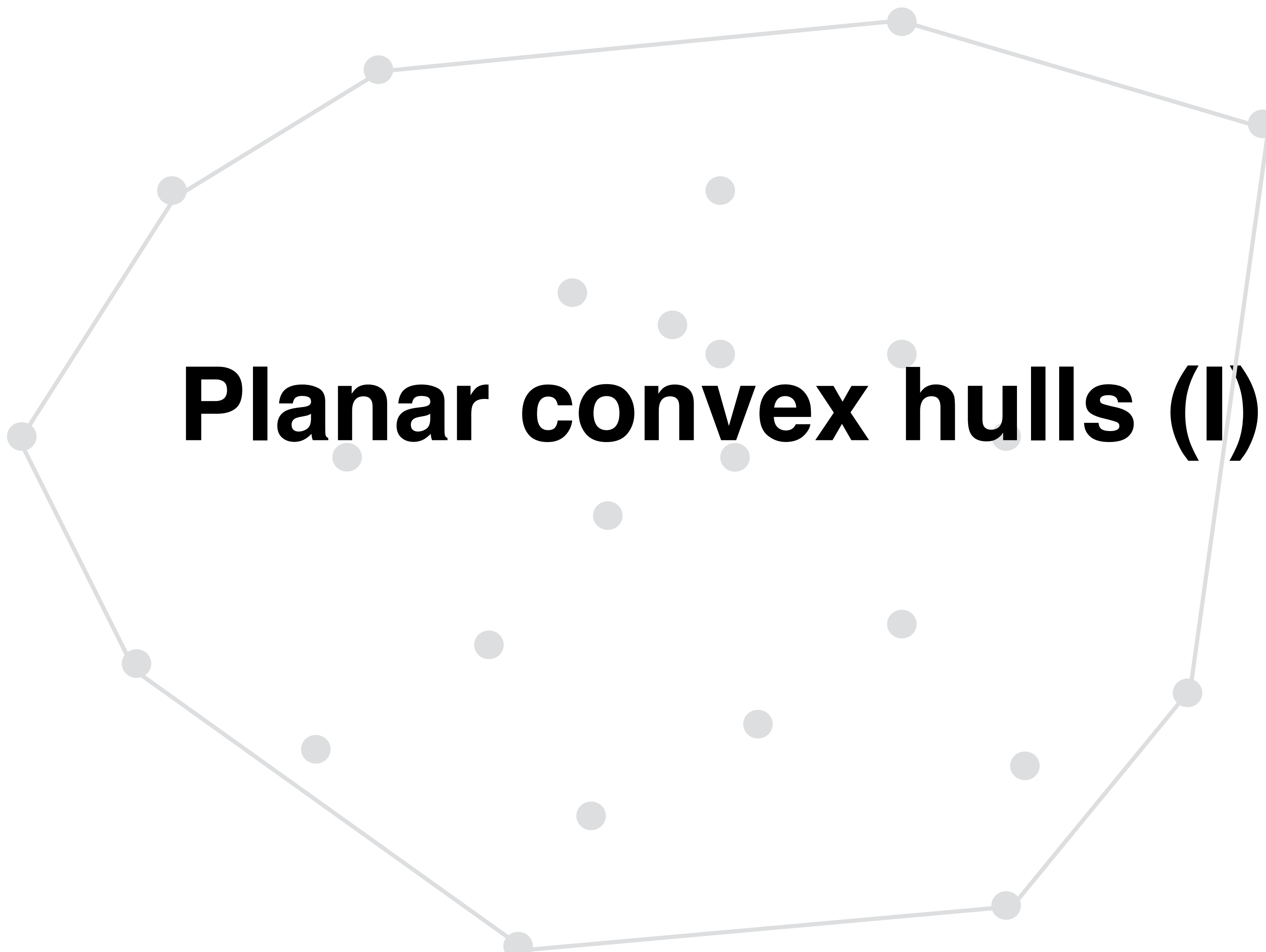
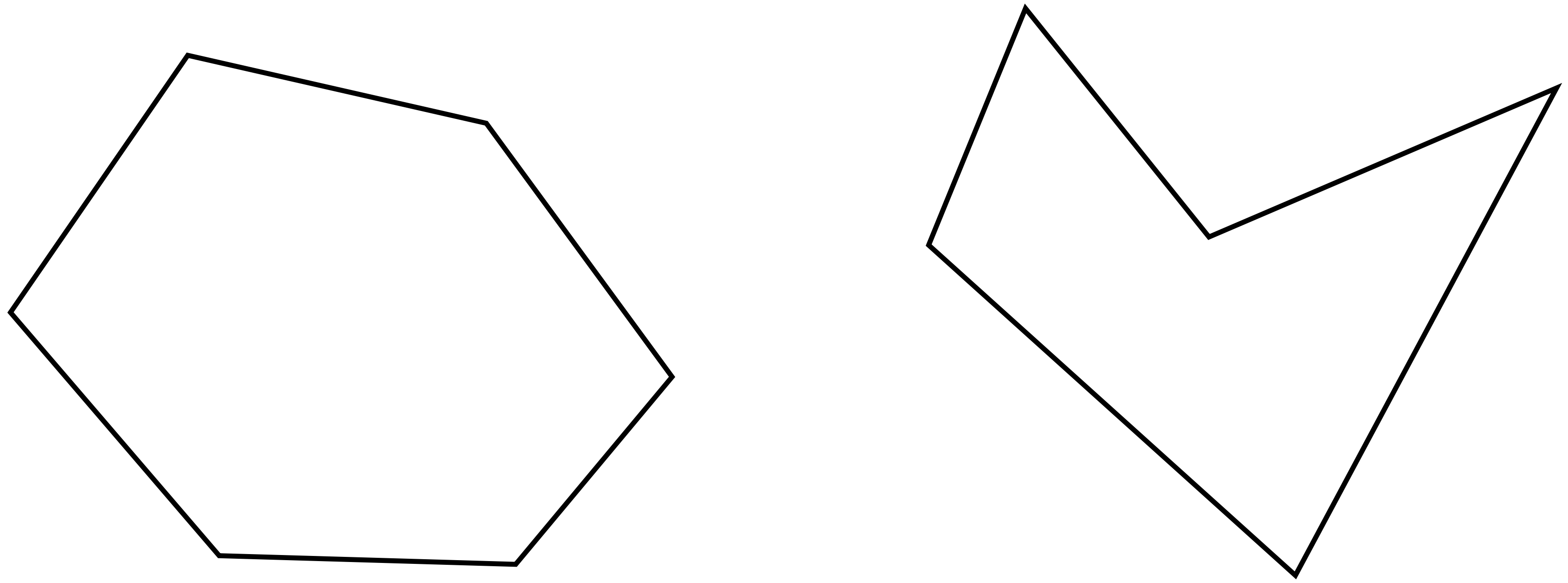
Planar convex hulls (I)



Computational Geometry [csci 3250]
Laura Toma
Bowdoin College

Convexity

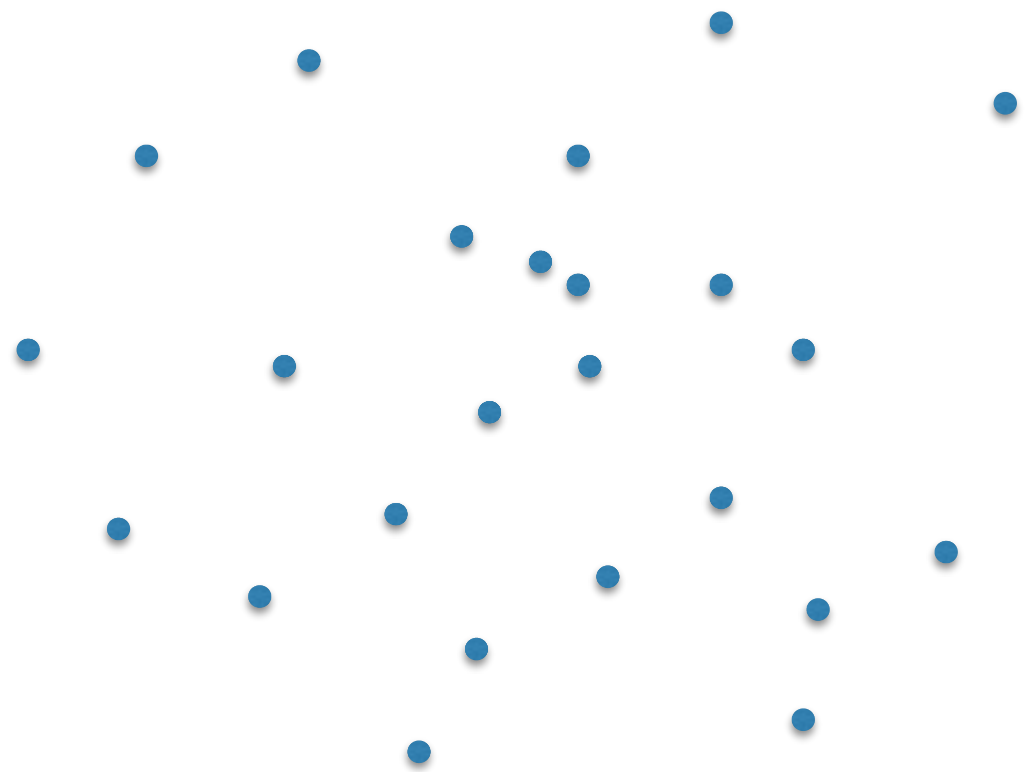
A polygon P is **convex** if for any p, q in P, the segment pq lies entirely in P.



convex non-convex

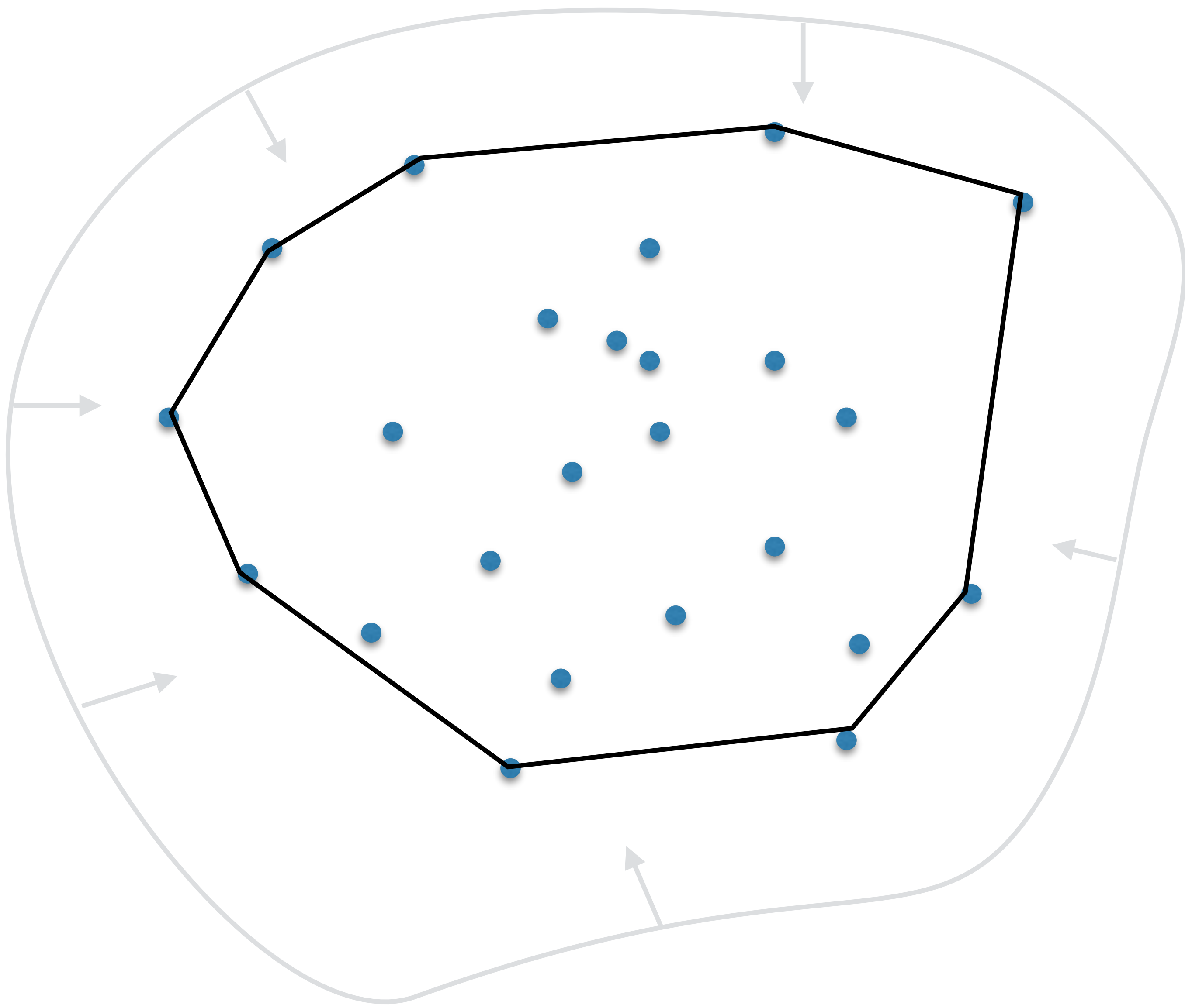
Convex Hull

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



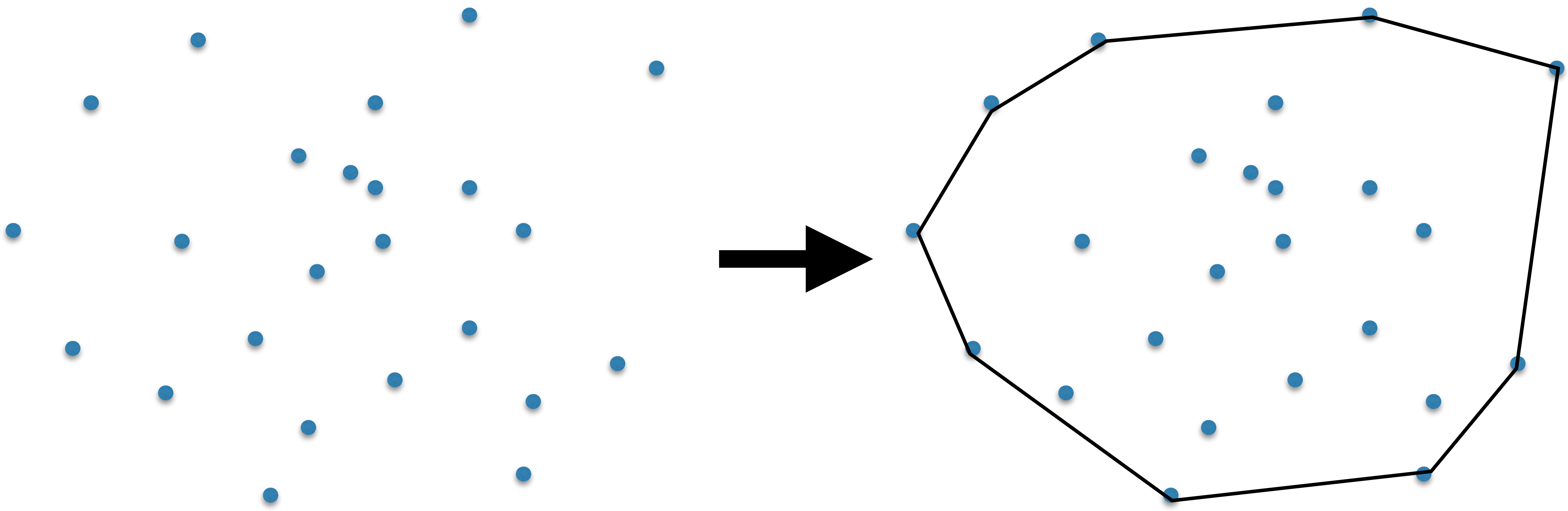
Convex Hull

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



Convex Hull

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



Input:
array P of points (in 2D)

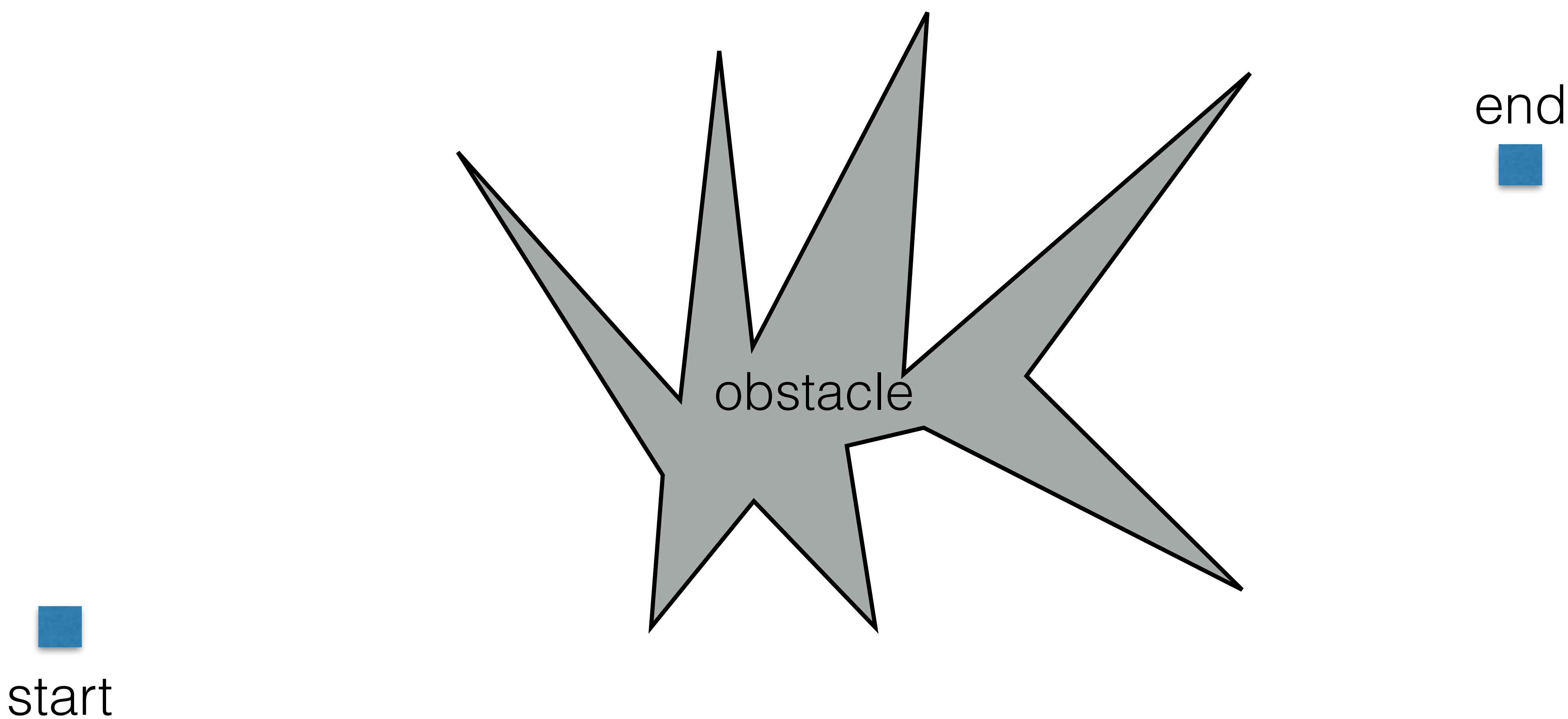
Output:
array/list of points on the CH (in boundary order)

Convex Hull

- One of the first problems studied in CG
- Many solutions
  - simple, elegant, intuitive, expose techniques
- Lots of applications
  - robotics
  - path planning
  - partitioning problems
  - shape recognition
  - separation problems

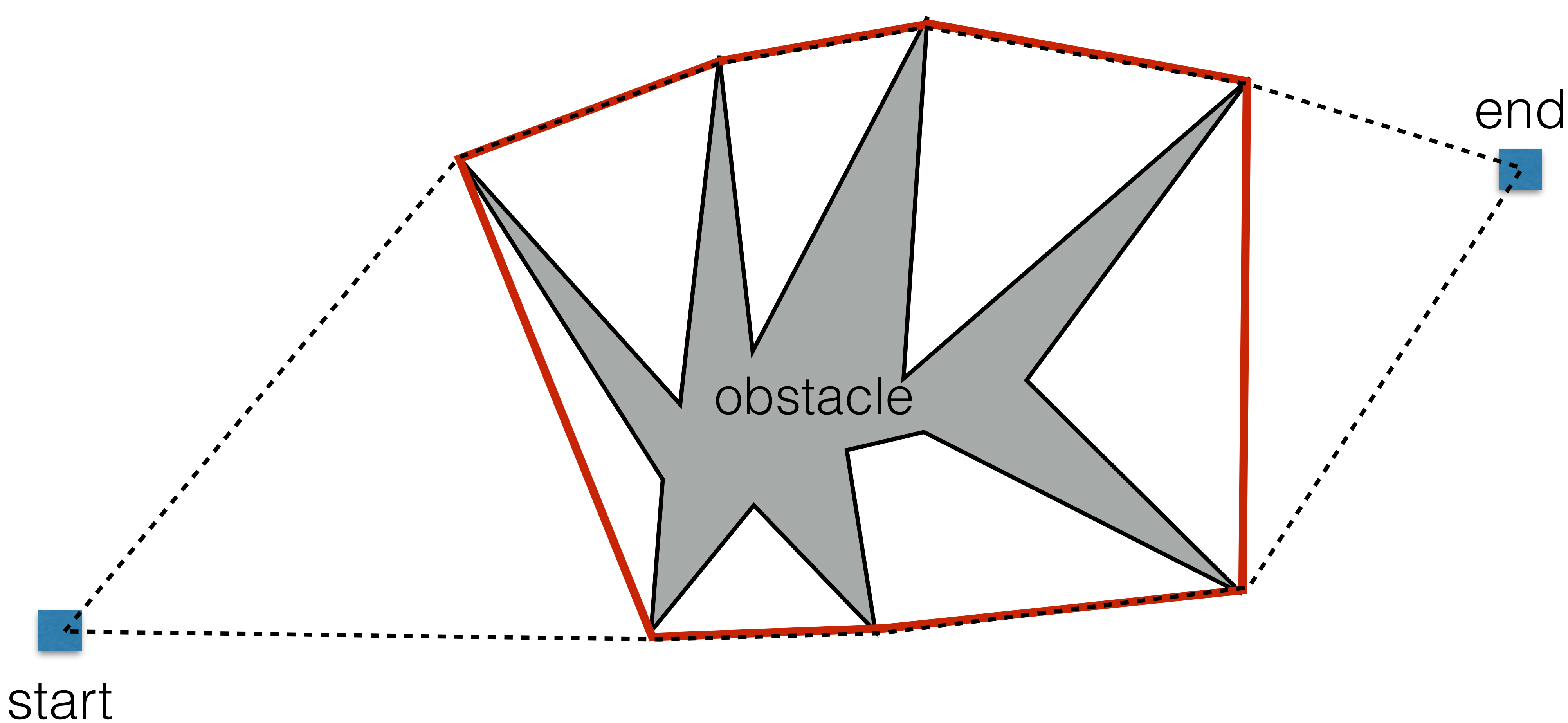
Applications

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



Applications

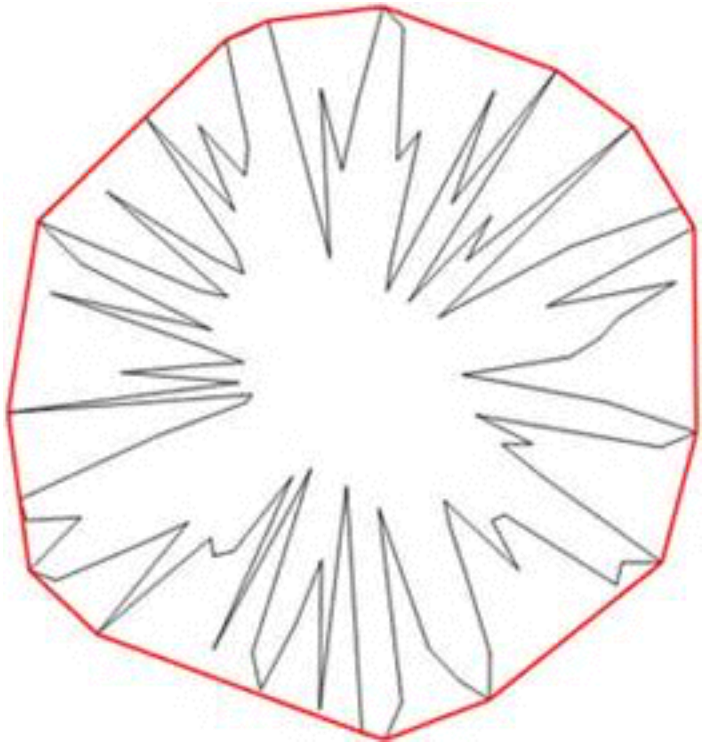
- Path planning: find (shortest) collision-free path from start to end



- It can be shown that the path follows CH(obstacle) and shortest path s to t is the shorter of the upper path and lower path

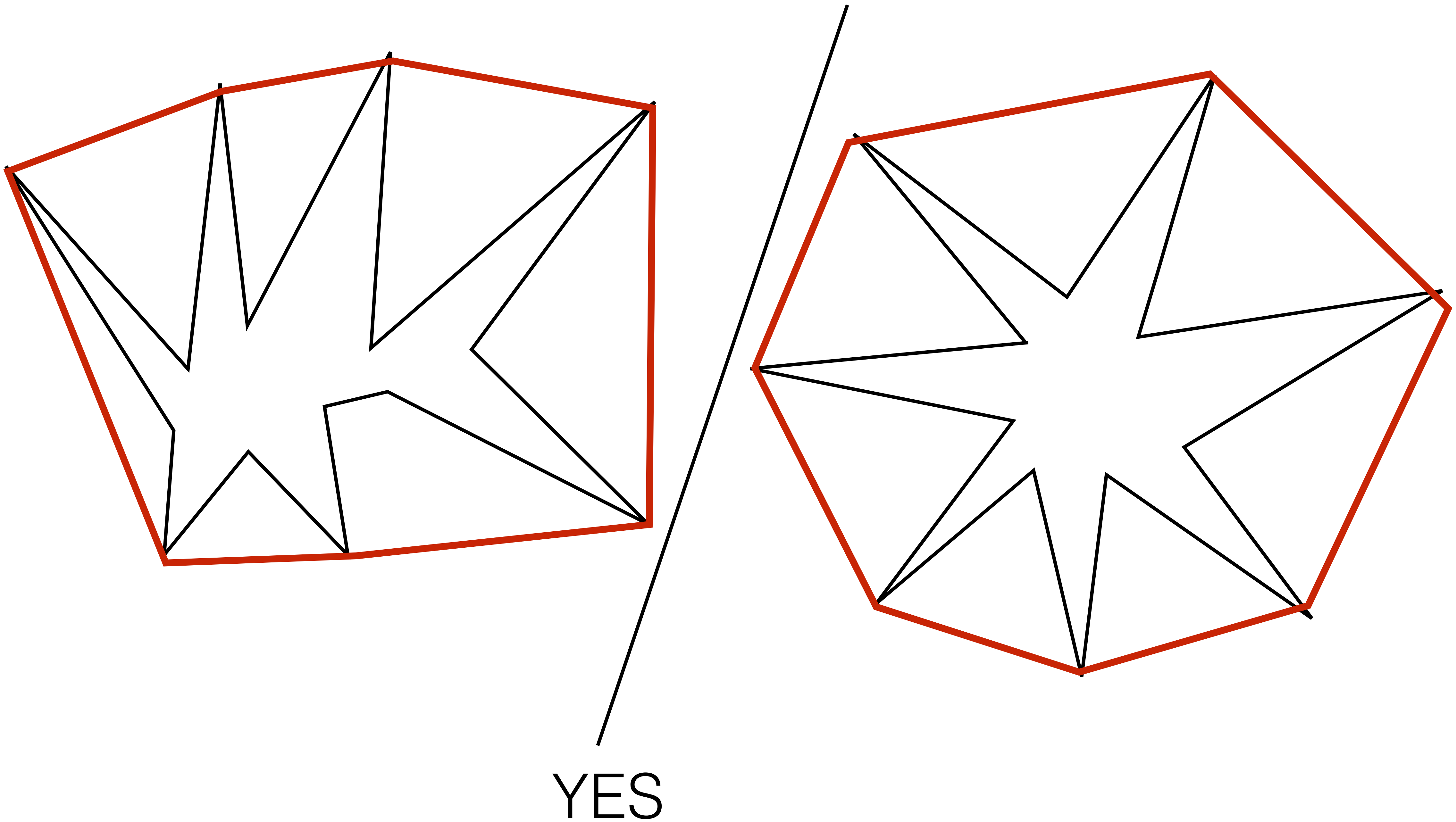
Applications

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



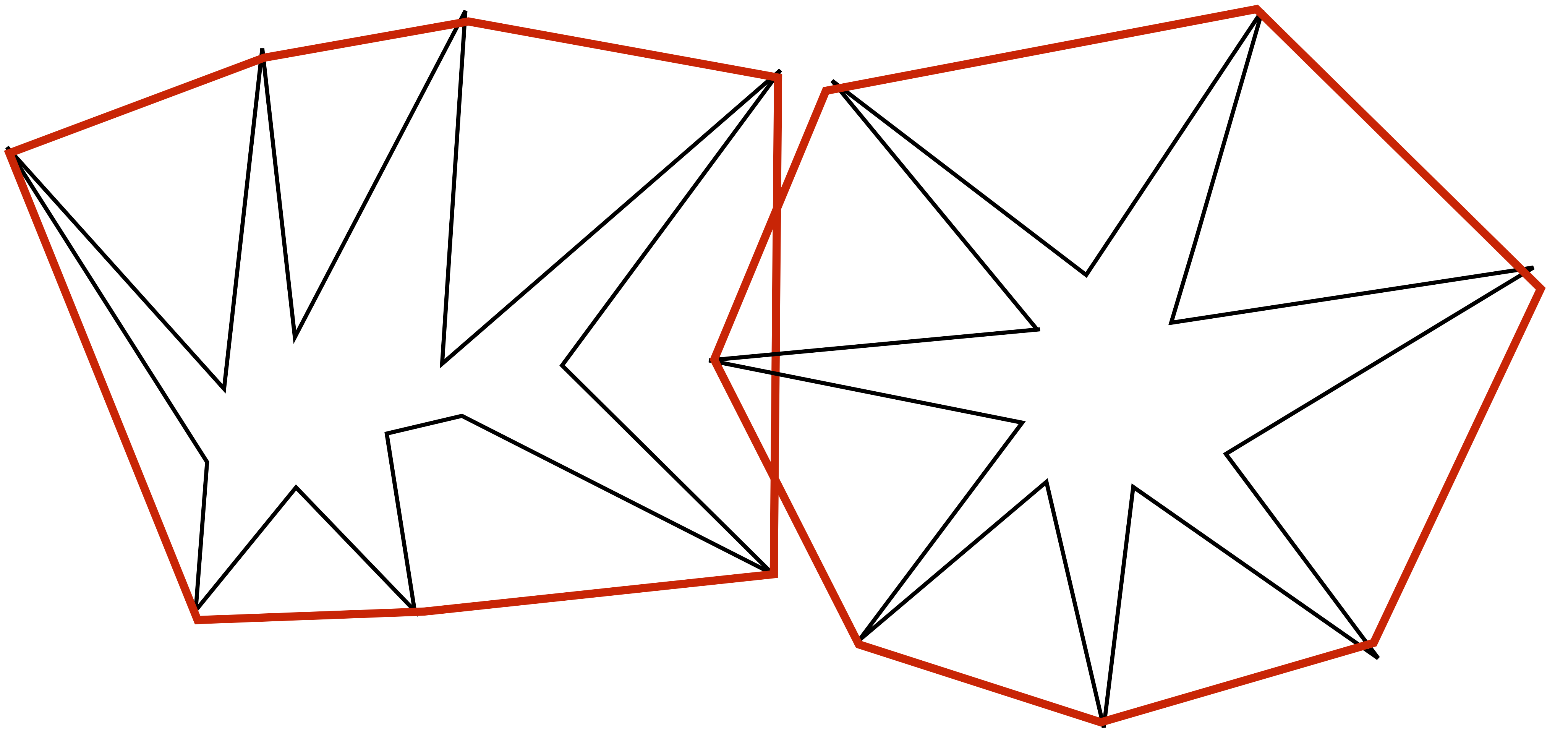
Applications

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



Applications

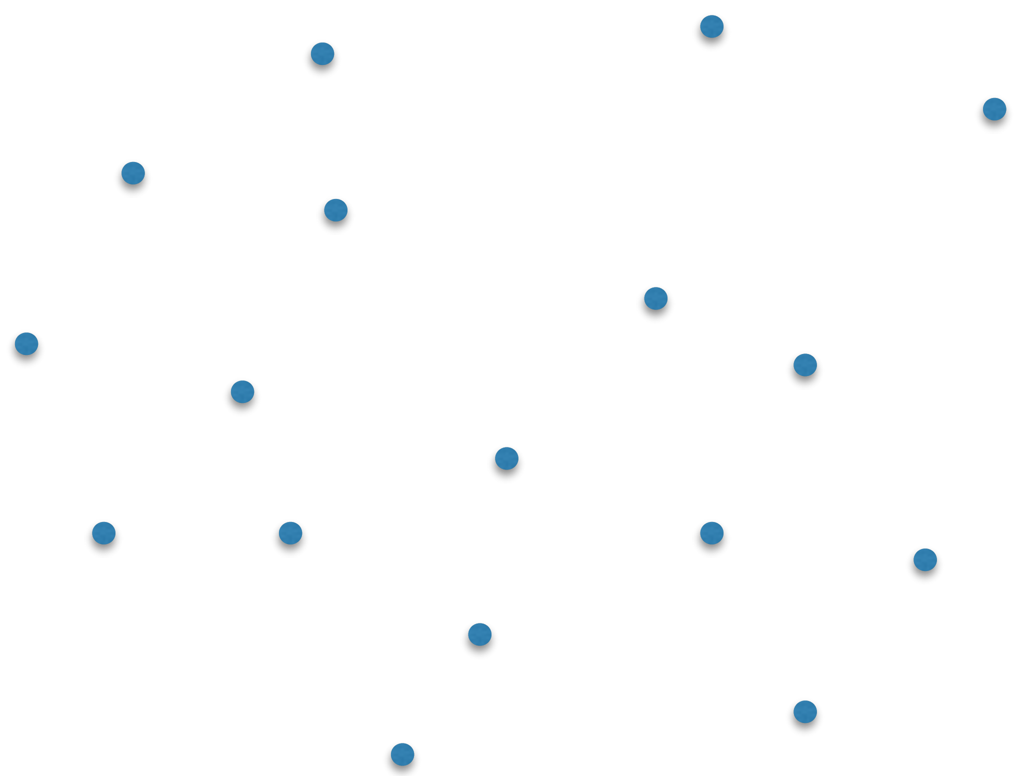
- Partitioning problems
  - does there exist a line separating two objects?



NO

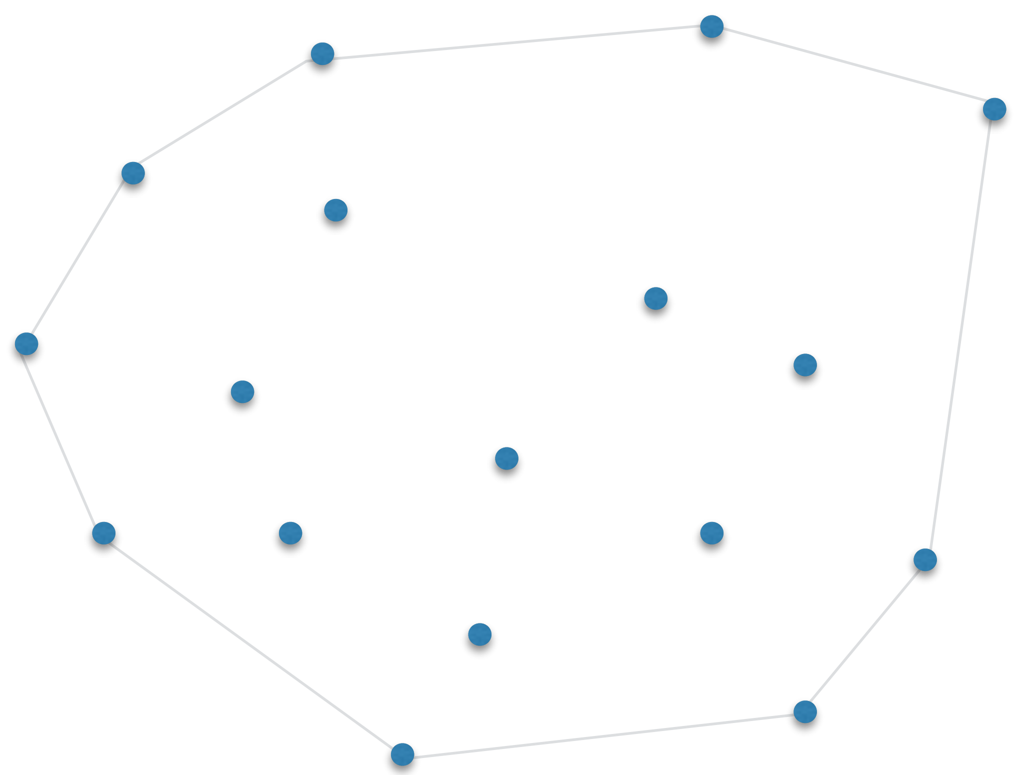
Applications

- Find the two points in P that are farthest away



Applications

- Find the two points in P that are farthest away

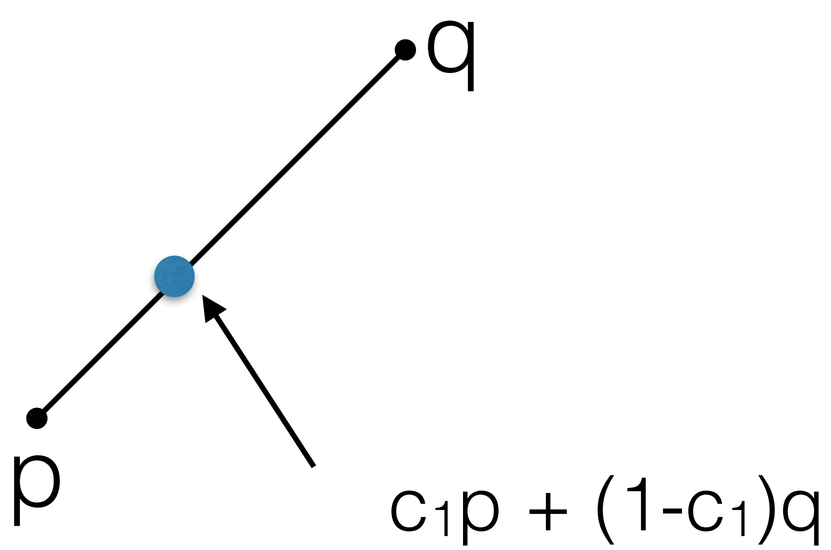


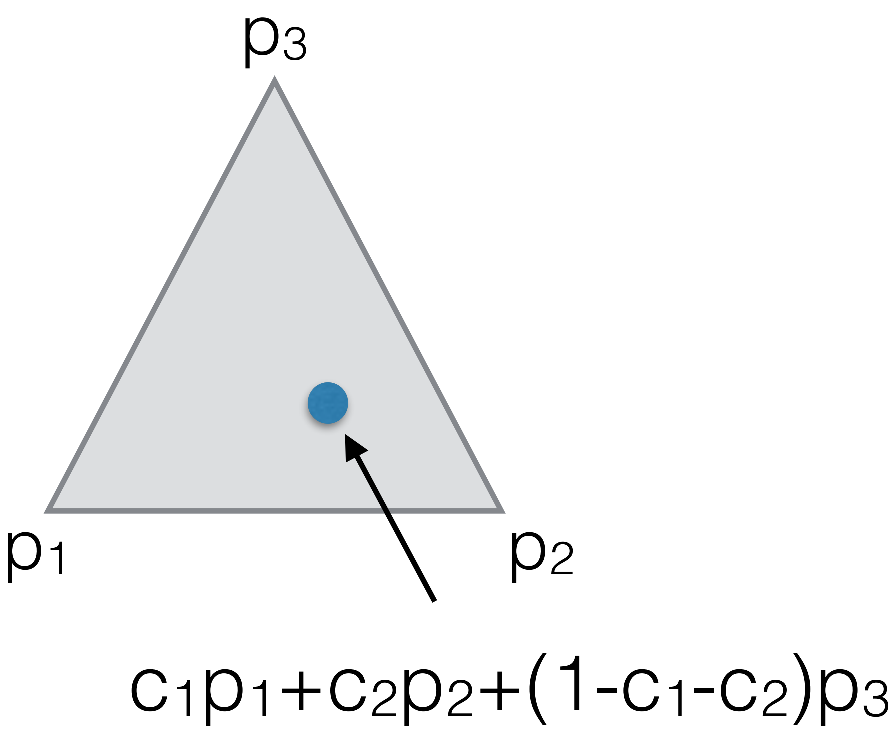
Outline

- Properties of CH
- Algorithms for computing the CH (P)
  - Brute-force
  - Gift wrapping (or: Jarviz march)
  - Quickhull
  - Graham scan
  - Andrew’s monotone chain
  - Incremental
  - Divide-and-conquer
- Can we do better?
  - Lower bound

Convexity: algebraic view

- Segment pq = set of all points of the form c_1p+c_2q, with c_1,c_2 in [0,1], $c_1+c_2=1$





