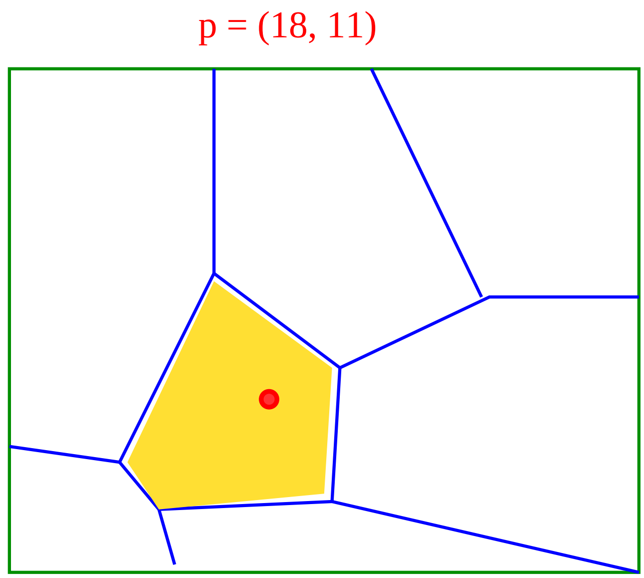
Point Location

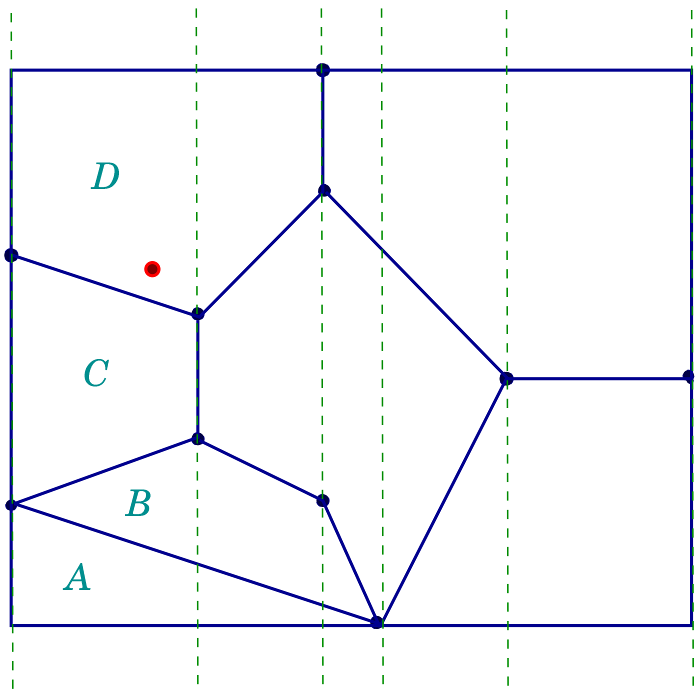
- **Preprocess a planar, polygonal subdivision for point location queries.**



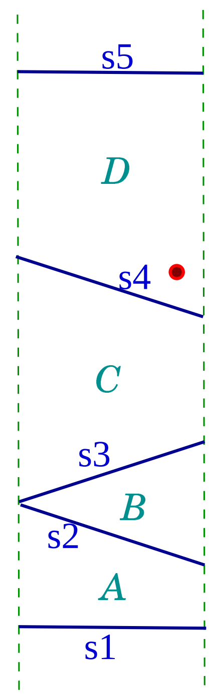
- **Input is a subdivision S of complexity n, say, number of edges.**
- **Build a data structure on S so that for a query point $p = (x, y)$, we can find the face containing p fast.**
- **Important metrics: space and query complexity.**

The Slab Method

- **Draw a vertical line through each vertex. This decomposes the plane into slabs.**
- **In each slab, the vertical order of line segments remains constant.**



Partition into slabs

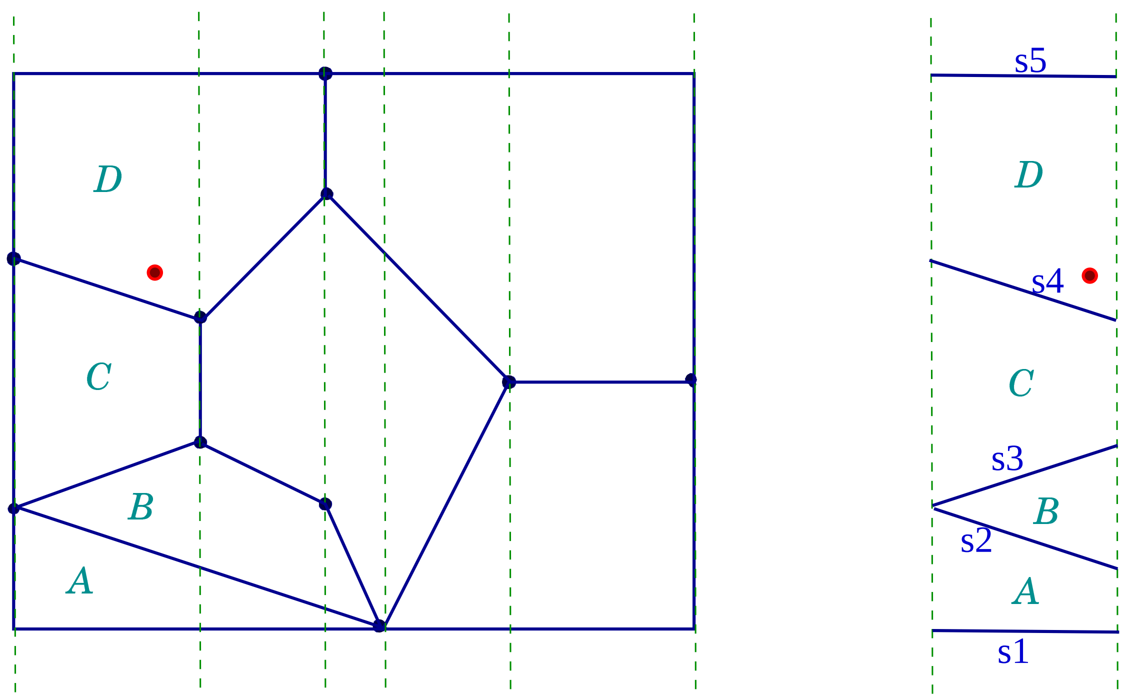


Slab 1

- **If we know which slab $p = (x, y)$ lies, we can perform a binary search, using the sorted order of segments.**

The Slab Method

- **To find which slab contains p, we perform a binary search on x, among slab boundaries.**
- **A second binary search in the slab determines the face containing p.**



Partition into slabs

Slab 1

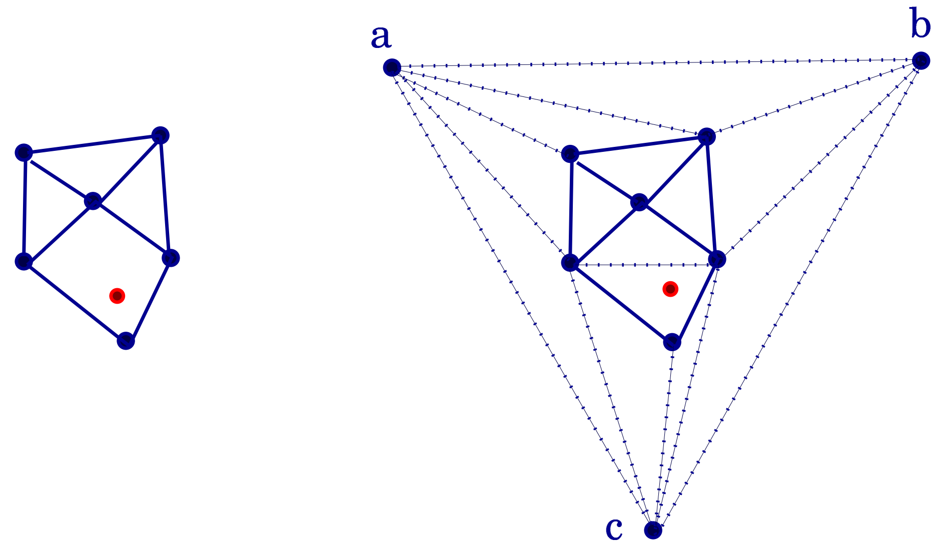
- **Thus, the search complexity is $O(\log n)$.**
- **But the space complexity is $\Theta(n^2)$.**

Optimal Schemes

- **There are other schemes (kd-tree, quad-trees) that can perform point location reasonably well, they lack theoretical guarantees. Most have very bad worst-case performance.**
- **Finding an optimal scheme was challenging. Several schemes were developed in 70's that did either $O(\log n)$ query, but with $O(n \log n)$ space, or $O(\log^2 n)$ query with $O(n)$ space.**
- **Today, we will discuss an elegant and simple method that achieved optimality, $O(\log n)$ time and $O(n)$ space [D. Kirkpatrick '83].**
- **Kirkpatrick's scheme however involves large constant factors, which make it less attractive in practice.**
- **Later we will discuss a more practical, randomized optimal scheme.**

