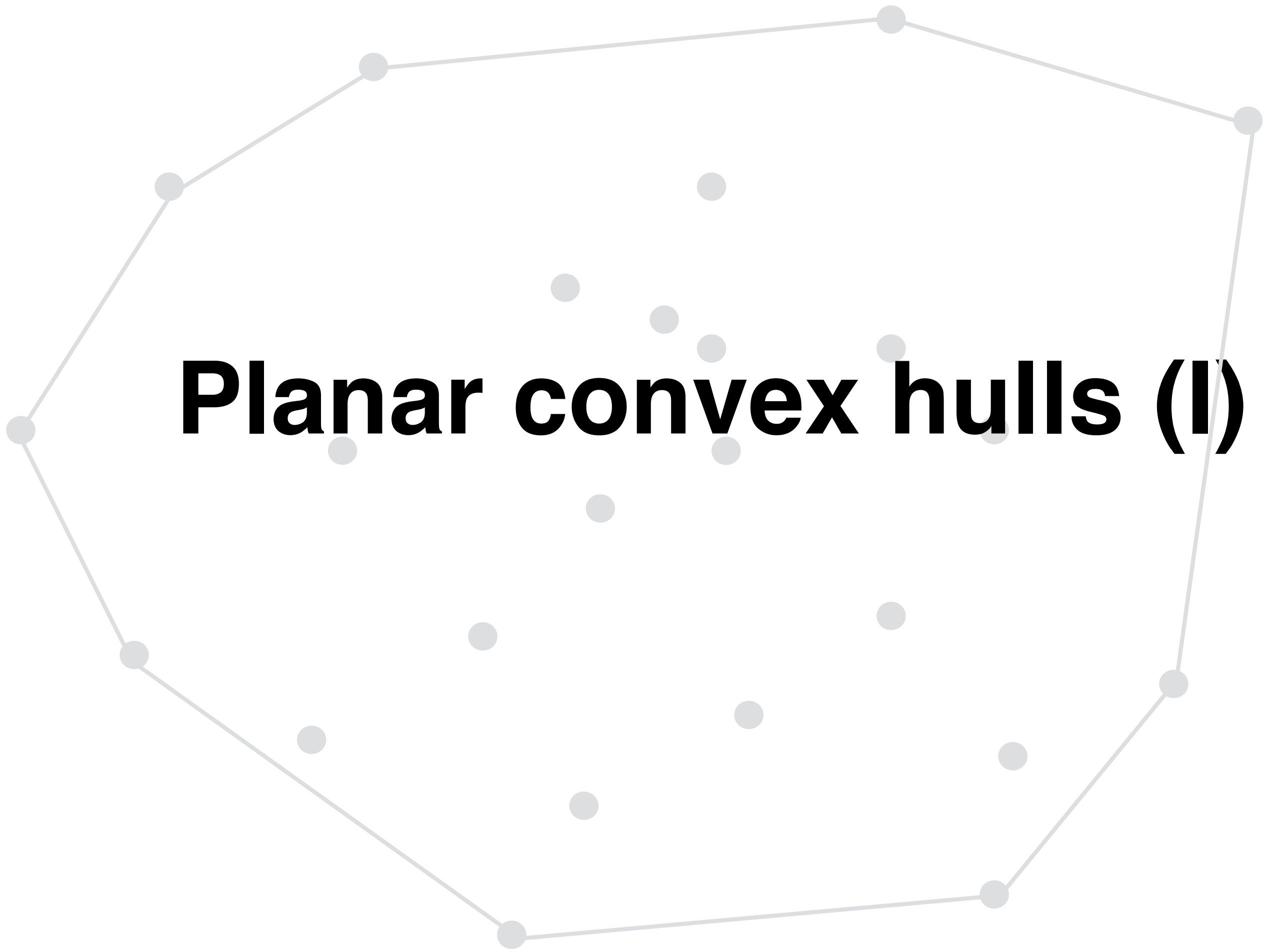
Planar convex hulls (I)



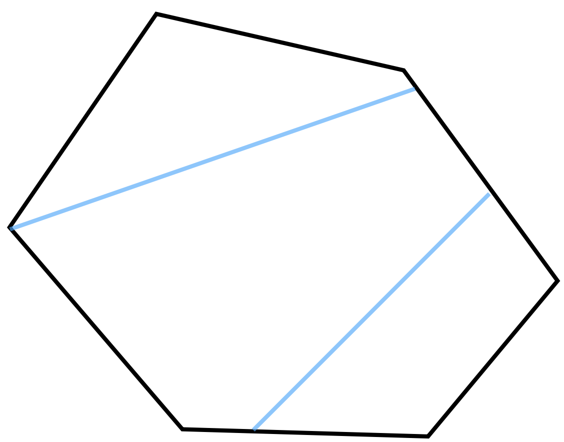
Computational Geometry [csci 3250]
Laura Toma
Bowdoin College

Outline

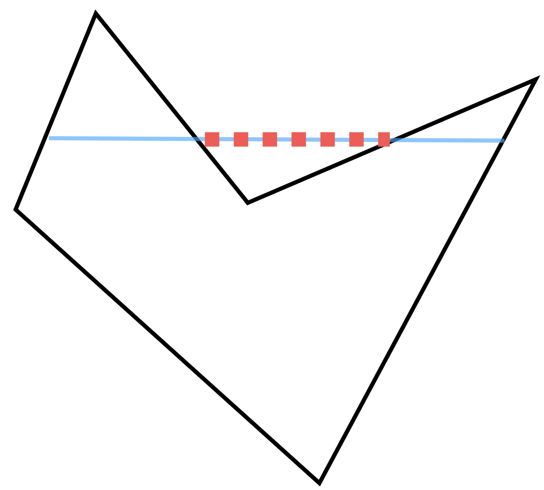
- Definition and properties
- Algorithms for computing the convex hull
    - Brute force
    - Gift wrapping
- Next times
    - Quickhull
    - Graham scan
    - Andrew's monotone chain
    - Incremental hull
    - Divide-and -conquer hull
    - Lower bound

Convexity

A polygon P is **convex** if for any p, q in P, the segment pq lies entirely in P.



convex

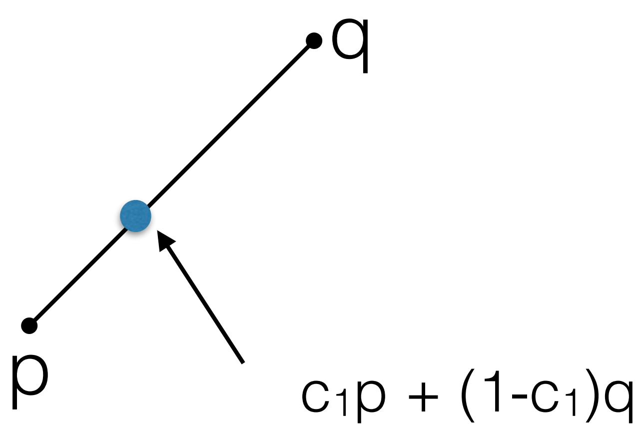


non-convex

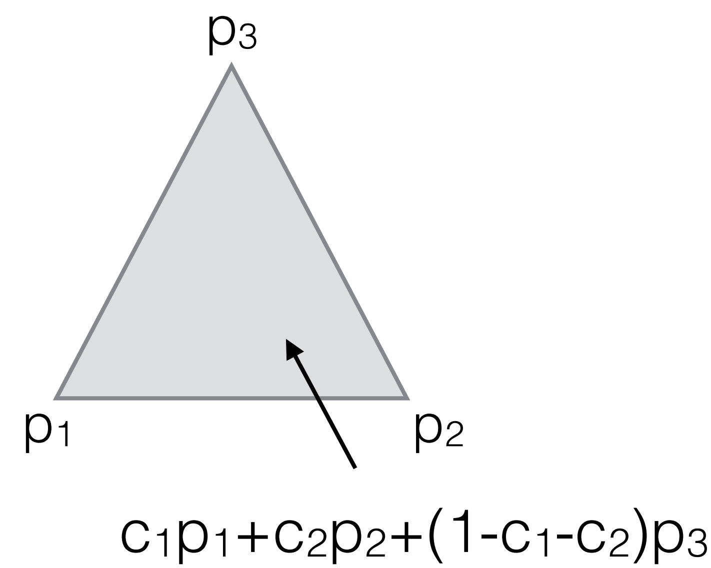
Convexity: algebraic view

- A **convex combination** of points p_1, p_2, …p_k is a point of the form

$$c_1 p_1 + c_2 p_2 + \ldots + c_k p_k \text{ with } c_i \in [0,1], c_1 + c_2 + \ldots + c_k = 1$$



a segment consists of all convex combinations of its 2 vertices

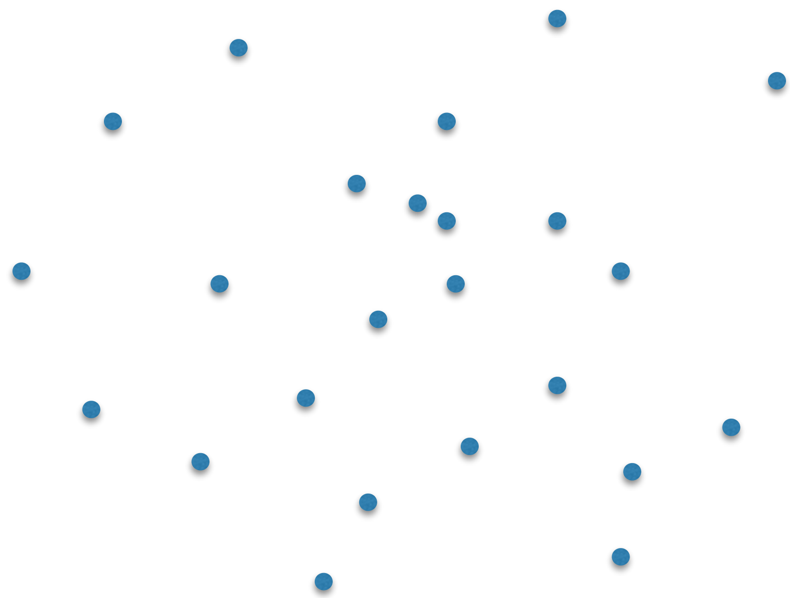


a triangle consists of all convex combinations of its 3 vertices

- The convex hull CH(P) = all convex combinations of points in P

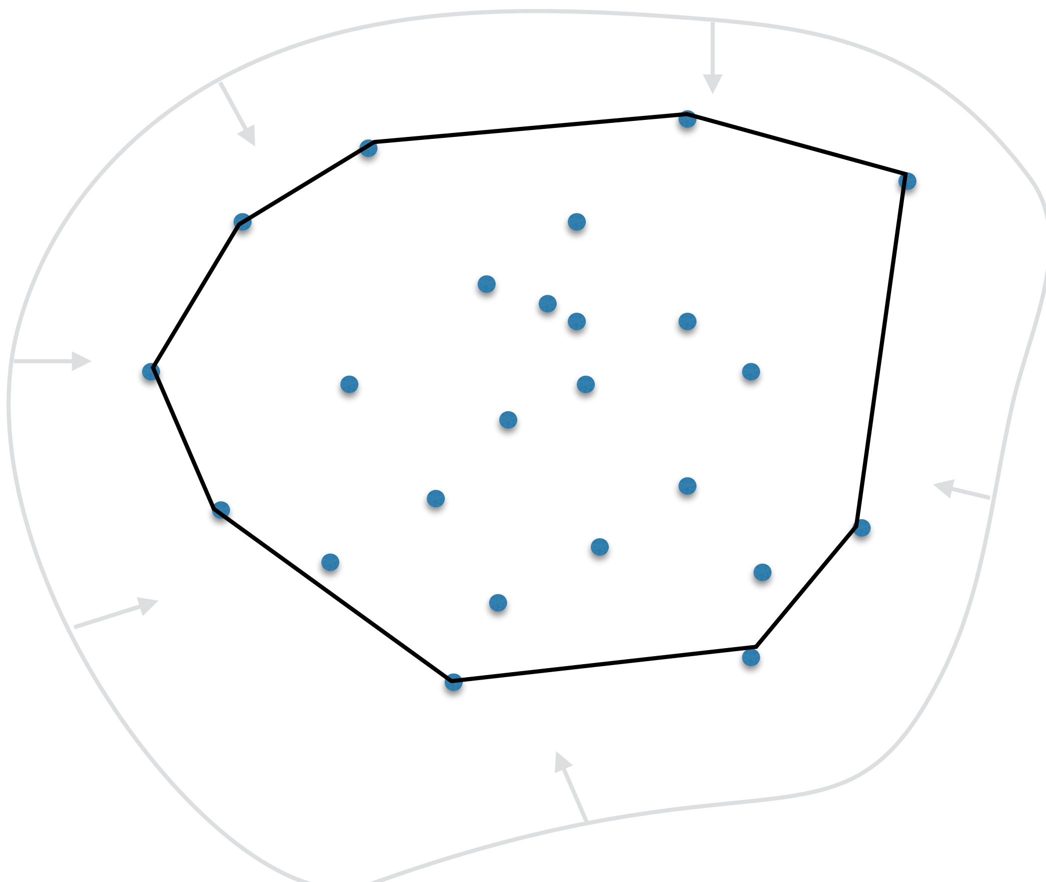
Convex Hull

Given a set P of points in 2D, their convex hull is the smallest convex polygon that contains all points of P



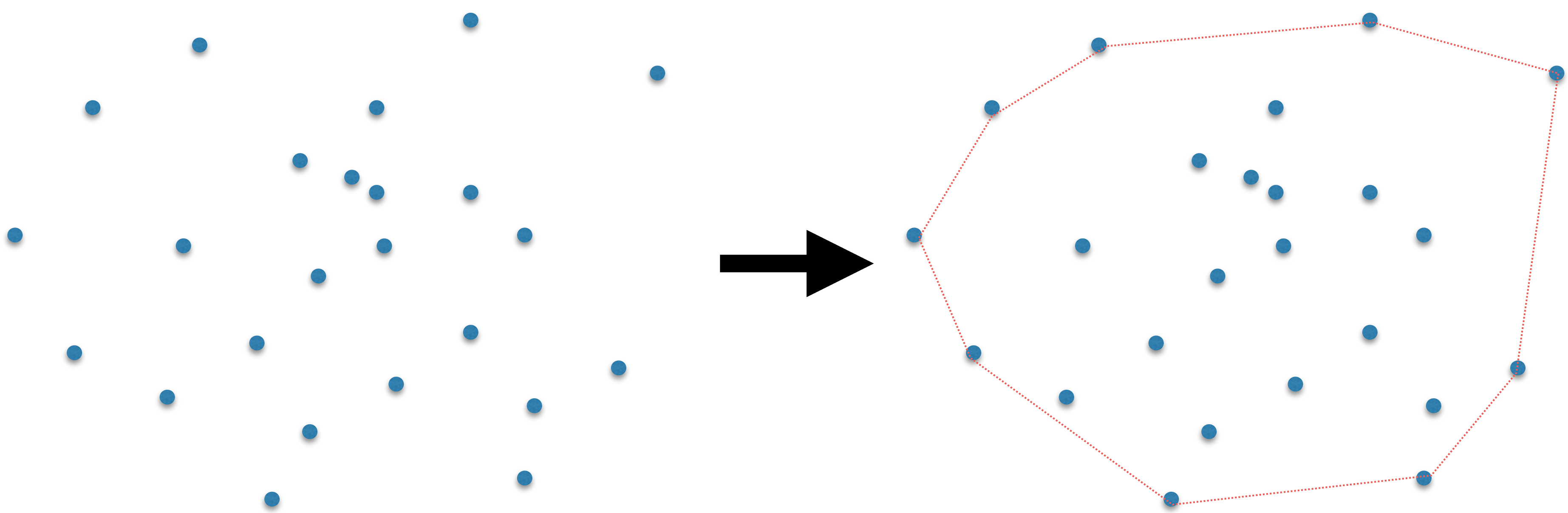
Convex Hull

Given a set P of points in 2D, their convex hull is the smallest convex polygon that contains all points of P



