Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

Institute for Informatics

Computational Geometry

Winter term 2016/17

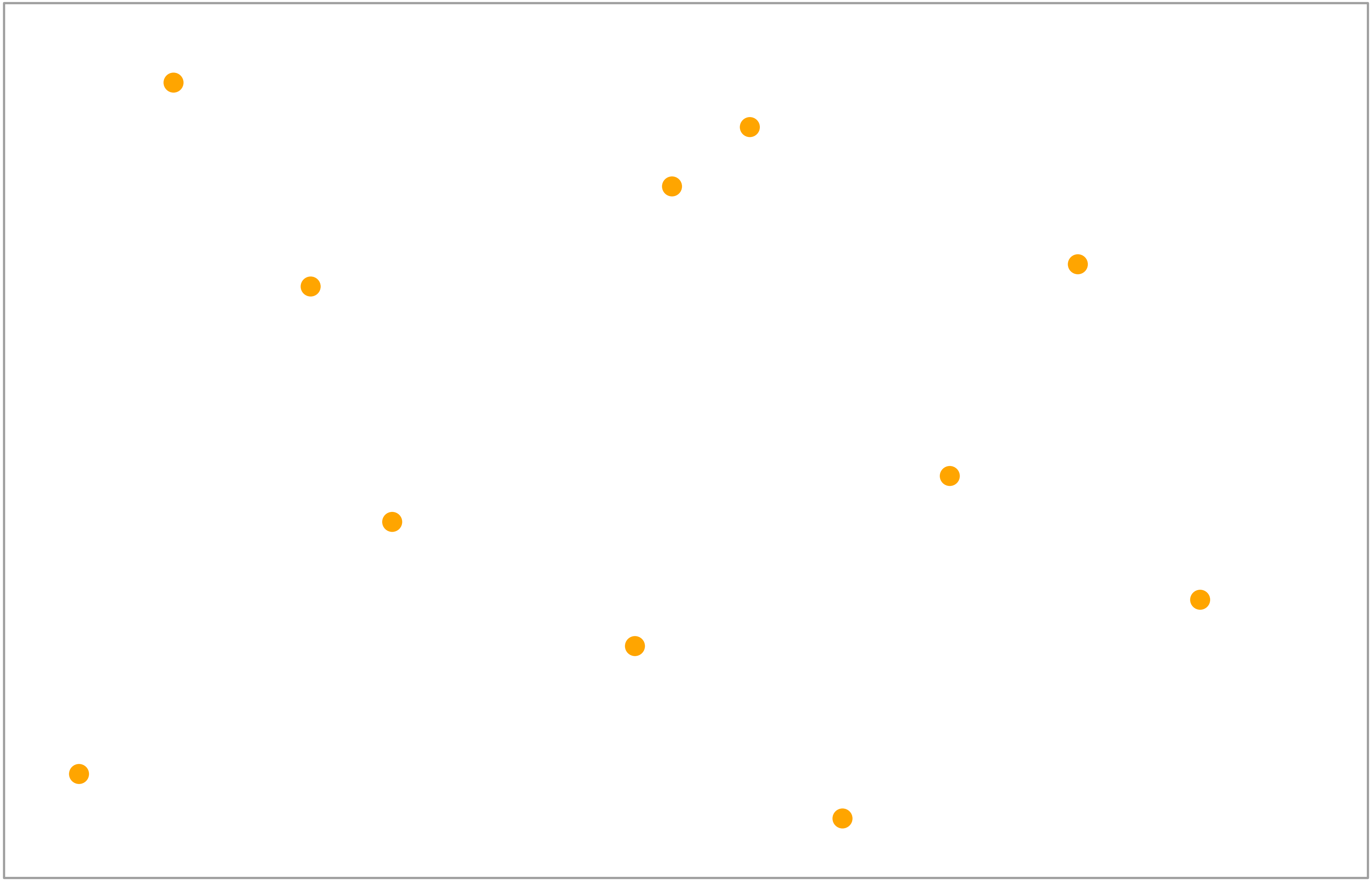
The Post-Office Problem

Lecture #7

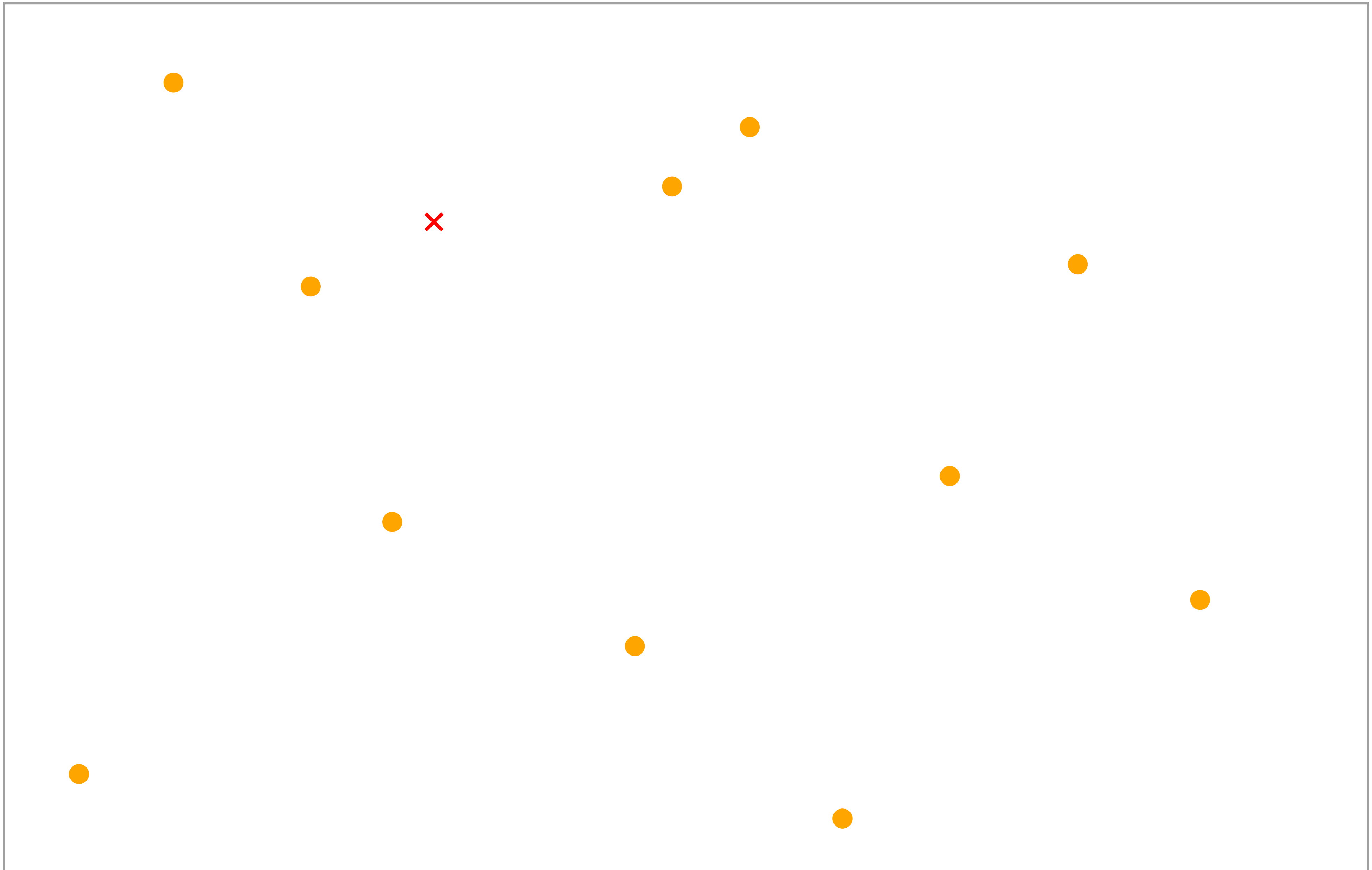
Joachim Spoerhase

Chair for Computer Science I

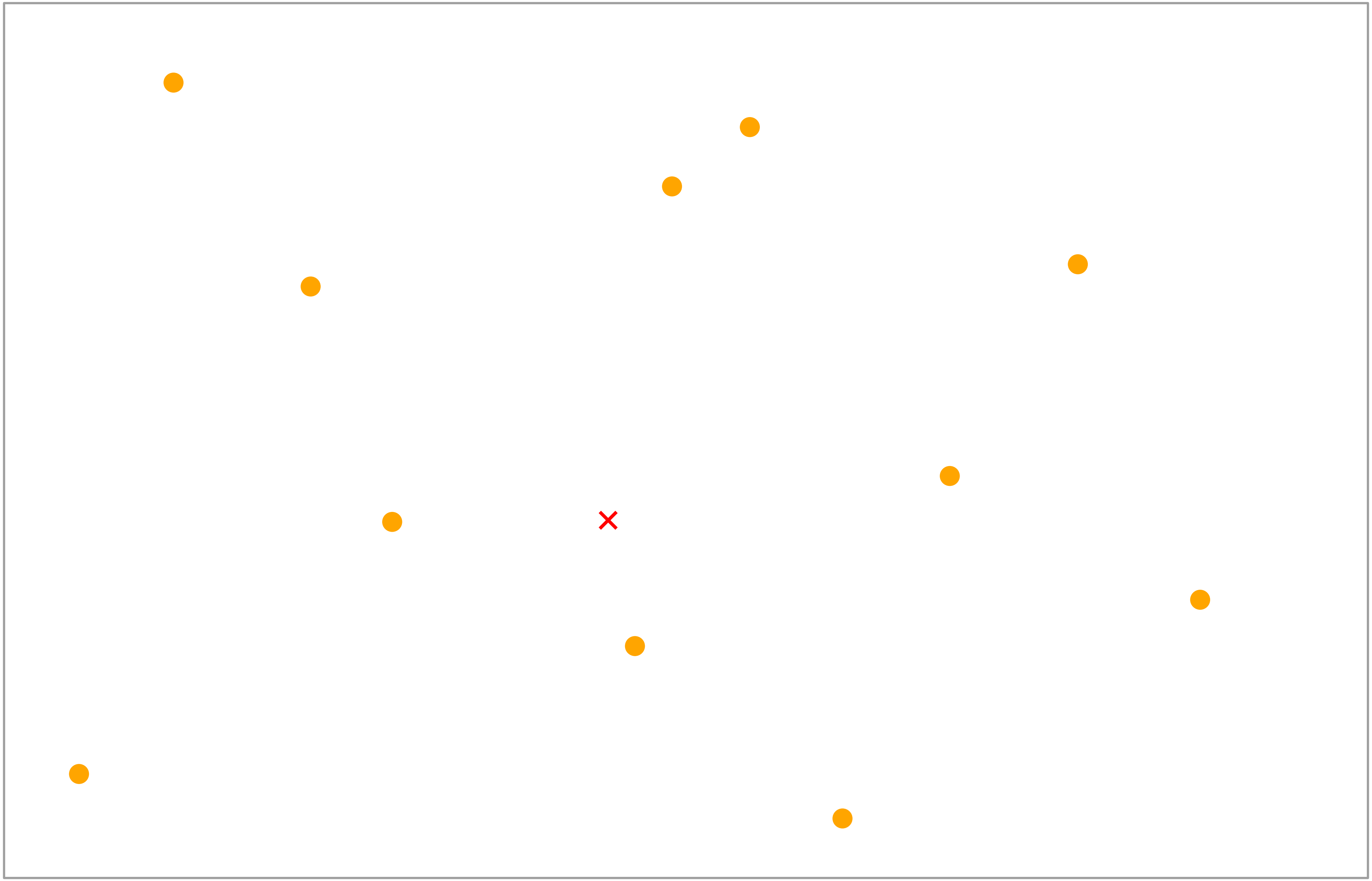
The Post-Office Problem



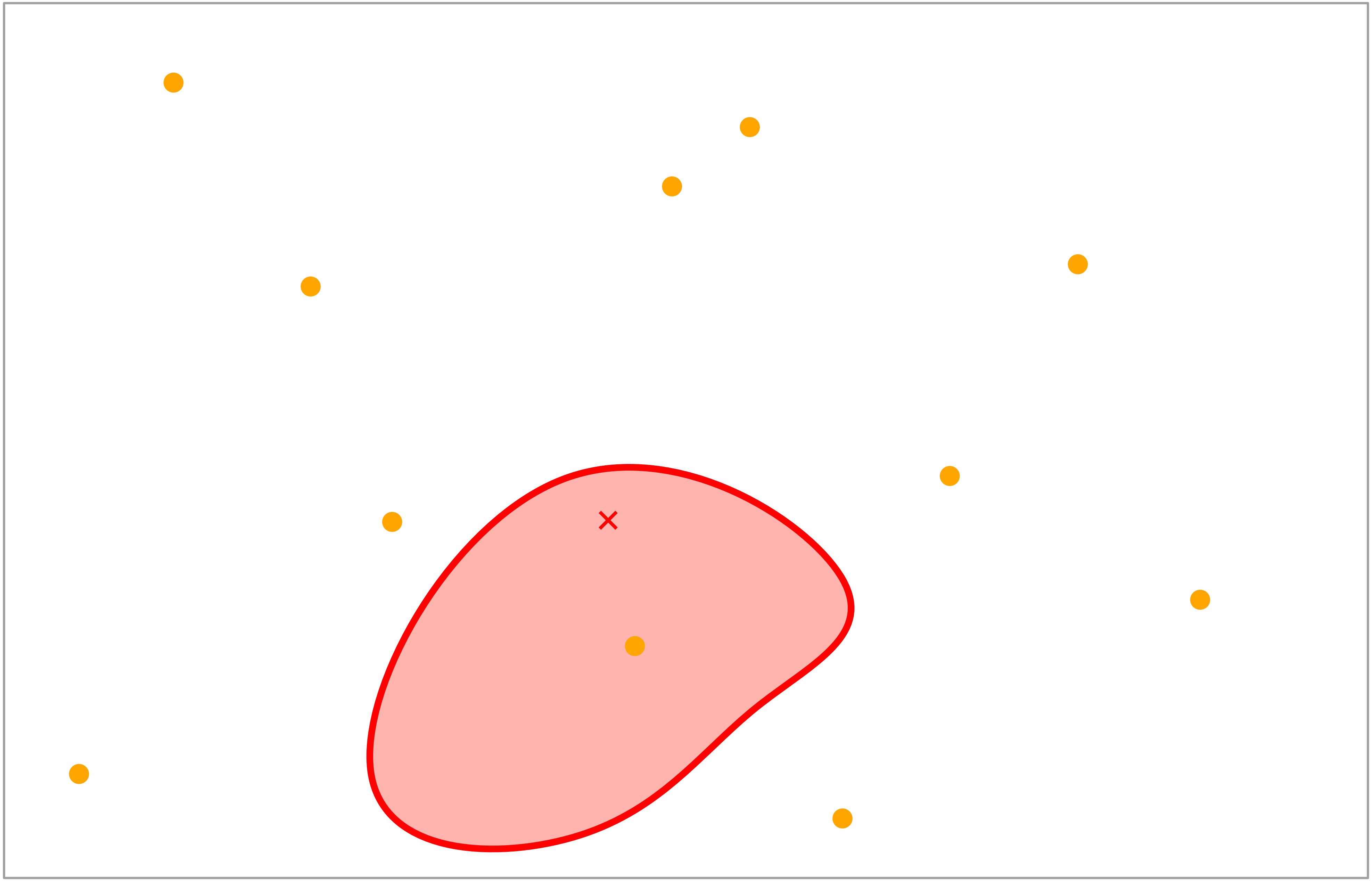
The Post-Office Problem



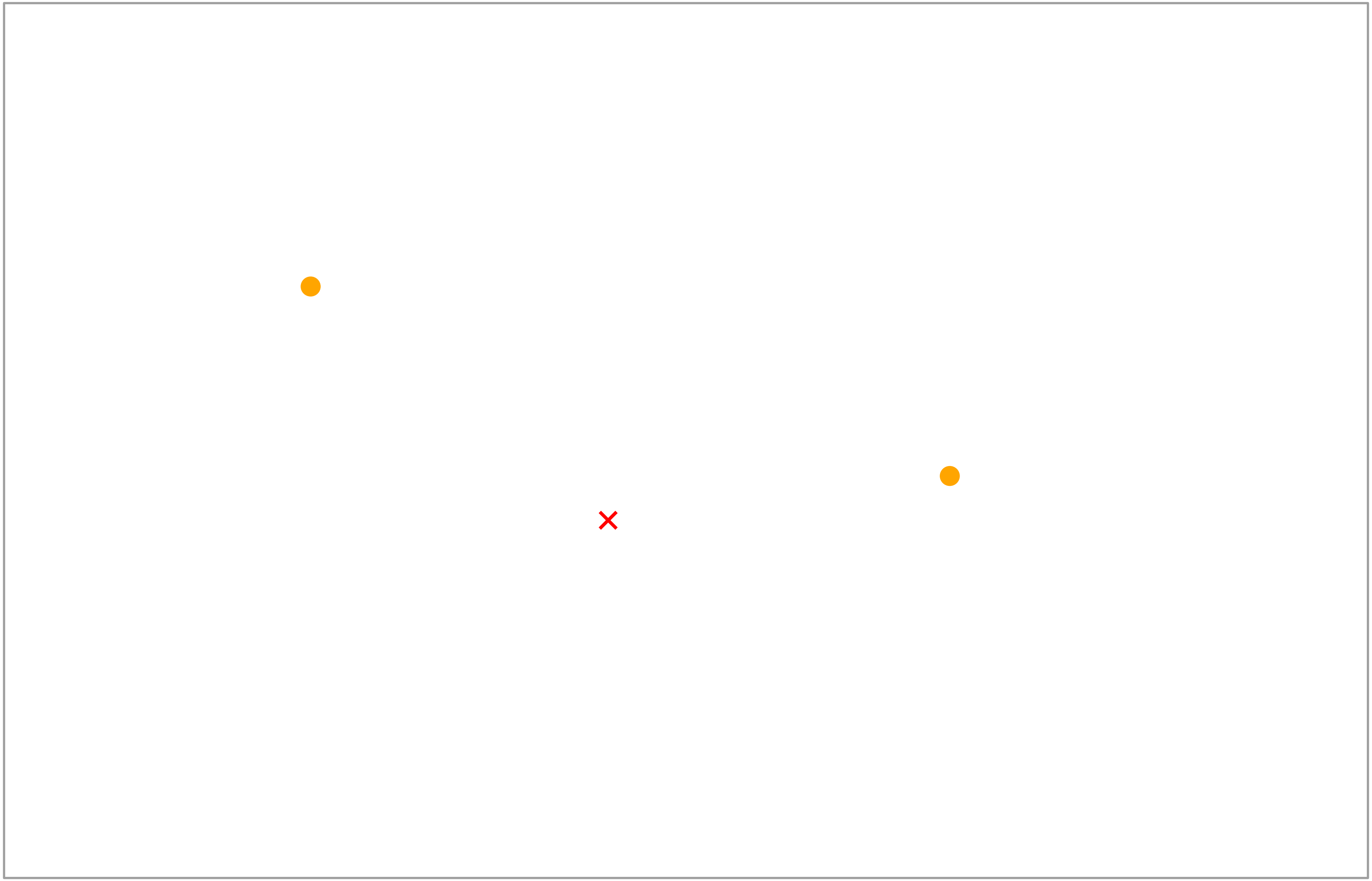
The Post-Office Problem



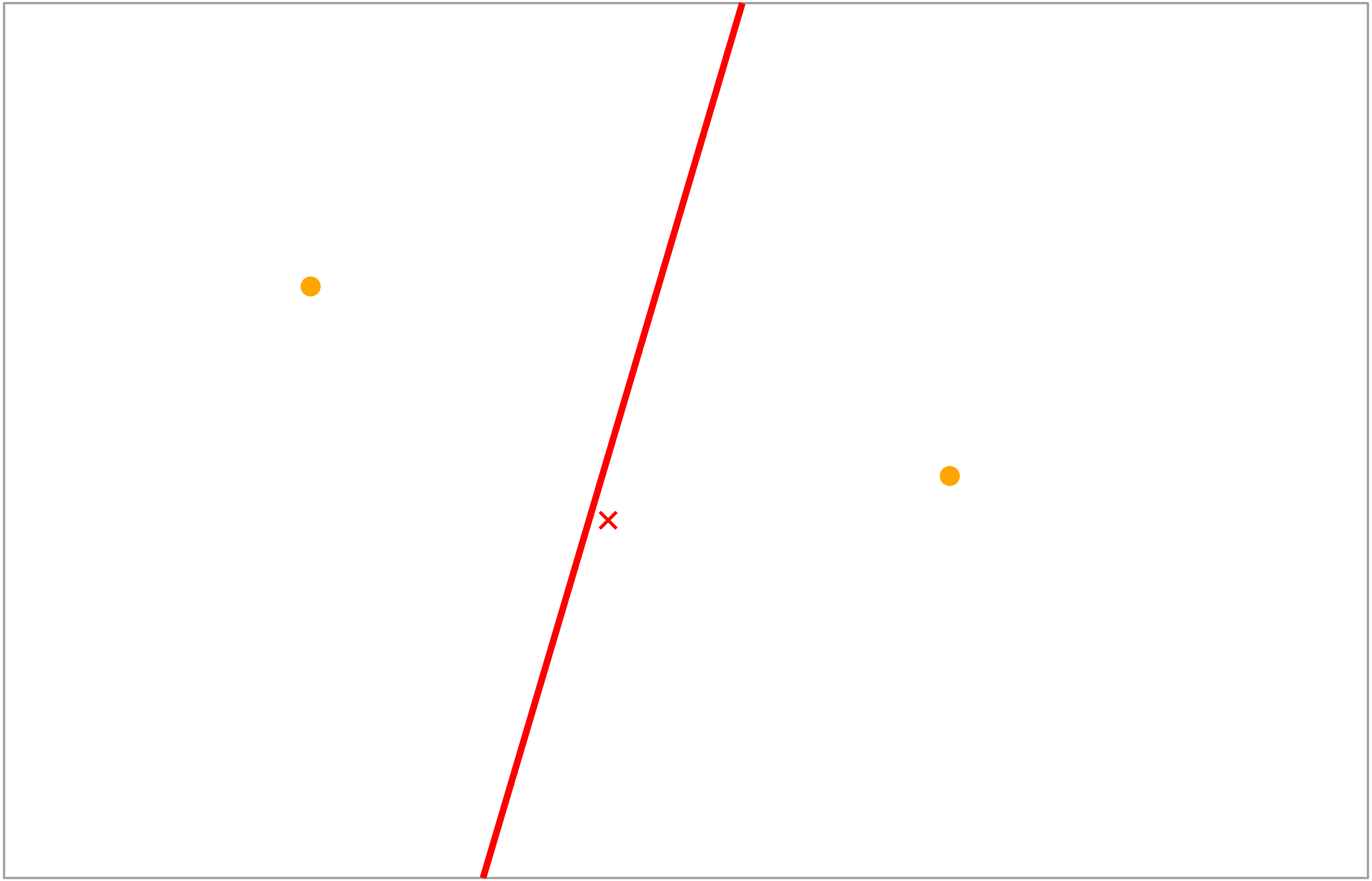
The Post-Office Problem



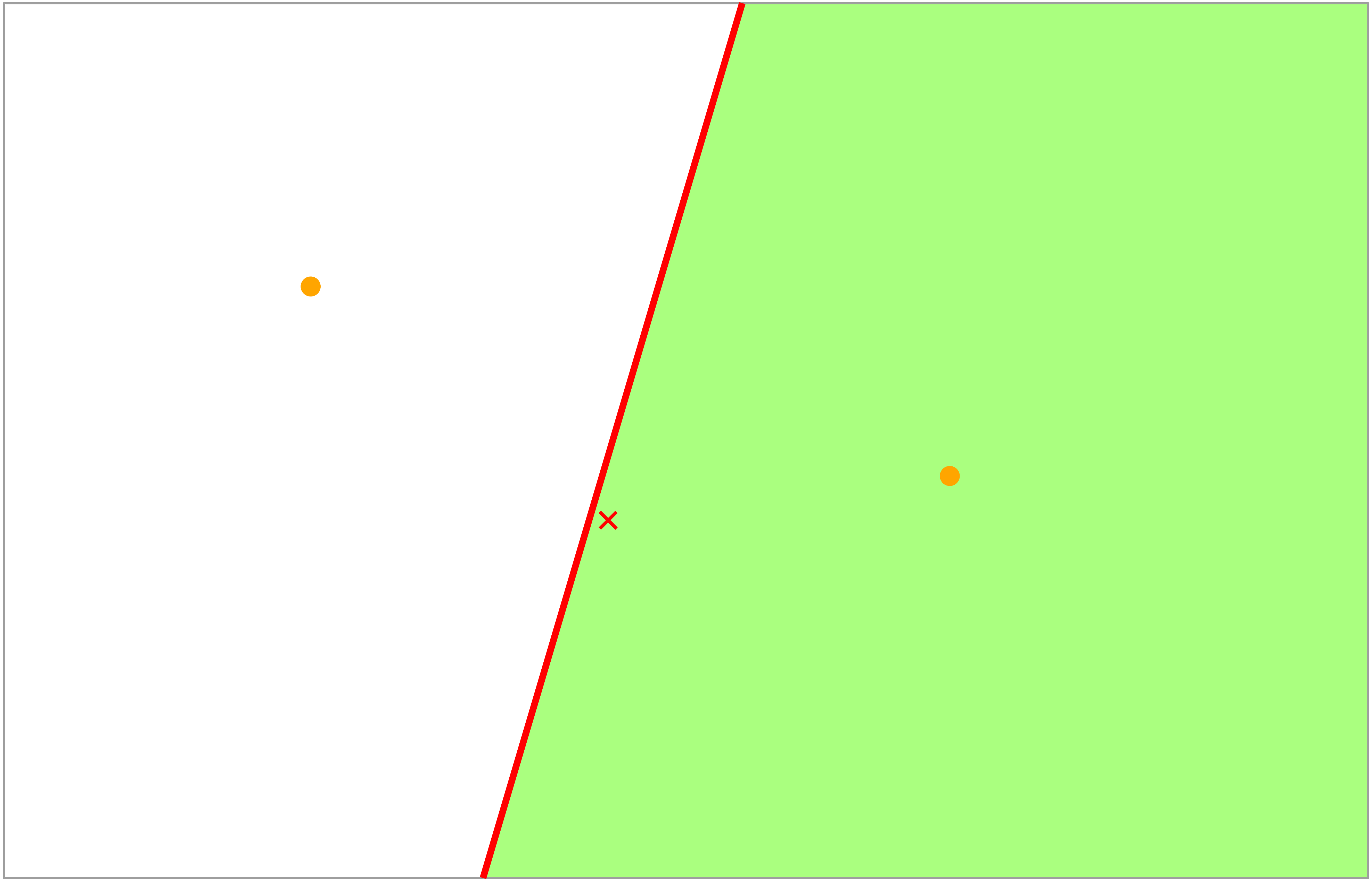
The Post-Office Problem



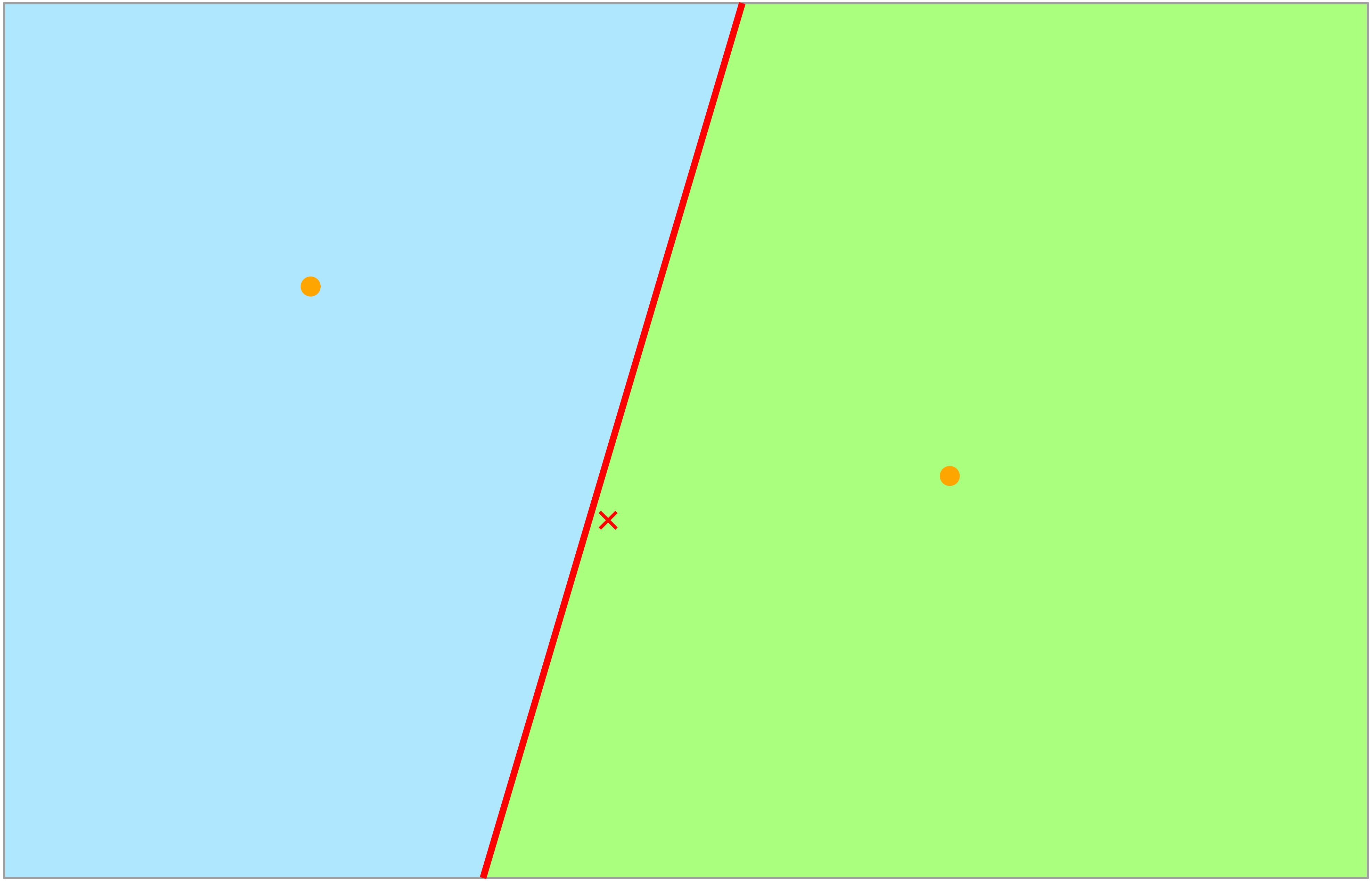
The Post-Office Problem



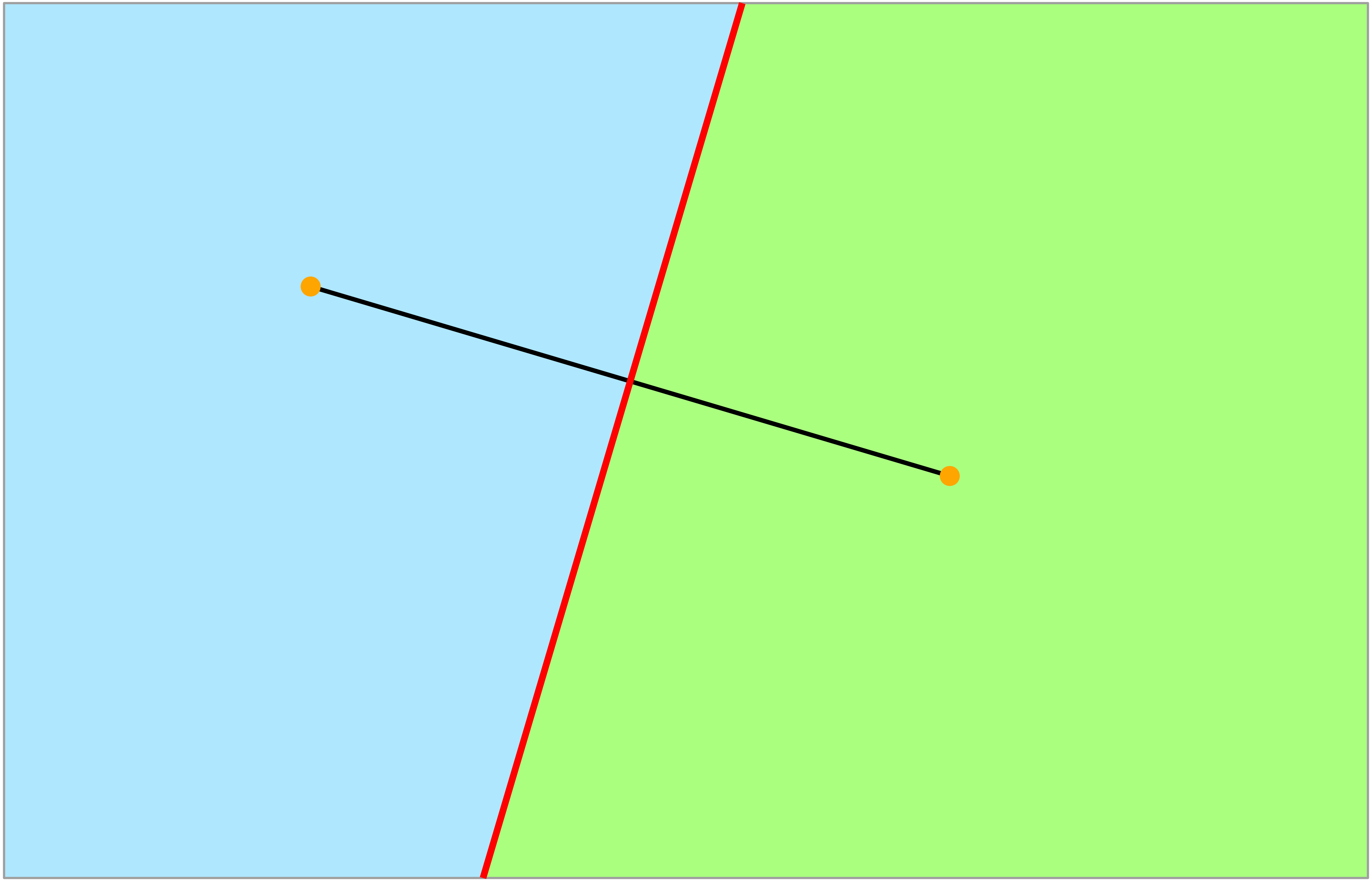
The Post-Office Problem



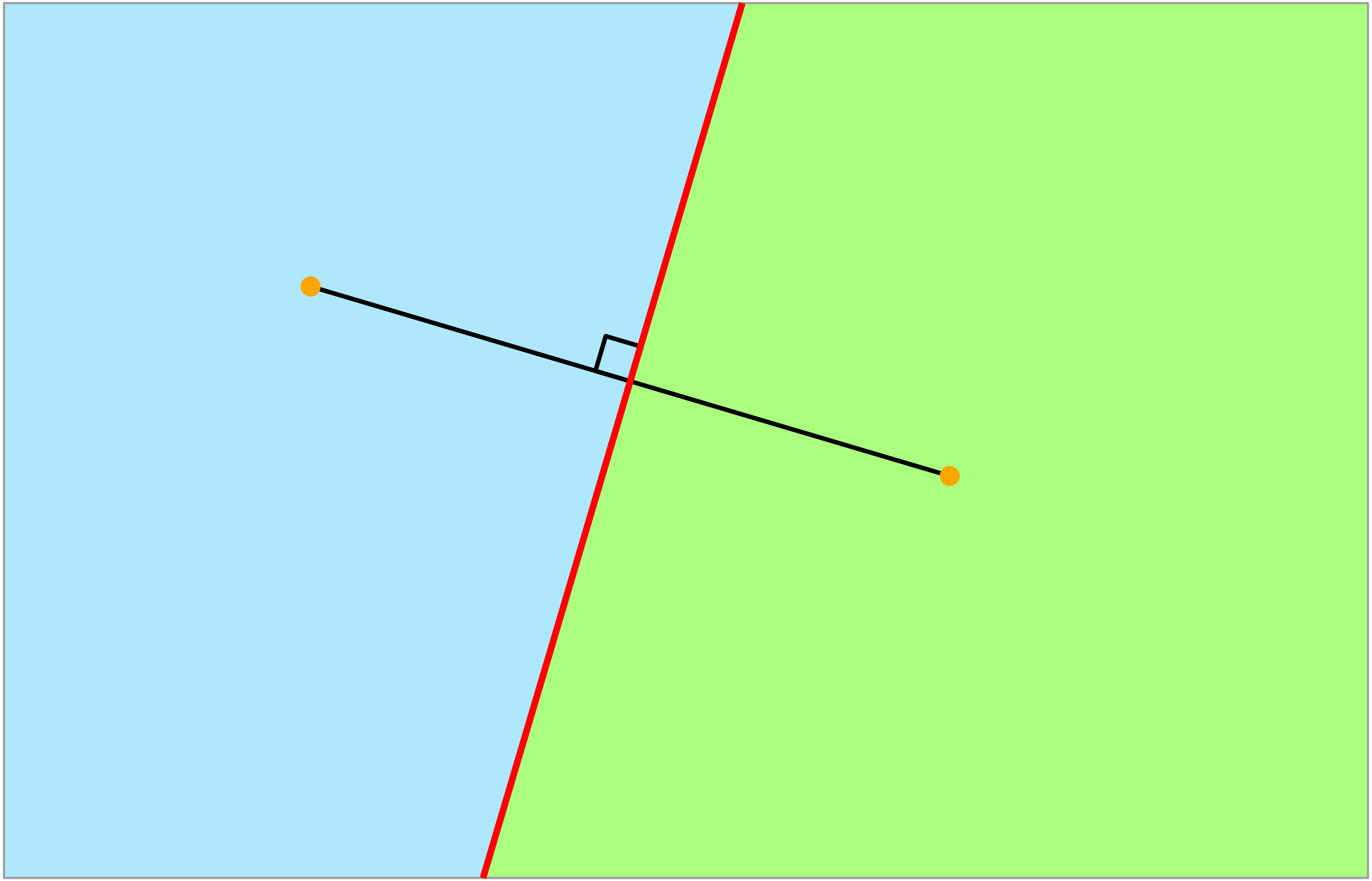
The Post-Office Problem



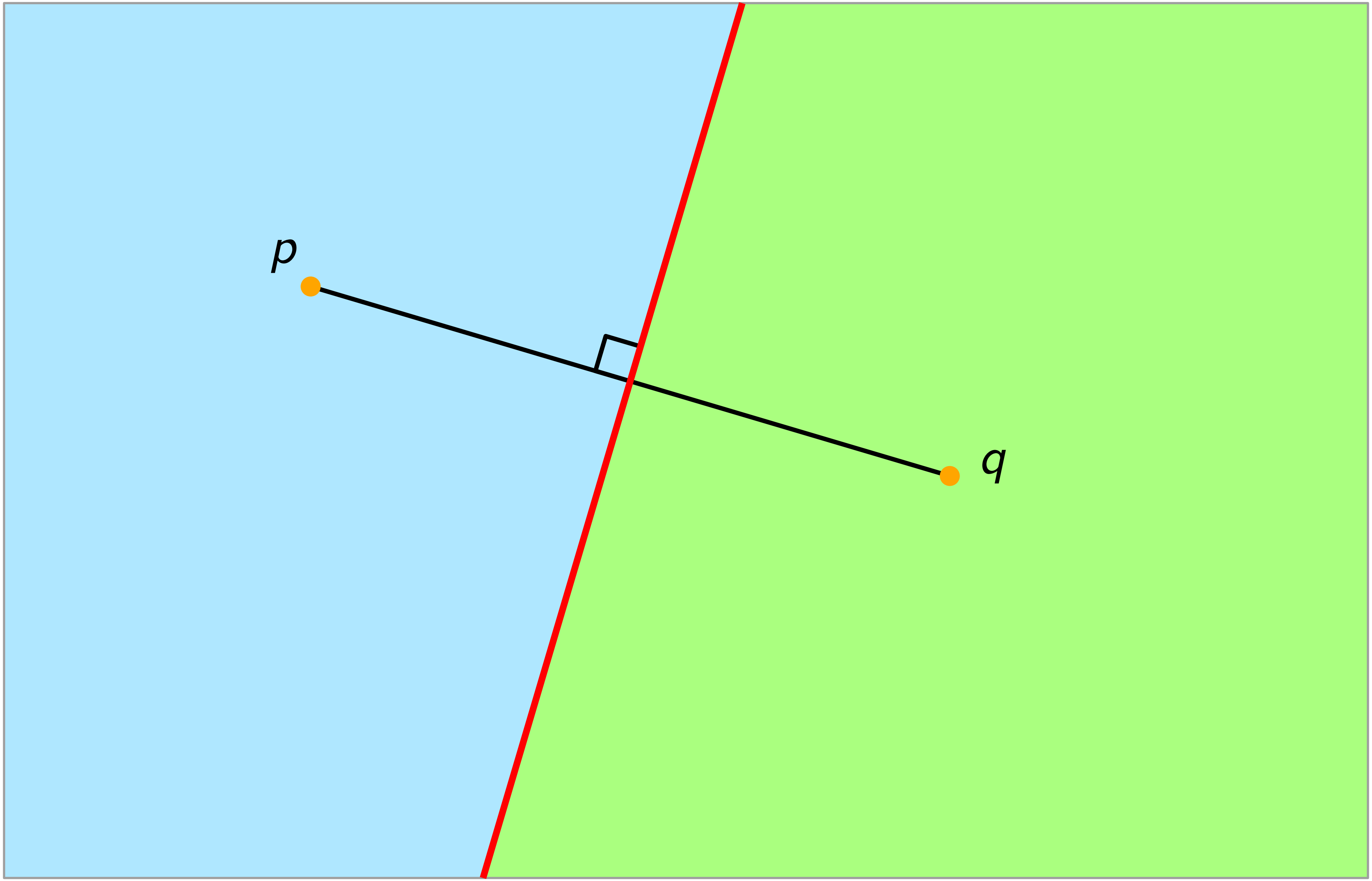
The Post-Office Problem



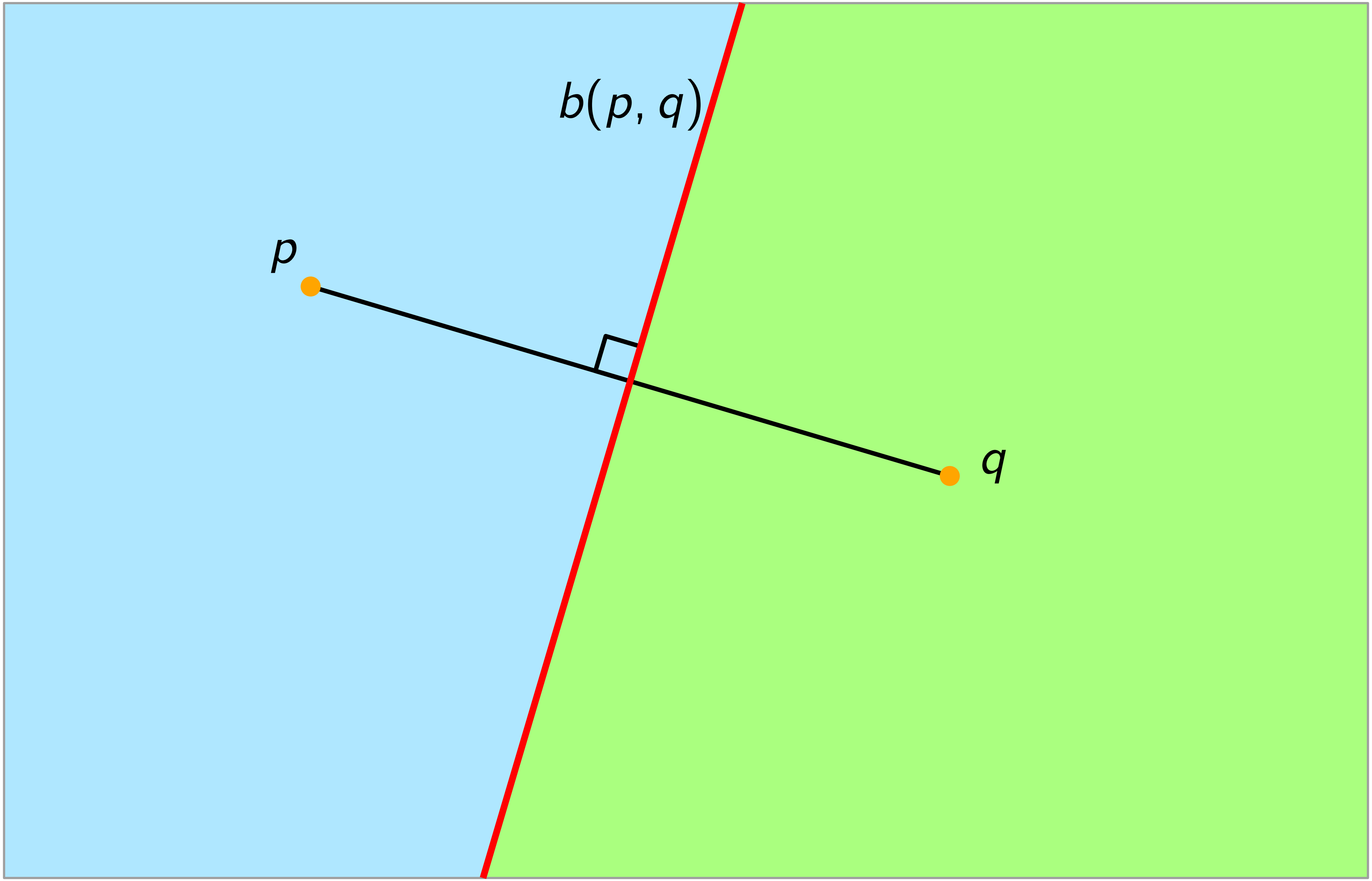
The Post-Office Problem



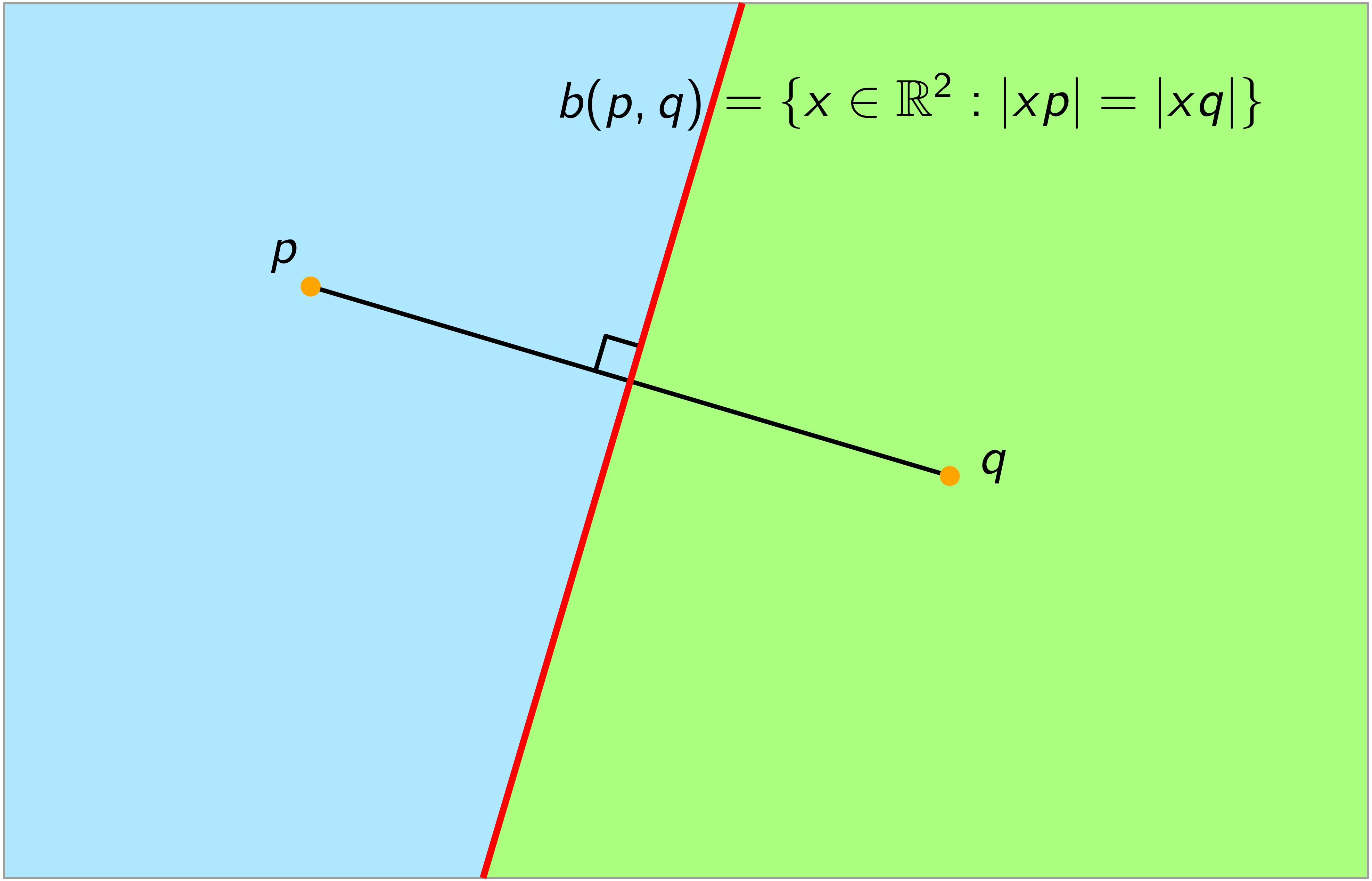
The Post-Office Problem



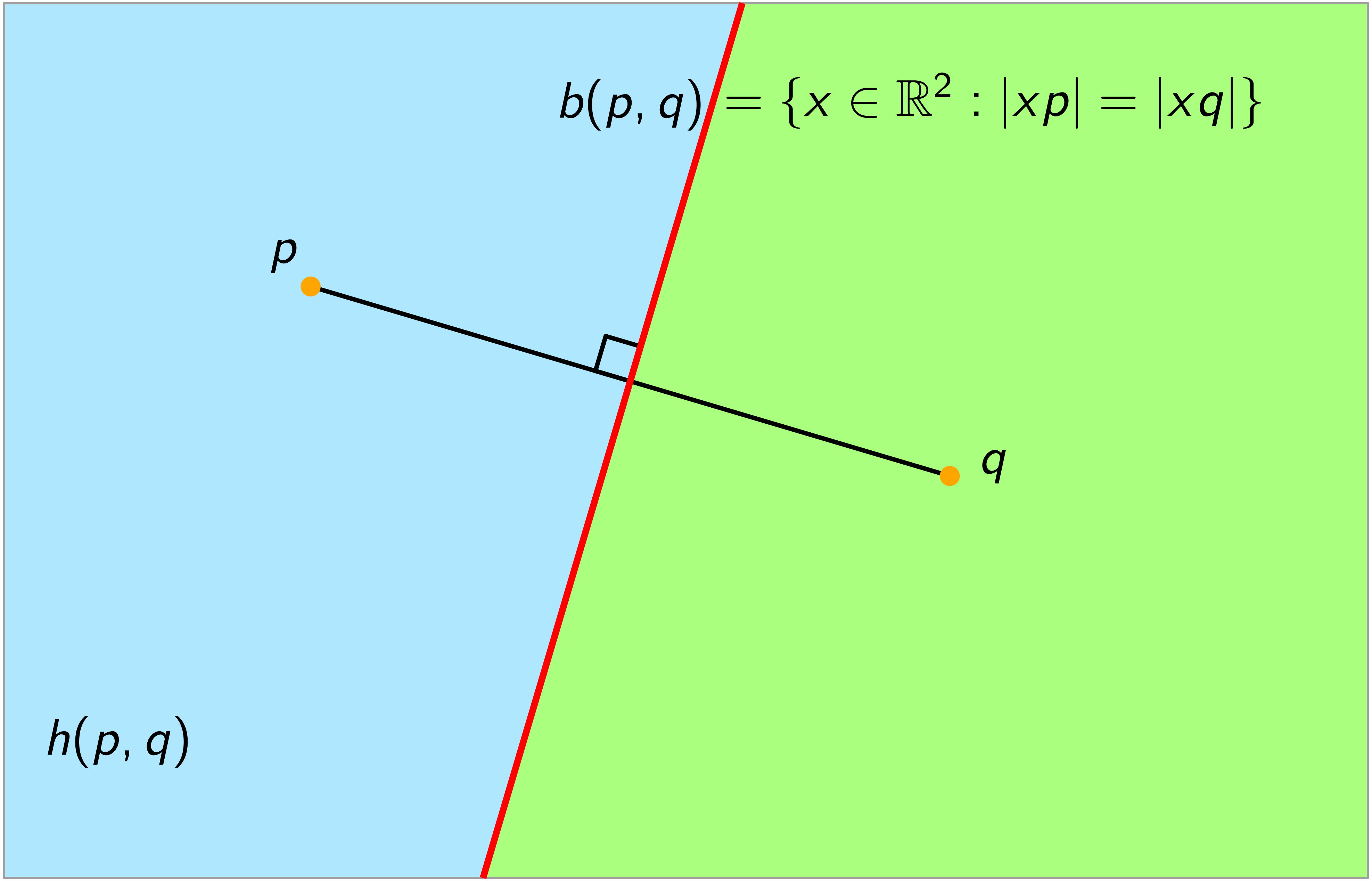
The Post-Office Problem



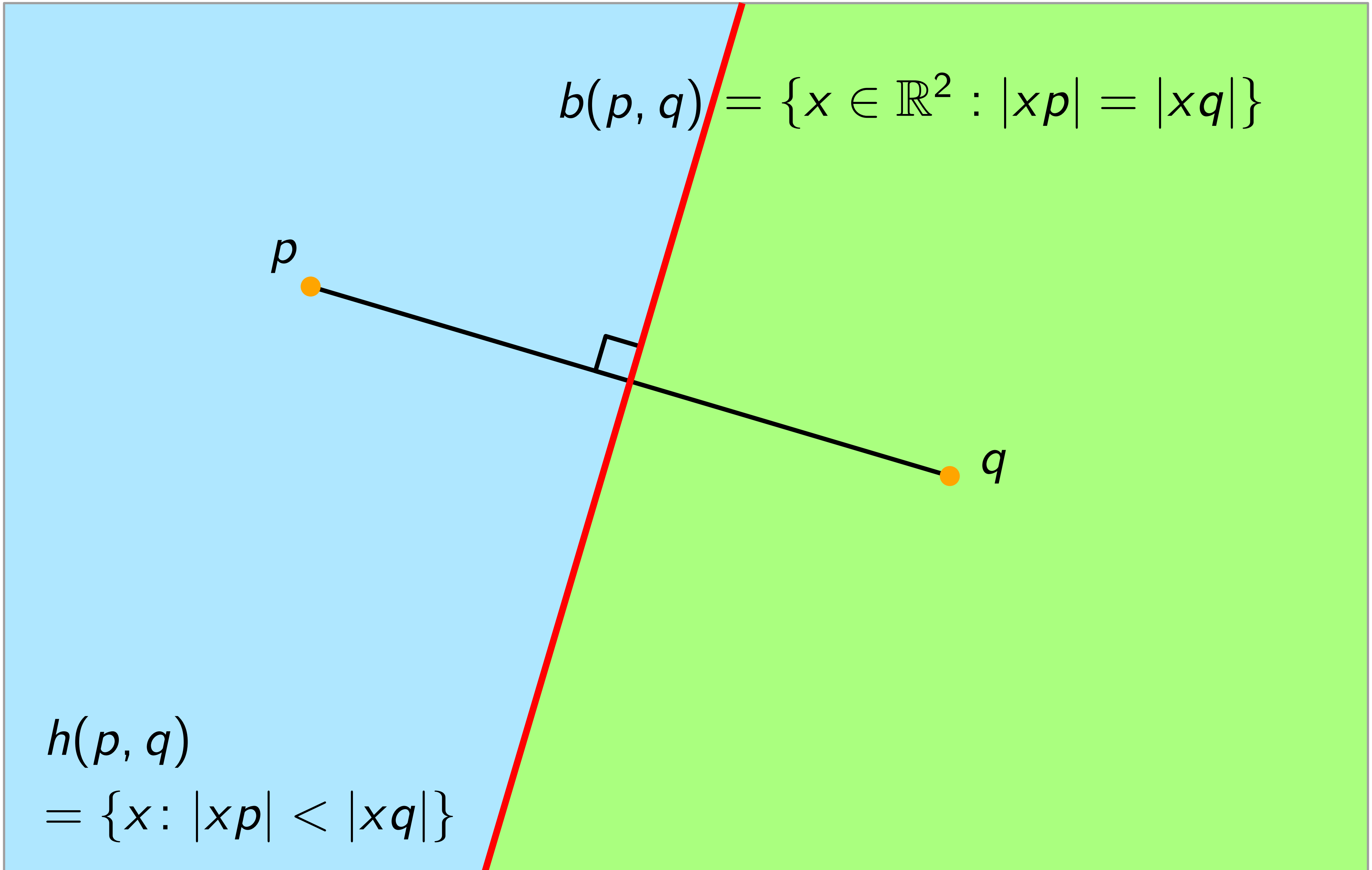
The Post-Office Problem



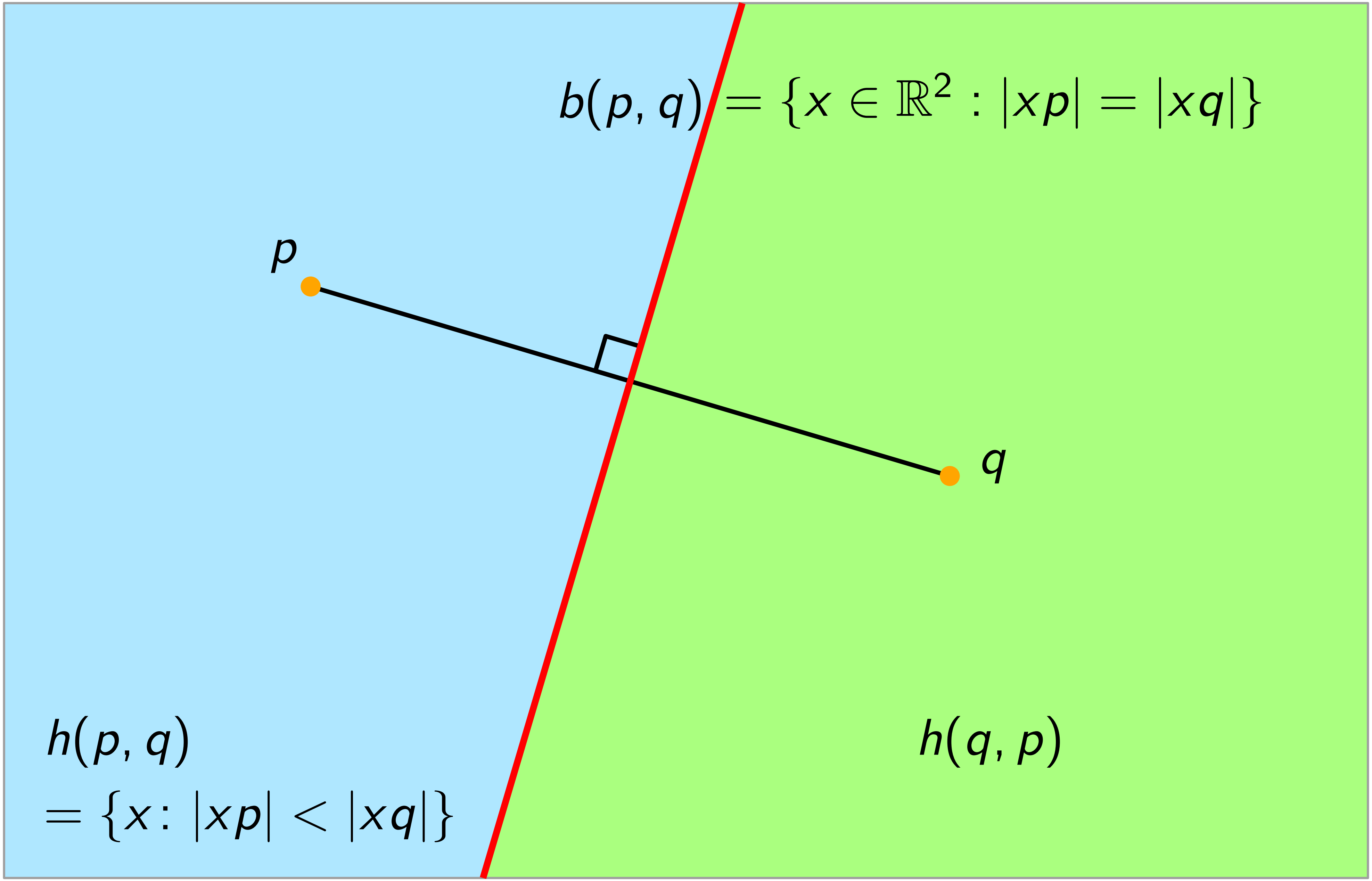
The Post-Office Problem



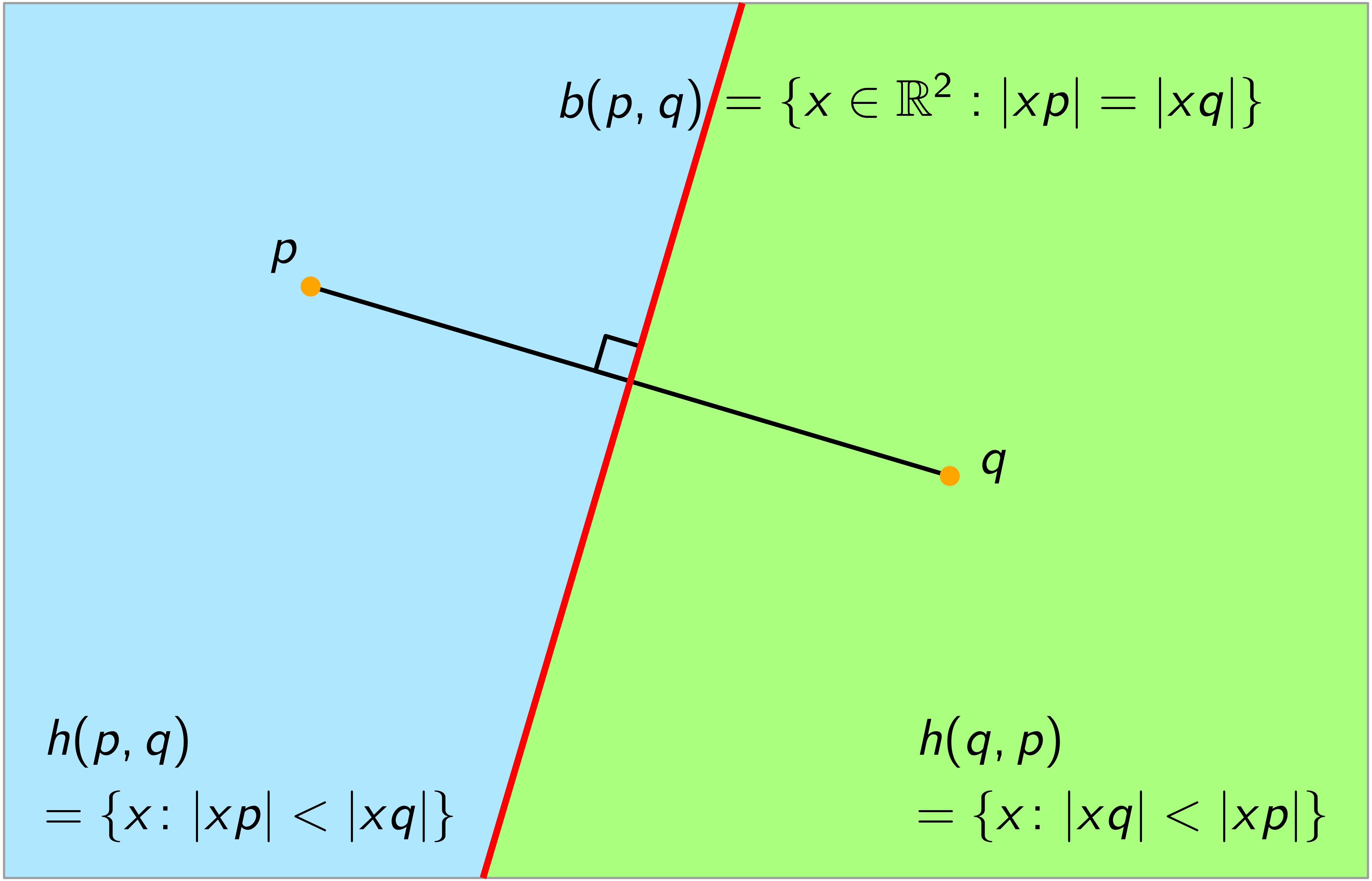
The Post-Office Problem



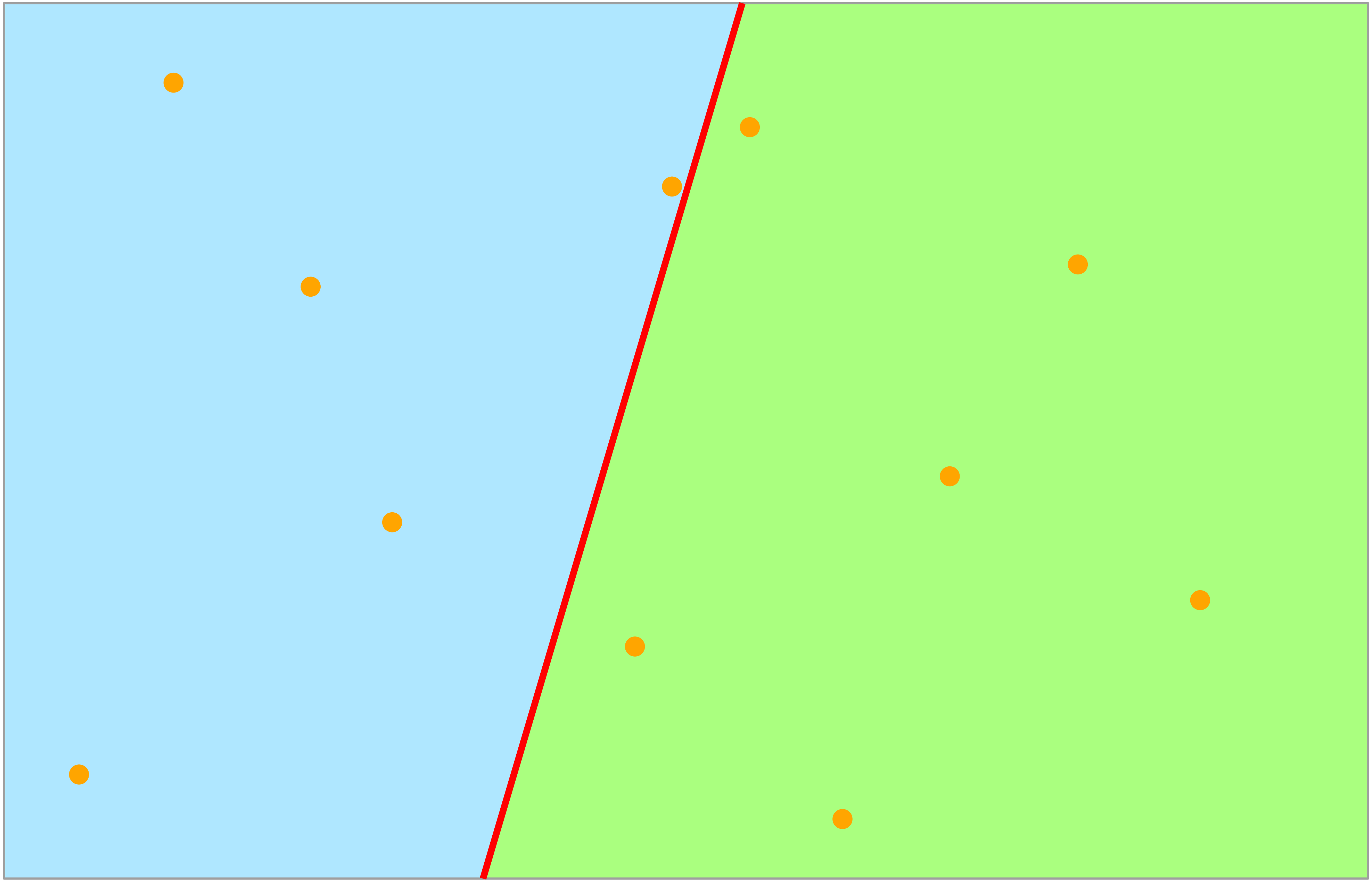
The Post-Office Problem



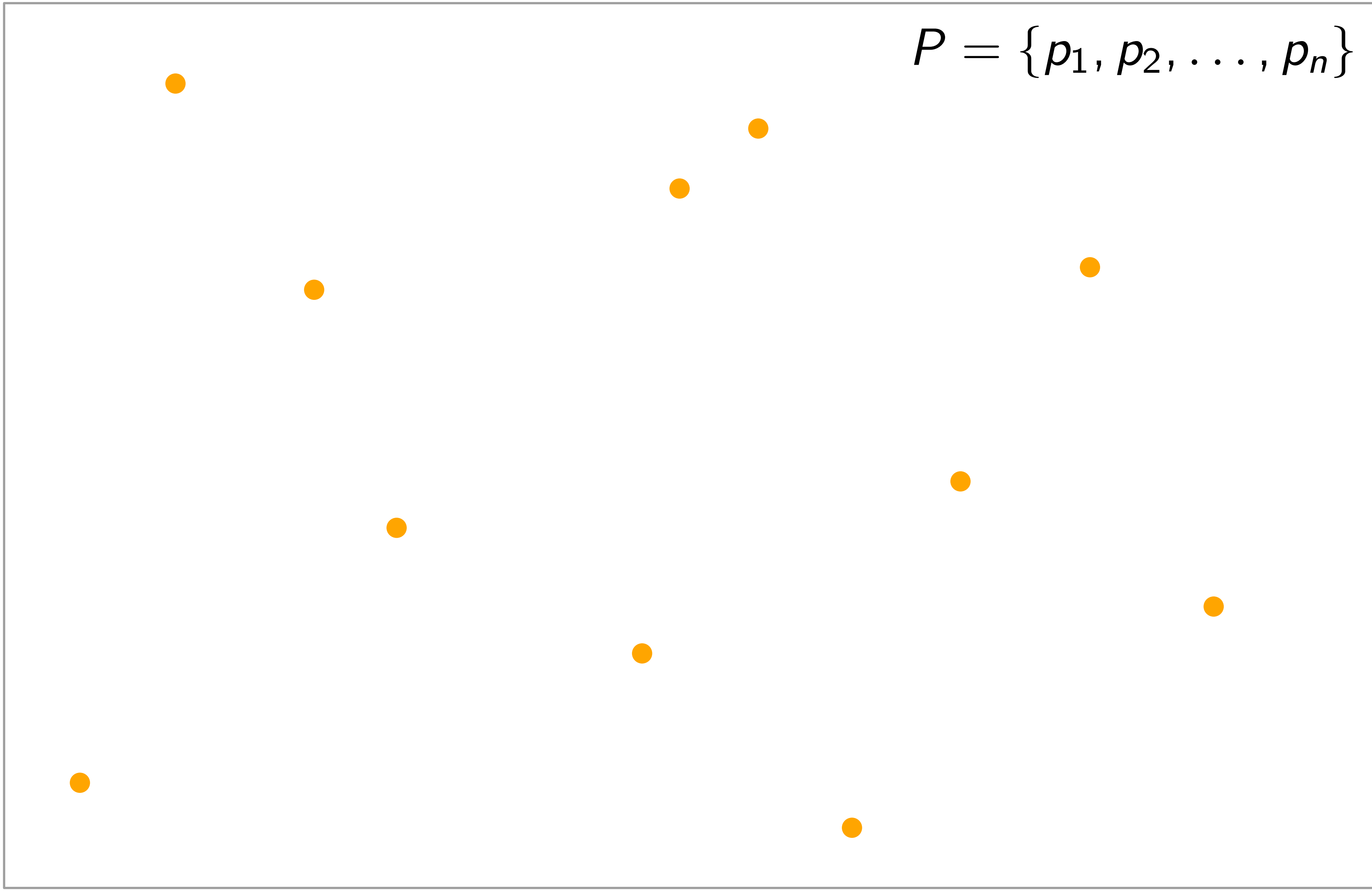
The Post-Office Problem



The Post-Office Problem

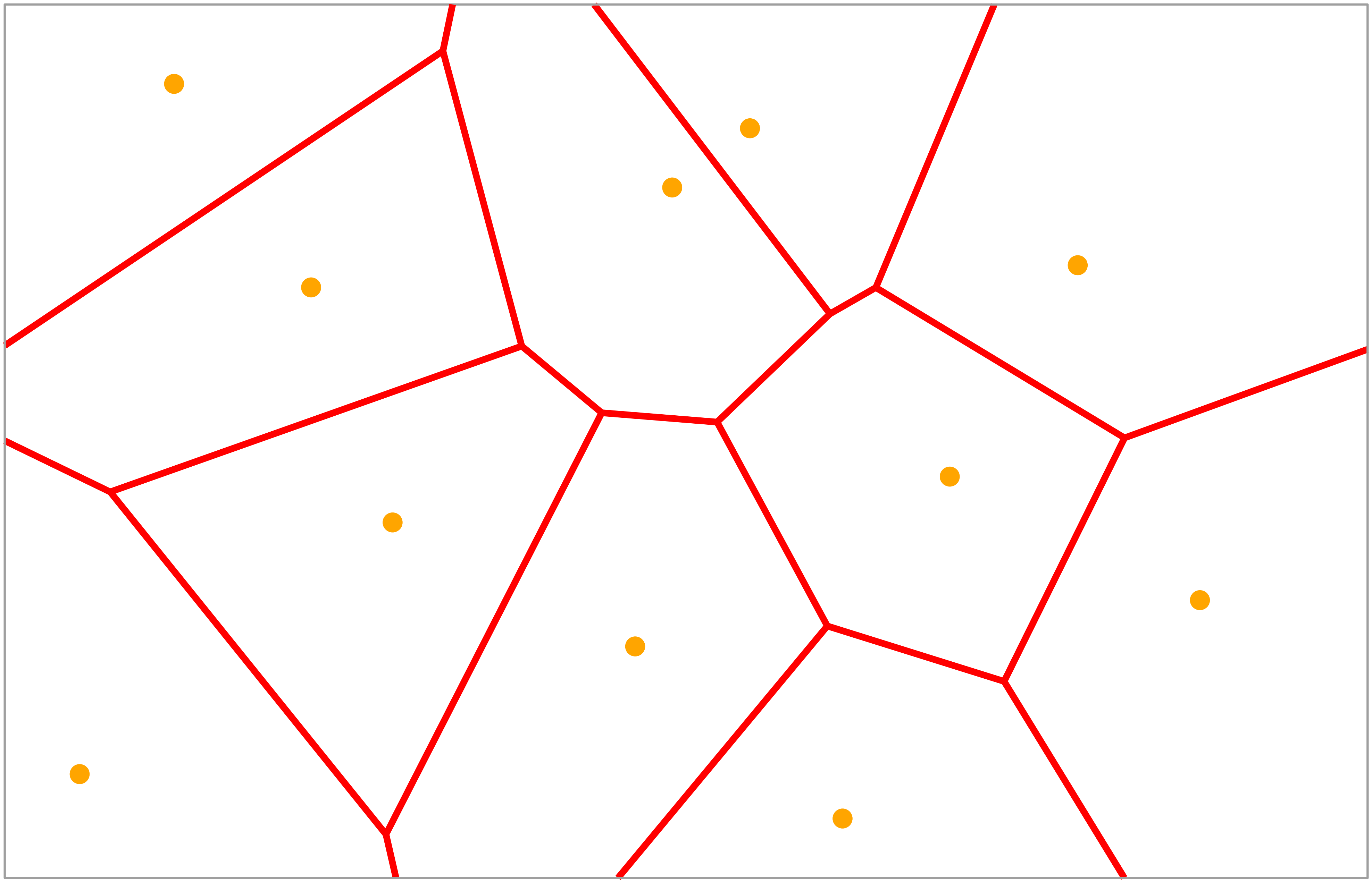


The Post-Office Problem

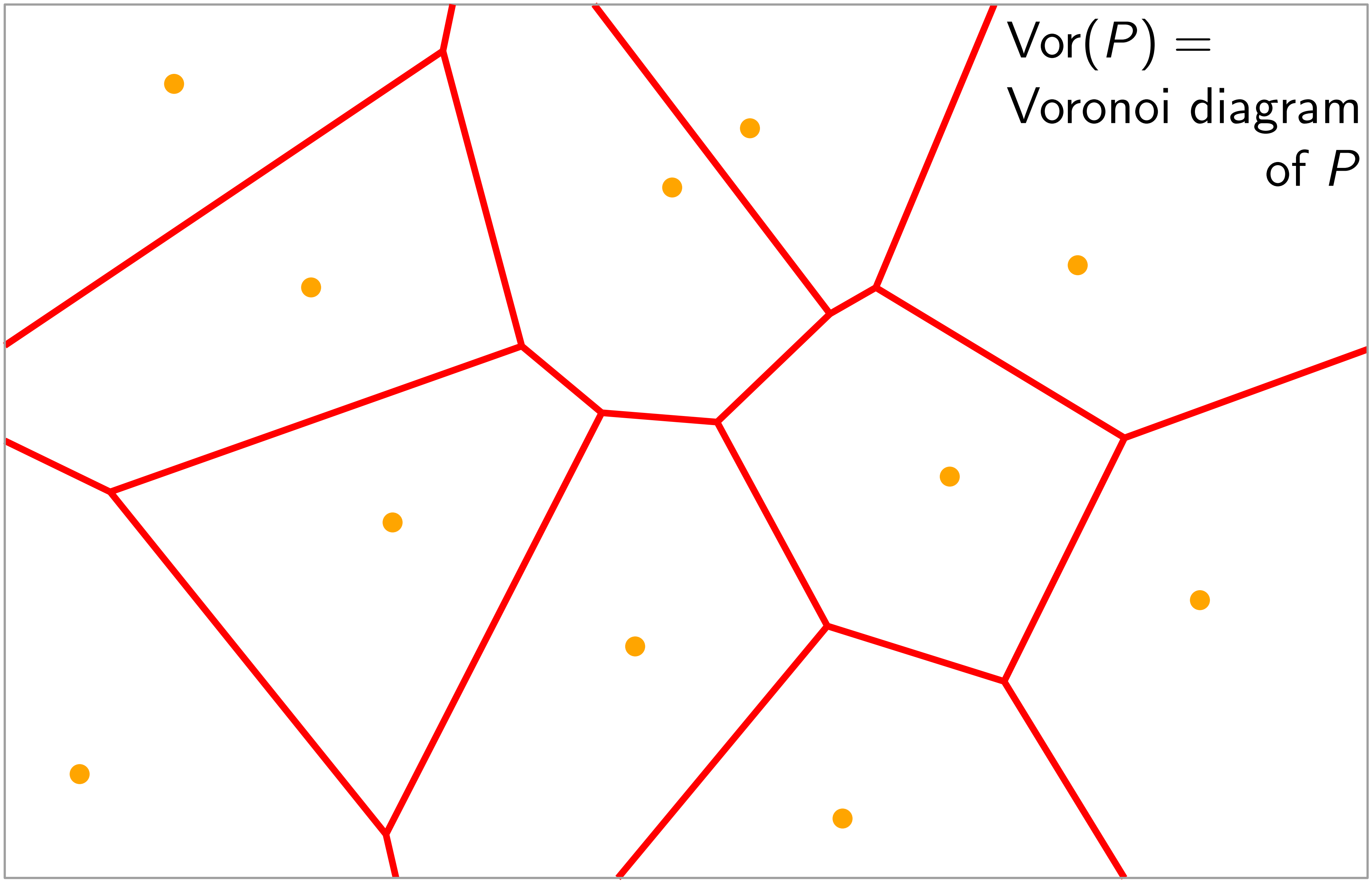


$$P = \{p_1, p_2, \ldots, p_n\}$$

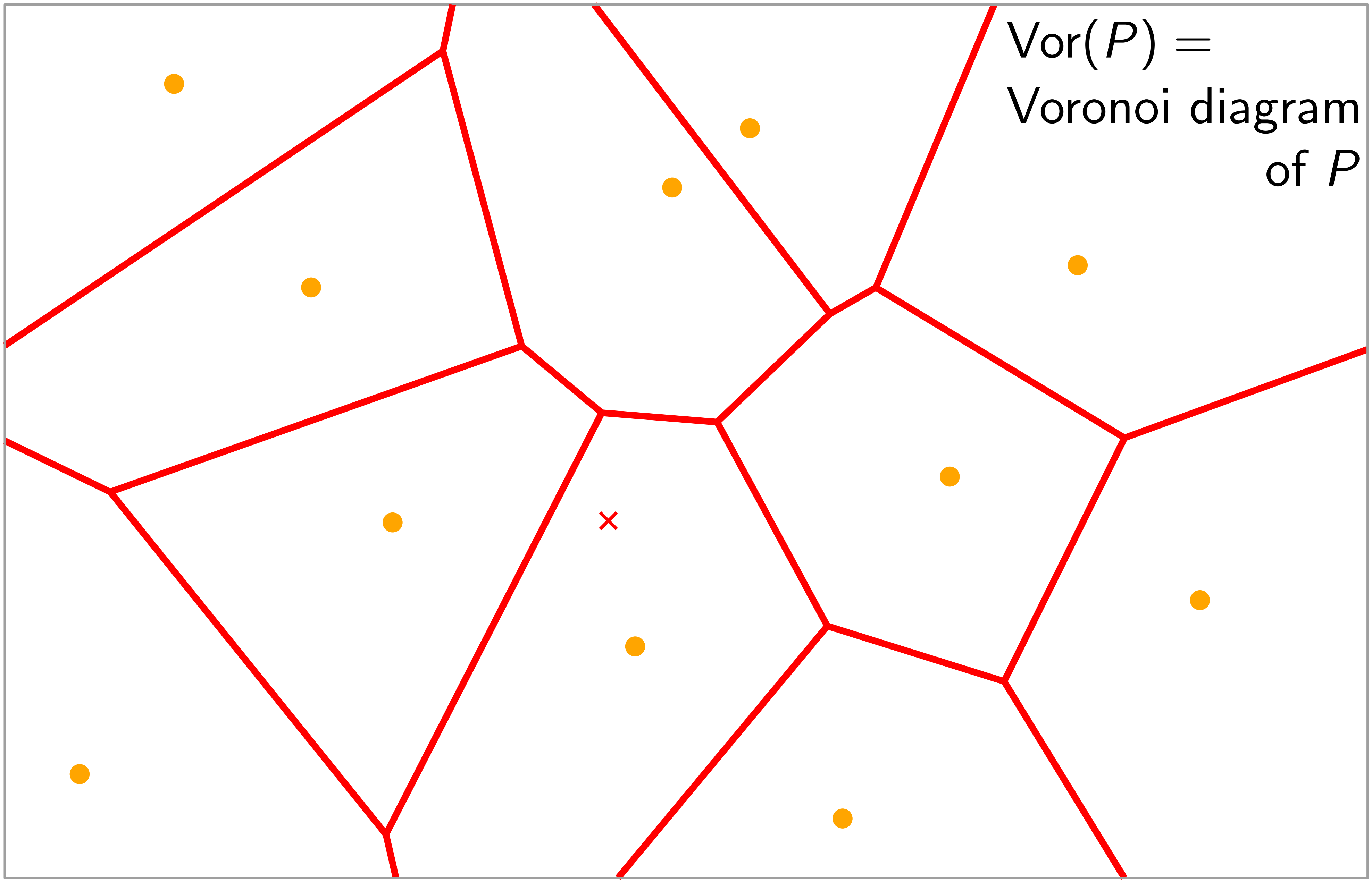
The Post-Office Problem



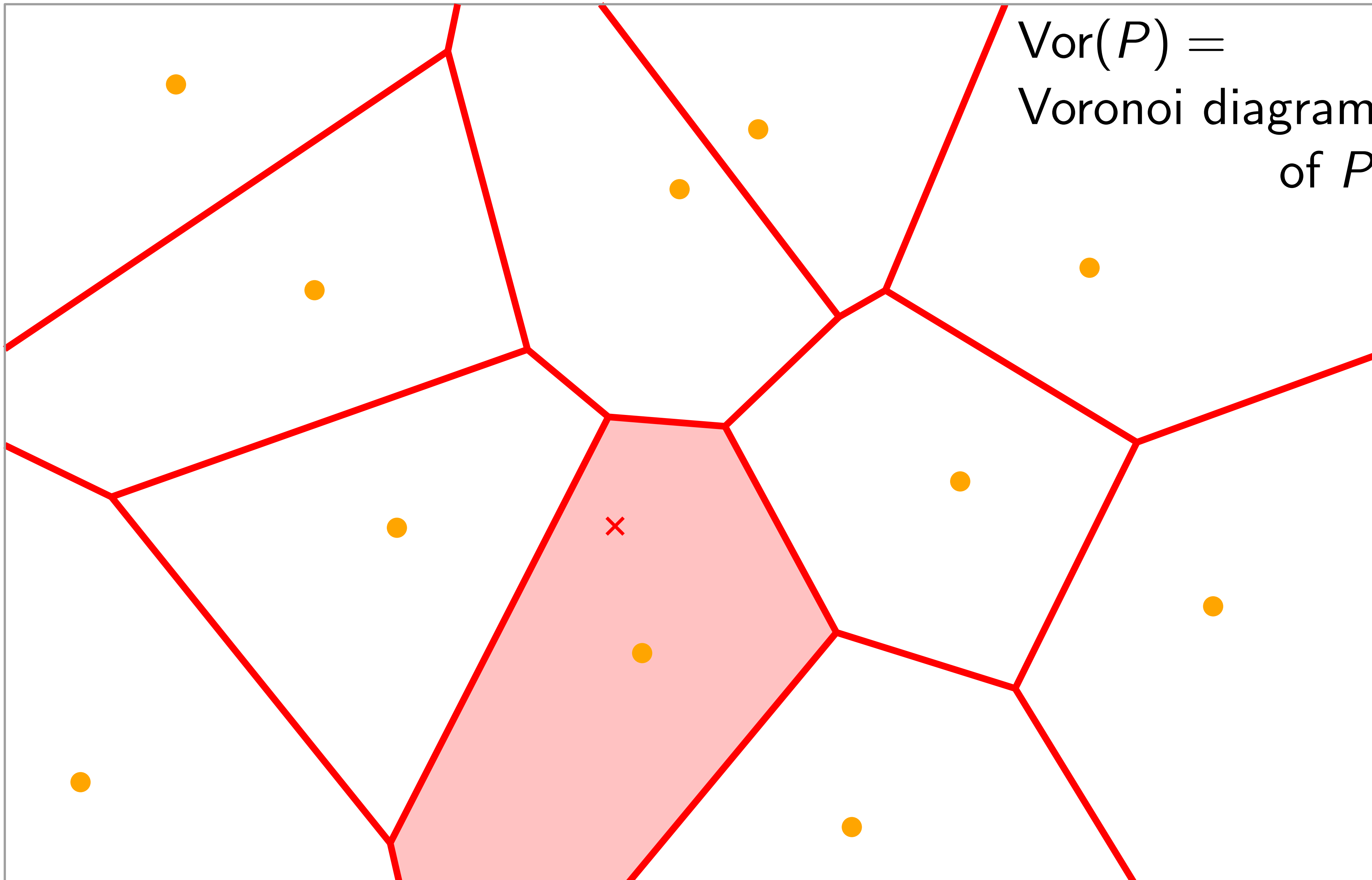
The Post-Office Problem



The Post-Office Problem

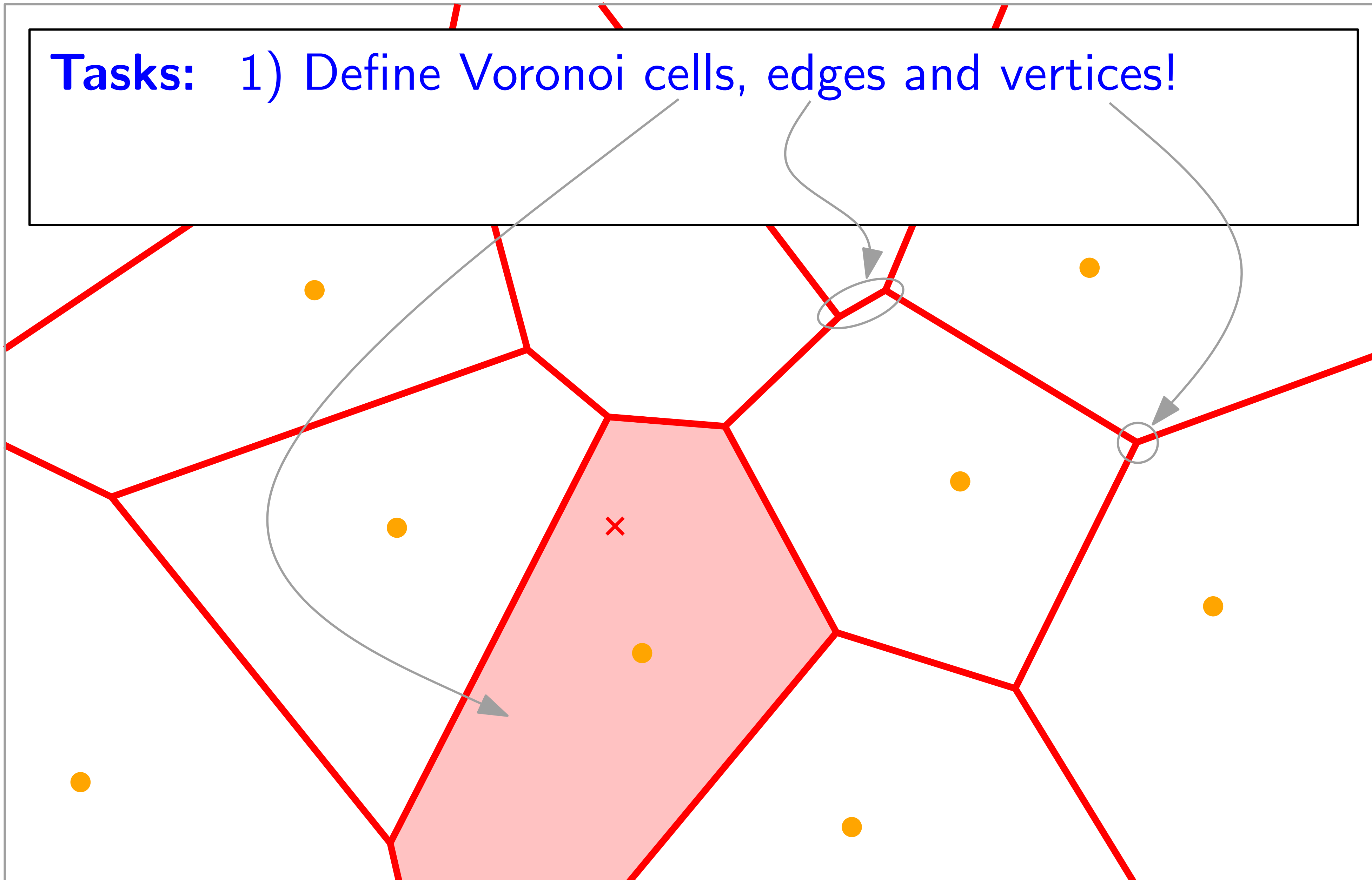


The Post-Office Problem



The Post-Office Problem

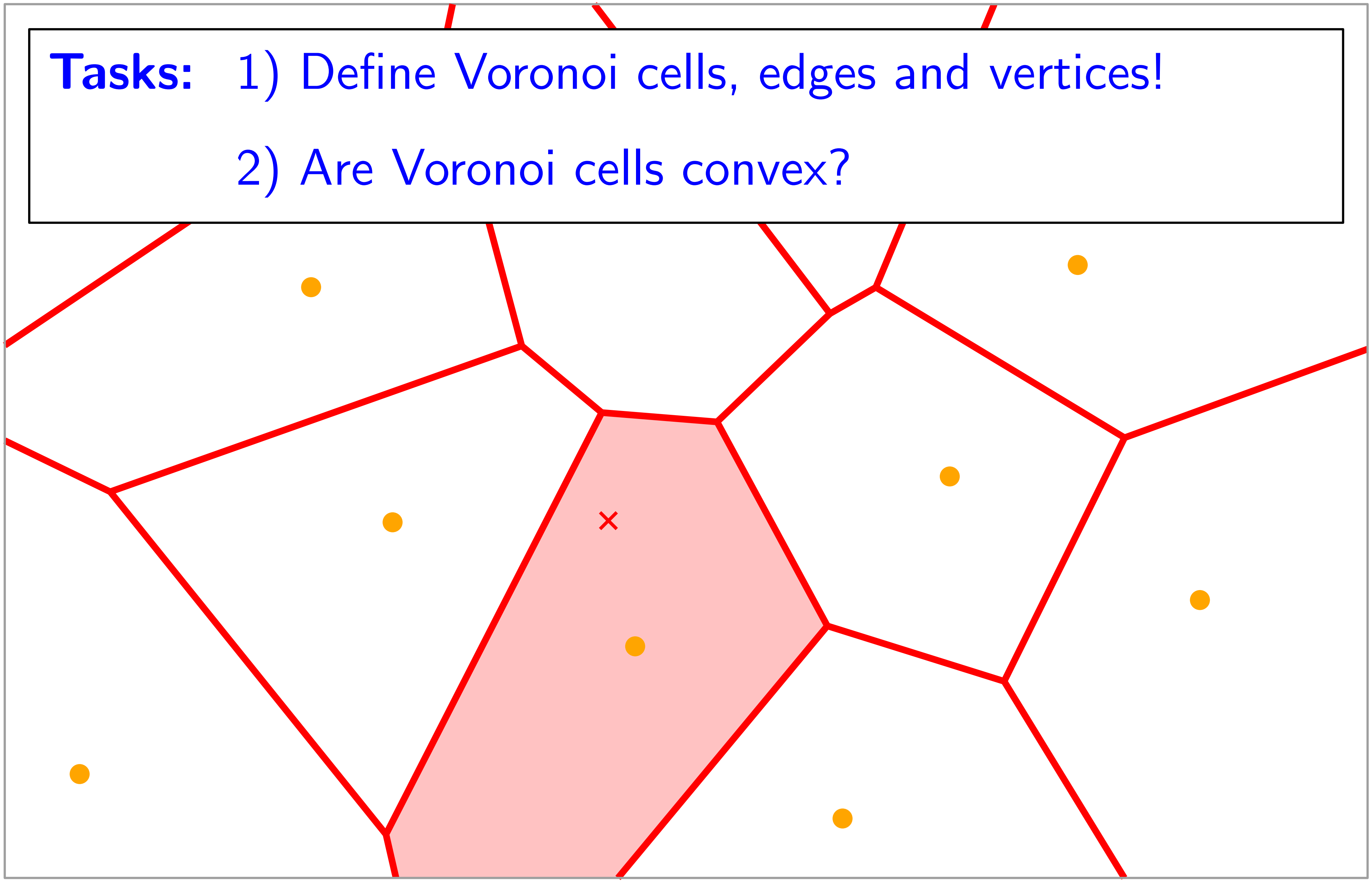
Tasks: 1) Define Voronoi cells, edges and vertices!



The Post-Office Problem

Tasks: 1) Define Voronoi cells, edges and vertices!

2) Are Voronoi cells convex?



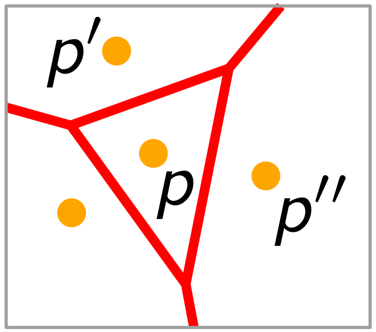
The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]

