COMPUTING VORONOI DIAGRAMS

Vera Sacristán

Seminar: Geometric Algorithms (MIRI)
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Naive algorithm

Naive algorithm

Naive algorithm

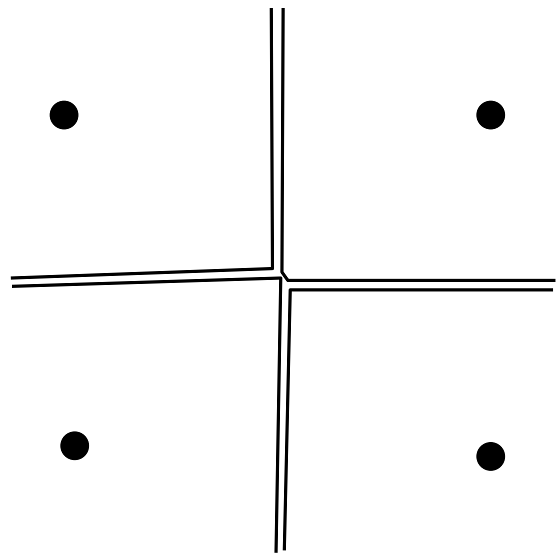
For each p_i, construct its Voronoi region $Vor(p_i) = \bigcap_{j \neq i} H_{ij}$.

Naive algorithm

For each p_i, construct its Voronoi region $Vor(p_i) = \bigcap_{j \neq i} H_{ij}$.

Inconvenients:

- It can cause inconsistency due to precision problems



Naive algorithm

For each p_i, construct its Voronoi region $Vor(p_i) = \bigcap_{j \neq i} H_{ij}$.

Inconvenients:

- It can cause inconsistency due to precision problems
- It does not produce immediate neighborhood information

Naive algorithm

For each p_i, construct its Voronoi region $Vor(p_i) = \bigcap_{j \neq i} H_{ij}$.

Inconvenients:

- It can cause inconsistency due to precision problems
- It does not produce immediate neighborhood information
- It runs in $O(n^2 \log n)$ time

Naive algorithm

For each p_i, construct its Voronoi region $Vor(p_i) = \bigcap_{j \neq i} H_{ij}$.

Inconvenients:

- It can cause inconsistency due to precision problems
- It does not produce immediate neighborhood information
- It runs in $O(n^2 \log n)$ time

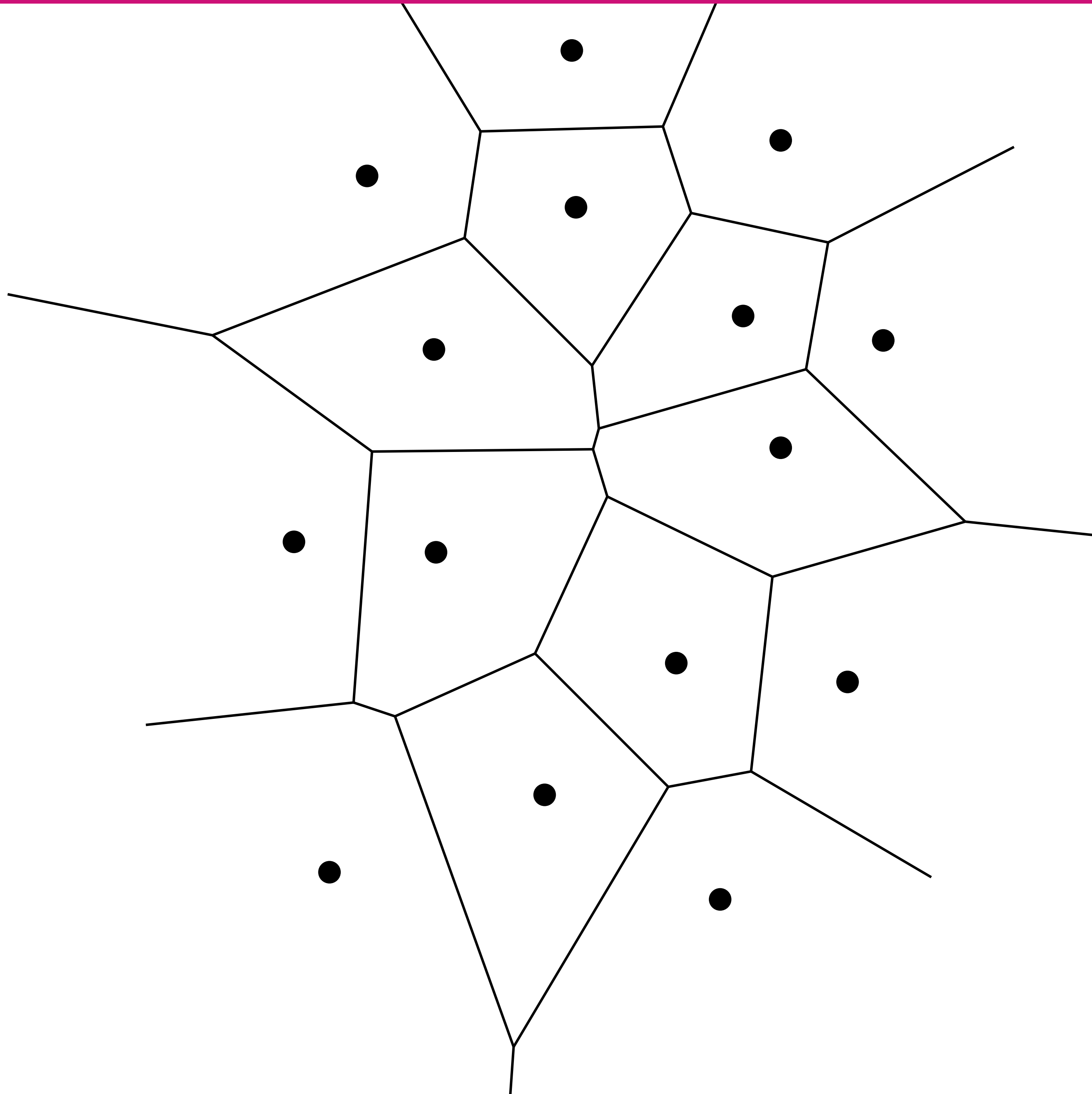
The fact that each Voronoi region, $Vor(p_i)$, is built in optimal $\Theta(n \log n)$ time does not imply that the construction of the entire diagram, $Vor(P)$, requires $\Omega(n^2 \log n)$ time, as we will see.

incremental algorithm

Incremental algorithm

Incremental algorithm

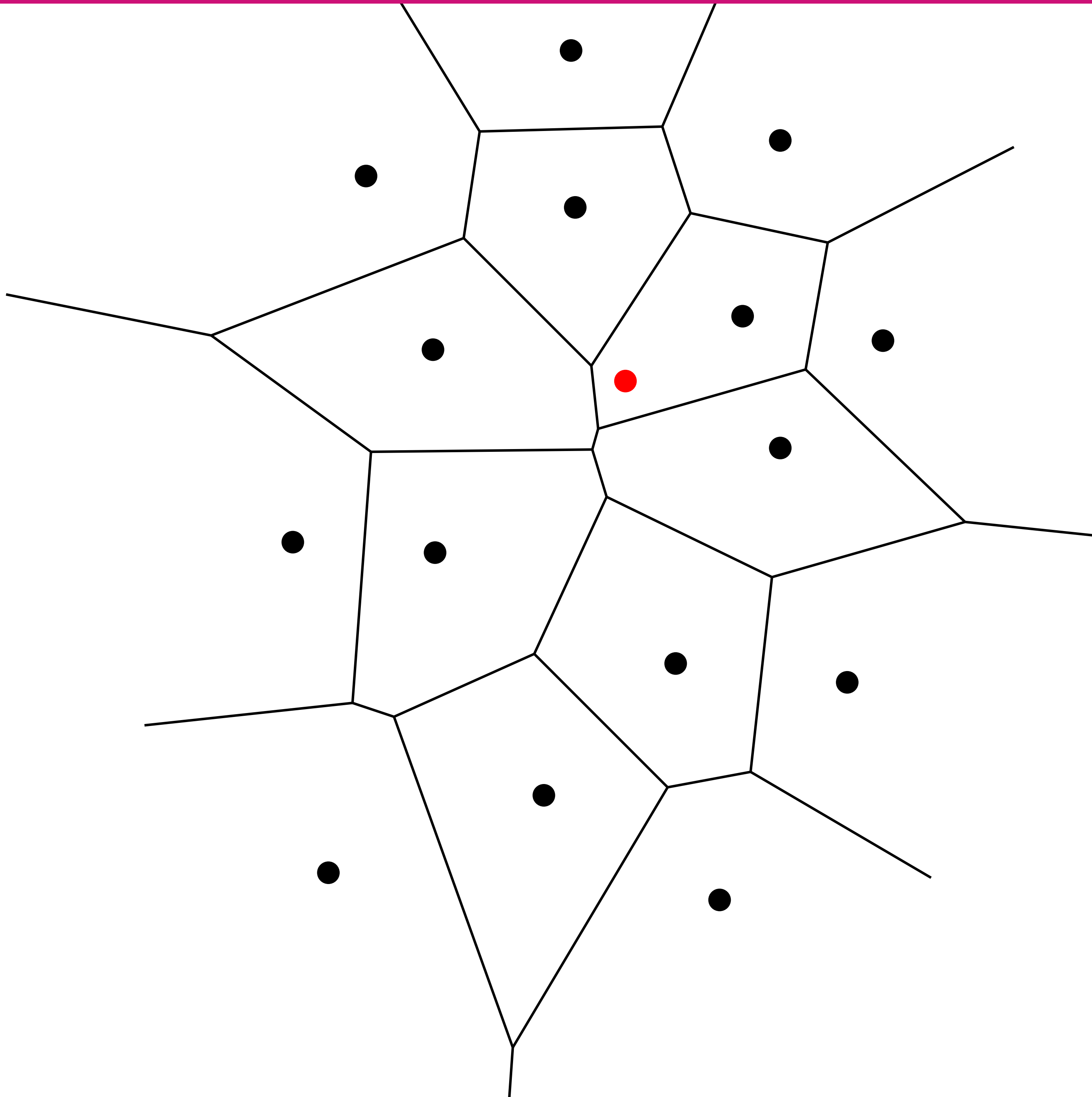
Starting with the Voronoi diagram of $\{p_1, ..., p_i\}$...



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

... add point p_{i+1}

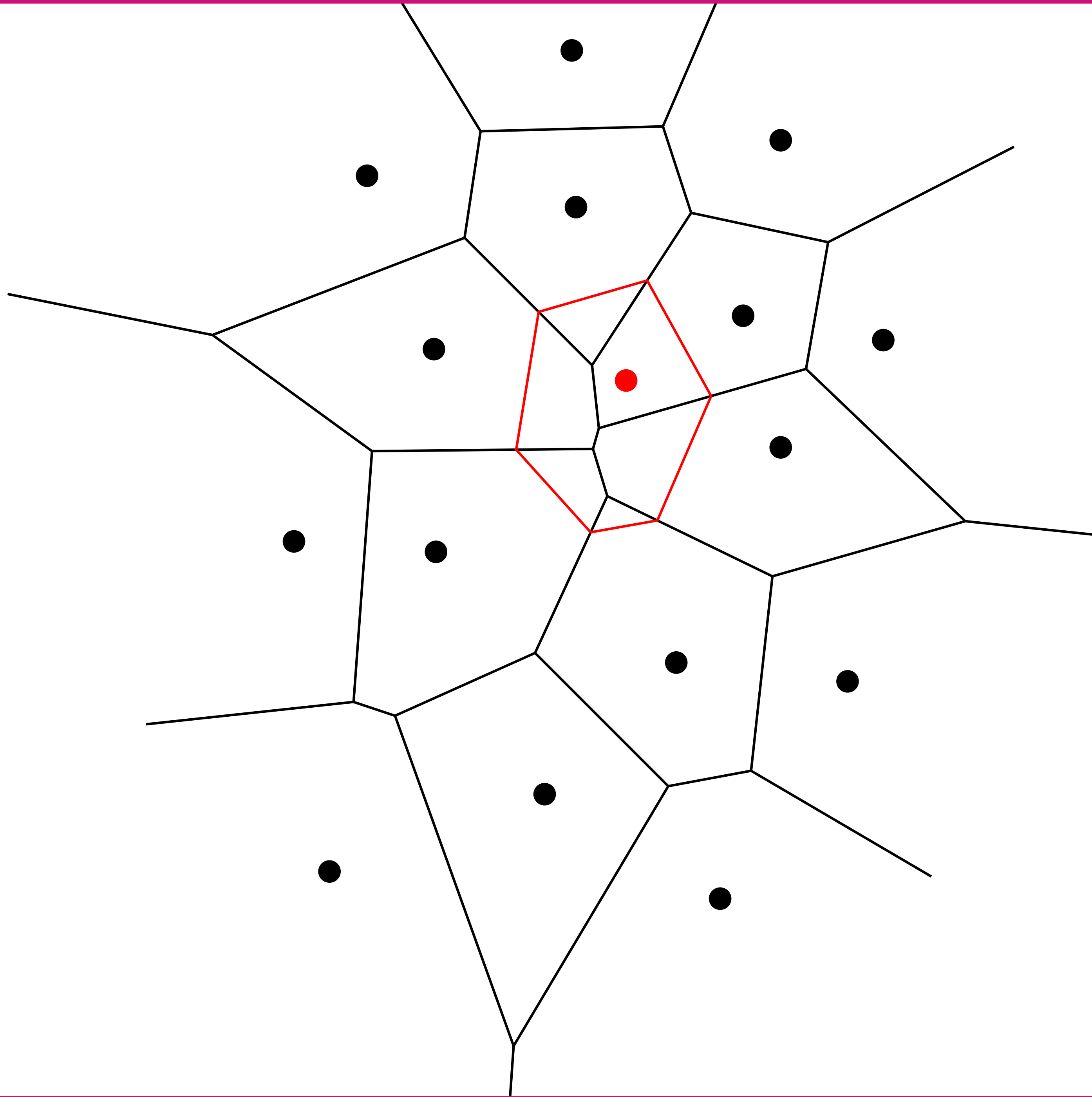


Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

... add point p_{i+1}

... compute its region



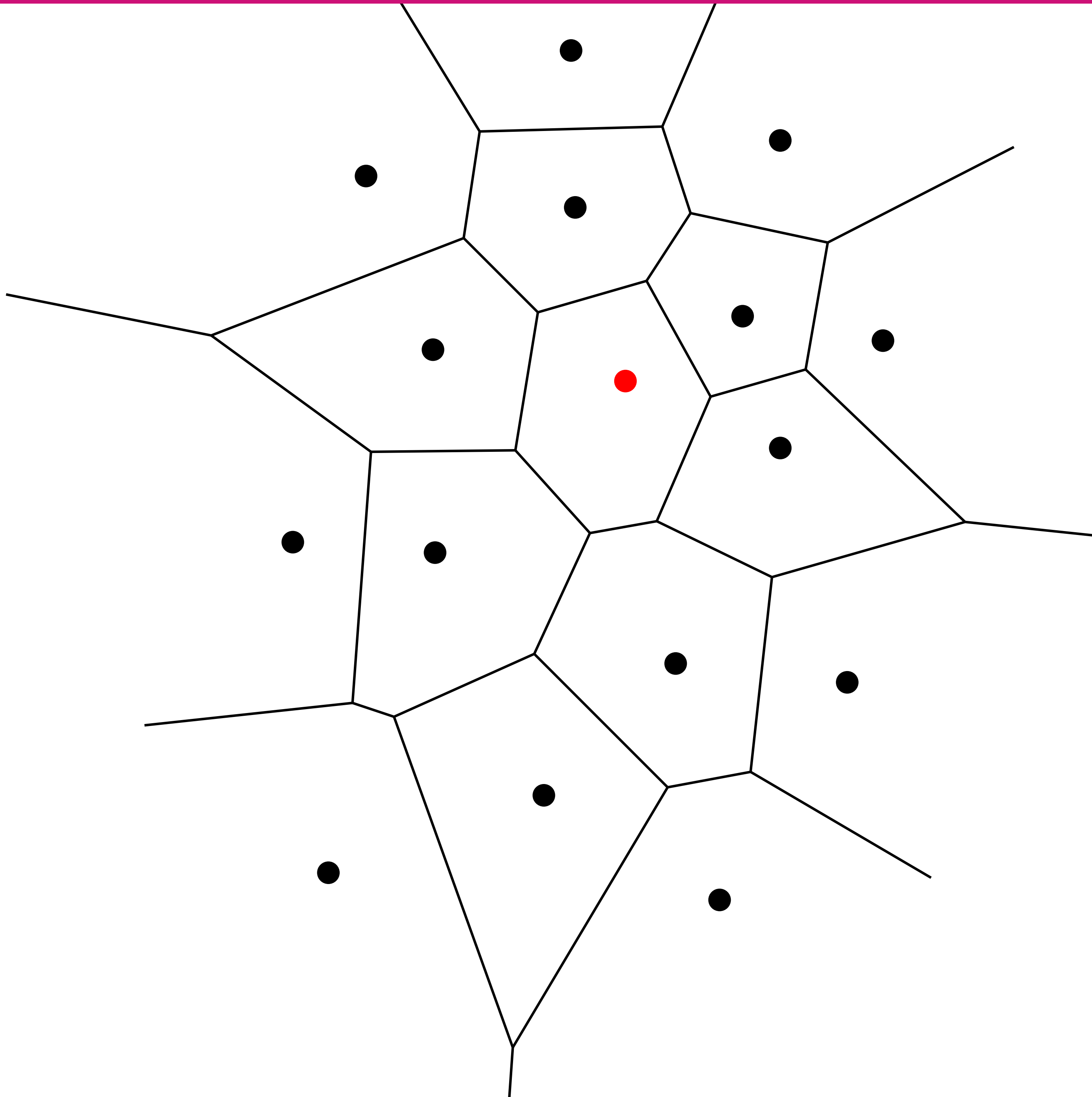
Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

... add point p_{i+1}

... compute its region

... and prune the initial diagram.



Incremental algorithm

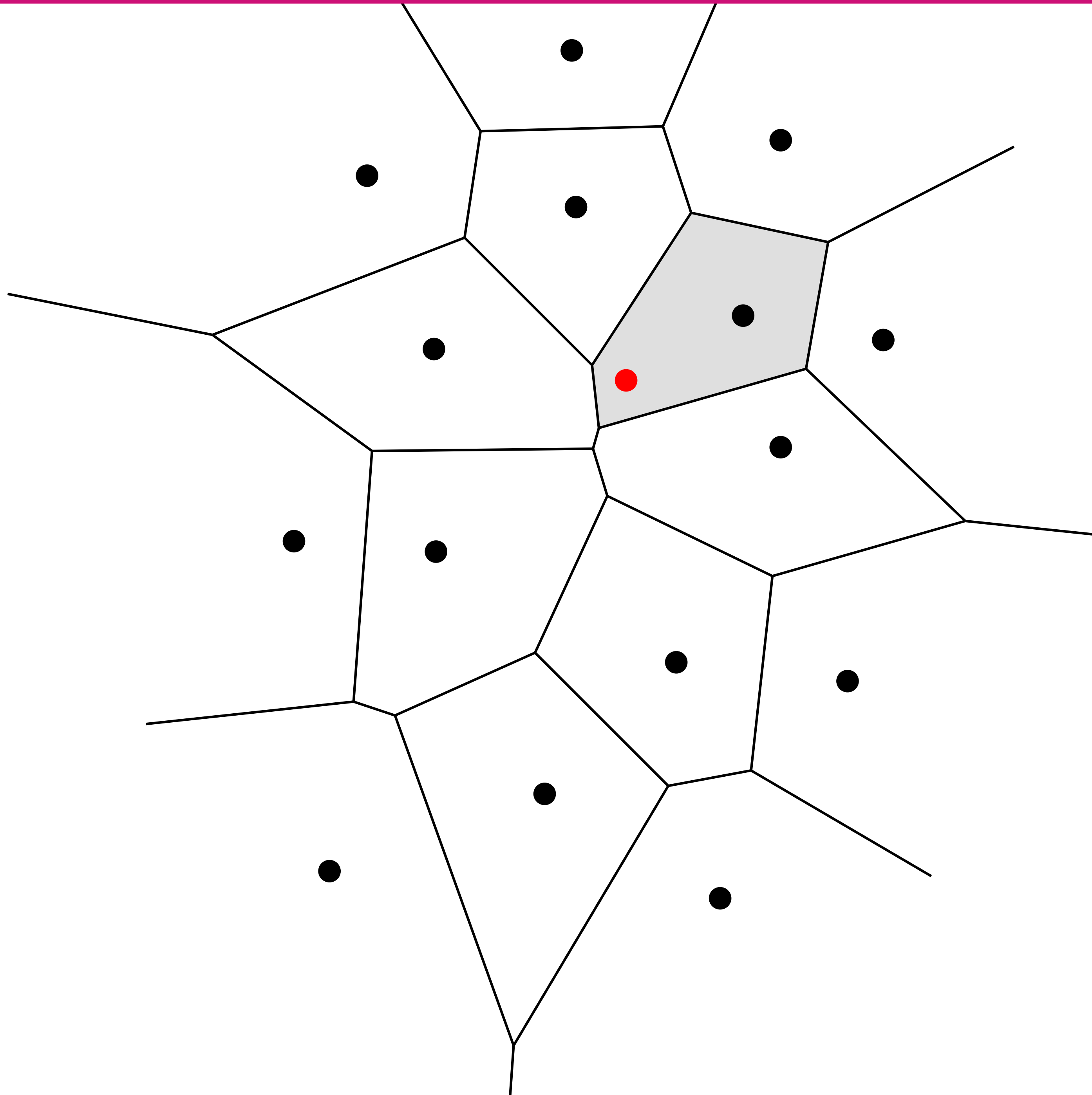
Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

... add point p_{i+1}

Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

... and prune the initial diagram.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

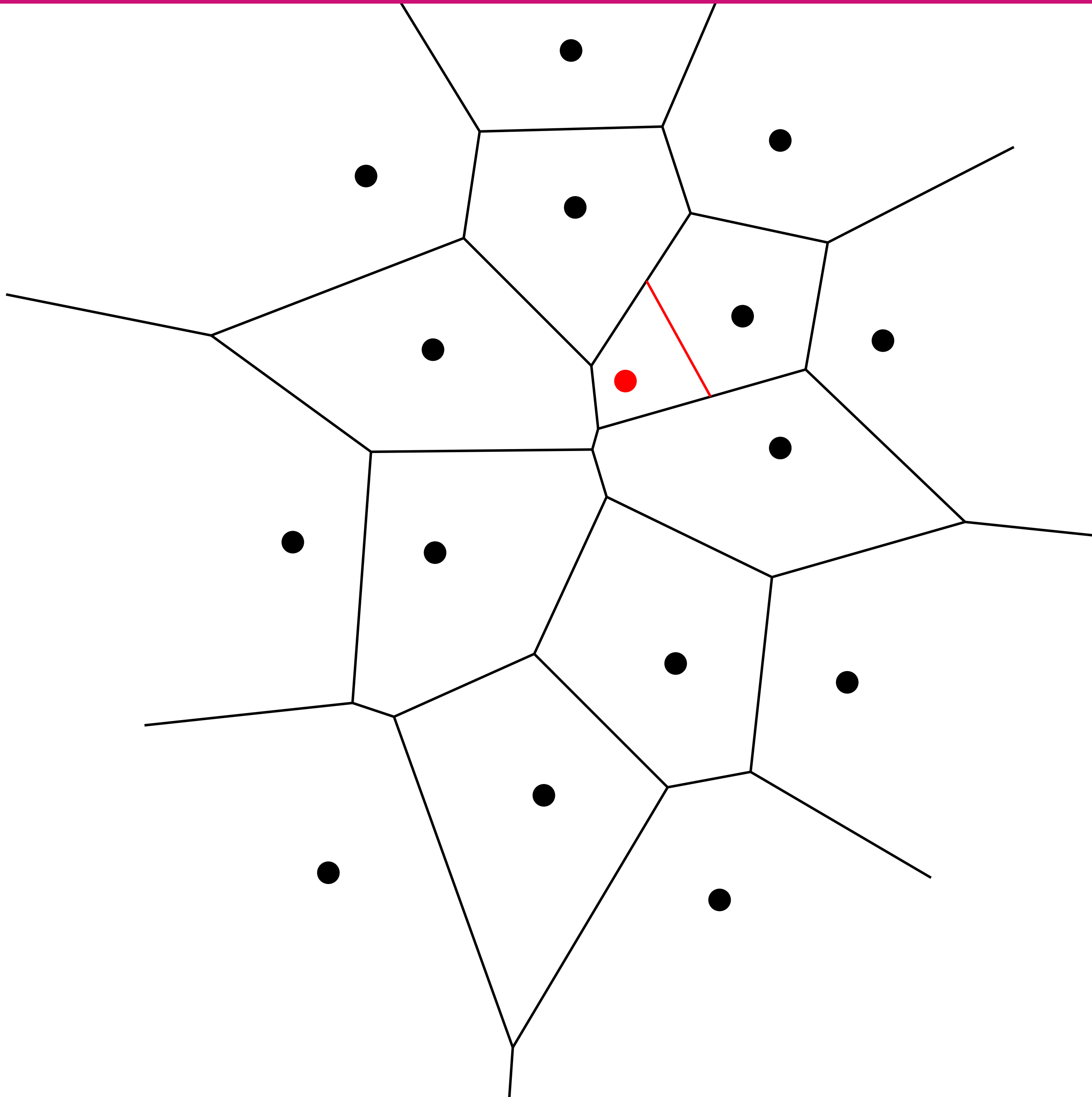
... add point p_{i+1}

Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

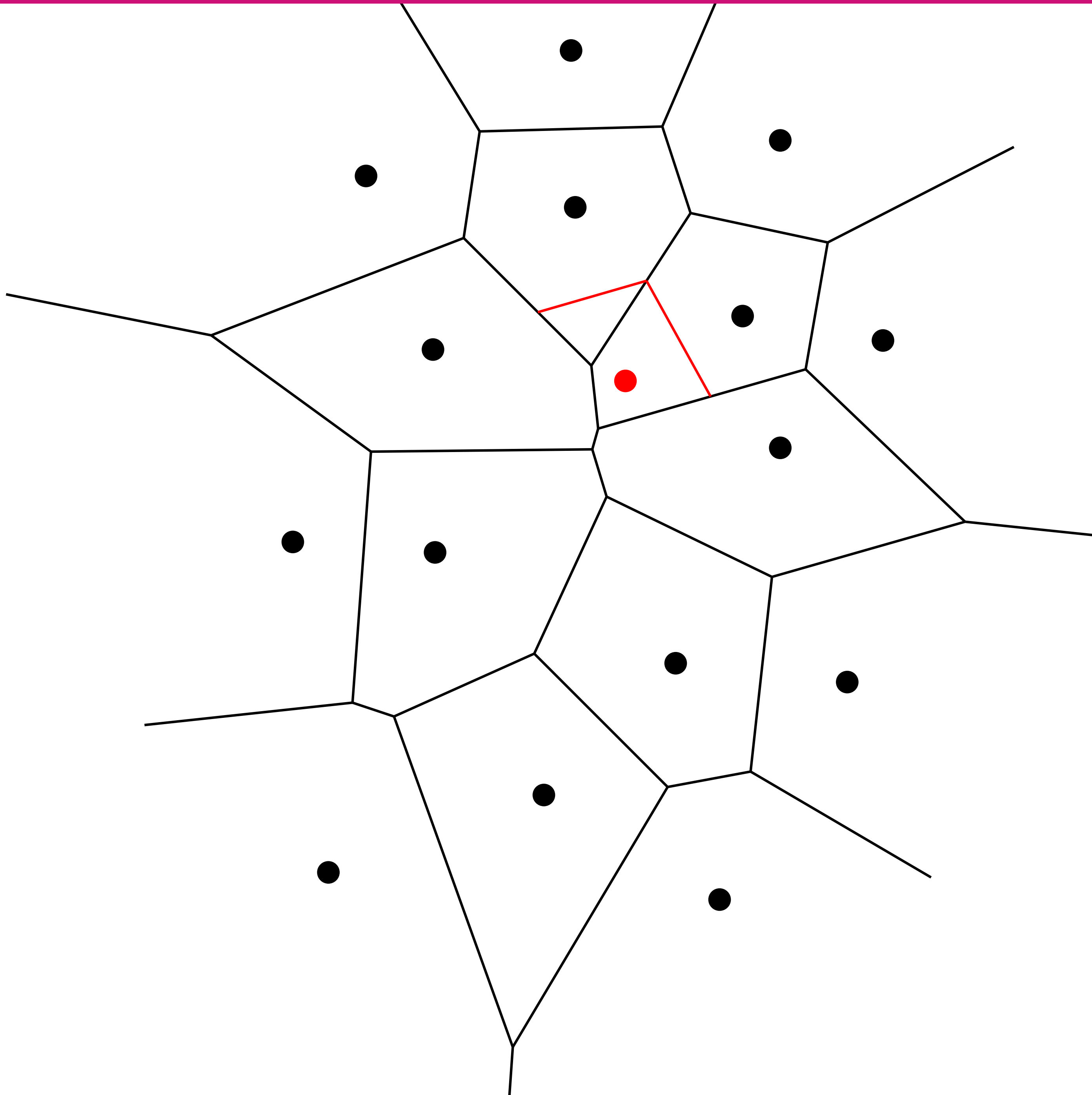
... add point p_{i+1}

Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

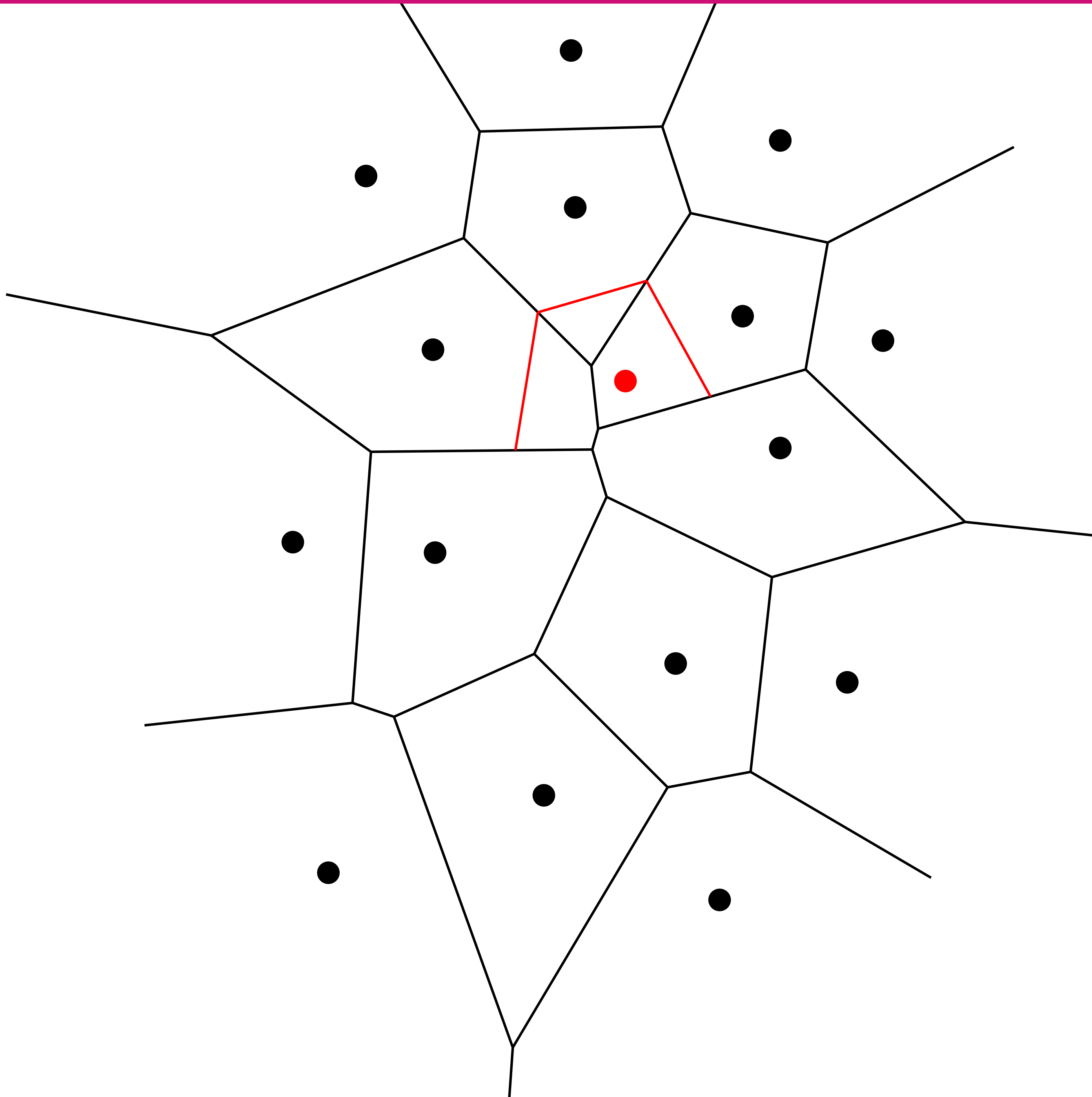
... add point p_{i+1}

Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

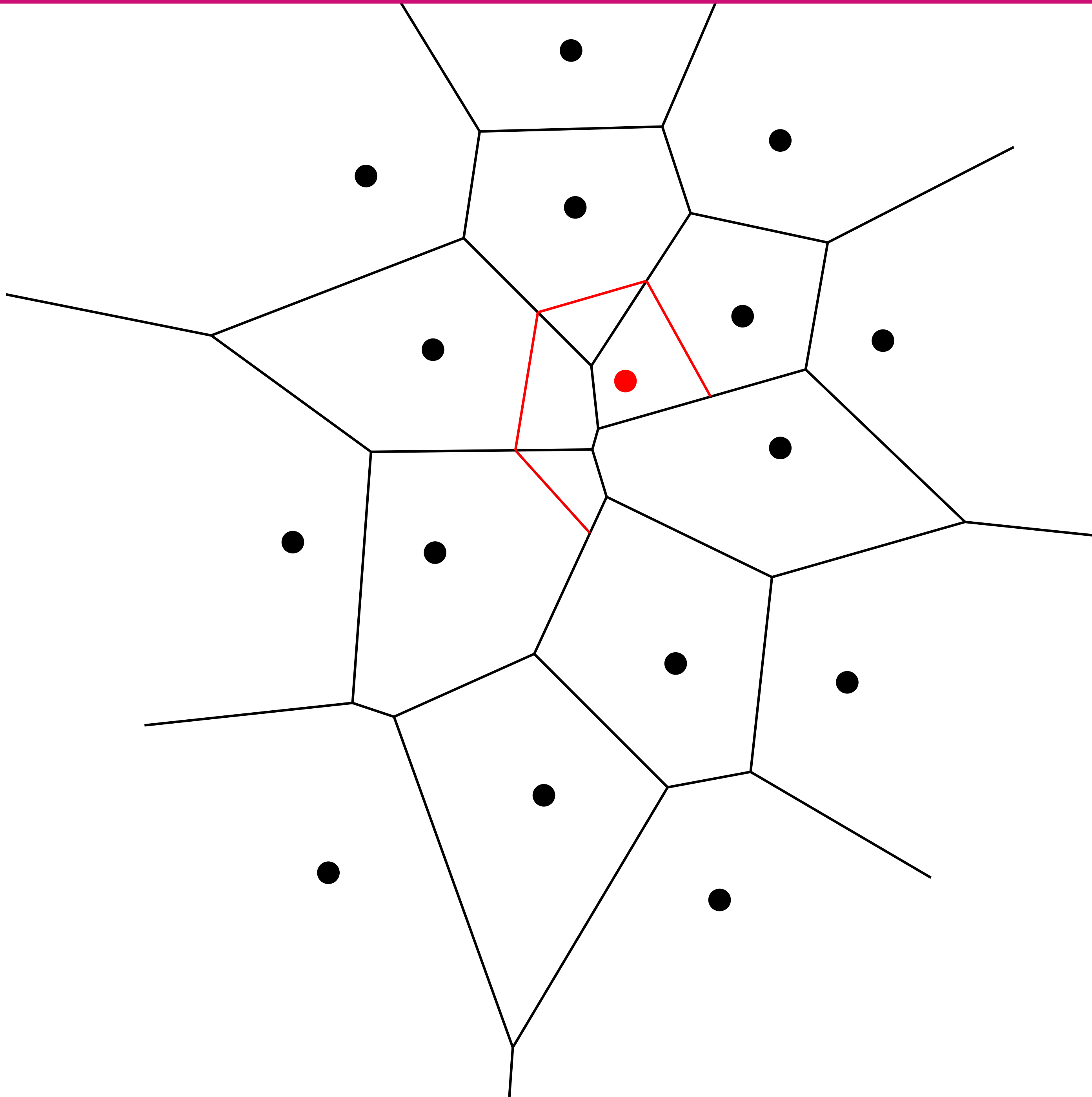
... add point p_{i+1}

Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

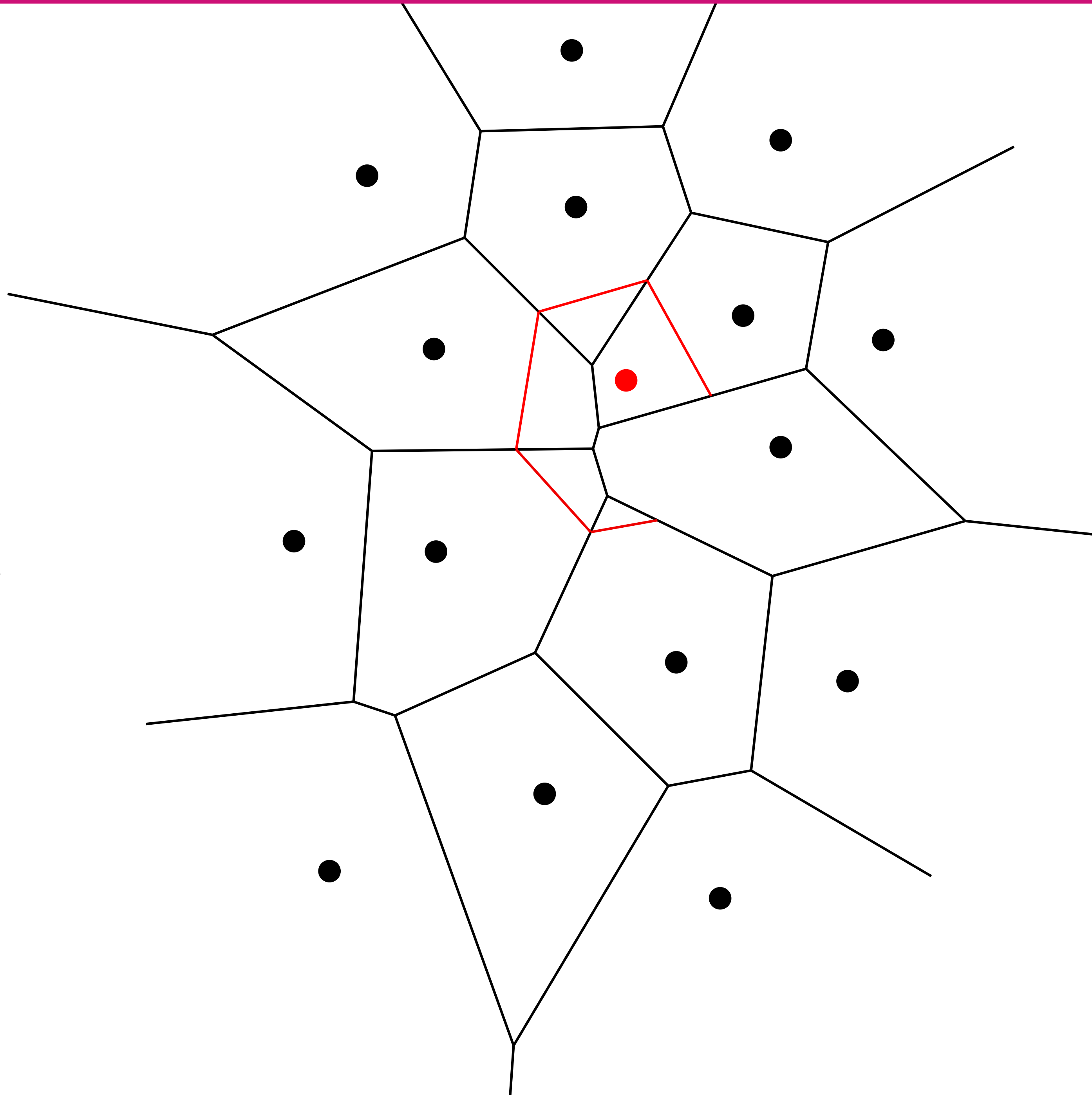
... add point p_{i+1}

Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

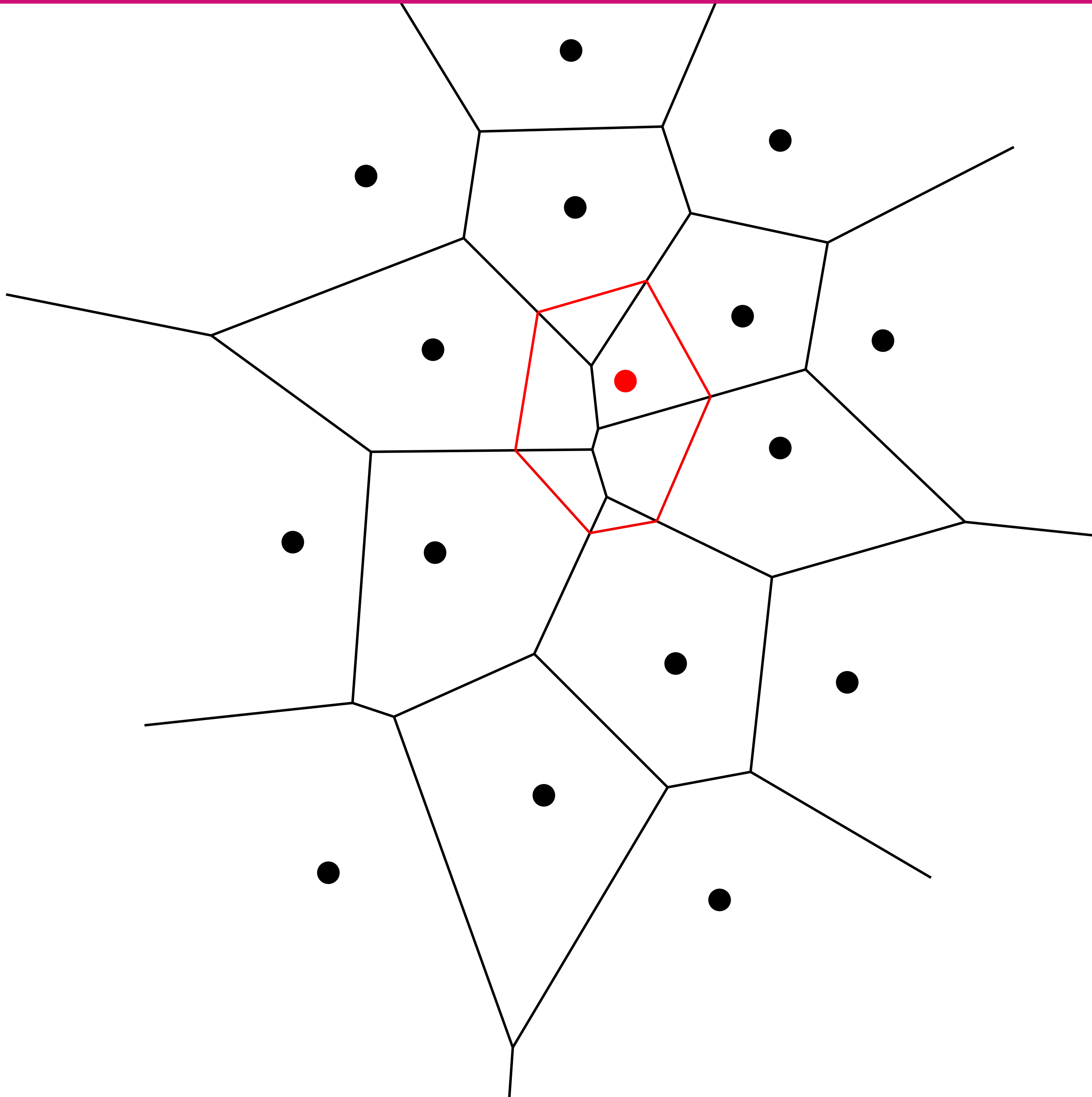
... add point p_{i+1}

Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

... add point p_{i+1}

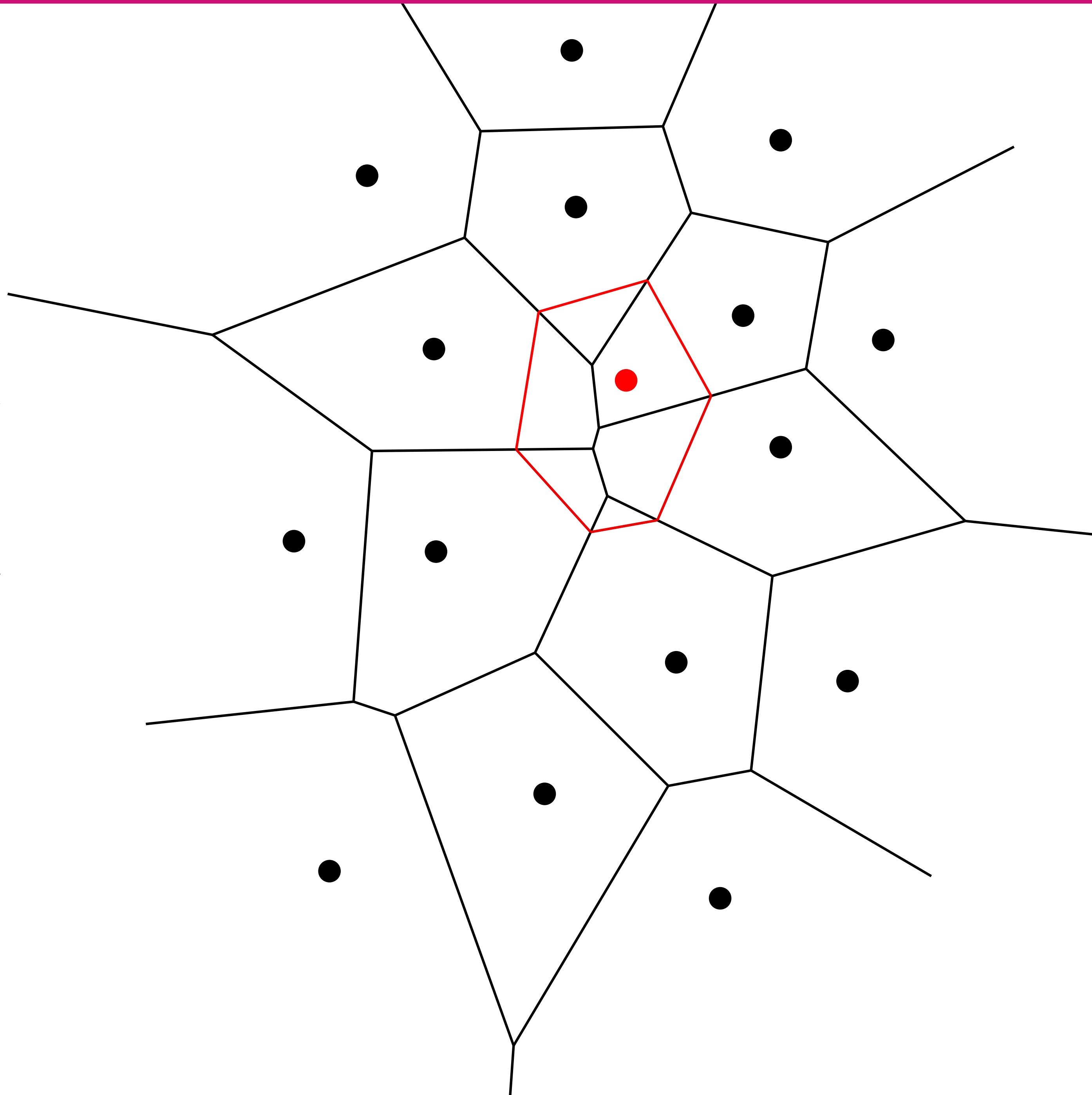
Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.

While building the Voronoi region of p_{i+1}, update the DCEL.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

... add point p_{i+1}

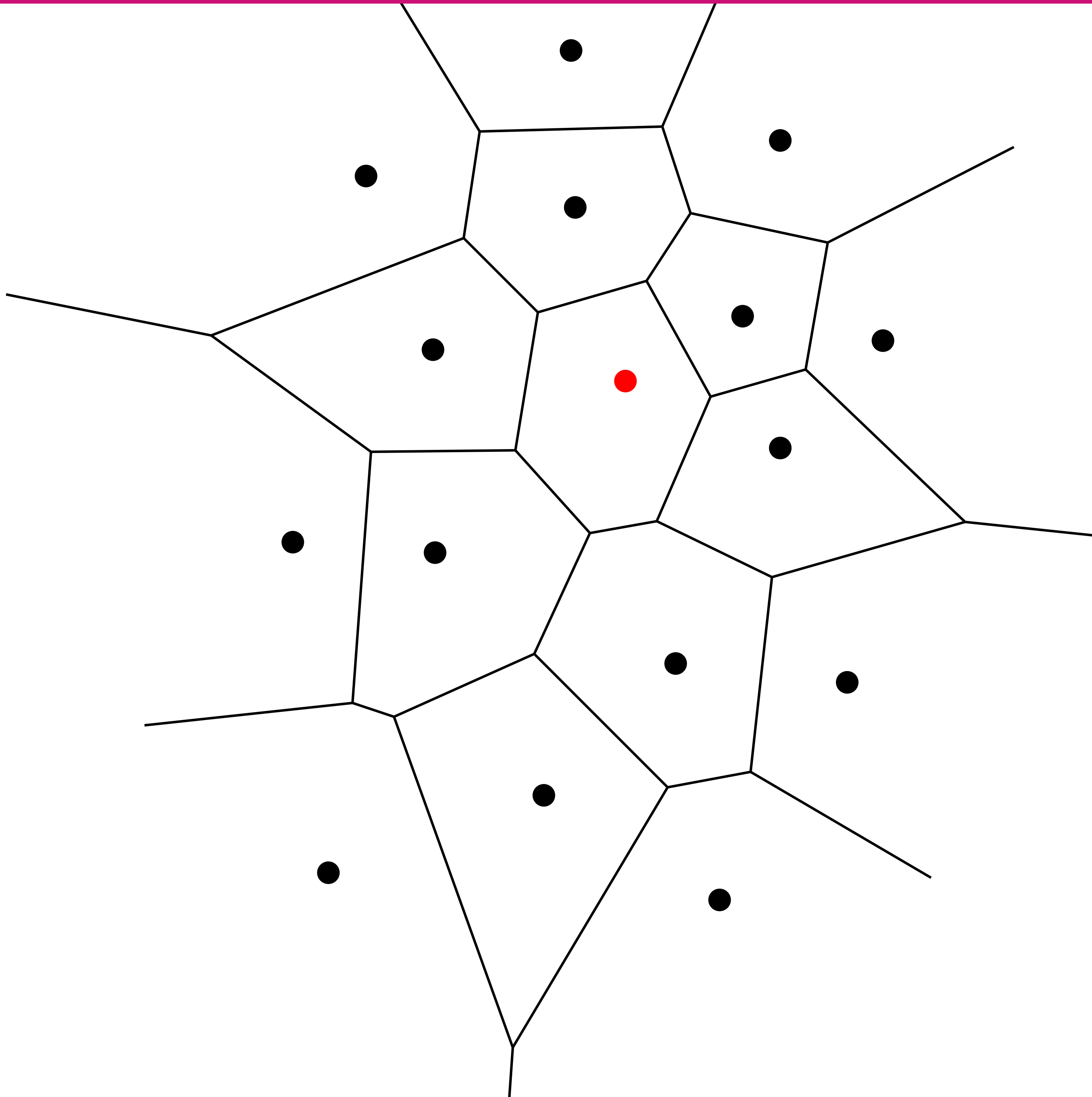
Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

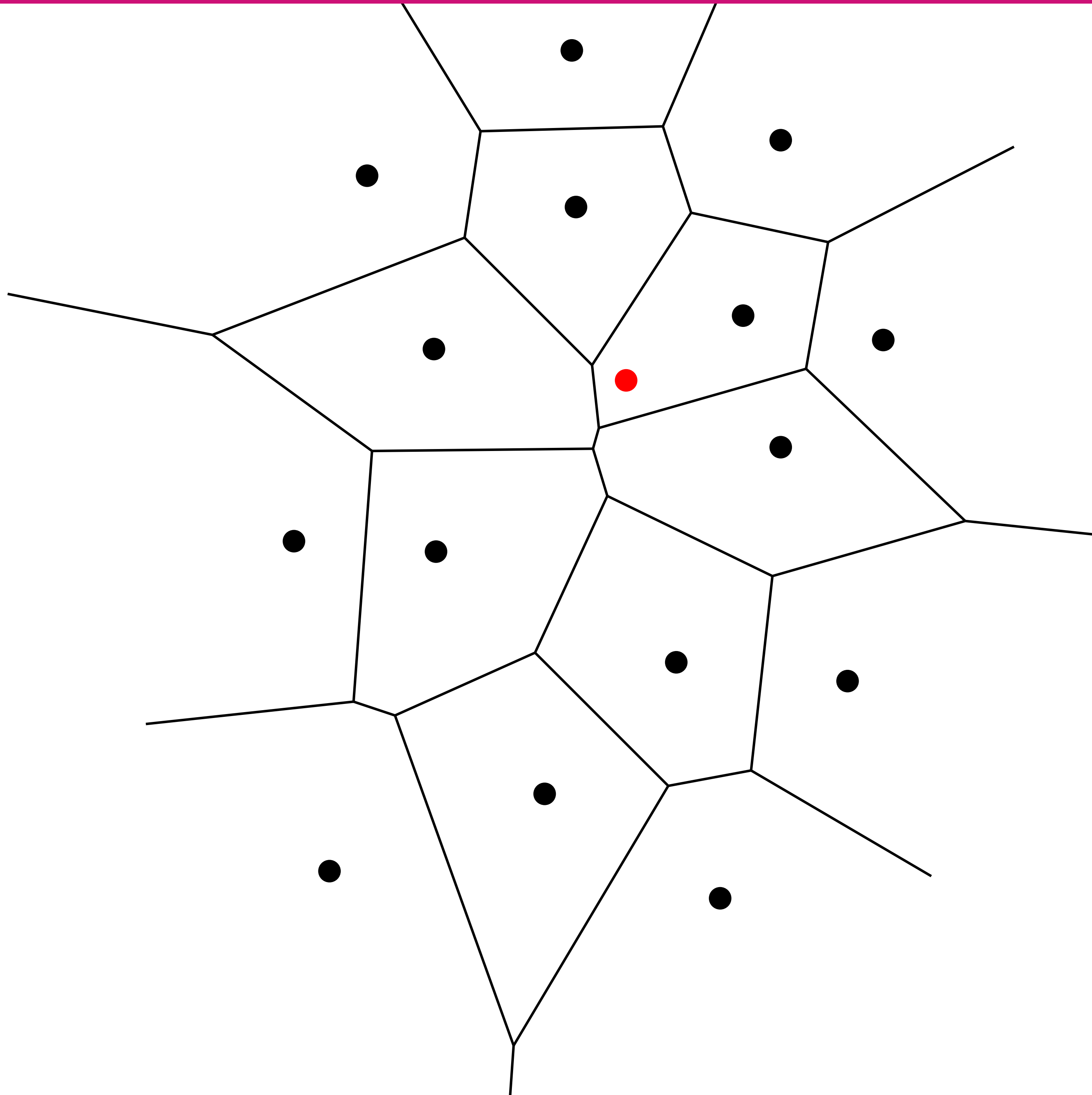
... and prune the initial diagram.

While building the Voronoi region of p_{i+1}, update the DCEL.



