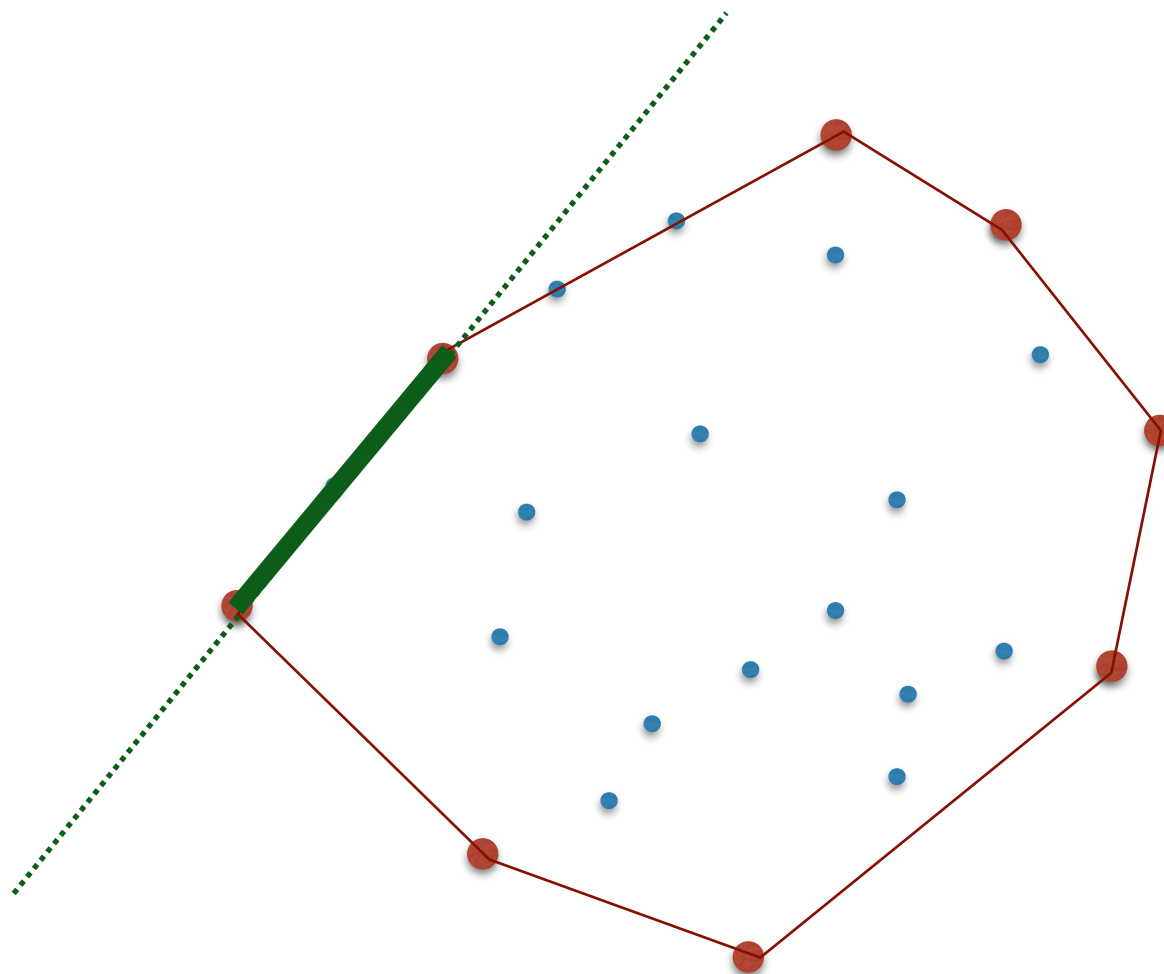


Computational Geometry [csci 3250]
Laura Toma
Bowdoin College

Properties of CH

- All edges of CH are extreme and all extreme edges of P are on the CH
- All points of CH are extreme and all extreme points of P are on the CH
- All internal angles are < 180
- Walking counterclockwise—> left turns
- Points on CH are sorted in radial order wrt a point inside

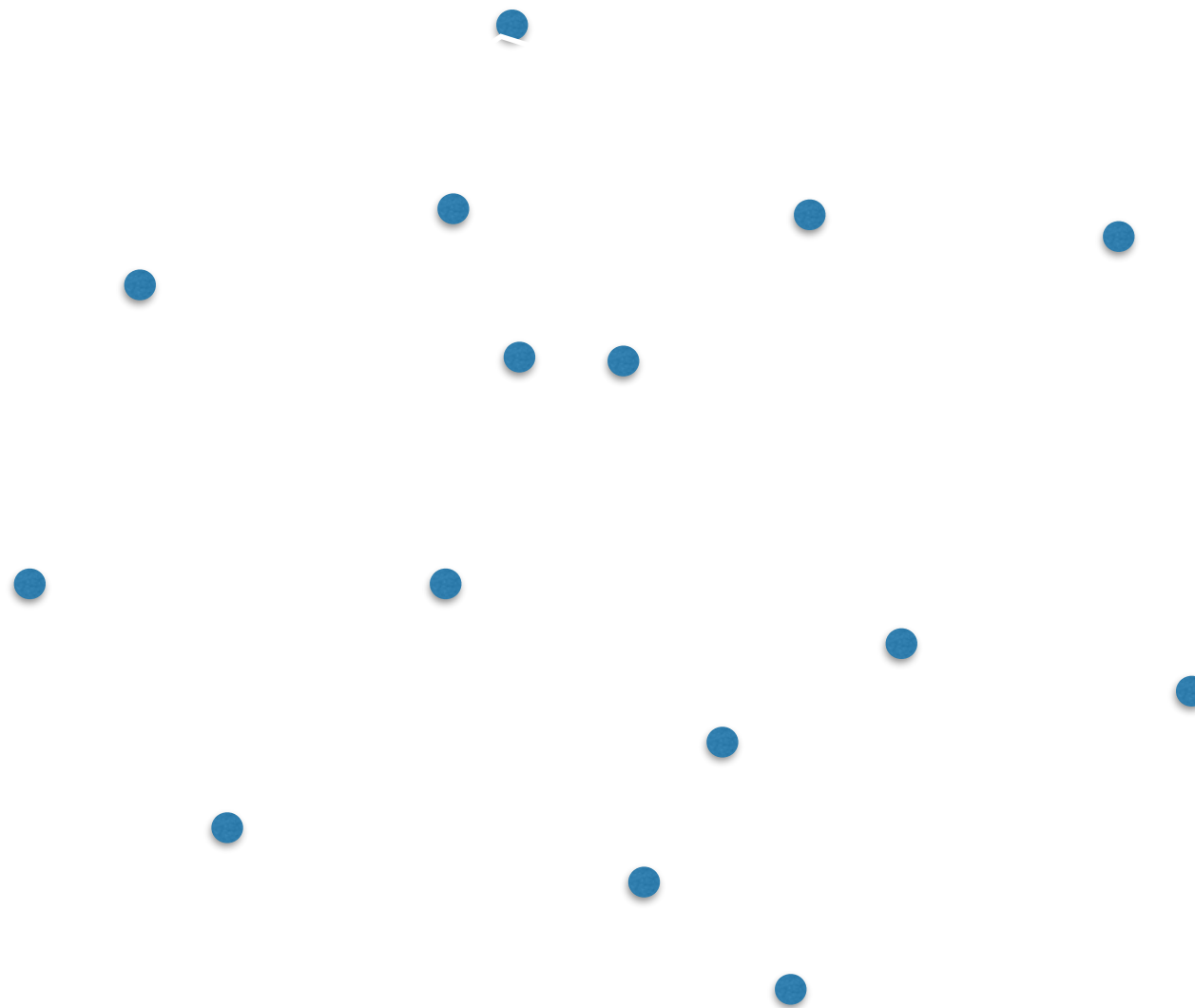


Outline

- Last time:
 - Brute force
 - Gift wrapping
 - Quickhull
 - Graham scan
- Next
 - Andrew's monotone chain algorithm
 - Exercises
 - Lower bound
 - More algorithms
 - Incremental CH
 - Divide-and-conquer CH

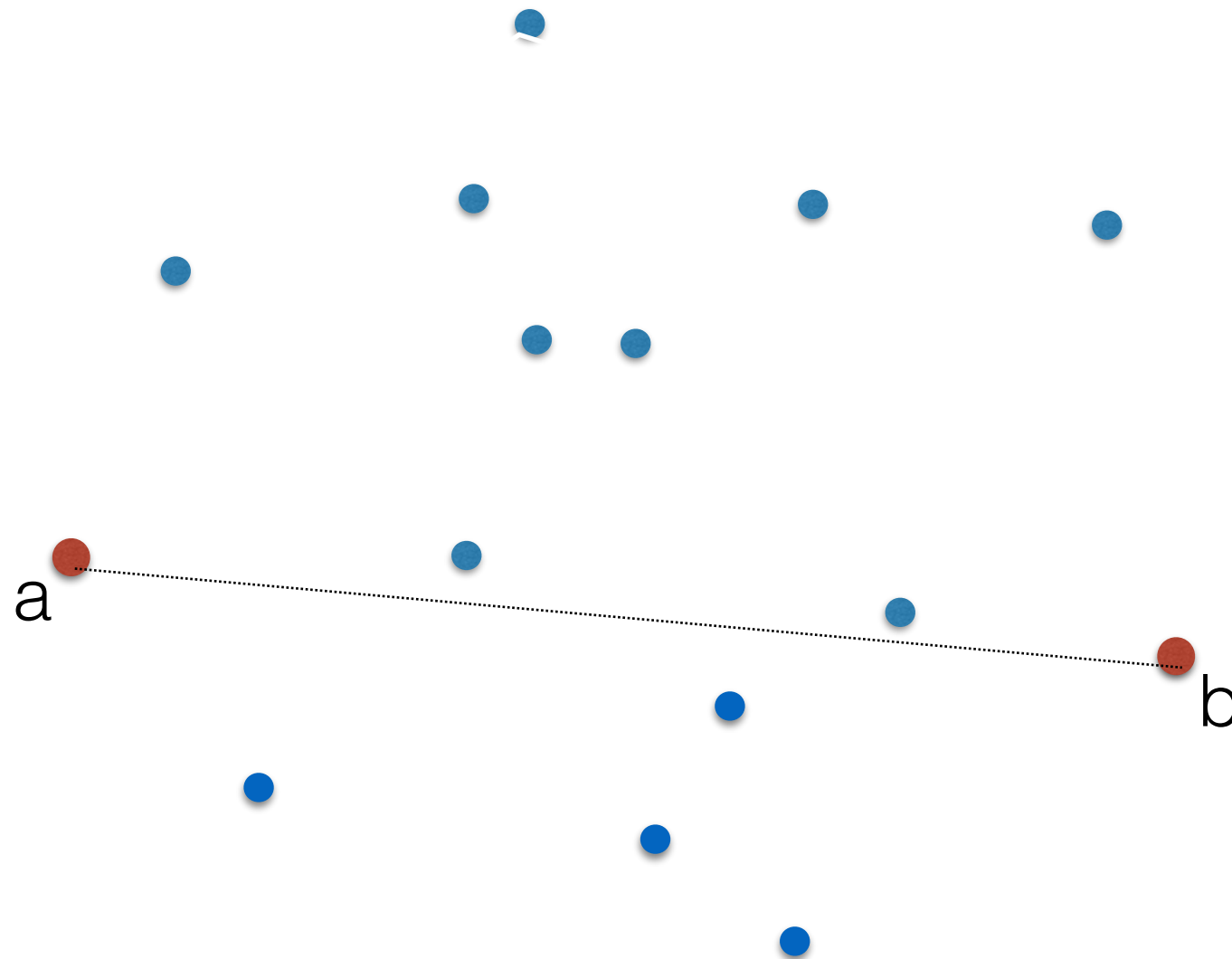
Andrew's Monotone Chain Algorithm (1979)

- Alternative to Graham's scan
- Idea: Find upper hull and lower hulls separately



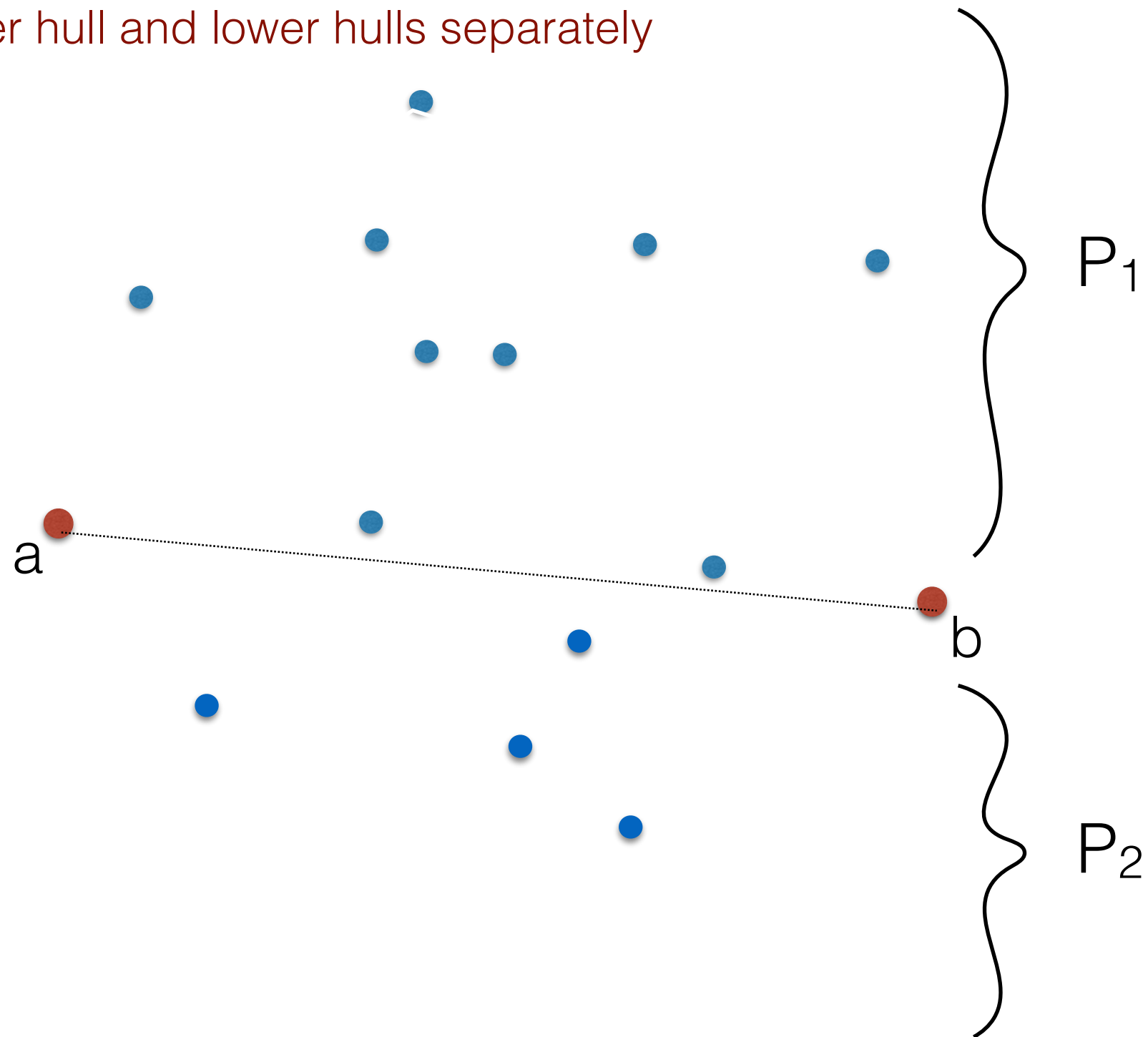
Andrew's Monotone Chain Algorithm (1979)

- Alternative to Graham's scan
- Idea: Find upper hull and lower hulls separately



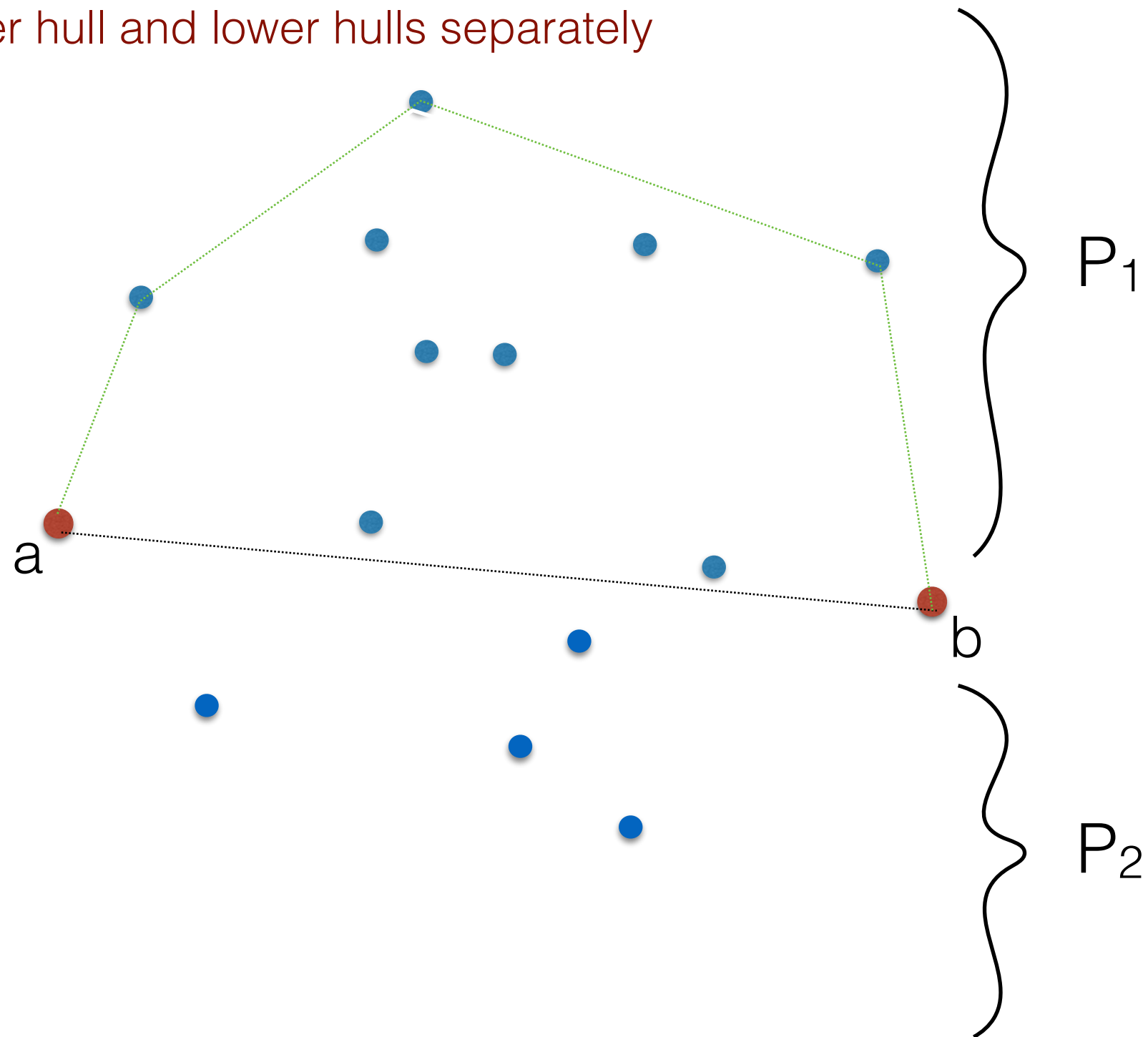
Andrew's Monotone Chain Algorithm (1979)

- Alternative to Graham's scan
- Idea: Find upper hull and lower hulls separately



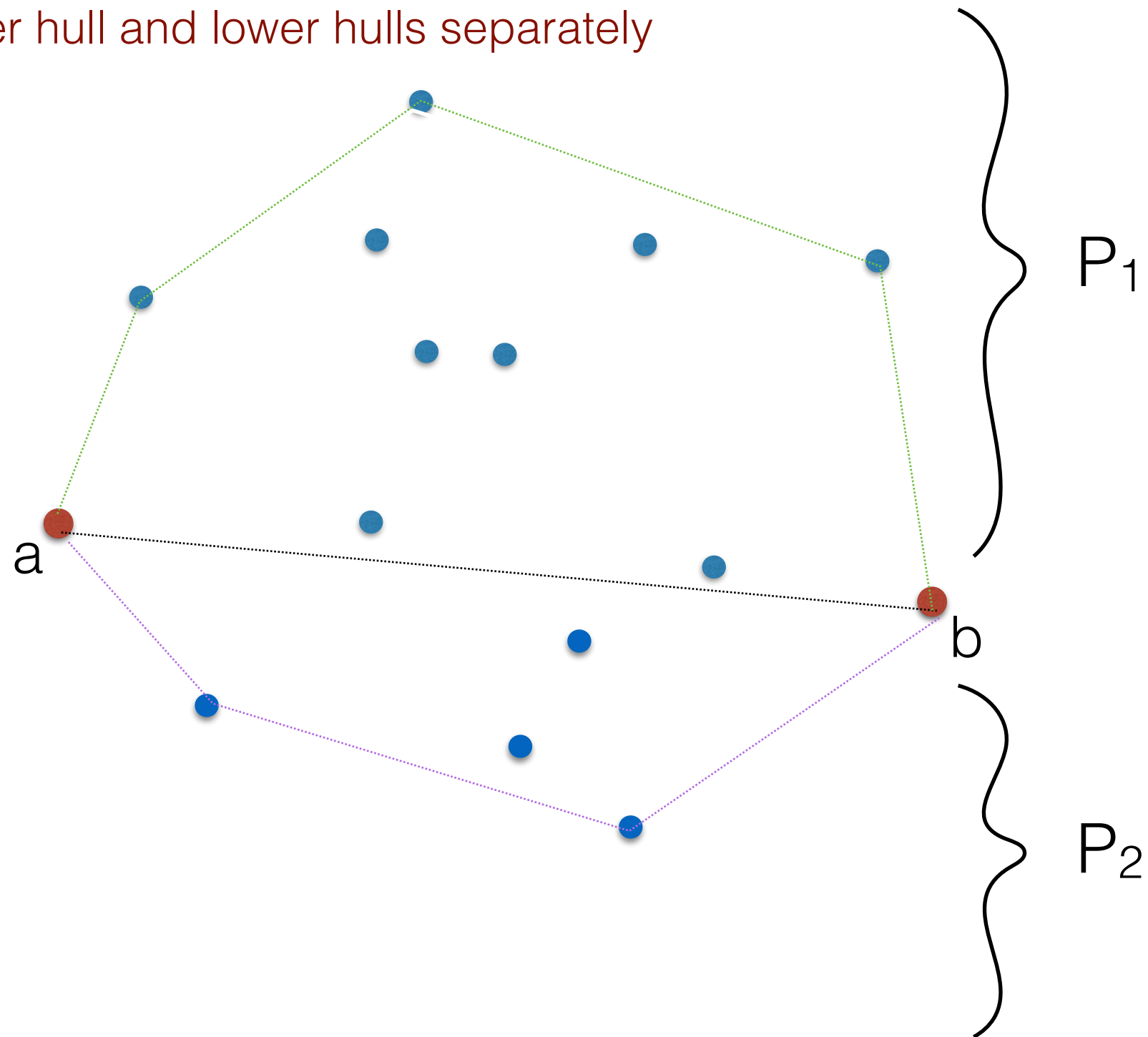
Andrew's Monotone Chain Algorithm (1979)

- Alternative to Graham's scan
- Idea: Find upper hull and lower hulls separately



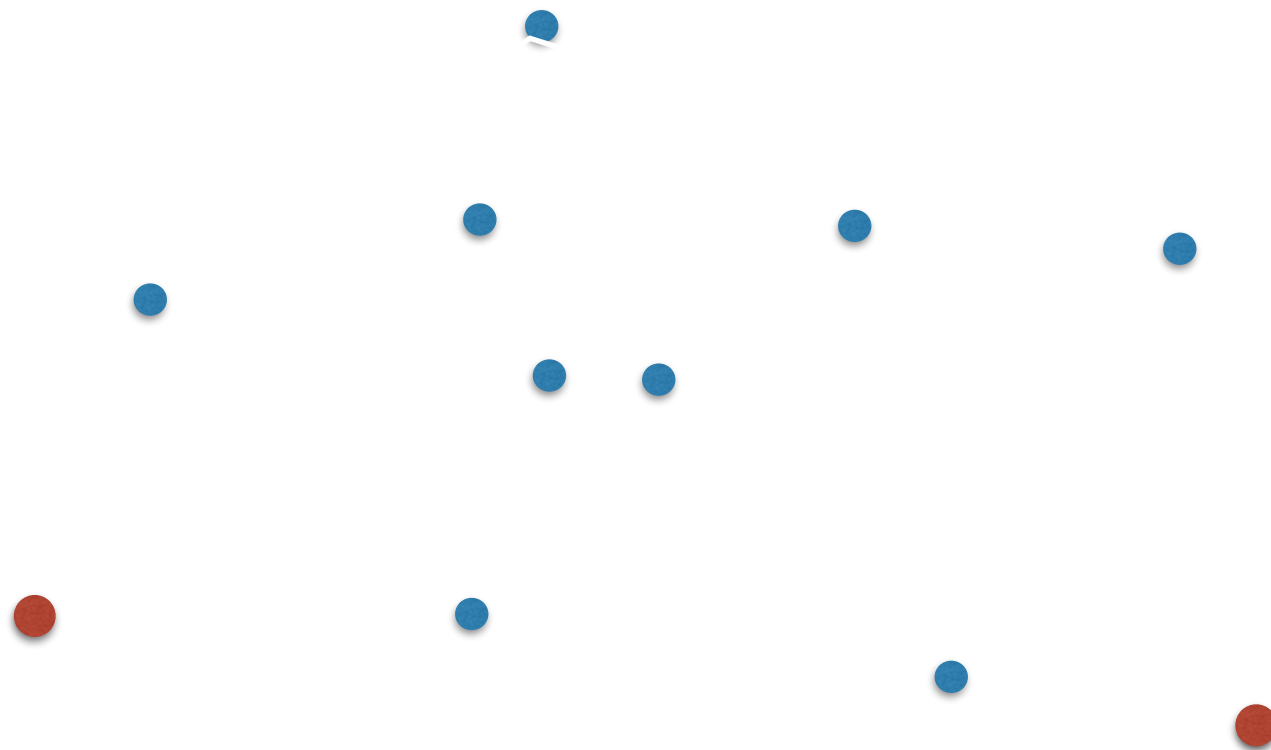
Andrew's Monotone Chain Algorithm (1979)

- Alternative to Graham's scan
- Idea: Find upper hull and lower hulls separately



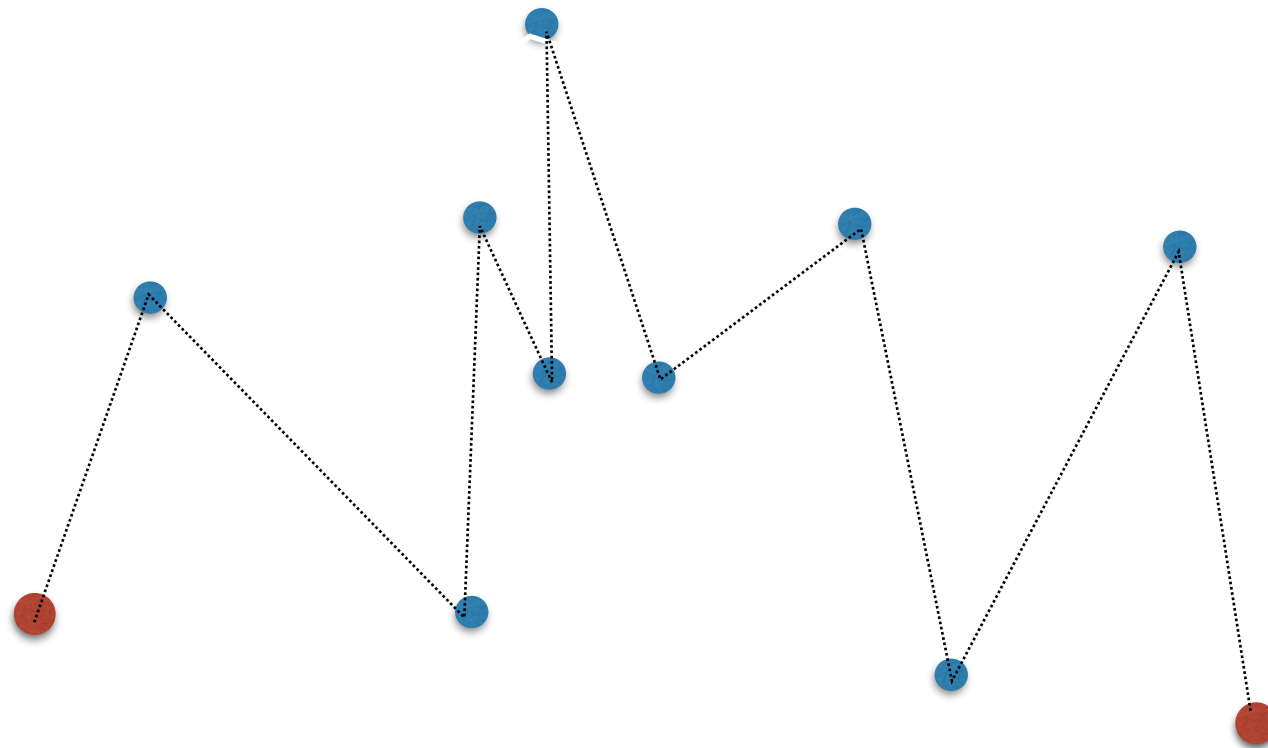
Andrew's Monotone Chain Algorithm (1979)

- Find CH(P1): Traverse points in (x,y) order (i.e. lexicographically)



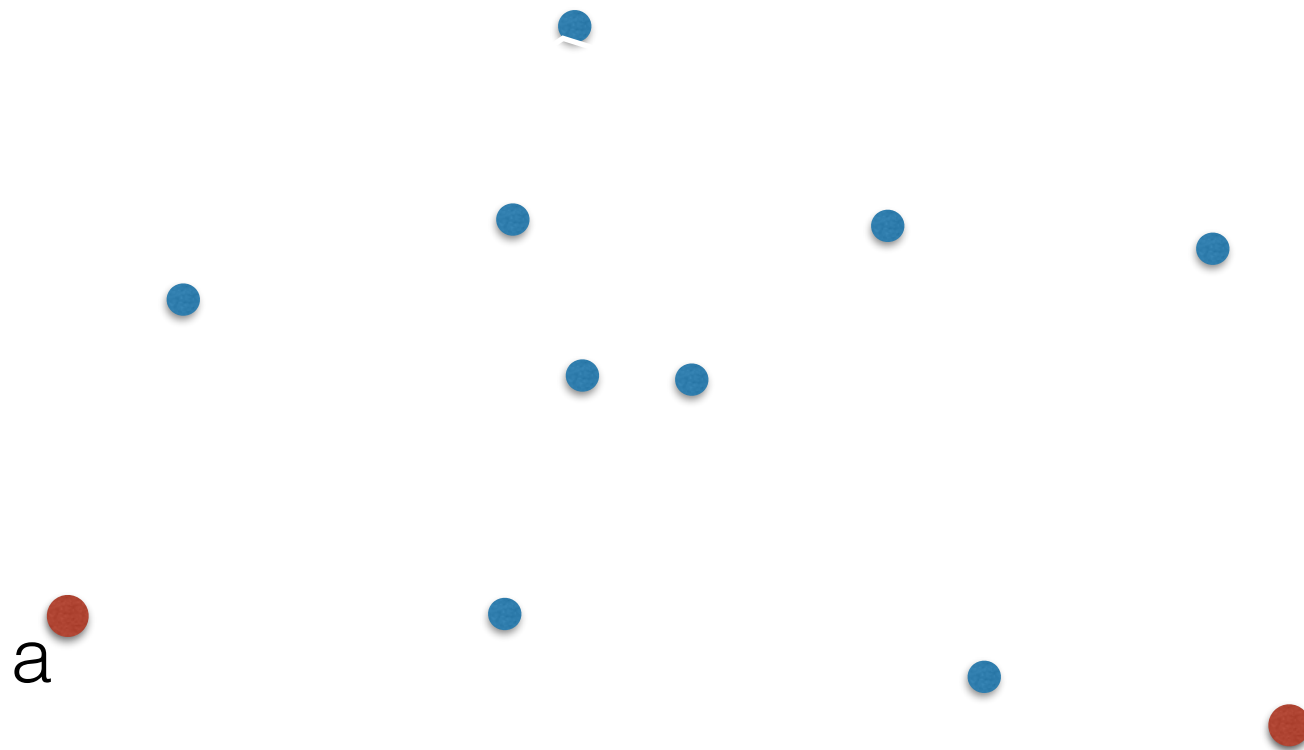
Andrew's Monotone Chain Algorithm (1979)

- Find CH(P1): Traverse points in (x,y) order (i.e. lexicographically)



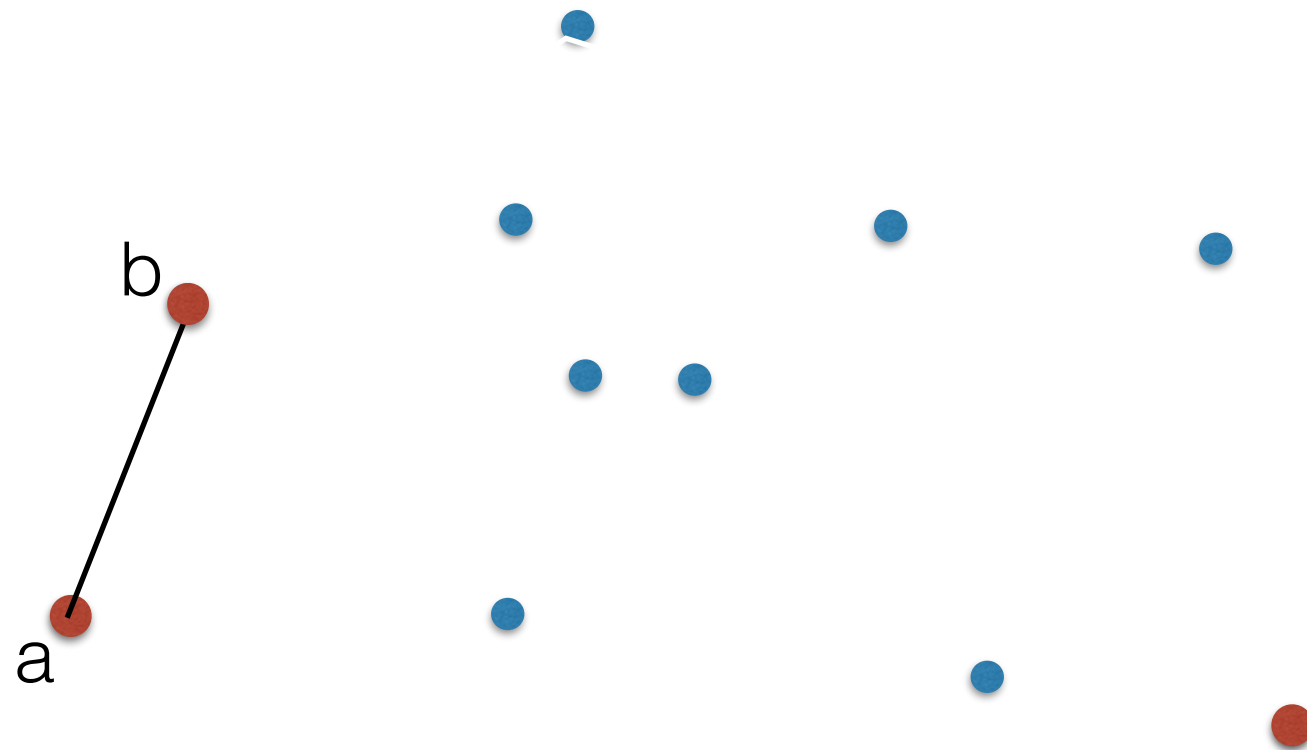
Andrew's Monotone Chain Algorithm (1979)

- Find CH(P1): Traverse points in (x,y) order (i.e. lexicographically)



