Computational Geometry: Line Sweeping

Lucas Wood

Outline

- Sweep line algorithms
  - Concepts
  - Applications

Outline

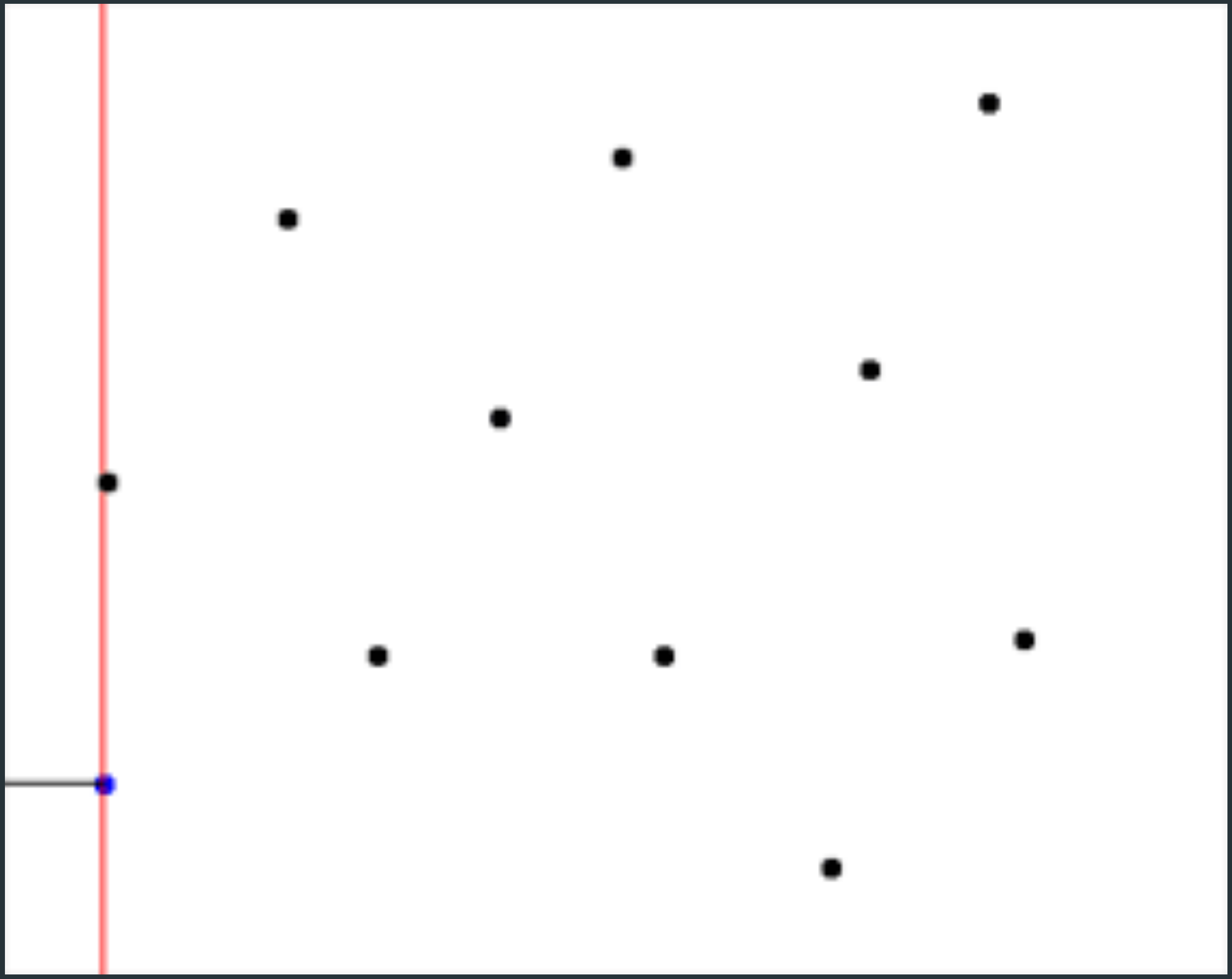
- Applications of Line Sweeping
  - Closest Pair
  - Area of union of rectangles
  - Line segment Intersections
  - Problem

Sweep Line Algorithms

- A technique for solving computational geometry problems with high efficiency.
- Imagine a vertical line being swept across a plane
- As the line encounters “events” on a plane, do some sort of computation.

Sweep Line Algorithms

- Efficiency is highly dependant on the data structures being used.
- Most require the set of events to be stored as a balanced BST.
- Can be generalized to higher dimensions, or sweeping radially.



Closest Pair

Motivating Problem

- Given a set of points, find the pair that is closest.
- Trivial problem in O(n^2)
- Can use line sweep to reduce to O(nlog(n))

Line Sweep

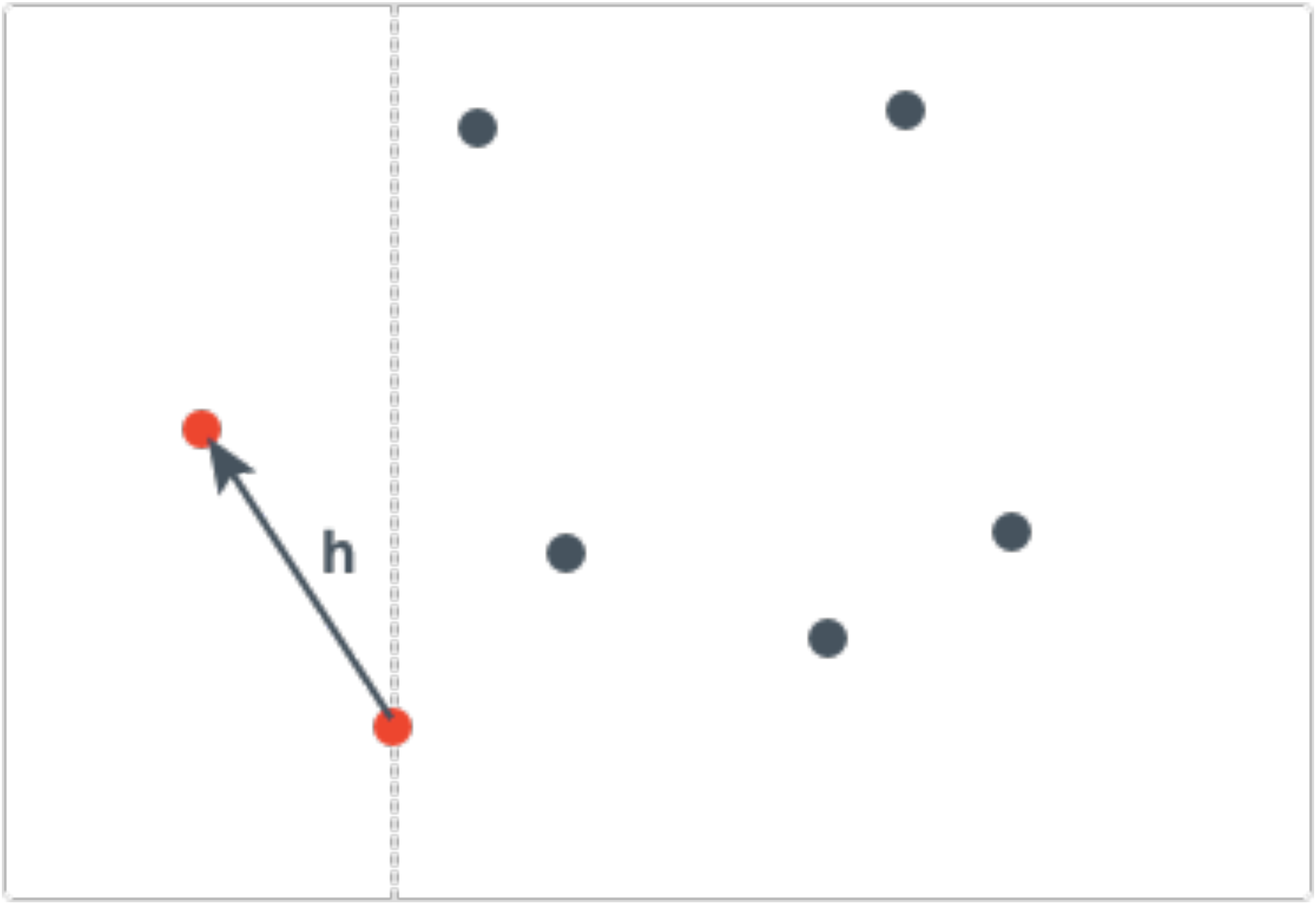
- Consider the points in as events
- Store already visited points in a set ordered by y coordinate.
- We first sort the events by the x direction to indicate our line moving right