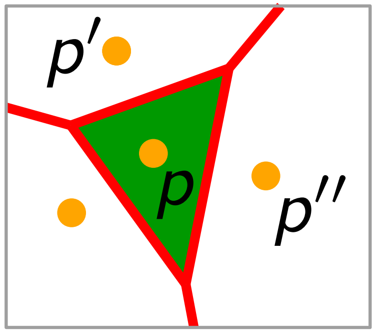


Vor(P)

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]

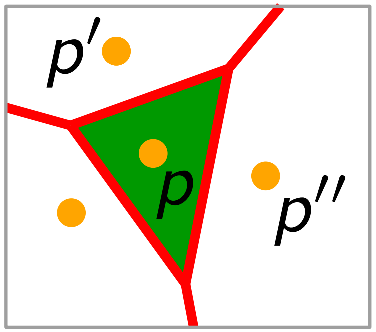


Vor(P)

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



$\text{Vor}(P)$

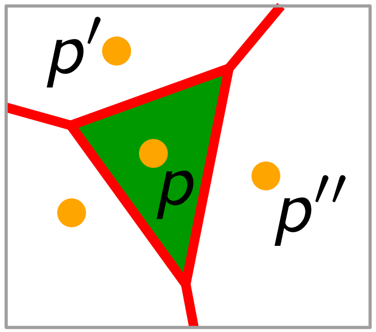
[*Voronoi cell*]

$\mathcal{V}(\{p\}) =$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



Vor(P)

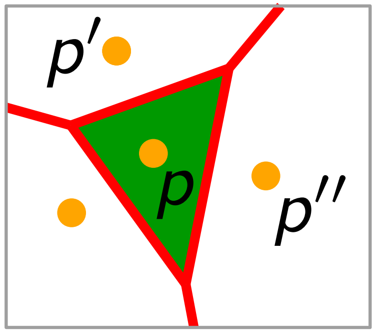
[*Voronoi cell*]

$\mathcal{V}(\{p\}) = \mathcal{V}(p) =$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



$\text{Vor}(P)$

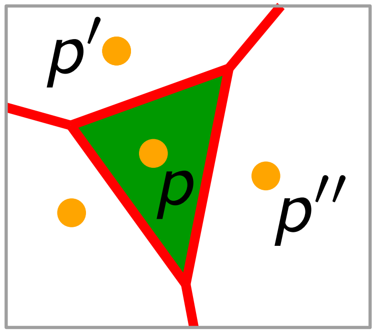
[*Voronoi cell*]

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \left\{ x \in \mathbb{R}^2 : |xp| < |xq| \text{ for all } q \in P \setminus \{p\} \right\}$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



Vor(P)

[*Voronoi cell*]

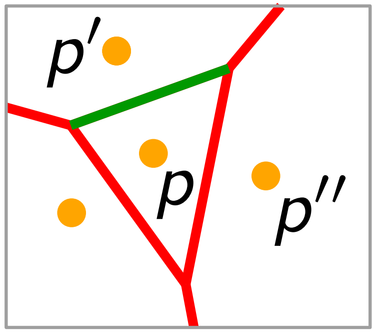
$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \{x \in \mathbb{R}^2 : |xp| < |xq| \text{ for all } q \in P \setminus \{p\}\}$$

$$= \bigcap_{q \neq p} h(p, q)$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



$\text{Vor}(P)$

[*Voronoi cell*]

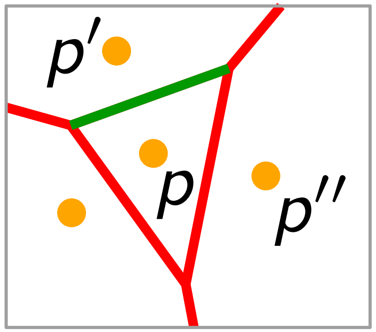
$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \left\{x \in \mathbb{R}^2 : |xp| < |xq| \text{ for all } q \in P \setminus \{p\}\right\}$$

$$= \bigcap_{q \neq p} h(p, q)$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



Vor(P)

[*Voronoi cell*]

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

$$= \bigcap_{q \neq p} h(p, q)$$

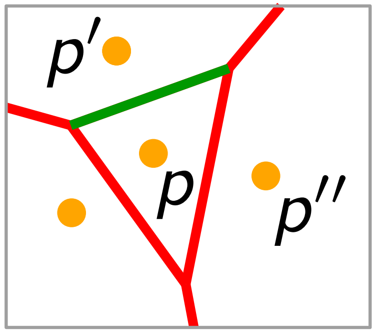
[*Voronoi edge*]

$$\mathcal{V}(\{p, p'\}) =$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



Vor(P)

[*Voronoi cell*]

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

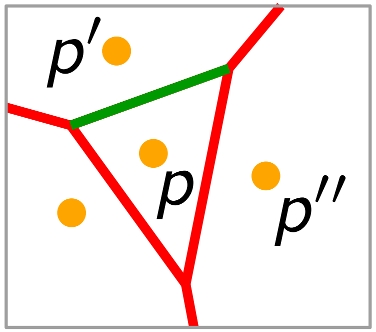
[*Voronoi edge*]

$$\mathcal{V}(\{p, p'\}) = \{x \colon |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[Voronoi diagram]



$\text{Vor}(P)$

[Voronoi cell]

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

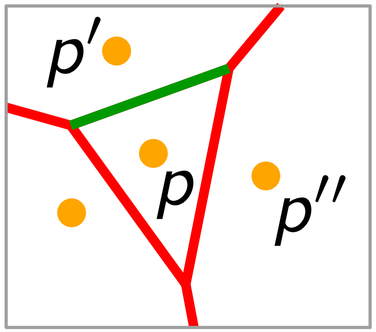
[Voronoi edge]

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$
$$= \partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[Voronoi diagram]