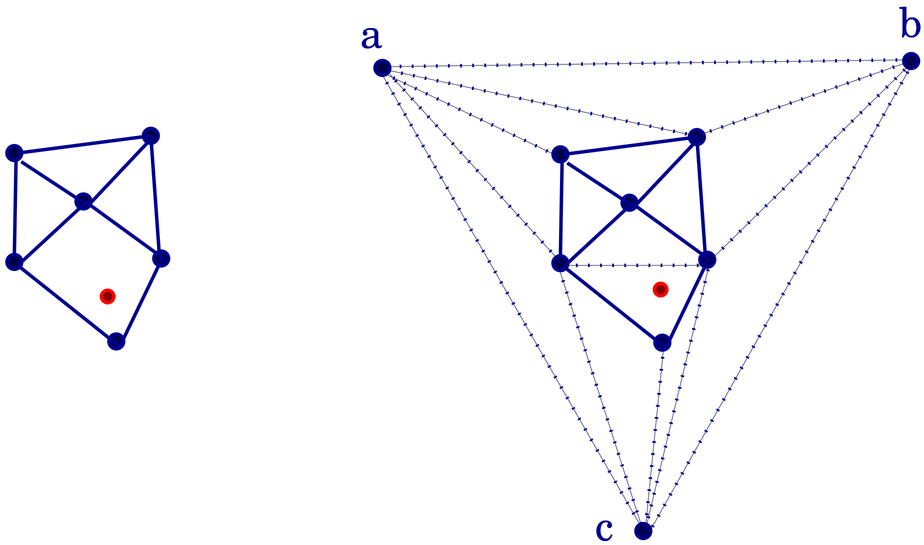
Kirkpatrick's Algorithm

- **Start with the assumption that planar subdivision is a triangulation.**
- **If not, triangulate each face, and label each triangular face with the same label as the original containing face.**
- **If the outer face is not a triangle, compute the convex hull, and triangulate the pockets between the subdivision and CH.**
- **Now put a large triangle *abc* around the subdivision, and triangulate the space between the two.**



Modifying Subdivision

- **By Euler'e formula, the final size of this triangulated subdivision is still $O(n)$.**
- **This transformation from S to triangulation can be performed in $O(n \log n)$ time.**



- **If we can find the triangle containing p, we will know the original subdivision face containing p.**

Hierarchical Method

- **Kirkpatrick's method is hierarchical: produce a sequence of increasingly coarser triangulations, so that the last one has $O(1)$ size.**

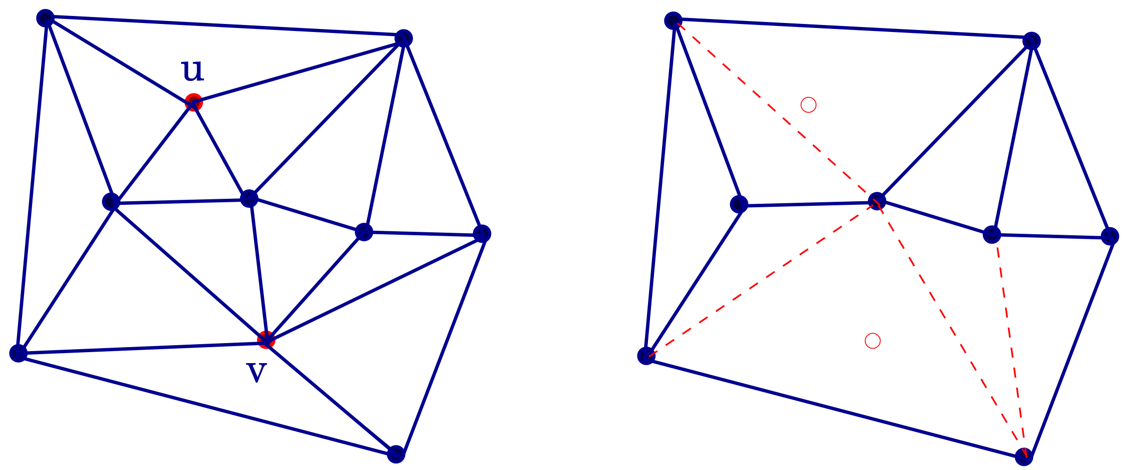
- **Sequence of triangulations $T_0, T_1, \ldots, T_k$, with following properties:**

  1. **T_0 is the initial triangulation, and T_k is just the outer triangle abc.**
  2. **k is $O(\log n)$.**
  3. **Each triangle in T_{i+1} overlaps $O(1)$ triangles of T_i.**

- **Let us first discuss how to construct this sequence of triangulations.**

Building the Sequence

- **Main idea is to delete some vertices of T_i.**
- **Their deletion creates holes, which we re-triangulate.**



Vertex deletion and re–triangulation

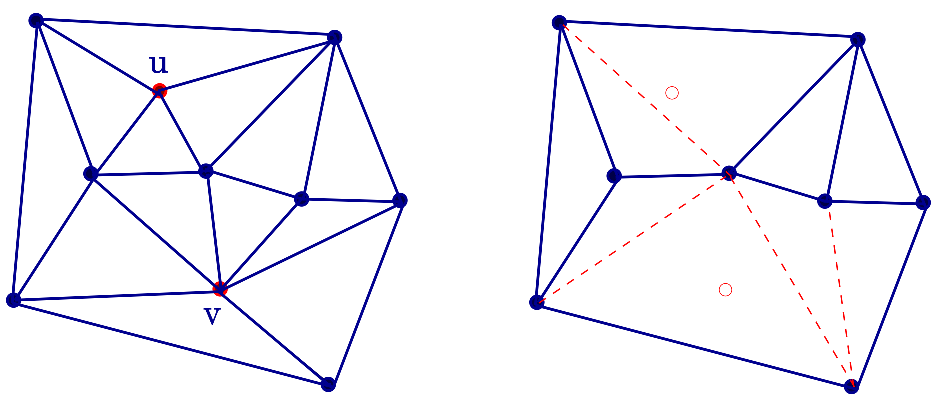
- **We want to go from $O(n)$ size subdivision T_0 to $O(1)$ size subdivision T_k in $O(\log n)$ steps.**
- **Thus, we need to delete a constant fraction of vertices from T_i.**
- **A critical condition is to ensure each new triangle in T_{i+1} overlaps with $O(1)$ triangles of T_i.**

Independent Sets

- **Suppose we want to go from T_i to T_{i+1}, by deleting some points.**
- **Kirkpatrick's choice of points to be deleted had the following two properties:**

[**Constant Degree**] **Each deletion candidate has $O(1)$ degree in graph T_i.**

  - **If p has degree d, then deleting p leaves a hole that can be filled with $d - 2$ triangles.**
  - **When we re-triangulate the hole, each new triangle can overlap at most d original triangles in T_i.**

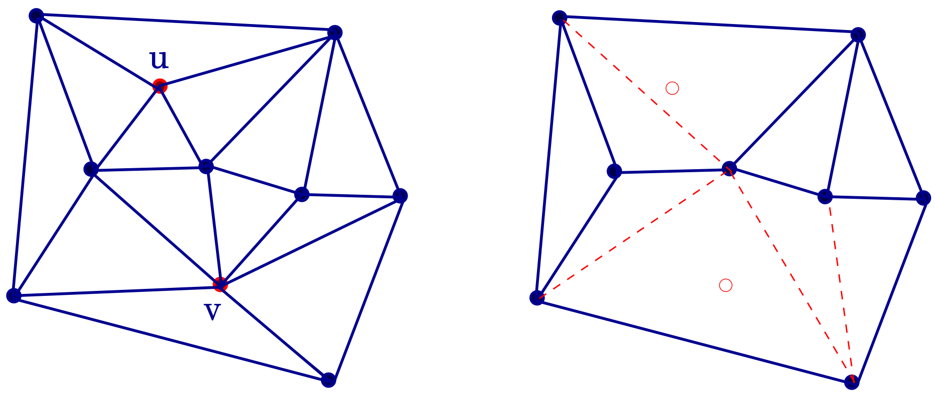


Vertex deletion and re–triangulation

Independent Sets

[Independent Sets] No two deletion candidates are adjacent.

- This makes re-triangulation easier; each hole handled independently.