Compute the Convex Hull

Given a set P of points in 2D, describe an algorithm to compute their convex hull



Input:
array P of points (in 2D)

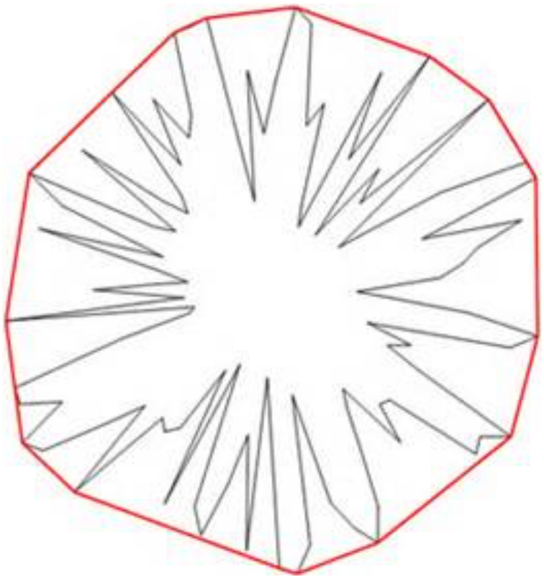
Output:
array/list of points on the CH (in boundary order)

Convex Hull

- One of the first problems studied in CG
- Many solutions
  - simple, elegant, intuitive
  - illustrate techniques for geometrical algorithms
- Used in many applications
  - robotics, path planning, partitioning problems, shape recognition, separation problems, etc

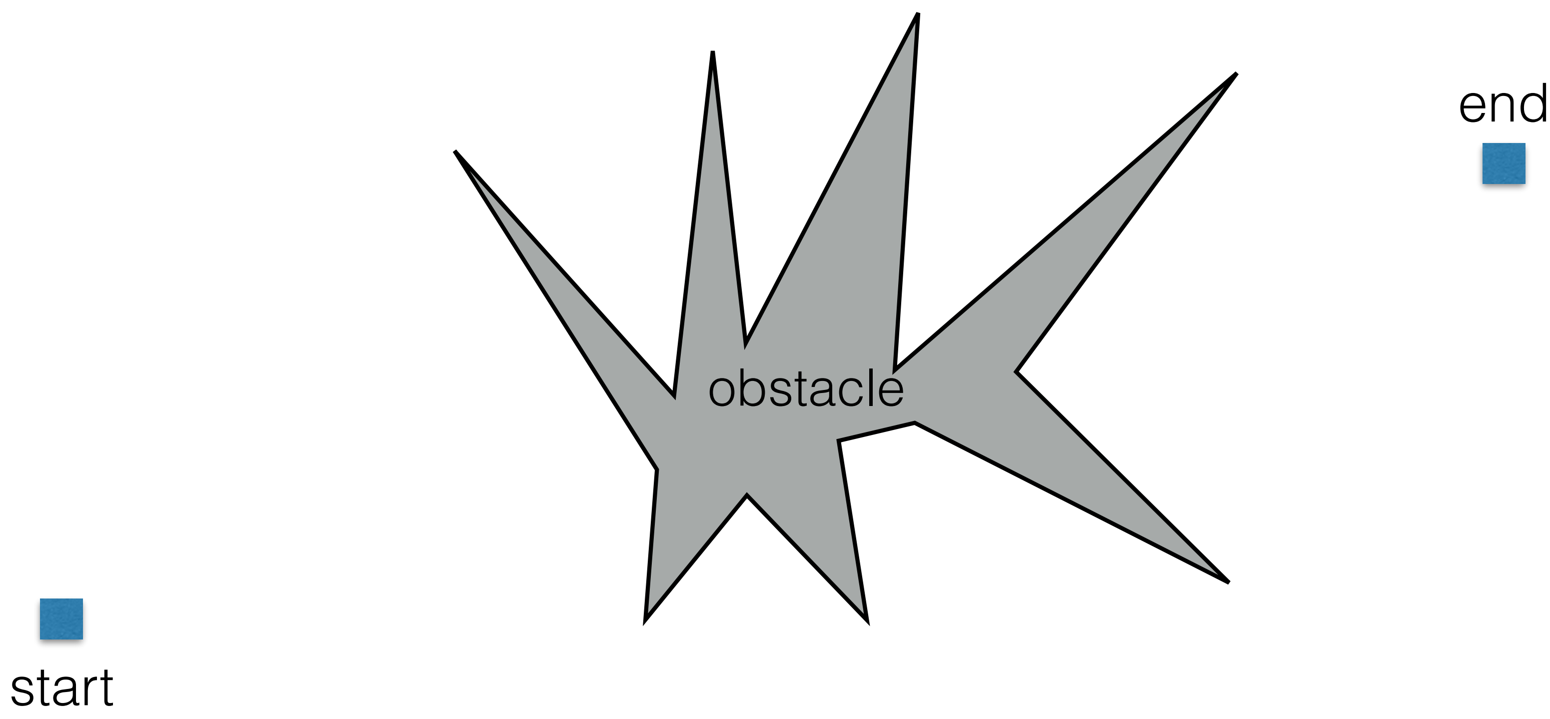
Applications

- Shape analysis, matching, recognition
  - approximate objects by their CH



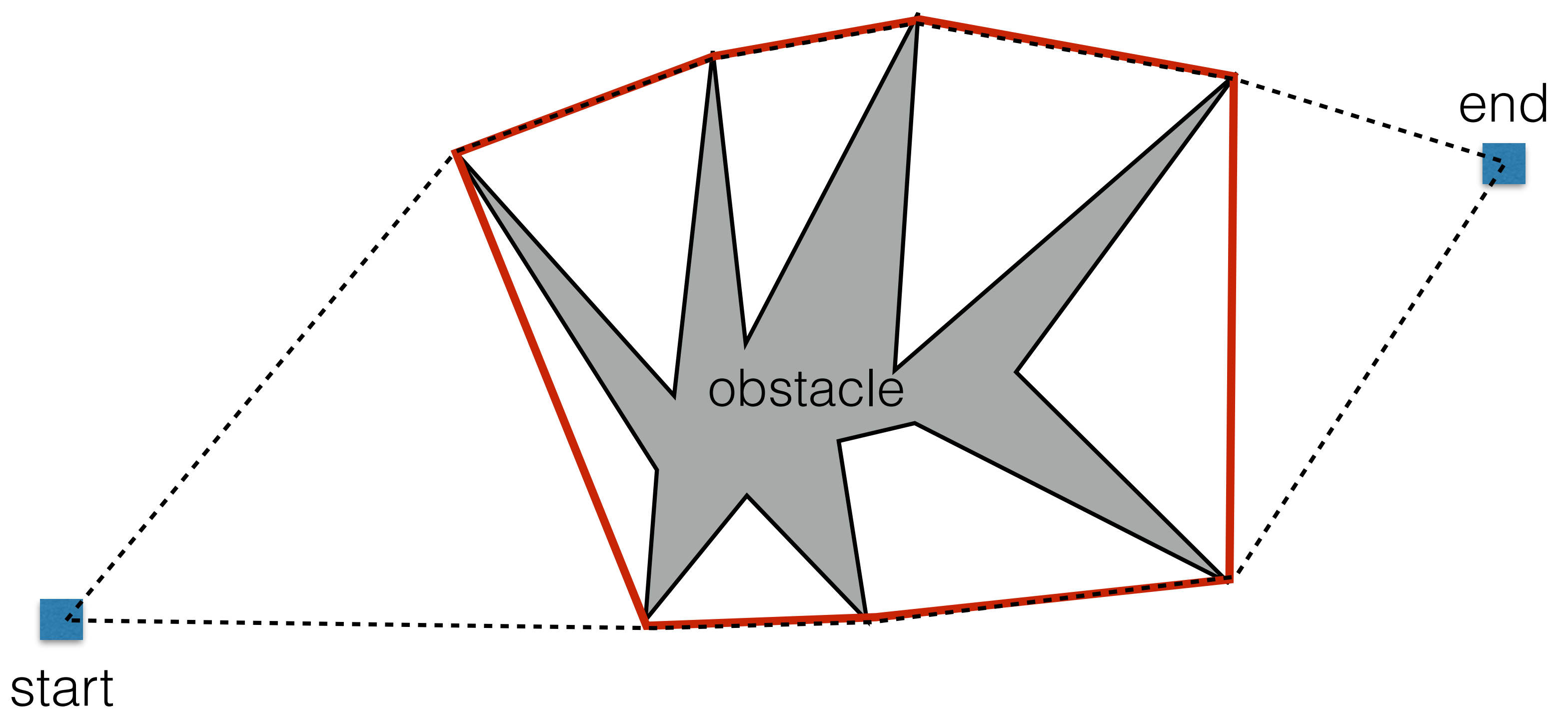
Applications

- Path planning: find (shortest) collision-free path from start to end



Applications

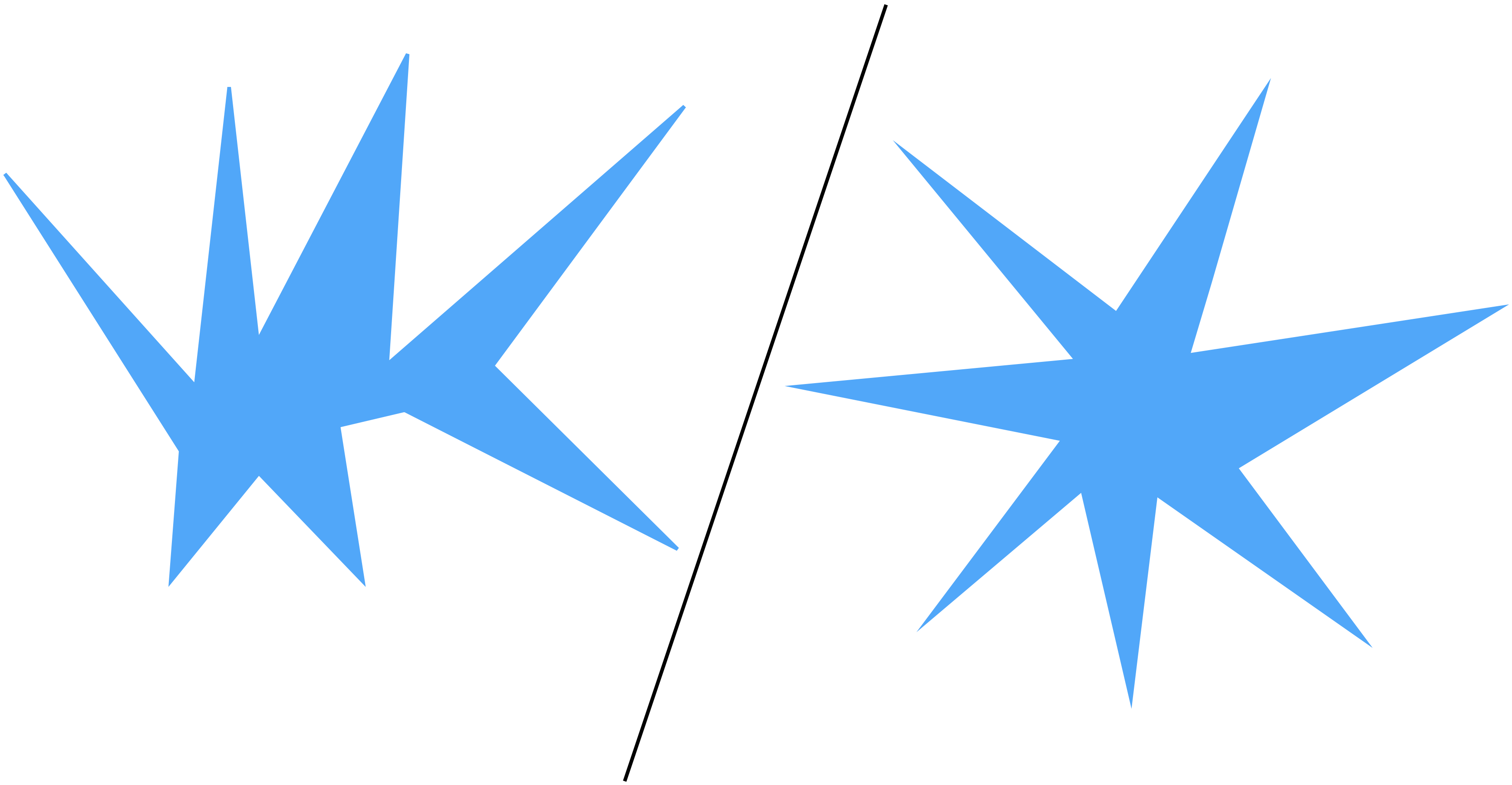
- Path planning: find (shortest) collision-free path from start to end



- The shortest path follows the CH(obstacle)
    - it is the shorter of the upper path and lower path

Applications

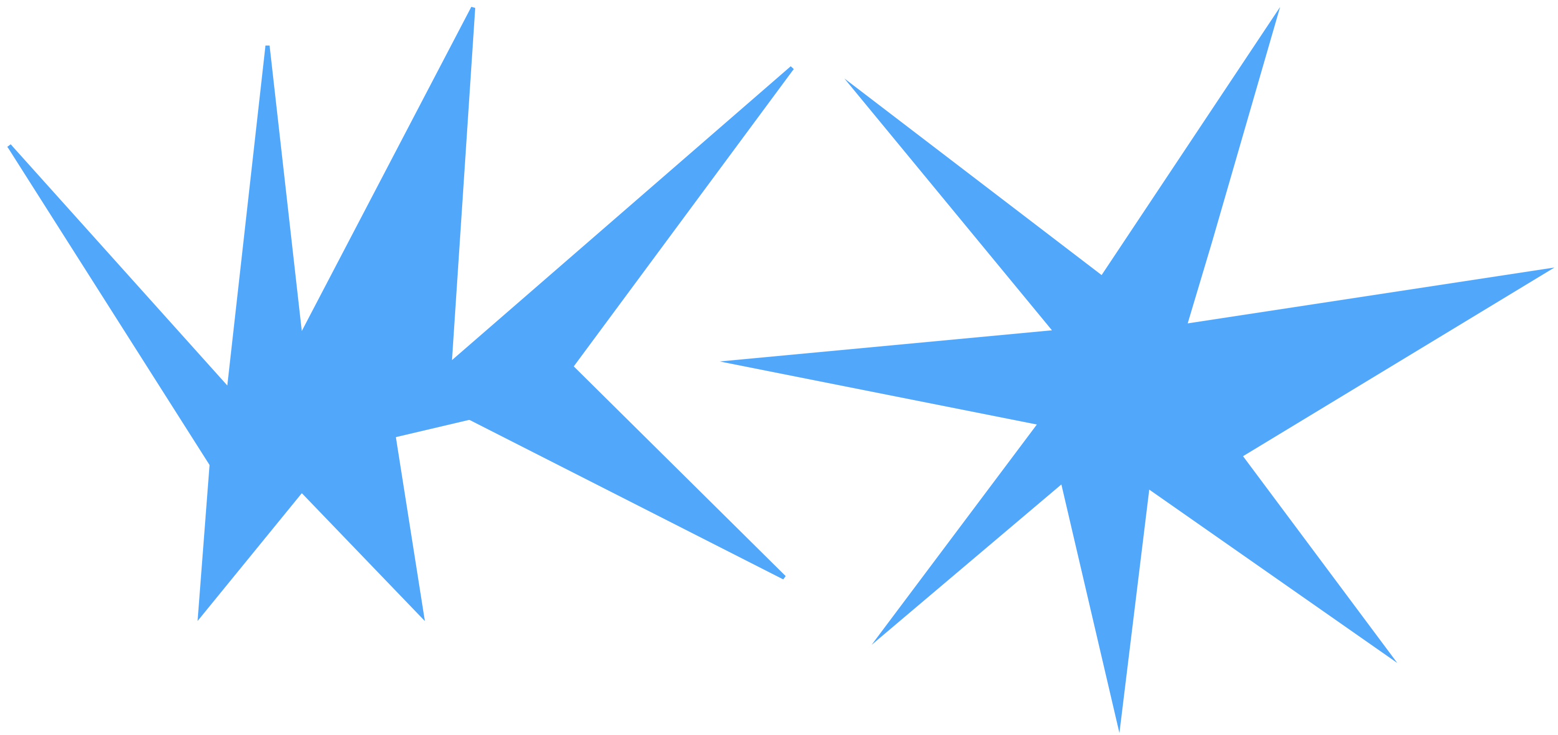
- Partitioning problems
  - does there exist a line separating two objects?



