Construction of Voronoi Diagrams

Outline:

I. Circle event

II. Data structures

III. Handling of circle events

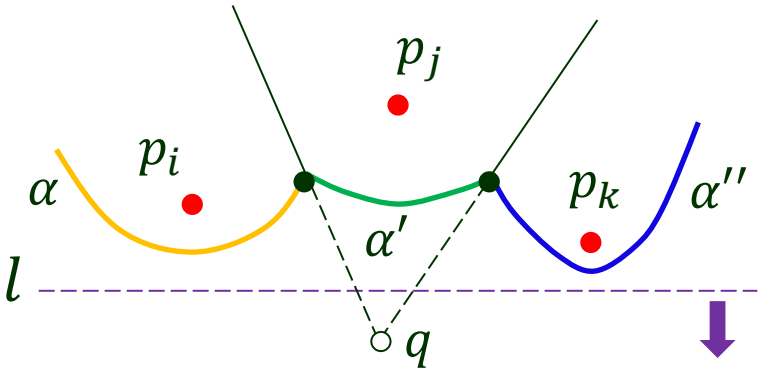
IV. Algorithm and time complexity

V. Handling of degeneracies

VI. Applications

I. Circle Event

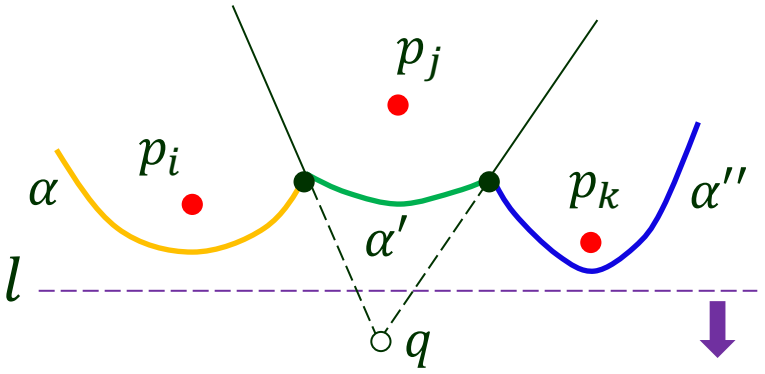
An existing arc shrinks to a point and disappears afterward.



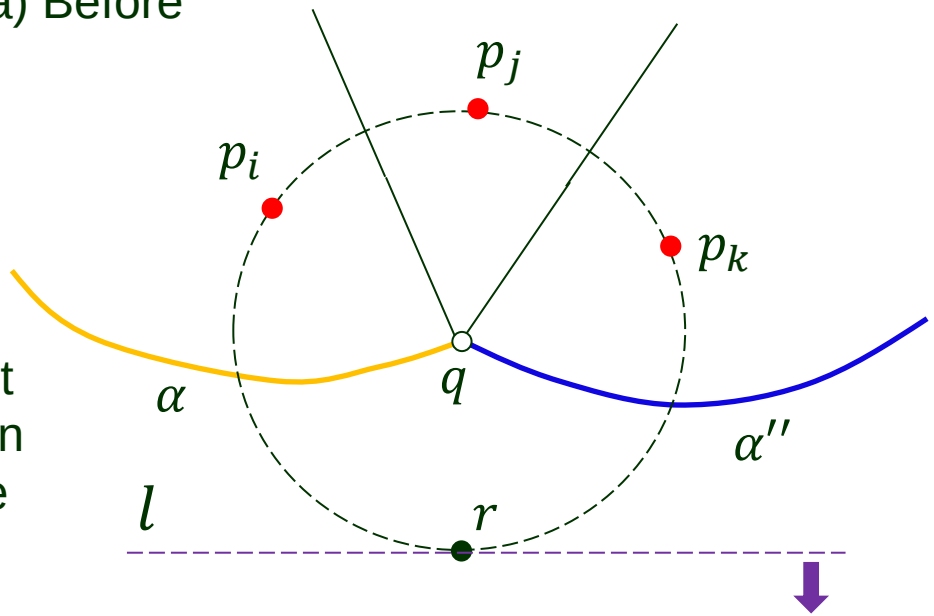
(a) Before

I. Circle Event

An existing arc shrinks to a point and disappears afterward.



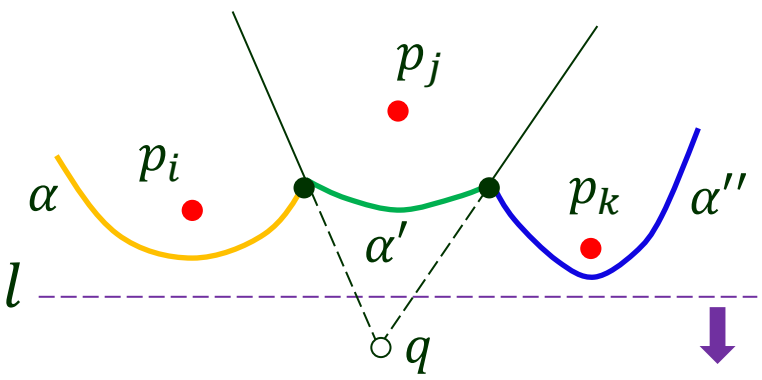
(a) Before



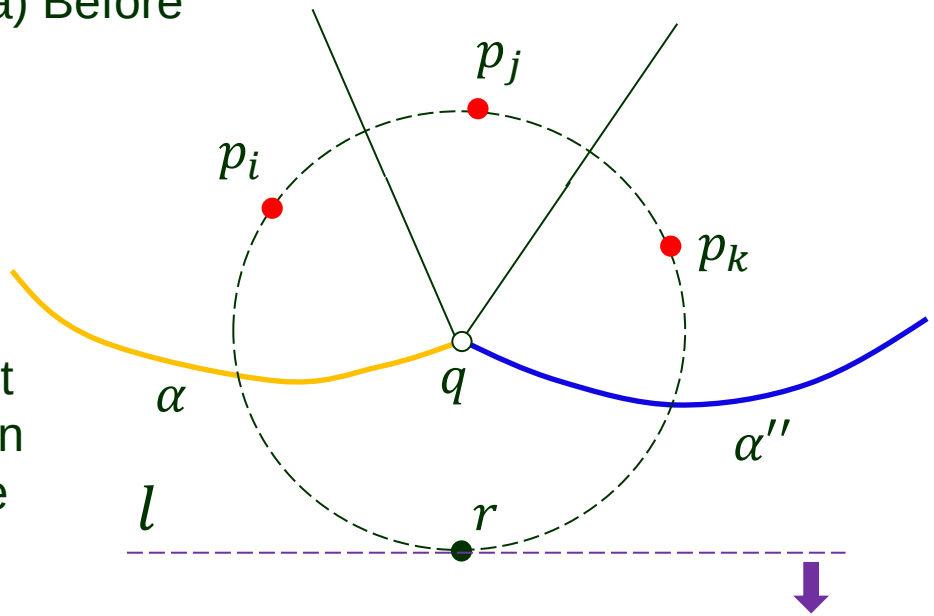
(b) At lowest point r on the circle

I. Circle Event

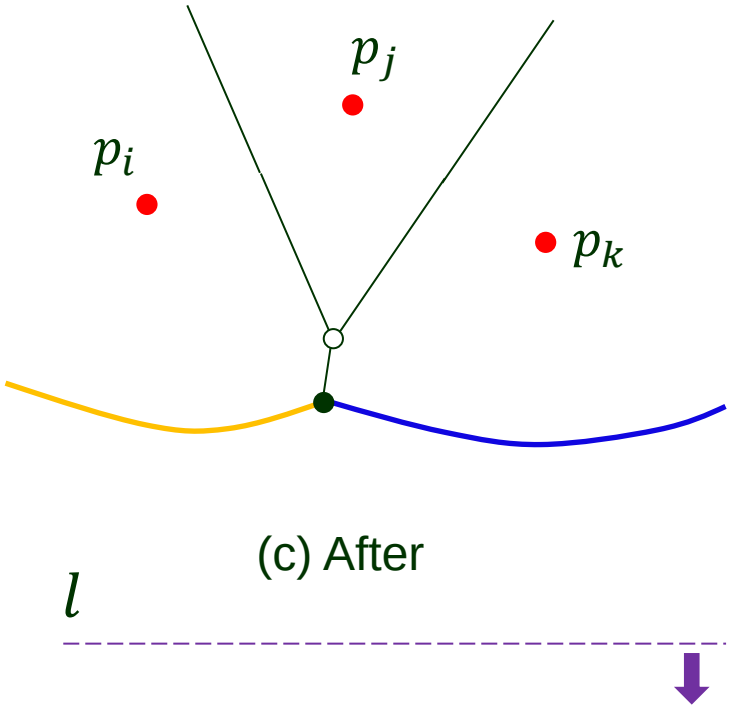
An existing arc shrinks to a point and disappears afterward.



(a) Before



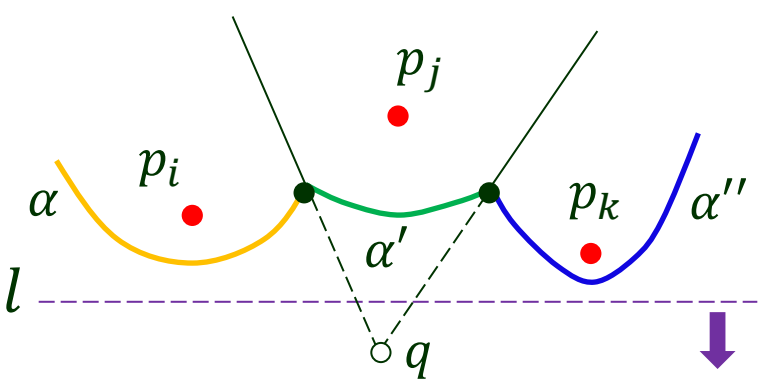
(b) At lowest point r on the circle



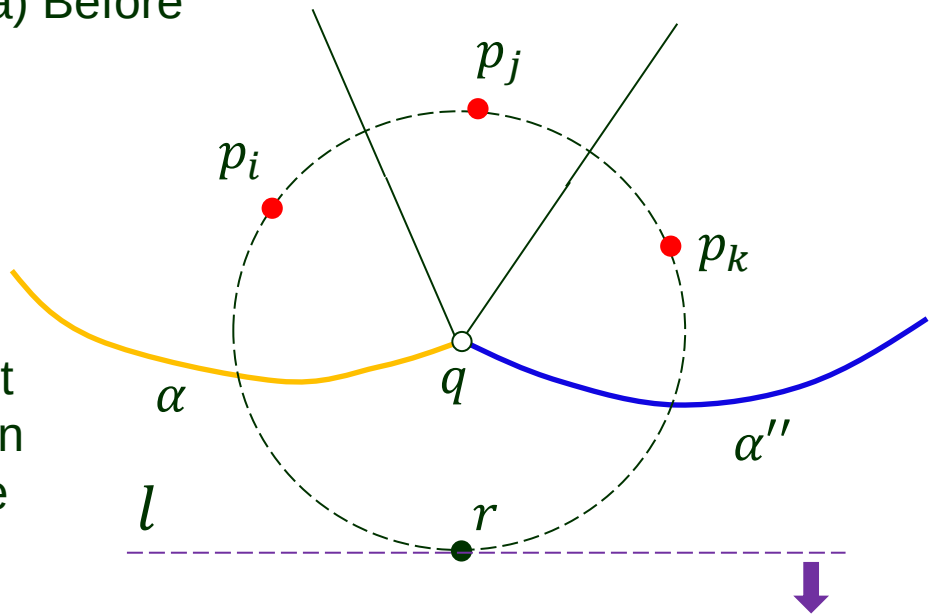
(c) After

I. Circle Event

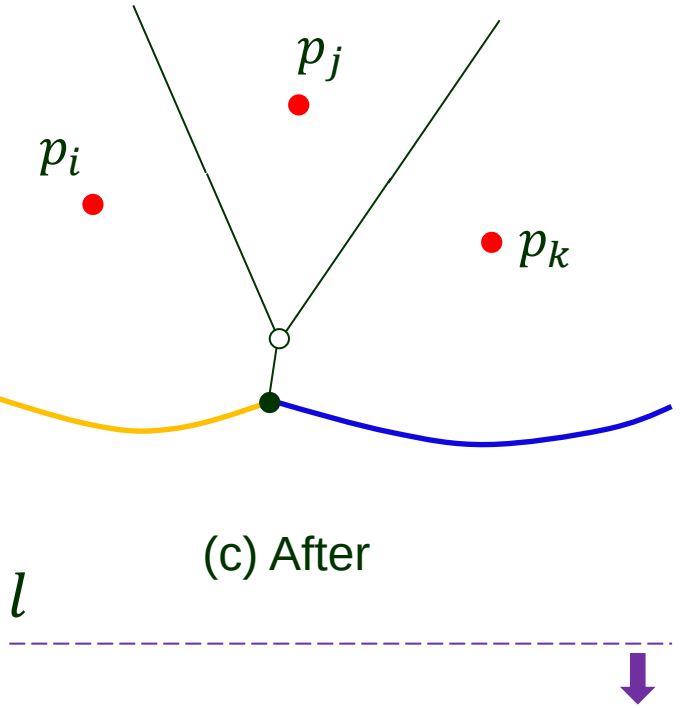
An existing arc shrinks to a point and disappears afterward.



(a) Before



(b) At lowest point r on the circle



(c) After

Event also referred to as $\langle p_i, p_j, p_k \rangle$.

Circle Event Summary

At a circle event:

- An existing arc drops out.
- Two growing Voronoi edges merge at a vertex.

Lemma An existing arc disappears from the beach line only through a circle event.

II. Data Structures

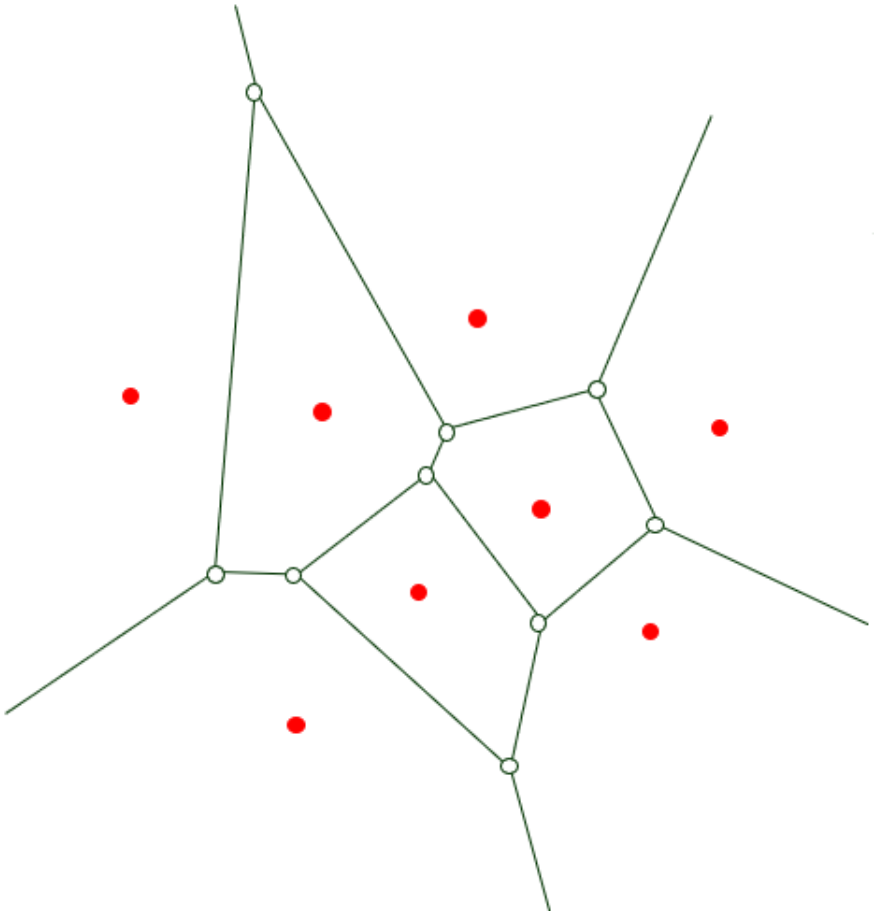
a) A data structure to store part of Vor(P) computed so far.

b) Standard data structures for sweeping.

- event queue
- sweep line status

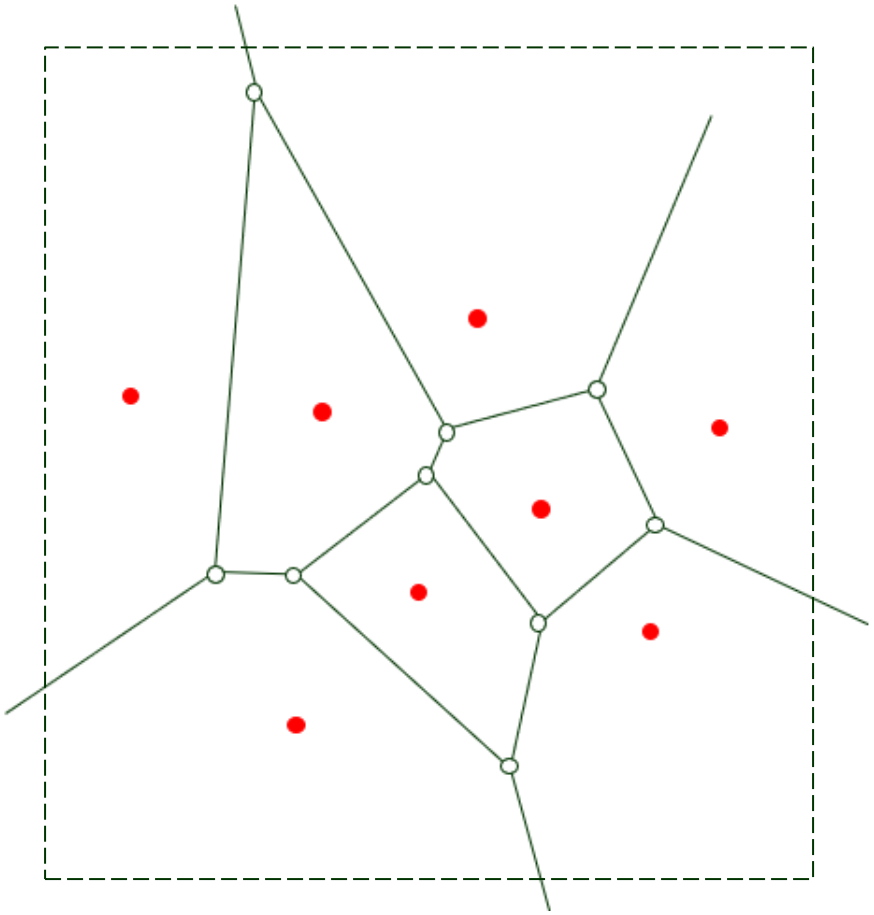
Voronoi Diagram Storage

As a planar subdivision (DCEL)



