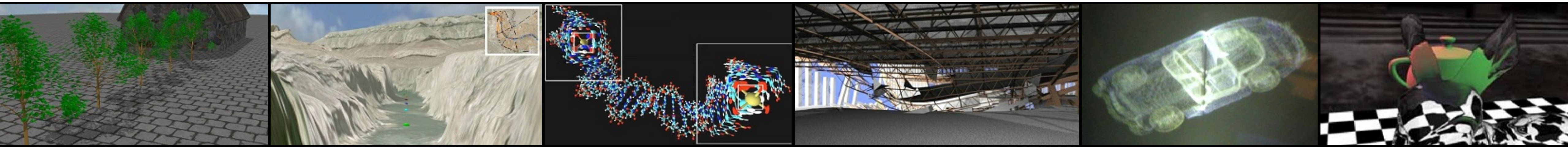
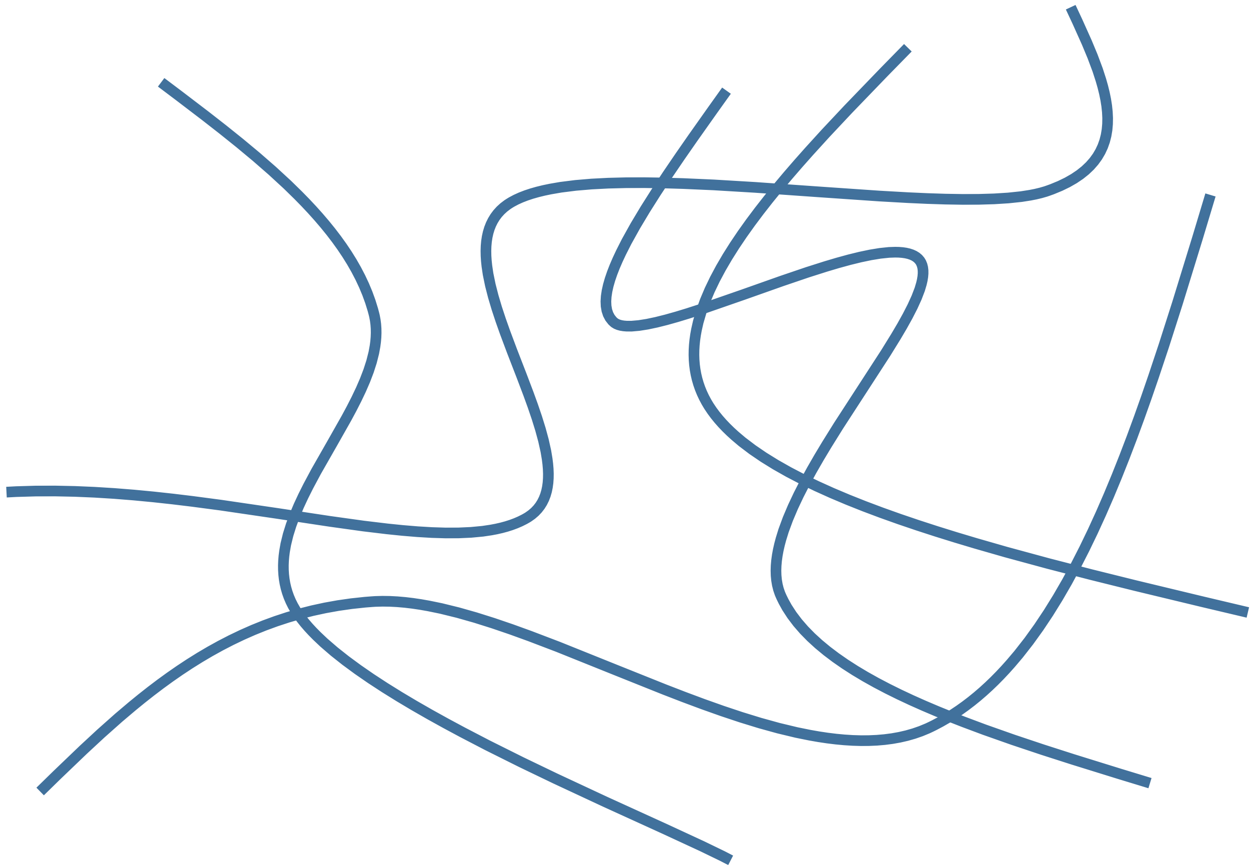
COT 4521: Introduction to Computational Geometry

Segment Intersection

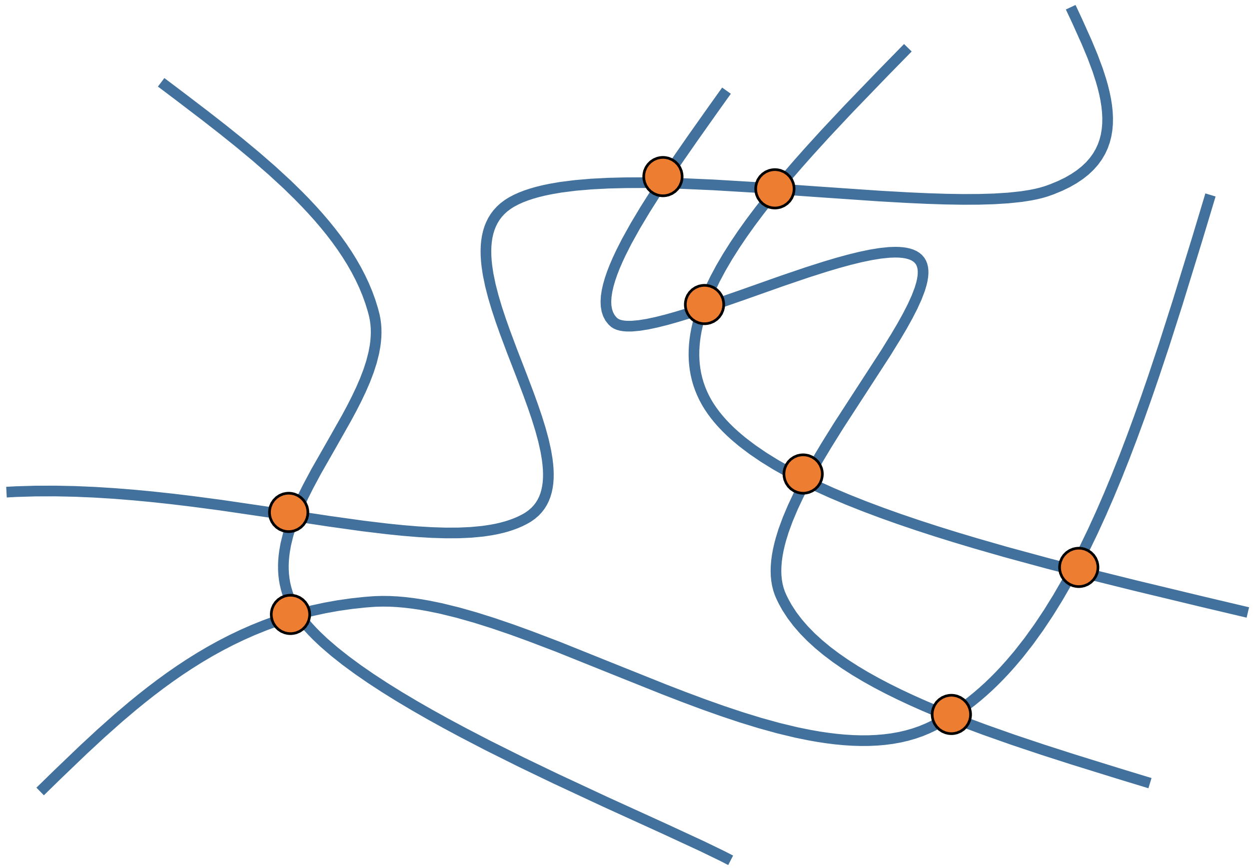
Paul Rosen
Assistant Professor
University of South Florida



