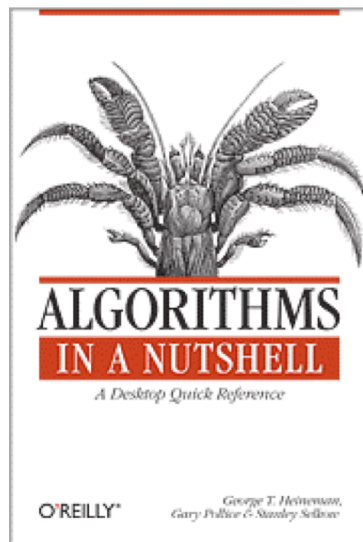


Algorithms in a Nutshell



Session 8

Computational Geometry

2:50 – 3:40

Outline

- Overview
- Themes
 - Divide and Conquer
- Problems
 - Line Segment Intersection

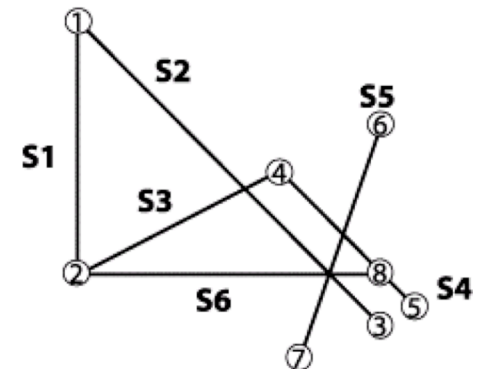
Line Segment Intersection

Input: Set of n line segments using Cartesian coordinates

Output: Set of k intersection points that describe all line segments that intersect at these k points.

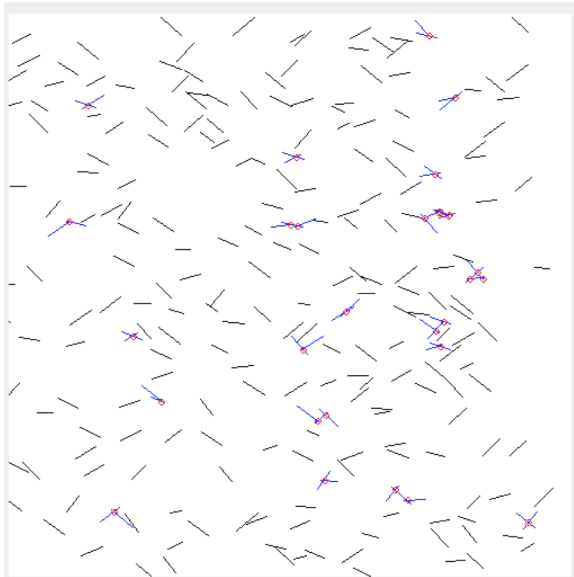
Quick Question:

How many distinct intersection points exist?

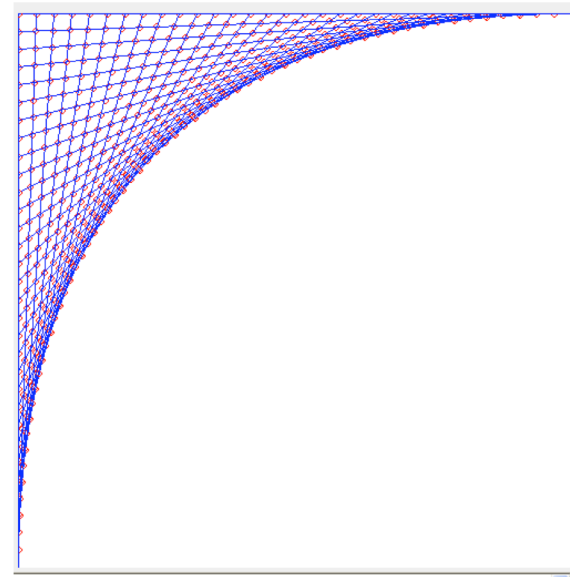


Quick Question

- How many intersection points exist?




$n=256$
 $k=28$



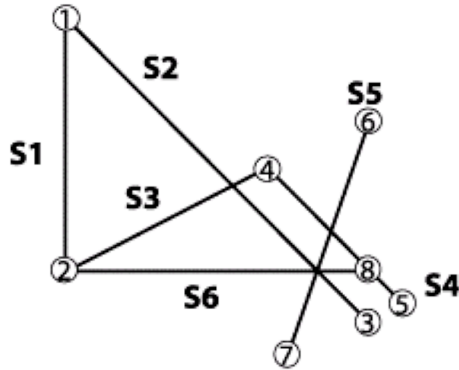
$n=32$
 $k=496$

Brute Force Intersection

BRUTE FORCE INTERSECTION			 Brute Force
Best	Average	Worst	
$O(n^2)$	$O(n^2)$	$O(n^2)$	

intersections (S)

- foreach** $s_1 \in S$ **do**
- foreach** $s_2 \in S - \{s_1\}$ **do**
- $p =$ intersection point of s_1 and s_2
- if** (p exists) **then** record (p, s_1, s_2)
- end**

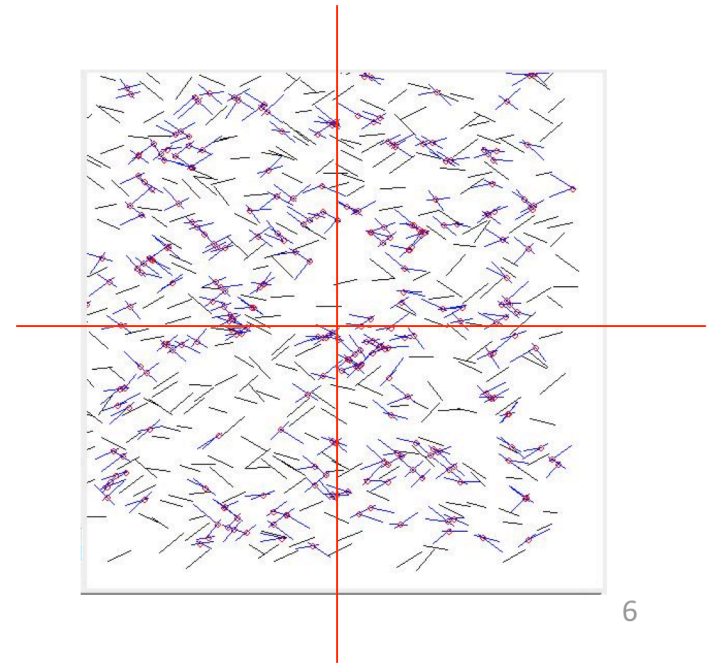


Works best when expected number of intersections $k \cong O(n^2)$

Doesn't take advantage of the actual **position** of lines

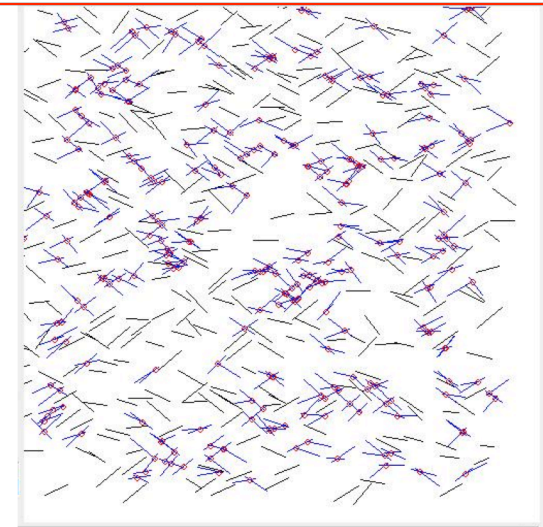
Decompose problem

- Divide and conquer
 - Essential strategy for dividing a problem into sub-problems
 - Does not seem applicable