Voronoi Diagram Storage

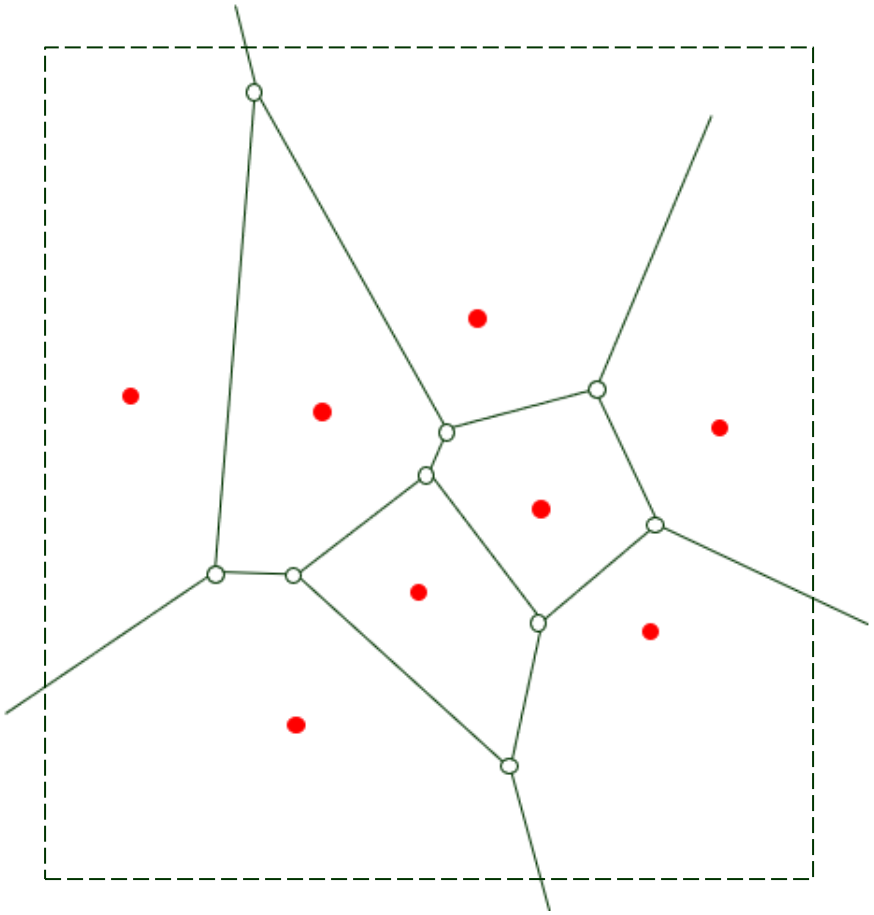
As a planar subdivision (DCEL)



Bounding box

Voronoi Diagram Storage

As a planar subdivision (DCEL)

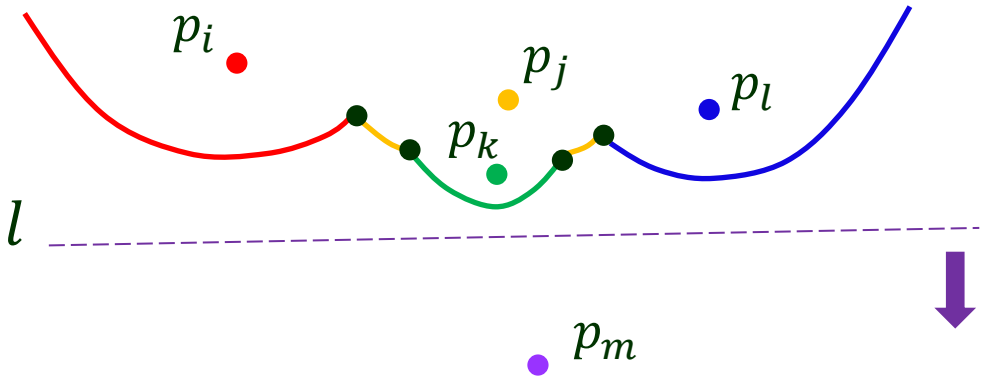


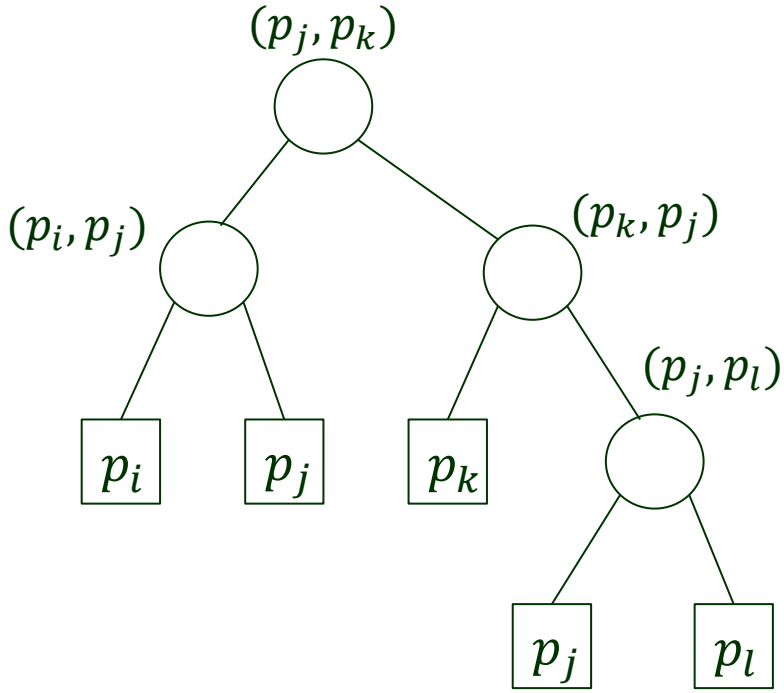
Bounding box

- added after the computation
- large enough to contain all vertices

Beach Line Storage

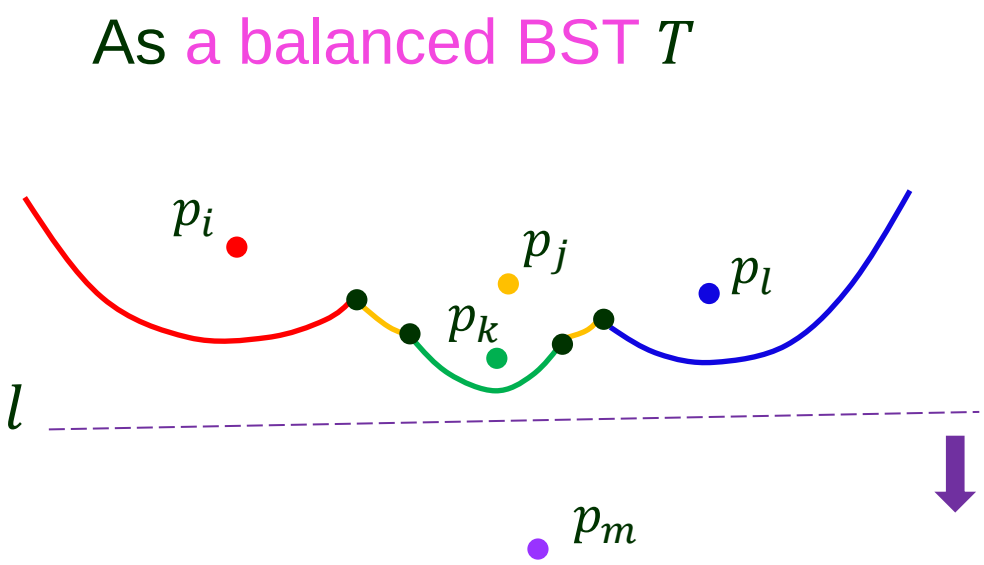
As a balanced BST T

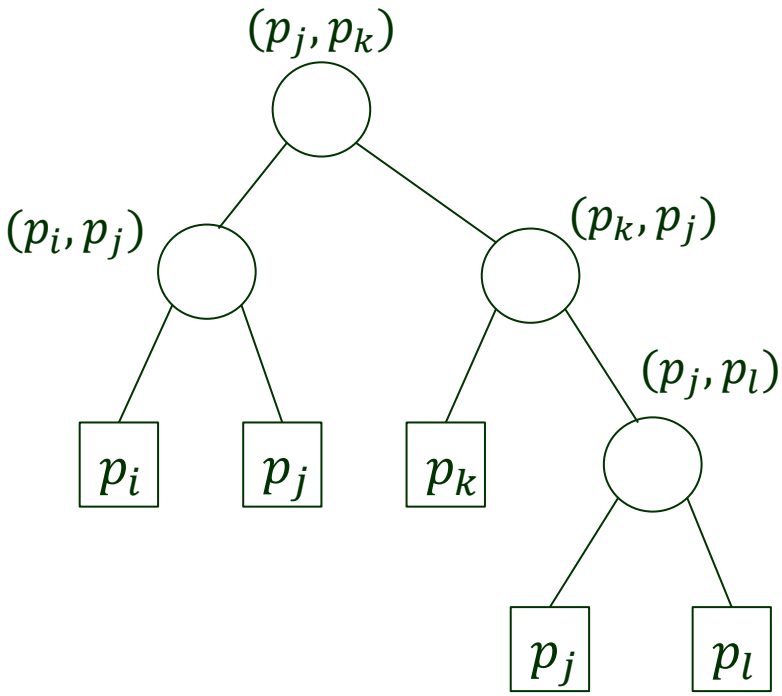




Beach Line Storage

As a balanced BST T

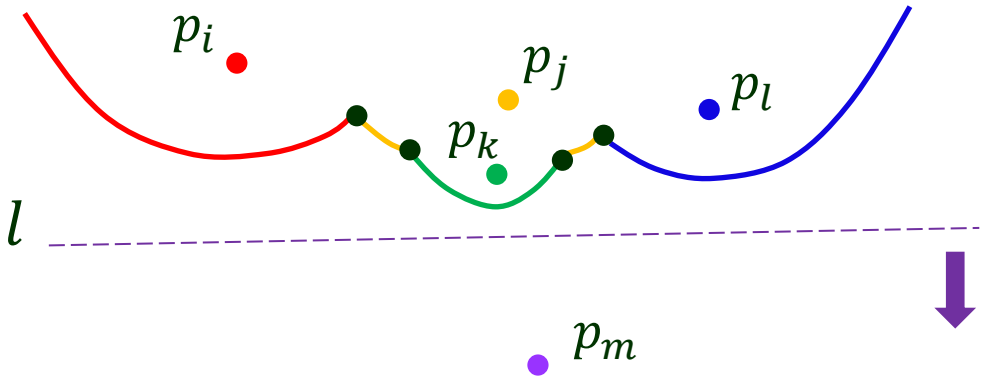


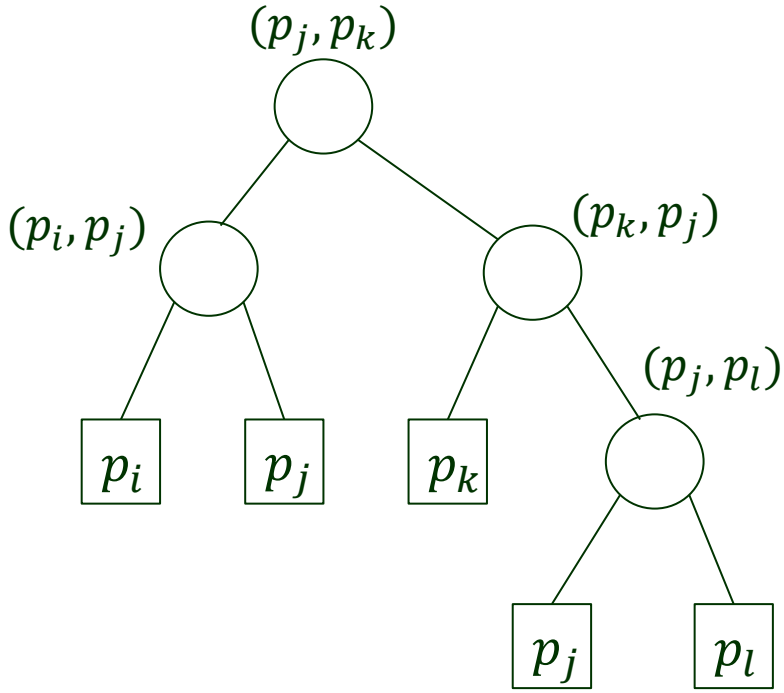


- *Leaves*: arcs of the beach line
  - ordered from left to right
  - labeled with their defining sites

Beach Line Storage

As a balanced BST T



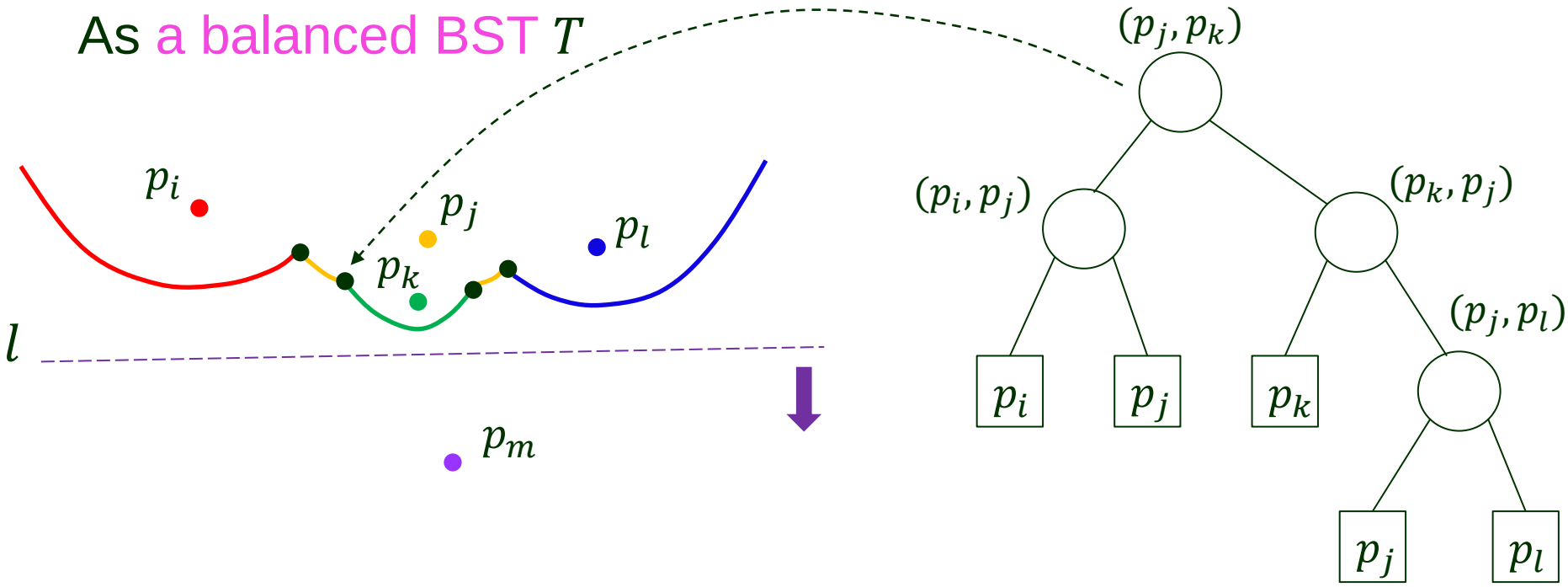


- *Leaves*: arcs of the beach line
  - ordered from left to right
  - labeled with their defining sites

- *Internal nodes*: break points

(p_i, p_j): joining a left arc defined by p_i and a right arc defined by p_j

Beach Line Storage

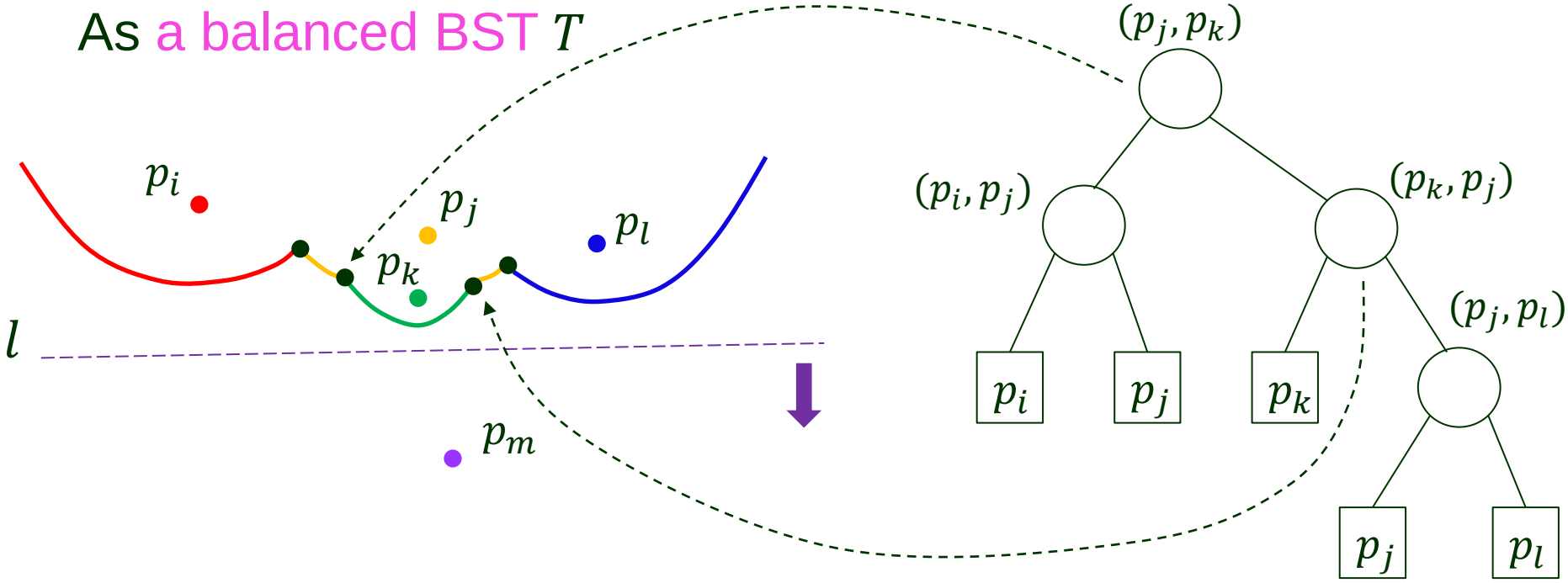


- *Leaves*: arcs of the beach line
  - ordered from left to right
  - labeled with their defining sites

- *Internal nodes*: break points

(p_i, p_j): joining a left arc defined by p_i and a right arc defined by p_j

Beach Line Storage

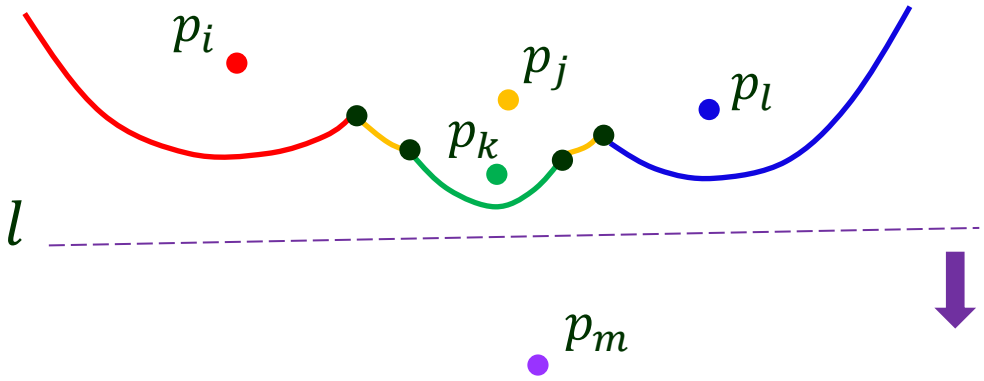


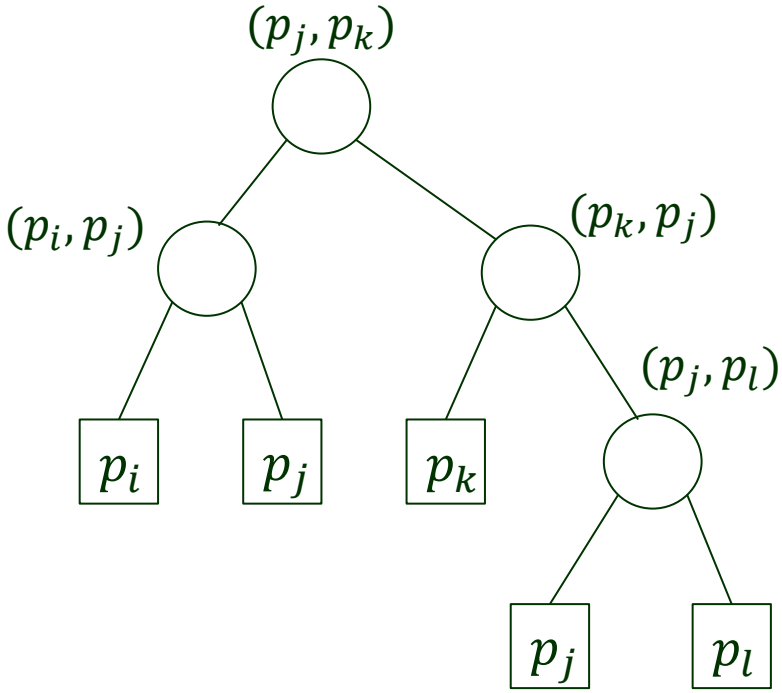
- *Leaves*: arcs of the beach line
  - ordered from left to right
  - labeled with their defining sites

