Voronoi Diagrams

Outline

- Definitions and Examples
- Properties of Voronoi diagrams
- Complexity of Voronoi diagrams
- Constructing Voronoi diagrams
  - Intuitions
  - Data Structures
  - Algorithm
- Running Time Analysis
- Demo
- Duality and degenerate cases

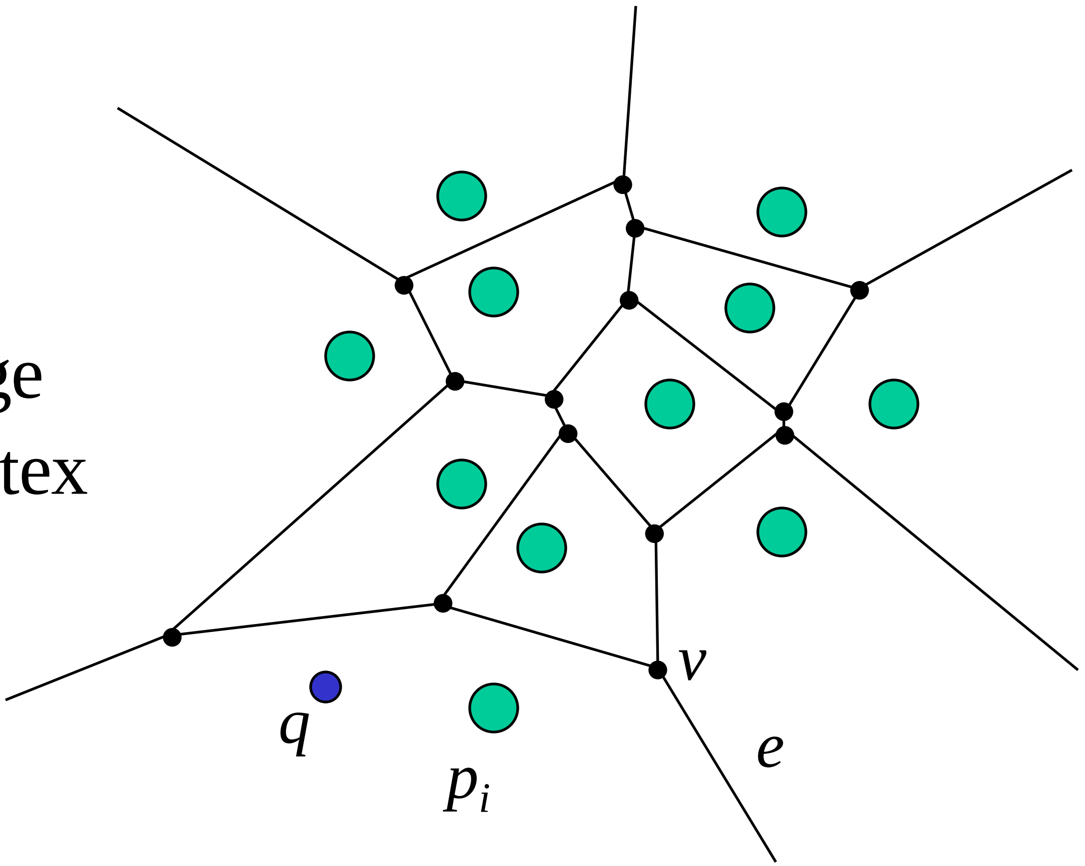
Post Office: What is the area of service?

p_i : site points

q : free point

e : Voronoi edge

v : Voronoi vertex

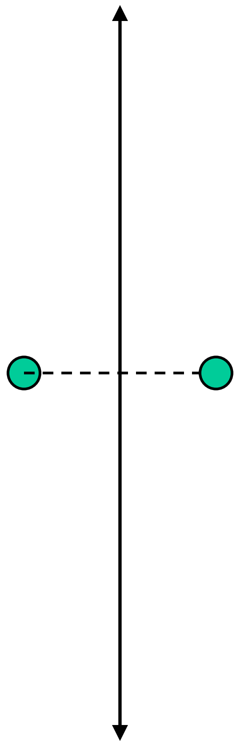


Definition of Voronoi Diagram

- Let P be a set of n distinct points (sites) in the plane.
- The Voronoi diagram of P is the subdivision of the plane into n cells, one for each site.
- A point q lies in the cell corresponding to a site $p_i \in P$ *iff*
  Euclidean_Distance(q, p_i) $<$ Euclidean_distance(q, p_j),
  for each $p_i \in P, j \neq i$.

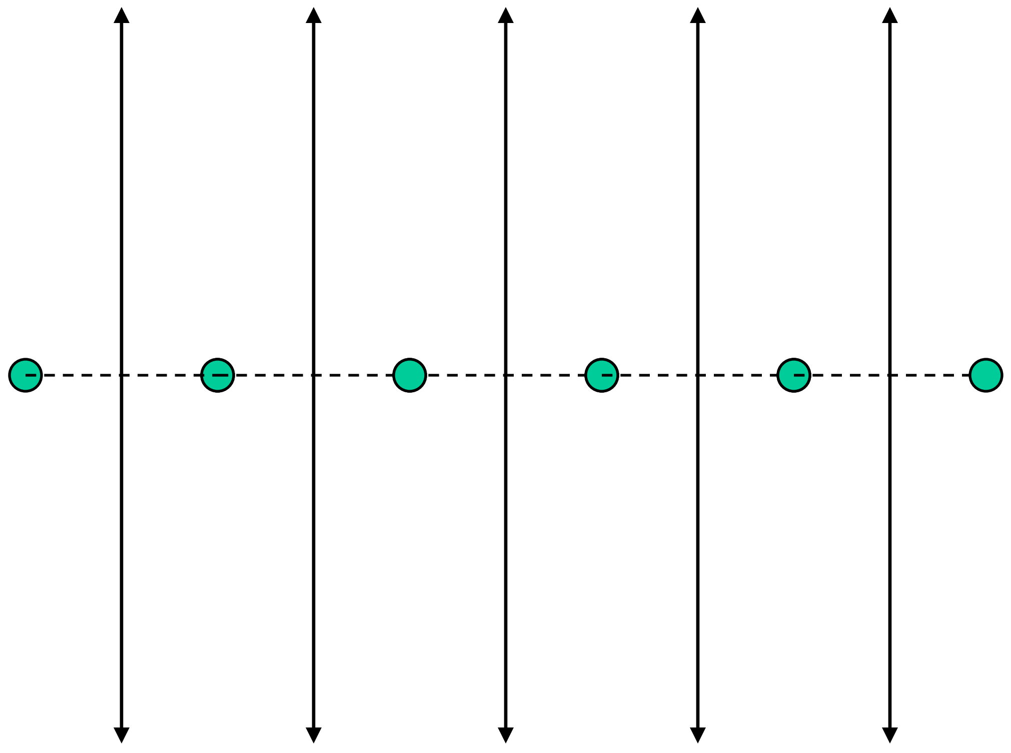
Voronoi Diagram Example: 1 site

Two sites form a perpendicular bisector

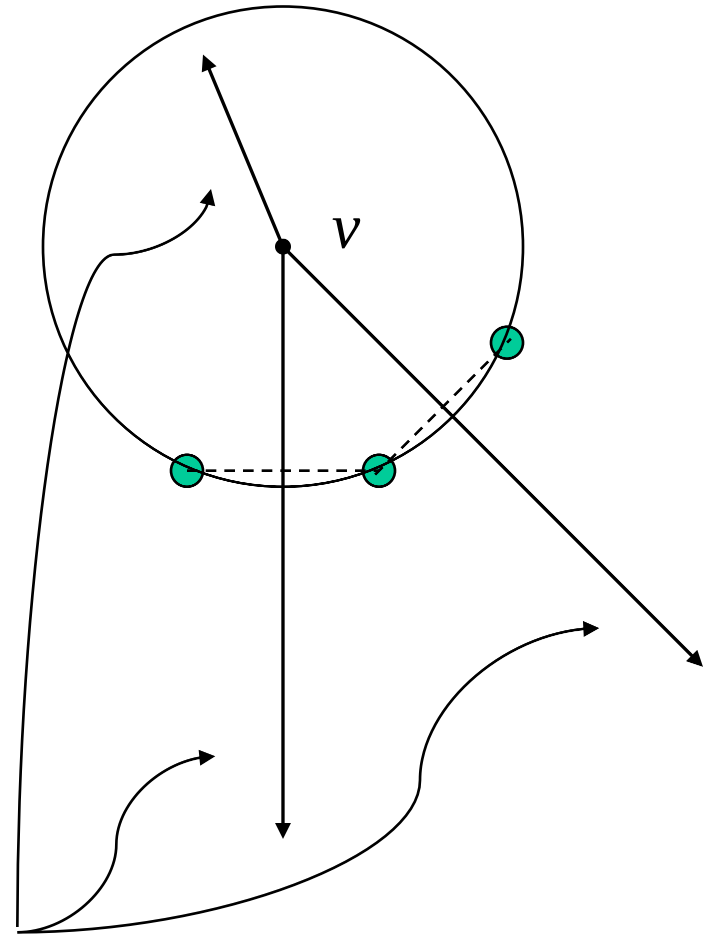


Voronoi Diagram is a line that extends infinitely in both directions, and the two half planes on either side.

Collinear sites form a series of parallel lines



Non-collinear sites form Voronoi half lines that meet at a vertex

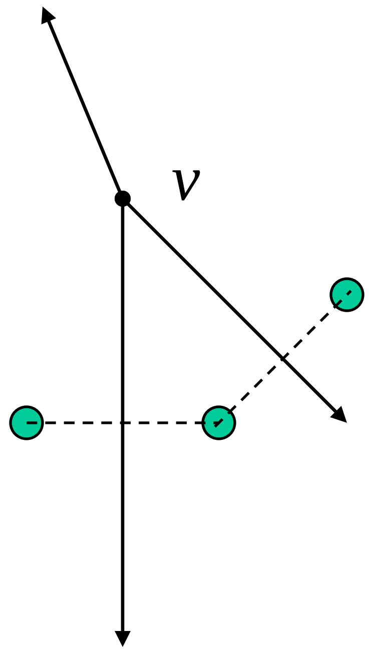


A vertex has degree ≥ 3

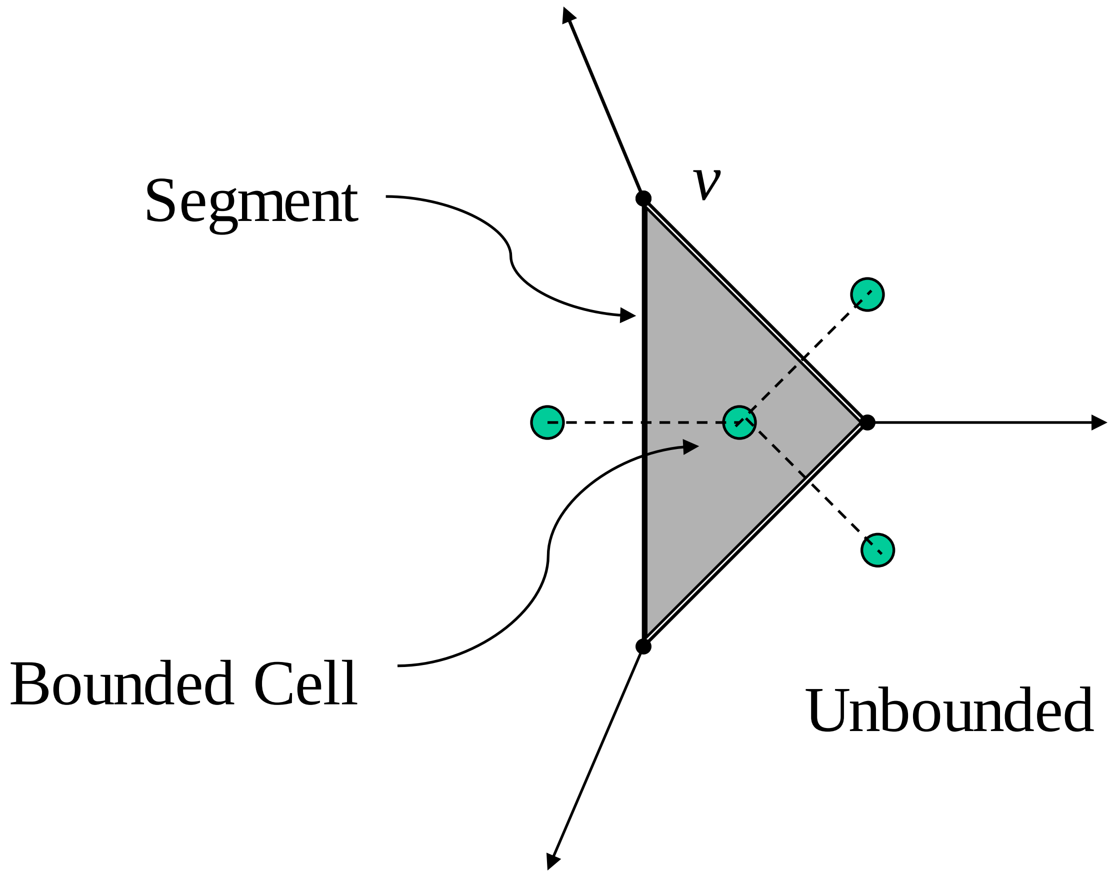
Half lines

A Voronoi vertex is the center of an *empty* circle touching 3 or more sites.

Voronoi Cells and Segments



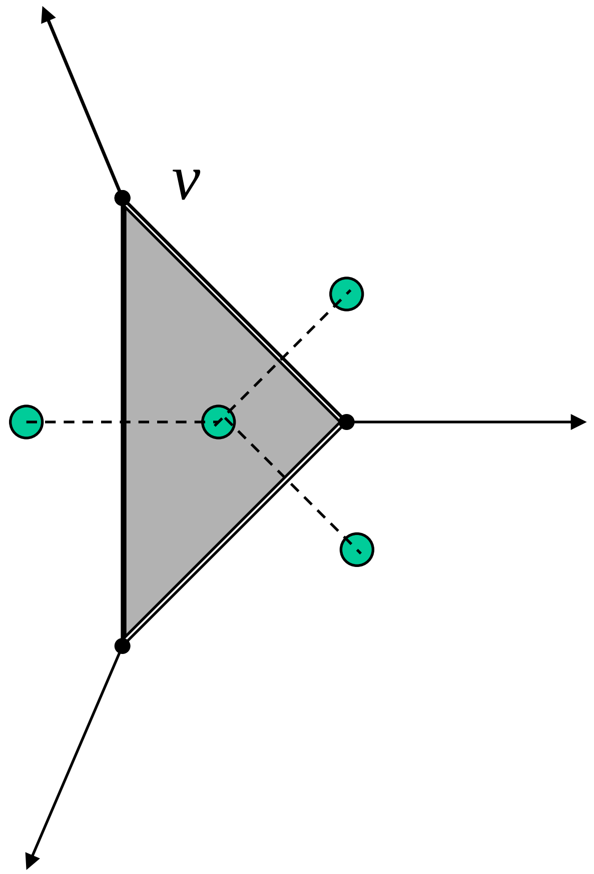
Voronoi Cells and Segments



Which of the following is true for 2-D Voronoi diagrams?

Four or more non-collinear sites are…

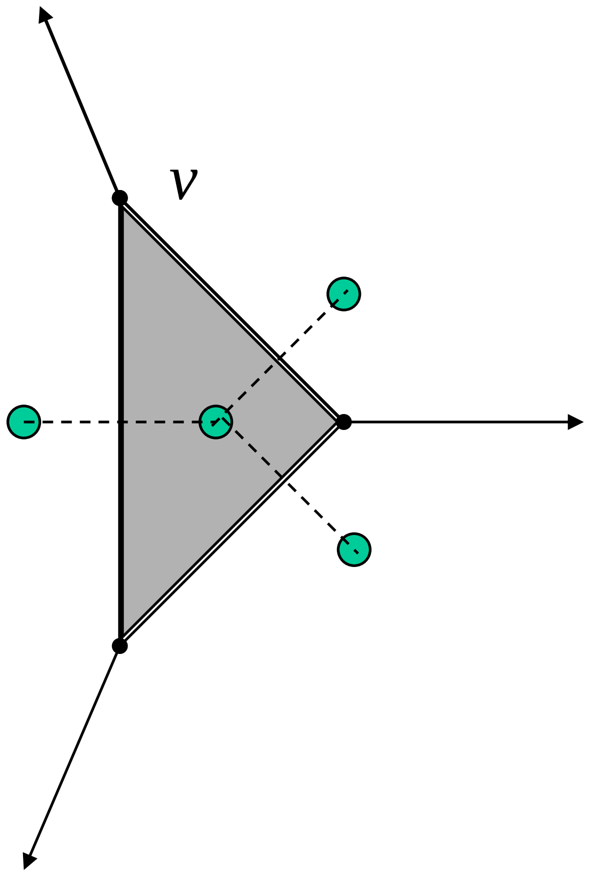
1. sufficient to create a bounded cell
2. necessary to create a bounded cell
3. 1 and 2
4. none of above



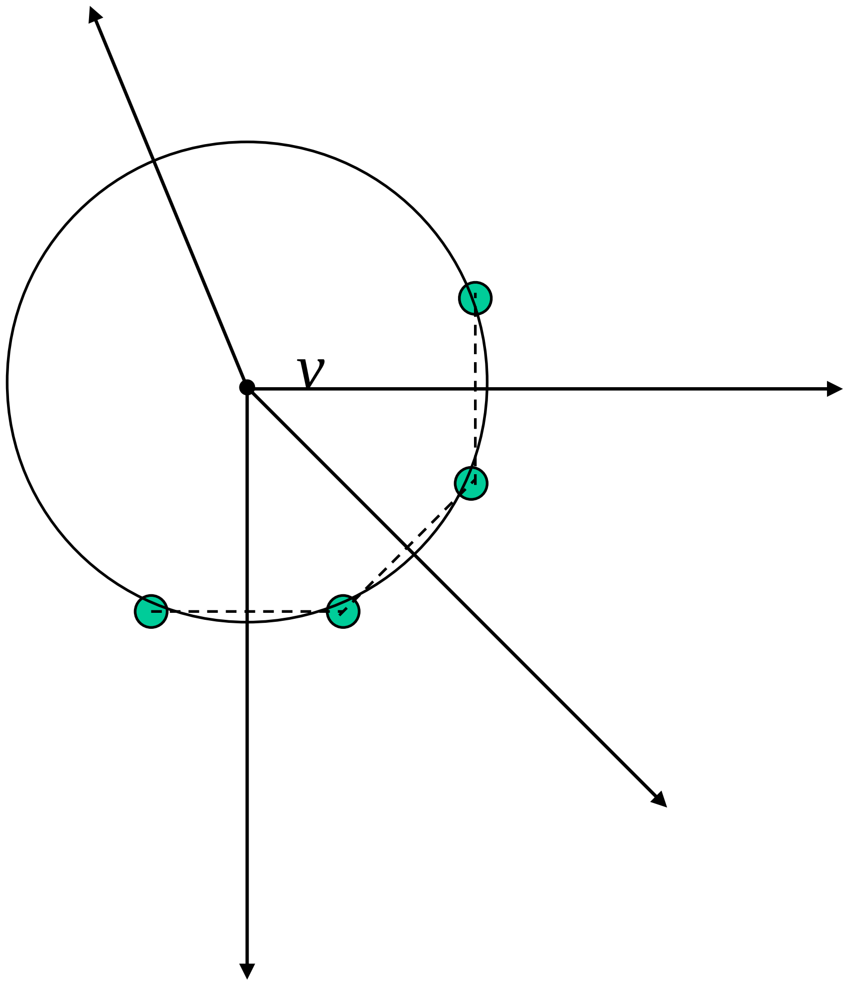
Which of the following is true for 2-D Voronoi diagrams?

Four or more non-collinear sites are…

1. sufficient to create a bounded cell
2. **necessary to create a bounded cell**
3. 1 and 2
4. none of above



Degenerate Case: no bounded cells!

