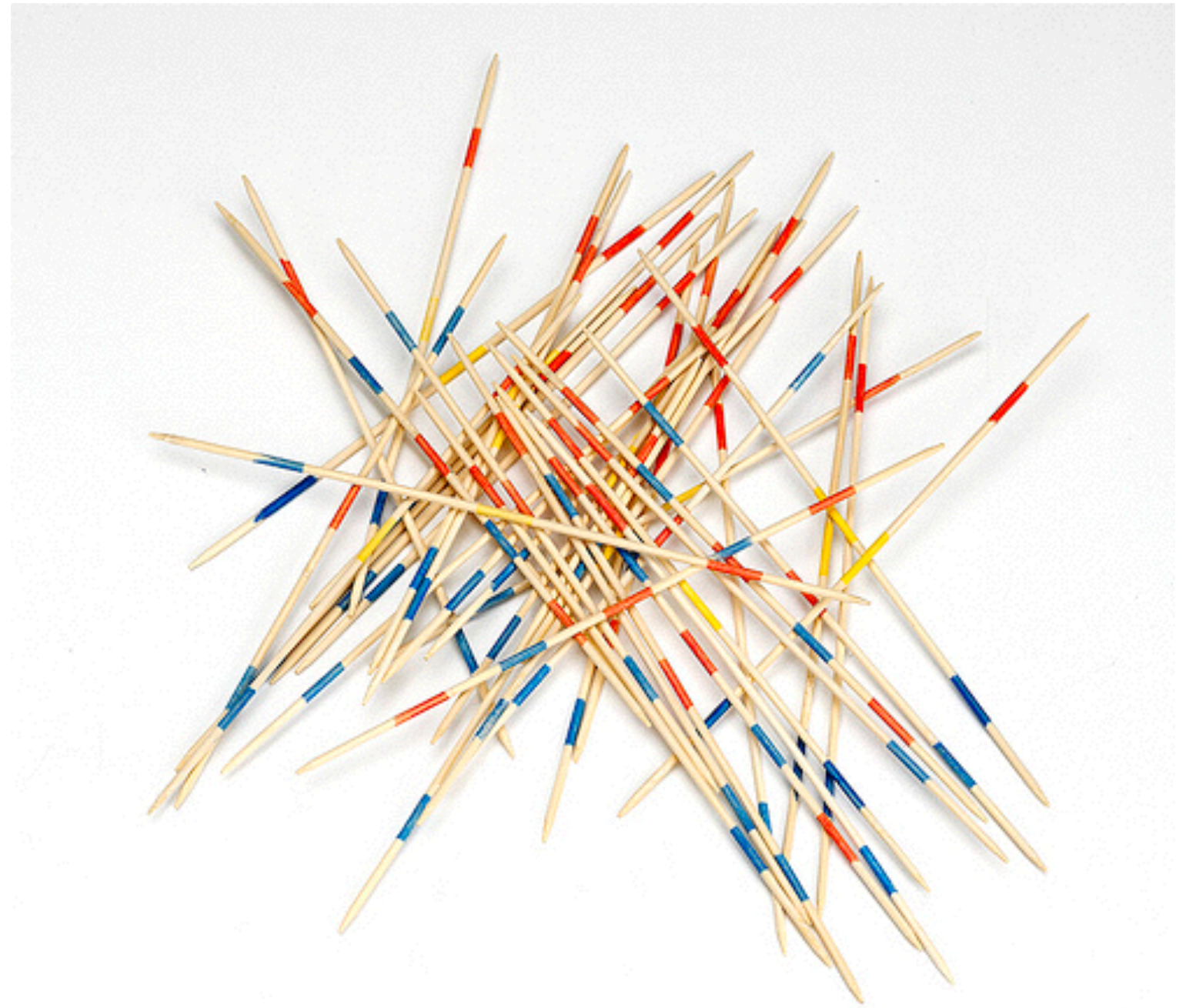
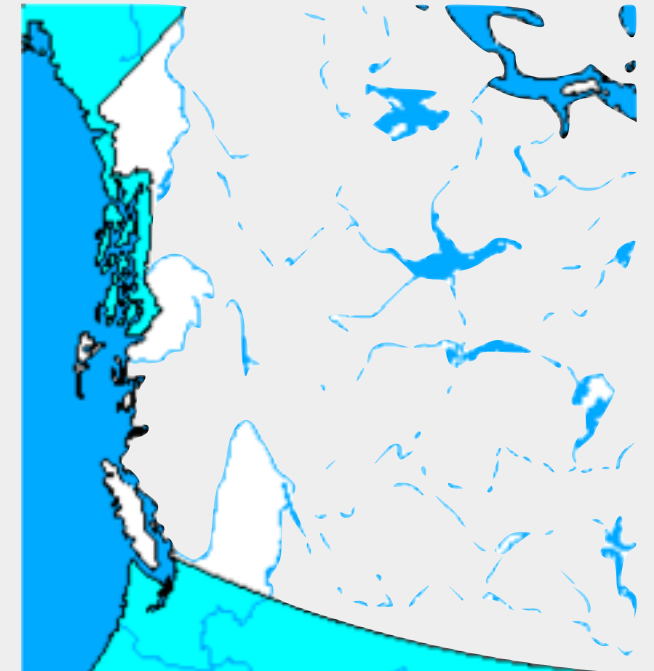
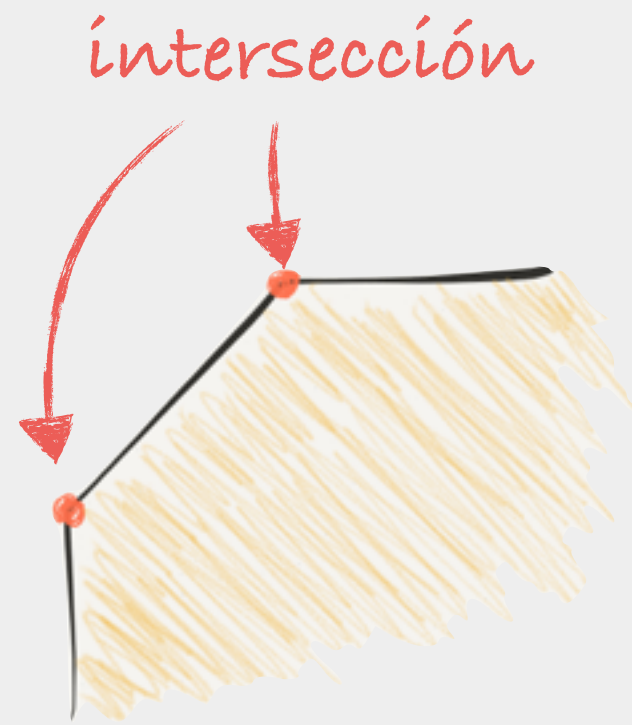


# Intersección de Segmentos de Recta

Geometría Computacional , MAT-125



Dados dos conjuntos de segmentos de recta, calcular todas las intersecciones entre los segmentos de un conjunto y los segmentos del otro conjunto.