- *Internal nodes*: break points

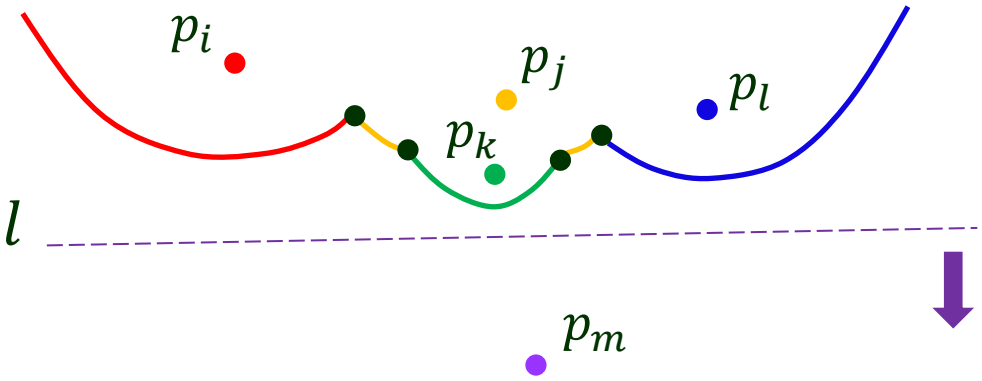
(p_i, p_j): joining a left arc defined by p_i and a right arc defined by p_j

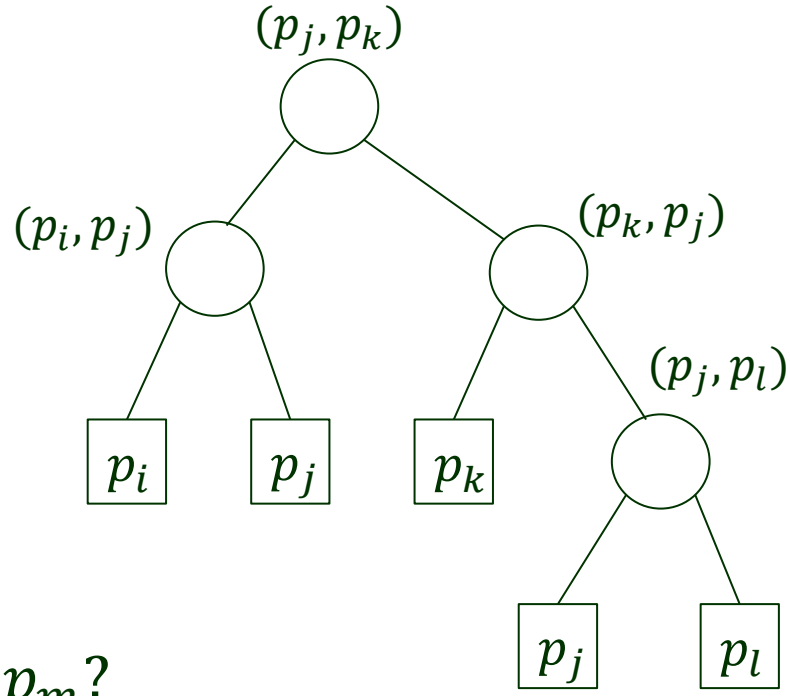
Arc Above a New Site





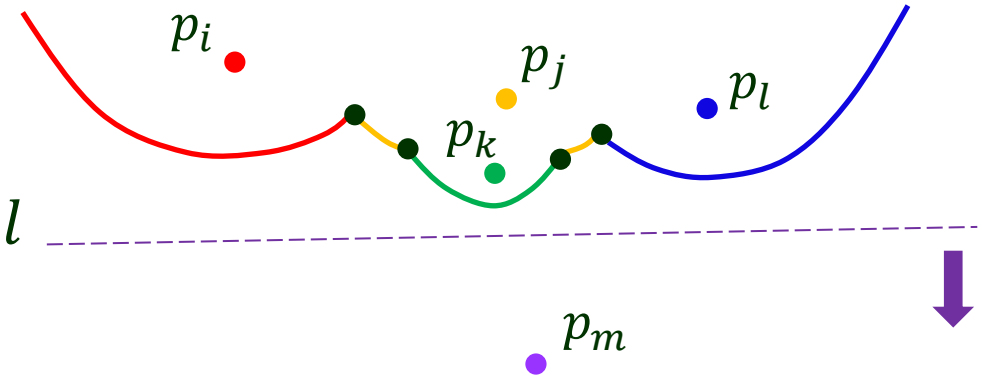
Arc Above a New Site

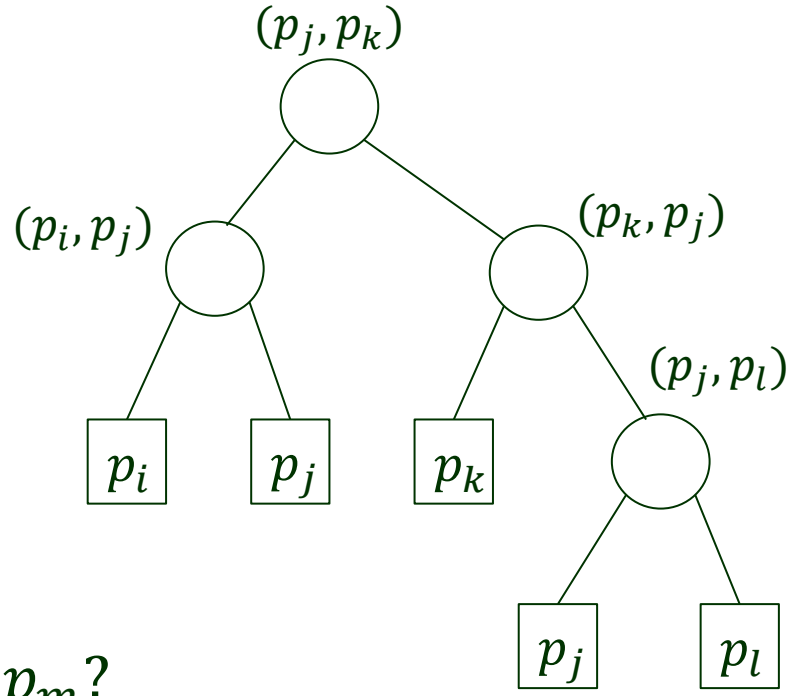




How to find the arc above a new site p_m?

Arc Above a New Site

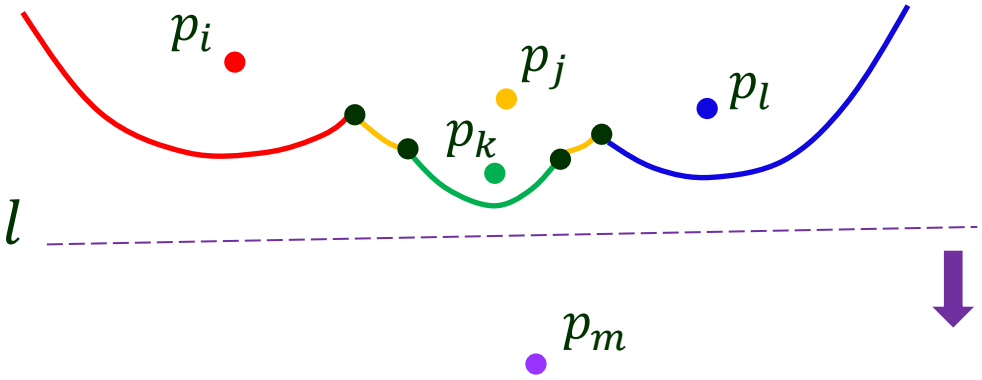


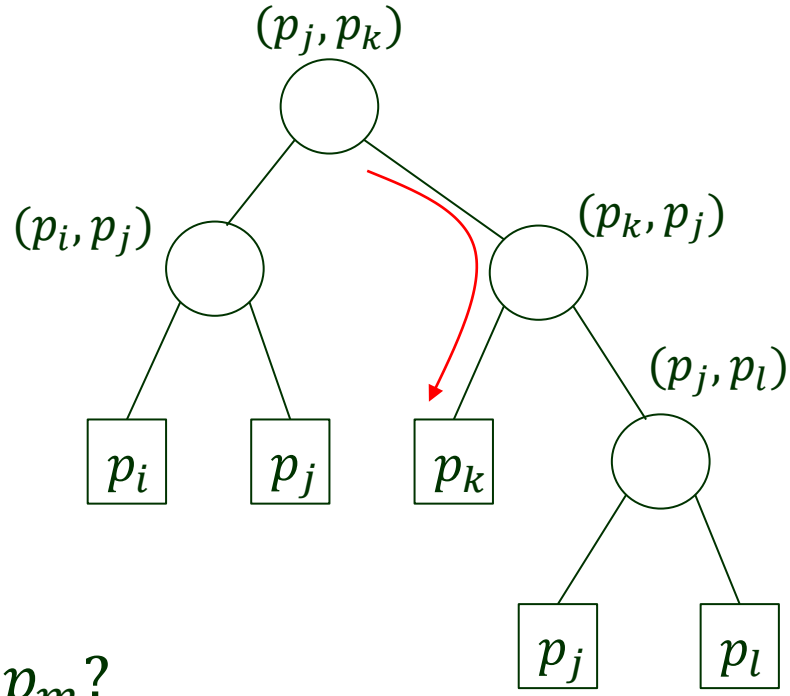


How to find the arc above a new site p_m?

- Compare its x-coordinate with that of the breakpoint (p_i, p_j) stored at an internal node.

Arc Above a New Site

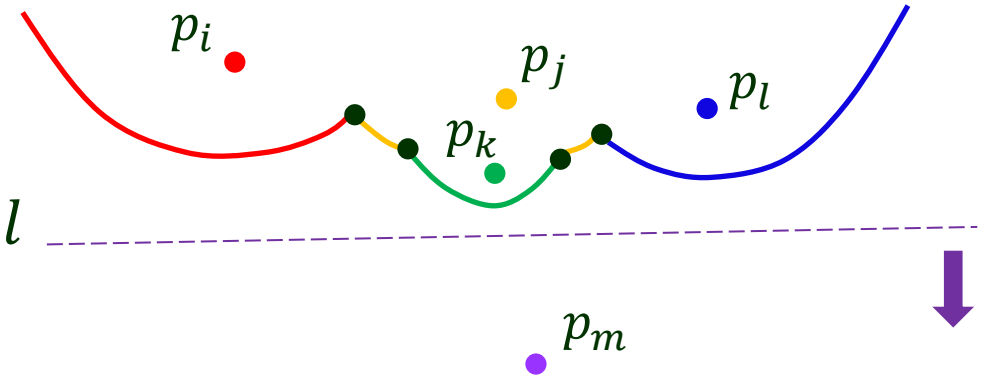


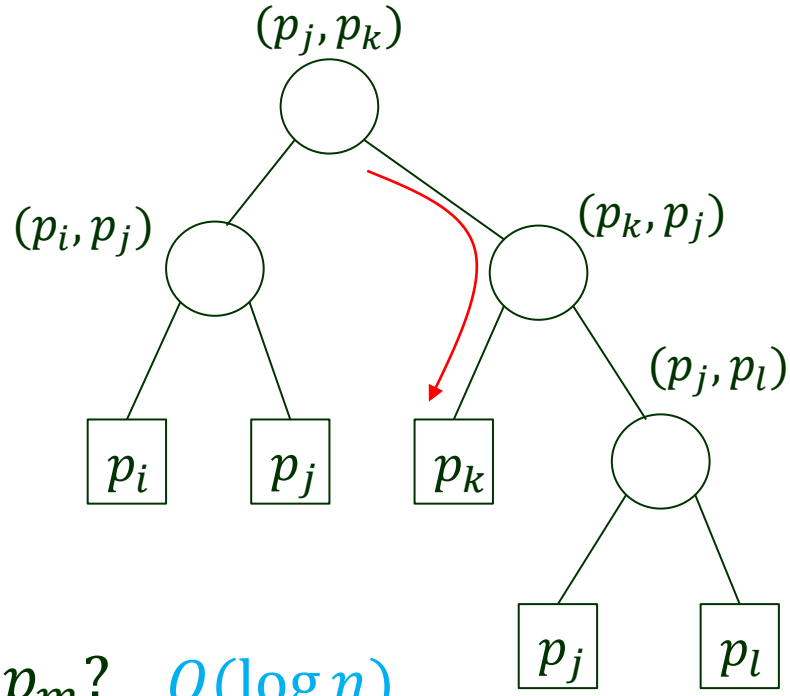


How to find the arc above a new site p_m?

- Compare its x-coordinate with that of the breakpoint (p_i, p_j) stored at an internal node.
- The x-coordinate of (p_i, p_j) can be computed in time $O(1)$.

Arc Above a New Site





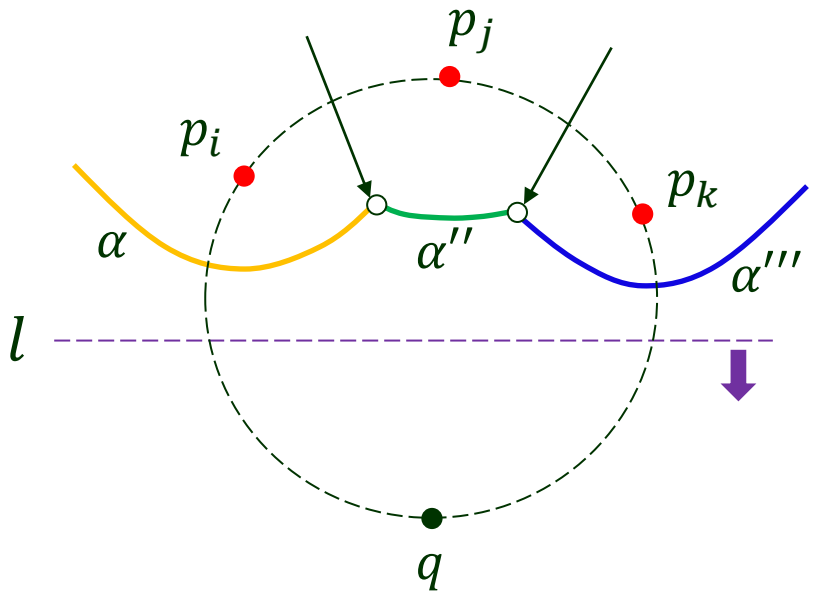
How to find the arc above a new site p_m? $O(\log n)$

- Compare its x-coordinate with that of the breakpoint (p_i, p_j) stored at an internal node.
- The x-coordinate of (p_i, p_j) can be computed in time $O(1)$.

Event Queue

As a priority queue Q in y-coordinate

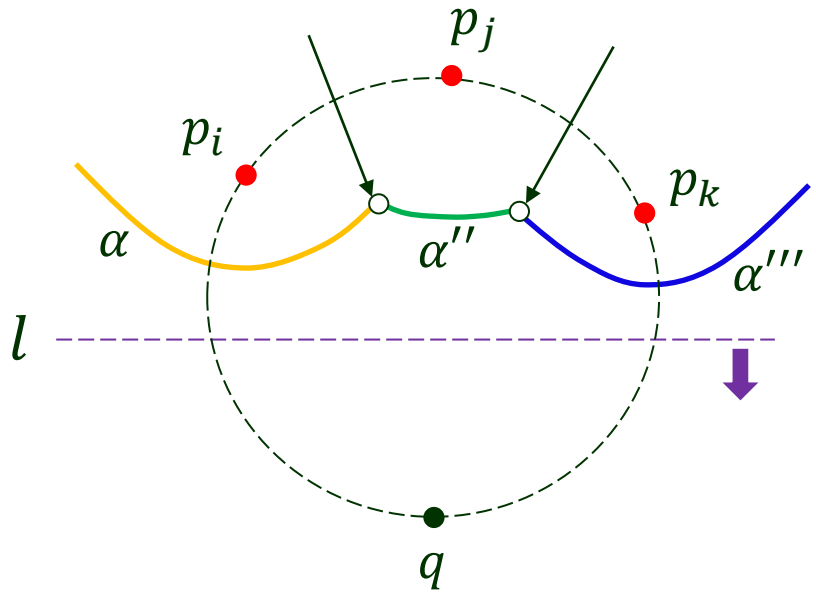
- For a site event, store the site.