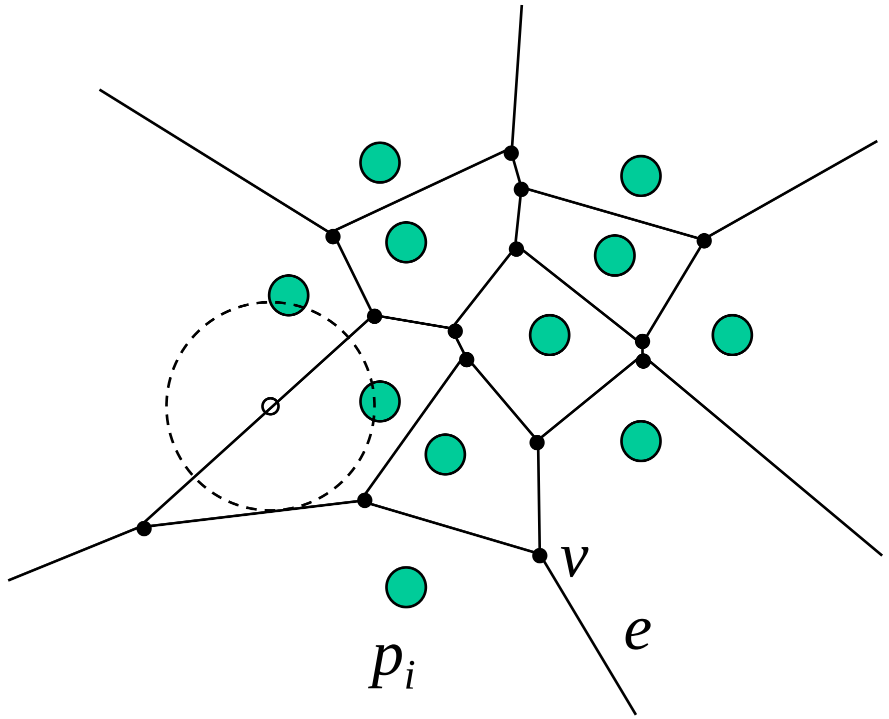


Summary of Voronoi Properties

A point q lies on a Voronoi edge between sites p_i and p_j *iff* the largest empty circle centered at q touches only p_i and p_j

– A Voronoi edge is a subset of locus of points equidistant from p_i and p_j

p_i : site points
e : Voronoi edge
v : Voronoi vertex

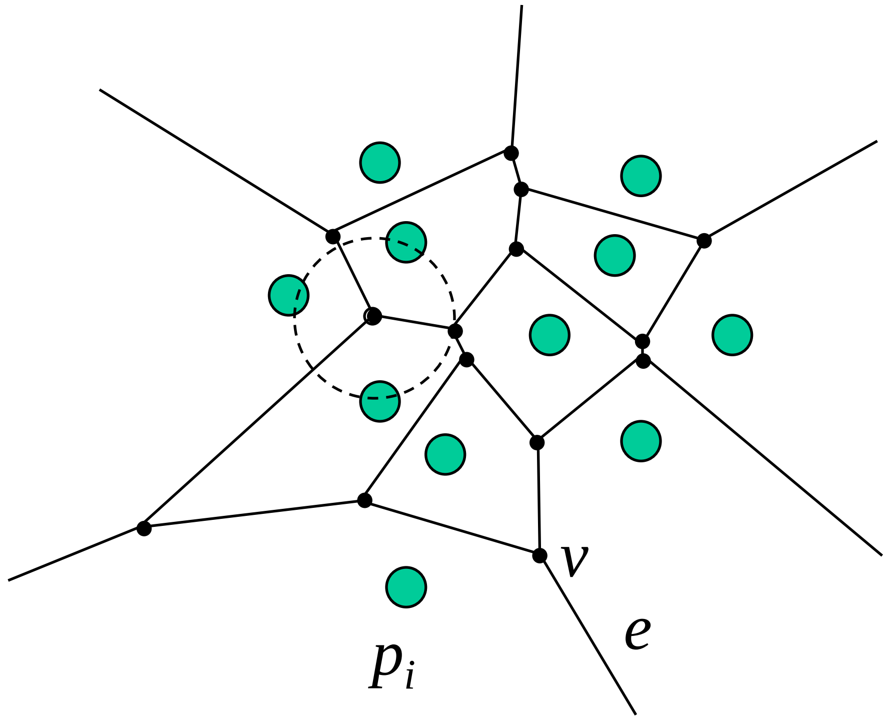


Summary of Voronoi Properties

A point *q* is a vertex *iff* the largest empty circle centered at *q* touches at least 3 sites

– A Voronoi vertex is an intersection of 3 more segments, each equidistant from a pair of sites

p_i : site points
e : Voronoi edge
v : Voronoi vertex

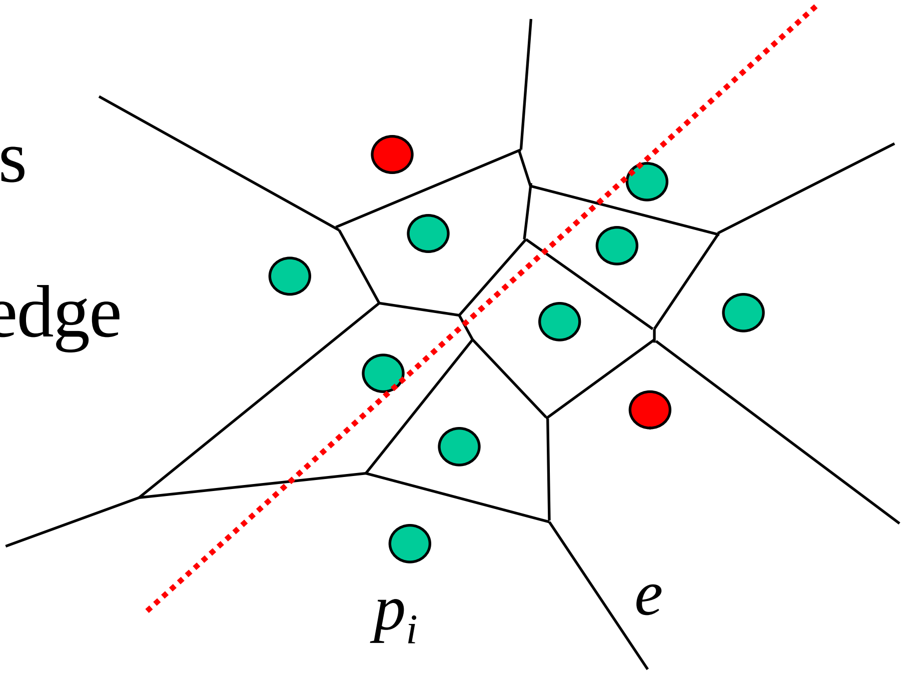


Voronoi diagrams have linear complexity $\{|v|, |e| = O(n)\}$

Intuition: Not all bisectors are Voronoi edges!

p_i : site points

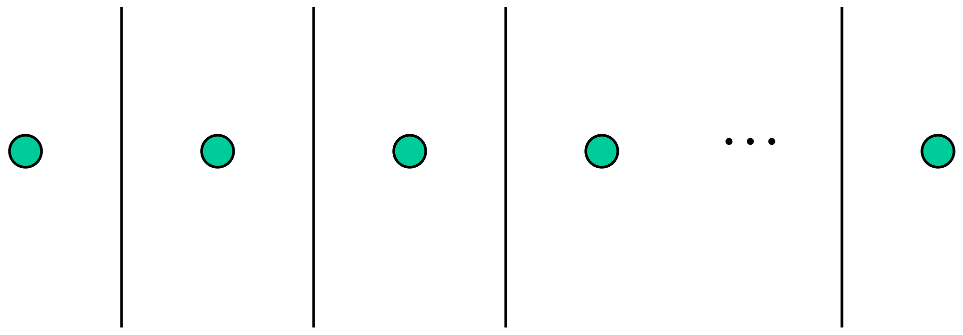
e : Voronoi edge



Voronoi diagrams have linear complexity $\{|v|, |e| = O(n)\}$

Claim: For $n \geq 3$, $|v| \leq 2n - 5$ and $|e| \leq 3n - 6$

Proof: (Easy Case)



Collinear sites → $|v| = 0$, $|e| = n - 1$

Voronoi diagrams have linear complexity $\{|v|, |e| = O(n)\}$

Claim: For $n \geq 3$, $|v| \leq 2n - 5$ and $|e| \leq 3n - 6$

Proof: (General Case)

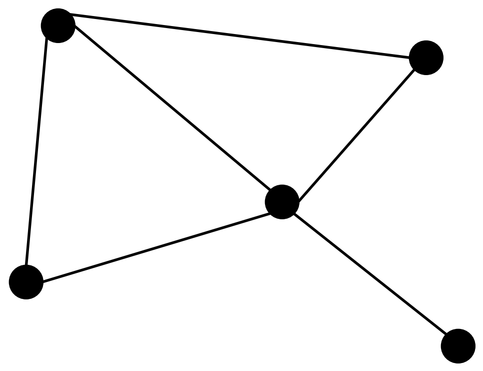
- Euler's Formula: for connected, planar graphs, $|v| - |e| + f = 2$

Where:

$|v|$ is the number of vertices

$|e|$ is the number of edges

f is the number of faces



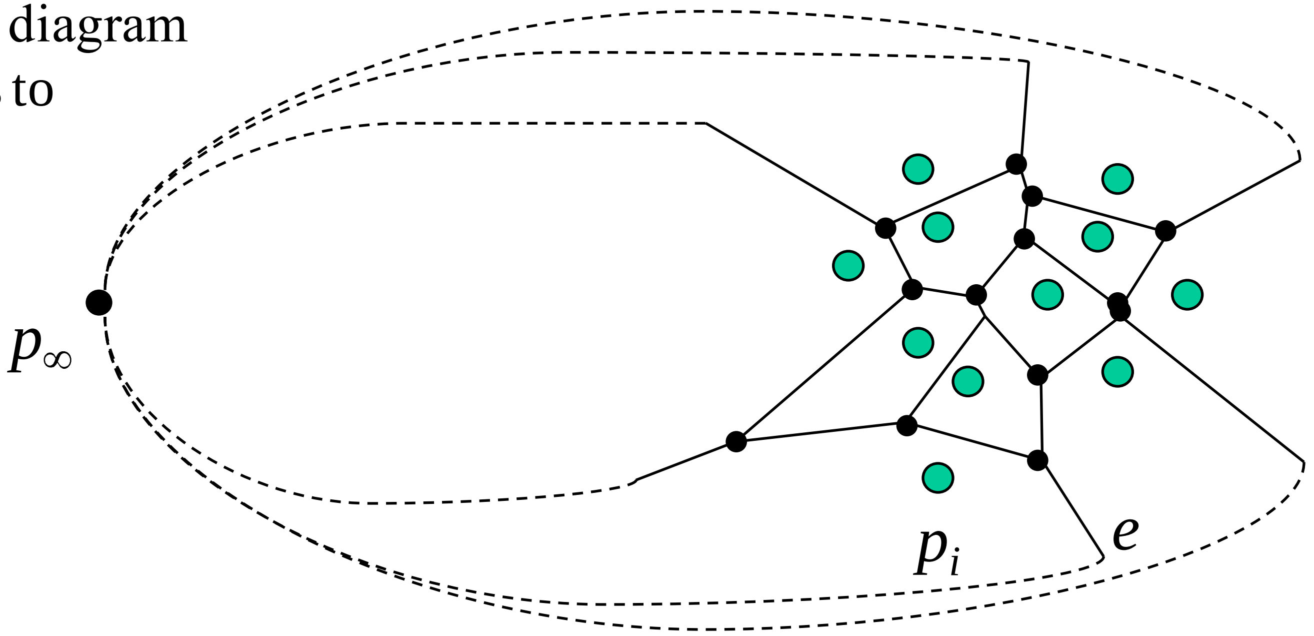
Voronoi diagrams have linear complexity $\{|v|, |e| = O(n)\}$

Claim: For $n \geq 3$, $|v| \leq 2n - 5$ and $|e| \leq 3n - 6$

Proof: (General Case)

- For Voronoi graphs, $f = n \rightarrow (|v| + 1) - |e| + n = 2$

To apply Euler's Formula, we "planarize" the Voronoi diagram by connecting half lines to an extra vertex.



Voronoi diagrams have linear complexity {|v|, |e| = O(n)}

Moreover,

$$\sum_{v \in Vor(P)} \deg(v) = 2 \cdot |e|$$

and

$$\forall v \in Vor(P), \quad \deg(v) \geq 3$$

so

$$2 \cdot |e| \geq 3(|v|+1)$$

together with

$$(|v|+1) - |e| + n = 2$$

we get, for $n \geq 3$

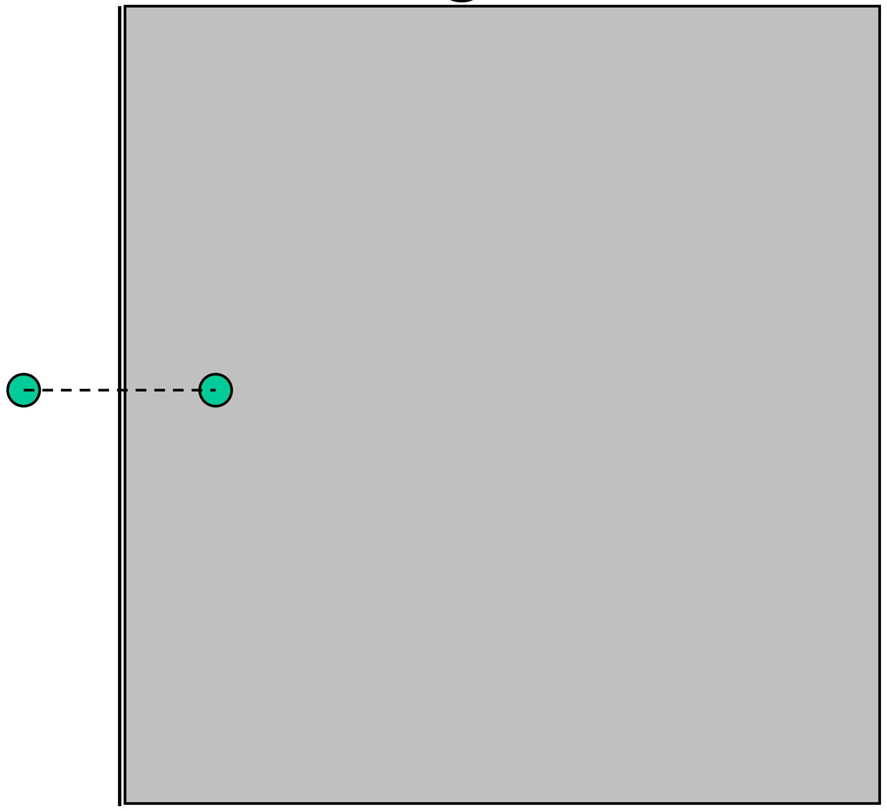
$$|v| \leq 2n - 5$$
$$|e| \leq 3n - 6$$

Constructing Voronoi Diagrams

Given a half plane intersection algorithm…

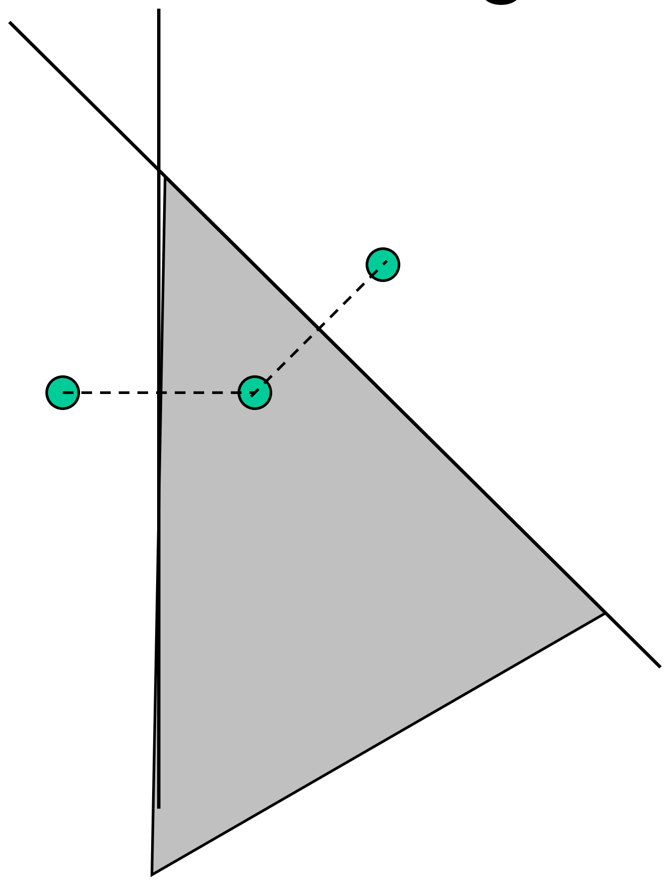
Constructing Voronoi Diagrams

Given a half plane intersection algorithm…



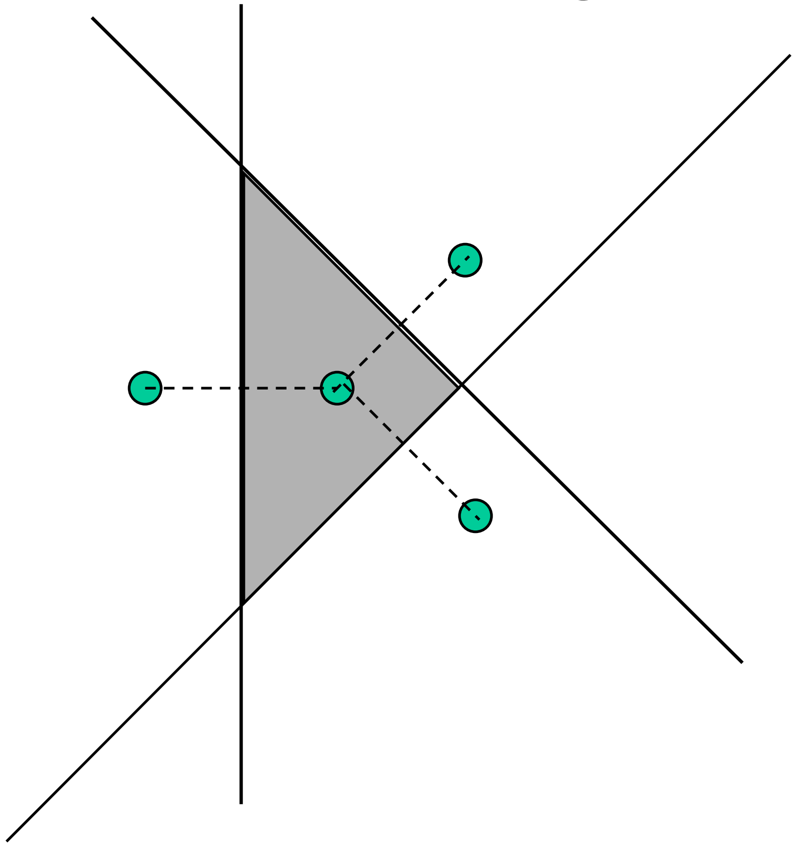
Constructing Voronoi Diagrams

Given a half plane intersection algorithm…



Constructing Voronoi Diagrams

Given a half plane intersection algorithm…



Repeat for each site

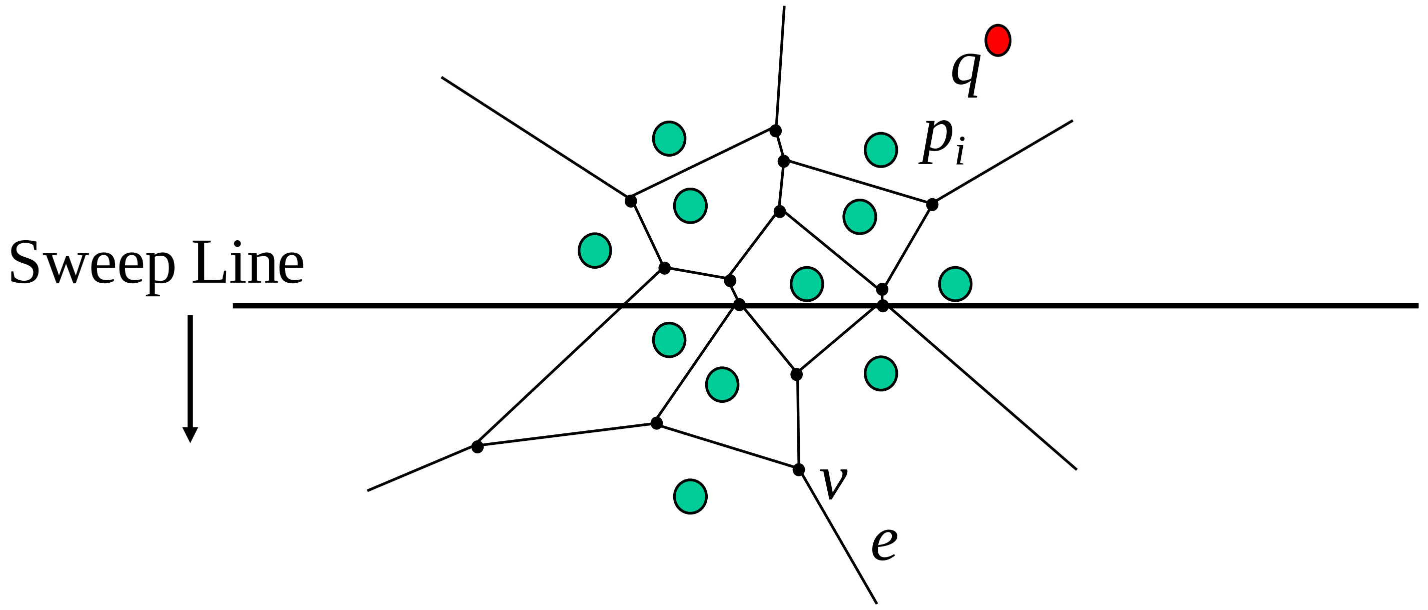
Running Time:
O($n^2 \log n$)