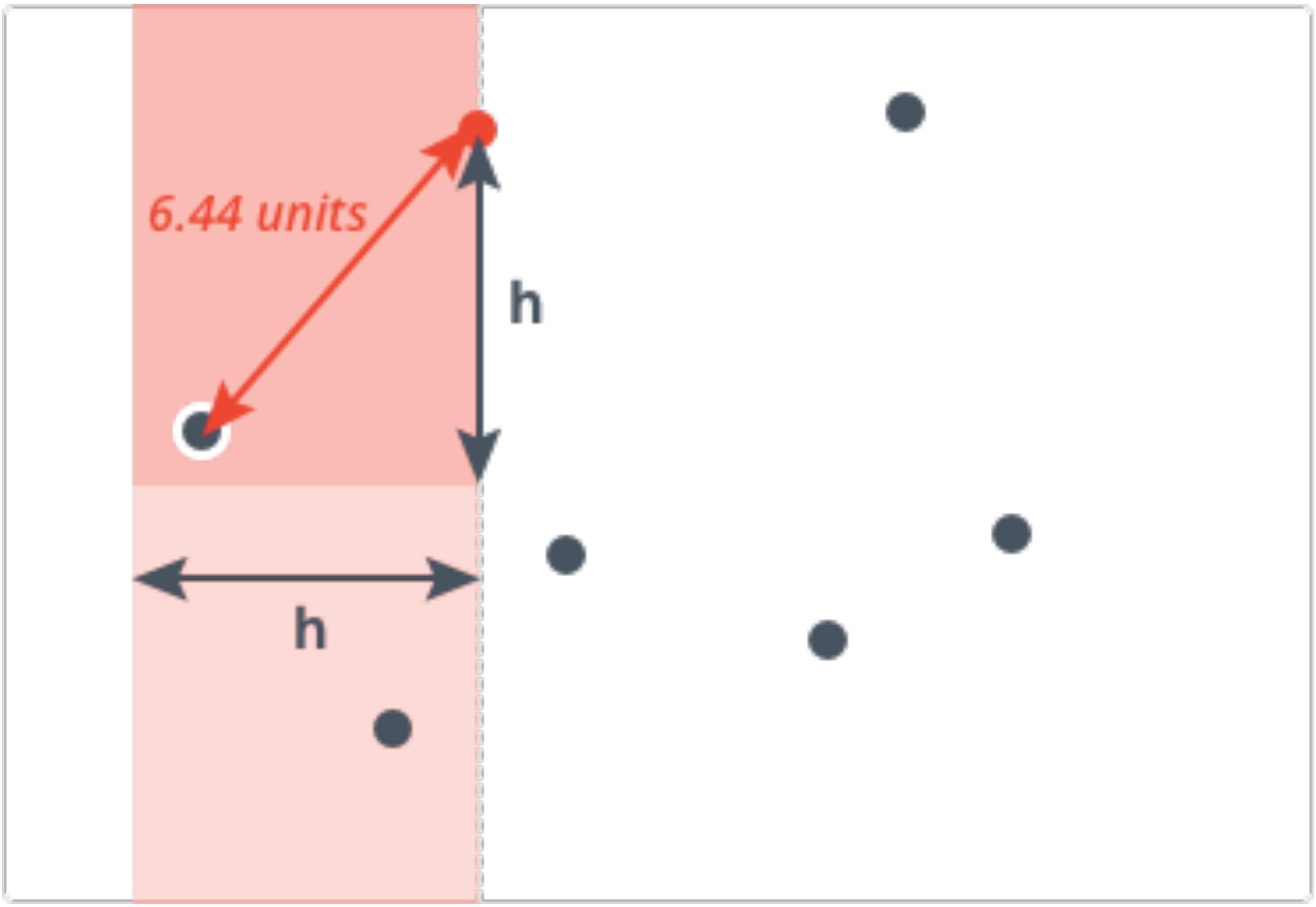
Line Sweep

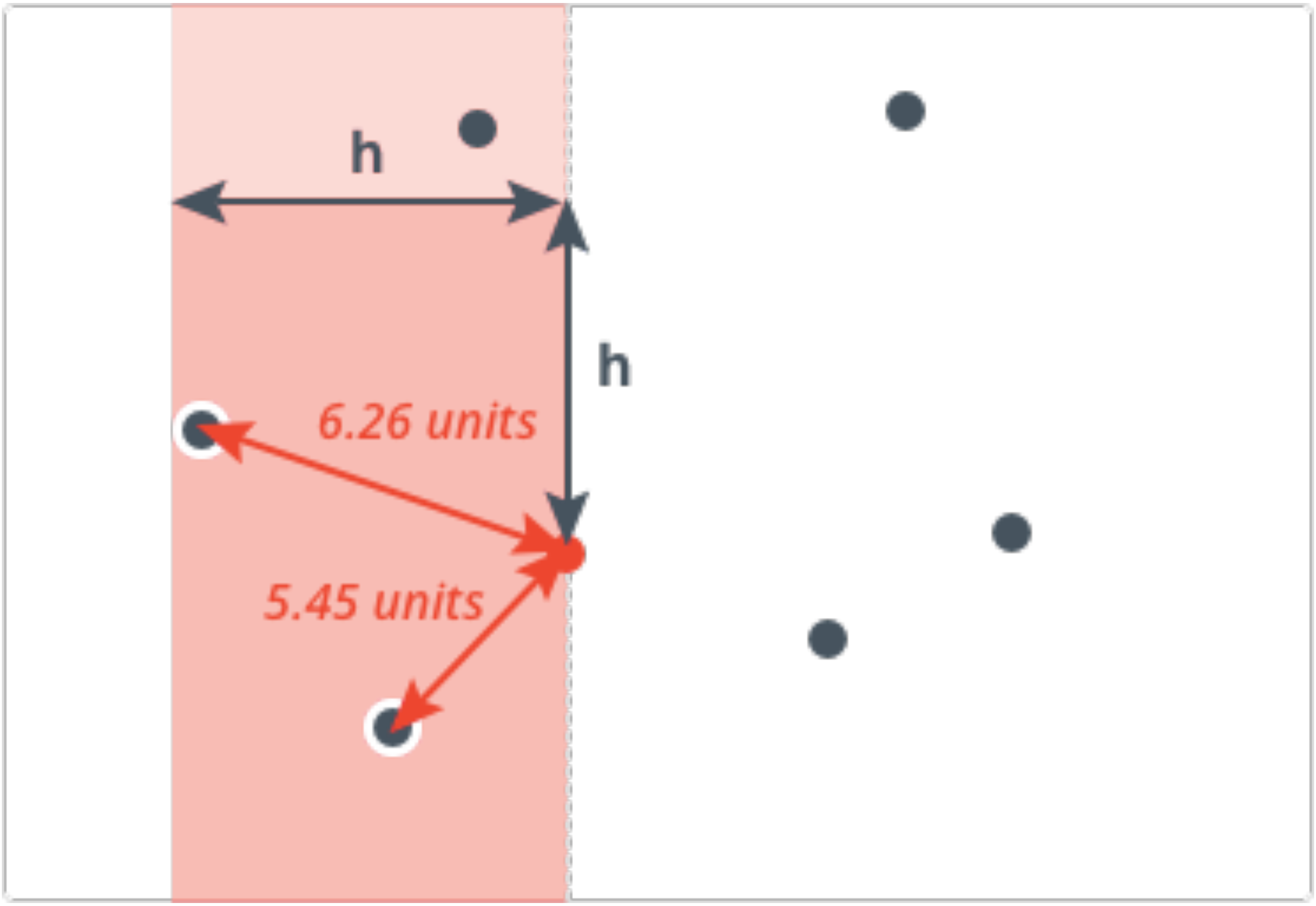
- Suppose we have processed the points from 1 to N-1 and h is the shortest distance so far.

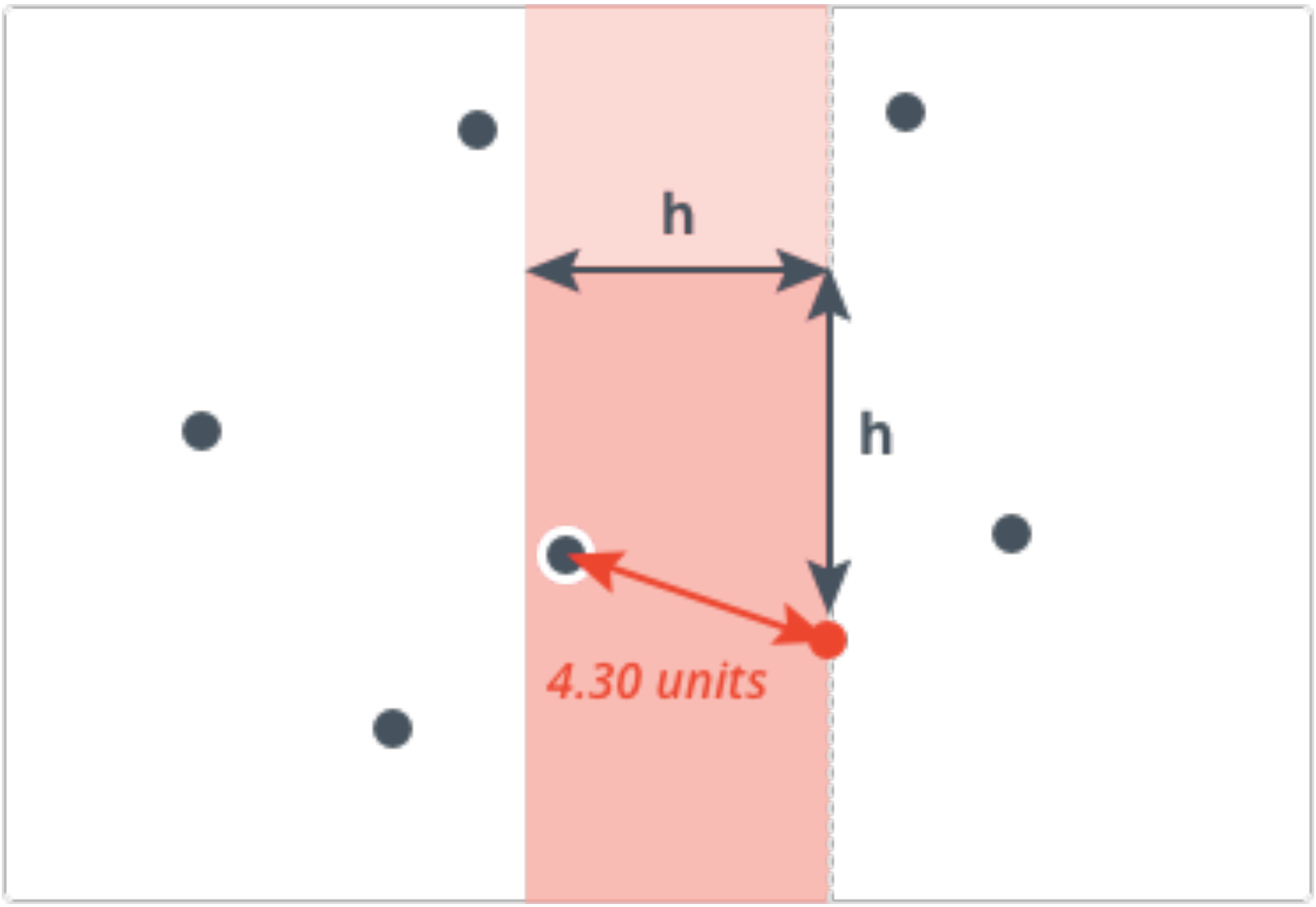
- Only need to consider points h distance from Xn, since points with greater distance are not closer.

- Also don’t need to consider points in front of the line as we will process those with time.


h


6.44 units
h
h


h
h
6.26 units
5.45 units



h
h
4.30 units

Area of Rectangle Unions

Motivating Problem

- Given a set of axis-aligned rectangles, what is the area of their union?
- Brute force approach is very slow
- Line sweep runs in O(n^2) with boolean array
- With clever use of segment trees, reducible to O(n log(n))

Line Sweep

- Consider vertical edges to be events.
- When we encounter an event we check if the edge is a left edge or a right edge and do some action depending on which it is
- Start by sorting events by x coordinate.

Line Sweep

- When a left edge is encountered, insert the rectangle into the set.
- When a right edge is encountered, remove the rectangle from the set.
- So at any instance, the set only contains rectangles which intersect the sweep line

Line Sweep

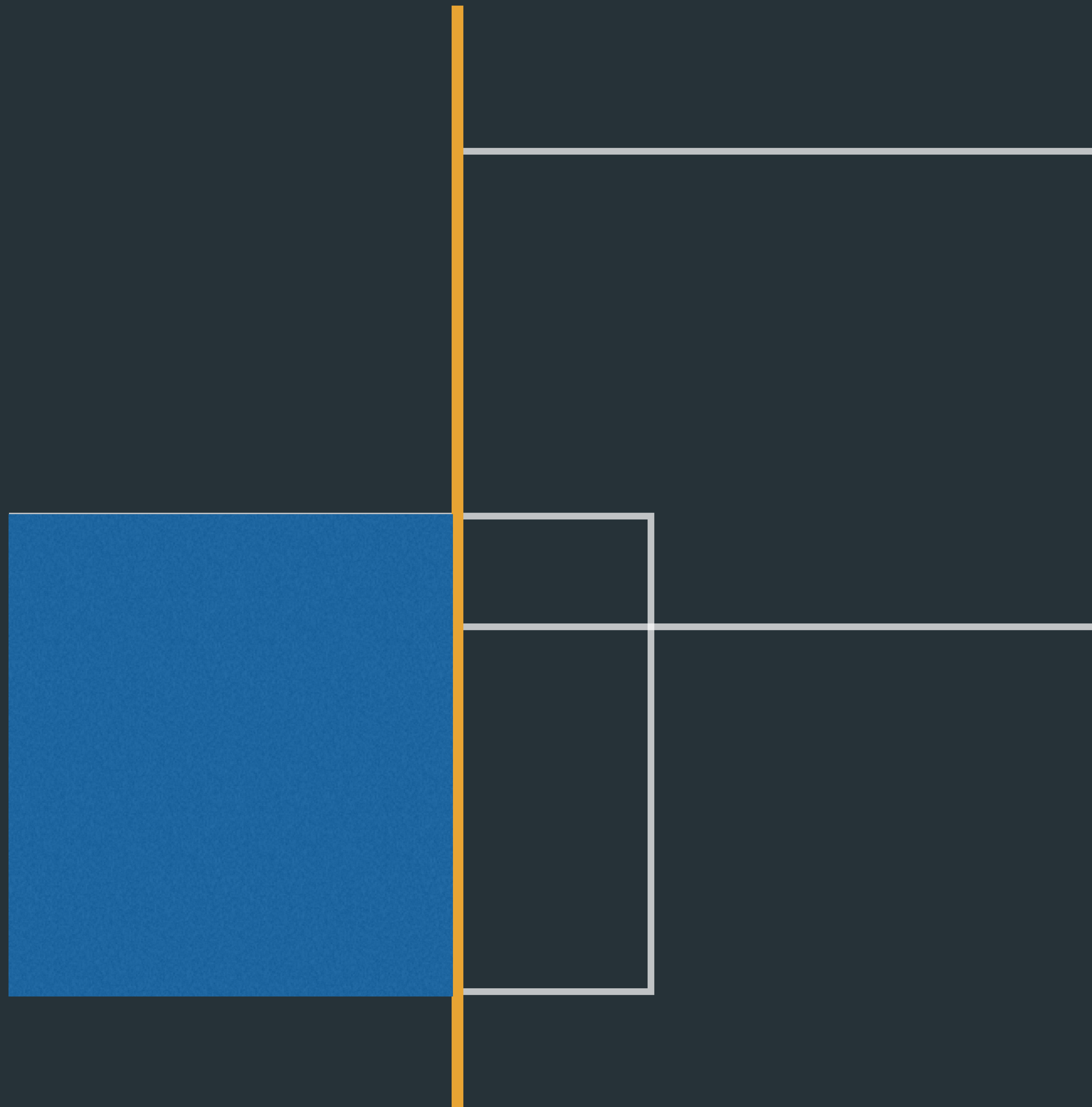
- The area swept at any instance is y*x where y is the length of the sweep line being cut by the rectangles and x is the distance between two events of the sweep line.

