Computational Geometry
[csci 3250]

Laura Toma

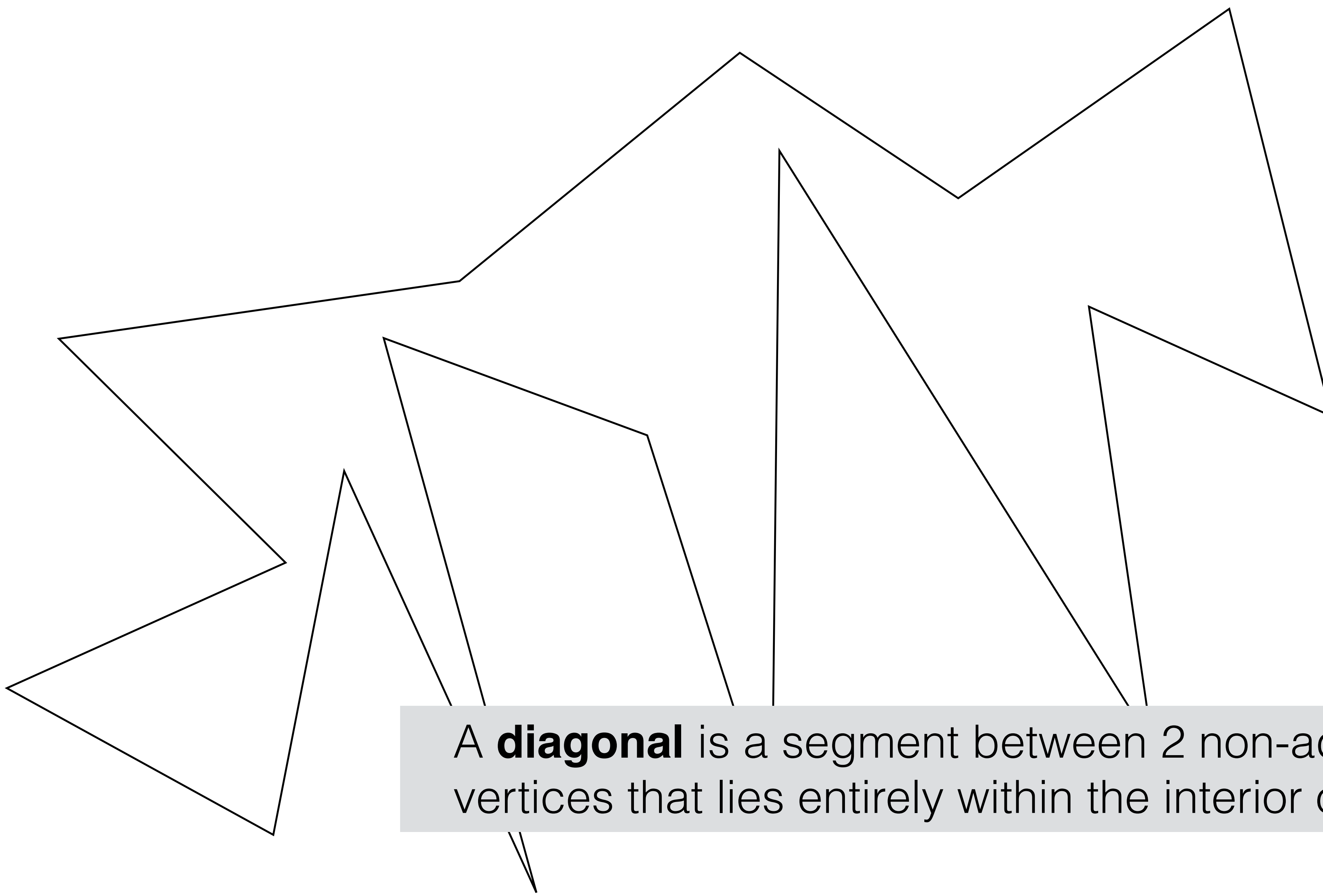
Bowdoin College

Polygon Triangulation

Polygon Triangulation: Definition

Polygon P

Triangulation of P: a partition of P into triangles using a set of diagonals.

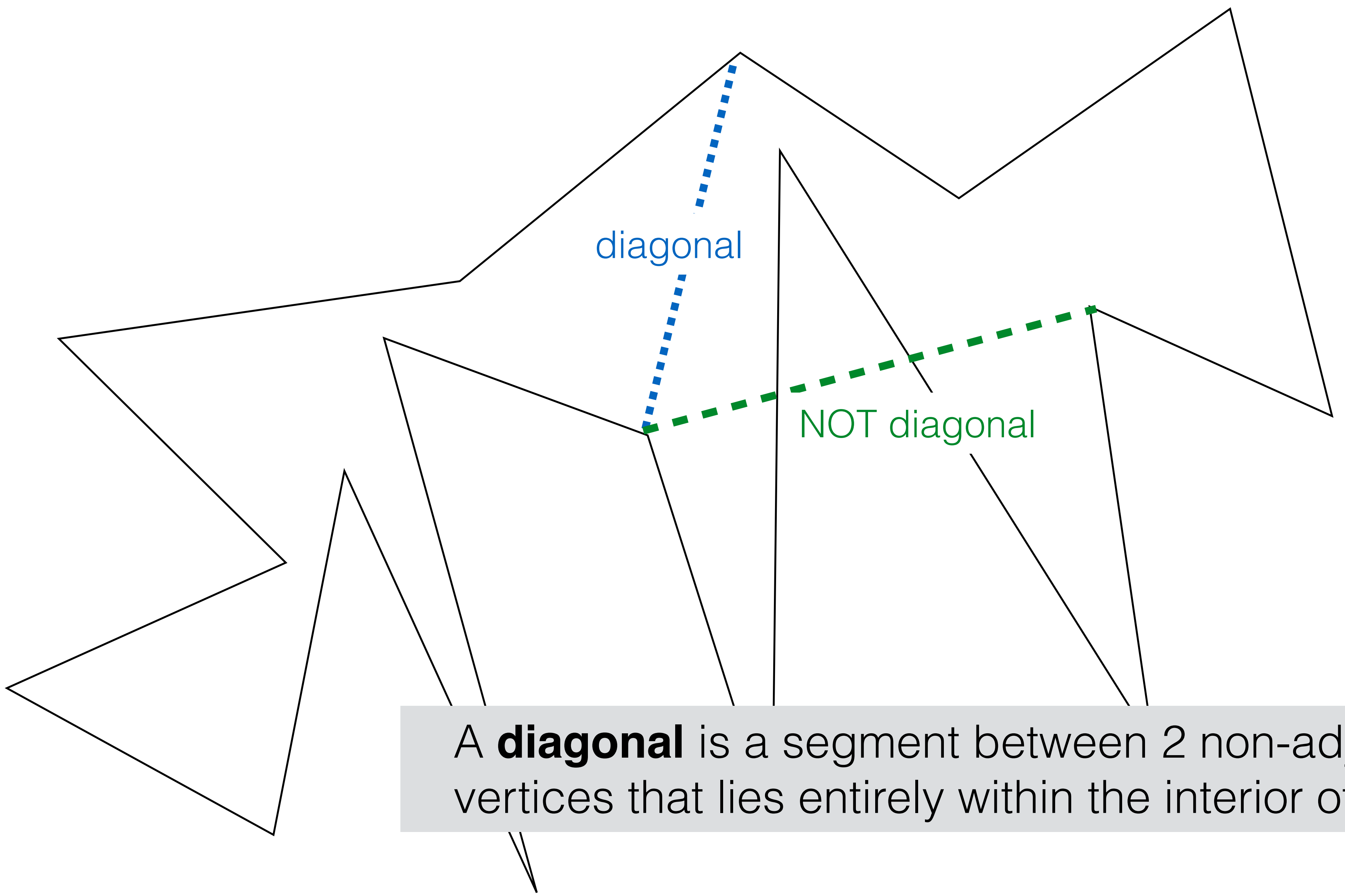


A **diagonal** is a segment between 2 non-adjacent vertices that lies entirely within the interior of the polygon.

Polygon Triangulation: Definition

Polygon P

Triangulation of P: a partition of P into triangles using a set of diagonals.

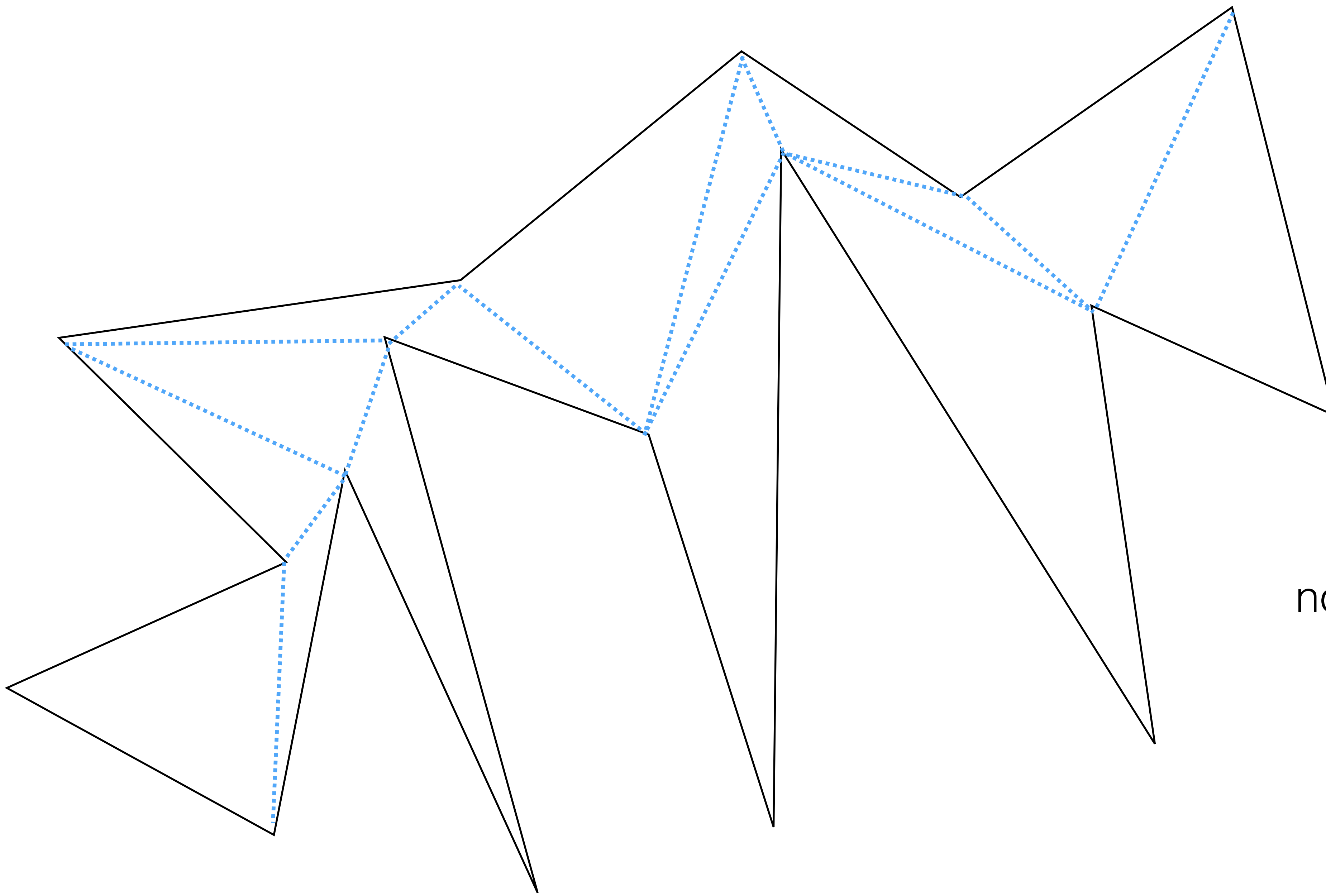


A **diagonal** is a segment between 2 non-adjacent vertices that lies entirely within the interior of the polygon.

Polygon Triangulation: Definition

Polygon P

Triangulation of P: a partition of P into triangles using a set of diagonals.

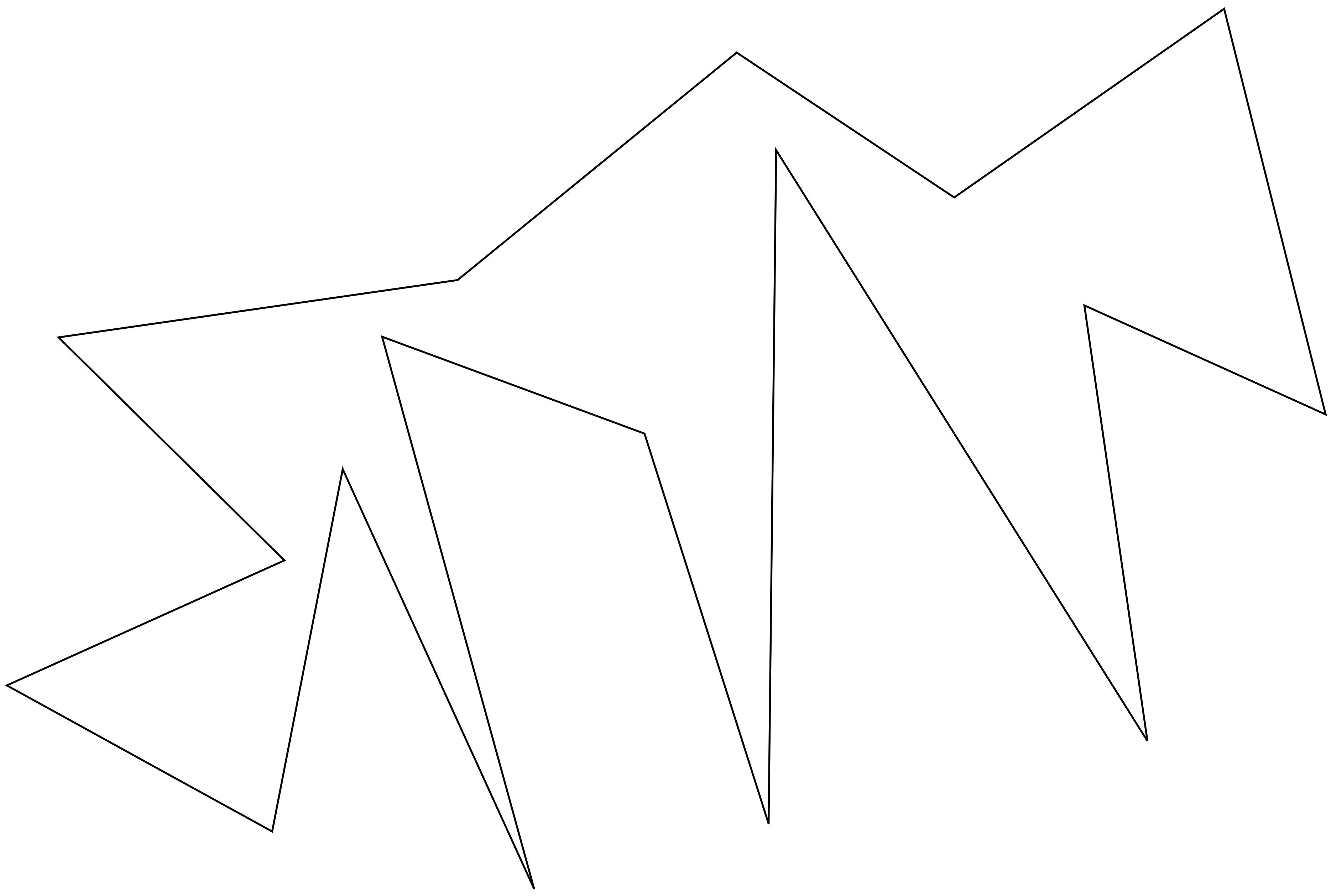


not unique

Polygon Triangulation: The problem

Given a polygon P, triangulate it.

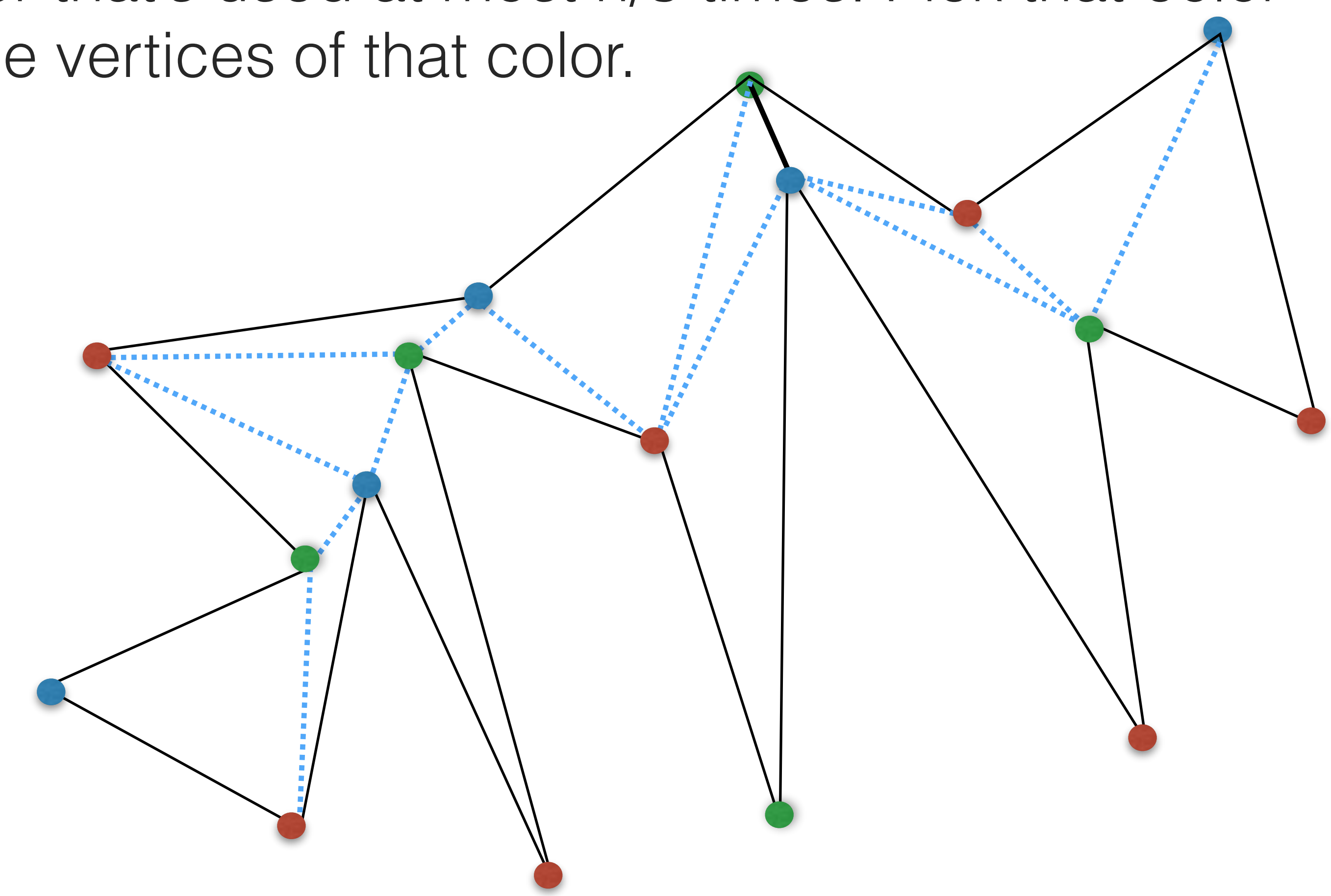
(output a set of diagonals that partition the polygon into triangles).



Motivation: Art gallery

Fisk's proof

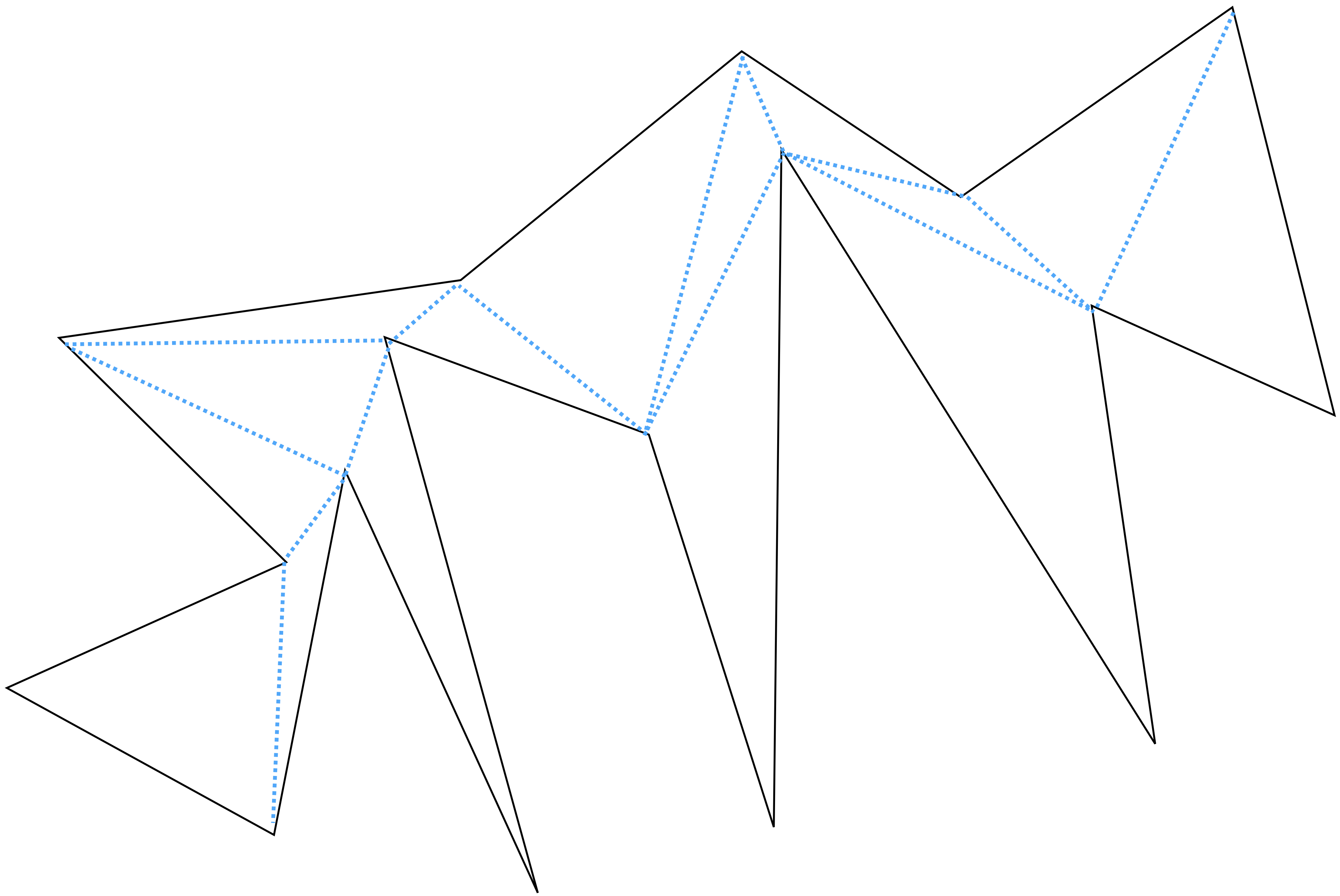
1. Any simple polygon can be triangulated.
2. Any triangulated simple polygon can be 3-colored.
3. Placing the guards at all the vertices assigned to one color guarantees the polygon is covered.
4. There must exist a color that's used at most n/3 times. Pick that color and place guards at the vertices of that color.



Polygon Triangulation

Does a triangulation always exist?

YES. The key to proving this is that any polygon n>3 has a diagonal.



Known Results

- Theorem: Any simple polygon must have a convex vertex (angle <180).
- Theorem: Any simple polygon with n>3 vertices contains (at least) a diagonal.
- Theorem: Any polygon can be triangulated by adding diagonals.
- Theorem: Any triangulation of a polygon of n vertices has n-2 triangles and n-3 diagonals.
- Theorem: Any simple polygon has at least two ears.

Naive triangulation by recursively finding diagonals

- A **diagonal** is a segment between 2 non-adjacent vertices that lies entirely within the interior of the polygon.

- Come up with algorithms to:
  - determine if two vertices of P form a diagonal
  - find a diagonal of P
  - triangulating a polygon by recursively finding diagonals.

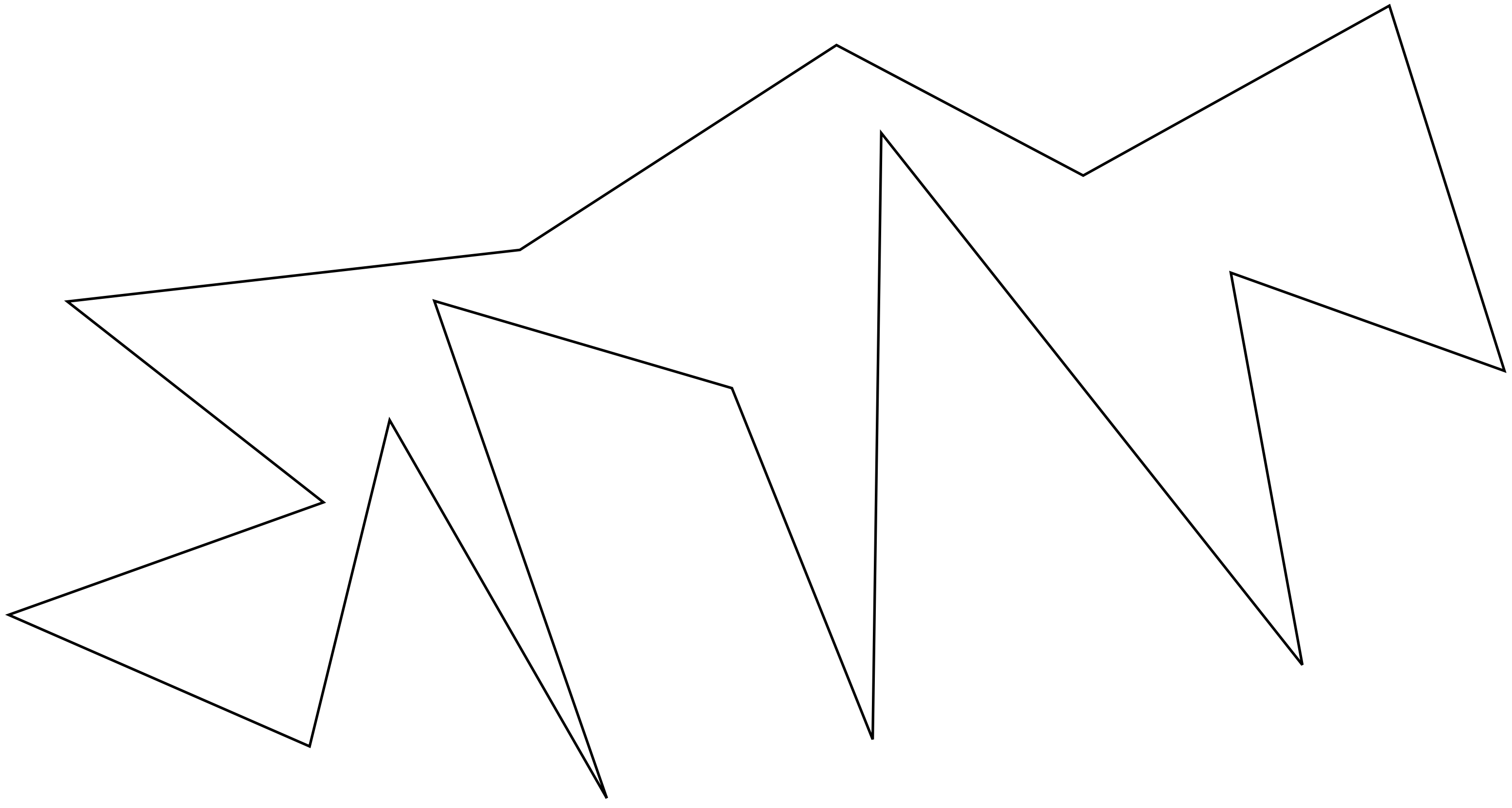
Naive triangulation by recursively finding diagonals

- **Algorithm 1:** Triangulation by finding diagonals
  - Idea: Check all pairs of vertices to find one which is a diagonal; repeat.
  - Analysis:
    - checking all vertices: $O(n^2)$ candidates for diagonals, checking each takes $O(n)$, overall $O(n^3)$
    - recurse, worst case on a problem of size n-1
    - overall $O(n^4)$

- **Algorithm 2:** Triangulation by *smartly* finding diagonals
  - Idea: Find a diagonal, output it, recurse.
  - A diagonal can be found in $O(n)$ time (using the proof that a diagonal exists)
  - $O(n^2)$

Algorithm 3: Triangulation by finding ears

- A **ear** with tip v_i is a set of 3 consecutive vertices v_{i-1}, v_i, v_{i+1} if $v_{i-1}v_{i+1}$ is a diagonal.
- Put differently, v_i is an ear tip if the vertex right before it and the vertex right after it are *visible* to each other
- Theorem: Any simple polygon has at least two ears.



Algorithm 3: Triangulation by finding ears

- Idea: Find an ear, output the diagonal, delete the ear tip, repeat.

- Analysis:
    - checking whether a vertex is ear tip or not: $O(n)$
    - checking all vertices: $O(n^2)$
    - overall $O(n^3)$

Algorithm 4: Triangulation by finding ears in $O(n^2)$

- **Idea:** Avoid recomputing ear status for all vertices every time
  - When you remove a ear tip from the polygon, which vertices might change their ear status?

History of Polygon Triangulation

- Early algorithms: $O(n^4)$, $O(n^3)$, $O(n^2)$
- Several $O(n \lg n)$ algorithms known
- …
- Many papers with improved bounds
- …
- 1991: Bernard Chazelle (Princeton) gave an $O(n)$ algorithm
    - https://www.cs.princeton.edu/~chazelle/pubs/polygon-triang.pdf
    - Ridiculously complicated, not practical
    - $O(1)$ people actually understand it (and I'm not one of them)

practical

not practical

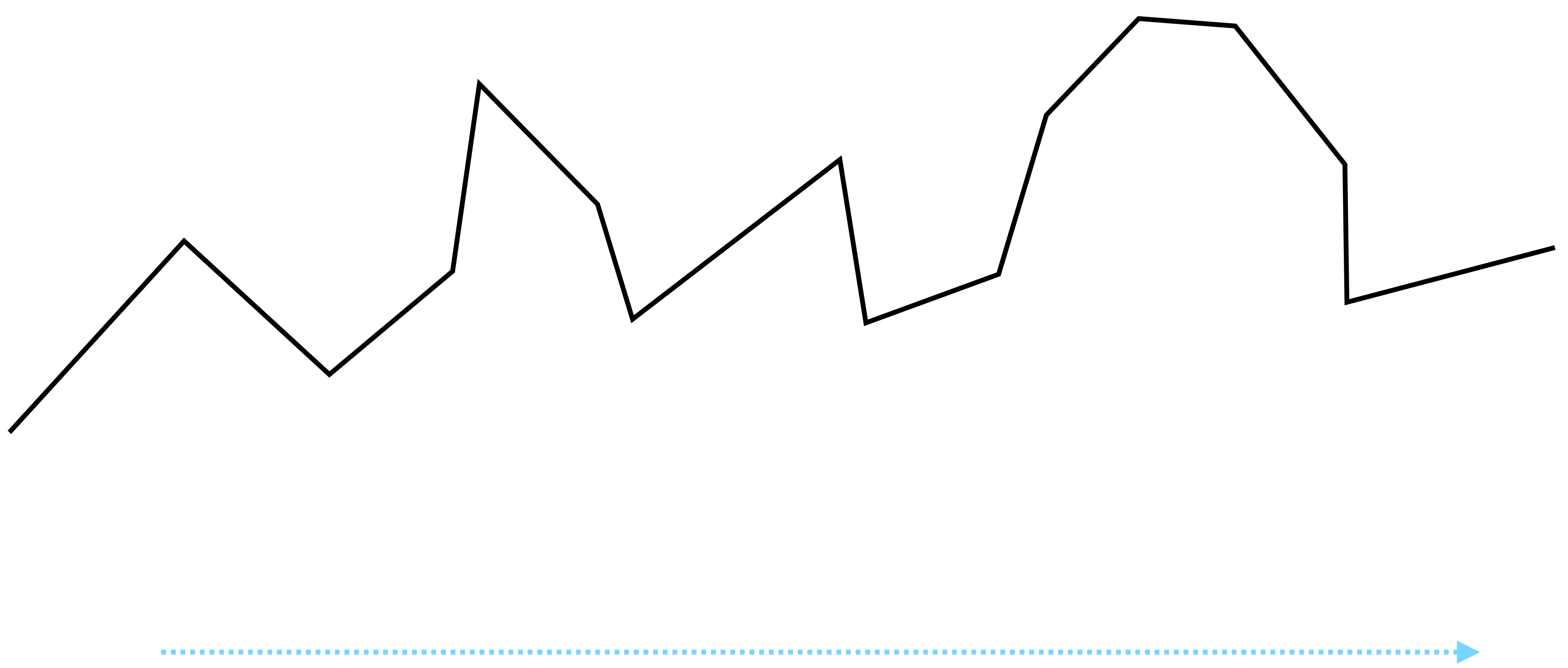
- No algorithm is known that is practical enough to run faster than the $O(n \lg n)$ algorithms
- OPEN problem
    - A practical algorithm that's theoretically better than $O(n \lg n)$.

An O(n lg n) Polygon Triangulation Algorithm

- Ingredients
  - Consider the special case of triangulating **monotone/unimonotone** polygons
  - Convert an arbitrary polygon into monotone/unimonotone polygons

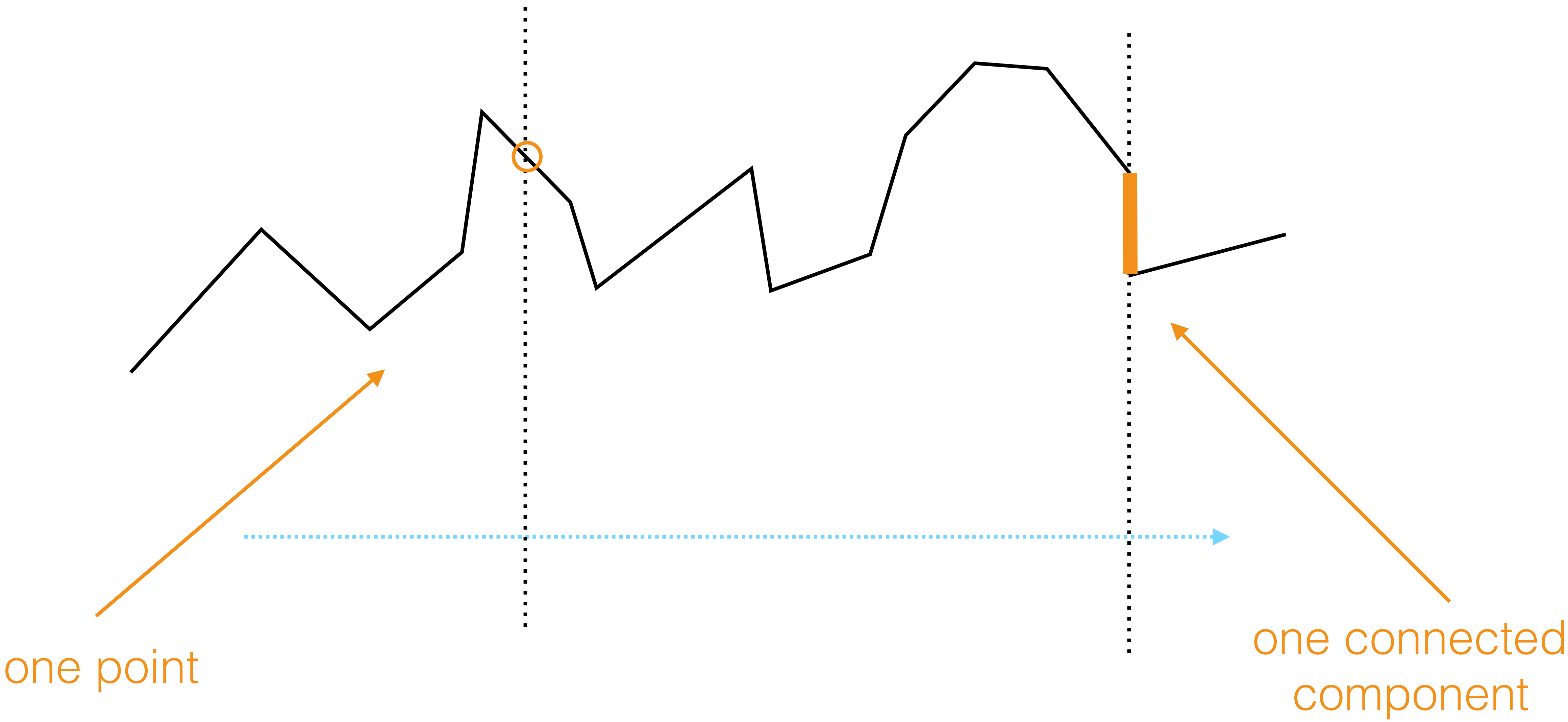
Monotone chains

A polygonal chain is **x-monotone** if any line perpendicular to x-axis intersects it in one point (one connected component).

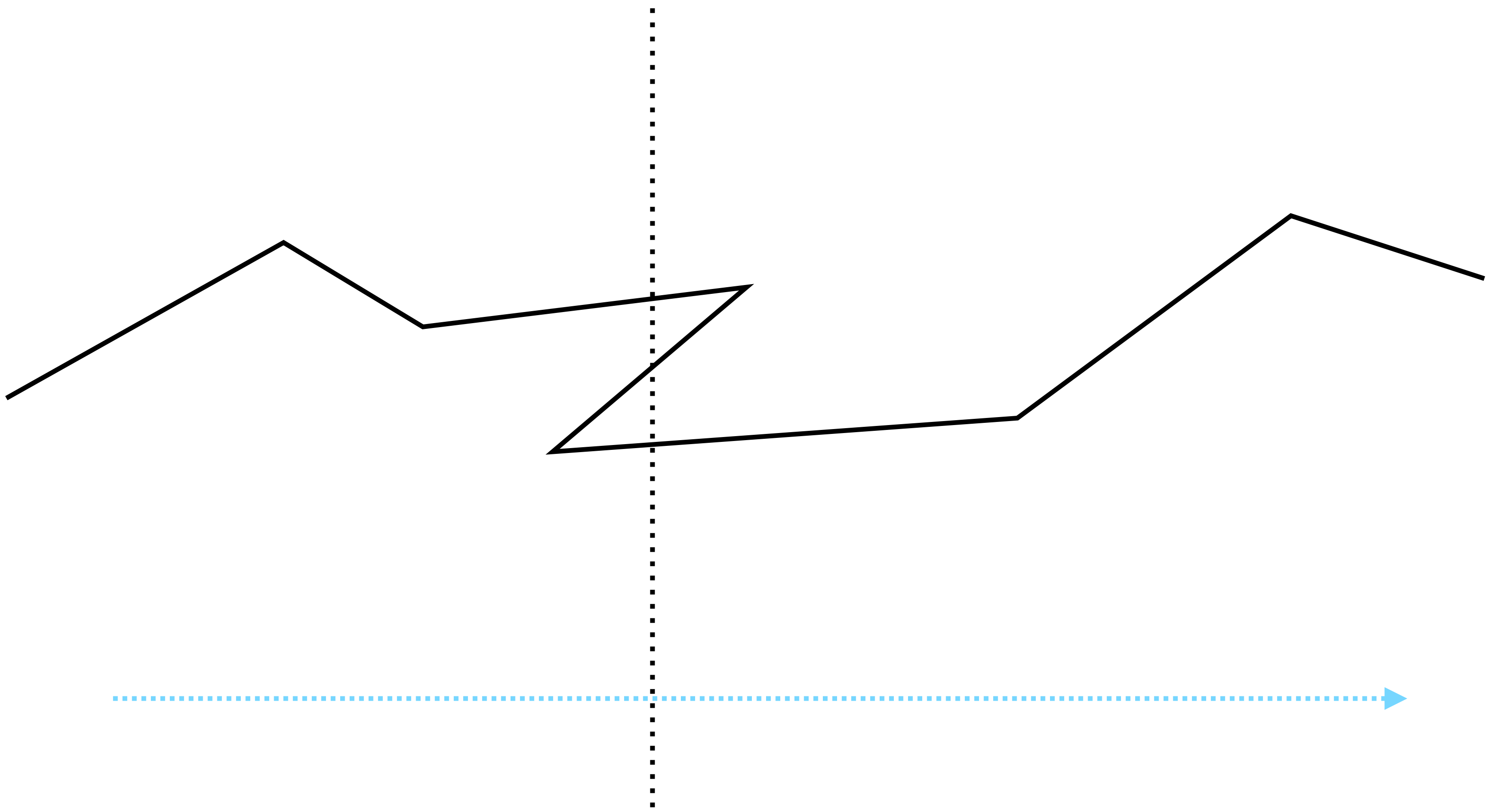


Monotone chains

A polygonal chain is **x-monotone** if any line perpendicular to **x**-**axis** intersects it in one point (one connected component).



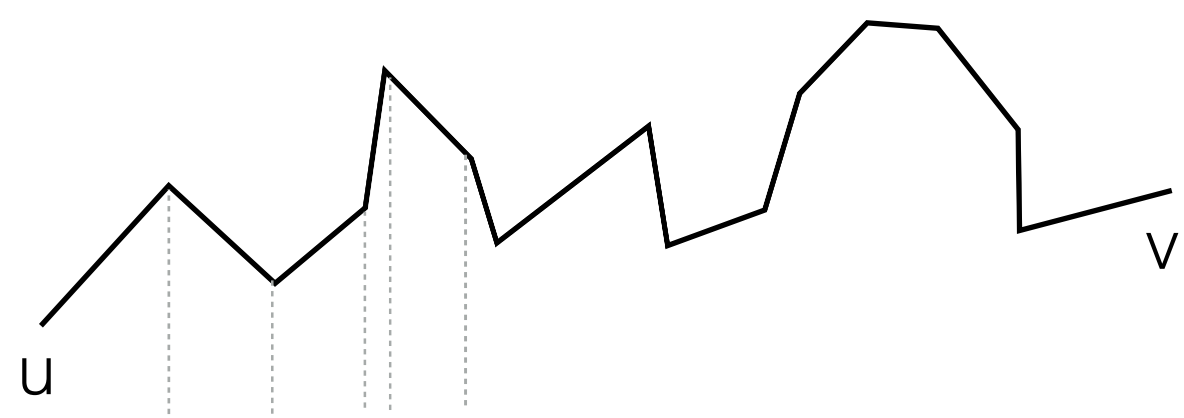
Monotone chains



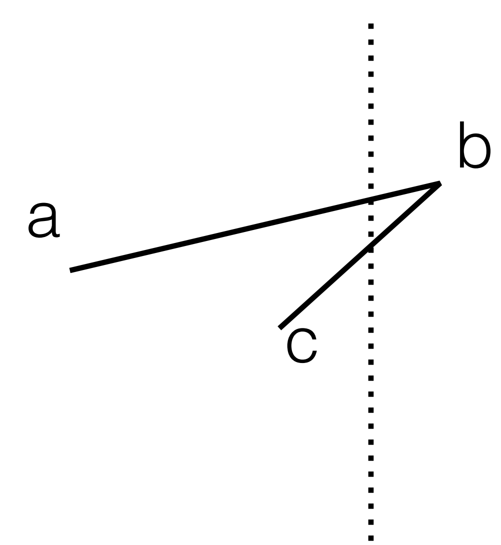
Not x-monotone

Monotone chains

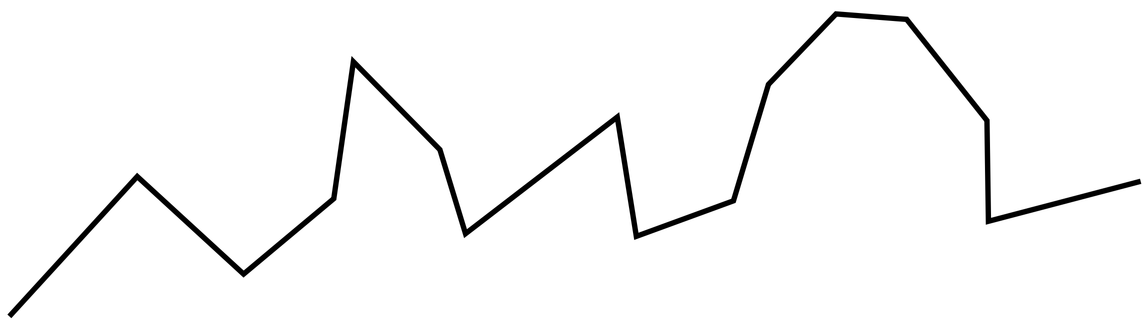
- Claim: Let u and v be the points on the chain with min/max x-coordinate. The vertices on the boundary of an x-monotone chain, going from u to v, are in x-order.



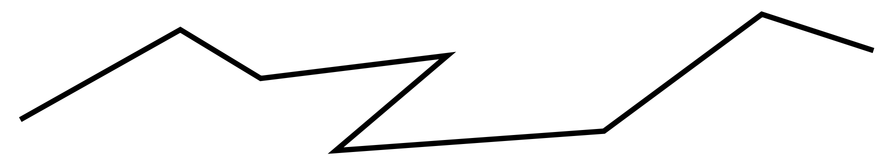
x-monotone



Monotone chains



x-monotone

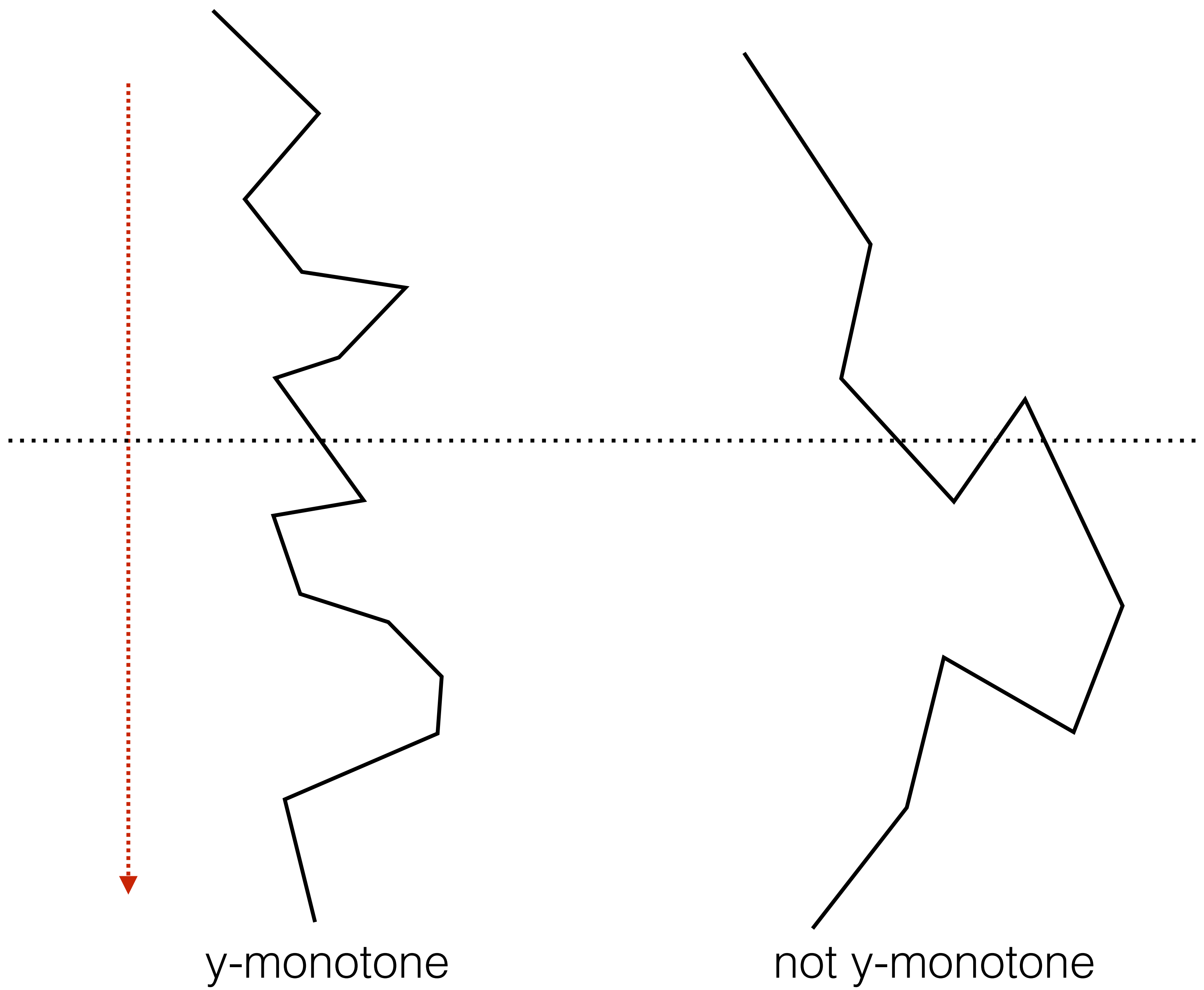


not x-monotone

As you travel along this chain, your x-coordinate is staying the same or increasing

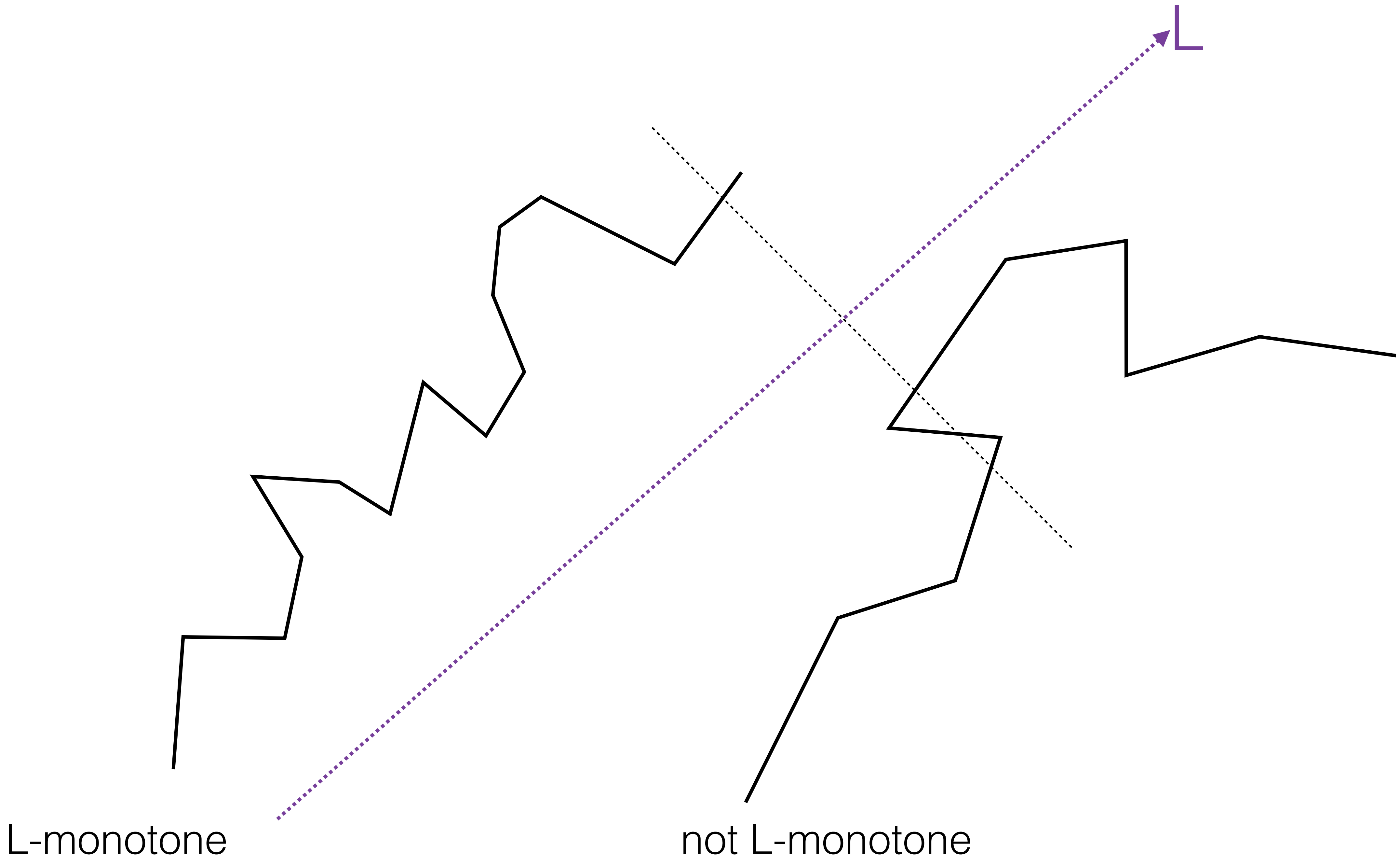
Monotone chains

A polygonal chain is **y-monotone** if any line perpendicular to **y-axis** intersects it in one point (one connected component).



