

# Voronoi Diagrams

(Slides mostly by Allen Miu)

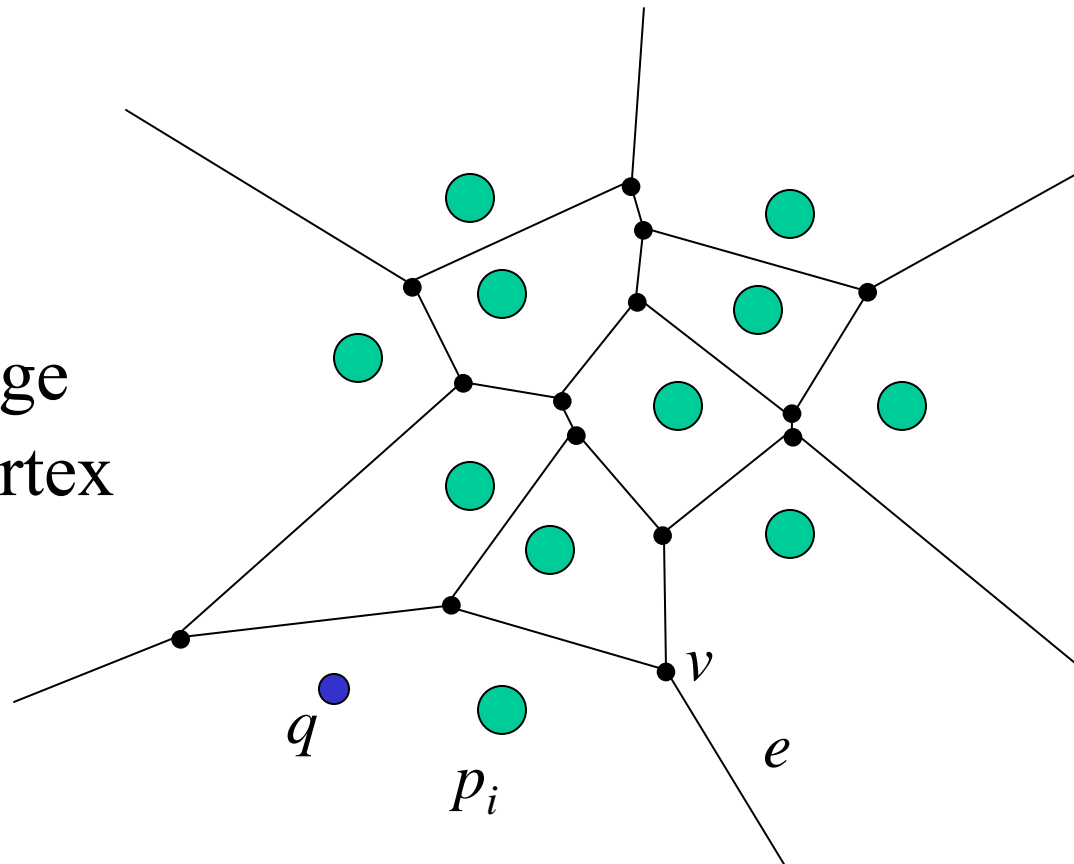
# Post Office: What is the area of service?

$p_i$  : 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  $\|q-p_i\| < \|q-p_j\|$ , for each  $p_j \in P, j \neq i$ .

# Demo

<http://www.diku.dk/students/duff/Fortune/>

<http://wwwpi6.fernuni-hagen.de/GeomLab/VoroGlide/>

# Jeff's Erickson Web Page



See also the [implementation page](#) from Christopher Gold's site [www.Voronoi.com](http://www.Voronoi.com).

Enough already!!

[Delaunay triangulations and farthest point Delaunay triangulations using 3d convex hulls](#) by [Daniel Mark Abrahams-Gessel](#), fortunately stolen by [Anirudh Modi](#) before the original page was taken off the Web. **This is the best one!**

[Convex hulls, Delaunay triangulations, Voronoi diagrams, and proximity graphs](#) by James E. Baker, Isabel F. Cruz, Luis D. Lejter, Giuseppe Liotta, and Roberto Tamassia. [Source code](#) is available.

[Incremental Delaunay triangulations and Voronoi diagrams](#) by Frank Bossen

[Voronoi Diagram/Delaunay Triangulation](#) by [Paul Chew](#) uses a randomized incremental algorithm with "brute force" point location.

[2-Site Voronoi diagrams](#) by [Matt Dickerson](#), from the [Middlebury College Undergraduate Research Project in Computational Geometry](#)

The [convex hull/Voronoi diagram applet](#) from the [GeomNet project](#) provides a secure Java wrapper for existing (non-Java) code. The applet calls [qhull](#) to build its convex hulls and [Steve Fortune's sweep2](#) to build its Voronoi diagrams. A [forms interface](#) to the same programs is also available.

[VoroGlide](#), by Christian Icking, Rolf Klein, Peter Köllner, and Lihong Ma. Smoothly maintains the convex hull, Voronoi diagram, and Delaunay triangulation as points are moved, illustrates incremental construction of the Delaunay triangulation, and includes a recorded demo. Now on a faster server!

[Delaunay triangulations](#) by Geoff Leach compares several (very) naive algorithms. [Source code](#) is available.

[Bisectors](#) and [Voronoi diagrams](#) under [convex \(polygonal\) distance functions](#) by [Lihong Ma](#). The diagram is updated on the fly while sites *or vertices of the unit ball* are inserted, deleted, or dragged around. Very cool!

[Delaunay triangulations and Dirichlet tessellations](#) and a short applet-enhanced [tutorial](#) by Eric C. Olson

[The Voronoi Game](#) by Dennis Shasha. Try to place points to maximize the area of your Voronoi regions.

[Higher-order Voronoi diagrams](#) by [Barry Schaudt](#)

[Tessy](#), yet another interactive Voronoi/Delaunay demo from [Keith Voegelé](#). Java *not* required.

[ModeMap](#), by David Watson, draws Voronoi diagrams, Delaunay triangulations, natural neighbor circles (circumcircles of Delaunay triangles), and (for the very patient) radial density contours on the *sphere*. Don't give it more than 80 points.

[Delaunay Triangulation](#) from [Zhiyuan Zhao's JAVA Gallery of Geometric Algorithms](#)

[Delaunay Triangulation Demo](#) at ESSI, Université de Nice/Sophia-Antipolis, France. X terminal required instead of Java. *Extremely* slow, at least on this side of the Atlantic.

March 1, 2005

Lecture 8: Voronoi Diagrams

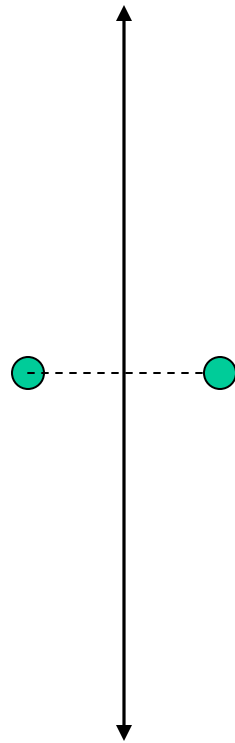
# Outline

- 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

# Voronoi Diagram Example: 1 site



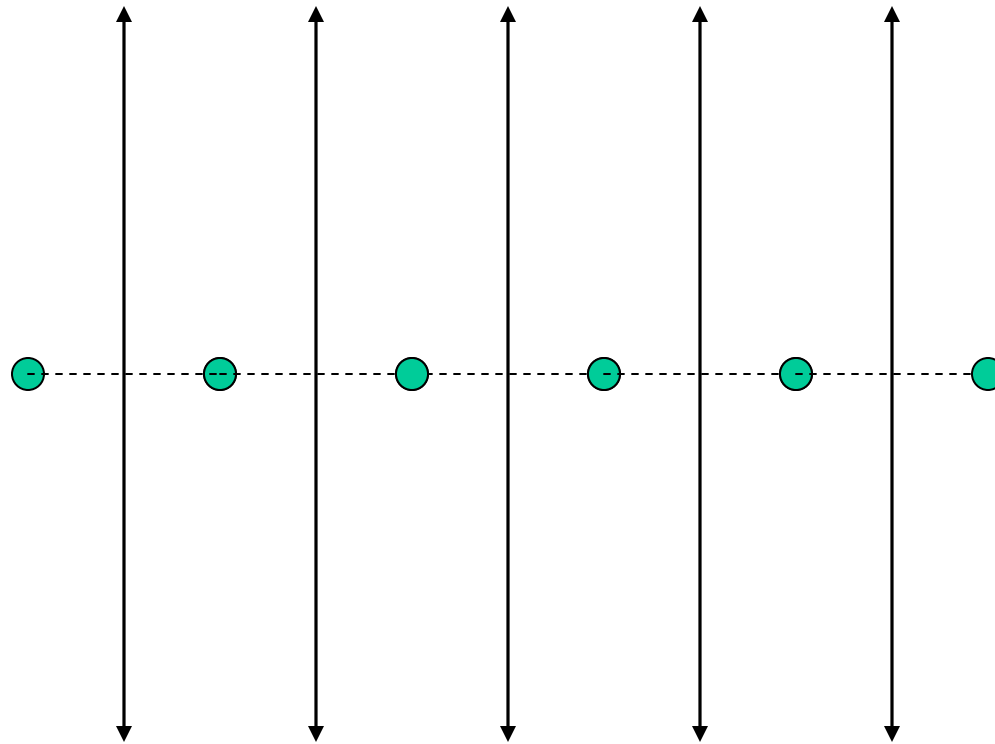
# Two sites form a perpendicular bisector



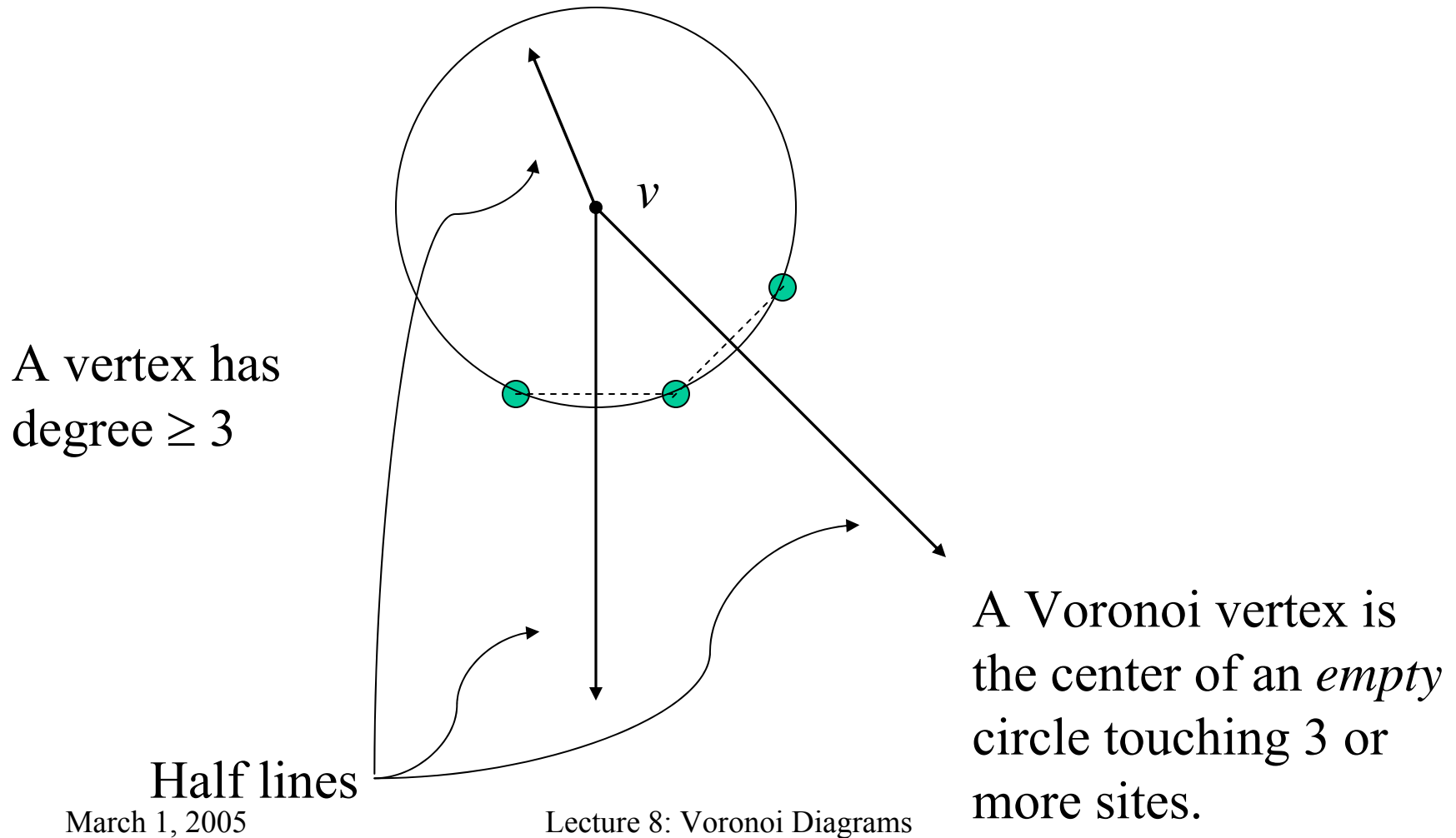
Voronoi Diagram is a line that extends infinitely in both directions, and the two half planes on either side.



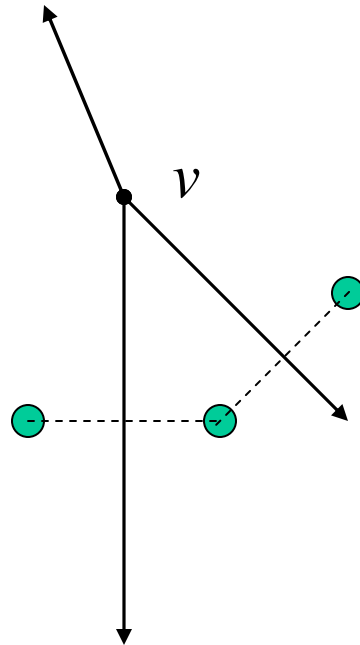
# Collinear sites form a series of parallel lines



