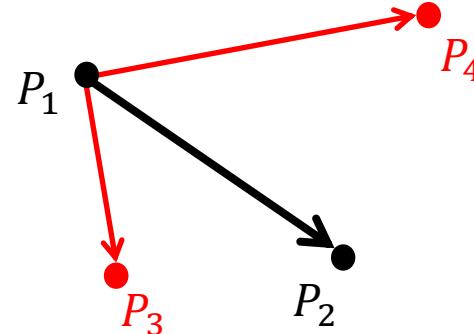
# DO THEY INTERSECT?

- OBSERVATION: IF THE TWO SEGMENTS INTERSECT, THE TWO RED POINTS MUST LIE ON DIFFERENT SIDES OF THE BLACK LINE (OR LIE EXACTLY ON IT)
- THE SAME HOLDS WITH BLACK/RED SWITCHED



# DO THEY INTERSECT?

- WHAT DOES “DIFFERENT SIDES” MEAN?
- USE THE CROSS PRODUCT TO DETERMINE SIDEDNESS



# REPRESENTING A LINE

- SLOPE-INTERCEPT FORM:

$$y = mx + b$$

- GIVEN 2 POINTS,  $P_1$  AND  $P_2$ , HOW DO YOU COMPUTE  $m$  AND  $b$ ?
- GIVEN 2 LINES,  $m_1, b_1$  AND  $m_2, b_2$ , HOW DO YOU COMPUTE THE INTERSECTION POINT,  $P_I$ , BETWEEN THEM?
- How DO YOU KNOW IF  $P_I$  IS ON THE SEGMENT DEFINED BY  $P_1$  AND  $P_2$ ?



## REPRESENTING A LINE

- STANDARD FORM:

$$Ax + By + C = 0$$

- GIVEN 2 POINTS,  $P_1$  AND  $P_2$ , HOW DO YOU COMPUTE  $A$ ,  $B$  AND  $C$ ?
- GIVEN 2 LINES,  $A_1, B_1, C_1$  AND  $A_2, B_2, C_2$ , HOW DO YOU COMPUTE THE INTERSECTION POINT,  $P_I$ , BETWEEN THEM?
- How DO YOU KNOW IF  $P_I$  IS ON THE SEGMENT DEFINED BY  $P_1$  AND  $P_2$ ?



# REPRESENTING A LINE

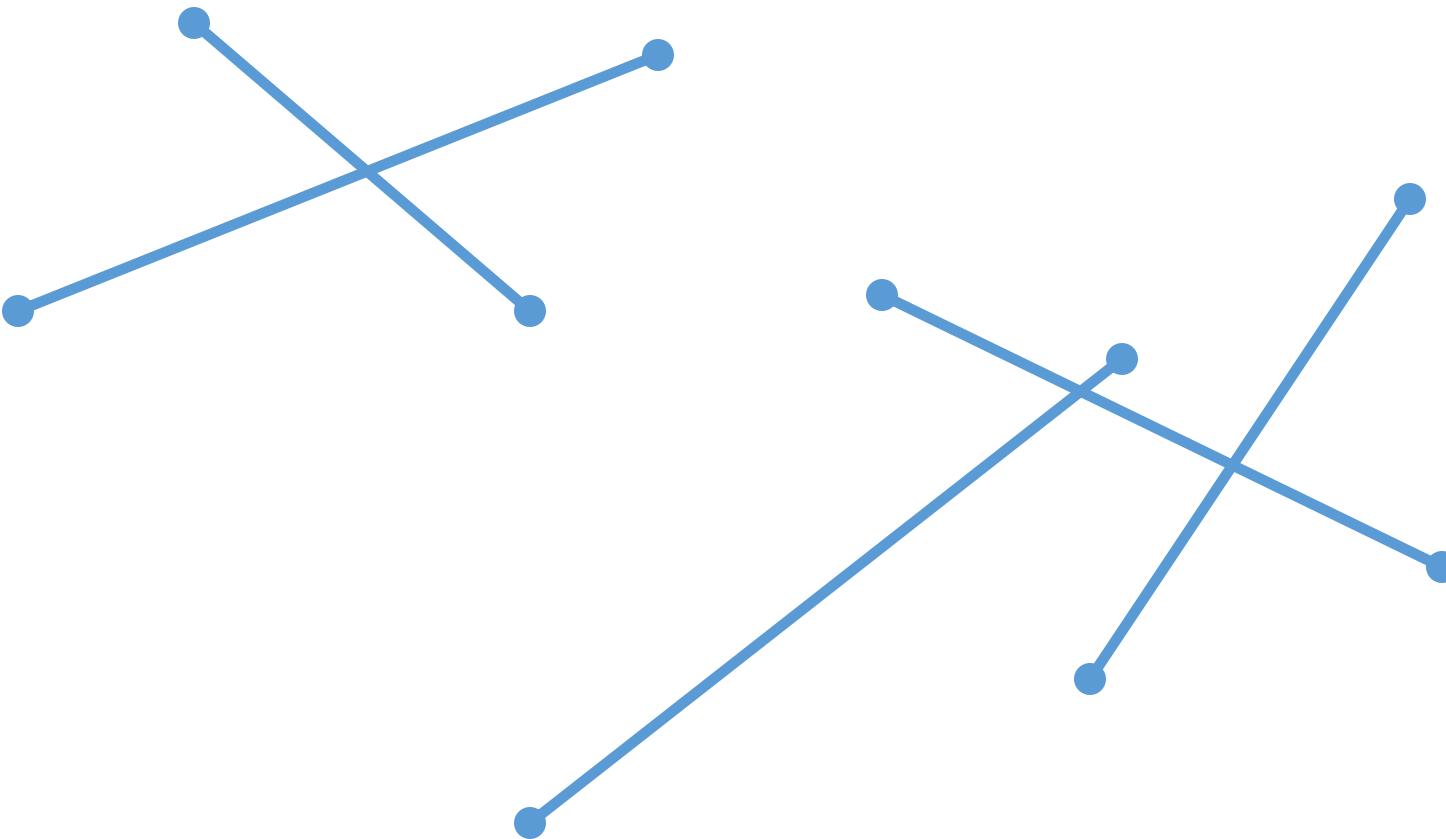
- PARAMETRIC FORM:

$$P = P_0 + Dt$$

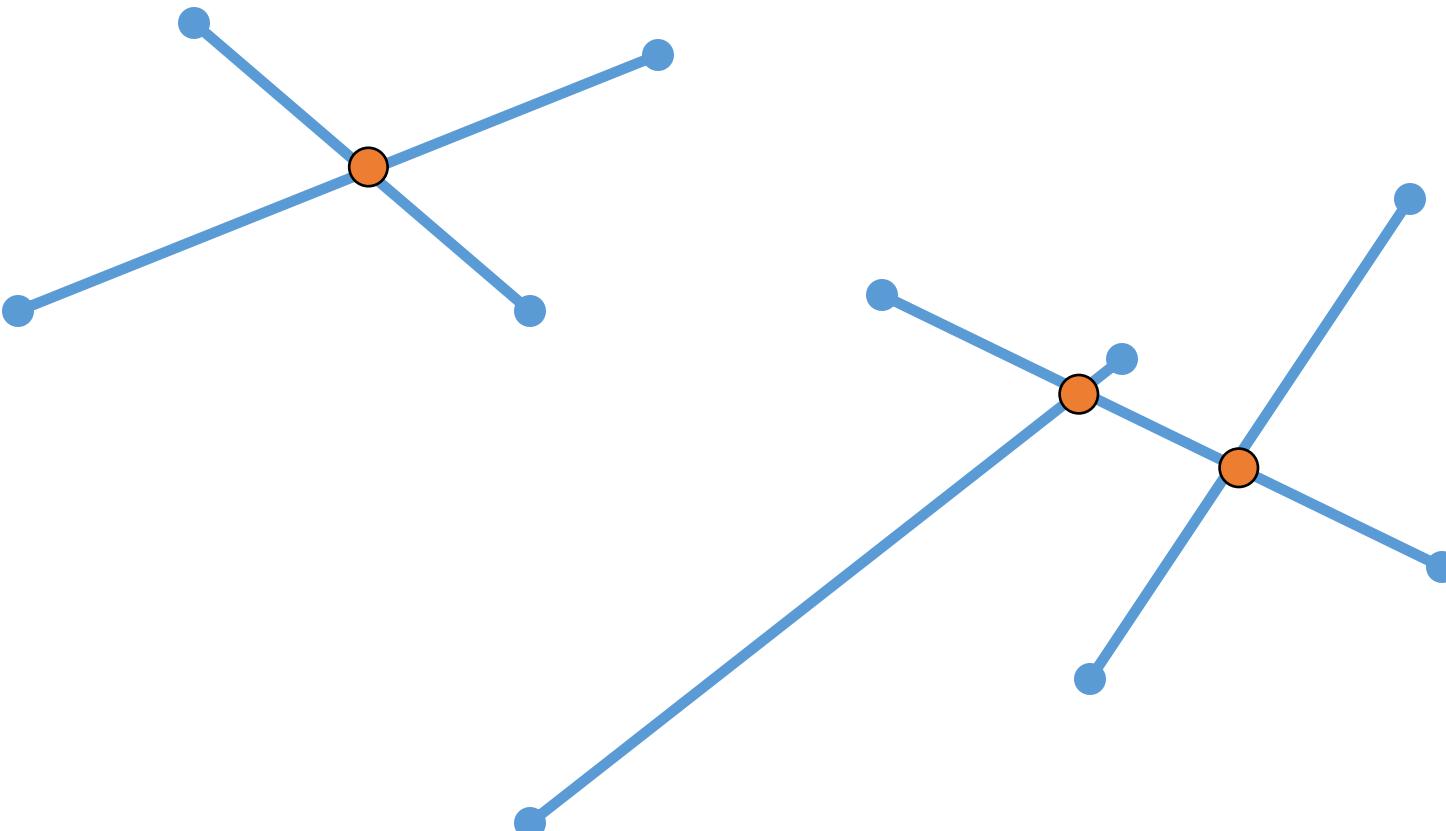
- GIVEN 2 POINTS,  $P_1$  AND  $P_2$ , HOW DO YOU REPRESENT THE PARAMETRIC LINE?
- GIVEN 2 LINES, HOW DO YOU COMPUTE THE INTERSECTION POINT,  $P_I$ , BETWEEN THEM?
- How DO YOU KNOW IF  $P_I$  IS ON THE SEGMENT DEFINED BY  $P_1$  AND  $P_2$ ?