Dado un conjunto  $S$  de  $n$  segmentos de recta cerrados en el plano, reportar todos los puntos de intersección en  $S$ .

- ▶ ¿Algoritmo de fuerza bruta?
- ▶ ¿Complejidad?
  
- ▶  $O(n^2)$
- ▶ Óptimo cuando hay intersección entre todos los pares.



Queremos un algoritmo cuyo tiempo de cálculo dependa no solo del número de segmentos en la entrada sino también del número de puntos de intersección.

*Output-Sensitive Algorithm*

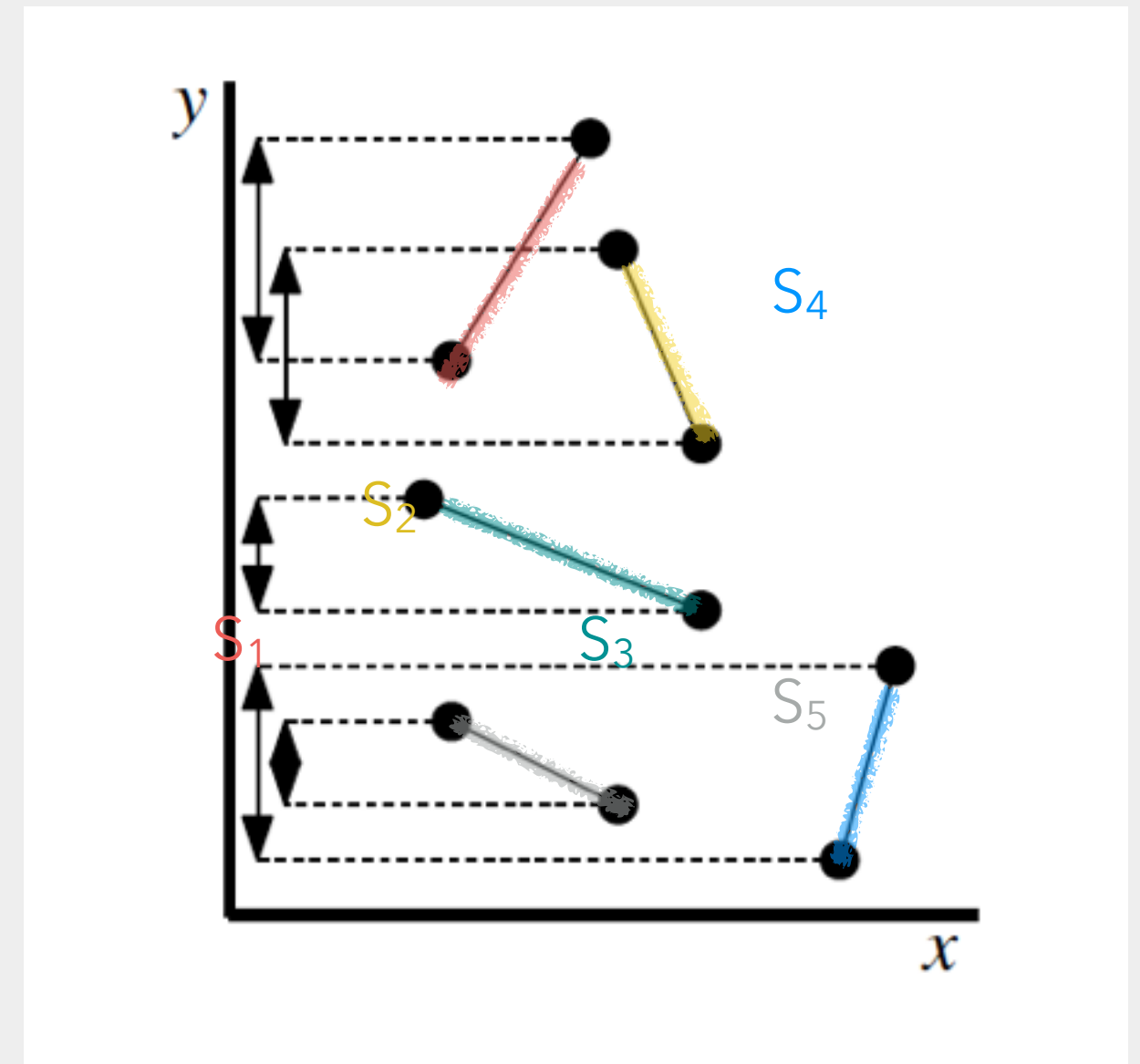
- ▶ ¿Cómo podemos evitar probar todos los segmentos?
- ▶ Segmentos cercanos son candidatos, lejanos no.

*Idea 1:*

*Sweepline algorithm*

Si el intervalo- $y$  de los segmentos no traslapa no hay intersección.

Barrer una línea horizontal de arriba hacia abajo y mantener una lista de segmentos que la intersecan.



► El **estado de la línea de barrido** es el conjunto de segmentos que la intersecan en ese momento.

► El estado cambia mientras la línea avanza hacia abajo pero no de forma continua.

¿Cuándo cambia el estado?