Constructing Voronoi Diagrams

- Half plane intersection $O(n^2 \log n)$
- Fortune's Algorithm
  - Sweep line algorithm
    - Voronoi diagram constructed as horizontal line sweeps the set of sites from top to bottom
    - Incremental construction → maintains portion of diagram which cannot change due to sites below sweep line, keeping track of incremental changes for each site (and Voronoi vertex) it "sweeps"

Constructing Voronoi Diagrams

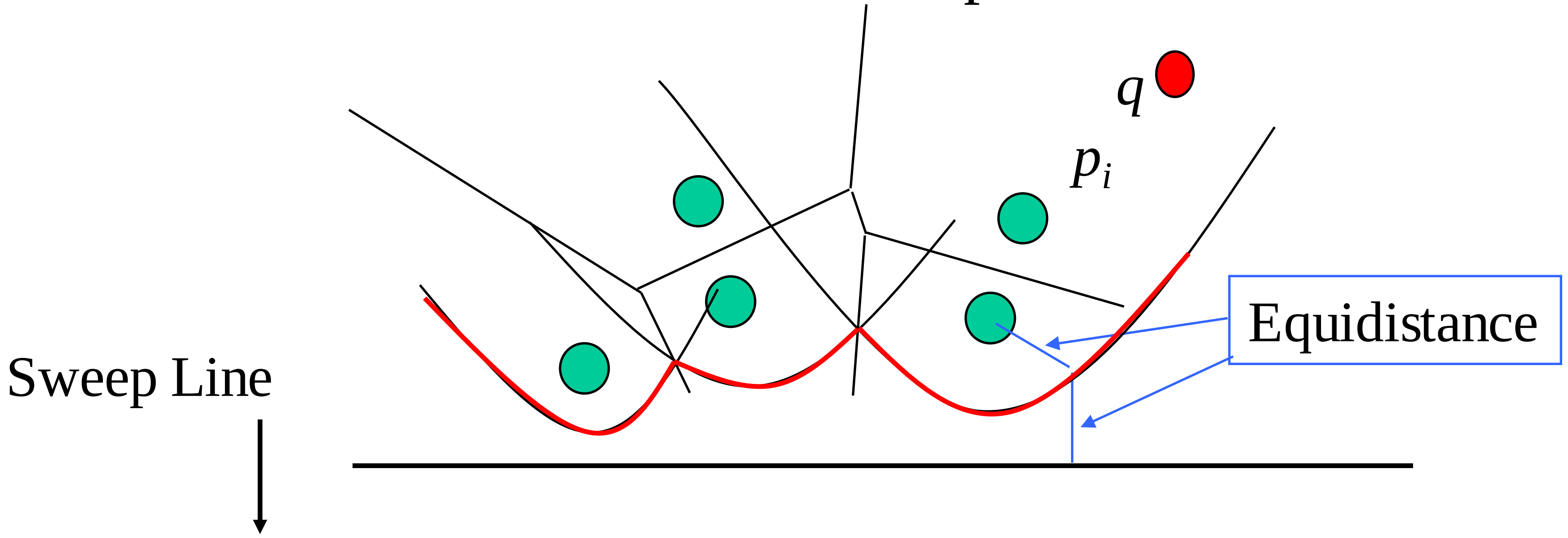
What is the invariant we are looking for?



Maintain a representation of the locus of points q that are closer to some site p_i *above* the sweep line than to the line itself (and thus to any site below the line).

Constructing Voronoi Diagrams

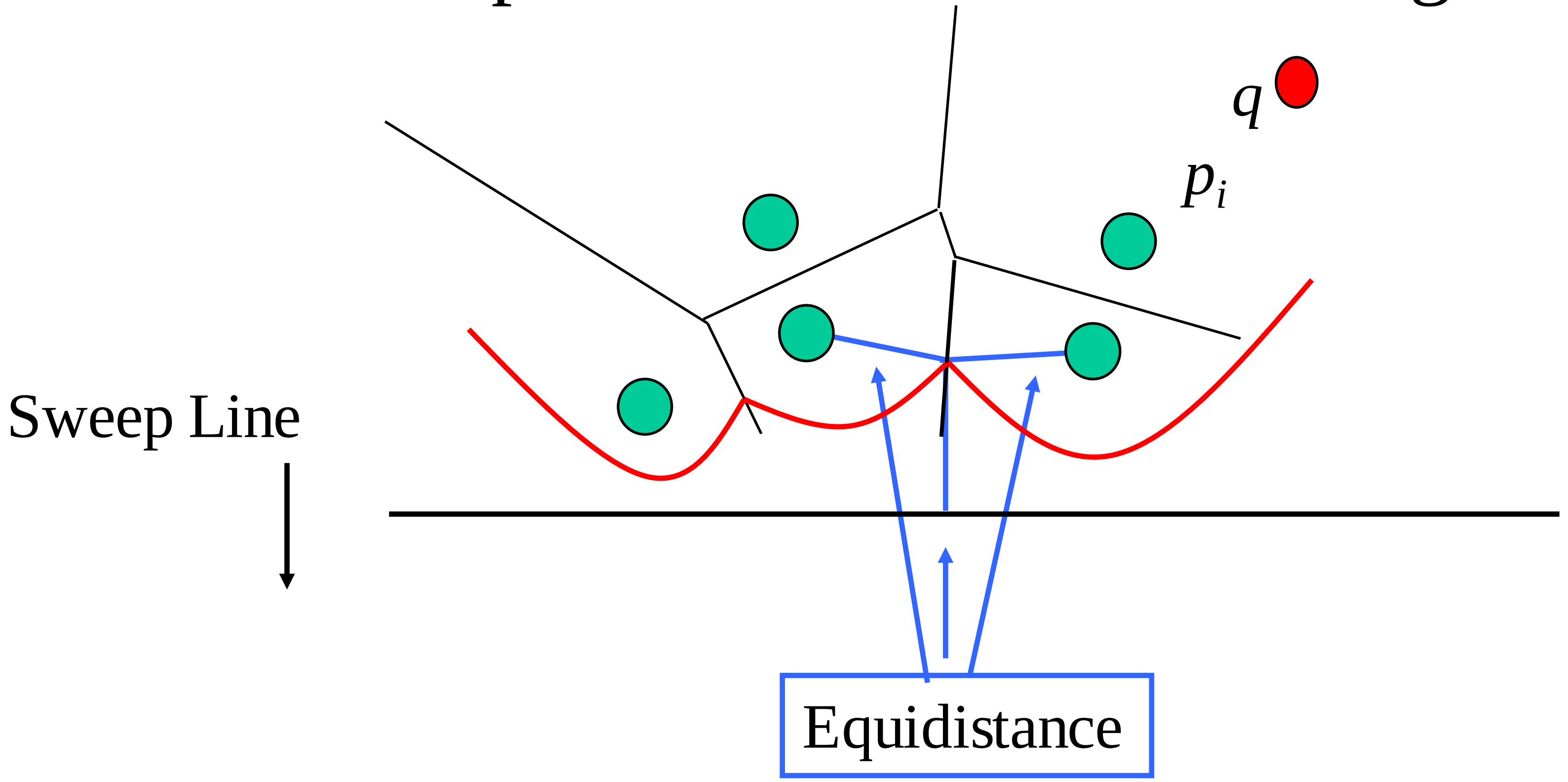
Which points are closer to a site above the sweep line than to the sweep line itself?



The set of parabolic arcs form a beach-line that bounds the locus of all such points

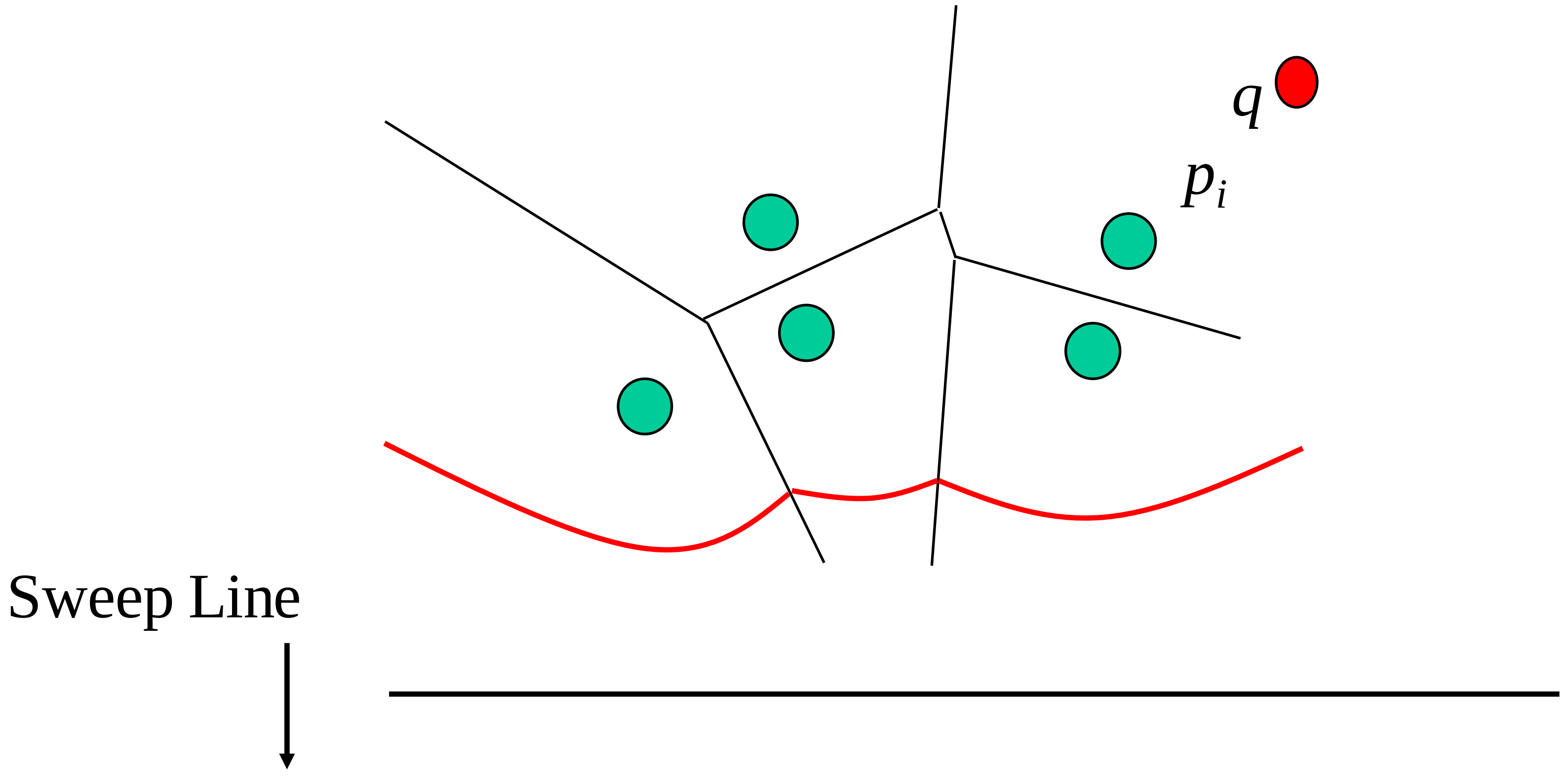
Constructing Voronoi Diagrams

Break points trace out Voronoi edges.



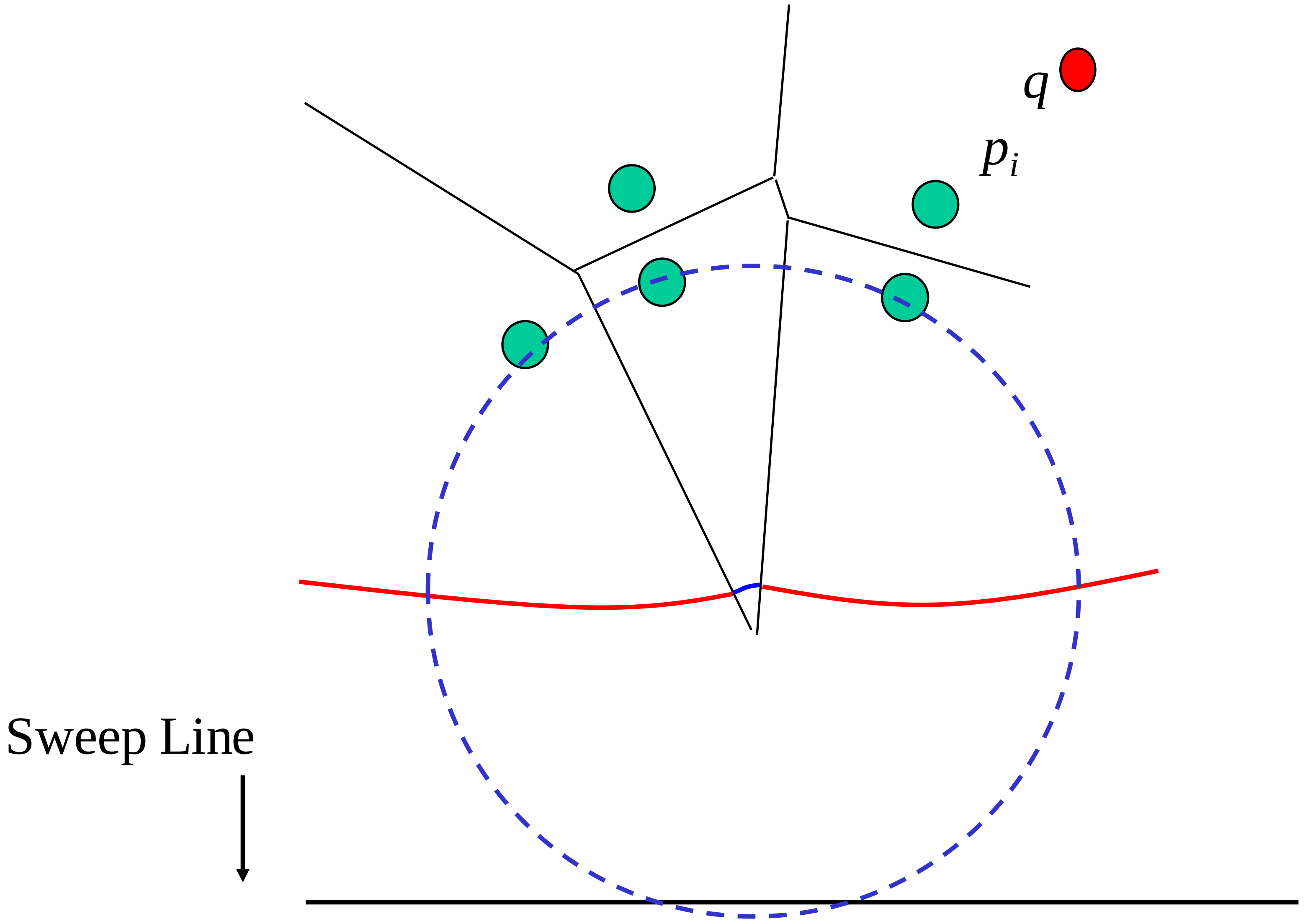
Constructing Voronoi Diagrams

Arcs flatten out as sweep line moves down.



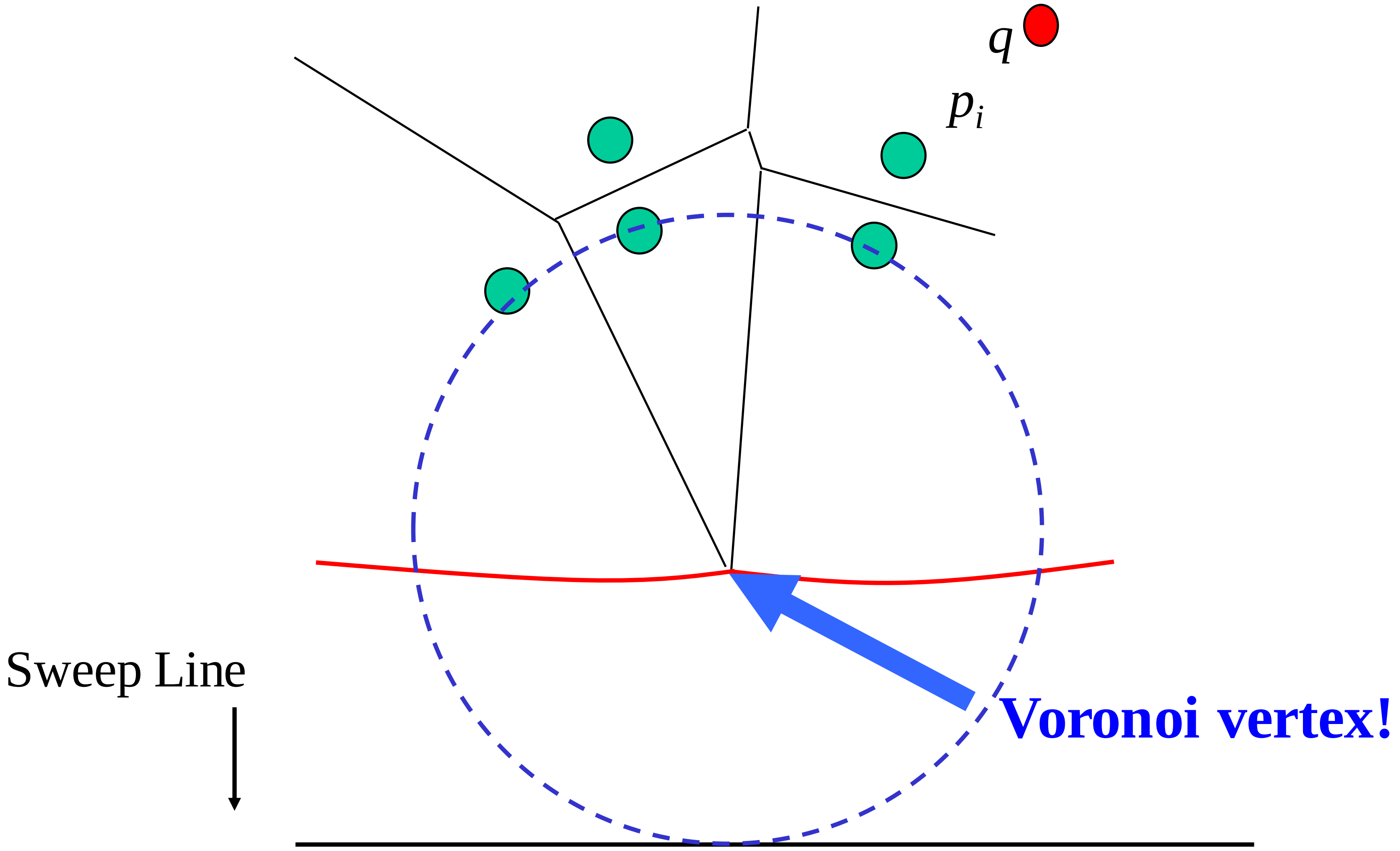
Constructing Voronoi Diagrams

Eventually, the middle arc disappears.



Constructing Voronoi Diagrams

We have detected a circle that is empty (contains no sites) and touches 3 or more sites.