# INTERSECTION OF >2 LINE SEGMENTS



# INTERSECTION OF >2 LINE SEGMENTS



# INTERSECTION OF LINE SEGMENTS

- PROBLEM DEFINITION
  - Given  $N$  line segments in the plane, report all their points of intersection (pairwise).
- LINE SEGMENT INTERSECTION (LSI).
  - Instance:
    - Set  $S = \{s_1, s_2, \dots, s_N\}$  of line segments in the plane;
    - For  $1 \leq i \leq N$ ,  $s_i = (P_{i1}, P_{i2})$  (endpoints of the segments); and
    - For  $1 \leq j \leq 2$ ,  $P_{ij} = (x_{ij}, y_{ij})$  (coordinates of the endpoints).
  - Question:
    - Report all points of intersection of segments in  $S$ .



# INTERSECTION OF LINE SEGMENTS

- ALGORITHM (BRUTE FORCE ALGORITHM)
  - For every pair of segments in  $S$ , test the two segments for intersection.
  - (Segment intersection test can be done in constant time using one of the methods we've already discussed.)
- ANALYSIS (PREPROCESSING, QUERY, AND STORAGE COSTS)
  - Preprocessing: None
  - Query:  $O(N^2)$ ; there are  $\frac{N(N - 1)}{2} = O(N^2)$  pairs, each requiring a constant time test.
  - Storage:  $O(N)$ ; for  $S$ .

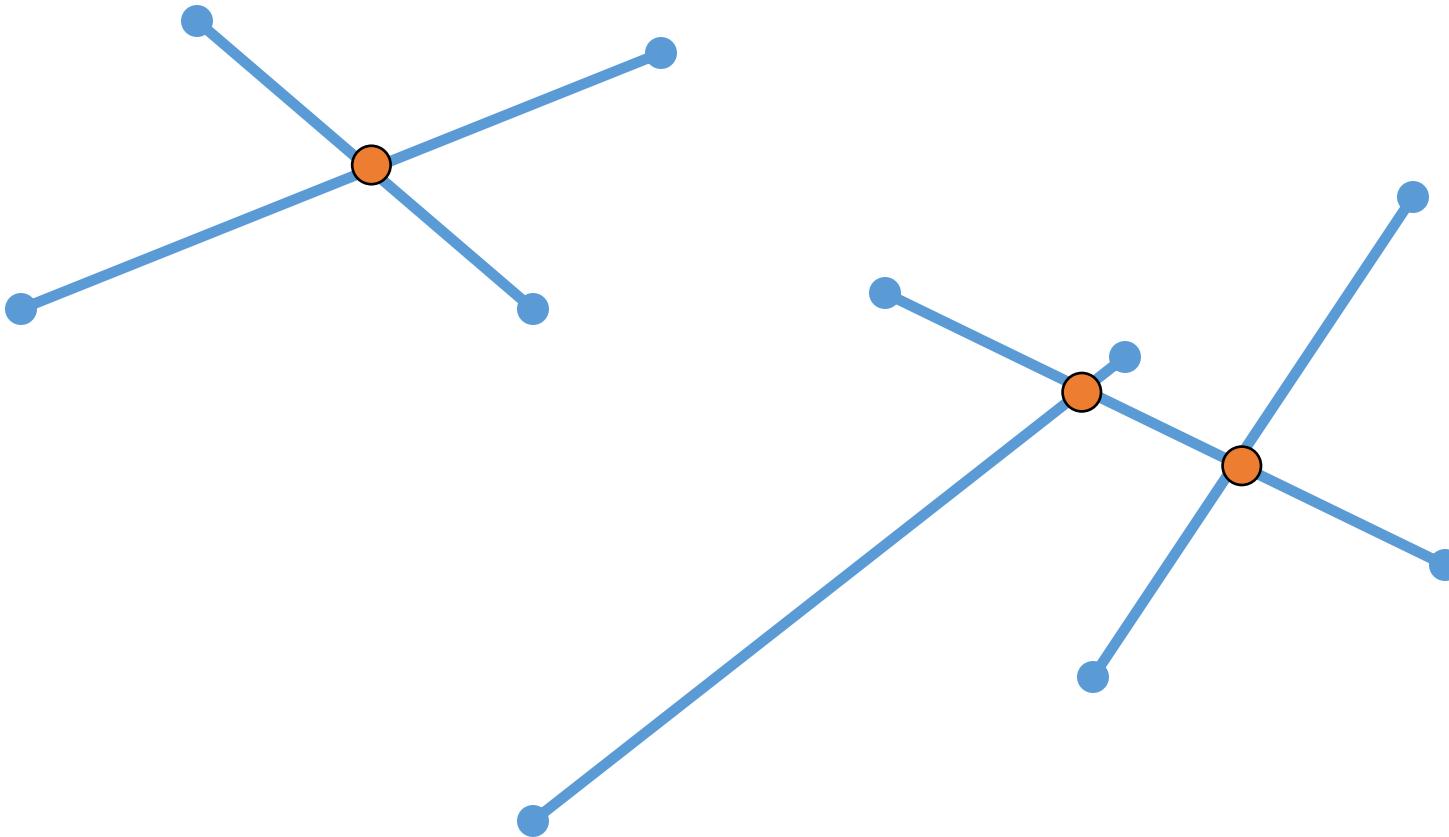


# NAÏVE INTERSECTION OF $>2$ LINE SEGMENTS

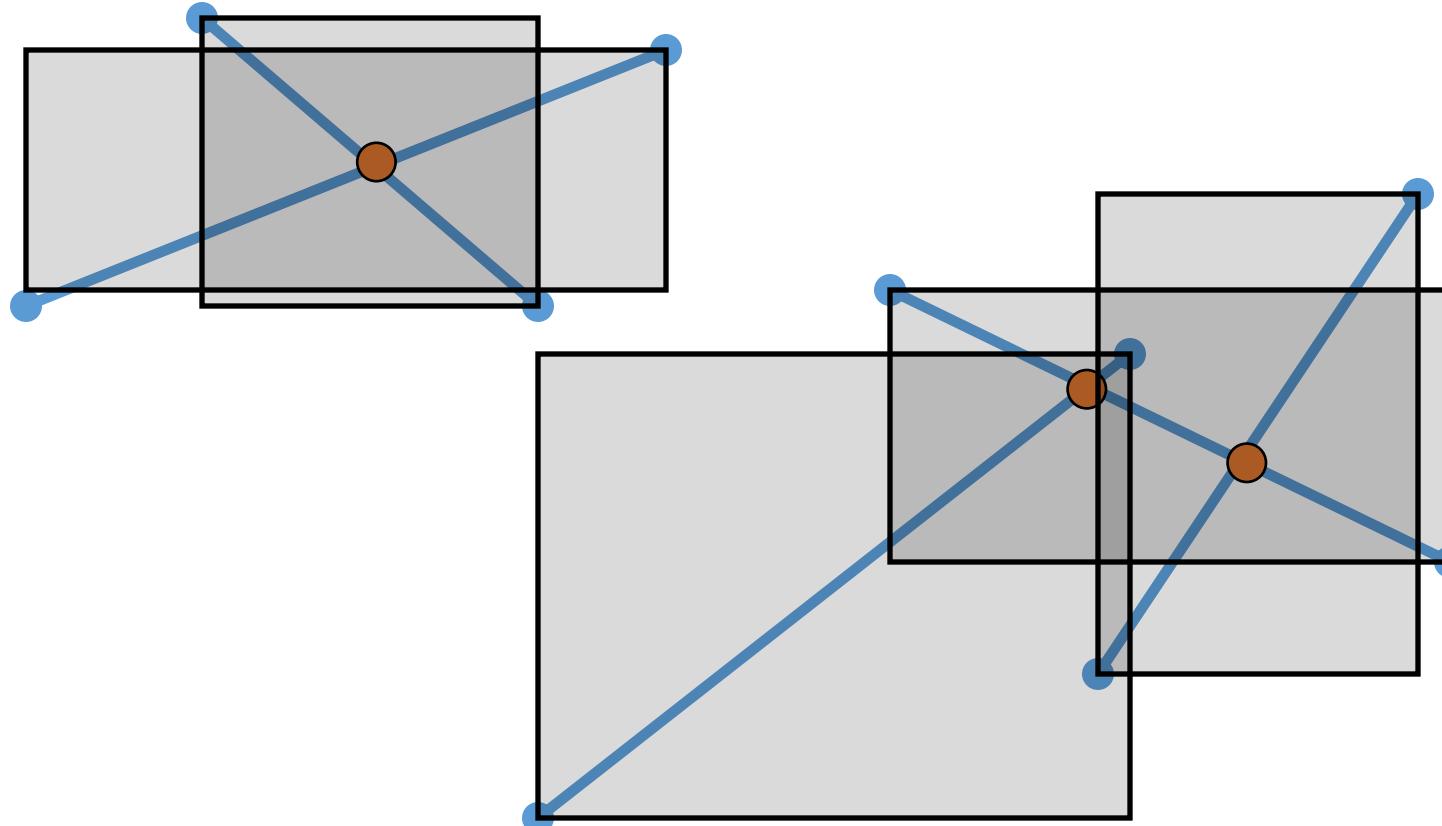
- **WHAT IS THE WORST CASE NUMBER OF INTERSECTIONS?**
  - If all pairs intersect there are  $(n^2)$  intersections, then our time bound is optimal as a function of  $n$ .
- **CAN WE IMPROVE PERFORMANCE?**
  - Yes, we will look for output-sensitive algorithms.