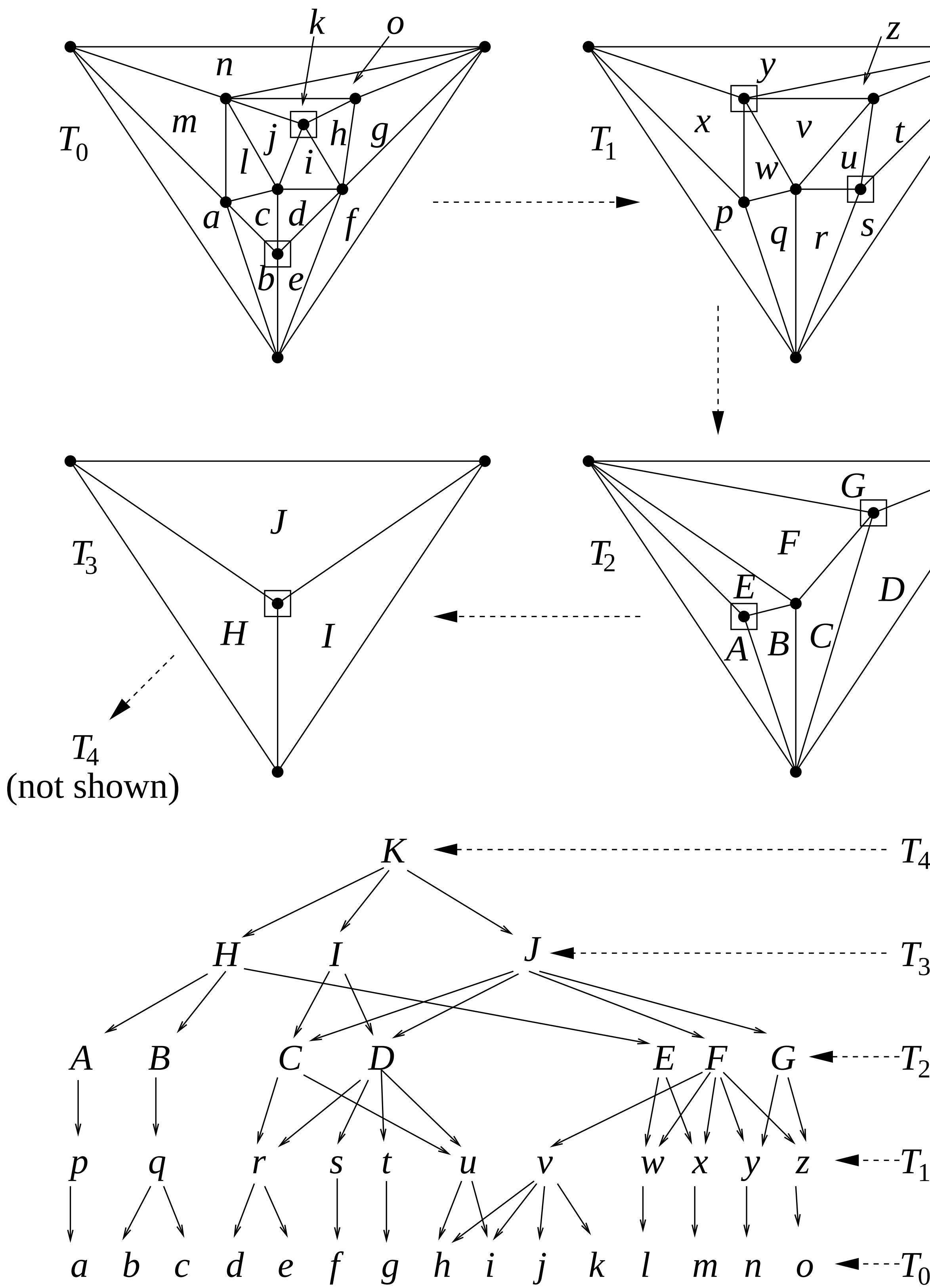
Vertex deletion and re–triangulation

I.S. Lemma

Lemma: **Every planar graph on n vertices contains an independent vertex set of size $n/18$ in which each vertex has degree at most 8. The set can be found in $O(n)$ time.**

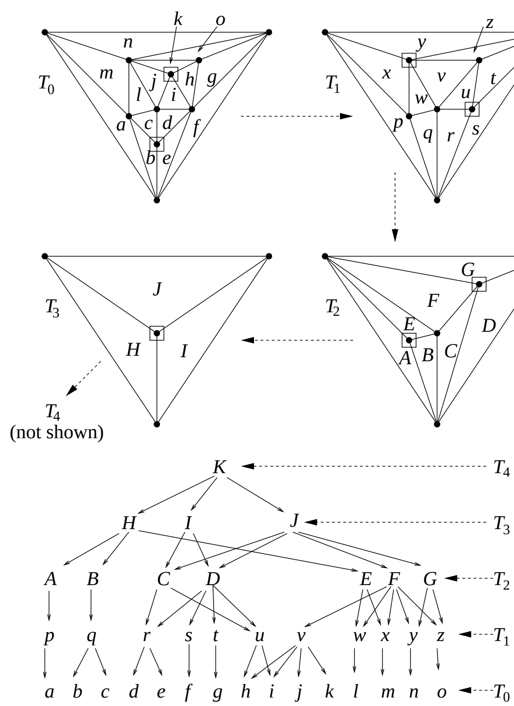
- **We prove this later. Let's use this now to build the triangle hierarchy, and show how to perform point location.**
- **Start with T_0. Select an ind set S_0 of size $n/18$, with max degree 8. Never pick a, b, c, the outer triangle's vertices.**
- **Remove the vertices of S_0, and re-triangulate the holes.**
- **Label the new triangulation T_1. It has at most $\frac{17}{18}n$ vertices. Recursively build the hierarchy, until T_k is reduced to abc.**
- **The number of vertices drops by $17/18$ each time, so the depth of hierarchy is** $k = \log_{18/17} n \approx 12 \log n$

Illustration

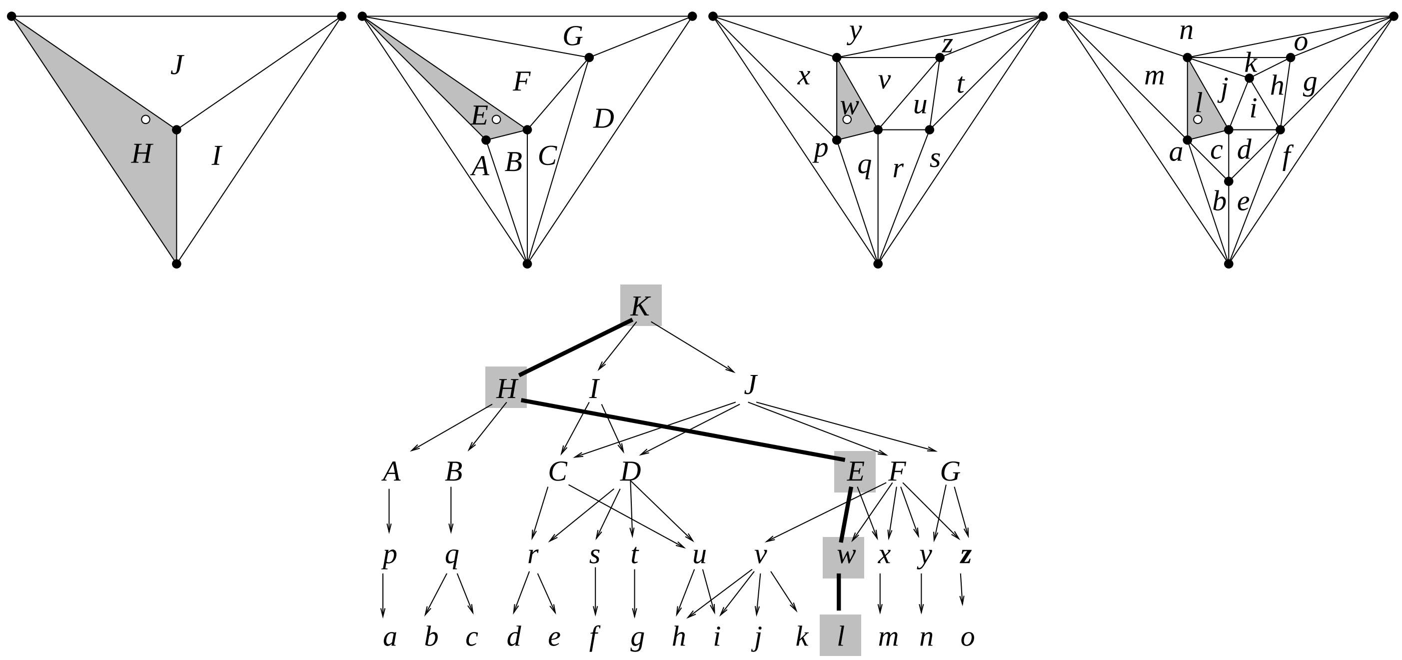


The Data Structure

- **Modeled as a DAG: the root corresponds to single triangle T_k.**
- **The nodes at next level are triangles of T_{k-1}.**
- **Each node for a triangle in T_{i+1} has pointers to all triangles of T_i that it overlaps.**
- **To locate a point p, start at the root. If p outside T_k, we are done (exterior face). Otherwise, set $t = T_k$, as the triangle at current level containing p.**

