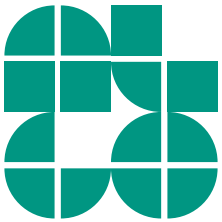
Computational Geometry Lecture

Voronoi Diagrams and Delaunay Triangulations

INSTITUT FÜR THEORETISCHE INFORMATIK · FAKULTÄT FÜR INFORMATIK

Chih-Hung Liu · Tamara Mchlededze

30.05.2018 & 06.06.2018



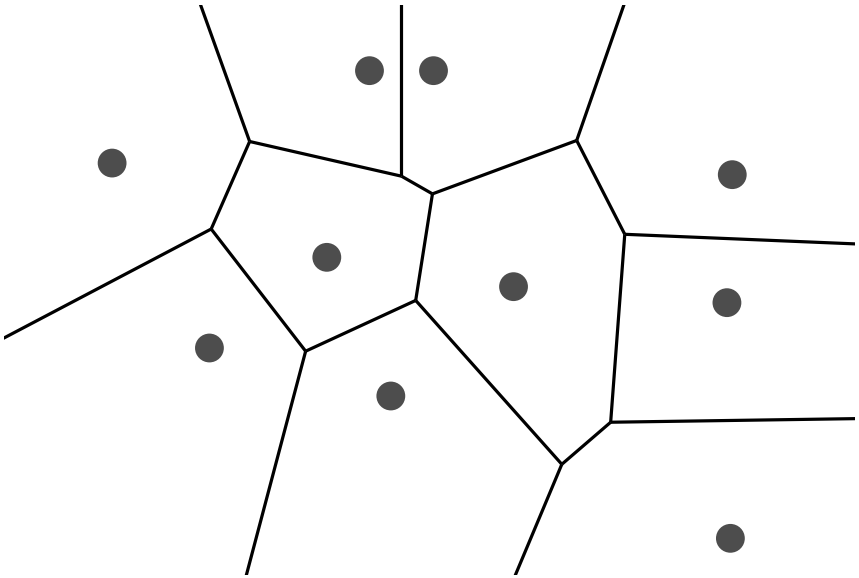
Two Mathematicians

Georgy Voronoi
(1868–1908)

Boris Delone
(1890–1980)

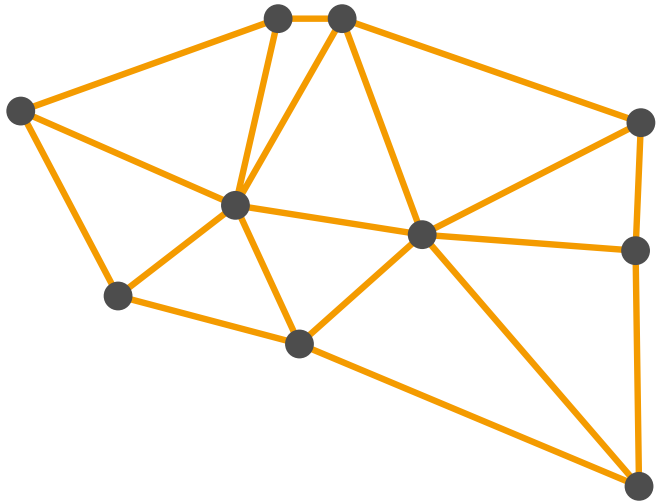
Two Mathematicians

Voronoi Diagram



Georgy Voronoi
(1868–1908)

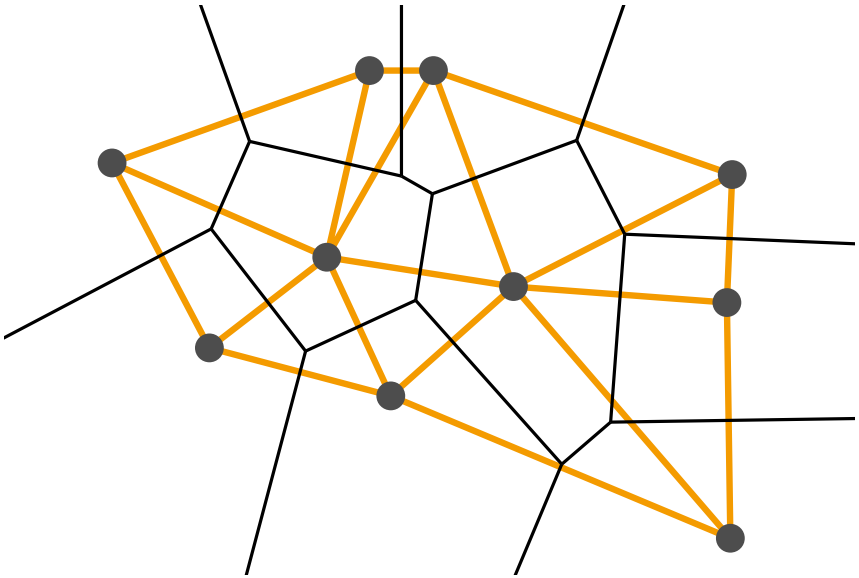
Delaunay Triangulation



Boris Delone
(1890–1980)

Two Mathematicians

Voronoi Diagram $\Leftrightarrow$ Delaunay Triangulation



Georgy Voronoi
(1868–1908)

Boris Delone
(1890–1980)

Outline

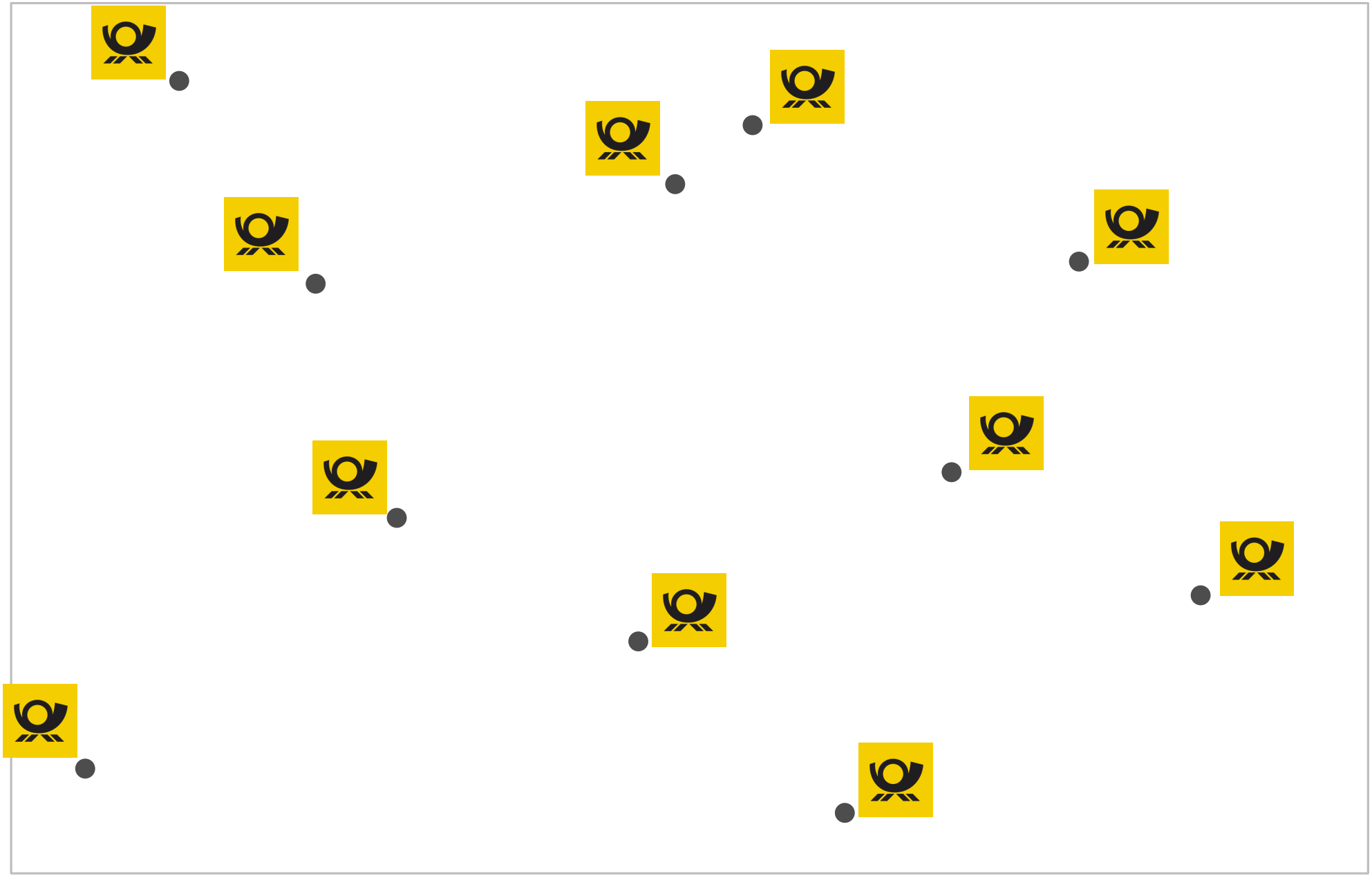
1. Voronoi Diagrams and Delaunay Triangulations
   - Properties
   - Duality
2. Algorithms
   - Plane Sweep
   - Divide and Conquer
3. 3D Geometric Transformation

Outline

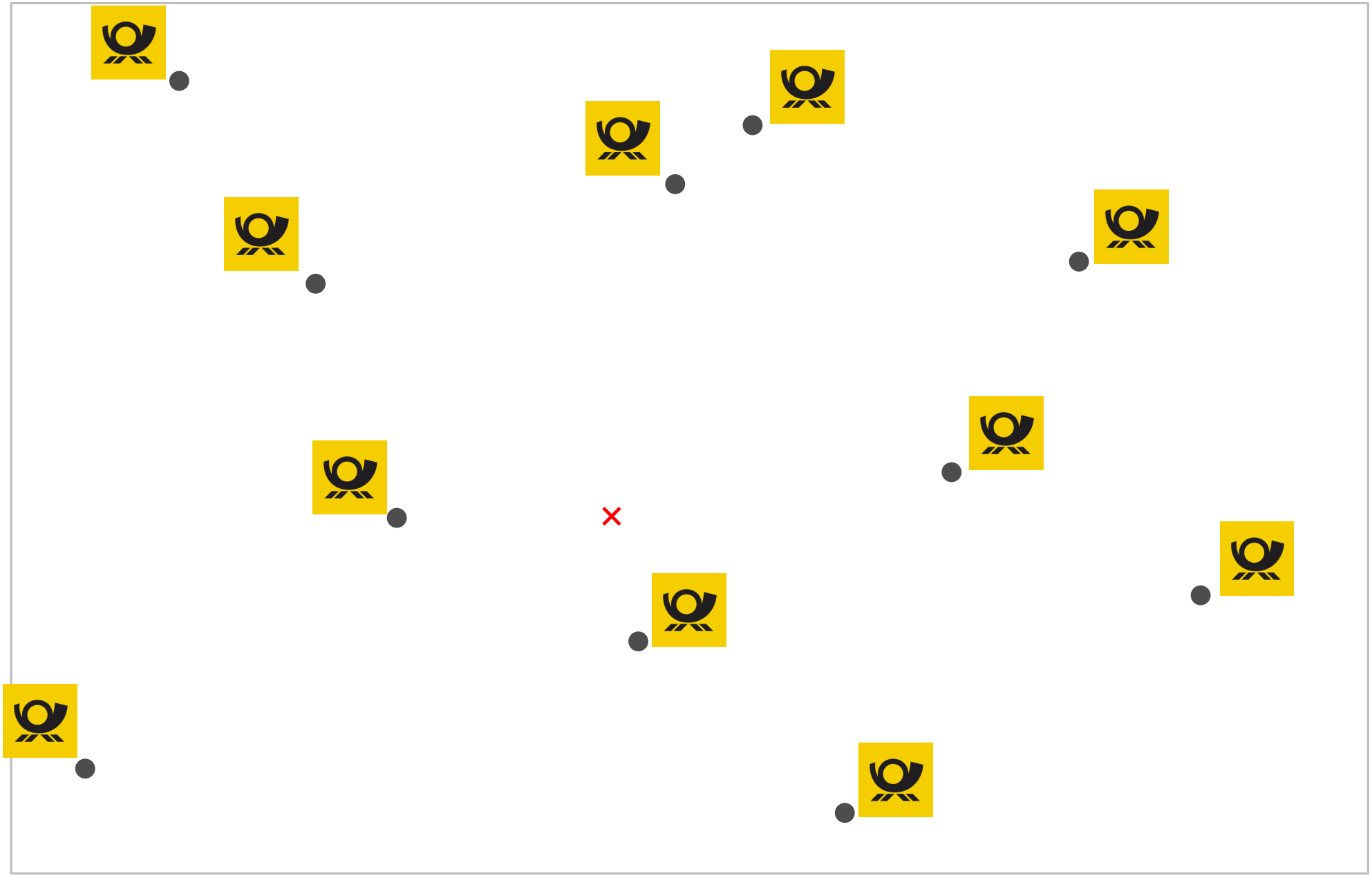
1. Voronoi Diagrams and Delaunay Triangulations
   - Properties
   - Duality
2. Algorithms
   - Plane Sweep
   - Divide and Conquer
3. 3D Geometric Transformation