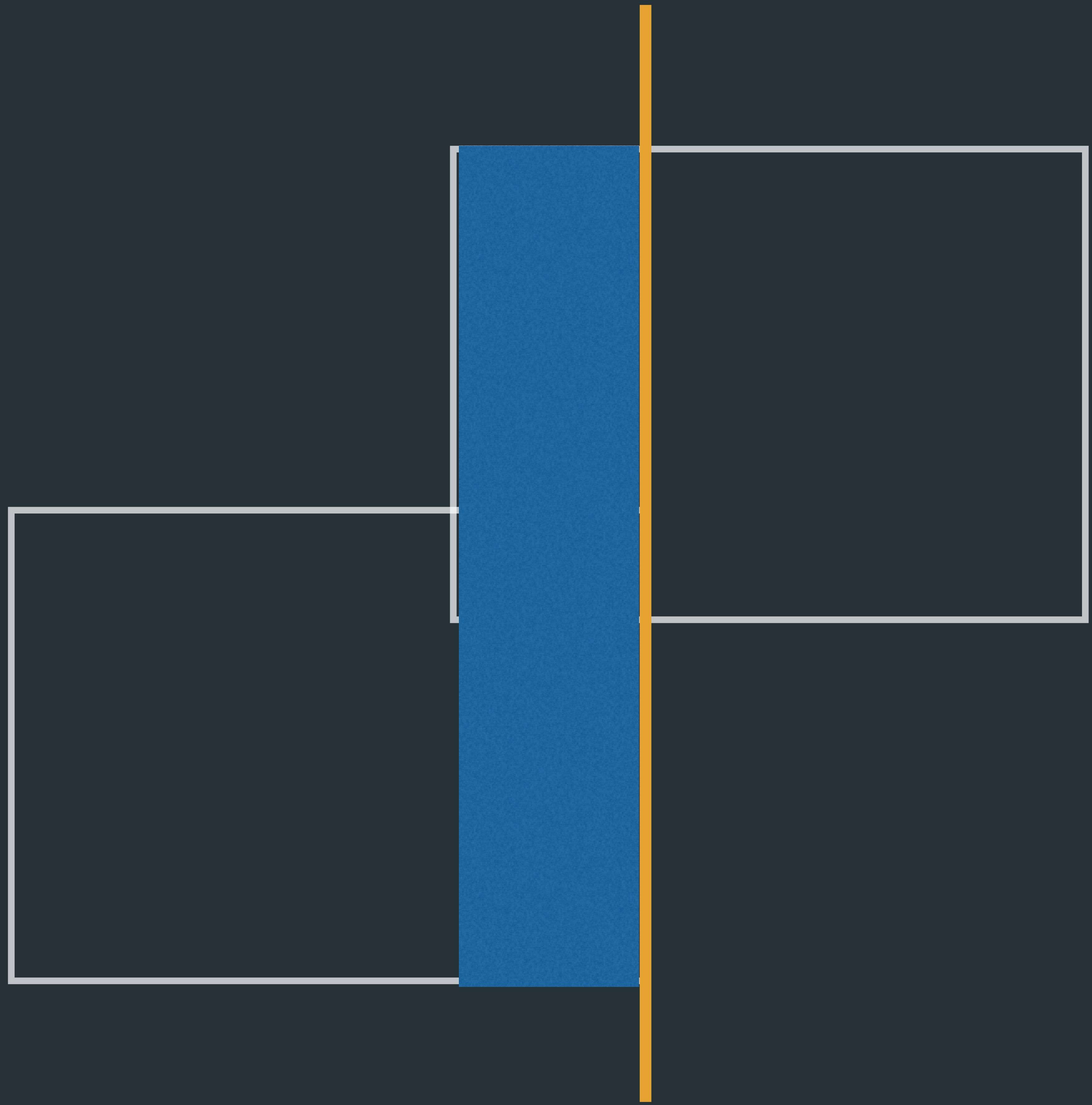
- To find length of sweep cut, run the same algorithm rotated 90 degrees on the events in the active set.

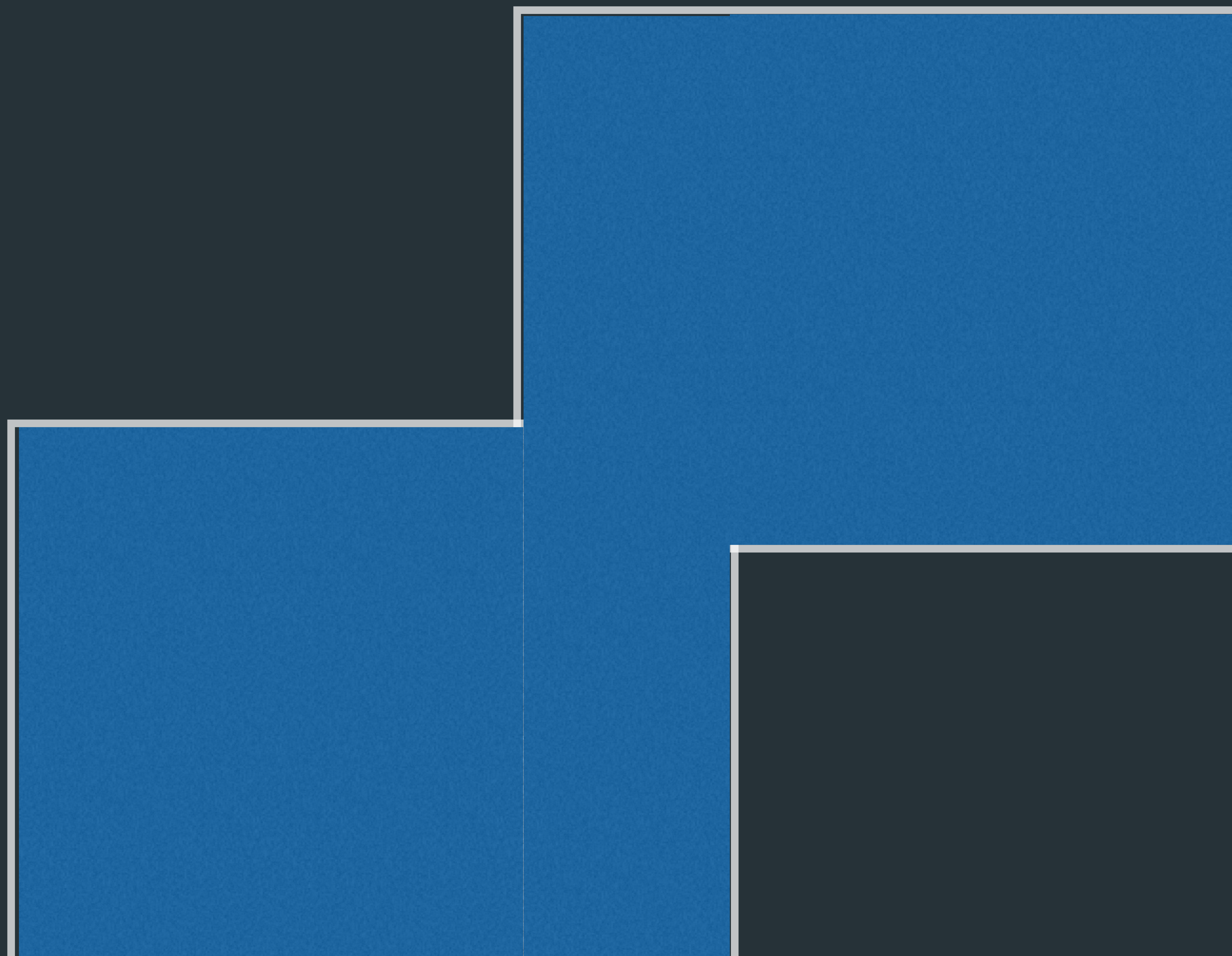
Line Sweep

- Moving top down on the set of events we can use a counter that says how many rectangles overlap at the current point.

- Whenever the counter is nonzero, add this to the length of the sweep cut







Line Segment Intersections

Motivating Problem

- Given a set of line segments find all the intersection points between the segments.
- Compute the intersection of two simple polygons
- Test if a polygon/polyline is simple
- Decompose a polygon into simple pieces

Motivating Problem

- Easy to do in O(n^2) — simply consider all pairs of line segments and test each pair for intersection.
- What if we want to do it faster than that?

Bentley-Ottman Algorithm

- Given a set of lines, computes all k intersections in O((n+k)log n) time and O(n+k) space
- Note: Slower than brute force with large k

Bentley-Ottman Algorithm

- Assumptions to make description easier:
- No two line segment endpoints or crossings have the same x-coordinate
- No line segment endpoint lies upon another line segment
- No three line segments intersect at a single point

Bentley-Ottman Algorithm

- Data Structures required:
- BST of line segments ordered by y-coordinates
- Priority Queue of “events”
    - Each event is associated with line segment endpoints, so prioritize queue by x-coordinate

- 1. Initialize the PQ, Q with an event for each endpoint of the input segments
- 2. Initialize a BST, T of the line segments that cross the sweep line.
- 3. While Q is nonempty, remove an event E from Q. Determine the type of event and process it

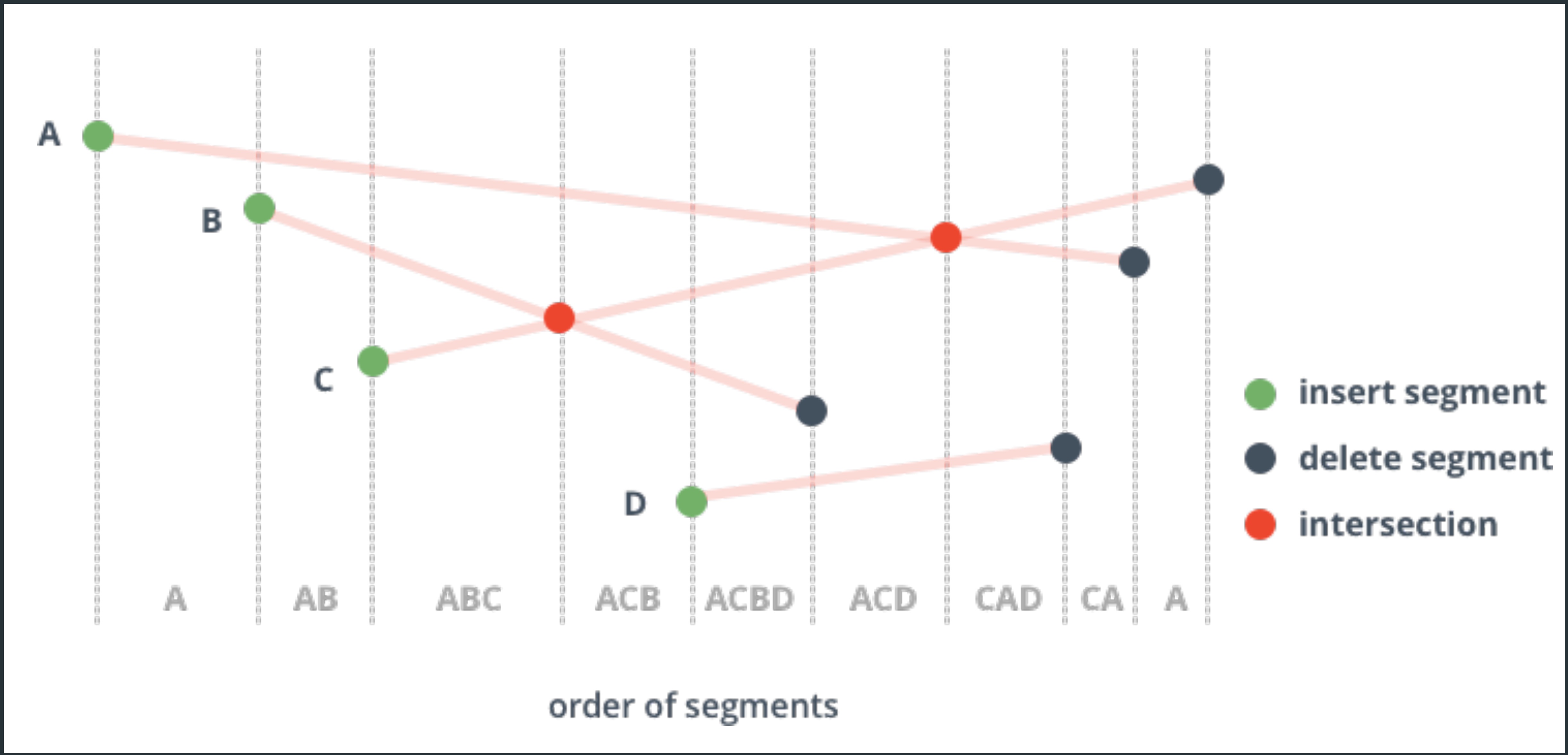
- If p is a left endpoint, insert it's segment s into T. Find r and t immediately below and above s in T. If r or t intersect with s, add these new points to Q.