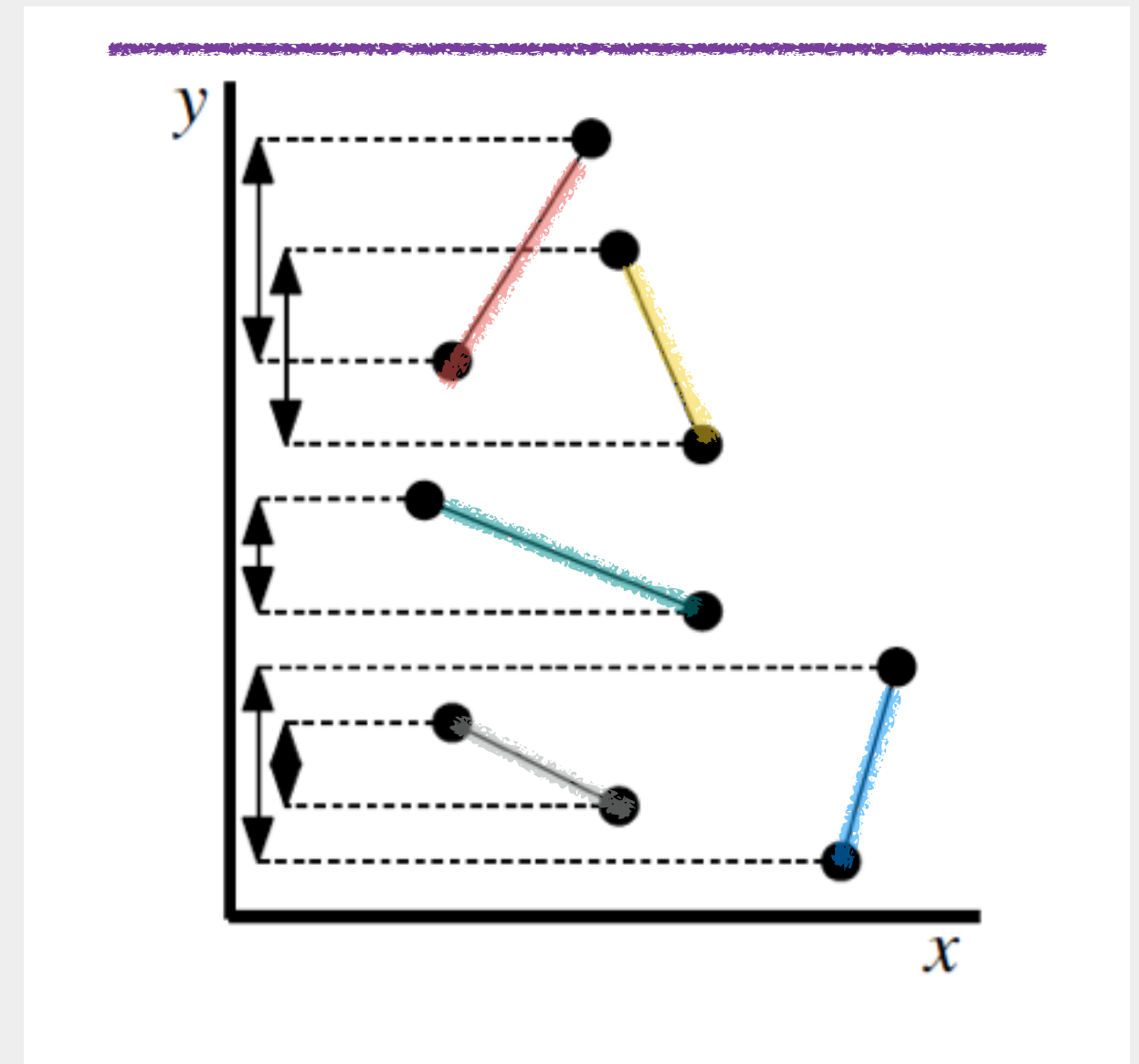
► **puntos evento**

► en este caso, puntos extremo de cada segmento.

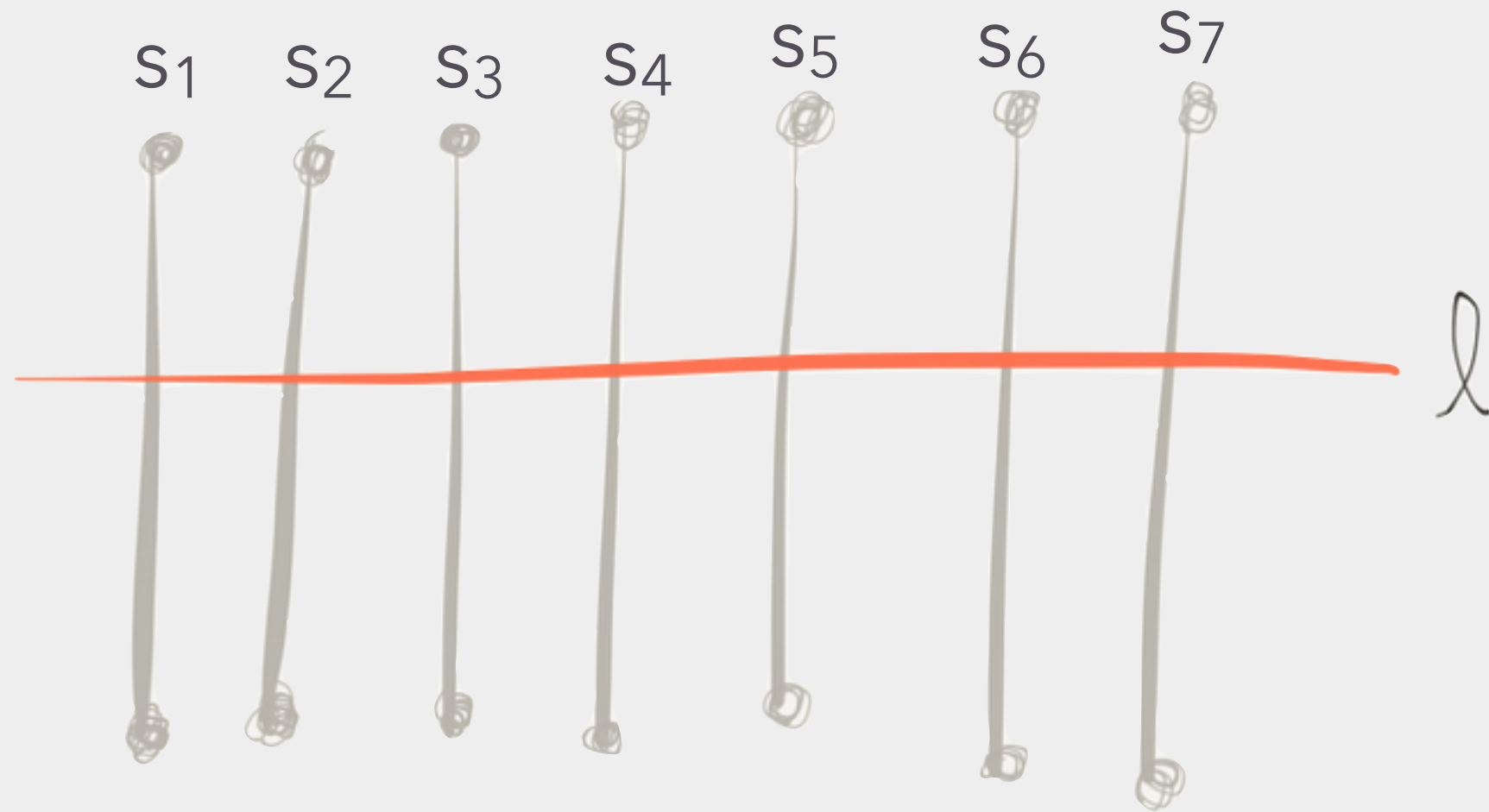
¿Qué pasa en el punto evento?