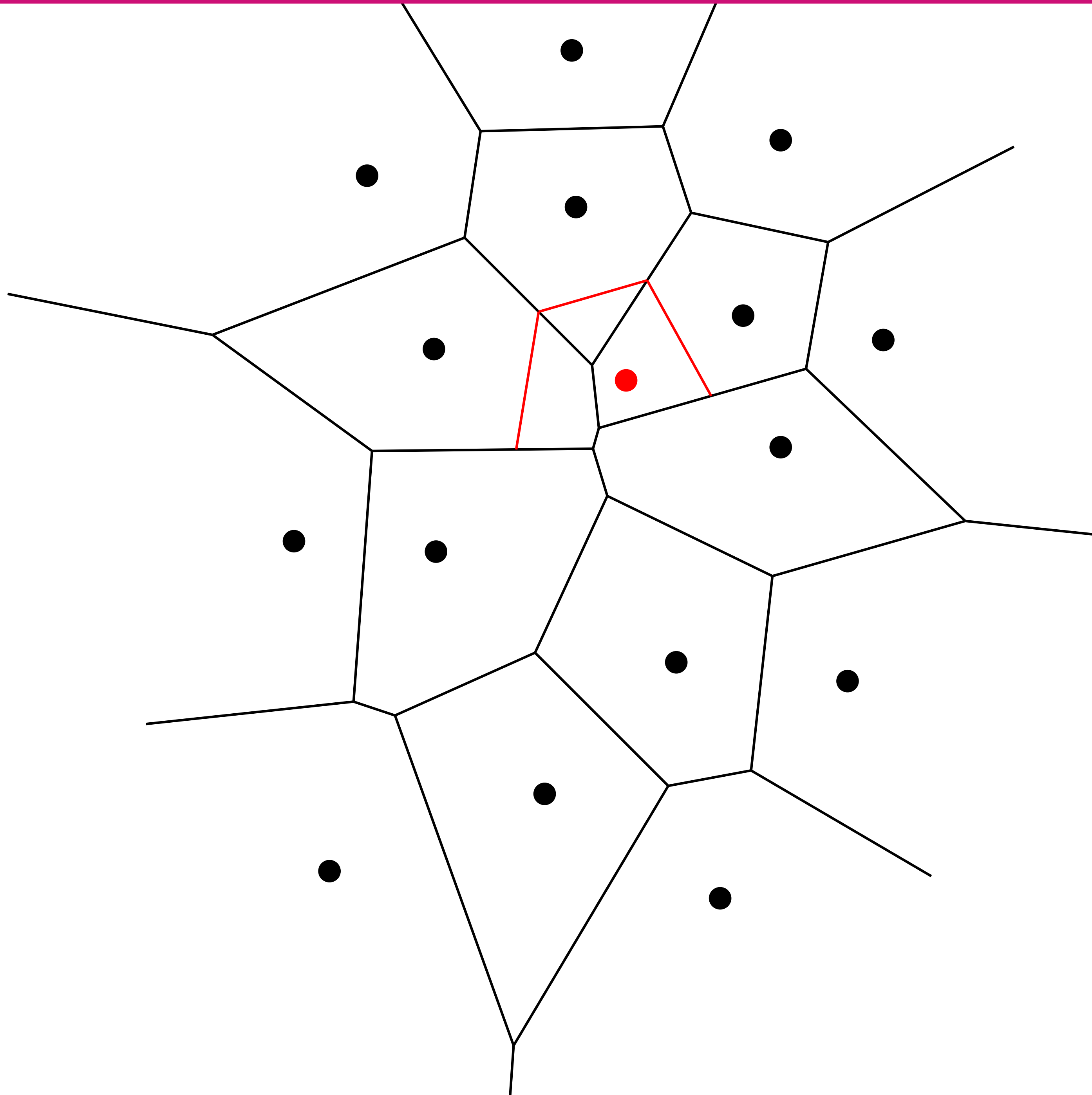
Incremental algorithm

How to update the DCEL



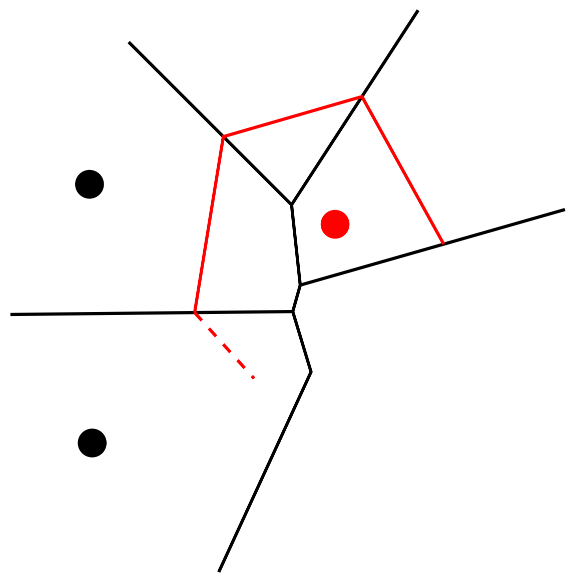
Incremental algorithm

How to update the DCEL



Incremental algorithm

How to update the DCEL

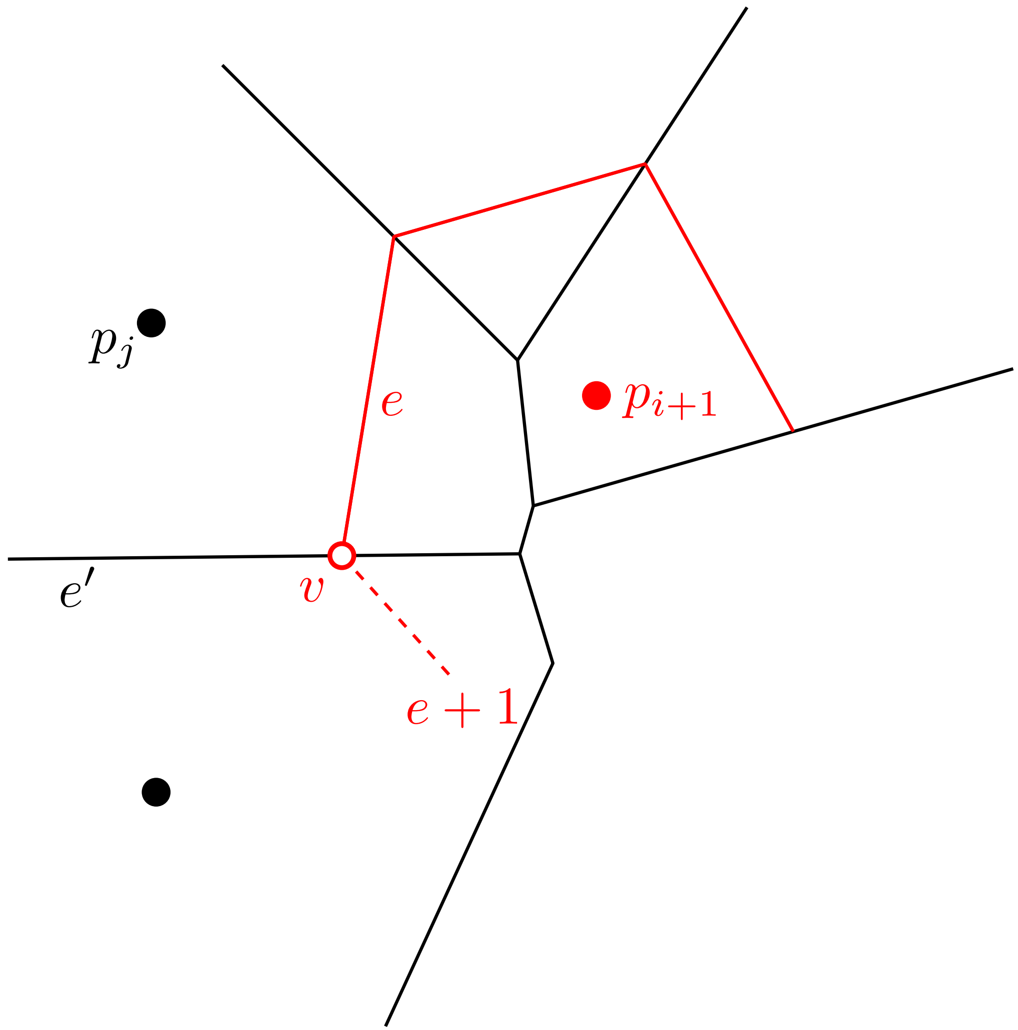


Incremental algorithm

How to update the DCEL

Each time an edge e, generated by p_{i+1} and p_j, intersects a preexistent edge, e', a new vertex v is created and a new edge starts, $e+1$. Then, these are the tasks to perform:

- Create v with $e(v) = e$
- Assign $v_E(e) = v$, $e_N(e) = e'$, $f_L(e) = i+1$, $f_R(e) = j$
- Create $e+1$ and assign $v_B(e+1) = v$, $e_P(e+1) = e$
- Delete all edges of the region of p_j, that lie between $v_B(e)$ and $v_E(e)$ in clockwise order
- Update $v_*(e') = v$ and $e_*(e') = e+1$
- Update $e(p_j) = e$



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

... add point p_{i+1}

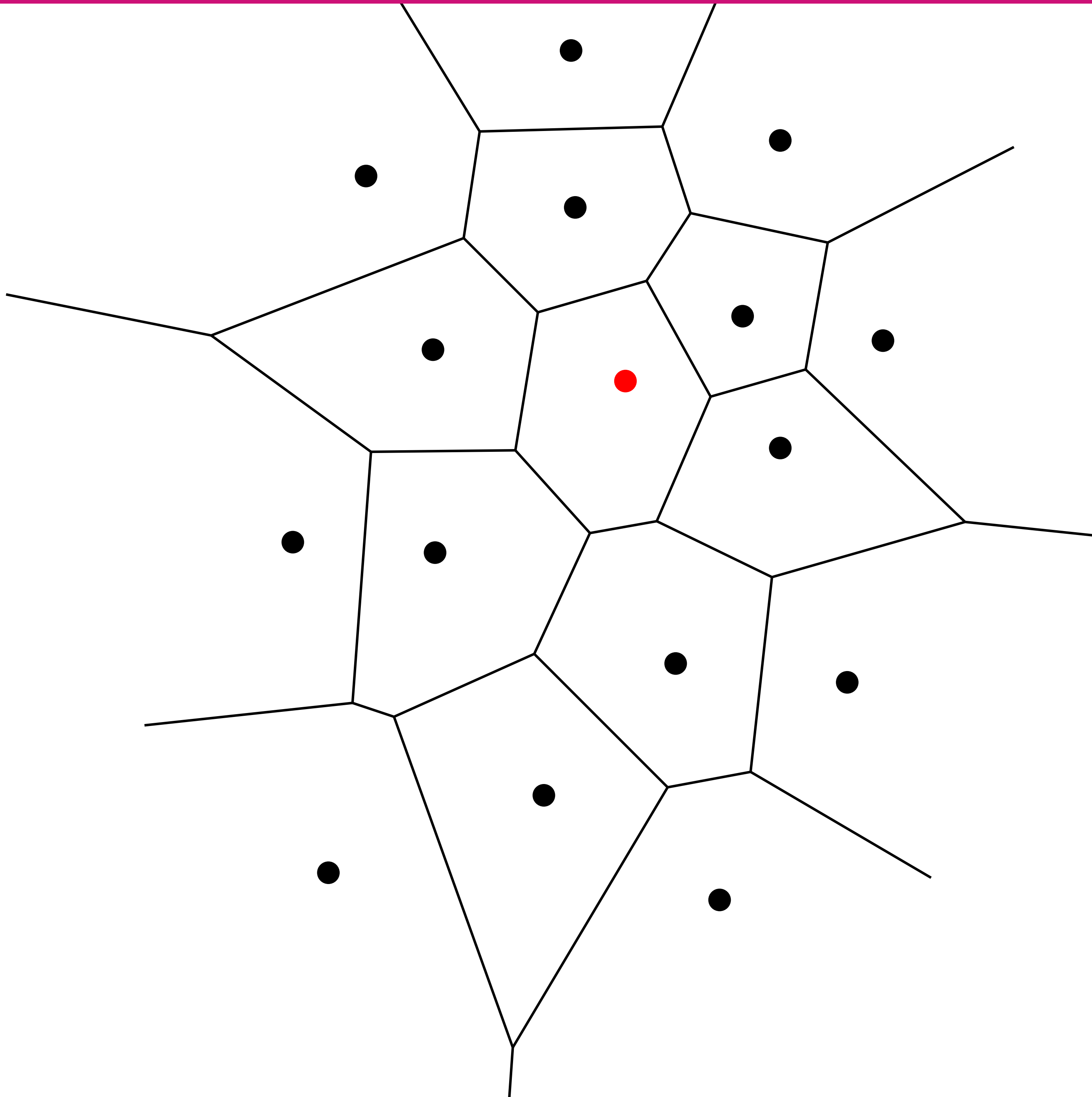
Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.

While building the Voronoi region of p_{i+1}, update the DCEL.



Incremental algorithm

Starting with the Voronoi diagram of $\{p_1, \ldots, p_i\}$...

... add point p_{i+1}

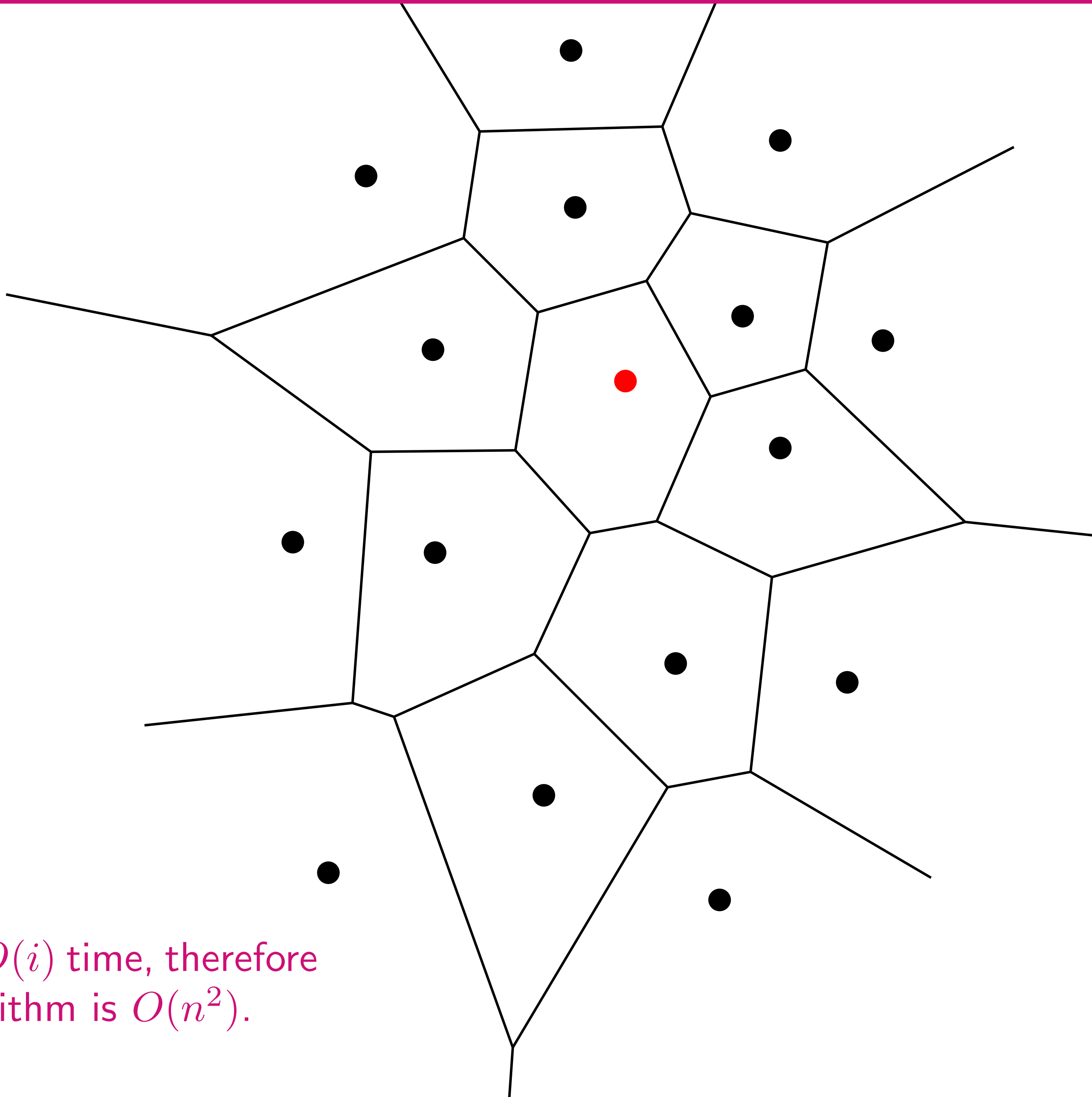
Explore all candidates to find the site p_j $(1 \leq j \leq i)$ closest to p_{i+1}.

... compute its region

Build its boundary starting from bisector $b_{i+1,j}$.

... and prune the initial diagram.

While building the Voronoi region of p_{i+1}, update the DCEL.

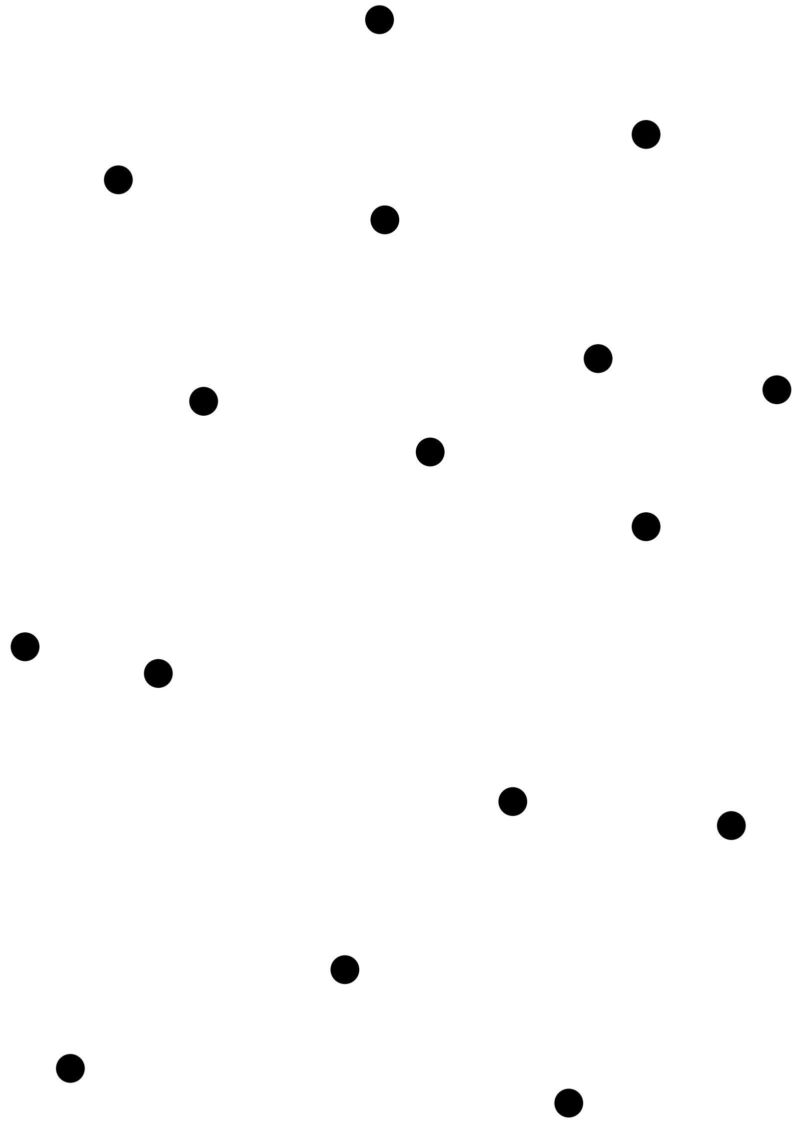


Running time: Each step runs in $O(i)$ time, therefore the total running time of the algorithm is $O(n^2)$.

divide and conquer algorithm

Divide and conquer algorithm

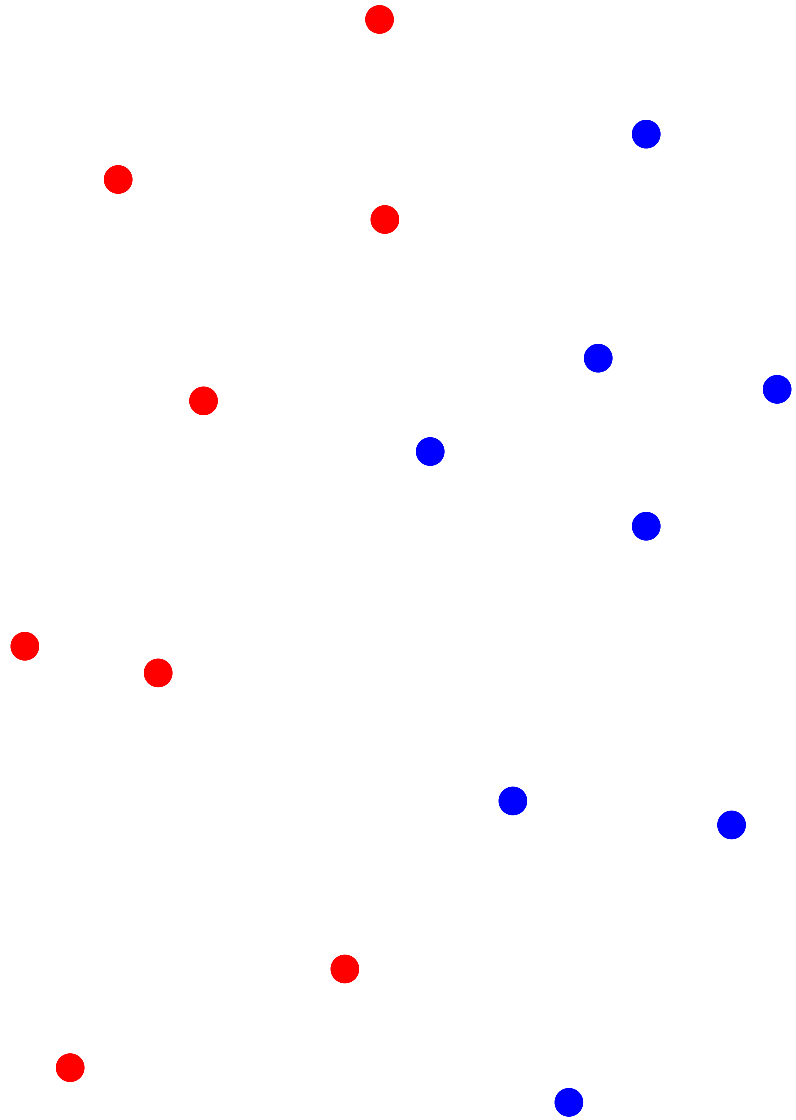
Let P be a set of n points in the plane.



Divide and conquer algorithm

Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

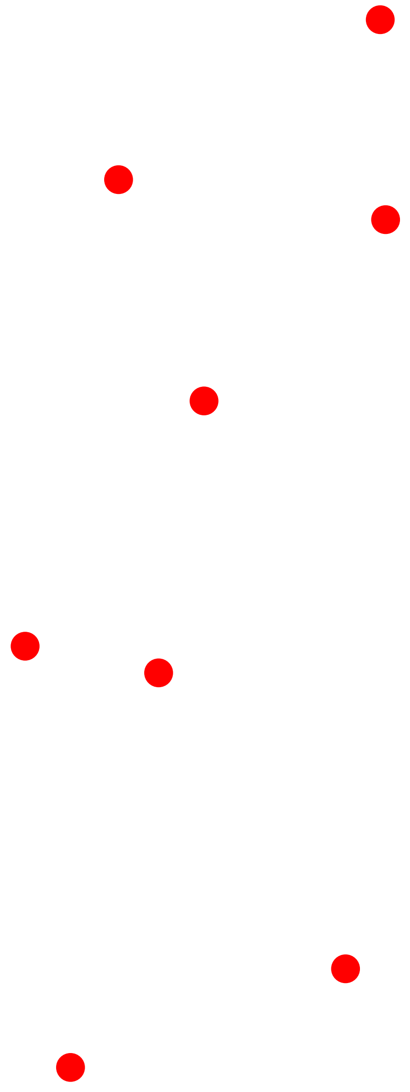


Divide and conquer algorithm

Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

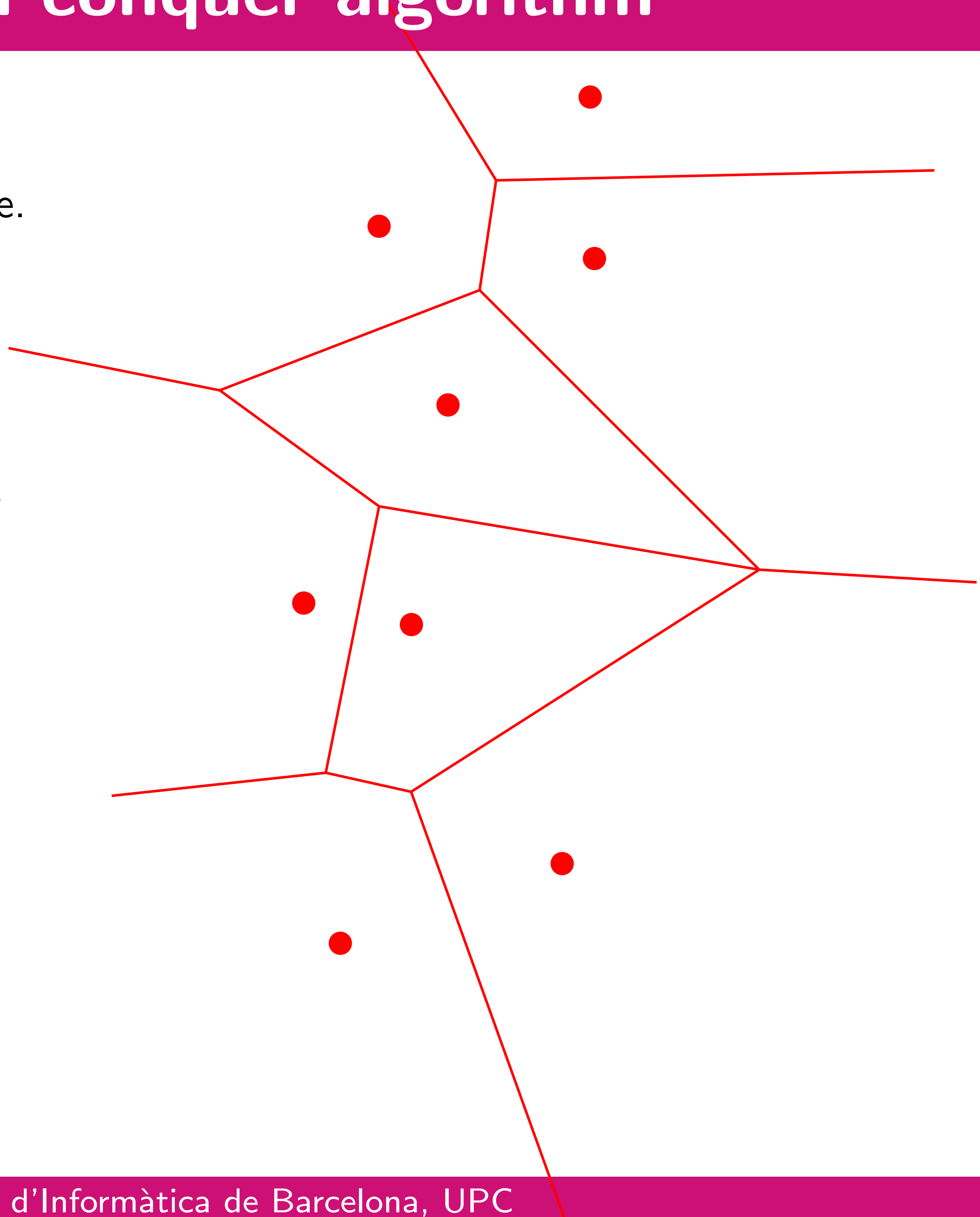


Divide and conquer algorithm

Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

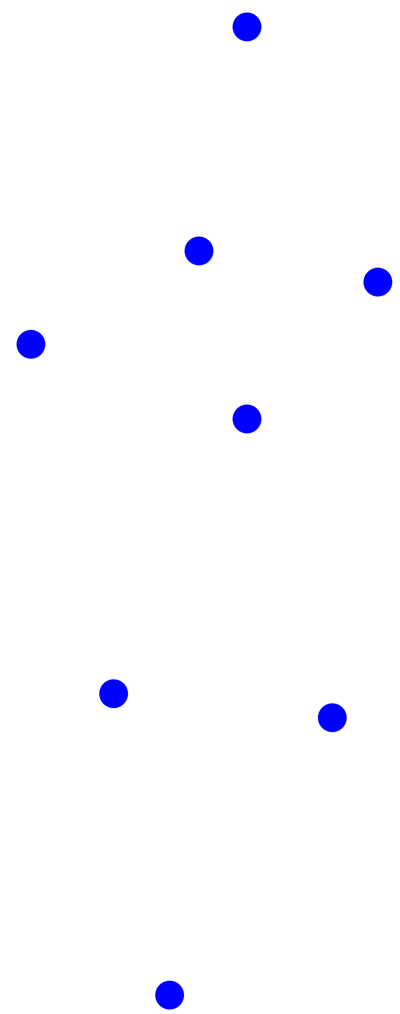


Divide and conquer algorithm

Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

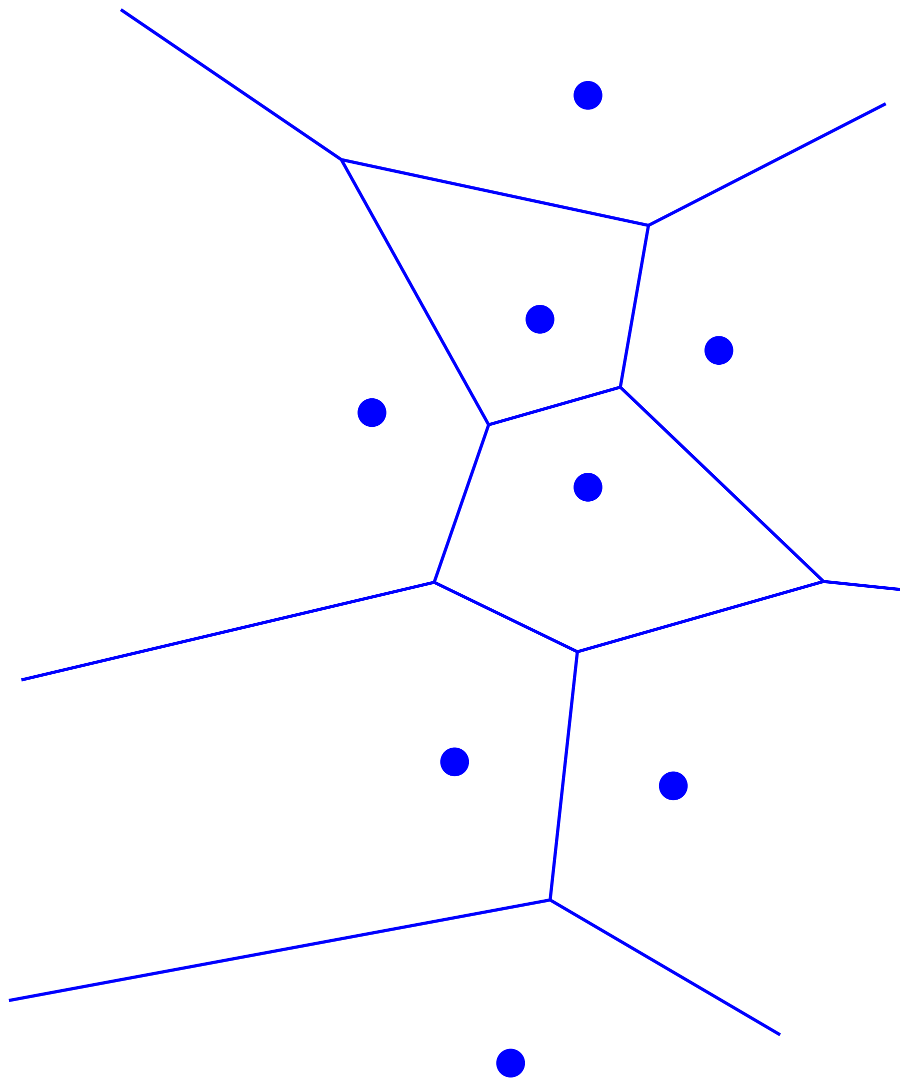


Divide and conquer algorithm

Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

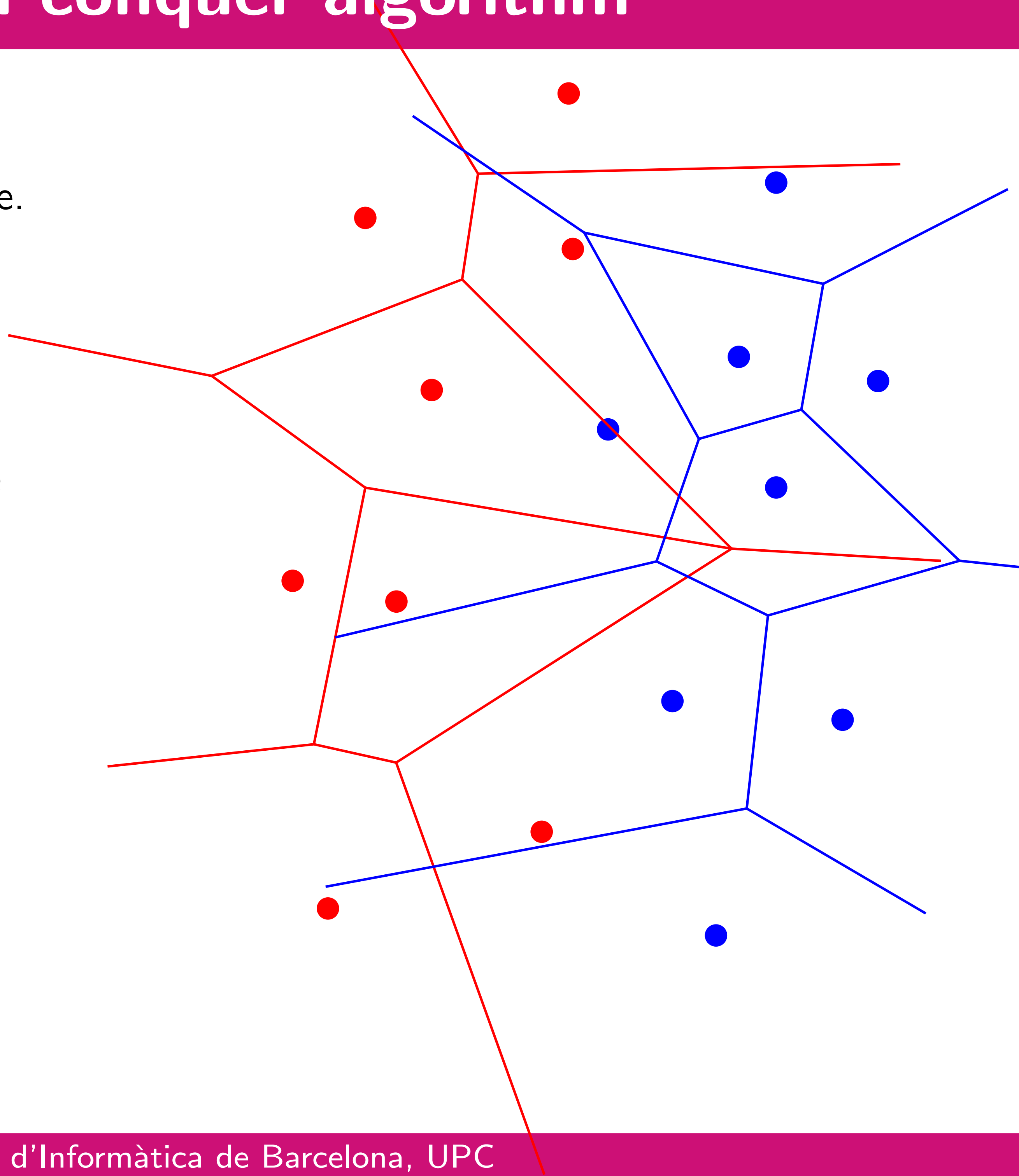


Divide and conquer algorithm

Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...



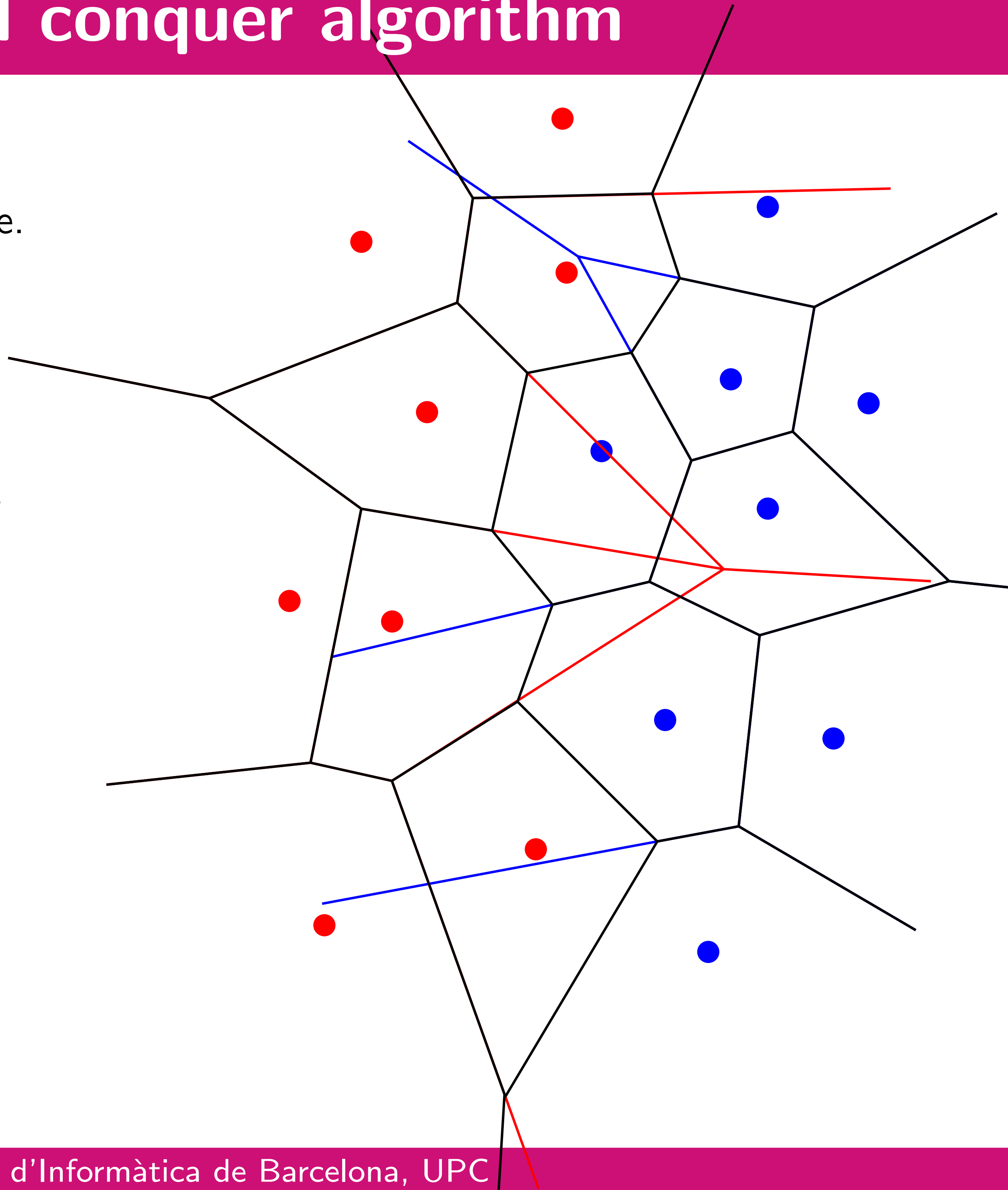
Divide and conquer algorithm

Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

...then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!



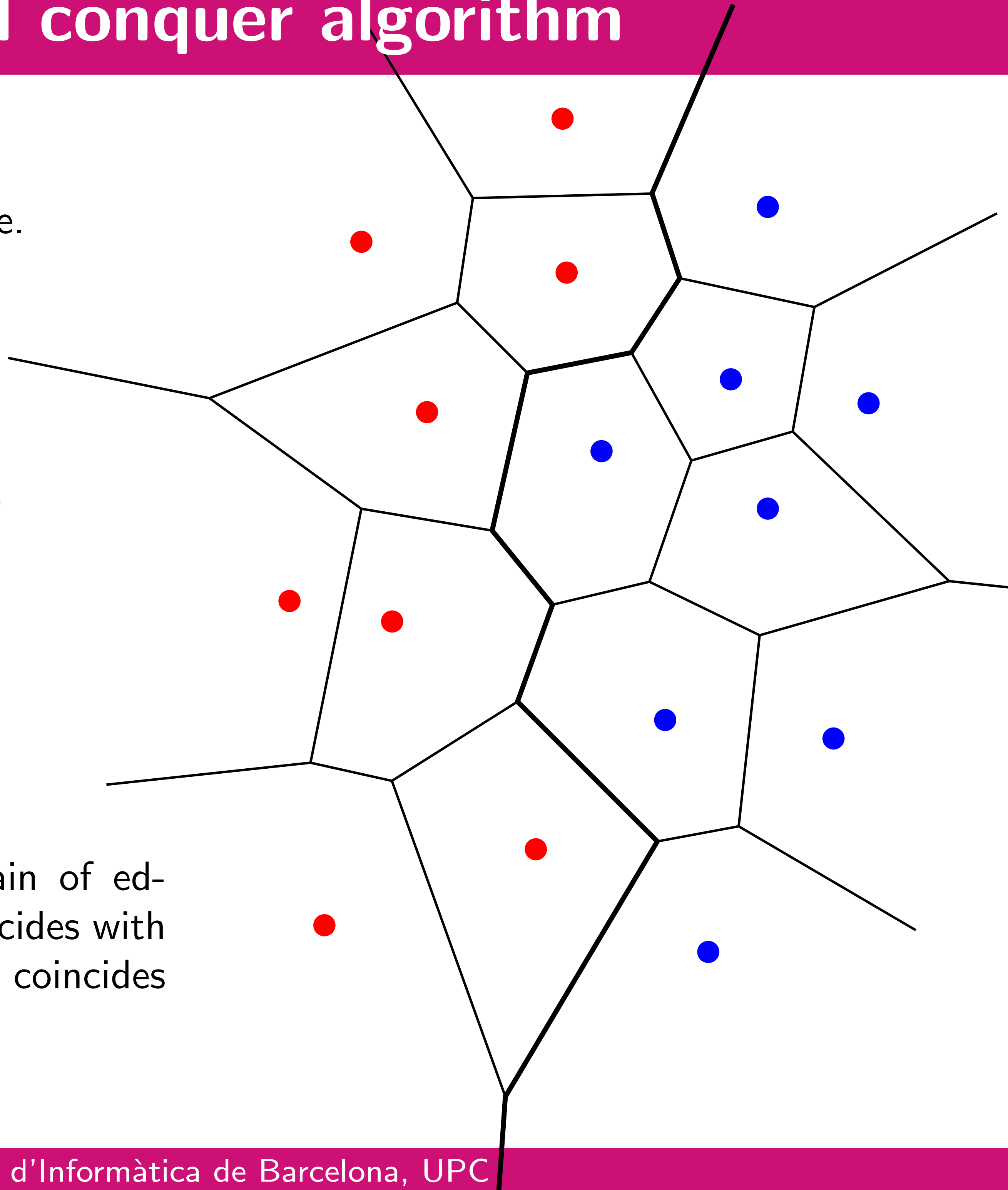
Divide and conquer algorithm

Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

...then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!



In fact, there exists a monotone chain of edges of $Vor(P)$ such that $Vor(P)$ coincides with $Vor(R)$ to the left of the chain, and it coincides with $Vor(B)$ to its right.

Divide and conquer algorithm

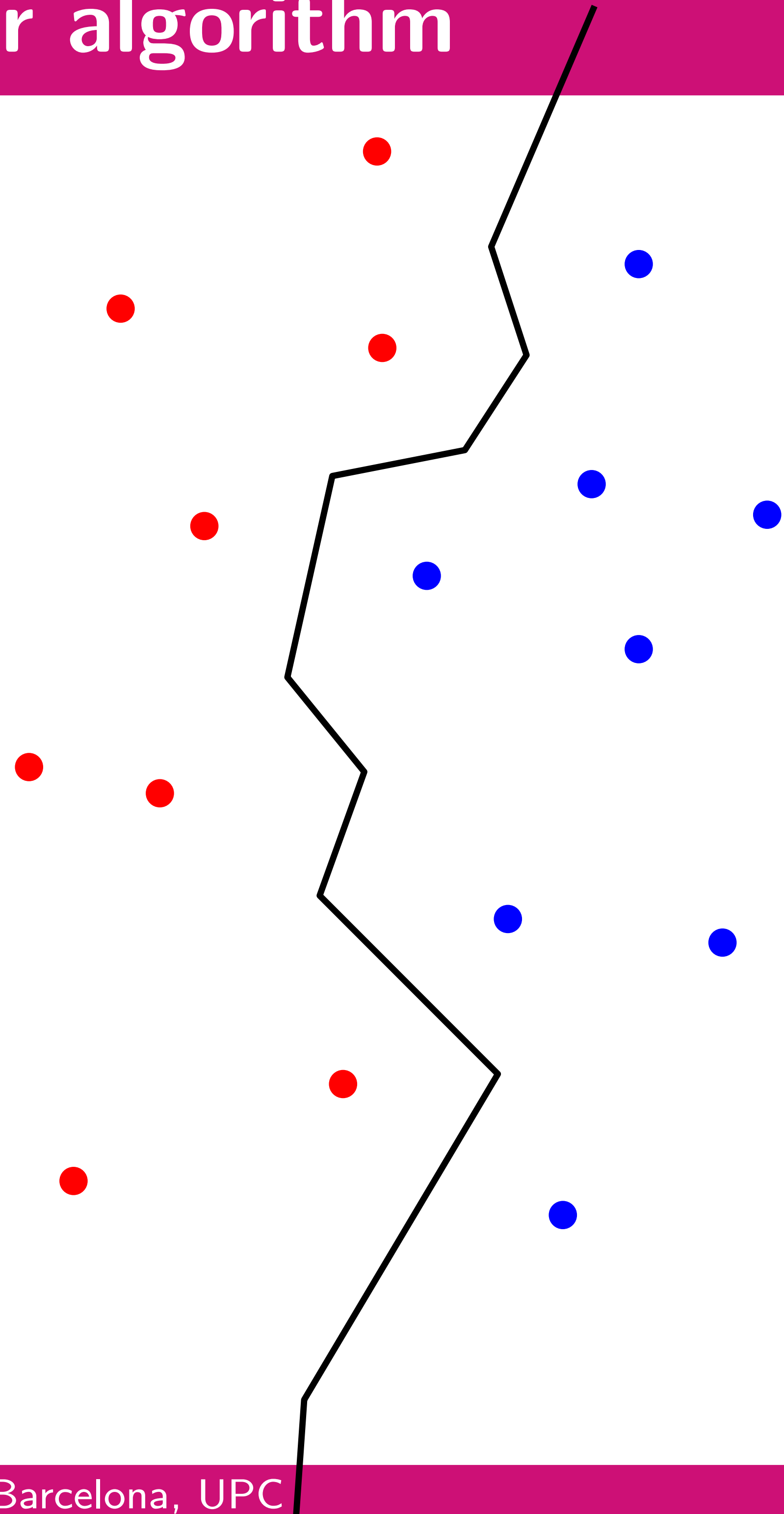
Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

...then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!

In fact, there exists a monotone chain of edges of $Vor(P)$ such that $Vor(P)$ coincides with $Vor(R)$ to the left of the chain, and it coincides with $Vor(B)$ to its right.



Divide and conquer algorithm

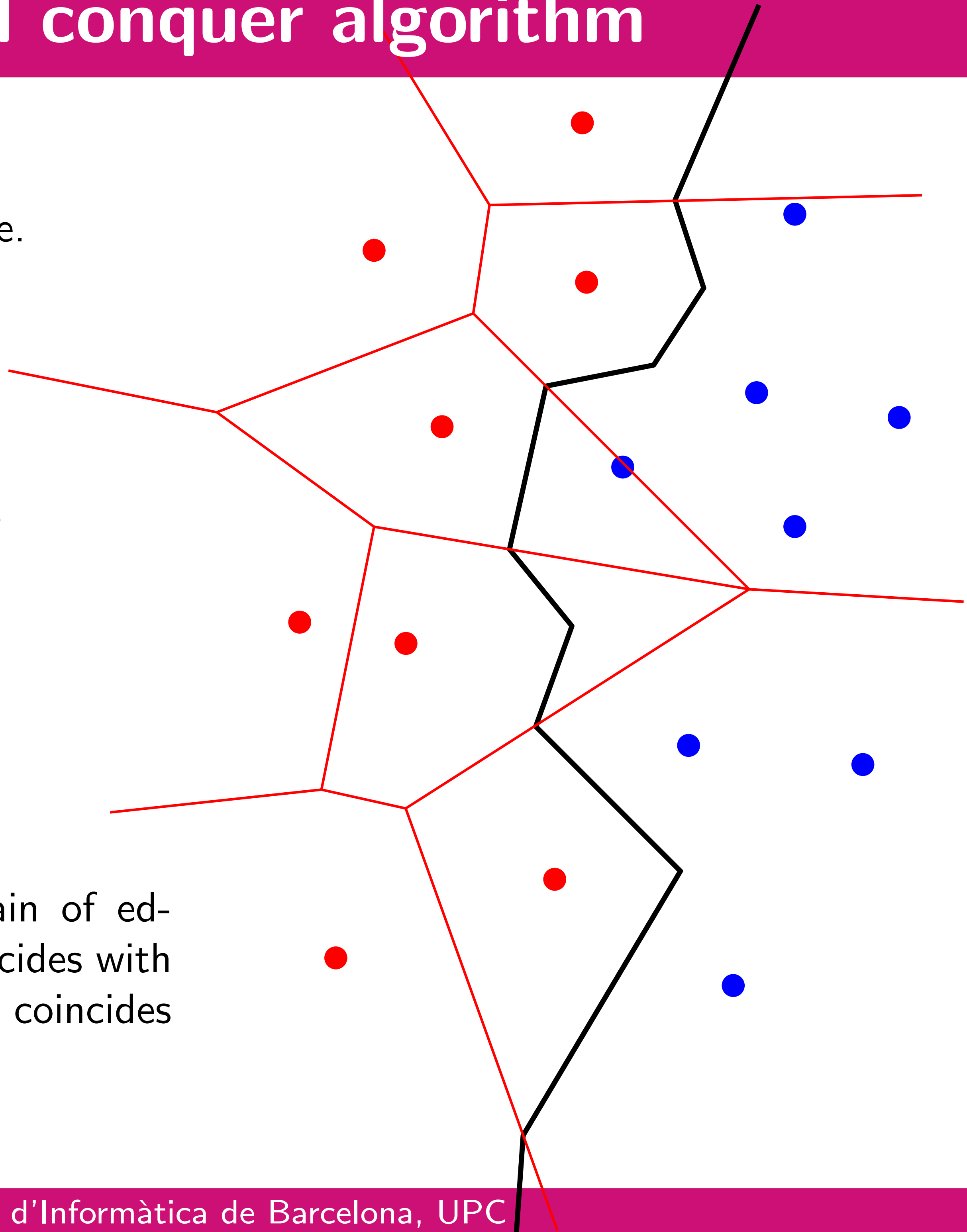
Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

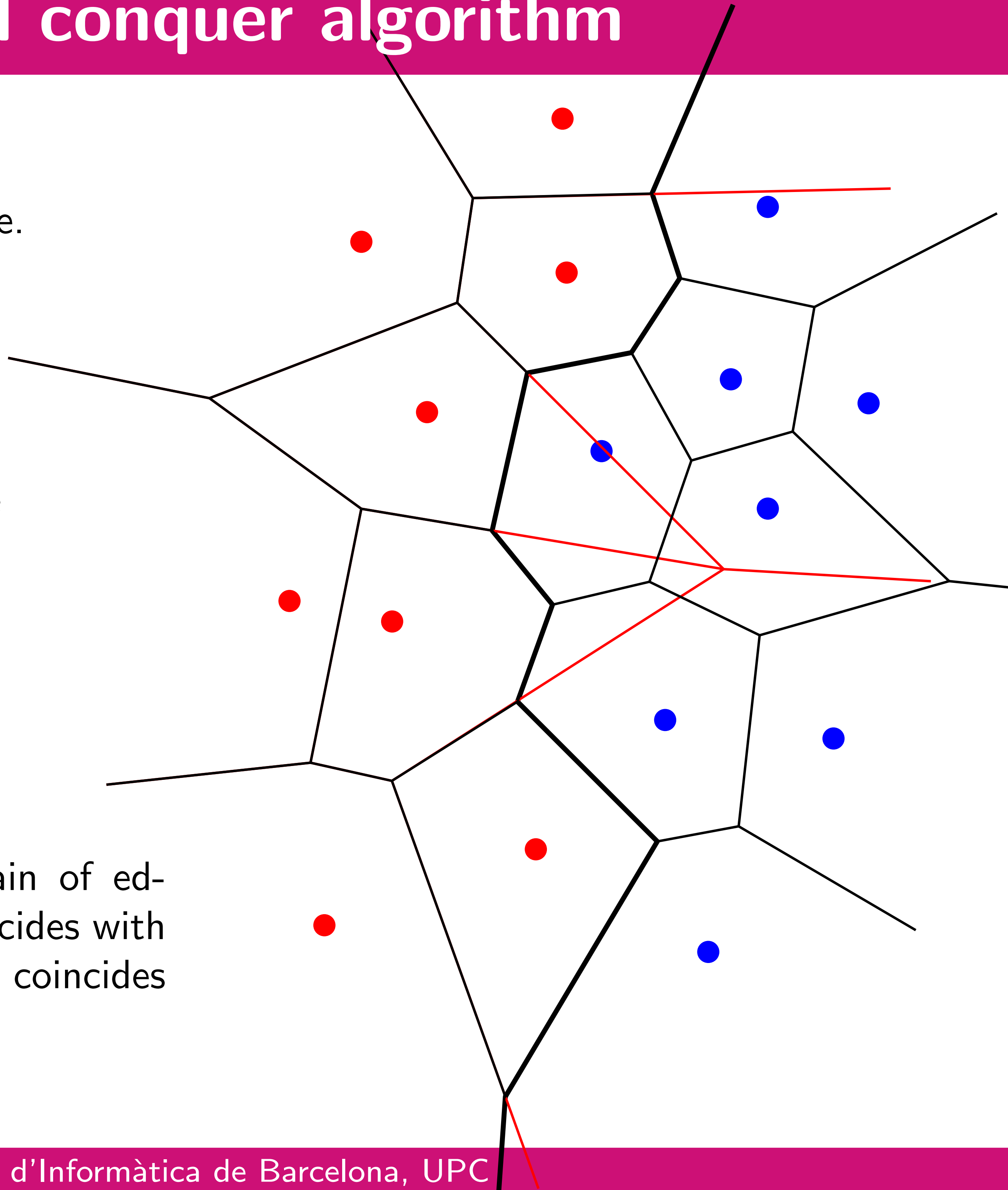
...consider the Voronoi diagram of the sets R and B...

...then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!

In fact, there exists a monotone chain of edges of $Vor(P)$ such that $Vor(P)$ coincides with $Vor(R)$ to the left of the chain, and it coincides with $Vor(B)$ to its right.



Divide and conquer algorithm



Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

...then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!

In fact, there exists a monotone chain of edges of $Vor(P)$ such that $Vor(P)$ coincides with $Vor(R)$ to the left of the chain, and it coincides with $Vor(B)$ to its right.

Divide and conquer algorithm

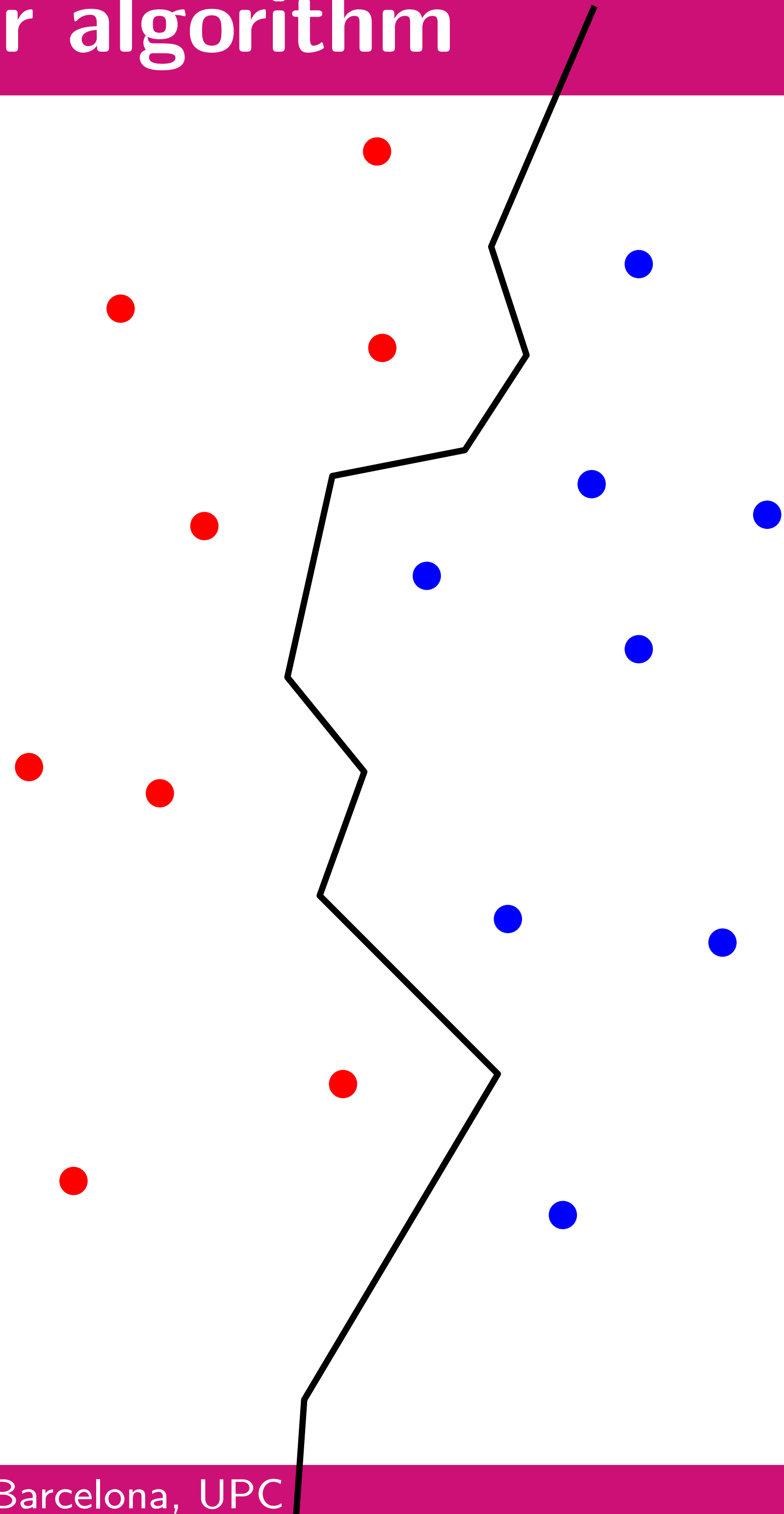
Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

...then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!

In fact, there exists a monotone chain of edges of $Vor(P)$ such that $Vor(P)$ coincides with $Vor(R)$ to the left of the chain, and it coincides with $Vor(B)$ to its right.



Divide and conquer algorithm

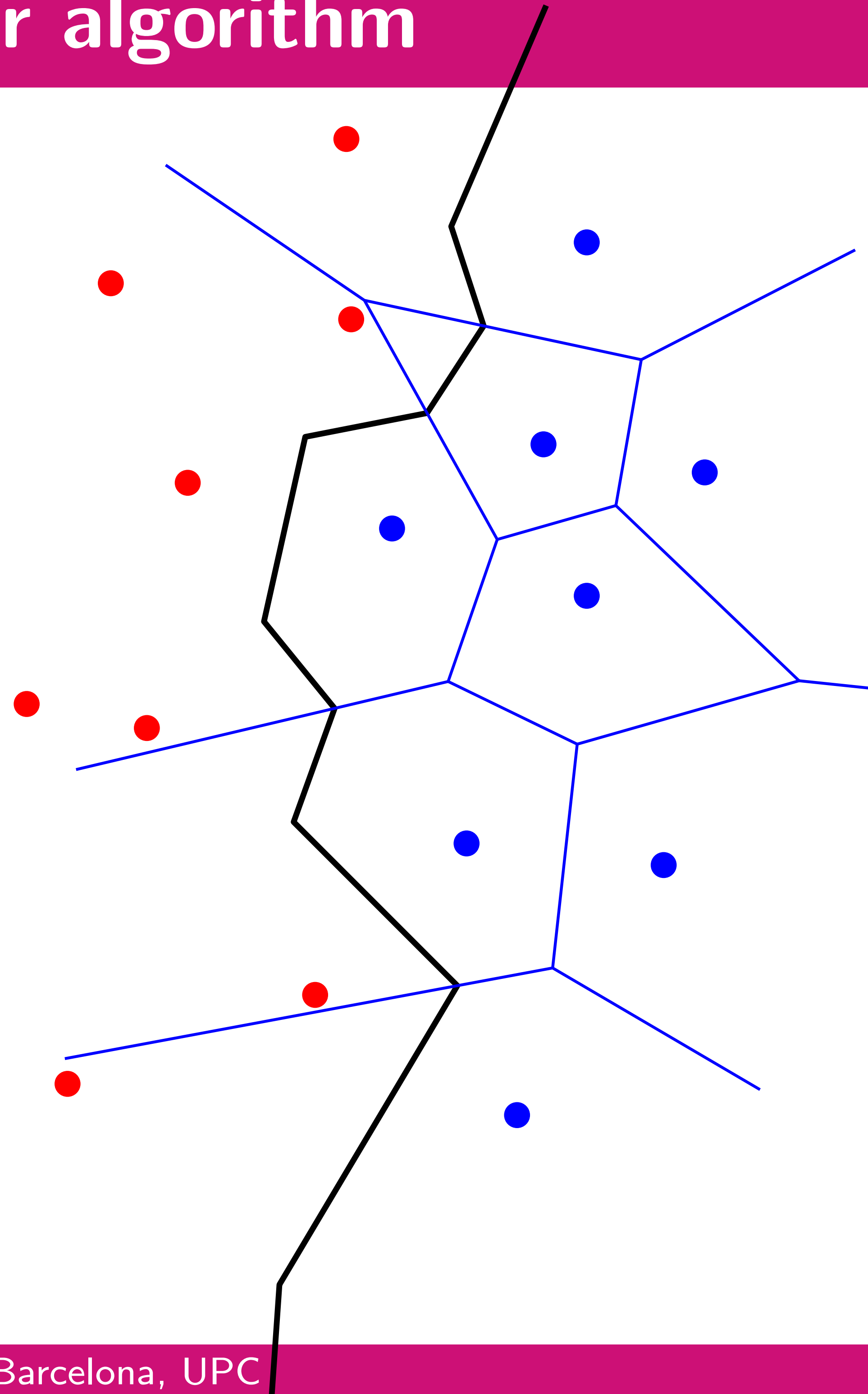
Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

...then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!

In fact, there exists a monotone chain of edges of $Vor(P)$ such that $Vor(P)$ coincides with $Vor(R)$ to the left of the chain, and it coincides with $Vor(B)$ to its right.