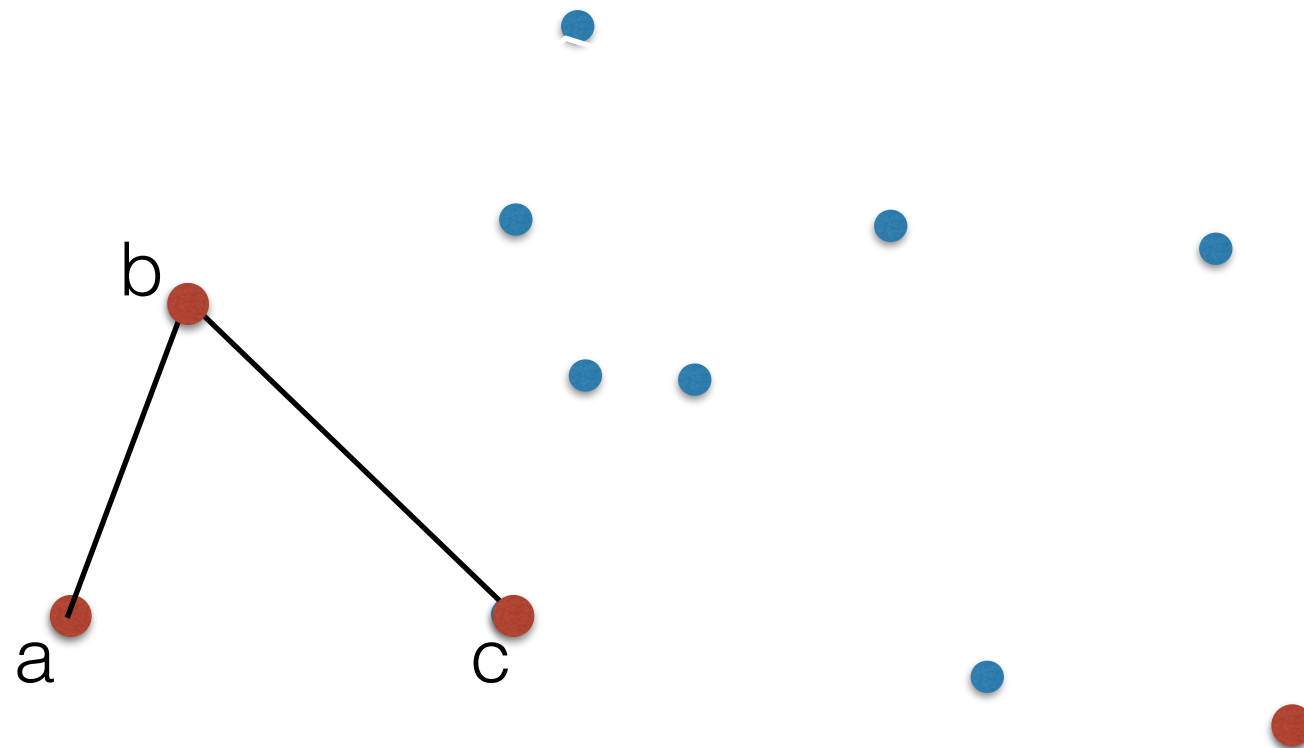
Andrew's Monotone Chain Algorithm (1979)

- Find CH(P1): Traverse points in (x,y) order (i.e. lexicographically)



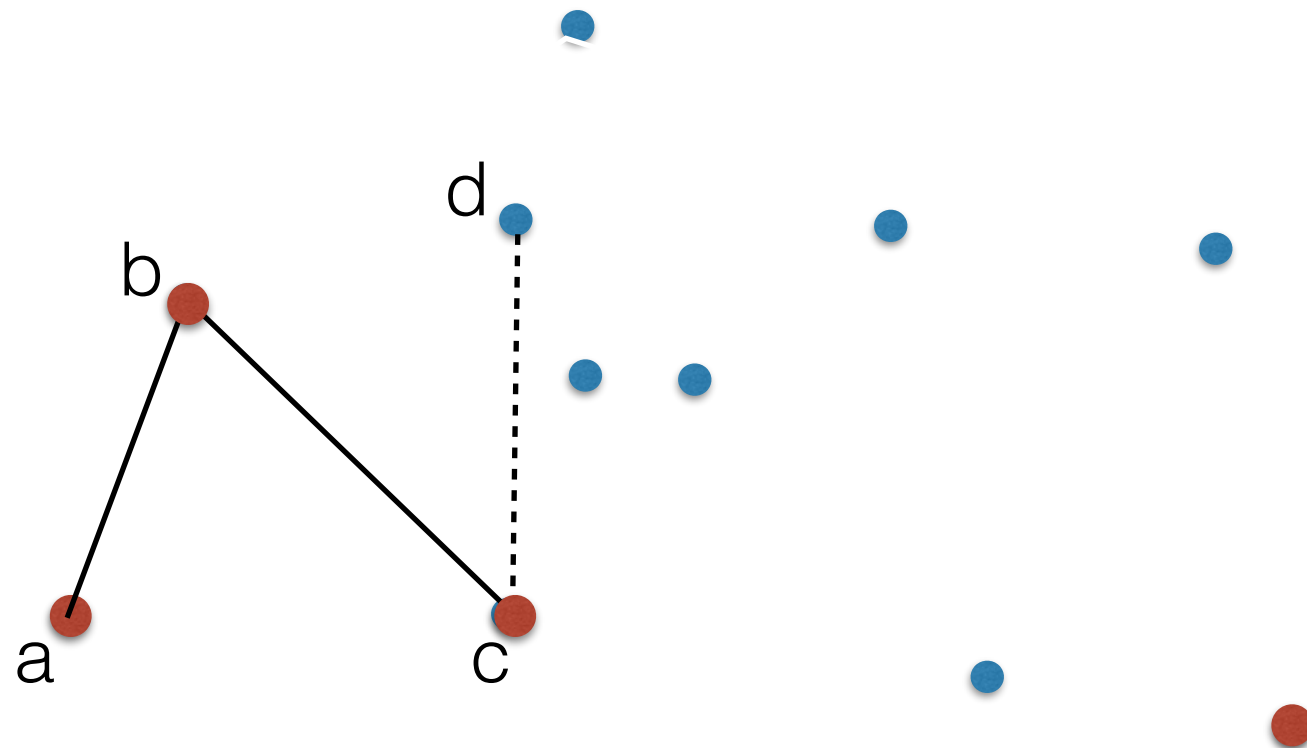
Andrew's Monotone Chain Algorithm (1979)

- Find CH(P1): Traverse points in (x,y) order (i.e. lexicographically)



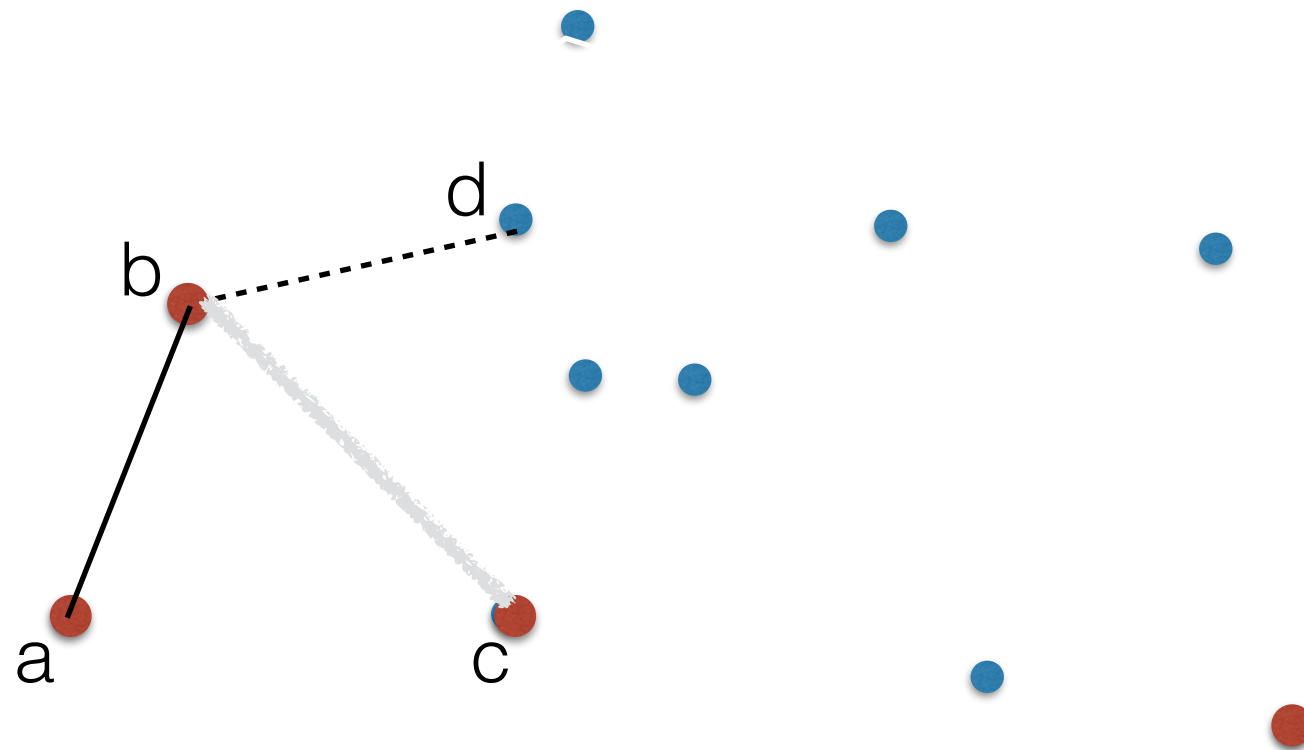
Andrew's Monotone Chain Algorithm (1979)

- Find CH(P1): Traverse points in (x,y) order (i.e. lexicographically)



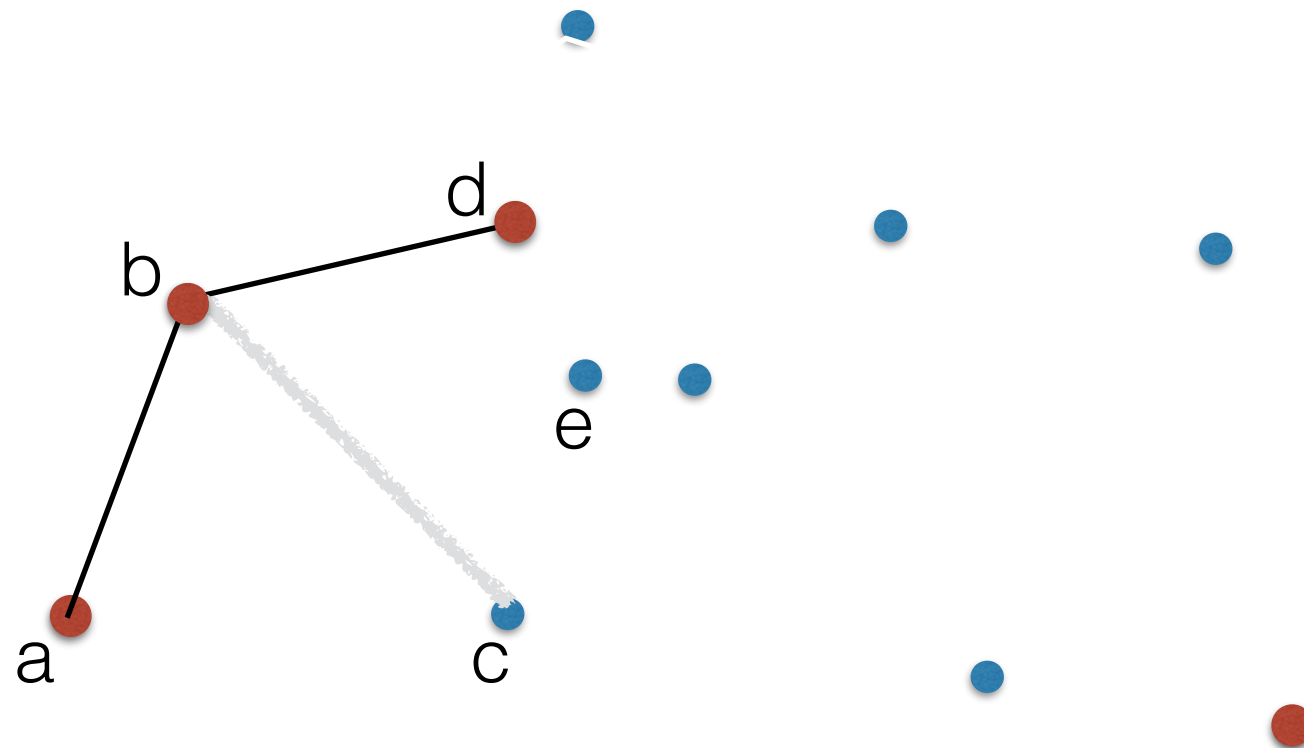
Andrew's Monotone Chain Algorithm (1979)

- Find CH(P1): Traverse points in (x,y) order (i.e. lexicographically)



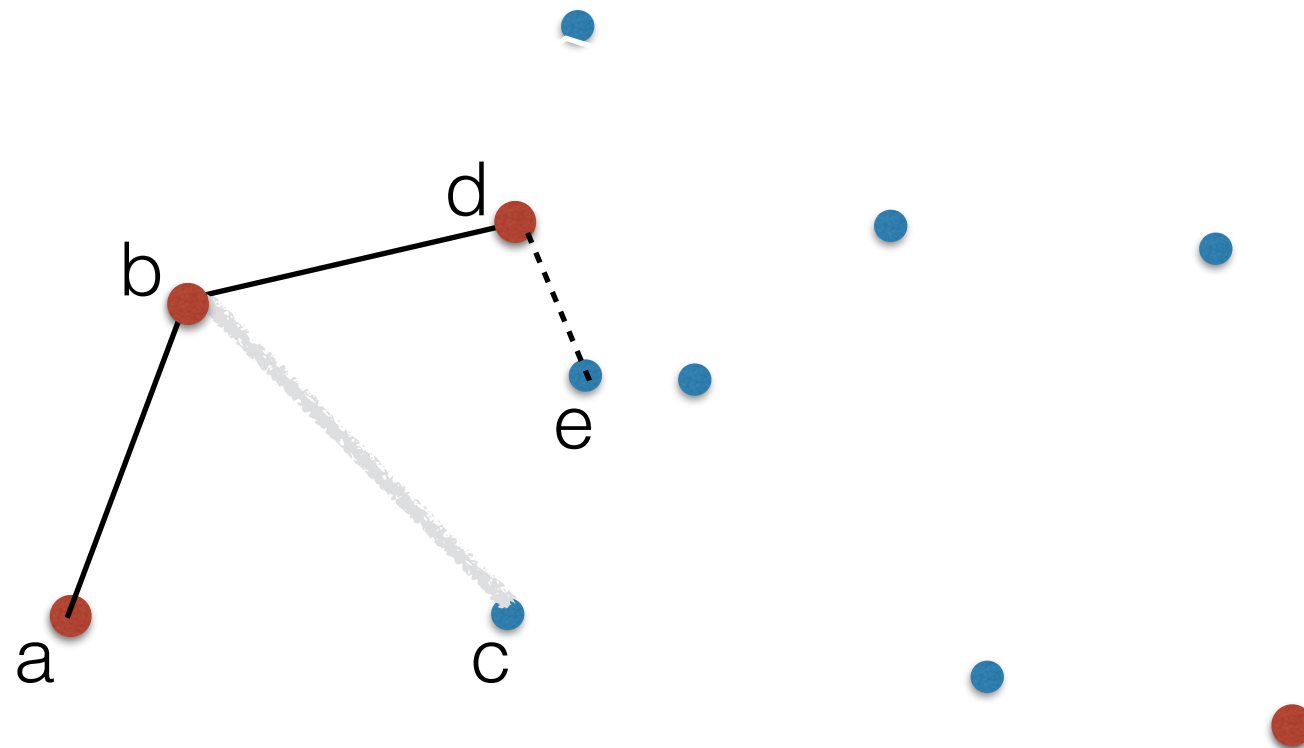
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P_1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



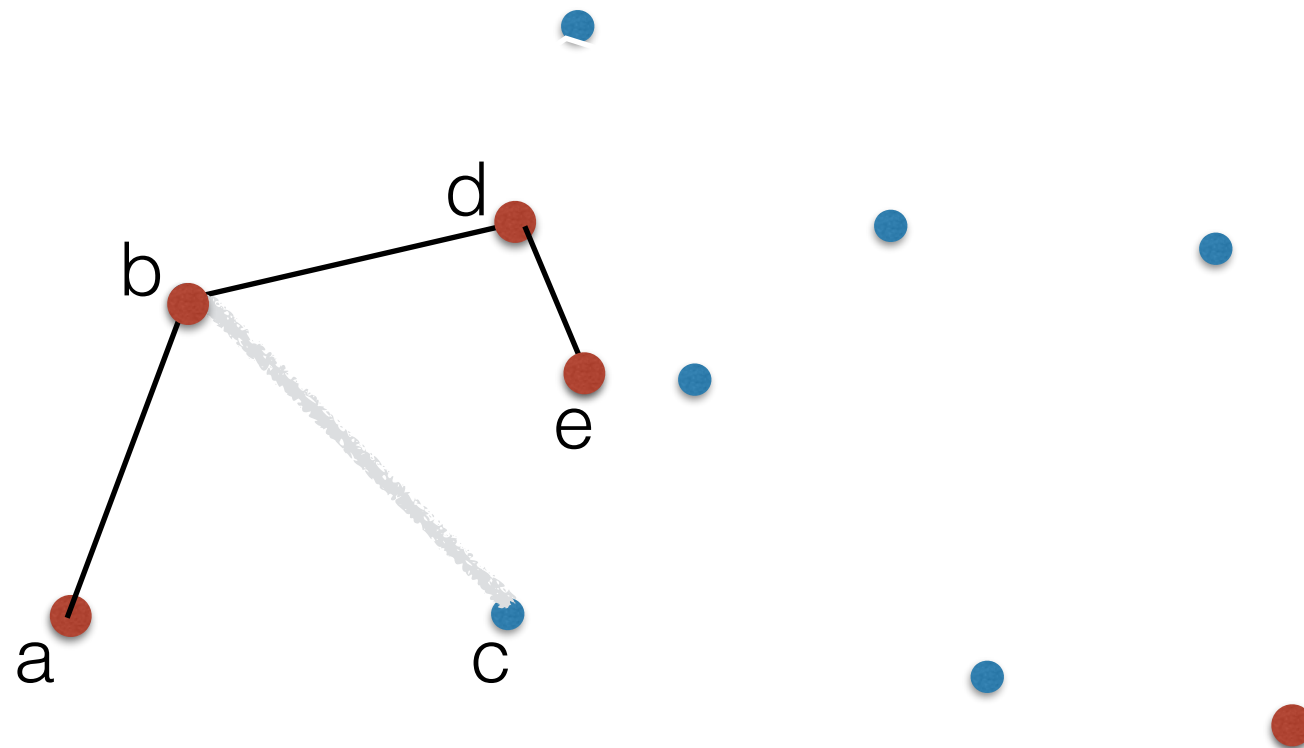
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P_1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



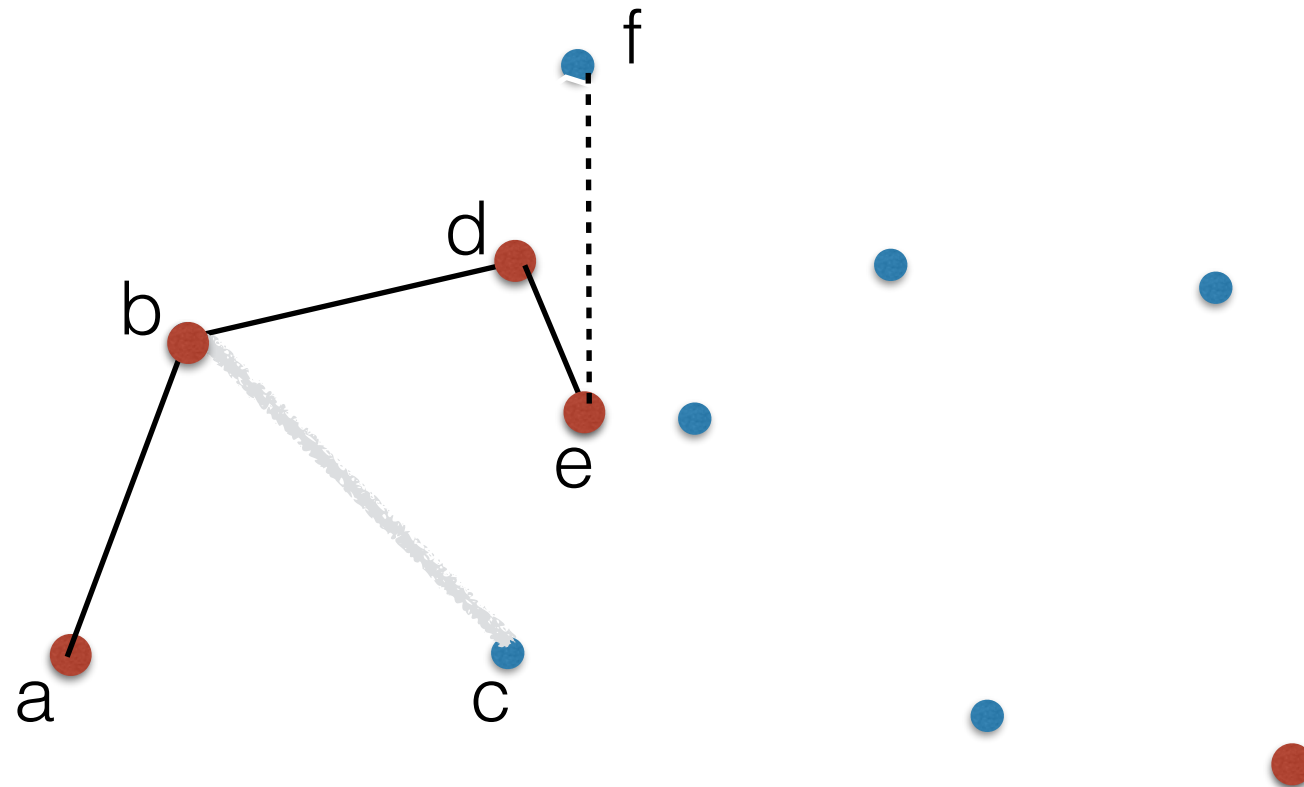
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P_1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



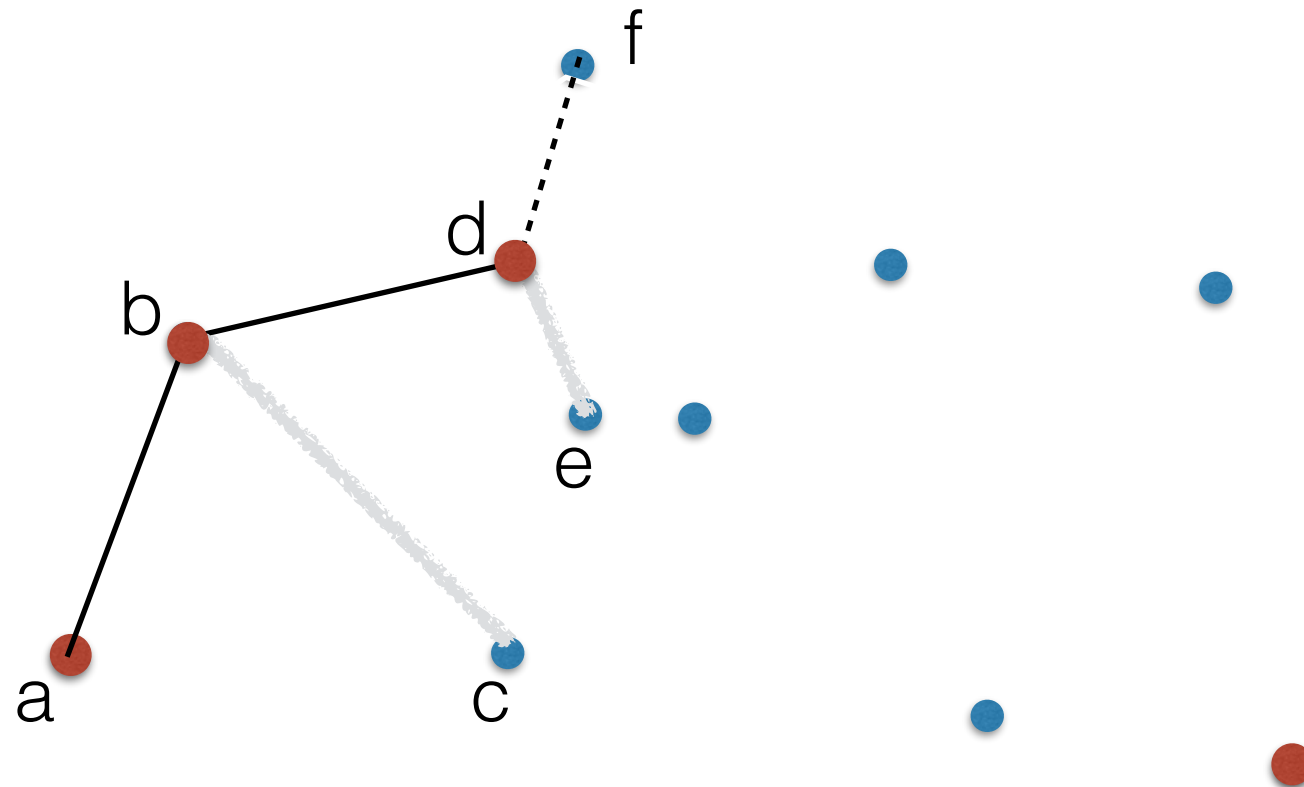
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P_1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



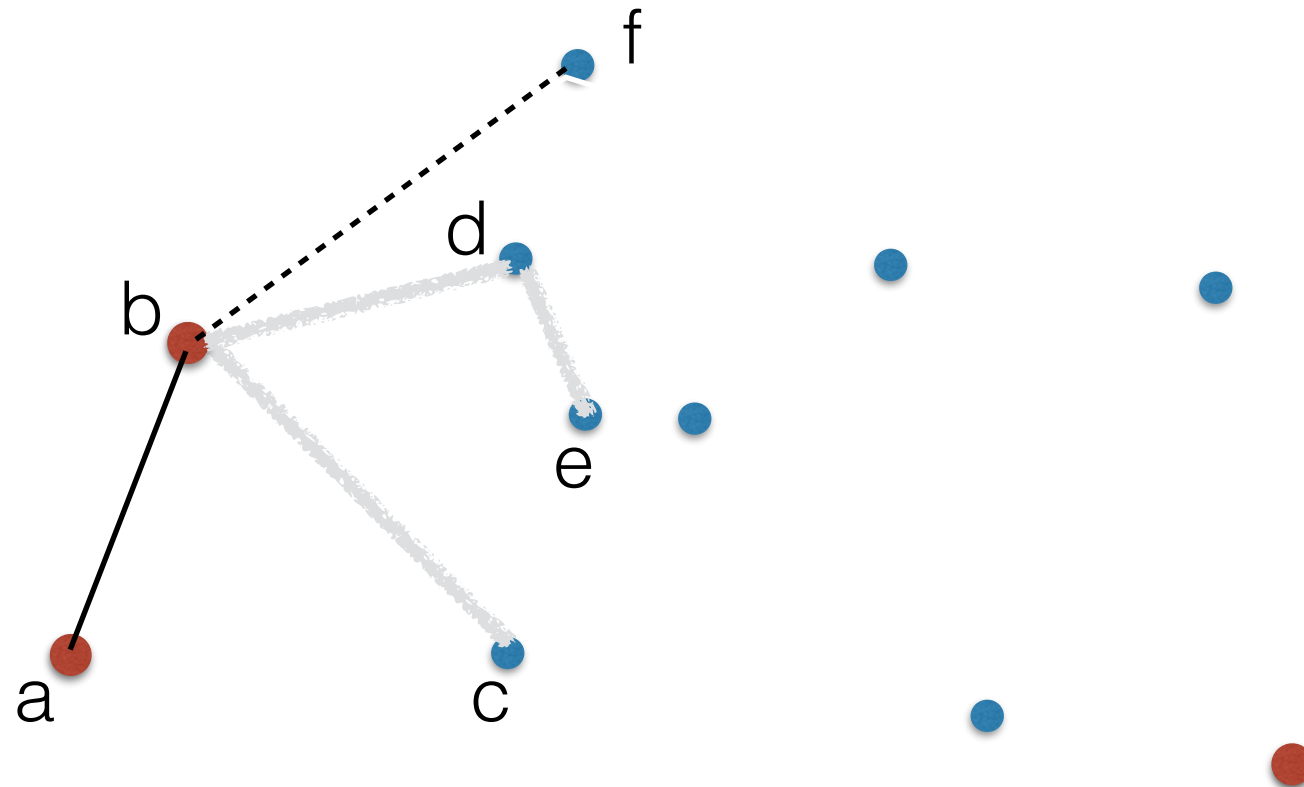
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



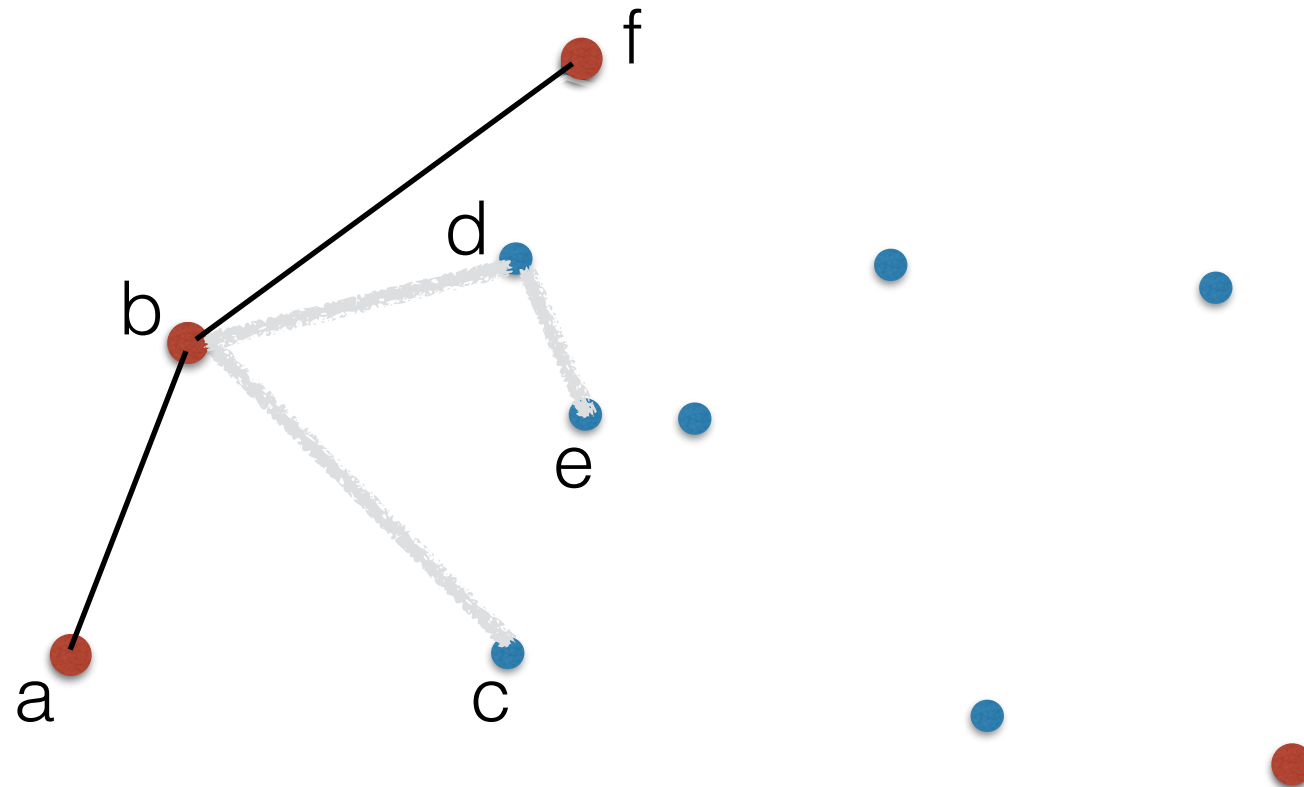
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



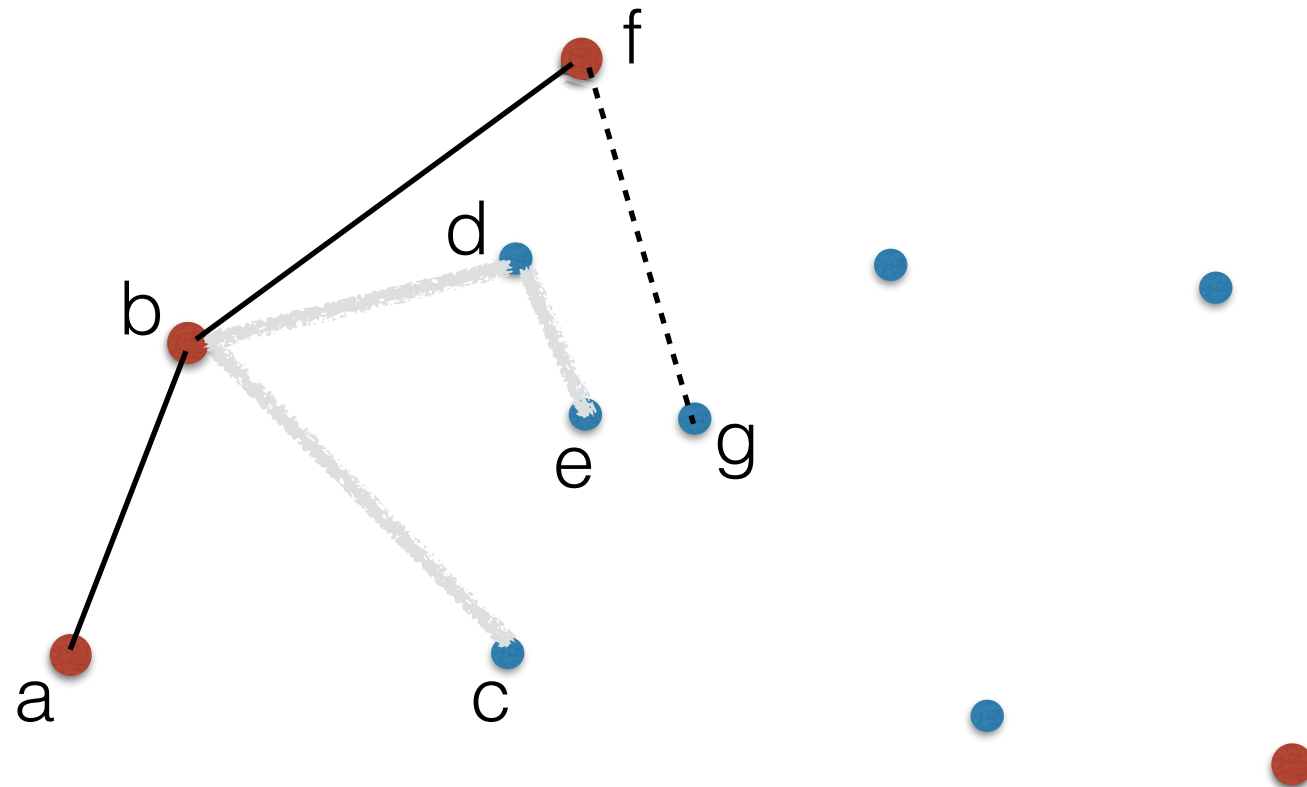
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



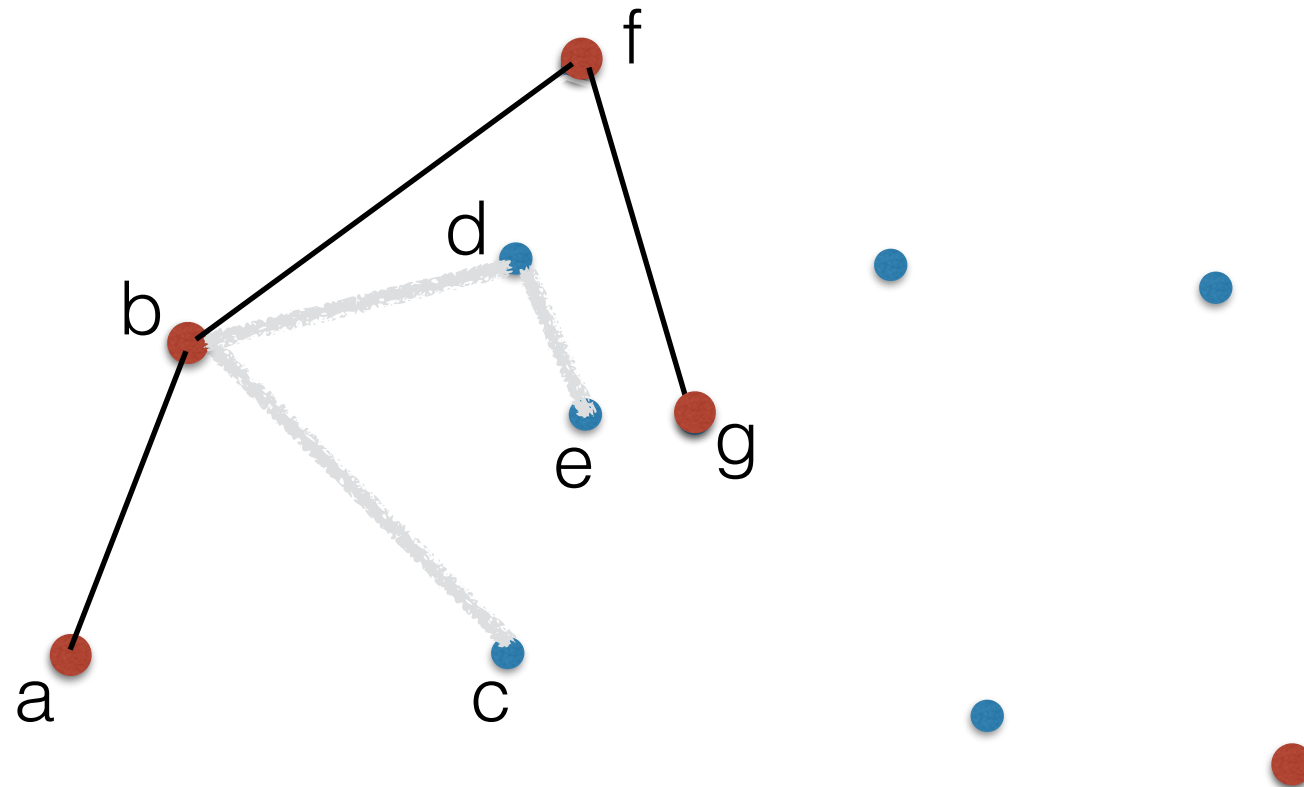
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



