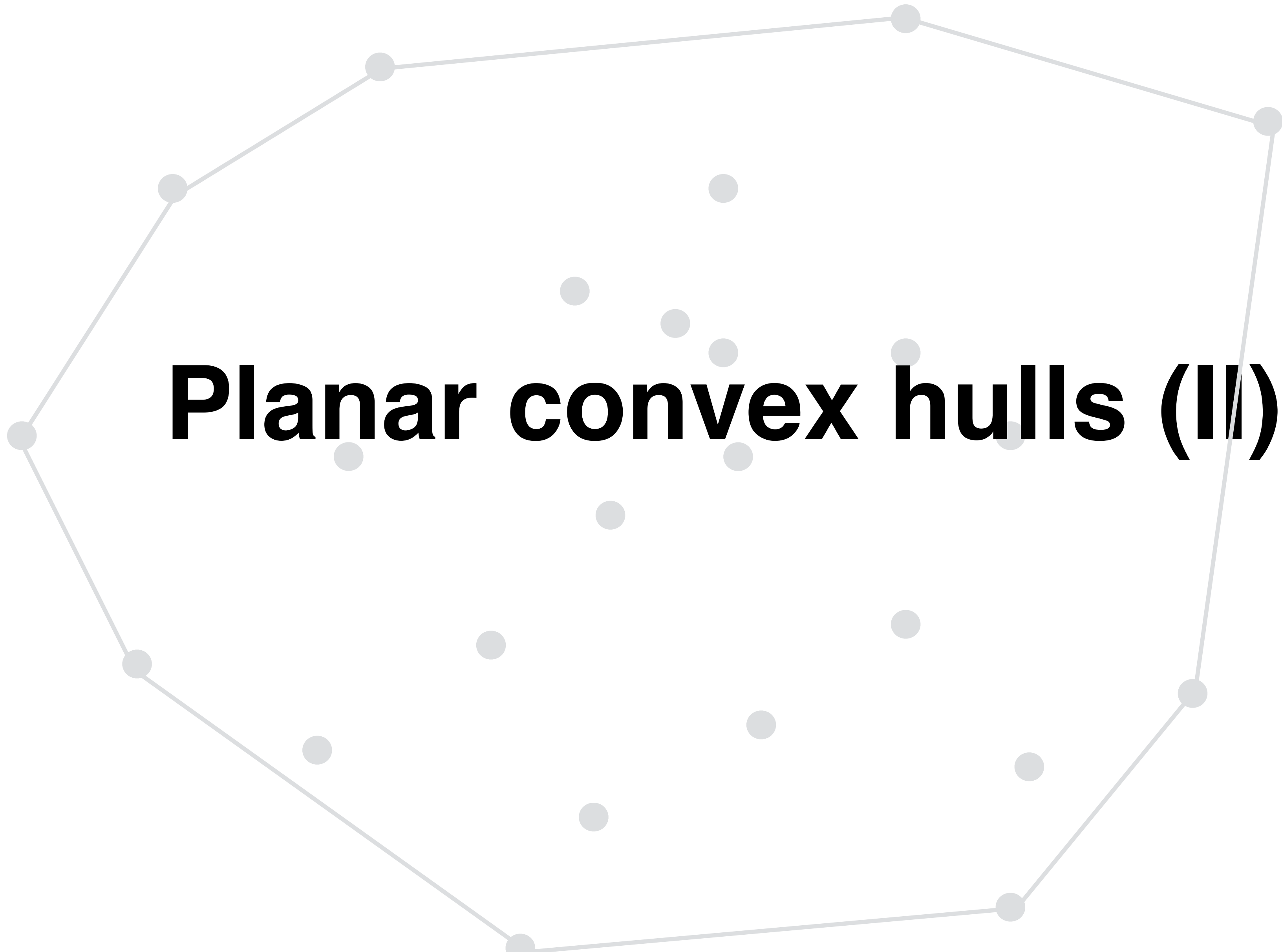
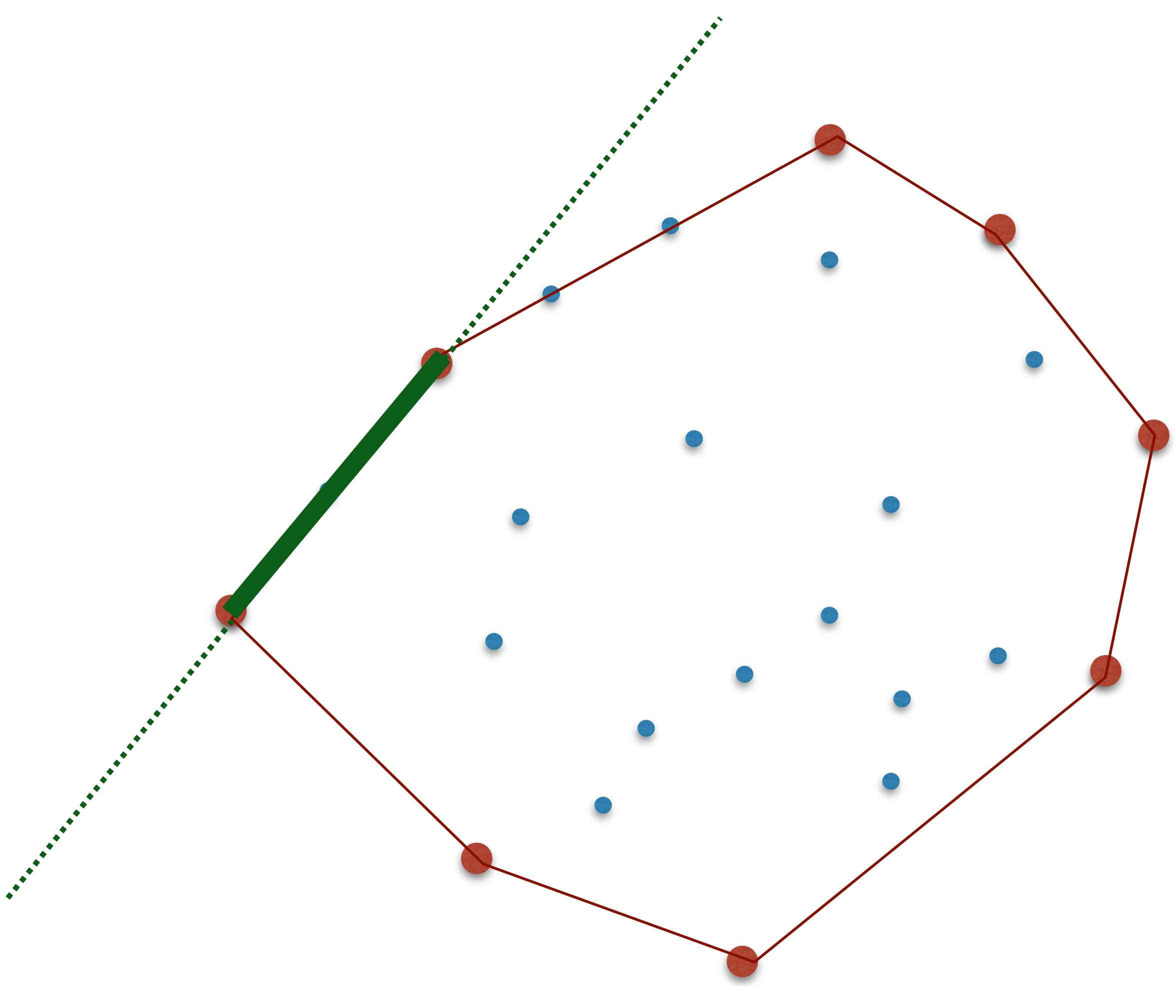


Planar convex hulls (II)

