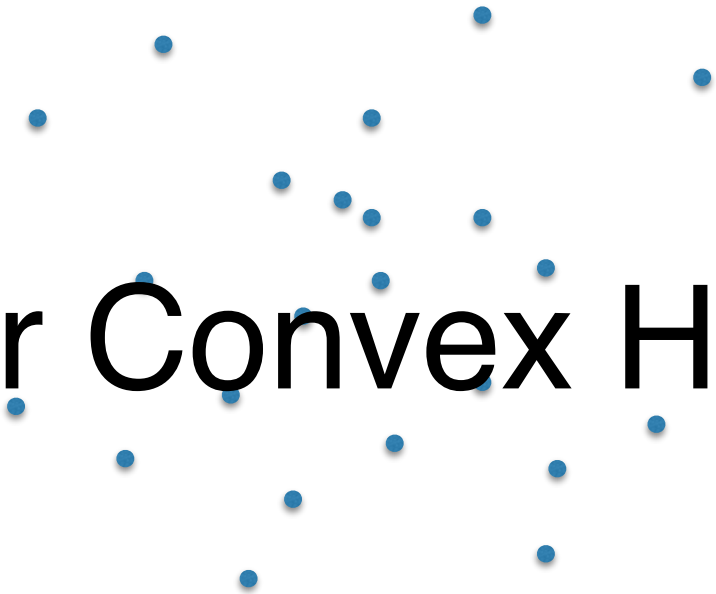
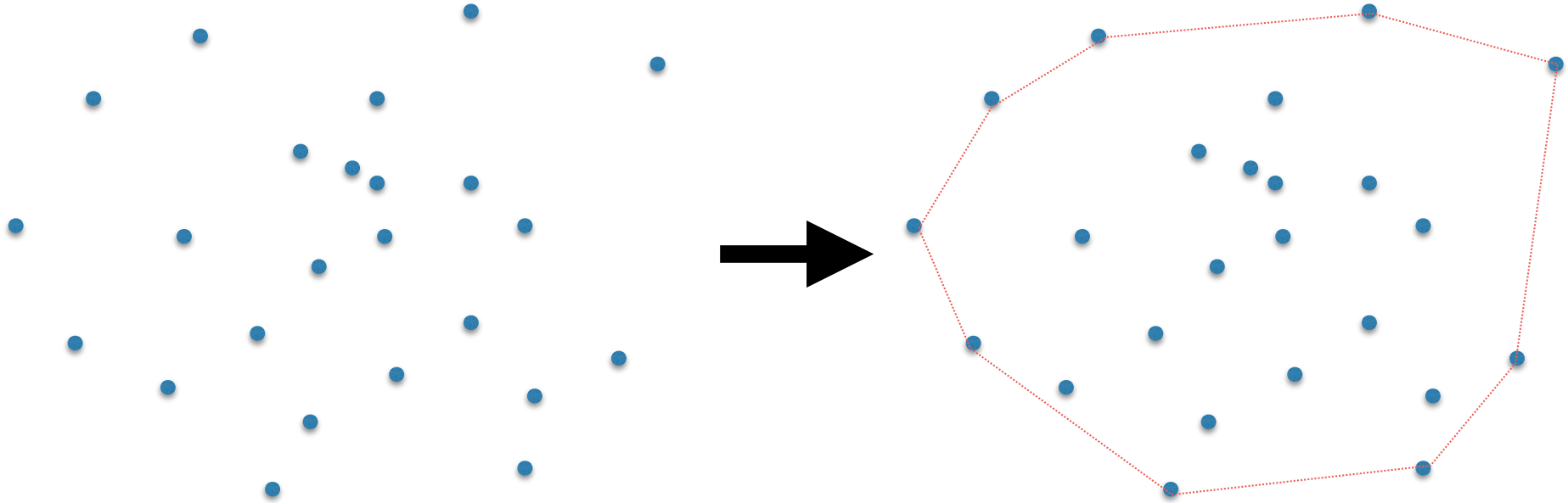


Planar Convex Hulls (II)

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)

Algorithms for computing the convex hull

- Last time
  - Brute force
  - Gift wrapping
- Today
  - Graham scan
  - Quickhull
- Next time
  - Andrew's monotone chain
  - Incremental hull
  - Divide-and -conquer hull
  - lower bound

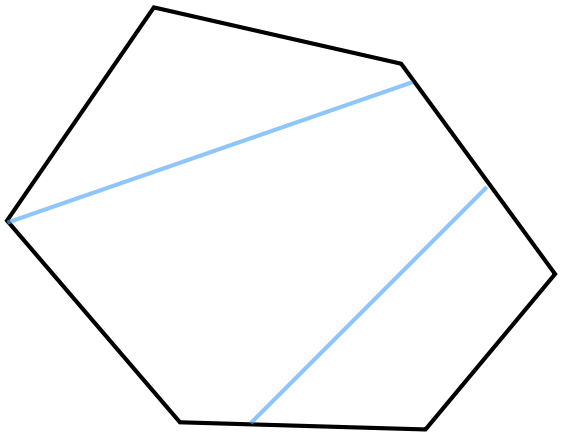
Start review

Convex hull properties

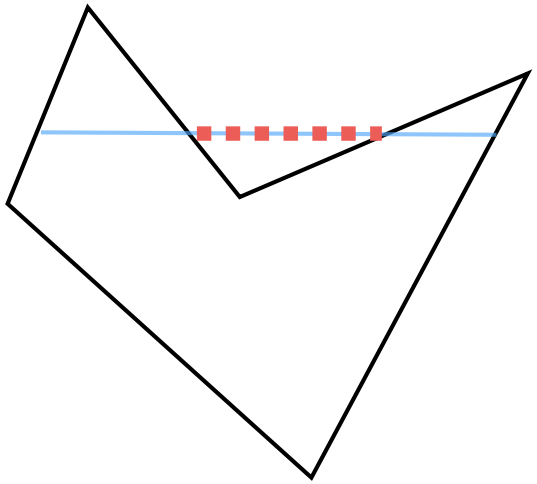
- CH consists of extreme points and edges
  - point is extreme <==> it is on the CH
  - (p_i, p_j) form an edge on the CH <==> edge (p_i, p_j) is extreme
  - point p is interior <==> p not on the CH
- Walking counter-clockwise on the boundary of the CH makes only left turns
- Consider a point p inside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

Convexity

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

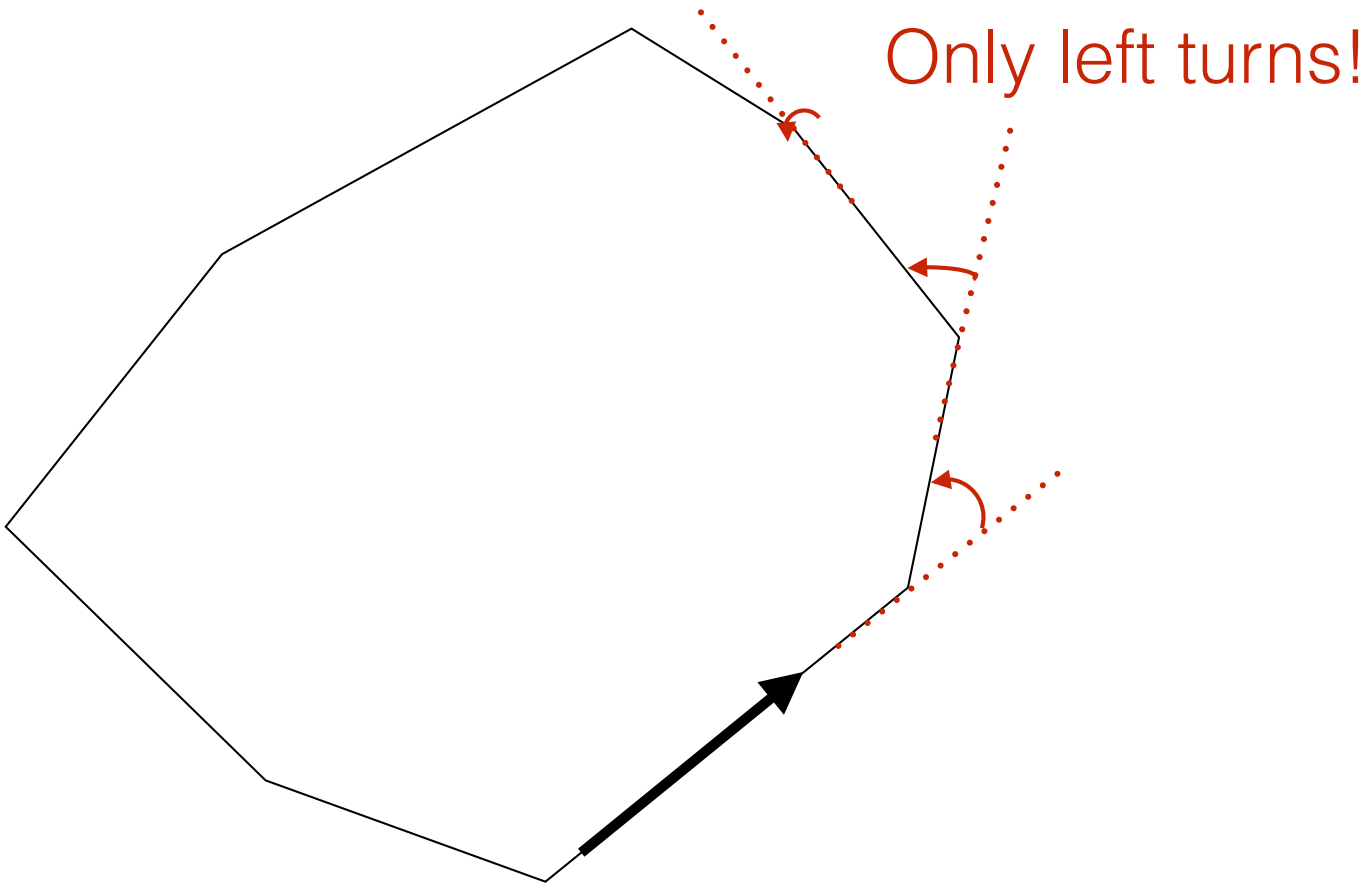


convex

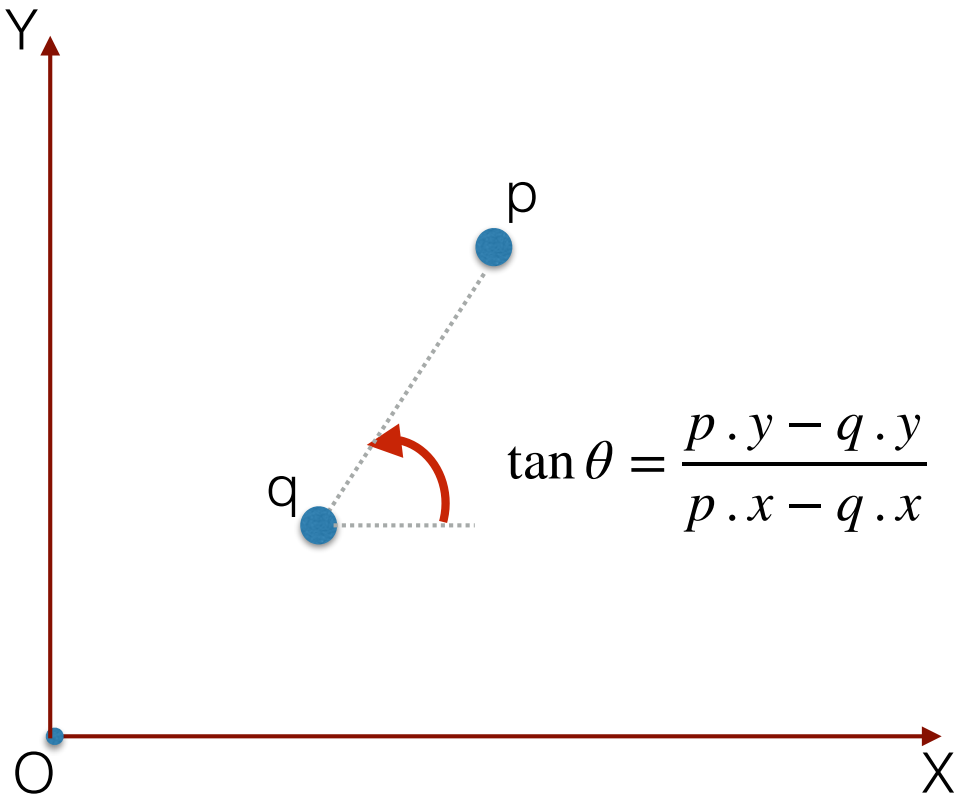


non-convex

Walk ccw along the boundary of a convex polygon

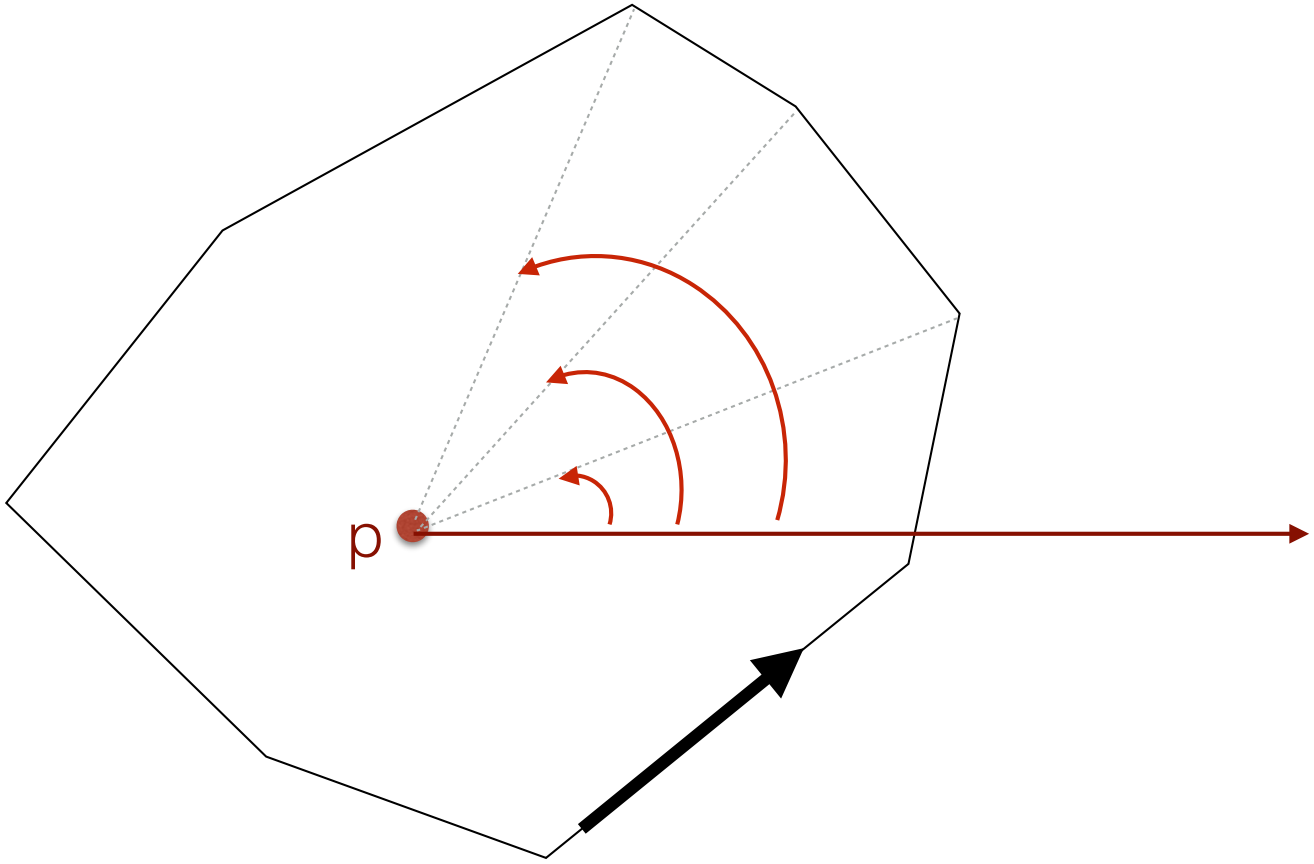


The radial angle of p with-respect-to q



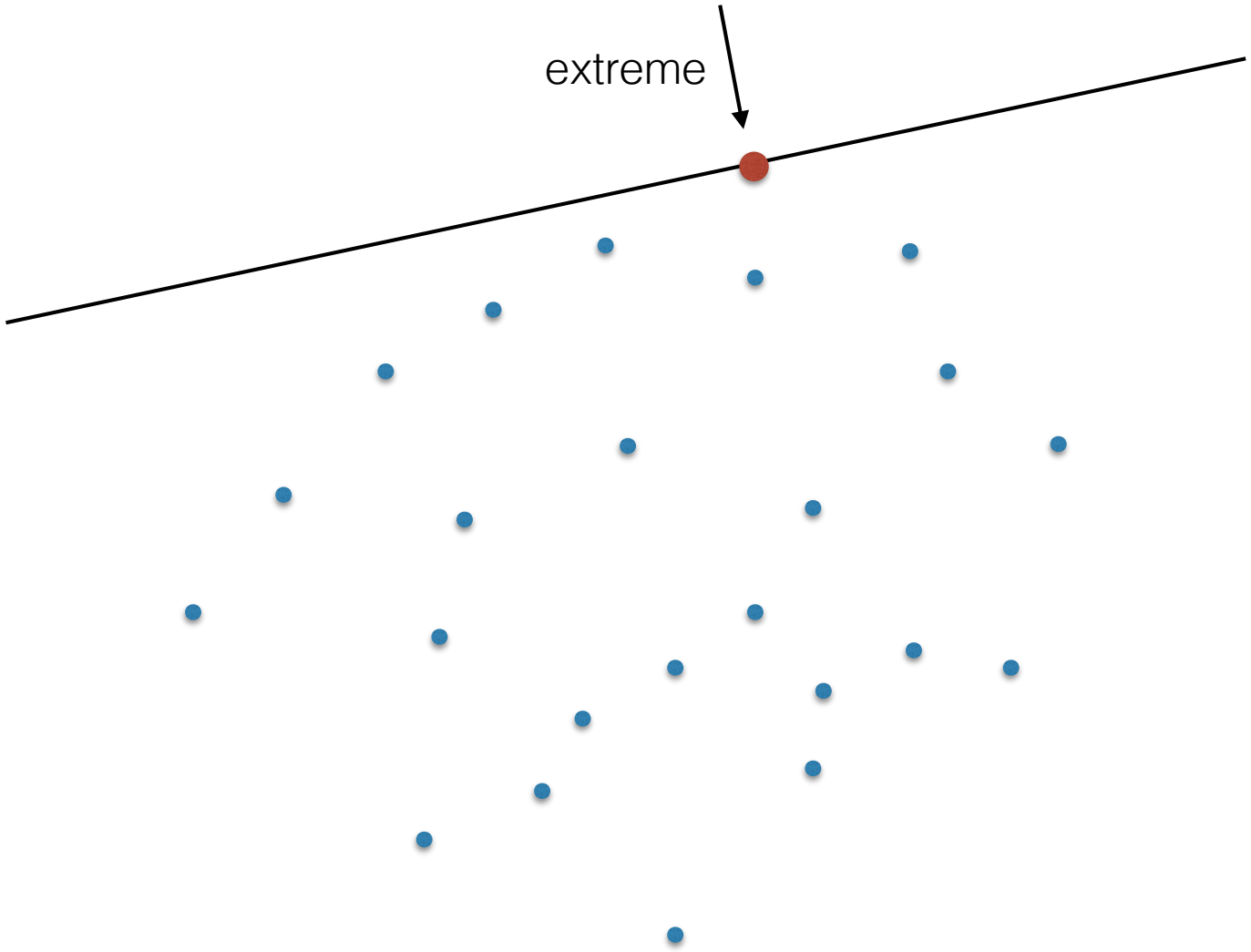
$$\tan\theta = \frac{p.y - q.y}{p.x - q.x}$$