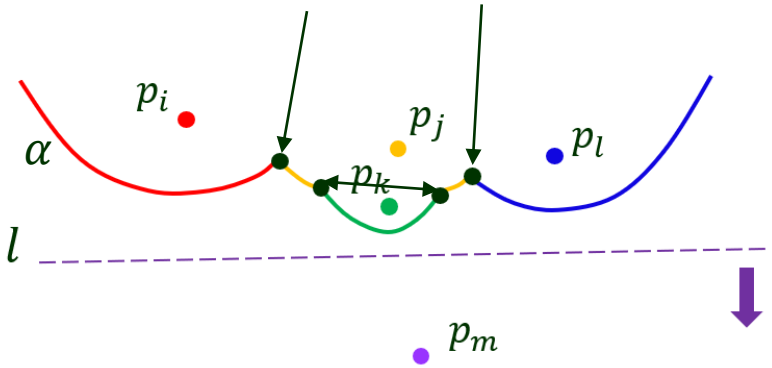
Event Queue

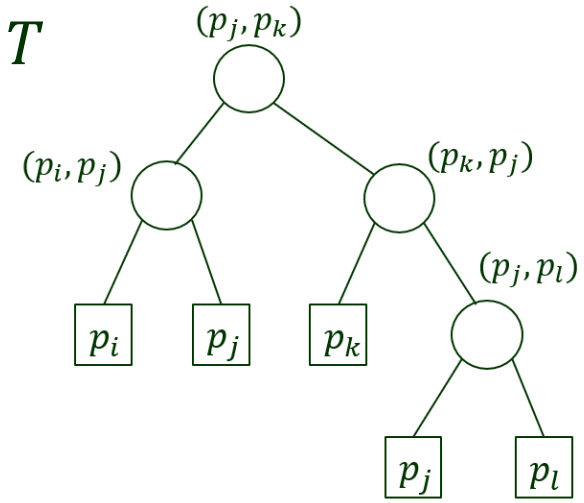
As a priority queue Q in y-coordinate

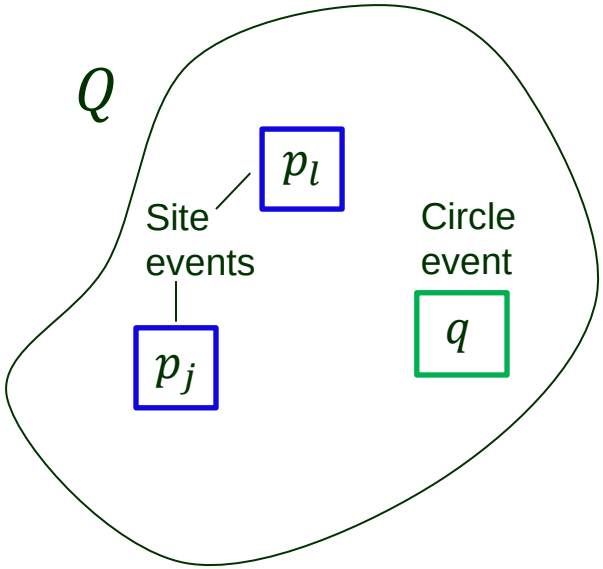
- For a site event, store the site.
- For a circle event, store
  - the lowest point (q) of the circle
  - a pointer to the leaf representing the arc to disappear



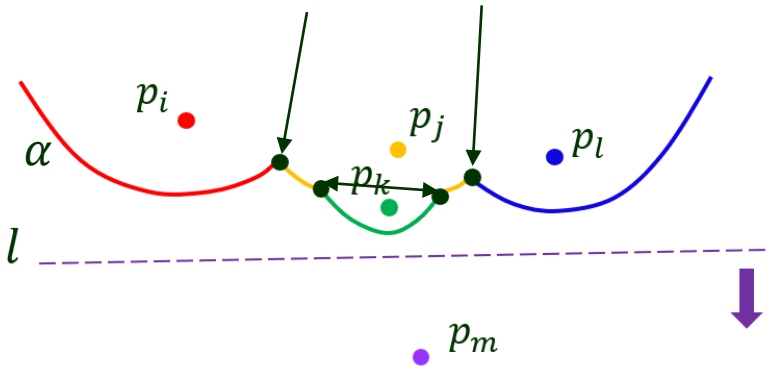
Cross Referencing





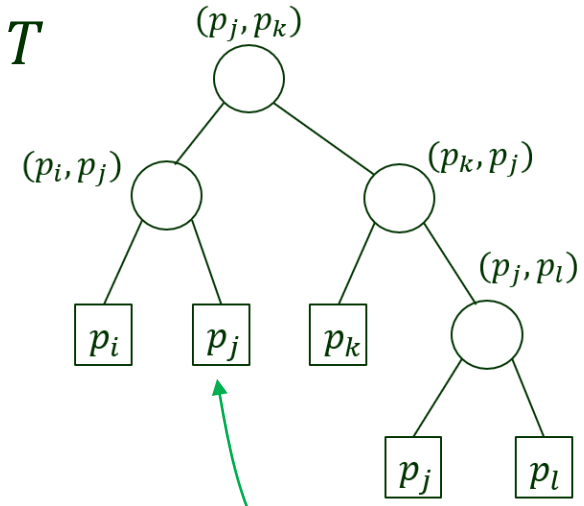


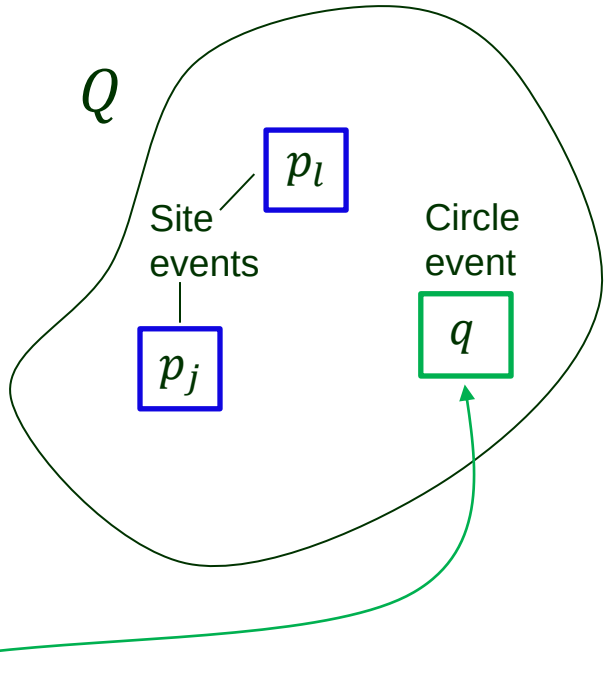
Cross Referencing



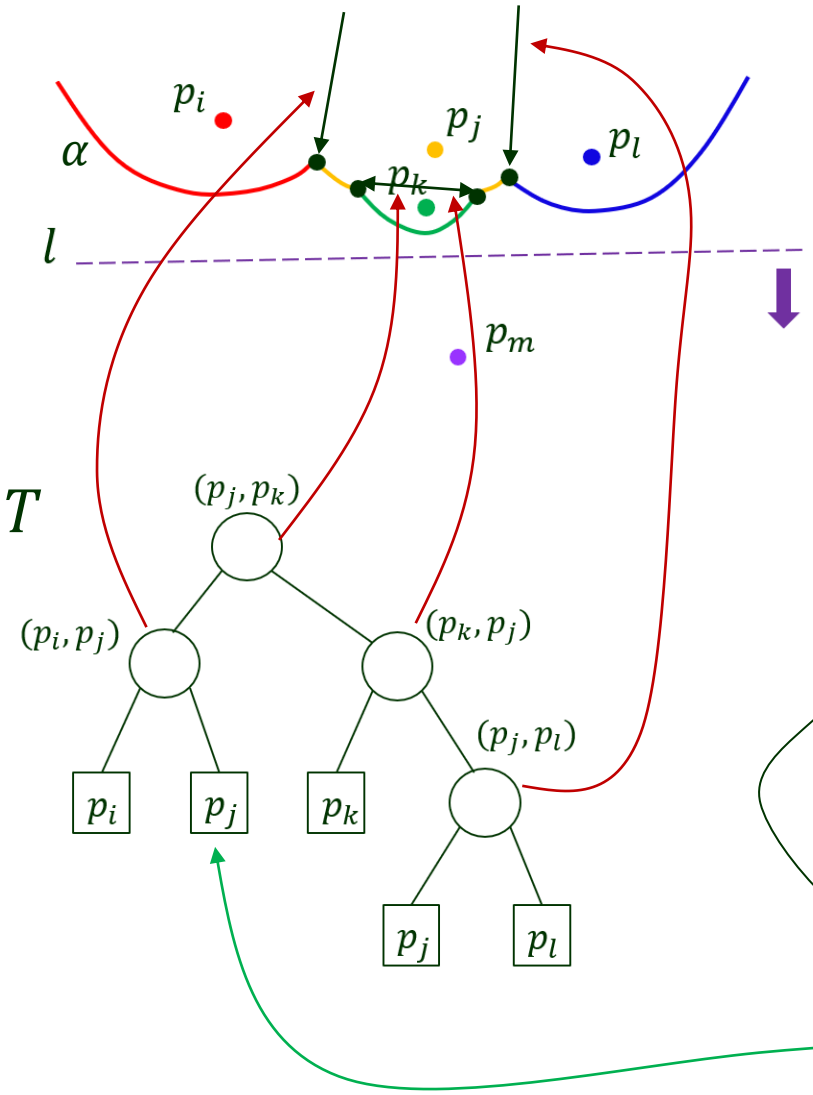
In T

- Every leaf (arc) points to a node (circle event) in Q, in which the arc will disappear.



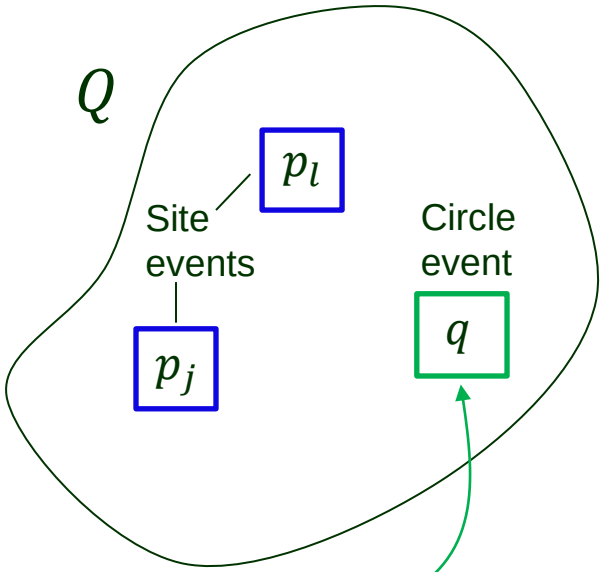


Cross Referencing

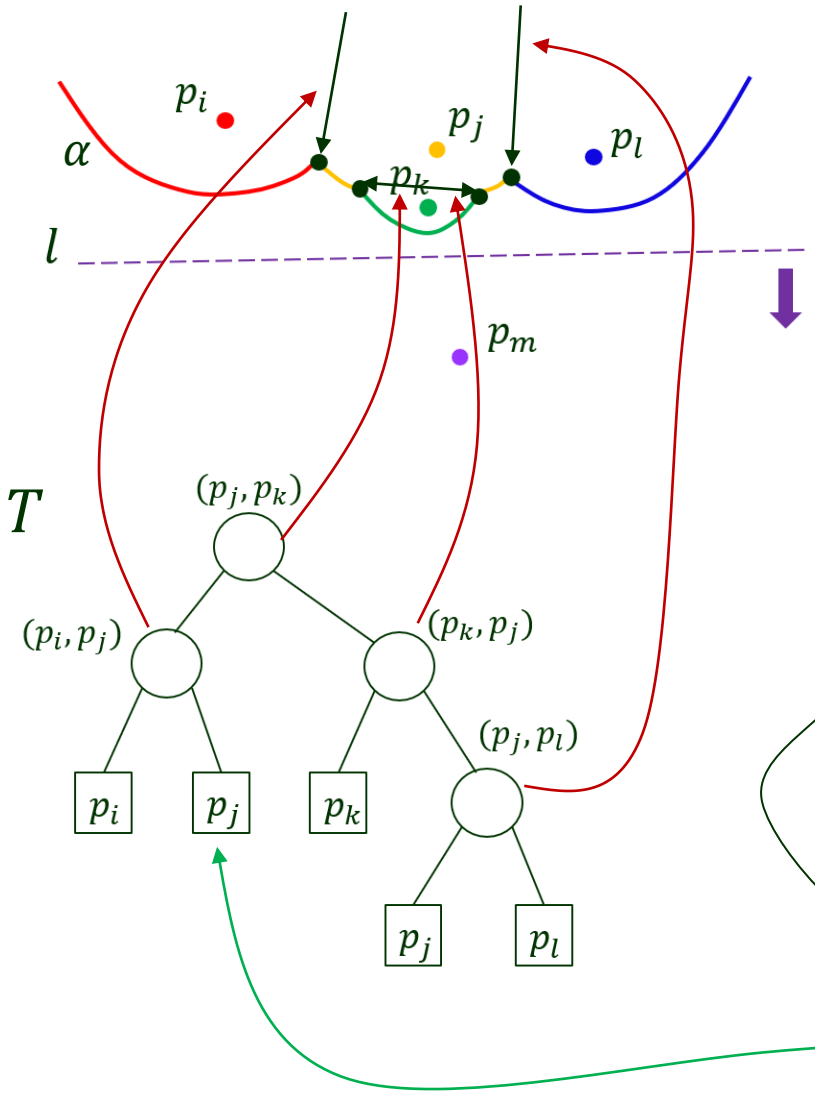


In T

- Every leaf (arc) points to a node (circle event) in Q, in which the arc will disappear.
- Every internal node points to a half-edge in Vor(P) being traced out by the break point.

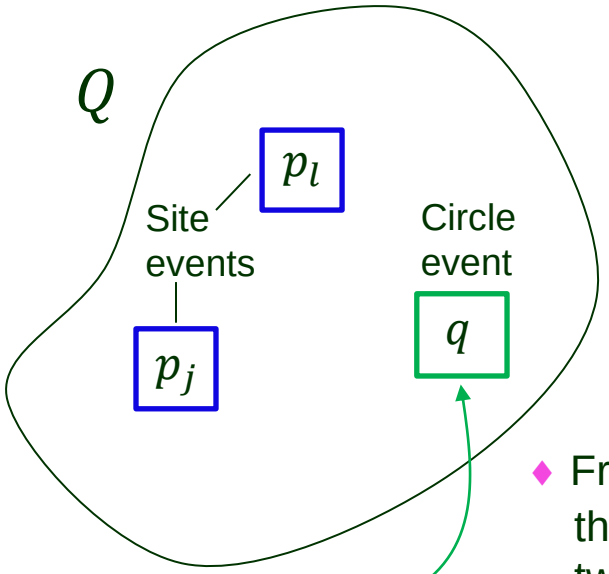


Cross Referencing



In T

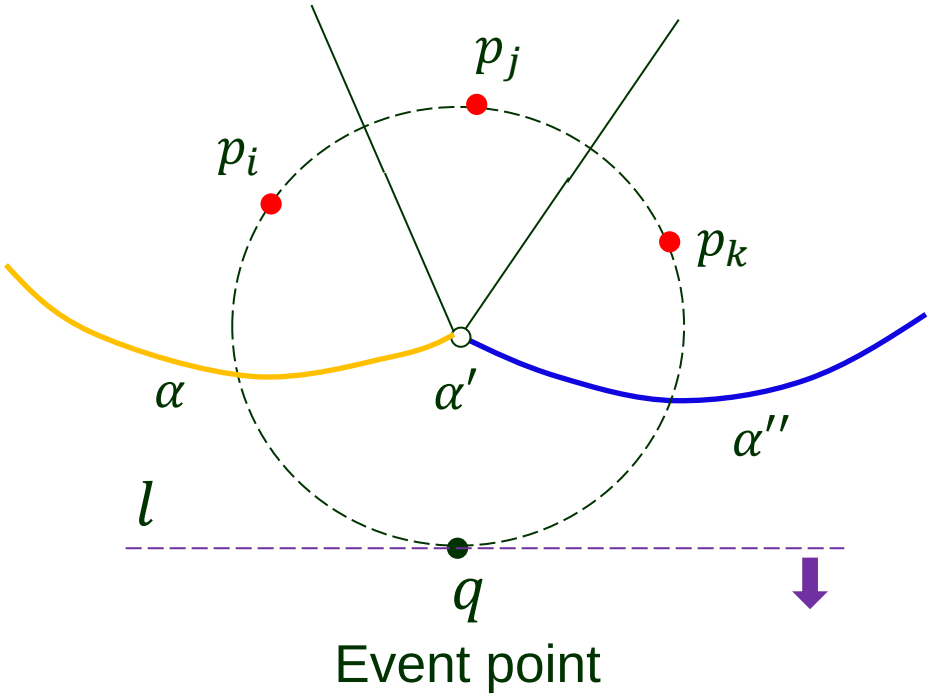
- Every leaf (arc) points to a node (circle event) in Q, in which the arc will disappear.
- Every internal node points to a half-edge in Vor(P) being traced out by the break point.



In Q

- Every circle event points to the leaf in T representing the arc to disappear.
- From the site p_j defining the arc we can find its two adjacent sites p_i and p_k, retrieving the circle event $\langle p_i, p_j, p_k \rangle$.

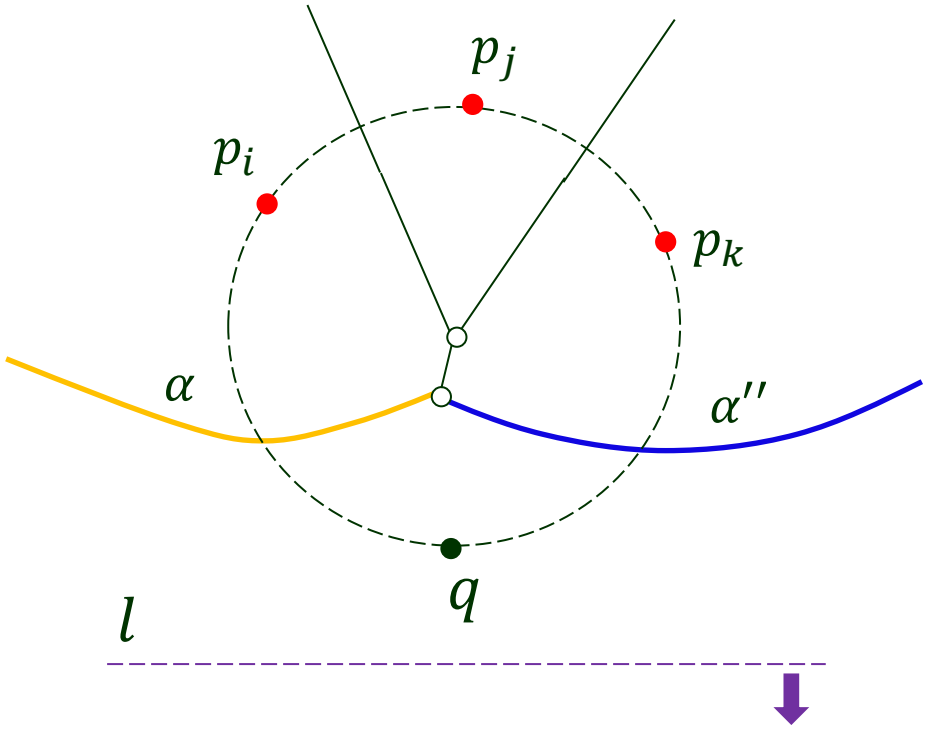
III. Detection of a Circle Event



α' is to disappear when q is on l.

Handled Event

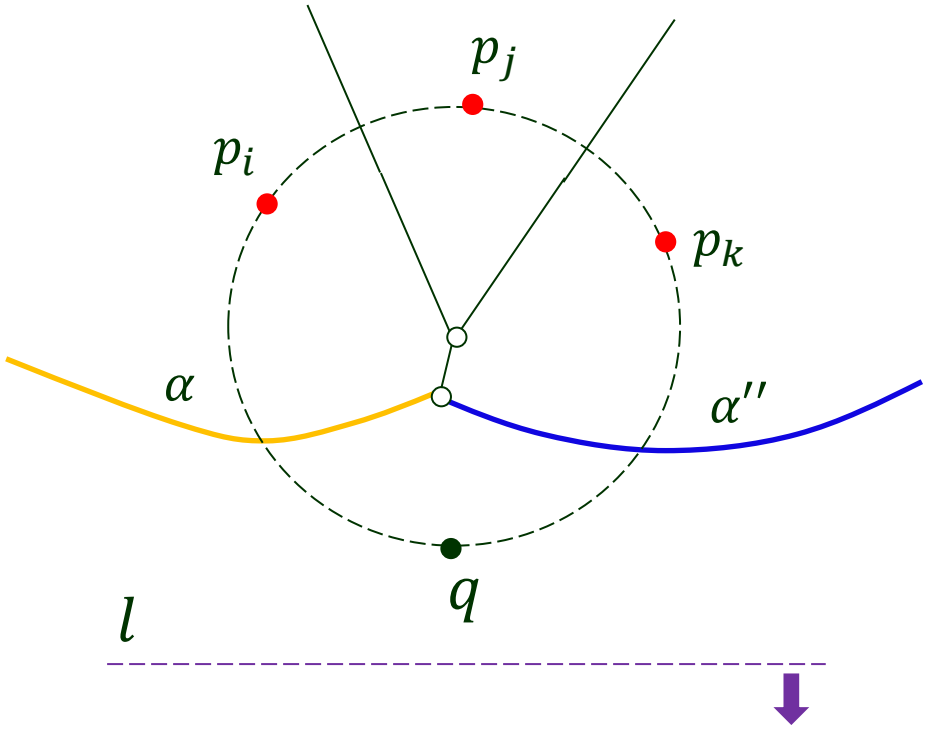
Every three consecutive arcs on the beach line introduce a circle event.