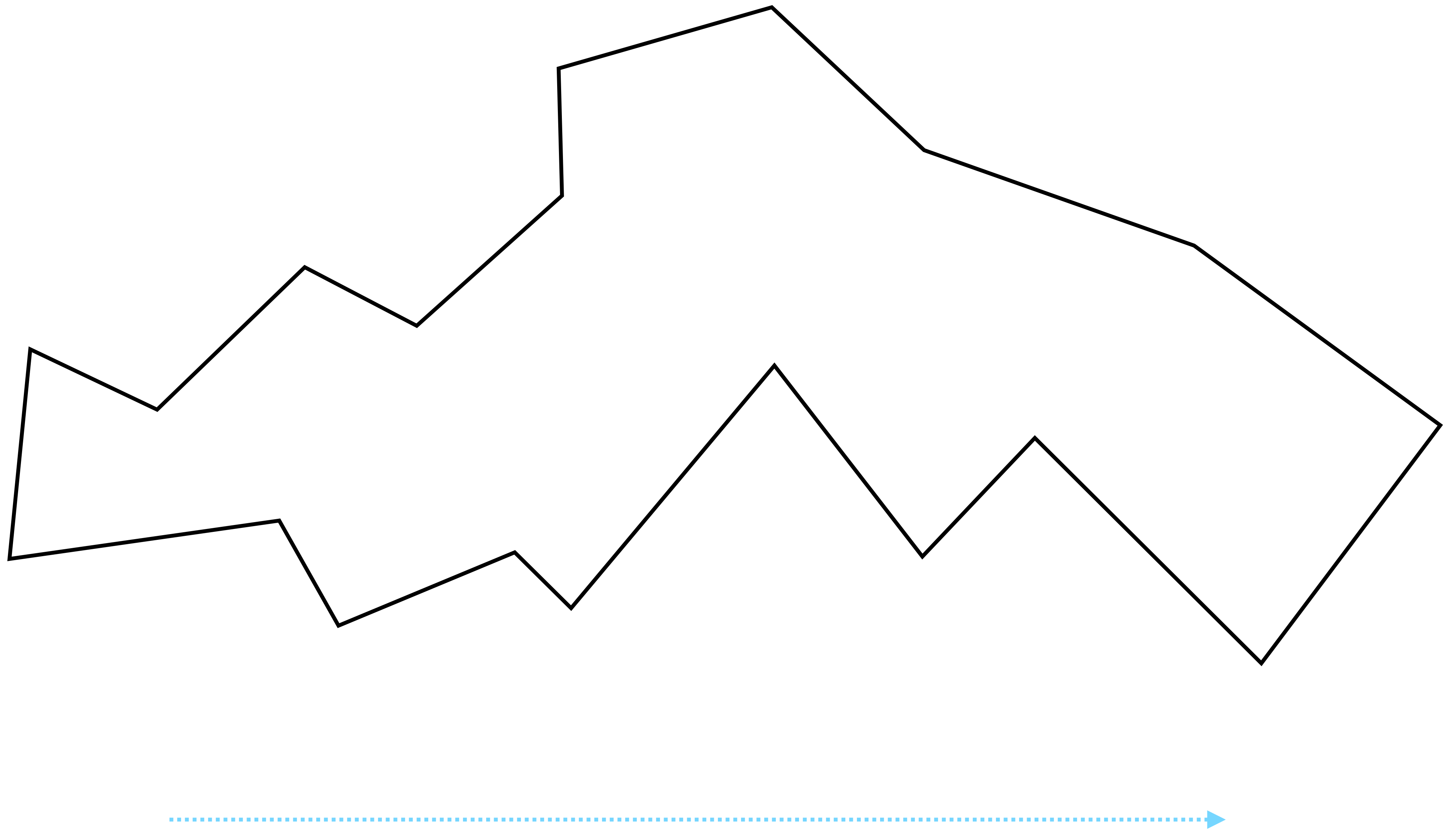
Monotone chains

A polygonal chain is **L-monotone** if any line perpendicular to **line L** intersects it in one point (one connected component).



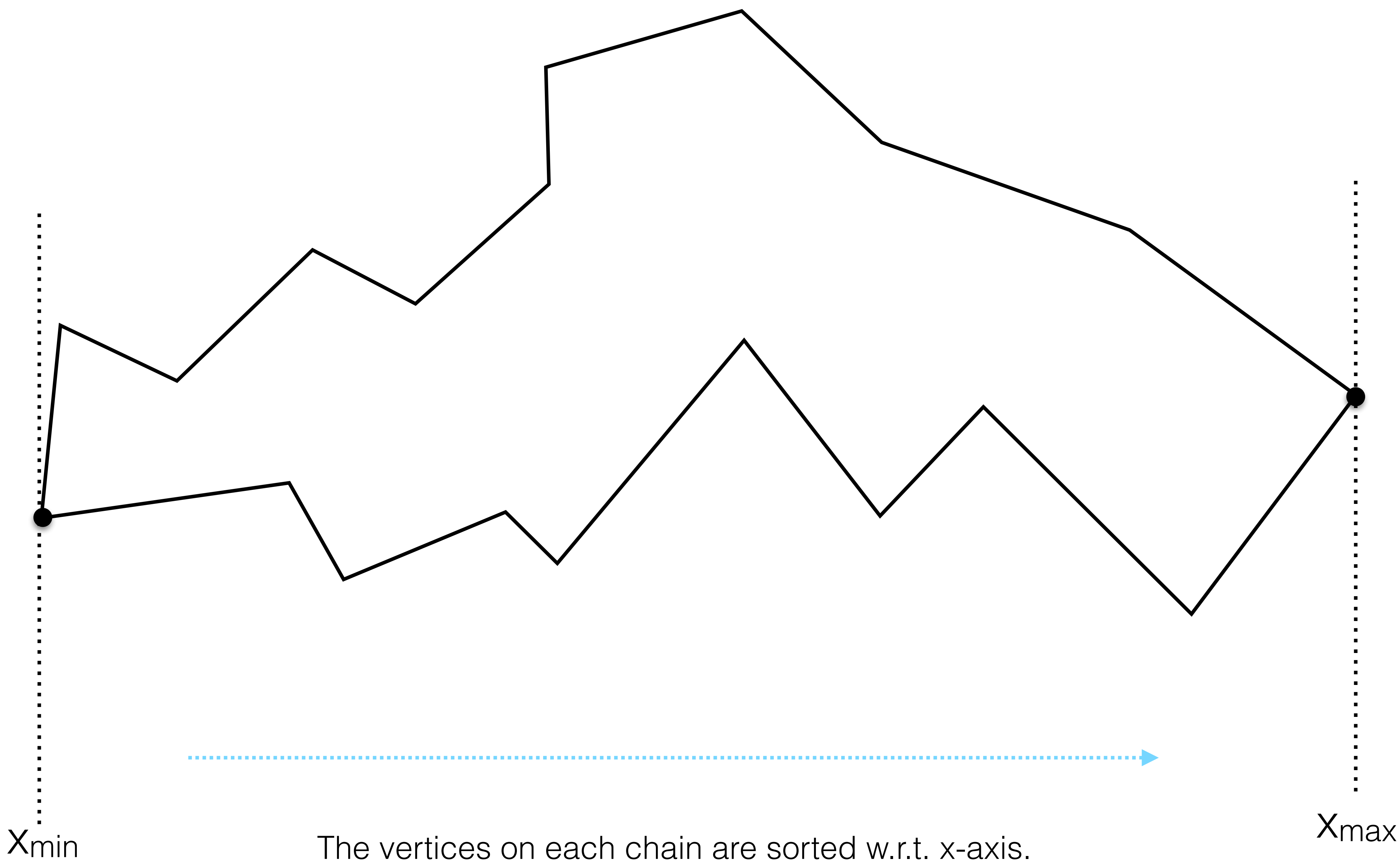
Monotone polygons

A polygon is x-monotone if its boundary can be split into two x-monotone chains.



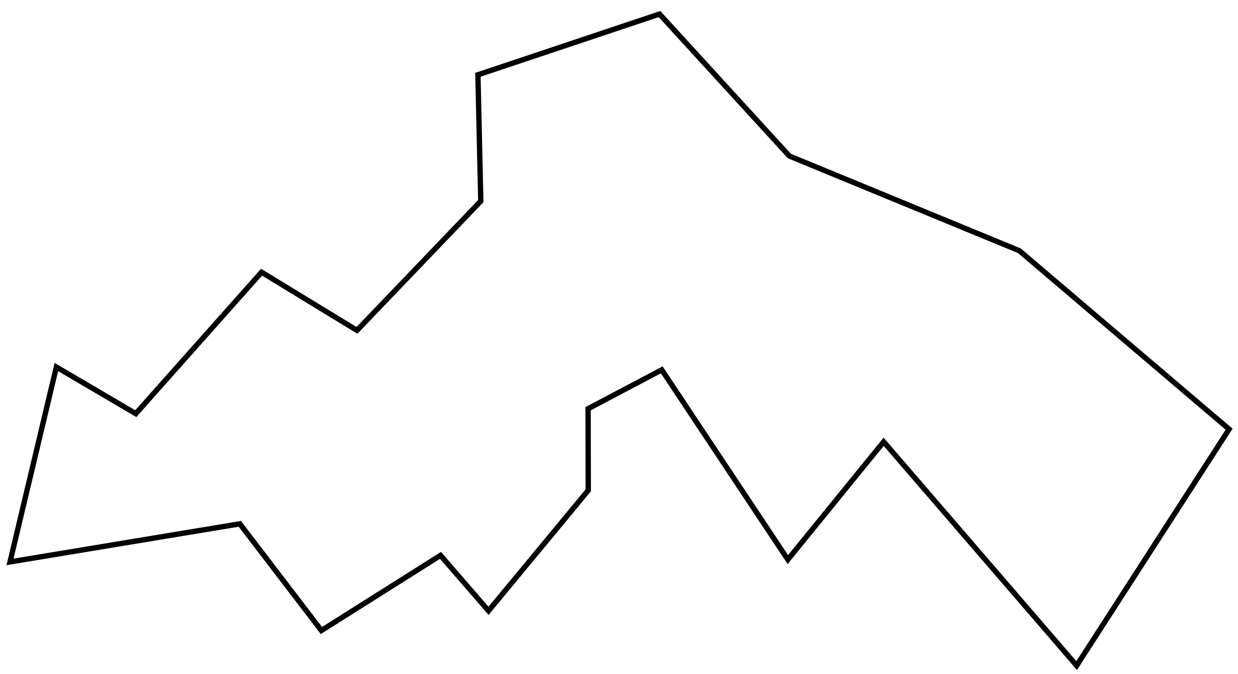
Monotone polygons

A polygon is x-monotone if its boundary can be split into two x-monotone chains.

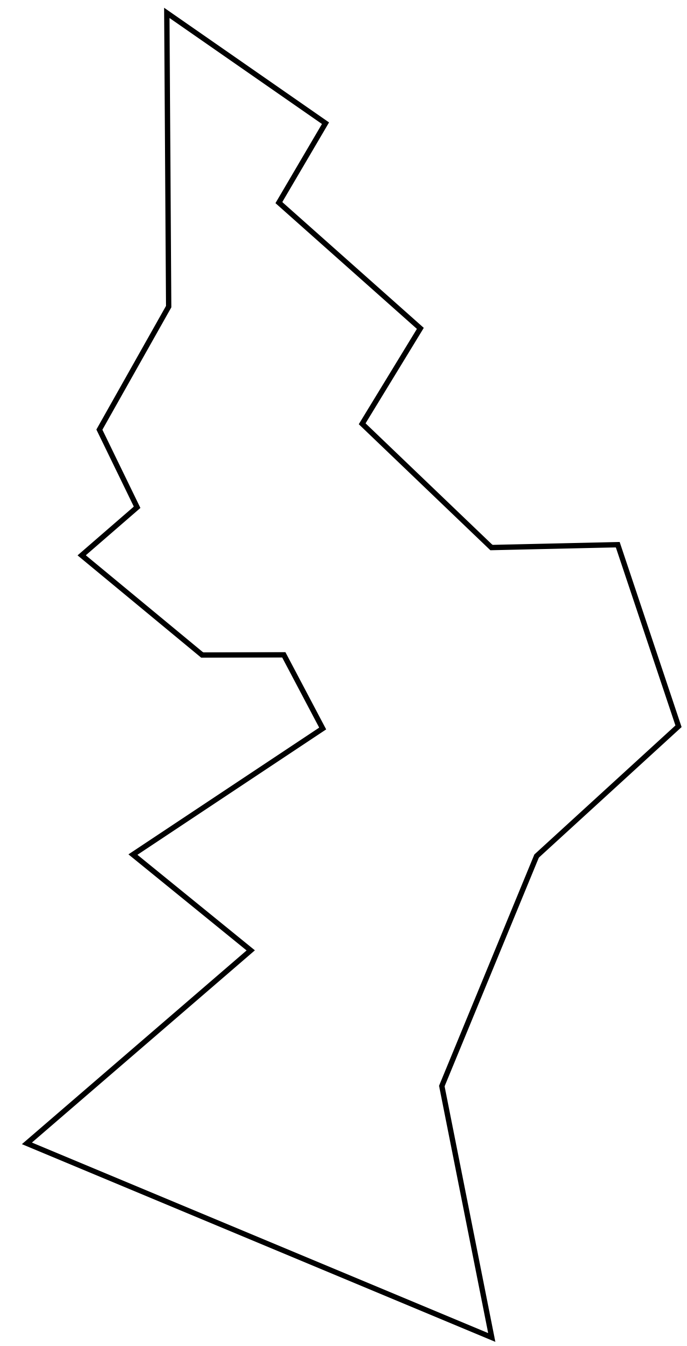


The vertices on each chain are sorted w.r.t. x-axis.

Monotone polygons



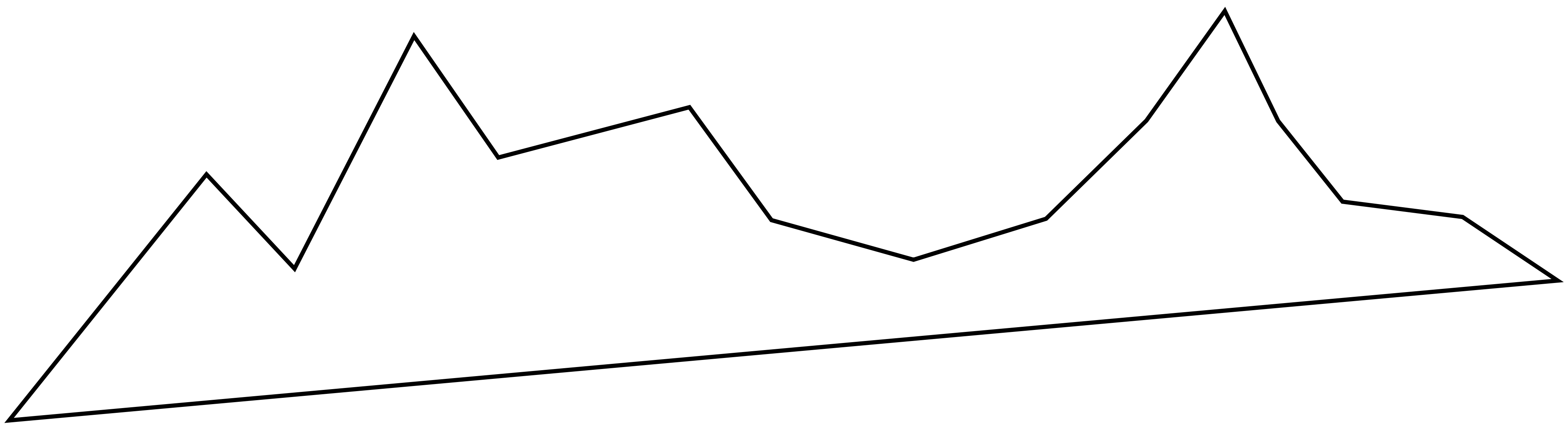
x-monotone



y-monotone

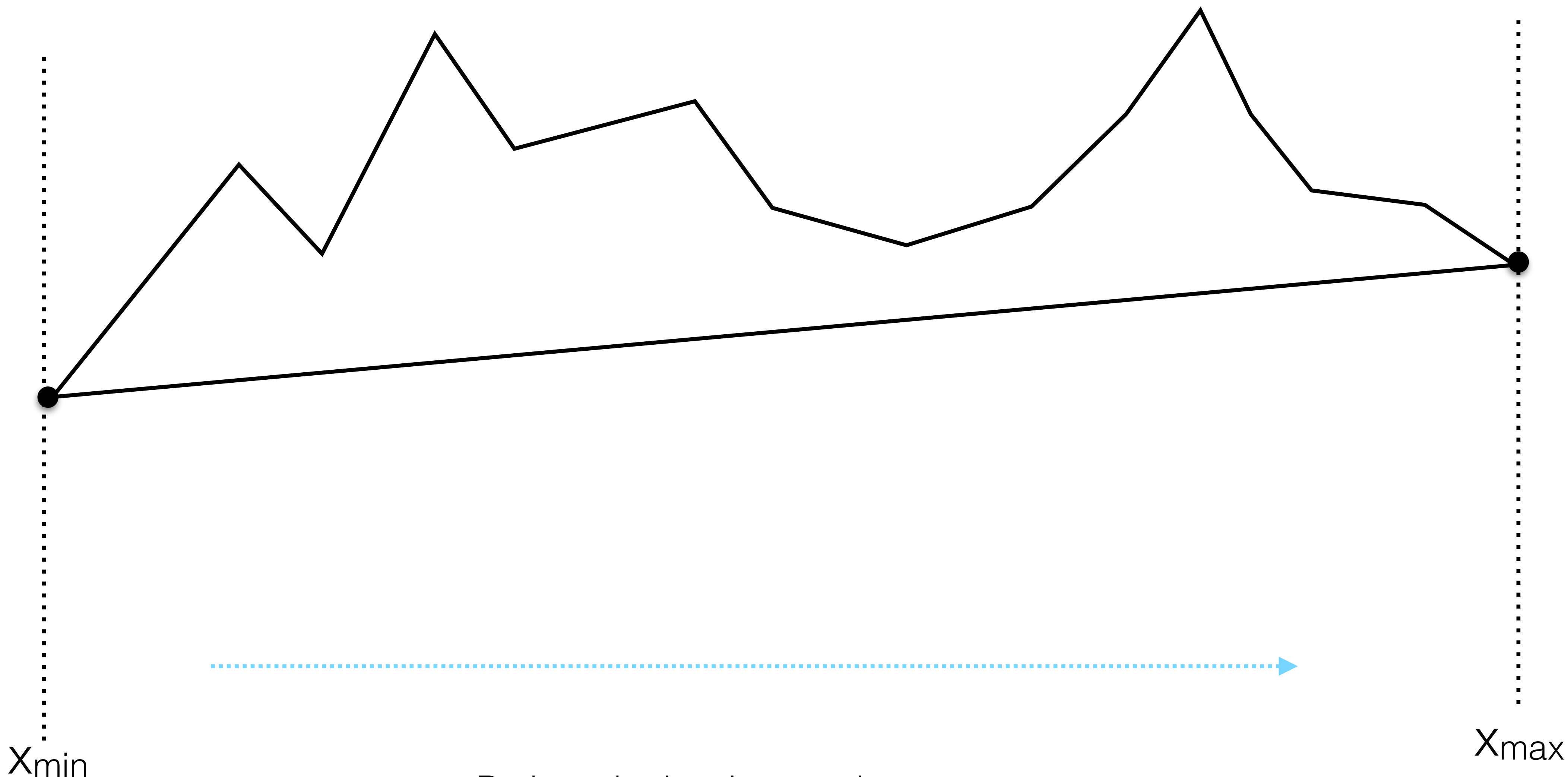
Monotone Mountains

A polygon is an **x-monotone mountain** if it is monotone and one of the two chains is a single segment.



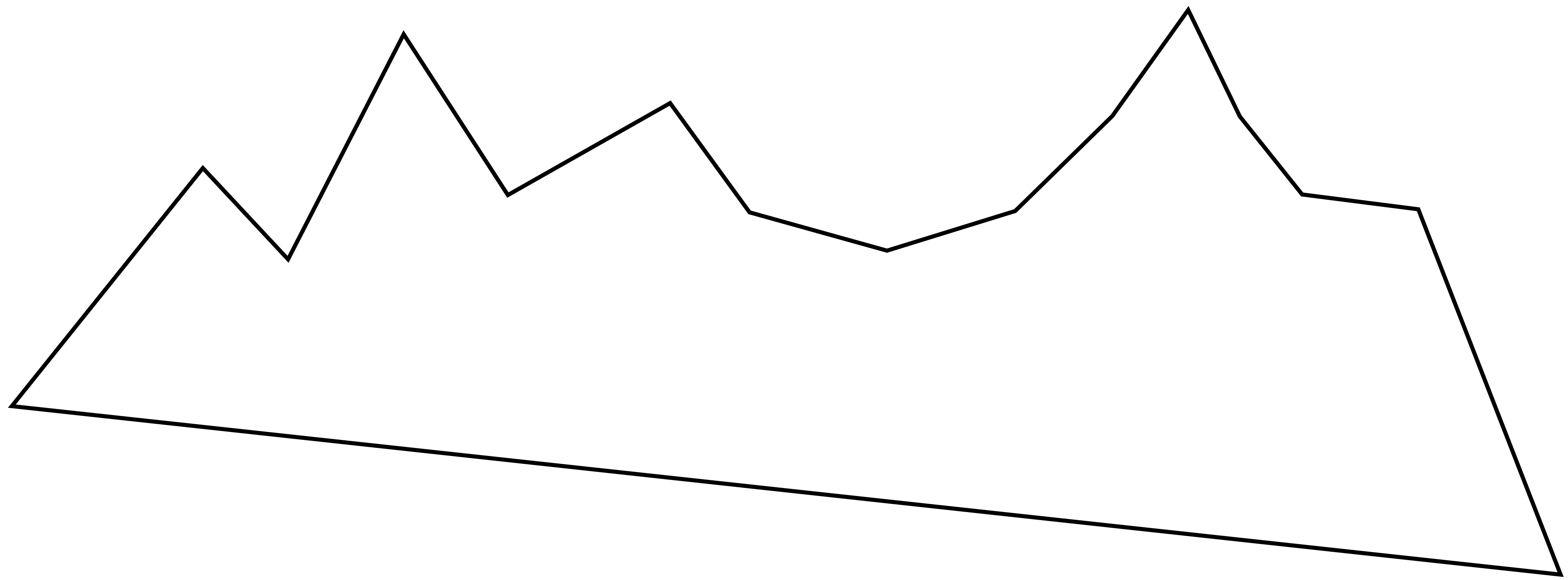
Monotone Mountains

A polygon is an **x-monotone mountain** if it is monotone and one of the two chains is a single segment.



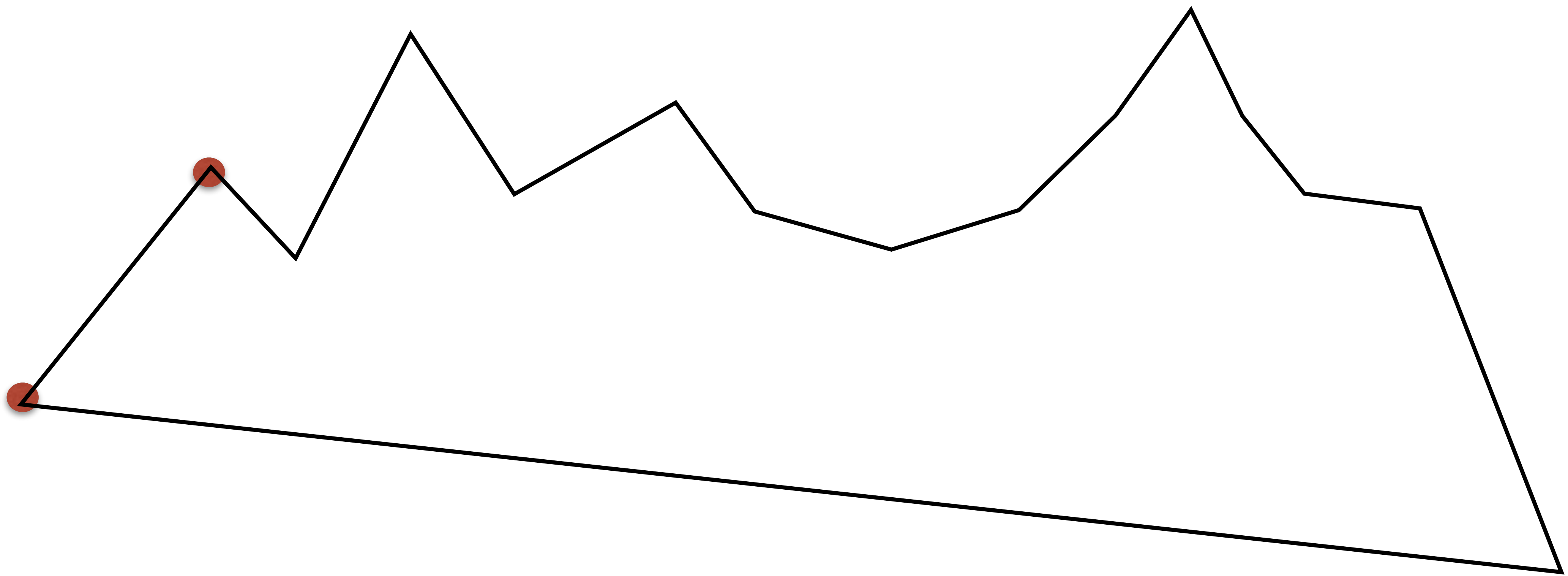
Both endpoints have to be convex.

Monotone mountains are easy to triangulate!

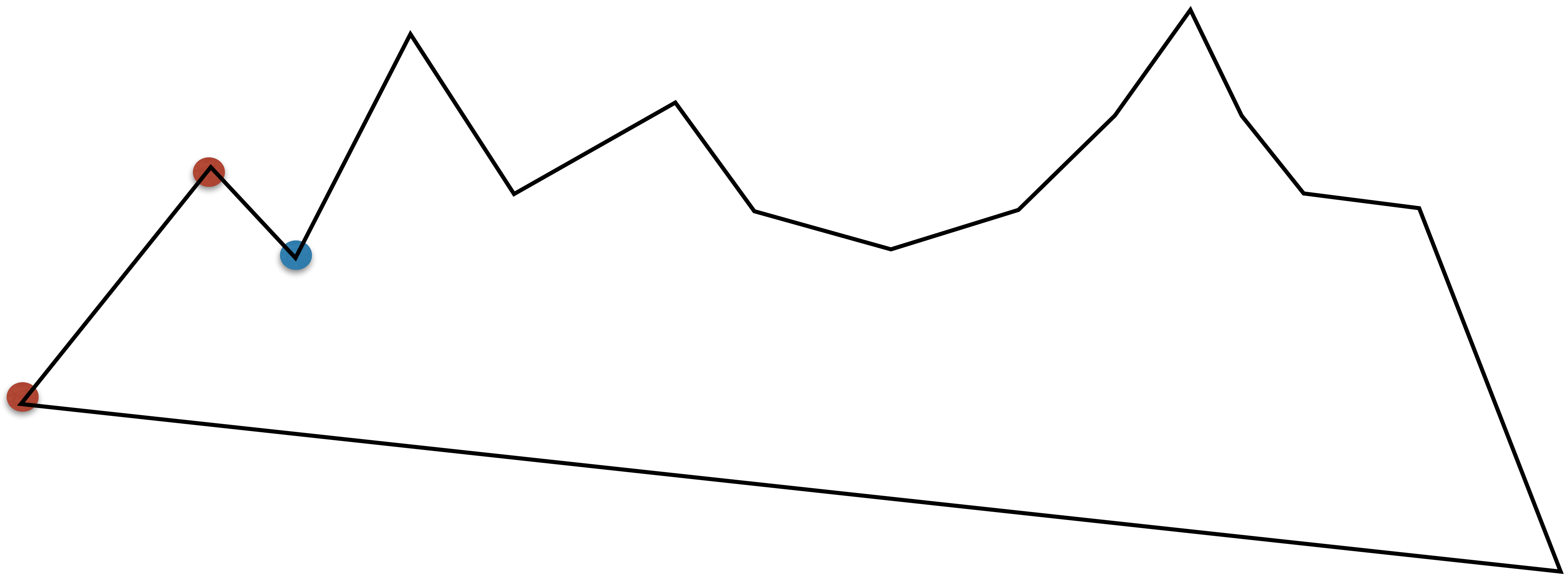


Class work: Let’s come up with an algorithm (and analyze it).

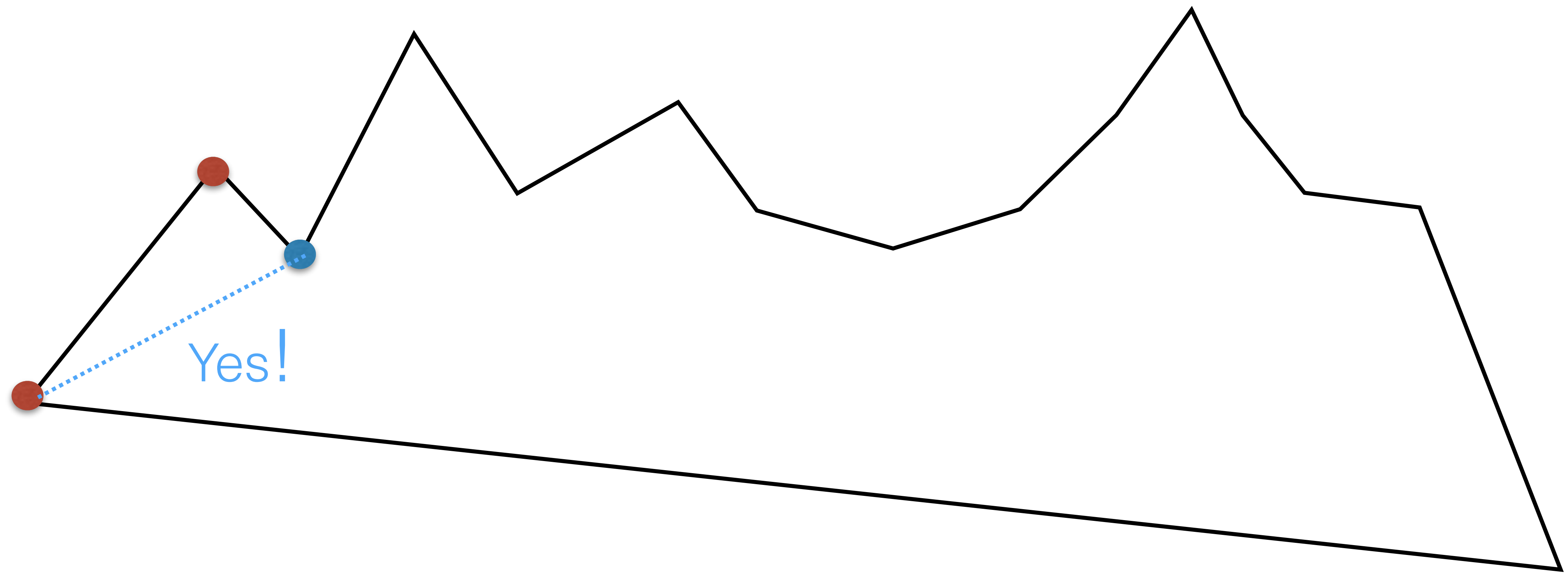
Monotone mountains are easy to triangulate!



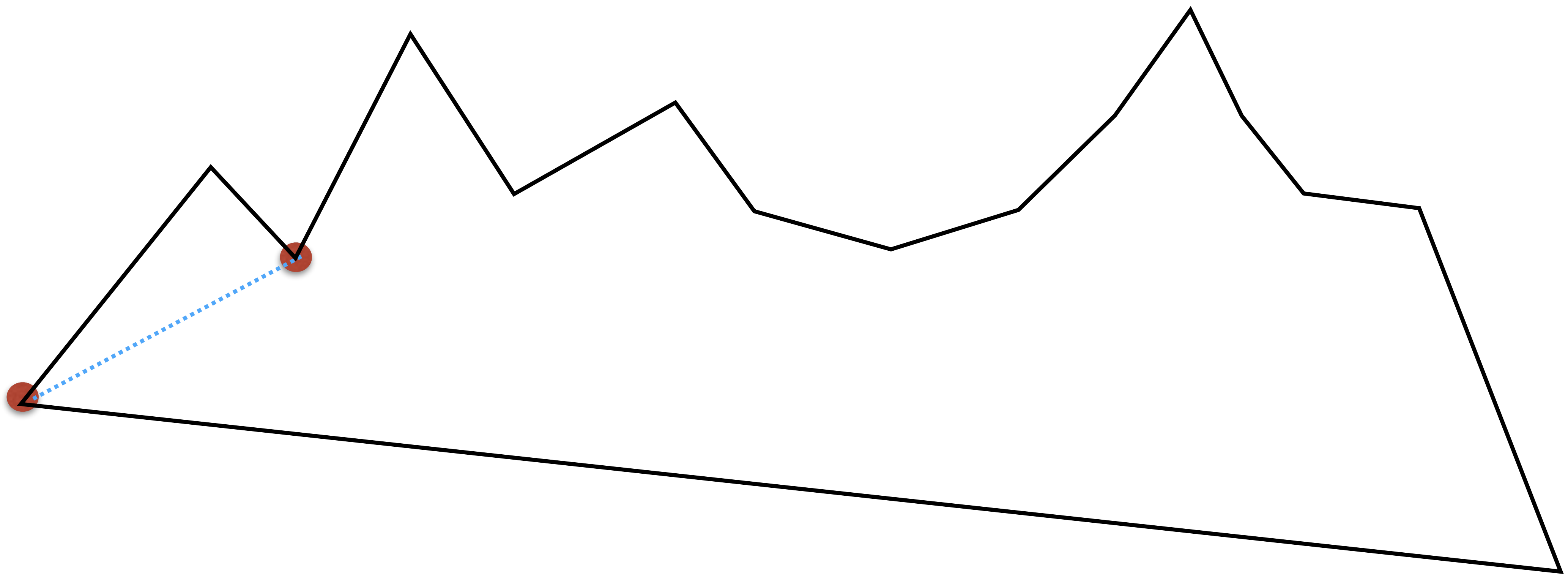
Monotone mountains are easy to triangulate!



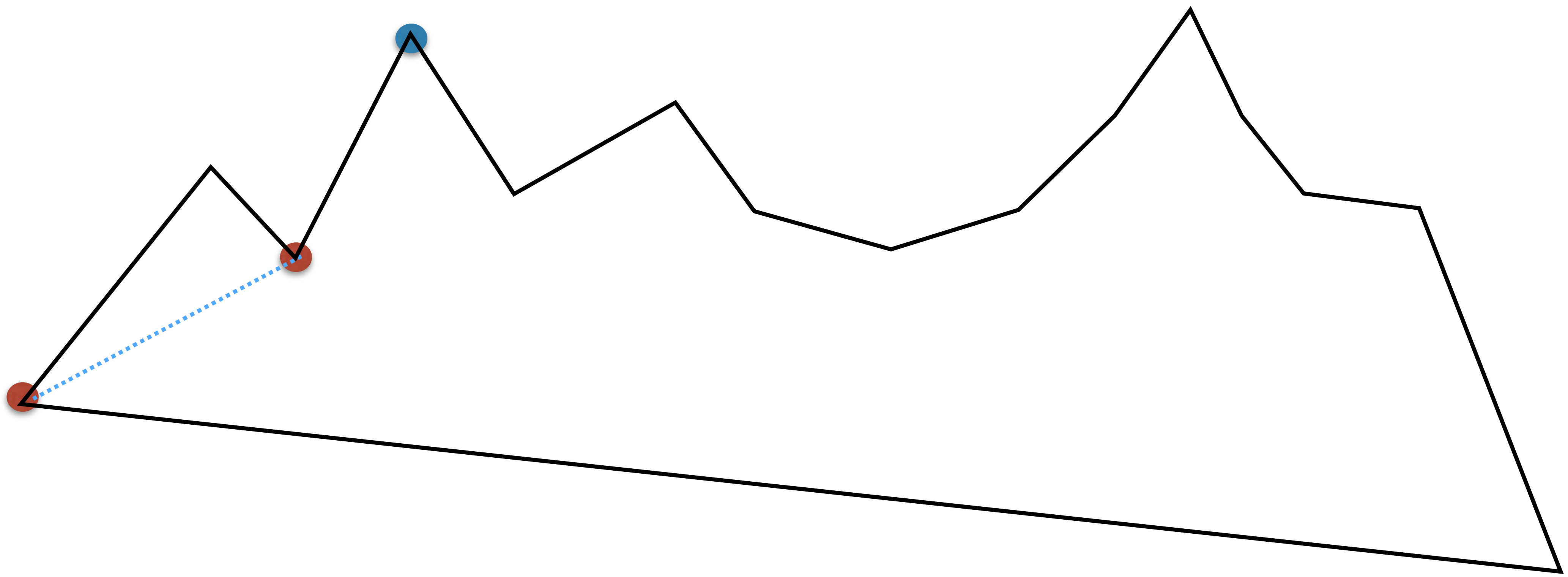
Monotone mountains are easy to triangulate!



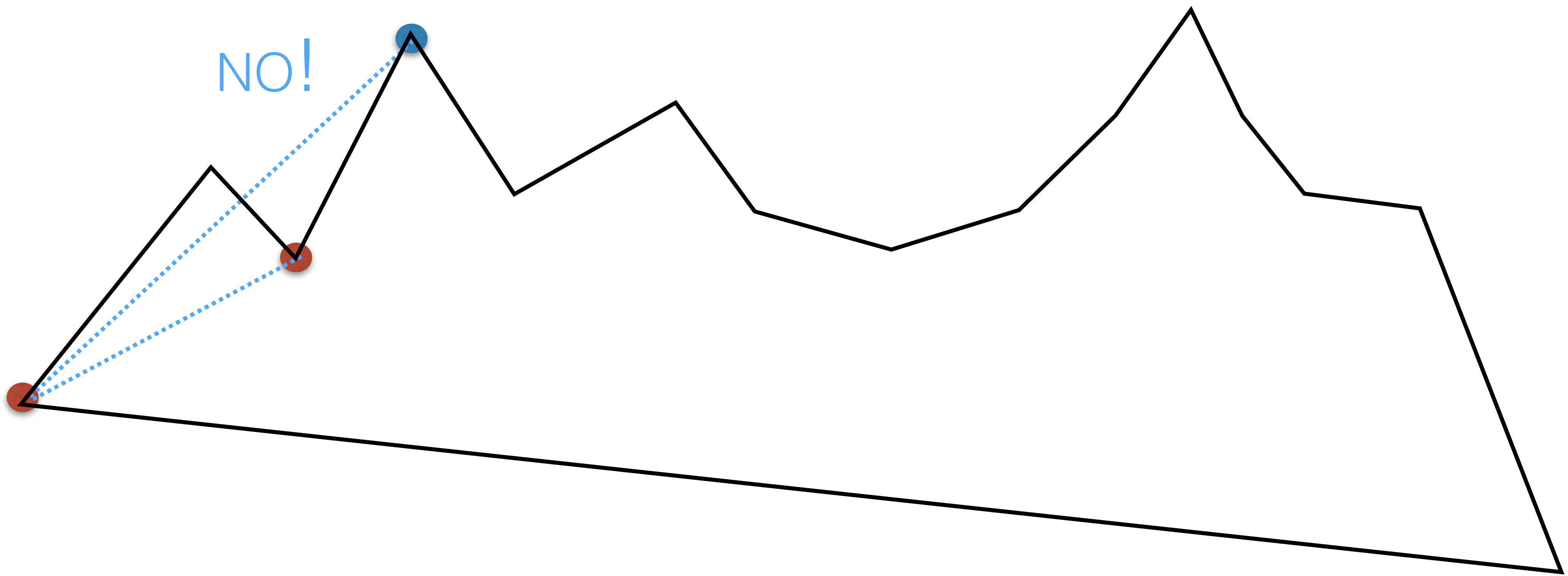
Monotone mountains are easy to triangulate!



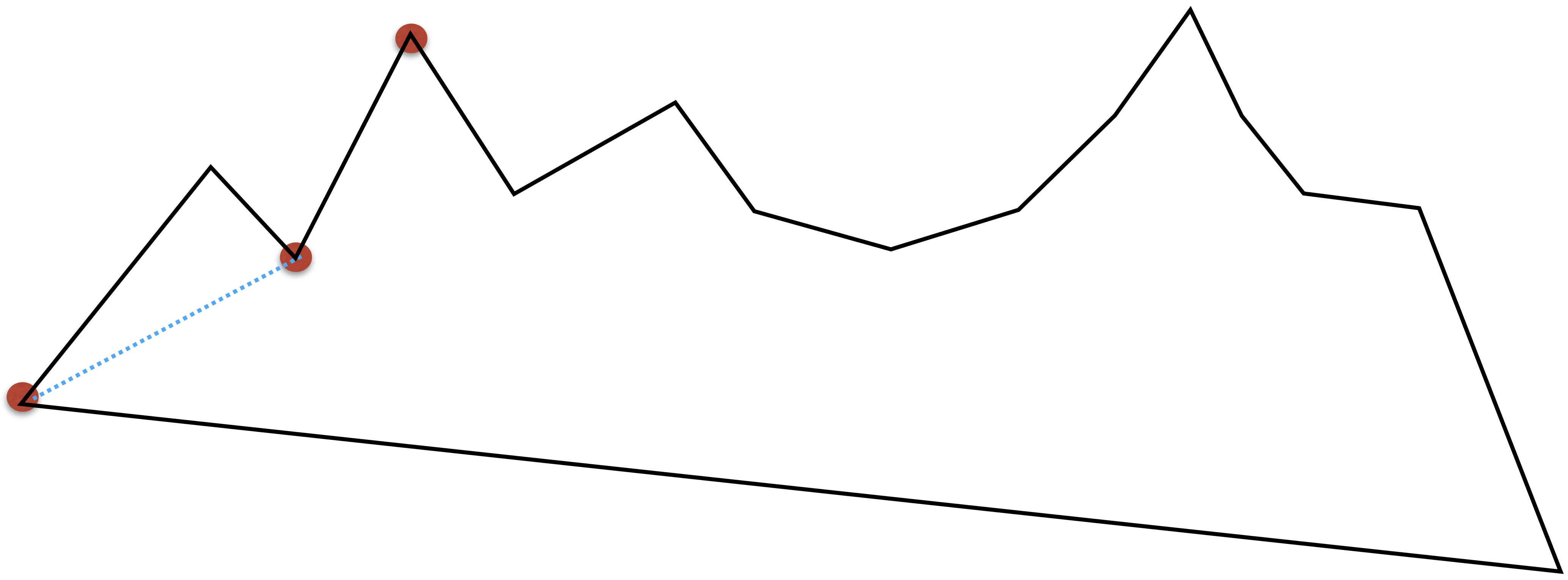
Monotone mountains are easy to triangulate!



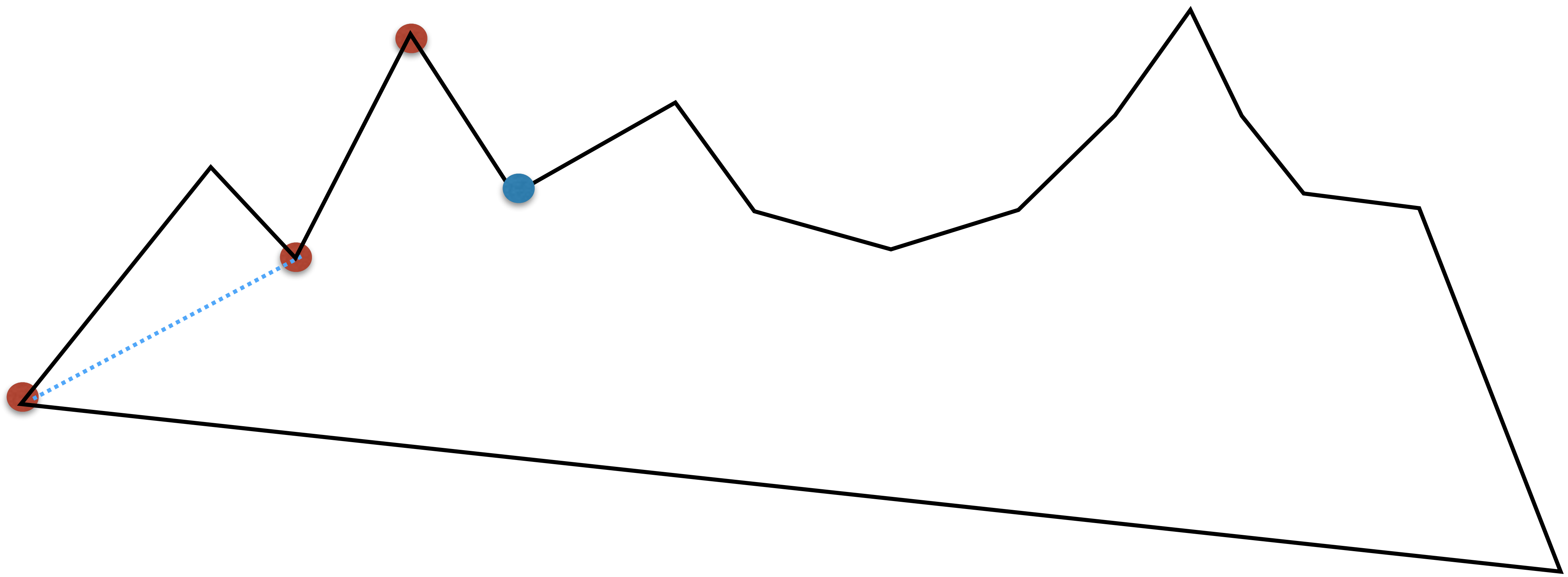
Monotone mountains are easy to triangulate!