YES

Applications

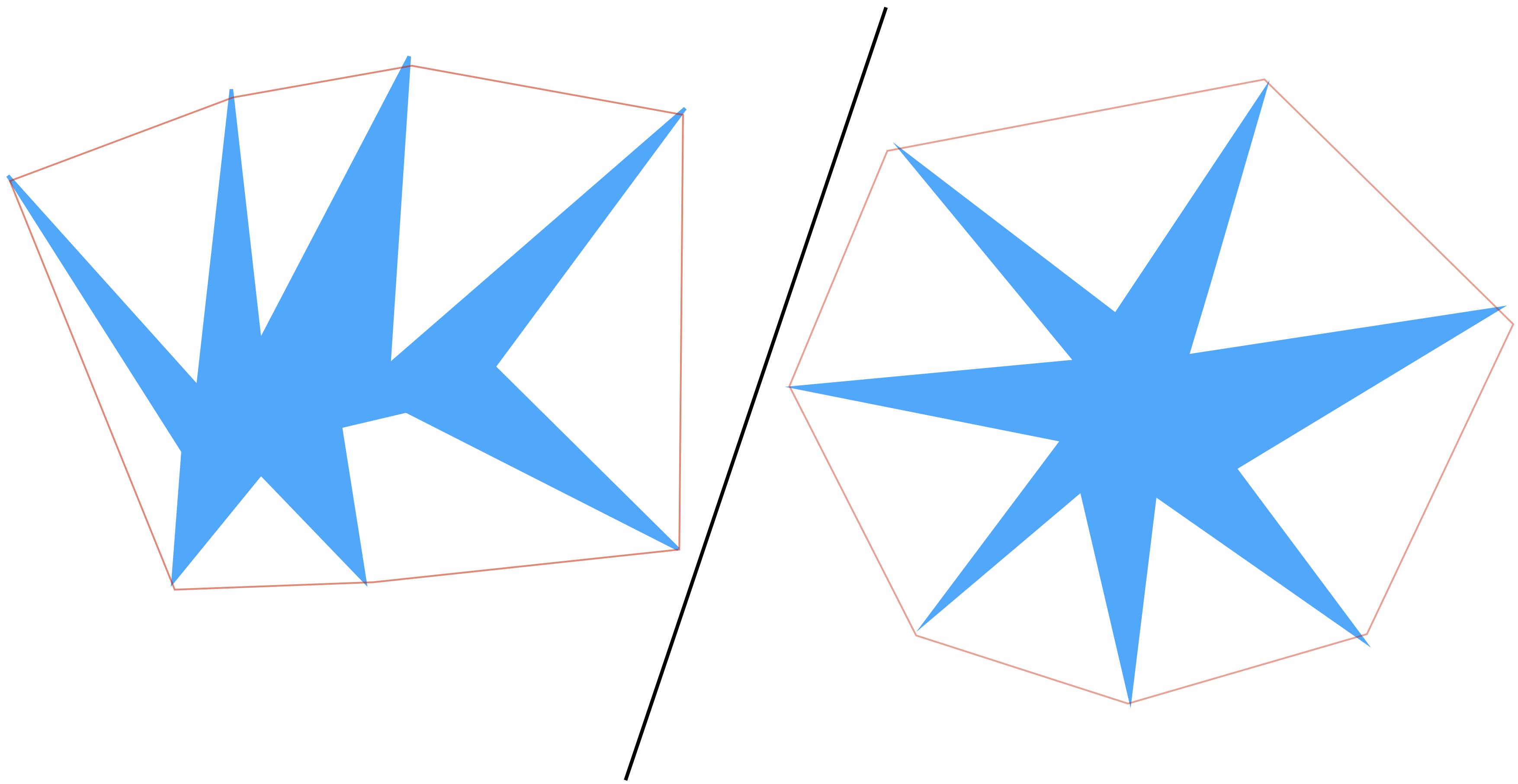
- Partitioning problems
  - does there exist a line separating two objects?



NO

Applications

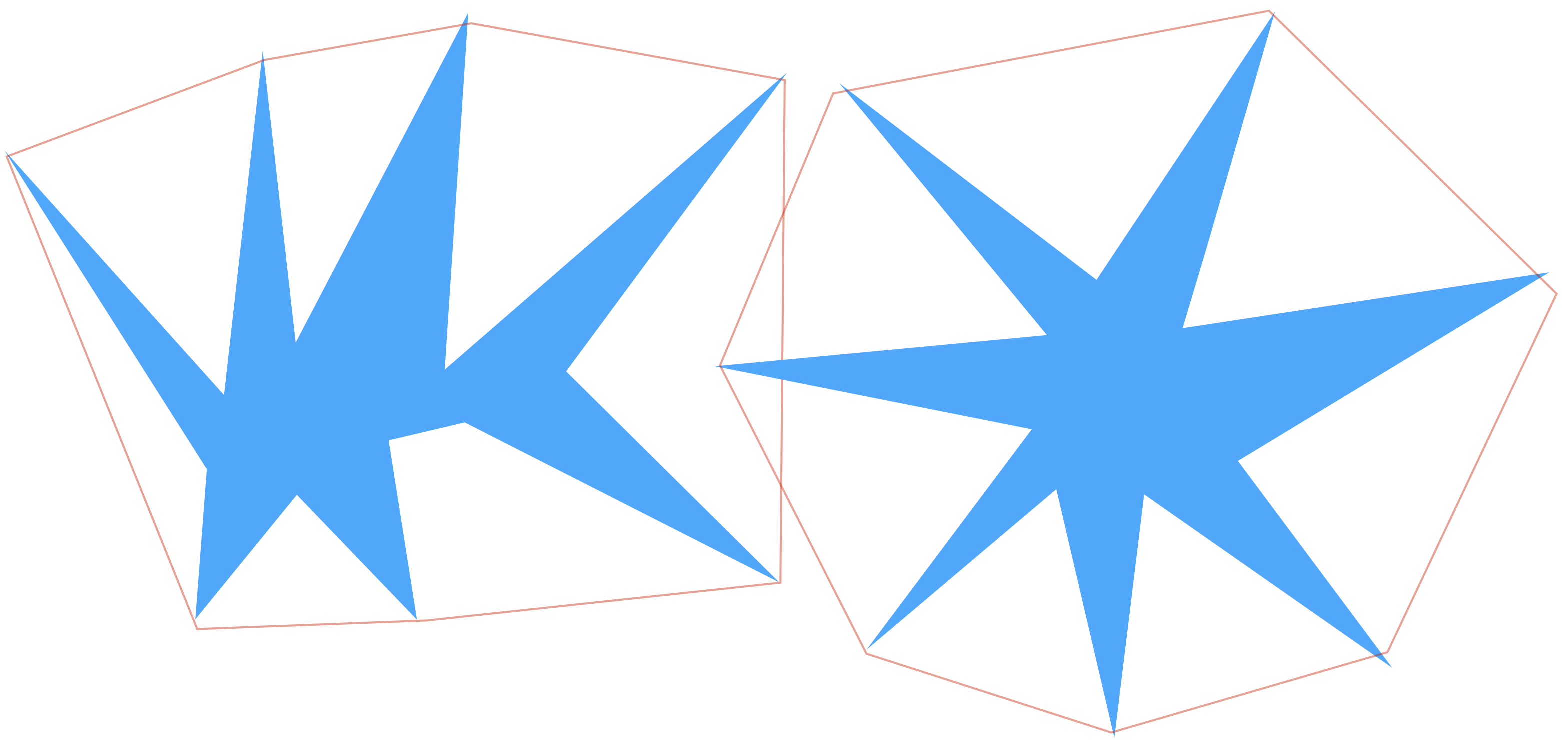
- Partitioning problems
  - does there exist a line separating two objects?



YES

Applications

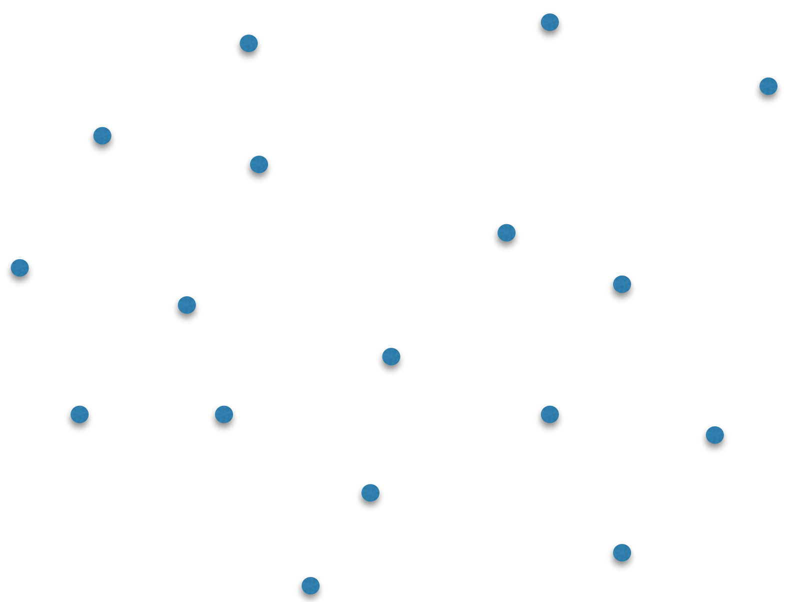
- Partitioning problems
  - does there exist a line separating two objects?



NO

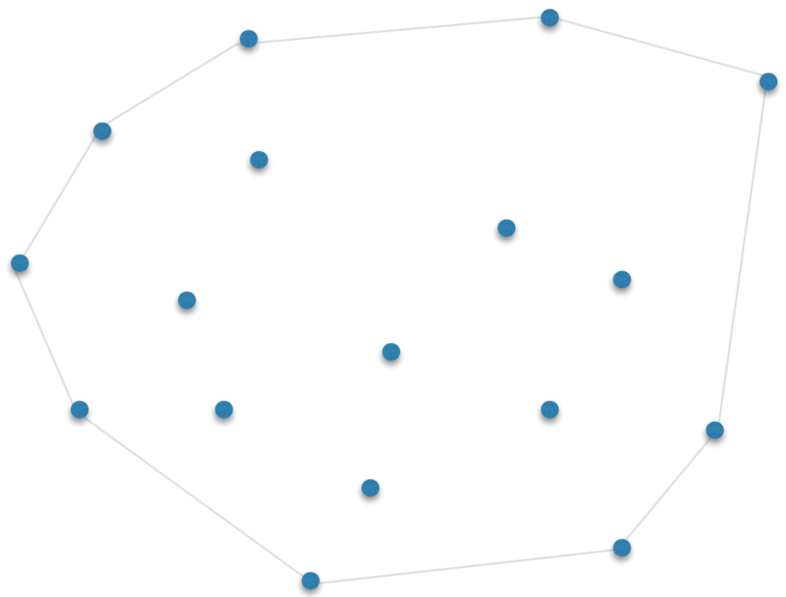
Applications

- Find the two points in P that are farthest away

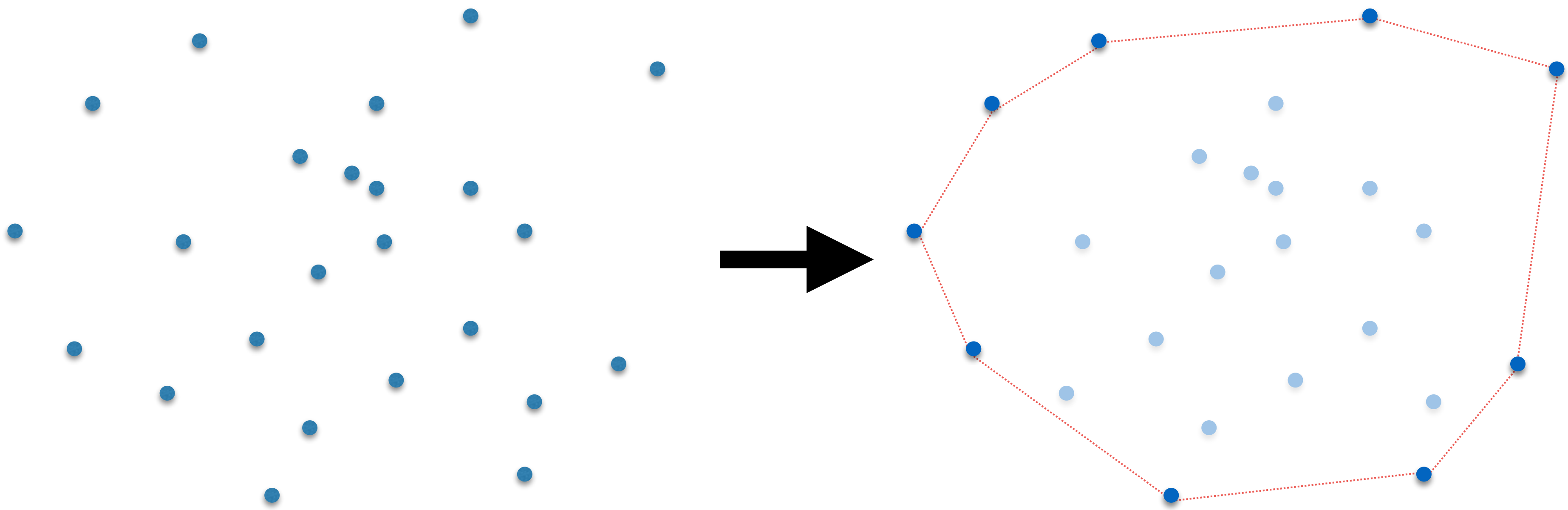


Applications

- Find the two points in P that are farthest away



So, we want to compute the convex hull



Input:
array P of points (in 2D)