PROBLEM STATEMENT

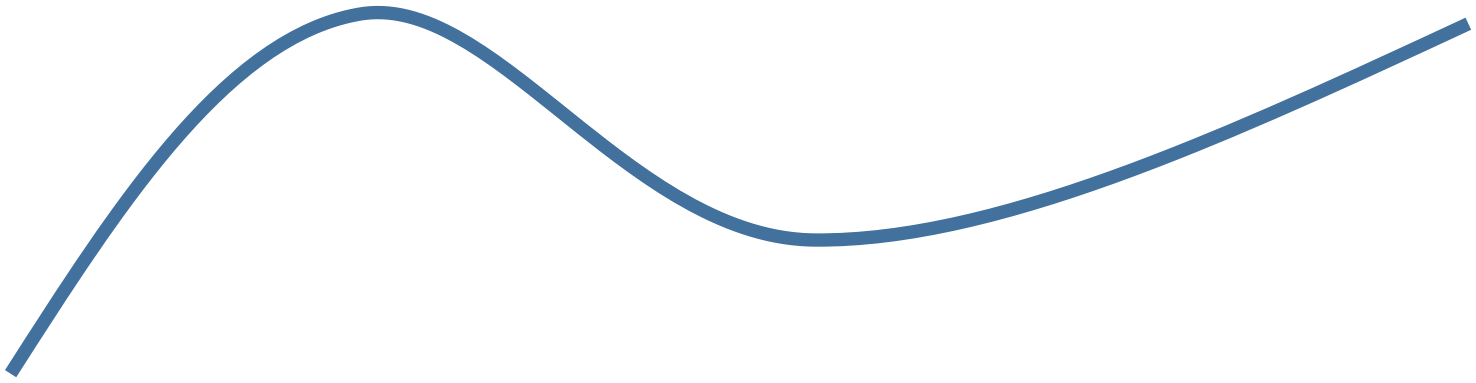


Problem Statement



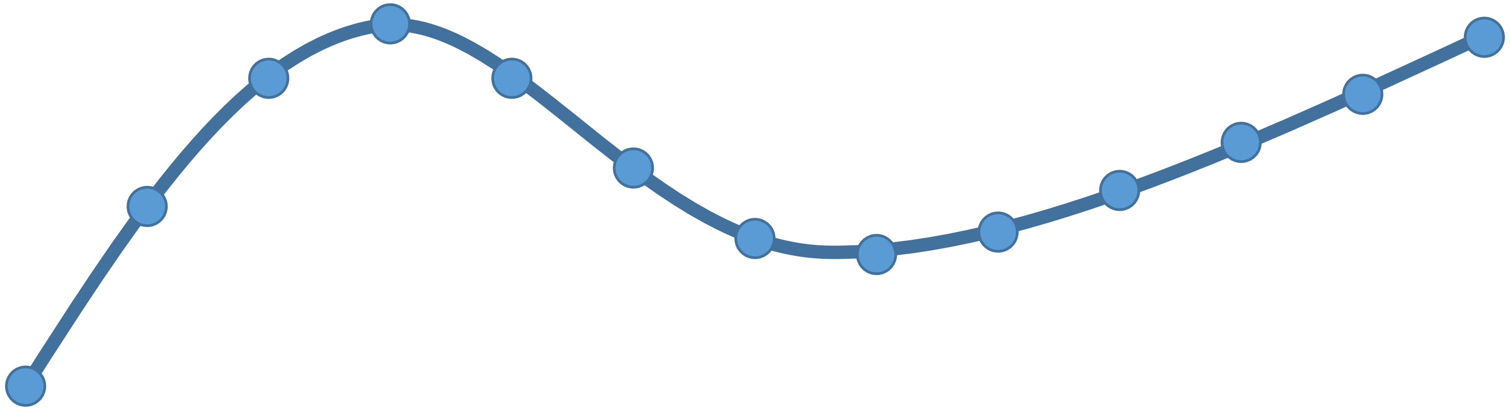
Representing Curves

- Curves often represented by a polynomial or polynomial spline
  - Bezier, NURBS, etc.
- Tesselate Curve into many small line segments



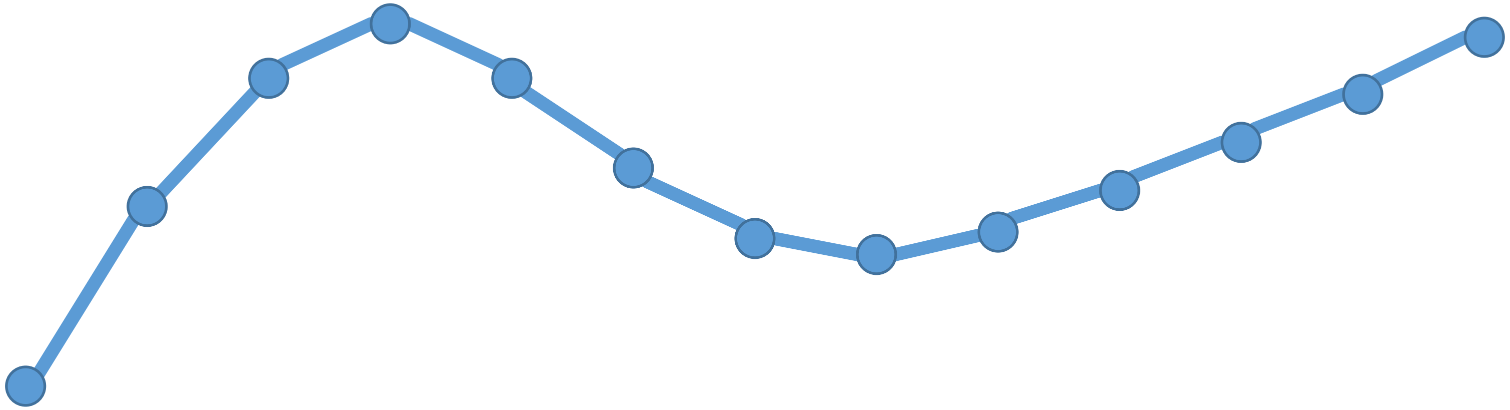
Representing Curves

- Curves usually represented by a polynomial or polynomial spline
  - Bezier, NURBS, etc.
- Tesselate Curve into many small line segments



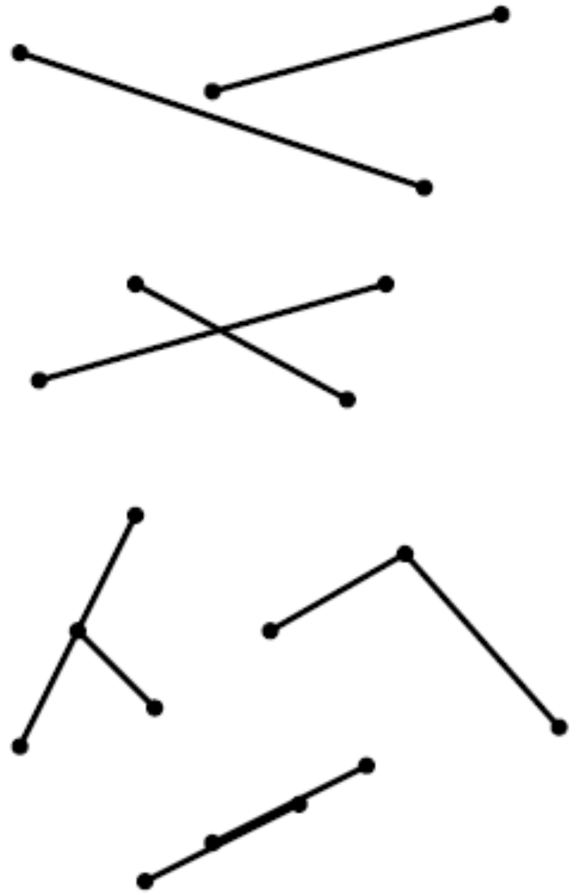
Representing Curves

- Curves usually represented by a polynomial or polynomial spline
  - Bezier, NURBS, etc.
- Tesselate Curve into many small line segments



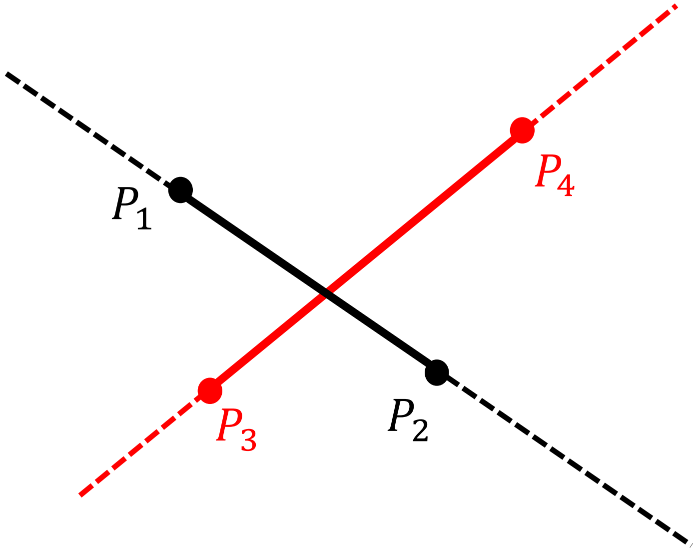
Segment-Segment Intersection

- 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*



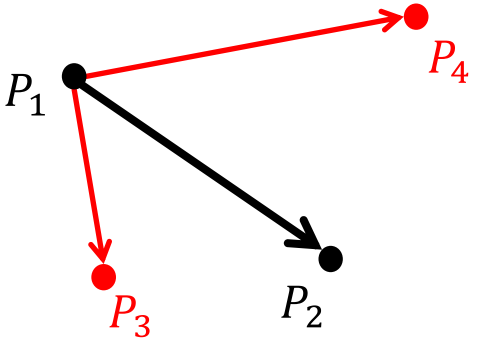
Do they intersect?