P: a set of n points in the plane

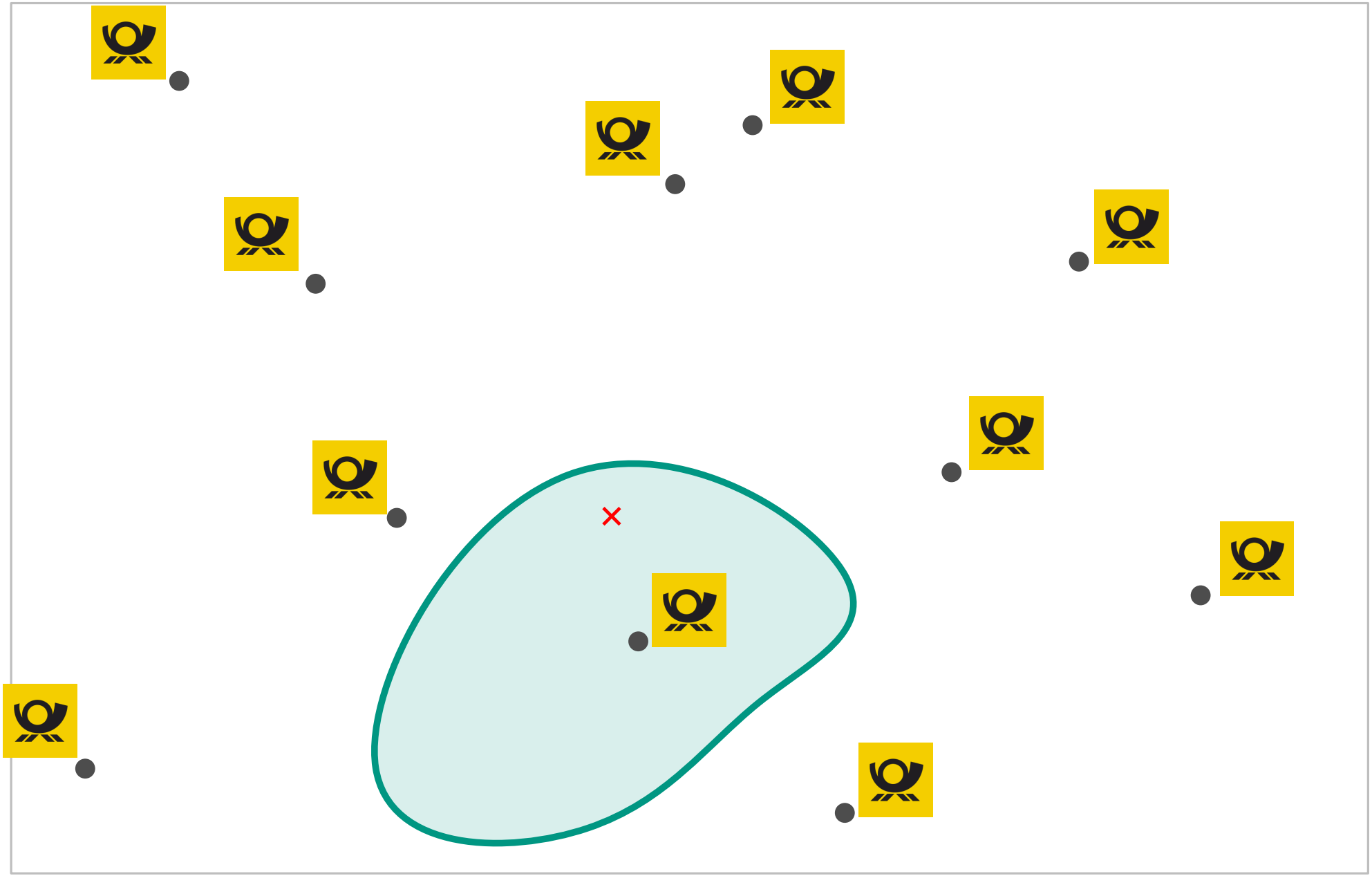
The Post Office Problem



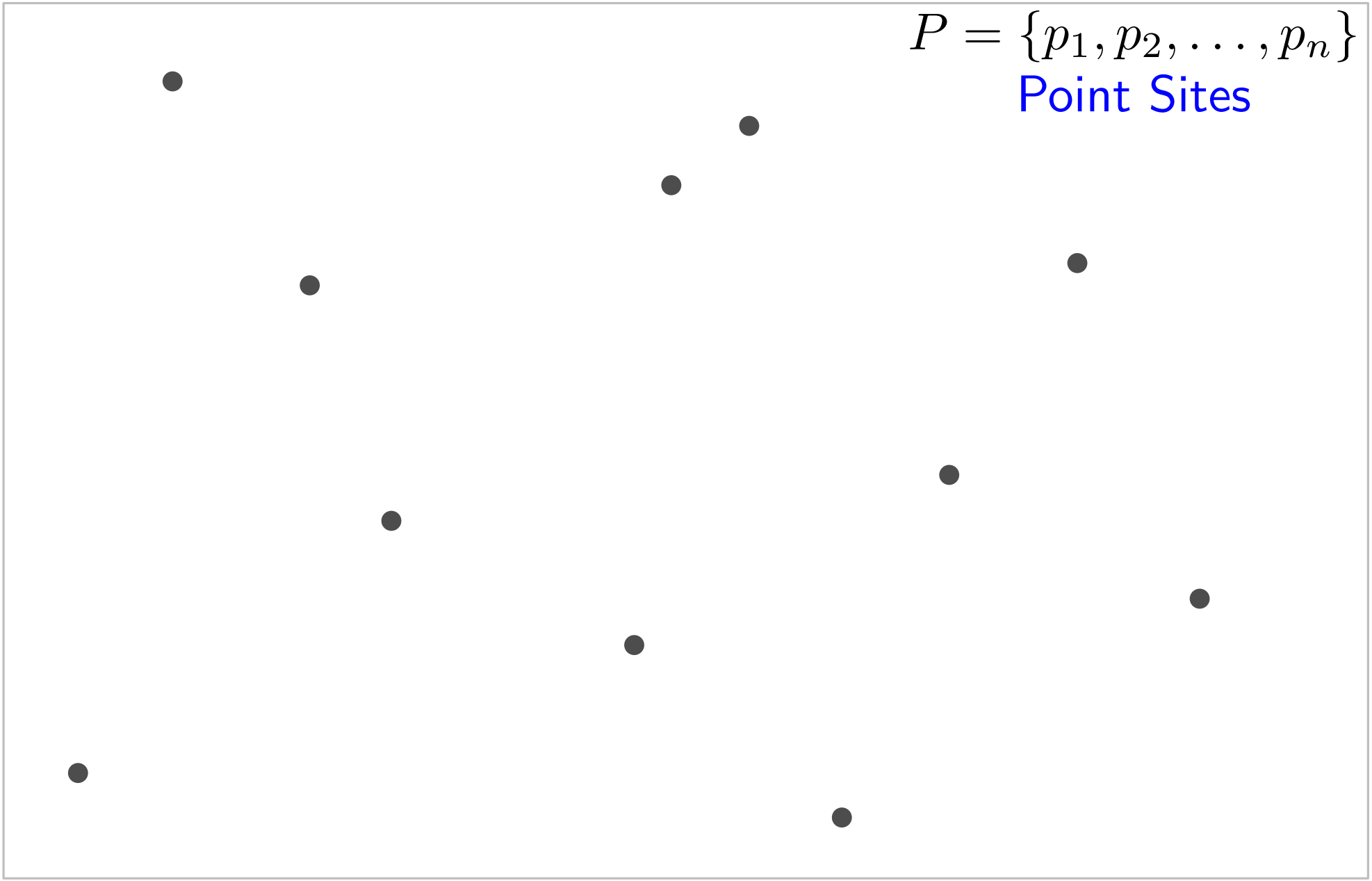
The Post Office Problem



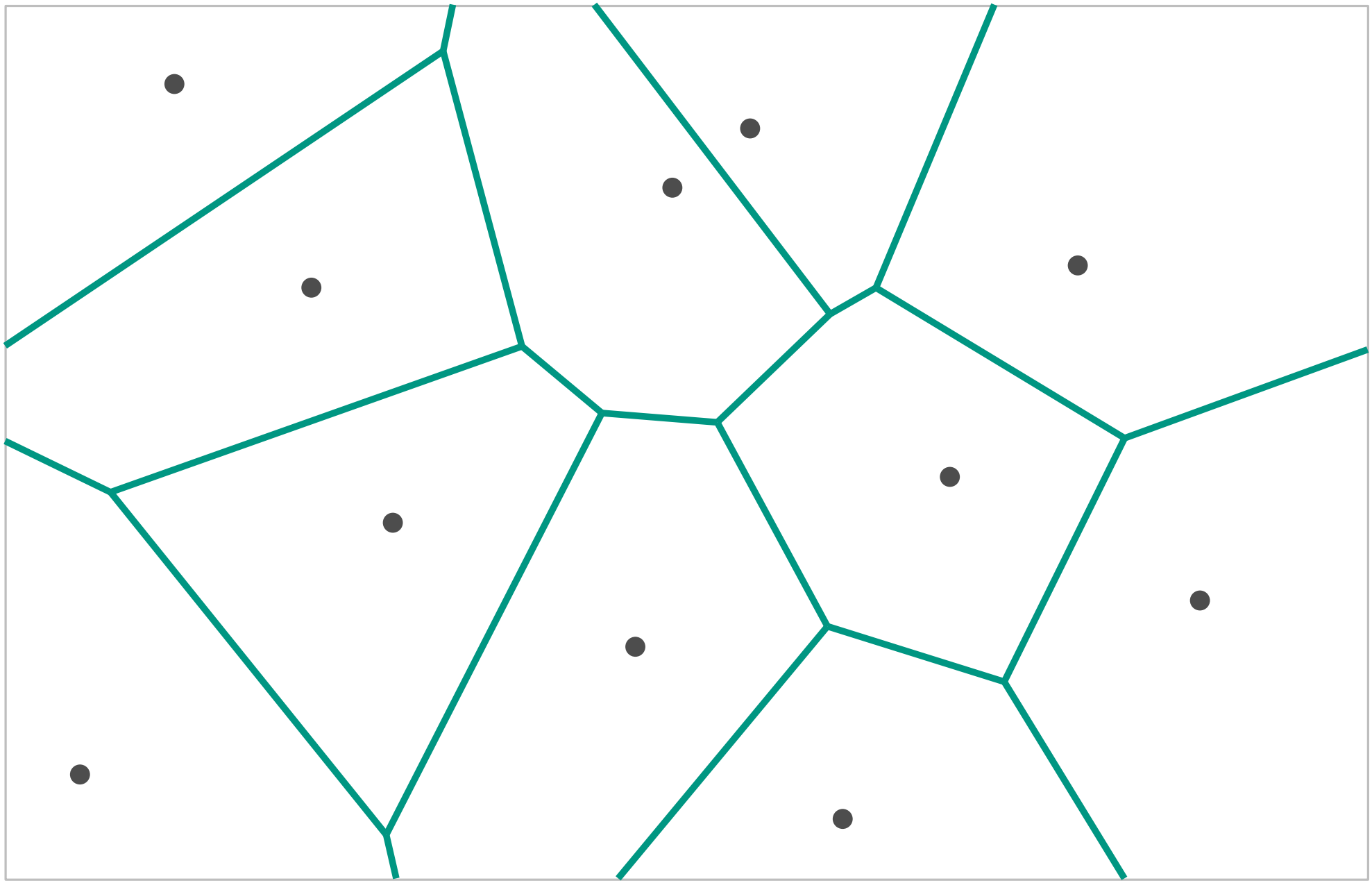
The Post Office Problem



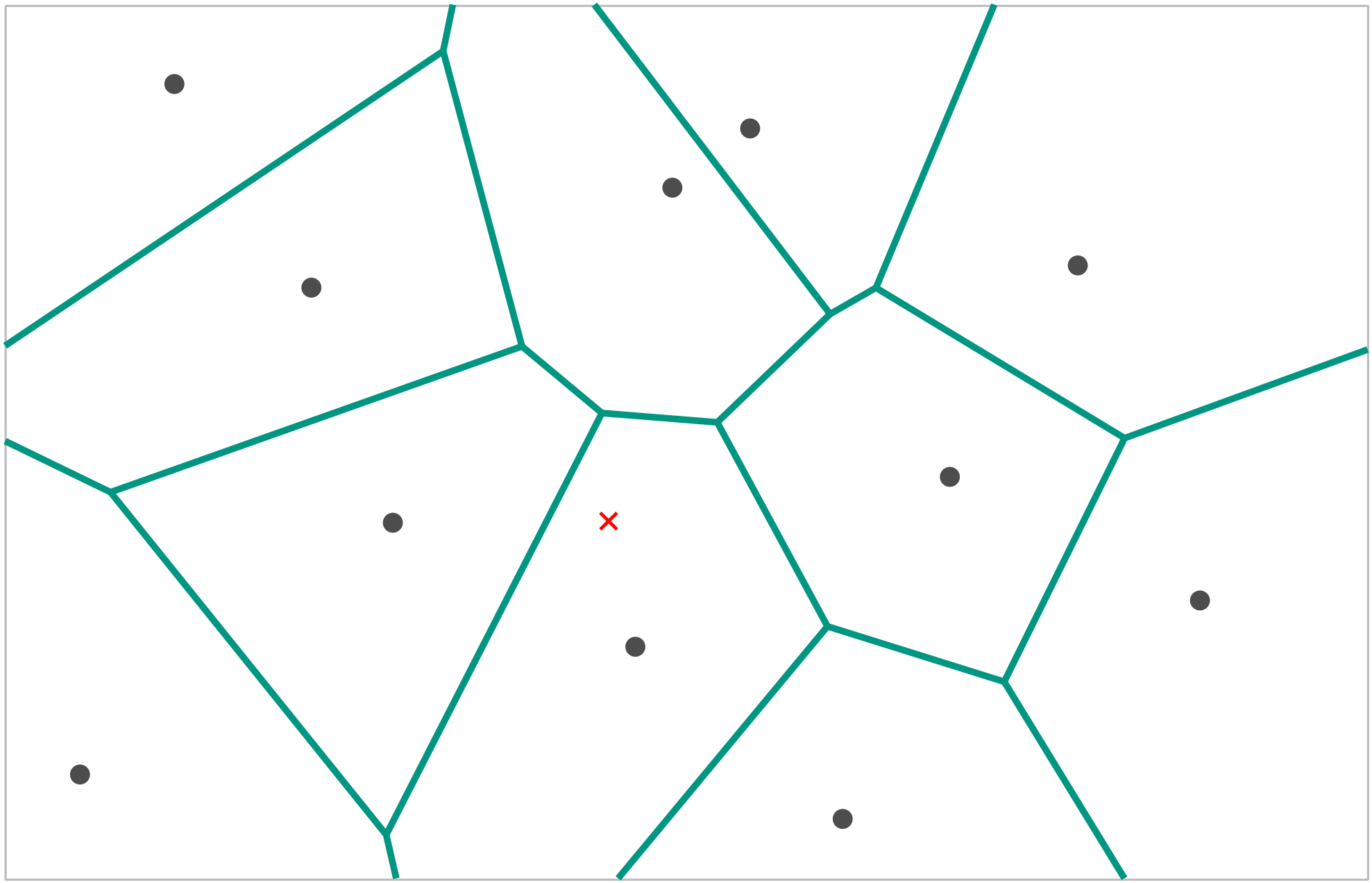
The Post Office Problem



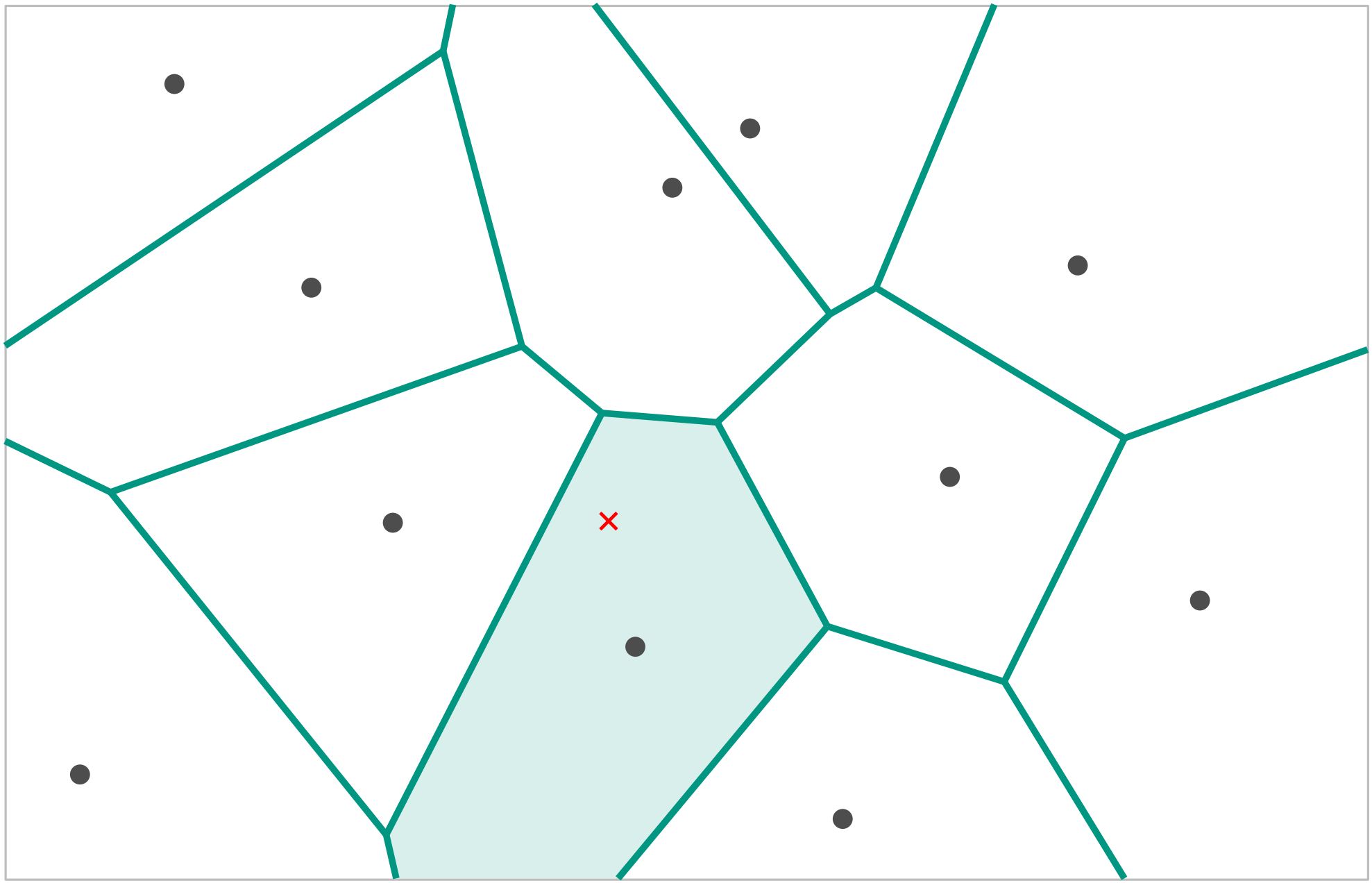
The Post Office Problem



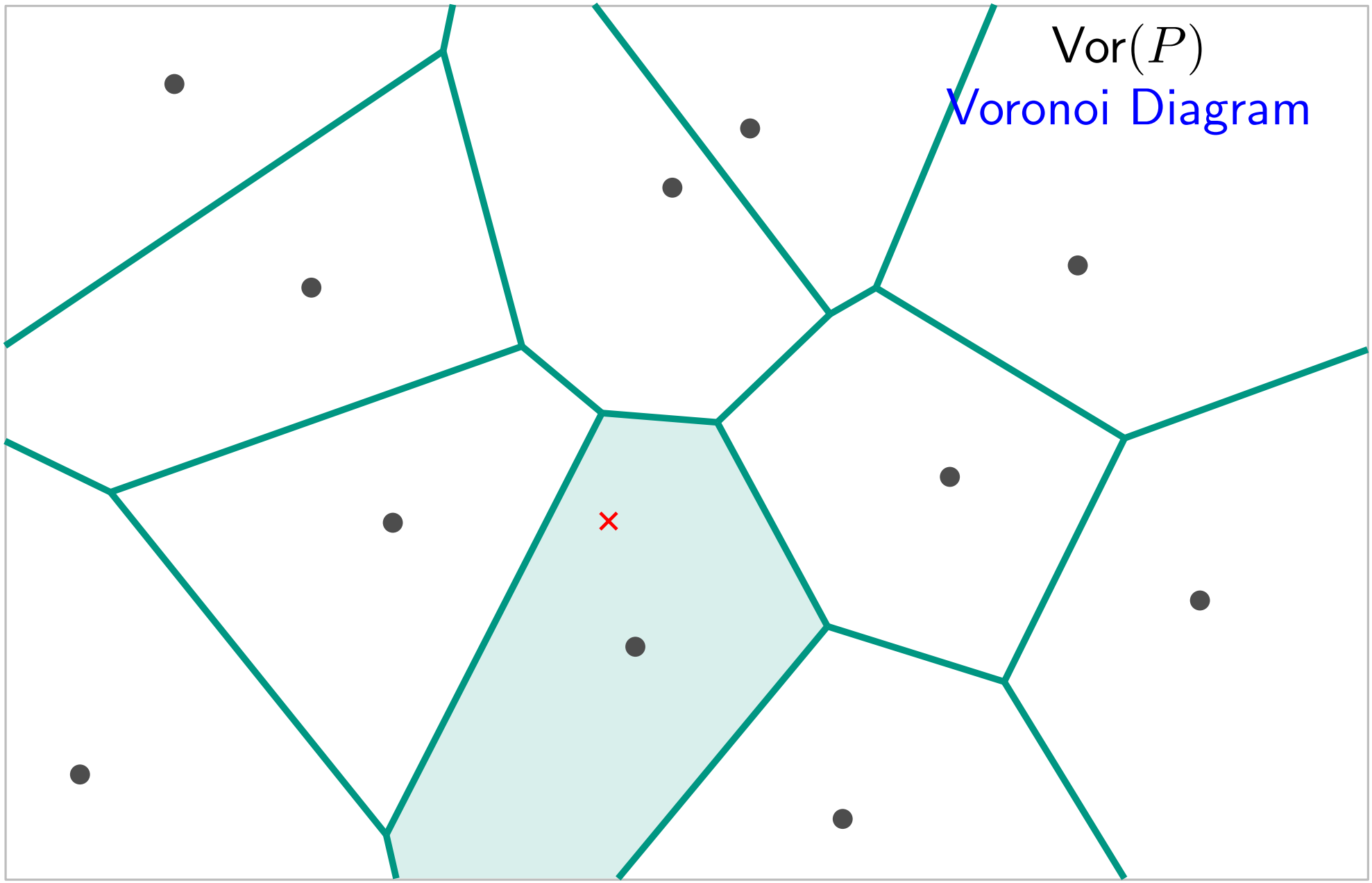
The Post Office Problem



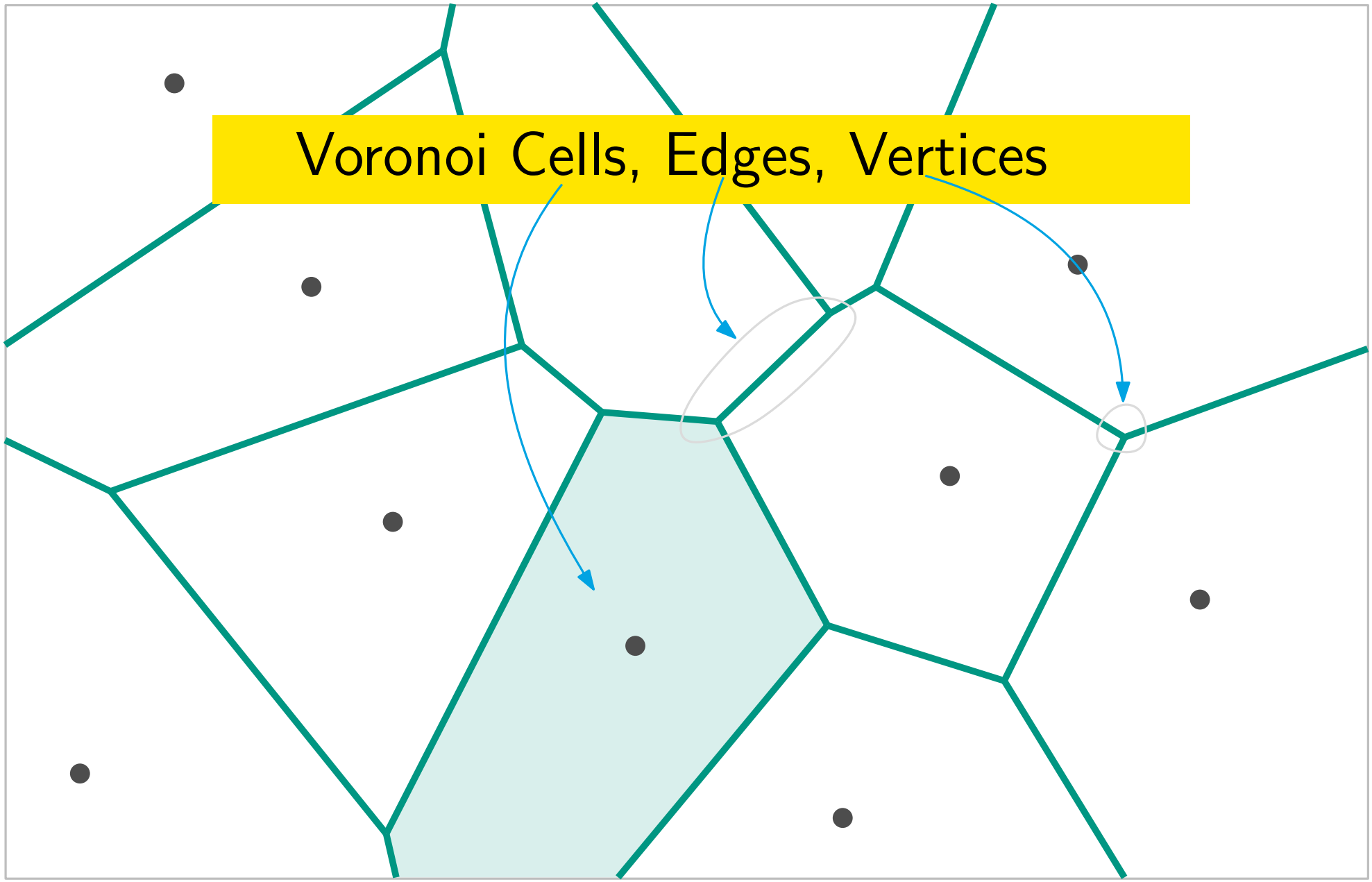
The Post Office Problem



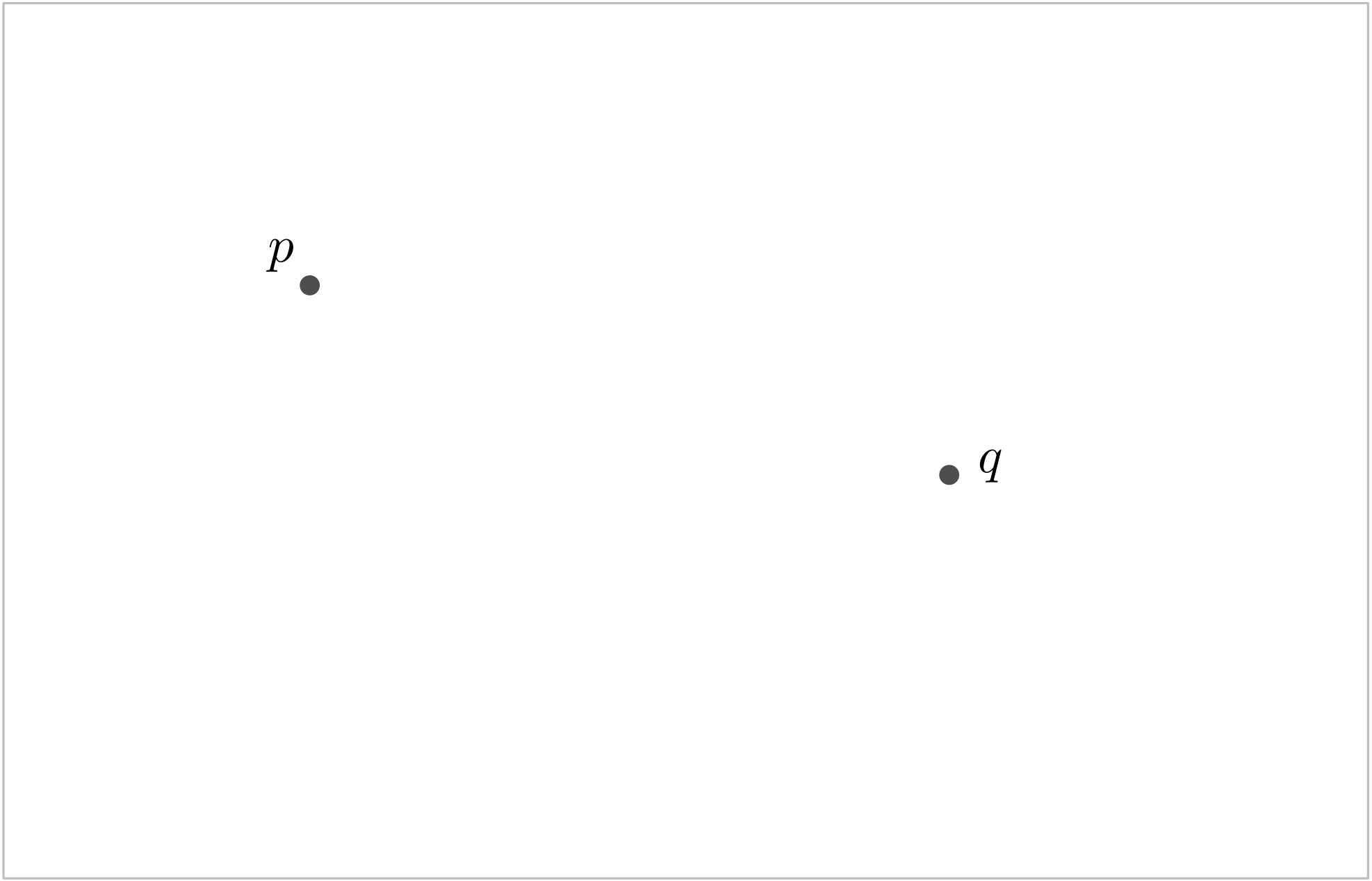
The Post Office Problem



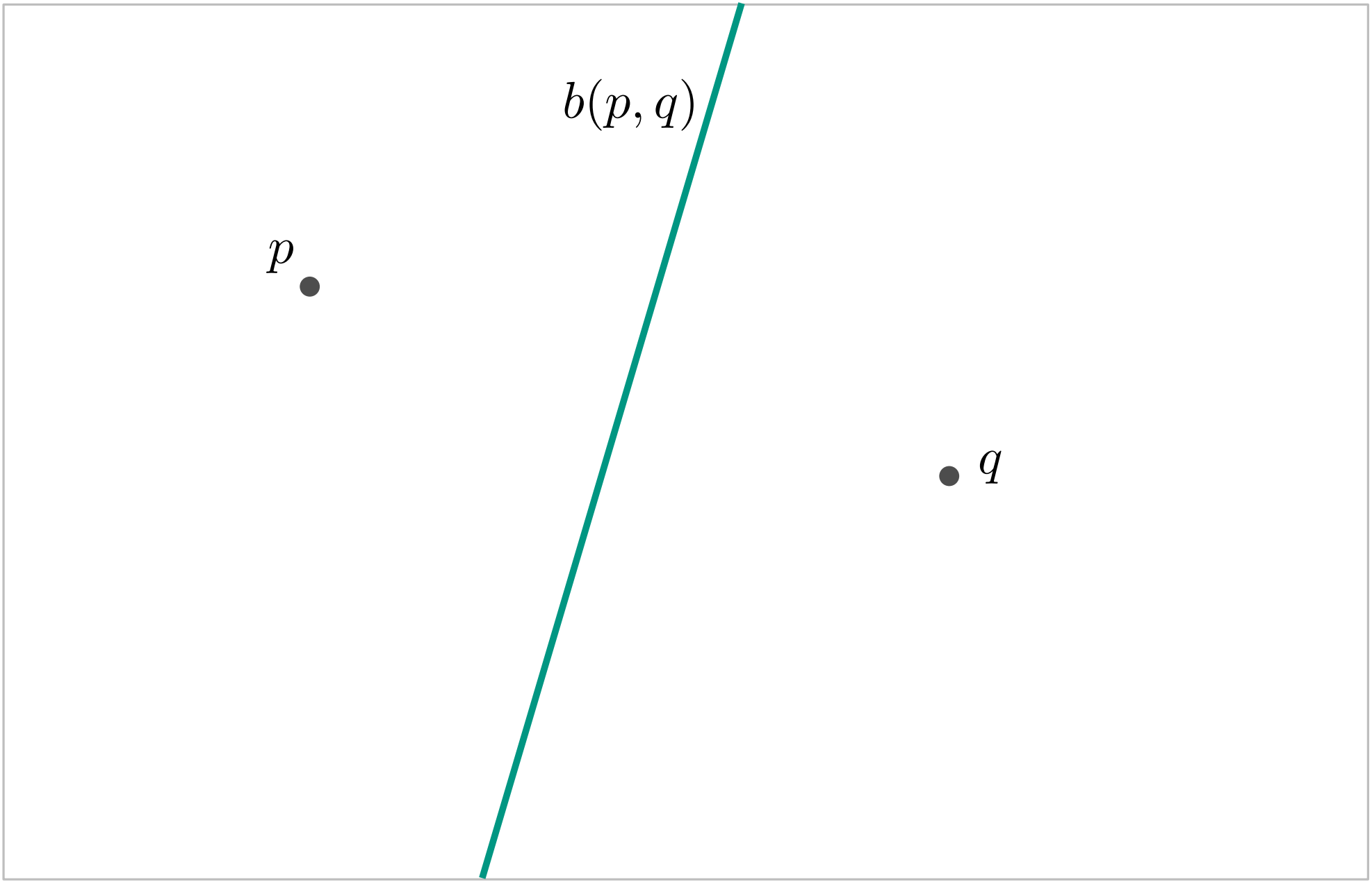
The Post Office Problem



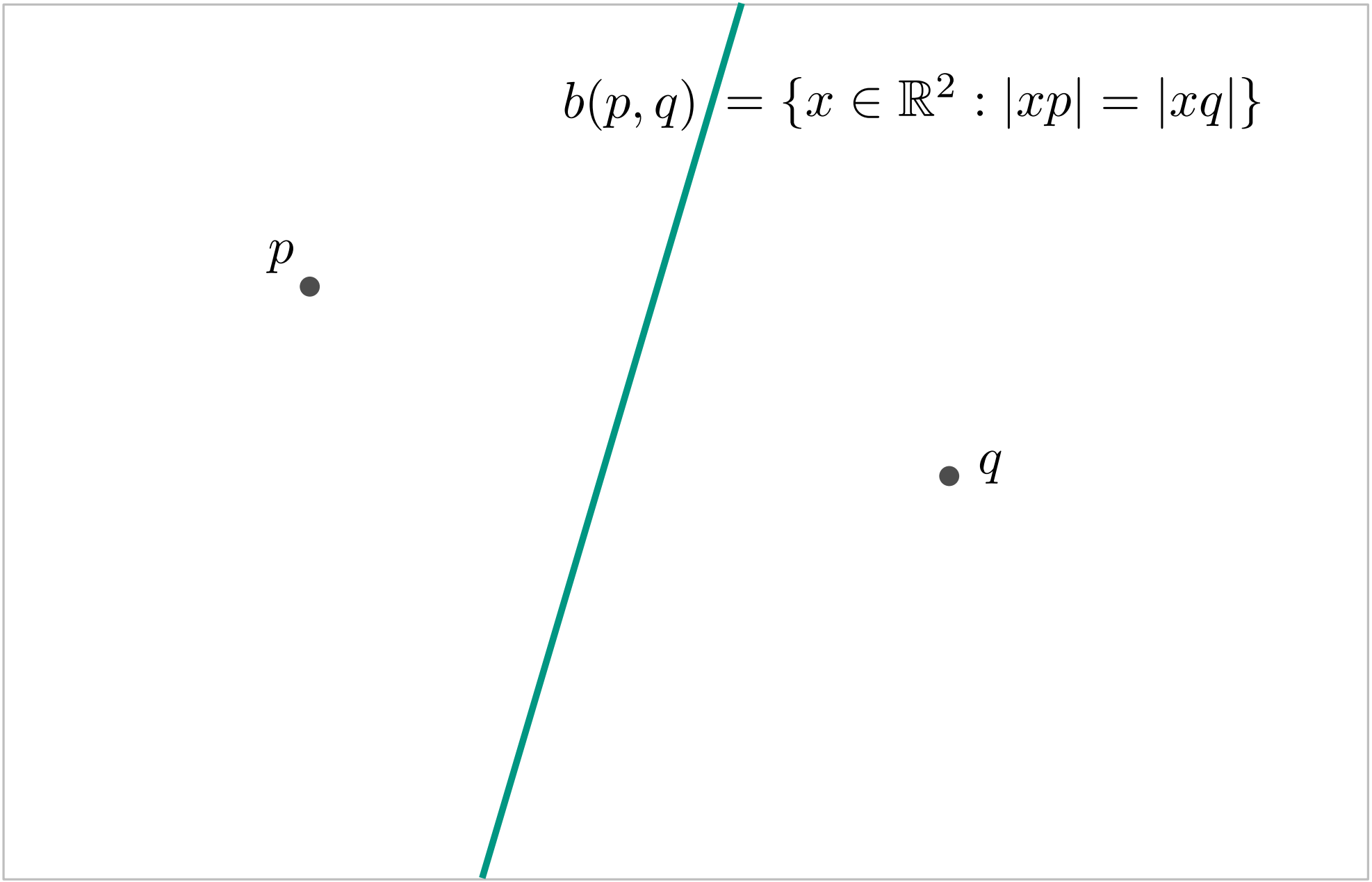
Bisector and Half-Plane



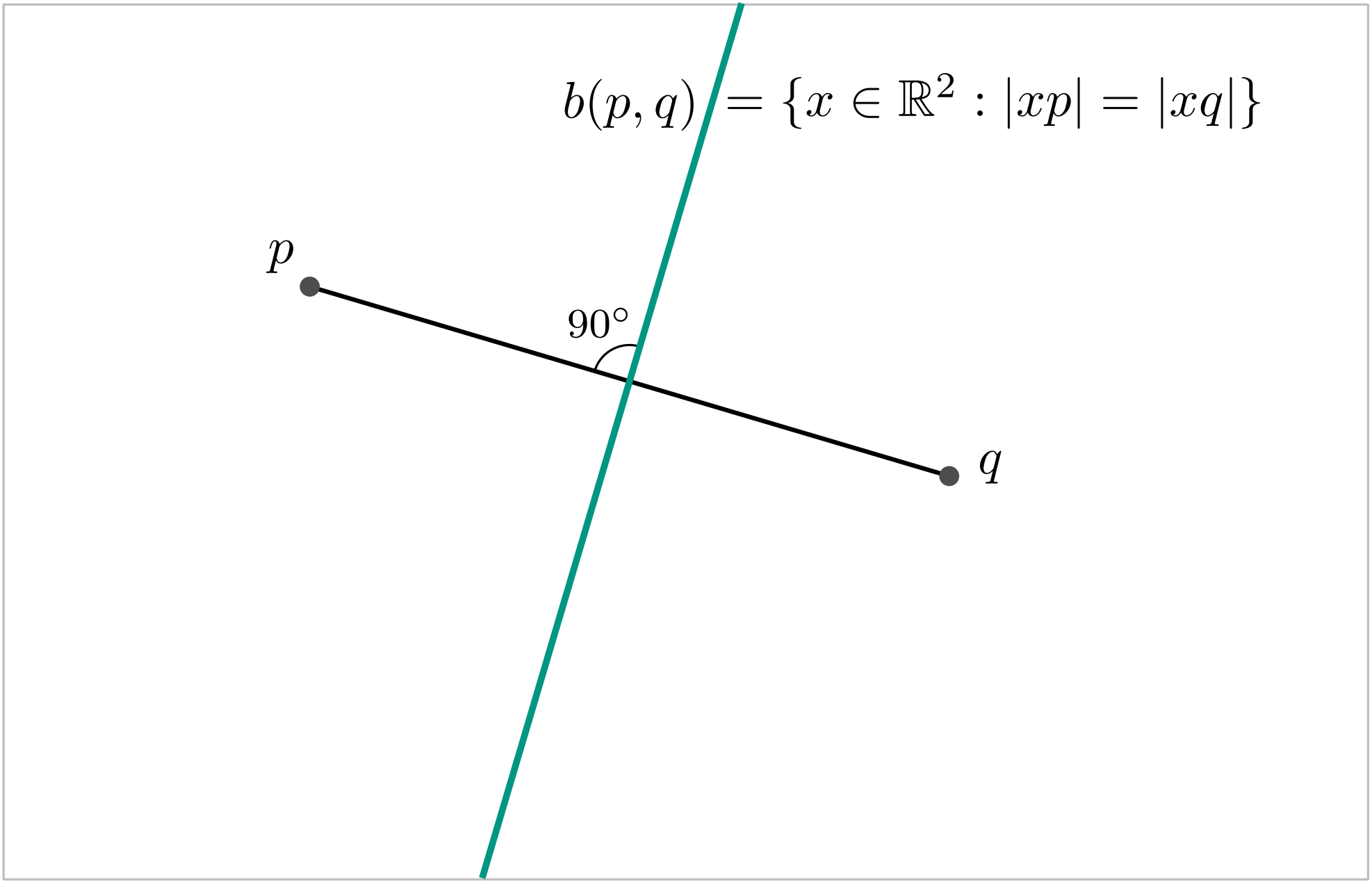
Bisector and Half-Plane



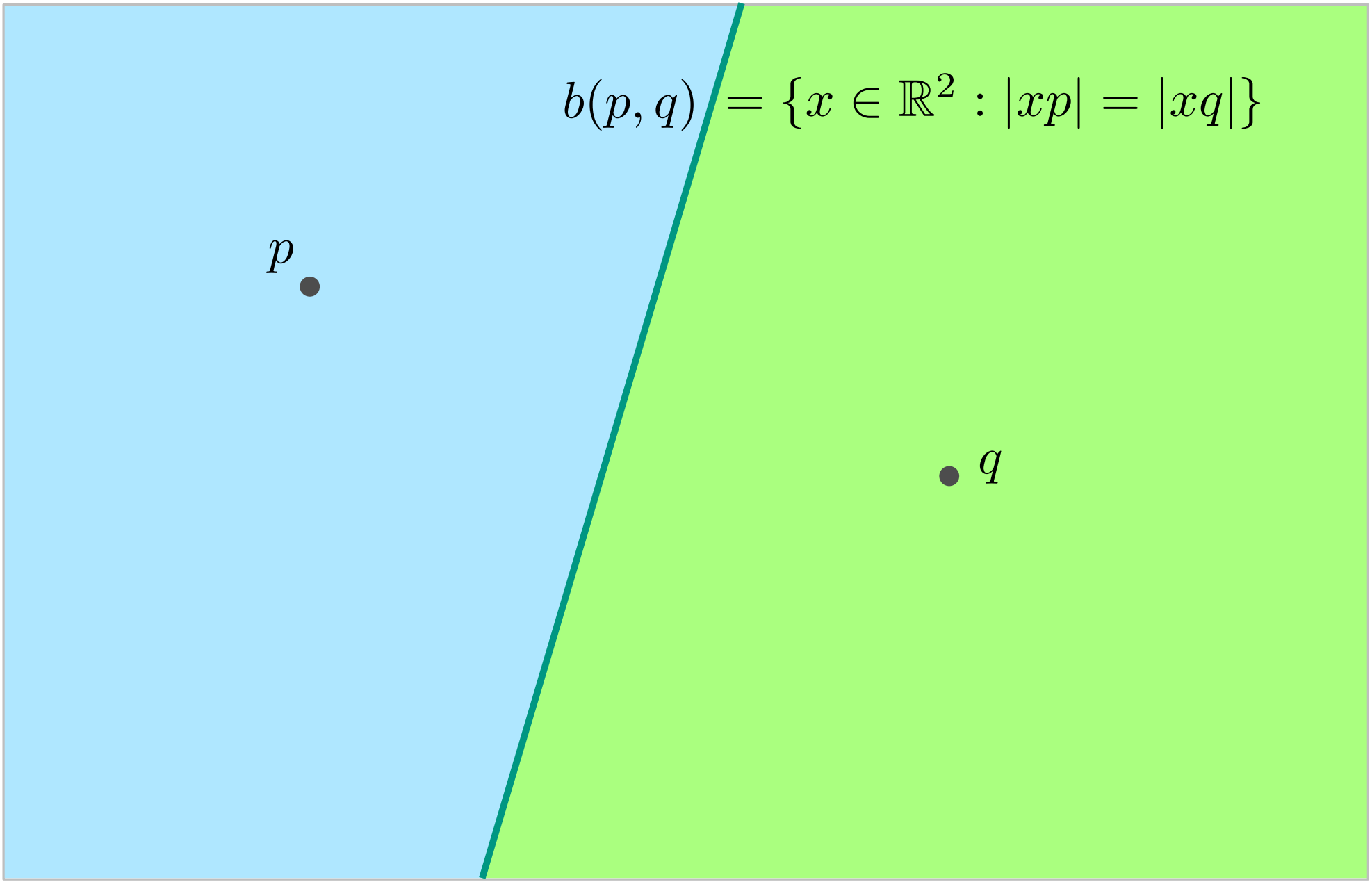
Bisector and Half-Plane



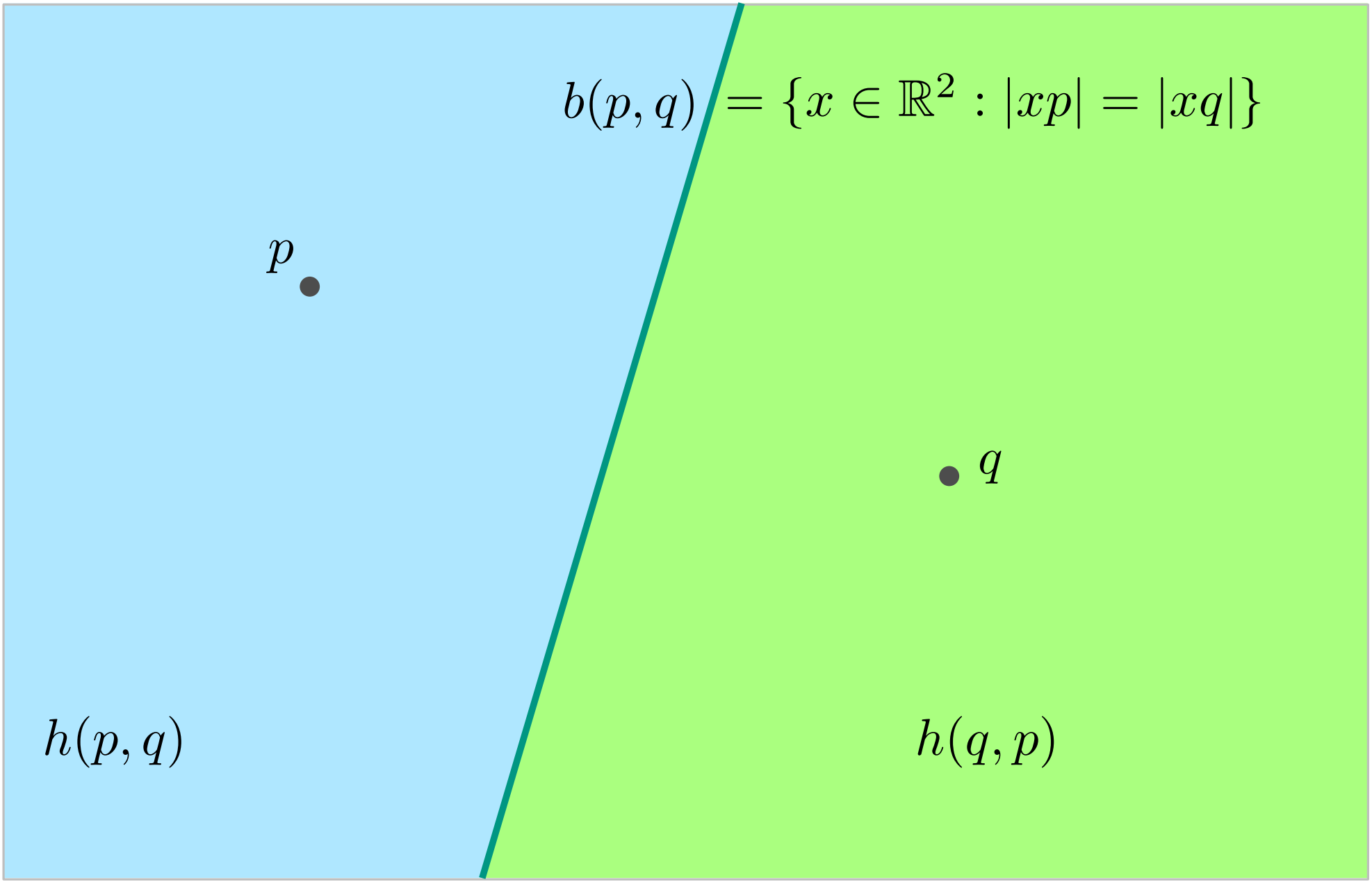
Bisector and Half-Plane



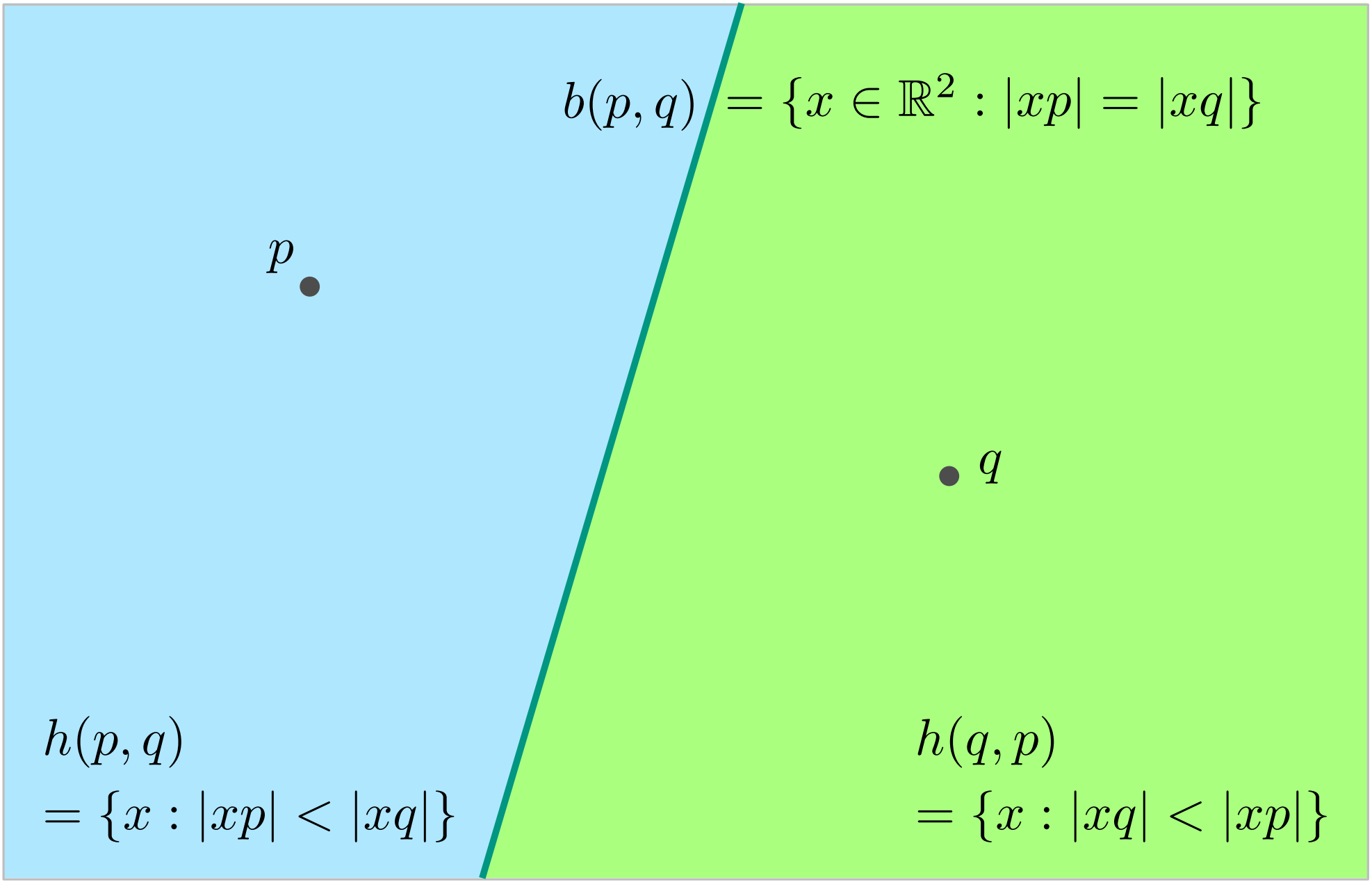
Bisector and Half-Plane



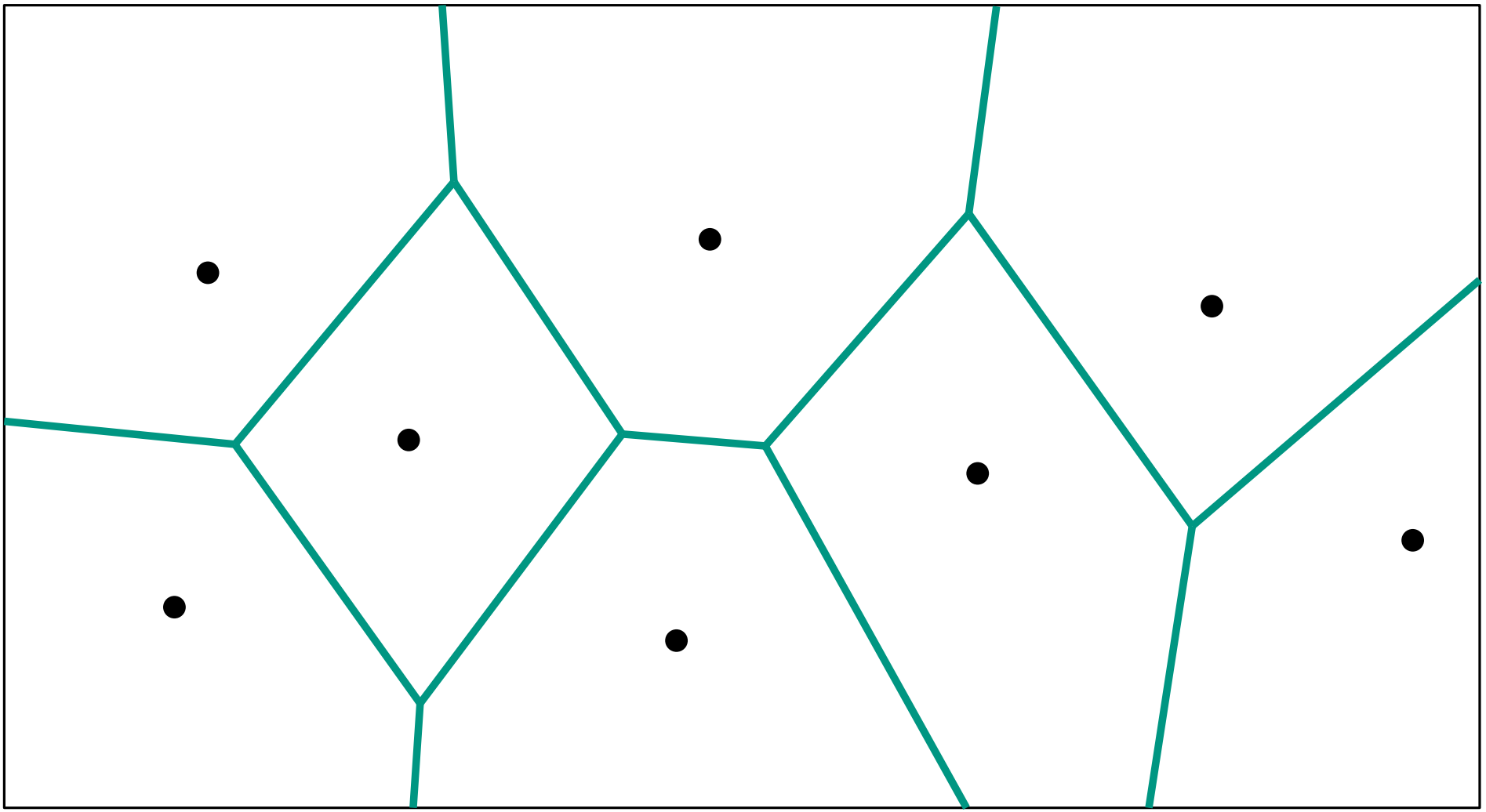
Bisector and Half-Plane



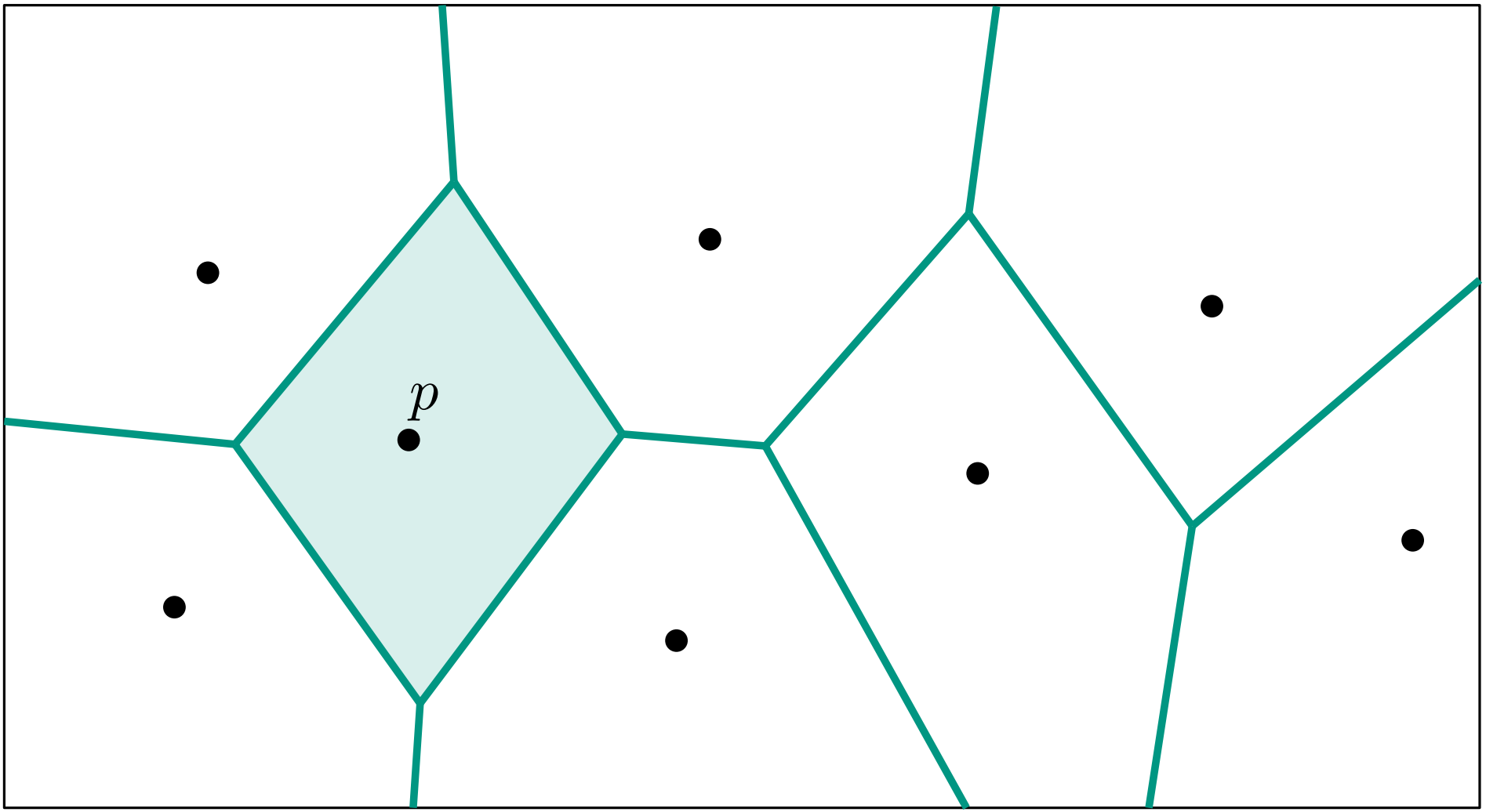
Bisector and Half-Plane



Voronoi Cell

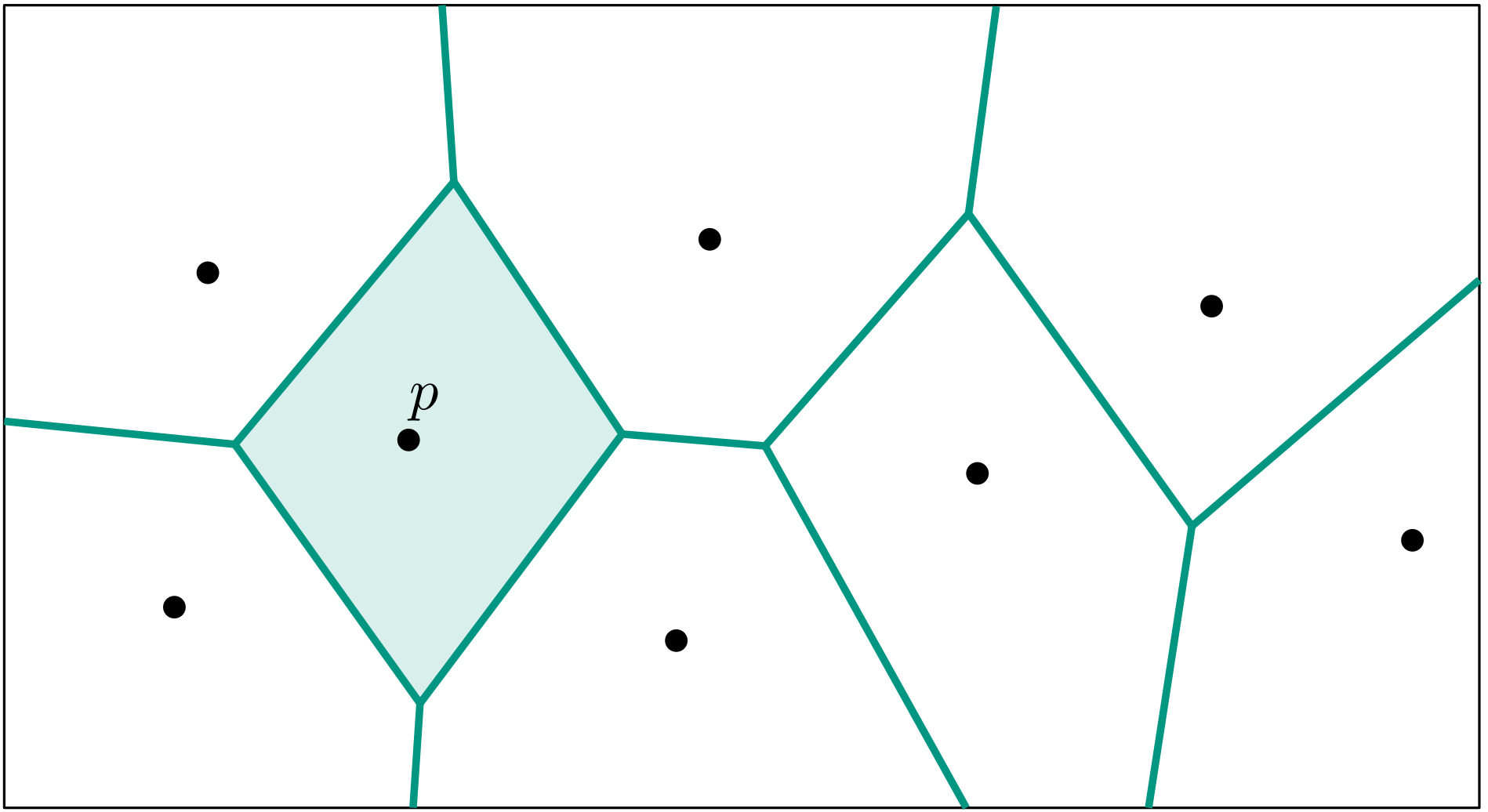


Voronoi Cell



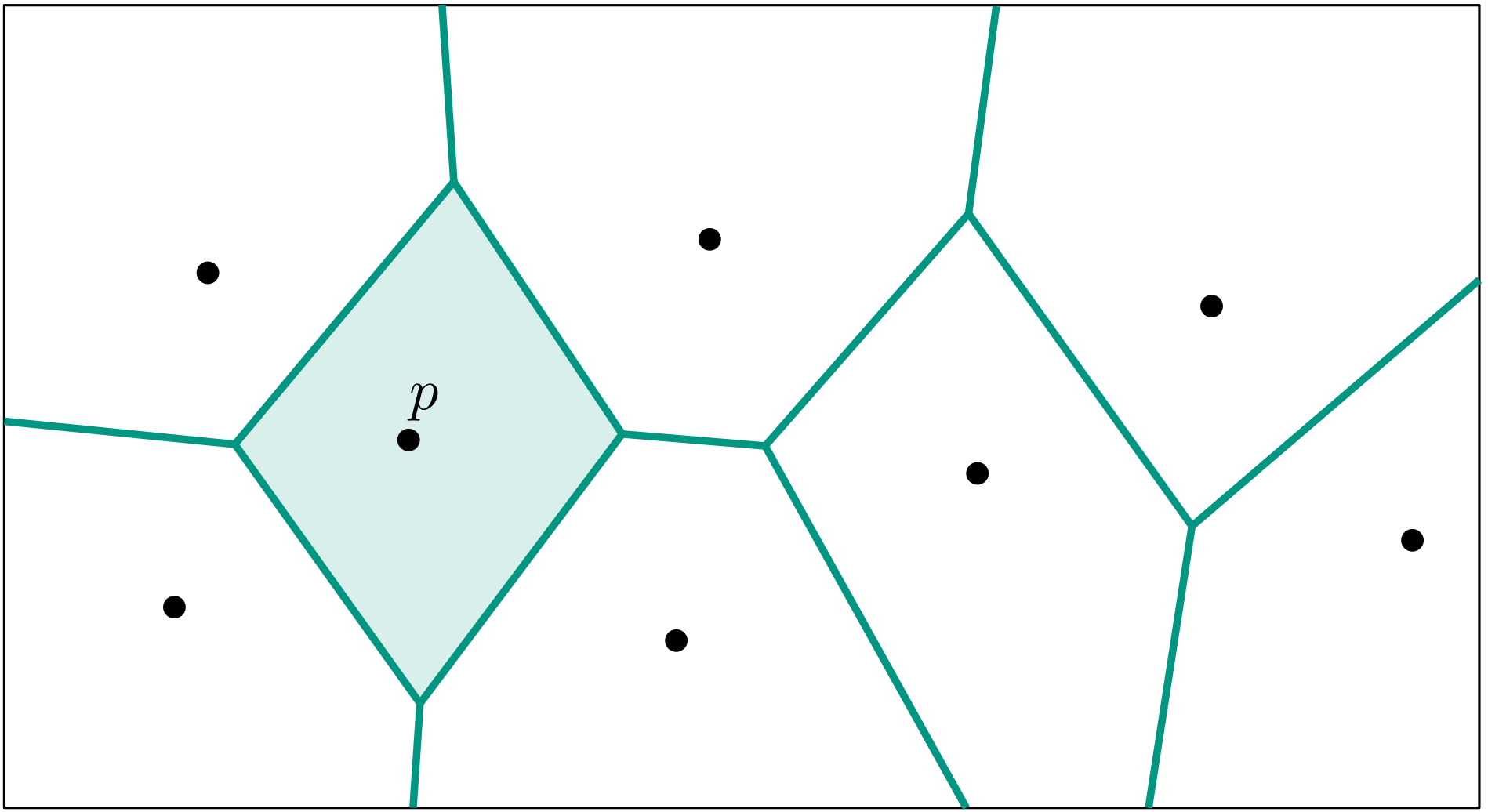
Voronoi Cell

$\mathcal{V}(p)$



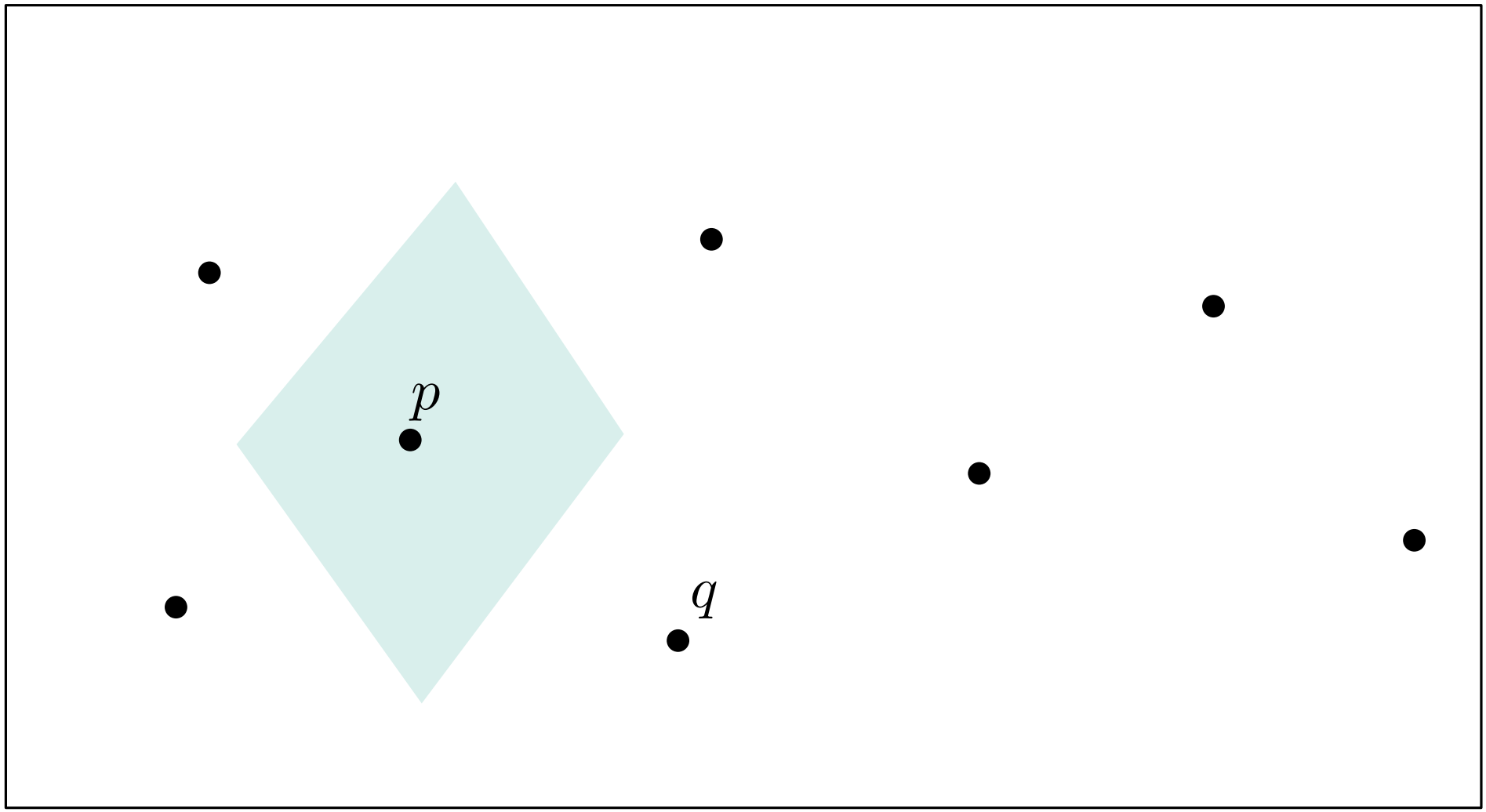
Voronoi Cell

$$\mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq|, \forall q \in P \setminus \{p\}\}$$



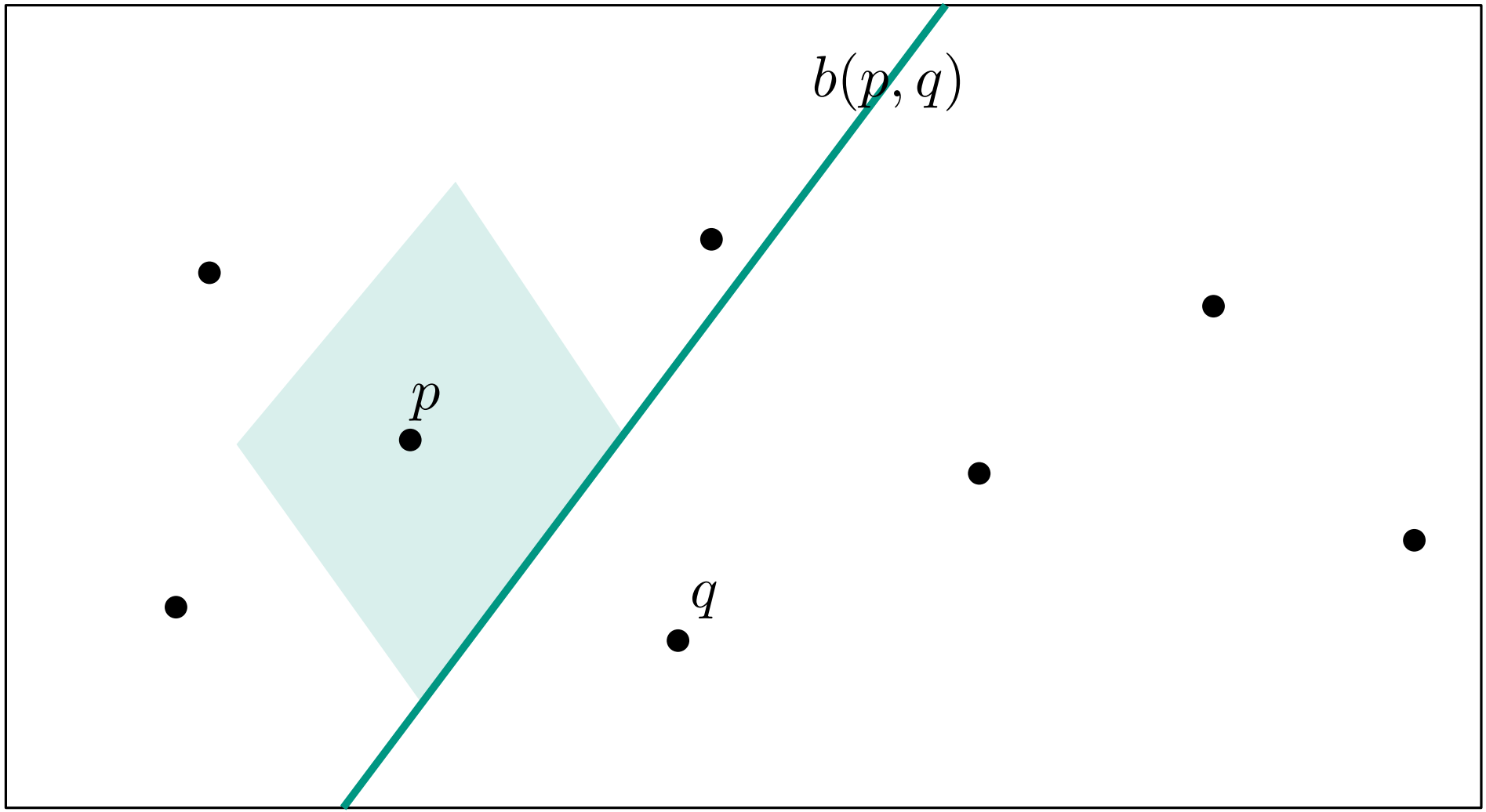
Voronoi Cell

$$\mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq|, \forall q \in P \setminus \{p\}\}$$



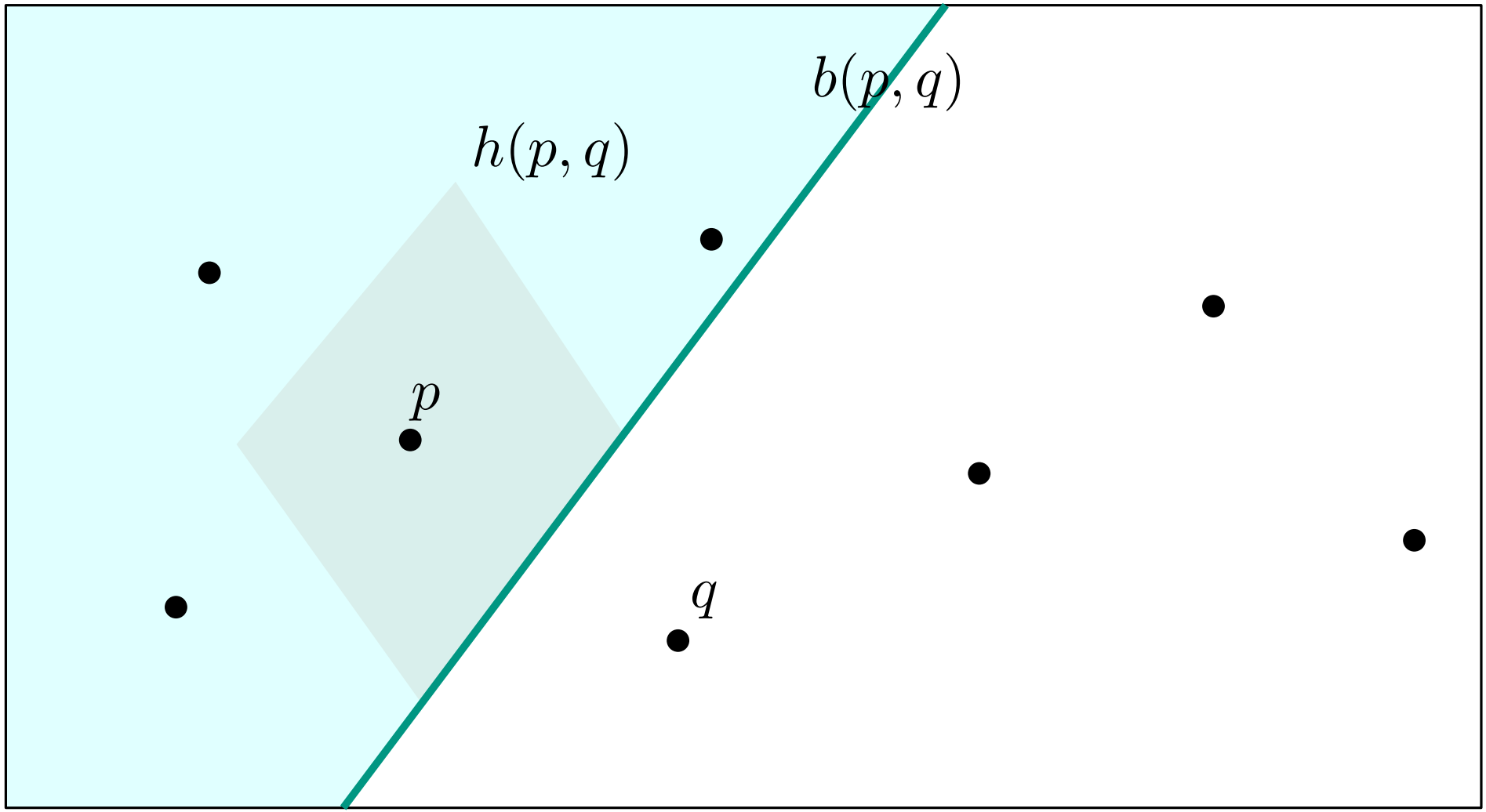
Voronoi Cell

$$\mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq|, \forall q \in P \setminus \{p\}\}$$



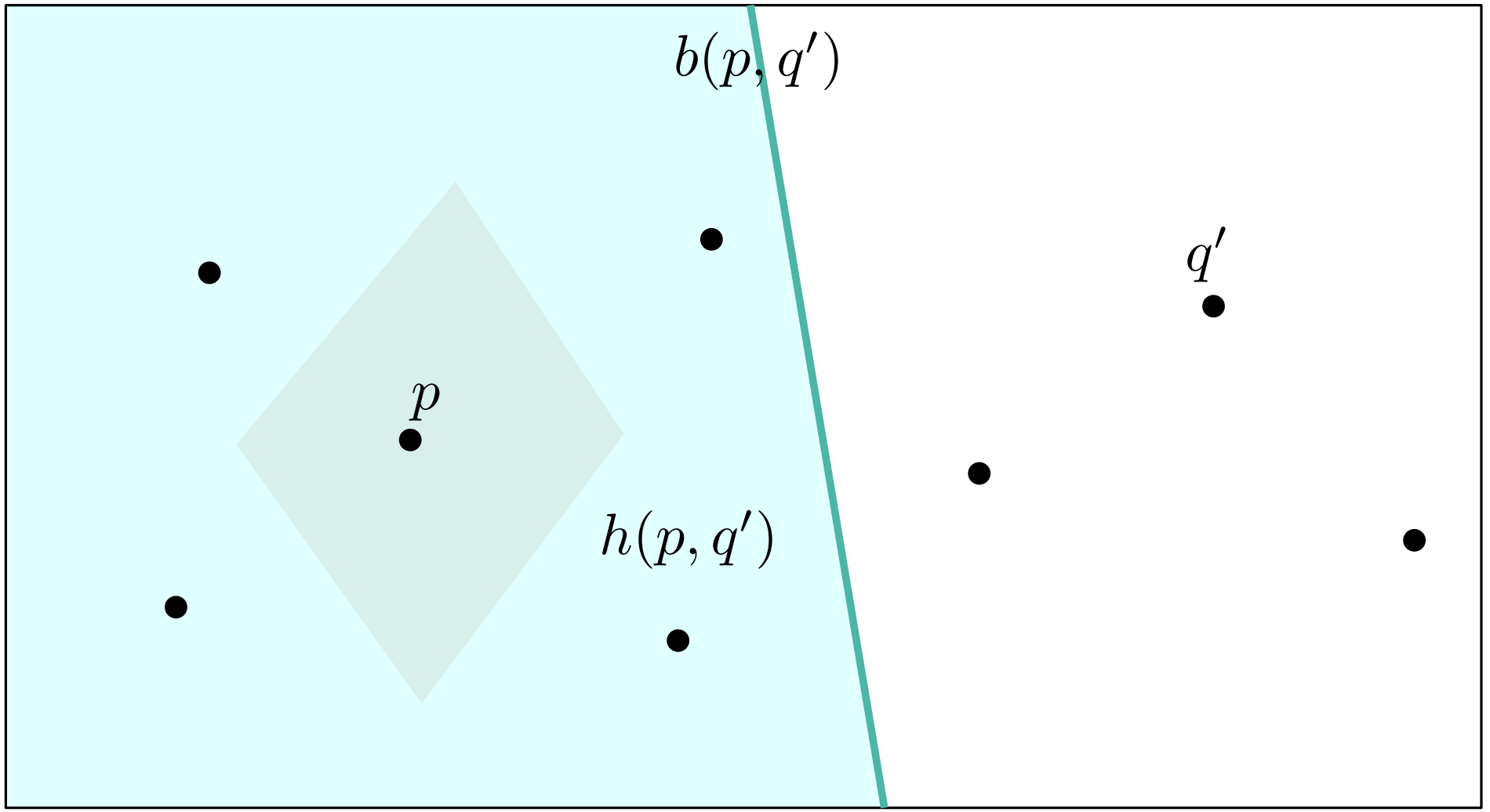
Voronoi Cell

$$\mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq|, \forall q \in P \setminus \{p\}\}$$



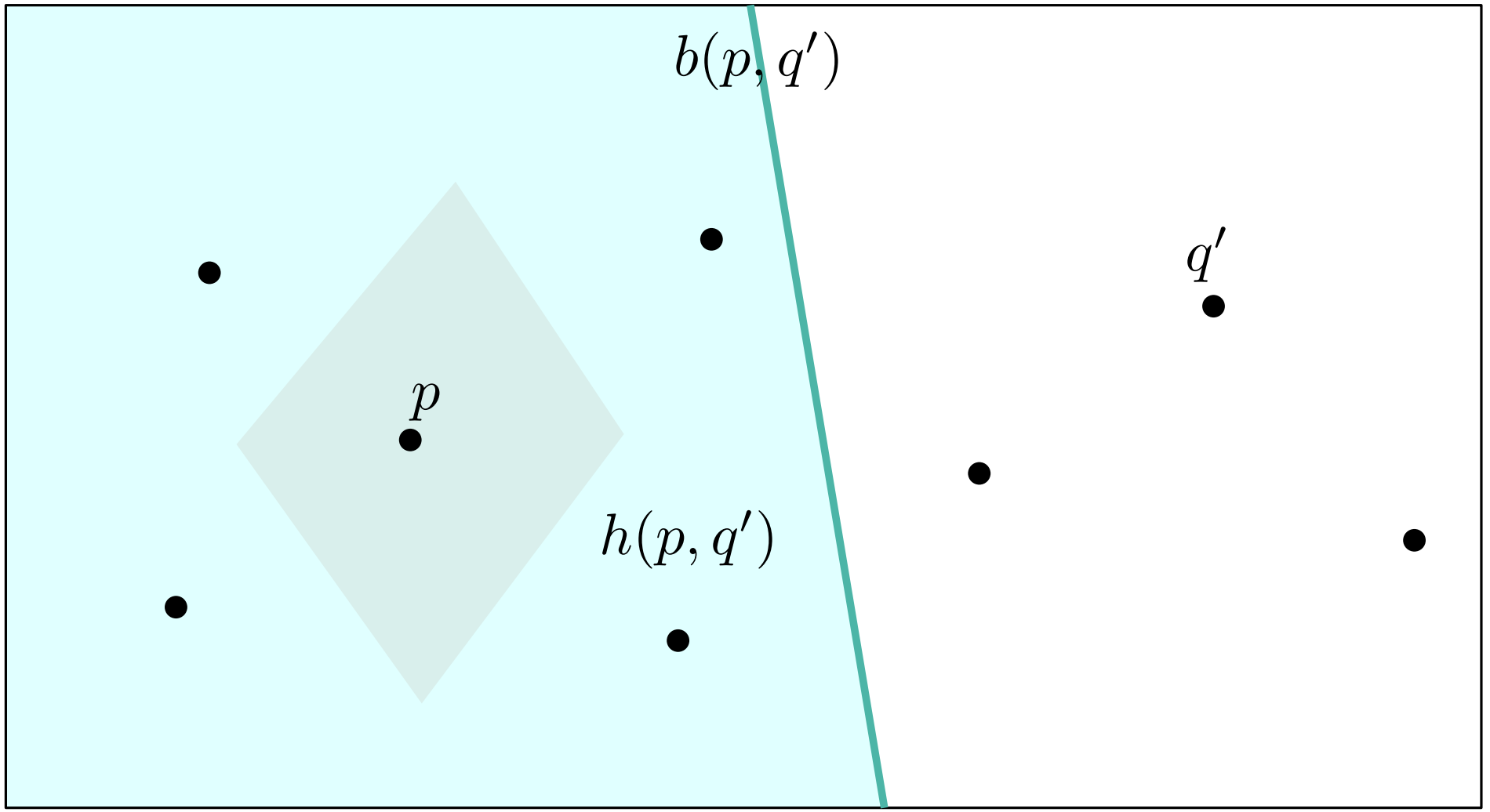
Voronoi Cell

$$\mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq|, \forall q \in P \setminus \{p\}\}$$

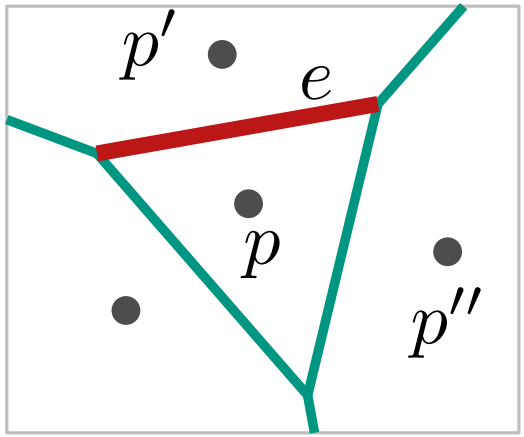


Voronoi Cell

$$\mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq|, \forall q \in P \setminus \{p\}\}$$
$$= \bigcap_{q \in P \setminus \{p\}} h(p, q)$$

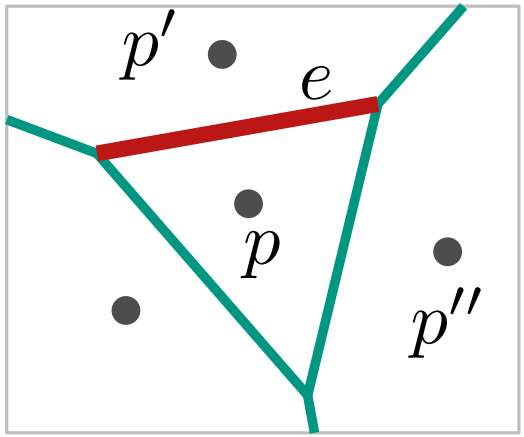


Voronoi Edges and Vertices



- Voronoi edge e between $\mathcal{V}(p)$ and $\mathcal{V}(p')$

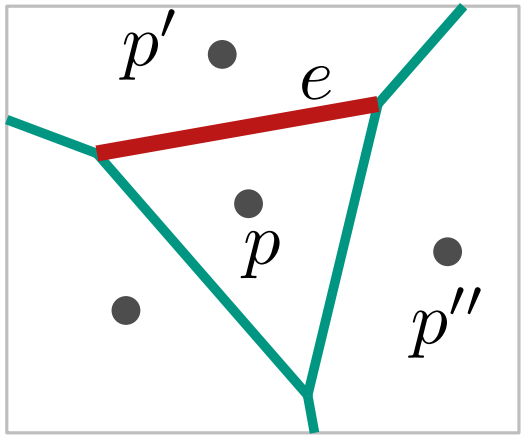
Voronoi Edges and Vertices



- Voronoi edge e between $\mathcal{V}(p)$ and $\mathcal{V}(p')$

$$e = \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')$$

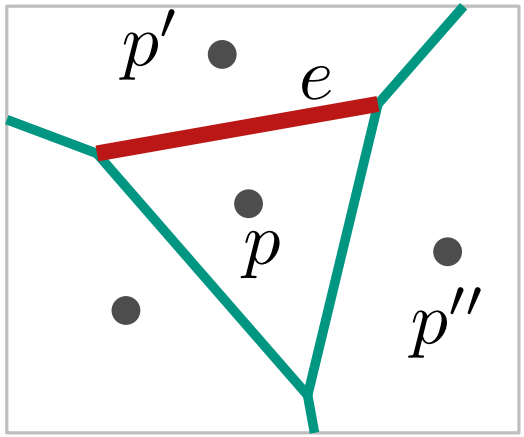
Voronoi Edges and Vertices



- Voronoi edge e between $\mathcal{V}(p)$ and $\mathcal{V}(p')$

$$e = \text{int}\Big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p') \Big)$$ d.h. without endpoints

Voronoi Edges and Vertices

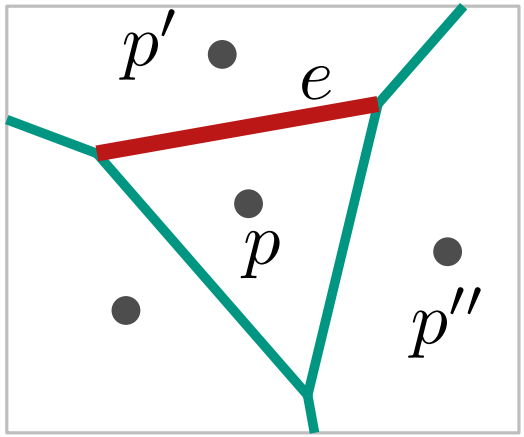


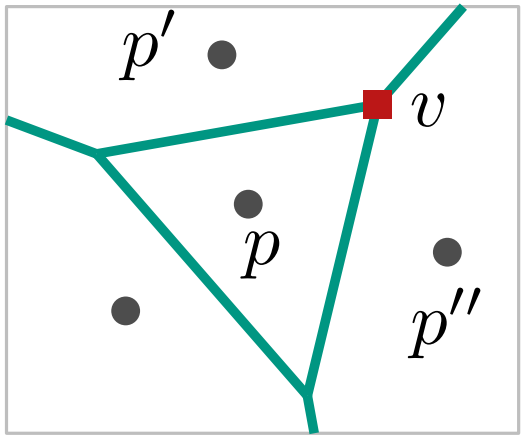
- Voronoi edge e between $\mathcal{V}(p)$ and $\mathcal{V}(p')$

$$e = \text{int}\big(\ \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\ \big)$$ d.h. without endpoints

$$= \{x \in \mathbb{R}^2 : |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$

Voronoi Edges and Vertices



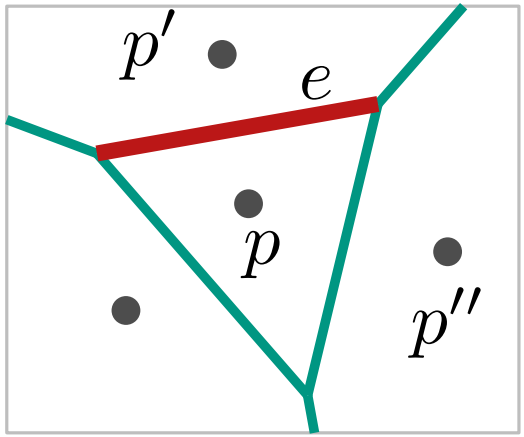


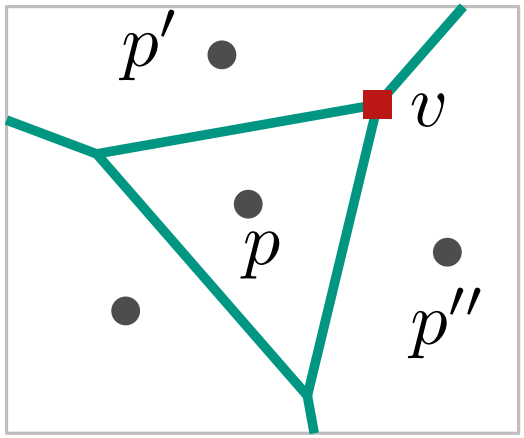
- Voronoi edge e between $\mathcal{V}(p)$ and $\mathcal{V}(p')$

$$e = \text{int}\big(\ \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\ \big)$$ d.h. without endpoints

$$= \{x \in \mathbb{R}^2 : |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$

Voronoi Edges and Vertices





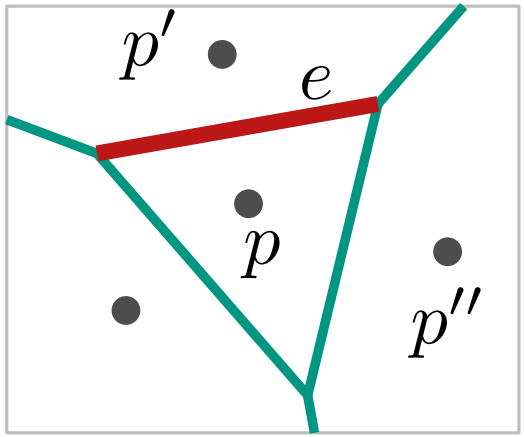
- Voronoi edge e between $\mathcal{V}(p)$ and $\mathcal{V}(p')$

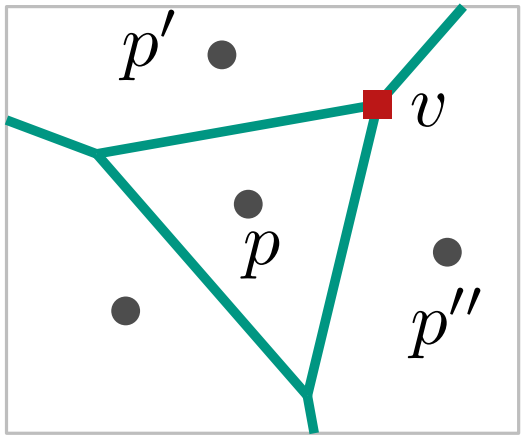
$$e = \text{int}\big(\ \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\ \big)$$ d.h. without endpoints

$$= \{x \in \mathbb{R}^2 : |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$

- Voronoi vertex v among $\mathcal{V}(p)$, $\mathcal{V}(p')$, and $\mathcal{V}(p'')$

Voronoi Edges and Vertices





- Voronoi edge e between $\mathcal{V}(p)$ and $\mathcal{V}(p')$

$$e = \text{int}\big(\ \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\ \big)$$ d.h. without endpoints

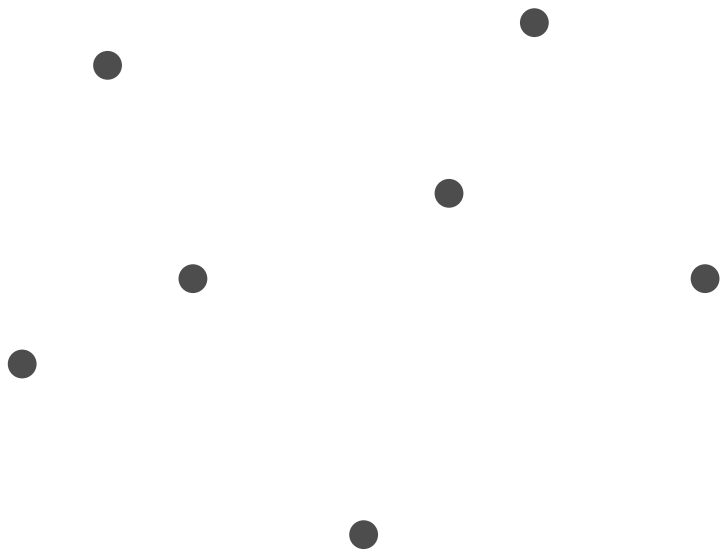
$$= \{x \in \mathbb{R}^2 : |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$

- Voronoi vertex v among $\mathcal{V}(p)$, $\mathcal{V}(p')$, and $\mathcal{V}(p'')$

$$v = \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p') \cap \partial\mathcal{V}(p'')$$

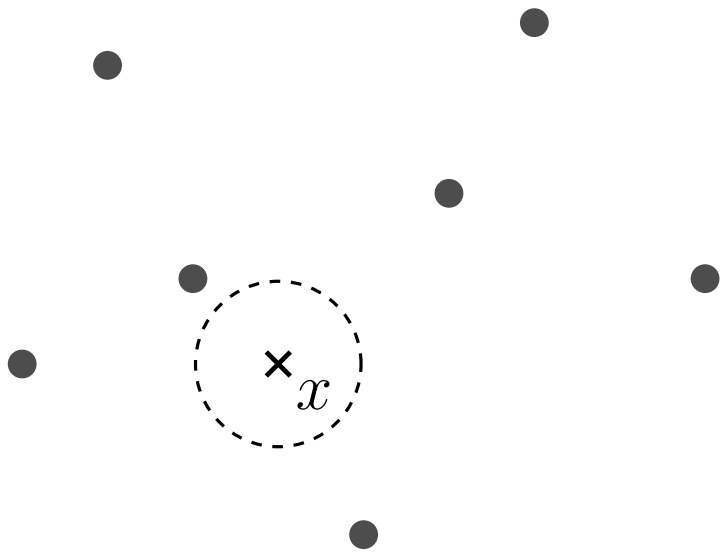
Growing Circle

- Growing a circle from a point $x \in \mathbb{R}^2$ first hits



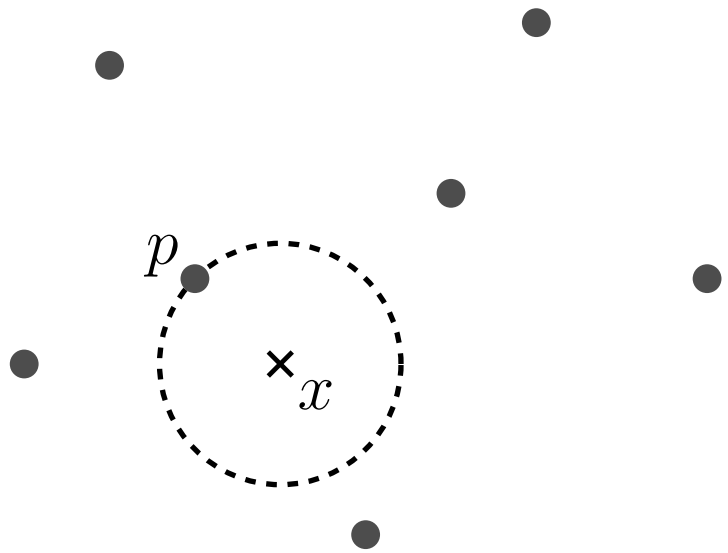
Growing Circle

- Growing a circle from a point $x \in \mathbb{R}^2$ first hits



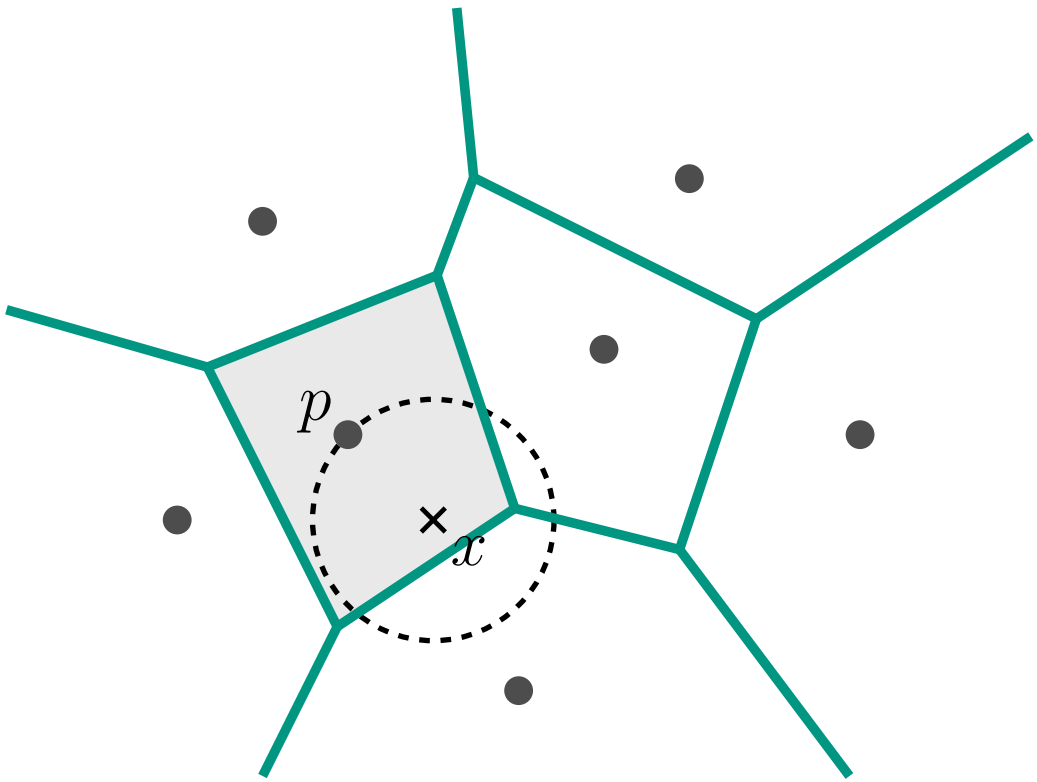
Growing Circle

- Growing a circle from a point $x \in \mathbb{R}^2$ first hits
  - one site $p \rightarrow x \in \mathcal{V}(p)$



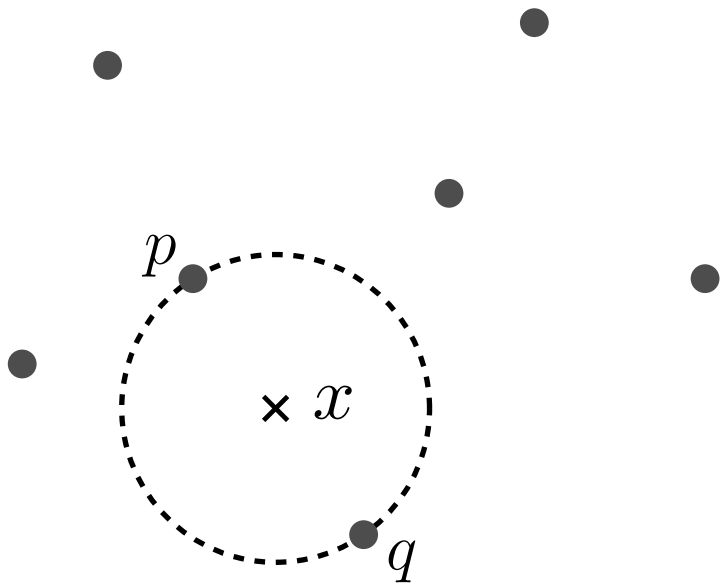
Growing Circle

- Growing a circle from a point $x \in \mathbb{R}^2$ first hits
  - one site $p \rightarrow x \in \mathcal{V}(p)$



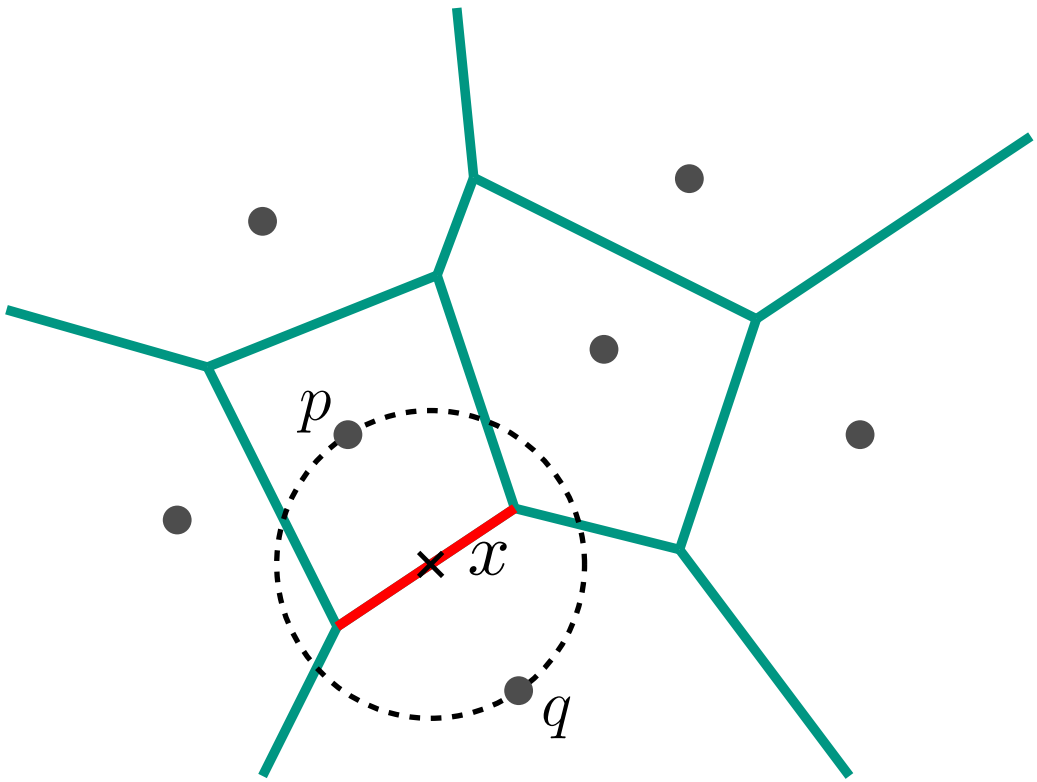
Growing Circle

- Growing a circle from a point $x \in \mathbb{R}^2$ first hits
  - one site $p \rightarrow x \in \mathcal{V}(p)$
  - two sites p, $q \rightarrow x \in$ edge $= \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$



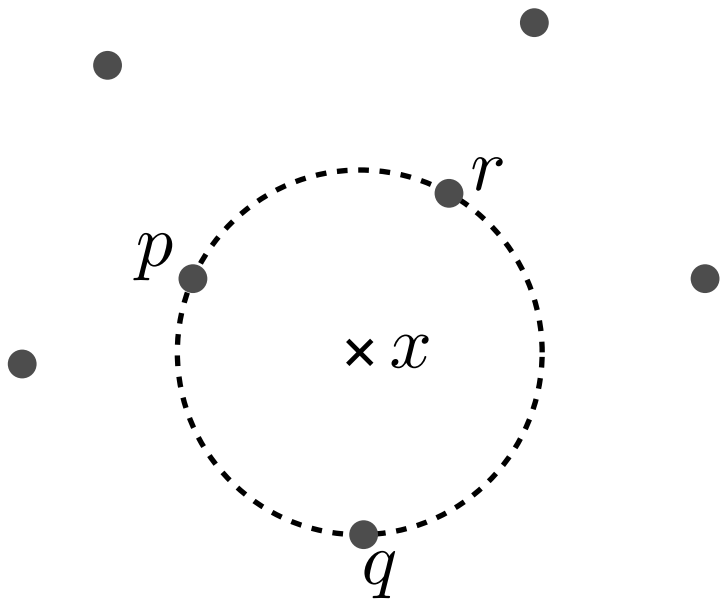
Growing Circle

- Growing a circle from a point $x \in \mathbb{R}^2$ first hits
  - one site $p \rightarrow x \in \mathcal{V}(p)$
  - two sites p, $q \rightarrow x \in$ edge $= \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$



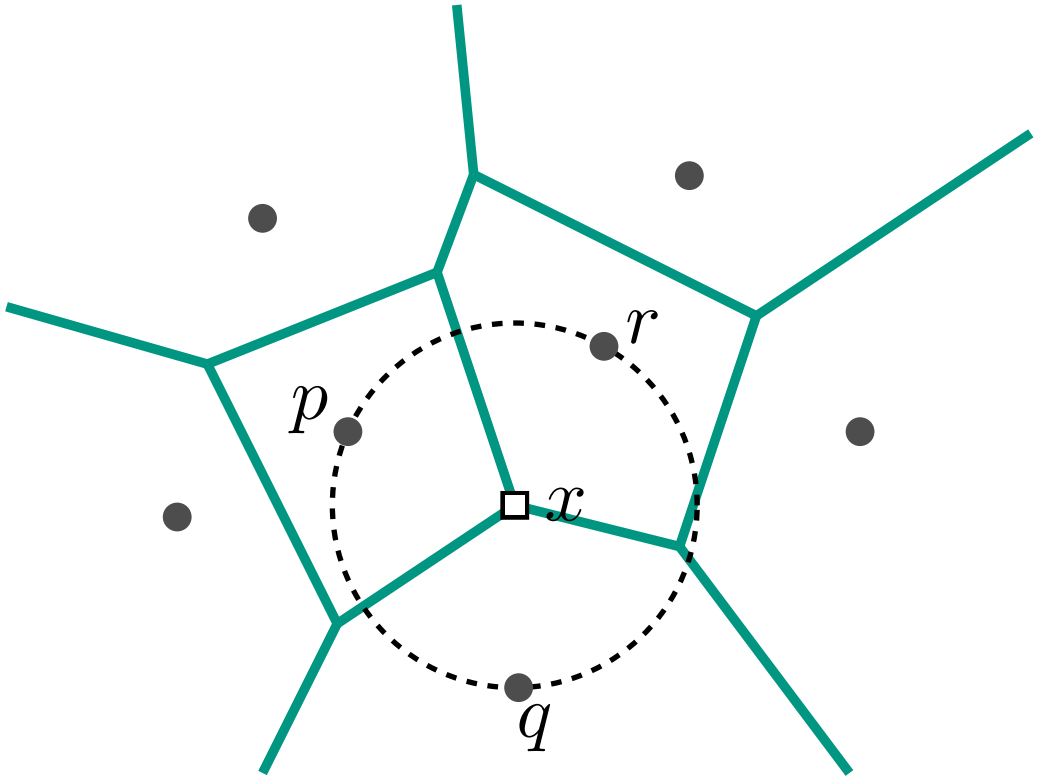
Growing Circle

- Growing a circle from a point $x \in \mathbb{R}^2$ first hits
  - one site $p \rightarrow x \in \mathcal{V}(p)$
  - two sites p, $q \rightarrow x \in$ edge $= \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$
  - $>$ two sites p, q , r, ...
    $\rightarrow x =$ vertex $= \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q) \cap \partial\mathcal{V}(r)$



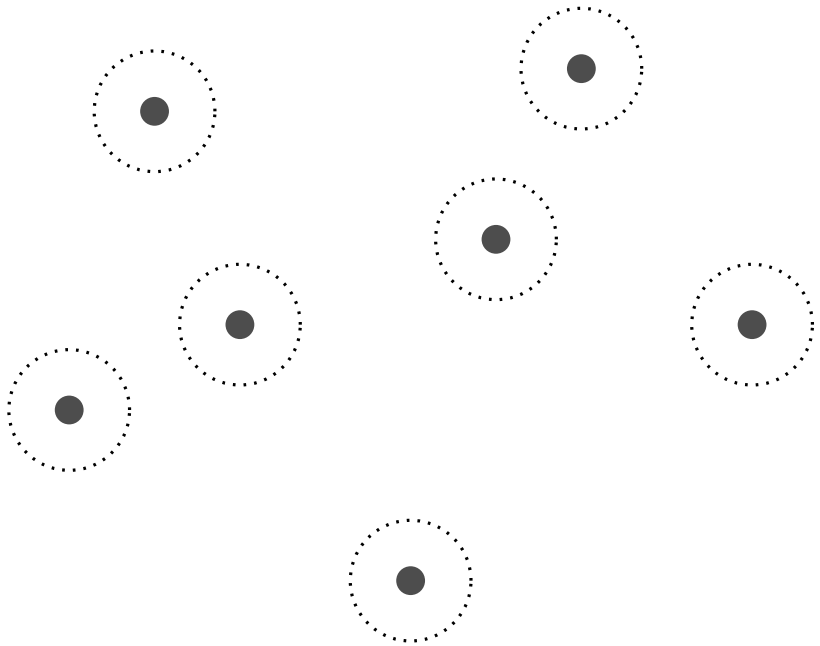
Growing Circle

- Growing a circle from a point $x \in \mathbb{R}^2$ first hits
  - one site $p \rightarrow x \in \mathcal{V}(p)$
  - two sites p, $q \rightarrow x \in$ edge $= \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$
  - $>$ two sites p, q , r, ...
    $\rightarrow x =$ vertex $= \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q) \cap \partial\mathcal{V}(r)$



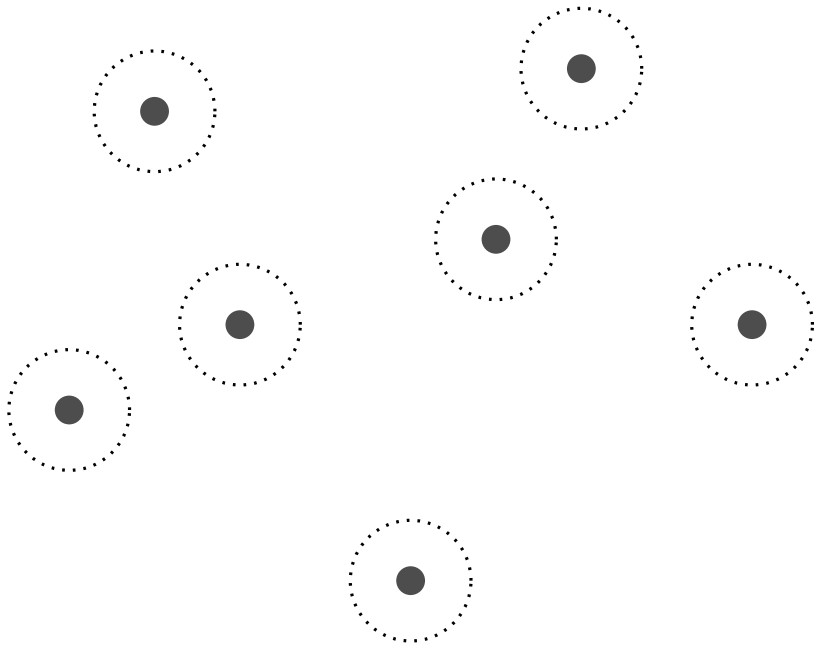
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.