# OBSERVATIONS?

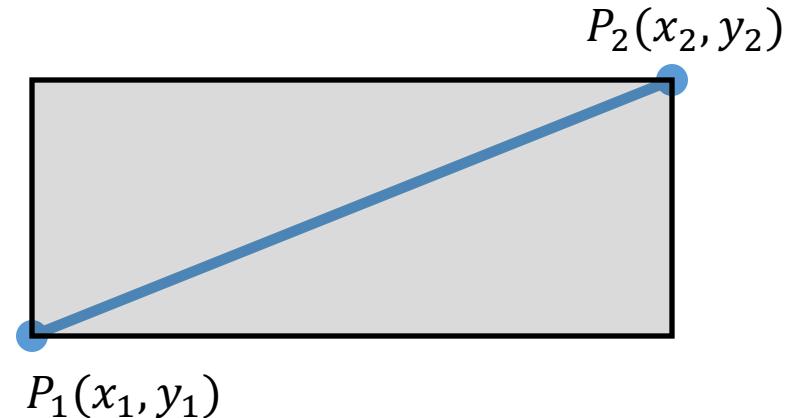


# AXIS ALIGNED BOUNDING BOXES (AABB)



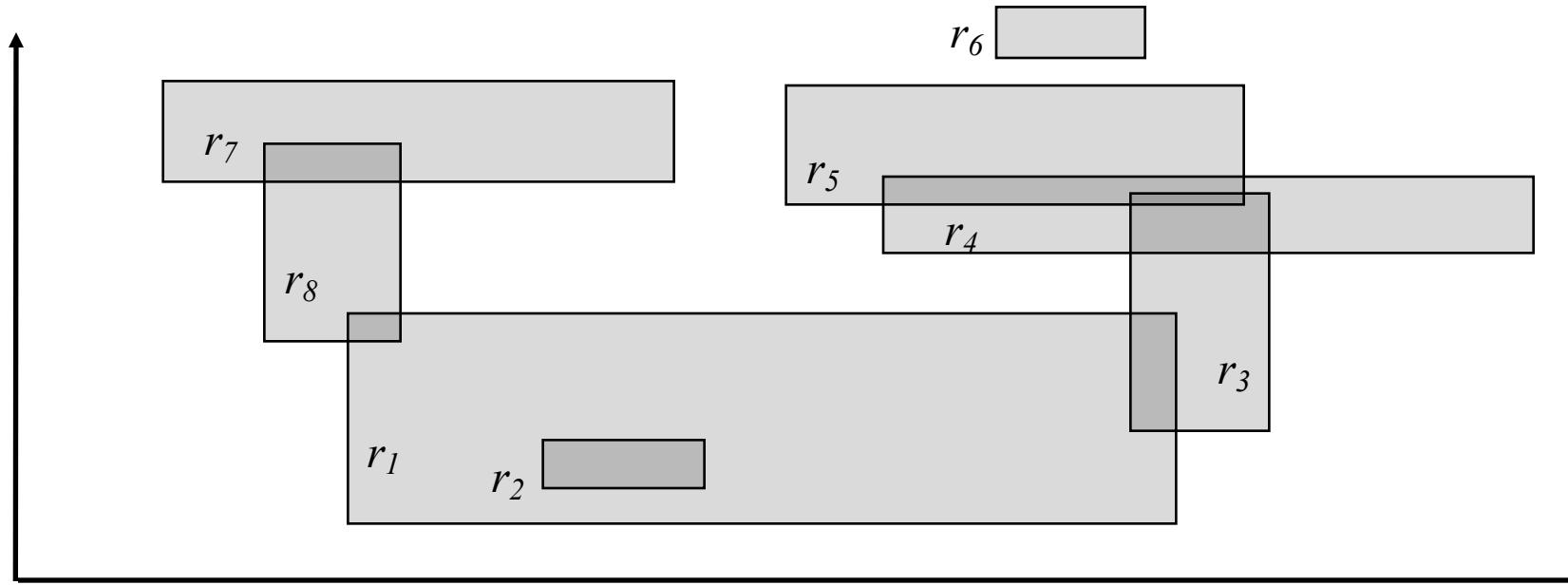
# AXIS ALIGNED BOUNDING BOXES (AABB)

- MANY REPRESENTATIONS
  - 2 Points
  - Point, width, height
  - Intervals
- FOR OUR CONTEXT, WE WILL USE INTERVALS
  - $x \in [\min(x_1, x_2), \max(x_1, x_2)]$
  - $y \in [\min(y_1, y_2), \max(y_1, y_2)]$



# AABB INTERSECTION

- GIVEN A SET OF  $N$  AXIS-PARALLEL RECTANGLES IN THE PLANE, REPORT ALL INTERSECTING PAIRS
  - Intersect  $\equiv$  share at least one point

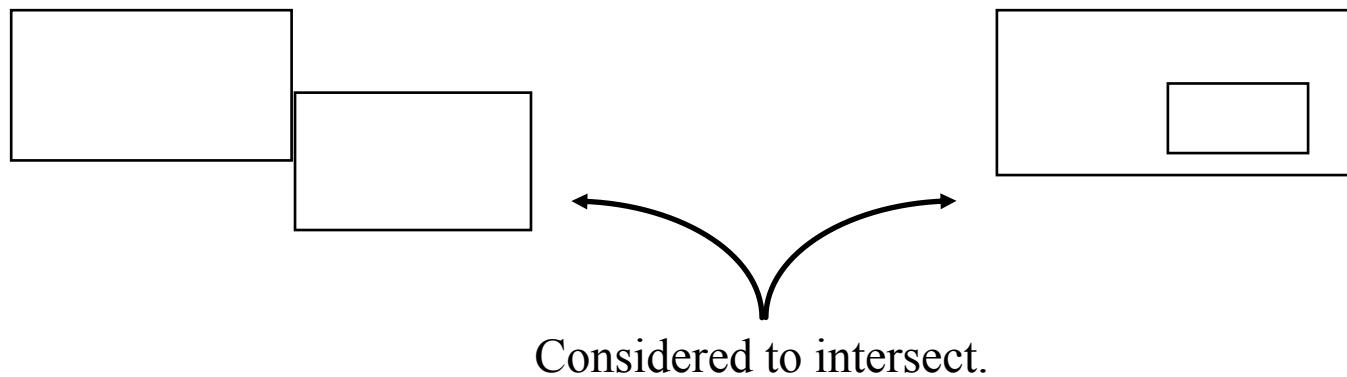
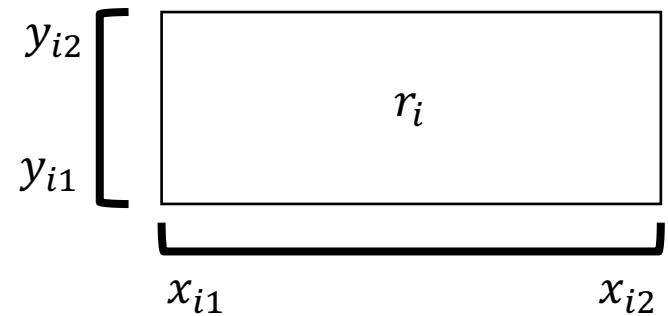


Answer:  $(r_1, r_2)$   $(r_1, r_3)$   $(r_1, r_8)$   $(r_3, r_4)$   $(r_3, r_5)$   $(r_4, r_5)$   $(r_7, r_8)$



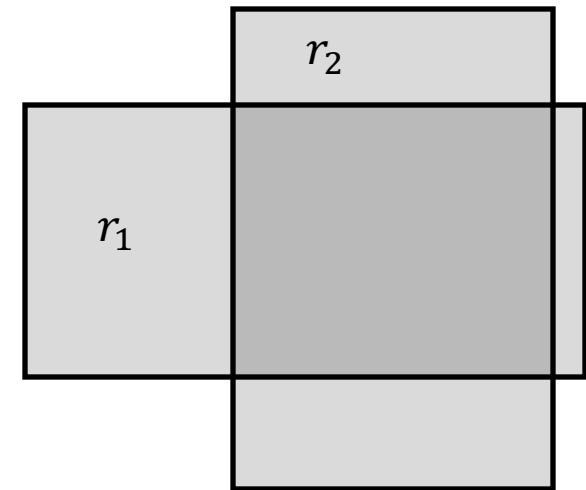
# INTERSECTION OF RECTANGLES PROBLEM DEFINITION

- **RECTANGLE INTERSECTION**
- **INSTANCE:** Set  $S = \{r_1, r_2, \dots, r_N\}$  of rectangles in the plane.
- For  $1 \leq i \leq N$ ,  $r_i = ([x_{i1}, x_{i2}], [y_{i1}, y_{i2}])$
- **QUESTION:** Report all pairs of rectangles that intersect
  - (Edge and interior intersections should be reported.)