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

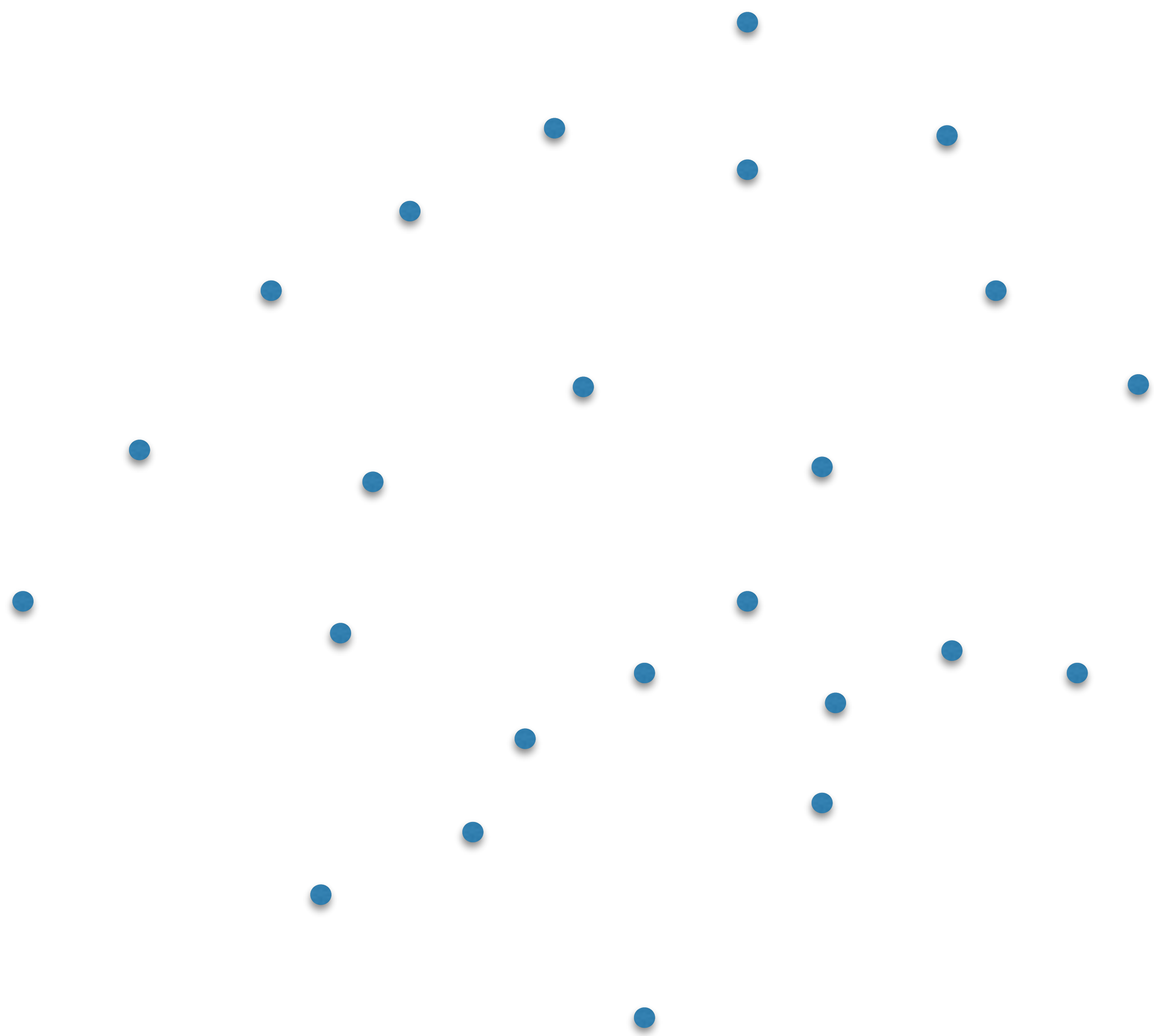
  $c_1p_1+c_2p_2+\dots c_kp_k$, with c_i in [0,1], $c_1+c_2+\dots+c_k=1$

- Example: a triangle consists of all convex combinations of its 3 vertices
- With this notation, the convex hull CH(P) = all convex combinations of points in P

Convex Hull Properties

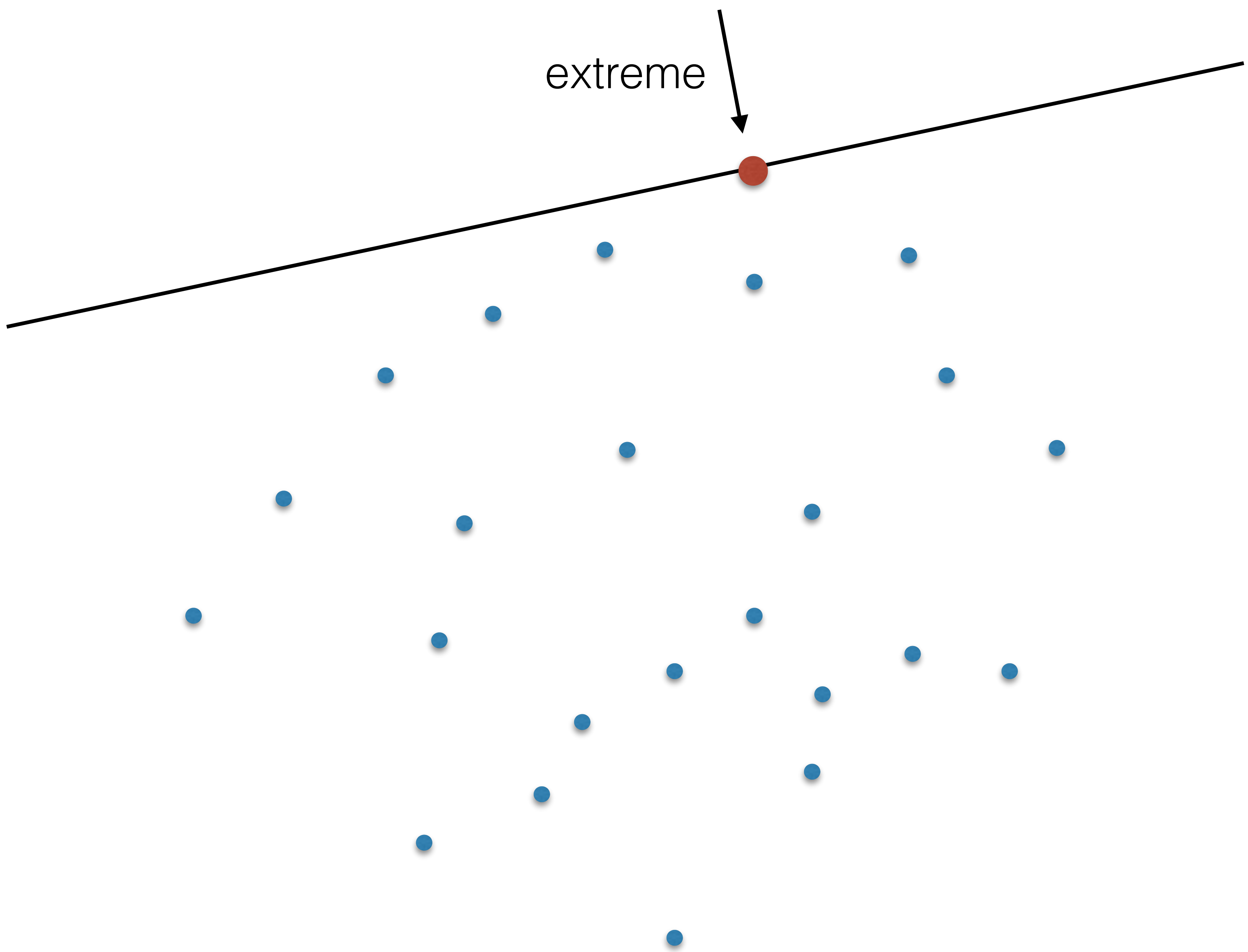
Extreme points

- A point p is extreme if there exists a line l through p, such that all the other points of P are on the same side of l (or on l)



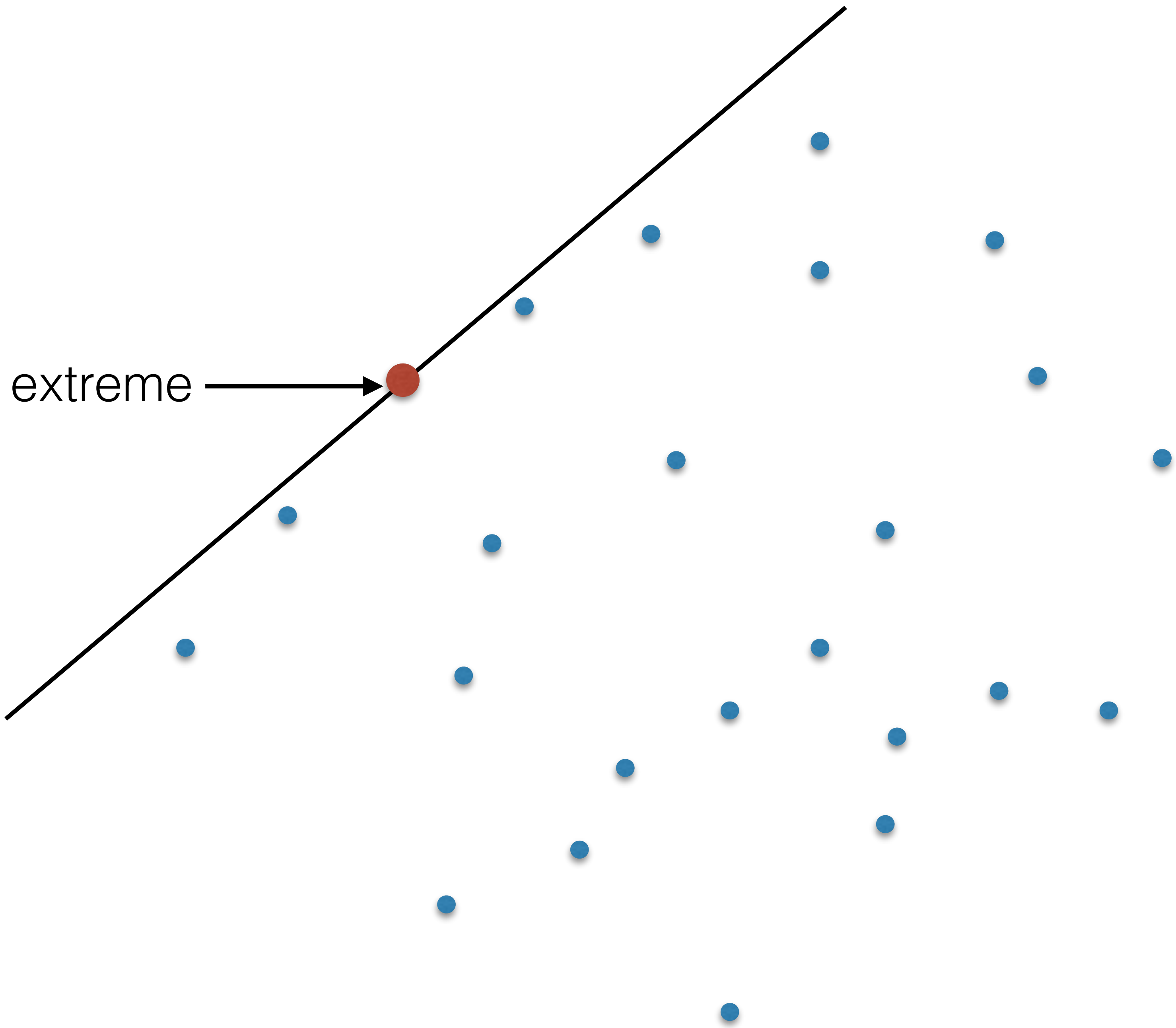
Extreme points

- A point p is 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)



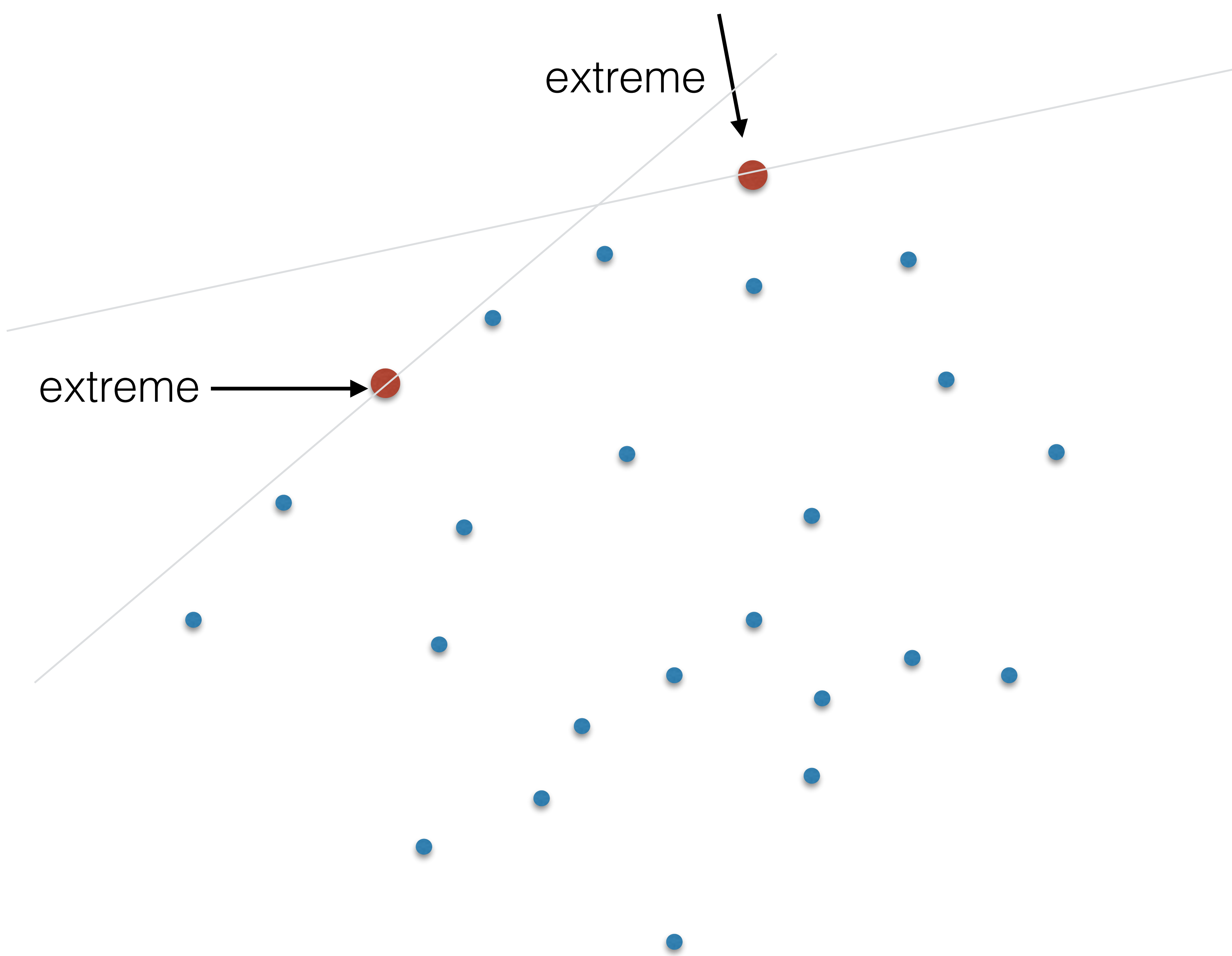
Extreme points

- A point p is 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)



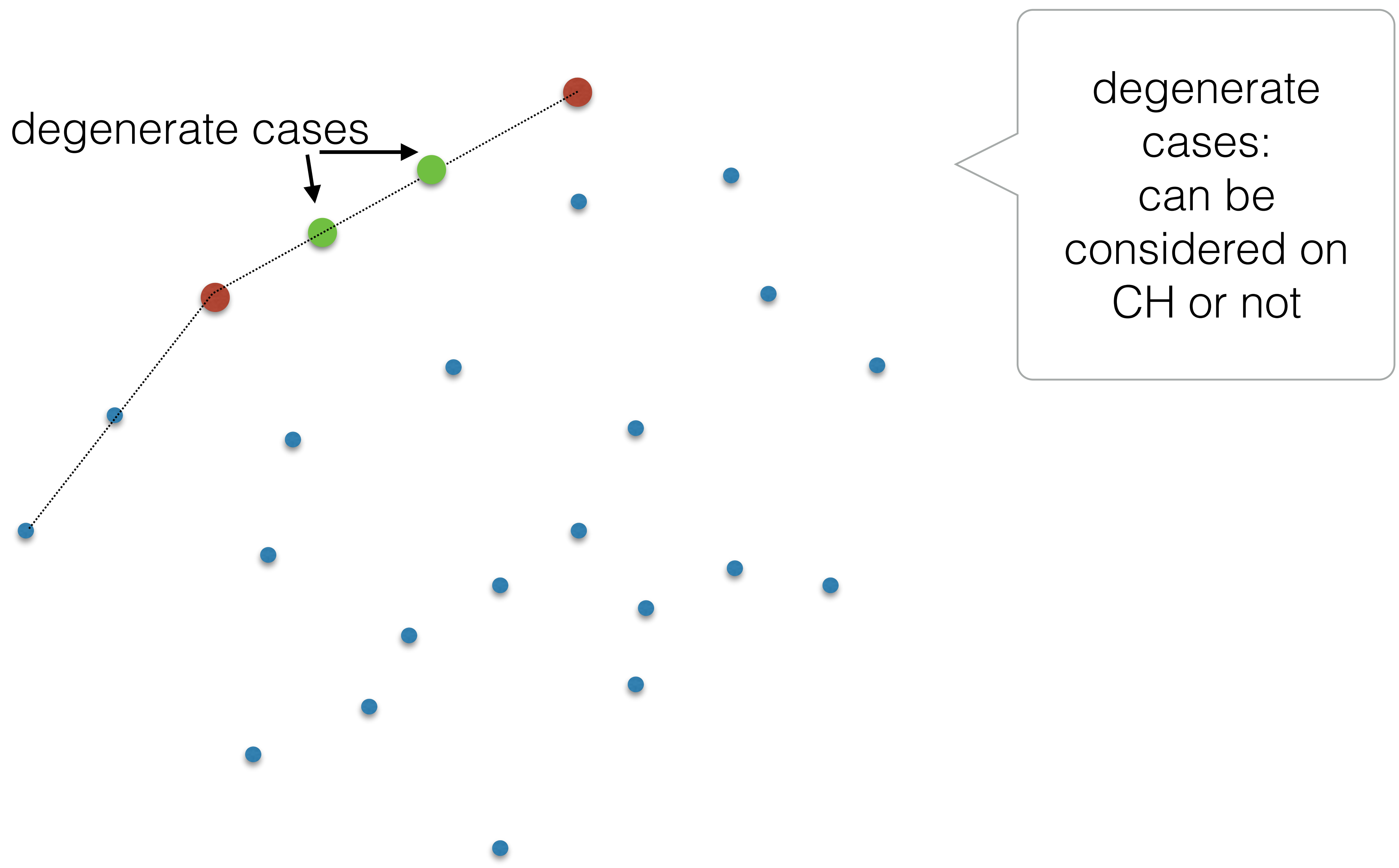
Extreme points

- A point p is 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)



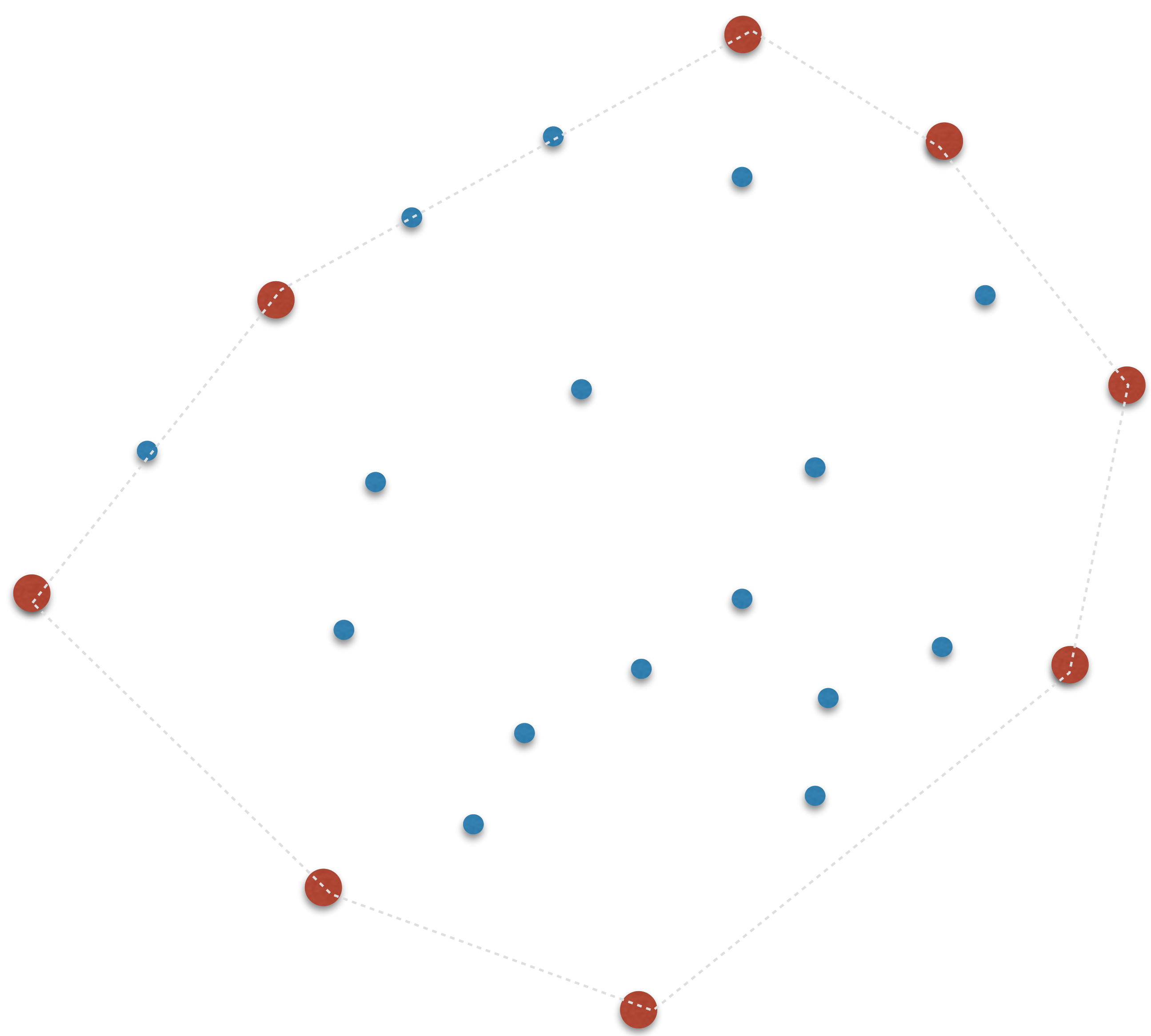
Extreme points

- A point p is 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)



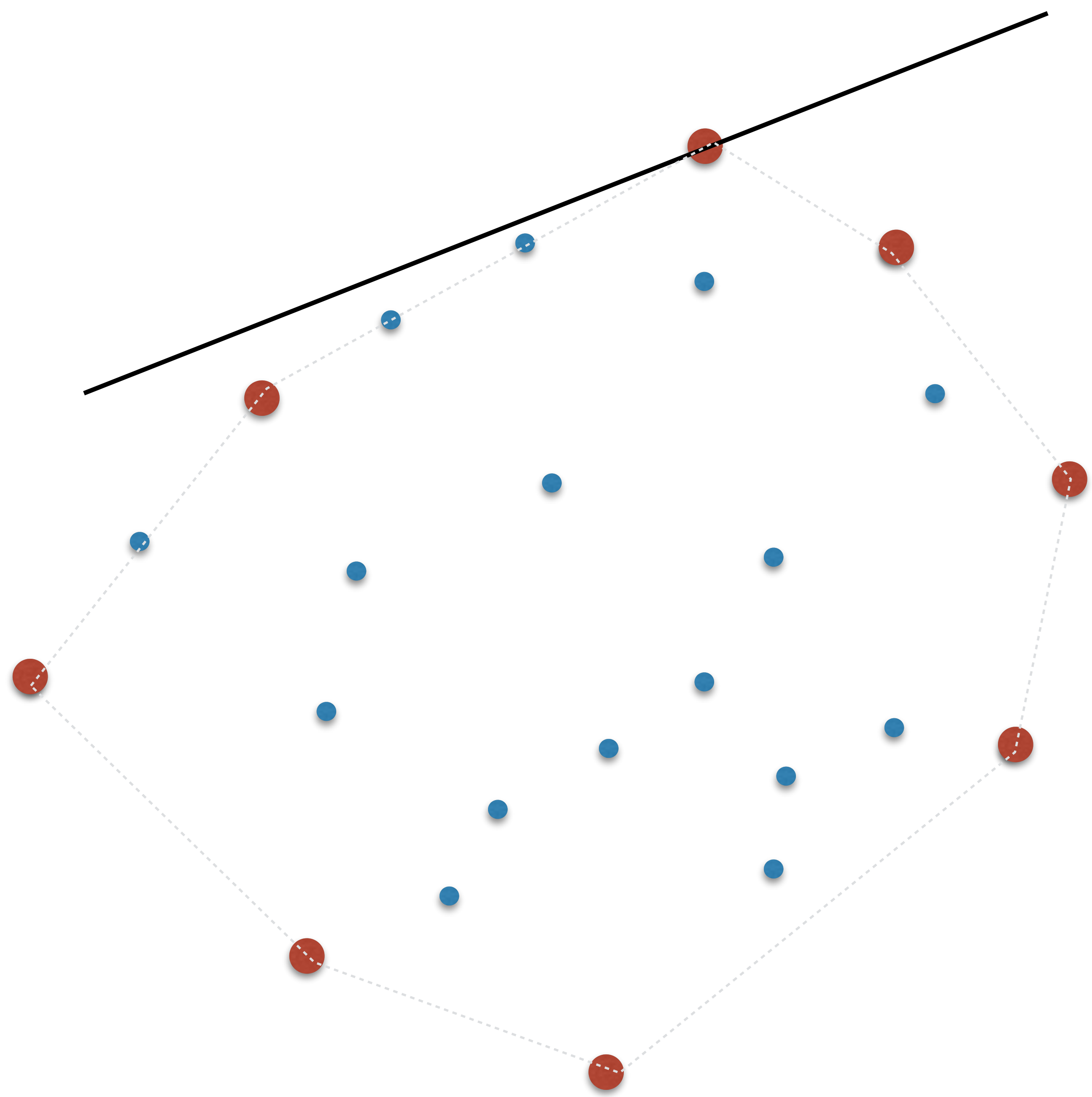
Extreme points

- Claim: If a point is on the CH if and only if (iff) it is extreme.



Extreme points

- Claim: If a point is on the CH if and only if (iff) it is extreme.



CH Variants

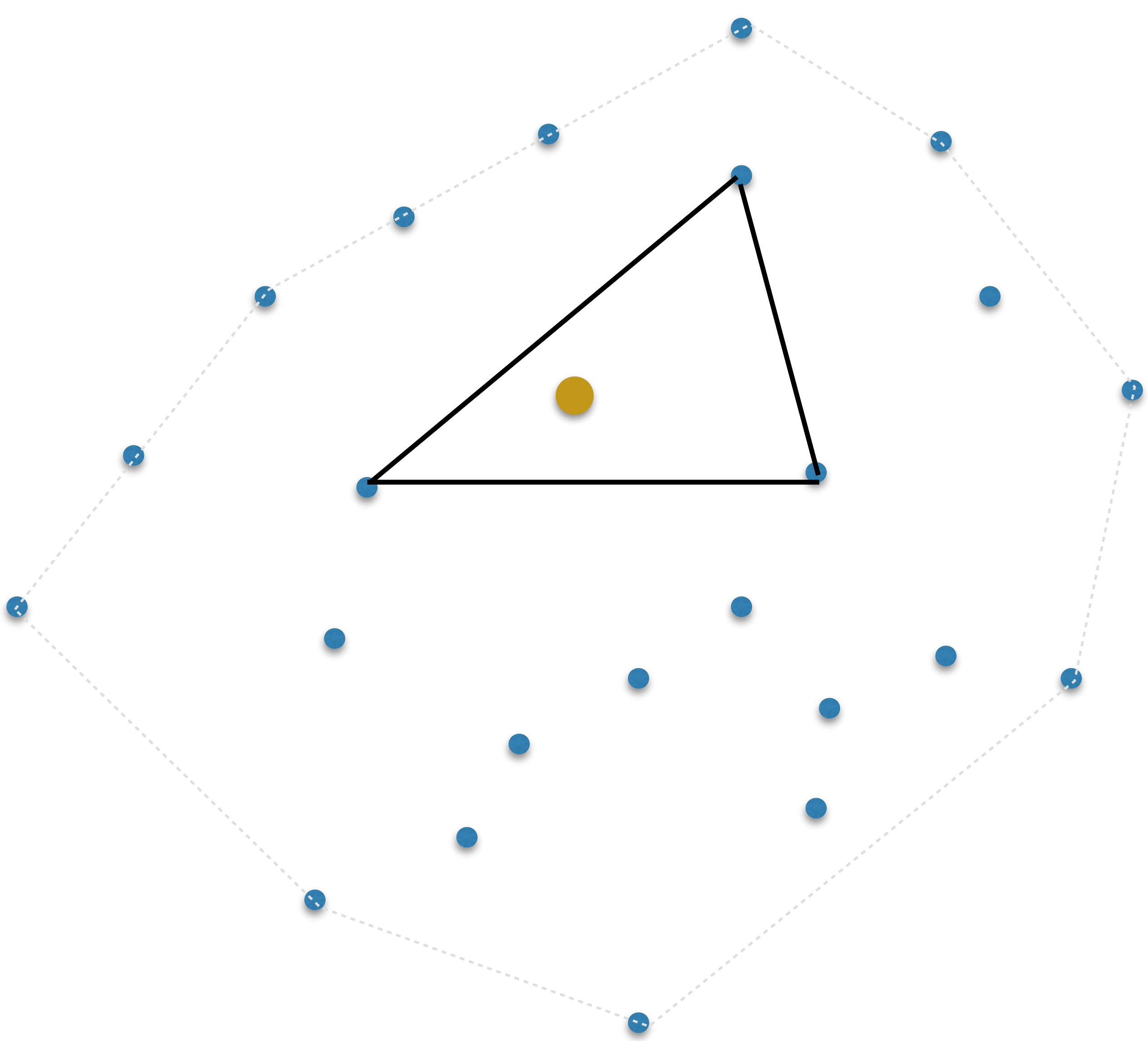
- Several types of convex hull output are conceivable
  - all points on the convex hull in arbitrary order
  - all points on the convex hull in boundary order
  - only non-collinear points in arbitrary order
  - only non-collinear points in boundary order

<——— exclude collinear points

- It may seem that computing in boundary order is harder
  - we'll see that identifying the extreme points is Omega(n lg n)
  - so sorting is not dominant

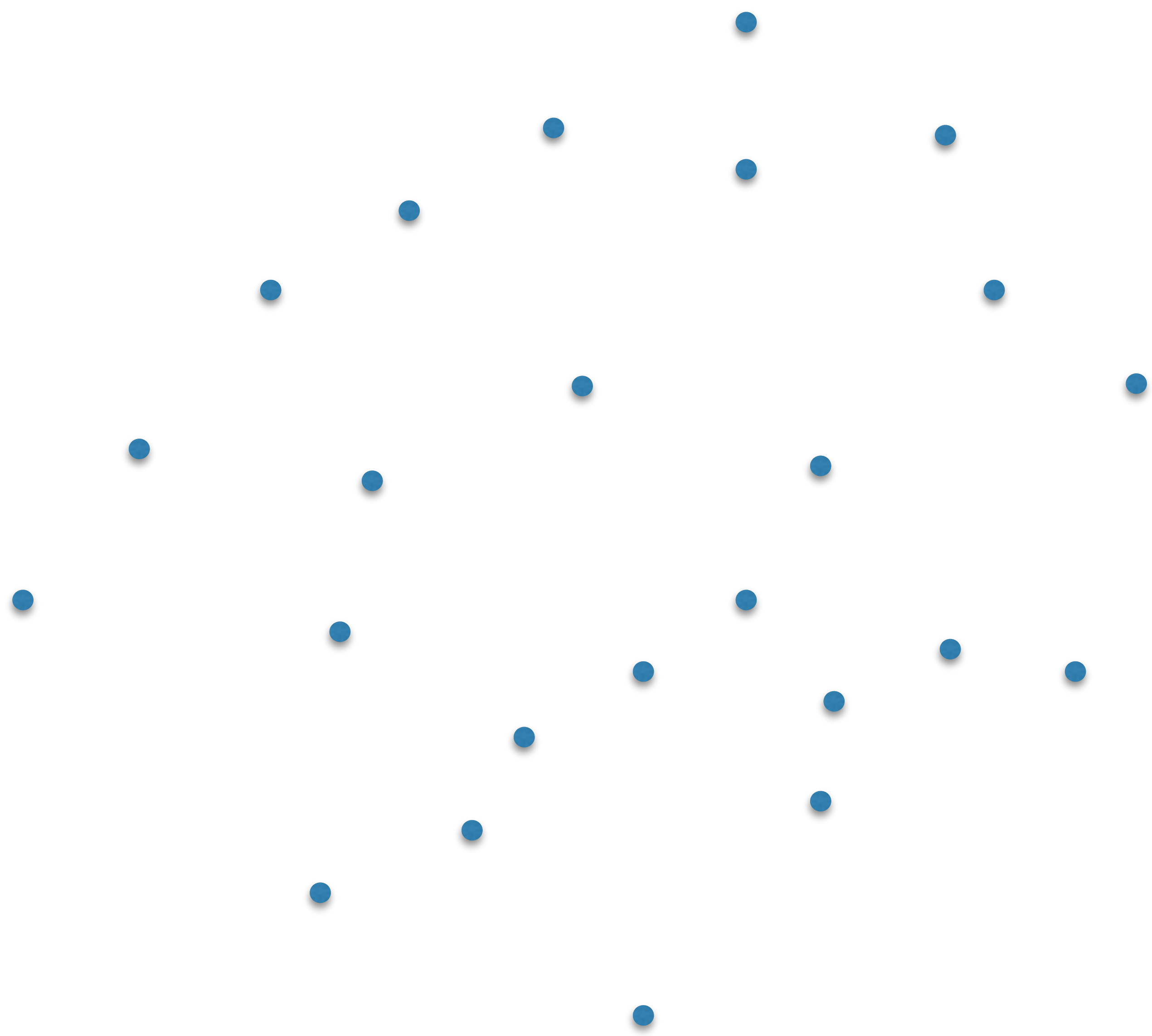
Interior points

- A point p is **not** on the CH if and only if p is contained in the interior of a triangle formed by three other points of P (or in interior of a segment formed by two points).



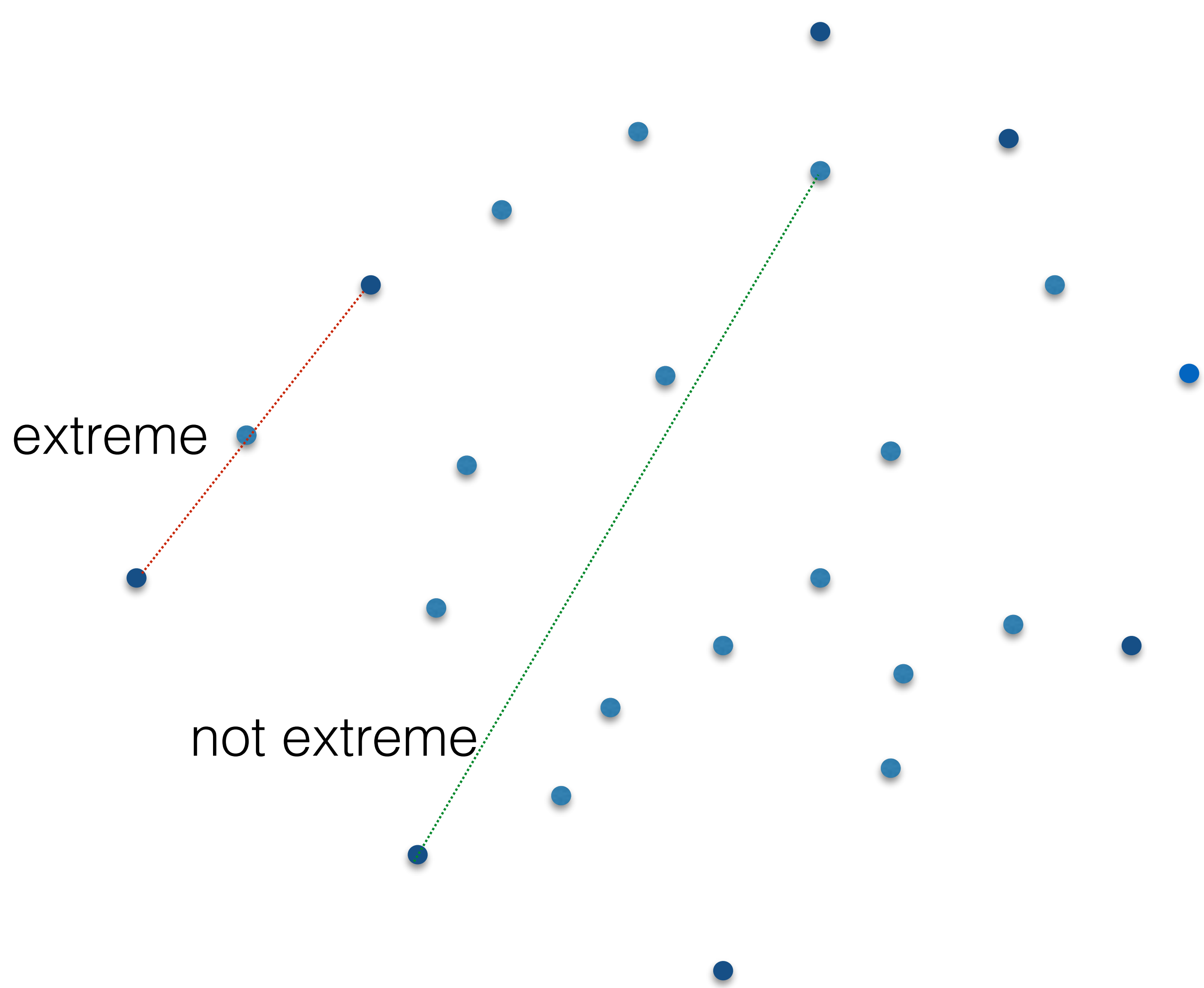
Extreme edges

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



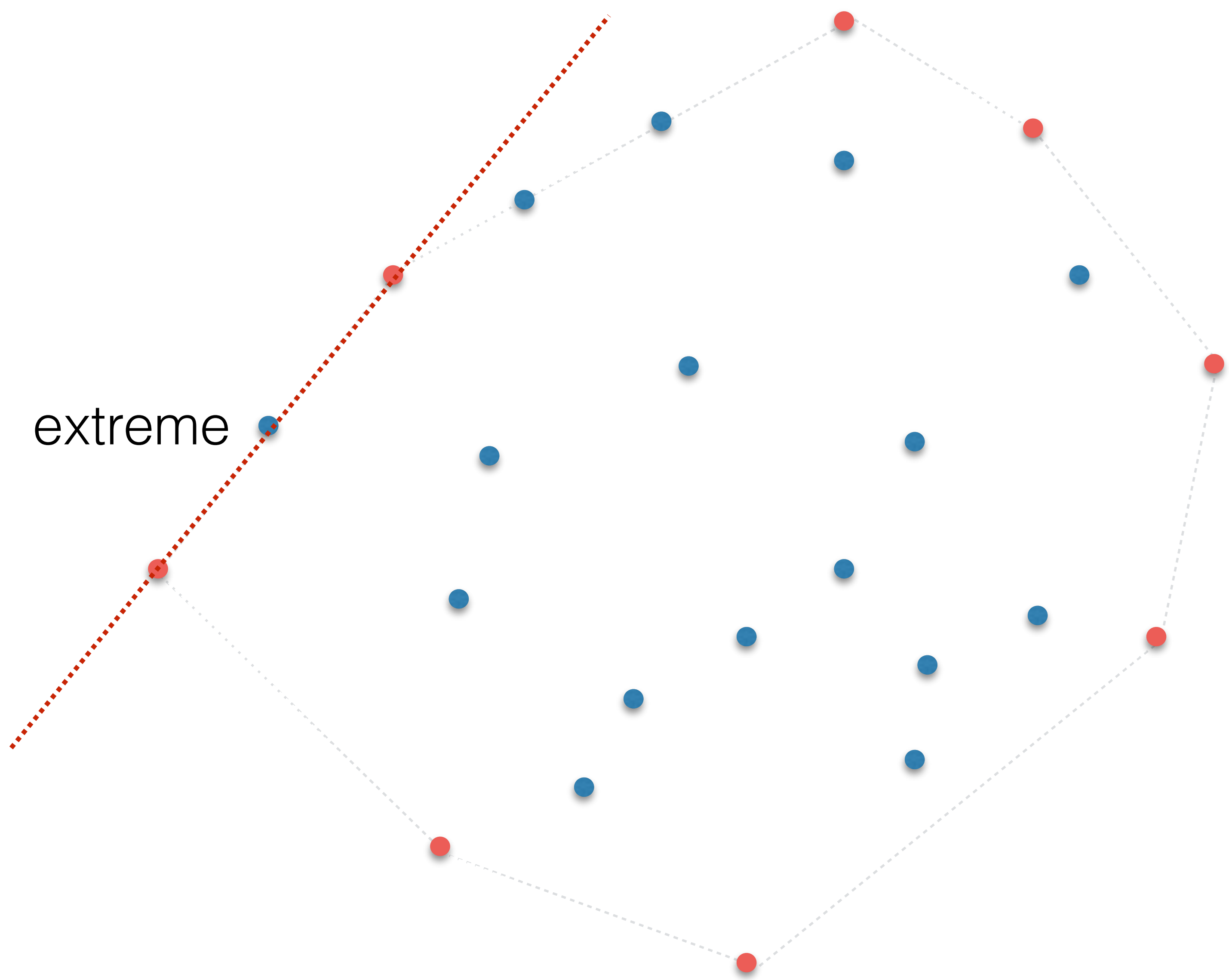
Extreme edges

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



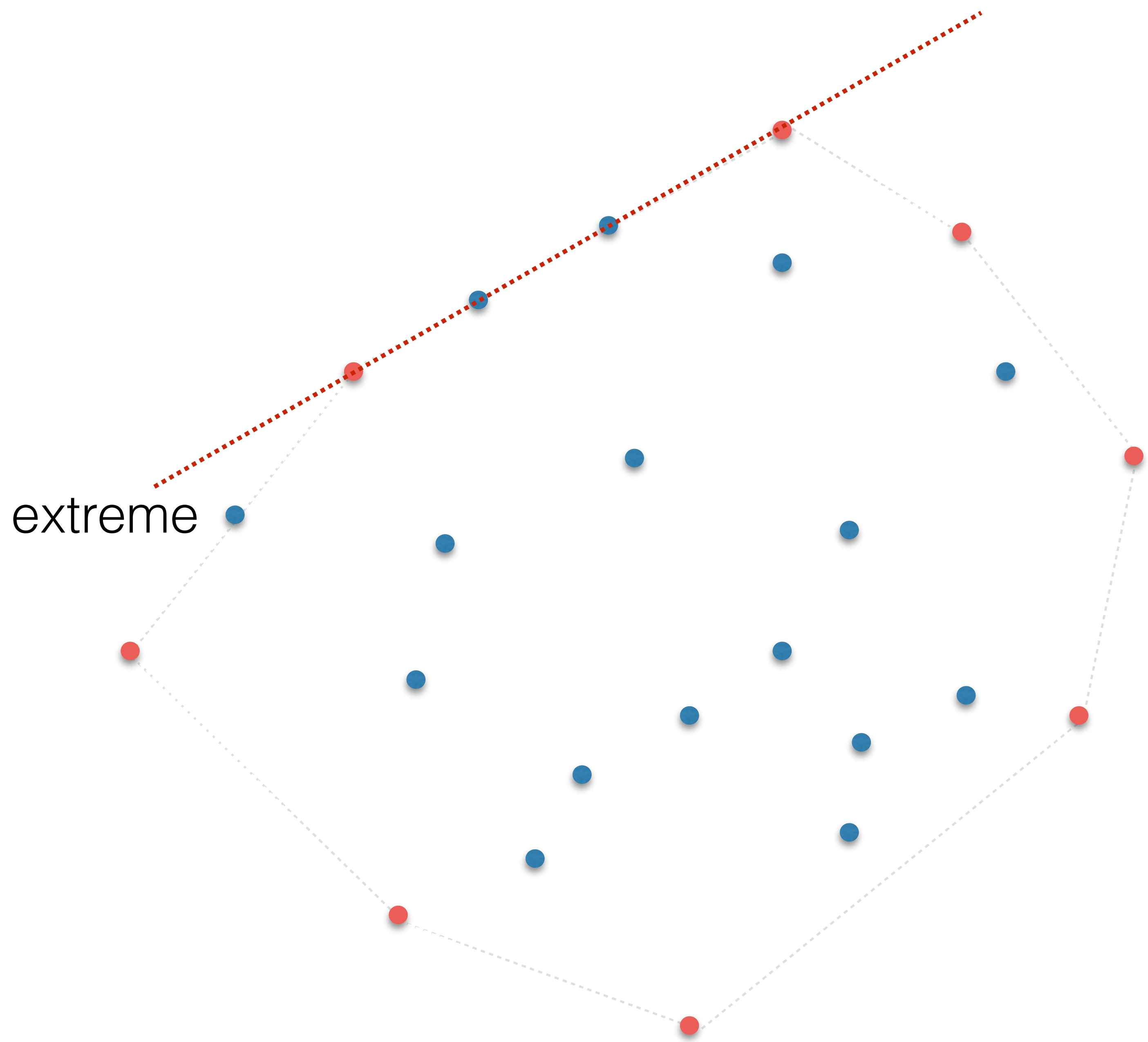
Extreme edges

- An edge (p_i, p_j) is extreme if all the other points of P are on one side of it (or on)
- Claim: A pair of points (p_i, p_j) form an edge on the CH iff edge (p_i, p_j) is extreme.