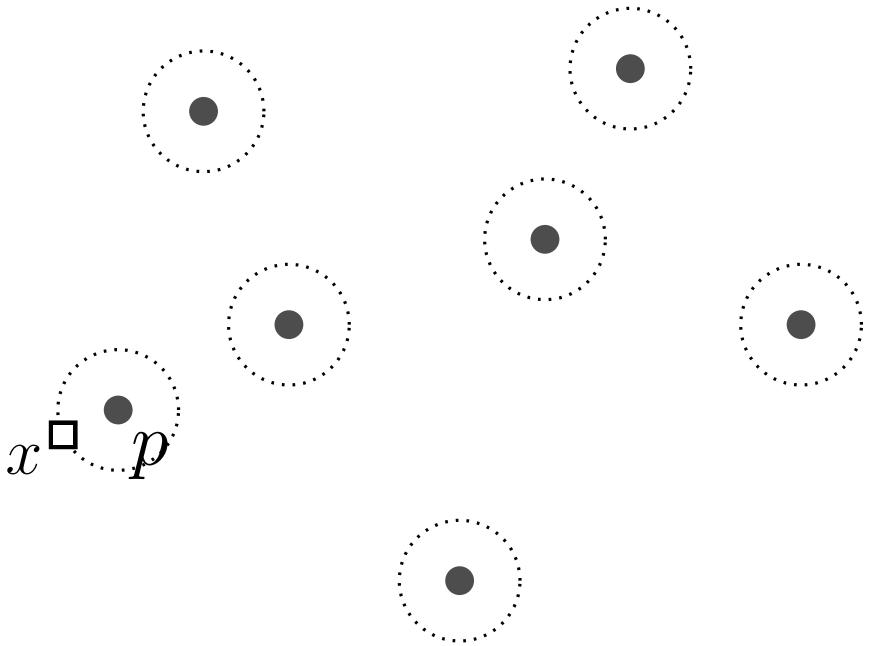
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by



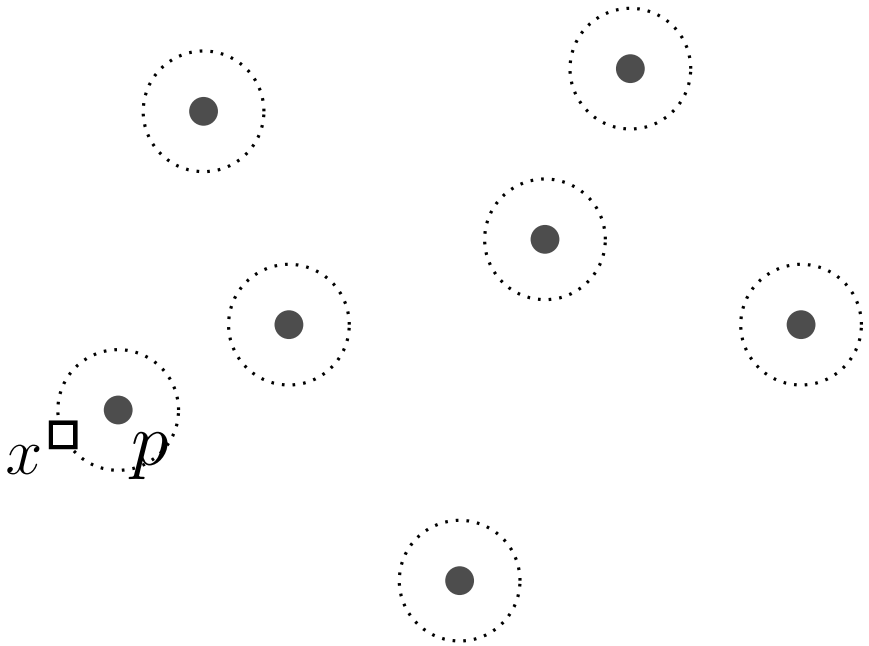
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by



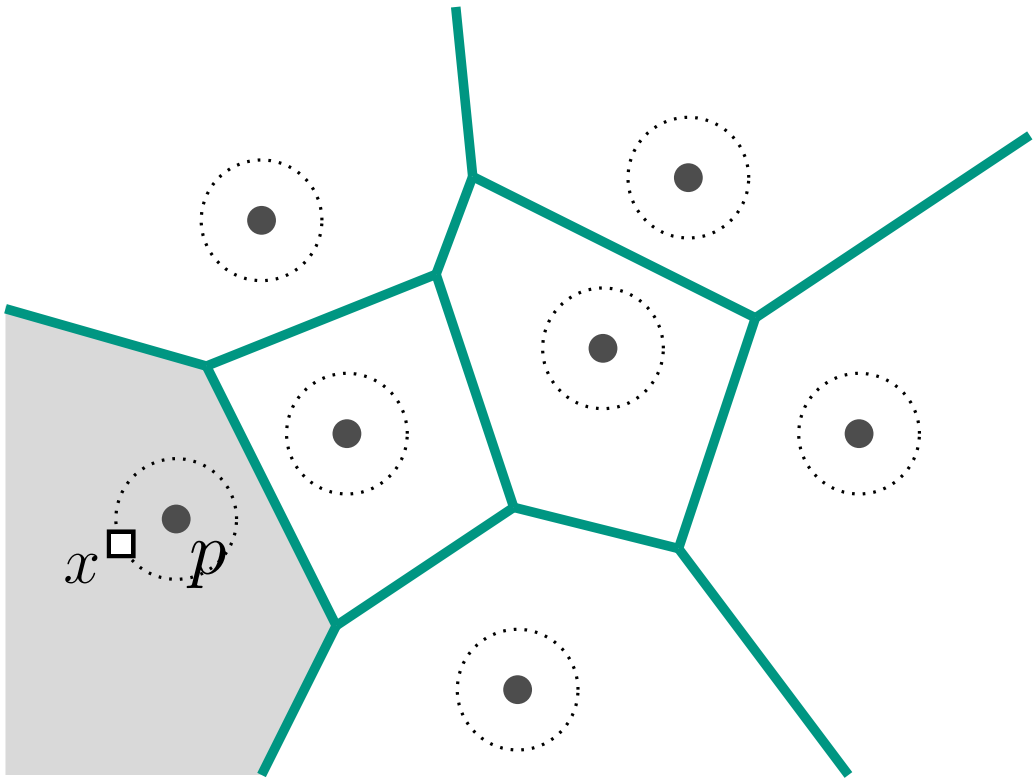
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by
  - $C_p \rightarrow x \in \mathcal{V}(p)$.



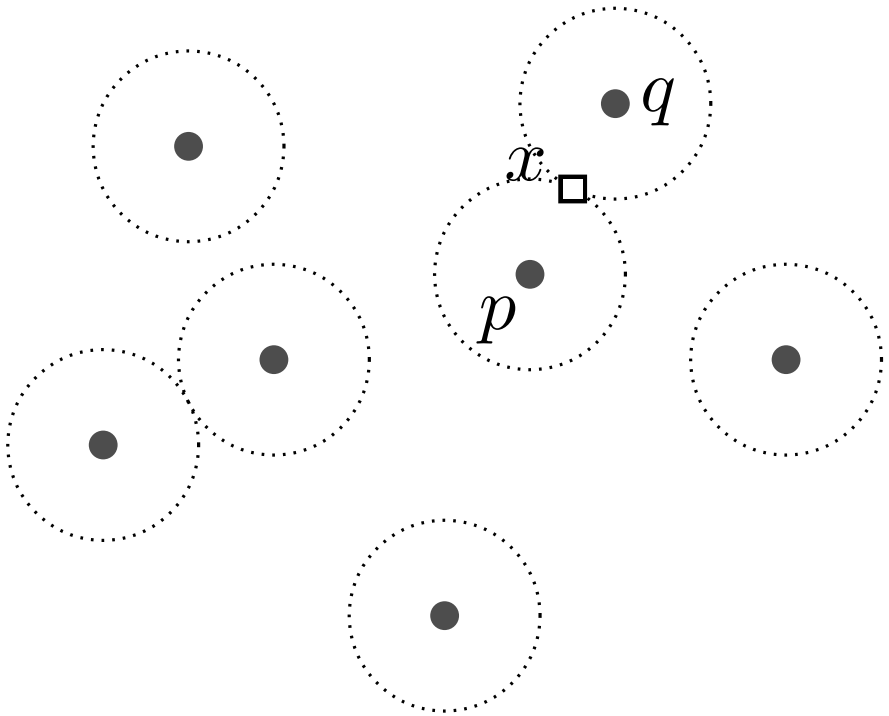
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by
  - $C_p \rightarrow x \in \mathcal{V}(p)$.



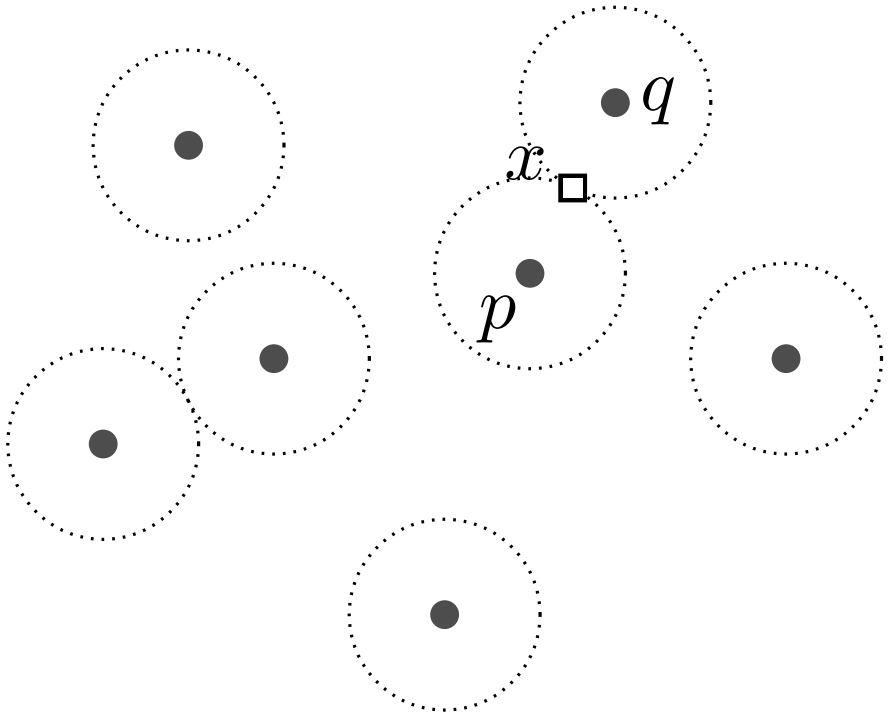
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by
  - $C_p \rightarrow x \in \mathcal{V}(p)$.



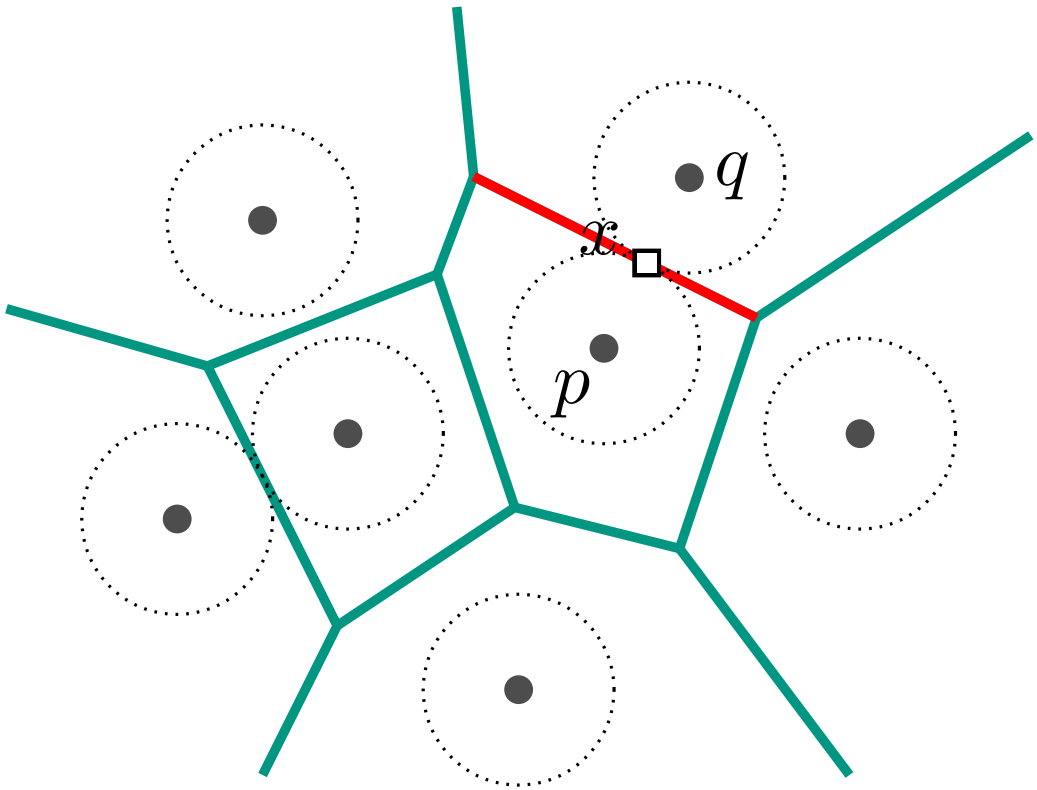
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by
  - $C_p \rightarrow x \in \mathcal{V}(p)$.
  - C_p, $C_q \rightarrow x \in$ edge $: \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$.



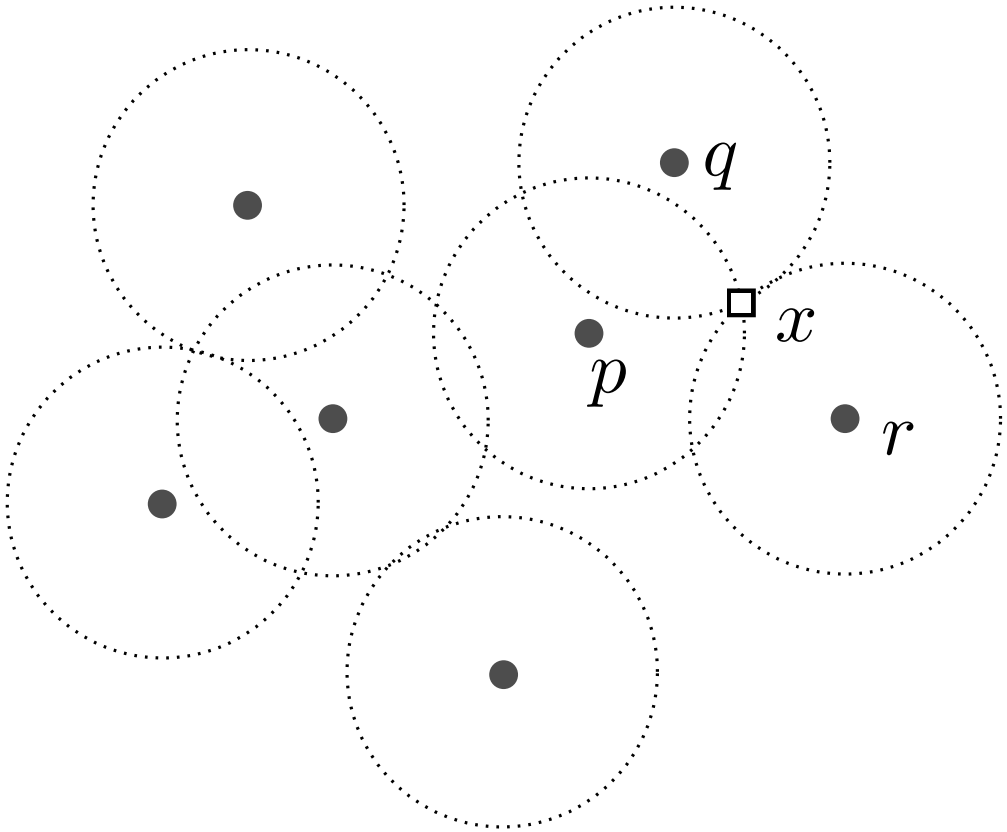
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by
  - $C_p \rightarrow x \in \mathcal{V}(p)$.
  - C_p, $C_q \rightarrow x \in$ edge $: \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$.



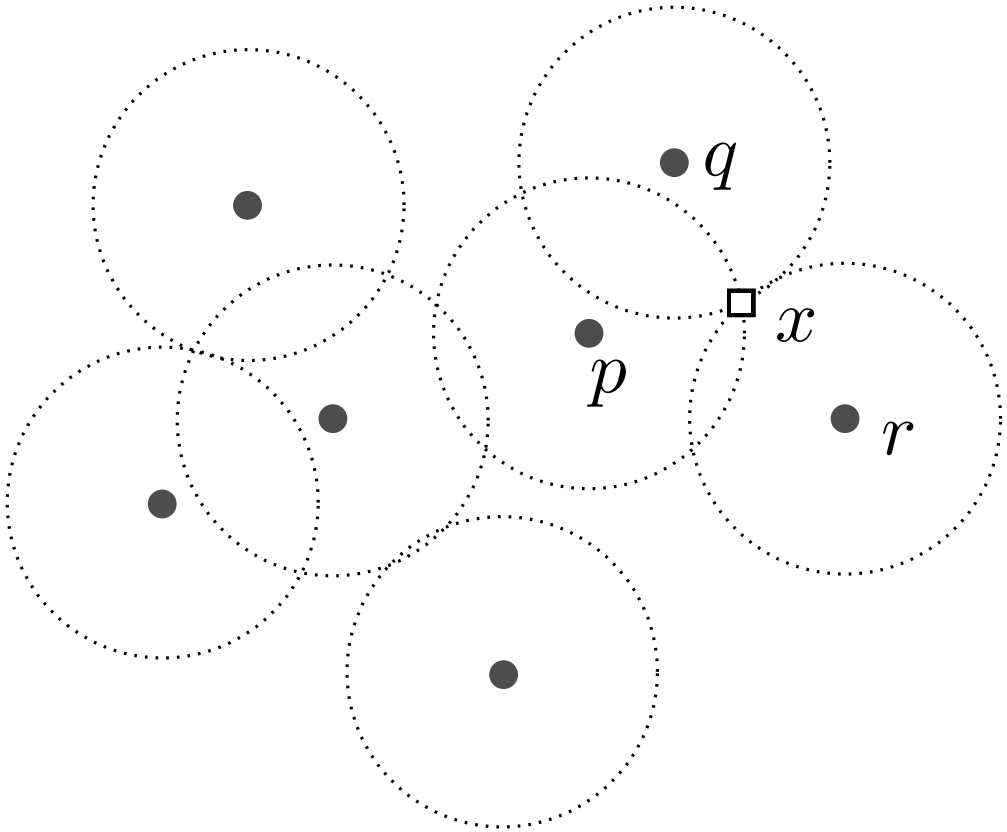
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by
  - $C_p \rightarrow x \in \mathcal{V}(p)$.
  - C_p, $C_q \rightarrow x \in$ edge $: \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$.



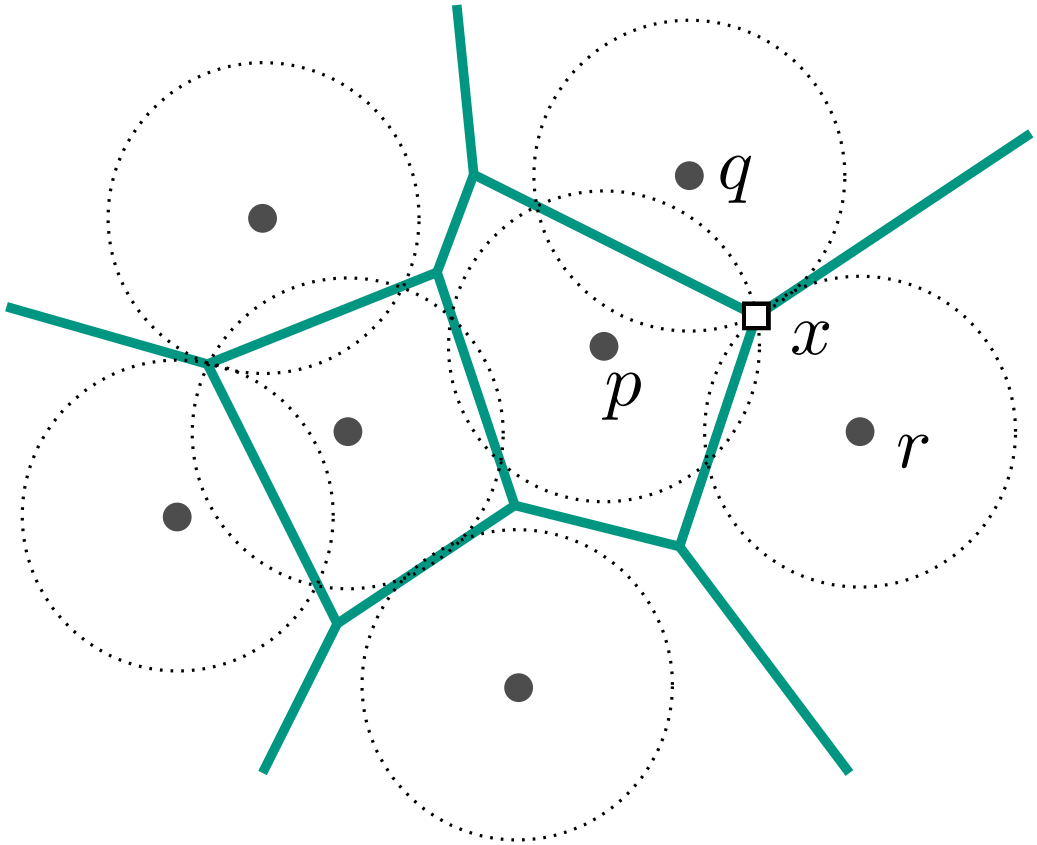
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by
  - $C_p \rightarrow x \in \mathcal{V}(p)$.
  - C_p, $C_q \rightarrow x \in$ edge $: \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$.
  - C_p, C_q, $C_r \rightarrow x =$ vertex $: \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)\partial\mathcal{V}(r)$.



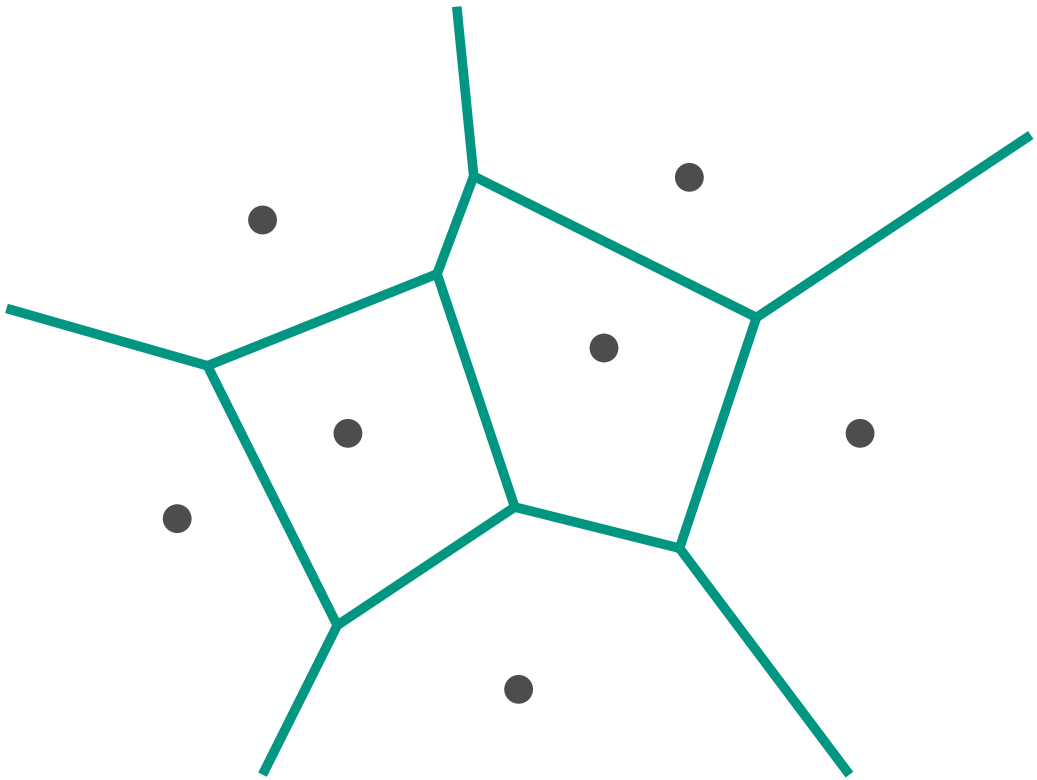
Wavefront Model (Growth Model)

- Grows circles C_p from all sites $\forall p \in P$ at unit speed.
- For each point $x \in \mathbb{R}^2$, if x is first hit by
  - $C_p \rightarrow x \in \mathcal{V}(p)$.
  - C_p, $C_q \rightarrow x \in$ edge $: \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)$.
  - C_p, C_q, $C_r \rightarrow x =$ vertex $: \partial\mathcal{V}(p) \cap \partial\mathcal{V}(q)\partial\mathcal{V}(r)$.



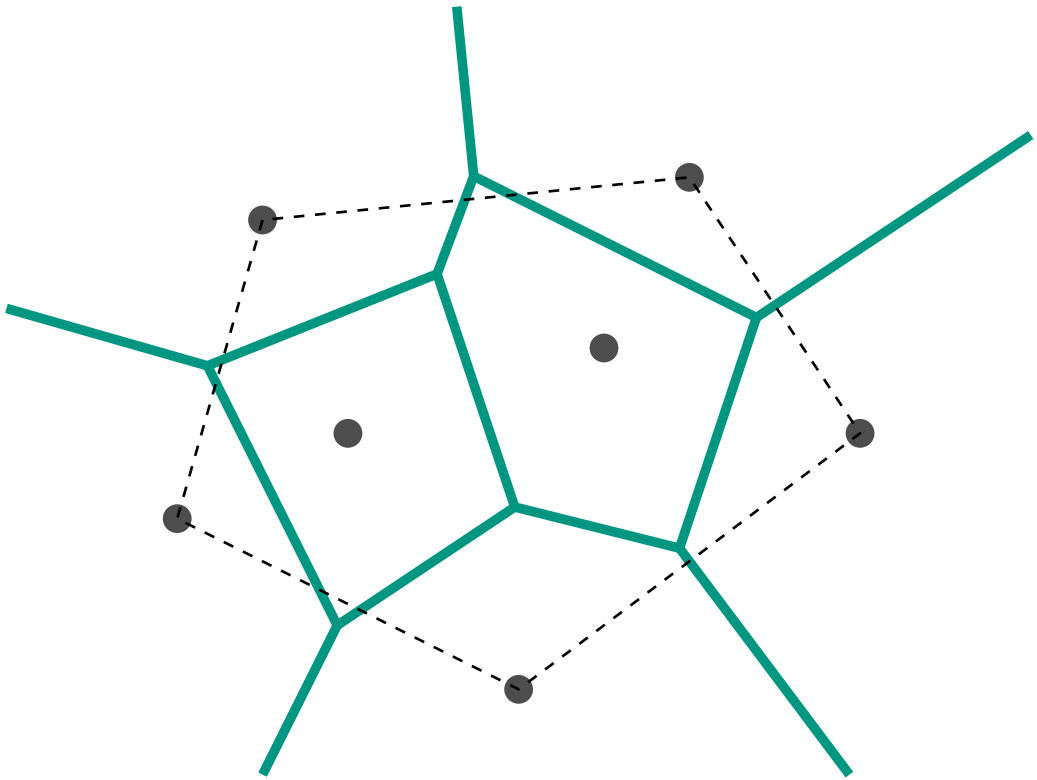
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.



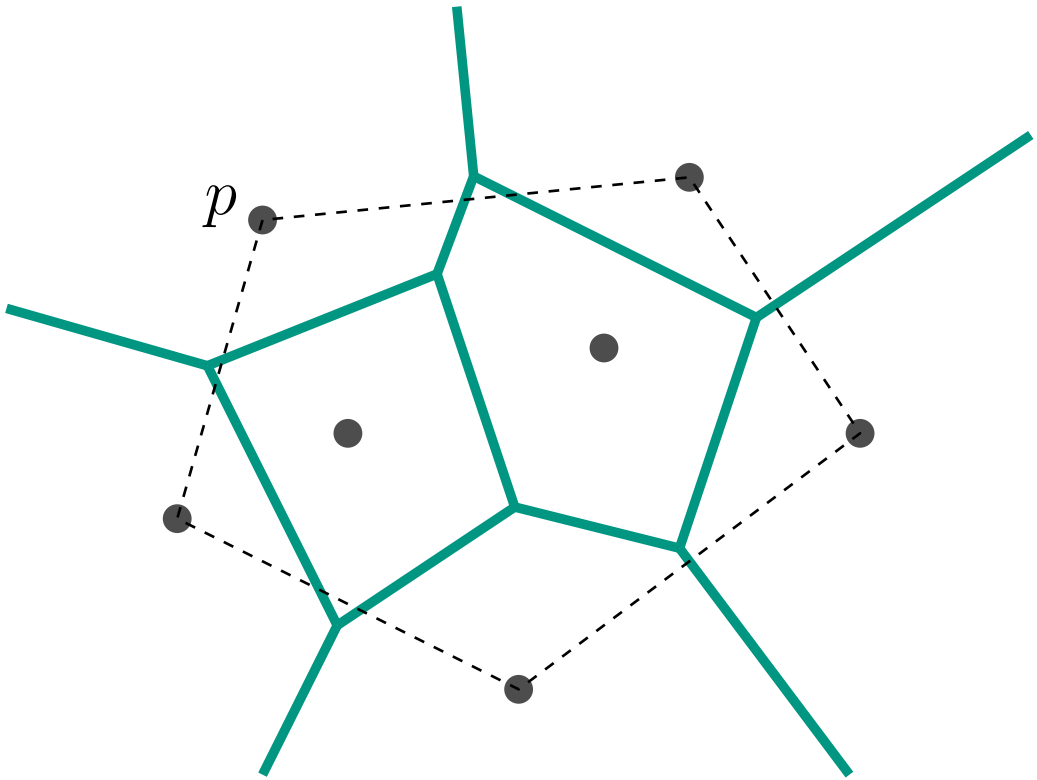
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.



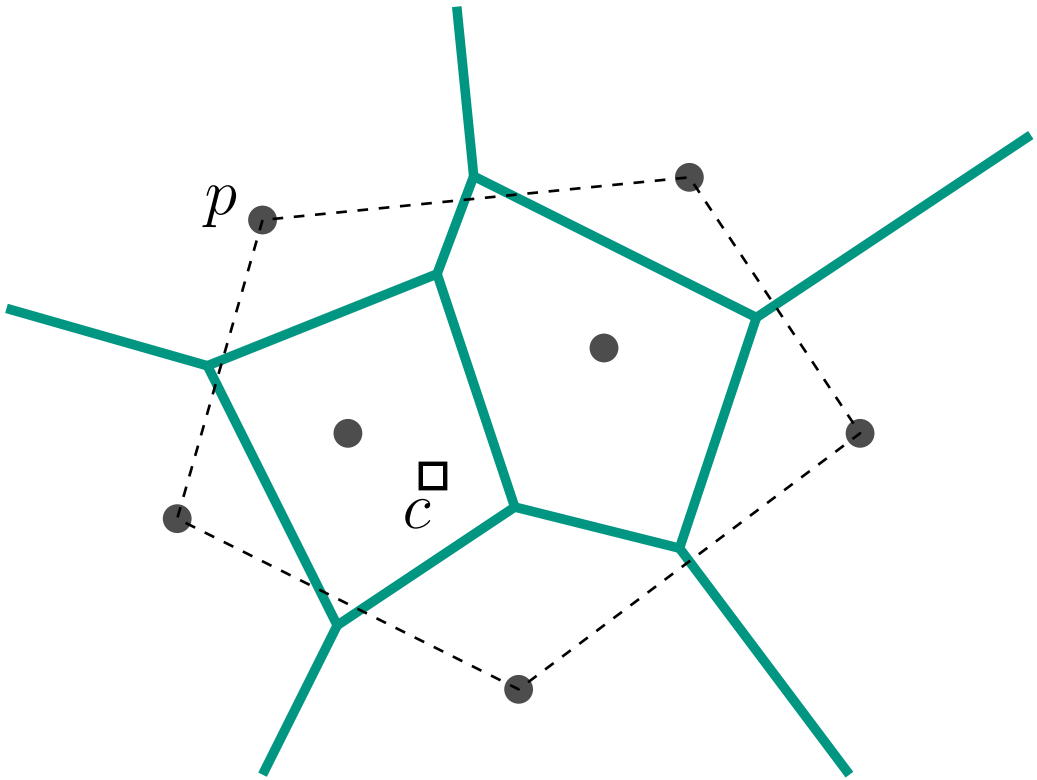
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.



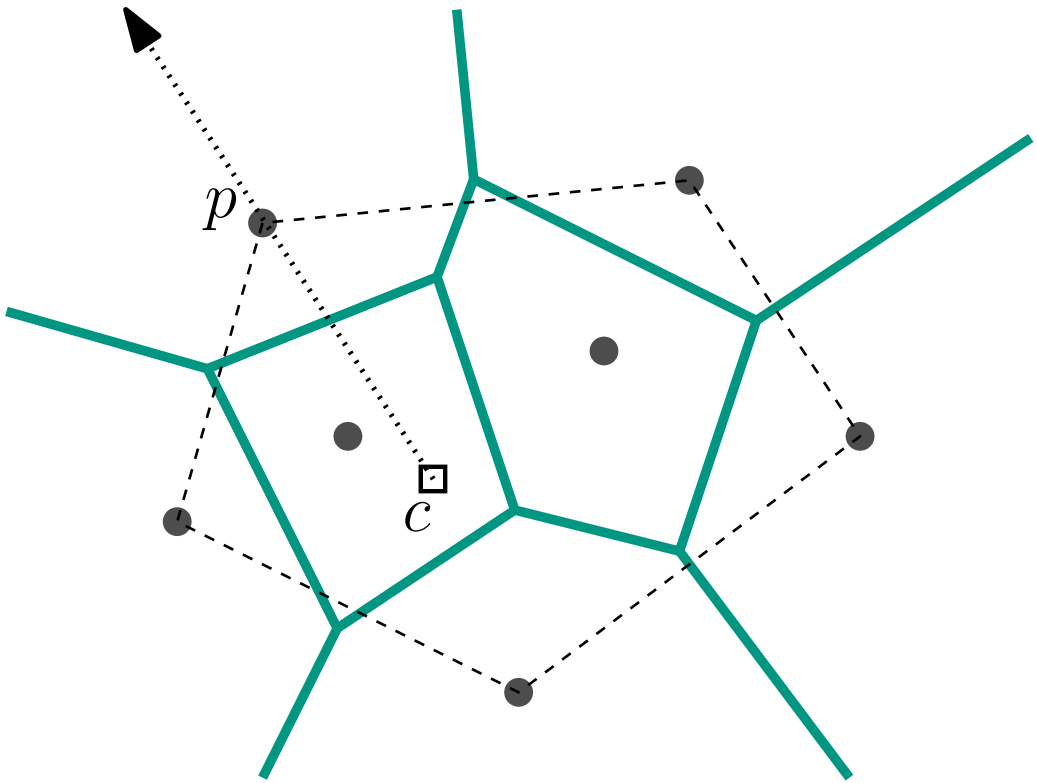
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull



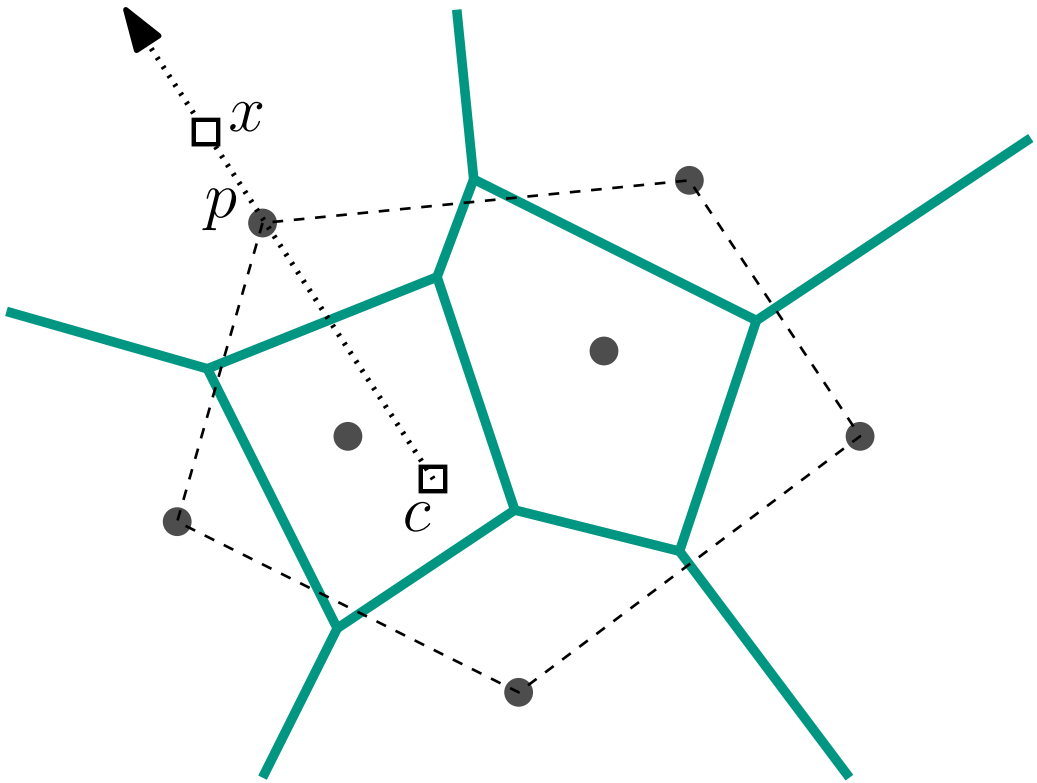
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull
  - Shoot a ray $\overrightarrow{cp}$ from c to p



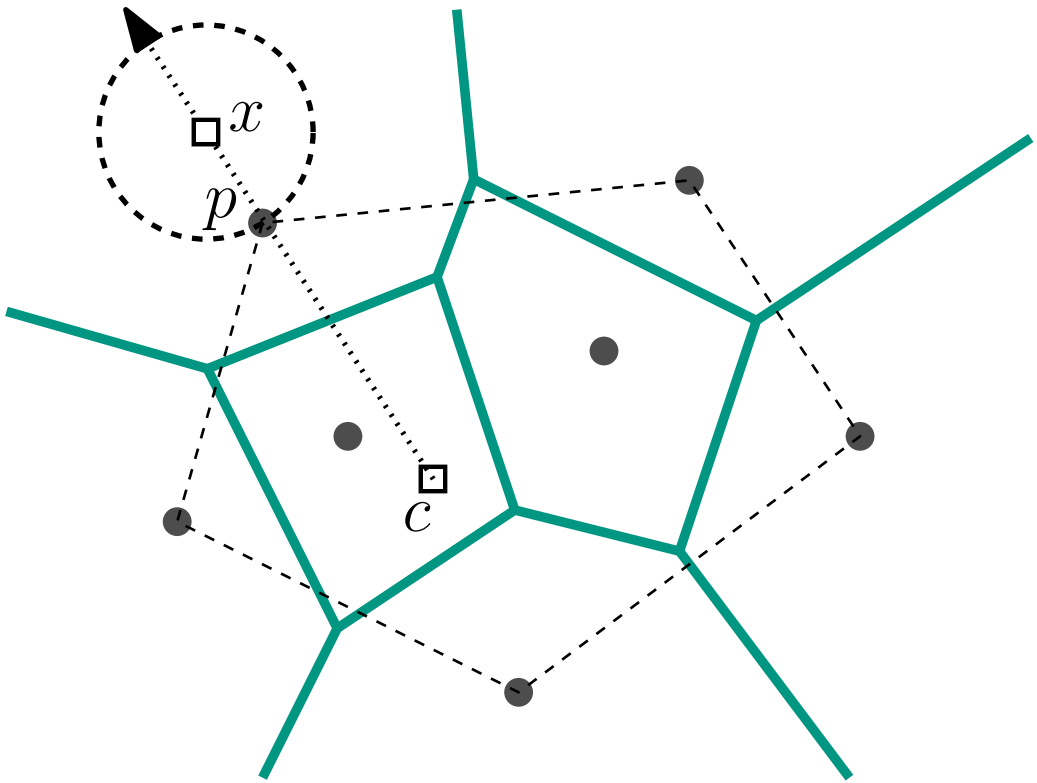
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull
  - Shoot a ray $\overrightarrow{cp}$ from c to p
  - For any point $x \in \overrightarrow{cp} \setminus \overline{cp}$, x belongs to $\mathcal{V}(p)$



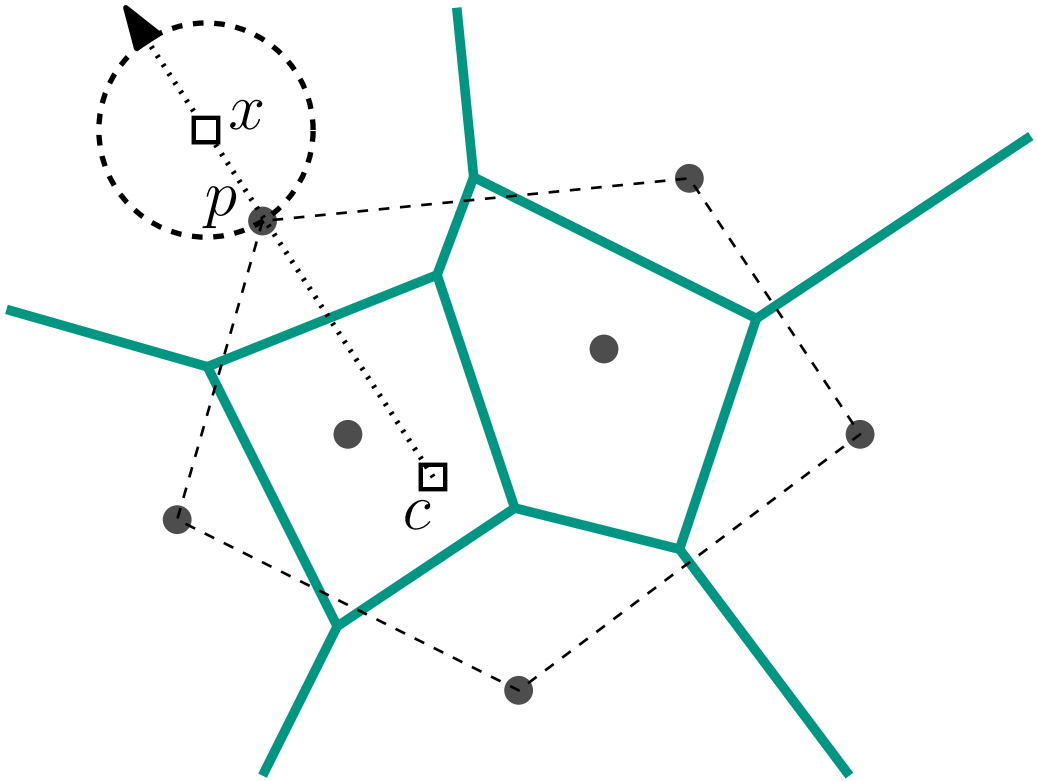
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull
  - Shoot a ray $\overrightarrow{cp}$ from c to p
  - For any point $x \in \overrightarrow{cp} \setminus \overline{cp}$, x belongs to $\mathcal{V}(p)$



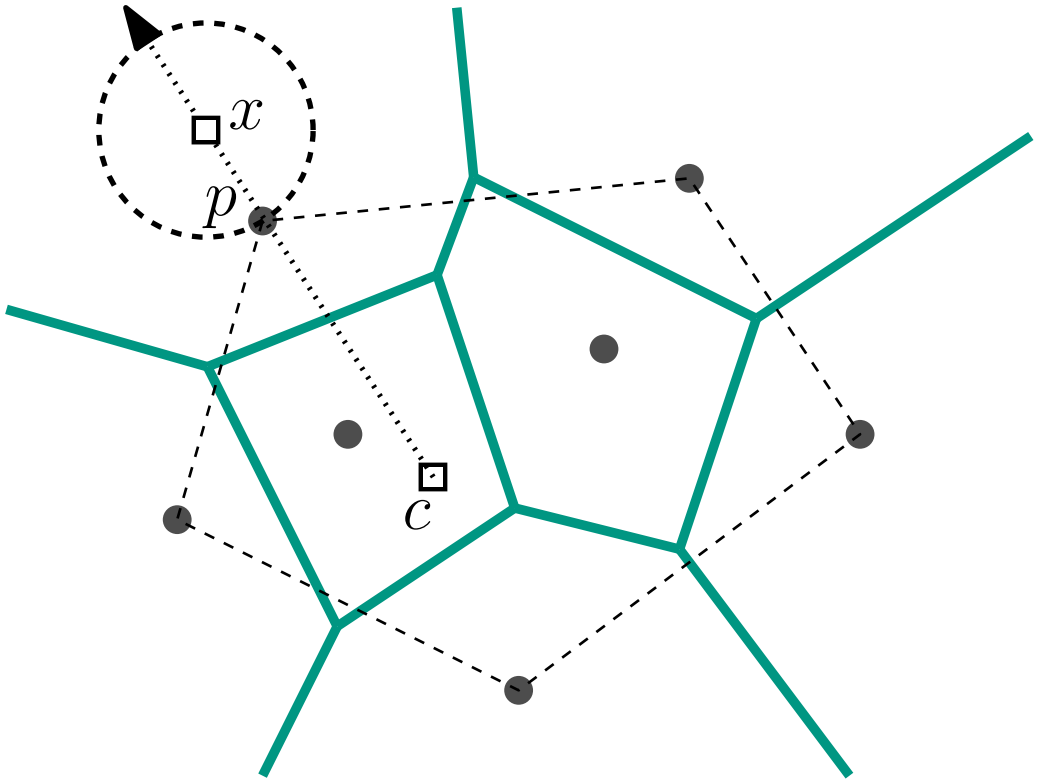
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull
  - Shoot a ray $\overrightarrow{cp}$ from c to p
  - For any point $x \in \overrightarrow{cp} \setminus \overline{cp}$, x belongs to $\mathcal{V}(p)$
  - $\overrightarrow{cp}$ extends to the infinity.



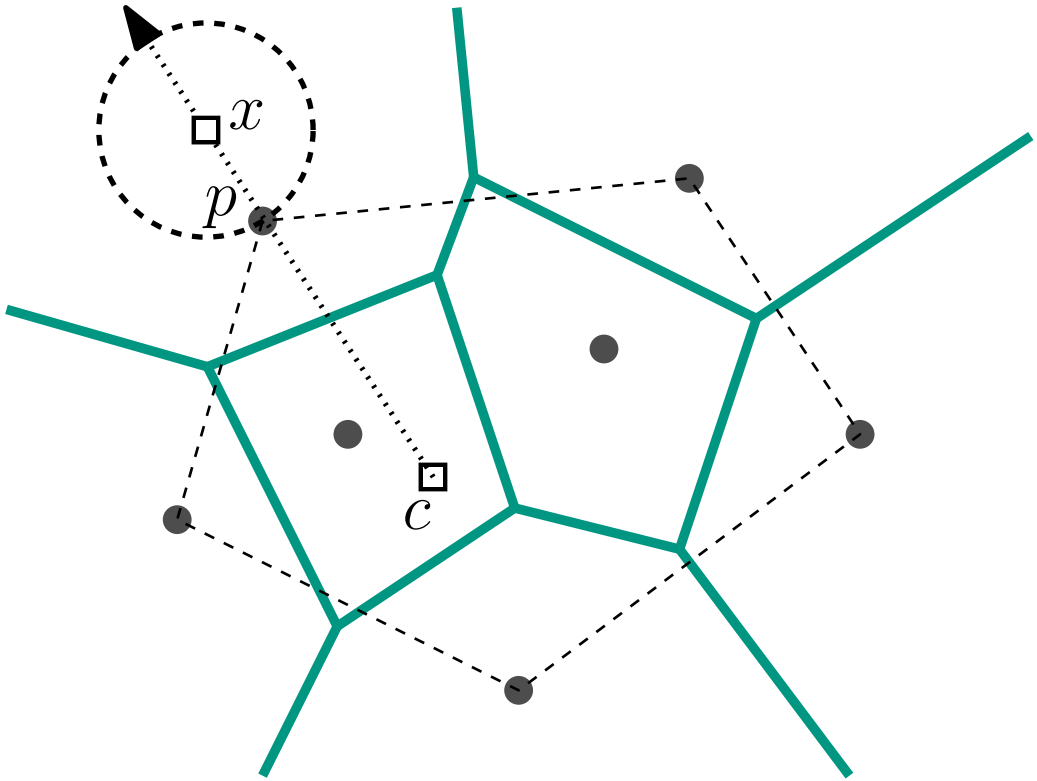
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull
  - Shoot a ray $\overrightarrow{cp}$ from c to p
  - For any point $x \in \overrightarrow{cp} \setminus \overline{cp}$, x belongs to $\mathcal{V}(p)$
  - $\overrightarrow{cp}$ extends to the infinity.
- If P is in convex position, $\mathrm{Vor}(P)$ is a tree.



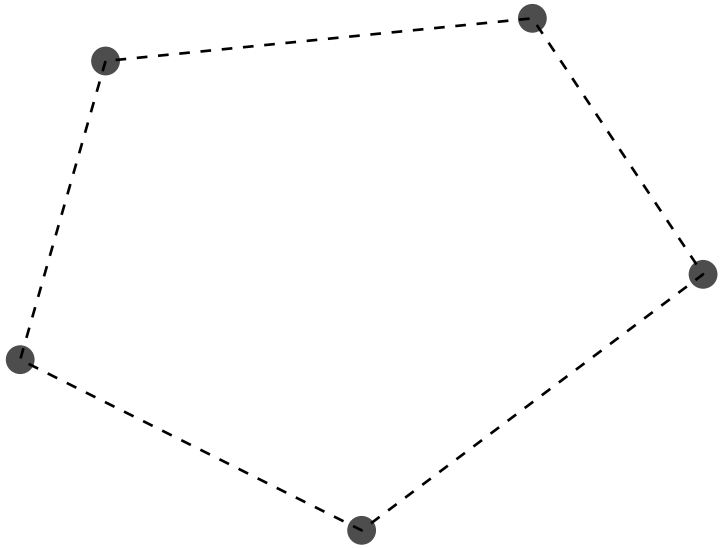
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull
  - Shoot a ray $\overrightarrow{cp}$ from c to p
  - For any point $x \in \overrightarrow{cp} \setminus \overline{cp}$, x belongs to $\mathcal{V}(p)$
  - $\overrightarrow{cp}$ extends to the infinity.
- If P is in convex position, $\mathrm{Vor}(P)$ is a tree.



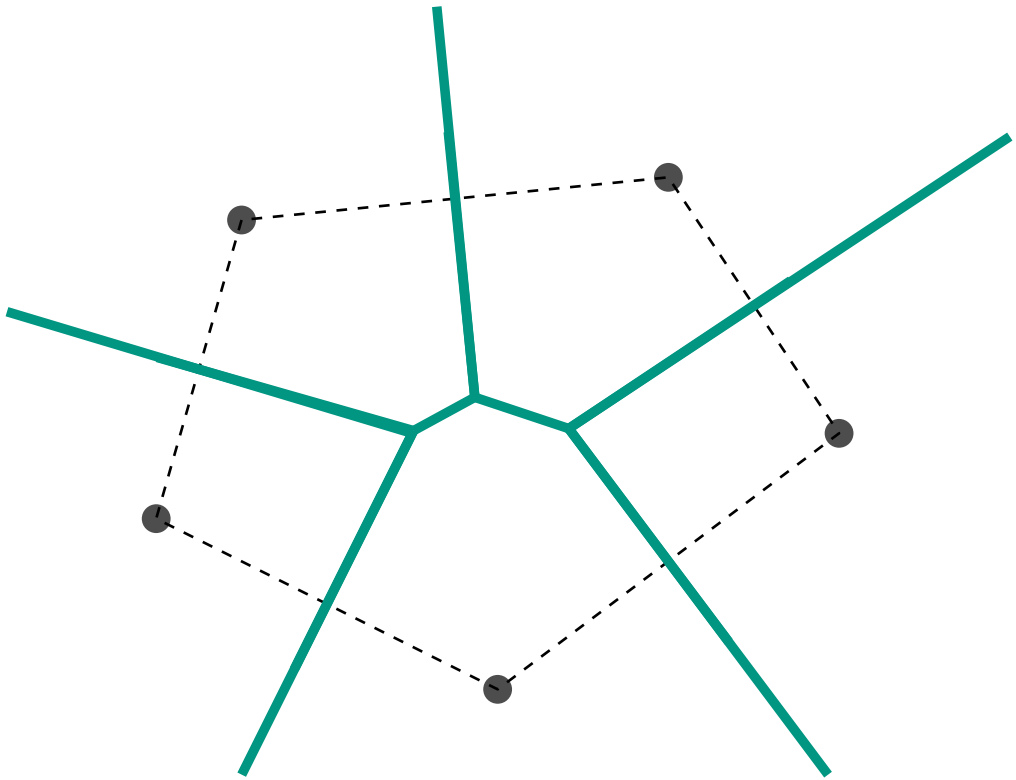
Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull
  - Shoot a ray $\overrightarrow{cp}$ from c to p
  - For any point $x \in \overrightarrow{cp} \setminus \overline{cp}$, x belongs to $\mathcal{V}(p)$
  - $\overrightarrow{cp}$ extends to the infinity.
- If P is in convex position, $\mathrm{Vor}(P)$ is a tree.



Unbounded Cell

- $\mathcal{V}(p)$ is unbounded if p is a vertex of the convex hull of P.
  - Select a point c in the convex hull
  - Shoot a ray $\overrightarrow{cp}$ from c to p
  - For any point $x \in \overrightarrow{cp} \setminus \overline{cp}$, x belongs to $\mathcal{V}(p)$
  - $\overrightarrow{cp}$ extends to the infinity.
- If P is in convex position, Vor(P) is a tree.



Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\mathrm{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\mathrm{Vor}(P)$ is connected and its edges are either segments or half lines.

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\text{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\text{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\text{Vor}(P)$ has?

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\text{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\text{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\text{Vor}(P)$ has?

Find a set P so that a cell in $\text{Vor}(P)$ has linear complexity.

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\text{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\text{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\text{Vor}(P)$ has?

Find a set P so that a cell in $\text{Vor}(P)$ has linear complexity.

Can this happen with (almost) all cells?

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\text{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\text{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\text{Vor}(P)$ has?

Find a set P so that a cell in $\text{Vor}(P)$ has linear complexity.

Can this happen with (almost) all cells?

Theorem 2: Let $P \subset \mathbb{R}^2$ be a set on n points. $\text{Vor}(P)$ has at most $2n-5$ nodes and $3n-6$ edges.

Properties

Theorem 1: Let $P \subset \mathbb{R}^2$ be a set of n points. If all points are collinear, then $\text{Vor}(P)$ consists of $n-1$ parallel lines. Otherwise $\text{Vor}(P)$ is connected and its edges are either segments or half lines.

How many edges and vertices $\text{Vor}(P)$ has?

Find a set P so that a cell in $\text{Vor}(P)$ has linear complexity.

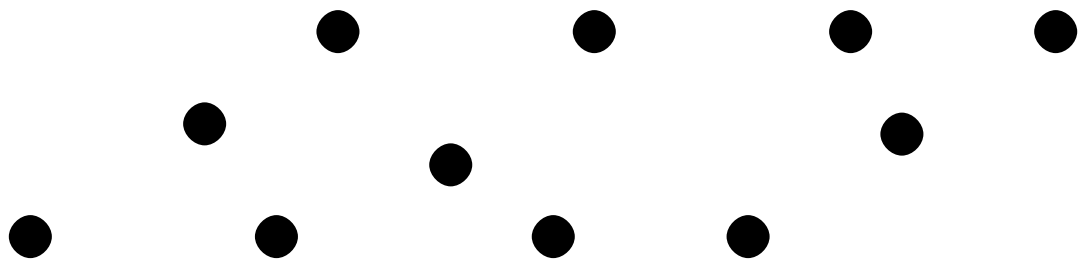
Can this happen with (almost) all cells?

Theorem 2: Let $P \subset \mathbb{R}^2$ be a set on n points. $\text{Vor}(P)$ has at most $2n-5$ nodes and $3n-6$ edges.

Exercise!

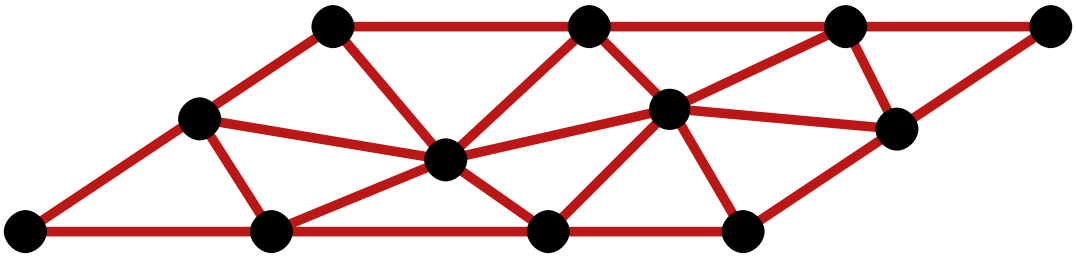
Triangulation

Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.



Triangulation

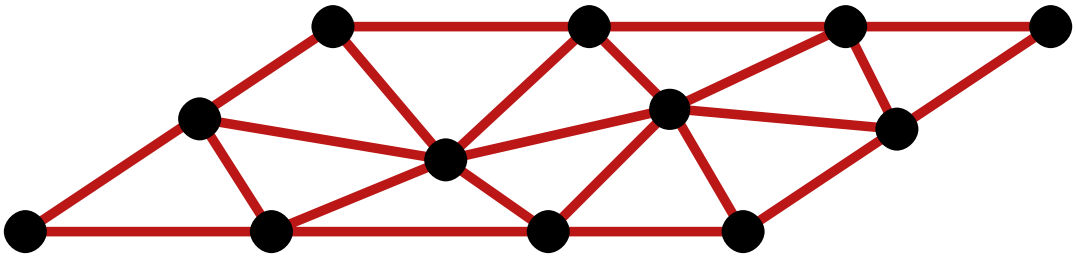
Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.



Obs.:

Triangulation

Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.

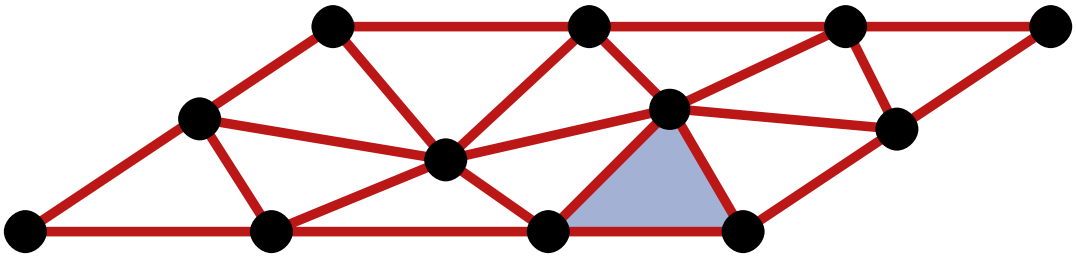


Obs.:

- all internal faces are triangles

Triangulation

Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.

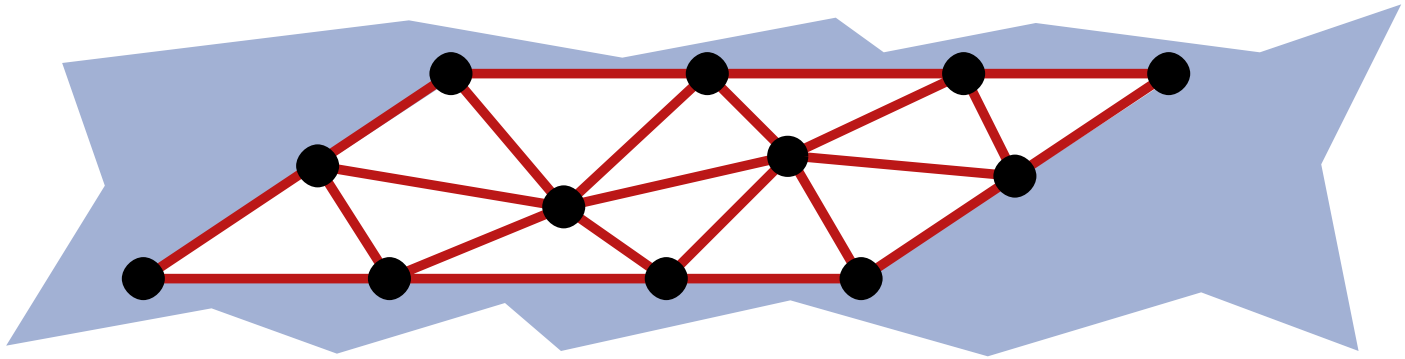


Obs.:

- all internal faces are triangles

Triangulation

Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.