♣ Circle completely above l.

Handled Event

Every three consecutive arcs on the beach line introduce a circle event.



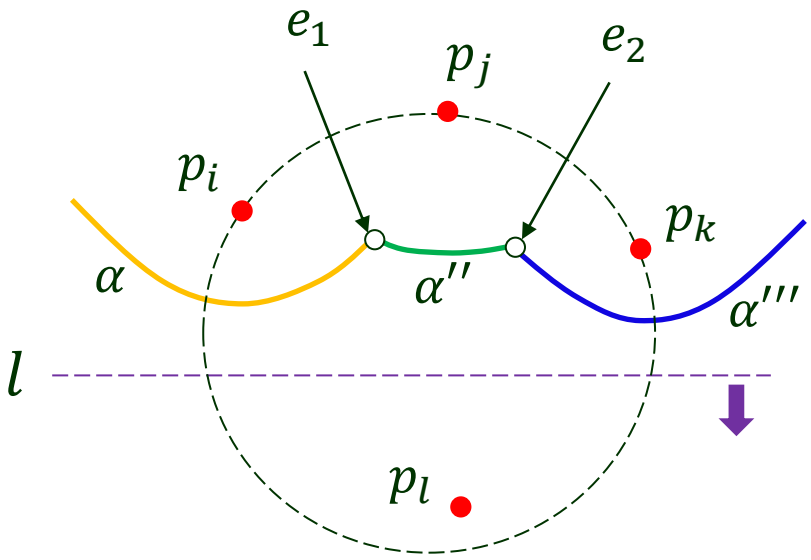
♣ Circle completely above l.

⇩

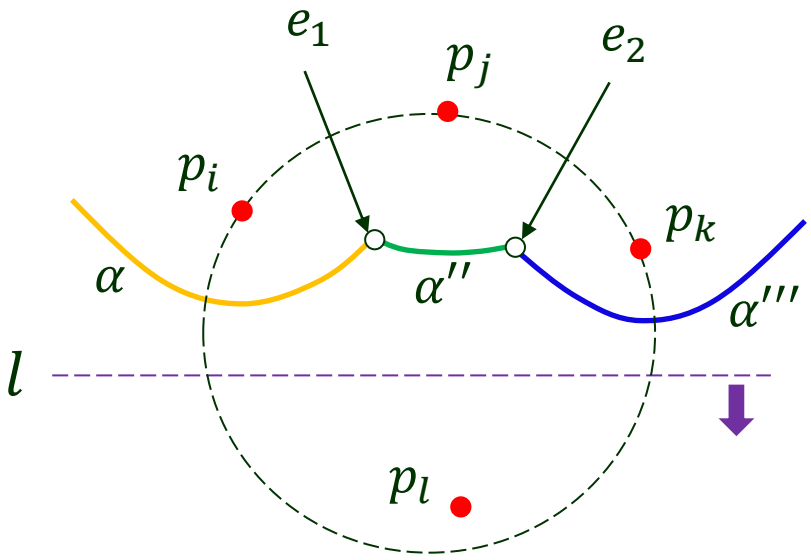
Event (q) has been dealt with.

False Alarm

♣ Circle contains some other site.



False Alarm

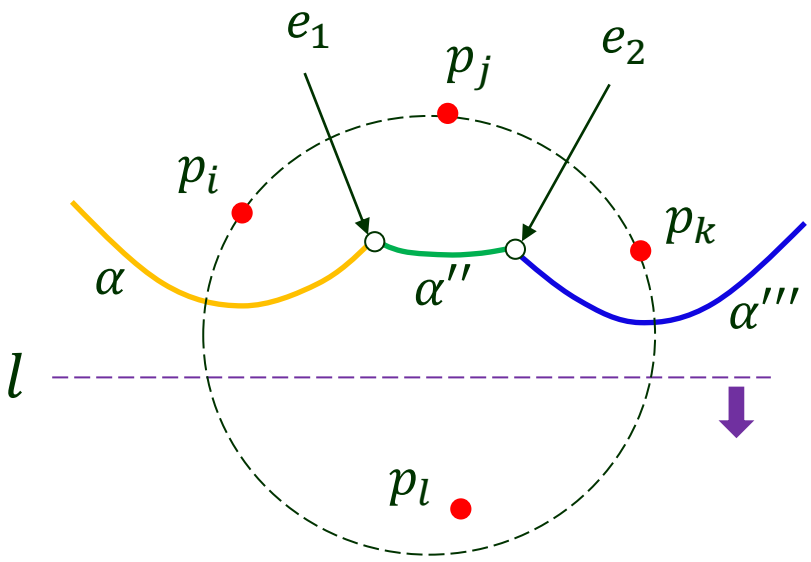


♣ Circle contains some other site.

⇩

Event should not be handled.

False Alarm



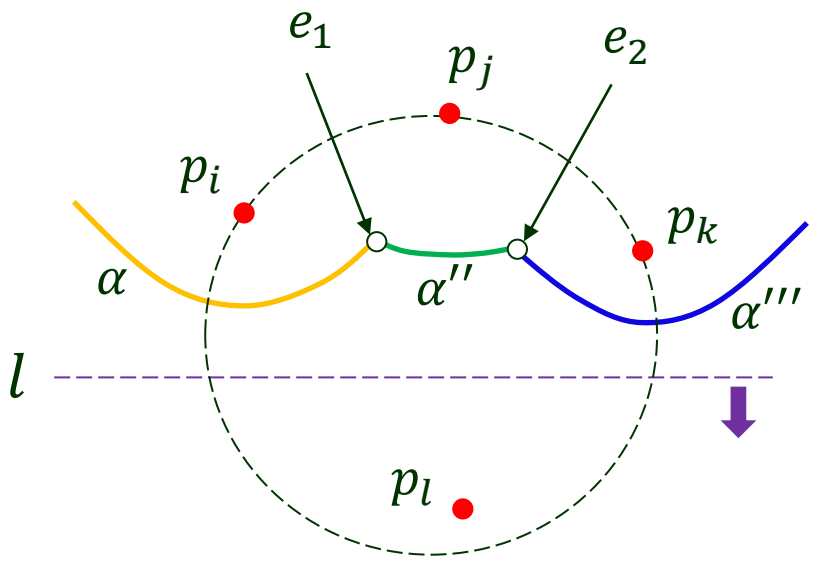
♣ Circle contains some other site.

⇩

Event should not be handled.

The two Voronoi edges e_1 and e_2 being traced out will not meet at a vertex.

False Alarm



♣ Circle contains some other site.

⇩

Event should not be handled.

The two Voronoi edges e_1 and e_2 being traced out will not meet at a vertex.

Lemma Every Voronoi vertex is detected at a circle event.

Structural Changes of the Beach Line

The beach line *changes its topological structure* at every event.

Structural Changes of the Beach Line

The beach line *changes its topological structure* at every event.

↑

Triples of adjacent arcs are introduced or destroyed.

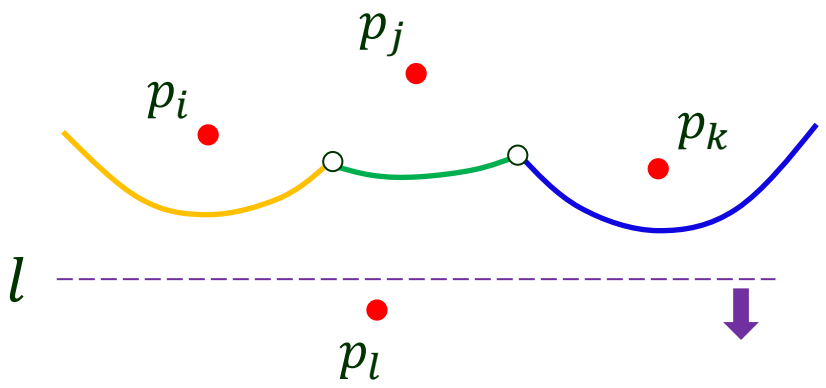
Structural Changes of the Beach Line

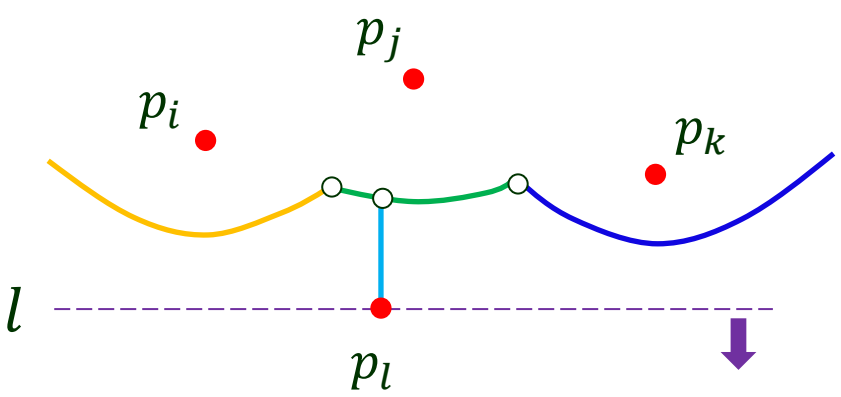
The beach line *changes its topological structure* at every event.

↑

Triples of adjacent arcs are introduced or destroyed.

Site event





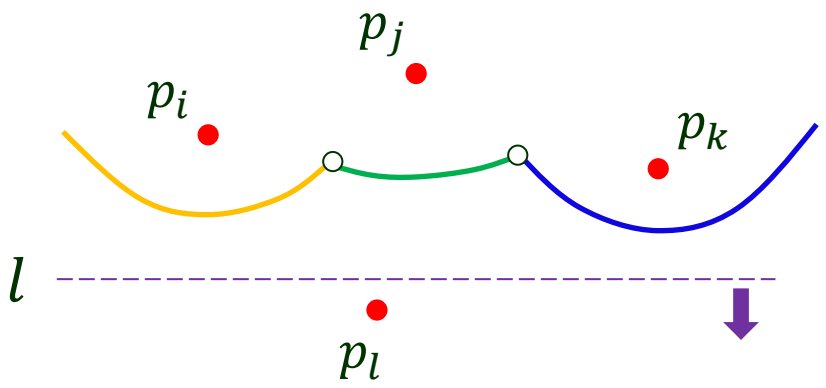
Structural Changes of the Beach Line

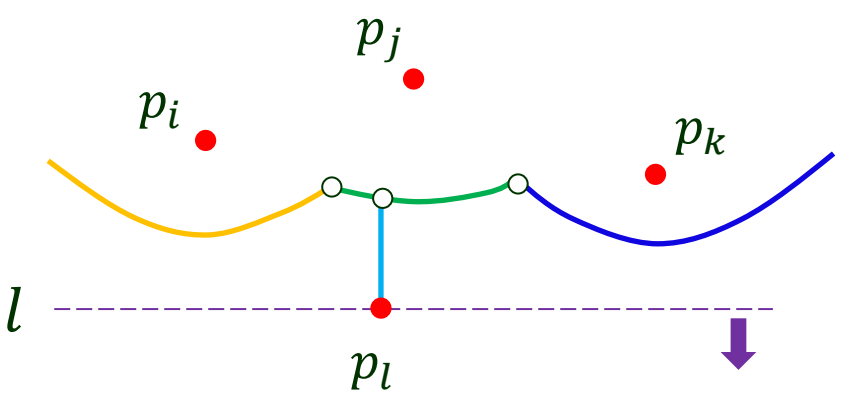
The beach line *changes its topological structure* at every event.

↑

Triples of adjacent arcs are introduced or destroyed.

Site event





Triple $\langle p_i, p_j, p_k \rangle$

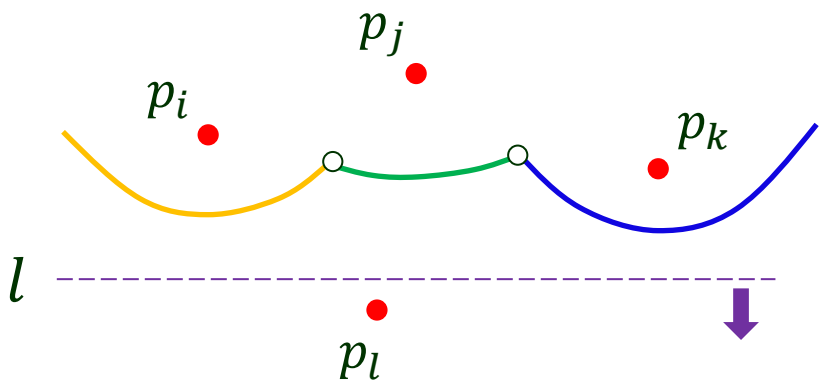
Structural Changes of the Beach Line

The beach line *changes its topological structure* at every event.

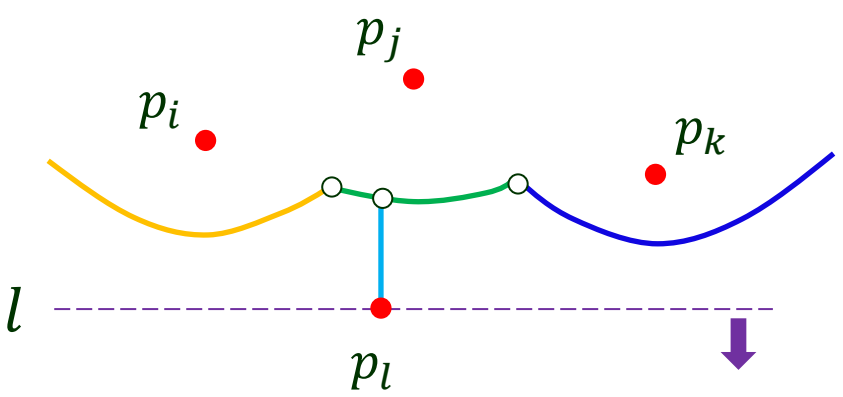
↑

Triples of adjacent arcs are introduced or destroyed.

Site event



Triple $\langle p_i, p_j, p_k \rangle$



Delete $\langle p_i, p_j, p_k \rangle$
Add $\langle p_i, p_j, p_l \rangle$
$\langle p_l, p_j, p_k \rangle$

Circle Event Insertion

Circle event will happen only if the corresponding triple stays until the sweep line *reaches the lowest point* of the circle.

Circle Event Insertion

Circle event will happen only if the corresponding triple stays until the sweep line *reaches the lowest point* of the circle.

- Insert a new triple of consecutive arcs as a circle event if all three conditions below hold:

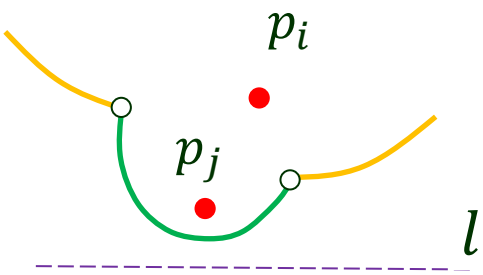
  a) the arcs are defined by 3 sites instead of 2;

Circle Event Insertion

Circle event will happen only if the corresponding triple stays until the sweep line *reaches the lowest point* of the circle.

- Insert a new triple of consecutive arcs as a circle event if all three conditions below hold:

  a) the arcs are defined by 3 sites instead of 2;



$\langle p_i, p_j, p_i \rangle$
not inserted

Circle Event Insertion

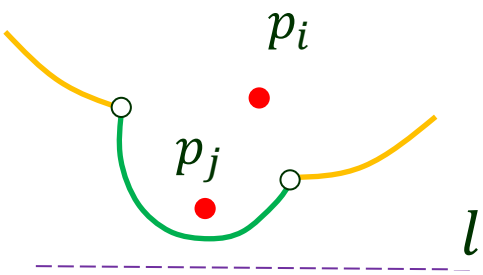
Circle event will happen only if the corresponding triple stays until the sweep line *reaches the lowest point* of the circle.

- Insert a new triple of consecutive arcs as a circle event if all three conditions below hold:

  a) the arcs are defined by 3 sites instead of 2;

  b) the triple is not in the event queue Q;

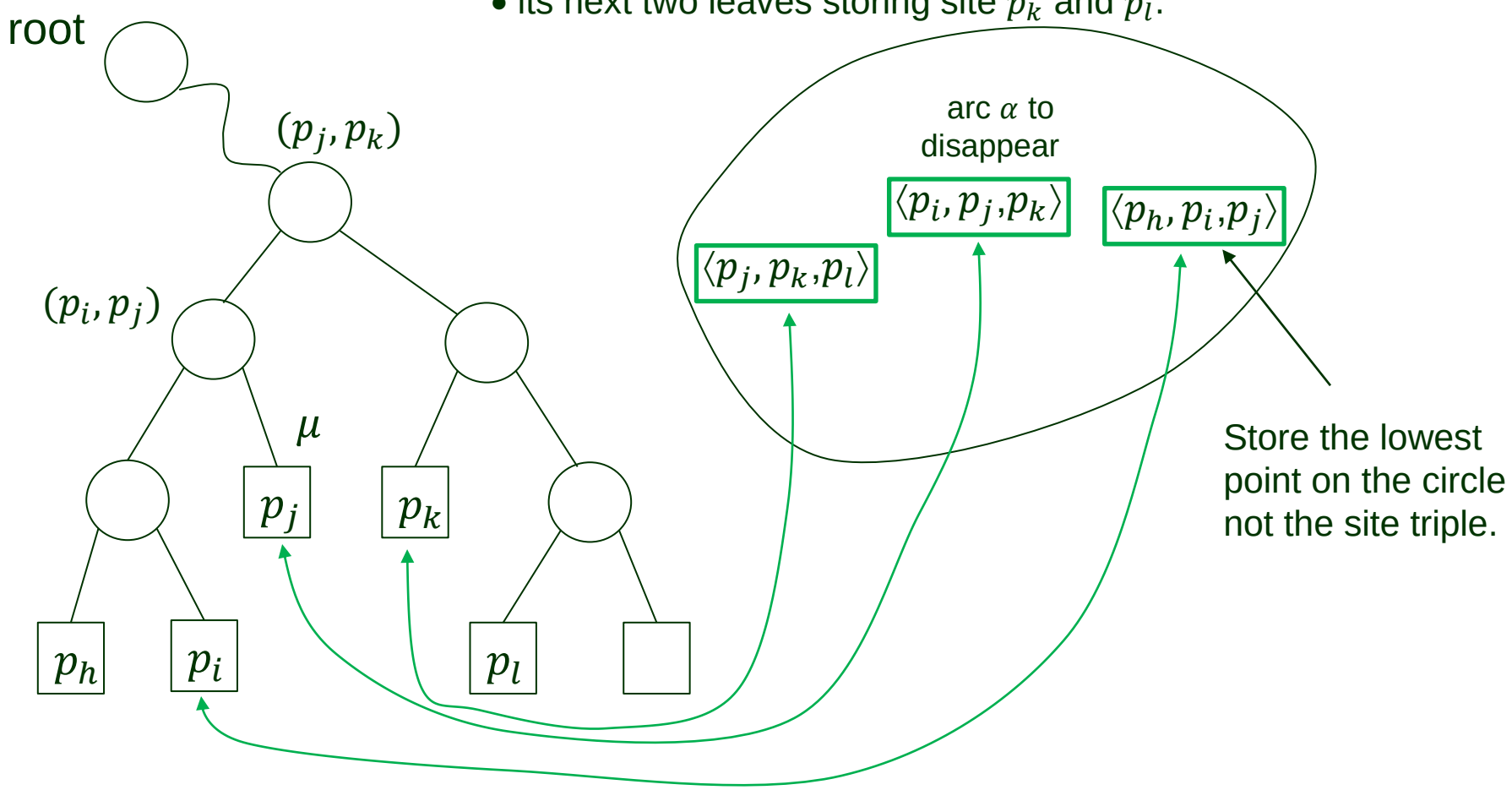
  c) the circle through the three defining sites intersects the sweep line.