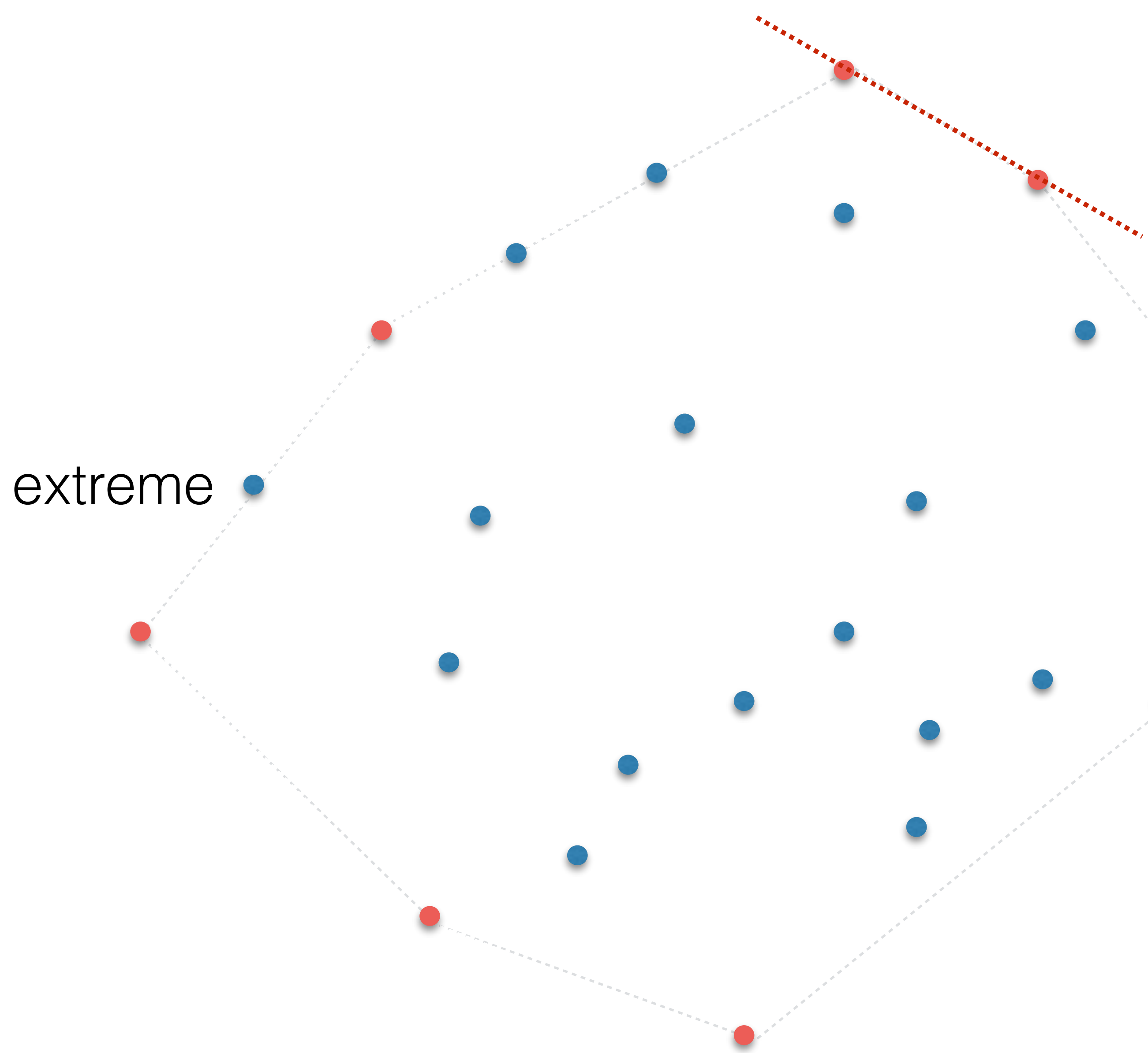
Extreme edges

- An edge (p_i, p_j) is extreme if all the other points of P are on one side of it (or on)
- Claim: A pair of points (p_i, p_j) form an edge on the CH iff edge (p_i, p_j) is extreme.



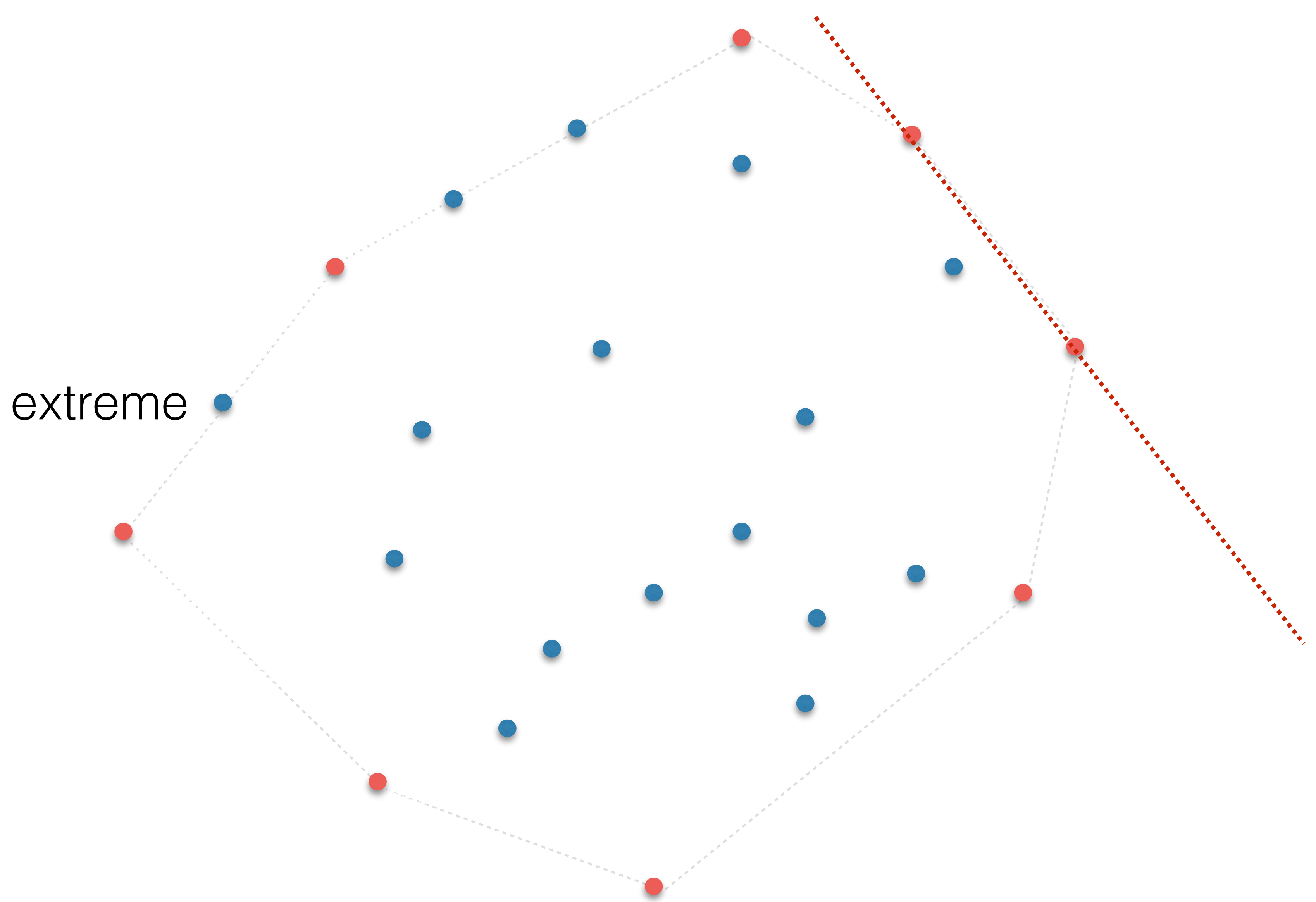
Extreme edges

- An edge (p_i, p_j) is extreme if all the other points of P are on one side of it (or on)
- Claim: A pair of points (p_i, p_j) form an edge on the CH iff edge (p_i, p_j) is extreme.



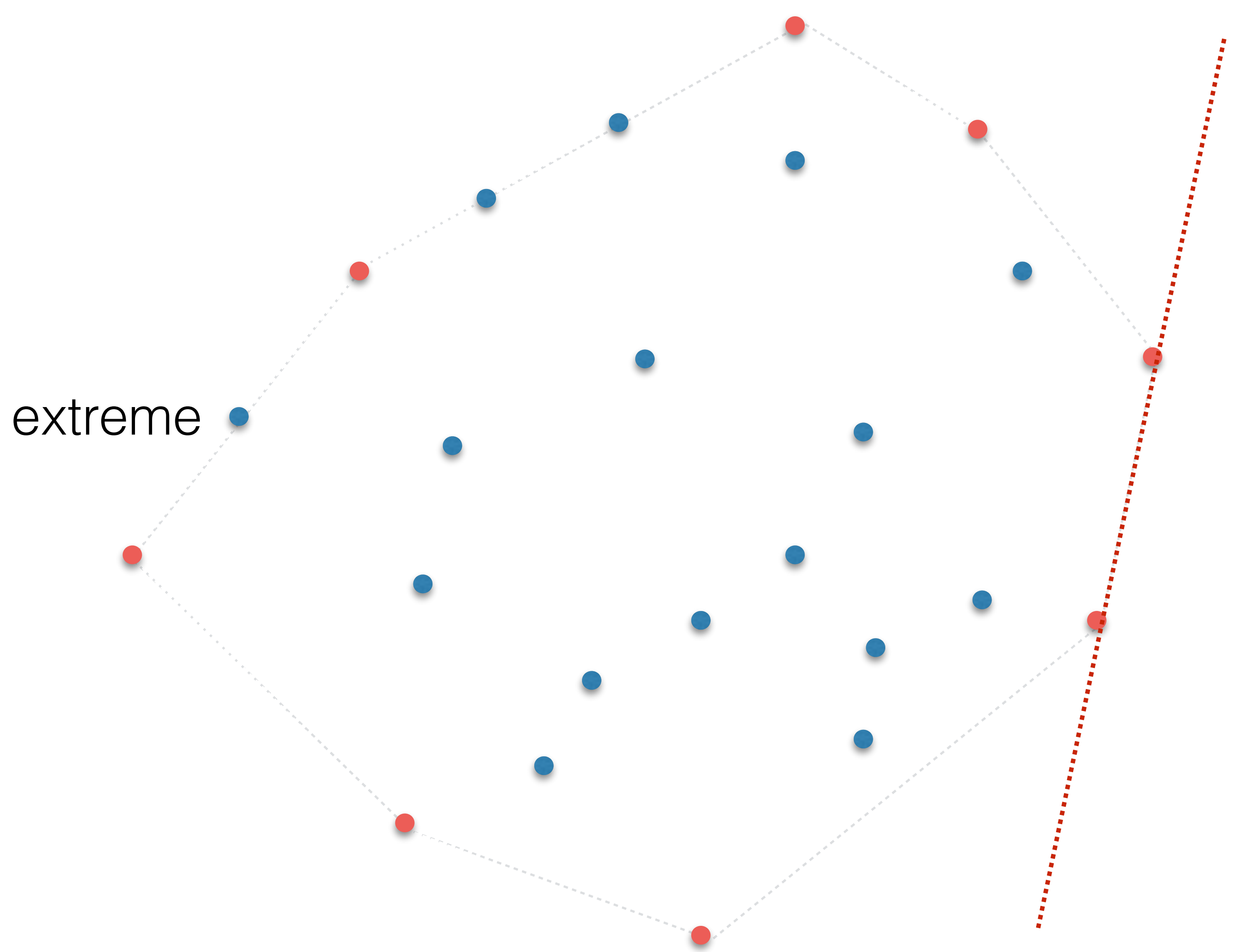
Extreme edges

- An edge (p_i, p_j) is extreme if all the other points of P are on one side of it (or on)
- Claim: A pair of points (p_i, p_j) form an edge on the CH iff edge (p_i, p_j) is extreme.



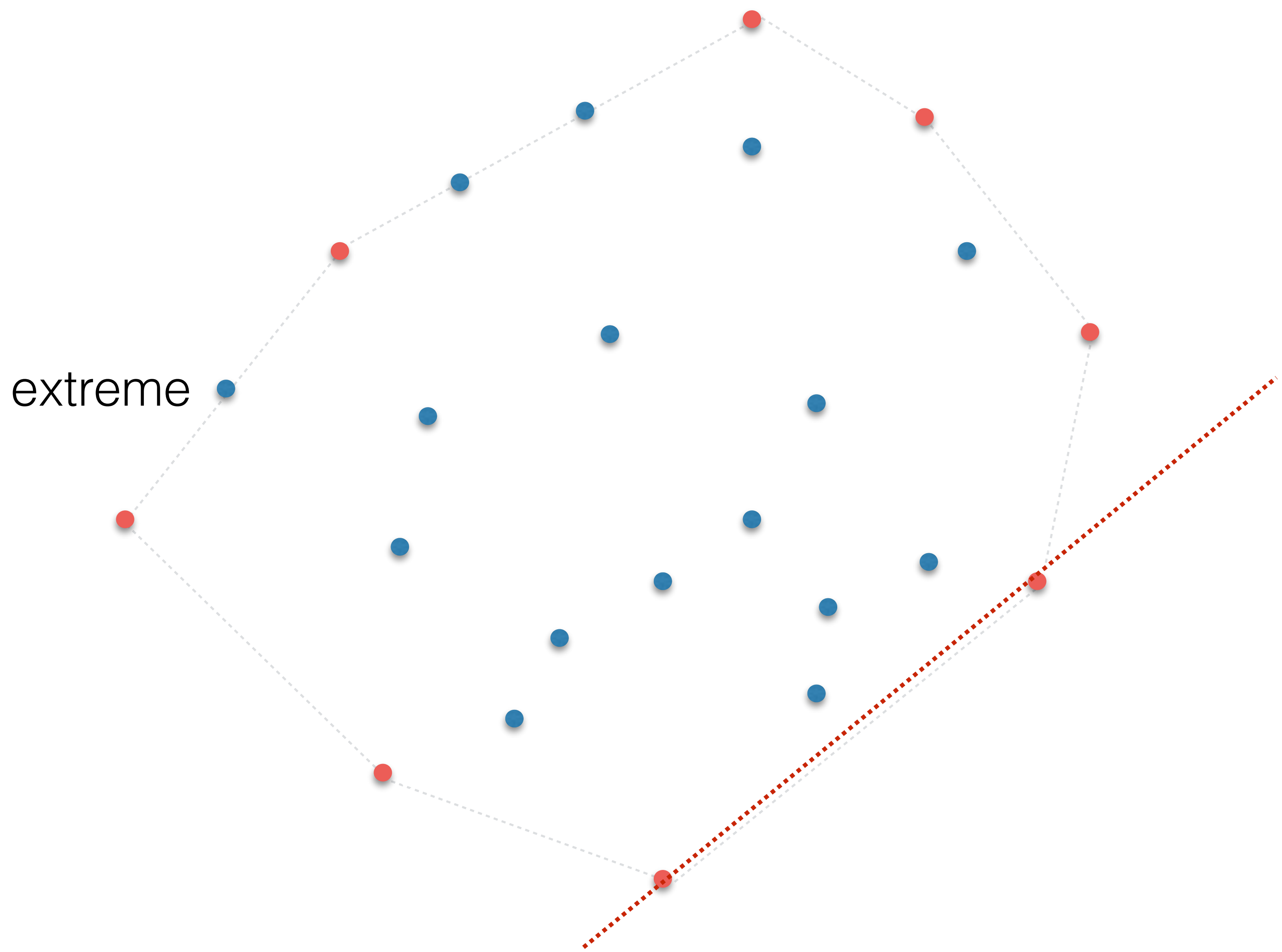
Extreme edges

- An edge (p_i, p_j) is extreme if all the other points of P are on one side of it (or on)
- Claim: A pair of points (p_i, p_j) form an edge on the CH iff edge (p_i, p_j) 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)
- Claim: A pair of points (p_i, p_j) form an edge on the CH iff edge (p_i, p_j) is extreme.

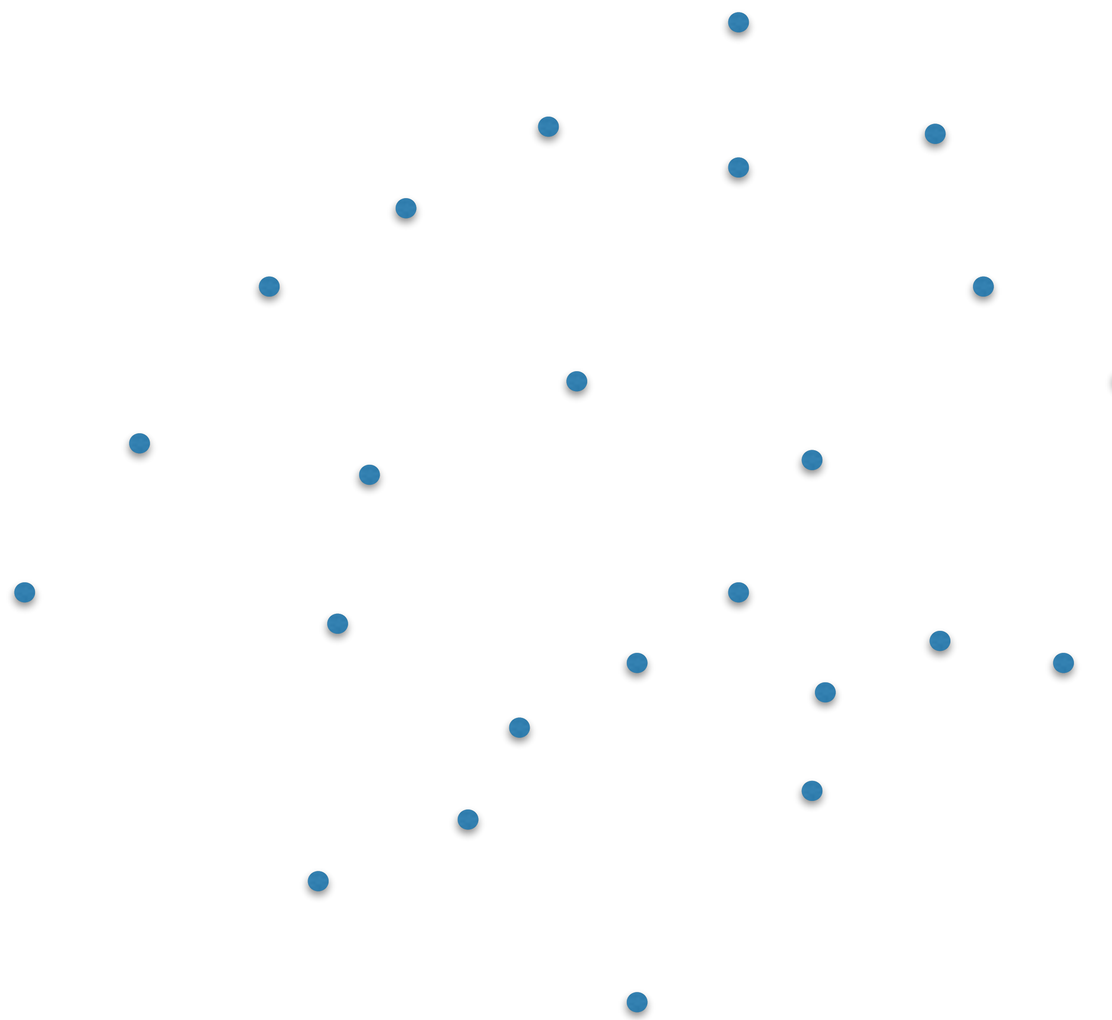


CH by finding extreme edges

Brute force

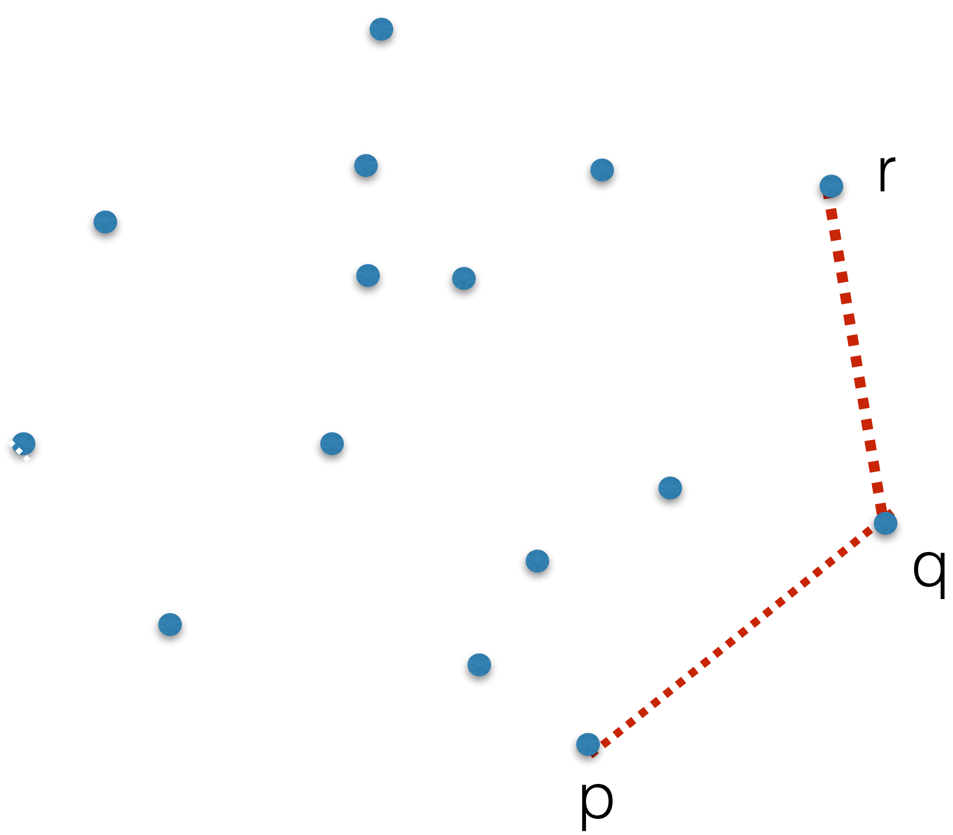
Algorithm (input P)

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



- Analysis?

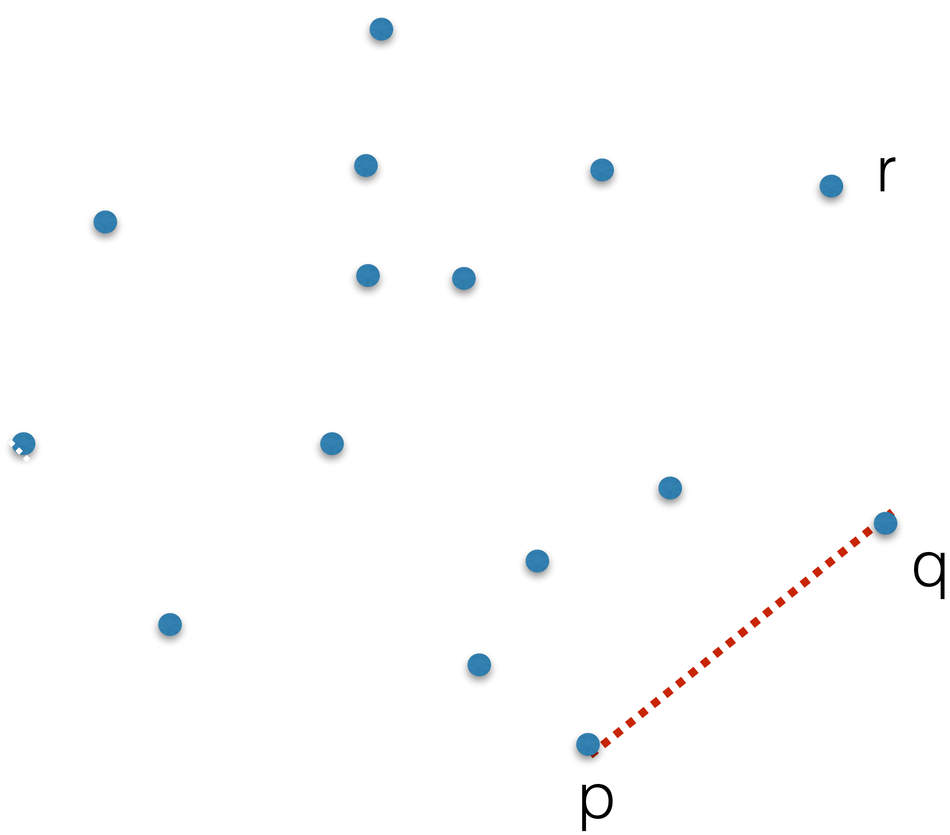
Gift wrapping (1970)



- Observations
  - CH consists of extreme edges
  - 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?
- Given an extreme edge, how to find the next one?

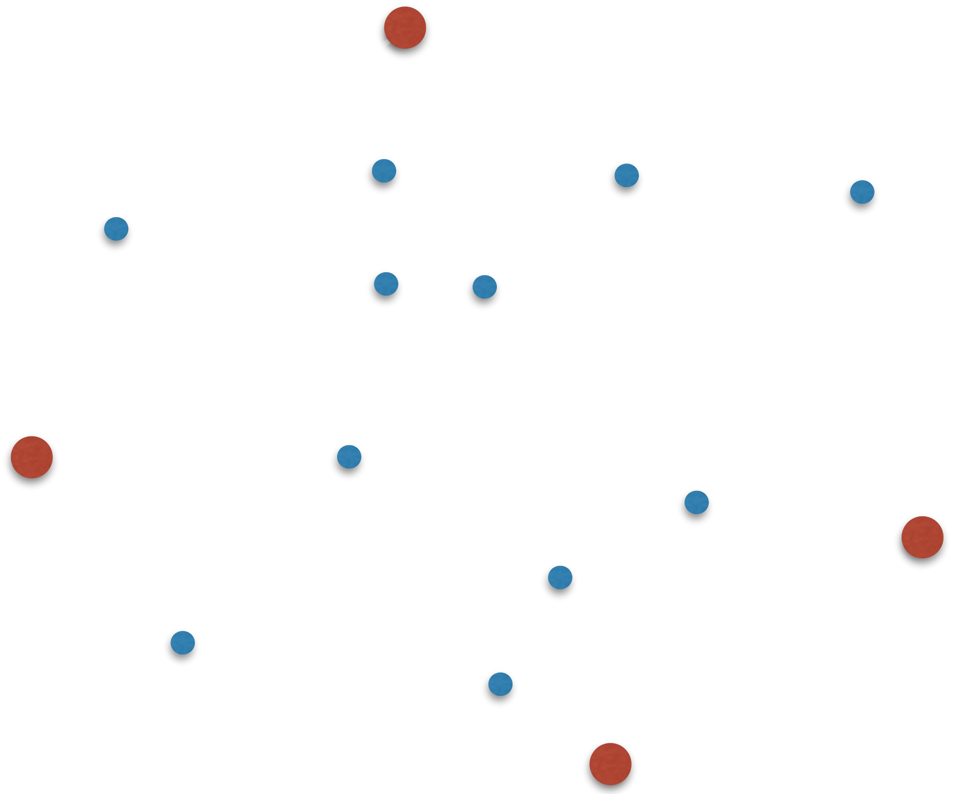
Gift wrapping (1970)



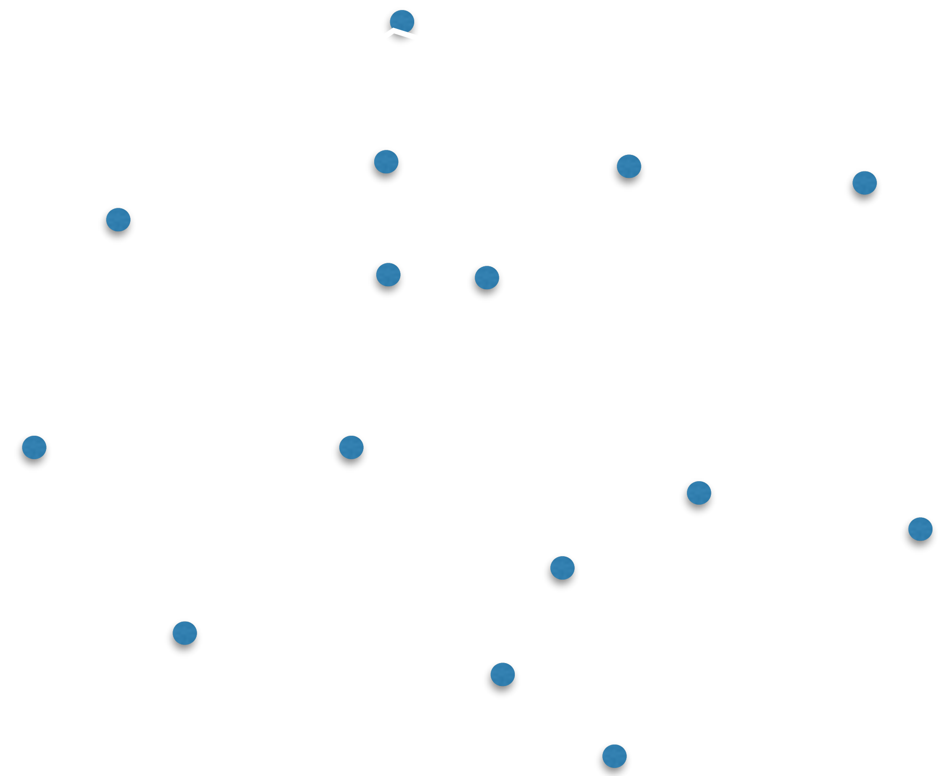
Can you think of some points that are guaranteed to be in CH?

Gift wrapping (1970)

- 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
- Proof

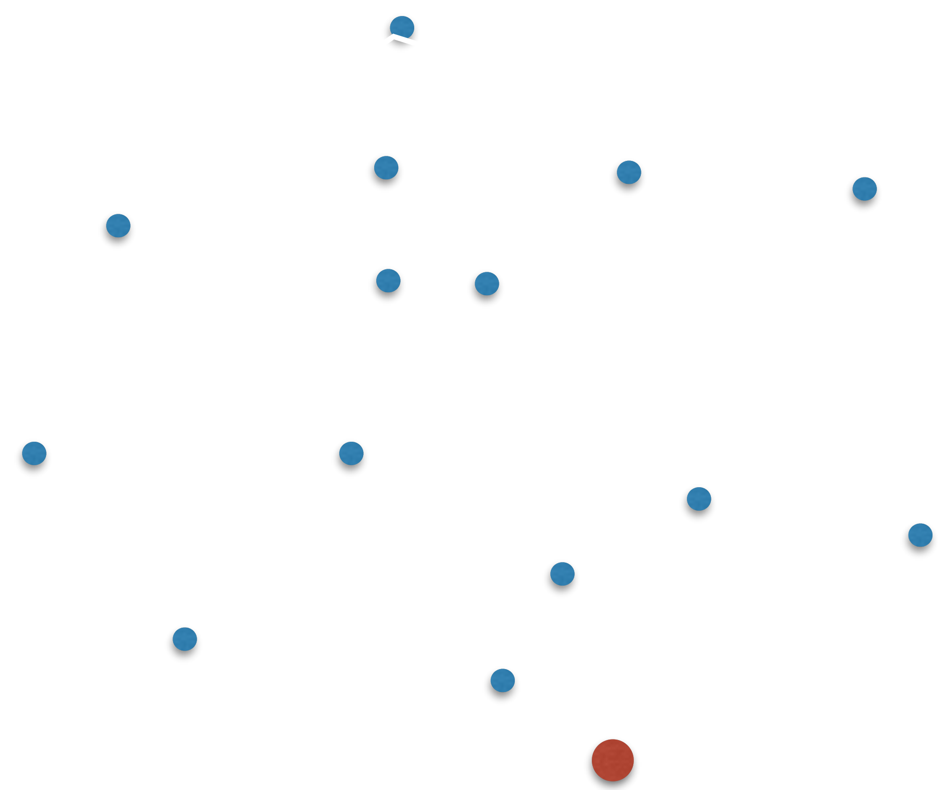


Gift wrapping (1970)



Gift wrapping (1970)

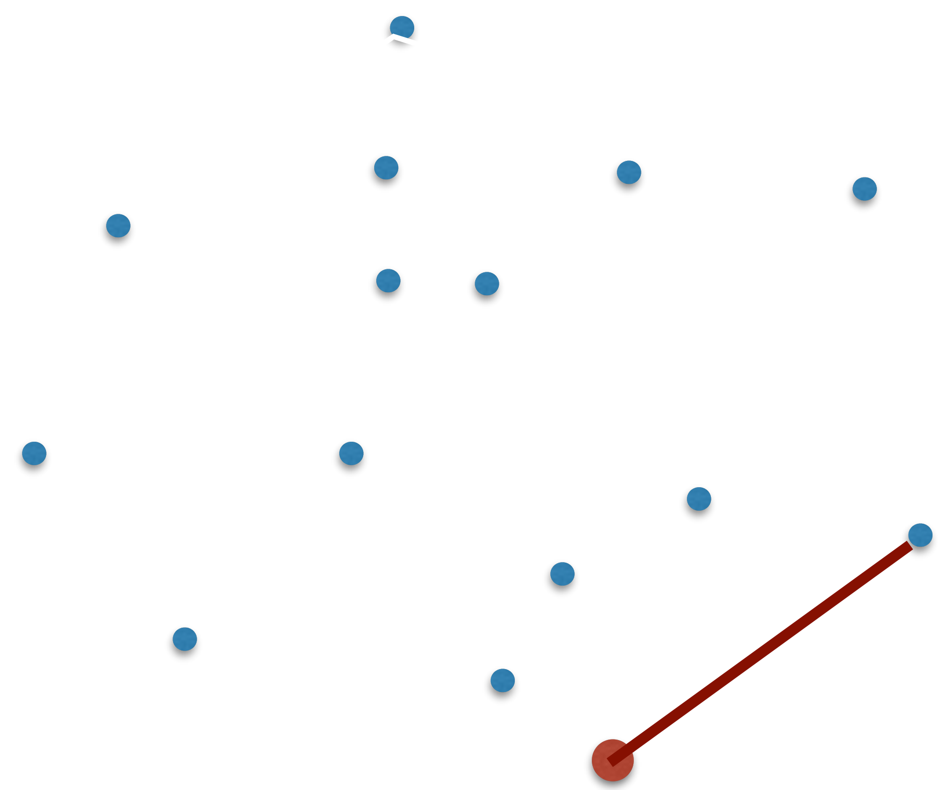
- Start from bottom-most point
    - if more then one, pick right most



Gift wrapping (1970)

- Start from bottom-most point
    - if more then one, pick right most

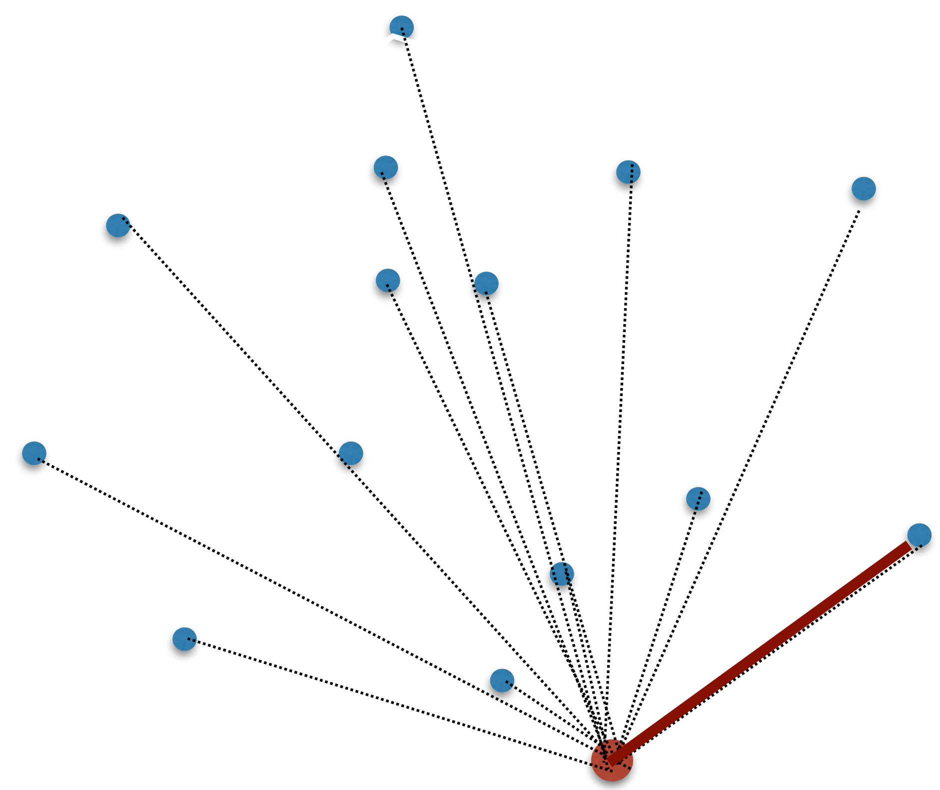
//find first edge. HOW ?



Gift wrapping (1970)

- Start from bottom-most point
    - if more then one, pick right most

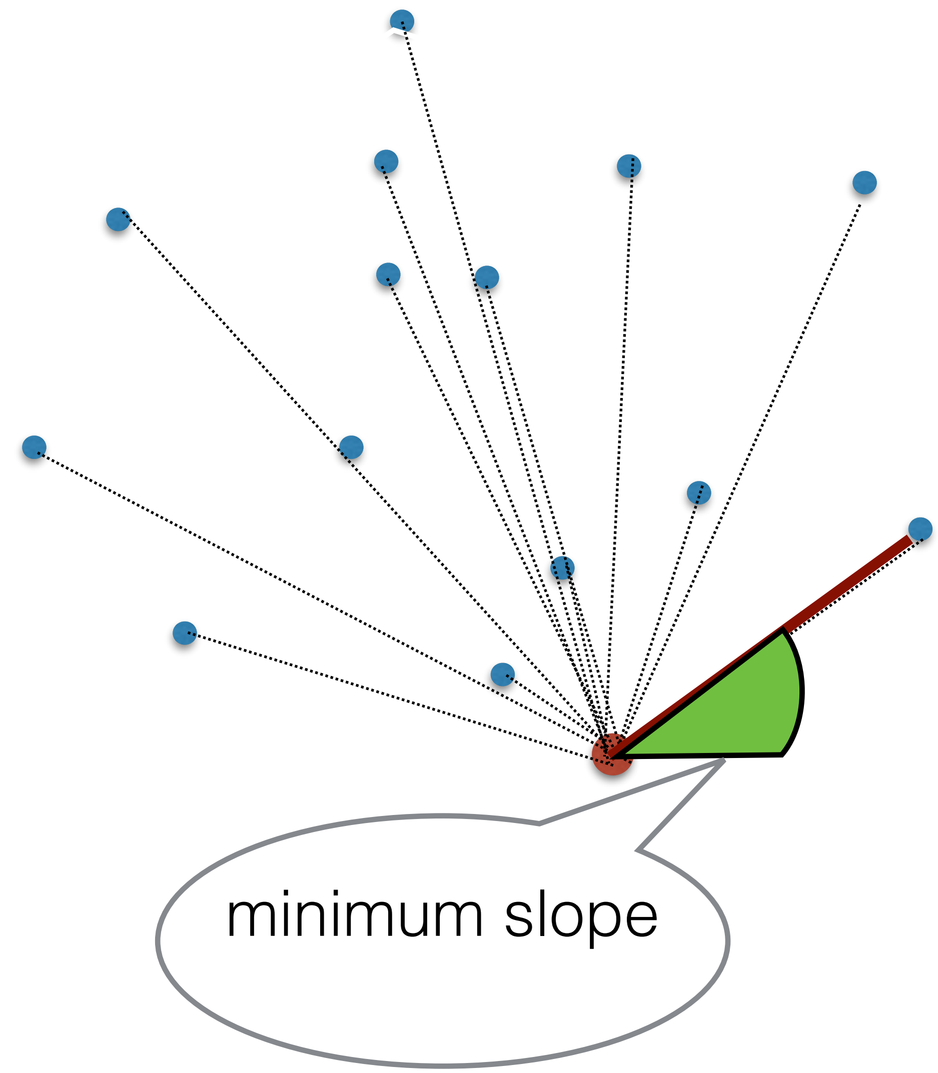
//find first edge. HOW ?