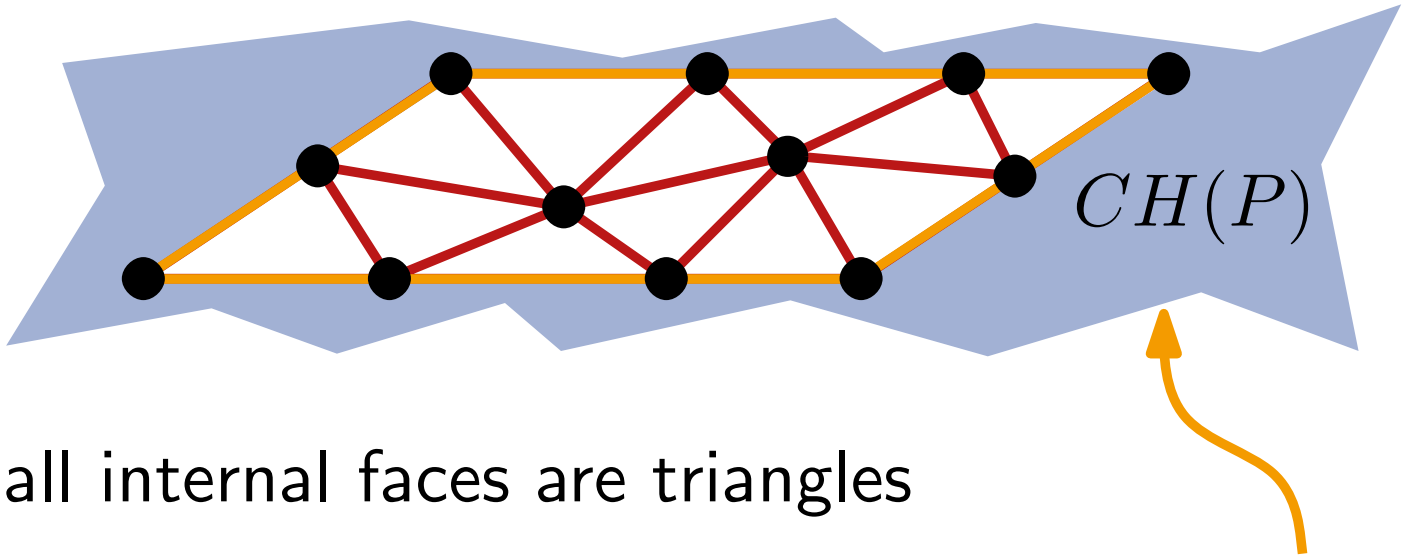


Obs.:

- all internal faces are triangles
- outer face is the complement of the convex hull

Triangulation

Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.

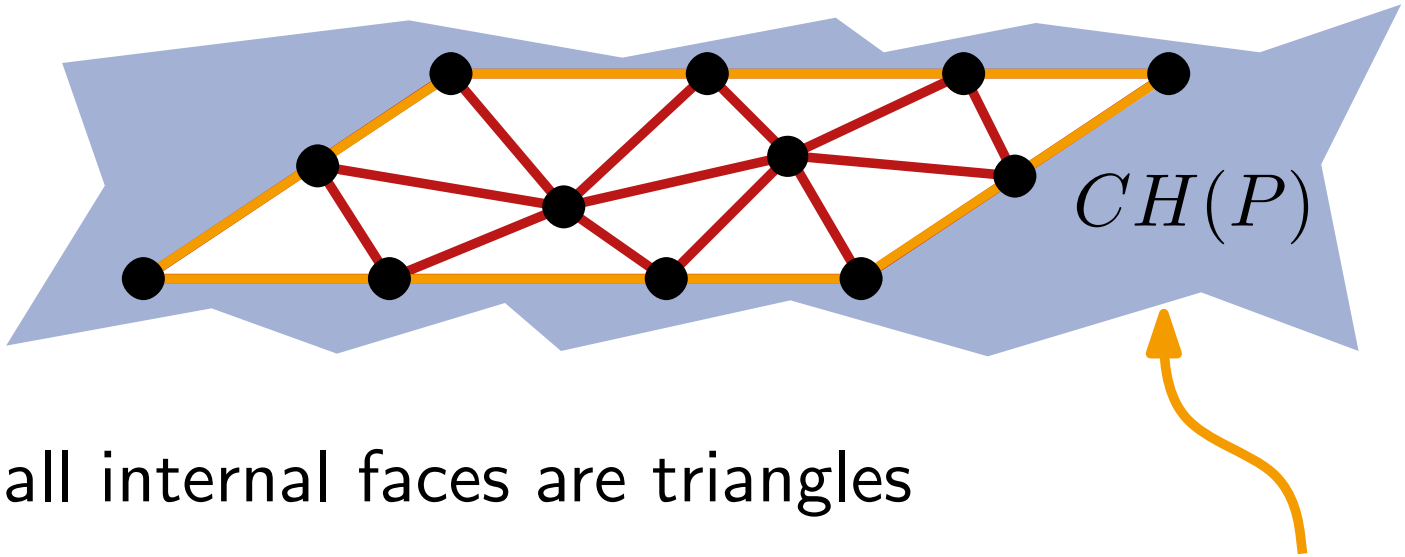


Obs.:

- all internal faces are triangles
- outer face is the complement of the convex hull

Triangulation

Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.



Obs.:

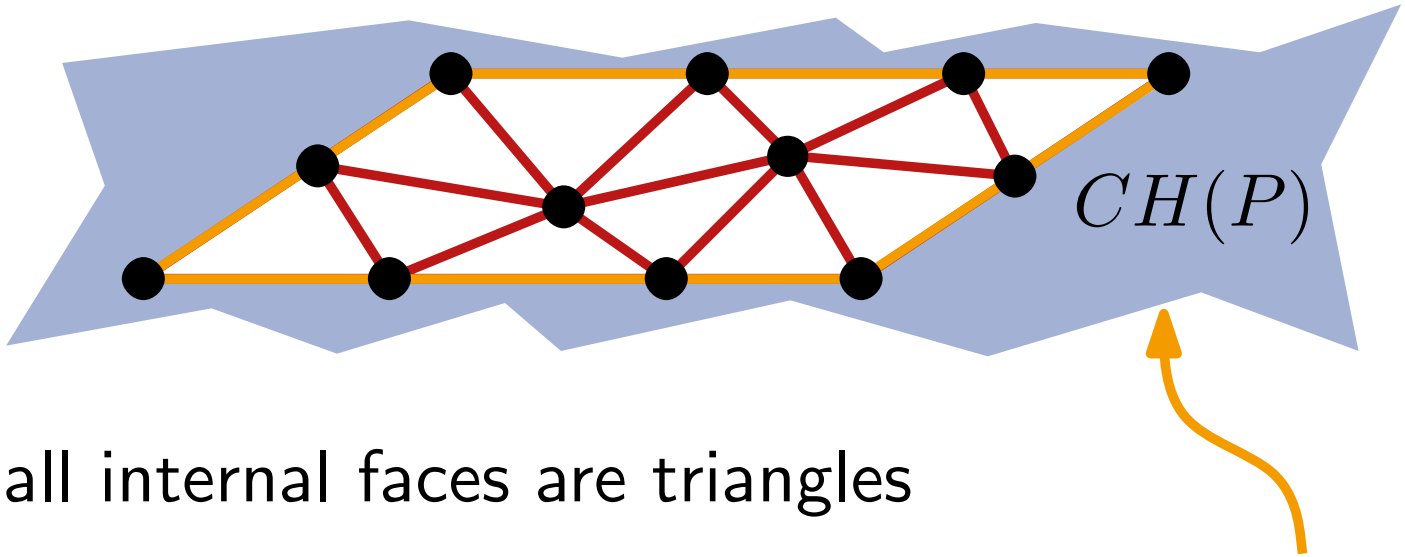
- all internal faces are triangles
- outer face is the complement of the convex hull

Theorem 3: Let P be a set of n points, not all collinear. Let h be the number of points in $CH(P)$.

Then any triangulation of P has $t(n, h)$ triangles and $e(n, h)$ edges.

Triangulation

Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.



Obs.:

- all internal faces are triangles
- outer face is the complement of the convex hull

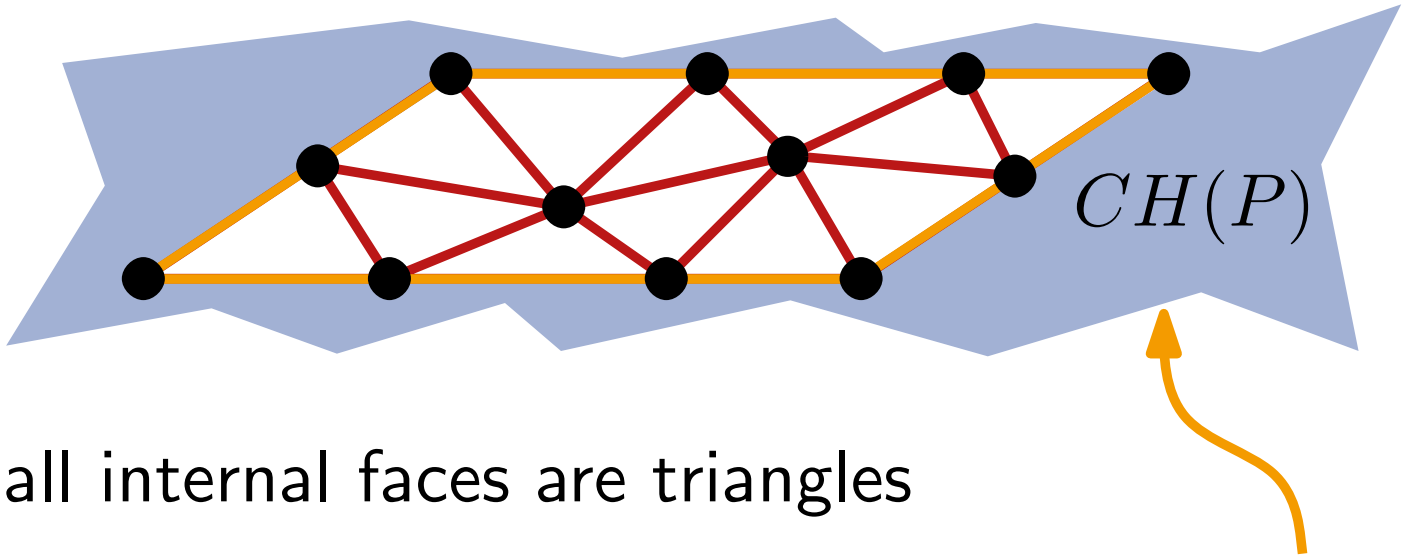
Theorem 3: Let P be a set of n points, not all collinear. Let h be the number of points in $CH(P)$.

Then any triangulation of P has $t(n, h)$ triangles and $e(n, h)$ edges.

Compute t and e!

Triangulation

Def.: A **triangulation** of a point set $P \subset \mathbb{R}^2$ is a maximal planar subdivision with a vertex set P.



Obs.:

- all internal faces are triangles
- outer face is the complement of the convex hull

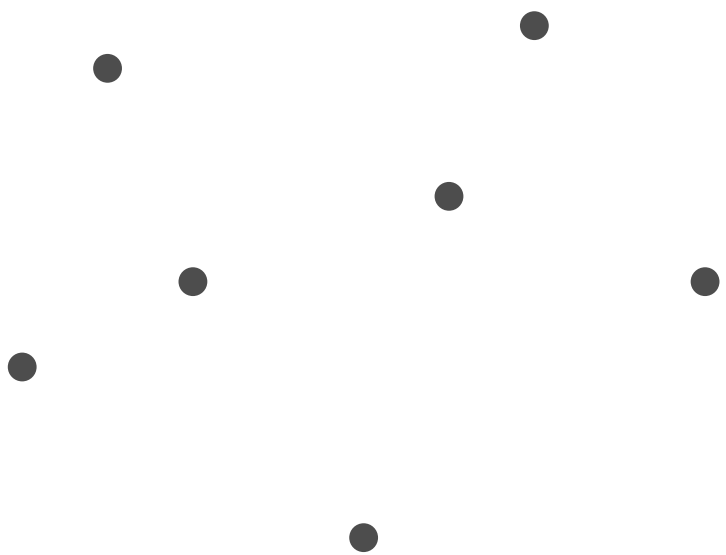
Theorem 3: Let P be a set of n points, not all collinear. Let h be the number of points in $CH(P)$.

Then any triangulation of P has $(2n - 2 - h)$ triangles and $(3n - 3 - h)$ edges.

Delaunay Edge

Def.:

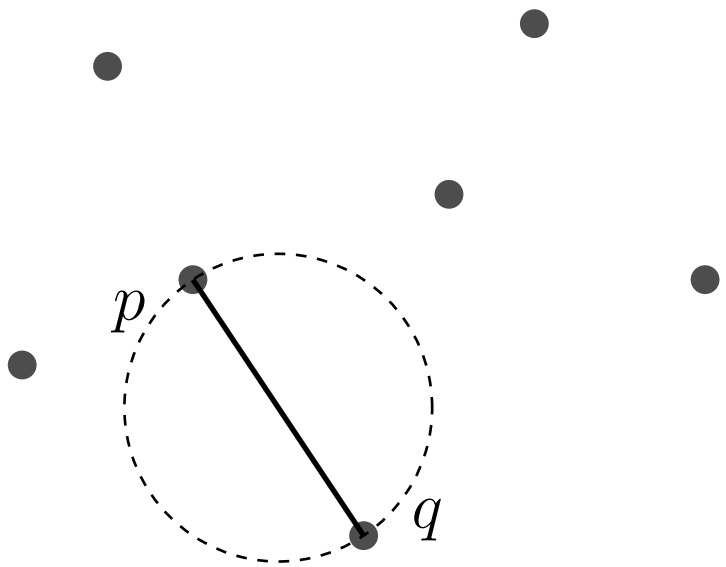
An edge $\overline{pq}$ is called **Delaunay** if there exists a circle passing through p and q and containing no other point in its interior.



Delaunay Edge

Def.:

An edge $\overline{pq}$ is called **Delaunay** if there exists a circle passing through p and q and containing no other point in its interior.

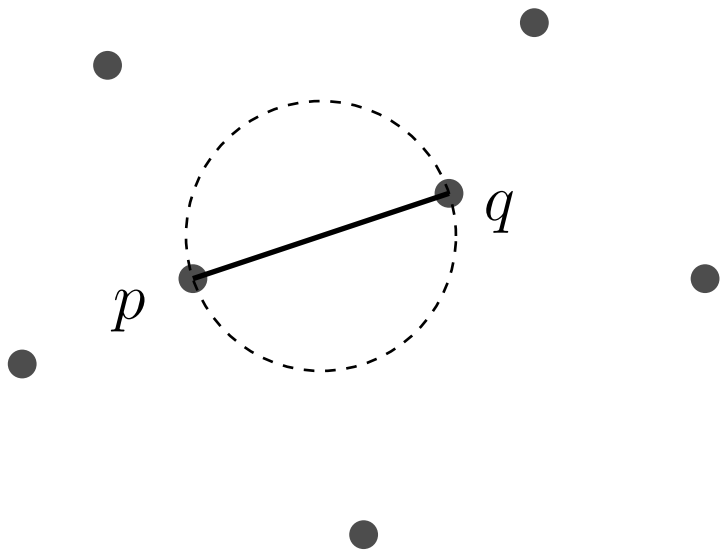


$\overline{pq}$ is **Delaunay**

Delaunay Edge

Def.:

An edge $\overline{pq}$ is called **Delaunay** if there exists a circle passing through p and q and containing no other point in its interior.

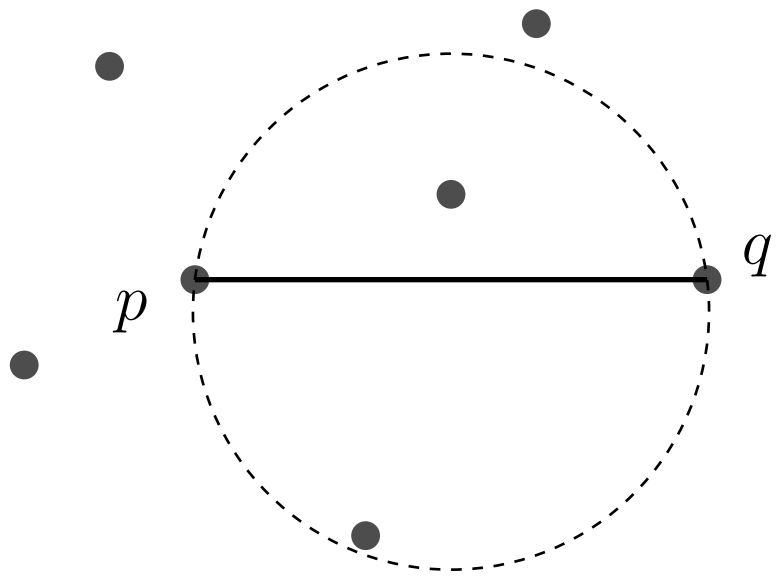


$\overline{pq}$ is **Delaunay**

Delaunay Edge

Def.:

An edge $\overline{pq}$ is called **Delaunay** if there exists a circle passing through p and q and containing no other point in its interior.

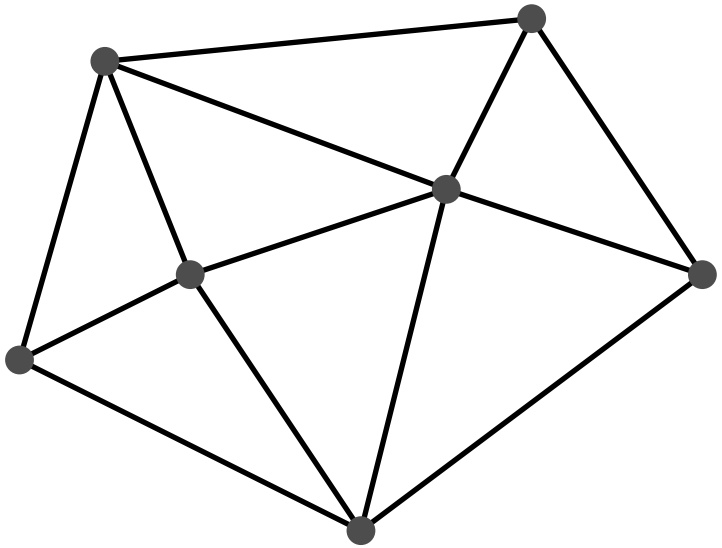


$\overline{pq}$ is **NOT** Delaunay

Delaunay Triangulation

Def.:

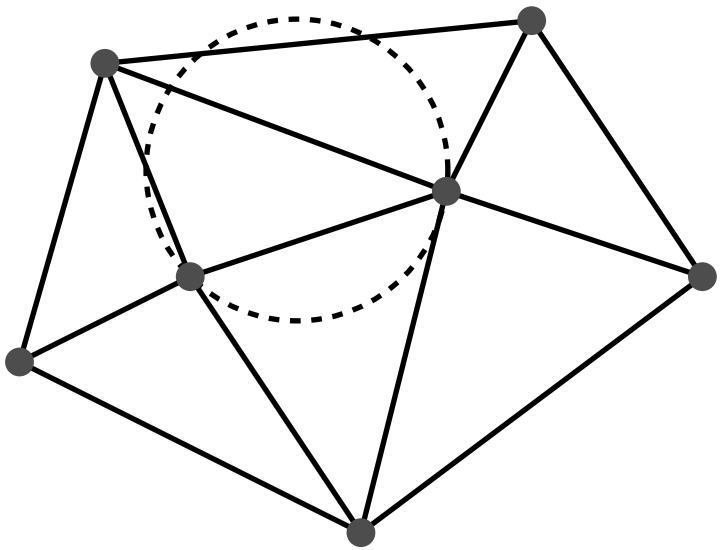
A **Delaunay Triangulation** is a triangulation whose edges are all Delaunay.



Delaunay Triangulation

Def.:

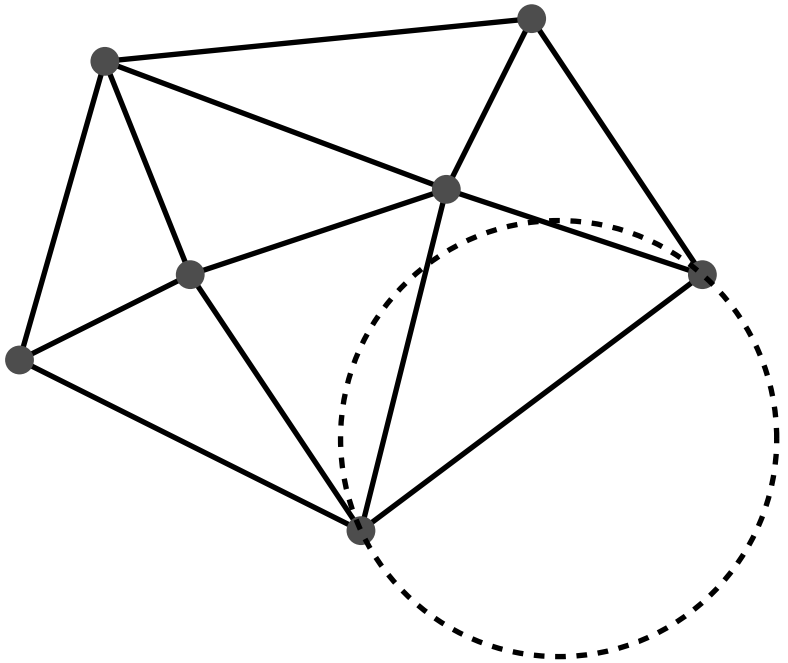
A **Delaunay Triangulation** is a triangulation whose edges are all Delaunay.



Delaunay Triangulation

Def.:

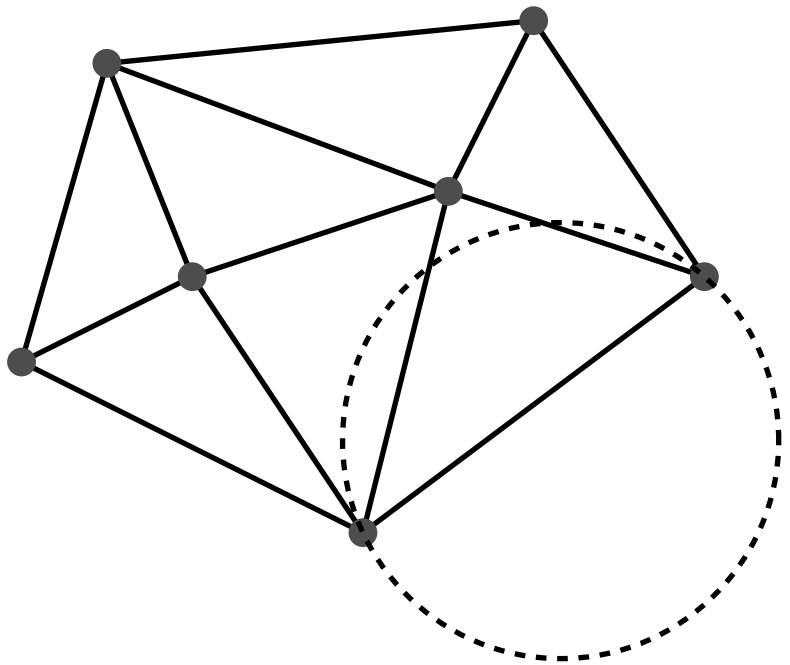
A **Delaunay Triangulation** is a triangulation whose edges are all Delaunay.



Delaunay Triangulation

A **Delaunay Triangulation** is a triangulation whose edges are all Delaunay.

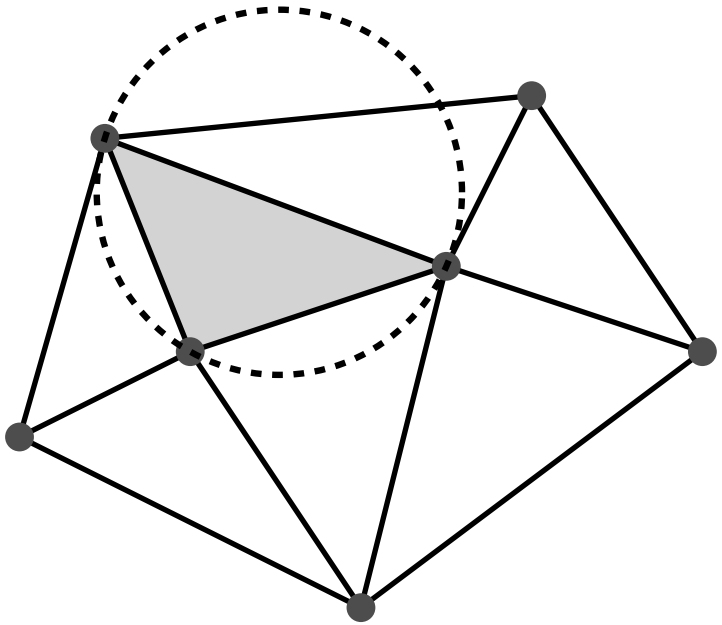
- For each face, there exists a circle passing all its vertices and containing no other point.



Delaunay Triangulation

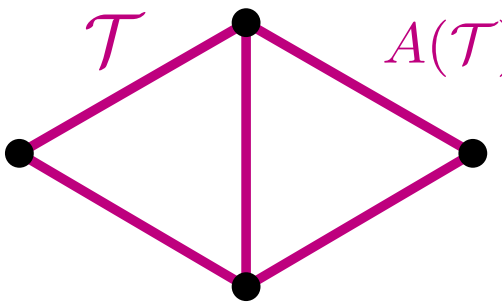
A **Delaunay Triangulation** is a triangulation whose edges are all Delaunay.

- For each face, there exists a circle passing all its vertices and containing no other point.



Angle-optimal Triangulations

Def.: Let $P \subset \mathbb{R}^2$ be a set of points, $\mathcal{T}$ be a triangulation of P and m be the number of the triangles.
$A(\mathcal{T}) = (\alpha_1, \ldots, \alpha_{3m})$ is called **angle-vector** of $\mathcal{T}$ where $\alpha_1 \leq \ldots \leq \alpha_{3m}$.

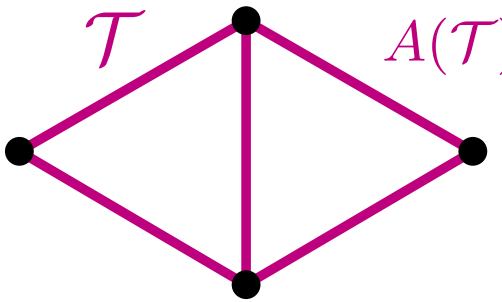


$A(\mathcal{T}) = (60°, 60°, 60°, 60°, 60°, 60°)$

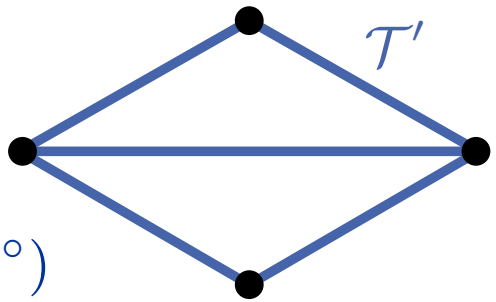
Angle-optimal Triangulations

Def.: Let $P \subset \mathbb{R}^2$ be a set of points, $\mathcal{T}$ be a triangulation of P and m be the number of the triangles.
$A(\mathcal{T}) = (\alpha_1, \ldots, \alpha_{3m})$ is called **angle-vector** of $\mathcal{T}$ where $\alpha_1 \leq \ldots \leq \alpha_{3m}$.

For two triangulations $\mathcal{T}$ and $\mathcal{T}'$ of P we define the **order** of the angle-vectors $A(\mathcal{T}) > A(\mathcal{T}')$ as lexicographic order of corresponding angle sequences.



$A(\mathcal{T}) = (60°, 60°, 60°, 60°, 60°, 60°)$



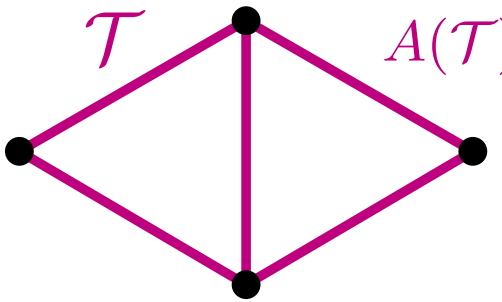
$A(\mathcal{T}') = (30°, 30°, 30°, 30°, 120°, 120°)$

Angle-optimal Triangulations

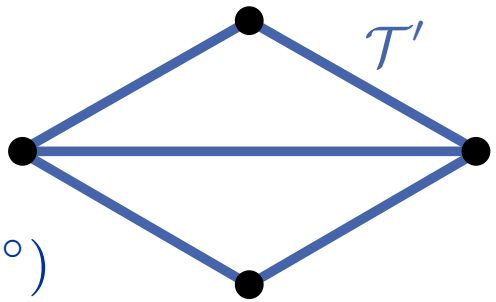
Def.: Let $P \subset \mathbb{R}^2$ be a set of points, $\mathcal{T}$ be a triangulation of P and m be the number of the triangles.
$A(\mathcal{T}) = (\alpha_1, \ldots, \alpha_{3m})$ is called **angle-vector** of $\mathcal{T}$ where $\alpha_1 \leq \ldots \leq \alpha_{3m}$.

For two triangulations $\mathcal{T}$ and $\mathcal{T}'$ of P we define the **order** of the angle-vectors $A(\mathcal{T}) > A(\mathcal{T}')$ as lexicographic order of corresponding angle sequences.

$\mathcal{T}$ is called **angle-optimal**, if $A(\mathcal{T}) \geq A(\mathcal{T}')$ for all triangulations $\mathcal{T}'$ of P.



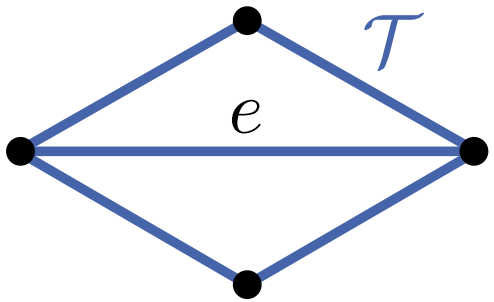
$A(\mathcal{T}) = (60°, 60°, 60°, 60°, 60°, 60°)$



$A(\mathcal{T}') = (30°, 30°, 30°, 30°, 120°, 120°)$

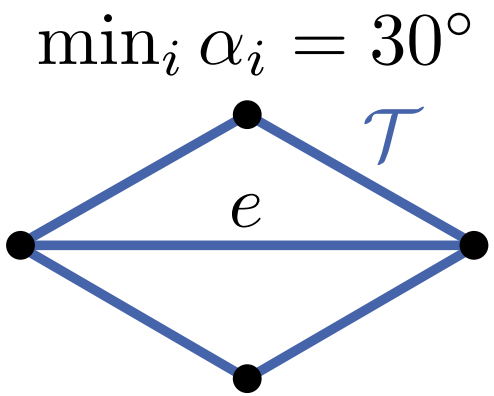
Edge Flips

Def.: Let $\mathcal{T}$ be a triangulation. An edge e of $\mathcal{T}$ is called **illegal**, when the smallest angle incident to e increases after the flip of e.



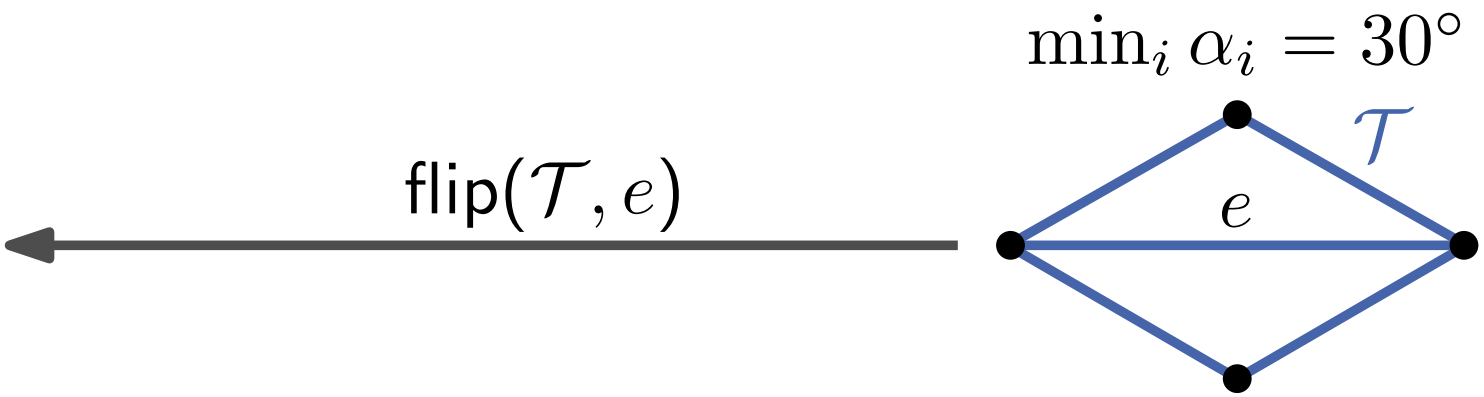
Edge Flips

Def.: Let $\mathcal{T}$ be a triangulation. An edge e of $\mathcal{T}$ is called **illegal**, when the smallest angle incident to e increases after the flip of e.



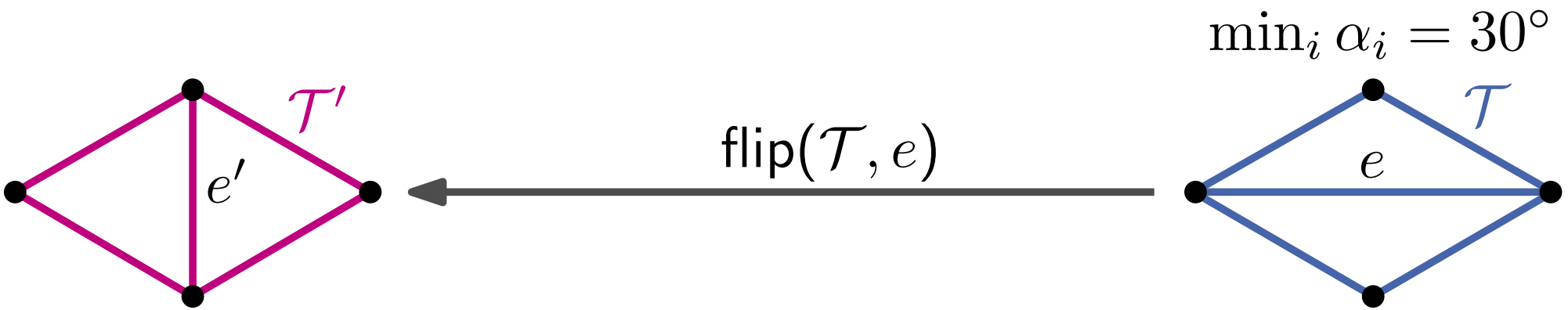
Edge Flips

Def.: Let $\mathcal{T}$ be a triangulation. An edge e of $\mathcal{T}$ is called **illegal**, when the smallest angle incident to e increases after the flip of e.



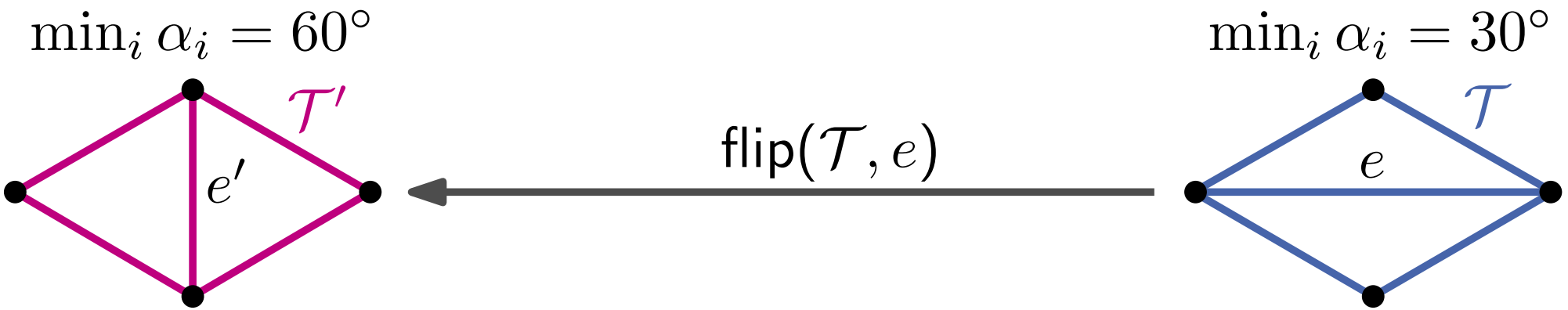
Edge Flips

Def.: Let $\mathcal{T}$ be a triangulation. An edge e of $\mathcal{T}$ is called **illegal**, when the smallest angle incident to e increases after the flip of e.



Edge Flips

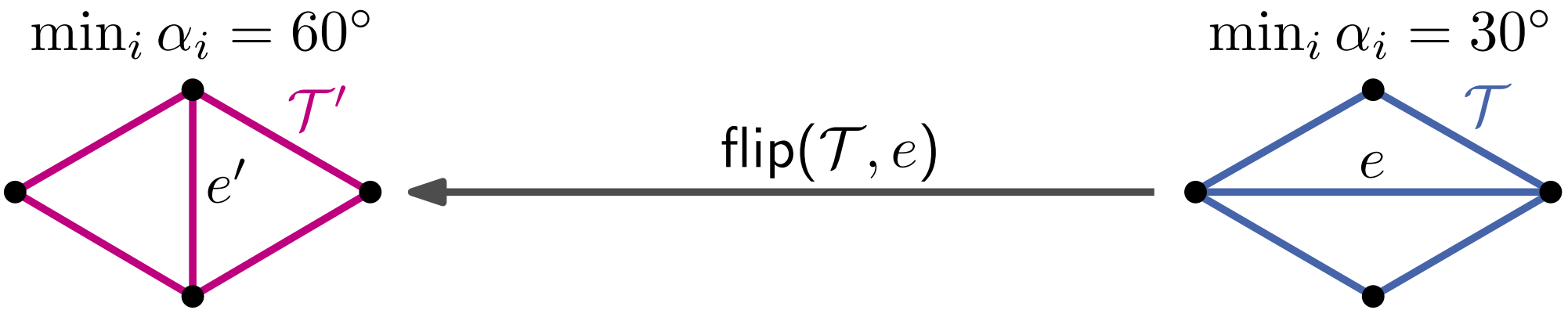
Def.: Let $\mathcal{T}$ be a triangulation. An edge e of $\mathcal{T}$ is called **illegal**, when the smallest angle incident to e increases after the flip of e.



Edge Flips

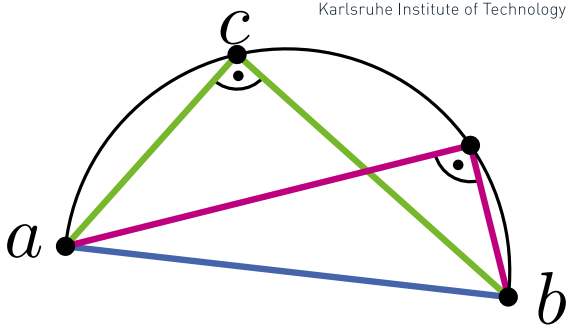
Def.: Let $\mathcal{T}$ be a triangulation. An edge e of $\mathcal{T}$ is called **illegal**, when the smallest angle incident to e increases after the flip of e.

Obs.: Let e be an illegal edge of $\mathcal{T}$ and $\mathcal{T}' = \text{flip}(\mathcal{T}, e)$. Then $A(\mathcal{T}') > A(\mathcal{T})$.



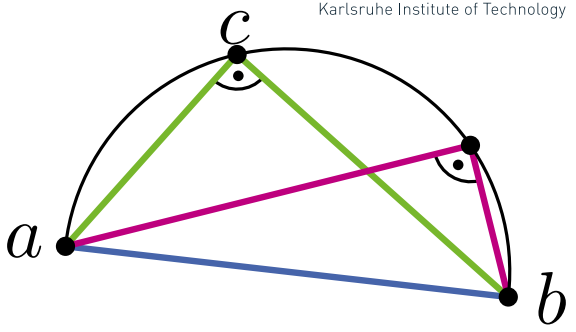
Thales's Theorem

Theorem 4: If a, b and c are points on a circle where the segment ab is a diameter of the circle, then the angle $\angle bca$ is a right angle.

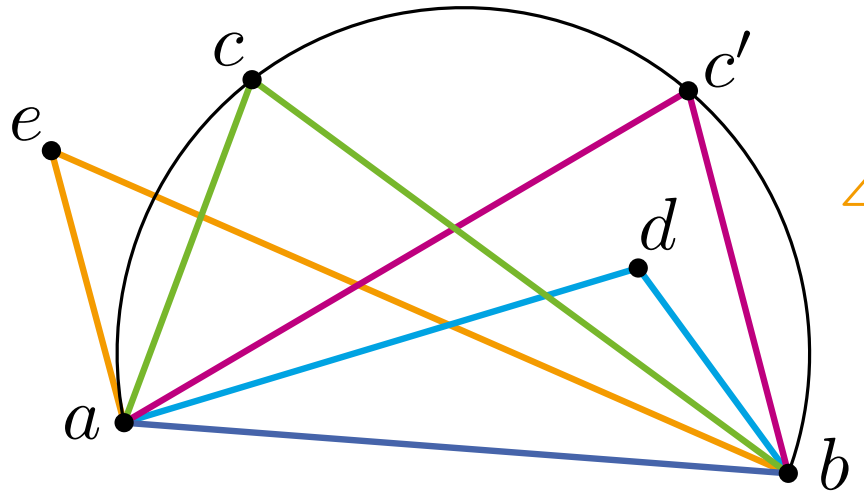


Thales's Theorem

Theorem 4: If a, b and c are points on a circle where the segment ab is a diameter of the circle, then the angle $\angle bca$ is a right angle.



Theorem 4′: Consider a circle C through a, b, c. For any point c' on C on the same side of ab as c , holds that $\angle acb = \angle ac'b$. For any point d inside C holds that $\angle adb > \angle acb$, and for point e outside C, holds that $\angle aeb < \angle acb$.

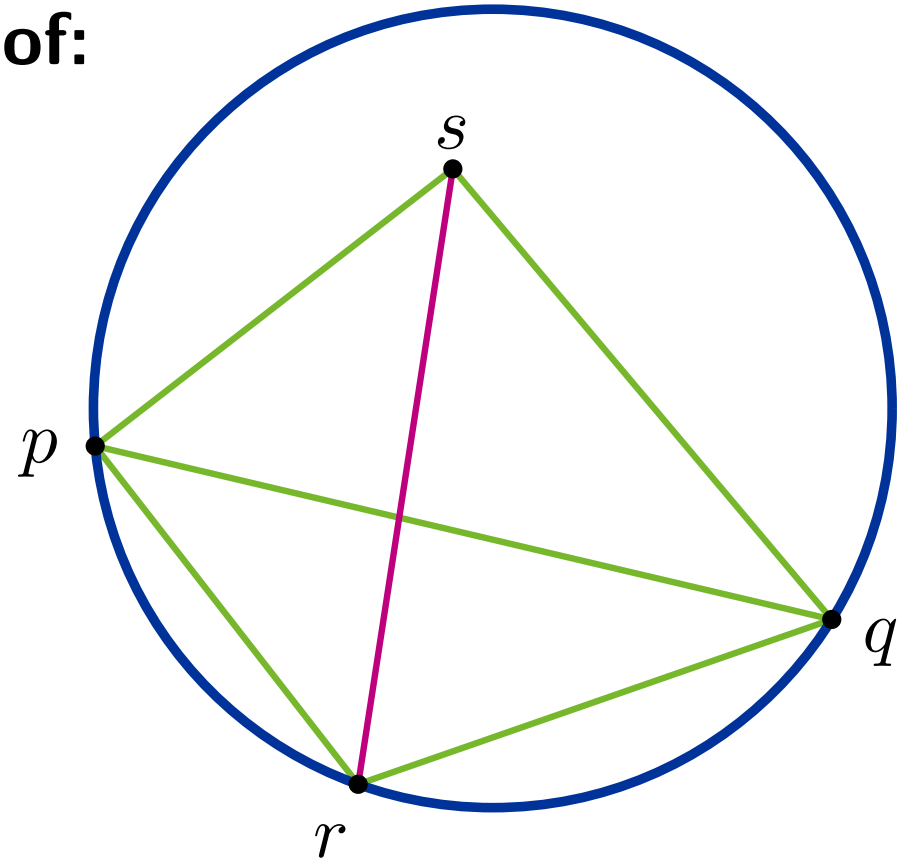


$$\angle aeb < \angle acb = \angle ac'b < \angle adb$$

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$. If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Sketch of proof:

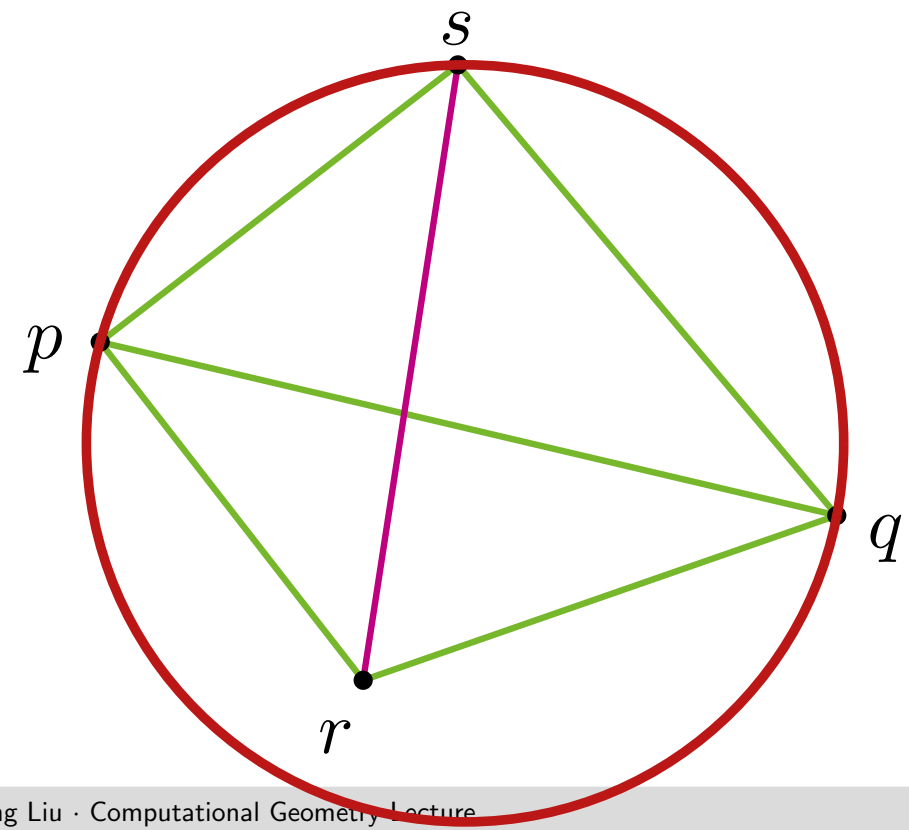


$\angle rsq > \angle rpq$
$\angle psr > \angle pqr$

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$. If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Sketch of proof:



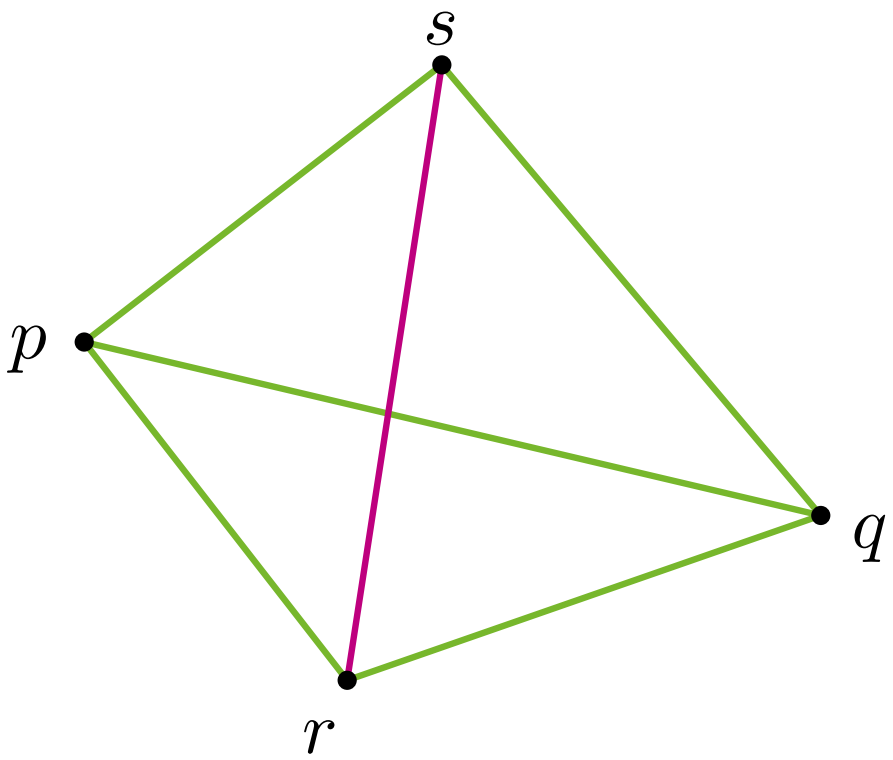
$\angle rsq > \angle rpq$
$\angle psr > \angle pqr$

$\angle prs > \angle pqs$
$\angle srq > \angle spq$

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$. If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Sketch of proof:



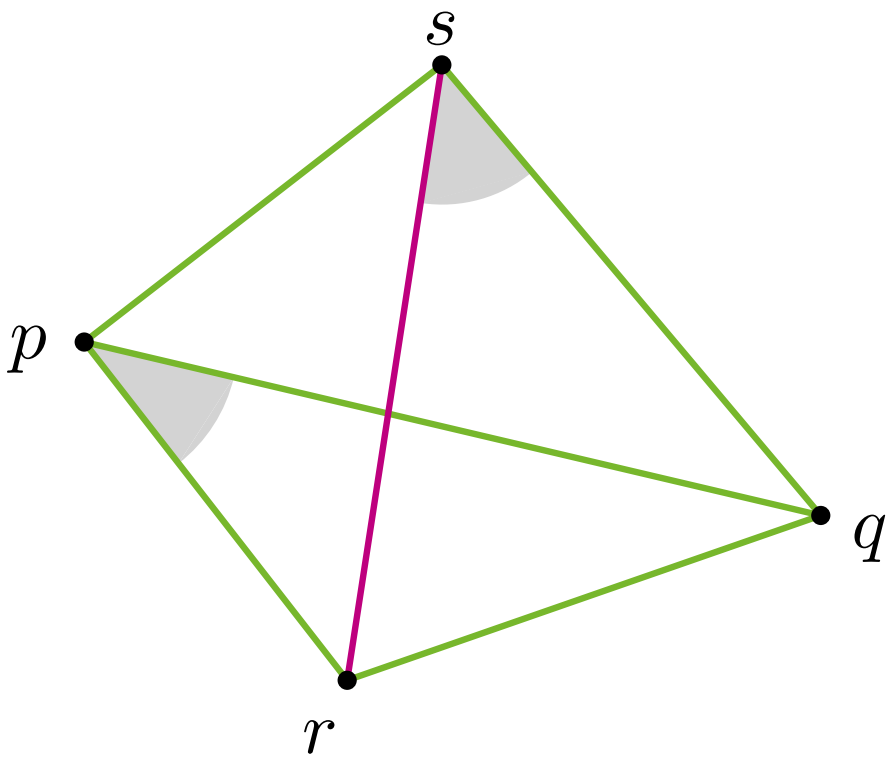
$\angle rsq > \angle rpq$
$\angle psr > \angle pqr$

$\angle prs > \angle pqs$
$\angle srq > \angle spq$

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$. If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Sketch of proof:



$\angle rsq > \angle rpq$
$\angle psr > \angle pqr$

$\angle prs > \angle pqs$
$\angle srq > \angle spq$

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$.
If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Obs.:

- The characterization is symmetric w.r.t. r and s
- $s \in \partial C \Rightarrow \overline{pq}$ and $\overline{rs}$ are legal
- an illegal edge $\Rightarrow$ quadrilateral is convex

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$.
If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Obs.:
- The characterization is symmetric w.r.t. r and s
- $s \in \partial C \Rightarrow \overline{pq}$ and $\overline{rs}$ are legal
- an illegal edge $\Rightarrow$ quadrilateral is convex

Def.: A triangulation without illegal edges in called **legal**.

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$.
If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Obs.:
- The characterization is symmetric w.r.t. r and s
- $s \in \partial C \Rightarrow \overline{pq}$ and $\overline{rs}$ are legal
- an illegal edge $\Rightarrow$ quadrilateral is convex

Def.: A triangulation without illegal edges in called **legal**.

Is there always a legal triangulation?

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$.
If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Obs.:
- The characterization is symmetric w.r.t. r and s
- $s \in \partial C \Rightarrow \overline{pq}$ and $\overline{rs}$ are legal
- an illegal edge $\Rightarrow$ quadrilateral is convex

Def.: A triangulation without illegal edges in called **legal**.

while $\mathcal{T}$ has an illegal edge e **do**
  flip$(\mathcal{T}, e)$
return $\mathcal{T}$

Legal Triangulation

Lemma 5: Let Δpqr and Δpqs be two triangles in $\mathcal{T}$ and let C be the circle through Δpqr. Then $\overline{pq}$ is illegal iff $s \in \text{int}(C)$. If p, q, r, s form a convex quadrilateral and do not lie on a common circle ($s \notin \partial C$) then exactly one of $\overline{pq}$ and $\overline{rs}$ is an illegal edge.

Obs.:
- The characterization is symmetric w.r.t. r and s
- $s \in \partial C \Rightarrow \overline{pq}$ and $\overline{rs}$ are legal
- an illegal edge $\Rightarrow$ quadrilateral is convex

Def.: A triangulation without illegal edges in called **legal**.

while $\mathcal{T}$ has an illegal edge e **do**
  flip($\mathcal{T}, e$)
return $\mathcal{T}$

terminates, since $A(\mathcal{T})$ increases and #Triangulations is finite ($< 30^n$, [Sharir, Sheffer 2011])

Reverse statement?

It holds that: each angle-optimal triangulation is legal.

But is each legal triangulation also angle-optimal?

Legality and Delaunay-Triangulation

Karlsruhe Institute of Technology

Theorem 6: Let P be a set of points and $\mathcal{T}$ a triangulation of P. $\mathcal{T}$ is legal $\Leftrightarrow$ $\mathcal{T}$ is Delaunay-Triangulation.

Sketch of proof:

„$\Leftarrow$“ clear; use

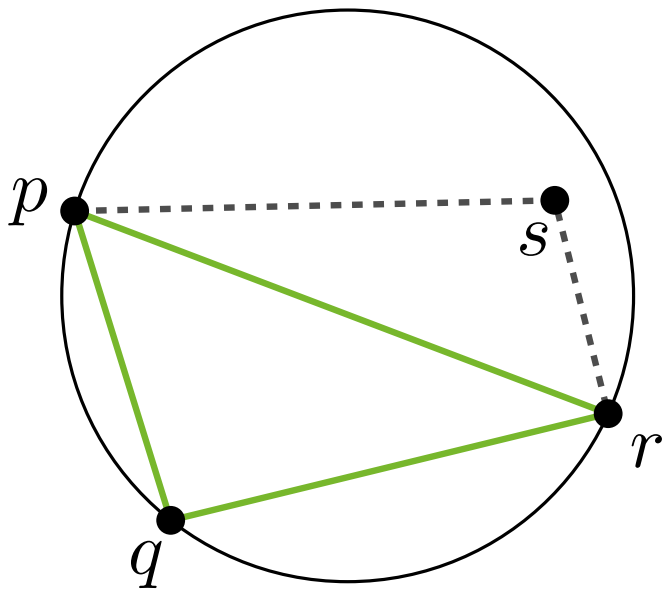
Lemma 5: Let Δprq and Δpqs be two triangles of $\mathcal{T}$ and C the circumcircle of Δprq. Edge $\overline{pq}$ is illegal iff $s \in \text{int}(C)$.

Legality and Delaunay-Triangulation

Theorem 6: Let P be a set of points and $\mathcal{T}$ a triangulation of P. $\mathcal{T}$ is legal $\Leftrightarrow$ $\mathcal{T}$ is Delaunay-Triangulation.

Sketch of proof:

„$\Rightarrow$“

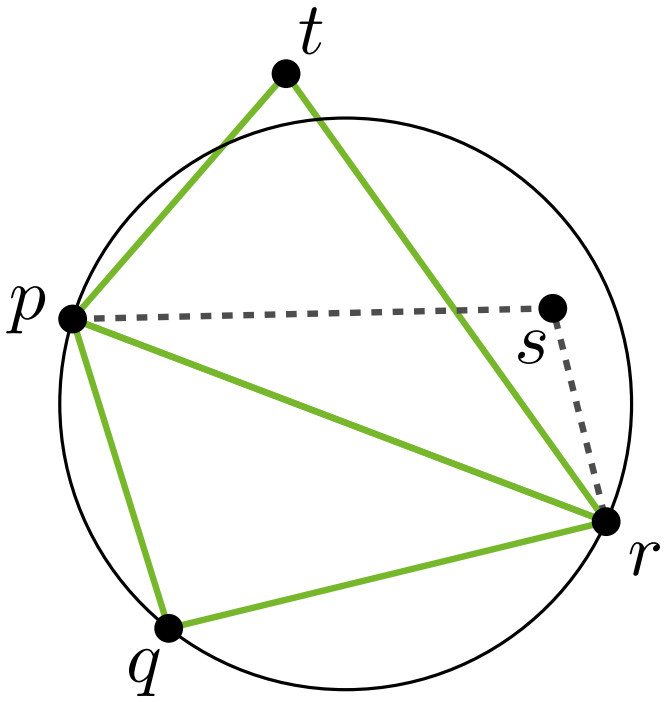


Legality and Delaunay-Triangulation

Theorem 6: Let P be a set of points and $\mathcal{T}$ a triangulation of P. $\mathcal{T}$ is legal $\Leftrightarrow$ $\mathcal{T}$ is Delaunay-Triangulation.

Sketch of proof:

„$\Rightarrow$“

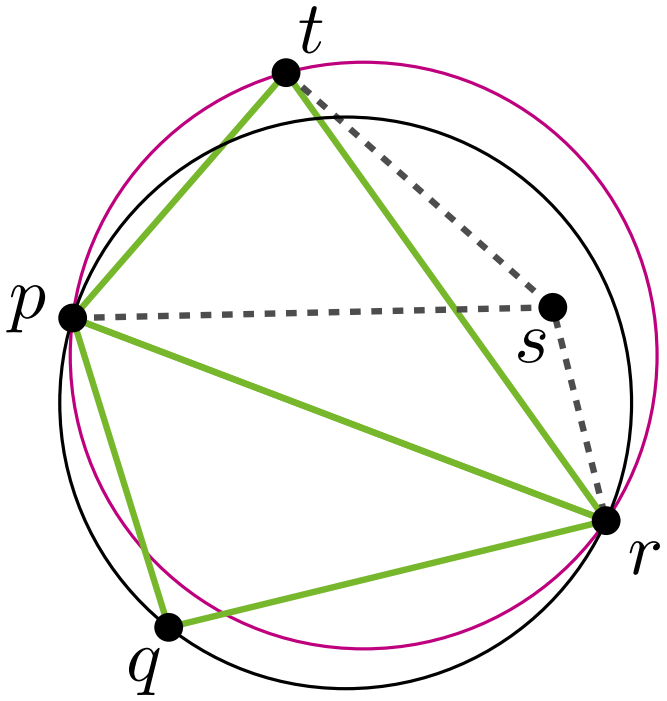


Legality and Delaunay-Triangulation

Theorem 6: Let P be a set of points and $\mathcal{T}$ a triangulation of P. $\mathcal{T}$ is legal $\Leftrightarrow$ $\mathcal{T}$ is Delaunay-Triangulation.

Sketch of proof:

„$\Rightarrow$“



General Position Assumption

- No more than two point sites are colinear

General Position Assumption

- No more than two point sites are colinear
  - $\mathrm{Vor}(P)$ is **connected**

General Position Assumption

- No more than two point sites are colinear
  - Vor(P) is **connected**

- No more than three point sites are cocircular
  (At most three points are on the same circle)

General Position Assumption

- No more than two point sites are colinear
  - Vor(P) is **connected**

- No more than three point sites are cocircular
  (At most three points are on the same circle)
  - Degree of each Voronoi vertex is exactly **3**.

General Position Assumption

- No more than two point sites are colinear
  - Vor(P) is **connected**
- No more than three point sites are cocircular
  (At most three points are on the same circle)
  - Degree of each Voronoi vertex is exactly **3**.
- Delaunay Triangulation is unique
  - Angle-Optimal!!

Duality

Theorem 7:

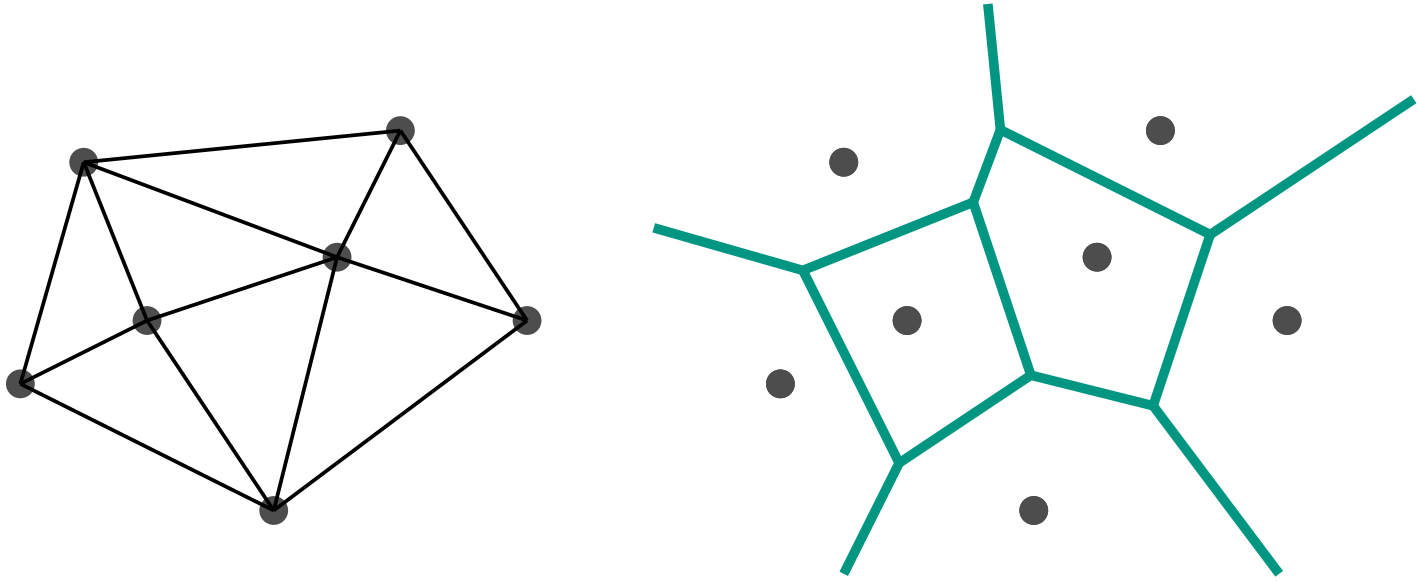
Under the general position assumption, the Delaunay triangulation is a dual graph of the Voronoi diagram.