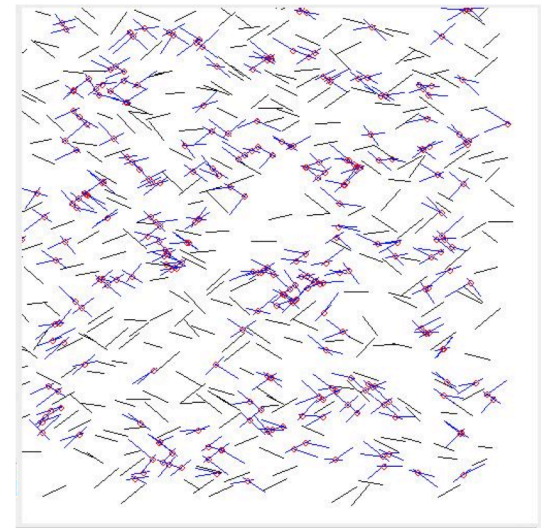
Decompose problem

- But borrow a concept from Calculus
 - Consider Infinitesimal slice moving from top to bottom in a “Sweep”
 - If two segments intersect then at some point they were neighbors on sweep line



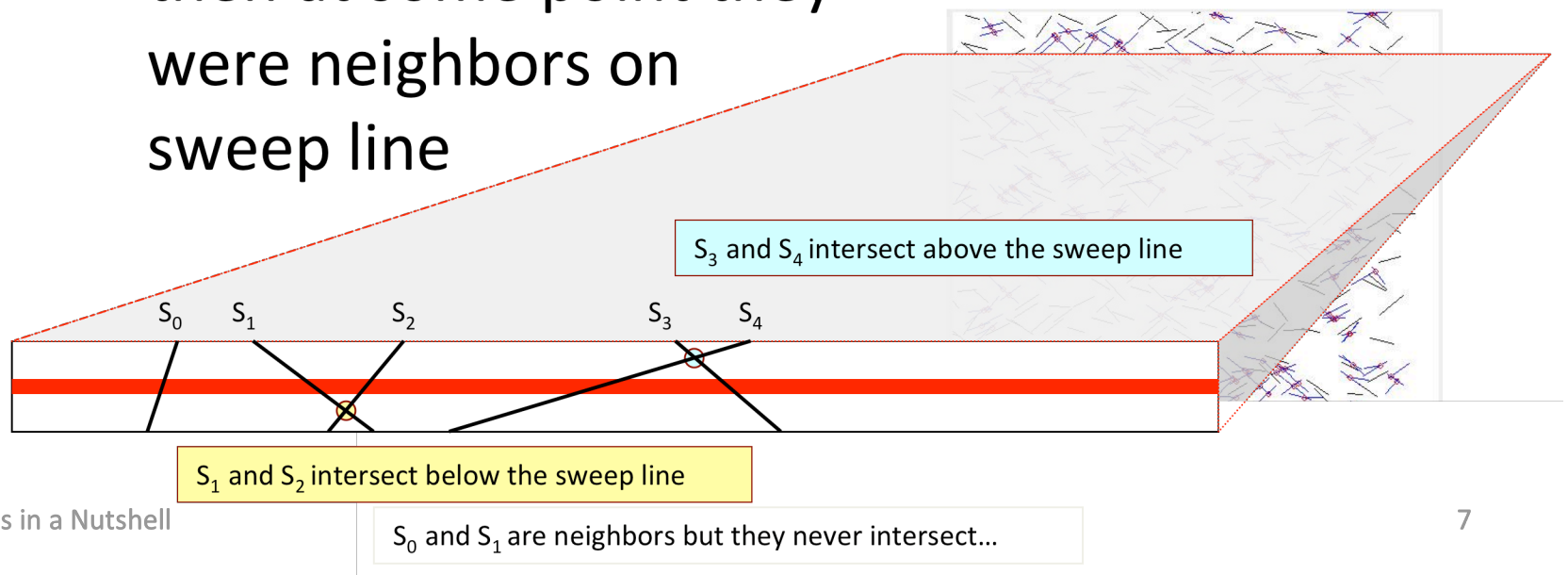
Decompose problem

- But borrow a concept from Calculus
 - Consider Infinitesimal slice moving from top to bottom in a “Sweep”
 - If two segments intersect then at some point they were neighbors on sweep line



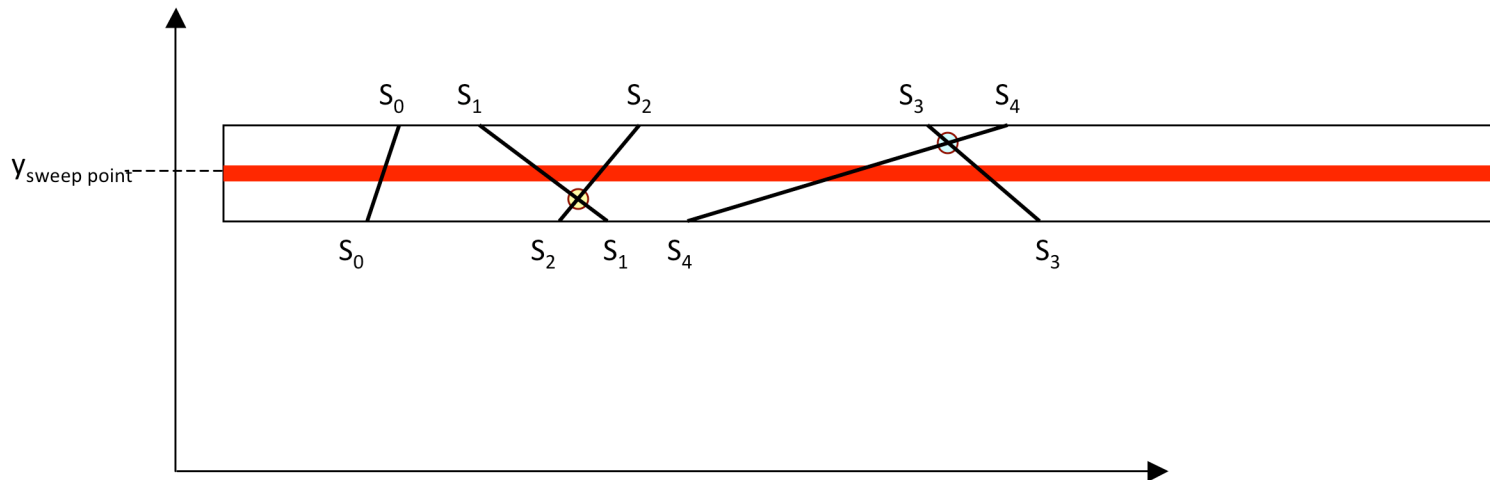
Decompose problem

- But borrow a concept from Calculus
 - Consider Infinitesimal slice moving from top to bottom in a “Sweep”
 - If two segments intersect then at some point they were neighbors on sweep line



Line state: Swap Neighbors

- After intersection, neighbors swap relative position
- Line State ordered left to right at the sweep point (y-coordinate of sweep line)



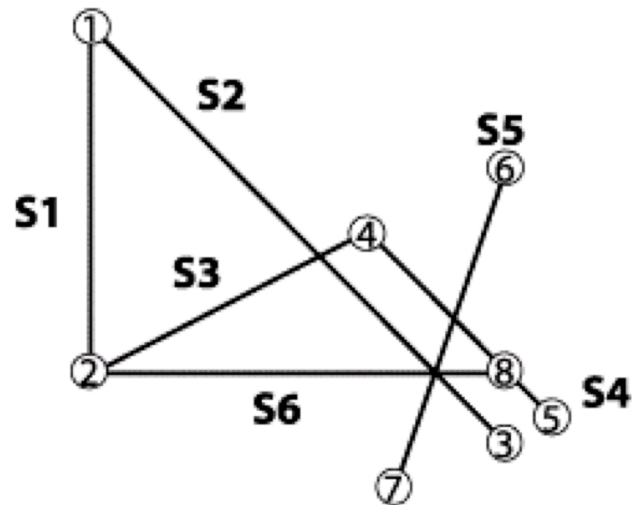
Algorithm Outline

- Initialize
 - Create Event Queue with *interesting initial points*
 - Each point knows its {upper, lower, intersecting}
 - Set LineState to \emptyset
- Process queue from top to bottom
 - “Handle” each Event Point