Divide and conquer algorithm

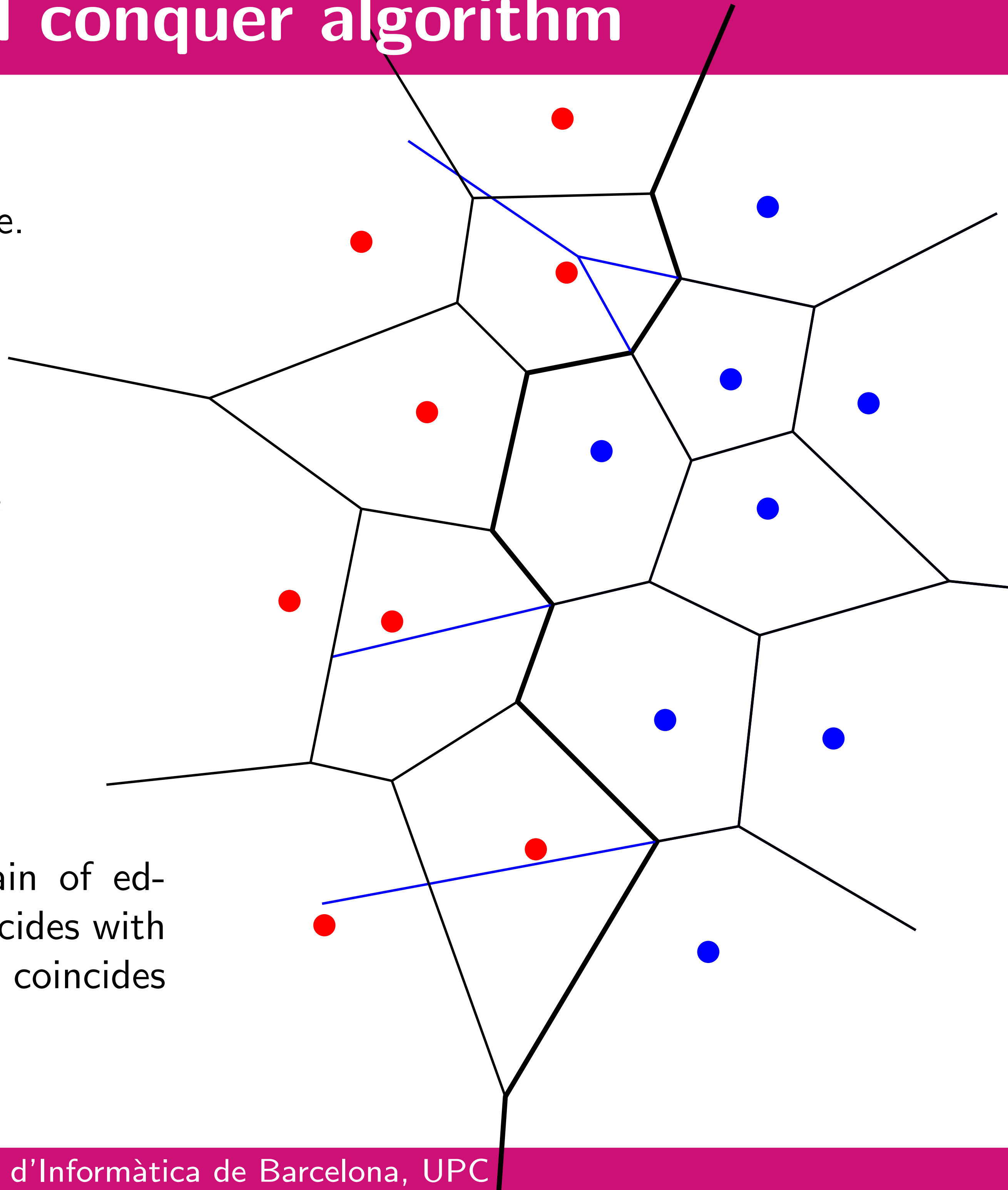
Let P be a set of n points in the plane.

If the points are vertically partitioned into two subsets R and B...

...consider the Voronoi diagram of the sets R and B...

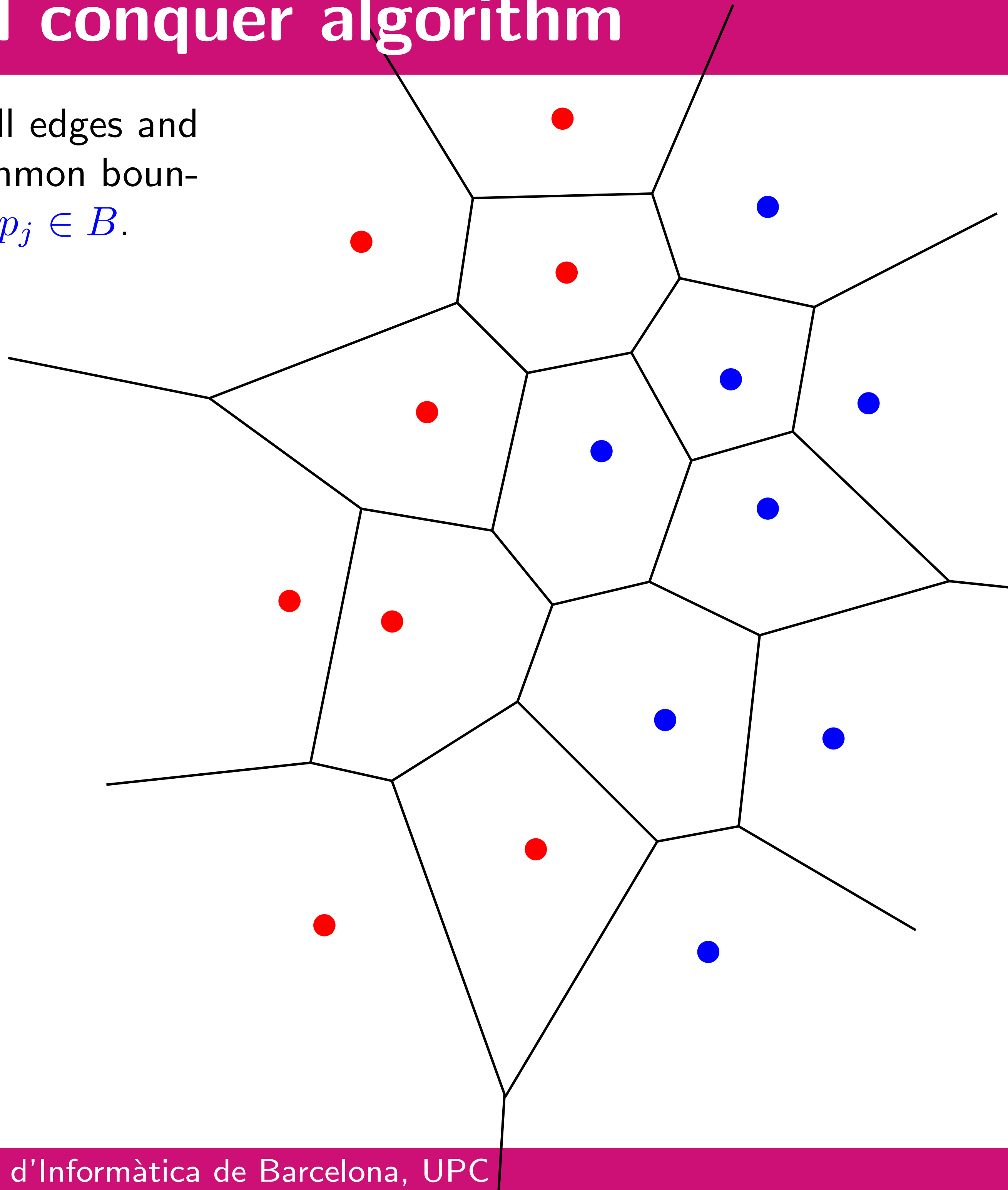
...then the Voronoi diagram of P substantially coincides with the Voronoi diagrams of R and B!

In fact, there exists a monotone chain of edges of $Vor(P)$ such that $Vor(P)$ coincides with $Vor(R)$ to the left of the chain, and it coincides with $Vor(B)$ to its right.



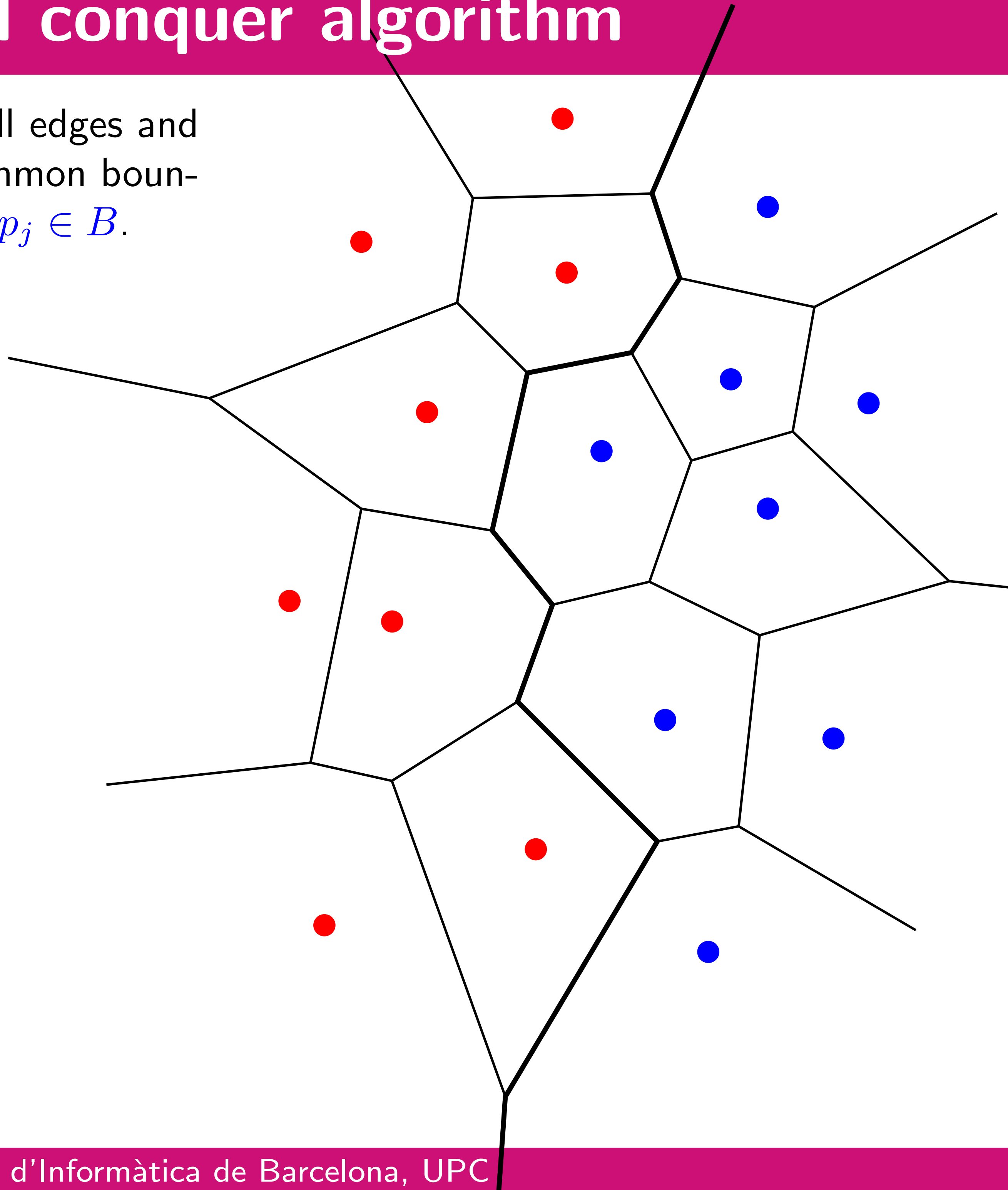
Divide and conquer algorithm

Definition. Let $b(R, B)$ be the set of all edges and vertices of $Vor(P)$ belonging to the common boundary of the regions of some $p_i \in R$ and $p_j \in B$.



Divide and conquer algorithm

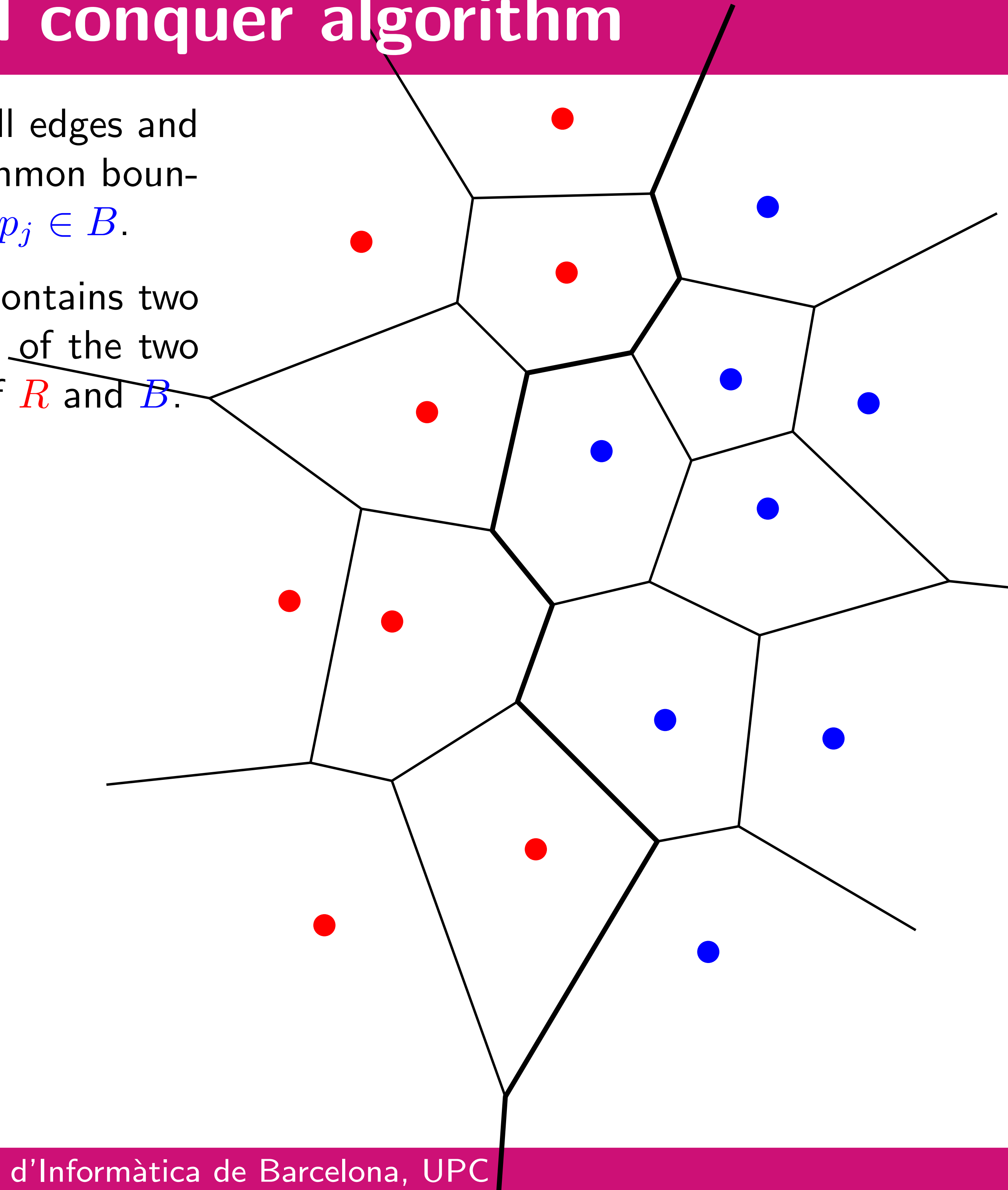
Definition. Let $b(R,B)$ be the set of all edges and vertices of $Vor(P)$ belonging to the common boundary of the regions of some $p_i \in R$ and $p_j \in B$.



Divide and conquer algorithm

Definition. Let $b(R, B)$ be the set of all edges and vertices of $Vor(P)$ belonging to the common boundary of the regions of some $p_i \in R$ and $p_j \in B$.

Observation 1. The bisector $b(R, B)$ contains two half-lines, belonging to the bisectors b_{ij} of the two "bridges" connecting the convex hulls of R and B.

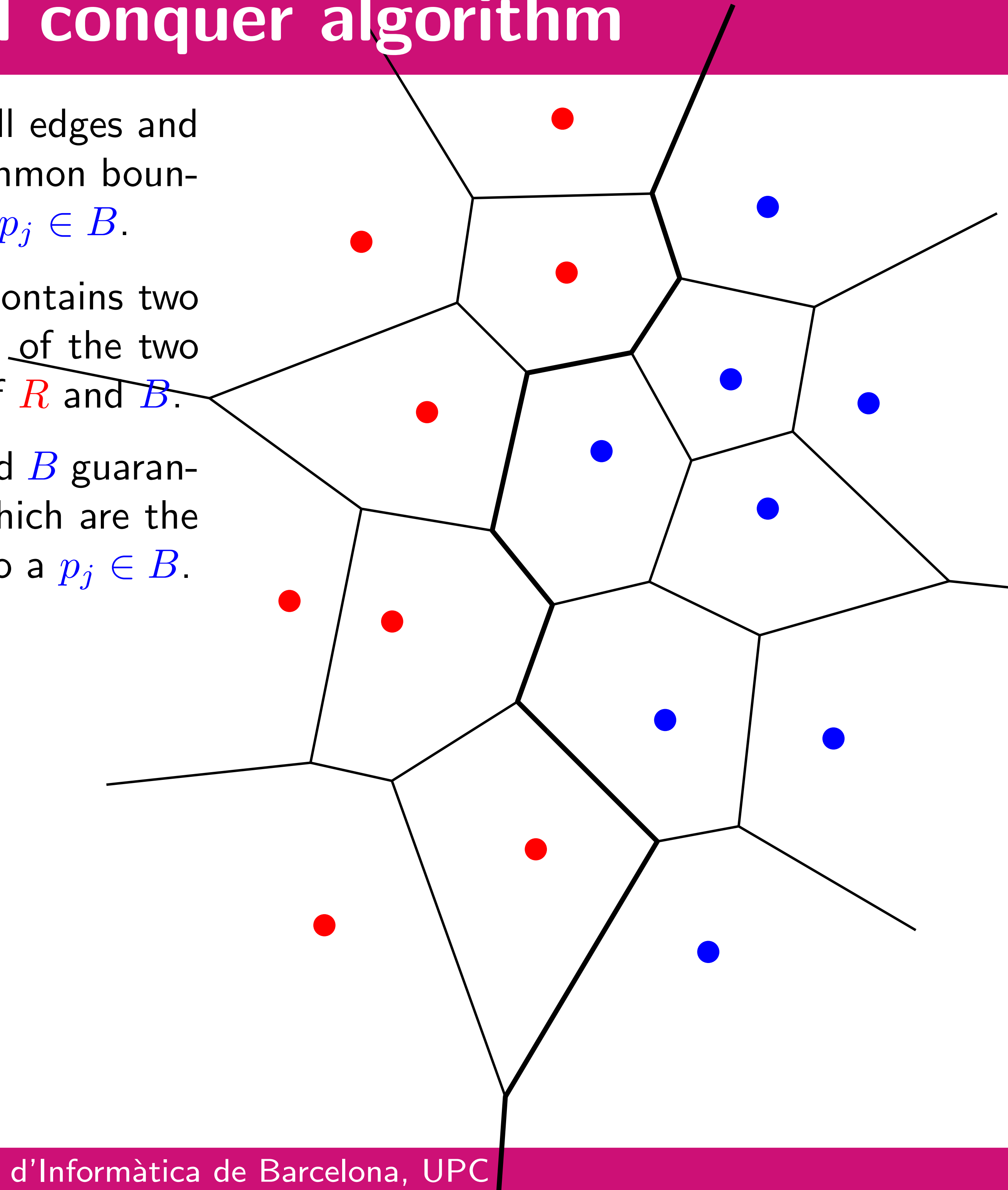


Divide and conquer algorithm

Definition. Let $b(R, B)$ be the set of all edges and vertices of $Vor(P)$ belonging to the common boundary of the regions of some $p_i \in R$ and $p_j \in B$.

Observation 1. The bisector $b(R, B)$ contains two half-lines, belonging to the bisectors b_{ij} of the two "bridges" connecting the convex hulls of R and B.

Proof. The vertical separation of R and B guarantees the existence of the "bridges", which are the edges of $ch(P)$ connecting a $p_i \in R$ to a $p_j \in B$.

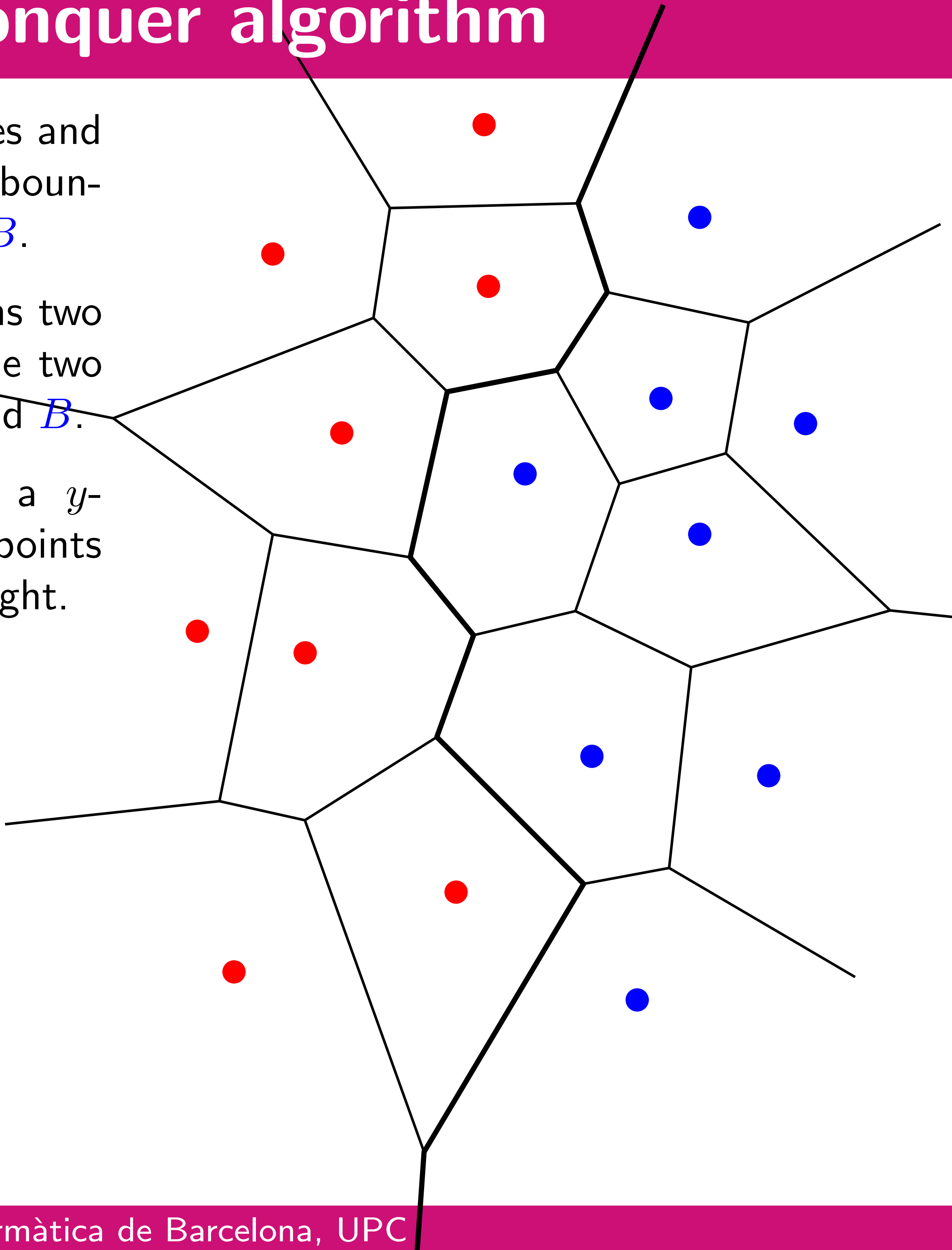


Divide and conquer algorithm

Definition. Let $b(R, B)$ be the set of all edges and vertices of $Vor(P)$ belonging to the common boundary of the regions of some $p_i \in R$ and $p_j \in B$.

Observation 1. The bisector $b(R, B)$ contains two half-lines, belonging to the bisectors b_{ij} of the two "bridges" connecting the convex hulls of R and B.

Observation 2. The bisector $b(R, B)$ is a y-monotone chain leaving the regions of the points $p_i \in R$ to its left and those of $p_j \in B$ to its right.



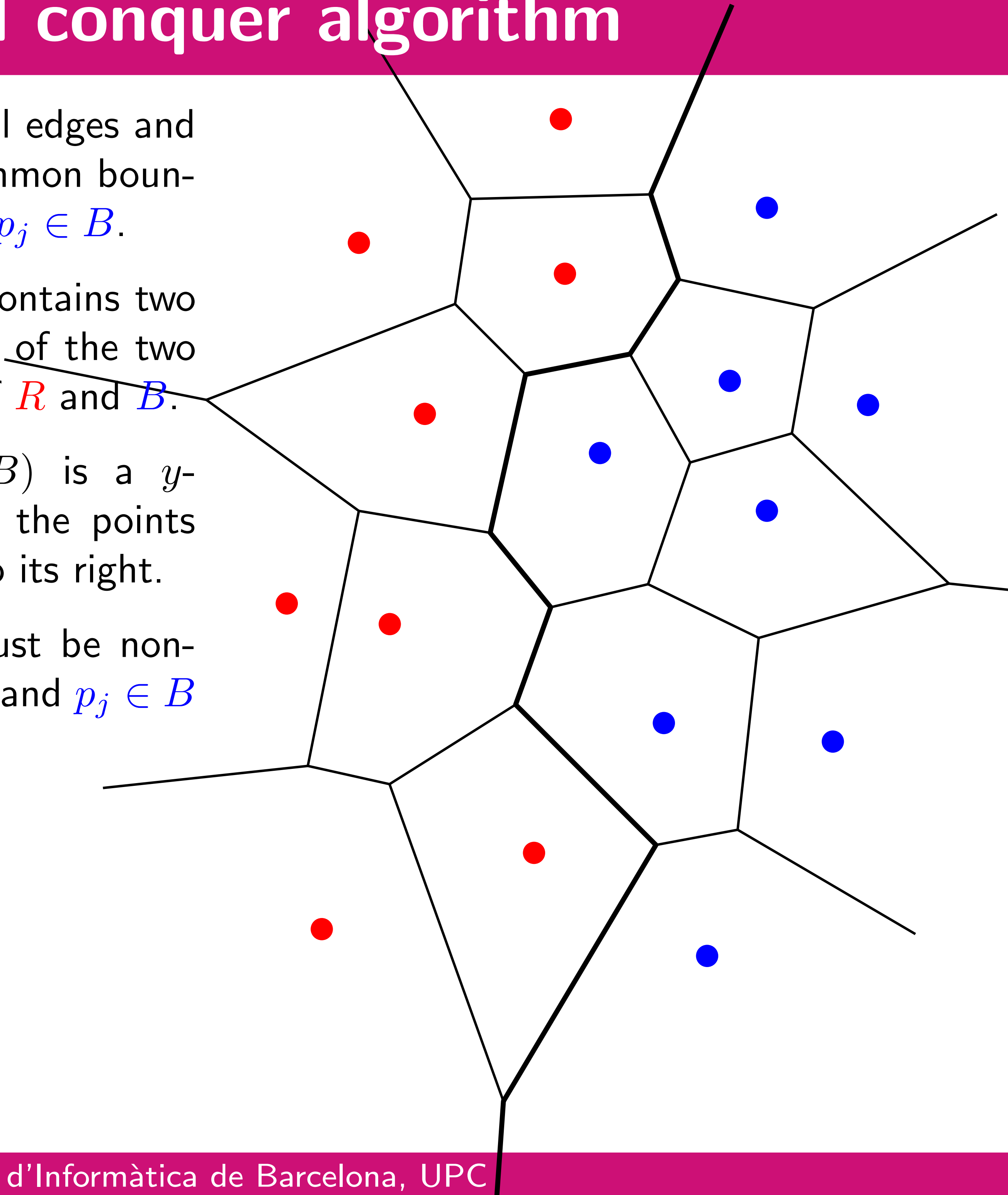
Divide and conquer algorithm

Definition. Let $b(R, B)$ be the set of all edges and vertices of $Vor(P)$ belonging to the common boundary of the regions of some $p_i \in R$ and $p_j \in B$.

Observation 1. The bisector $b(R, B)$ contains two half-lines, belonging to the bisectors b_{ij} of the two "bridges" connecting the convex hulls of R and B.

Observation 2. The bisector $b(R, B)$ is a y-monotone chain leaving the regions of the points $p_i \in R$ to its left and those of $p_j \in B$ to its right.

Proof. Every edge e_{ij} of $b(R, B)$ must be non-horizontal, and leave $p_i \in R$ to its left and $p_j \in B$ to its right.



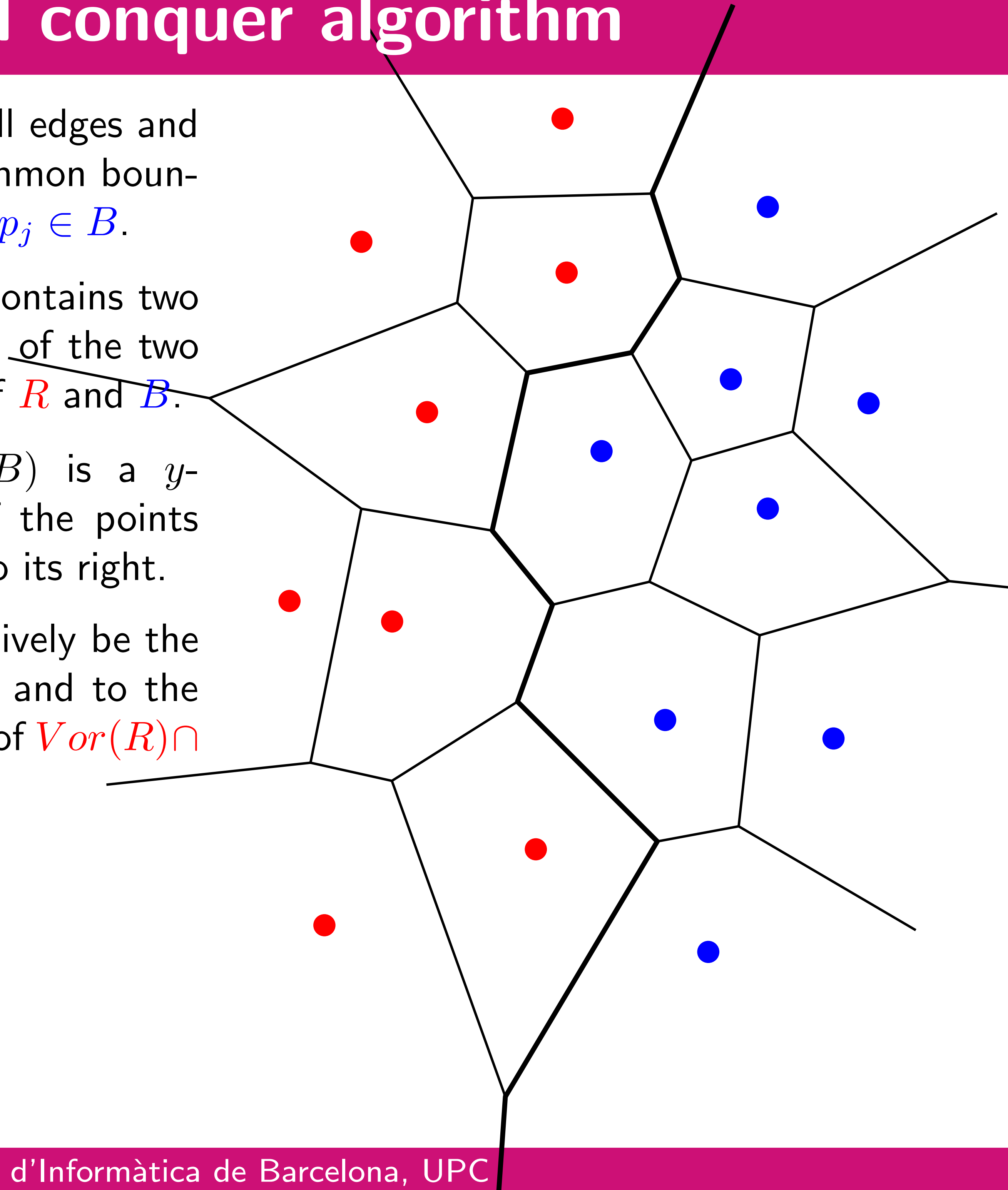
Divide and conquer algorithm

Definition. Let $b(R, B)$ be the set of all edges and vertices of $Vor(P)$ belonging to the common boundary of the regions of some $p_i \in R$ and $p_j \in B$.

Observation 1. The bisector $b(R, B)$ contains two half-lines, belonging to the bisectors b_{ij} of the two "bridges" connecting the convex hulls of R and B.

Observation 2. The bisector $b(R, B)$ is a y-monotone chain leaving the regions of the points $p_i \in R$ to its left and those of $p_j \in B$ to its right.

Observation 3. Let π_R and π_B respectively be the regions of the plane located to the left and to the right of $b(R, B)$. Then $Vor(P)$ consists of $Vor(R) \cap \pi_R$, $Vor(B) \cap \pi_B$ and $b(R, B)$.



Divide and conquer algorithm

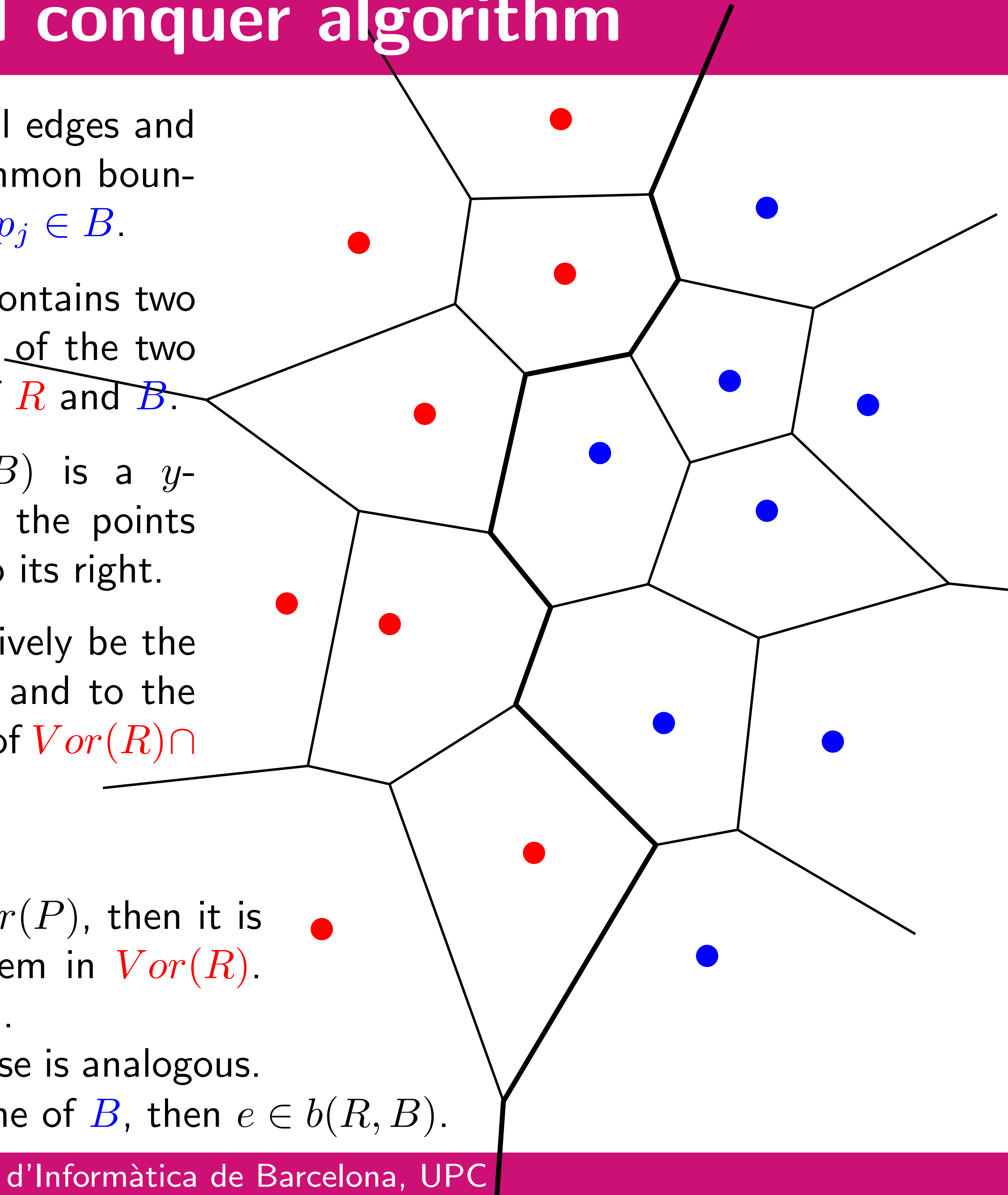
Definition. Let $b(R, B)$ be the set of all edges and vertices of $Vor(P)$ belonging to the common boundary of the regions of some $p_i \in R$ and $p_j \in B$.

Observation 1. The bisector $b(R, B)$ contains two half-lines, belonging to the bisectors b_{ij} of the two "bridges" connecting the convex hulls of R and B.

Observation 2. The bisector $b(R, B)$ is a y-monotone chain leaving the regions of the points $p_i \in R$ to its left and those of $p_j \in B$ to its right.

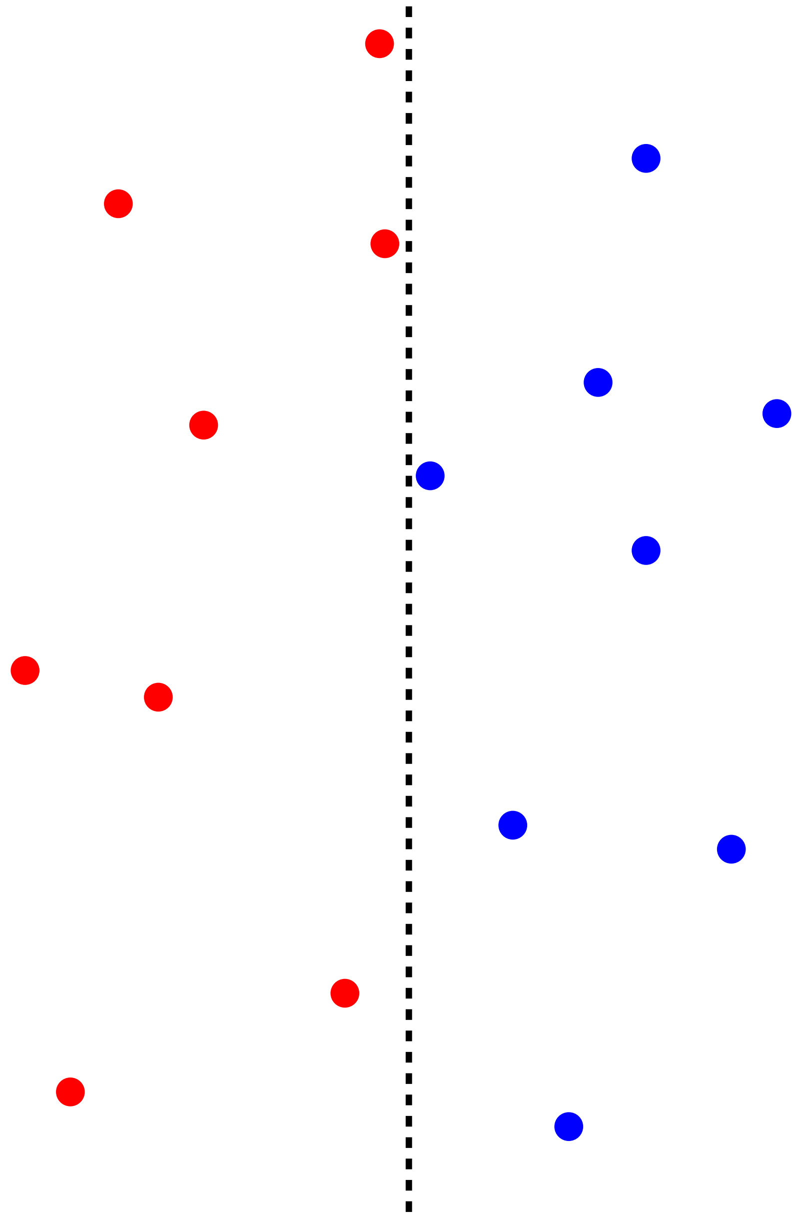
Observation 3. Let π_R and π_B respectively be the regions of the plane located to the left and to the right of $b(R, B)$. Then $Vor(P)$ consists of $Vor(R) \cap \pi_R$, $Vor(B) \cap \pi_B$ and $b(R, B)$.

Proof. Let e be an edge of $Vor(P)$:
- If e separates two points of R in $Vor(P)$, then it is (a portion of) the edge separating them in $Vor(R)$. Due to Obs. 2, e cannot belong to π_B.
- If e separates two points of B, the case is analogous.
- If e separates one point of R from one of B, then $e \in b(R, B)$.



Divide and conquer algorithm

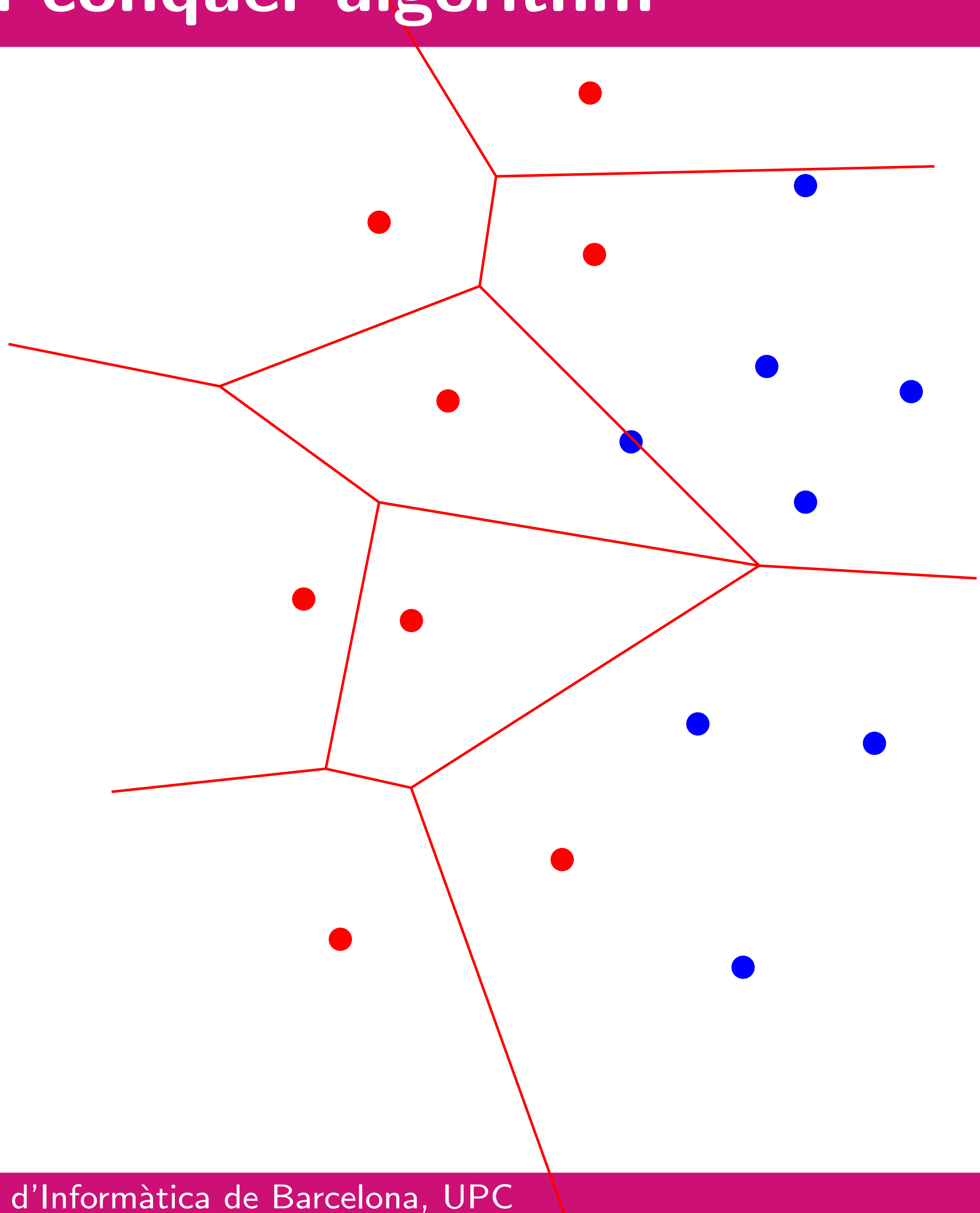
1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.



Divide and conquer algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

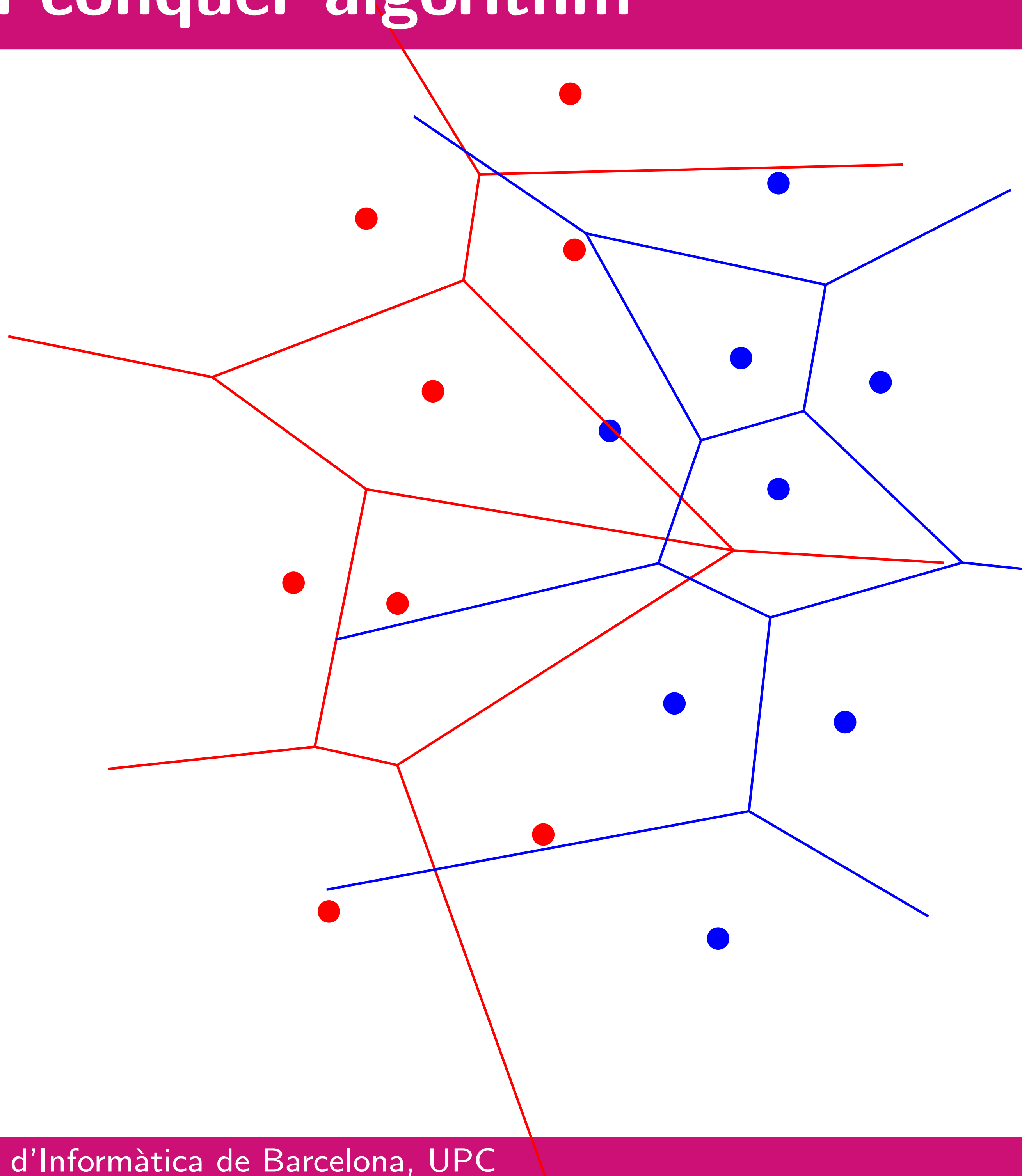
2. Recursively compute $Vor(R)$ and $Vor(B)$.



Divide and conquer algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

2. Recursively compute $Vor(R)$ and $Vor(B)$.

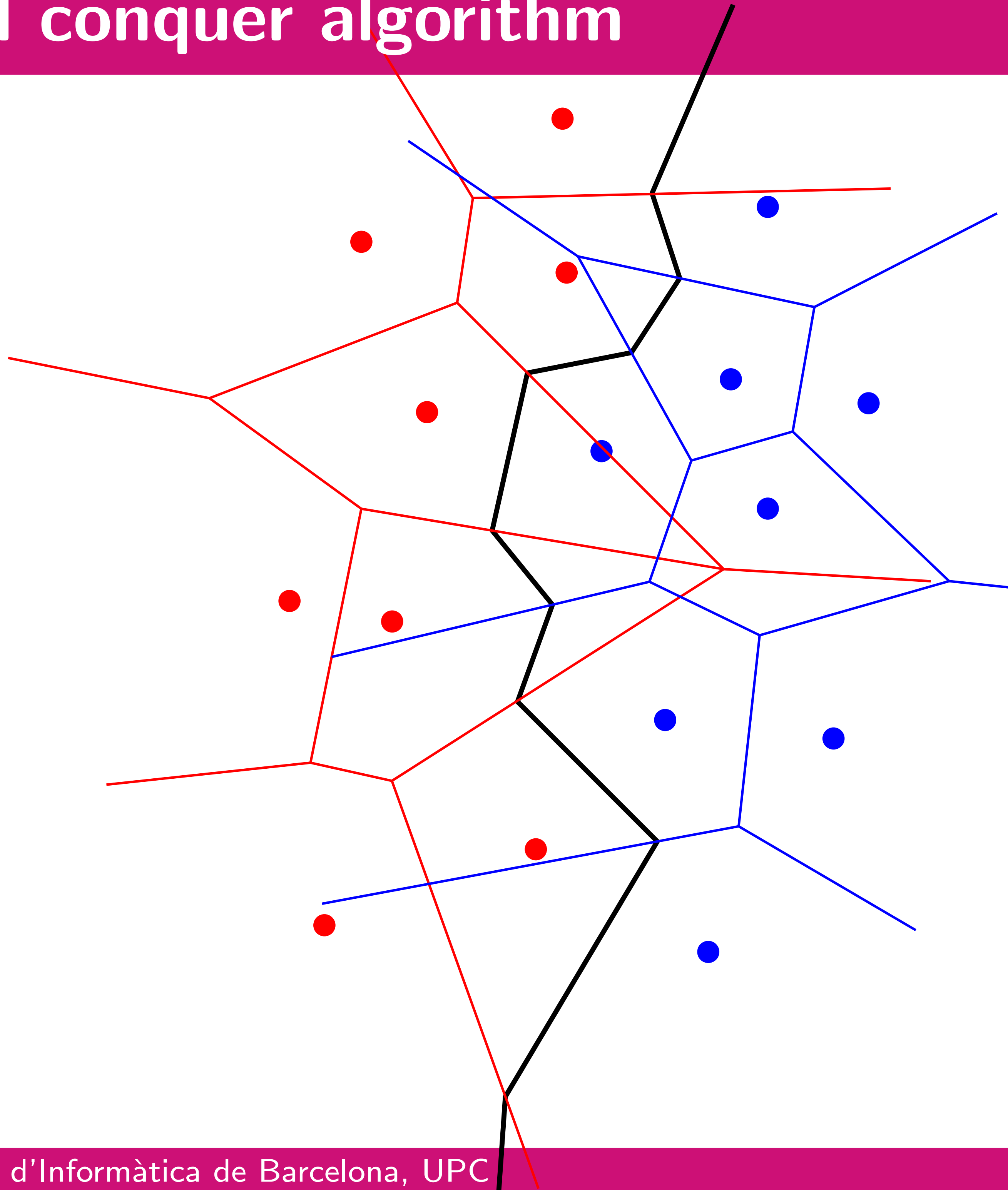


Divide and conquer algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

2. Recursively compute $Vor(R)$ and $Vor(B)$.

3. Compute the separating chain.



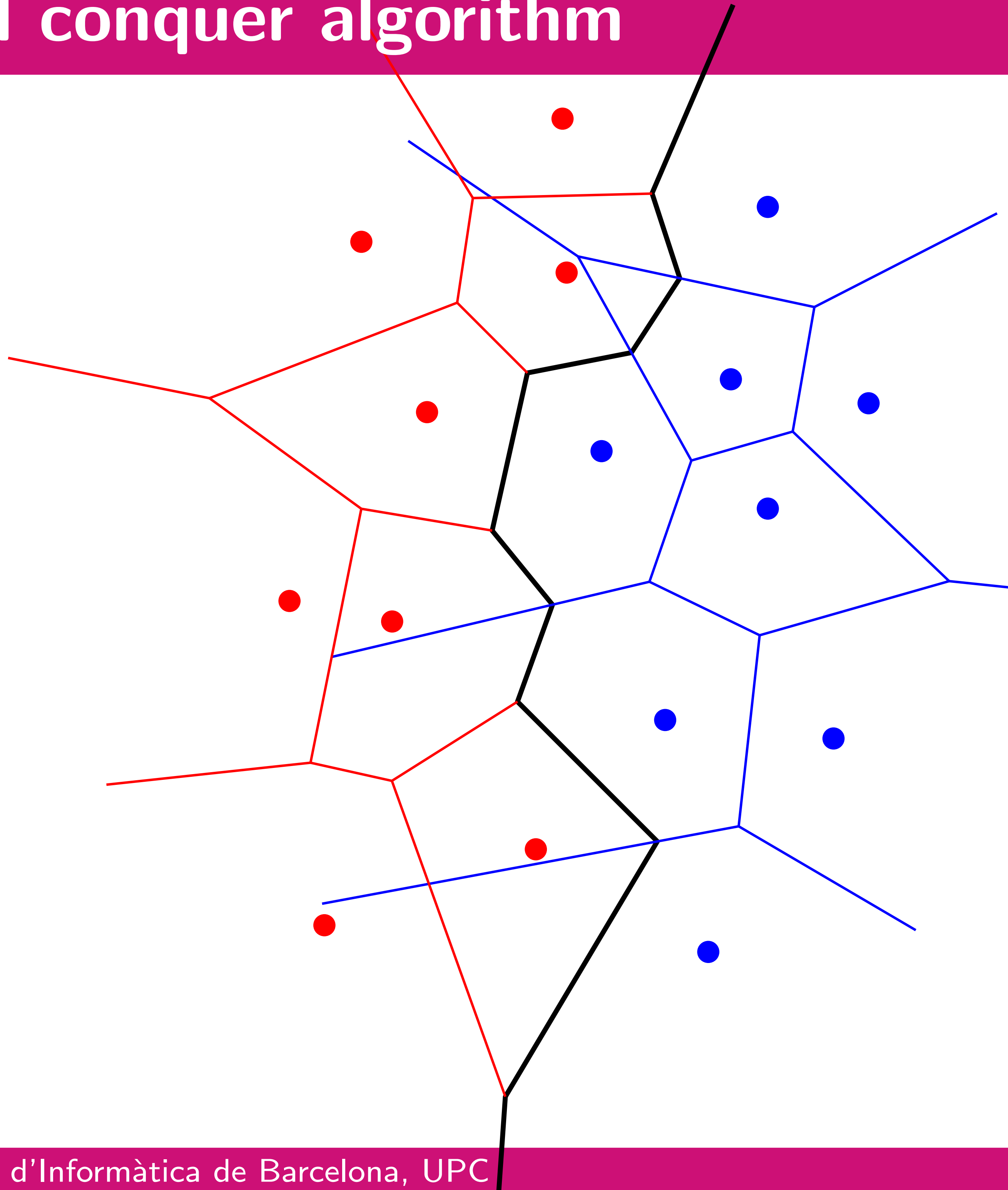
Divide and conquer algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

2. Recursively compute $Vor(R)$ and $Vor(B)$.

3. Compute the separating chain.

4. Prune the portion of $Vor(R)$ lying to the right of the chain and the portion of $Vor(B)$ lying to its left.



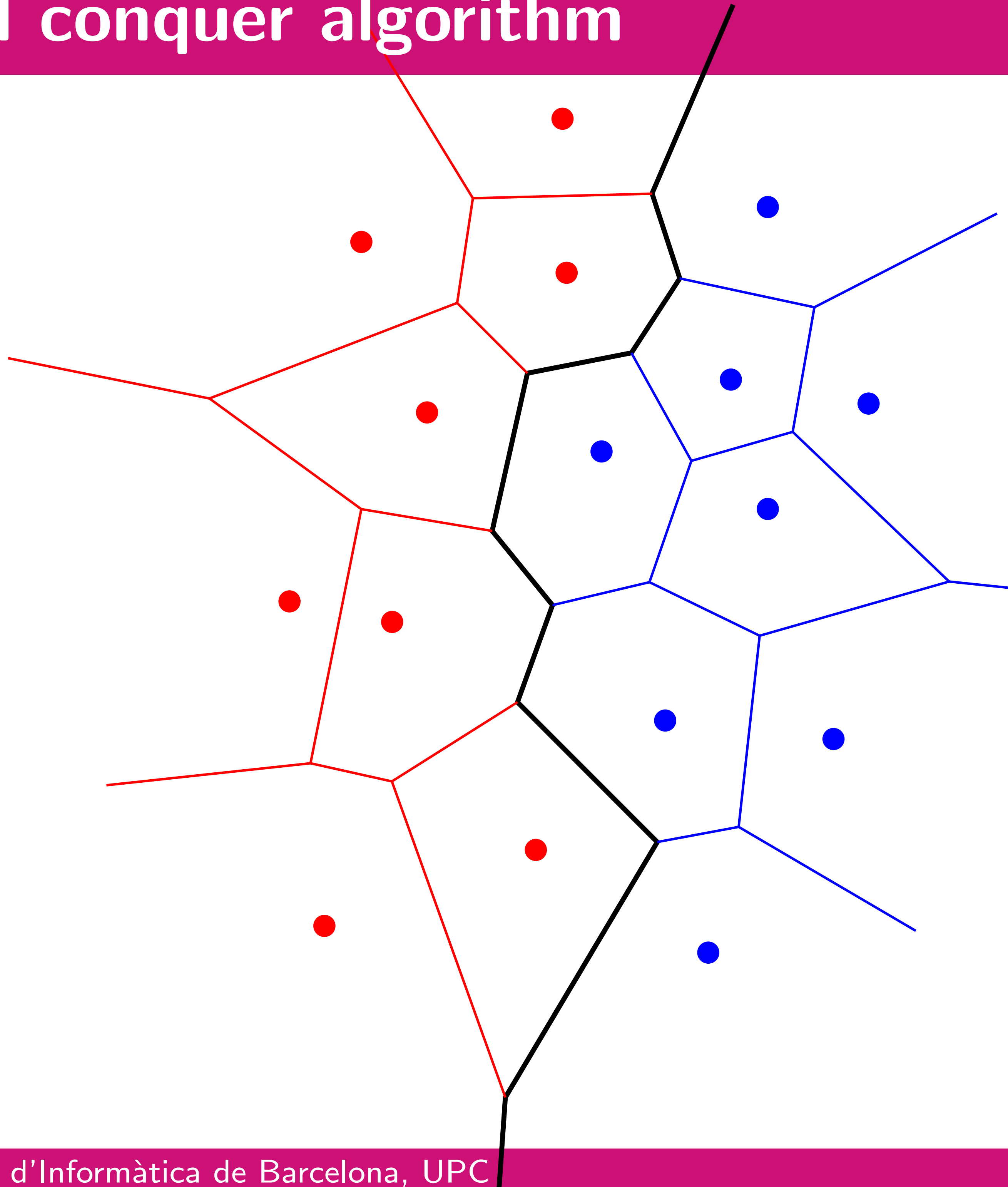
Divide and conquer algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

2. Recursively compute $Vor(R)$ and $Vor(B)$.

3. Compute the separating chain.

4. Prune the portion of $Vor(R)$ lying to the right of the chain and the portion of $Vor(B)$ lying to its left.



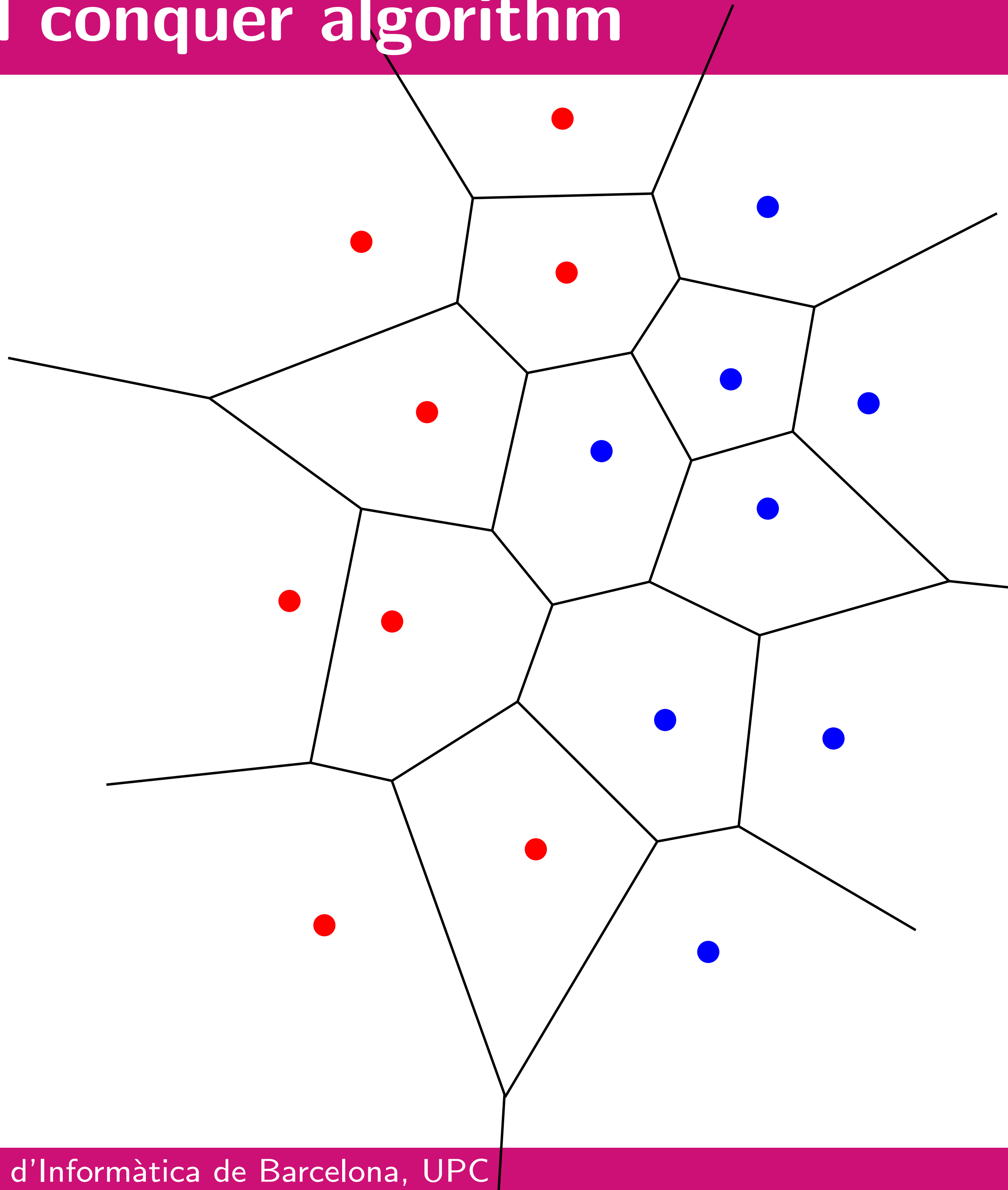
Divide and conquer algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

2. Recursively compute $Vor(R)$ and $Vor(B)$.

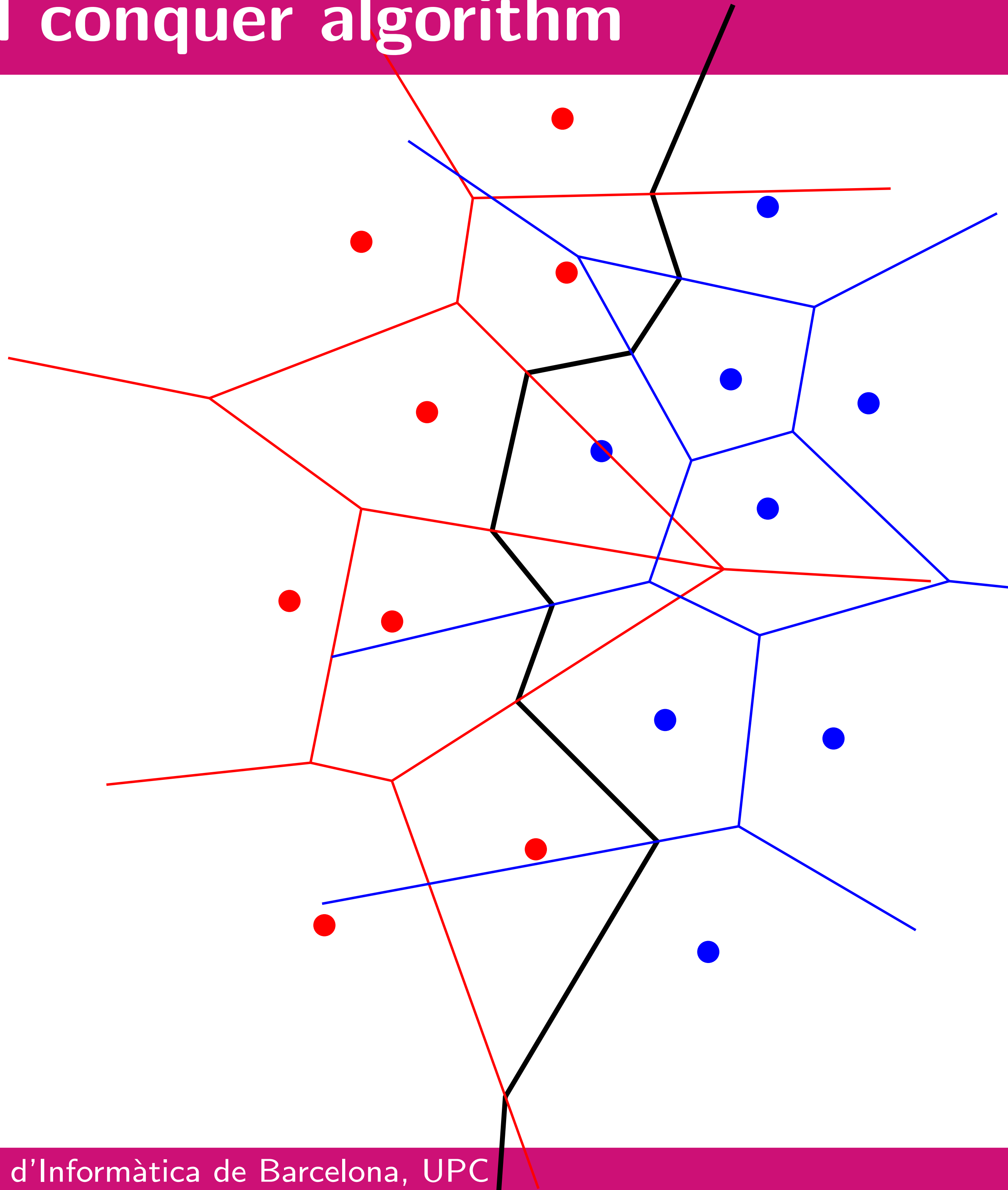
3. Compute the separating chain.