# CHECKING IF 2 RECTANGLES INTERSECT

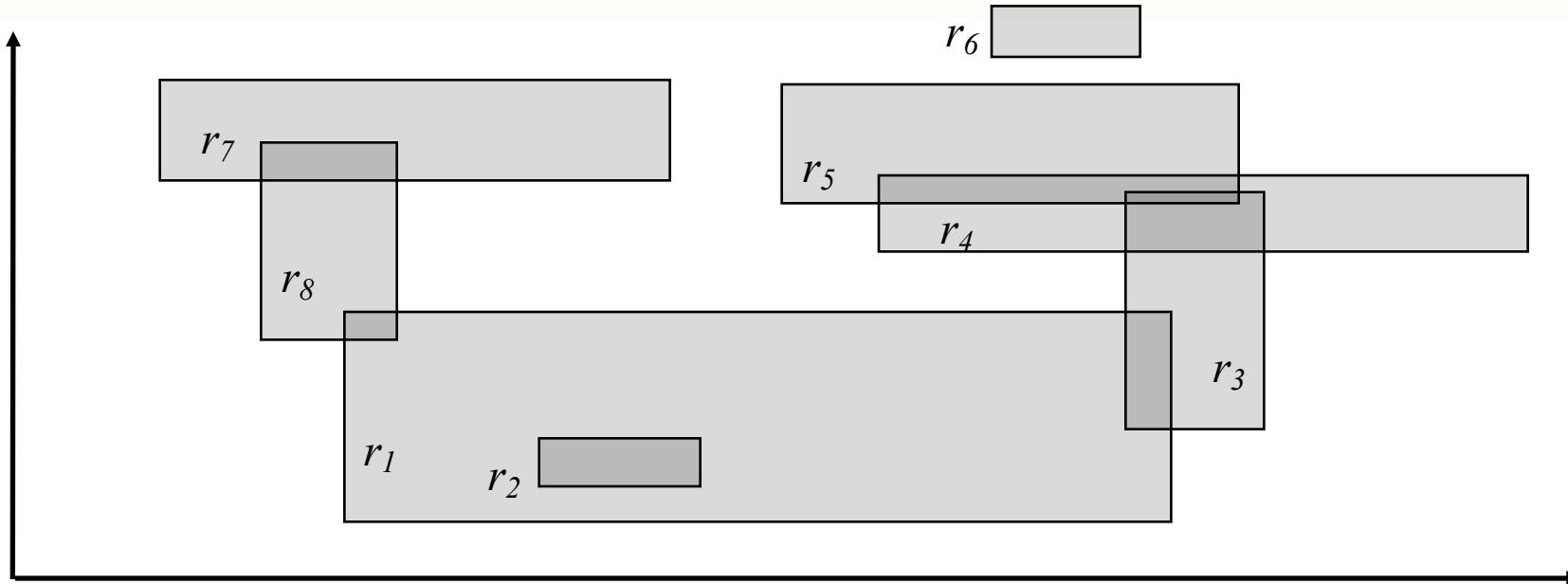
- $r_1 \cap r_2$ , IF?
  - $[x_{11}, x_{12}] \cap [x_{21}, x_{22}] \neq \emptyset$  AND  
 $[y_{11}, y_{12}] \cap [y_{21}, y_{22}] \neq \emptyset$
- HOW DO WE CODE THIS INTERSECTION?
  - $[x_{11}, x_{12}] \cap [x_{21}, x_{22}] =$   
 $[\max(x_{11}, x_{21}), \min(x_{12}, x_{22})]$
  - $[y_{11}, y_{12}] \cap [y_{21}, y_{22}] =$   
 $[\max(y_{11}, y_{21}), \min(y_{12}, y_{22})]$
  - Check that the range of both intersections is  $\geq 0$



# INTERSECTION OF A SET OF RECTANGLES

Brute force algorithm

1. for every pair  $(r_i, r_j)$  of rectangles  $\in S, i < j$
2. if  $(r_i \cap r_j \neq \emptyset)$  then
3. report  $(r_i, r_j)$



Answer:  $(r_1, r_2) (r_1, r_3) (r_1, r_8) (r_3, r_4) (r_3, r_5) (r_4, r_5) (r_7, r_8)$



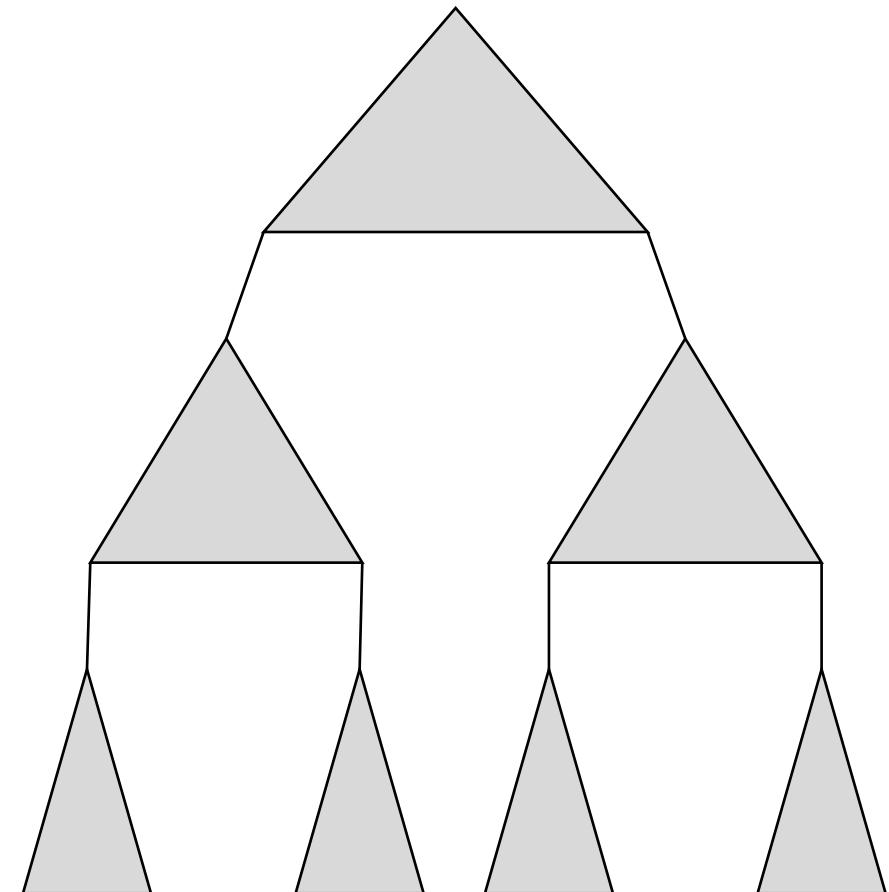
# INTERSECTION OF RECTANGLES

- ANALYSIS
  - Preprocessing?
    - None
  - Query?
    - $\frac{N(N - 1)}{2} = O(N^2)$
  - Storage?
    - $O(N)$
- DOES THIS REALLY HELP US WITH THE SEGMENT INTERSECTION PROBLEM?

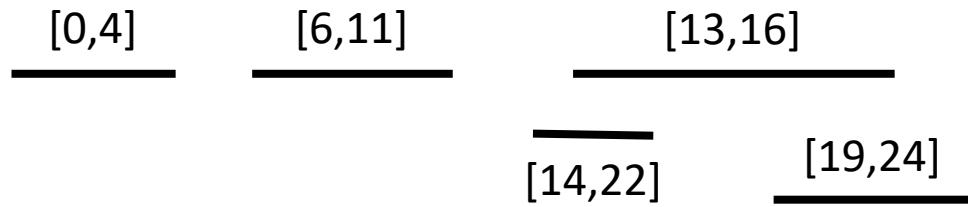


# ID CENTERED INTERVAL TREES

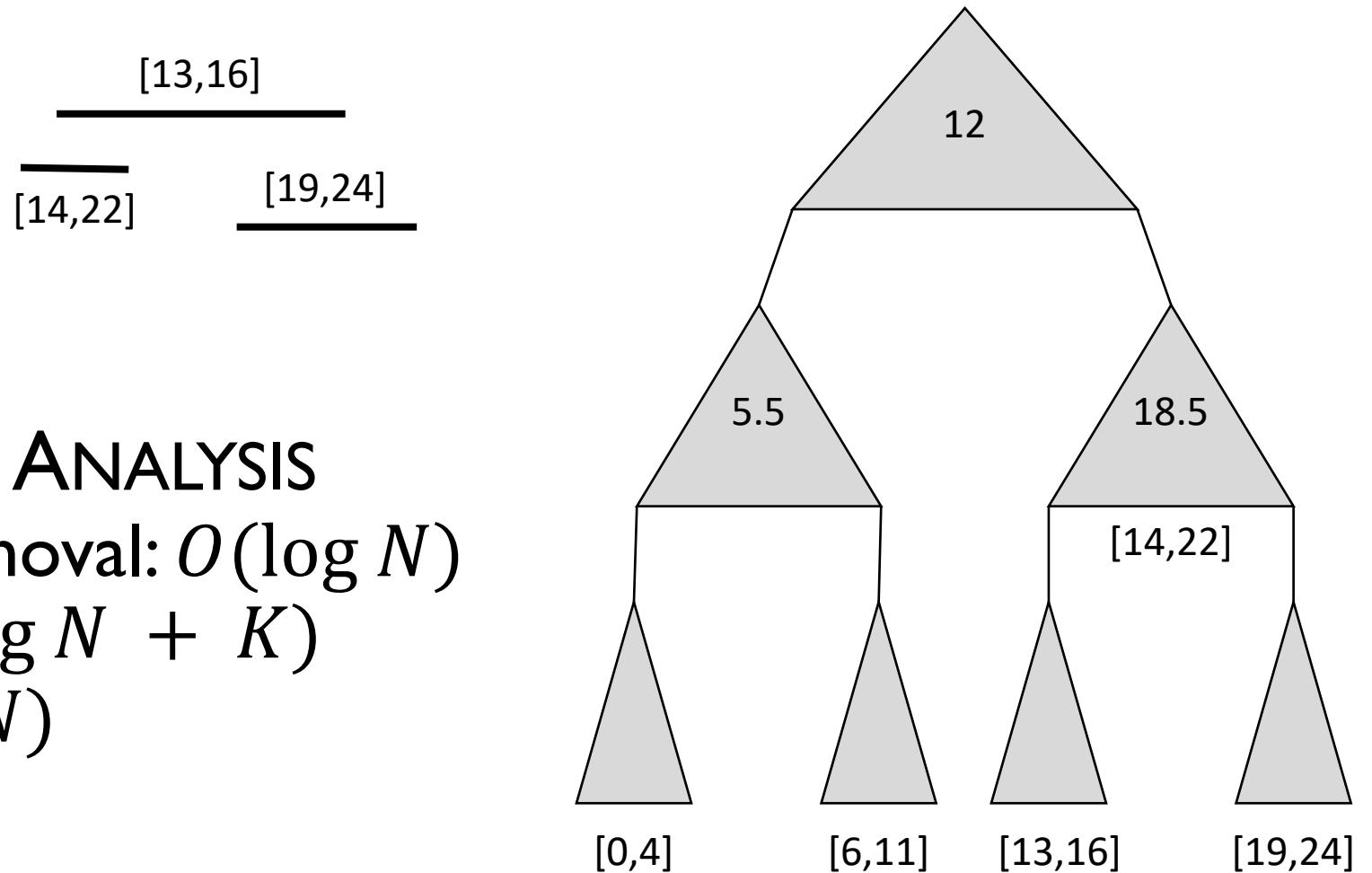
- TREE-BASED DATA STRUCTURE
- EACH NODE STORES A CENTER POINT
  - Intervals are placed into 3 group,
    - Left of center—placed in left subtree
    - Right of center—placed in right subtree
    - Covering center—placed in a specialized list\*