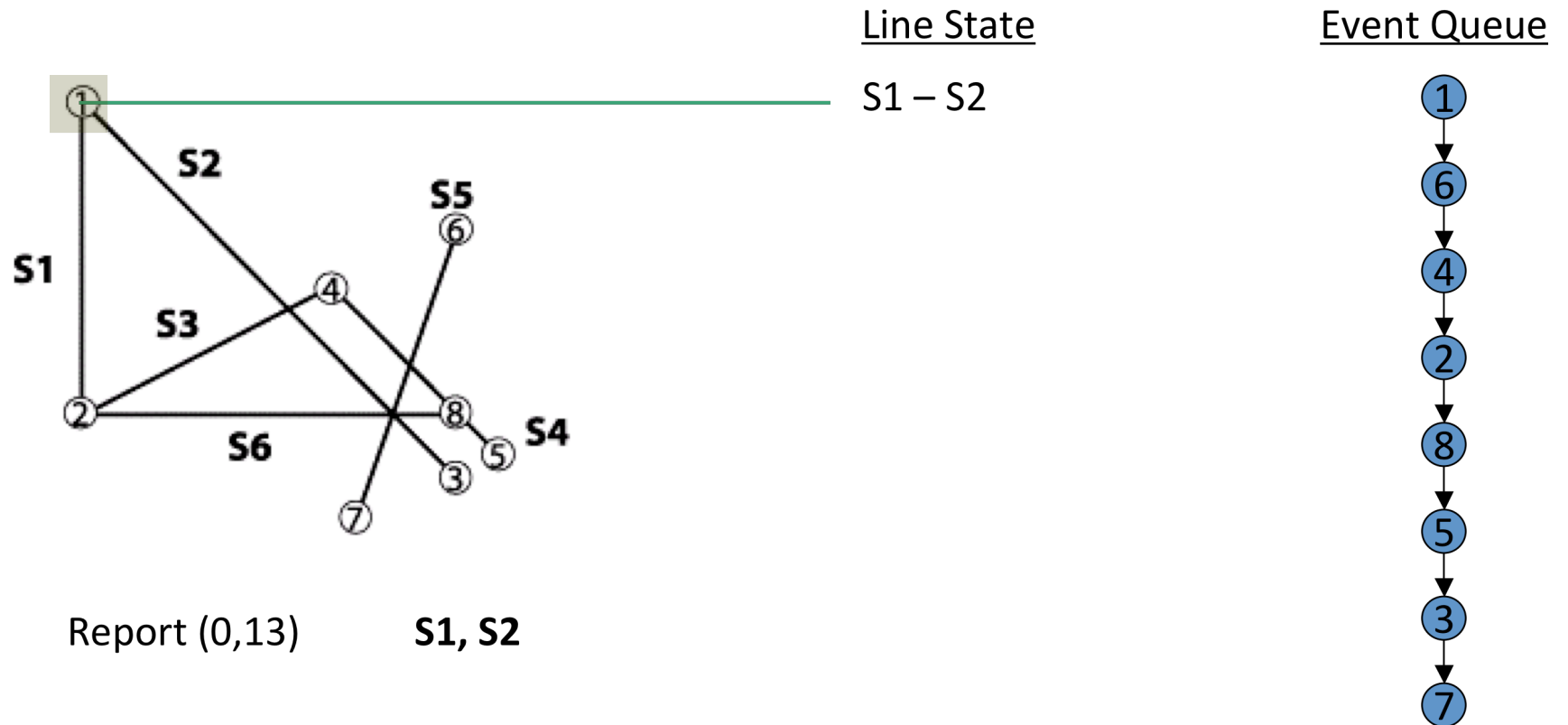
Example Execution

Line State

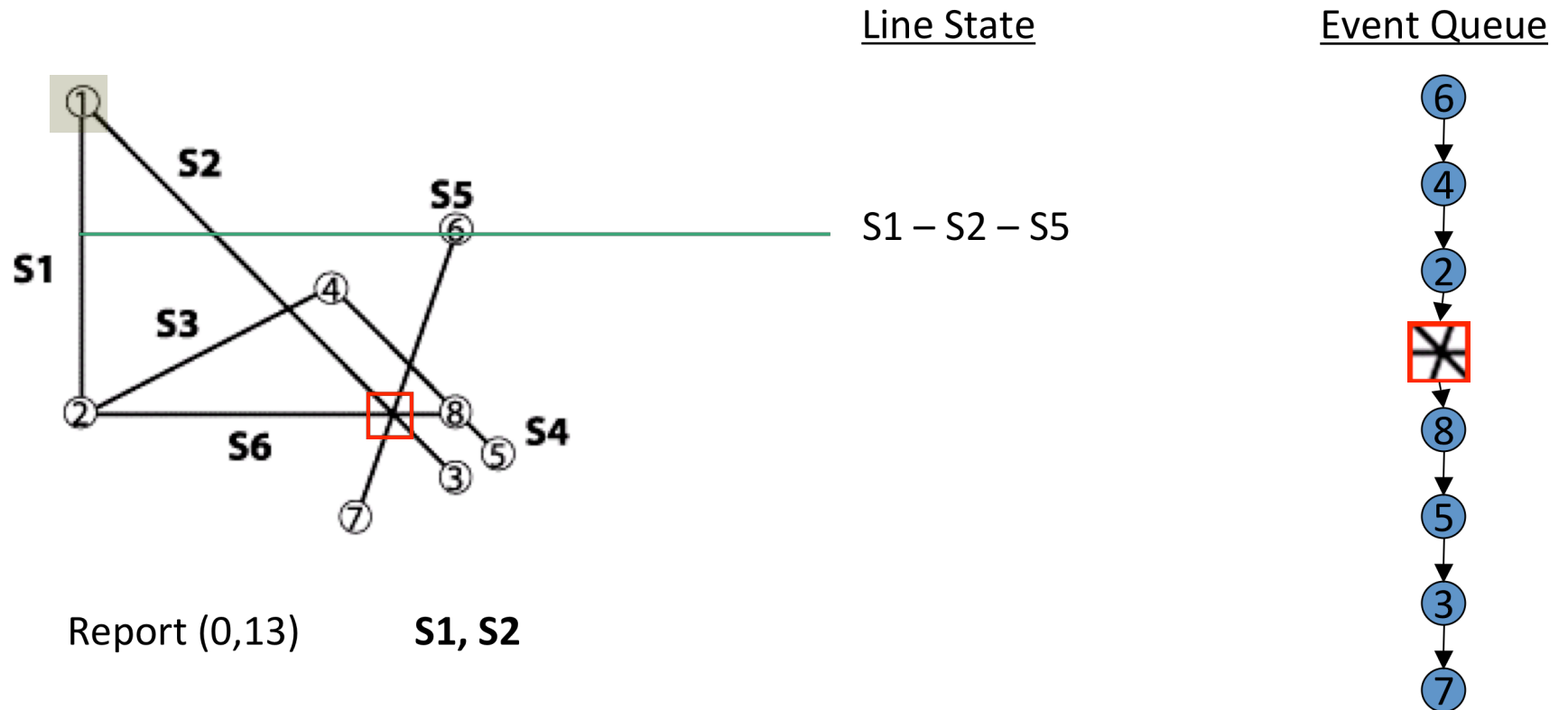
Event Queue



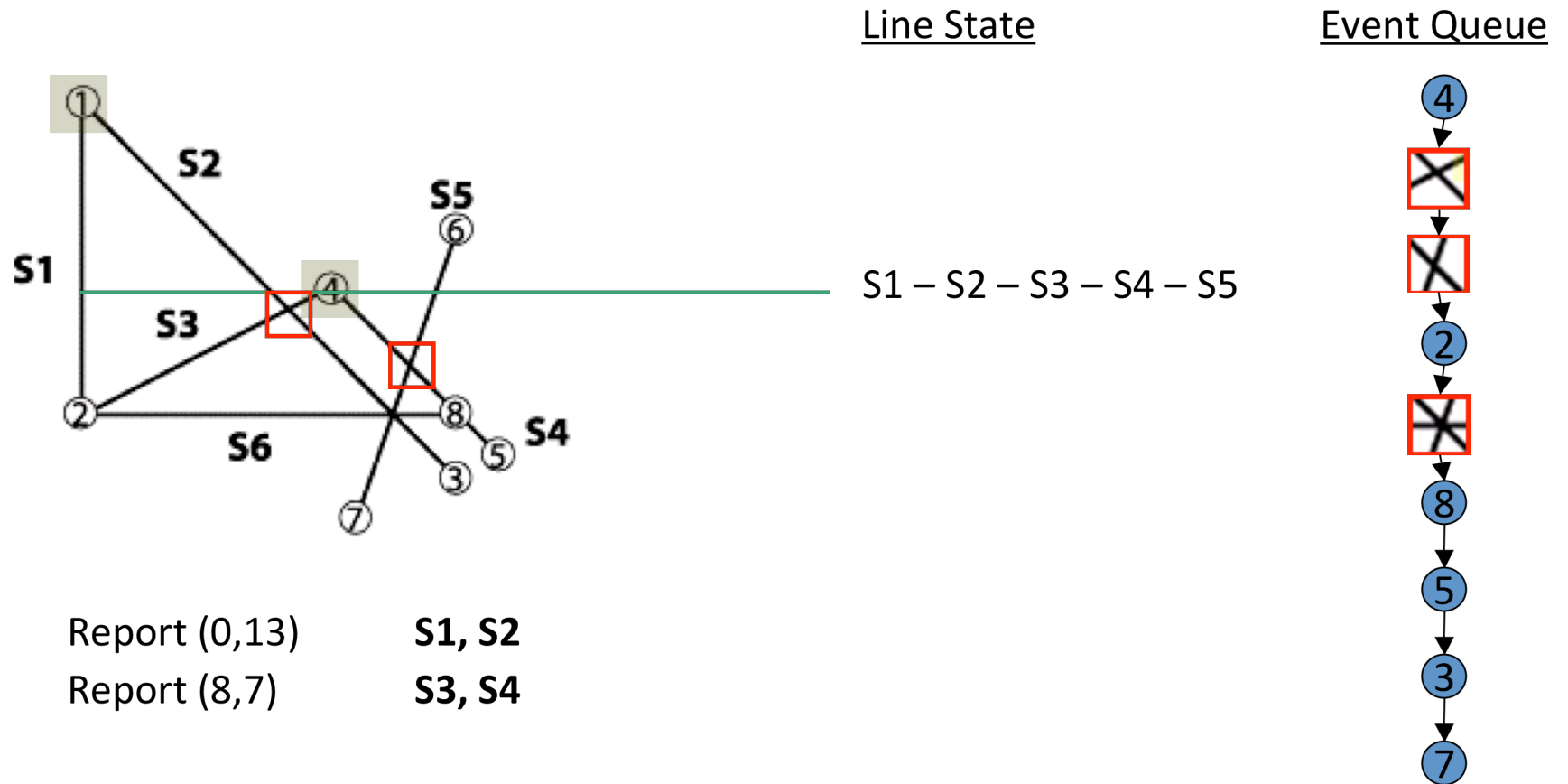
Example Execution



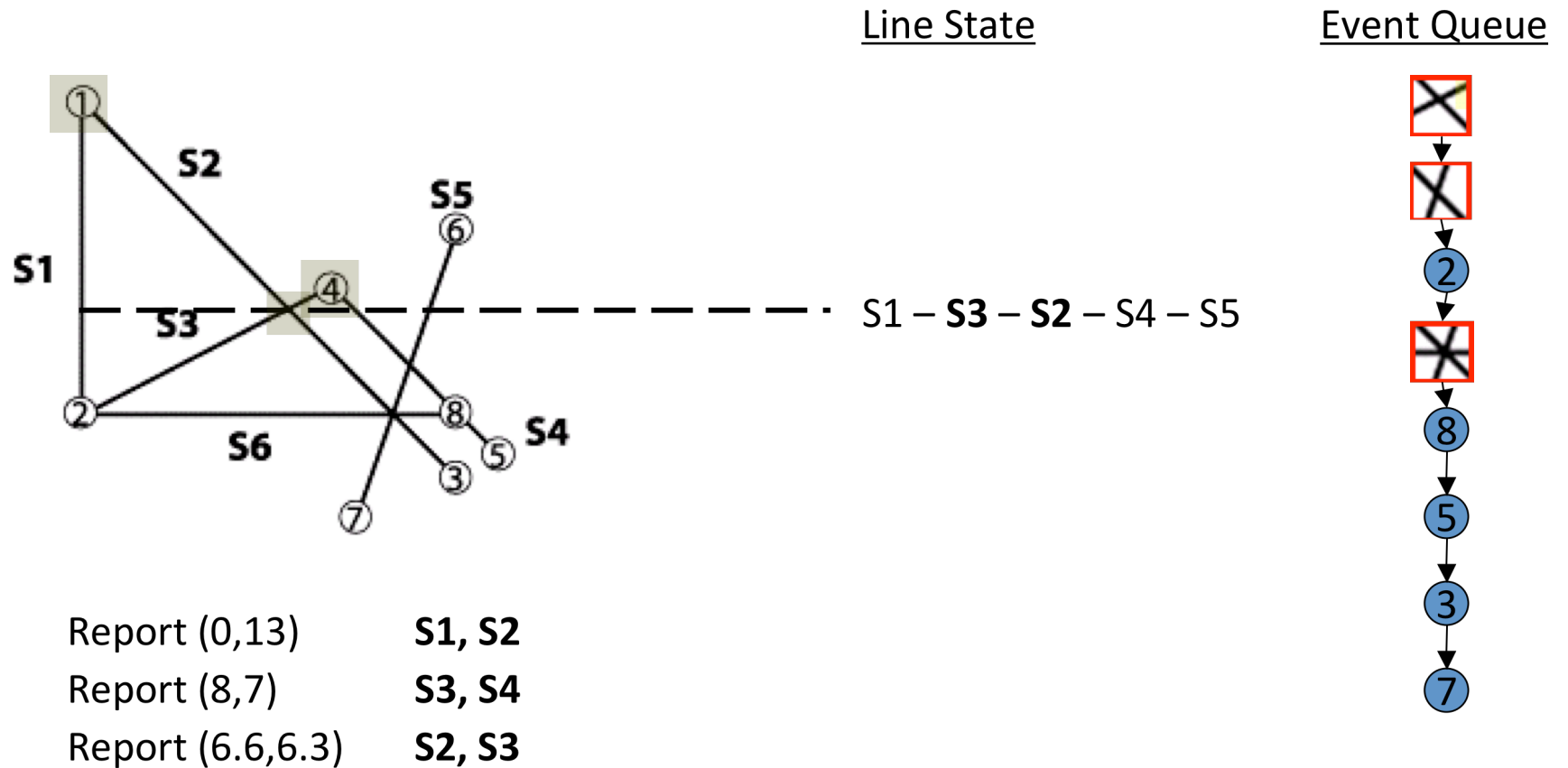
Example Execution