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



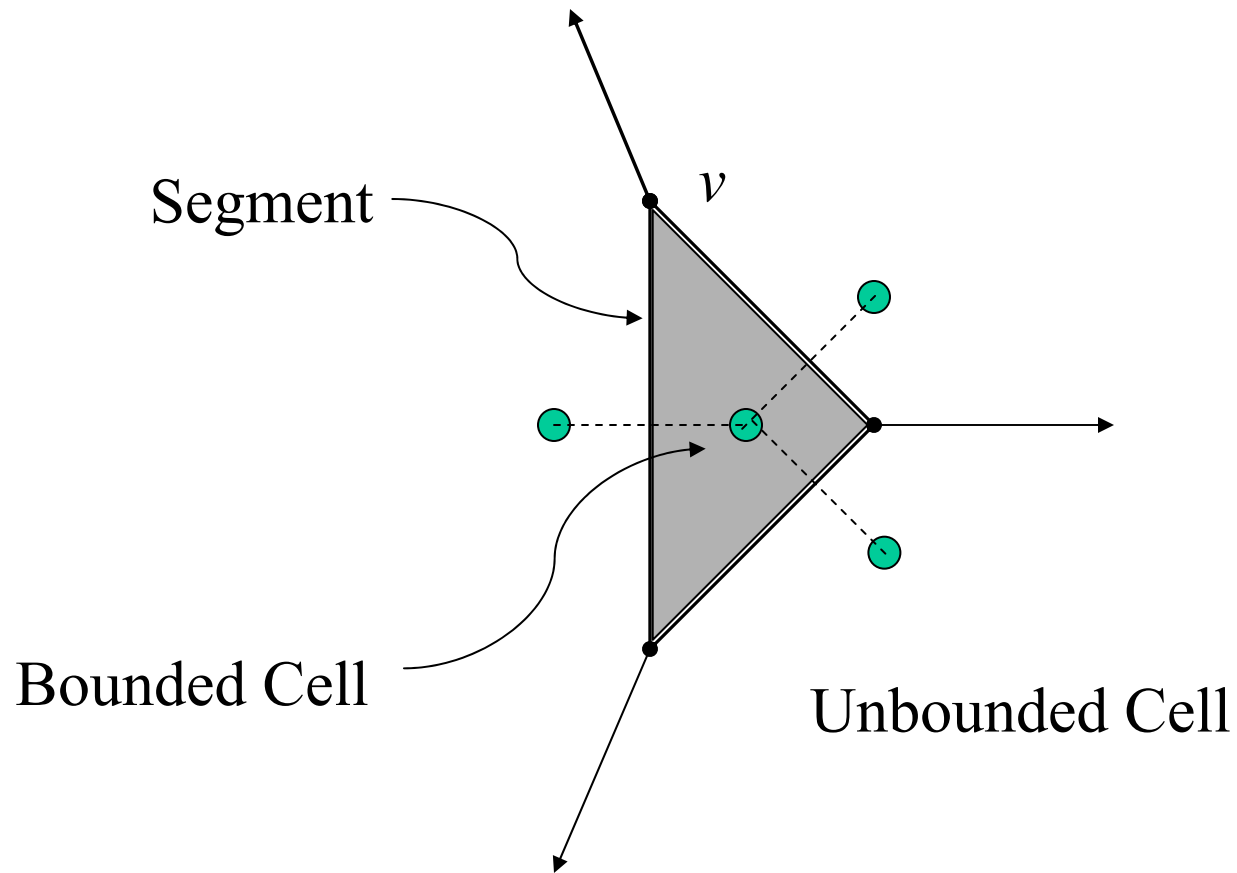
# Voronoi Cells and Segments



March 1, 2005

Lecture 8: Voronoi Diagrams

# Voronoi Cells and Segments

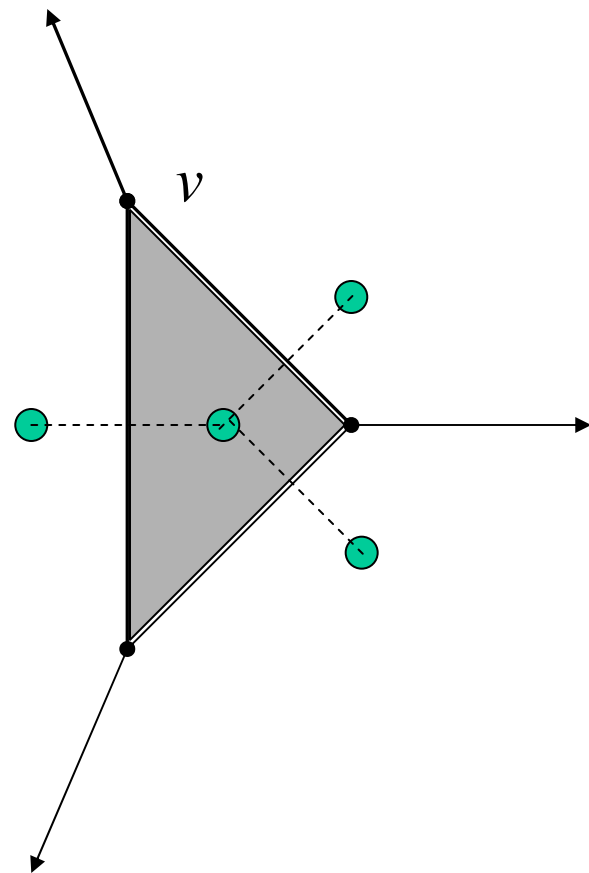


# Pop quiz

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

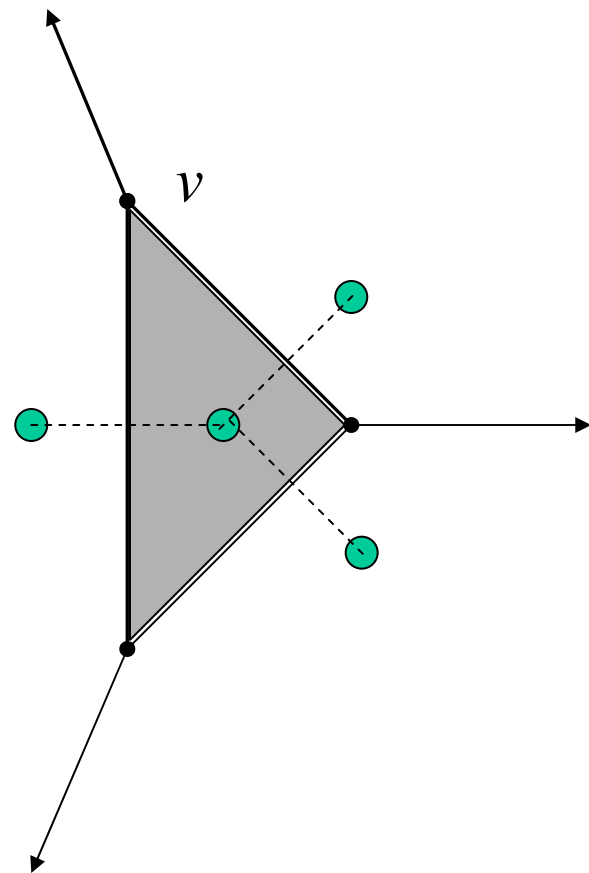


# Pop quiz

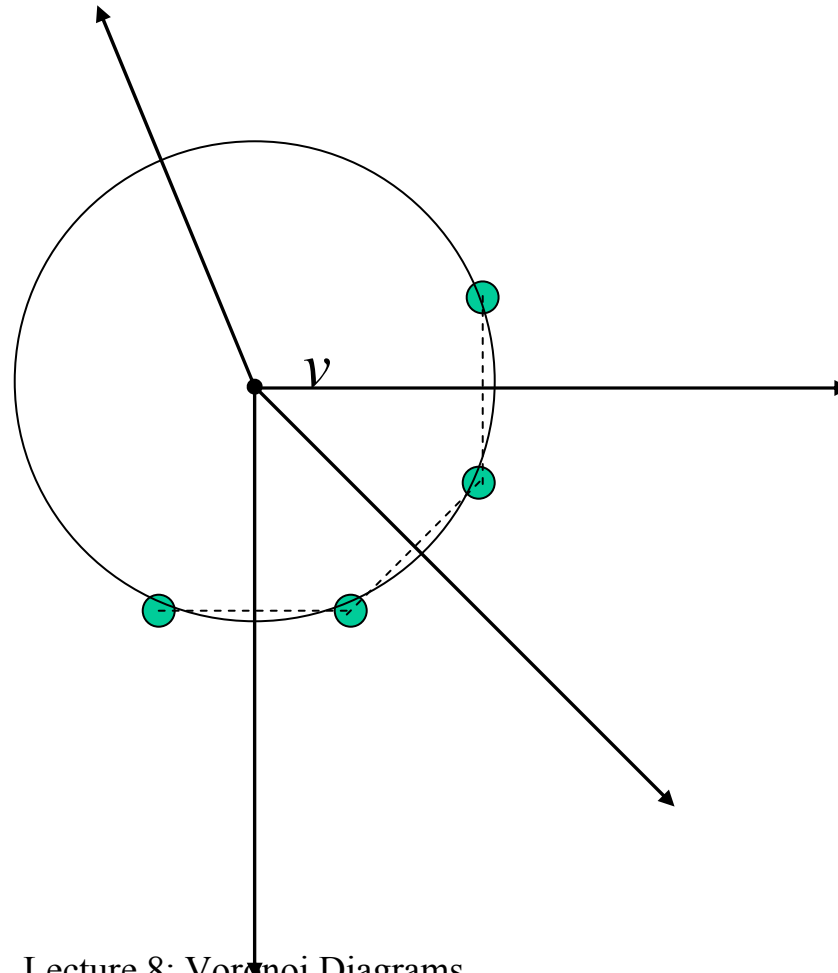
Which of the following is true for  
2-D Voronoi diagrams?

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!

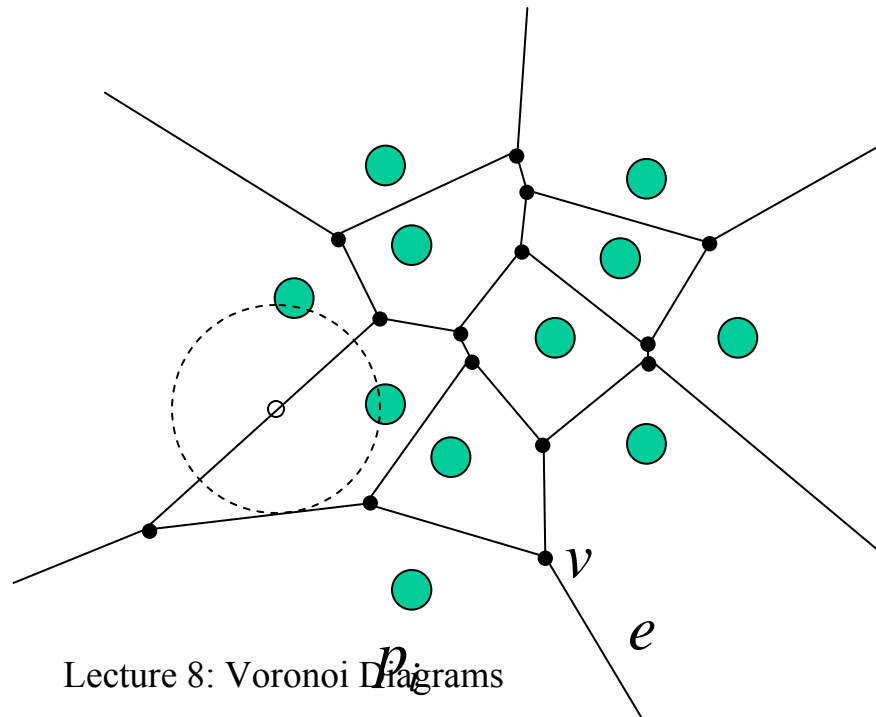


# Summary of Voronoi Properties

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



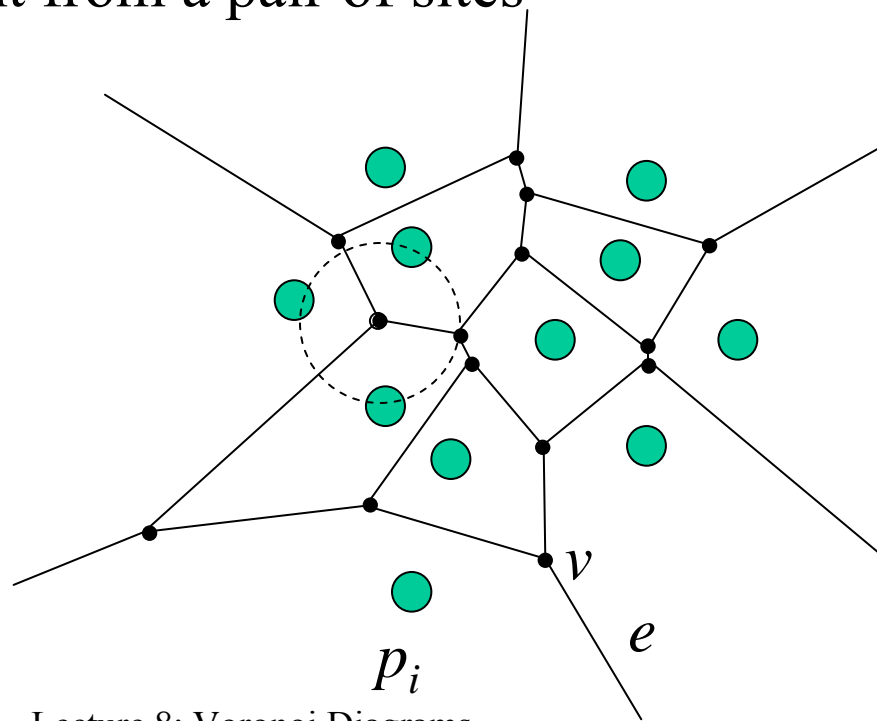


# Summary of Voronoi Properties

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

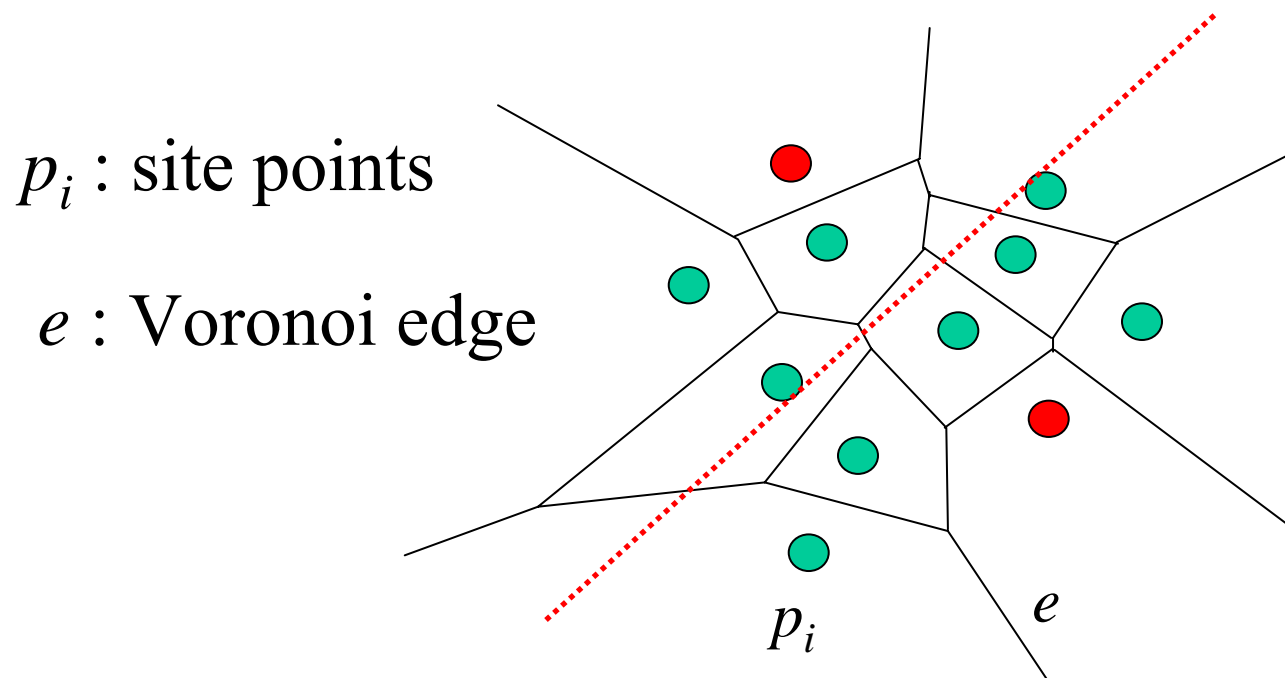
- A Voronoi vertex is an intersection of 3 more segments, each equidistant from a pair of sites

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



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

Intuition: Not all bisectors are Voronoi edges!



# 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)

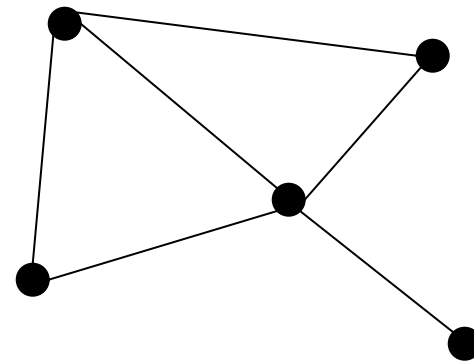
- Euler's Formula: for connected, planar graphs,  
 $v - e + f = 2$

Where:

$v$  is the number of vertices

$e$  is the number of edges

$f$  is the number of faces



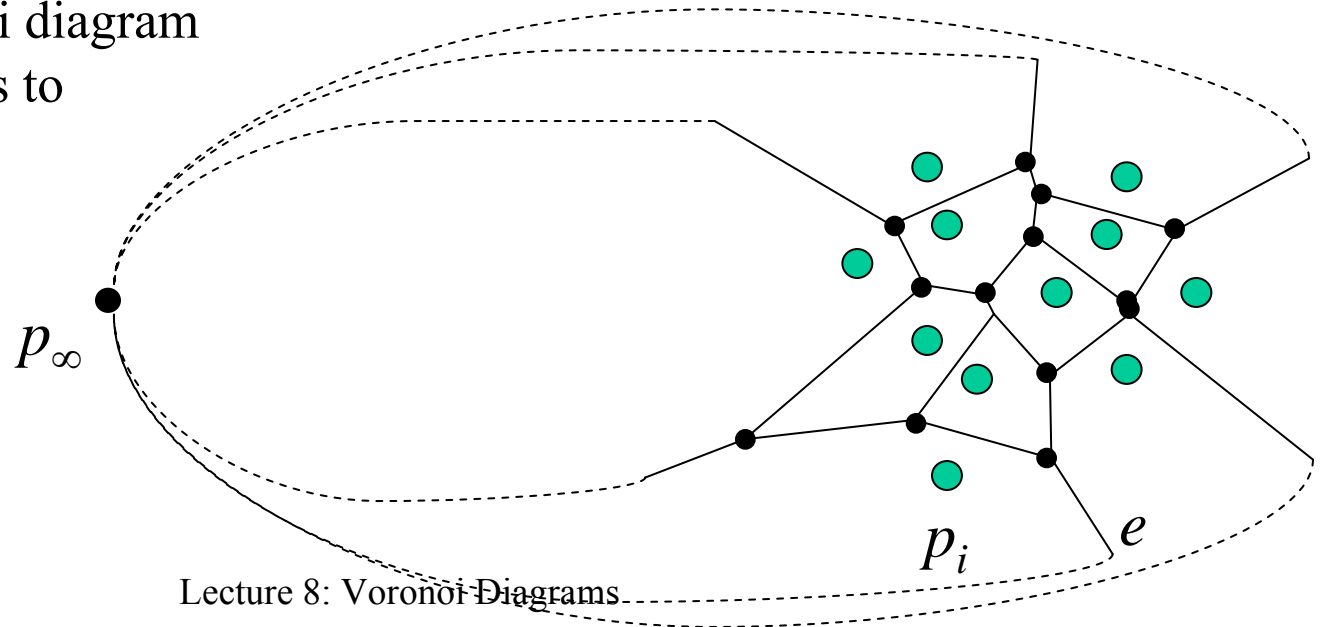
# Voronoi diagrams have linear complexity $\{v, e = O(n)\}$

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,

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

and

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

so

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

together with

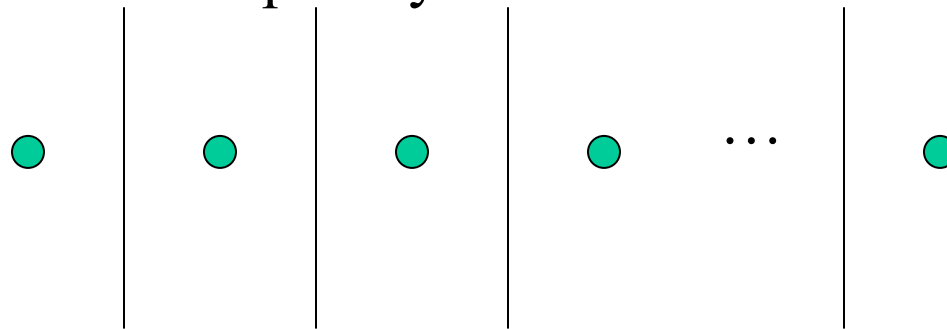
$$(v + 1) - e + n = 2$$

we get, for  $n \geq 3$

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

# A really degenerate case

- The graph has “loops”, i.e., edges from  $p_\infty$  to itself
- The “standard” Euler formula does not apply
- But:
  - One can extend Euler formula to loops (each loop creates a new face) and show that it still works
  - Or, one can recall that the Voronoi diagram for this case has still a linear complexity



# 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

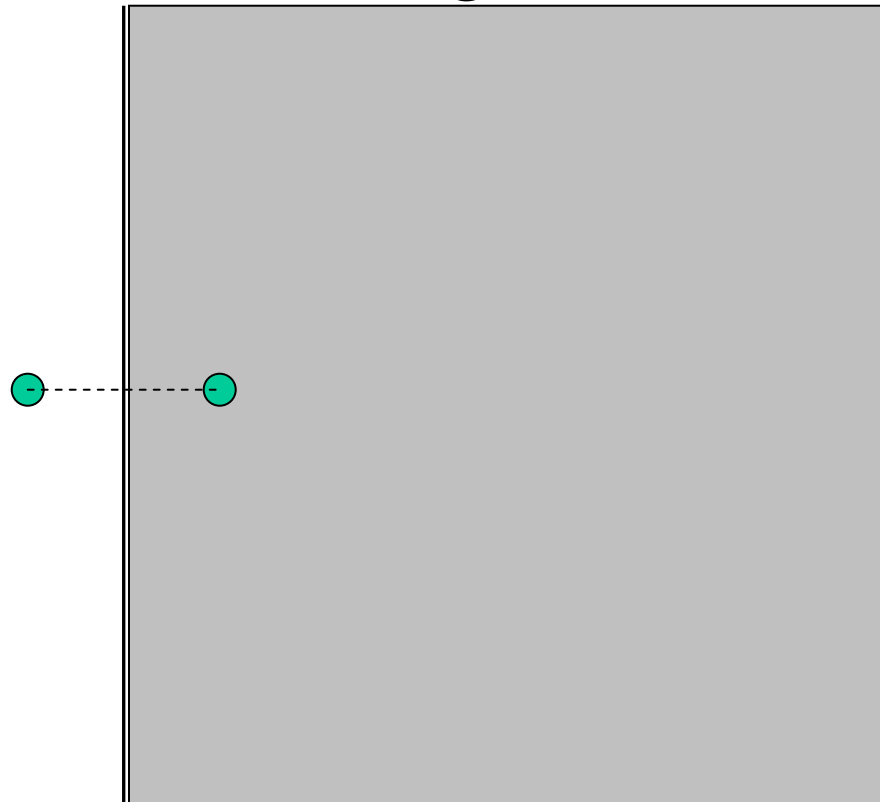
# Constructing Voronoi Diagrams

Given a half plane intersection algorithm...