Beach Line properties

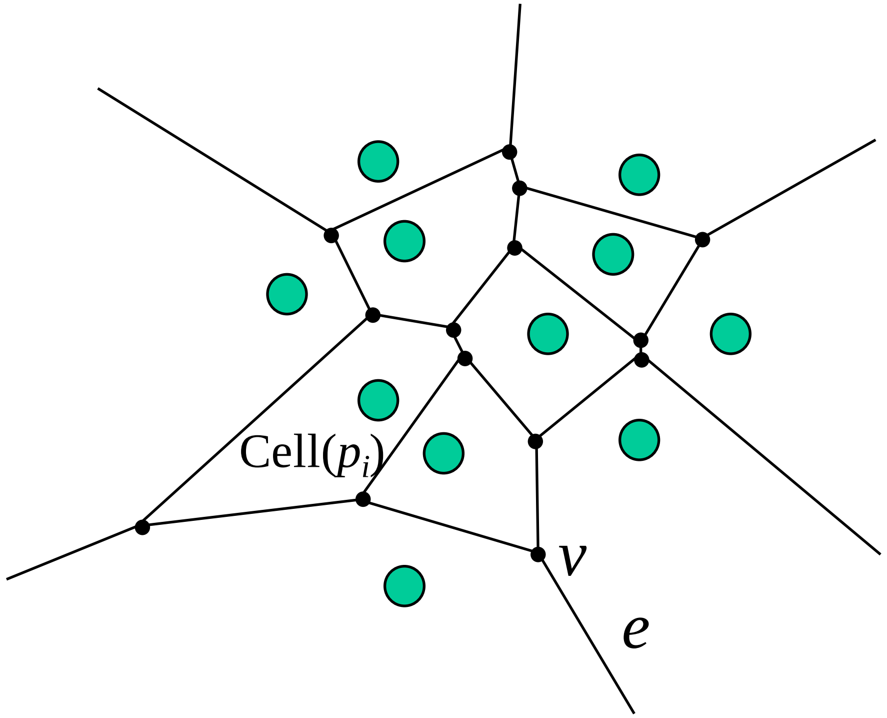
- Voronoi edges are traced by the break points as the sweep line moves down.
  - Emergence of a new break point(s) (from formation of a new arc or a fusion of two existing break points) identifies a new edge
- Voronoi vertices are identified when two break points meet (fuse).
  - Decimation of an old arc identifies new vertex

Data Structures

- Current state of the Voronoi diagram
  - Doubly linked list of half-edge, vertex, cell records
- Current state of the beach line
  - Keep track of break points
  - Keep track of arcs currently on beach line
- Current state of the sweep line
  - Priority event queue sorted on decreasing y-coordinate

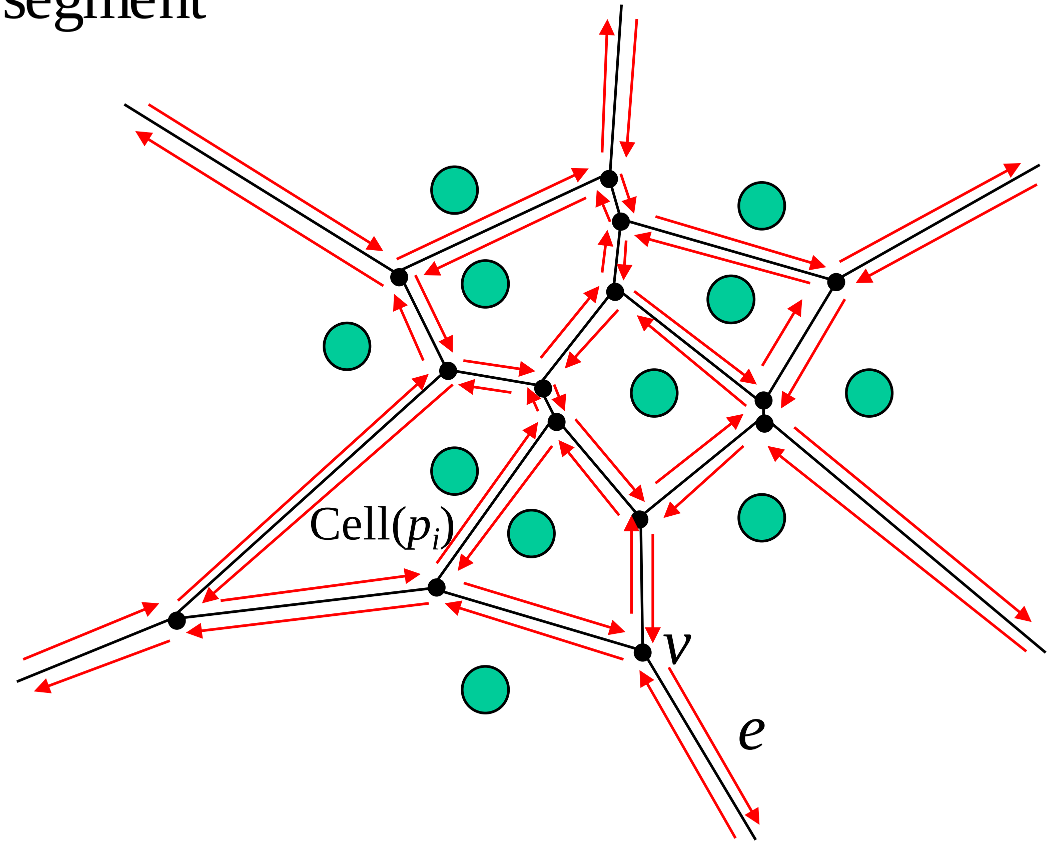
Doubly Linked List (*D*)

- Goal: a simple data structure that allows an algorithm to traverse a Voronoi diagram's segments, cells and vertices



Doubly Linked List (D)

- Divide segments into uni-directional half-edges
- A chain of counter-clockwise half-edges forms a cell
- Define a half-edge's "twin" to be its opposite half-edge of the same segment

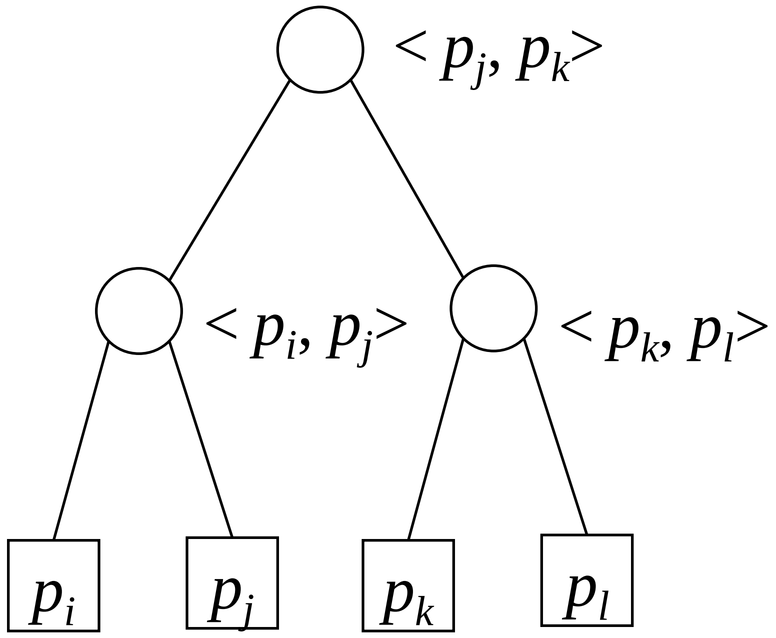


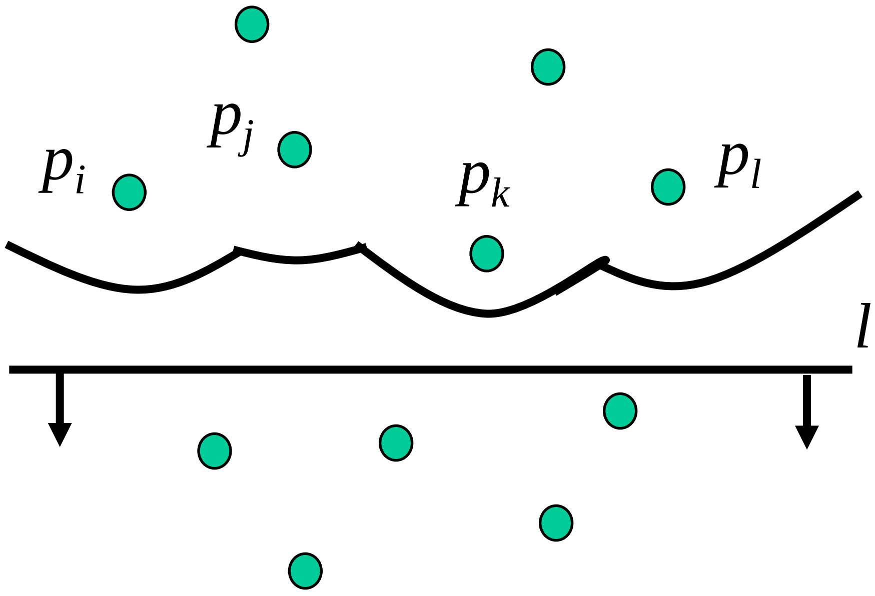
Doubly Linked List (*D*)

- Cell Table
  - Cell(p_i) : pointer to any incident half-edge
- Vertex Table
  - v_i : list of pointers to all incident half-edges
- Doubly Linked-List of half-edges; each has:
  - Pointer to Cell Table entry
  - Pointers to start/end vertices of half-edge
  - Pointers to previous/next half-edges in the CCW chain
  - Pointer to twin half-edge

Balanced Binary Tree (*T*)

- Internal nodes represent break points between two arcs
  - Also contains a pointer to the *D* record of the edge being traced
- Leaf nodes represent arcs, each arc is in turn represented by the site that generated it
  - Also contains a pointer to a potential circle event



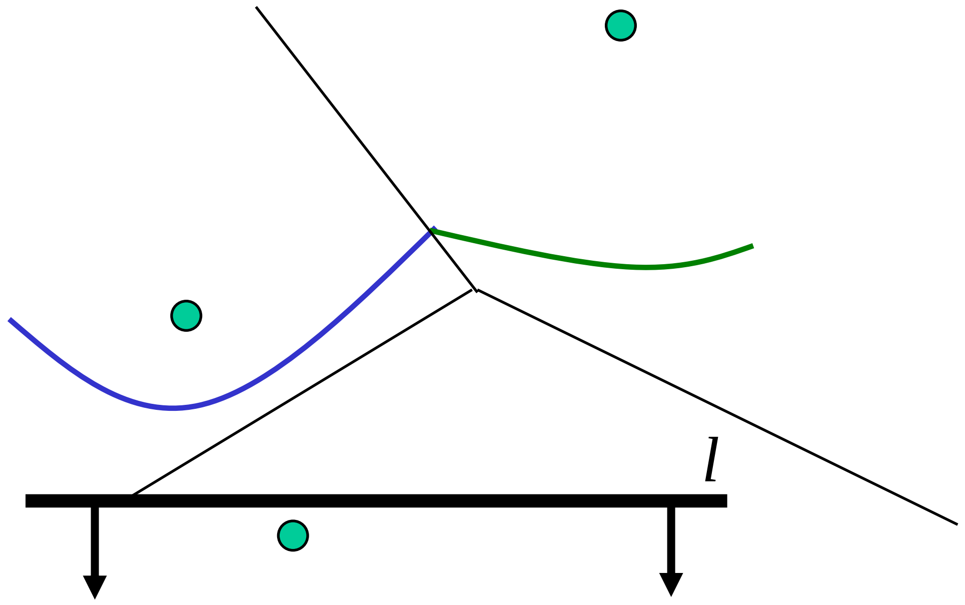


Event Queue (*Q*)

- An event is an interesting point encountered by the sweep line as it sweeps from top to bottom
  - Sweep line makes discrete stops, rather than a continuous sweep
- Consists of Site Events (when the sweep line encounters a new site point) and Circle Events (when the sweep line encounters the *bottom* of an empty circle touching 3 or more sites).
- Events are prioritized based on y-coordinate

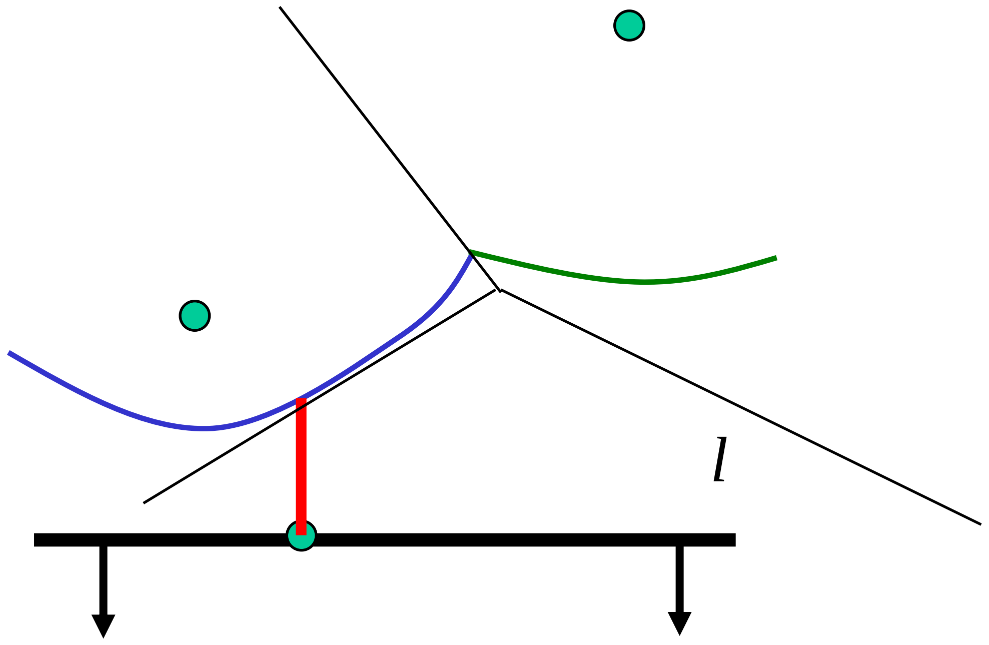
Site Event

A new arc appears when a new site appears.



Site Event

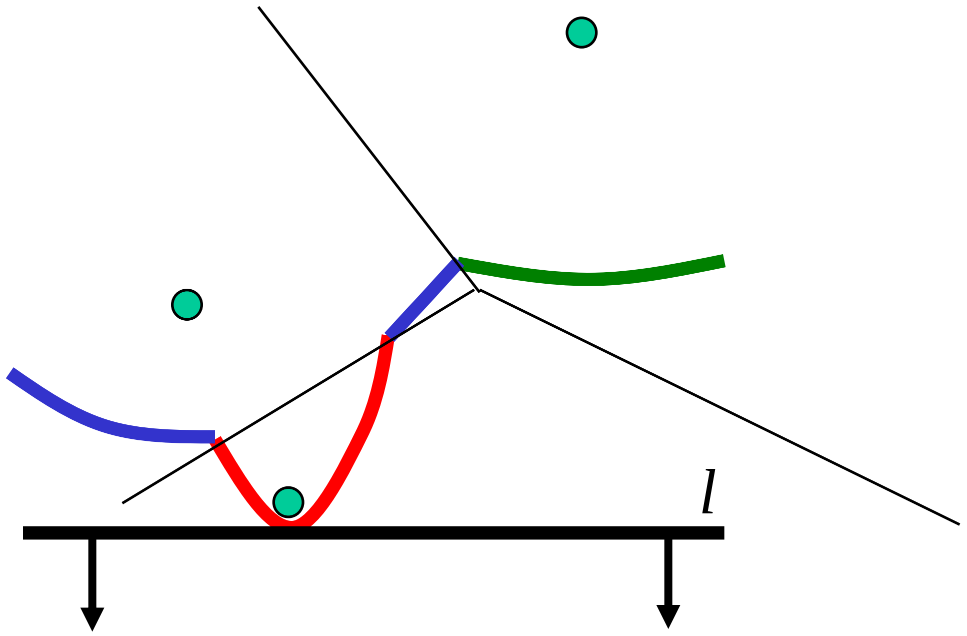
A new arc appears when a new site appears.



Site Event

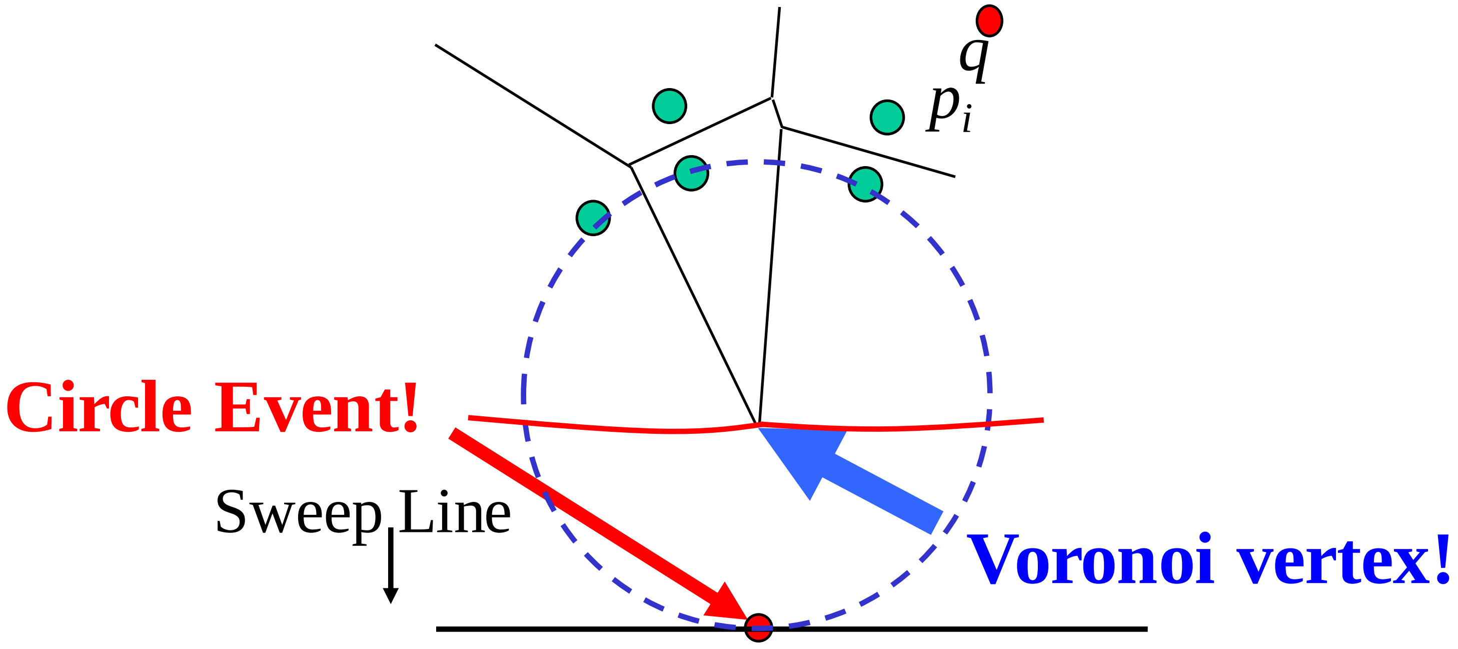
Original arc above the new site is broken into two

→ Number of arcs on beach line is O(*n*)



Circle Event

An arc disappears whenever an empty circle touches three or more sites and is tangent to the sweep line.



Sweep line helps determine that the circle is indeed empty.

Event Queue Summary

- Site Events are
  - given as input
  - represented by the xy-coordinate of the site point
- Circle Events are
  - computed on the fly (intersection of the two bisectors in between the three sites)
  - represented by the xy-coordinate of the lowest point of an empty circle touching three or more sites
  - “anticipated”, these newly generated events may be false and need to be removed later
- Event Queue prioritizes events based on their y-coordinates

Summarizing Data Structures

- Current state of the Voronoi diagram
  - Doubly linked list of half-edge, vertex, cell records
- Current state of the beach line
  - Keep track of break points
    - Inner nodes of binary search tree; represented by a tuple
  - Keep track of arcs currently on beach line
    - Leaf nodes of binary search tree; represented by a site that generated the arc
- Current state of the sweep line
  - Priority event queue sorted on decreasing y-coordinate

Algorithm

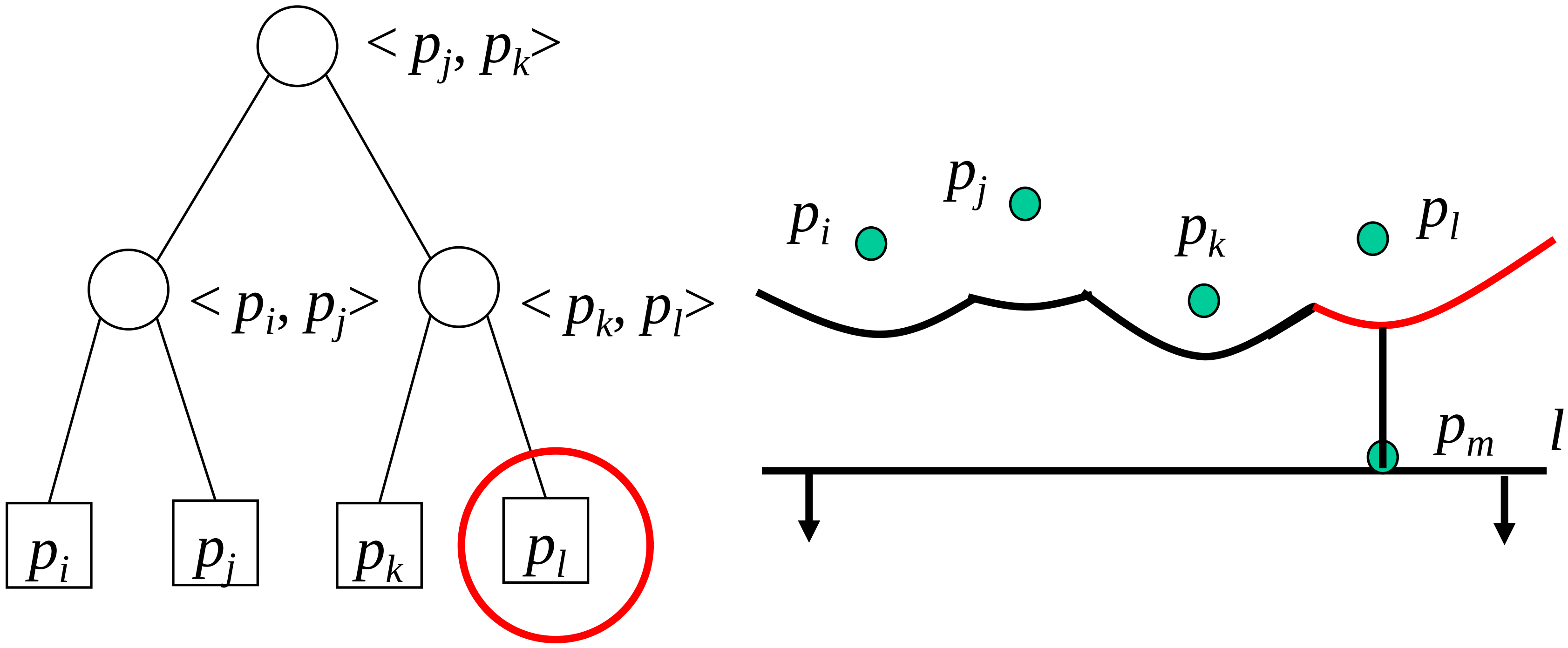
1. Initialize
   - Event queue Q ← all site events
   - Binary search tree T ← ∅
   - Doubly linked list D ← ∅
2. While Q not ∅,
   - Remove event (e) from Q with largest y-coordinate
     - HandleEvent(e, T, D)