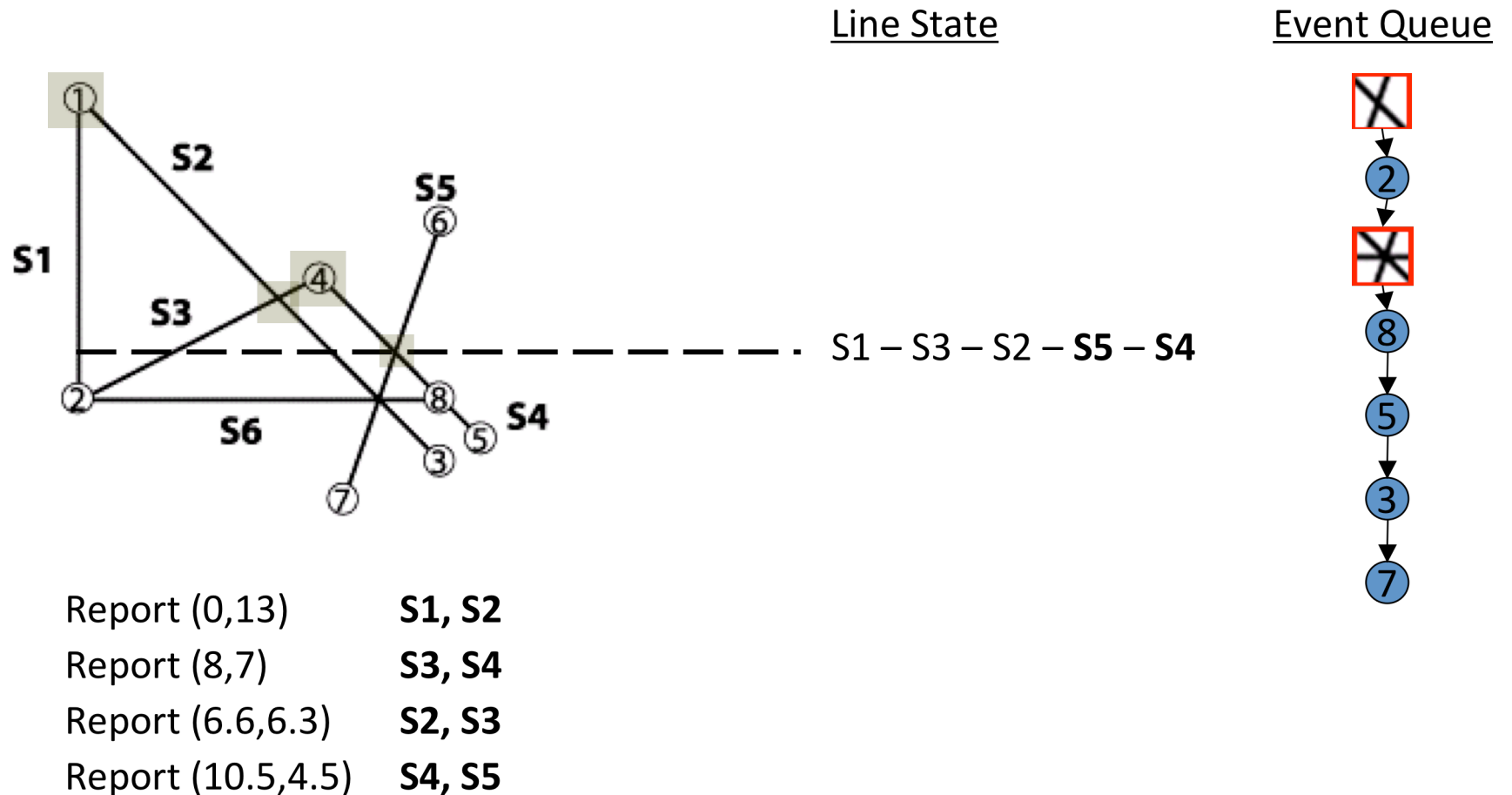
Example Execution



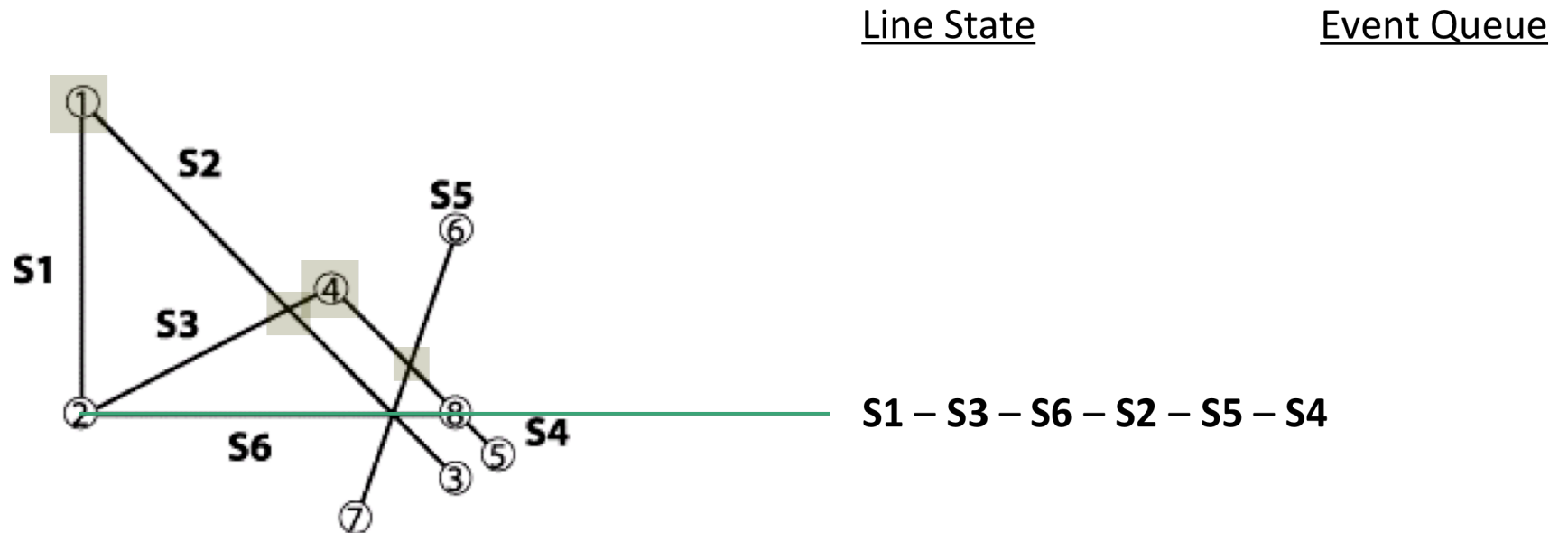
Example Execution



Example Execution



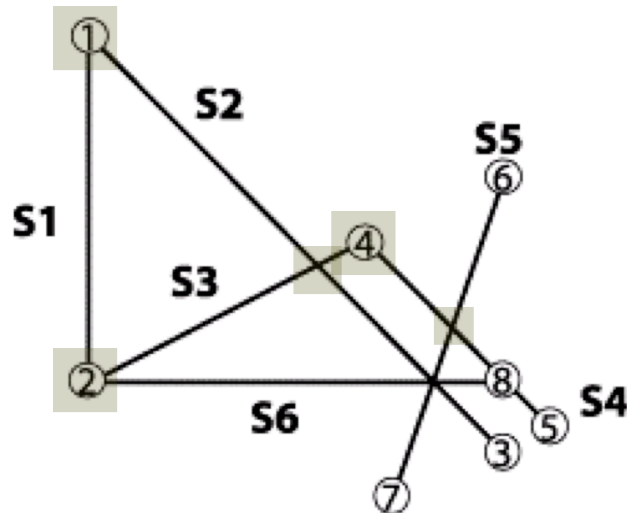
Example Execution



Report (0,13)	S1, S2
Report (8,7)	S3, S4