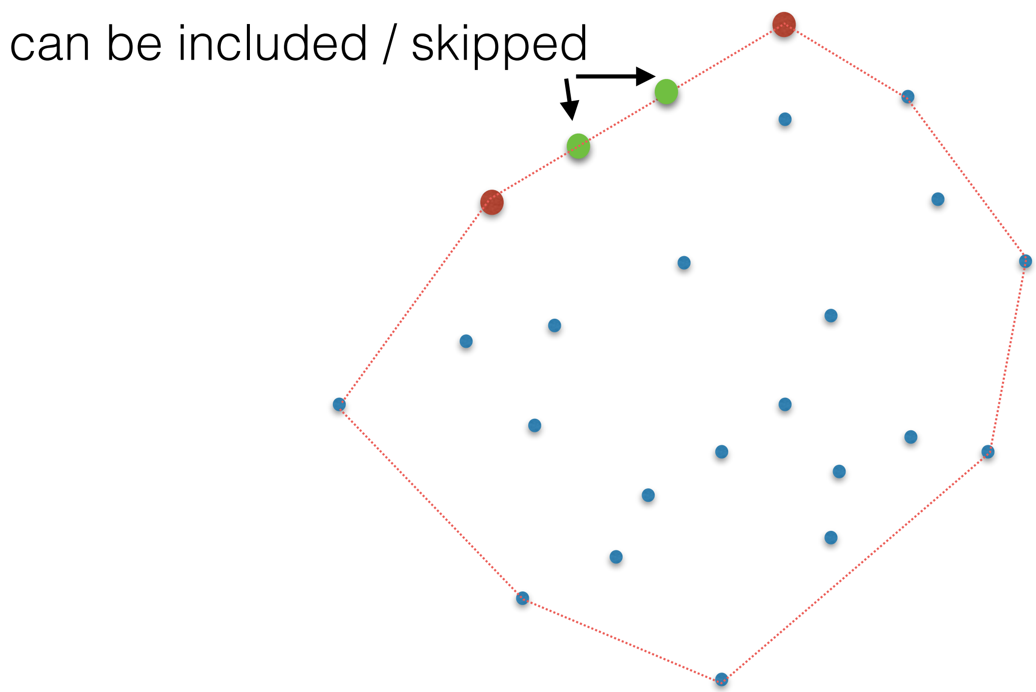
Output:
list of points on the CH (in boundary order)

Convex Hull Variants

Several types of convex hull output are conceivable

- **all** points on the hull
- **only non-collinear** points

- in **boundary** order
- in **arbitrary** order

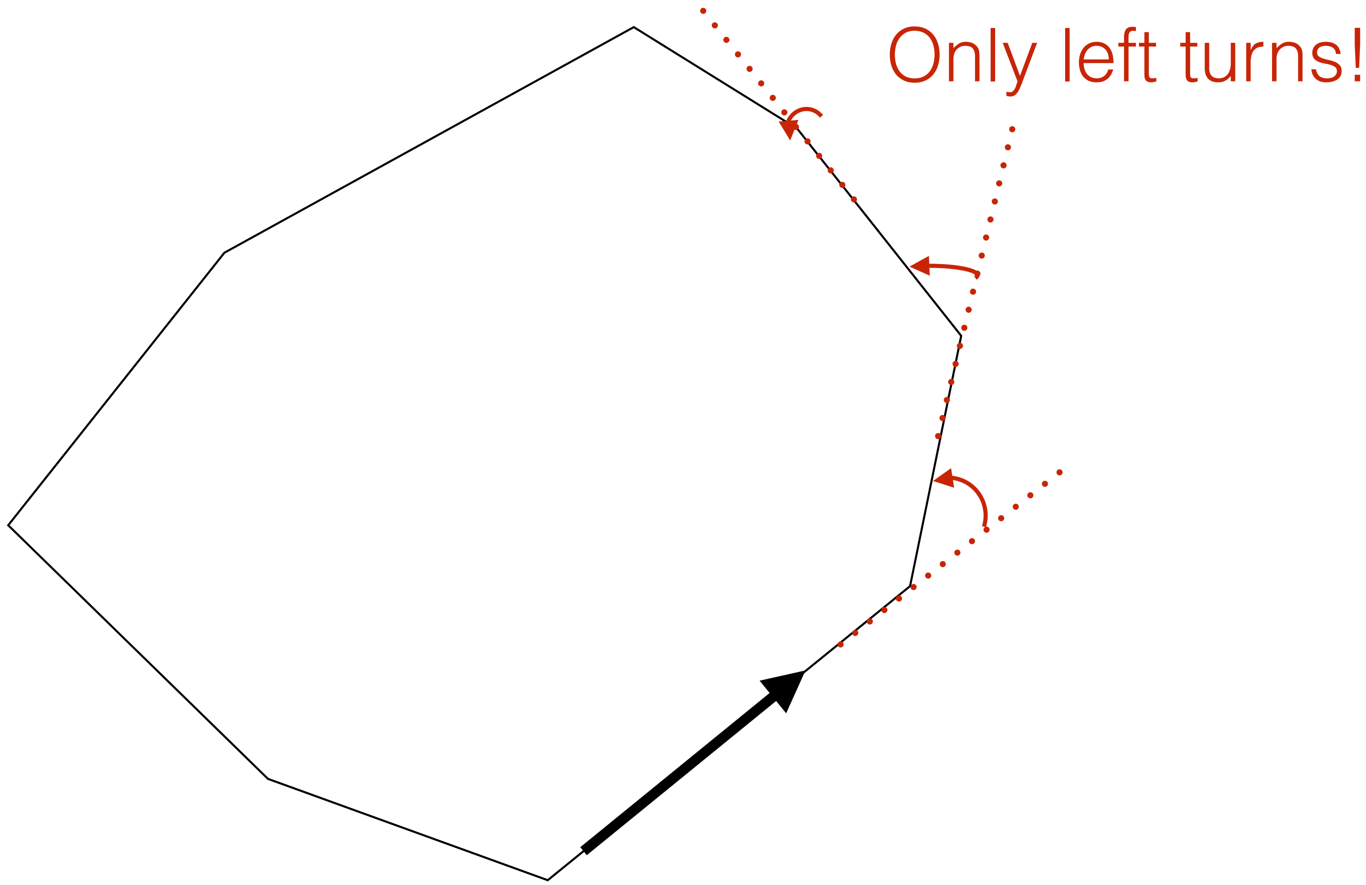


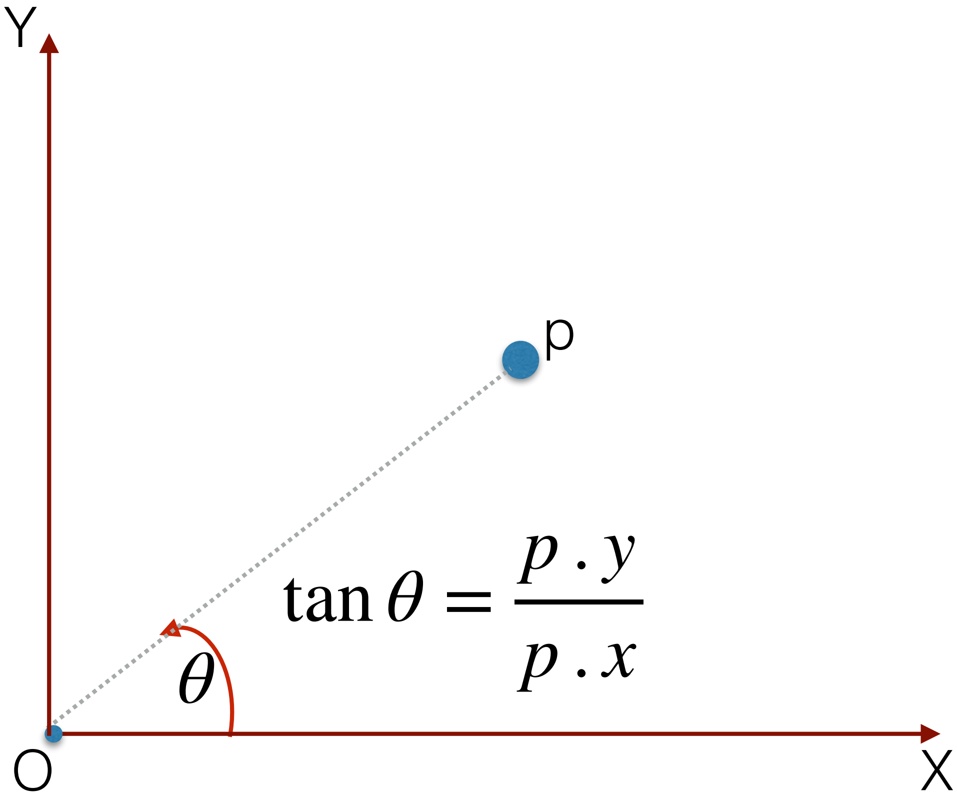
- It may seem that computing in boundary order is harder. It is known that identifying th epoints on th eCH has a lower bound of $\Omega(n \lg n)$. Therefore sorting is not the bottleneck.

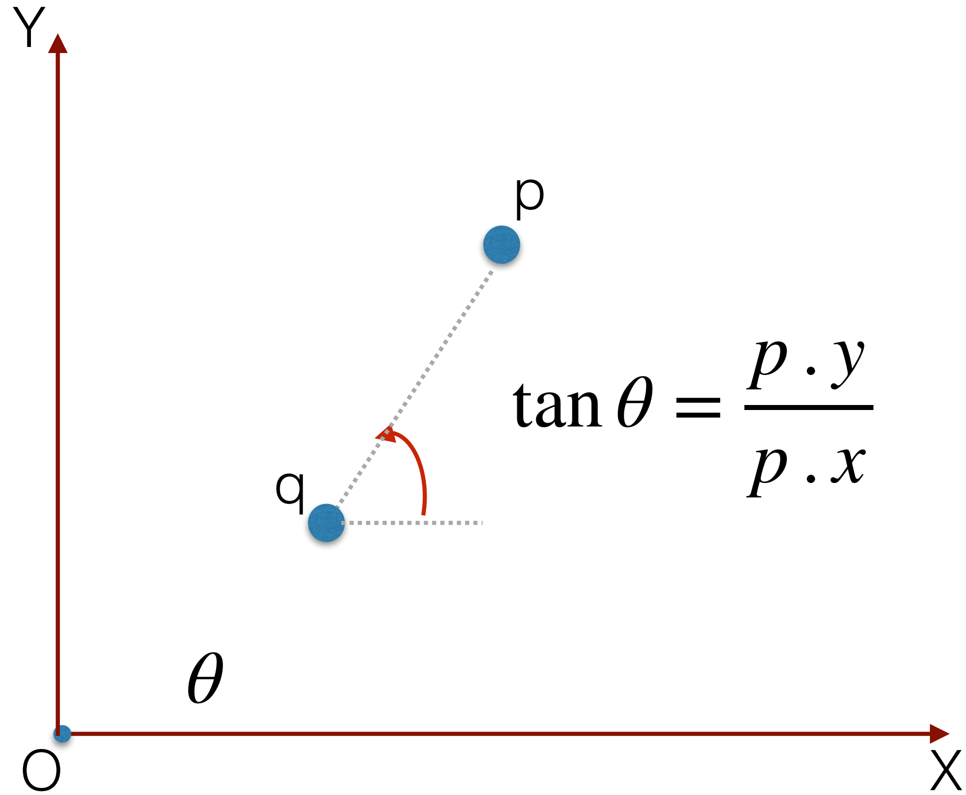
Convex Hull:

Some basic properties

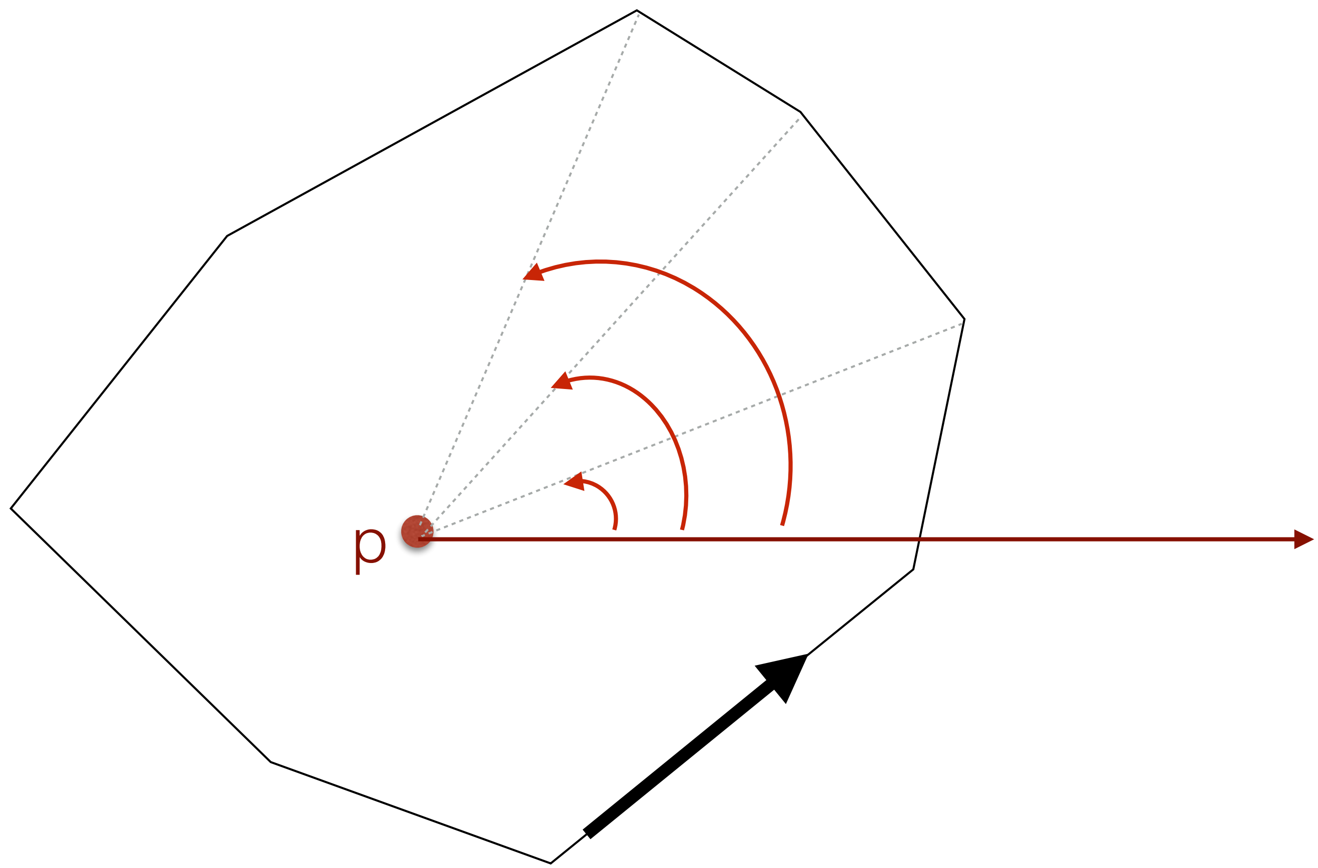
Walk ccw along the boundary of a convex polygon