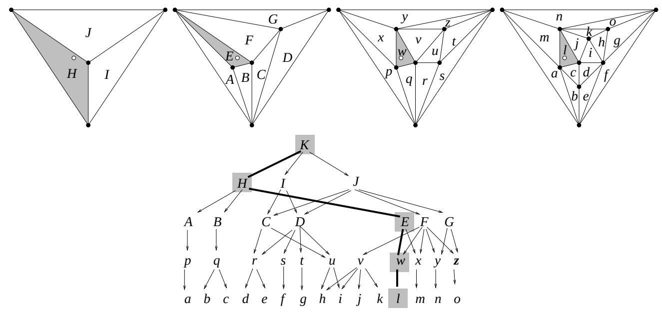


The Search



- **Check each triangle of T_{k-1} that overlaps with t—at most 6 such triangles. Update t, and descend the structure until we reach T_0.**
- **Output t.**

Analysis



- **Search time is** $O(\log n)$**—there are** $O(\log n)$ **levels, and it takes** $O(1)$ **time to move from level** i **to level** $i-1$**.**
- **Space complexity requires summing up the sizes of all the triangulations.**
- **Since each triangulation is a planar graph, it is sufficient to count the number of vertices.**
- **The total number of vertices in all triangulations is**

$$n\left(1 + (17/18) + (17/18)^2 + (17/18)^3 + \cdots\right) \leq 18n.$$

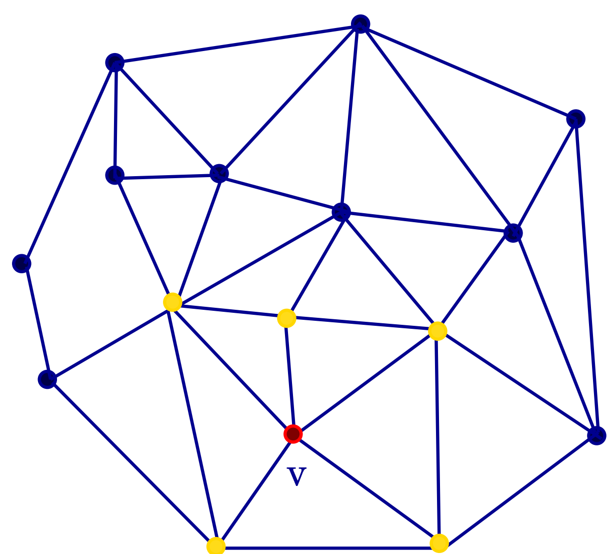
- **Kirkpatrick structure has** $O(n)$ **space and** $O(\log n)$ **query time.**

Finding I.S.

- We describe an algorithm for finding the independent set with desired properties.
- Mark all nodes of degree ≥ 9.
- While there is an unmarked node, do

  1. Choose an unmarked node v.
  2. Add v to IS.
  3. Mark v and all its neighbors.

- Algorithm can be implemented in $O(n)$ time—keep unmarked vertices in list, and representing T so that neighbors can be found in $O(1)$ time.



I.S. Analysis

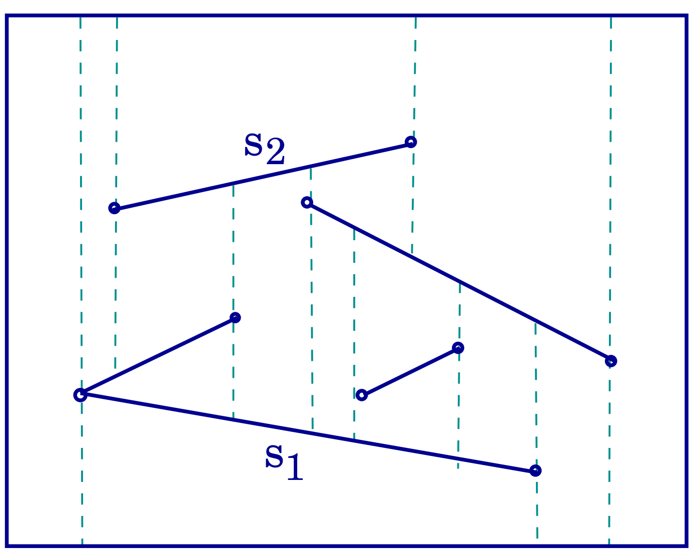
- **Existence of large size, low degree IS follows from Euler's formula for planar graphs.**
- **A triangulated planar graph on n vertices has $e = 3n - 6$ edges.**
- **Summing over the vertex degrees, we get**

$$\sum_v deg(v) = 2e = 6n - 12 < 6n.$$

- **We now claim that at least $n/2$ vertices have degree ≤ 8.**
- **Suppose otherwise. Then $n/2$ vertices all have degree ≥ 9. The remaining have degree at least 3. (Why?)**
- **Thus, the sum of degrees will be at least $9\frac{n}{2} + 3\frac{n}{2} = 6n$, which contradicts the degree bound above.**
- **So, in the beginning, at least $n/2$ nodes are unmarked. Each chosen v marks at most 8 other nodes (total 9 counting itself.)**
- **Thus, the node selection step can be repeated at least $n/18$ times.**
- **So, there is a I.S. of size $\geq n/18$, where each node has degree ≤ 8.**

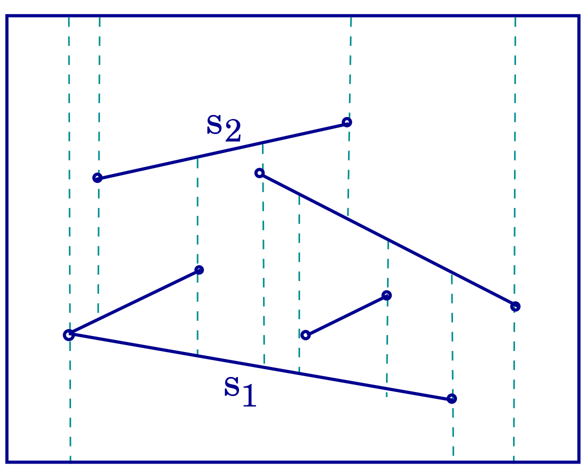
Trapezoidal Maps

- **A randomized point location scheme, with (expected) query $O(\log n)$, space $O(n)$, and construction time $O(n \log n)$.**
- **The expectation does not depend on the polygonal subdivision. The bounds holds for any subdivision.**
- **It appears simpler to implement, and its constant factors are better than Kirkpatrick's.**
- **The algorithm is based on trapezoidal maps, or decompositions, also encountered earlier in triangulation.**



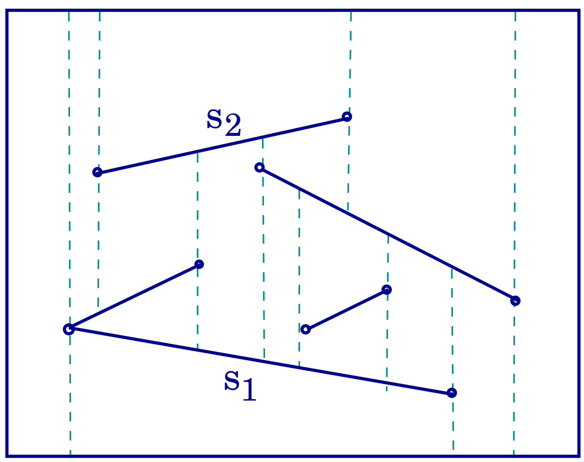
Trapezoidal Maps

- **Input a set of non-intersecting line segments $S = \{s_1, s_2, \ldots, s_n\}$.**
- **Query: given point p, report the segment directly above p.**
- **The region label can be easily encoded into the line segments.**
- **Map is created by shooting a ray vertically from each vertex, up and down, until a segment is hit.**
- **In order to avoid degeneracies, assume that no segment is vertical.**
- **The resulting rays plus the segments define the trapezoidal map.**



Trapezoidal Maps

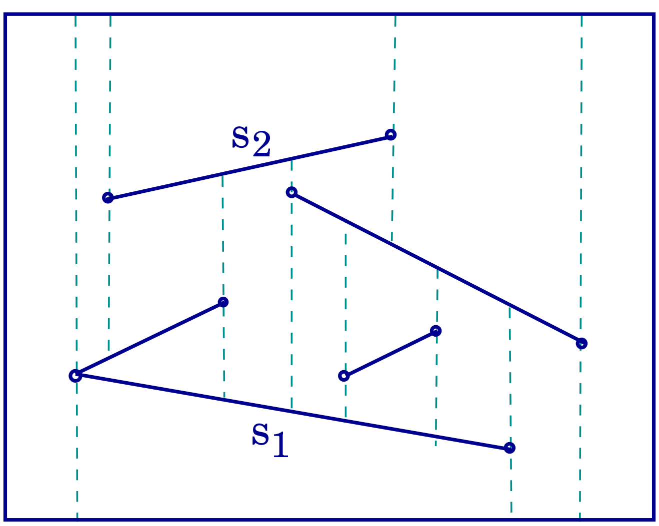
- **Enclose S into a bounding box to avoid infinite rays.**
- **All faces of the subdivision are trapezoids, with vertical sides.**
- **Size Claim: If S has n segments, the map has at most $6n + 4$ vertices and $3n + 1$ traps.**