Report (6.6,6.3)	S2, S3
Report (10.5,4.5)	S4, S5

Example Execution

Line State



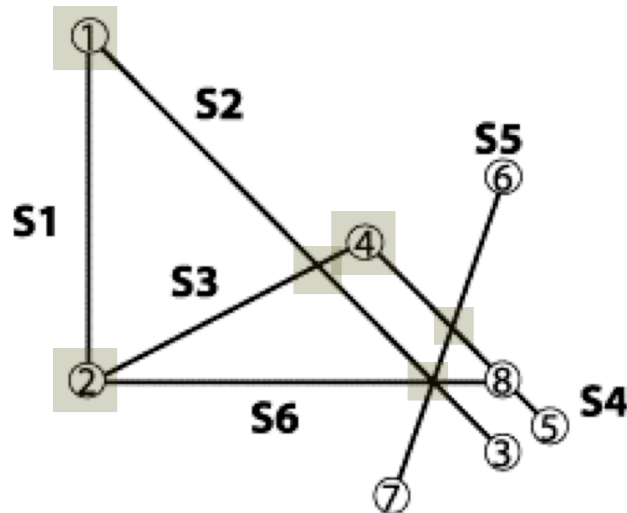
Event Queue



Report (0,13)	S1, S2
Report (8,7)	S3, S4
Report (6.6,6.3)	S2, S3
Report (10.5,4.5)	S4, S5
Report (0,3)	S1, S3, S6