Gift wrapping (1970)

- Start from bottom-most point
  - if more then one, pick right most

//find first edge. HOW ?



Gift wrapping (1970)

- Start from bottom-most point
  - if more then one, pick right most

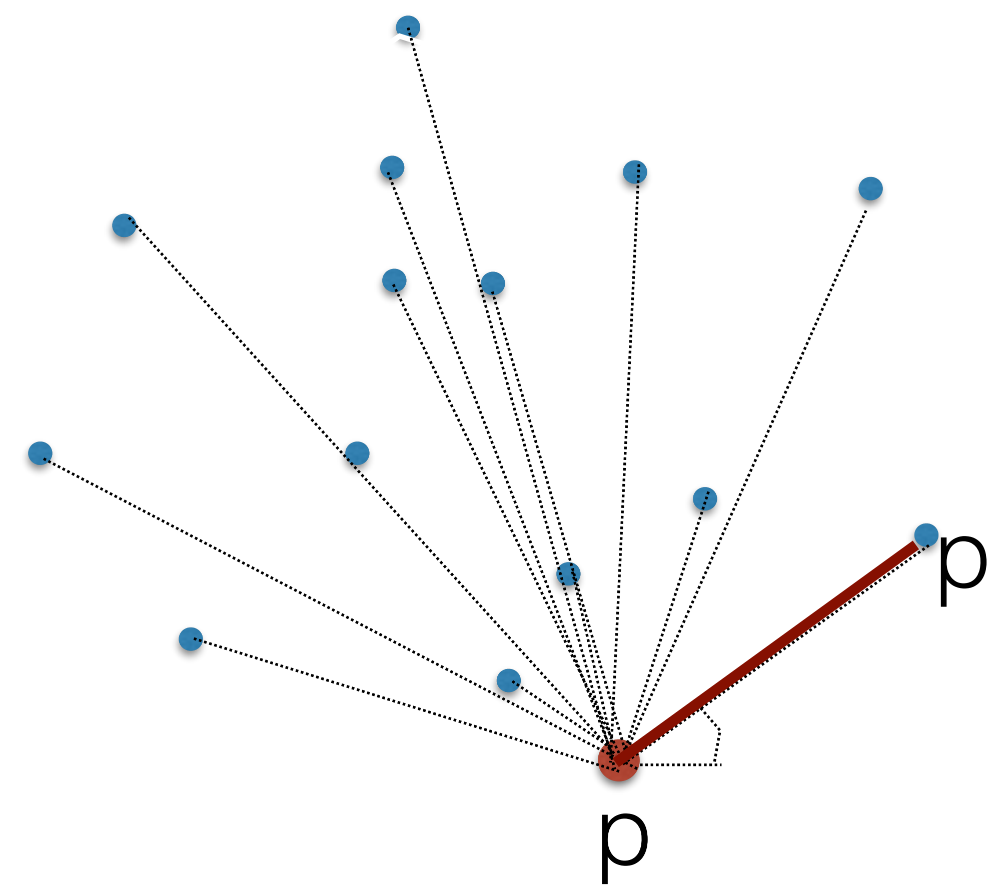
/******** find first edge ********/

- for each point q (q != p)
  - compute slope of q wrt p
- let p’ = point with smallest slope

//claim: pp’ is extreme edge

- output (p, p’) as first edge

/*********what next ? ********/



Gift wrapping (1970)

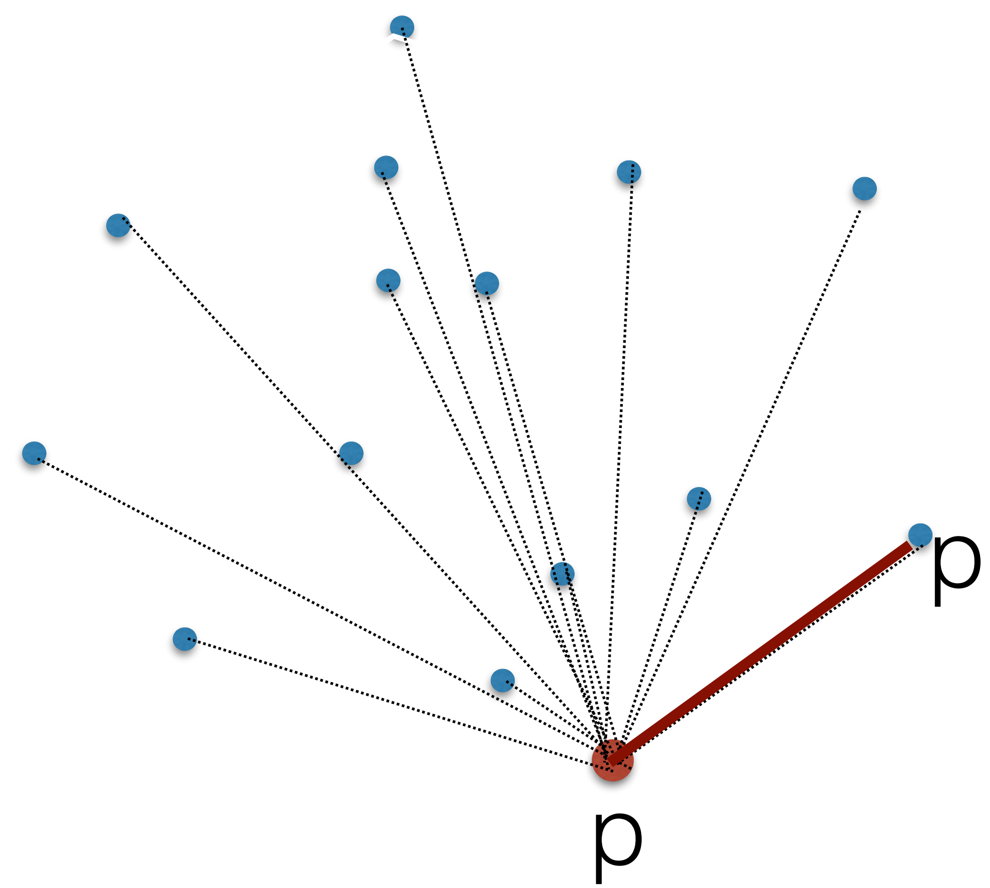
- Start from bottom-most point
  - if more then one, pick right most

/******** find first edge ********/

- for each point q (q != p)
  - compute slope of q wrt p
- let p' = point with smallest slope

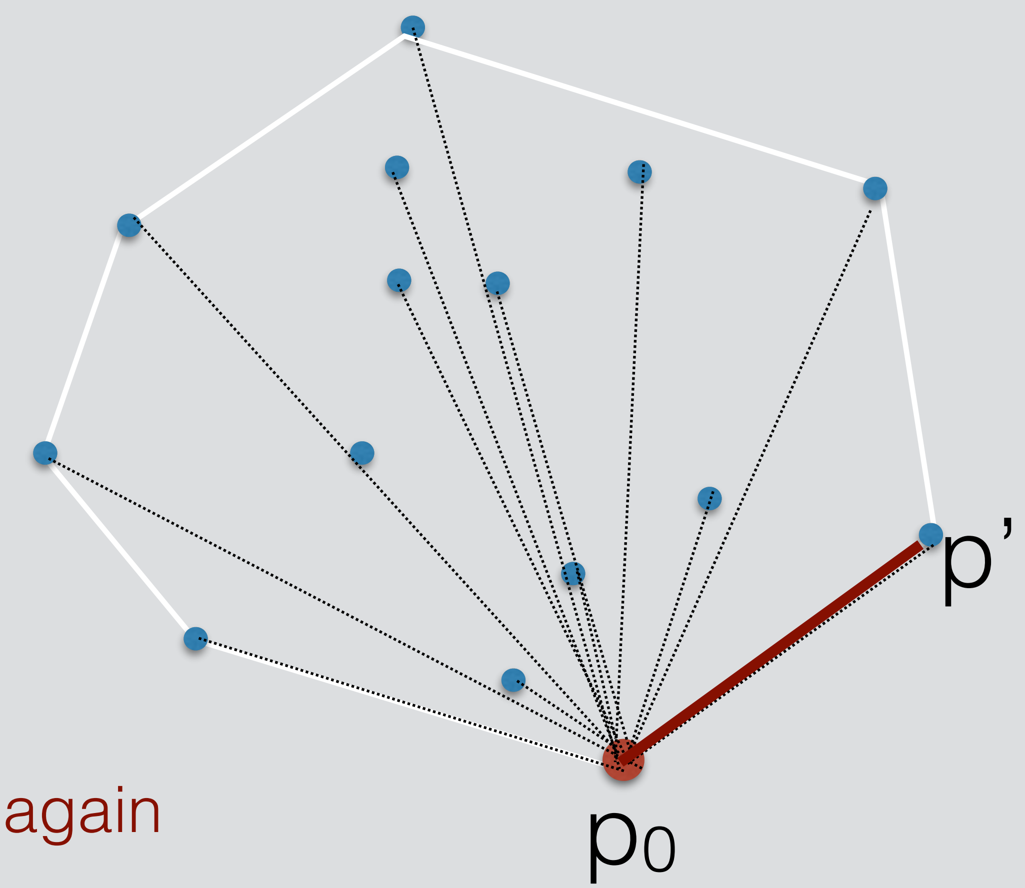
//claim: pp' is extreme edge

- output (p, p') as first edge
- repeat from p'



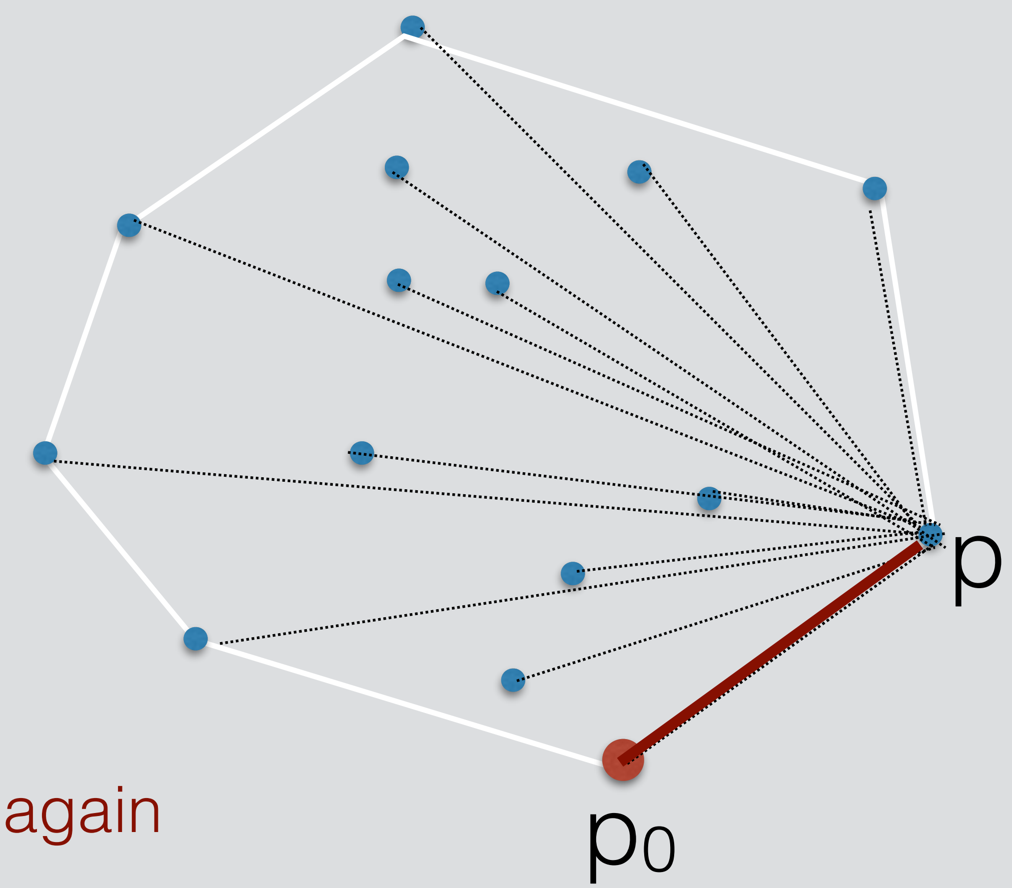
Gift wrapping (1970)

- p_0 = point with smallest y-coordinate (if more then one, pick right most)
- $p = p_0$
- repeat
  - for each point q (q != p)
    - compute ccw-angle of q wrt p
  - let p' = point with smallest angle
  - output (p, p') as CH edge
  - p = p'
- until $p = p_0$ //until it discovers first point again



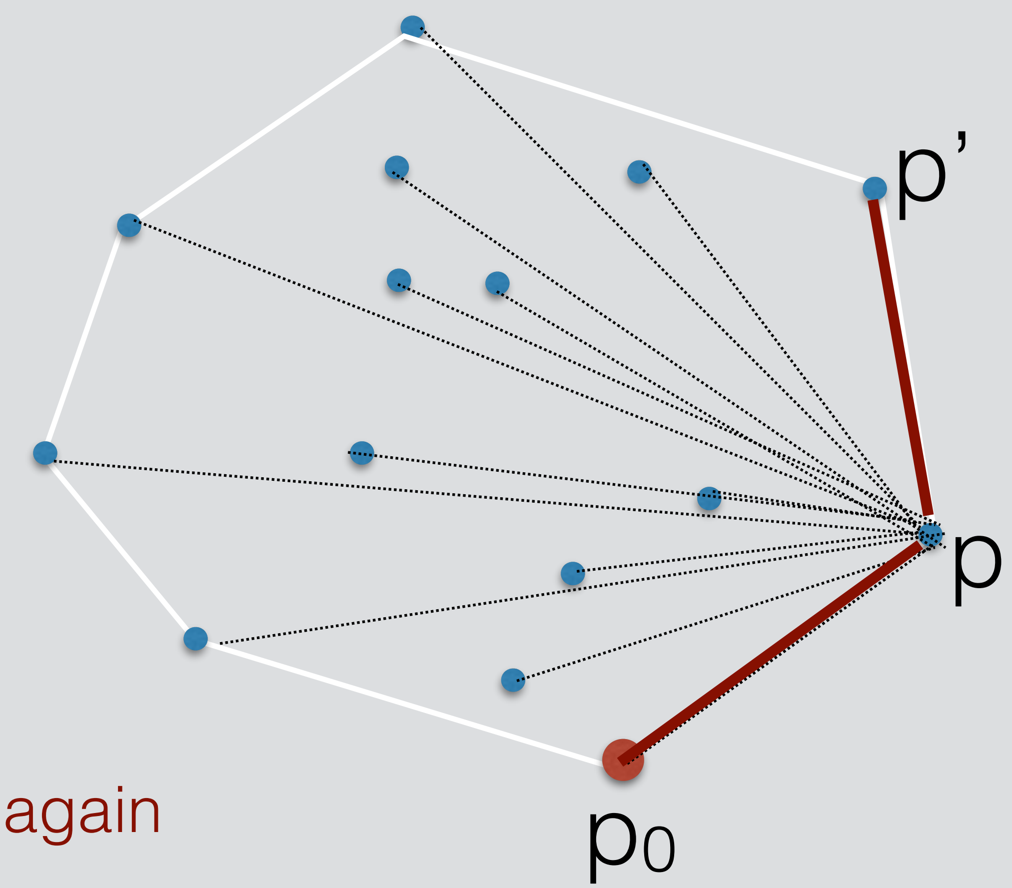
Gift wrapping (1970)

- p_0 = point with smallest y-coordinate (if more then one, pick right most)
- p = p_0
- repeat
  - for each point q (q != p)
    - compute ccw-angle of q wrt p
  - let p' = point with smallest angle
  - output (p, p') as CH edge
  - p = p'
- until p = p_0 //until it discovers first point again



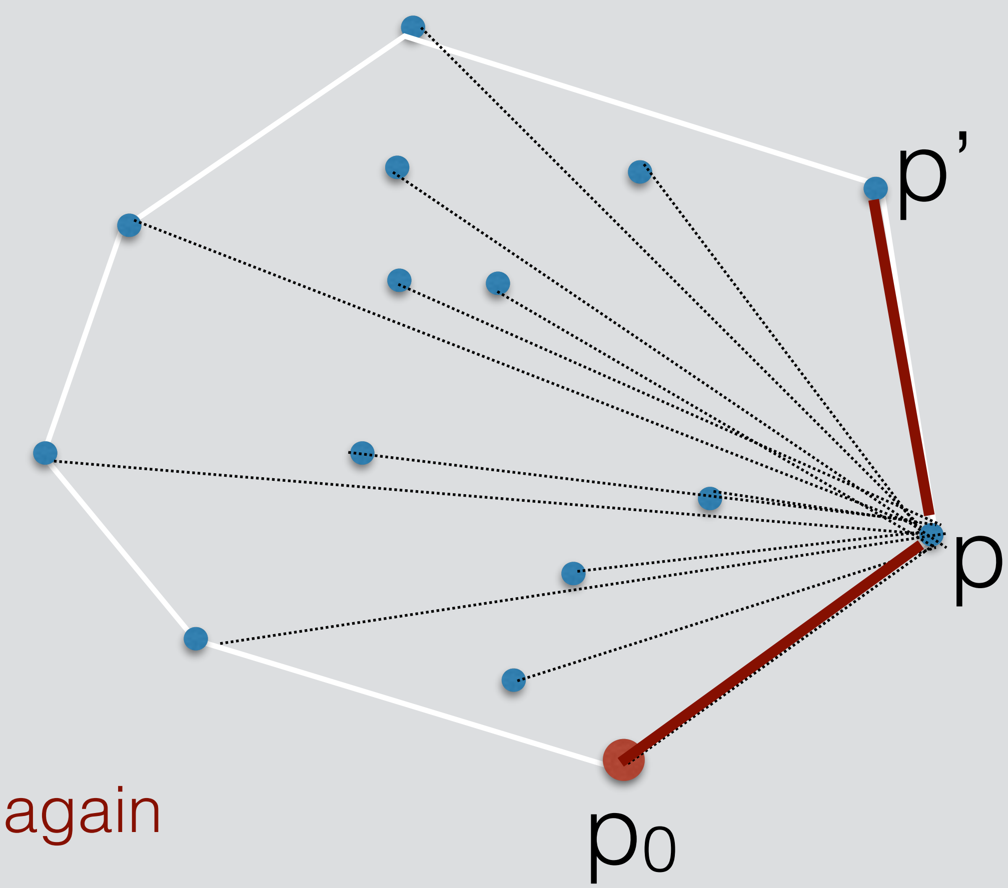
Gift wrapping (1970)

- p_0 = point with smallest y-coordinate (if more then one, pick right most)
- $p = p_0$
- repeat
    - for each point q (q != p)
        - compute ccw-angle of q wrt p
    - let p' = point with smallest angle
    - output (p, p') as CH edge
    - p = p'
- until $p = p_0$ //until it discovers first point again



Gift wrapping (1970)

- p_0 = point with smallest y-coordinate (if more then one, pick right most)
- p = p_0
- repeat
  - for each point q (q != p)
    - compute ccw-angle of q wrt p
  - let p' = point with smallest angle
  - output (p, p') as CH edge
  - p = p'
- until p = p_0 //until it discovers first point again



Gift wrapping : Classwork

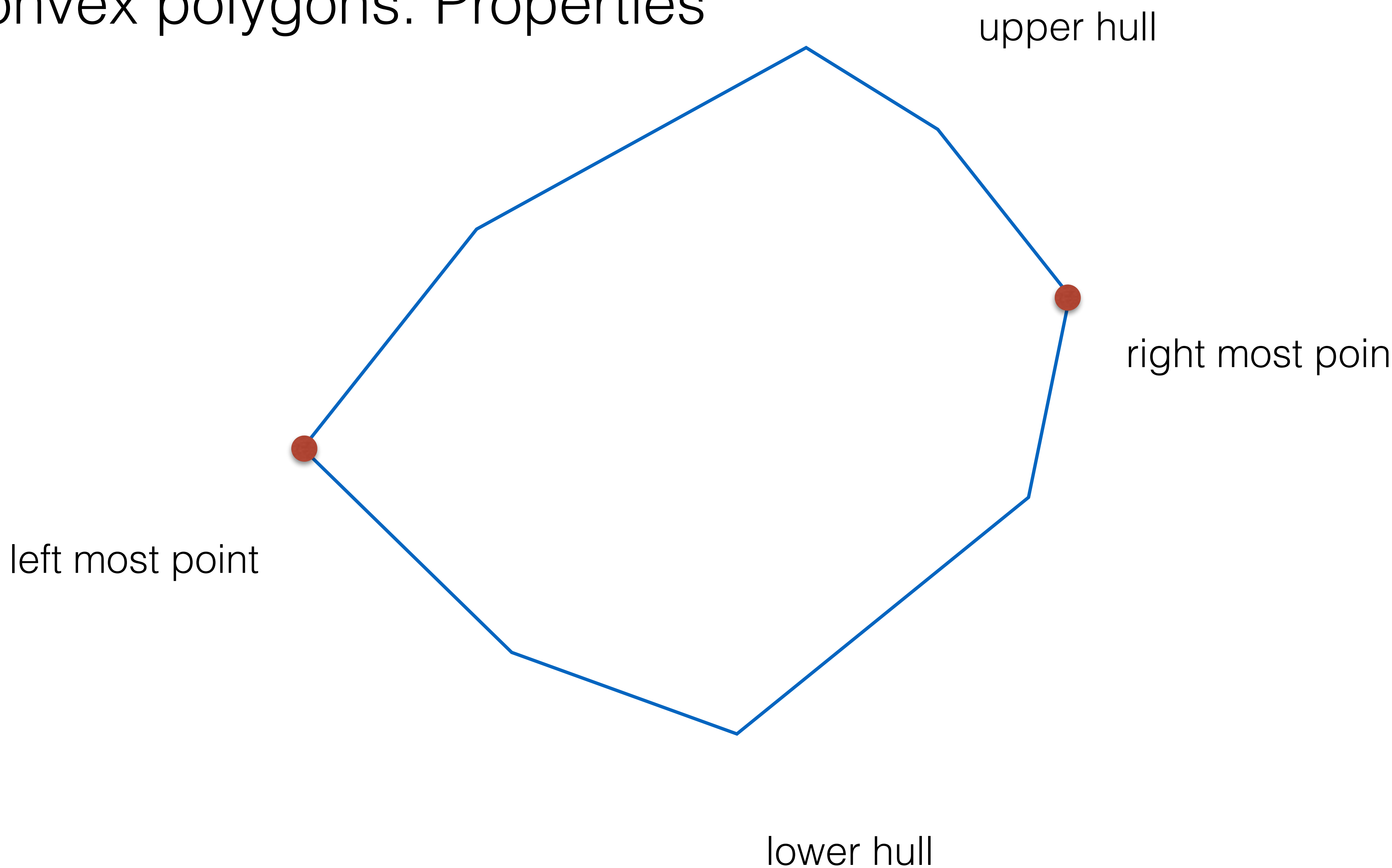
- Simulate GiftWrapping on a set of points and think how it works in degenerate cases
- Analysis: Running time? Express function of n and k, where k is the output size (number of points on the convex hull)
    - How small/large can k be for a set of n points?
    - Show examples that trigger best/worst cases
- Discuss when gift-wrapping is a good choice

Summary

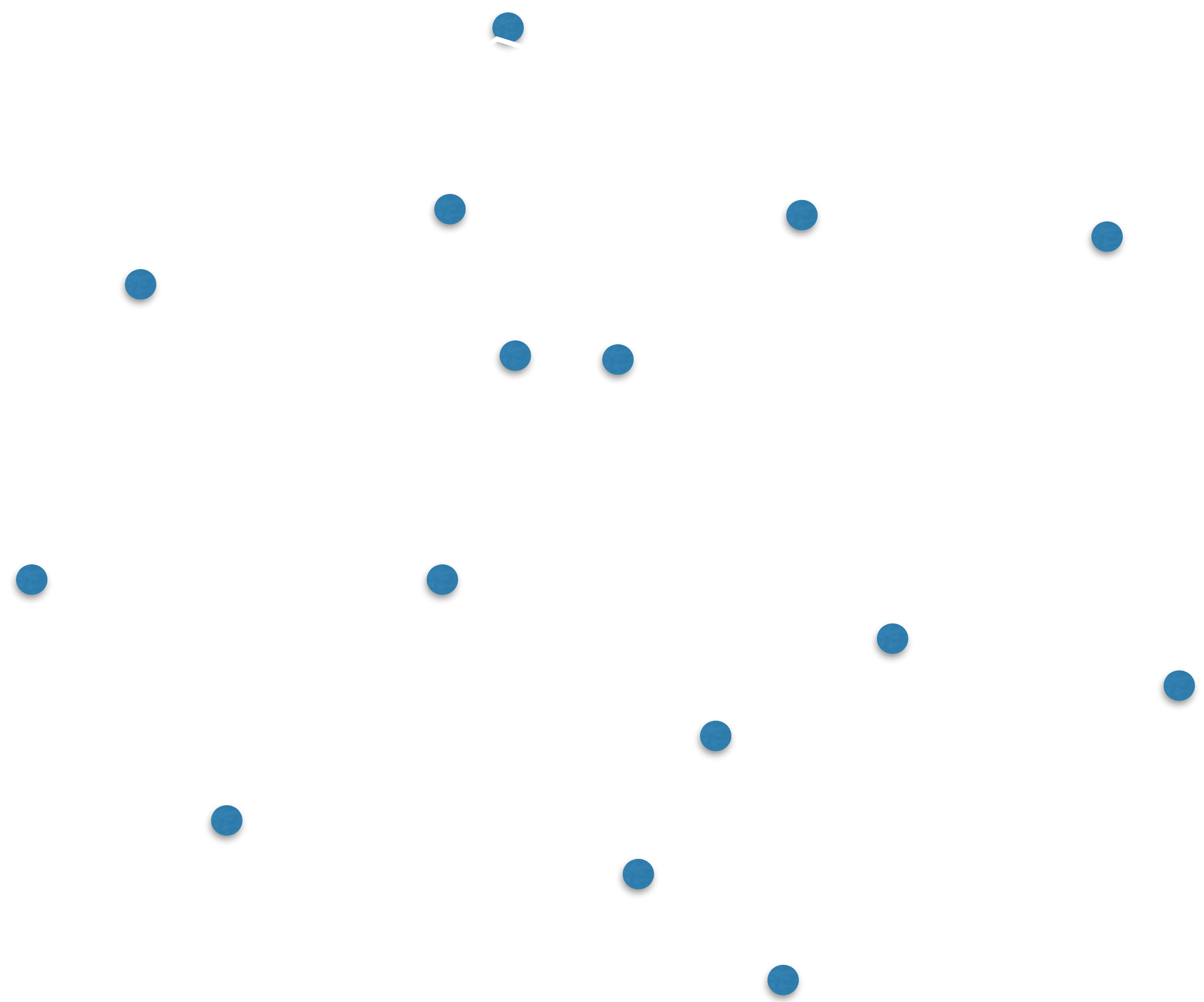
- **Gift wrapping**
  - Runs in O(kn) time, where k is the size of the CH(P)
  - Efficient if k is small
  - For k = O(n), gift wrapping takes O(n^2)
  - Faster algorithms are known
  - Gift wrapping extends easily to 3D and for many years was the primary algorithm for 3D

Quickhull

Convex polygons: Properties

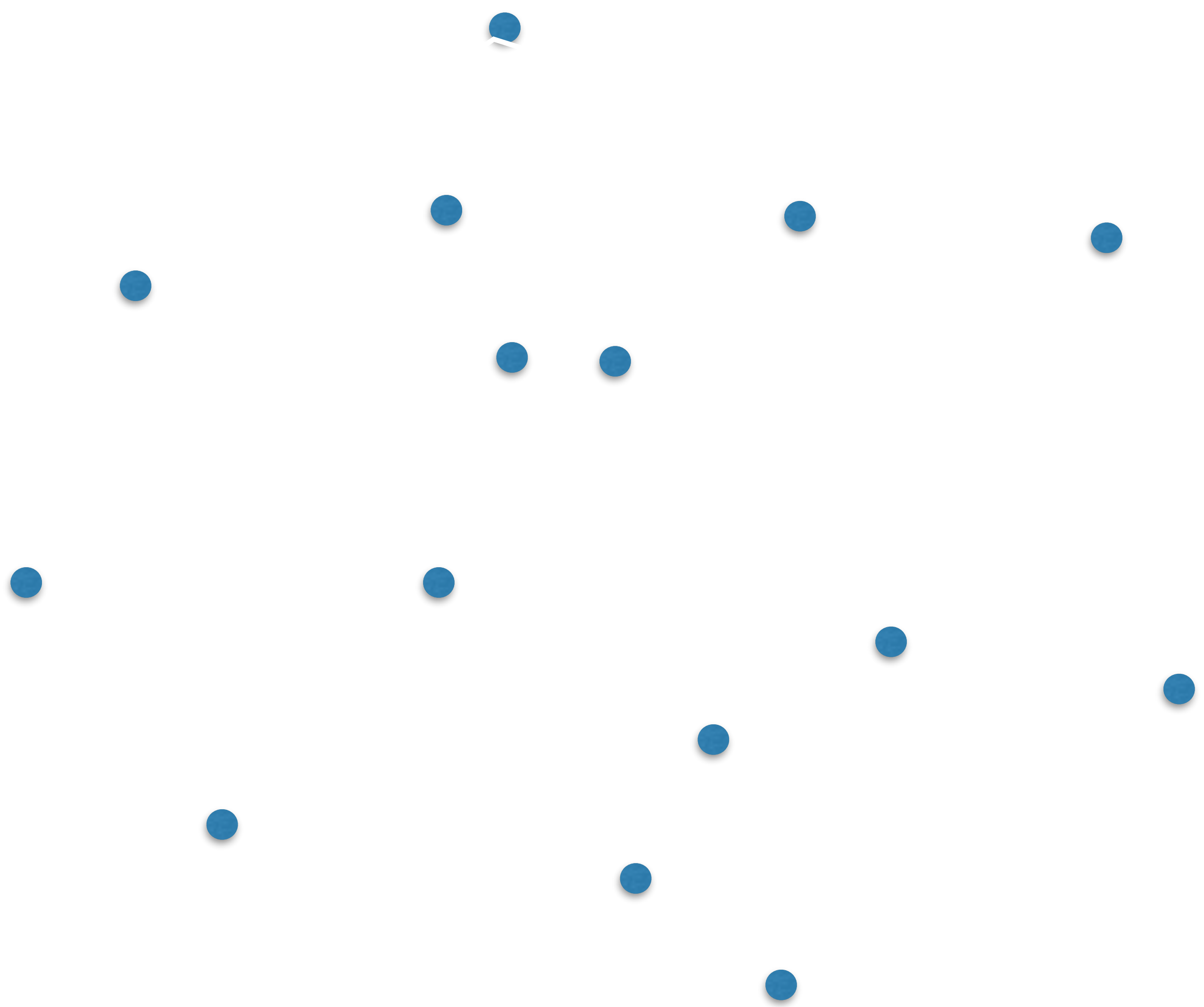


Quickhull (late 1970s)



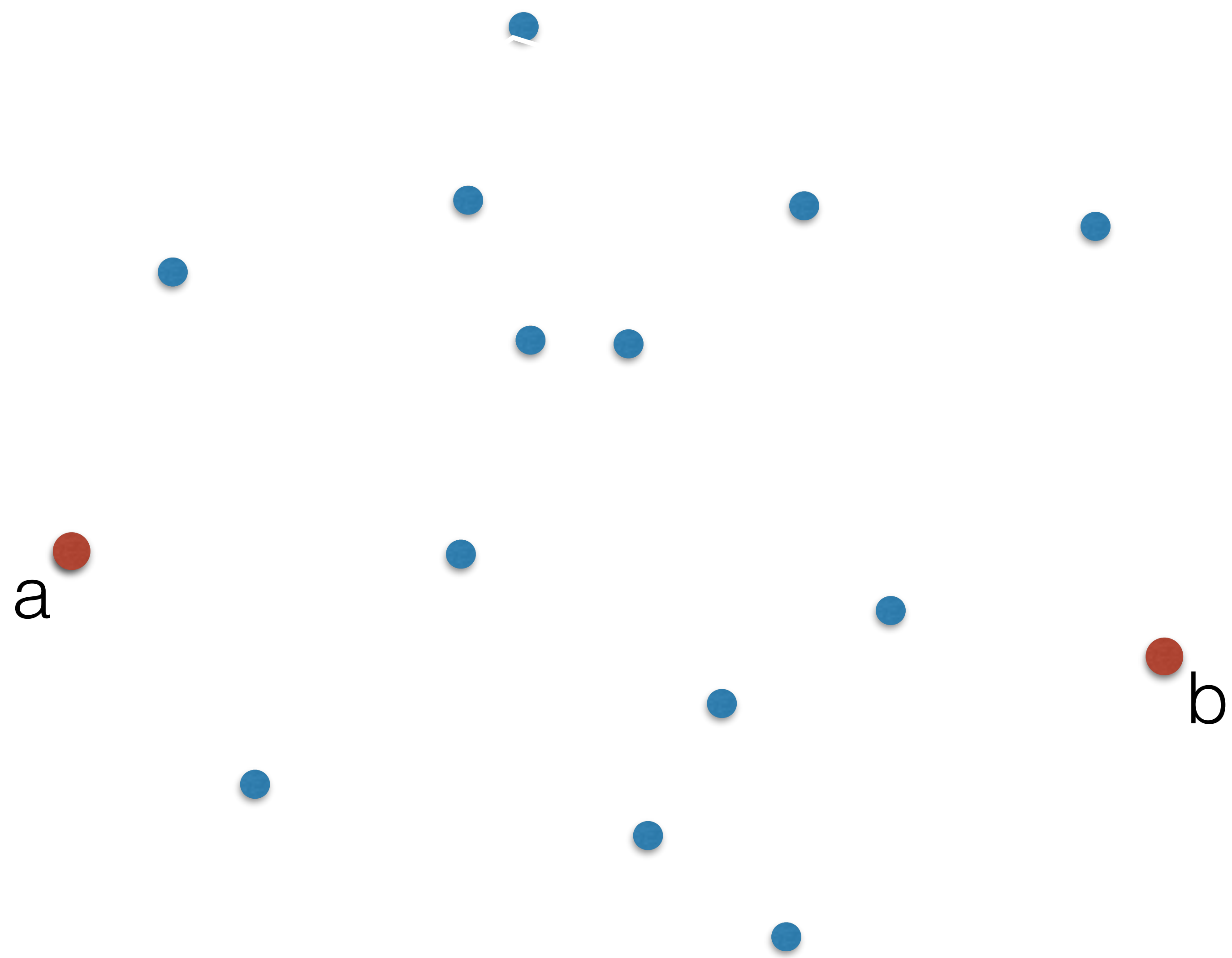
Quickhull (late 1970s)

- Similar to Quicksort (in some way)



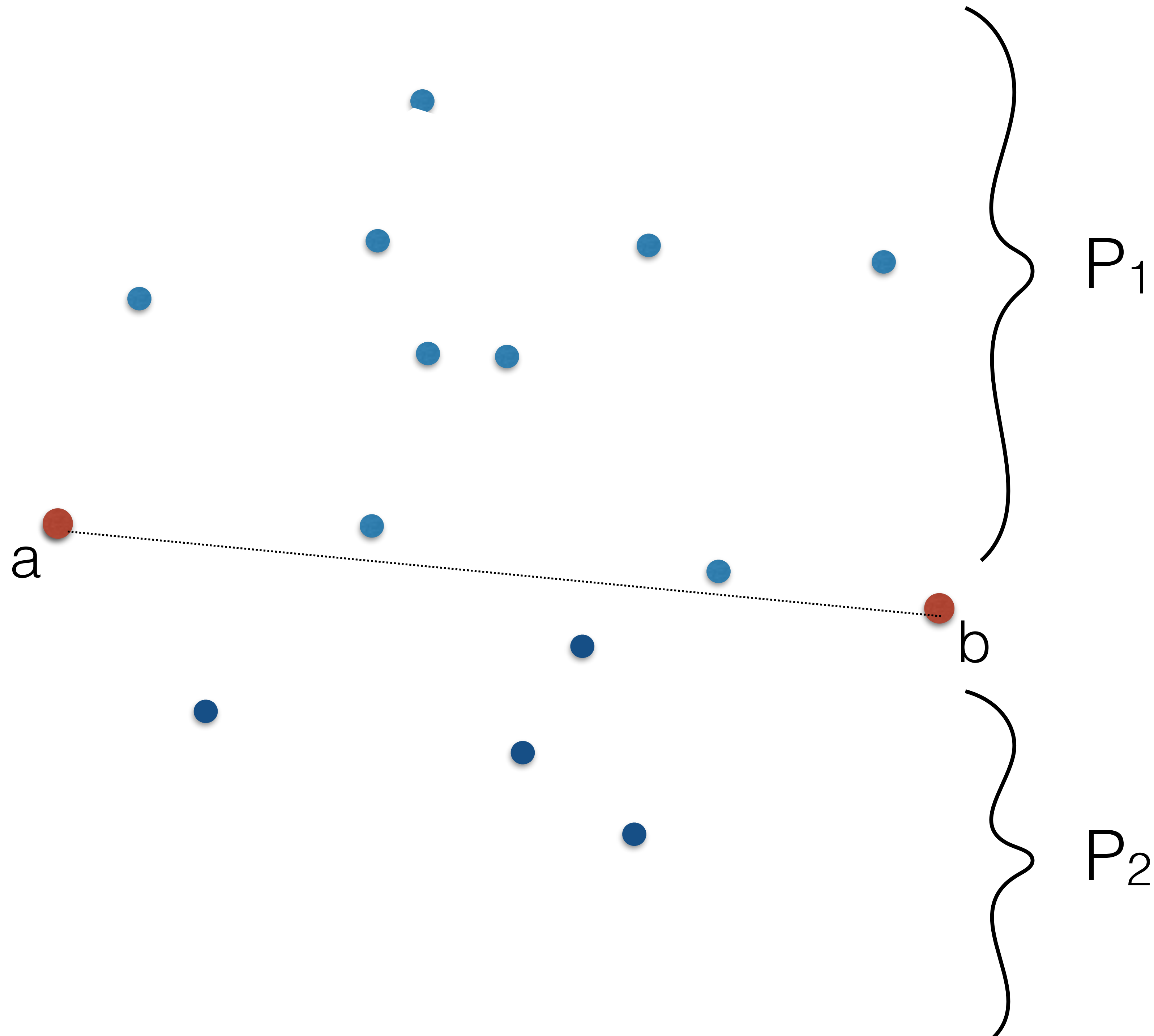
Quickhull (late 1970s)

- Idea: start with 2 extreme points



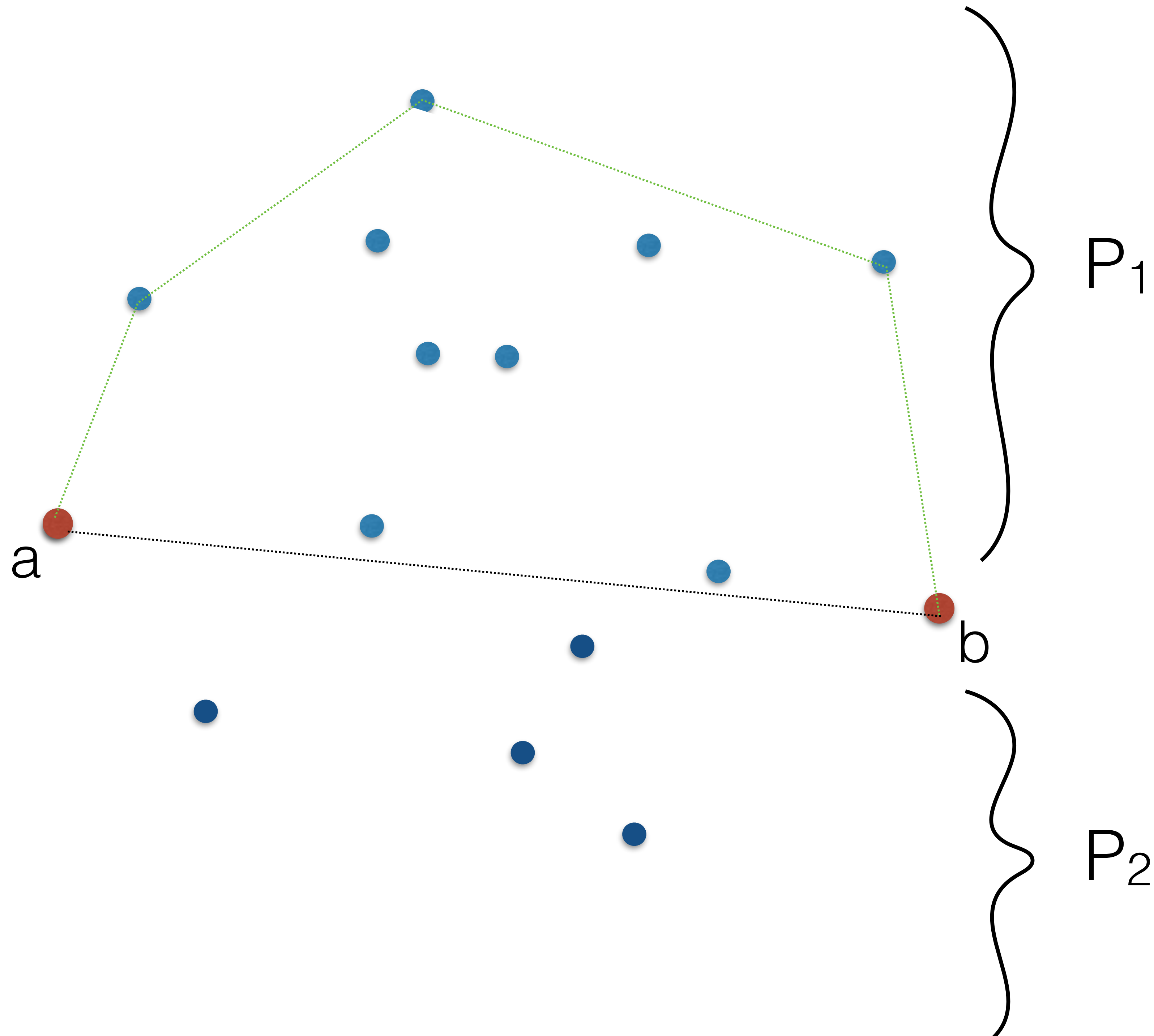
Quickhull (late 1970s)