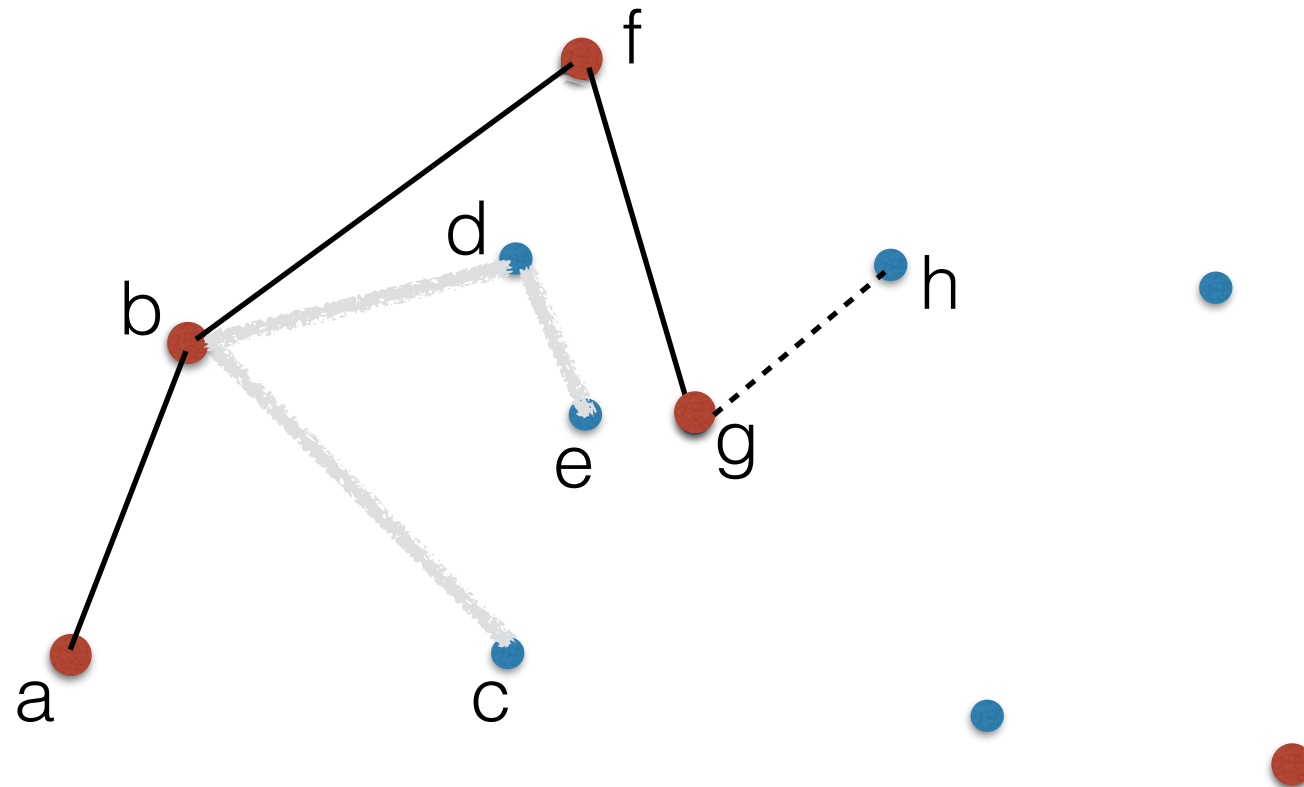
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



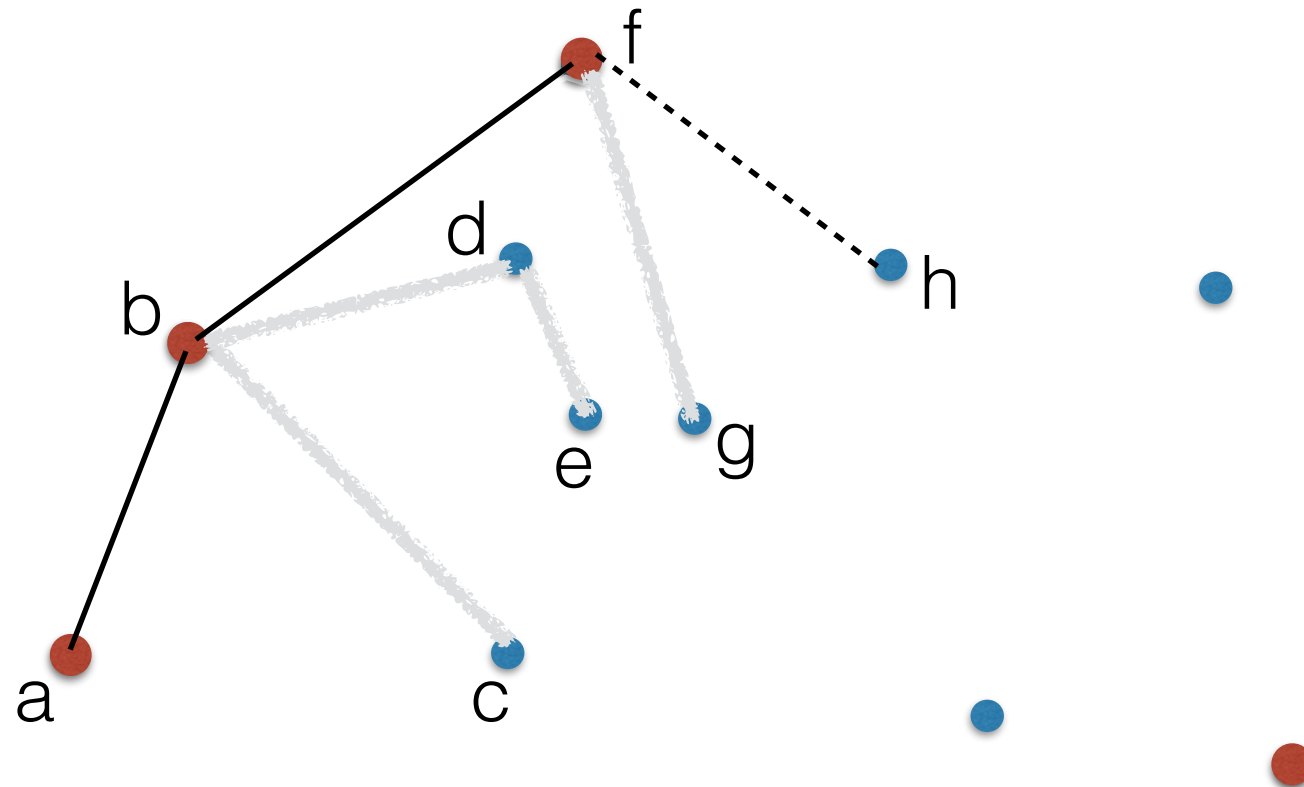
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



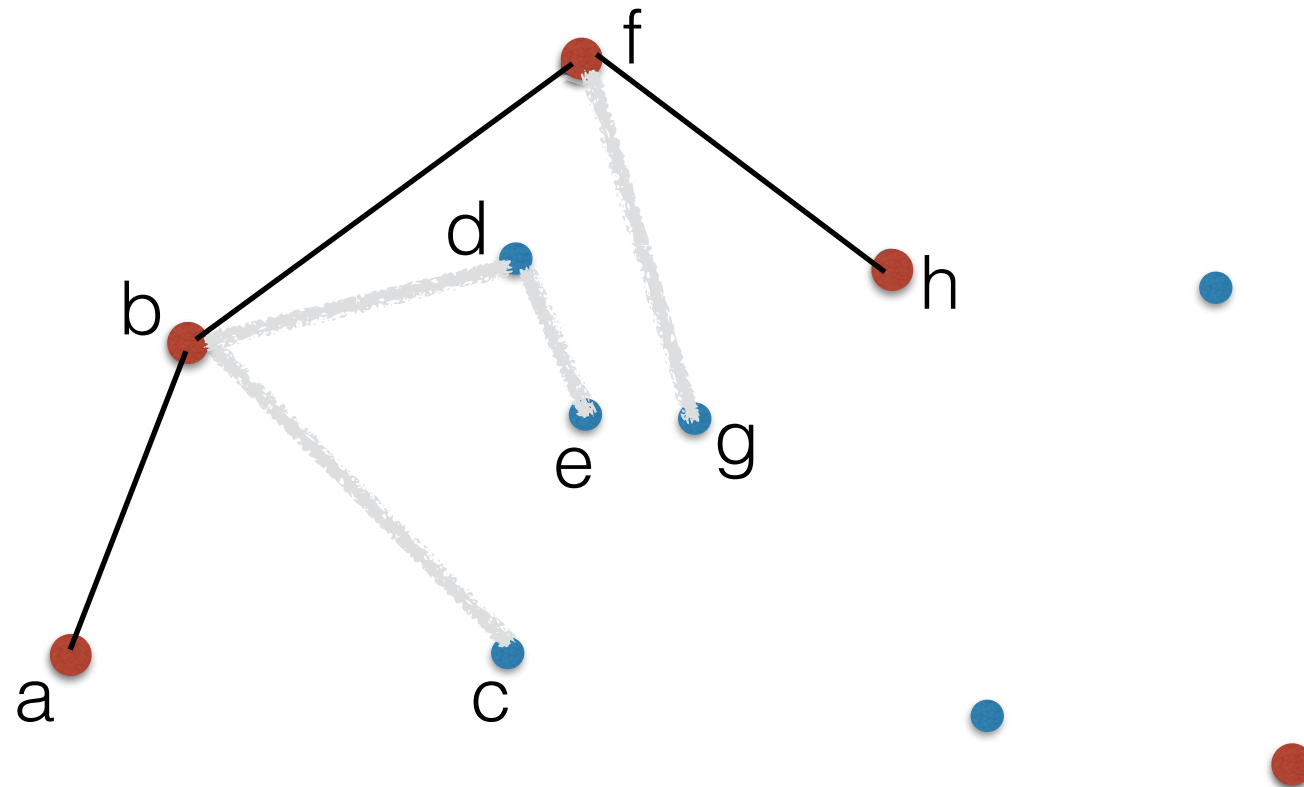
Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



Andrew's Monotone Chain Algorithm (1979)

- Goal: find the CH of P1
- Idea: Traverse points in (x,y) order (i.e. lexicographically)



and so on..

Andrew's Monotone Chain Algorithm (1979)

- Alternative to Graham's scan
- Idea: Traverse points in (x,y) lexicographic order (instead of radial order)
- Runs in sort + scan
- Sorting lexicographically is faster than sorting radially

Convex hull: summary

Naive	$O(n^3)$	
Gift wrapping	$O(nh)$	1970
Graham scan	$O(n \lg n)$	1972
Quickhull	$O(n^2)$	1977

Can we do better?

Lower bound

What is a lower bound?

- Given an algorithm A, its **worst-case running time** is the largest running time on any input of size n

$\max_{|P|=n} \{T(n) \mid T(n) \text{ is the running time of algorithm A on input P}\}$

What is a lower bound?

- Given an algorithm A, its **worst-case running time** is the largest running time on any input of size n

$$WC_A(n) = \max_{|P|=n} \{T(n) \mid T(n) \text{ is the running time of algorithm A on input P}\}$$

- A lower bound for CH:
 - What is the **worst-case running time** of the **best possible** CH algorithm?
 - Consider all possible CH algorithms A, and take the overall smallest worst-case running time

$$\min_A \{WC_A(n)\}$$

Lower bounds

Lower bounds depend on the machine model.

The standard model is the decision tree (comparison) model.

Sorting lower bound in comparison model: $\Omega(n \lg n)$

- Theorem: Any sorting algorithm that uses only comparisons must take at least $\Omega(n \lg n)$ in the worst case.
- Proof: We saw this in Algorithms..

Lower bounds

Prove directly

Or via **reduction** from a problem known to have a lower bound

Lower bounds by reduction

- We know that

$$\Omega(n \lg n) \leq \text{sorting}$$

- If we could show that ConvexHull is at least as hard as sorting

$$\text{sorting} \leq \text{ConvexHull}$$

- This would imply that

$$\Omega(n \lg n) \leq \text{ConvexHull}$$

Lower bounds by reduction

- We know that

$$\Omega(n \lg n) \leq \text{sorting}$$

- If we could show that ConvexHull is at least as hard as sorting

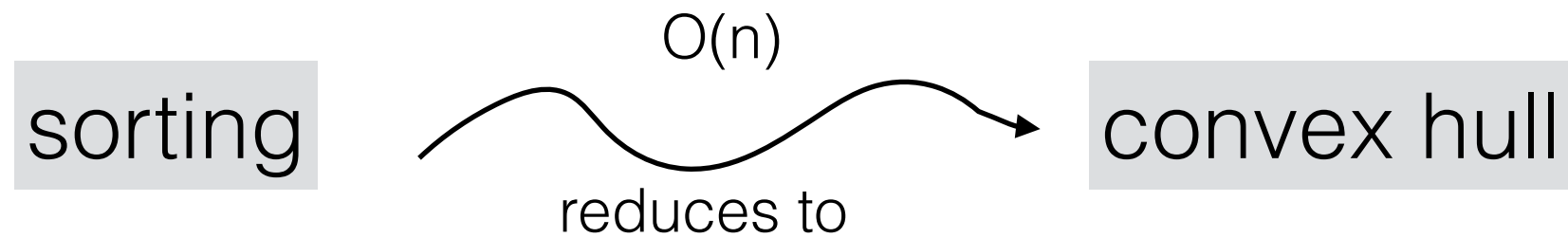
$$\text{sorting} \leq \text{ConvexHull}$$

How can we show
that
CH is harder than sorting?

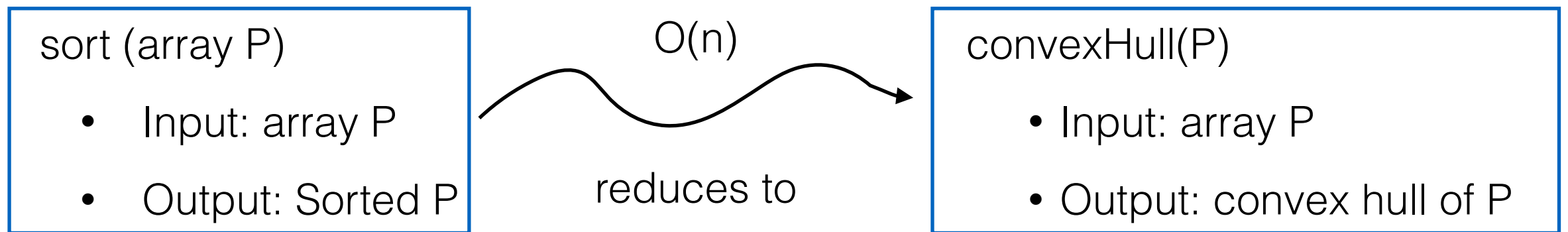
- This would imply that

$$\Omega(n \lg n) \leq \text{ConvexHull}$$

Convex Hull is harder than sorting



- We'll show that we can use Convex Hull to solve Sorting
- This means that Sorting "reduces" to Convex Hull



- We'll show that (for any P) there exists some instance of the CH problem that sorts P, and we can build this instance in $O(n)$ time

sort (array P)

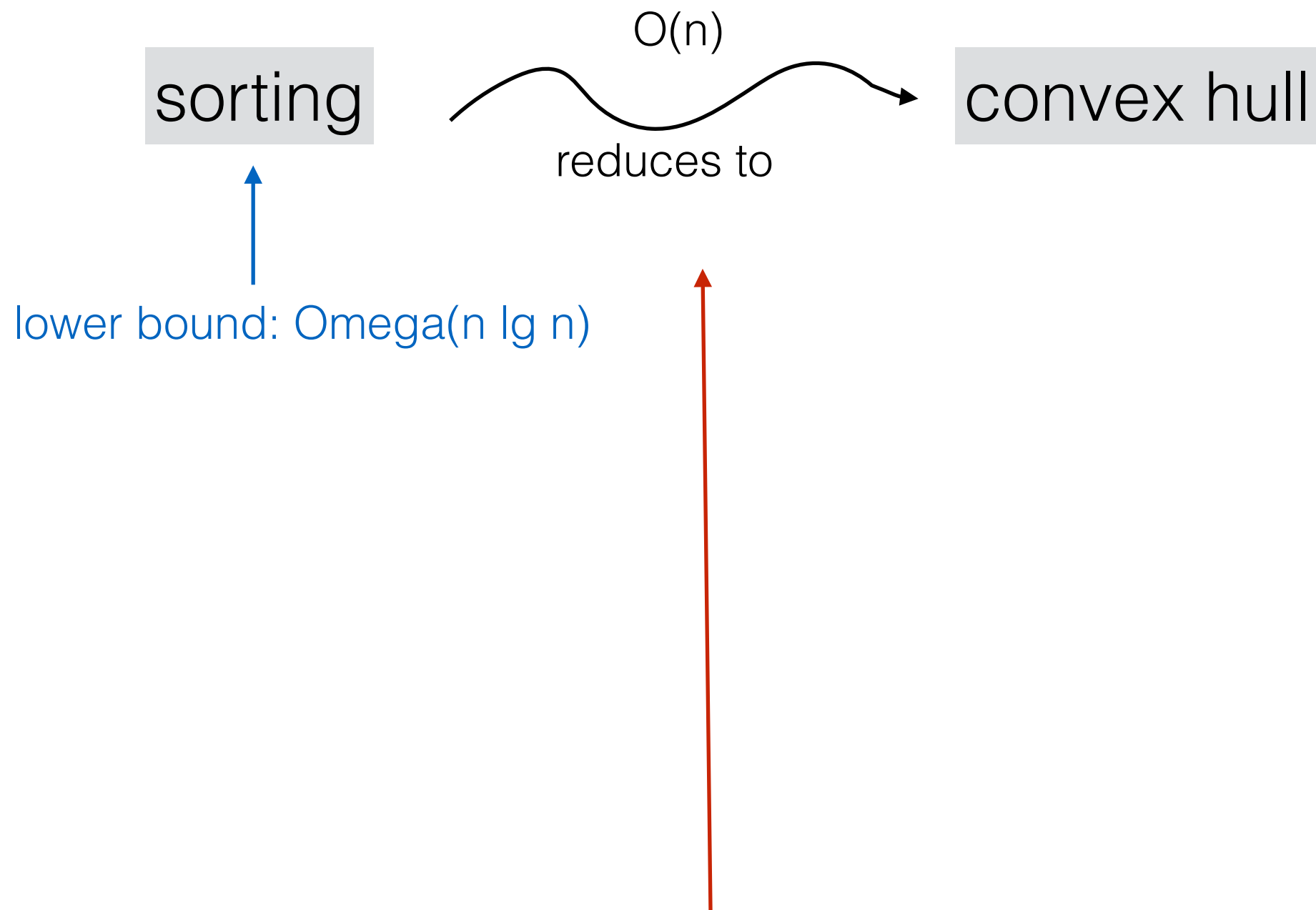
- create a set P' of points from P
- find CH(P')
- use the convex hull to infer sorted order of P

sort (array P)

- create a set P' of points from P
- find $CH(P')$
- use the convex hull to infer sorted order of P

- ANALYSIS:
 - run $CH(P)$
 - $O(n)$ to create P' and infer sorted order of P
 - running time of sorting = running time of convexhull + $O(n)$
- Therefore the reduction gives us an $O(CH(n)) + O(n)$ algorithm for sorting, which means that ConvexHull is an upper bound for sorting
- But we know that we cannot sort faster than $\Omega(n \lg n)$ in the worst case
- Therefore we cannot solve CH faster than $\Omega(n \lg n)$ in the worst case

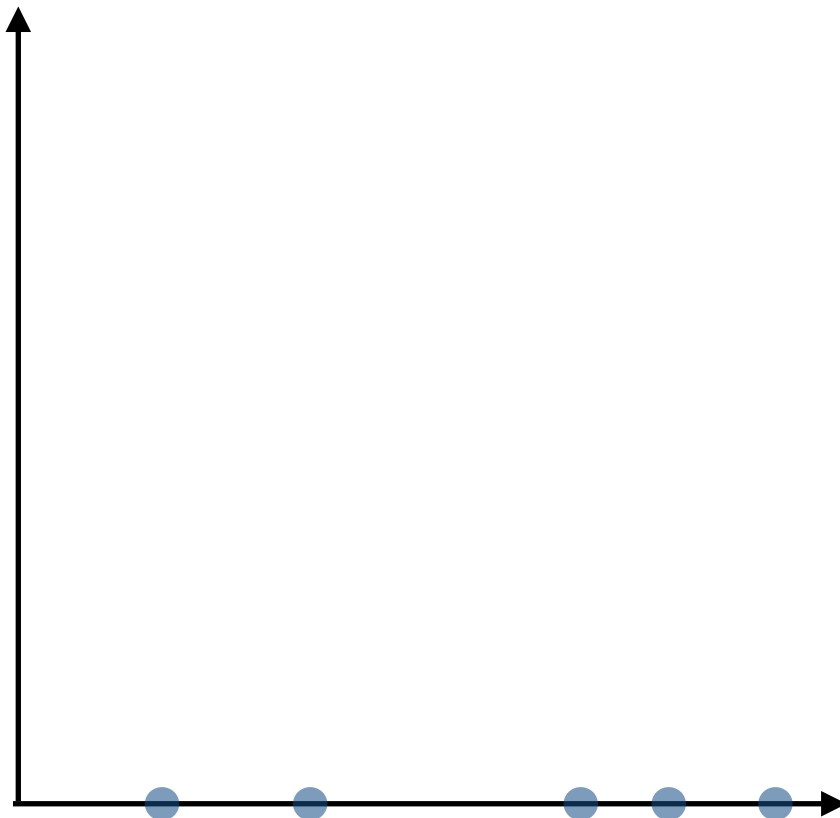
Lower bounds by reduction



- If we can find such a reduction, this proves an $\Omega(n \lg n)$ lower bound for CH

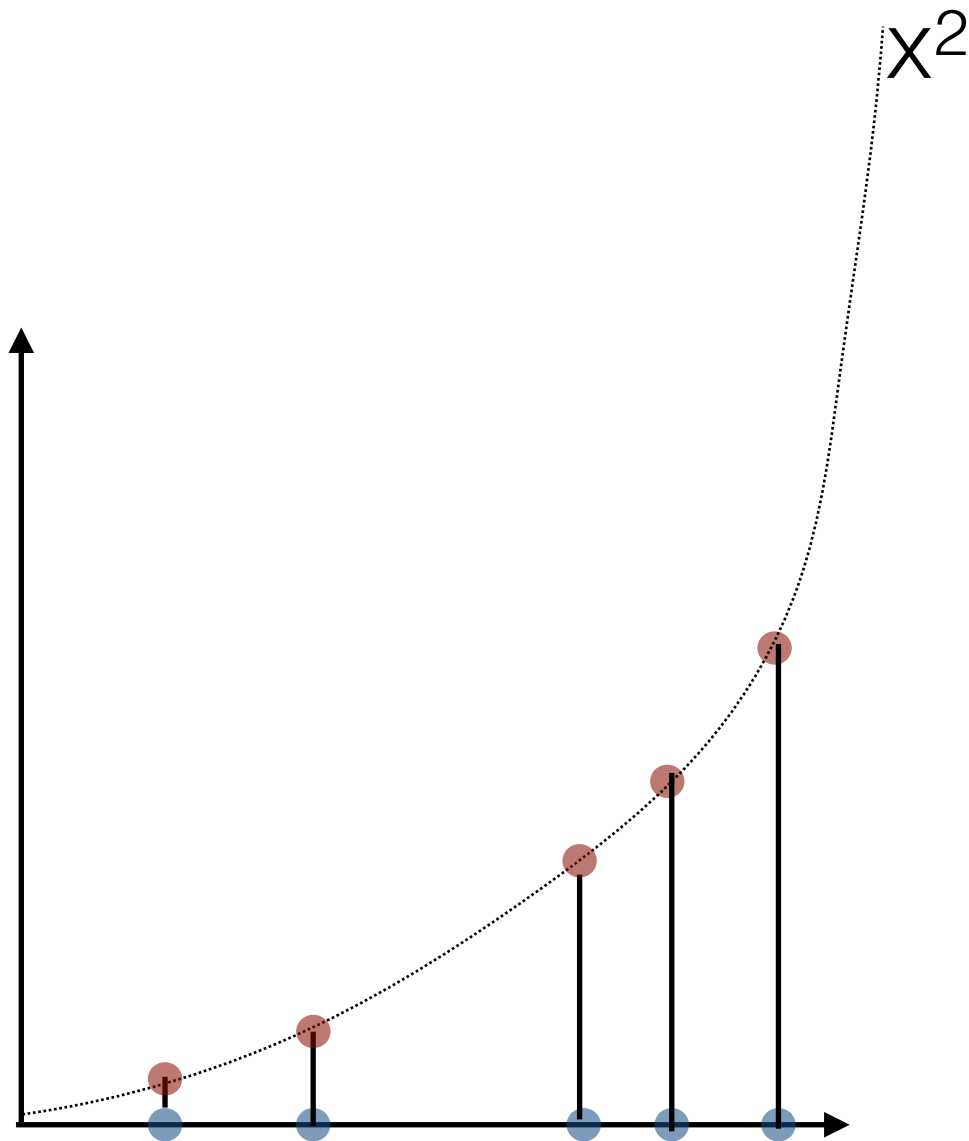
Sorting reduces to CH

- Assume we are given a set of numbers x_1, x_2, \dots, x_n to sort.
- Our goal is to argue that there exists some instance of a convex hull problem that sorts our numbers.



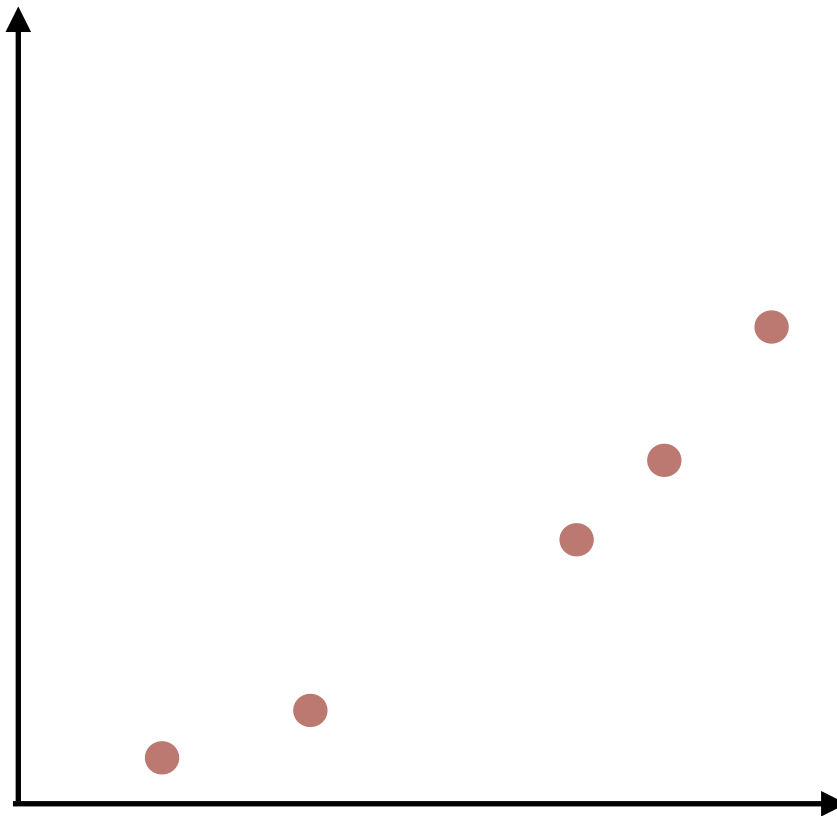
Sorting reduces to CH

- P : set of points x_1, x_2, \dots, x_n
- P' : set of 2D points (x_i, x_i^2) .



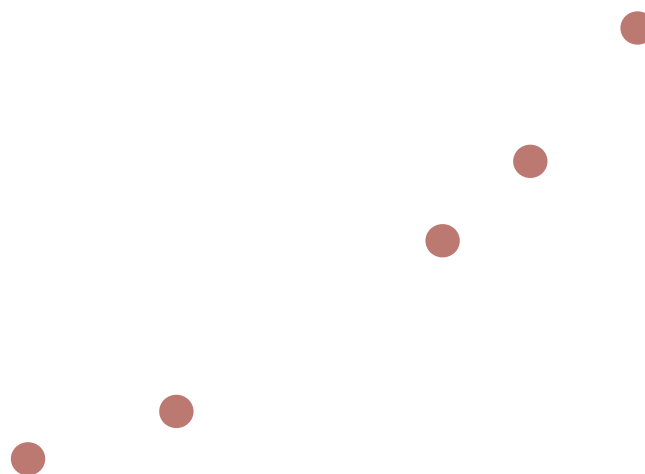
Sorting reduces to CH

- P : set of points x_1, x_2, \dots, x_n
- P' : set of 2D points (x_i, x_i^2) .



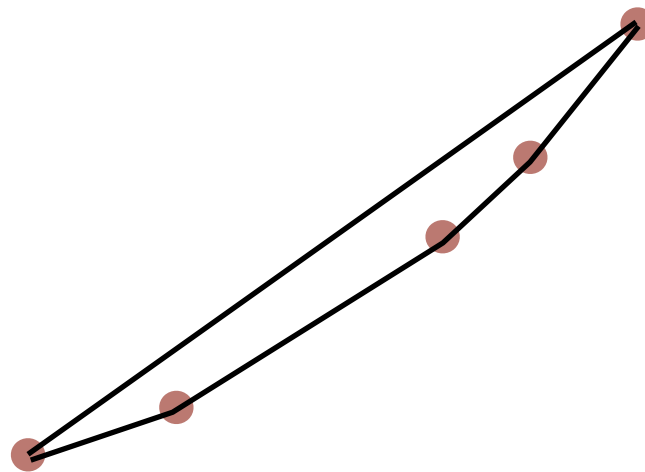
Sorting reduces to CH

- P : set of points x_1, x_2, \dots, x_n
- P' : set of 2D points (x_i, x_i^2) .
- Run $CH(P')$ to find their convex hull



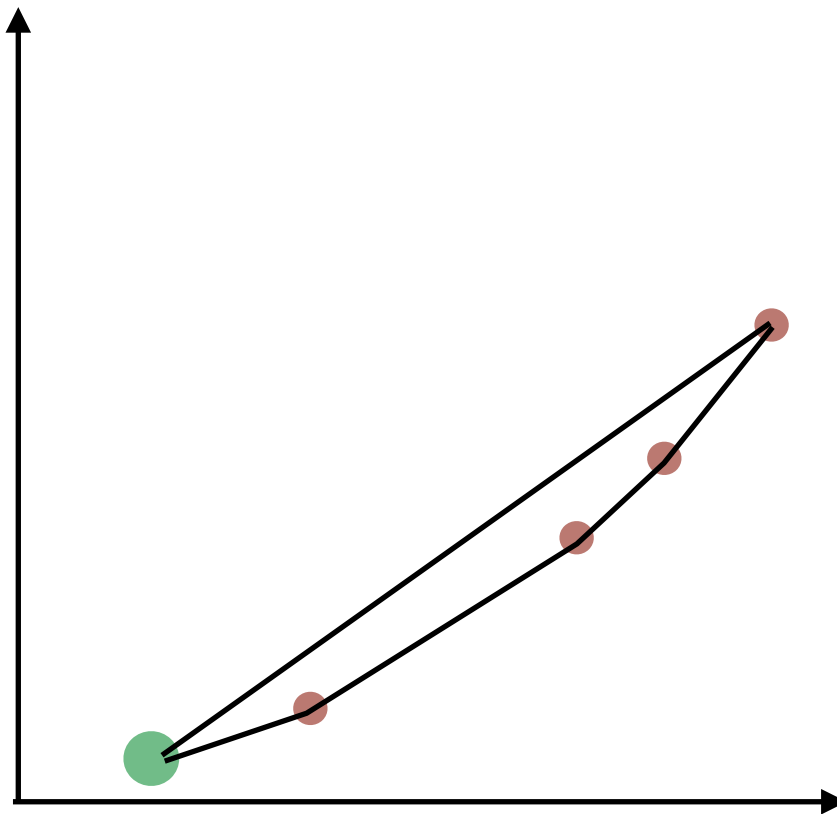
Sorting reduces to CH

- They fall on a parabola, so every point is on the hull



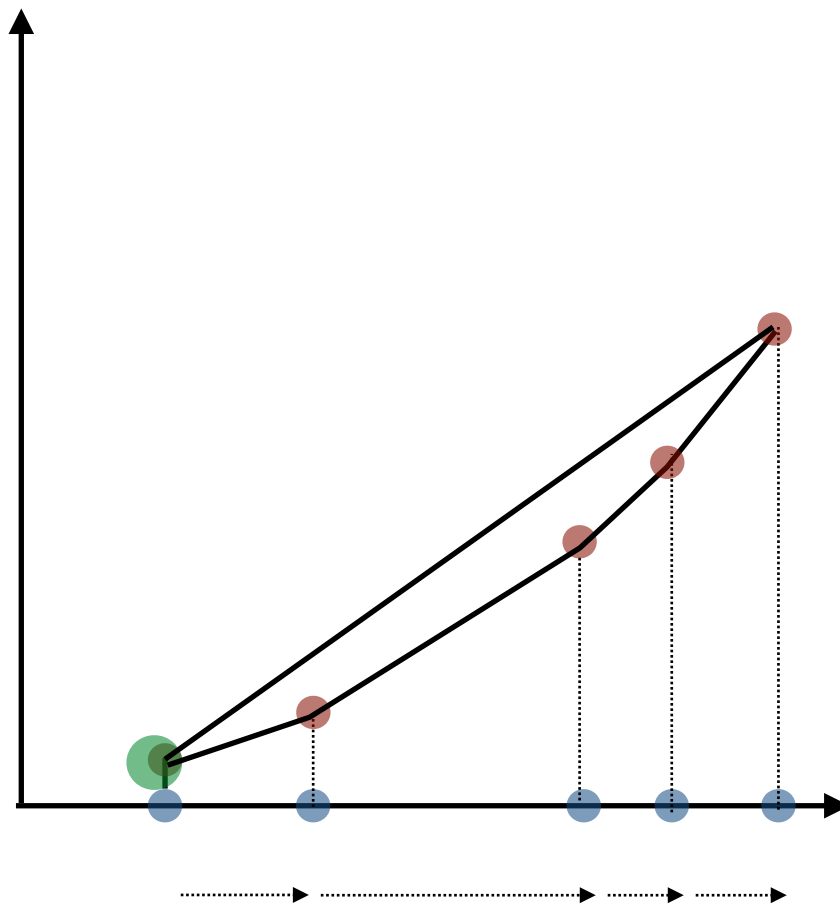
Sorting reduces to CH

- Find the lowest point on the hull, and walk from in ccw order.