► Solo probar segmentos que están simultáneamente en el estado de la línea de barrido.

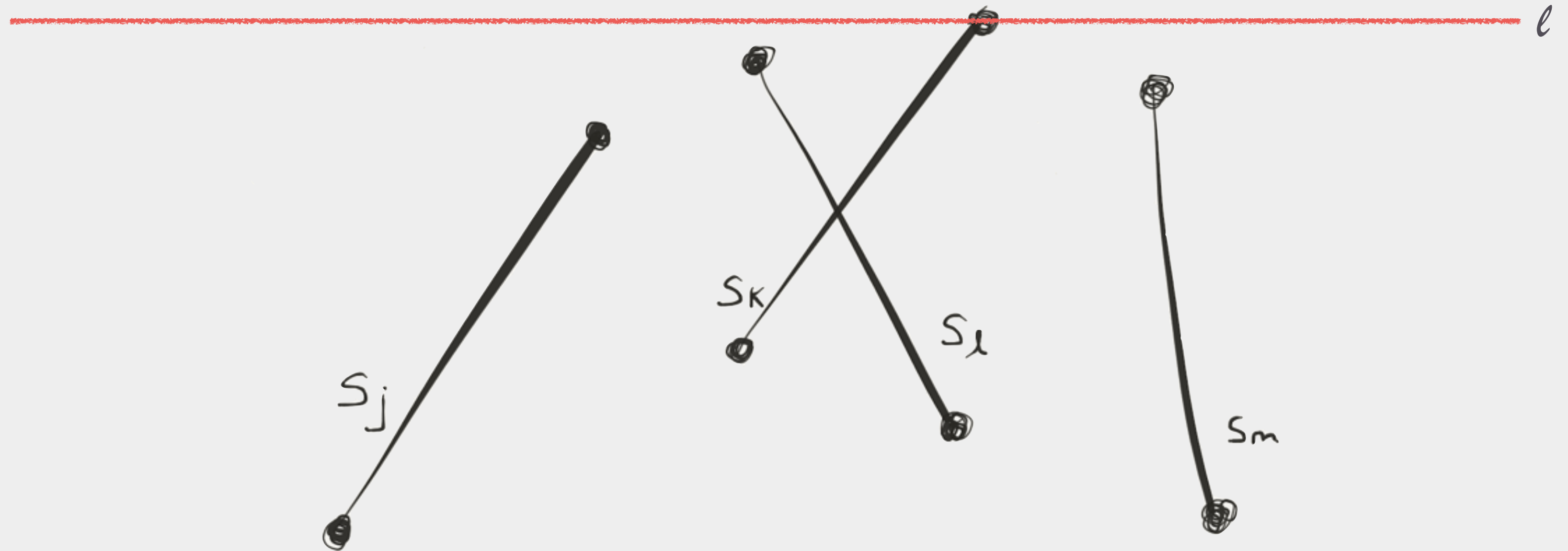
sweepline - l



¿Suficiente para que el algoritmo sea sensible al número de intersecciones?



¿Cuándo cambia la adyacencia de dos segmentos en el estado de la línea de barrido?