# ID CENTERED INTERVAL TREES



- PERFORMANCE ANALYSIS
  - Insertion/removal:  $O(\log N)$
  - Query:  $O(\log N + K)$
  - Storage:  $O(N)$

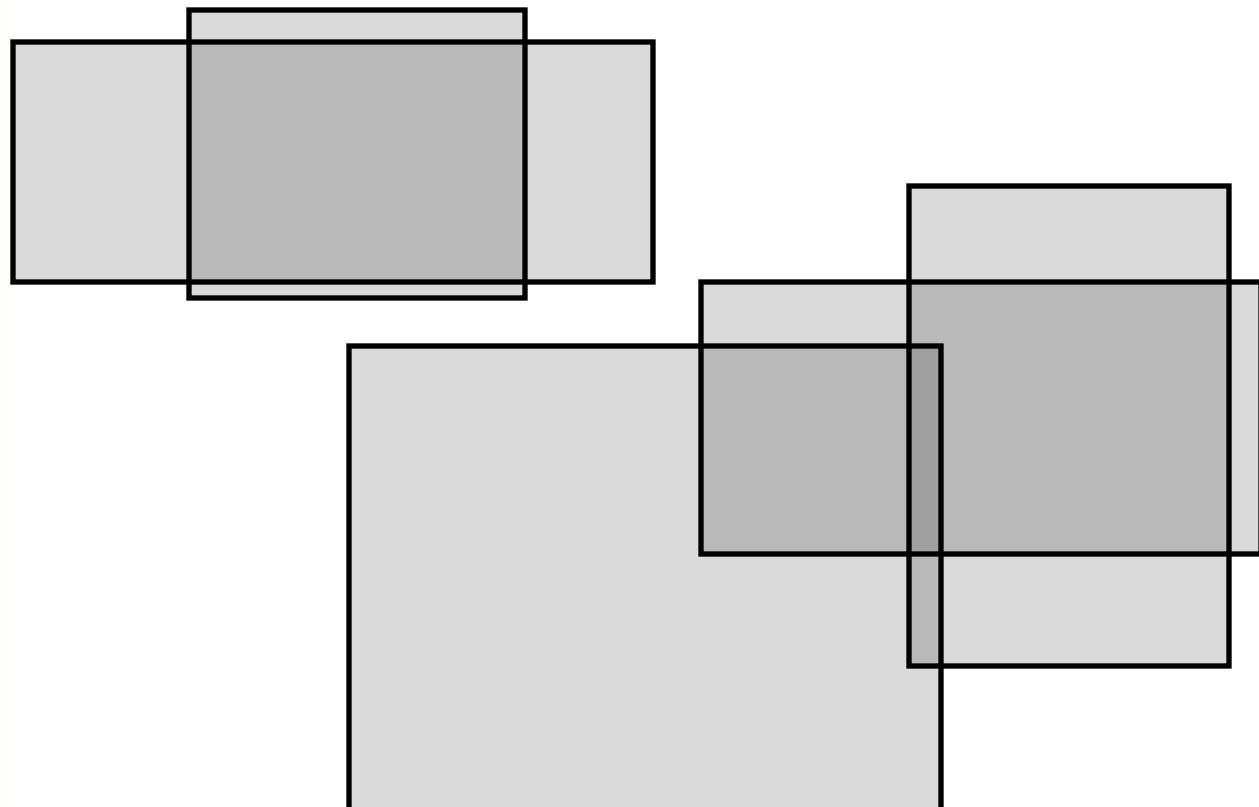


# INTERSECTION OF RECTANGLES

Algorithm using interval trees

1. Plane sweep algorithm, vertical (top-to-bottom) sweep Event points are Beginning and end of rectangle intervals
2. At the starting interval:
3. Compare rectangle x-interval to active set for overlap.
4. Add rectangle x-interval to active set.
5. At the ending interval:
6. Remove rectangle x-interval from active set.

Active set maintained in interval tree.



# INTERSECTION OF RECTANGLES

- ANALYSIS
  - Preprocessing:  $O(N \log N)$ ; ordering intervals for sweep
  - Query:  $O(N \log N + K)$
  - Storage:  $O(N)$ ; interval tree  $O(N)$ , event queue  $O(N)$ .
- COMMENTS
  - $O(N \log N + K)$  is lower bound for rectangle intersection problem. Can be shown by lower bounds proof.
  - We've gone to a lot of trouble to improve the time from  $O(N^2)$  to  $O(N \log N)$  via the interval tree, for good reason.
    - E.g. if  $N = 10^6$ ,  $N^2 = 10^{12}$  and  $N \log N = 2 \cdot 10^7$

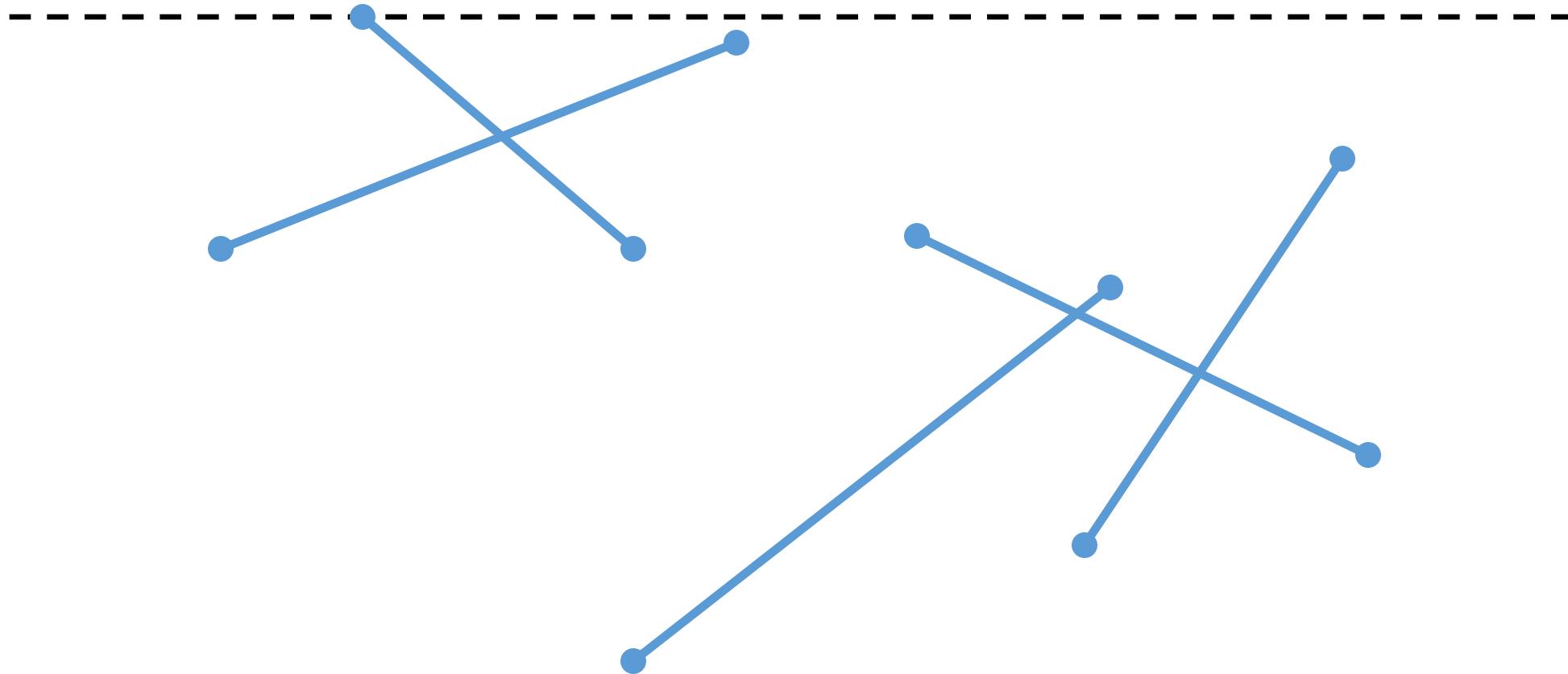


## BACK TO THE SEGMENT INTERSECTION PROBLEM

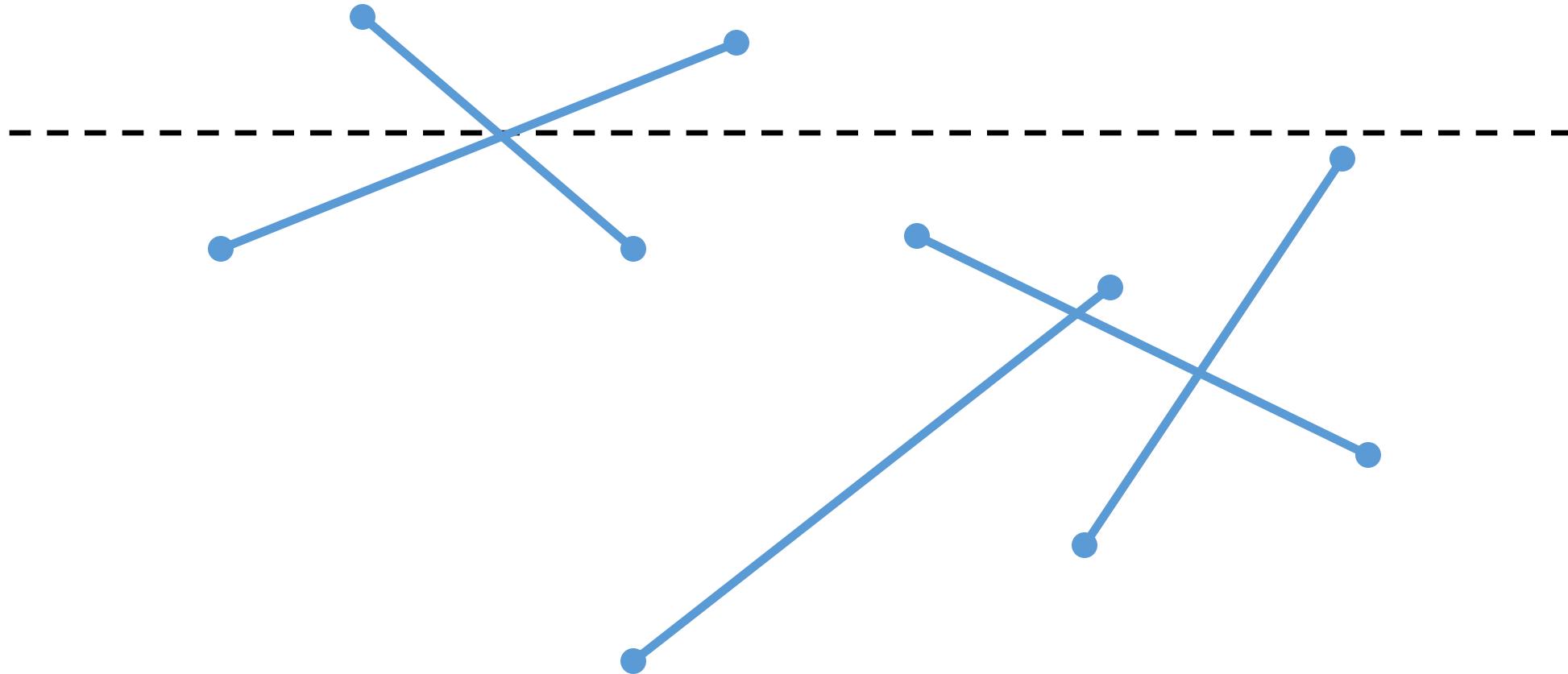
- CAN WE DO BETTER THAN REDUCING IT TO THE AABB INTERSECTION PROBLEM?
- YES AND NO, CAN'T DO BETTER THAN  $O(N \log N + K)$ , BUT CAN IMPROVE CONSTANTS