Y
p
$\tan\theta = \frac{p.y}{p.x}$
θ
O
X

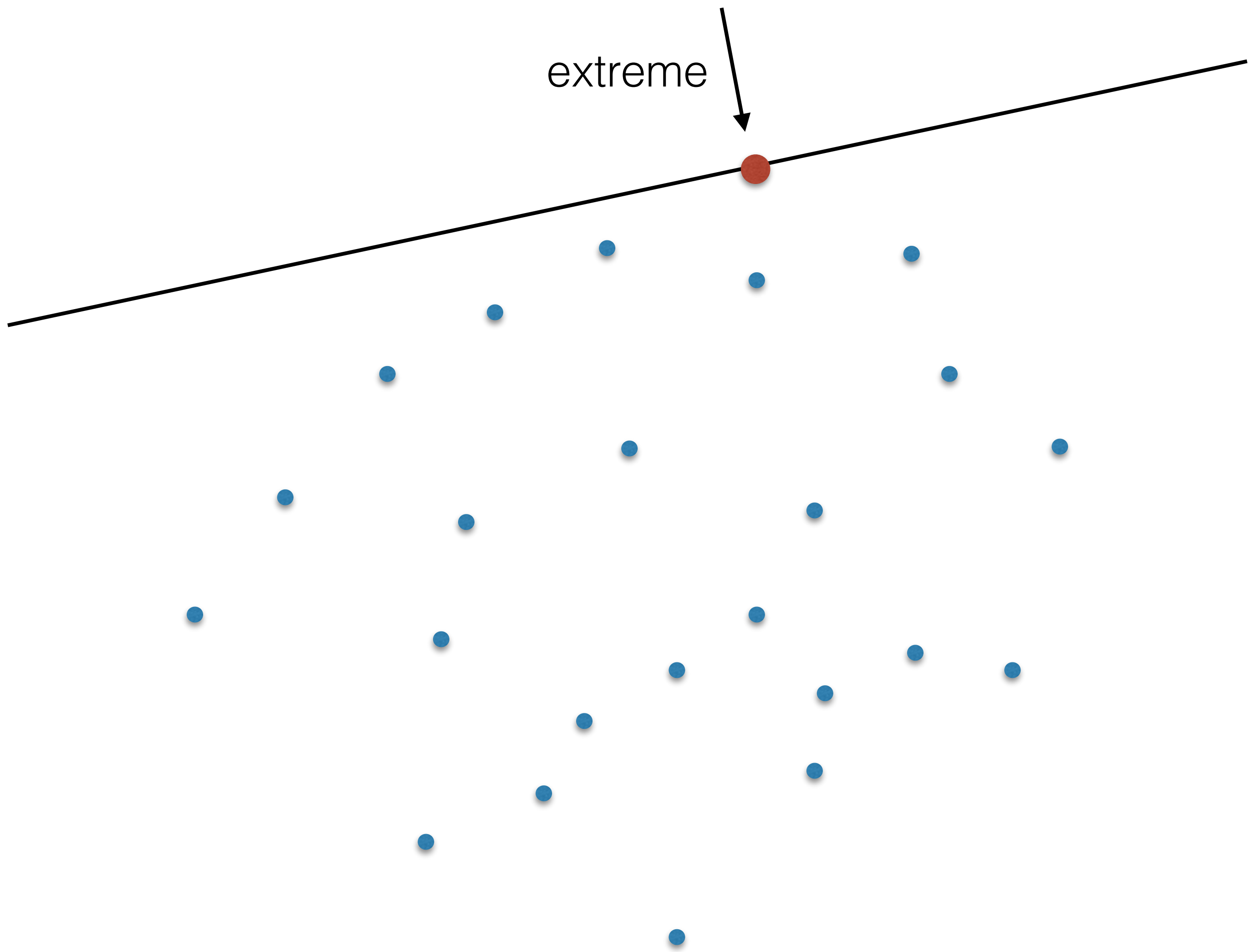

Y
p
$\tan\theta = \frac{p.y}{p.x}$
q
θ
O
X

For any point p inside, the points on the boundary are in radial order around p



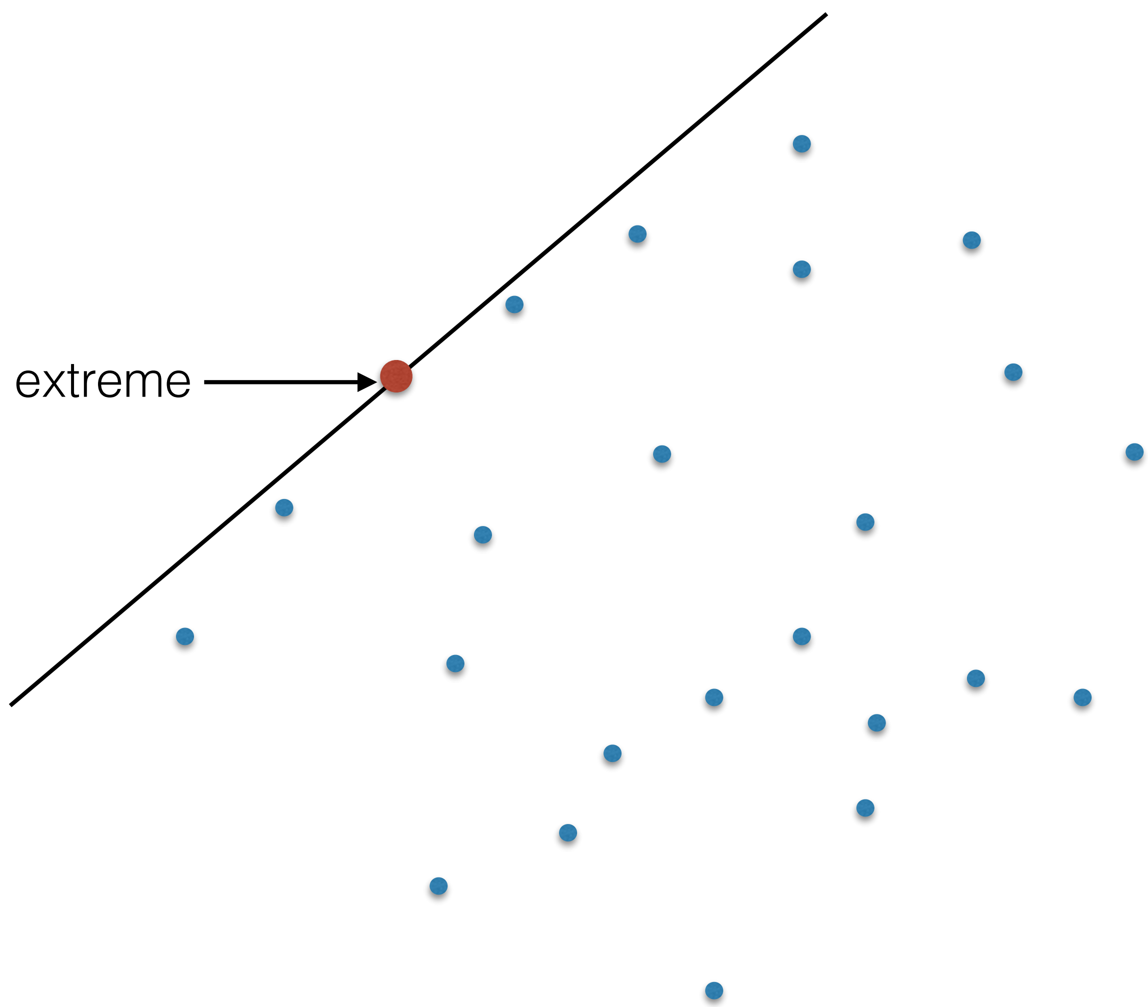
Extreme points

- A point p is called **extreme** if there exists a line l through p, such that all the other points of P are on the same side of l (and not on l)



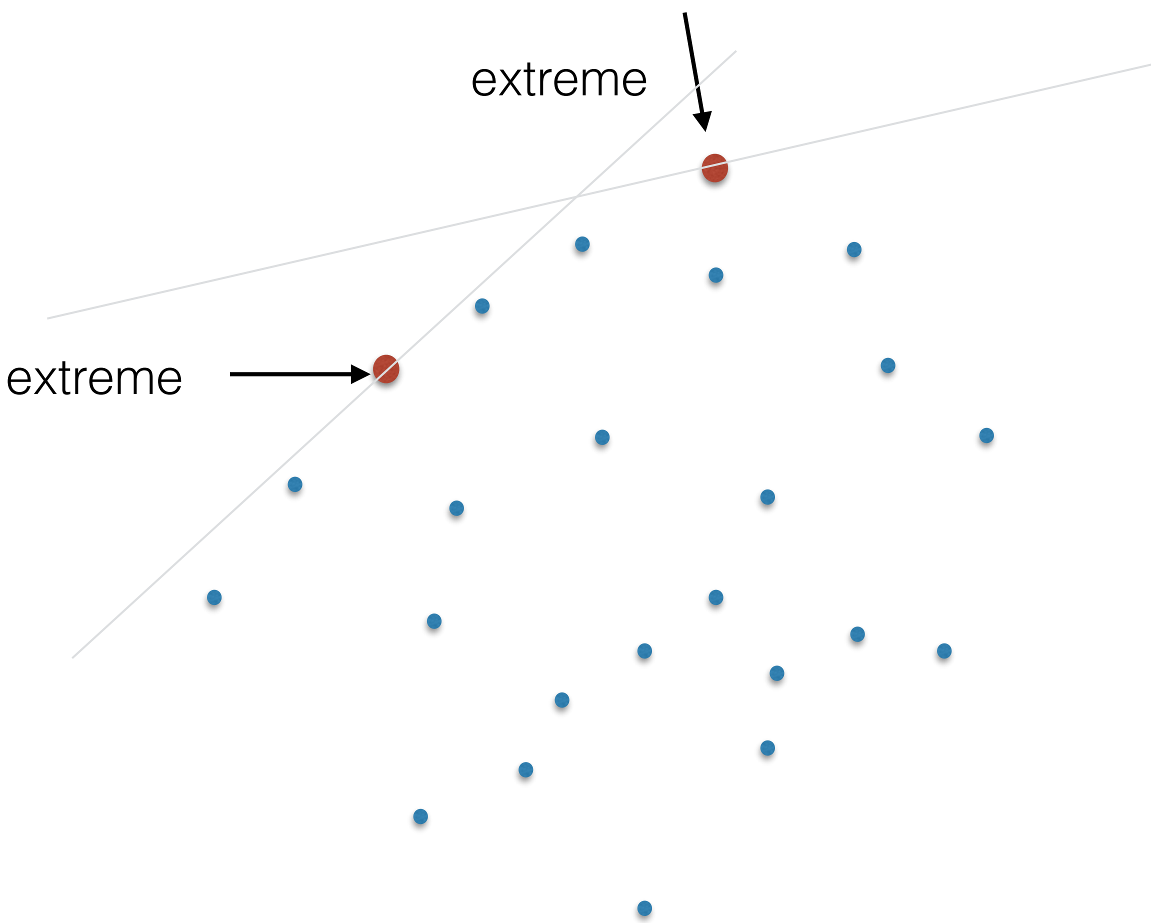
Extreme points

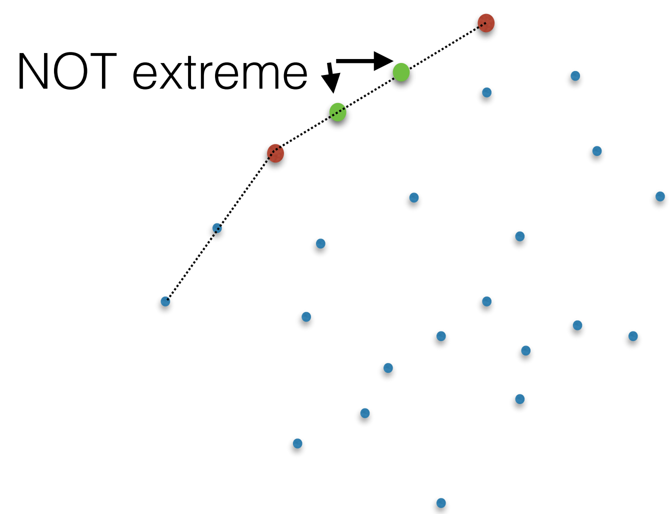
- A point p is called **extreme** if there exists a line l through p, such that all the other points of P are on the same side of l (and not on l)



Extreme points

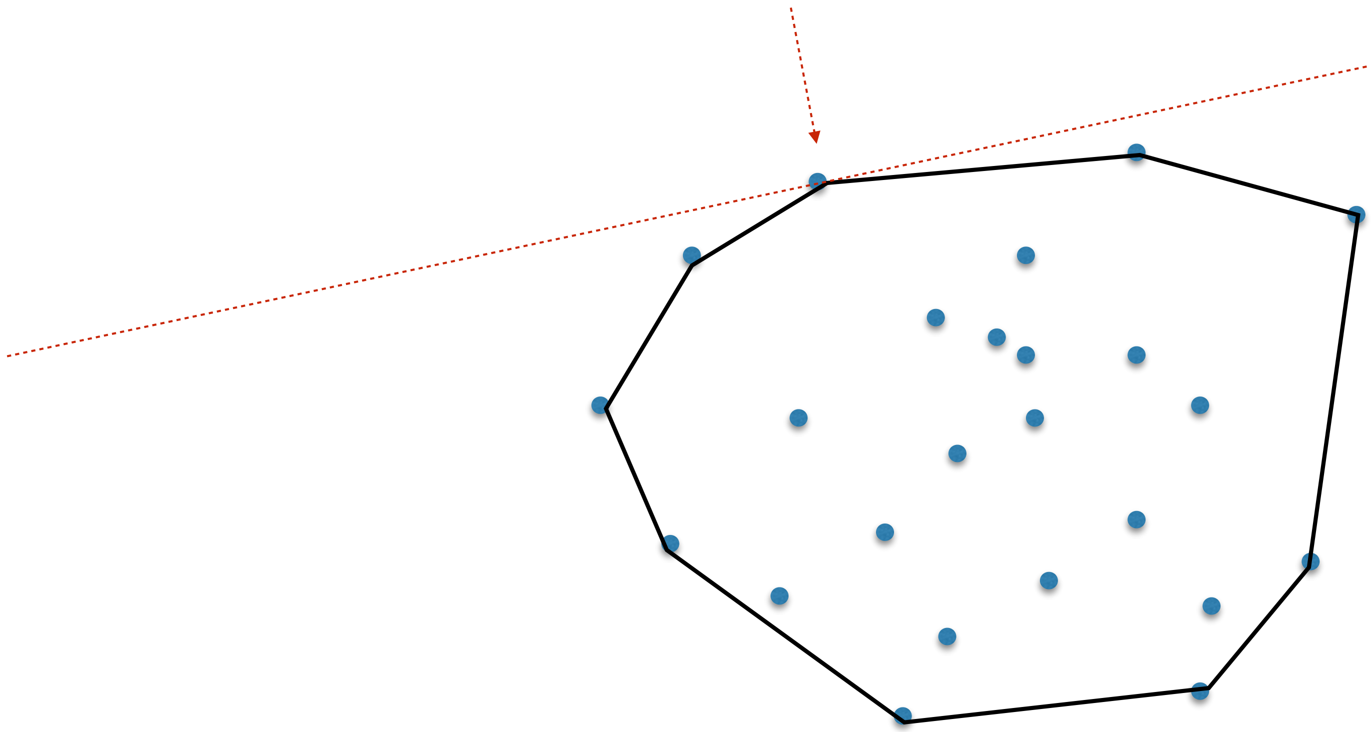
- A point p is called **extreme** if there exists a line l through p, such that all the other points of P are on the same side of l (and not on l)