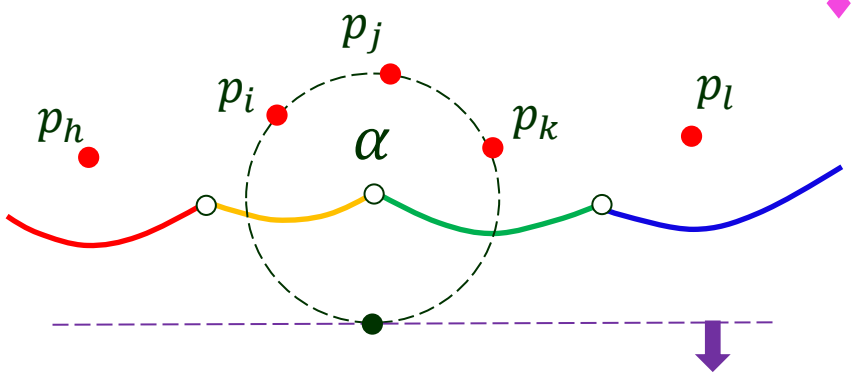
$\langle p_i, p_j, p_i \rangle$
not inserted

Circle Event Deletion

- Takes place when arc α disappears or splits. To find all affected circle events, search in T for
  - the leaf μ storing the site p_j defining α.
  - its previous two leaves storing sites p_i and p_h.
  - its next two leaves storing site p_k and p_l.



Circle Event Deletion

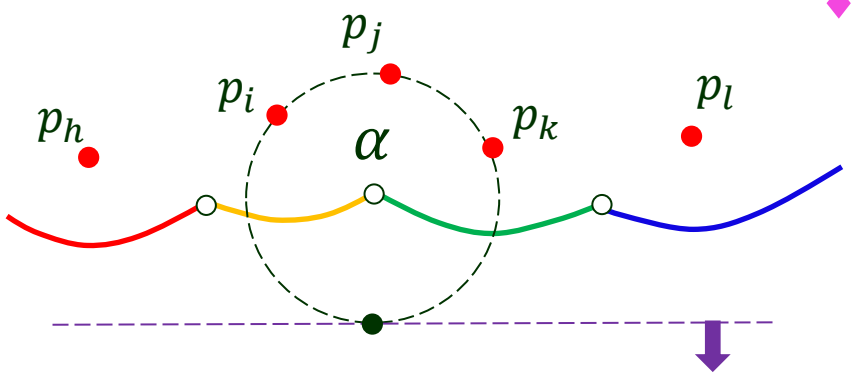


- In the case arc α disappears

Delete

Add

Circle Event Deletion

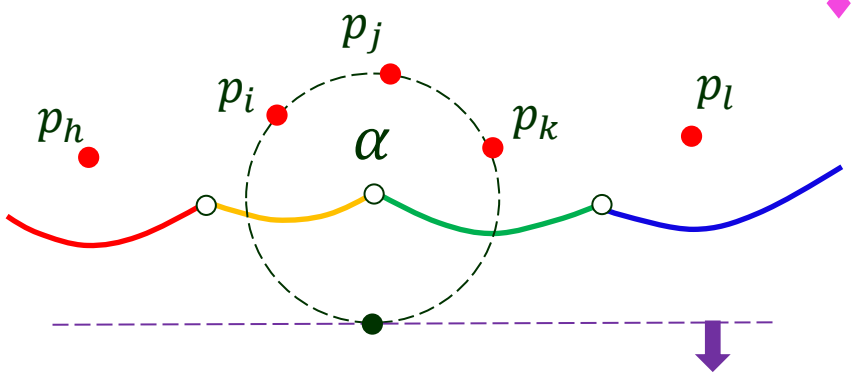


- In the case arc α disappears

Delete $\langle p_h, p_i, p_j \rangle$
$\langle p_i, p_j, p_k \rangle$
$\langle p_j, p_k, p_l \rangle$

Add

Circle Event Deletion

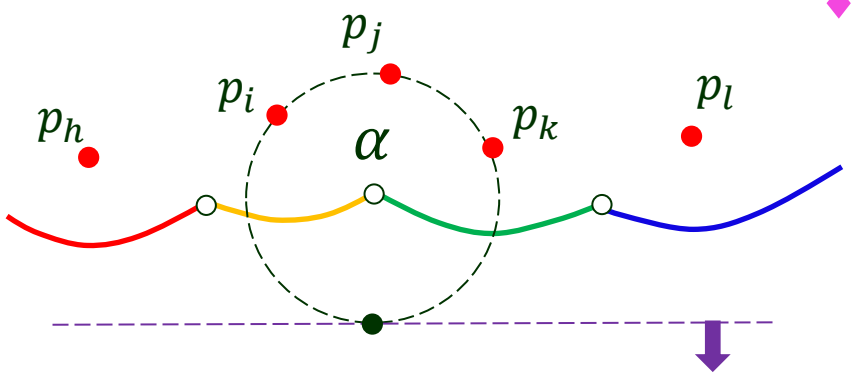


- In the case arc α disappears

Delete $\langle p_h, p_i, p_j \rangle$, $\langle p_i, p_j, p_k \rangle$, $\langle p_j, p_k, p_l \rangle$ } Circle events involving α

Add

Circle Event Deletion



- In the case arc α disappears

Delete

May have been removed already

$\langle p_h, p_i, p_j \rangle$

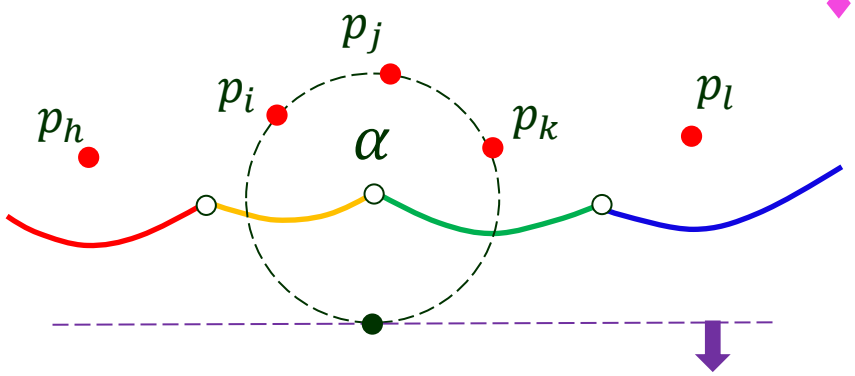
$\langle p_i, p_j, p_k \rangle$

$\langle p_j, p_k, p_l \rangle$

Circle events involving α

Add

Circle Event Deletion



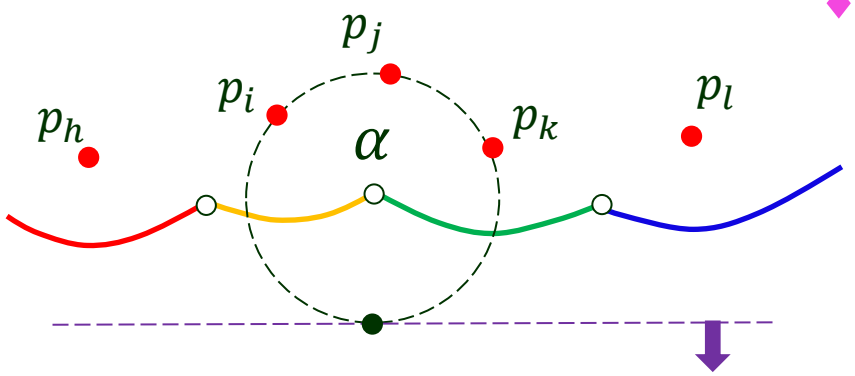
- In the case arc α disappears

Delete $\langle p_h, p_i, p_j \rangle$, $\langle p_i, p_j, p_k \rangle$, $\langle p_j, p_k, p_l \rangle$ — Circle events involving α

May have been removed already

Add $\langle p_h, p_i, p_k \rangle$ if $h \neq k$ and circle intersecting l

Circle Event Deletion



- In the case arc α disappears

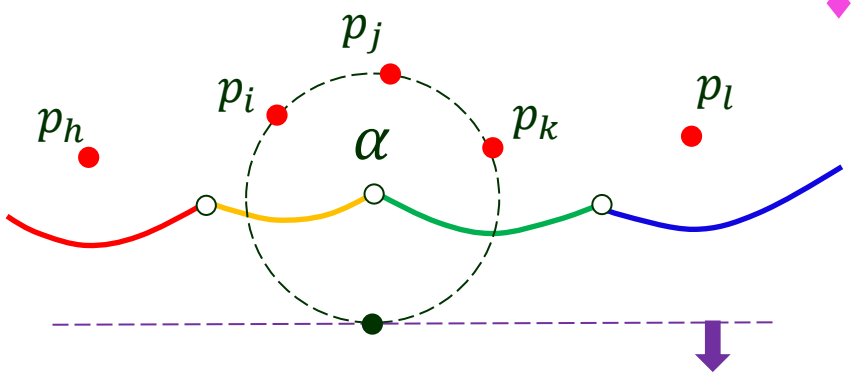
Delete $\langle p_h, p_i, p_j \rangle$, $\langle p_i, p_j, p_k \rangle$, $\langle p_j, p_k, p_l \rangle$ — Circle events involving α

May have been removed already

Add $\langle p_h, p_i, p_k \rangle$ if $h \neq k$ and circle intersecting l

$\langle p_i, p_k, p_l \rangle$ if $i \neq l$ and circle intersecting l

Circle Event Deletion



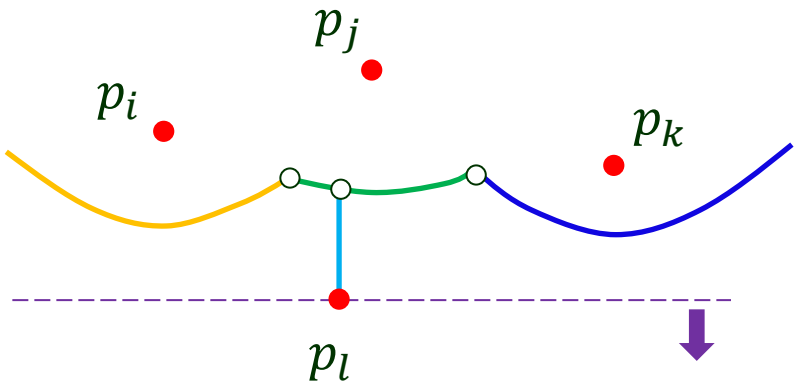
- In the case arc α disappears

Delete $\langle p_h, p_i, p_j \rangle$, $\langle p_i, p_j, p_k \rangle$, $\langle p_j, p_k, p_l \rangle$ — Circle events involving α

May have been removed already

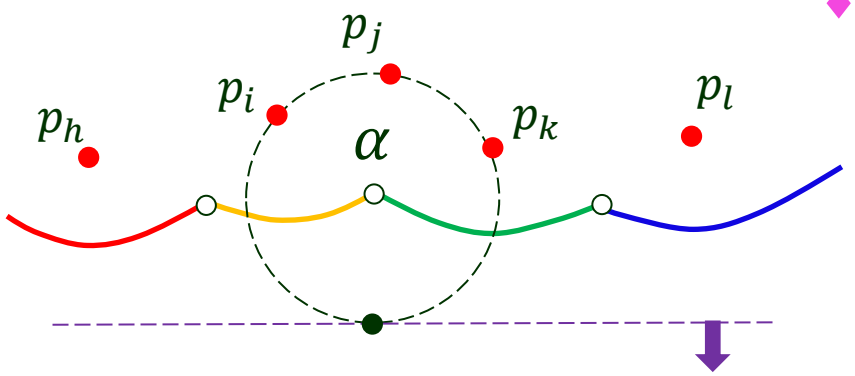
Add $\langle p_h, p_i, p_k \rangle$ if $h \neq k$ and circle intersecting l

$\langle p_i, p_k, p_l \rangle$ if $i \neq l$ and circle intersecting l



- In the case arc α splits due to a site event p_l

Circle Event Deletion



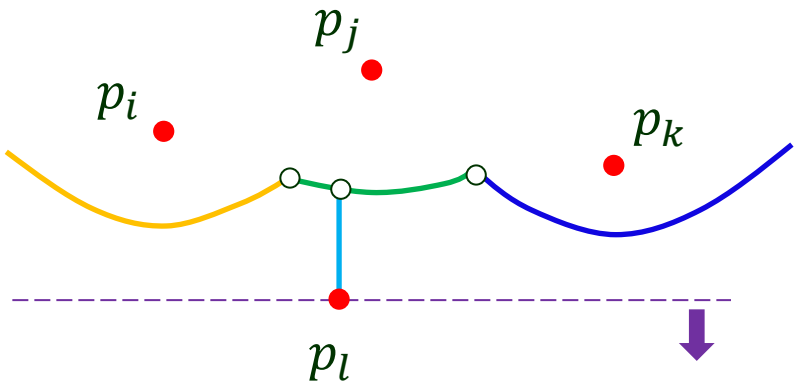
- In the case arc α disappears

Delete $\langle p_h, p_i, p_j \rangle$, $\langle p_i, p_j, p_k \rangle$, $\langle p_j, p_k, p_l \rangle$ (Circle events involving α)

May have been removed already

Add $\langle p_h, p_i, p_k \rangle$ if $h \neq k$ and circle intersecting l

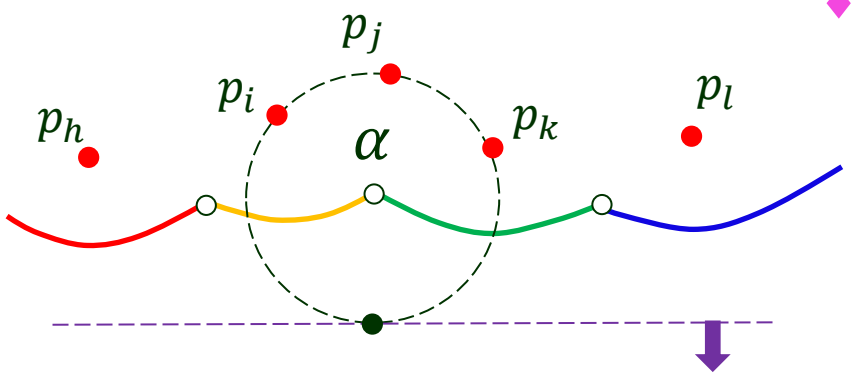
$\langle p_i, p_k, p_l \rangle$ if $i \neq l$ and circle intersecting l



- In the case arc α splits due to a site event p_l

Delete $\langle p_i, p_j, p_k \rangle$

Circle Event Deletion



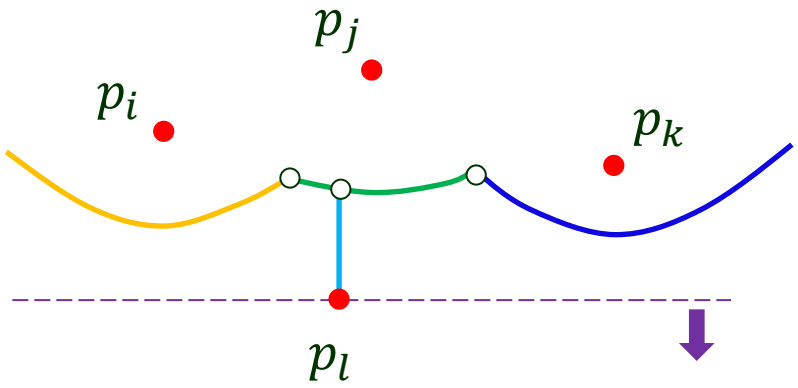
- In the case arc α disappears

Delete $\langle p_h, p_i, p_j \rangle$, $\langle p_i, p_j, p_k \rangle$, $\langle p_j, p_k, p_l \rangle$ (Circle events involving α)

May have been removed already

Add $\langle p_h, p_i, p_k \rangle$ if $h \neq k$ and circle intersecting l

$\langle p_i, p_k, p_l \rangle$ if $i \neq l$ and circle intersecting l



- In the case arc α splits due to a site event p_l

Delete $\langle p_i, p_j, p_k \rangle$

Add $\langle p_i, p_j, p_l \rangle$

$\langle p_l, p_j, p_k \rangle$

IV. Construction Algorithm

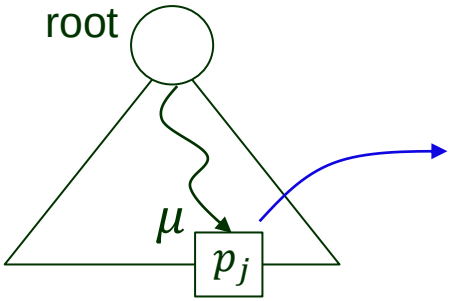
VoronoiDiagram(P)

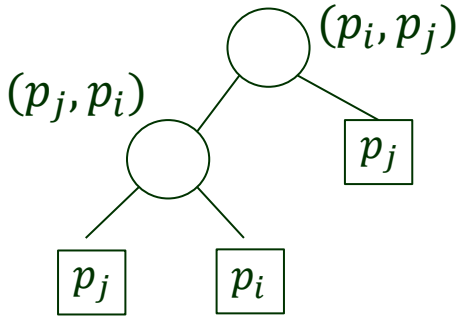
1. Initialize event queue Q with $p_1, p_2 \ldots, p_n$
2. while $Q \neq \emptyset$
3. do extract event e with the largest y-coordinate from Q.
4. if e is a site event at p_i
5. then HandleSiteEvent(p_i)
6. else HandleCircleEvent(p_s) // $p_s, s > n$ is the lowest
   // point on the corresponding circle.
7. Compute a bounding box with all Voronoi vertices in its interior.
8. Update DCEL for half-infinite edges.
9. Traverse DCEL to add cell records and related pointers.

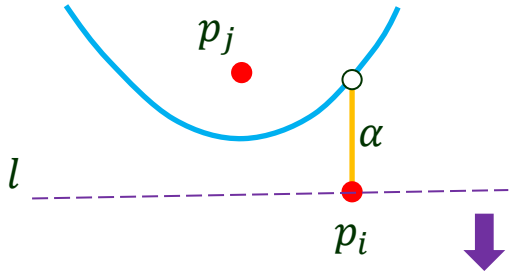
Site Event Handling

HandleSiteEvent(p_i)

1. $\alpha \leftarrow$ arc vertically above p_i in the plane // search in T to find α defined by p_j
2. $Q \leftarrow Q - \{$ circle events involving α $\}$
3. Replace the leaf p_j with a subtree below.