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

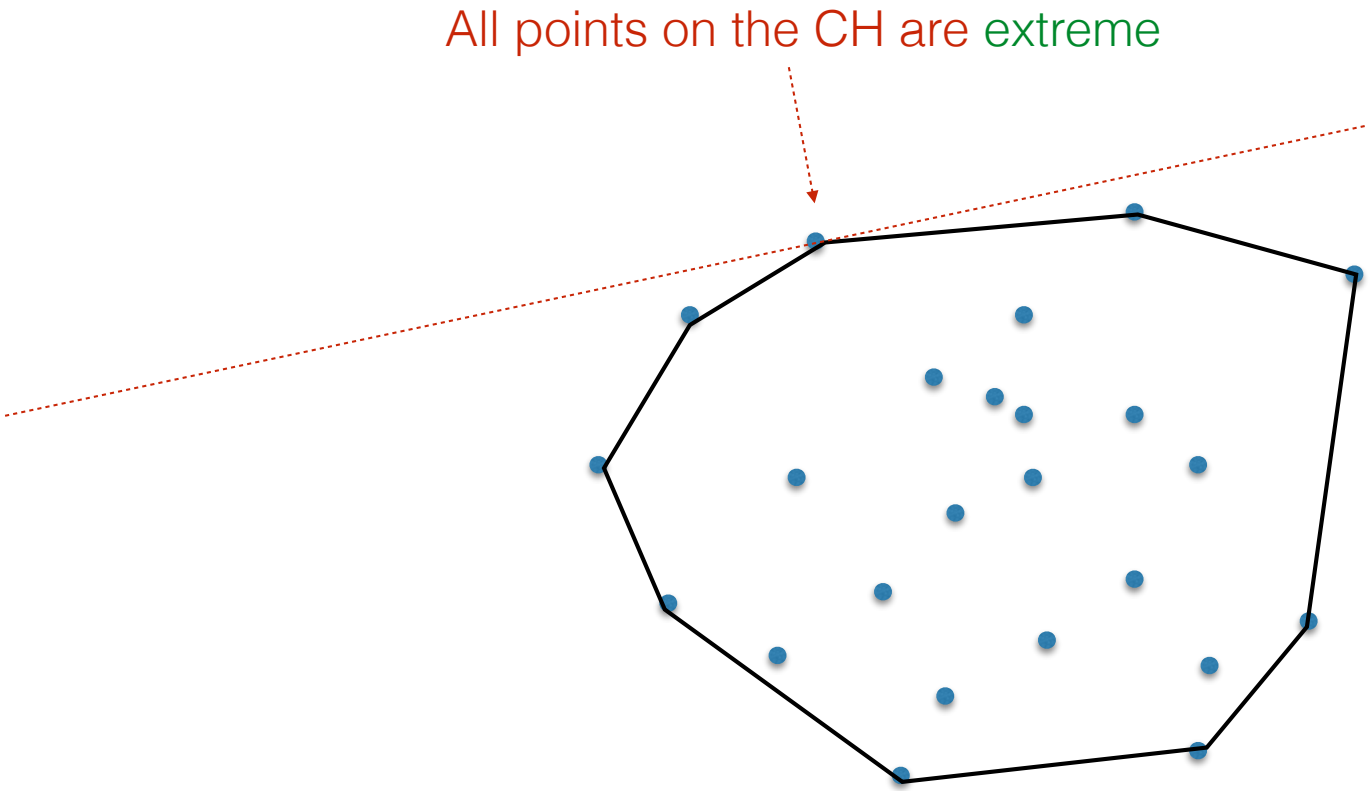


Extreme points

- 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)

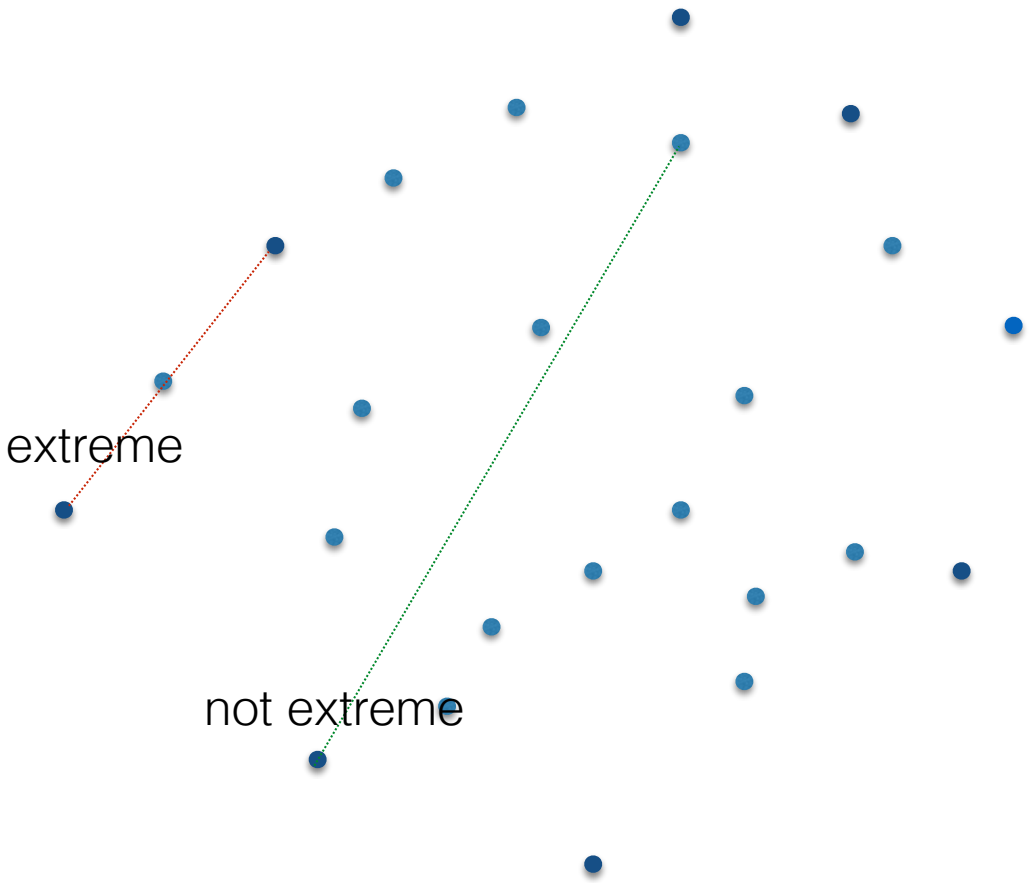


A point is on the CH <==> it is extreme



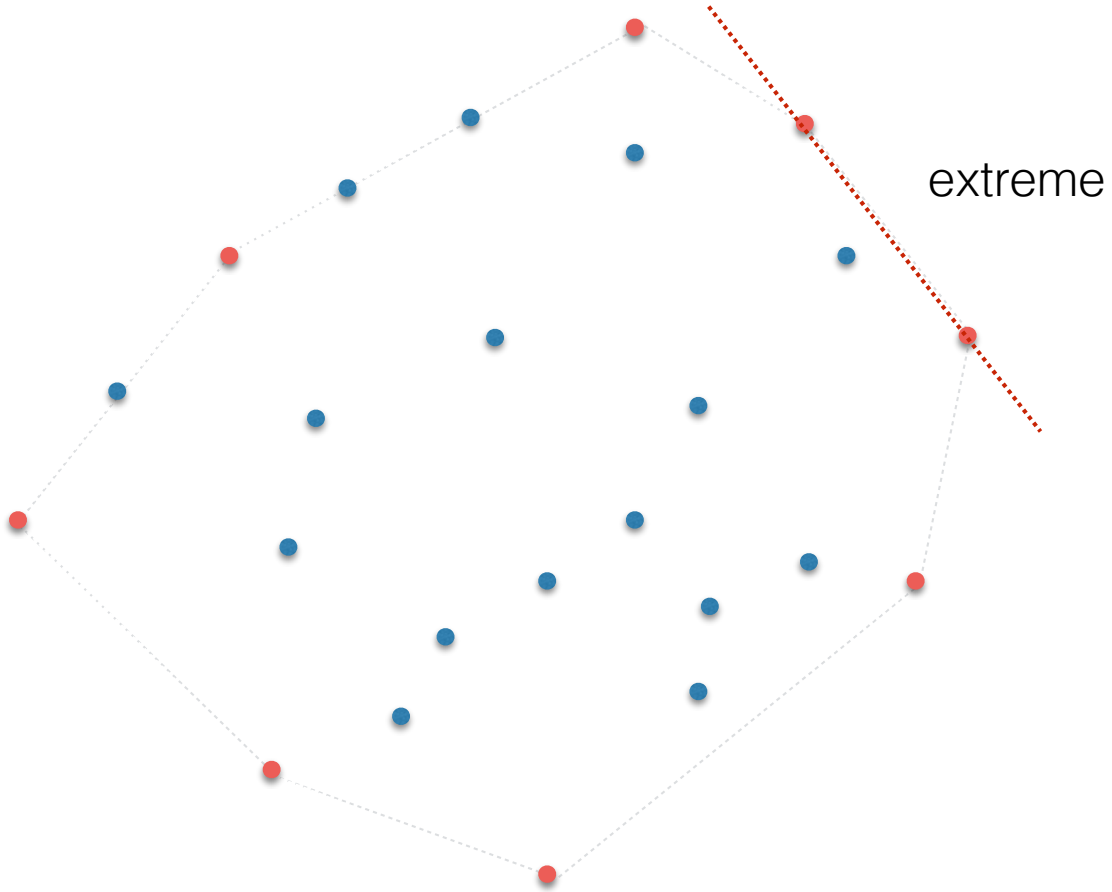
Extreme edges

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



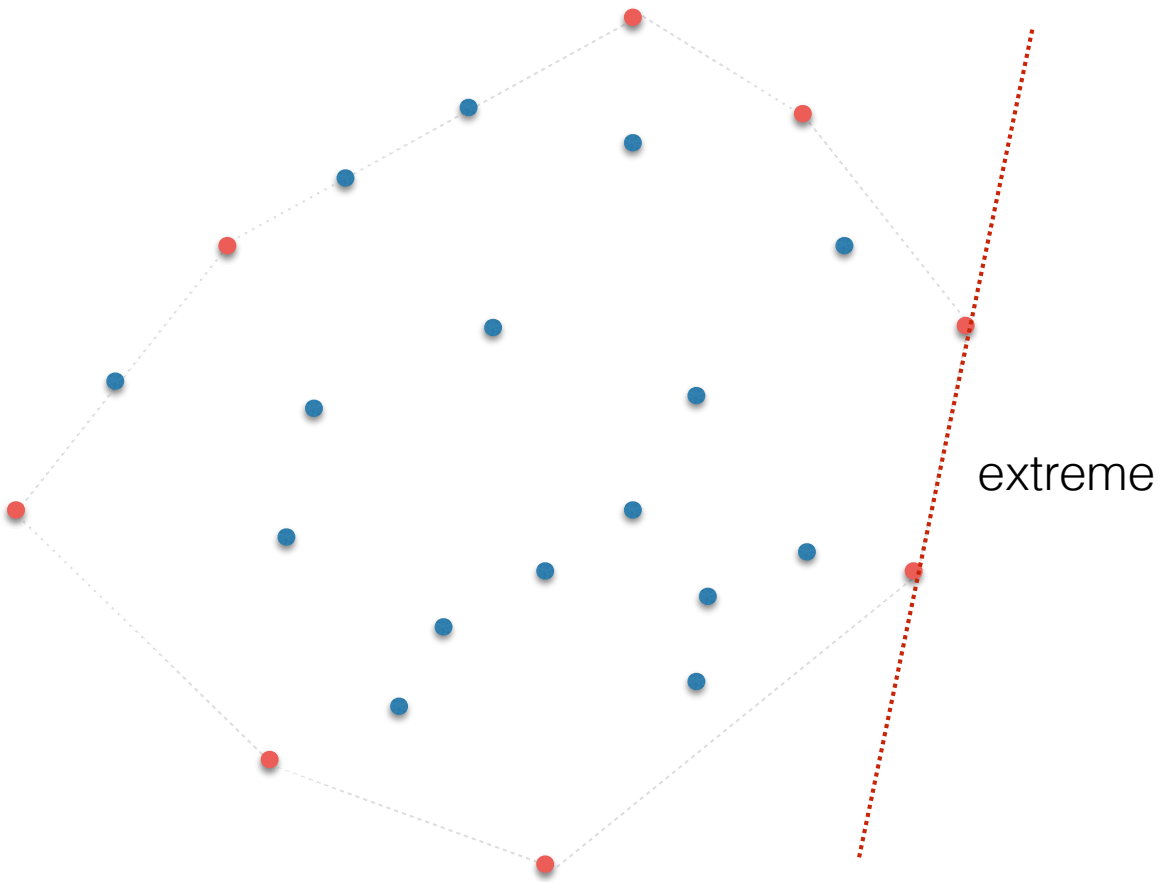
Extreme edges

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



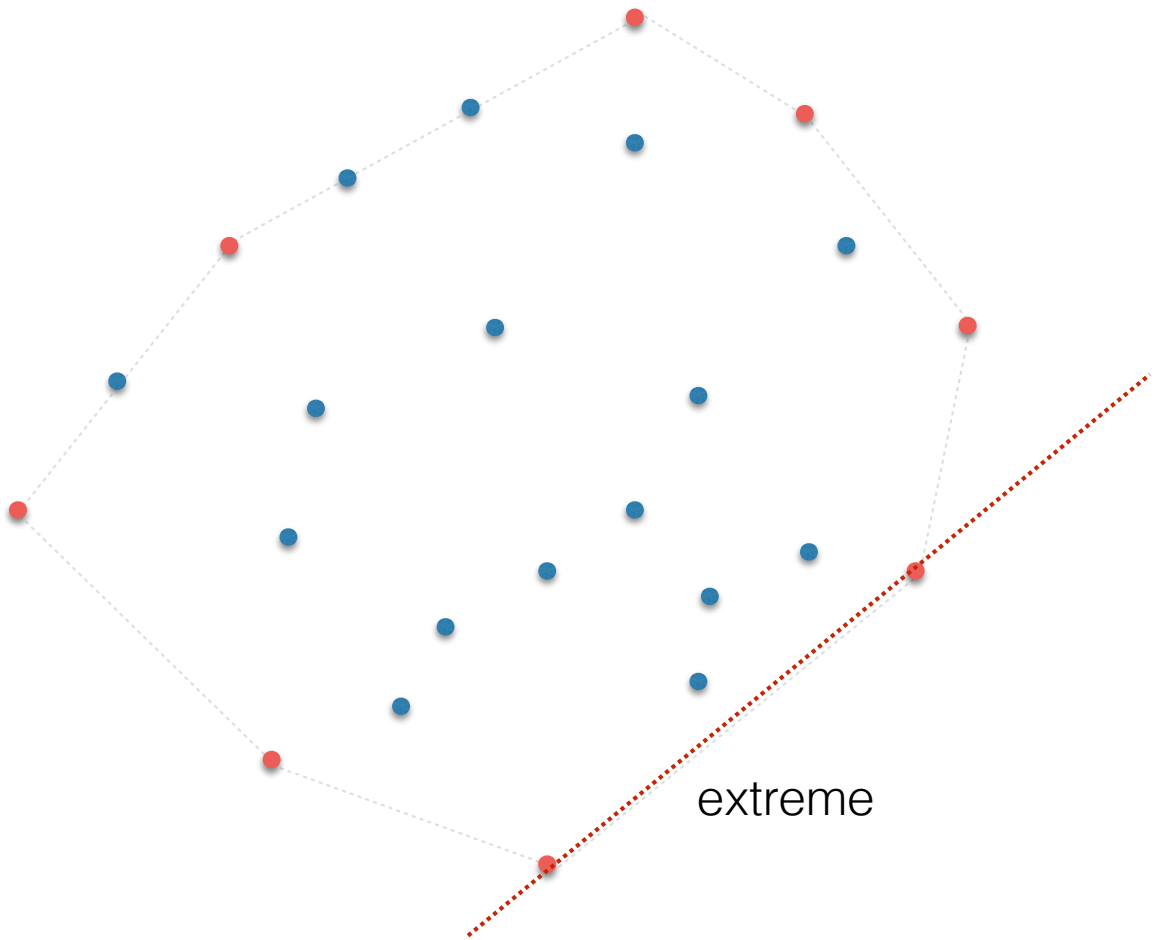
Extreme edges

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

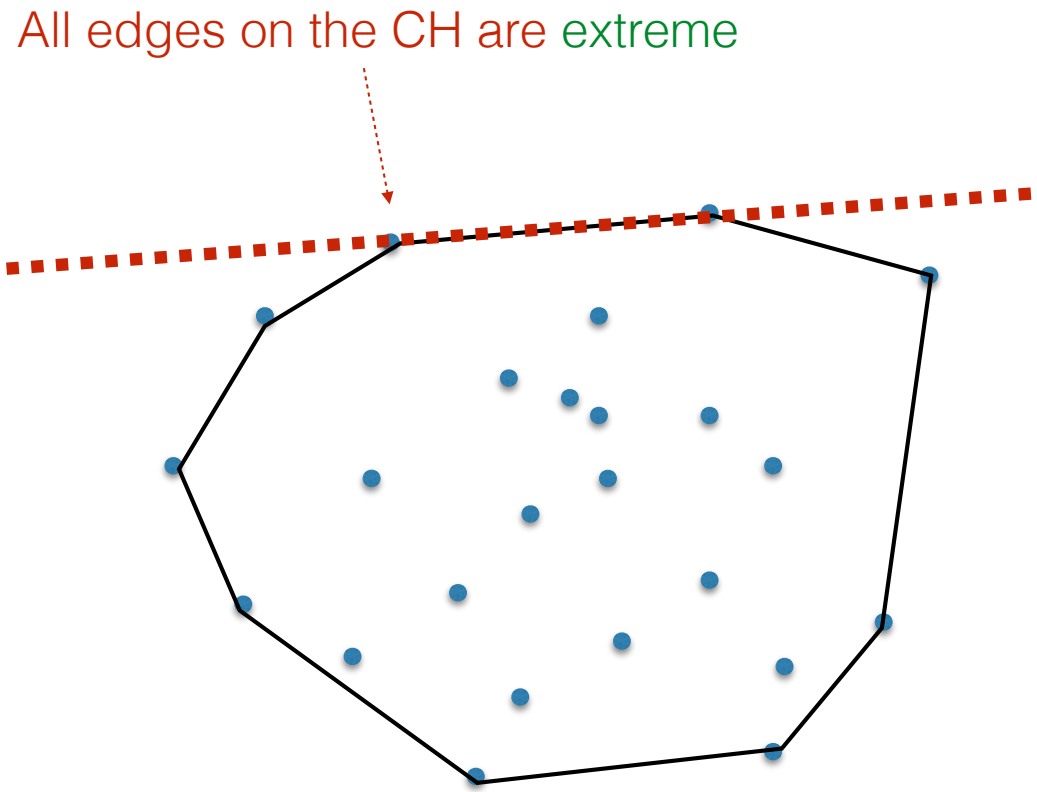


Extreme edges

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



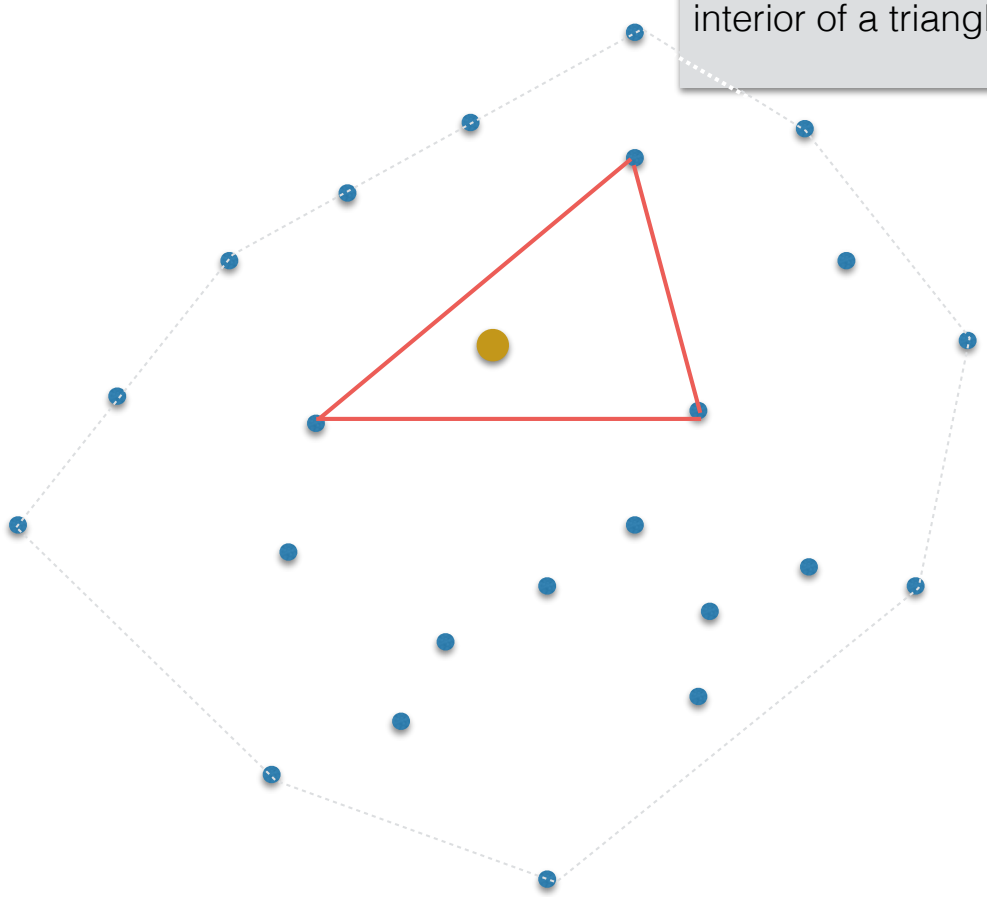
An edge is on the CH <==> it is extreme



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



End review

Algorithms

- Brute force: $O(n^3)$
- Gift wrapping: $O(kn)$
    - 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
- Next
    - Graham scan
    - Quickhull