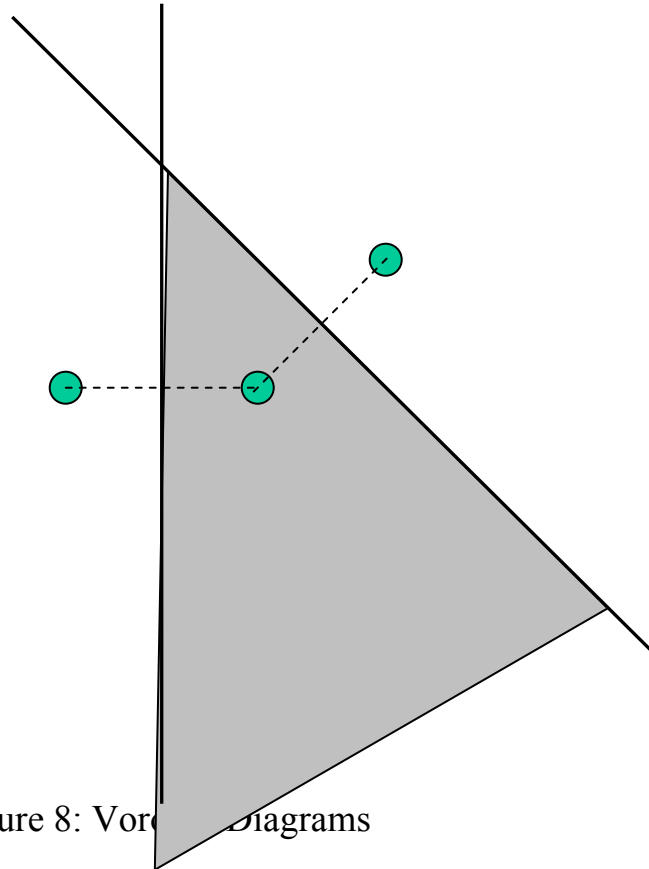
# Constructing Voronoi Diagrams

Given a half plane intersection algorithm...



# Constructing Voronoi Diagrams

Given a half plane intersection algorithm...

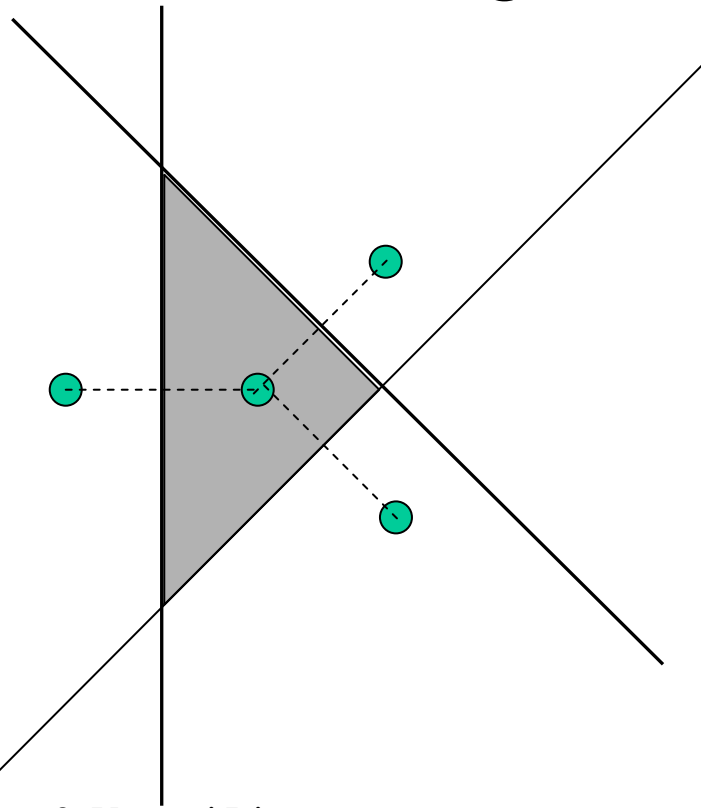


# Constructing Voronoi Diagrams

Given a half plane intersection algorithm...

Repeat for each site

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



# Faster Algorithm

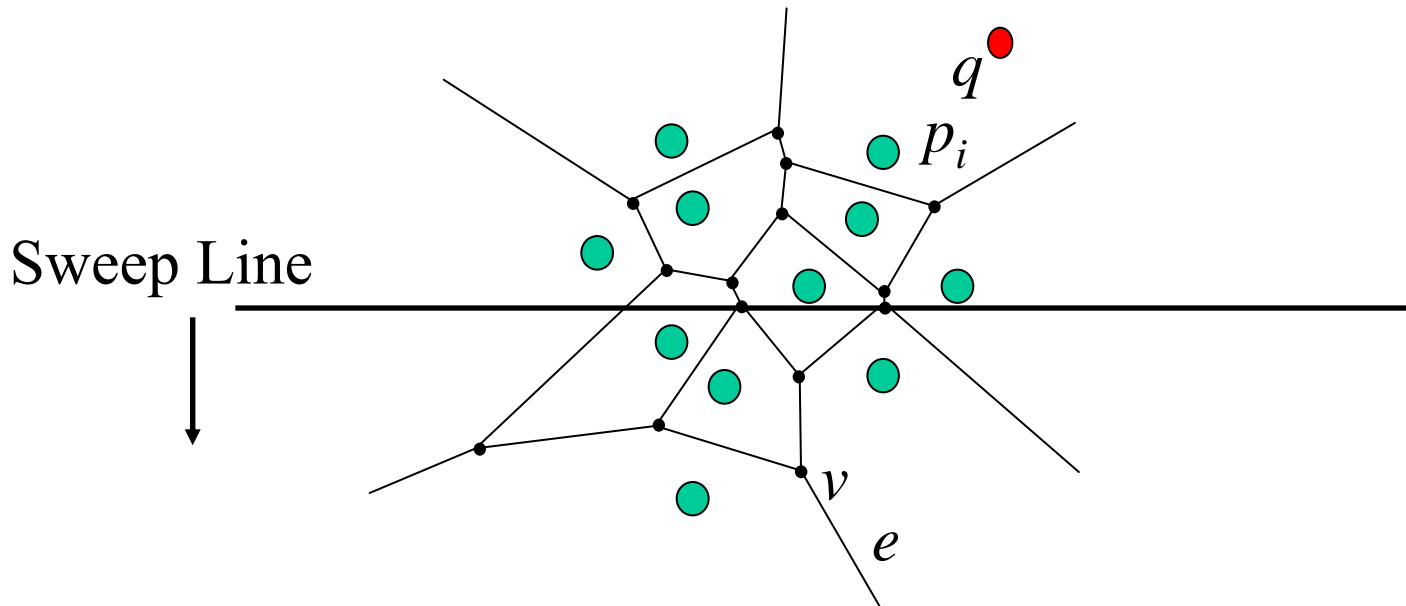
- Fortune's Algorithm:
  - Sweep line approach
  - Voronoi diagram constructed as horizontal line sweeps the set of sites from top to bottom
  - Incremental construction:
    - maintains portion of diagram which cannot change due to sites below sweep line,
    - keeps track of incremental changes for each site (and Voronoi vertex) it “sweeps”

# Algorithms Outline

- Ideas
- Data structures
- Events

# Invariant

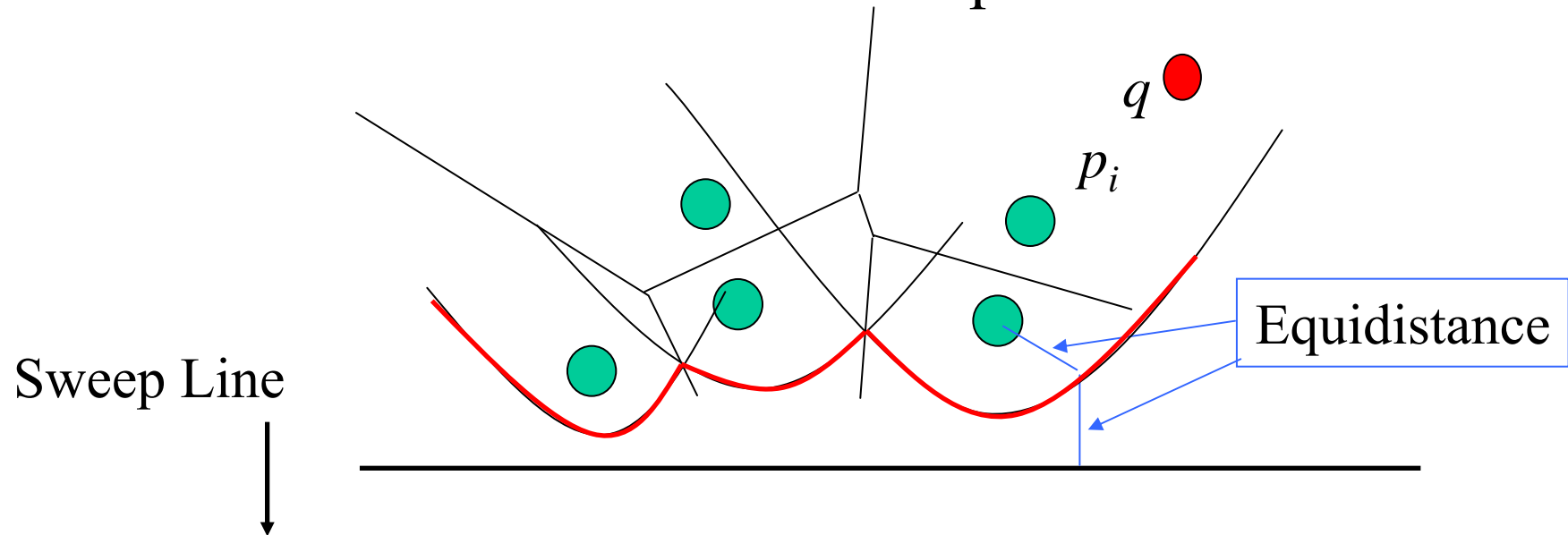
What is the invariant we are looking for?



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).

# Beach line

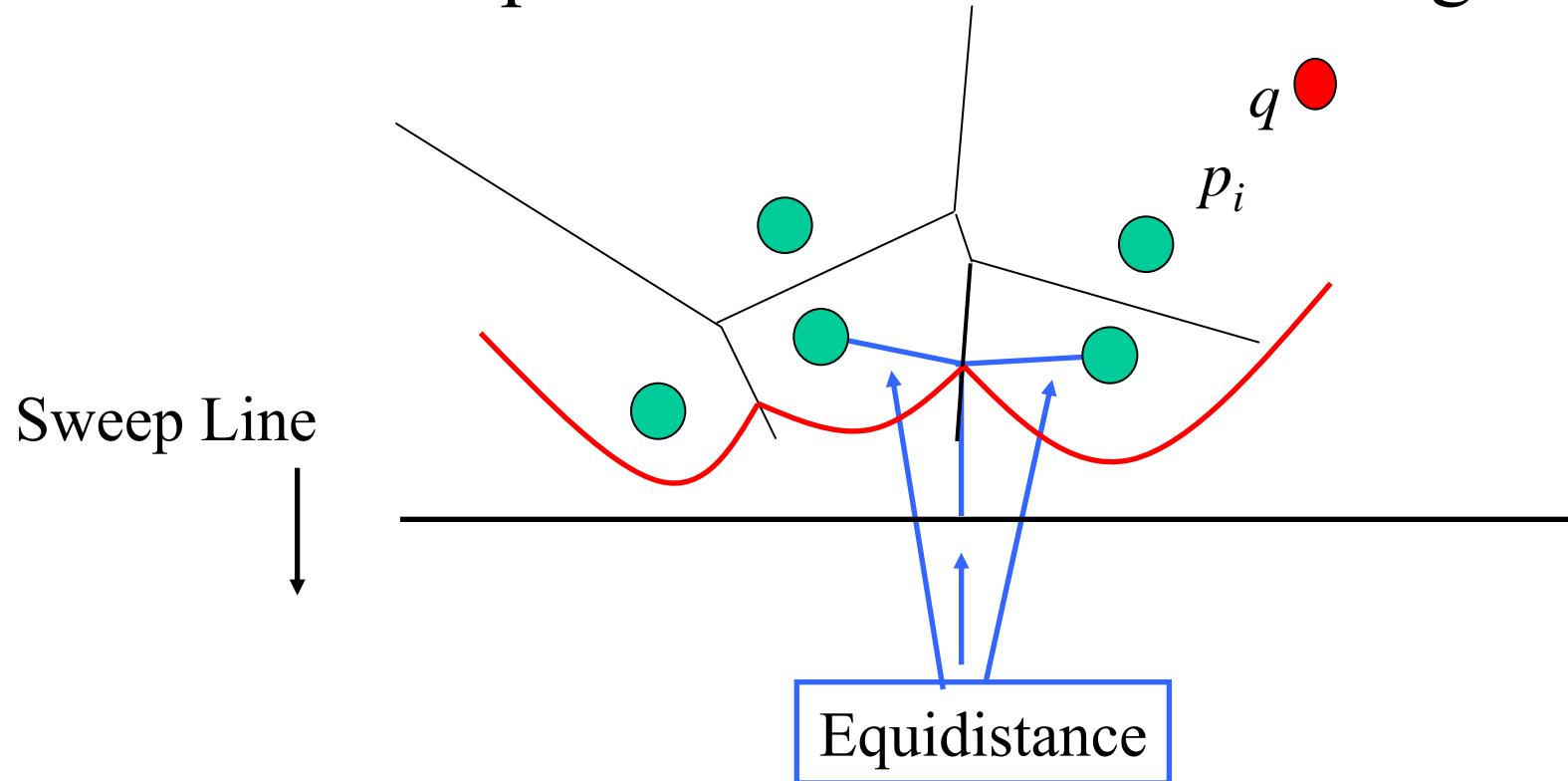
Which points are closer to a site above the sweep line than to the sweep line itself?



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

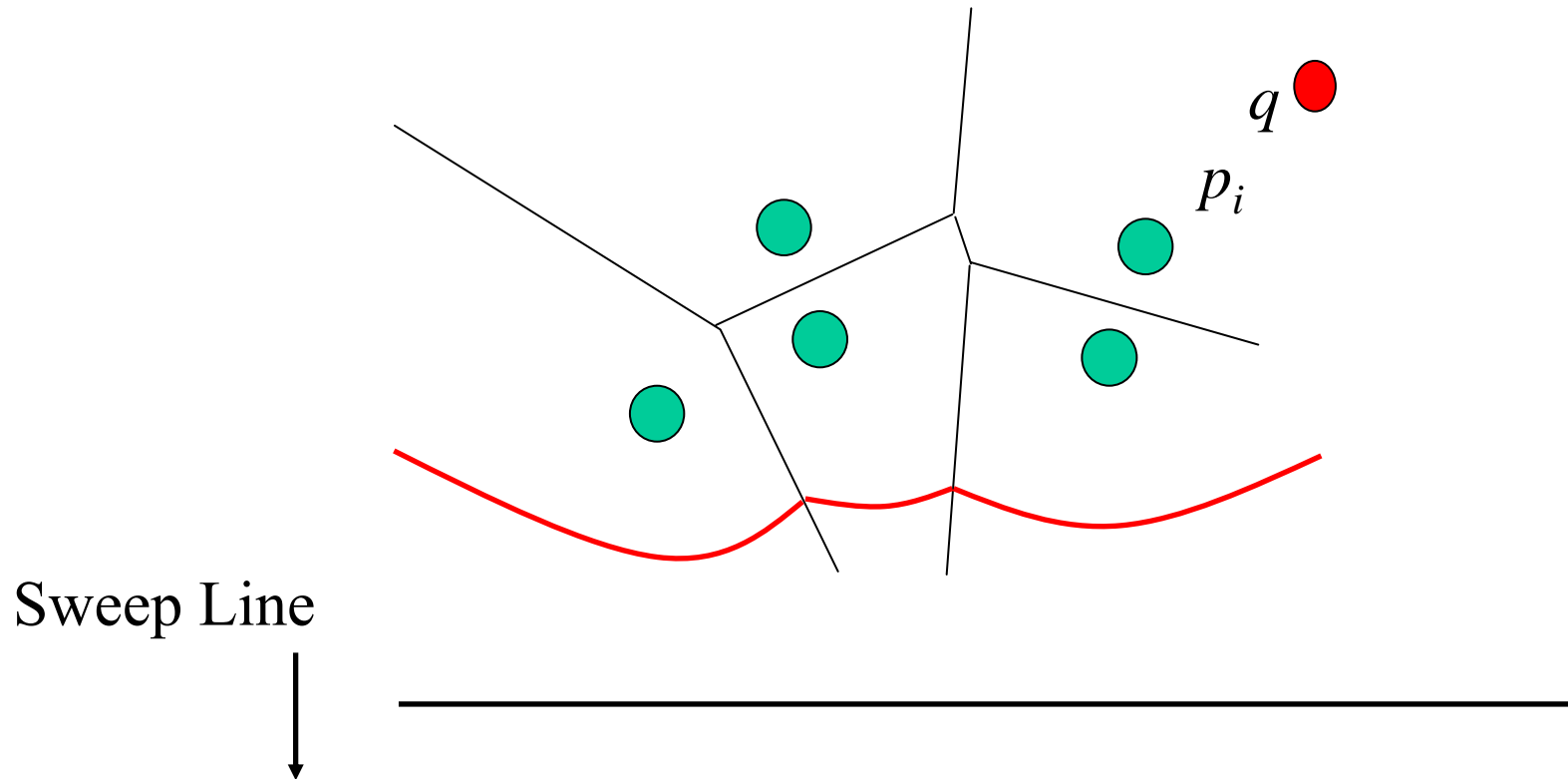
# Edges

Break points trace out Voronoi edges.