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



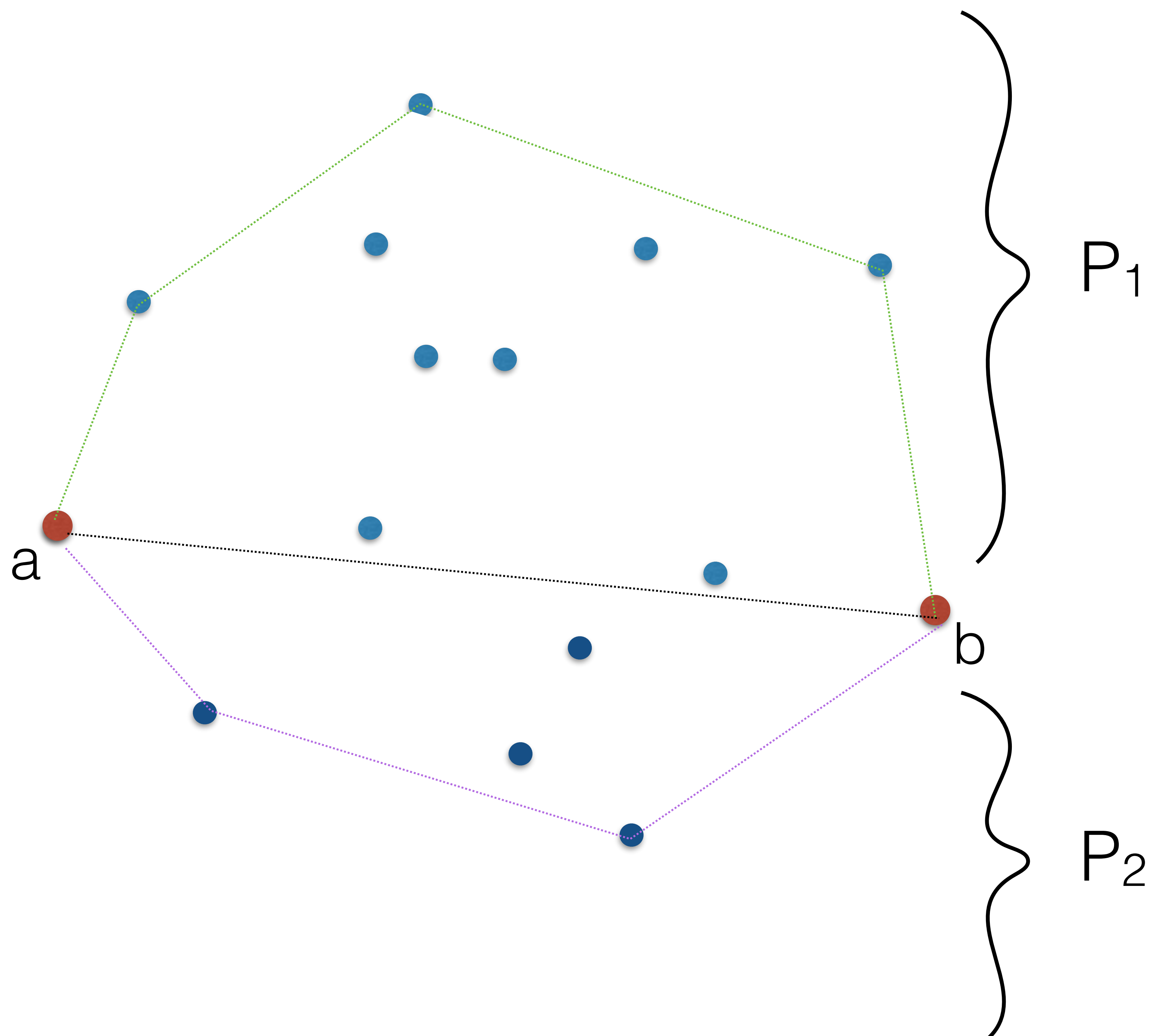
Quickhull (late 1970s)

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



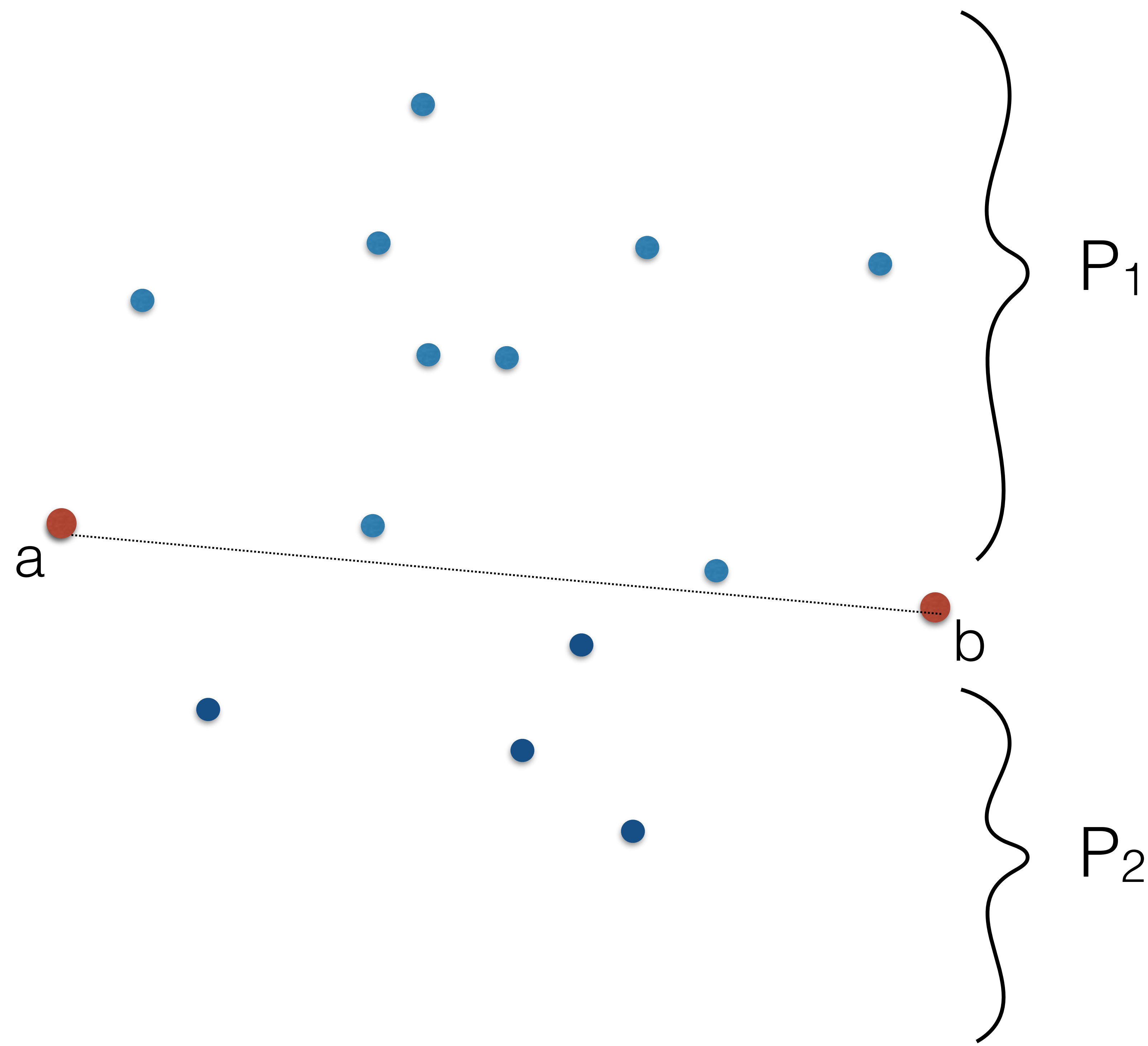
Quickhull (late 1970s)

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



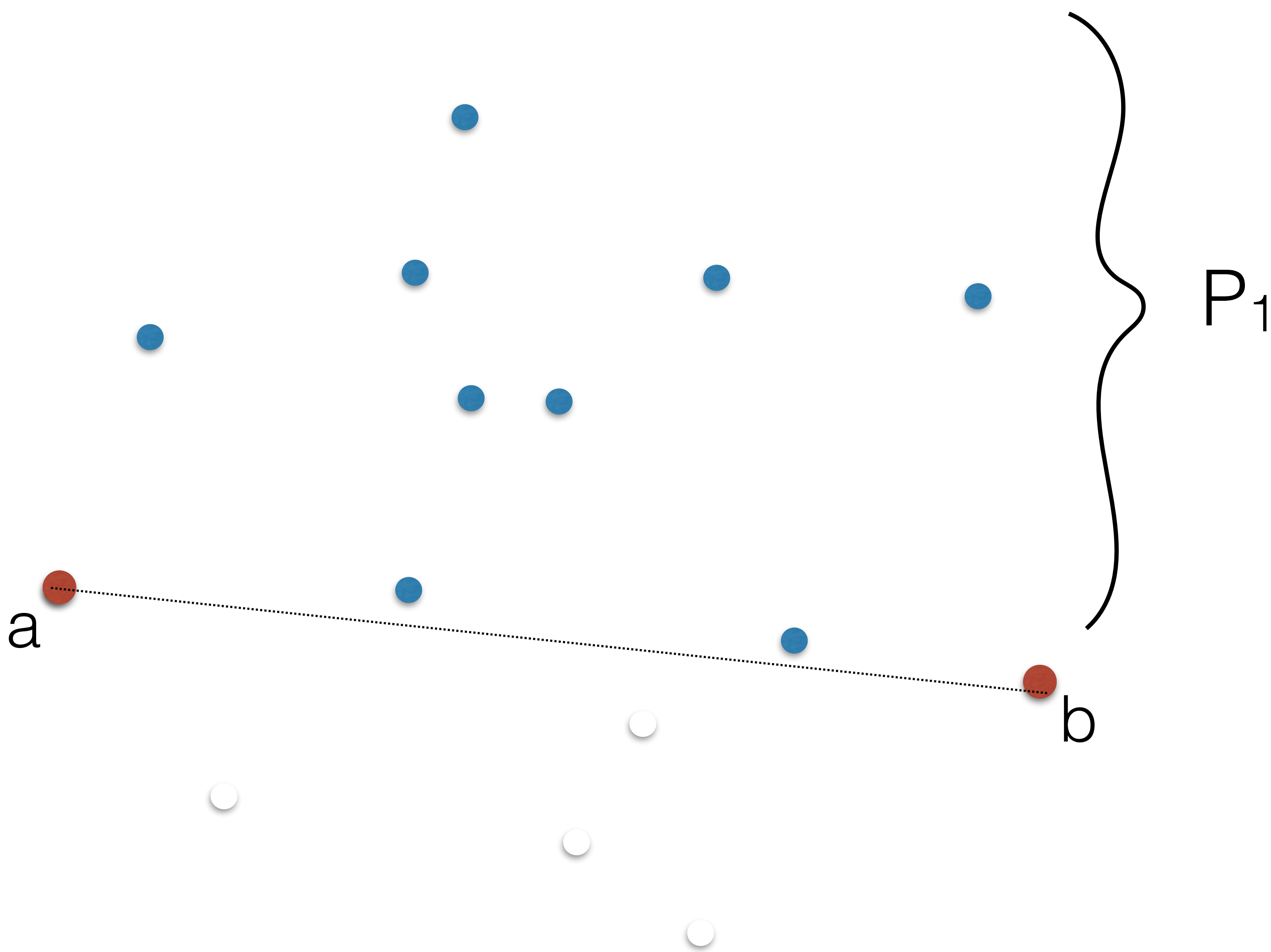
Quickhull (late 1970s)

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



Quickhull (late 1970s)

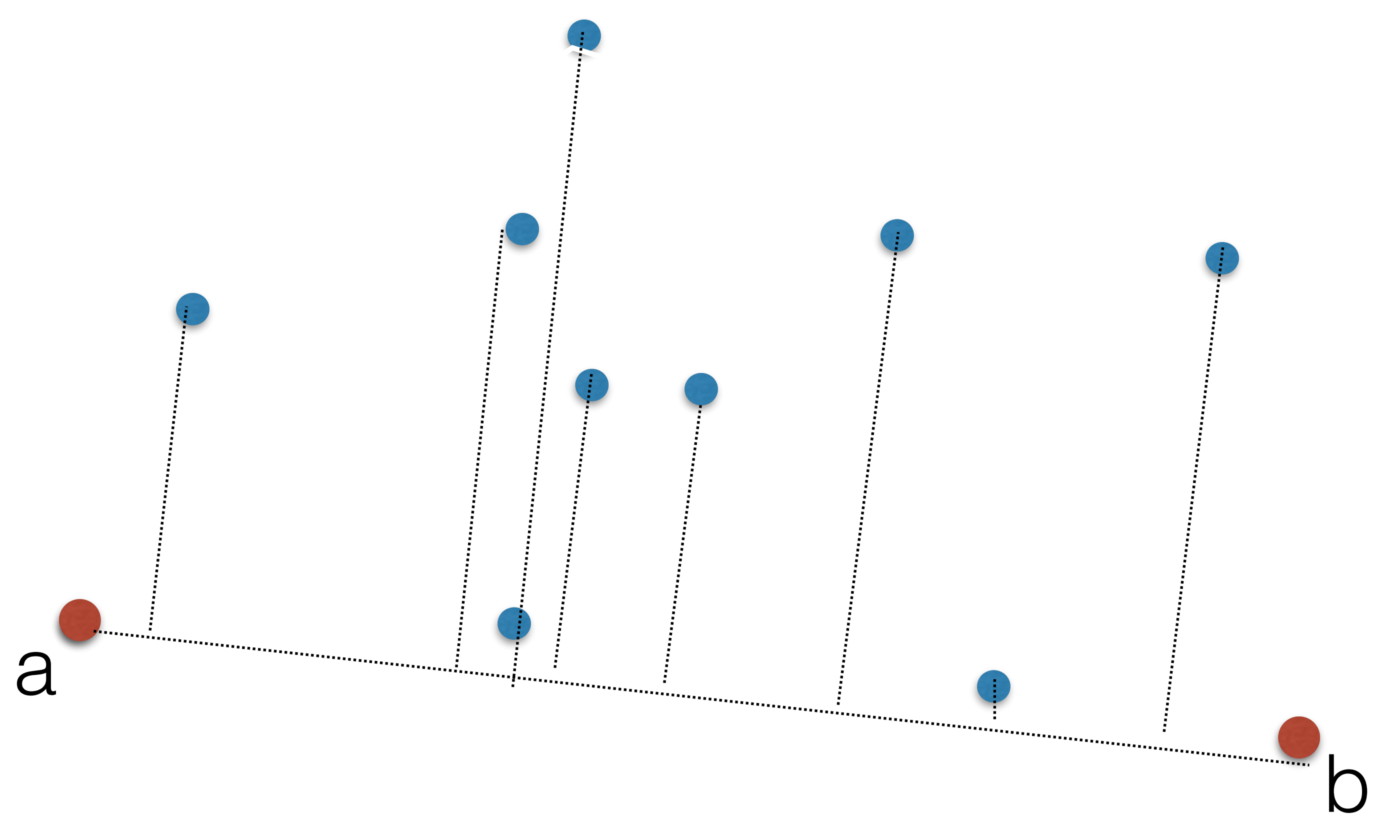
- First let's focus on P1



Quickhull (late 1970s)

let's ignore collinear points for now

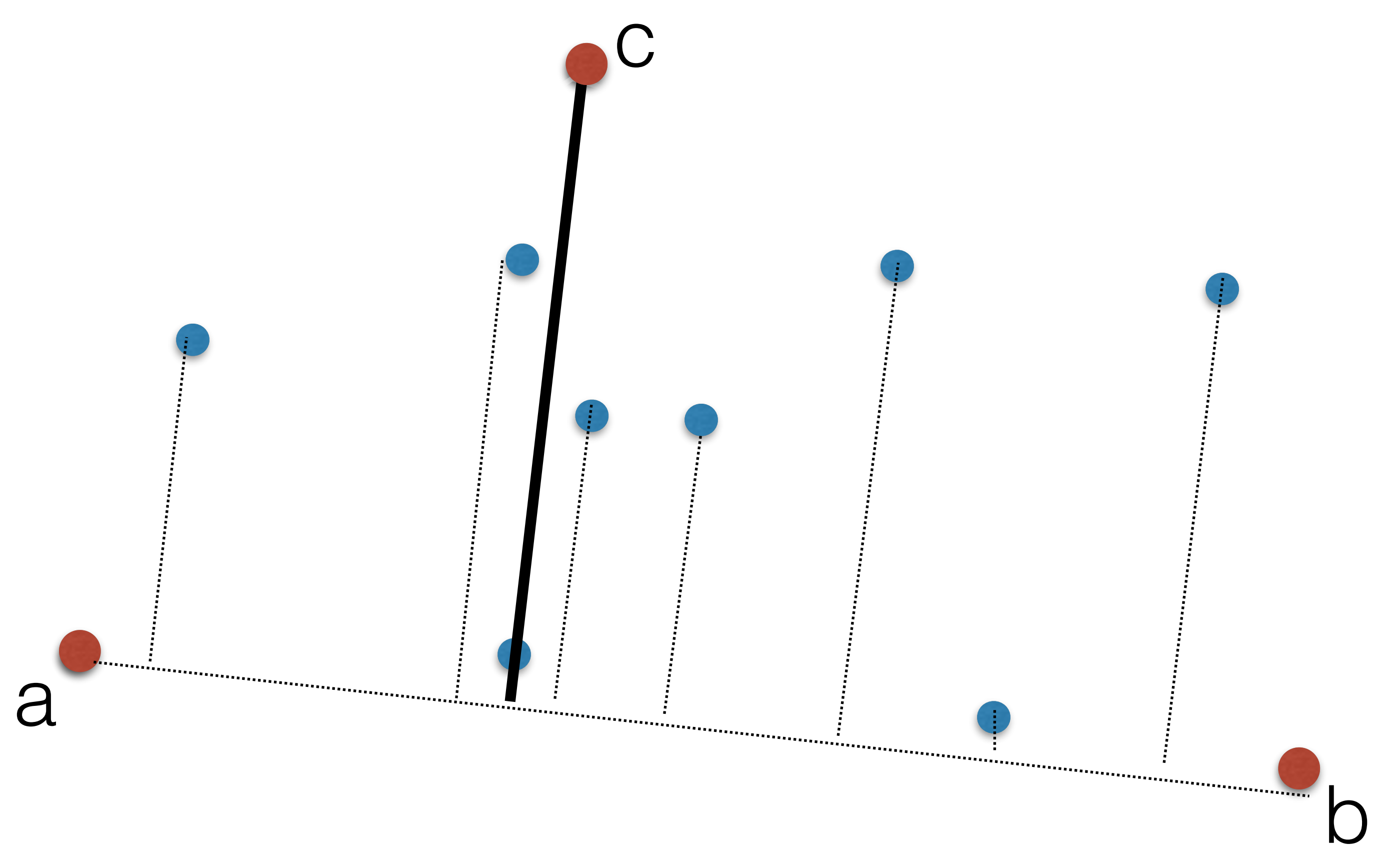
- For all points p in P1: compute dist(p, ab)



Quickhull (late 1970s)

let's ignore collinear points for now

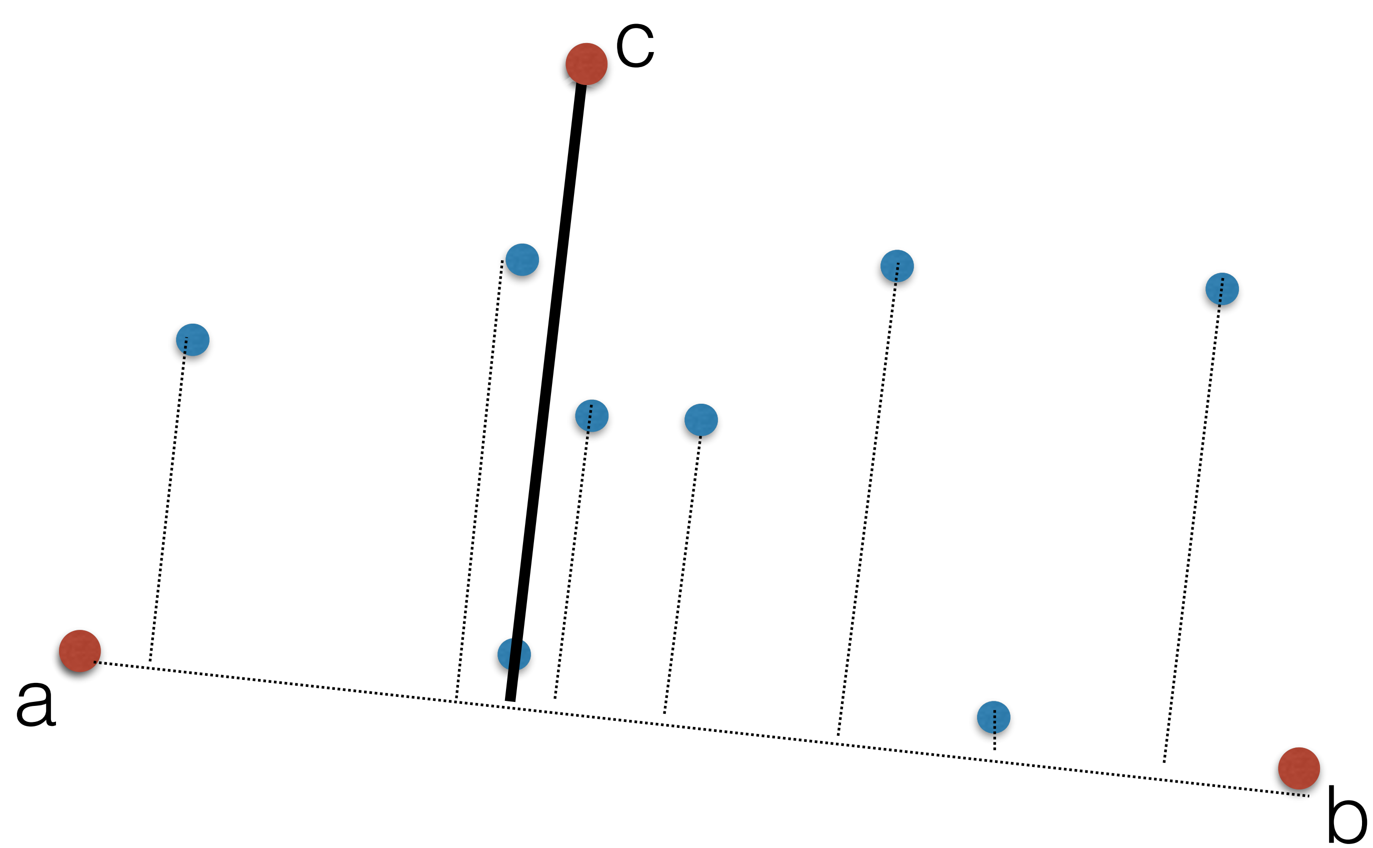
- Find the point c with largest distance (i.e. furthest away from ab)



Quickhull (late 1970s)

let's ignore collinear points for now

- Find the point c with largest distance (i.e. furthest away from ab)

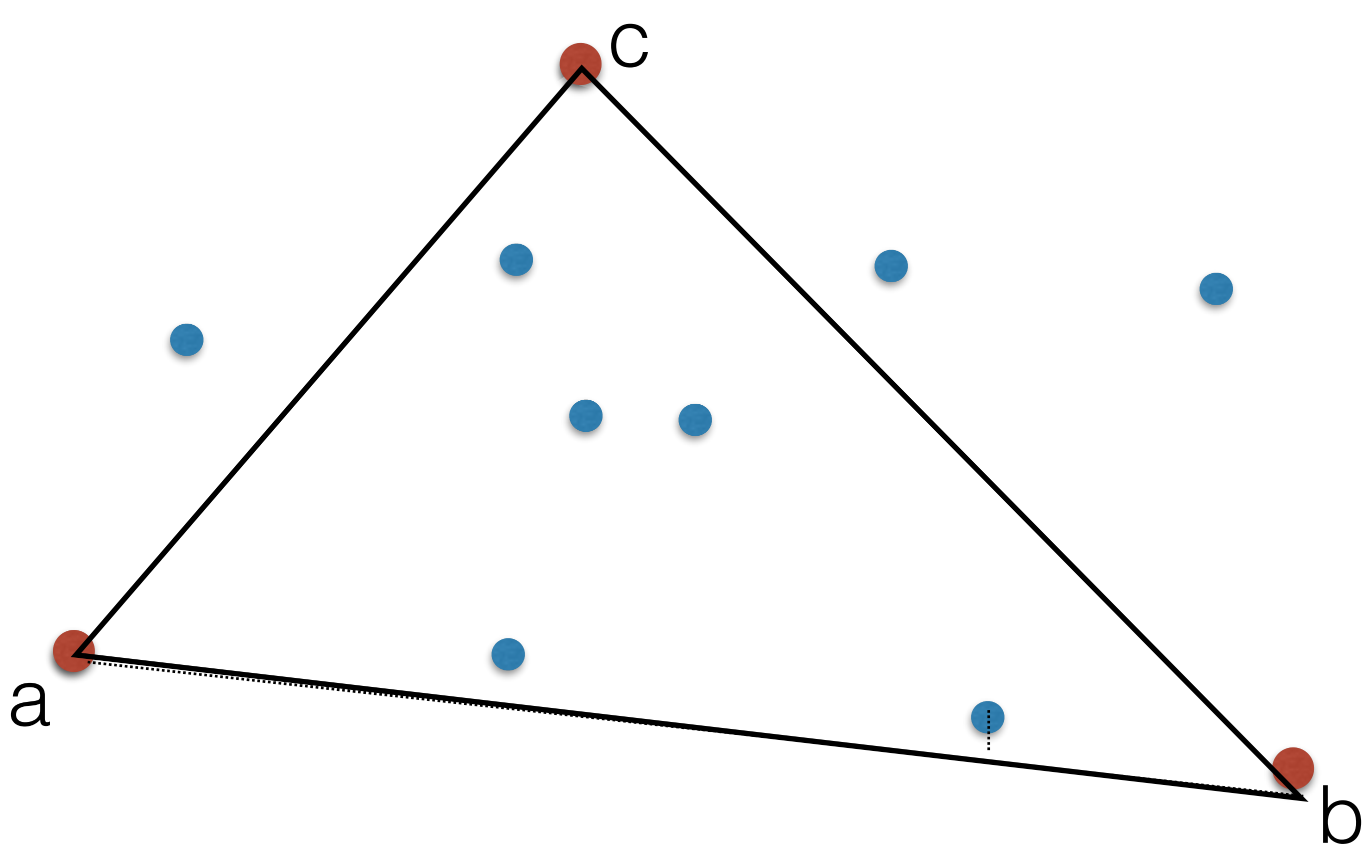


- Claim: c must be an extreme point (and thus on the CH of P1)
- Proof:

Quickhull (late 1970s)

let's ignore collinear points for now

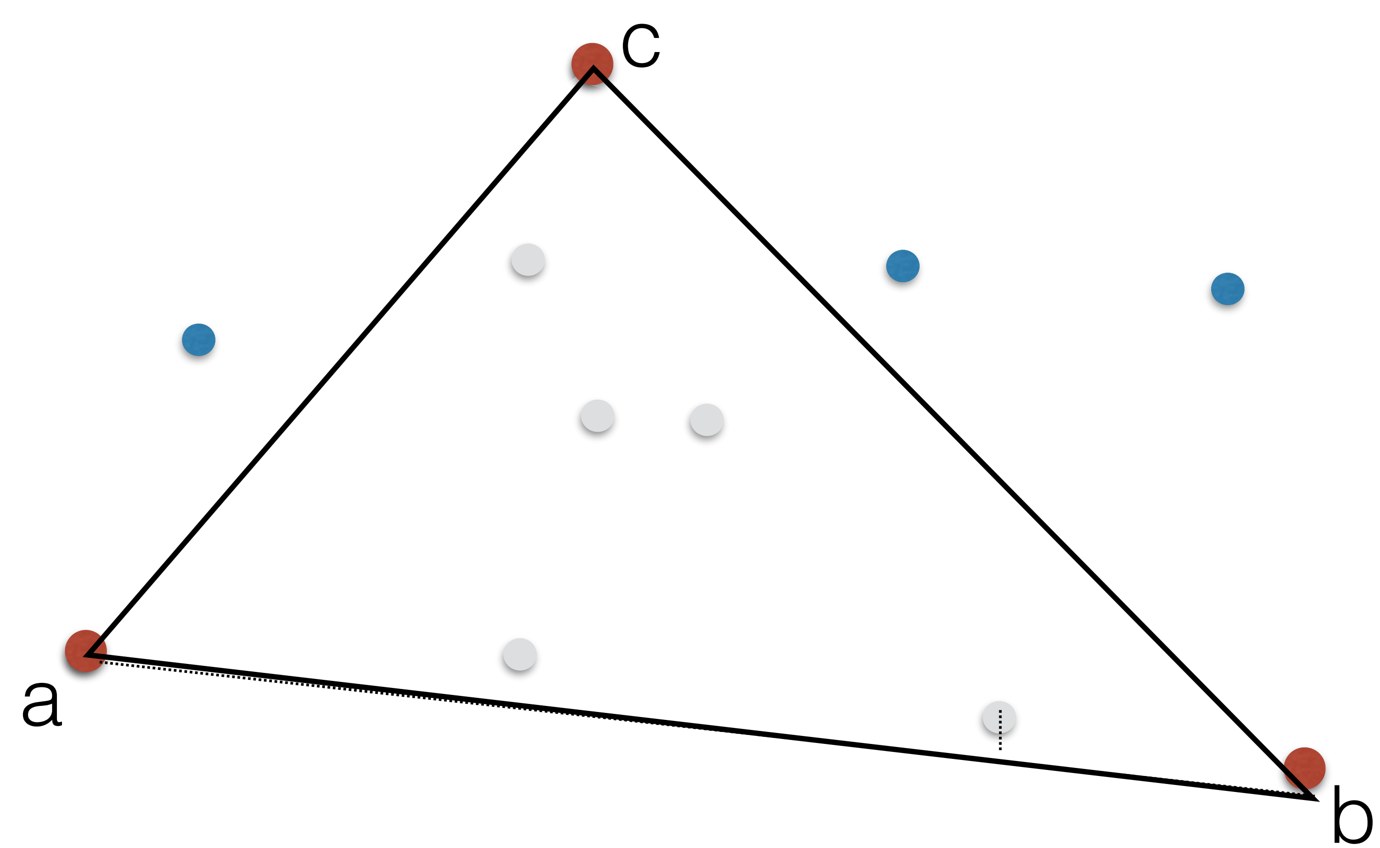
- Discard all points inside triangle abc



Quickhull (late 1970s)

- Discard all points inside triangle abc

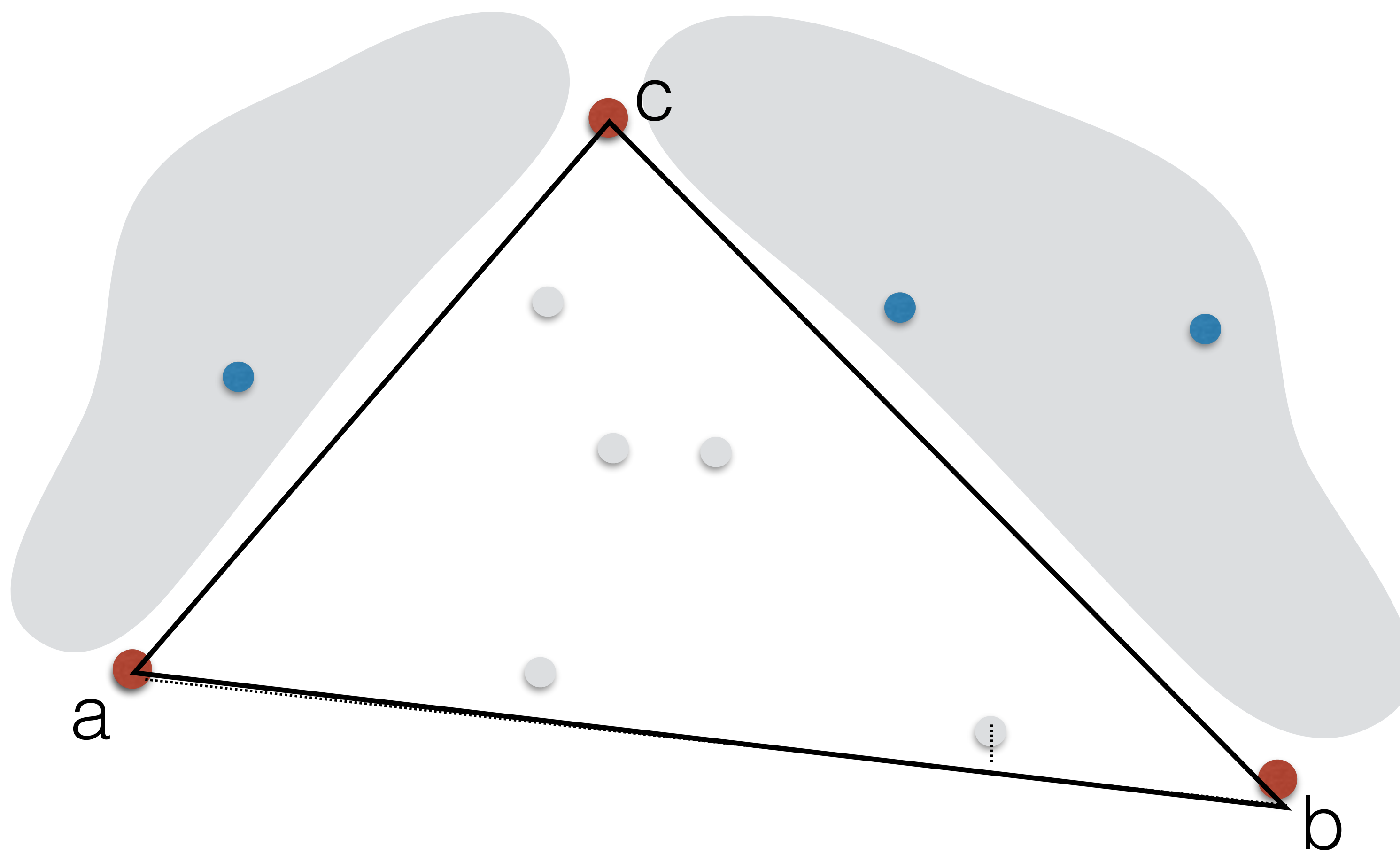
let's ignore collinear points for now



Quickhull (late 1970s)

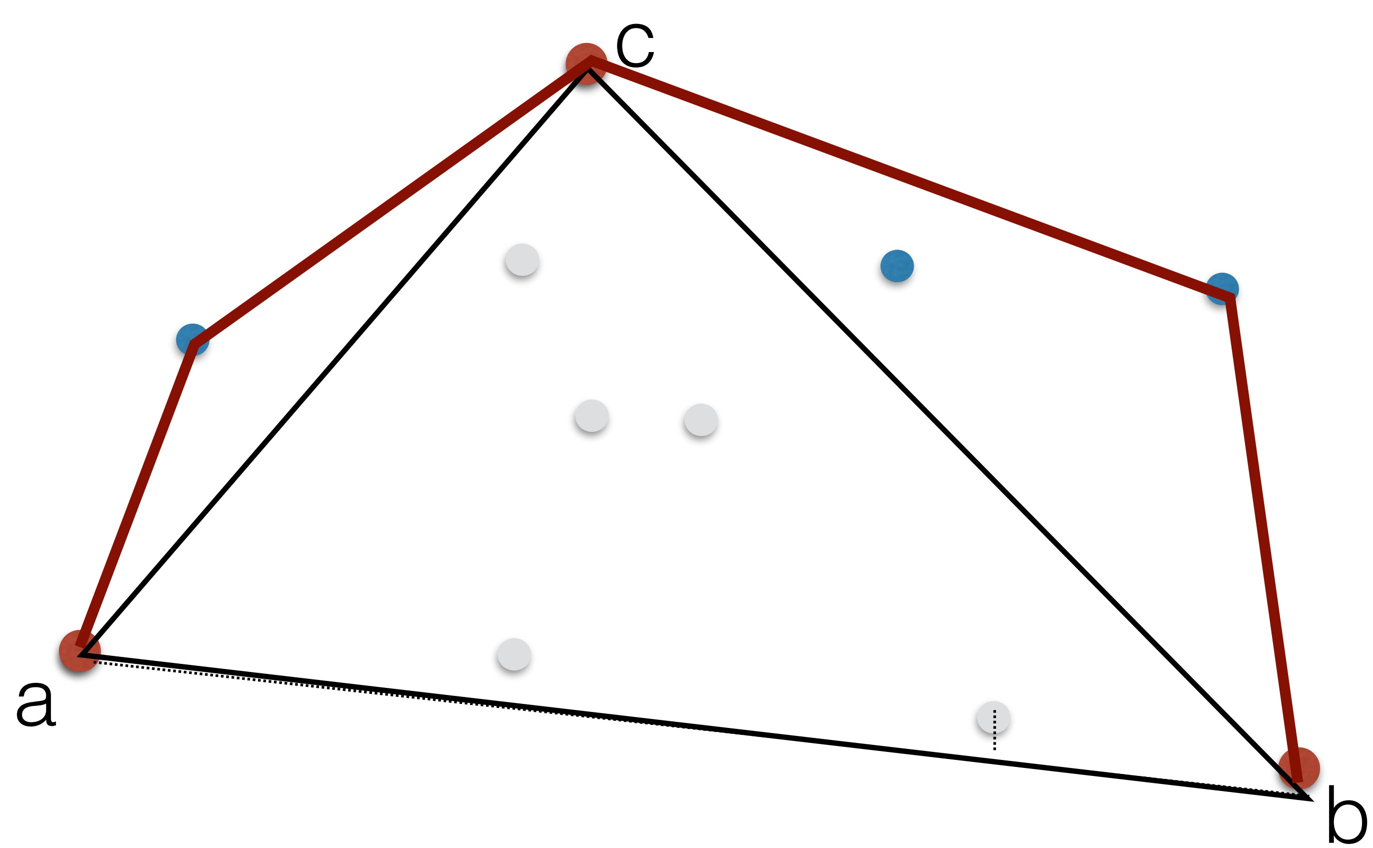
let's ignore collinear points for now

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



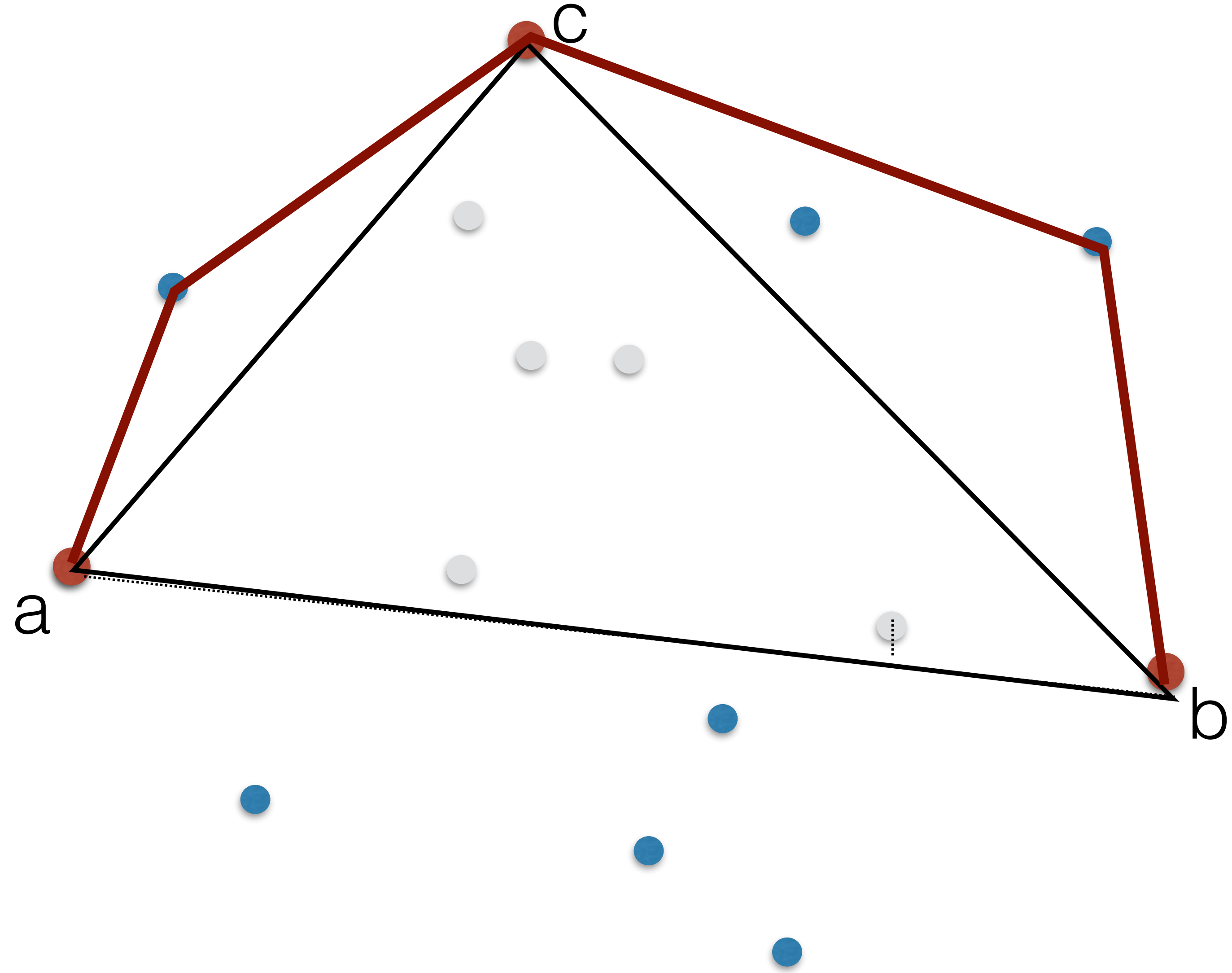
Quickhull (late 1970s)

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



Quickhull (late 1970s)