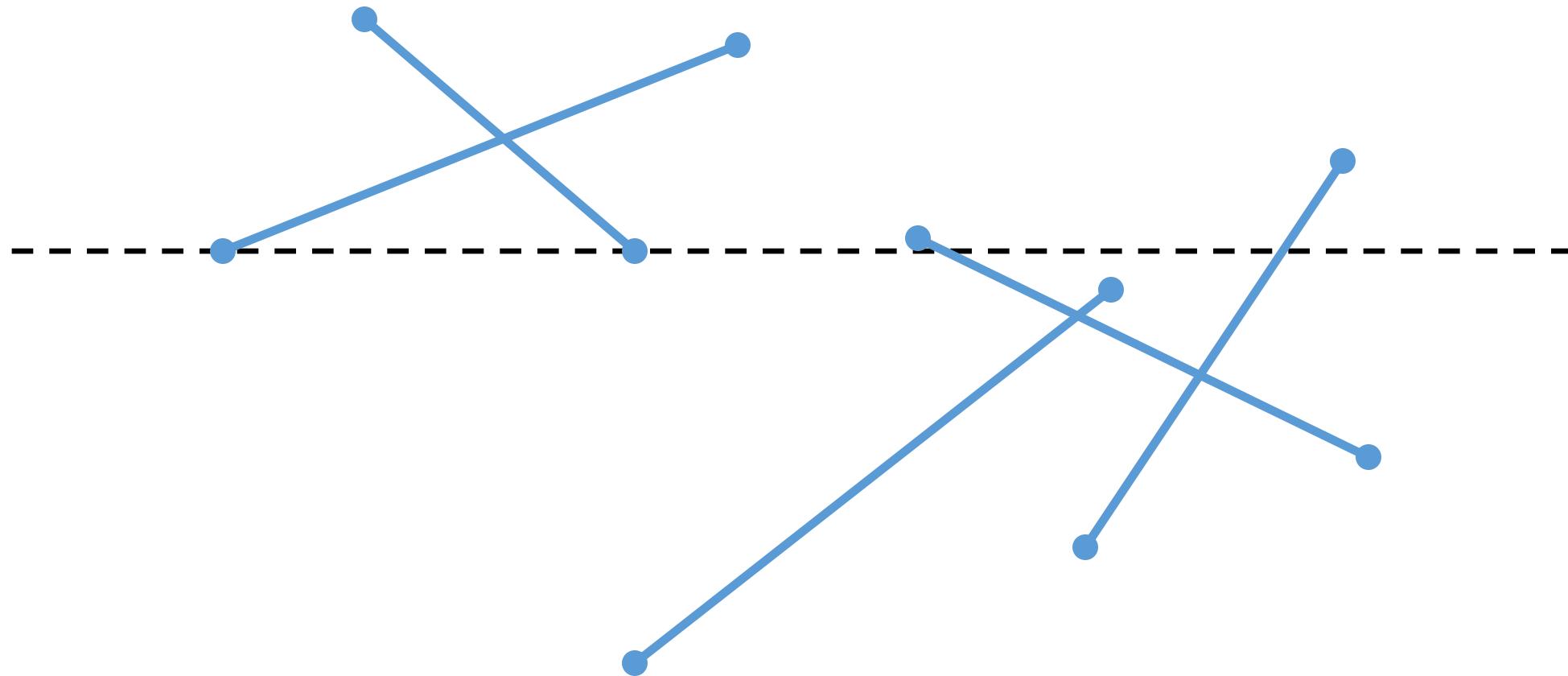
# SWEET ALGORITHM



# SWEET ALGORITHM



# SWEEP ALGORITHM – WHERE DO EVENTS OCCUR?



# EVENTS

- WHEN DO THE EVENTS HAPPEN?
  - When the sweep is:
    - At an upper endpoint of the line segment;
    - At a lower endpoint of the line segment; or
    - At an intersection point of line segments
  - At each type, the status changes



# INTERSECTION OF LINE SEGMENTS

- ALGORITHM IDEA (SHAMOS-HOEY ALGORITHM)
- CRUCIAL OBSERVATION:
  - For two segments  $s_1$  and  $s_2$  to intersect, there must be some Y for which  $s_1$  and  $s_2$  are consecutive in the X ordering.
  - This suggests that the sequence of intersections of the segments with the horizontal line contains the information needed to find the intersections between the segments.
- PLANE SWEEP ALGORITHMS OFTEN USE TWO DATA STRUCTURES:
  1. Sweep-line status
  2. Event-point schedule



# INTERSECTION OF LINE SEGMENTS

- SWEEP-LINE STATUS
  - The sweep-line status is a list of the currently comparable segments, ordered by the relation in  $X$ .
- THE SWEEP-LINE STATUS DATA STRUCTURE  $L$  IS USED TO STORE THE ORDERING OF THE CURRENTLY COMPARABLE SEGMENTS. BECAUSE THE SET OF CURRENTLY COMPARABLE SEGMENTS CHANGES, THE DATA STRUCTURE FOR  $L$  MUST SUPPORT THESE OPERATIONS:
  1.  $\text{INSERT}(s, L)$ . Insert segment  $s$  into the total order in  $L$ .
  2.  $\text{DELETE}(s, L)$ . Delete segment  $s$  from  $L$ .
  3.  $\text{LEFT}(s, L)$ . Return the name of the segment immediately left of  $s$  in the ordering in  $L$ .
  4.  $\text{RIGHT}(s, L)$ . Return the name of the segment immediately right of  $s$  in the ordering in  $L$ .
- THESE OPERATIONS CAN BE PERFORMED IN  $O(\log N)$  TIME (OR BETTER).