Vor(P)

[Voronoi cell]

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

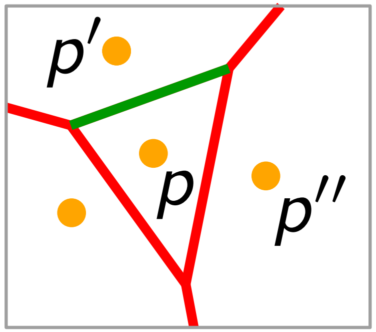
[Voronoi edge]

$$\mathcal{V}(\{p, p'\}) = \{x \colon |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \text{rel-int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big)$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



Vor(P)

[*Voronoi cell*]

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

$$= \bigcap_{q \neq p} h(p, q)$$

[*Voronoi edge*]

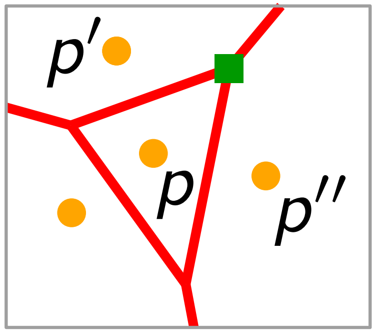
$$\mathcal{V}(\{p, p'\}) = \{x \colon |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$

$$= \text{rel-int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big), \text{ i.e., w/o the endpts}$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[Voronoi diagram]



$\text{Vor}(P)$

[Voronoi cell]

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

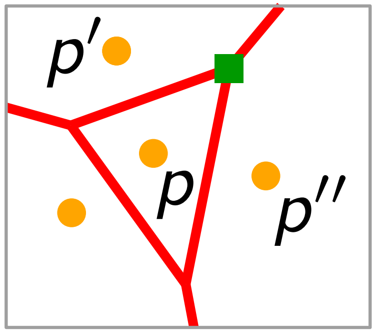
[Voronoi edge]

$$\mathcal{V}(\{p, p'\}) = \{x \colon |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$
$$= \text{rel-int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big), \text{ i.e., w/o the endpts}$$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[Voronoi diagram]



Vor(P)

[Voronoi cell]

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

[Voronoi edge]

$$\mathcal{V}(\{p, p'\}) = \{x \colon |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \text{rel-int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big), \text{ i.e., w/o the endpts}$$

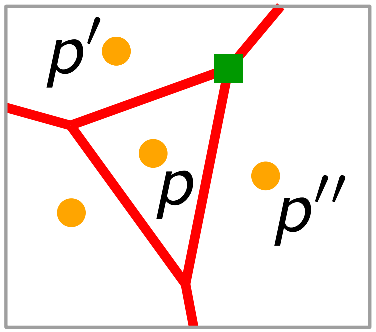
[Voronoi vertex]

$\mathcal{V}(\{p, p', p''\})$

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



$\mathrm{Vor}(P)$

[*Voronoi cell*]

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

[*Voronoi edge*]

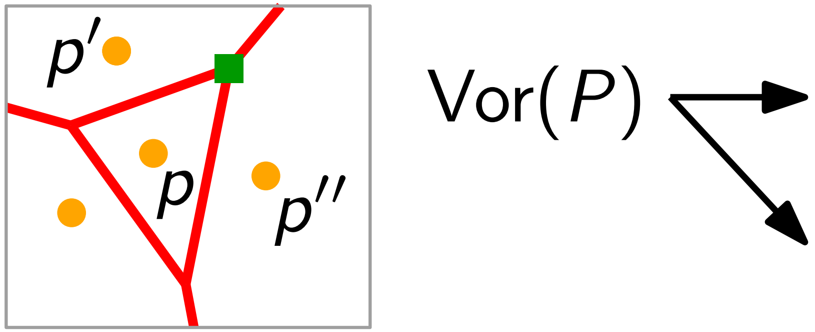
$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \text{rel-int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big), \text{ i.e., w/o the endpts}$$

[*Voronoi vertex*]

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

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*] 

[*Voronoi cell*]

$$\mathcal{V}(\{p\}) = \mathcal{V}(p) = \left\{x \in \mathbb{R}^2 : |xp| < |xq| \text{ for all } q \in P \setminus \{p\}\right\}$$