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



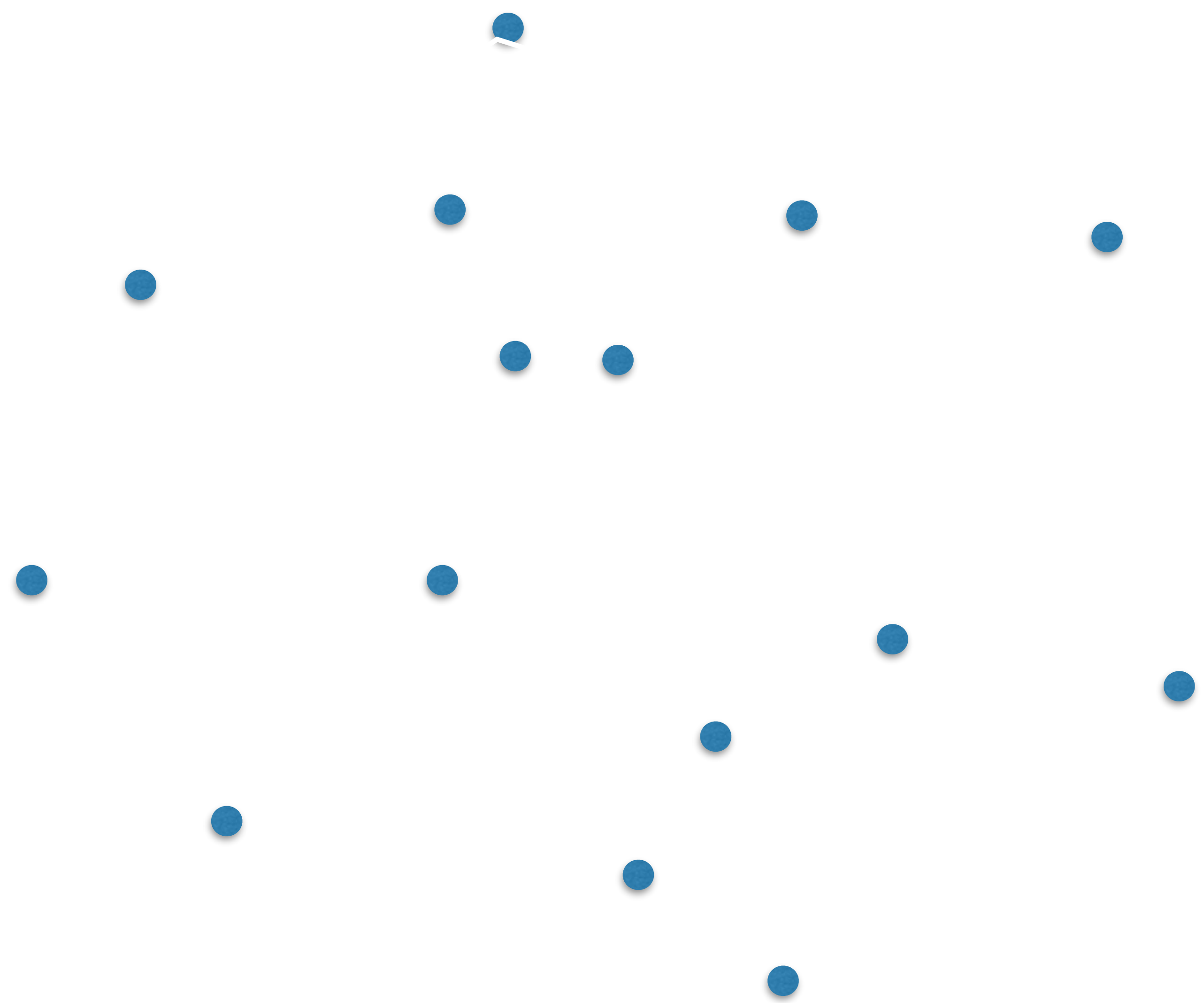
Outline

- 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

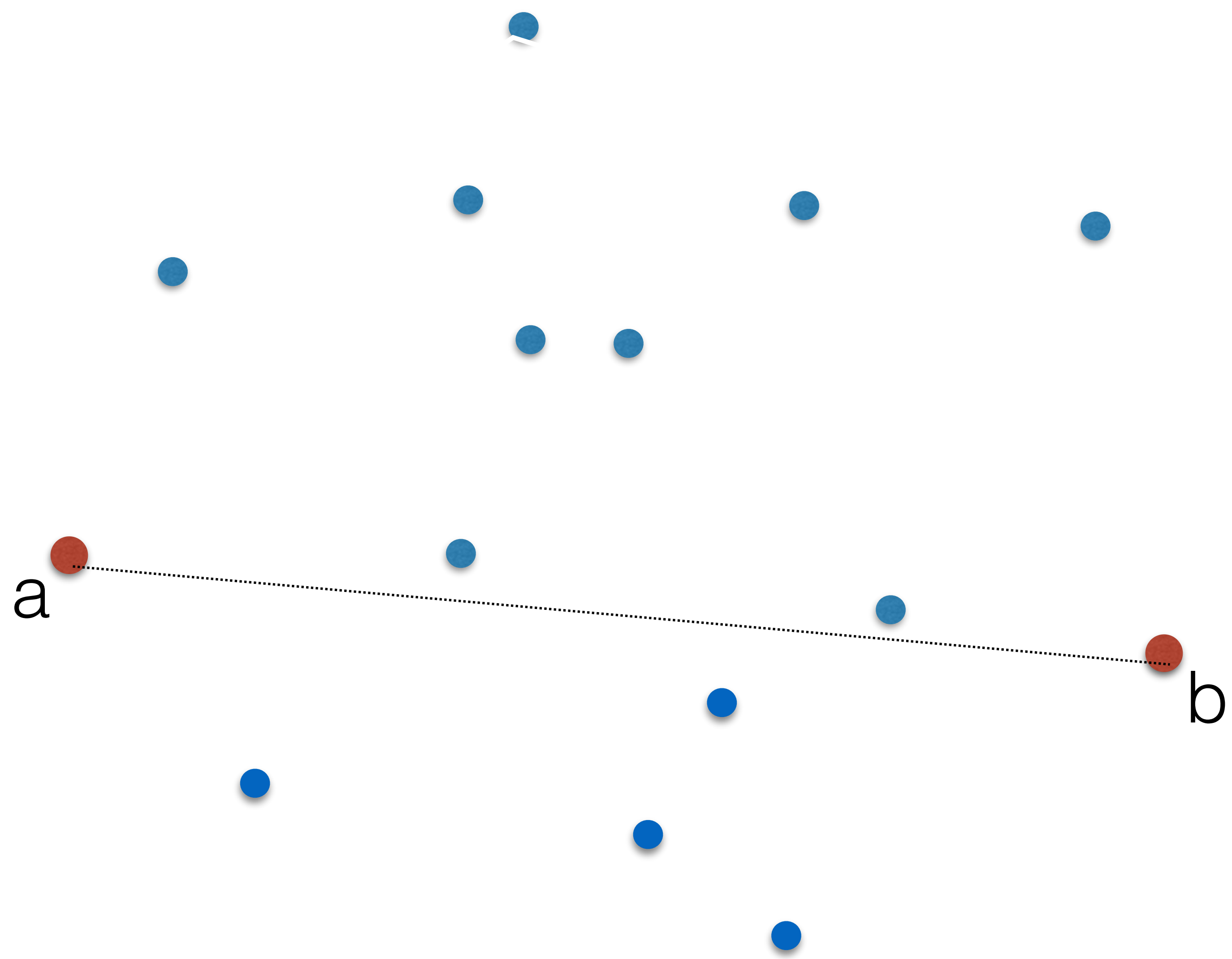
Andrew’s Monotone Chain Algorithm (1979)

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



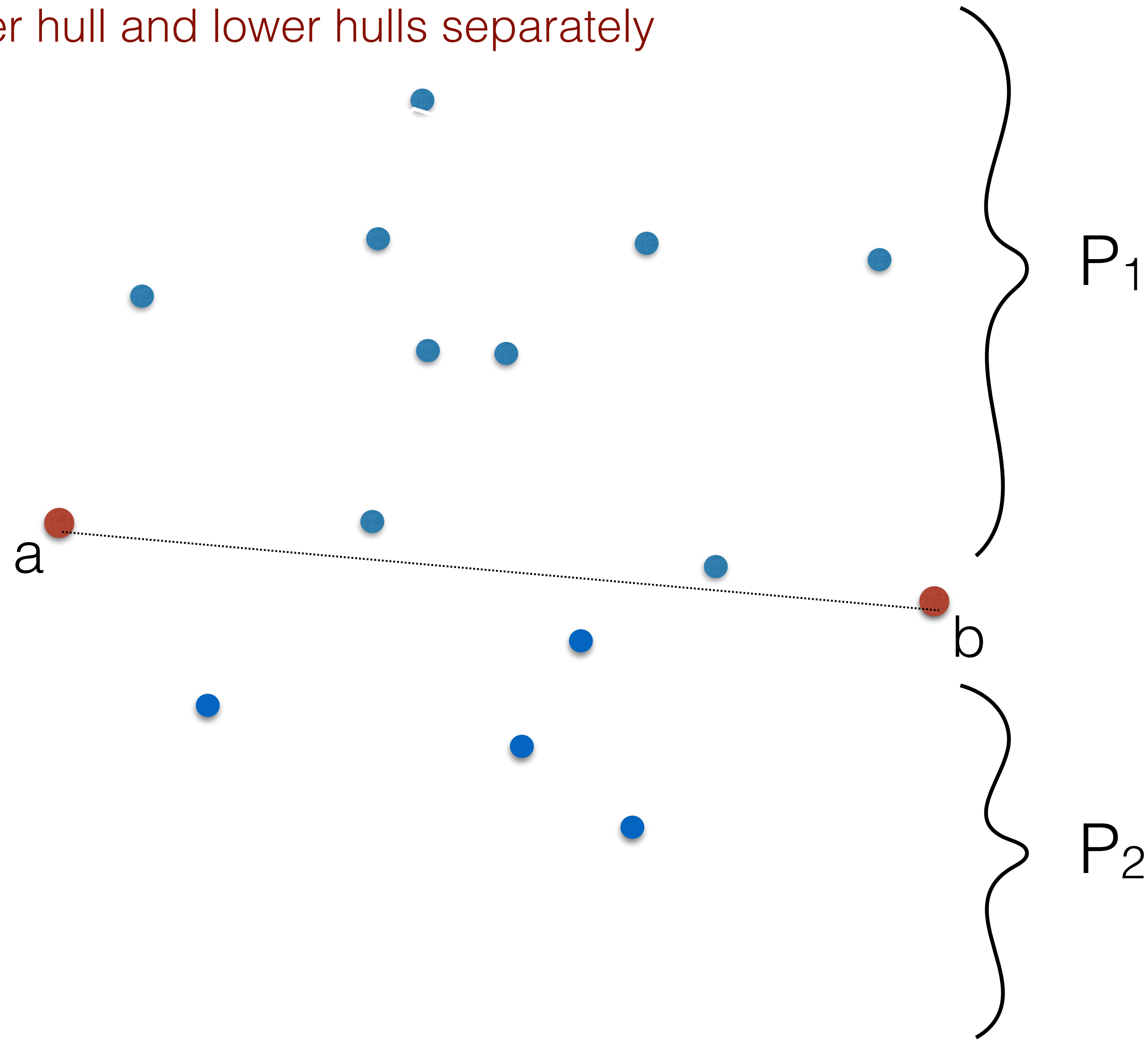
Andrew's Monotone Chain Algorithm (1979)