- 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?

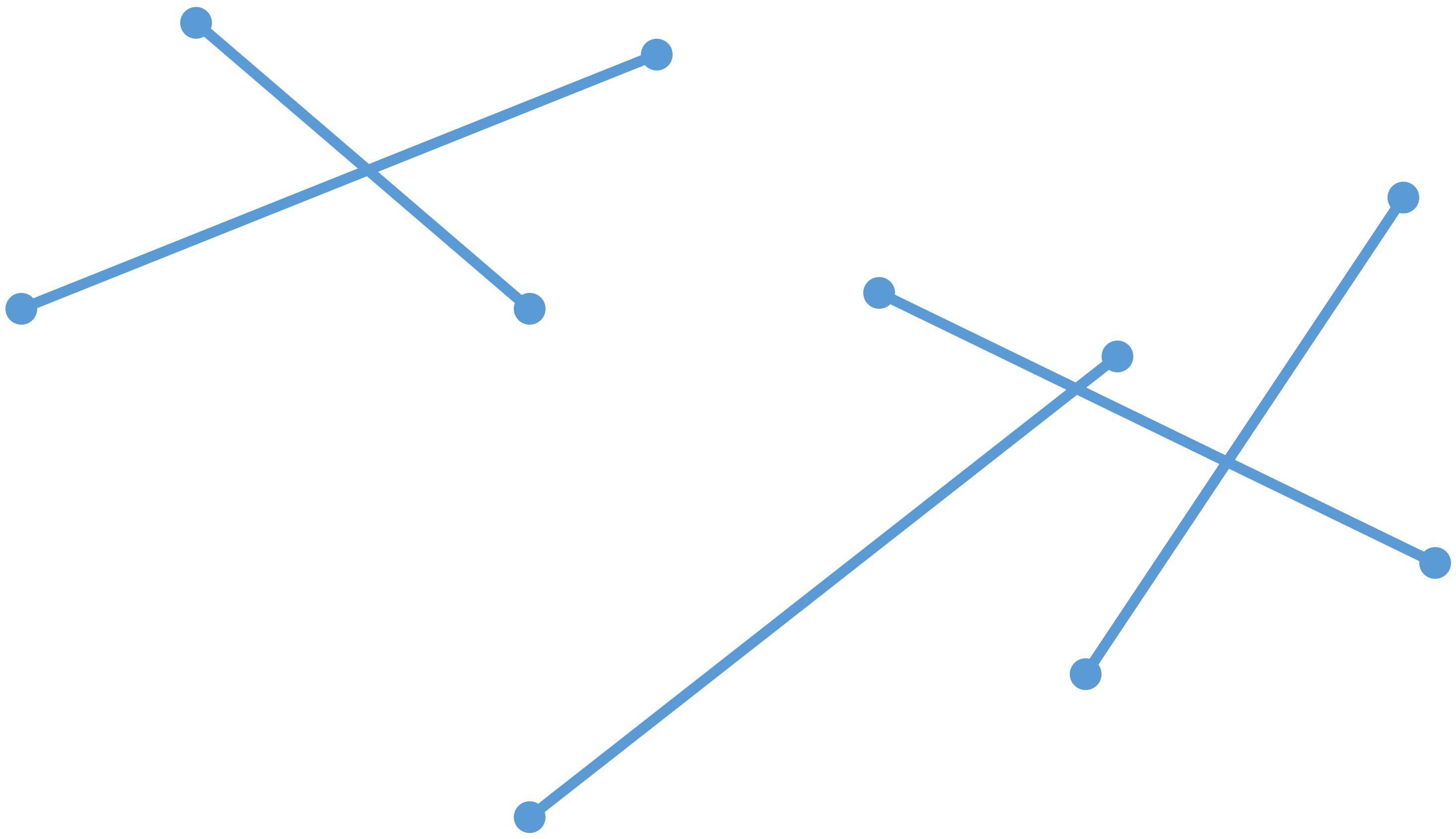
Representing a Line

- Parametric Form:

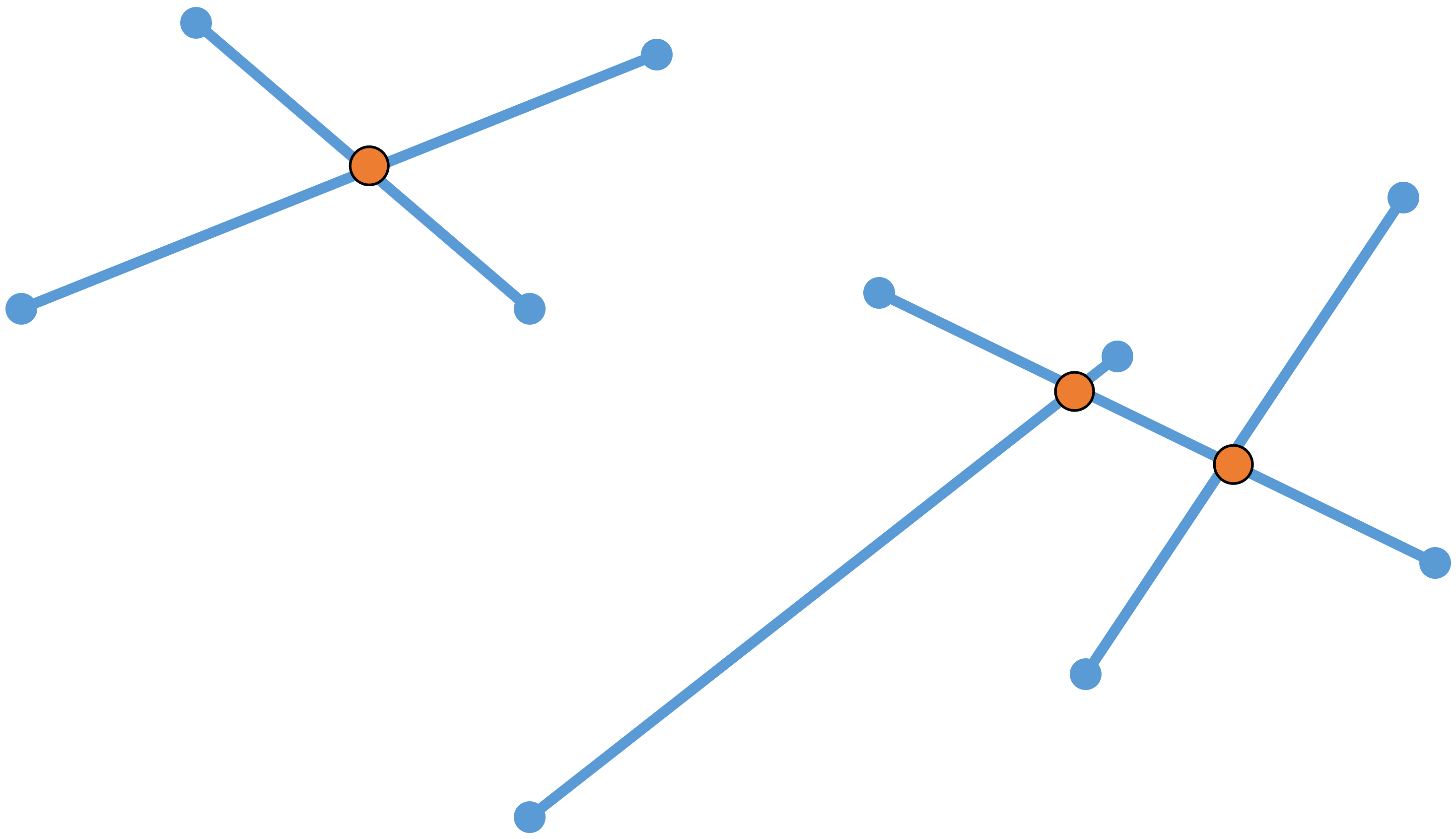
$$P = P_0 + Dt$$

- 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 $S = \{s_1, s_2, \ldots, s_N\}$ of line segments in the plane;
    - For $1 \le i \le N, s_i = (\mathrm{P}_{i1}, \mathrm{P}_{i2})$ (endpoints of the segments); and
    - For $1 \le j \le 2, \mathrm{P}_{ij} = (x_{ij}, y_{ij})$ (coordinates of the endpoints).
  - Question:
    - Report all points of intersection of segments in S.