Example Execution

Line State



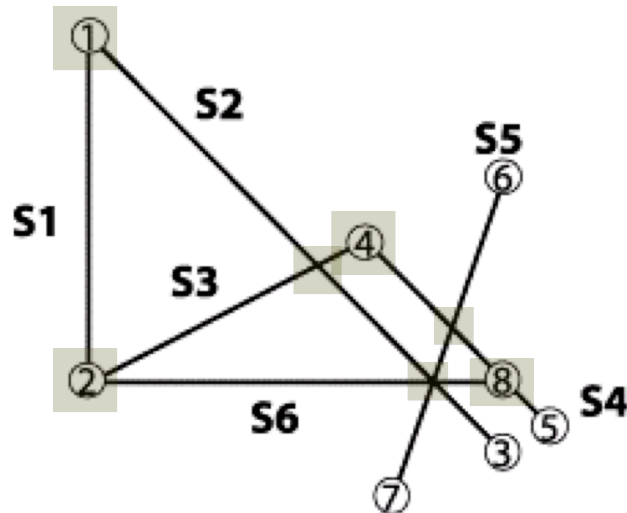
Event Queue



Report (0,13)	S1, S2
Report (8,7)	S3, S4
Report (6.6,6.3)	S2, S3
Report (10.5,4.5)	S4, S5
Report (0,3)	S1, S3, S6
Report (10,3)	S2, S5, S6

Example Execution

Line State

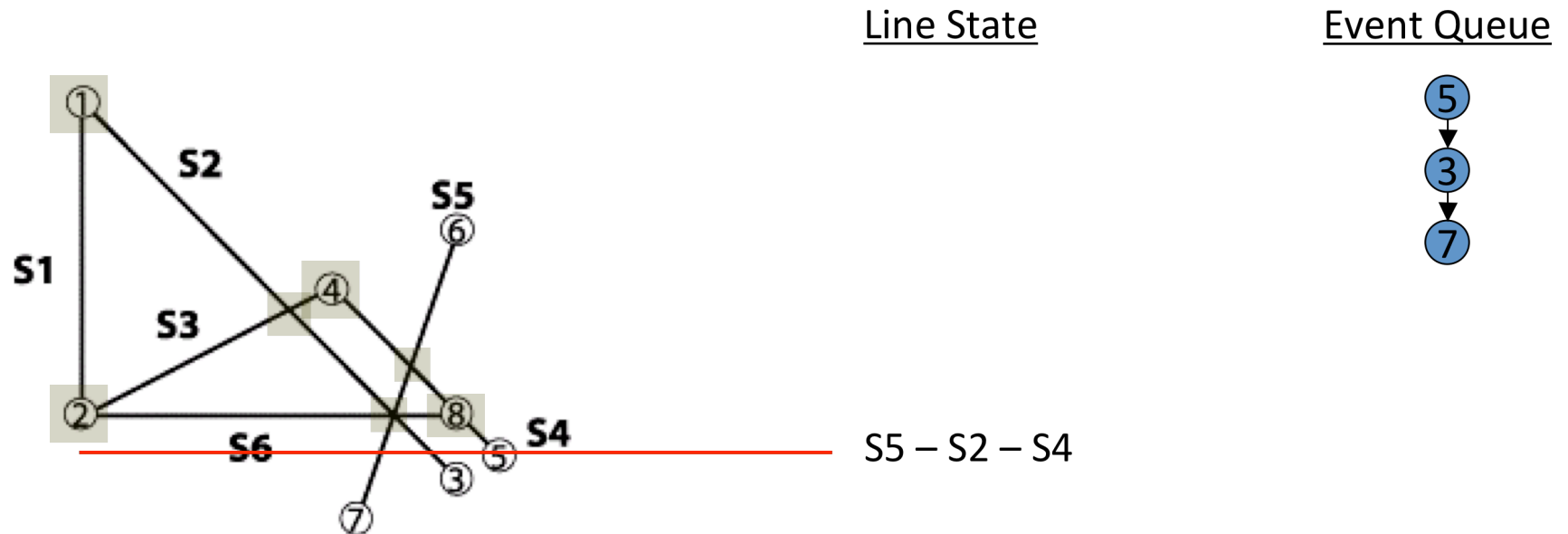


Event Queue



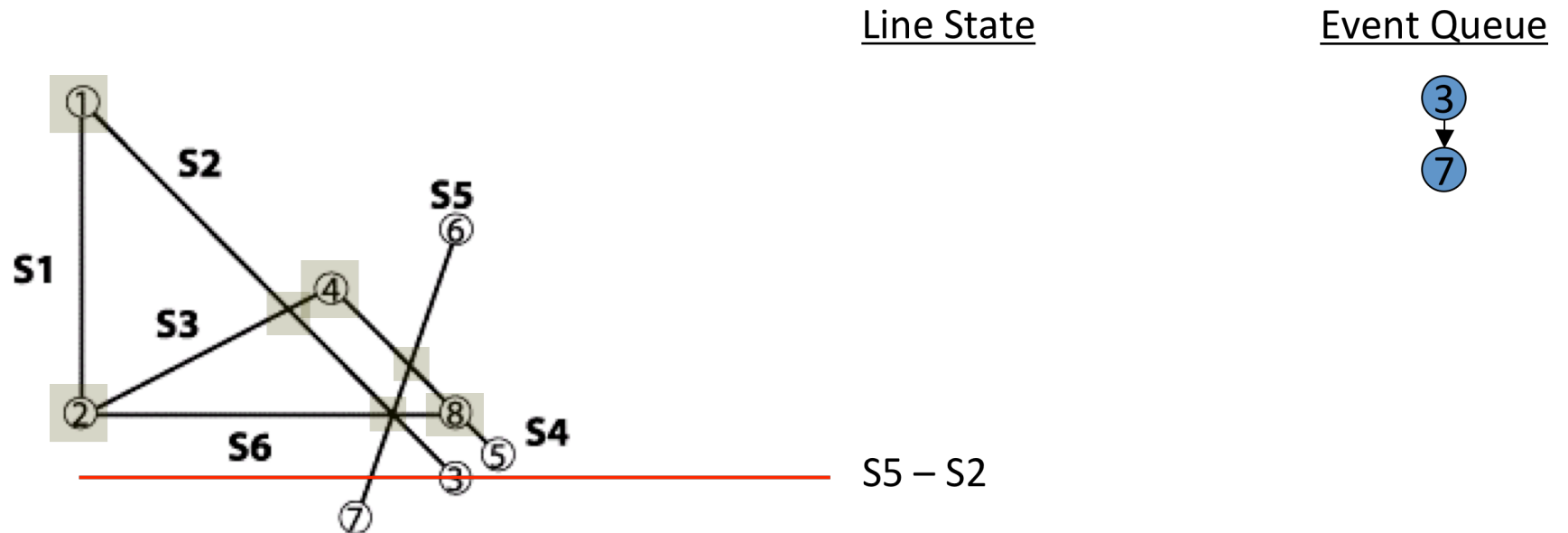
Report (0,13)	S1, S2
Report (8,7)	S3, S4
Report (6.6,6.3)	S2, S3
Report (10.5,4.5)	S4, S5
Report (0,3)	S1, S3, S6
Report (10,3)	S2, S5, S6
Report (12,3)	S4, S6