Handling Site Events

1. Locate the existing arc (if any) that is above the new site
2. Break the arc by replacing the leaf node with a sub tree representing the new arc and its break points
3. Add two half-edge records in the doubly linked list
4. Check for potential circle event(s), add them to event queue if they exist

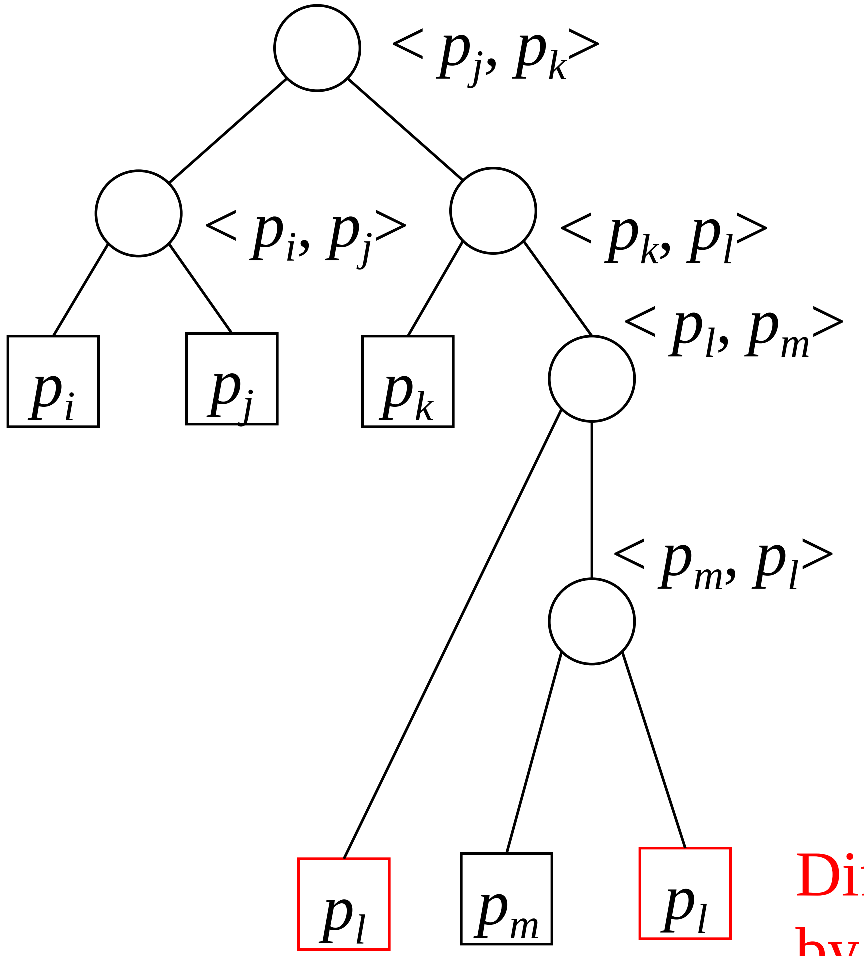
Locate the existing arc that is above the new site

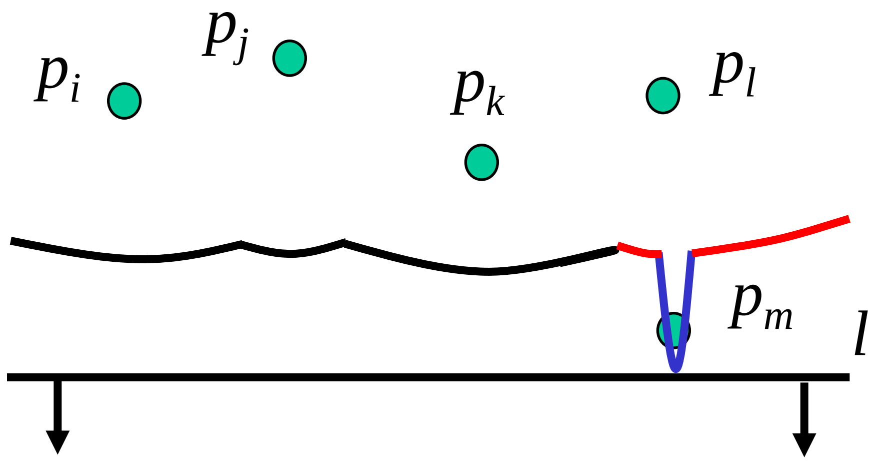
- The x coordinate of the new site is used for the binary search
- The x coordinate of each breakpoint along the root to leaf path is computed on the fly



Break the Arc

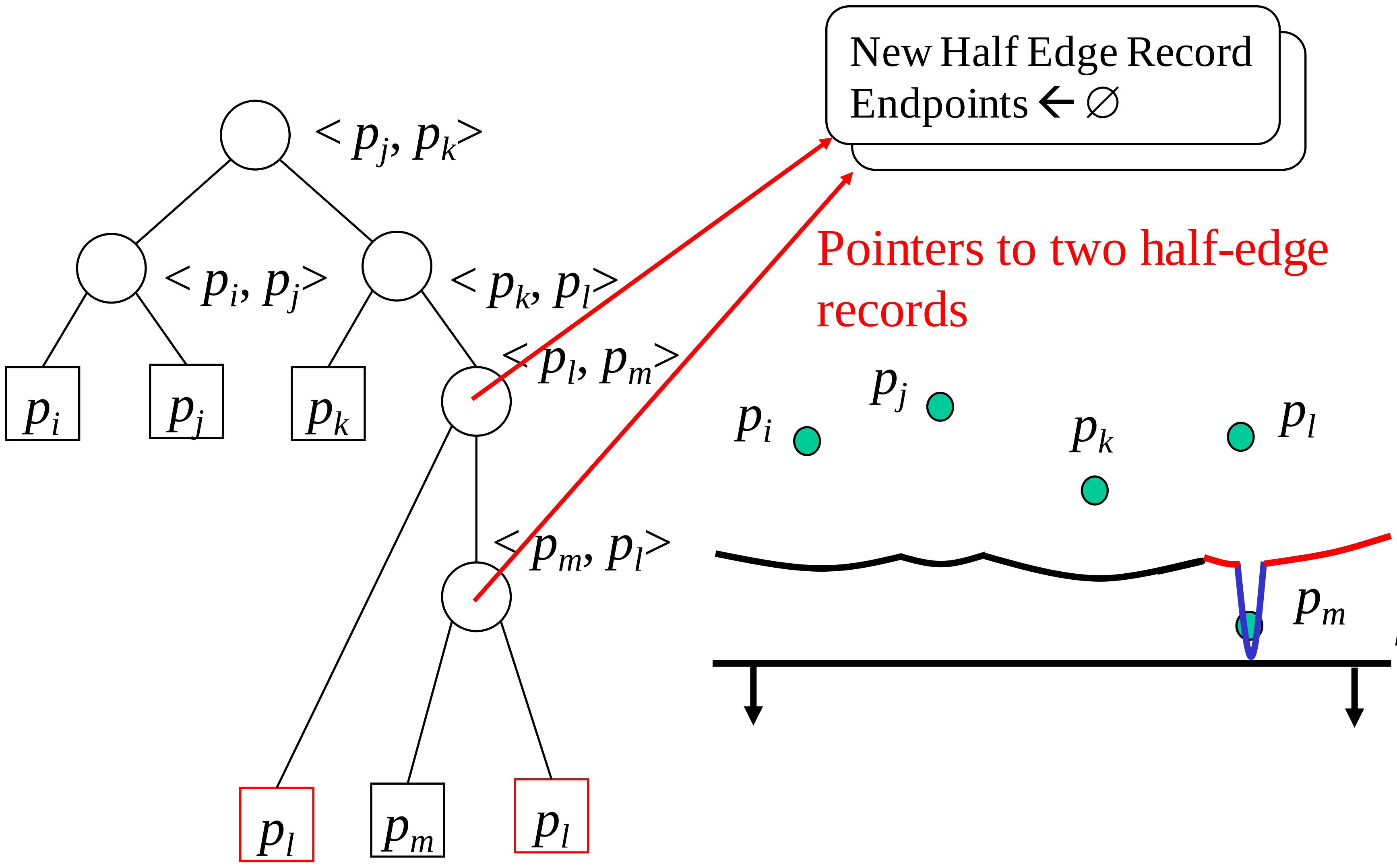
Corresponding leaf replaced by a new sub-tree





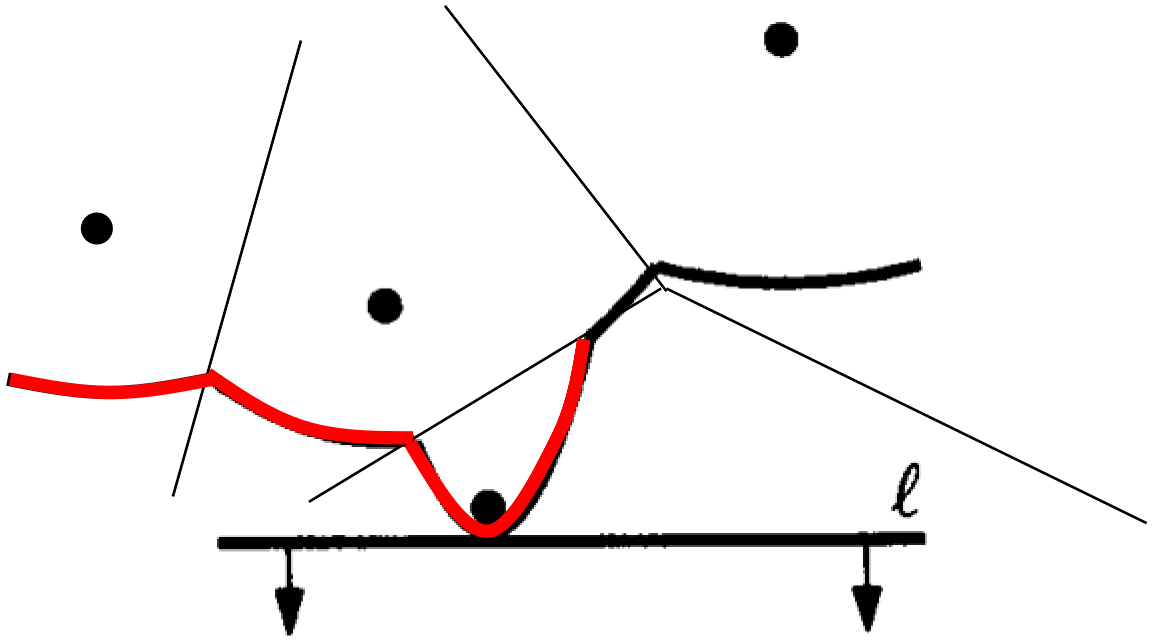
Different arcs can be identified by the same site!

Add a new edge record in the doubly linked list



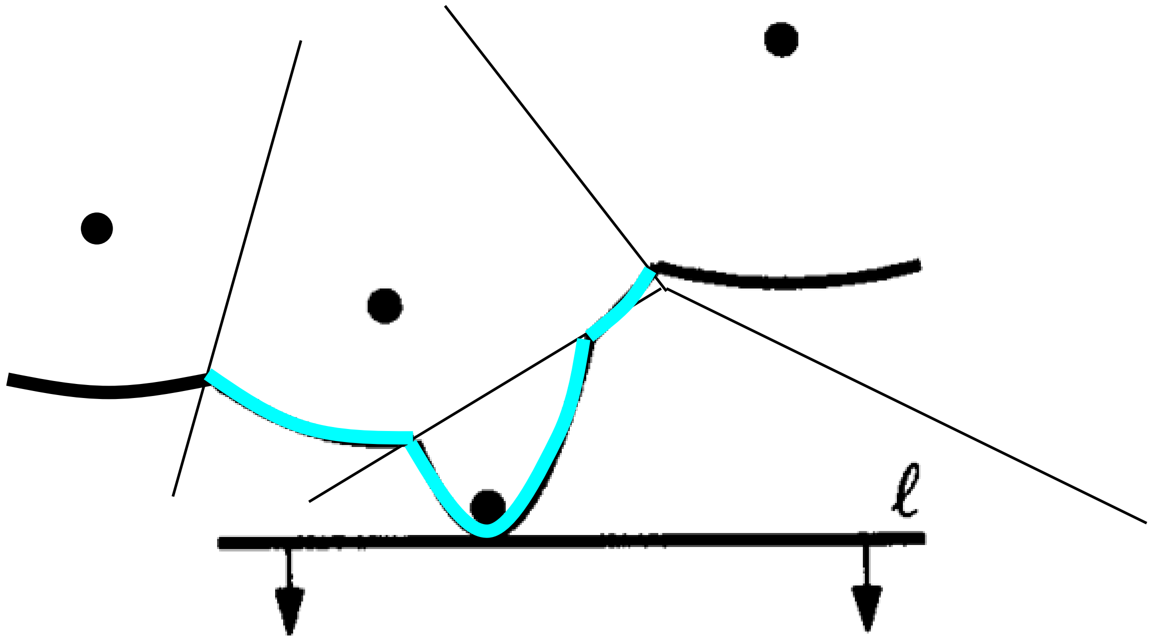
Checking for Potential Circle Events

- Scan for triple of consecutive arcs and determine if breakpoints converge
  - Triples with new arc in the middle do not have break points that converge



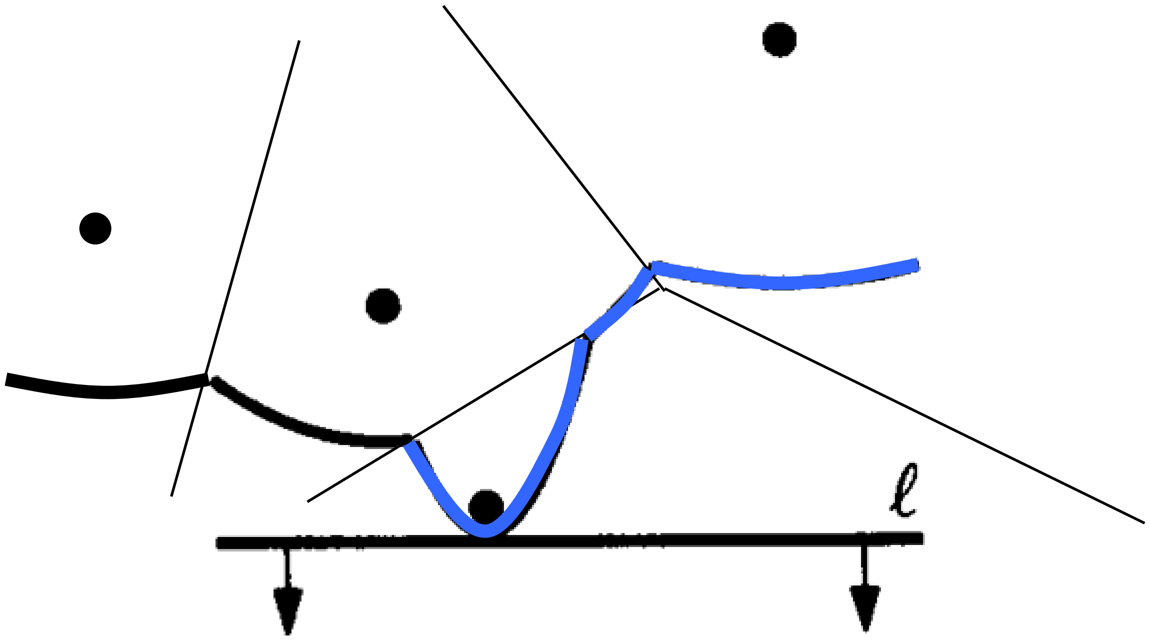
Checking for Potential Circle Events

- Scan for triple of consecutive arcs and determine if breakpoints converge
  - Triples with new arc in the middle do not have break points that converge



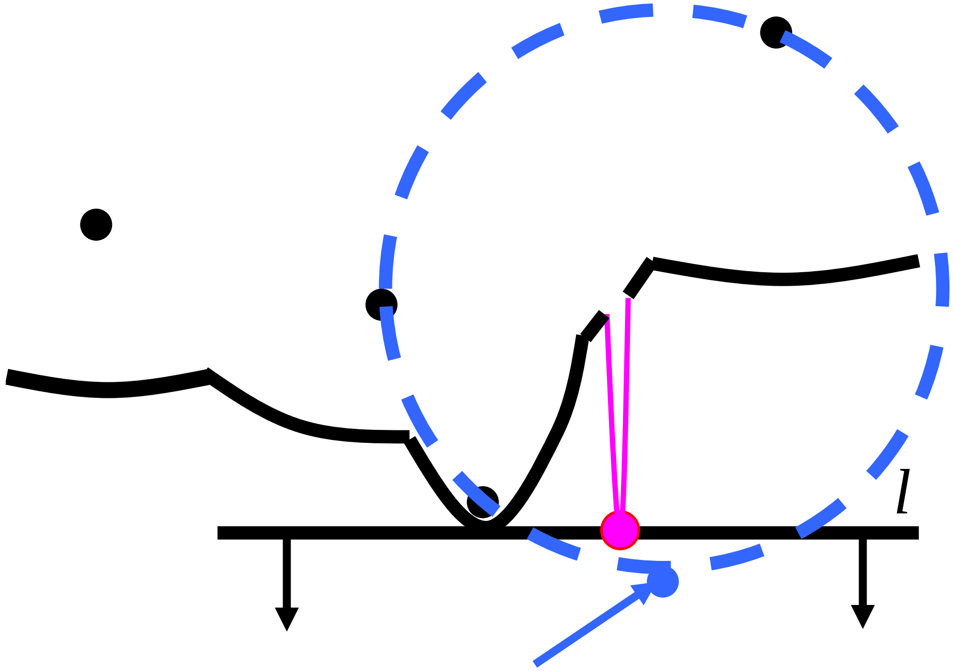
Checking for Potential Circle Events

- Scan for triple of consecutive arcs and determine if breakpoints converge
  - Triples with new arc in the middle do not have break points that converge



Converging break points may not always yield a circle event

- Appearance of a new site before the circle event makes the potential circle non-empty



(The original circle event becomes a *false alarm*)