$$= \bigcap_{q \neq p} h(p, q)$$

[*Voronoi edge*]

$$\mathcal{V}(\{p, p'\}) = \{x \colon |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \text{rel-int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big), \text{ i.e., w/o the endpts}$$

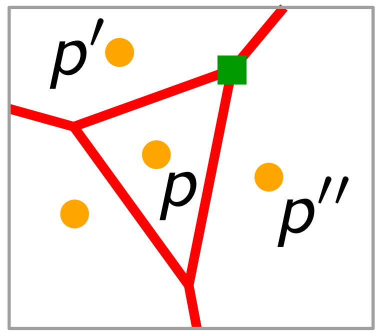
[*Voronoi vertex*]

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

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[Voronoi diagram]



Vor(P) → subdivision of $\mathbb{R}^2$

[Voronoi cell]

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

[Voronoi edge]

$$\mathcal{V}(\{p, p'\}) = \{x : |xp| = |xp'| \text{ and } |xp| < |xq| \;\; \forall q \neq p, p'\}$$
$$= \text{rel-int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big), \text{ i.e., w/o the endpts}$$

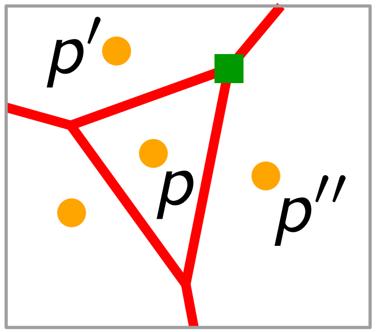
[Voronoi vertex]

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

The Voronoi diagram

Let P be a set of points in the plane and let $p, p', p'' \in P$.

[*Voronoi diagram*]



Vor(P) → subdivision of $\mathbb{R}^2$
→ geometric graph

[*Voronoi cell*]

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

[*Voronoi edge*]

$$\mathcal{V}(\{p, p'\}) = \{x \colon |xp| = |xp'| \text{ and } |xp| < |xq| \ \ \forall q \neq p, p'\}$$
$$= \text{rel-int}\big(\partial\mathcal{V}(p) \cap \partial\mathcal{V}(p')\big), \text{ i.e., w/o the endpts}$$

[*Voronoi vertex*]

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

Overall Shape of Vor(P)

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

Overall Shape of Vor(P)

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

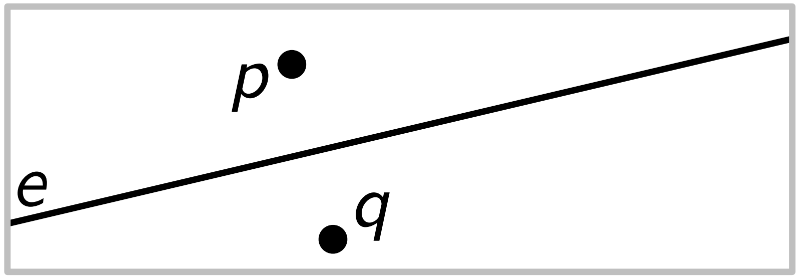
Proof. Assume that P is not collinear.

Overall Shape of Vor(P)

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

Proof. Assume that P is not collinear.

– Assume that Vor(P) contains an edge e that is a full line, say, $e = b(p, q)$.

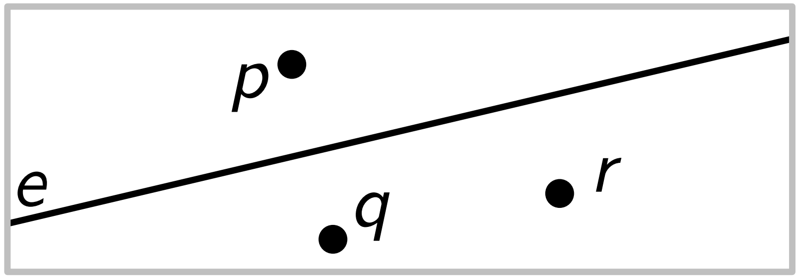


Overall Shape of Vor(P)

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

Proof. Assume that P is not collinear.

– Assume that Vor(P) contains an edge e that is a full line, say, $e = b(p, q)$.