4. Prune the portion of $Vor(R)$ lying to the right of the chain and the portion of $Vor(B)$ lying to its left.



Divide and conquer algorithm

How to compute the chain?

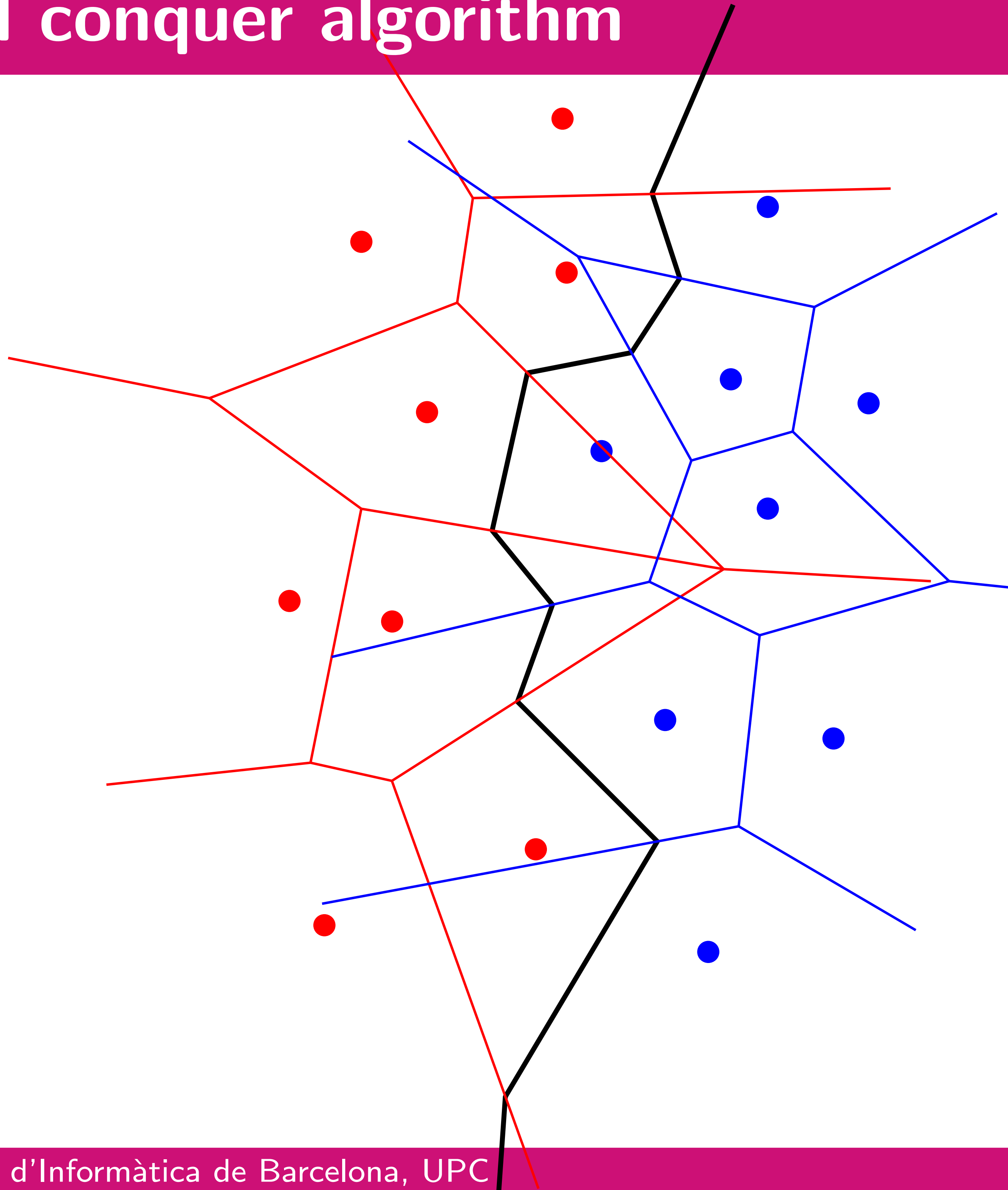


Divide and conquer algorithm

How to compute the chain?

Initialization

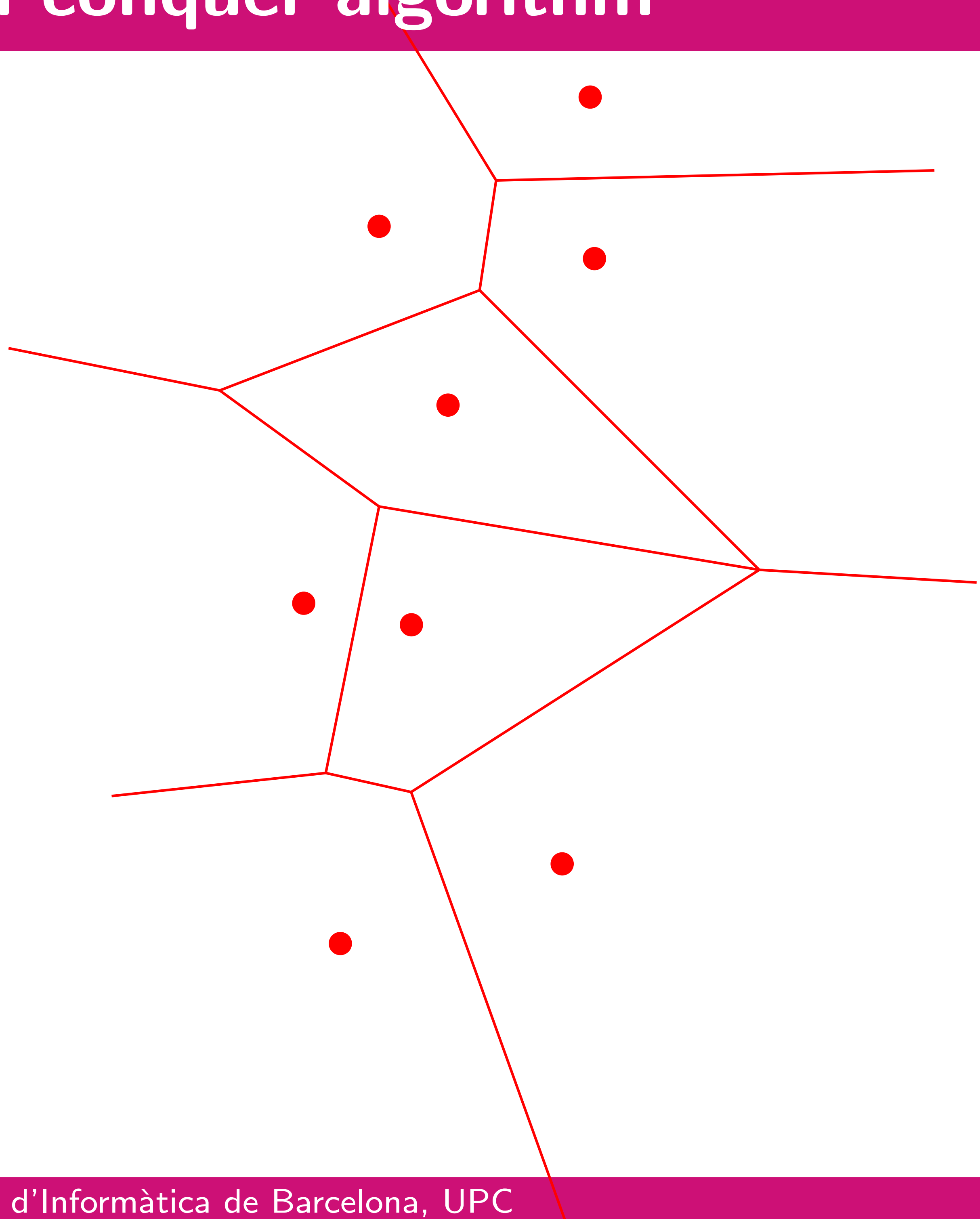
Find the two halflines



How to compute the chain?

Initialization

Find the two halflines

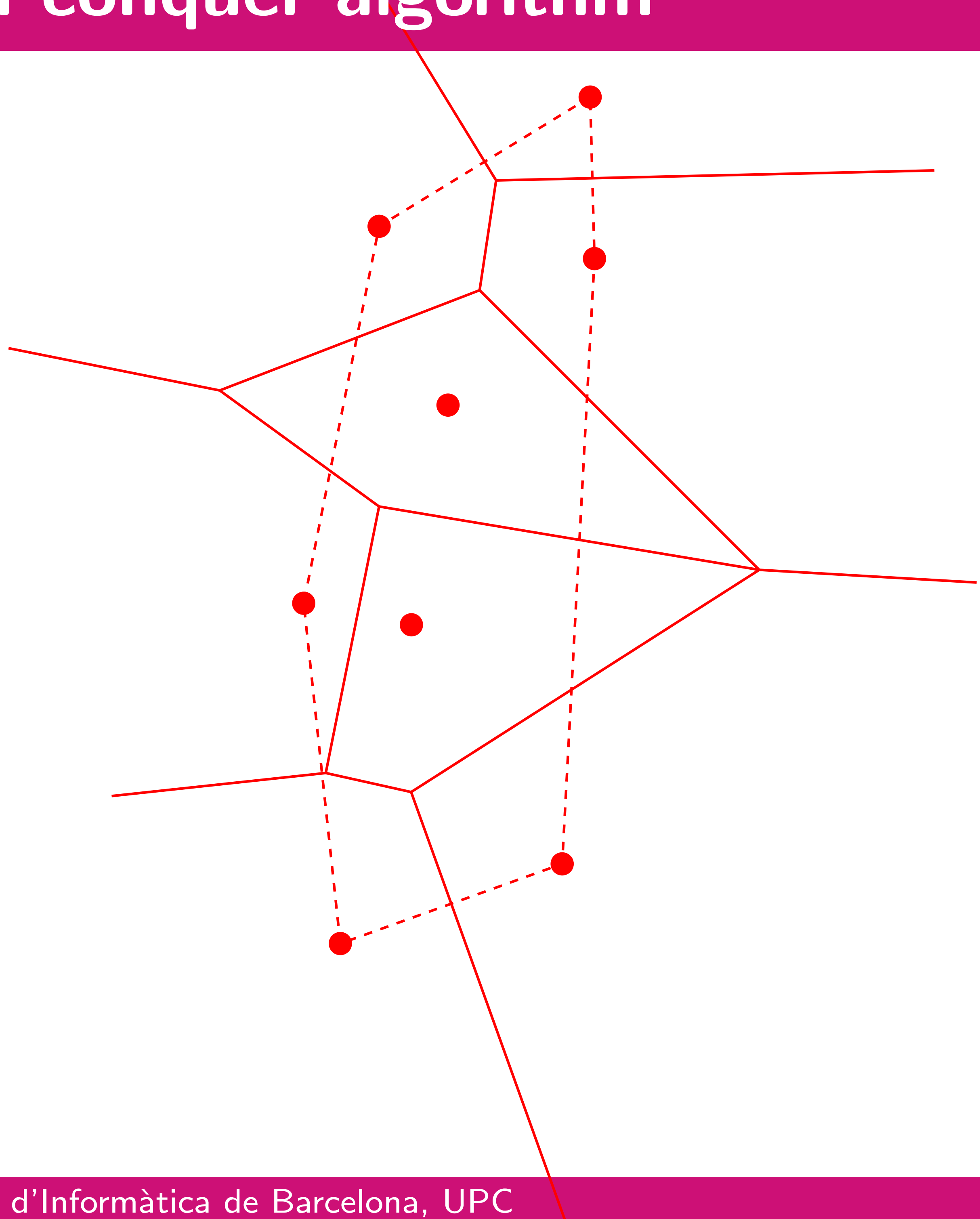


Divide and conquer algorithm

How to compute the chain?

Initialization

Find the two halflines

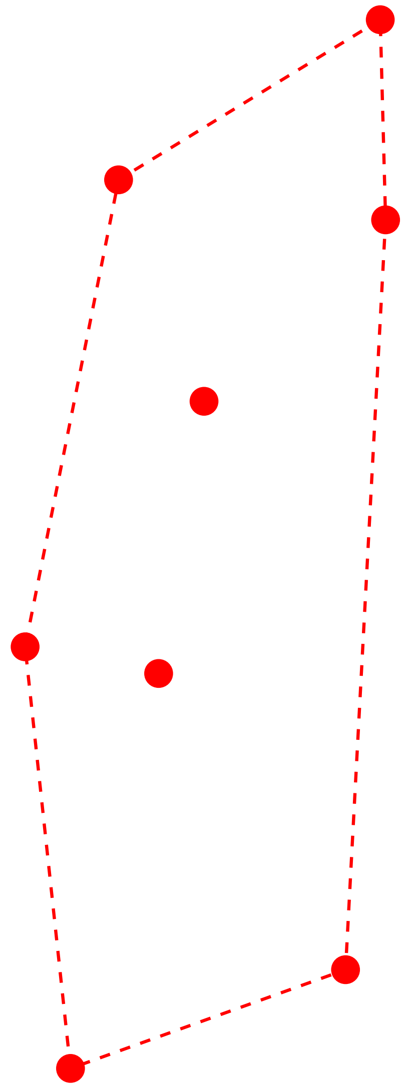


Divide and conquer algorithm

How to compute the chain?

Initialization

Find the two halflines

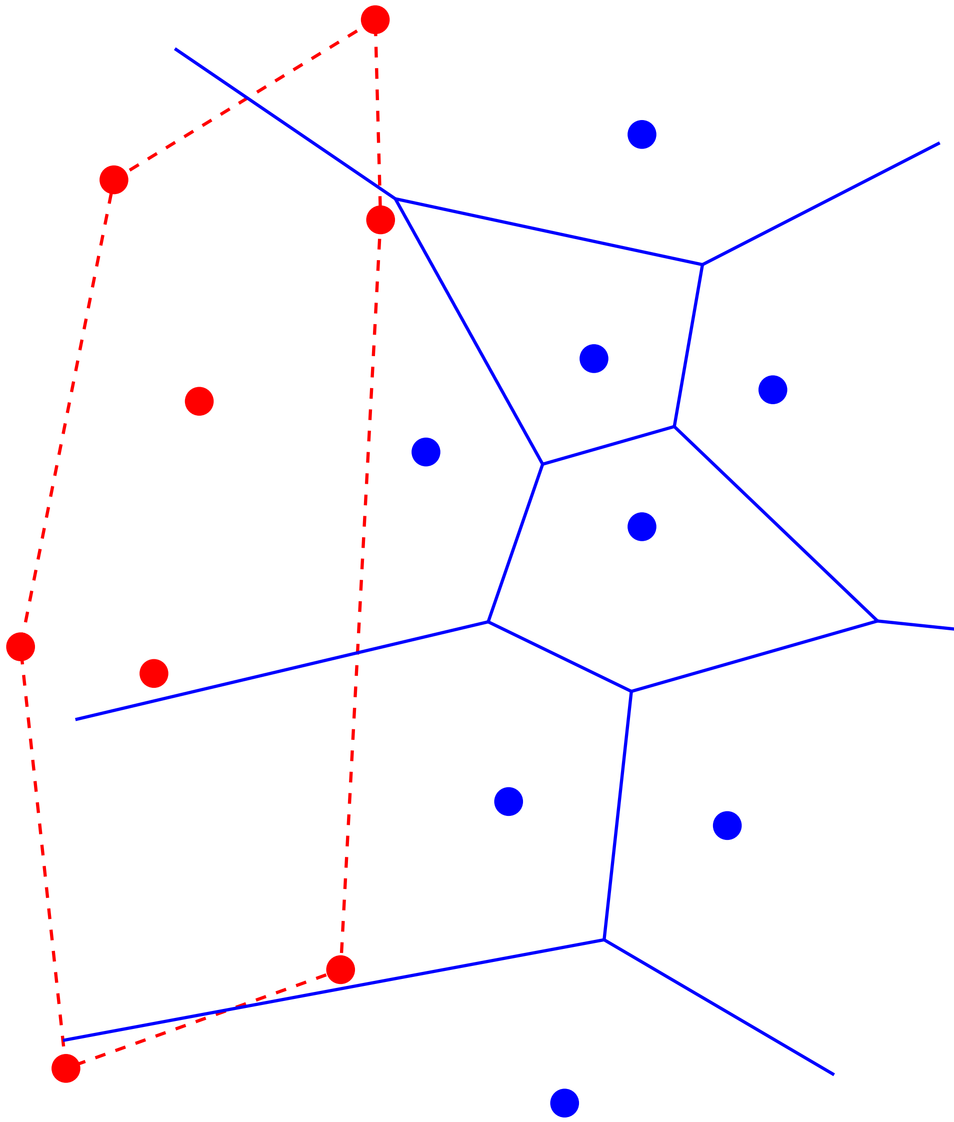


Divide and conquer algorithm

How to compute the chain?

Initialization

Find the two halflines

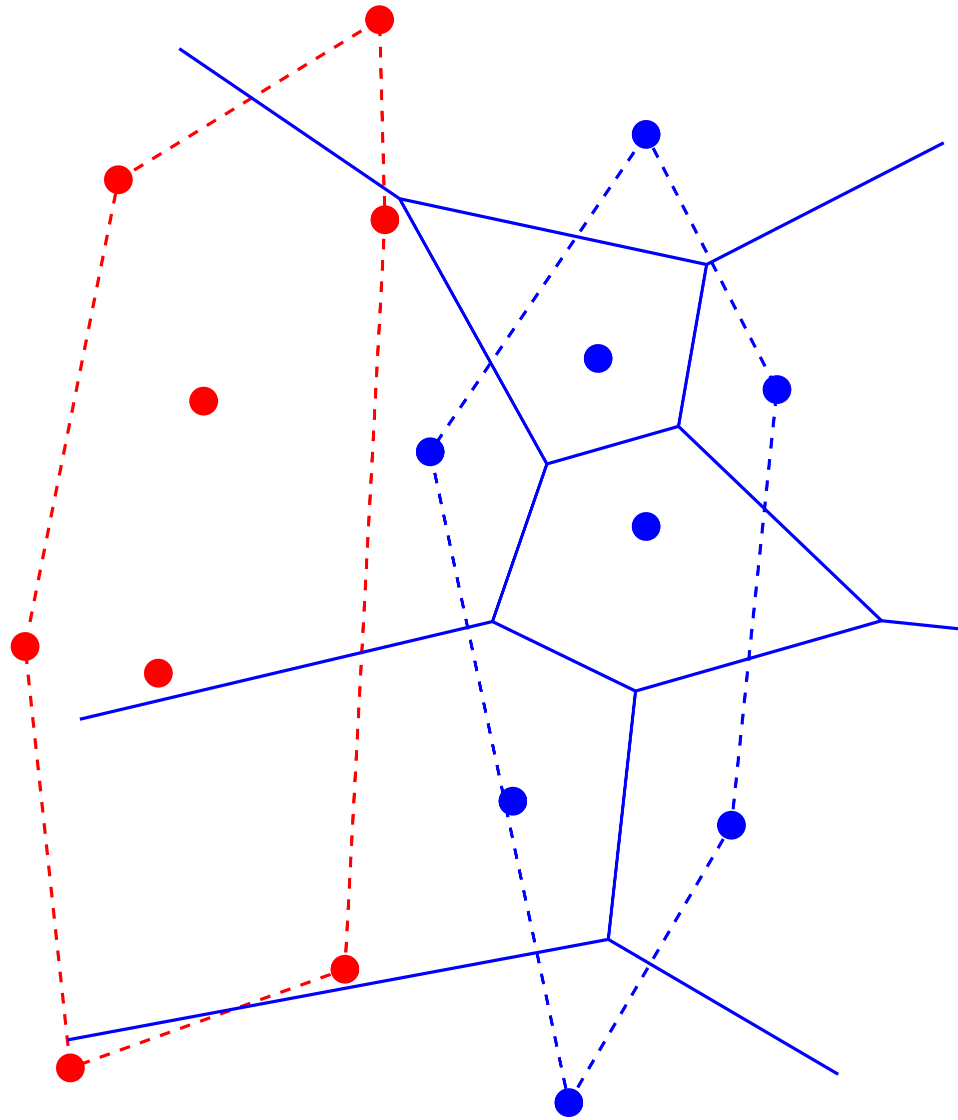


Divide and conquer algorithm

How to compute the chain?

Initialization

Find the two halflines

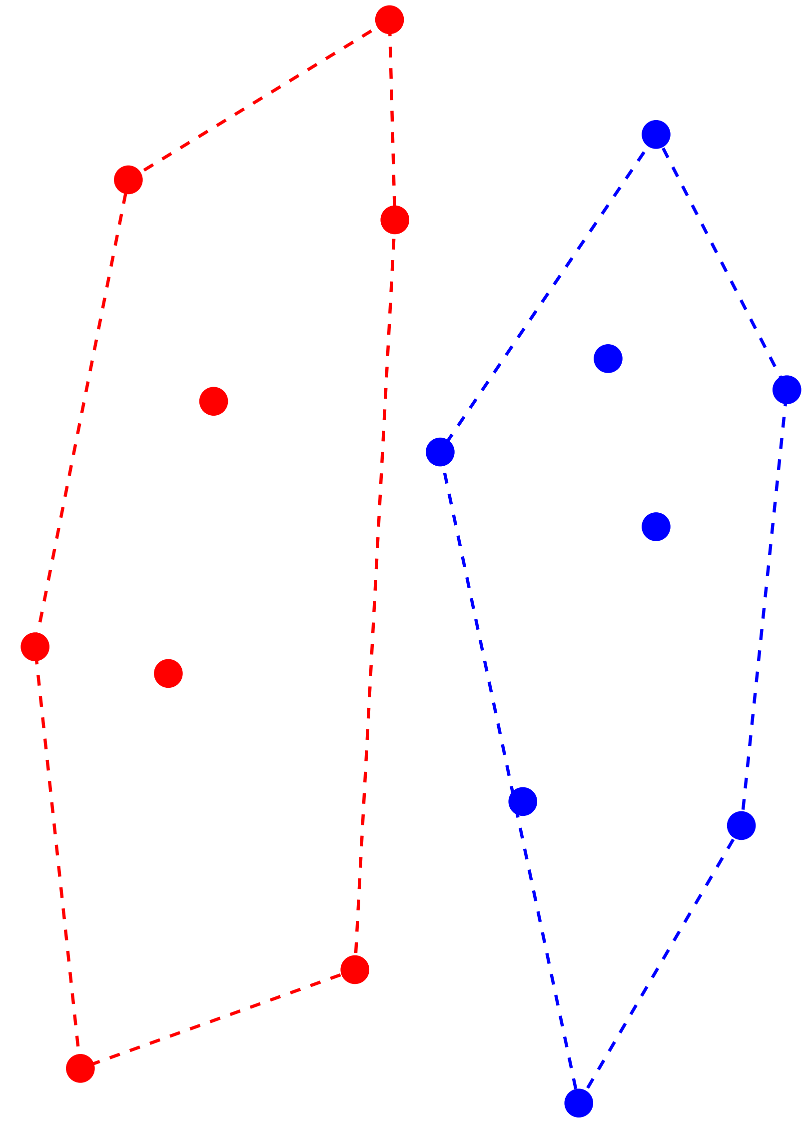


Divide and conquer algorithm

How to compute the chain?

Initialization

Find the two halflines

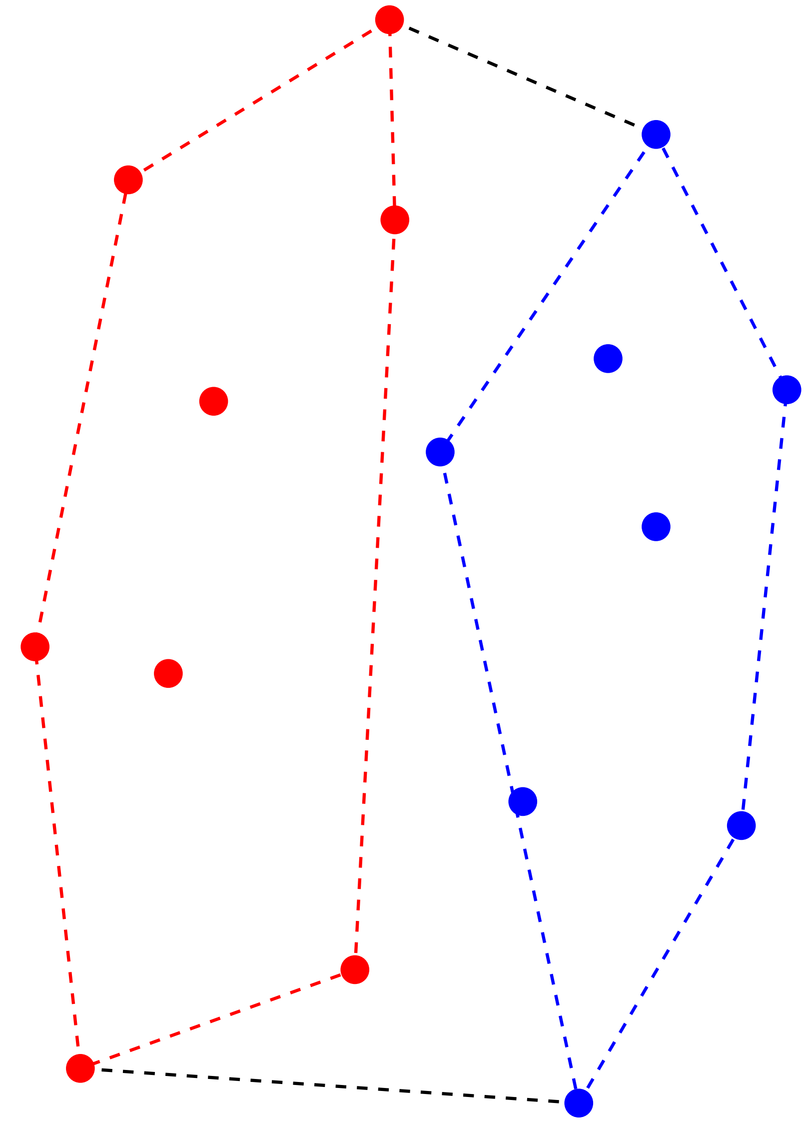


Divide and conquer algorithm

How to compute the chain?

Initialization

Find the two halflines

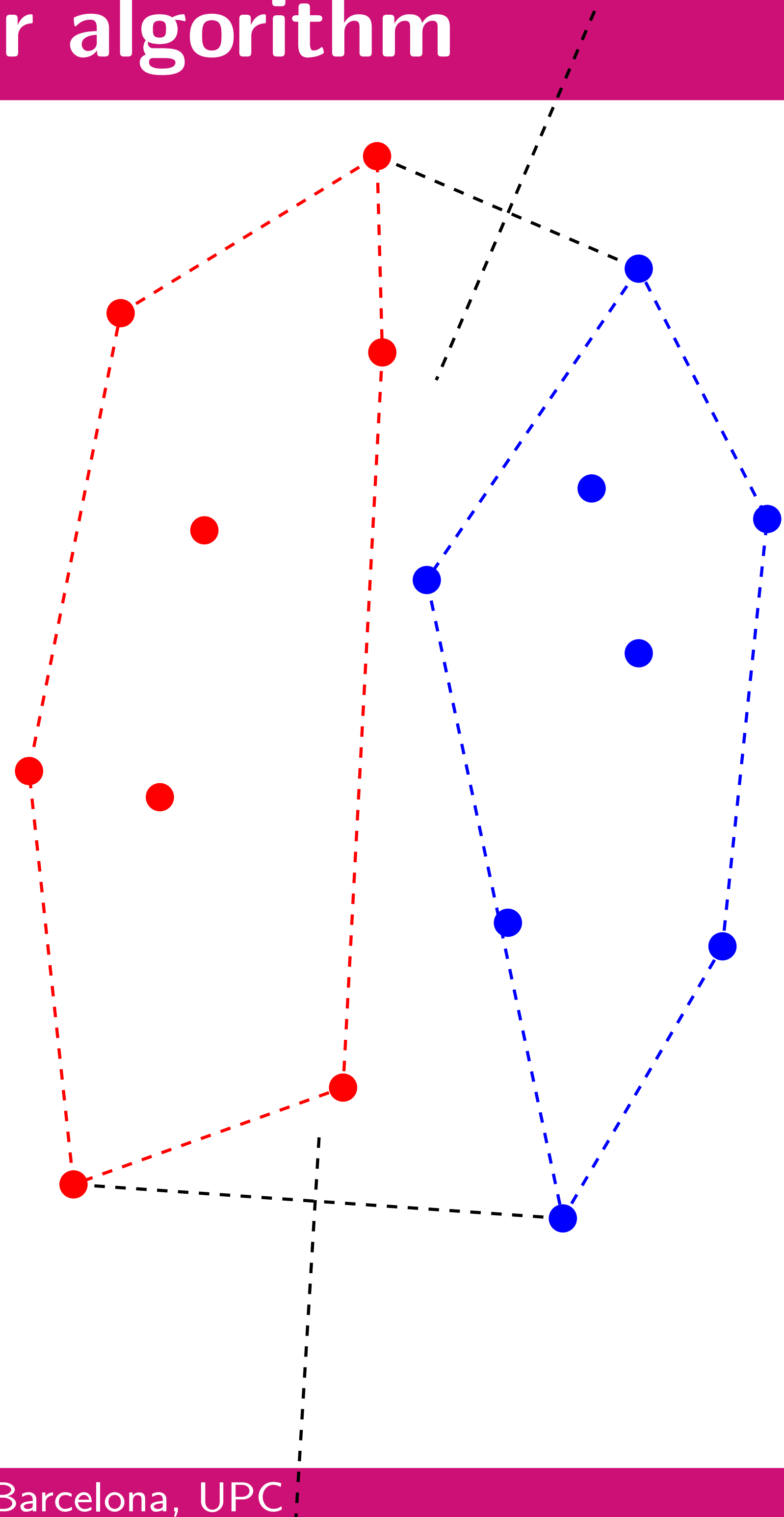


Divide and conquer algorithm

How to compute the chain?

Initialization

Find the two halflines



Divide and conquer algorithm

How to compute the chain?

Initialization

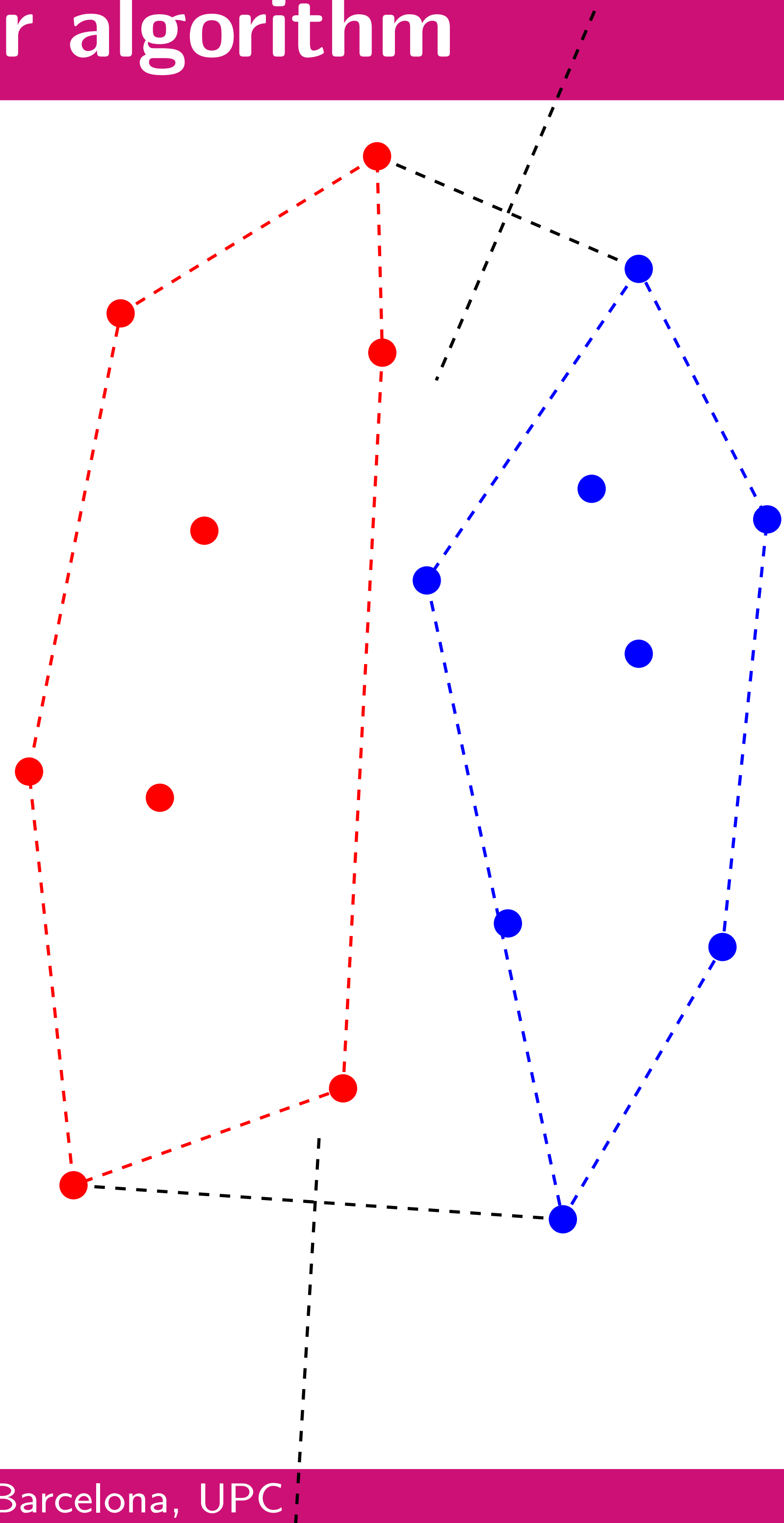
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

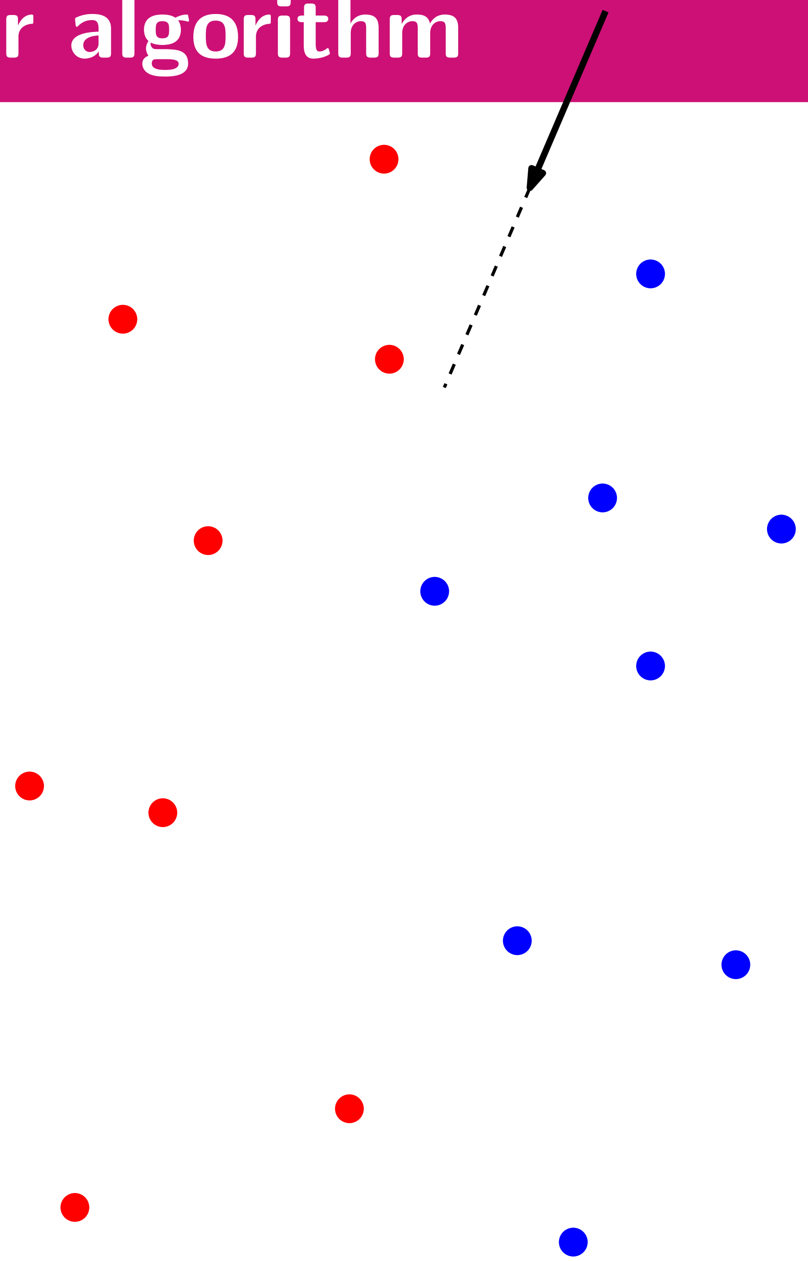
Find the two halflines

Advance

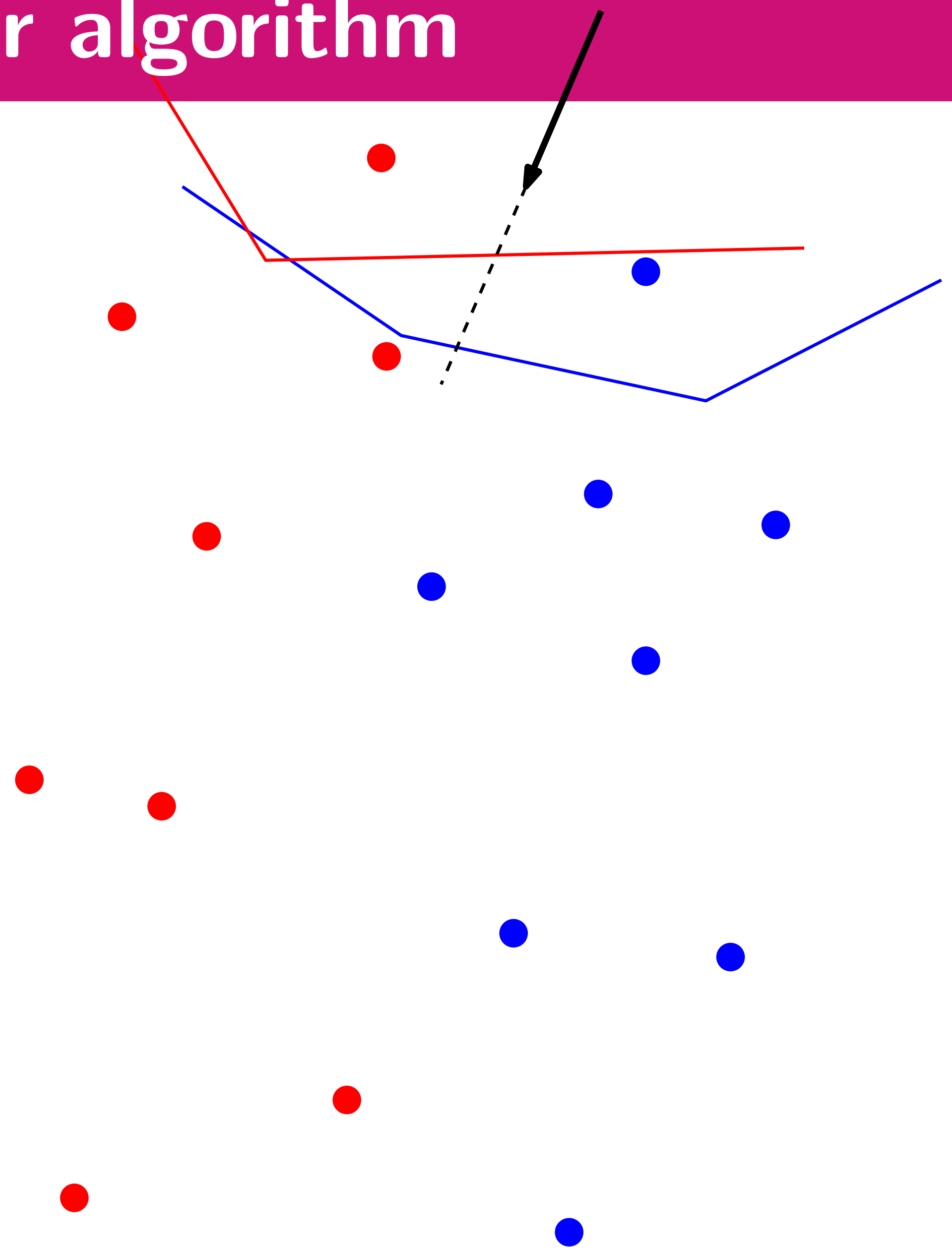
Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm



How to compute the chain?

Initialization

Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge

Divide and conquer algorithm

How to compute the chain?

Initialization

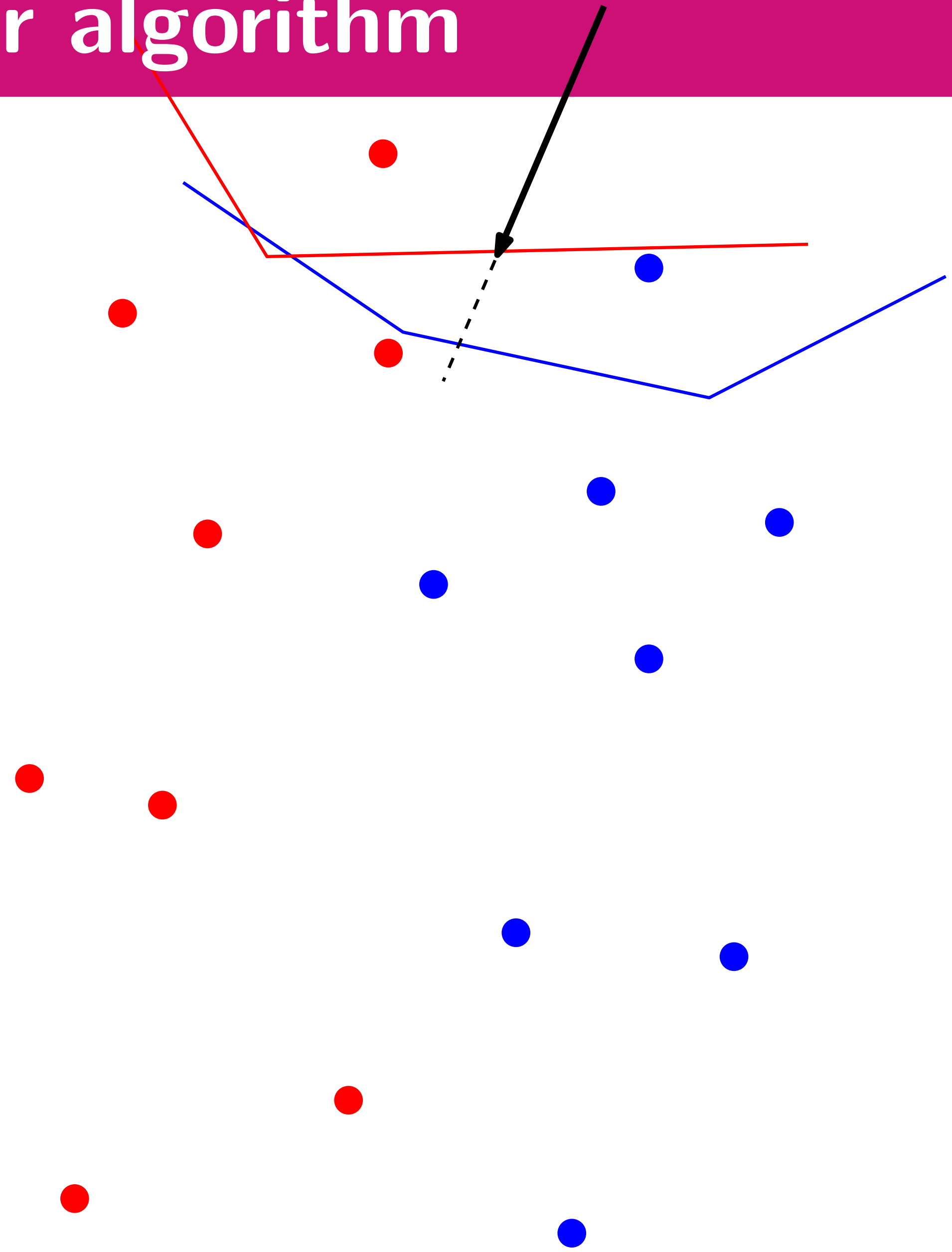
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

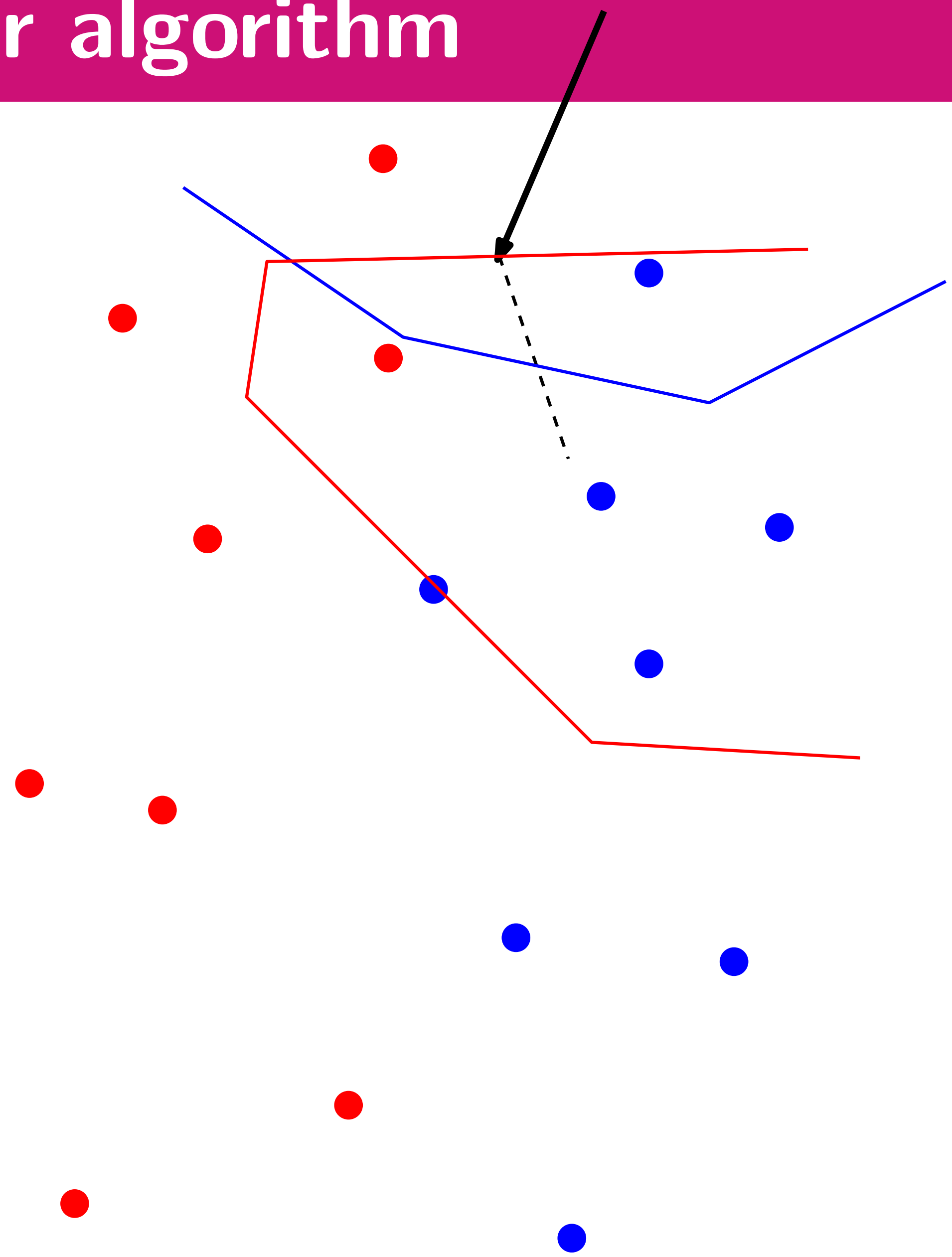
Find the two halflines

Advance

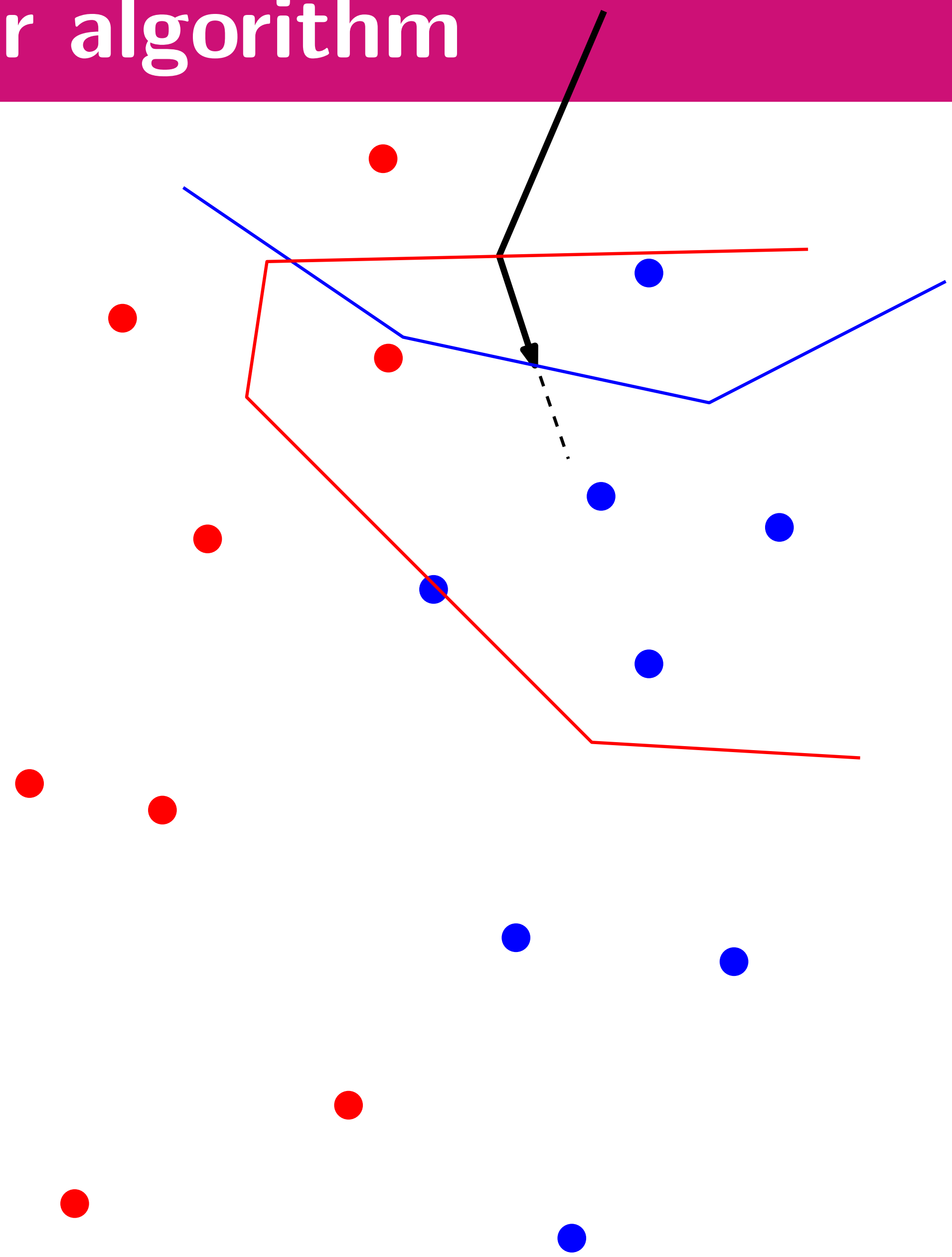
Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm



How to compute the chain?

Initialization

Find the two halflines

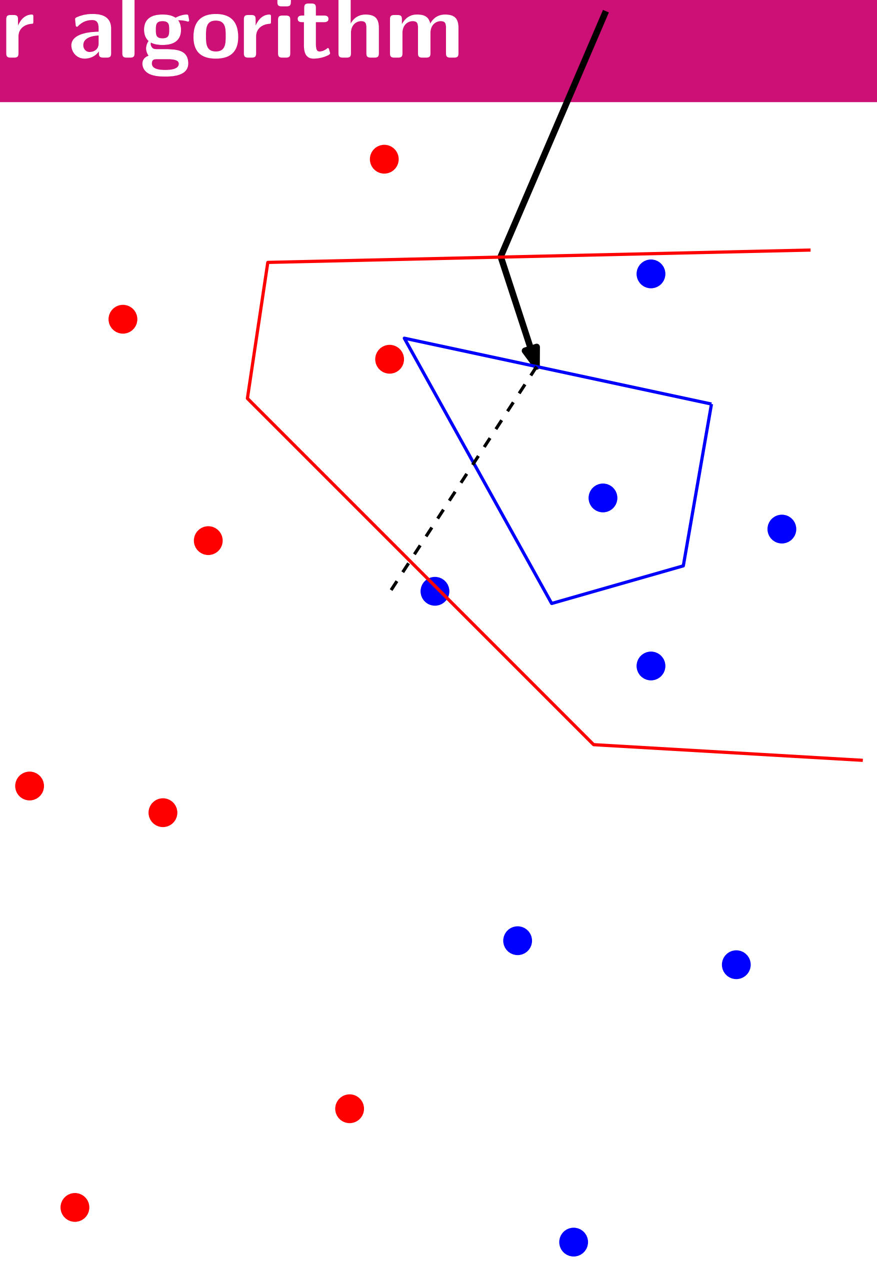
Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge

Divide and conquer algorithm



How to compute the chain?