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



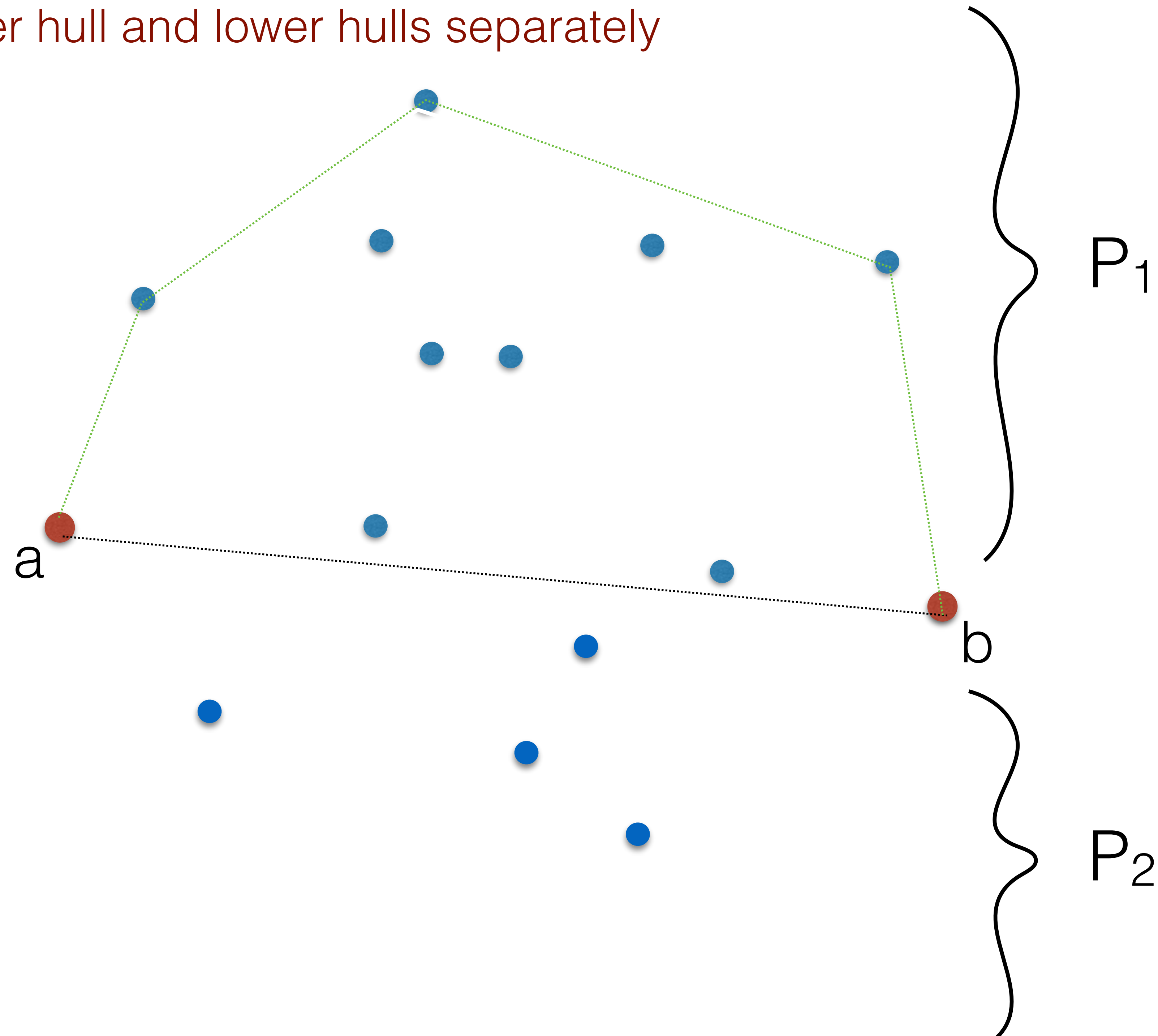
Andrew's Monotone Chain Algorithm (1979)

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



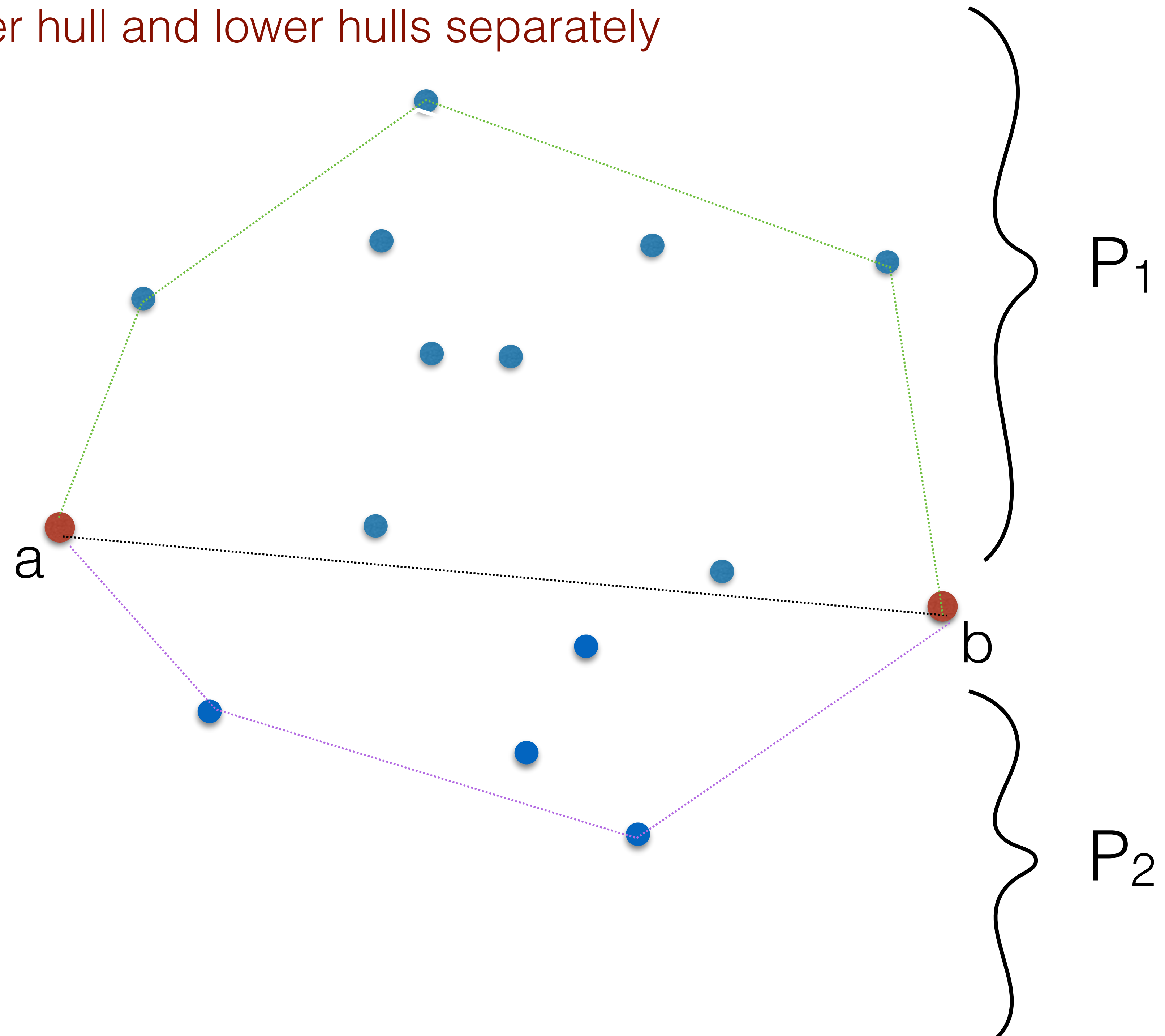
Andrew's Monotone Chain Algorithm (1979)