Duality

Theorem 7:

Under the general position assumption, the Delaunay triangulation is a dual graph of the Voronoi diagram.

- A site $p \leftrightarrow$ a Voronoi cell $\mathcal{V}(p)$

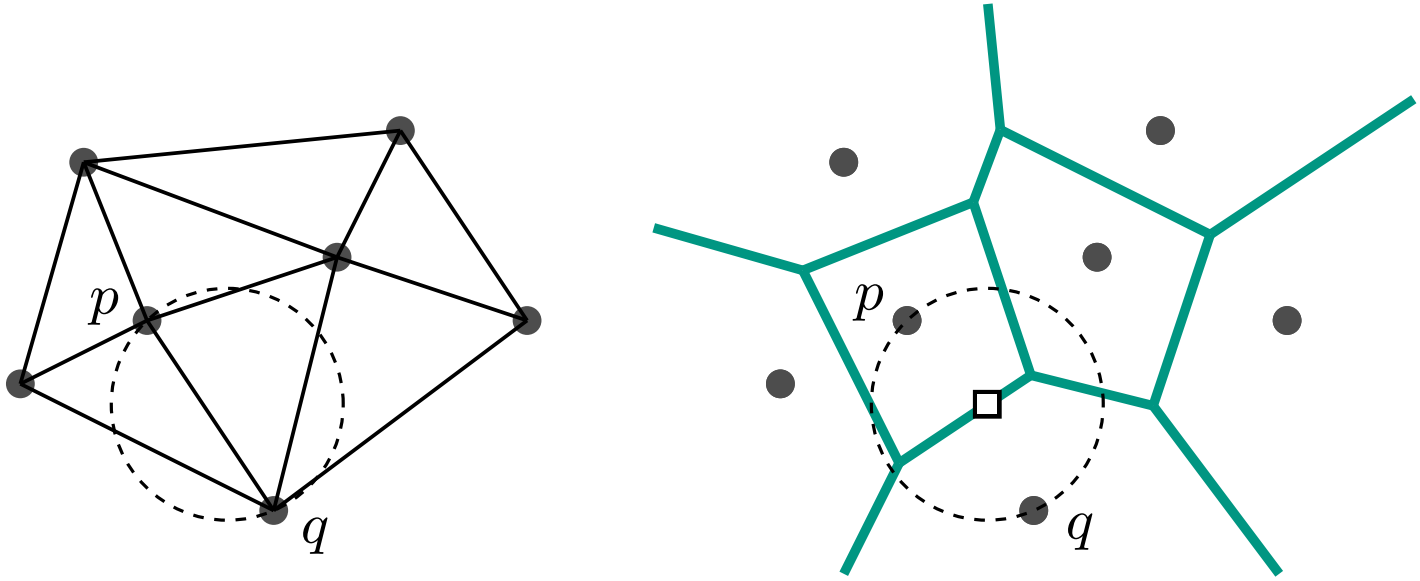


Duality

Theorem 7:

Under the general position assumption, the Delaunay triangulation is a dual graph of the Voronoi diagram.

- A site $p \leftrightarrow$ a Voronoi cell $\mathcal{V}(p)$
- A Delaunay edge $\overline{pq}$
  $\leftrightarrow$ a Voronoi edge between $\mathcal{V}(p)$ and $\mathcal{V}(q)$

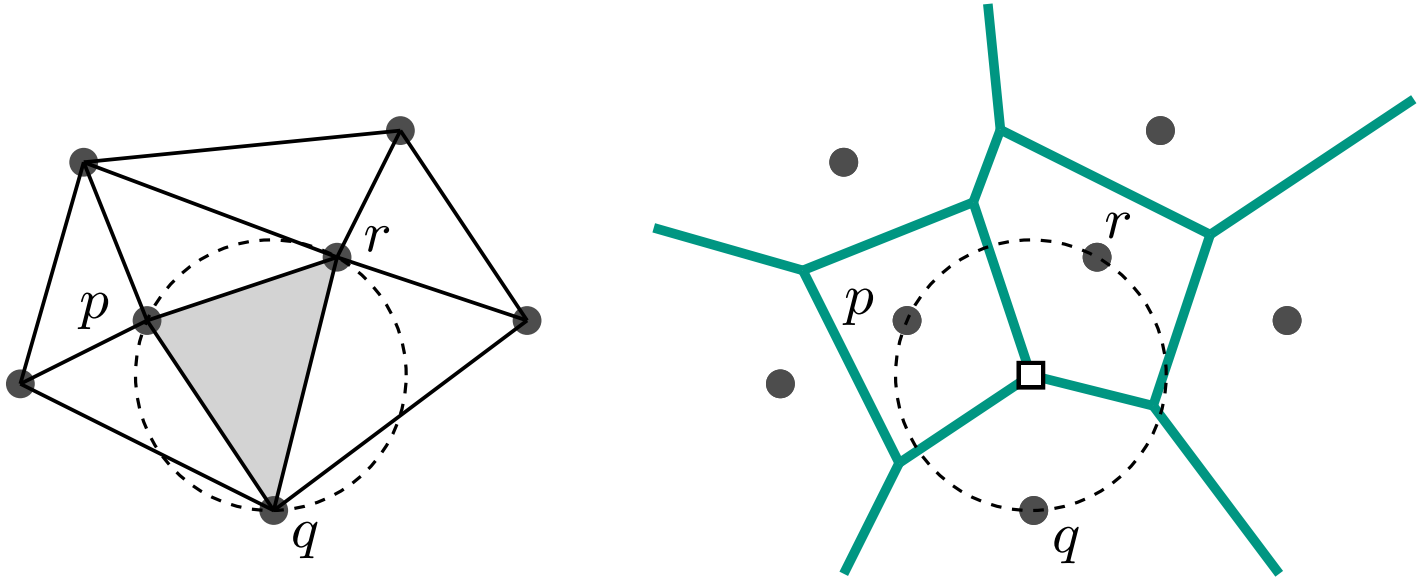


Duality

Theorem 7:

Under the general position assumption, the Delaunay triangulation is a dual graph of the Voronoi diagram.

- A site $p \leftrightarrow$ a Voronoi cell $\mathcal{V}(p)$
- A Delaunay edge $\overline{pq}$
  $\leftrightarrow$ a Voronoi edge between $\mathcal{V}(p)$ and $\mathcal{V}(q)$
- A Delaunay triangle $\triangle pqr$
  $\leftrightarrow$ a Voronoi vertex among $\mathcal{V}(p)$, $\mathcal{V}(q)$ and $\mathcal{V}(r)$



Computing Vor(P) (Upper Bound)

Karlsruhe Institute of Technology

How can we calculate Vor(P) with methods we already know?

Computing Vor(P) (Upper Bound)

Karlsruhe Institute of Technology

How can we calculate Vor(P) with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

Computing Vor(P) (Upper Bound)

Karlsruhe Institute of Technology

How can we calculate Vor(P) with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

foreach $p \in P$ **do**
  compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$

Computing Vor(P) (Upper Bound)

Karlsruhe Institute of Technology

How can we calculate Vor(P) with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

foreach $p \in P$ **do**

compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Computing Vor(P) (Upper Bound)

How can we calculate Vor(P) with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

foreach $p \in P$ **do** $O(n^2 \log n)$

  compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Computing Vor(P) (Upper Bound)

Karlsruhe Institute of Technology

How can we calculate Vor(P) with methods we already know?

For each $p \in P$ is $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ is the intersection of $n - 1$ half planes.

foreach $p \in P$ **do** $O(n^2 \log n)$

  compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$ $O(n \log n)$ [Lecture 4]

Is $O(n^2 \log n)$ running time for a linear-size object necessary?

Computing Vor(P) (Lower Bound)

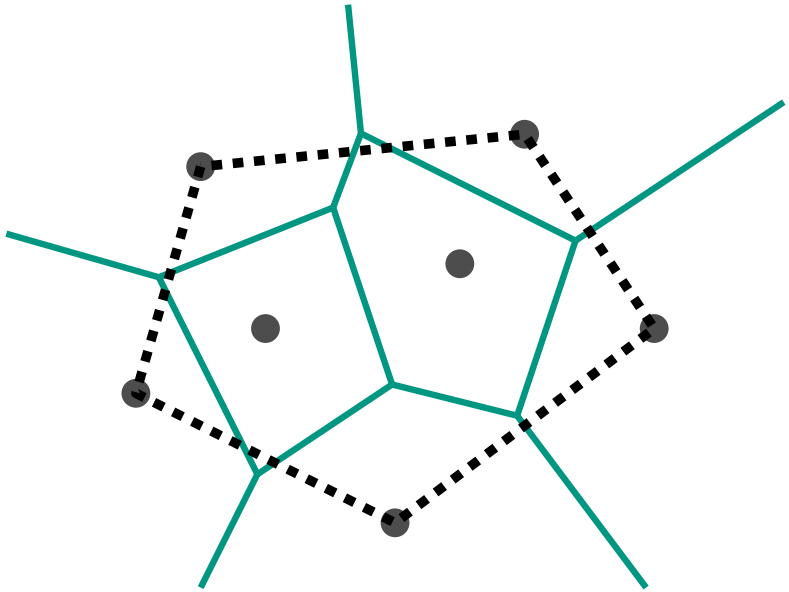
- Lower Bound: $\Omega(n \log n)$ Time

Computing Vor(P) (Lower Bound)

- Lower Bound: $\Omega(n \log n)$ Time
  - Convex hull of $P \Leftarrow$ Linear time from Vor(P)

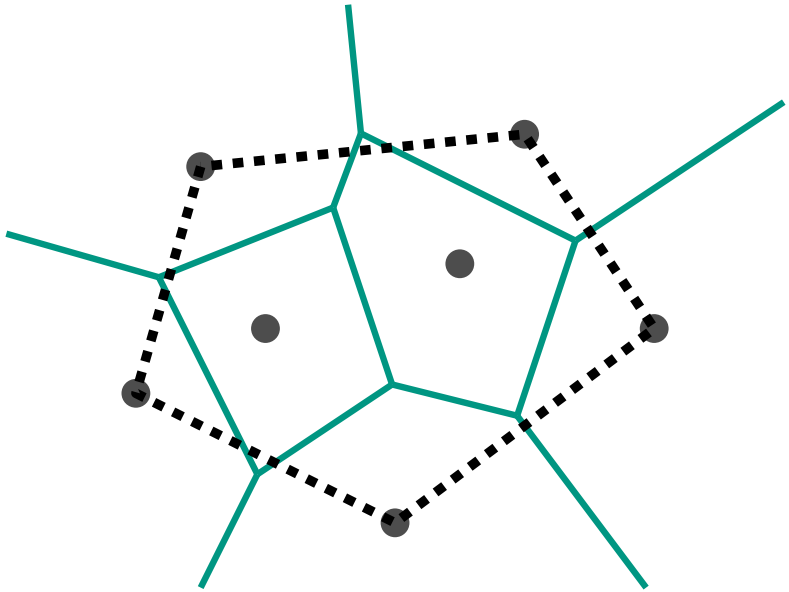
Computing Vor(P) (Lower Bound)

- Lower Bound: $\Omega(n \log n)$ Time
  - Convex hull of $P \Leftarrow$ Linear time from Vor(P)



Computing Vor(P) (Lower Bound)

- Lower Bound: $\Omega(n \log n)$ Time
  - Convex hull of $P \Leftarrow$ Linear time from Vor(P)



- $O(n \log n)$-Time Algorithms
  - Plane Sweep
  - Divide and Conquer

Plane-Sweep Paradigm

- Move a line H to sweep the plane (from left to right)

Plane-Sweep Paradigm

- Move a line H to sweep the plane (from left to right)
  - Compute **substructure** left to H for **input** left to H

Plane-Sweep Paradigm

- Move a line H to sweep the plane (from left to right)
  - Compute **substructure** left to H for **input** left to H
- Line Segment Intersection Problem

Plane-Sweep Paradigm

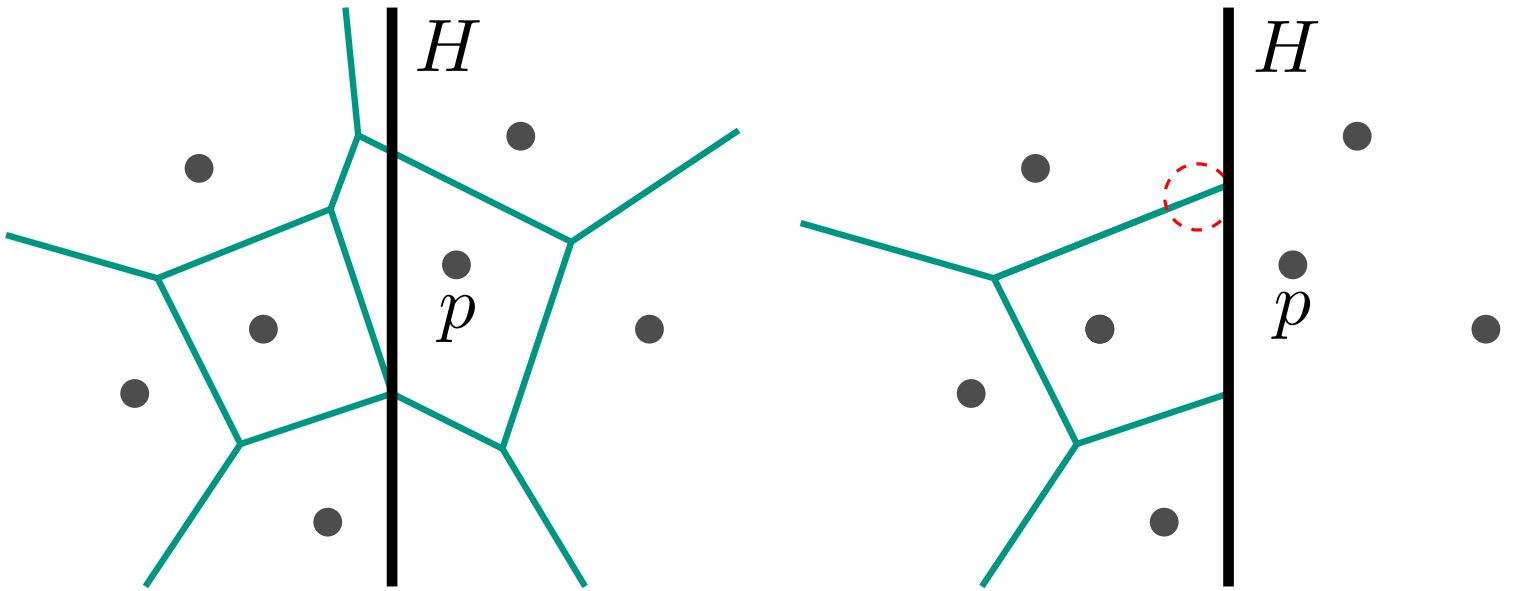
- Move a line H to sweep the plane (from left to right)
  - Compute **substructure** left to H for **input** left to H
- Line Segment Intersection Problem
  - Line segments intersected by H are sorted in y-order

Plane-Sweep Paradigm

- Move a line H to sweep the plane (from left to right)
  - Compute **substructure** left to H for **input** left to H
- Line Segment Intersection Problem
  - Line segments intersected by H are sorted in y-order
- H can intersect Voronoi region of a site right to it.

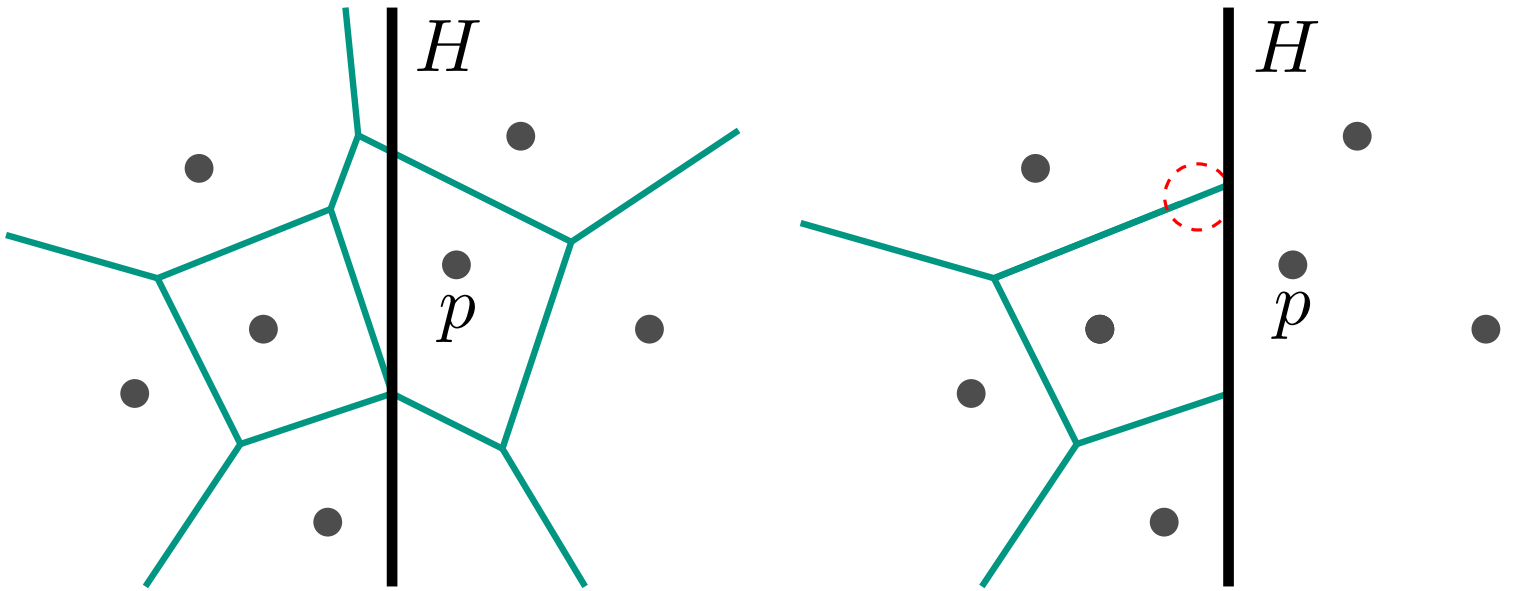
Plane-Sweep Paradigm

- Move a line H to sweep the plane (from left to right)
  - Compute **substructure** left to H for **input** left to H
- Line Segment Intersection Problem
  - Line segments intersected by H are sorted in y-order
- H can intersect Voronoi region of a site right to it.

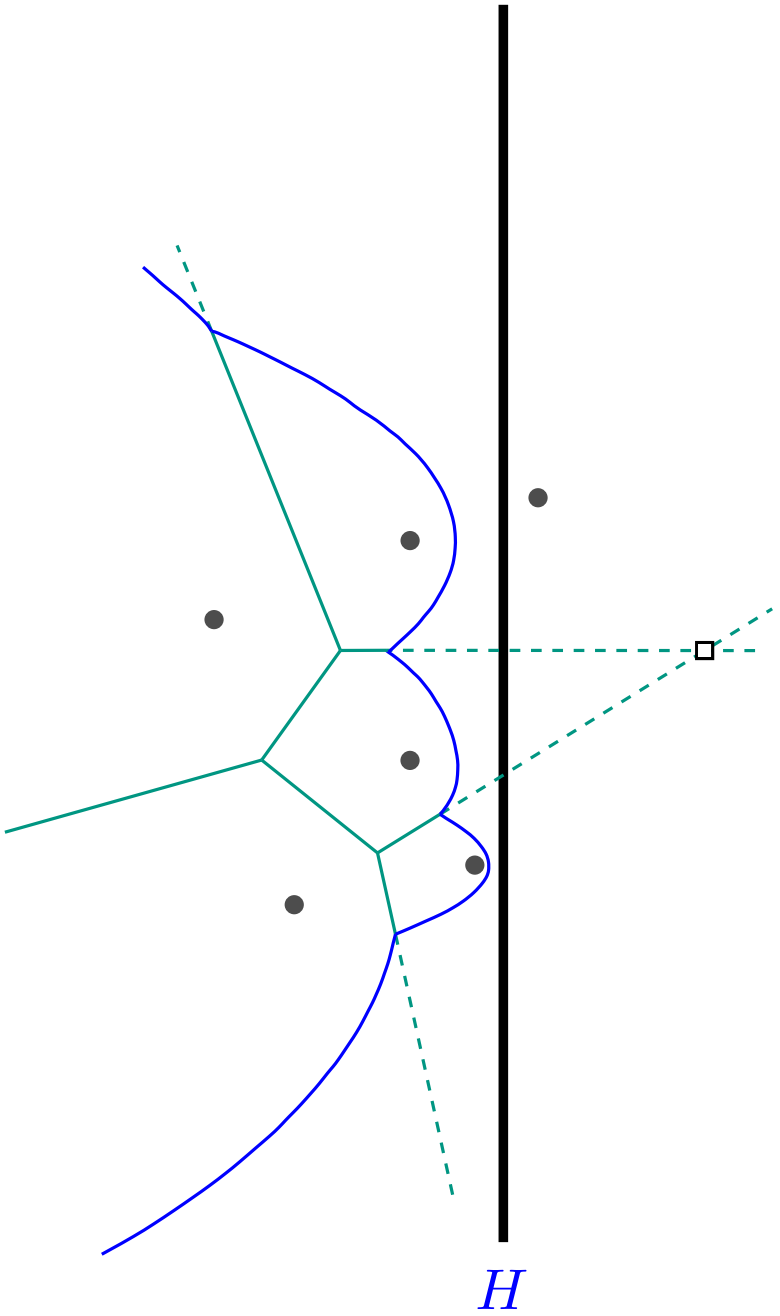


Plane-Sweep Paradigm

- Move a line H to sweep the plane (from left to right)
  - Compute **substructure** left to H for **input** left to H
- Line Segment Intersection Problem
  - Line segments intersected by H are sorted in y-order
- H can intersect Voronoi region of a site right to it.
  - only consider sites left to $H \rightarrow$ wrong substructure

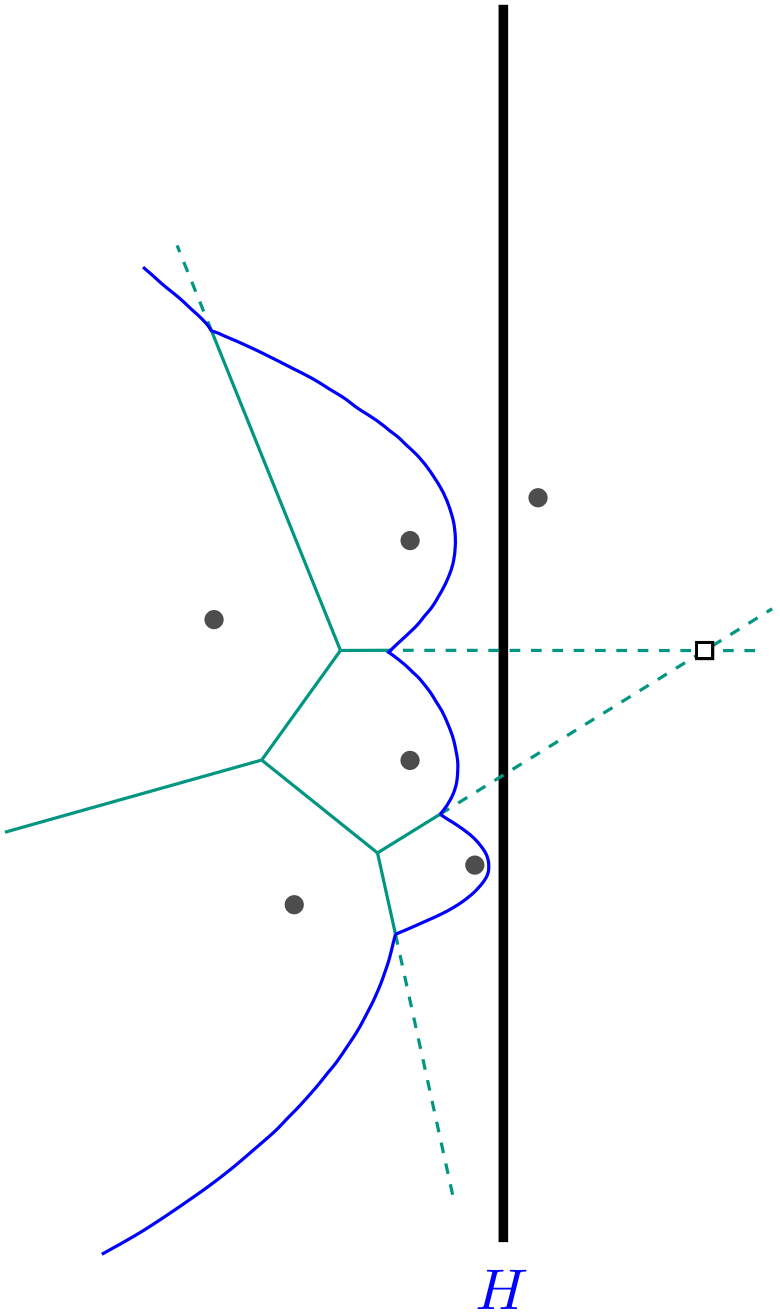


Voronoi Diagram with Sweep Line



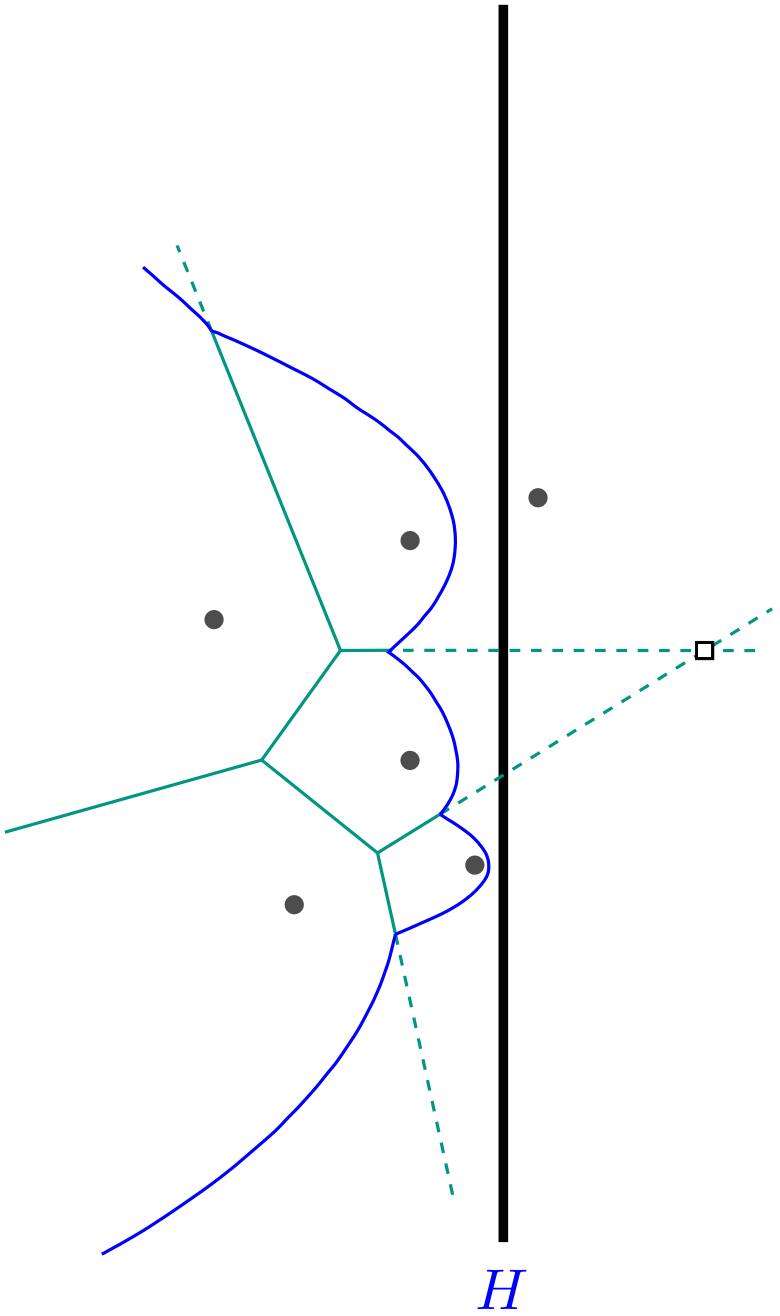
- View H as a site

Voronoi Diagram with Sweep Line



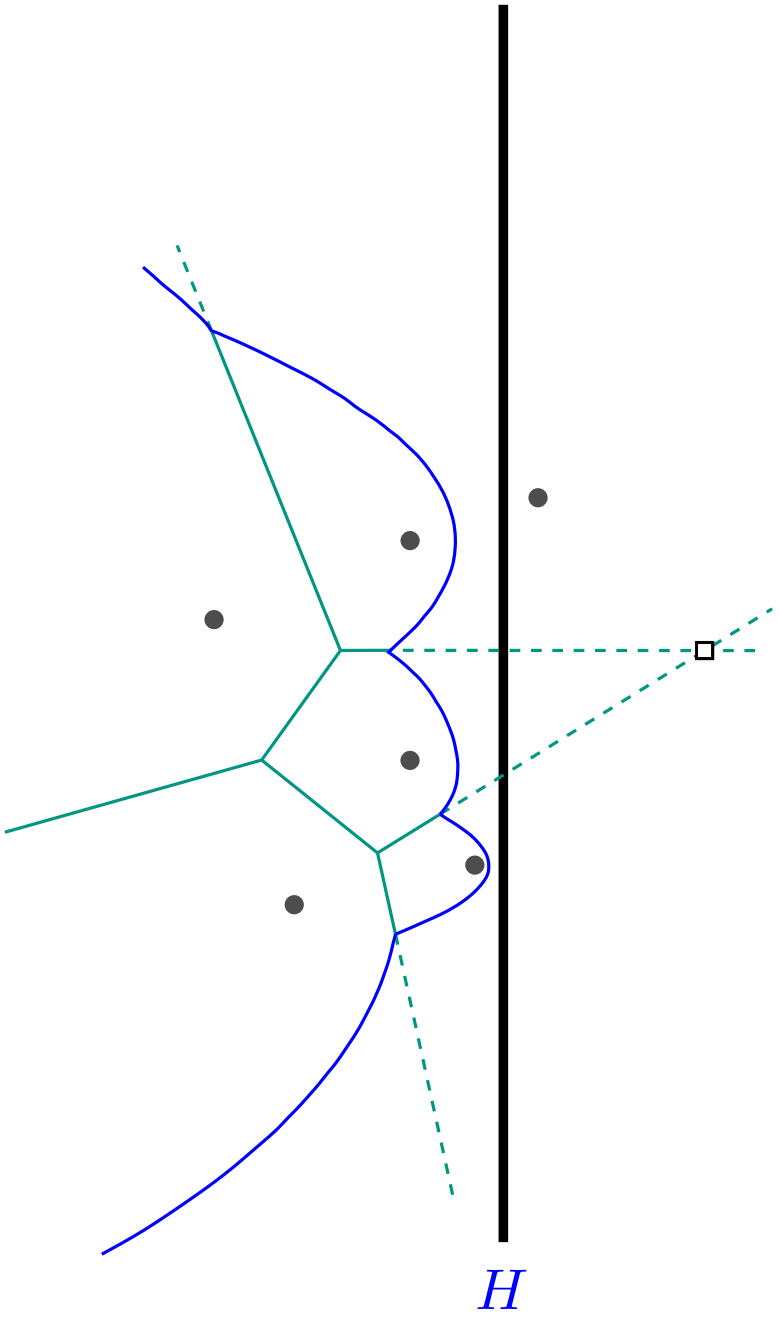
- View H as a site
- $\mathrm{Vor}(P_L \cup \{H\})$

Voronoi Diagram with Sweep Line



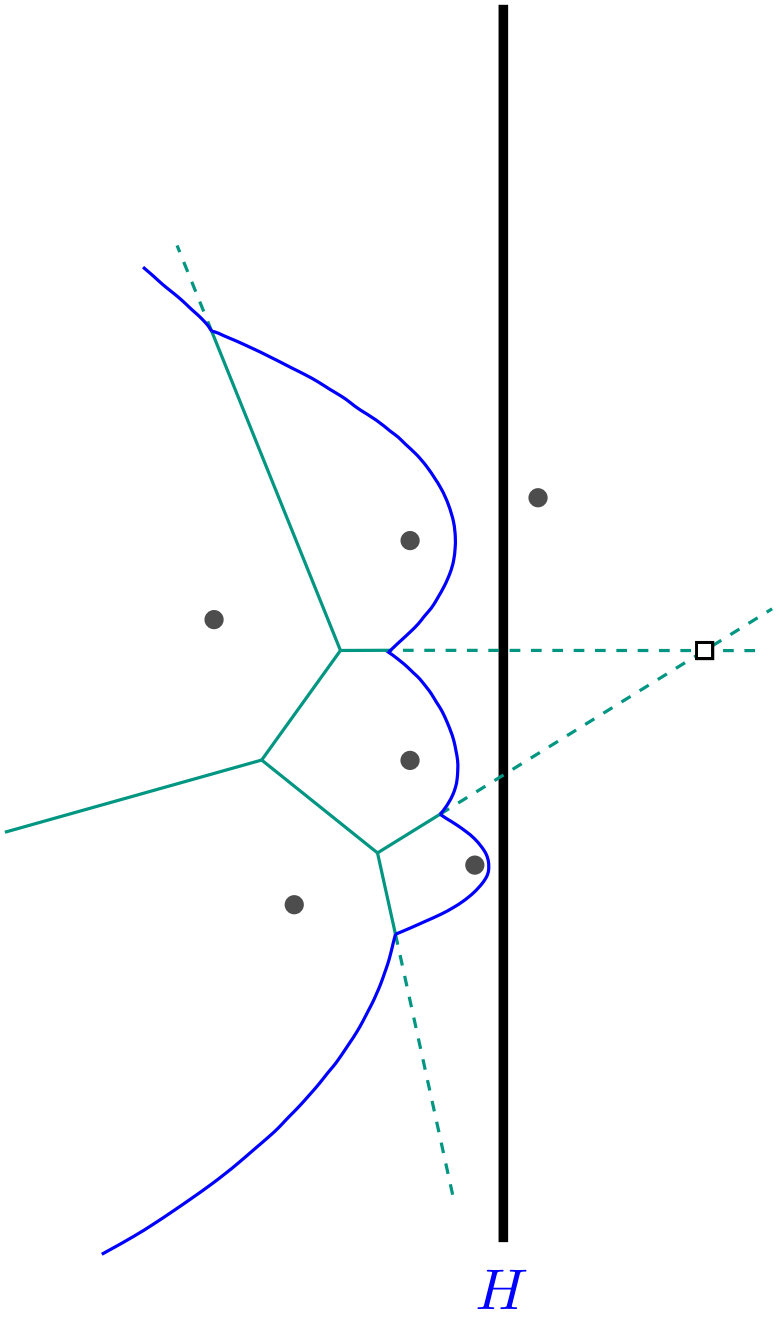
- View H as a site
- $\text{Vor}(P_L \cup \{H\})$
  - $P_L = \{p \in P \mid p \text{ is left to } H\}$

Voronoi Diagram with Sweep Line



- View H as a site
- $\mathrm{Vor}(P_L \cup \{H\})$
  - $P_L = \{p \in P \mid p \text{ is left to } H\}$
- Edges of $\mathcal{V}(H)$ are **parabolic**

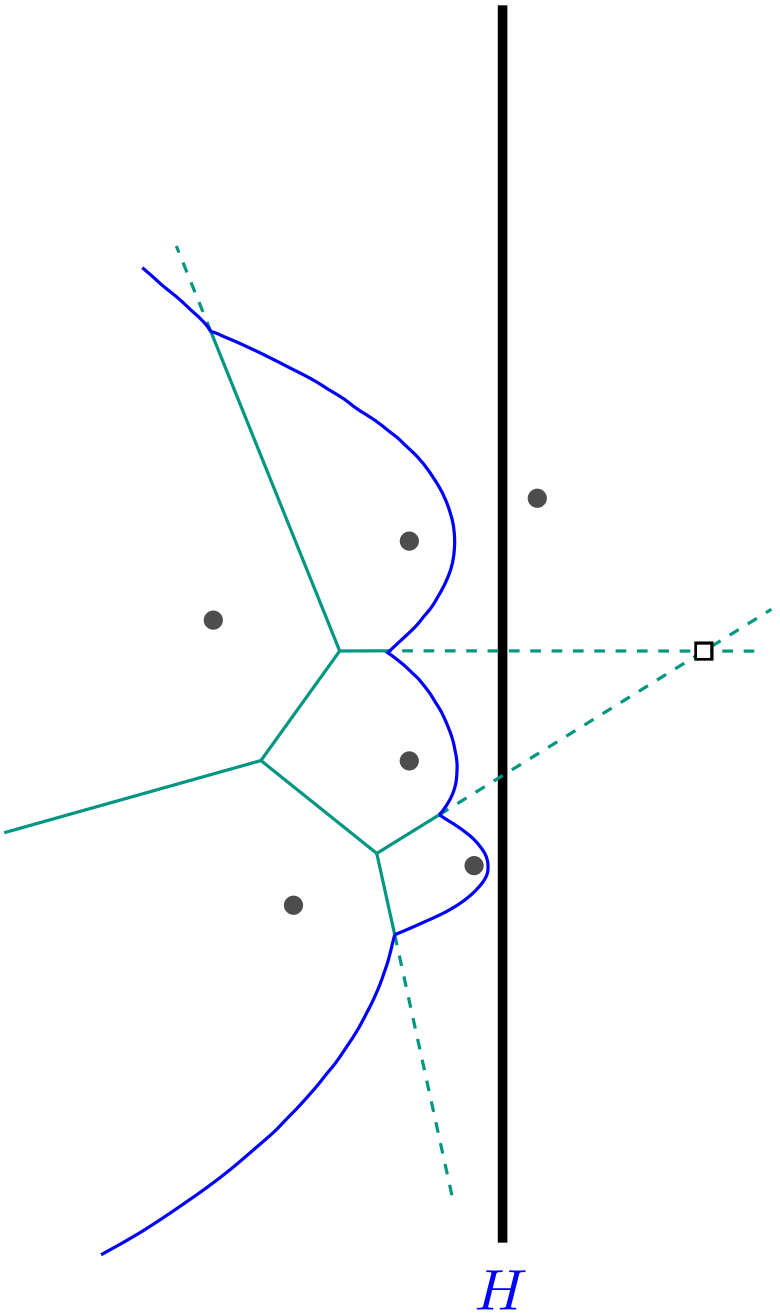
Voronoi Diagram with Sweep Line



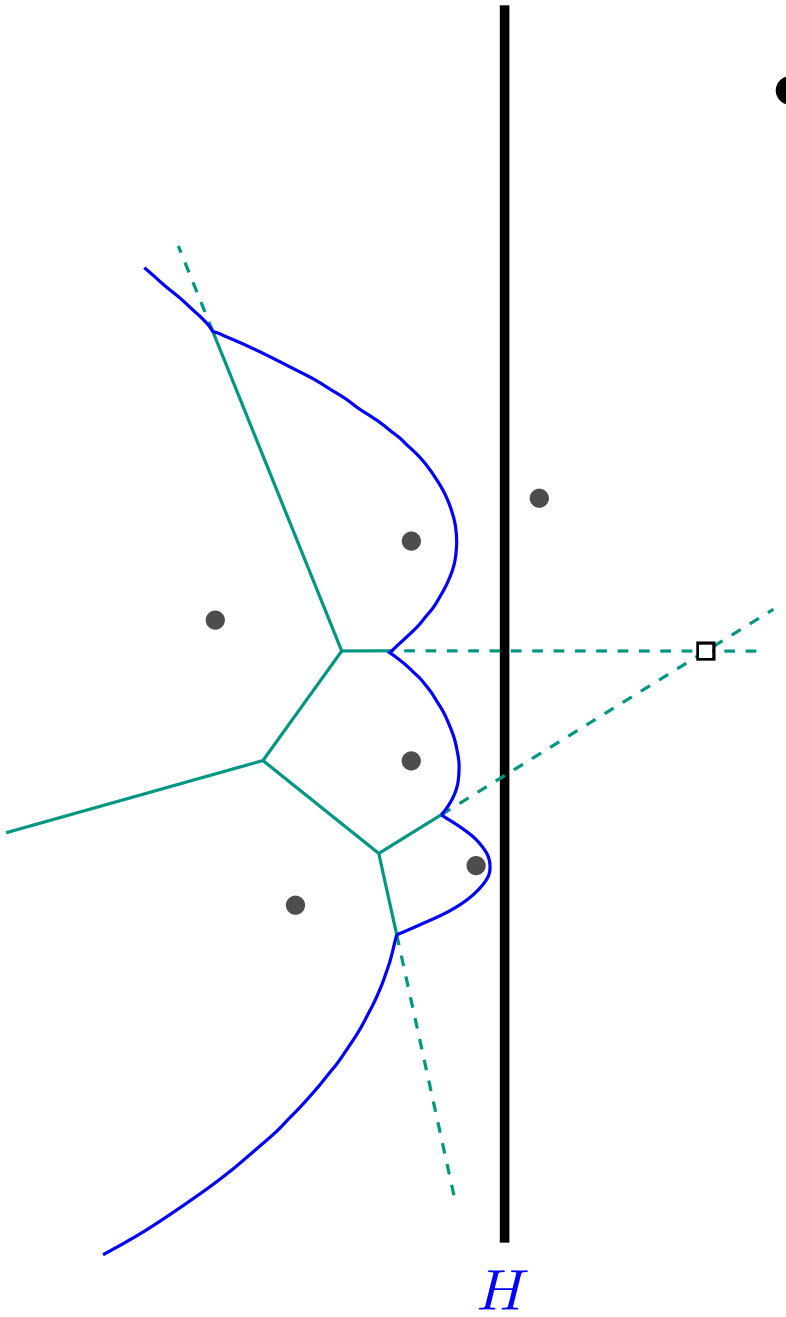
- View H as a site
- $\text{Vor}(P_L \cup \{H\})$
  - $P_L = \{p \in P \mid p \text{ is left to } H\}$
- Edges of $\mathcal{V}(H)$ are **parabolic**

$$b(p, H) = \{x \in R^2 : d(x, p) = d(x, H)\}$$

Wavefront and Spikes

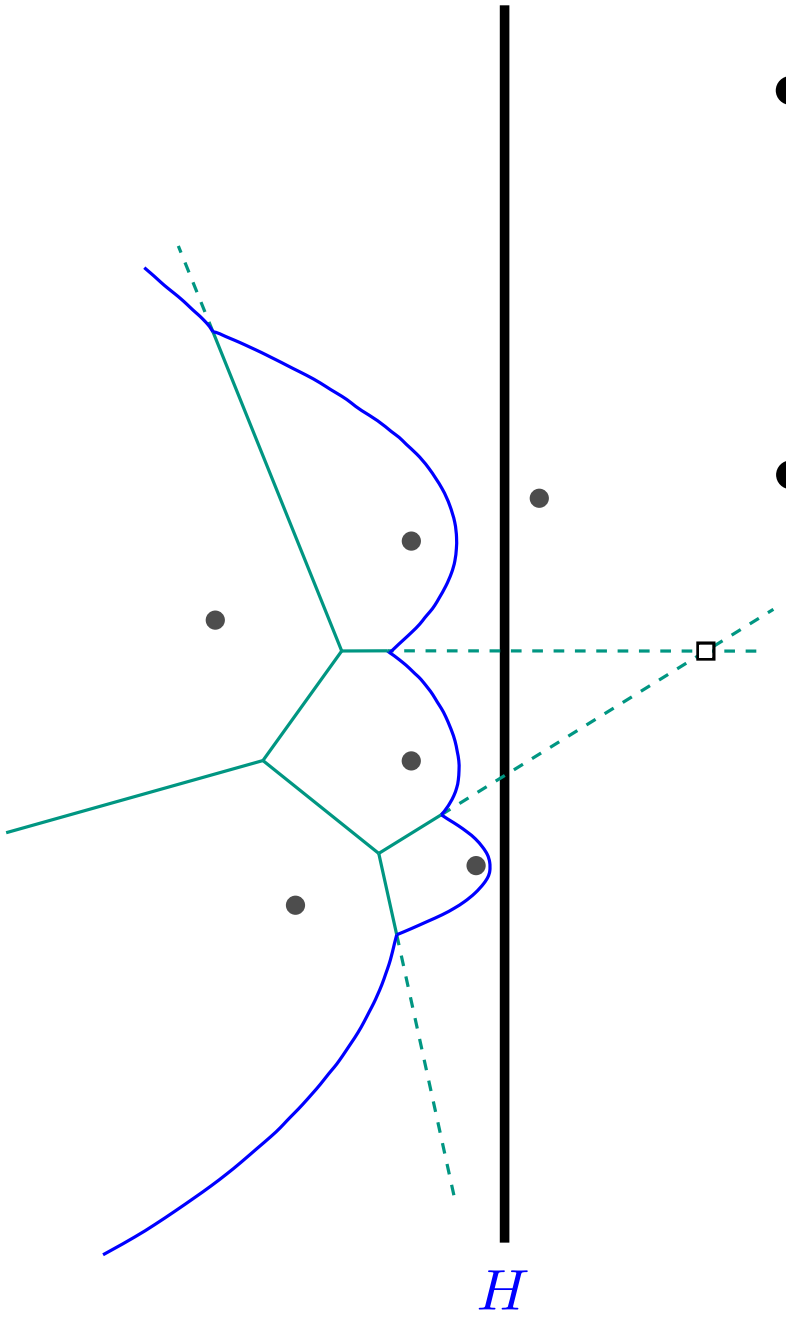


Wavefront and Spikes



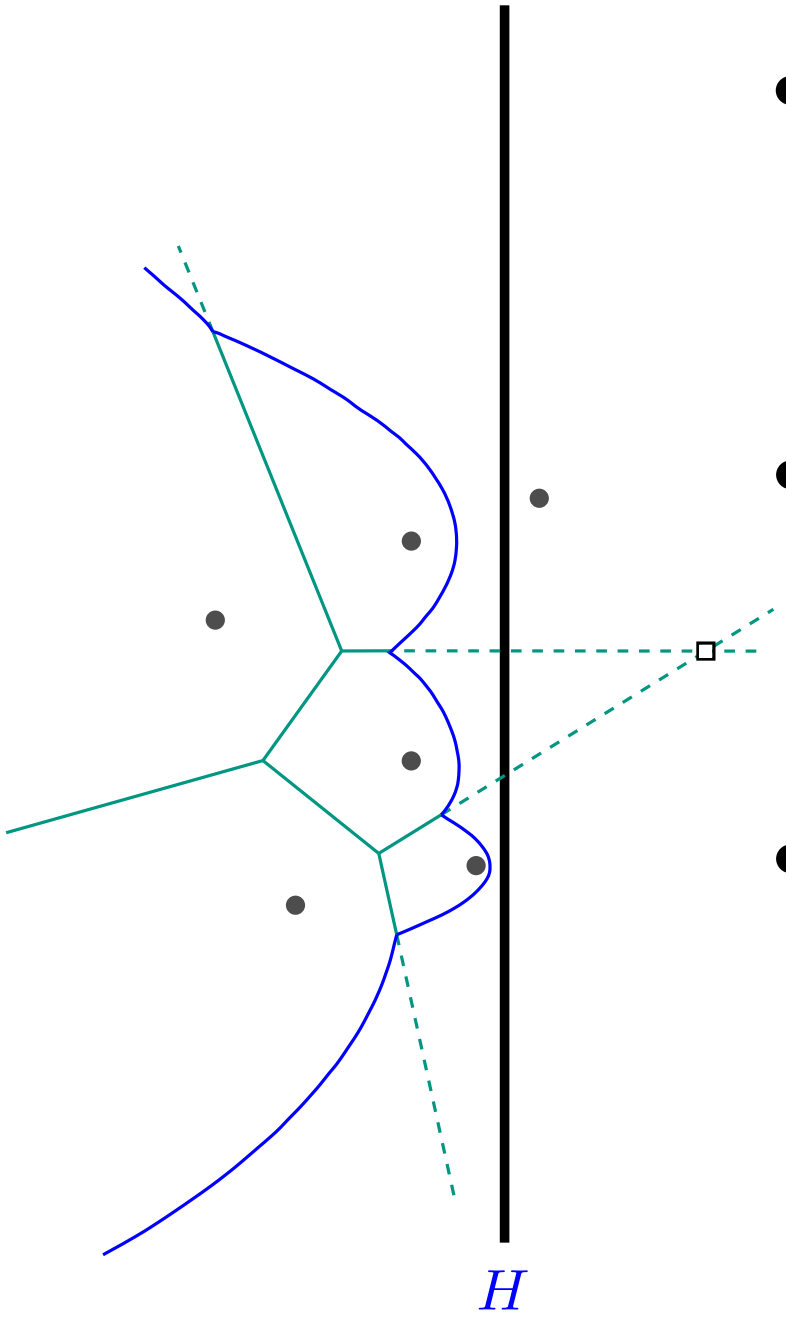
- **Wavefront** W: parabolic pieces (**wavelets**)
  - A y-monotone curve
  - A wavelet: an edge of $\mathcal{V}(H)$
  - A point site may have > 1 wavelets

Wavefront and Spikes



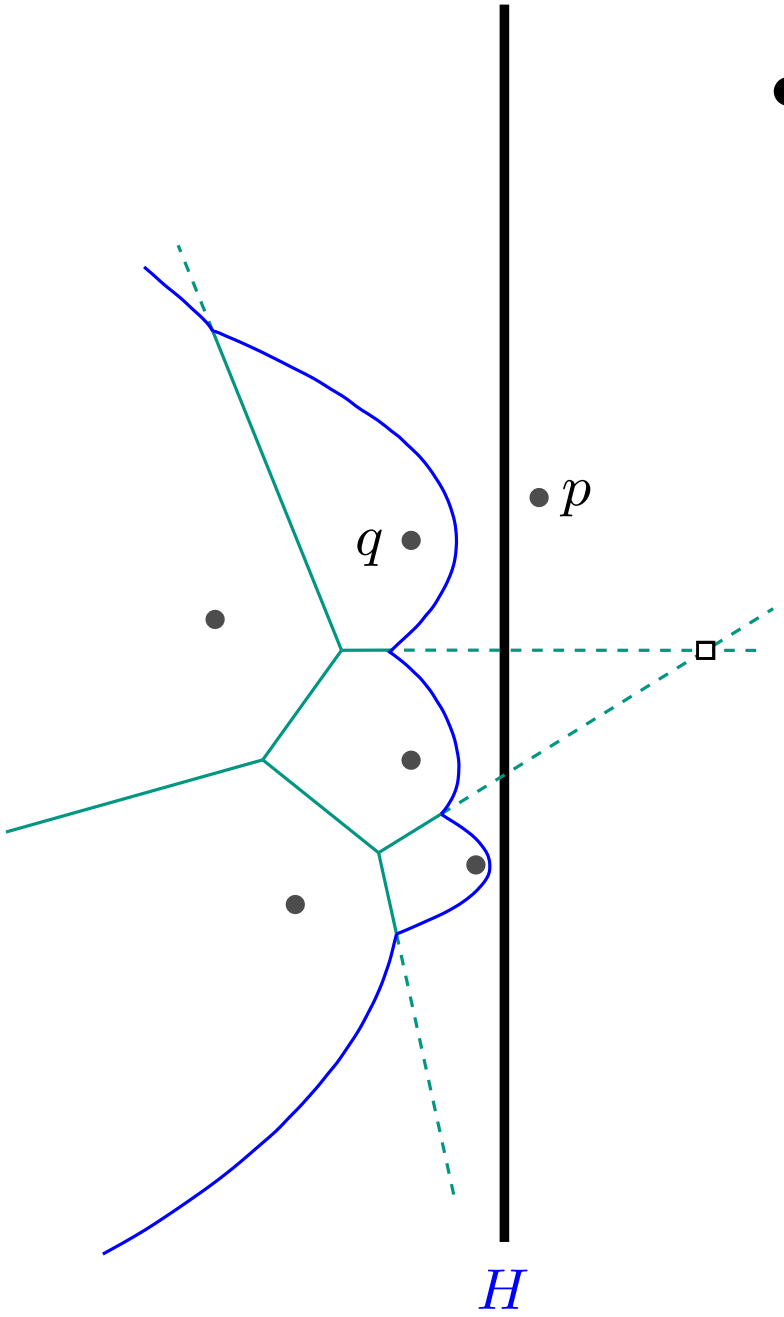
- **Wavefront** W: parabolic pieces (**wavelets**)
  - A y-monotone curve
  - A wavelet: an edge of $\mathcal{V}(H)$
  - A point site may have > 1 wavelets
- **Spike**: separates two adjacent wavelets
  - an extended part of an Voronoi edge
  - Intersection of two adjacent spikes
    $\rightarrow$ Candidate of Voronoi vertices

Wavefront and Spikes



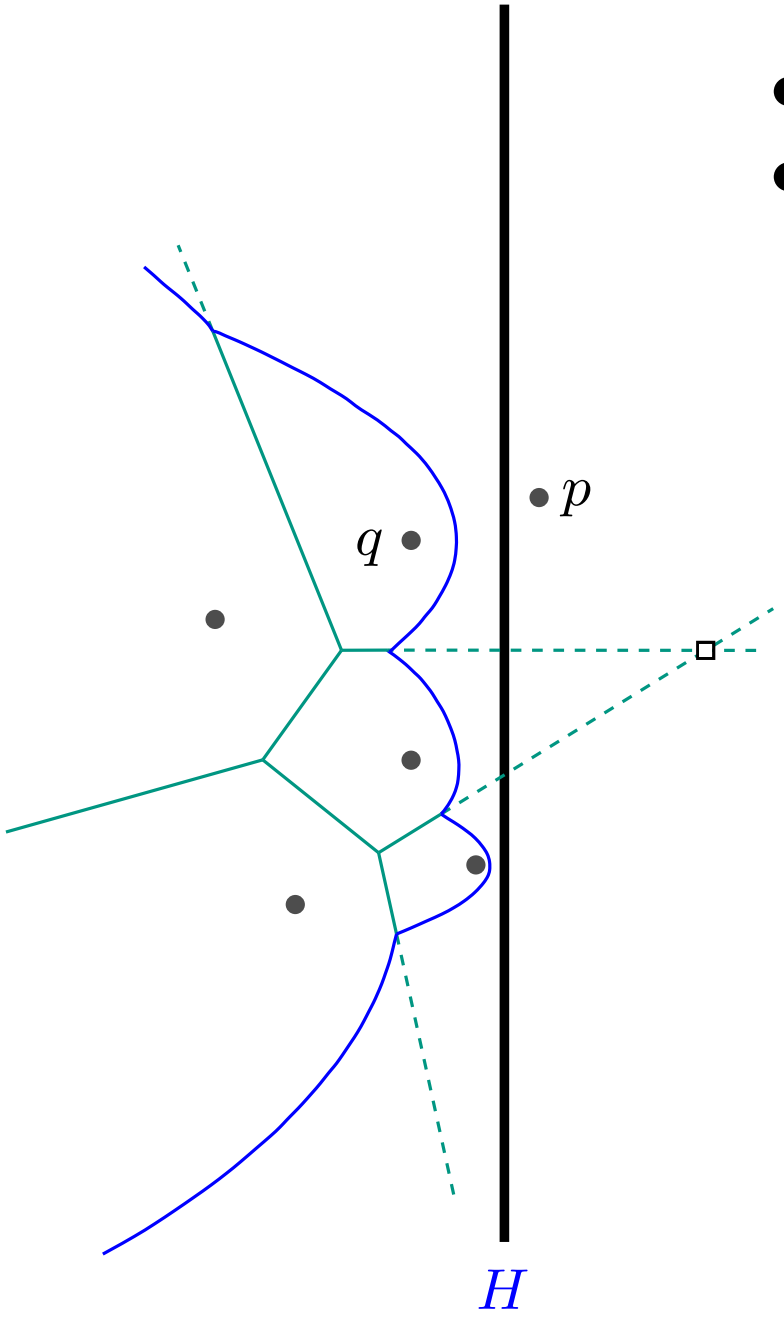
- **Wavefront** W: parabolic pieces (**wavelets**)
  - A y-monotone curve
  - A wavelet: an edge of $\mathcal{V}(H)$
  - A point site may have > 1 wavelets
- **Spike**: separates two adjacent wavelets
  - an extended part of an Voronoi edge
  - Intersection of two adjacent spikes
    $\rightarrow$ Candidate of Voronoi vertices
- H moves right
  - W moves right
  - Wavelets move along their spikes

Site Event



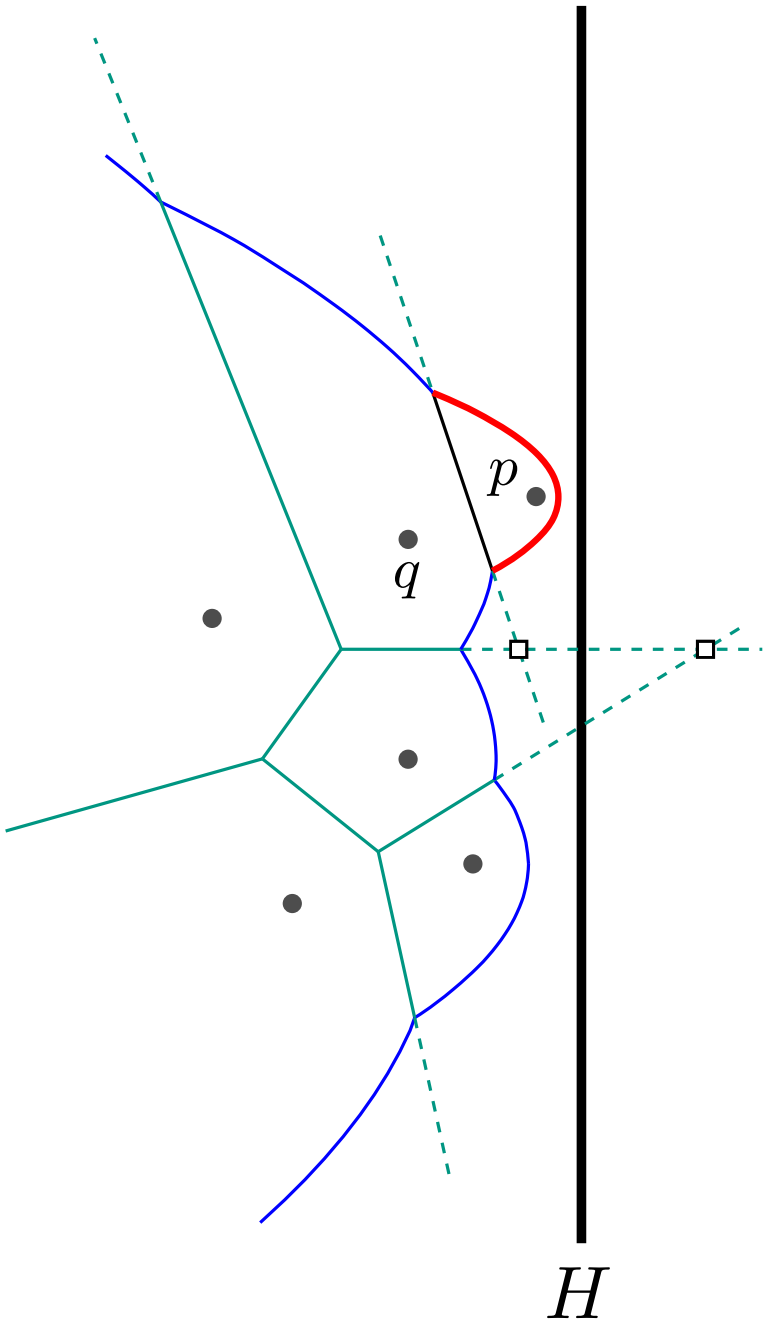
- H hits a new site p

Site Event



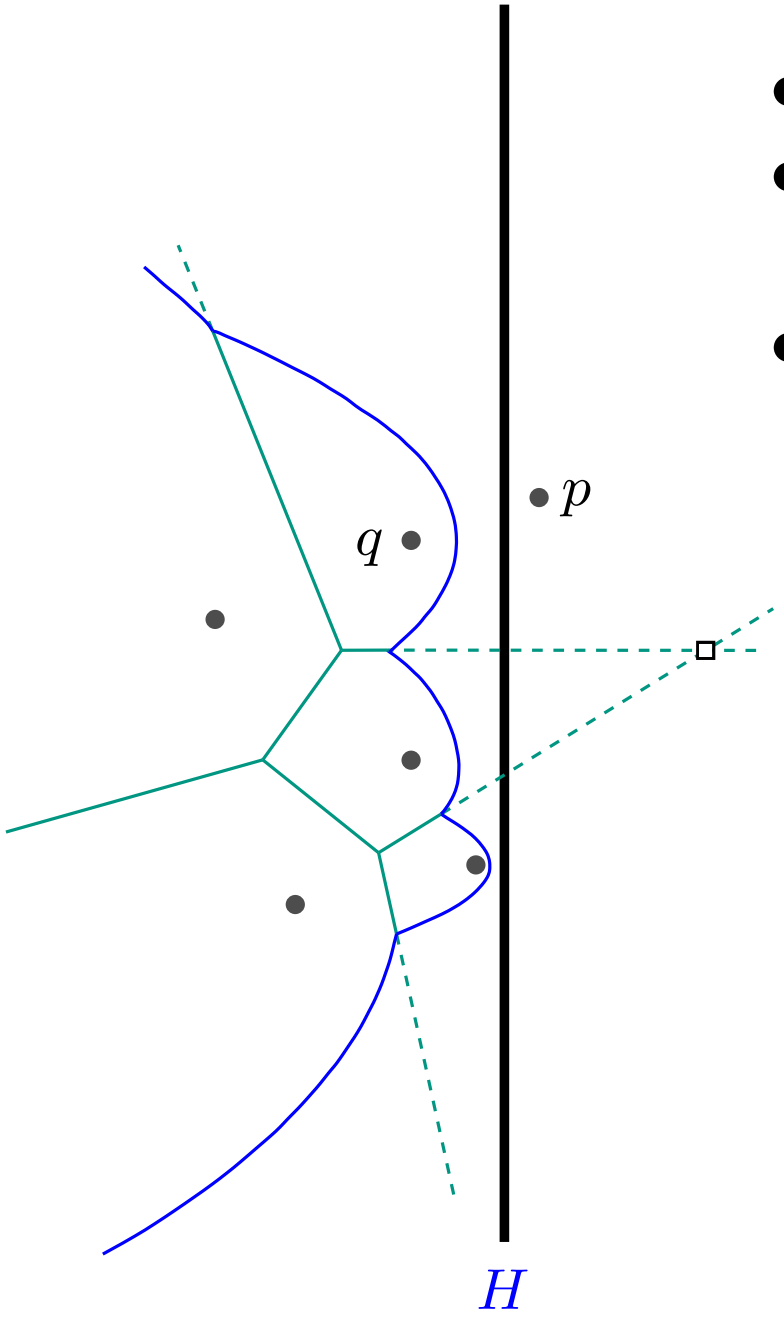
- H hits a new site p
- A new Voronoi cell forms
  - Where ?