- If p is a right endpoint, delete it's segment s from T. find r and t immediately below and above s in T. If r and t intersect, and you haven't seen it before, add this point to Q.

- If p is an intersection with segments s, t (s below and t to the left), add it to the output list. Swap the positions of p's segments in T. Find the segments above and below these segments r, u. Remove crossing points between rs and tu from Q and add crossing points of r and t or s and u.

Segment List:

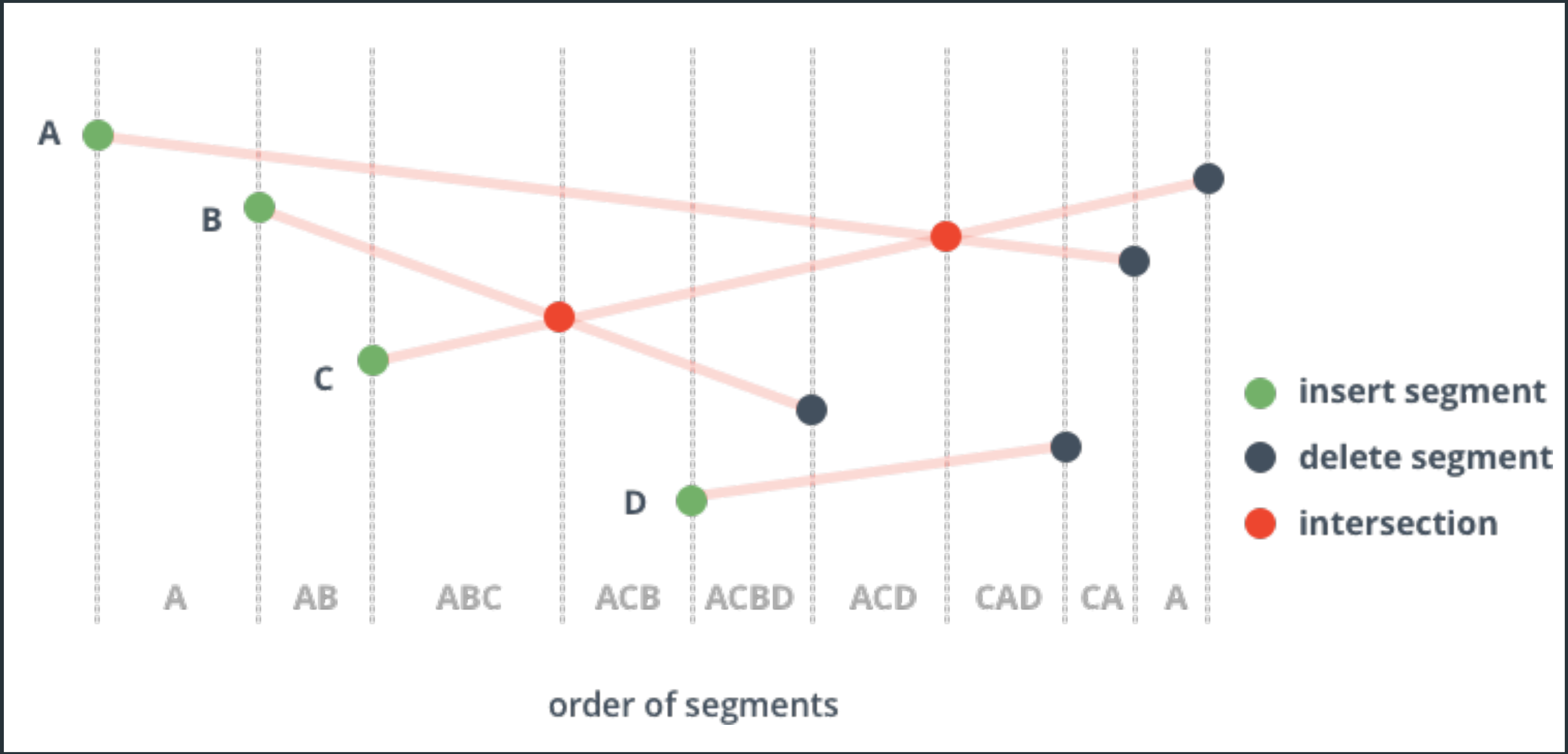
Event Queue: AL, BL, CL, DL, BR, DR, AR, CR



Current Event:

Segment List: A

Event Queue: BL, CL, DL, BR, DR, AR, CR



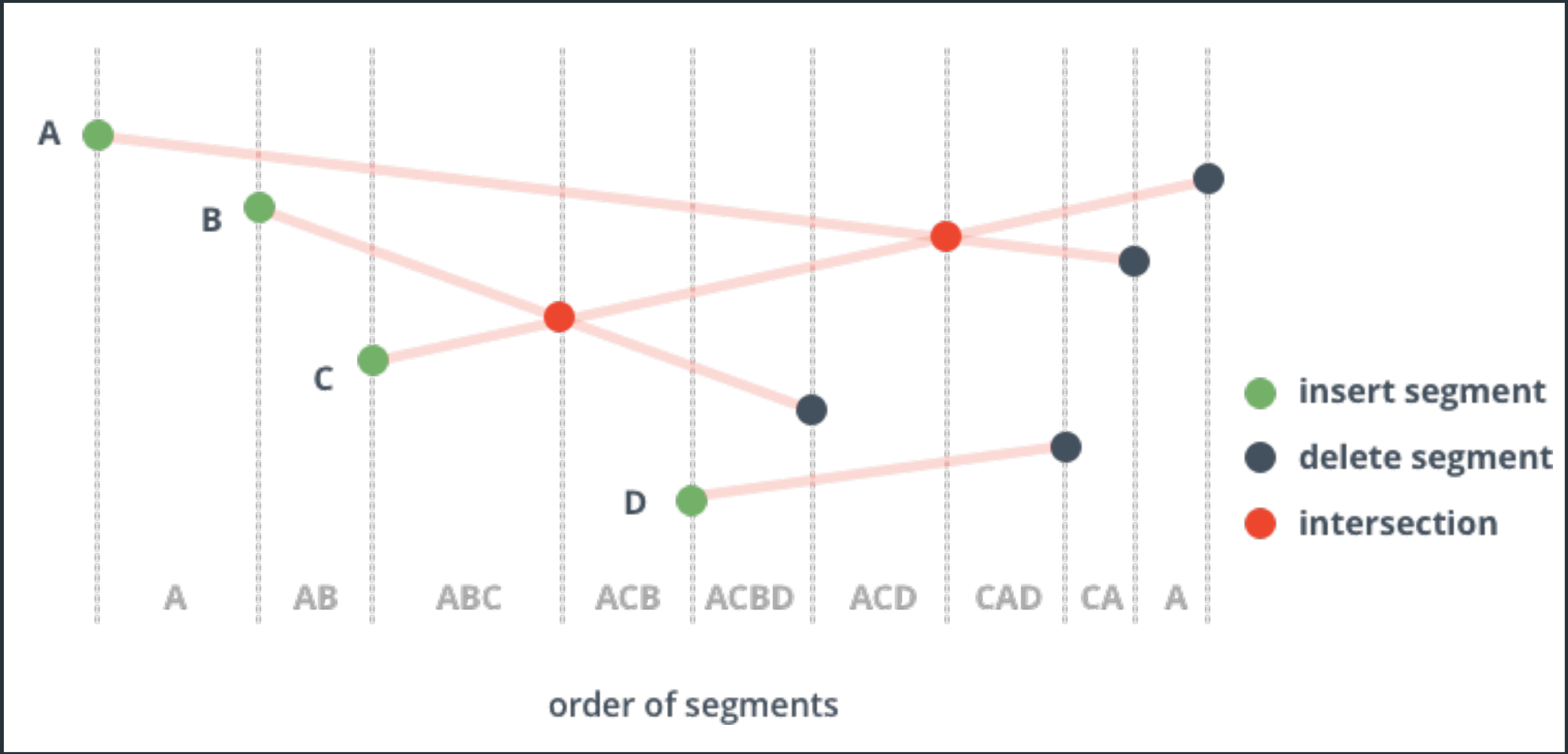
Current Event: AL

Seg Above: null

Seg Below: null

Segment List: A, B

Event Queue: CL, DL, BR, DR, AR, CR



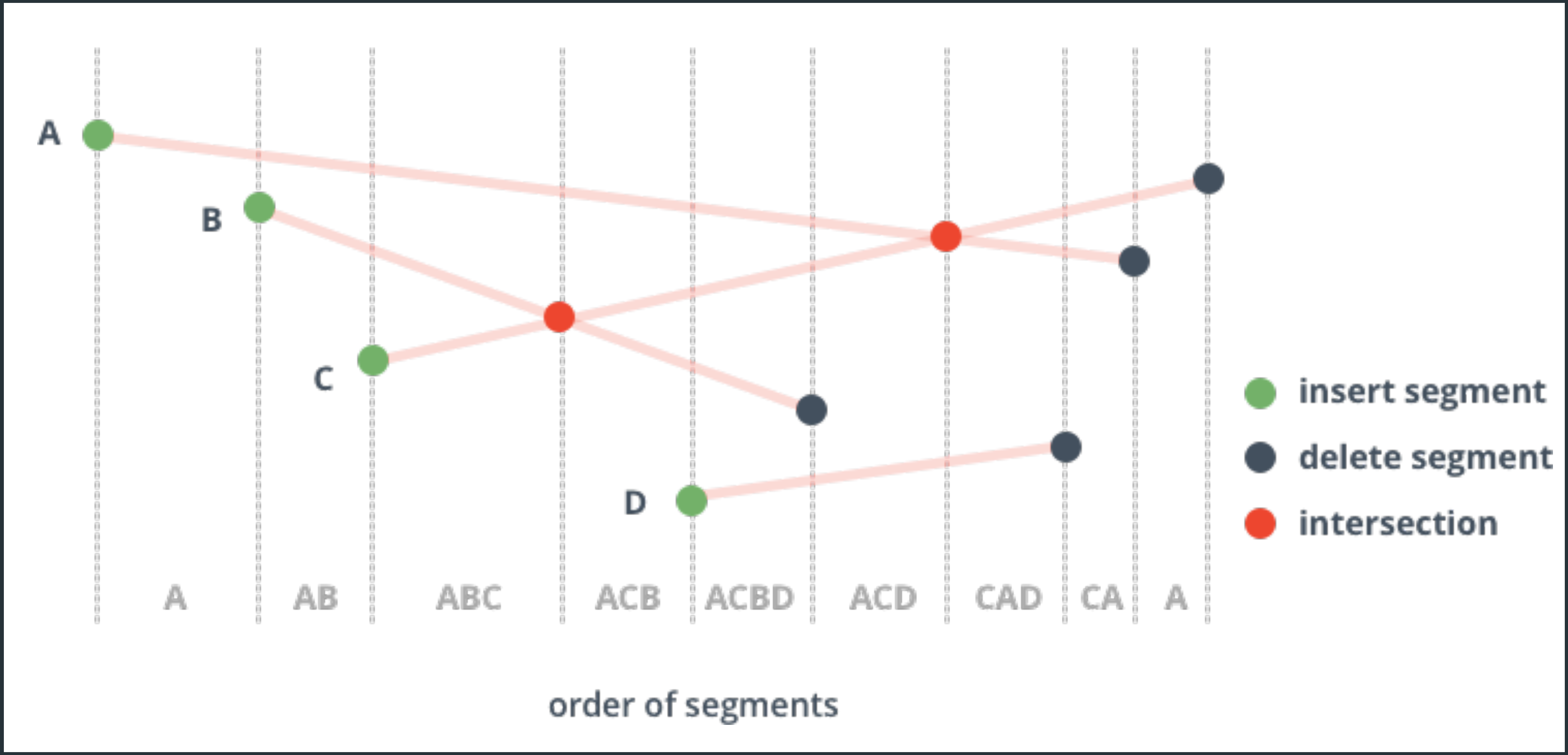
Current Event: BL

Seg Above: A

Seg Below: null

Segment List: A, B, C

Event Queue: DL, BR, DR, AR, CR



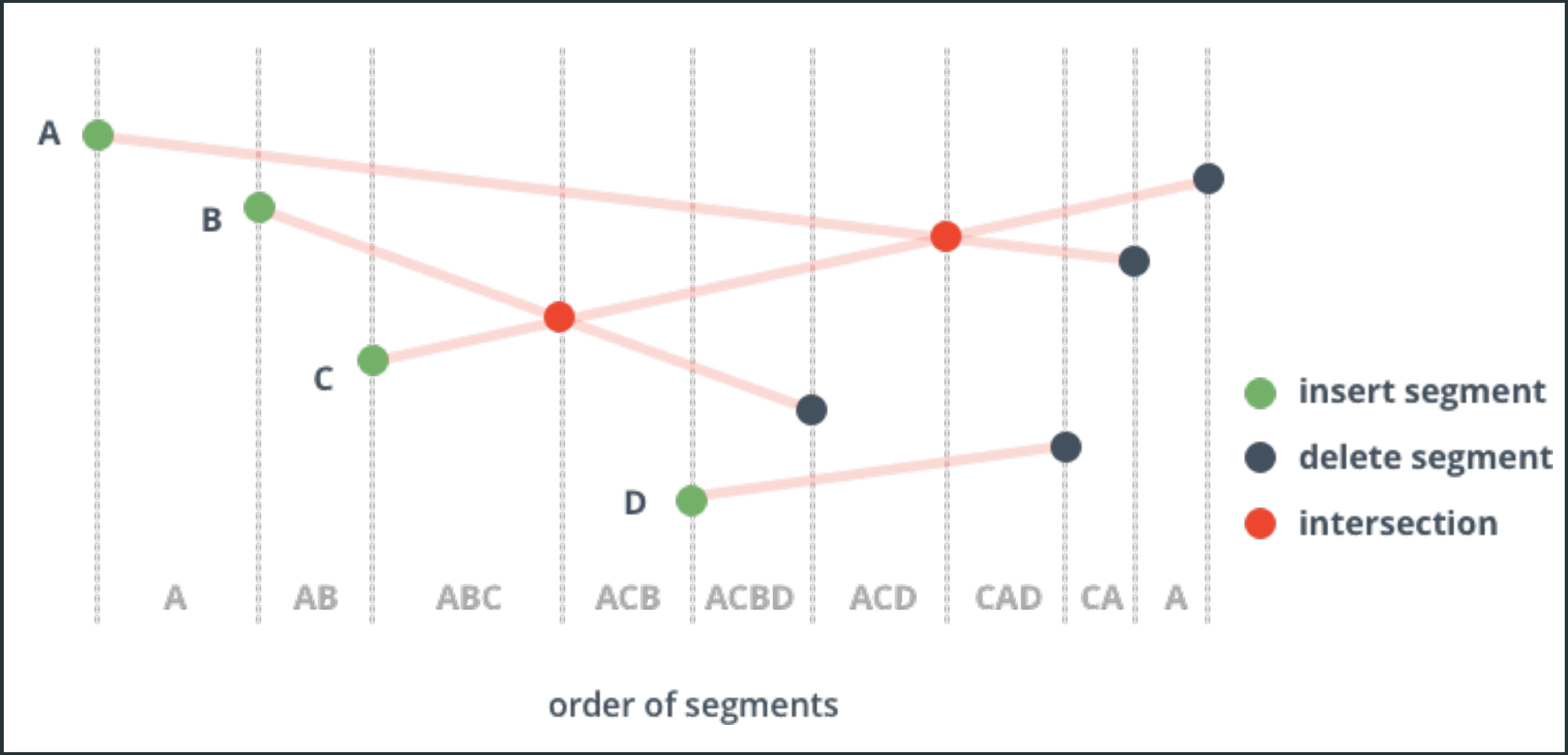
Current Event: CL

Seg Above: B

Seg Below: null

Segment List: A, B, C

Event Queue: BCI, DL, BR, DR, AR, CR



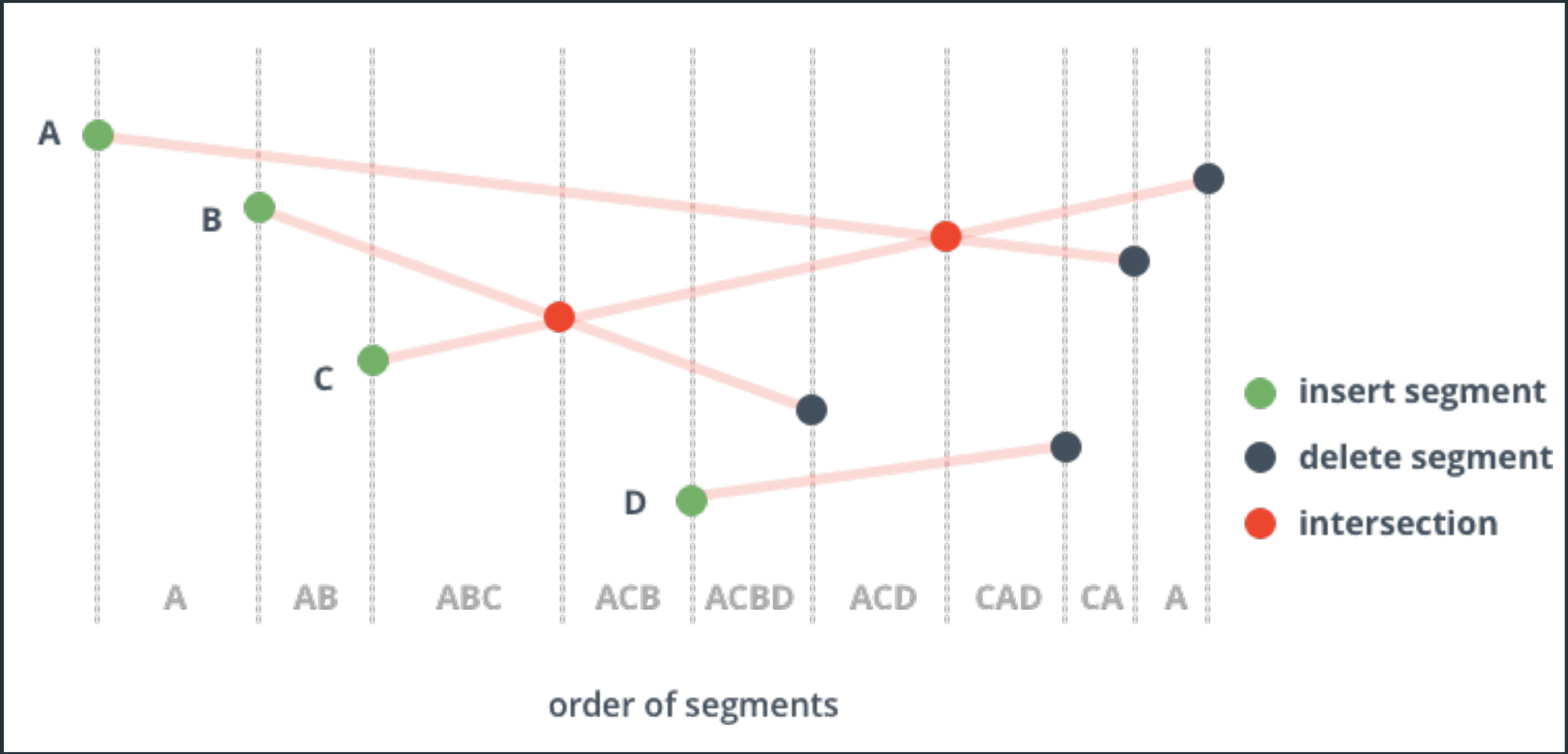
Current Event: CL

Seg Above: B

Seg Below: null

Segment List: A, B, C

Event Queue: DL, BR, DR, AR, CR



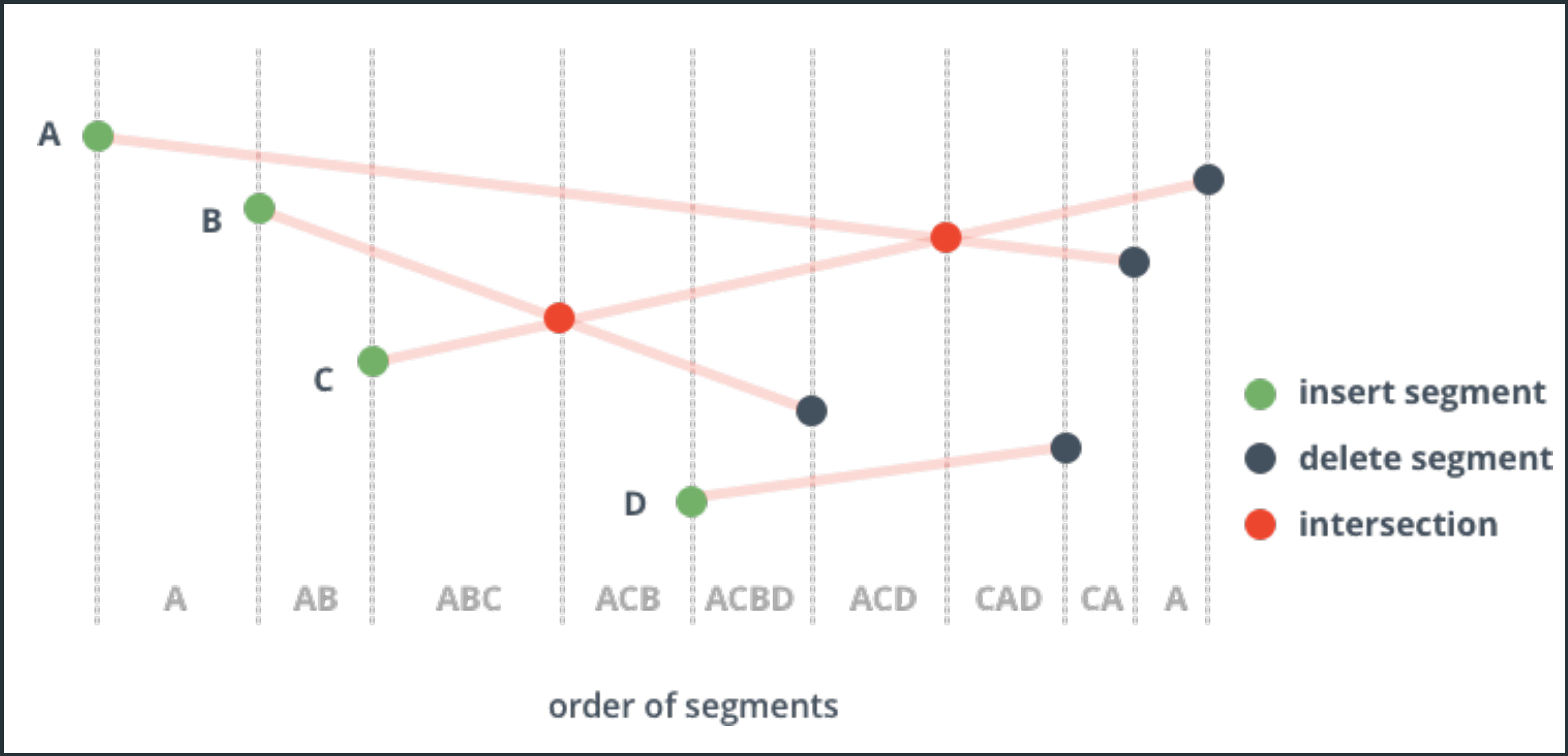
Current Event: BCI

Seg Above: B

Seg Below: C

Segment List: A, C, B

Event Queue: DL, BR, ACI, DR, AR, CR



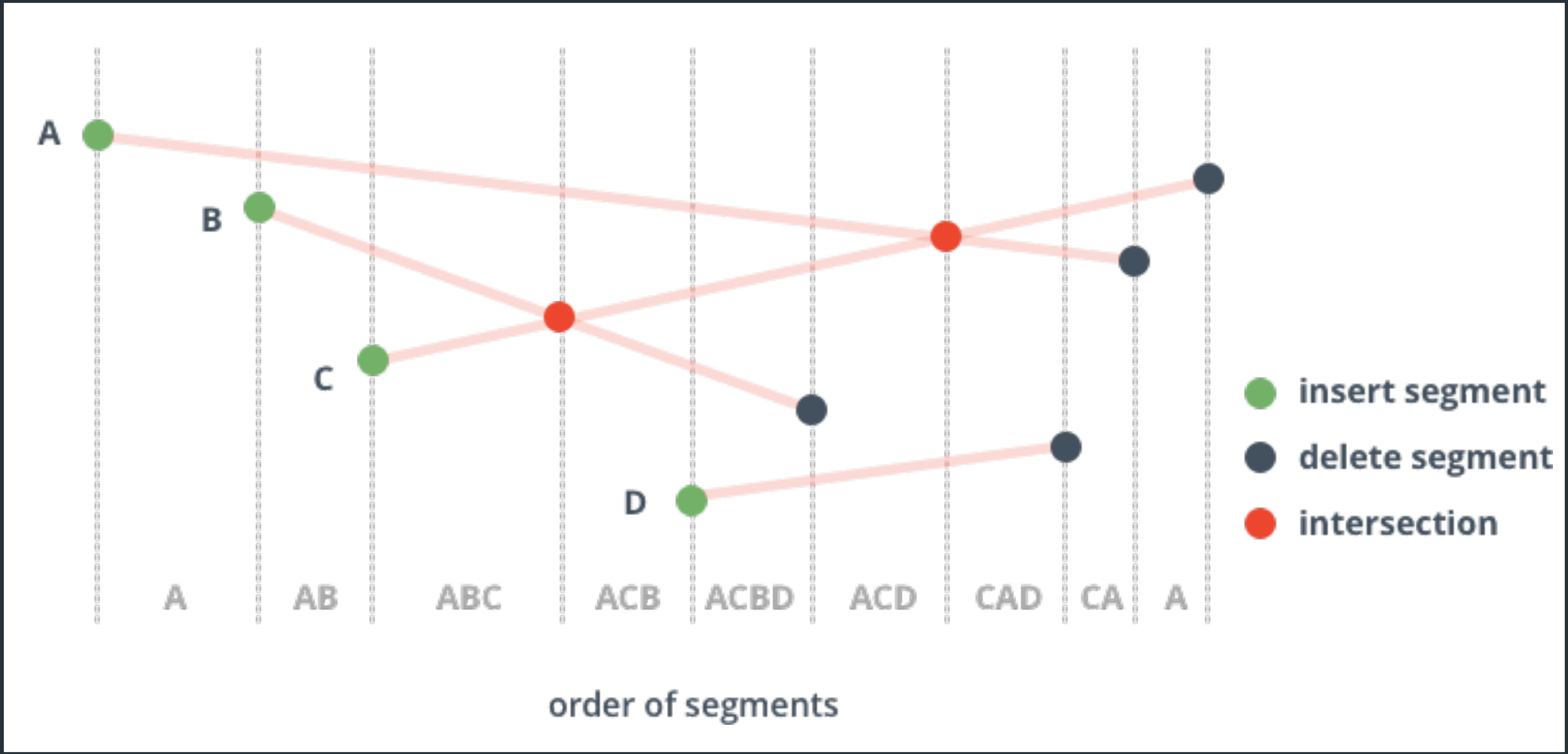
Current Event: BCI

Seg Above C: A

Seg Below B: null

Segment List: A, C, B, D

Event Queue: BR, ACI, DR, AR, CR



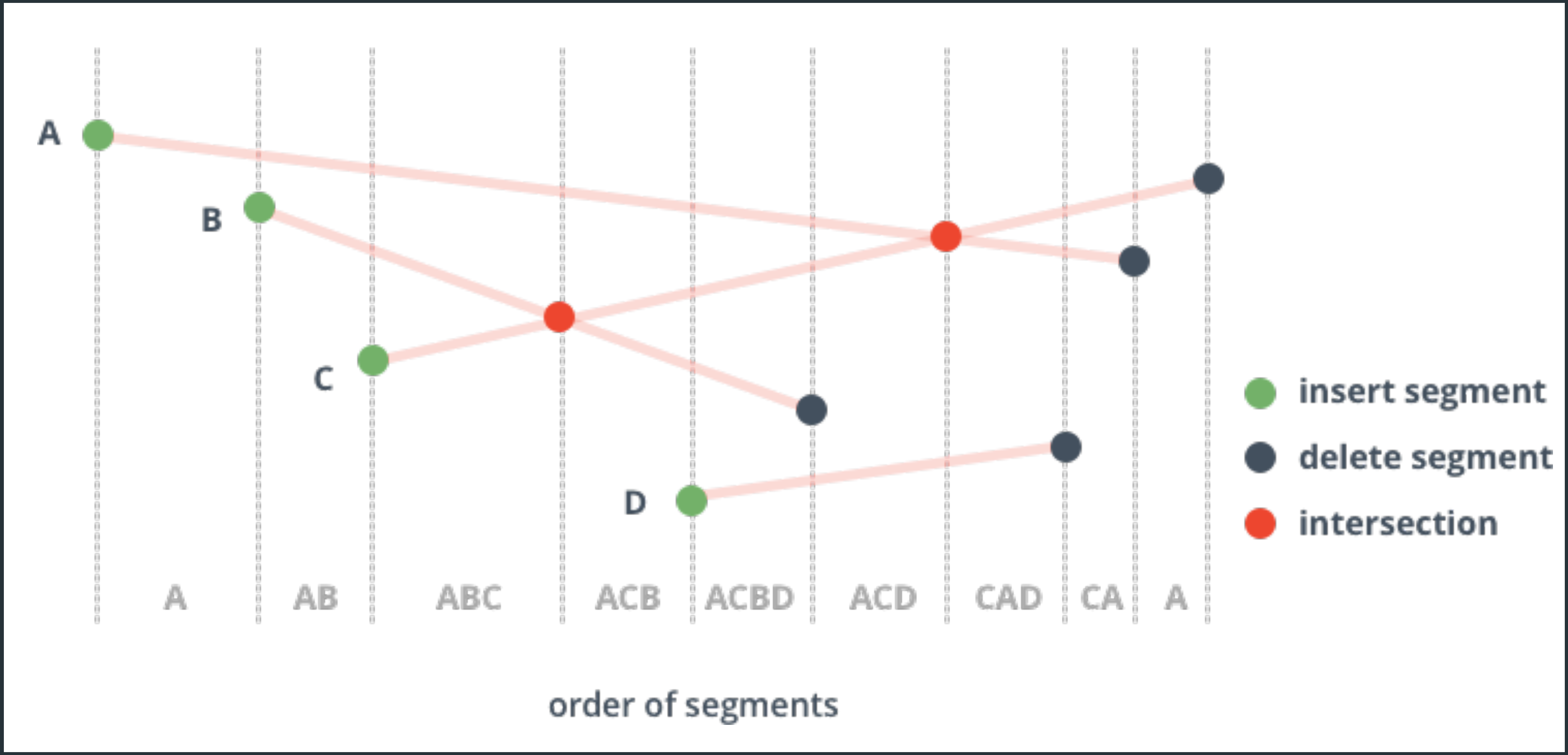
Current Event: DL

Seg Above: B

Seg Below: null

Segment List: A, C, D

Event Queue: ACI, DR, AR, CR



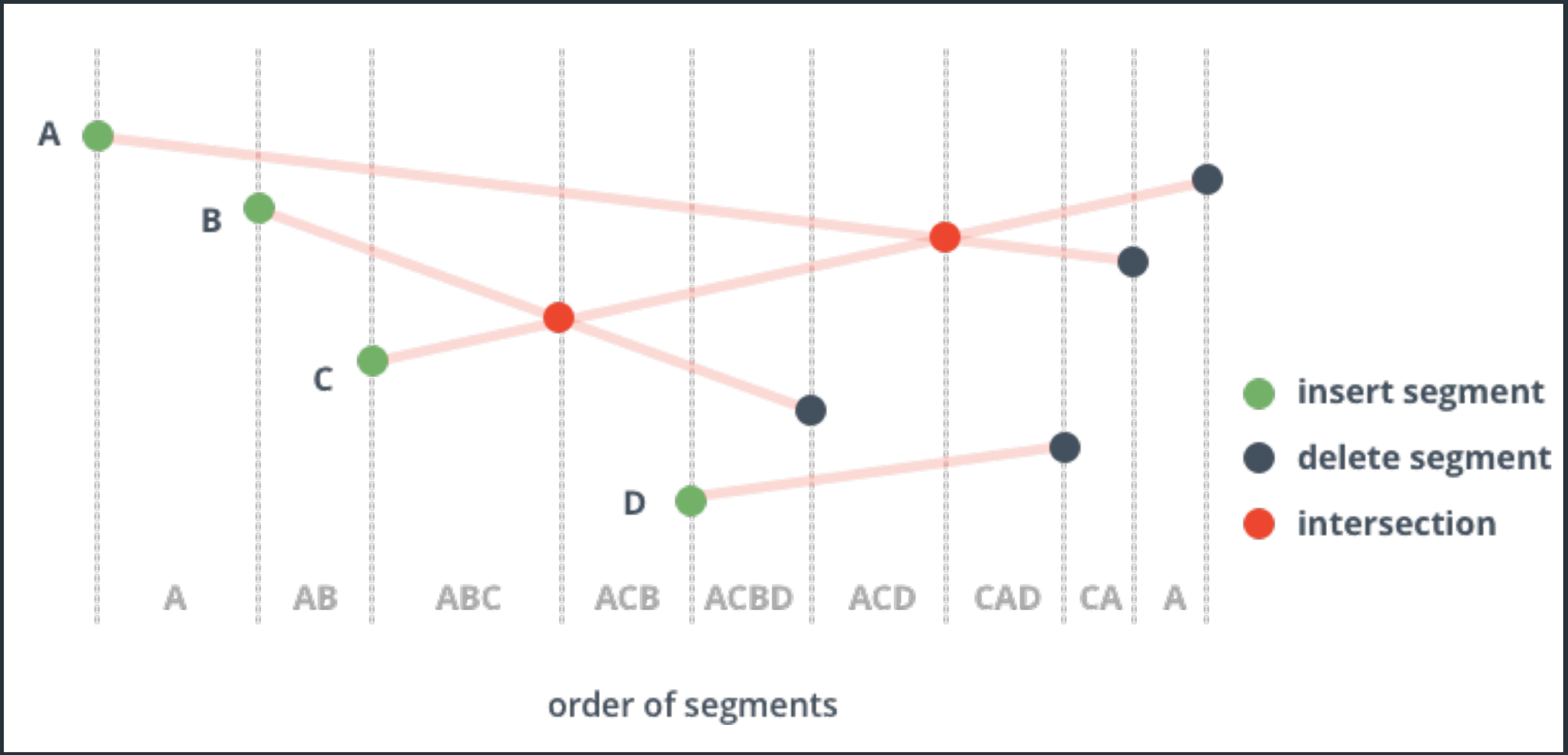
Current Event: BR

Seg Above: C

Seg Below: D

Segment List: C, A, D

Event Queue: DR, AR, CR



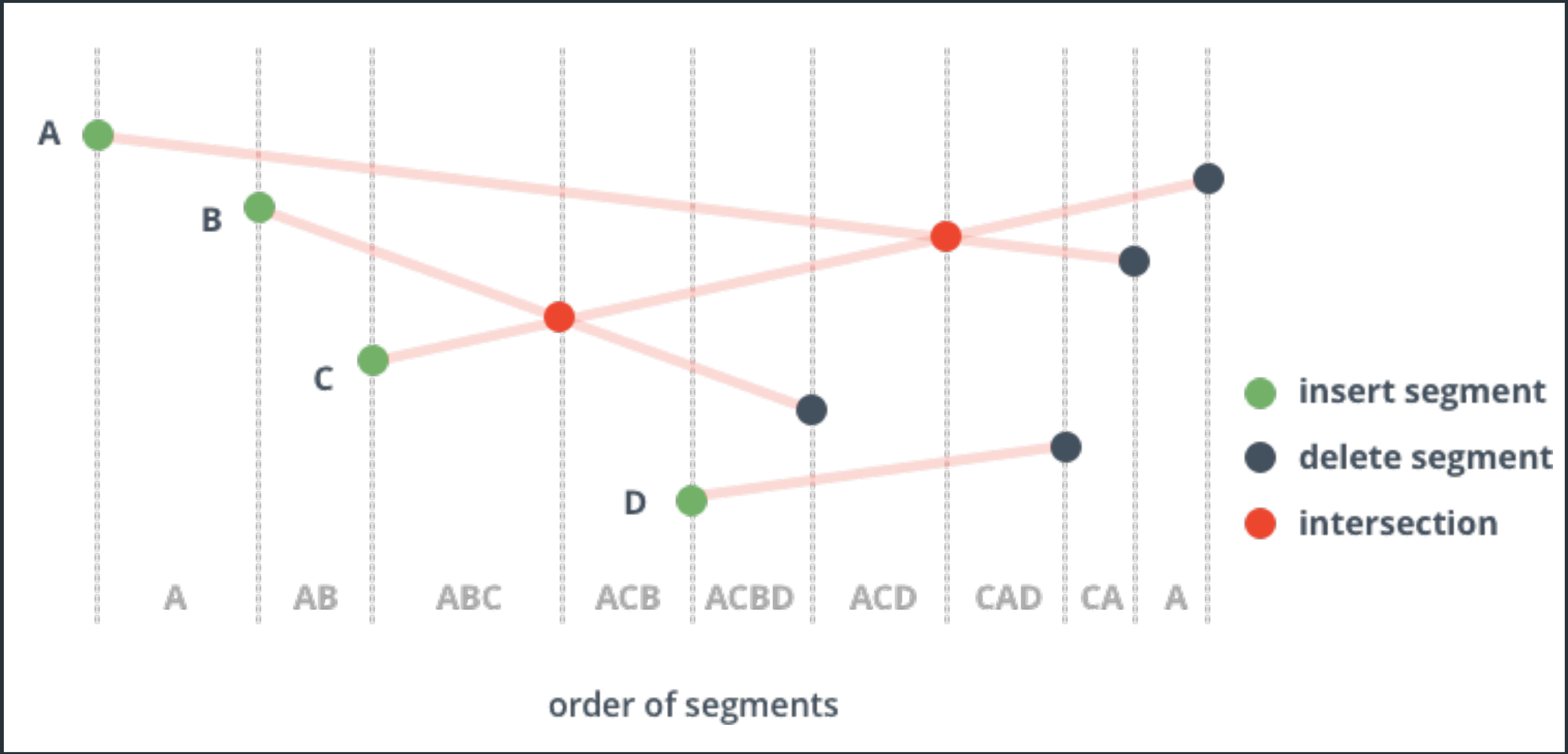
Current Event: ACI

Seg Above: A

Seg Below: C

Segment List: C, A, D

Event Queue: DR, AR, CR



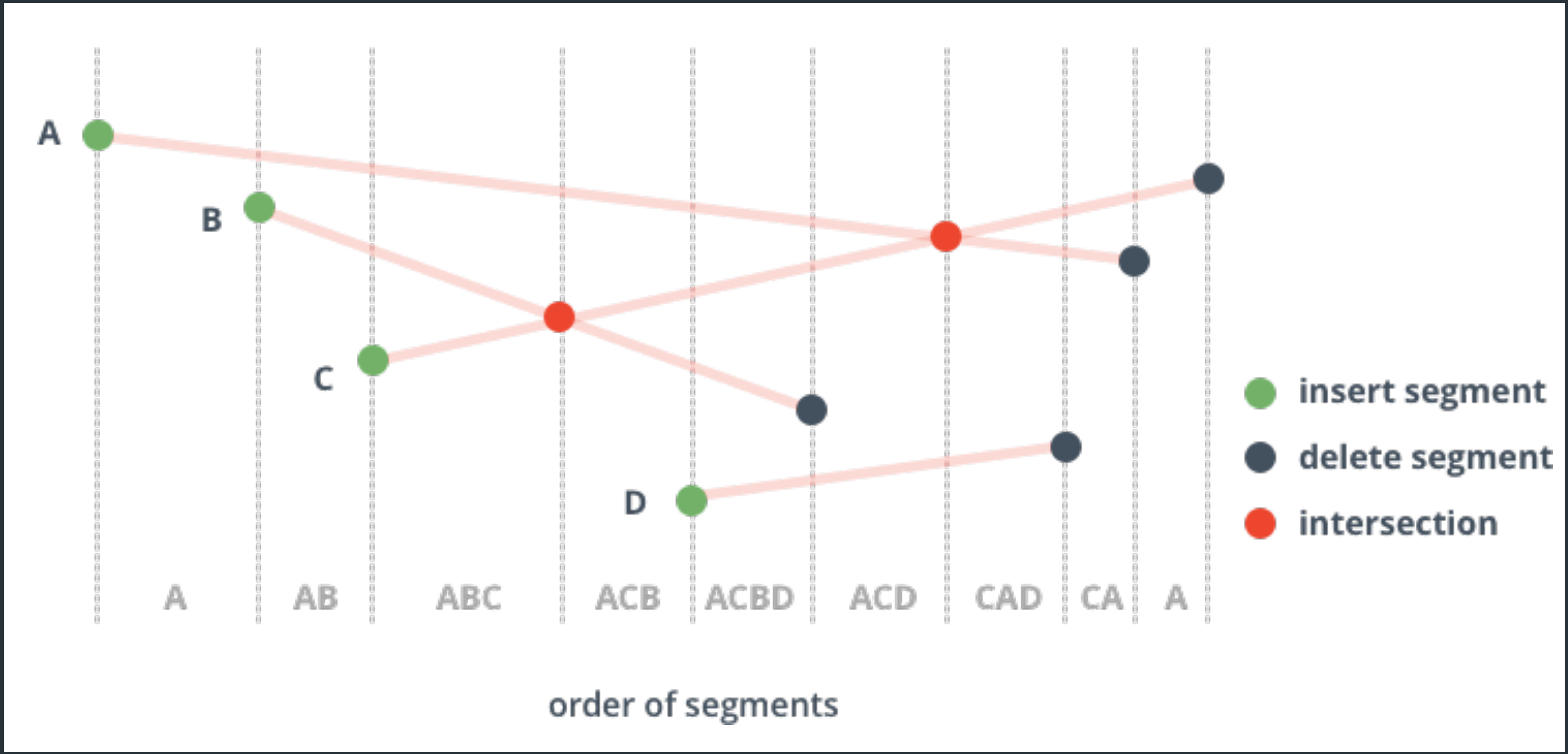
Current Event: ACI

Seg Above C : null

Seg Below A: D

Segment List: C, A,

Event Queue: AR, CR