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



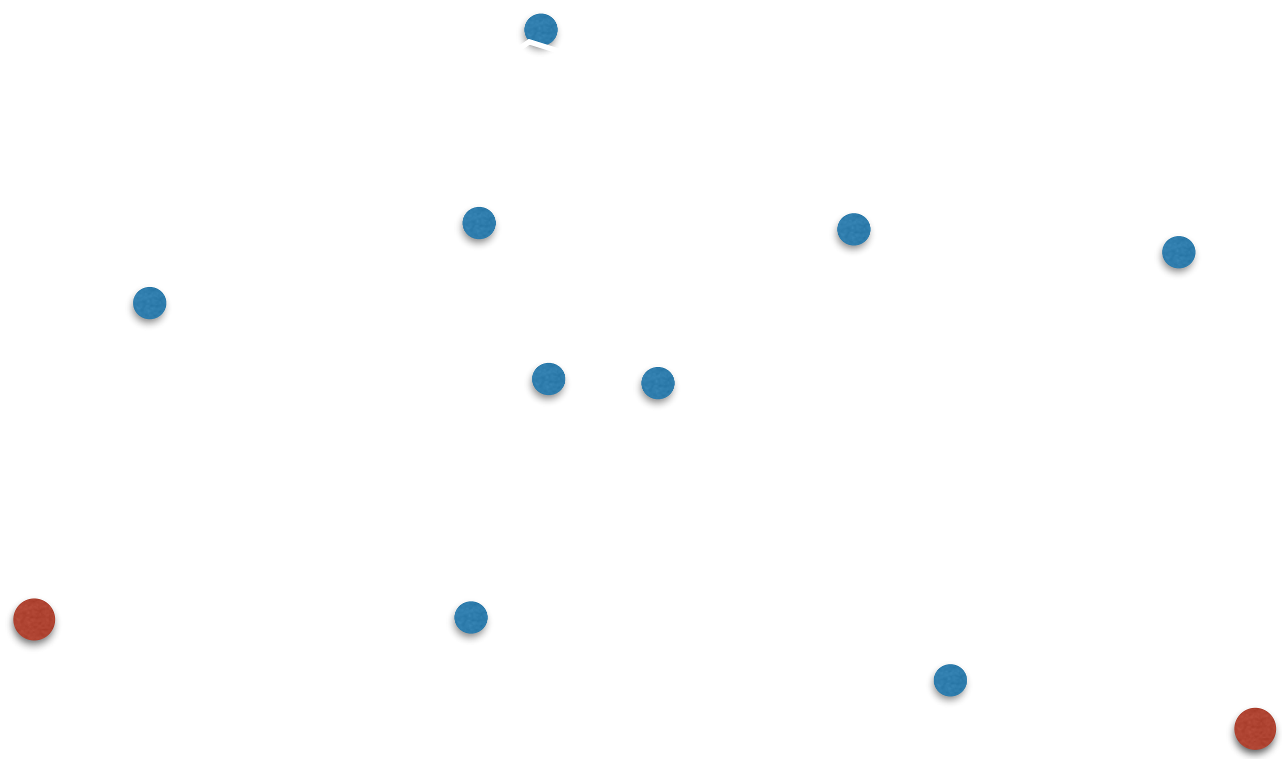
Andrew's Monotone Chain Algorithm (1979)

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



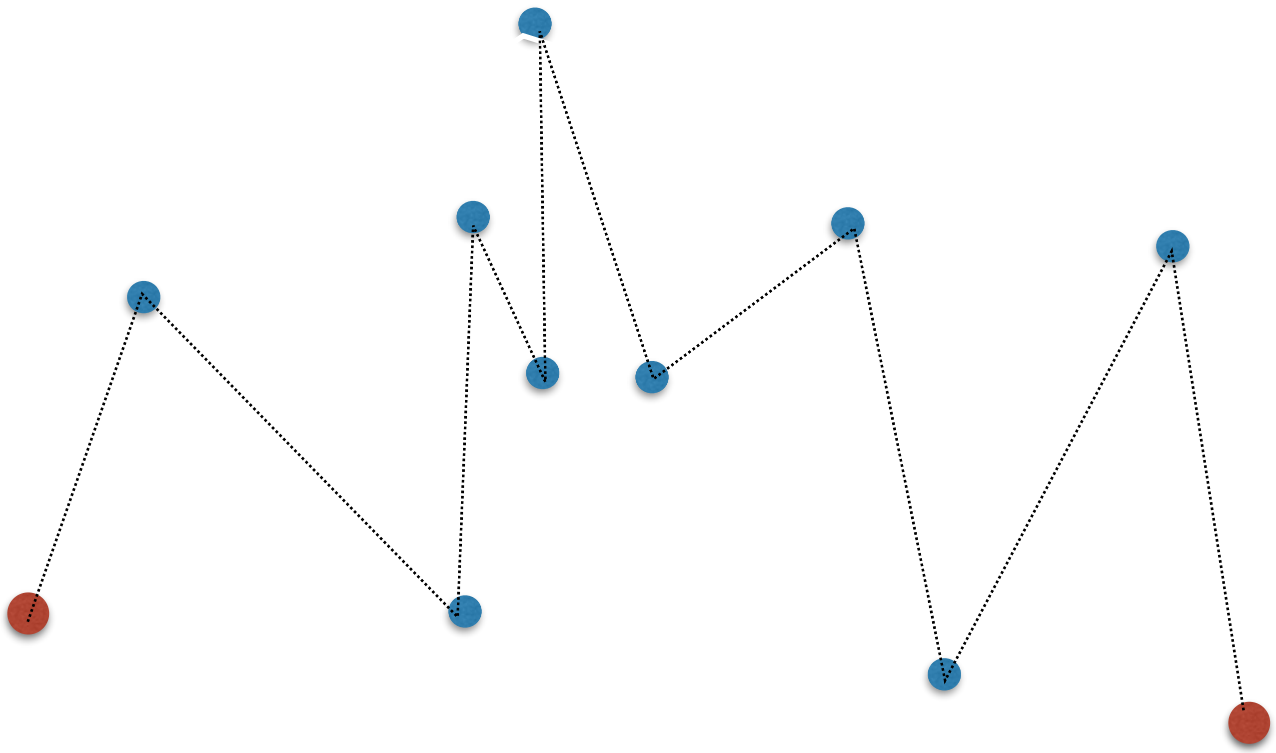
Andrew's Monotone Chain Algorithm (1979)

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



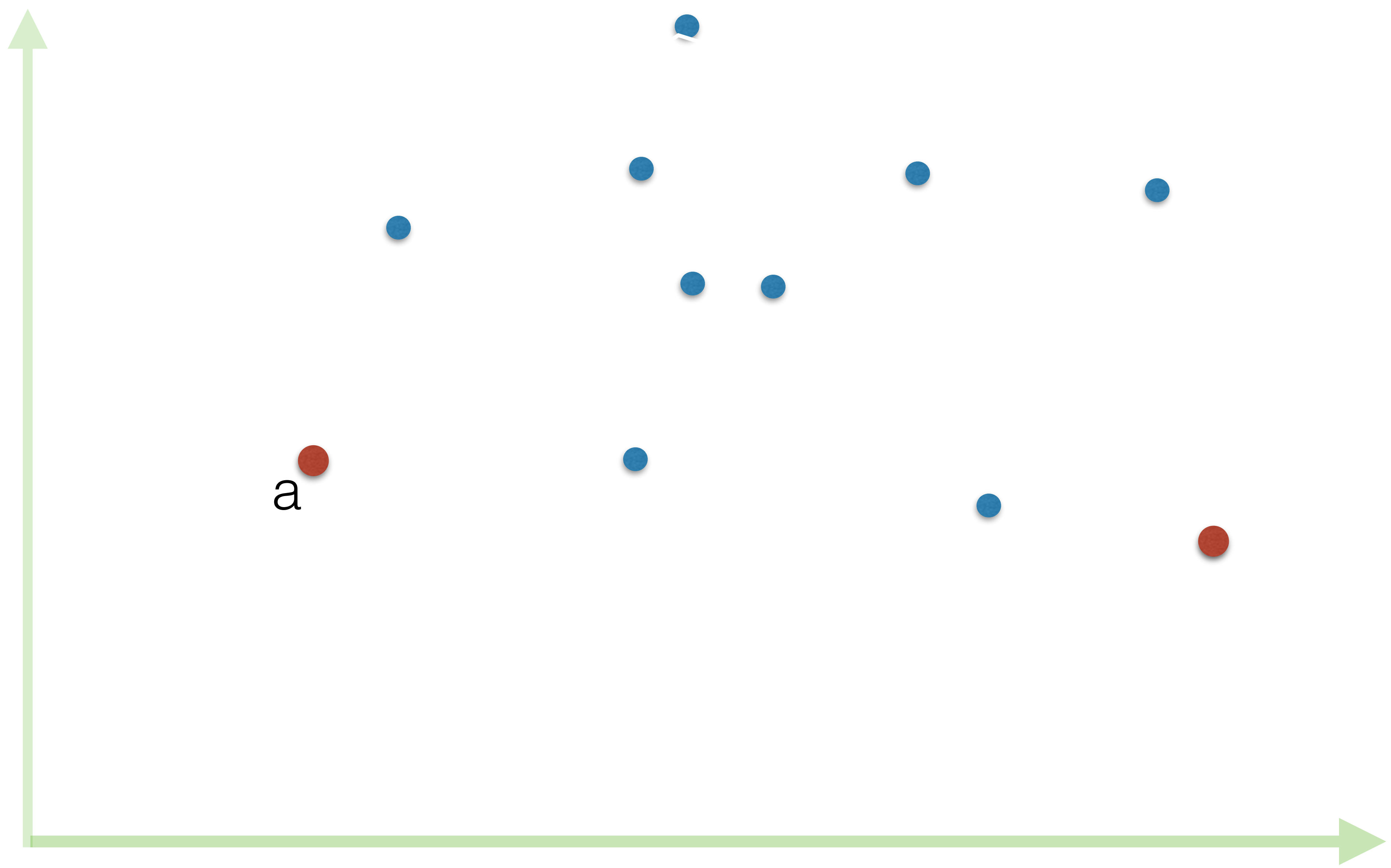
Andrew's Monotone Chain Algorithm (1979)