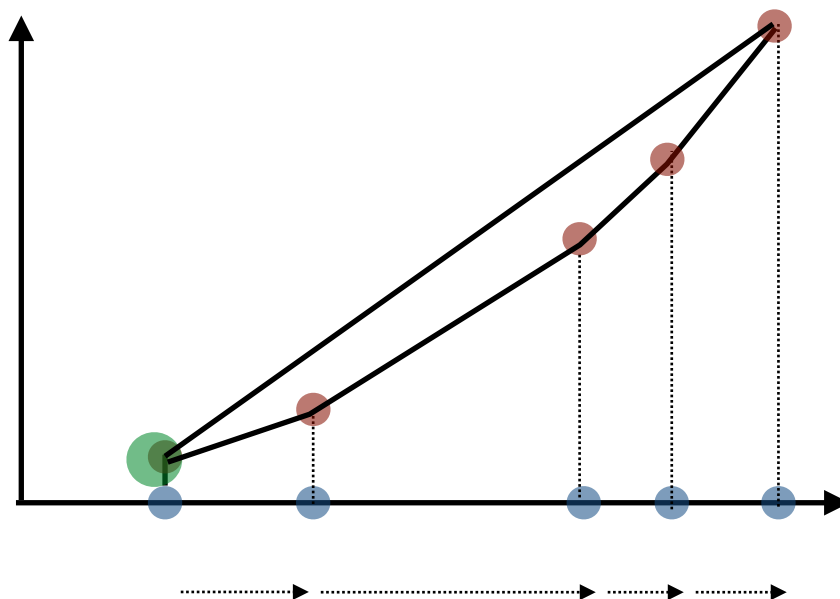
Sorting reduces to CH

- Find the lowest point on the hull, and walk from in ccw order.
- This is sorted order!



Sorting reduces to CH

- Input: set of points x_1, x_2, \dots, x_n
 - Form a set of 2D points (x_i, x_i^2) .
 - Run the CH algorithm to construct their convex hull.
 - Find the lowest point on the hull, and walk from in ccw order. This is sorted order!
- If we could find the CH faster than $n \lg n$, then we could sort faster than $n \lg n$! Impossible!



Sorting reduces to CH

- What we actually proved is that
 - Any CH algorithm **that produces the boundary in order** must take $\Omega(n \lg n)$ in the worst case.
- If we did not want the boundary in order, can the CH be constructed faster?
 - It was an open problem for a while
 - Finally, it was established that a convex hull algorithm, even if it does not produce the boundary in order, still needs $\Omega(n \lg n)$

- Yes, Graham scan is the ultimate CH algorithm but...
 - not output sensitive
 - does not extend to 3D
- The (re)search continues

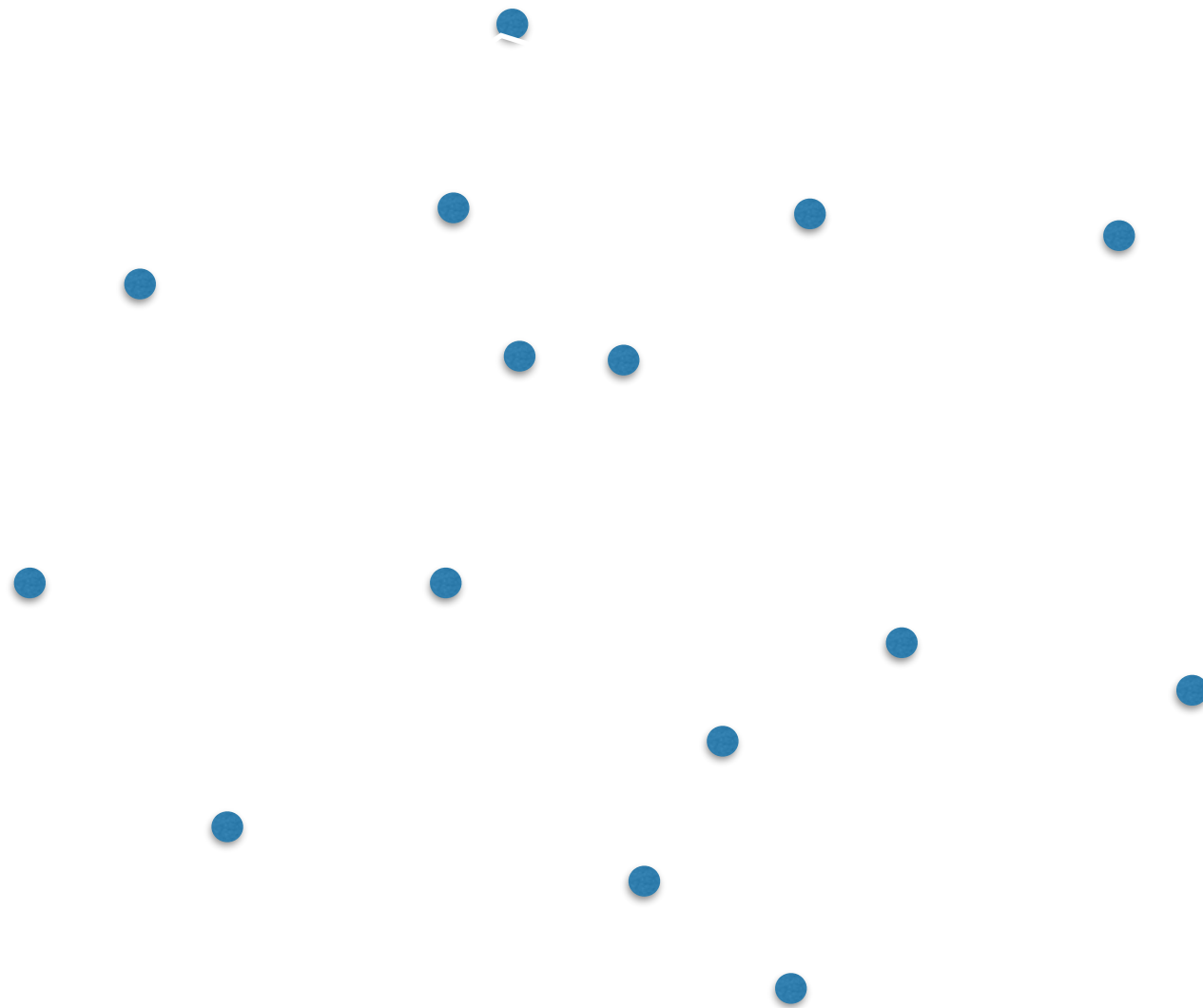
An incremental algorithm for CH

Incremental algorithms

- Goal: solve problem P
- Idea: traverse points one at a time and solve the problem for points seen so far
- Incremental Algorithm
 - initialize solution $S = \text{initial solution}$
 - for $i=1$ to n
 - S represents solution of p_1, \dots, p_{i-1}
 - update S to represent solution of p_1, \dots, p_{i-1}, p_i

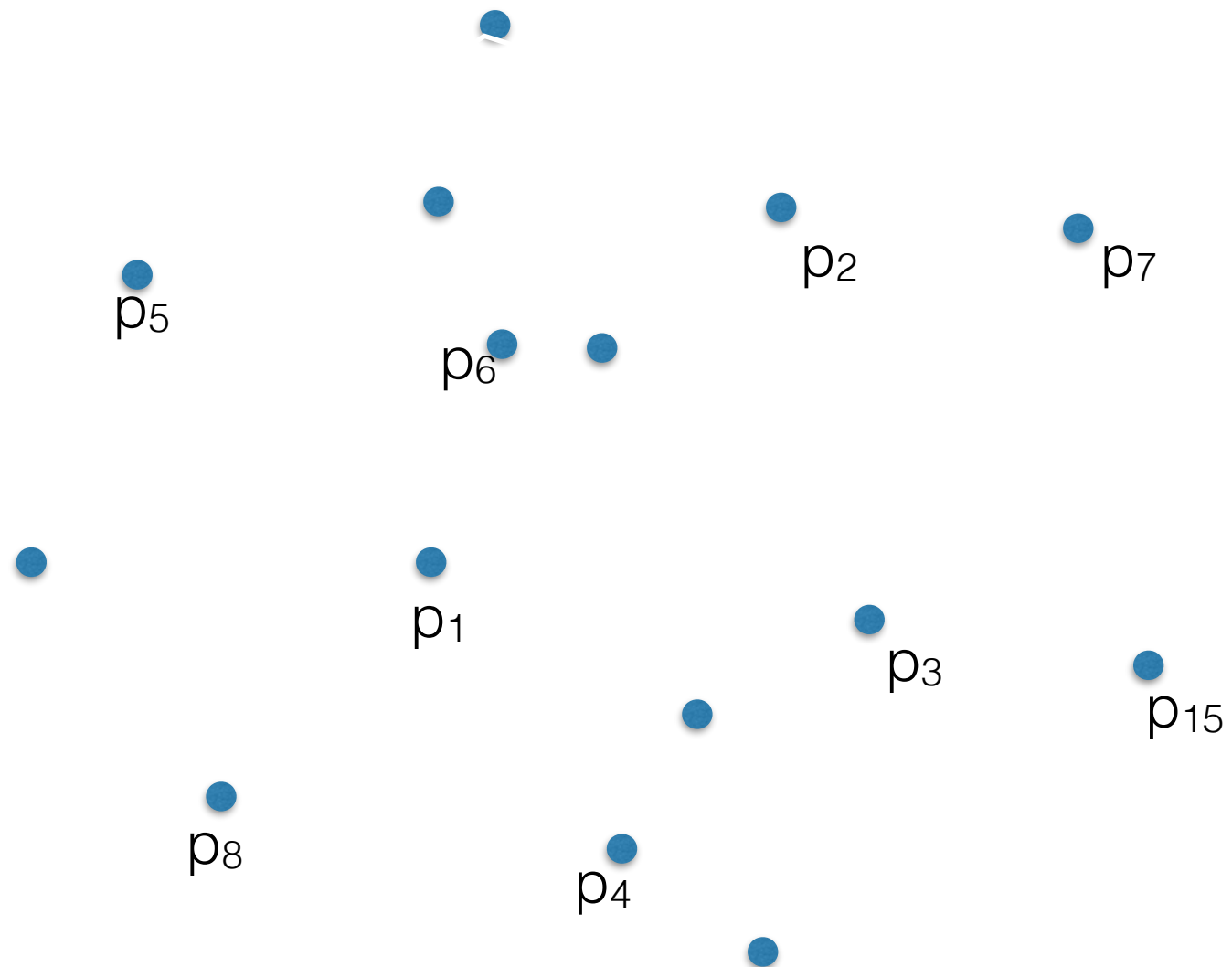
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



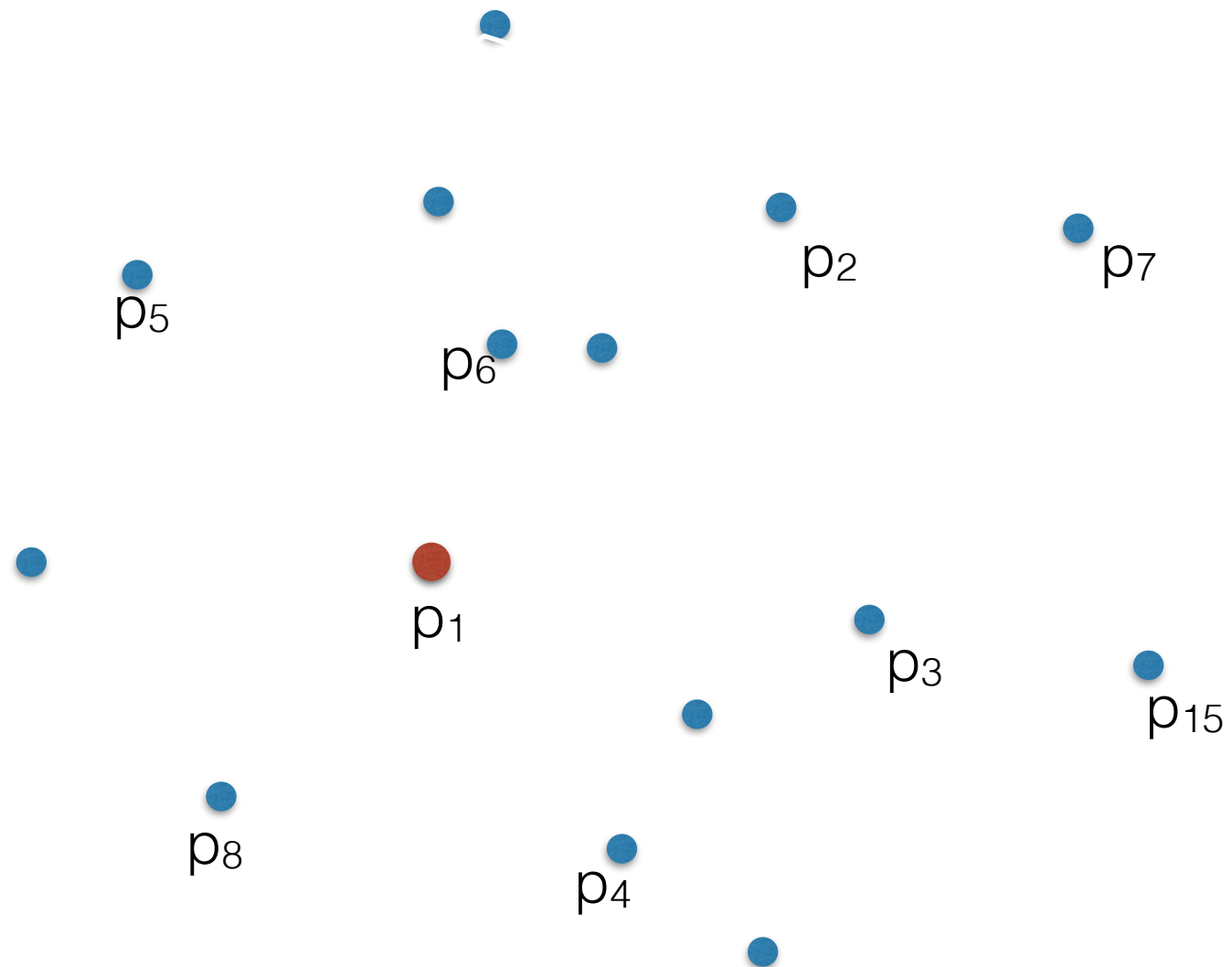
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



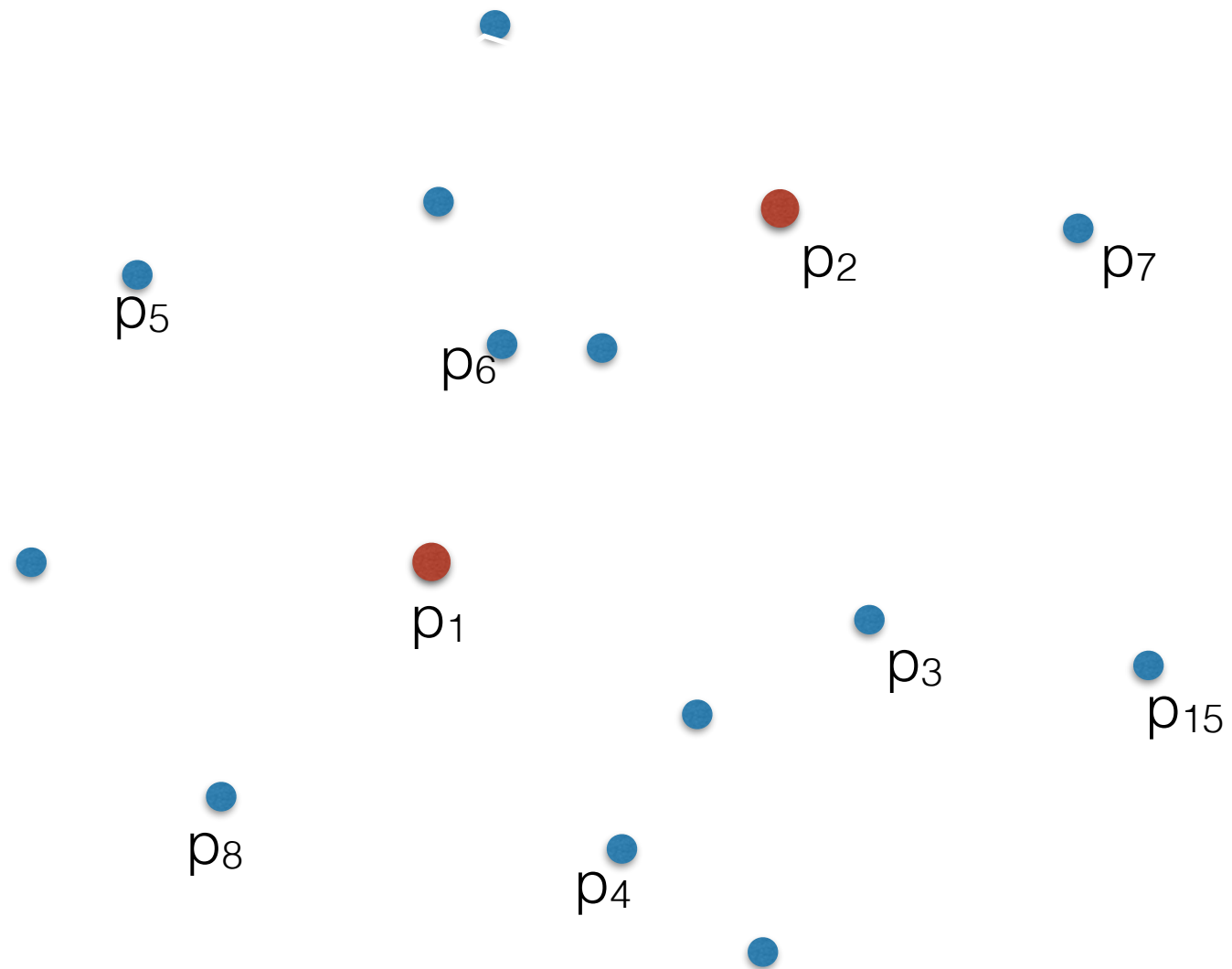
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



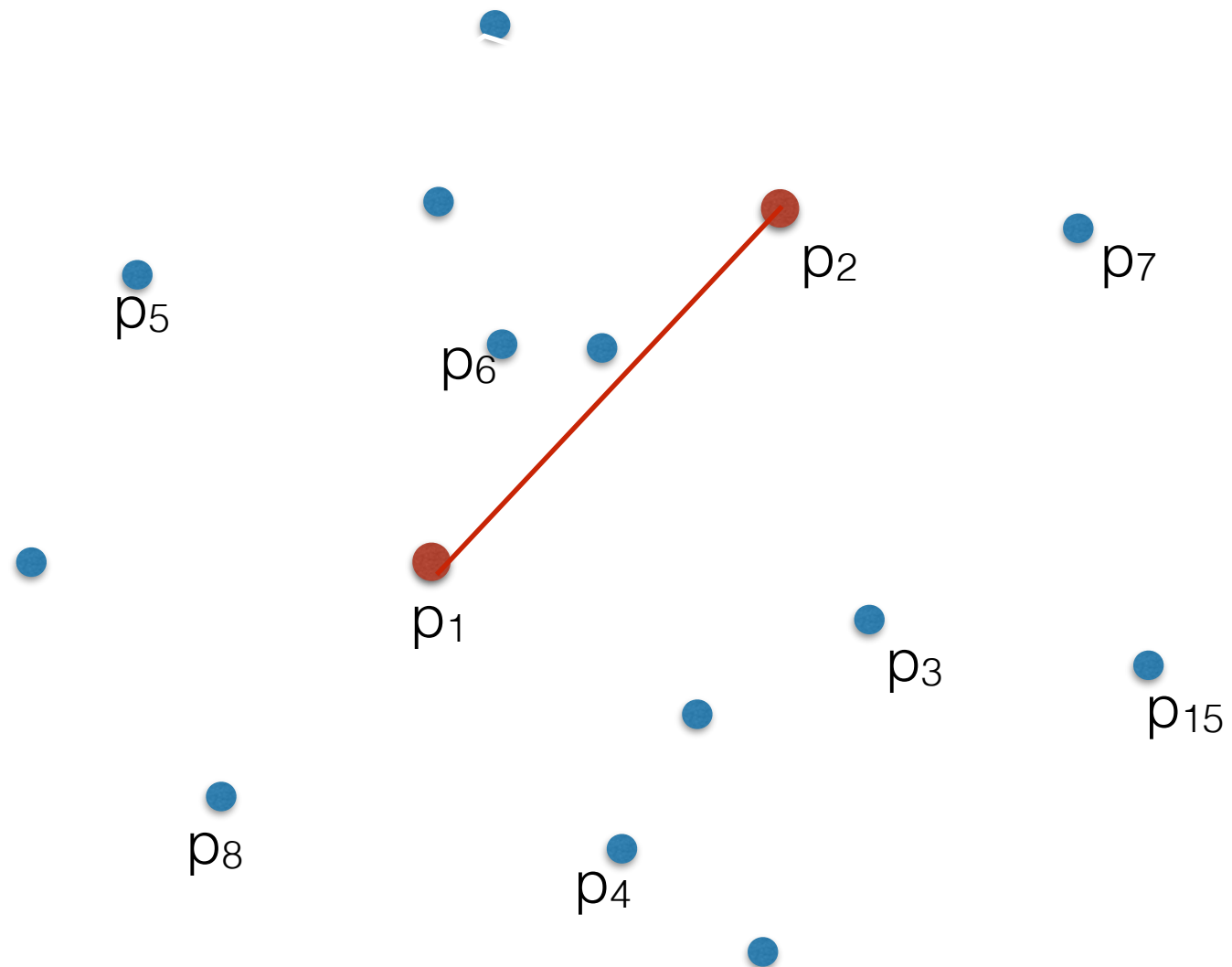
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



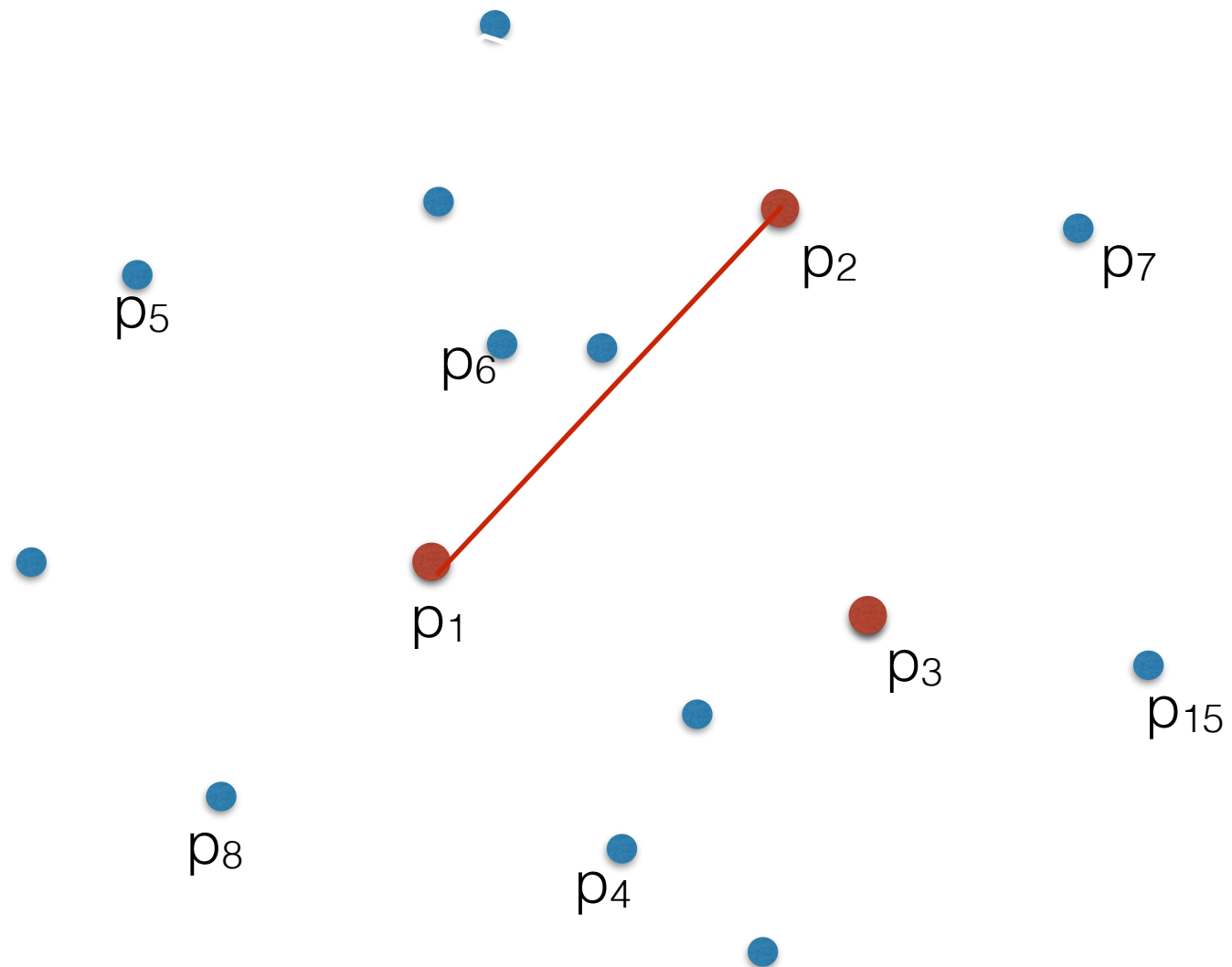
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - $//CH$ represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



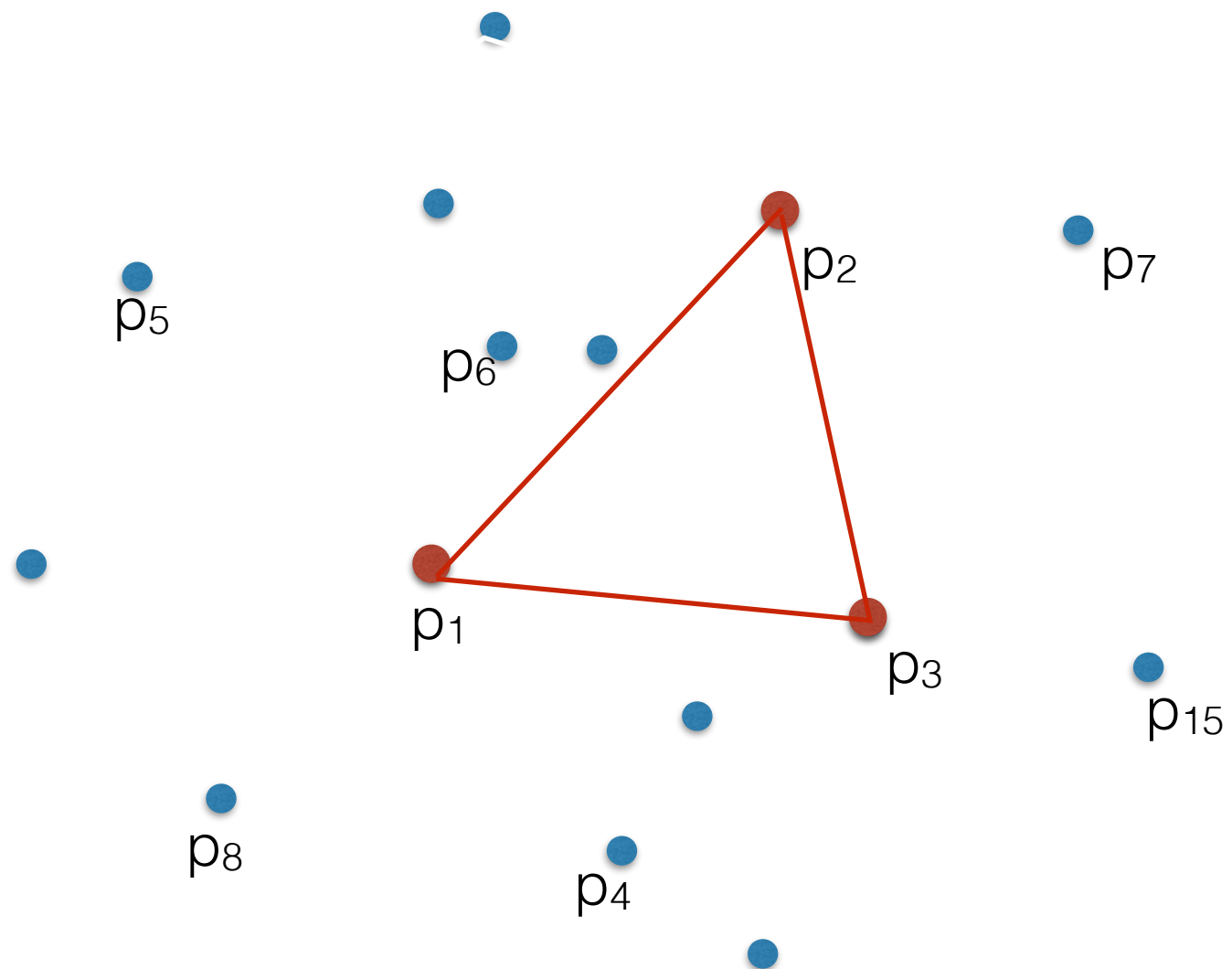
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



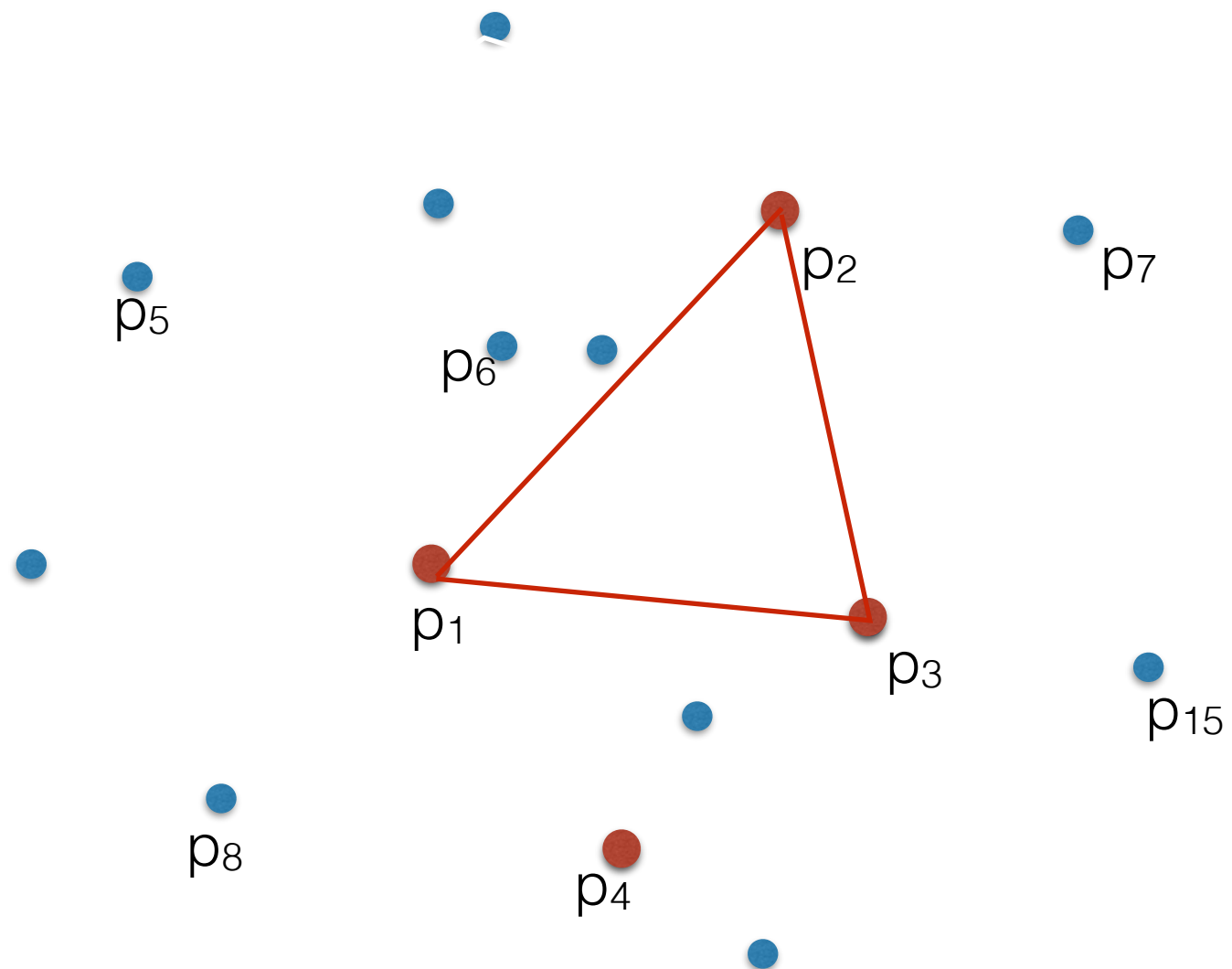
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - $//CH$ represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



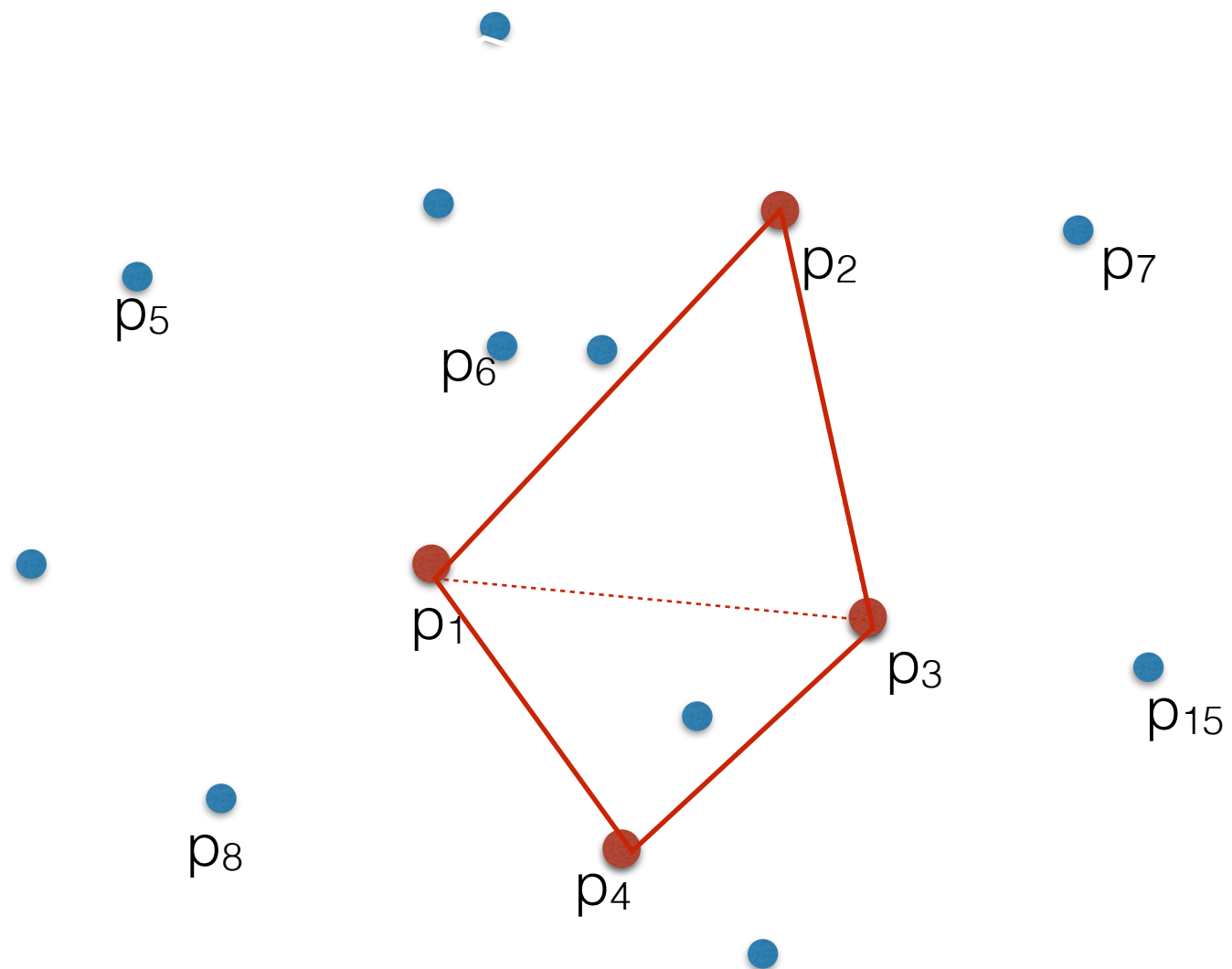
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - $//CH$ represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