Initialization

Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R,B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge

Divide and conquer algorithm

How to compute the chain?

Initialization

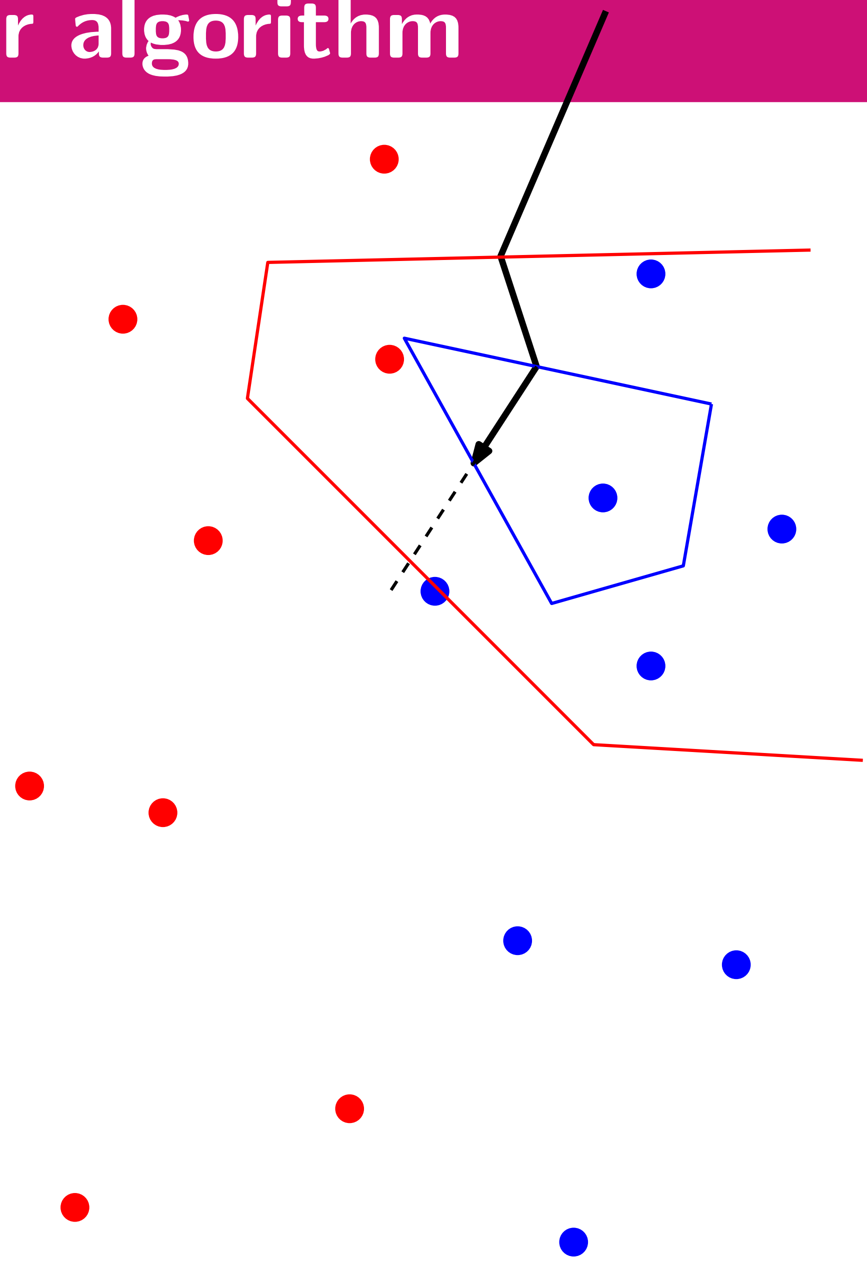
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

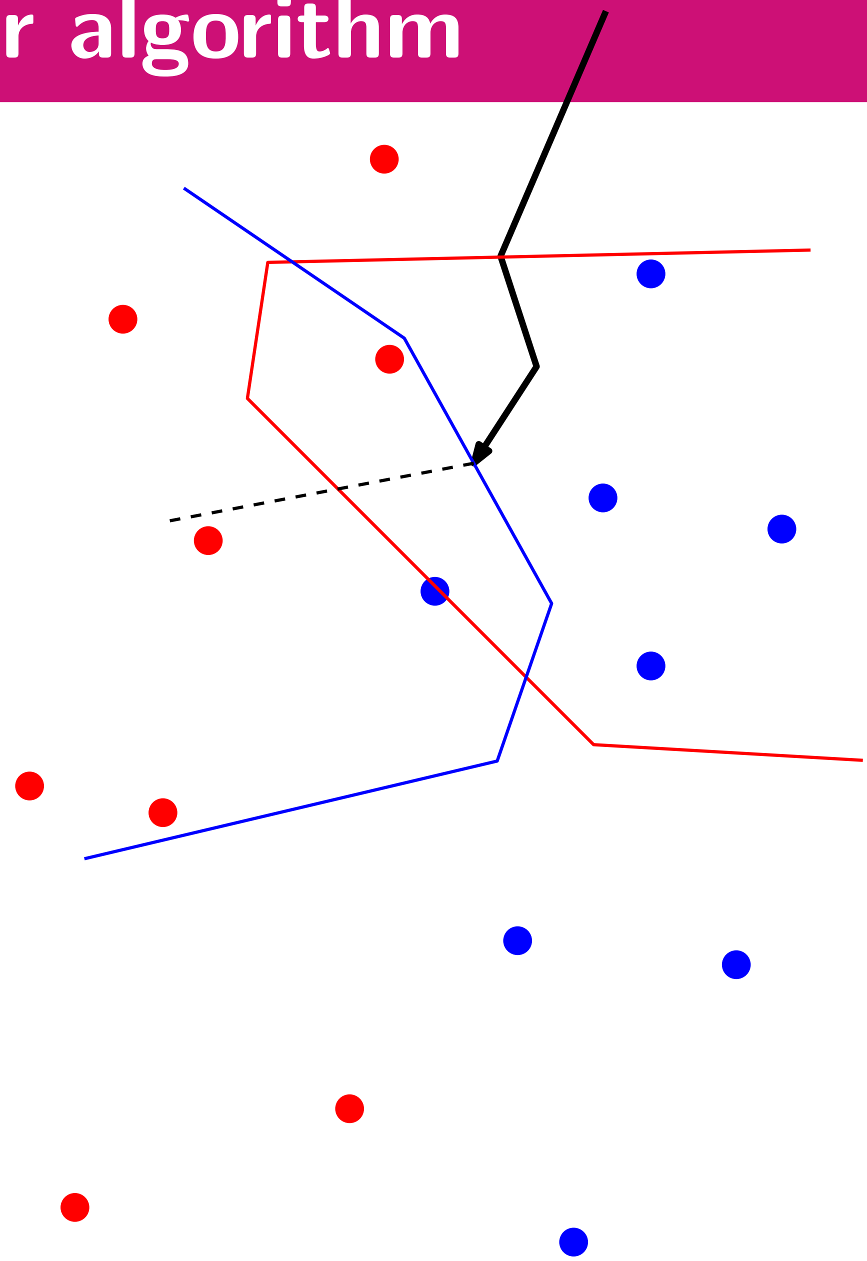
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

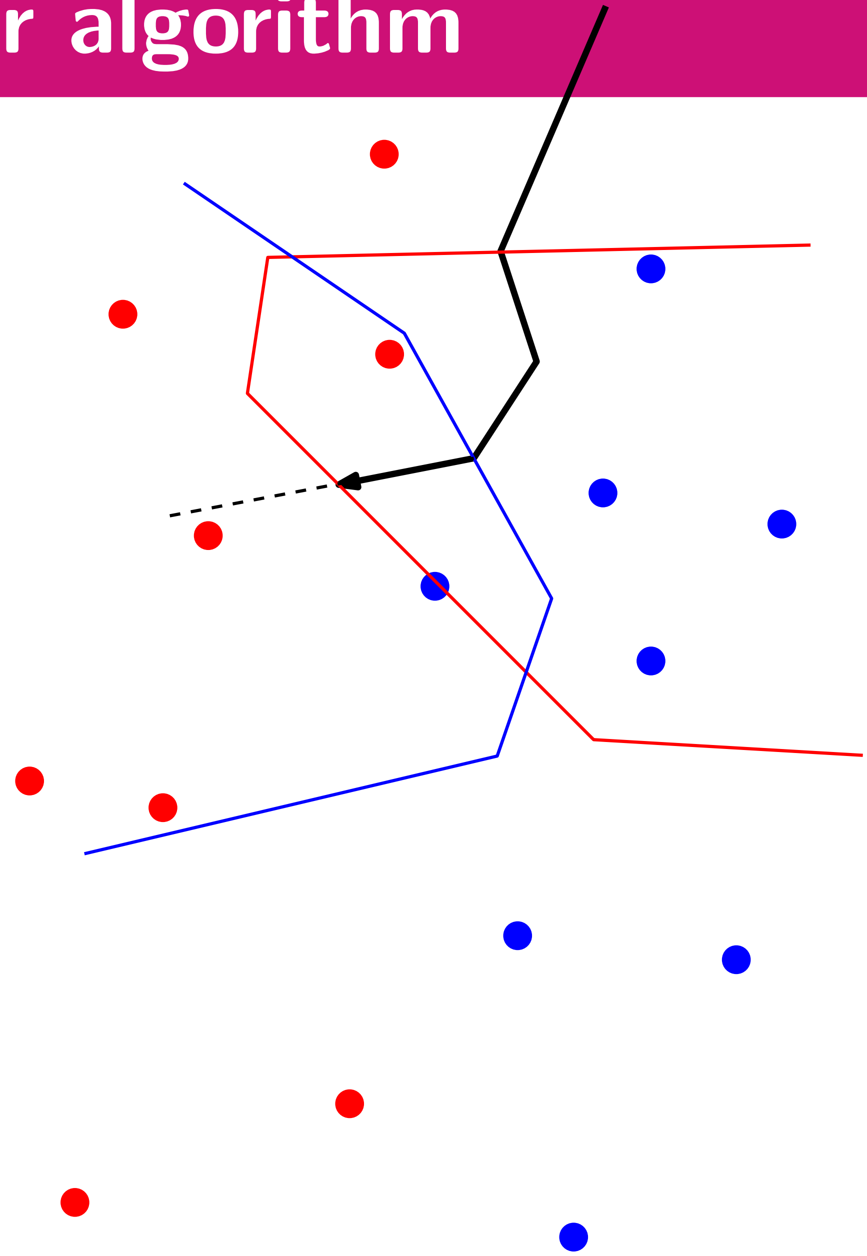
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

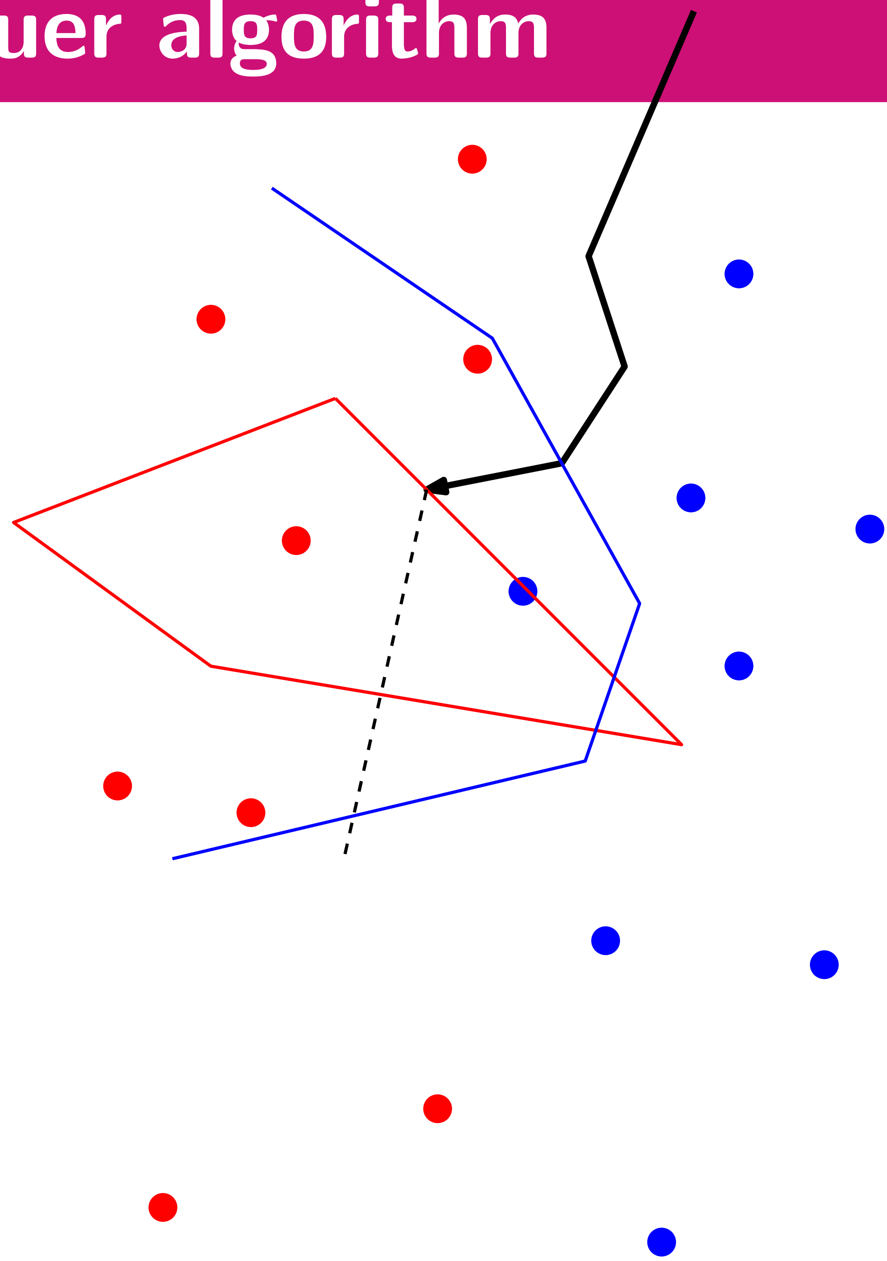
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

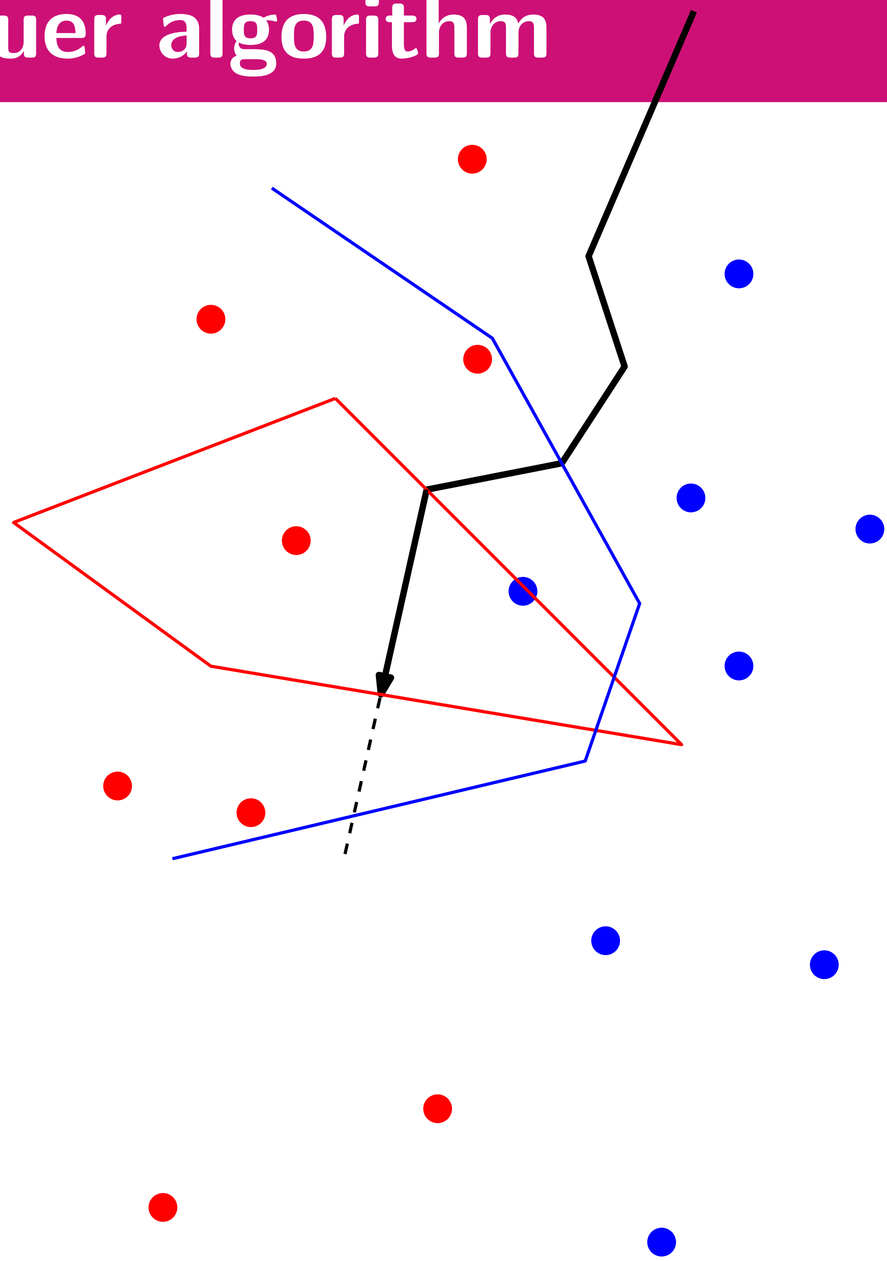
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

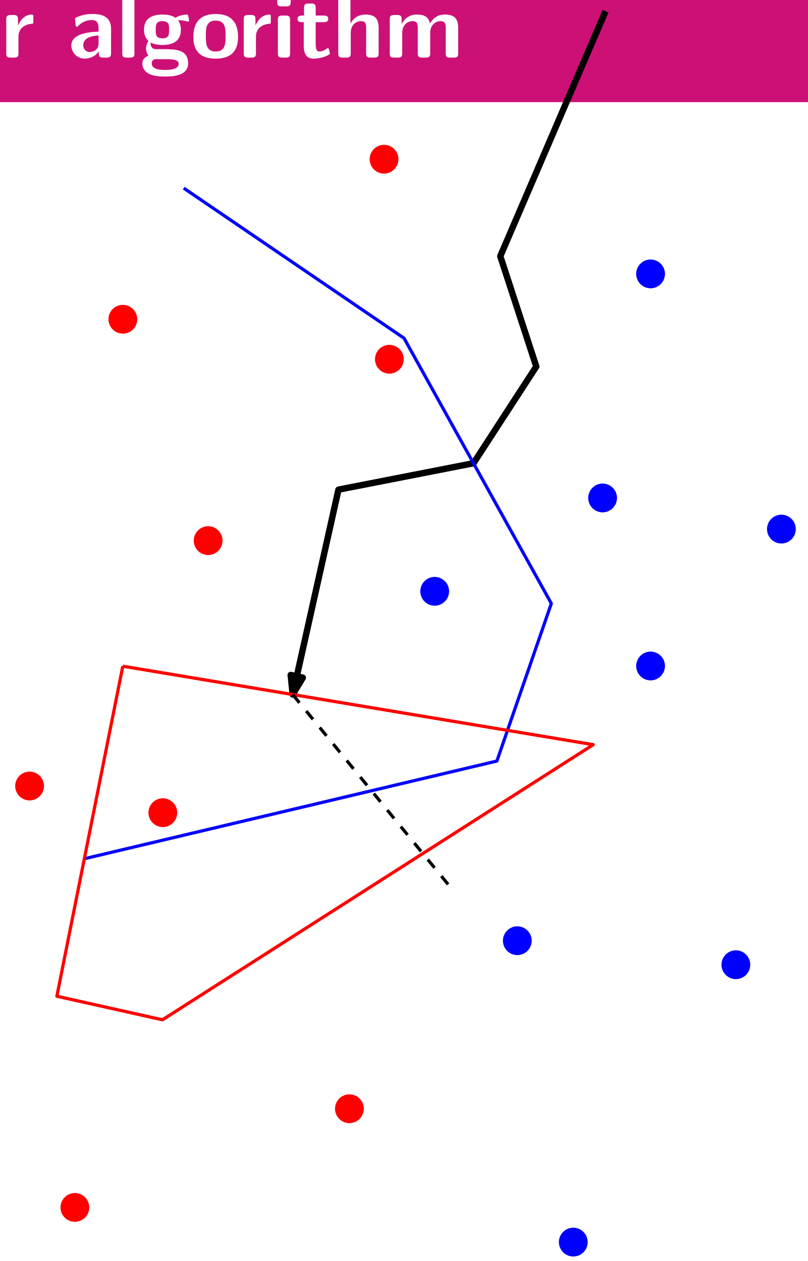
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

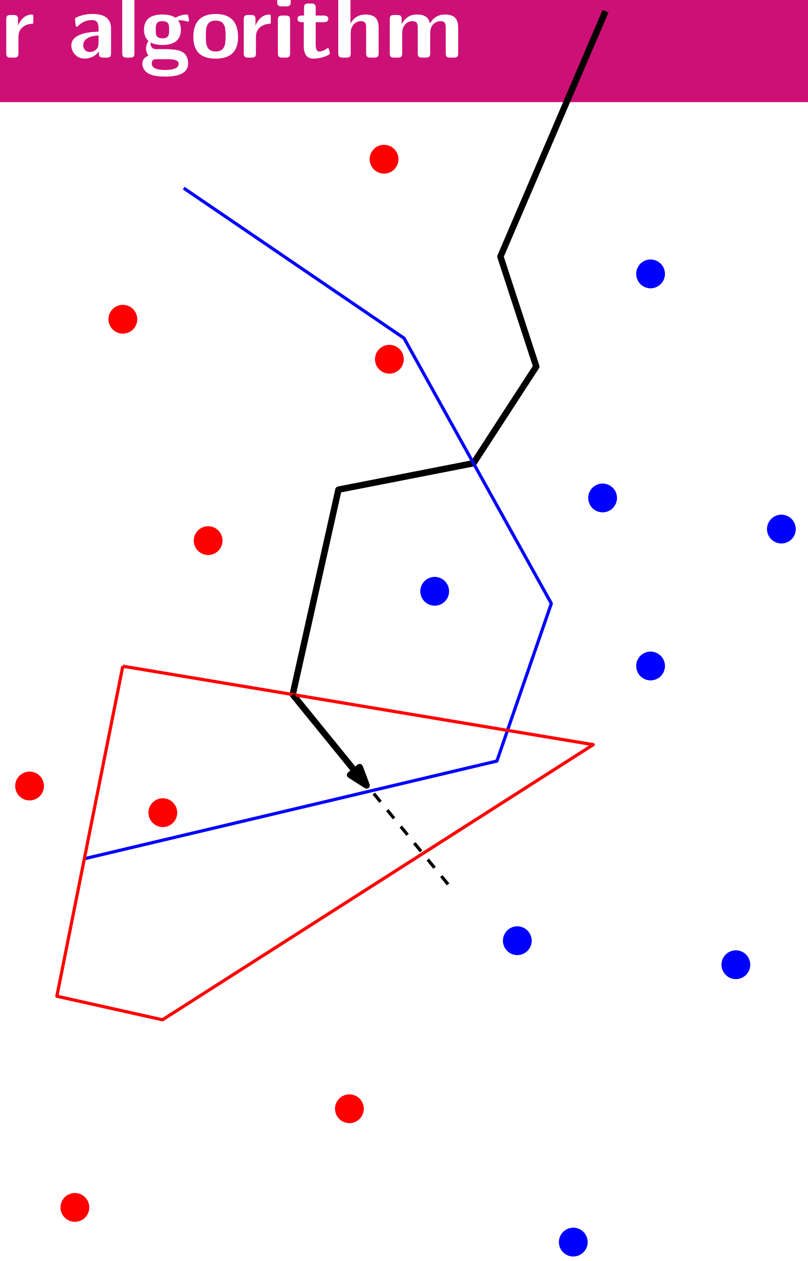
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

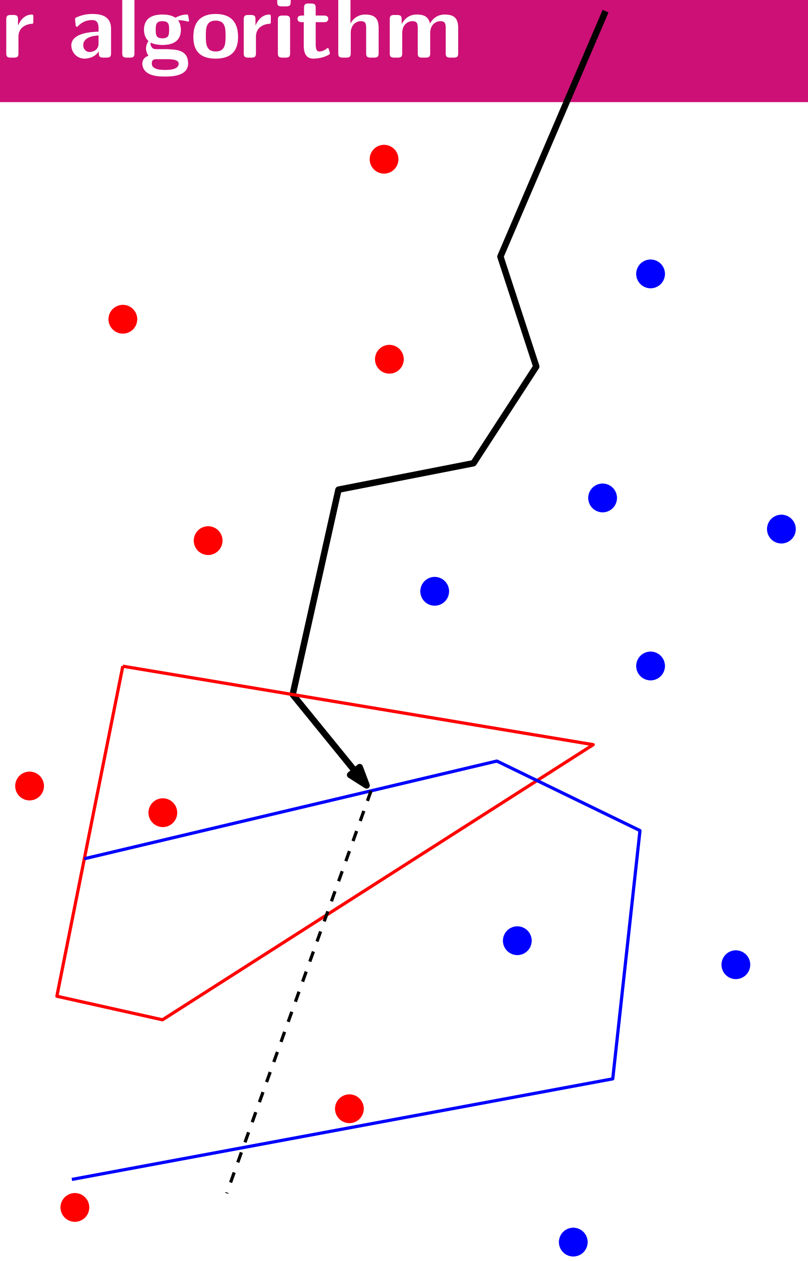
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

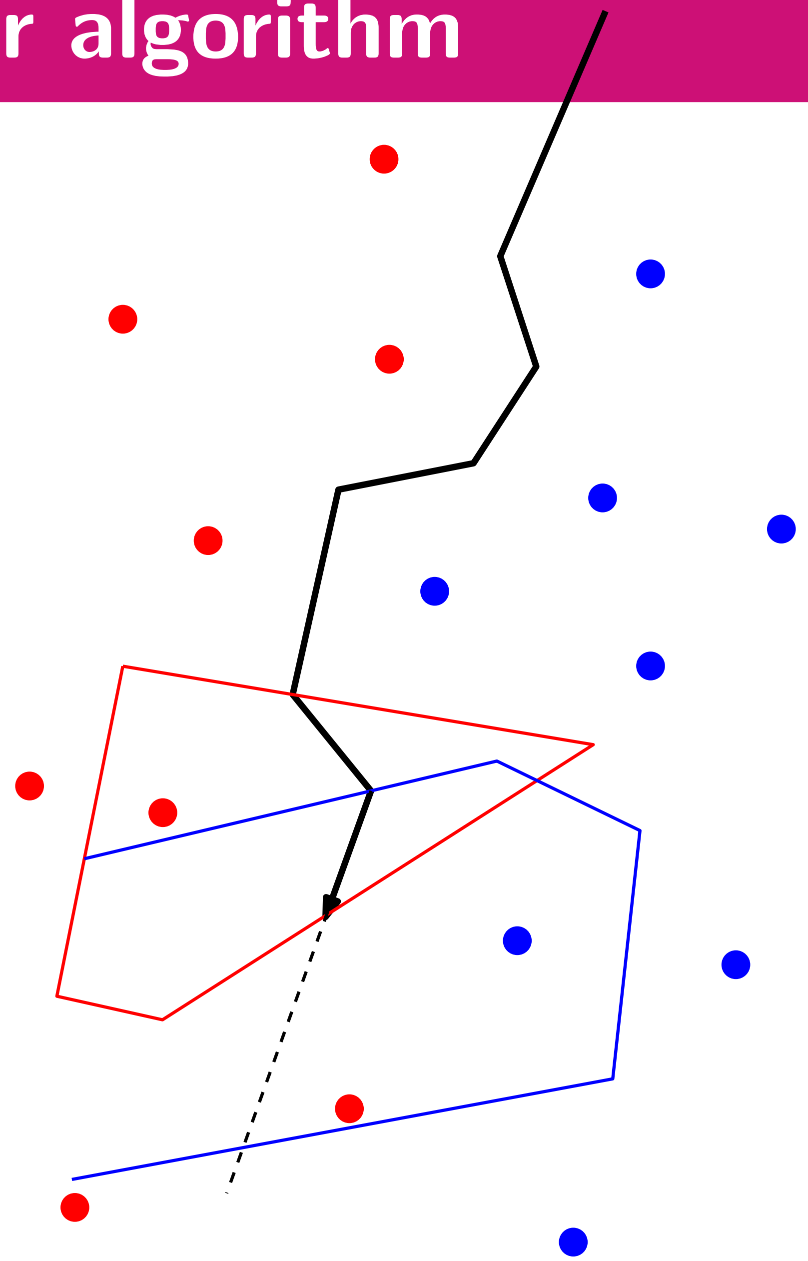
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

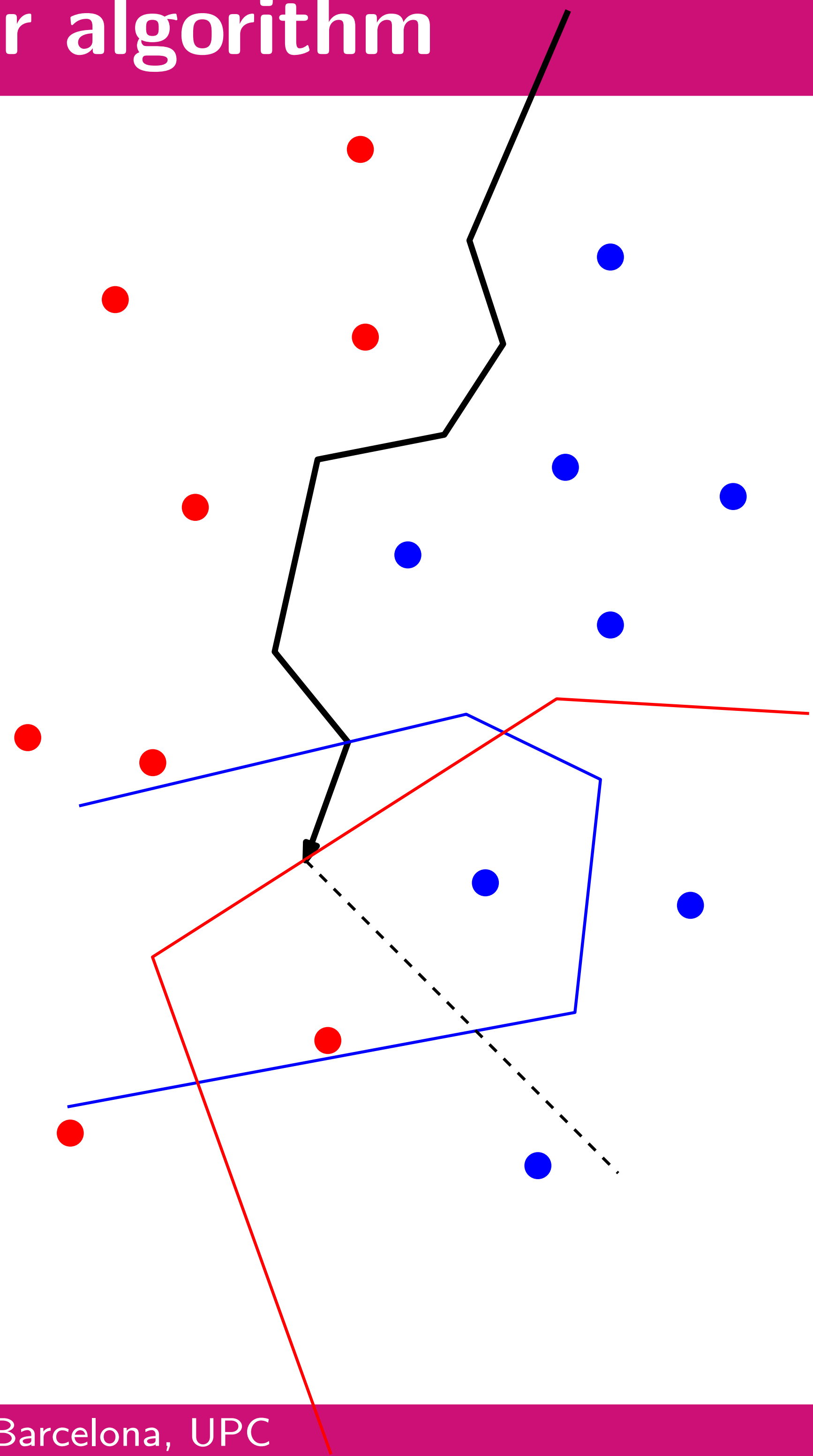
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

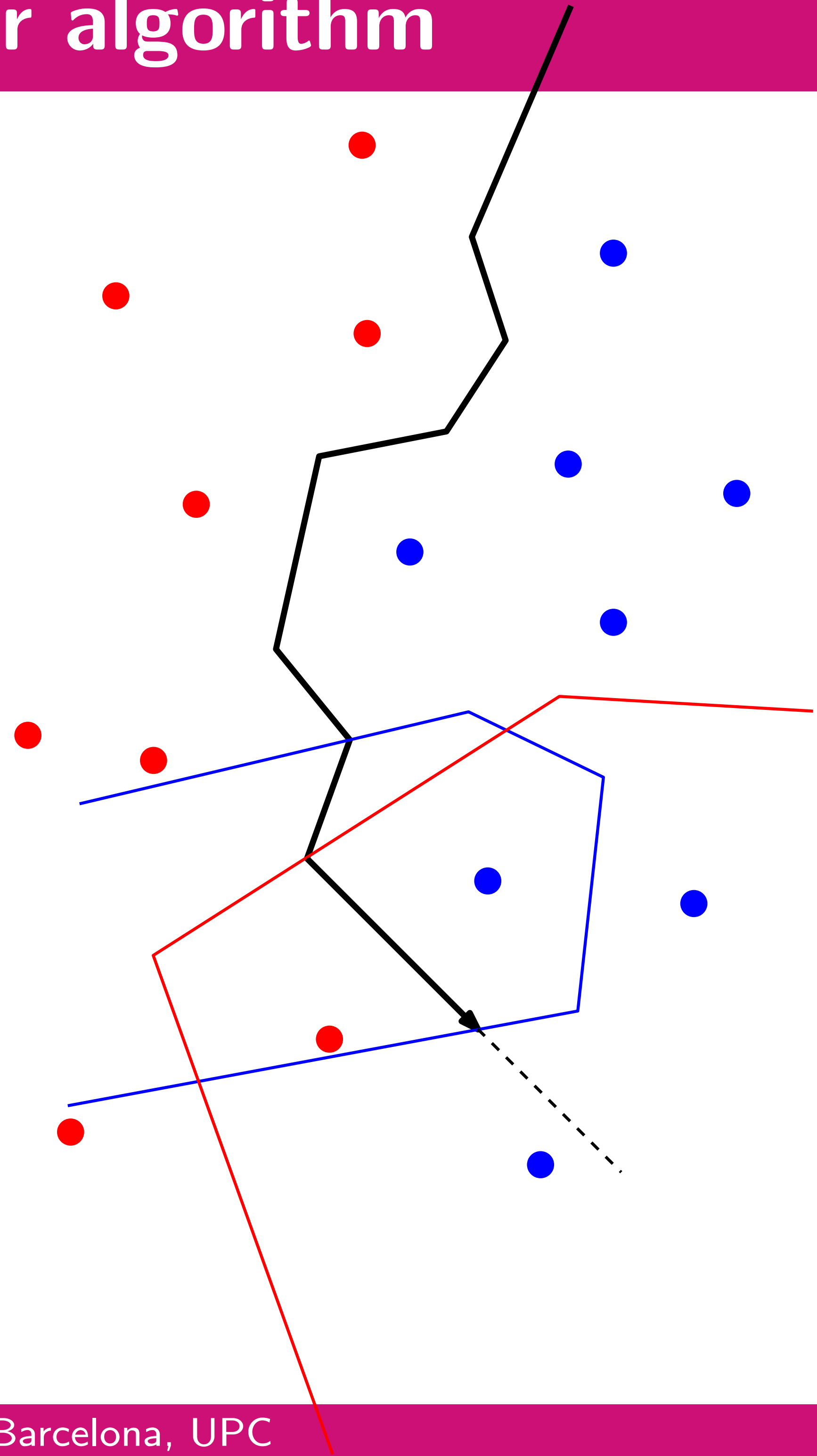
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

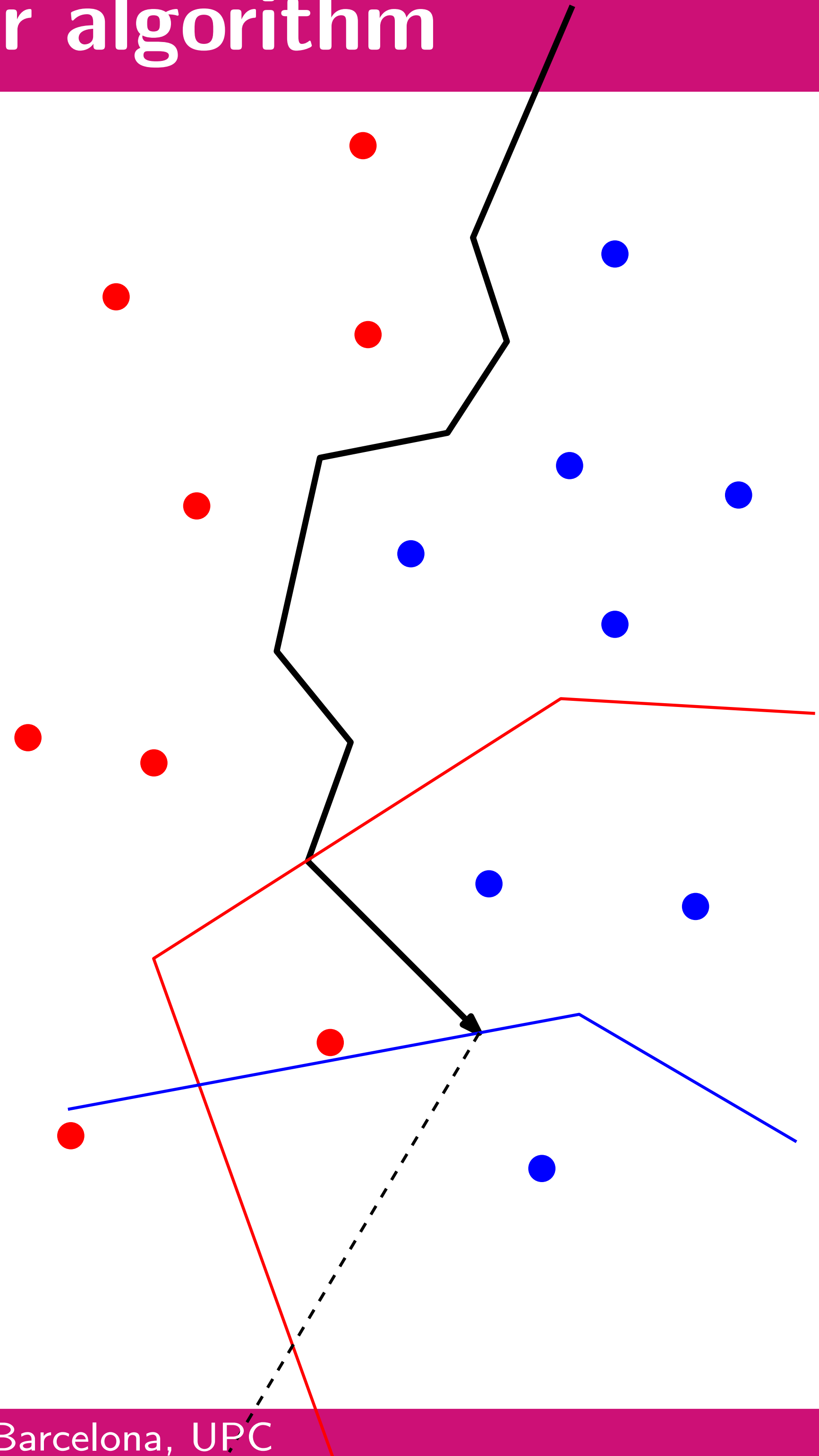
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

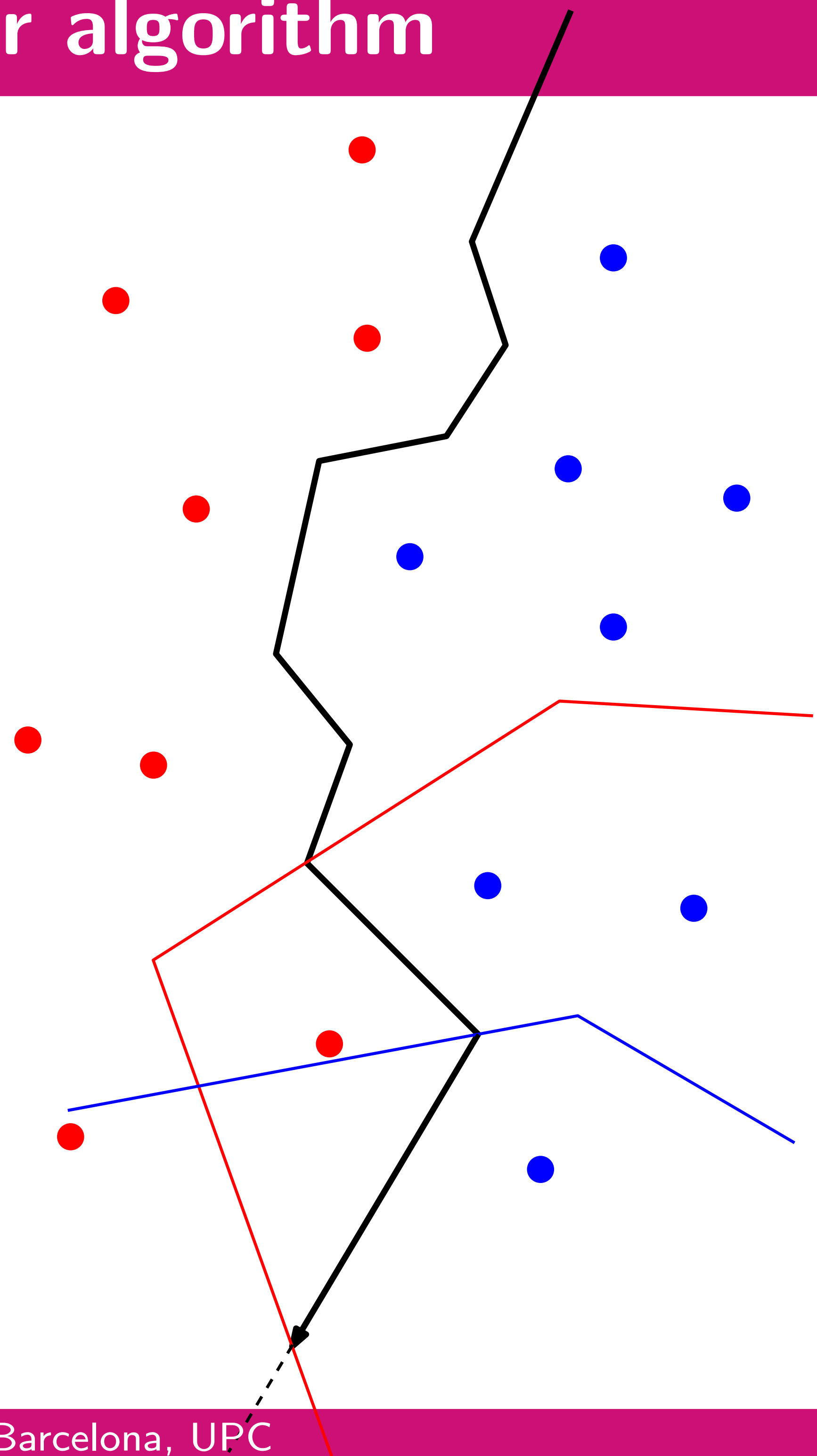
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

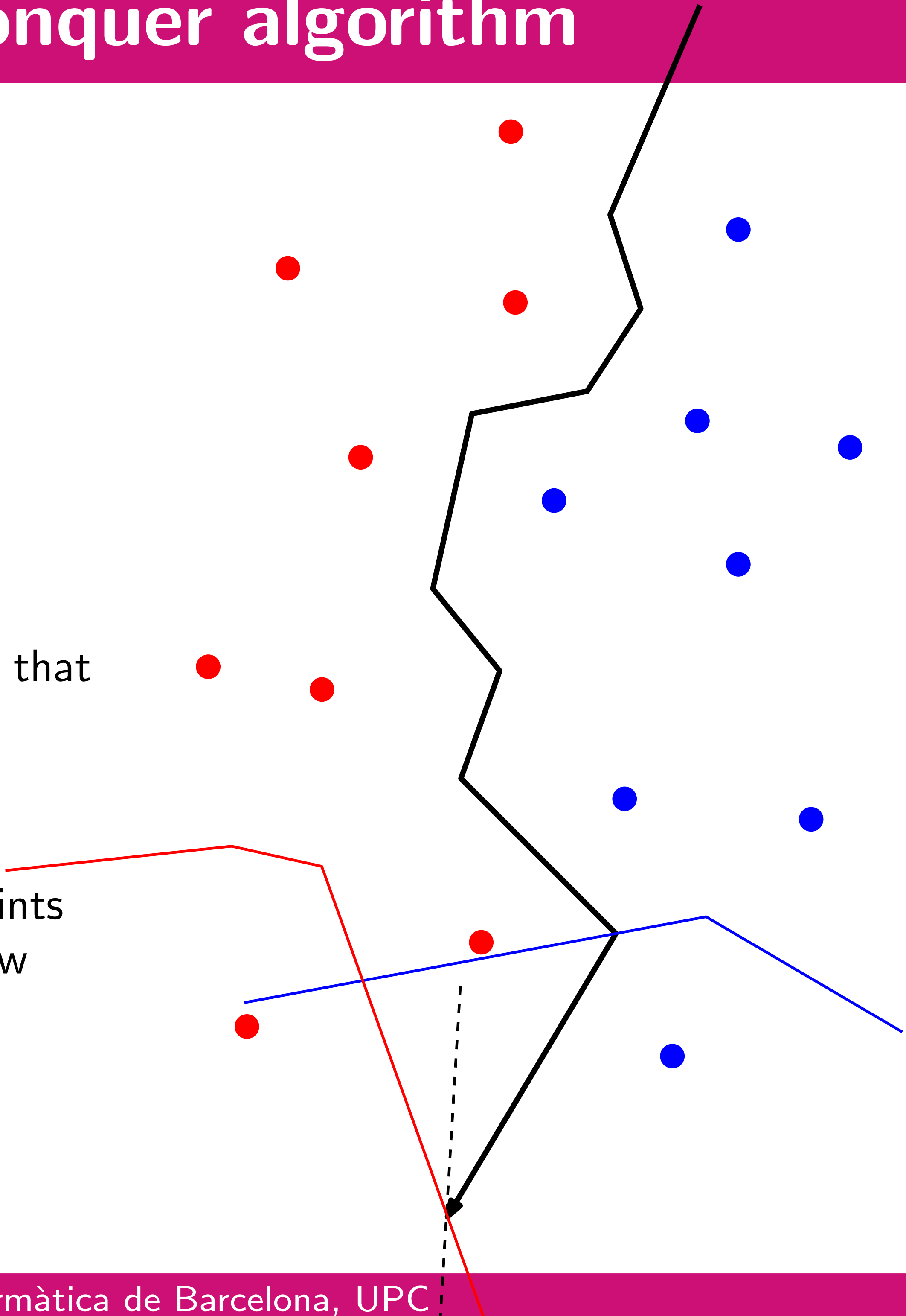
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

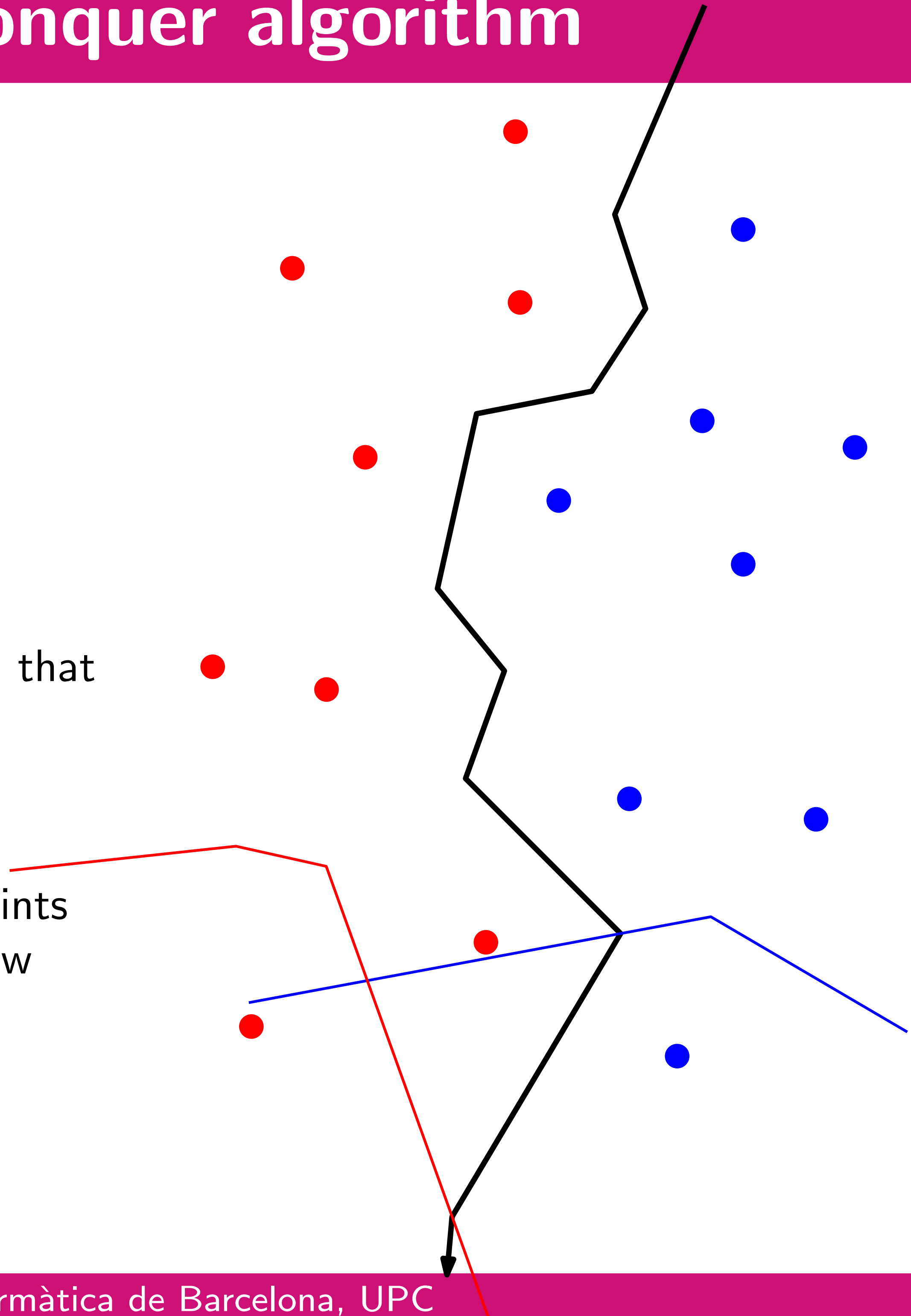
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization

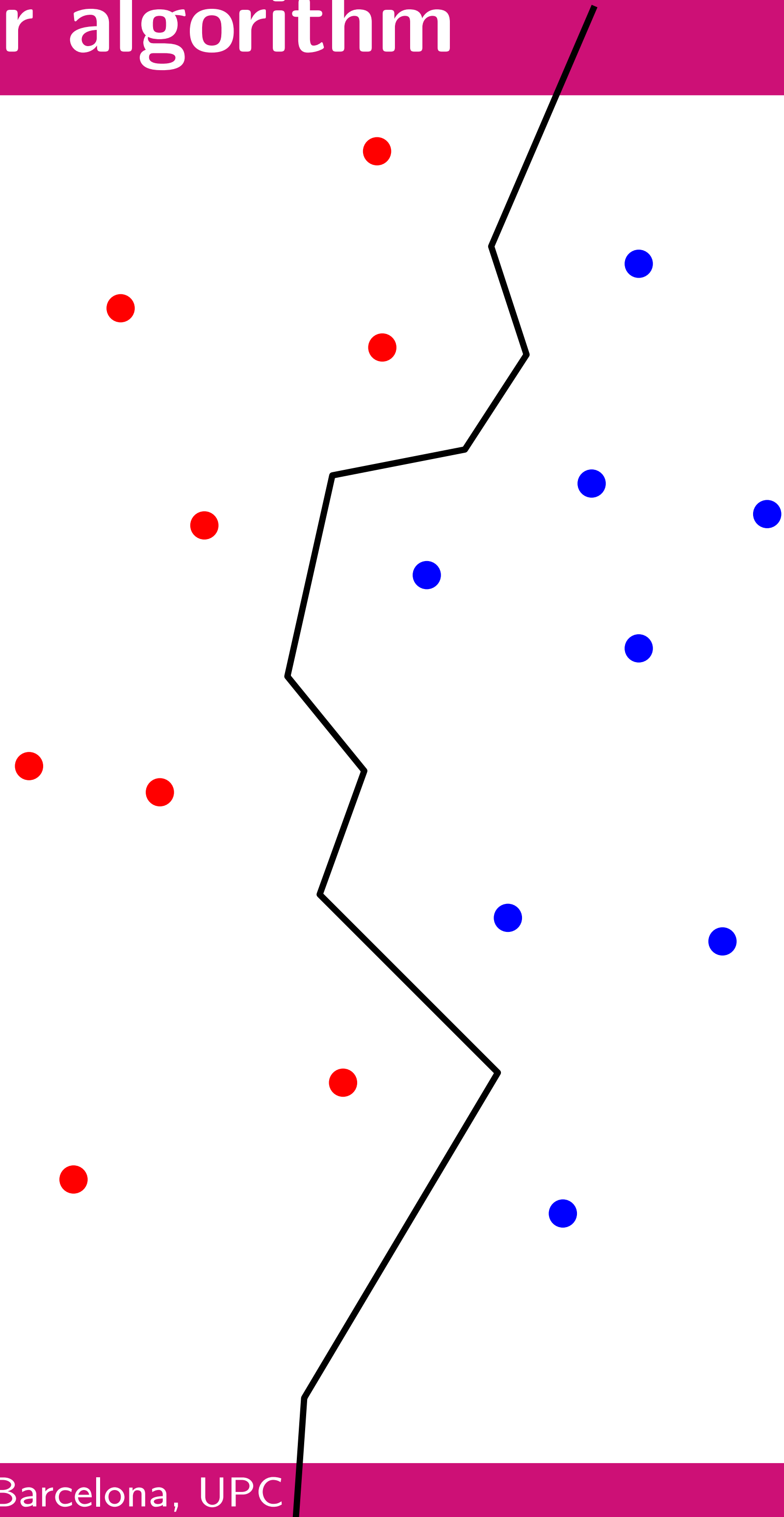
Find the two halflines

Advance

Starting with one of the halflines,
and until getting to the other one, do:

Each time an edge $e \in b(R, B)$ begins, such that $e \subset b_{ij}$, $p_i \in R$ and $p_j \in B$, do:

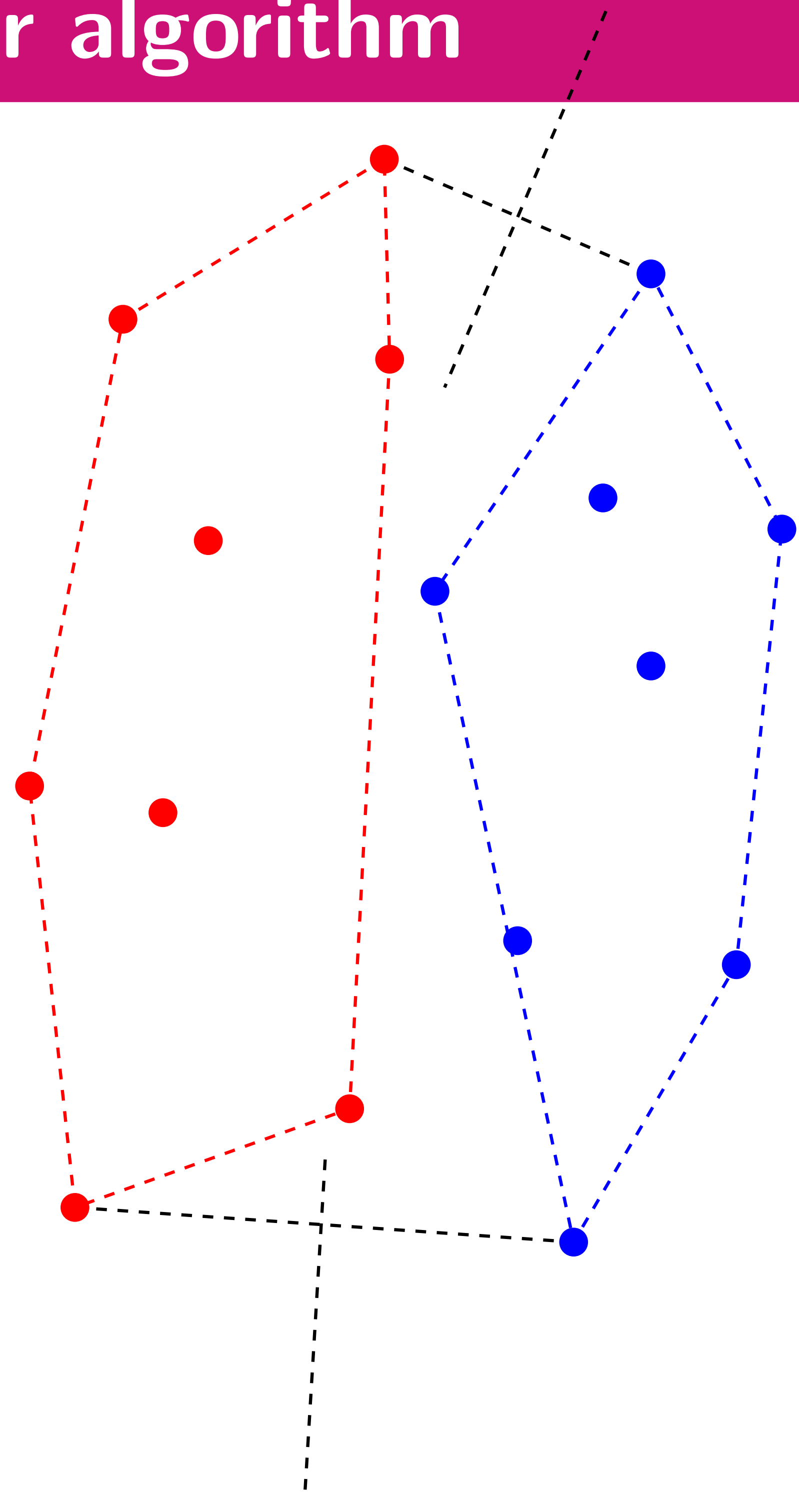
- Detect its intersection with $Vor_R(p_i)$
- Detect its intersection with $Vor_B(p_j)$
- Choose the first of the two intersection points
- Detect the site p_k corresponding to the new starting region
- Replace p_i or p_j (as required) by p_k
- Restart with the new edge



Divide and conquer algorithm

How to compute the chain?