Example Execution



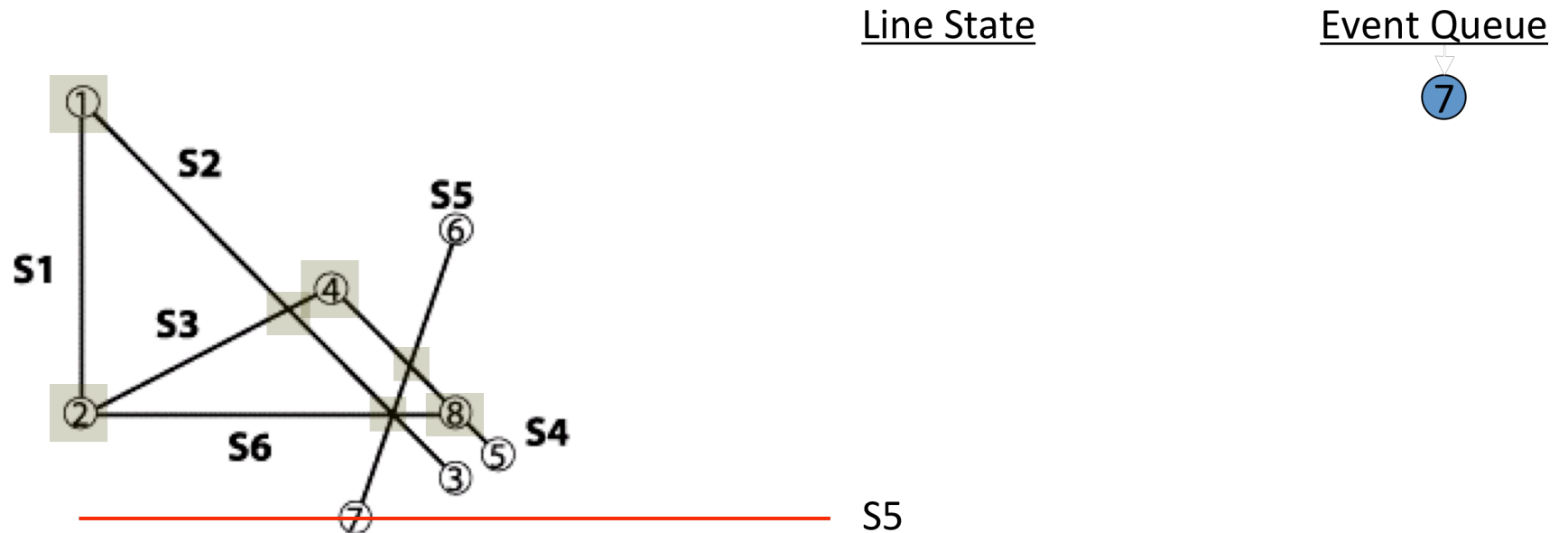
Report (0,13)	S1, S2
Report (8,7)	S3, S4
Report (6.6,6.3)	S2, S3
Report (10.5,4.5)	S4, S5
Report (0,3)	S1, S3, S6
Report (10,3)	S2, S5, S6
Report (12,3)	S4, S6

Example Execution



Report (0,13)	S1, S2
Report (8,7)	S3, S4
Report (6.6,6.3)	S2, S3
Report (10.5,4.5)	S4, S5
Report (0,3)	S1, S3, S6
Report (10,3)	S2, S5, S6
Report (12,3)	S4, S6

Example Execution

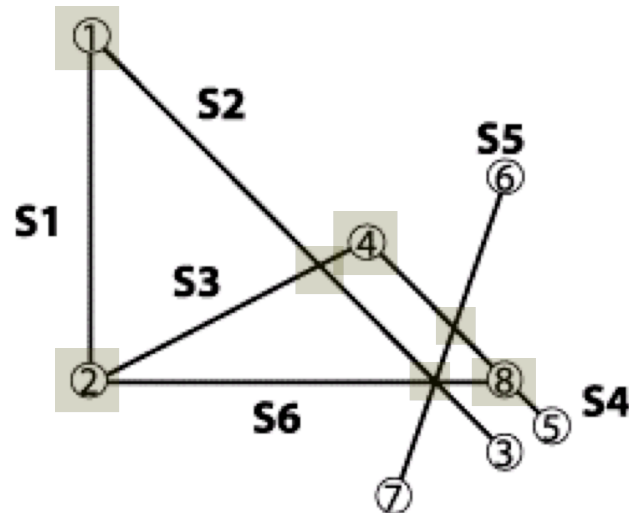


Report (0,13)	S1, S2
Report (8,7)	S3, S4
Report (6.6,6.3)	S2, S3
Report (10.5,4.5)	S4, S5
Report (0,3)	S1, S3, S6
Report (10,3)	S2, S5, S6
Report (12,3)	S4, S6

Example Execution

Line State

Event Queue



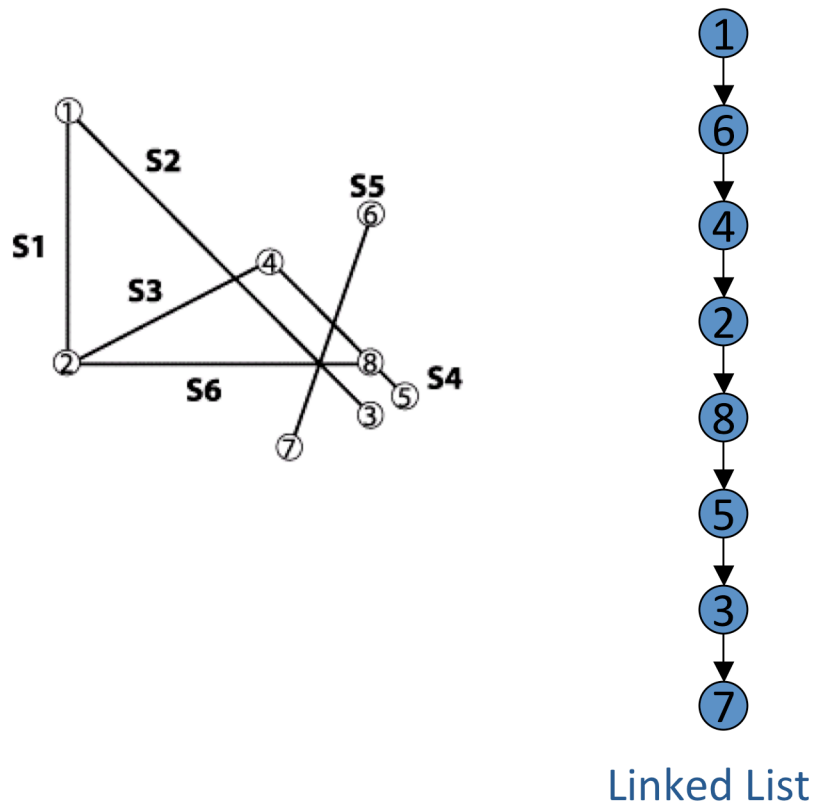
Report (0,13)	S1, S2
Report (8,7)	S3, S4
Report (6.6,6.3)	S2, S3
Report (10.5,4.5)	S4, S5
Report (0,3)	S1, S3, S6
Report (10,3)	S2, S5, S6
Report (12,3)	S4, S6

Tasks

- Must Sweep Line from top to bottom
 - Process each line exactly once
 - Sub-task: Represent Line State
- Report Intersections
 - Sub-task: Ensure accuracy of report
 - Miss no intersection point
 - Report each intersection point exactly once
 - Consider end points in computation