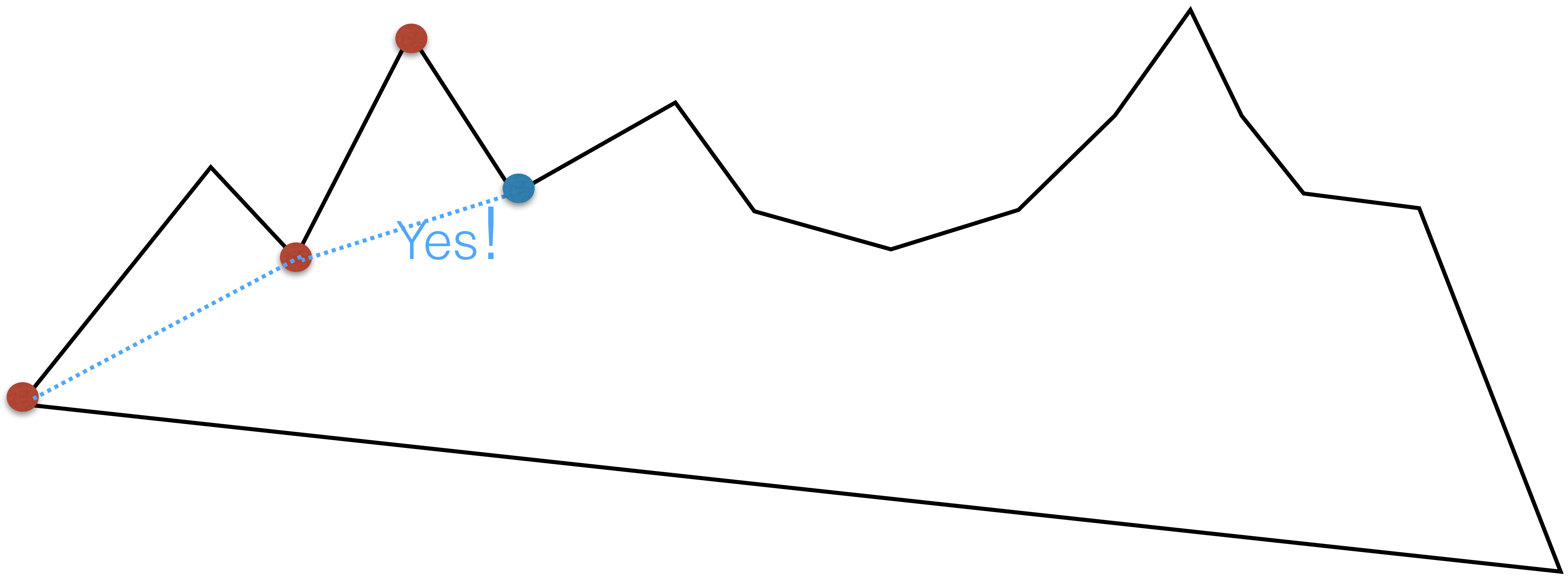
Monotone mountains are easy to triangulate!



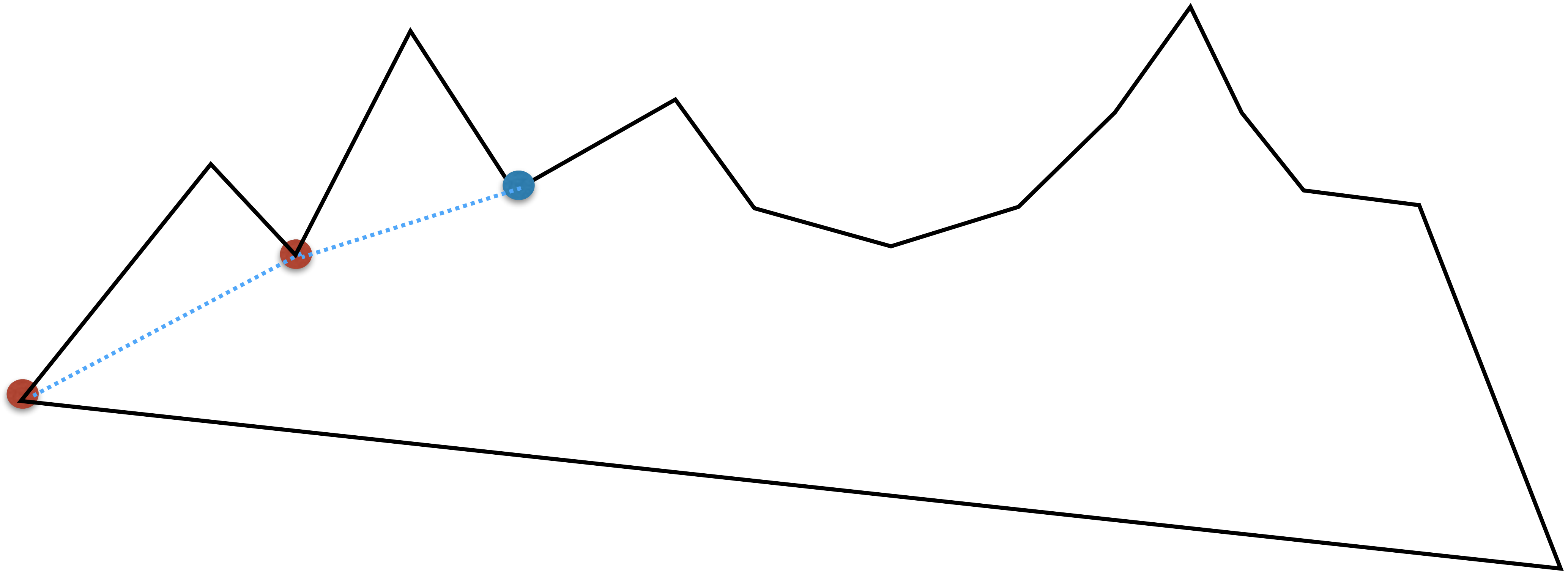
Monotone mountains are easy to triangulate!



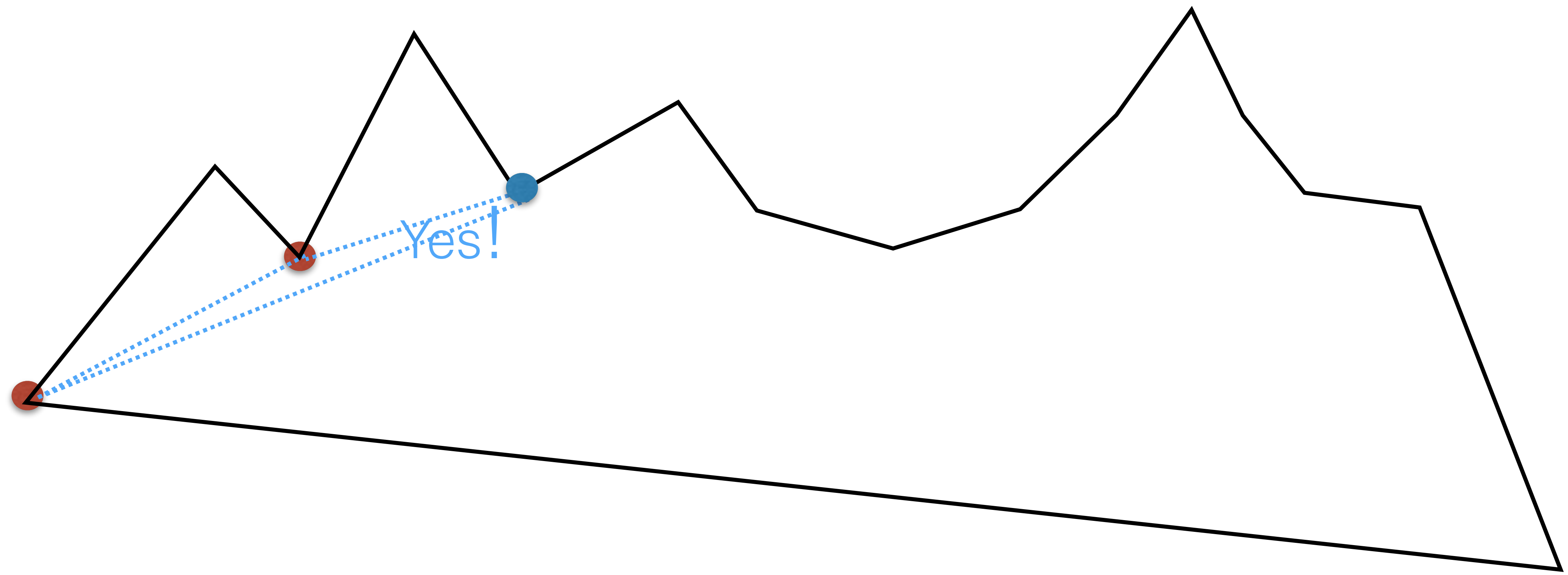
Monotone mountains are easy to triangulate!



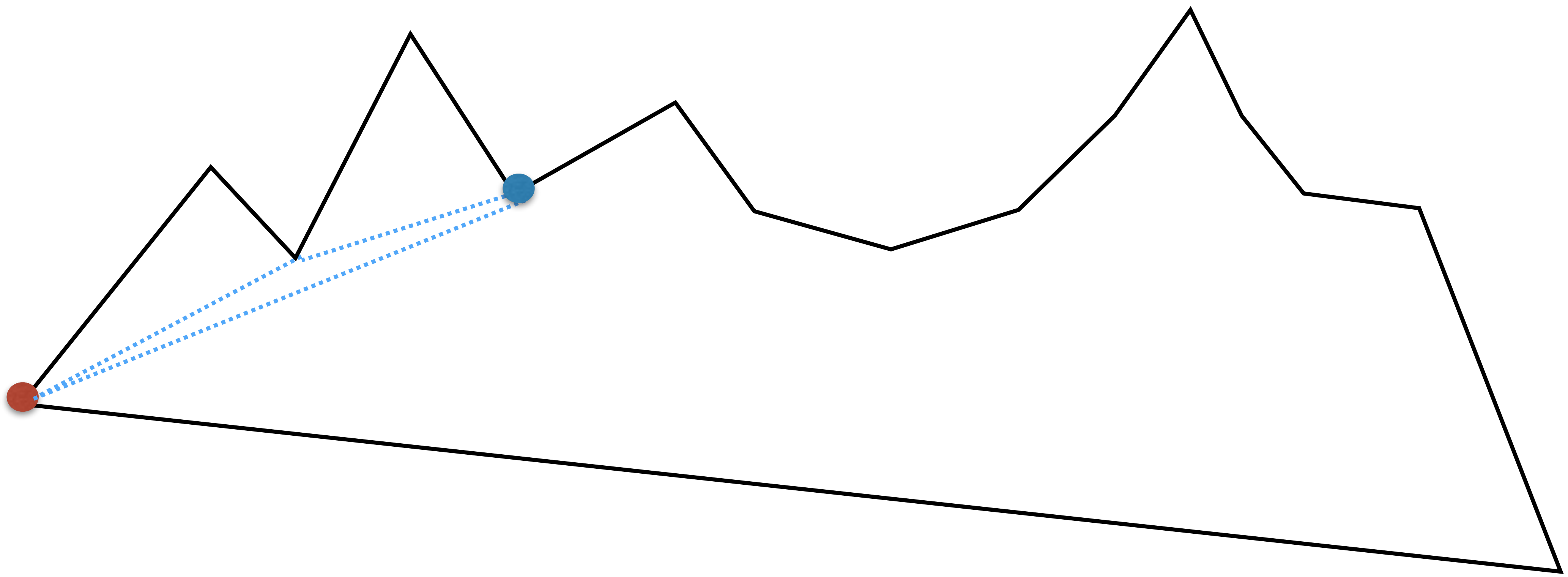
Monotone mountains are easy to triangulate!



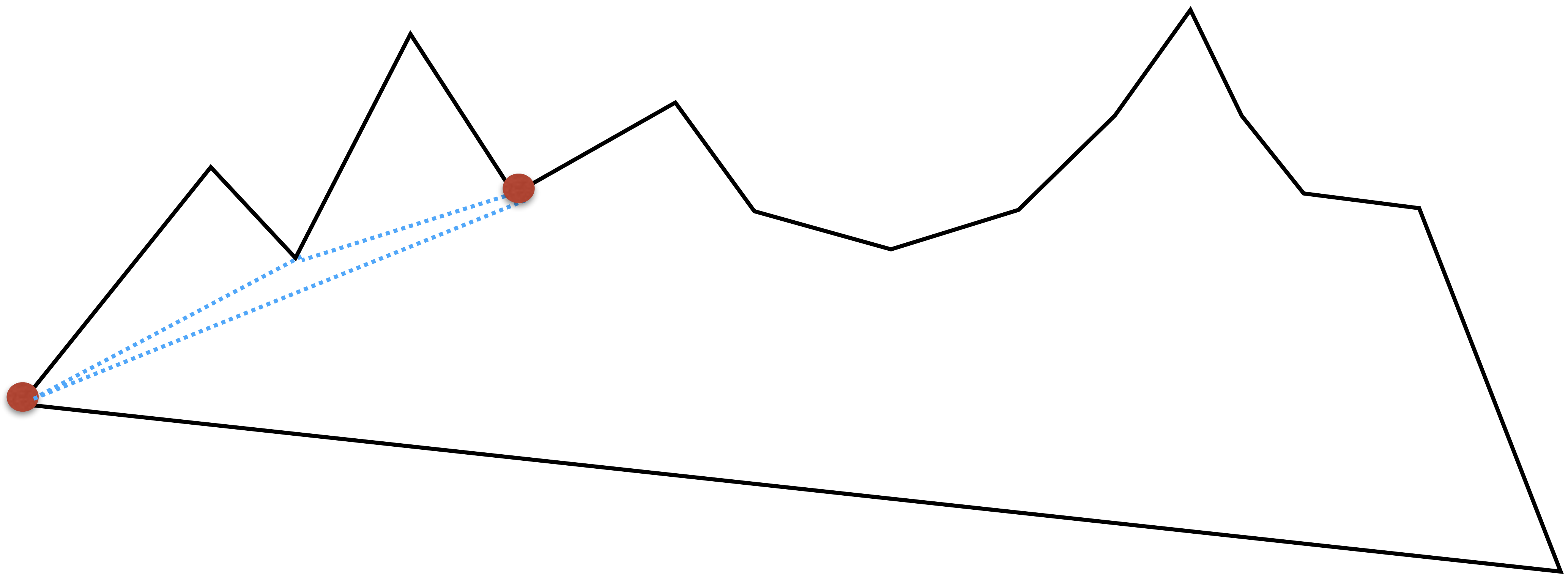
Monotone mountains are easy to triangulate!



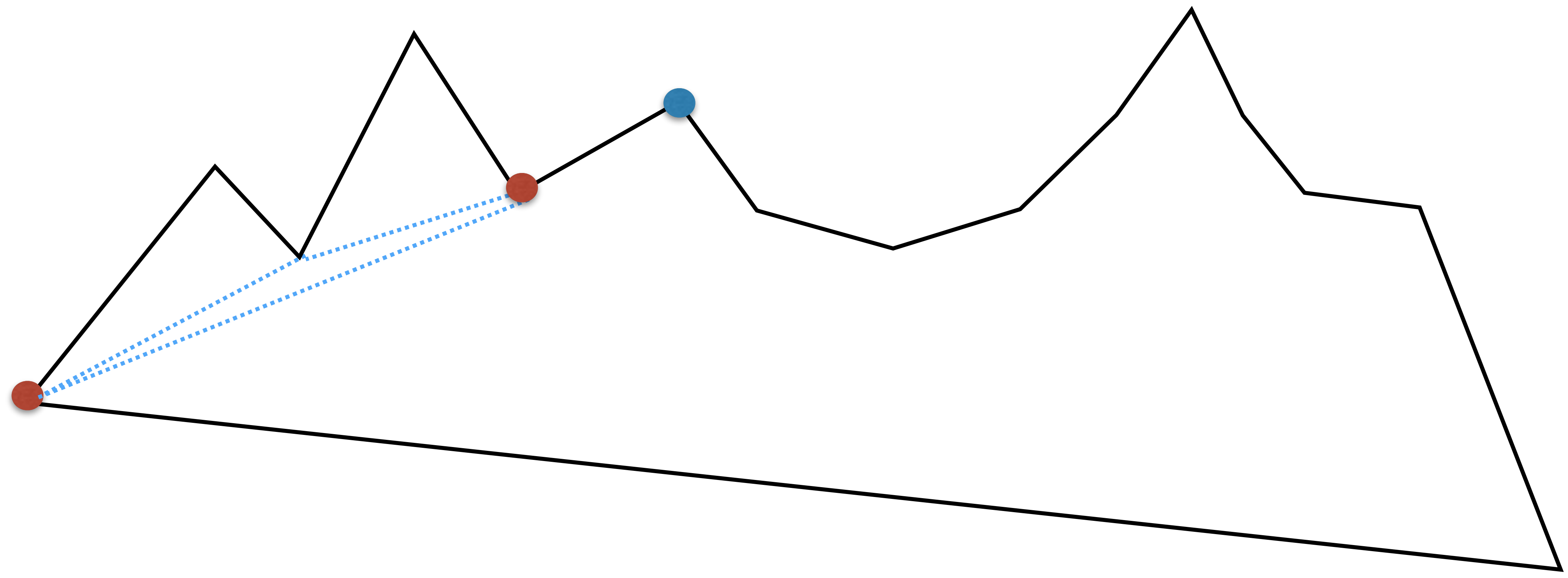
Monotone mountains are easy to triangulate!



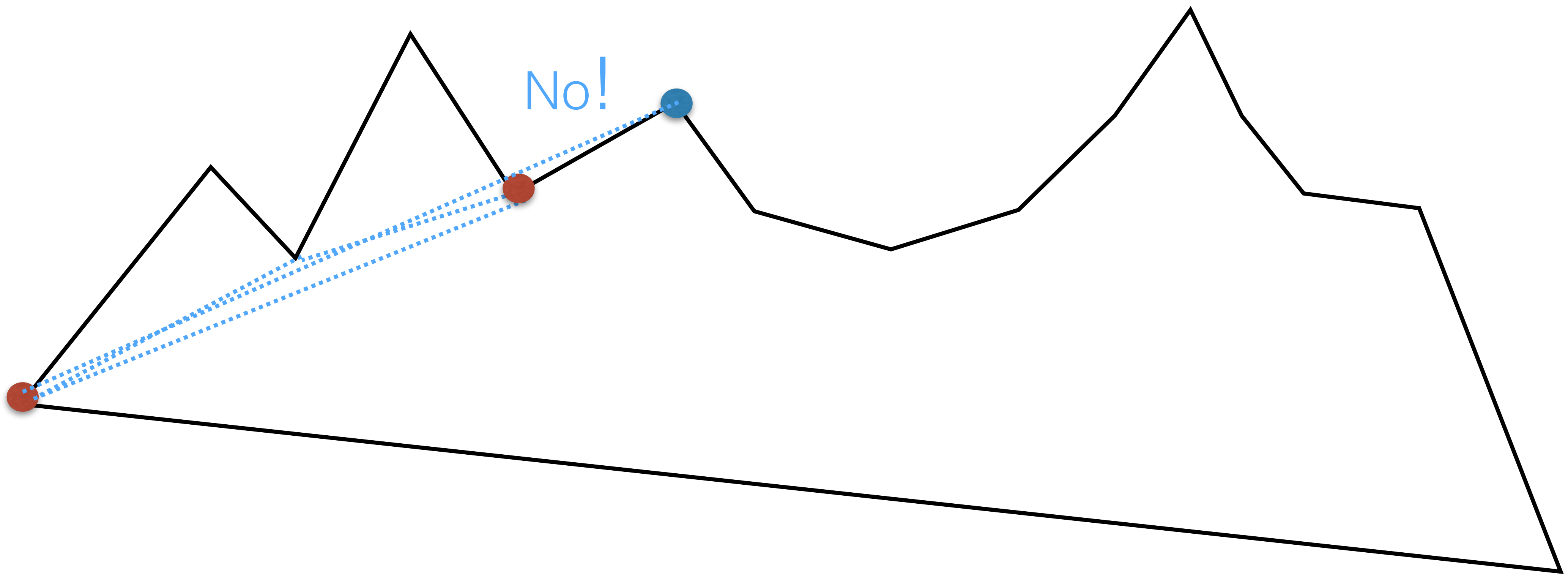
Monotone mountains are easy to triangulate!



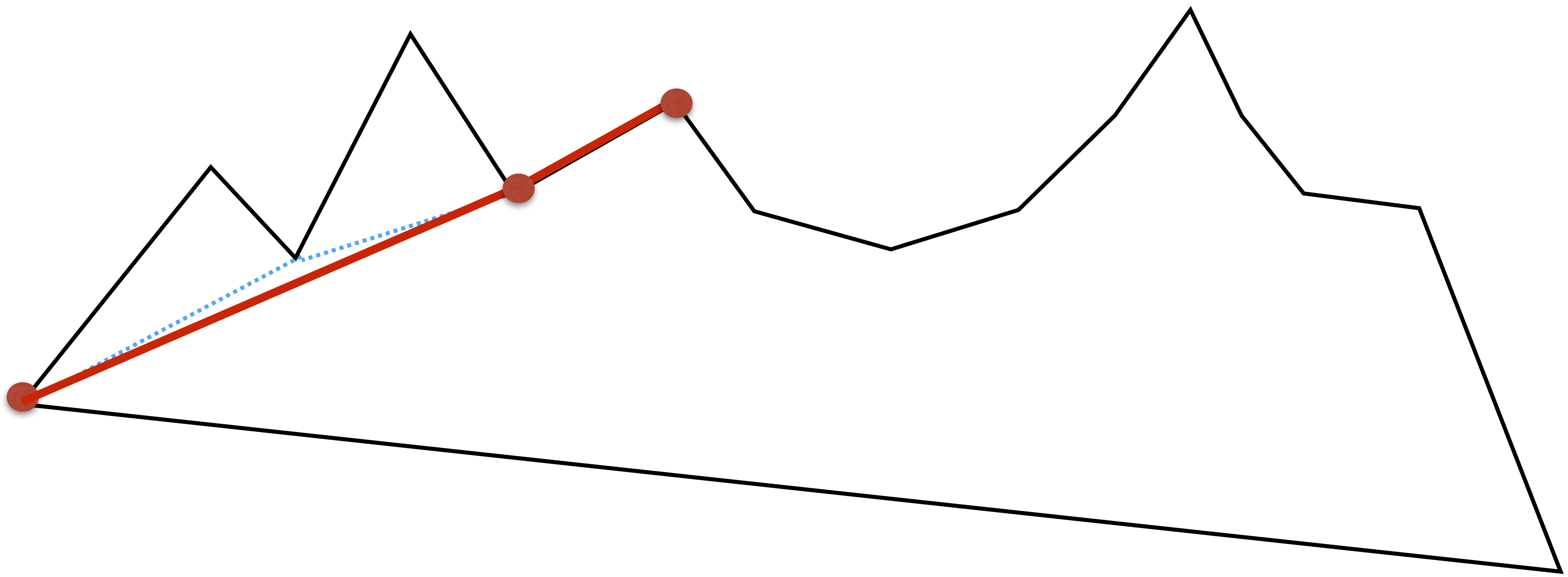
Monotone mountains are easy to triangulate!



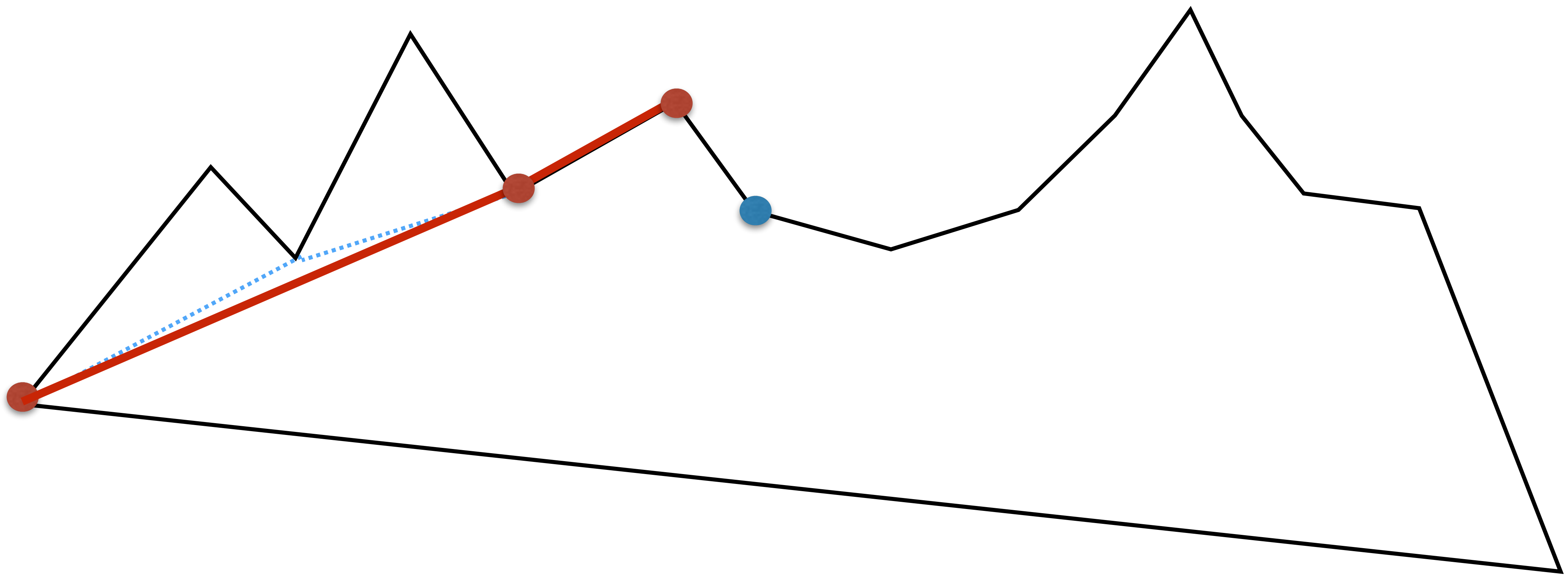
Monotone mountains are easy to triangulate!



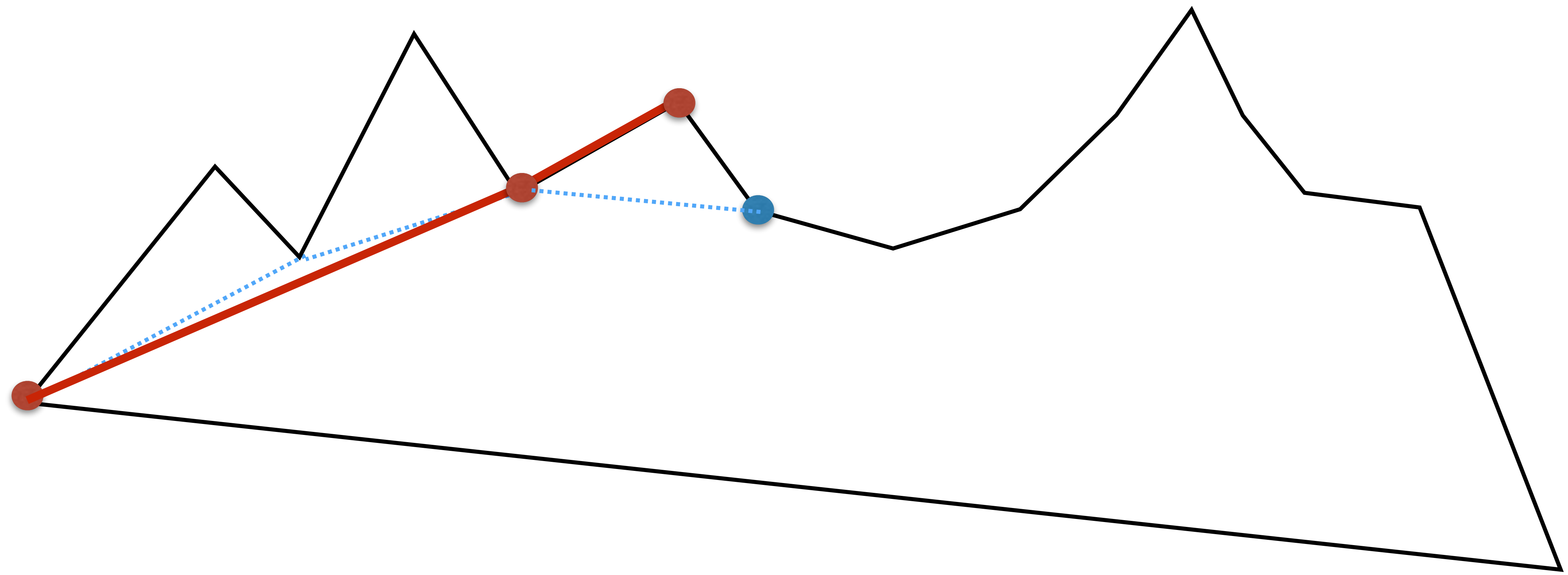
Monotone mountains are easy to triangulate!



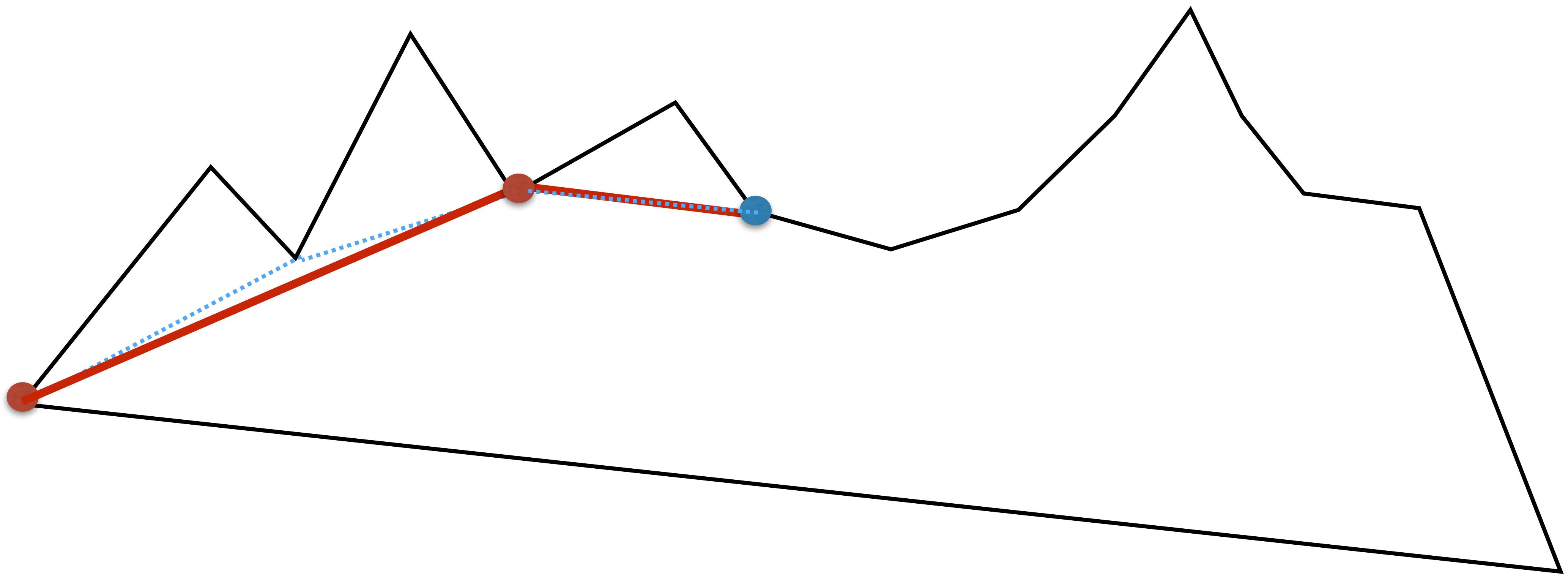
Monotone mountains are easy to triangulate!



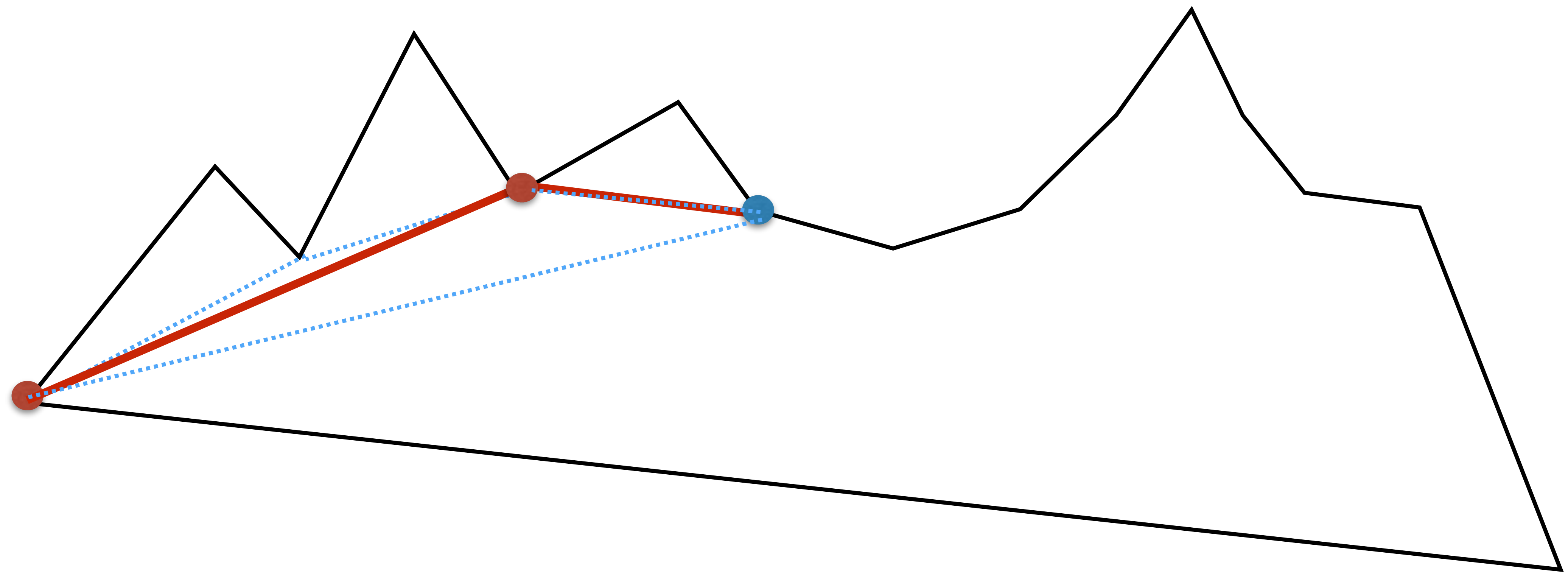
Monotone mountains are easy to triangulate!



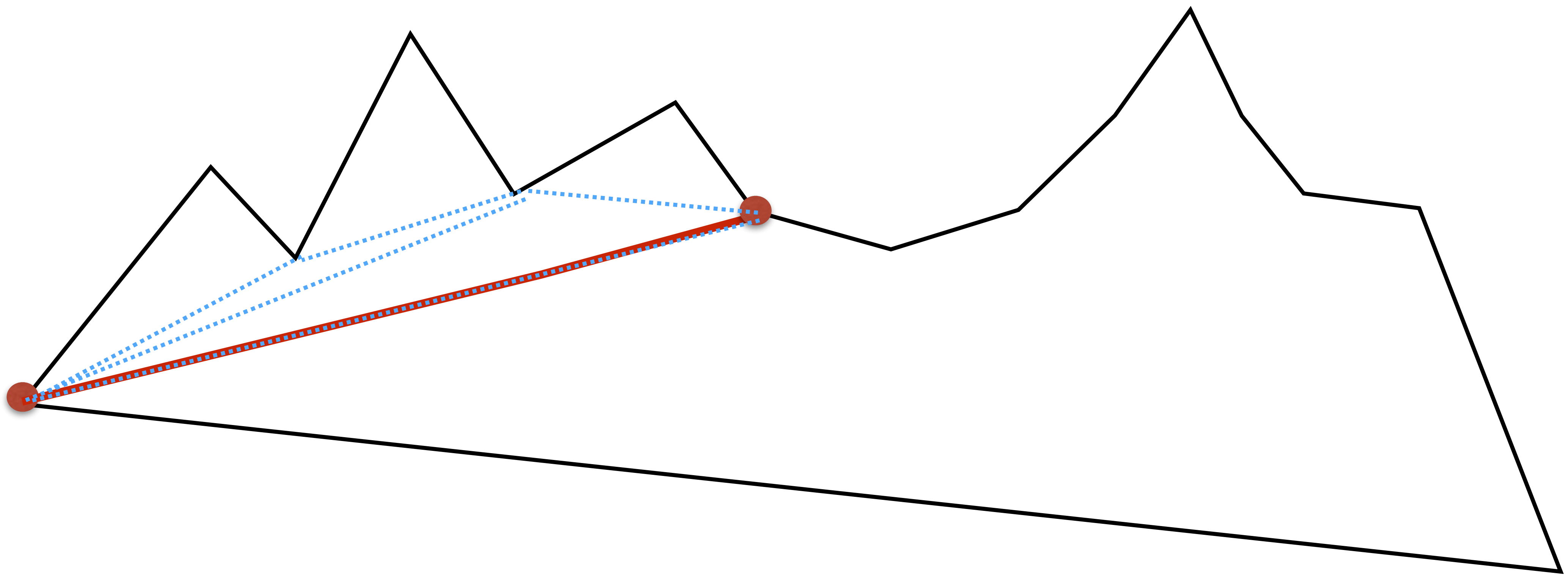
Monotone mountains are easy to triangulate!



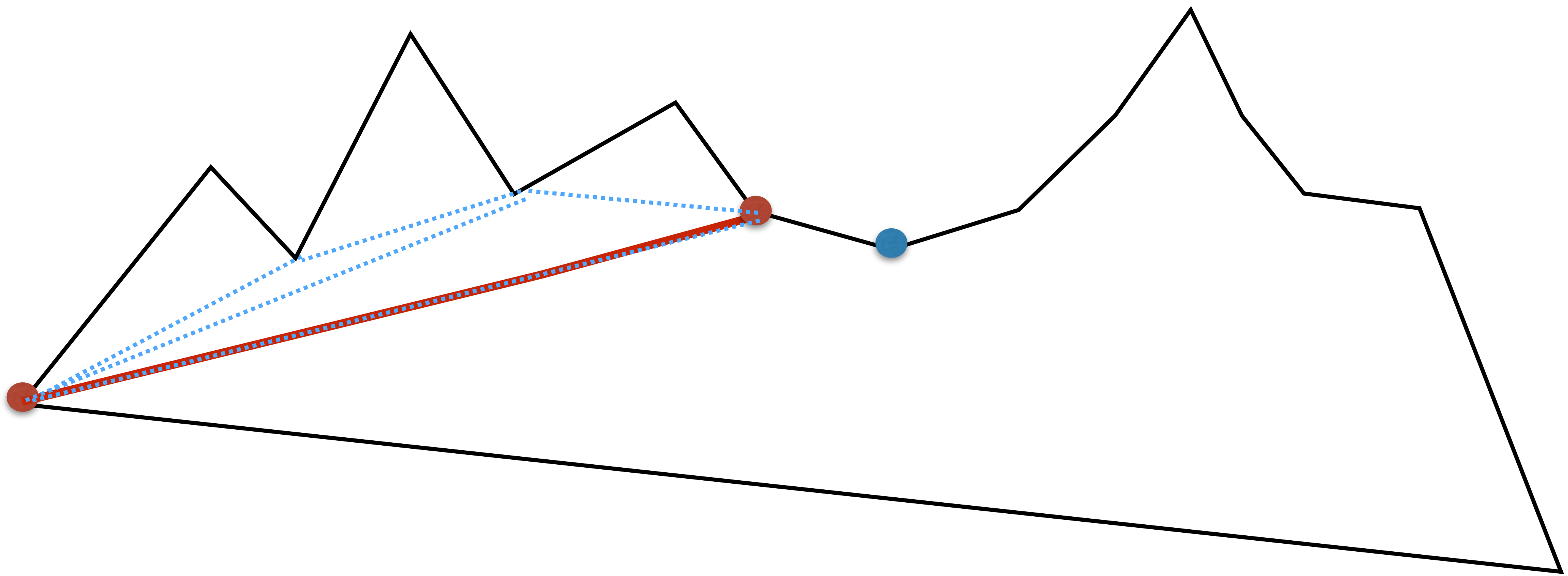
Monotone mountains are easy to triangulate!



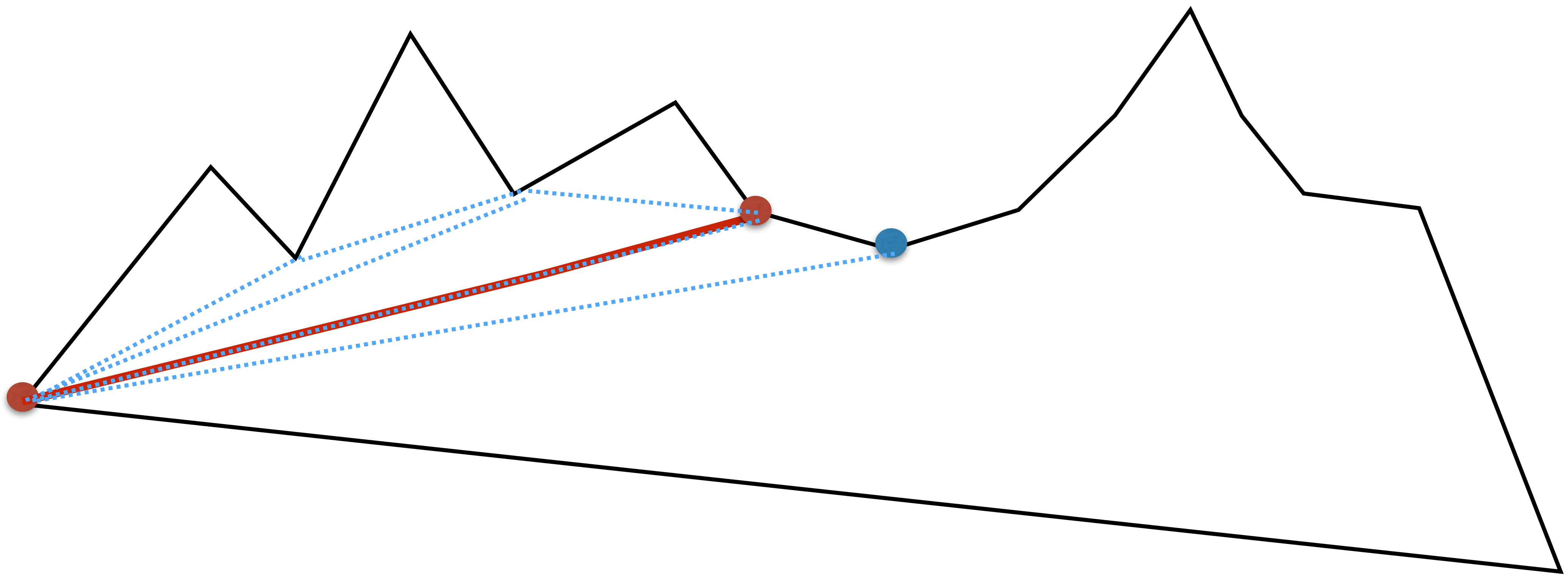
Monotone mountains are easy to triangulate!



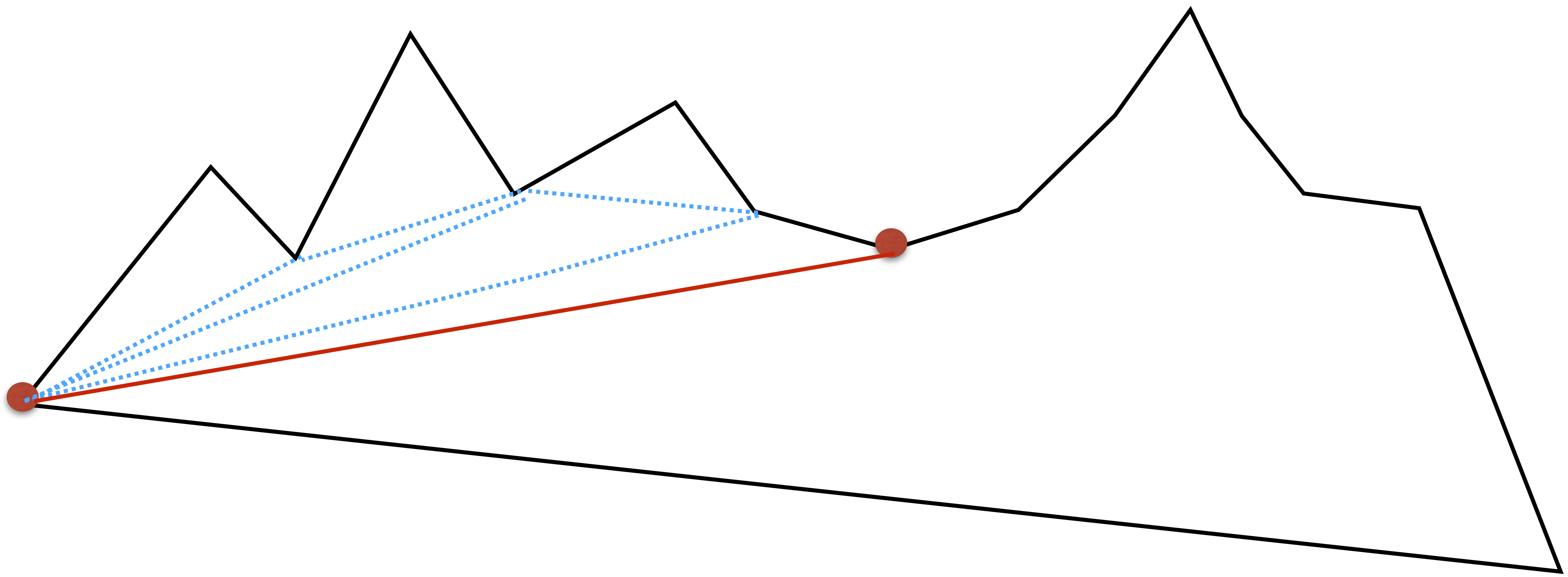
Monotone mountains are easy to triangulate!



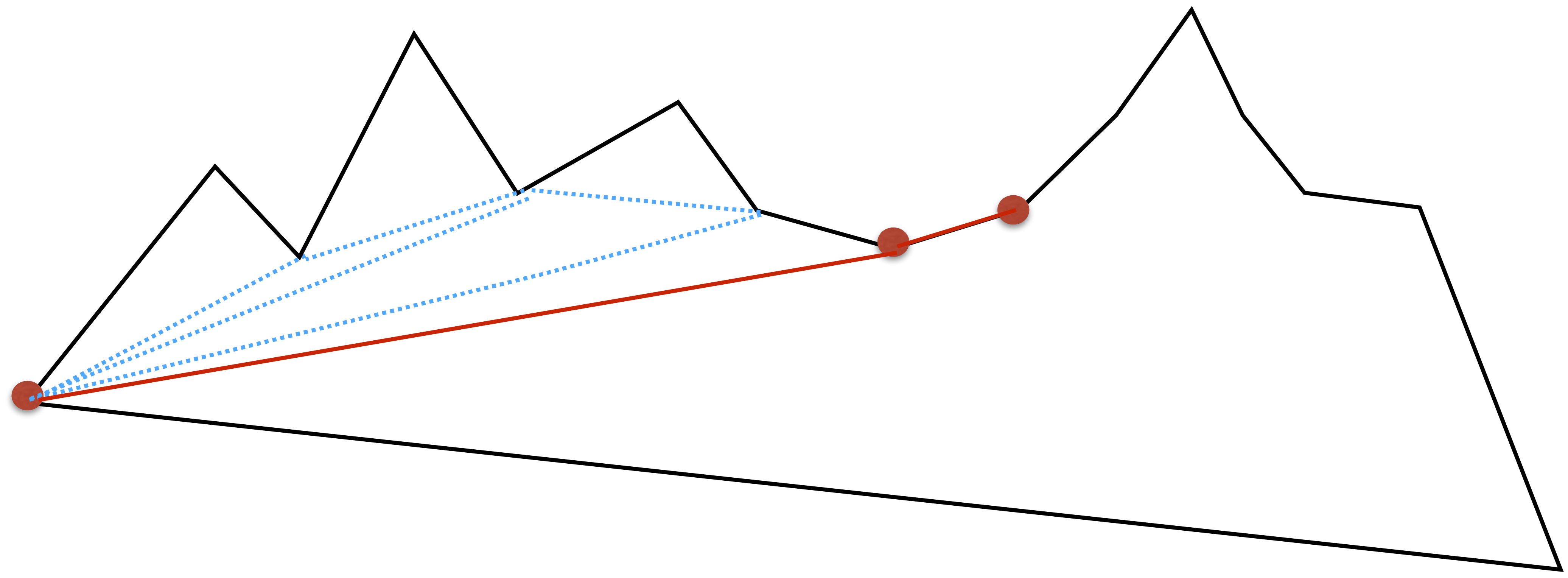
Monotone mountains are easy to triangulate!