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



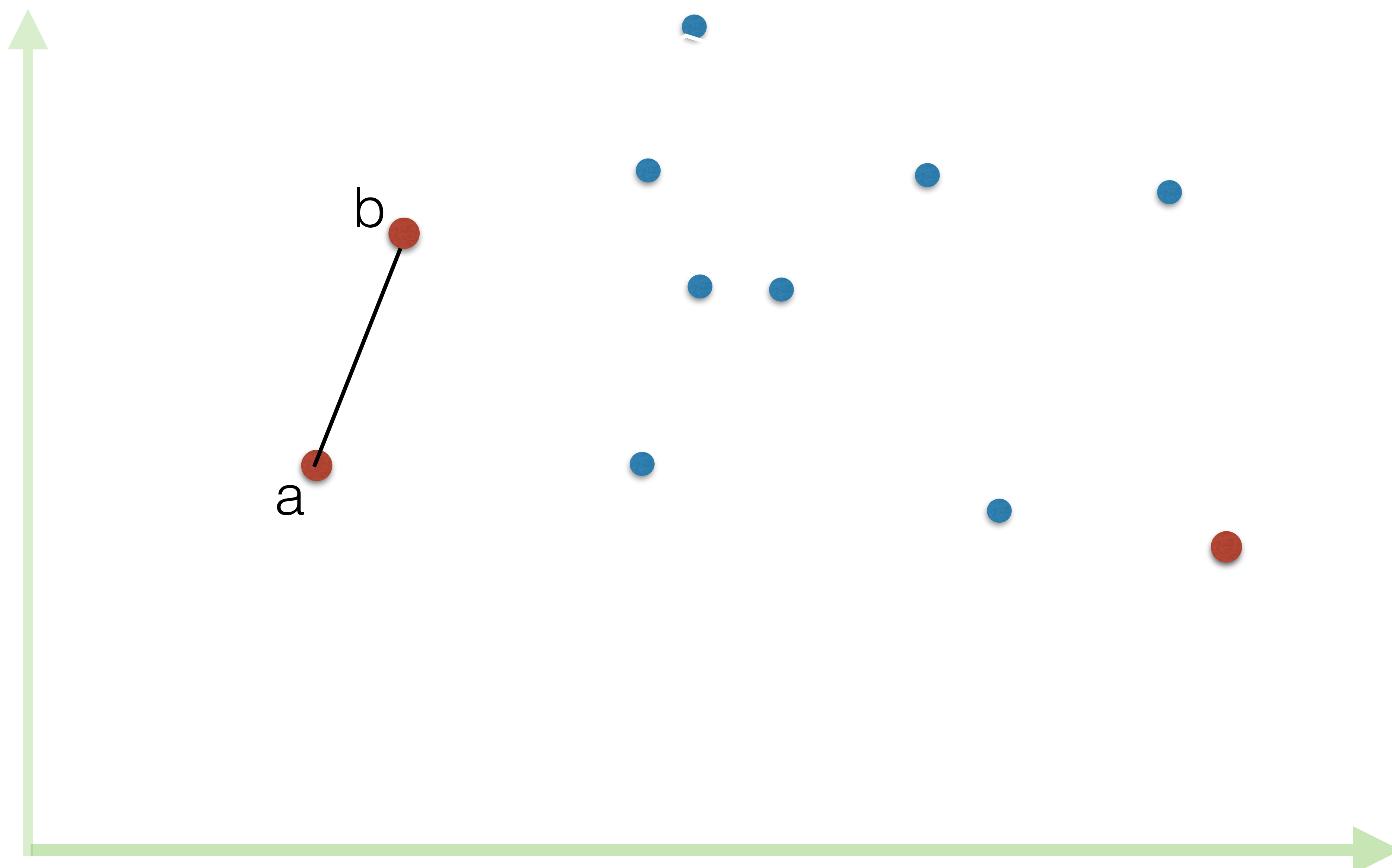
Andrew's Monotone Chain Algorithm (1979)

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



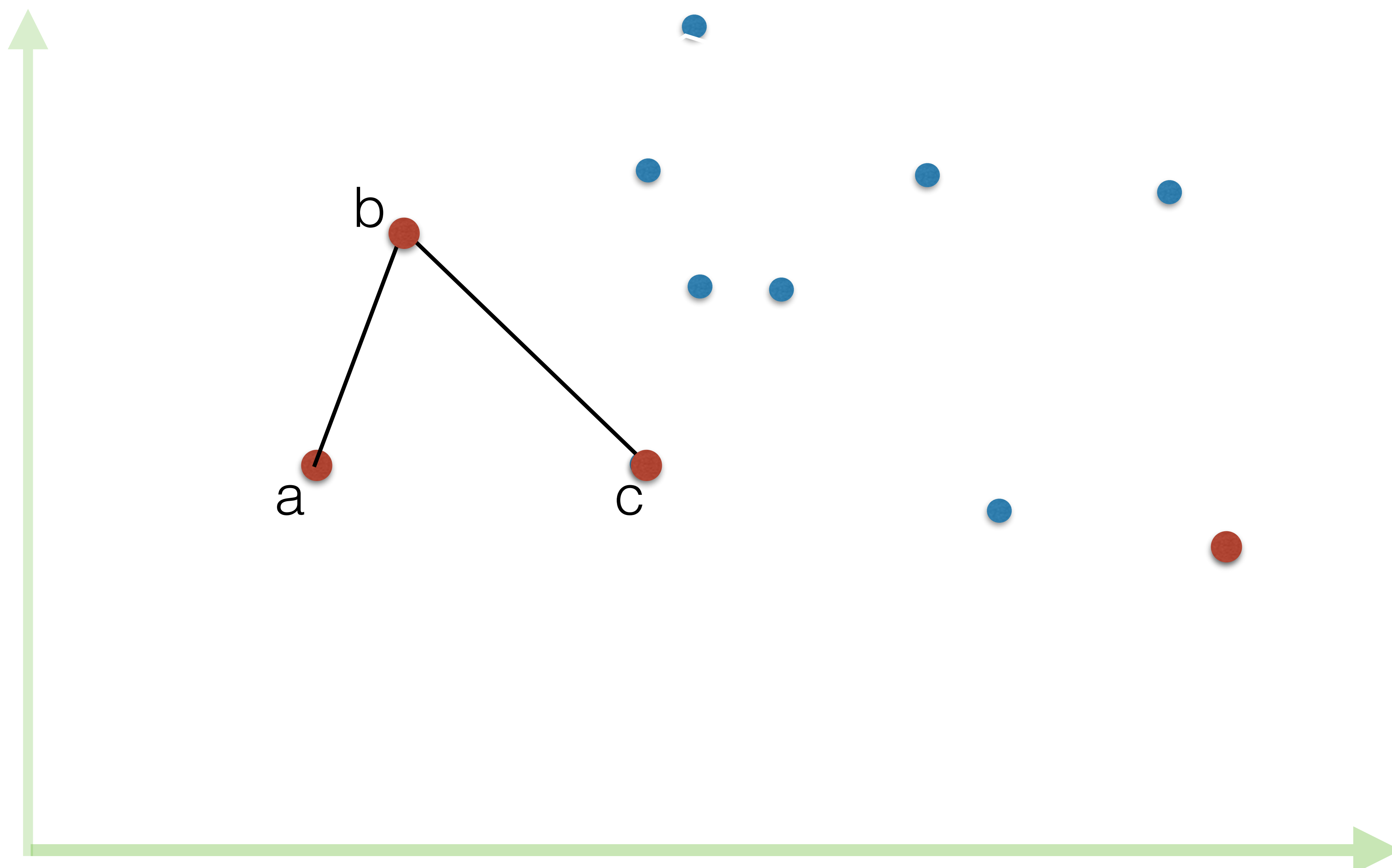
Andrew's Monotone Chain Algorithm (1979)