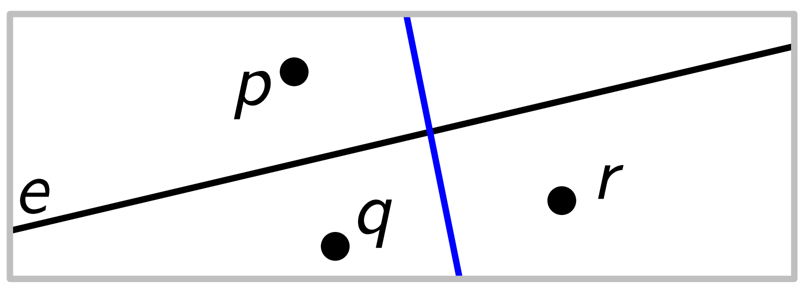
Let $r \in P$ be not collinear with p and q.

Overall Shape of Vor(P)

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

Proof. Assume that P is not collinear.

– Assume that Vor(P) contains an edge e that is a full line, say, $e = b(p, q)$.



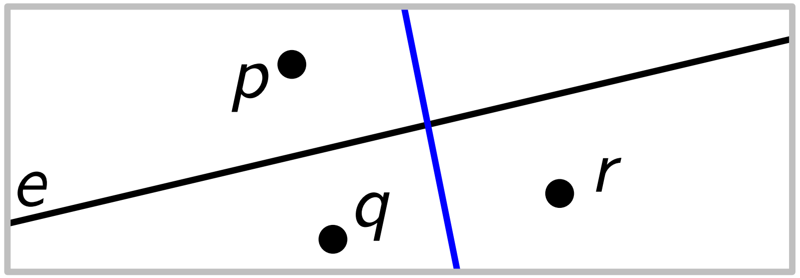
Let $r \in P$ be not collinear with p and q.
Then $b(q, r)$ is not parallel to e.

Overall Shape of Vor(P)

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

Proof. Assume that P is not collinear.

– Assume that Vor(P) contains an edge e that is a full line, say, $e = b(p, q)$.



Let $r \in P$ be not collinear with p and q.

Then $b(q, r)$ is not parallel to e.

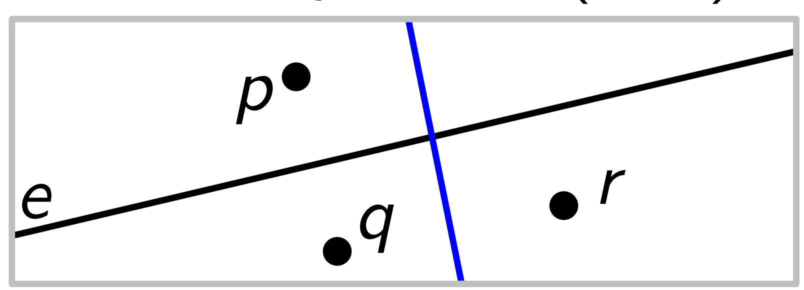
$\Rightarrow e \cap h(r, q)$ is closer to r than to p or q.

Overall Shape of Vor(P)

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

Proof. Assume that P is not collinear.

– Assume that Vor(P) contains an edge e that is a full line, say, $e = b(p, q)$.



Let $r \in P$ be not collinear with p and q.

Then $b(q, r)$ is not parallel to e.

$\Rightarrow$ $e \cap h(r, q)$ is closer to r than to p or q.

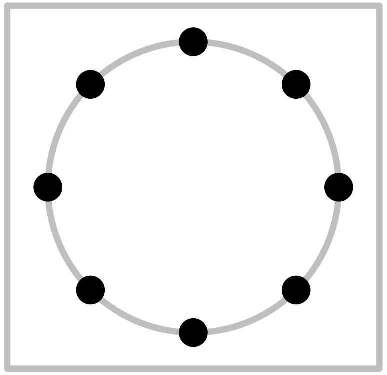
$\Rightarrow$ e is bounded on at least one side. $\square$

Complexity

Task: Construct a set P of point sites such that $\text{Vor}(P)$ has a cell of linear complexity!

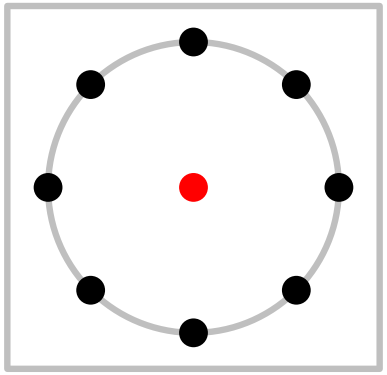
Complexity

Task: Construct a set P of point sites such that $\text{Vor}(P)$ has a cell of linear complexity!



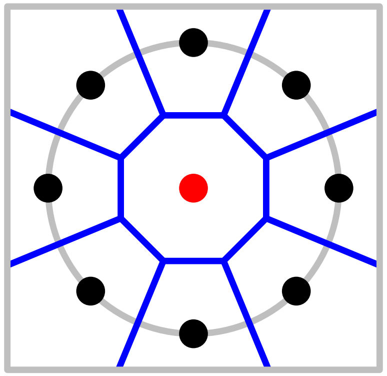
Complexity

Task: Construct a set P of point sites such that $\text{Vor}(P)$ has a cell of linear complexity!



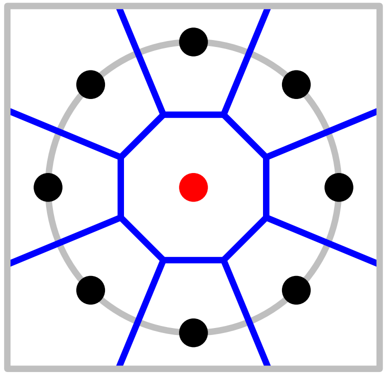
Complexity

Task: Construct a set P of point sites such that $\mathrm{Vor}(P)$ has a cell of linear complexity!



Complexity

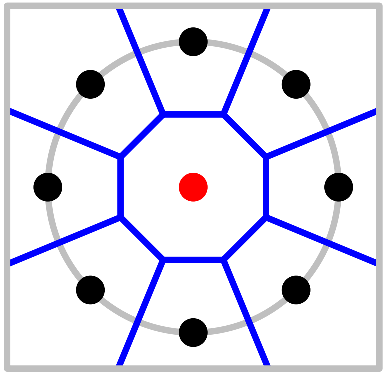
Task: Construct a set P of point sites such that $\text{Vor}(P)$ has a cell of linear complexity!



Theorem. Given a set $P \subset \mathbb{R}^2$ of n sites, $\text{Vor}(P)$ consists of at most [   ] vertices and [   ] edges.

Complexity

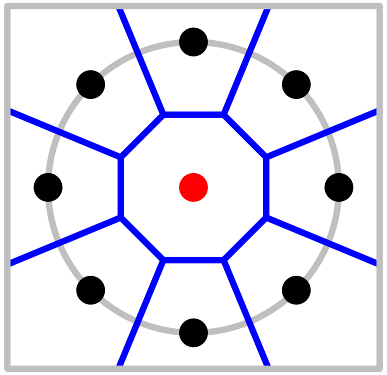
Task: Construct a set P of point sites such that $\text{Vor}(P)$ has a cell of linear complexity!



Theorem. Given a set $P \subset \mathbb{R}^2$ of n sites, $\text{Vor}(P)$ consists of at most $2n - 5$ vertices and $3n - 6$ edges.

Complexity

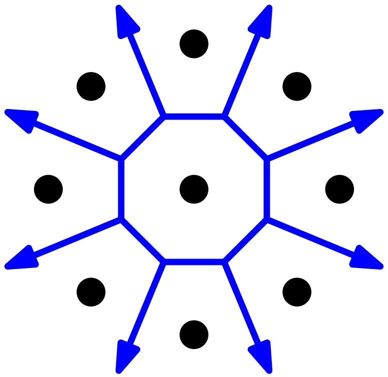
Task: Construct a set P of point sites such that $\text{Vor}(P)$ has a cell of linear complexity!



Theorem. Given a set $P \subset \mathbb{R}^2$ of n sites, $\text{Vor}(P)$ consists of at most $2n - 5$ vertices and $3n - 6$ edges.

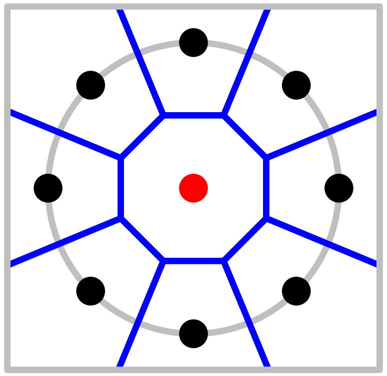
Proof. *Problem:* unbounded edges!

$\Rightarrow$ can't apply Euler directly, but...



Complexity

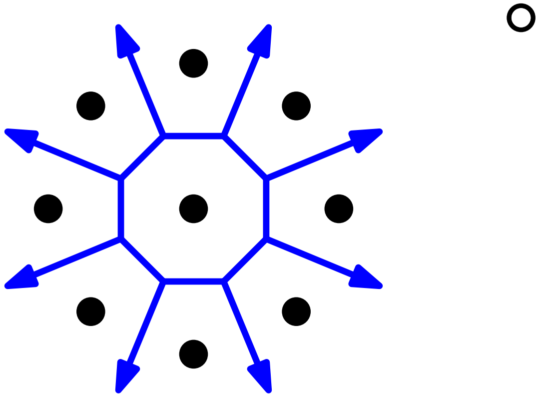
Task: Construct a set P of point sites such that $\text{Vor}(P)$ has a cell of linear complexity!



Theorem. Given a set $P \subset \mathbb{R}^2$ of n sites, $\text{Vor}(P)$ consists of at most $2n - 5$ vertices and $3n - 6$ edges.

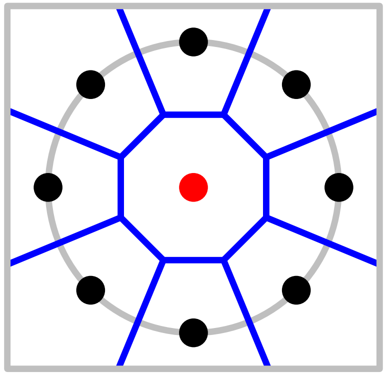
Proof. *Problem:* unbounded edges!

$\Rightarrow$ can't apply Euler directly, but...



Complexity

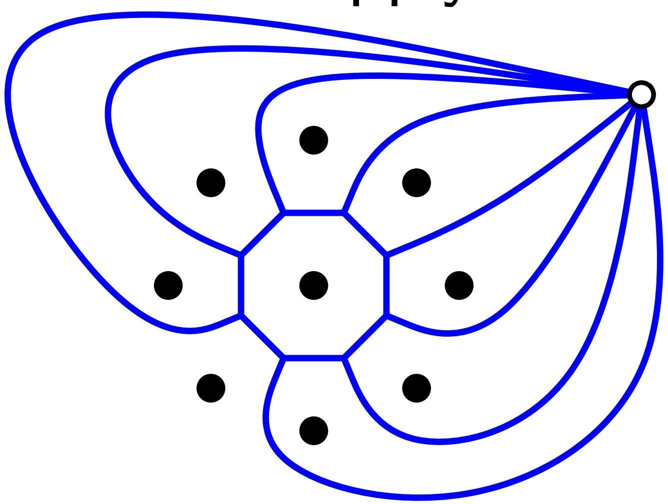
Task: Construct a set P of point sites such that $\text{Vor}(P)$ has a cell of linear complexity!



Theorem. Given a set $P \subset \mathbb{R}^2$ of n sites, $\text{Vor}(P)$ consists of at most $2n - 5$ vertices and $3n - 6$ edges.

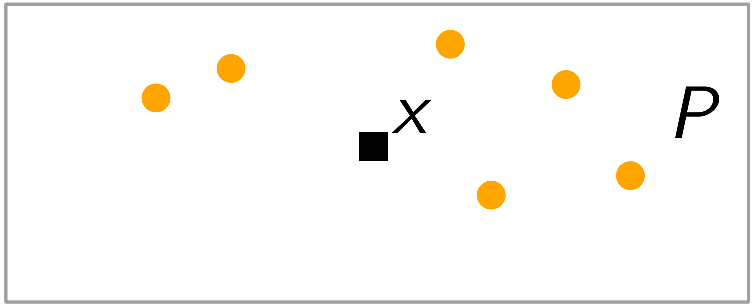
Proof. *Problem:* unbounded edges!

$\Rightarrow$ can't apply Euler directly, but...



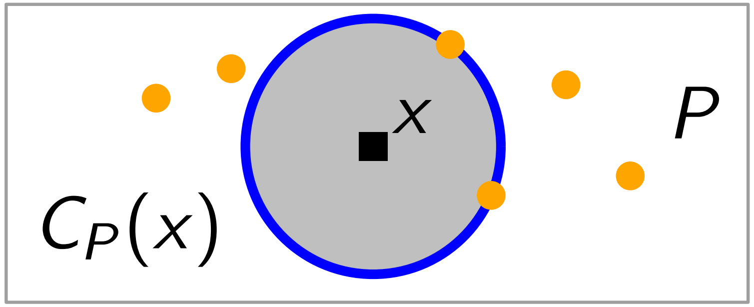
Characterization of Voronoi vtc and edges

$C_P(x) :=$ largest circle centered at x w/o sites in its interior



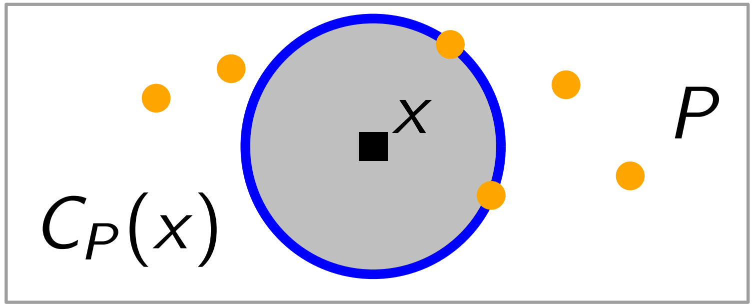
Characterization of Voronoi vtc and edges

$C_P(x) :=$ largest circle centered at x w/o sites in its interior



Characterization of Voronoi vtc and edges

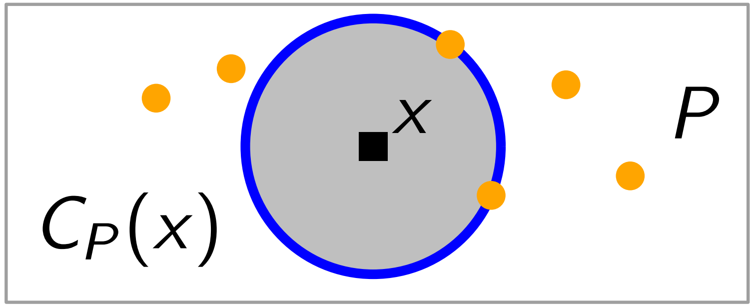
$C_P(x) :=$ largest circle centered at x w/o sites in its interior



Theorem: (i) x Voronoi vtx $\Leftrightarrow$

Characterization of Voronoi vtc and edges

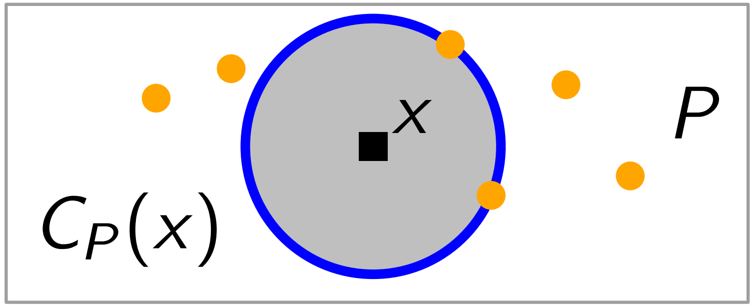
$C_P(x) :=$ largest circle centered at x w/o sites in its interior



Theorem: (i) x Voronoi vtx $\Leftrightarrow |C_P(x) \cap P| \geq 3$

Characterization of Voronoi vtc and edges

$C_P(x) :=$ largest circle centered at x w/o sites in its interior

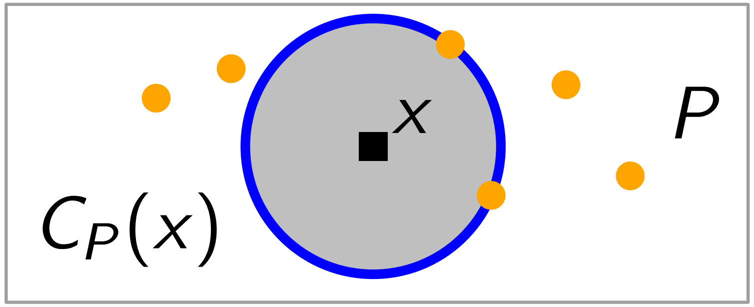


Theorem: (i) x Voronoi vtx $\Leftrightarrow$ $|C_P(x) \cap P| \geq 3$

(ii) $b(p, p')$ contains a Voronoi edge $\Leftrightarrow$

Characterization of Voronoi vtc and edges

$C_P(x) :=$ largest circle centered at x w/o sites in its interior



Theorem: (i) x Voronoi vtx $\Leftrightarrow |C_P(x) \cap P| \geq 3$

(ii) $b(p, p')$ contains a Voronoi edge $\Leftrightarrow$
$\exists x \in b(p, p')\colon C_P(x) \cap P = \{p, p'\}$

Computation

Brute force:

Computation

Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$.

Computation

Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')$.

[Chapter 4]

Computation

Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p, p')}$.

[Chapter 4]

Computation

Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p, p')}$.

[Chapter 4] $O(n \log n)$ time

Computation

Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p, p')}$.

$$\underbrace{\hspace{20em}}$$

[Chapter 4] $O(n \log n)$ time

Computation

Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p, p')}_{O(n \log n) \text{ time}}$.

[Chapter 4]

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{in total: } O(n^2 \log n) \text{ time}}$

Computation

Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p, p')}_{O(n \log n) \text{ time}}$.

[Chapter 4]

$$\underbrace{\hphantom{xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx}}_{\text{in total: } O(n^2 \log n) \text{ time}}$$

– but the complexity of Vor(P) is *linear!*

Computation

Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p, p')}_{O(n \log n) \text{ time}}$.

$$\underbrace{\hspace{15cm}}_{\text{in total: } O(n^2 \log n) \text{ time}}$$

[Chapter 4]

in total: $O(n^2 \log n)$ time

– but the complexity of Vor(P) is *linear!*

Sweep?

Computation

Brute force: For each $p \in P$, compute $\underbrace{\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')}_{}$.

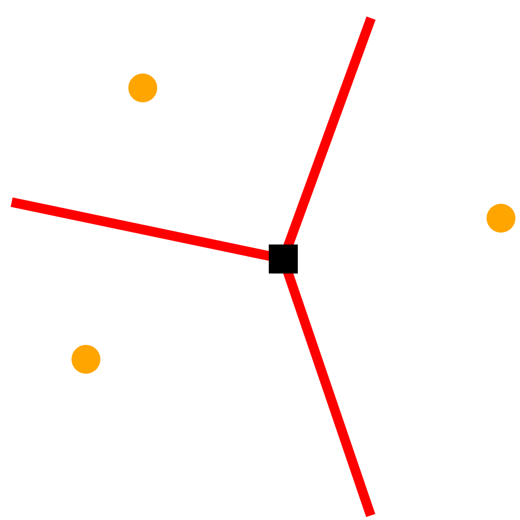
$$\underbrace{\text{For each } p \in P, \text{ compute } \mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p,p')}_{O(n \log n) \text{ time}}}_{\text{in total: } O(n^2 \log n) \text{ time}}$$

[Chapter 4] $O(n \log n)$ time

in total: $O(n^2 \log n)$ time

– but the complexity of $\mathrm{Vor}(P)$ is *linear!*

Sweep?



Computation

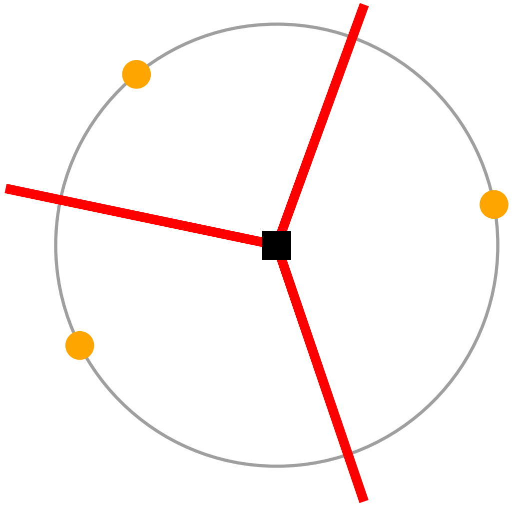
Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p, p')}_{O(n \log n) \text{ time}}$.

[Chapter 4]

in total: $O(n^2 \log n)$ time

– but the complexity of Vor(P) is *linear!*

Sweep?



Computation

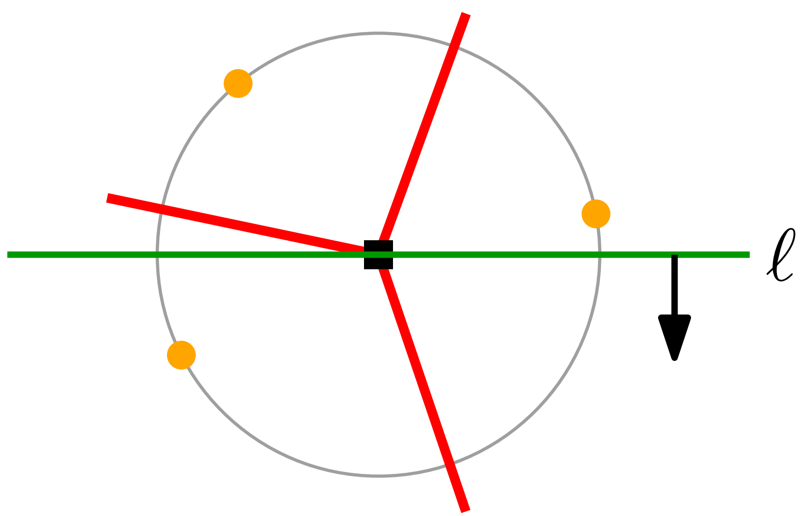
Brute force: For each $p \in P$, compute $\underbrace{\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')}$.

[Chapter 4] $O(n \log n)$ time

in total: $O(n^2 \log n)$ time

– but the complexity of $\mathrm{Vor}(P)$ is *linear!*

Sweep?



Computation

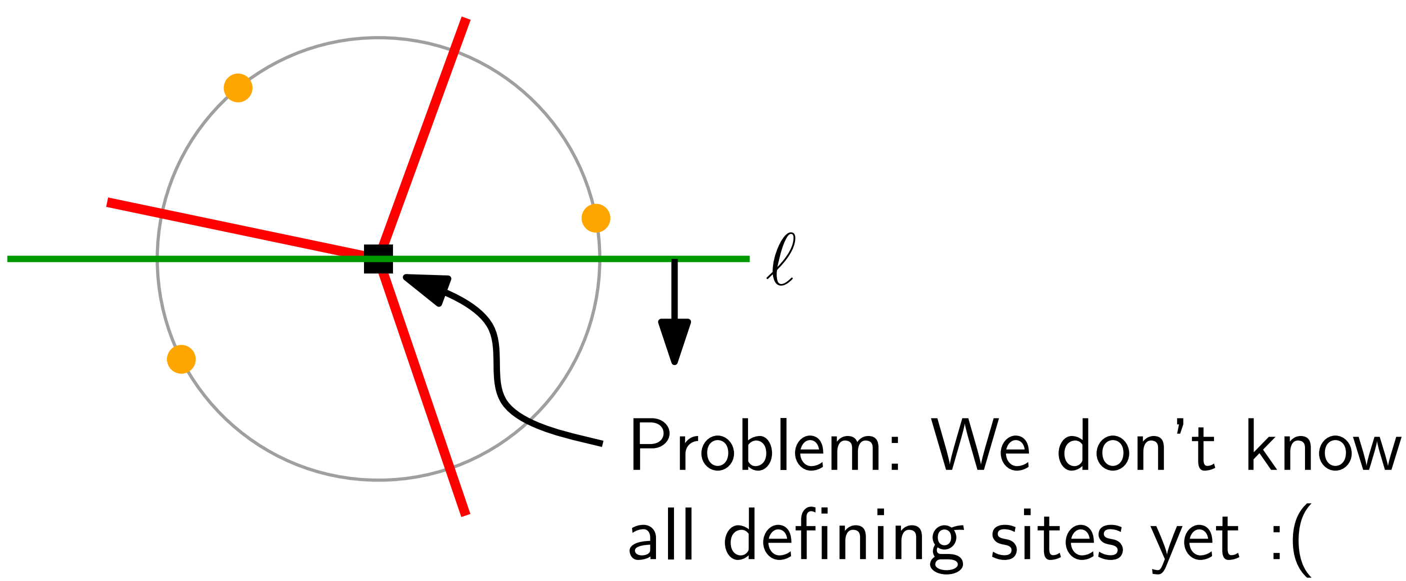
Brute force: For each $p \in P$, compute $\mathcal{V}(p) = \underbrace{\bigcap_{p' \neq p} h(p, p')}$.

[Chapter 4] $O(n \log n)$ time

in total: $O(n^2 \log n)$ time

– but the complexity of Vor(P) is *linear!*

Sweep?



Computation

Brute force: For each $p \in P$, compute $\underbrace{\mathcal{V}(p) = \bigcap_{p' \neq p} h(p, p')}_{O(n \log n) \text{ time}}$.

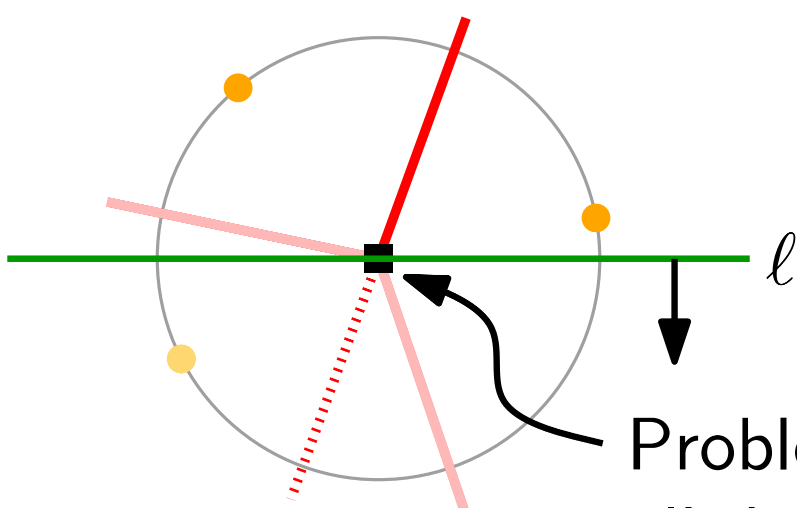
[Chapter 4]

$O(n \log n)$ time

in total: $O(n^2 \log n)$ time

– but the complexity of Vor(P) is *linear!*

Sweep?