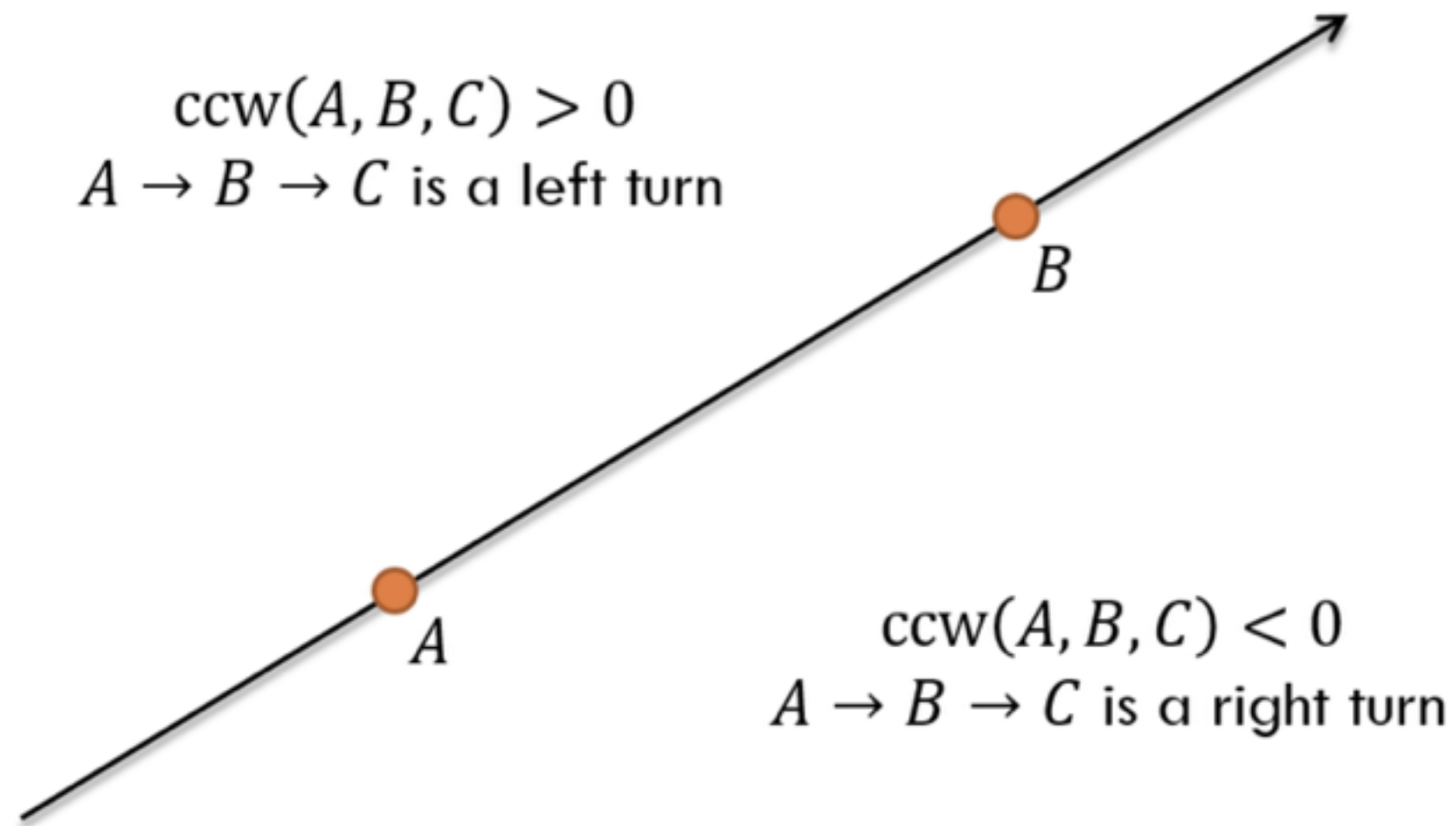


Nuevo punto evento: ¡Puntos de intersección!

¿ y cómo sabemos si dos segmentos de recta intersecan o no?

# Producto Cruz

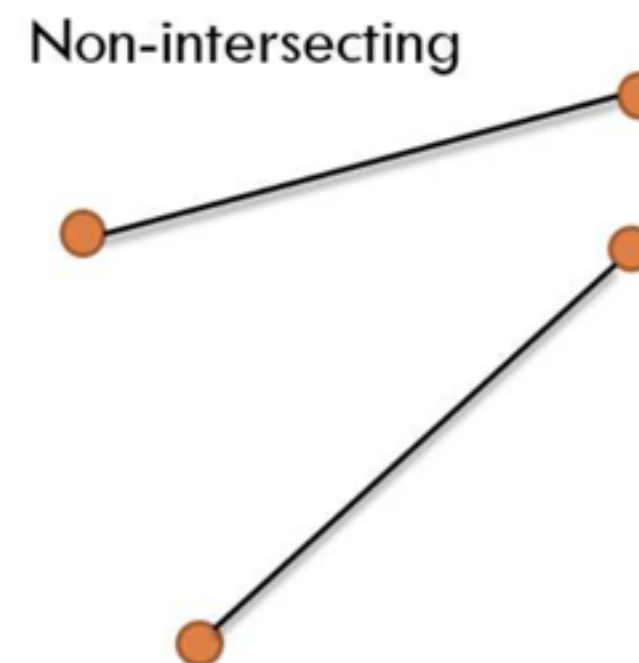
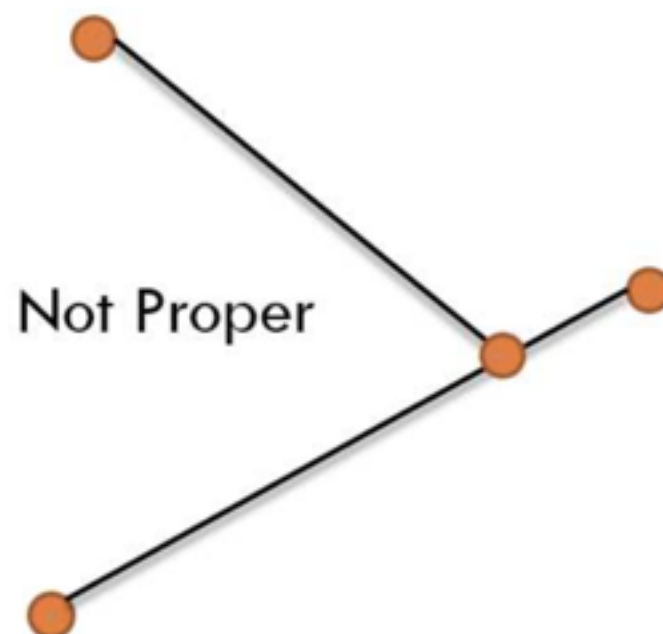
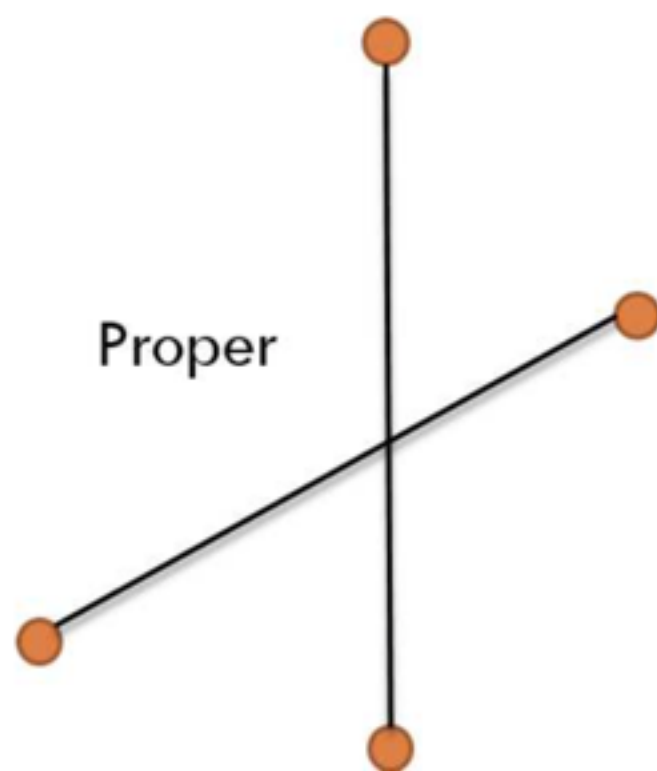
Define  $\text{ccw}(A, B, C) = (B - A) \times (C - A) =$   
 $(b_x - a_x)(c_y - a_y) - (b_y - a_y)(c_x - a_x)$