A point is on the CH <==> it is extreme

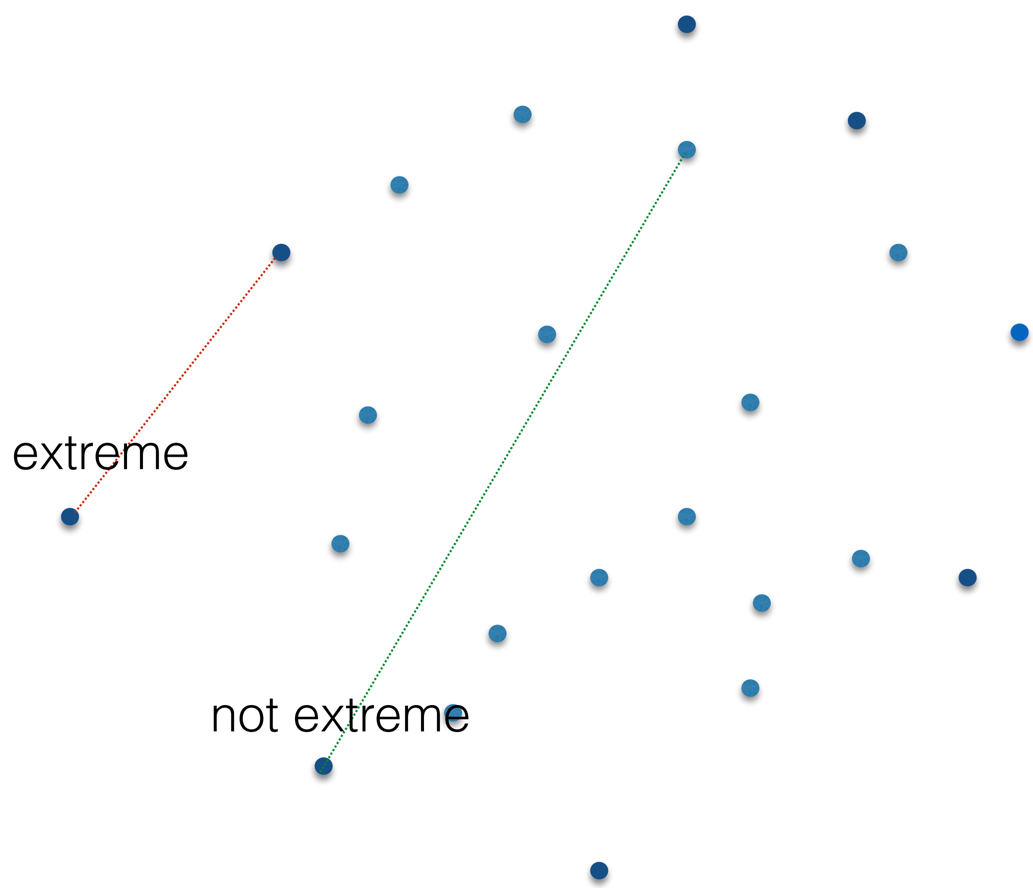
All points on the CH are extreme



All extreme points are on the CH

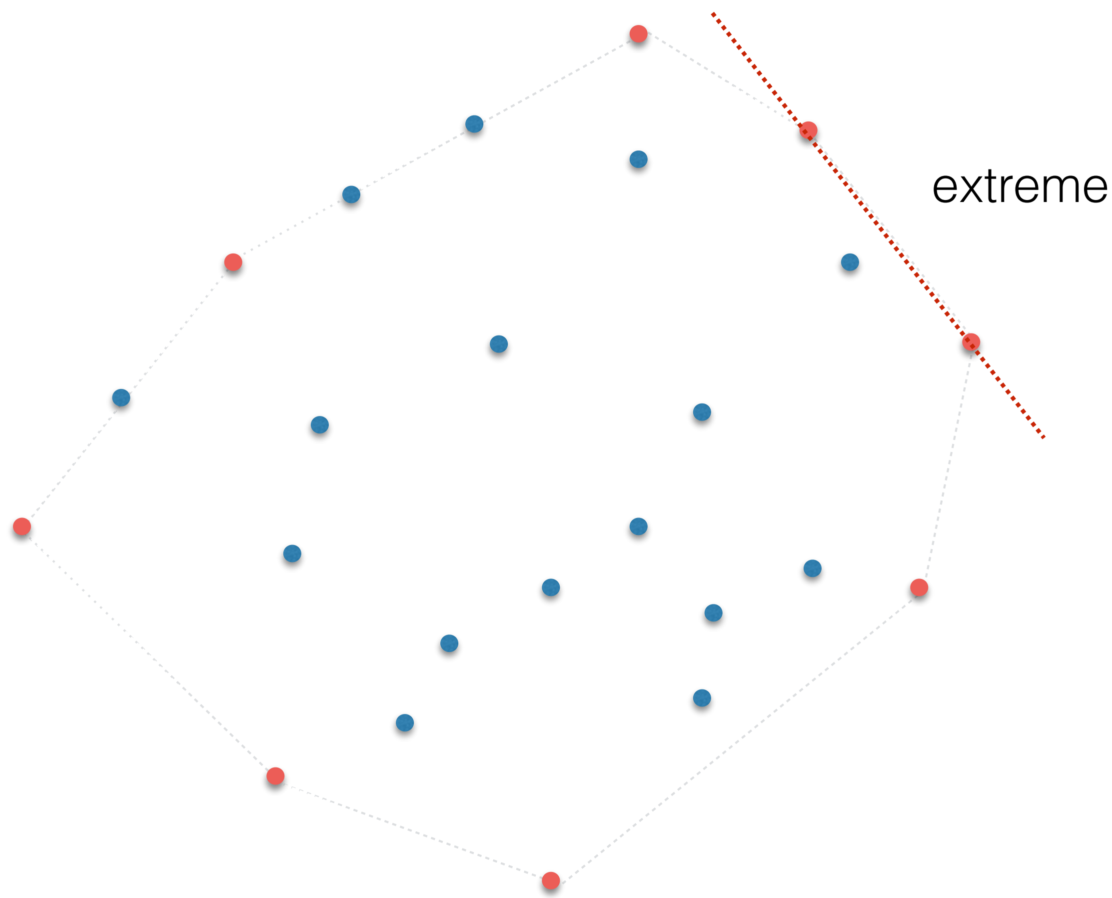
Extreme edges

- An edge (p_i, p_j) is **extreme** if all the other points of P are on one side of it (or on)



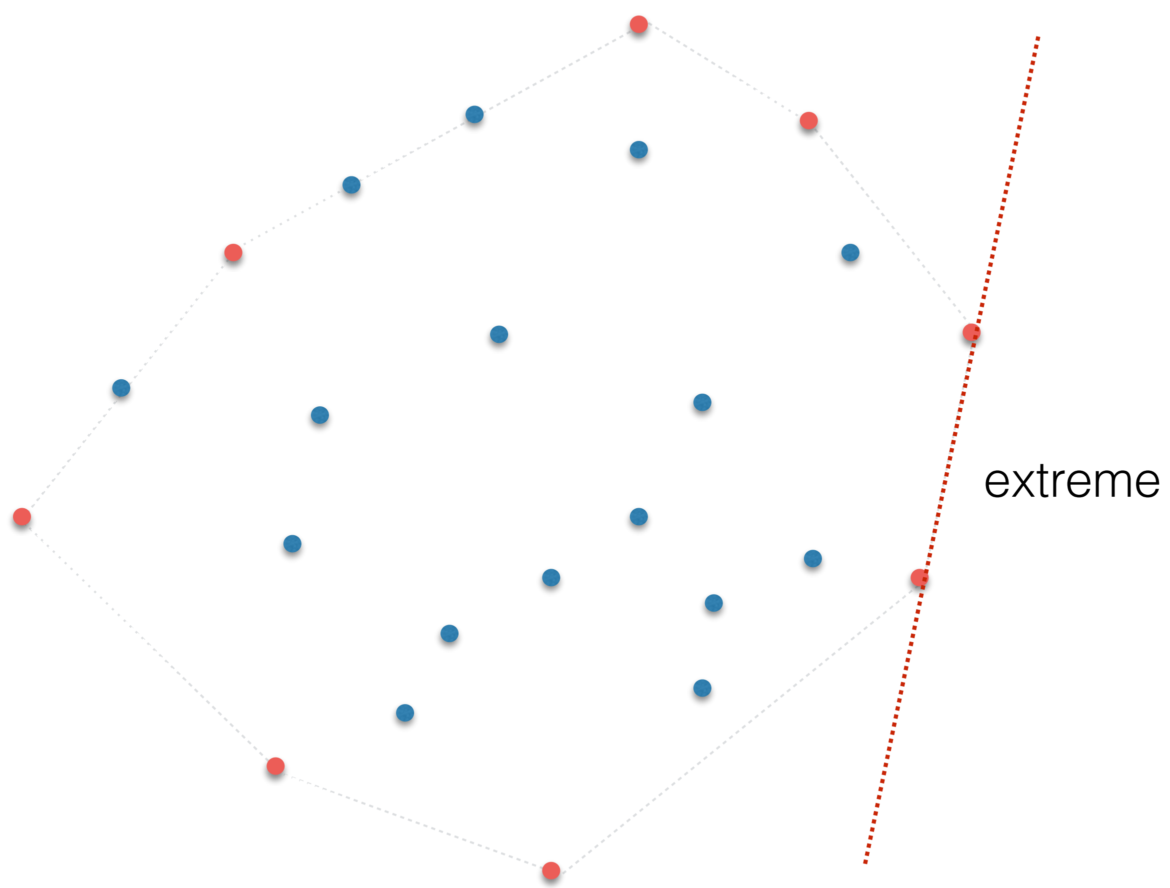
Extreme edges

- An edge (p_i, p_j) is **extreme** if all the other points of P are on one side of it (or on)



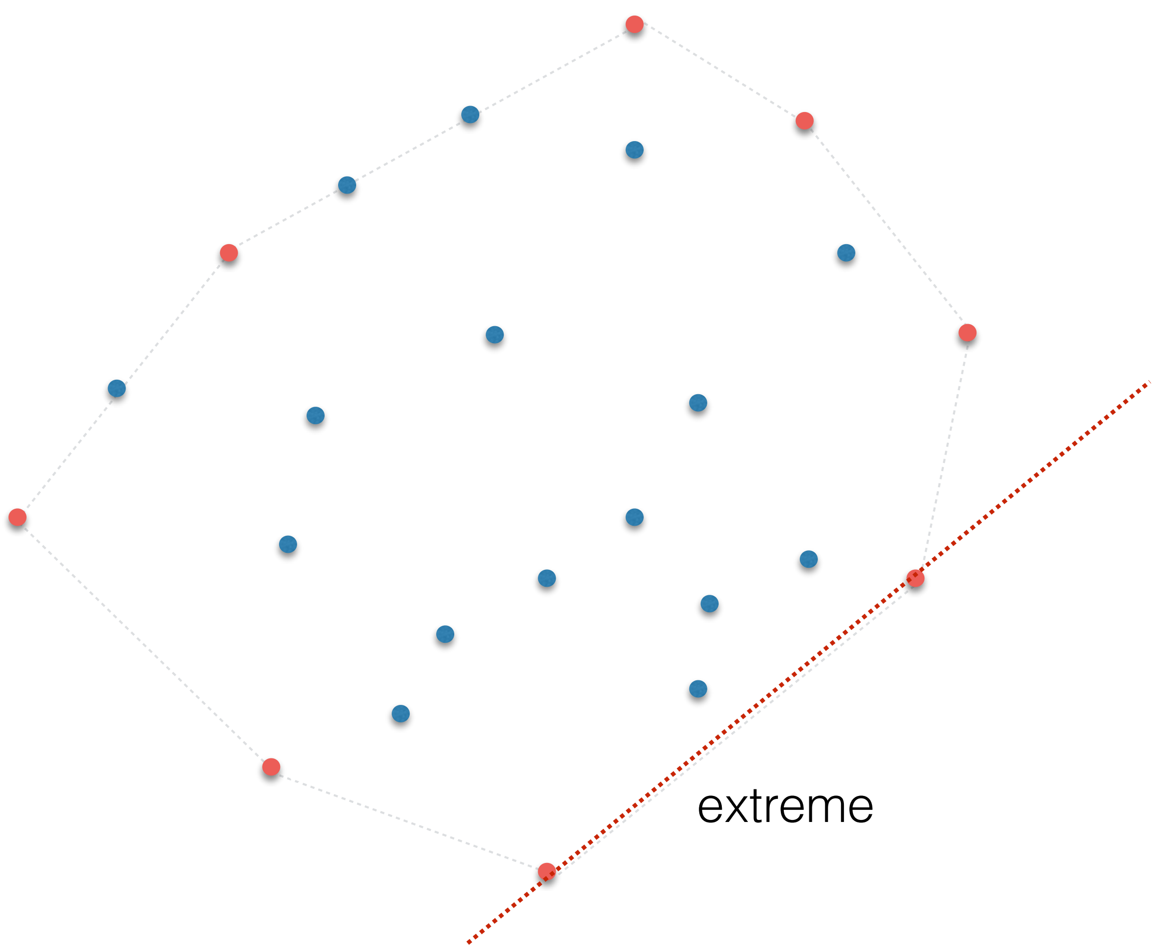
Extreme edges

- An edge (p_i, p_j) is **extreme** if all the other points of P are on one side of it (or on)



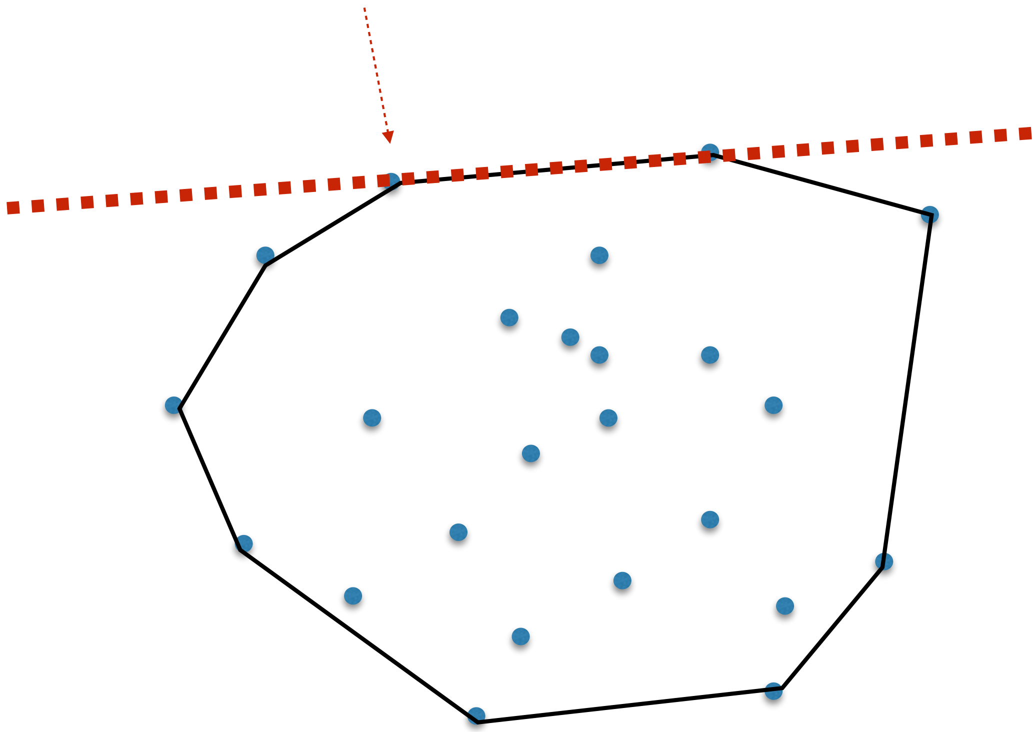
Extreme edges

- An edge (p_i, p_j) is extreme if all the other points of P are on one side of it (or on)