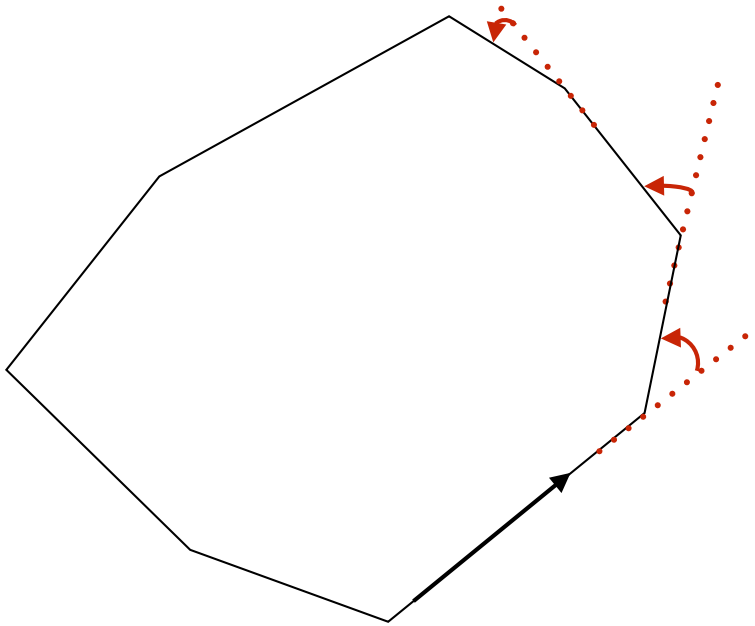
Algorithm:

Graham scan

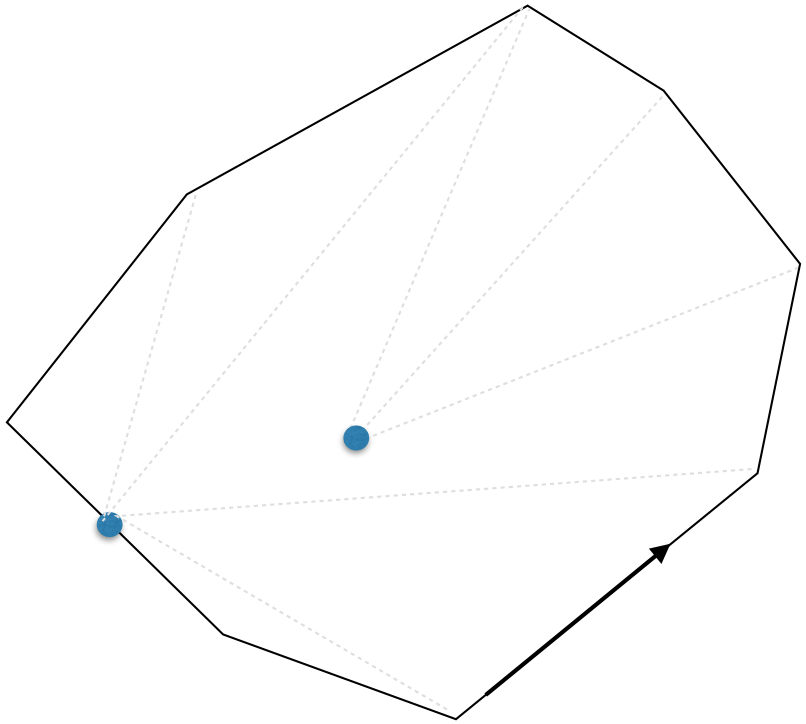
- In late 60s an application at Bell Labs required the hull of 10,000 points, for which a quadratic algorithm was too slow
- Graham developed an algorithm which runs in $O(n \lg n)$
    - It runs in one sort plus a linear pass!!
    - Simple, intuitive, elegant and practical

Ideas

Only left turns!



Walk ccw along the boundary of a convex polygon

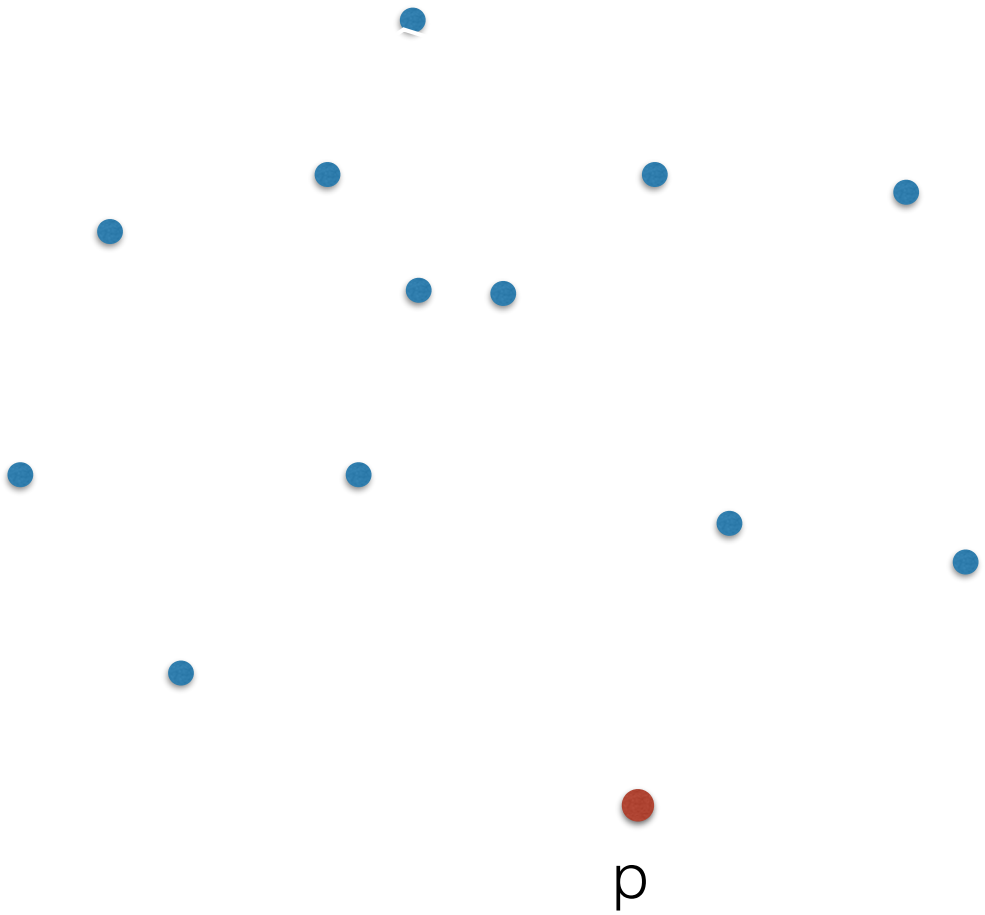


For any point p inside or on the CH, the points on the boundary are in radial order around p

```
//return true if c is (strictly) left of ab, false otherwise
bool left(point2d a, b, c) {
   return two_signed_area(a, b, c) > 0;
}
```

Graham scan

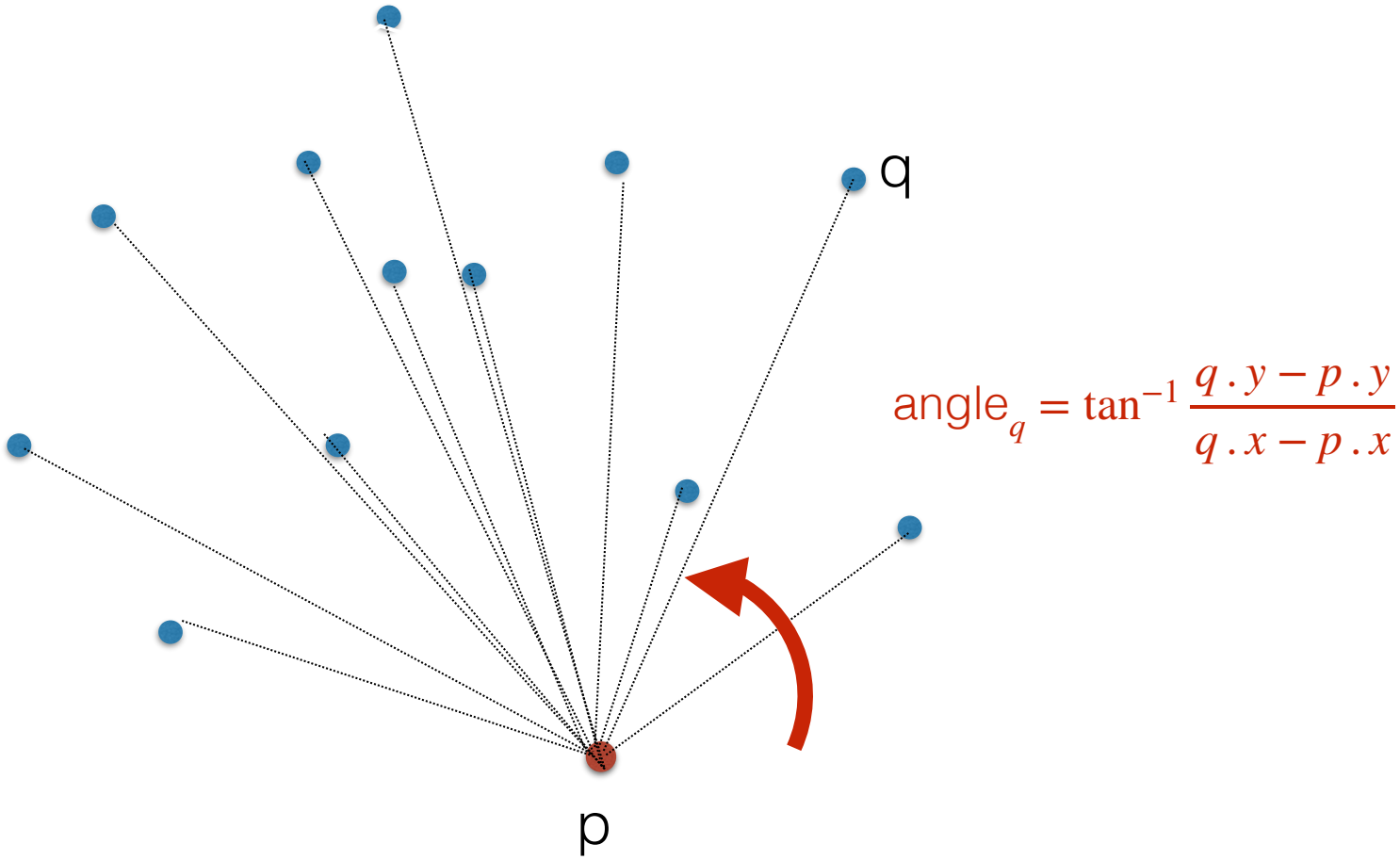
- Idea: start from a point p on the hull (e.g. lowest point)



Graham scan

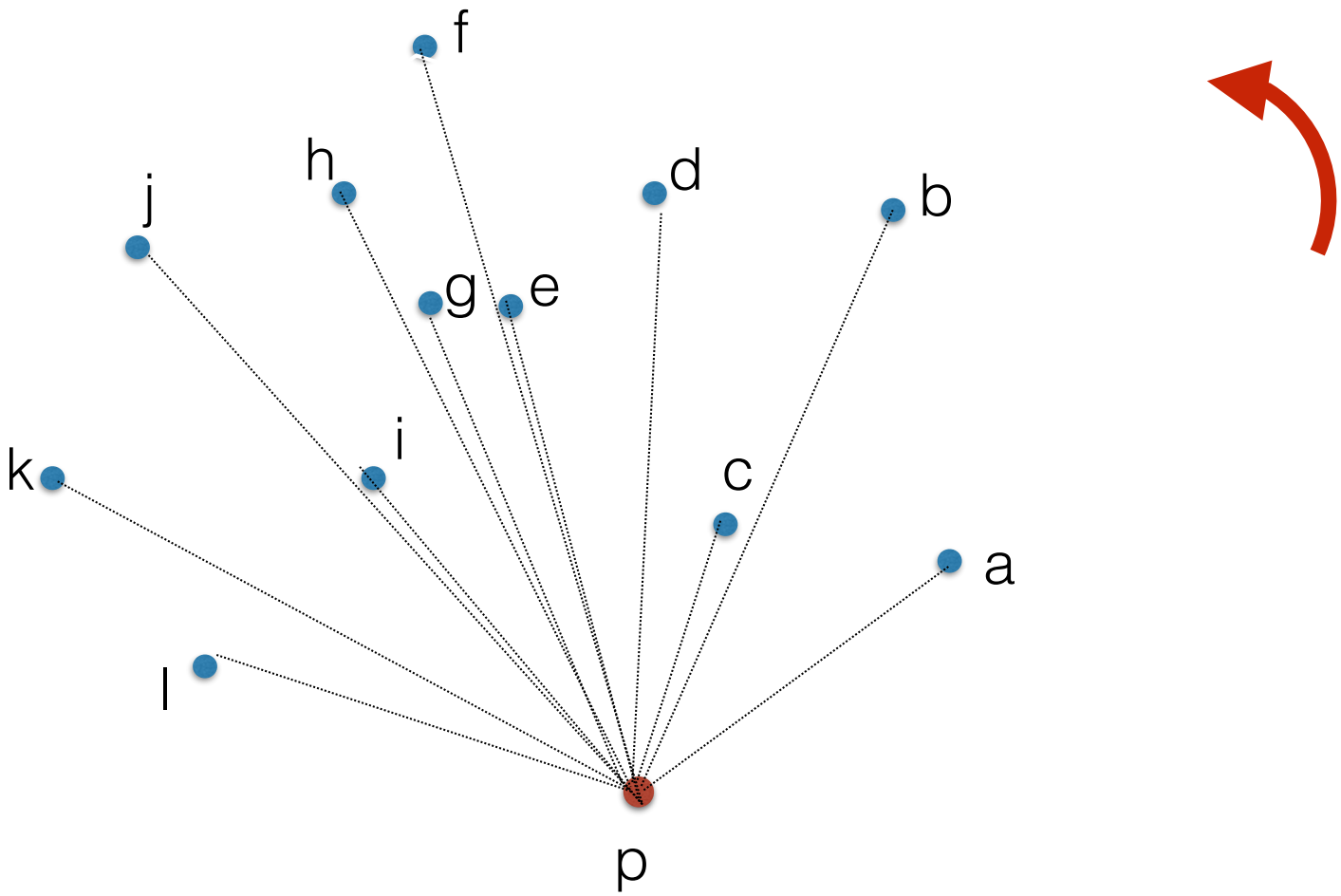
- Idea: start from a point p on the hull (e.g. lowest point)