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



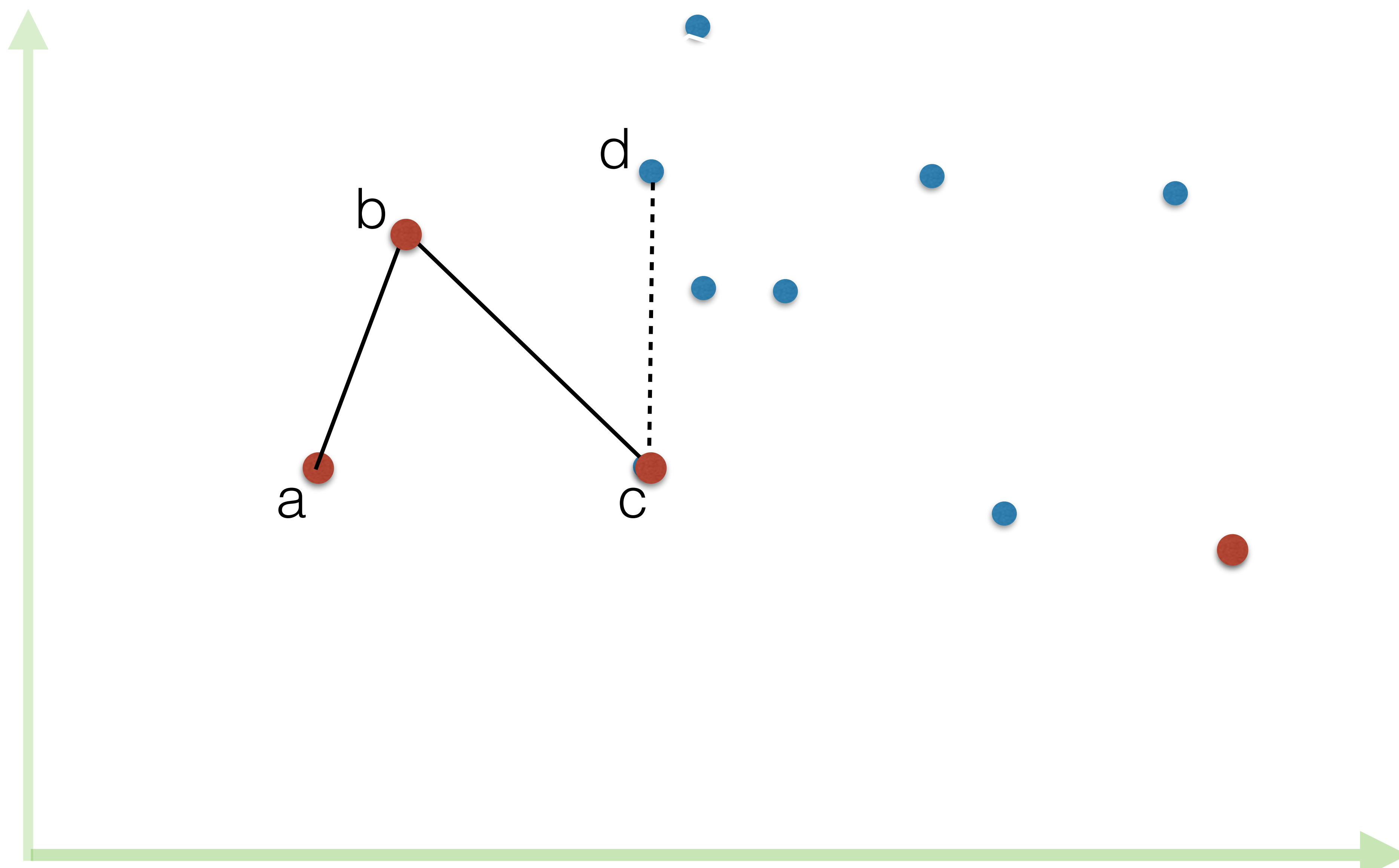
Andrew's Monotone Chain Algorithm (1979)

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



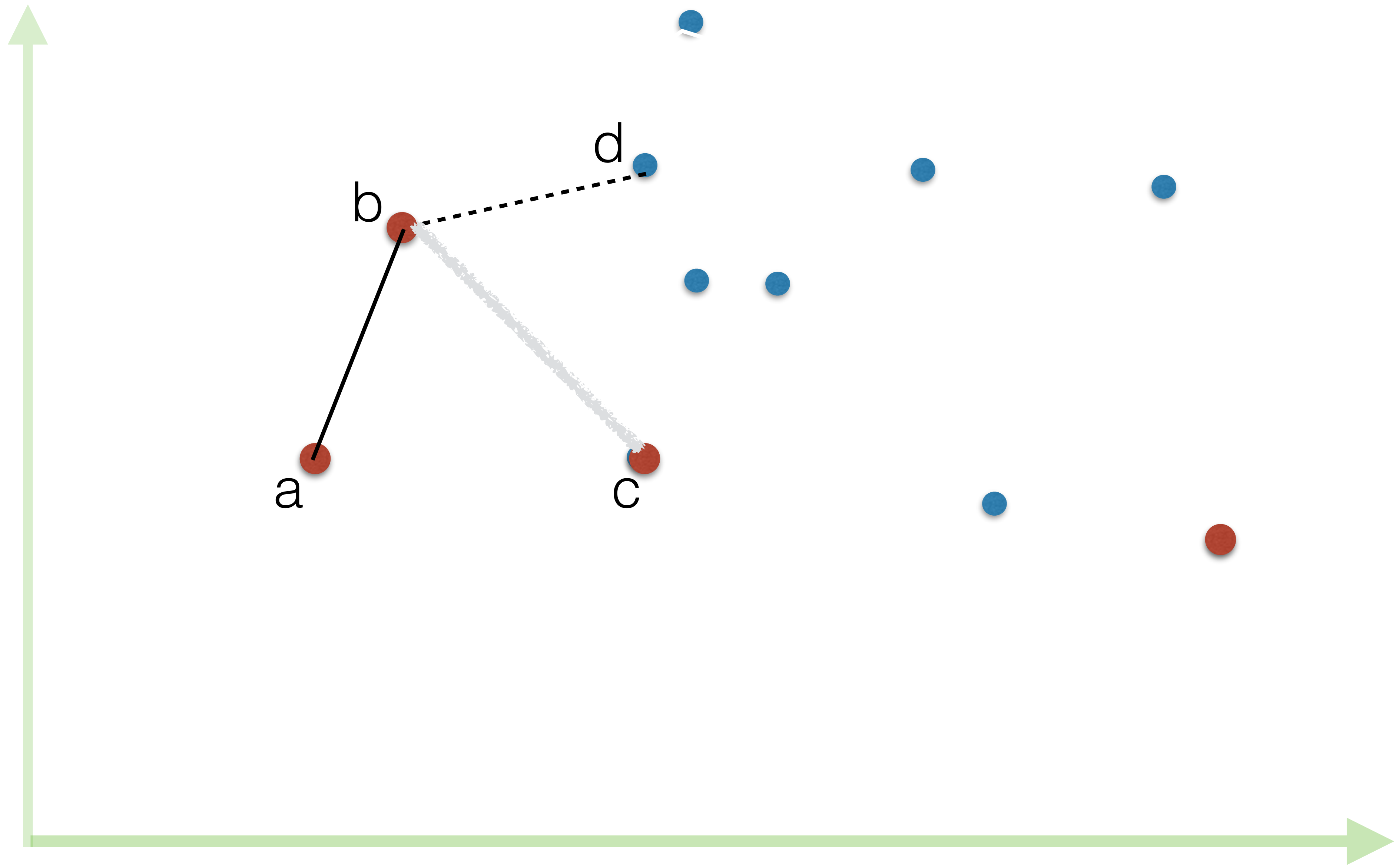
Andrew's Monotone Chain Algorithm (1979)

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



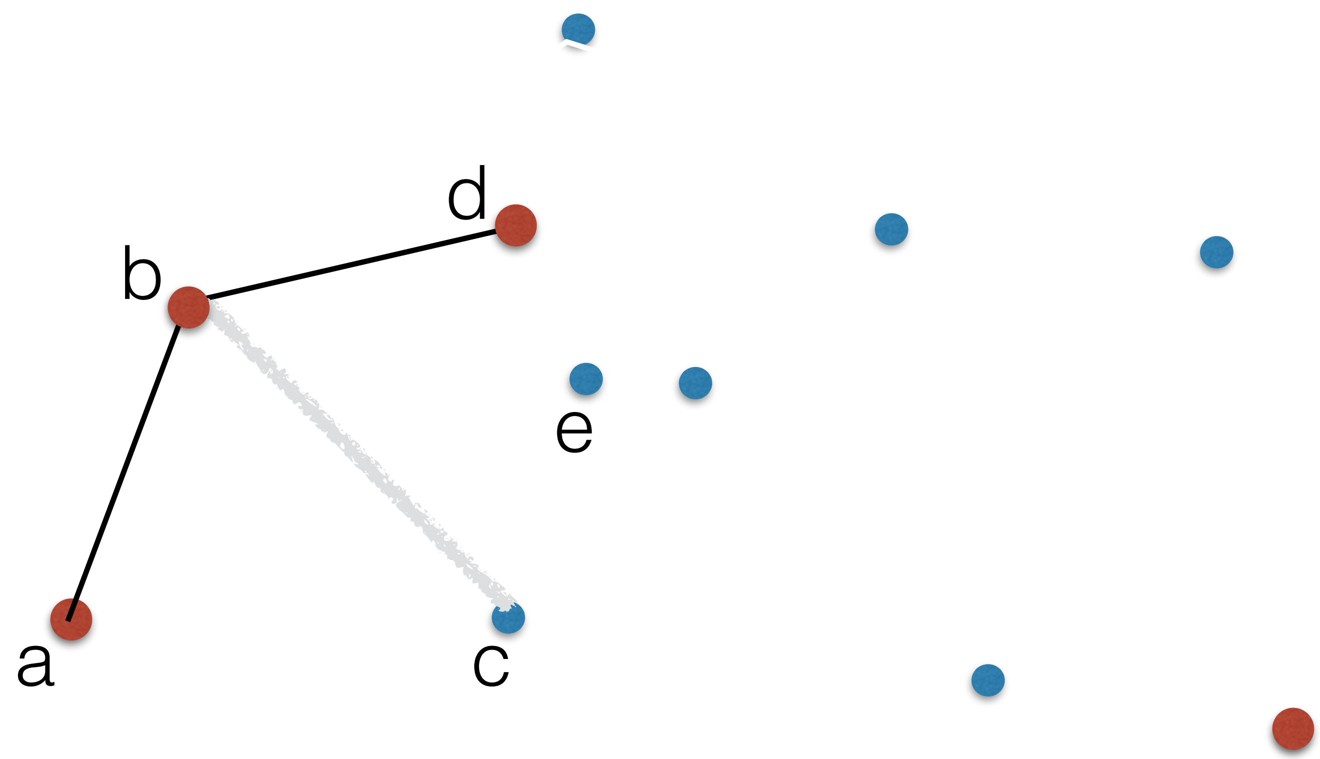
Andrew's Monotone Chain Algorithm (1979)