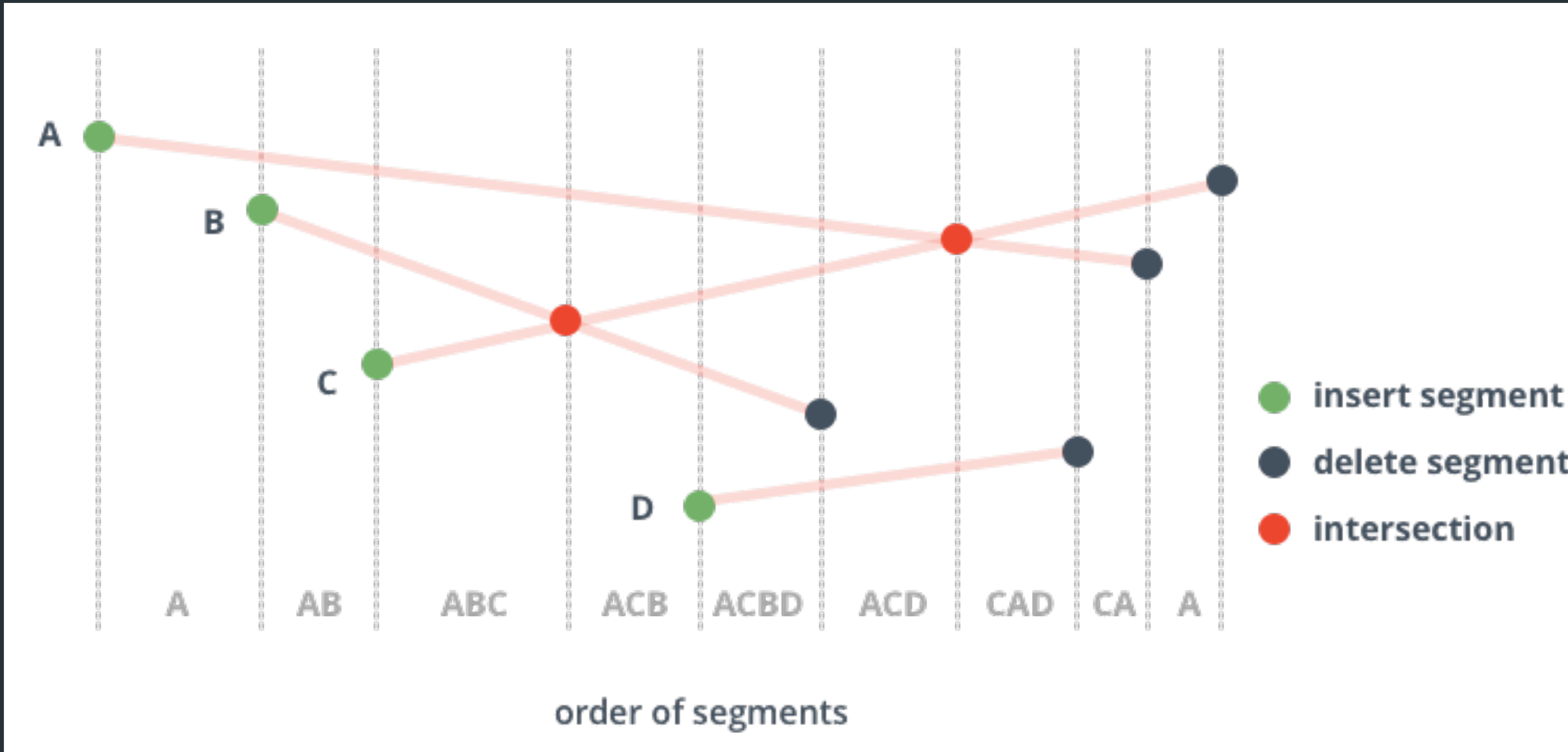
Current Event: DR

Seg Above: A

Seg Below: null

Segment List: C

Event Queue: CR



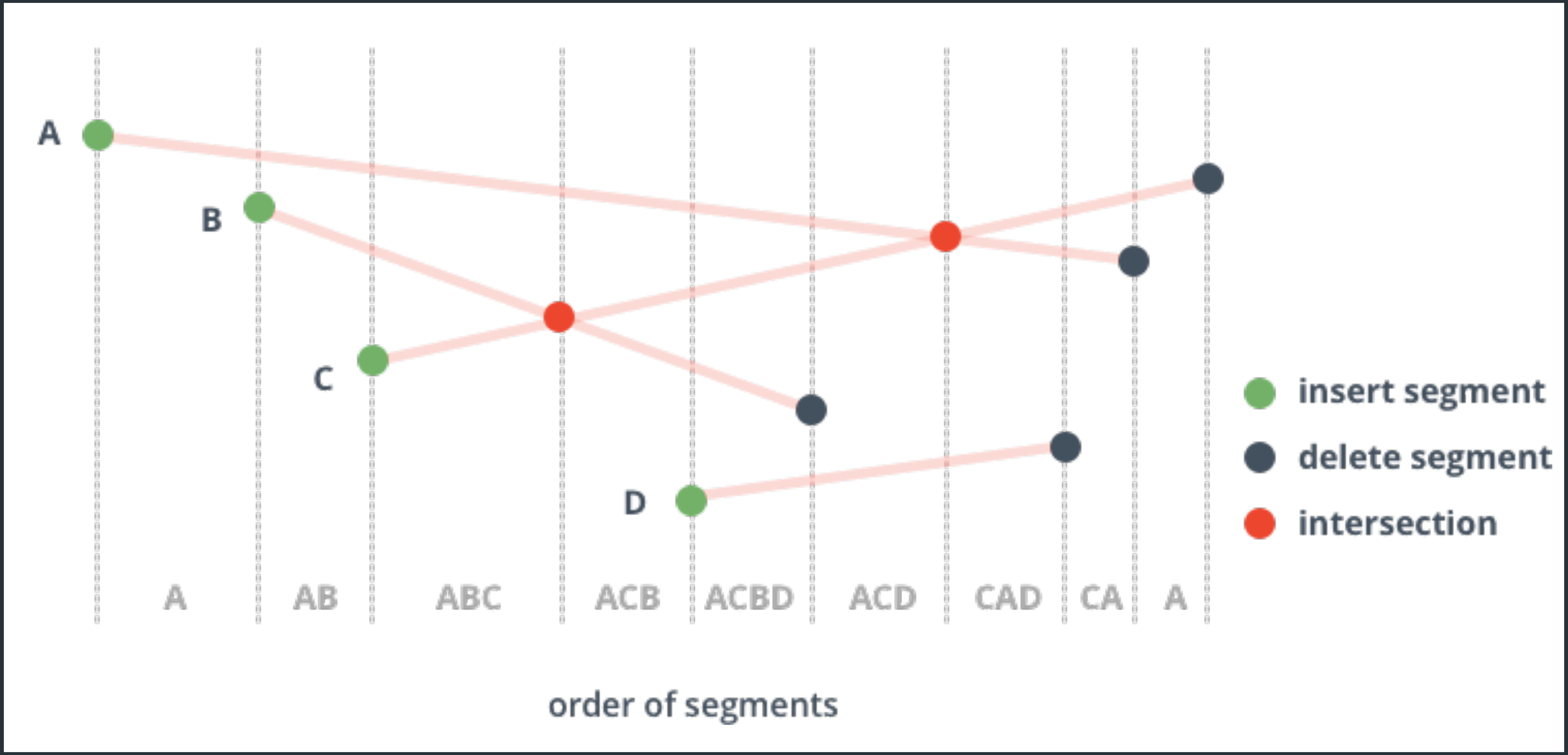
Current Event: AR

Seg Above: C

Seg Below: null

Segment List:

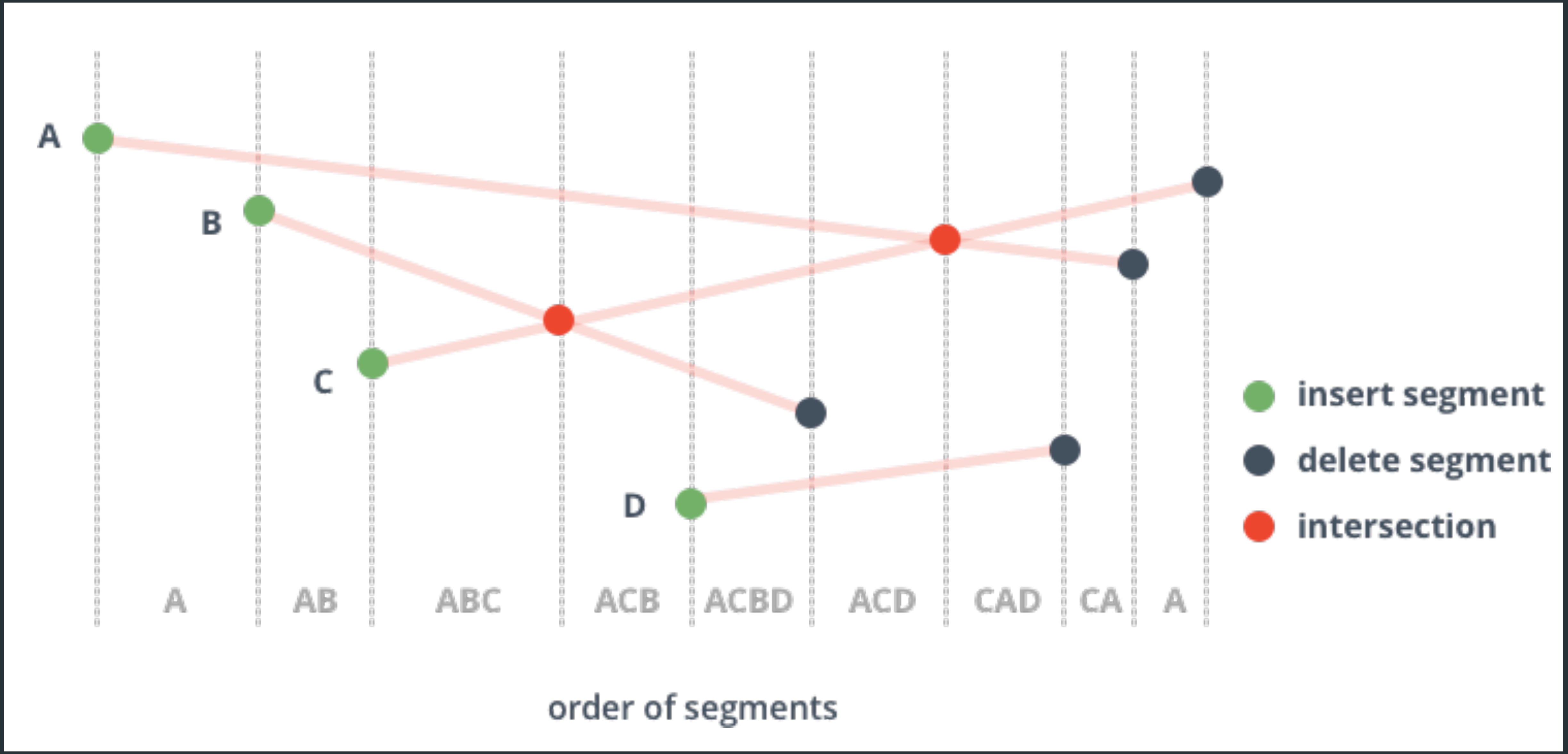
Event Queue:



Current Event: CR

Seg Above: null

Seg Below: null


A
B
C
D
insert segment
delete segment
intersection
A
AB
ABC
ACB
ACBD
ACD
CAD
CA
A
order of segments

Done!

Shamos-Hoey Algorithm

- This is the algorithm which Bentley ottoman was adopted from.
- Used for checking if an intersection exists rather than finding the set of intersections.
- Almost identical, but when you find and intersection, just return true.

Problem

- KattisID: polygon
- https://open.kattis.com/problems/polygon
- Given a set of points that define a polygon, determine if it is simple or not.
- Difficulty: 8.5

Problem

- Input size: 150 test cases
- Points are all integers
- n points 1 <= n <= 40 000
- 4 second time limit
- output: YES if polygon is simple, NO otherwise

Solution Idea

- If we have a polygon, and non-adjacent lines intersect (with some exceptions) it is not simple.
- We can use Shamos-Hoey/Bentley-Ottman
- Since we only want to output YES or NO we can use Shamos-Hoey.

```java
public class Point implements Comparable {
    int x;
    int y;

    public Point(int x, int y) {...
    }

    @Override
    public int compareTo(Object o) {...
    }
}

public class Line implements Comparable{
    Point left;
    Point right;
    int order;

    public Line(Point p1, Point p2, int order) {...