An edge is on the CH <==> it is extreme

All edges on the CH are extreme

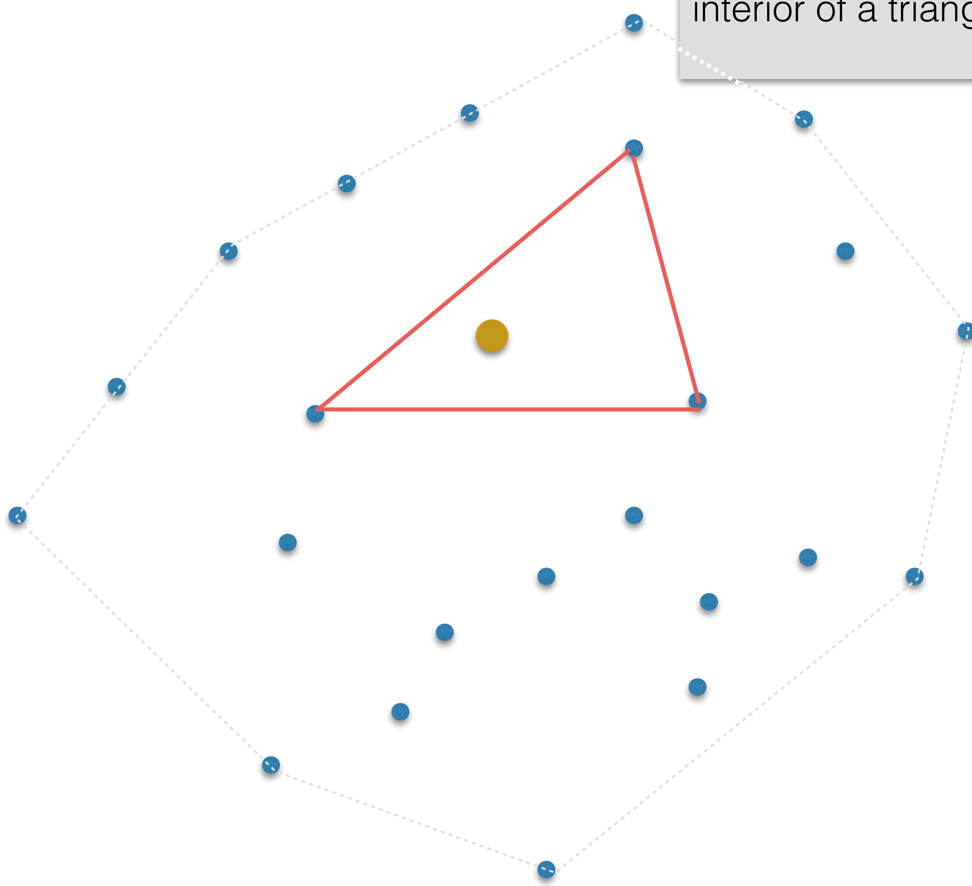


All extreme edges are on the CH

Interior points

- p interior <==> p **not** on the CH

A point p is called interior if p is contained in the interior of a triangle formed by three other points of P



Convex hull properties: Summary

- Walking counter-clockwise on the boundary of the CH you make only left turns
- Consider a point p inside the CH. Then the points on the boundary of the CH are encountered in sorted radial order around p
- CH consists of extreme points and edges
    - point is extreme <==> it is on the CH
    - (p_i, p_j) form an edge on the CH <==> edge (p_i, p_j) is extreme
    - point p is interior <==> p not on the CH

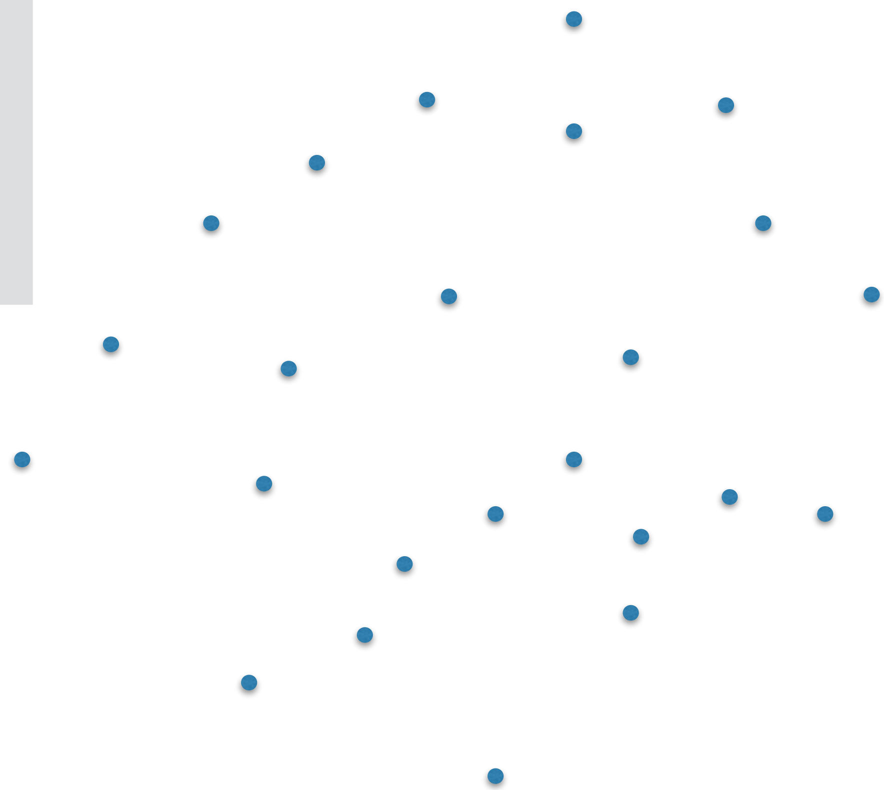
Algorithm: Brute force

Algorithm: Brute force

Idea: Find extreme edges

Algorithm (input P)

- for all distinct pairs (p_i, p_j)
    - check if edge (p_i,p_j) is extreme



- Analysis?

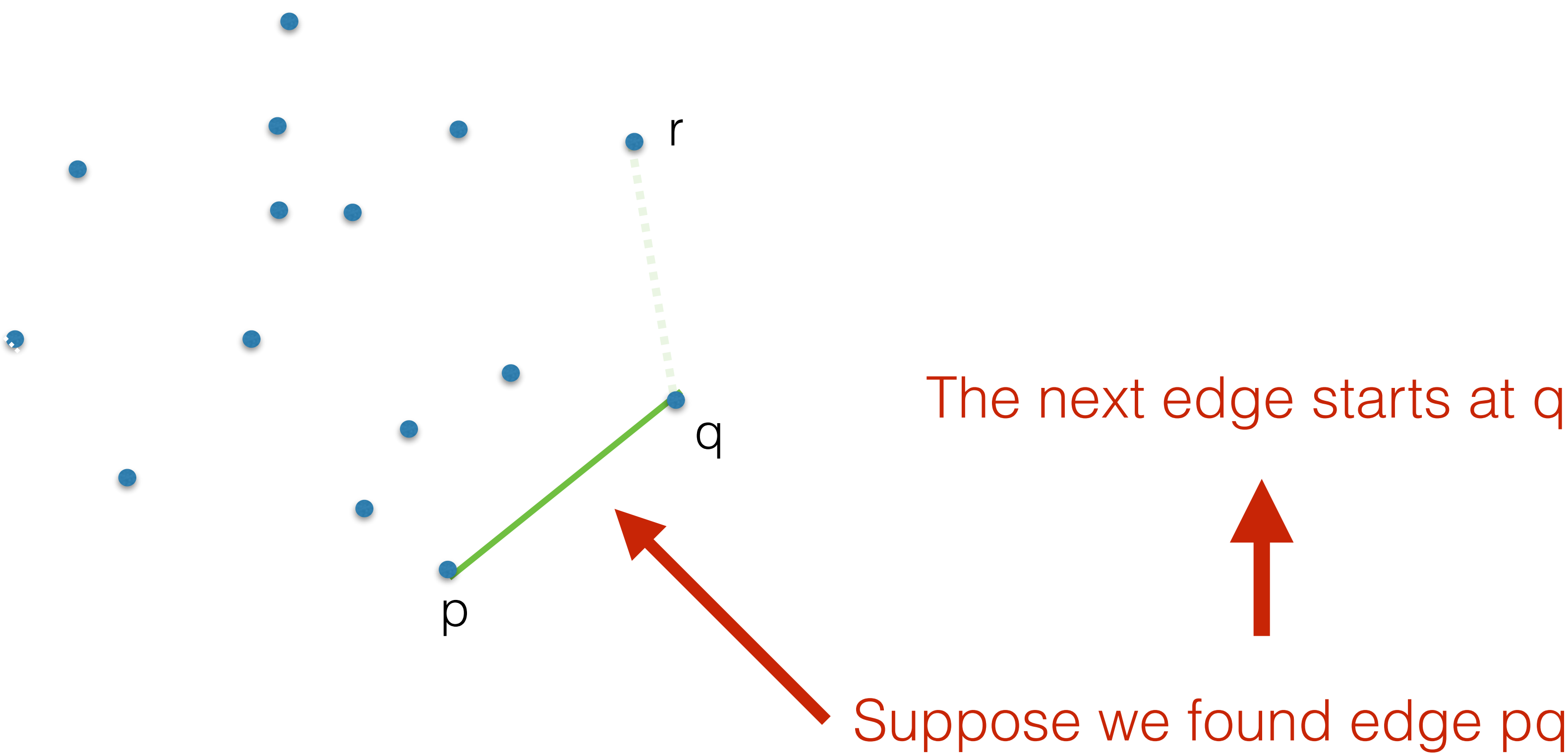
Algorithm: Gift wrapping

- by Chand and Kapur [1970].

Algorithm: Gift wrapping

We know that CH consists of extreme edges, and each edge shares a vertex with next edge

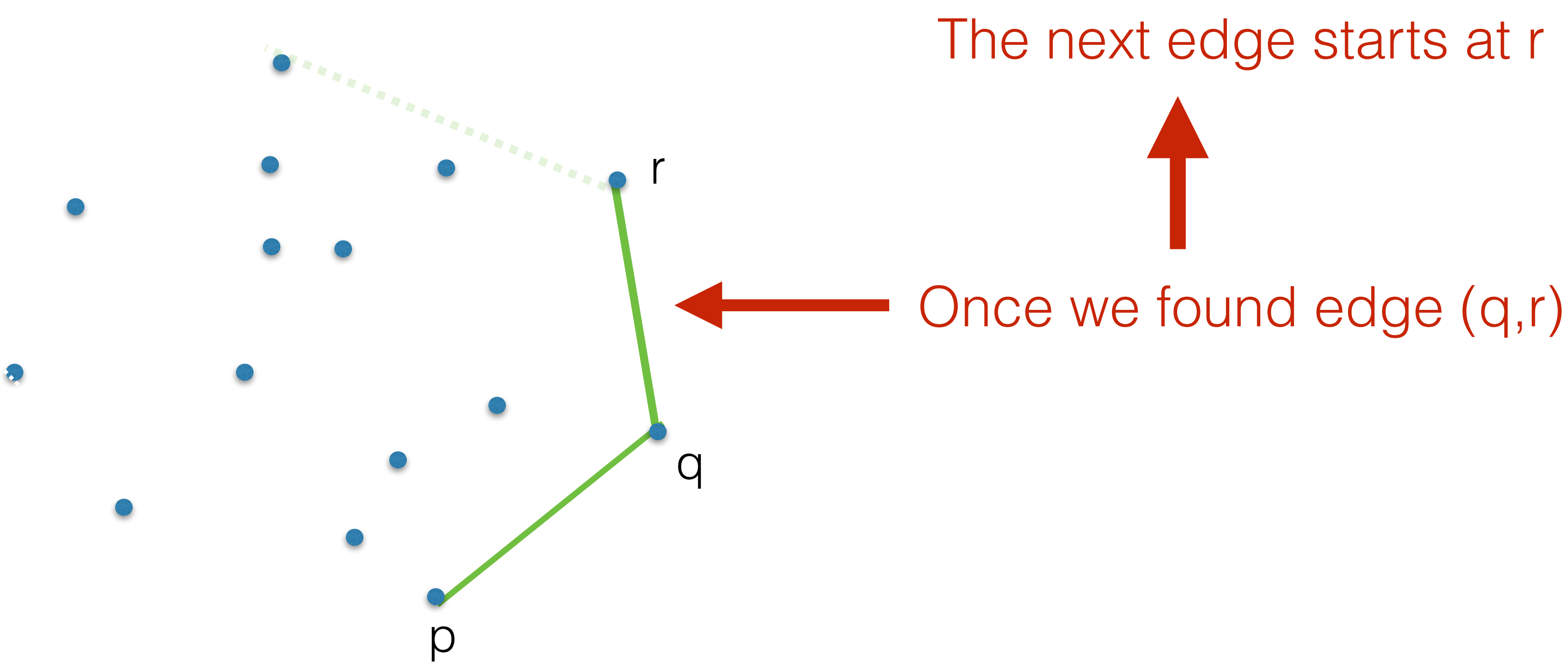
Idea: use an edge to find the next one



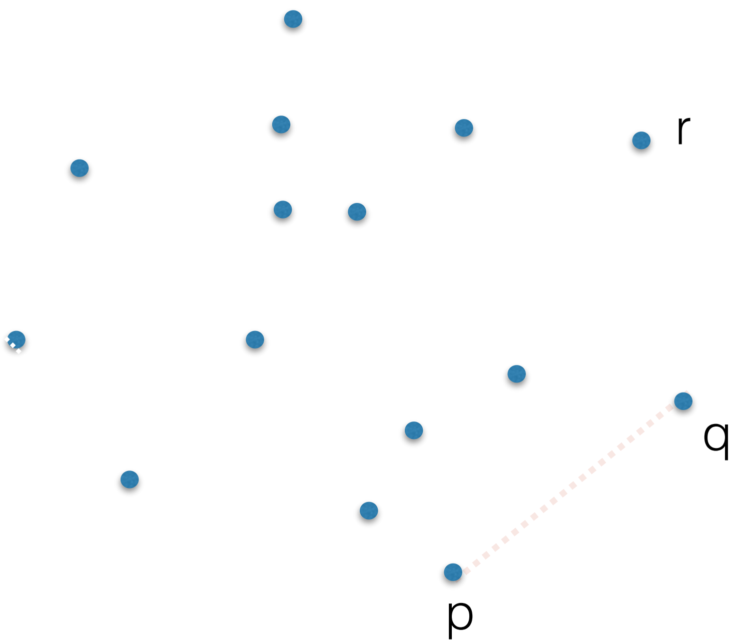
Algorithm: Gift wrapping

We know that CH consists of extreme edges, and each edge shares a vertex with next edge

Idea: use an edge to find the next one

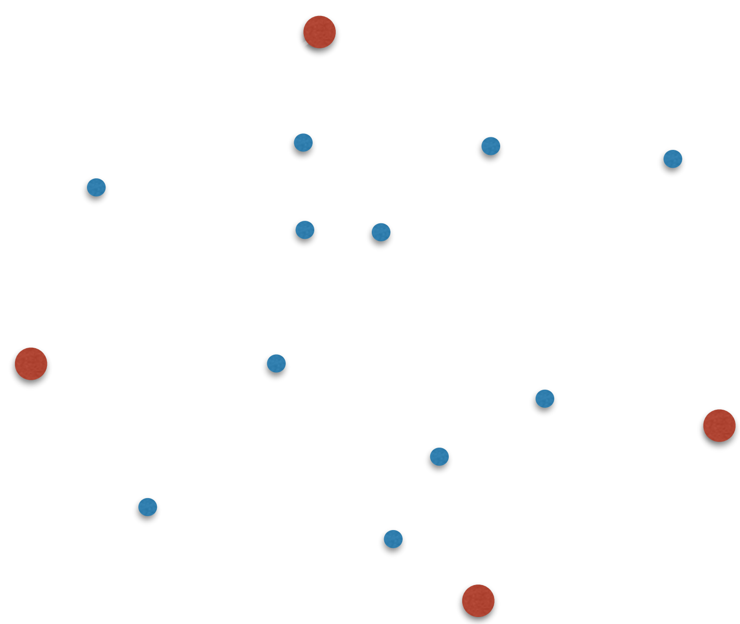


How to find an extreme edge to start from?