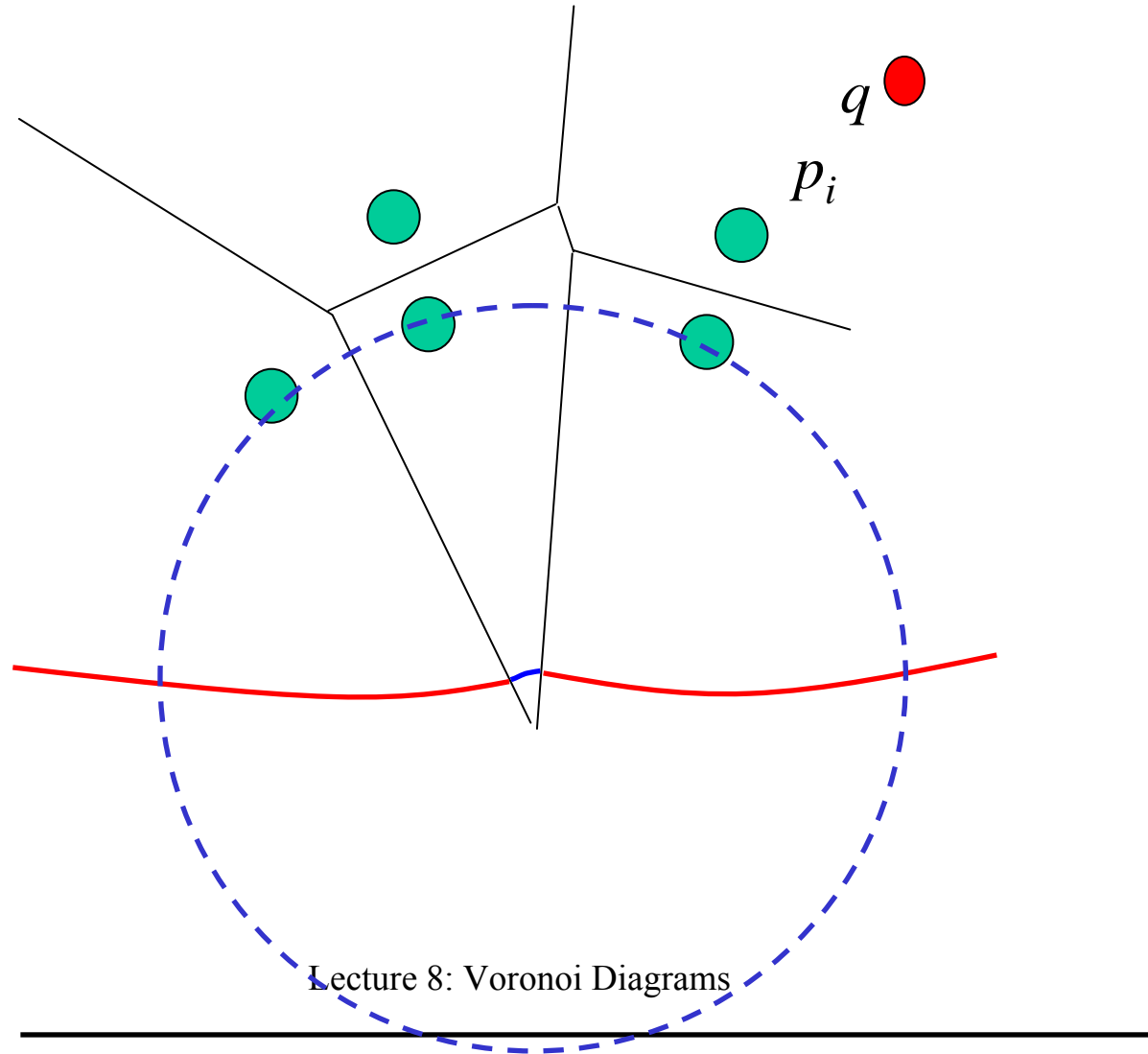




Arcs flatten out as sweep line moves down.



Eventually, the middle arc disappears.



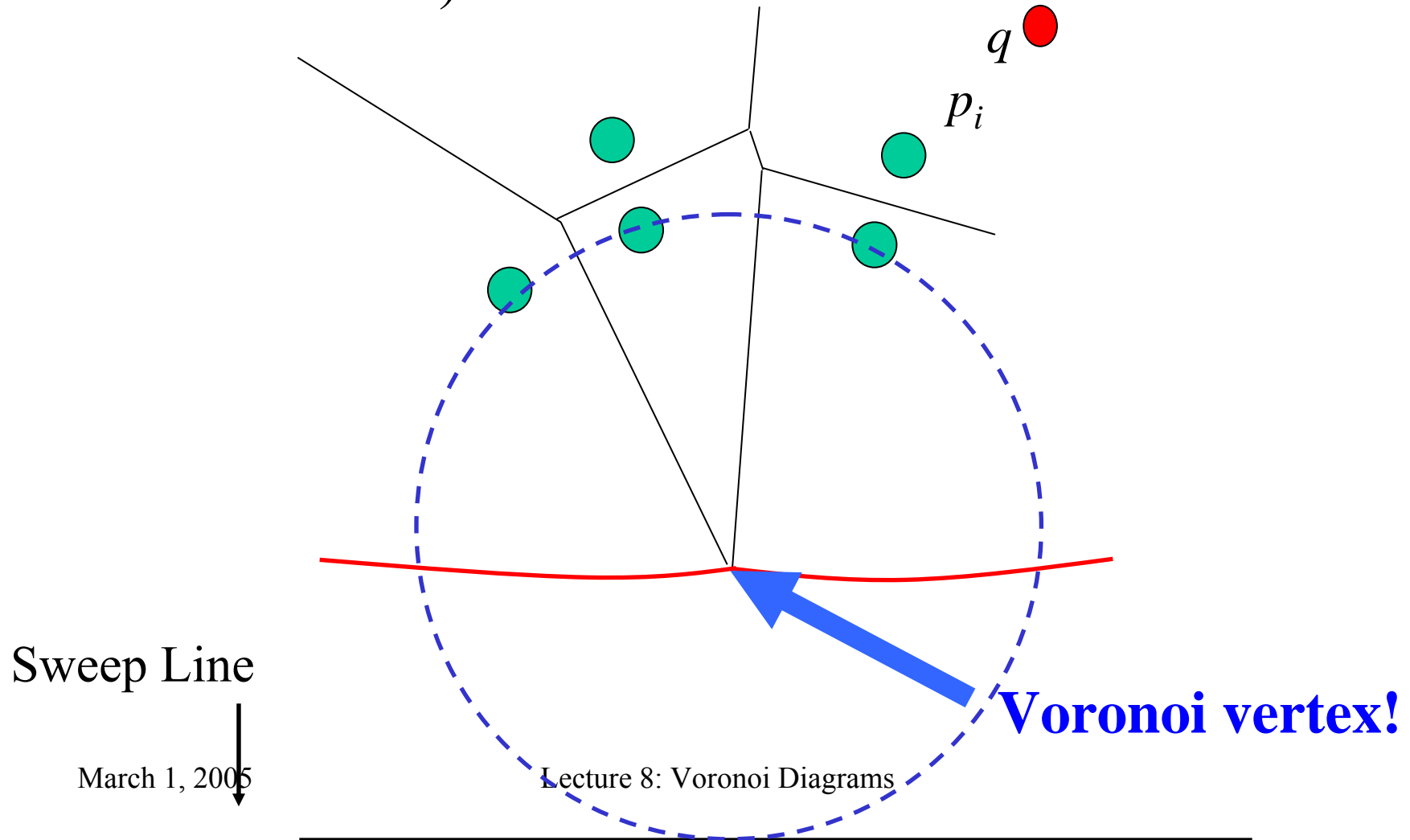
Sweep Line

March 1, 2005

Lecture 8: Voronoi Diagrams

# Circle Event

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



Sweep Line

March 1, 2005

Lecture 8: Voronoi Diagrams

# 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

# Algorithms Outline

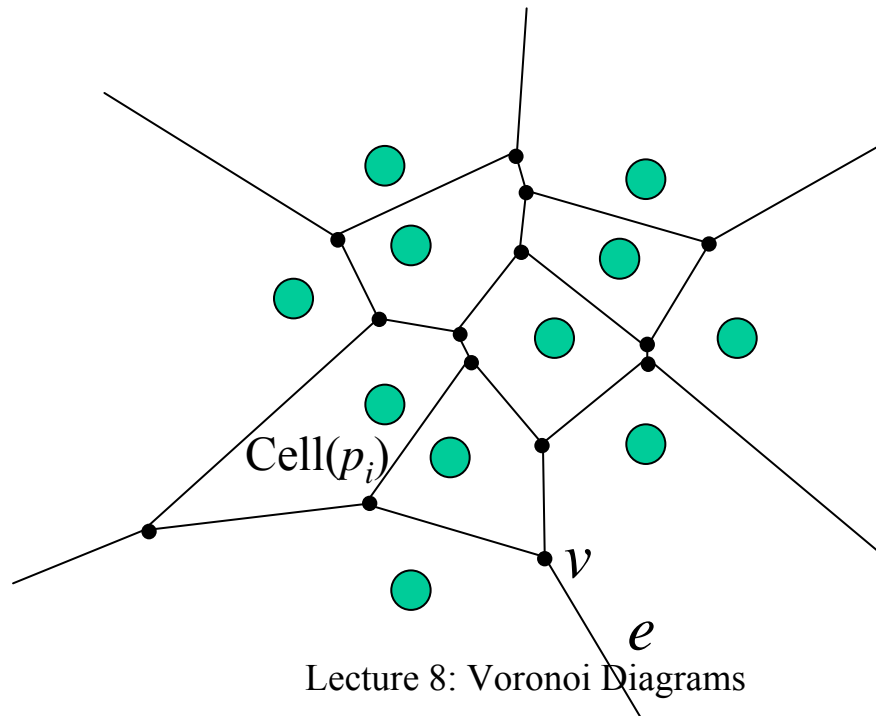
- Ideas
- Data structures
- Events

# 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

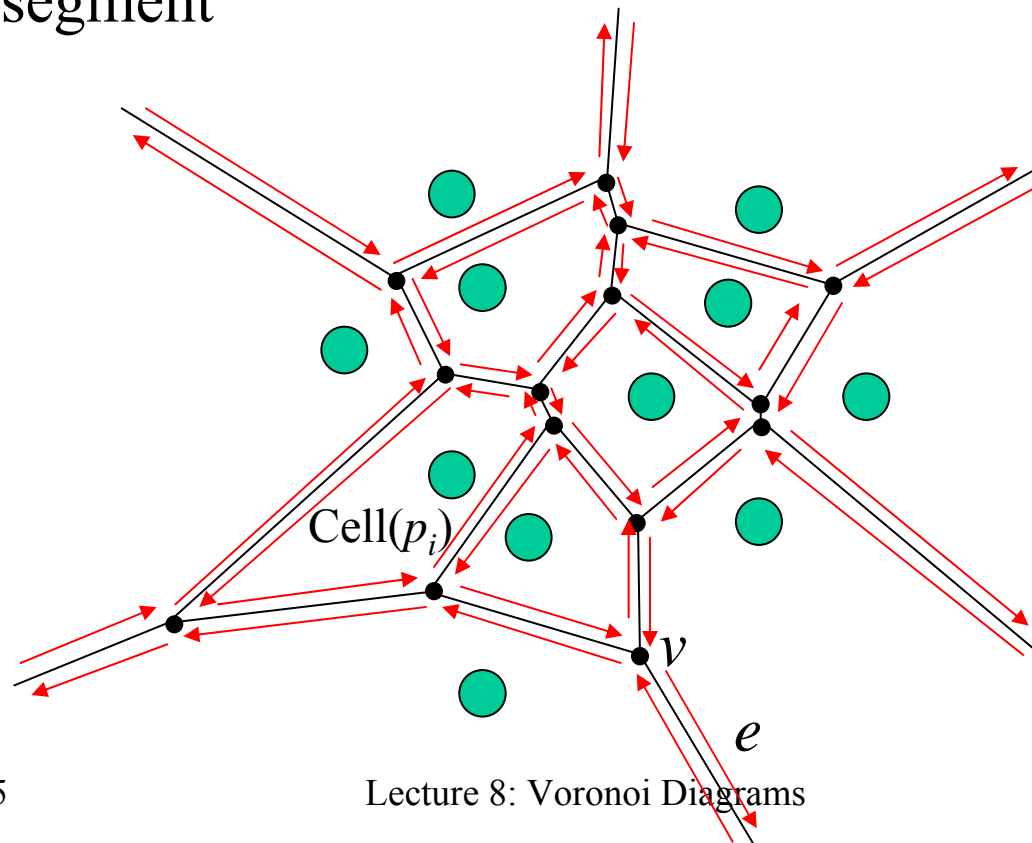
# Doubly Linked List ( $D$ )

- 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



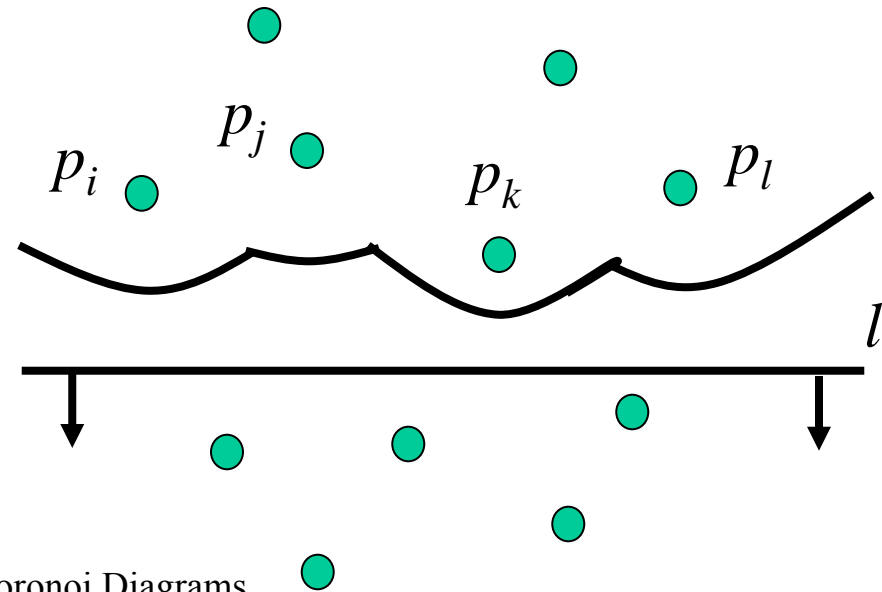
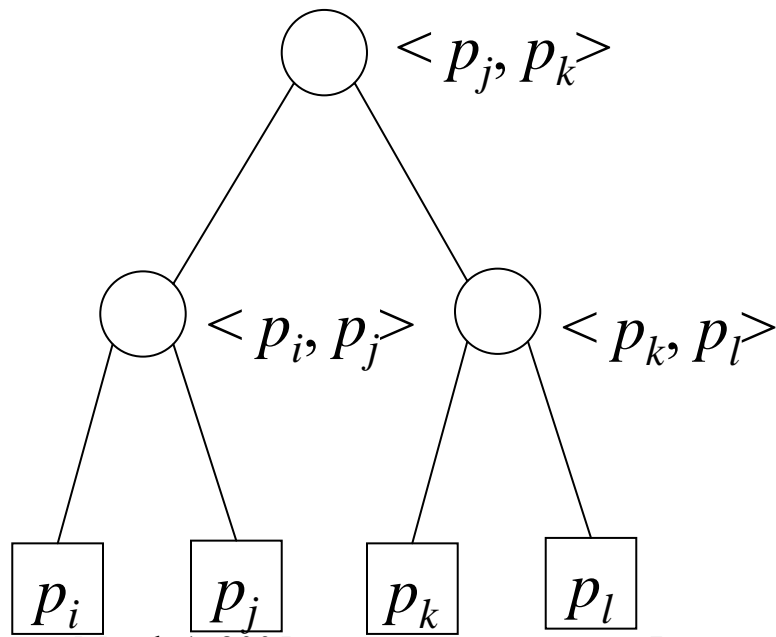


# Doubly Linked List ( $D$ )

- Cell Table
  - $\text{Cell}(p_i)$  : 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

