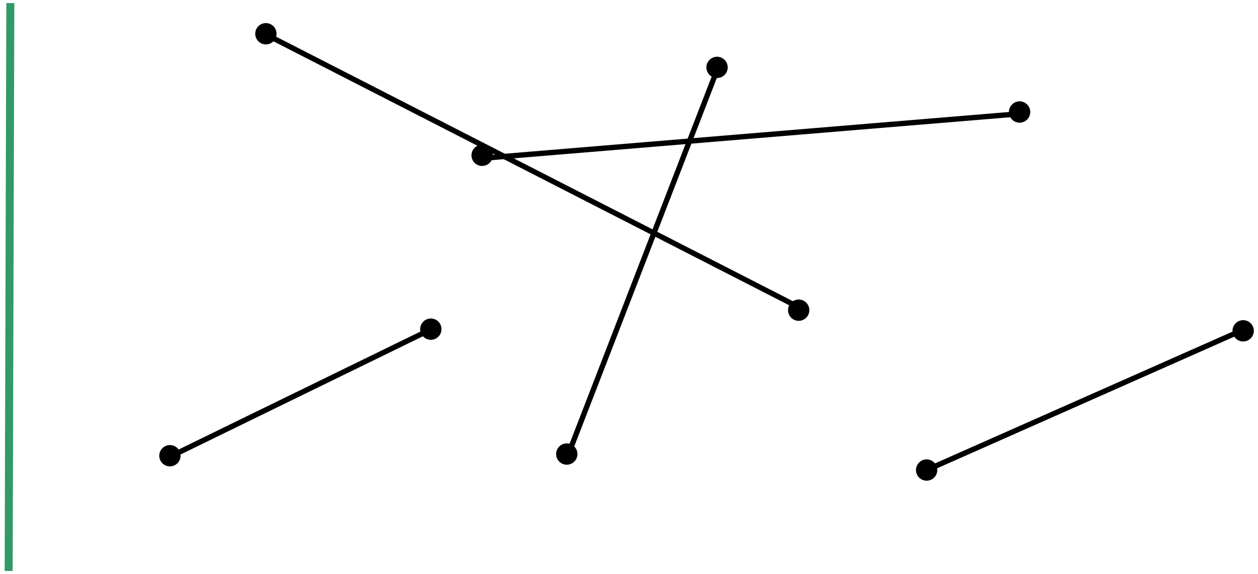


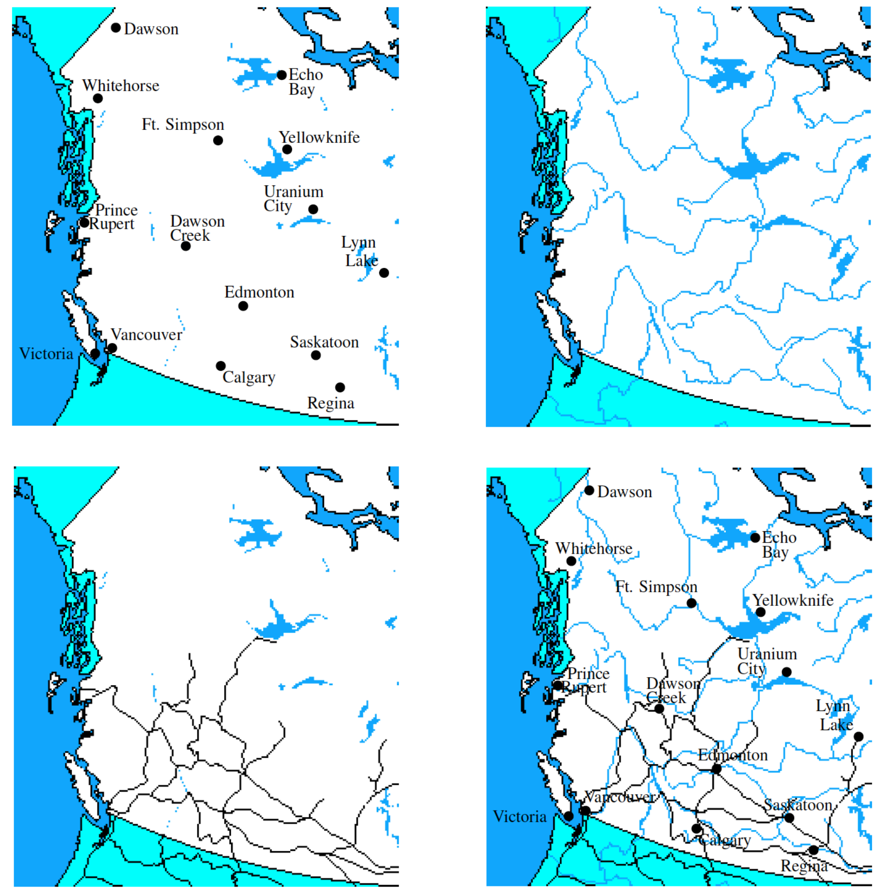
Line Segment Intersection

Michael Goodrich
Univ. of California, Irvine

Some slides from CMPS 3130/6130 Computational Geometry, Spring 2015, by Carola Wenk

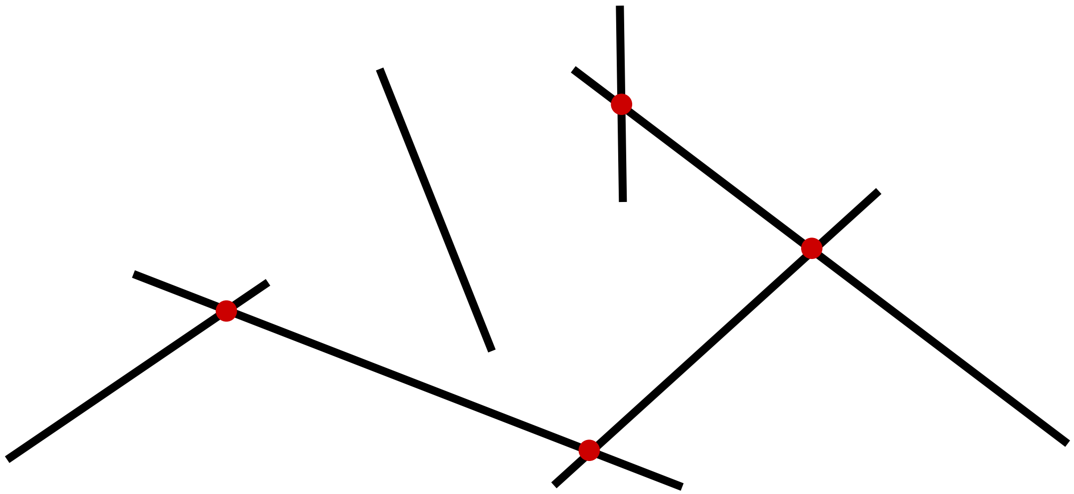
Geometric Intersections

- Important problem in Computational Geometry
- Solid modeling: Build shapes by applying set operations (intersection, union).
- Robotics: Collision detection and avoidance
- Geographic information systems: Overlay two subdivisions (e.g., road network and river network)



Line Segment Intersection

- Input: A set $S=\{s_1, \ldots, s_n\}$ of (closed) line segments in $\mathbf{R}^2$
- Output: All **intersection points** between segments in S