    }
    @Override
    public int compareTo(Object o) {...
    }

    public boolean checkNeighbors(Line l) {...
    }

    public boolean intersects(Line l) {...
    }
}
```

```java
public class Event implements Comparable{
    Point p;
    Line l;
    int side; // 0 = left, 1 = right
    Event(Point p, Line l, int side) {
    }

    @Override
    public int compareTo(Object o) {
    }
}

public static class YComparator implements Comparator<Line> {
}
```

```java
TreeSet<Line> activeLines = new TreeSet<Line>(new YComparator());
PriorityQueue<Event> eventQueue = new PriorityQueue<Event>();
```

Special cases we need to check for

- Two adjacent lines “intersect” but we don’t want that to say the polygon is intersecting.

- Solve by keeping track of adjacent lines while reading in data.

    - Additional problem: Neighbors *can* intersect.

```java
for(int i = 1; i < n; i++) {
    s = br.readLine().split(" ");
    x = Integer.parseInt(s[0]);
    y = Integer.parseInt(s[1]);
    Point cur = new Point(x,y);
    if(i!= 1){
        lastLine = aLine;
    }
    aLine = new Line(last, cur, i);
    if(i!=1){
        flag = lastLine.checkNeighbors(aLine);
    }
    eventQueue.add(new Event(aLine.left, aLine, 0));
    eventQueue.add(new Event(aLine.right, aLine, 1));
    last = cur;
    //System.out.println(aLine.left.x + "," + aLine.left.y
}

lastLine = aLine;
aLine = new Line(last, first, n);
flag = lastLine.checkNeighbors(aLine);
```

If (flag == true) before the start of the while loop, we know the polygon is not simple.

```java