- Compute CH of P_2 similarly

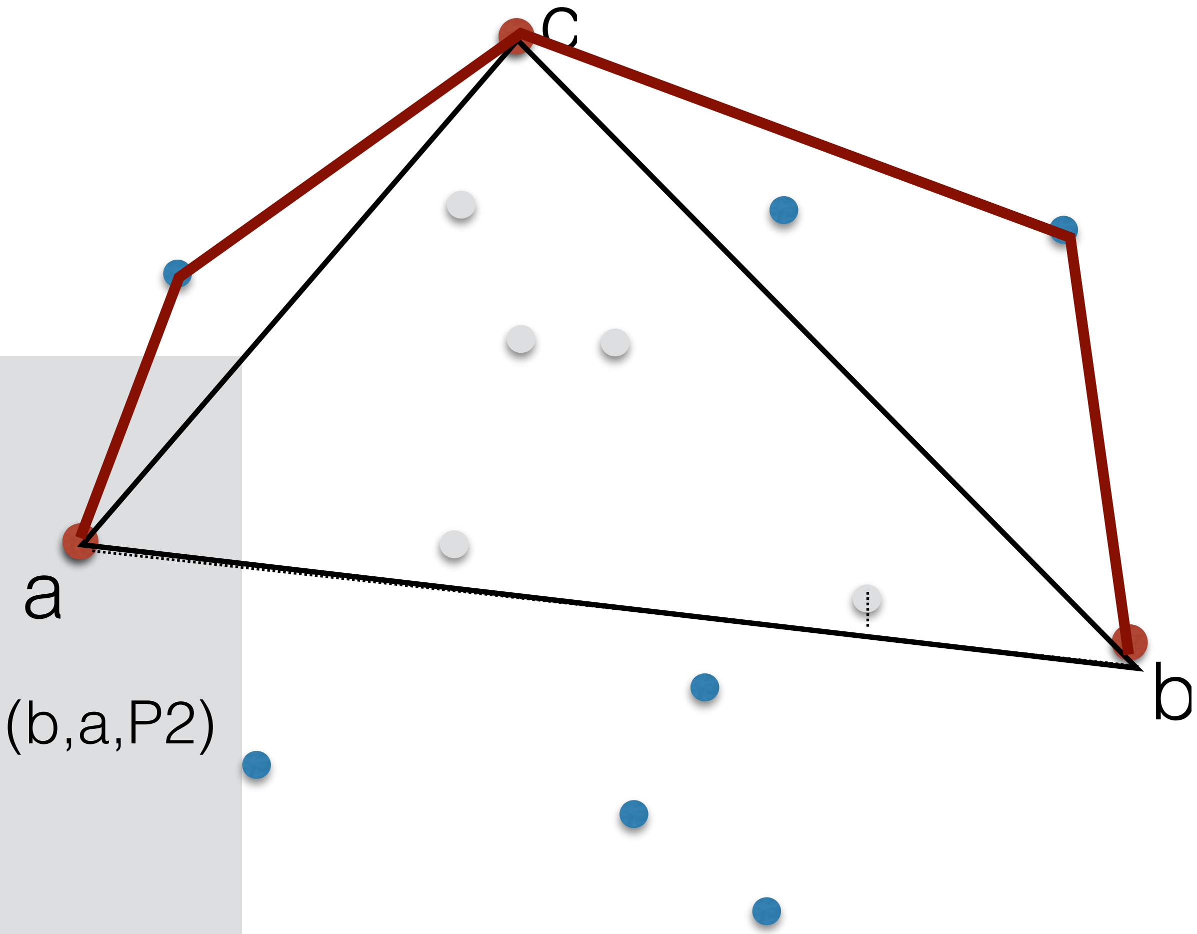


Quickhull (late 1970s)

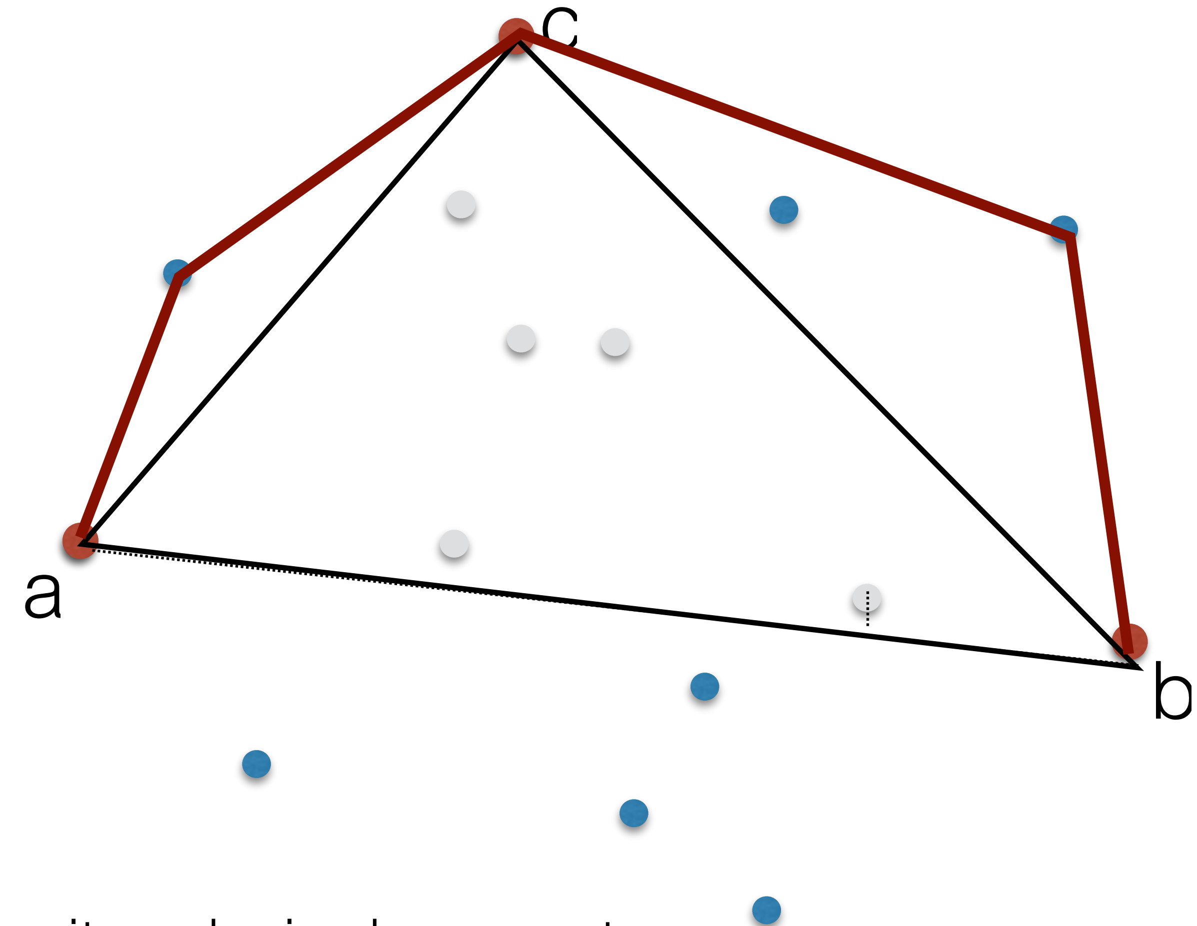
- **Quickhull (P)**
  - find a, b
  - partition P into P1, P2
  - return a + Quickhull(a,b, P1) + b + Quickhull(b,a,P2)

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

  //invariant: P is a set of points all on the left of ab
  - if P empty => return emptyset
  - 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)



Quickhull : Classwork



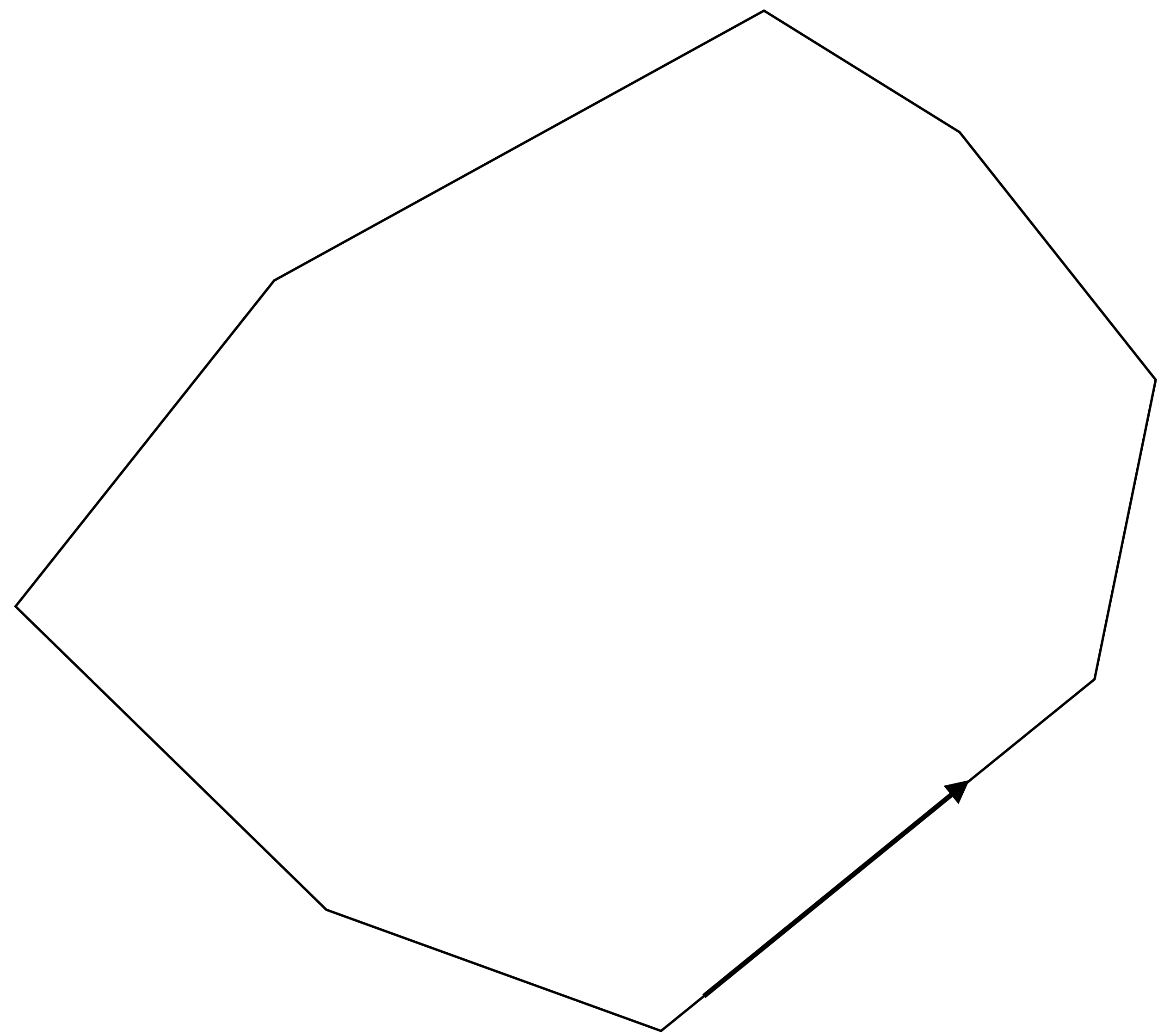
- Simulate Quickhull on a set of points and think how it works in degenerate cases
- Analysis:
  - Write a recurrence relation for its running time
  - What/when is the worst case running time ?
  - What/when is the best case running time ?
- Argue that Quickhull's average complexity is O(n) on points that are uniformly distributed.

Graham scan

Graham scan (late 1960s)

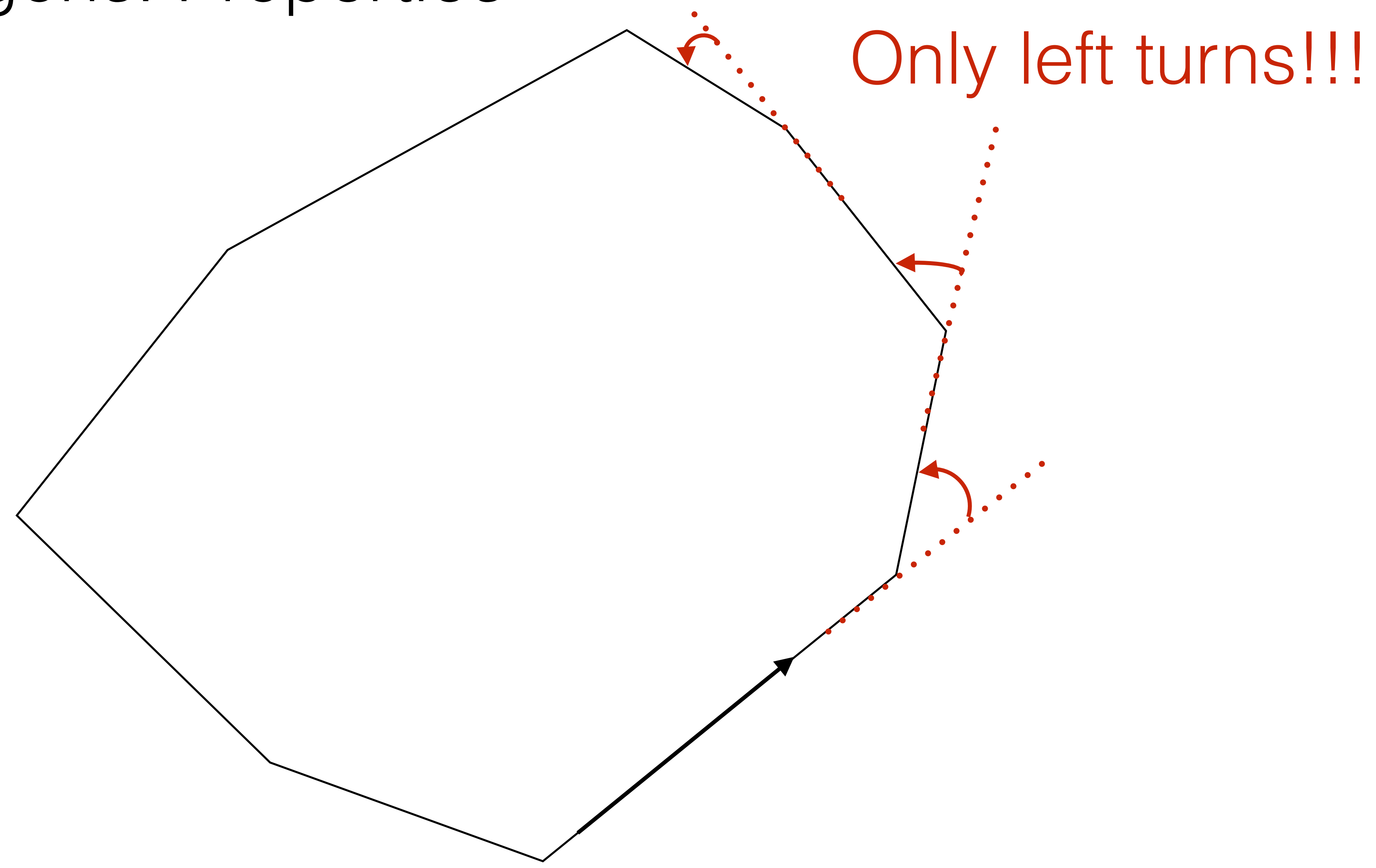
- In late 60s an application at Bell Labs required the hull of 10,000 points, for which a quadratic algorithm was too slow
- Graham developed an algorithm which runs in O(n lg n)
    - It runs in one sort plus a linear pass!!
    - Simple, intuitive, elegant and practical
    - You'll love it

Convex polygons: Properties



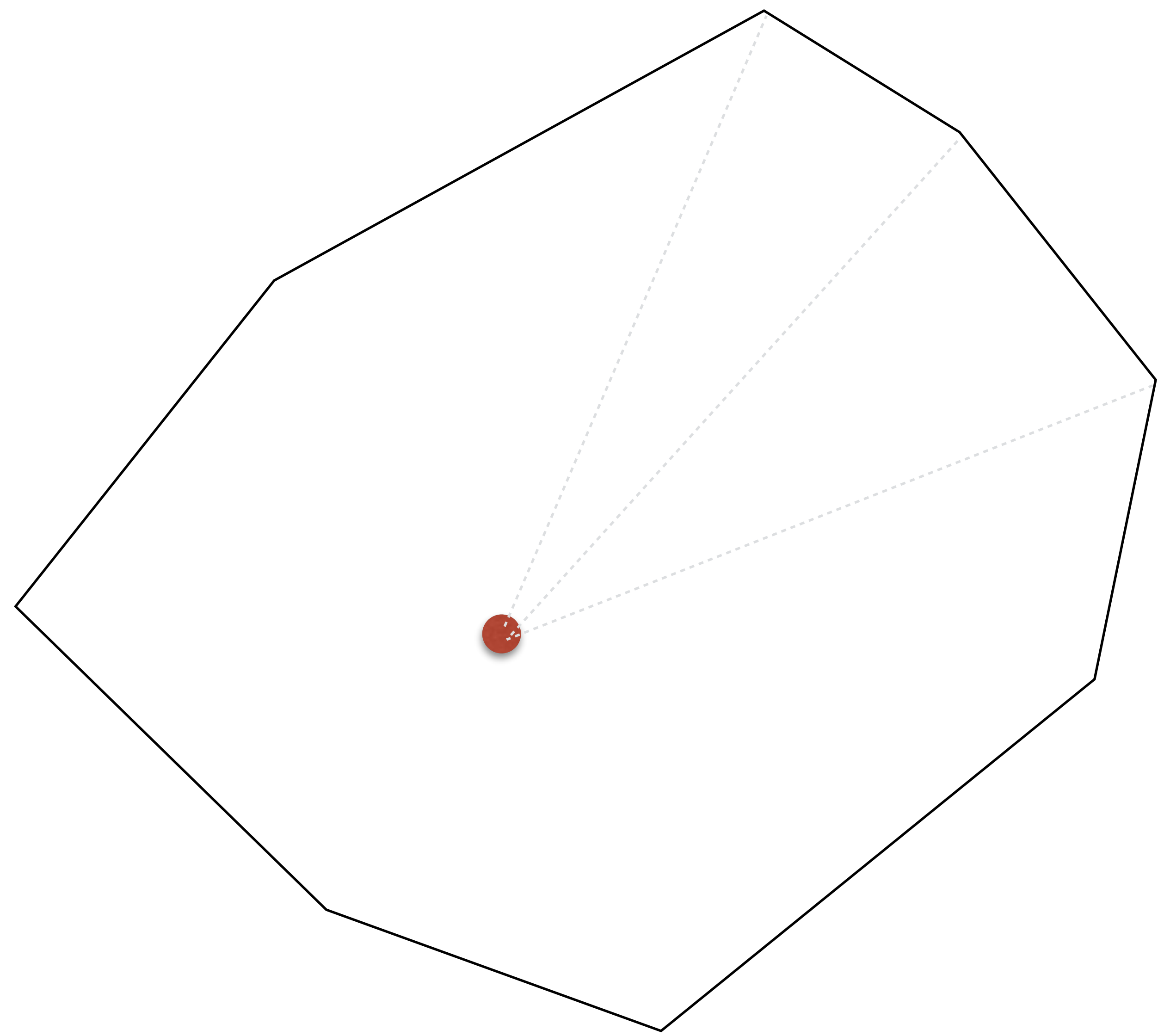
Walk ccw along the boundary of a convex polygon

Convex polygons: Properties



Walk ccw along the boundary of a convex polygon

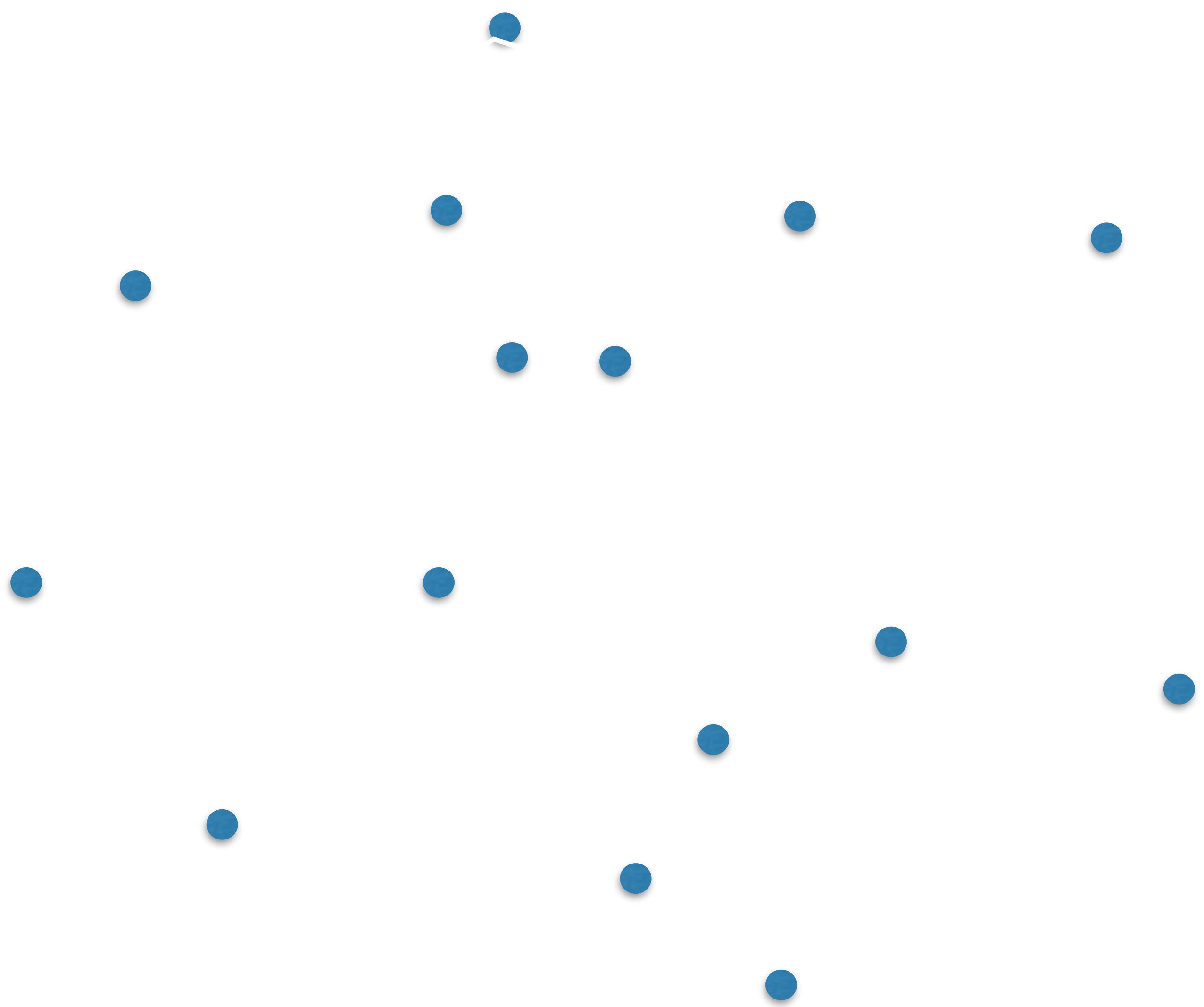
Convex polygons: Properties



Walk ccw along the boundary of a convex polygon

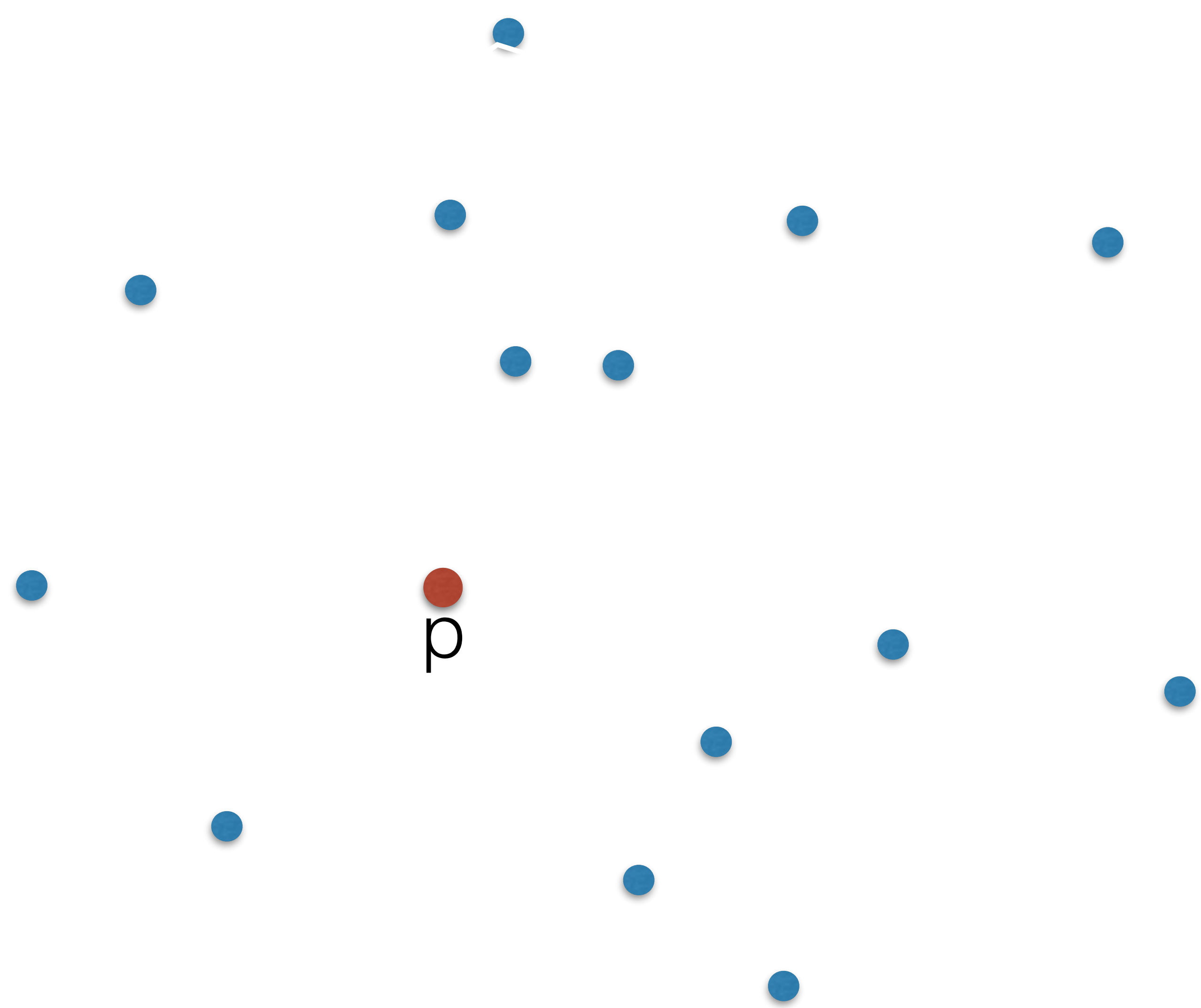
For any point p inside, the points on the boundary are in radial order around p

Graham scan (late 1960s)



Graham scan (late 1960s)

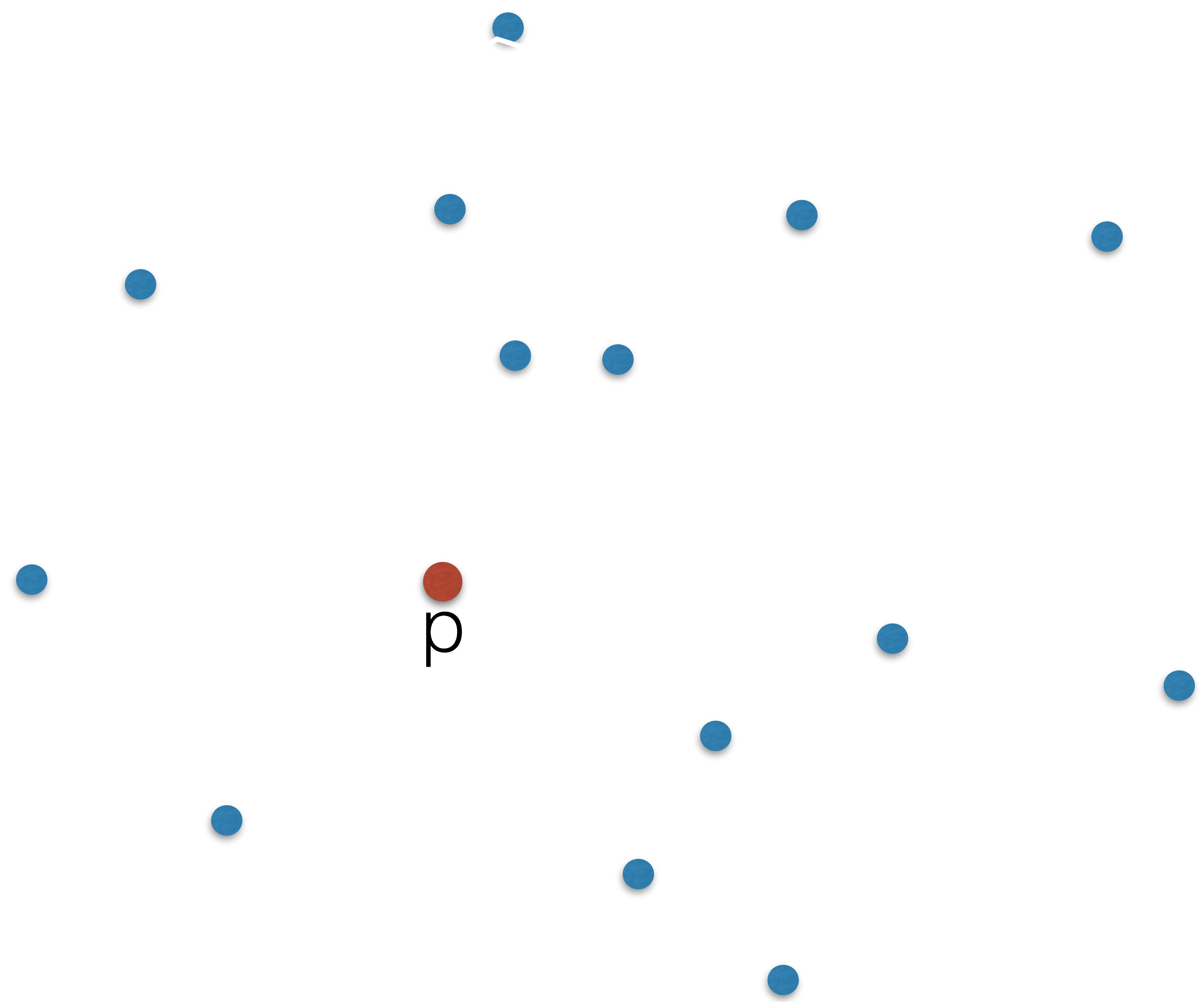
- Idea: start from a point p interior to the hull <——— we'll think about how to get it later



Graham scan (late 1960s)

- Idea: start from a point p interior to the hull <——— we'll think about how to get it later

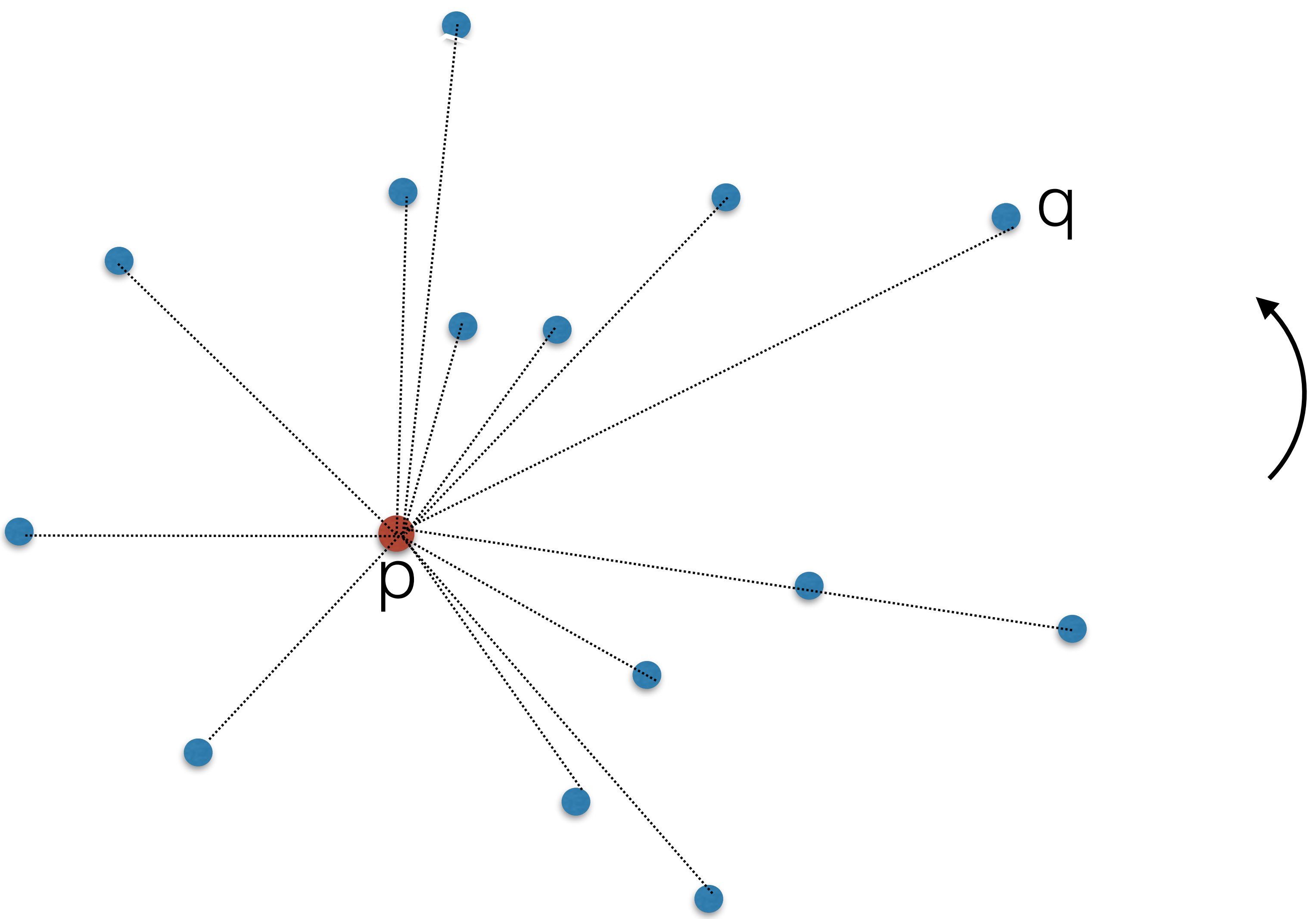
  order all points by their ccw angle wrt p



Graham scan (late 1960s)

- Idea: start from a point p interior to the hull

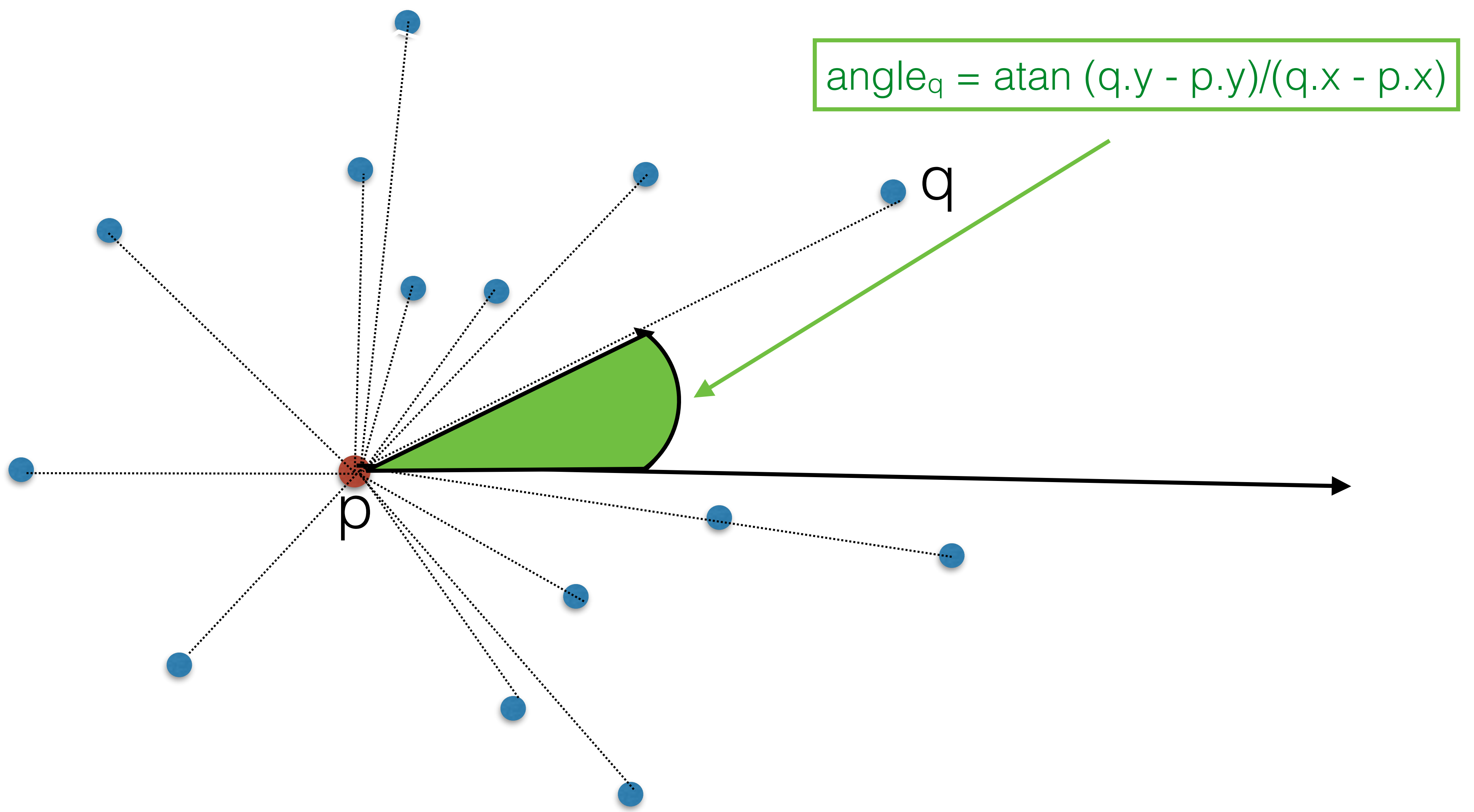
  order all points by their ccw angle wrt p



Graham scan (late 1960s)

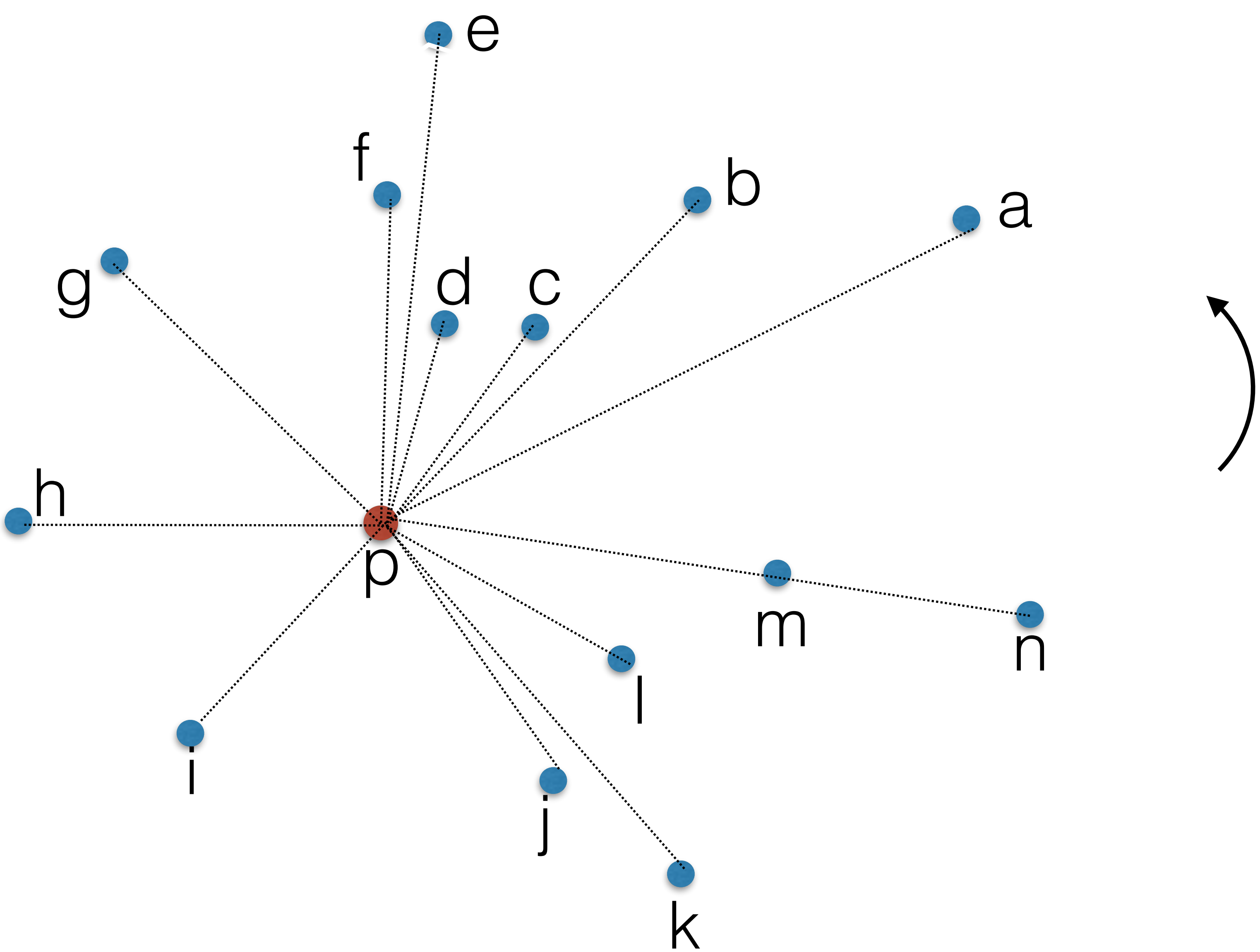
- Idea: start from a point p interior to the hull

  order all points by their ccw angle wrt p



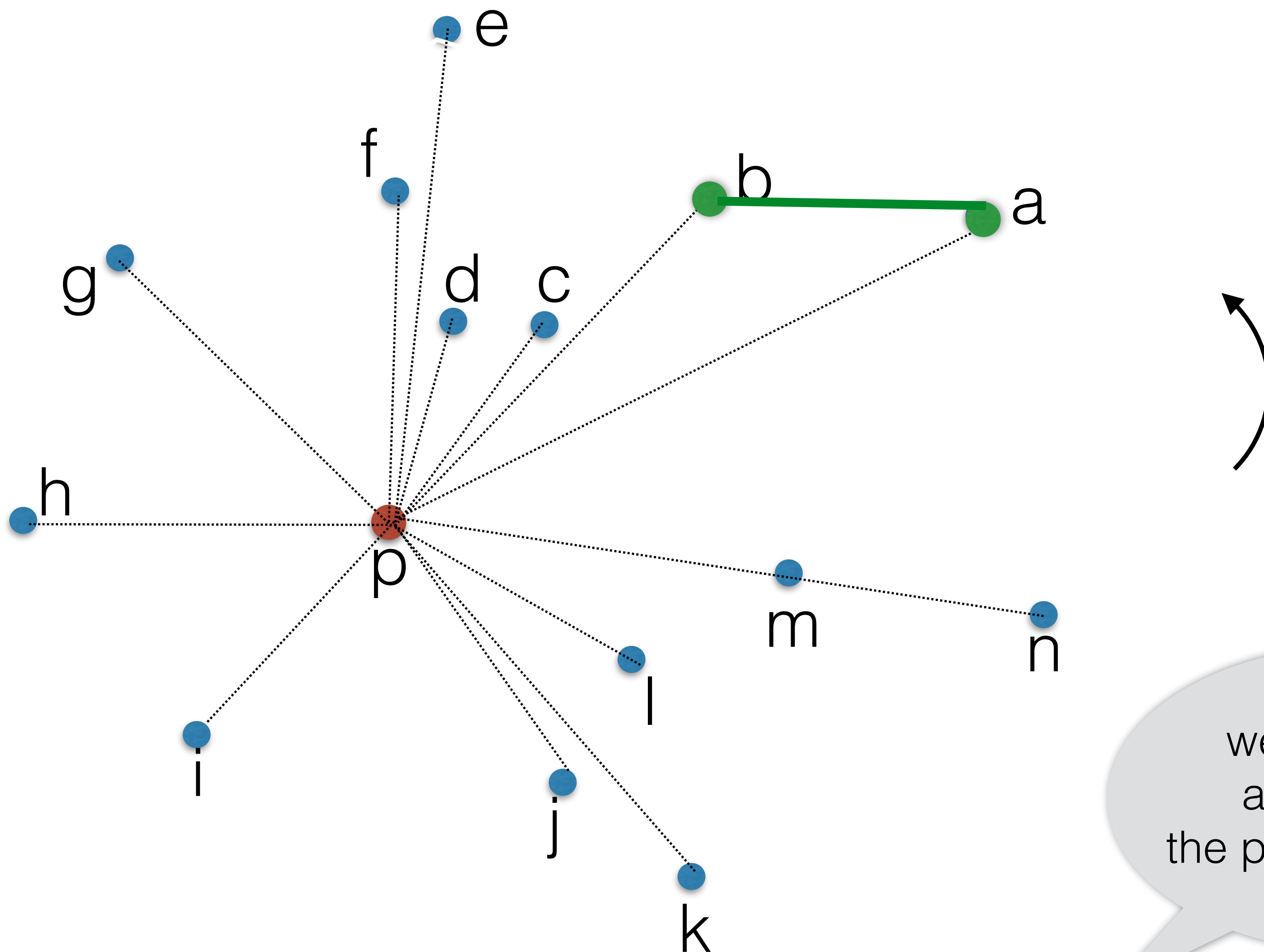
Graham scan (late 1960s)