Initialization running time: $O(n)$

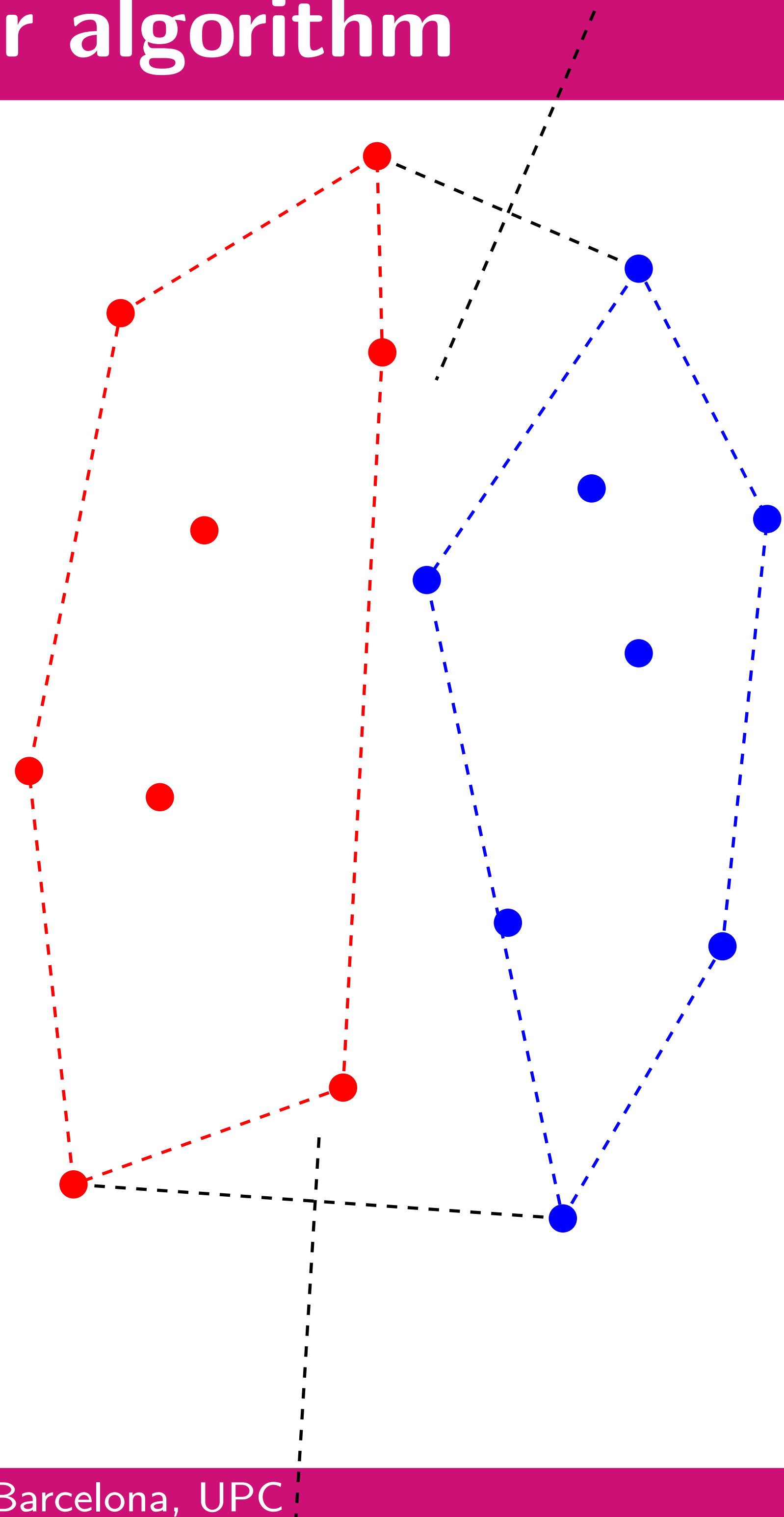


Divide and conquer algorithm

How to compute the chain?

Initialization running time: $O(n)$

From $Vor(R)$ and $Vor(B)$.

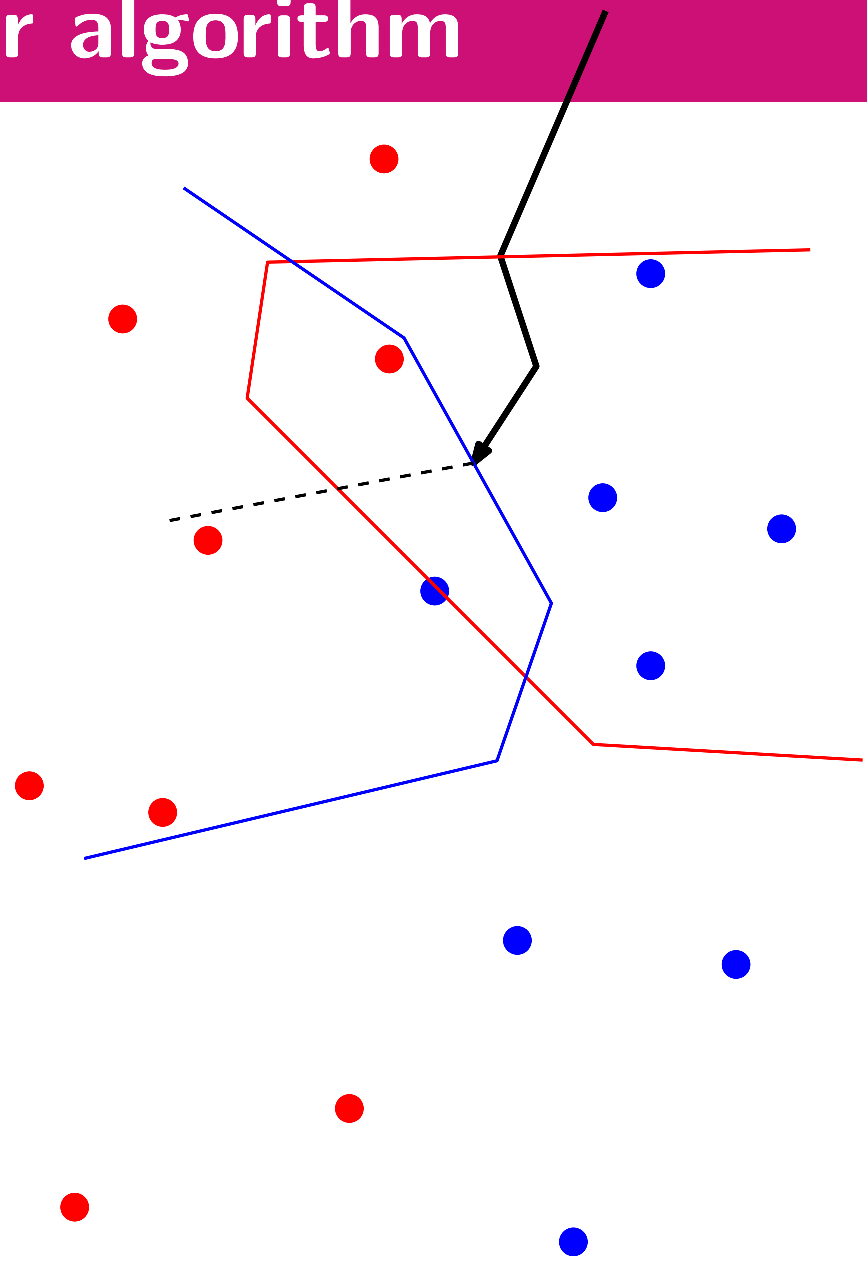


Divide and conquer algorithm

How to compute the chain?

Initialization running time: $O(n)$

Advance running time: $O(n)$



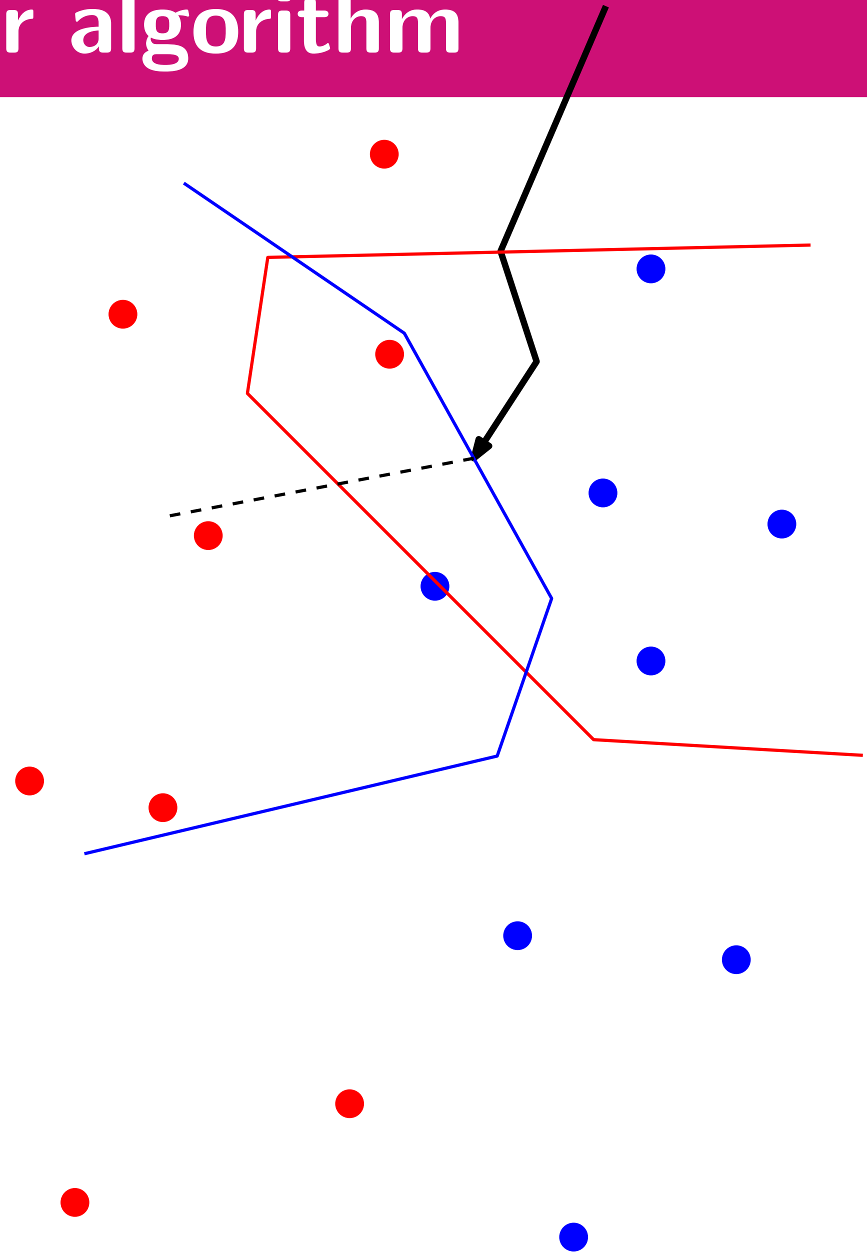
Divide and conquer algorithm

How to compute the chain?

Initialization running time: $O(n)$

Advance running time: $O(n)$

If e is an edge of $b(R, B)$ that entered $Vor_R(p_i)$ through some vertex $v \in Vor(P)$, then the exit point of $b(R, B)$ is found clockwise along the boundary of $Vor_R(p_i)$.



Divide and conquer algorithm

How to do the merging?

Divide and conquer algorithm

How to do the merging?

It consists in updating the DCEL:

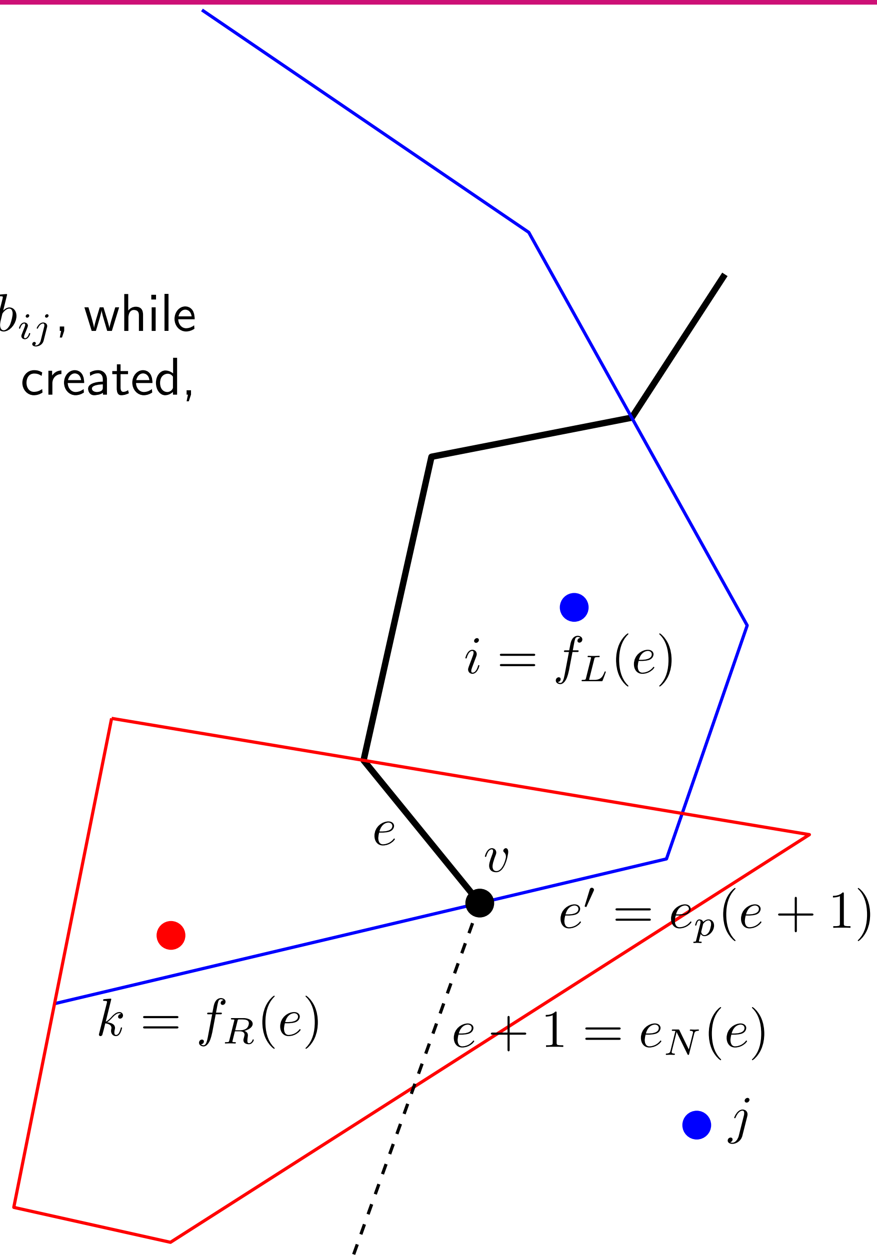
Divide and conquer algorithm

How to do the merging?

It consists in updating the DCEL:

Each time a face $Vor_B(p_i)$ is left through an edge $e' \in b_{ij}$, while staying in the same face $Vor_R(p_k)$, a new vertex v is created, an edge e ends and another edge $e+1$ begins:

- Create $e+1$ and assign to it $v_B = v$ and $e_P = e'$
- Assign to e: $v_E = v$, $e_N = e+1$, $f_L = i$ and $f_R = k$
- Modify for e': $v_* = v$, $e_* = e$
- Delete all edges of $Vor_B(p_i)$ found in counter-clockwise order between the entry and exit points
- Update $e(p_i) = e$
- Create the new vertex v and assign $e(v) = e$



The procedure is analogous when exiting a face $Vor_R(p_i)$.

Divide and conquer algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

2. Recursively compute $Vor(R)$ and $Vor(B)$.

3. Compute the separating chain.

4. Prune the portion of $Vor(R)$ lying to the right of the chain and the portion of $Vor(B)$ lying to its left.

The total running time of the algorithm is $O(n \log n)$

Divide and conquer algorithm

1. Sort the points of P by abscissa (only once) and vertically partition P into two subsets R and B, of approximately the same size.

2. Recursively compute $Vor(R)$ and $Vor(B)$.

3. Compute the separating chain.

4. Prune the portion of $Vor(R)$ lying to the right of the chain and the portion of $Vor(B)$ lying to its left.

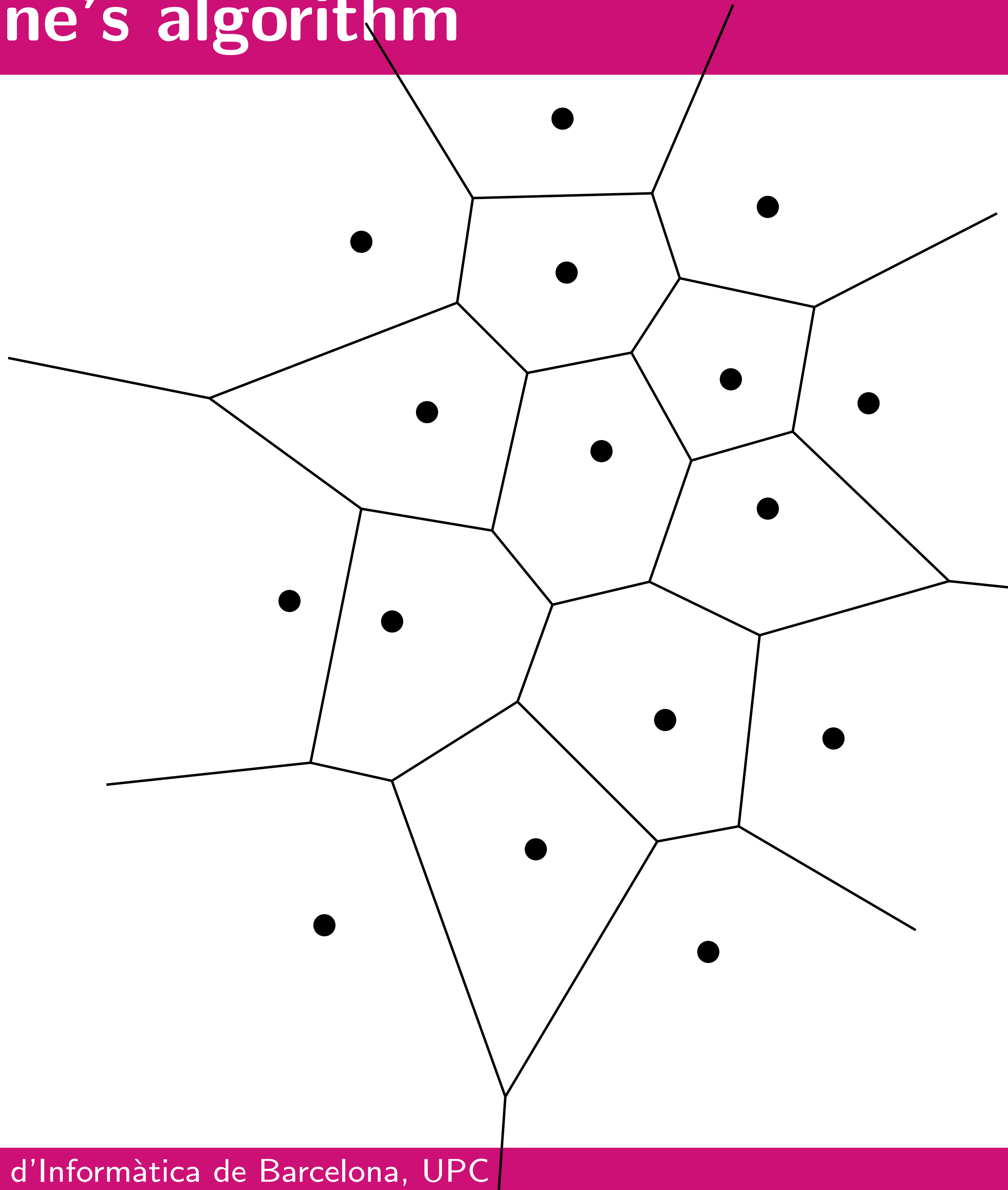
The total running time of the algorithm is $O(n \log n)$

This running time is optimal, because $ch(P)$ can be computed from $Vor(P)$ in $O(n)$ time.

Fortune's algorithm

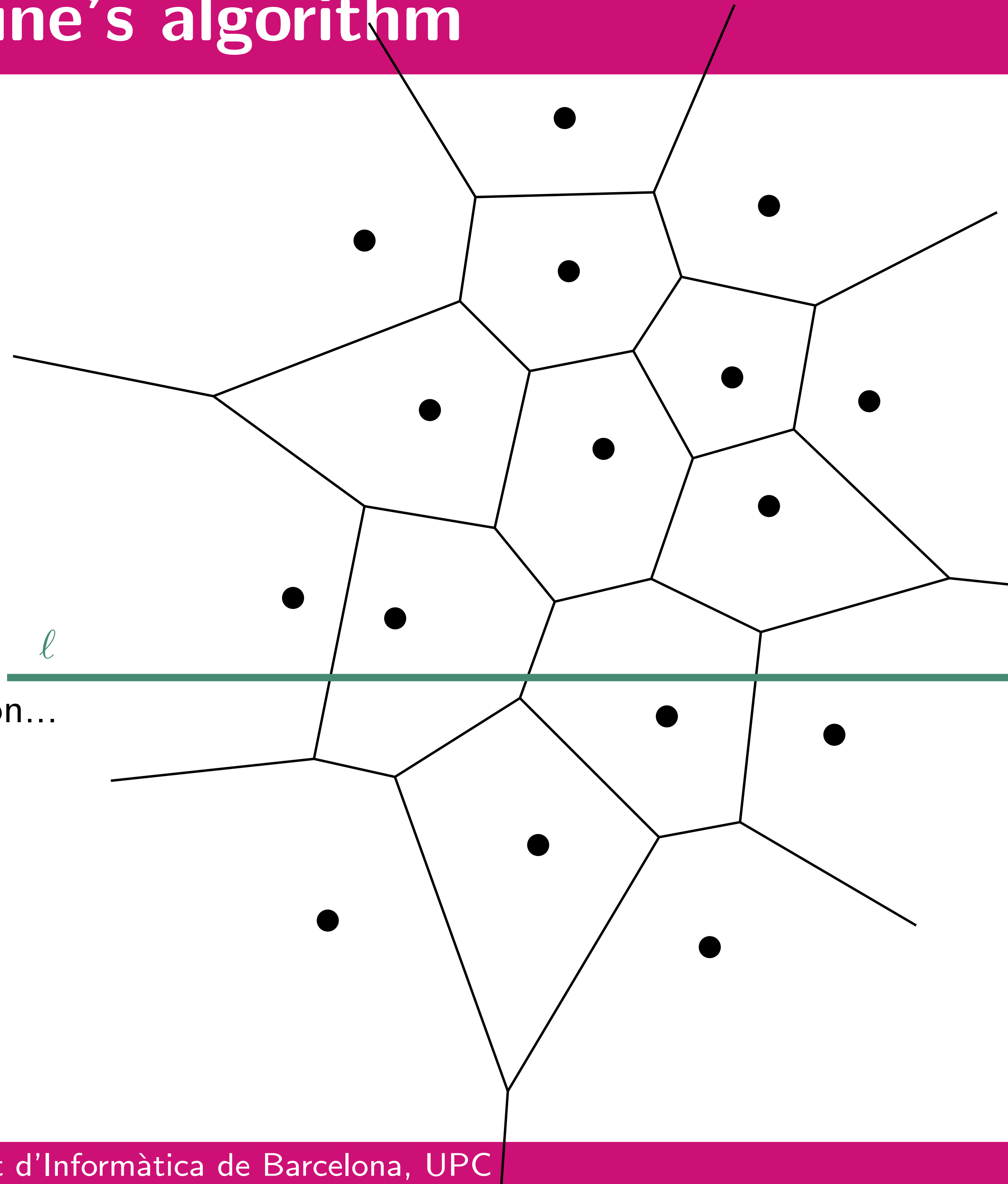
Fortune's algorithm

How to compute the Voronoi diagram with a sweep line algorithm?



Fortune's algorithm

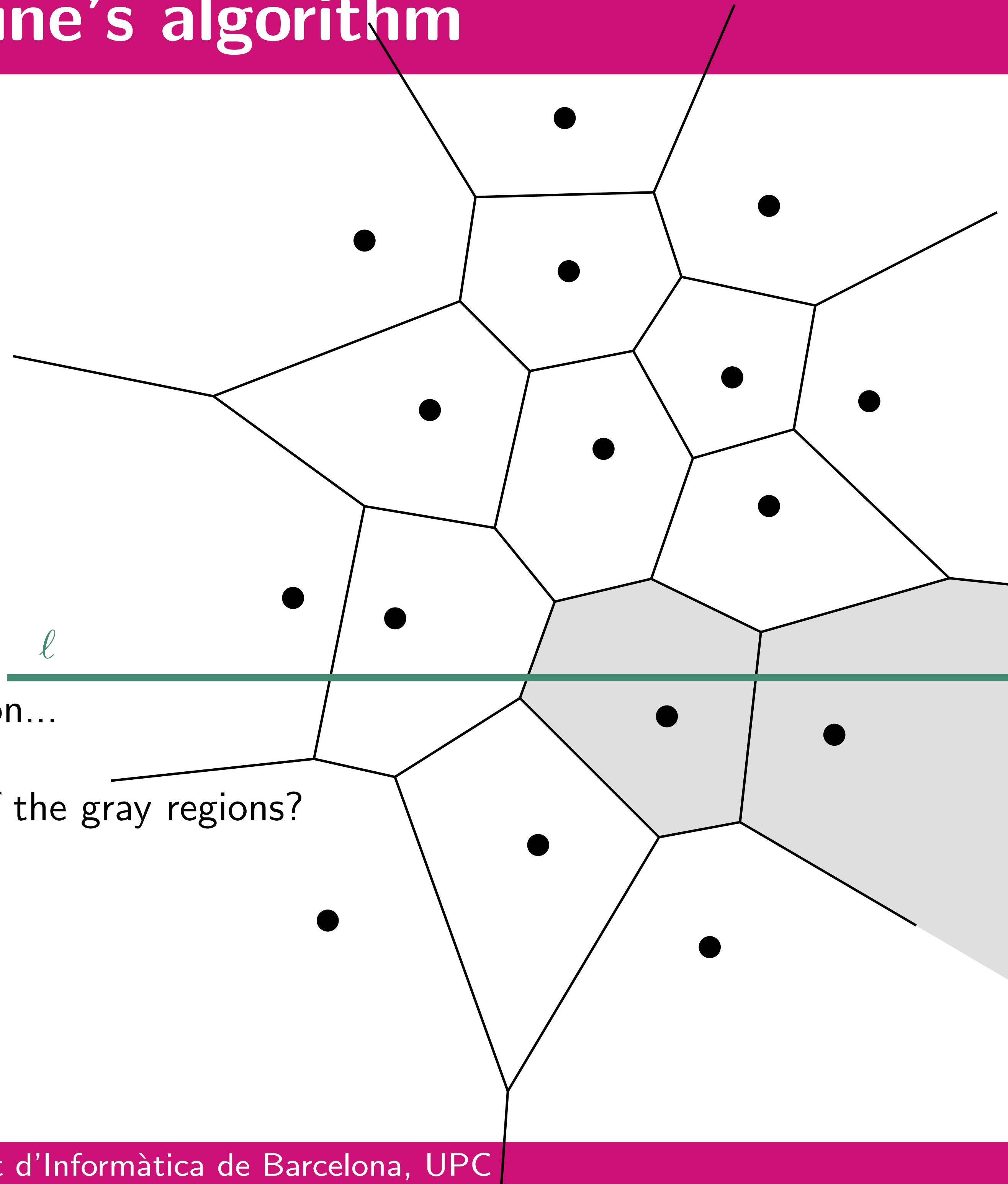
How to compute the Voronoi diagram with a sweep line algorithm?



When the sweep line gets to this position...

Fortune's algorithm

How to compute the Voronoi diagram with a sweep line algorithm?

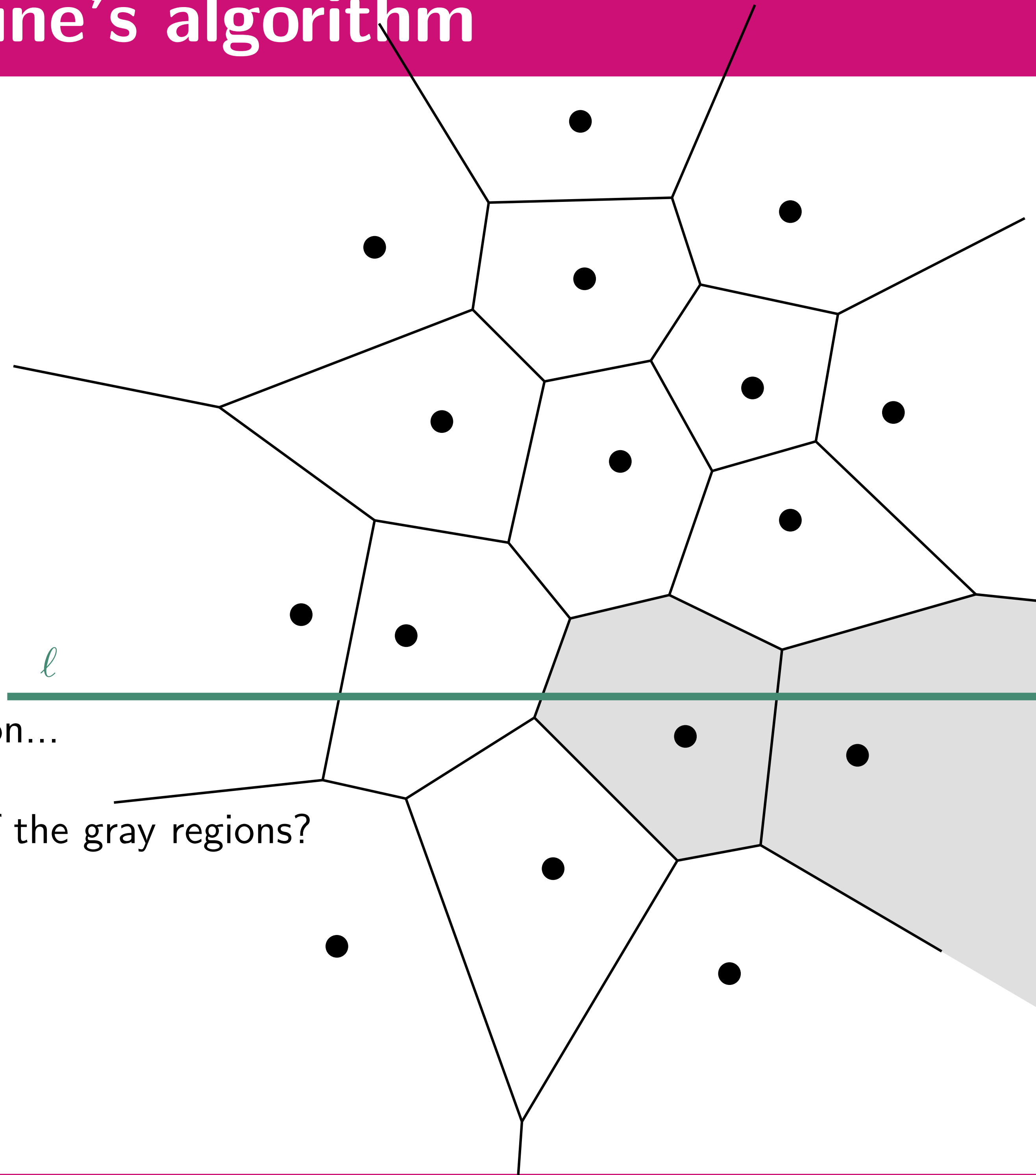


When the sweep line gets to this position...

... how can we know of the existance of the gray regions?

Fortune's algorithm

How to compute the Voronoi diagram with a sweep line algorithm?



When the sweep line gets to this position...

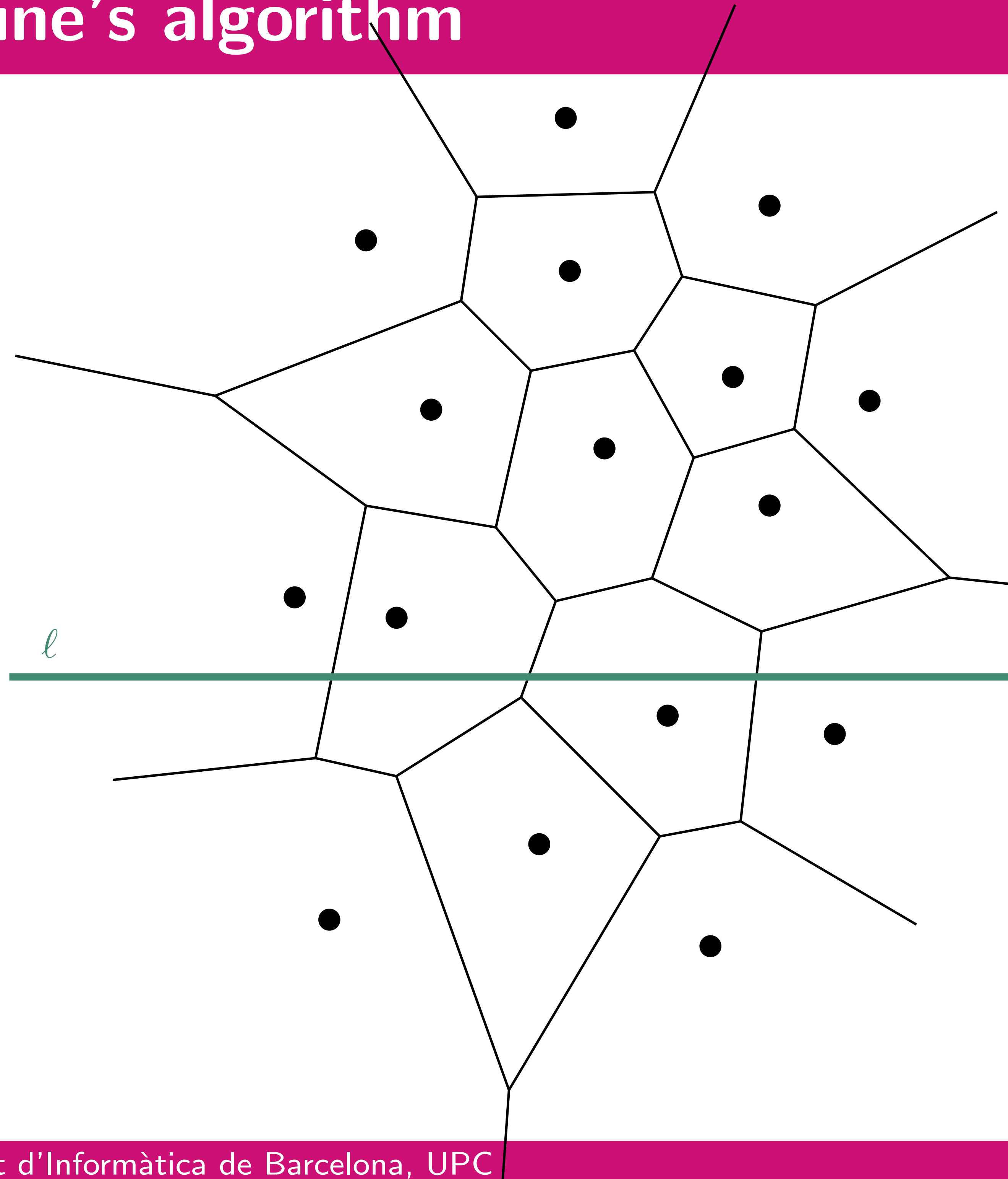
... how can we know of the existance of the gray regions?

We cannot.

Fortune's algorithm

How to compute the Voronoi diagram with a sweep line algorithm?

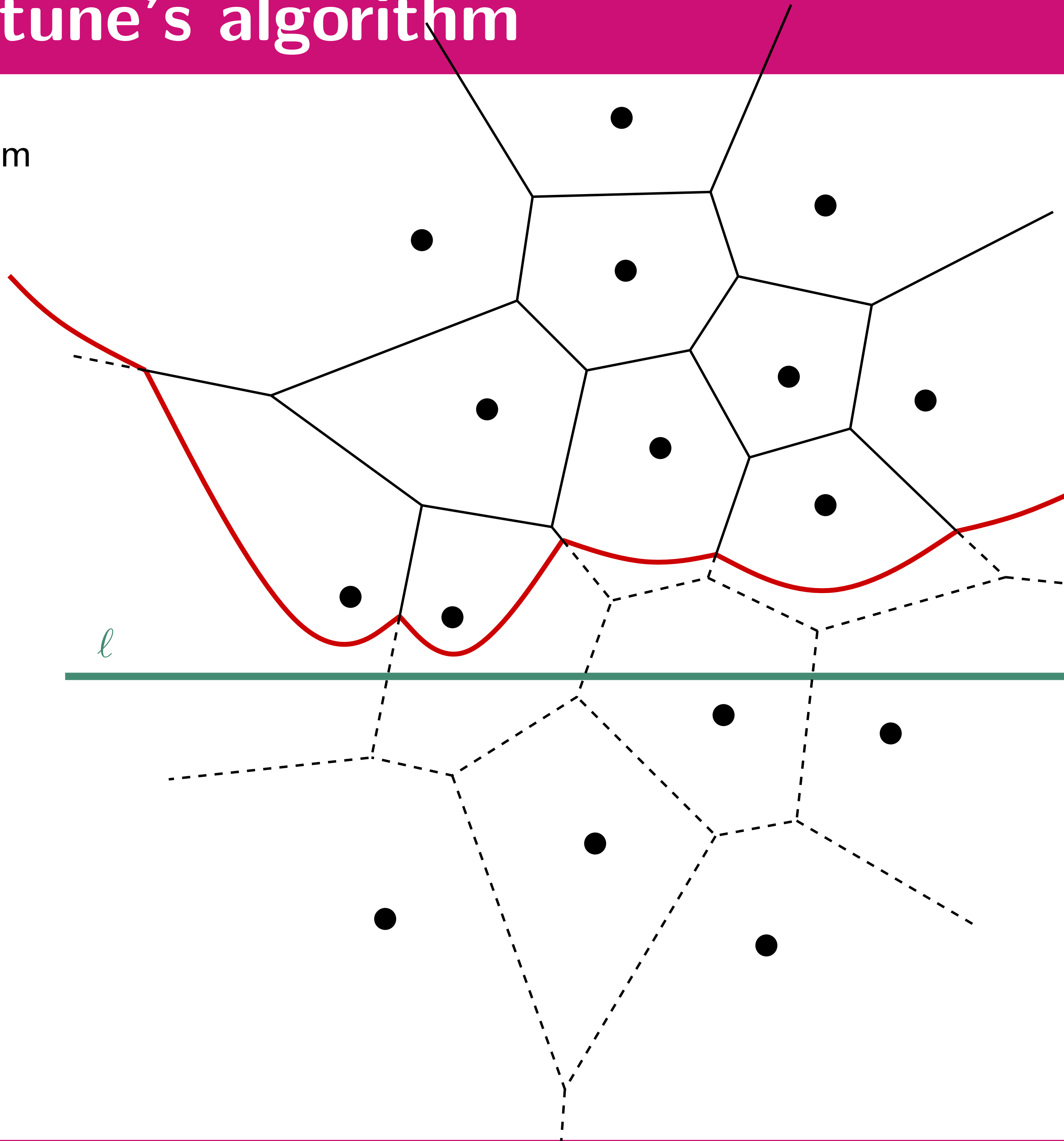
What can we know?



Fortune's algorithm

How to compute the Voronoi diagram
with a sweep line algorithm?

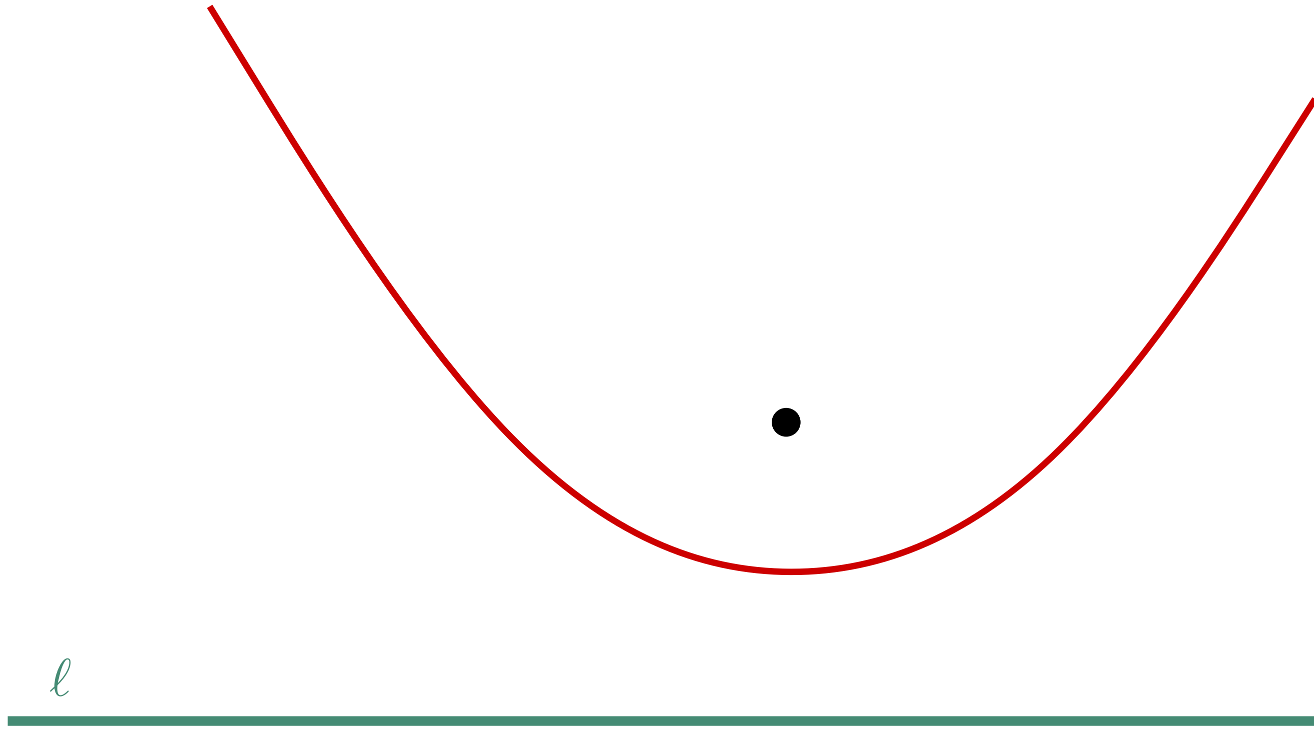
What can we know?



Fortune's algorithm

How to compute the Voronoi diagram with a sweep line algorithm?

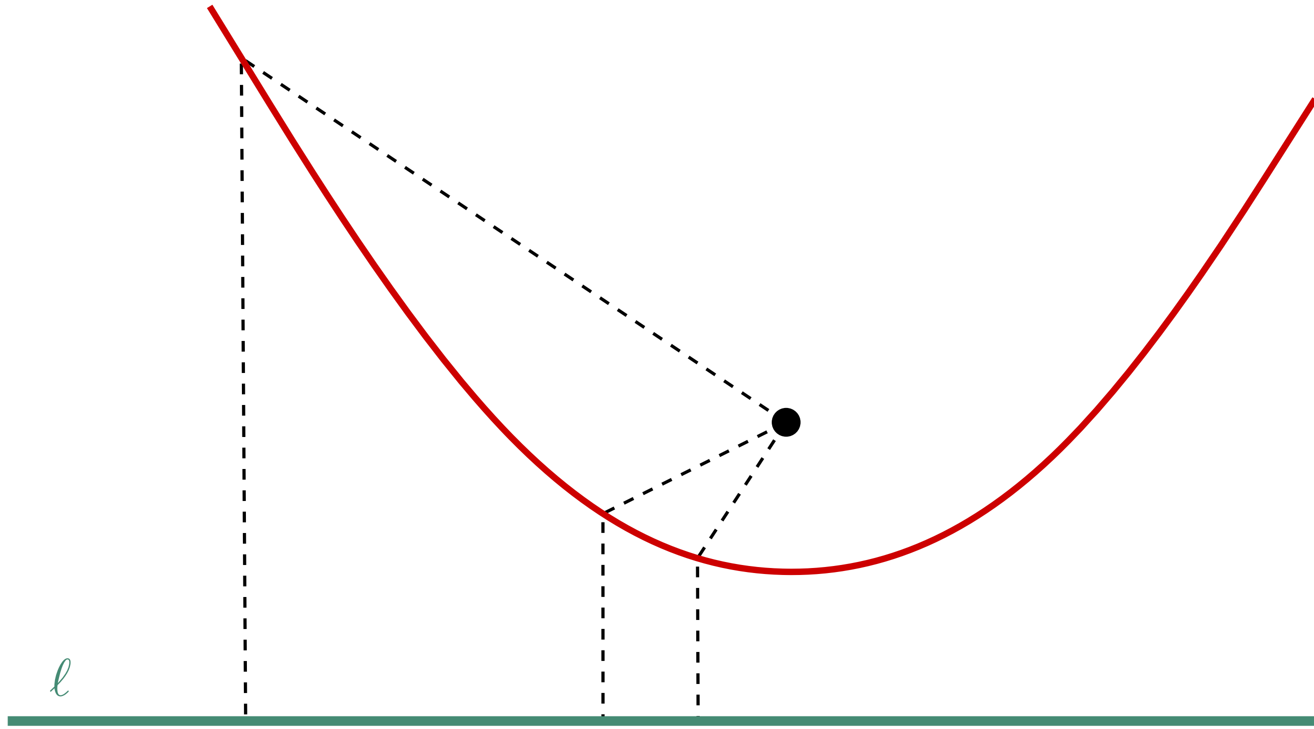
What can we know?



Fortune's algorithm

How to compute the Voronoi diagram with a sweep line algorithm?

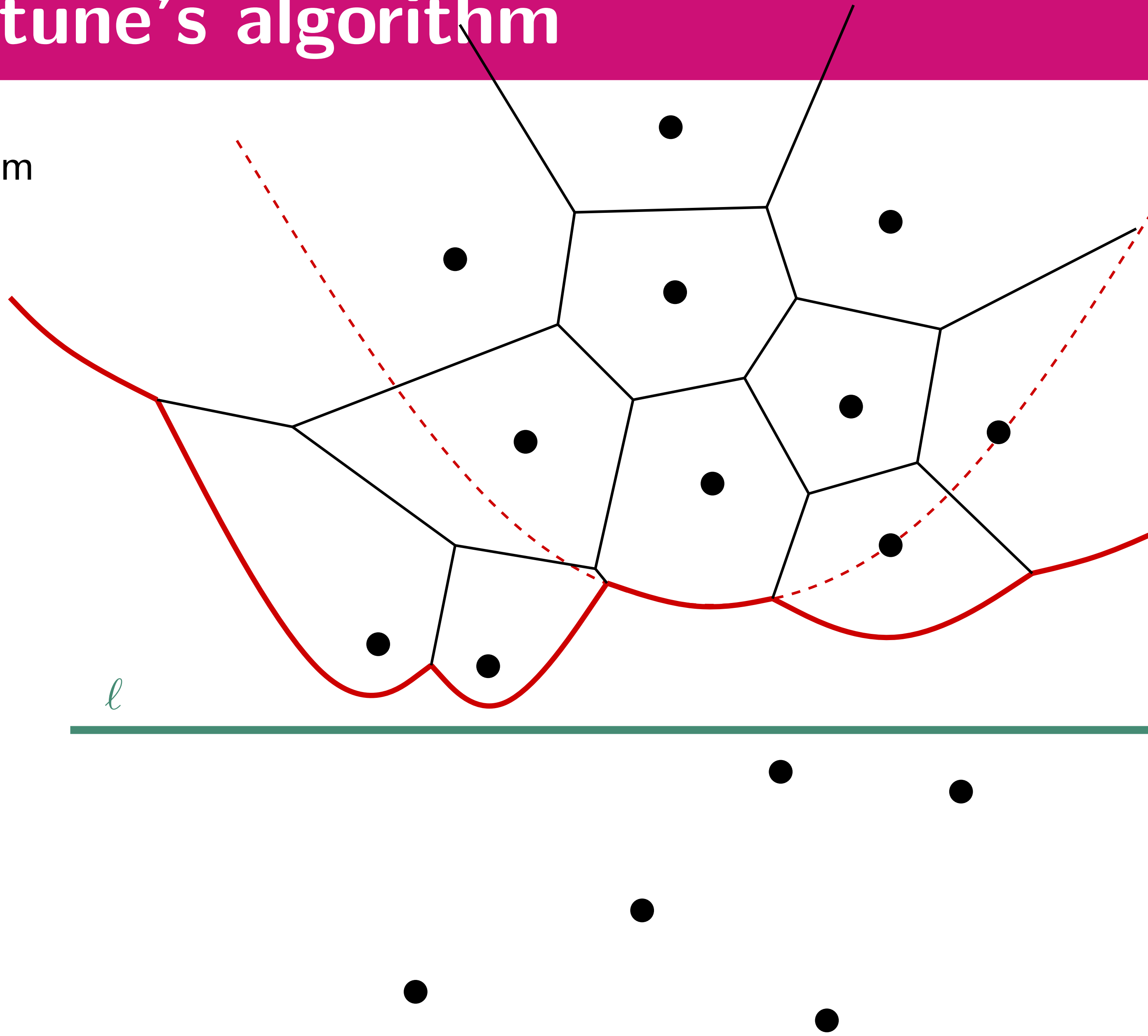
What can we know?



Fortune's algorithm

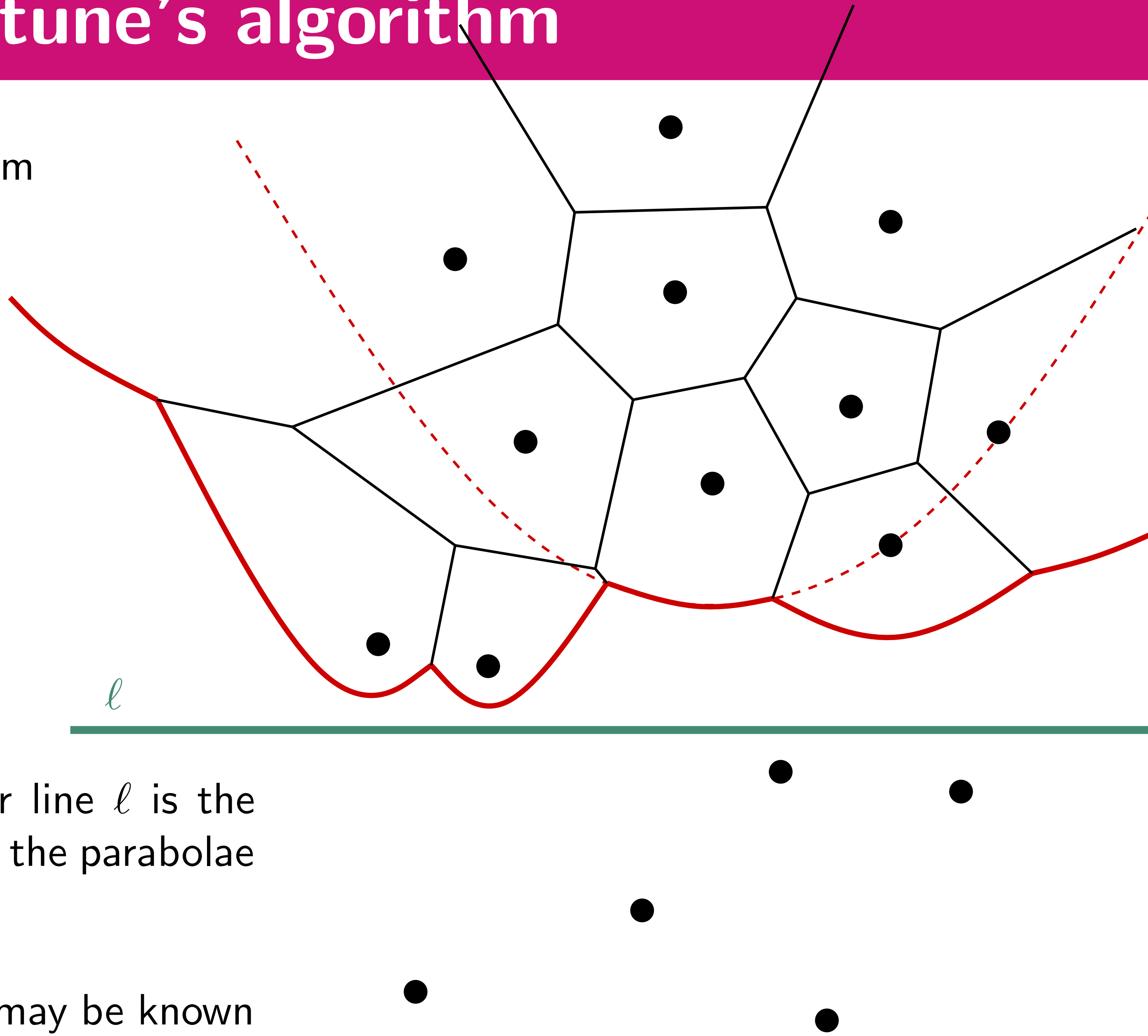
How to compute the Voronoi diagram with a sweep line algorithm?

What can we know?



How to compute the Voronoi diagram with a sweep line algorithm?

What can we know?

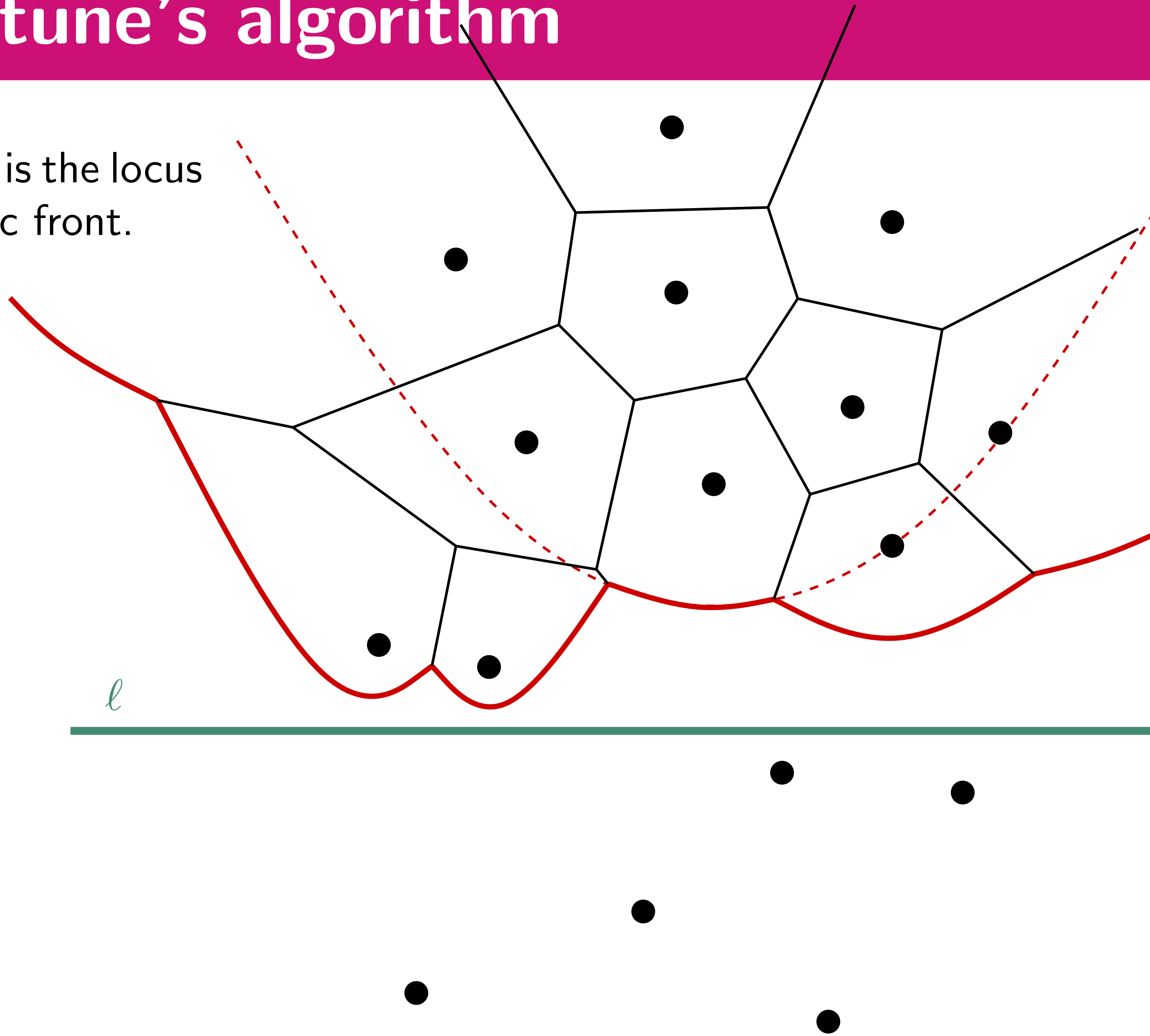


Definition. The *parabolic front* for line ℓ is the lower envelope of the arrangement of the parabolae (p_i, ℓ), for all p_i above ℓ.

Proposition. The Voronoi diagram may be known only above the parabolic front.