# Summarizing Data Structures

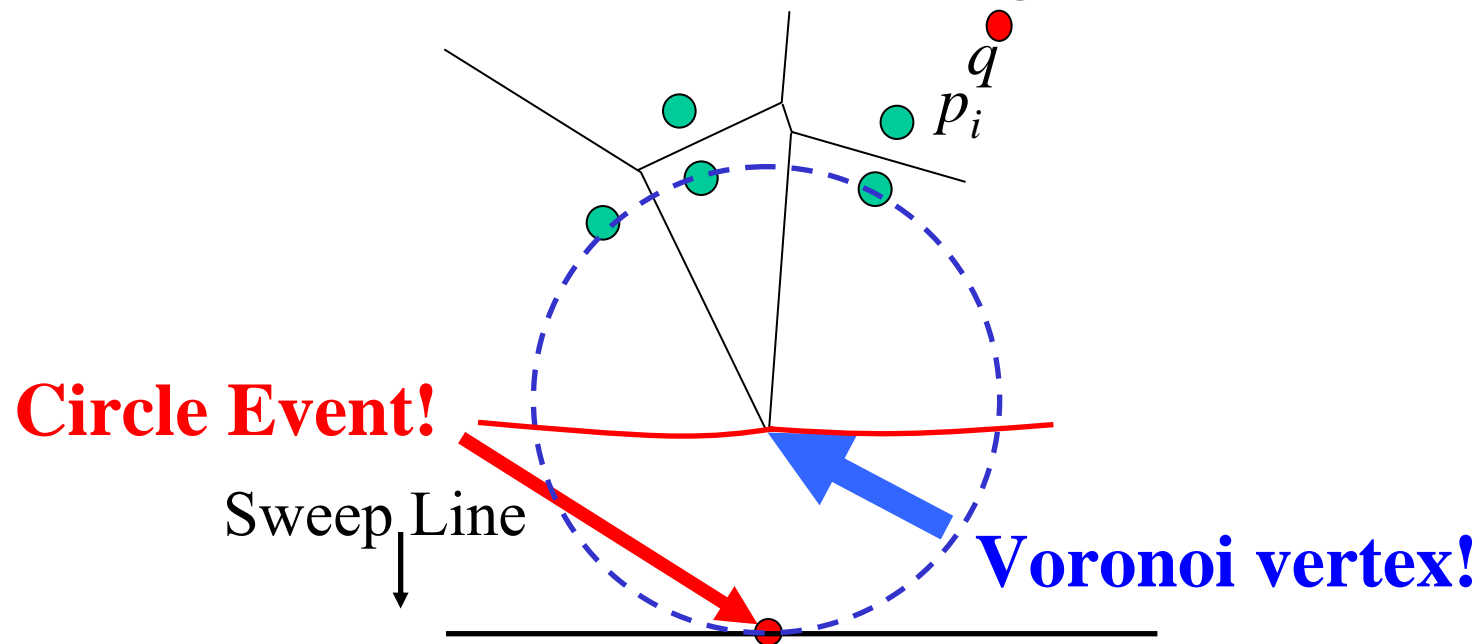
- Current state of the Voronoi diagram
  - Doubly linked list of half-edge, vertex, cell records
- Current state of the beach line
  - Keep track of break points
    - Inner nodes of binary search tree; represented by a tuple
  - Keep track of arcs currently on beach line
    - Leaf nodes of binary search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

# Algorithms Outline

- Ideas
- Data structures
- Events

# Circle Event

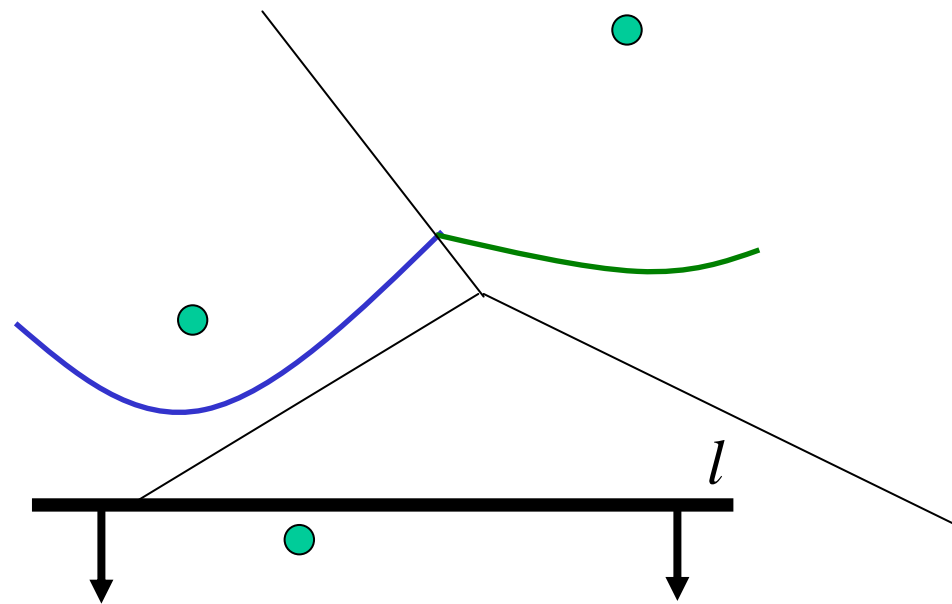
An arc disappears whenever an empty circle touches three or more sites and is tangent to the sweep line.



Sweep line helps determine that the circle is indeed empty.

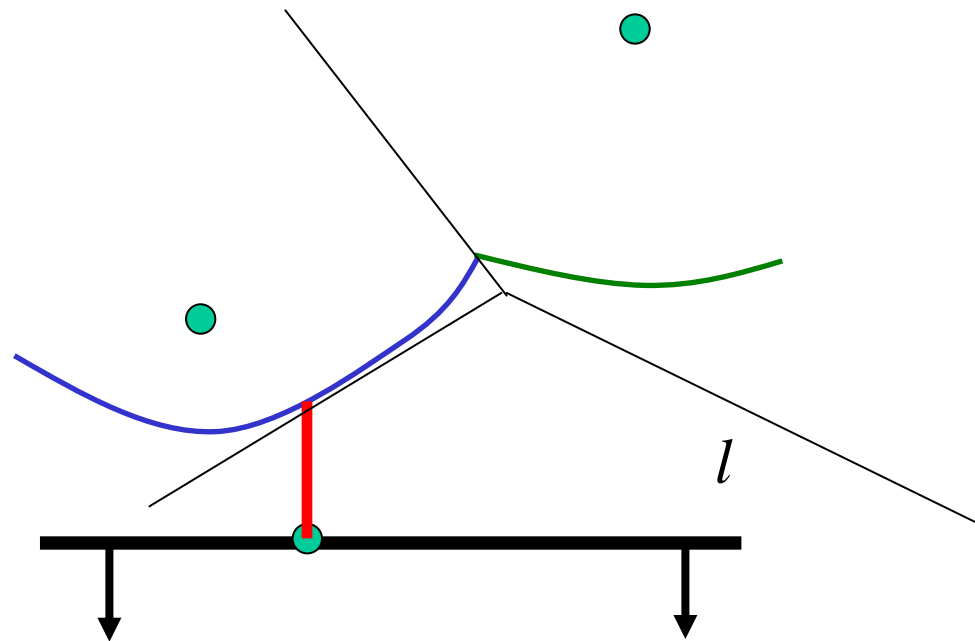
# Site Event

A new arc appears when a new site appears.



# Site Event

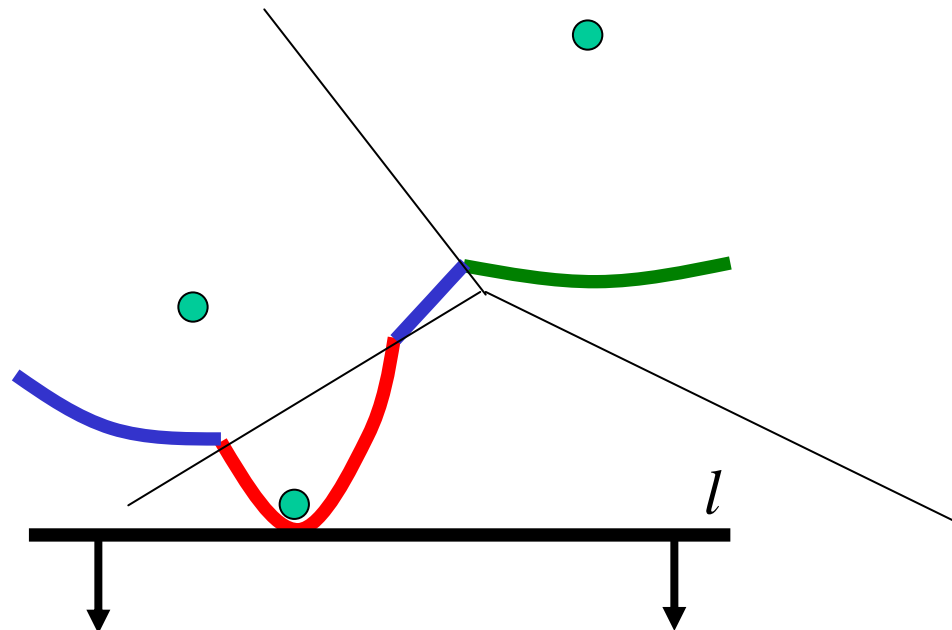
A new arc appears when a new site appears.





# Site Event

Original arc above the new site is broken into two  
→ Number of arcs on beach line is  $O(n)$



# Event Queue Summary

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

# “Algorithm”

## 1. Initialize

- Event queue  $Q \leftarrow$  all site events
- Binary search tree  $T \leftarrow \emptyset$
- Doubly linked list  $D \leftarrow \emptyset$

## 2. While $Q$ not $\emptyset$ ,

- Remove event ( $e$ ) from  $Q$  with largest  $y$ -coordinate
  - $\text{HandleEvent}(e, T, D)$

# 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

# 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

# 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

# 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	$O(1)$



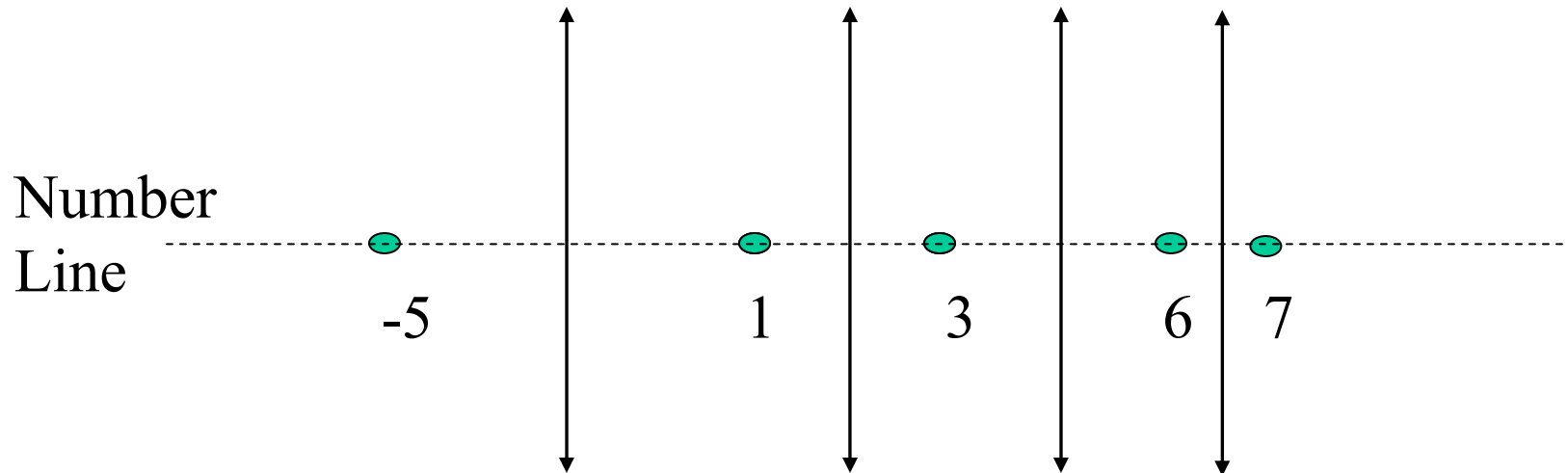
# Total Running Time

- Each new site can generate at most two new arcs  $\rightarrow$  beach line can have at most  $2n - 1$  arcs
- Each “false circle event” can be charged to a real event  $\rightarrow O(n)$  events
- Site/Circle Event Handler  $O(\log n)$

$\rightarrow O(n \log n)$  total running time

# Is Fortune's Algorithm Optimal?

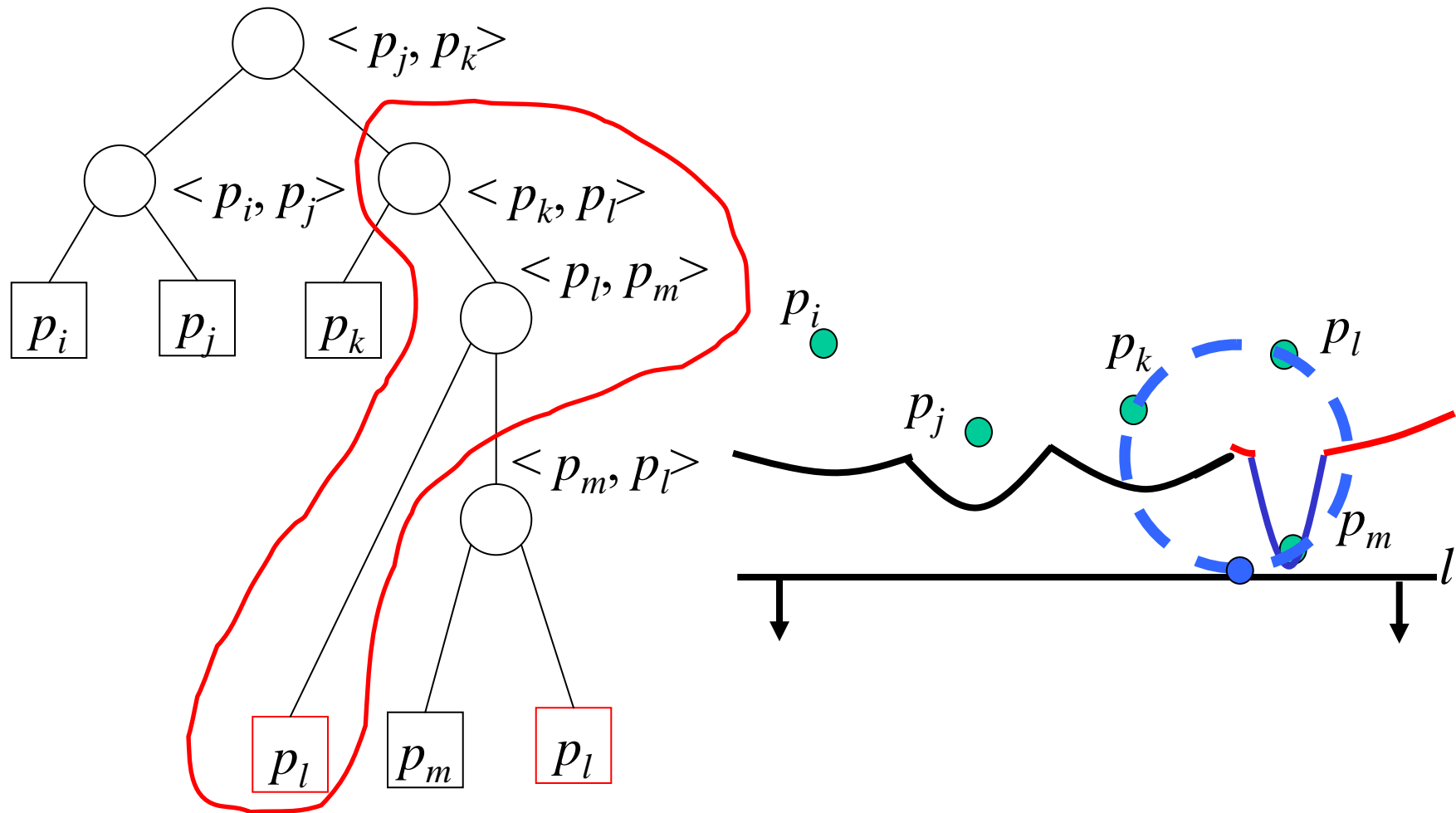
- We can sort numbers using any algorithm that constructs a Voronoi diagram!



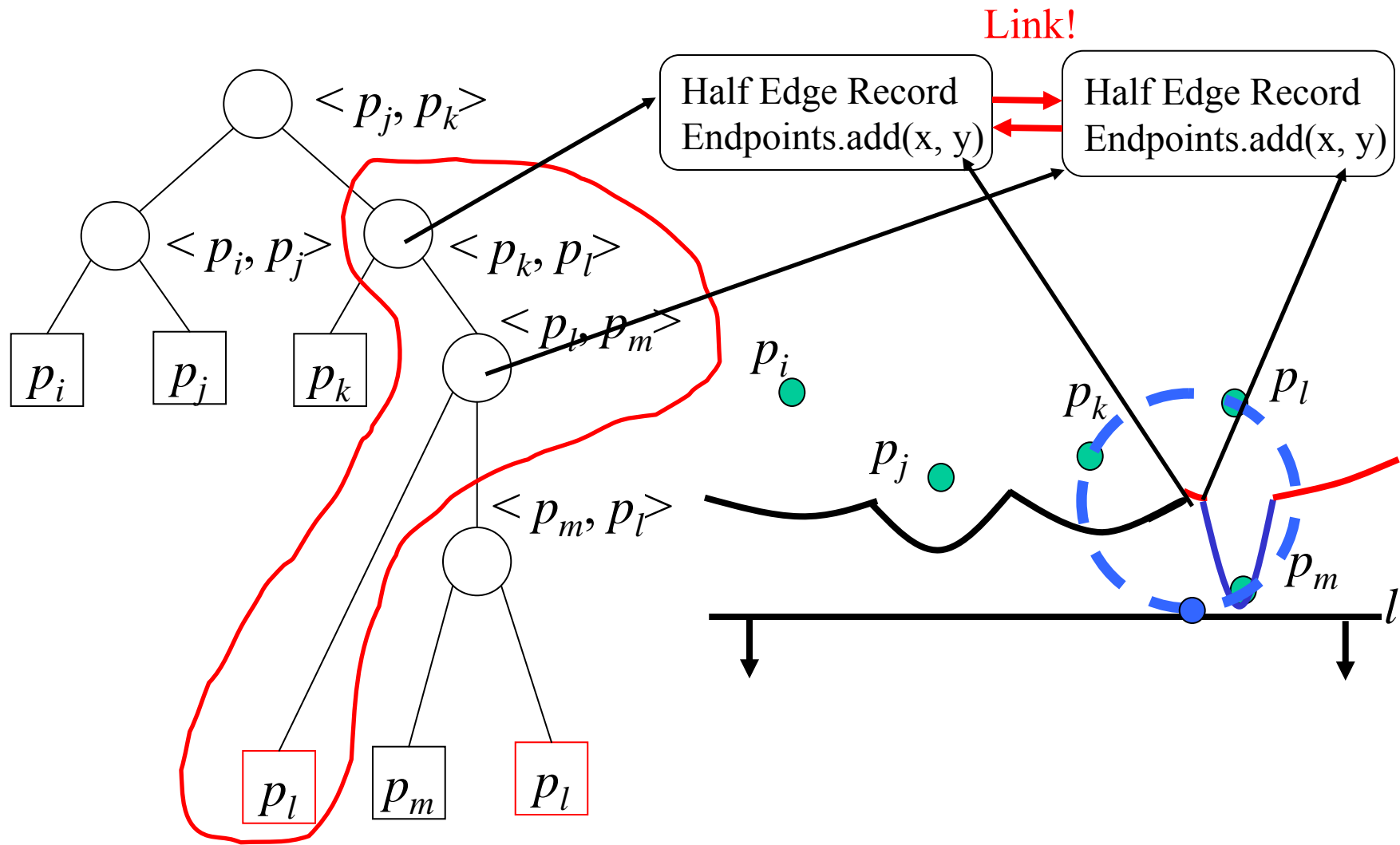
- Map input numbers to a position on the number line. The resulting Voronoi diagram is doubly linked list that forms a chain of unbounded cells in the left-to-right (sorted) order.