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



Graham scan (late 1960s)

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

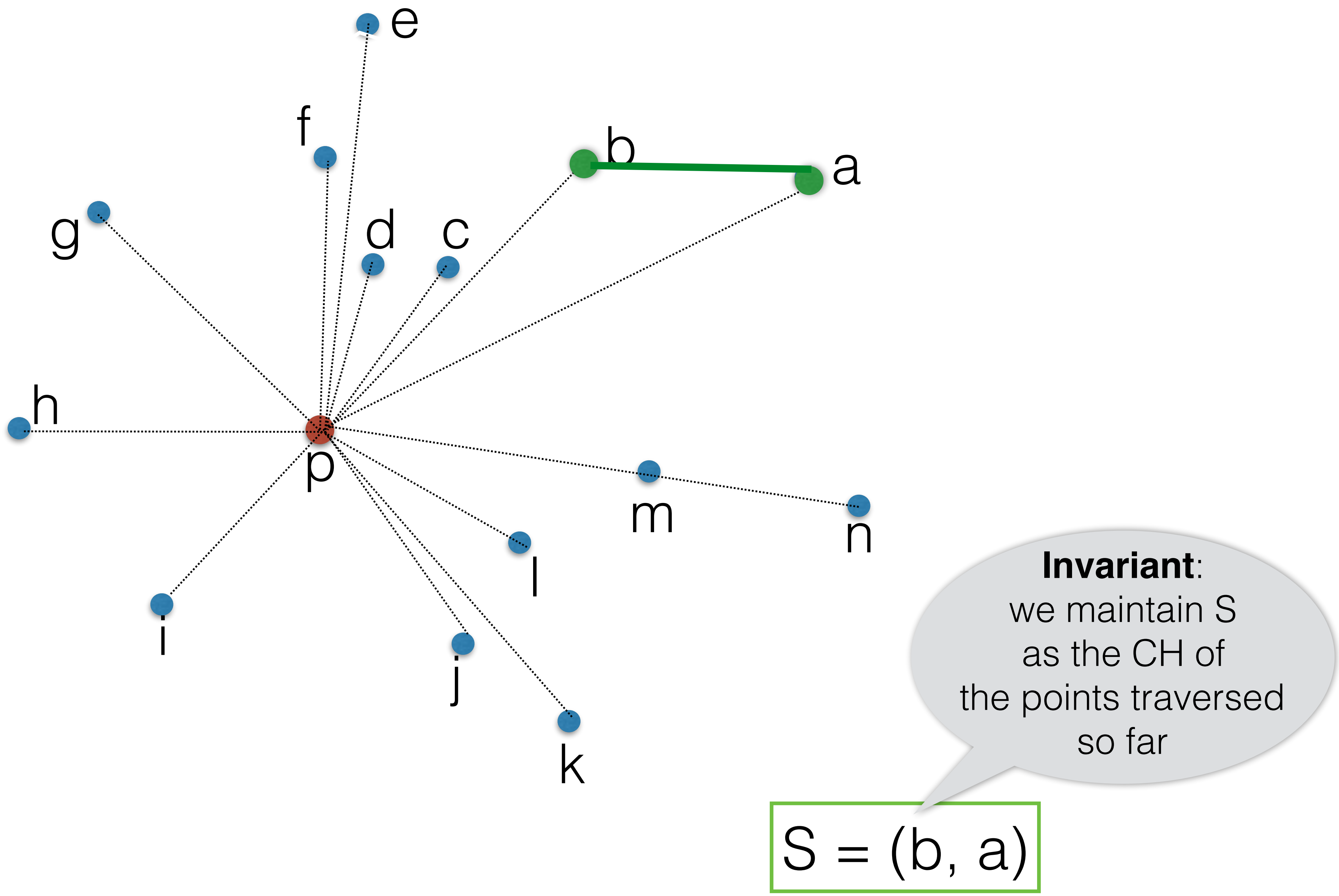


Invariant:
we maintain S
as the CH of
the points traversed
so far

S = (b, a)

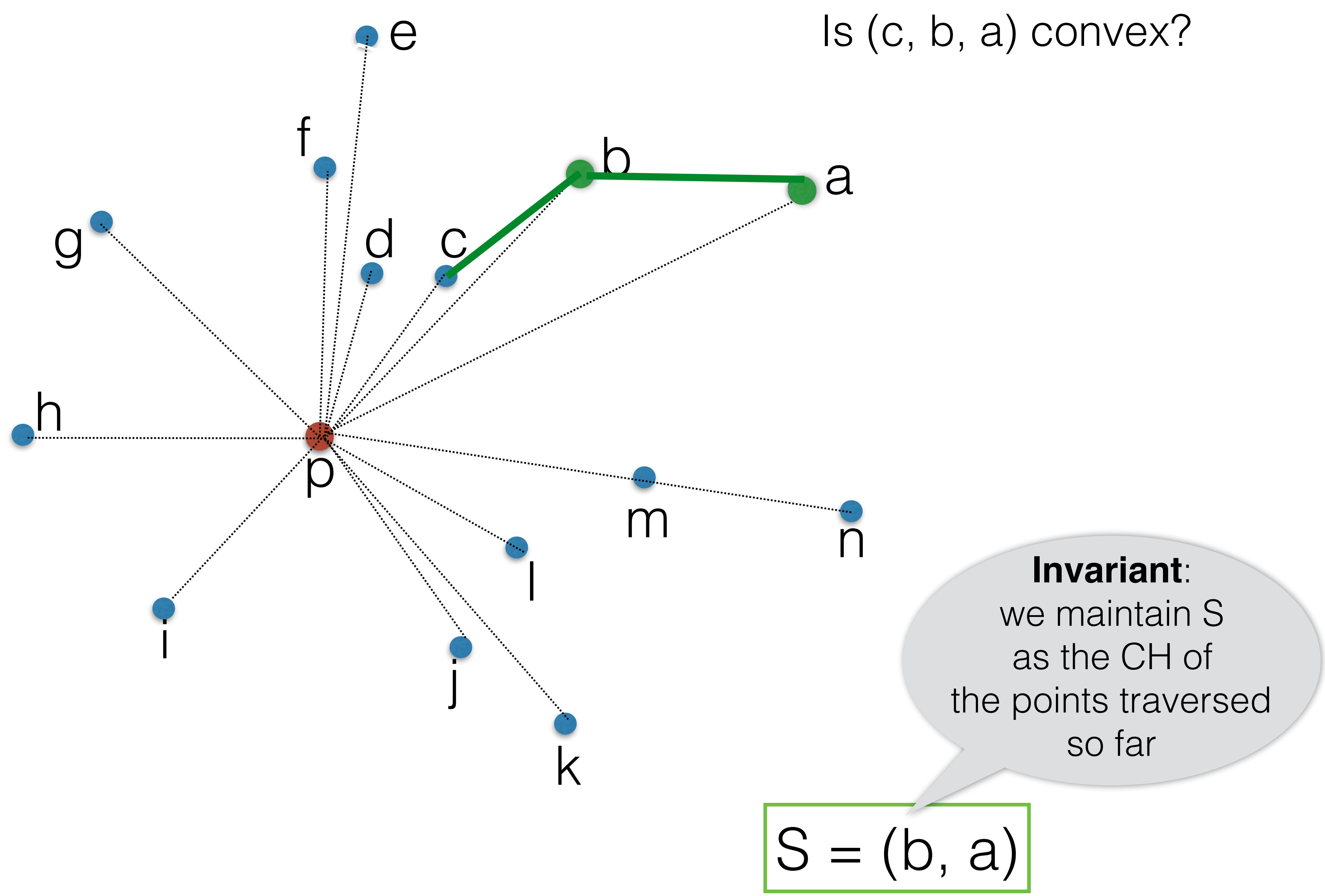
Graham scan (late 1960s)

Now we read point c: what do we do with it?



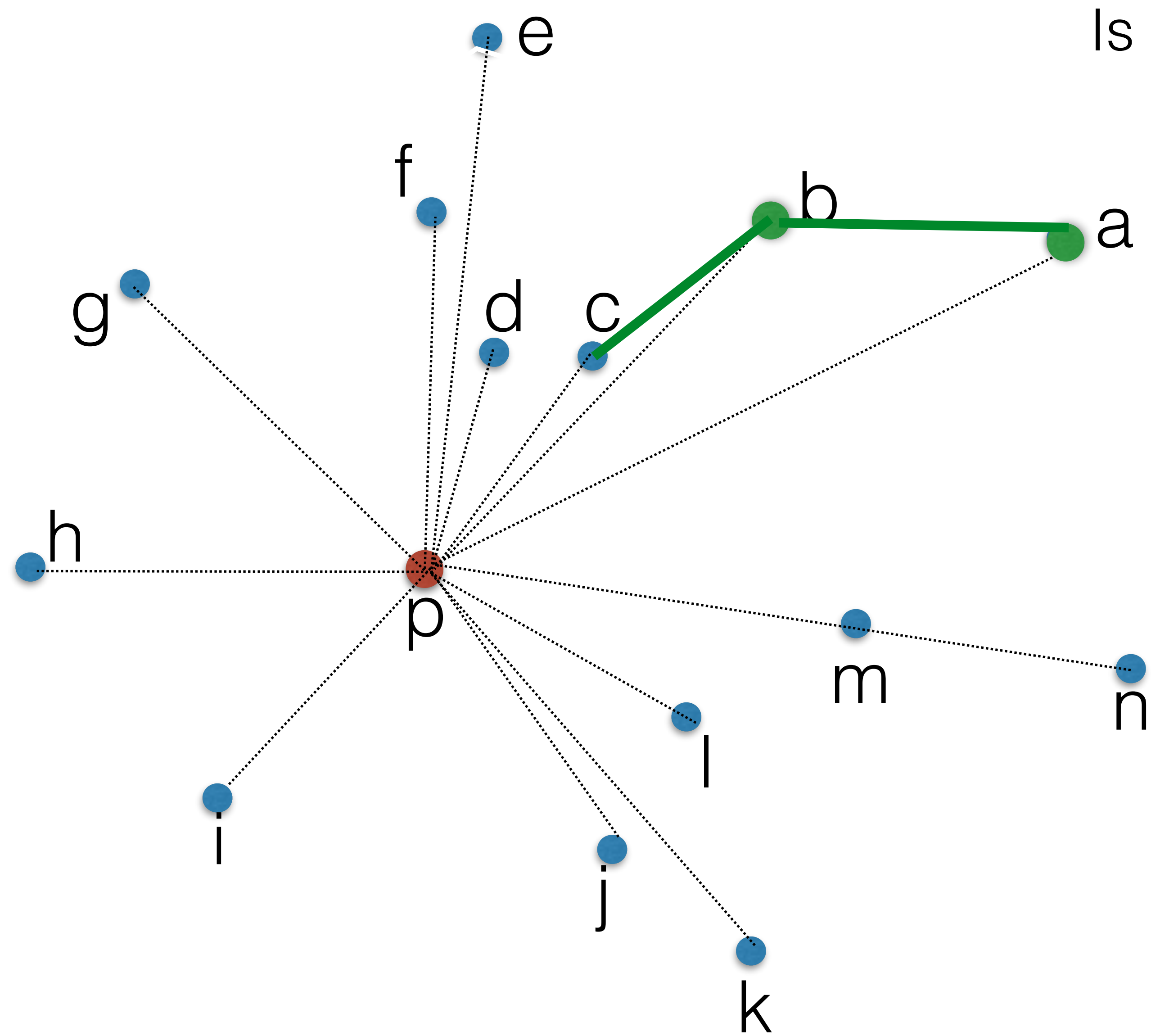
Graham scan (late 1960s)

Now we read point c: what do we do with it?



Graham scan (late 1960s)

Now we read point c: what do we do with it?



Is (c, b, a) convex?

YES!

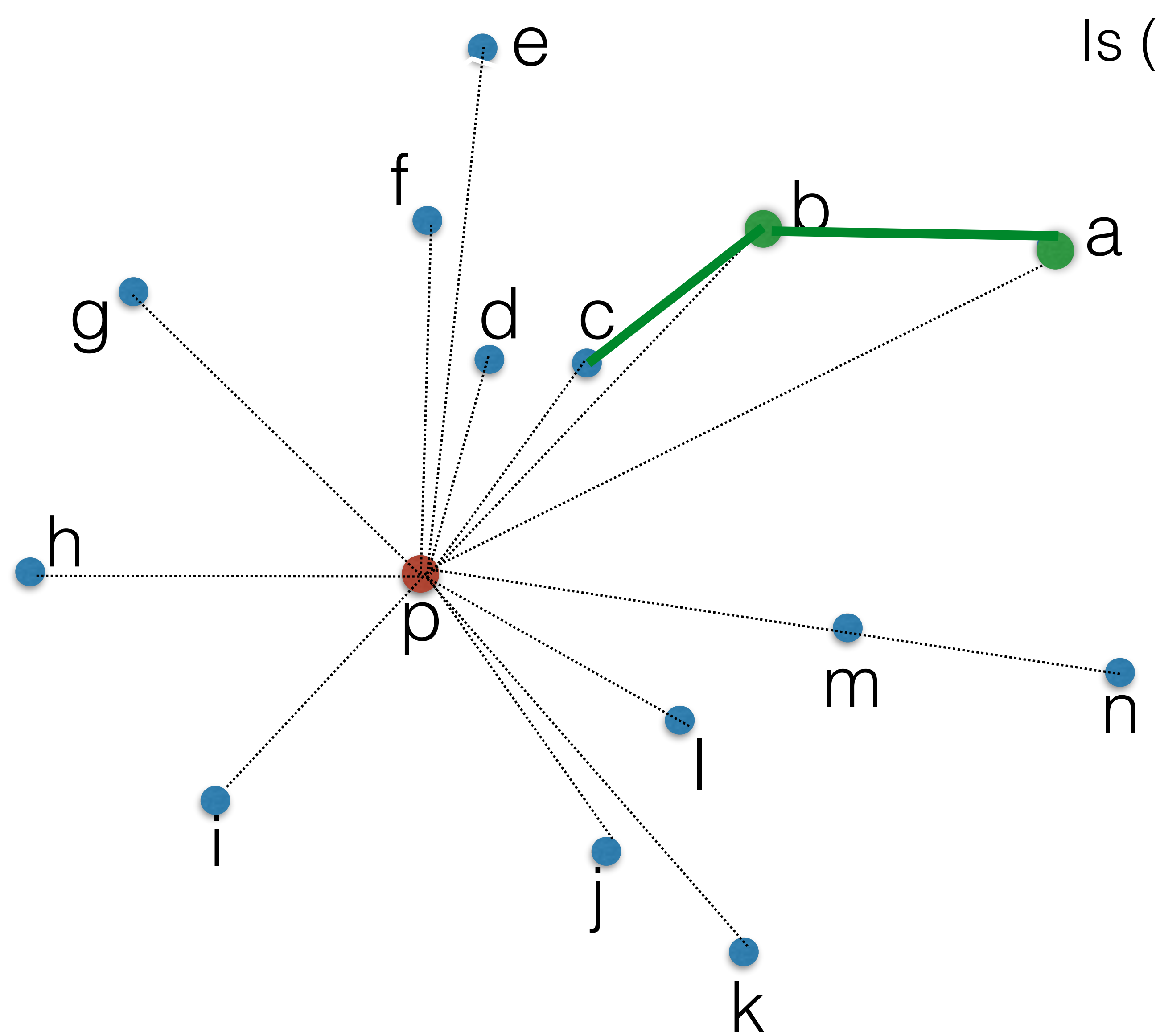
Invariant:
we maintain S
as the CH of
the points traversed
so far

S = (b, a)

Graham scan

is c left of ab

Now we read point c: if $\boxed{\text{(c+S) stays convex}}$: add c to S



Is (c, b, a) convex?

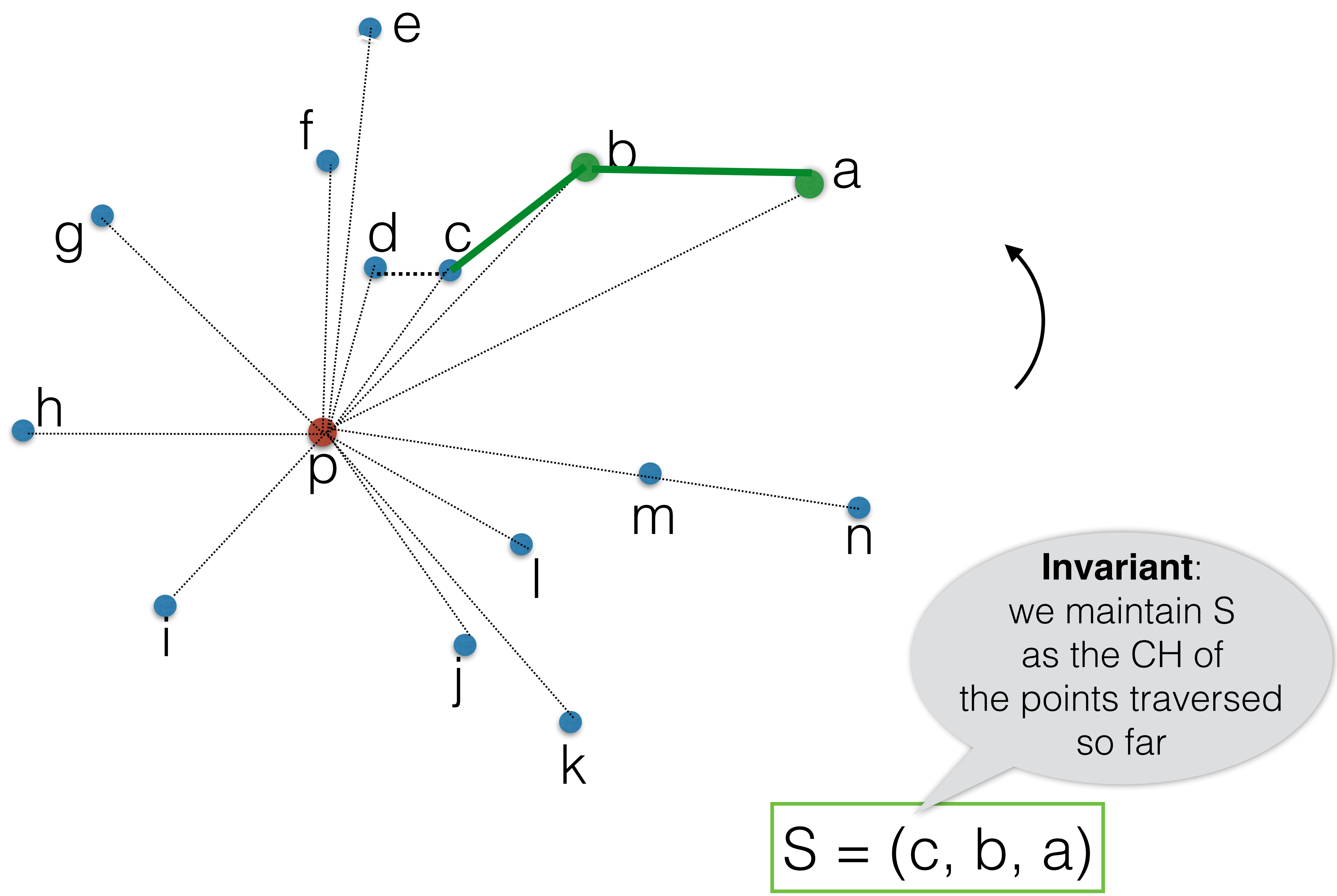
YES!

Invariant:
we maintain S
as the CH of
the points traversed
so far

$\boxed{S = (c, b, a)}$

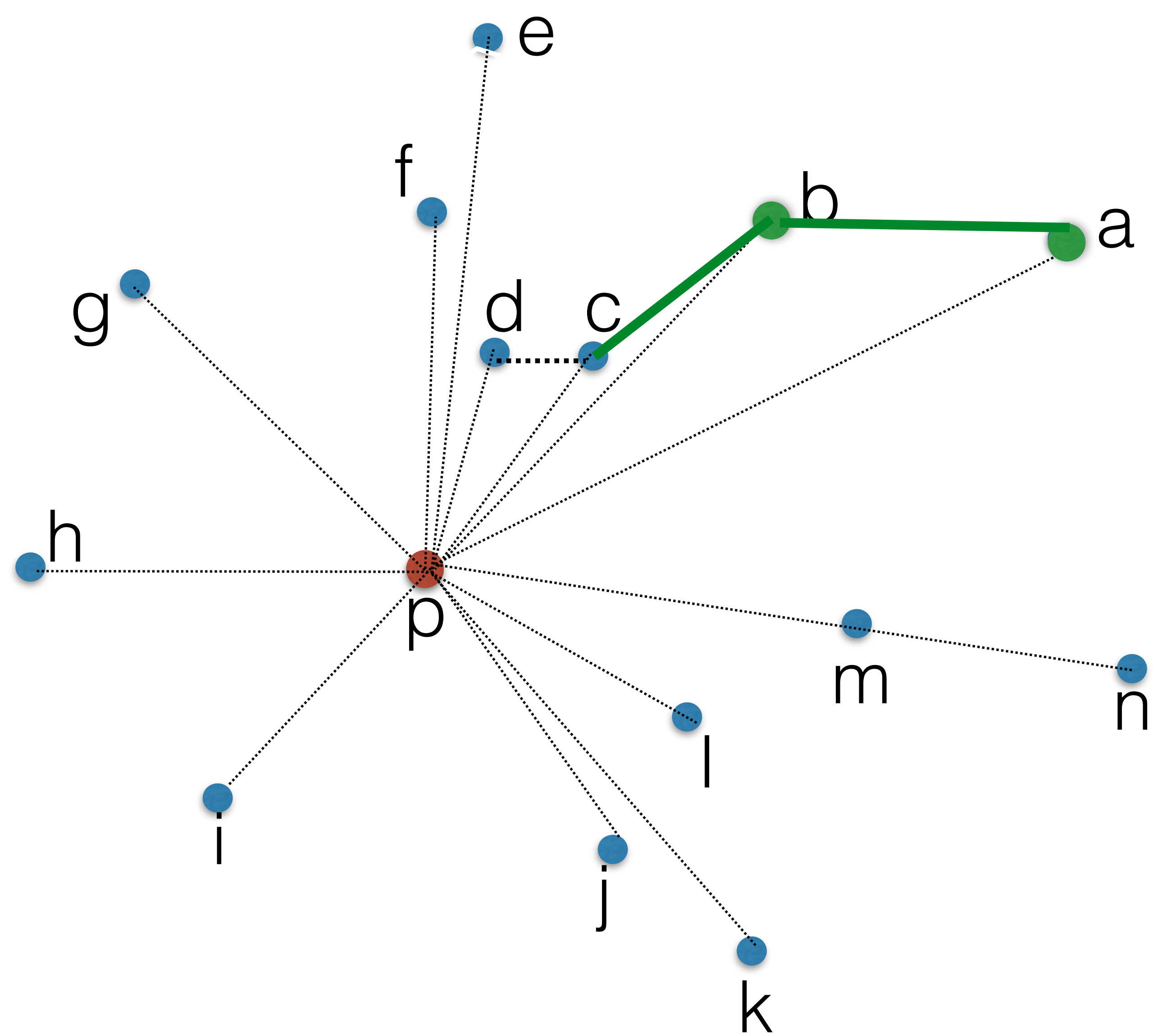
Graham scan (late 1960s)

Now we read point d:



Graham scan (late 1960s)

Now we read point d: is d left of bc? NO



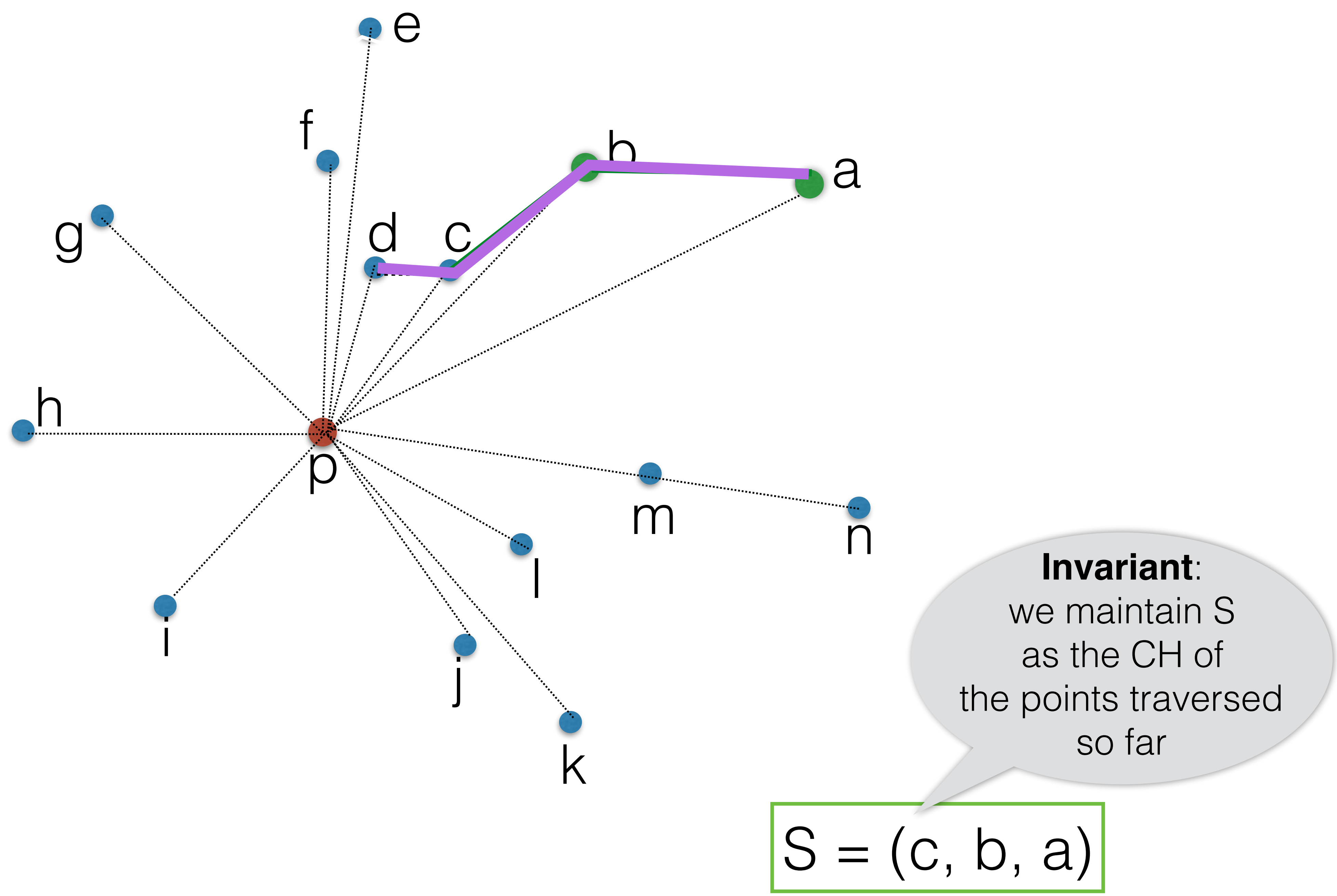
Invariant:
we maintain S
as the CH of
the points traversed
so far

S = (c, b, a)

Graham scan (late 1960s)

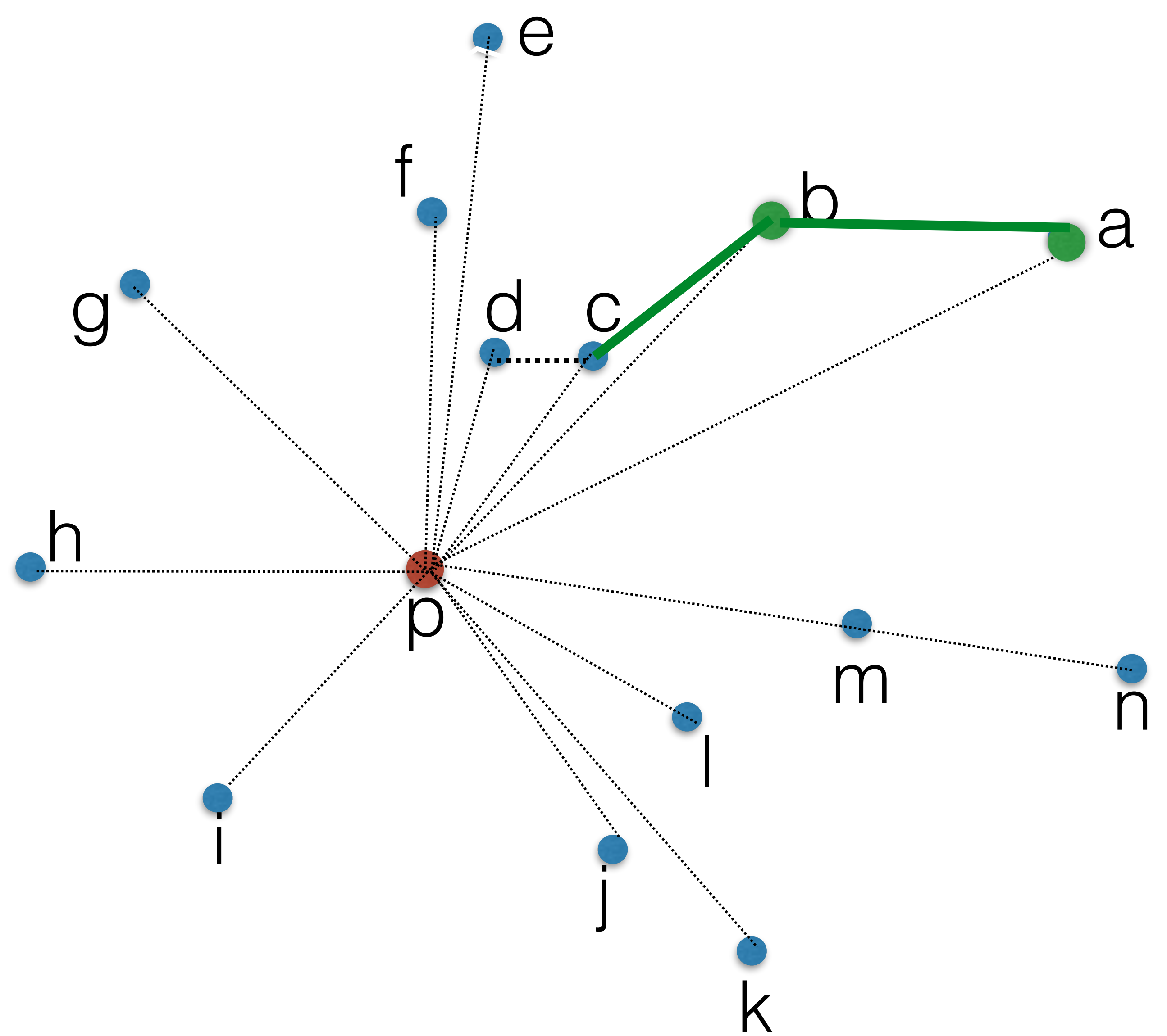
Now we read point d: is d left of bc? NO

//can’t add d, because (d,c,b,a) not convex



Graham scan (late 1960s)

Now we read point d: is d left of bc? NO



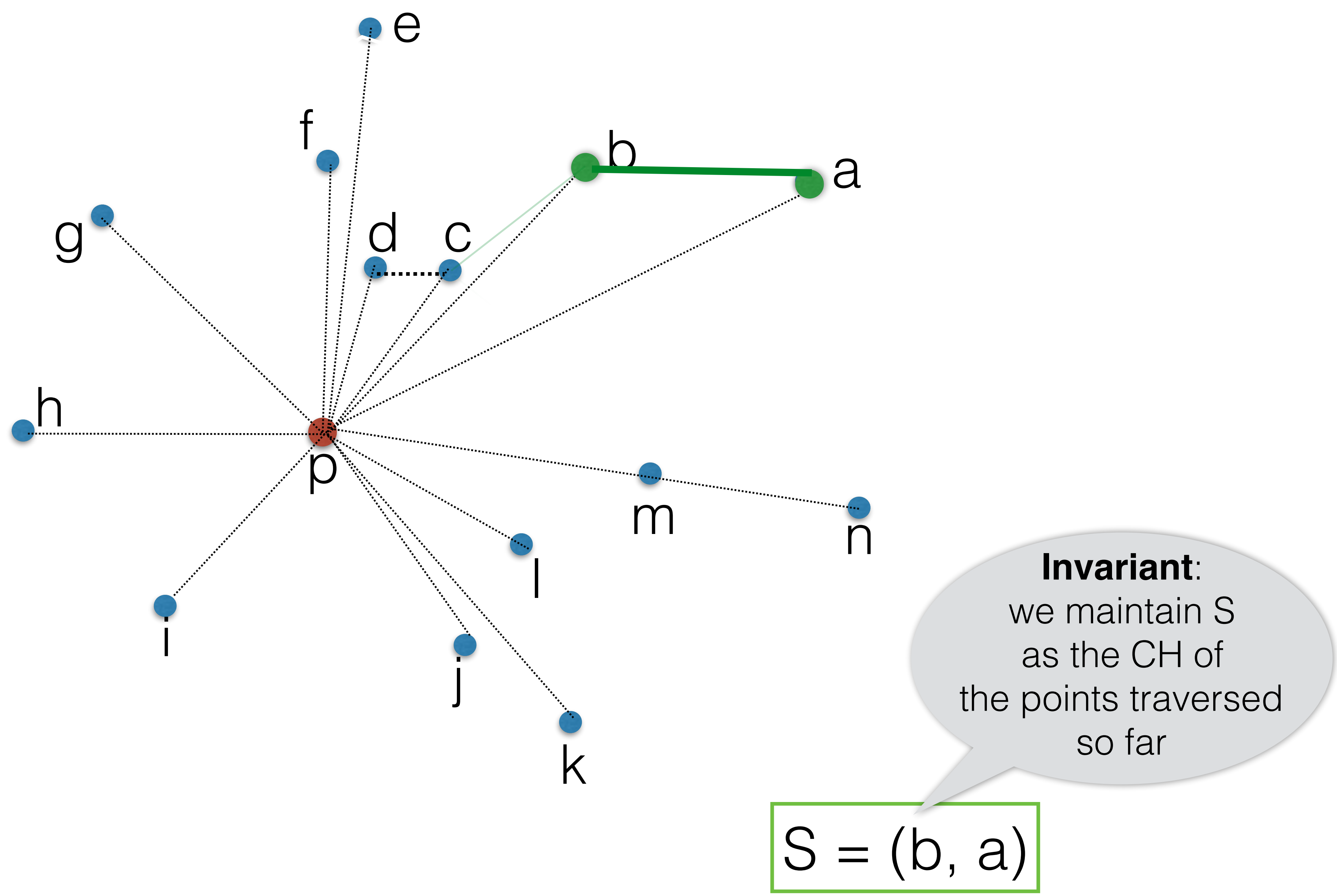
Invariant:
we maintain S
as the CH of
the points traversed
so far

S = (c, b, a)

Graham scan (late 1960s)

Now we read point d: is d left of bc? NO

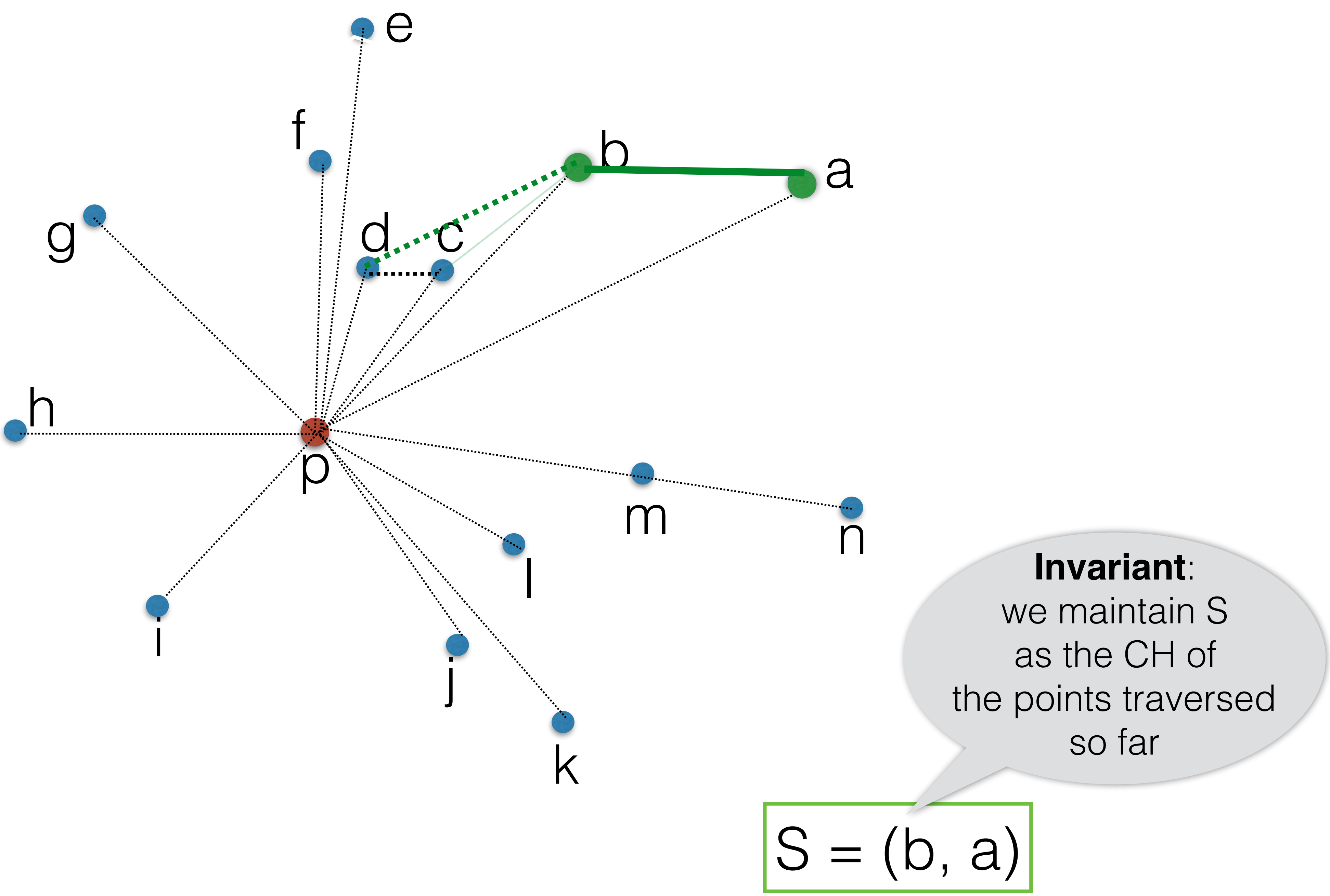
pop c; is d left of ab?