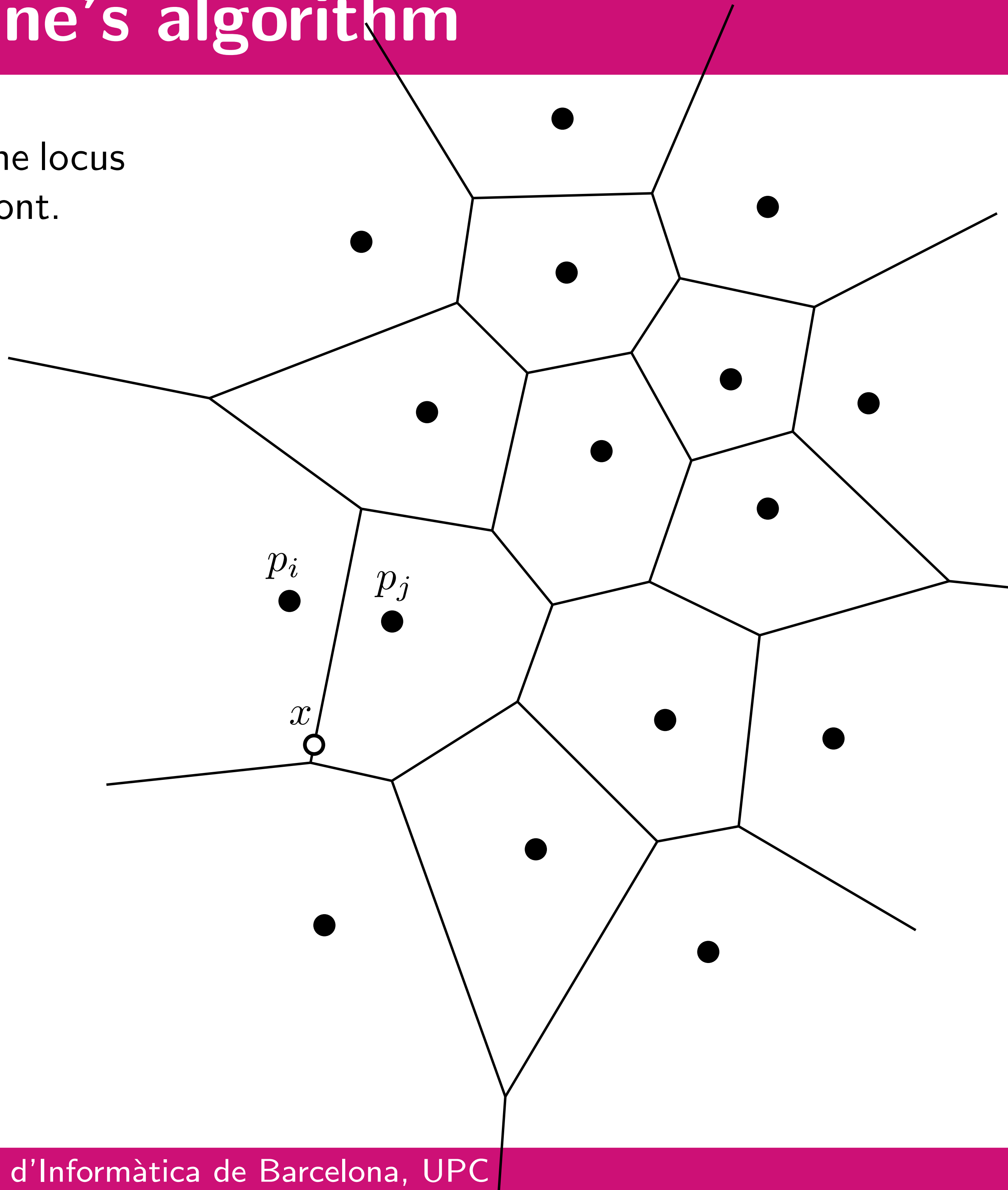
Fortune's algorithm

Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.



Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.

Proof: Let x be a point in edge e_{ij} of $Vor(p_i)$ and $Vor(p_j)$.

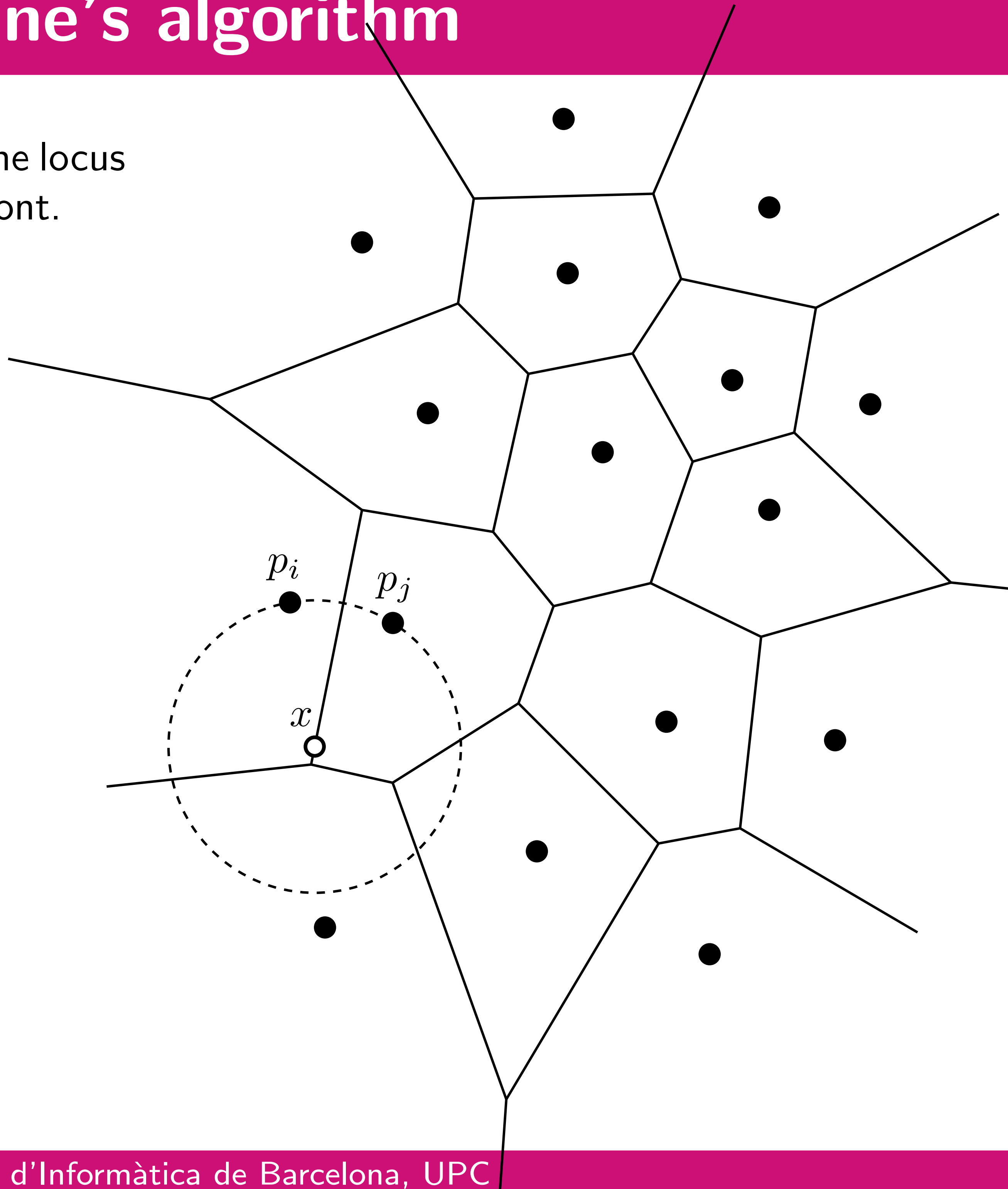


Fortune's algorithm

Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.

Proof: Let x be a point in edge e_{ij} of $Vor(p_i)$ and $Vor(p_j)$.

Consider the disc centered at x with radius $r = d(x, p_i) = d(x, p_j)$.

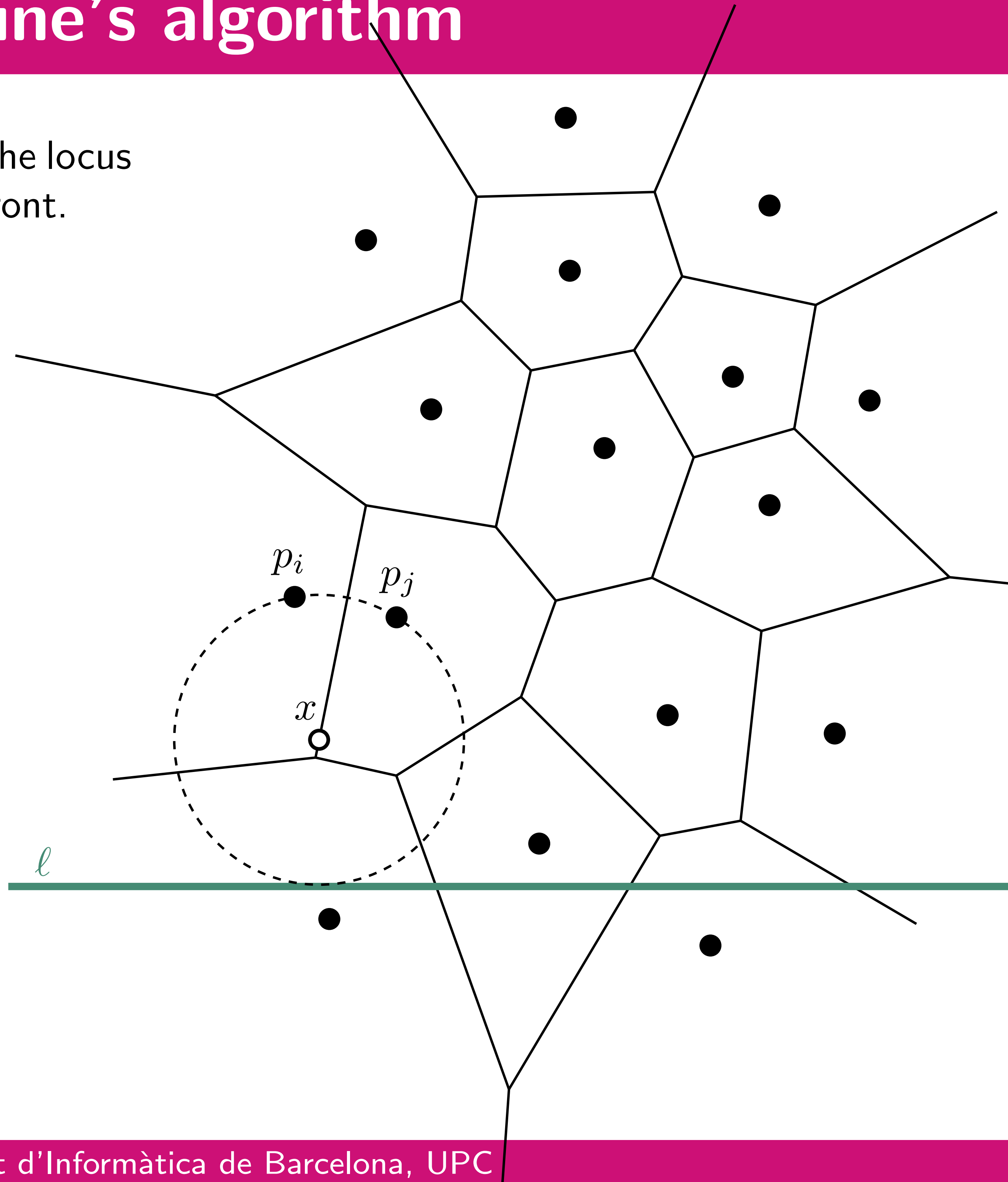


Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.

Proof: Let x be a point in edge e_{ij} of $Vor(p_i)$ and $Vor(p_j)$.

Consider the disc centered at x with radius $r = d(x, p_i) = d(x, p_j)$.

Let ℓ be the bottommost horizontal line tangent to the circle.



Fortune's algorithm

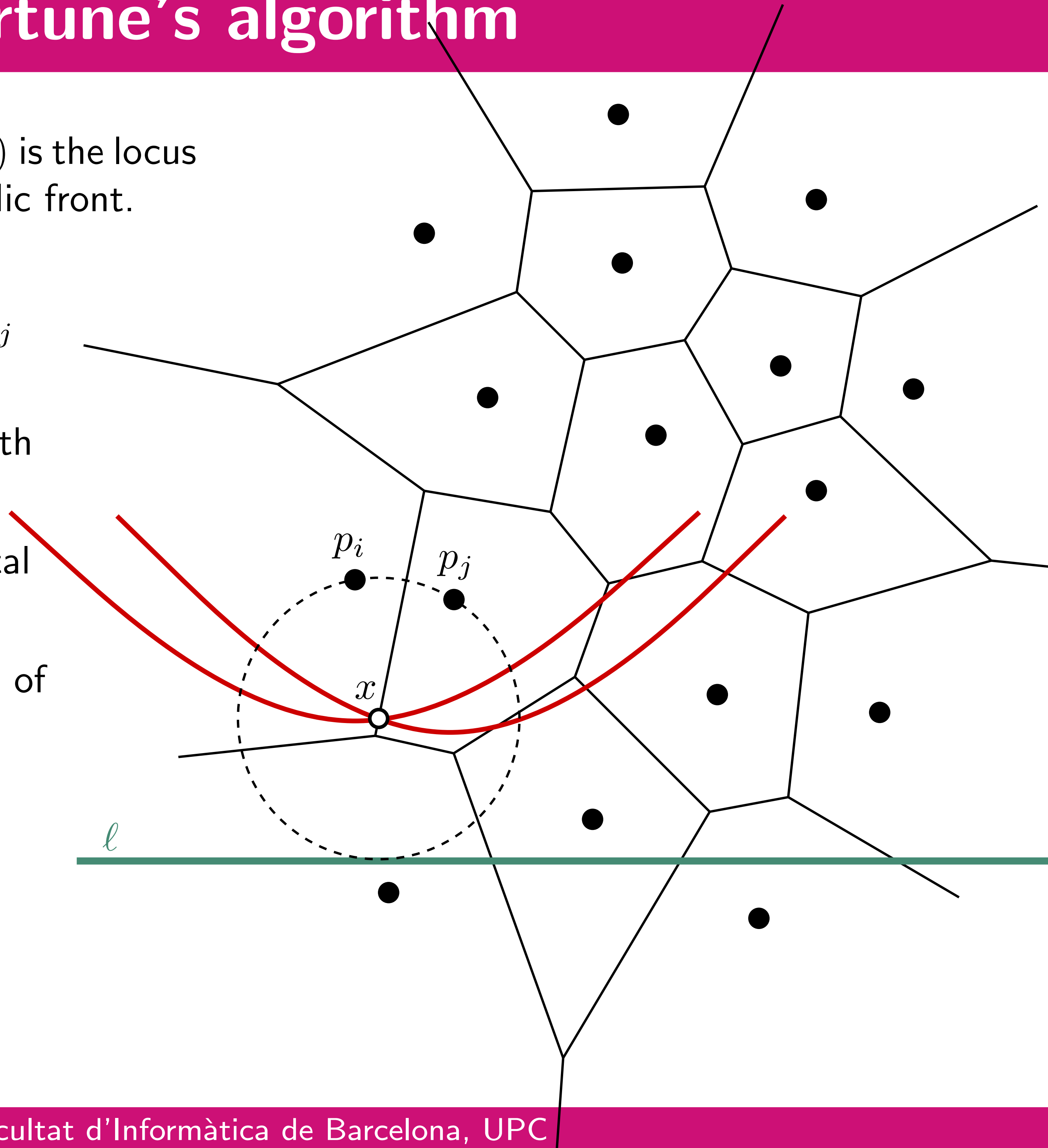
Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.

Proof: Let x be a point in edge e_{ij} of $Vor(p_i)$ and $Vor(p_j)$.

Consider the disc centered at x with radius $r = d(x, p_i) = d(x, p_j)$.

Let ℓ be the bottommost horizontal line tangent to the circle.

Then x belongs to the intersection of the parabolas (p_i, ℓ) and (p_j, ℓ).



Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.

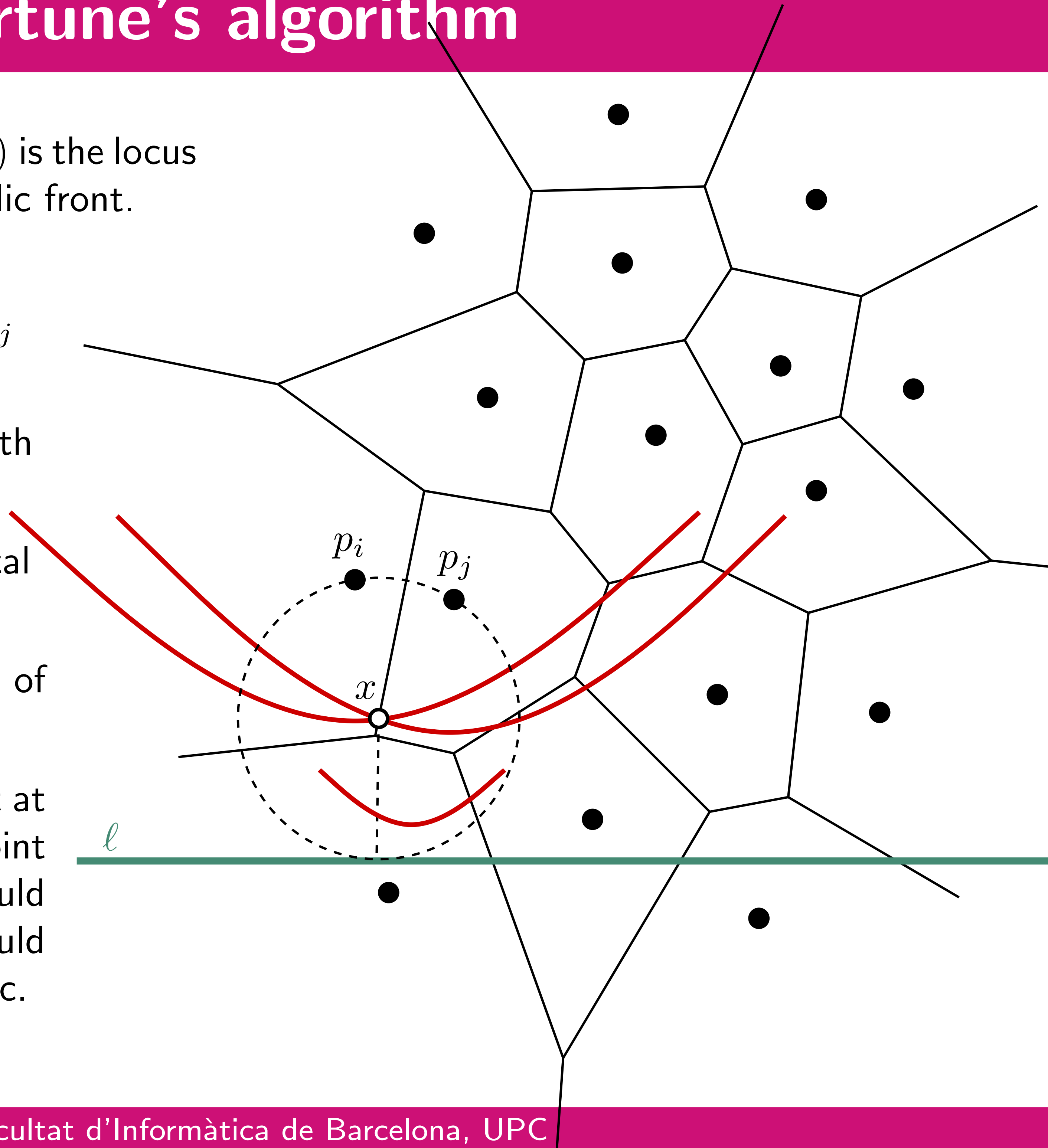
Proof: Let x be a point in edge e_{ij} of $Vor(p_i)$ and $Vor(p_j)$.

Consider the disc centered at x with radius $r = d(x, p_i) = d(x, p_j)$.

Let ℓ be the bottommost horizontal line tangent to the circle.

Then x belongs to the intersection of the parabolas (p_i, ℓ) and (p_j, ℓ).

And x belongs to the parabolic front at this position of ℓ. Otherwise, a point below x in the parabolic front would exist, and the corresponding site would lie in the interior of the (empty) disc.



Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.

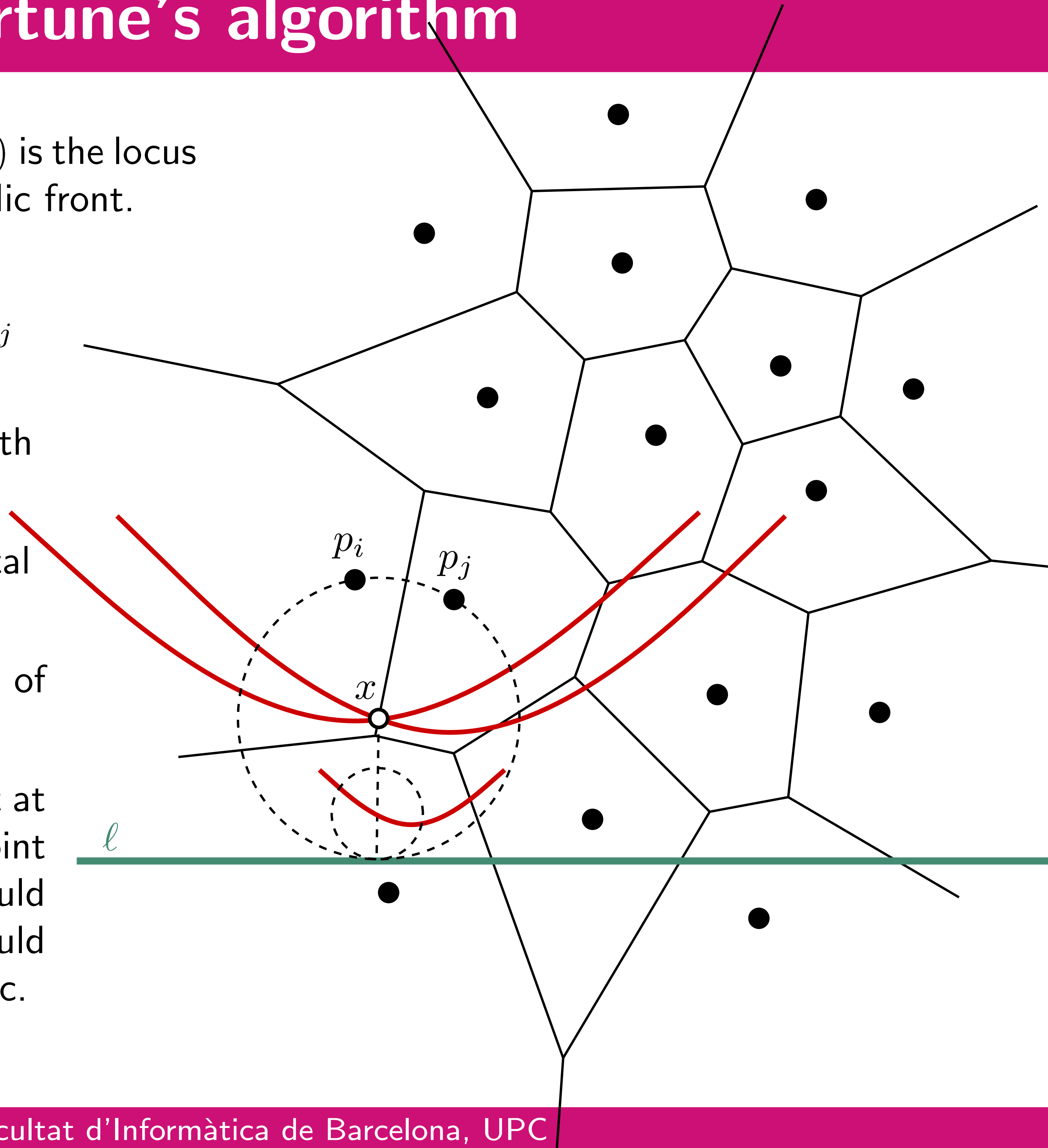
Proof: Let x be a point in edge e_{ij} of $Vor(p_i)$ and $Vor(p_j)$.

Consider the disc centered at x with radius $r = d(x, p_i) = d(x, p_j)$.

Let ℓ be the bottommost horizontal line tangent to the circle.

Then x belongs to the intersection of the parabolas (p_i, ℓ) and (p_j, ℓ).

And x belongs to the parabolic front at this position of ℓ. Otherwise, a point below x in the parabolic front would exist, and the corresponding site would lie in the interior of the (empty) disc.



Fortune's algorithm

Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.

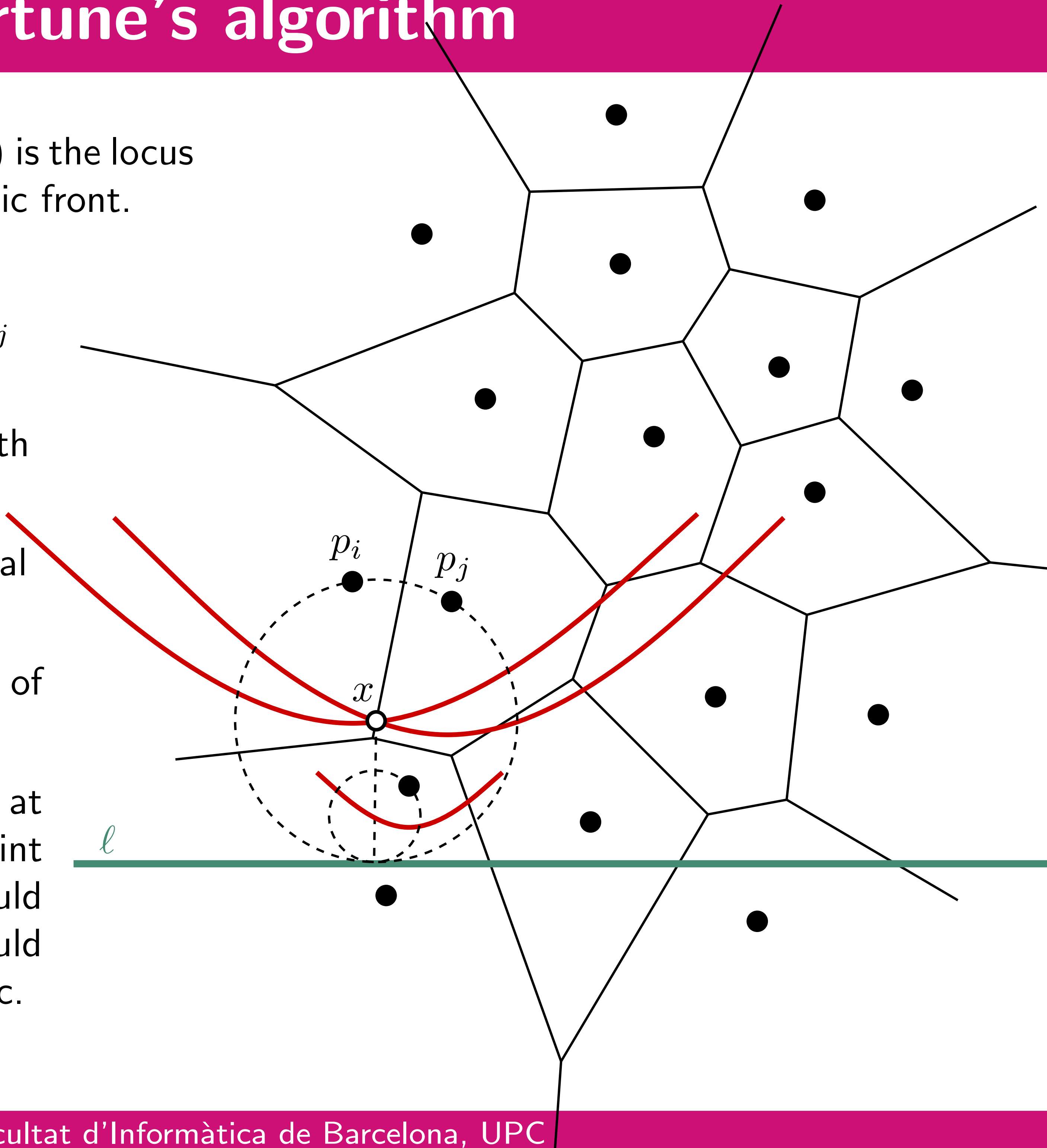
Proof: Let x be a point in edge e_{ij} of $Vor(p_i)$ and $Vor(p_j)$.

Consider the disc centered at x with radius $r = d(x, p_i) = d(x, p_j)$.

Let ℓ be the bottommost horizontal line tangent to the circle.

Then x belongs to the intersection of the parabolas (p_i, ℓ) and (p_j, ℓ).

And x belongs to the parabolic front at this position of ℓ. Otherwise, a point below x in the parabolic front would exist, and the corresponding site would lie in the interior of the (empty) disc.



Fortune's algorithm

Lemma 1. The 1-skeleton of $Vor(P)$ is the locus of all breaking points of the parabolic front.

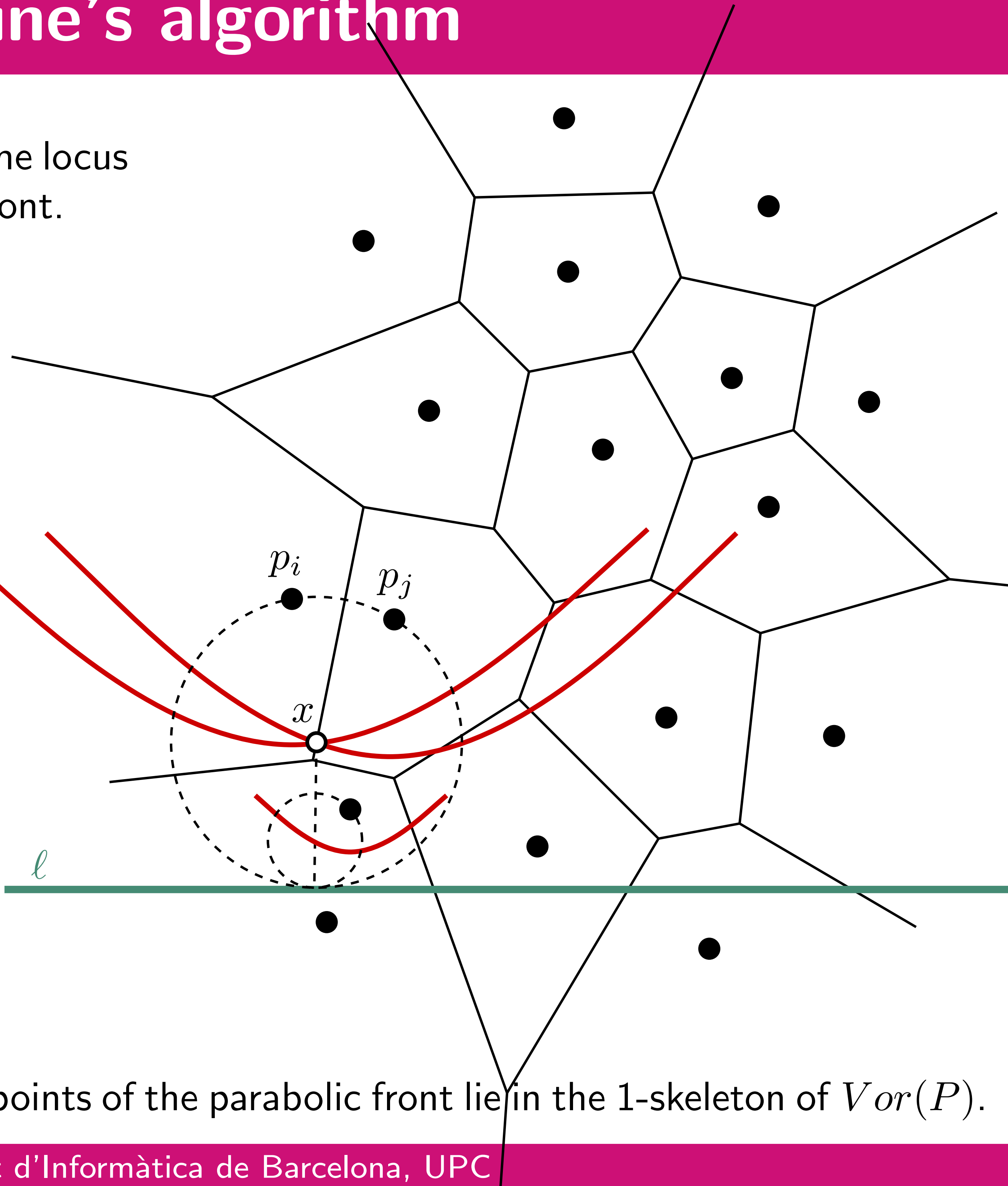
Proof: Let x be a point in edge e_{ij} of $Vor(p_i)$ and $Vor(p_j)$.

Consider the disc centered at x with radius $r = d(x, p_i) = d(x, p_j)$.

Let ℓ be the bottommost horizontal line tangent to the circle.

Then x belongs to the intersection of the parabolas (p_i, ℓ) and (p_j, ℓ).

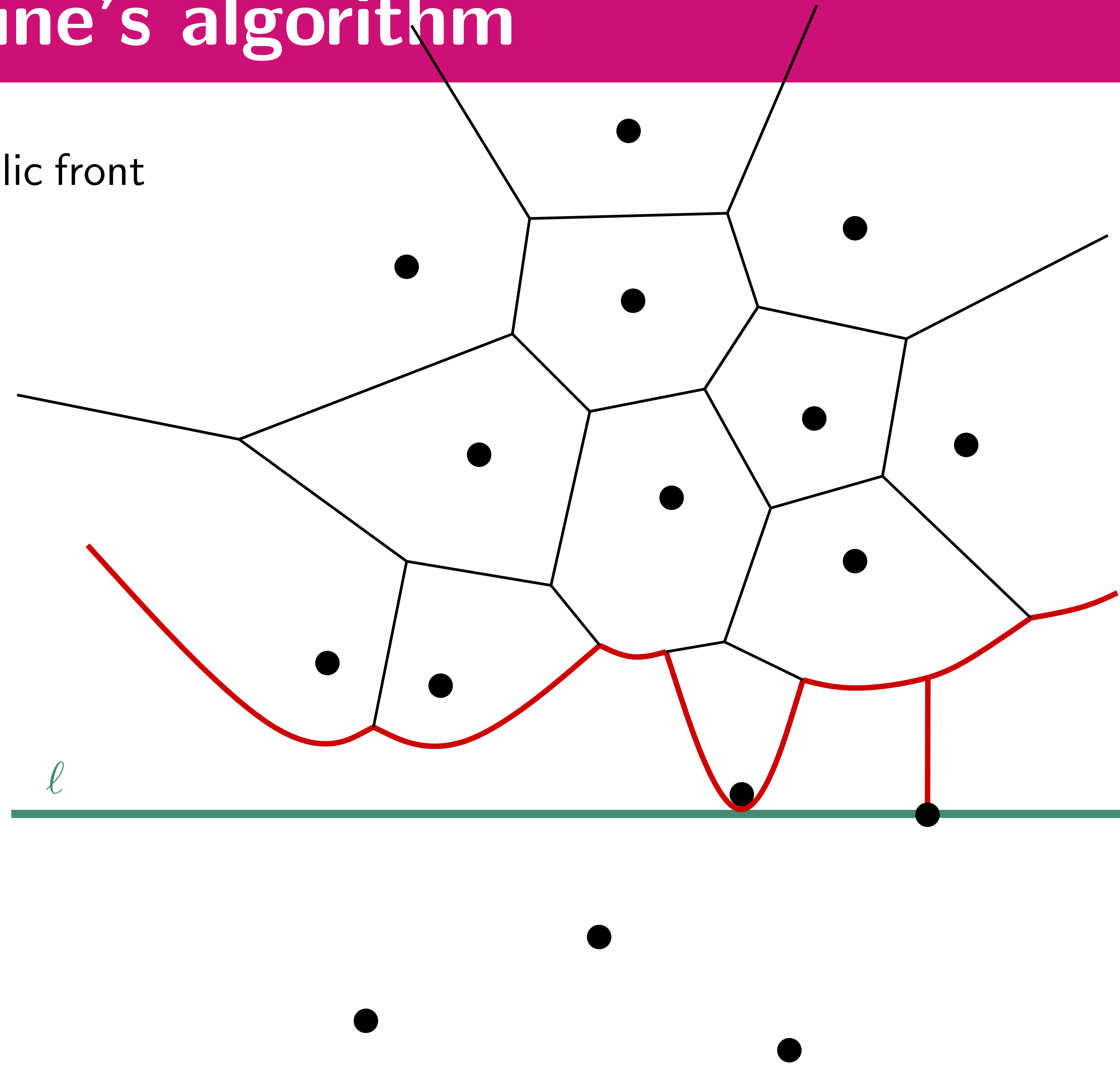
And x belongs to the parabolic front at this position of ℓ. Otherwise, a point below x in the parabolic front would exist, and the corresponding site would lie in the interior of the (empty) disc.



The reverse is trivially true: all breaking points of the parabolic front lie in the 1-skeleton of $Vor(P)$.

Fortune's algorithm

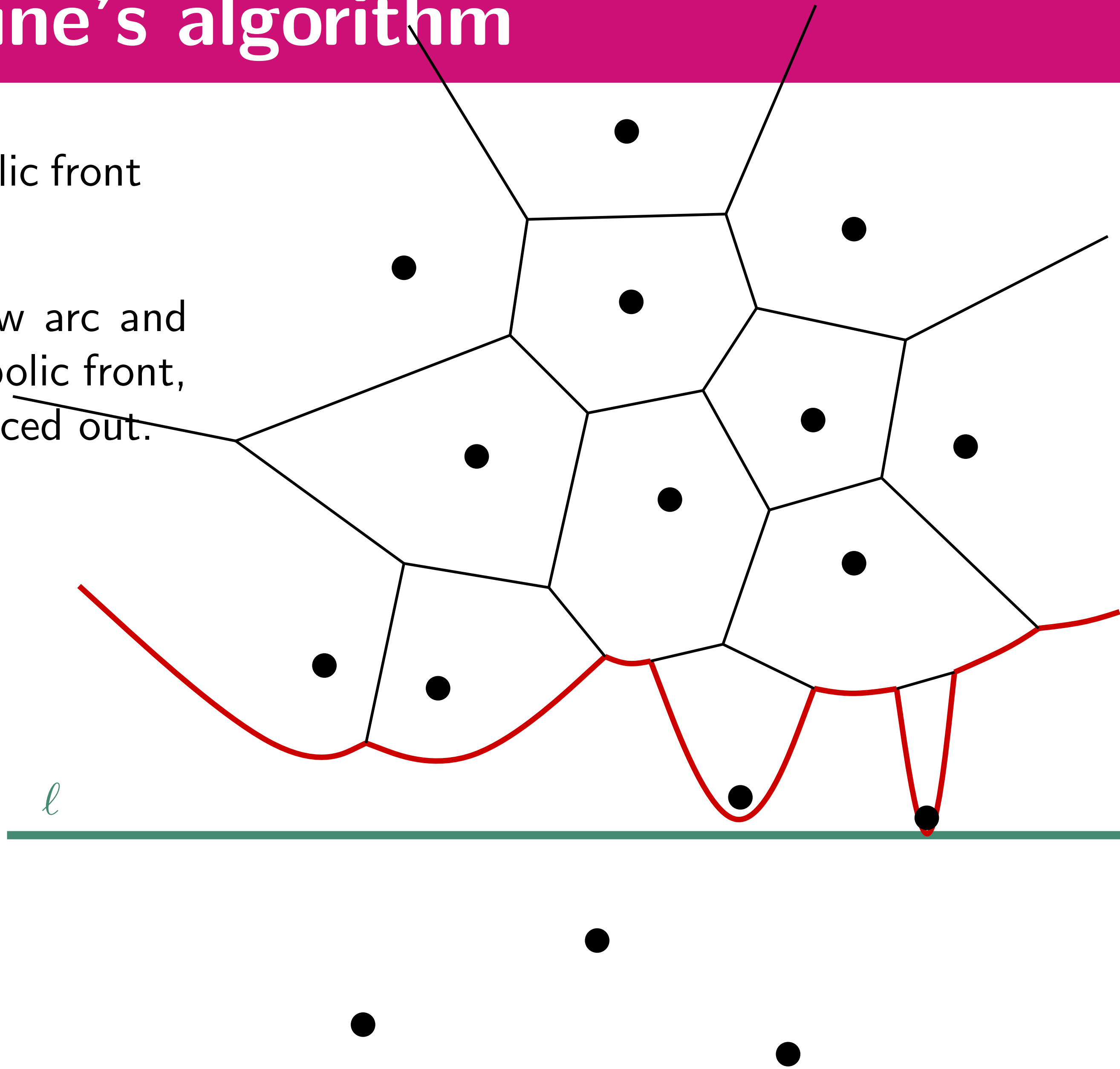
Lemma 2. An arc appears in the parabolic front if and only if ℓ reaches a site.



Fortune's algorithm

Lemma 2. An arc appears in the parabolic front if and only if ℓ reaches a site.

Proof: When ℓ reaches a site p_i, a new arc and two new breakpoints appear in the parabolic front, and a new Voronoi edge starts to be traced out.

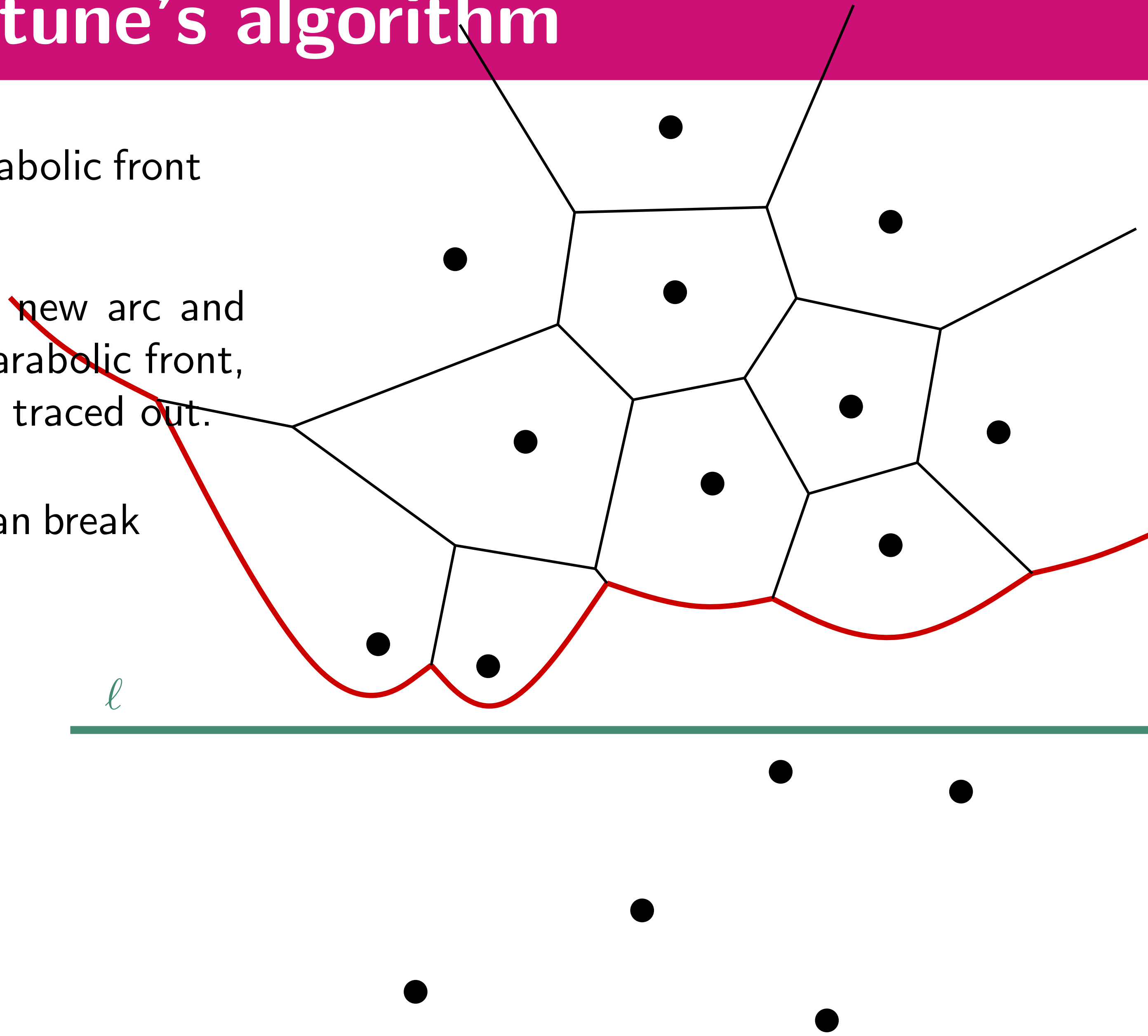


Fortune's algorithm

Lemma 2. An arc appears in the parabolic front if and only if ℓ reaches a site.

Proof: When ℓ reaches a site p_i, a new arc and two new breakpoints appear in the parabolic front, and a new Voronoi edge starts to be traced out.

No already existing parabola (p_j, ℓ) can break through the parabolic front:



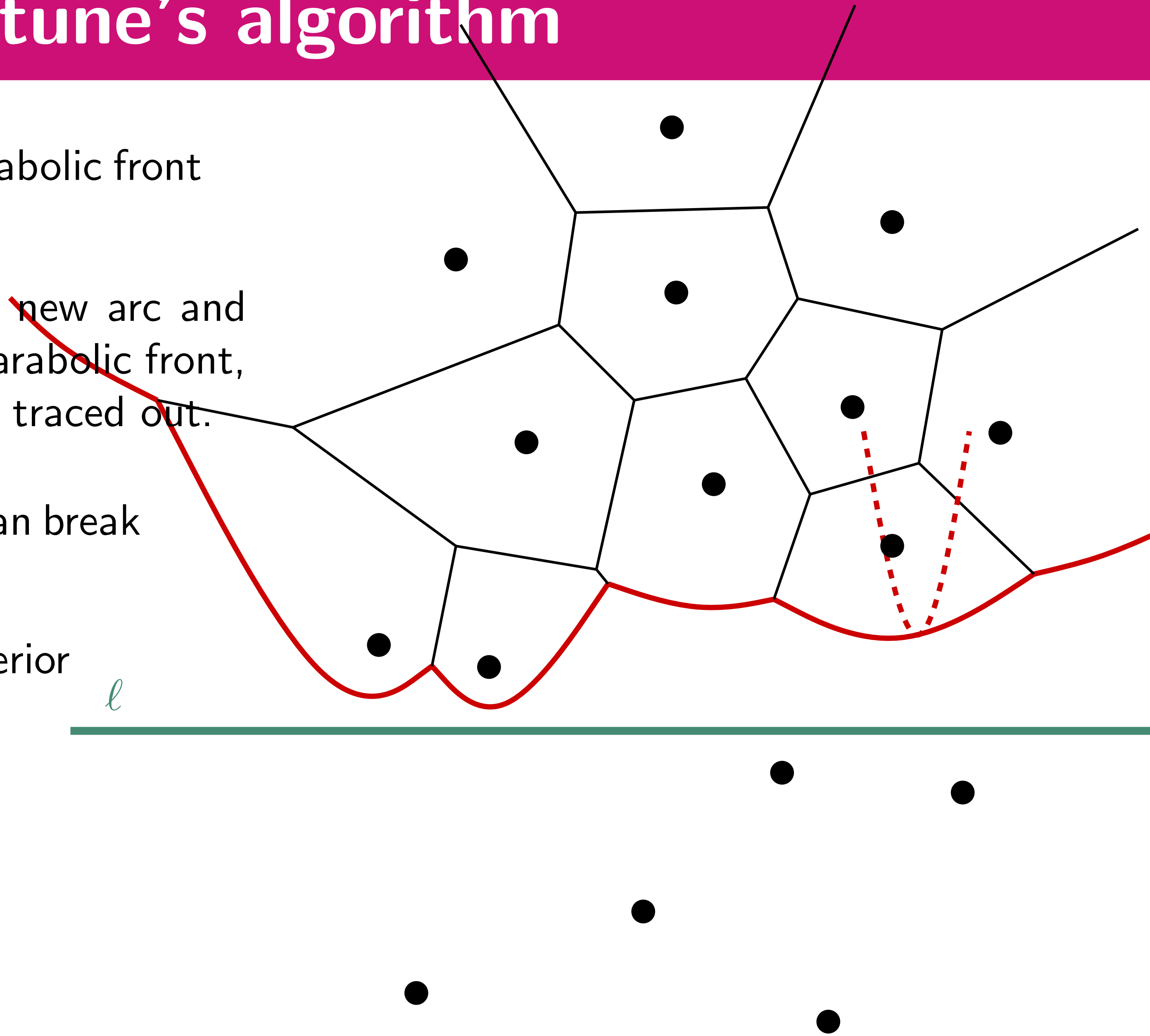
Fortune's algorithm

Lemma 2. An arc appears in the parabolic front if and only if ℓ reaches a site.

Proof: When ℓ reaches a site p_i, a new arc and two new breakpoints appear in the parabolic front, and a new Voronoi edge starts to be traced out.

No already existing parabola (p_j, ℓ) can break through the parabolic front:

- It cannot break through an interior point of an arc (p_i, ℓ)



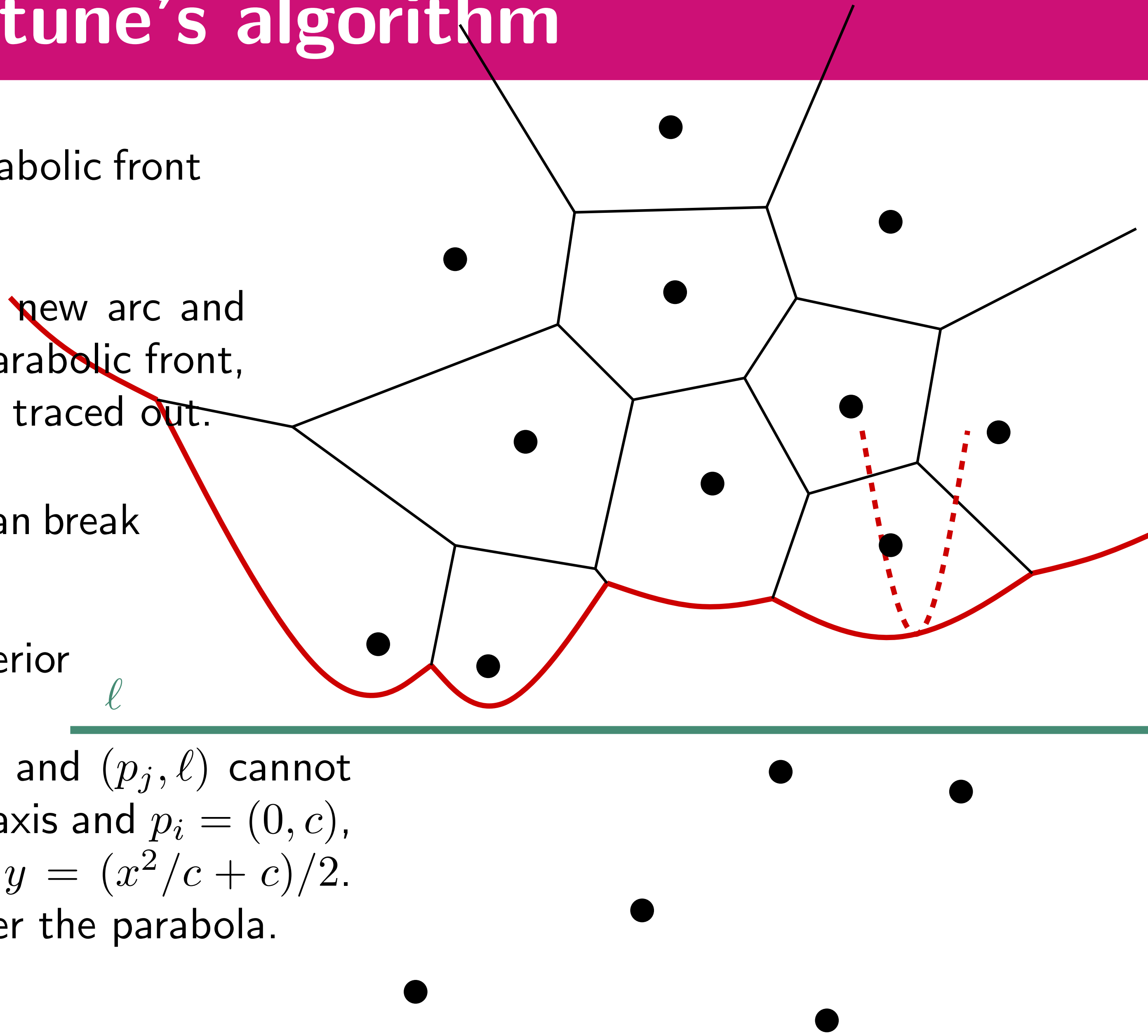
Fortune's algorithm

Lemma 2. An arc appears in the parabolic front if and only if ℓ reaches a site.

Proof: When ℓ reaches a site p_i, a new arc and two new breakpoints appear in the parabolic front, and a new Voronoi edge starts to be traced out.

No already existing parabola (p_j, ℓ) can break through the parabolic front:

- It cannot break through an interior point of an arc (p_i, ℓ)
  ... because two parabolae (p_i, ℓ) and (p_j, ℓ) cannot be tangent. In fact, if ℓ is the x-axis and $p_i = (0, c)$, the equation of the parabola is $y = (x^2/c + c)/2$. Therefore, the higher p_i the wider the parabola.



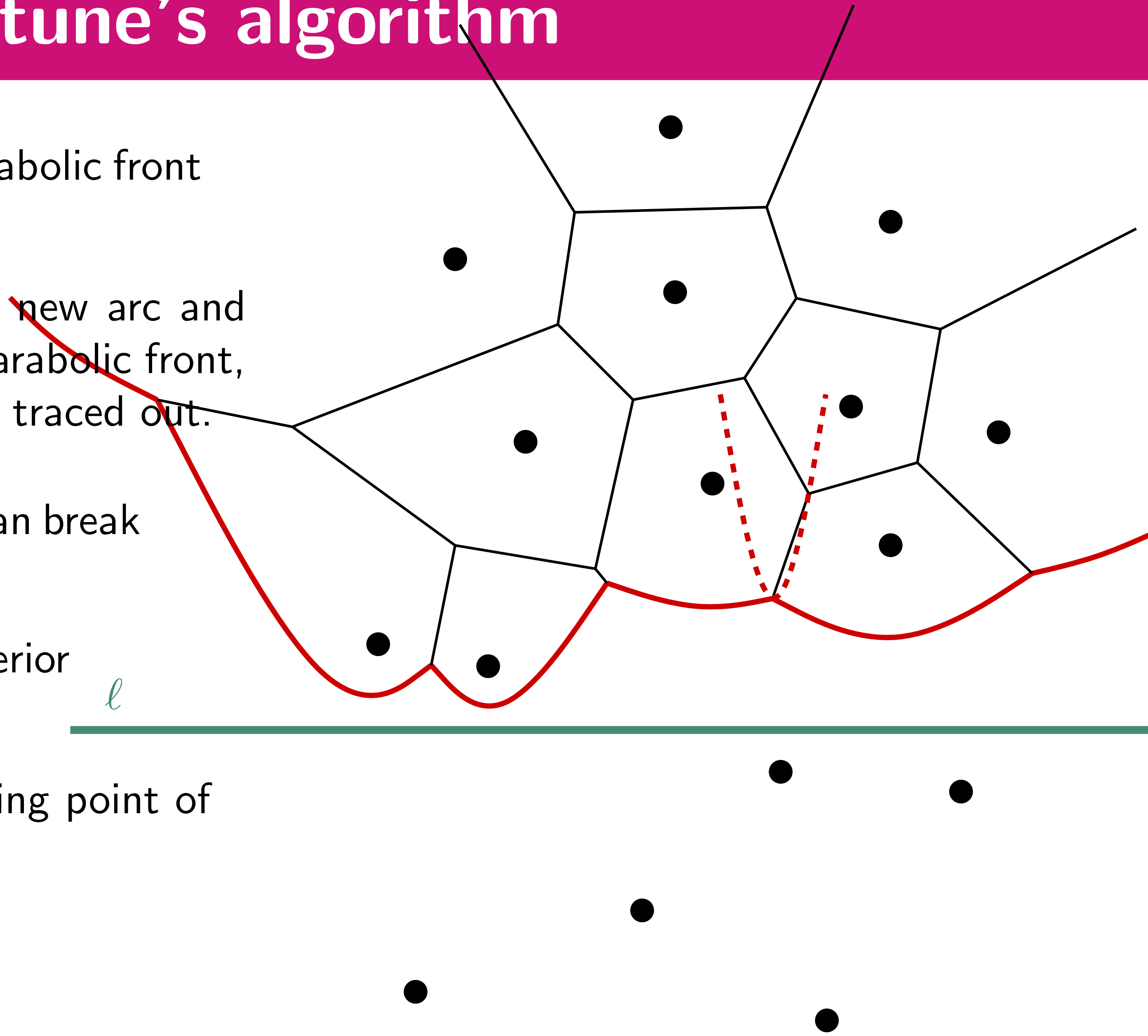
Fortune's algorithm

Lemma 2. An arc appears in the parabolic front if and only if ℓ reaches a site.

Proof: When ℓ reaches a site p_i, a new arc and two new breakpoints appear in the parabolic front, and a new Voronoi edge starts to be traced out.

No already existing parabola (p_j, ℓ) can break through the parabolic front:

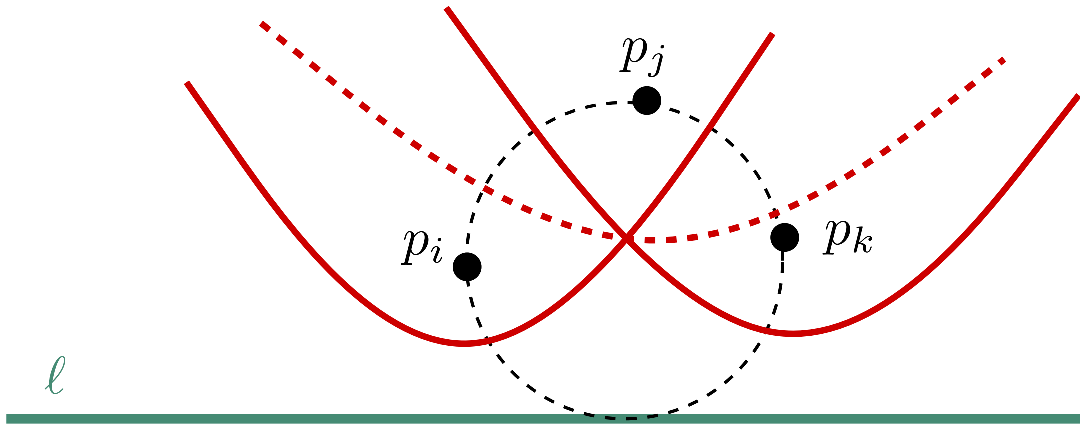
- It cannot break through an interior point of an arc (p_i, ℓ)
- It cannot break through a breaking point of two arcs $(p_i \ell)$ and (p_k, ℓ)



Fortune's algorithm

Lemma 2. An arc appears in the parabolic front if and only if ℓ reaches a site.

Proof: When ℓ reaches a site p_i, a new arc and two new breakpoints appear in the parabolic front, and a new Voronoi edge starts to be traced out.



No already existing parabola (p_j, ℓ) can break through the parabolic front:

- It cannot break through an interior point of an arc (p_i, ℓ)
- It cannot break through a breaking point of two arcs $(p_i \ell)$ and (p_k, ℓ)

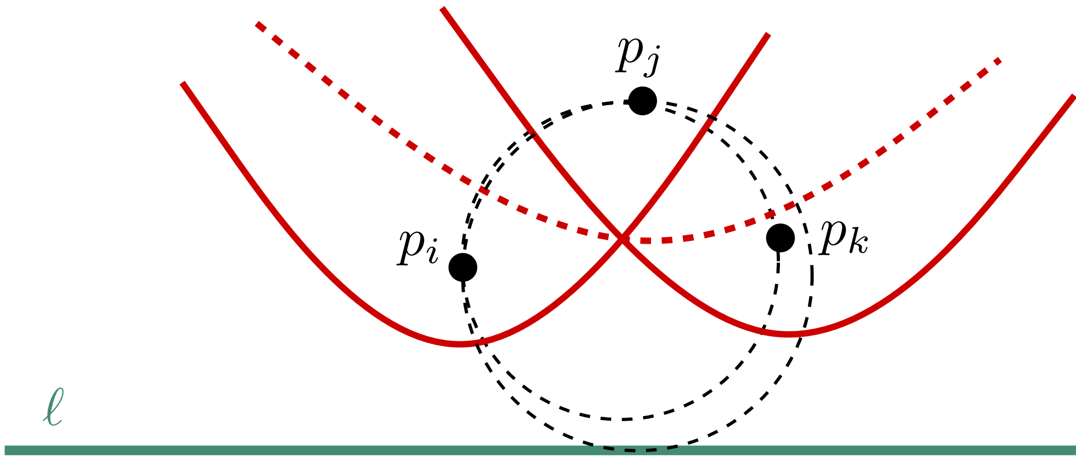
  ... because there would be an empty cicle through p_i, p_j, p_k, tangent to ℓ. Moving ℓ further down would make (p_j, ℓ) disappear from the parabolic front, as the circles through p_i, p_j, ℓ and p_j, p_k, ℓ would not be empty.

Fortune's algorithm

Lemma 2. An arc appears in the parabolic front if and only if ℓ reaches a site.