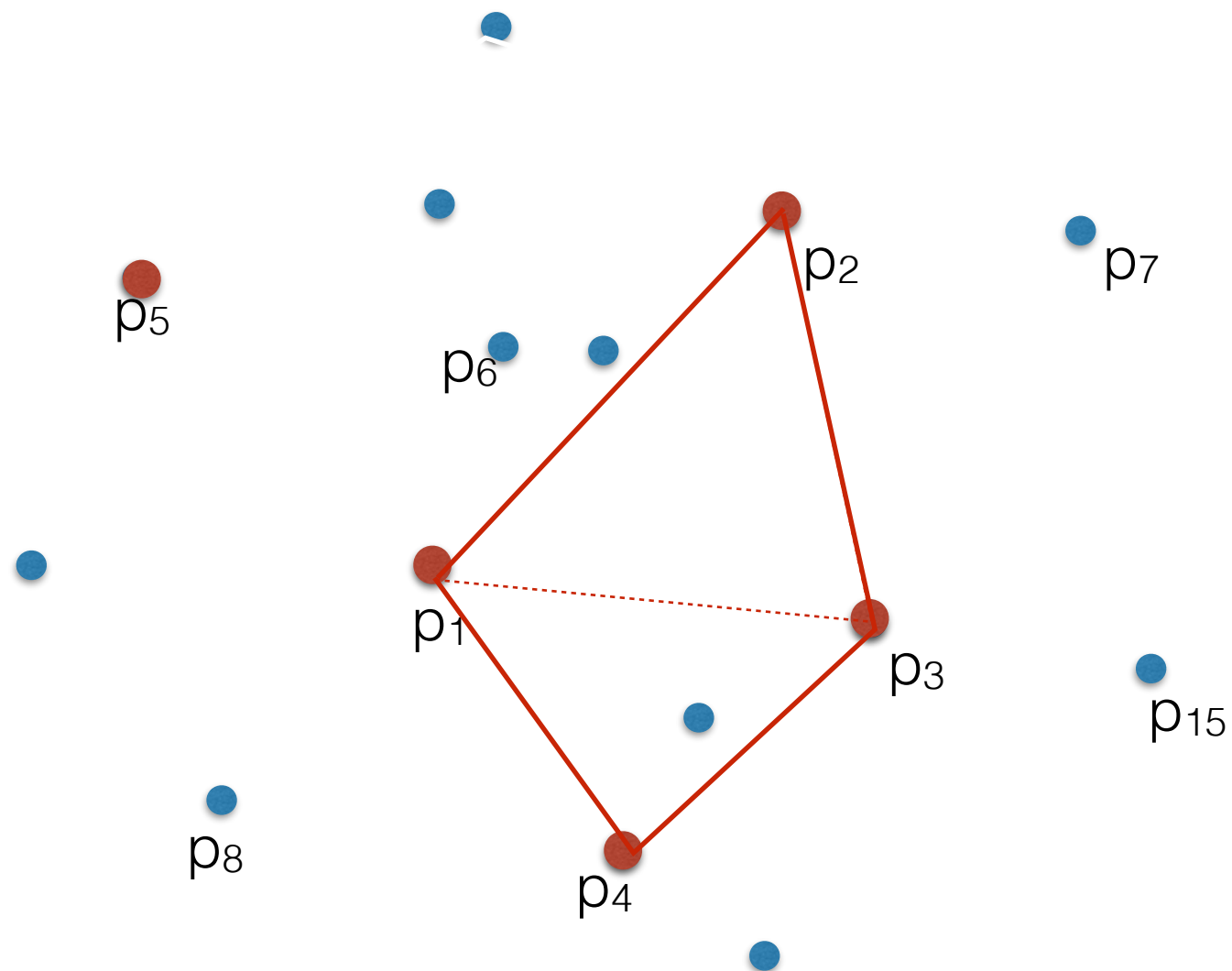
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



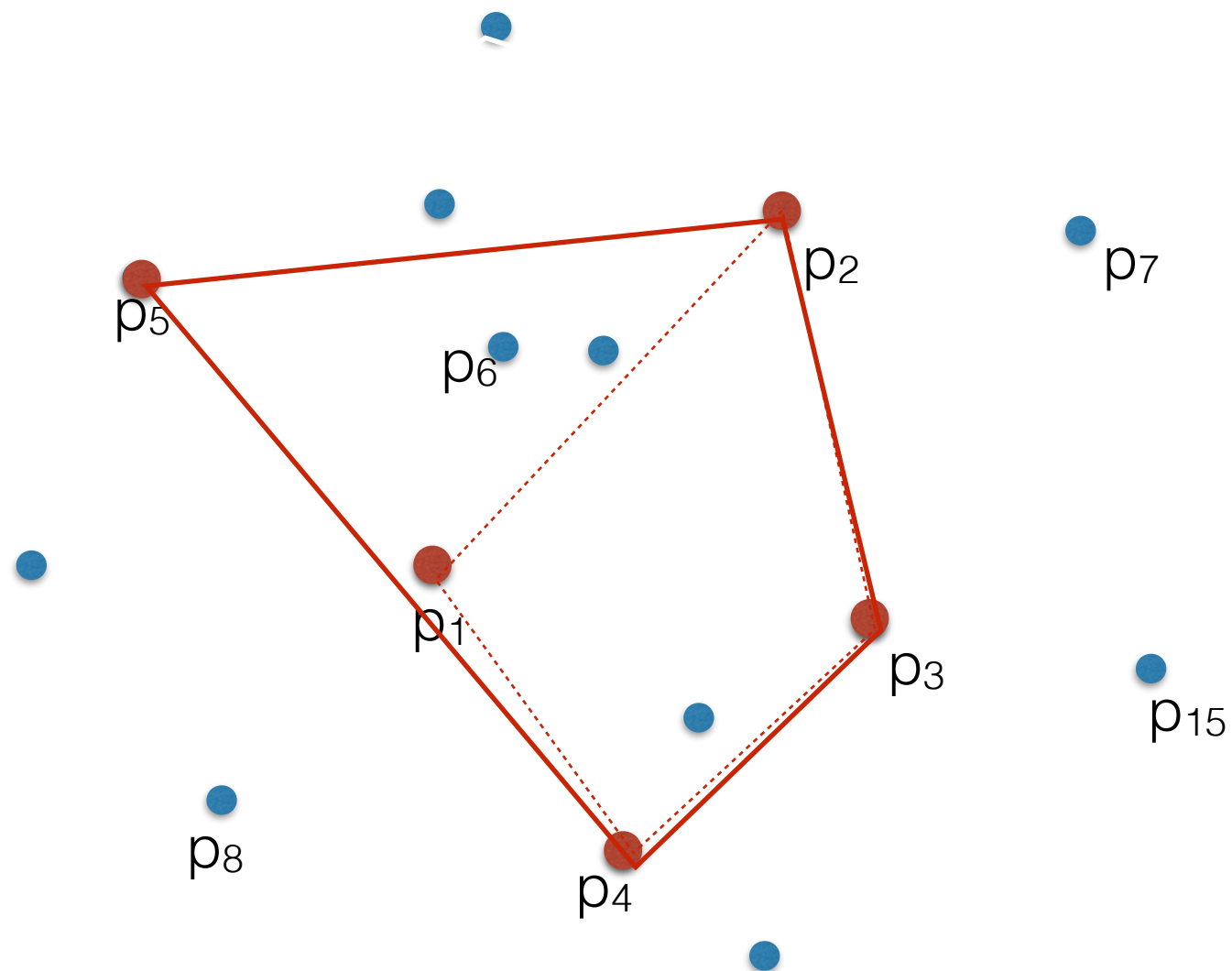
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - $//CH$ represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



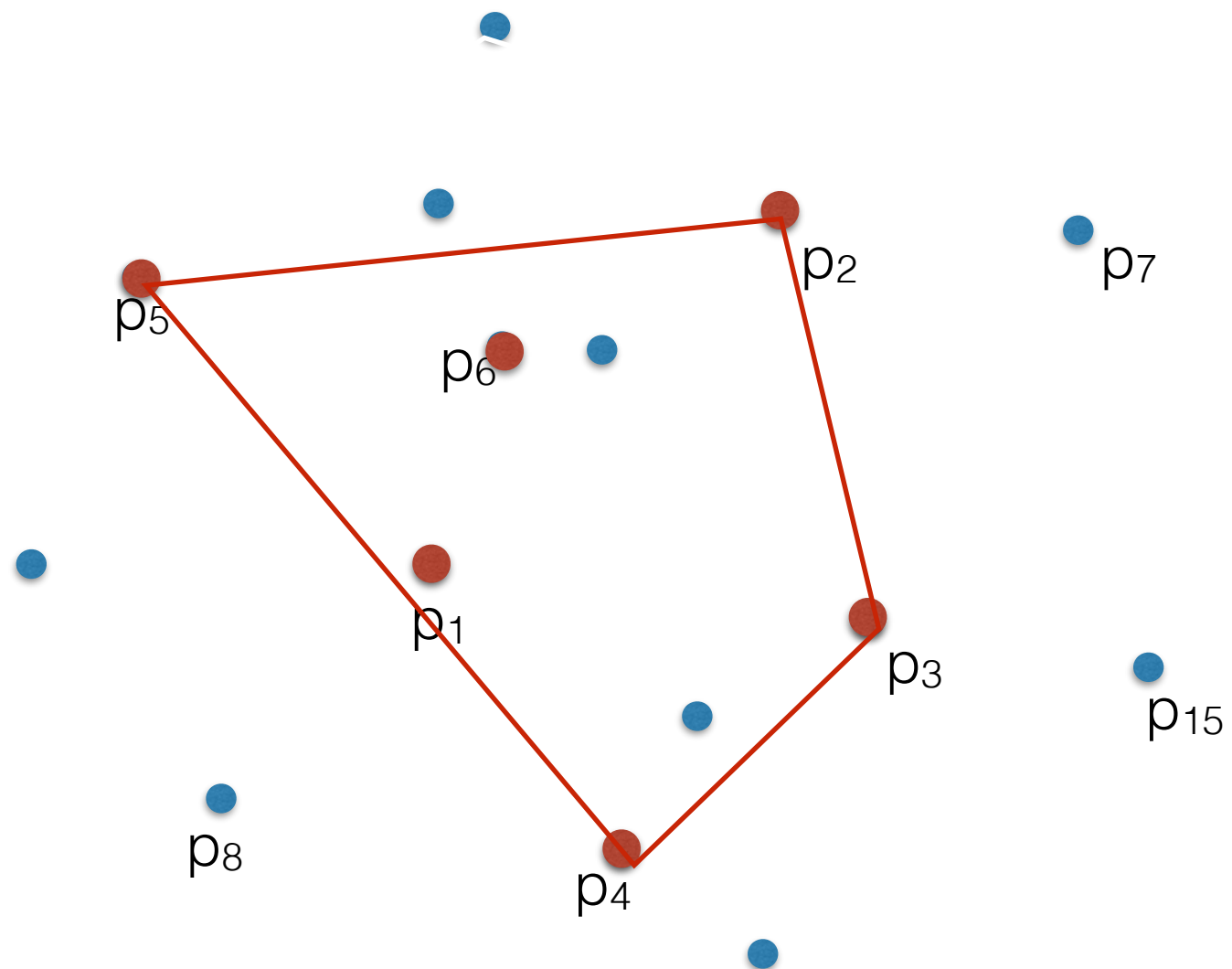
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



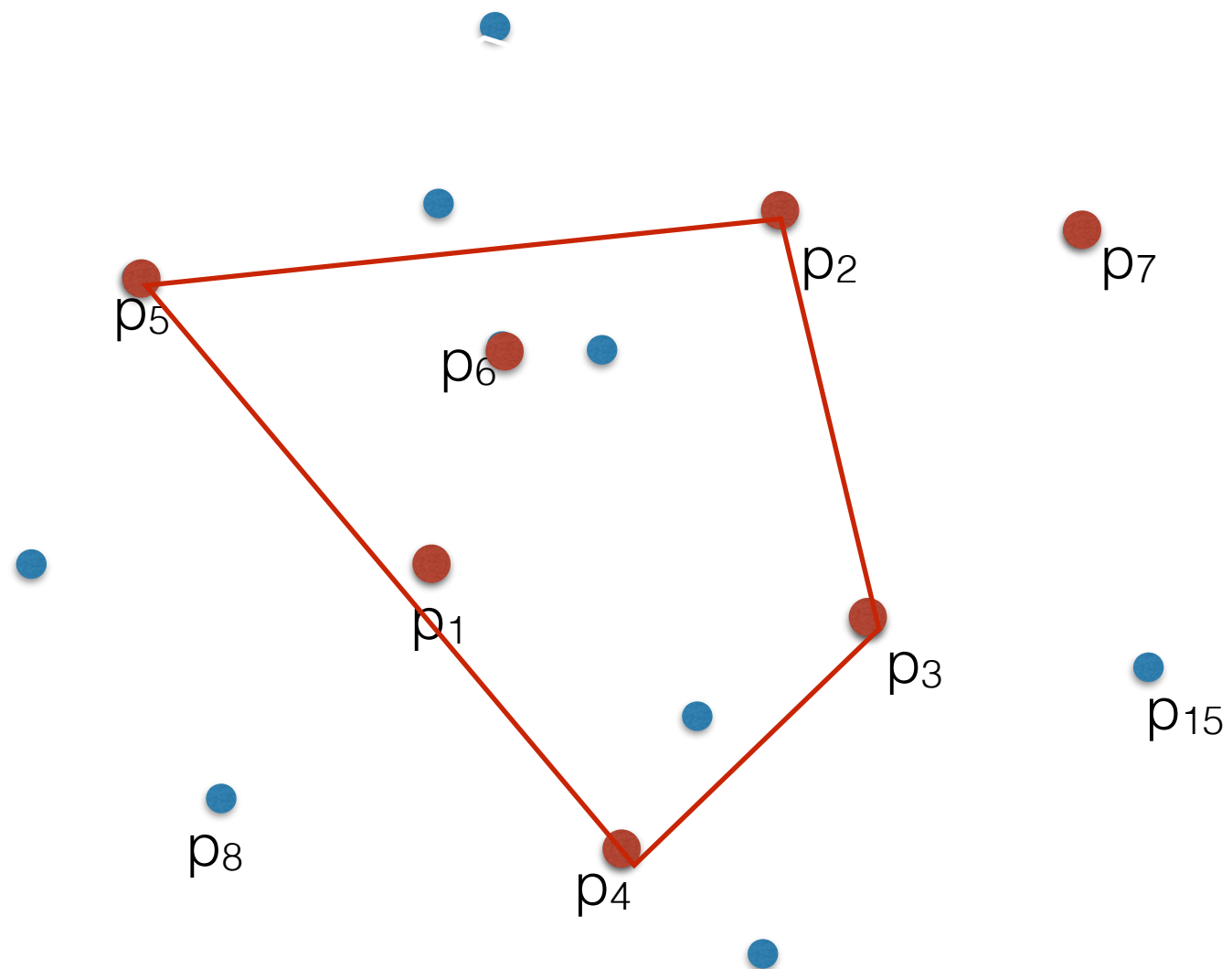
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - $//CH$ represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



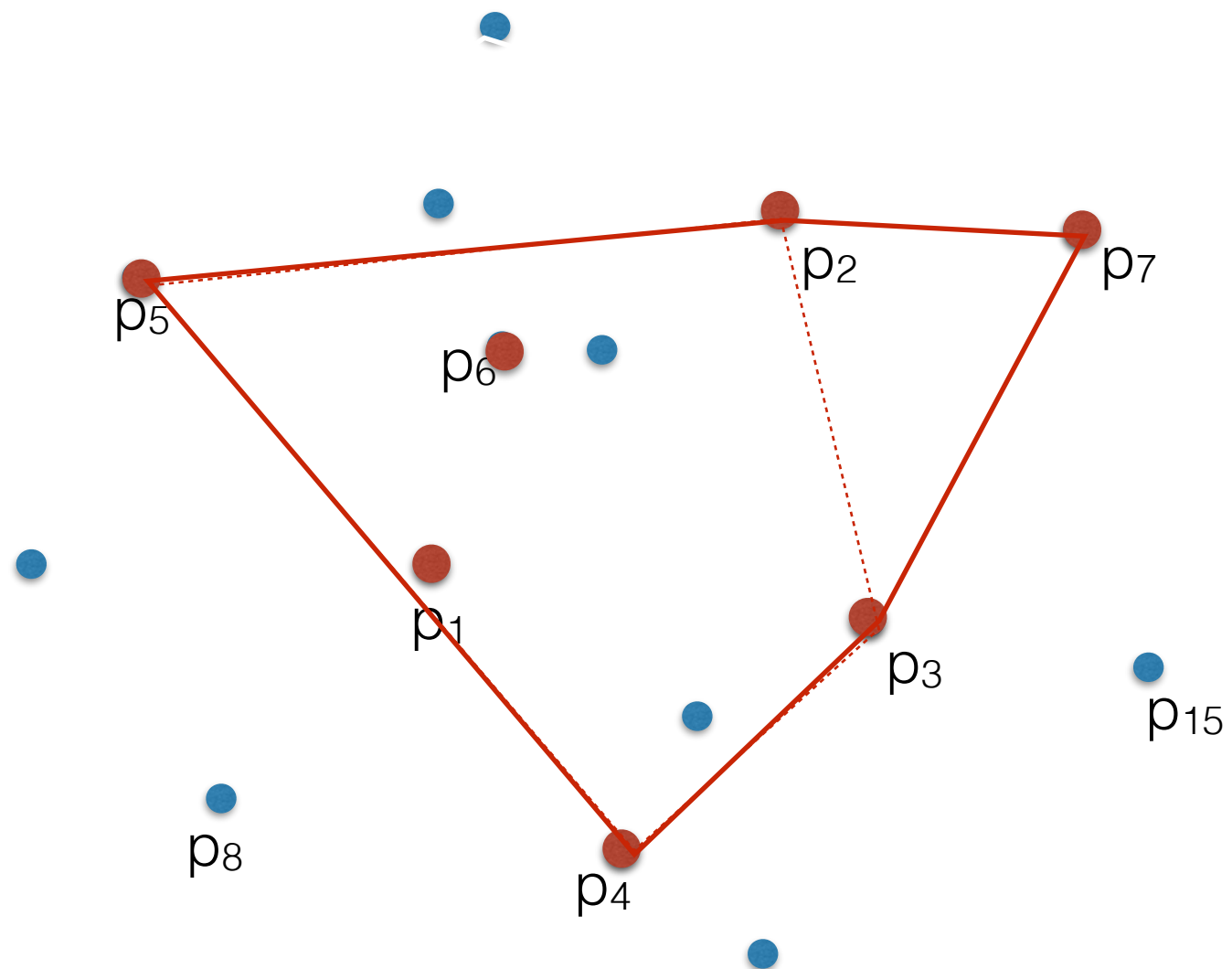
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



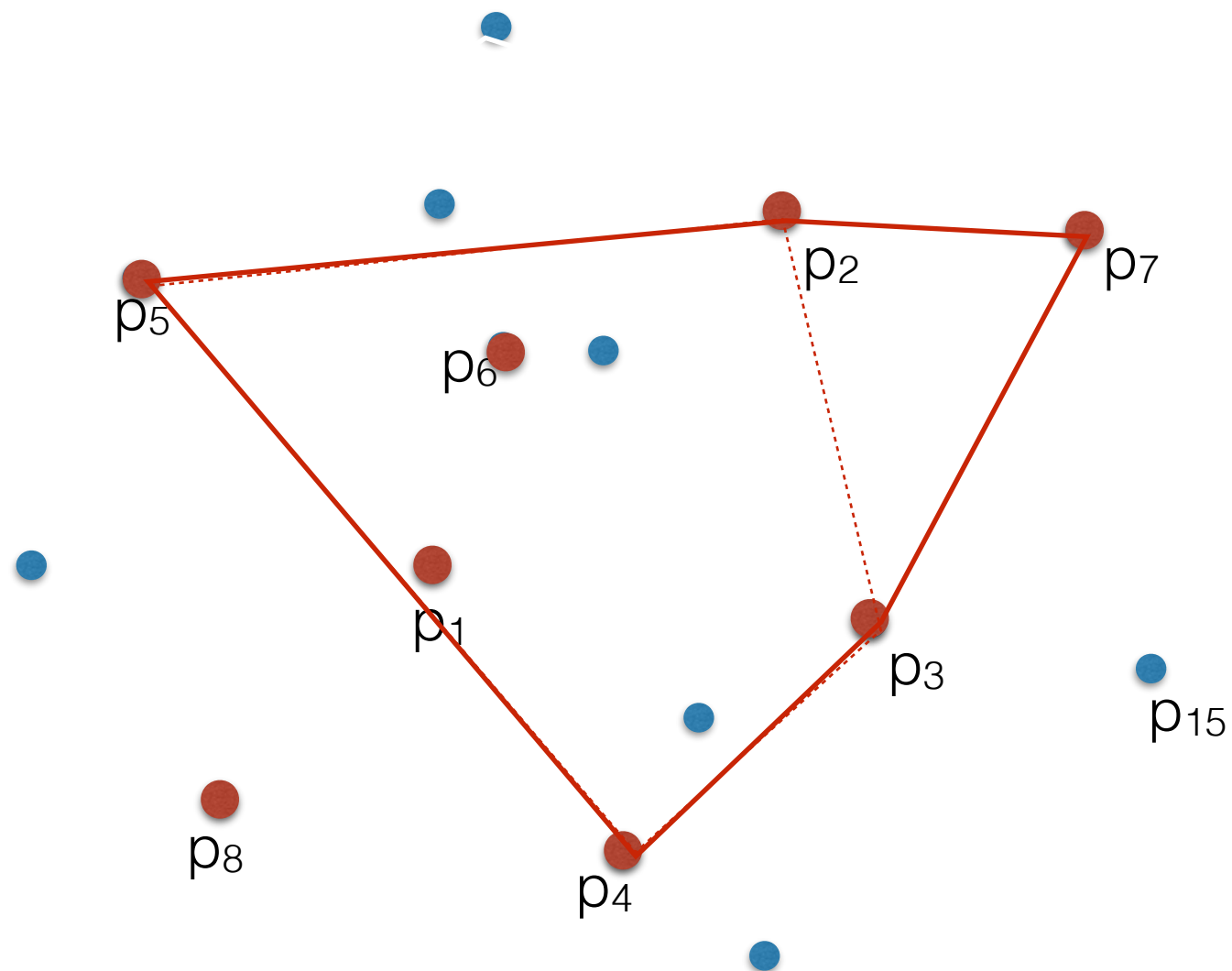
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



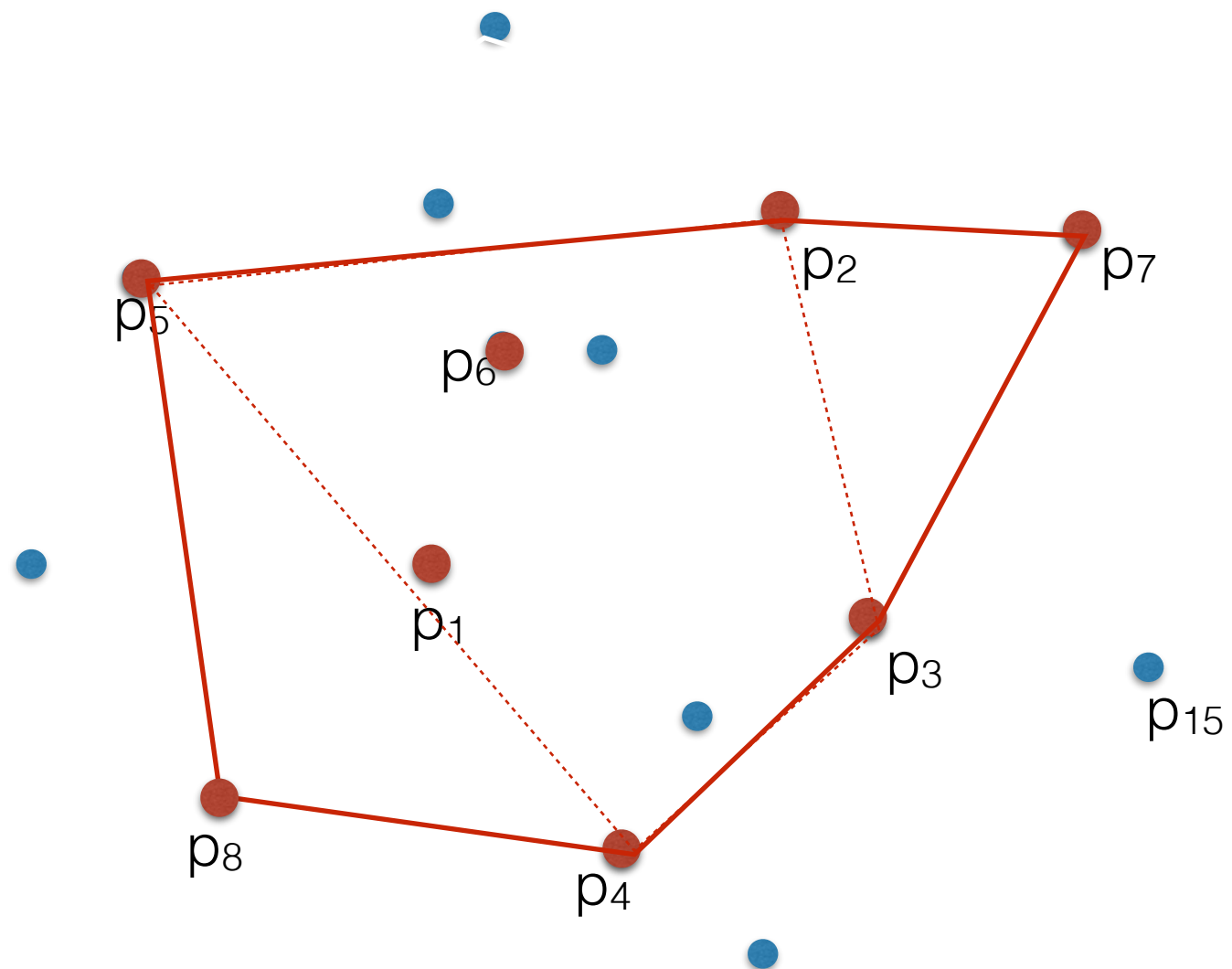
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - $//CH$ represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



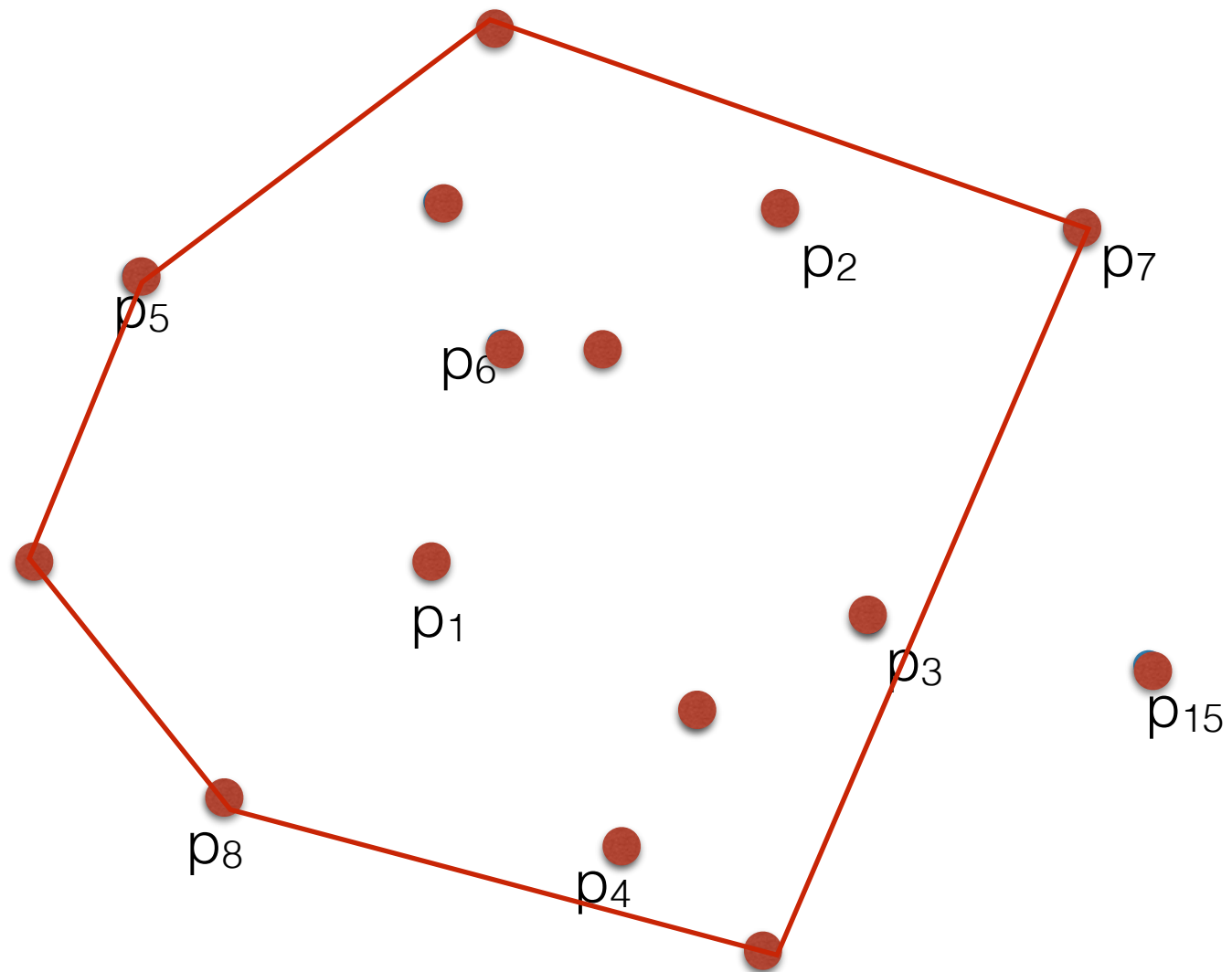
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



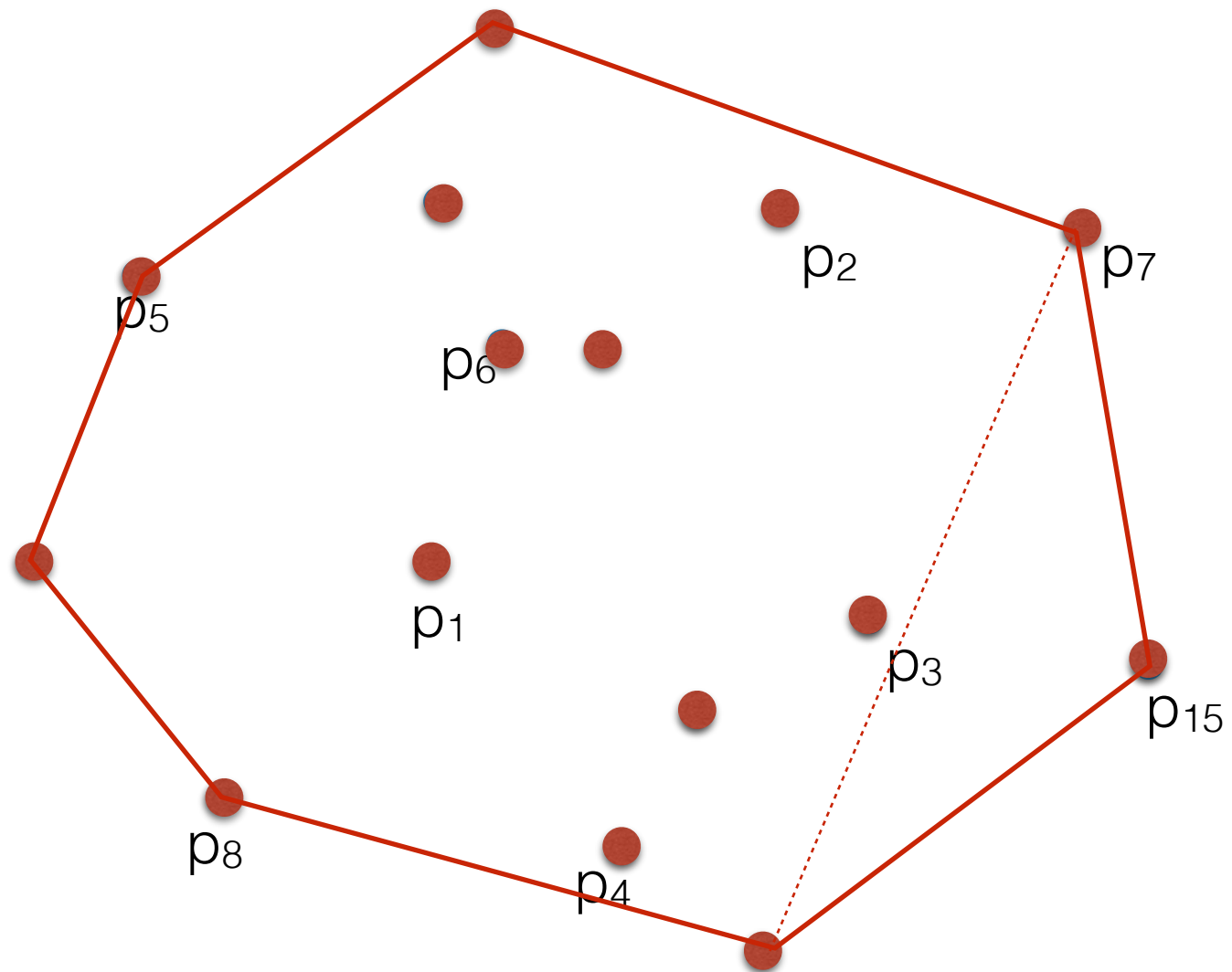
Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - $//CH$ represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



Incremental algo for CH

- $CH = \{\}$
- for $i=1$ to n
 - $//CH$ represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$



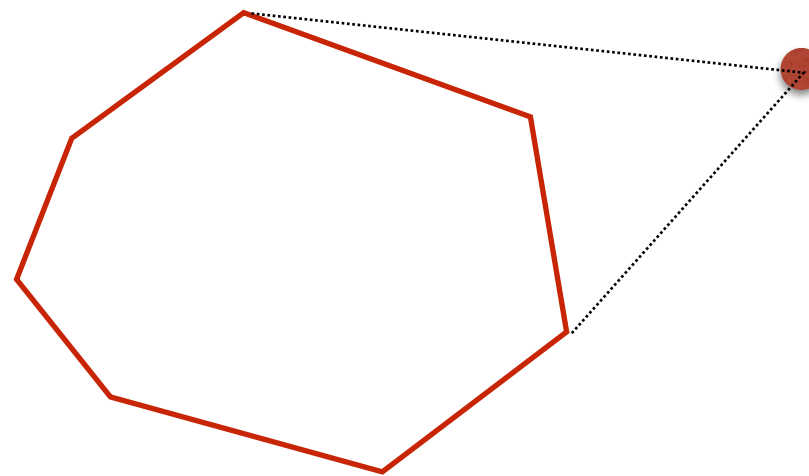
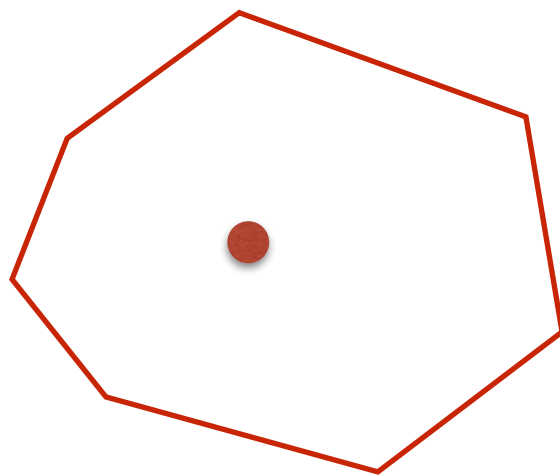
Incremental algo for CH

- $CH = \{\}$
 - for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$
-
- The basic operation is adding a point to a convex polygon
 - How many cases?
 - How to handle each case?
-
- Class work: Pick a set of point, Simulate the approach and try to answer these questions.