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



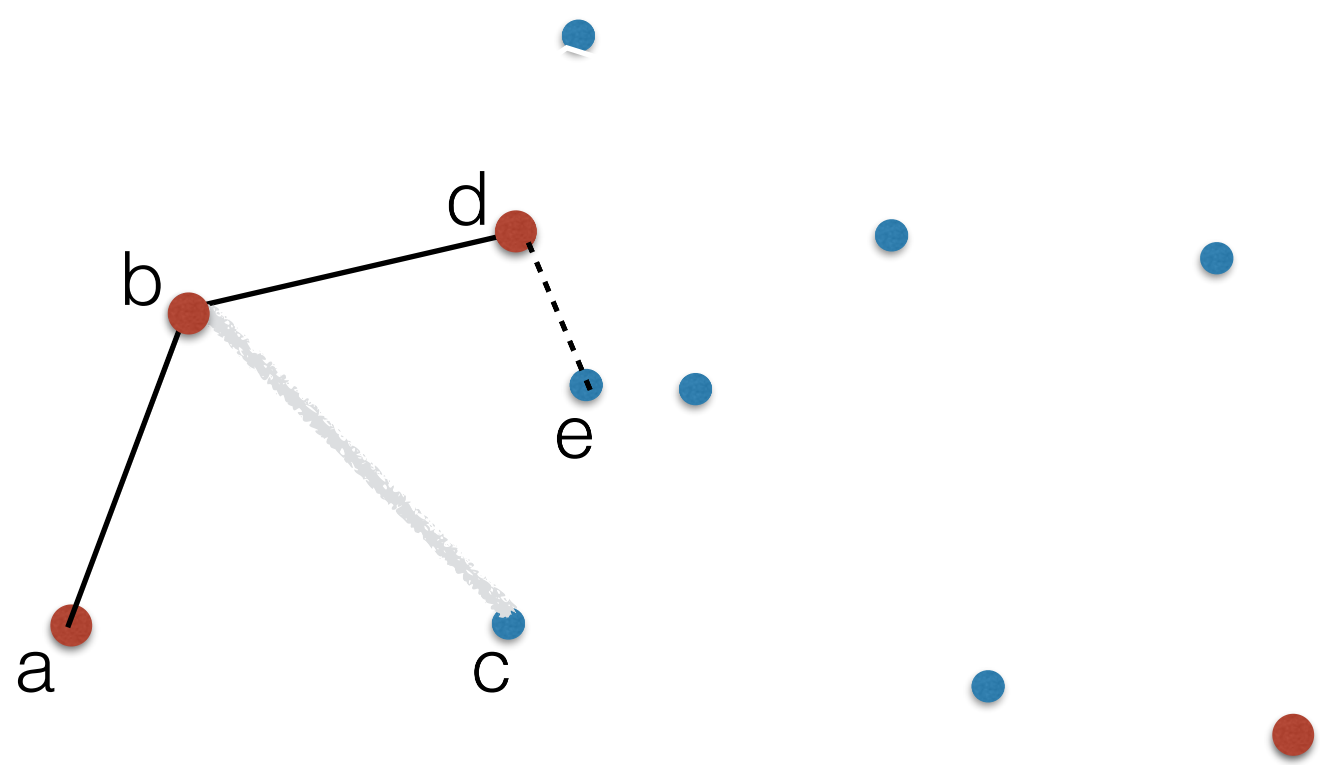
Andrew's Monotone Chain Algorithm (1979)

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



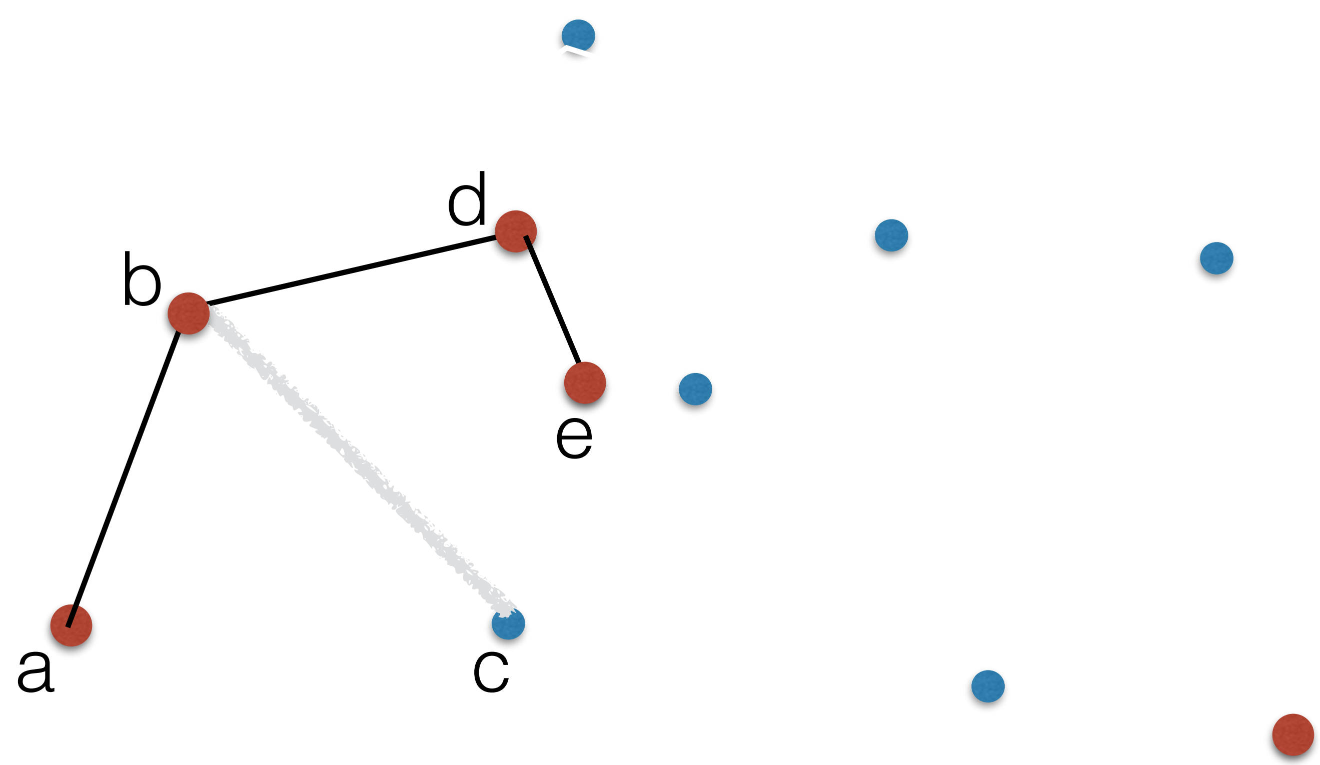
Andrew’s Monotone Chain Algorithm (1979)