Site Event



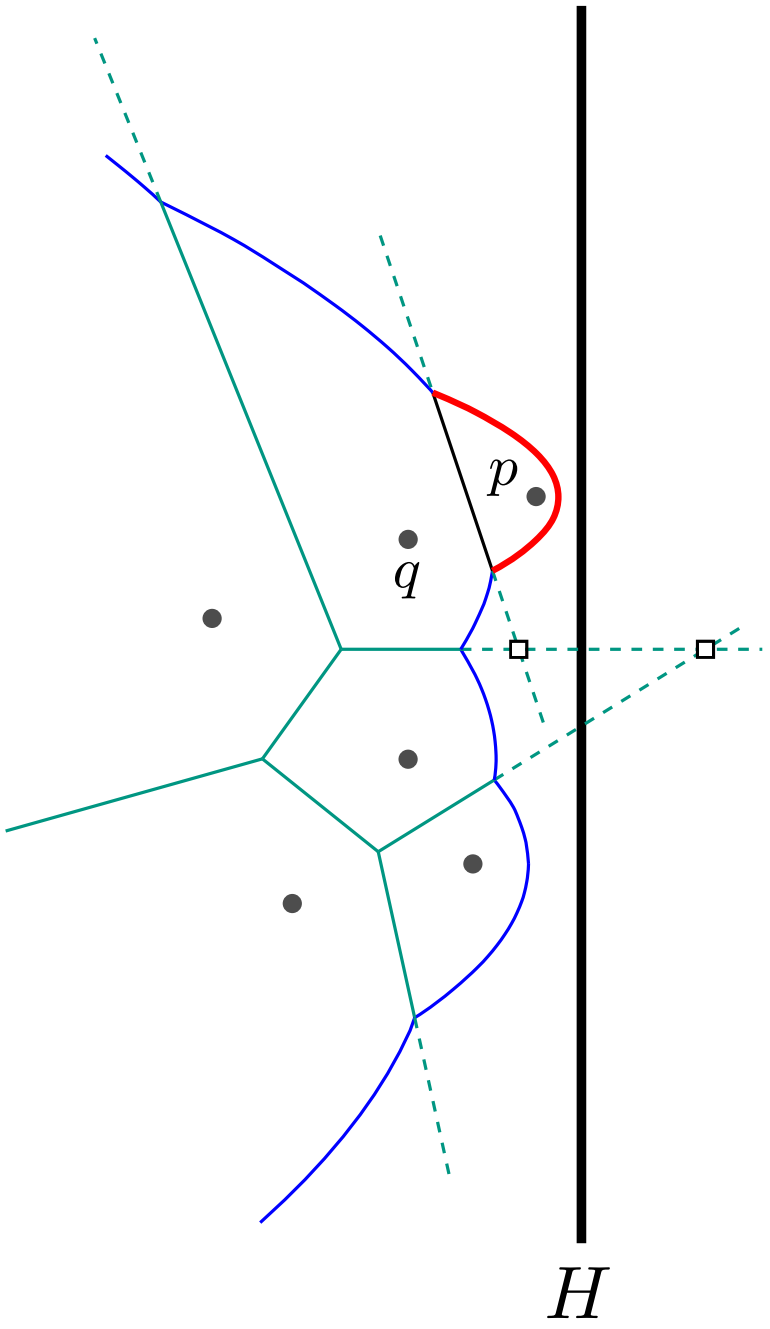
- H hits a new site p
- A new Voronoi cell forms
  - Where ?

Site Event



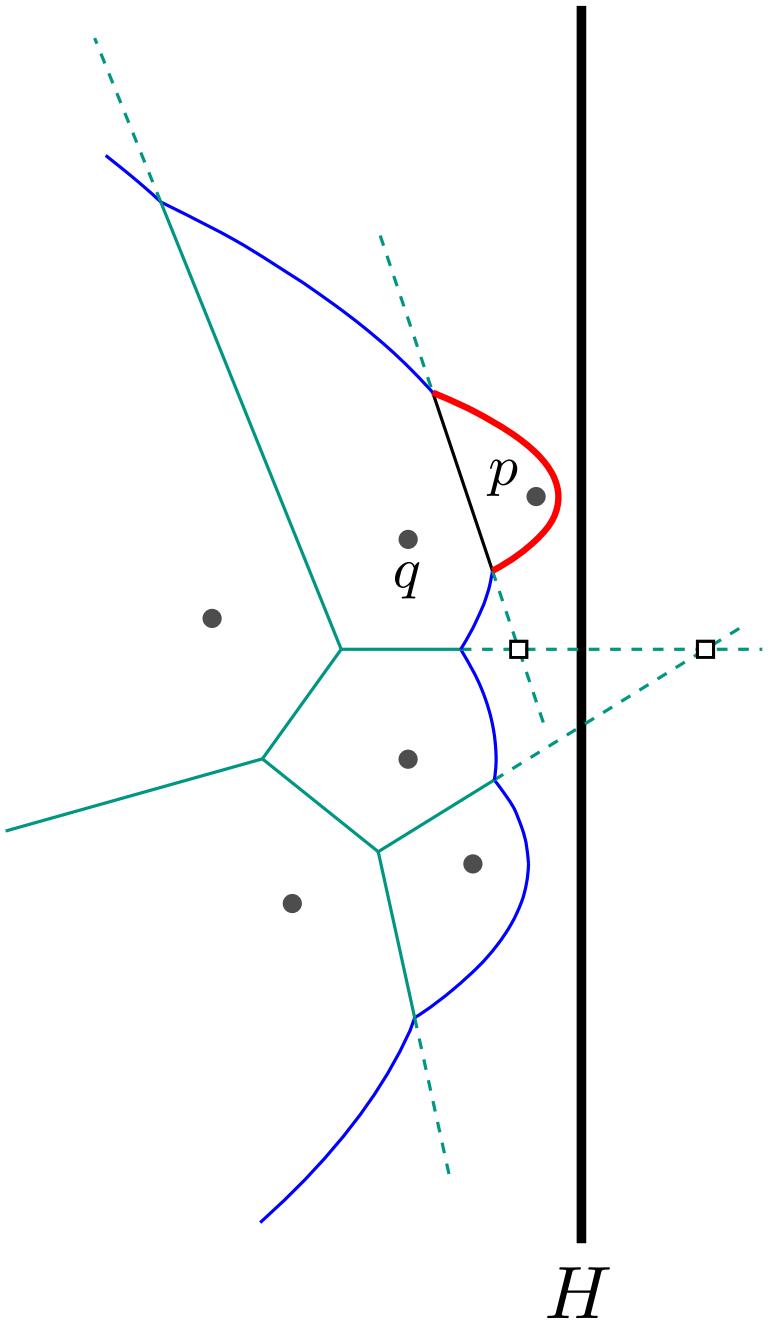
- H hits a new site p
- A new Voronoi cell forms
  - Where ?
- Shoot a left horizontal ray from p to hit a wavelet of a site q.

Site Event



- H hits a new site p
- A new Voronoi cell forms
  - Where ?
- Shoot a left horizontal ray from p to hit a wavelet of a site q.

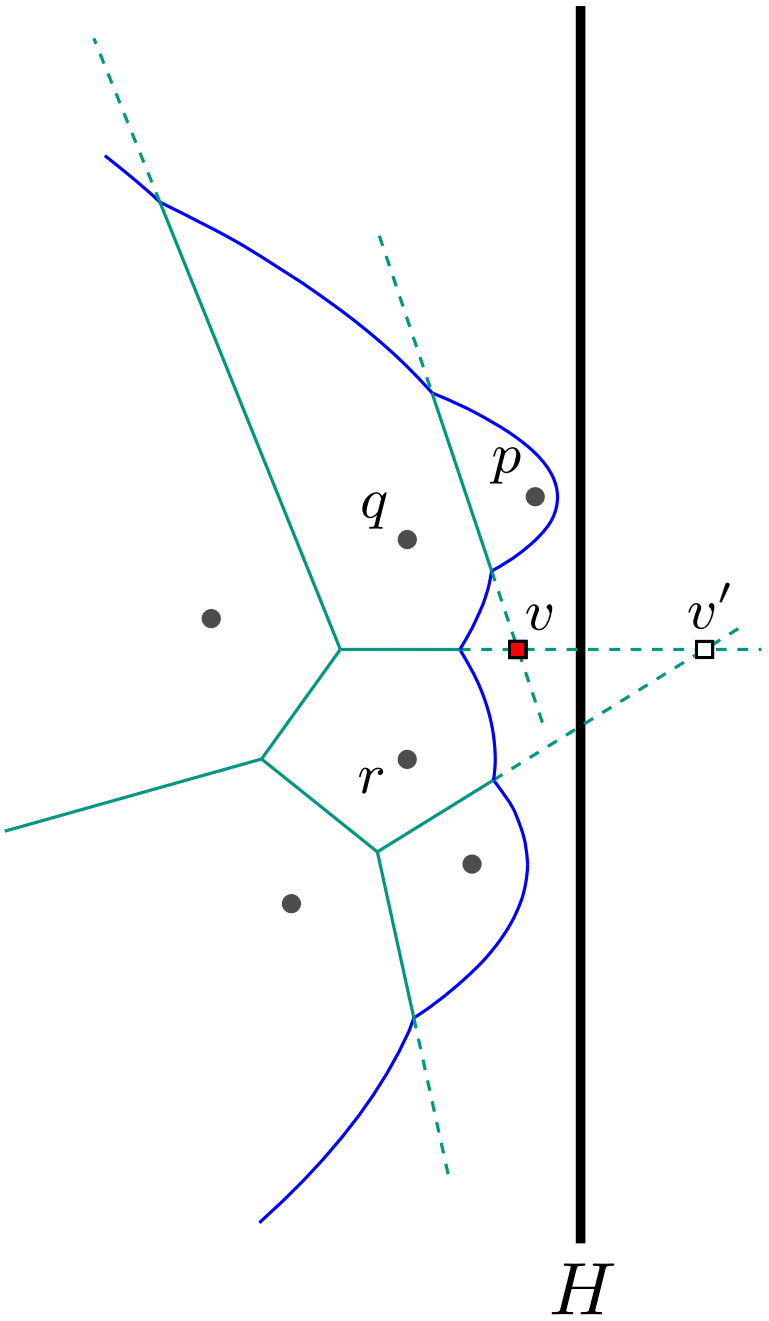
Site Event



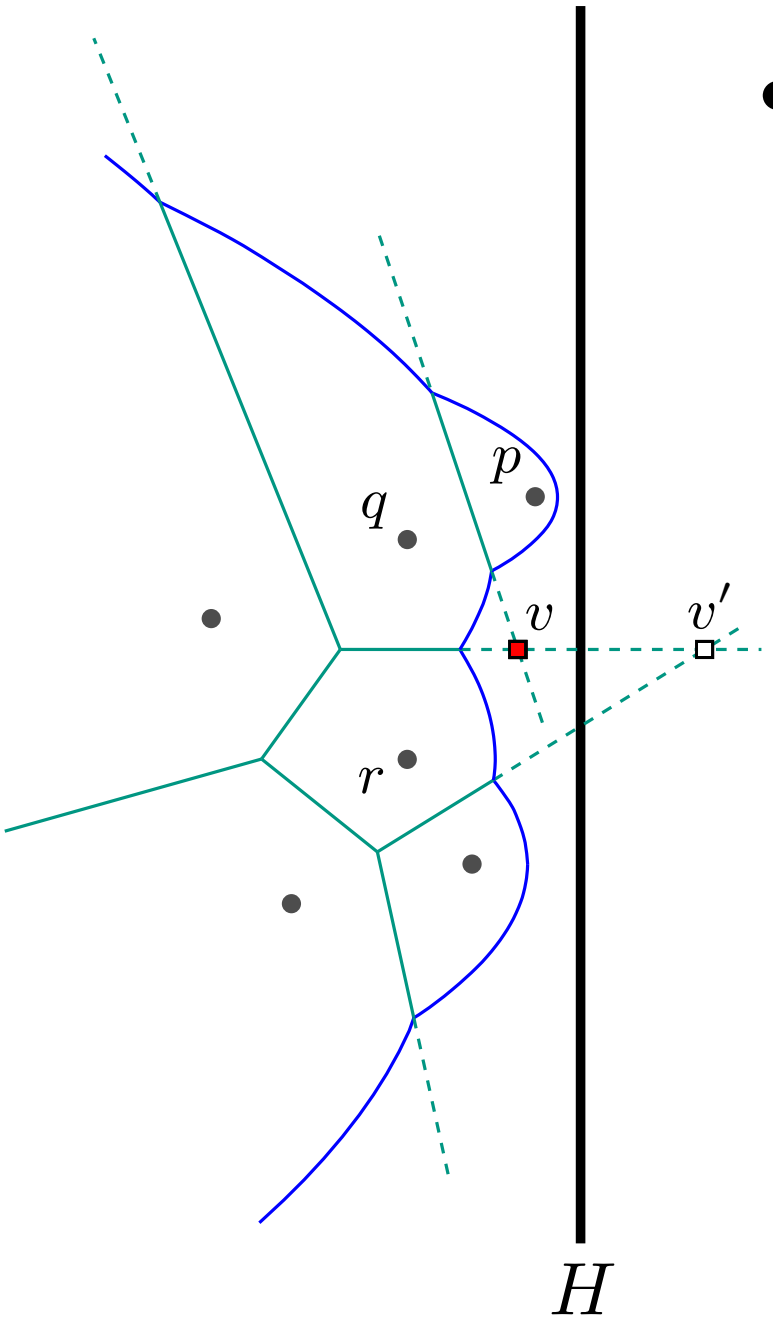
- H hits a new site p
- A new Voronoi cell forms
  - Where ?
- Shoot a left horizontal ray from p to hit a wavelet of a site q.

1. Add one new wavelet
2. Separate one old wavelet into two
3. Add one new Voronoi edge between p's and q's regions
4. Add two new spikes
   - Add at most two intersections with two adjacent spikes

Spike Event

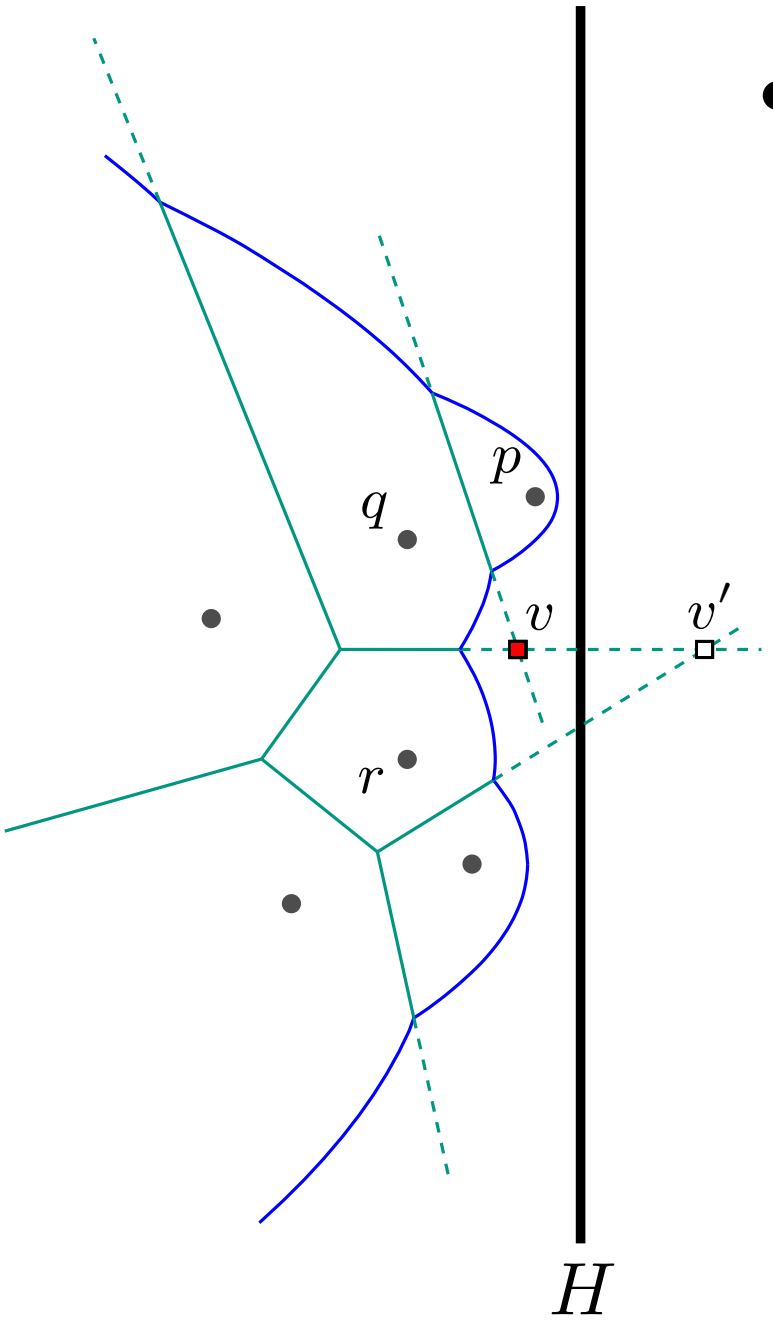


Spike Event



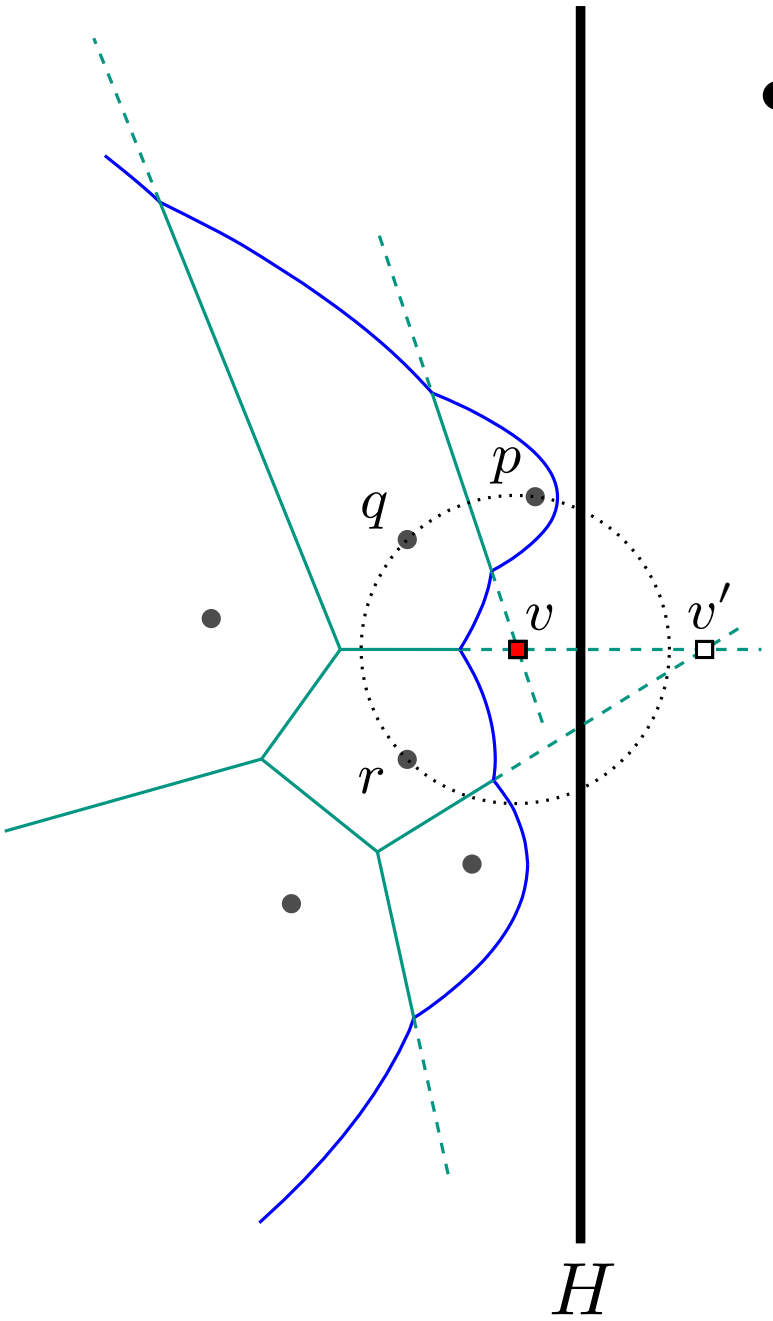
- W hits the intersection between two spikes

Spike Event



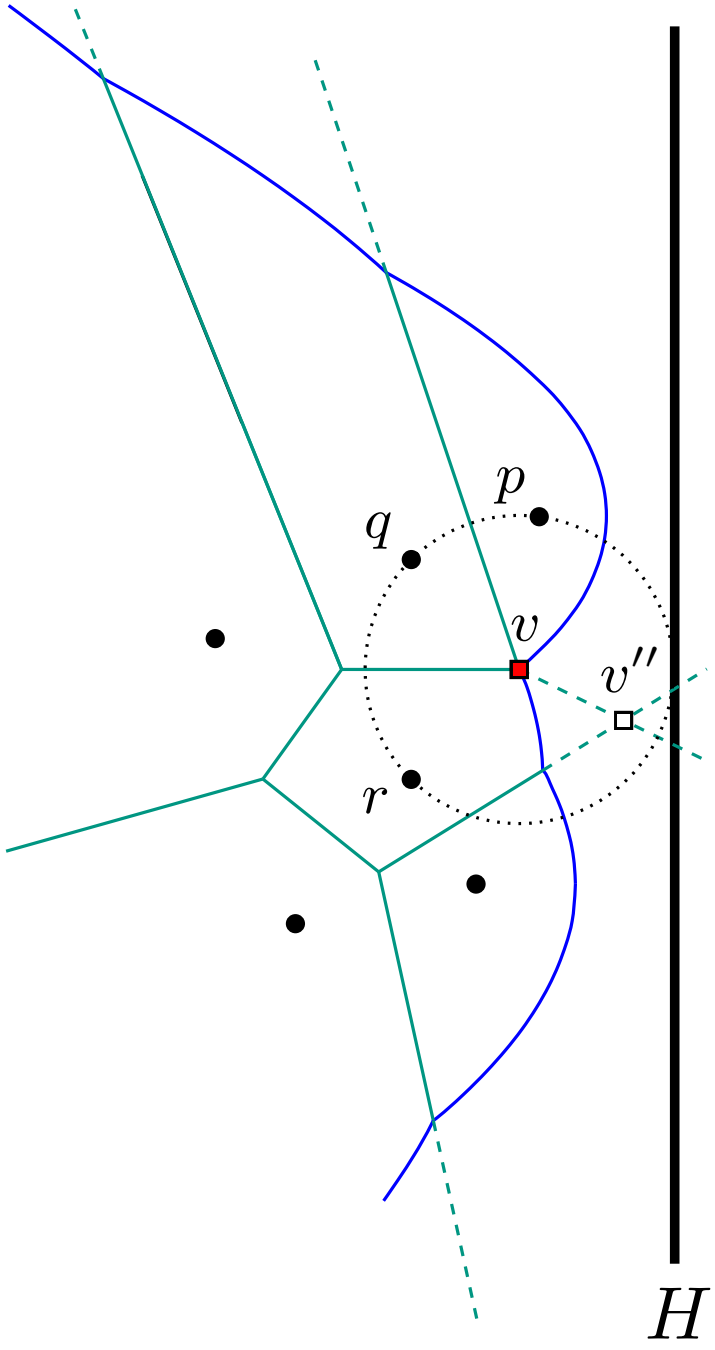
- W hits the intersection between two spikes
  - H leaves the circumcircle.

Spike Event



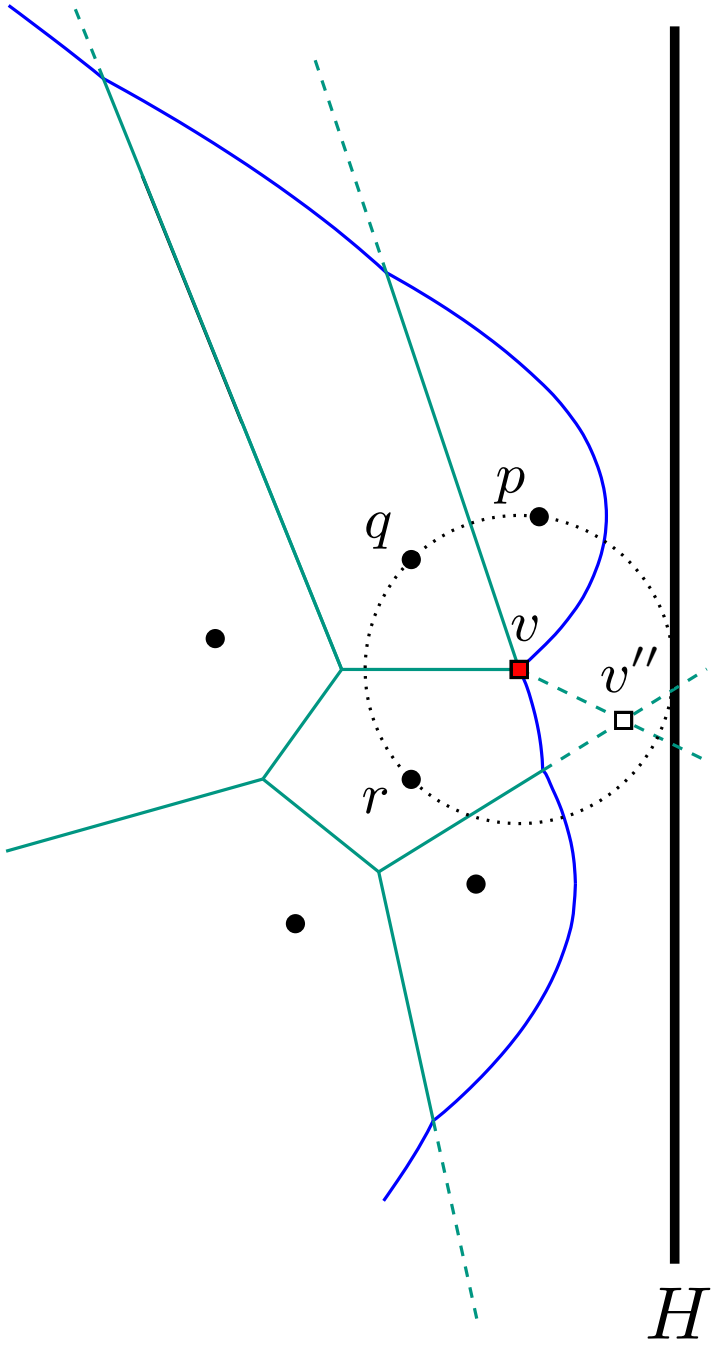
- W hits the intersection between two spikes
  - H leaves the circumcircle.

Spike Event



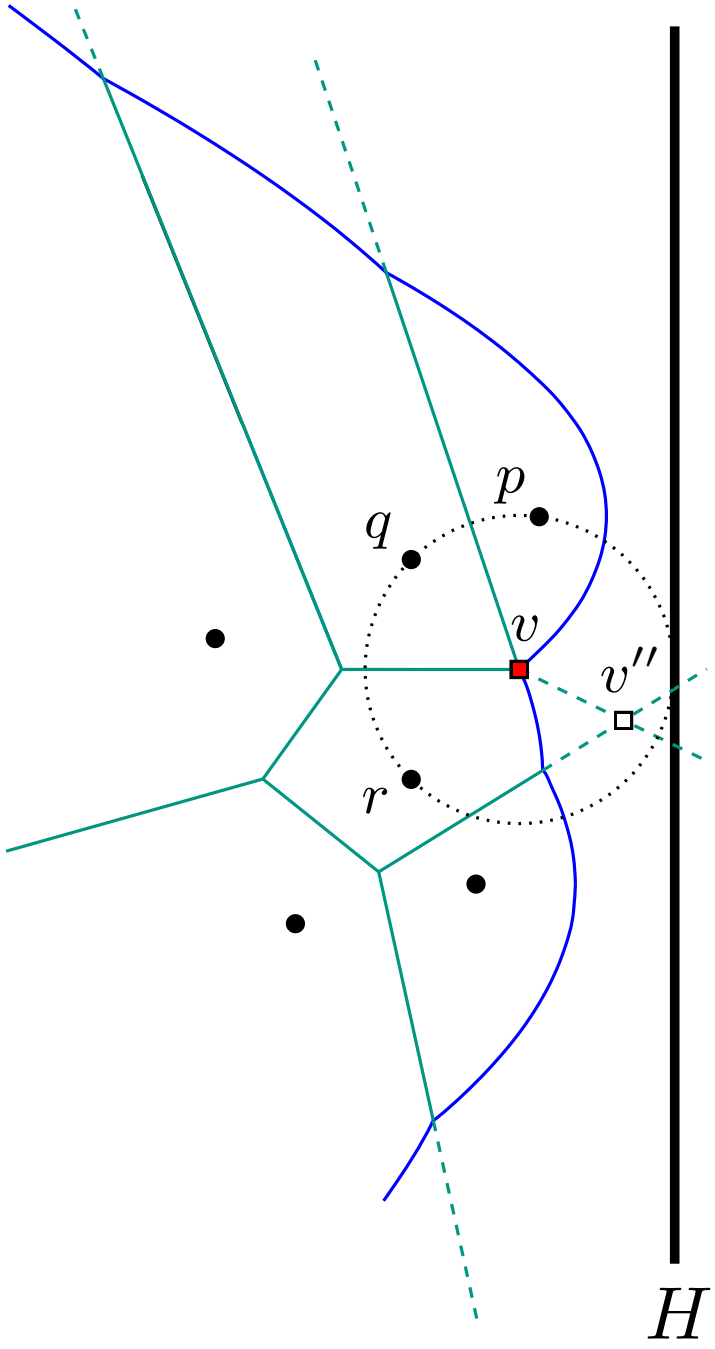
- W hits the intersection between two spikes
  - H leaves the circumcircle.

Spike Event



- W hits the intersection between two spikes
  - H leaves the circumcircle.
- A new Voronoi vertex forms
  - v: Voronoi vertex of $\mathcal{V}(p)$, $\mathcal{V}(q)$, $\mathcal{V}(r)$

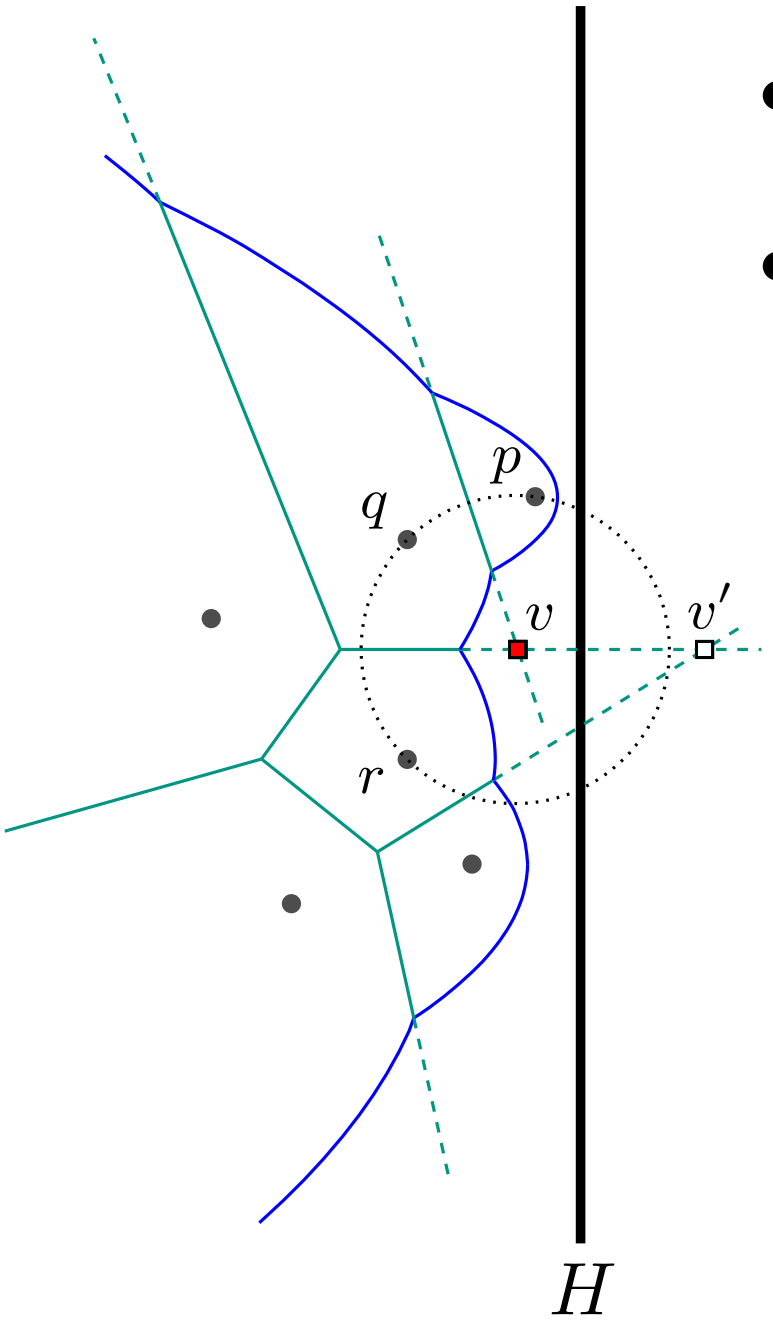
Spike Event



- W hits the intersection between two spikes
  - H leaves the circumcircle.
- A new Voronoi vertex forms
  - v: Voronoi vertex of $\mathcal{V}(p)$, $\mathcal{V}(q)$, $\mathcal{V}(r)$

1. Remove two old spikes
2. Add one new spike
3. Add intersections with adjacent spikes resp.

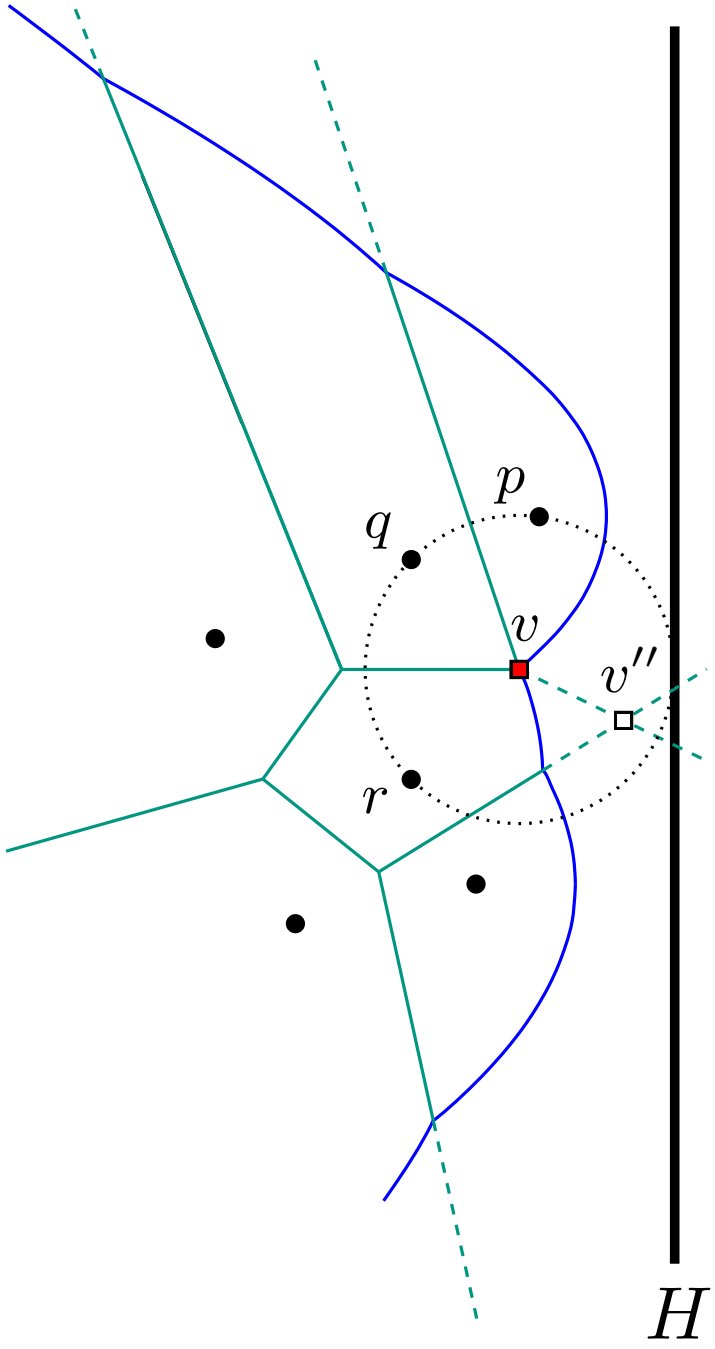
Spike Event



- W hits the intersection between two spikes
  - H leaves the circumcircle.
- A new Voronoi vertex forms
  - v: Voronoi vertex of $\mathcal{V}(p)$, $\mathcal{V}(q)$, $\mathcal{V}(r)$

1. Remove two old spikes
2. Add one new spike
3. Add intersections with adjacent spikes resp.

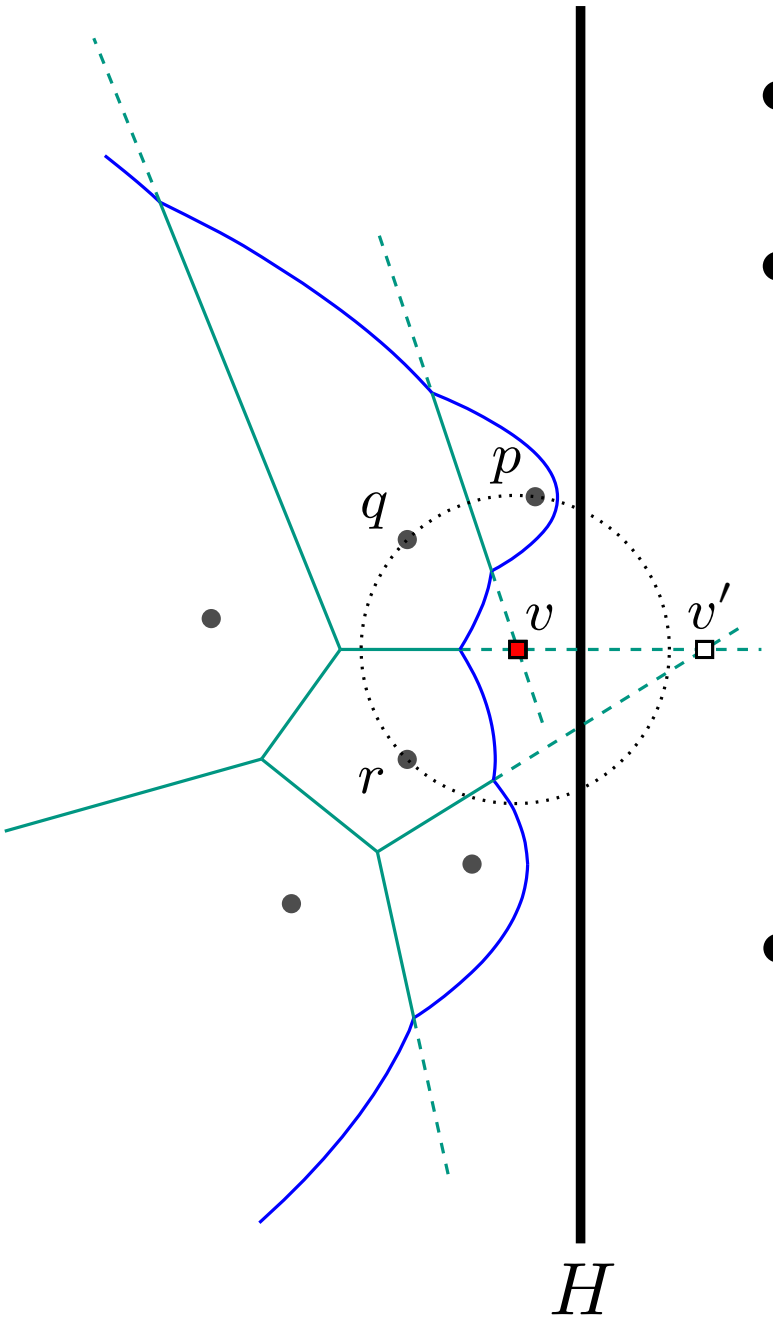
Spike Event



- W hits the intersection between two spikes
  - H leaves the circumcircle.
- A new Voronoi vertex forms
  - v: Voronoi vertex of $\mathcal{V}(p)$, $\mathcal{V}(q)$, $\mathcal{V}(r)$

1. Remove two old spikes
2. Add one new spike
3. Add intersections with adjacent spikes resp.

Spike Event

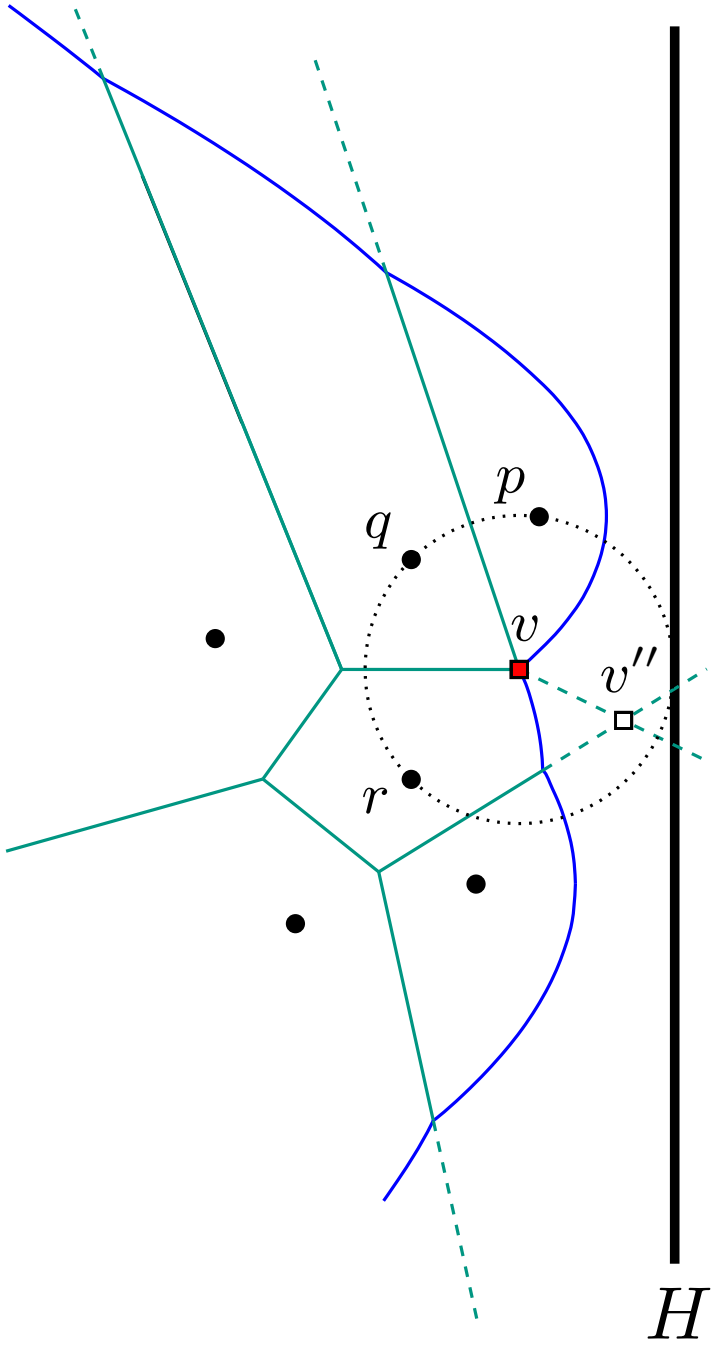


- W hits the intersection between two spikes
  - H leaves the circumcircle.
- A new Voronoi vertex forms
  - v: Voronoi vertex of $\mathcal{V}(p)$, $\mathcal{V}(q)$, $\mathcal{V}(r)$

1. Remove two old spikes
2. Add one new spike
3. Add intersections with adjacent spikes resp.

- One spike only makes one spike event
  - Due to v, v' has no spike event.

Spike Event



- W hits the intersection between two spikes
  - H leaves the circumcircle.
- A new Voronoi vertex forms
  - v: Voronoi vertex of $\mathcal{V}(p)$, $\mathcal{V}(q)$, $\mathcal{V}(r)$

1. Remove two old spikes
2. Add one new spike
3. Add intersections with adjacent spikes resp.

- One spike only makes one spike event
  - Due to v', v has no spike event.

Data Structures

Wavefront

A balanced binary tree

- wavelets
- y-coordinate

Events

A priority queue with respect to x-coordinate

- Site event: x-coordinate of a site
- Spike event: x-coordinate of H when W hits an intersection between two spikes

Construction Time

- A site event takes $O(\log n)$ time
  - $O(1)$ operations on the wavefront
    - Add one wavelet and Separate one old wavelet into two.
  - $O(1)$ operations on the event queue
    - Remove one site event and update two spike events

Construction Time

- A site event takes $O(\log n)$ time
  - $O(1)$ operations on the wavefront
    - Add one wavelet and Separate one old wavelet into two.
  - $O(1)$ operations on the event queue
    - Remove one site event and update two spike events
- A spike event takes $O(\log n)$ time
  - $O(1)$ operations on the wavefront
    - Remove one wavelet
  - $O(1)$ operations on the event queue
    - Remove one spike event and add two spike events

Construction Time

- A site event takes $O(\log n)$ time
  - $O(1)$ operations on the wavefront
    - Add one wavelet and Separate one old wavelet into two.
  - $O(1)$ operations on the event queue
    - Remove one site event and update two spike events
- A spike event takes $O(\log n)$ time
  - $O(1)$ operations on the wavefront
    - Remove one wavelet
  - $O(1)$ operations on the event queue
    - Remove one spike event and update two spike events
- $O(n)$ site events and $O(n)$ spike events
  - one spike makes only one spike event
  - one spike corresponds to one Voronoi edge

Construction Time

- A site event takes $O(\log n)$ time
  - $O(1)$ operations on the wavefront
    - Add one wavelet and Separate one old wavelet into two.
  - $O(1)$ operations on the event queue
    - Remove one site event and update two spike events
- A spike event takes $O(\log n)$ time
  - $O(1)$ operations on the wavefront
    - Remove one wavelet
  - $O(1)$ operations on the event queue
    - Remove one spike event and update two spike events
- $O(n)$ site events and $O(n)$ spike events
  - one spike makes only one spike event
  - one spike corresponds to one Voronoi edge
- $O(n \log n)$ time

Correctness

Theorem 8:

For a point $x \in \mathbb{R}^2$, if $x \in \mathcal{V}(p, S_L \cup \{H\})$, $x \in \mathcal{V}(p, P)$

Proof:

Correctness

Theorem 8:

For a point $x \in \mathbb{R}^2$, if $x \in \mathcal{V}(p, S_L \cup \{H\})$, $x \in \mathcal{V}(p, P)$

Proof:

- Let P_R be $P \setminus P_L$

Correctness

Theorem 8:

For a point $x \in \mathbb{R}^2$, if $x \in \mathcal{V}(p, S_L \cup \{H\})$, $x \in \mathcal{V}(p, P)$

Proof:

- Let P_R be $P \setminus P_L$
- $d(x,p) < d(x,q)$, $\forall q \in P_L \setminus \{p\}$ and $d(x,p) < d(x,H)$

Correctness

Theorem 8:

For a point $x \in \mathbb{R}^2$, if $x \in \mathcal{V}(p, S_L \cup \{H\})$, $x \in \mathcal{V}(p, P)$

Proof:

- Let P_R be $P \setminus P_L$
- $d(x,p) < d(x,q)$, $\forall q \in P_L \setminus \{p\}$ and $d(x,p) < d(x,H)$
- $d(x,H) \leq d(x,q')$, $\forall q' \in P_R$

Correctness

Theorem 8:

For a point $x \in \mathbb{R}^2$, if $x \in \mathcal{V}(p, S_L \cup \{H\})$, $x \in \mathcal{V}(p, P)$

Proof:

- Let P_R be $P \setminus P_L$
- $d(x,p) < d(x,q)$, $\forall q \in P_L \setminus \{p\}$ and $d(x,p) < d(x,H)$
- $d(x,H) \leq d(x,q')$, $\forall q' \in P_R$
- $d(x,p) < d(x,q')$, $\forall q' \in P_R$

Correctness

Theorem 8:

For a point $x \in \mathbb{R}^2$, if $x \in \mathcal{V}(p, S_L \cup \{H\})$, $x \in \mathcal{V}(p, P)$

Proof:

- Let P_R be $P \setminus P_L$
- $d(x,p) < d(x,q)$, $\forall q \in P_L \setminus \{p\}$ and $d(x,p) < d(x,H)$
- $d(x,H) \leq d(x,q')$, $\forall q' \in P_R$
- $d(x,p) < d(x,q')$, $\forall q' \in P_R$
- $d(x,p) < d(x,q)$, $\forall q \in P \setminus \{p\}$

Divide and Conquer

- Basic Steps
  1. Divide an instance into c equal-size sub-instances
     - If the instance is extremely small, solve it directly
  2. Recursively compute the sub-solution for each sub-instance
     - Until a sub-instance can be solved in $O(1)$ time
  3. Merge all the c sub-solutions into one solution

Divide and Conquer

- Basic Steps
  1. Divide an instance into c equal-size sub-instances
     - If the instance is extremely small, solve it directly
  2. Recursively compute the sub-solution for each sub-instance
     - Until a sub-instance can be solved in $O(1)$ time
  3. Merge all the c sub-solutions into one solution

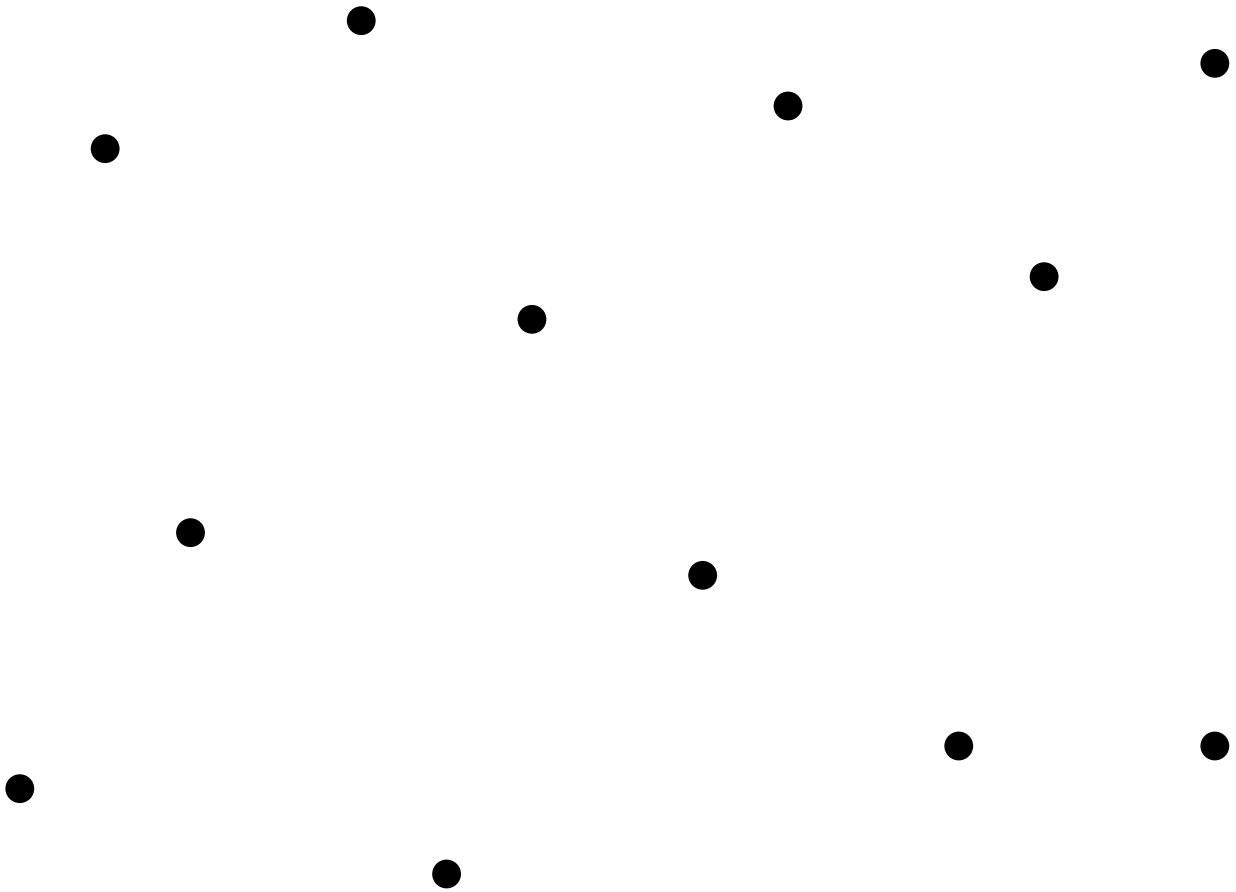
- If both Divide and Merge take linear time,

$$T(n) = c \cdot T(n/c) + O(n) => O(n \log n)$$

  - The i^{th} level has c^i parts and each part has n/c^i elements
  - A level takes $c^i \times O(n/c^i) = O(n)$ time
  - $O(\log n)$ levels