  order all points by their ccw angle wrt p



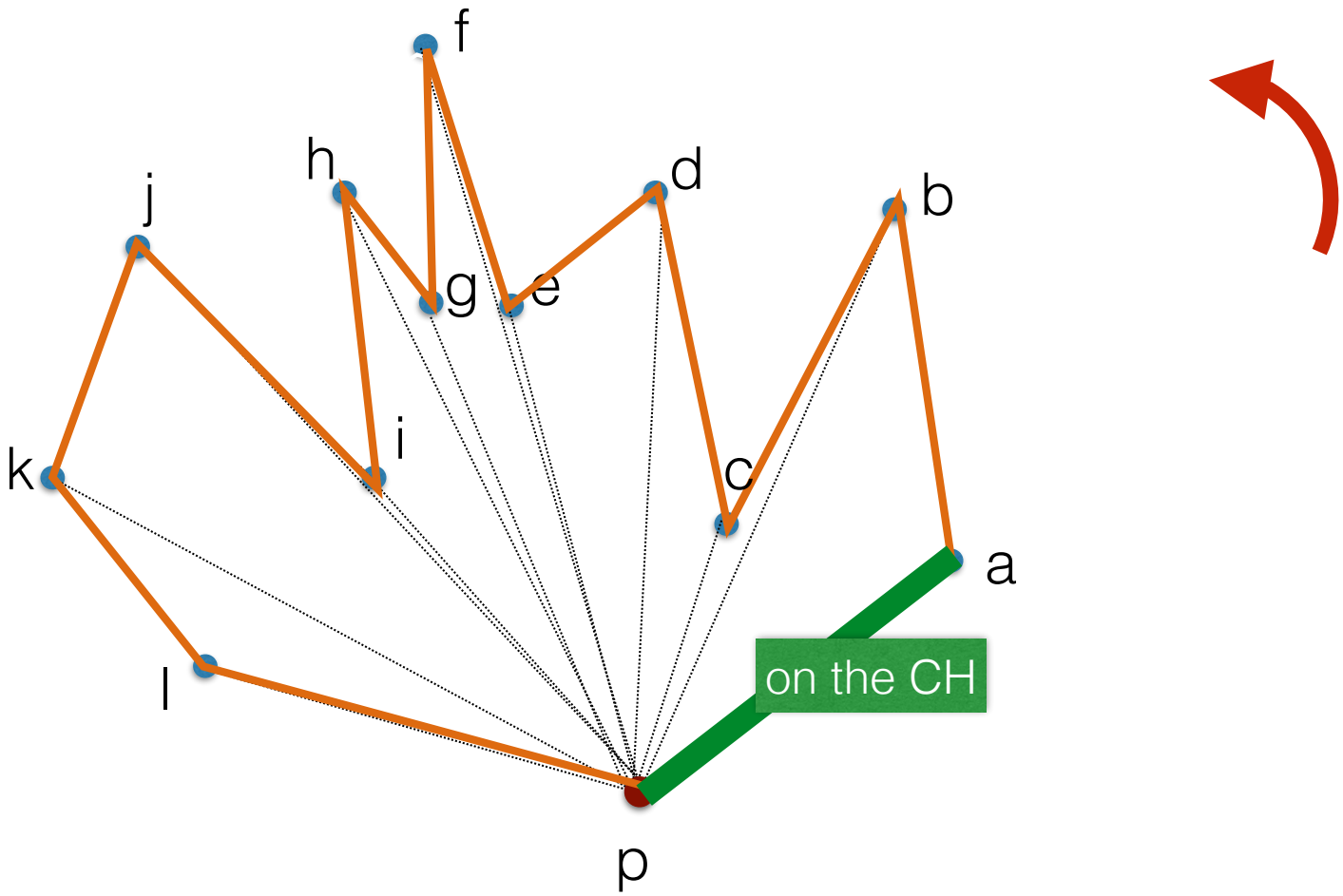
Graham scan

- Idea: traverse the points in this order a, b, c, d, e, f, g, …

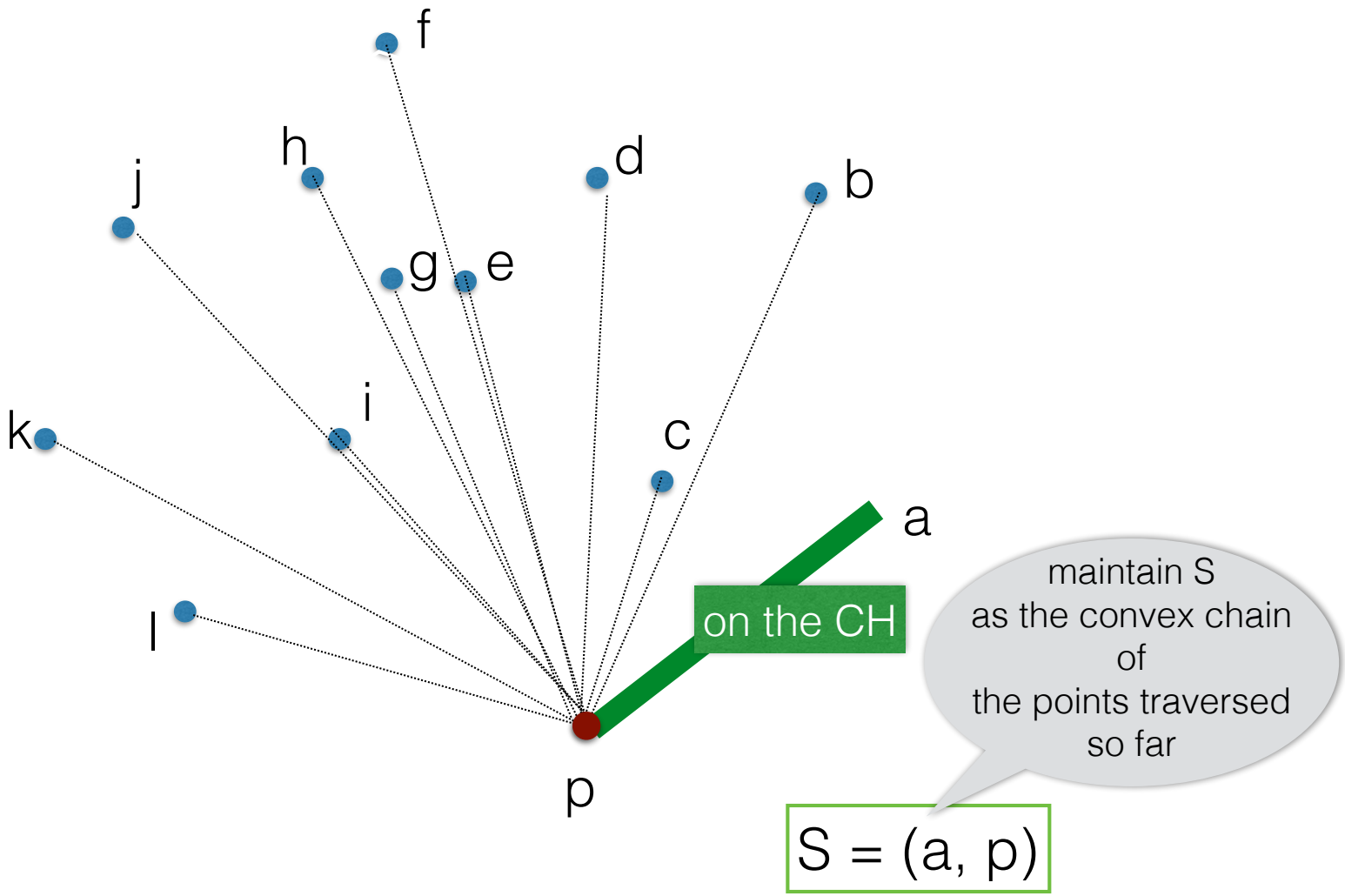


Graham scan

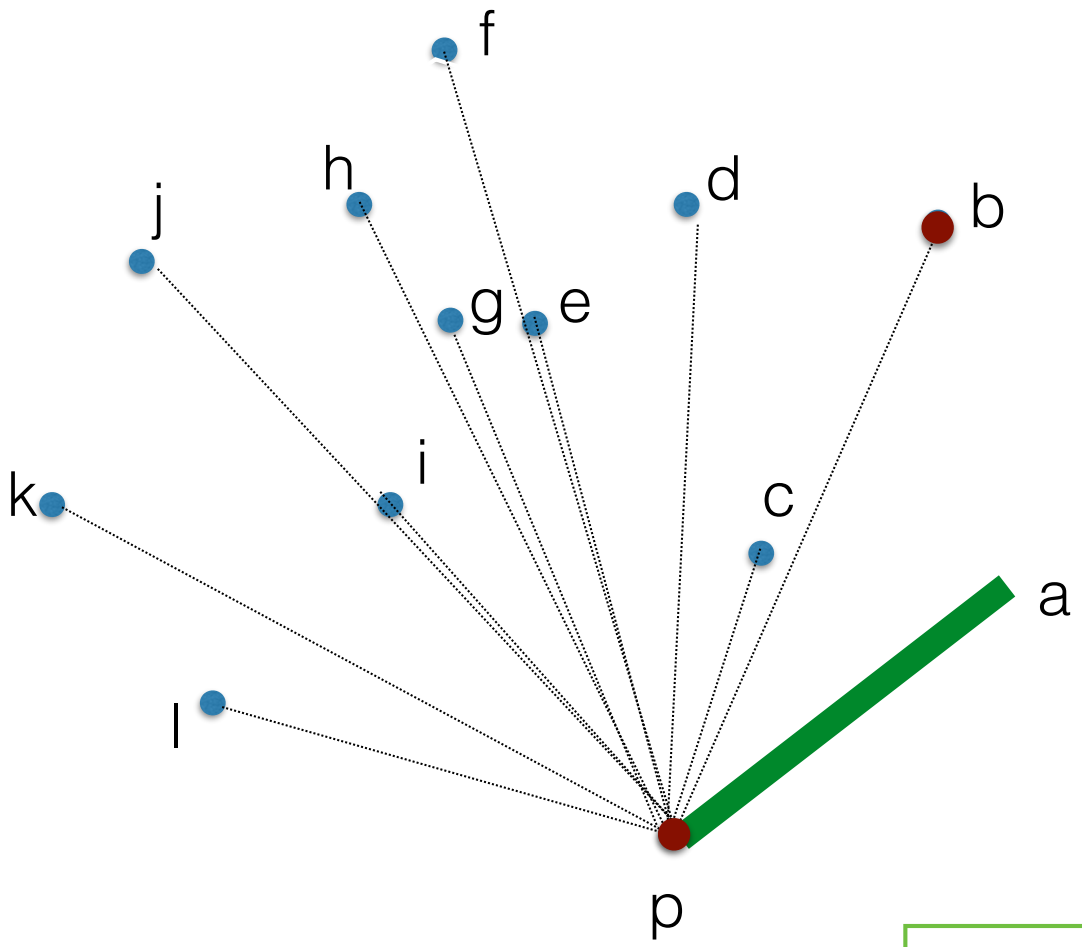
- Idea: traverse the points in this order a, b, c, d, e, f, g, …, making it convex



Graham scan


f
h
d
j
b
g
e
i
k
c
a
l
on the CH
p
maintain S
as the convex chain
of
the points traversed
so far
S = (a, p)

Next point b: what do we do with it?



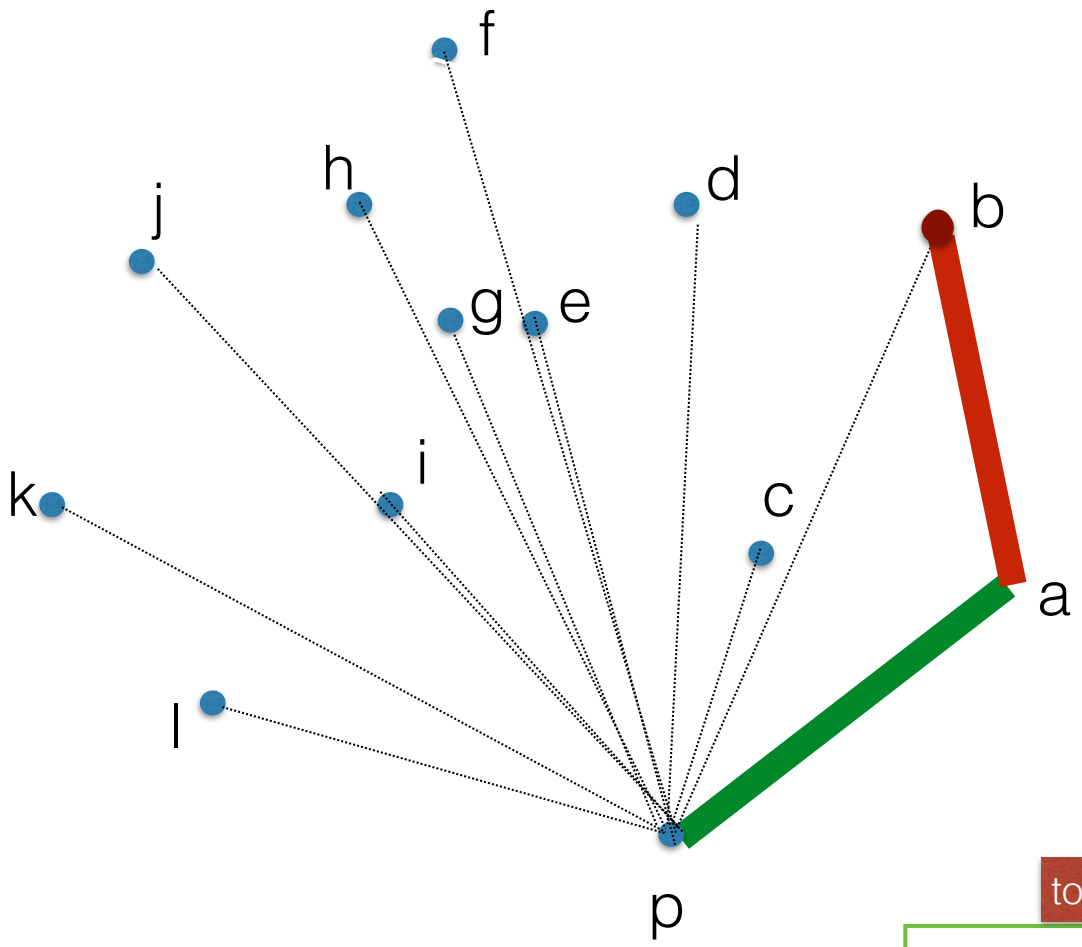
S = (a, p)

is b left of pa?

Next point b: what do we do with it?

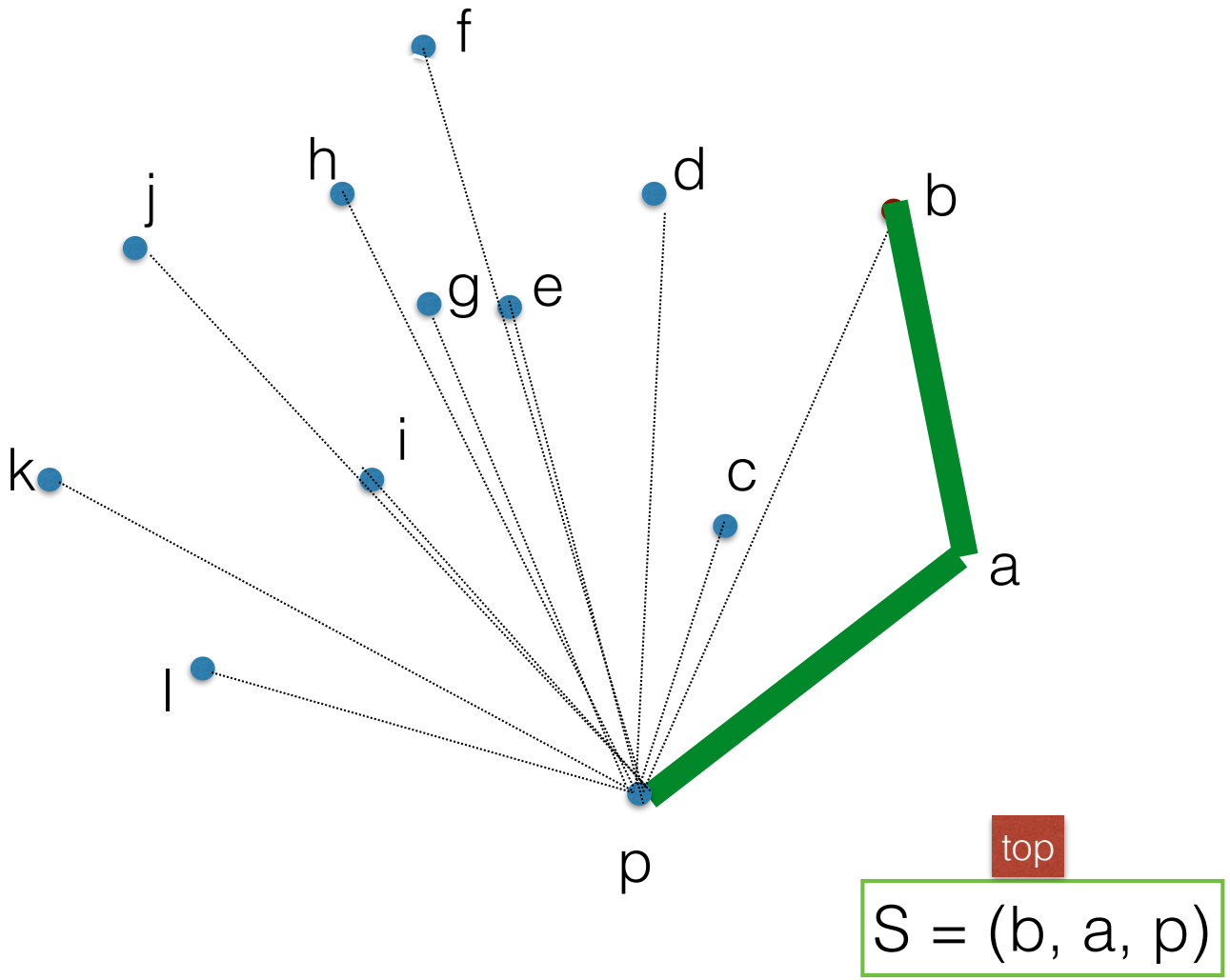
Is b + (a, p) convex? YES!

push b to S



top

S = (a, p)


f
h
d
j
b
g
e
i
k
c
a
l
p
top
S = (b, a, p)

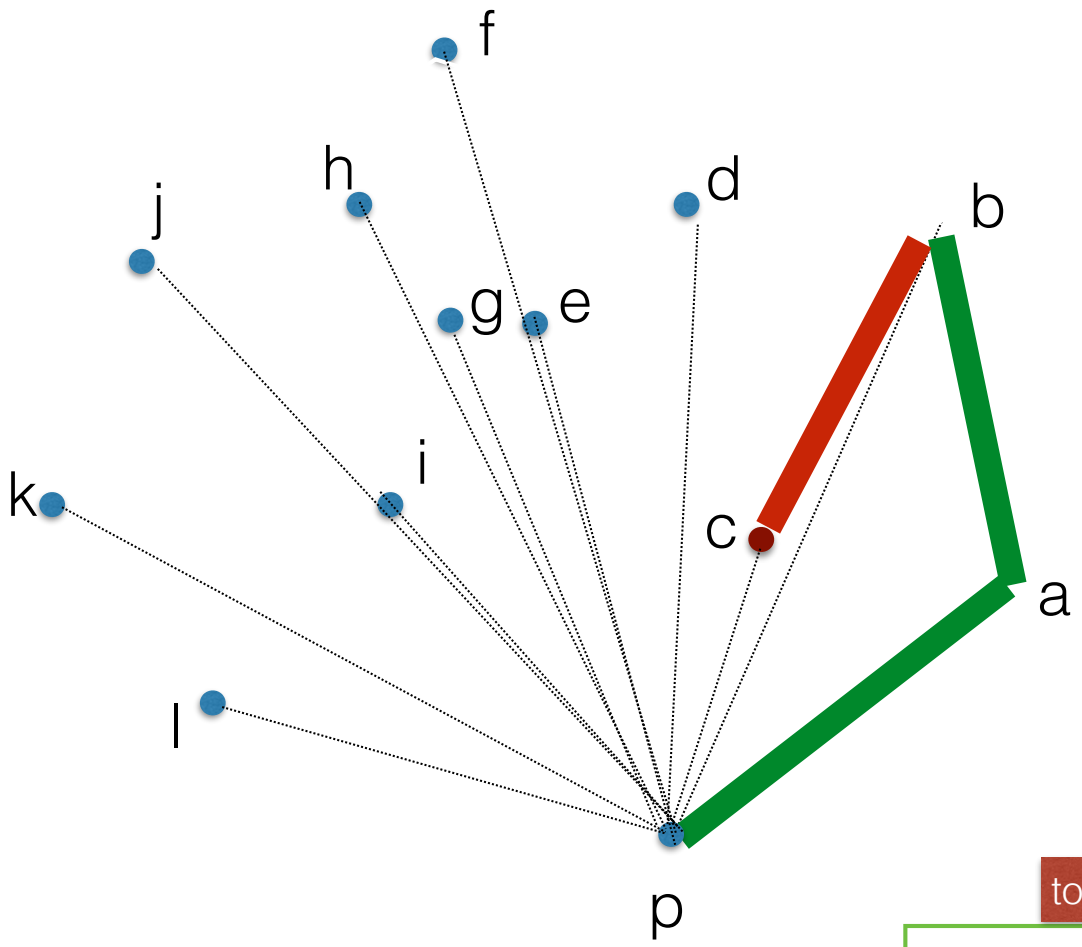
is c left of ab?

Next point c:

Is c + S convex?