Proof: When ℓ reaches a site p_i, a new arc and two new breakpoints appear in the parabolic front, and a new Voronoi edge starts to be traced out.

No already existing parabola (p_j, ℓ) can break through the parabolic front:



- It cannot break through an interior point of an arc (p_i, ℓ)
- It cannot break through a breaking point of two arcs $(p_i \ell)$ and (p_k, ℓ)

  ... because there would be an empty cicle through p_i, p_j, p_k, tangent to ℓ. Moving ℓ further down would make (p_j, ℓ) disappear from the parabolic front, as the circles through p_i, p_j, ℓ and p_j, p_k, ℓ would not be empty.

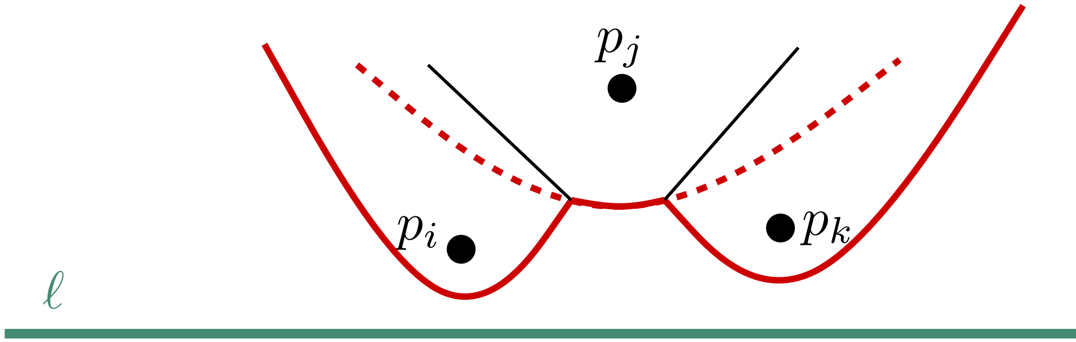
Fortune's algorithm

Lemma 3. An arc (p_j, ℓ) disappears from the parabolic front only if ℓ leaves the disc through p_j and the sites corresponding to the two adjacent arcs.

Fortune's algorithm

Lemma 3. An arc (p_j, ℓ) disappears from the parabolic front only if ℓ leaves the disc through p_j and the sites corresponding to the two adjacent arcs.

Proof: Let $(p_i, \ell), (p_j, \ell), (p_k, \ell)$ be consecutive arcs of the parabolic front. Notice that $k \neq i$ for the same reason as in the first case of Lemma 2.

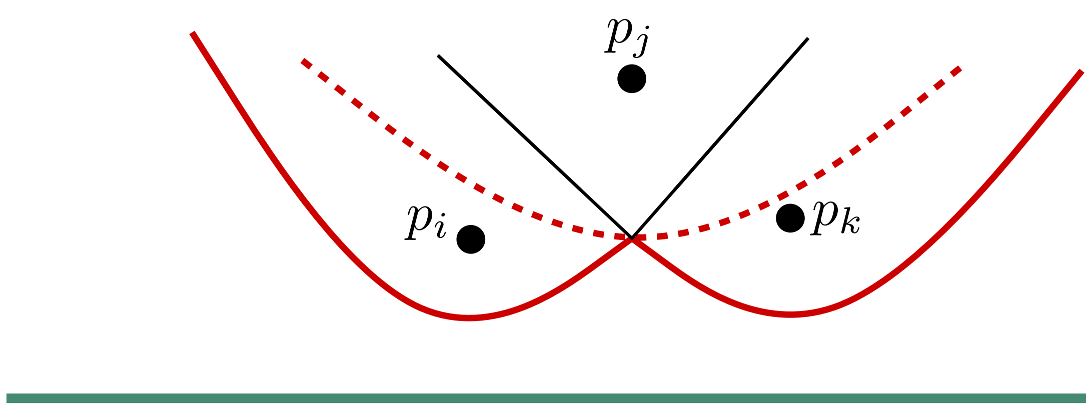


Fortune's algorithm

Lemma 3. An arc (p_j, ℓ) disappears from the parabolic front only if ℓ leaves the disc through p_j and the sites corresponding to the two adjacent arcs.

Proof: Let $(p_i, \ell), (p_j, \ell), (p_k, \ell)$ be consecutive arcs of the parabolic front. Notice that $k \neq i$ for the same reason as in the first case of Lemma 2.

The moment (p_j, ℓ) disappears, the three arcs go through a common point q, equidistant from p_i, p_j, p_k, ℓ.



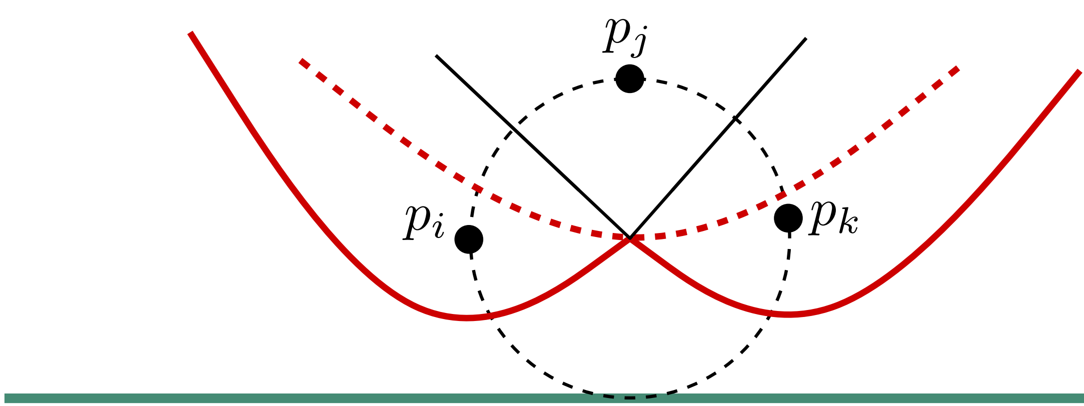
Fortune's algorithm

Lemma 3. An arc (p_j, ℓ) disappears from the parabolic front only if ℓ leaves the disc through p_j and the sites corresponding to the two adjacent arcs.

Proof: Let $(p_i, \ell), (p_j, \ell), (p_k, \ell)$ be consecutive arcs of the parabolic front. Notice that $k \neq i$ for the same reason as in the first case of Lemma 2.

The moment (p_j, ℓ) disappears, the three arcs go through a common point q, equidistant from p_i, p_j, p_k, ℓ.

Therefore, the circle though p_i, p_j, p_k is tangent to ℓ.



Fortune's algorithm

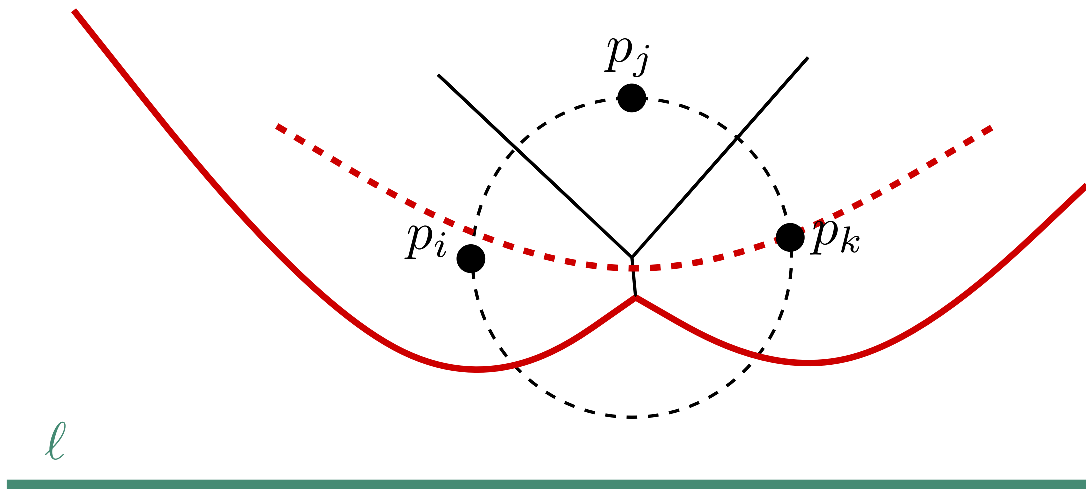
Lemma 3. An arc (p_j, ℓ) disappears from the parabolic front only if ℓ leaves the disc through p_j and the sites corresponding to the two adjacent arcs.

Proof: Let $(p_i, \ell), (p_j, \ell), (p_k, \ell)$ be consecutive arcs of the parabolic front. Notice that $k \neq i$ for the same reason as in the first case of Lemma 2.

The moment (p_j, ℓ) disappears, the three arcs go through a common point q, equidistant from p_i, p_j, p_k, ℓ.

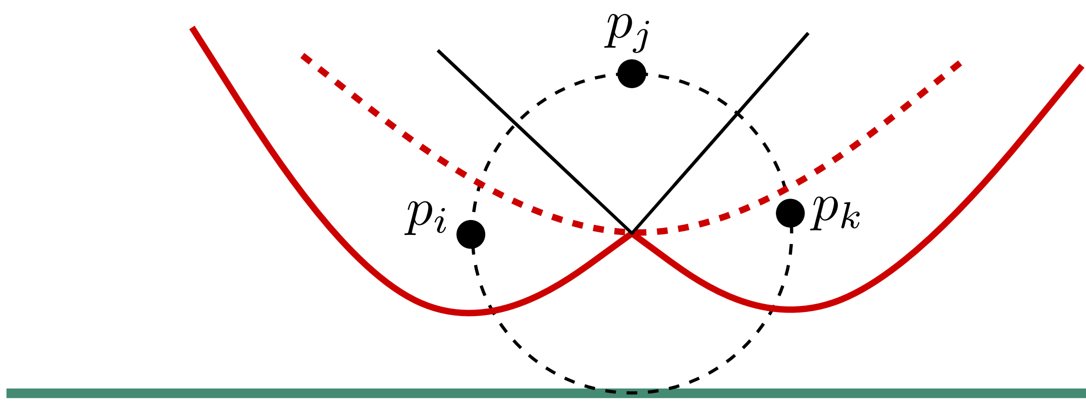
Therefore, the circle though p_i, p_j, p_k is tangent to ℓ.

In fact, the circle is empty and a Vornoi vertex has been found.



Fortune's algorithm

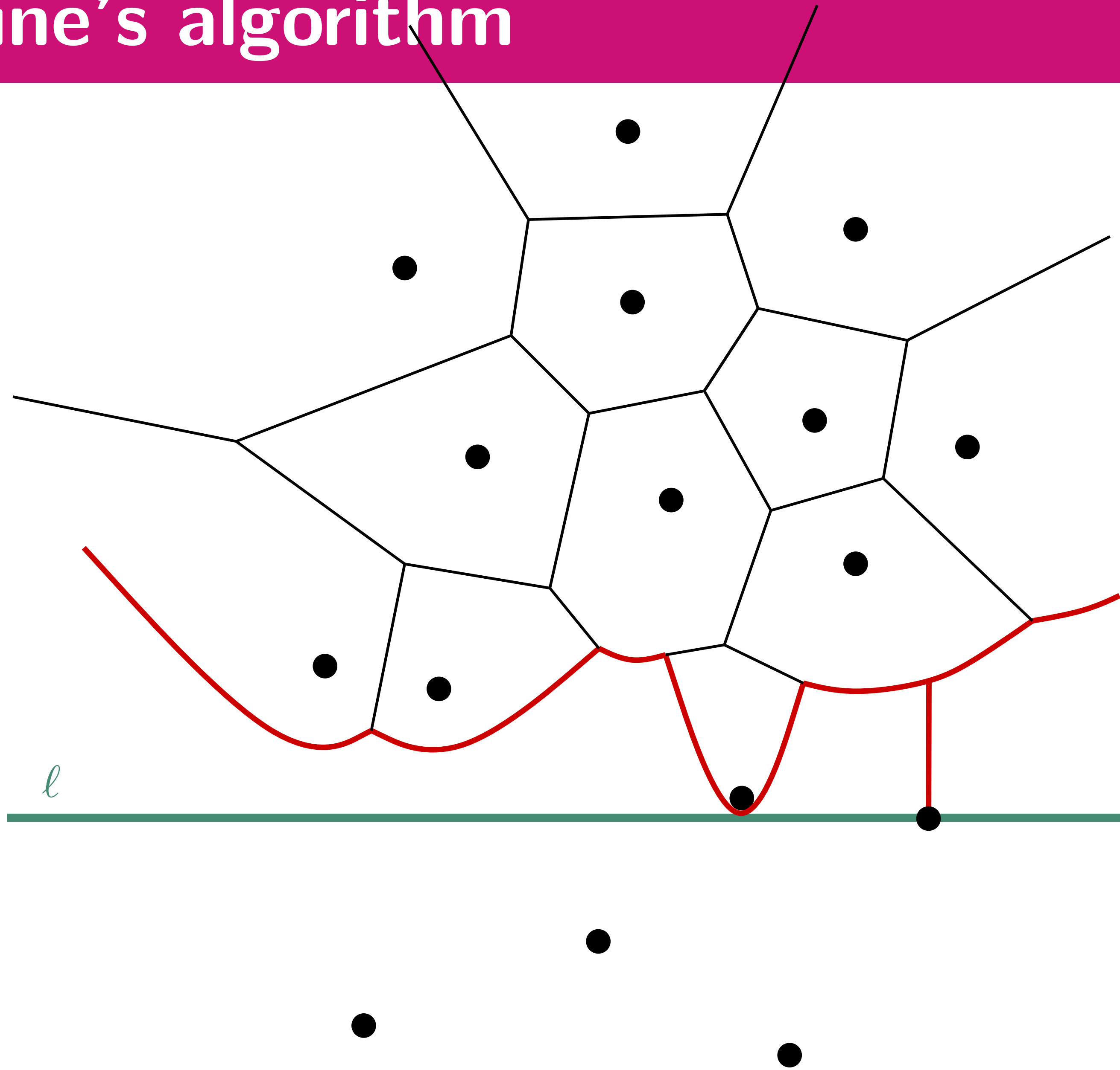
Corollary 1. Every Voronoi vertex is detected in a circle event.



Fortune's algorithm

Corollary 1. Every Voronoi vertex is detected in a circle event.

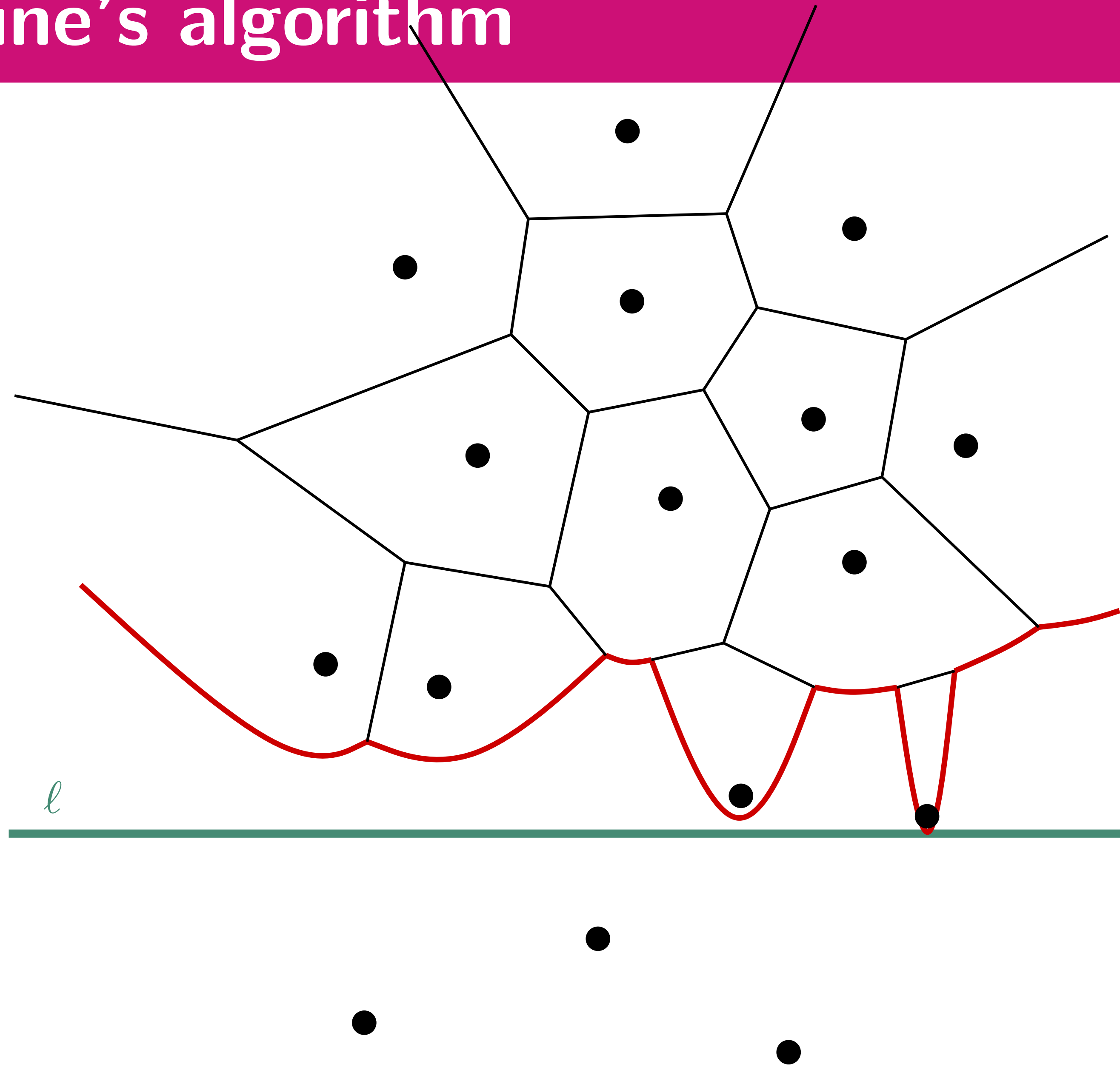
Corollary 2. The parabolic front consists of at most $2n - 1$ arcs.



Fortune's algorithm

Corollary 1. Every Voronoi vertex is detected in a circle event.

Corollary 2. The parabolic front consists of at most $2n - 1$ arcs.



Fortune's algorithm

Sweep line status: **parabolic front**

Events queue: **site events and circle events**

Output: **Voronoi diagram**

Fortune's algorithm

Sweep line status: **parabolic front**

The sweep line status is stored in a balanced binary search tree in which leaves are arcs and internal nodes are breakpoints of the parabolic front, sorted by x-coordinate.

- Each arc α of a parablola (p_i, ℓ) is represented by the index i of the site p_i, and it has a pointer to the circle event for α.
- Each breaking point is represented by a pair of indices (i, j), and has a pointer to the corresponding Voronoi edge.

Events queue: **site events and circle events**

Output: **Voronoi diagram**

Fortune's algorithm

Sweep line status: parabolic front

The sweep line status is stored in a balanced binary search tree in which leaves are arcs and internal nodes are breakpoints of the parabolic front, sorted by x-coordinate.

- Each arc α of a parablola (p_i, ℓ) is represented by the index i of the site p_i, and it has a pointer to the circle event for α.
- Each breaking point is represented by a pair of indices (i, j), and has a pointer to the corresponding Voronoi edge.

Events queue: site events and circle events

The events are stored in a priority queue, where the priority of an event is its y-coordinate.

- For every site event $p_i = (x_i, y_i)$, the point itself is stored.
- For every circle event, the lowest point of the circle is stored.

Output: Voronoi diagram

Fortune's algorithm

Sweep line status: parabolic front

The sweep line status is stored in a balanced binary search tree in which leaves are arcs and internal nodes are breakpoints of the parabolic front, sorted by x-coordinate.

- Each arc α of a parablola (p_i, ℓ) is represented by the index i of the site p_i, and it has a pointer to the circle event for α.
- Each breaking point is represented by a pair of indices (i, j), and has a pointer to the corresponding Voronoi edge.

Events queue: site events and circle events

The events are stored in a priority queue, where the priority of an event is its y-coordinate.

- For every site event $p_i = (x_i, y_i)$, the point itself is stored.
- For every circle event, the lowest point of the circle is stored.

Output: Voronoi diagram

The Voronoi diagram is stored in a DCEL, specifically dealing with halflines.

Fortune's algorithm

ALGORITHM

Input: A set $P = \{p_1, \ldots, p_n\}$ of n sites in the plane.
Output: $Vor(P)$

Fortune's algorithm

ALGORITHM

Input: A set $P = \{p_1, \ldots, p_n\}$ of n sites in the plane.
Output: $Vor(P)$

Initialize

$E = \{p_1, \ldots, p_n\}$, sorted by decreasing y-coordinate.

$S = \{ \}$

$Vor(P) = \{ \}$

Fortune's algorithm

ALGORITHM

Input: A set $P = \{p_1, \ldots, p_n\}$ of n sites in the plane.
Output: $Vor(P)$

Initialize

$E = \{p_1, \ldots, p_n\}$, sorted by decreasing y-coordinate.

$S = \{ \}$

$Vor(P) = \{ \}$

Advance

While E is not empty:
- Extract the first event from E.
- Handle the event, depending on whether it is a site or a circle event.

Fortune's algorithm

ALGORITHM

Input: A set $P = \{p_1, \ldots, p_n\}$ of n sites in the plane.
Output: $Vor(P)$

Initialize

$E = \{p_1, \ldots, p_n\}$, sorted by decreasing y-coordinate.

$S = \{ \}$

$Vor(P) = \{ \}$

Advance

While E is not empty:
- Extract the first event from E.
- Handle the event, depending on whether it is a site or a circle event.

Complete

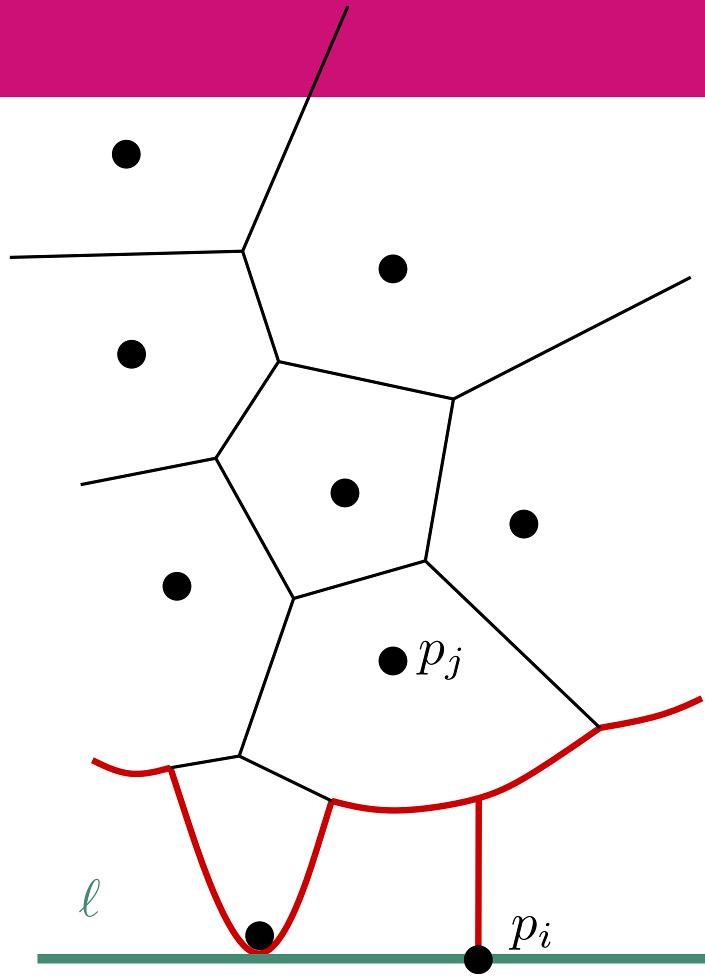
Compute a bounding box of the Vornoi vertices.
Traverse the unbounded faces to update the DCEL appropriately.

Fortune's algorithm

ALGORITHM

Handle a site event p_i

1. Search in the sweep line S for the arc α vertically above p_i. Let p_j be its site.
2. Replace in S the leaf α by a subtree with three leaves: p_j, p_i, p_j and two internal nodes $(p_j, p_i), (p_i, p_j)$. Rebalance S if necessary.
3. Update the events queue:
   - Delete from E all circle events involving α.
   - Insert in E, if necessary, the circle events of all triples involving the three new leaves.
4. Update the Voronoi diagram:
   - Create a new face, with a pointer to the new edge.
   - Create a new edge, with pointers to the incident faces.

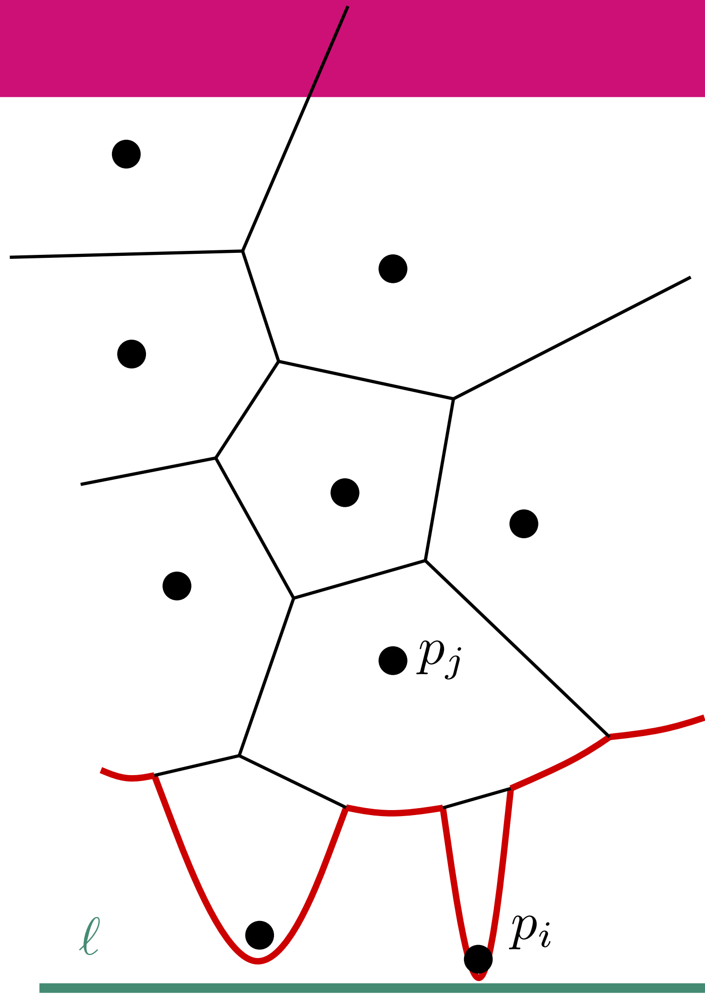


Fortune's algorithm

ALGORITHM

Handle a site event p_i

1. Search in the sweep line S for the arc α vertically above p_i. Let p_j be its site.
2. Replace in S the leaf α by a subtree with three leaves: p_j, p_i, p_j and two internal nodes $(p_j, p_i), (p_i, p_j)$. Rebalance S if necessary.
3. Update the events queue:
   - Delete from E all circle events involving α.
   - Insert in E, if necessary, the circle events of all triples involving the three new leaves.
4. Update the Voronoi diagram:
   - Create a new face, with a pointer to the new edge.
   - Create a new edge, with pointers to the incident faces.



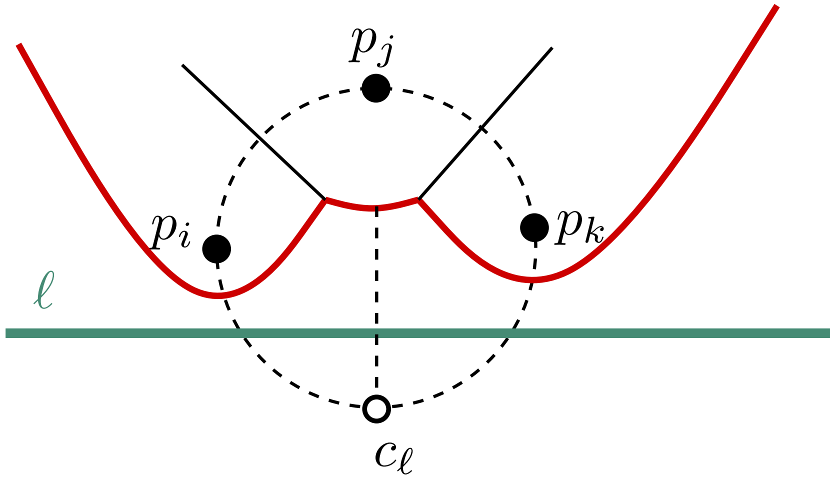
Fortune's algorithm

ALGORITHM

Handle a circle event c_ℓ

1. Search in the sweep line S for the arc α vertically above c_ℓ. Let p_j be its site. Let p_i, p_k respectively be the previous and the next sites in S.

2. Delete in S the leaf α and replace the internal nodes $(p_i, p_j), (p_j, p_k)$ by (p_i, p_k). Rebalance S if necessary.



3. Update the events queue:

   - Delete from E all circle events involving α.
   - Insert in E, if necessary, the circle events of all triples involving the new consecutive arcs.

4. Update the Voronoi diagram:

   - Create a new vertex (the center of the circle), with a pointer to the new edge.
   - Complete the information of the two ending edges: final vertex and next edges.
   - Create a new edge, with pointers to its starting vertex, previous edge, and incident faces.

Fortune's algorithm

ALGORITHM

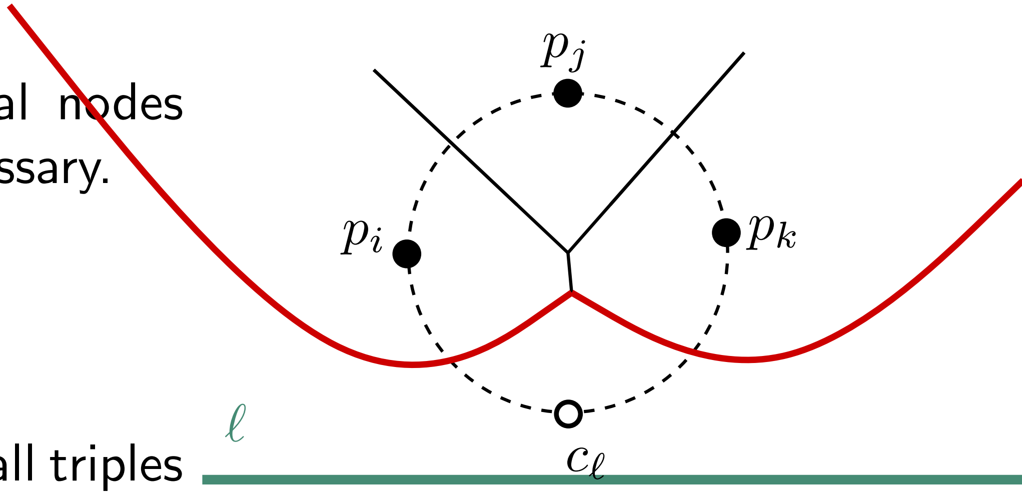
Handle a circle event c_ℓ

1. Search in the sweep line S for the arc α vertically above c_ℓ. Let p_j be its site. Let p_i, p_k respectively be the previous and the next sites in S.

2. Delete in S the leaf α and replace the internal nodes $(p_i, p_j), (p_j, p_k)$ by (p_i, p_k). Rebalance S if necessary.

3. Update the events queue:

    - Delete from E all circle events involving α.
    - Insert in E, if necessary, the circle events of all triples involving the new consecutive arcs.



4. Update the Voronoi diagram:

    - Create a new vertex (the center of the circle), with a pointer to the new edge.
    - Complete the information of the two ending edges: final vertex and next edges.
    - Create a new edge, with pointers to its starting vertex, previous edge, and incident faces.

Fortune's algorithm

COMPLEXITY

The algorithm runs in $O(n \log n)$ time and uses $O(n)$ space.

Fortune's algorithm

COMPLEXITY

The algorithm runs in $O(n \log n)$ time and uses $O(n)$ space.

Proof. As for running time:

- Each primitive operation on the events queue E and the sweep line statuts S takes $O(\log n)$ time. Each operation on the Voronoi diagram takes $O(1)$ time.
- Handling each event requires a constant number of such primitive oprations.
- The number of site events is n. The number of circle events actually processed is $2n-3$, each corresponding to a Voronoi vertex (false alarms are not processed, as they are deleted before they are processed).

As for space:

- The sweep line complexity is $\leq 2n-1$.
- The events queue stores $\leq 3n-3$ events.
- The DCEL structure storing the Voronoi diagram uses $O(n)$ space.

Fortune's algorithm

COMPLEXITY

The algorithm runs in $O(n \log n)$ time and uses $O(n)$ space.

Proof. As for running time:

- Each primitive operation on the events queue E and the sweep line statuts S takes $O(\log n)$ time. Each operation on the Voronoi diagram takes $O(1)$ time.
- Handling each event requires a constant number of such primitive oprations.
- The number of site events is n. The number of circle events actually processed is $2n-3$, each corresponding to a Voronoi vertex (false alarms are not processed, as they are deleted before they are processed).

As for space:

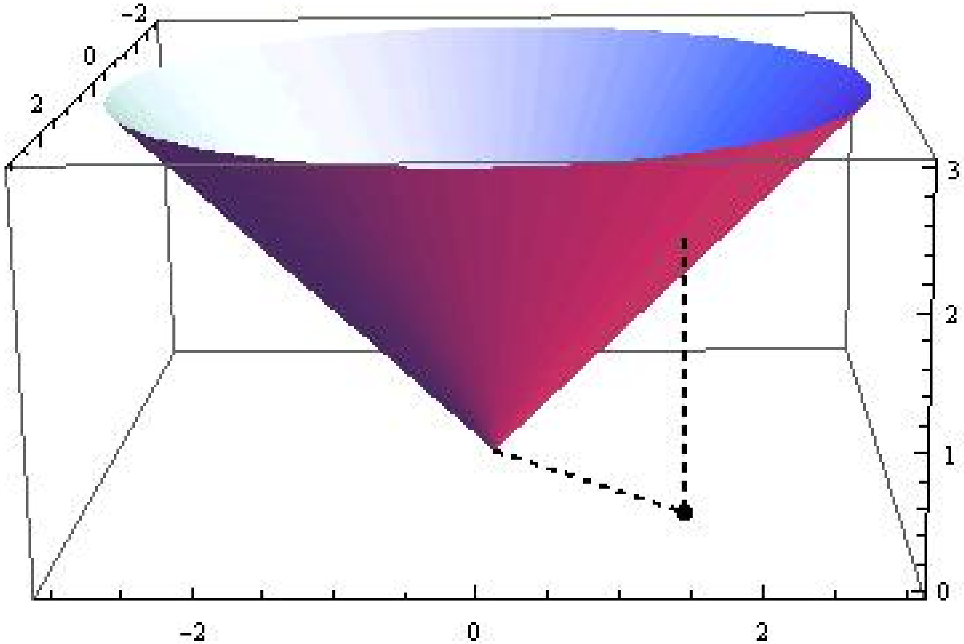
- The sweep line complexity is $\leq 2n-1$.
- The events queue stores $\leq 3n-3$ events.
- The DCEL structure storing the Voronoi diagram uses $O(n)$ space.

Degeneracies can be handled within the same complexity.

Fortune's algorithm

3-DIMENSIONAL INTERPRETATION

3-DIMENSIONAL INTERPRETATION



Consider the point set P to be embedded in the plane $z = 0$.

For each site $p_i = (x_i, y_i)$, consider the cone C_i:

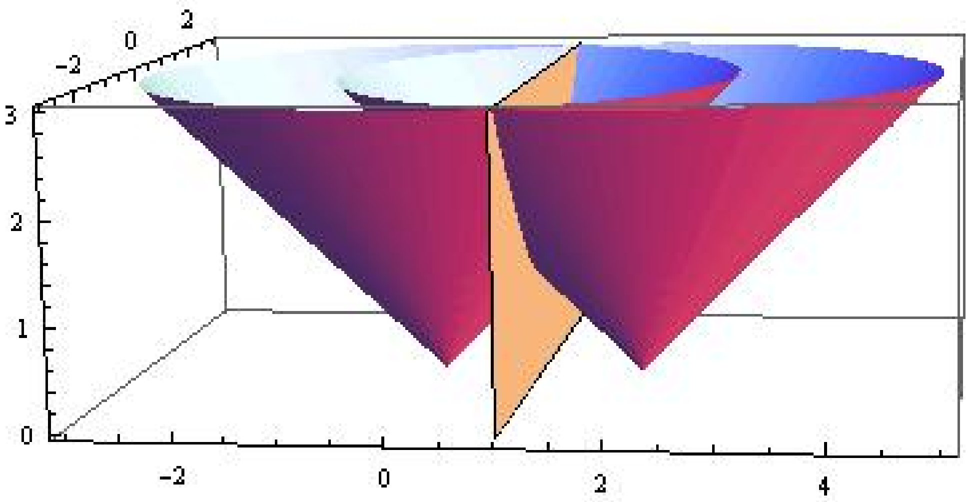
$$z^2 = (x - x_i)^2 + (y - y_i)^2.$$

Notice that, for every point q in the plane,

$$d(q, p_i) = \text{vertical\_distance}(q, C_i).$$

In other words: the cones are the distance functions to the sites.

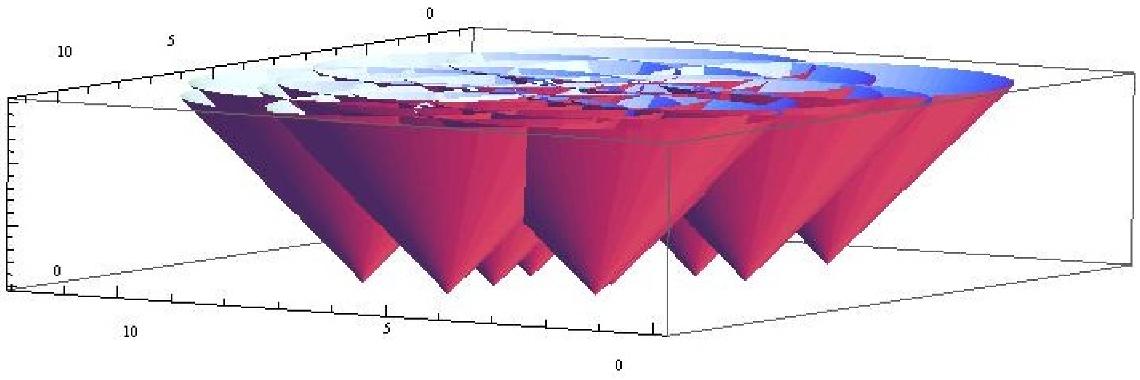
3-DIMENSIONAL INTERPRETATION



The intersection of two cones $C_i \cap C_j$ vertically projects onto the bisector b_{ij} of the sites p_i and p_j.

Fortune's algorithm

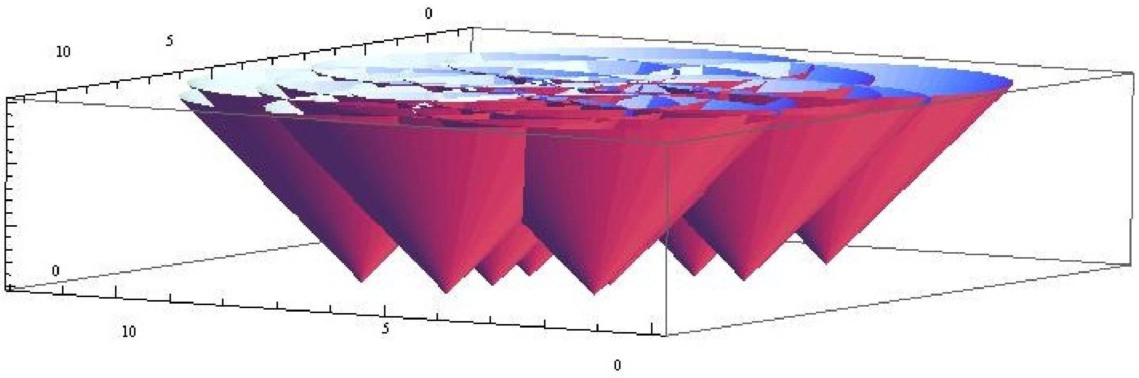
3-DIMENSIONAL INTERPRETATION



Consider the arrangement of all such cones.

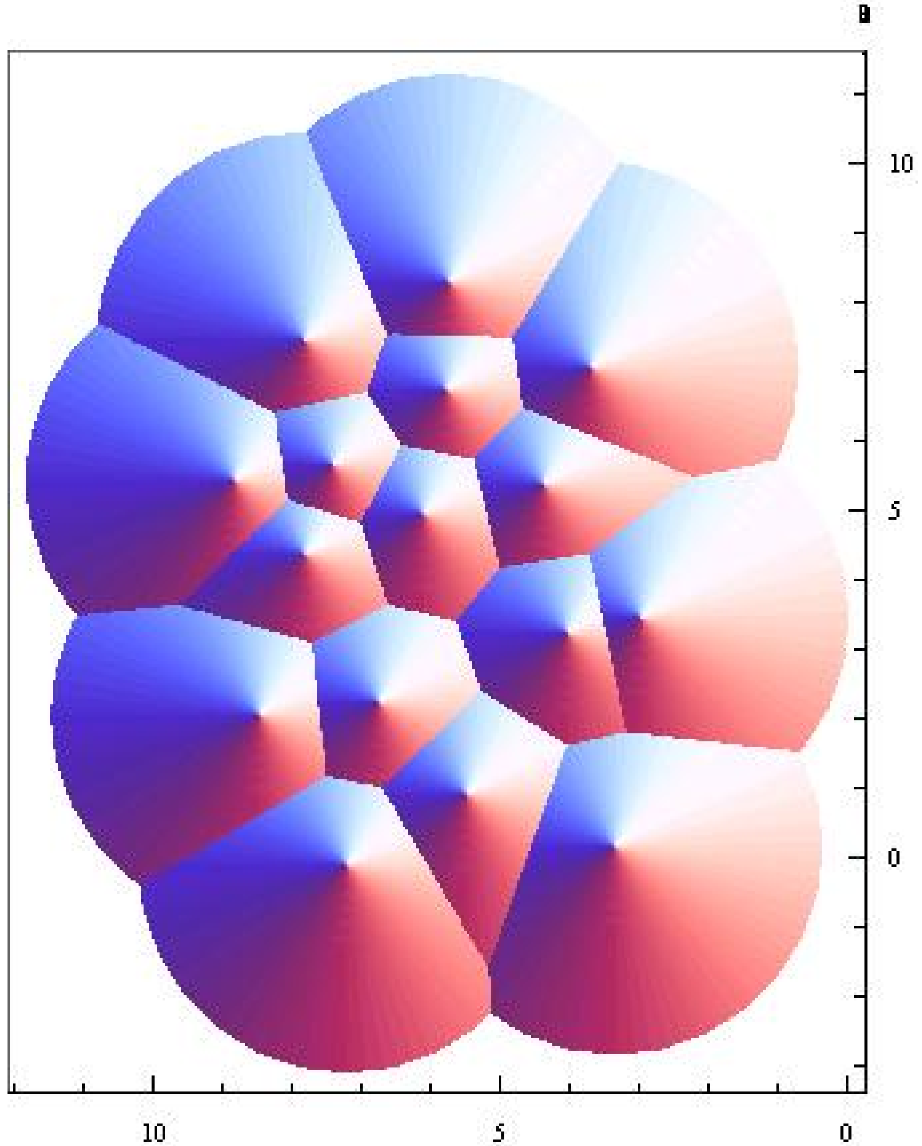
Fortune's algorithm

3-DIMENSIONAL INTERPRETATION



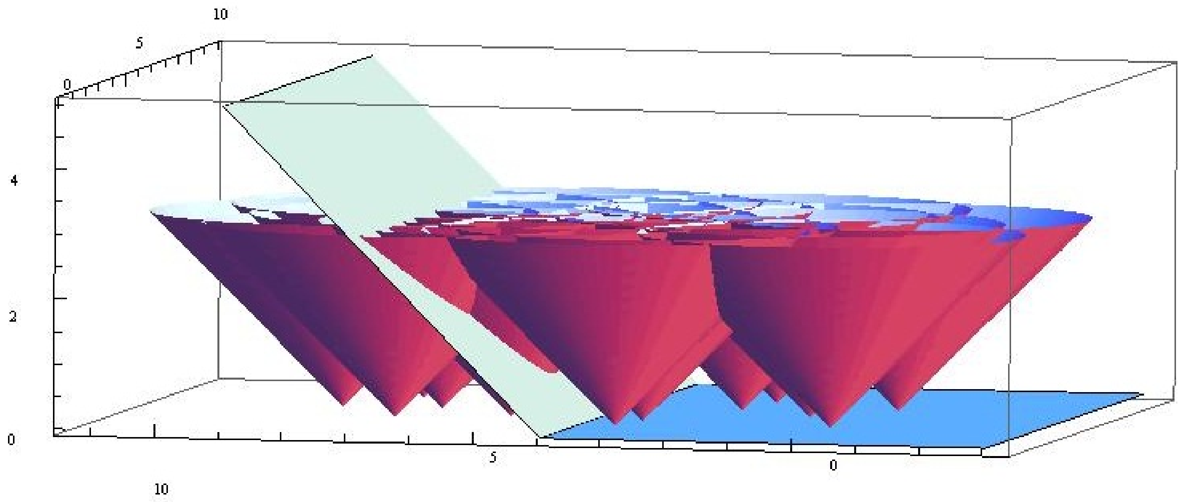
Consider the arrangement of all such cones.

Its lower envelope vertically projects onto $Vor(P)$.



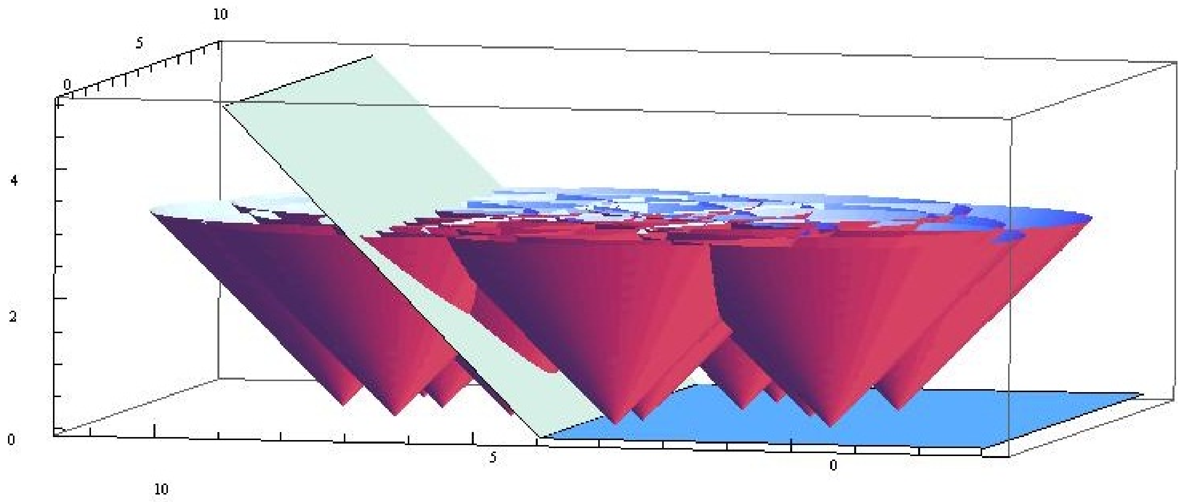
Fortune's algorithm

3-DIMENSIONAL INTERPRETATION



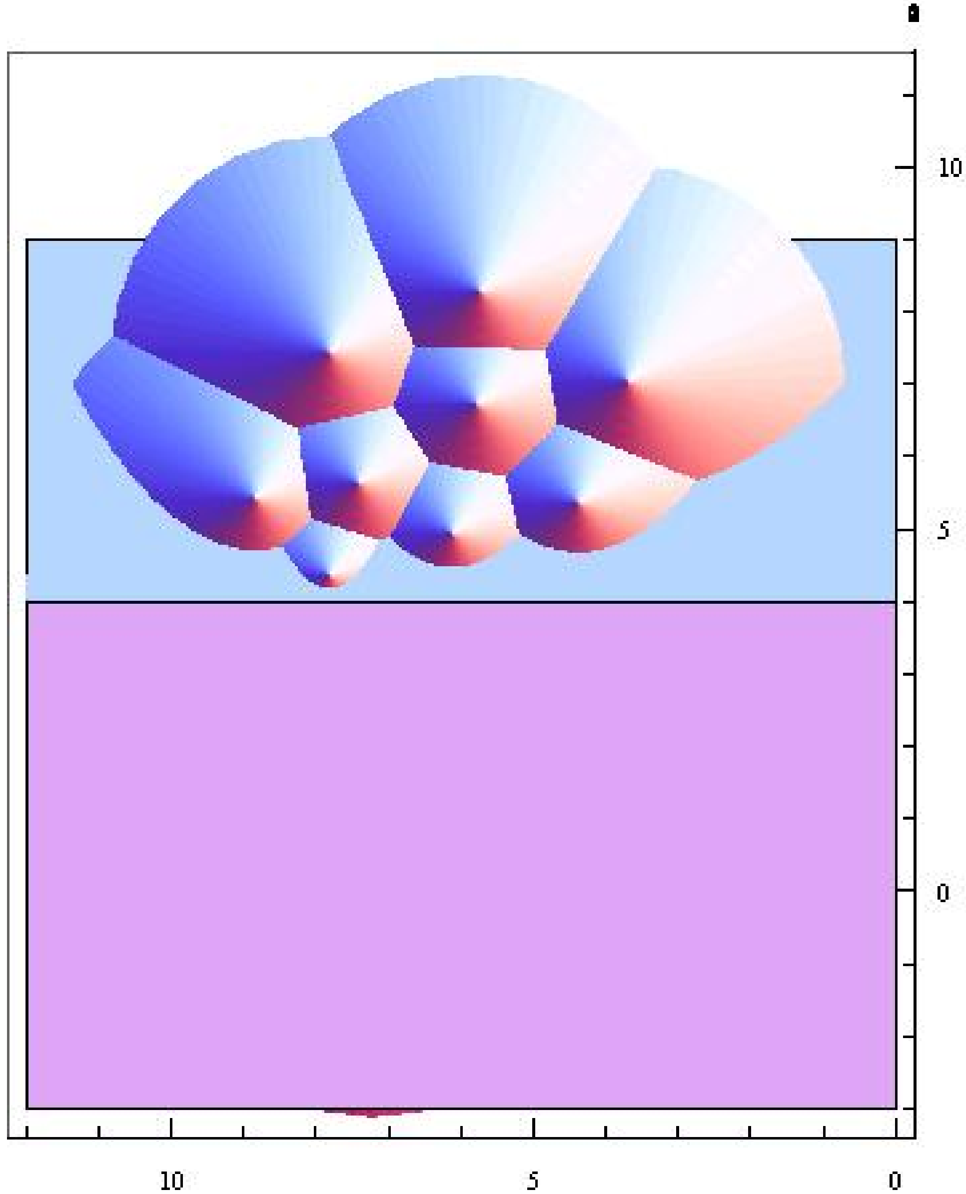
Sweep the arrangement of cones with a plane forming angle $\pi/2$ with the horizontal (which is the distance function to the line).

3-DIMENSIONAL INTERPRETATION



Sweep the arrangement of cones with a plane forming angle $\pi/2$ with the horizontal (which is the distance function to the line).

This is the same as Fortune's algorithm!



3D projection algorithm

3D projection algorithm

Consider the point set P as being embedded in the plane $z = 0$.

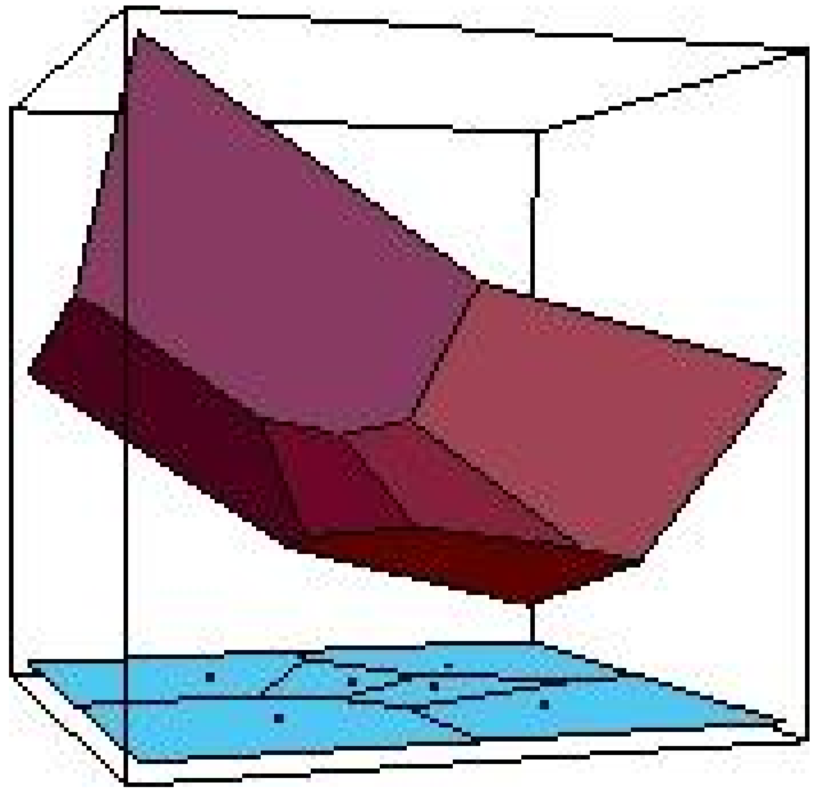
Consider the paraboloid $z = x^2 + y^2$.

For each point $p_i \in P$ let p_i^* be its vertical projection of p_i onto the paraboloid, i.e.:

$$\text{if } p_i = (a_i, b_i, 0), \text{ then } p_i^* = (a_i, b_i, a_i^2 + b_i^2).$$

For each point p_i^* consider the plane which is tangent to the paraboloid at p_i^*.

The Voronoi diagram of P is the orthogonal projection onto the plane $z = 0$ of the polyhedral convex region obtained when intersecting the upper halfspaces defined by these planes.



3D projection algorithm

Consider the point set P as being embedded in the plane $z = 0$.

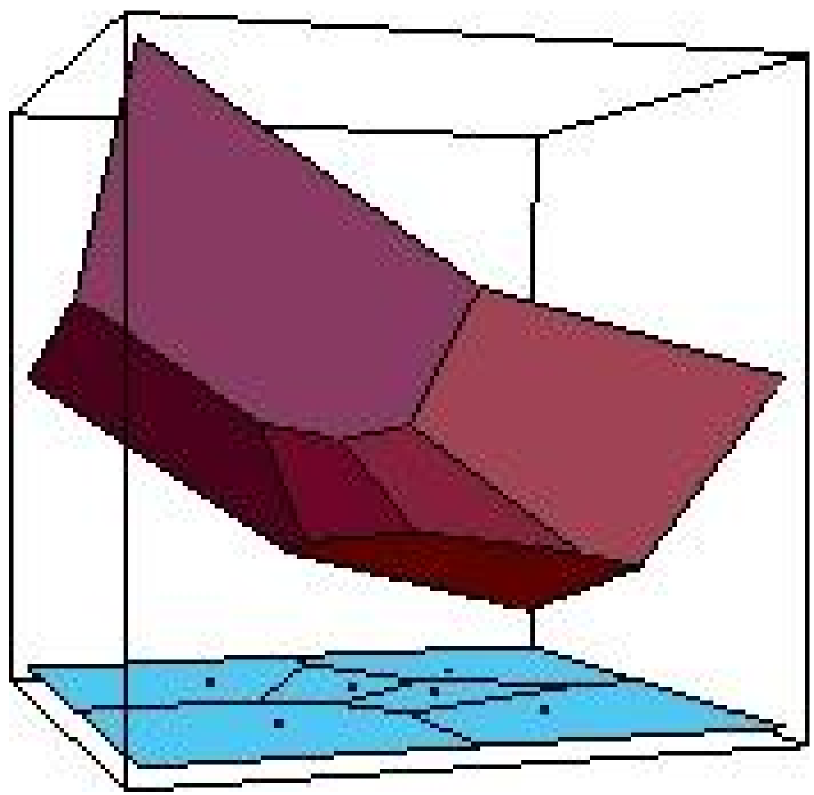
Consider the paraboloid $z = x^2 + y^2$.

For each point $p_i \in P$ let p_i^* be its vertical projection of p_i onto the paraboloid, i.e.:

$$\text{if } p_i = (a_i, b_i, 0), \text{ then } p_i^* = (a_i, b_i, a_i^2 + b_i^2).$$

For each point p_i^* consider the plane which is tangent to the paraboloid at p_i^*.

The Voronoi diagram of P is the orthogonal projection onto the plane $z = 0$ of the polyhedral convex region obtained when intersecting the upper halfspaces defined by these planes.



This gives rise to an algorithm that runs in $O(n \log n)$ time and uses $O(n)$ space.

3D projection algorithm

Consider the point set P as being embedded in the plane $z = 0$.

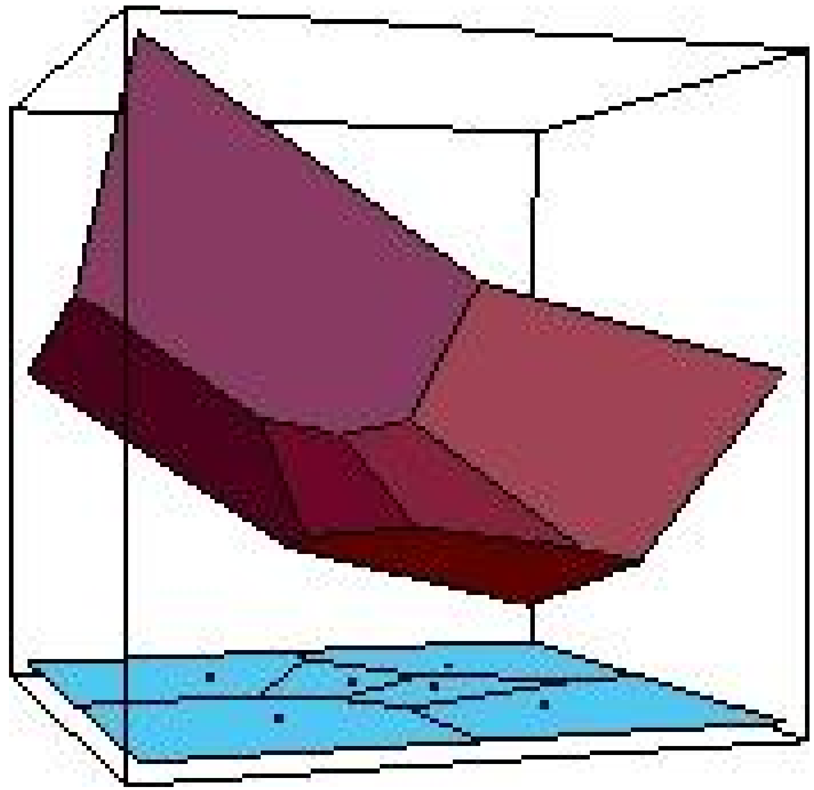
Consider the paraboloid $z = x^2 + y^2$.

For each point $p_i \in P$ let p_i^* be its vertical projection of p_i onto the paraboloid, i.e.:

$$\text{if } p_i = (a_i, b_i, 0), \text{ then } p_i^* = (a_i, b_i, a_i^2 + b_i^2).$$

For each point p_i^* consider the plane which is tangent to the paraboloid at p_i^*.

The Voronoi diagram of P is the orthogonal projection onto the plane $z = 0$ of the polyhedral convex region obtained when intersecting the upper halfspaces defined by these planes.



This gives rise to an algorithm that runs in $O(n \log n)$ time and uses $O(n)$ space.

Notice that the DCEL is the same for the polyhedron and for the Voronoi diagram!

References

TWO ADDRESSES TO PLAY WITH VORONOI DIAGRAMS

http://www.pi6.fernuni-hagen.de/GeomLab/VoroGlide/

http://www.dma.fi.upm.es/docencia/segundociclo/geomcomp/voronoi.html

AND TWO BOOKS WITH MUCH MORE INFORMATION

A. Okabe, B. Boots, K. Sugihara, S. N. Chiu
Spatial Tessellations
2nd ed., J. Wiley & Sons, 2000.

F. Aurenhammer, R. Klein, D.-T. Lee
Voronoi Diagrams and Delaunay Triangulations
World Scientific, 2013.