while(!eventQueue.isEmpty()) {
    if(flag) break;
    Event e = eventQueue.poll();
    //System.out.println(e.p.x + "," + e.p.y + " " + e.side);

    if(e.side == 0) { //left
        Line lineE = e.l;
        activeLines.add(lineE);
        Line lineA = activeLines.higher(lineE);
        Line lineB = activeLines.lower(lineE);
        if(lineA != null){
            if(lineE.intersects(lineA)) {
                simple = false;
                break;
            }
        }
        if(lineB != null){
            if(lineE.intersects(lineB)) {
                simple = false;
                break;
            }
        }
    }
    else {
        Line lineE = e.l;
        Line lineA = activeLines.higher(lineE);
        Line lineB = activeLines.lower(lineE);
        activeLines.remove(lineE);
        if(lineA == null || lineB == null);
        else if(lineA.intersects(lineB)){
            simple = false;
            break;
        }
    }
    eventQueue.remove(e);
}
```

Failed Solution

- My solution gets TLE on the 4th and final test case
- Note: There are currently no Java solutions in the problem
- Fastest solution: 0.31s in C++
- Checking CTU open website for judge data seems futile: their Java solution gives WA on the 4th test case.

Sources

- O’Rourke, Joseph. *Computational Geometry in C*. 2nd ed. 1998.

- https://www.hackerearth.com/practice/math/geometry/line-sweep-technique/

- http://geomalgorithms.com/a09-_intersect-3.html

- https://en.wikipedia.org/wiki/Bentley%E2%80%93Ottmann_algorithm

- open.kattis.com/problems/polygon