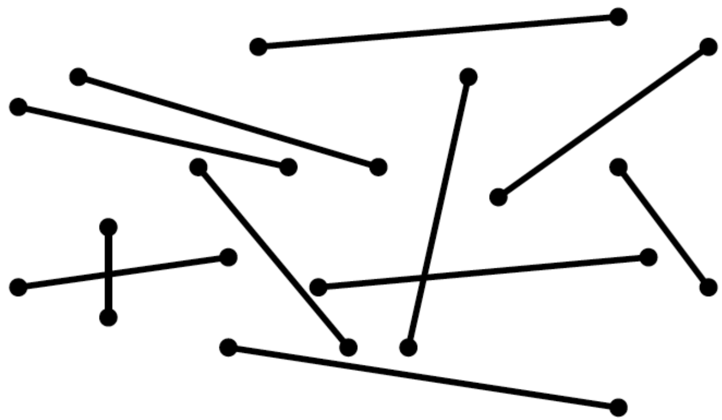


Line Segment Intersection

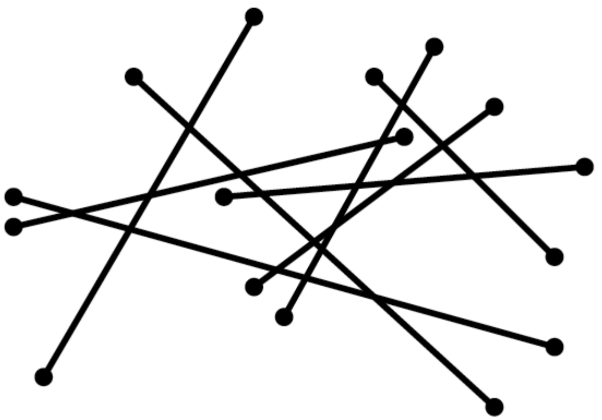
- n line segments can intersect as few as 0 and as many as $\binom{n}{2} = O(n^2)$ times
- Simple algorithm: Try out all pairs of line segments
  $\rightarrow$ Takes $O(n^2)$ time
  $\rightarrow$ Is optimal in worst case
- Challenge: Develop an **output-sensitive algorithm**
  - Runtime depends on size k of the output
  - Here: $0 \le k \le n^2$
  - Our algorithm will have runtime: $O((n+k) \log n)$
    - This algorithm is due to Bentley and Ottmann
  - Best possible runtime: $O(n \log n + k)$
    $\rightarrow$ $O(n^2)$ in worst case, but better in general

Output size for intersections

- k = # of intersections



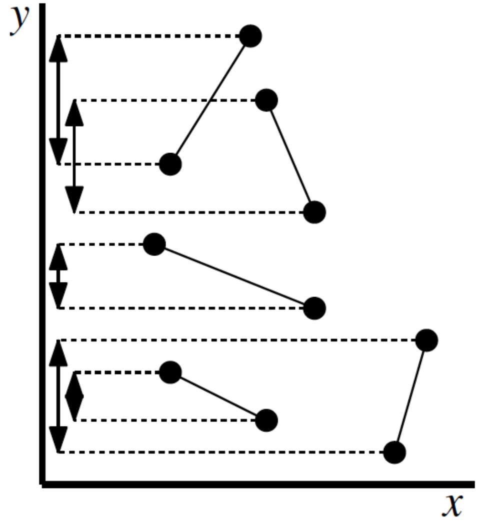
k = 2



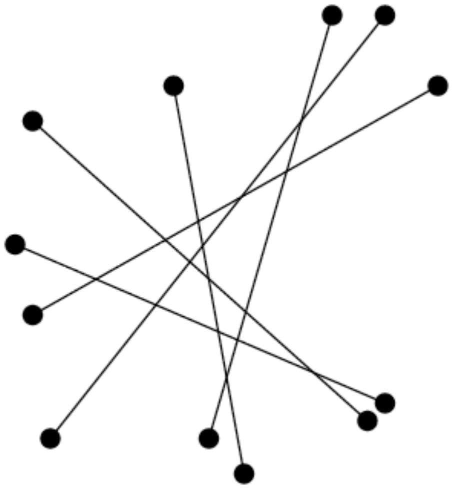
k = 18

Output size for intersections

- Range of k can go from 0 to n choose 2, i.e., $n(n-1)/2$, which is $O(n^2)$.



k = 0



k = 6(5)/2 = 15

Complexity

- Why is runtime $O(n \log n + k)$ optimal?
- The **element uniqueness** problem requires $\Omega(n \log n)$ time in algebraic decision tree model of computation (Ben-Or ’83)
- **Element uniqueness**: Given n real numbers, are all of them distinct?
- Solve **element uniqueness** using **line segment intersection**:
  - Take n numbers, convert into vertical line segments. There is an intersection iff there are duplicate numbers.
  - If we could solve line segment intersection in $o(n \log n)$ time, i.e., strictly faster than $\Theta(n \log n)$, then **element uniqueness** could be solved faster. Contradiction.