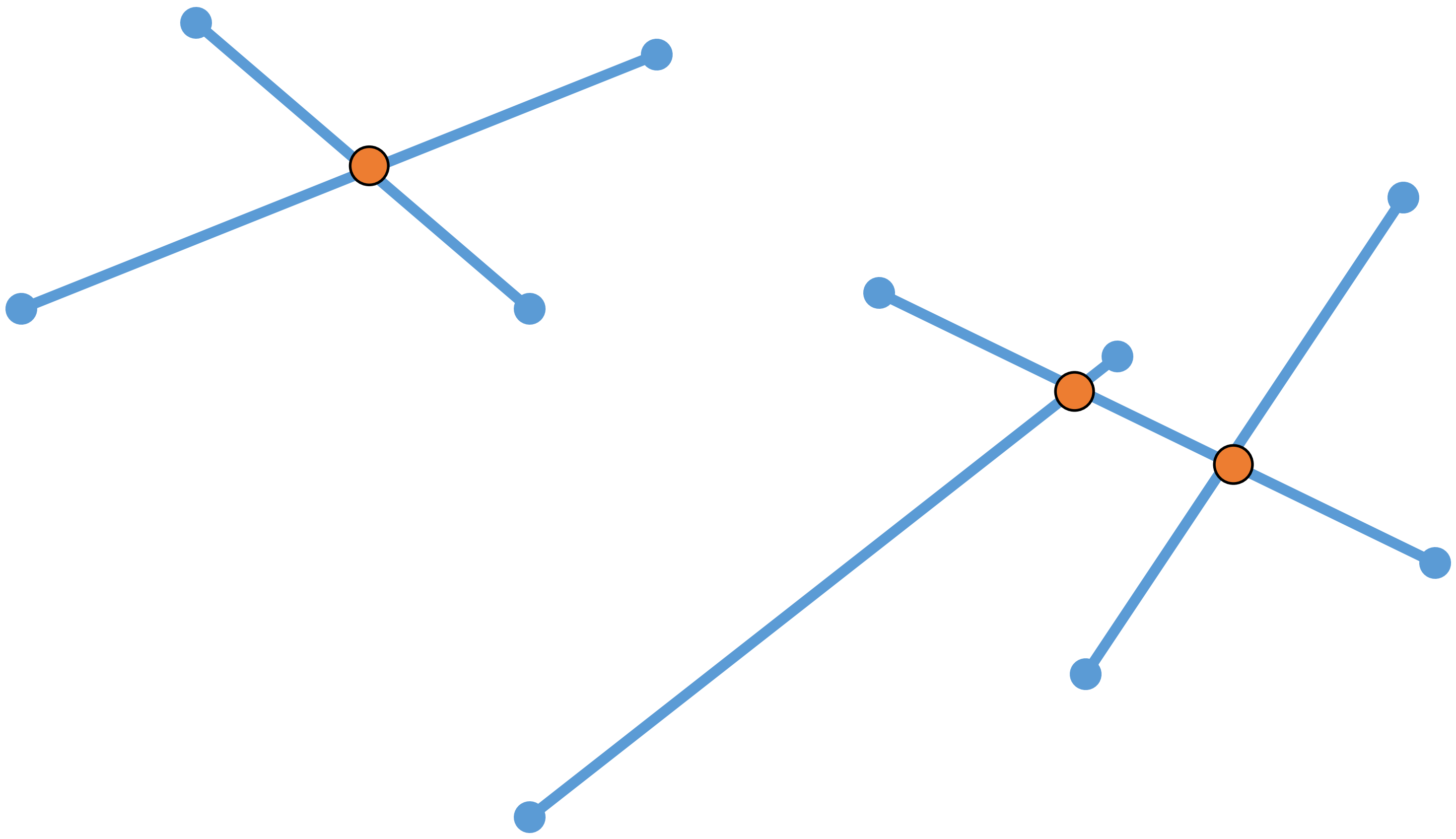
Intersection of line segments

- 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.

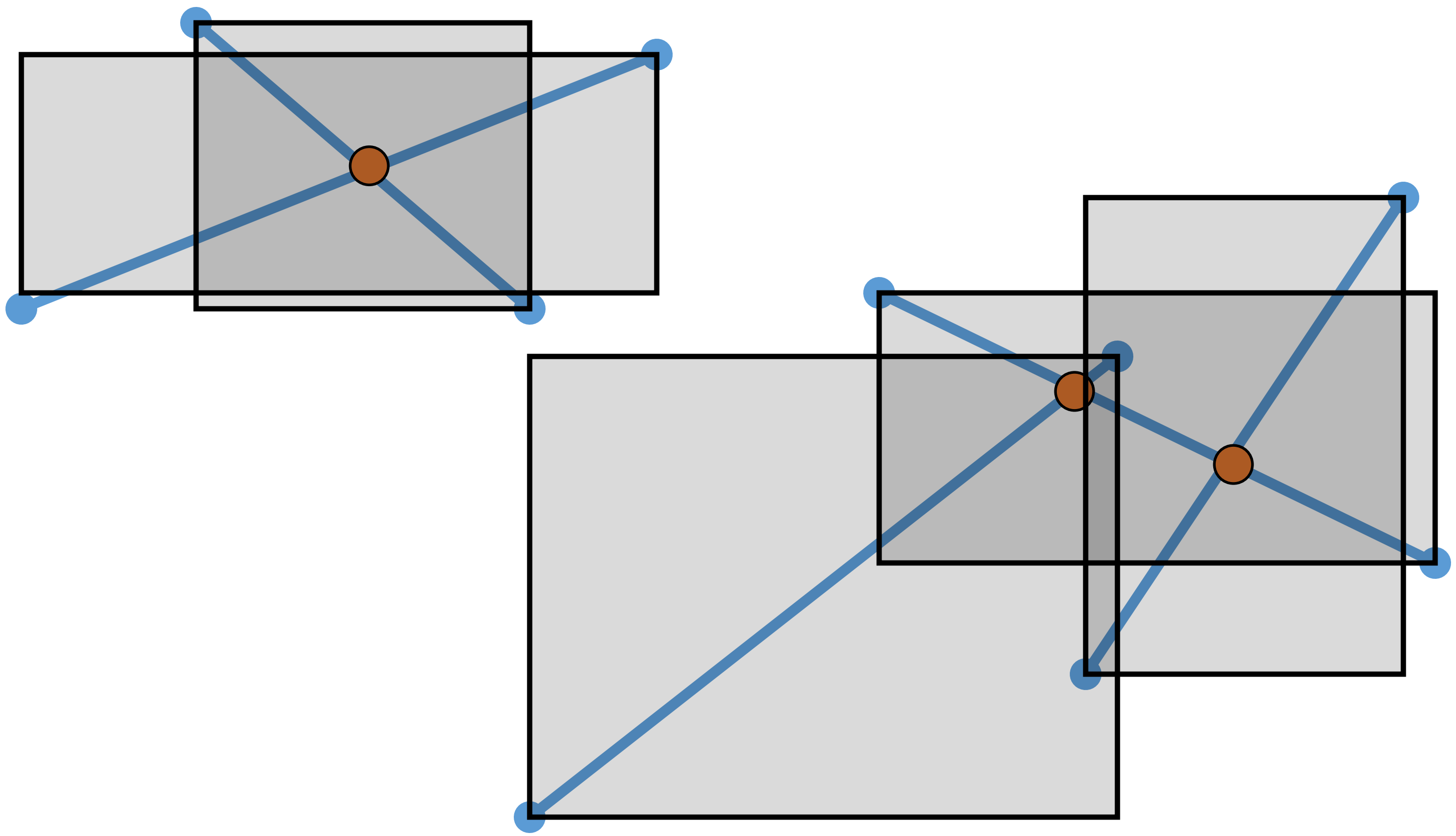
Naïve Intersection of >2 Line Segments

- 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?

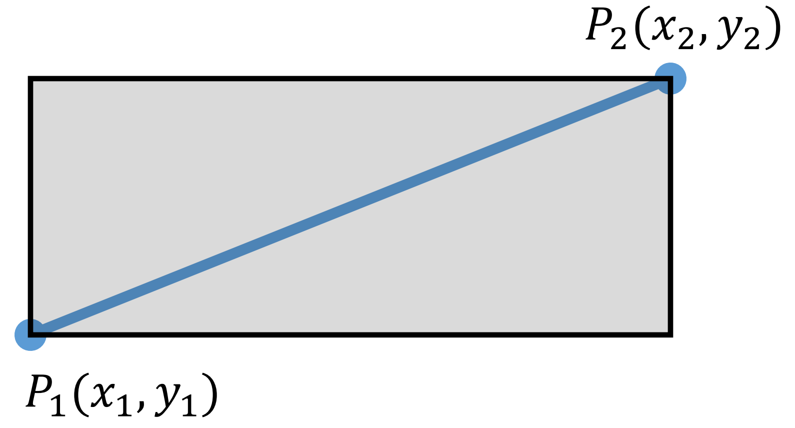


Axis Aligned Bounding Boxes (AABB)