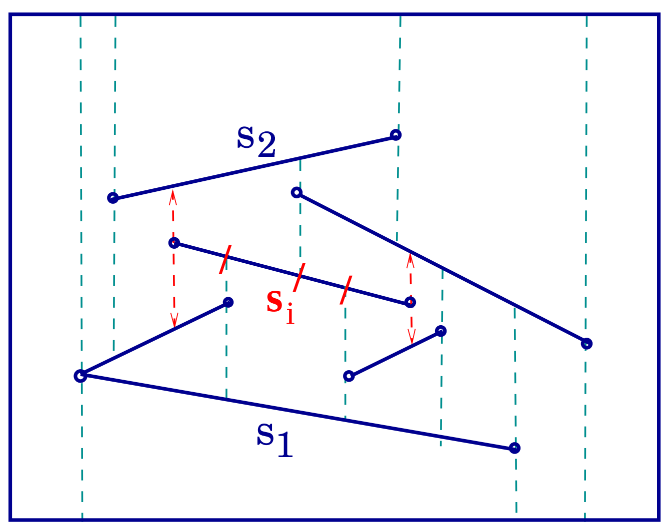
- **Each vertex shoots one ray, each resulting in two new vertices, so at most $6n$ vertices, plus 4 for the outer box.**
- **The left boundary of each trapezoid is defined by a segment endpoint, or lower left corner of enclosing box.**
- **The corner of box acts as leftpoint for one trap; the right endpoint of any segment also for one trap; and left endpoint of any segment for at most 2 trapezoids. So total of $3n + 1$.**

Construction

- **Plane sweep possible, but not helpful for point location.**
- **Instead we use randomized incremental construction.**
- **Historically, invented for randomized segment intersection. Point location an intermediate problem.**
- **Start with outer box, one trapezoid. Then, add one segment at a time, in an arbitrary, not sorted, order.**



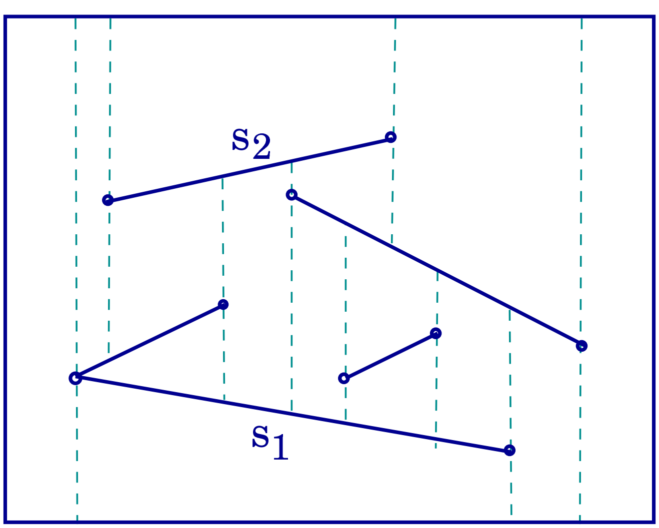
Before



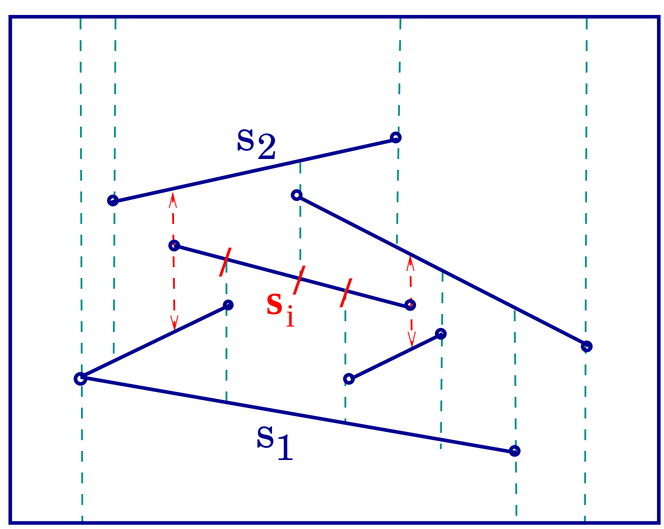
After inserting s

Construction

- **Let $S_i = \{s_1, s_2, \ldots, s_i\}$ be first i segments, and $\mathcal{T}_i$ be their trapezoidal map.**
- **Suppose $\mathcal{T}_{i-1}$ built, and we add s_i.**
- **Find the trapezoid containing the left endpoint of s_i. Defer for now: this is point location.**
- **Walk through $\mathcal{T}_{i-1}$, identifying trapezoids that are cut. Then, "fix them up".**
- **Fixing up means, shoot rays from left and right endpoints of s_i, and trim the earlier rays that are cut by s_i.**



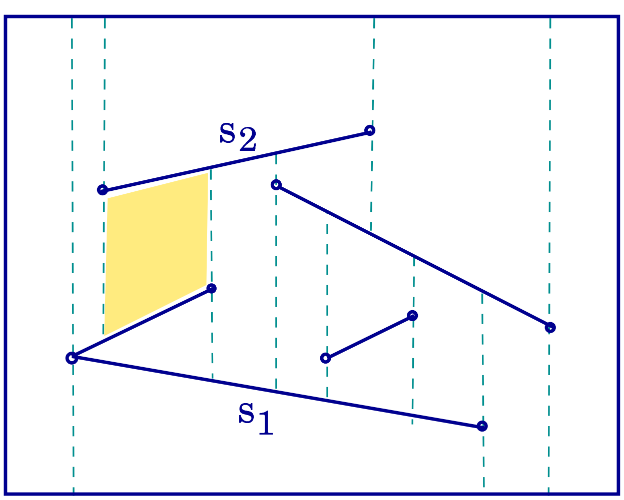
Before



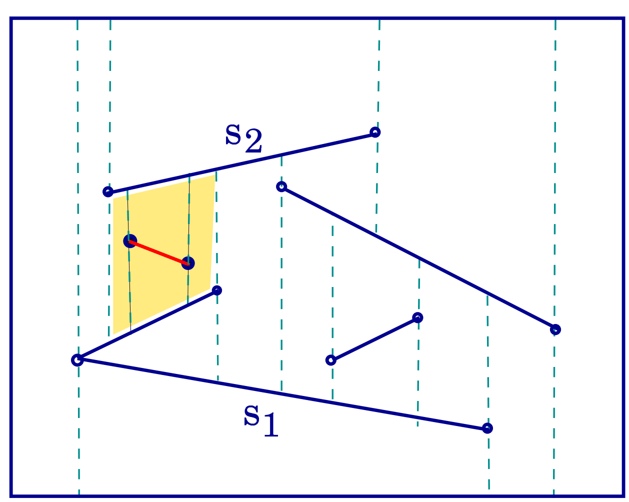
After inserting s

Analysis

- **Observation: Final structure of trap map does not depend on the order of segments. (Why?)**
- **Claim: Ignoring point location, segment i's insertion takes $O(k_i)$ time if k_i new trapezoids created.**
- **Proof:**
  - **Each endpoint of s_i shoots two rays.**
  - **Additionally, suppose s_i interrupts K existing ray shots, so total of $K + 4$ rays need processing.**
  - **If $K = 0$, we get exactly 4 new trapezoids.**
  - **For each interrupted ray shot, a new trapezoid created.**
  - **With DCEL, update takes $O(1)$ per ray.**



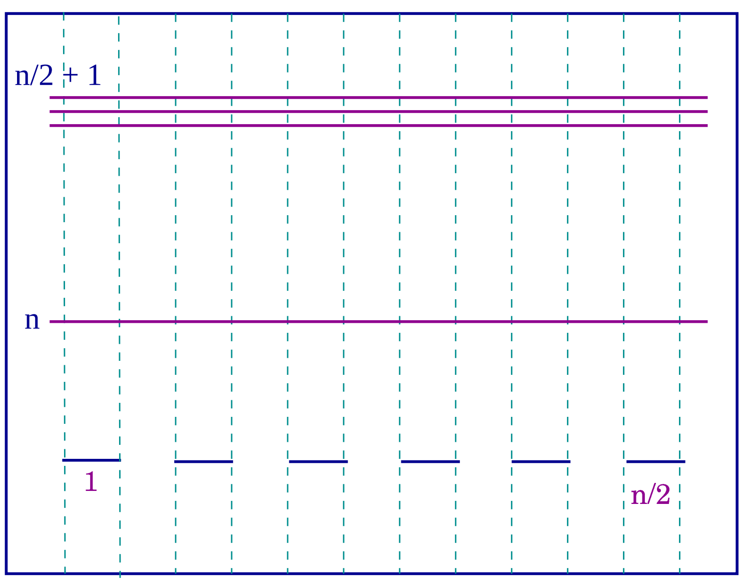
Before



After

Worst Case

- **In a worst-case, k_i can be $\Theta(i)$. This can happen for all i, making the worst-case run time $\sum_{i=1}^{n} i = \Theta(n^2)$.**
- **Using randomization, we prove that if segments are inserted in random order, then expected value of k_i is $O(1)$!**
- **So, for each segment s_i, the expected number of new trapezoids created is a constant.**
- **Figure below shows a worst-case example. How will randomization help?**