# Remaining slides

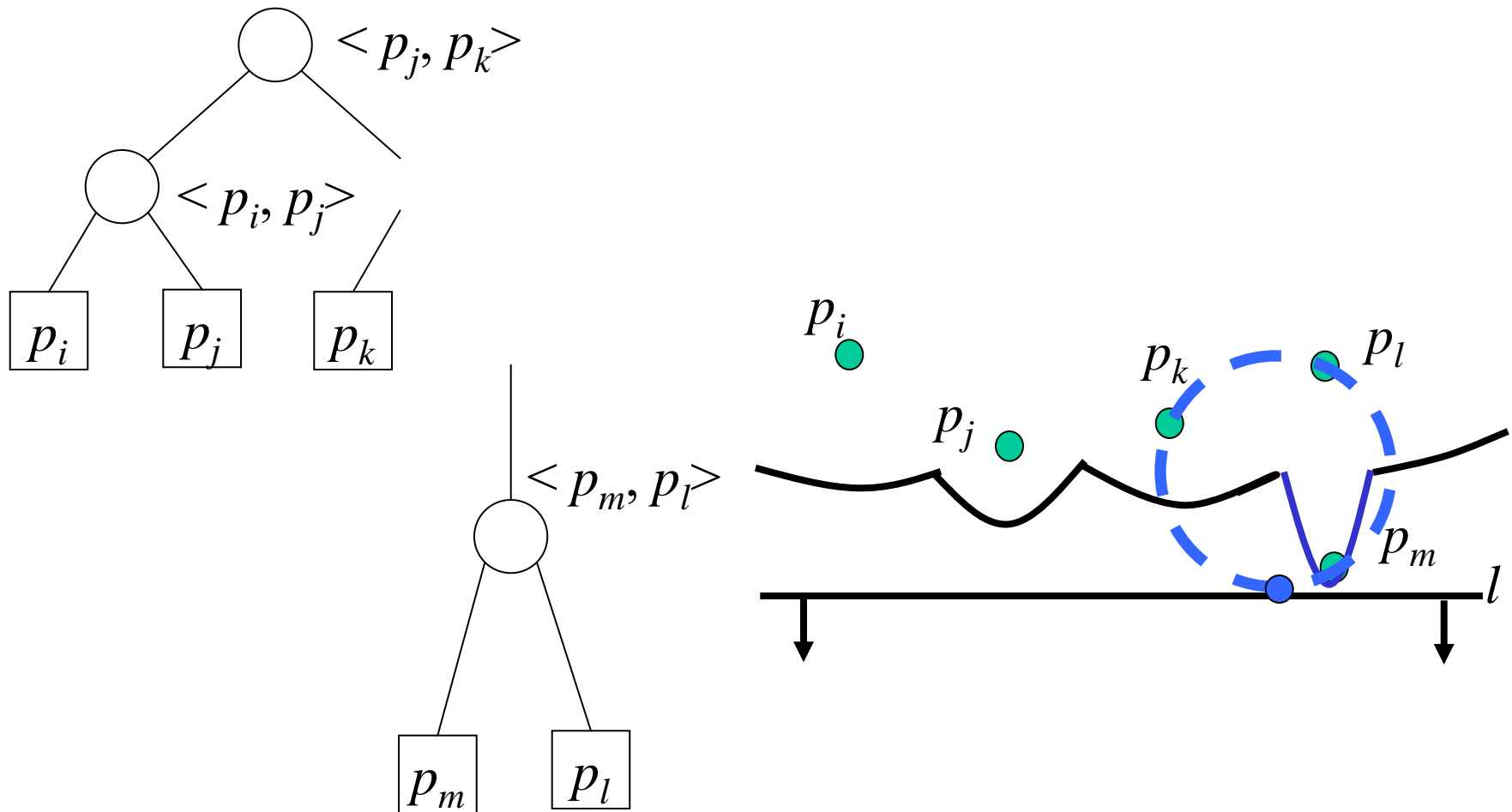
# A Circle Event



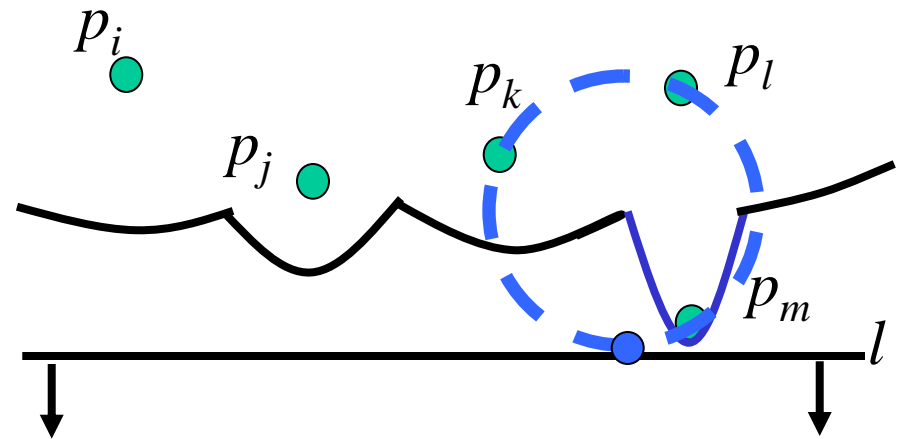
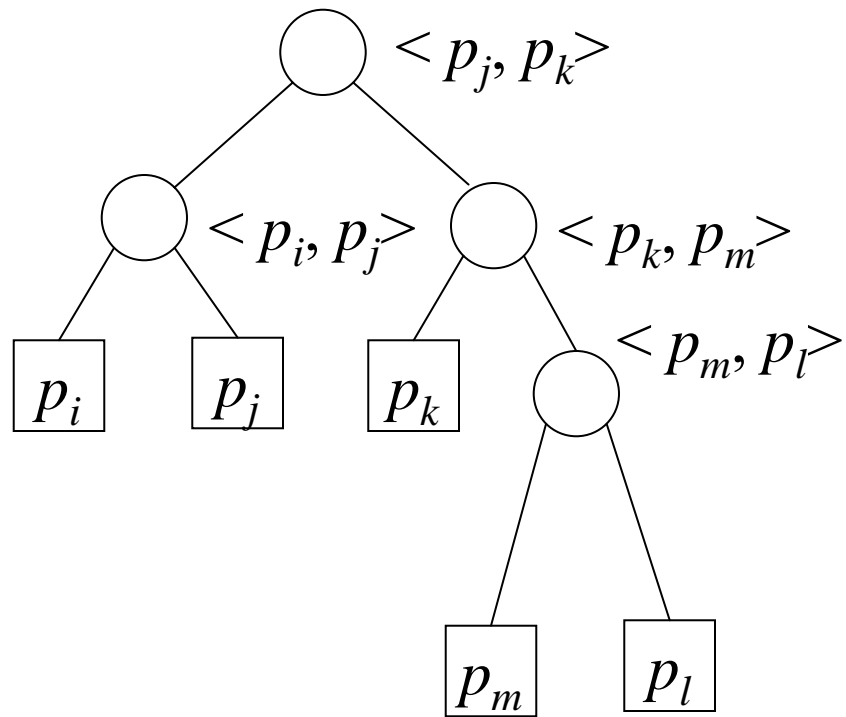
# Add vertex to corresponding edge record



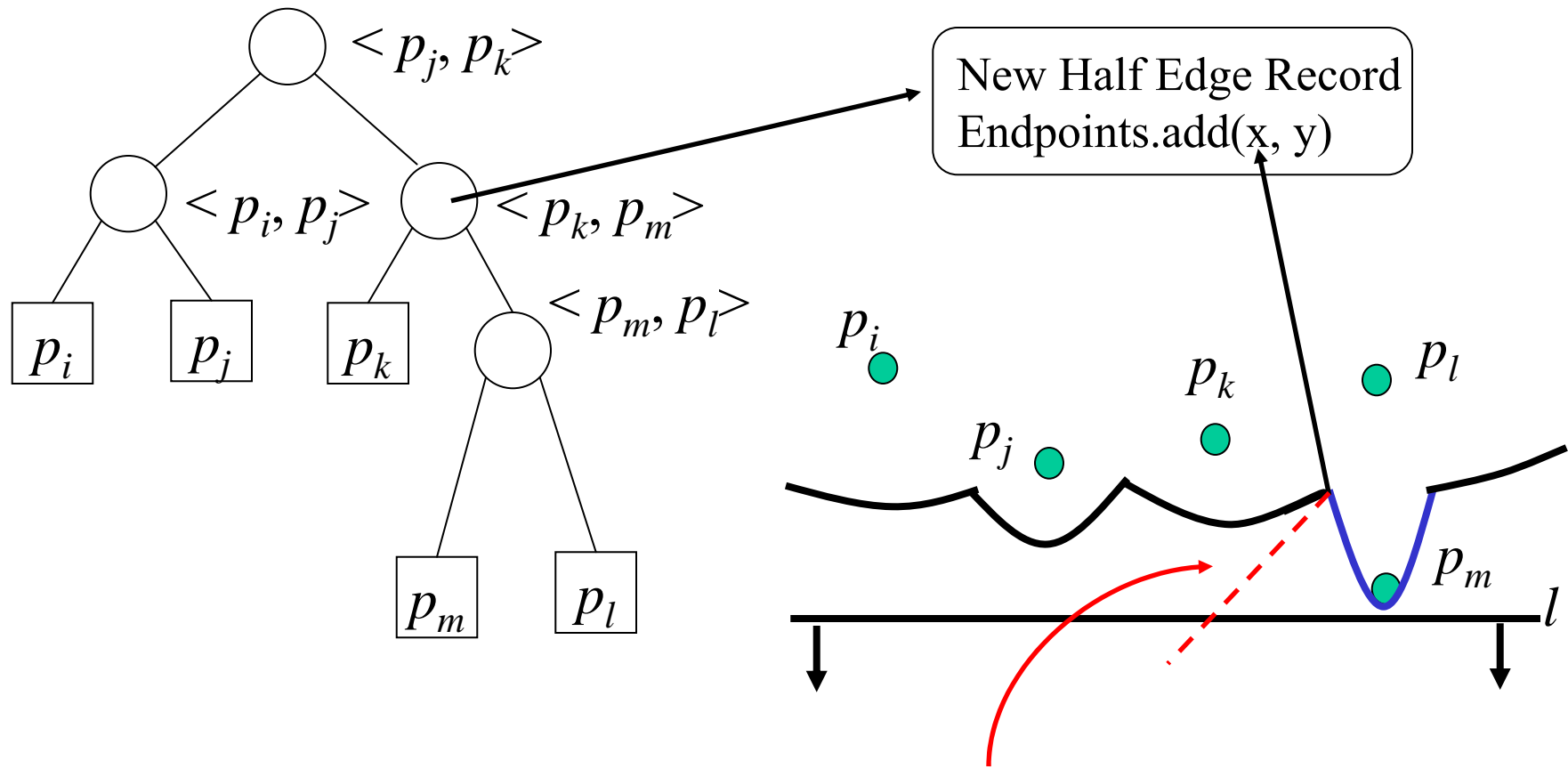
# Deleting disappearing arc



# Deleting disappearing arc



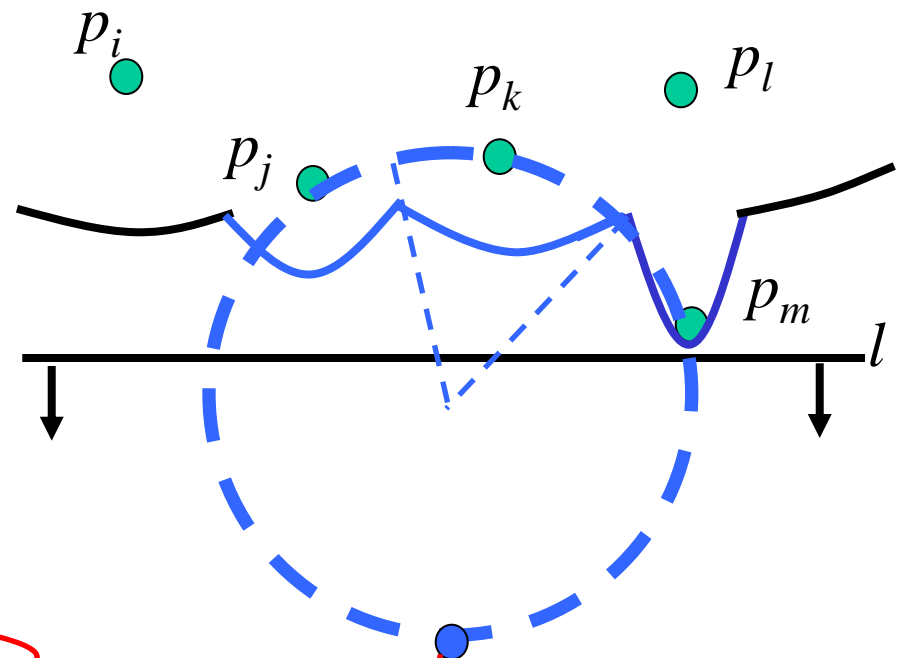
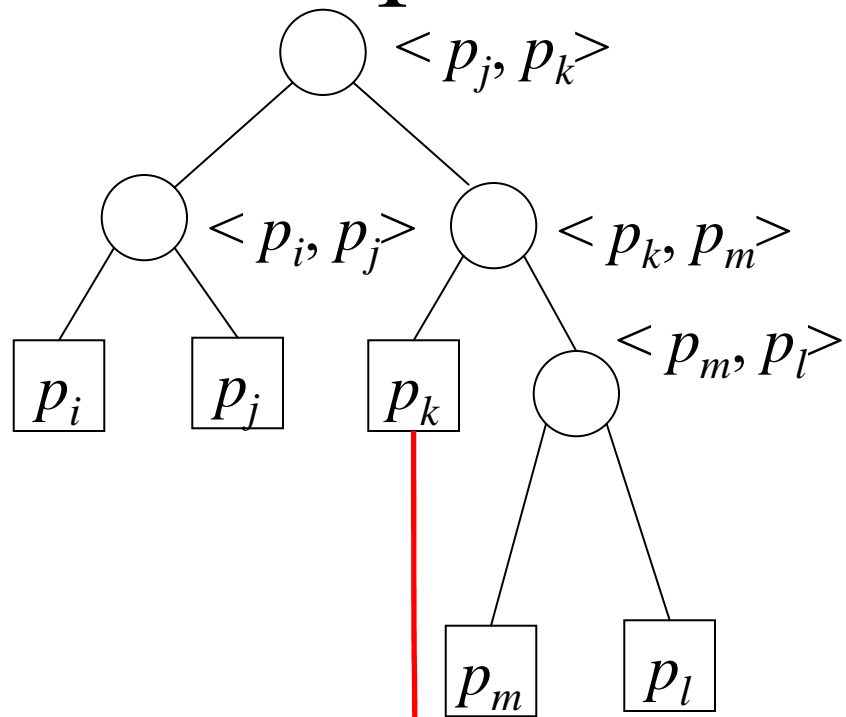
# Create new edge record



A new edge is traced out by the new break point  $\langle p_k, p_m \rangle$



# Check the new triplets for potential circle events

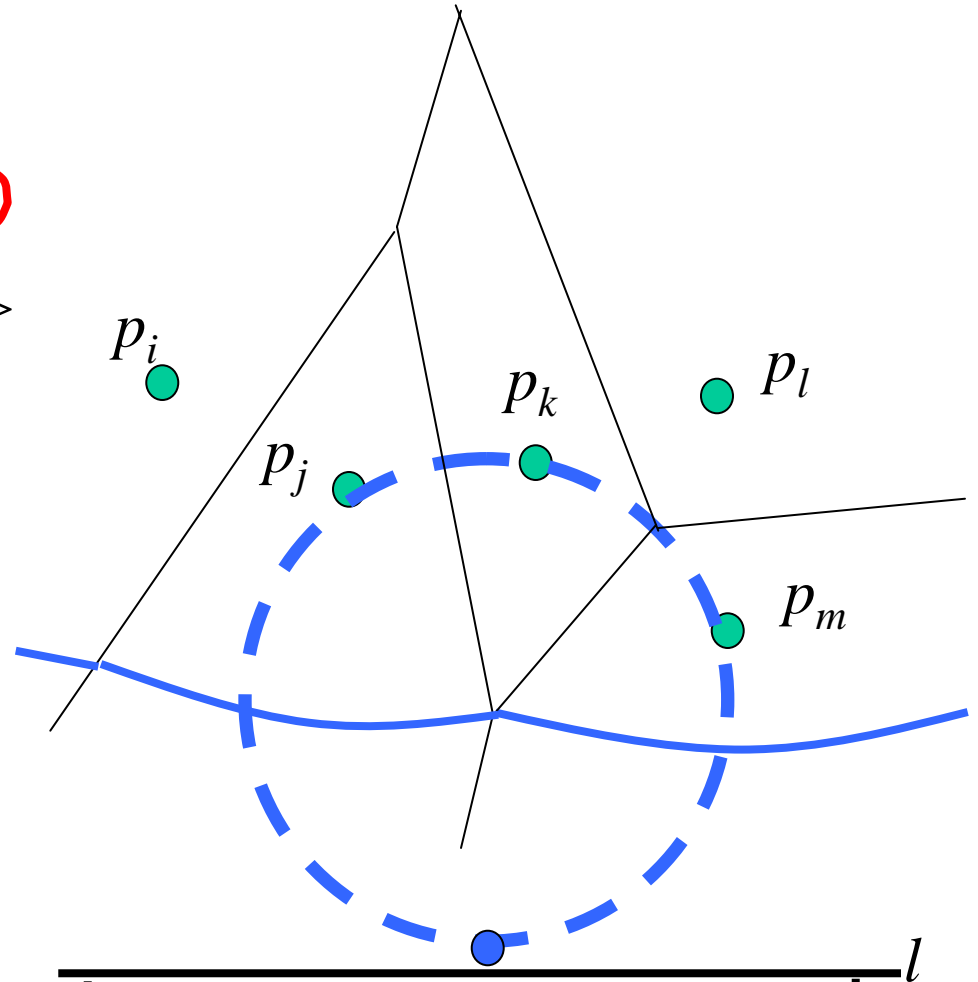
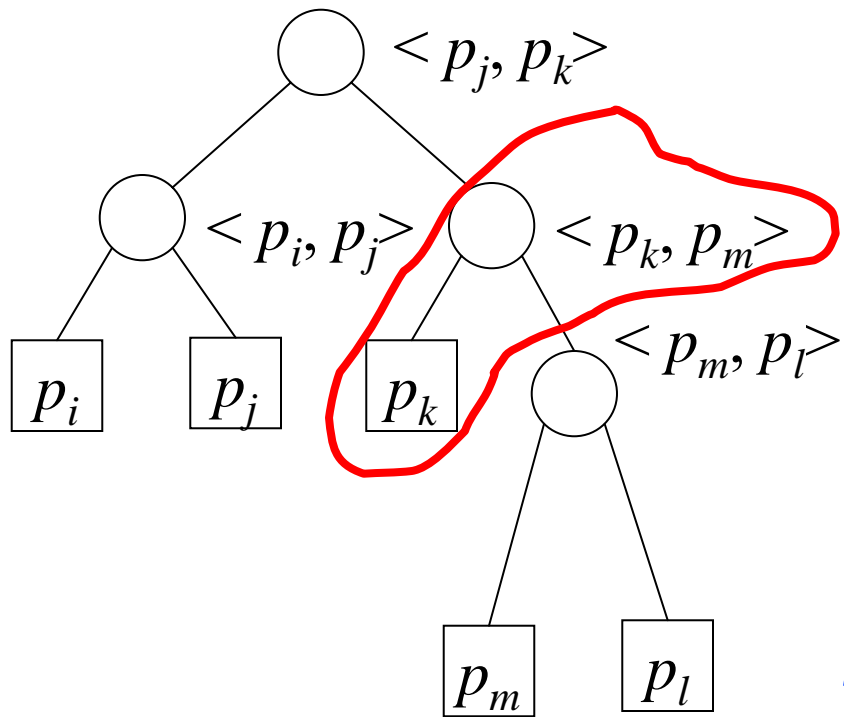