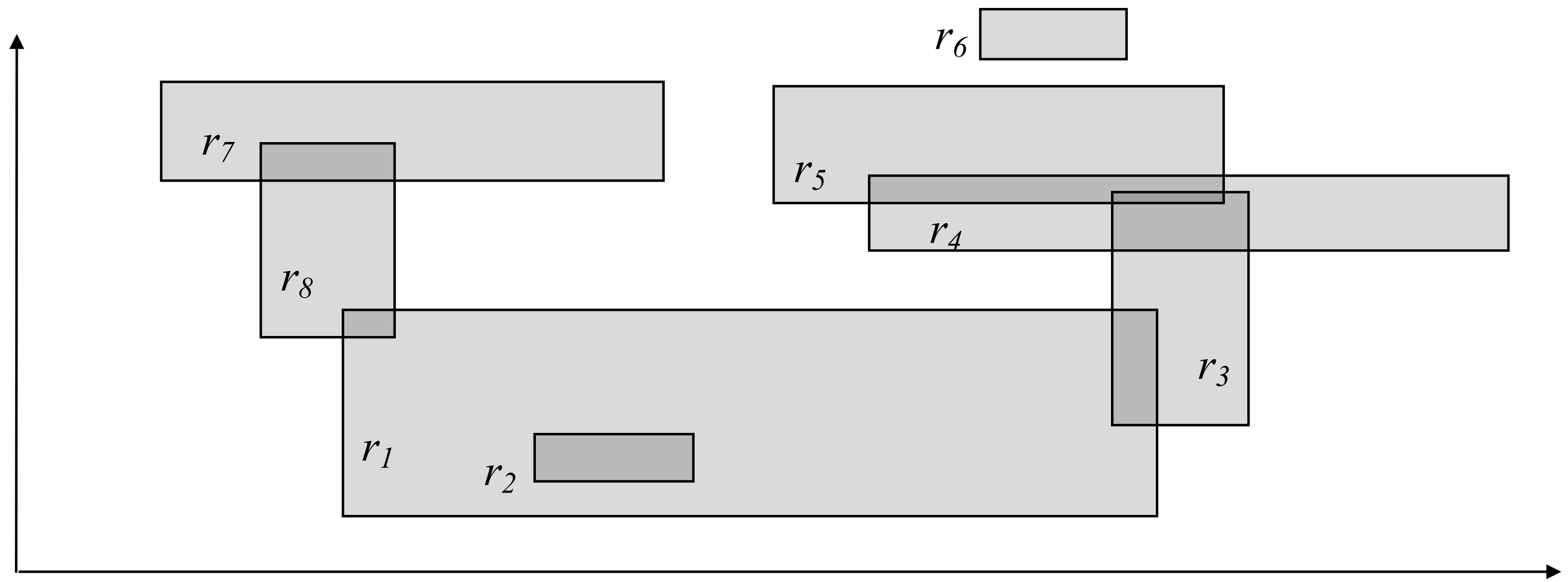
Axis Aligned Bounding Boxes (AABB)

- 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

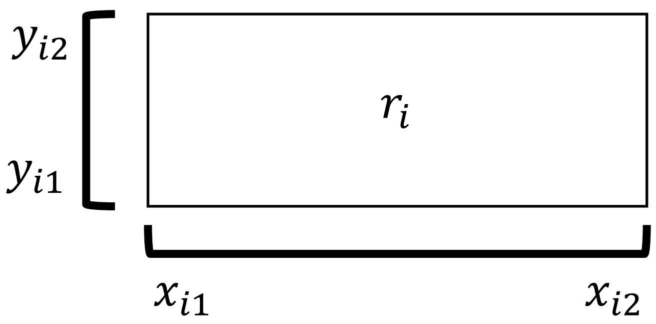
- Given a set of N axis-parallel rectangles in the plane, report all intersecting pairs
  - Intersect ≡ share at least one point

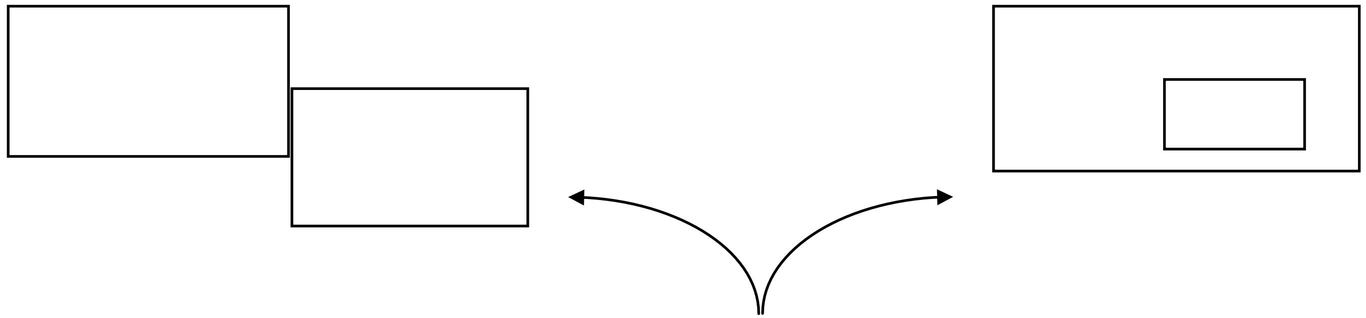


Answer: (r_1, r_2) (r_1, r_3) (r_1, r_8) (r_3, r_4) (r_3, r_5) (r_4, r_5) (r_7, r_8)

INTERSECTION OF RECTANGLES PROBLEM DEFINITION

- RECTANGLE INTERSECTION
- INSTANCE: Set $S = \{r_1, r_2, \ldots, r_N\}$ 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.)

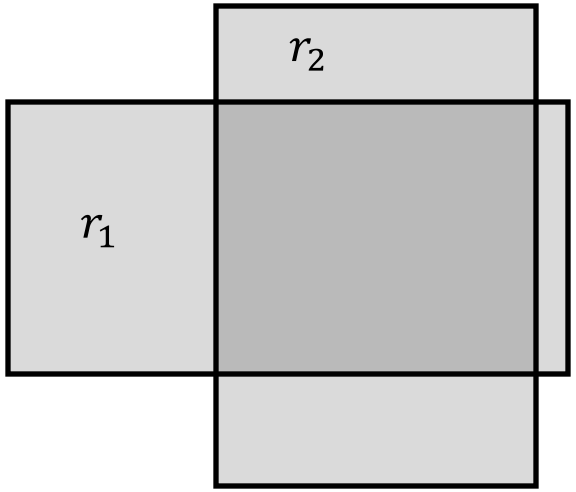




Considered to intersect.

Checking if 2 Rectangles Intersect

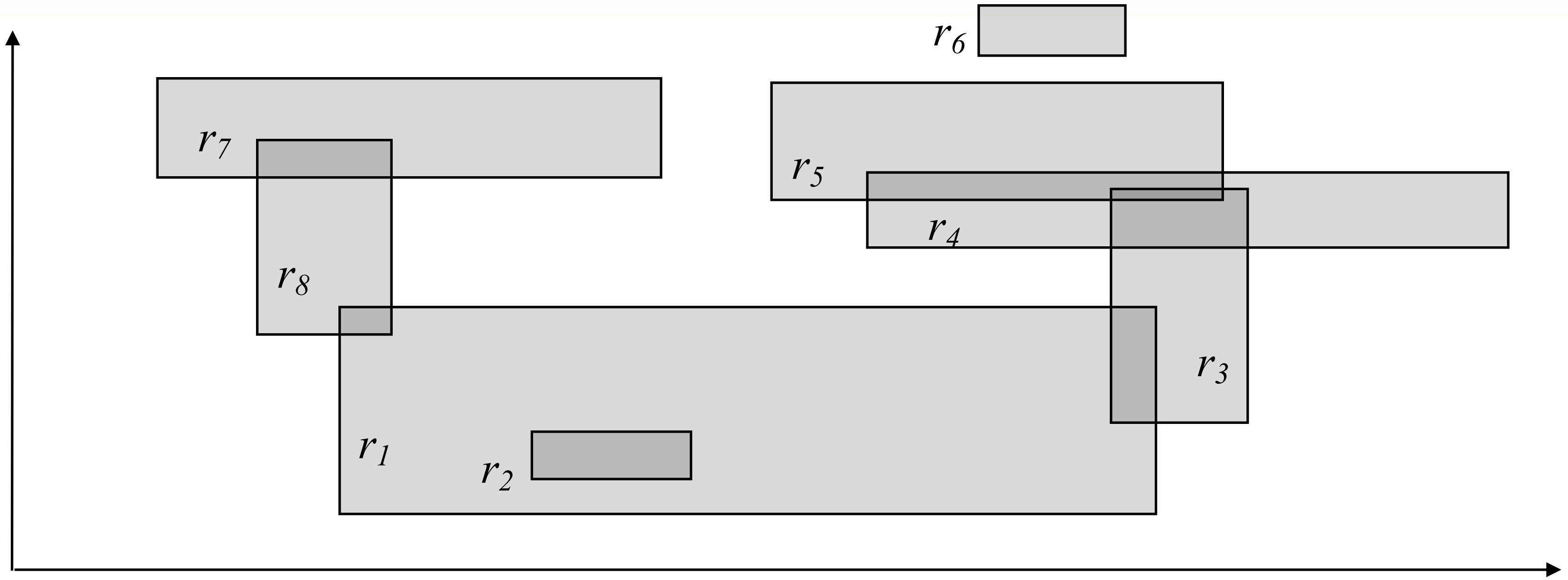
- $r_1 \cap r_2$, IF?
  - $[x_{11}, x_{12}] \cap [x_{21}, x_{22}] \neq \emptyset$ AND $[y_{11}, y_{12}] \cap [y_{21}, y_{22}] \neq \emptyset$
- How do we code this intersection?
  - $[x_{11}, x_{12}] \cap [x_{21}, x_{22}] = [\max(x_{11}, x_{21}), \min(x_{12}, x_{22})]$
  - $[y_{11}, y_{12}] \cap [y_{21}, y_{22}] = [\max(\mathrm{y}_{11}, y_{21}), \min(y_{12}, y_{22})]$
  - Check that the range of both intersections is ≥ 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: (r1, r2) (r1, r3) (r1, r8) (r3, r4) (r3, r5) (r4, r5) (r7, r8)