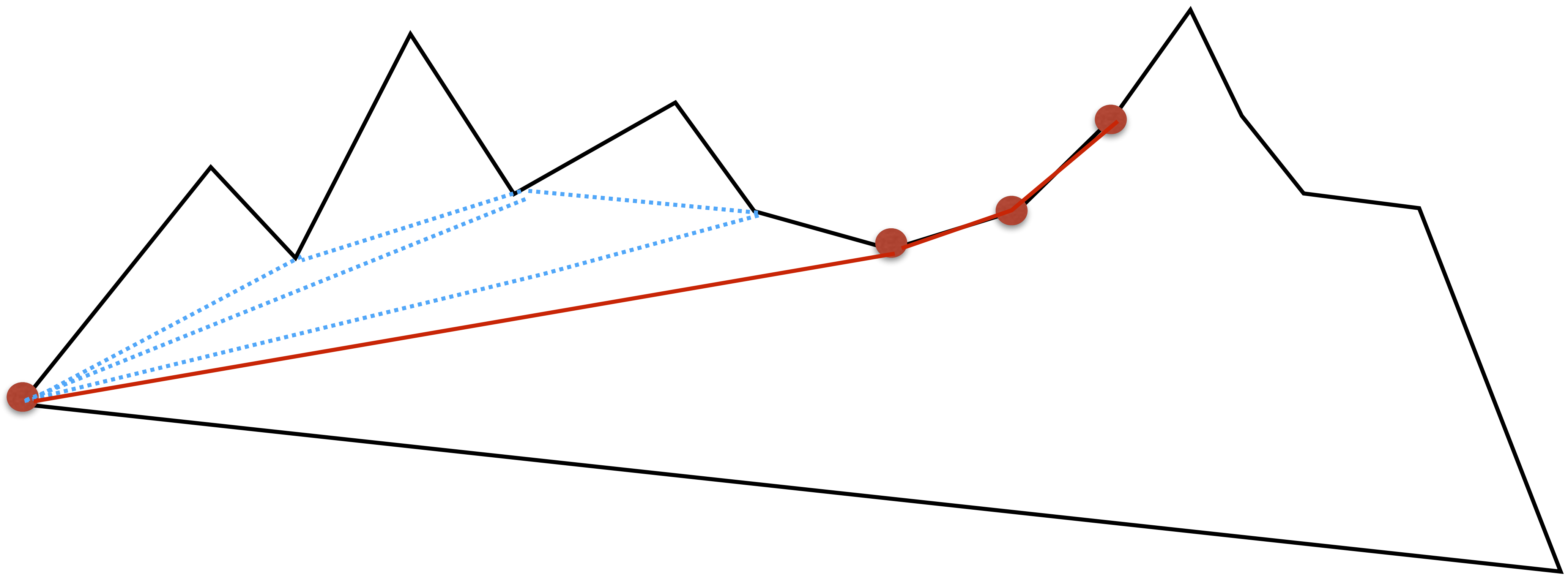
Monotone mountains are easy to triangulate!



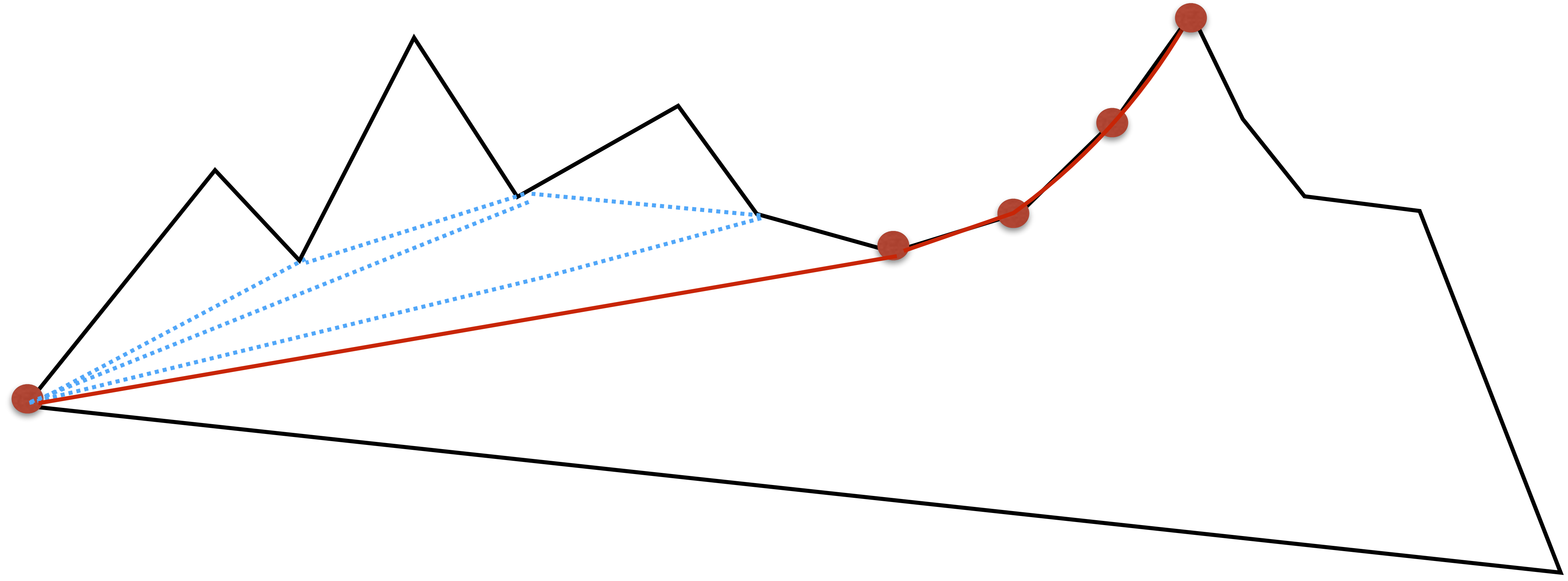
Monotone mountains are easy to triangulate!



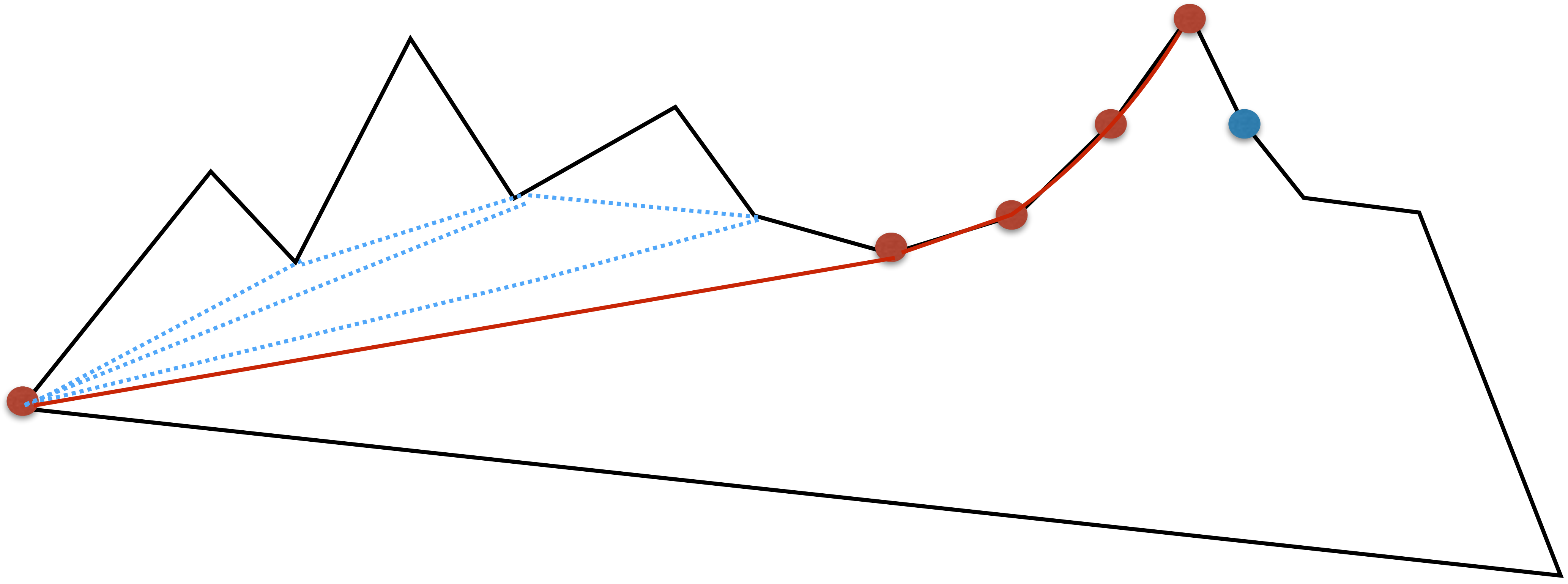
Monotone mountains are easy to triangulate!



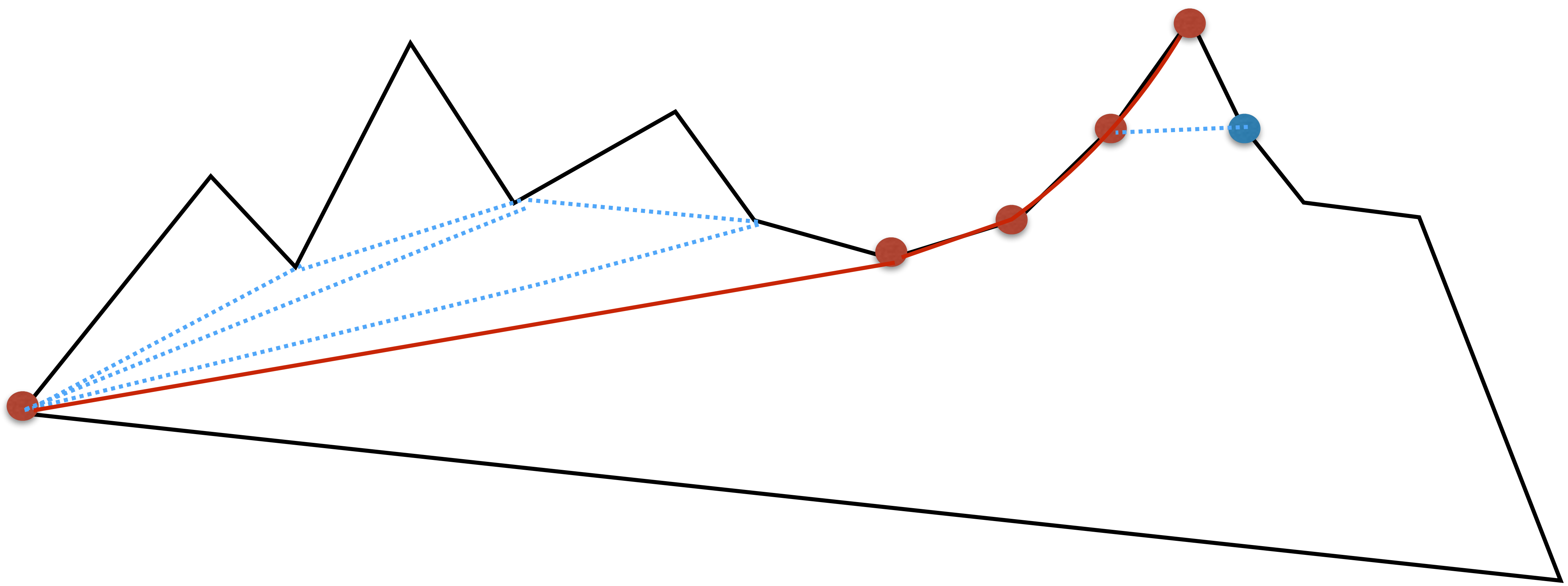
Monotone mountains are easy to triangulate!



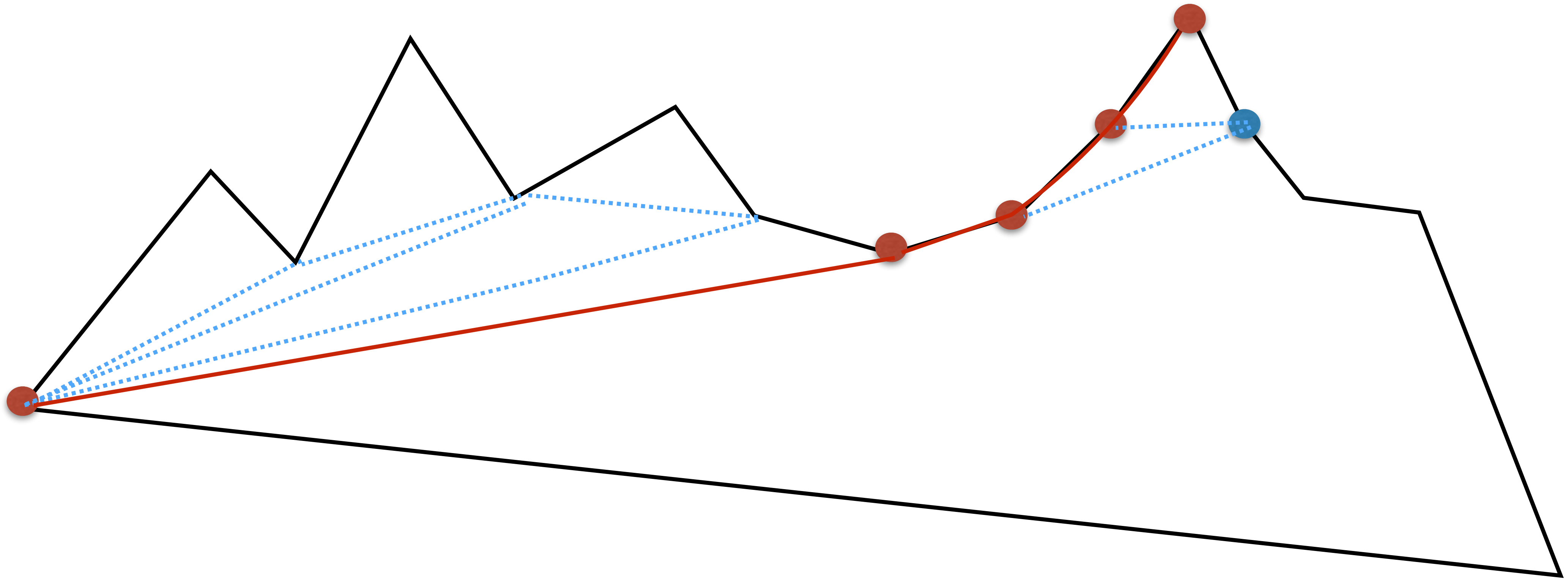
Monotone mountains are easy to triangulate!



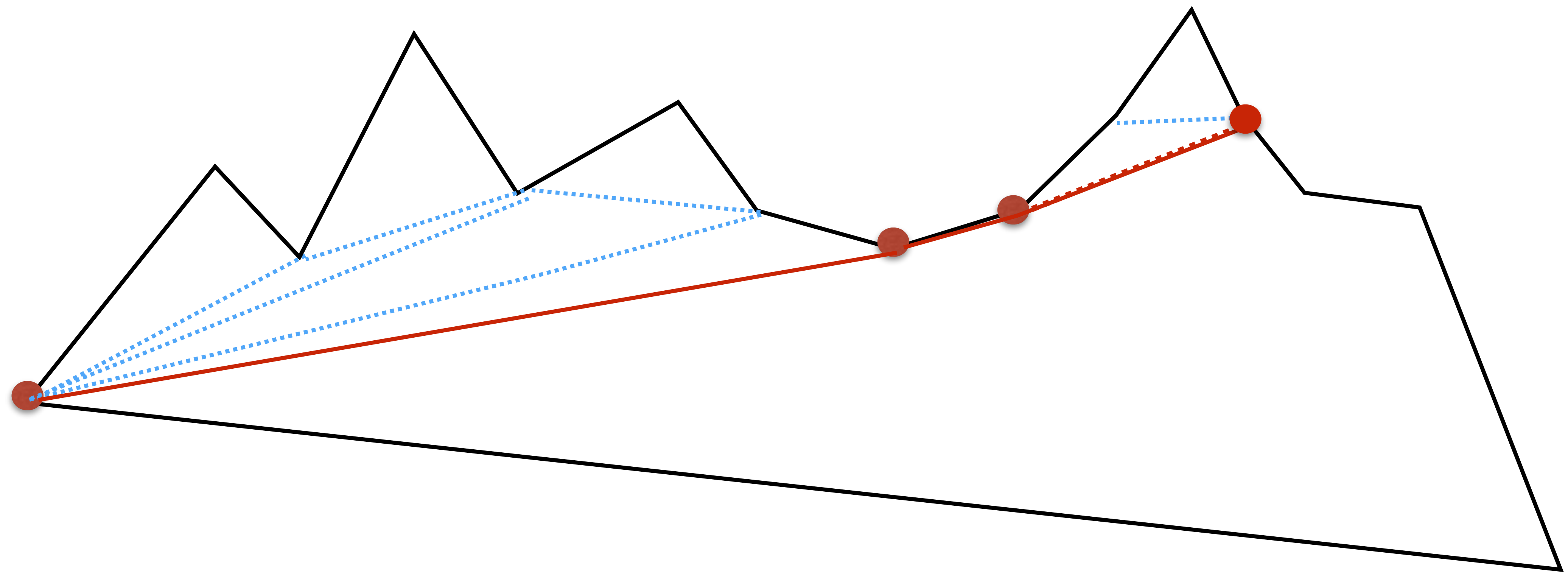
Monotone mountains are easy to triangulate!



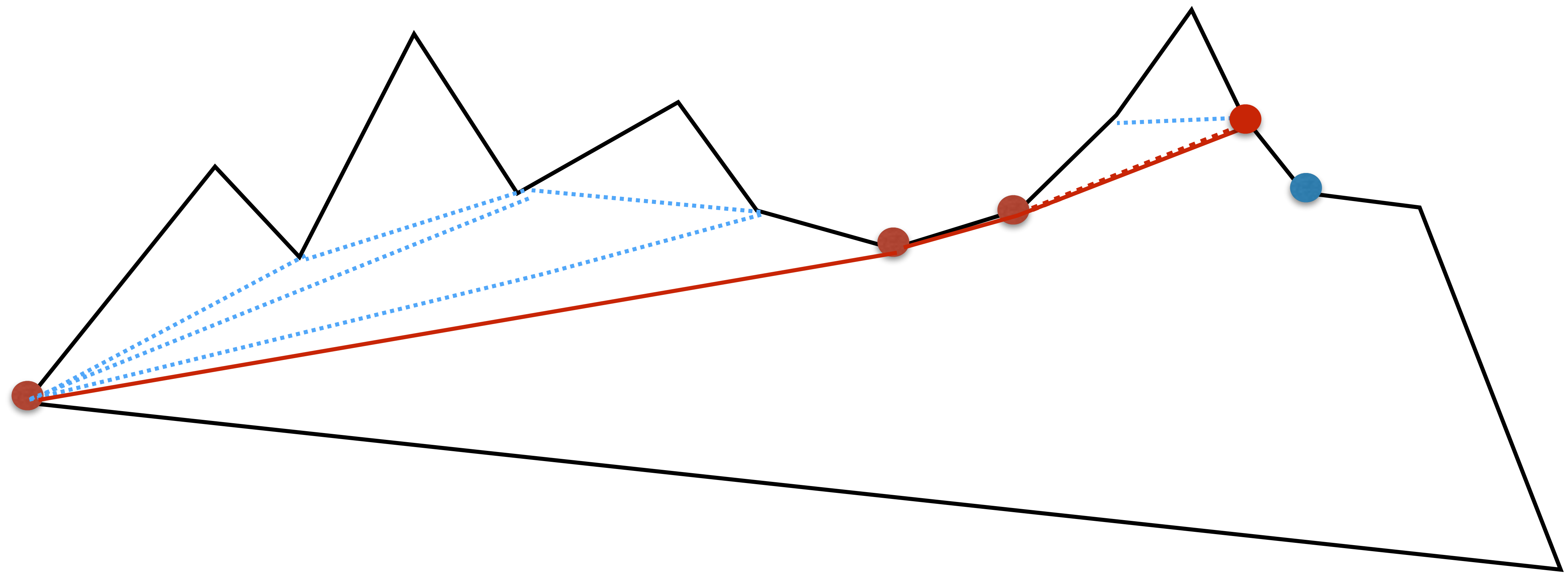
Monotone mountains are easy to triangulate!



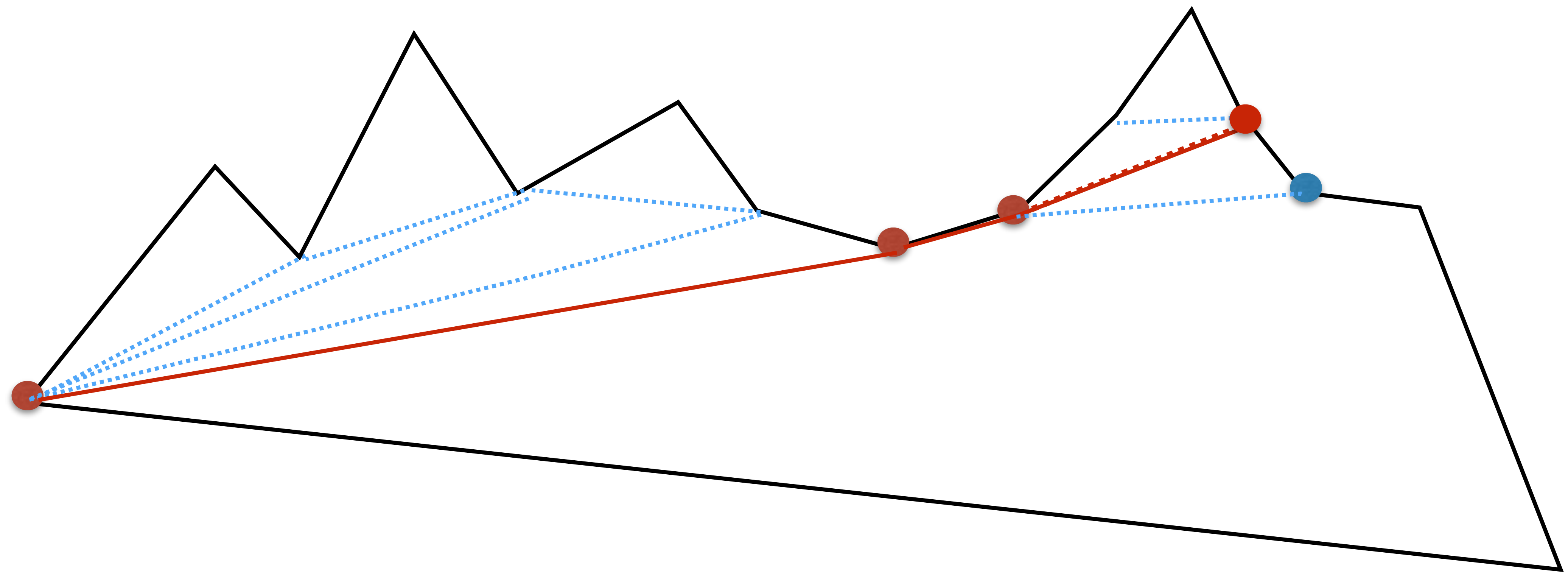
Monotone mountains are easy to triangulate!



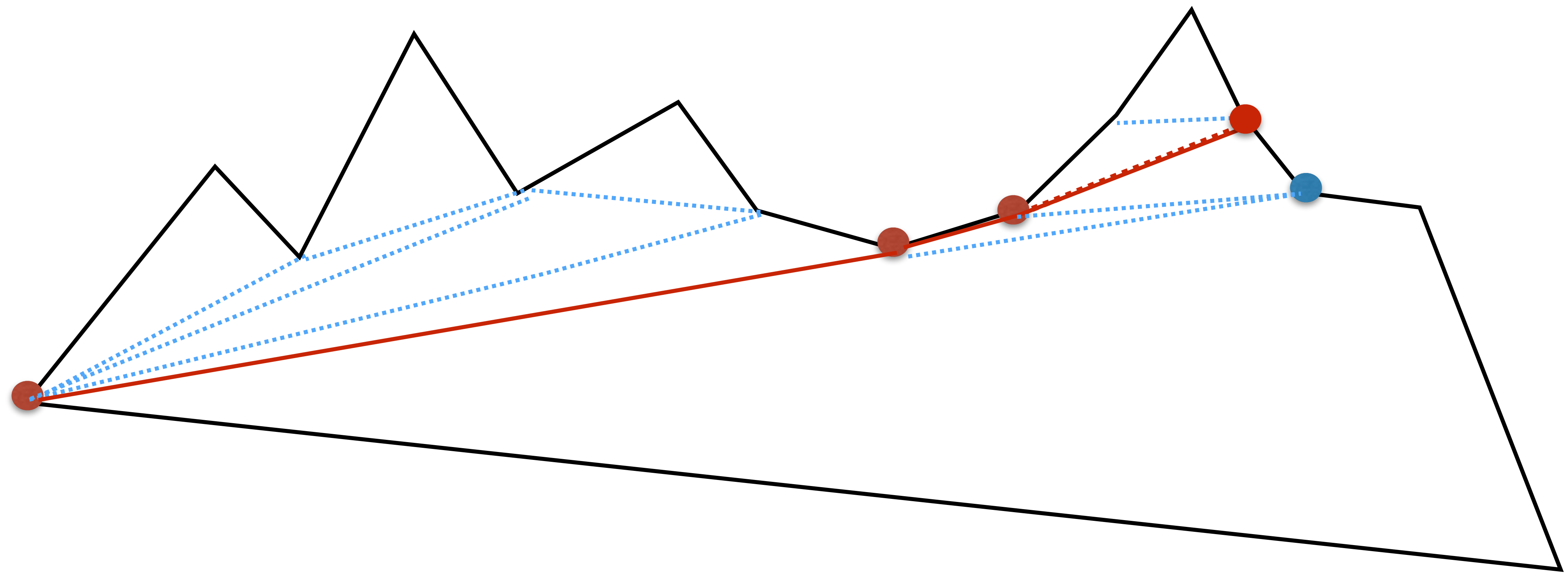
Monotone mountains are easy to triangulate!



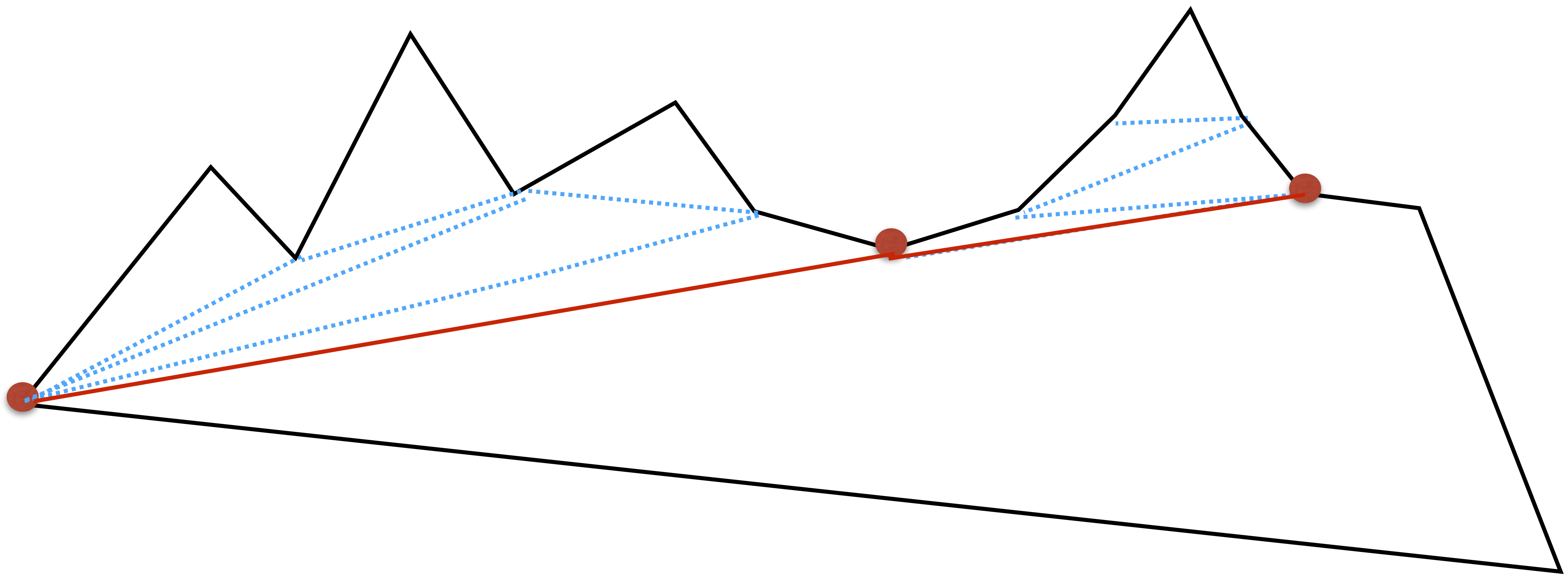
Monotone mountains are easy to triangulate!



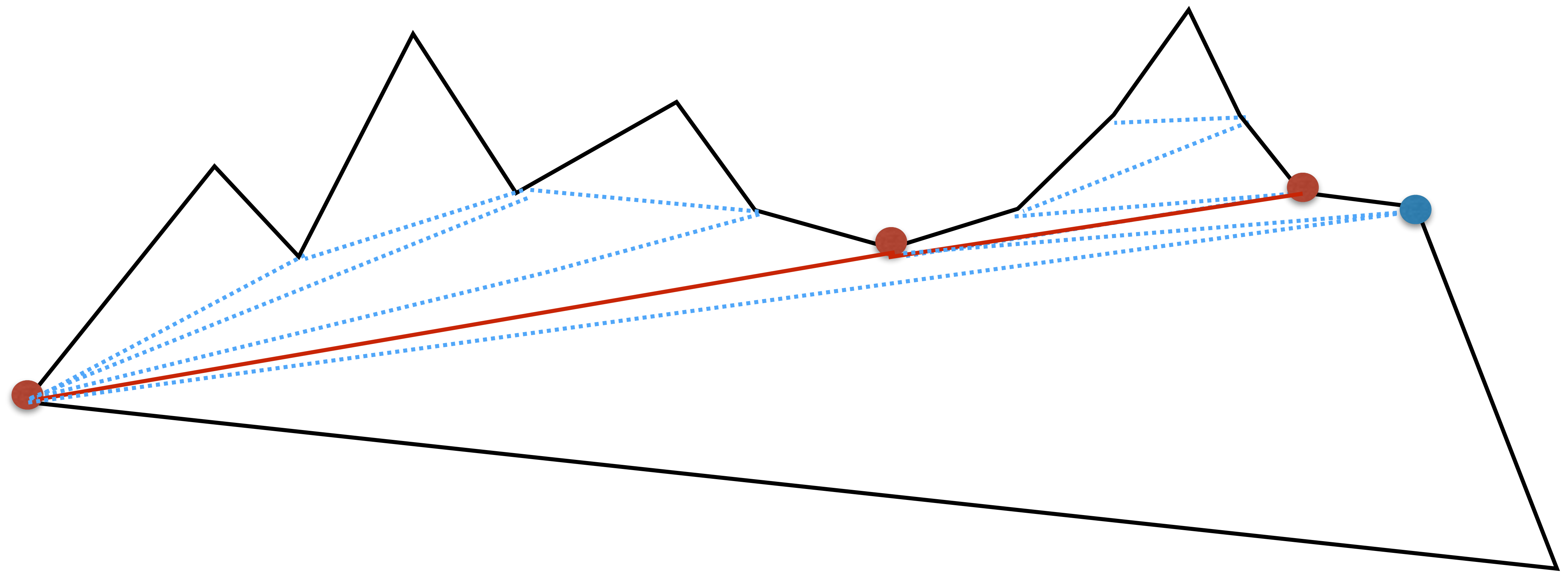
Monotone mountains are easy to triangulate!



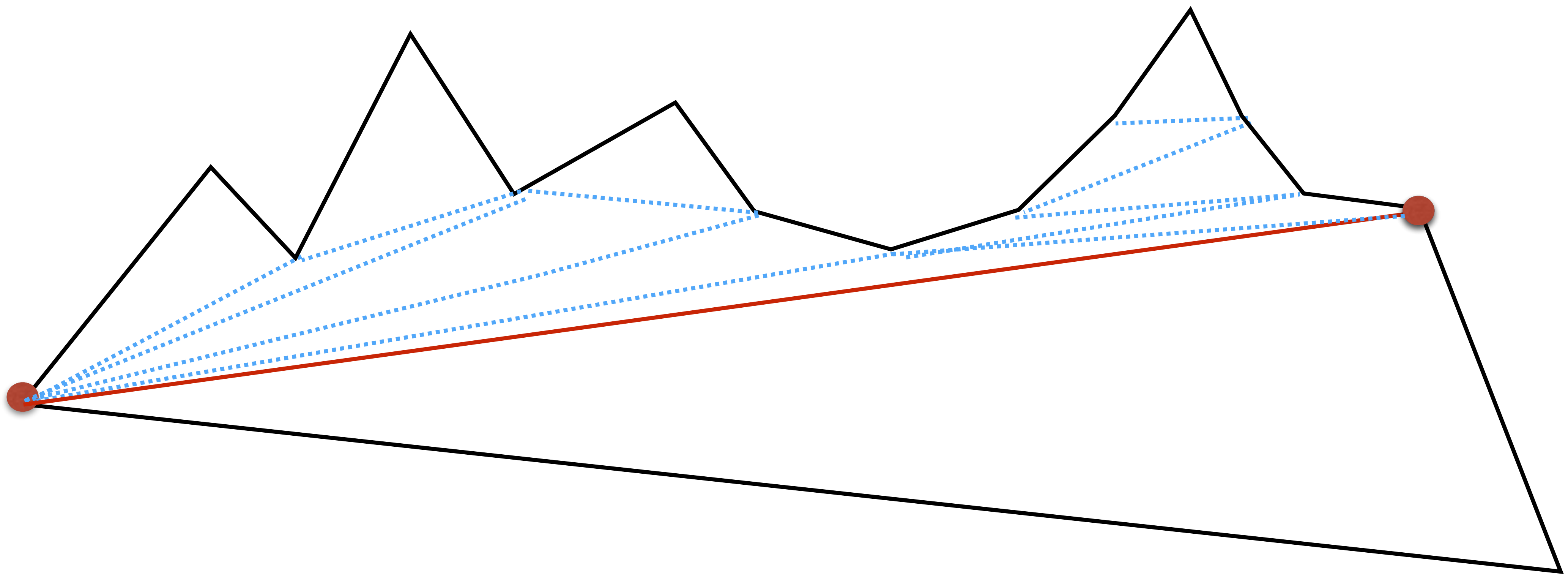
Monotone mountains are easy to triangulate!



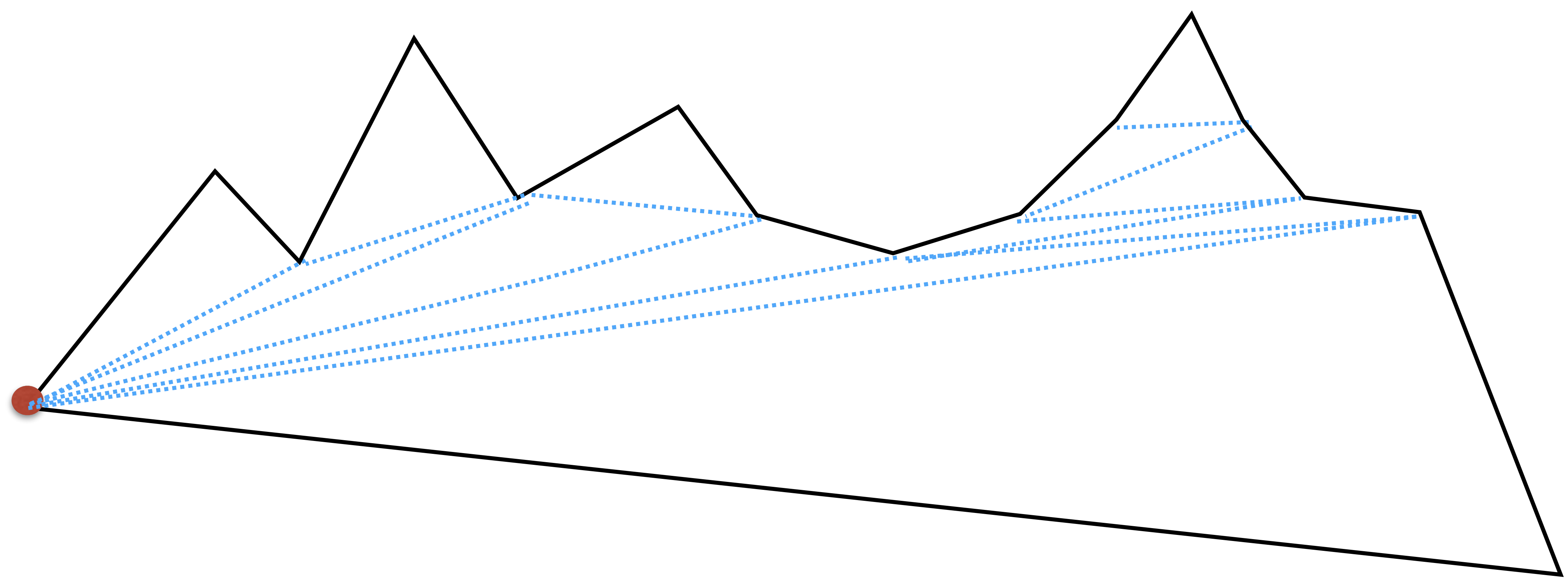
Monotone mountains are easy to triangulate!



Monotone mountains are easy to triangulate!



Monotone mountains are easy to triangulate!

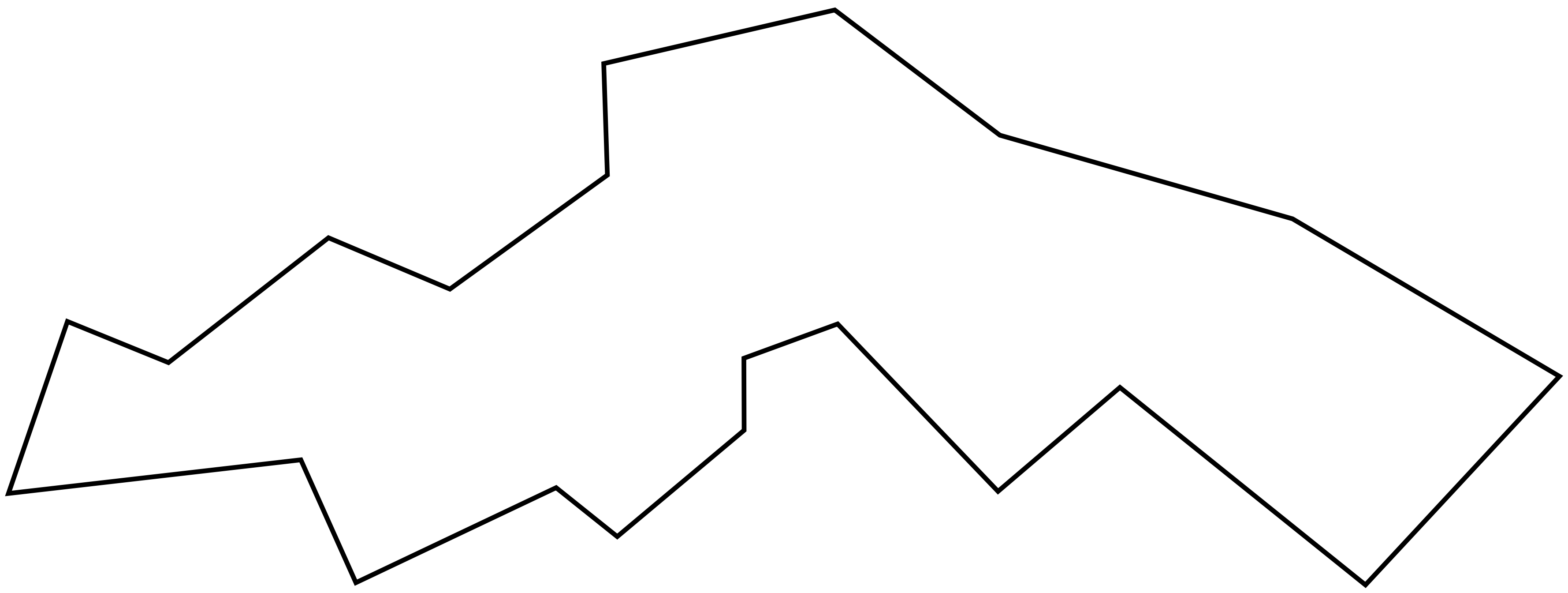


Analysis: O(n) time

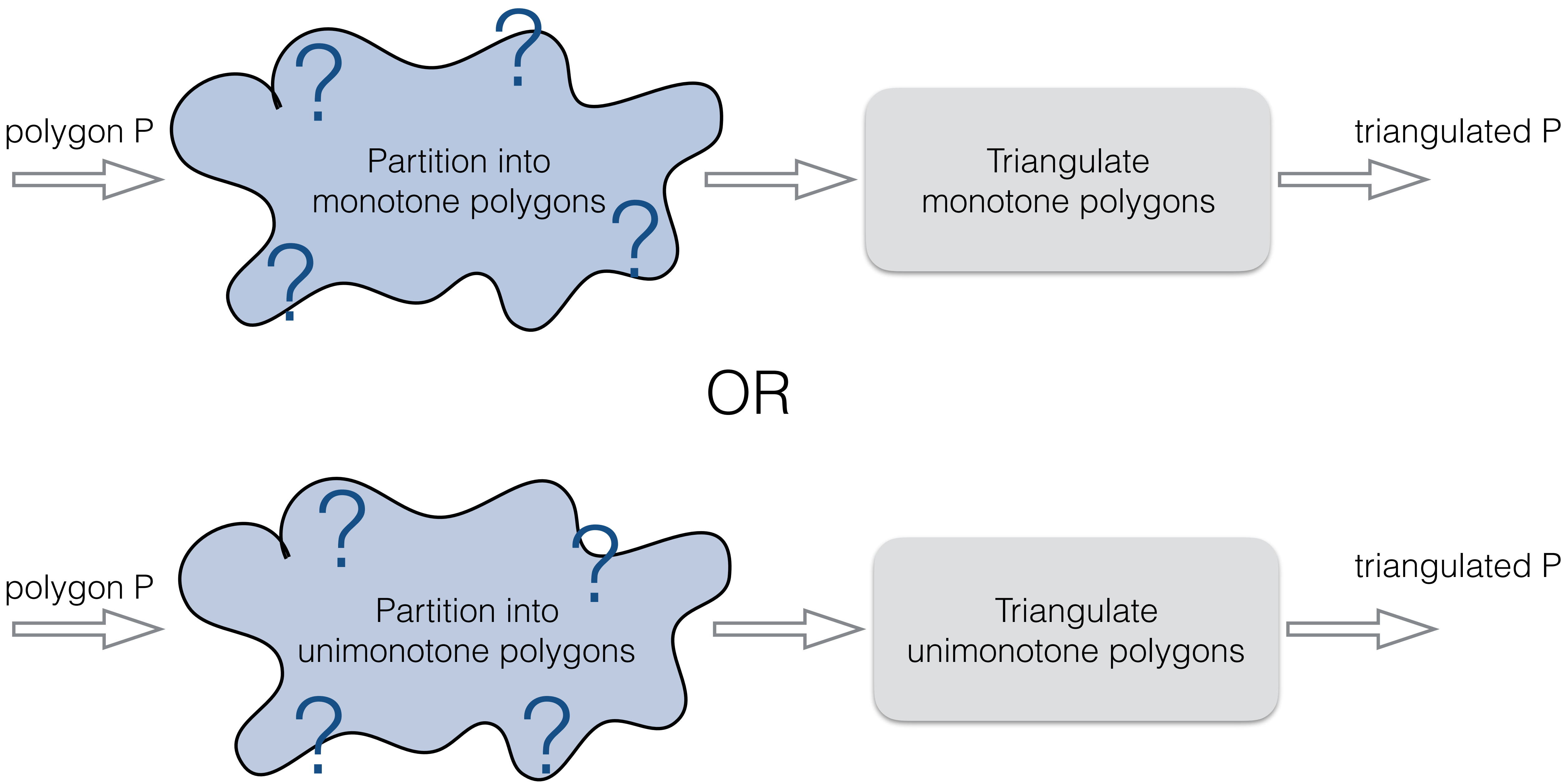
From monotone mountains to monotone polygons

Same idea: pick the next vertex in x-order. It can be on upper or lower chain.

O(n) time

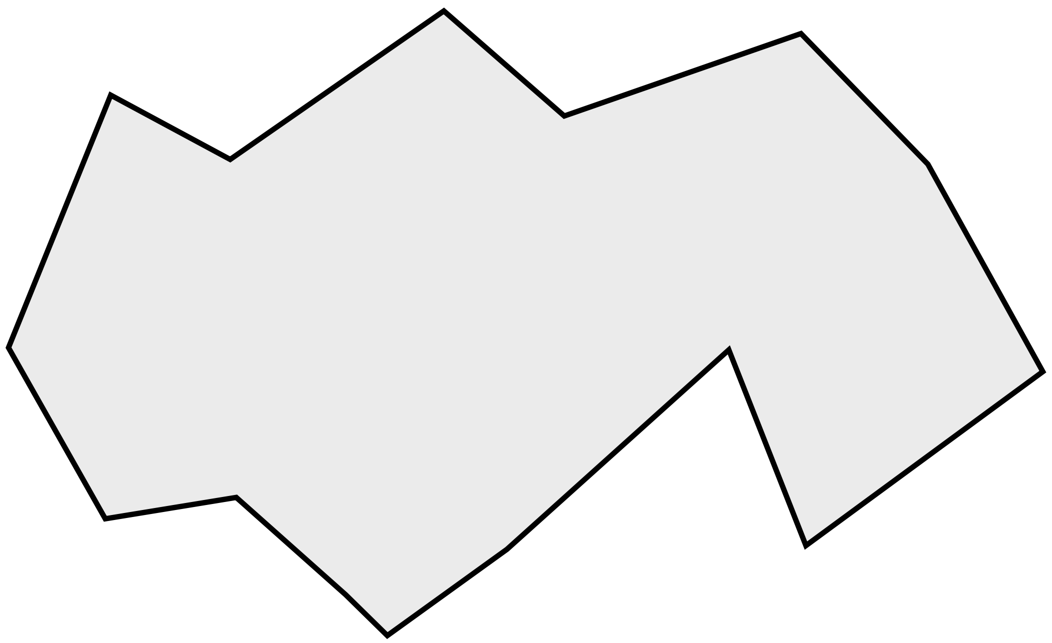


Towards an O(n lg n) Polygon Triangulation Algorithm

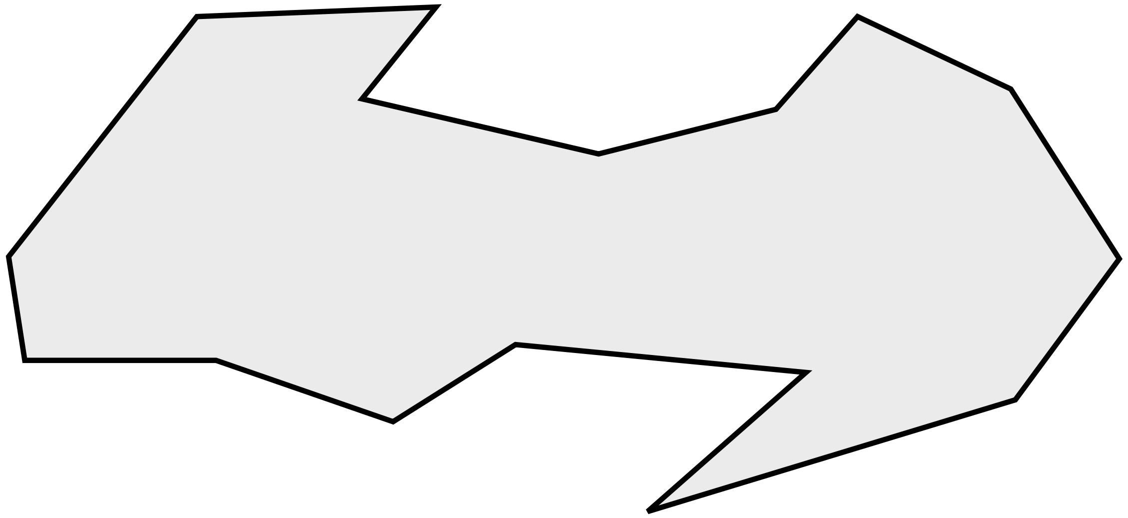


How can we partition a polygon into (uni)monotone pieces?