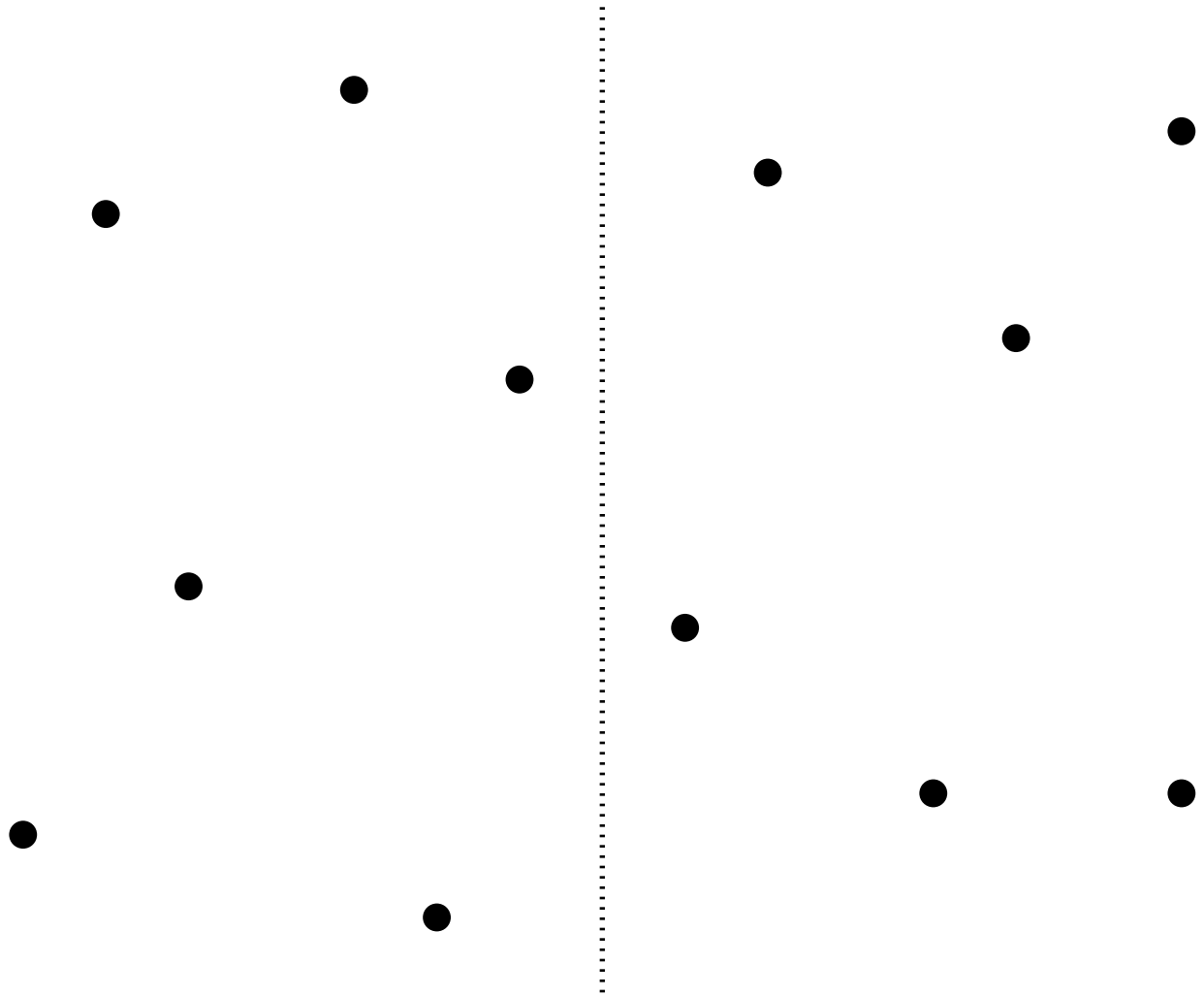
Example: Convex Hull

- $n = 12$ and $c = 2$



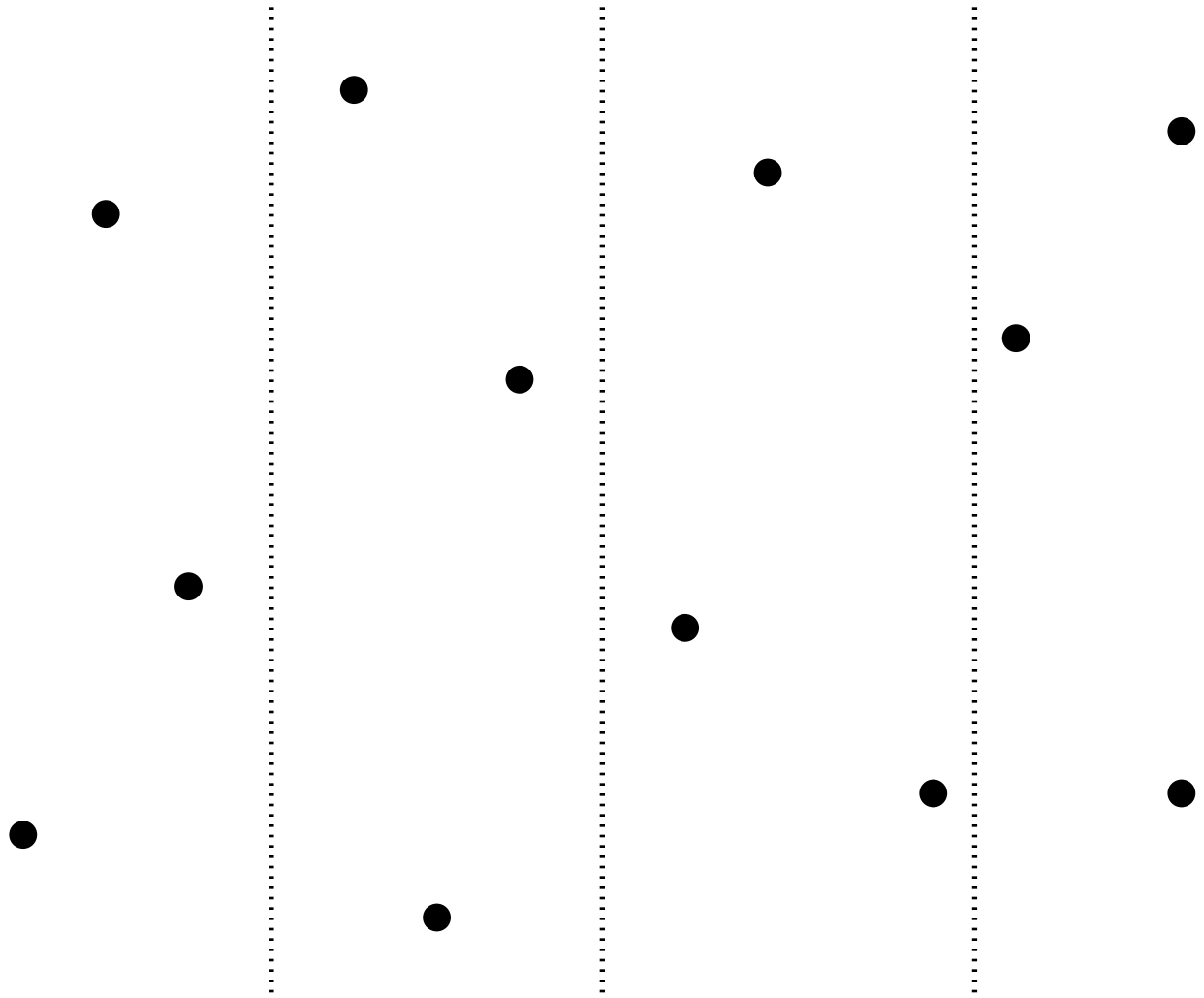
Example: Convex Hull

- $n = 12$ and $c = 2$



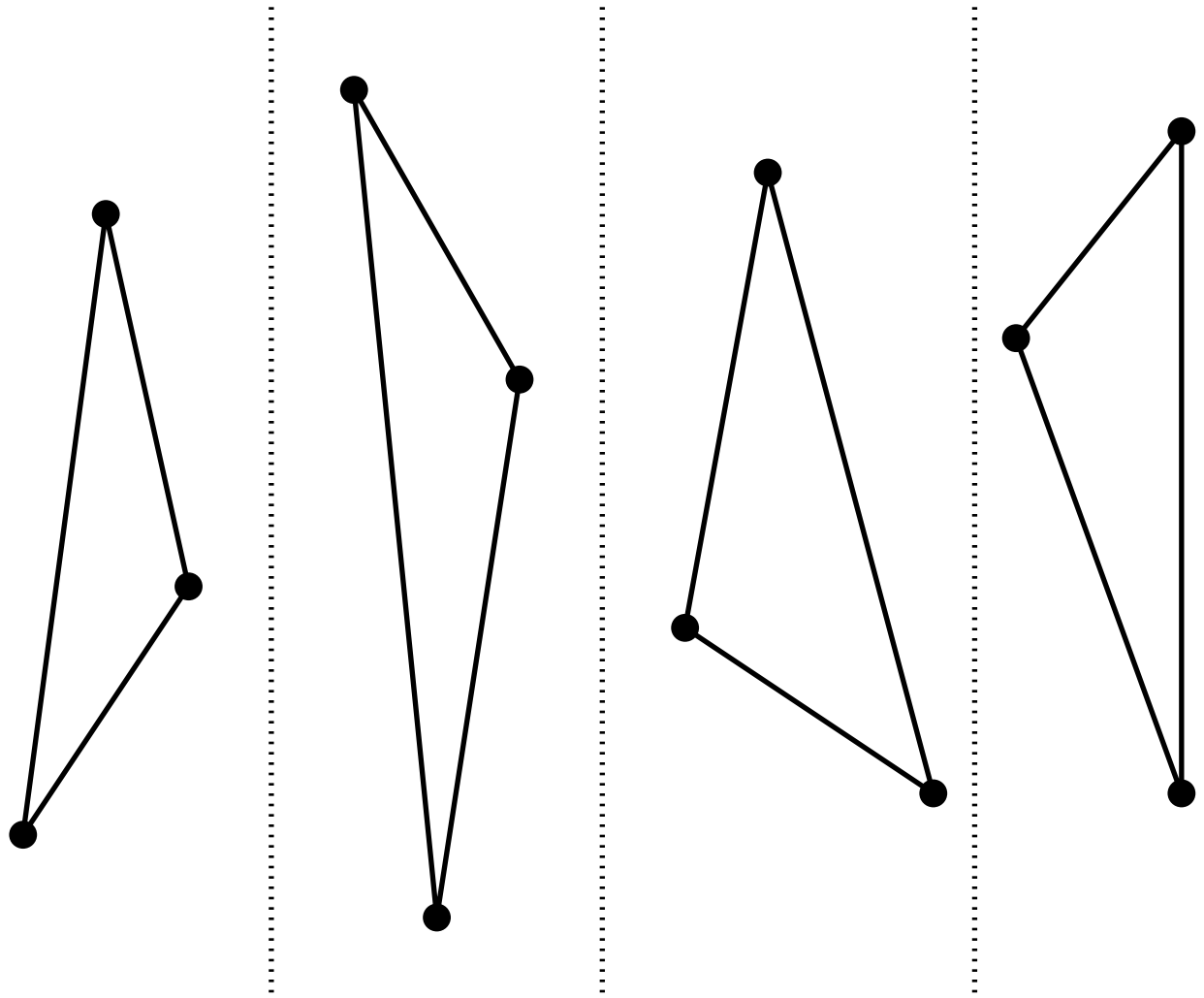
Example: Convex Hull

- $n = 12$ and $c = 2$



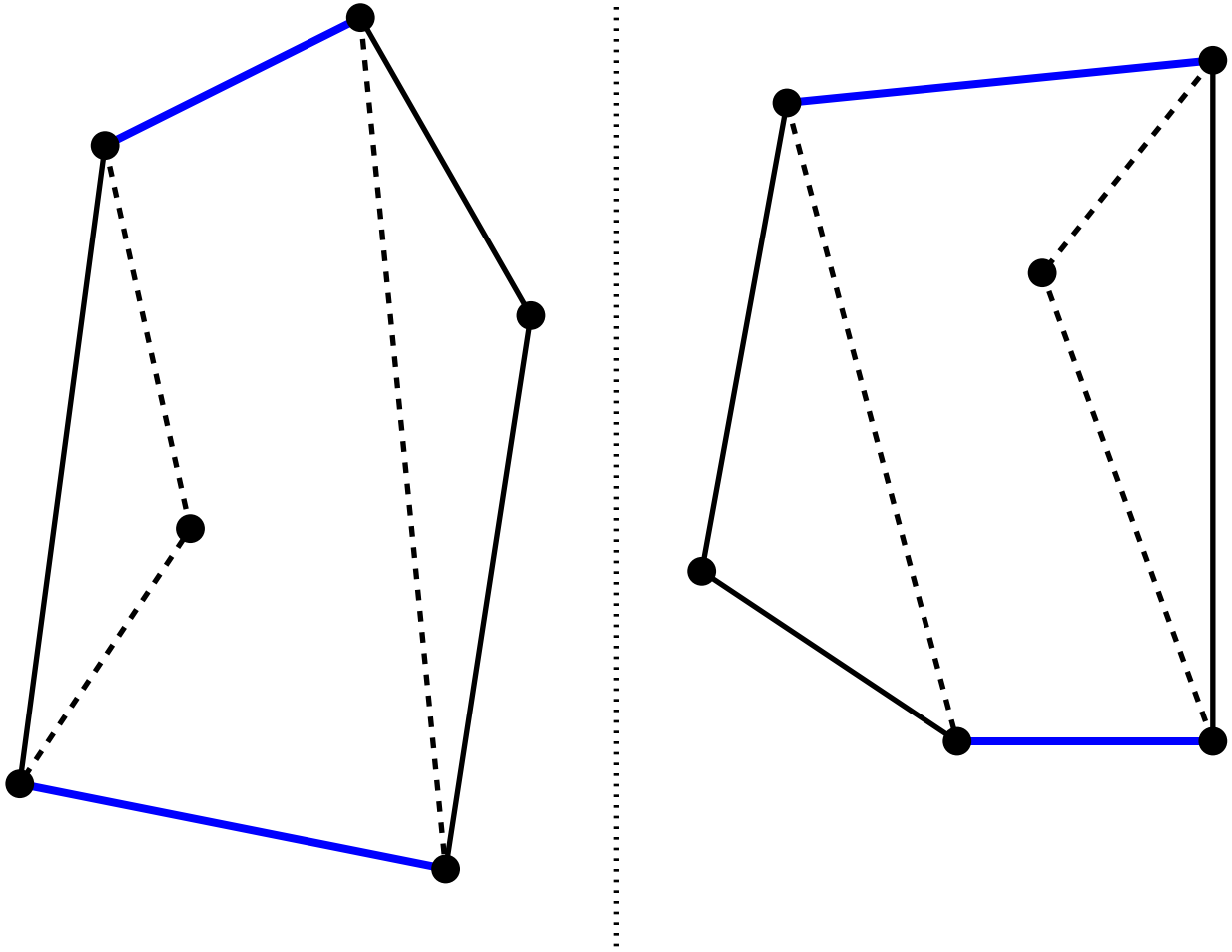
Example: Convex Hull

- $n = 12$ and $c = 2$



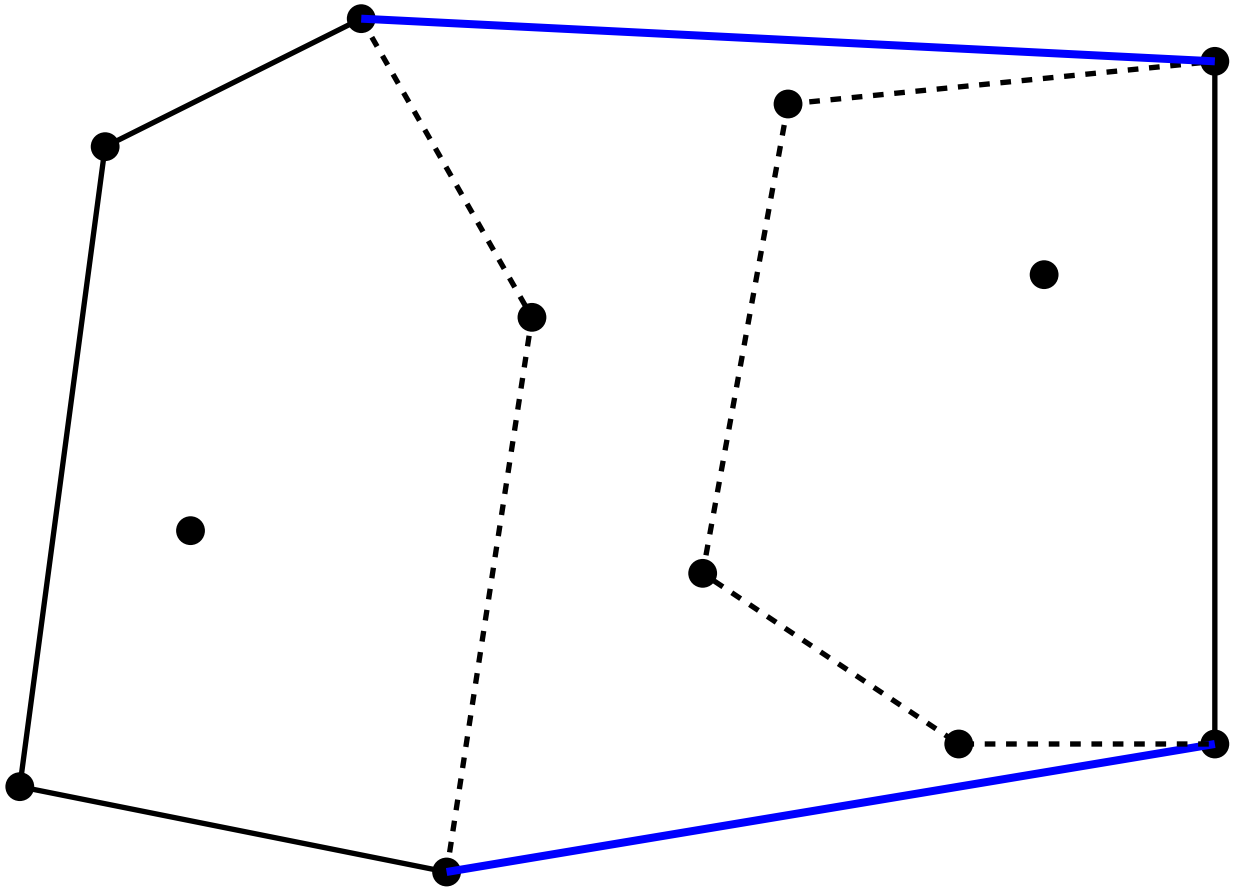
Example: Convex Hull

- $n = 12$ and $c = 2$



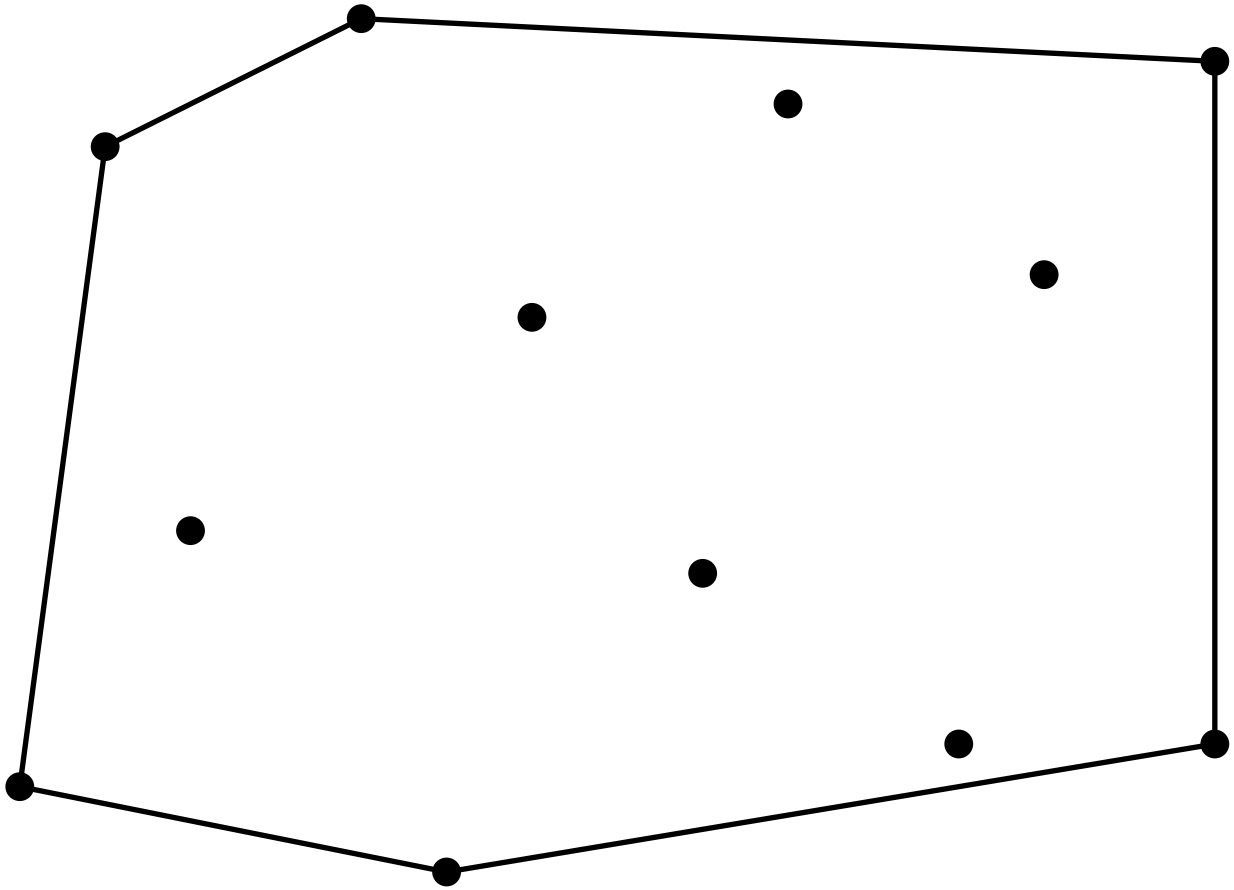
Example: Convex Hull

- $n = 12$ and $c = 2$



Example: Convex Hull

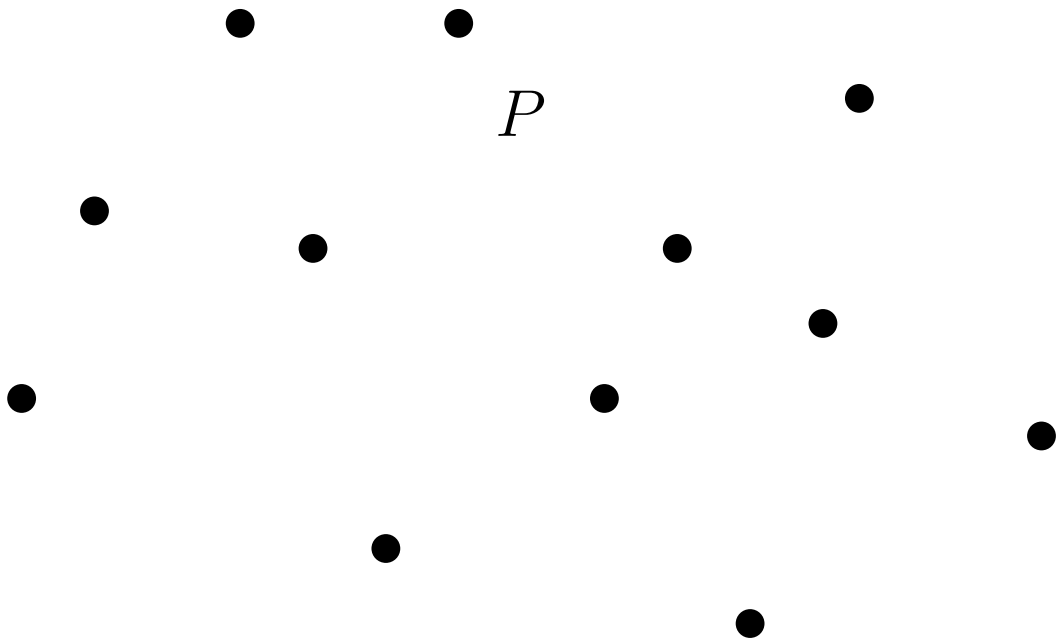
- $n = 12$ and $c = 2$



D & C for Voronoi Diagram

Divide and Conquer to Compute $\text{Vor}(P)$

1. Use a vertical line to partition P into L and R with $|L| \sim |R|$
   - If $|P|$ is a constant, directly compute $\text{Vor}(P)$ instead
2. Recursively compute $\text{Vor}(L)$ and $\text{Vor}(R)$
3. Merge $\text{Vor}(L)$ and $\text{Vor}(R)$ into $\text{Vor}(P)$

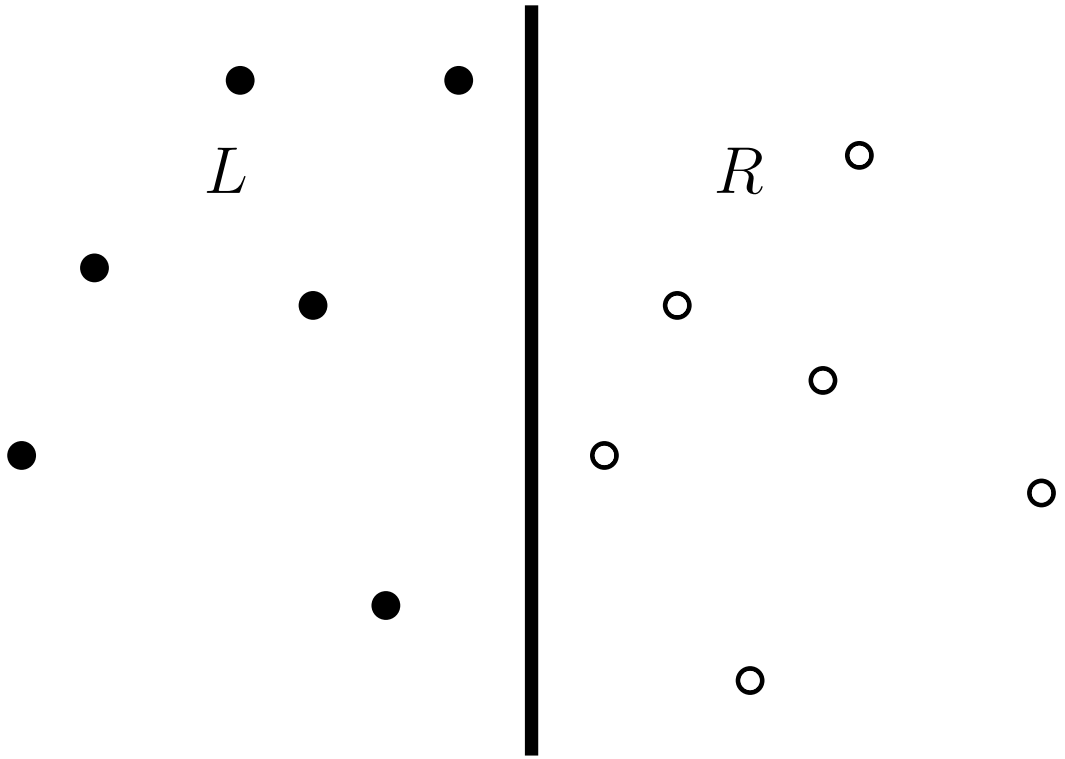


D & C for Voronoi Diagram

Karlsruhe Institute of Technology

Divide and Conquer to Compute $\mathrm{Vor}(P)$

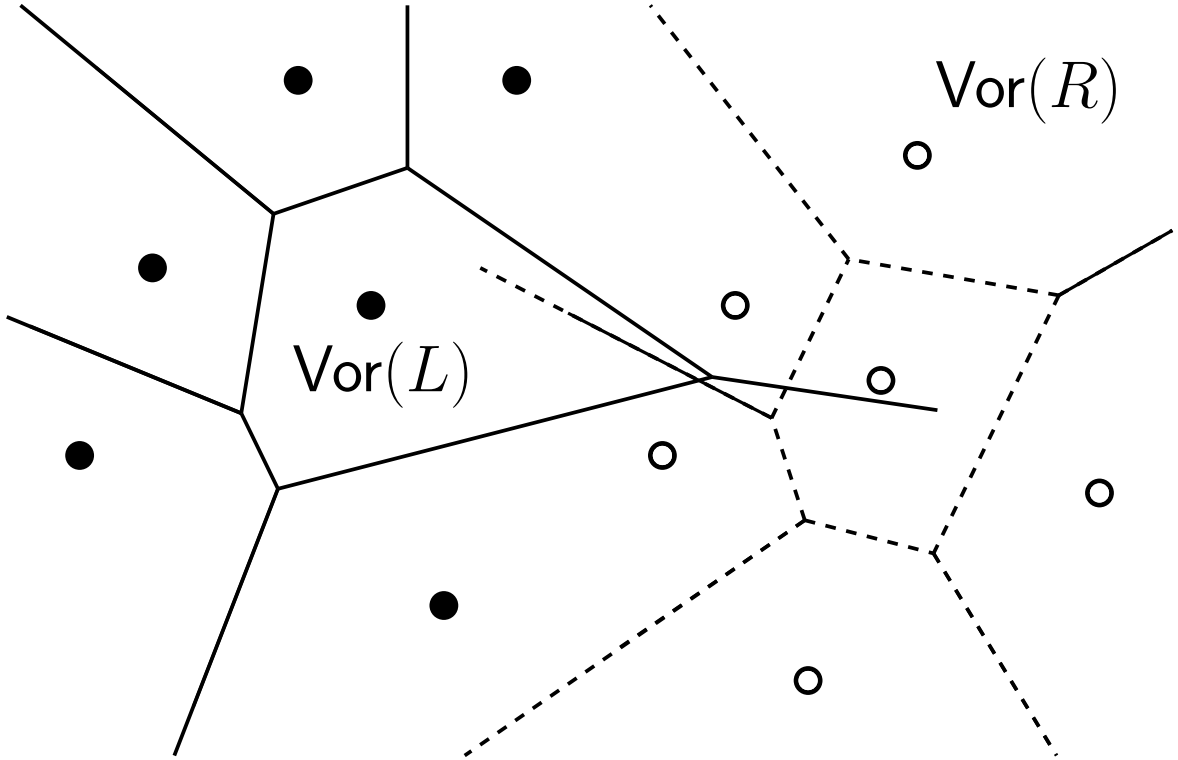
1. Use a vertical line to partition P into L and R with $|L| \sim |R|$
   - If $|P|$ is a constant, directly compute $\mathrm{Vor}(P)$ instead
2. Recursively compute $\mathrm{Vor}(L)$ and $\mathrm{Vor}(R)$
3. Merge $\mathrm{Vor}(L)$ and $\mathrm{Vor}(R)$ into $\mathrm{Vor}(P)$



D & C for Voronoi Diagram

Divide and Conquer to Compute $\mathrm{Vor}(P)$

1. Use a vertical line to partition P into L and R with $|L| \sim |R|$
   - If $|P|$ is a constant, directly compute $\mathrm{Vor}(P)$ instead
2. Recursively compute $\mathrm{Vor}(L)$ and $\mathrm{Vor}(R)$
3. Merge $\mathrm{Vor}(L)$ and $\mathrm{Vor}(R)$ into $\mathrm{Vor}(P)$

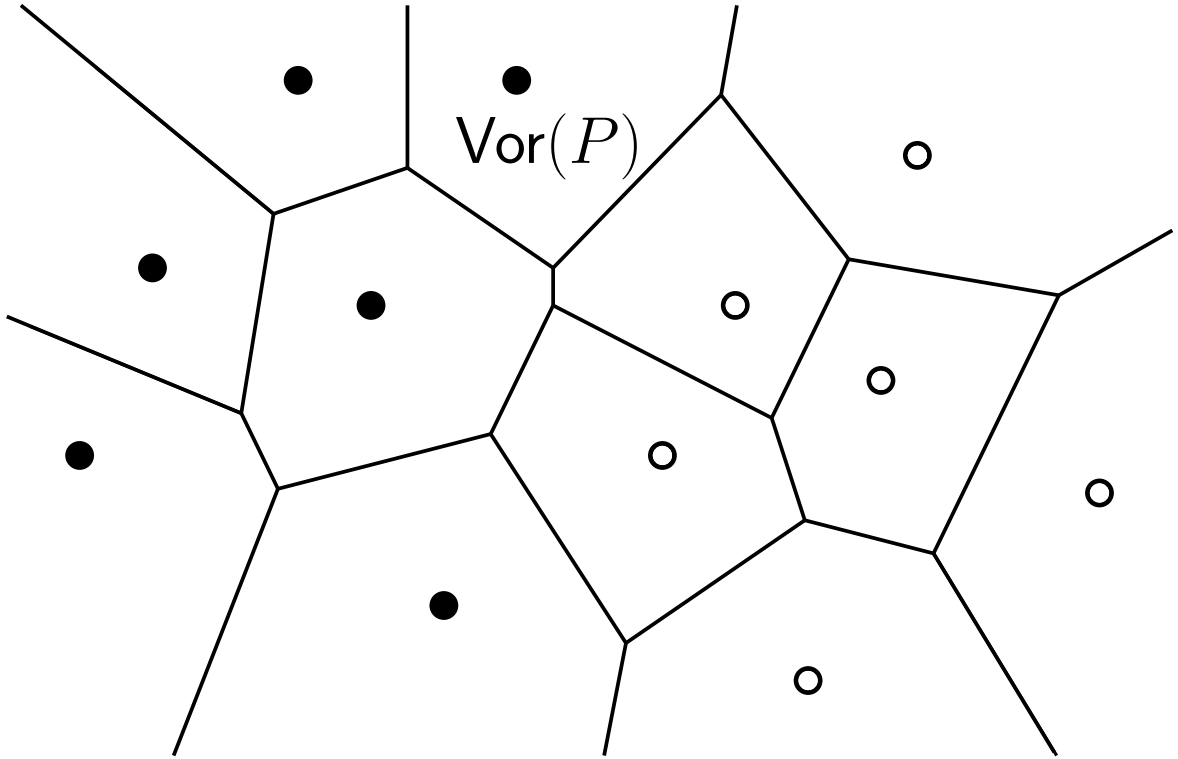


D & C for Voronoi Diagram

Karlsruhe Institute of Technology

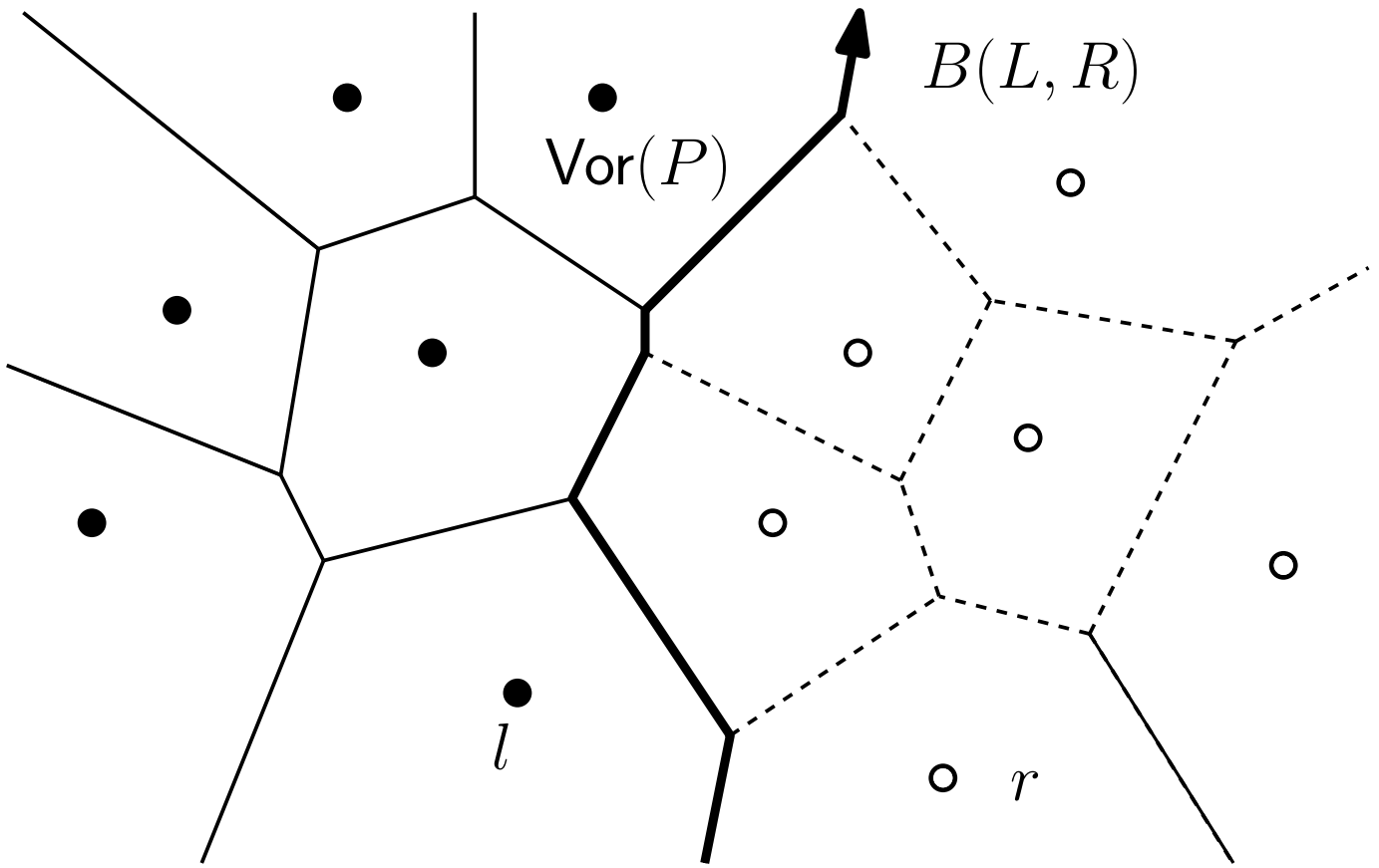
Divide and Conquer to Compute $\mathrm{Vor}(P)$

1. Use a vertical line to partition P into L and R with $|L| \sim |R|$
   - If $|P|$ is a constant, directly compute $\mathrm{Vor}(P)$ instead
2. Recursively compute $\mathrm{Vor}(L)$ and $\mathrm{Vor}(R)$
3. Merge $\mathrm{Vor}(L)$ and $\mathrm{Vor}(R)$ into $\mathrm{Vor}(P)$



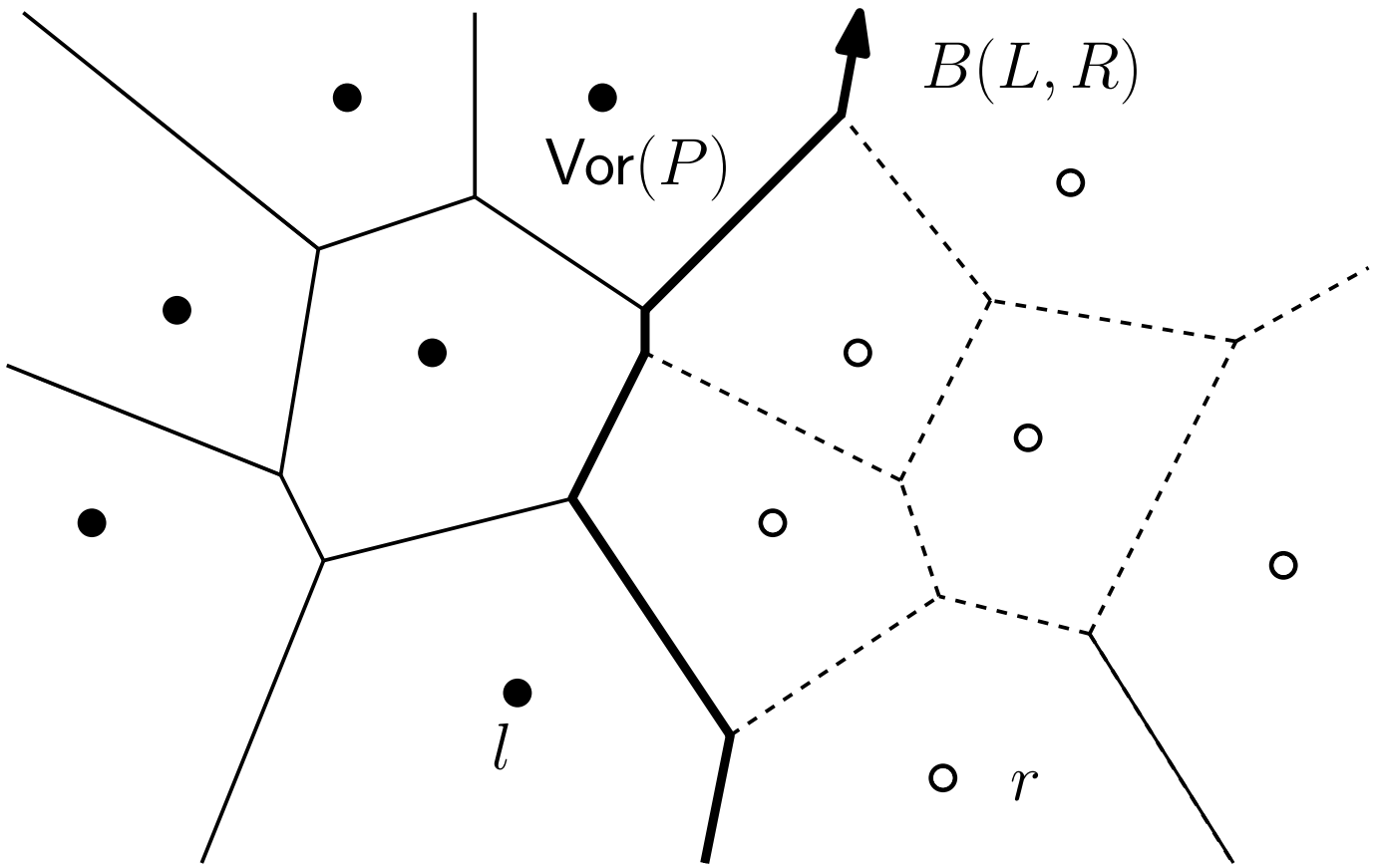
Merge Chain

- **Merge Chain** $B(L, R)$ consists of Voronoi edges between $\mathcal{V}(l)$ and $\mathcal{V}(r)$ where $l \in L$ and $r \in R$.



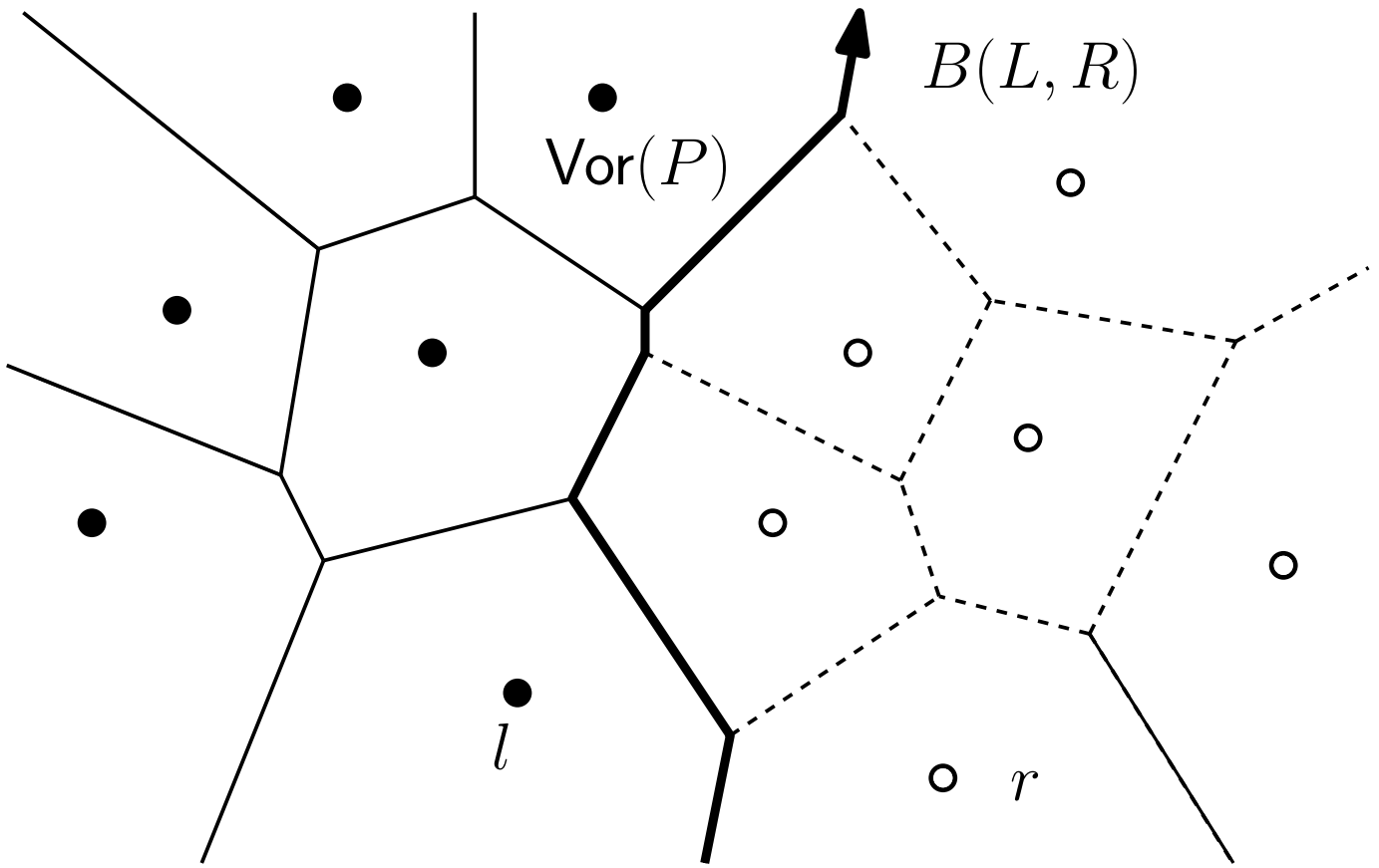
Merge Chain

- **Merge Chain** $B(L, R)$ consists of Voronoi edges between $\mathcal{V}(l)$ and $\mathcal{V}(r)$ where $l \in L$ and $r \in R$.
  - Voronoi edges between L's cells and R's cells



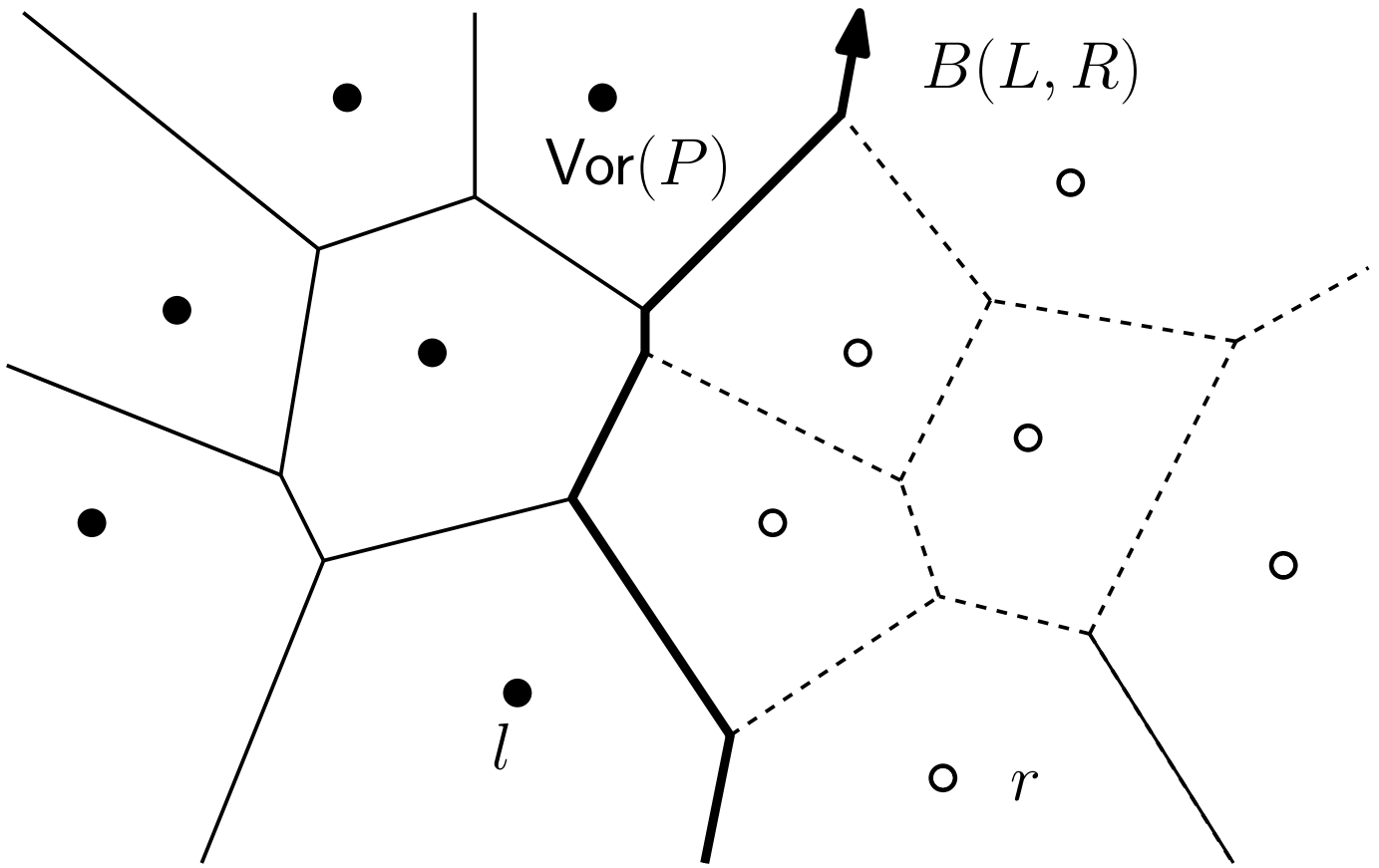
Merge Chain

- **Merge Chain** $B(L, R)$ consists of Voronoi edges between $\mathcal{V}(l)$ and $\mathcal{V}(r)$ where $l \in L$ and $r \in R$.
  - Voronoi edges between L's cells and R's cells
  - $d(x, L) = \min_{l \in L} d(x, l)$ (reps. $d(x, R)$)



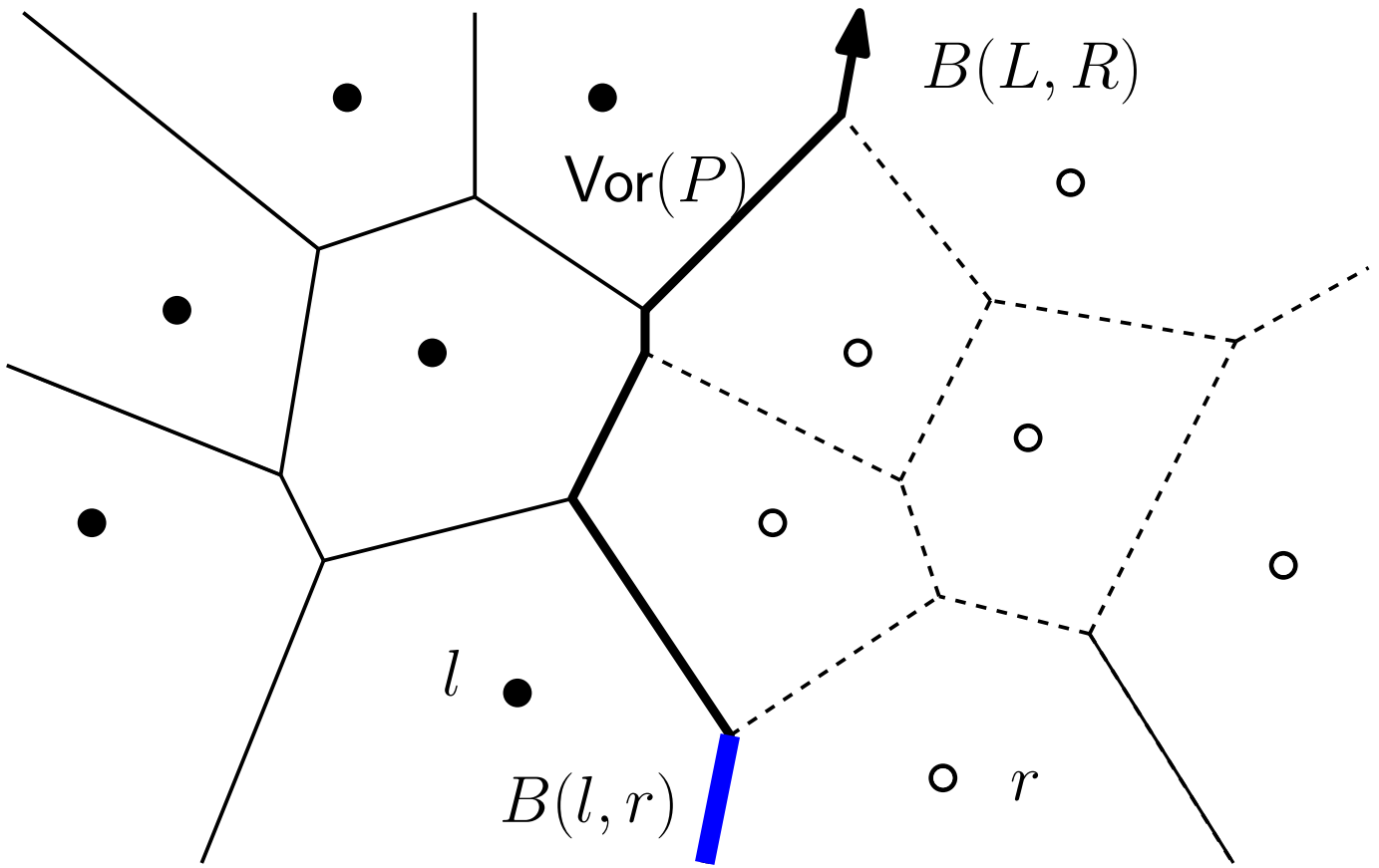
Merge Chain

- **Merge Chain** $B(L, R)$ consists of Voronoi edges between $\mathcal{V}(l)$ and $\mathcal{V}(r)$ where $l \in L$ and $r \in R$.
  - Voronoi edges between L's cells and R's cells
  - $d(x, L) = \min_{l \in L} d(x, l)$ (reps. $d(x, R)$)
  - $B(L, R) = \{x \in R^2 \mid d(x, L) = d(x, R)\}$



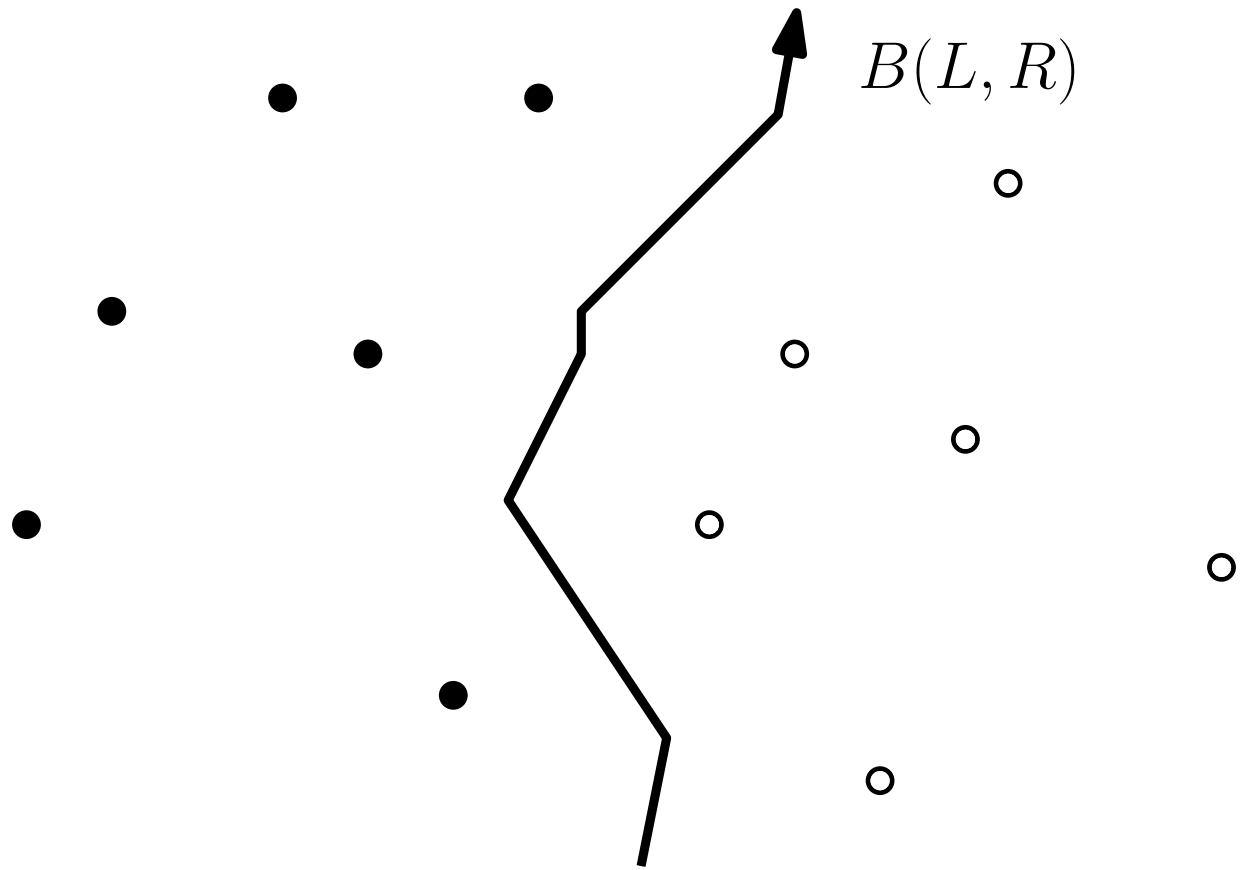
Merge Chain

- **Merge Chain** $B(L, R)$ consists of Voronoi edges between $\mathcal{V}(l)$ and $\mathcal{V}(r)$ where $l \in L$ and $r \in R$.
  - Voronoi edges between L's regions and R's regions
  - $d(x, L) = \min_{l \in L} d(x, l)$ (reps. $d(x, R)$)
  - $B(L, R) = \{x \in R^2 \mid d(x, L) = d(x, R)\}$
    - $\forall e \in B(L, R)$ belongs to $B(l, r)$, $l \in L$ and $r \in R$



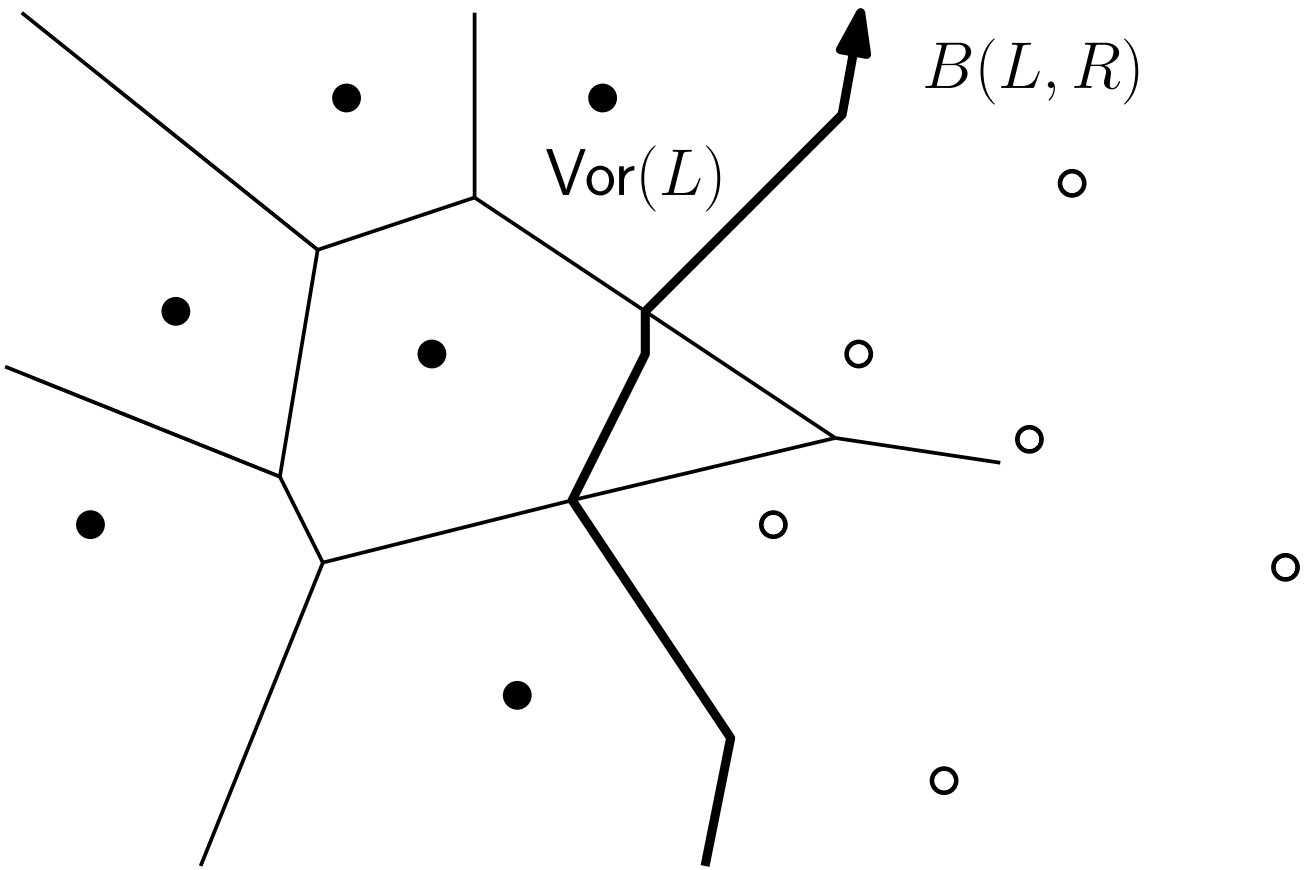
If We Have the Merge Chain...

- Compute Vor(P)
  - Remove the part of Vor(L) right to $B(L, R)$
  - Remove the part of Vor(R) left to $B(L, R)$
  - Glue the two remaining parts



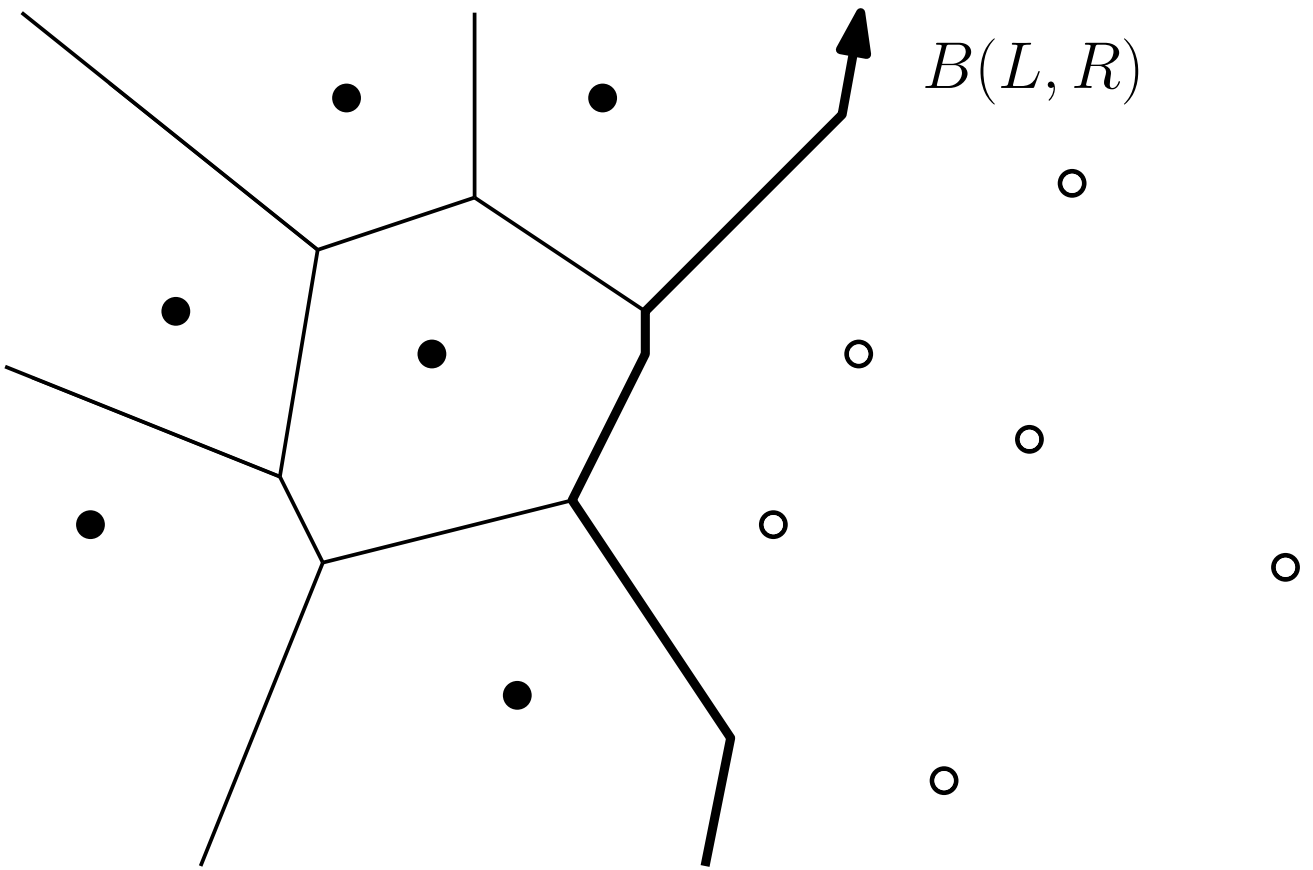
If We Have the Merge Chain...

- Compute $\mathrm{Vor}(P)$
  - Remove the part of $\mathrm{Vor}(L)$ right to $B(L, R)$
  - Remove the part of $\mathrm{Vor}(R)$ left to $B(L, R)$
  - Glue the two remaining parts



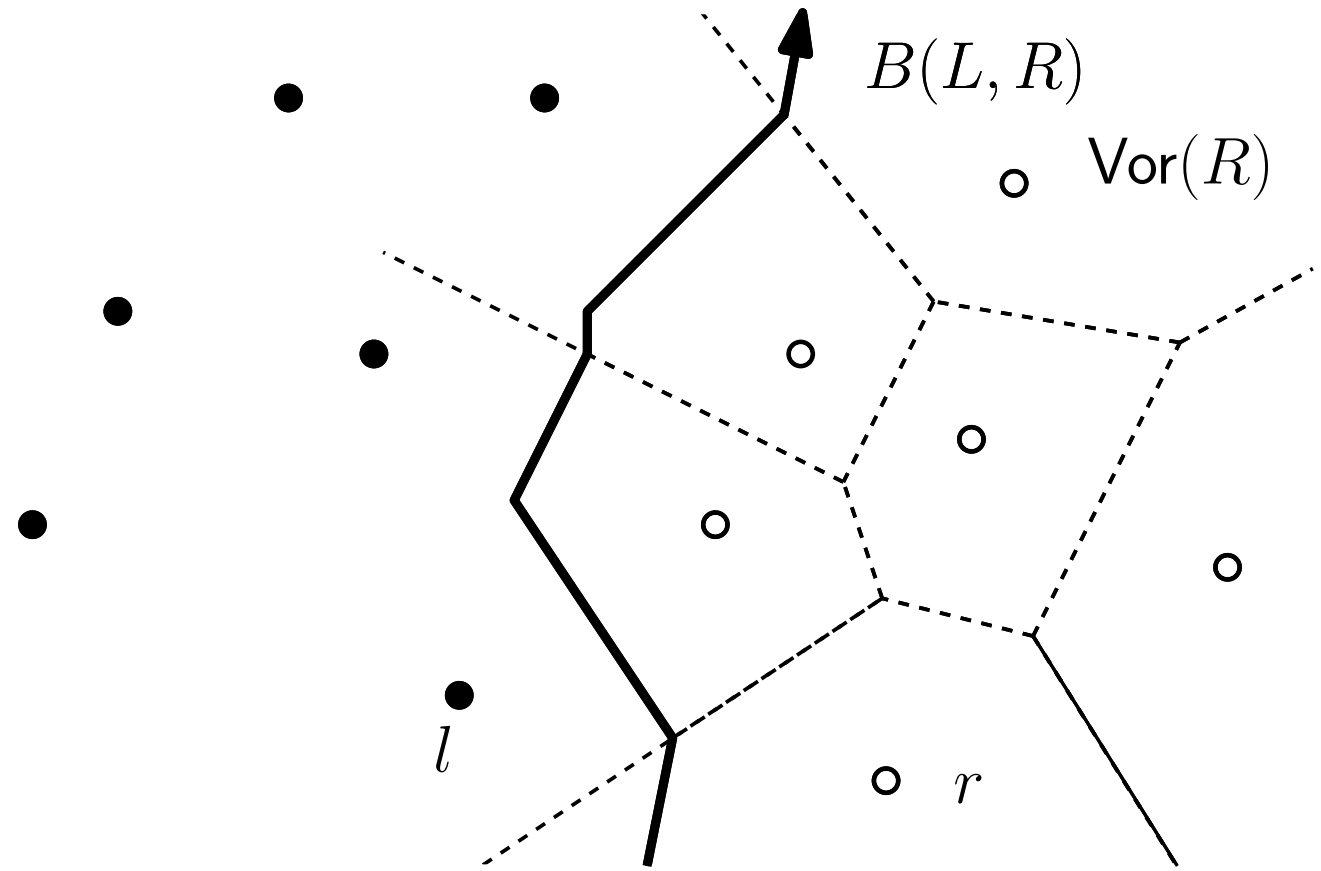
If We Have the Merge Chain...

- Compute Vor(P)
  - Remove the part of Vor(L) right to $B(L, R)$
  - Remove the part of Vor(R) left to $B(L, R)$
  - Glue the two remaining parts



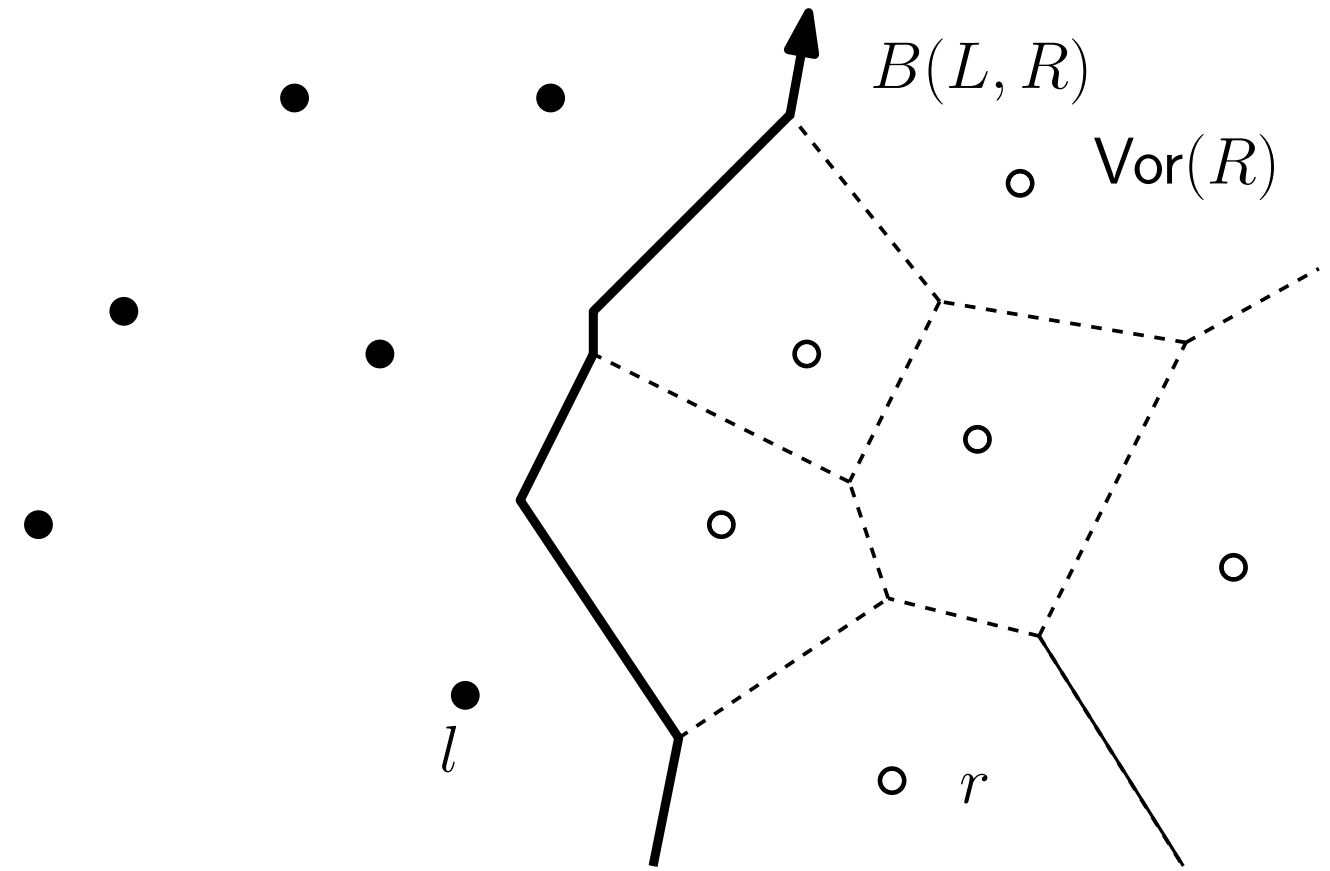
If We Have the Merge Chain...