Intersection of rectangles

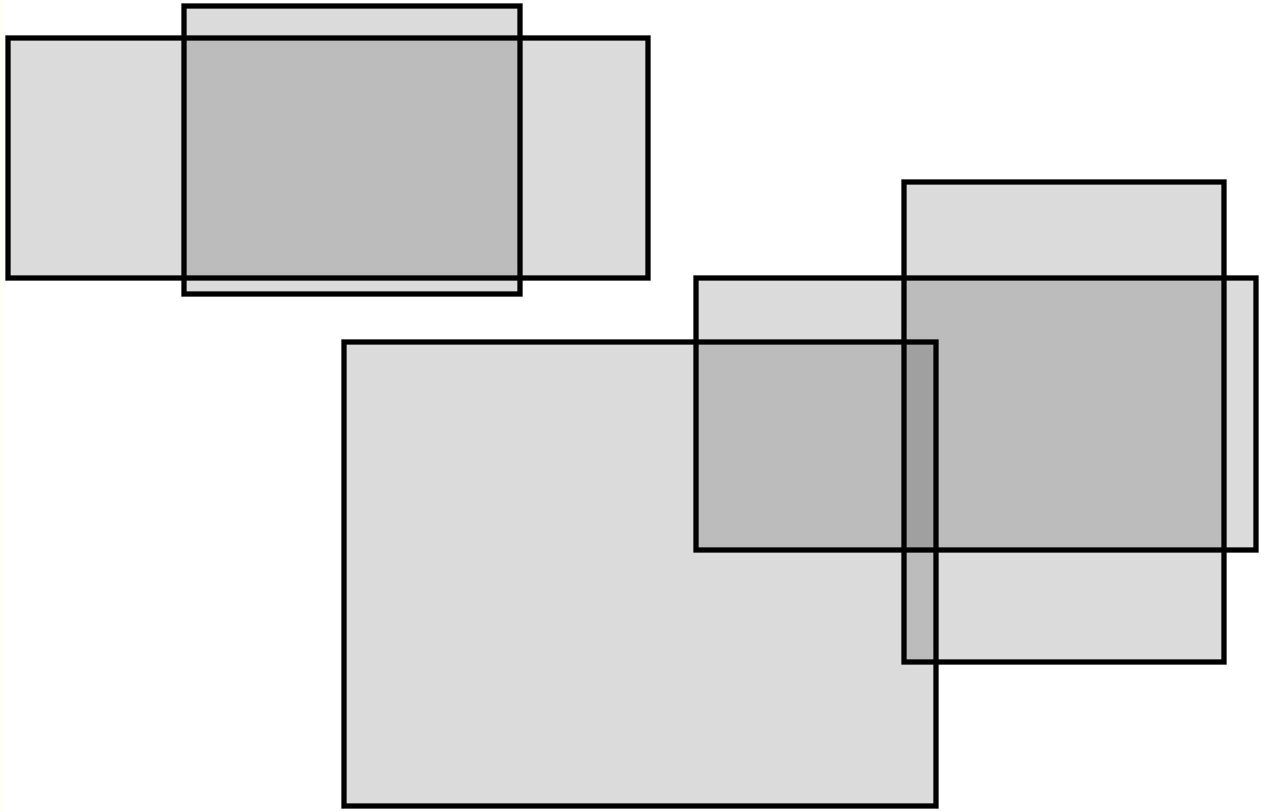
- Analysis
  - Preprocessing?
    - None
  - Query?
    - $\frac{N(N-1)}{2} = O(N^2)$
  - Storage?
    - $O(N)$

- Does this really help us with the Segment intersection problem?

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 remove rectangle X-interval from active set.

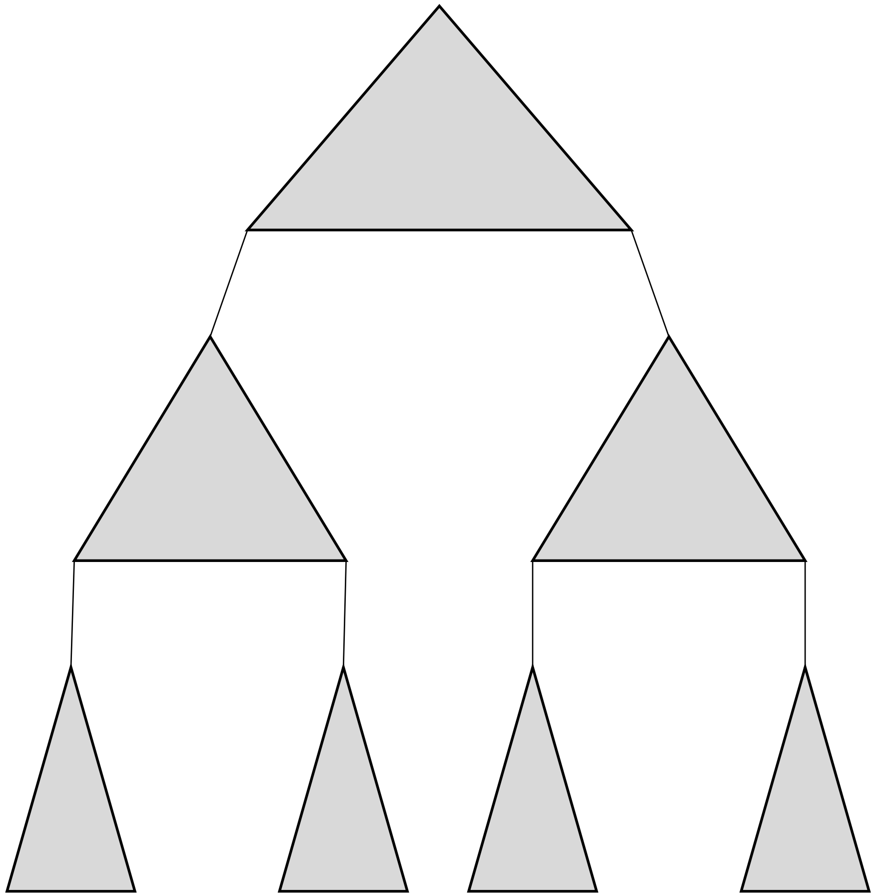


INTERSECTION OF RECTANGLES

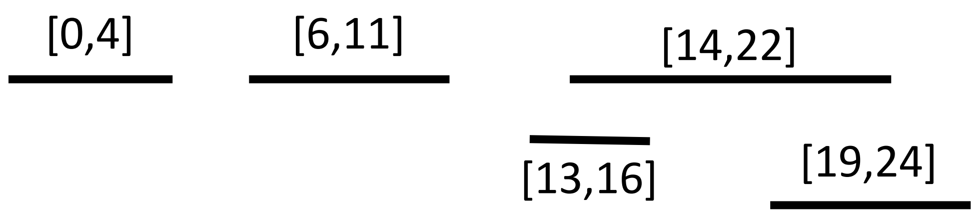
- ANALYSIS
  - Preprocessing: $O(N \log N)$; ordering intervals for sweep
  - Query: Worst case $O(N^2)$; Best case $O(N)$
  - Storage: $O(N)$; active rectangles $O(N)$, event queue $O(N)$.
- COMMENTS
  - We haven't improved worst case, but the best case has gotten significantly better.
  - We are now output sensitive.
  - Can we do any better?