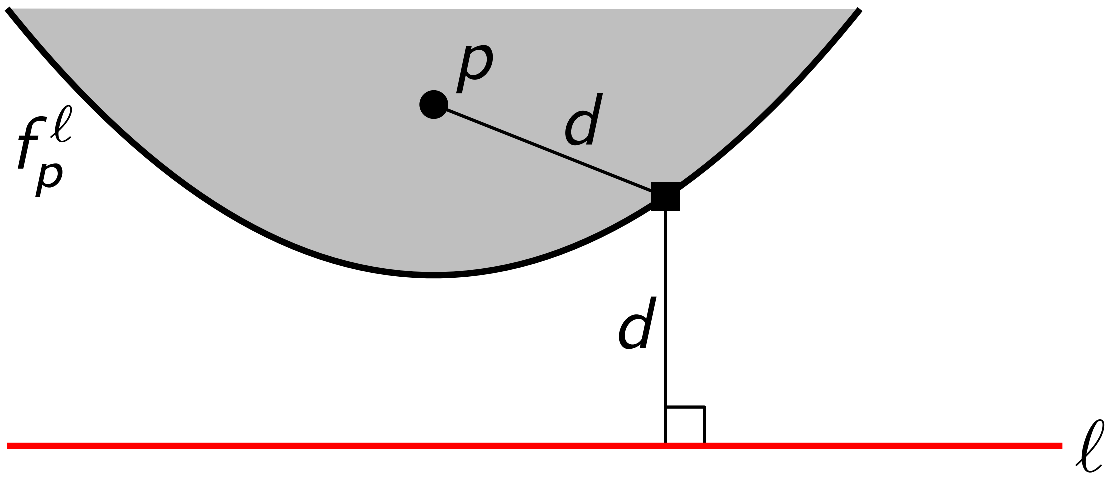
Problem: We don't know all defining sites yet :(

Sweep?

Which part of the plane above ℓ is fixed by what we've seen?

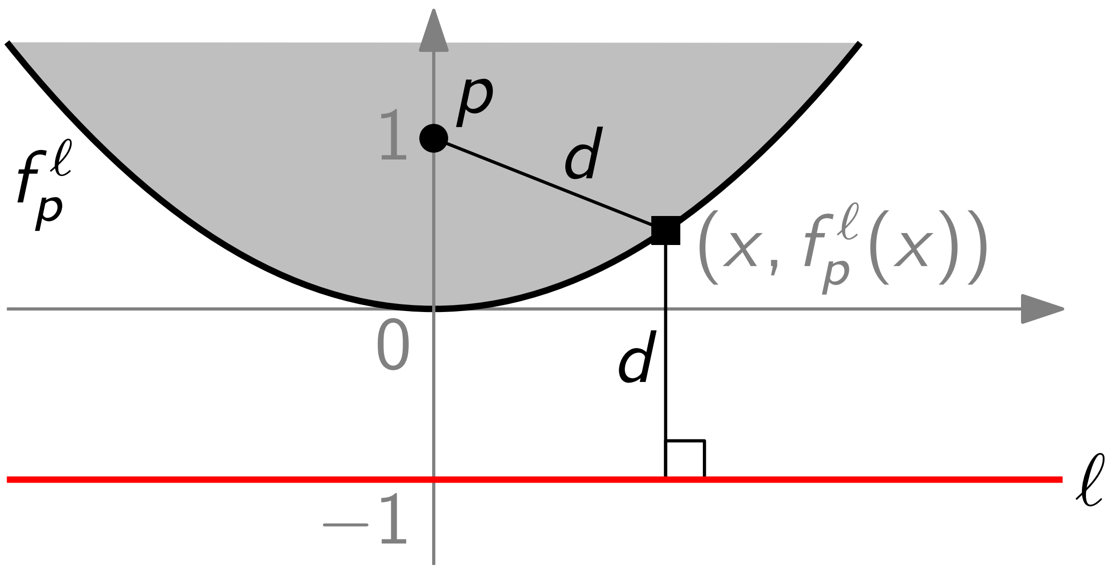
Sweep?

Which part of the plane above ℓ is fixed by what we've seen?



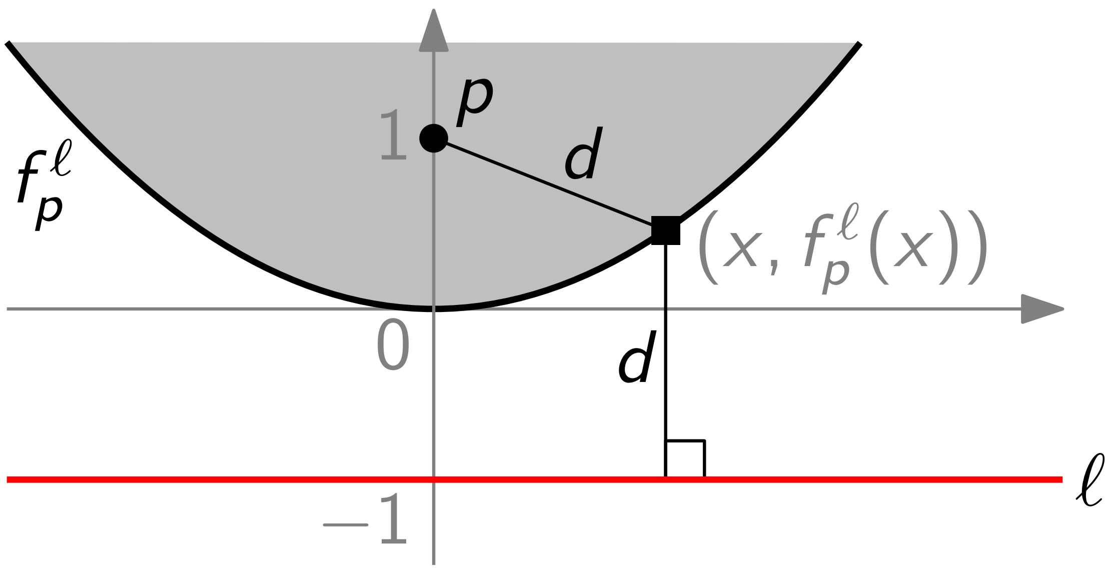
Sweep?

Which part of the plane above ℓ is fixed by what we've seen?



Sweep?

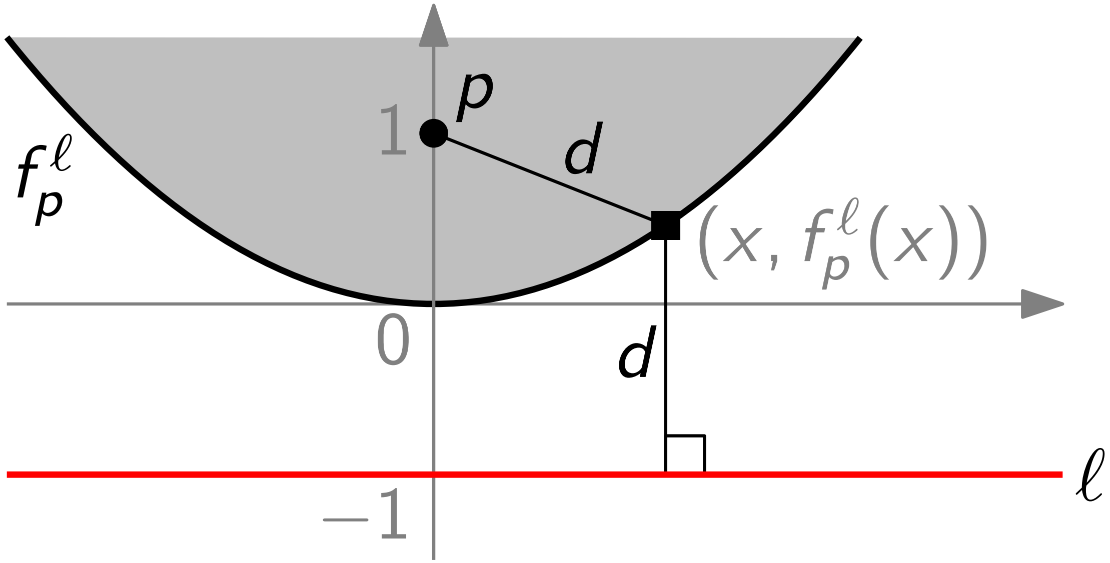
Which part of the plane above ℓ is fixed by what we've seen?



Task: Compute f_p^ℓ for $p = (0, 1)$ and ℓ: $y = -1$!

Sweep?

Which part of the plane above ℓ is fixed by what we've seen?



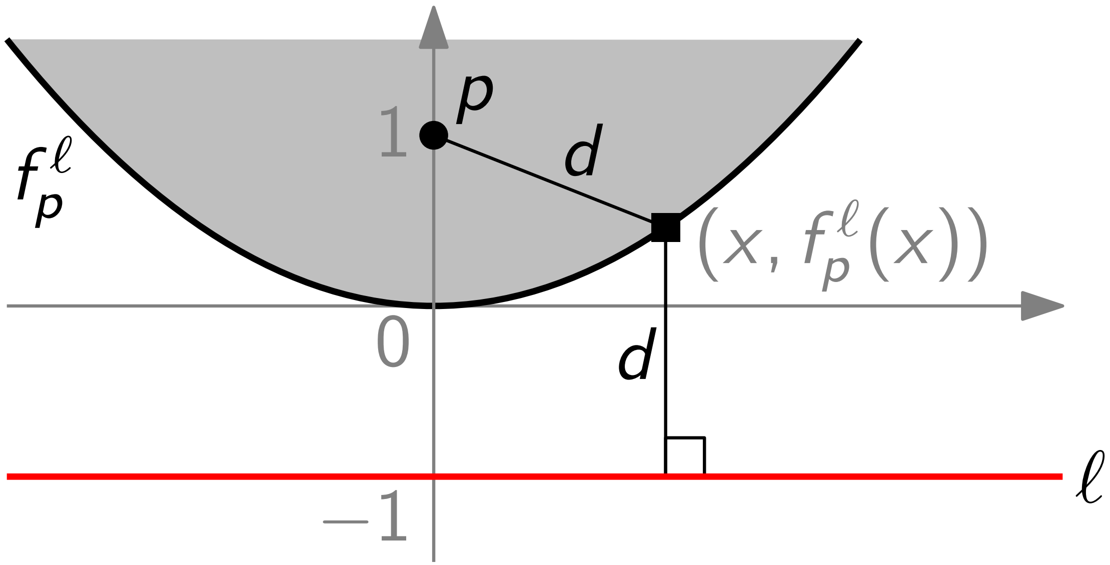
Solution:

f_p^ℓ is the parabola with focus p and directrix ℓ.

Task: Compute f_p^ℓ for $p = (0, 1)$ and ℓ: $y = -1$!

Sweep?

Which part of the plane above ℓ is fixed by what we've seen?



Solution:

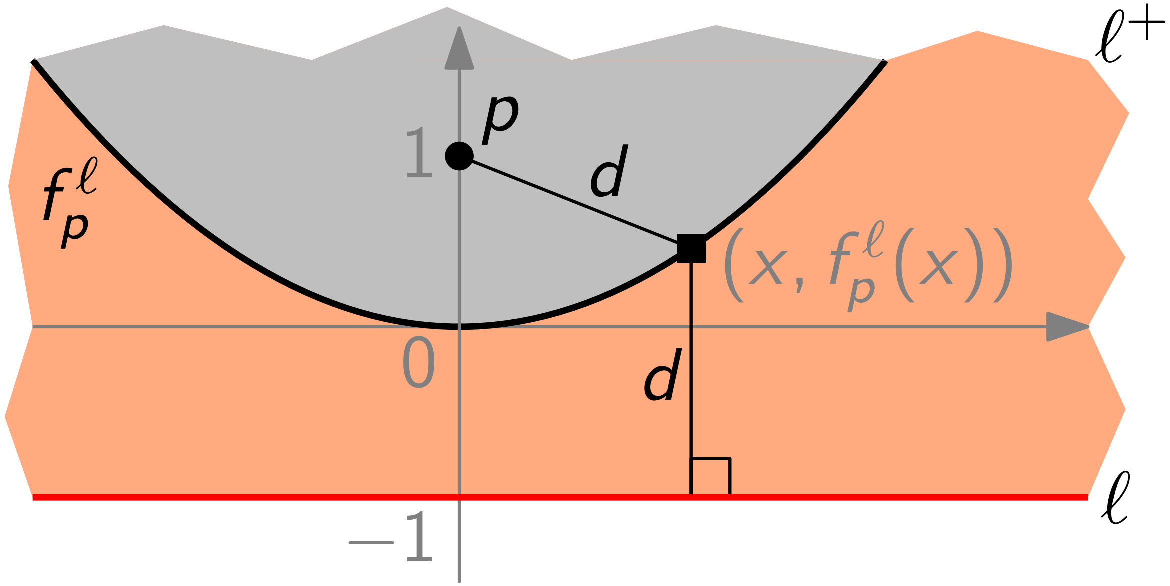
f_p^ℓ is the parabola with focus p and directrix ℓ.

Task: Compute f_p^ℓ for $p = (0, 1)$ and $\ell \colon y = -1$!

Definition. *beachline* $\beta \equiv$ lower envelope of $(f_p^\ell)_{p \in P \cap \ell^+}$

Sweep?

Which part of the plane above ℓ is fixed by what we've seen?



Solution:

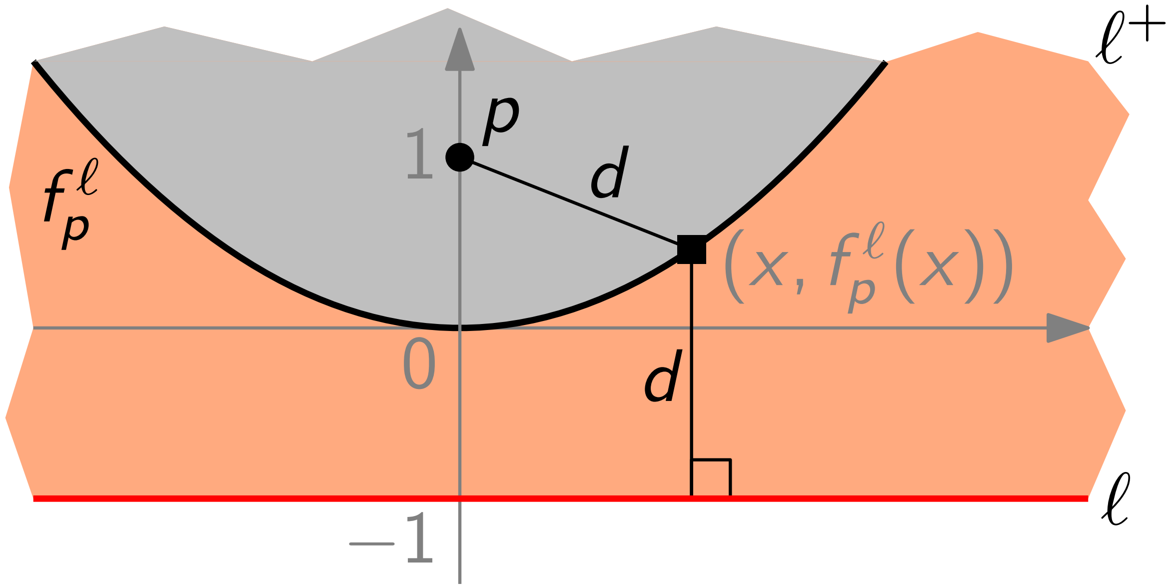
f_p^ℓ is the parabola with focus p and directrix ℓ.

Task: Compute f_p^ℓ for $p = (0, 1)$ and $\ell\colon y = -1$!

Definition. *beachline* $\beta \equiv$ lower envelope of $(f_p^\ell)_{p \in P \cap \ell^+}$

Sweep?

Which part of the plane above ℓ is fixed by what we've seen?

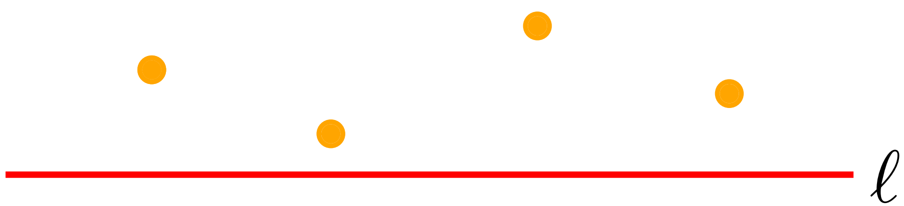


Solution:

f_p^ℓ is the parabola with focus p and directrix ℓ.

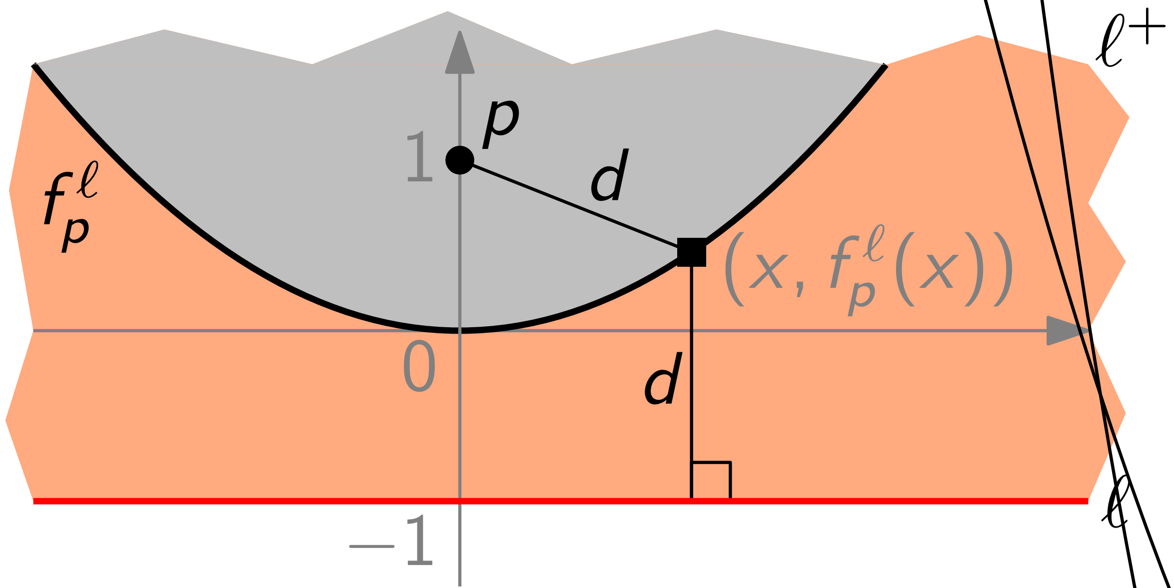
Task: Compute f_p^ℓ for $p = (0, 1)$ and $\ell\colon y = -1$!

Definition. *beachline* $\beta \equiv$ lower envelope of $(f_p^\ell)_{p \in P \cap \ell^+}$



Sweep?

Which part of the plane above ℓ is fixed by what we've seen?

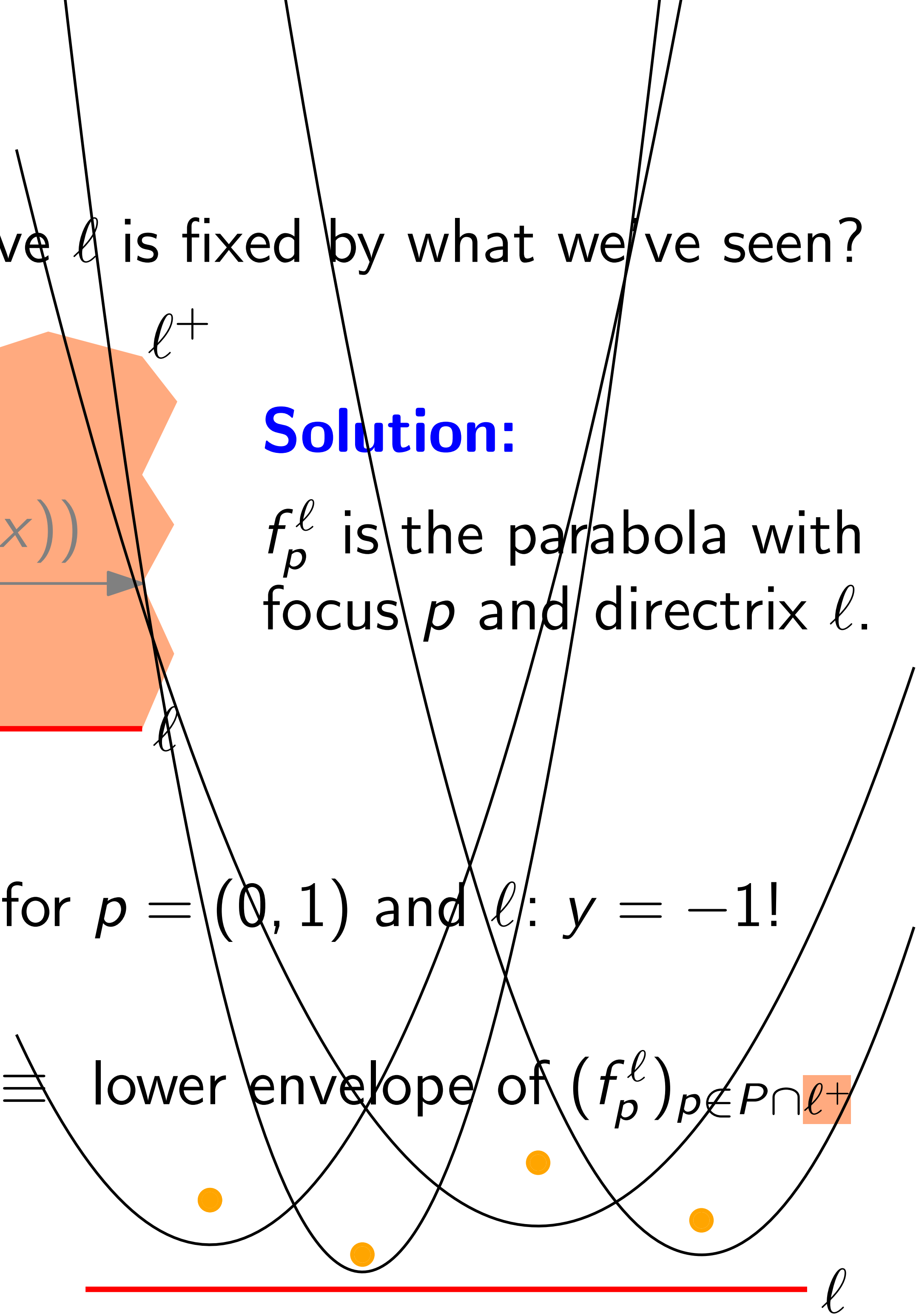


Solution:

f_p^ℓ is the parabola with focus p and directrix ℓ.

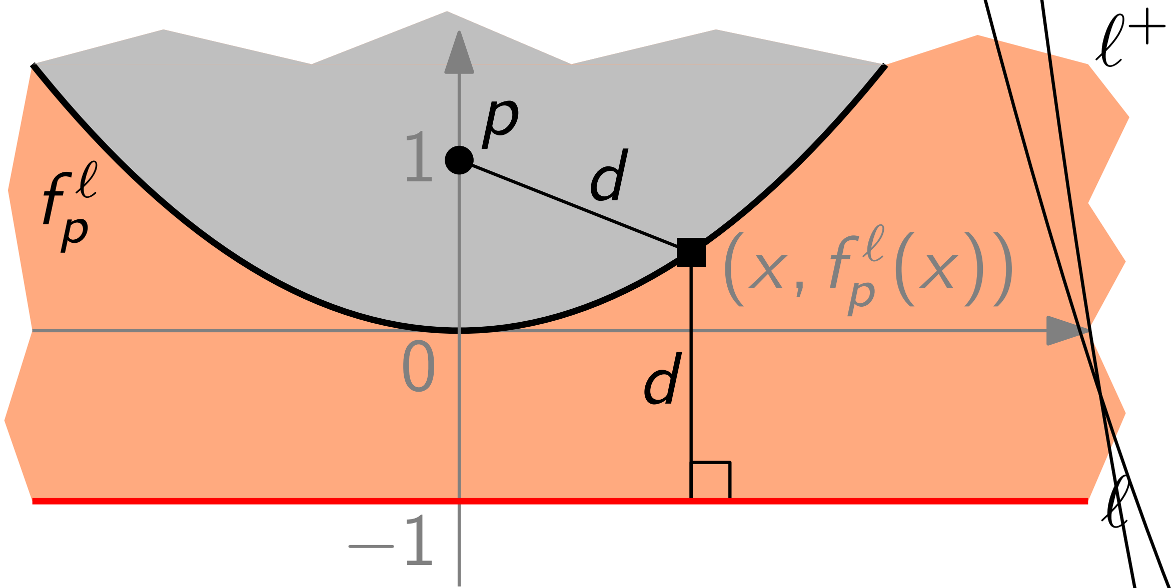
Task: Compute f_p^ℓ for $p = (0, 1)$ and ℓ: $y = -1$!

Definition. *beachline* $\beta \equiv$ lower envelope of $(f_p^\ell)_{p \in P \cap \ell^+}$



Sweep?

Which part of the plane above ℓ is fixed by what we've seen?

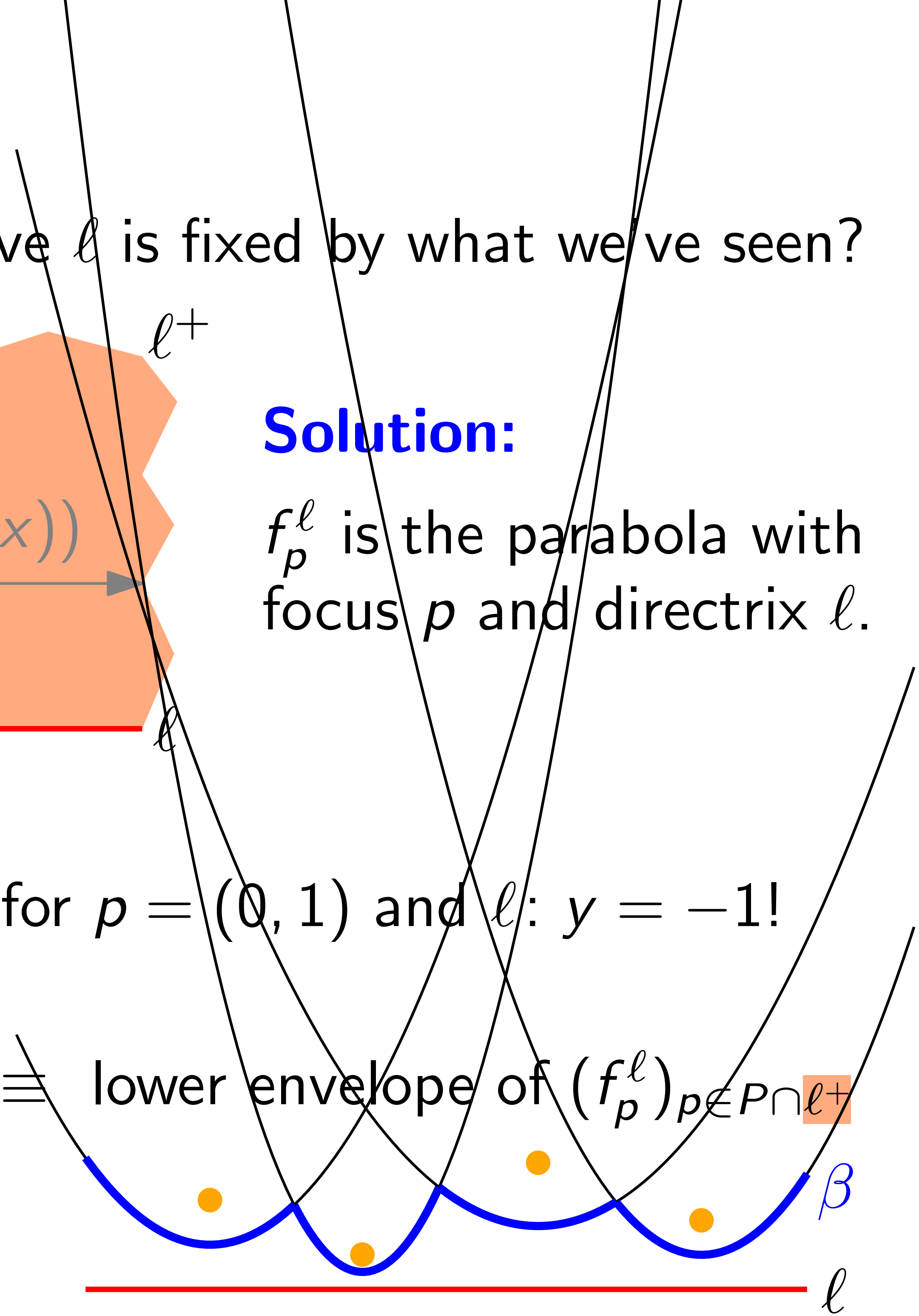


Solution:

f_p^ℓ is the parabola with focus p and directrix ℓ.