# Intersección de dos segmentos de recta

- ▶ Given two segments  $AB$  and  $CD$
- ▶ Want to determine if they intersect properly: two segments meet at a single point that are strictly inside both segments



# Intersección de dos segmentos de recta

- ▶ Assume that the segments intersect
  - From  $A$ 's point of view, looking straight to  $B$ ,  $C$  and  $D$  must lie on different sides
  - Holds true for the other segment as well
- ▶ The intersection exists and is proper if:
  - $ccw(A, B, C) \times ccw(A, B, D) < 0$
  - *and*  $ccw(C, D, A) \times ccw(C, D, B) < 0$

# Intersección de dos segmentos de recta

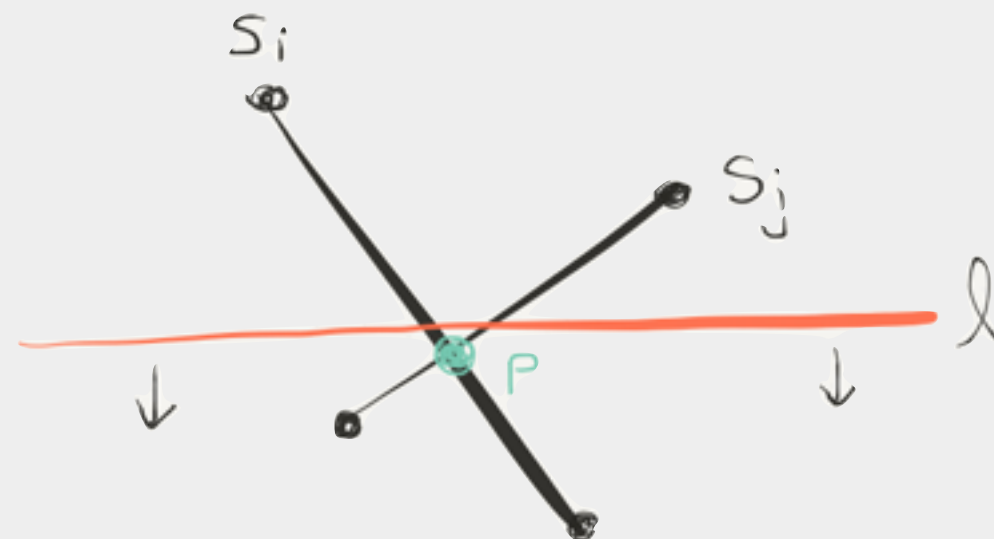
- ▶ We need more special cases to consider!
- ▶ *e.g.*, If  $ccw(A, B, C)$ ,  $ccw(A, B, D)$ ,  $ccw(C, D, A)$ ,  $ccw(C, D, B)$  are all zeros, then two segments are collinear
- ▶ Very careful implementation is required

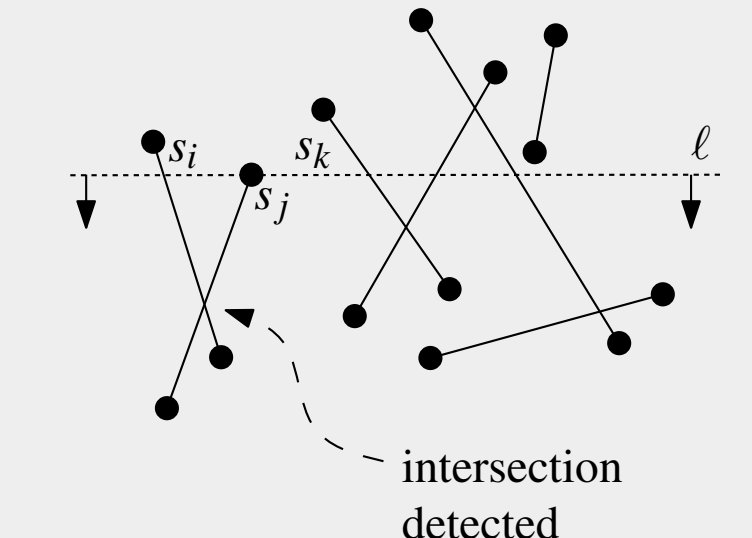
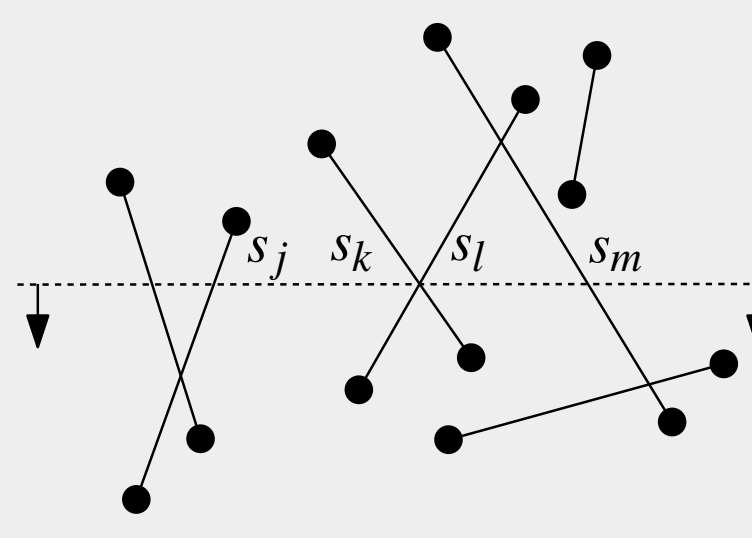
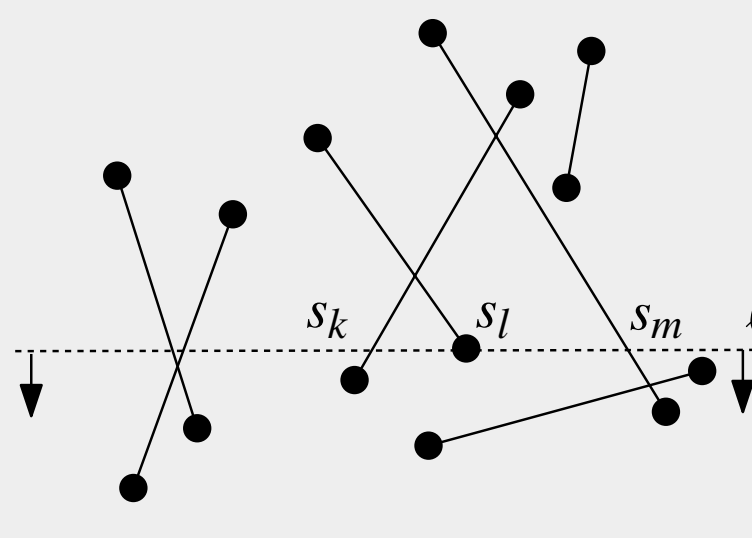
- ▶ ¿Estamos seguros que vamos a encontrar todas las intersecciones?
- ▶ Si dos segmentos  $s_i$  y  $s_j$  intersecan, ¿existe siempre una posición en la línea de barrido donde  $s_i$  y  $s_j$  sean adyacentes?