1D Centered Interval Trees

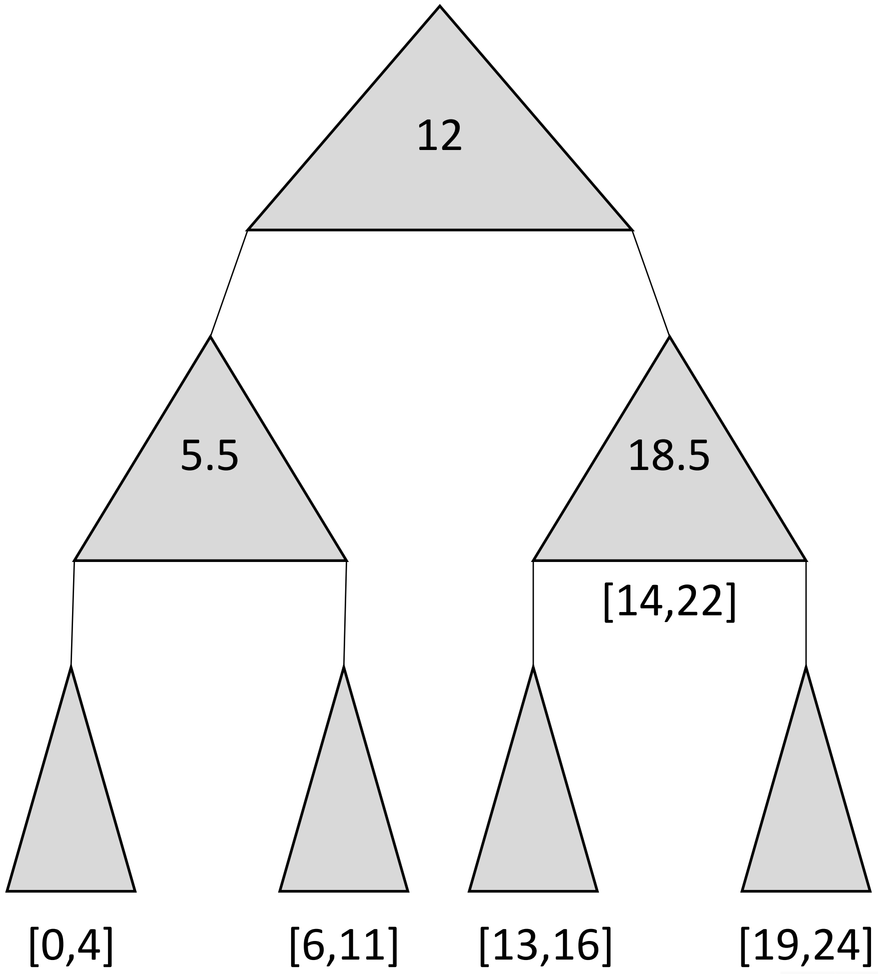
- Tree-based Data structure
- Each node stores a center point
  - Intervals are placed into 3 group,
    - Left of center—placed in left subtree
    - Right of center—placed in right subtree
    - Covering center—placed in a specialized list*



1D Centered Interval Trees



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



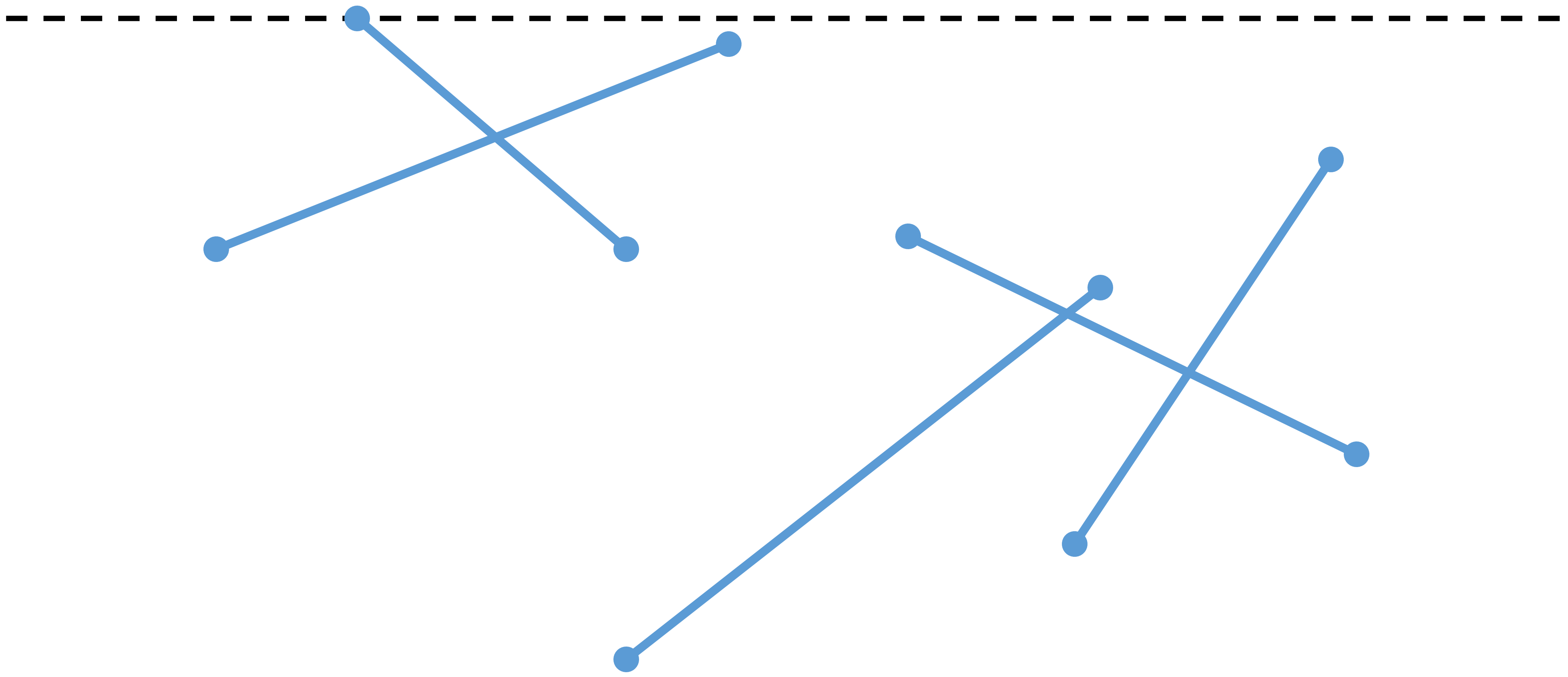
Intersection of rectangles

- 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 $N \log N = 2 \cdot 10^7$

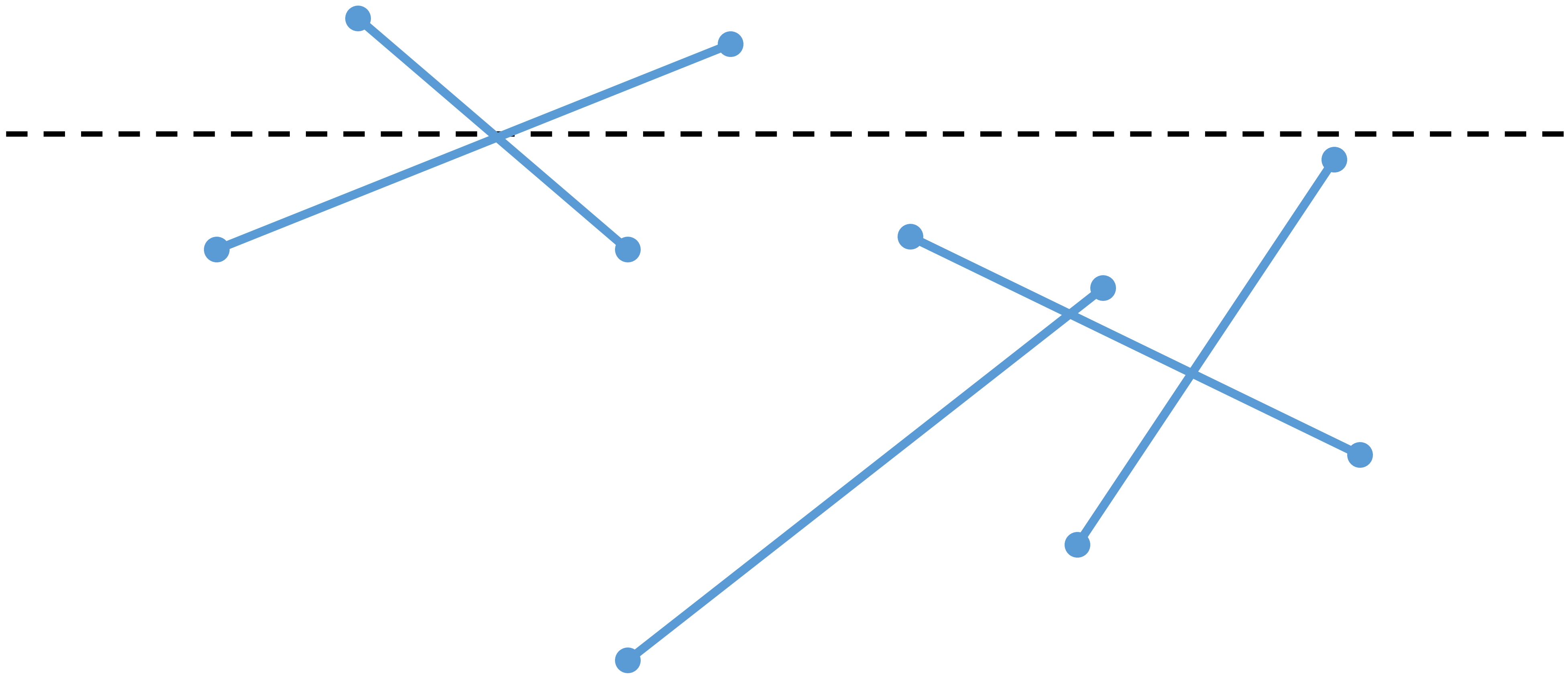
Back to the Segment Intersection Problem