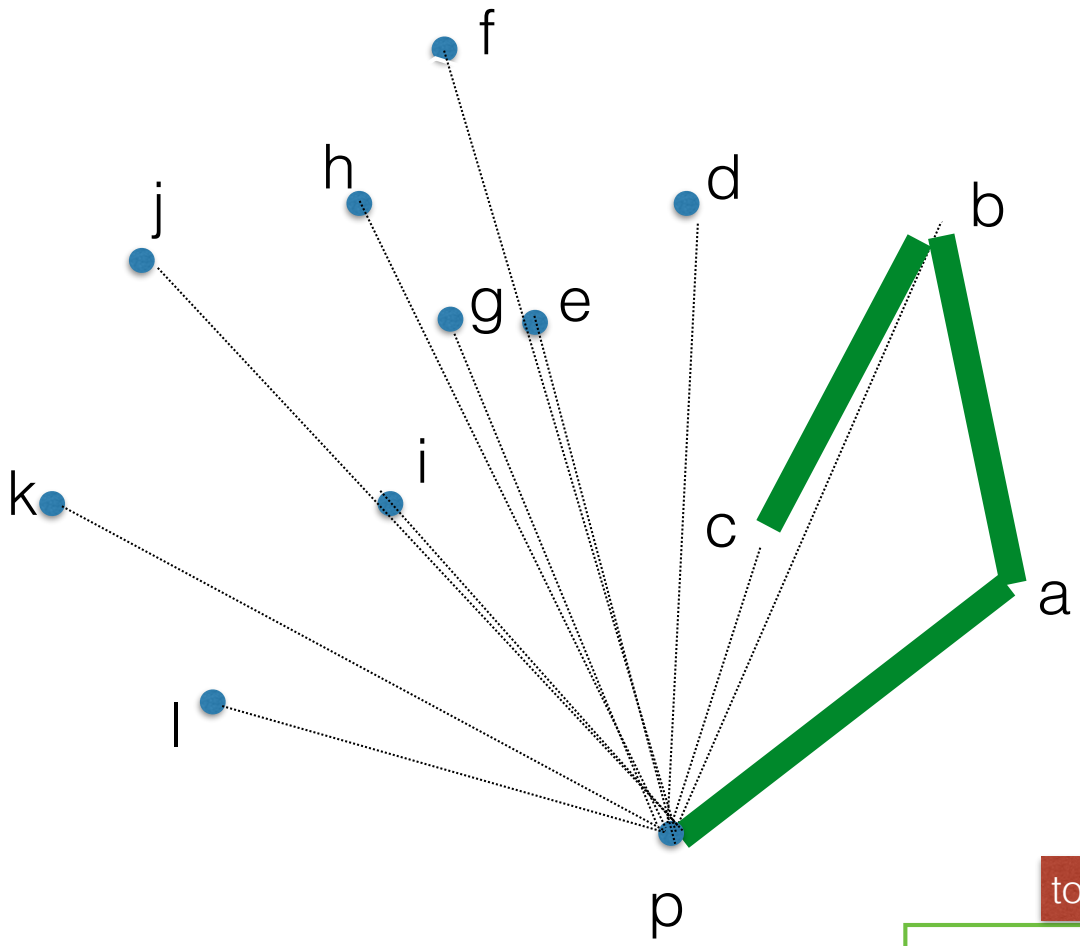
YES!

push c to S



top

S = (b, a, p)



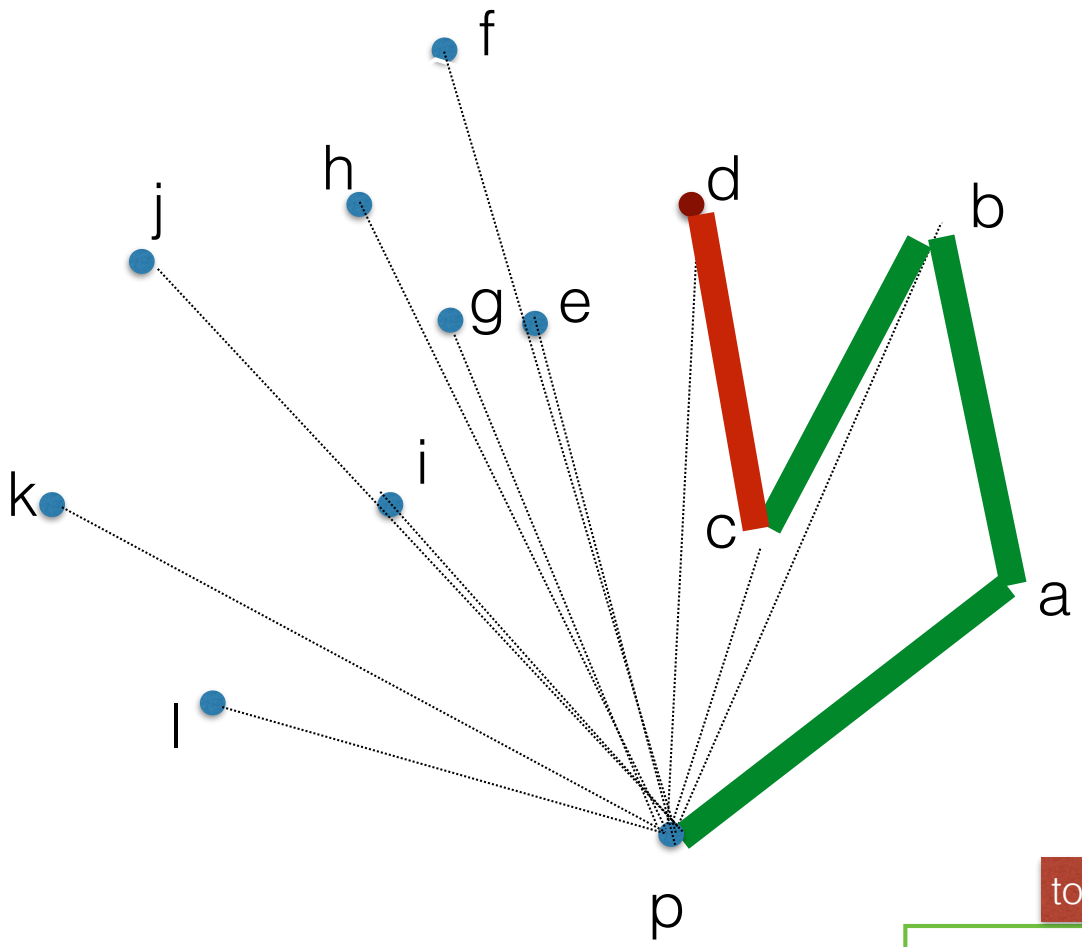
top

S = (c, b, a, p)

is d left of bc?

Is d + S convex? NO!

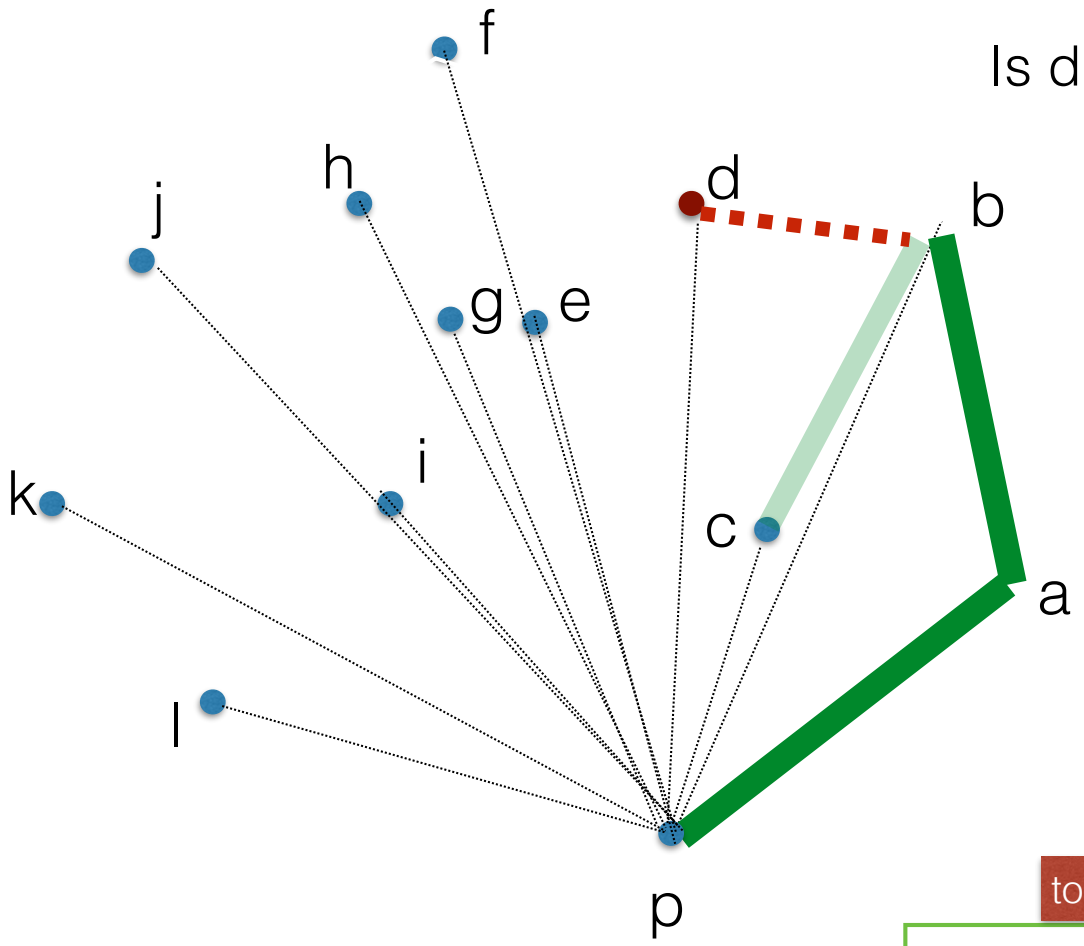
Next point d:



top

S = (c, b, a, p)

Next point d:



is d left of bc?

Is d + S convex?

NO!

pop c

Is d + S convex?

YES!

push d to S

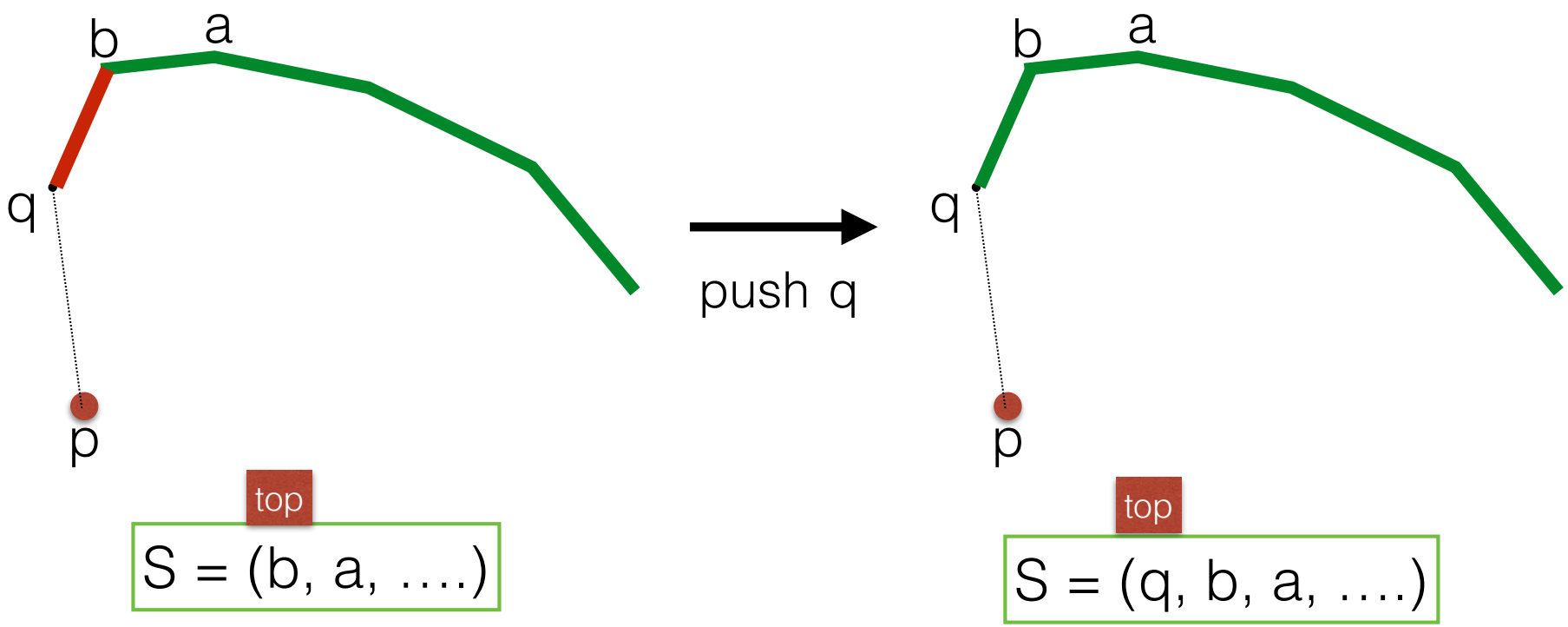
top

S = (d, ~~c~~, b, a, p)

In general

b = top(S), a = second(S)

Next point q: • if q is left of a, b: push(q, S)

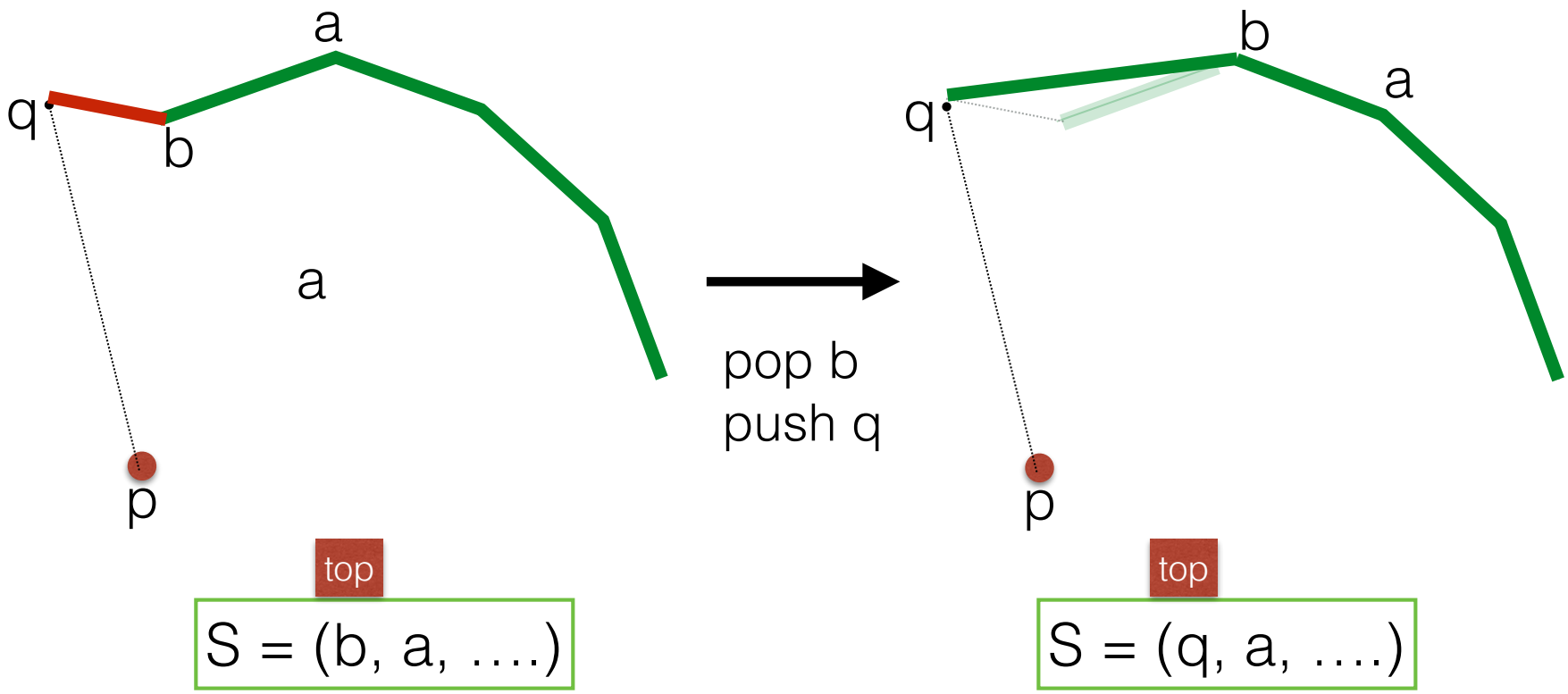


In general

b = top(S), a = second(S)

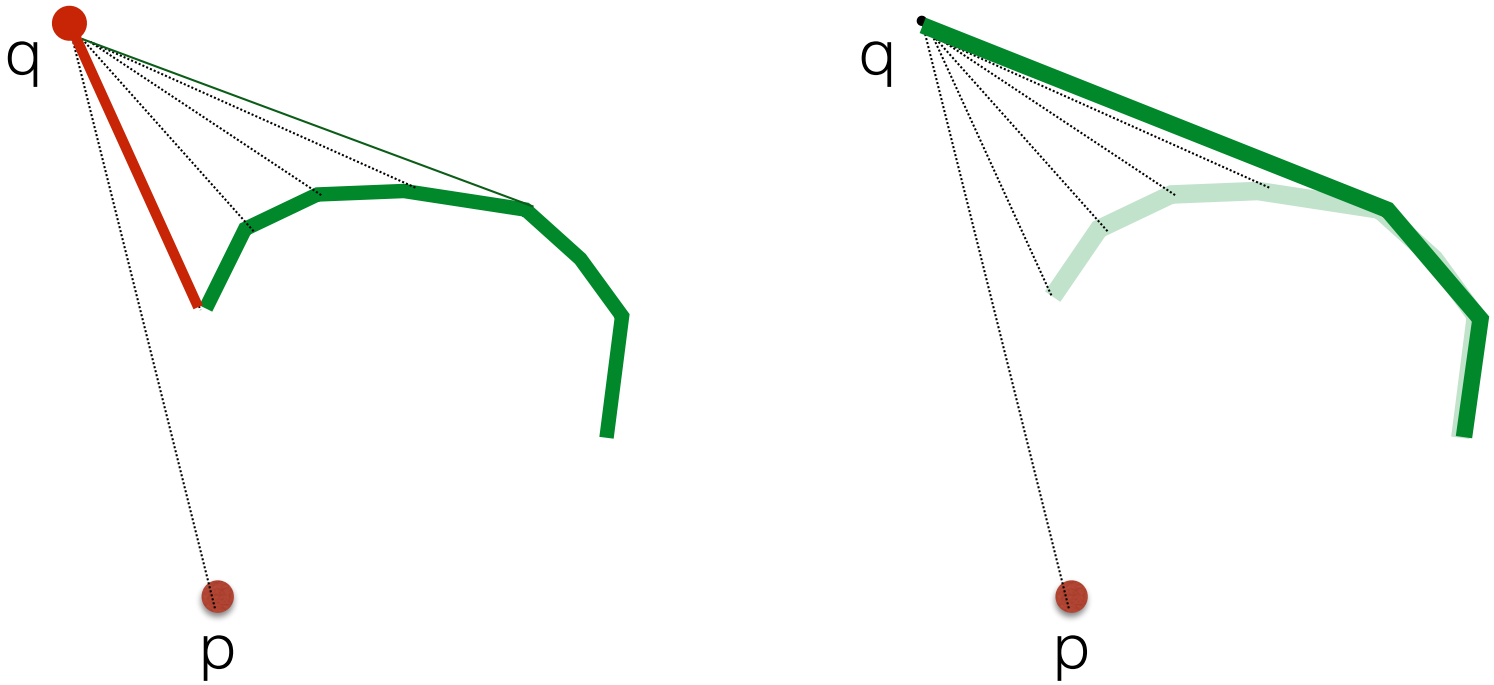
Next point q:

- while q is right of a, b:
  - pop (S)
  - b = top(S), a = second(S)
- push(q, S)



In general

A vertex can trigger more than one pop



Graham scan

- Find lowest point p_0
- Sort all other points ccw around p_0 ← call them $p_1, p_2, p_3, \ldots p_{n-1}$ in this order
- Initialize stack $S = (p_2, p_1)$ (top)
- for $i = 3$ to $n - 1$ do
  - if p_i is left of (second(S),first(S)):
    - push p_i on S
  - else
    - while p_i is right of (second(S), first(S))
      - pop S
    - push p_i on S

note that we are ignoring some details, such as collinear points

Graham scan

- Find lowest point p_0
- Sort all other points ccw around p_0 ← call them $p_1, p_2, p_3, \ldots p_{n-1}$ in this order
- Initialize stack $S = (p_2, p_1)$ (top)
- for $i = 3$ to $n - 1$ do
  - if p_i is left of (second(S),first(S)):
    - push p_i on S
  - else
  - while p_i is right of (second(S), first(S))
    - pop S
  - push p_i on S

ANALYSIS

- Find lowest point p_0
- Sort all other points ccw around p_0 ← $O(n \lg n)$
- Initialize stack $S = (p_2, p_1)$ (top)
- for i=3 to n-1 do
  - while p_i is right of (second(S), first(S))
    - pop(S)
  - push p_i on S

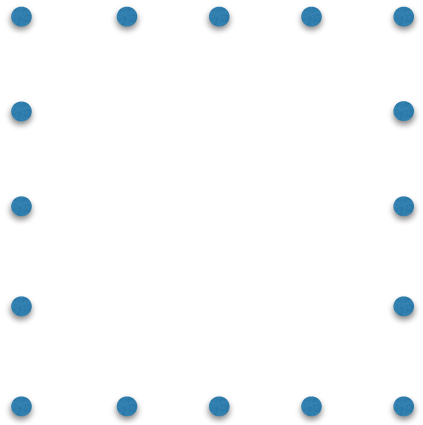
We can show that this takes $O(n)$

Graham-scan runs in $O(n \lg n) + O(n)$

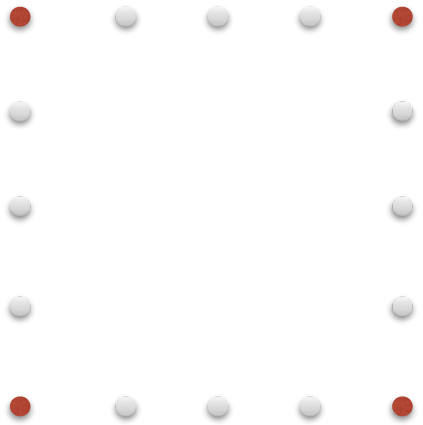
One point can trigger many pops. But we can only pop points that were previously pushed. Every point is pushed once and popped at most once.