Intuition



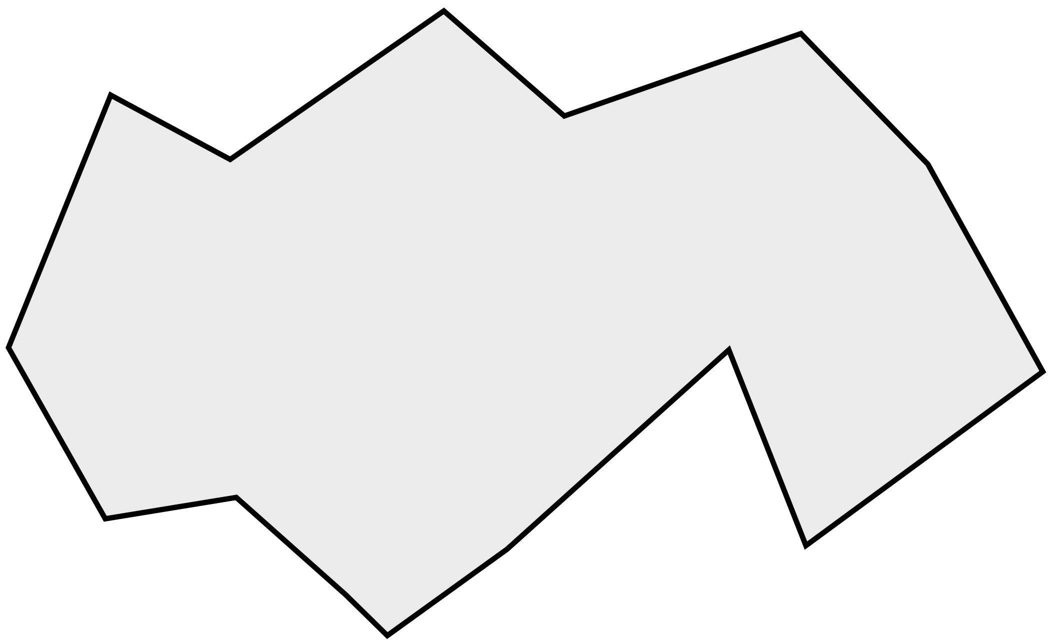
x-monotone



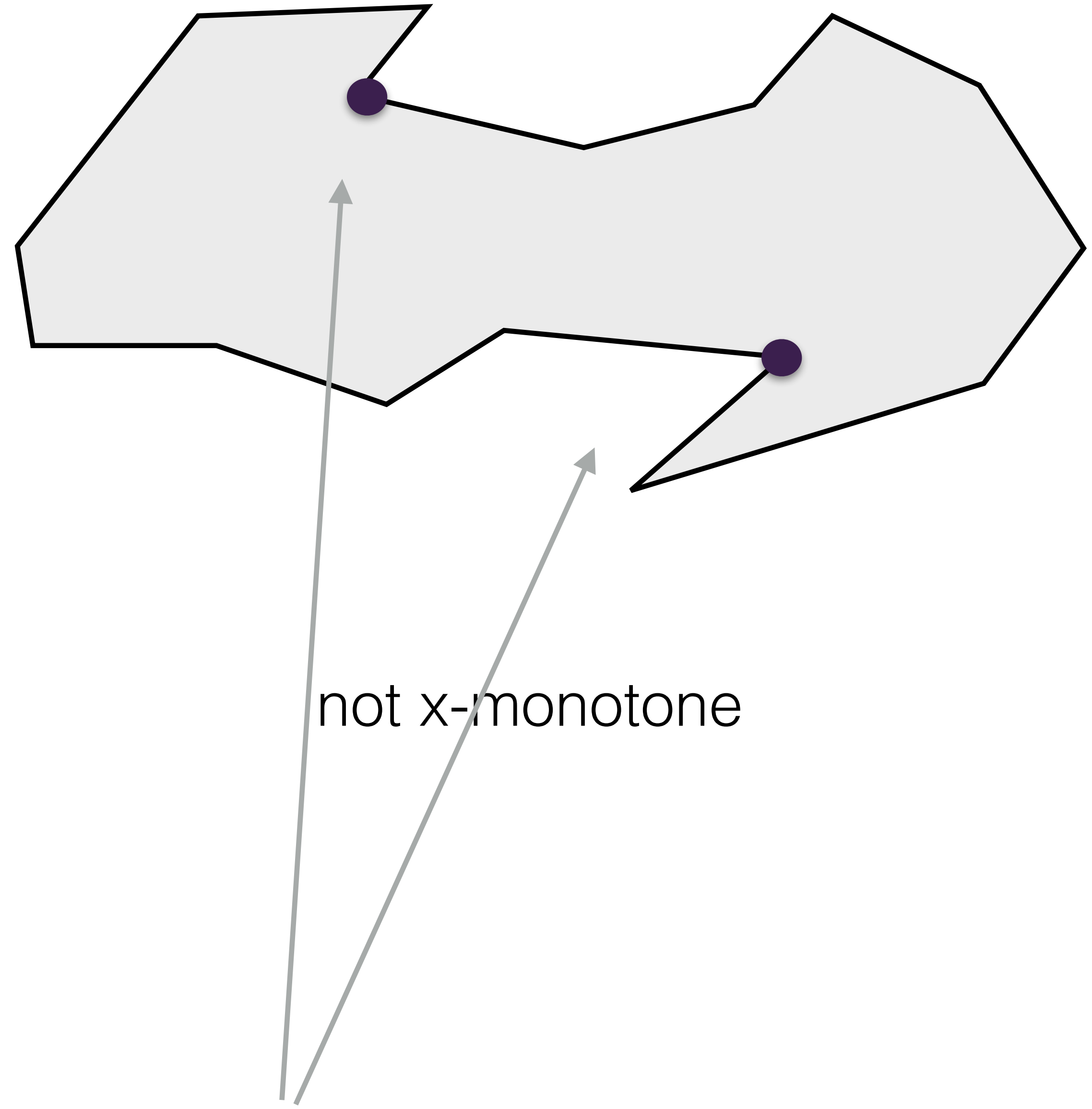
not x-monotone

What makes a polygon **not** monotone?

Intuition



x-monotone

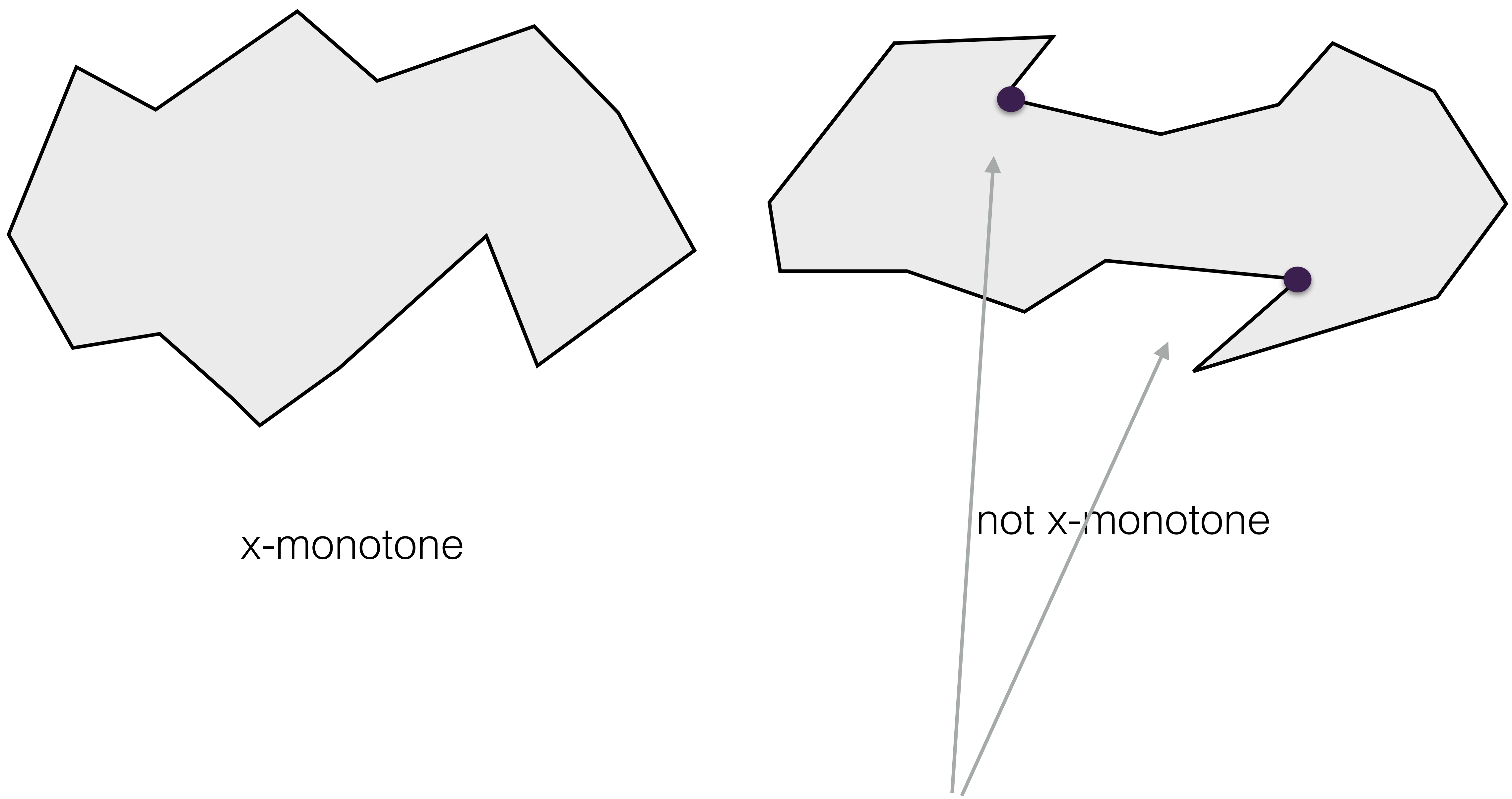


not x-monotone

What makes a polygon **not** monotone?

Intuition

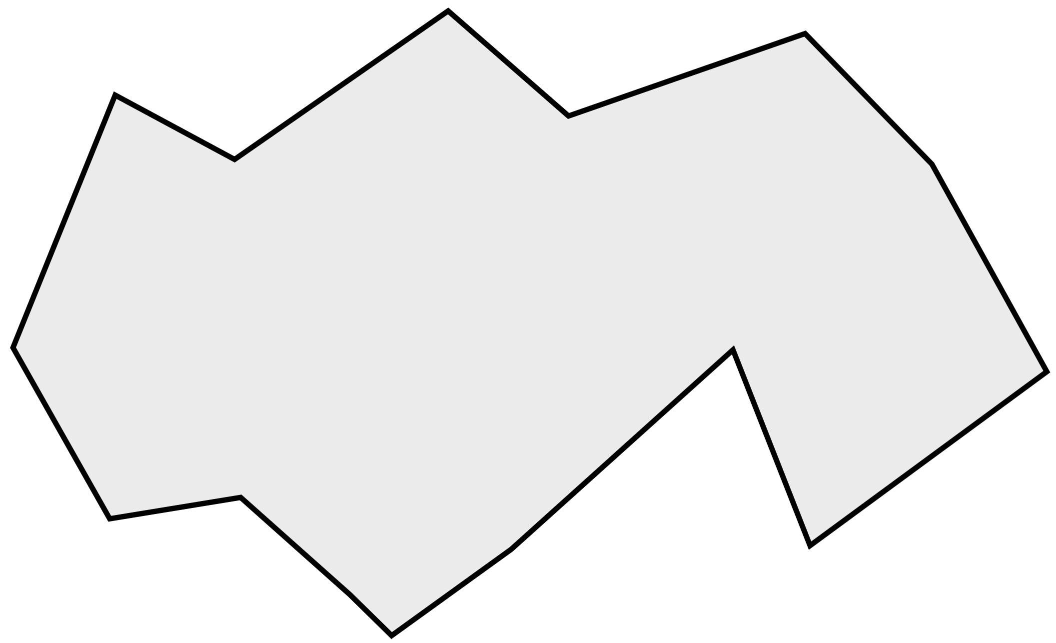
Cusp: a reflex vertex v such that the vertices before and after are both smaller or both larger than v (in terms of x-coords).



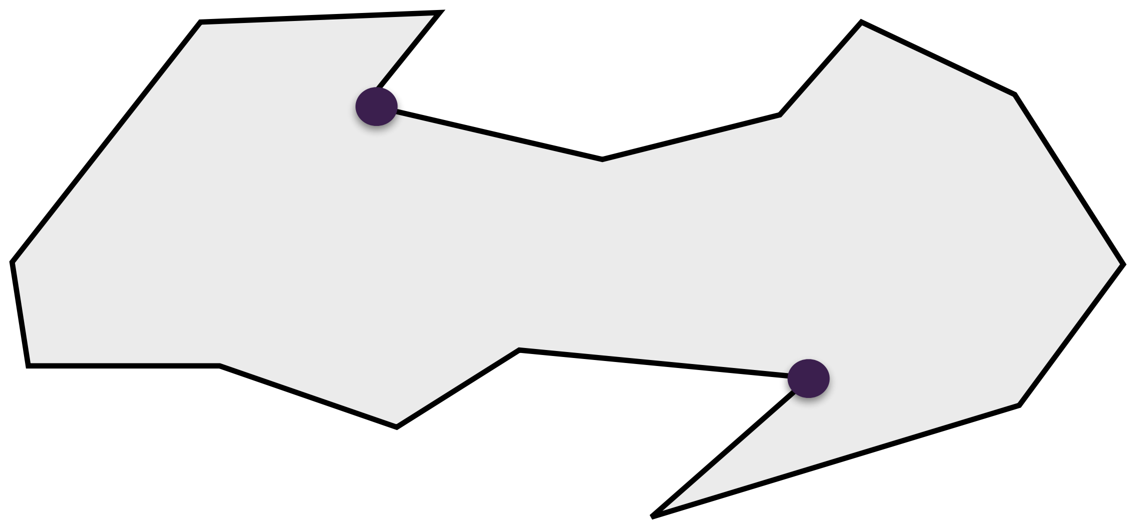
What makes a polygon **not** monotone?

Intuition

Cusp: a reflex vertex v such that the vertices before and after are both smaller or both larger than v (in terms of x-coords).



x-monotone



not x-monotone

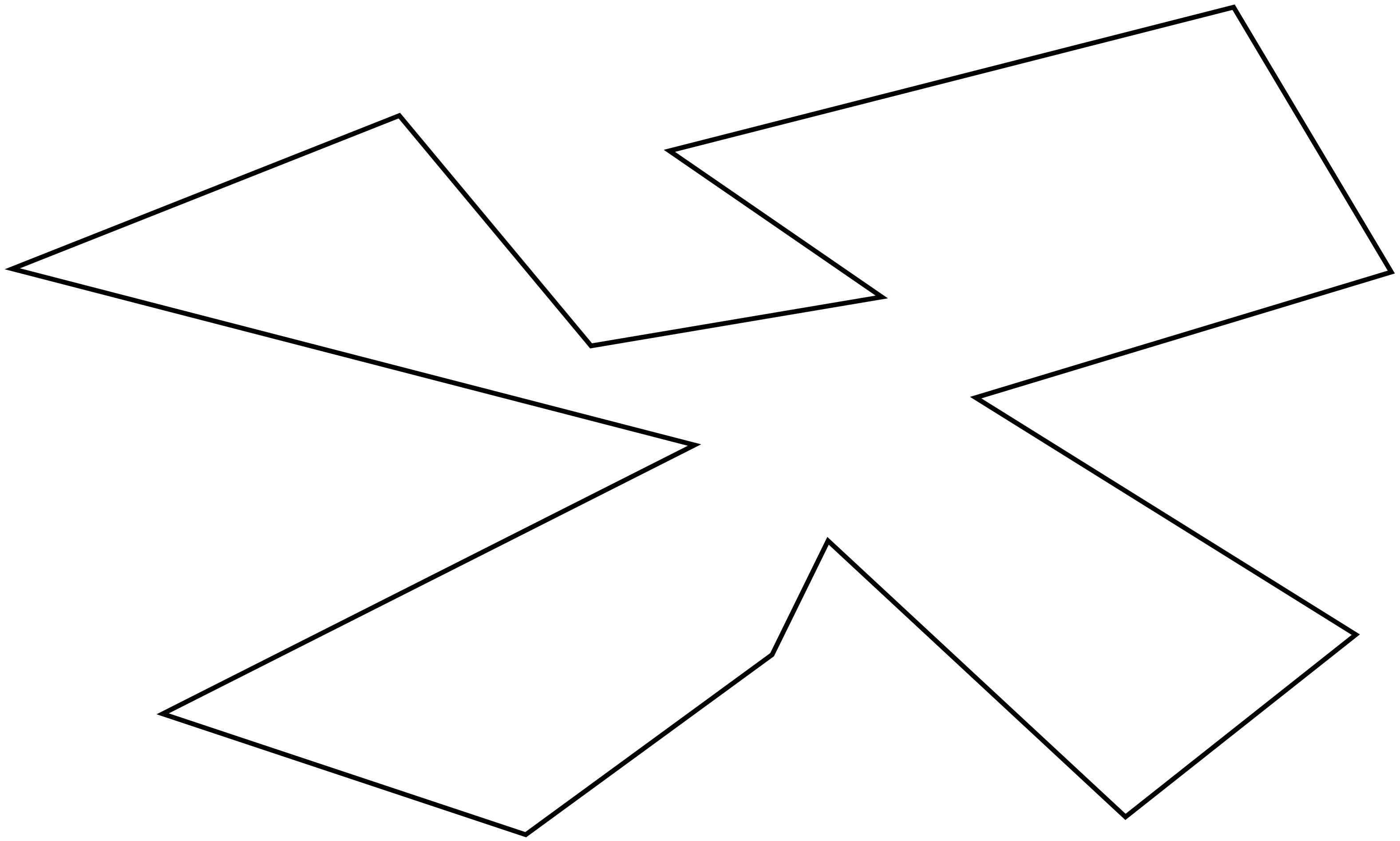
- Theorem: If a polygon has no cusps, then it's monotone.
- Proof: Intuitively clear, but proof a little tedious.

We’ll get rid of cusps using a trapezoidation of P.

Trapezoid partitions

Shoot vertical rays

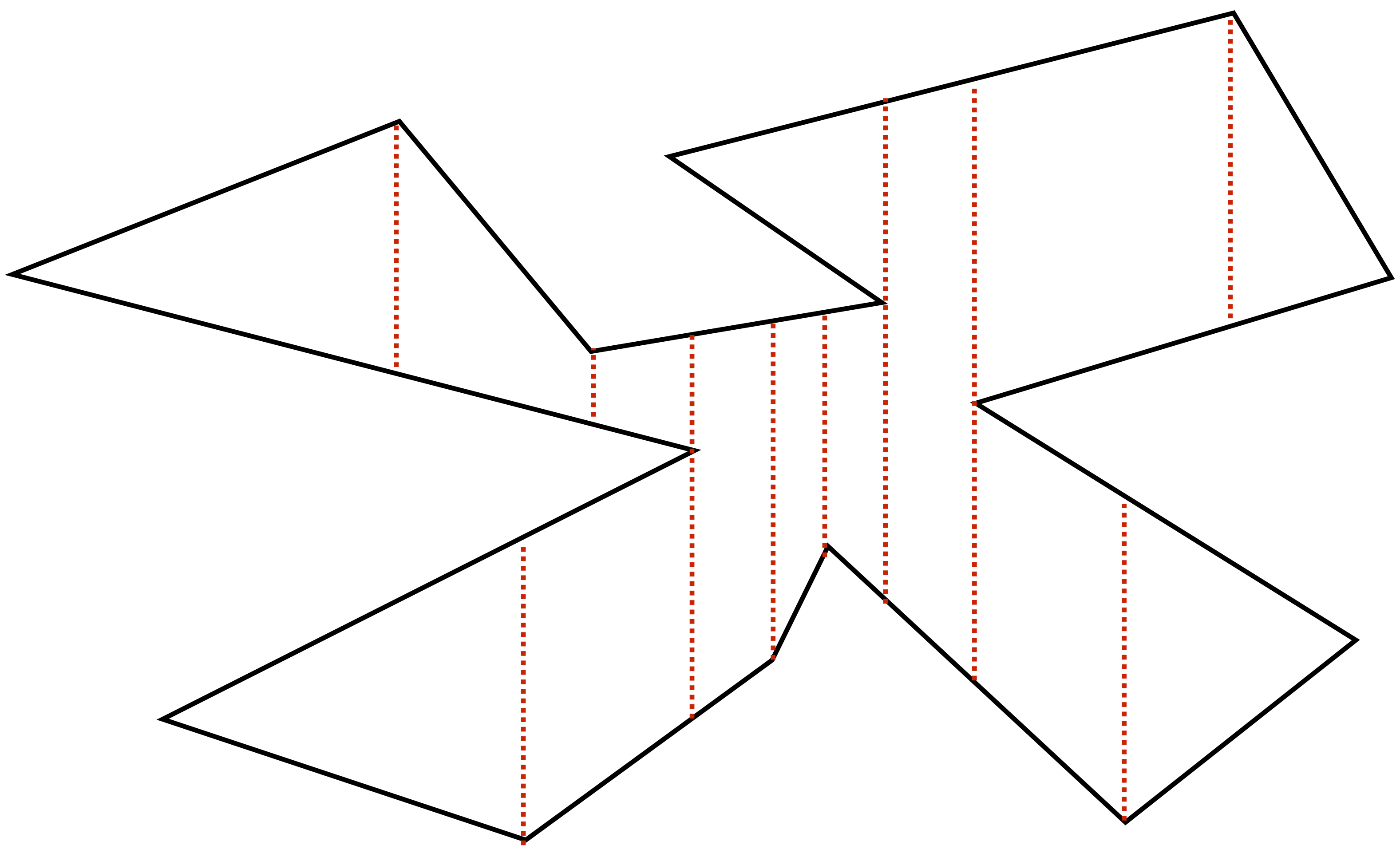
- If polygon is above vertex, shoot vertical ray up until reaches boundary
- If f polygon is below vertex, shoot down
- If polygon is above and below vertex, shoot both up and down



Trapezoid partitions

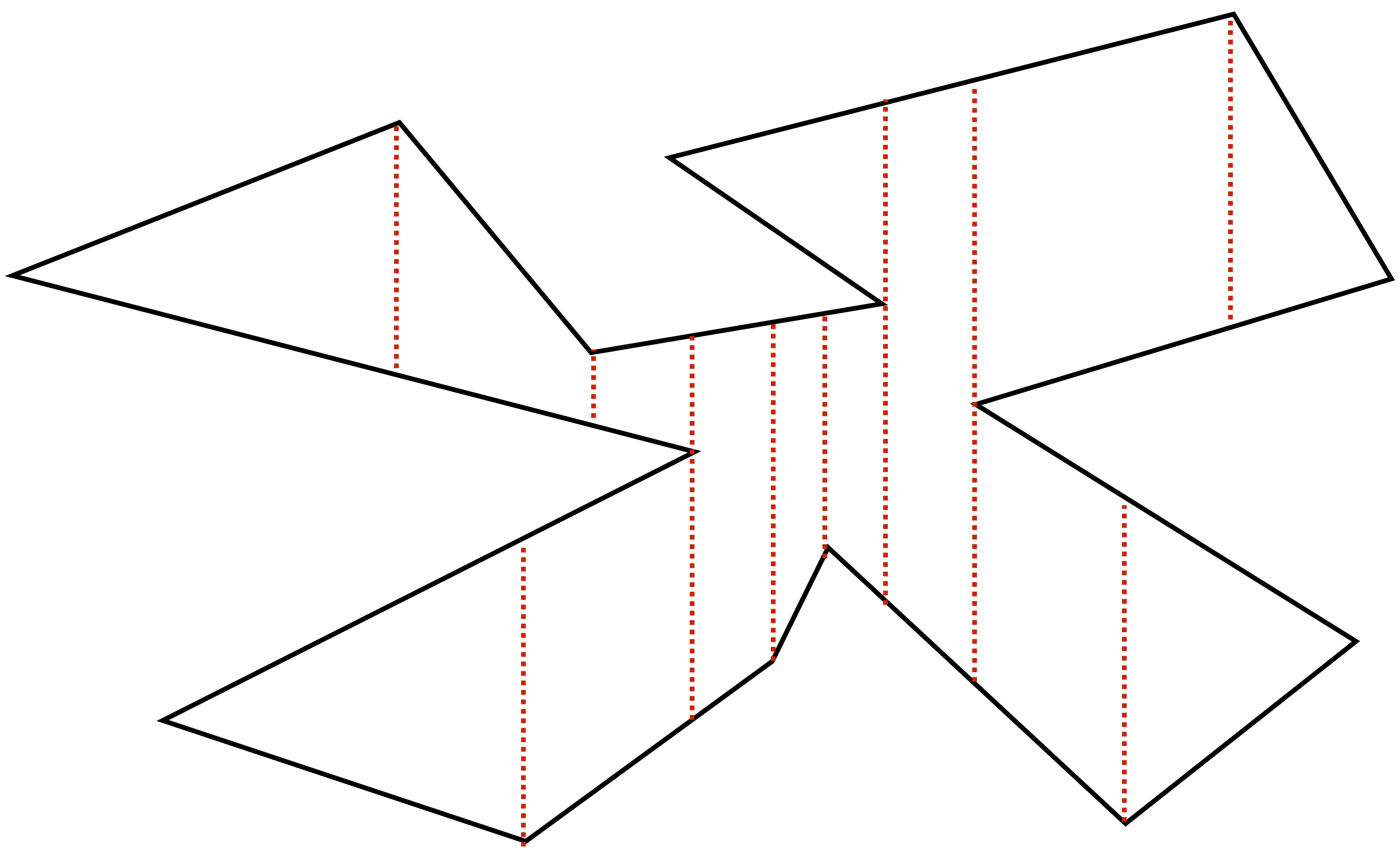
Shoot vertical rays

- If polygon is above vertex, shoot vertical ray up until reaches boundary
- If f polygon is below vertex, shoot down
- If polygon is above and below vertex, shoot both up and down



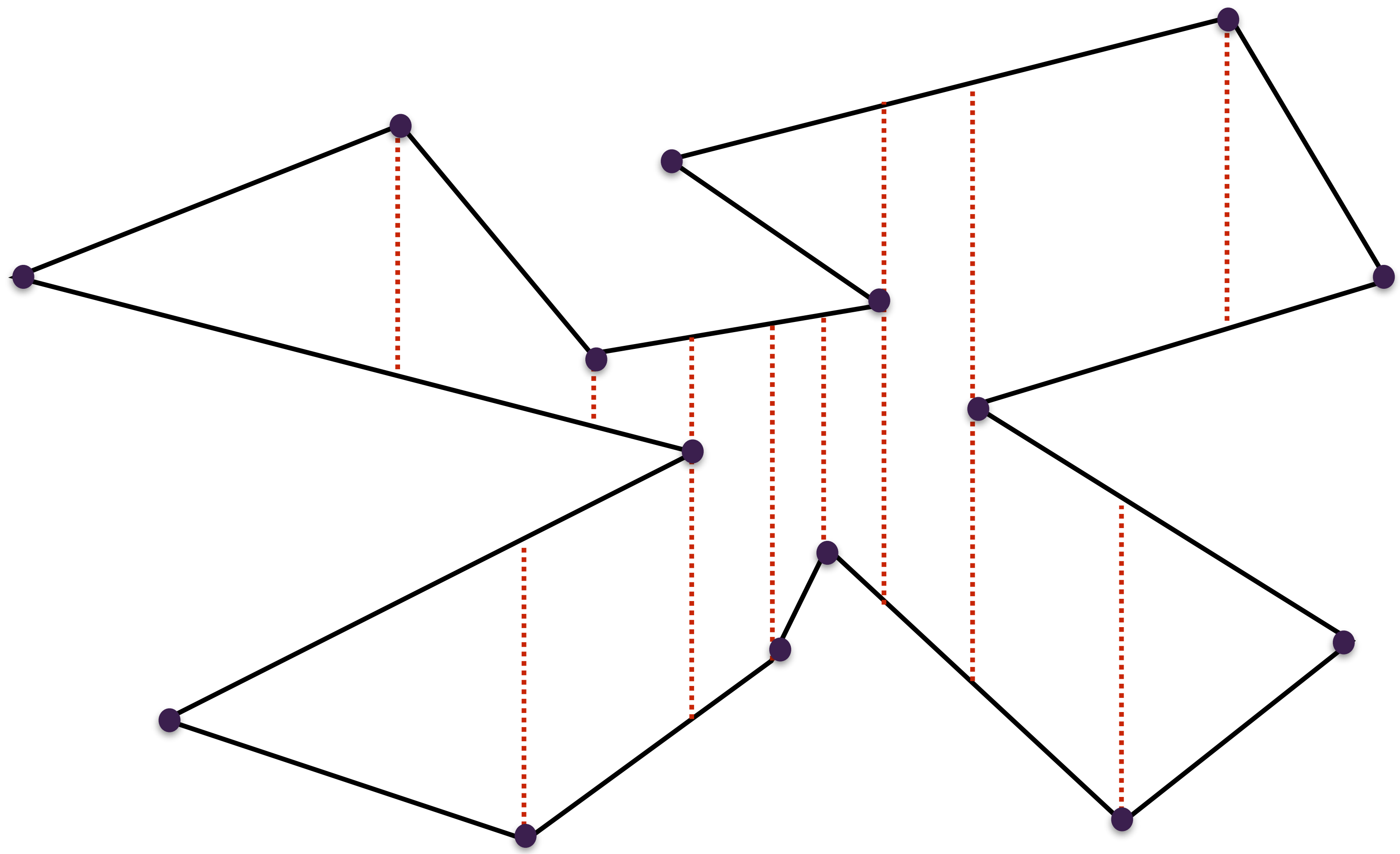
Trapezoid partitions: Properties

- Each polygon in the partition is a trapezoid, because:
  - It has one or two rays as sides.
  - If it has two, they must both hit the same edge above, and the same edge below.
- How many ?
  - At most one ray through each vertex => O(n) threads => O(n) trapezoids



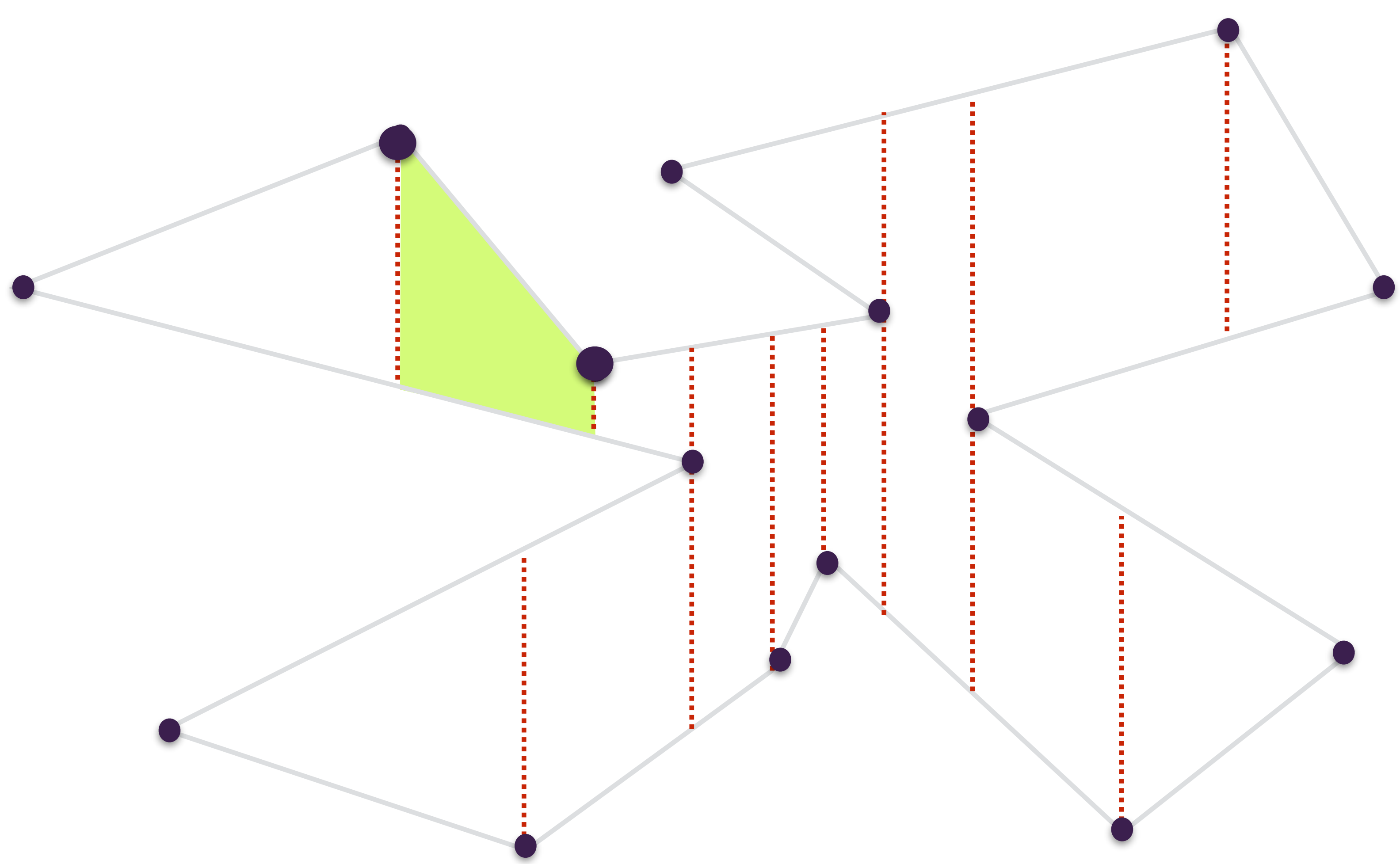
Trapezoid partitions

- Each trapezoid has precisely two vertices of the polygon, one on the left and one on the right. They can be on the top, bottom or middle of the trapezoid.



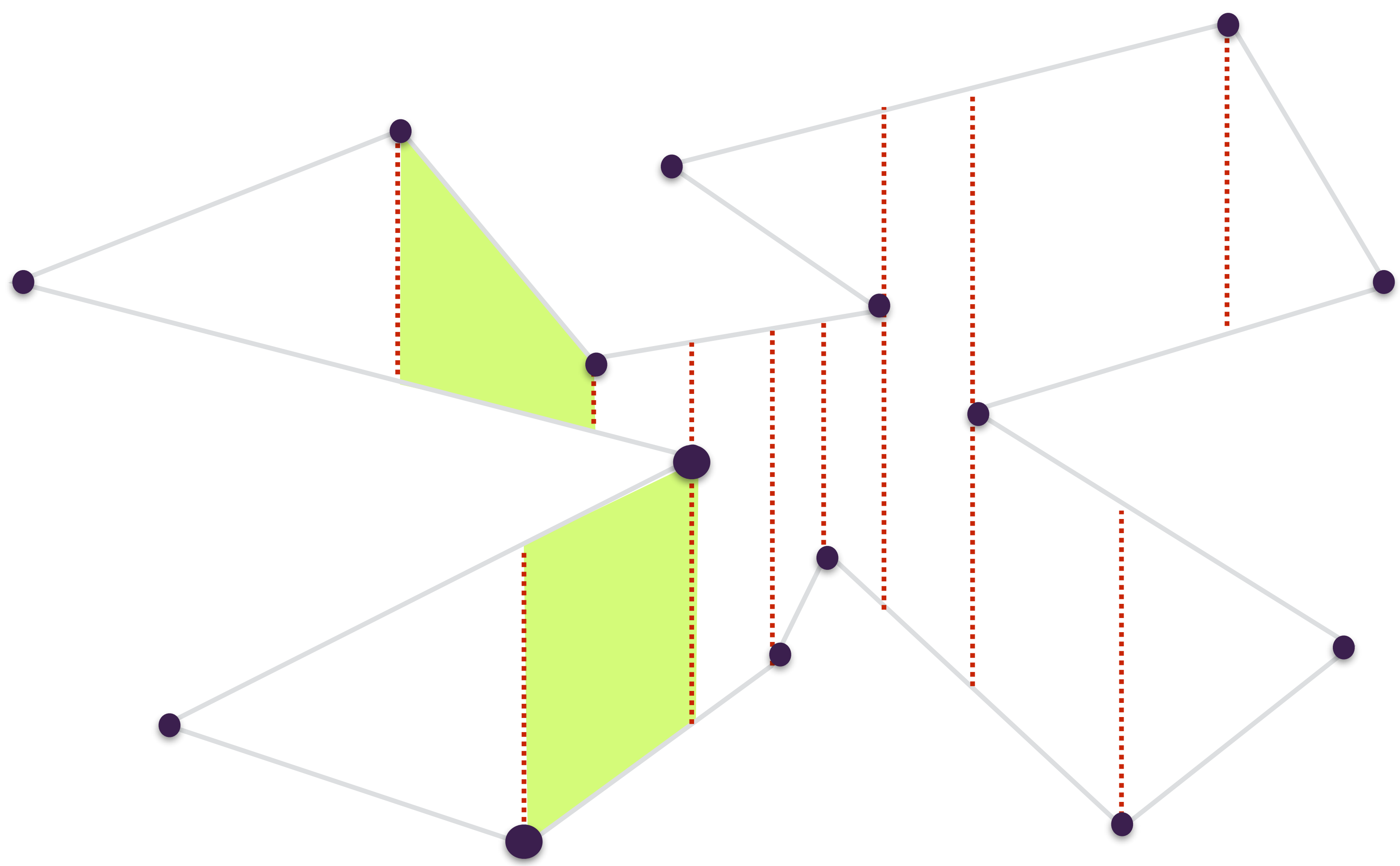
Trapezoid partitions

- Each trapezoid has precisely two vertices of the polygon, one on the left and one on the right. They can be on the top, bottom or middle of the trapezoid.



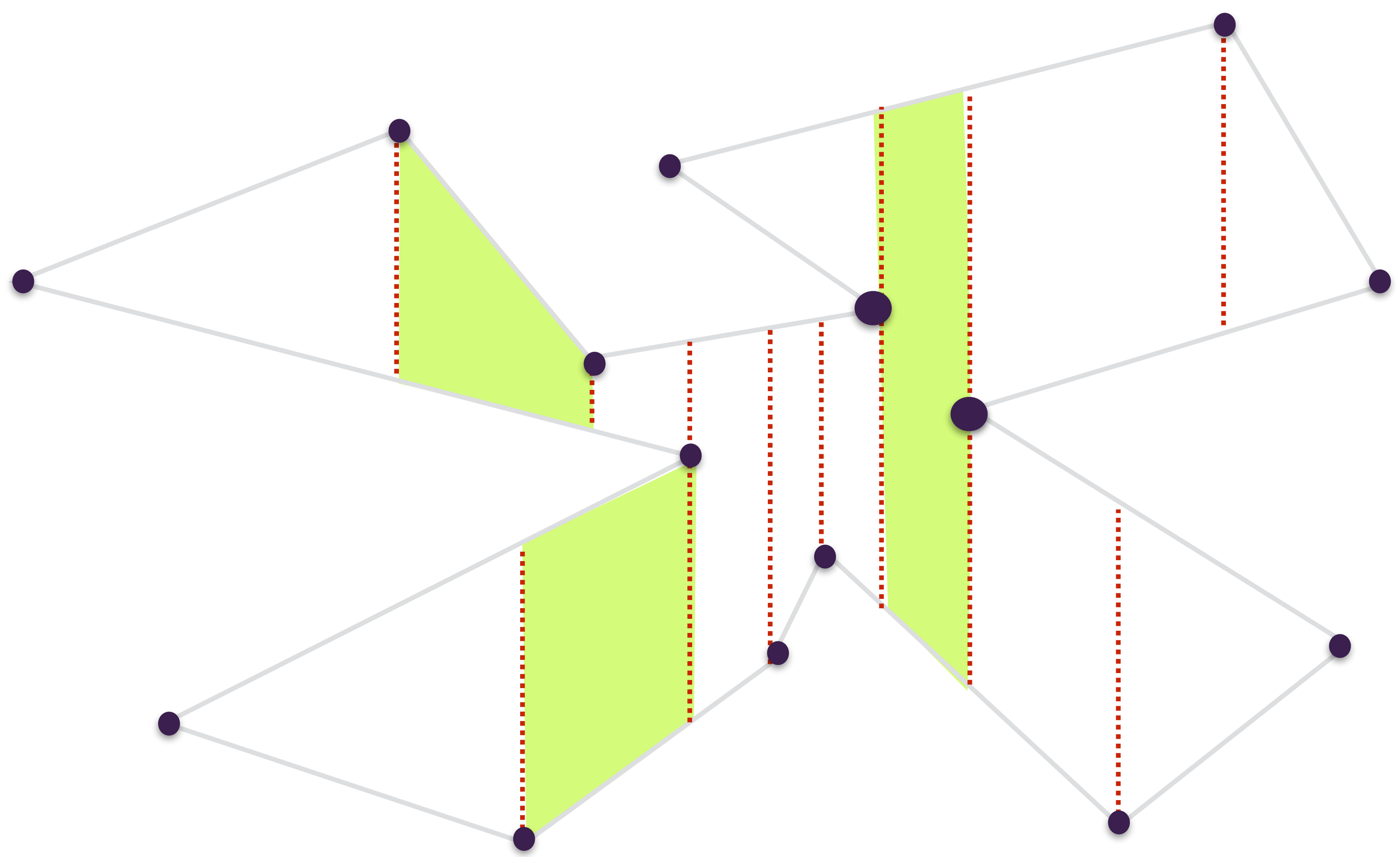
Trapezoid partitions

- Each trapezoid has precisely two vertices of the polygon, one on the left and one on the right. They can be on the top, bottom or middle of the trapezoid.



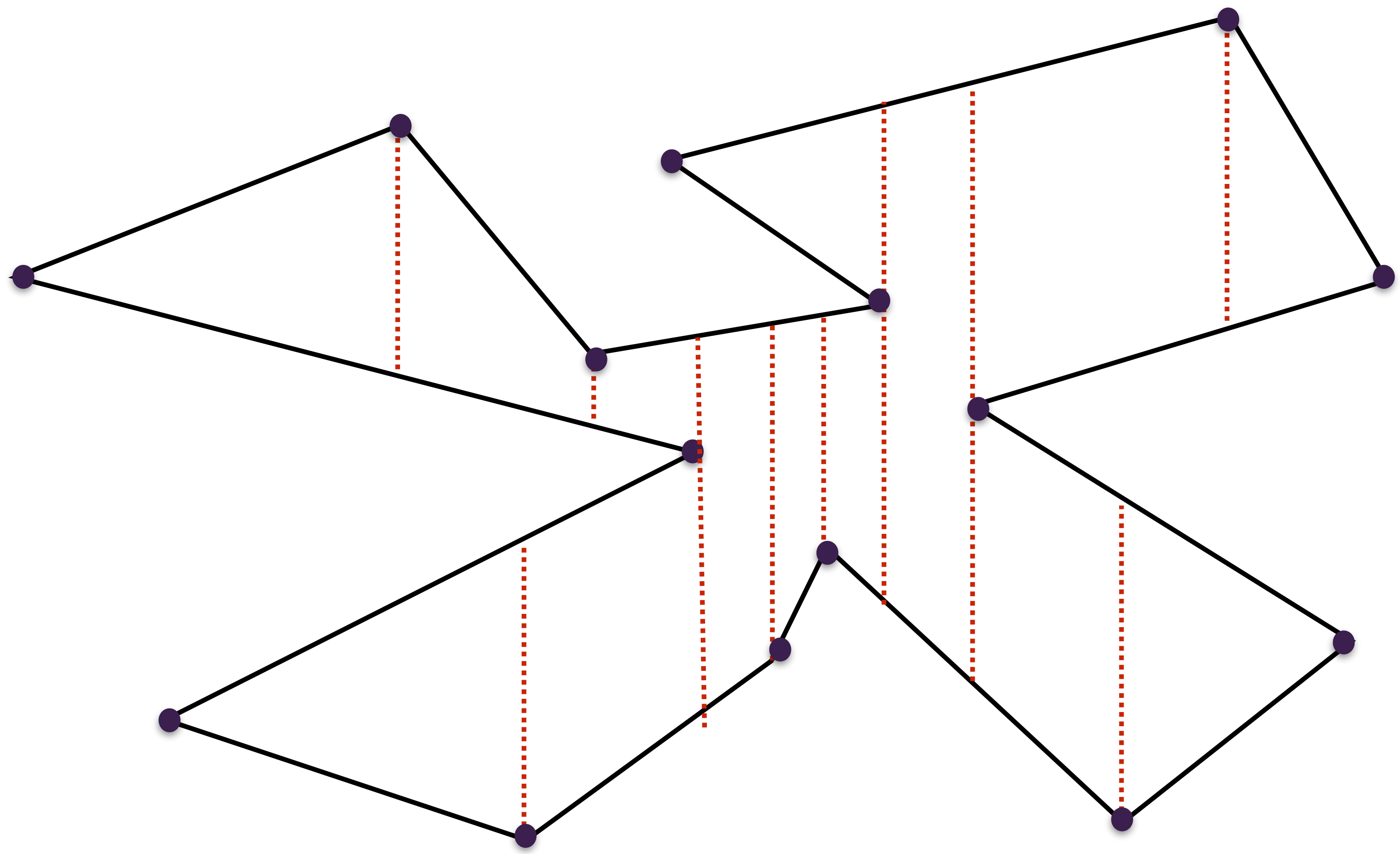
Trapezoid partitions

- Each trapezoid has precisely two vertices of the polygon, one on the left and one on the right. They can be on the top, bottom or middle of the trapezoid.



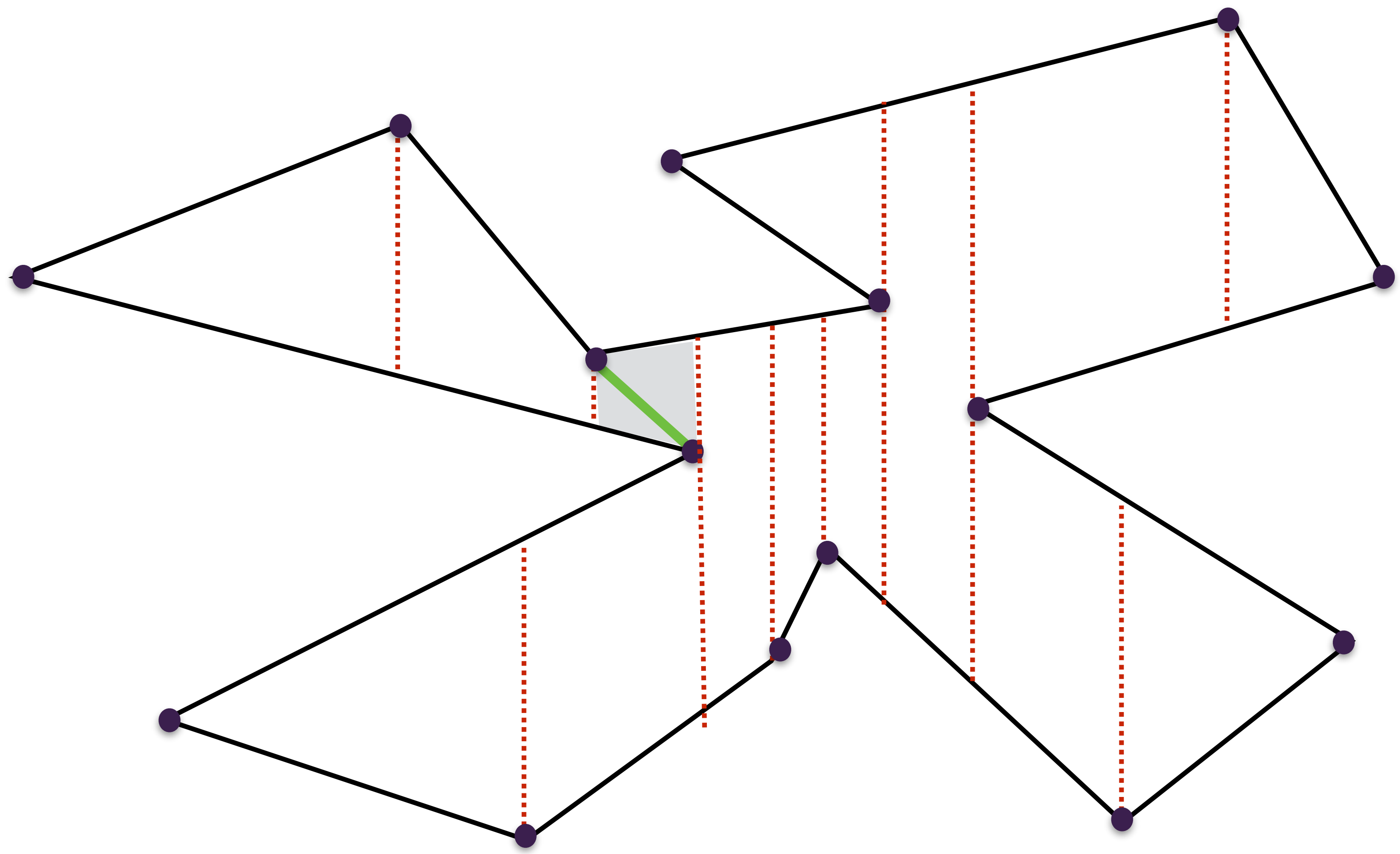
Diagonals

- In each trapezoid: if its two vertices are not on the same edge, they define a **diagonal**.