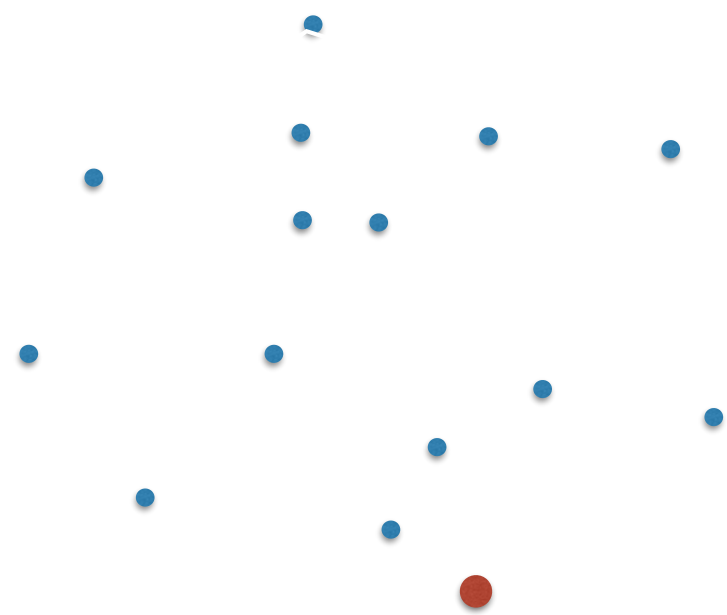
Start from a point p that is guaranteed to be in CH

- Claim
  - point with minimum x-coordinate is extreme
  - point with maximum x-coordinate is extreme
  - point with minimum y-coordinate is extreme
  - point with maximum y-coordinate is extreme
- Can you justify why?



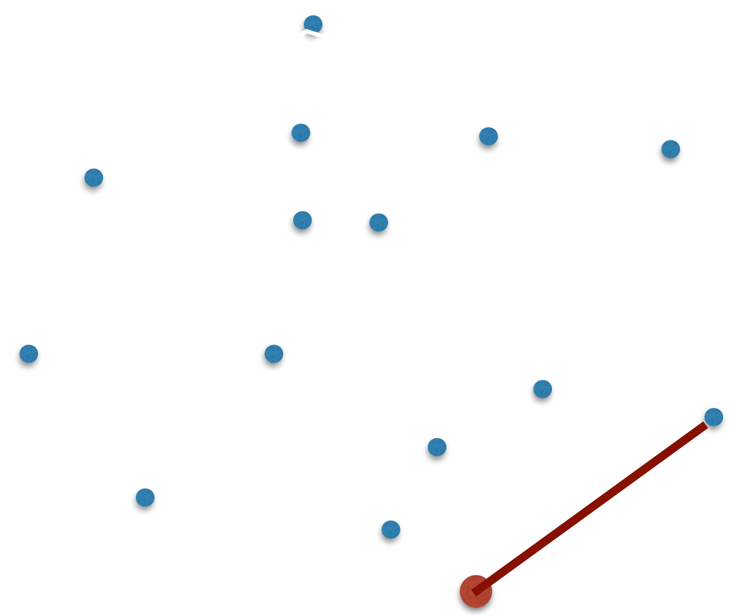
Algorithm: Gift wrapping

- Start from bottom-most point (if more than one, pick right most)



Algorithm: Gift wrapping

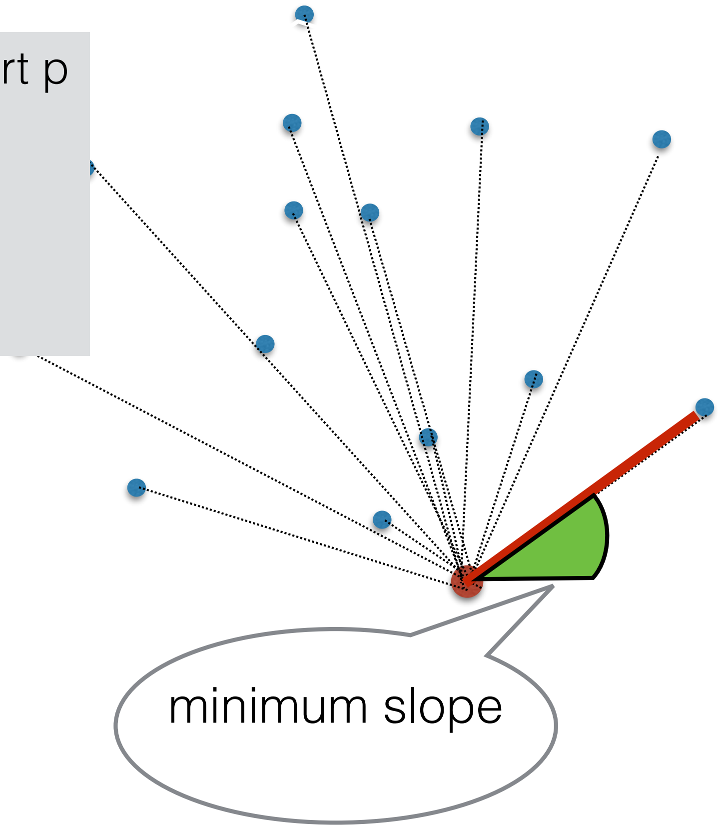
- Start from bottom-most point (if more than one, pick right most)
- Find first edge: how??



Algorithm: Gift wrapping

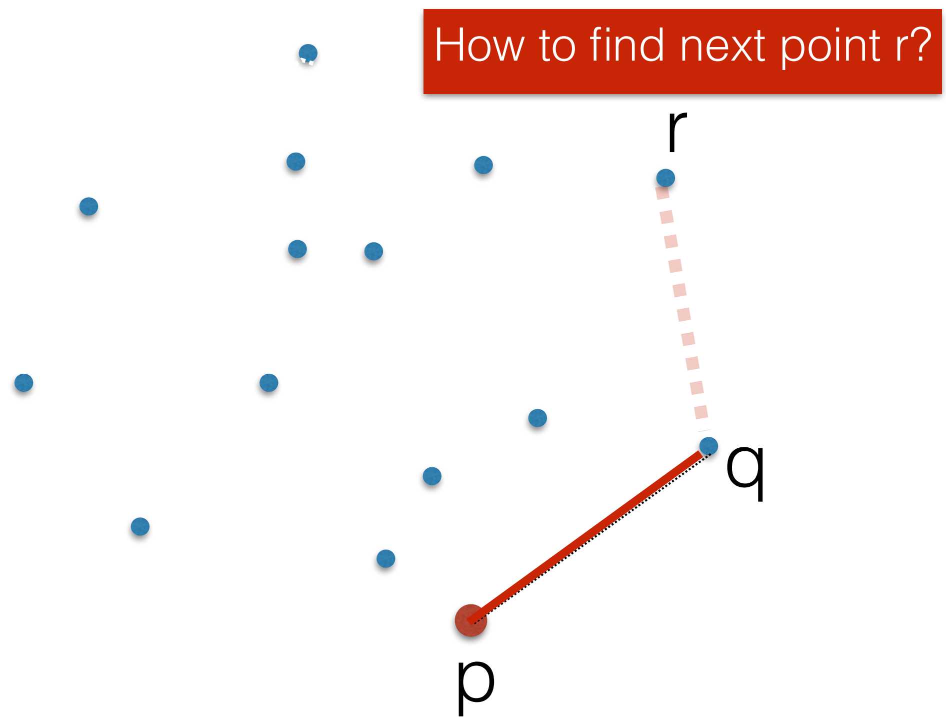
- Start from bottom-most point (if more than one, pick right most)
- Find first edge:
  - for each point p': compute slope of p' wrt p
  - let q = point with smallest slope

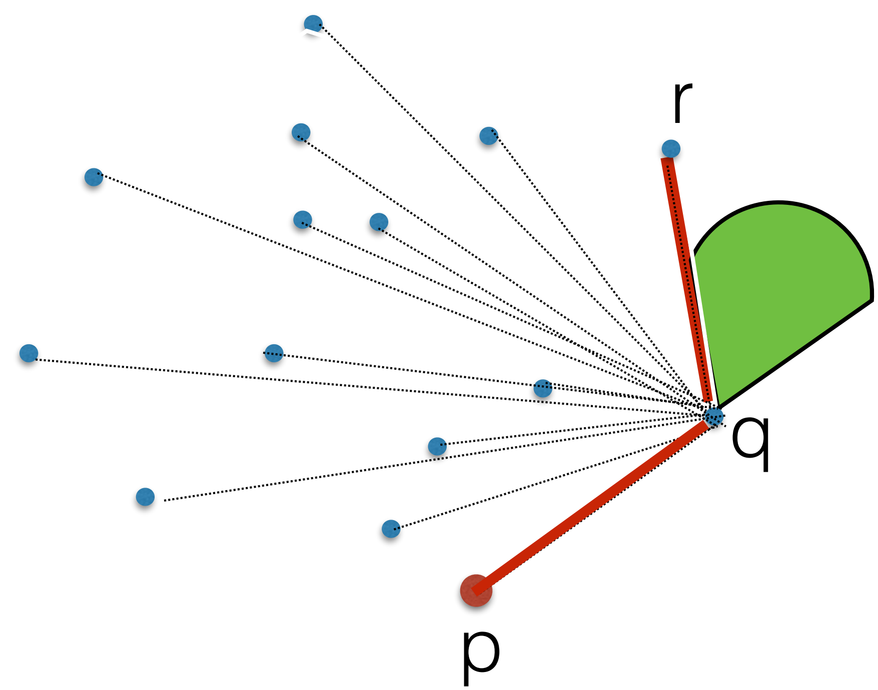
    //claim: pq is extreme edge
  - output (p, q) as first edge



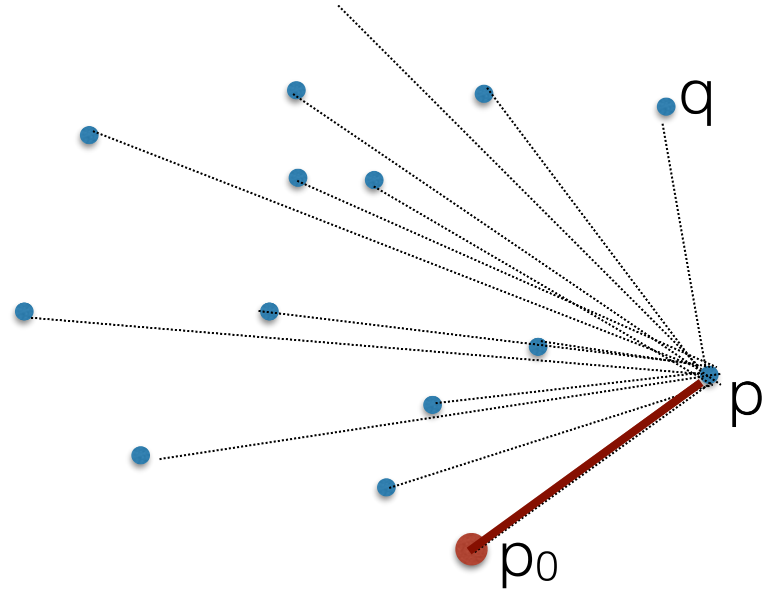
Algorithm: Gift wrapping

- Start from bottom-most point   (if more than one, pick right most)
- Find first edge pq
- Repeat: find extreme edge from q




r
q
p

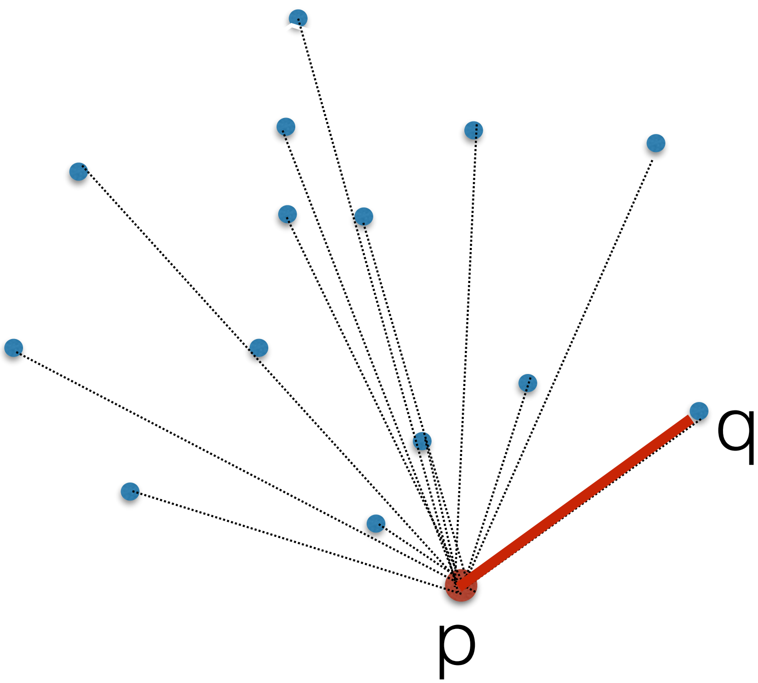
Algorithm: Gift wrapping

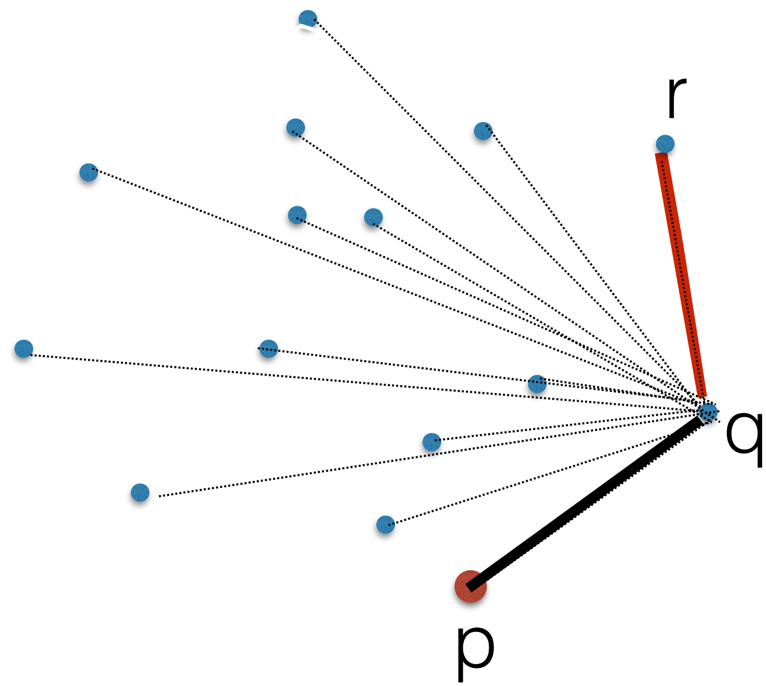


- Let p_0 = point with smallest y-coord (if more than one, pick right-most)
- Let p = point with smallest slope wrt p_0
- add points p_0, p to the CH
- repeat
  - let q = point with smallest slope wrt prev edge on the hull
  - add point q to the CH
- until q = p_0

Can be implemented with left()

- q is the point that appears to be furthest to the right to someone standing at p





- initialize q to be an arbitrary point
- for each point u   (u != q):
    - if left(p, u, q):   q = u

Class work

- Simulate Gift-Wrapping on an arbitrary (small) set of points
- What are configurations of points that cause troubles for Gift Wrapping? (referred to as degenerate cases)
- Running time: Express function of n and k, where k is the output size (number of points on the convex hull)
    - How small/large can k be for a set of n points?
    - Show examples that trigger best/worst cases
    - Based on this, when is Gift-wrapping a good choice to compute CH (i.e. when is it efficient)?

Gift wrapping summary

- Runs in $O(k \cdot n)$ time, where k is the size of the CH(P)
- Efficient if k is small:
    - For k = O(1), it takes $O(n)$
- Not efficient if k is large:
    - For $k = O(n)$, Gift wrapping takes $O(n^2)$
- Faster algorithms are known
- Gift wrapping extends easily to 3D and for many years was the primary algorithm for 3D

Summary

- Brute force: $O(n^3)$
- Gift wrapping: $O(k \cdot n)$
    - output-size sensitive: O(n) best case, O(n^2) worst case
    - ✦ by Chand and Kapur [1970]. Extends to 3D and to arbitrary dimensions; for many years was the primary algorithm for higher dimensions
- Graham scan
- Quickhull
- incremental,
- divide-and-conquer
- $\Omega(n \lg n)$ lower bound