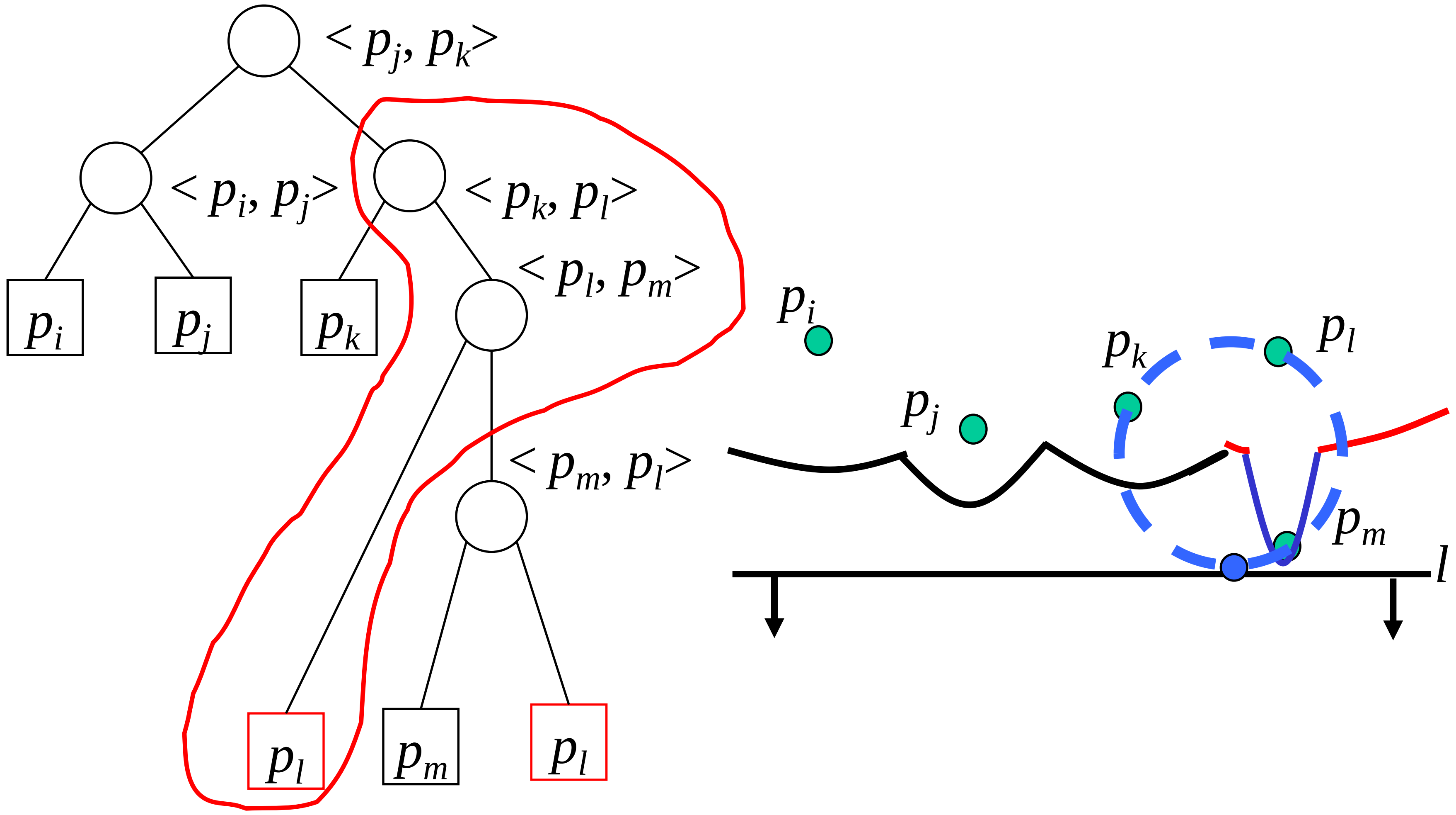
Handling Site Events

1. Locate the leaf representing the existing arc that is above the new site
   - Delete the potential circle event in the event queue
2. Break the arc by replacing the leaf node with a sub tree representing the new arc and break points
3. Add a new edge record in the doubly linked list
4. Check for potential circle event(s), add them to queue if they exist
   - Store in the corresponding leaf of T a pointer to the new circle event in the queue

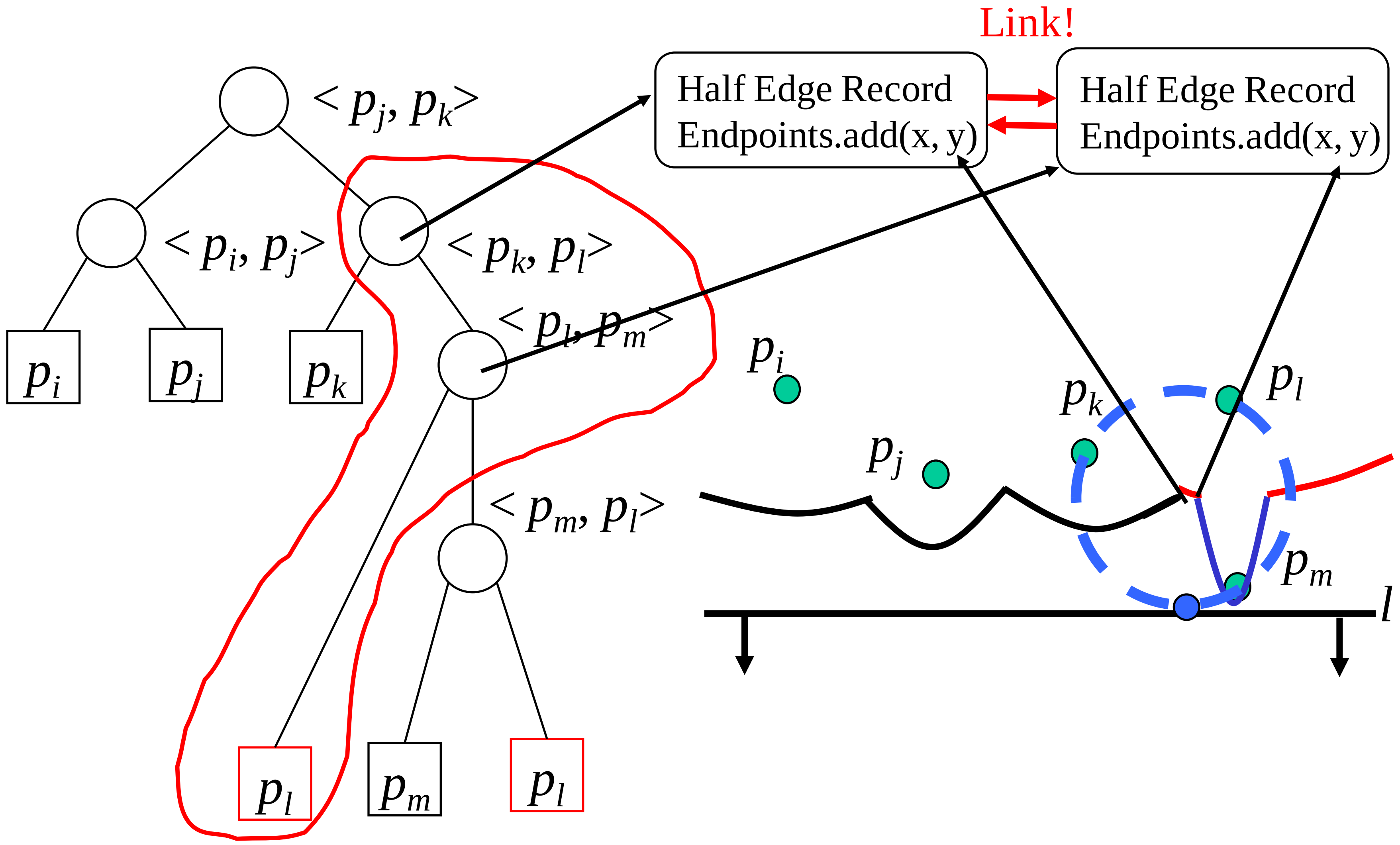
Handling Circle Events

1. Add vertex to corresponding edge record in doubly linked list
2. Delete from T the leaf node of the disappearing arc and its associated circle events in the event queue
3. Create new edge record in doubly linked list
4. Check the new triplets formed by the former neighboring arcs for potential circle events

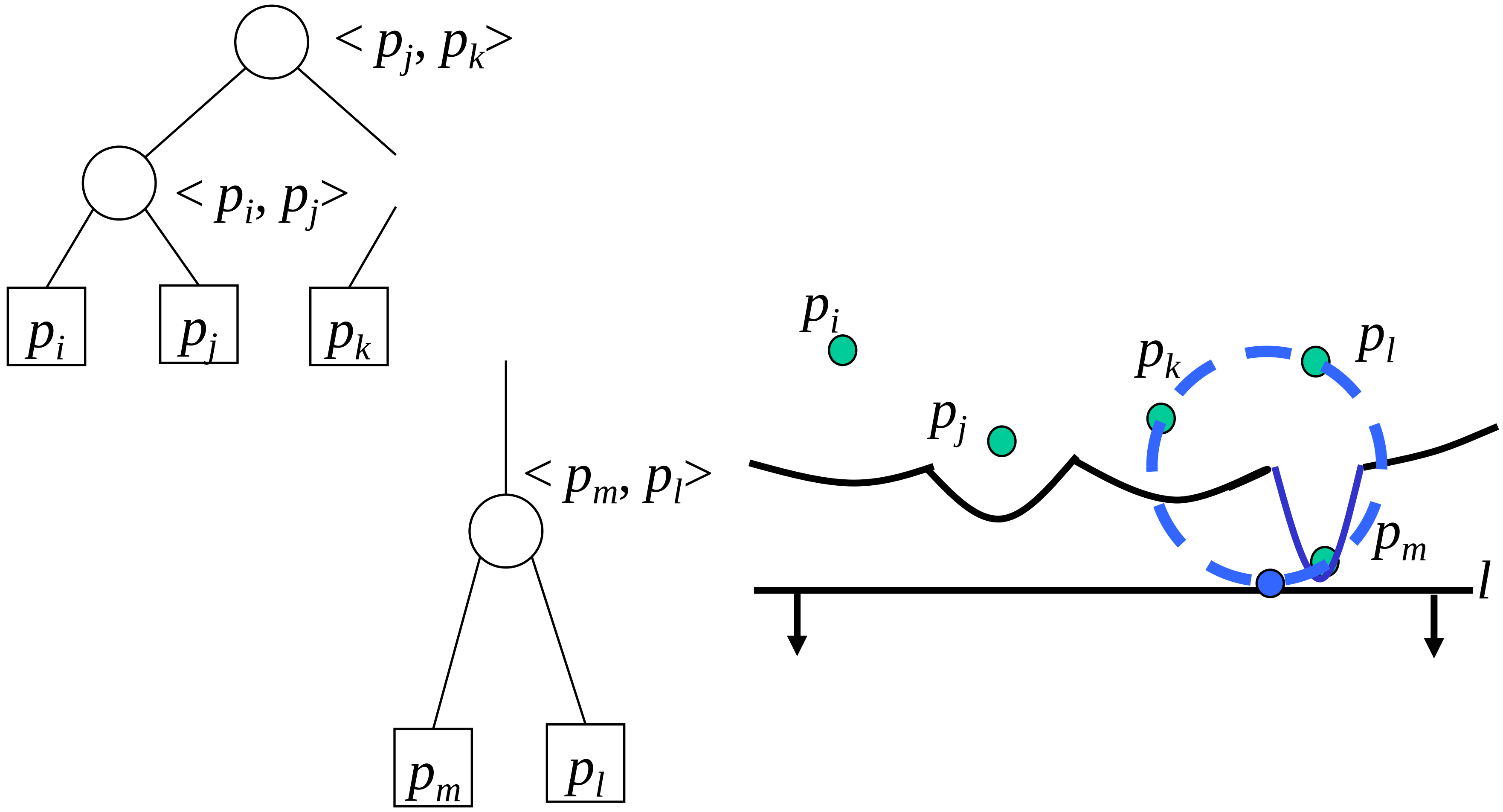
A Circle Event



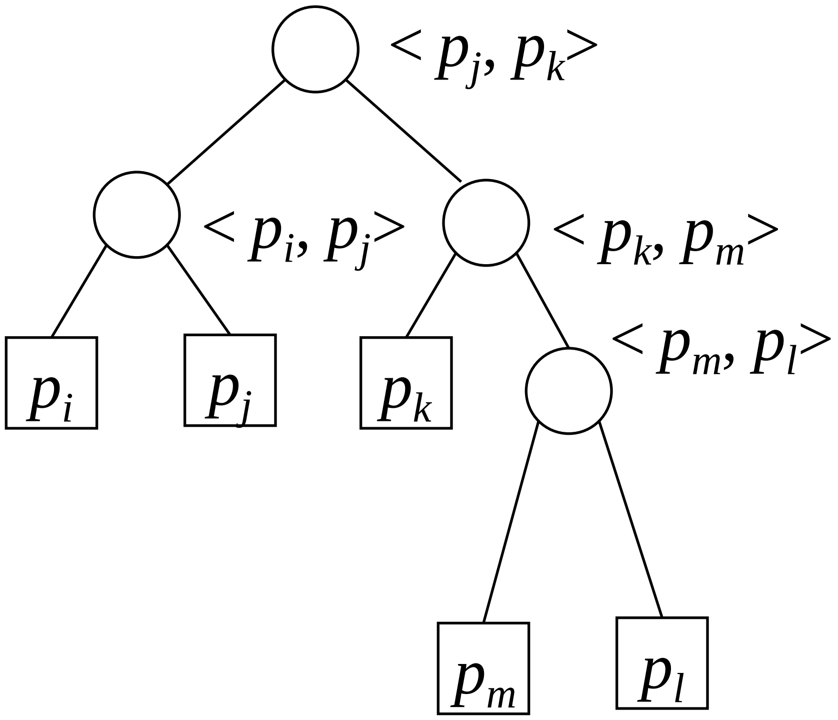
Add vertex to corresponding edge record

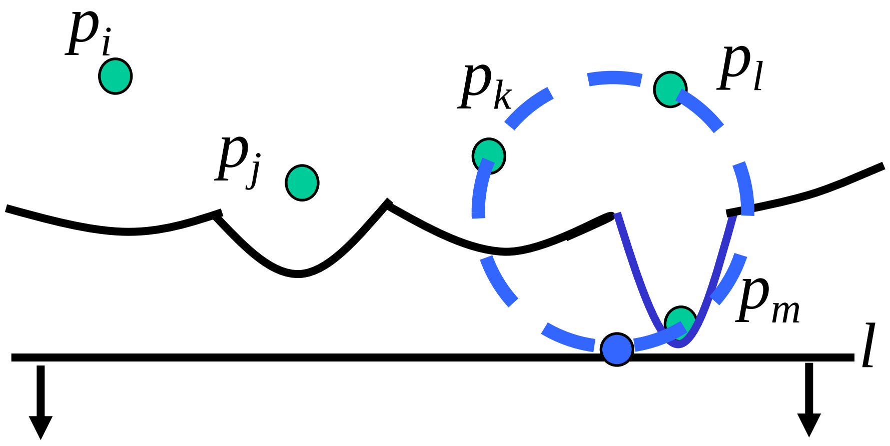


Deleting disappearing arc

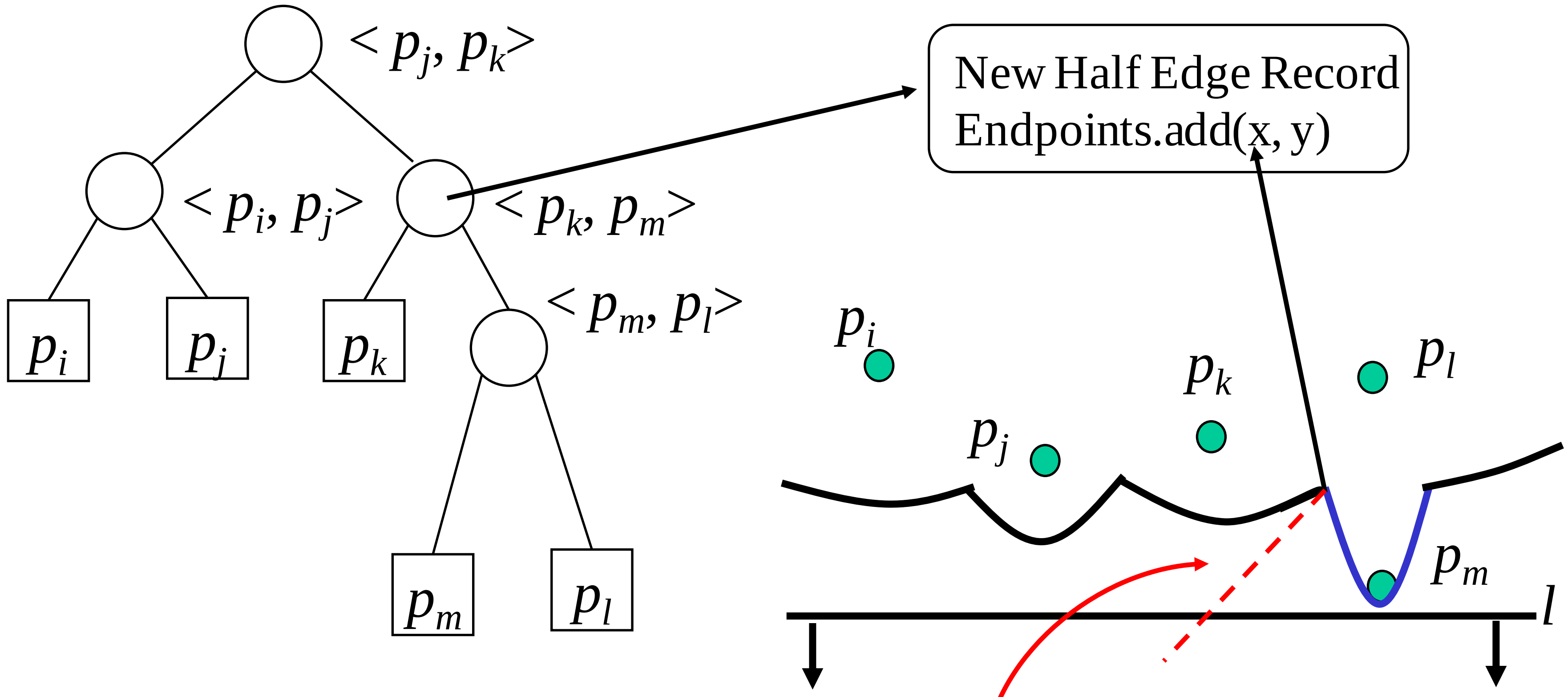


Deleting disappearing arc





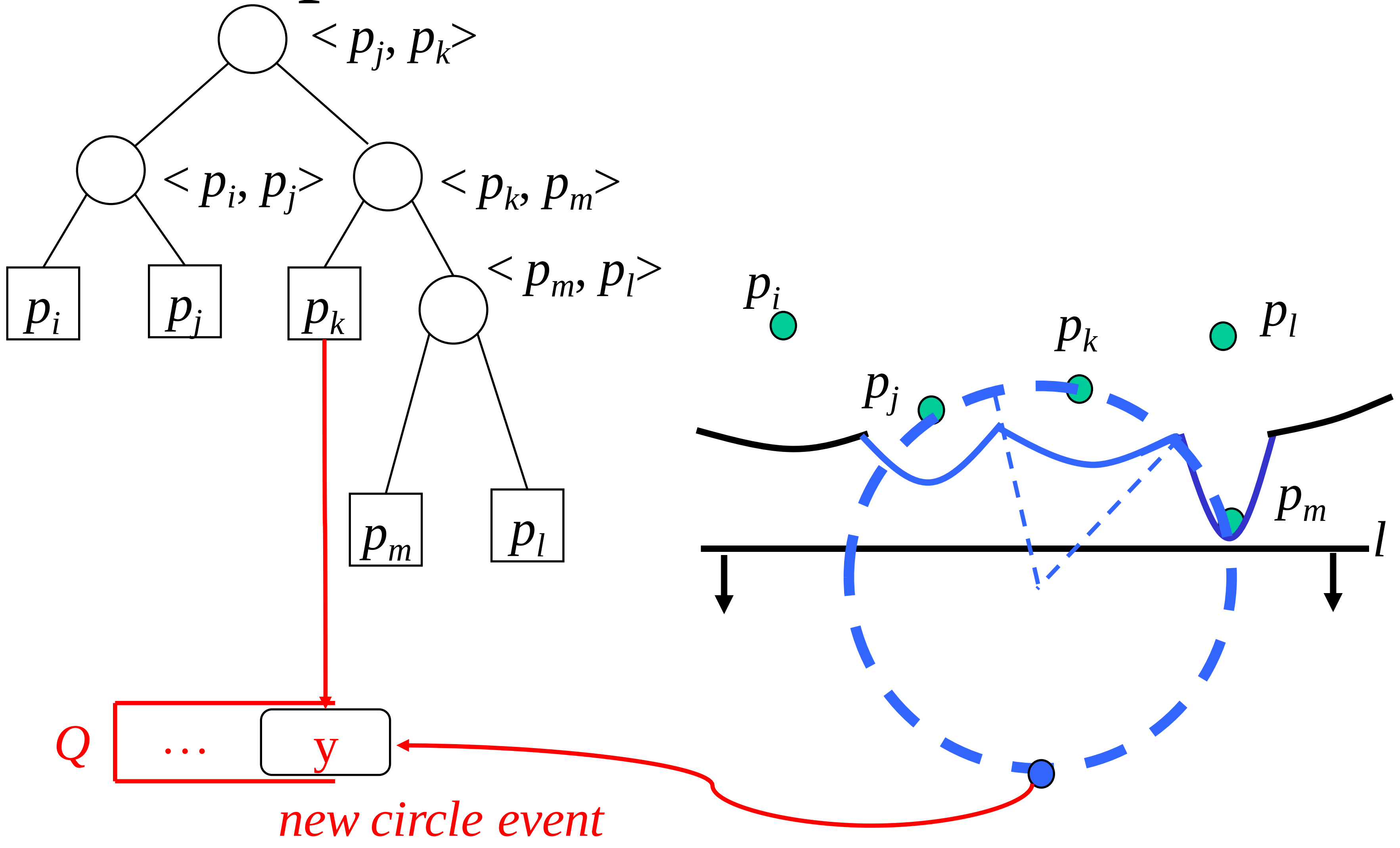
Create new edge record



New Half Edge Record
Endpoints.add(x, y)