Incremental algo for CH

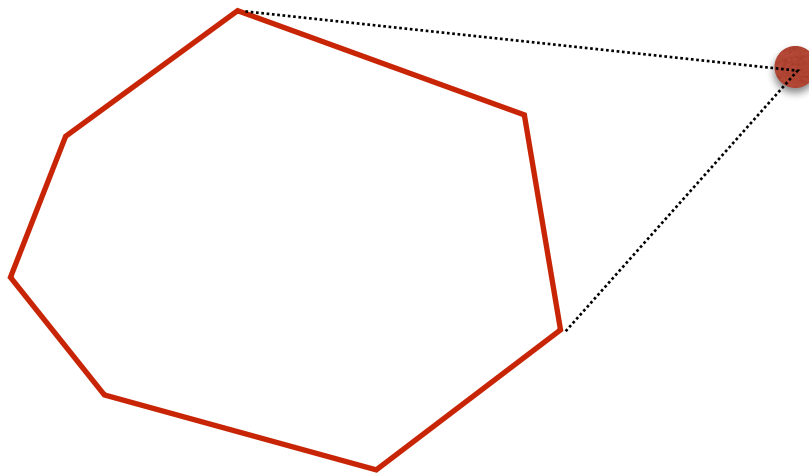
- $CH = \{\}$
- for $i=1$ to n
 - //CH represents the CH of $p_1..p_{i-1}$
 - update CH to represent the CH of $p_1..p_i$

- The basic operation is adding a point to a convex polygon
 - CASE 1: p is in polygon
 - CASE 2: p outside polygon



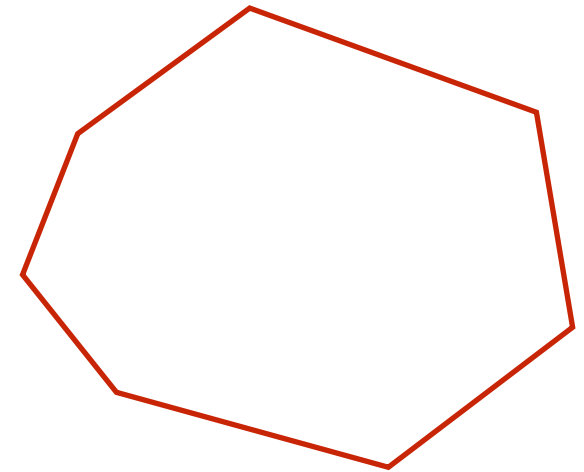
Incremental algo for CH

- Issues to solve
 - What's a good representation for a polygon?
 - We need a point-in-polygon test ?
 - How to handle CASE 2 ?



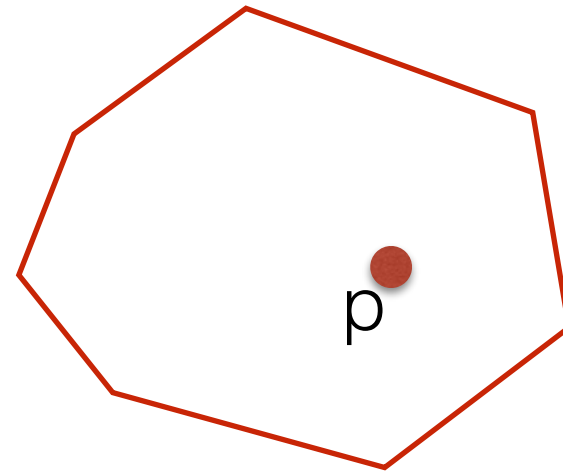
Representing a polygon

A polygon is represented as a list of vertices in boundary order.
(the convention is counter-clockwise order)



```
typedef struct _polygon{  
    int k; //number of vertices  
    Point* vertices; //the vertices, ccw in boundary order  
} Polygon;  
  
or  
  
Vector<Point> //note: the vertices, ccw in boundary order
```

Point in convex polygon

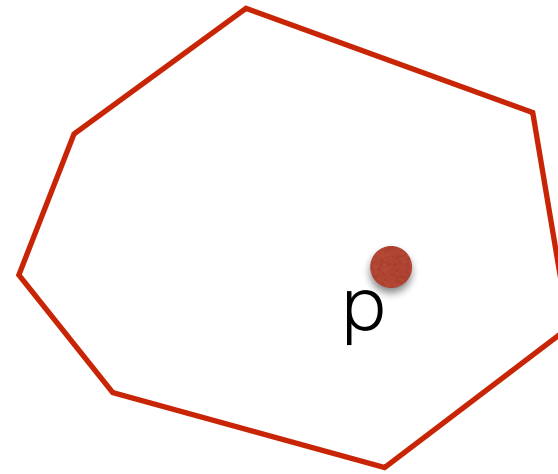


//return TRUE iff p on the boundary or inside H; H is convex a polygon

bool point_in_polygon(point p, polygon H)

What has to be true in order for p to be inside?

Point in convex polygon



//return TRUE iff p on the boundary or inside H; H is convex a polygon

bool point_in_polygon(point p, polygon H)

//p is inside if and only if it is on or to the left of all edges, oriented ccw

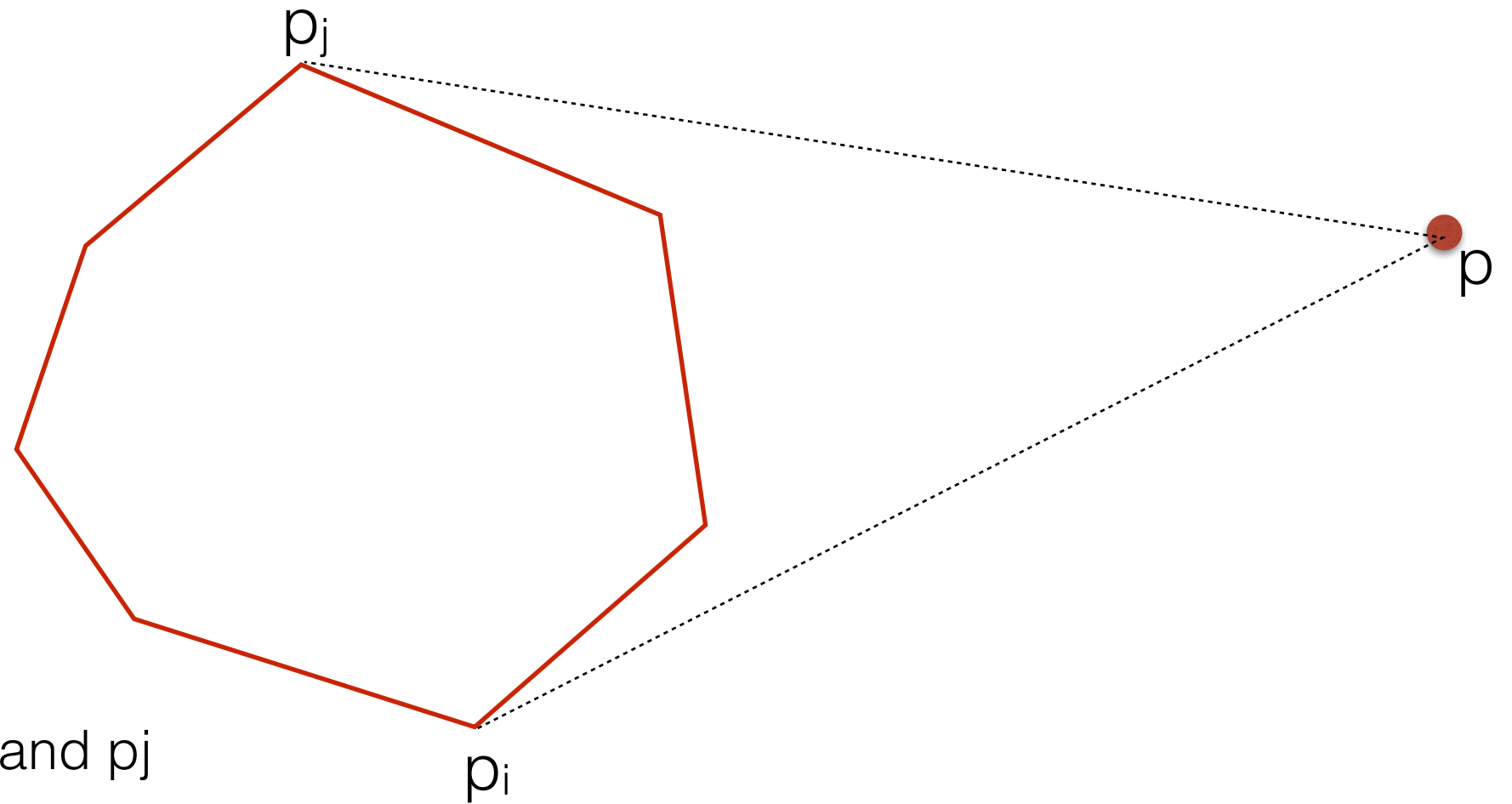
//note: this is NOT true for a non-convex polygon — can you show a

//counter-example?

Analysis:

- $O(k)$ where k is the size of the polygon

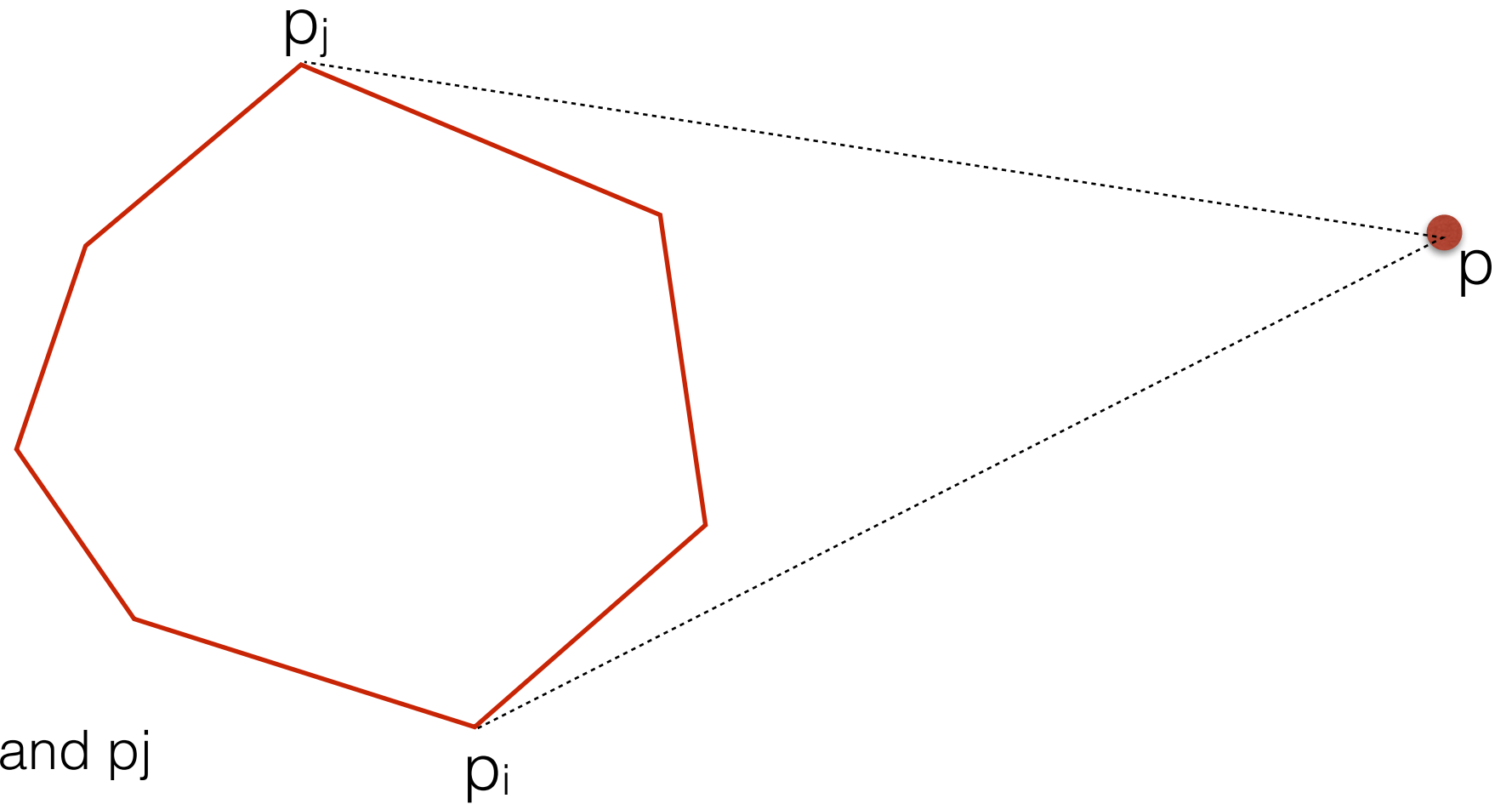
Case 2:



We want to find p_i and p_j

IDEAS?

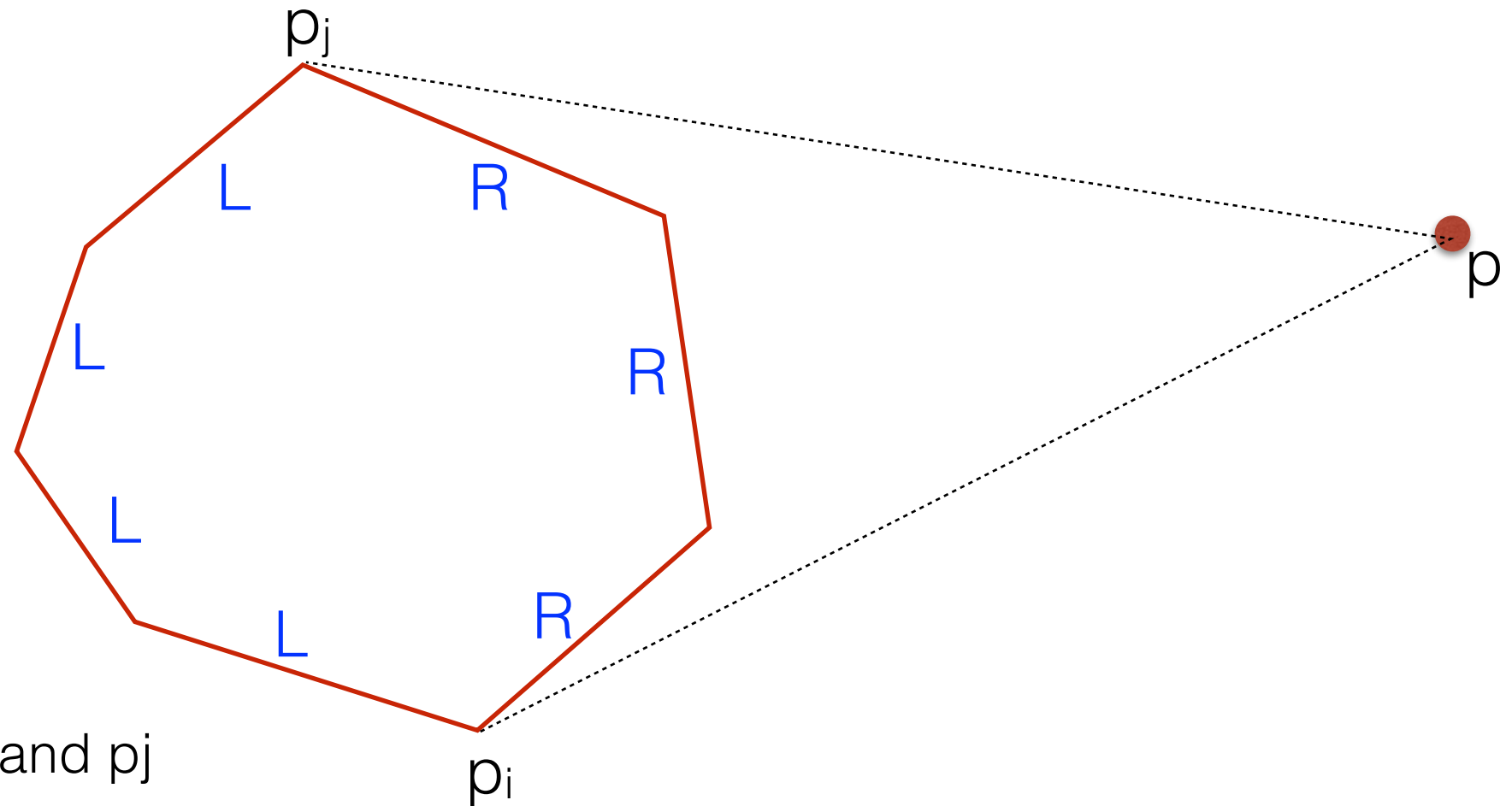
Case 2:



We want to find p_i and p_j

Hint: Check the orientation of p wrt the edges of the polygon.

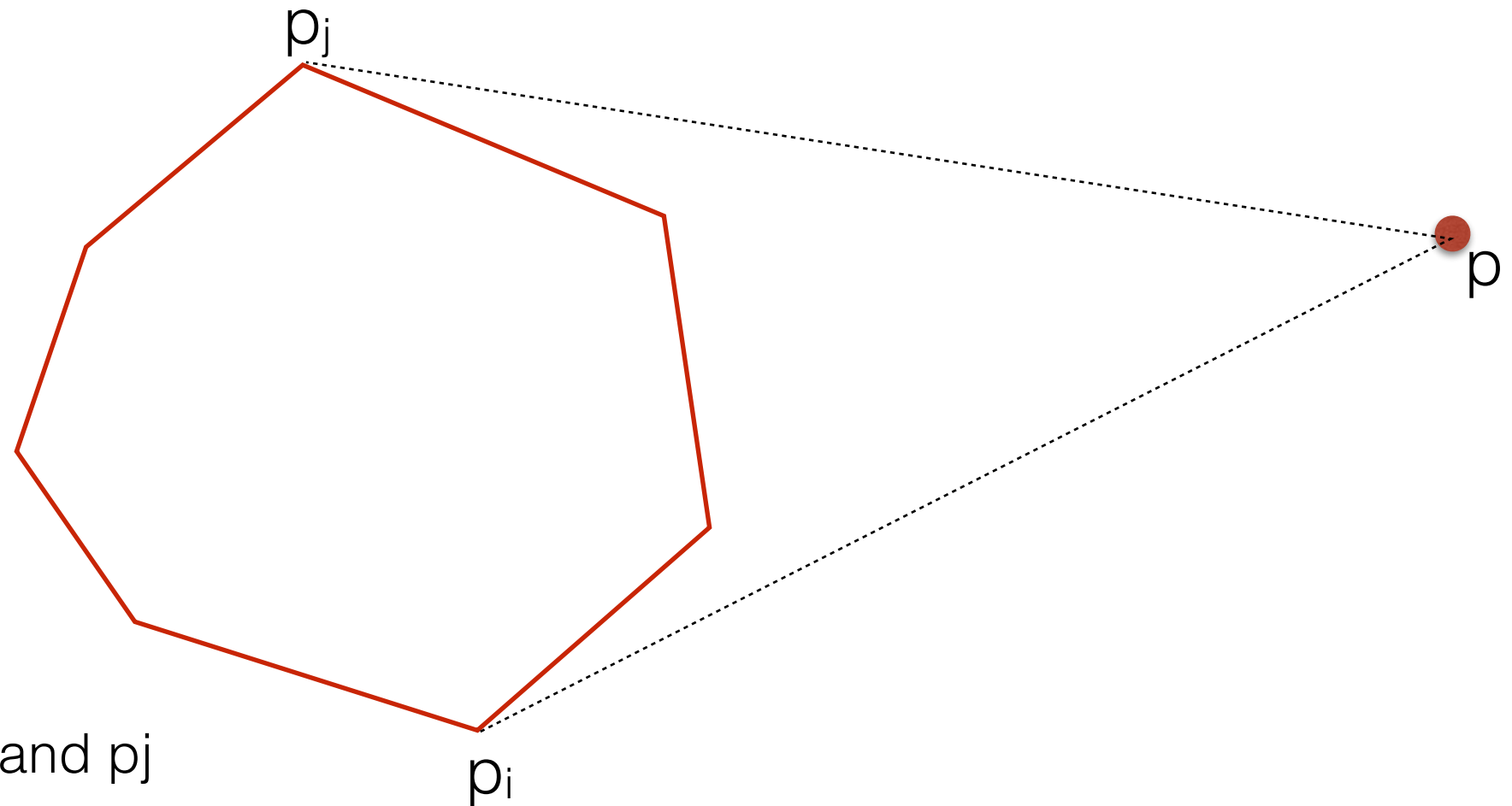
Case 2:



We want to find p_i and p_j

Hint: Check the orientation of p wrt the edges of the polygon.

Case 2:



We want to find p_i and p_j

Hint: Check the orientation of p wrt the edges of the polygon.

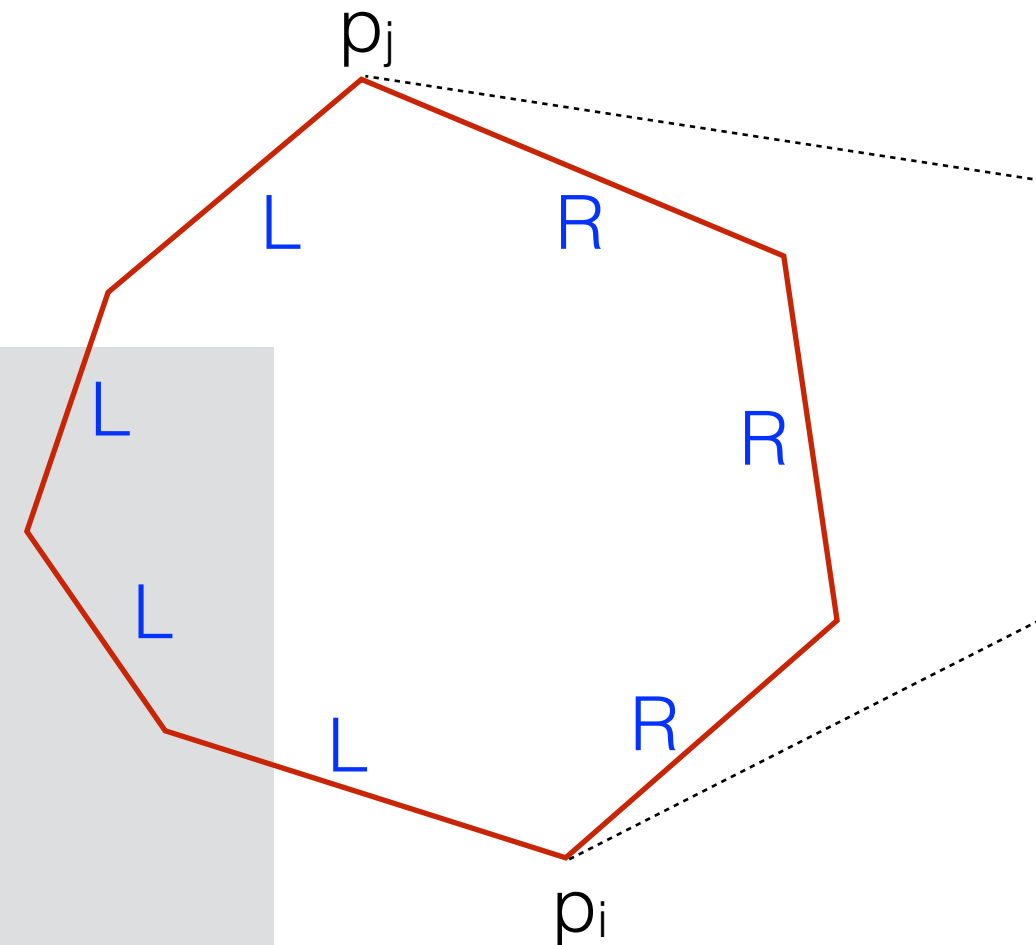
What do you notice? How can we use this to find the tangent points? Sketch an algorithm. How long does it take?

Finding tangent points

Input: point p outside H

polygon $H = [p_0, p_1, \dots, p_{k-1}]$ convex

- *for $i=0$ to $k-1$ do*
 - *$prev = ((i == 0)? k-1: i-1);$*
 - *$next = (i==k-1)? 0; k+1);$*
 - *if XOR (p is left-or-on (p_{prev}, p_i) , p is left-or-on (p_i, p_{next}))*
 - *then p_i is a tangent point*



After finding p_i and p_j , how would you update H ?

Back to an incremental algorithm for CH

Incremental algo for CH

- $H = [p_1, p_2, p_3]$
- for $i=4$ to n do
 - //add p_i to H
 - if $\text{point_in_polygon}(p, H)$
 - //do nothing
 - else
 - find p_i the tangent point where orientation changes from L to R
 - find p_j the tangent point where orientation changes from R to L
 - //note: p_i not necessarily before p_j in the vertex array of H
 - cut out the part from p_i to p_j in H (note: view H as wrapping around) and replace it with vertex p

Incremental algo for CH

- $H = [p_1, p_2, p_3]$
- for $i=4$ to n do
 - //add p_i to H
 - if $\text{point_in_polygon}(p, H)$
 - //do nothing
 - else
 - find p_i the tangent point where orientation changes from L to R
 - find p_j the tangent point where orientation changes from R to L
 - //note: p_i not necessarily before p_j in the vertex array of H
 - cut out the part from p_i to p_j in H (note: view H as wrapping around) and replace it with vertex p

Simulate the algorithm on a couple of examples.

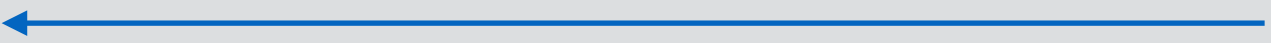

Think how p_i could come before p_j in H or the other way around.

Incremental algo for CH

- $H = [p_1, p_2, p_3]$
- for $i=4$ to n do
 - //add p_i to H
 - if $\text{point_in_polygon}(p, H)$
 - //do nothing
 - else
 - find p_i the tangent point where orientation changes from L to R
 - find p_j the tangent point where orientation changes from R to L
 - //note: p_i not necessarily before p_j in the vertex array of H
 - cut out the part from p_i to p_j in H (note: view H as wrapping around) and replace it with vertex p

Analysis:

Incremental algo for CH

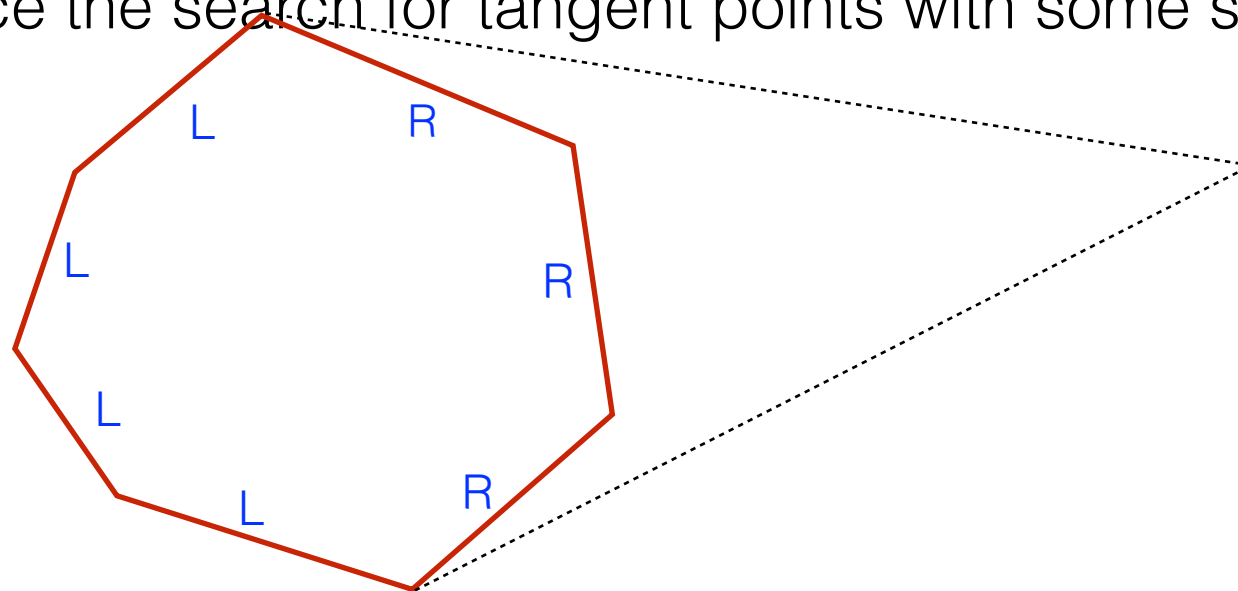
- $H = [p_1, p_2, p_3]$
- for $i=4$ to n do
 - //add p_i to H
 - if $\text{point_in_polygon}(p, H)$  $O(i)$
 - //do nothing
 - else
 - find p_i the tangent point where orientation changes from L to R  $O(i)$
 - find p_j the tangent point where orientation changes from R to L
 - //note: p_i not necessarily before p_j in the vertex array of H
 - cut out the part from p_i to p_j in H (note: view H as wrapping around) and replace it with vertex p

Analysis: $\sum_i O(i) = O(n^2)$

Incremental algo for CH

- The “straightforward” incremental algorithm is $O(n^2)$
- Improvement:
 - pre-sort the points by their x-coordinates and add them in this order

What does this give us?
- It was shown that $O(n \lg n)$ incremental algorithm is possible.
 - avoid re-computing all orientations every time
 - replace the search for tangent points with some sort of binary search



A divide-and-conquer algorithm for CH

Divide-and-conquer

DC(input P)

if P is small, solve and return

else

//divide

divide input P into two halves, P1 and P2

//recurse

result1 = **DC(P1)**

result2 = **DC(P2)**

//merge

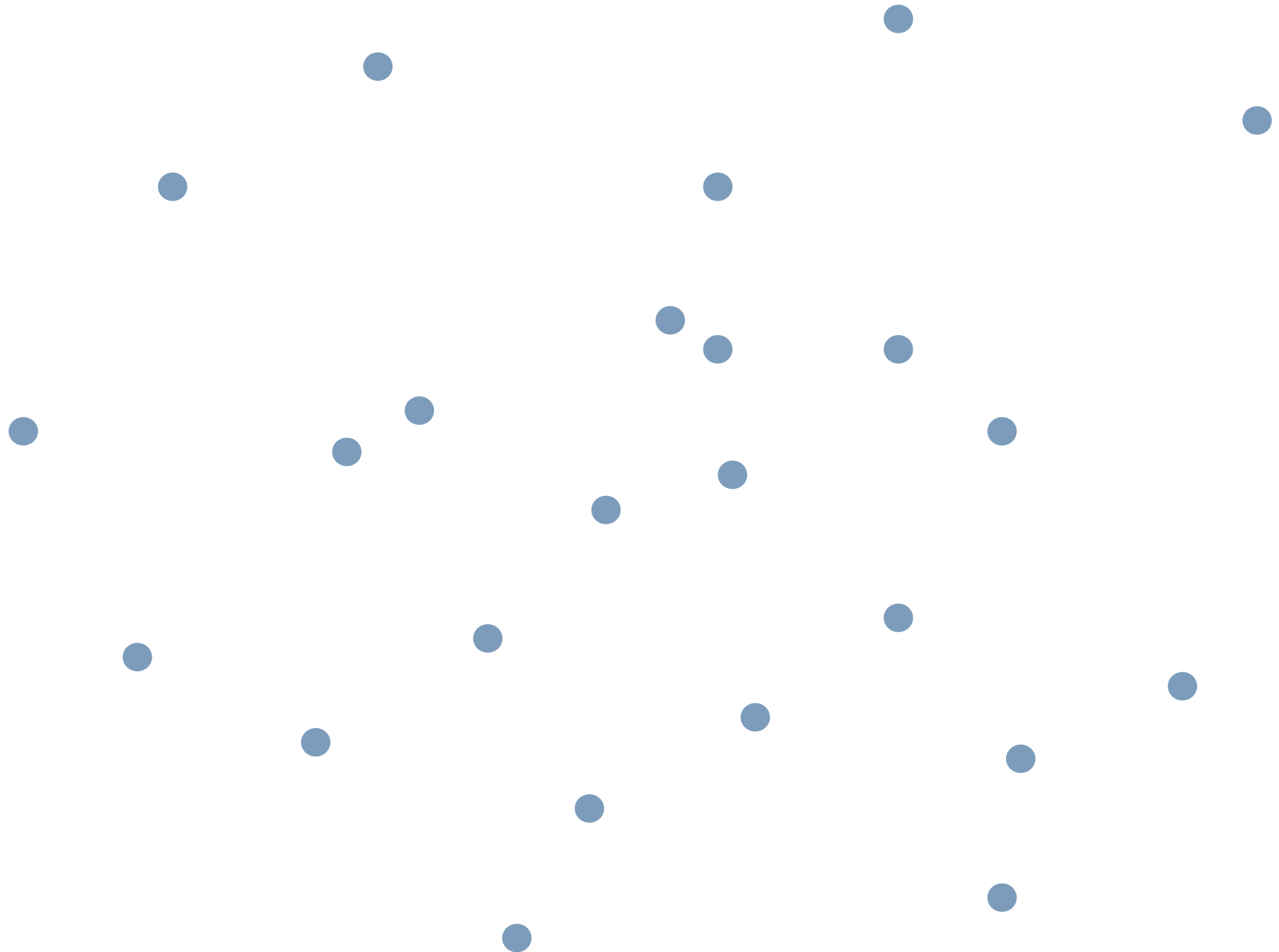
do_something_to_figure_out_result_for_P

return result

Analysis: $T(n) = 2T(n/2) + O(\text{merge phase})$

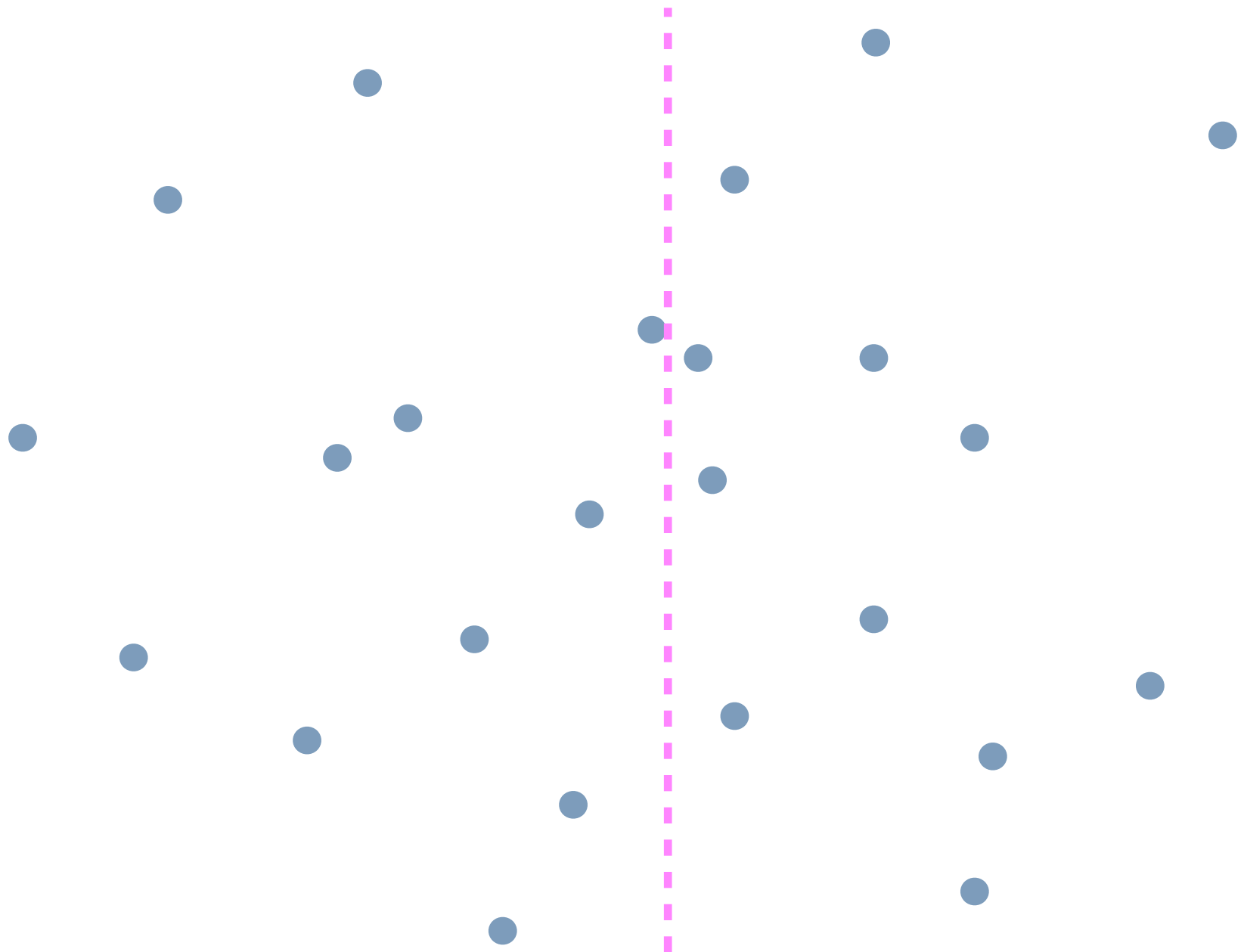
- if merge phase is **$O(n)$** : $T(n) = 2T(n/2) + O(n) \Rightarrow O(n \lg n)$