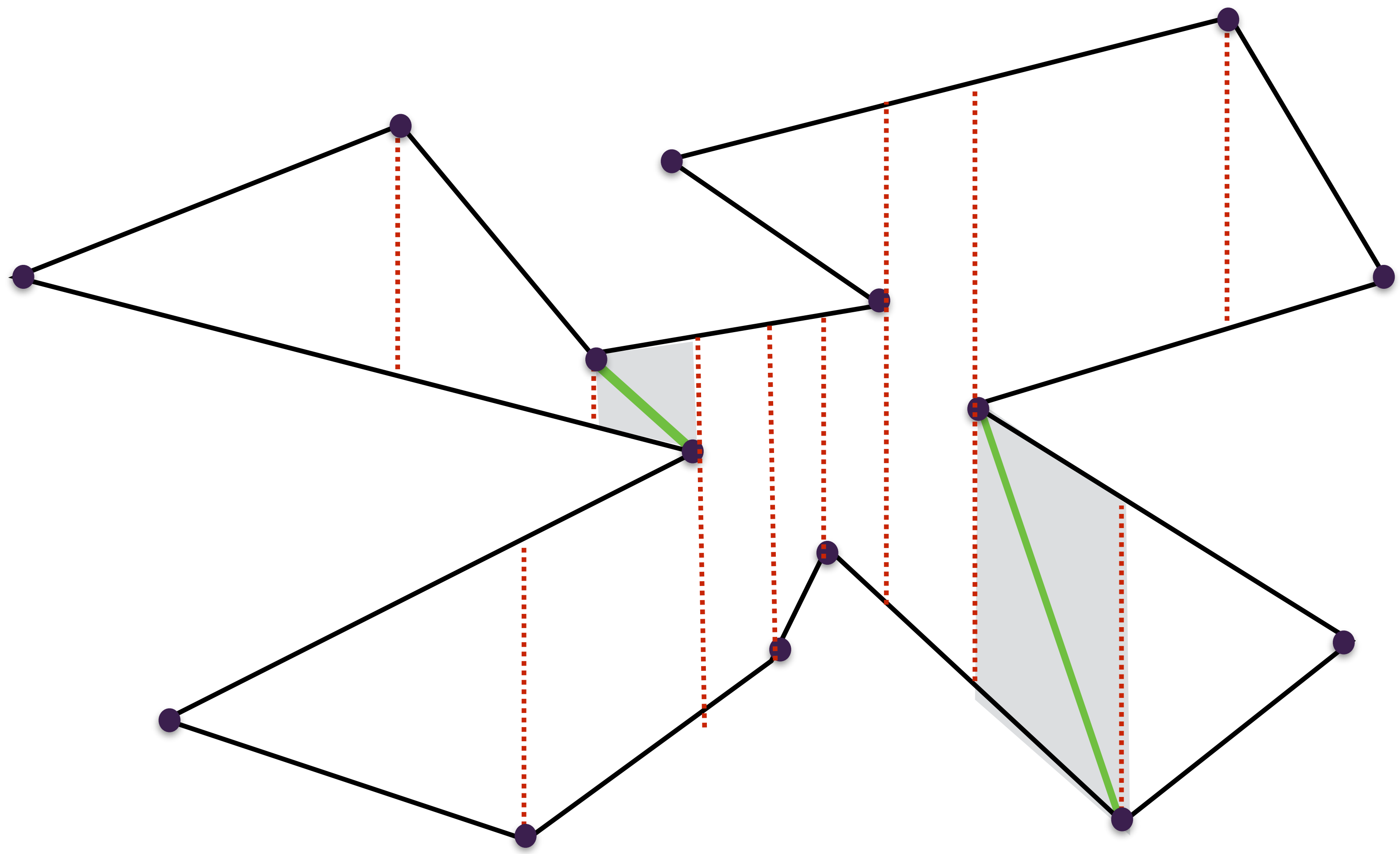
Diagonals

- In each trapezoid: if its two vertices are not on the same edge, they define a **diagonal**.



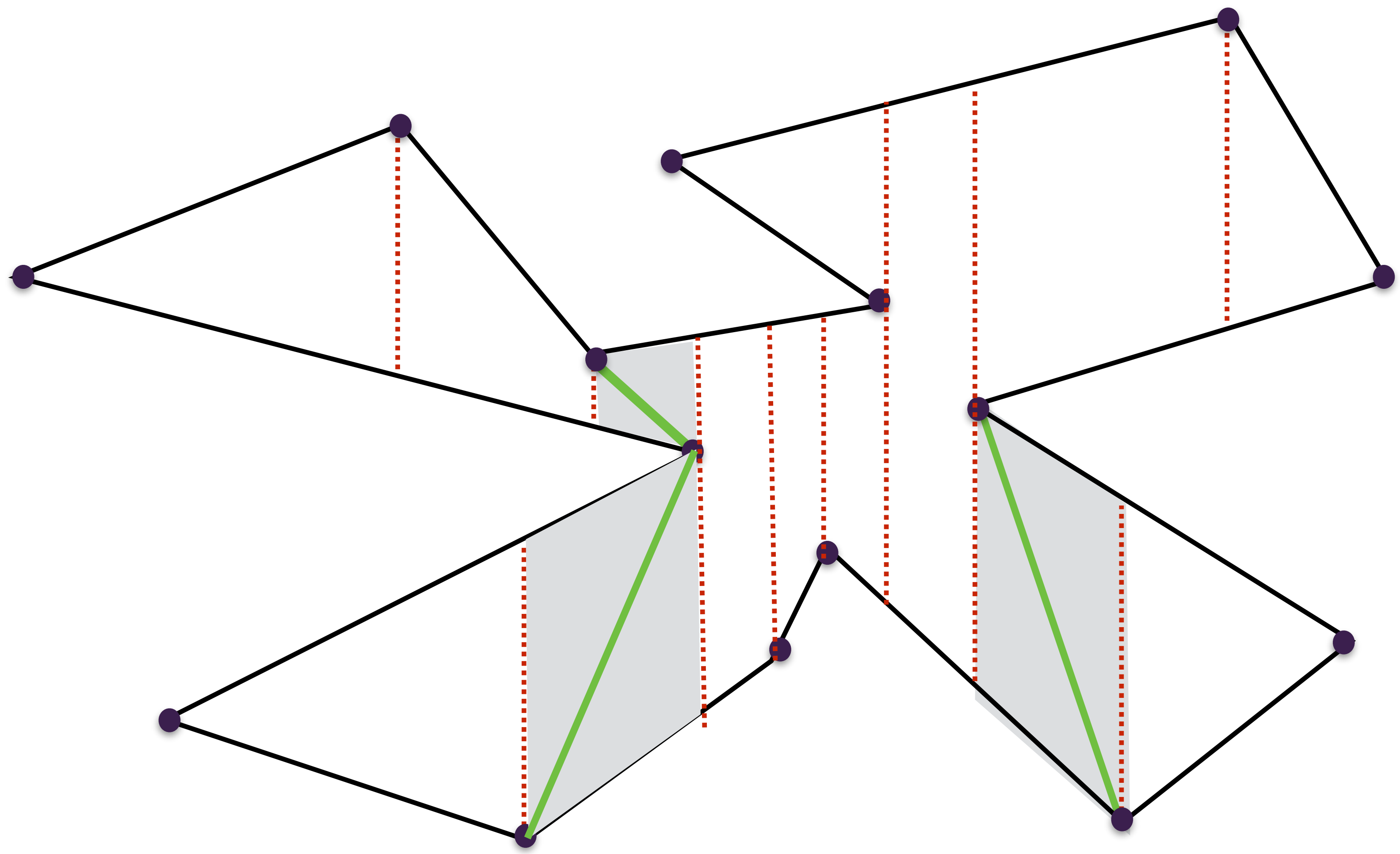
Diagonals

- In each trapezoid: if its two vertices are not on the same edge, they define a **diagonal**.



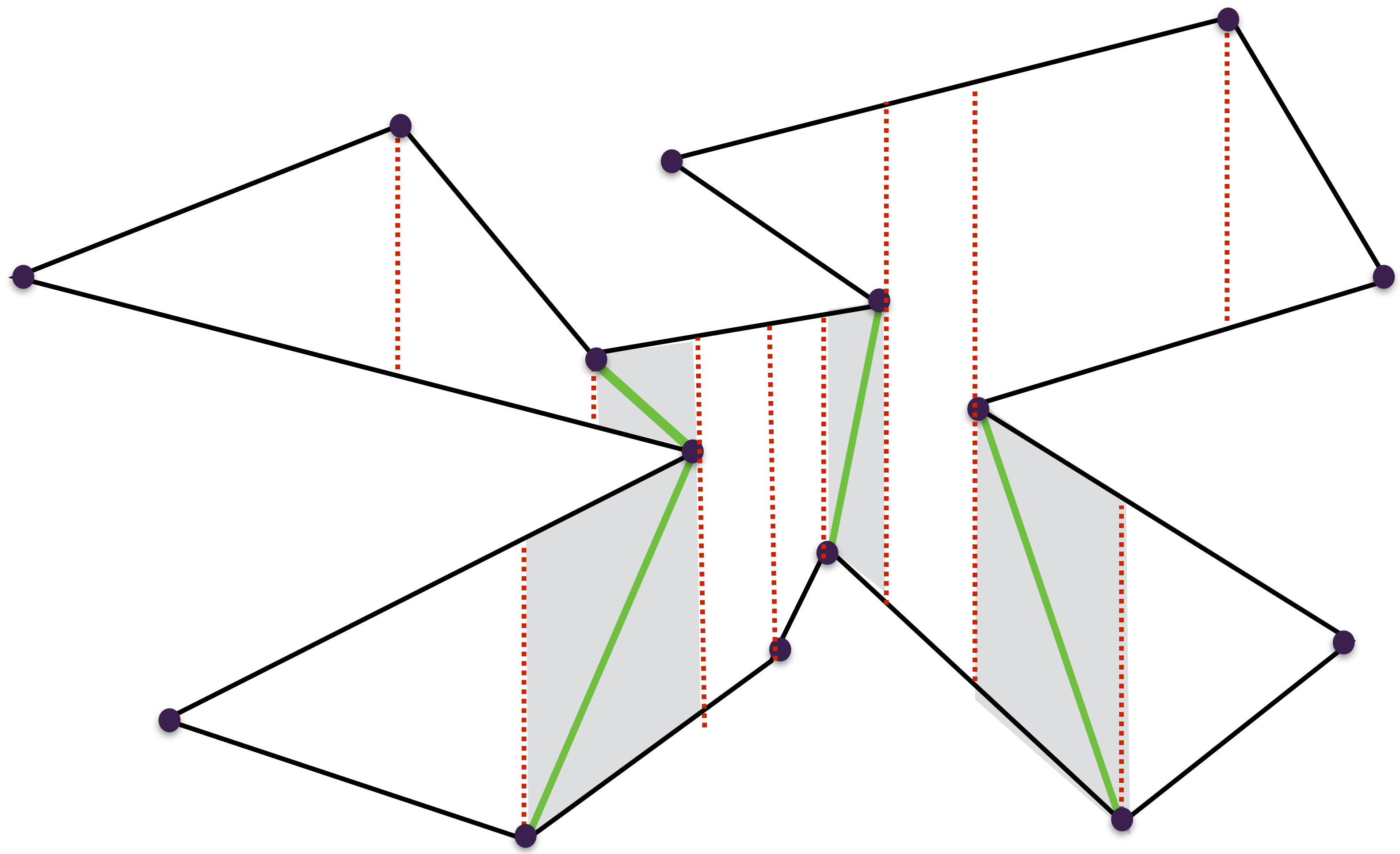
Diagonals

- In each trapezoid: if its two vertices are not on the same edge, they define a **diagonal**.



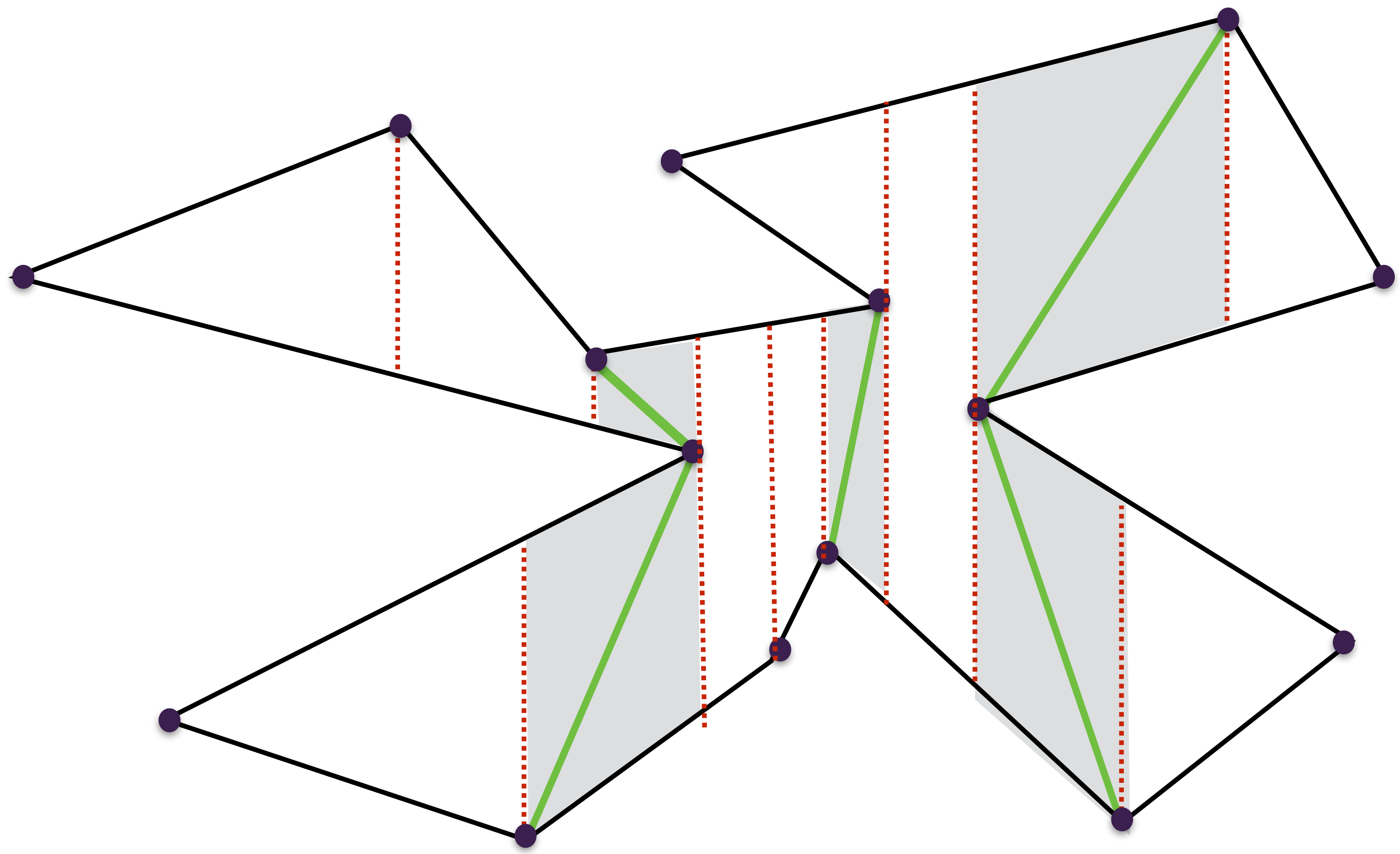
Diagonals

- In each trapezoid: if its two vertices are not on the same edge, they define a **diagonal**.



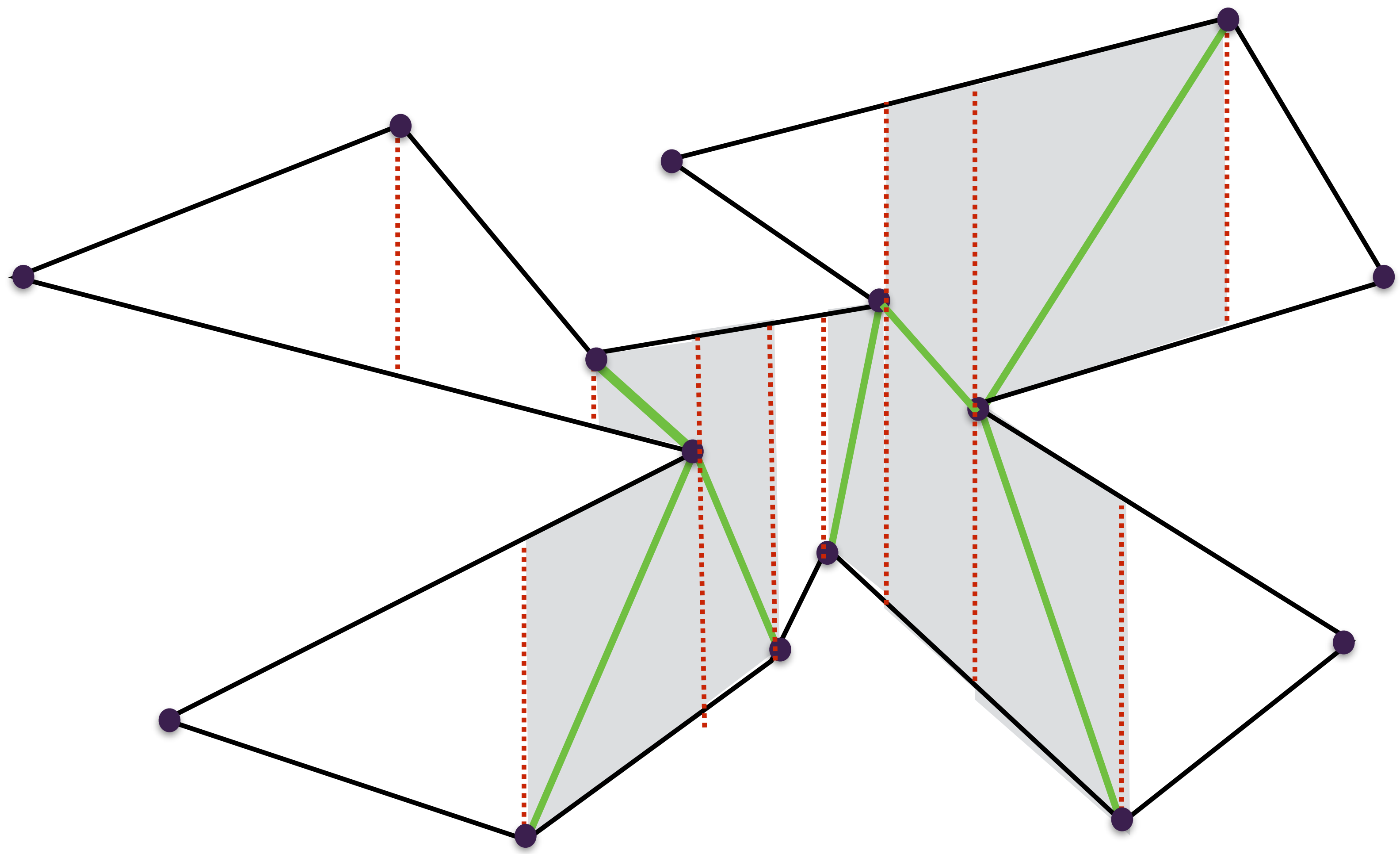
Diagonals

- In each trapezoid: if its two vertices are not on the same edge, they define a **diagonal**.

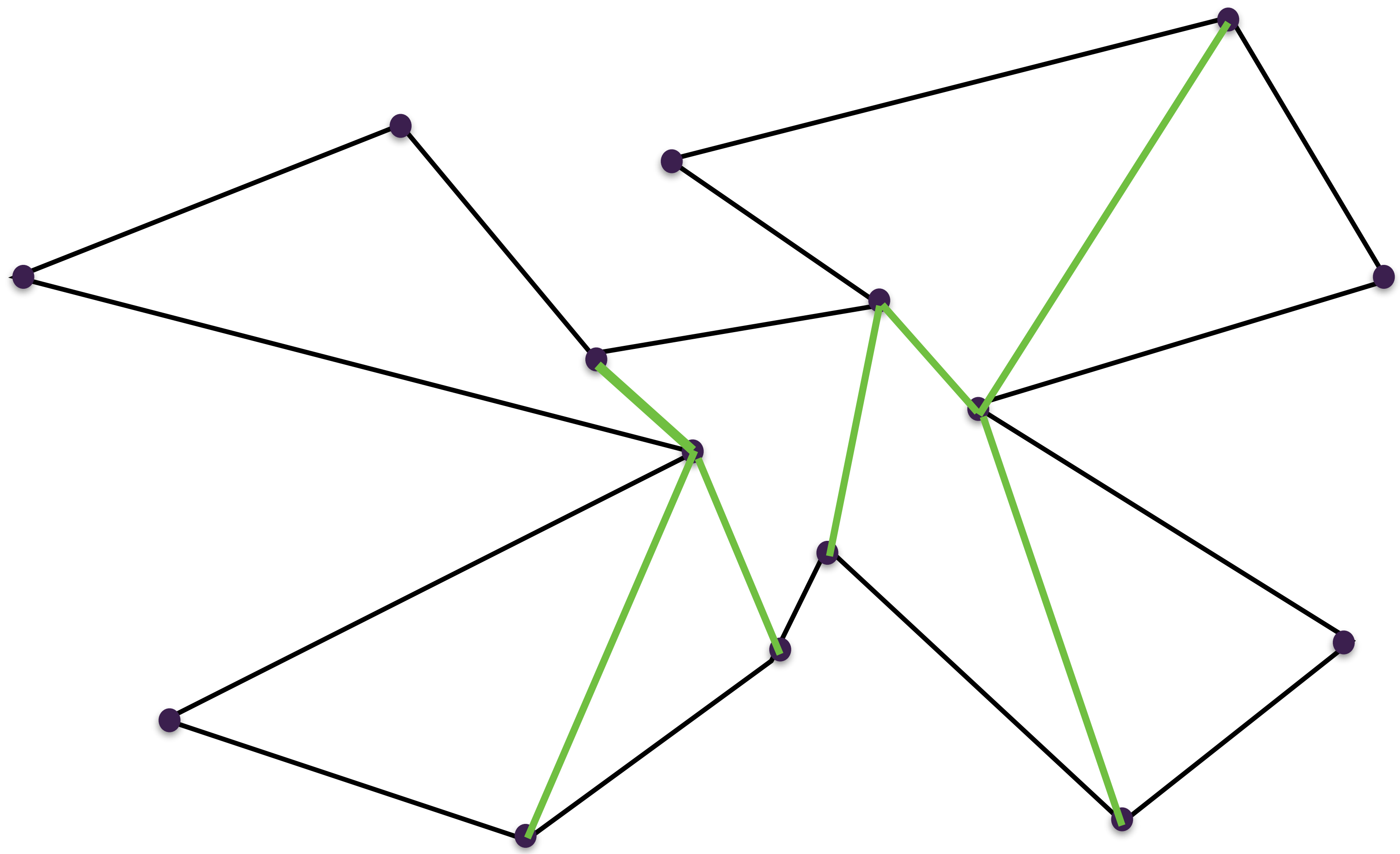


Diagonals

- In each trapezoid: if its two vertices are not on the same edge, they define a **diagonal**.

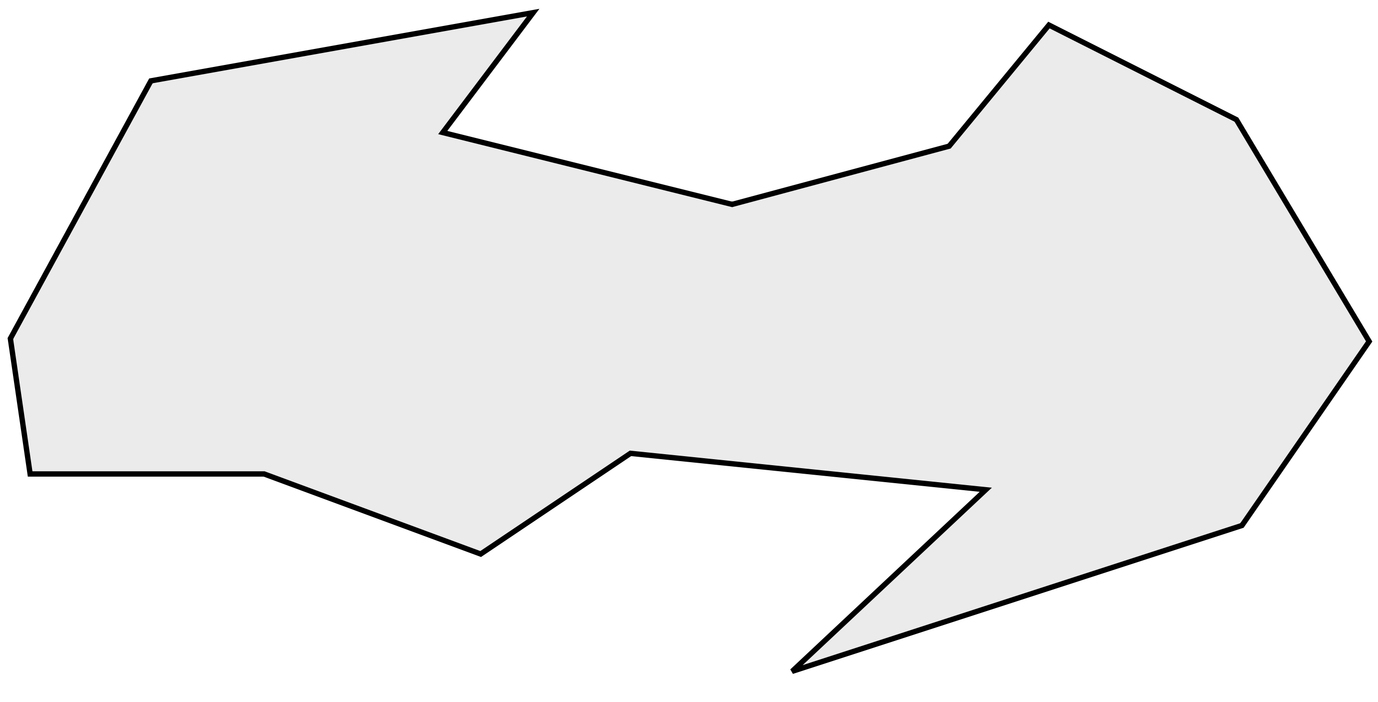


Diagonals in the trapezoid partition of P

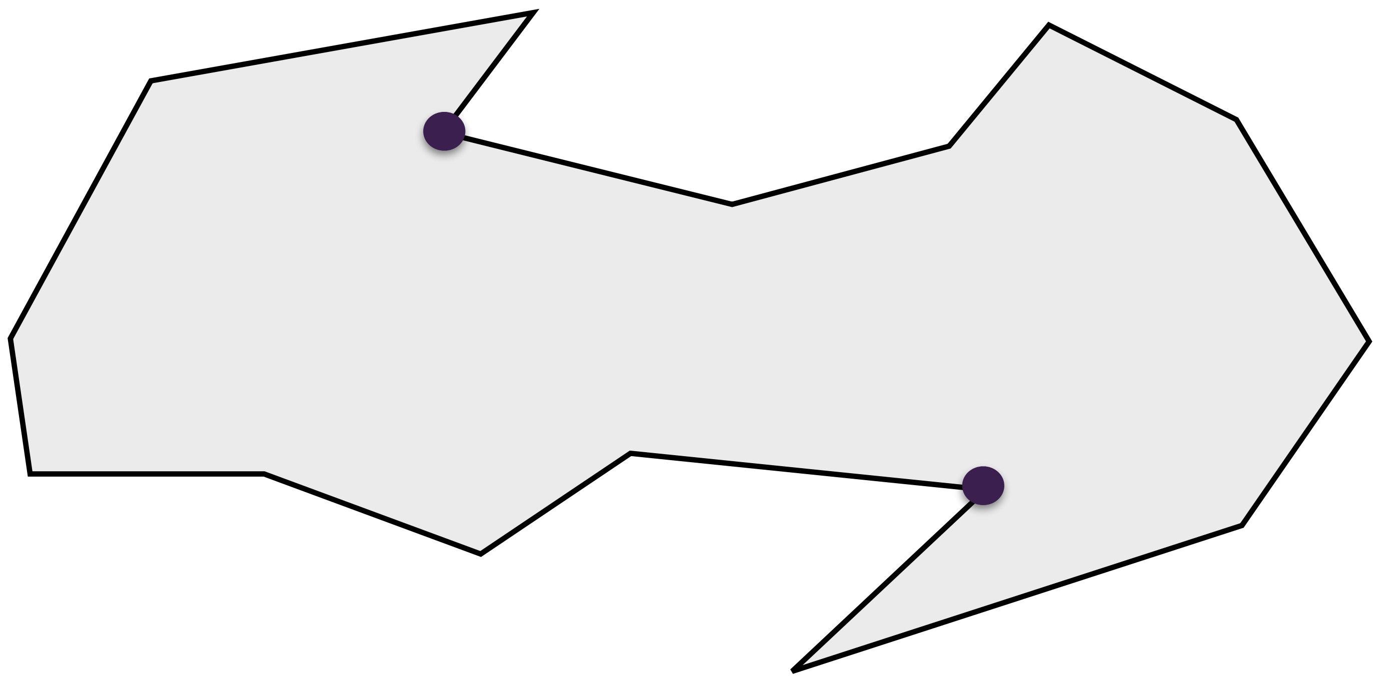


We can use the trapezoid partition of P to “split” the cusps

Removing cusps

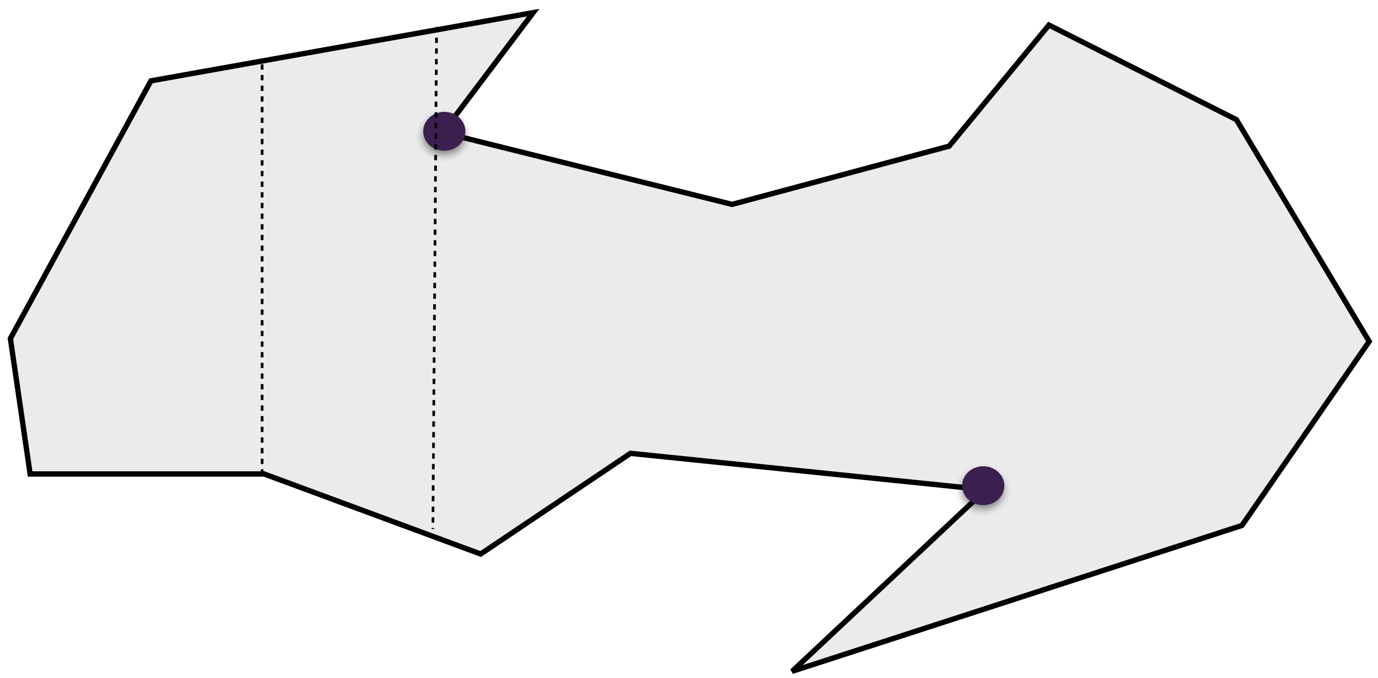


Removing cusps



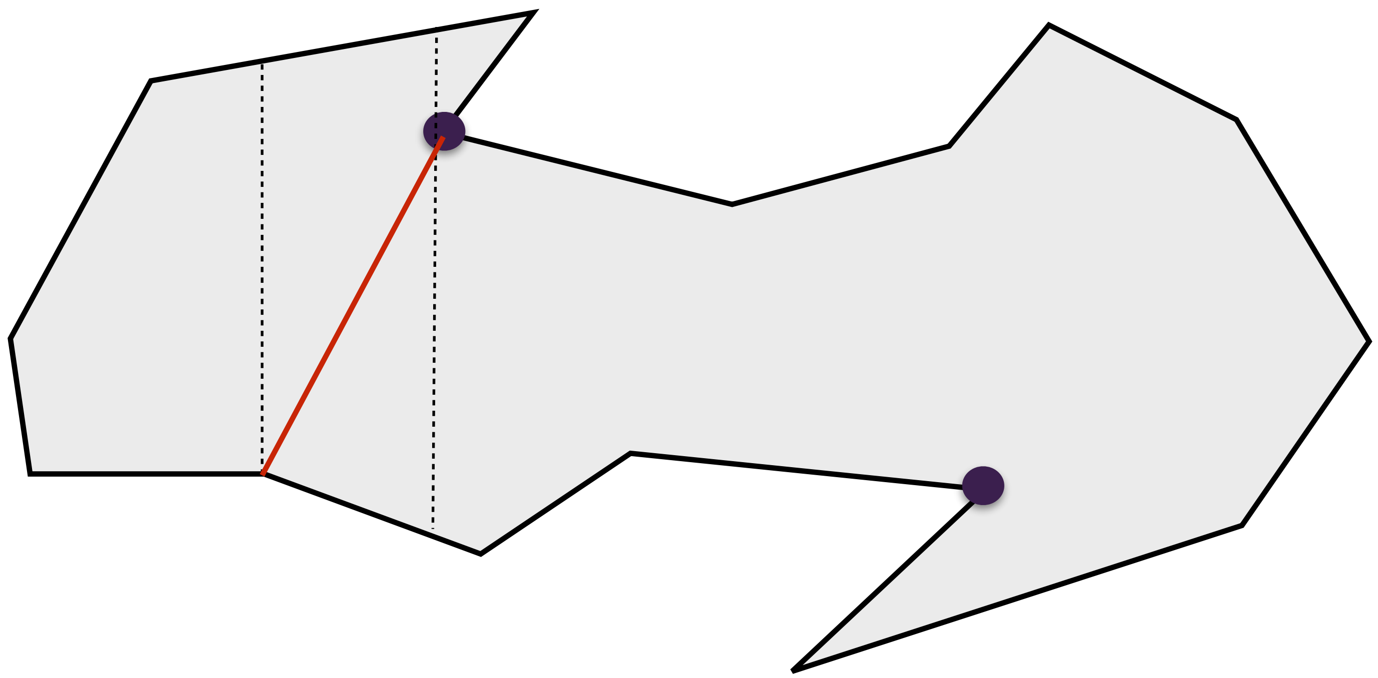
1. Identify cusp vertices
2. Compute a trapezoid partition of P
3. For each cusp vertex, add diagonal in trapezoid before/after the cusp

Removing cusps



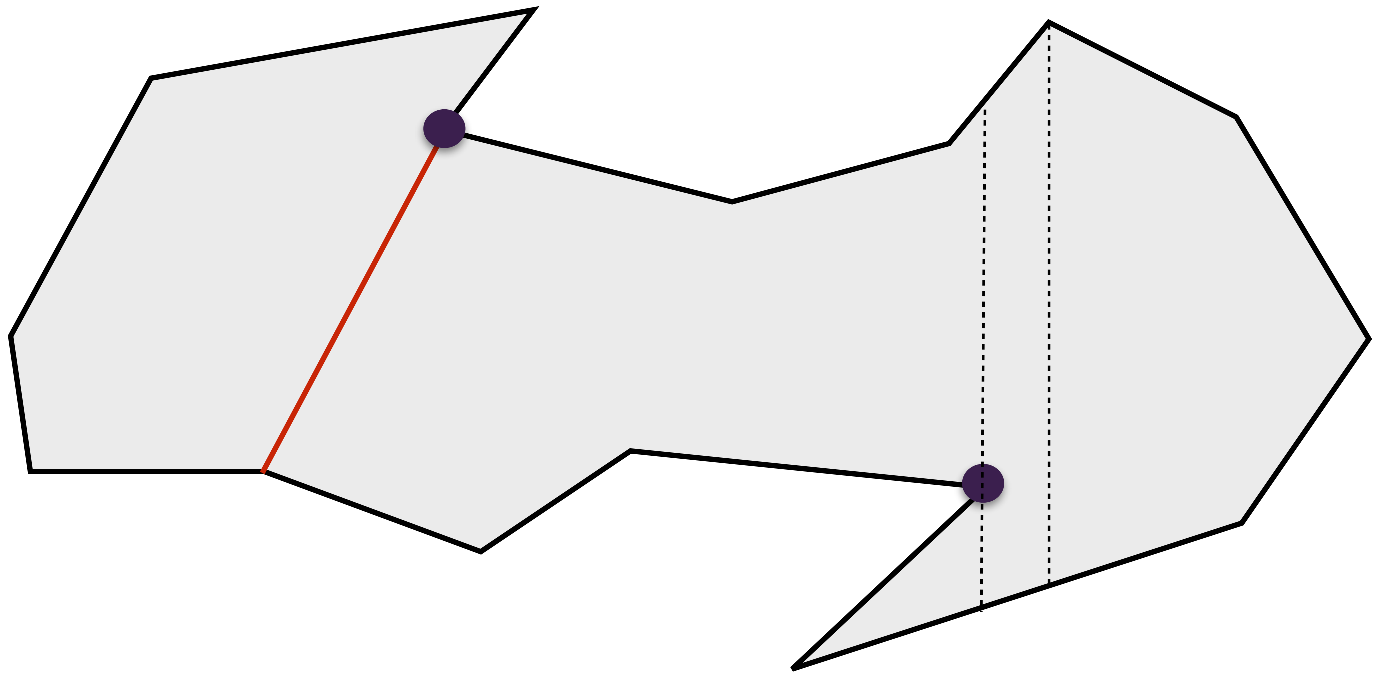
1. Identify cusp vertices
2. Compute a trapezoid partition of P
3. For each cusp vertex, add diagonal in trapezoid before/after the cusp

Removing cusps



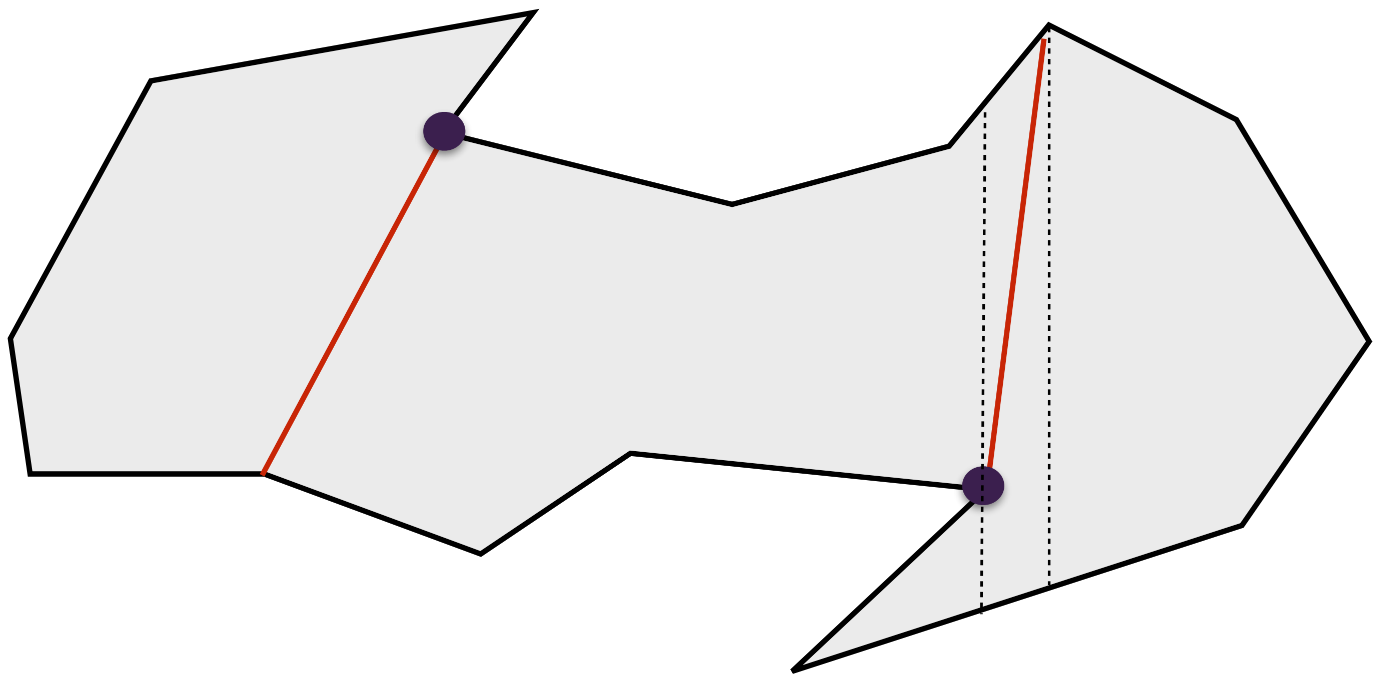
1. Identify cusp vertices
2. Compute a trapezoid partition of P
3. For each cusp vertex, add diagonal in trapezoid before/after the cusp

Removing cusps



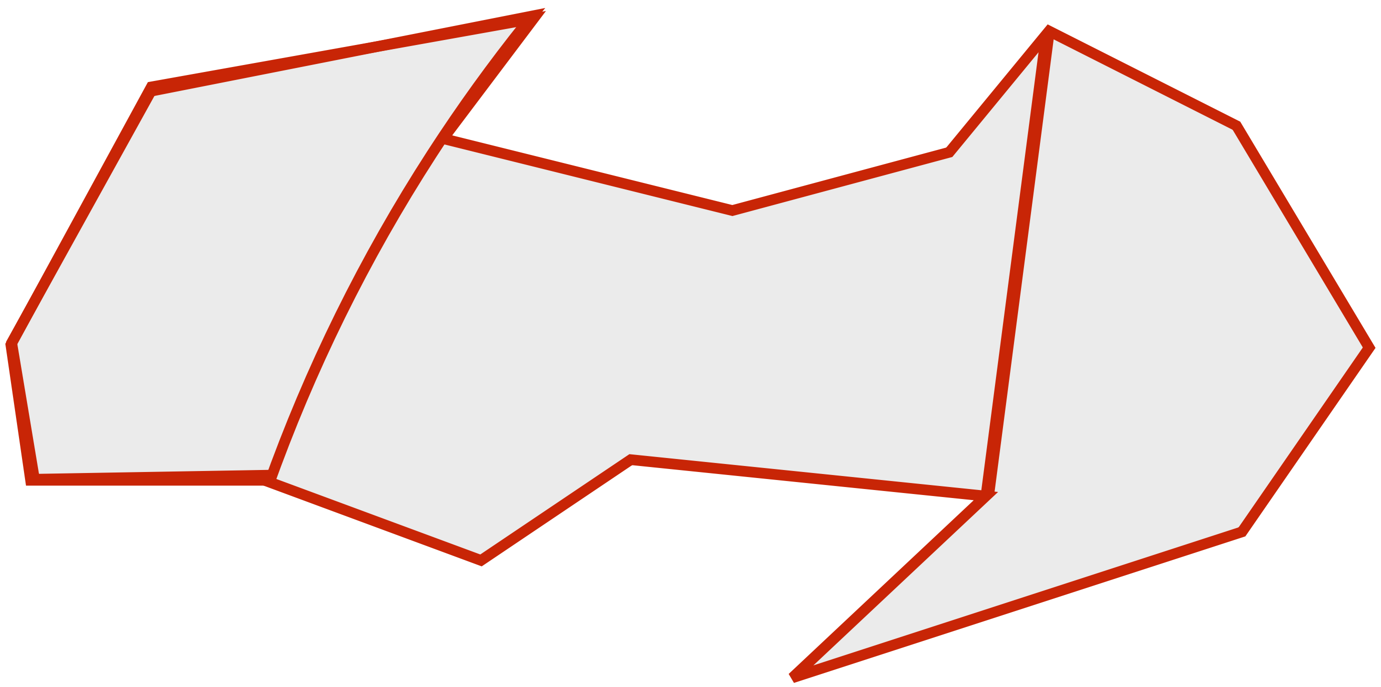
1. Identify cusp vertices
2. Compute a trapezoid partition of P
3. For each cusp vertex, add diagonal in trapezoid before/after the cusp

Removing cusps



1. Identify cusp vertices
2. Compute a trapezoid partition of P
3. For each cusp vertex, add diagonal in trapezoid before/after the cusp

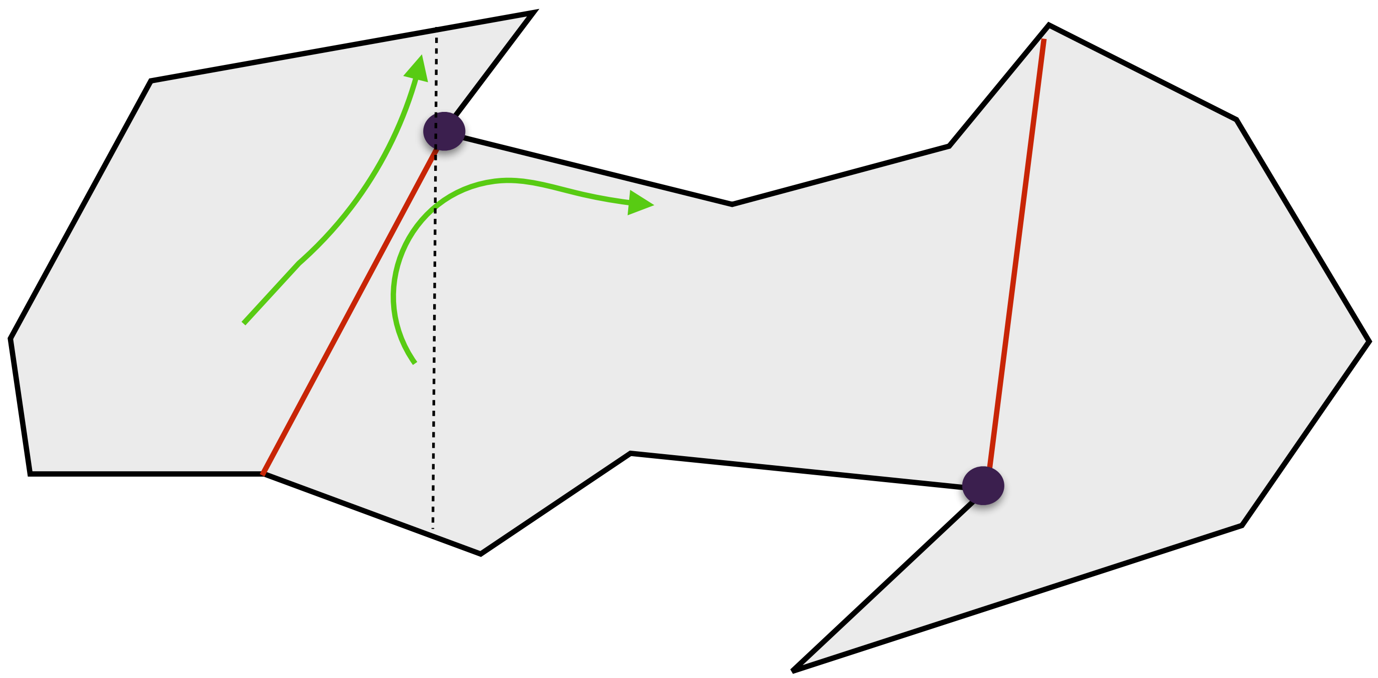
Removing cusps

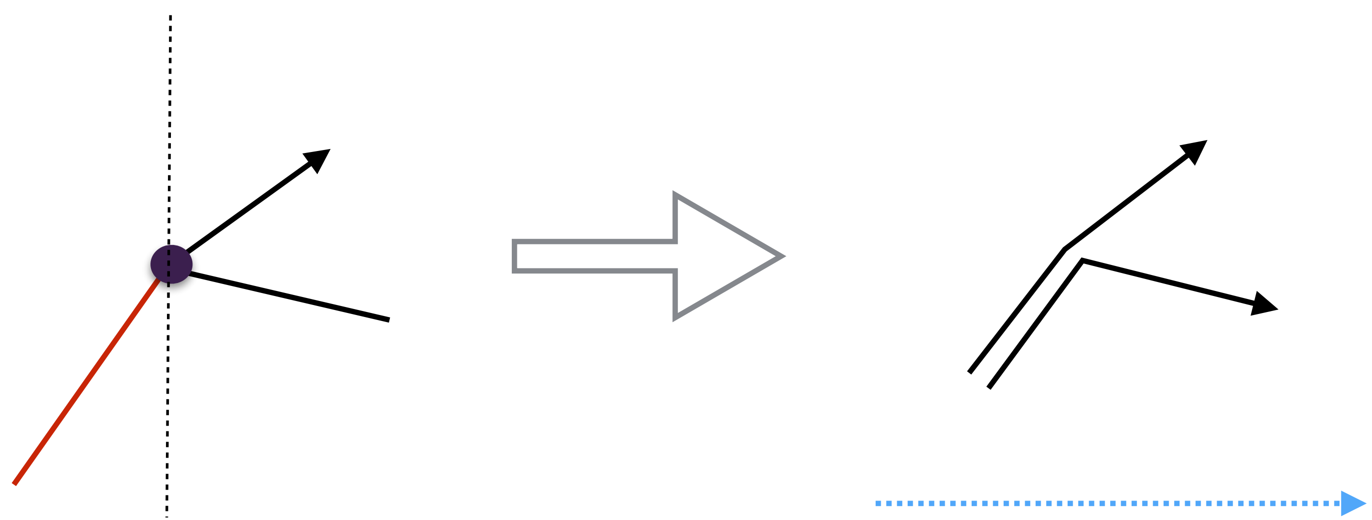


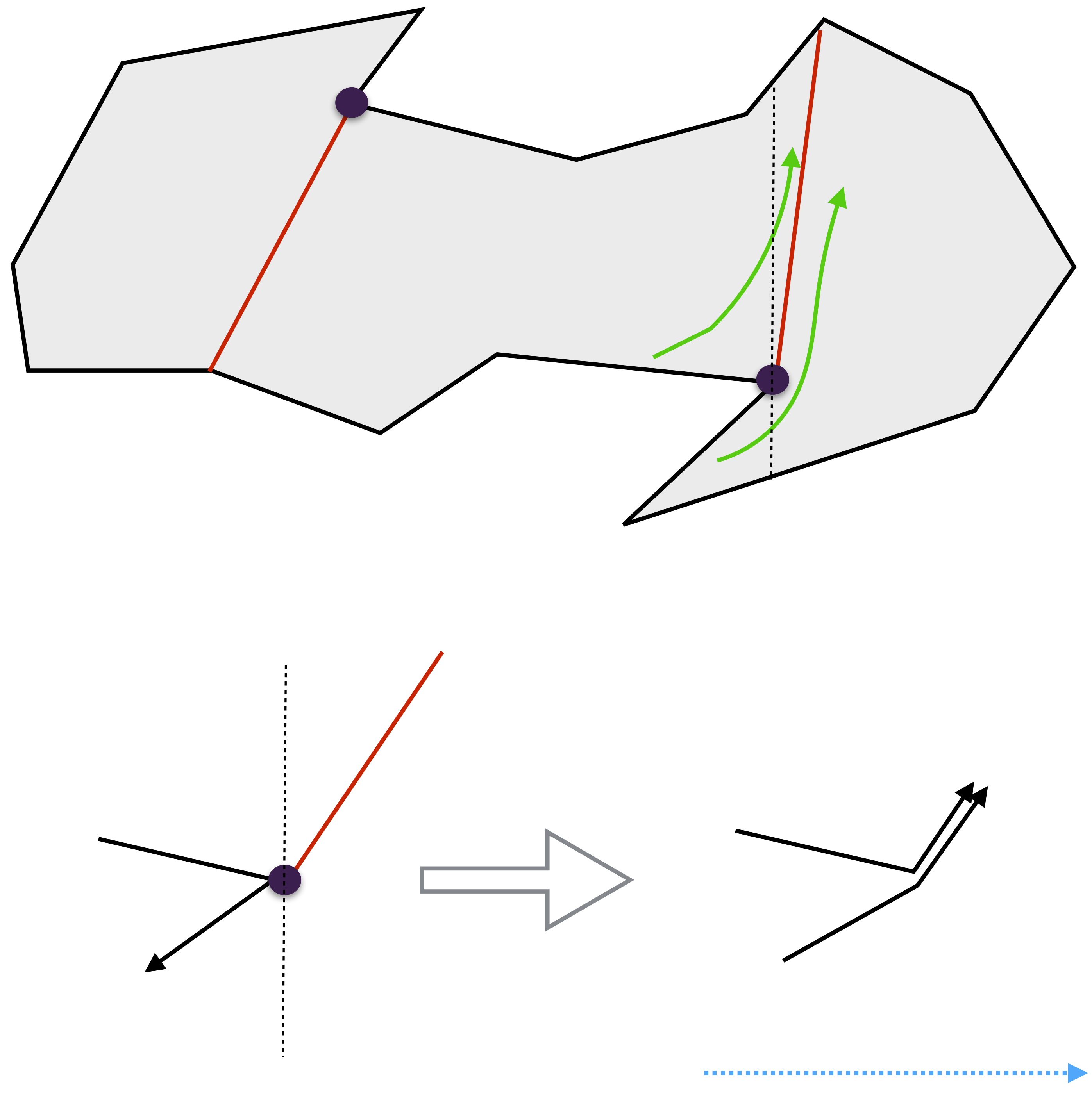
1. Identify cusp vertices
2. Compute a trapezoid partition of P
3. For each cusp vertex, add diagonal in trapezoid before/after the cusp

This creates a partition of P.