A new edge is traced out by the new break point $< p_k, p_m >$

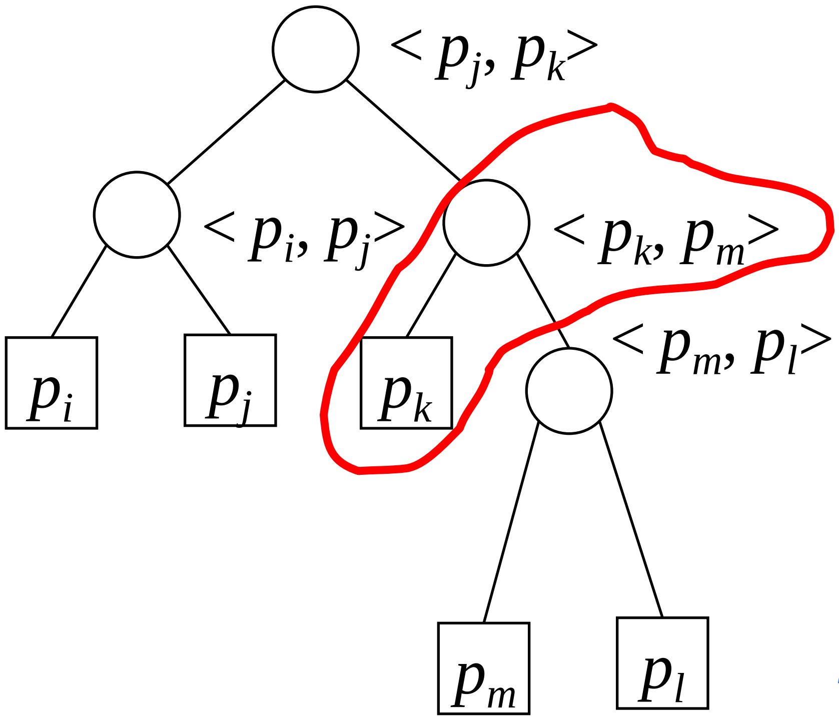
Check the new triplets for potential circle events

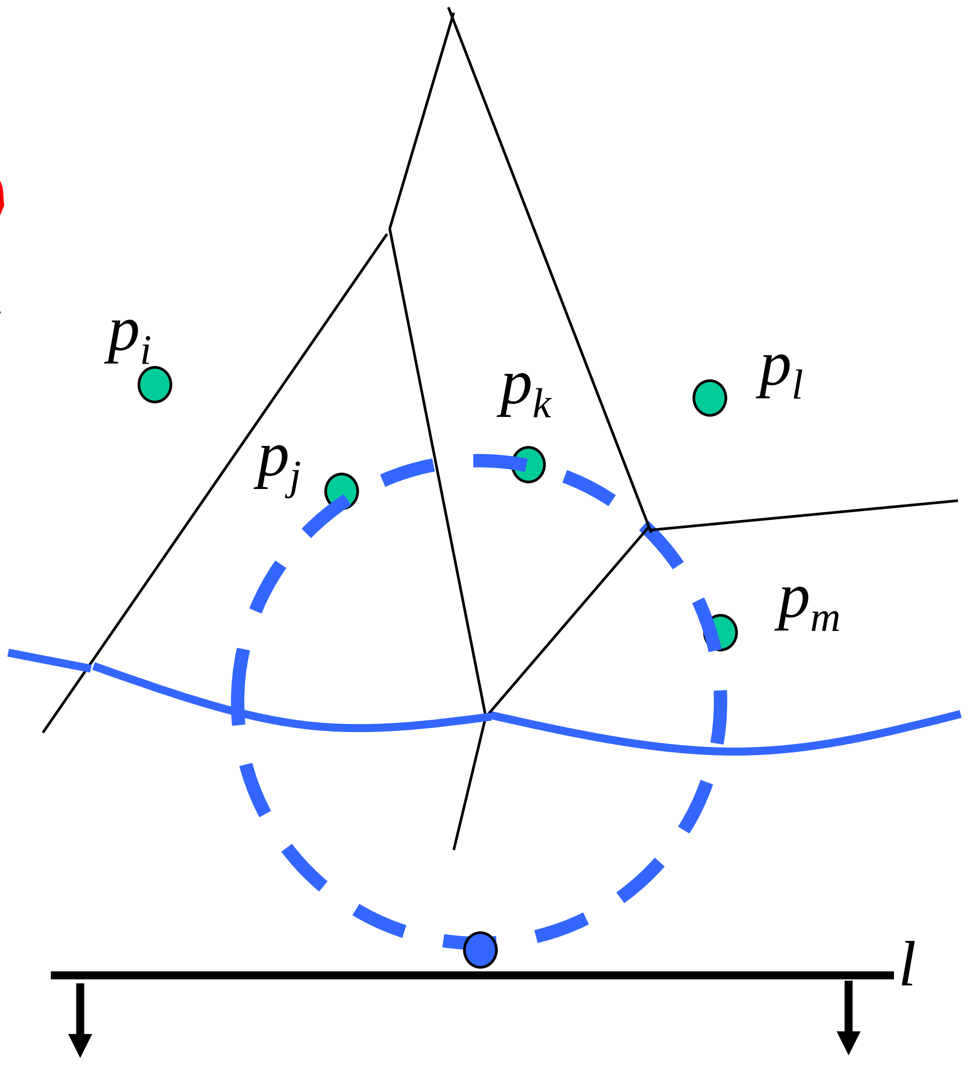


Minor Detail

- Algorithm terminates when $Q = \emptyset$, but the beach line and its break points continue to trace the Voronoi edges
  - Terminate these “half-infinite” edges via a bounding box

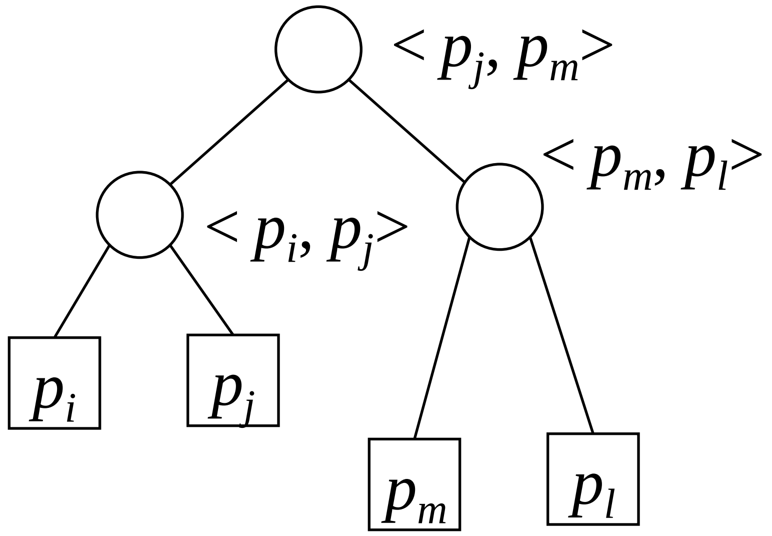
Algorithm Termination

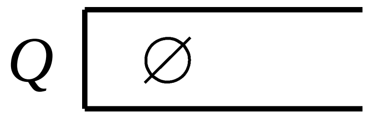


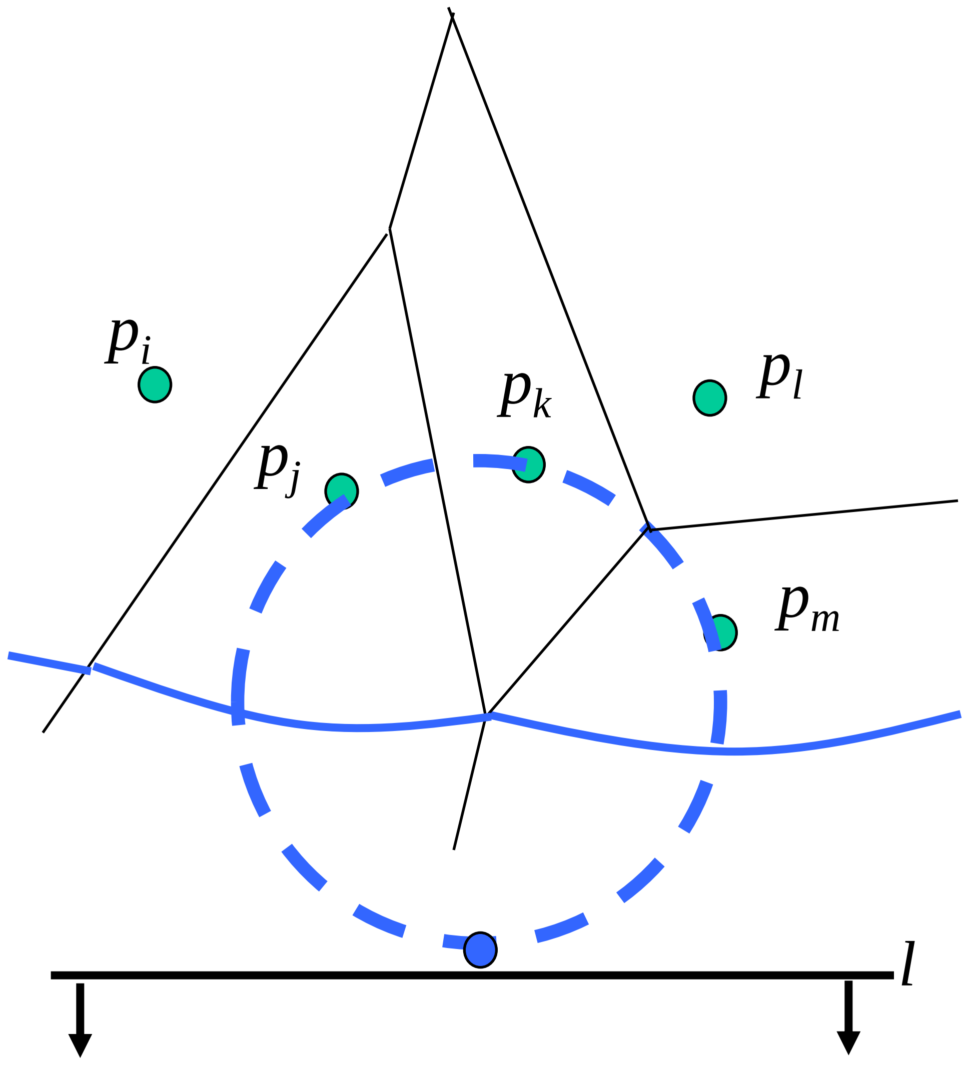


Q $\emptyset$

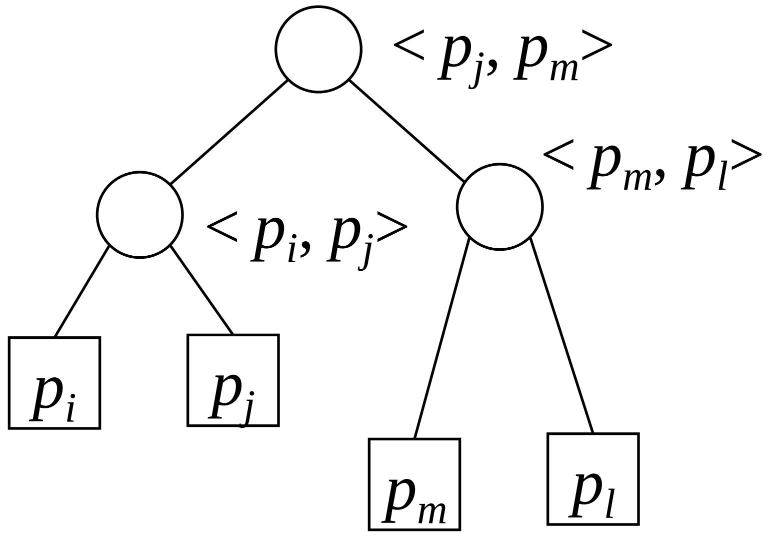
Algorithm Termination

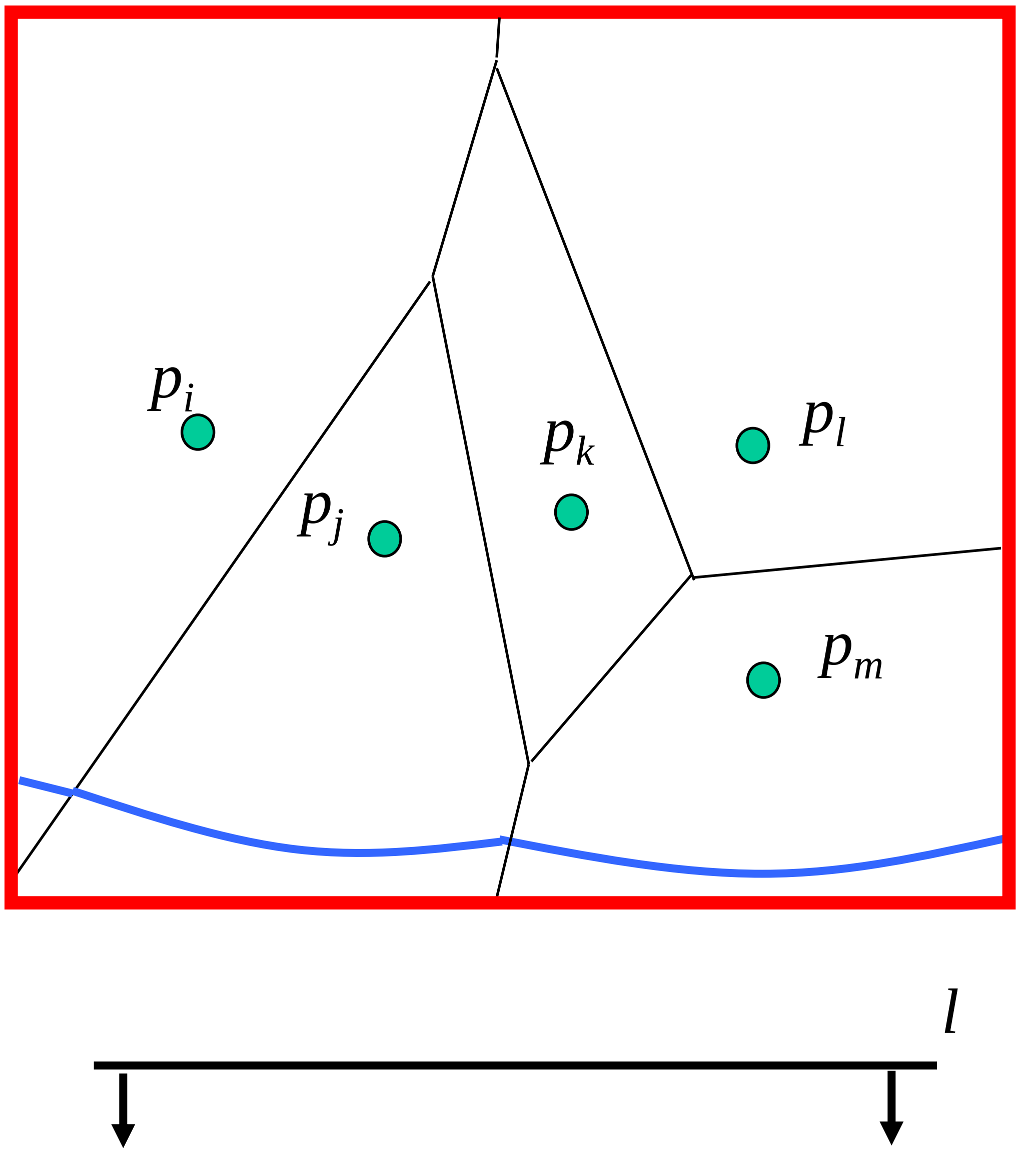




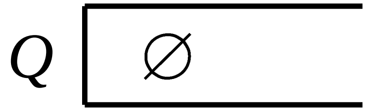


Algorithm Termination





Terminate half-lines with a bounding box!



Handling Site Events

	Running Time
1. Locate the leaf representing the existing arc that is above the new site	O(log *n*)
– Delete the potential circle event in the event queue	
2. Break the arc by replacing the leaf node with a sub tree representing the new arc and break points	O(1)
3. Add a new edge record in the link list	O(1)
4. Check for potential circle event(s), add them to queue if they exist	O(1)
– Store in the corresponding leaf of T a pointer to the new circle event in the queue	

Handling Circle Events

		Running Time
1.	Delete from T the leaf node of the disappearing arc and its associated circle events in the event queue	O(log *n*)
2.	Add vertex record in doubly link list	O(1)
3.	Create new edge record in doubly link list	O(1)
4.	Check the new triplets formed by the former neighboring arcs for potential circle events	O(1)

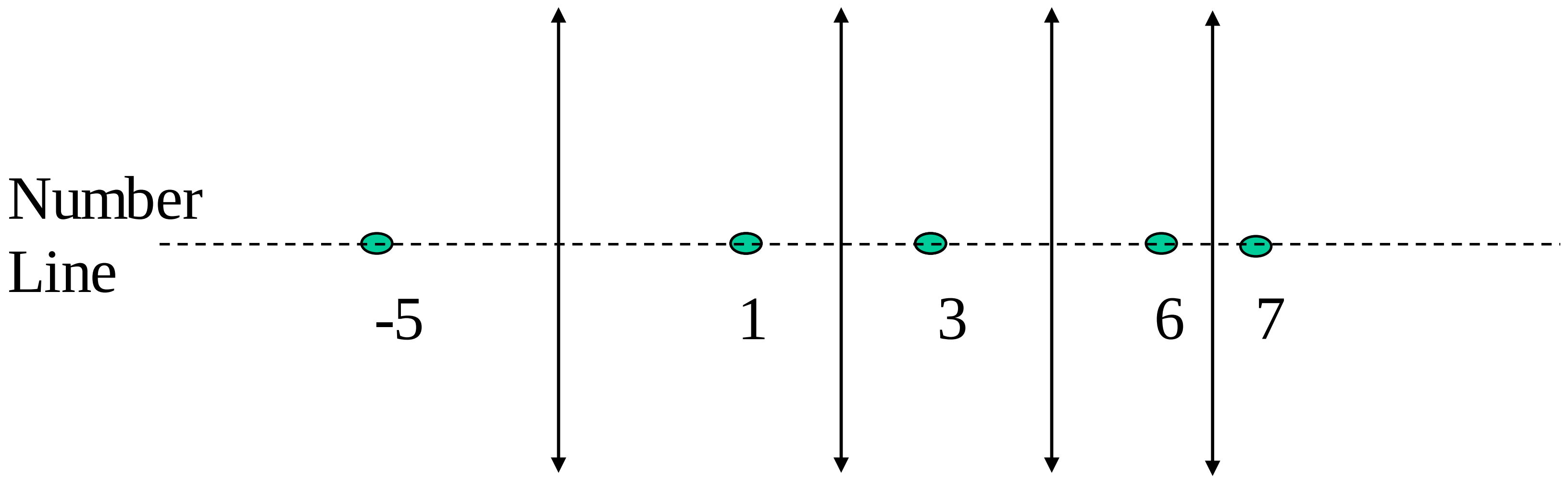
Total Running Time

- Each new site can generate at most two new arcs
  - → beach line can have at most $2n - 1$ arcs
  - → at most O(n) site and circle events in the queue
- Site/Circle Event Handler O(log n)

→ O(n log n) total running time

Is Fortune’s Algorithm Optimal?

- We can sort numbers using any algorithm that constructs a Voronoi diagram!



- Map input numbers to a position on the number line. The resulting Voronoi diagram is doubly linked list that forms a chain of unbounded cells in the left-to-right (sorted) order.