*new circle event*

# 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

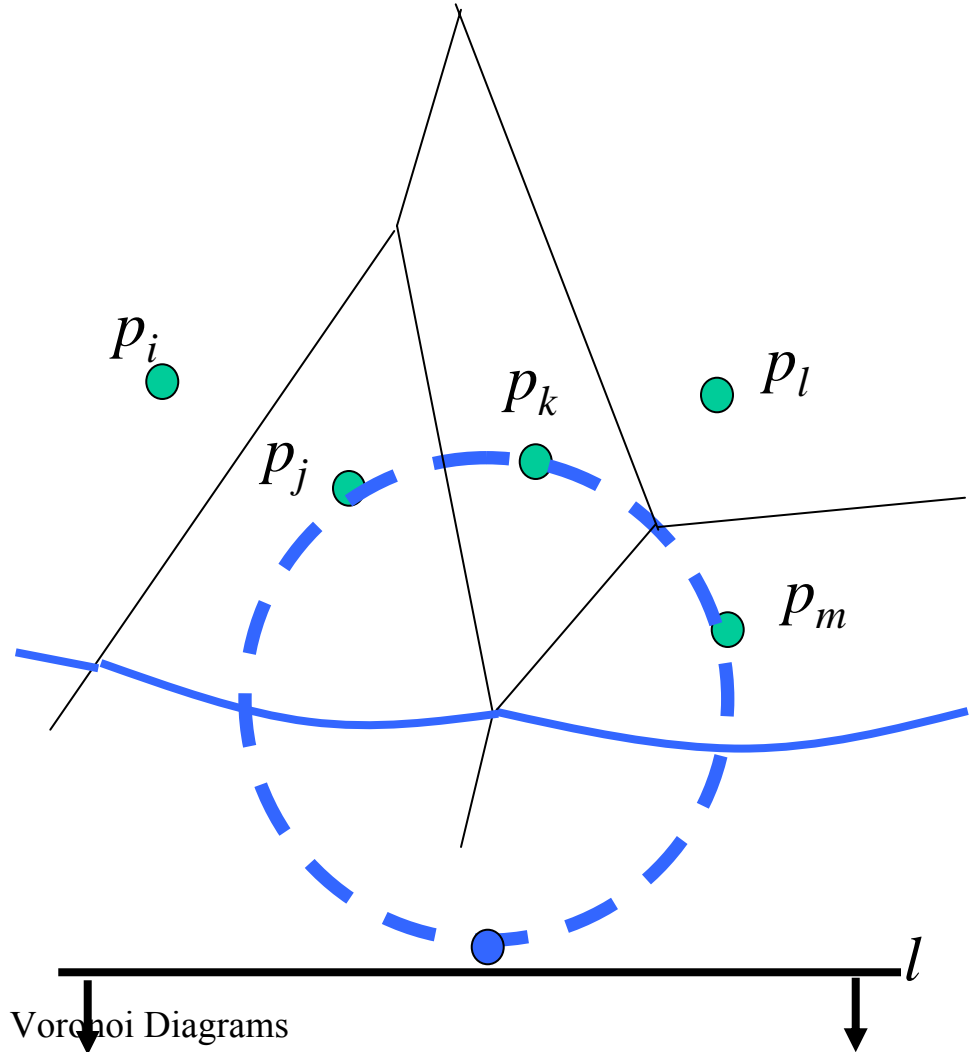
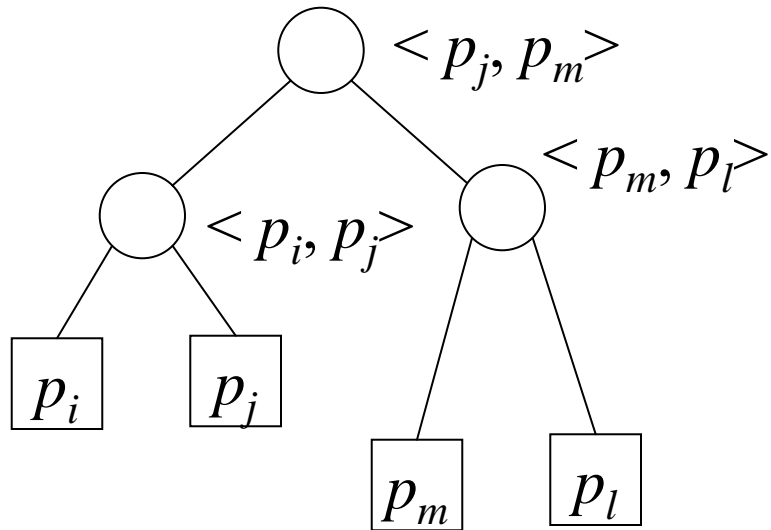


$Q$   $\emptyset$

March 1, 2005

Lecture 8: Voronoi Diagrams

# Algorithm Termination

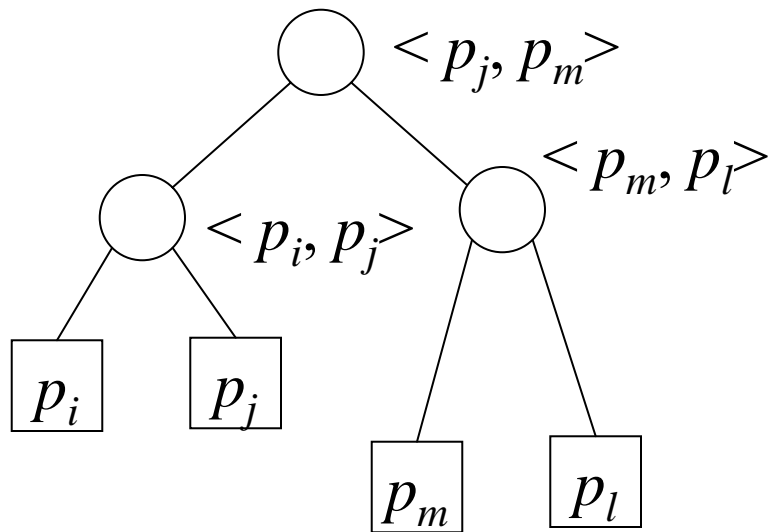


$Q$   $\boxed{\emptyset}$

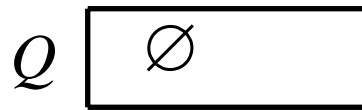
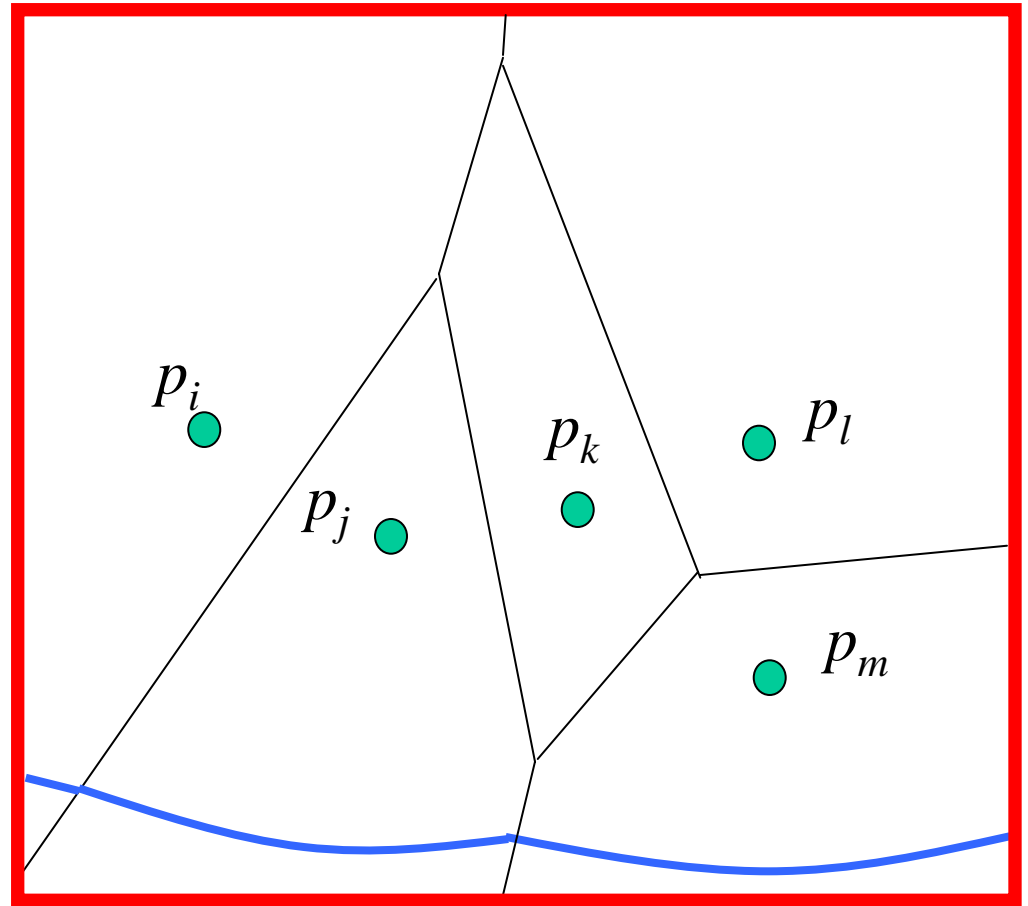
March 1, 2005

Lecture 8: Voronoi Diagrams

# Algorithm Termination



Terminate half-lines  
with a bounding box!