Claim: The resulting polygons have no cusps and thus are monotone (by theorem).

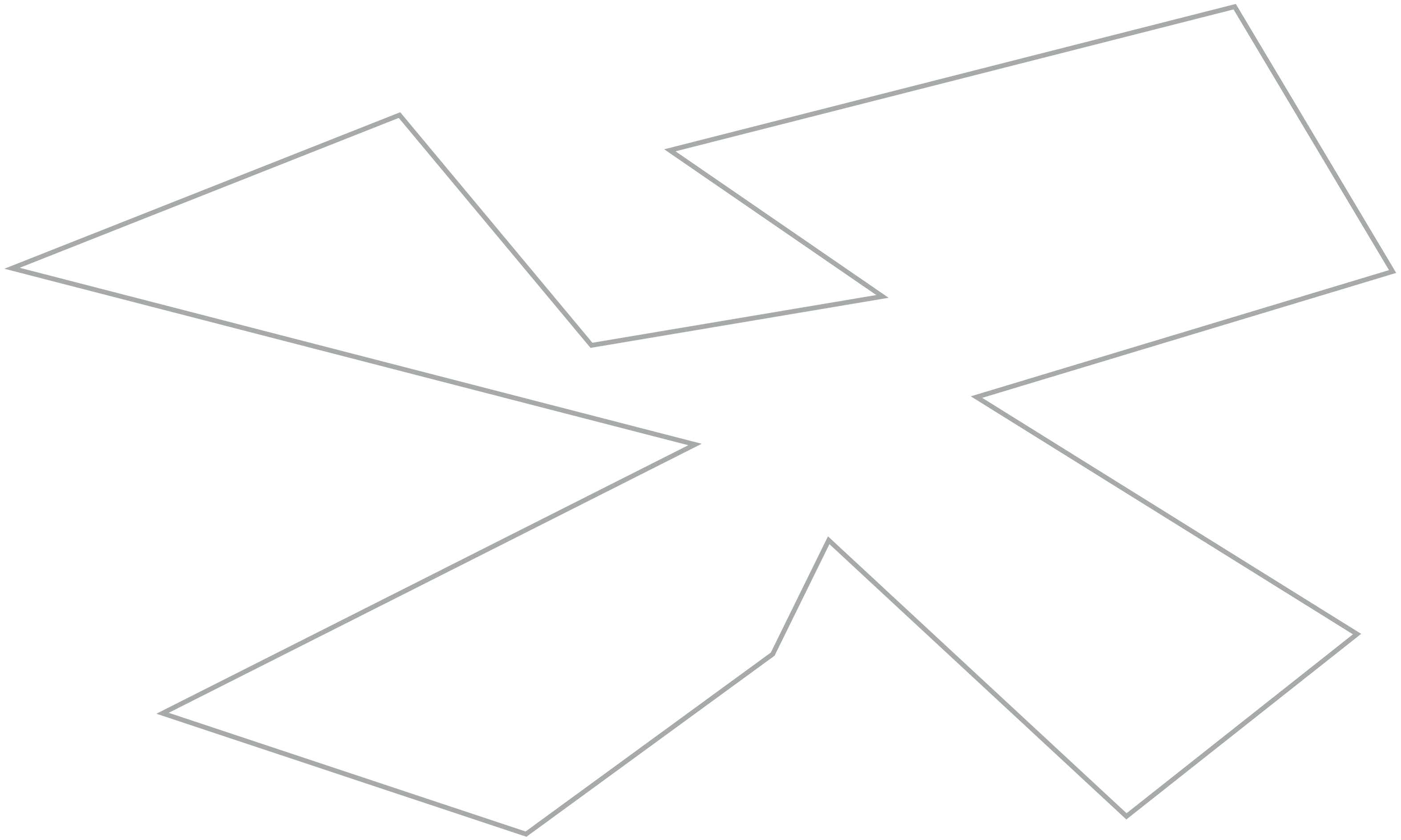




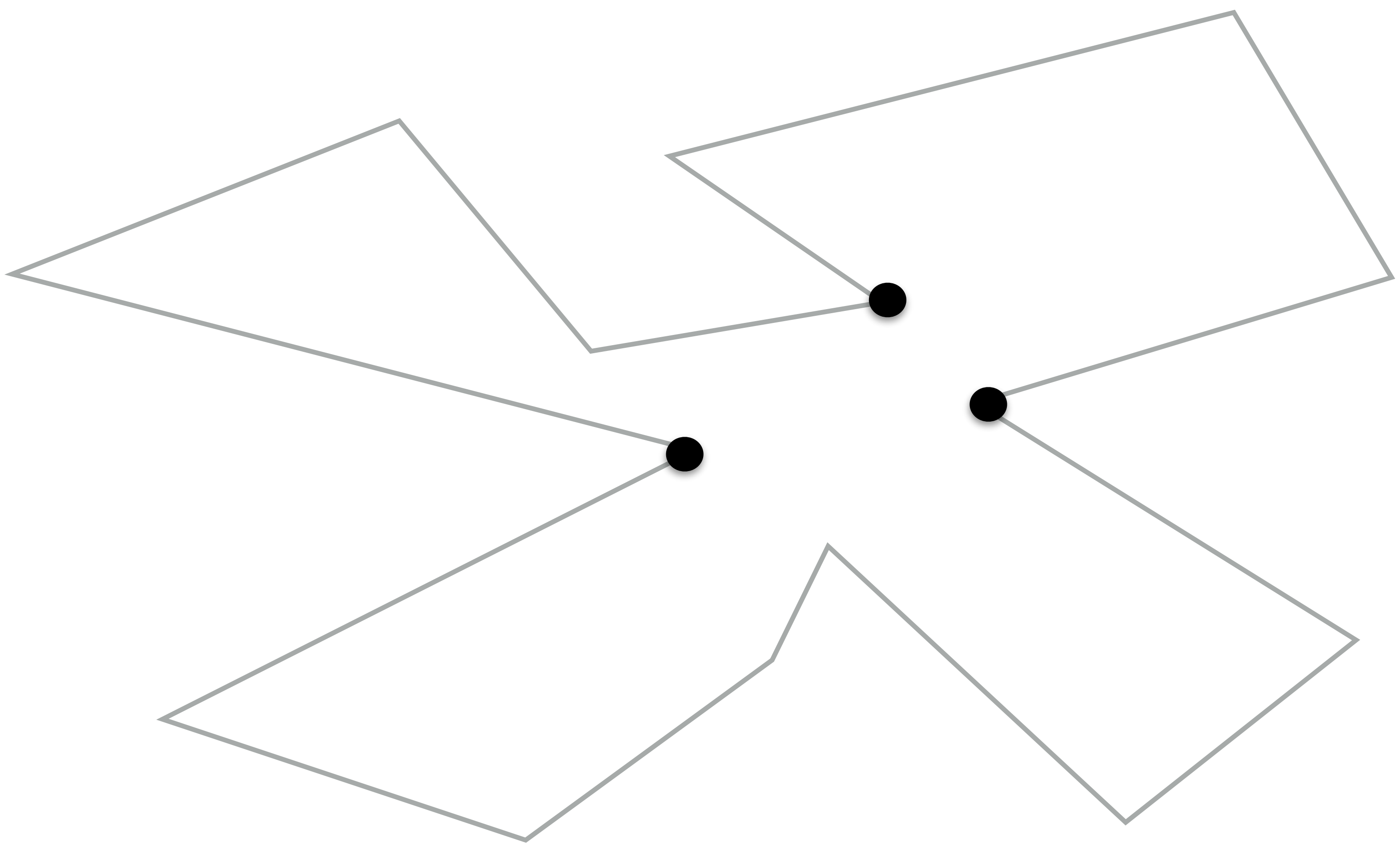


Removing cusps

- Another example

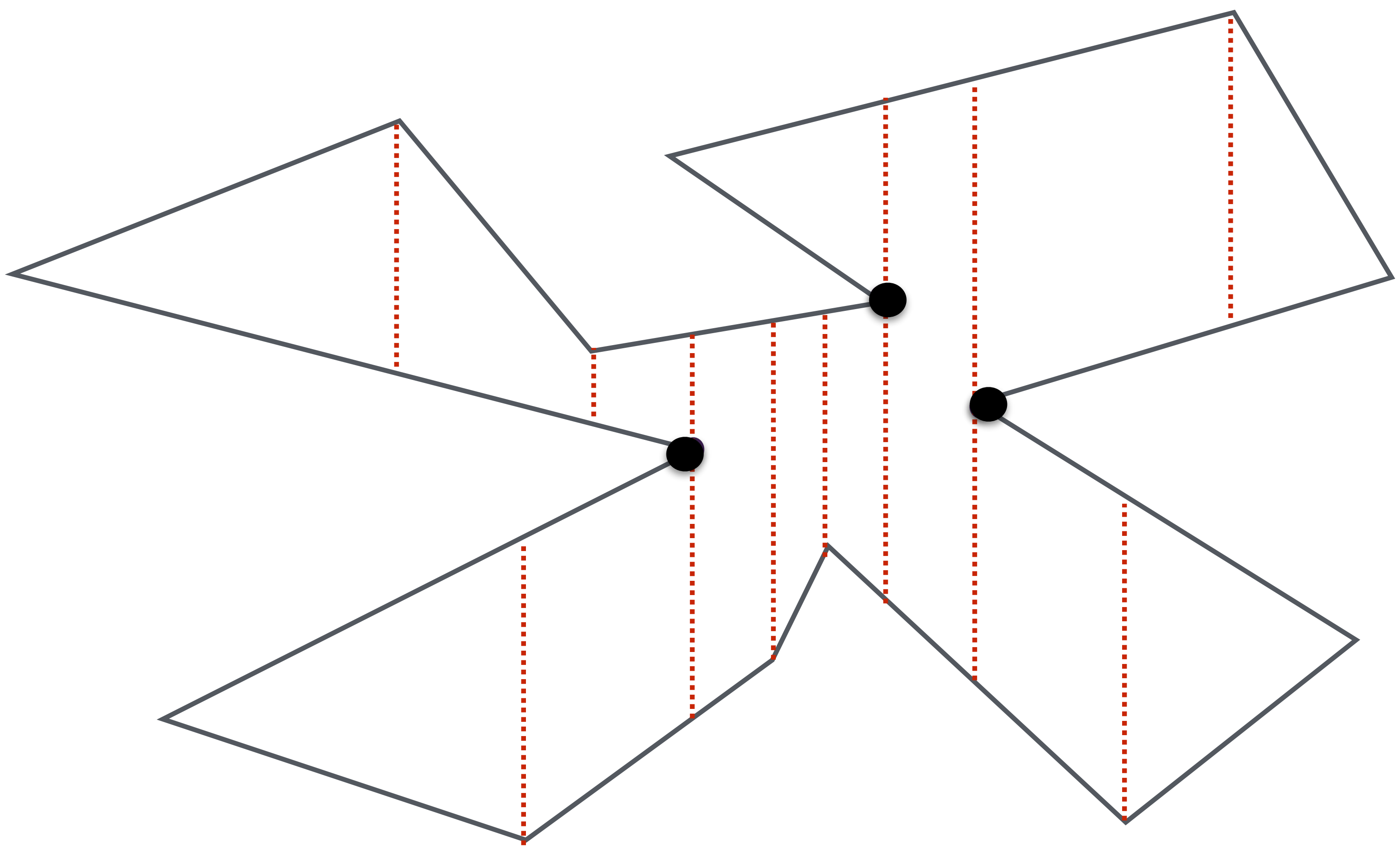


Removing cusps



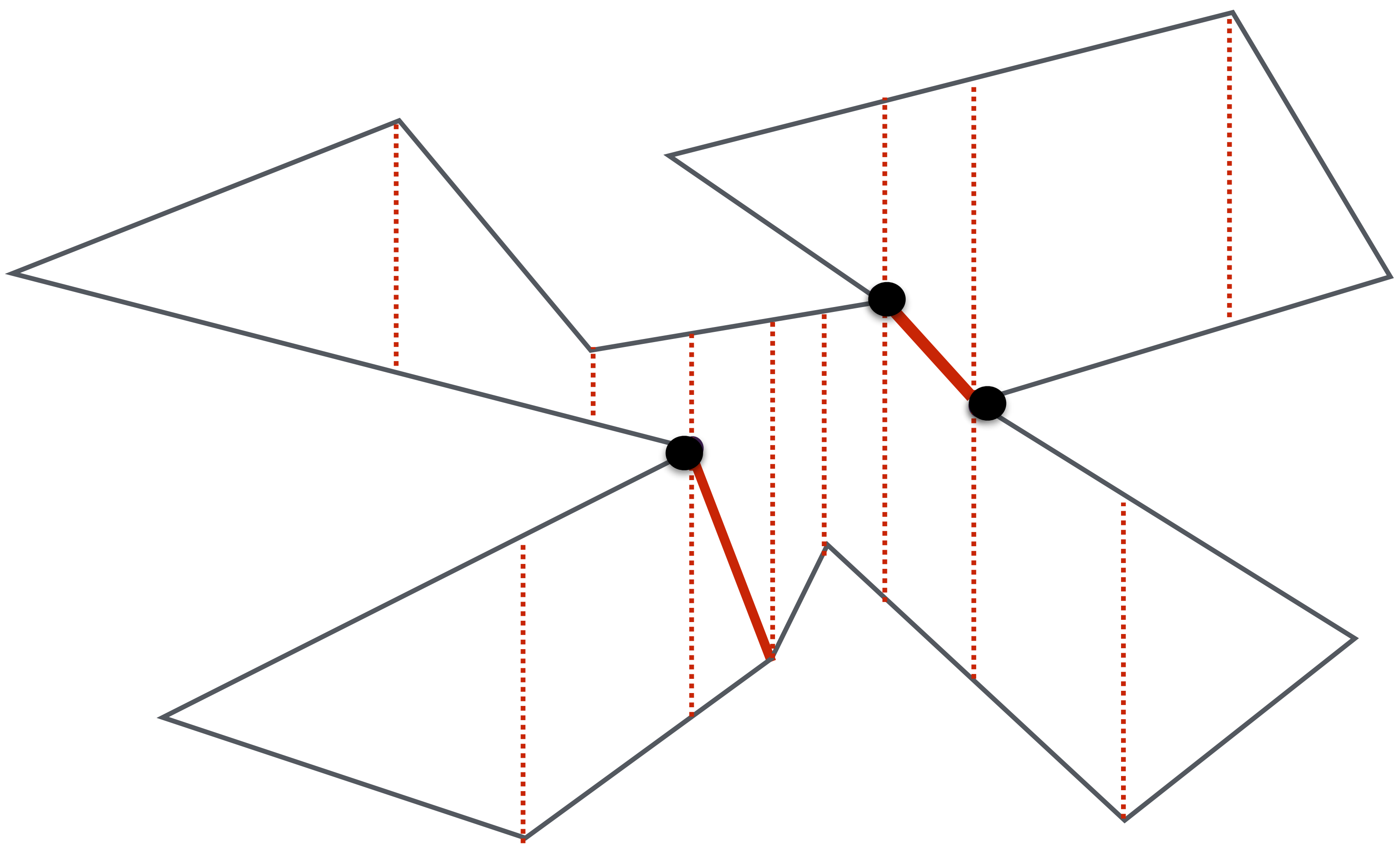
1. Identify cusp vertices

Removing cusps



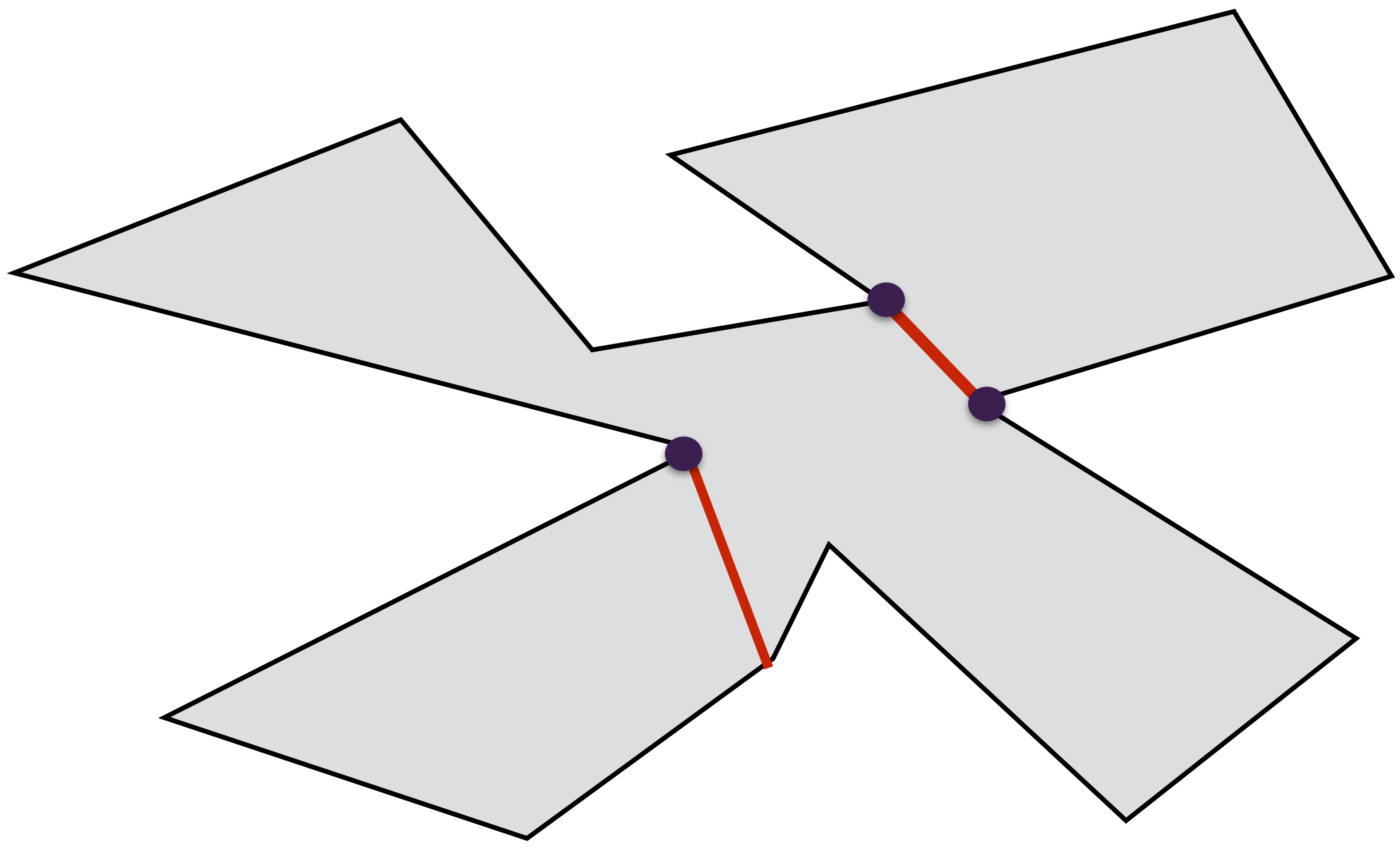
1. Identify cusp vertices
2. Compute a trapezoid partition of P

Removing cusps



1. Identify cusp vertices
2. Compute a trapezoid partition of P
3. Add obvious diagonal before/after each cusp

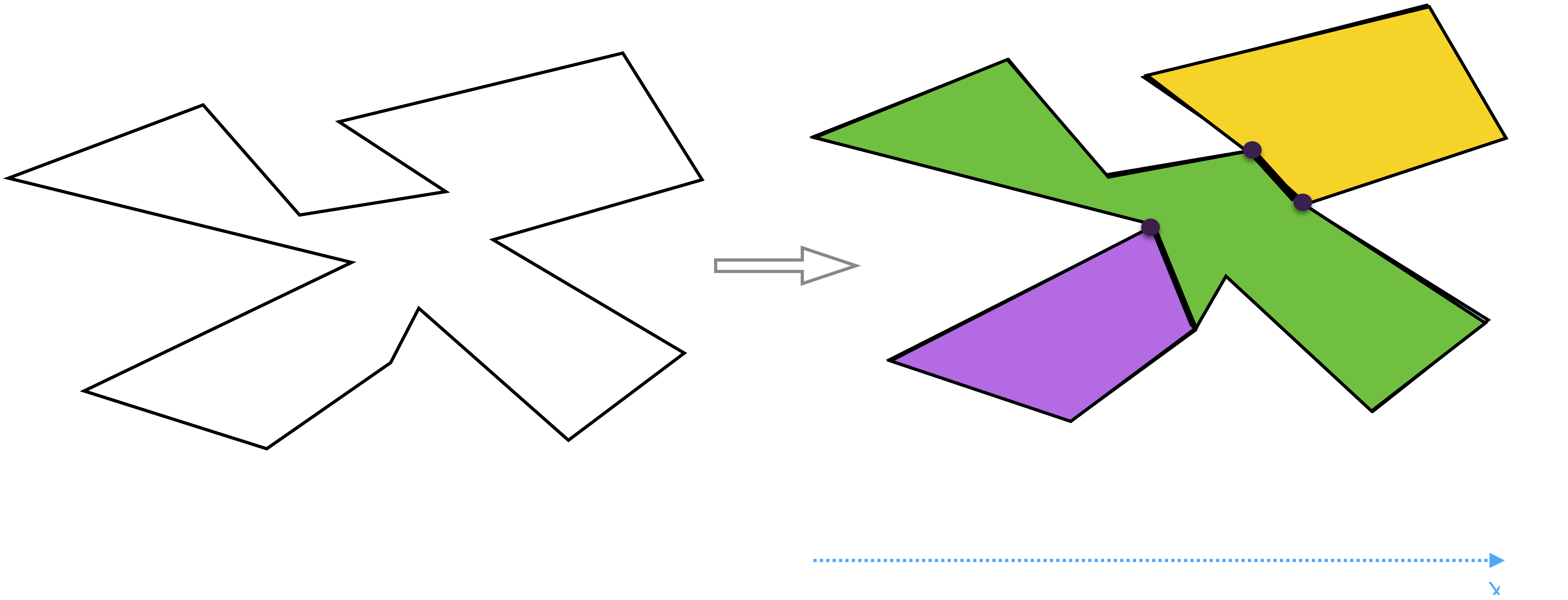
Removing cusps



This partitions the polygon into monotone pieces.

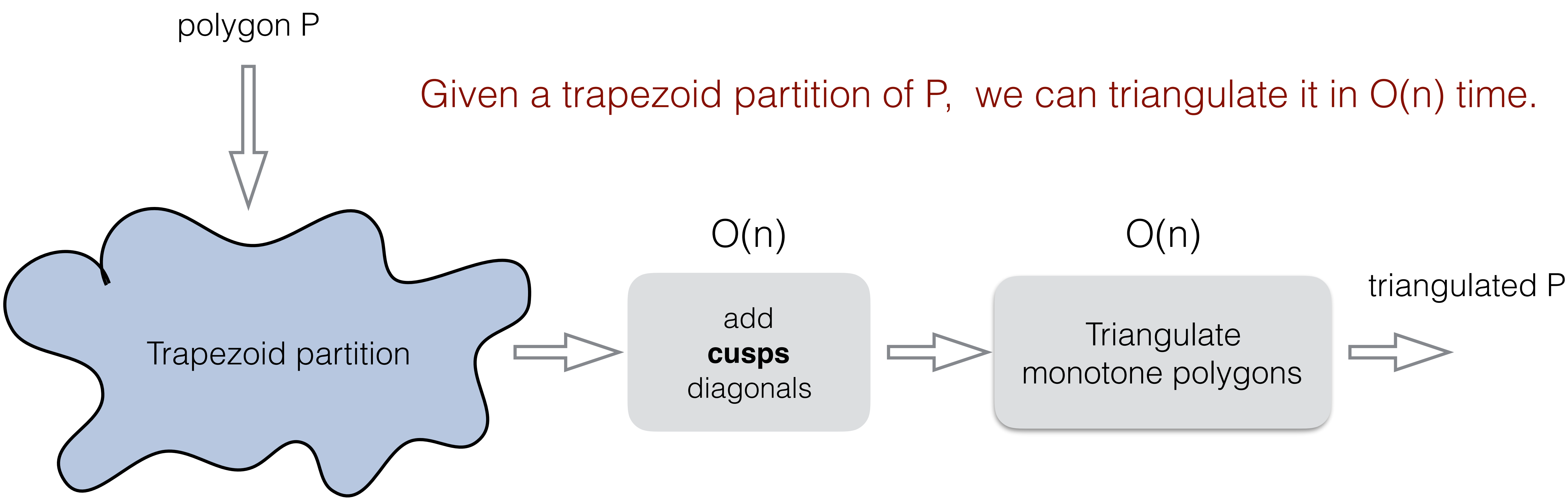
Partition P into monotone polygons

1. Identify cusp vertices
2. Compute a trapezoid partition of P
3. Add obvious diagonal before/after each cusp



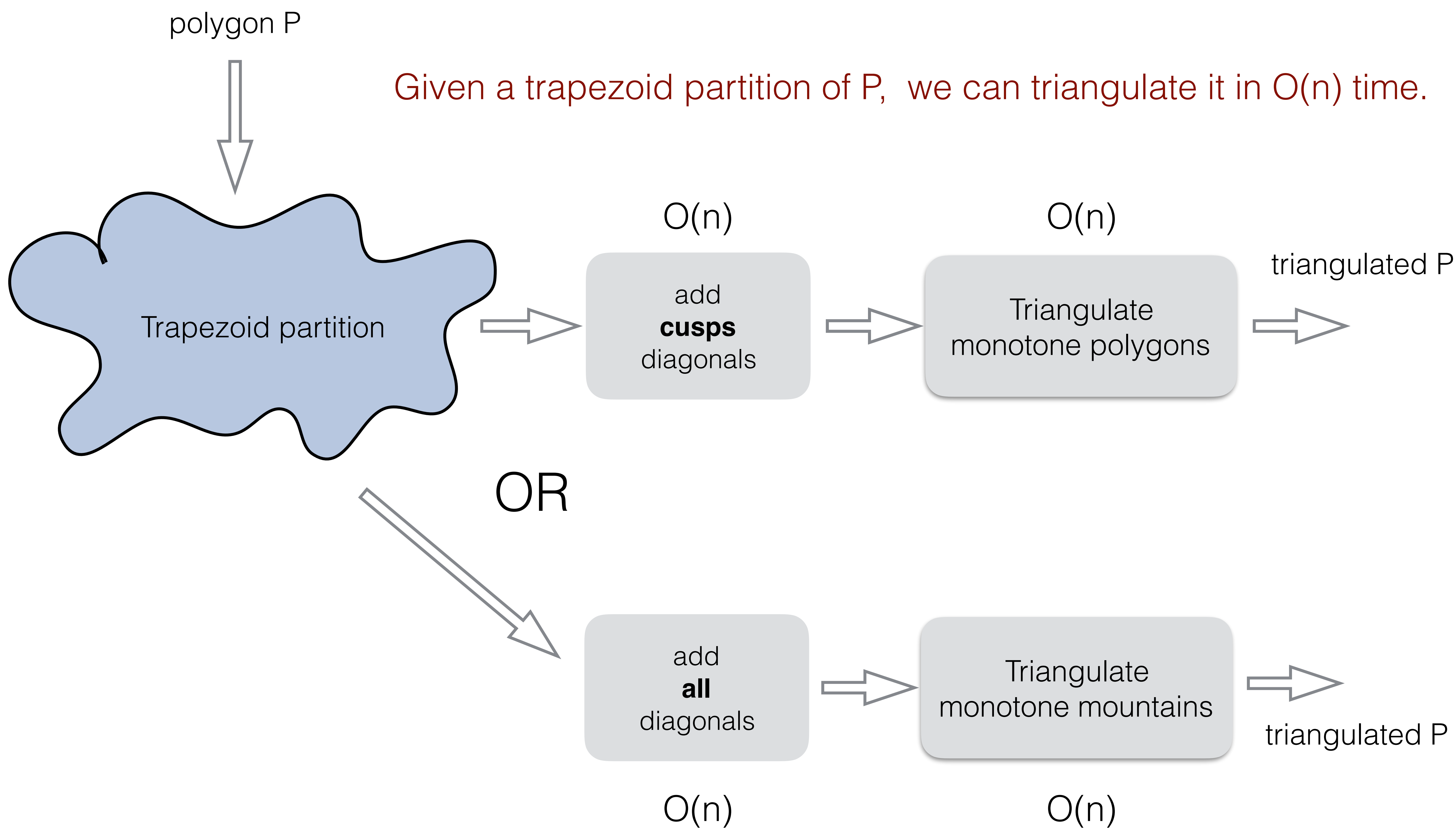
Towards an O(n lg n) Polygon Triangulation Algorithm

Given a trapezoid partition of P, we can triangulate it in O(n) time.

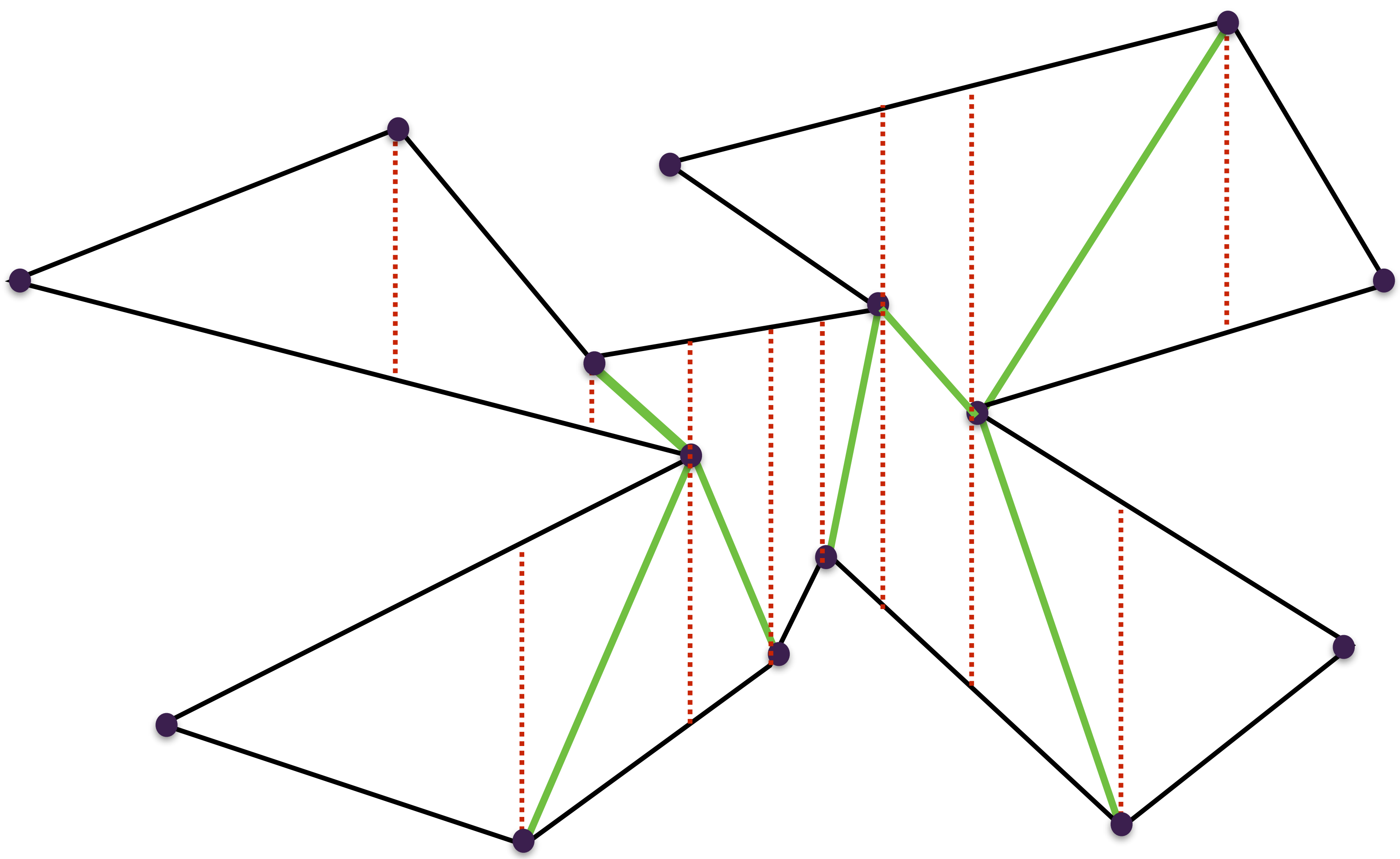


Towards an O(n lg n) Polygon Triangulation Algorithm

Given a trapezoid partition of P, we can triangulate it in O(n) time.

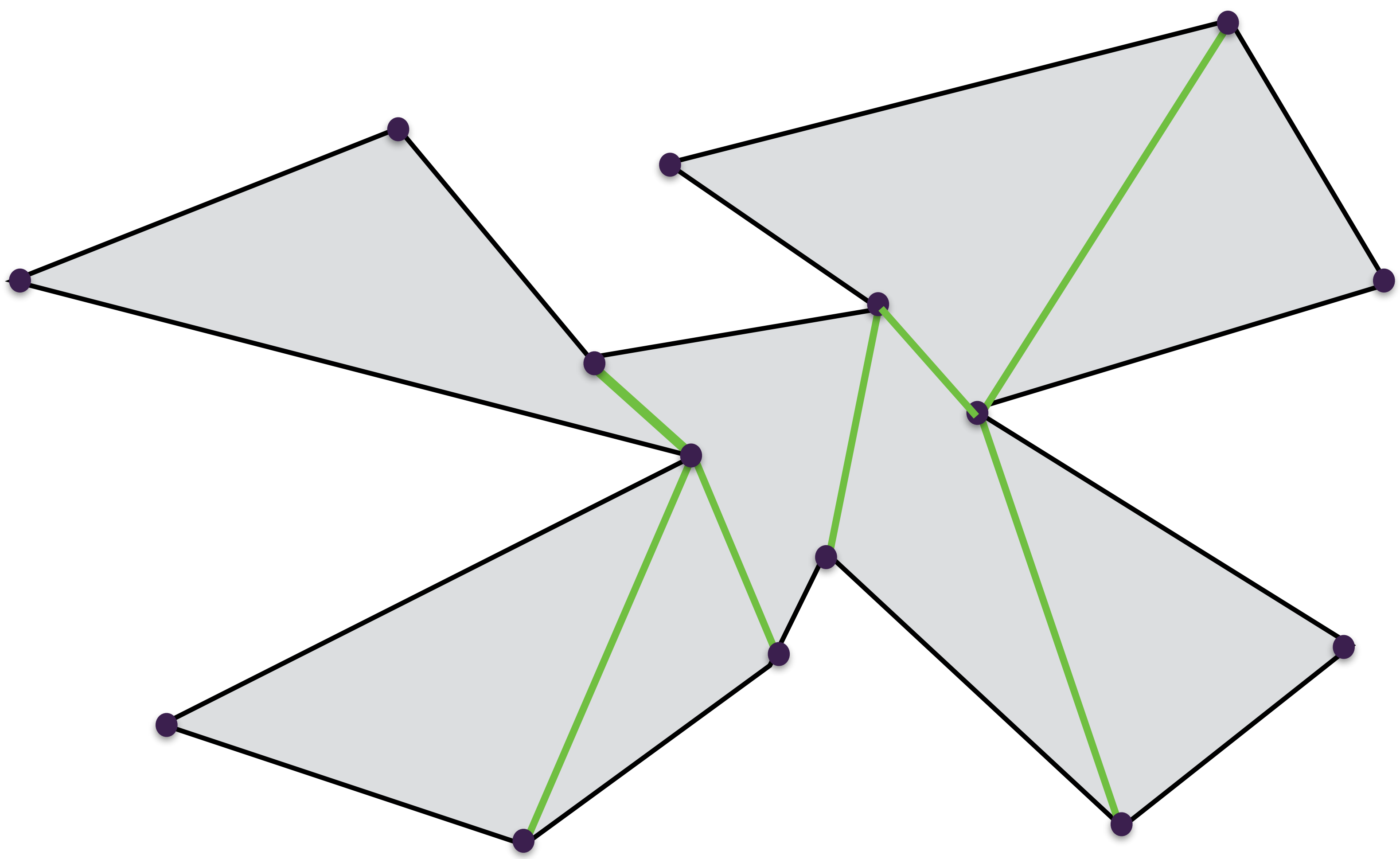


Partitioning into monotone mountains



1. Compute a trapezoid partition of P
2. Output **all** diagonals.

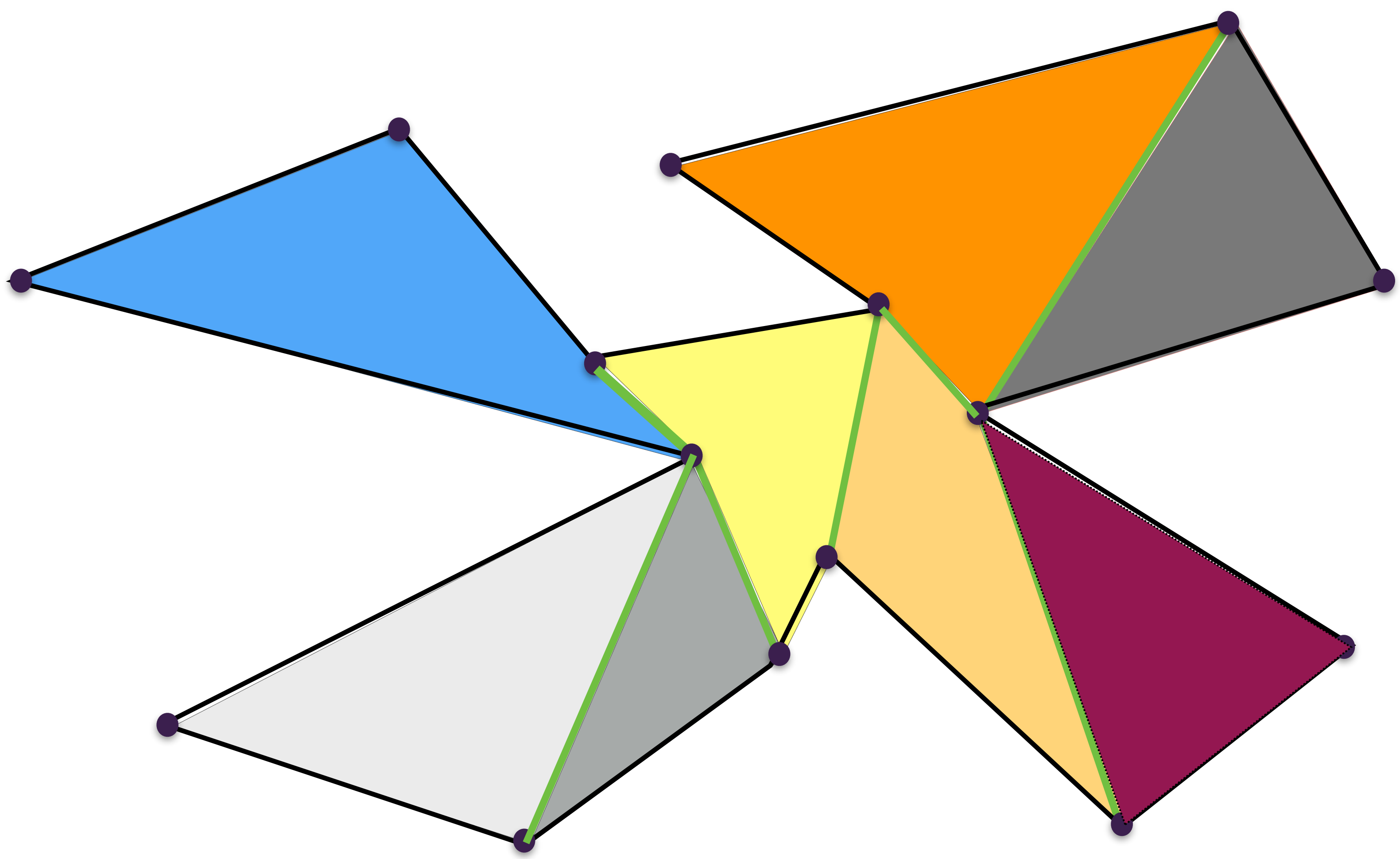
Partitioning into monotone mountains



1. Compute a trapezoid partition of P
2. Output **all** diagonals.

   Claim: All diagonals partition the polygon into monotone mountains.

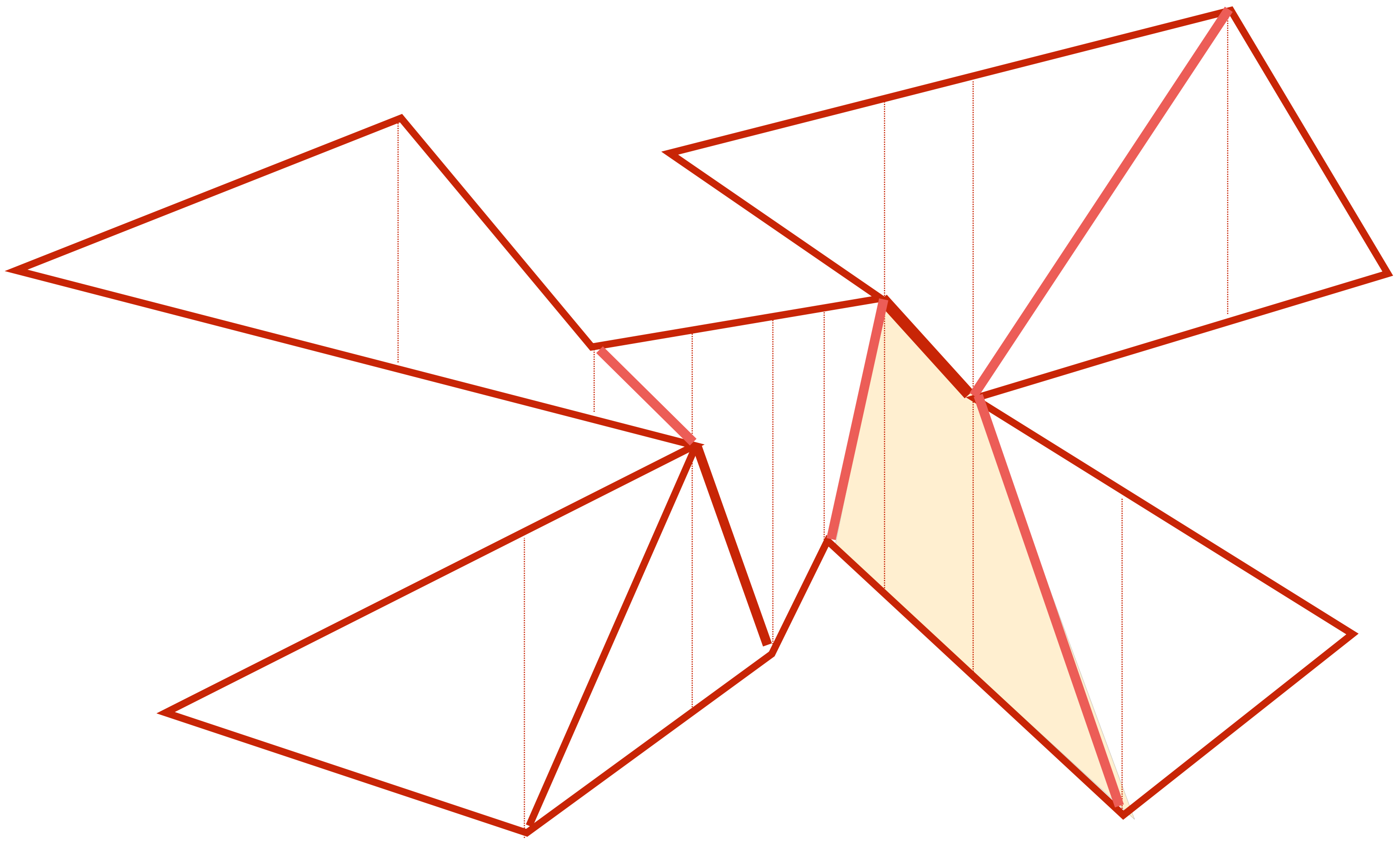
Partitioning into monotone mountains



1. Compute a trapezoid partition of P
2. Output **all** diagonals.

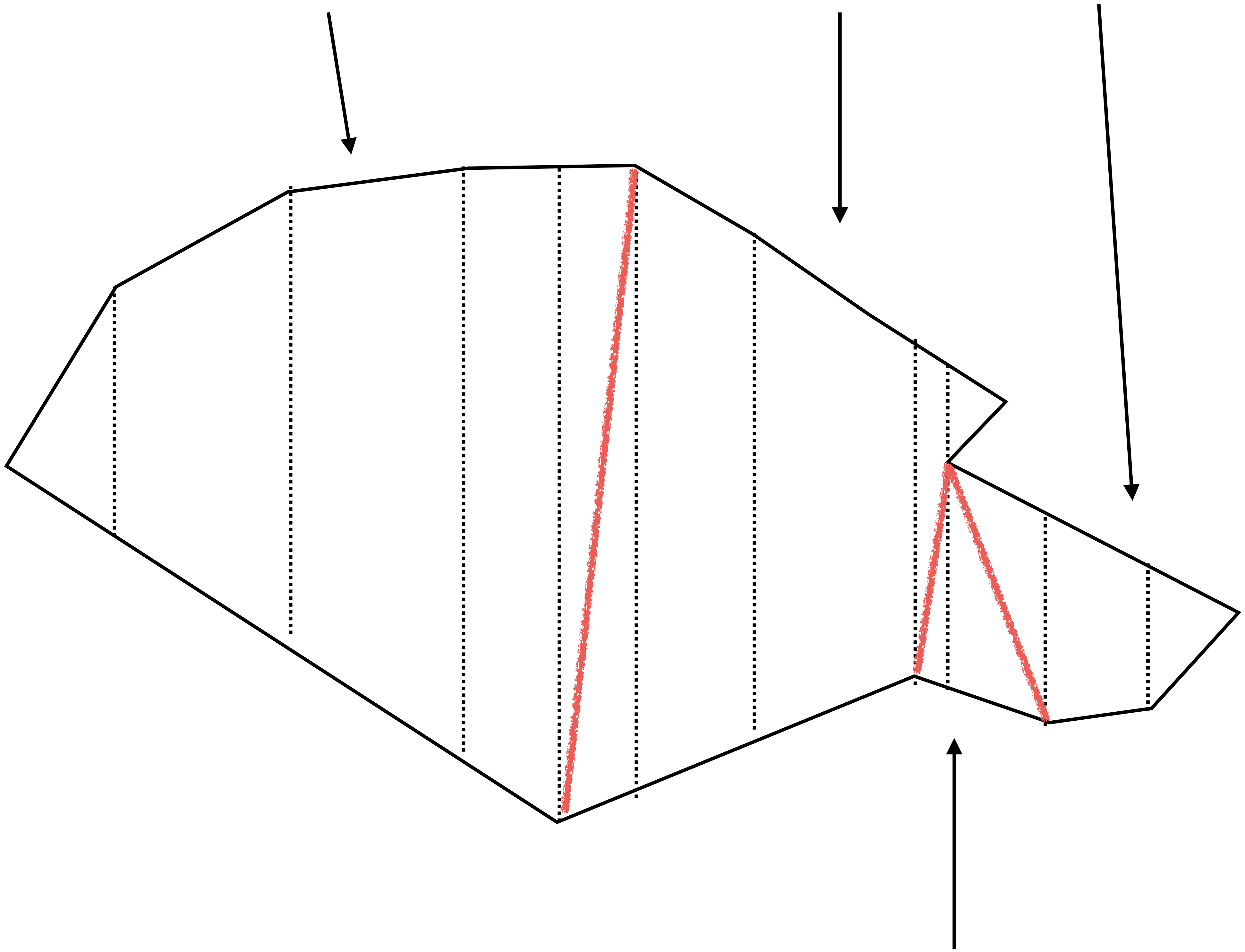
   Claim: The diagonals partition the polygon into monotone mountains.