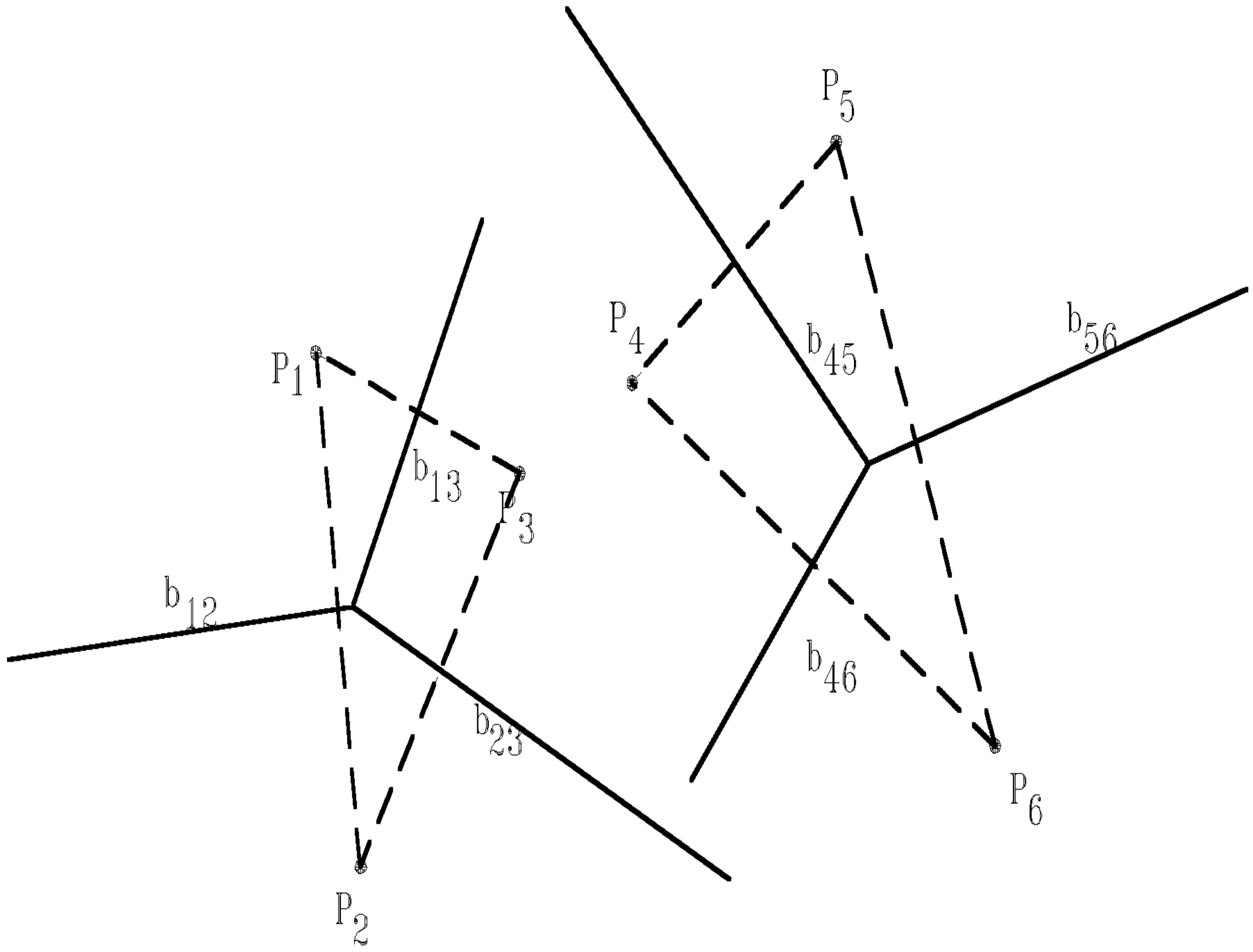
Divide-and-Conquer approach

- Input : A set S of n planar points.
- Output : The Voronoi diagram of S.
- Step 1 If S contains less than 4 point, solve directly and return.
- Step 2 Find a median line L perpendicular to the X-axis which divides S into S_L and S_R such that S_L (S_R) lies to the left(right) of L and the sizes of S_L and S_R are equal.

Divide-and-Conquer approach

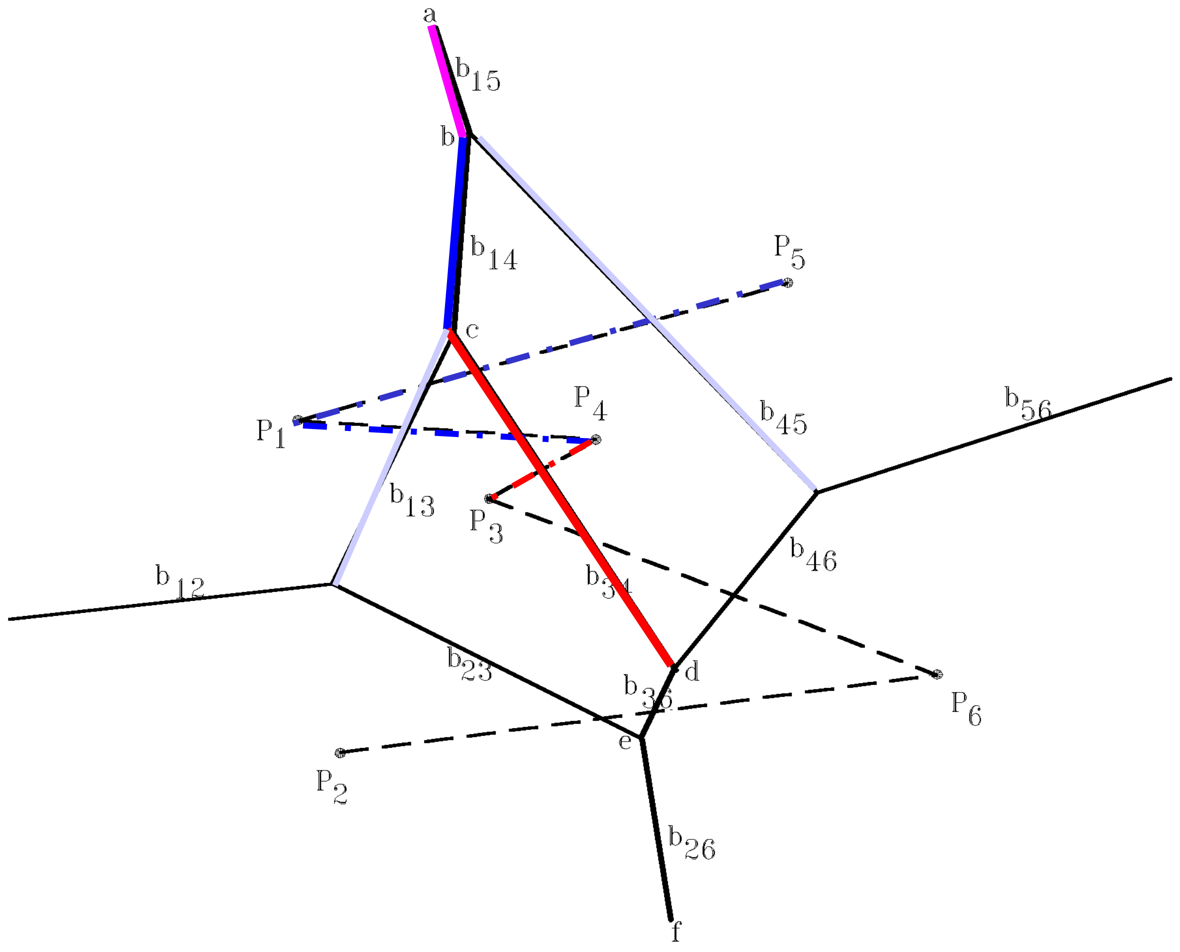
- Step 3 Construct Voronoi diagrams of S_L and S_R recursively. Denote these Voronoi diagrams by $VD(S_L)$ and $VD(S_R)$.
- Step 4 Construct a dividing piece-wise linear hyperplane HP which is the locus of points simultaneously closest to a point in S_L and a point in S_R. Discard all segments of $VD(S_L)$ which lie to the right of HP and all segments of $VD(S_R)$ that lie to the left of HP. The resulting graph is the Voronoi diagram of S.

How to merge two Voronoi diagrams ?



How to merge two Voronoi diagrams ?

- Merging:



b15→b45→b14→b13→b34→b46→b36→b23→b26

Merges Two Voronoi Diagrams into One Voronoi Diagram

- Input : (a) S_L and S_R where S_L and S_R are divided by a perpendicular line L.

  (b) VD(S_L) and VD(S_R).
- Output : VD(S) where S = $S_L \cap S_R$
- Step 1 Find the convex hulls of S_L and S_R . Let them be denoted as Hull(S_L) and Hull(S_R), respectively. (A special algorithm for finding a convex hull in this case will by given later.)

Merges Two Voronoi Diagrams into One Voronoi Diagram

- Step 2 Find segments $\overline{P_aP_b}$ and $\overline{P_cP_d}$ which join HULL(S_L) and HULL(S_R) into a convex hull (P_a and P_c belong to S_L and P_b and P_d belong to S_R) Assume that $\overline{P_aP_b}$ lies above $\overline{P_cP_d}$. Let x = a, y = b, SG= $\overline{P_xP_y}$ and HP = $\varnothing$.
- Step 3 Find the perpendicular bisector of SG. Denote it by BS. Let HP = HP.{BS}. If SG = $\overline{P_cP_d}$, go to Step 5; otherwise, go to Step 4.

Merges Two Voronoi Diagrams into One Voronoi Diagram

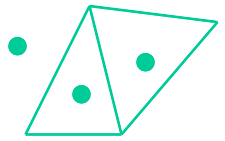
- Step 4 The ray from VD(S_L) and VD(S_R) which BS first intersects with must be a perpendicular bisector of either $\overline{P_xP_z}$ or $\overline{P_yP_z}$ for some z. If this ray is the perpendicular bisector of $\overline{P_yP_z}$, then let SG = $\overline{P_xP_z}$; otherwise, let SG = $\overline{P_zP_y}$. Go to Step 3.
- Step 5 Discard the edges of VD(S_L) which extend to the right of HP and discard the edges of VD(S_R) which extend to the left of HP. The resulting graph is the Voronoi diagram of S = S_L.S_R.

Merges Two Voronoi Diagrams into One Voronoi Diagram

- **Def :** Given a point P and a set S of points, the distance between P and S is the distance between P and P_i which is the nearest neighbor of P in S.
- The HP obtained from the above algorithm is the locus of points which keep equal distances to S_L and S_R .
- The HP is monotonic in y.

Merges Two Voronoi Diagrams into One Voronoi Diagram

- # of edges of a Voronoi diagram $\leq$ 3n - 6, where n is # of points.
- Reasoning:
  i. # of edges of a planar graph with n vertices $\leq$ 3n - 6.
  ii. A Delaunay triangulation is a planar graph.
  iii. Edges in Delaunay triangulation $\xleftrightarrow{1-1}$ edges in Voronoi diagram.



Construct Convex Hull from Voronoi diagram

- After a Voronoi diagram is constructed, a convex hull can by found in O(n) time.

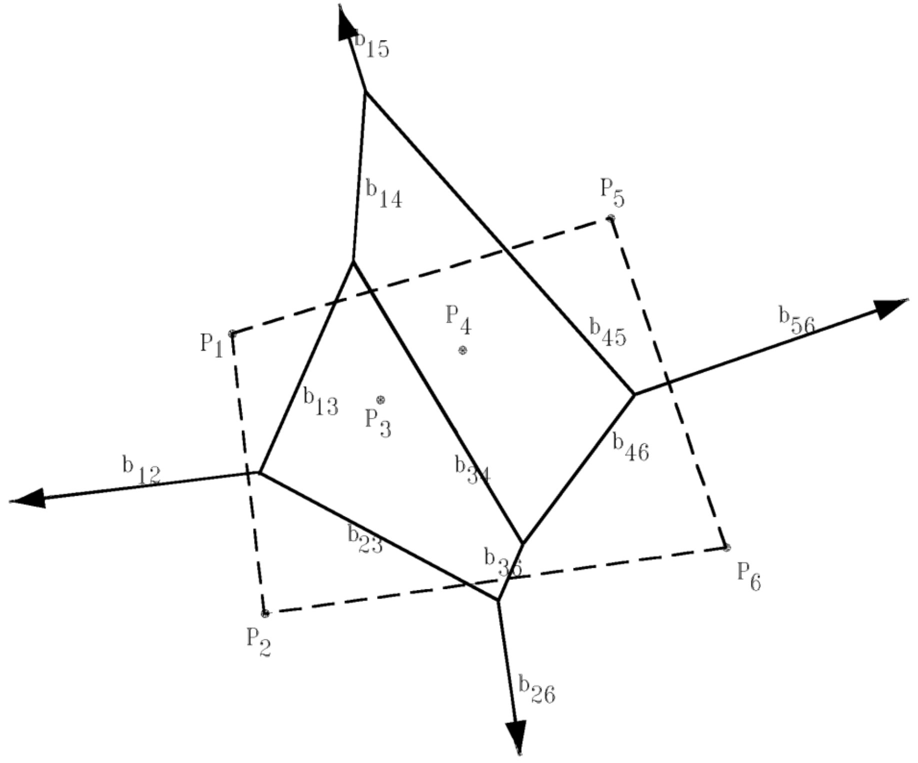
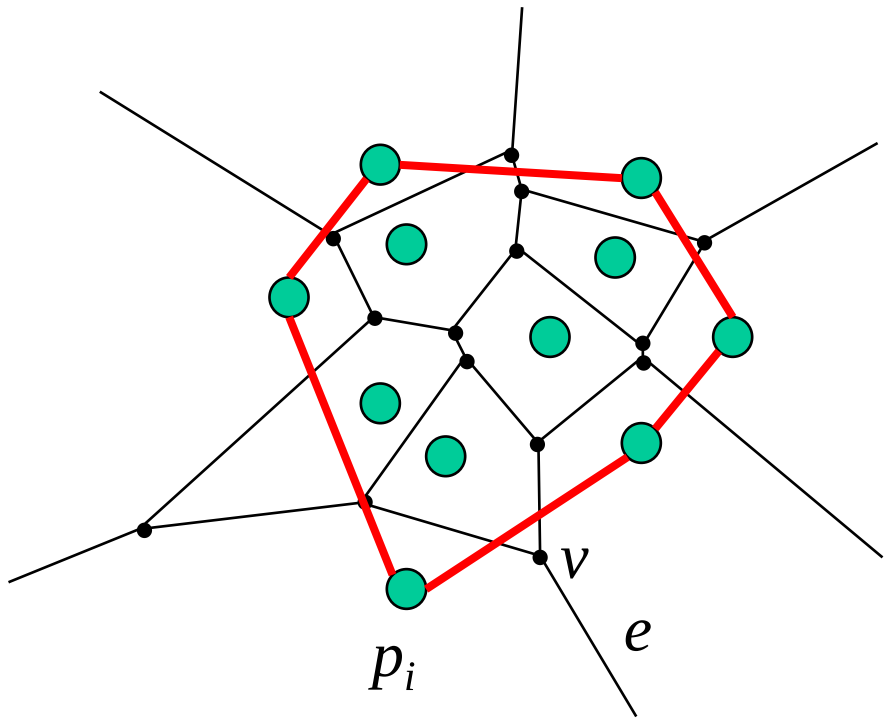


Fig. 5-25: Constructing a Convex Hull from a Voronoi Diagram

Voronoi Diagram/Convex Hull Duality

Sites sharing a half-infinite edge are convex hull vertices



Construct Convex Hull from Voronoi diagram

- Step 1 : Find an infinite ray by examining all Voronoi edges.
- Step 2 : Let P_i be the point to the left of the infinite ray. P_i is a convex hull vertex. Examine the Voronoi polygon of P_i to find the next infinite ray.
- Step 3 : Repeat Step 2 until we return to the Starting ray.

Time complexity

- Time complexity for merging 2 Voronoi diagrams:
  - Step 1: O(n)
  - Step 2: O(n)
  - Step 3 ~ Step 5: O(n)

  (at most 3n - 6 edges in VD(S_L) and VD(S_R) and at most n segments in HP)

  $\Rightarrow$ T(n) = 2T(n/2) + O(n)=O(n log n)

Applications 1(Static Point Set)

Closest pair of points:

Go through edge list for *VD(P)* and determine minimum

All next neighbors :

Go through edge list for *VD(P)* for all points and get next neighbors in each case

Minimum Spanning tree (after Kruskal)

1. Each point *p* from *P* defines 1-element set of
2. More than a set of *T* exists

   2.1) find *p,p´* with *p* in *T* and *p´* not in *T* with *d(p, p´)*
   minimum.

   2.2) connect *T* and *p´* contained in *T´* (union)

Theorem: The MST can be computed in time *O(n log n)*

Applications (dynamic object set)

Search for next neighbor :

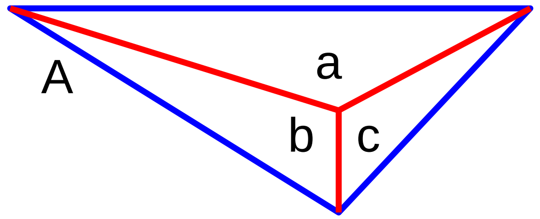
Idea : Hierarchical subdivision of *VD(P)*

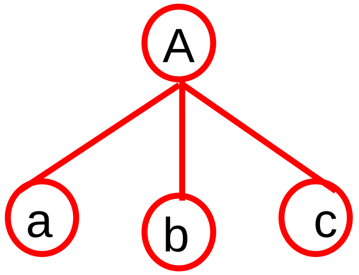
Step 1 : Triangulation of final Voronoi regions

Step 2 : Summary of triangles and structure of a search tree

Rule of Kirkpatrick :

Remove in each case points with degree < 12,
its neighbor is already far.





Theorem: Using the rule of Kirkpatrick a search tree
of logarithmic depth develops.

Degenerate Cases

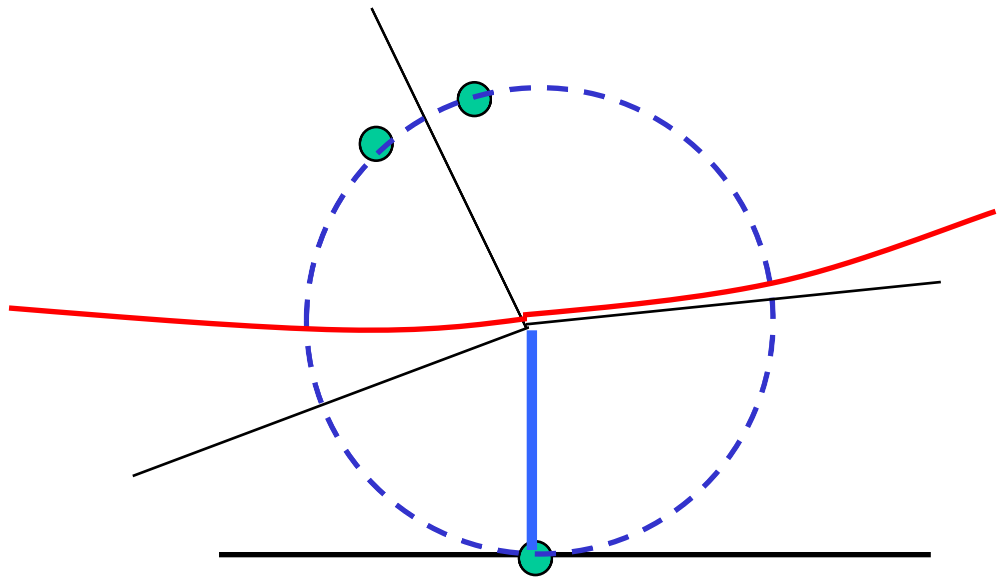
- Events in Q share the same y-coordinate
  - Can additionally sort them using x-coordinate
- Circle event involving more than 3 sites
  - Current algorithm produces multiple degree 3 Voronoi vertices joined by zero-length edges
  - Can be fixed in post processing

Degenerate Cases

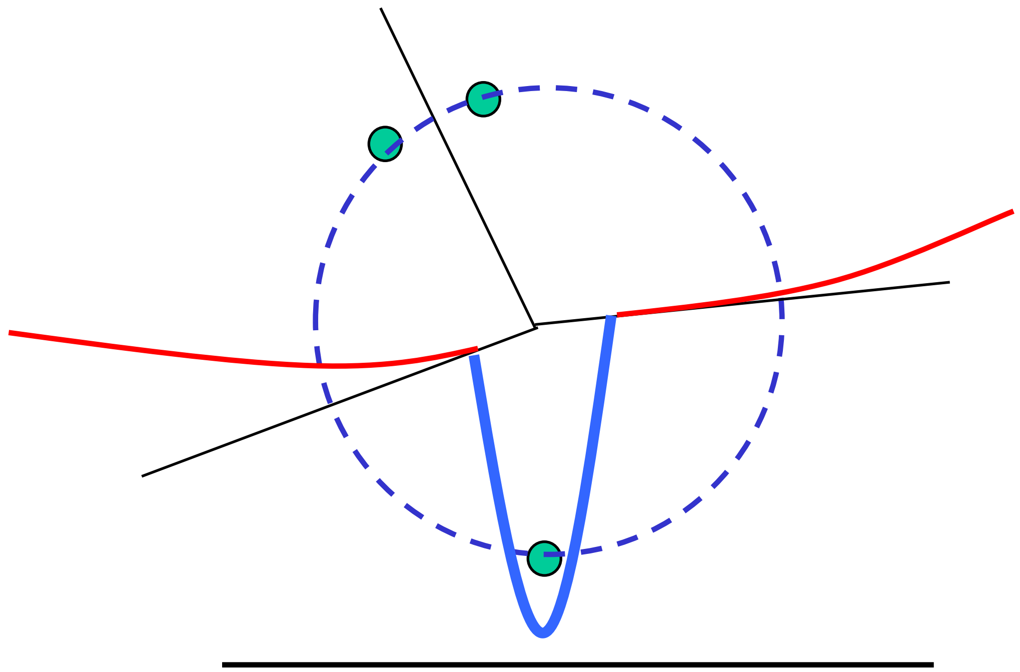
- Site points are collinear (break points neither converge or diverge)
  - Bounding box takes care of this
- One of the sites coincides with the lowest point of the circle event
  - Do nothing

Site coincides with circle event: the same algorithm applies!

1. New site detected
2. Break one of the arcs an infinitesimal distance away from the arc's end point



Site coincides with circle event



Summary

- Voronoi diagram is a useful planar subdivision of a discrete point set
- Voronoi diagrams have linear complexity and can be constructed in O(*n* log *n*) time
- Fortune’s algorithm (optimal)