Graham scan (late 1960s)

Now we read point d: is d left of bc? NO

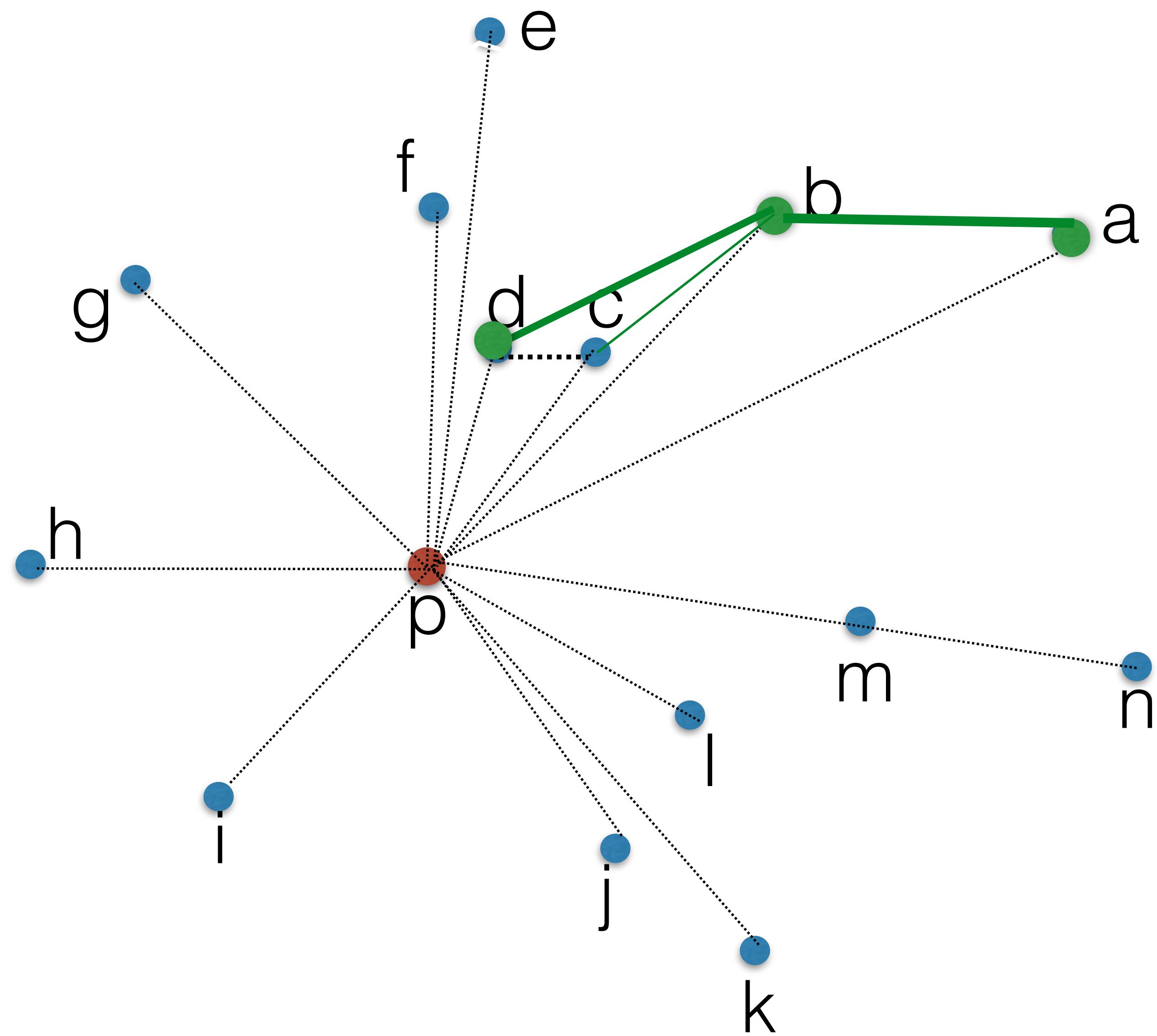
pop c; is d left of ab?



Graham scan (late 1960s)

Now we read point d: is d left of bc? NO

pop c; is d left of ab? YES ==> insert d in S



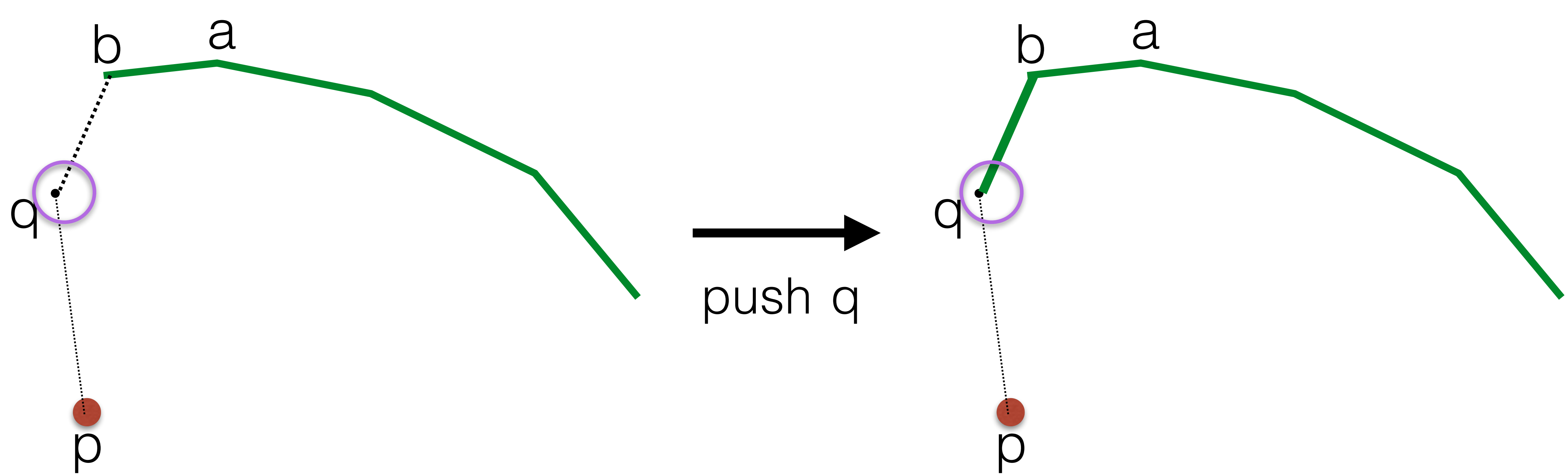
Invariant:
we maintain S
as the CH of
the points traversed
so far

S = (d, b, a)

Graham scan (late 1960s)

In general, we read next point q:

- let b = head(S), a = next(b)
- if q is left of ab: add q to S



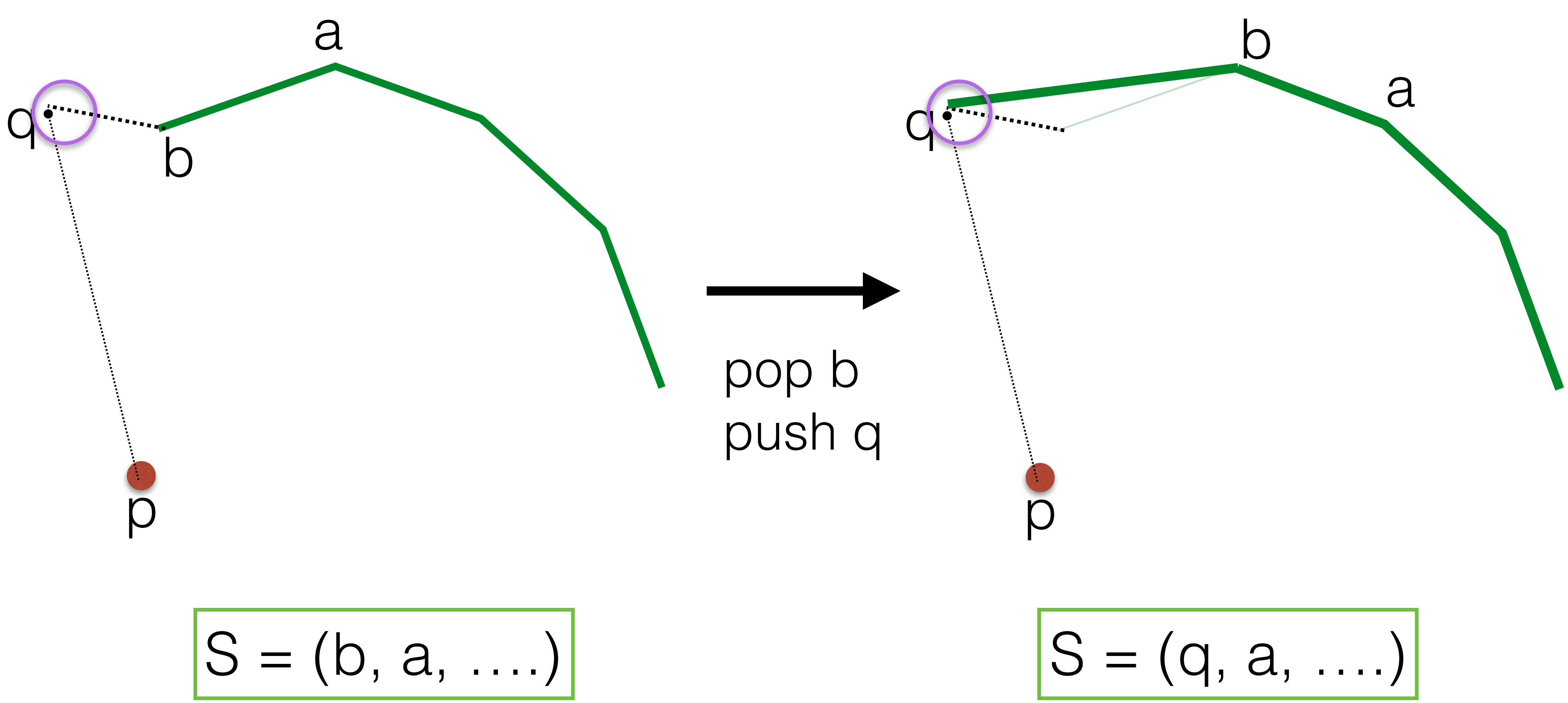
S = (b, a, ...)

S = (q, b, a, ...)

Graham scan (late 1960s)

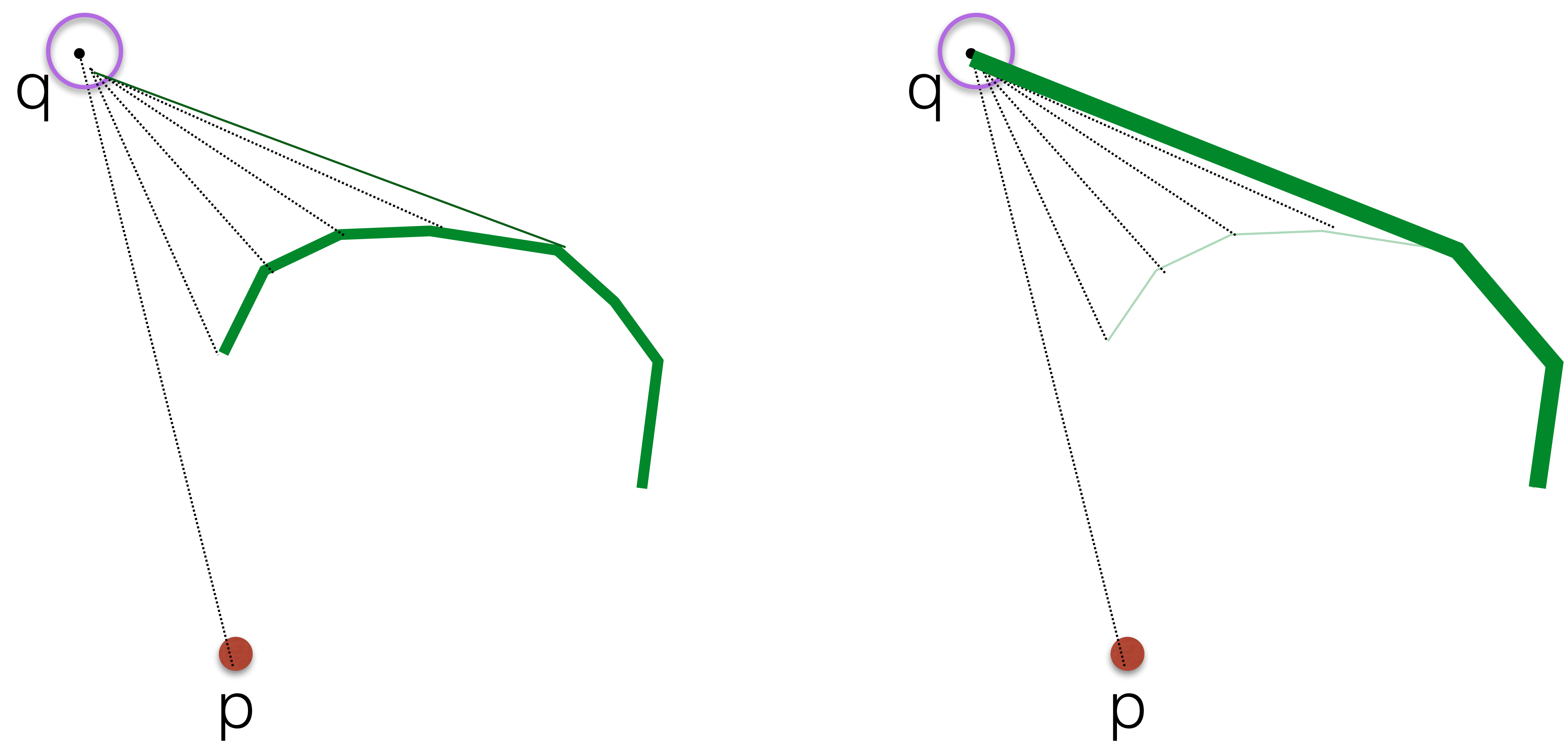
In general, we read next point q:

- let b = head(S), a = next(b)
- if q is right of ab: pop b; repeat until q is left of ab, then add q to S



Graham scan (late 1960s)

Cascading pops



Graham scan (late 1960s)

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

Graham scan (late 1960s)

- Choose a set of “interesting” points and go through the algorithm
- Does the algorithm handle degenerate cases? If not, how do you fix it?
- How to find an interior point?
- Analysis: How long does it take?

Graham scan: ANALYSIS

- Find interior point p_0
- Sort all other points ccw around p_0.....
- Initialize stack S = (p_2, p_1)
- for i=3 to n-1 do
  - if p_i is left of (second(S),first(S)):
    - push p_i on S
  - else
    - do
      - pop S
    - while p_i is right of (second(S), first(S))
    - push p_i on S

Graham scan: ANALYSIS

- Find interior point p_0
- Sort all other points ccw around p_0…..
- Initialize stack S = (p_2, p_1)
- for i=3 to n-1 do
  - if p_i is left of (second(S),first(S)):
    - push p_i on S
  - else
    - do
      - pop S
    - while p_i is right of (second(S), first(S))
    - push p_i on S

O(n) (we'll think of it later)

Graham scan: ANALYSIS

- Find interior point p_0 ← O(n) (we'll think of it later)
- Sort all other points ccw around p_0….. ← O(n lg n)
- Initialize stack S = (p_2, p_1)
- for i=3 to n-1 do
  - if p_i is left of (second(S),first(S)):
    - push p_i on S
  - else
    - do
      - pop S
    - while p_i is right of (second(S), first(S))
    - push p_i on S

Graham scan: ANALYSIS

- Find interior point p_0
- Sort all other points ccw around p_0.....
- Initialize stack S = (p_2, p_1)
- for i=3 to n-1 do
  - if p_i is left of (second(S),first(S)):
    - push p_i on S
  - else
    - do
      - pop S
    - while p_i is right of (second(S), first(S))
    - push p_i on S

O(n) (we'll think of it later)

O(n lg n)

How long does this take?

Graham scan: ANALYSIS

- Find interior point p_0
- Sort all other points ccw around p_0.....
- Initialize stack S = (p_2, p_1)
- for i=3 to n-1 do
  - if p_i is left of (second(S),first(S)):
    - push p_i on S
  - else
    - do
      - pop S
    - while p_i is right of (second(S), first(S))
    - push p_i on S

O(n) (we'll think of it later)

O(n lg n)

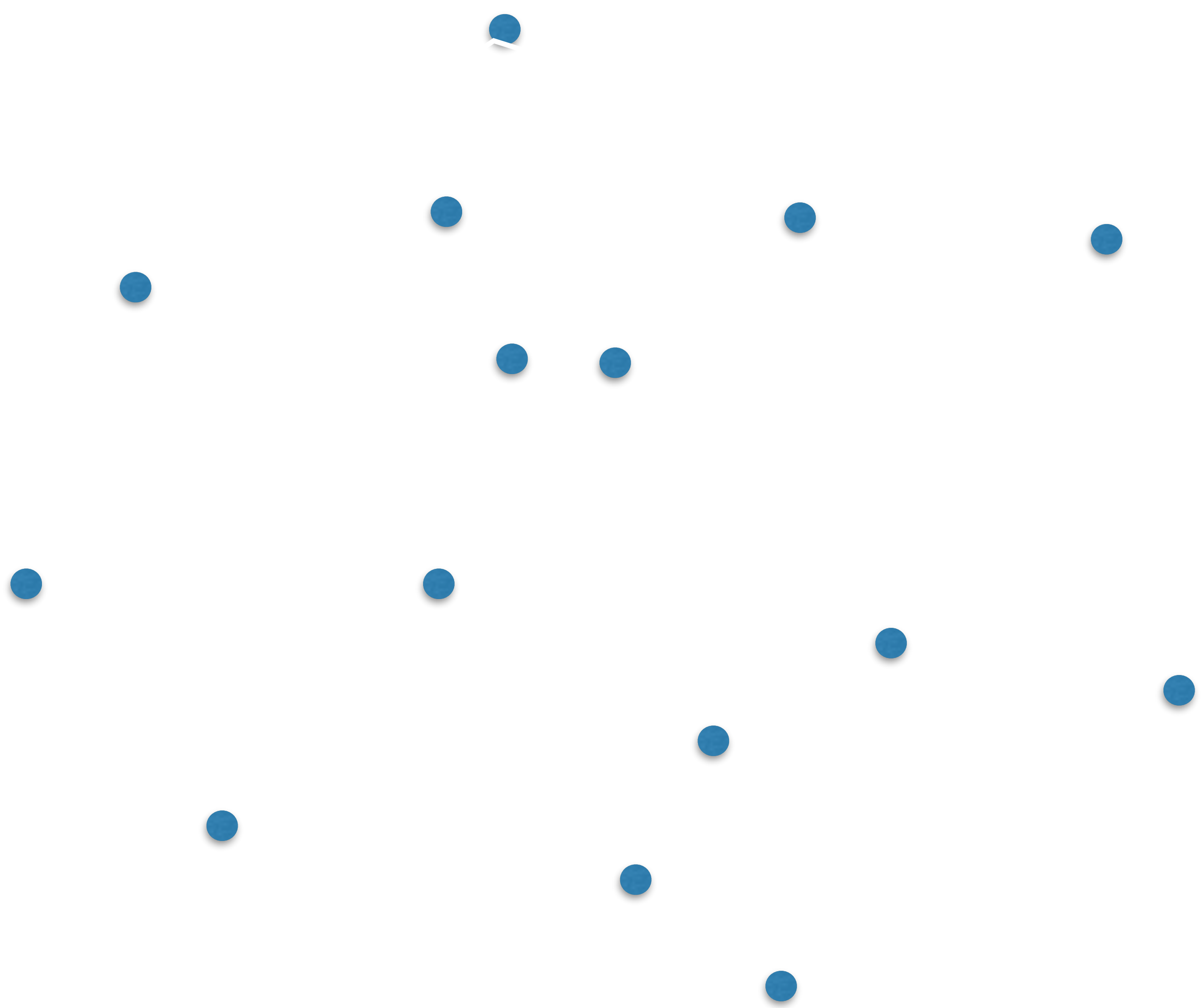
How long does this take?

O(n)

every point is pushed once and popped at most once

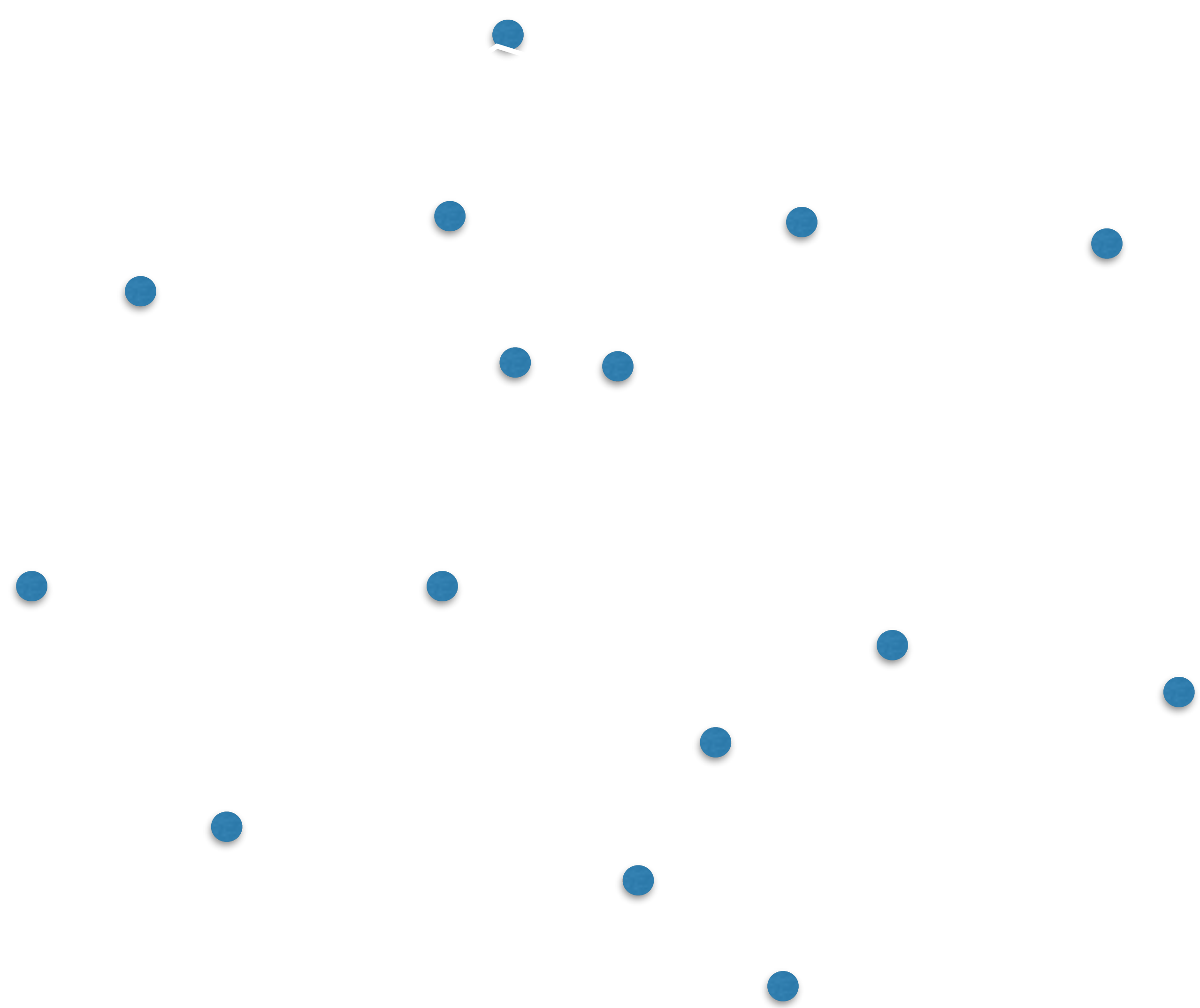
Graham scan: Details

- How to find an interior point?



Graham scan: Details

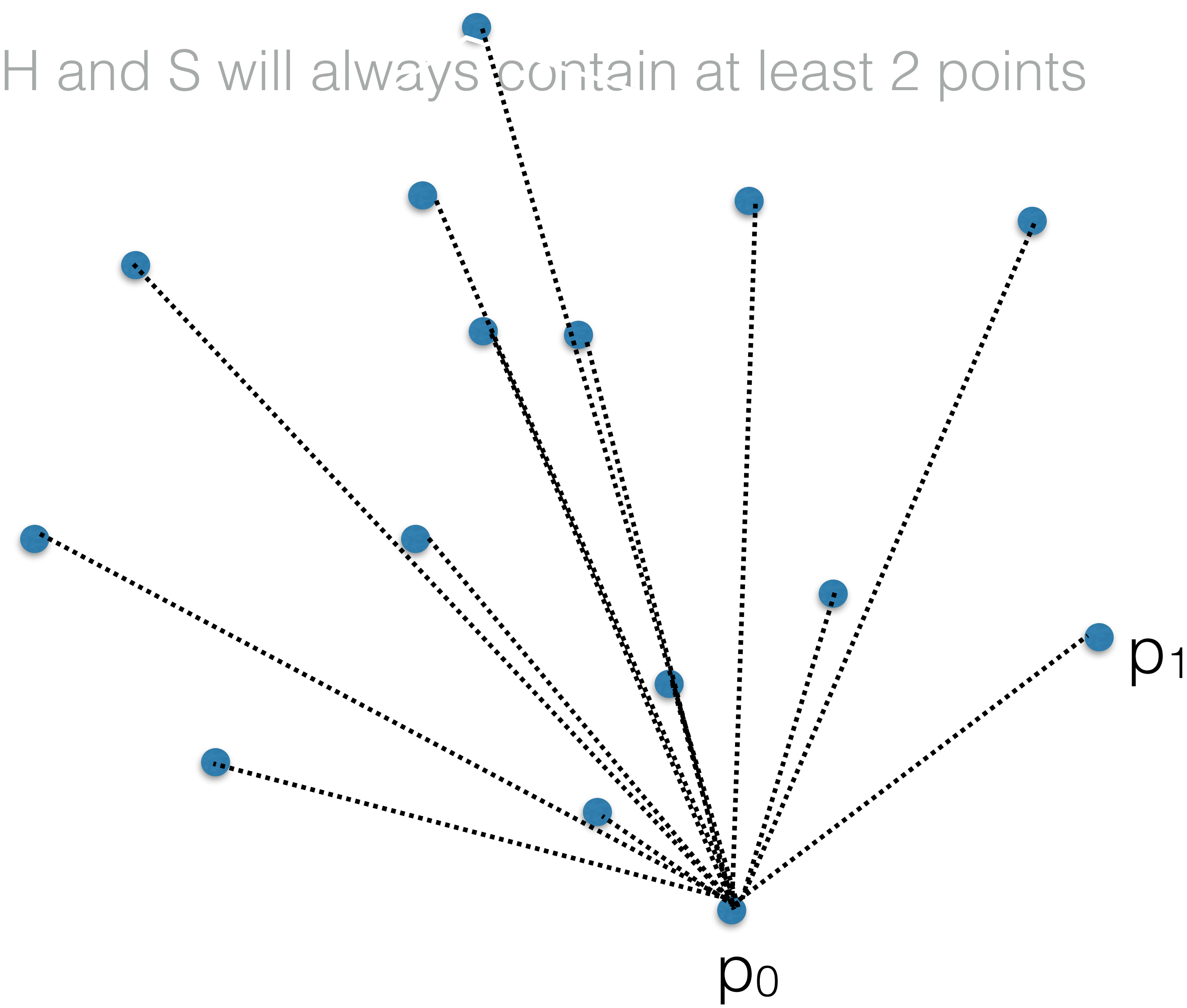
- How to find an interior point?
- A simplification is to pick p_0 as the lowest point



Graham scan: Details

- How to find an interior point?
- A simplification is to pick p_0 as the lowest point
  - initialize stack S = (p1, p0)

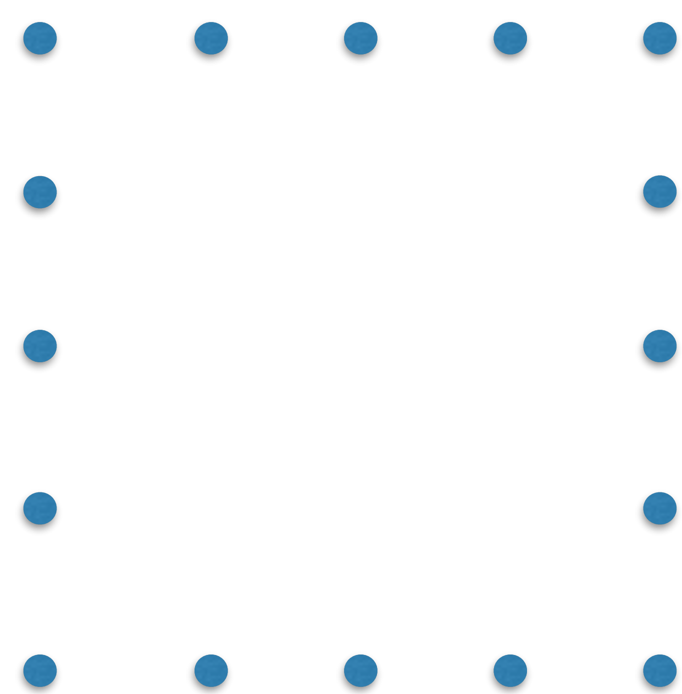
  //both are on CH and S will always contain at least 2 points



Graham scan: Details

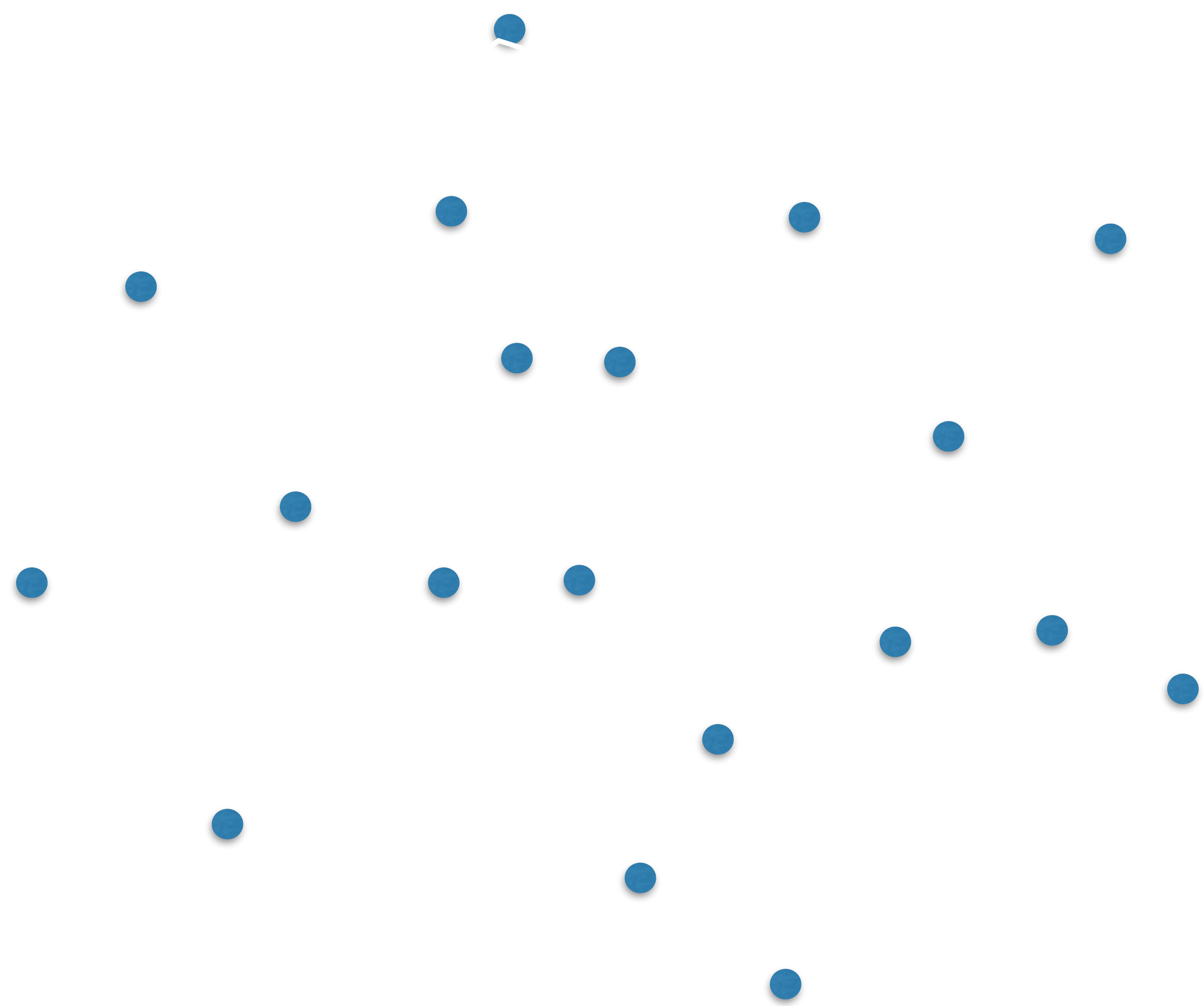
- Handling collinear-ities

What happens when you run on this input?

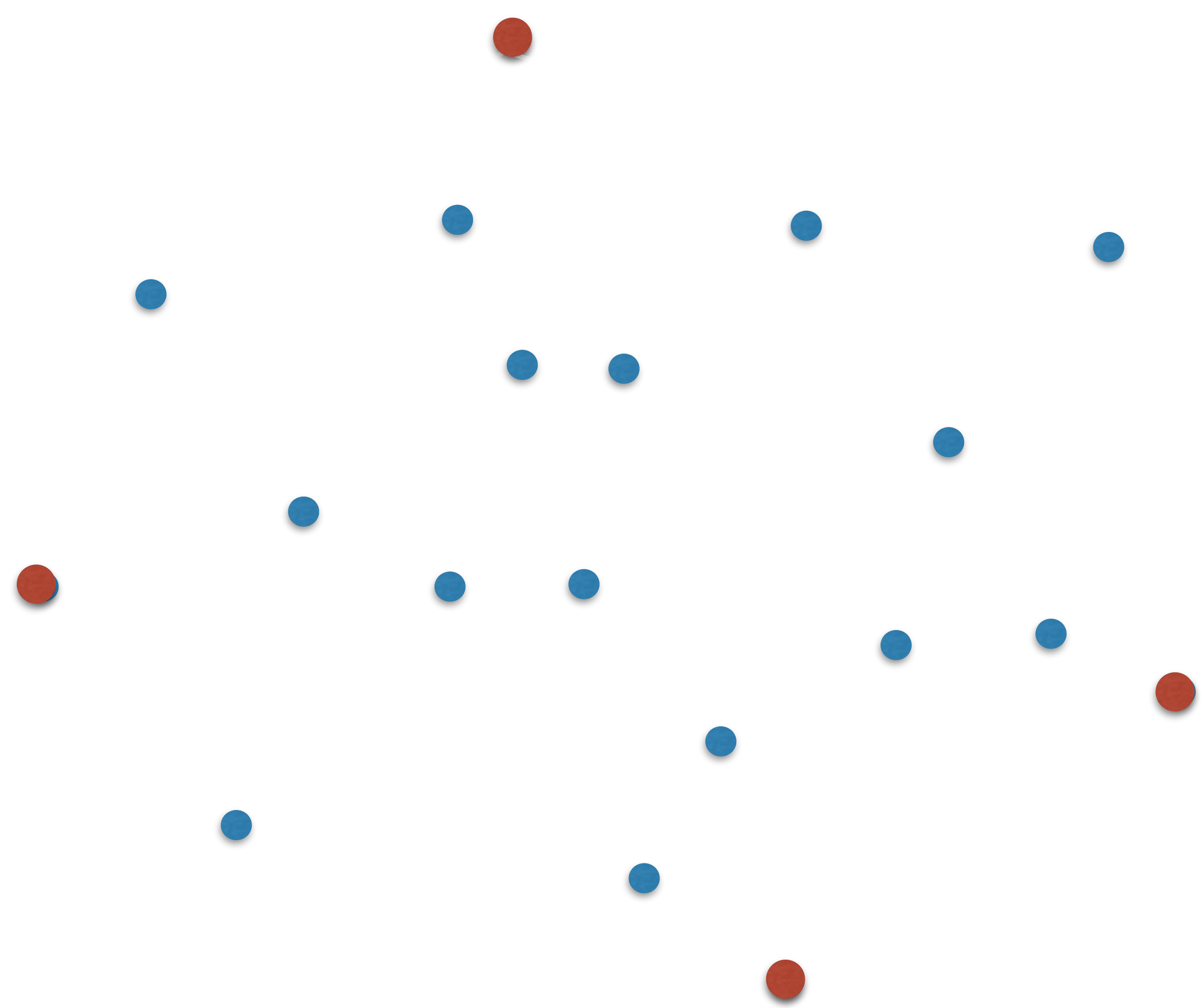


How can you fix it?

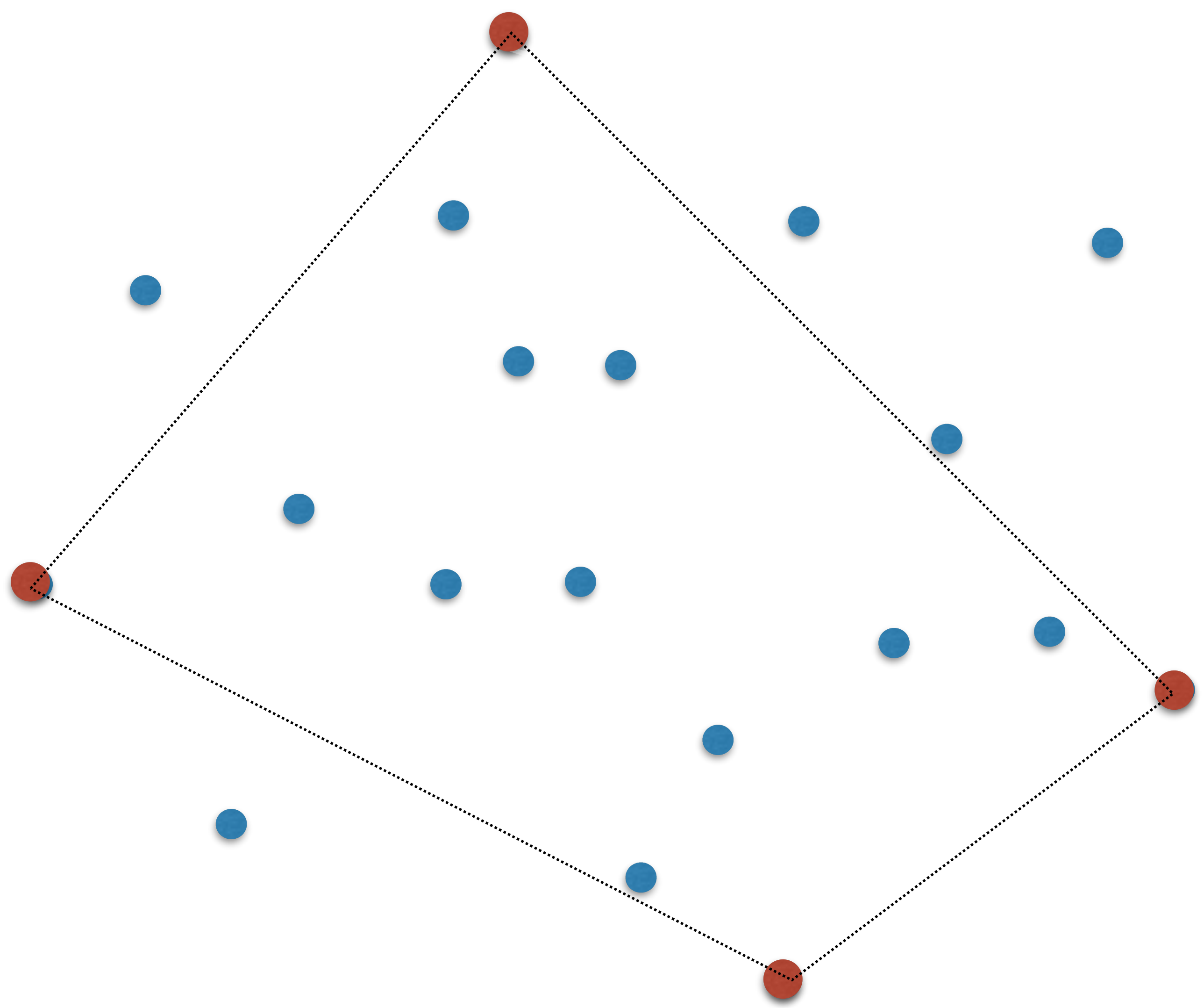
Speeding up Graham scan



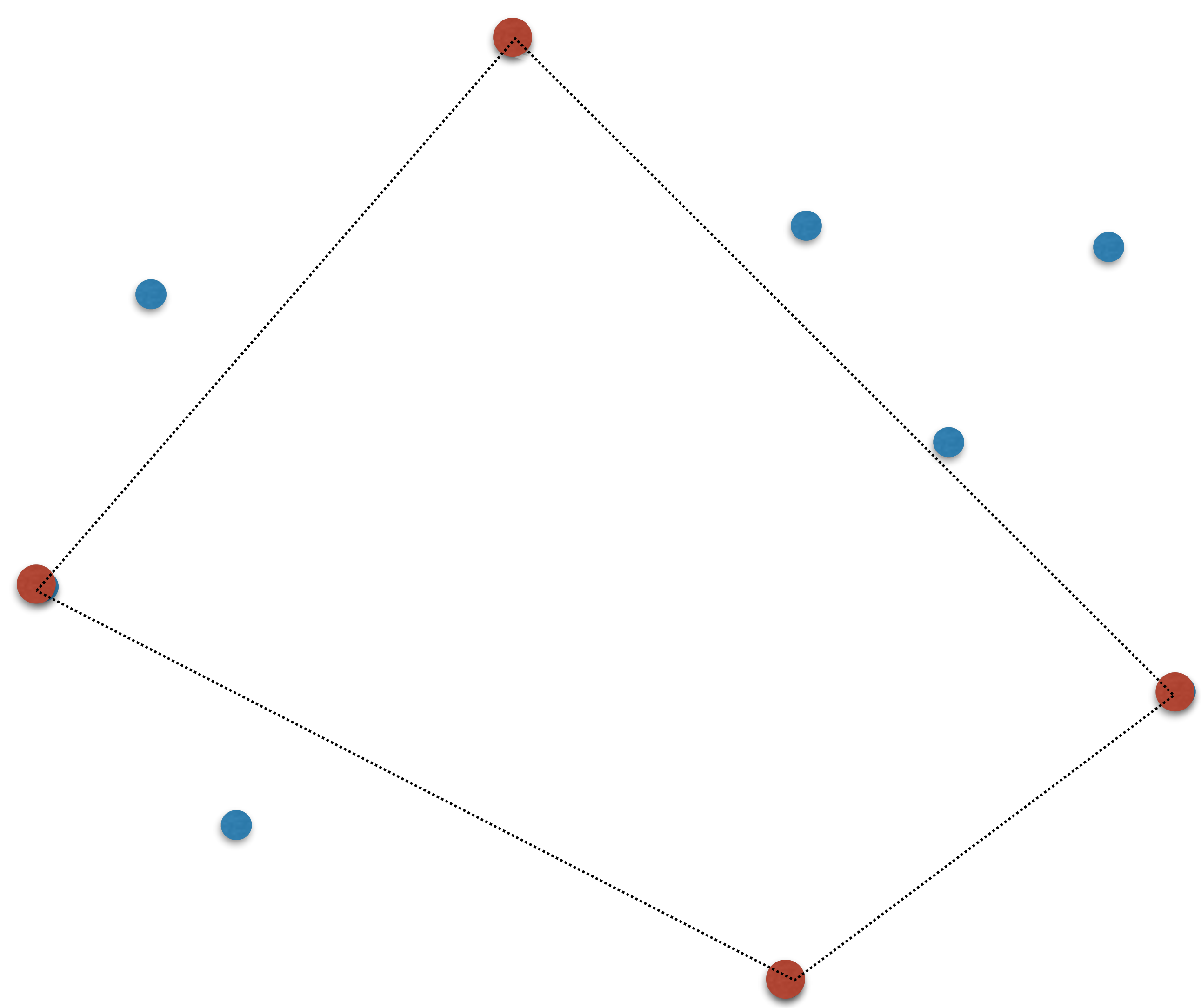
Speeding up Graham scan



Speeding up Graham scan



Speeding up Graham scan



Speeding up Graham scan