Claim: The diagonals partition the polygon into monotone mountains.

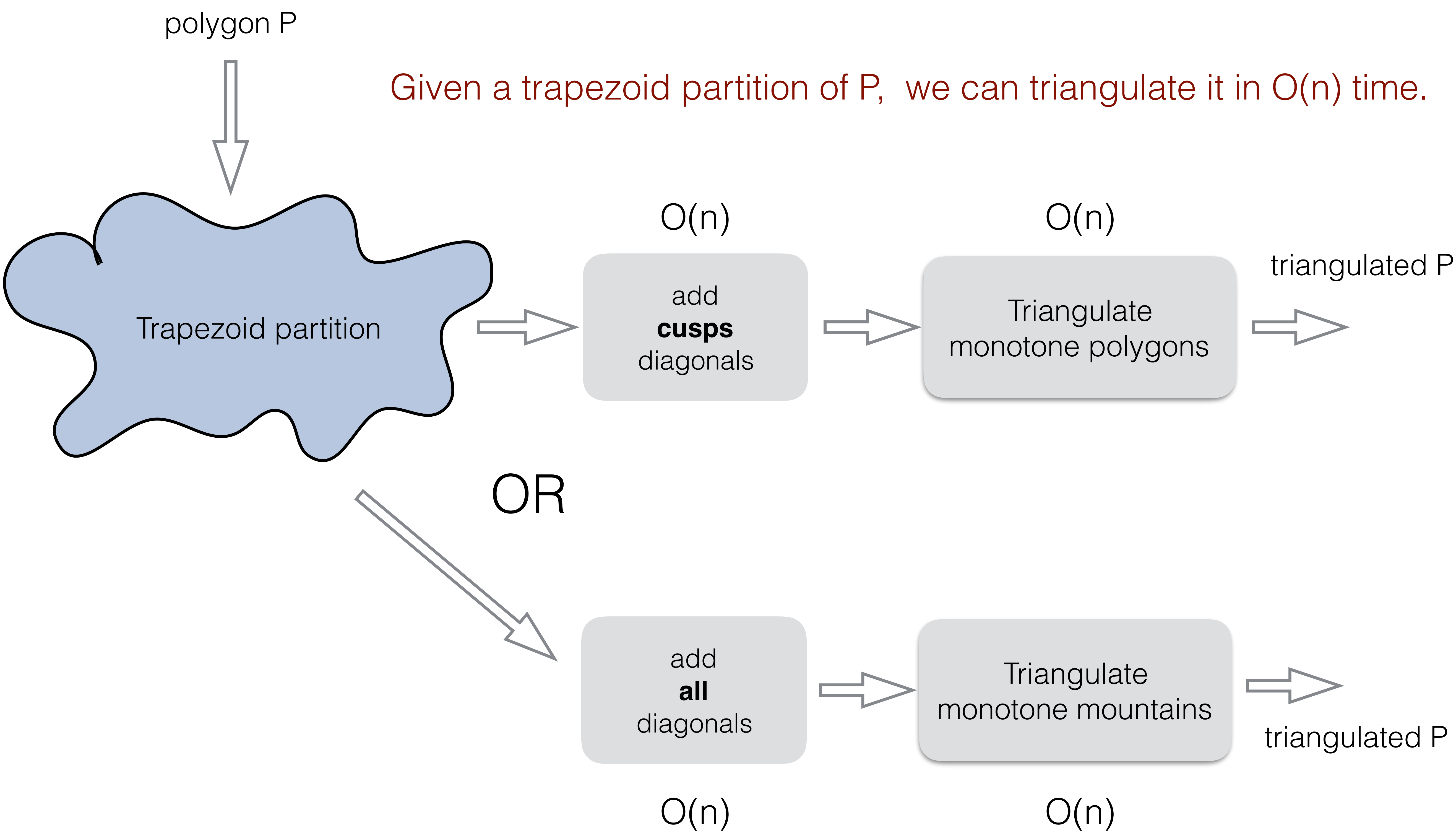


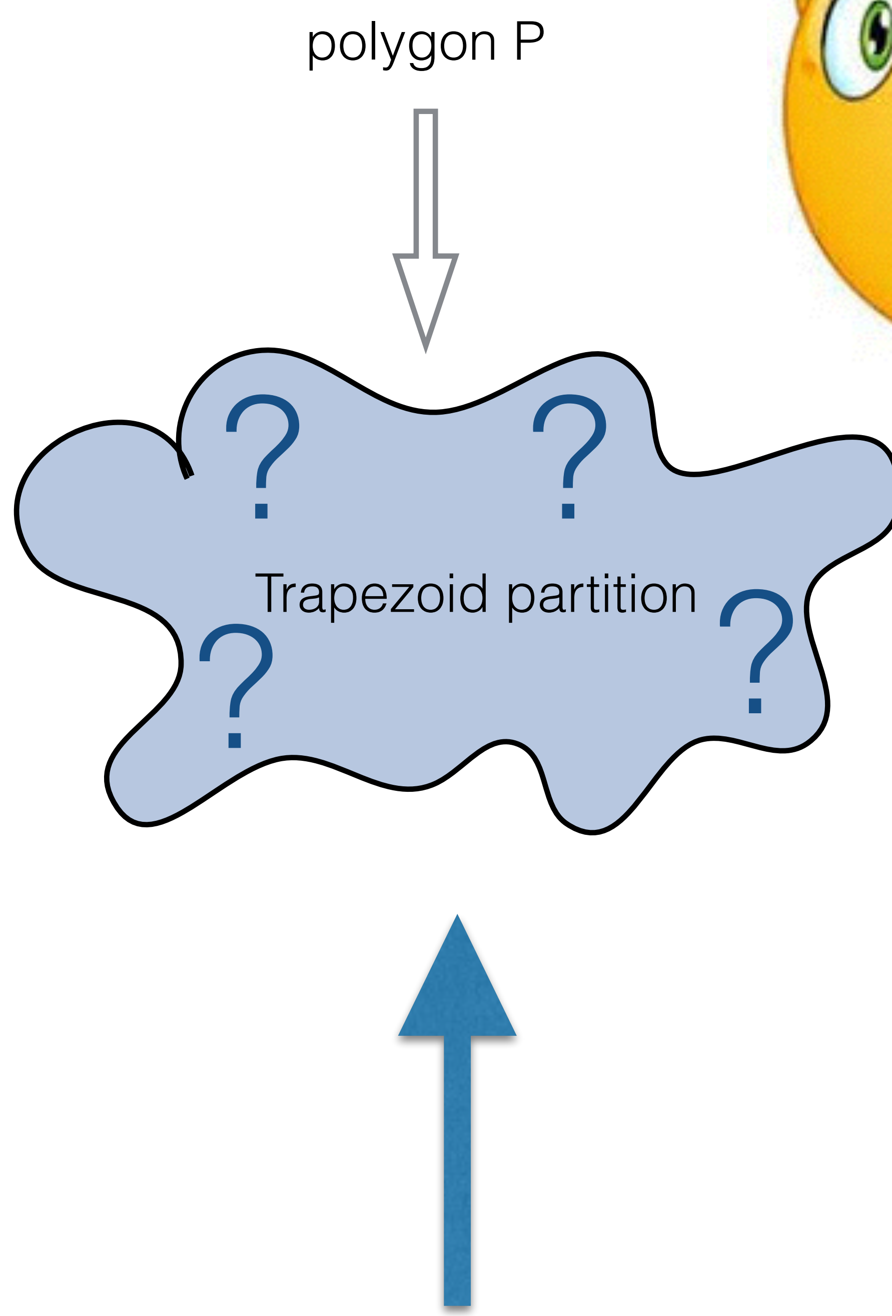
Proof idea:

- Each "internal" trapezoid in a polygon must have the polygon vertices either both above (or both below), otherwise they would generate a diagonal => all internal trapezoids have same edge below/above => one edge there cannot be any diagonal here



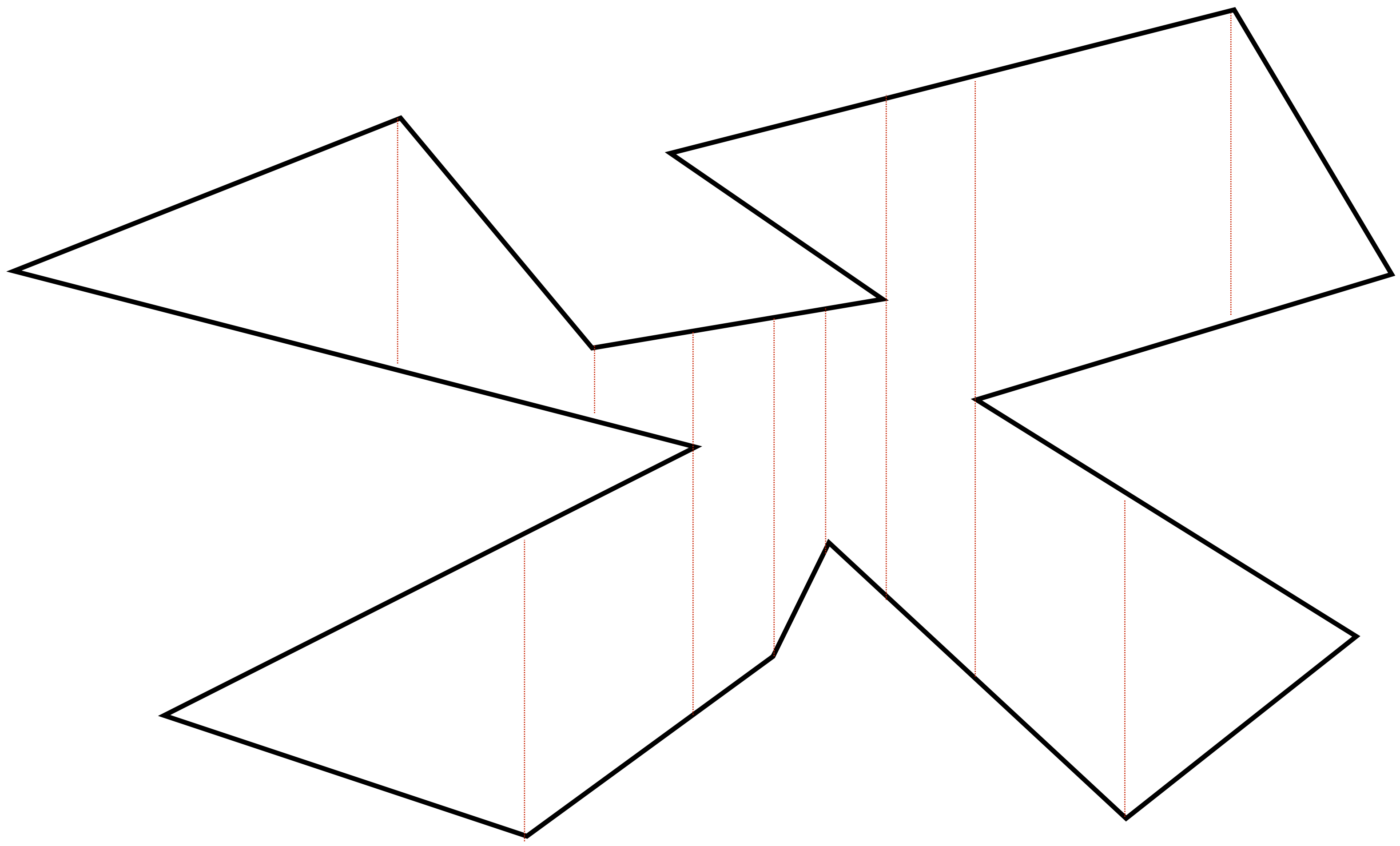
Towards an O(n lg n) Polygon Triangulation Algorithm





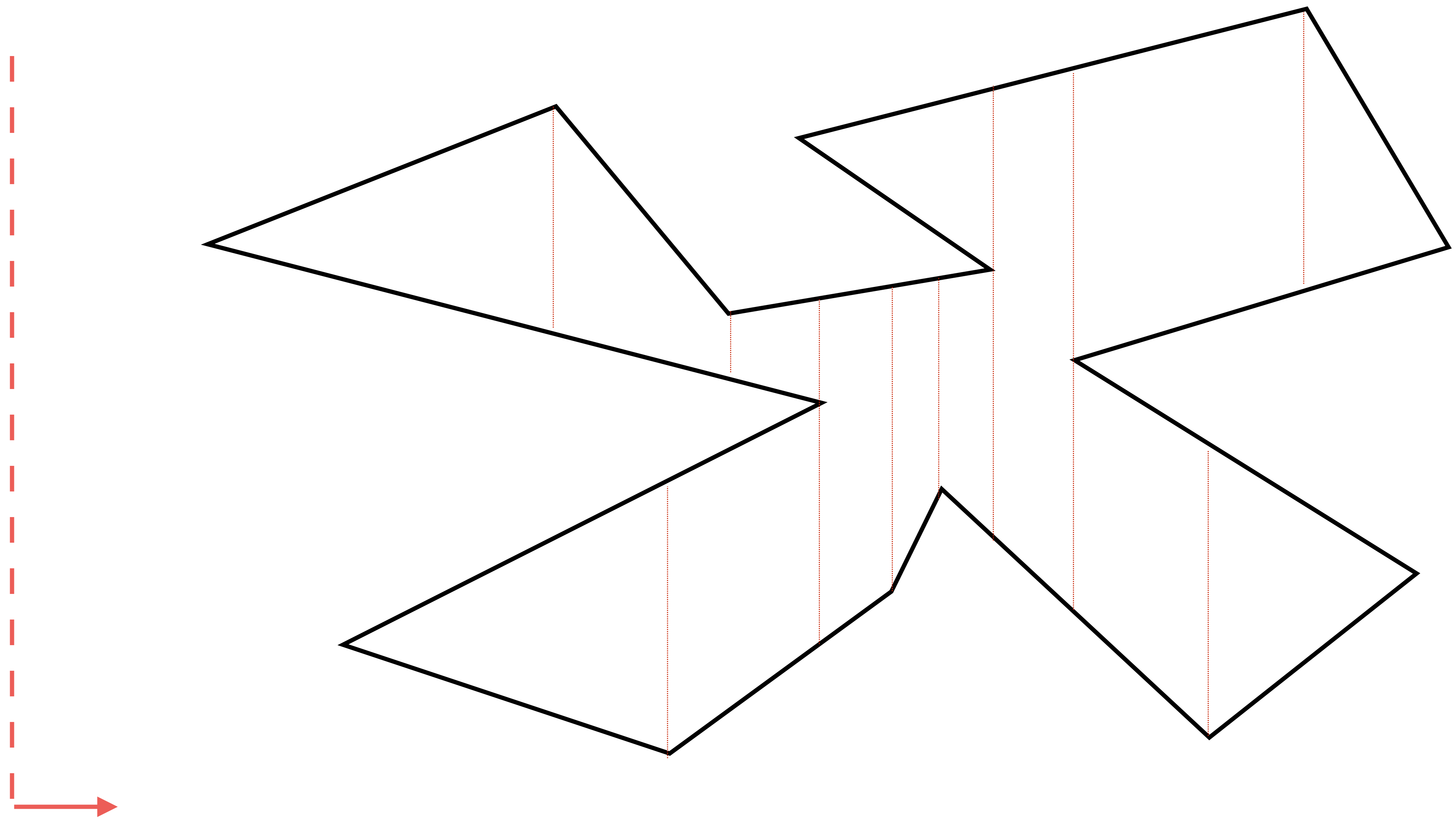
How do we compute
the trapezoid partition?

Computing the trapezoid partition



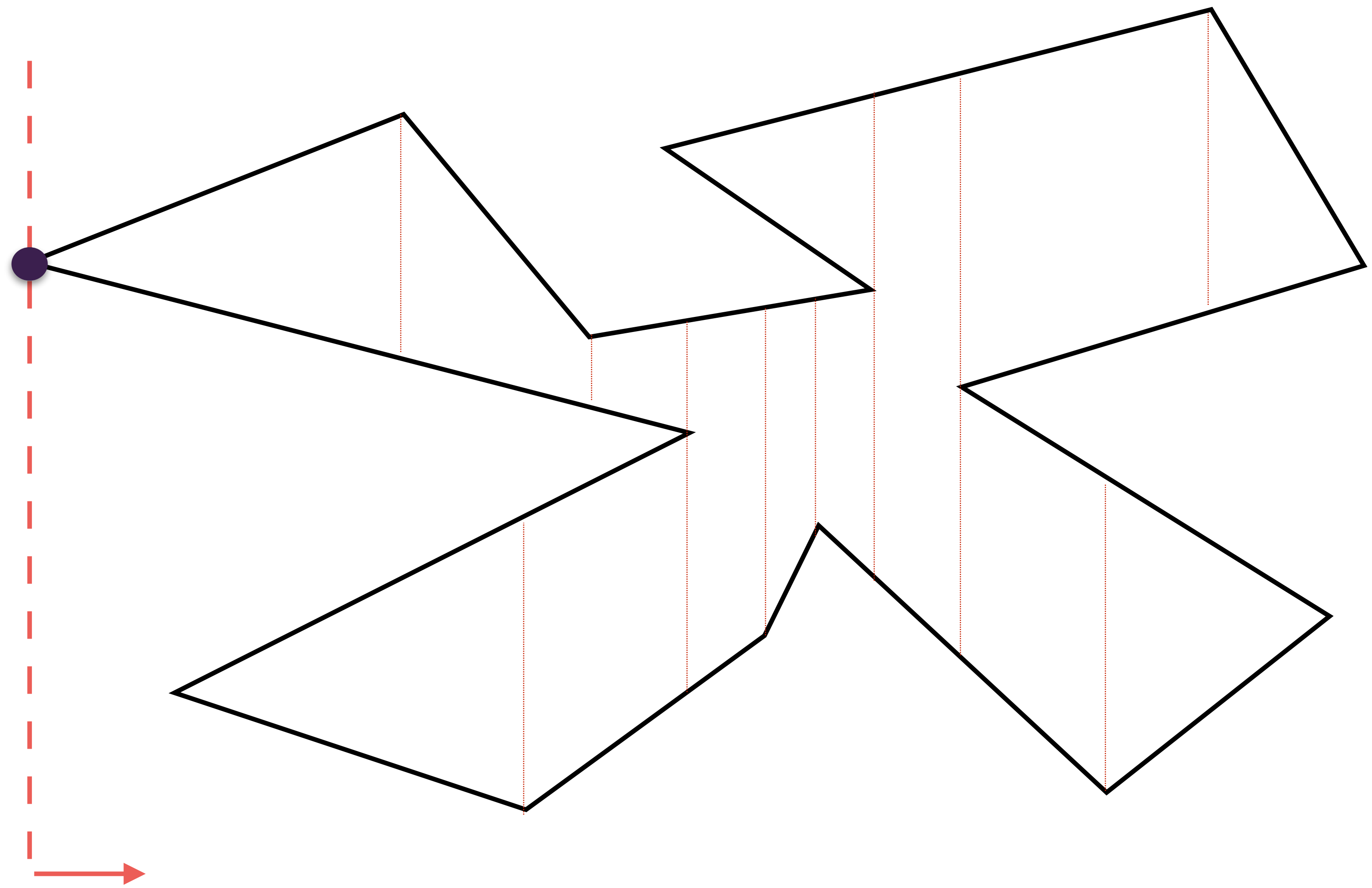
Computing the trapezoid partition in O(n lg n)

- Plane sweep



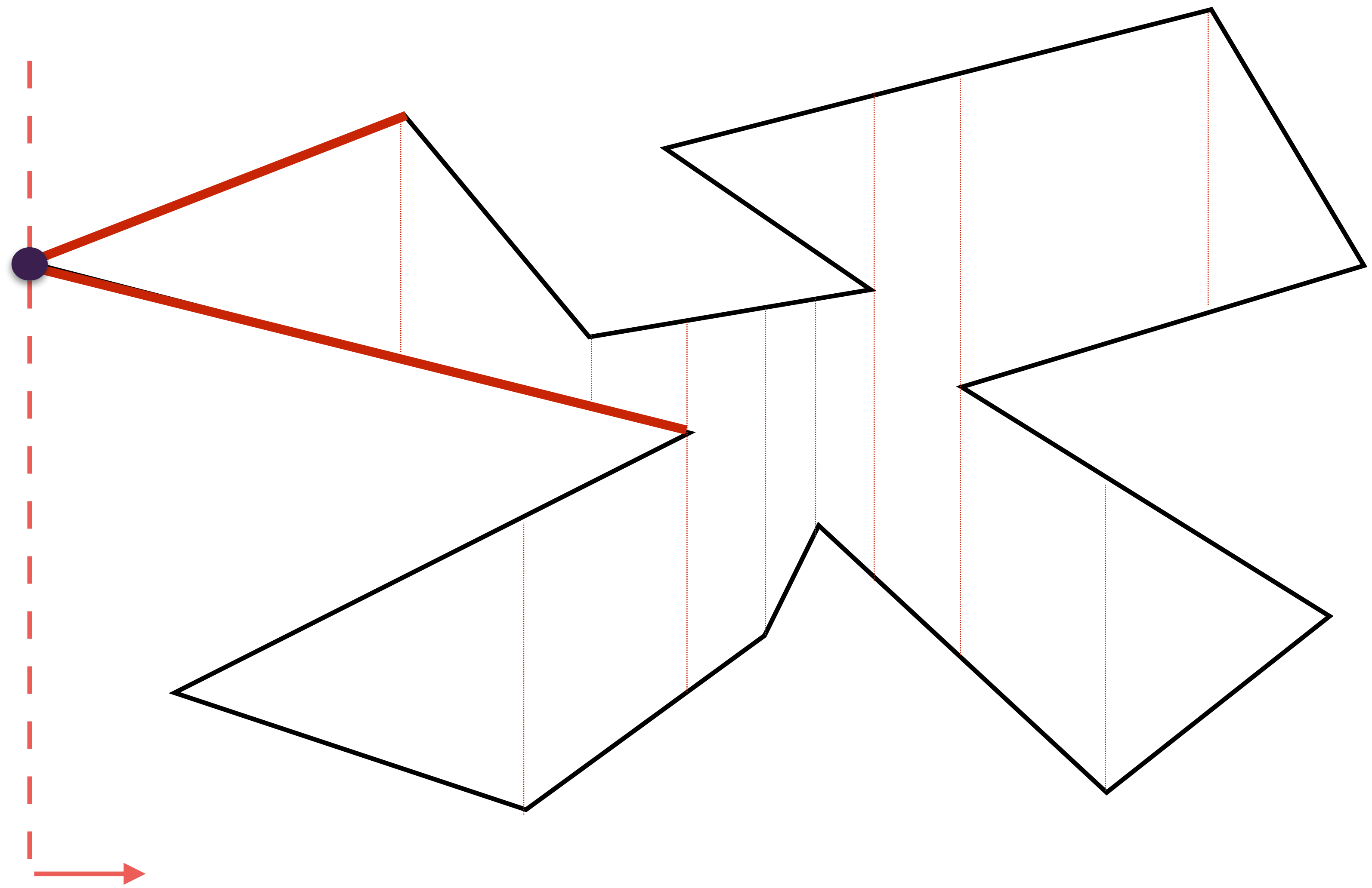
Computing the trapezoid partition in O(n lg n)

- Plane sweep



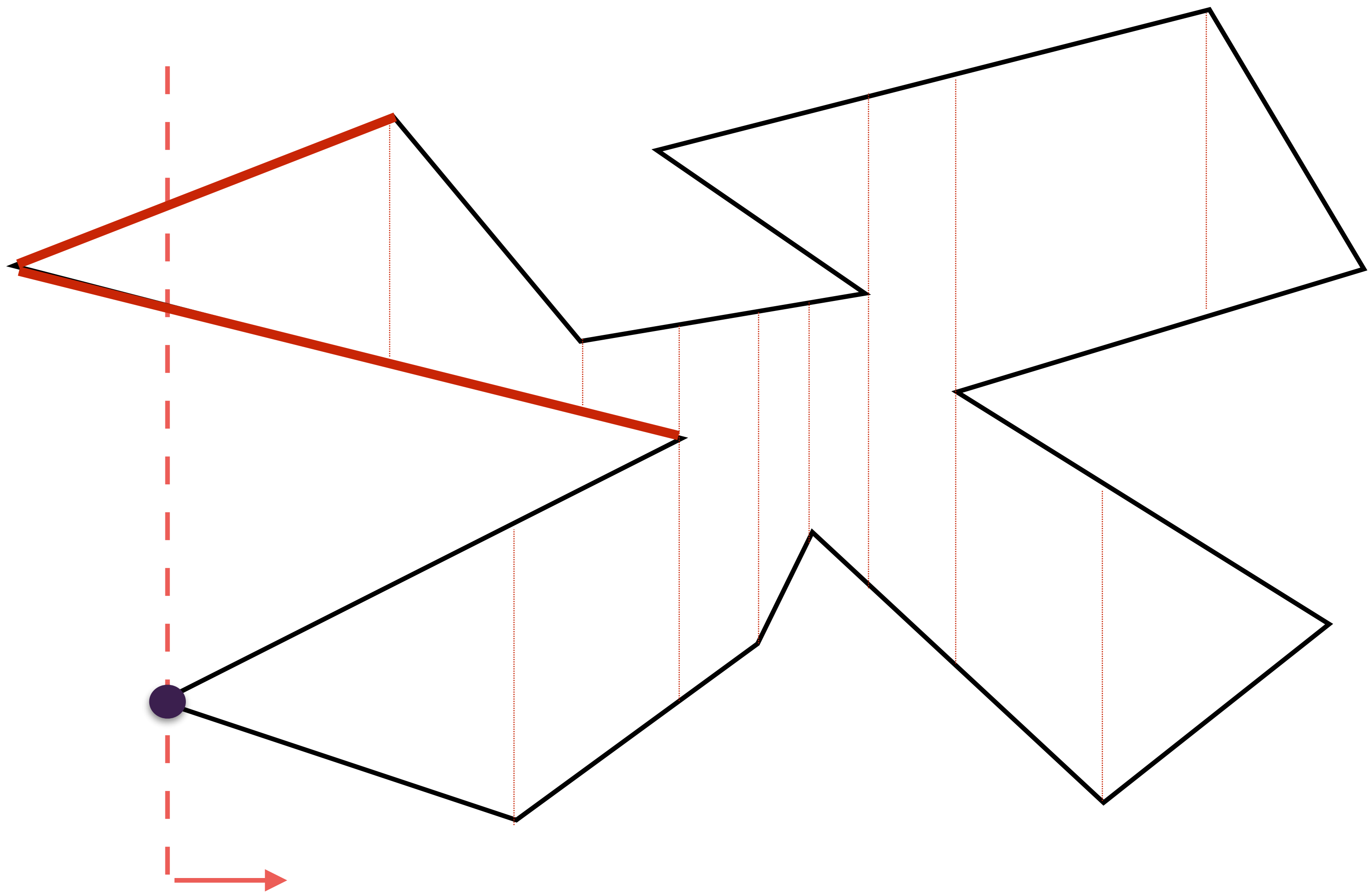
Computing the trapezoid partition in O(n lg n)

- Plane sweep



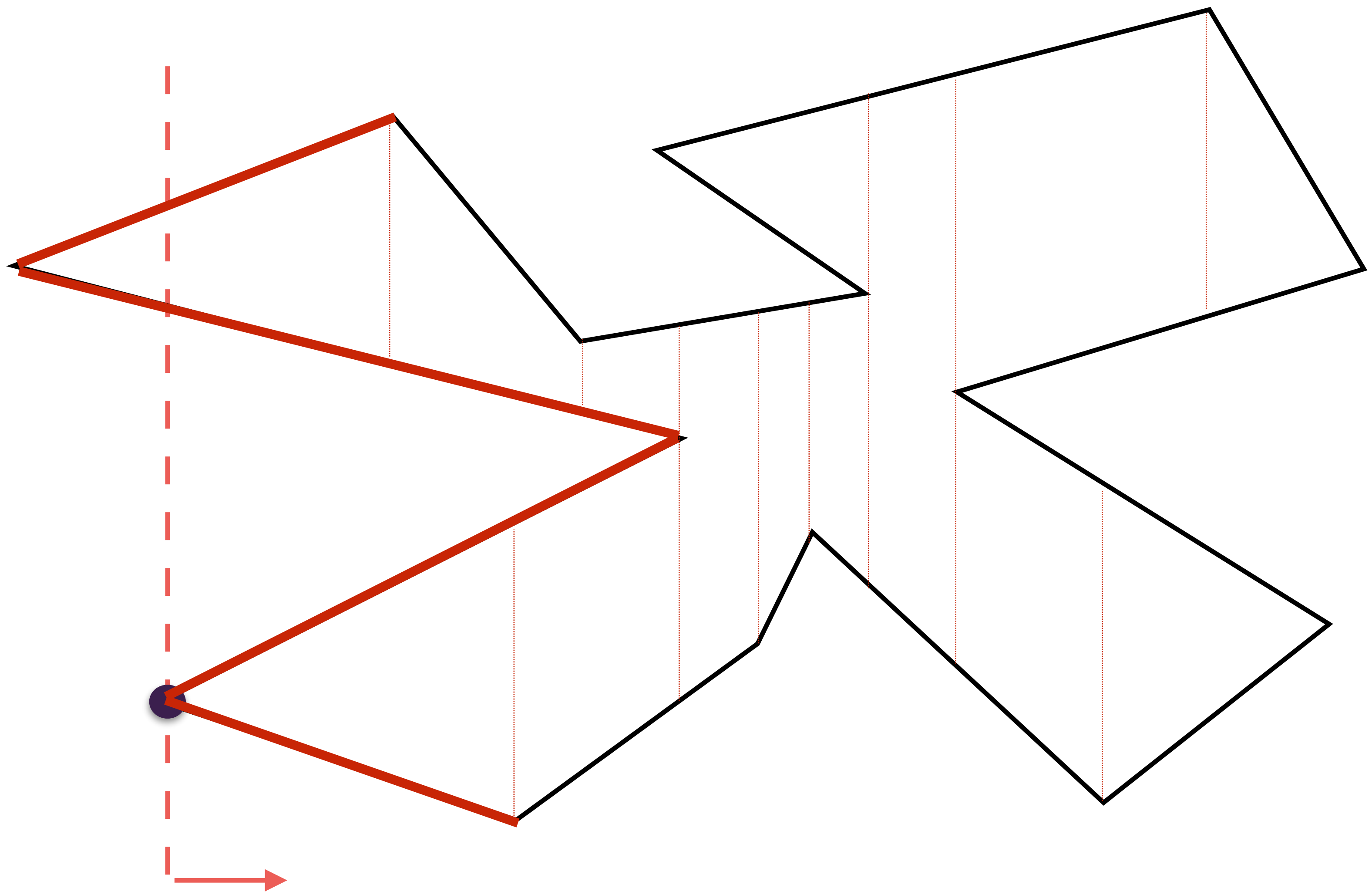
Computing the trapezoid partition in O(n lg n)

- Plane sweep



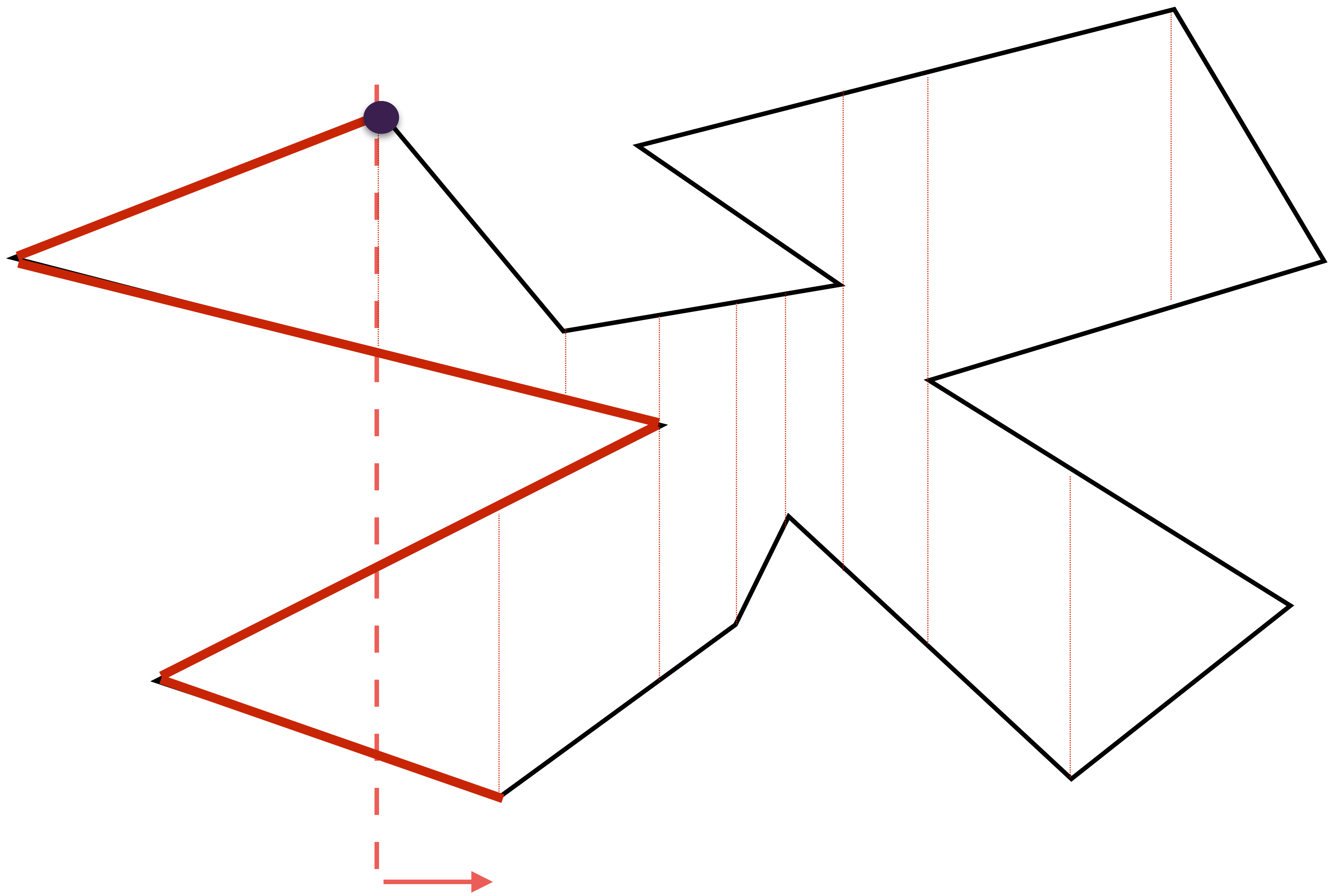
Computing the trapezoid partition in O(n lg n)

- Plane sweep



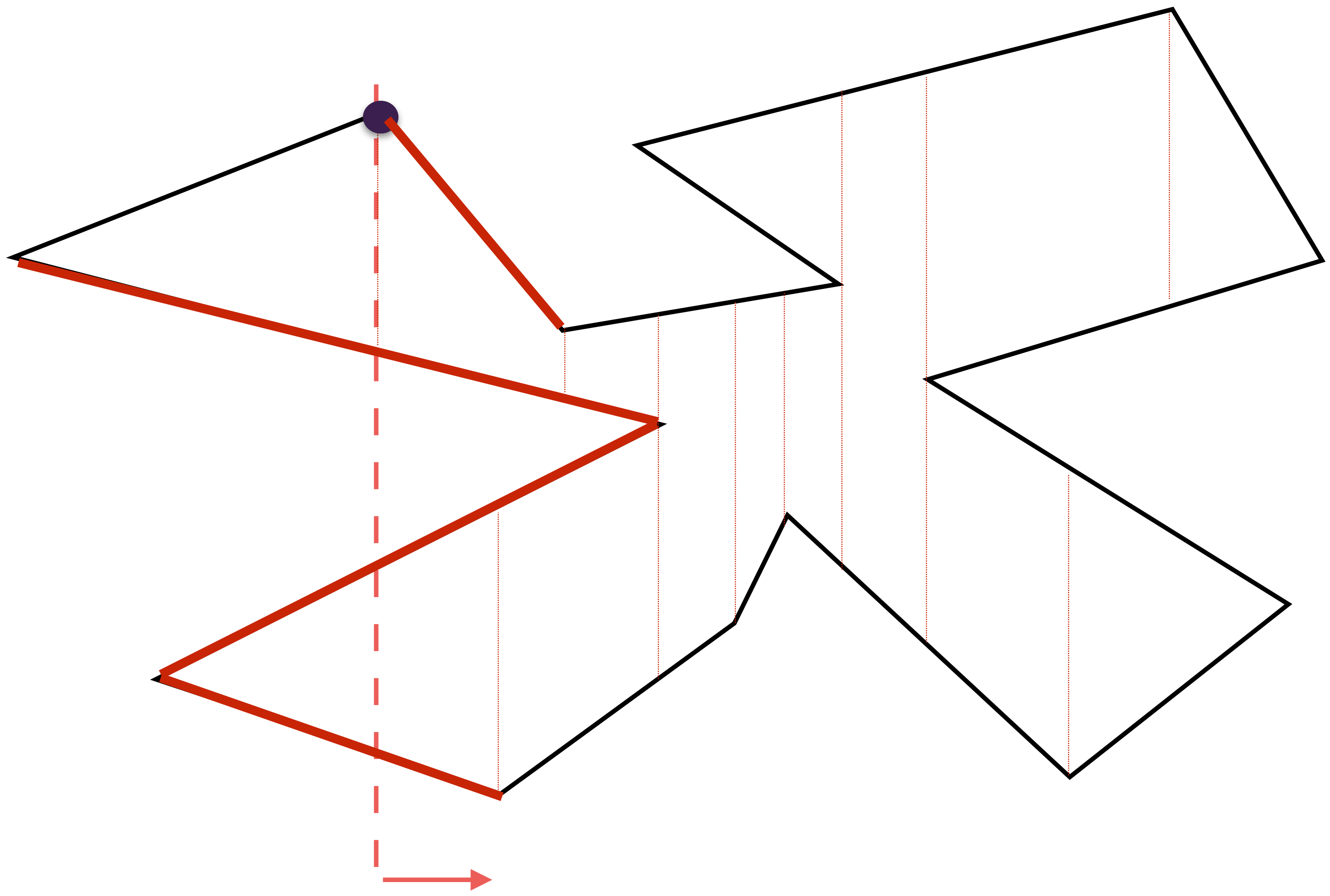
Computing the trapezoid partition in O(n lg n)

- Plane sweep



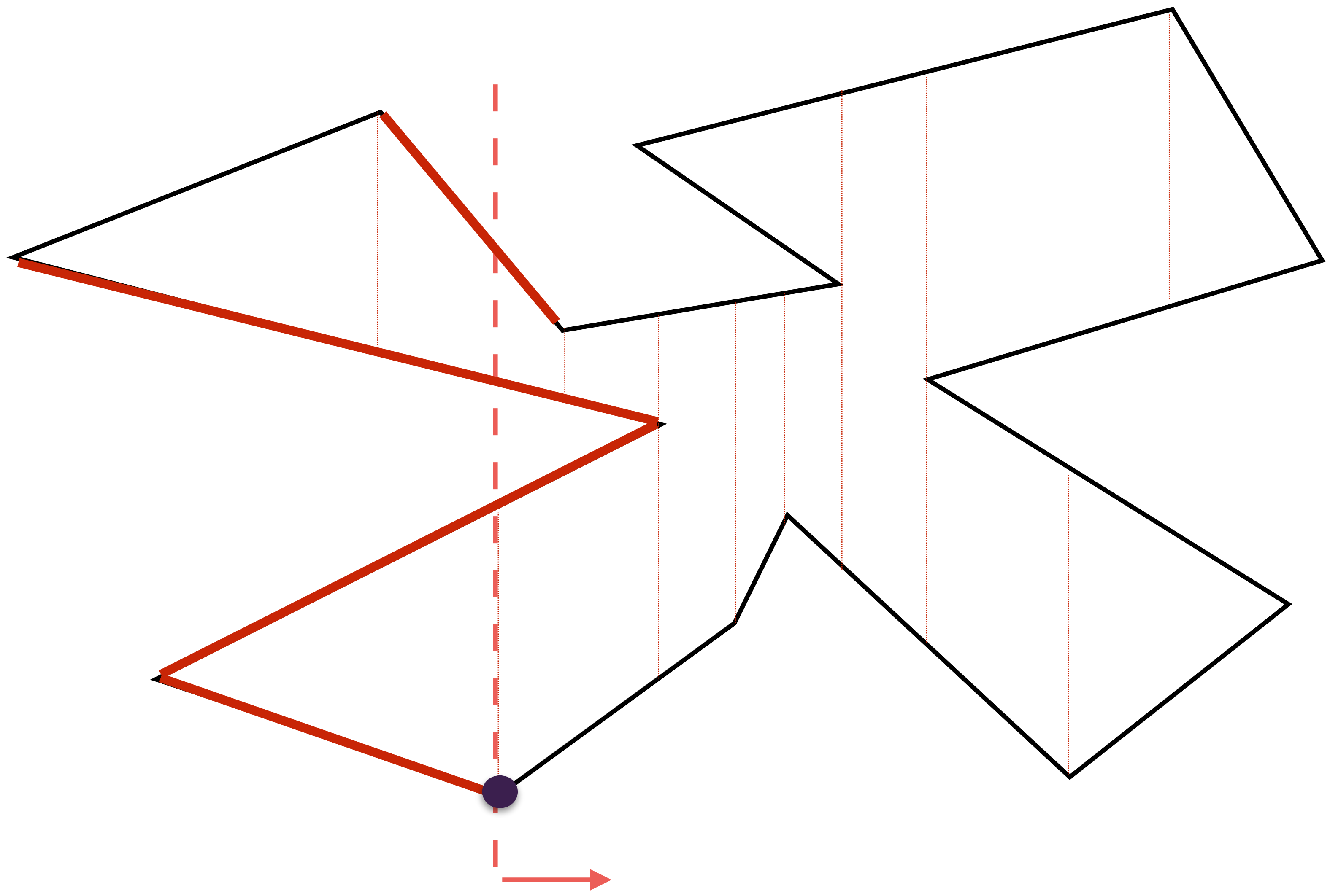
Computing the trapezoid partition in O(n lg n)

- Plane sweep



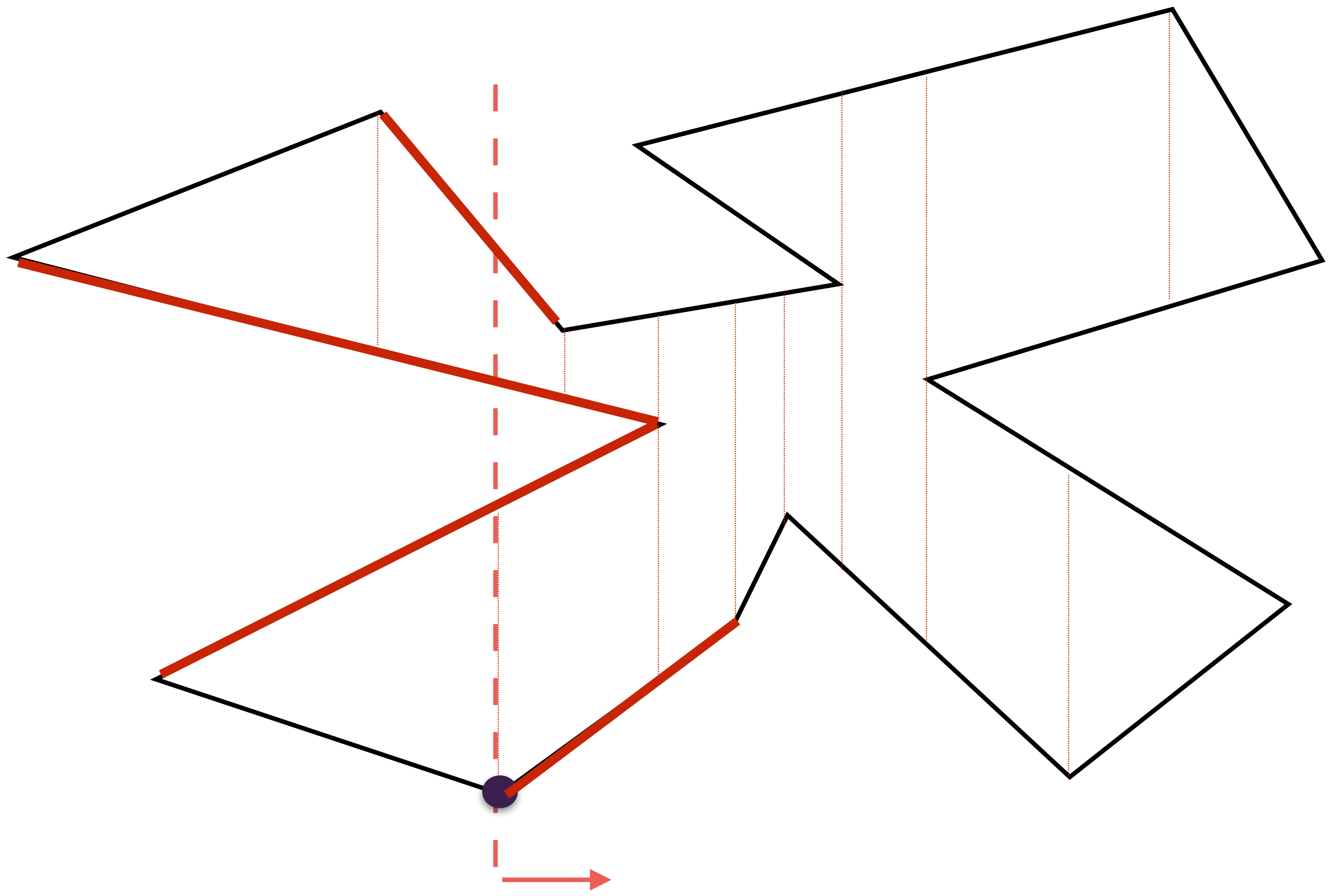
Computing the trapezoid partition in O(n lg n)

- Plane sweep



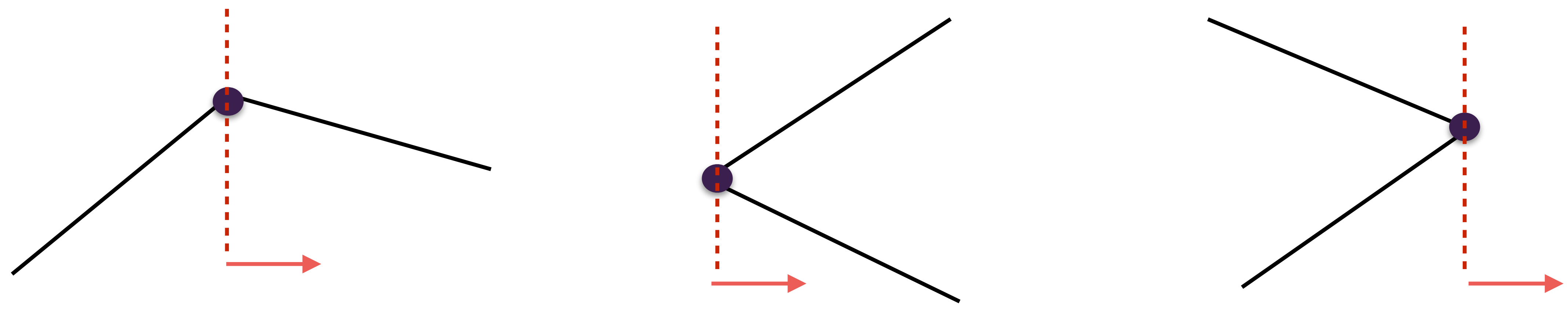
Computing the trapezoid partition in O(n lg n)

- Plane sweep



Computing the trapezoid partition in O(n lg n)

- Plane sweep
- Events: polygon vertices
- Status structure: edges that intersect current sweep line, in y-order
- Events:



How do we determine the trapezoids?

Computing the trapezoid partition in O(n lg n)

- Algorithm