CH via divide-and-conquer



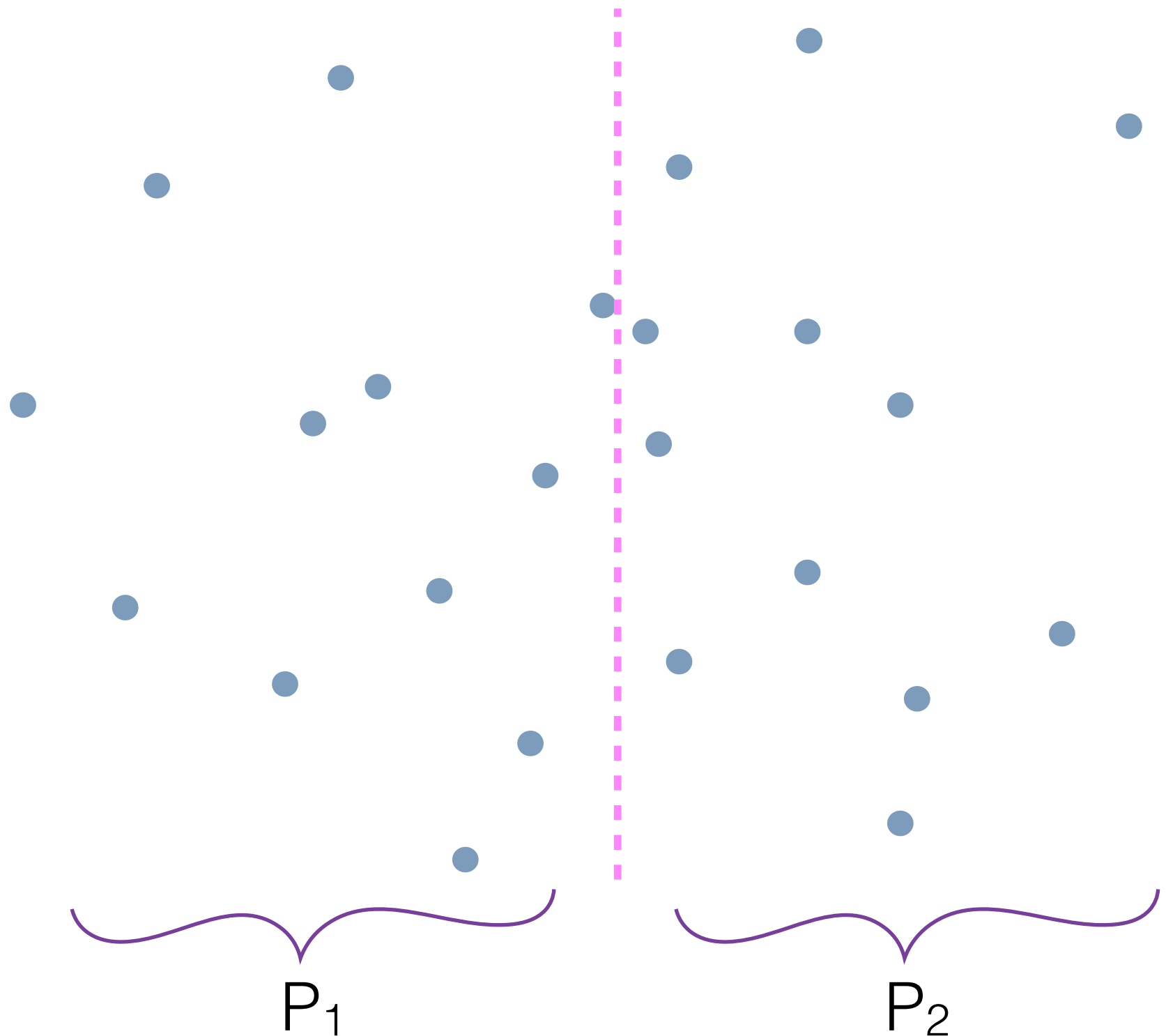
CH via divide-and-conquer

- find vertical line that splits P in half



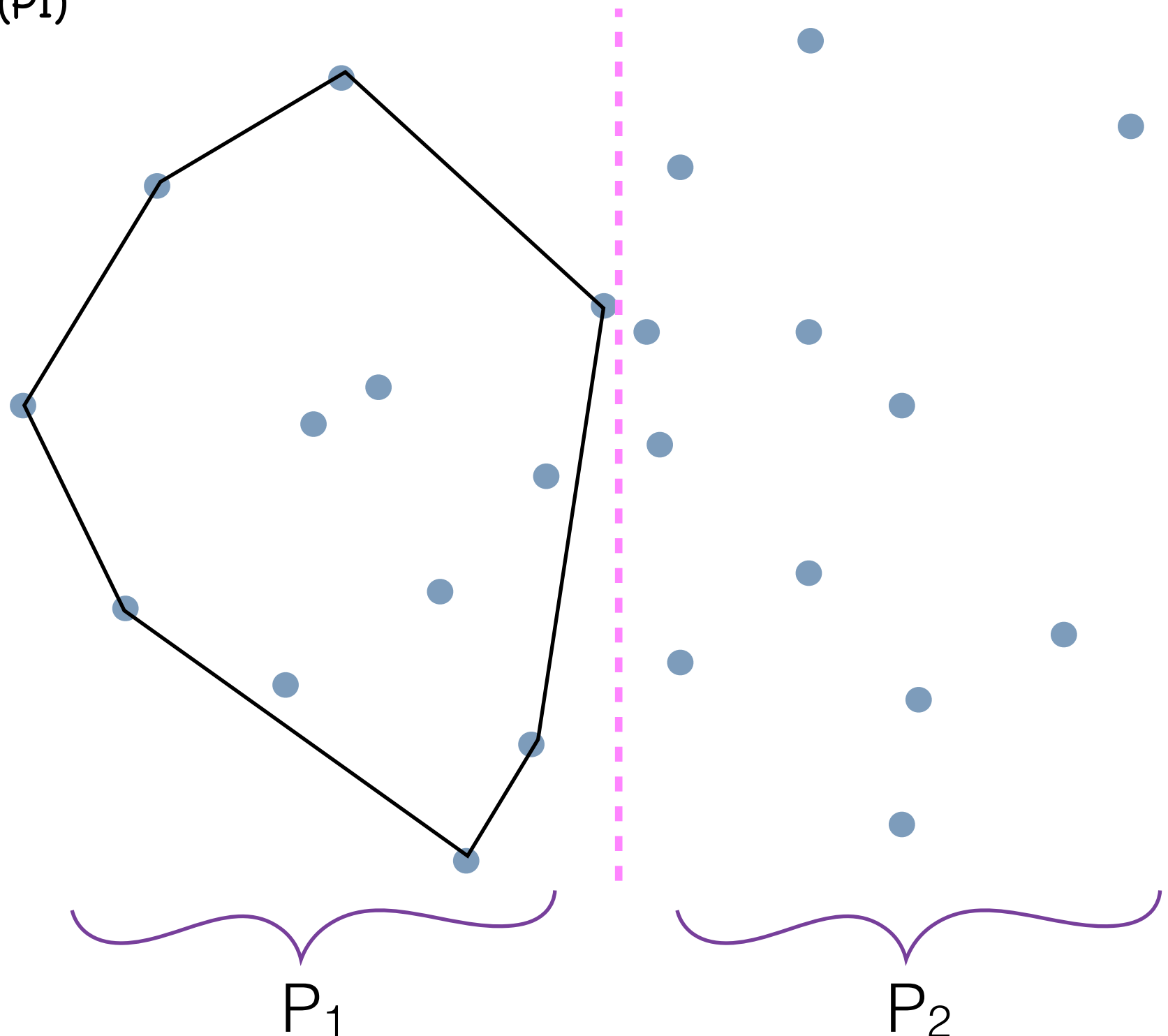
CH via divide-and-conquer

- find vertical line that splits P in half
- let P_1, P_2 = set of points to the left/right of line



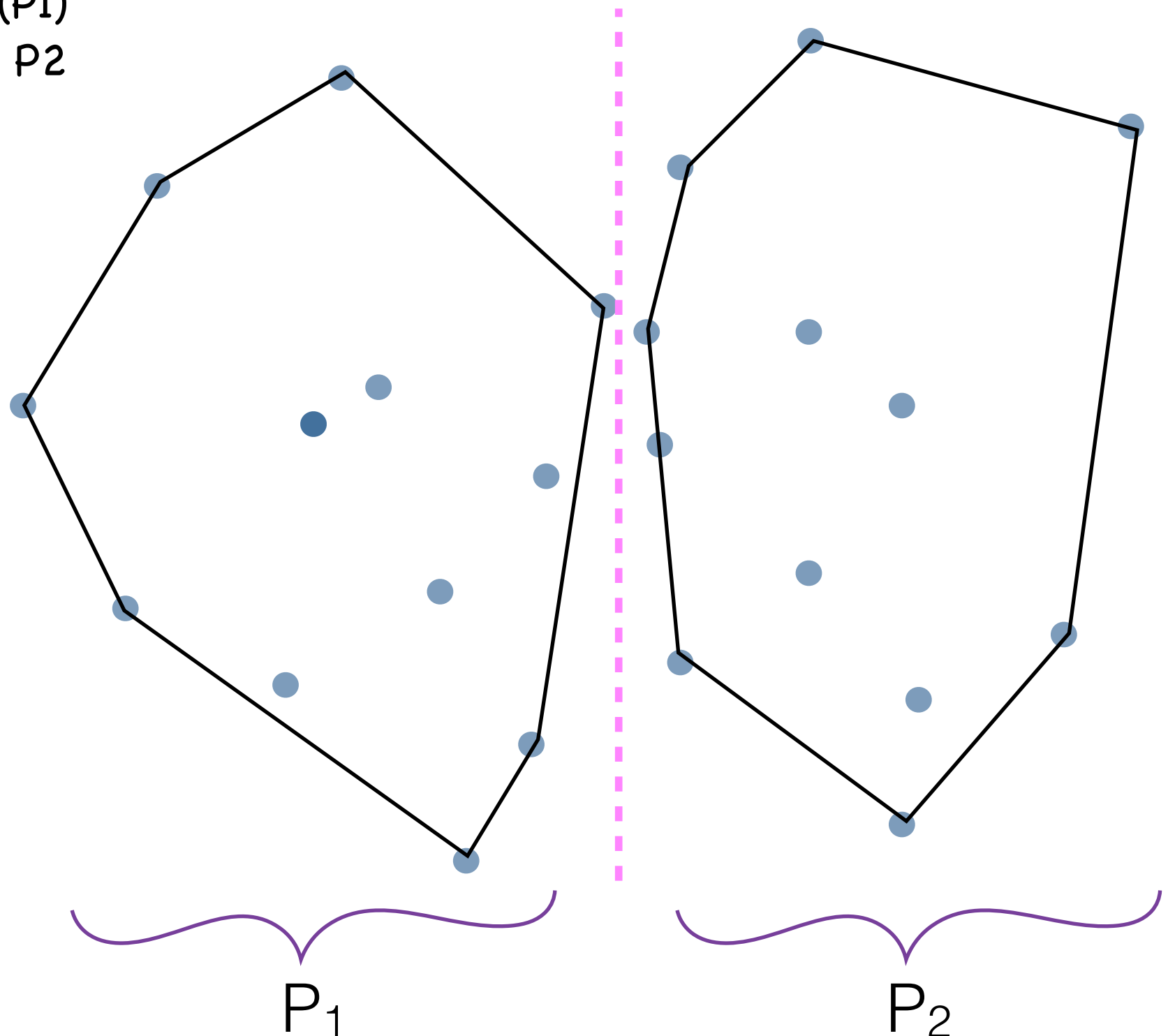
CH via divide-and-conquer

- find vertical line that splits P in half
- let P_1, P_2 = set of points to the left/right of line
- recursively find $CH(P_1)$



CH via divide-and-conquer

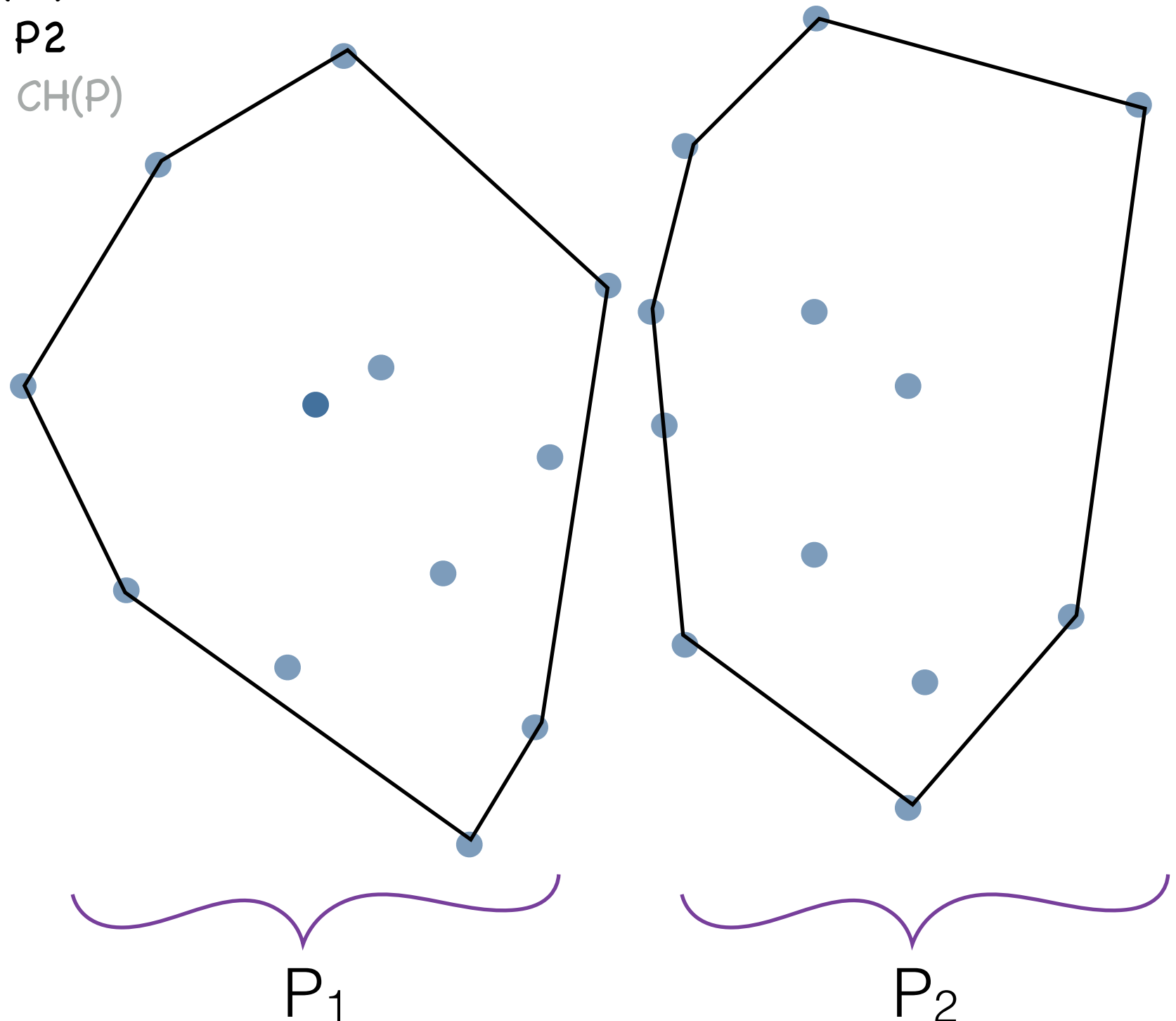
- find vertical line that splits P in half
- let P_1, P_2 = set of points to the left/right of line
- recursively find $CH(P_1)$
- recursively find $CH P_2$



CH via divide-and-conquer

- find vertical line that splits P in half
- let P_1, P_2 = set of points to the left/right of line
- recursively find $CH(P_1)$
- recursively find $CH(P_2)$

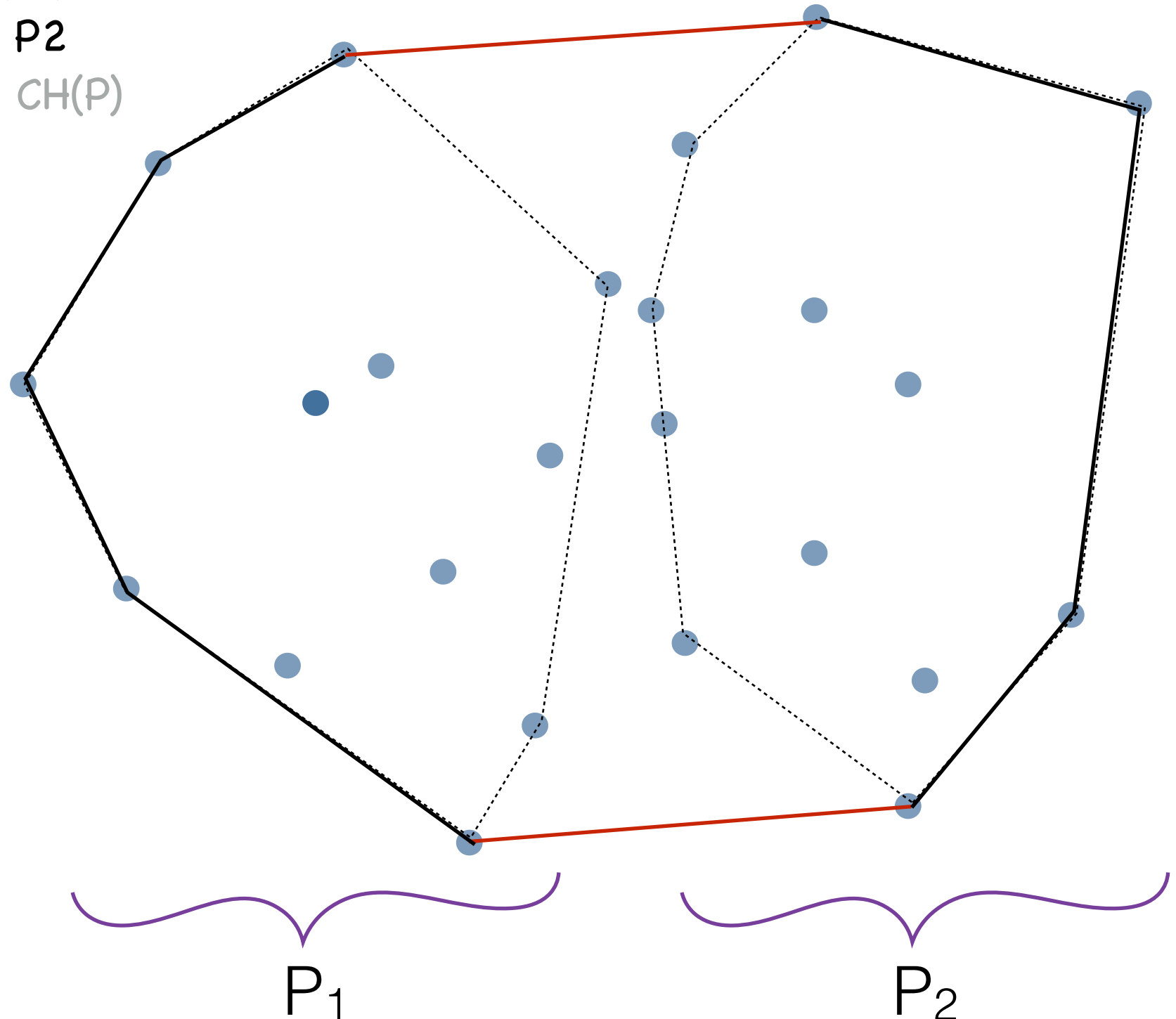
//now get somehow $CH(P)$



CH via divide-and-conquer

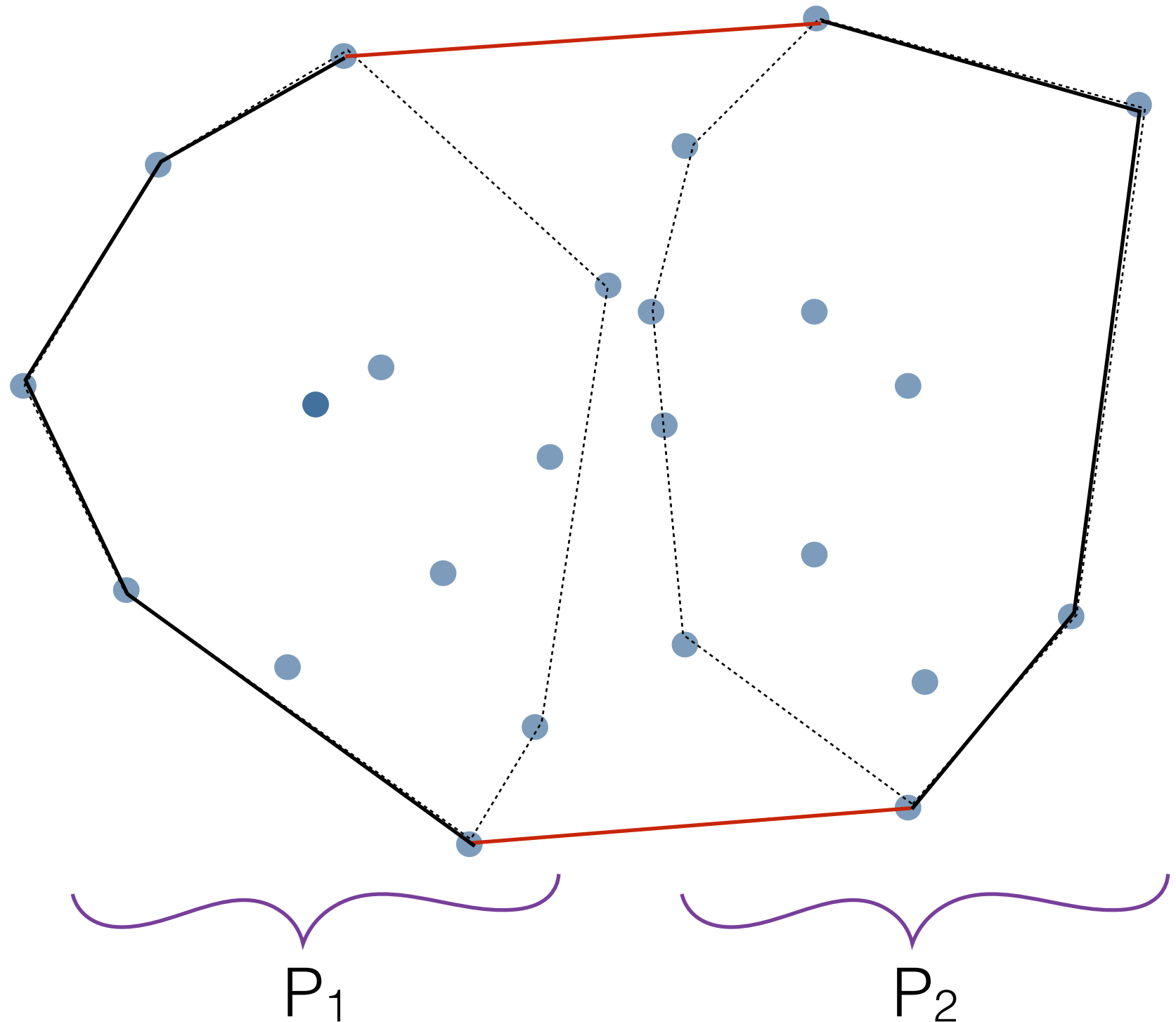
- find vertical line that splits P in half
- let P_1, P_2 = set of points to the left/right of line
- recursively find $CH(P_1)$
- recursively find $CH(P_2)$

//now get somehow $CH(P)$



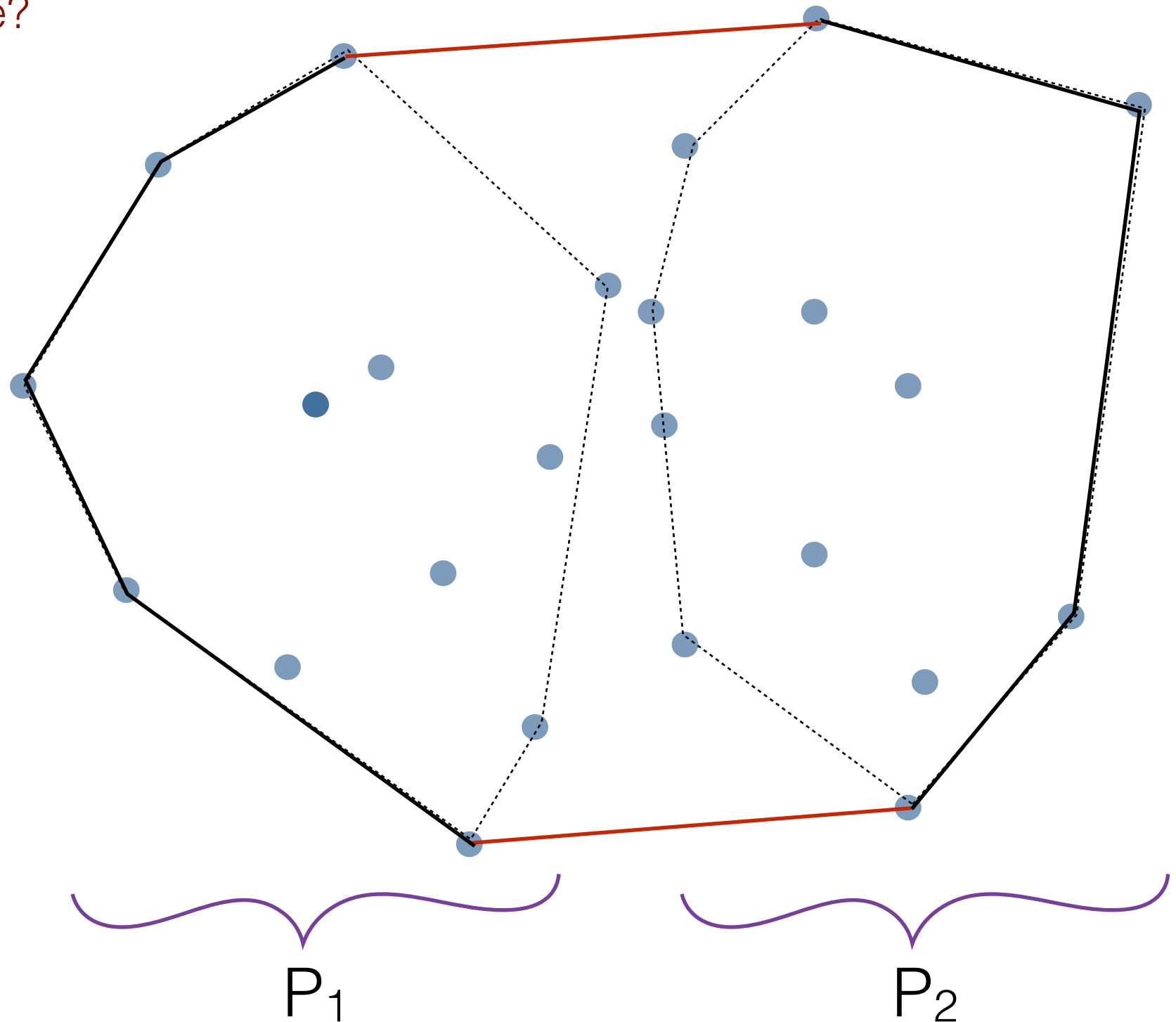
Merging two hulls..in linear time

- Need to find the two tangents

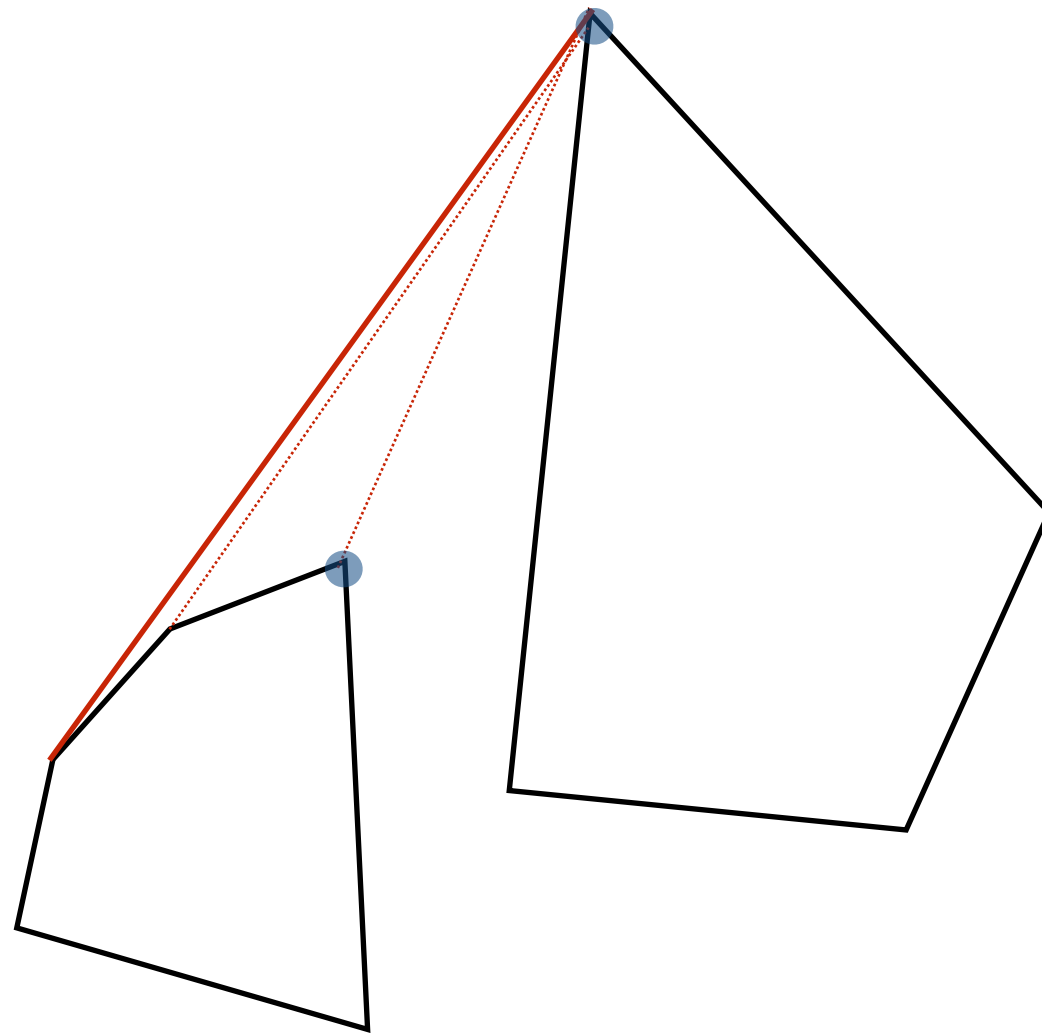


Merging two hulls..in linear time

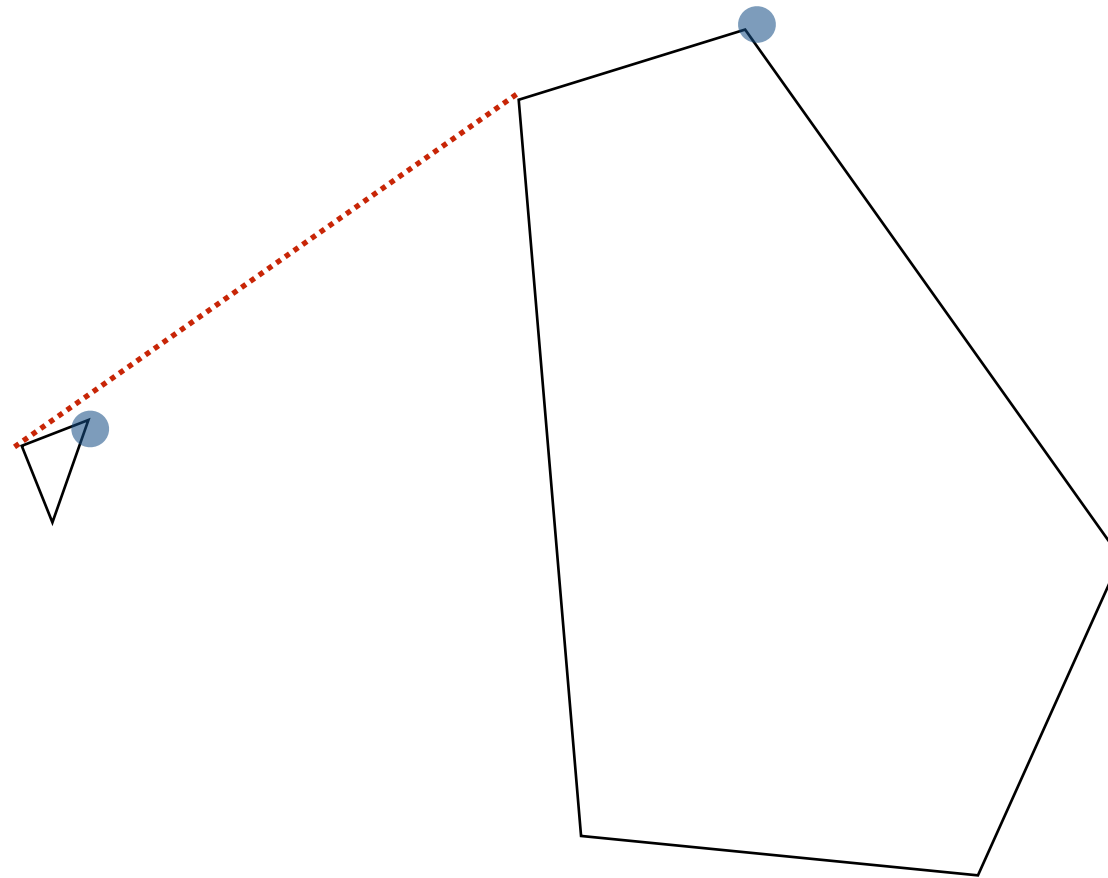
- Here it looks like the upper tangent is between the top points in P_1 and P_2
- Is that always true?



Not necessarily...