March 1, 2005

Lecture 8: Voronoi Diagrams

$l$

# 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**

# 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

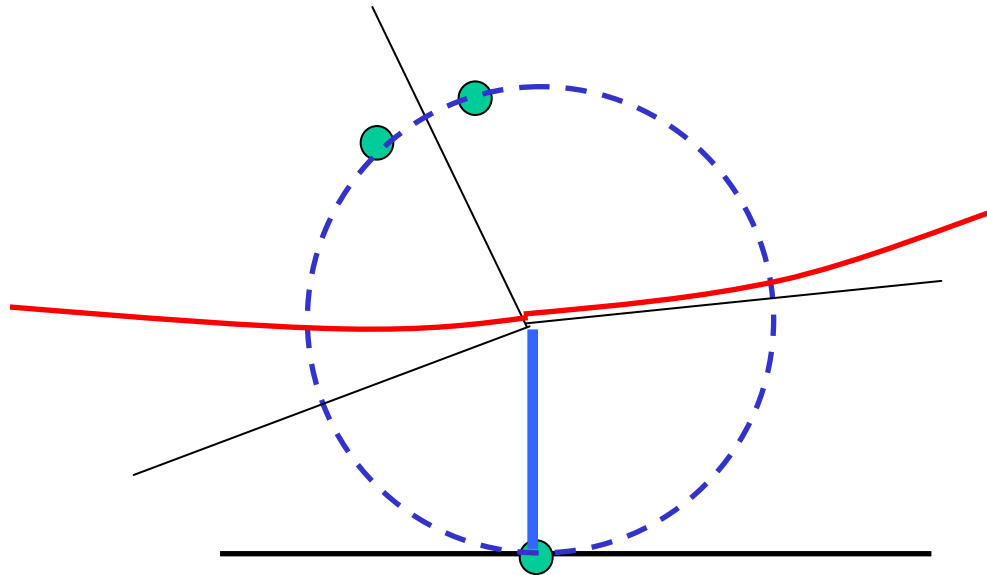
# Degenerate Cases

- 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

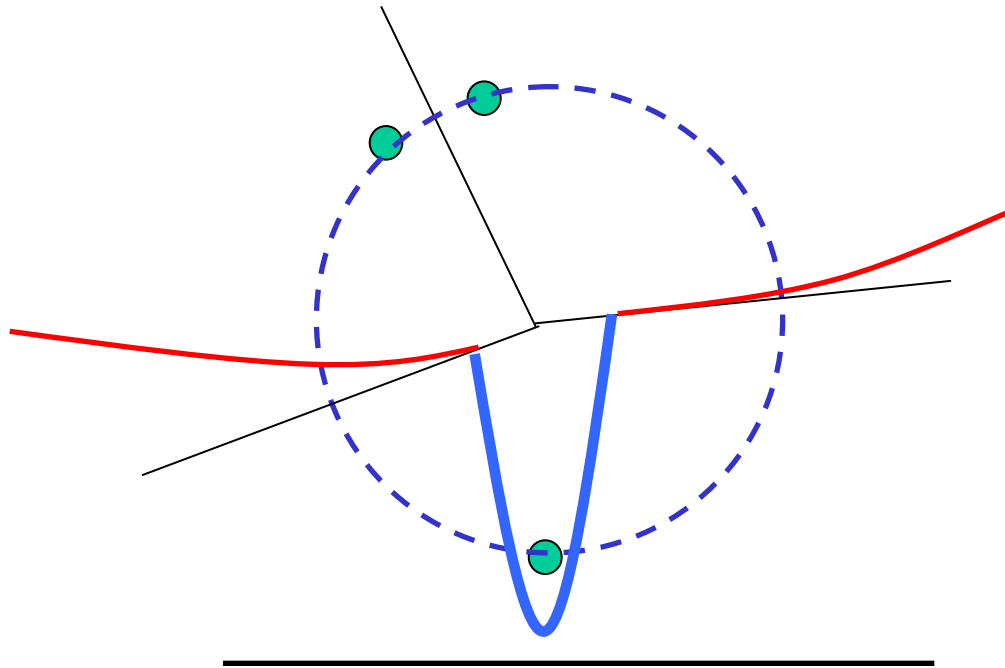


# Site coincides with circle event: the same algorithm applies!

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

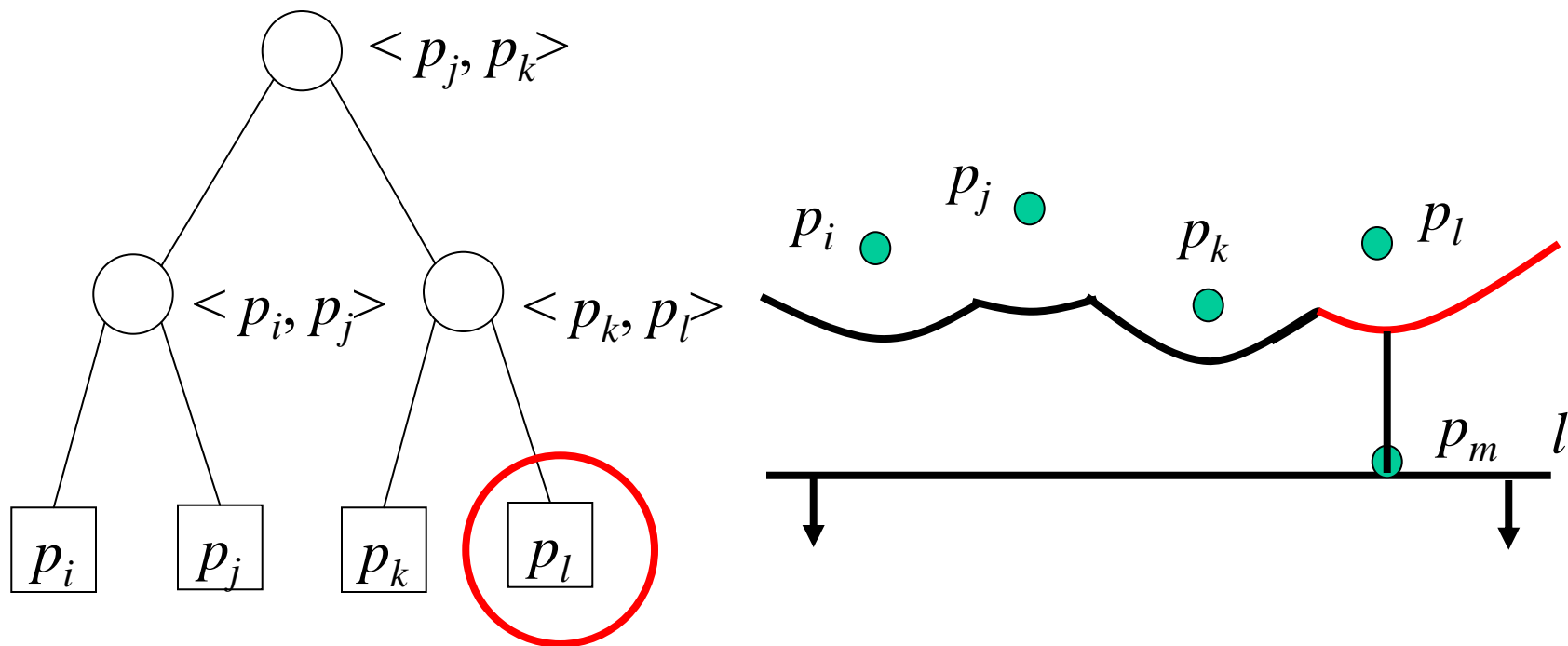


March 1, 2005

Lecture 8: Voronoi Diagrams

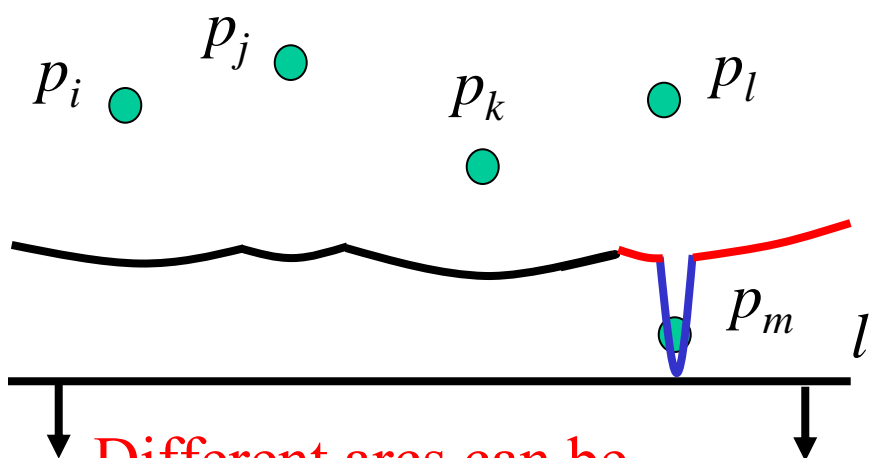
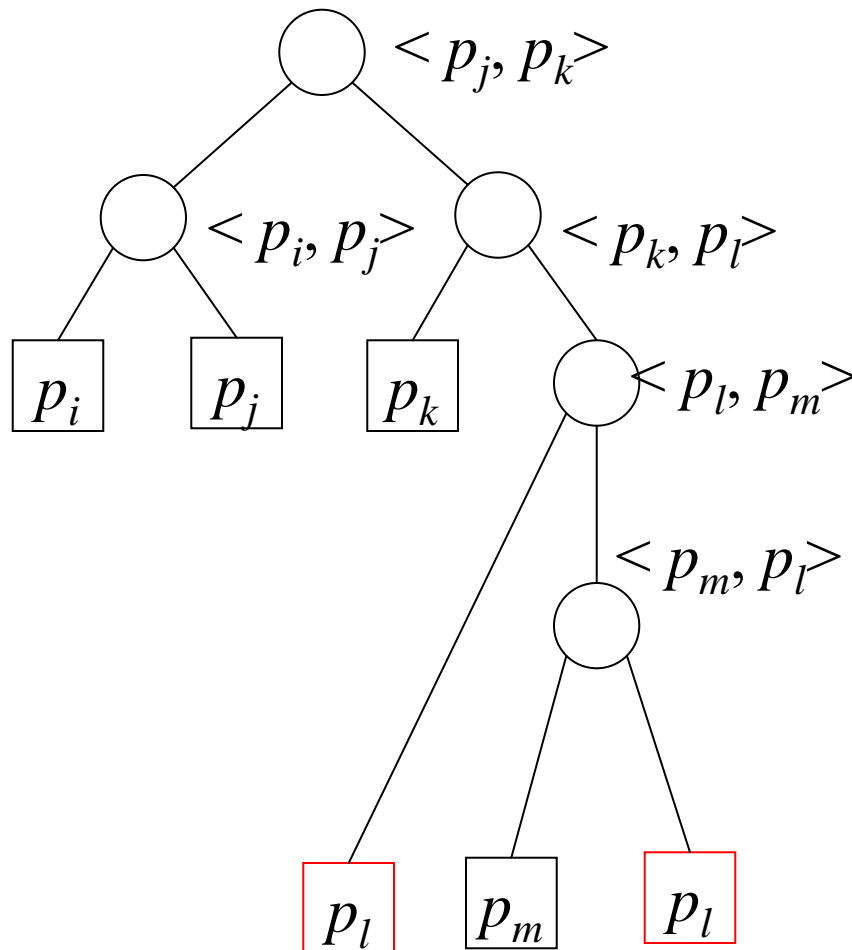
# Locate the existing arc that is above the new site

- 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



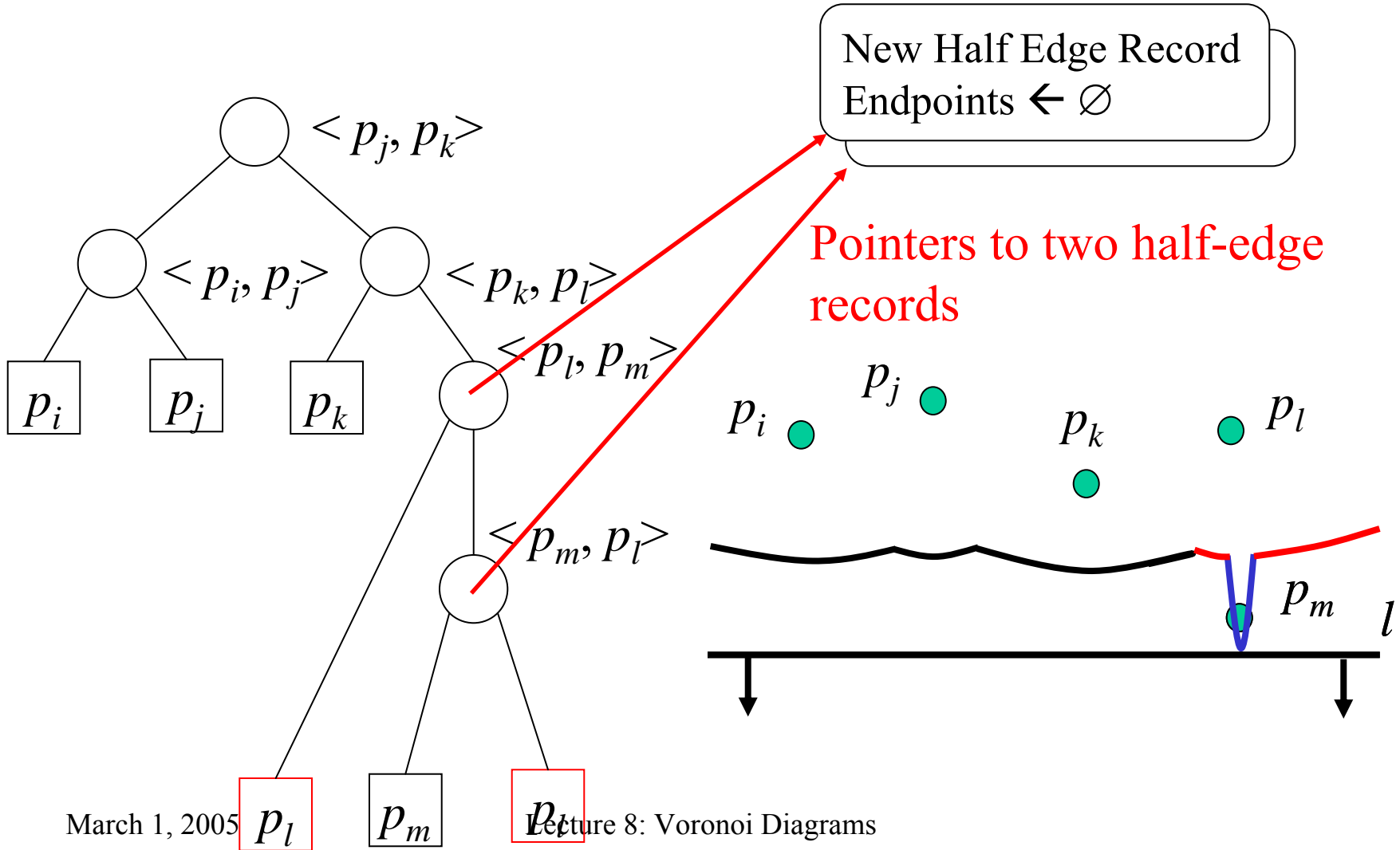
# Break the Arc

Corresponding leaf replaced by a new sub-tree



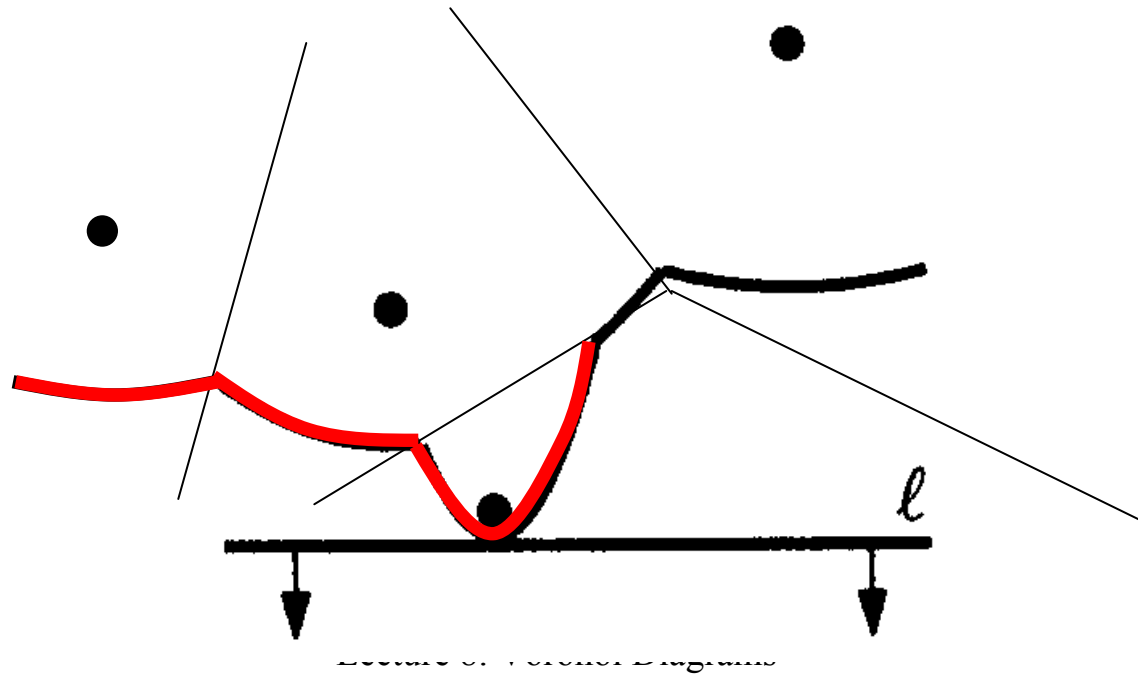
Different arcs can be induced by the same site!

# Add a new edge record in the doubly linked list



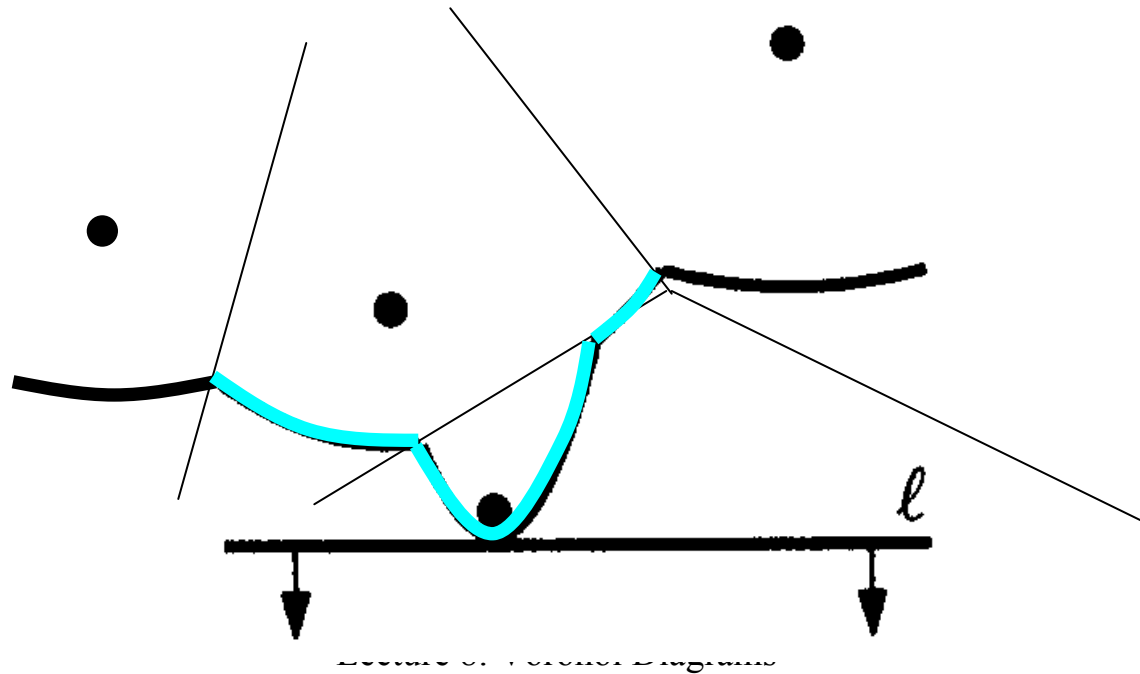
# Checking for Potential Circle Events

- 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



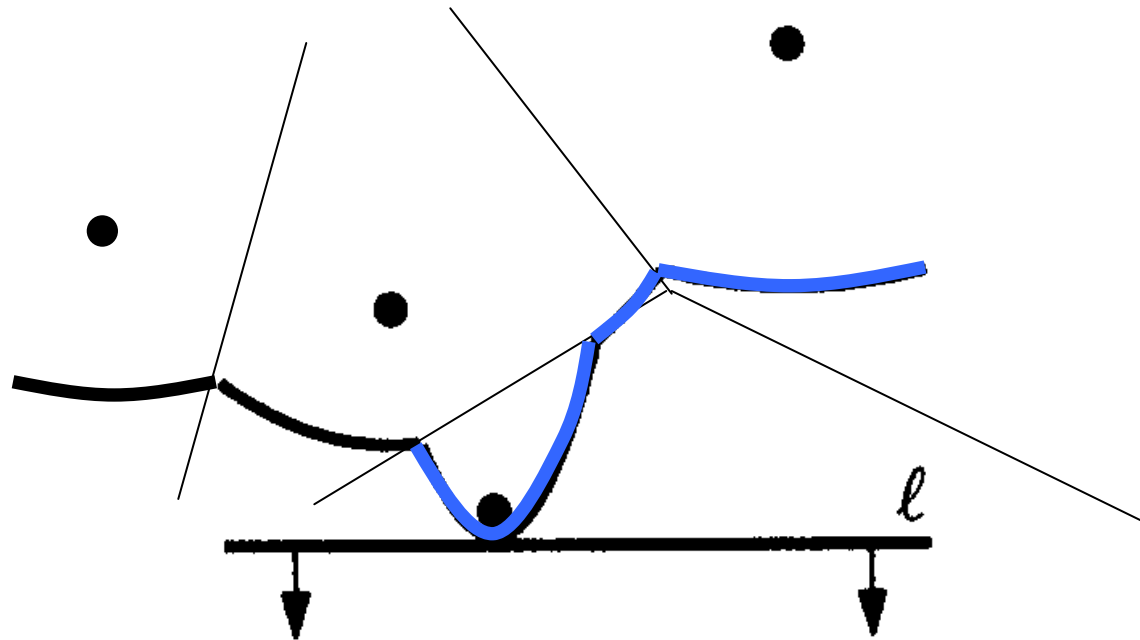
# Checking for Potential Circle Events

- 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

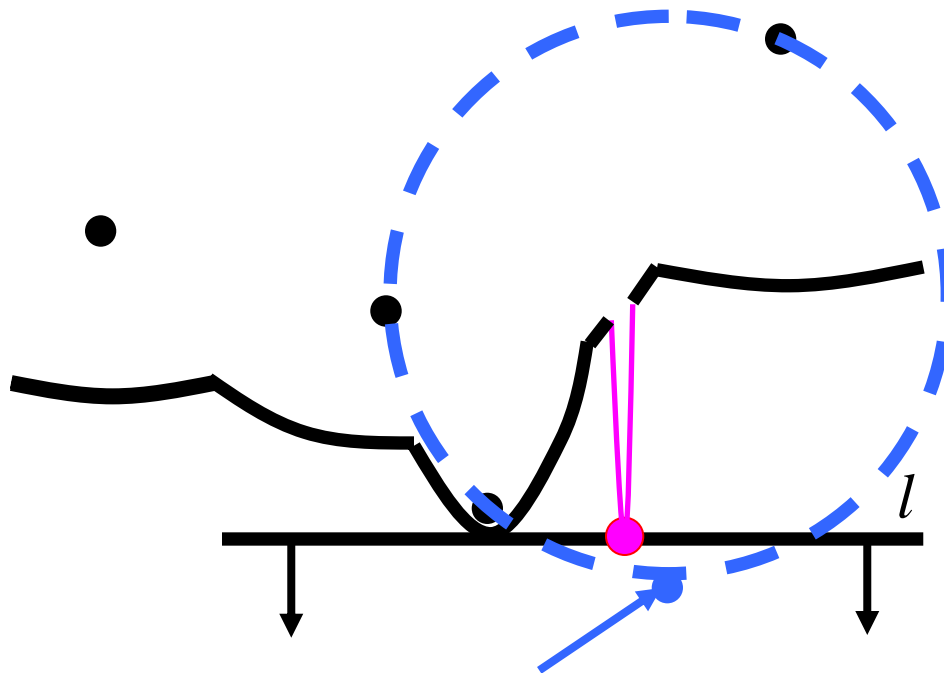


March 1, 2005



# 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



March 1, 2005 (The original circle event becomes a *false alarm*)  
© 2005 by Eric W. Steinbock

# 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