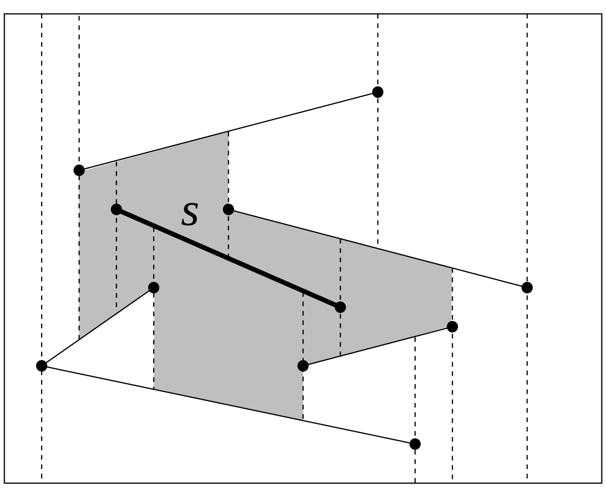


Randomization

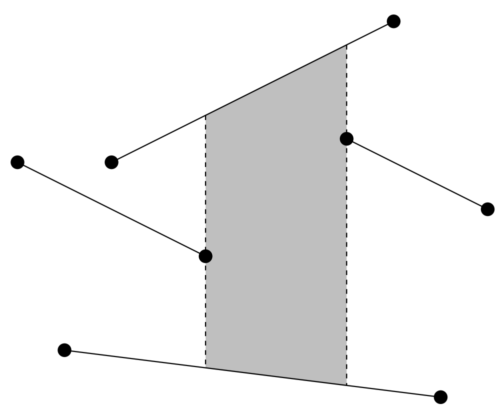
- **Theorem:** **Assume $s_1, s_2, \ldots, s_n$ is a random permutation. Then, $E[k_i] = O(1)$, where k_i trapezoids created upon s_i's insertion, and the expectation is over all permutations.**

- **Proof.**

  1. **Consider $\mathcal{T}_i$, the map after s_i's insertion.**
  2. **$\mathcal{T}_i$ does not depend on the order in which segments $s_1, \ldots, s_i$ were added.**
  3. **Reshuffle $s_1, \ldots, s_i$. What's the probability that a particular s was the last segment added?**
  4. **The probability is $1/i$.**
  5. **We want to compute the number of trapezoids that would have been created if s were the last segment.**



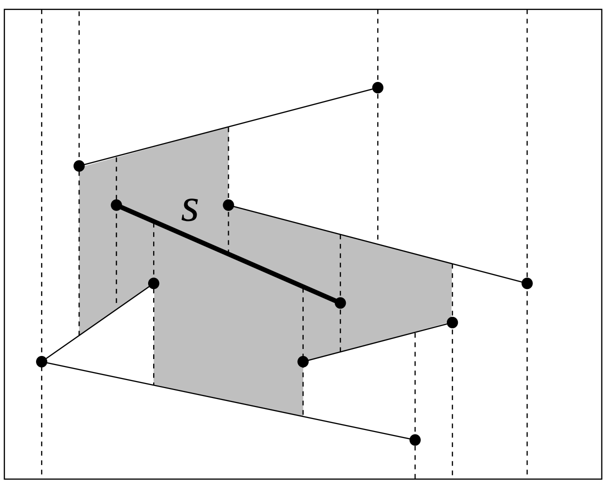
The trapezoids that depend on s



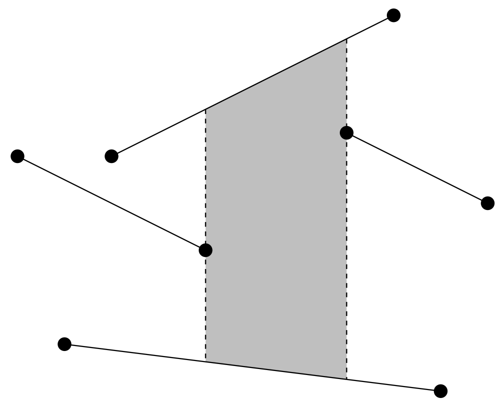
The segments that the trapezoid depends on.

Proof

- **Say trapezoid Δ depends on s if Δ would be created by s if s were added last.**
- **Want to count trapezoids that depend on each segment, and then find the average over all segments.**
- **Define $\delta(\Delta, s) = 1$ if Δ depends on s; otherwise, $\delta(\Delta, s) = 0$.**



The trapezoids that depend on s



The segments that the trapezoid depends on.

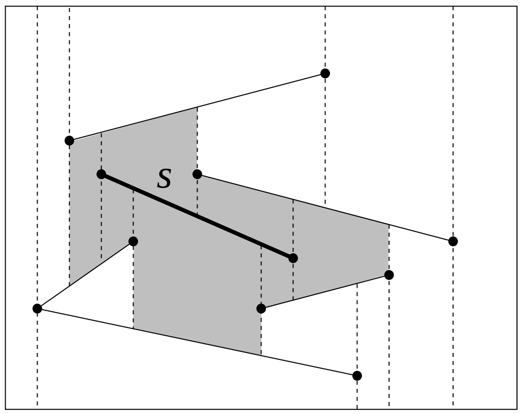
- **The expected complexity is**

$$E[k_i] = \frac{1}{i} \sum_{s \in S_i} \sum_{\Delta \in \mathcal{T}_i} \delta(\Delta, s)$$

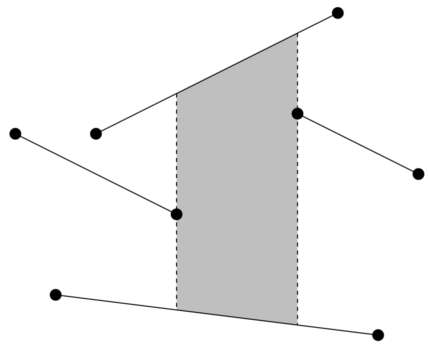
- **Some segments create a lot of trapezoids; others very few.**
- **Switch the order of summation:**

$$E[k_i] = \frac{1}{i} \sum_{\Delta \in \mathcal{T}_i} \sum_{s \in S_i} \delta(\Delta, s)$$

Proof



The trapezoids that depend on s



The segments that the trapezoid depends on.

- **Now we are counting number of segments each trapezoid depents on.**

$$E[k_i] = \frac{1}{i} \sum_{\Delta \in \mathcal{T}_i} \sum_{s \in S_i} \delta(\Delta, s)$$

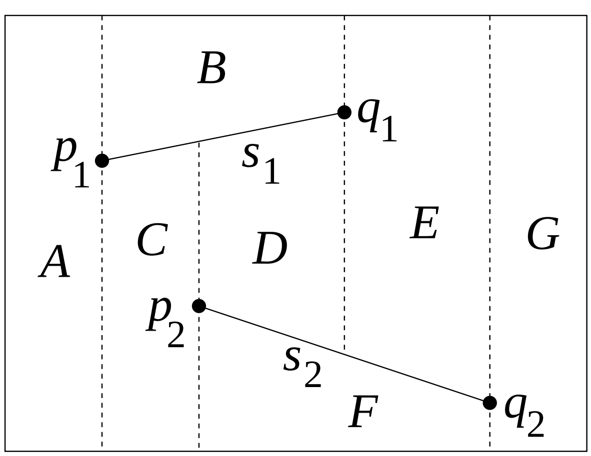
- **This is much easier—each Δ depends on at most 4 segments.**
- **Top and bottom of Δ defined by two segments; if either of them added last, then Δ comes into existence.**
- **Left and right sides defined by two segments endpoints, and if either one added last, Δ is created.**
- **Thus, $\sum_{s \in S_i} \delta(\Delta, s) \leq 4$.**
- **$\mathcal{T}_i$ has $O(i)$ trapezoids, so**

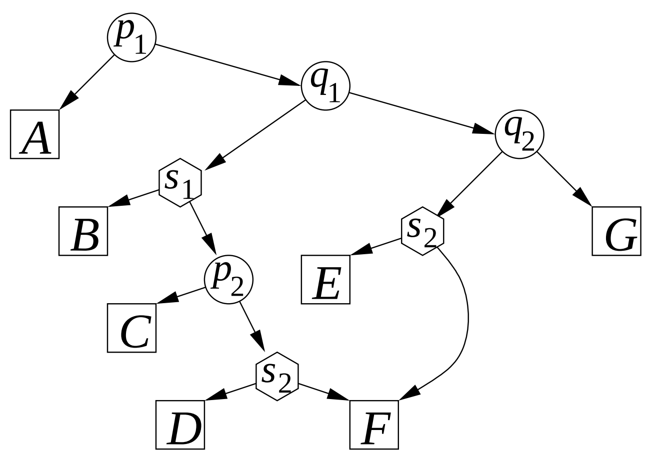
$$E[k_i] = \frac{1}{i} \sum_{\Delta \in \mathcal{T}_i} 4 = \frac{1}{i} 4 |\mathcal{T}_i| = \frac{1}{i} O(i) = O(1).$$