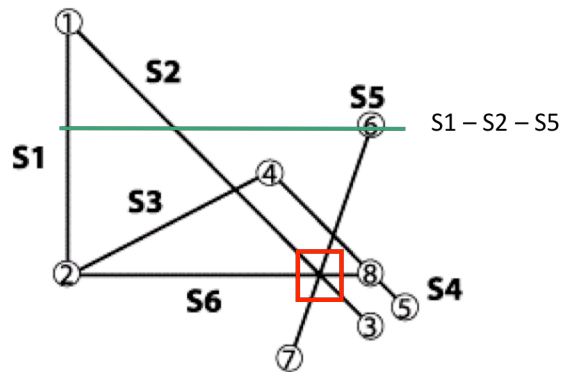


Initial Step: Create Event Queue



$2n+k$ event points in queue. Best performance: $O((n+k)^2)$

Initial Step: Create Event Queue



Linked List

GetMin: constant time

Contains: We must probe event queue to see if point is already present

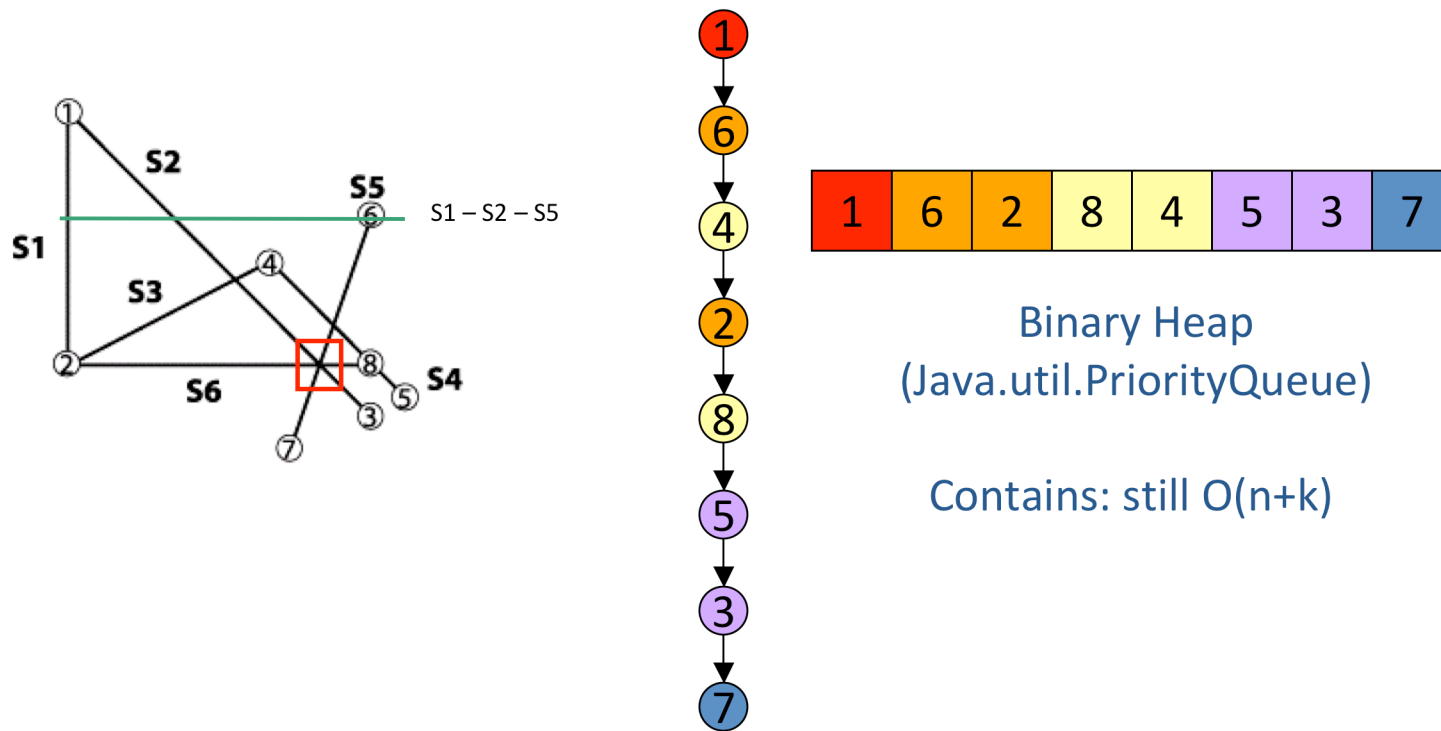
If so, merge (S2,S5) as intersecting points

whoops

In Linked List, this is $O(n+k)$

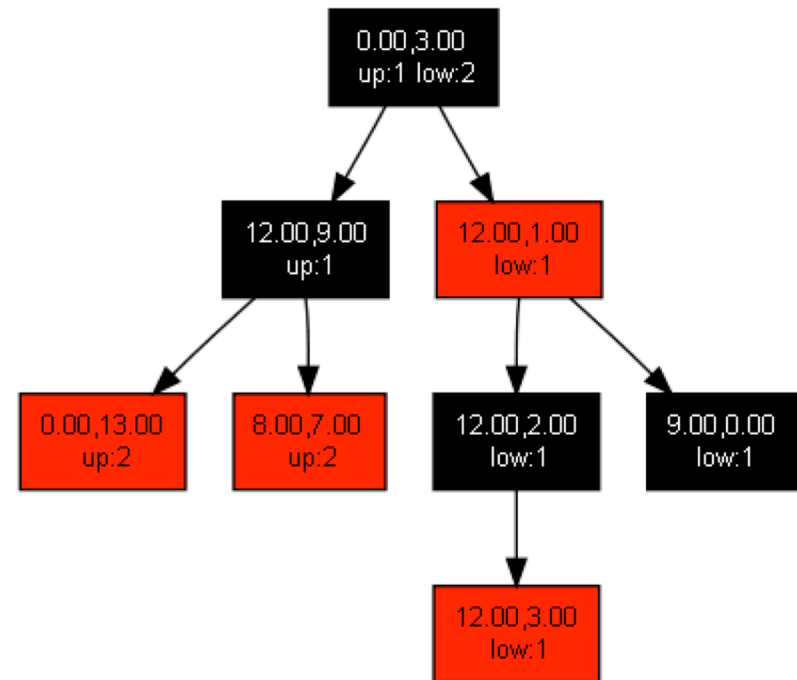
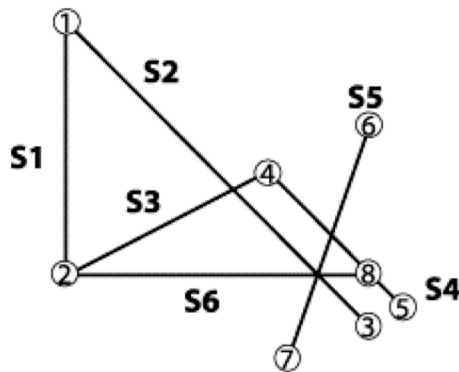
$2n+k$ event points in queue. Best performance: $O((n+k)^2)$

Initial Step: Create Event Queue



$2n+k$ event points in queue. Best performance: $O((n+k)^2)$

Initial Step: Create Event Queue

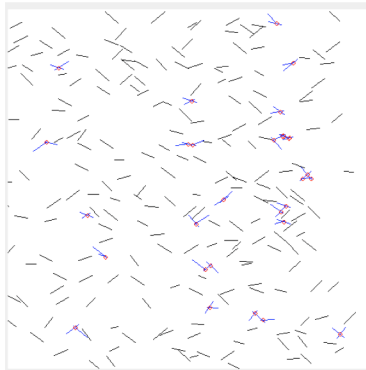


Event Queue stored as Balanced Binary Tree.
Ordered from top to bottom.

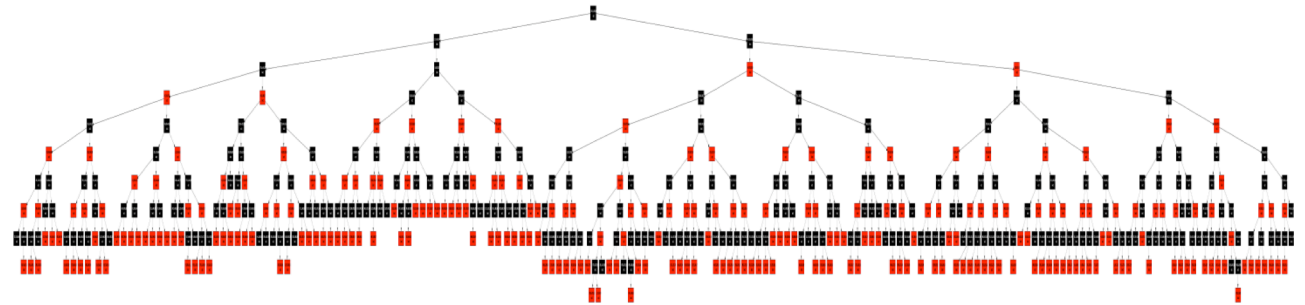
GetMin() and Probe() are $O(\log(n+k))$

Total so far: $O((2n+k) * \log(n+k)) = O((n+k) * \log n)$

Does this scale?



n=256
k=28



Longest path to leaf is 10

Final Task: Handle Event Point

- For each minimum Event Point ep
 - Find [left, right] in LineState
 - Compute and report these intersections
 - Update Line State
 - Delete all segments in [left, right]
 - Advance Sweep Point
 - Re-insert $ep.upper$ s & $ep.intersections$
 - Add to EventQueue new intersections
 - But only if they occur below sweep line

Line State Representation

- Balanced Binary Tree used again
 - Special Structure: Only leaves contain segments.
 - Internal nodes maintain pointers to leaves
 - maintenance of LineState is $O(\log (n+k))$

LineSegment Intersection

- Analysis
 - Resulting performance: $O((n+k) * \log n)$
 - Lots of special cases
 - Horizontal Lines, Vertical lines
 - Intersecting end points, etc...
- Testing code
 - Your basic nightmare...