Plane sweep algorithm

```
Algorithm Generic_Plane_Sweep:
Initialize sweep line status S at time x=-∞
Store initial events in event queue Q, a priority queue ordered by x-coordinate
while Q ≠ ∅
    // extract next event e:
    e = Q.extractMin();
    // handle event:
    Update sweep line status
    Discover new upcoming events and insert them into Q
```

- **Cleanliness property:**
  - All intersections to the left of sweep line l have been reported
- **Sweep line status:**
  - Store segments that intersect the sweep line l, ordered along the intersection with l .
- **Events:**
  - Points in time when sweep line status changes combinatorially (i.e., the order of segments intersecting l changes)

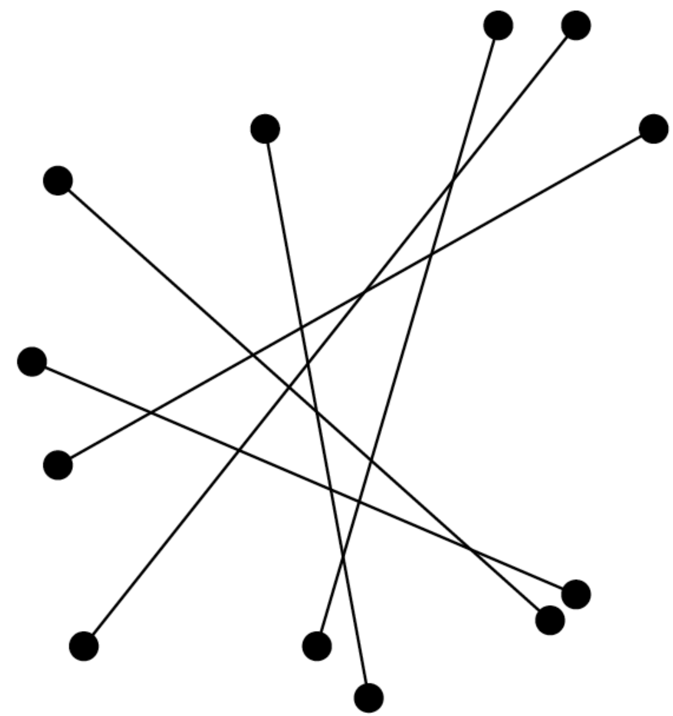
  → Endpoints of segments (insert in beginning)

  → Intersection points (compute on the fly during plane sweep)

General position

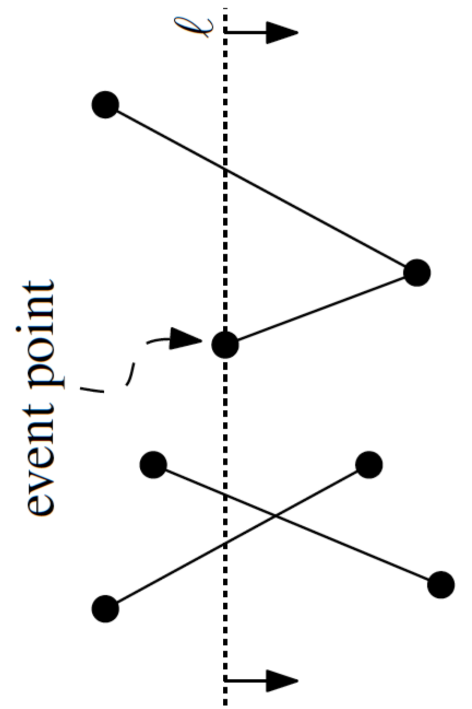
Assume that “nasty” special cases don’t happen:

- No line segment is vertical
- Two segments intersect in at most one point
- No three segments intersect in a common point