Project 2

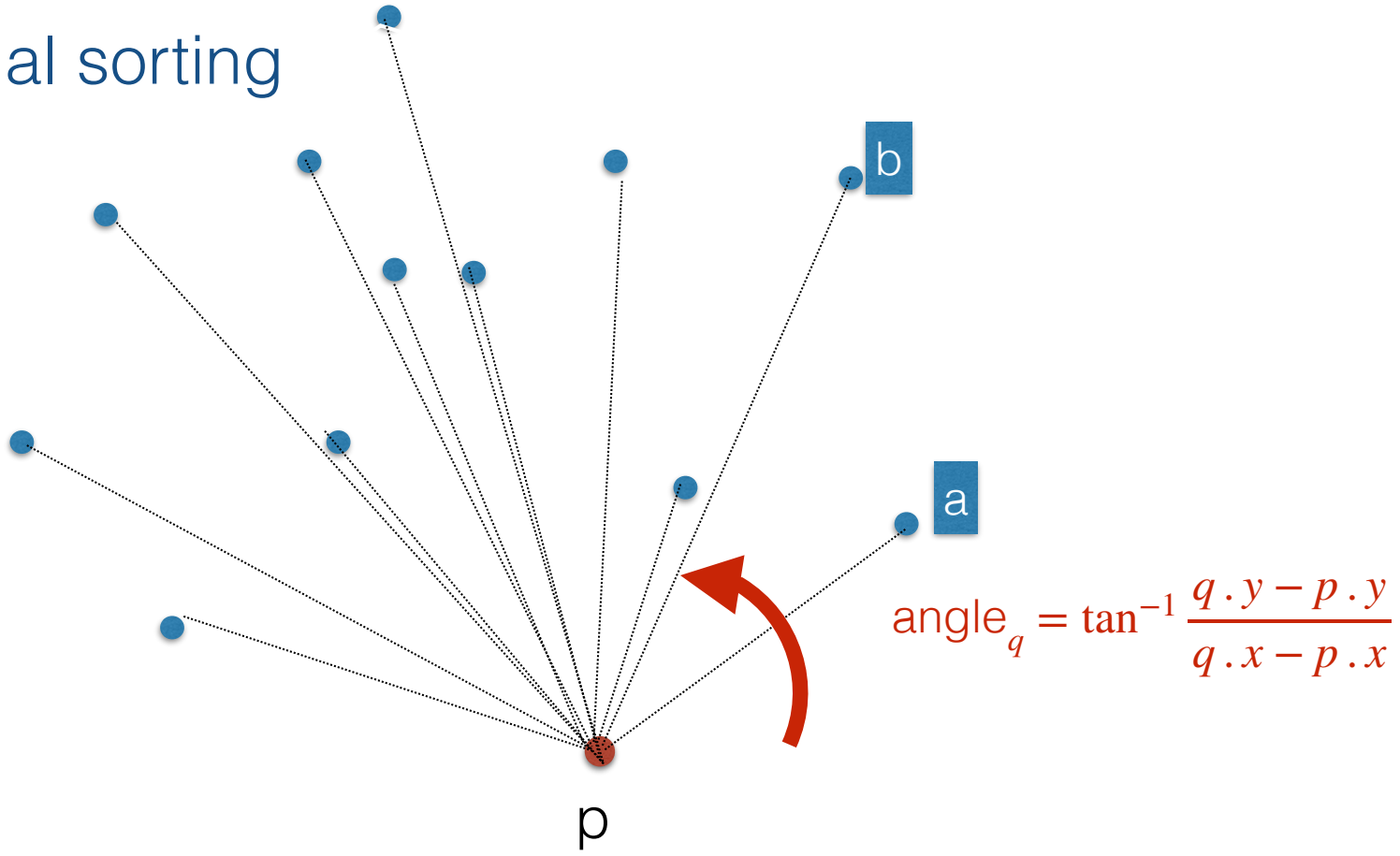
- You'll implement this!
- Along the way you'll get to figure out what situations can cause problems and how to handle them



This is what we want



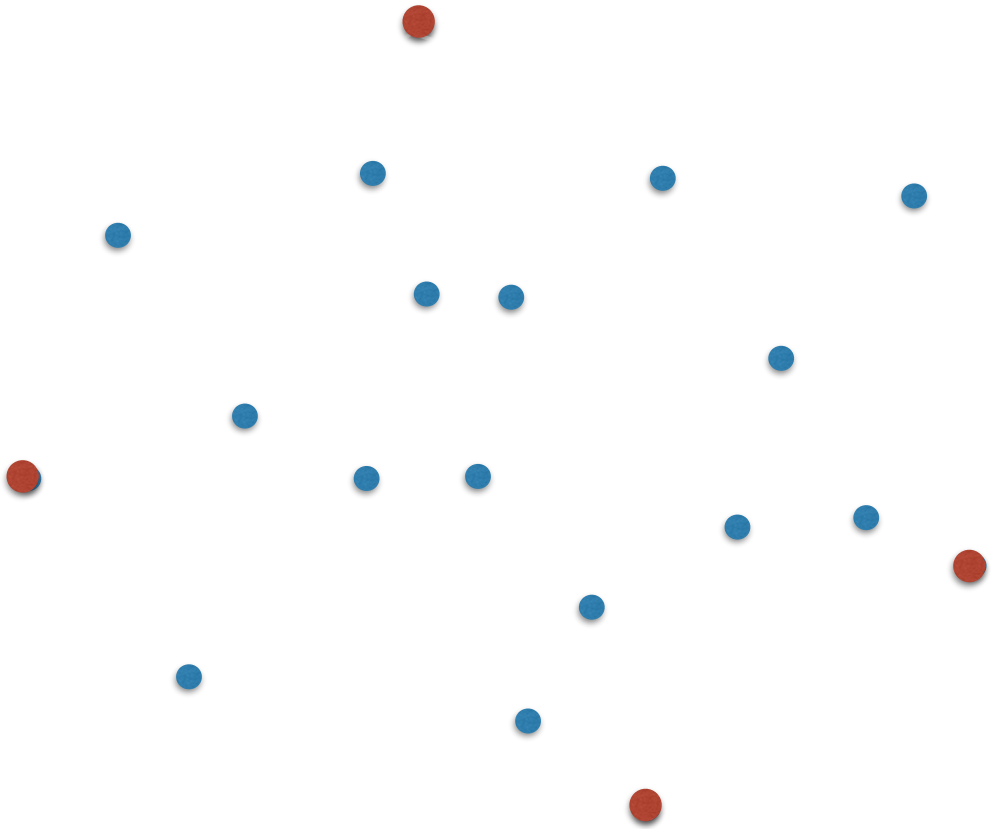
Note on radial sorting



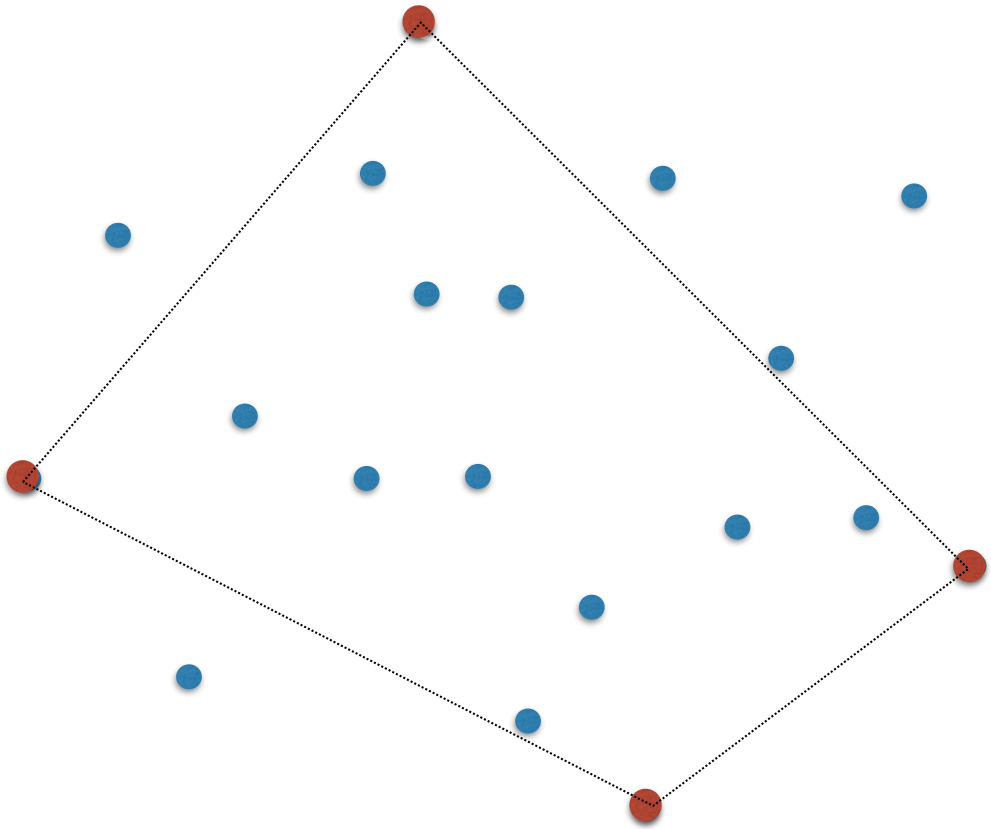
- point a comes before point b in radial order around p if a is right of pb
- rightOf() is to radial sort what < is to sort

And final note..