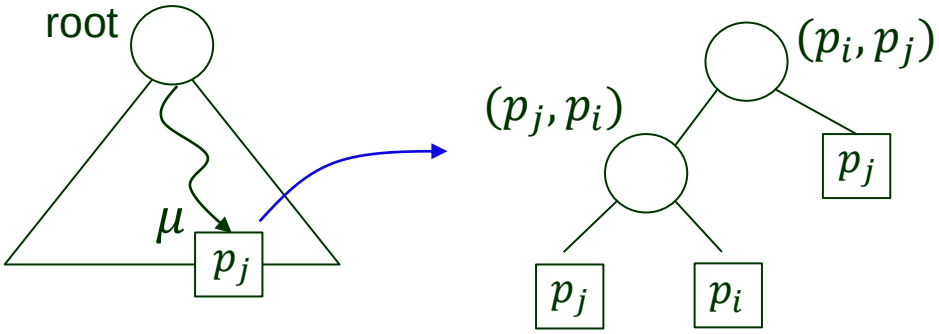


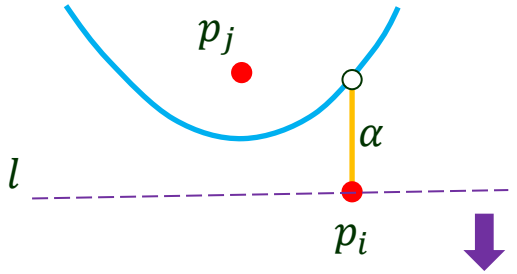


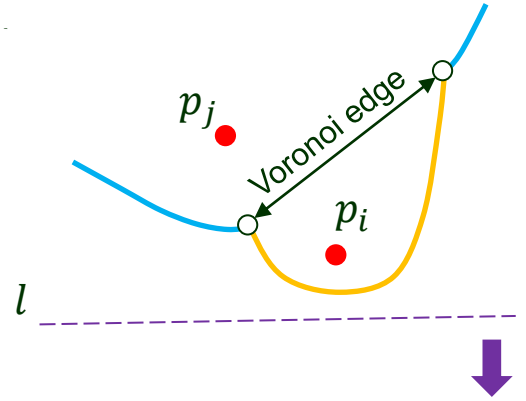
Site Event Handling

HandleSiteEvent(p_i)

1. $\alpha \leftarrow$ arc vertically above p_i in the plane // search in T to find α defined by p_j
2. $Q \leftarrow Q - \{$ circle events involving α $\}$
3. Replace the leaf p_j with a subtree below.



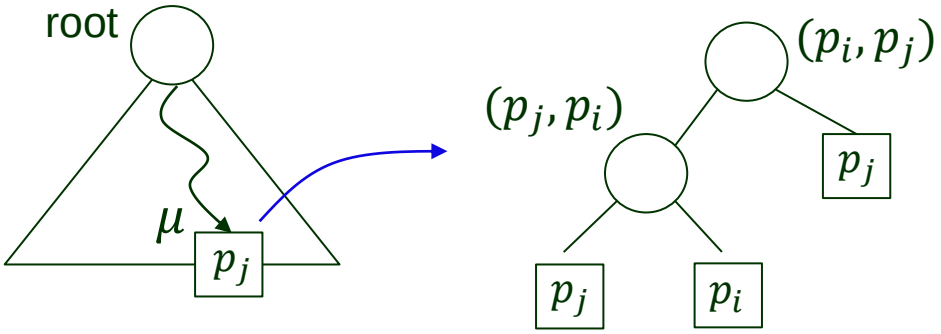


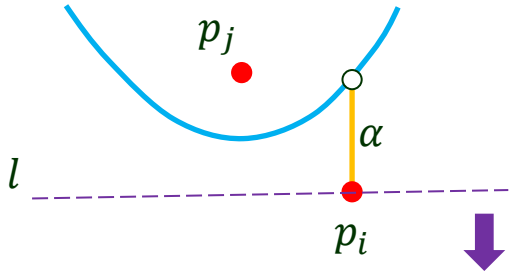


Site Event Handling

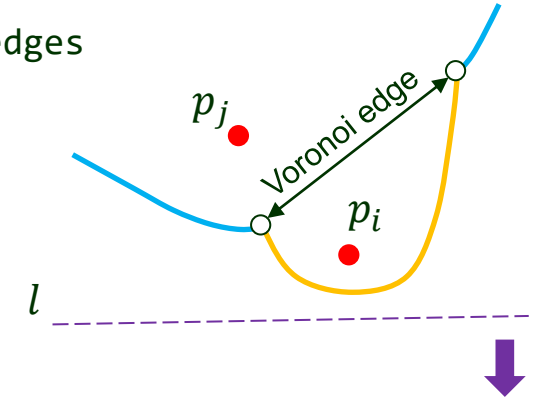
HandleSiteEvent(p_i)

1. $\alpha \leftarrow$ arc vertically above p_i in the plane // search in T to find α defined by p_j
2. $Q \leftarrow Q - \{$ circle events involving α $\}$
3. Replace the leaf p_j with a subtree below.





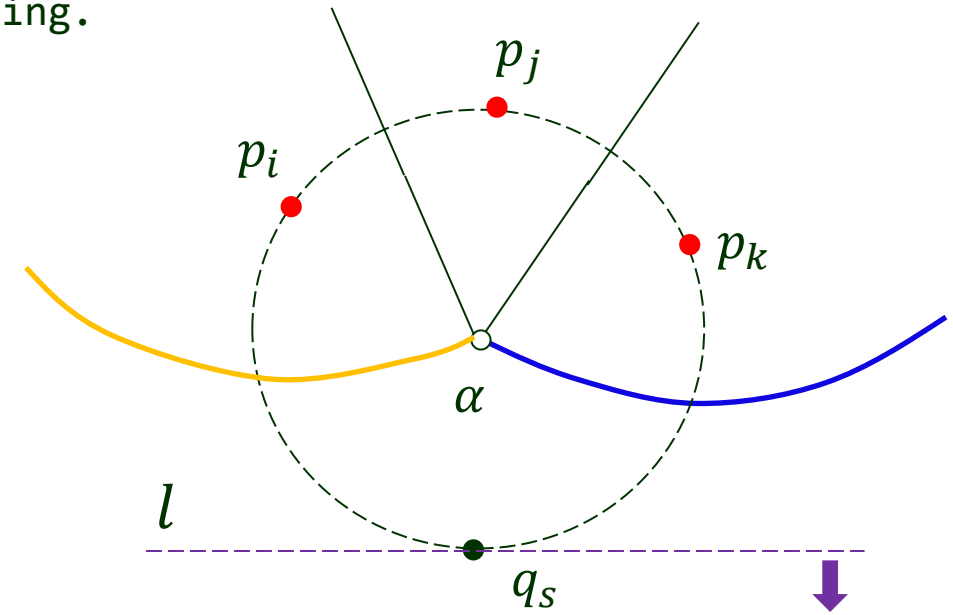
4. Add new records in Voronoi diagram (DCEL) for two half-edges separating $V(p_i)$ and $V(p_j)$ to be traced out.
5. Check arc triples $\langle \_, p_j, p_i \rangle$ and $\langle p_i, p_j, \_ \rangle$, inserting each as a circle event if the three conditions below hold:
   a) The triple is not present in Q.
   b) The defined circle intersects l.
   c) The two breakpoints in the triple are converging (moving toward each other).



Circle Event Handling

HandleCircleEvent(q_s)

1. $\alpha \leftarrow$ arc vertically above q_s to disappear // search in T
2. $Q \leftarrow Q -$ { circle events involving α }
3. Delete leaf μ representing α in T.
4. Update the tuples (representing the breakpoints) at the internal nodes.
5. Add the center of the circle as a vertex record to DCEL.
6. Create two half-edge records (between cells V_i and V_k).
7. Check new triples that arise: $\langle \_, p_i, p_k \rangle$ and $\langle p_i, p_k, \_ \rangle$, inserting each as a circle event into Q if the two breakpoints in the triple are converging.



Time Complexity

Theorem The algorithms runs in $O(n \log n)$ time using $O(n)$ storage.

Proof (sketch)

Time Complexity

Theorem The algorithms runs in $O(n \log n)$ time using $O(n)$ storage.

Proof (sketch)

- Insertions and deletions in T and Q take $O(\log n)$ each.

Time Complexity

Theorem The algorithms runs in $O(n \log n)$ time using $O(n)$ storage.

Proof (sketch)

- Insertions and deletions in T and Q take $O(\log n)$ each.
- Operations on DCEL take $O(1)$ each.

Time Complexity

Theorem The algorithms runs in $O(n \log n)$ time using $O(n)$ storage.

Proof (sketch)

- Insertions and deletions in T and Q take $O(\log n)$ each.
- Operations on DCEL take $O(1)$ each.
- Each event requires $O(1)$ operations thus $O(\log n)$ time to process.

Time Complexity

Theorem The algorithms runs in $O(n \log n)$ time using $O(n)$ storage.

Proof (sketch)

- Insertions and deletions in T and Q take $O(\log n)$ each.
- Operations on DCEL take $O(1)$ each.
- Each event requires $O(1)$ operations thus $O(\log n)$ time to process.
  - n site events

Time Complexity

Theorem The algorithms runs in $O(n \log n)$ time using $O(n)$ storage.

Proof (sketch)

- Insertions and deletions in T and Q take $O(\log n)$ each.
- Operations on DCEL take $O(1)$ each.
- Each event requires $O(1)$ operations thus $O(\log n)$ time to process.
  - n site events
  - #circle events proportional to #vertices $\leq 2n - 5$
    - false alarms simply deleted
    - vertices handled

Time Complexity

Theorem The algorithms runs in $O(n \log n)$ time using $O(n)$ storage.

Proof (sketch)

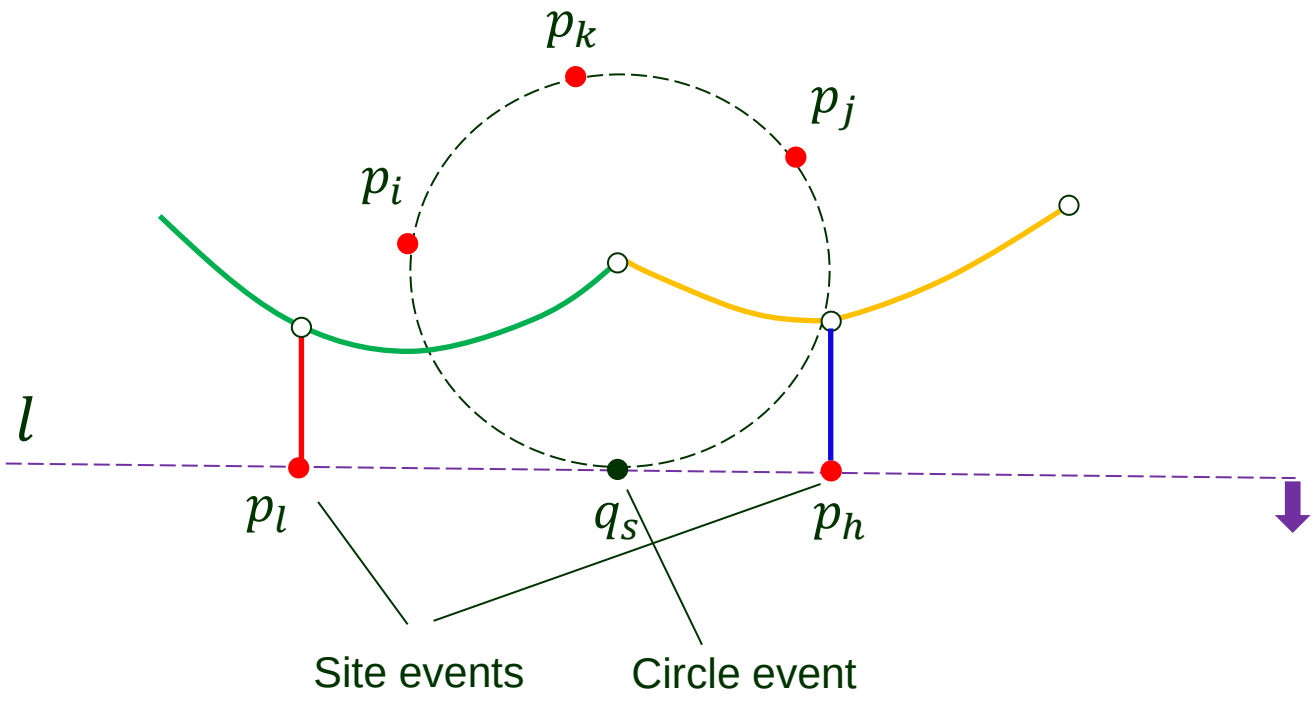
- Insertions and deletions in T and Q take $O(\log n)$ each.
- Operations on DCEL take $O(1)$ each.
- Each event requires $O(1)$ operations thus $O(\log n)$ time to process.
  - n site events
  - #circle events proportional to #vertices $\leq 2n - 5$
    - false alarms simply deleted
    - vertices handled

□

V. Degeneracy (1)

Two or more event sites with the same y-coordinate.

1a) different x-coordinates

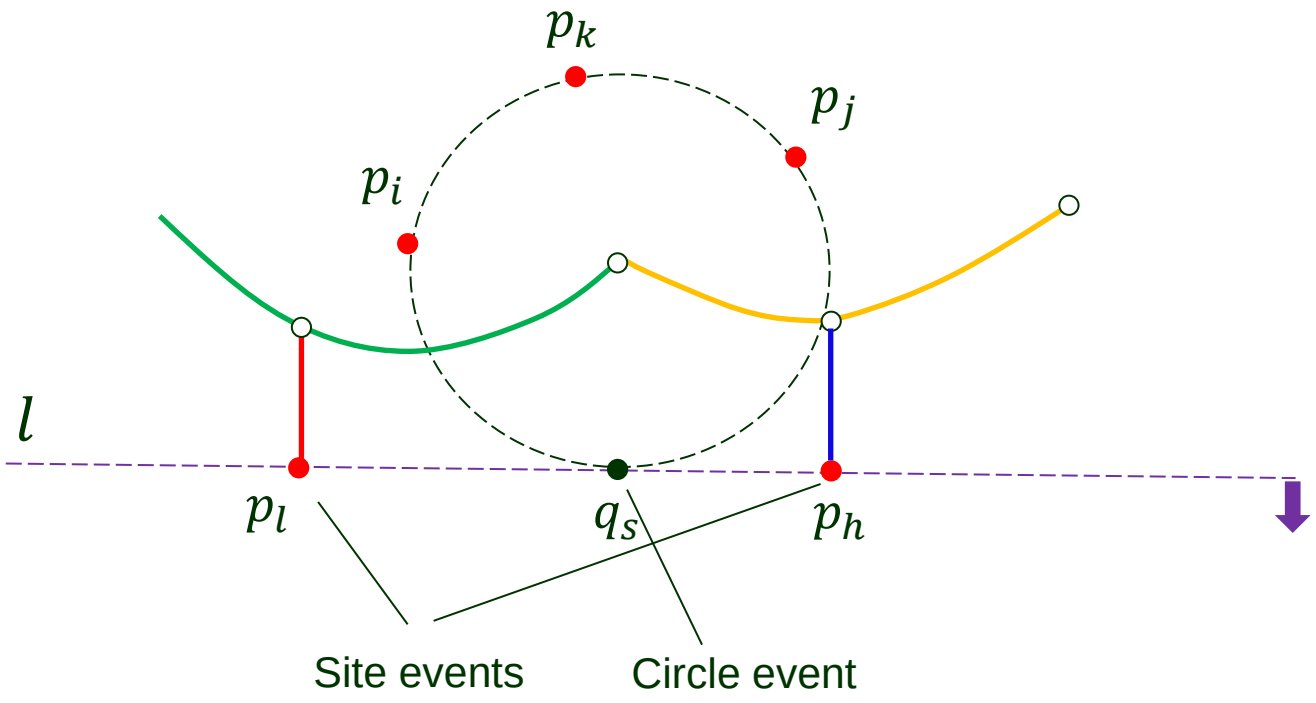


V. Degeneracy (1)

Two or more event sites with the same y-coordinate.

1a) different x-coordinates

- ≥ 2 site and/or circle events.



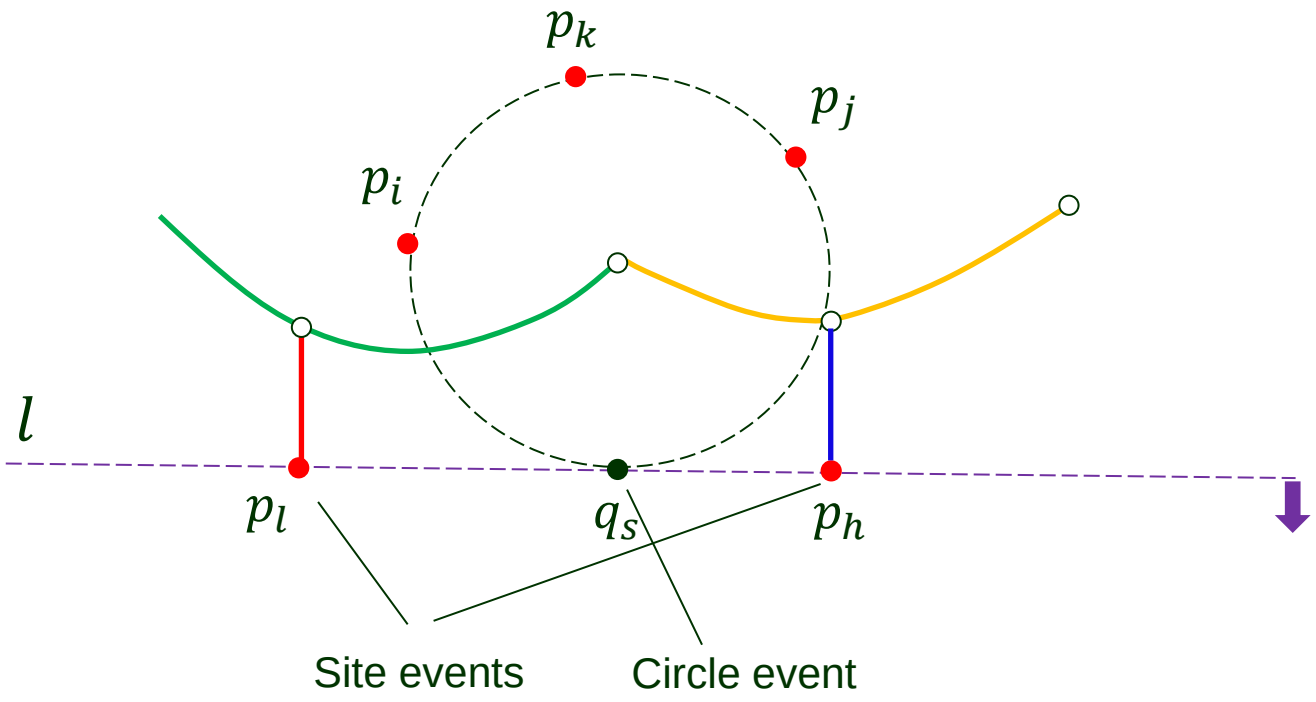
V. Degeneracy (1)

Two or more event sites with the same y-coordinate.

1a) different x-coordinates

- ≥ 2 site and/or circle events.