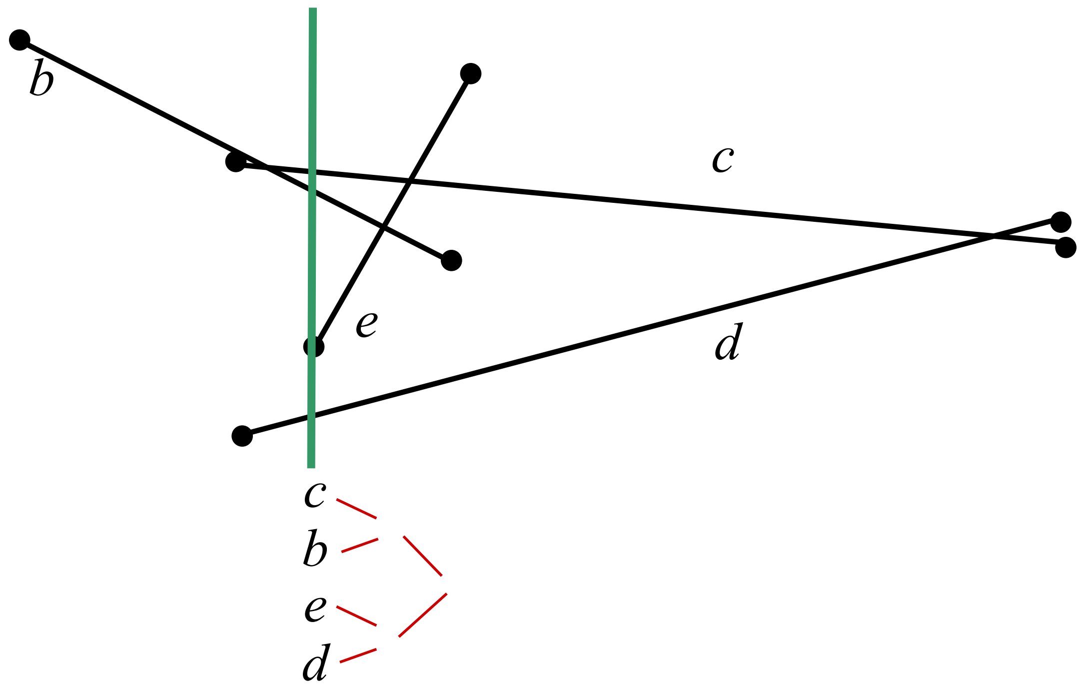
Event Queue

- Need to keep events sorted:
  - Lexicographic order (first by x-coordinate, and if two events have same x-coordinate then by y-coordinate)
- Need to be able to remove next point, and insert new points in O(log n) time
  - Use a balanced binary search tree (e.g., a WAVL tree)
- The de Berg book sweeps top to bottom, but I like to sweep left-to-right.
- So pictures from the book are "sideways"



Sweep Line Status

- Store segments that intersect the sweep line *l*, ordered along the intersection with *l* .
- Need to insert, delete, and find adjacent neighbor in O(log *n*) time
- Use **balanced binary search** tree, storing the order in which segments intersect *l* in leaves