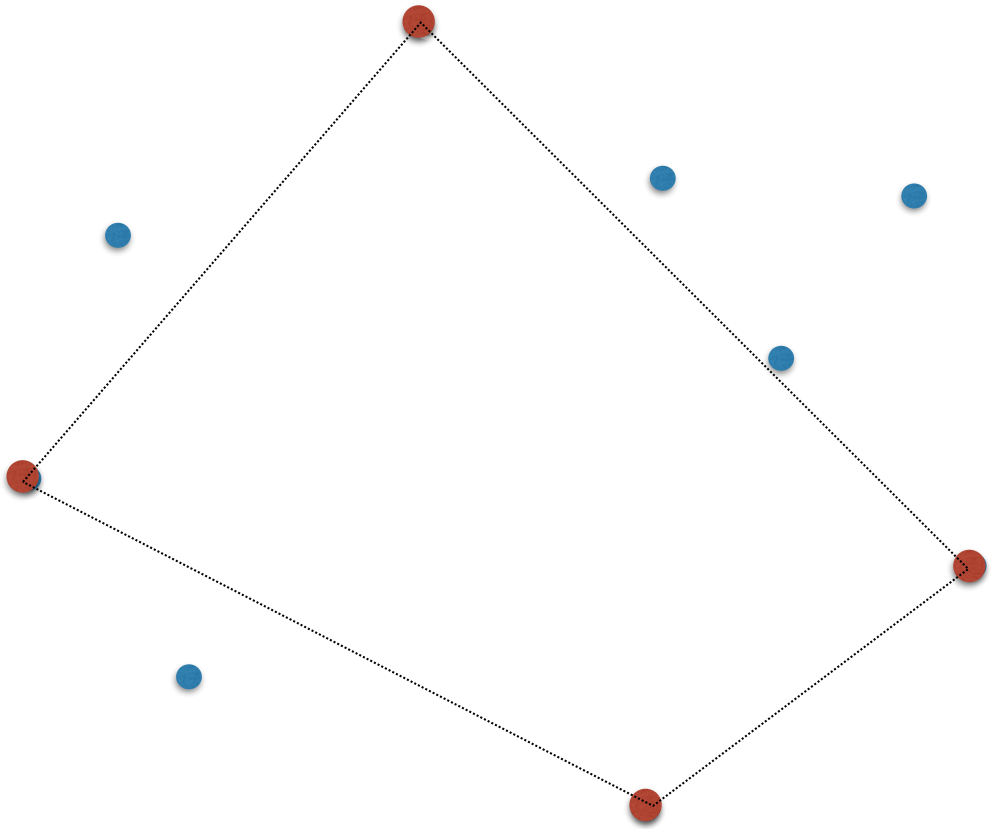
It is possible to speed up Graham scan



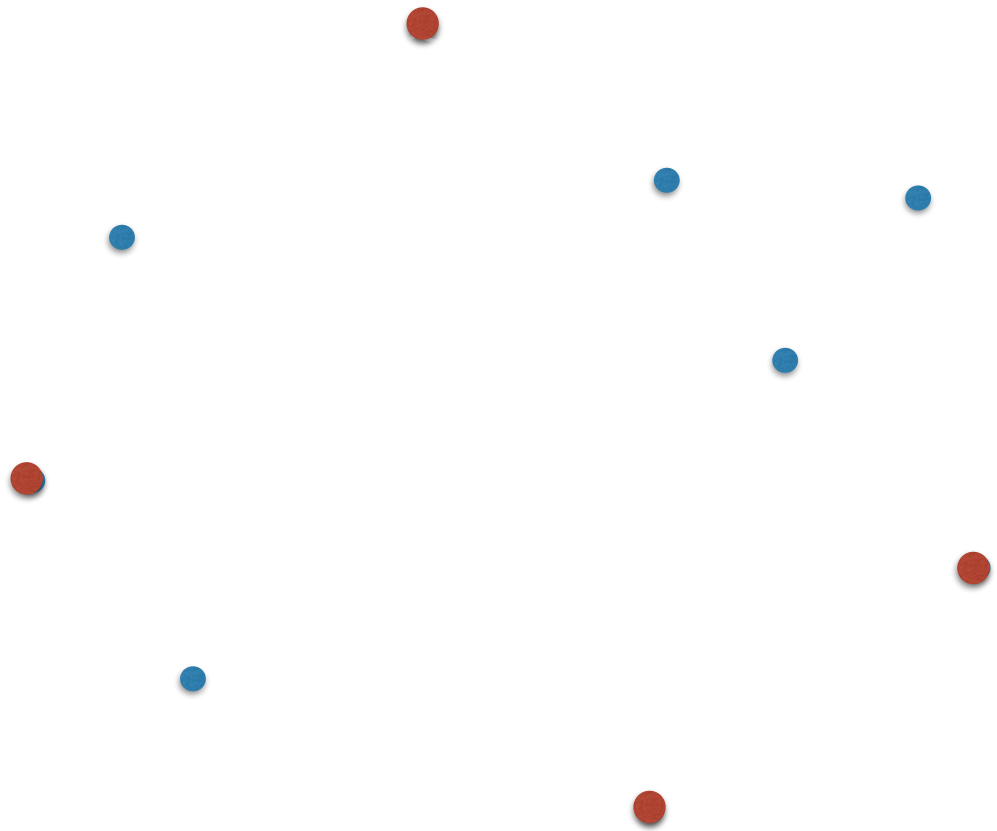
It is possible to speed up Graham scan



It is possible to speed up Graham scan



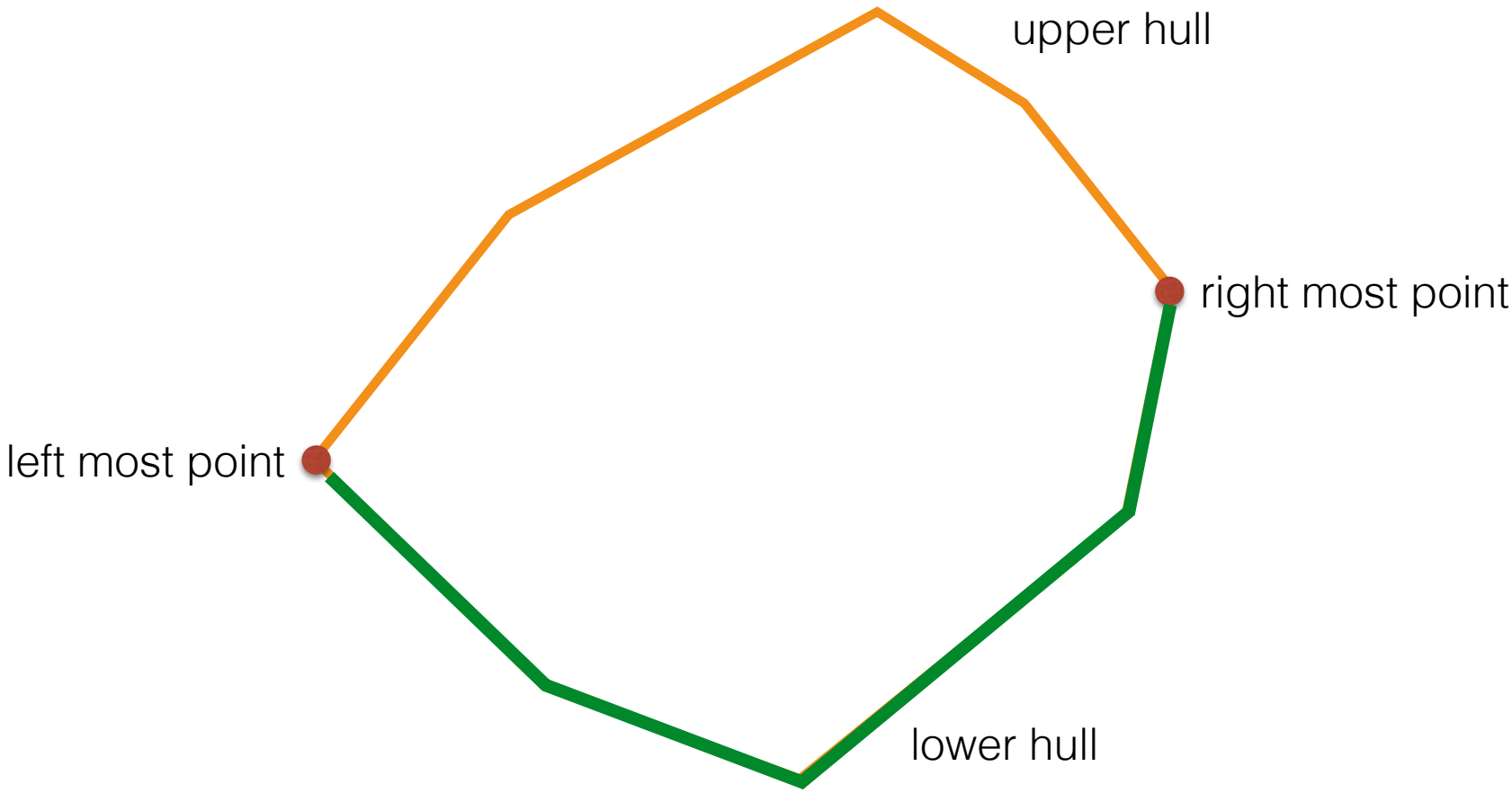
It is possible to speed up Graham scan



Algorithm: Quickhull

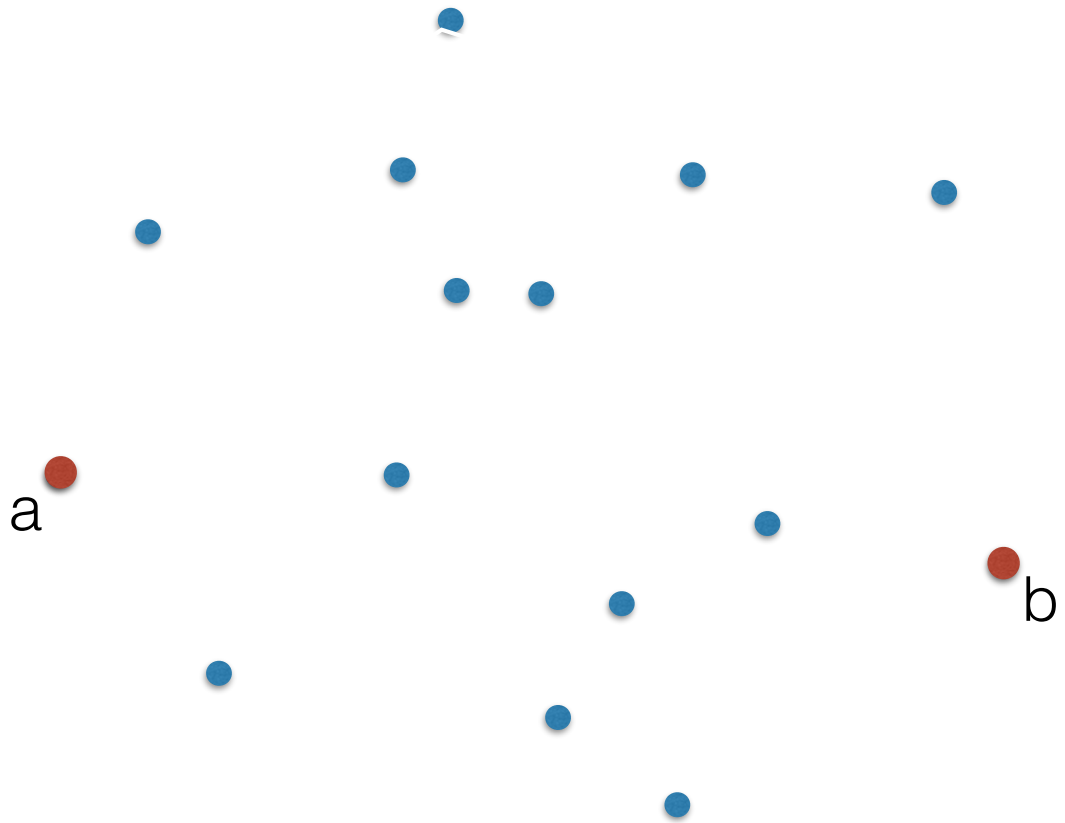
(late 1970s)

Convex polygons: Properties



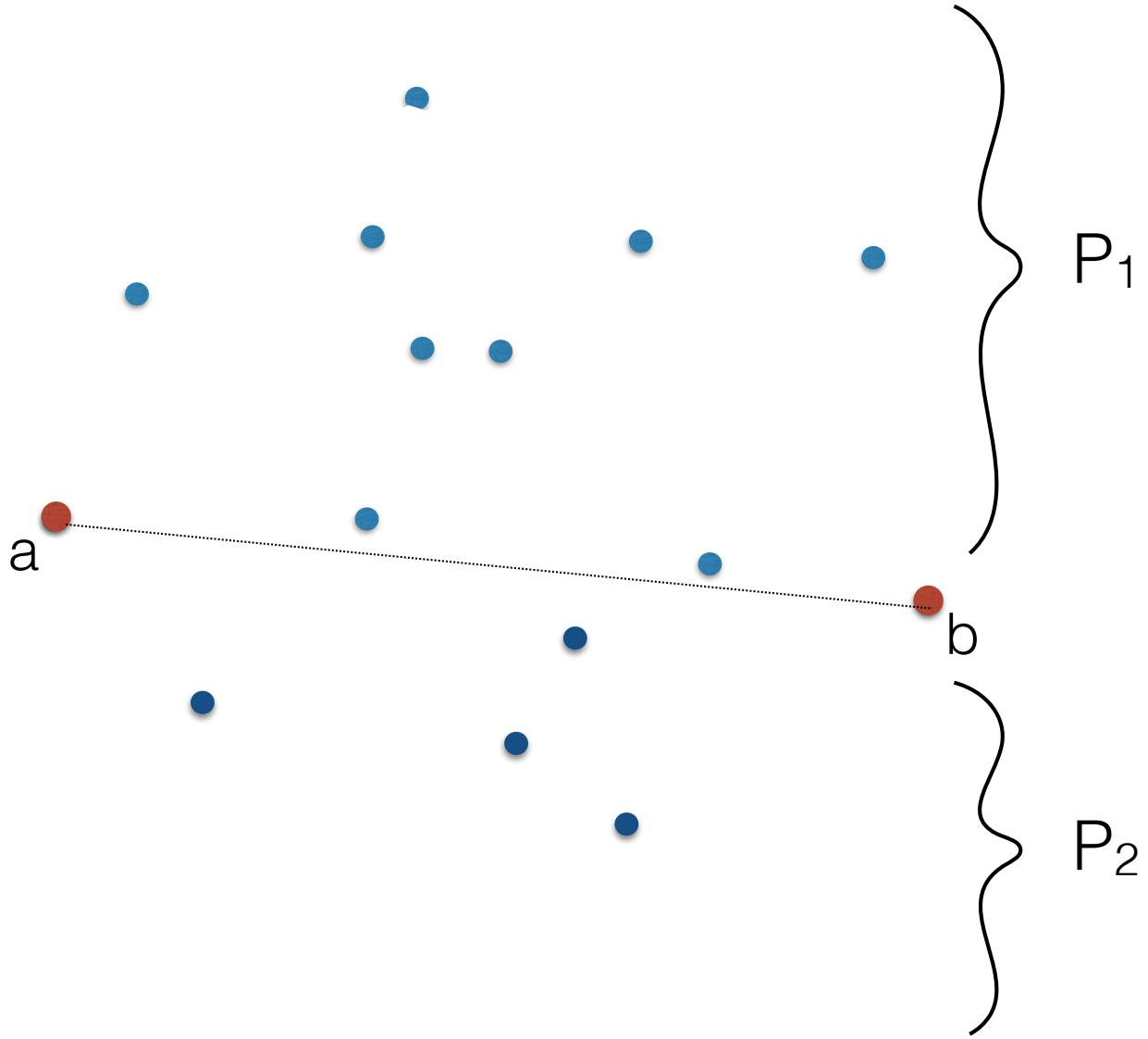
Quickhull (late 1970s)

- Similar to Quicksort
- Idea: start with 2 extreme points



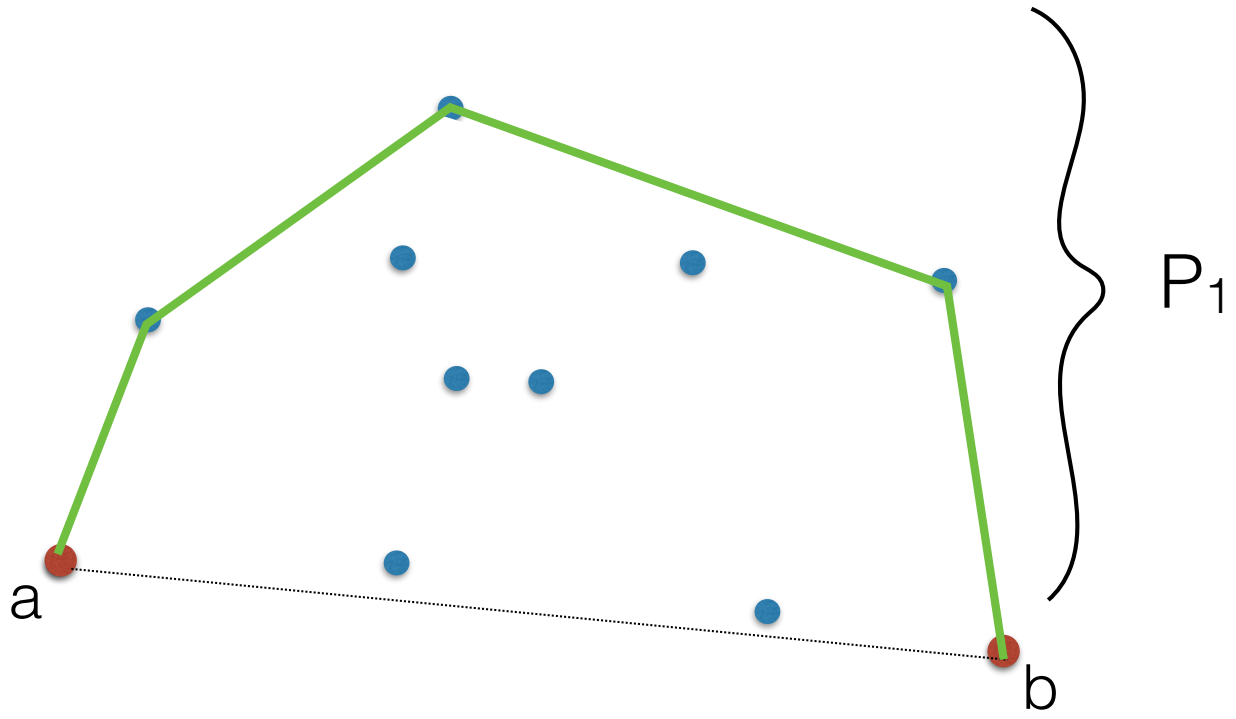
Quickhull

- CH = CH of P_1 (upper hull) + CH of P_2 (lower hull)



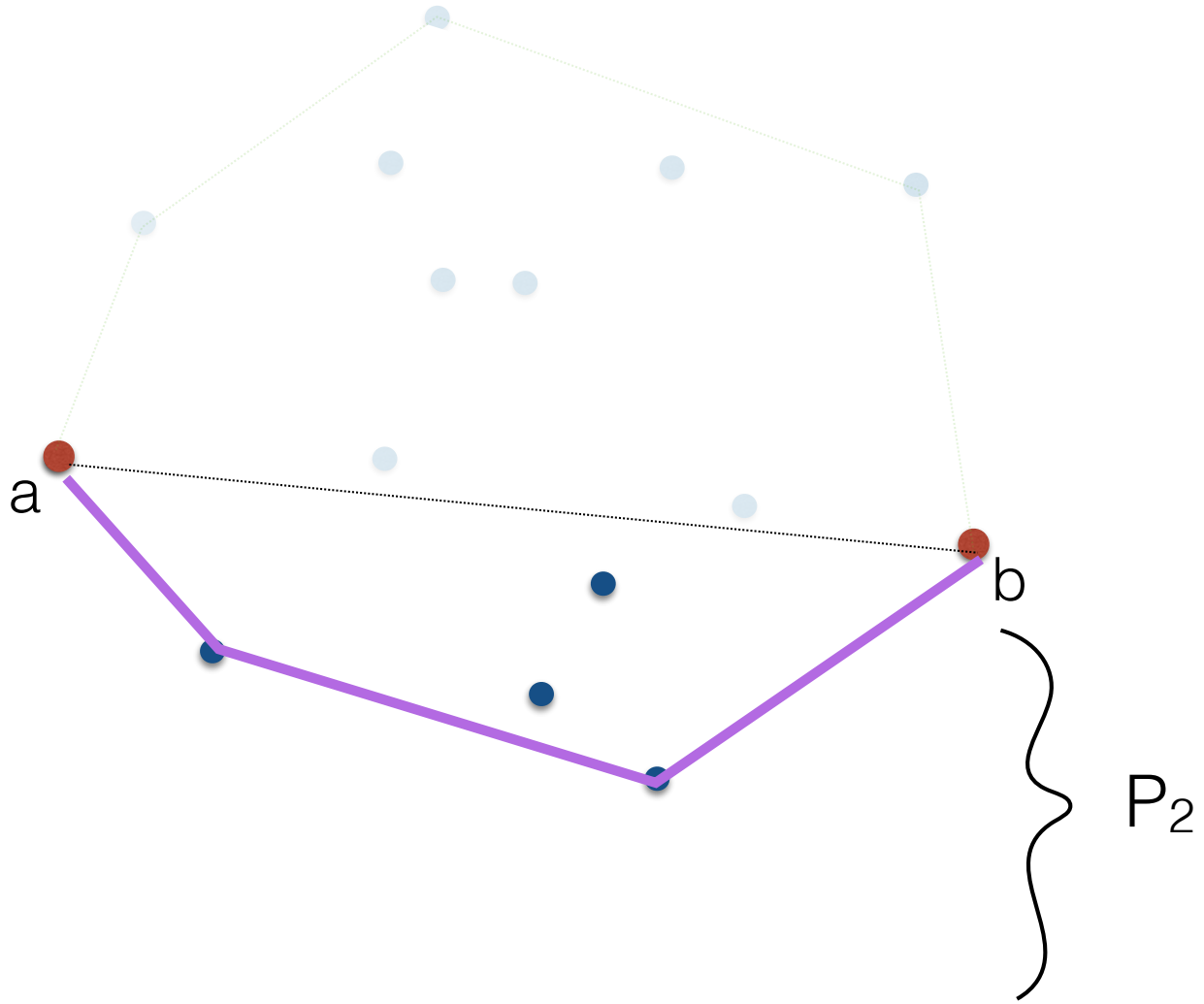
Quickhull

- CH = CH of P_1 (upper hull) + CH of P_2 (lower hull)



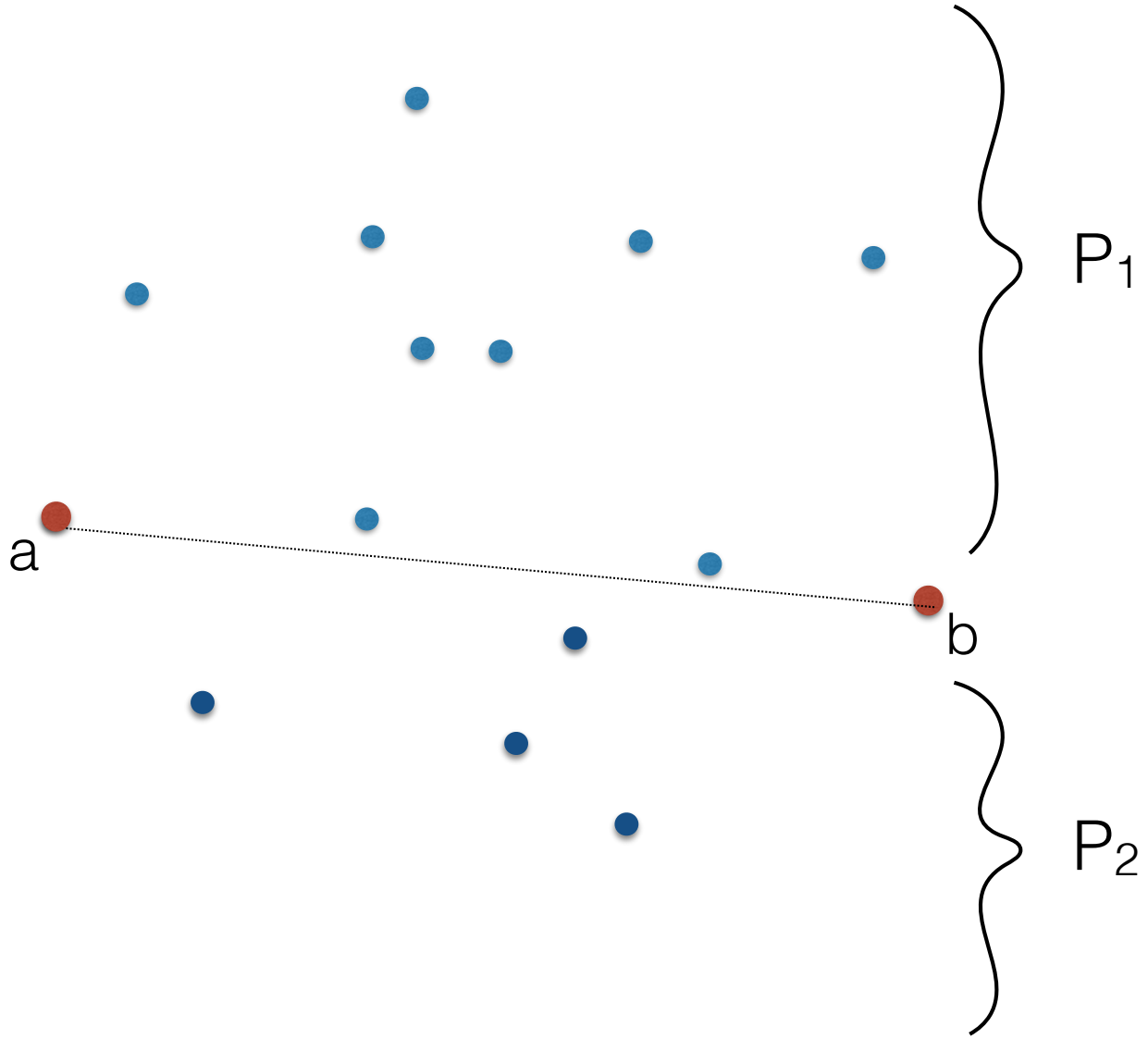
Quickhull

- CH = CH of P_1 (upper hull) + CH of P_2 (lower hull)