Evitamos por el momento casos degenerados:



¿Encontramos todas las intersecciones inferiores?



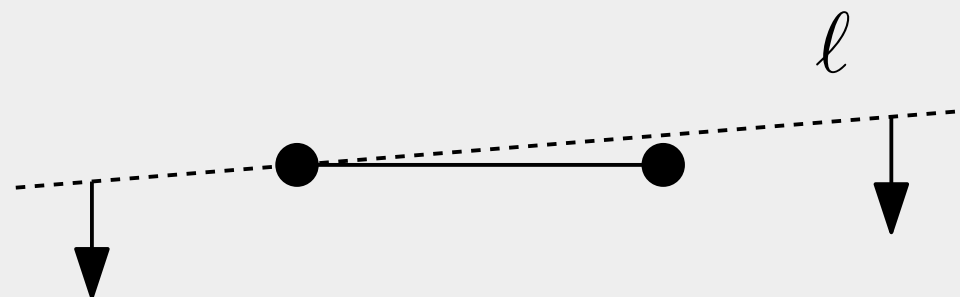
Evento	Acción	Ejemplo
extremo superior alcanzado	probar el segmento contra sus dos vecinos sobre la línea de barrido.	 <p>Diagram illustrating the event where the upper endpoint of a segment is reached. A horizontal sweep line <math>l</math> (dashed line with downward arrows) is positioned at the top endpoint of segment <math>s_i</math>. It intersects segments <math>s_i</math> and <math>s_j</math>. A dashed line indicates an intersection detected between <math>s_i</math> and <math>s_j</math>.</p>
cambio de adyacencia entre segmentos	cada segmento toma un nuevo vecino a lo más contra quién debería ser probado.	 <p>Diagram illustrating the change of adjacency between segments. The sweep line <math>l</math> has moved down and now intersects segments <math>s_j</math>, <math>s_k</math>, <math>s_l</math>, and <math>s_m</math>. The neighbors of <math>s_j</math> have changed.</p>
extremo inferior alcanzado	sus dos vecinos se hacen adyacentes y deben ser probados.	 <p>Diagram illustrating the event where the lower endpoint of a segment is reached. The sweep line <math>l</math> has moved down and now intersects segments <math>s_k</math>, <math>s_l</math>, and <math>s_m</math>. The neighbors of <math>s_l</math> are now adjacent.</p>

¿Qué estructuras de datos necesitamos para implementar este algoritmo?

▶ **cola de eventos  $Q$ .**

▶ **Operaciones:**

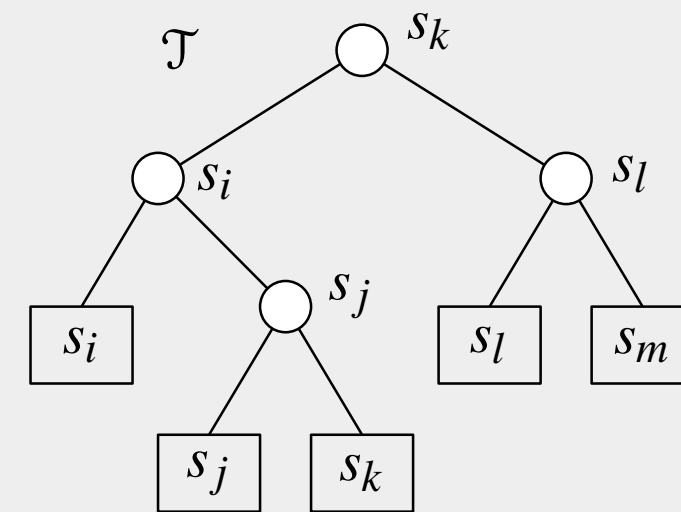
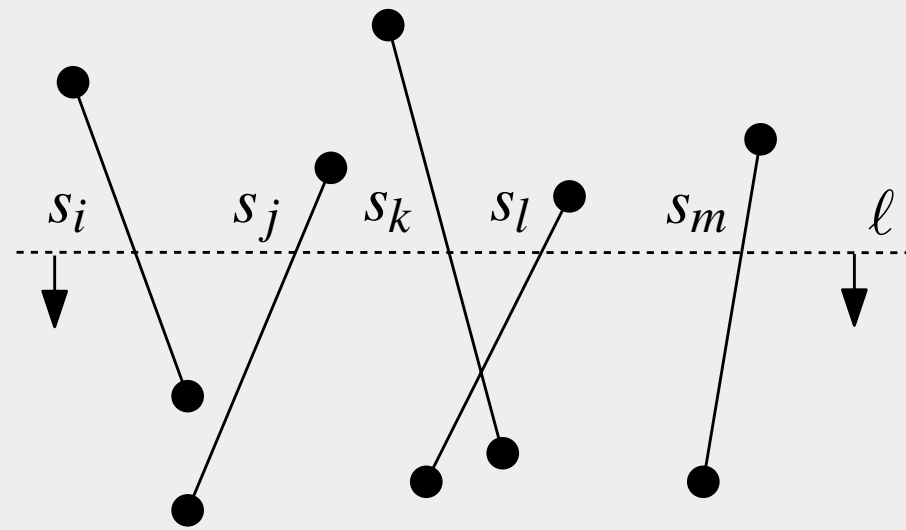
- ▶ **Eliminar el próximo evento** (el más alto abajo de la línea de barrido) en  $Q$  y regresar el punto evento.
- ▶ Si dos puntos evento tienen la misma coordenada  $y$ , regresar aquel con la coordenada  $x$  más pequeña.
- ▶ En una línea horizontal el punto más a la izquierda será el evento superior.