- **End of proof.**

Point Location

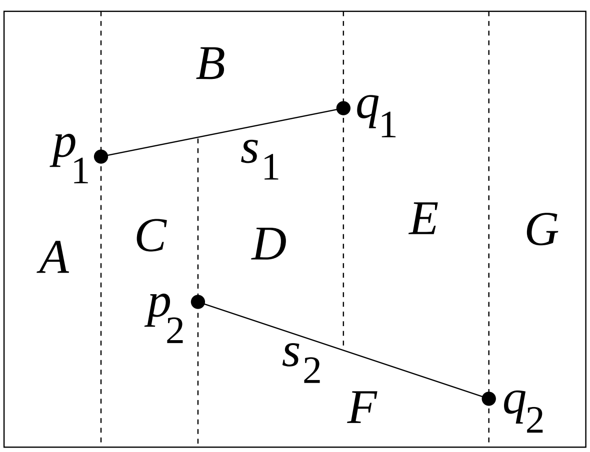
- **Like Kirkpatrick's, point location structure is a rooted directed acyclic graph.**
- **To query processor, it looks like a binary tree, but subtree may be shared.**
- **Tree has two types of nodes:**
  - x**-node: contains the** x**-coordinate of a segment endpoint. (Circle)**
  - y**-node: pointer to a segment. (Hexagon)**
- **A leaf for each trapzedoid.**

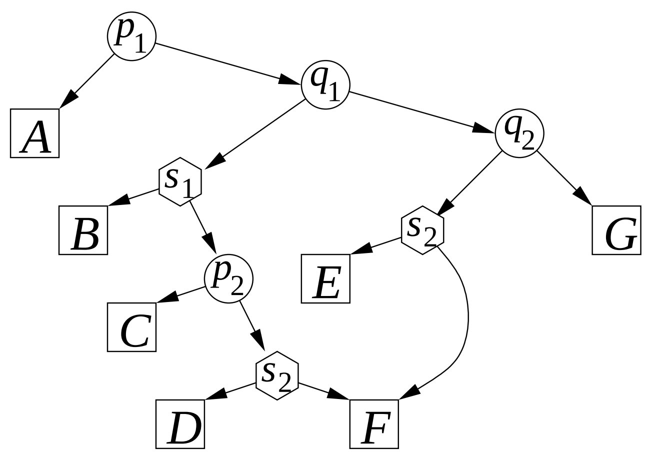




Point Location

- **Children of x-node correspond to points lying to the left and right of x coord.**
- **Children of y-node correspond to space below and above the segment.**
- **y-node searched only when query's x-coordinate is within segment's span.**
- **Example: query in region D.**

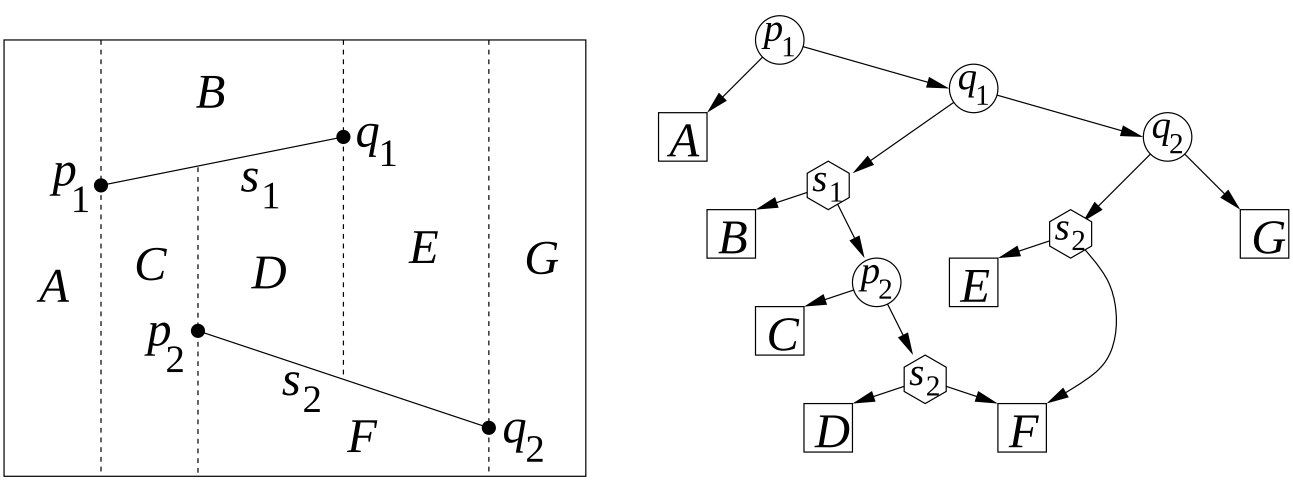




- **Encodes the trap decomposition, and enables point location during the construction as well.**

Building the Structure

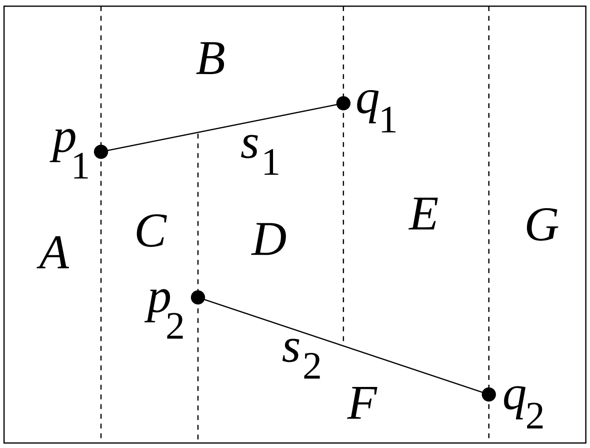
- Incremental construction, mirroring the trapezoidal map.
- When a segment s added, modify the tree to account for changes in trapezoids.
- Essentially, some leaves will be replaced by new subtrees.
- Like Kirkpatrick's, each old trapezoid will overlap $O(1)$ new trapezoids.

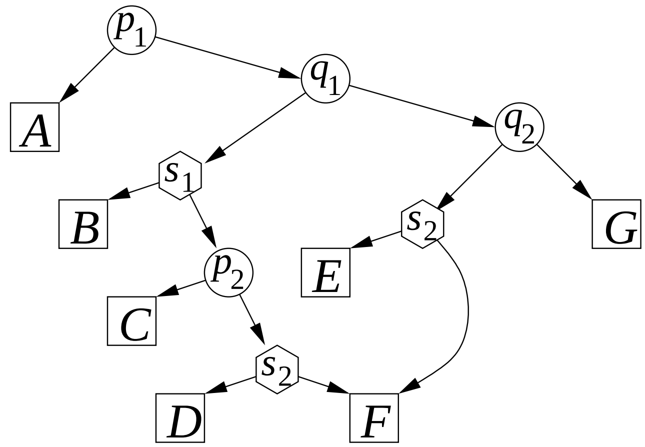


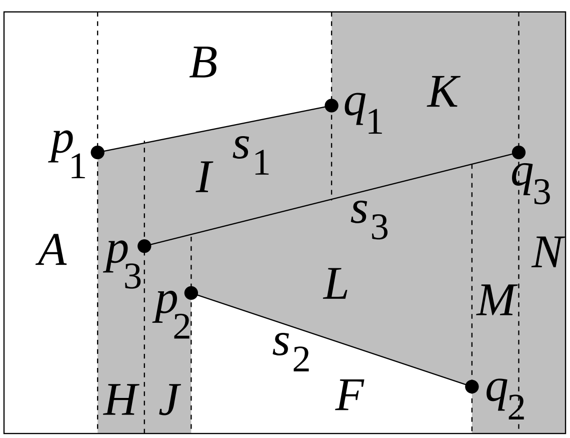
- Each trapezoid appears exactly once as a leaf. For instance, F.