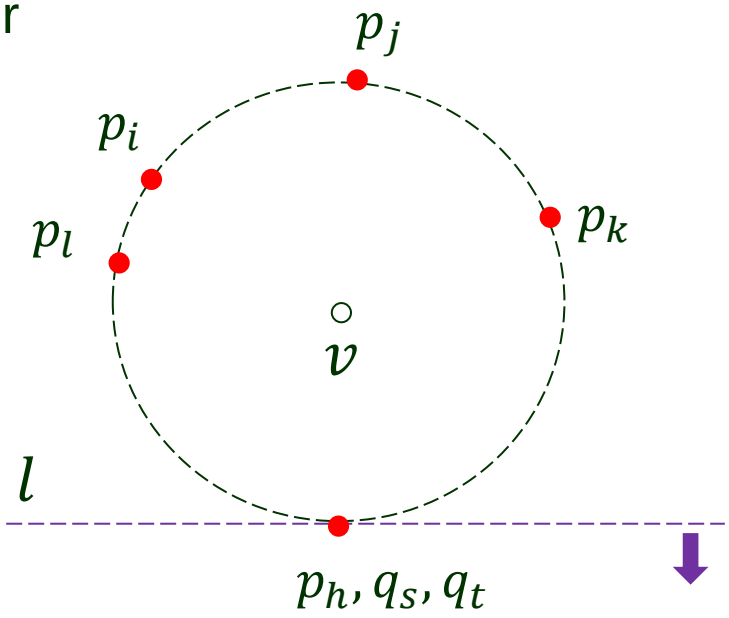
Handle them in any order.



Degeneracy (1) – cont'd

1b) same x-coordinate

- 0 site event and ≥ 2 circle events, or
- 1 site event and ≥ 1 circle event



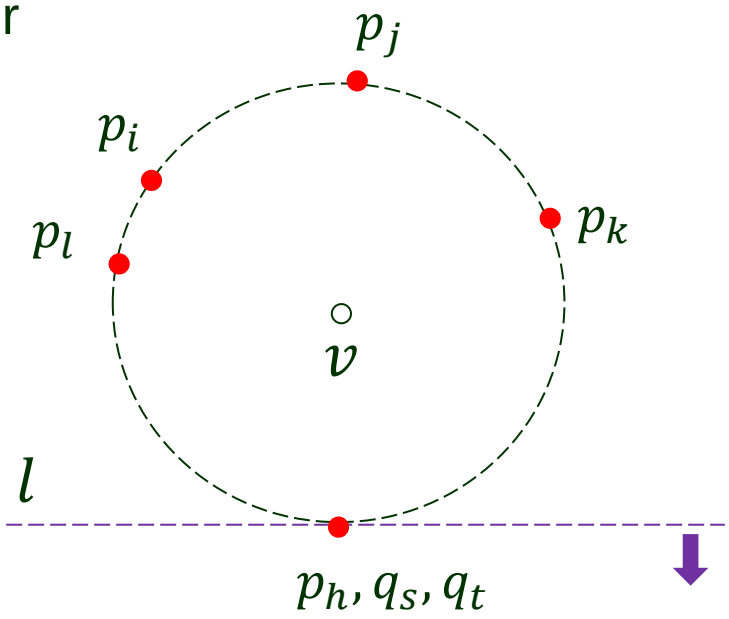
Degeneracy (1) – cont'd

1b) same x-coordinate

- 0 site event and ≥ 2 circle events, or
- 1 site event and ≥ 1 circle event

⇩

a Voronoi vertex of degree ≥ 4



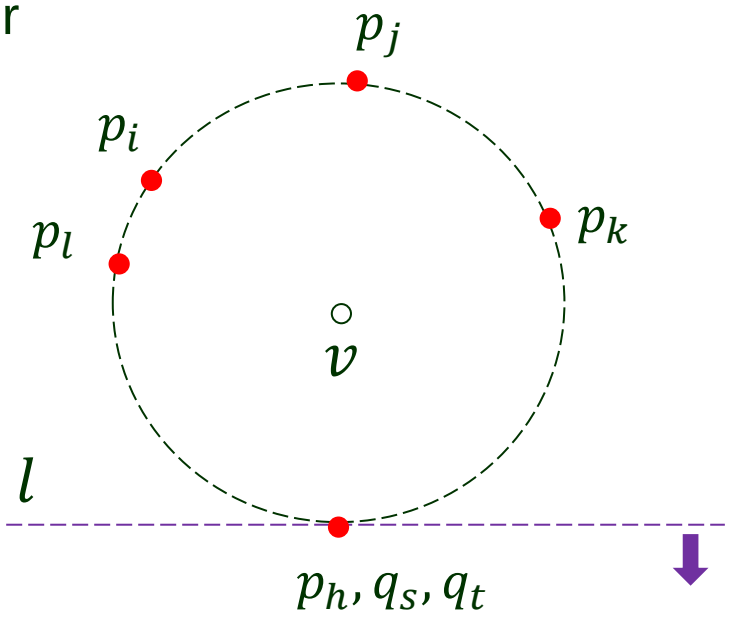
Degeneracy (1) – cont'd

1b) same x-coordinate

- 0 site event and ≥ 2 circle events, or
- 1 site event and ≥ 1 circle event

⇩

a Voronoi vertex of degree ≥ 4



Three events coincide:

p_h

$q_s = \langle p_l, p_i, p_j \rangle$

$q_t = \langle p_i, p_j, p_k \rangle$

Degeneracy (1) – cont'd

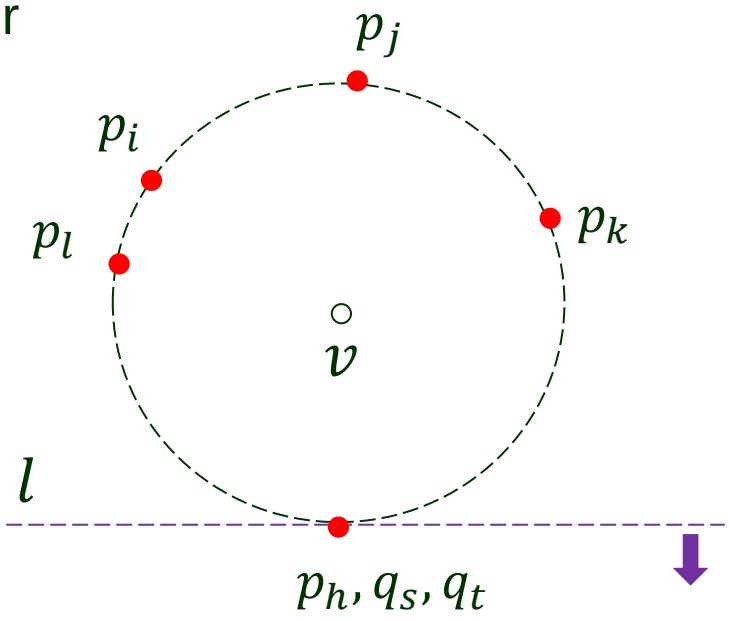
1b) same x-coordinate

- 0 site event and ≥ 2 circle events, or
- 1 site event and ≥ 1 circle event

⇩

a Voronoi vertex of degree ≥ 4

- Handle circle events first.



Three events coincide:

p_h

$q_s = \langle p_l, p_i, p_j \rangle$

$q_t = \langle p_i, p_j, p_k \rangle$

Degeneracy (1) – cont'd

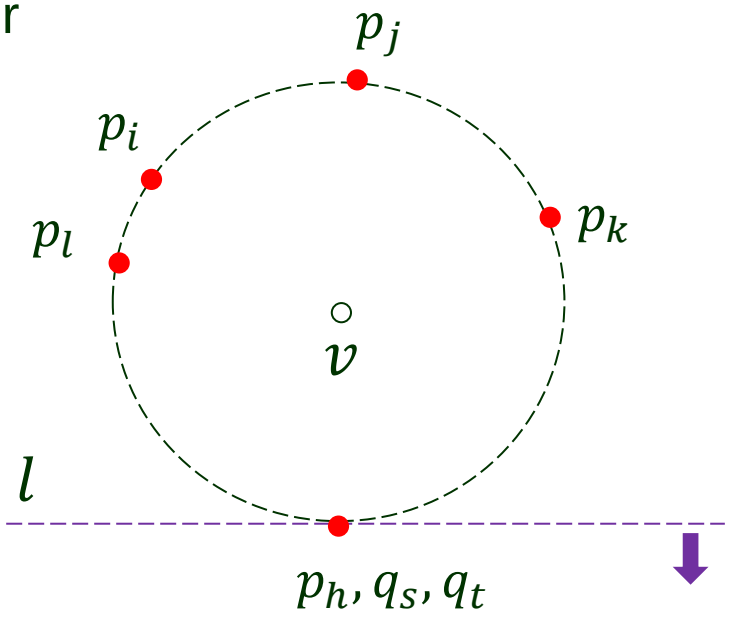
1b) same x-coordinate

- 0 site event and ≥ 2 circle events, or
- 1 site event and ≥ 1 circle event

⇩

a Voronoi vertex of degree ≥ 4

- Handle circle events first.
  - as ≥ 2 vertices coinciding



Three events coincide:

p_h

$q_s = \langle p_l, p_i, p_j \rangle$

$q_t = \langle p_i, p_j, p_k \rangle$

Degeneracy (1) – cont'd

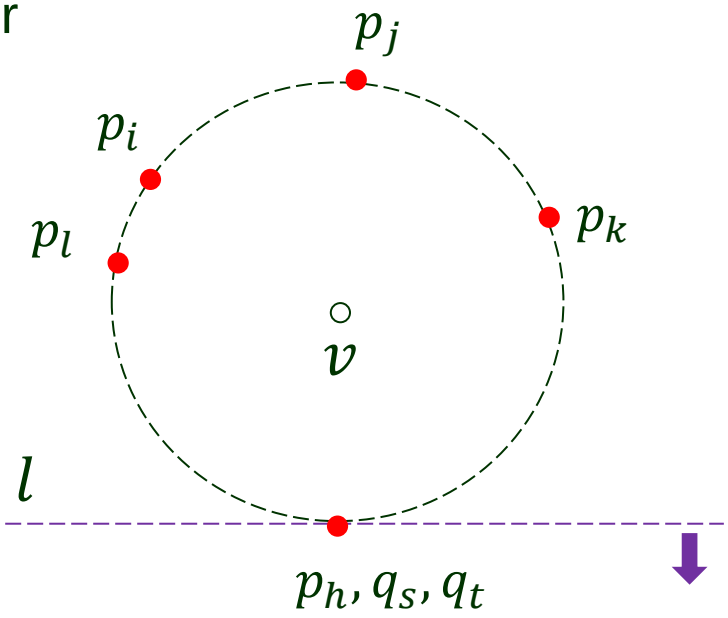
1b) same x-coordinate

- 0 site event and ≥ 2 circle events, or
- 1 site event and ≥ 1 circle event

⇩

a Voronoi vertex of degree ≥ 4

- Handle circle events first.
  - as ≥ 2 vertices coinciding
  - each with degree 3 and 0 length in between



Three events coincide:

p_h

$q_s = \langle p_l, p_i, p_j \rangle$

$q_t = \langle p_i, p_j, p_k \rangle$

Degeneracy (1) – cont'd

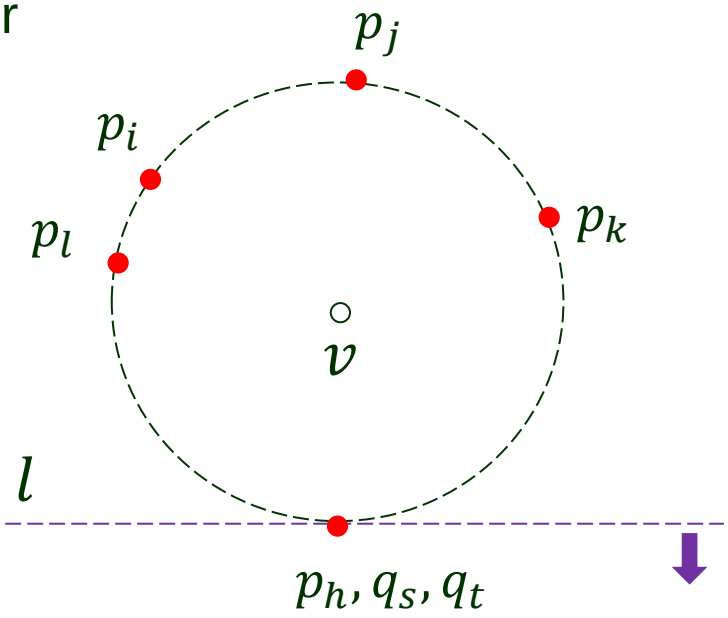
1b) same x-coordinate

- 0 site event and ≥ 2 circle events, or
- 1 site event and ≥ 1 circle event

⇩

a Voronoi vertex of degree ≥ 4

- Handle circle events first.
  - as ≥ 2 vertices coinciding
  - each with degree 3 and 0 length in between
- Handle the sole site event last.



Three events coincide:

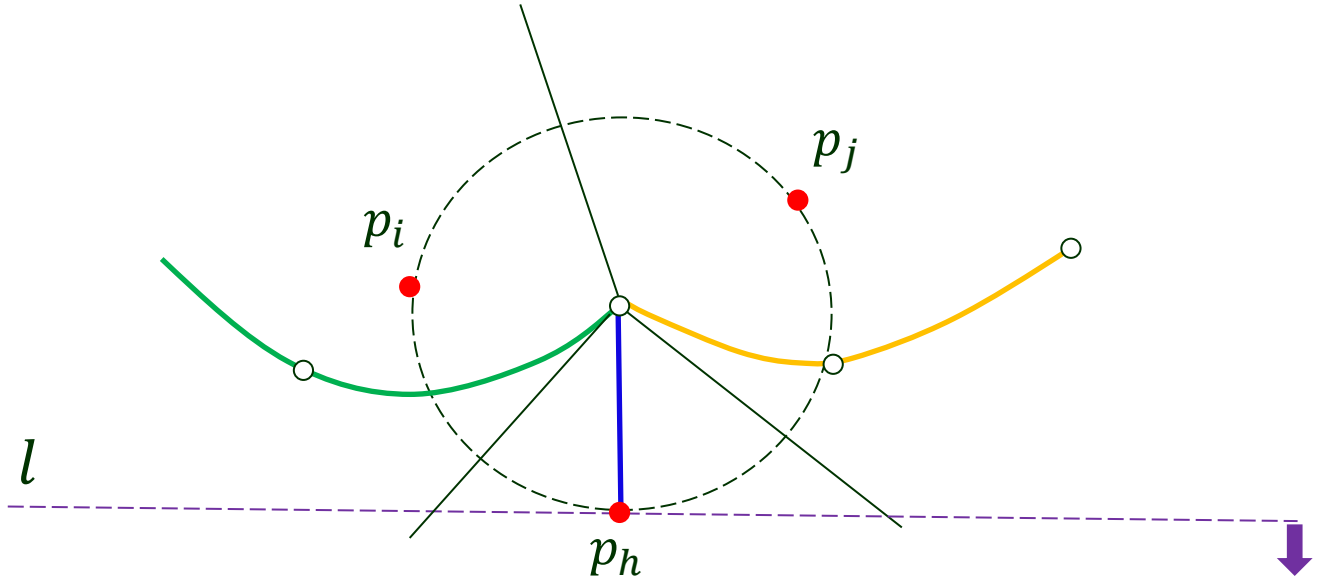
p_h

$q_s = \langle p_l, p_i, p_j \rangle$

$q_t = \langle p_i, p_j, p_k \rangle$

Degeneracy (2)

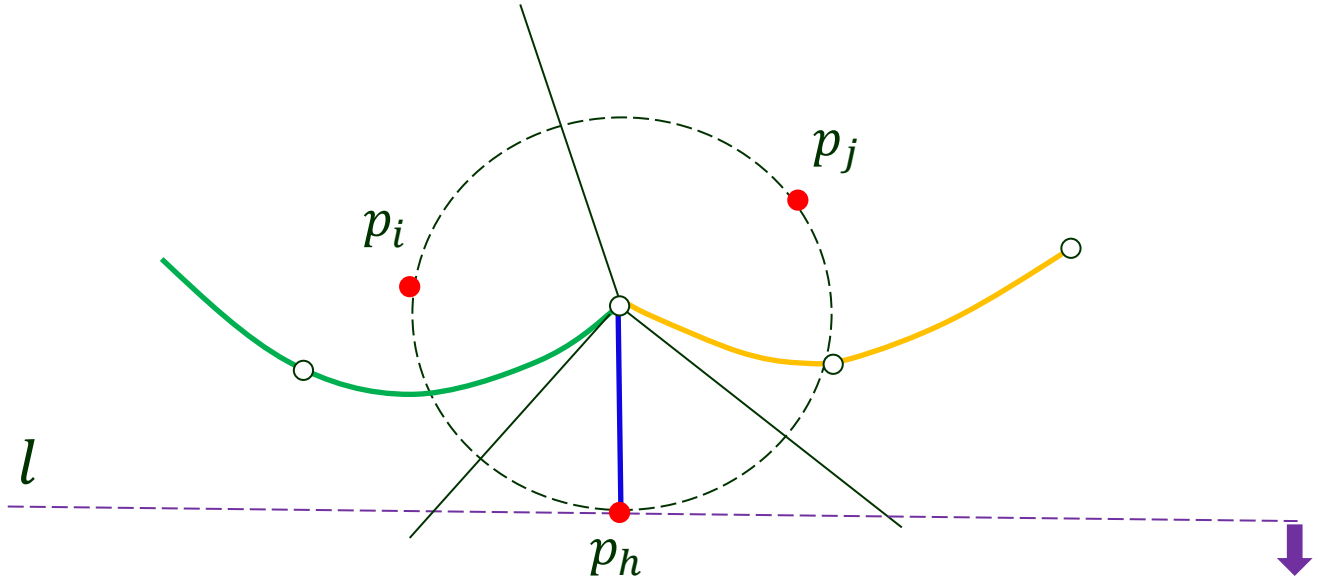
A new site appears right below a break point on the beach line.



Degeneracy (2)

A new site appears right below a break point on the beach line.

Handle it as a circle event, and then add a new arc (for p_h).



VI. Applications of Voronoi Diagrams

Given n points in the plane, we can use their VD to solve many problems efficiently.

VI. Applications of Voronoi Diagrams

Given n points in the plane, we can use their VD to solve many problems efficiently.

Closest Pair: Find two points that are the closest.

VI. Applications of Voronoi Diagrams

Given n points in the plane, we can use their VD to solve many problems efficiently.

Closest Pair: Find two points that are the closest.

All Nearest Neighbors: Find the nearest neighbor of every point.

VI. Applications of Voronoi Diagrams

Given n points in the plane, we can use their VD to solve many problems efficiently.

Closest Pair: Find two points that are the closest.

All Nearest Neighbors: Find the nearest neighbor of every point.

Euclidean Minimum Spanning Tree: Construct a tree of minimum total length whose vertices are the given points.

VI. Applications of Voronoi Diagrams

Given n points in the plane, we can use their VD to solve many problems efficiently.

Closest Pair: Find two points that are the closest.

All Nearest Neighbors: Find the nearest neighbor of every point.

Euclidean Minimum Spanning Tree: Construct a tree of minimum total length whose vertices are the given points.

Triangulation: Join the points by nonintersecting straight line segments so that every region internal to the convex hull is a triangle.

VI. Applications of Voronoi Diagrams

Given n points in the plane, we can use their VD to solve many problems efficiently.

Closest Pair: Find two points that are the closest.

All Nearest Neighbors: Find the nearest neighbor of every point.

Euclidean Minimum Spanning Tree: Construct a tree of minimum total length whose vertices are the given points.

Triangulation: Join the points by nonintersecting straight line segments so that every region internal to the convex hull is a triangle.

Nearest Neighbor Search: With preprocessing allowed, how quickly can a nearest neighbor of a new given query point q be found?