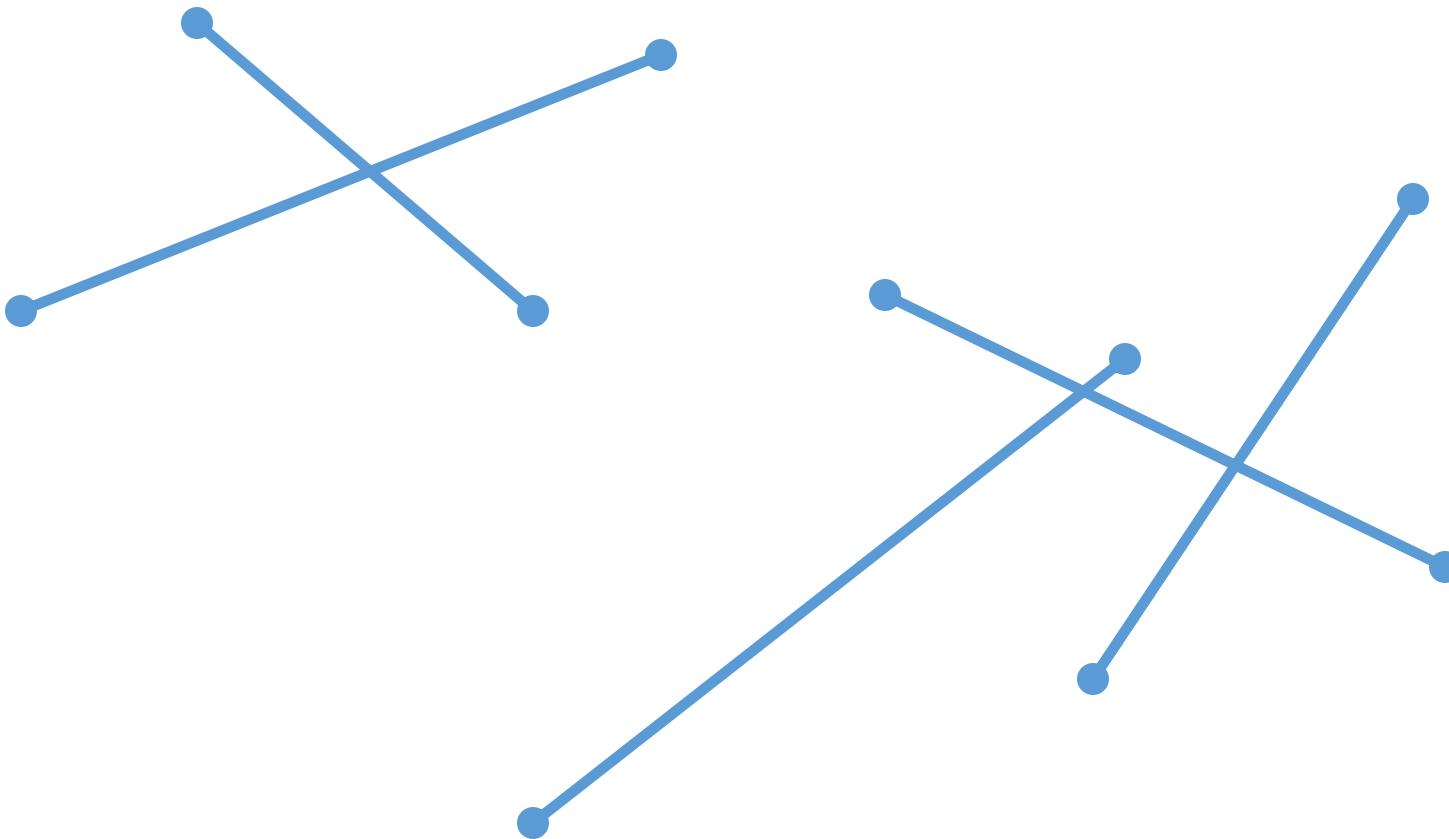


# INTERSECTION OF LINE SEGMENTS

- EVENT-POINT SCHEDULE
  - As the sweep-line is swept from top to bottom, the set of the currently comparable segments and/or their ordering by the relation of X changes at a finite set of y values; those are known as events.
- THE EVENTS FOR THIS PROBLEM ARE SEGMENT ENDPOINTS AND SEGMENT INTERSECTIONS. THE EVENT-POINT SCHEDULE DATA STRUCTURE E IS USED TO STORE EVENTS PRIOR TO THEIR PROCESSING. FOR E THE FOLLOWING OPERATIONS ARE NEEDED:
  1. MIN(E). Determine the smallest element in E (based on y), return it, and delete it.
  2. INSERT(p, E). Insert abscissa p, representing an event, into E.
  3. MEMBER(p, E). Determine if abscissa p is a member of E.
- THE PRIORITY QUEUE DATA STRUCTURE CAN PERFORM ALL OF THESE OPERATIONS IN  $O(\log N)$  TIME.



# SWEET ALGORITHM



# What Data Structures Should We Use?

- Self-balancing trees (AVL, Red-black)

Red–black tree		
Type	tree	
Invented	1972	
Invented by	Rudolf Bayer	
Time complexity in big O notation		
Algorithm	Average	Worst case
Space	$O(n)$	$O(n)$
Search	$O(\log n)^{[1]}$	$O(\log n)^{[1]}$
Insert	$O(\log n)^{[1]}$	$O(\log n)^{[1]}$
Delete	$O(\log n)^{[1]}$	$O(\log n)^{[1]}$



# What Data Structures Should We Use?

- Priority Queue

Operation	Binary <sup>[6]</sup>	Leftist	Binomial <sup>[6]</sup>	Fibonacci <sup>[6][2]</sup>	Pairing <sup>[7]</sup>	Brodal <sup>[8][a]</sup>	Rank-pairing <sup>[10]</sup>	Strict Fibonacci <sup>[11]</sup>	2-3 heap
find-min	$\Theta(1)$	$\Theta(1)$	$\Theta(\log n)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	?
delete-min	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$O(\log n)^{[b]}$	$O(\log n)^{[b]}$	$O(\log n)$	$O(\log n)^{[b]}$	$O(\log n)$	$O(\log n)^{[b]}$
insert	$O(\log n)$	$\Theta(\log n)$	$\Theta(1)^{[b]}$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$O(\log n)^{[b]}$
decrease-key	$\Theta(\log n)$	$\Theta(n)$	$\Theta(\log n)$	$\Theta(1)^{[b]}$	$o(\log n)^{[b][c]}$	$\Theta(1)$	$\Theta(1)^{[b]}$	$\Theta(1)$	$\Theta(1)$
merge	$\Theta(n)$	$\Theta(\log n)$	$O(\log n)^{[d]}$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	?

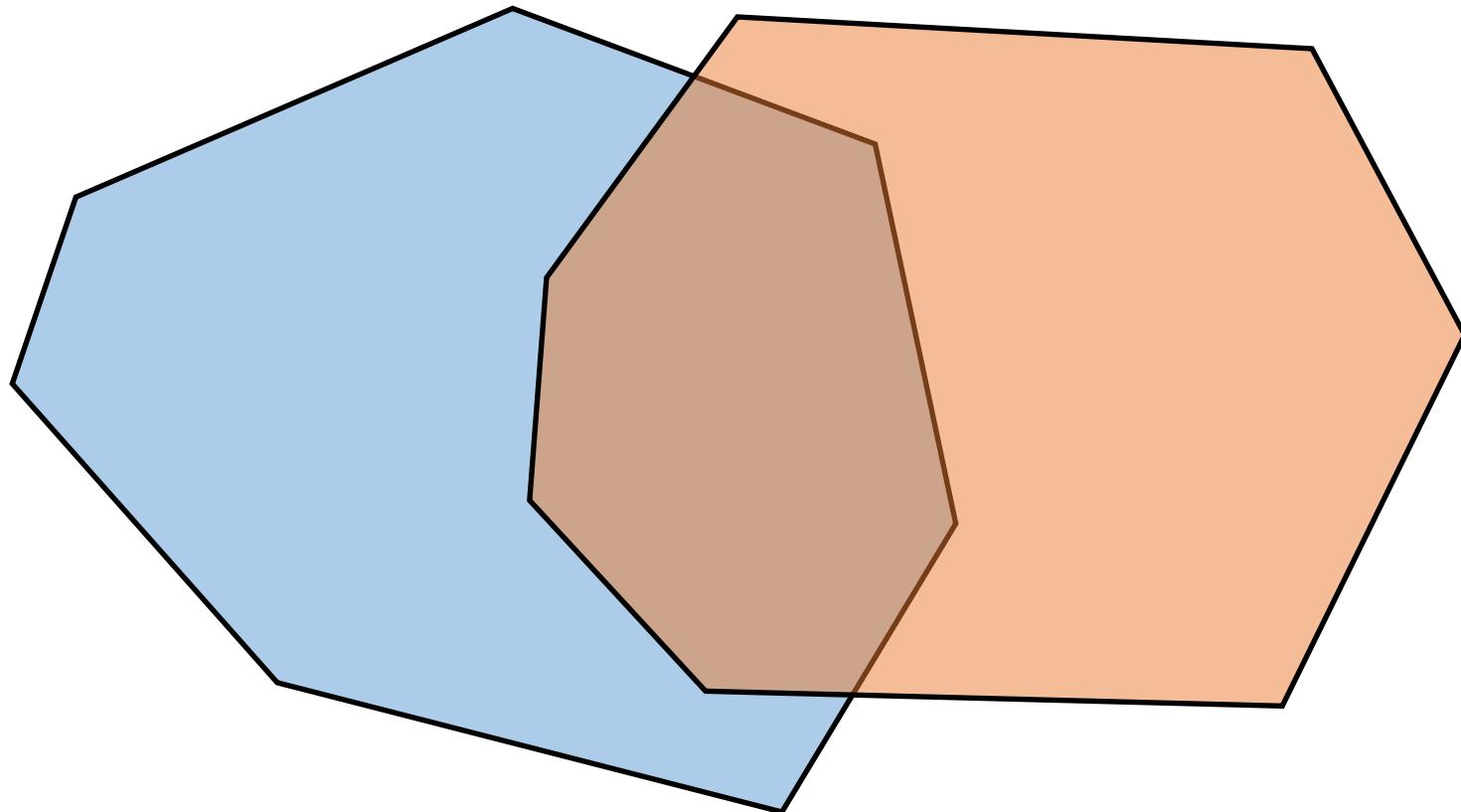


# INTERSECTION OF LINE SEGMENT

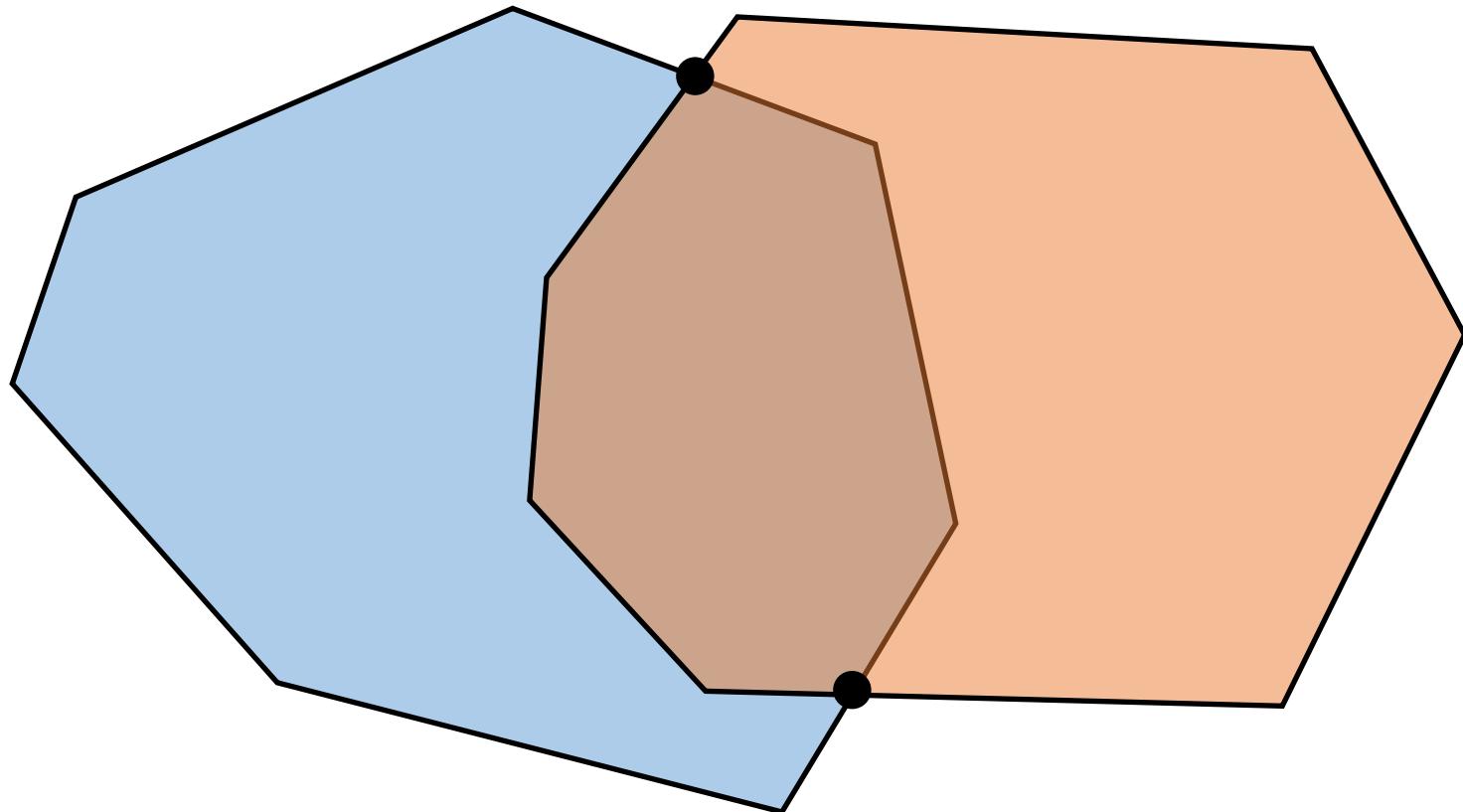
- ANALYSIS OF SHAMOS-HOEY ALGORITHM
  - Preprocessing:  $O(N \log N)$ ; sort of endpoints.
  - Query:  $O((N + K) \log N)$ ;
    - each of  $2N$  endpoints and  $K$  intersections are inserted into  $E$ , an  $O(\log N)$  operation.
  - Storage:  $O(N + K)$ ; at most  $2N$  endpoints and  $K$  intersections are stored in  $E$ .
- COMMENTS
  - As given, assumption that no segments of  $S$  are horizontal.
  - As given, assumption that no three (or more) segments meet at a point.
  - Care must be taken with intersections at segment end points.
  - Query time of  $O((N + K) \log N)$  is suboptimum; an optimum  $O(N \log N + K)$  algorithm exists, but is quite difficult.