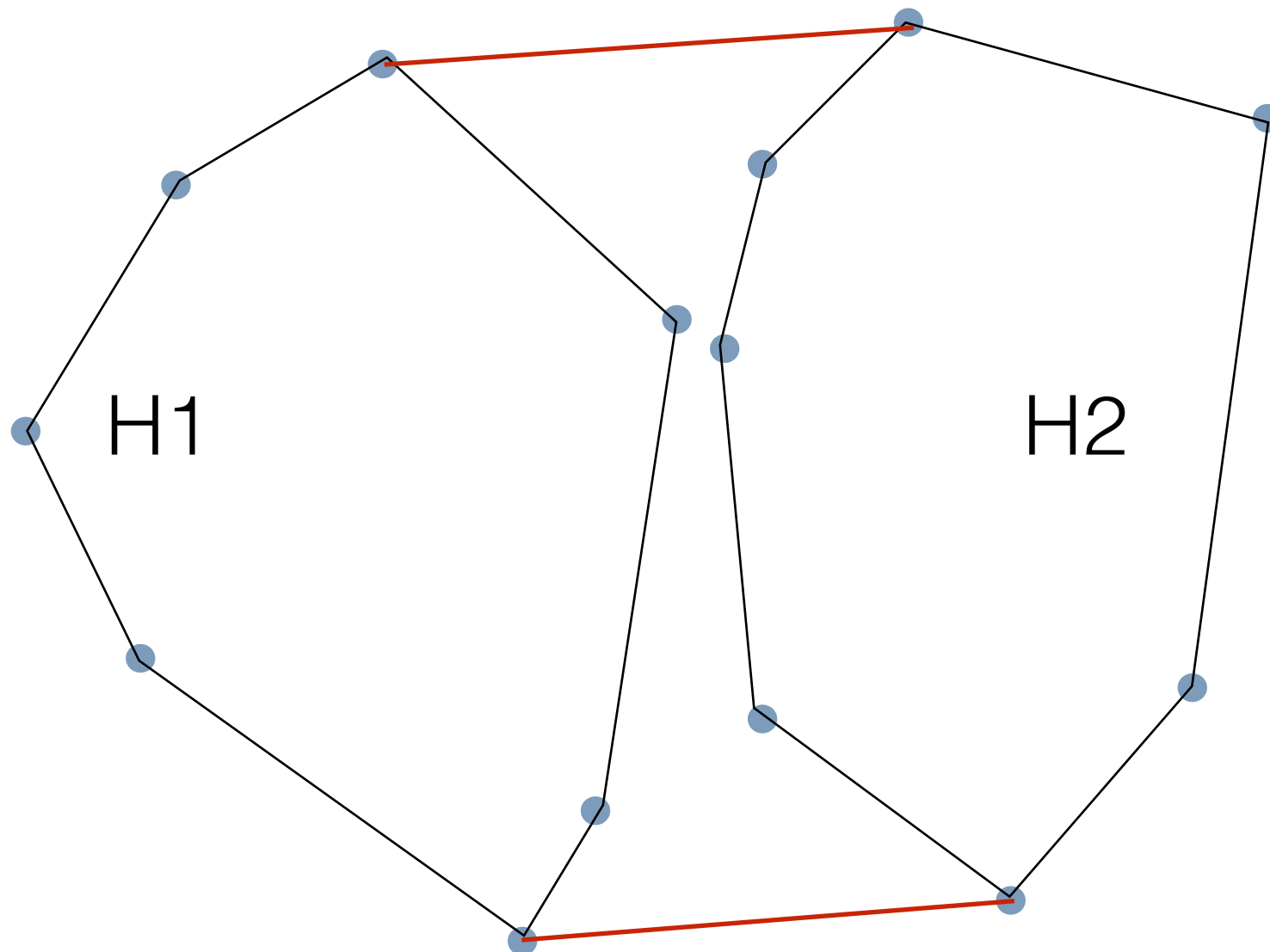
The top-most point overall is on the CH, but not necessarily on the upper tangent



Merging two hulls..in linear time

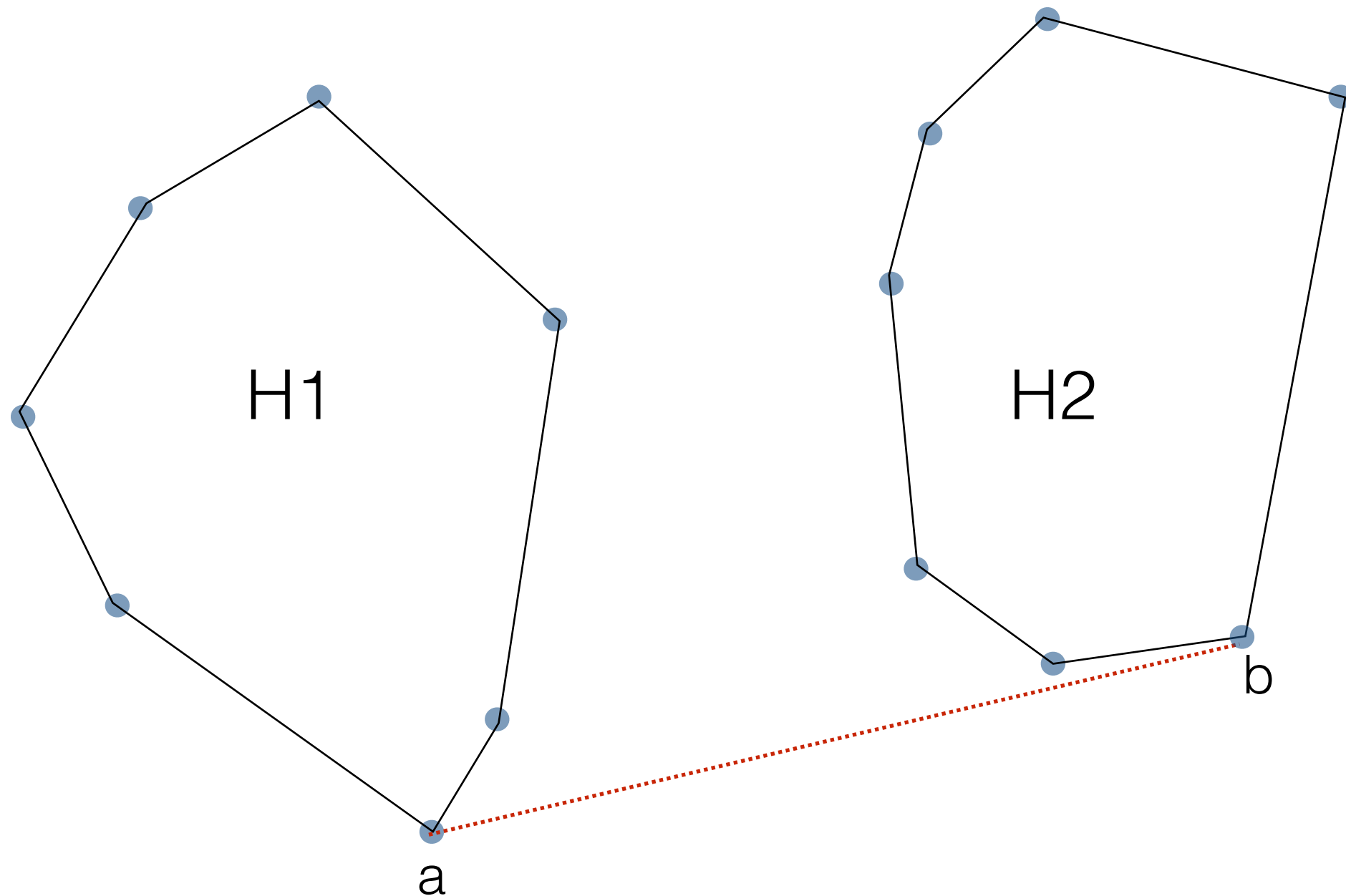
- Naive algorithm: try all segments (a,b) with a in H_1 and b in H_2

Too slow. $\Rightarrow O(n^2)$ merge, $O(n^2 \lg n)$ CH algorithm



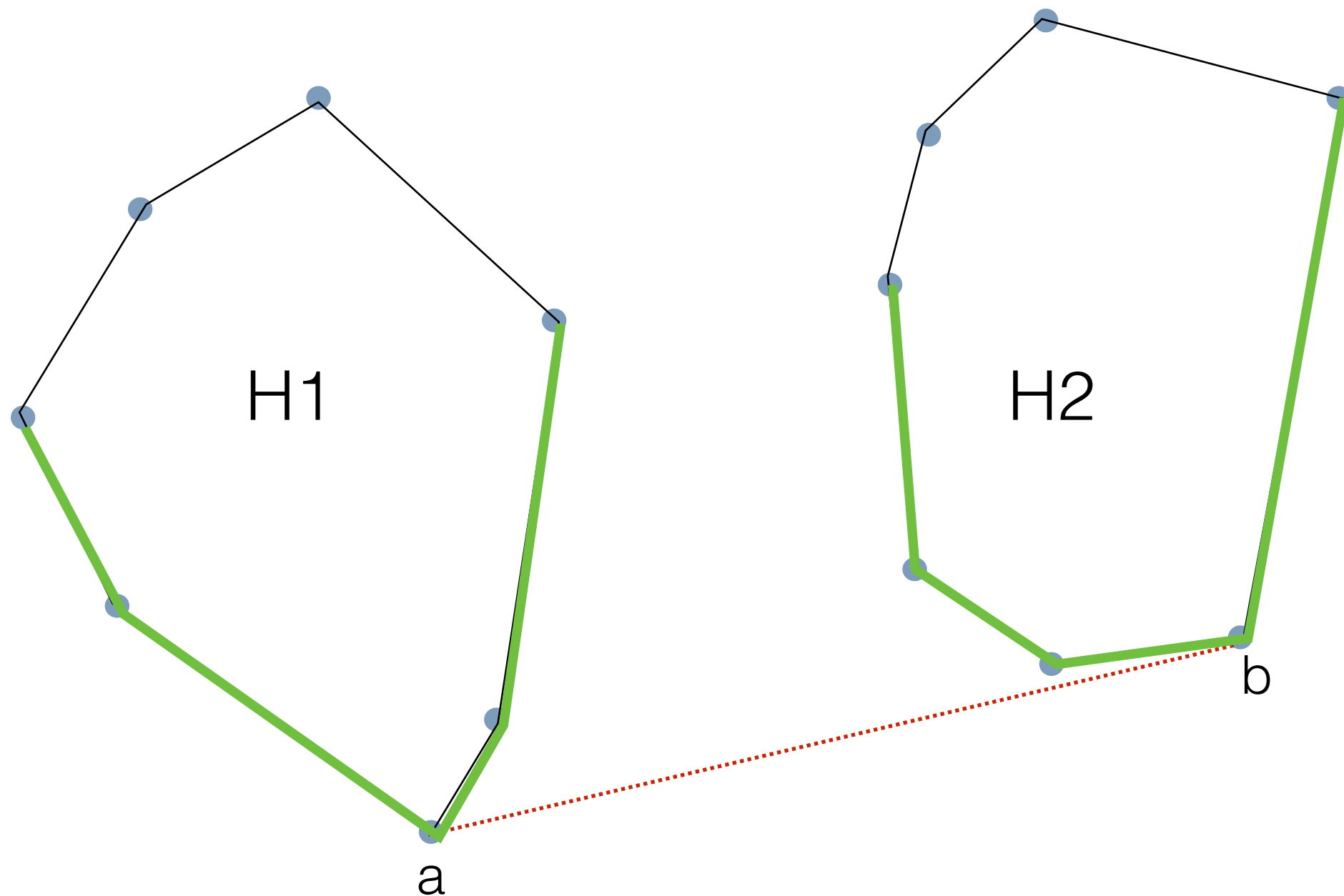
Finding the lower tangent

- Claim: All points in $H1$ and $H2$ are to the left of ab



Finding the lower tangent

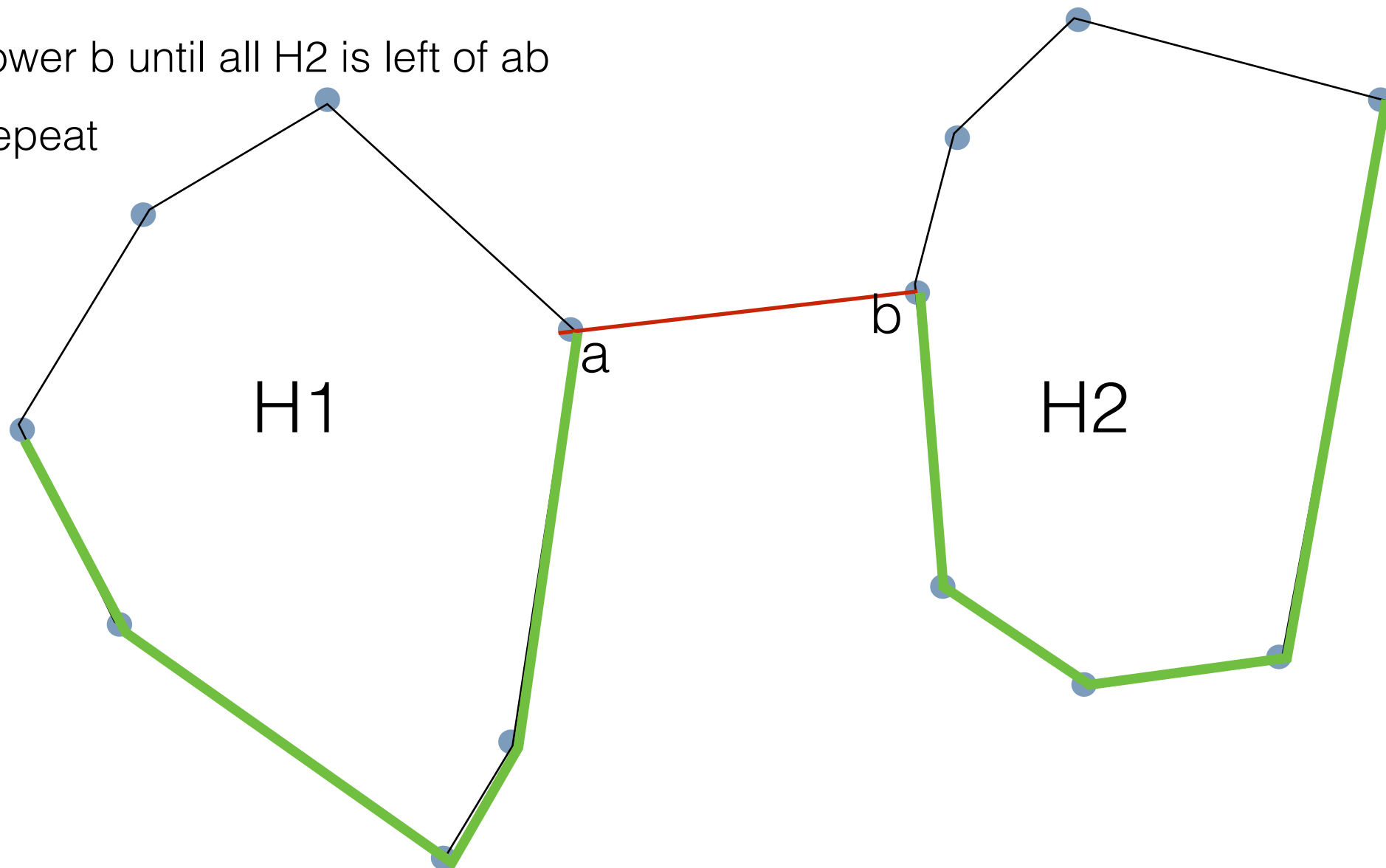
- Claim: Points a, b are on the lower hulls of $H1$ and $H2$, respectively.



Finding the lower tangent

- Idea:

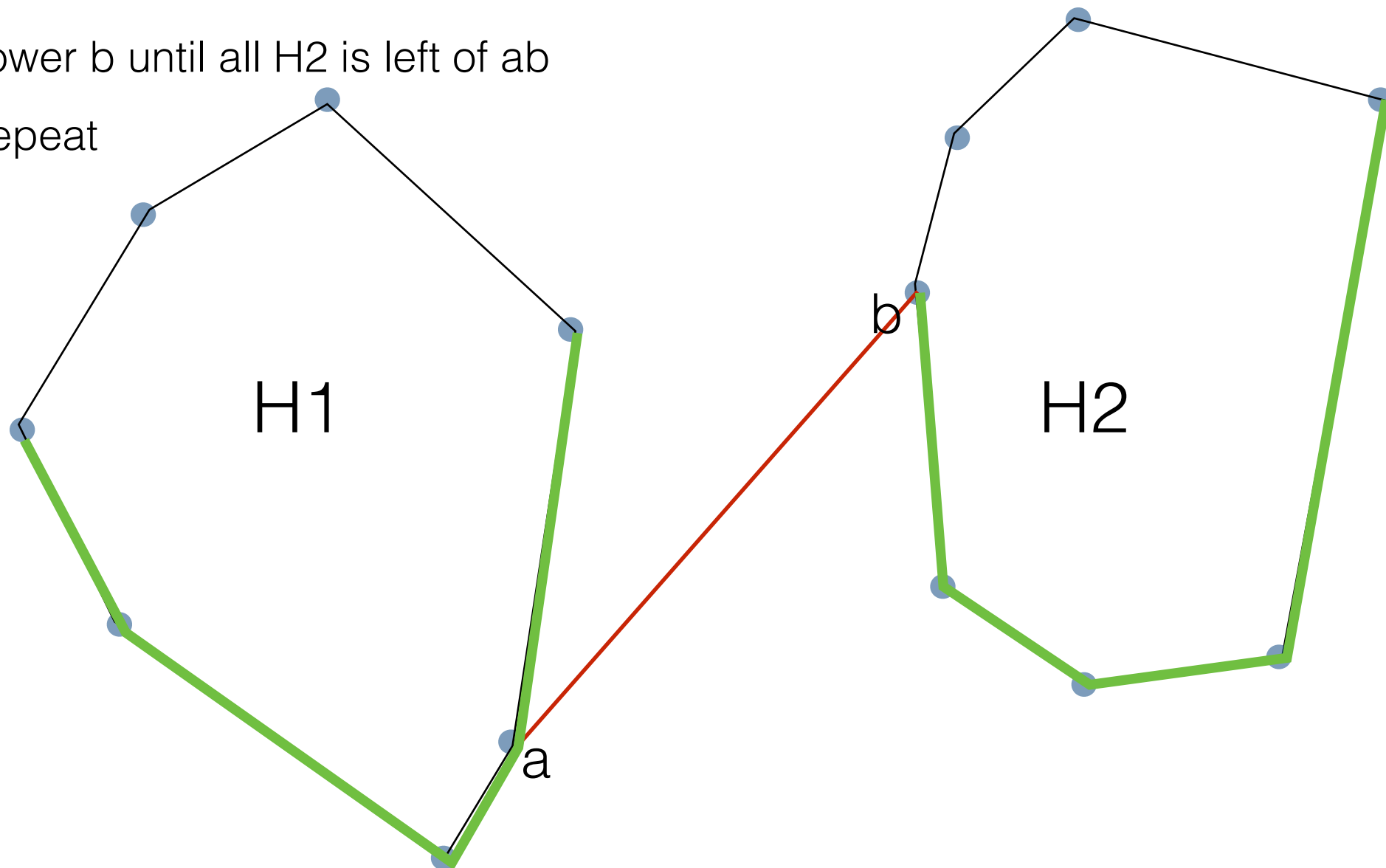
- start with a = rightmost point in $H1$, b = leftmost point in $H2$
- lower a until all $H1$ is left of ab
- lower b until all $H2$ is left of ab
- repeat



Finding the lower tangent

- Idea:

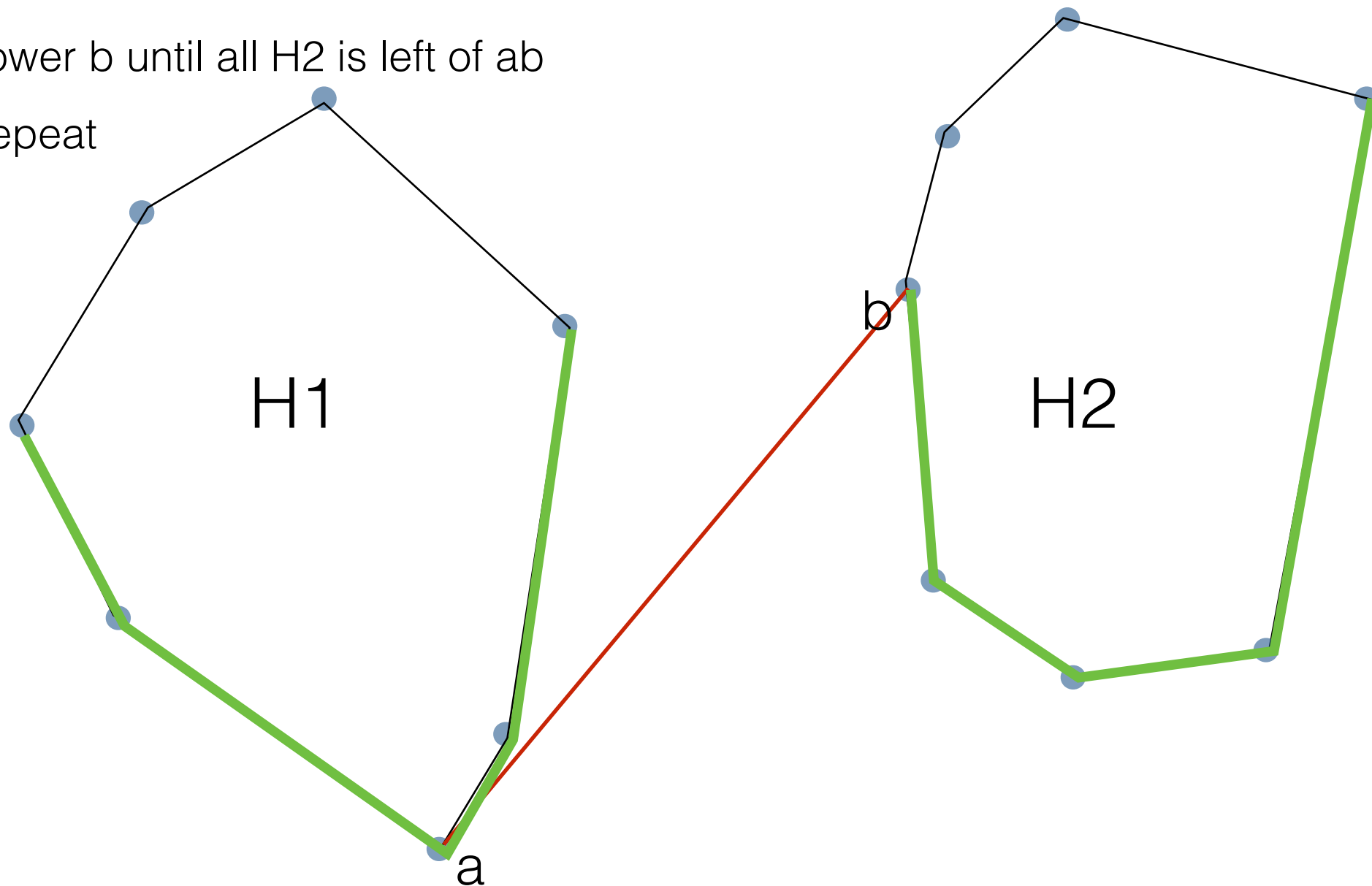
- start with a = rightmost point in $H1$, b = leftmost point in $H2$
- lower a until all $H1$ is left of ab
- lower b until all $H2$ is left of ab
- repeat



Finding the lower tangent

- Idea:

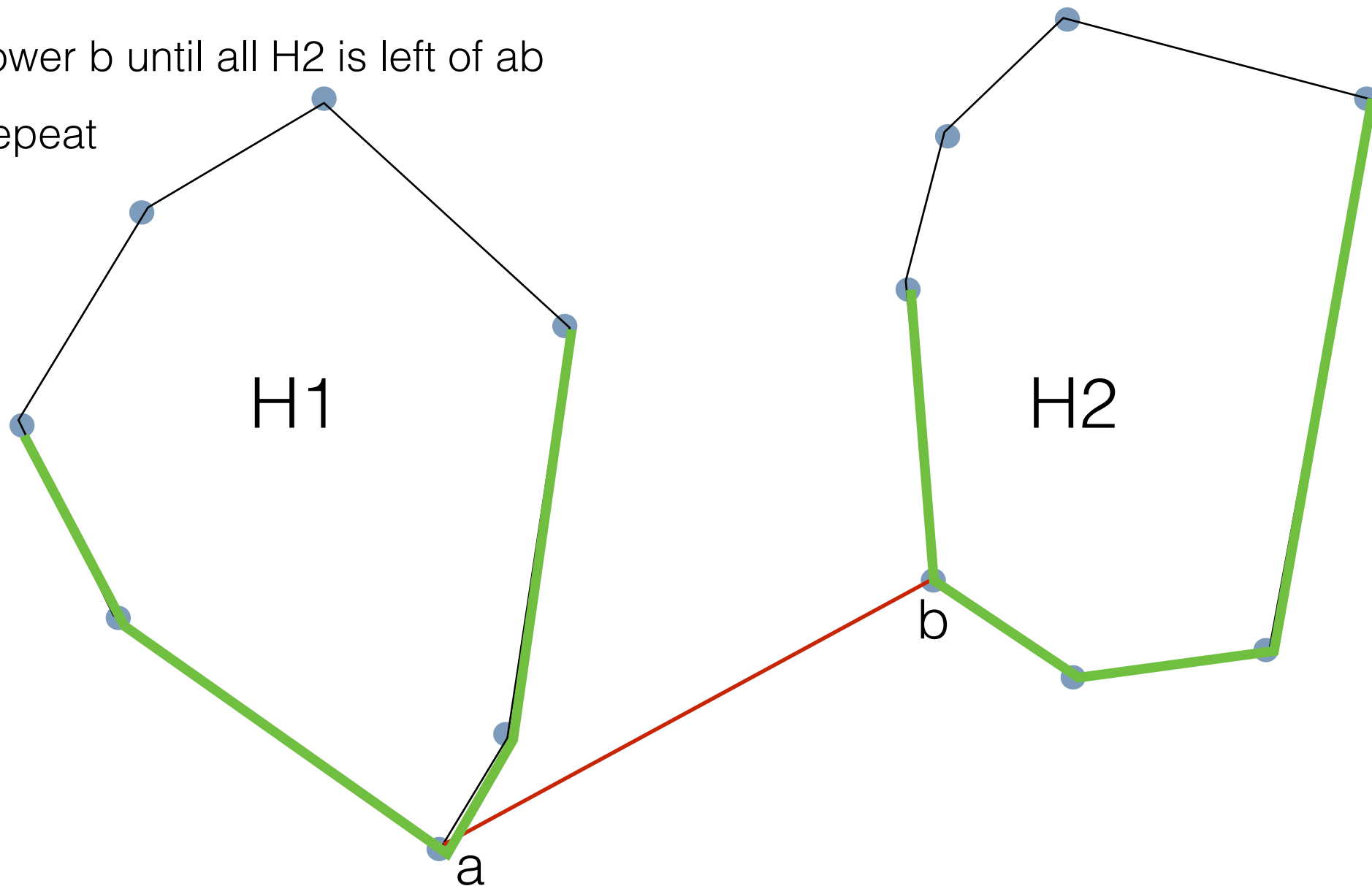
- start with a = rightmost point in $H1$, b = leftmost point in $H2$
- lower a until all $H1$ is left of ab
- lower b until all $H2$ is left of ab
- repeat



Finding the lower tangent

- Idea:

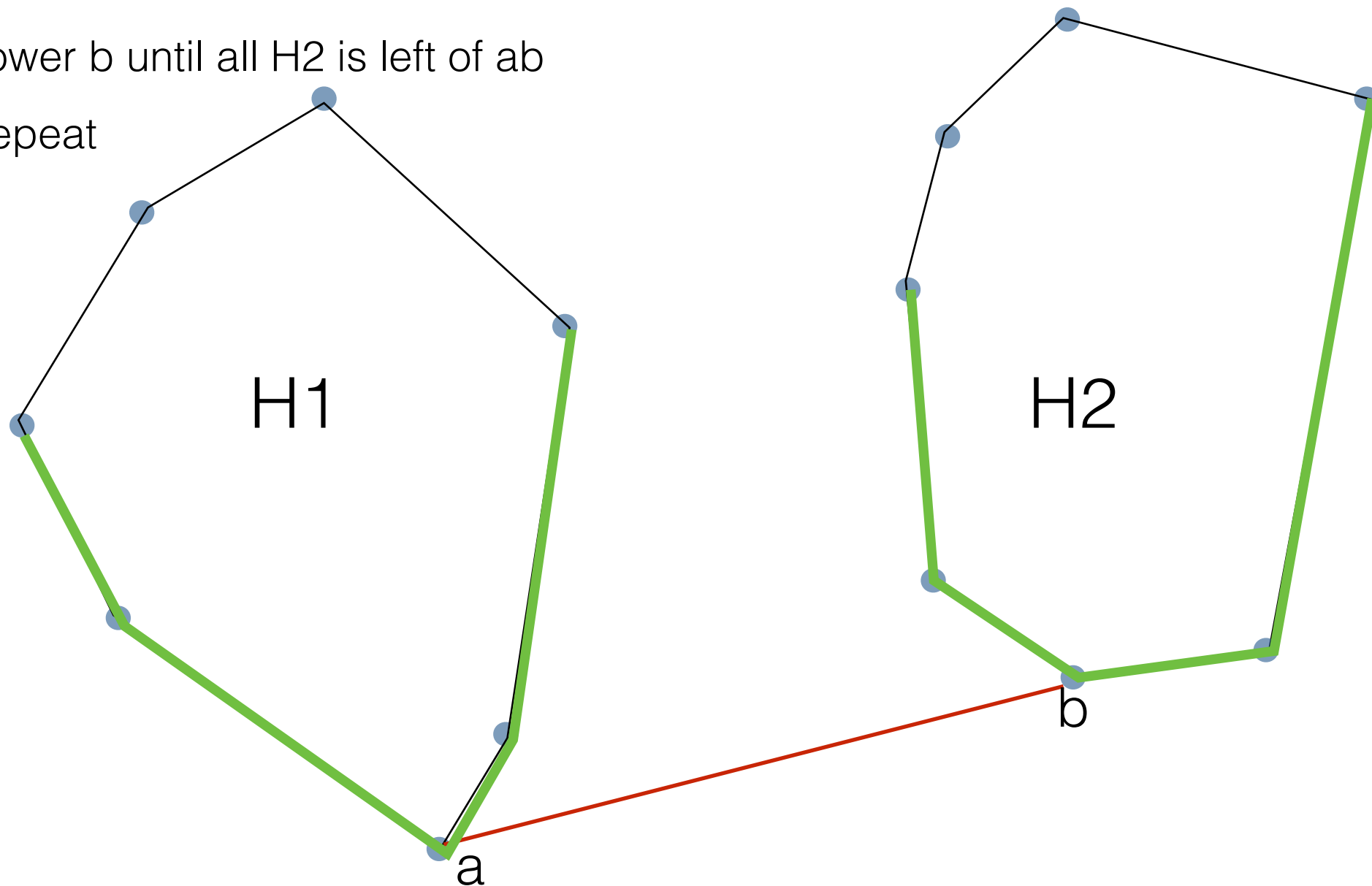
- start with a = rightmost point in $H1$, b = leftmost point in $H2$
- lower a until all $H1$ is left of ab
- lower b until all $H2$ is left of ab
- repeat



Finding the lower tangent

- Idea:

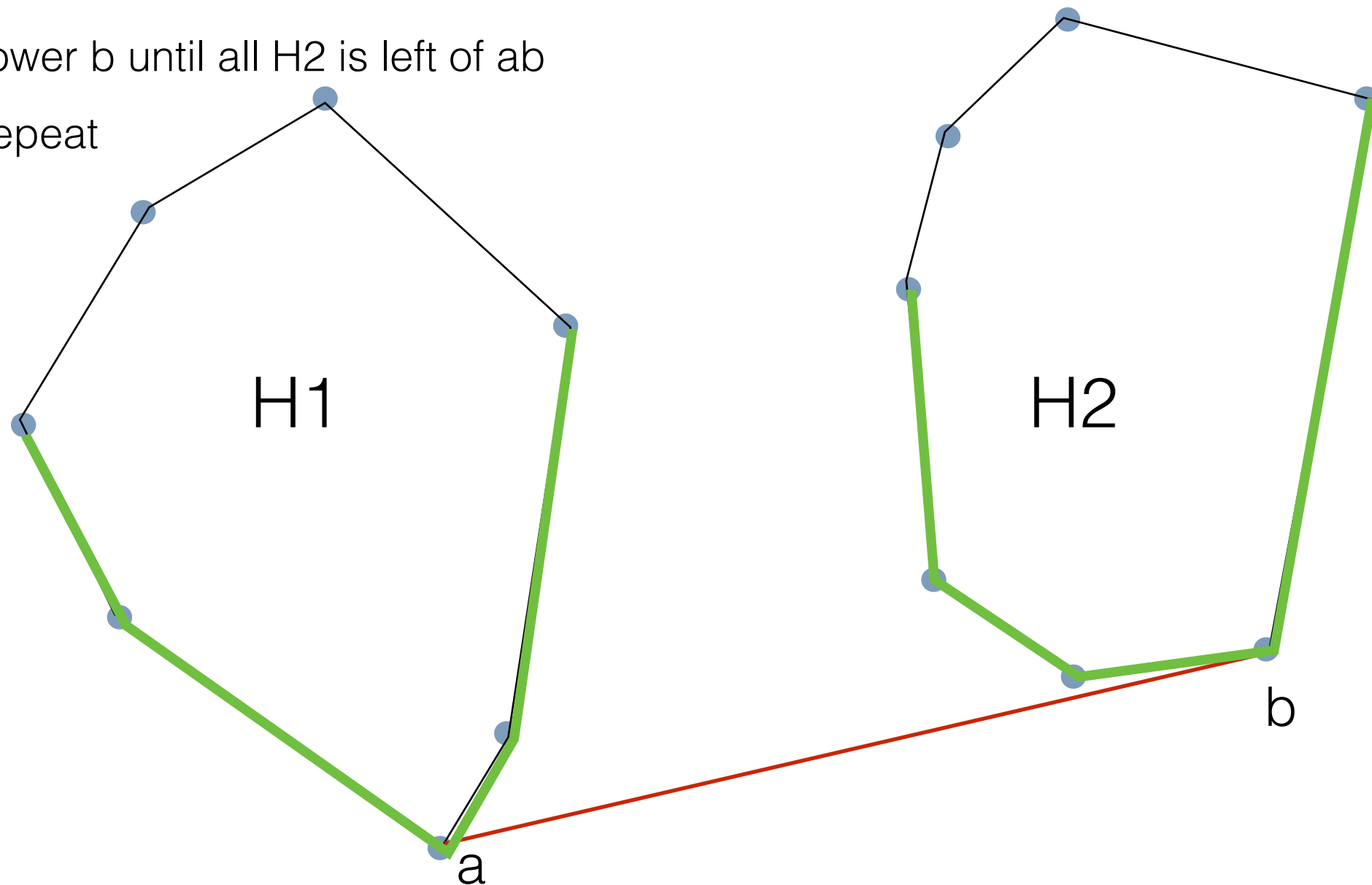
- start with a = rightmost point in $H1$, b = leftmost point in $H2$
- lower a until all $H1$ is left of ab
- lower b until all $H2$ is left of ab
- repeat



Finding the lower tangent

- Idea:

- start with a = rightmost point in $H1$, b = leftmost point in $H2$
- lower a until all $H1$ is left of ab
- lower b until all $H2$ is left of ab
- repeat



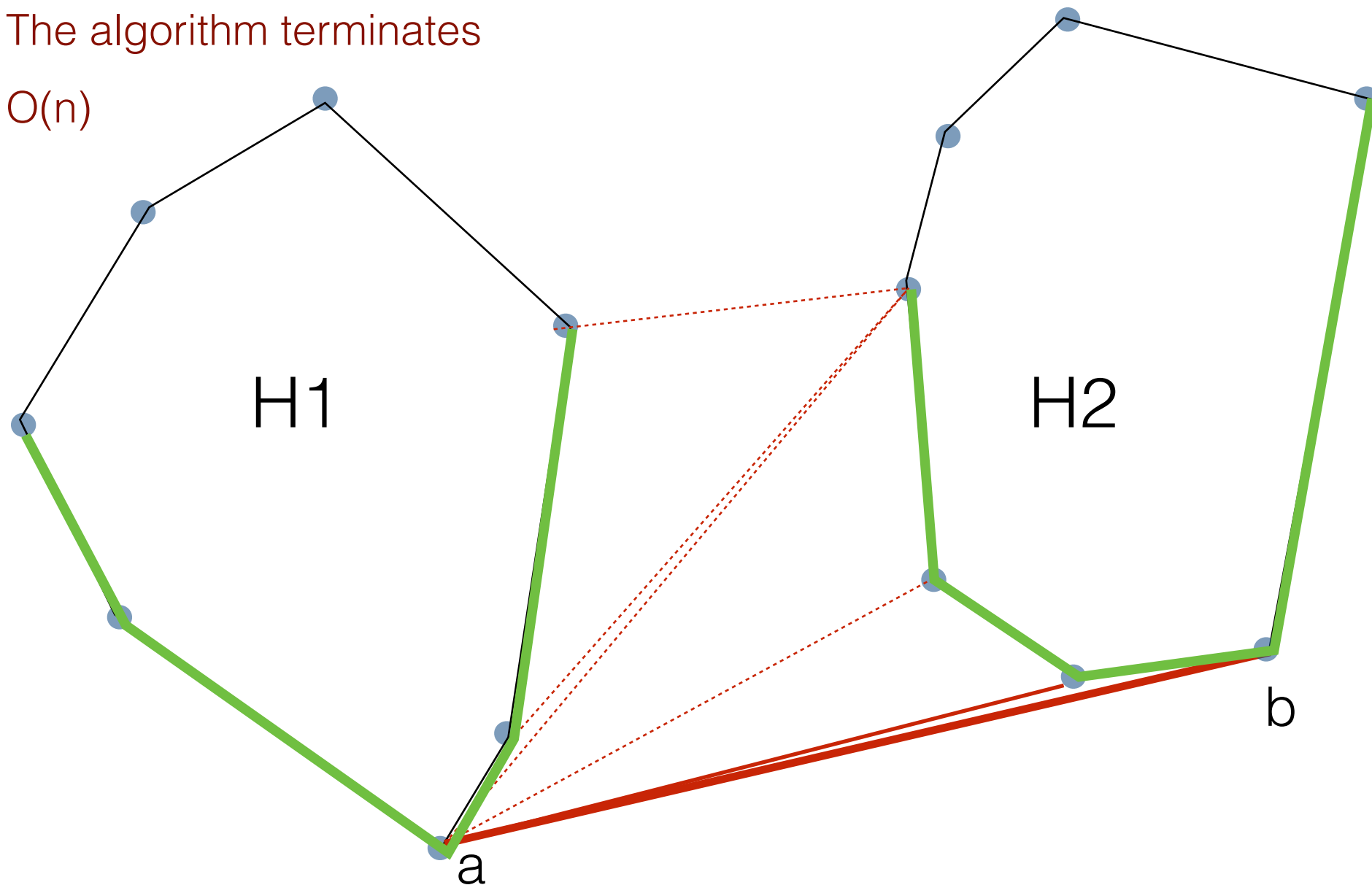
(why) does this work?

Claim: At any point during the algorithm, segment ab cannot intersect the interior of the polygons

\implies a cannot move into the upper hull of $P1$, b cannot move into the upper hull of $P2$

\implies The algorithm terminates

$\implies O(n)$



CH via divide-and-conquer

- Yet another illustration of divide-and-conquer paradigm
- Runs in $O(n \lg n)$
- Extends to $O(n \lg h)$
- Extends nicely to 3D