- Compute Vor(P)
  - Remove the part of Vor(L) right to $B(L, R)$
  - Remove the part of Vor(R) left to $B(L, R)$
  - Glue the two remaining parts



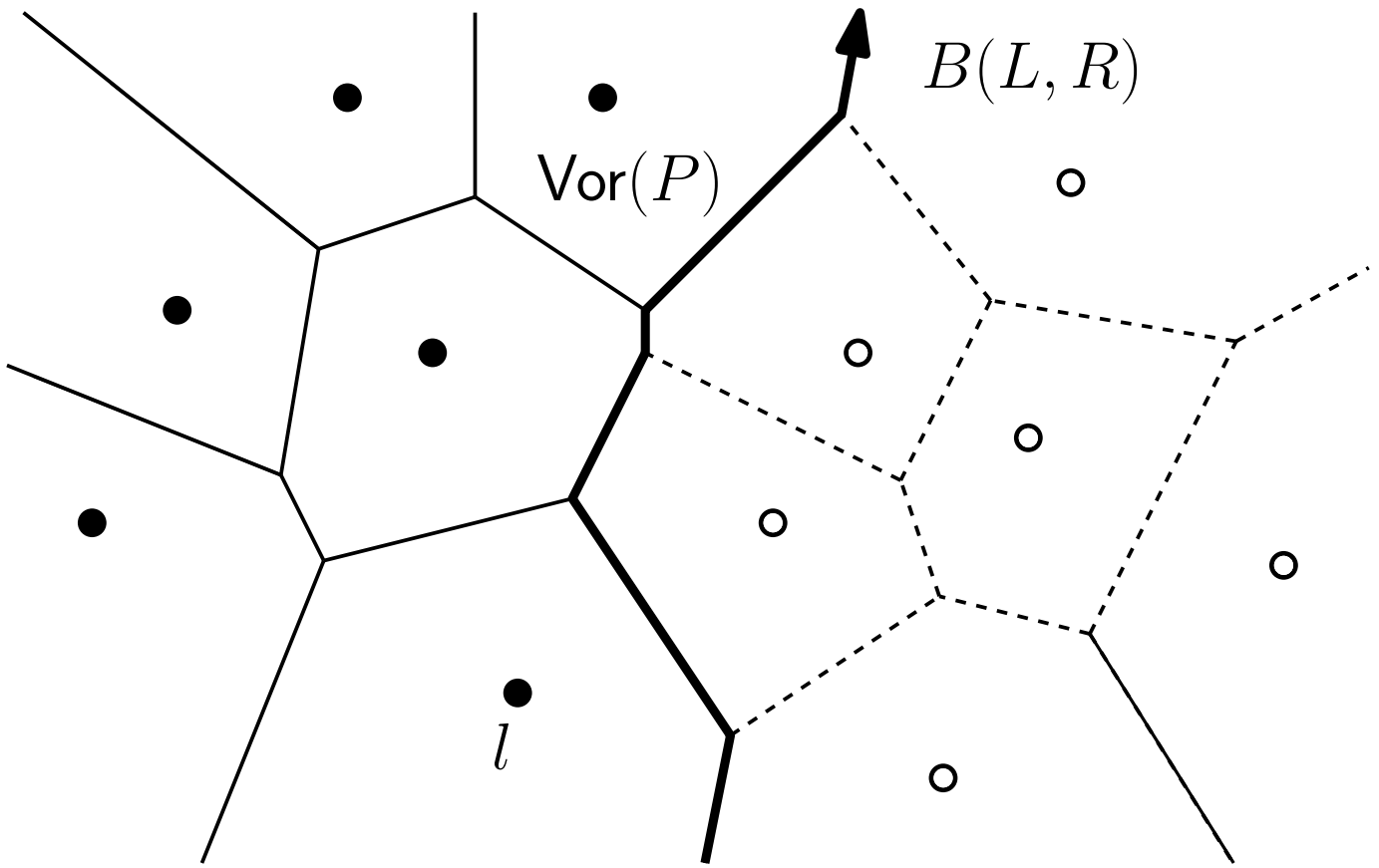
If We Have the Merge Chain...

- Compute Vor(P)
  - Remove the part of Vor(L) right to $B(L, R)$
  - Remove the part of Vor(R) left to $B(L, R)$
  - Glue the two remaining parts



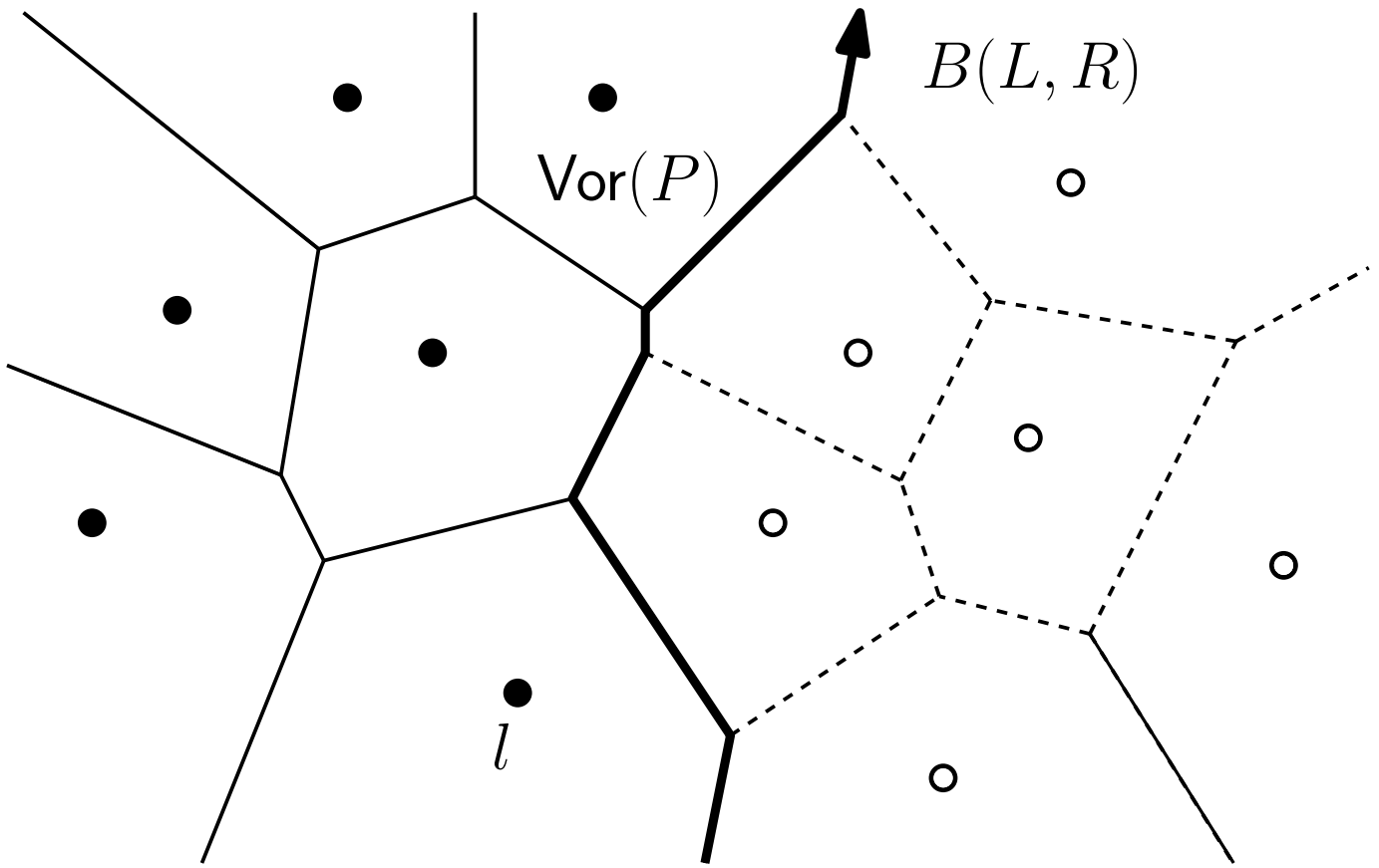
If We Have the Merge Chain...

- Compute $\mathrm{Vor}(P)$
  - Remove the part of $\mathrm{Vor}(L)$ right to $B(L, R)$
  - Remove the part of $\mathrm{Vor}(R)$ left to $B(L, R)$
  - Glue the two remaining parts



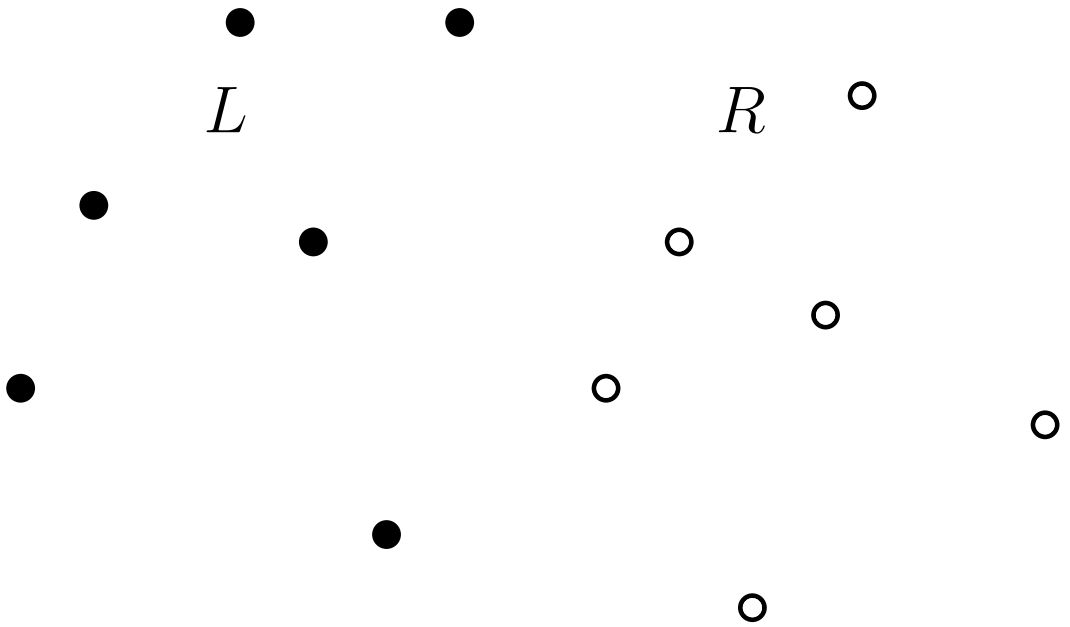
If We Have the Merge Chain...

- Compute Vor(P)
  - Remove the part of Vor(L) right to $B(L, R)$
  - Remove the part of Vor(R) left to $B(L, R)$
  - Glue the two remaining parts
- $B(L, R)$ is a part of Vor(P)



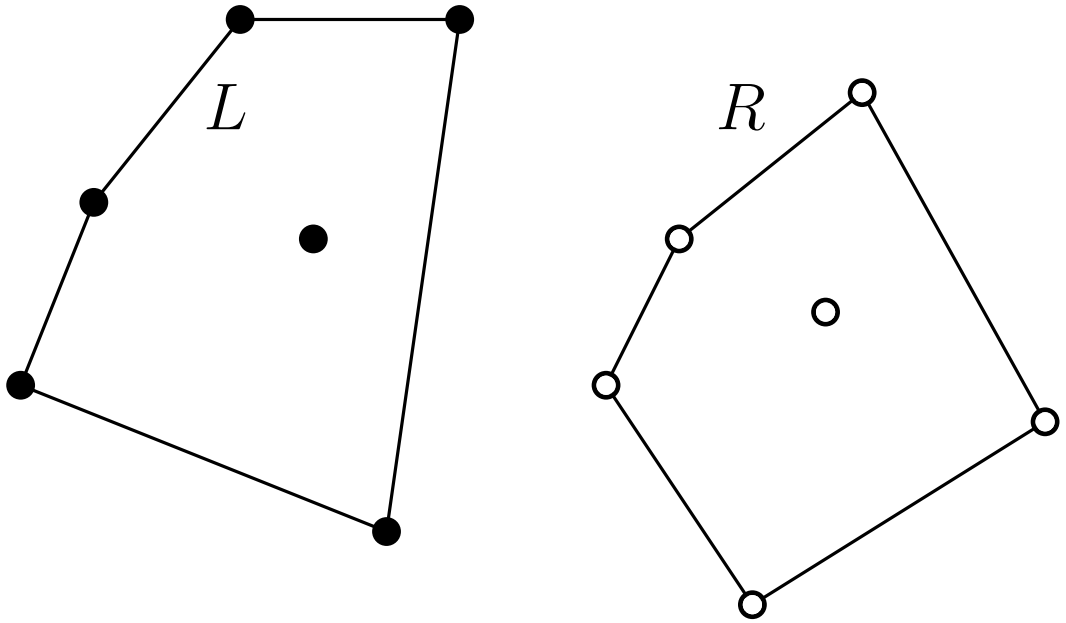
Finding a starting edge of $B(L, R)$

- An unbounded Voronoi edge $\leftrightarrow$ an edge of the convex hull $conv(P)$



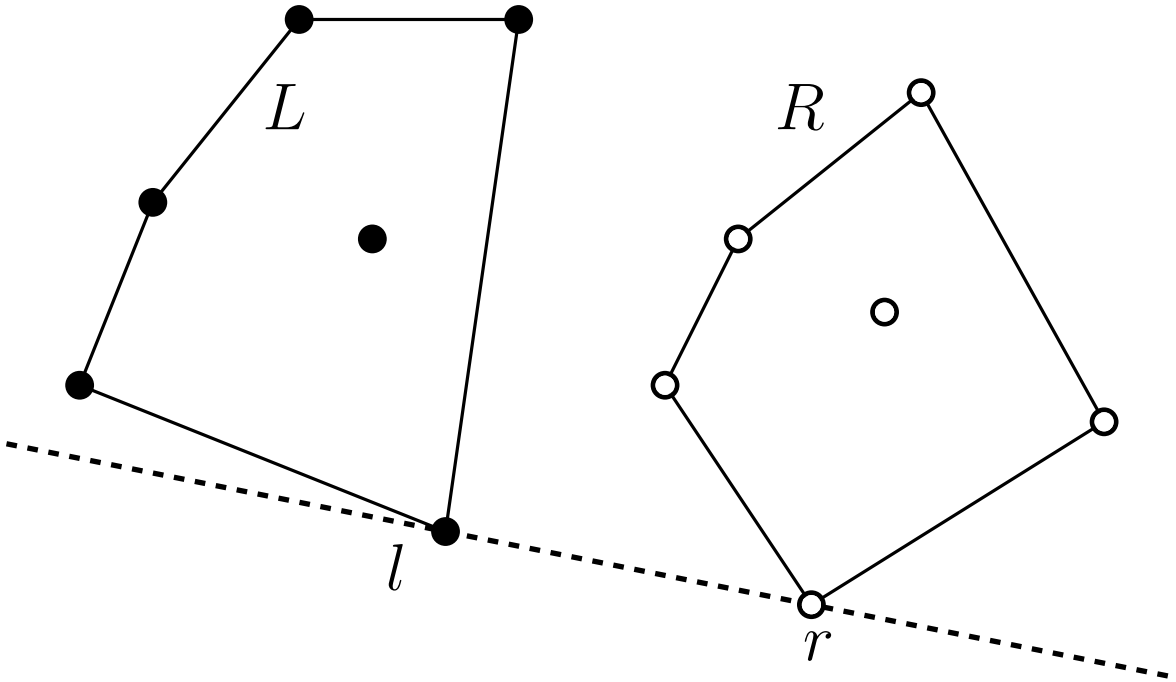
Finding a starting edge of $B(L, R)$

- An unbounded Voronoi edge $\leftrightarrow$ an edge of the convex hull $conv(P)$
  - Build $conv(L)$ and $conv(R)$
    - $*$ Vor(L) and Vor(R) are known.



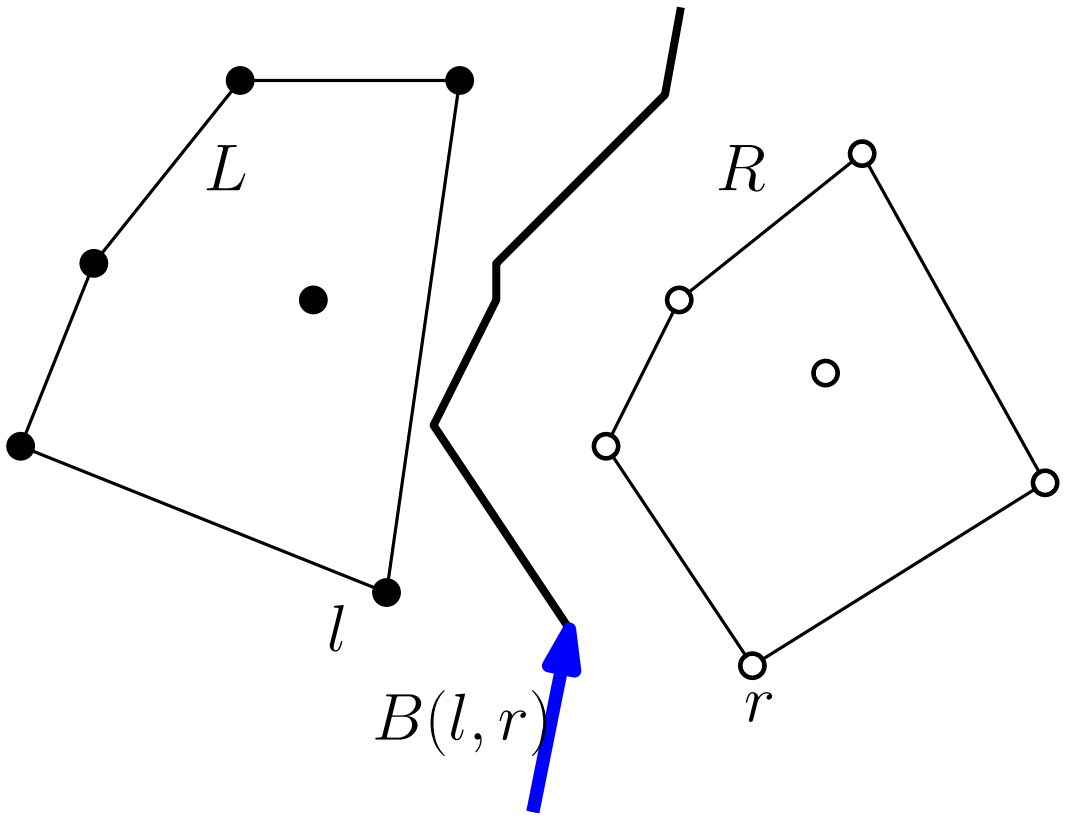
Finding a starting edge of $B(L, R)$

- An unbounded Voronoi edge $\leftrightarrow$ an edge of the convex hull $conv(P)$
  - Build $conv(L)$ and $conv(R)$
    - $*$ Vor(L) and Vor(R) are known.
  - Compute a tangent edge $\overline{lr}$ between $conv(L)$ and $conv(R)$



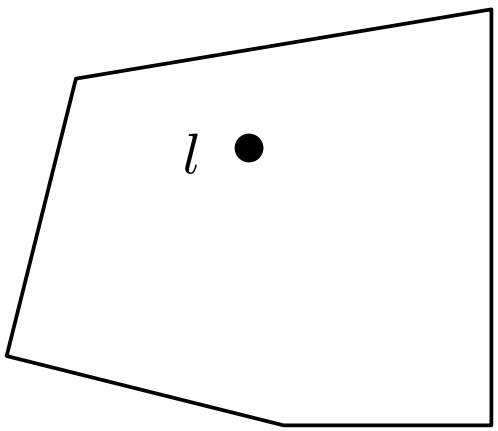
Finding a starting edge of $B(L, R)$

- An unbounded Voronoi edge $\leftrightarrow$ an edge of the convex hull $conv(P)$
  - Build $conv(L)$ and $conv(R)$
    - $*$ Vor(L) and Vor(R) are known.
  - Compute a tangent edge $\overline{lr}$ between $conv(L)$ and $conv(R)$
  - $B(l, r)$ is the starting edge



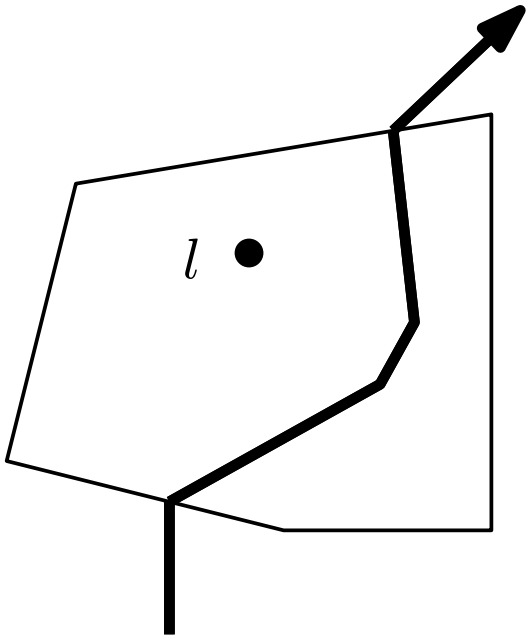
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (1)

$B(L, R) \cap \mathcal{V}(l, L)$



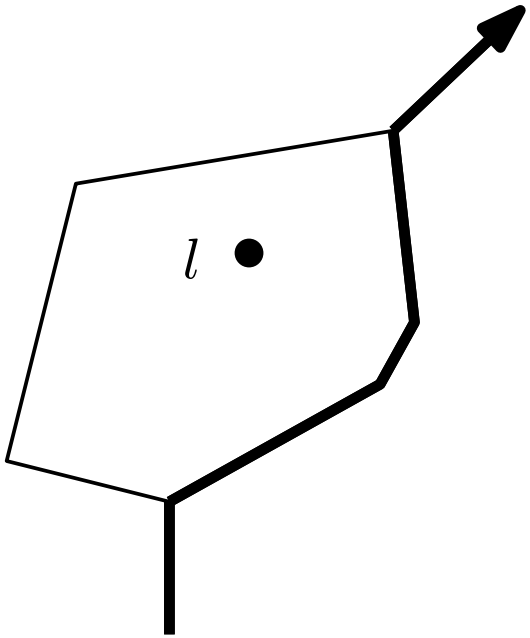
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (1)

$B(L, R) \cap \mathcal{V}(l, L)$



Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (1)

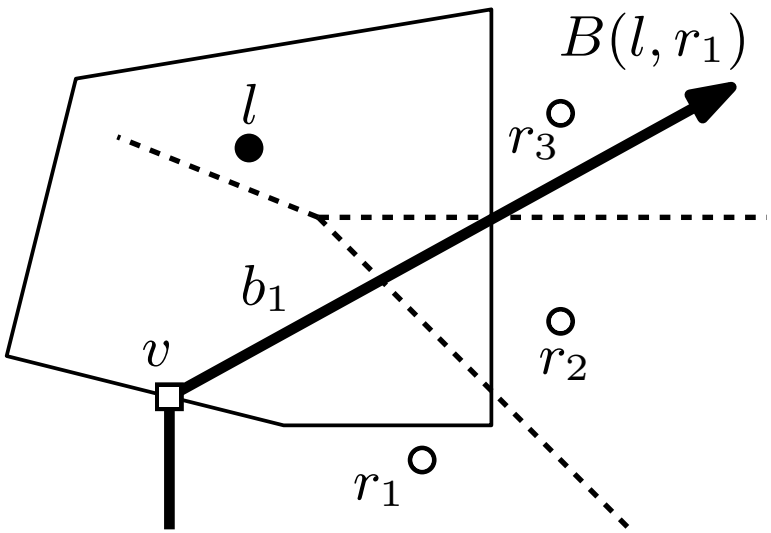
$B(L, R) \cap \mathcal{V}(l, L)$



Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (1)

$B(L, R) \cap \mathcal{V}(l, L)$

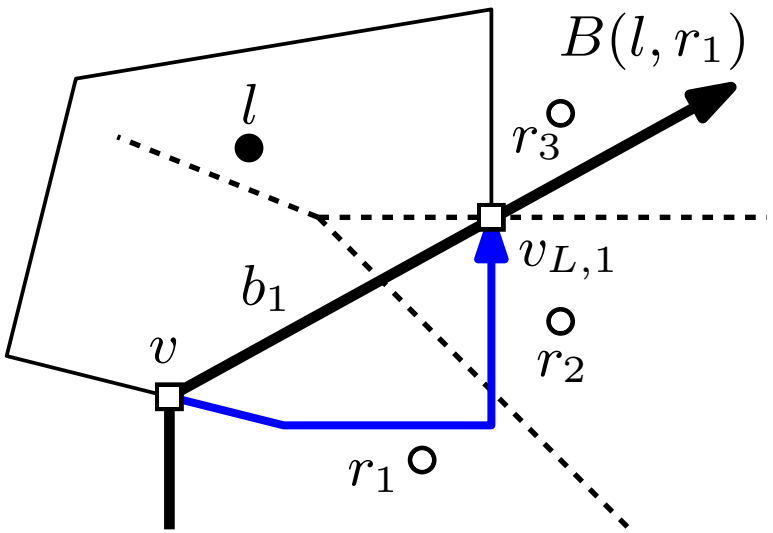
- $B(L, R)$ enters $\mathcal{V}(l, L)$ at v along $B(l, r_1)$



Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (1)

$B(L, R) \cap \mathcal{V}(l, L)$

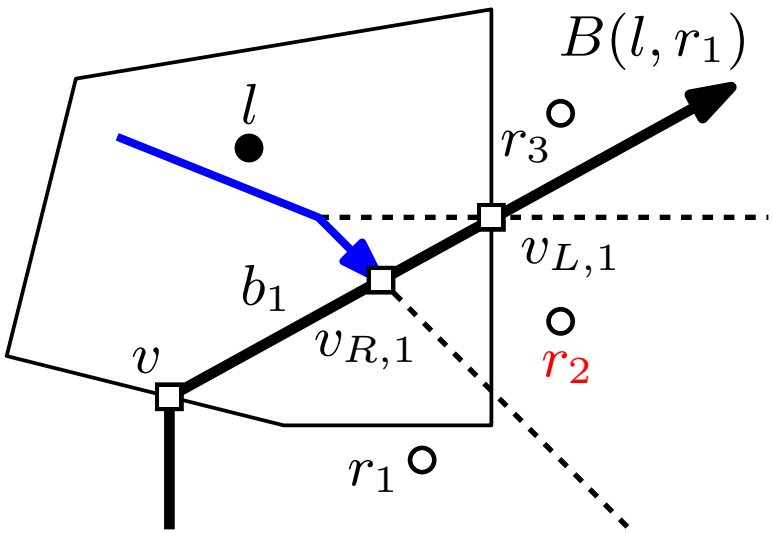
- $B(L, R)$ enters $\mathcal{V}(l, L)$ at v along $B(l, r_1)$
- Counterclockwise along $\partial\mathcal{V}(l, L)$ from v to find $v_{L,1} \in B(l, r_1)$
  - After $v_{L,1}$, $\mathcal{V}(l, L) \to \mathcal{V}(l', L)$, and $B(l, r_1) \to B(l', r_1)$



Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (1)

$B(L, R) \cap \mathcal{V}(l, L)$

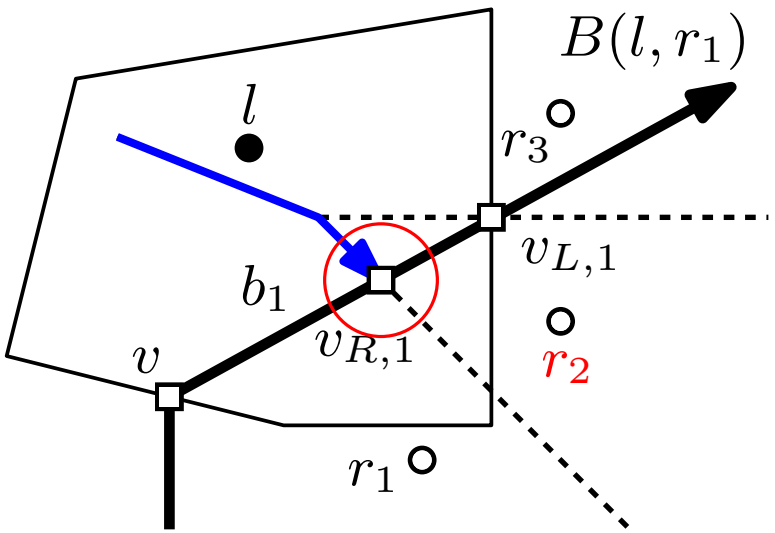
- $B(L, R)$ enters $\mathcal{V}(l, L)$ at v along $B(l, r_1)$
- Counterclockwise along $\partial\mathcal{V}(l, L)$ from v to find $v_{L,1} \in B(l, r_1)$
  - After $v_{L,1}$, $\mathcal{V}(l, L) \to \mathcal{V}(l', L)$, and $B(l, r_1) \to B(l', r_1)$
- Clockwise along $\partial\mathcal{V}(r_1, R)$ to find $v_{R,1} \in B(l, r_1)$
  - After $v_{R,1}$, $\mathcal{V}(r_1, R) \to \mathcal{V}(r_2, R)$, and $B(l, r_1) \to B(l, r_2)$



Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (1)

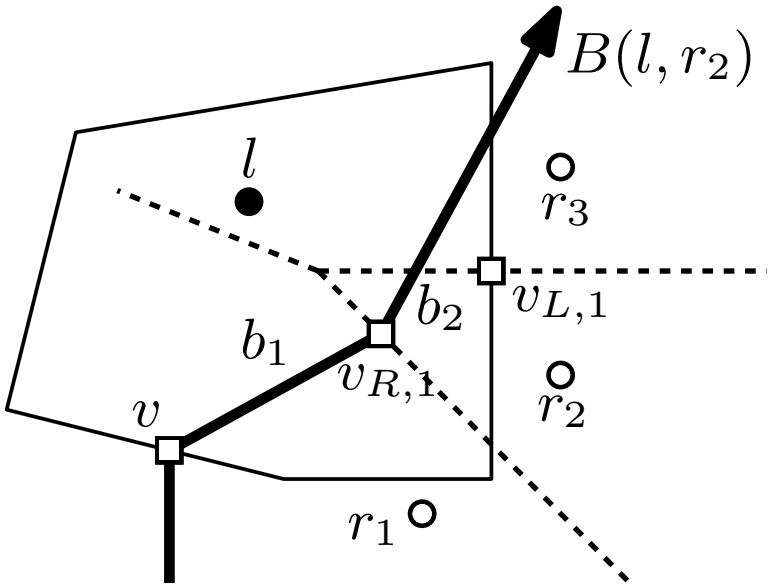
$B(L, R) \cap \mathcal{V}(l, L)$

- $B(L, R)$ enters $\mathcal{V}(l, L)$ at v along $B(l, r_1)$
- Counterclockwise along $\partial\mathcal{V}(l, L)$ from v to find $v_{L,1} \in B(l, r_1)$
  - After $v_{L,1}$, $\mathcal{V}(l, L) \to \mathcal{V}(l', L)$, and $B(l, r_1) \to B(l', r_1)$
- Clockwise along $\partial\mathcal{V}(r_1, R)$ to find $v_{R,1} \in B(l, r_1)$
  - After $v_{R,1}$, $\mathcal{V}(r_1, R) \to \mathcal{V}(r_2, R)$, and $B(l, r_1) \to B(l, r_2)$
- $v_{R,1}$ earlier than $v_{L,1}$, and $B(l, r_1) \to B(l, r_2)$ at $v_{R,1}$



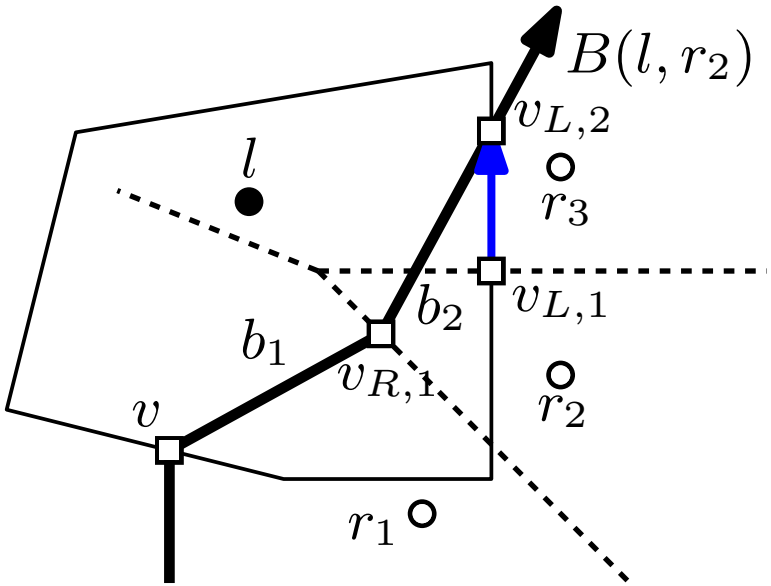
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (2)

- $B(L, R) \cap V(l, L)$
  - $B(L, R)$ from $v_{R,1}$ along $B(l, r_2)$



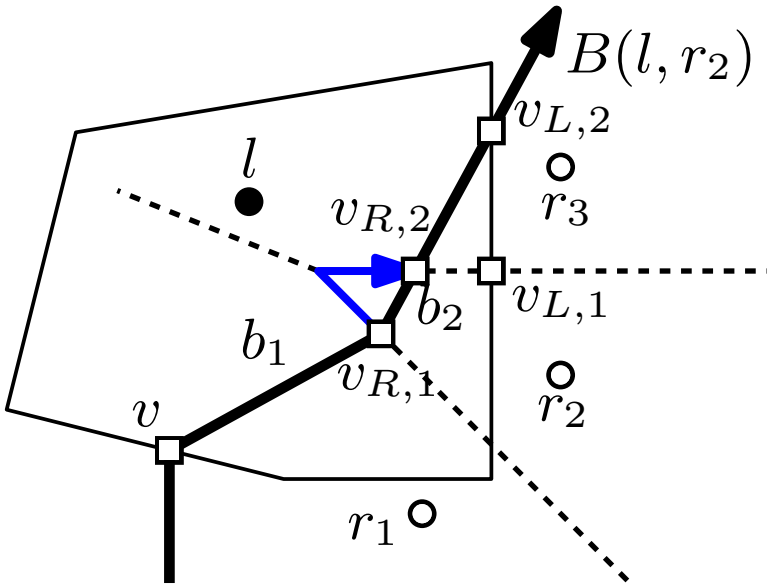
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (2)

- $B(L, R) \cap V(l, L)$
  - $B(L, R)$ from $v_{R,1}$ along $B(l, r_2)$
  - Counterclockwise along $\partial\mathcal{V}(l, L)$ from $v_{L,1}$ to find $v_{L,2} \in B(l, r_2)$
    - After $v_{L,2}$, $\mathcal{V}(l, L) \to \mathcal{V}(l', L)$, and $B(l, r_2) \to B(l', r_2)$



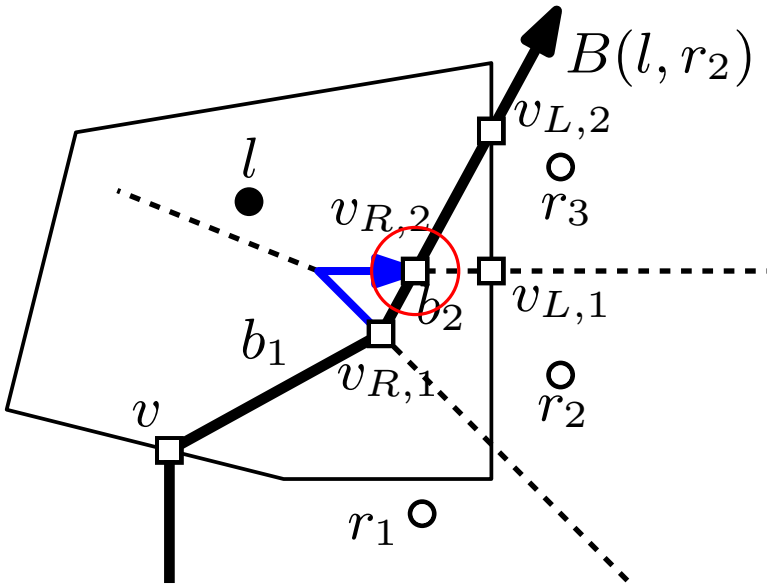
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (2)

- $B(L, R) \cap V(l, L)$
  - $B(L, R)$ from $v_{R,1}$ along $B(l, r_2)$
  - Counterclockwise along $\partial\mathcal{V}(l, L)$ from $v_{L,1}$ to find $v_{L,2} \in B(l, r_2)$
    - After $v_{L,2}$, $\mathcal{V}(l, L) \to \mathcal{V}(l', L)$, and $B(l, r_2) \to B(l', r_2)$
  - Clockwise along $\partial\mathcal{V}(r_2, R)$ from $v_{R,1}$ to find $v_{R,2} \in B(l, r_2)$
    - After $v_{R,2}$, $\mathcal{V}(r_2, R) \to \mathcal{V}(r_3, R)$, and $B(l, r_2) \to B(l, r_3)$



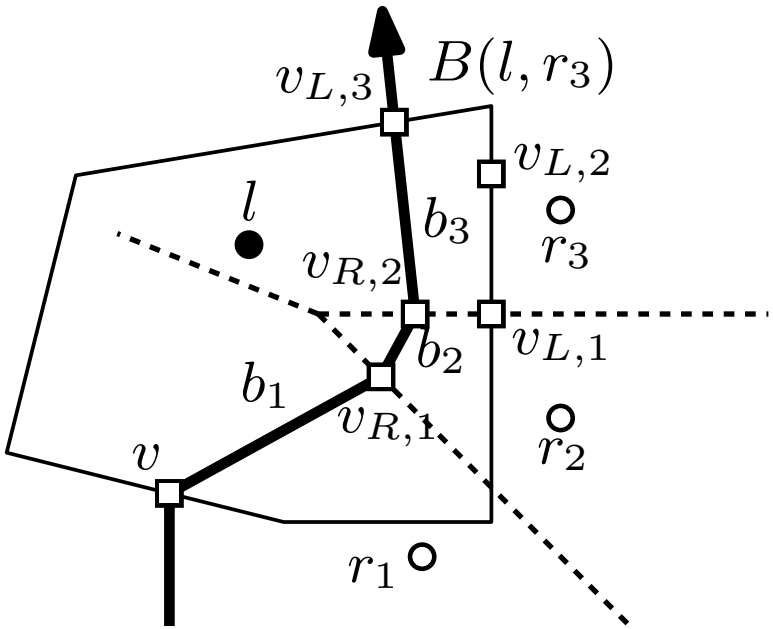
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (2)

- $B(L, R) \cap V(l, L)$
  - $B(L, R)$ from $v_{R,1}$ along $B(l, r_2)$
  - Counterclockwise along $\partial\mathcal{V}(l, L)$ from $v_{L,1}$ to find $v_{L,2} \in B(l, r_2)$
    - After $v_{L,2}$, $\mathcal{V}(l, L) \to \mathcal{V}(l', L)$, and $B(l, r_2) \to B(l', r_2)$
  - Clockwise along $\partial\mathcal{V}(r_2, R)$ from $v_{R,1}$ to find $v_{R,2} \in B(l, r_2)$
    - After $v_{R,2}$, $\mathcal{V}(r_2, R) \to \mathcal{V}(r_3, R)$, and $B(l, r_2) \to B(l, r_3)$
  - $v_{R,2}$ earlier than $v_{L,2}$, and $B(l, r_2) \to B(l, r_3)$ at $v_{R,2}$



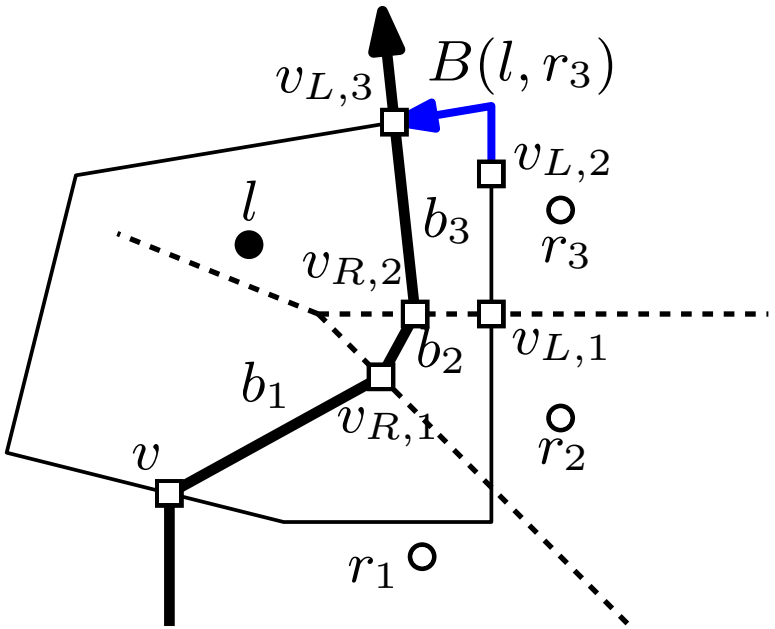
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (3)

- $B(L, R) \cap \mathcal{V}(l, L)$
  - $B(L, R)$ from $v_{R,2}$ along $B(l, r_3)$



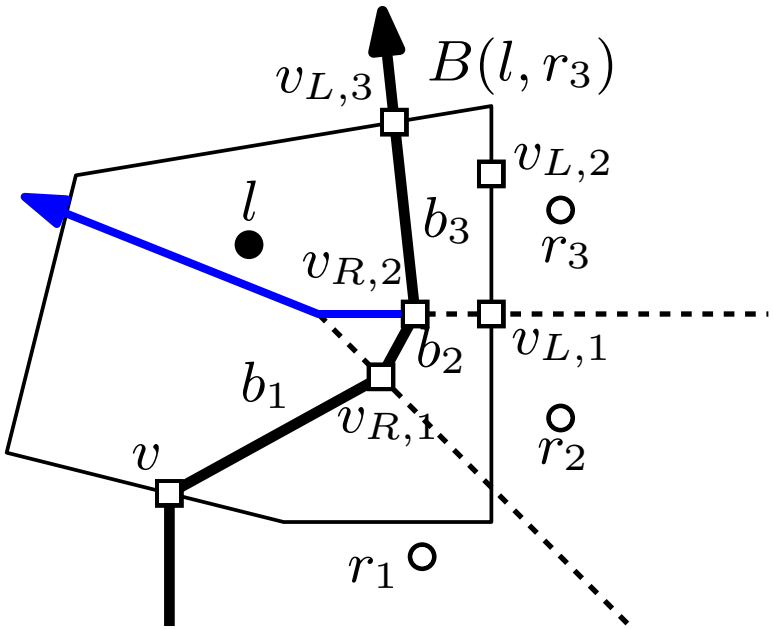
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (3)

- $B(L, R) \cap \mathcal{V}(l, L)$
  - $B(L, R)$ from $v_{R,2}$ along $B(l, r_3)$
  - Counterclockwise along $\partial\mathcal{V}(l, L)$ from $v_{L,2}$ to find $v_{L,3} \in B(l, r_3)$
    - After $v_{L,3}$, $\mathcal{V}(l, L) \to \mathcal{V}(l'', L)$, and $B(l, r_3) \to B(l'', r_3)$



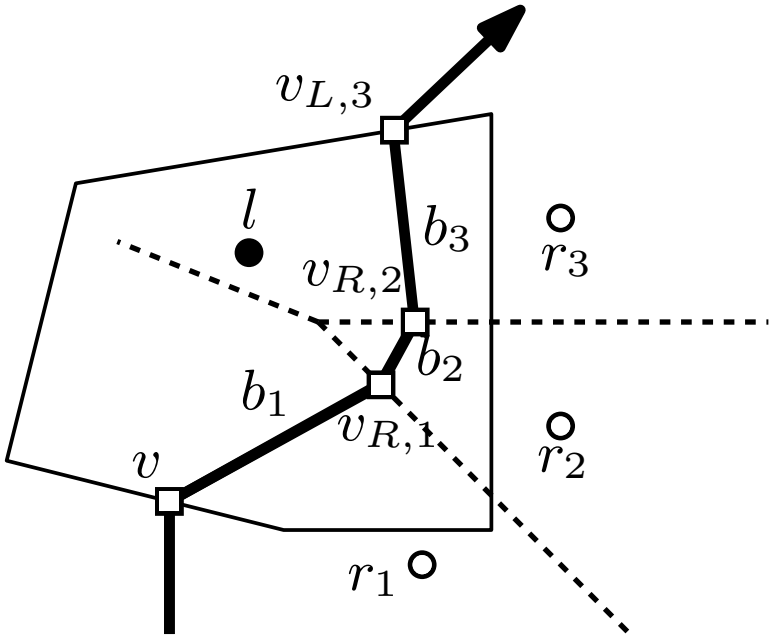
Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (3)

- $B(L, R) \cap \mathcal{V}(l, L)$
  - $B(L, R)$ from $v_{R,2}$ along $B(l, r_3)$
  - Counterclockwise along $\partial\mathcal{V}(l, L)$ from $v_{L,2}$ to find $v_{L,3} \in B(l, r_3)$
    - After $v_{L,3}$, $\mathcal{V}(l, L) \to \mathcal{V}(l'', L)$, and $B(l, r_3) \to B(l'', r_3)$
  - Clockwise along $\partial\mathcal{V}(r_3, R)$ from $v_{R,2}$ to find $v_{R,3} \in B(l, r_3)$
    - No $v_{R,3}$



Trace $B(L, R)$ in $\mathcal{V}(l, L)$ (3)

- $B(L, R) \cap \mathcal{V}(l, L)$
  - $B(L, R)$ from $v_{R,2}$ along $B(l, r_3)$
  - Counterclockwise along $\partial\mathcal{V}(l, L)$ from $v_{L,2}$ to find $v_{L,3} \in B(l, r_3)$
    - After $v_{L,3}$, $\mathcal{V}(l, L) \to \mathcal{V}(l'', L)$, and $B(l, r_3) \to B(l'', r_3)$
  - Clockwise along $\partial\mathcal{V}(r_3, R)$ from $v_{R,2}$ to find $v_{R,3} \in B(l, r_3)$
    - No $v_{R,3}$
  - $B(l, r_3) \to B(l'', r_3)$ at $v_{L,3}$



Time to Compute $B(L, R)$

Karlsruhe Institute of Technology

Lemma 9:

Computing $B(L, R)$ takes $O(|L| + |R|) = O(|P|)$ time

proof:

- Finding the starting edge takes $O(|L| + |R|)$ time
- Traversing $\text{Vor}(L)$ takes $O(|\text{Vor}(L)|)$ time ($\text{Vor}(R) \rightarrow O(|\text{Vor}(R)|)$)
  - Enter each $\mathcal{V}(l, L)$ and its boundary at most once
- There are $O(|B(L, R)|)$ intersections
  - Each edge of $B(L, R)$ makes two intersections.
- $O(|L| + |R|) + O(|\text{Vor}(L)|) + O(|\text{Vor}(R)|) + O(|B(L, R)|)$
  $= O(|L| + |R| + |L| + |R| + |P|) = O(|P|)$
  - $O(|\text{Vor}(L)|) = O(|L|)$, $O(|\text{Vor}(R)|) = O(|R|)$, and $O(|B(L, R)|) = O(|P|)$

Construction Time for Divide and Conquer

Karlsruhe Institute of Technology

Theorem 10:

The divide-and-conquer algorithm computes $\mathrm{Vor}(P)$ in $O(n \log n)$ time

proof:

- Sorting takes $O(n \log n)$ time (only do once)
- A sub-instance $P' \subset P$ takes $O(|P'|)$ time
  - Partitioning P' into L' and R' takes $O(|P'|)$ time
  - Computing $B(L', R')$ takes $O(|P'|)$ time
- The i^{th}-level takes $O(2^i) \times O(n/2^i) = O(n)$ time
  - $O(2^i)$ sub-instances each with $O(n/2^i)$ sites
- $O(\log n)$ levels
- $T(n) = O(n) + 2 \cdot T(n/2) => T(n) = O(n \log n)$

Comparison between two Algorithms

Plane Sweep

- $O(n)$ Voronoi vertices
  - A vertex must belong to $\mathrm{Vor}(P)$

Divide and Conquer

- $\Theta(n \log n)$ Voronoi vertices
  - Each level computes $\Theta(n)$ vertices.
  - A vertex does not necessarily belong to $\mathrm{Vor}(P)$

Comparison between two Algorithms

Plane Sweep

- $O(n)$ Voronoi vertices
  - A vertex must belong to $\mathrm{Vor}(P)$
- Better Space
  - Only keep the wavefront W

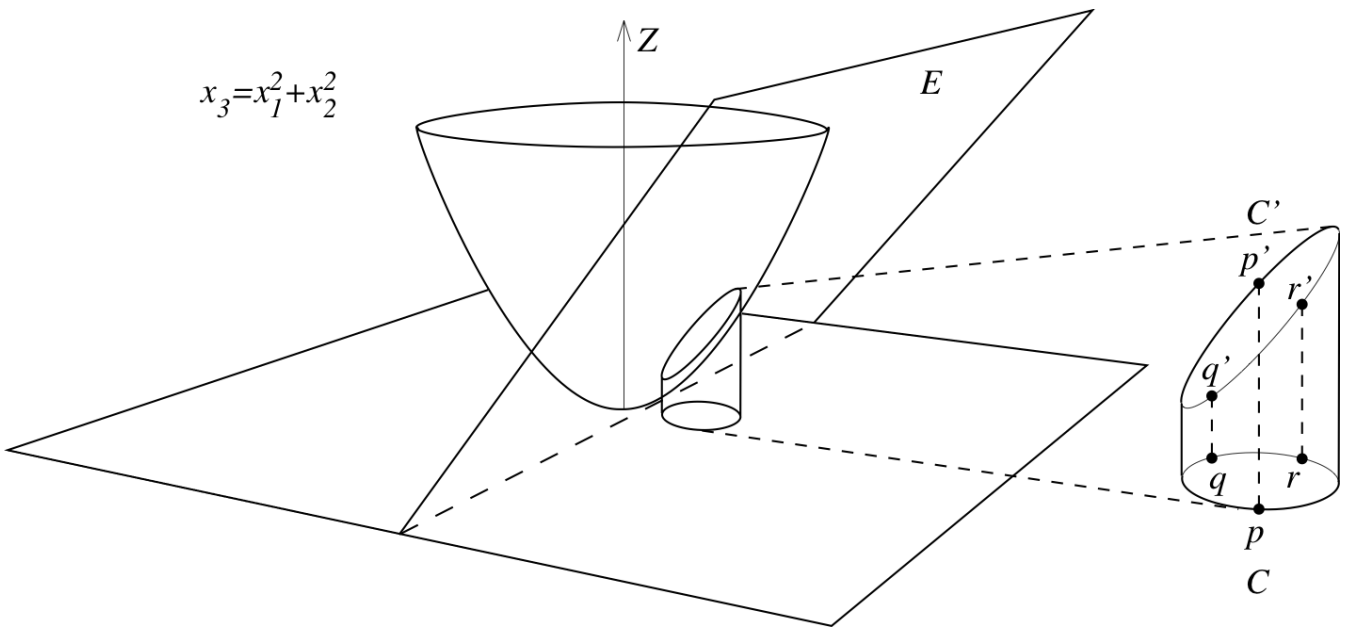
Divide and Conquer

- $\Theta(n \log n)$ Voronoi vertices
  - Each level computes $\Theta(n)$ vertices.
  - A vertex does not necessarily belong to $\mathrm{Vor}(P)$
- Parallelism (n machines and P is sorted)

$$\sum_{i=0}^{O(\log n)} O(n/2^i) = O(n)$$

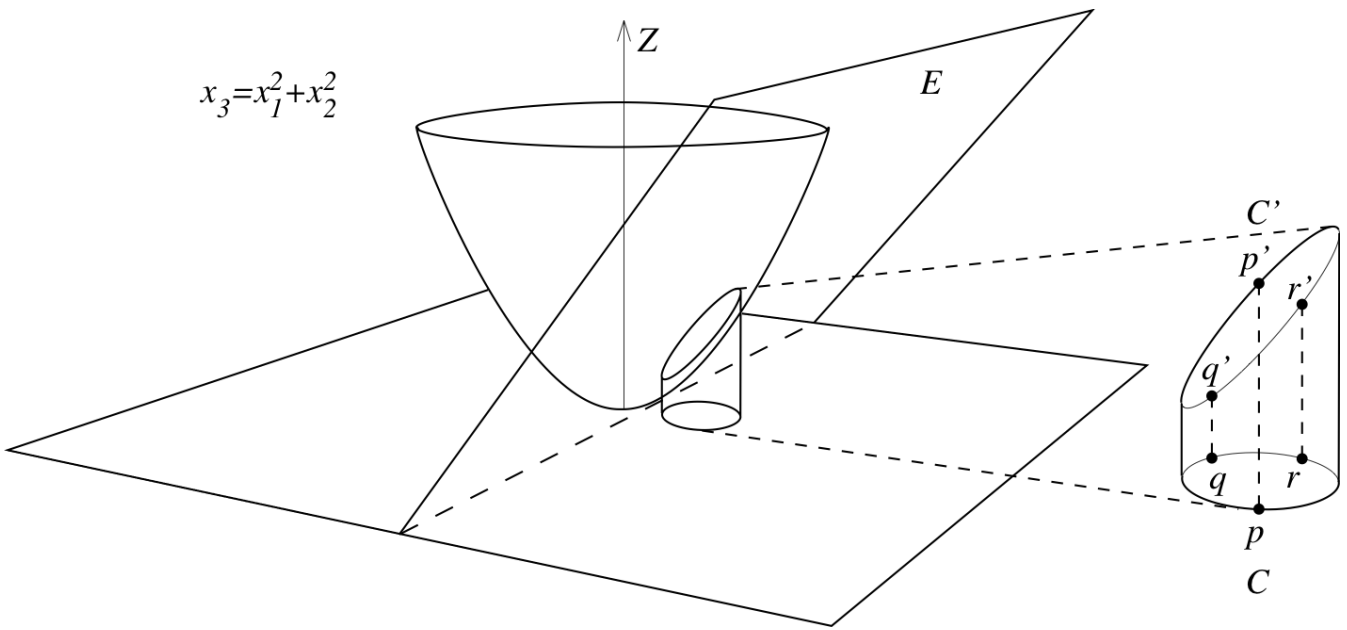
Geometric Transformation from 2D to 3D

- A paraboloid $\mathcal{P} = \{(x_1, x_2, x_3) \mid x_1^2 + x_2^2 = x_3\}$ in 3D



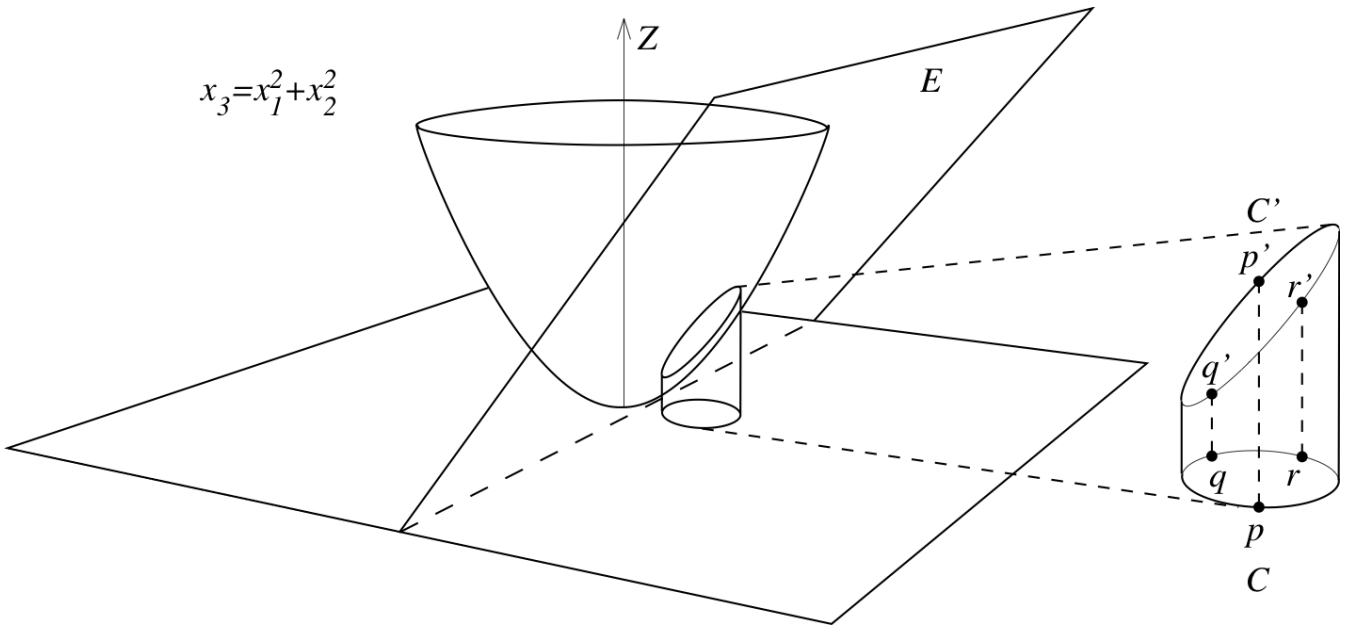
Geometric Transformation from 2D to 3D

- A paraboloid $\mathcal{P} = \{(x_1, x_2, x_3) \mid x_1^2 + x_2^2 = x_3\}$ in 3D
- For a point $x = (x_1, x_2)$ in 2D, $x' = (x_1, x_2, x_1^2 + x_2^2)$ is its lifted image in 3D
  - $x' \leftarrow$ vertical projection from x to $\mathcal{P}$



Geometric Transformation from 2D to 3D

- A paraboloid $\mathcal{P} = \{(x_1, x_2, x_3) \mid x_1^2 + x_2^2 = x_3\}$ in 3D
- For a point $x = (x_1, x_2)$ in 2D, $x' = (x_1, x_2, x_1^2 + x_2^2)$ is its lifted image in 3D
  - $x' \leftarrow$ vertical projection from x to $\mathcal{P}$
- For a set A of points in 2D, its lifted image $A' = \{x' = (x_1, x_2, x_1^2 + x_2^2) \mid x = (x_1, x_2) \in A\}$



Circle in 2D $\leftrightarrow$ Planar Curve in $\mathcal{P}$

Karlsruhe Institute of Technology

Lemma 11:

Let C be a circle in the plane. Then C' is a planar curve on the paraboloid $\mathcal{P}$

proof:

- C is given by $r^2 = (x_1 - c_1)^2 + (x_2 - c_2)^2$
  - $r^2 = x_1^2 + x_2^2 - 2x_1c_1 - 2x_2c_2 + c_1^2 + c_2^2$
- C' satisfies $x_1^2 + x_2^2 = x_3$
- Substituting $x_1^2 + x_2^2$ by x_3, we obtain a plane E

$$x_3 - 2x_1c_1 - 2x_2c_2 + c_1^2 + c_2^2 - r^2 = 0$$

- $C' = \mathcal{P} \cap E$
- Intersection between E and $\mathcal{P}$ is a planar curve

Lower Convex Hull

- $P' =$ lifted image of P on $\mathcal{P} \rightarrow P'$ in convex position
- Each point of P' is a vertex of $conv(P')$
- Lower convex hull $lconv(P')$ of P' is the part of $conv(P')$ visible from $x_3 = -\infty$

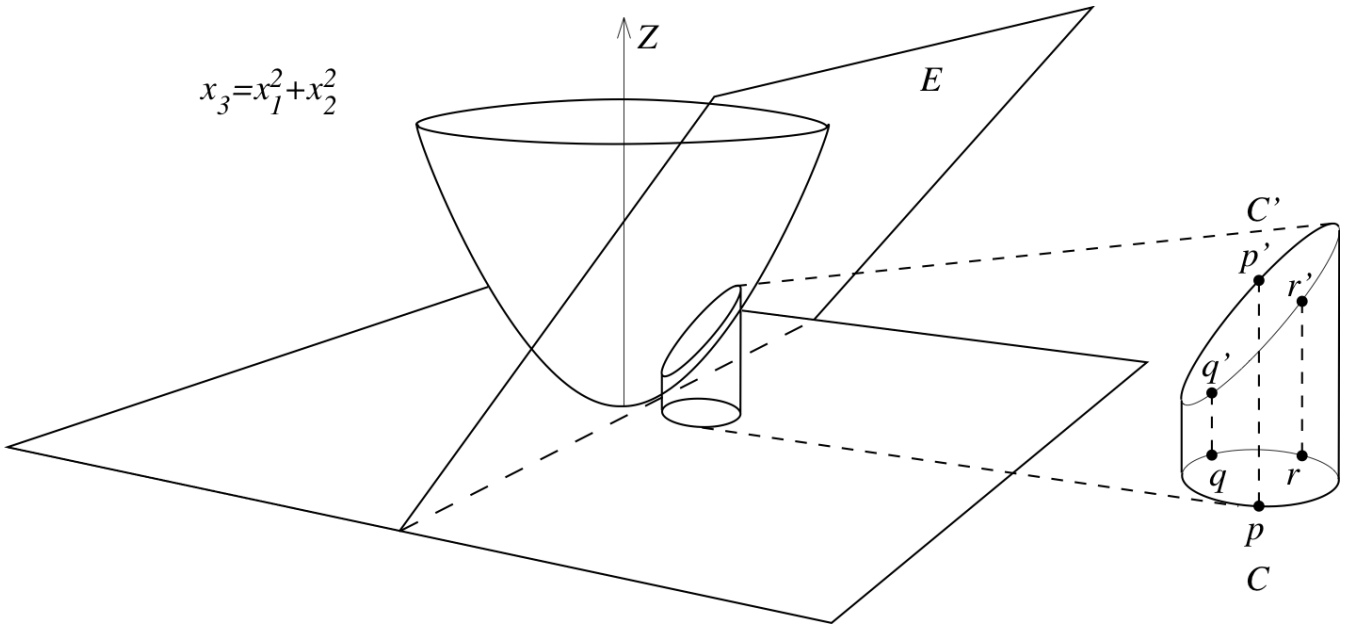
Duality between $\mathrm{DT}(P)$ and $lconv(P')$ (1)

Theorem 12:

Delaunay triangulation $\mathrm{DT}(P)$ equals to the vertical projection onto the x_1x_2-plane of lower convex hull of P'

proof:

- $p, q, r \in P$. C: circumcircle of p, q, r
- C' lies on a plane E defined by p', q', r'
- a point x inside $C \leftrightarrow$ lifted image x' below E



Duality between $\mathrm{DT}(P)$ and $lconv(P')$ (2)

Theorem 12:

Delaunay triangulation $\mathrm{DT}(P)$ equals to the vertical projection onto the x_1x_2-plane of lower convex hull of P'

proof:

- p, q, r defines a triangle of $\mathrm{DT}(P)$
  $\leftrightarrow$ no point of P lies in C defined by p, q, r
  $\leftrightarrow$ no point of P' lies below E defined by p', q', r'
  $\leftrightarrow$ p', q', r' defines a facet of $lconv(P')$
- Computing a convex hull in 3D takes $O(n \log n)$ time
  - Vor(P) in $O(n \log n)$ time