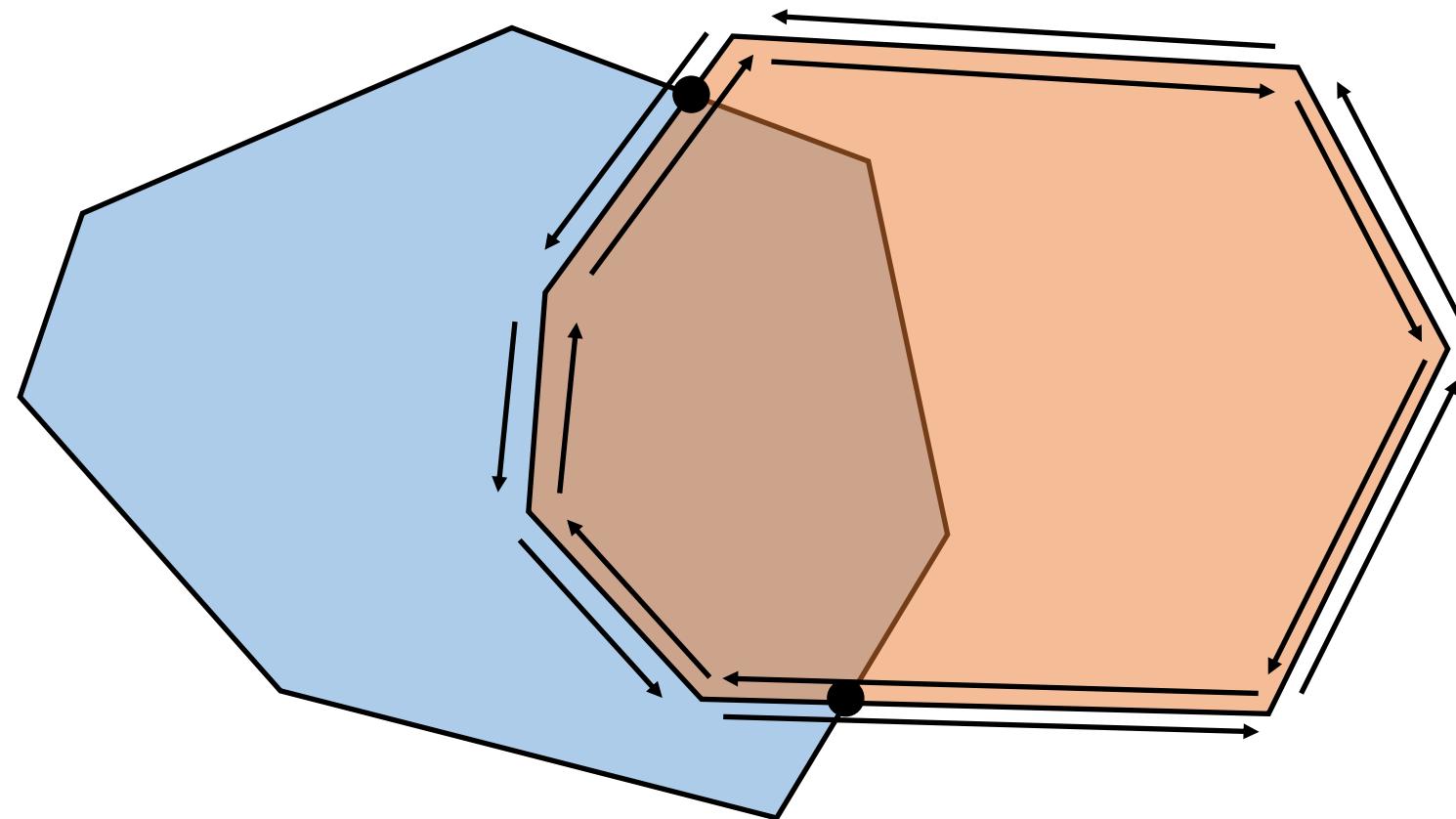
CAN WE USE THIS ALGORITHM TO FIND THE  
INTERSECTION OF 2 REGIONS?



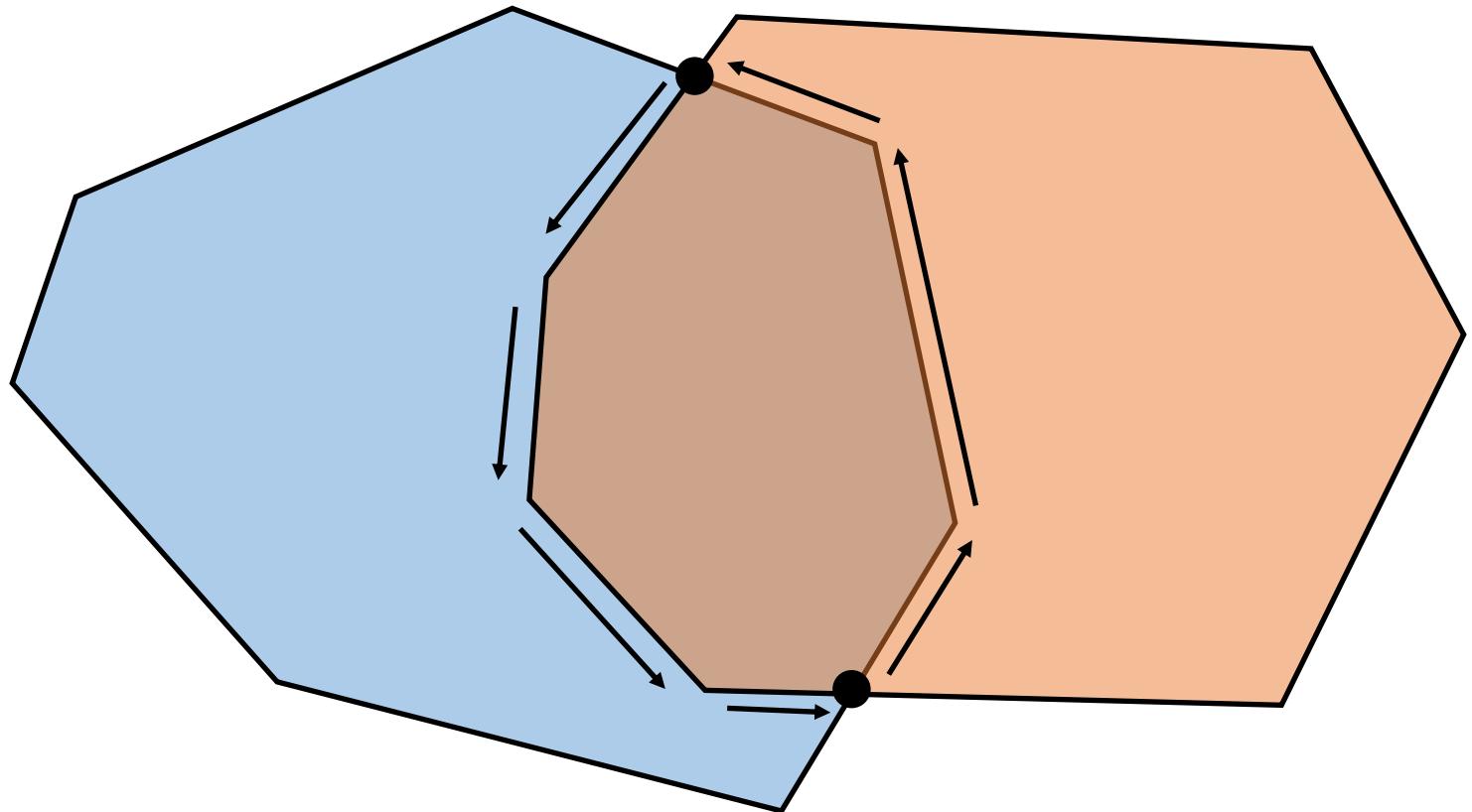
# RUN SEEPE ALGORITHM



# CONNECTIVITY STORED AS DOUBLY LINKED LIST



# WALK THE GRAPH



# INTERSECTION OF LINE SEGMENT

- ANALYSIS
  - Preprocessing: Intersection finding  $O((N + K) \log N)$
  - Query:  $O(N + M)$
  - Storage:  $O(N + M)$  edge links
- CAN WE DO ANY BETTER?