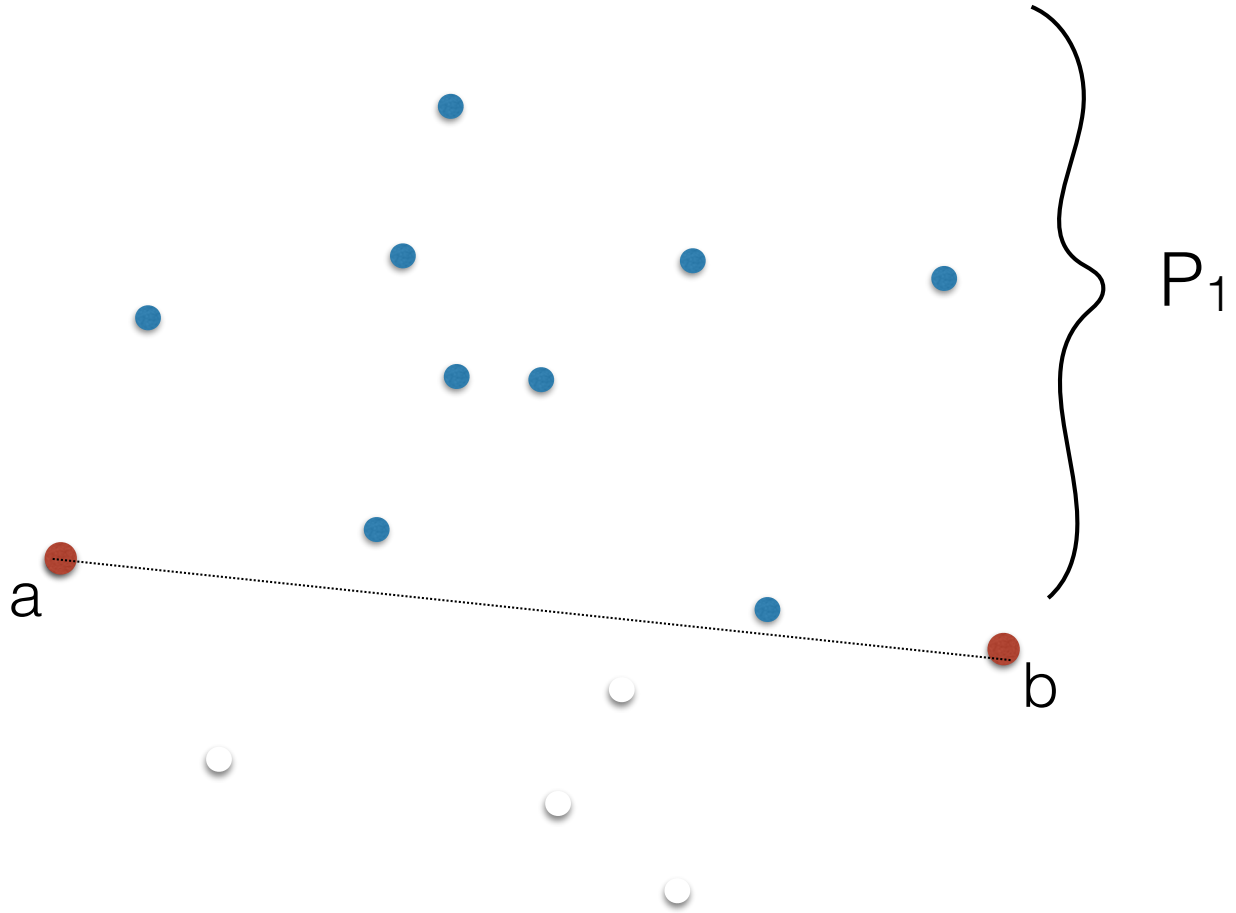
Quickhull

- We'll find the CH(P_1) and CH(P_2) separately



Quickhull

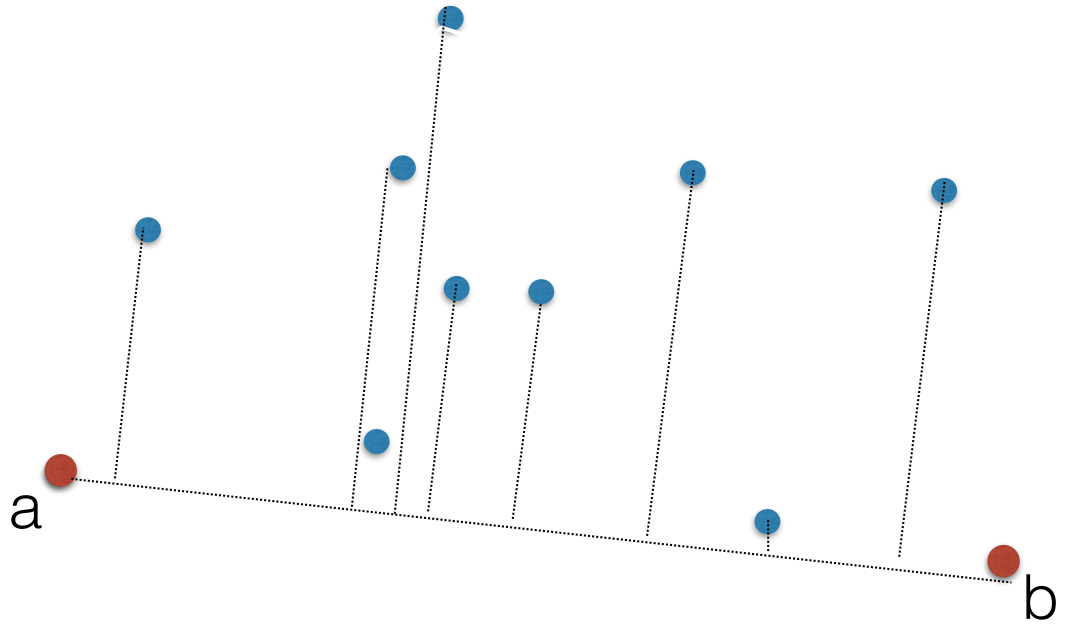
- First let's focus on P1



Quickhull

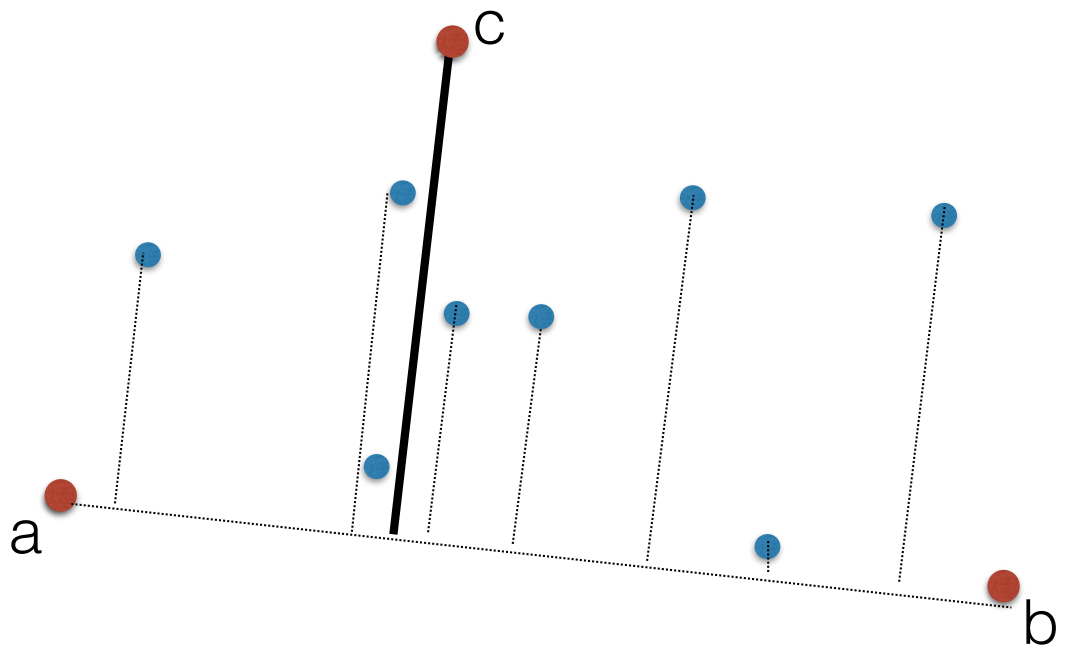
assume no collinear points (for now)

- For all points p in P1: compute dist(p, ab)



Quickhull

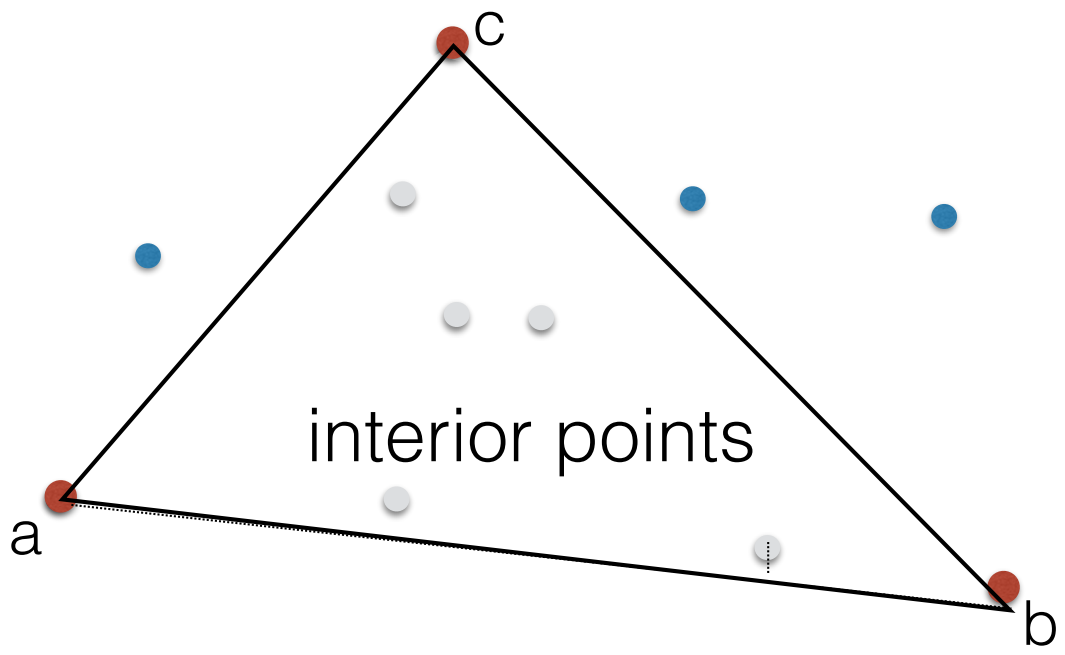
- For all points p in P1: compute dist(p, ab)
- Find the point c with largest distance (i.e. furthest away from ab)



- Claim: Point c must be an extreme point and thus on the CH of P1. (Why?)

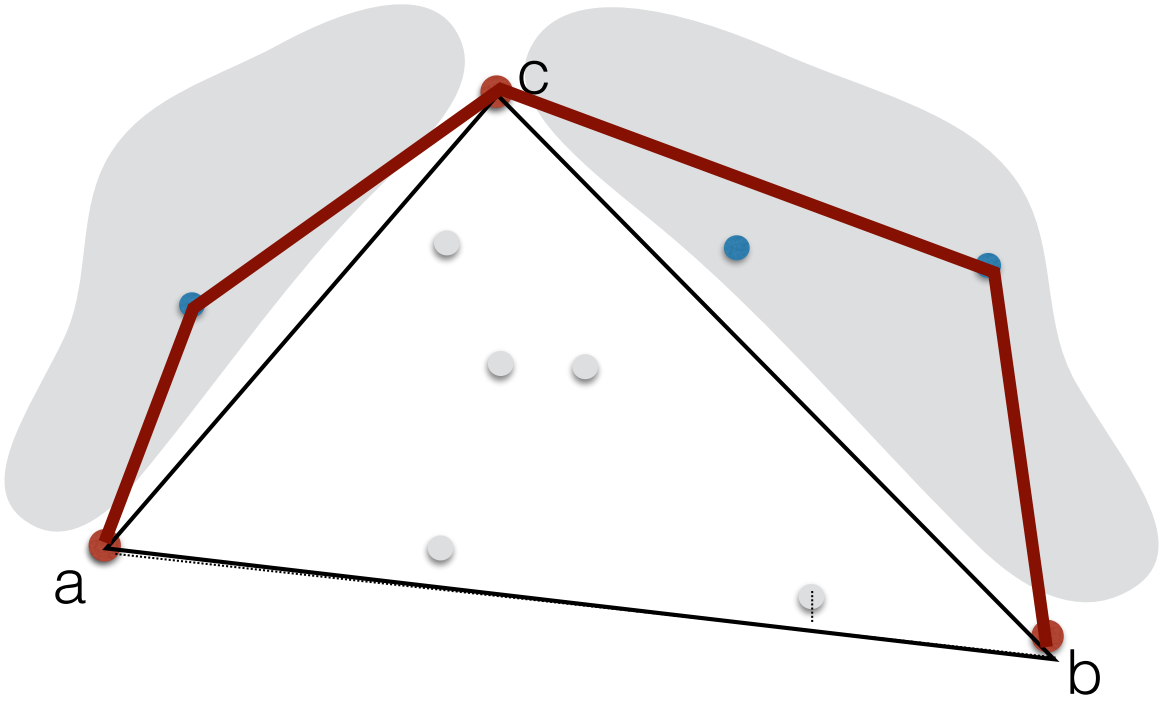
Quickhull

- Discard all points inside triangle abc



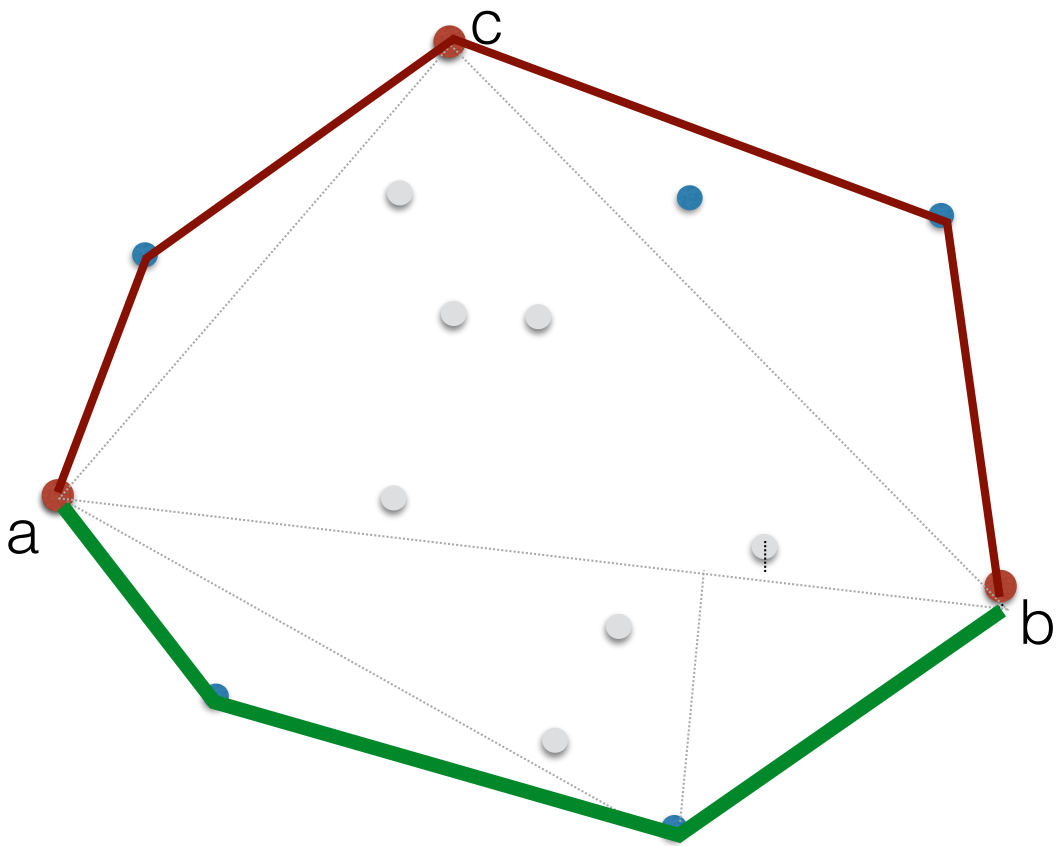
Quickhull

- **Recurse** on the points left of ac and right of bc

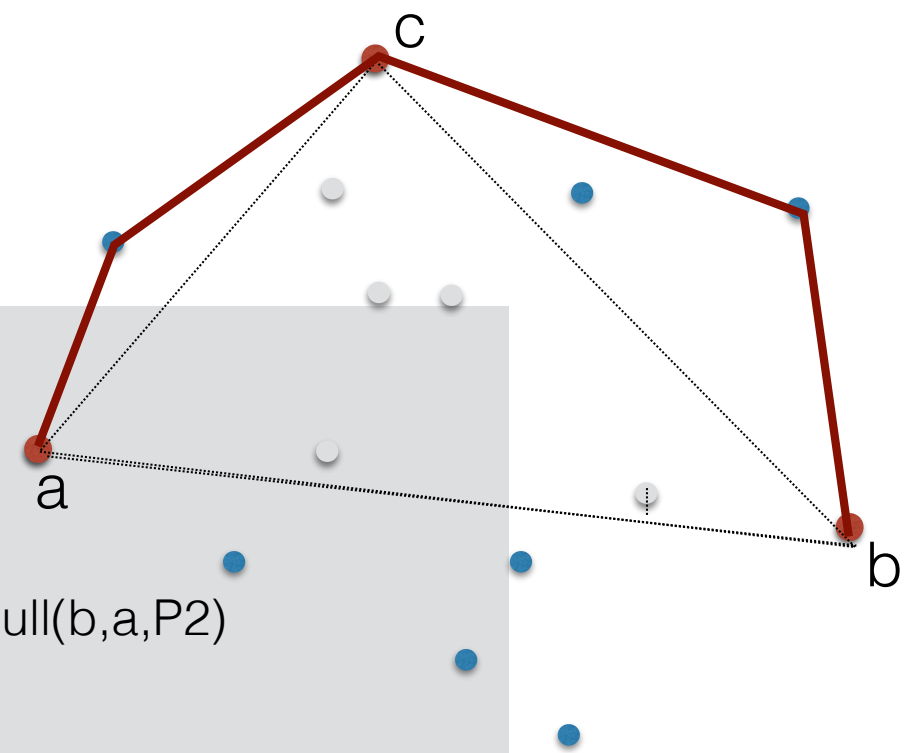


Quickhull

- Compute CH of P_2 similarly



Quickhull



- **Quickhull (P)**
  - find a, b
  - partition P into P1, P2
  - return a + Quickhull(a,b, P1) + b + Quickhull(b,a,P2)

- **Quickhull(a, b, P)**

  //P is a set of points all on the left of ab. return the upper hull of P

  - if P empty => return {}
  - for each point p in P: compute its distance to ab
  - let c = point with max distance
  - let P1 = points to the left of ac
  - let P2 = points to the left of cb
  - return Quickhull(a, c, P1) + c + Quickhull(c, b, P2)

Classwork

- Simulate Quickhull on an arbitrary small set of points (assume no collinearities)
- Analysis:
    - Write a recurrence relation for its running time
    - What/when is the worst case running time ?
    - What/when is the best case running time ?
- Challenge: Argue that Quickhull's average complexity is $O(n)$ on points that are uniformly distributed

Summary

- Brute force: $O(n^3)$
- Gift wrapping: $O(kn)$
  - 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: $O(n \lg n)$, but
  - not output-sensitive
  - does not transfer to 3d
- Quickhull: $O(n^2)$

- Next time
  - incremental, divide-and-conquer
  - $\Omega(n \lg n)$ lower bound

Algorithms

- Brute force: $O(n^3)$
- Gift wrapping: $O(kn)$
  - 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: $O(n \lg n)$, but
  - not output-sensitive
  - does not transfer to 3d
- Quickhull: $O(n^2)$
- Next time
  - incremental, divide-and-conquer
  - $\Omega(n \lg n)$ lower bound