- 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

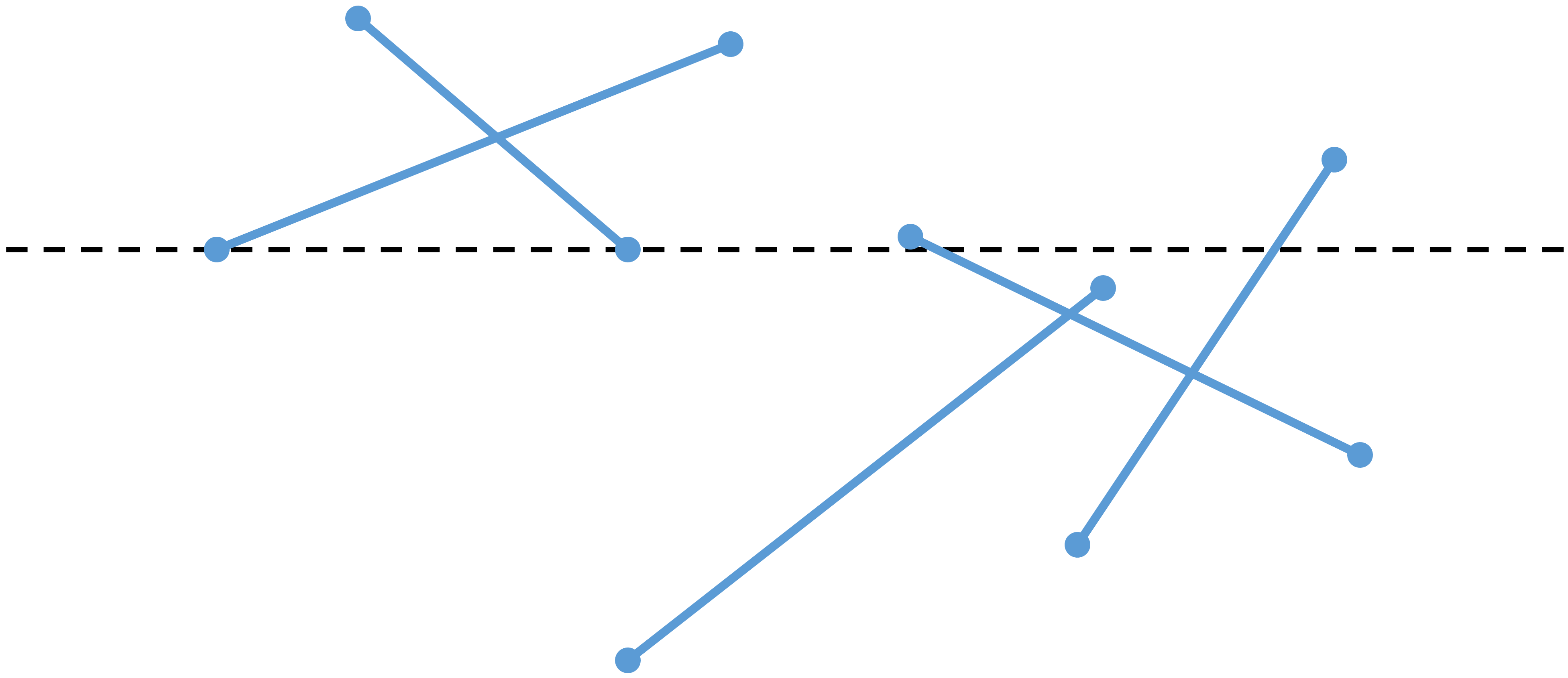
Sweep Algorithm



Sweep 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

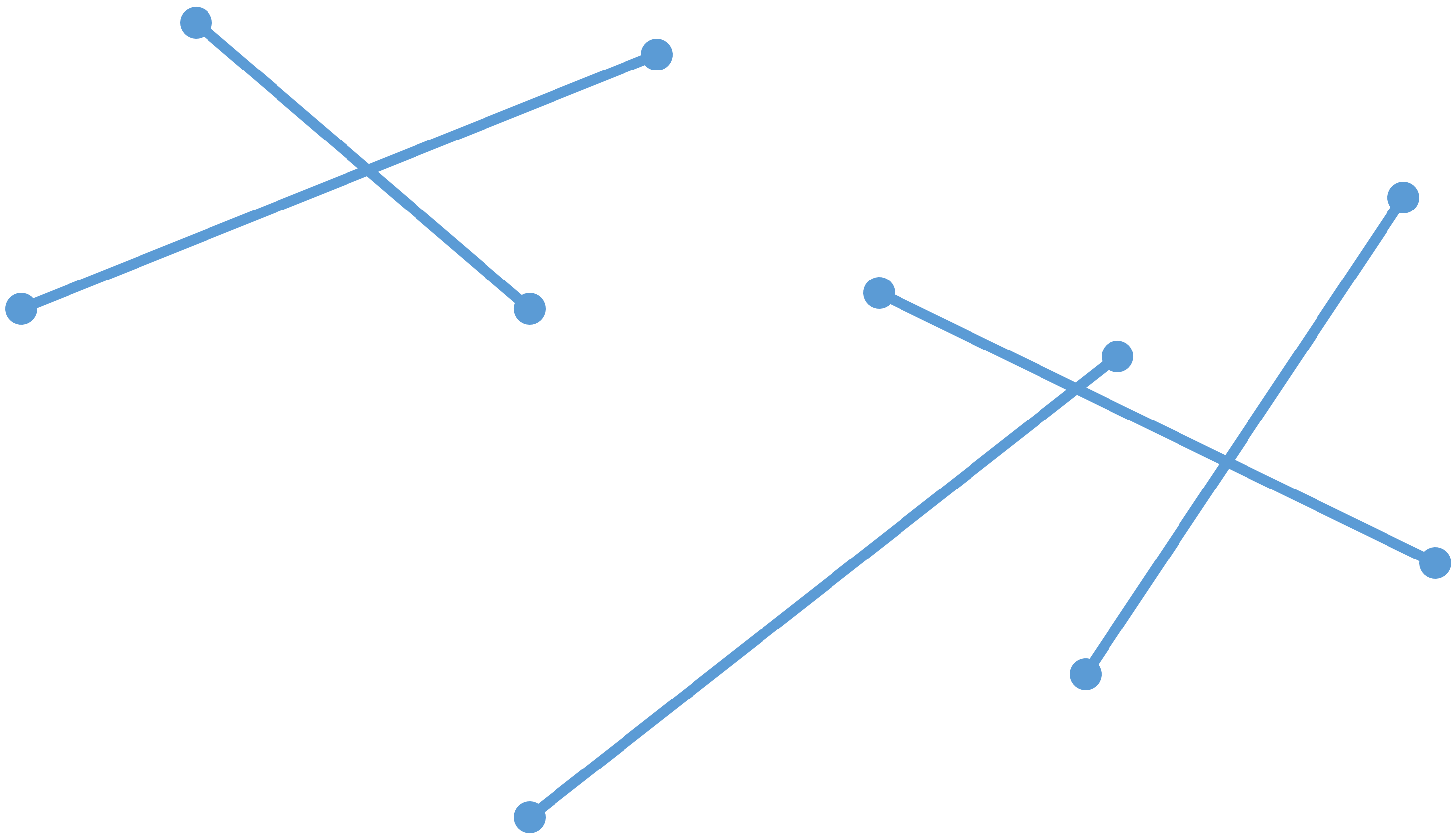
Intersection of line segments

- 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. INSERT(s, L). Insert segment s into the total order in L.
  2. DELETE(s, L). Delete segment s from L.
  3. LEFT(s, L). Return the name of the segment immediately left of s in the ordering in L.
  4. 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.

Sweep 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[6]	Leftist	Binomial[6]	Fibonacci[6][2]	Pairing[7]	Brodal[8][a]	Rank-pairing[10]	Strict Fibonacci[11]	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.