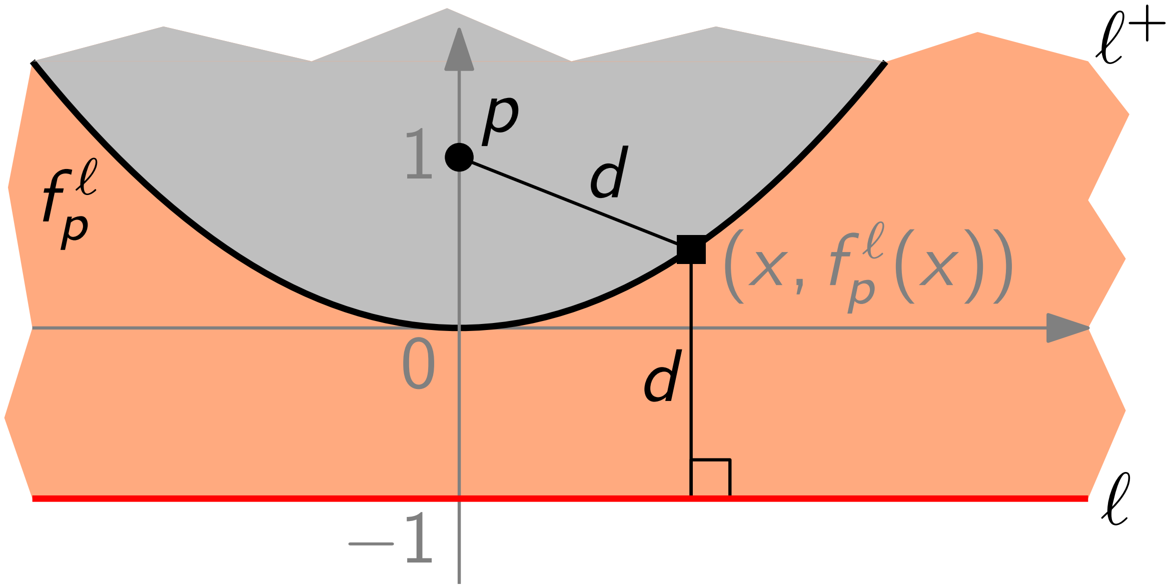
Task: Compute f_p^ℓ for $p = (0, 1)$ and ℓ: $y = -1$!

Definition. *beachline* $\beta \equiv$ lower envelope of $(f_p^\ell)_{p \in P \cap \ell^+}$



Sweep?

Which part of the plane above ℓ is fixed by what we've seen?

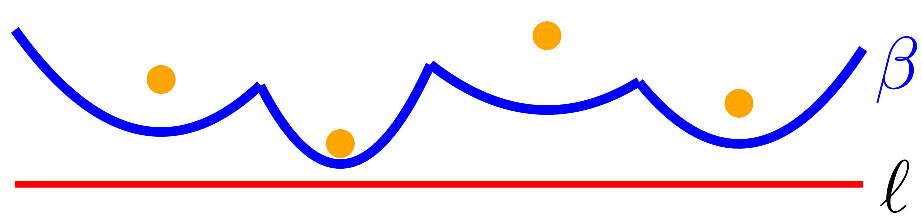


Solution:

f_p^ℓ is the parabola with focus p and directrix ℓ.

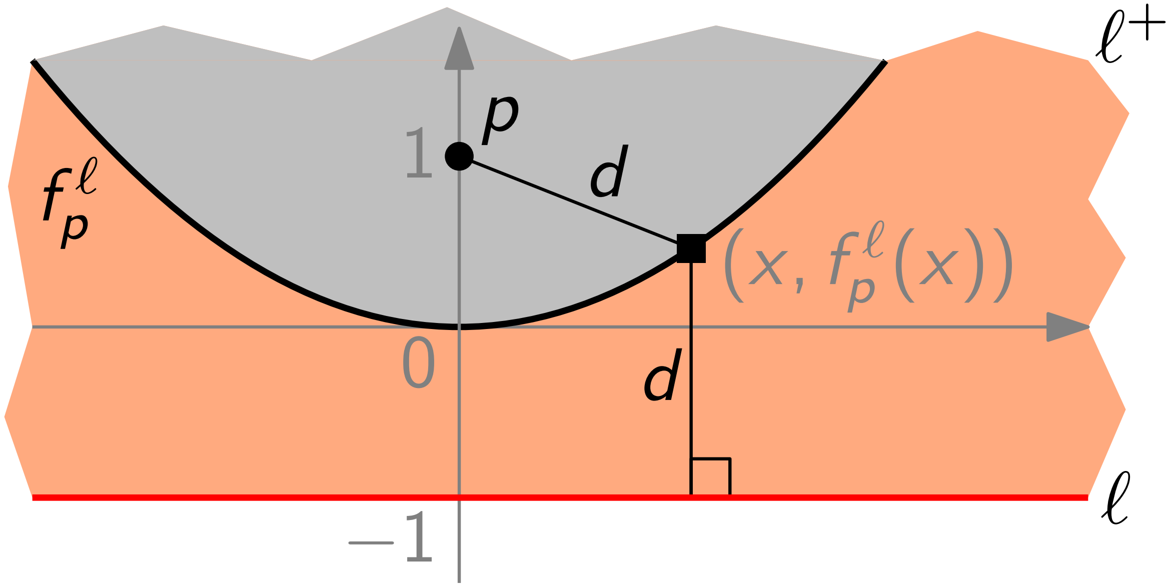
Task: Compute f_p^ℓ for $p = (0, 1)$ and $\ell\colon y = -1$!

Definition. *beachline* $\beta \equiv$ lower envelope of $(f_p^\ell)_{p \in P \cap \ell^+}$



Sweep?

Which part of the plane above ℓ is fixed by what we've seen?



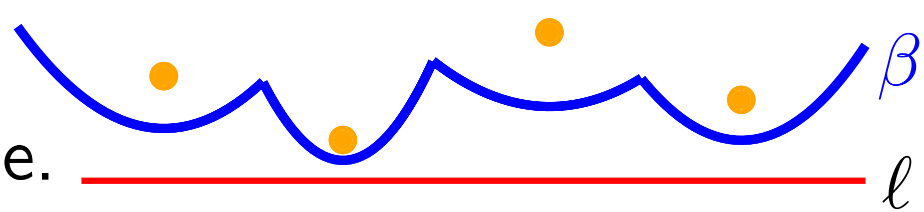
Solution:

f_p^ℓ is the parabola with focus p and directrix ℓ.

Task: Compute f_p^ℓ for $p = (0, 1)$ and ℓ: $y = -1$!

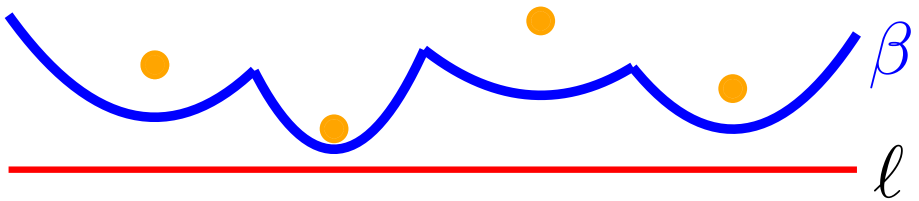
Definition. *beachline* $\beta \equiv$ lower envelope of $(f_p^\ell)_{p \in P \cap \ell^+}$

Observation. β is x-monotone.



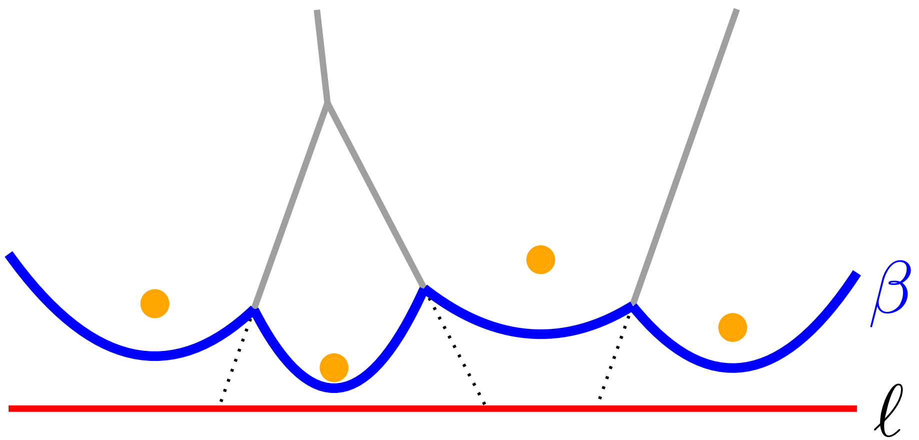
The beachline β

Question: What does β have to do with Vor(P)?



The beachline β

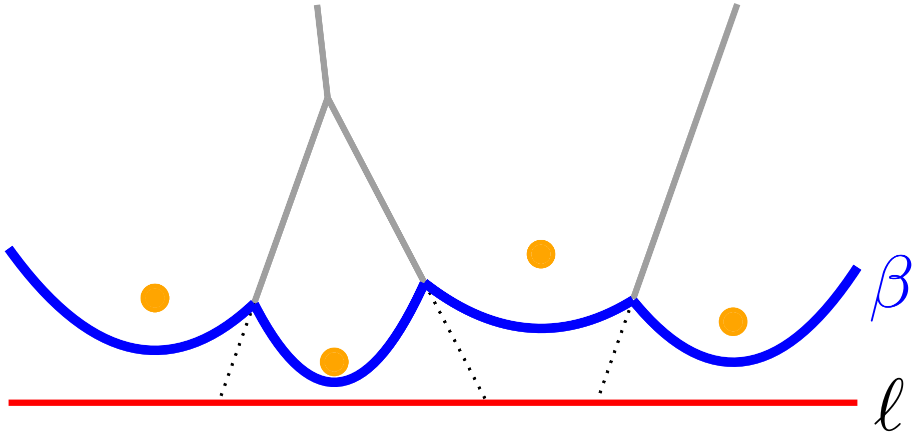
Question: What does β have to do with Vor(P)?



The beachline β

Question: What does β have to do with Vor(P)?

Answer: "Breakpoints" of β trace out the Voronoi edges!



The beachline β

Question: What does β have to do with Vor(P)?

Answer: "Breakpoints" of β trace out the Voronoi edges!

Lemma. New arcs on β only appear through *site events*

The beachline β

Question: What does β have to do with Vor(P)?

Answer: "Breakpoints" of β trace out the Voronoi edges!

Lemma. New arcs on β only appear through *site events*, that is, whenever ℓ hits a new site.

The beachline β

Question: What does β have to do with Vor(P)?

Answer: "Breakpoints" of β trace out the Voronoi edges!

Lemma. New arcs on β only appear through *site events*, that is, whenever ℓ hits a new site.

Corollary. β consists of at most $2n - 1$ arcs.

The beachline β

Question: What does β have to do with Vor(P)?

Answer: "Breakpoints" of β trace out the Voronoi edges!

Lemma. New arcs on β only appear through *site events*, that is, whenever ℓ hits a new site.

Corollary. β consists of at most $2n - 1$ arcs.

Definition. *Circle event:* ℓ reaches lowest pt of a circle through three sites above ℓ whose arcs are consecutive on β.

The beachline β

Question: What does β have to do with Vor(P)?

Answer: "Breakpoints" of β trace out the Voronoi edges!

Lemma. New arcs on β only appear through *site events*, that is, whenever ℓ hits a new site.

Corollary. β consists of at most $2n - 1$ arcs.

Definition. *Circle event:* ℓ reaches lowest pt of a circle through three sites above ℓ whose arcs are consecutive on β.

Lemma. Arcs disappear from β only at circle events.

The beachline β

Question: What does β have to do with Vor(P)?

Answer: "Breakpoints" of β trace out the Voronoi edges!

Lemma. New arcs on β only appear through *site events*, that is, whenever ℓ hits a new site.

Corollary. β consists of at most $2n - 1$ arcs.

Definition. *Circle event:* ℓ reaches lowest pt of a circle through three sites above ℓ whose arcs are consecutive on β.

Lemma. Arcs disappear from β only at circle events.

Lemma. The Voronoi vtc correspond 1:1 to circle events.

Fortune's Sweep

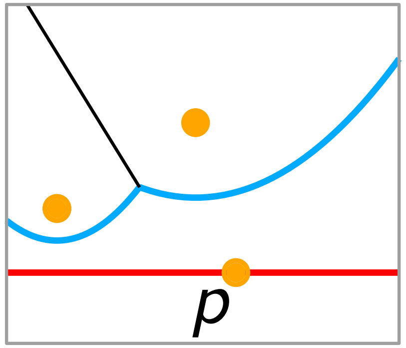
VoronoiDiagram($P \subset \mathbb{R}^2$)
$\mathcal{Q} \leftarrow$ new PriorityQueue(P) // site events sorted acc. y-coord.
$\mathcal{T} \leftarrow$ new BalancedBinarySearchTree() // sweep status (β)
$\mathcal{D} \leftarrow$ new DCEL() // to-be Vor(P)
while not $\mathcal{Q}$.empty() **do**

treat remaining internal nodes of $\mathcal{T}$ ($\equiv$ unbnd. edges of Vor(P))
return $\mathcal{D}$

Fortune's Sweep

```
VoronoiDiagram(P ⊂ ℝ²)
  𝒬 ← new PriorityQueue(P)   // site events sorted acc. y-coord.
  𝒯 ← new BalancedBinarySearchTree()   // sweep status (β)
  𝒟 ← new DCEL()   // to-be Vor(P)
  while not 𝒬.empty() do
      p ← 𝒬.ExtractMax()
      if p site event then
          HandleSiteEvent(p)
      else
          α ← arc on β that will disappear
          HandleCircleEvent(α)
  treat remaining internal nodes of 𝒯 (≡ unbnd. edges of Vor(P))
  return 𝒟
```

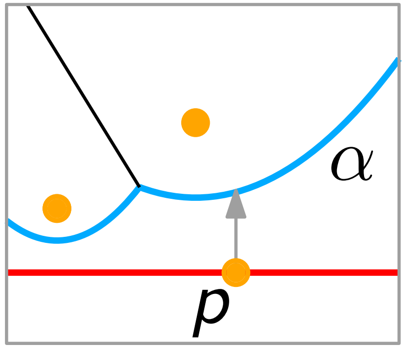
Handling Events



HandleSiteEvent(point p)

HandleCircleEvent(arc α)

Handling Events

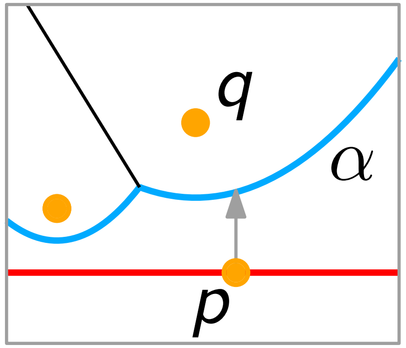


HandleSiteEvent(point p)

- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.

HandleCircleEvent(arc α)

Handling Events

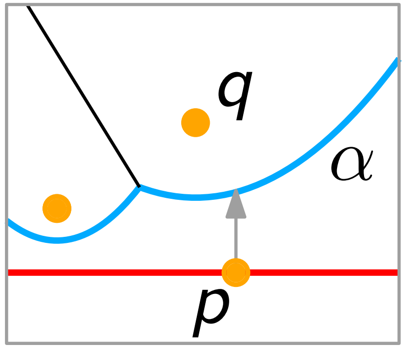


HandleSiteEvent(point p)

- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.

HandleCircleEvent(arc α)

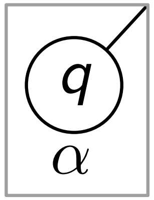
Handling Events



HandleSiteEvent(point p)

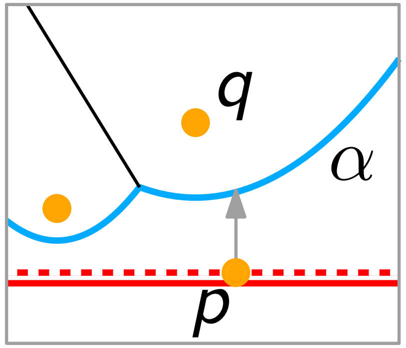
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.

In $\mathcal{T}$:



HandleCircleEvent(arc α)

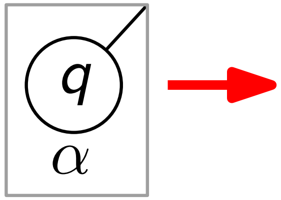
Handling Events



HandleSiteEvent(point p)

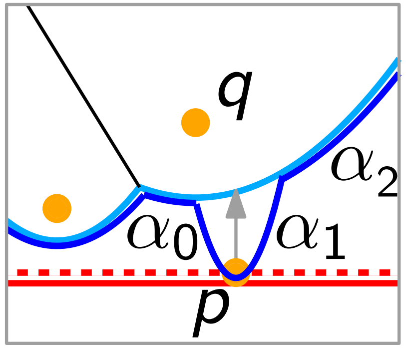
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.

In $\mathcal{T}$:



HandleCircleEvent(arc α)

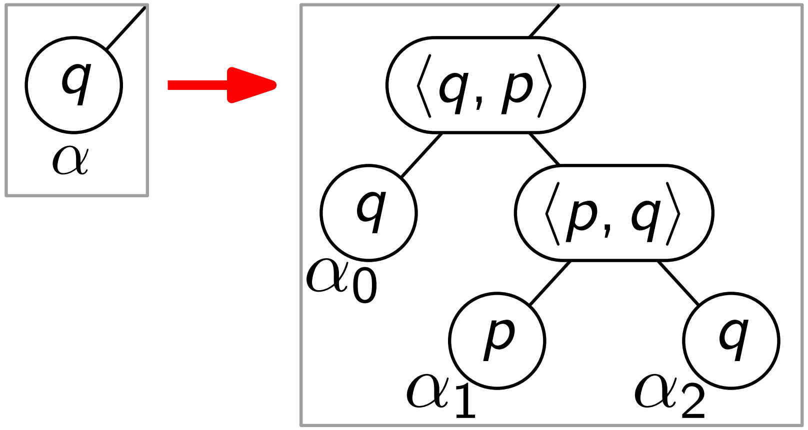
Handling Events



HandleSiteEvent(point p)

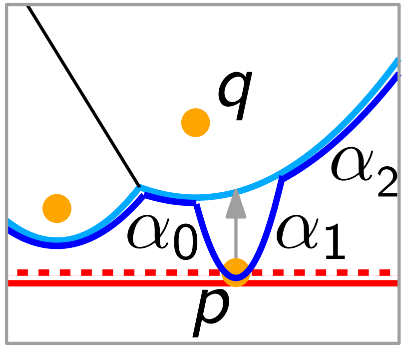
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.

In $\mathcal{T}$:



HandleCircleEvent(arc α)

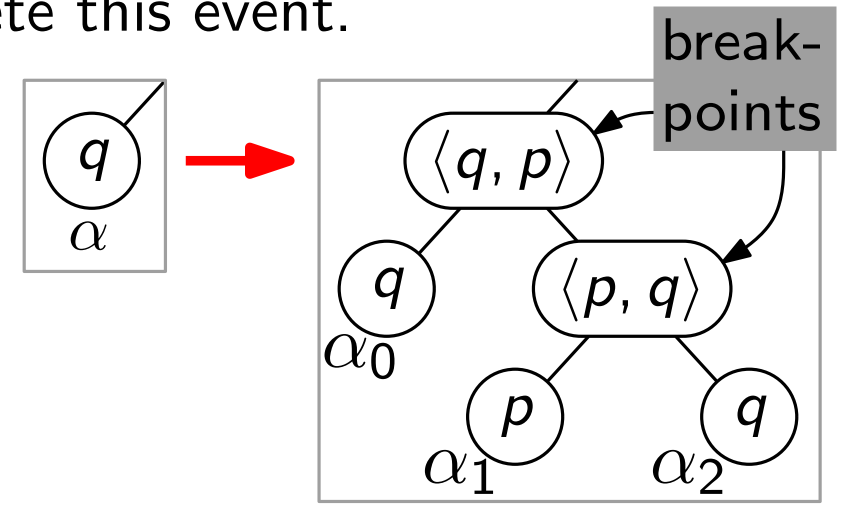
Handling Events



HandleSiteEvent(point *p*)

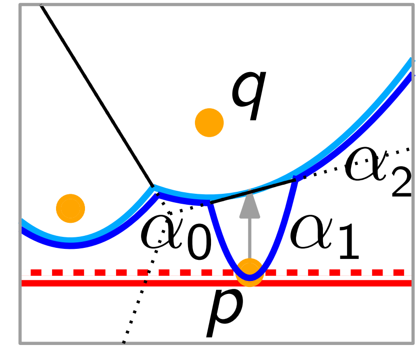
- Search in $\mathcal{T}$ for the arc α vertically above *p*.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of *p*.

In $\mathcal{T}$:



HandleCircleEvent(arc α)

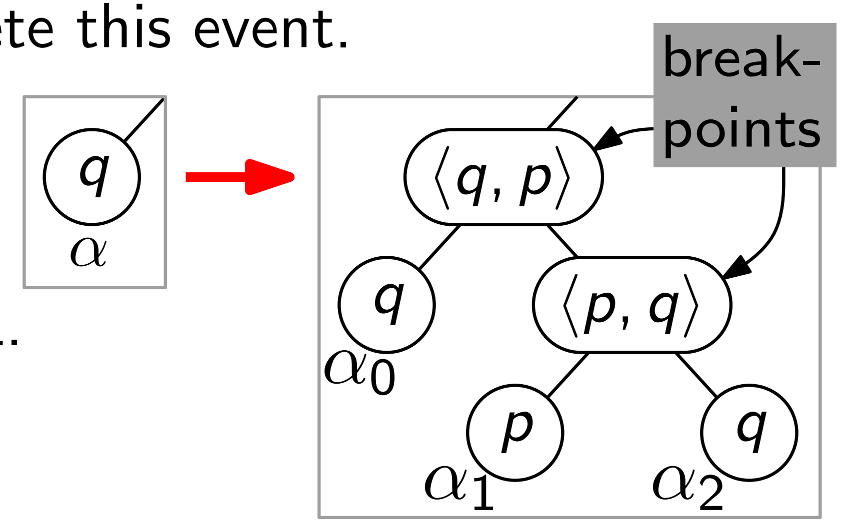
Handling Events



HandleSiteEvent(point p)

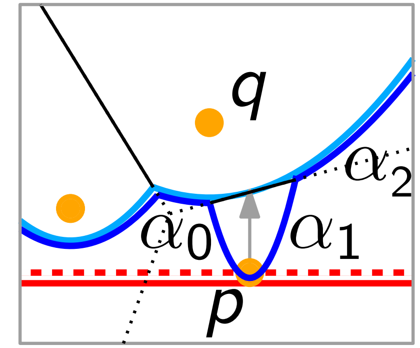
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p\rangle$ and $\langle p, q\rangle$ to DCEL.

In $\mathcal{T}$:



HandleCircleEvent(arc α)

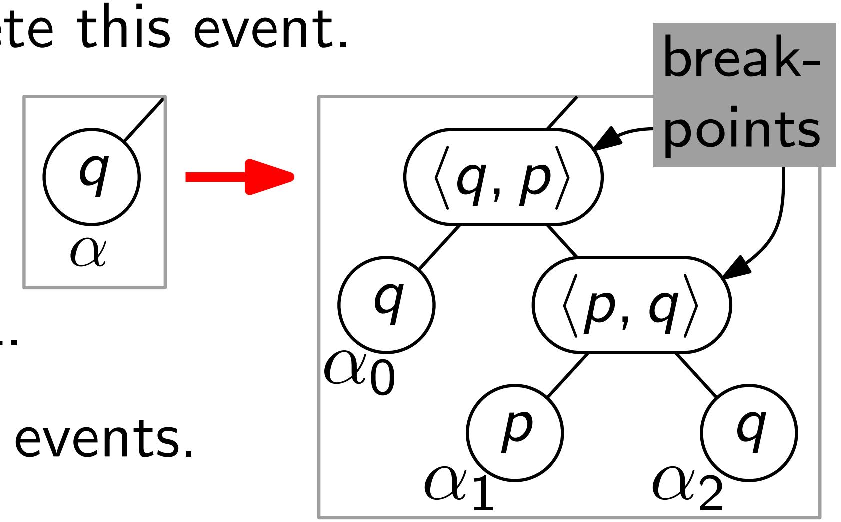
Handling Events



HandleSiteEvent(point p)

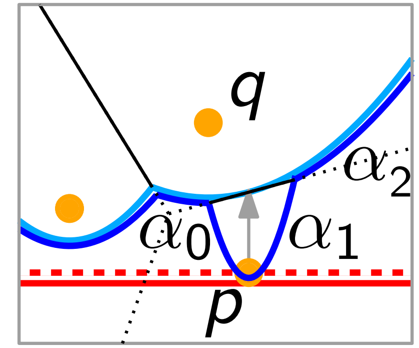
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

In $\mathcal{T}$:



HandleCircleEvent(arc α)

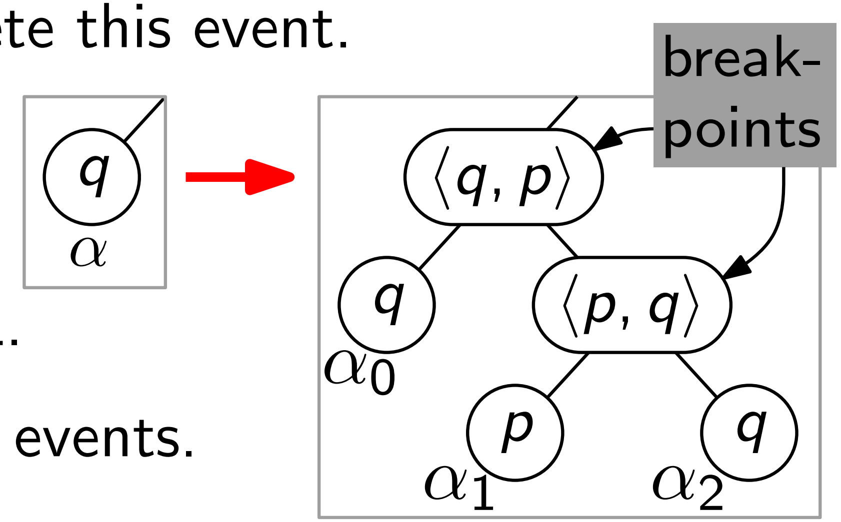
Handling Events



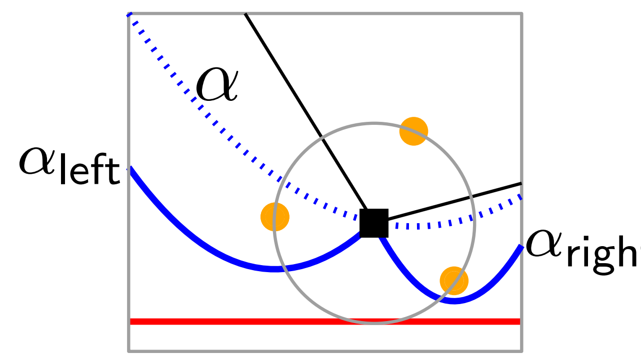
HandleSiteEvent(point p)

- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

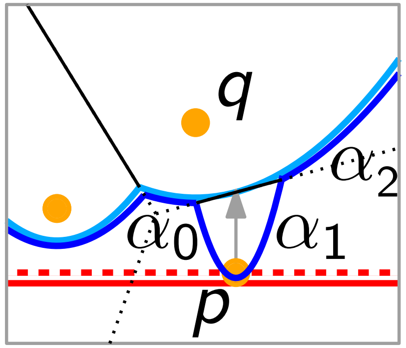
In $\mathcal{T}$:



HandleCircleEvent(arc α)



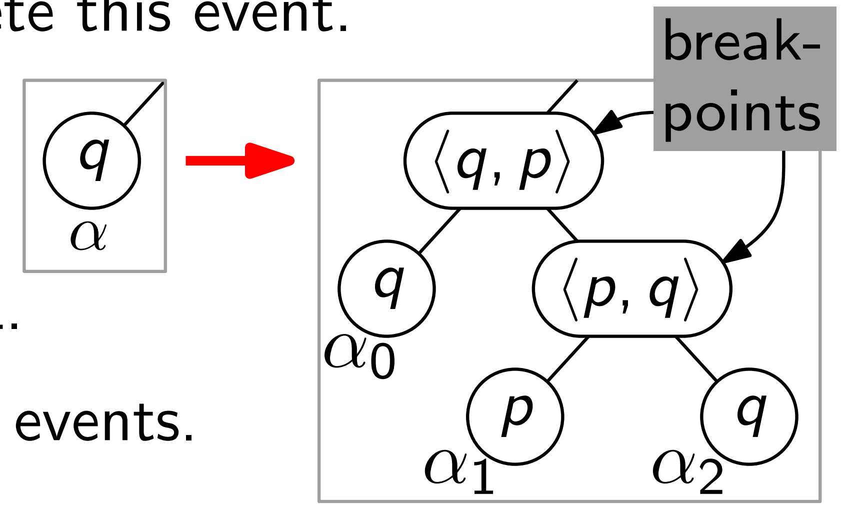
Handling Events



HandleSiteEvent(point p)

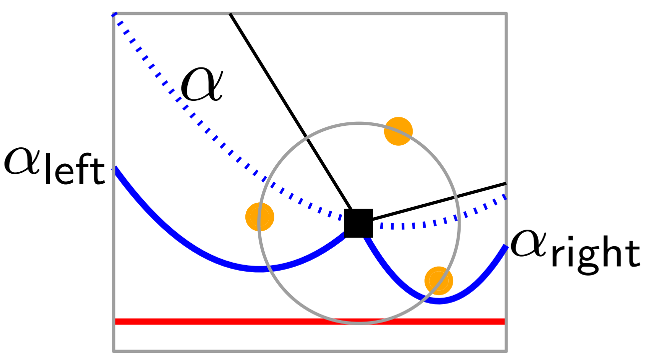
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

In $\mathcal{T}$:

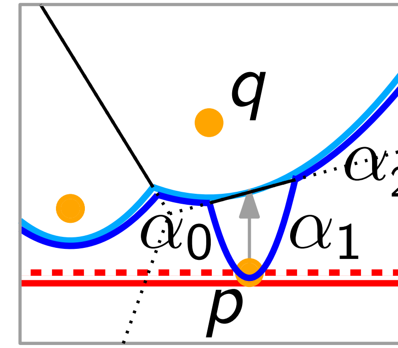


HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); update breakpts



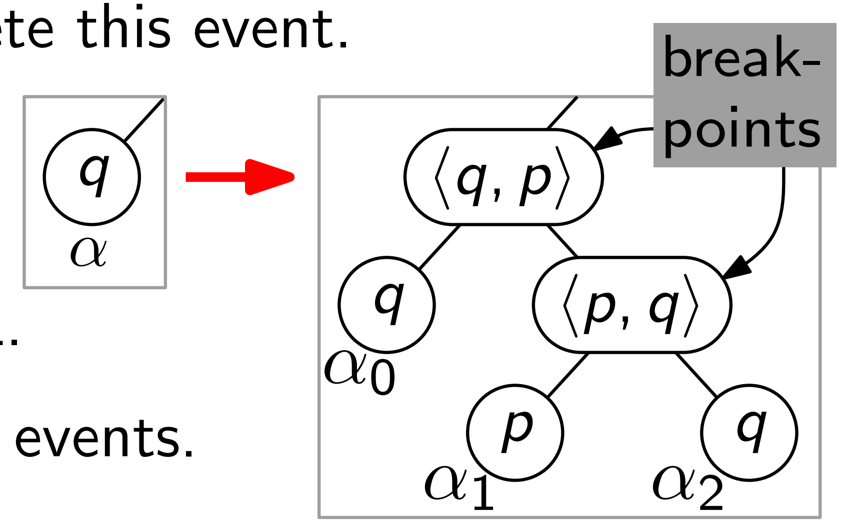
Handling Events



HandleSiteEvent(point p)

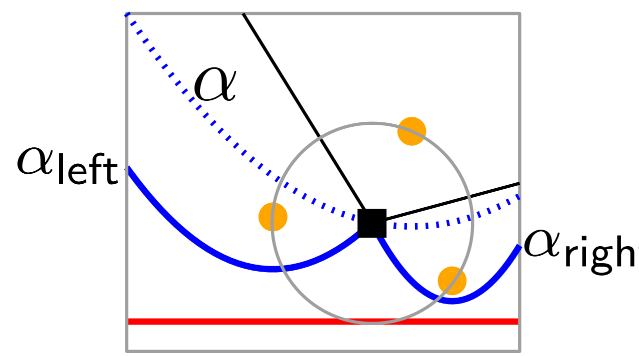
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p\rangle$ and $\langle p, q\rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1\rangle$ and $\langle \alpha_1, \alpha_2, \cdot\rangle$ for circle events.

In $\mathcal{T}$:

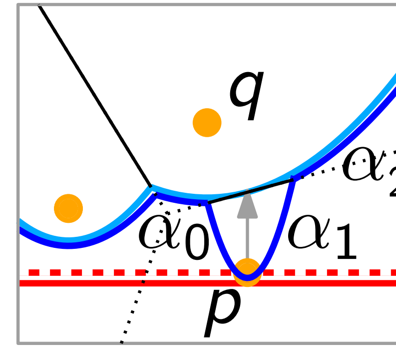


HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); update breakpts
- Delete all circle events involving α from $\mathcal{Q}$.



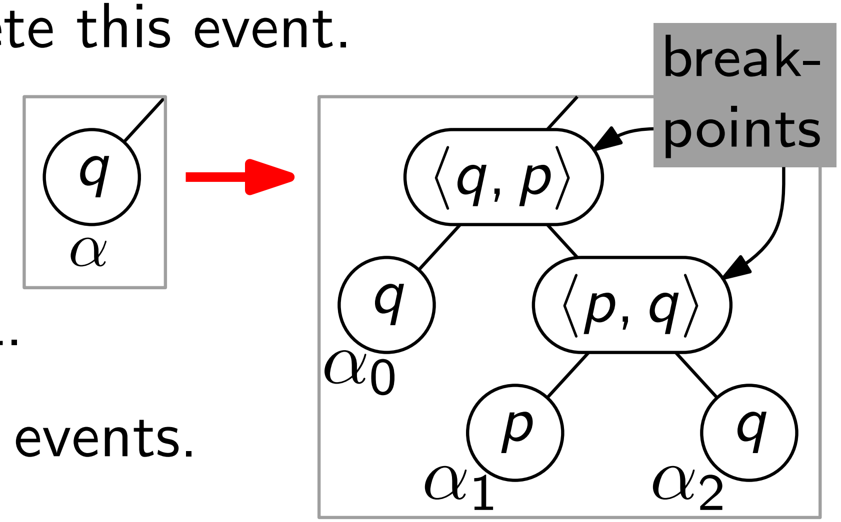
Handling Events



HandleSiteEvent(point p)

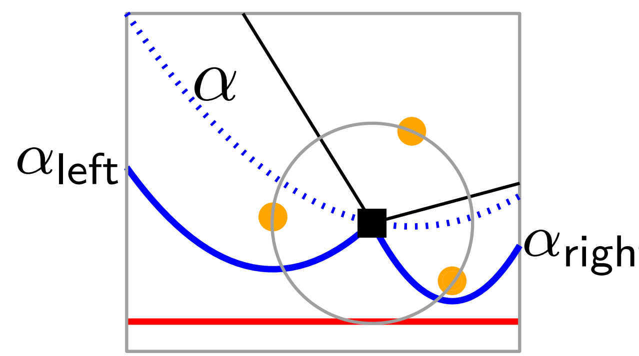
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.

In $\mathcal{T}$:



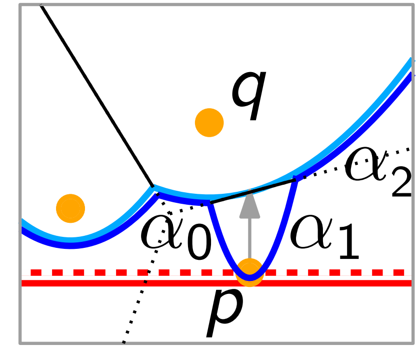
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

HandleCircleEvent(arc α)



- $\mathcal{T}$.delete(α); update breakpts
- Delete all circle events involving α from $\mathcal{Q}$.
- Add Vor-vtx $\alpha_{\text{left}} \cap \alpha_{\text{right}}$ and Vor-edge $\langle \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ to DCEL.

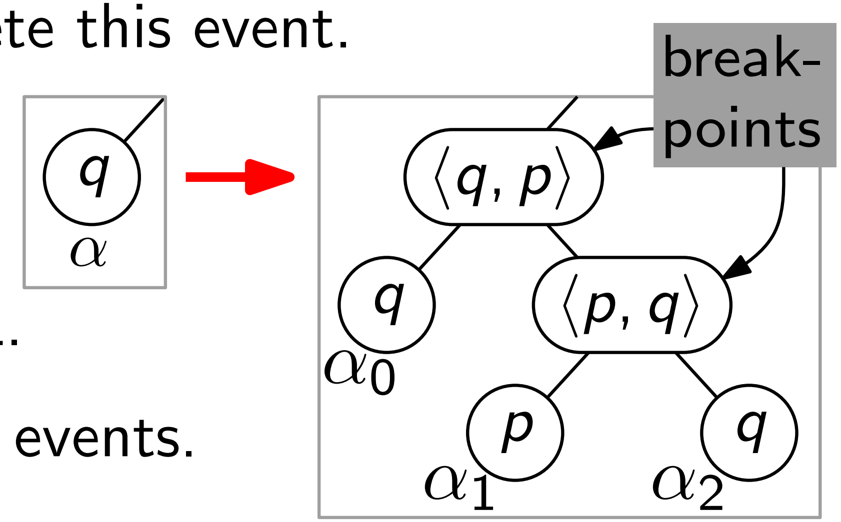
Handling Events



HandleSiteEvent(point p)

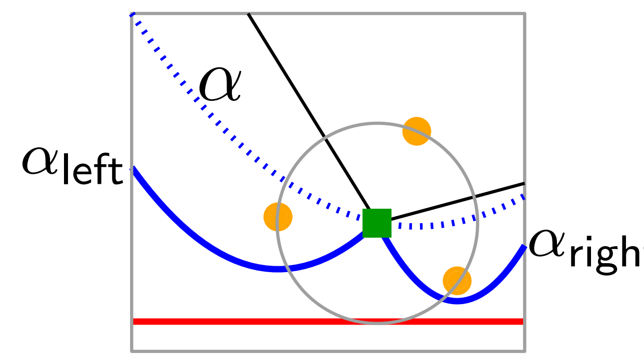
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.

In $\mathcal{T}$:



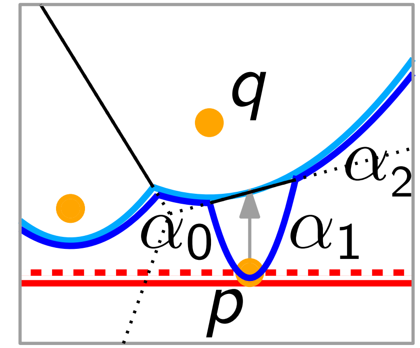
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

HandleCircleEvent(arc α)



- $\mathcal{T}$.delete(α); update breakpts
- Delete all circle events involving α from $\mathcal{Q}$.
- Add Vor-vtx $\alpha_{\text{left}} \cap \alpha_{\text{right}}$ and Vor-edge $\langle \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ to DCEL.

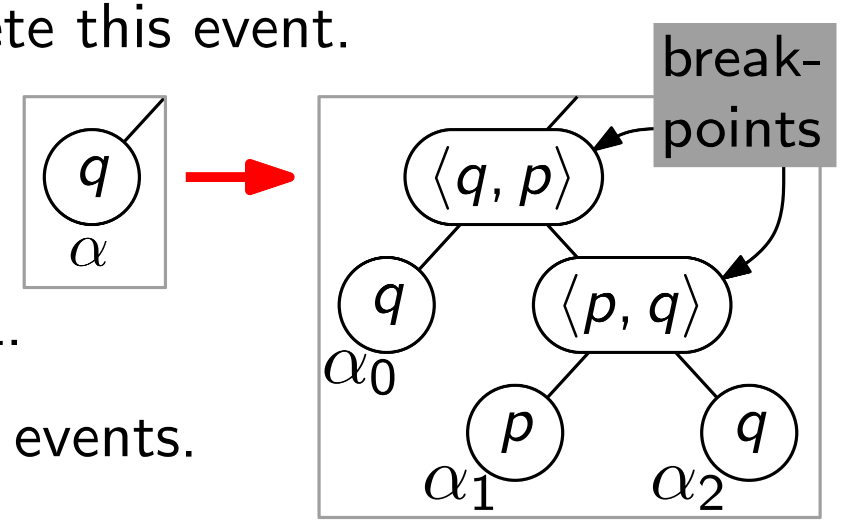
Handling Events



HandleSiteEvent(point p)

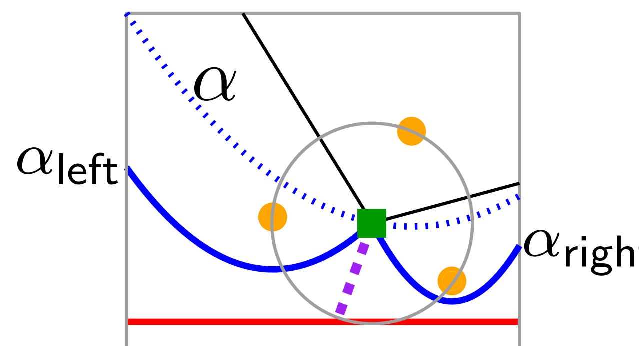
- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

In $\mathcal{T}$:

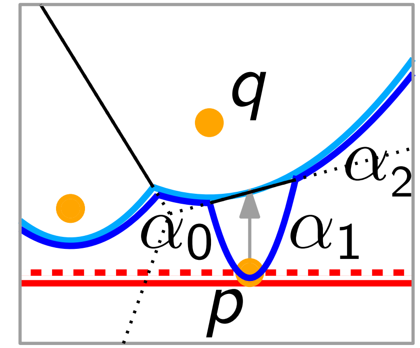


HandleCircleEvent(arc α)

- $\mathcal{T}$.delete(α); update breakpts
- Delete all circle events involving α from $\mathcal{Q}$.
- Add Vor-vtx $\alpha_{\text{left}} \cap \alpha_{\text{right}}$ and Vor-edge $\langle \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ to DCEL.



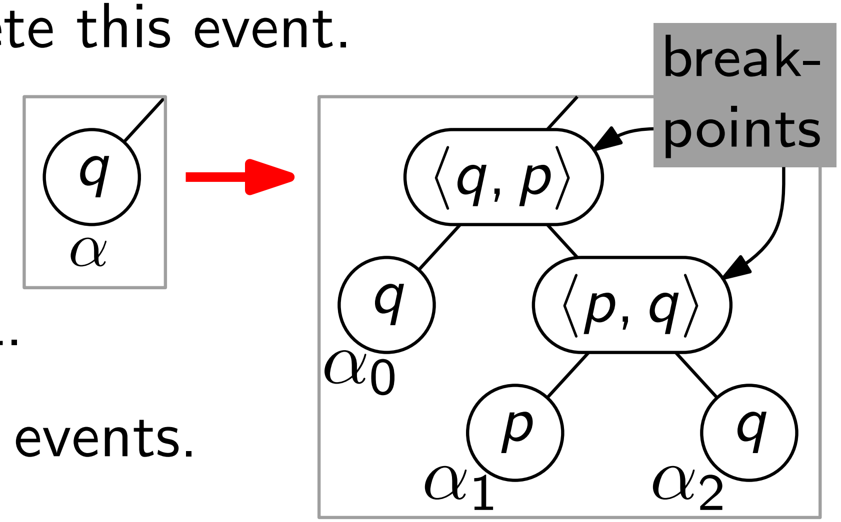
Handling Events



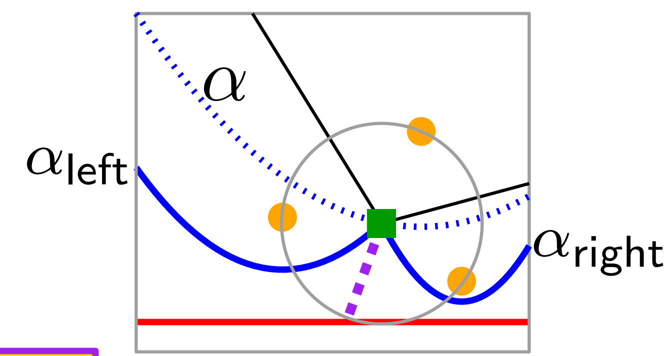
HandleSiteEvent(point p)

- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

In $\mathcal{T}$:

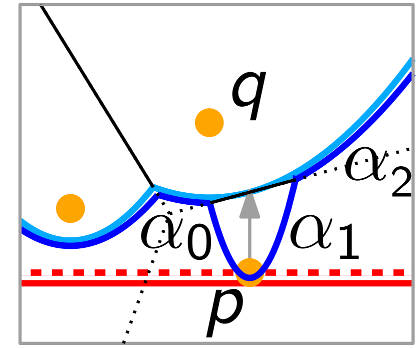


HandleCircleEvent(arc α)



- $\mathcal{T}$.delete(α); update breakpts
- Delete all circle events involving α from $\mathcal{Q}$.
- Add Vor-vtx $\alpha_{\text{left}} \cap \alpha_{\text{right}}$ and Vor-edge $\langle \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ and $\langle \alpha_{\text{left}}, \alpha_{\text{right}}, \cdot \rangle$ for circle events.

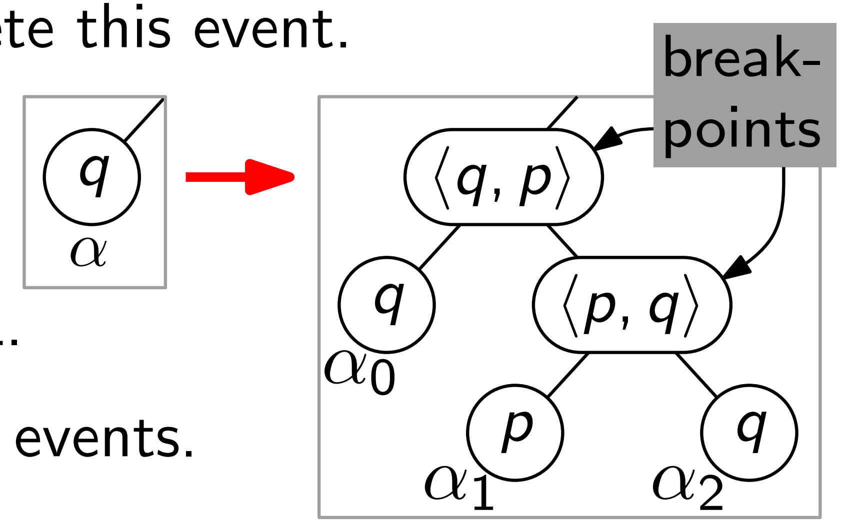
Handling Events



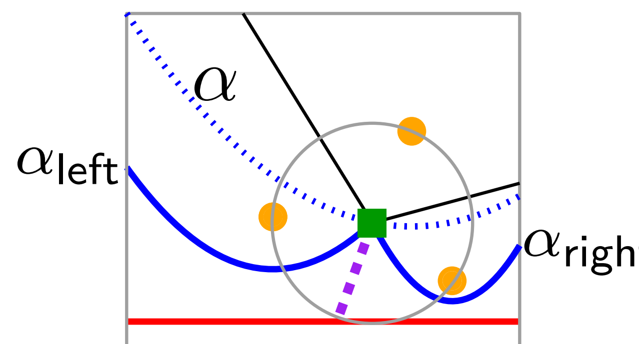
HandleSiteEvent(point p)

- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

In $\mathcal{T}$:



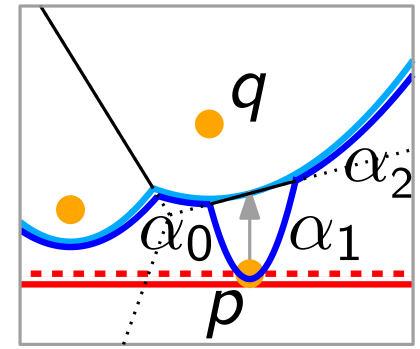
HandleCircleEvent(arc α)



- $\mathcal{T}$.delete(α); update breakpts
- Delete all circle events involving α from $\mathcal{Q}$.
- Add Vor-vtx $\alpha_{\text{left}} \cap \alpha_{\text{right}}$ and Vor-edge $\langle \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ and $\langle \alpha_{\text{left}}, \alpha_{\text{right}}, \cdot \rangle$ for circle events.

Running time?

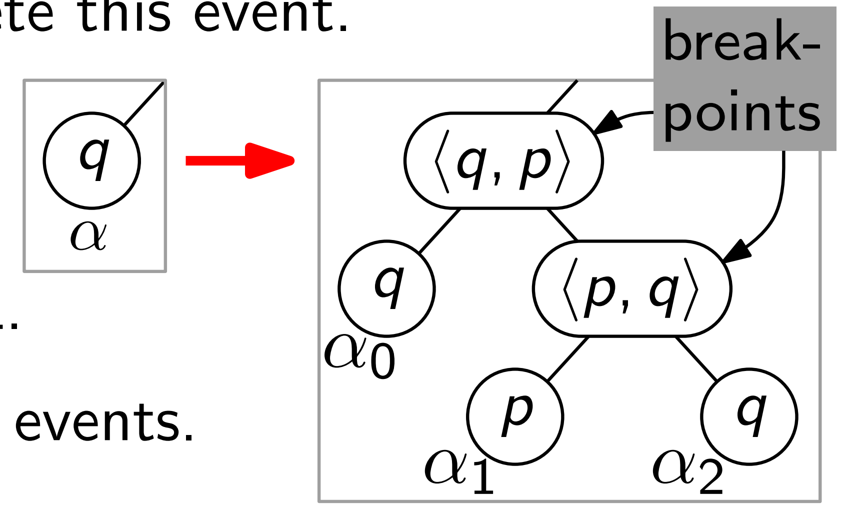
Handling Events



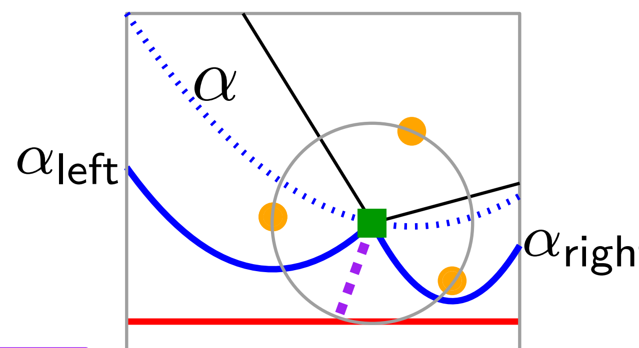
HandleSiteEvent(point p)

- Search in $\mathcal{T}$ for the arc α vertically above p.
  If α has pointer to circle event in $\mathcal{Q}$, delete this event.
- Split α into α_0 and α_2.
  Let α_1 be the new arc of p.
- Add Vor-edges $\langle q, p \rangle$ and $\langle p, q \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_0, \alpha_1 \rangle$ and $\langle \alpha_1, \alpha_2, \cdot \rangle$ for circle events.

In $\mathcal{T}$:



HandleCircleEvent(arc α)



- $\mathcal{T}$.delete(α); update breakpts
- Delete all circle events involving α from $\mathcal{Q}$.
- Add Vor-vtx $\alpha_{\text{left}} \cap \alpha_{\text{right}}$ and Vor-edge $\langle \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ to DCEL.
- Check $\langle \cdot, \alpha_{\text{left}}, \alpha_{\text{right}} \rangle$ and $\langle \alpha_{\text{left}}, \alpha_{\text{right}}, \cdot \rangle$ for circle events.

Running time? $O(\log n)$ per event...

Running Time?

```
VoronoiDiagram(P ⊂ ℝ²)
 𝒬 ← new PriorityQueue(P)   // site events sorted acc. y-coord.
 𝒯 ← new BalancedBinarySearchTree()   // sweep status (β)
 𝒟 ← new DCEL()   // to-be Vor(P)
 while not 𝒬.empty() do
     p ← 𝒬.ExtractMax()
     if p site event then
         HandleSiteEvent(p)
     else
         α ← arc on β that will disappear
         HandleCircleEvent(α)
 treat remaining internal nodes of 𝒯 (≡ unbnd. edges of Vor(P))
 return 𝒟
```

Running Time?

```
VoronoiDiagram(P ⊂ ℝ²)
 Q ← new PriorityQueue(P)   // site events sorted acc. y-coord.
 T ← new BalancedBinarySearchTree()   // sweep status (β)
 D ← new DCEL()   // to-be Vor(P)
 while not Q.empty() do
     p ← Q.ExtractMax()
     if p site event then
         HandleSiteEvent(p)      exactly n such events
     else
         α ← arc on β that will disappear
         HandleCircleEvent(α)
 treat remaining internal nodes of T (≡ unbnd. edges of Vor(P))
 return D
```

Running Time?

VoronoiDiagram($P \subset \mathbb{R}^2$)

$\mathcal{Q} \leftarrow$ new PriorityQueue(P) // site events sorted acc. y-coord.
$\mathcal{T} \leftarrow$ new BalancedBinarySearchTree() // sweep status (β)
$\mathcal{D} \leftarrow$ new DCEL() // to-be Vor(P)
while not $\mathcal{Q}$.empty() **do**
    $p \leftarrow \mathcal{Q}$.ExtractMax()
    if p site event **then**
        HandleSiteEvent(p)   exactly n such events
    else
        $\alpha \leftarrow$ arc on β that will disappear
        HandleCircleEvent(α)   at most $2n - 5$ such events
treat remaining internal nodes of $\mathcal{T}$ ($\equiv$ unbnd. edges of Vor(P))
return $\mathcal{D}$

Summary

Theorem. Given a set P of n pts in the plane, Fortune's sweep computes $\mathrm{Vor}(P)$ in $O(n \log n)$ time and $O(n)$ space.

Summary

Theorem. Given a set P of n pts in the plane, Fortune's sweep computes $\mathrm{Vor}(P)$ in $O(n \log n)$ time and $O(n)$ space.

Steven Fortune
Bell Labs

Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Proc. 2nd Annual ACM Symposium on Computational Geometry.* Yorktown Heights, NY, pp. 313–322. 1986.