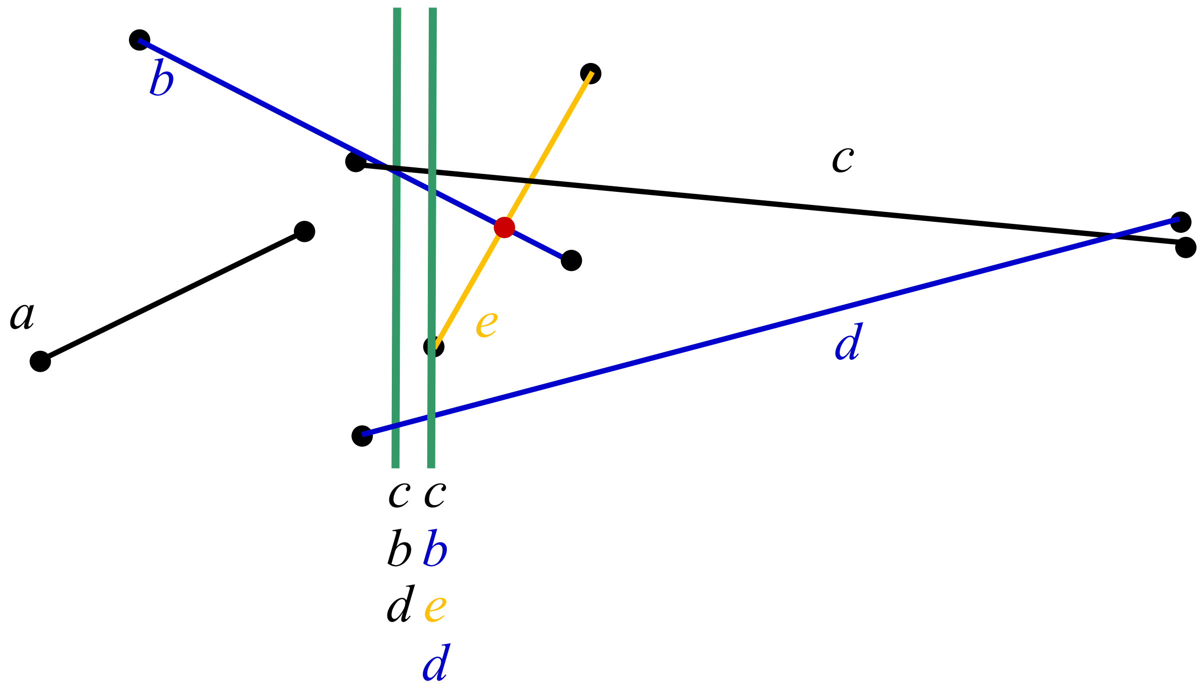


Event Queue

- The events in the event queue (sorted by x-coordinates):
  - Every line-segment endpoint (left and right)
  - The intersection point of every pair of line segments that are consecutive in the ordering along the sweep line.

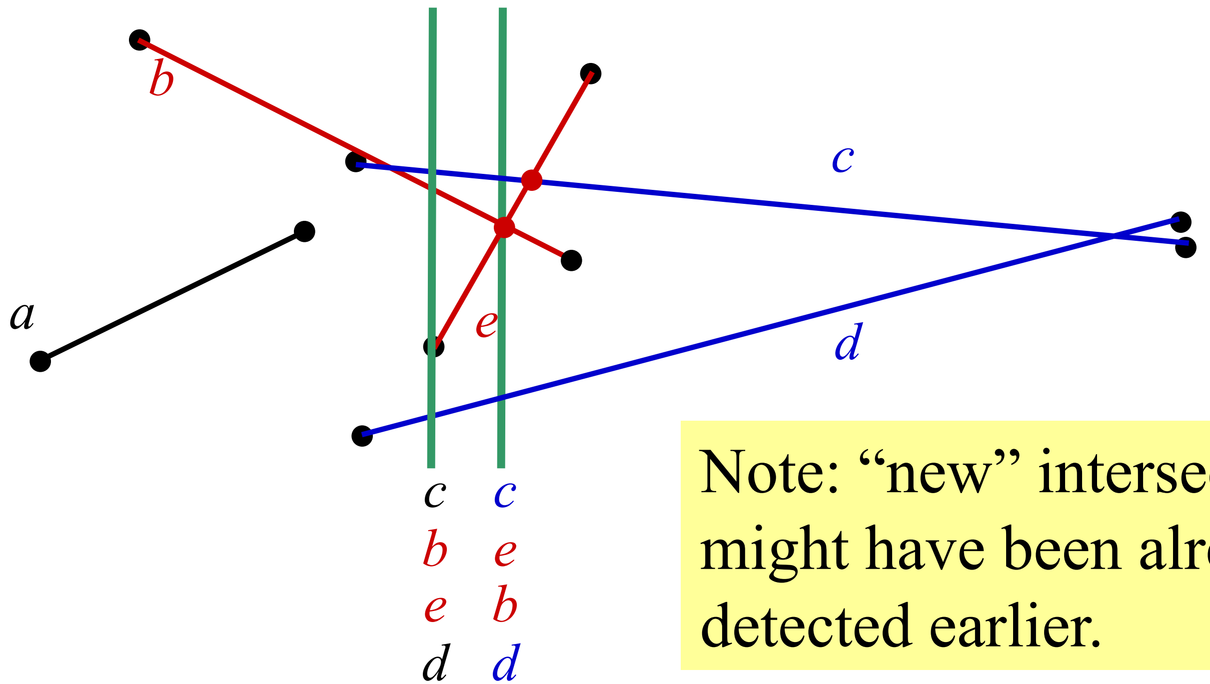
Event Handling

1. Left segment endpoint
   - **Add new segment** to sweep line status
   - Test **adjacent segments** on sweep line *l* for intersection with **new segment**
   - Add **new intersection points** to event queue