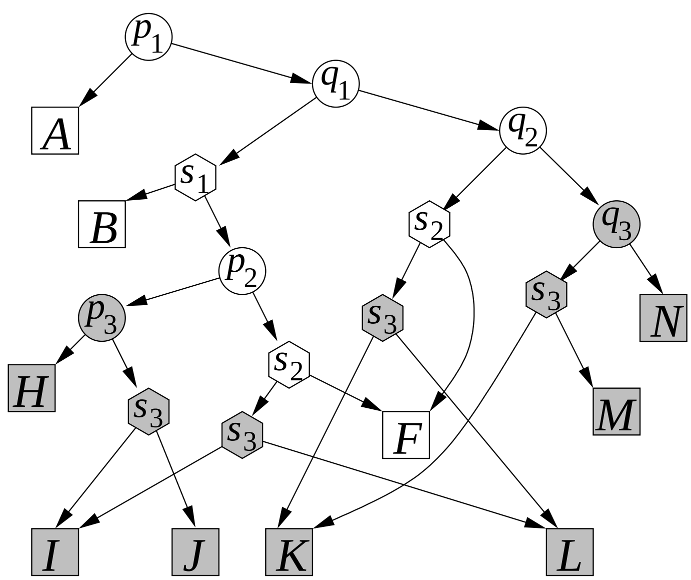
Adding a Segment

- **Consider adding segment s_3.**



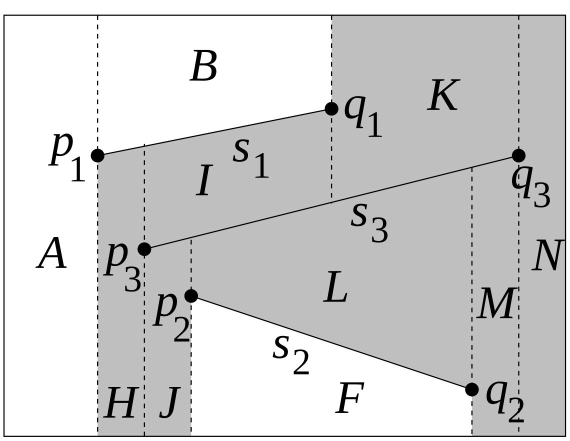


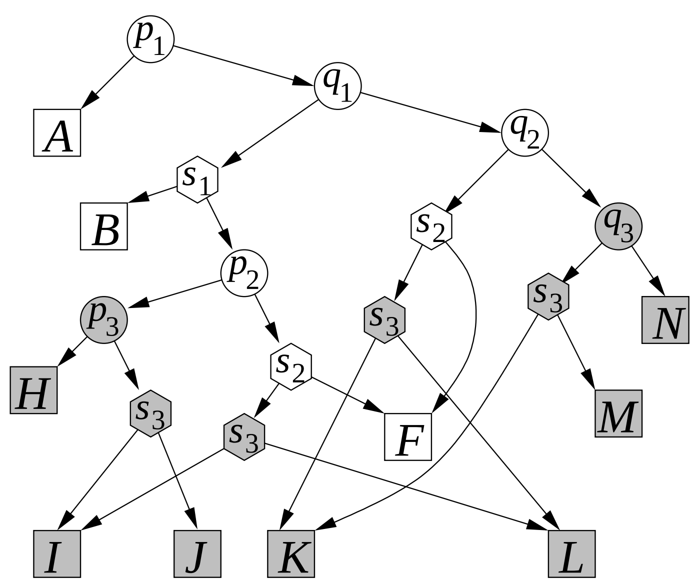




Adding a Segment

- Changes are highly local.
- If segment s passes entirely through an old trapezoid t, then t is replaced by two traps t', t''.
  - During search, we need to compare query point to s to decide above/below.
  - So, a new y-node added which is the parent of t' and t''.
- If an endpoint of s lies in t, then we add a x-node to decide left/right and a y-node for the segment.



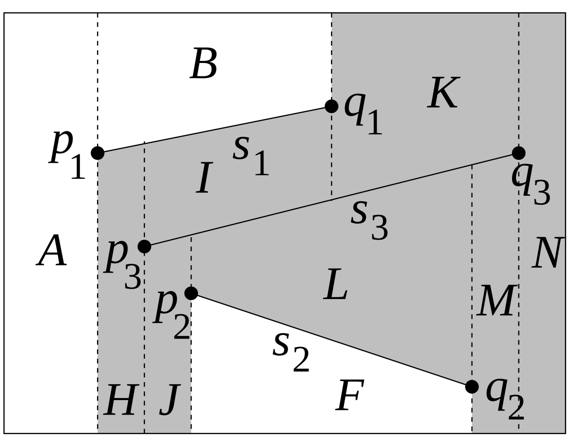


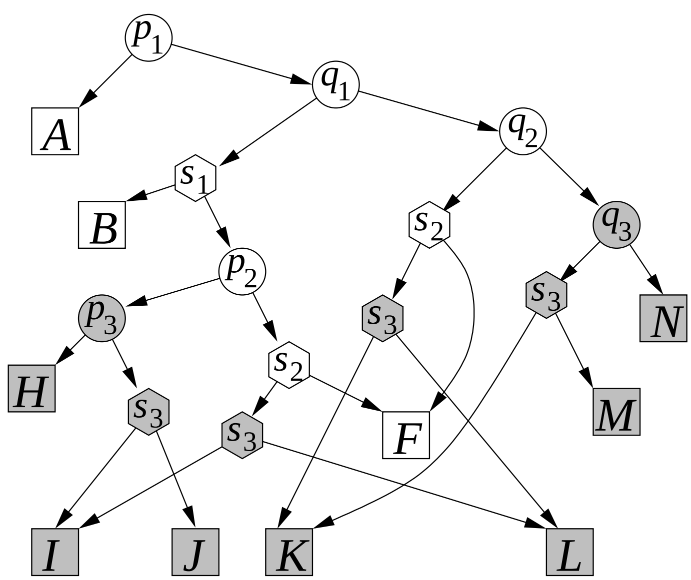
Analysis

- **Space is $O(n)$, and query time is $O(\log n)$, both in expectation.**
- **Expected bound depends on the random permutation, and not on the choice of input segments or the query point.**
- **The data structure size $\propto$ number of trapezpoids, which is $O(n)$, since $O(1)$ expected number of traps created when a new segment inserted.**
- **In order to analyze query bound, fix a query q.**
- **We consider how q moves incrementally through the trapezoidal map as new segments are inserted.**
- **Search complexity $\propto$ number of trapezoids encountered by q.**

Search Analysis

- Let Δ_i be trapezoid containing q after insertion of ith segment.
- If $\Delta_i = \Delta_{i-1}$ then new insertion does not affect q's trapezoid. (E.g. $q \in B$ and s_3's insertion.)
- If $\Delta_i \neq \Delta_{i-1}$, then new segment deleted q's trapezoid, and q needs to locate itself among the (at most 4) new traps.
- q could fall 3 levels in the tree. E.g. $q \in C$ falling to J after s_3's insertion.



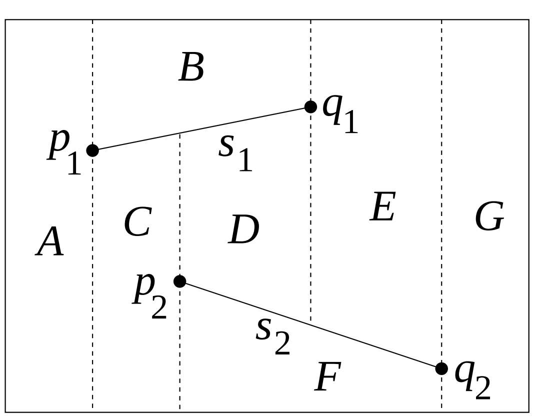


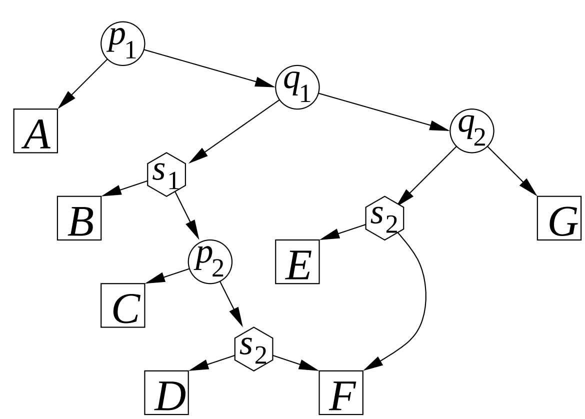
Search Analysis

- Let P_i be probability that $\Delta_i \neq \Delta_{i-1}$, over all random permutation.
- Since q can drop ≤ 3 levels, expected search path length is $\sum_{i=1}^{n} 3P_i$.
- We will show that $P_i \leq 4/i$. That will imply that expected search path length is

$$3\sum_{i=1}^{n}\frac{4}{i} = 12\sum_{i=1}^{n}\frac{1}{i} = 12\ln n$$

- Why is $P_i \leq 4/i$? Use backward analysis.
- The trapezoid Δ_i depends on at most 4 segments. The probability that ith segment is one of these 4 is at most $4/i$.





Final Remarks

- **Expectation only says that average search path is small. It can still have large variance.**
- **The trapezoidal map data structure has bounds on variance too. See the textbook for complete analysis.**

  Theorem: For any $\lambda > 0$, the probability that depth of the randomized seach structure exceeds $3\lambda \ln(n+1)$ is at most

  $$\frac{2}{(n+1)^{\lambda \ln 1.25 - 3}}$$

- **More careful analysis can provide better constants for the data structure.**