- ▶ Insertar un evento.
- ▶ Verificar si un segmento está dentro de  $Q$ .
- ▶ Definir un orden  $\prec$  en los puntos evento.
- ▶ Si  $p$  y  $q$  son puntos evento,  $p \prec q$  si y solo si  $p_y > q_y$  o si  $p_y = q_y, p_x < q_x$ .
- ▶ Guardar los puntos evento en un **árbol binario balanceado**, ordenado de acuerdo a  $\prec$
- ▶ Con cada punto evento  $p$  en  $Q$  se deben **almacenar también los segmentos que empiecen en  $p$** .
- ▶ Ambas operaciones toman  $O(\log m)$  donde  $m$  es el **número de eventos en  $Q$** .
- ▶ No se utiliza un montículo porque hay que verificar si un evento ya está presente en  $Q$ .

- Se debe mantener un **estado del algoritmo**,: una secuencia de segmentos ordenados que intersequen la línea de barrido.
- La **estructura del estado T**, se usa **para acceder a los vecinos de un segmento** dado  $s$ , de tal manera que se pueda probar si interseca con  $s$ .
- La estructura debe ser **dinámica** ya que los segmentos empiezan o terminan de intersectar a la línea de barrido (se añaden y eliminan).
- Como hay un orden bien definido en los segmentos dentro de la estructura de estado, se puede usar un **árbol binario de búsqueda balanceado**.
- Los segmentos que intersecan la línea de barrido se encuentran en el **mismo orden en las hojas del árbol binario de búsqueda**.





- El orden de izquierda a derecha sobre la línea de barrido corresponde al orden de izquierda a derecha de las hojas de  $\mathcal{T}$ .
- Los nodos internos mantienen la información necesaria para guiar la búsqueda hacia abajo.
- En cada nodo interno, almacenamos el segmento más a la derecha en el subárbol izquierdo.

- Supongamos que buscamos en  $T$  al segmento **inmediatamente a la izquierda** de un punto  $p$  sobre la línea de barrido.
- En cada nodo interno  $v$ , probamos si  $p$  se encuentra a la izquierda o a la derecha del segmento almacenado en  $v$ .
- Dependiendo de estas prueba bajamos hacia el **subárbol izquierdo o al derecho hasta llegar a una hoja**.
- El segmento buscado estará almacenado en esta hoja o en la inmediata izquierda.
- Cada actualización y búsqueda de vecino toma  $O(\log n)$ .
- Las únicas estructuras que necesitamos entonces son:
  - **La cola de eventos  $Q$ .**
  - **El estado de la línea de barrido  $T$ .**

**Algorithm** FINDINTERSECTIONS( $S$ )

*Input.* A set  $S$  of line segments in the plane.

*Output.* The set of intersection points among the segments in  $S$ , with for each intersection point the segments that contain it.

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

### HANDLEEVENTPOINT( $p$ )

1. Let  $U(p)$  be the set of segments whose upper endpoint is  $p$ ; these segments are stored with the event point  $p$ . (For horizontal segments, the upper endpoint is by definition the left endpoint.)
2. Find all segments stored in  $\mathcal{T}$  that contain  $p$ ; they are adjacent in  $\mathcal{T}$ . Let  $L(p)$  denote the subset of segments found whose lower endpoint is  $p$ , and let  $C(p)$  denote the subset of segments found that contain  $p$  in their interior.
3. **if**  $L(p) \cup U(p) \cup C(p)$  contains more than one segment
4.     **then** Report  $p$  as an intersection, together with  $L(p)$ ,  $U(p)$ , and  $C(p)$ .
5. Delete the segments in  $L(p) \cup C(p)$  from  $\mathcal{T}$ .
6. Insert the segments in  $U(p) \cup C(p)$  into  $\mathcal{T}$ . The order of the segments in  $\mathcal{T}$  should correspond to the order in which they are intersected by a sweep line just below  $p$ . If there is a horizontal segment, it comes last among all segments containing  $p$ .
7. (\* Deleting and re-inserting the segments of  $C(p)$  reverses their order. \*)
8. **if**  $U(p) \cup C(p) = \emptyset$
9.     **then** Let  $s_l$  and  $s_r$  be the left and right neighbors of  $p$  in  $\mathcal{T}$ .
10.         FINDNEWEVENT( $s_l, s_r, p$ )
11.     **else** Let  $s'$  be the leftmost segment of  $U(p) \cup C(p)$  in  $\mathcal{T}$ .
12.         Let  $s_l$  be the left neighbor of  $s'$  in  $\mathcal{T}$ .
13.         FINDNEWEVENT( $s_l, s', p$ )
14.         Let  $s''$  be the rightmost segment of  $U(p) \cup C(p)$  in  $\mathcal{T}$ .
15.         Let  $s_r$  be the right neighbor of  $s''$  in  $\mathcal{T}$ .
16.         FINDNEWEVENT( $s'', s_r, p$ )

FINDNEWEVENT( $s_l, s_r, p$ )

1. **if**  $s_l$  and  $s_r$  intersect below the sweep line, or on it and to the right of the current event point  $p$ , and the intersection is not yet present as an event in  $\mathcal{Q}$
2. **then** Insert the intersection point as an event into  $\mathcal{Q}$ .