Event Handling

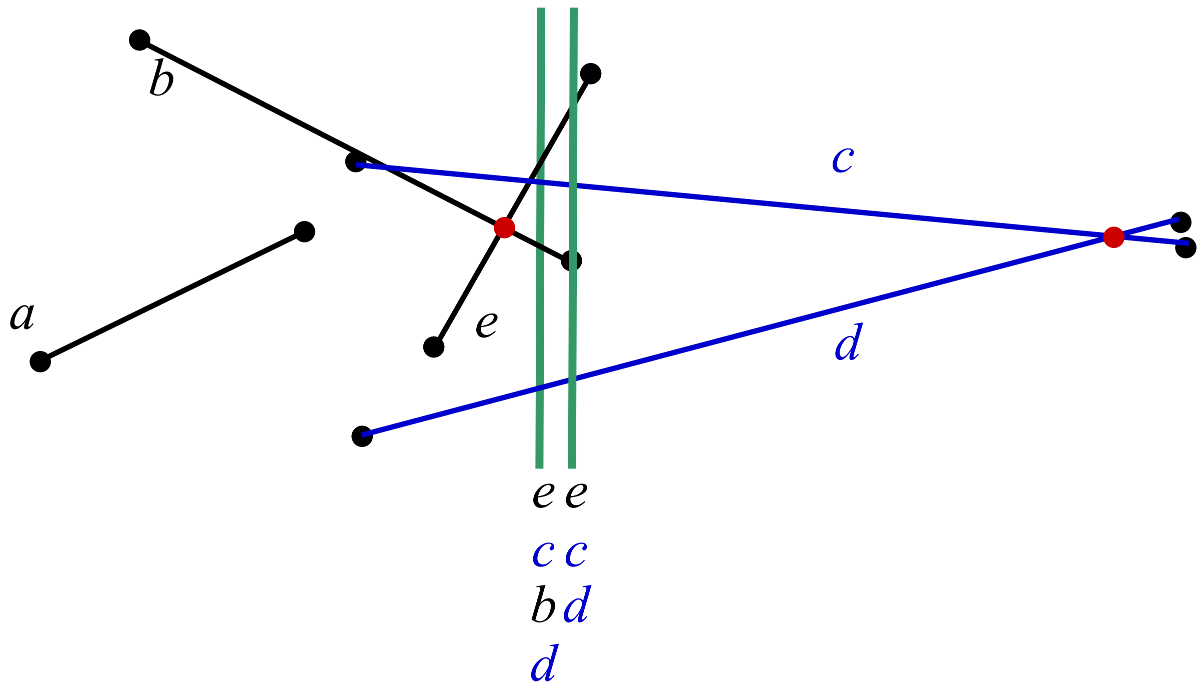
2. Intersection point
   - Report new intersection point
   - Two segments **change order** along *l*
     → Test new adjacent segments for new intersection points (to insert into event queue)



Note: “new” intersection might have been already detected earlier.

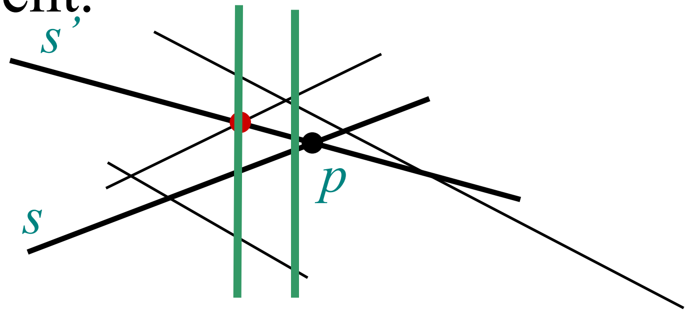
Event Handling

3. Right segment endpoint
   – Delete segment from sweep line status
   – **Two segments become adjacent**. Check for intersection points (to insert in event queue)



Intersection Lemma

- **Lemma:** Let *s*, *s'* be two non-vertical segments whose interiors intersect in a single point *p*. Assume there is no third segment passing through *p*. Then there is an event point to the left of *p* where *s* and *s'* become adjacent (and hence are tested for intersection).
- **Proof:** Consider placement of sweep line infinitesimally left of *p*. *s* and *s'* are adjacent along sweep line. Hence there must have been a **previous event point** where *s* and *s'* become adjacent.



Runtime

- Sweep line status updates: $O(\log n)$
- Event queue operations: $O(\log n)$, as the total number of stored events is $\leq 2n + k$, and each operation takes time
  $O(\log(2n+k)) = O(\log n^2) = O(\log n)$

  $k = O(n^2)$
- There are $O(n+k)$ events. Hence the total runtime is $O((n+k) \log n)$