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



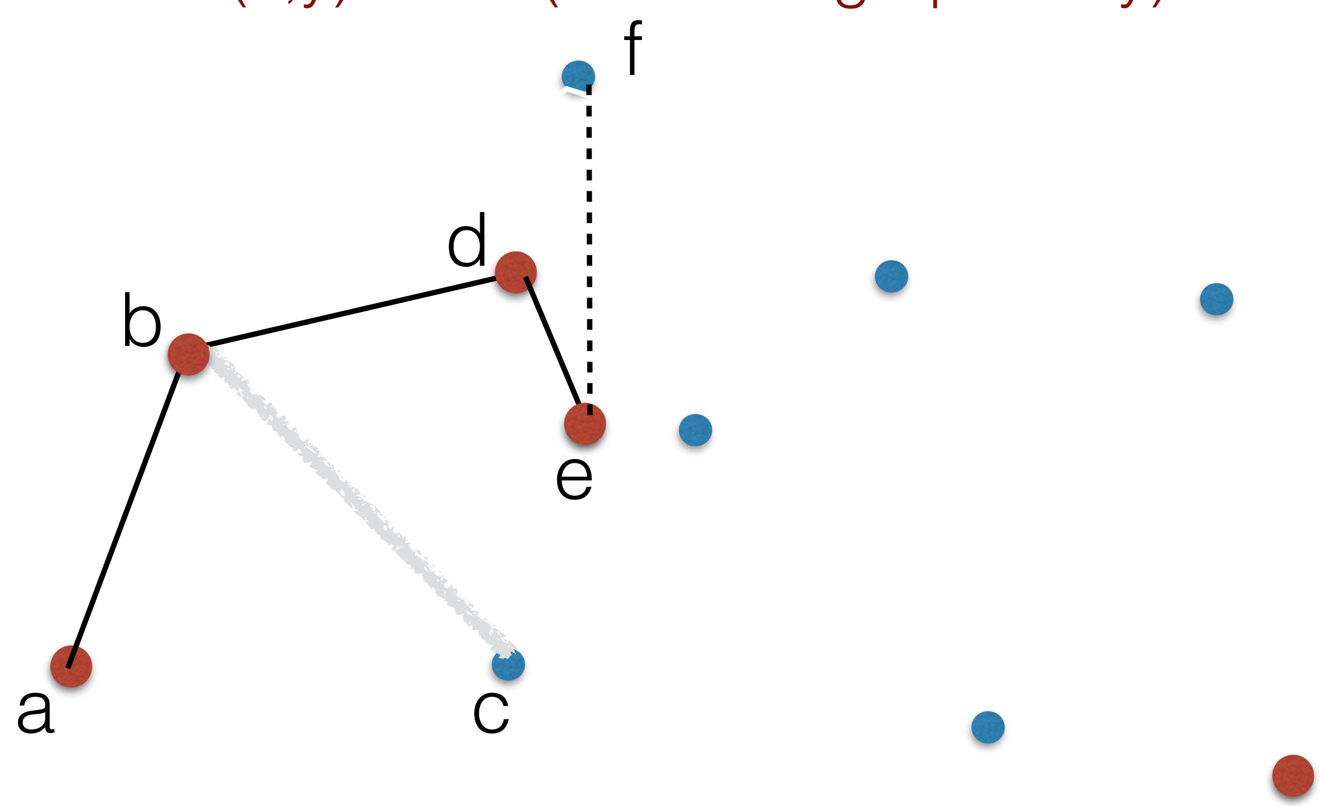
Andrew's Monotone Chain Algorithm (1979)

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



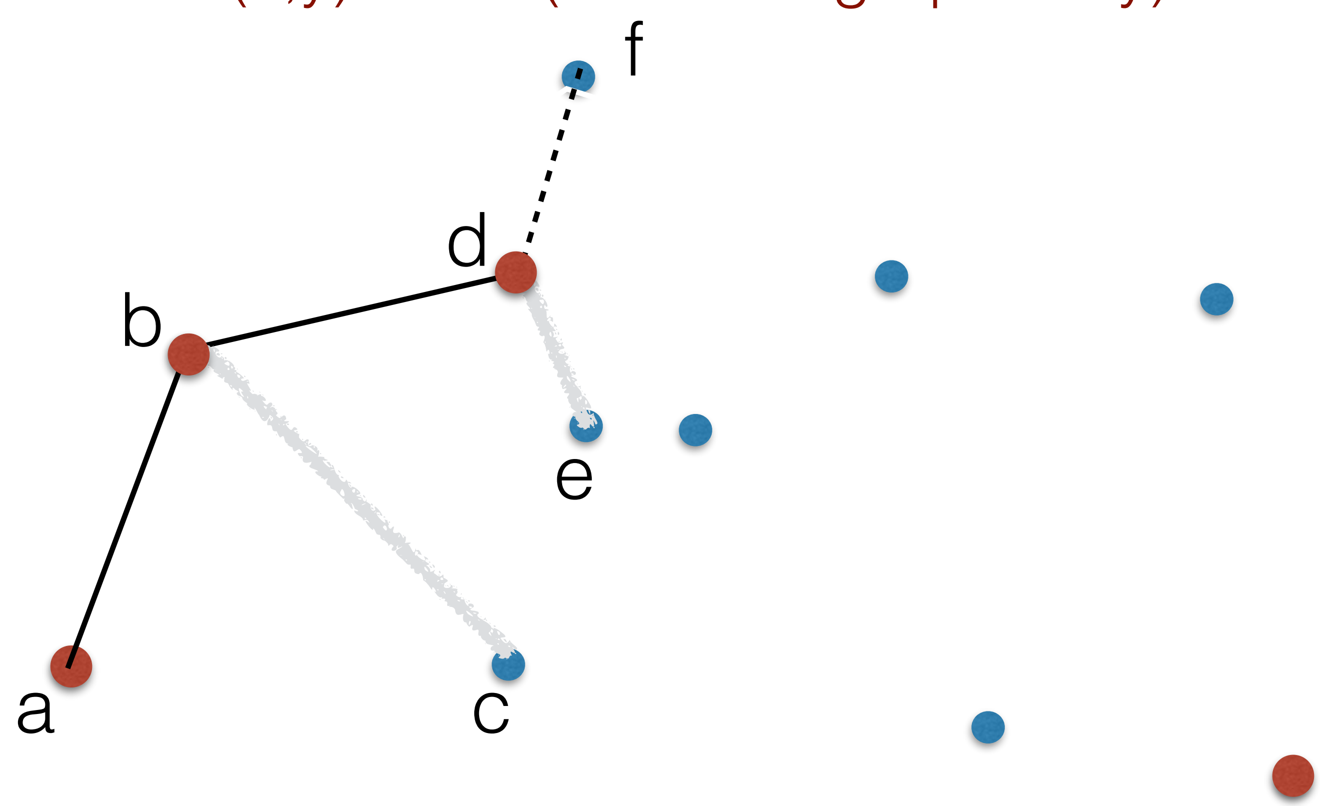
Andrew's Monotone Chain Algorithm (1979)

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



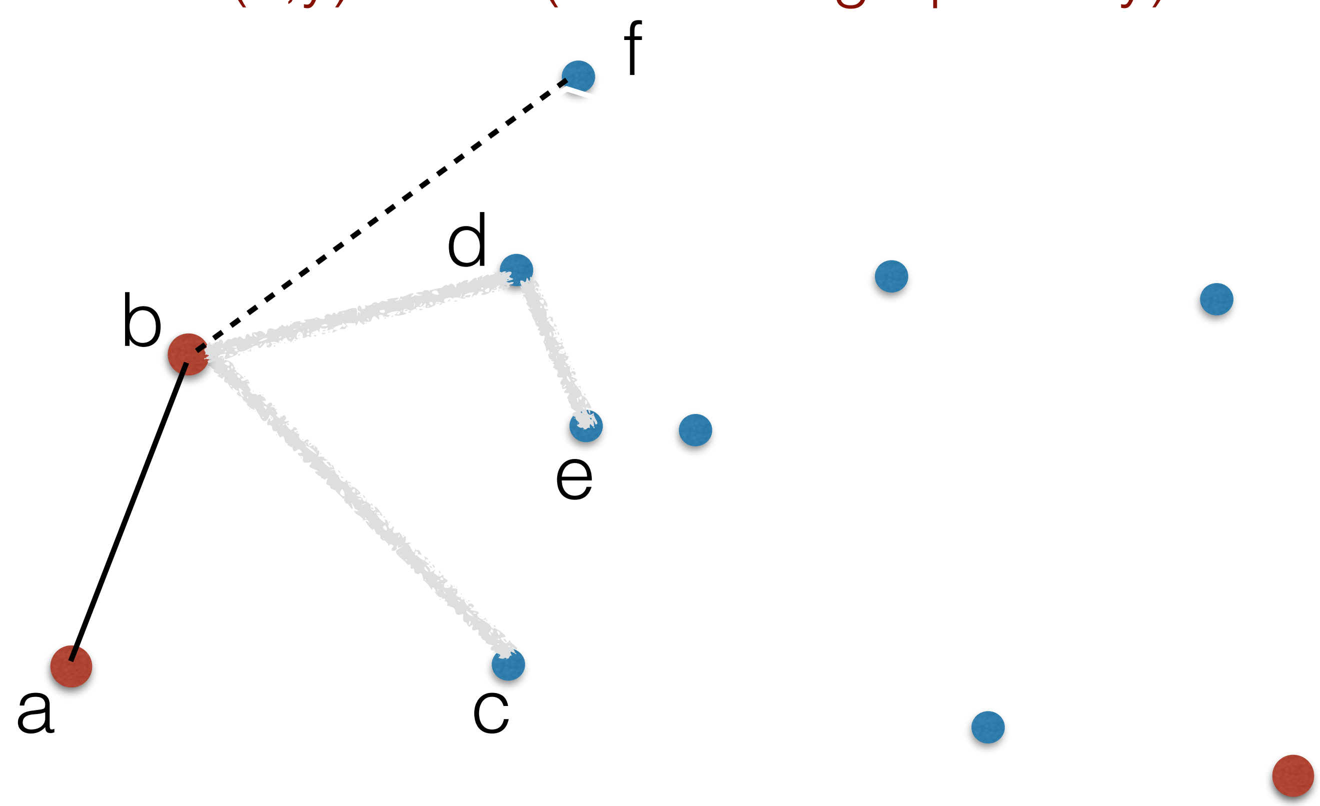
Andrew's Monotone Chain Algorithm (1979)

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



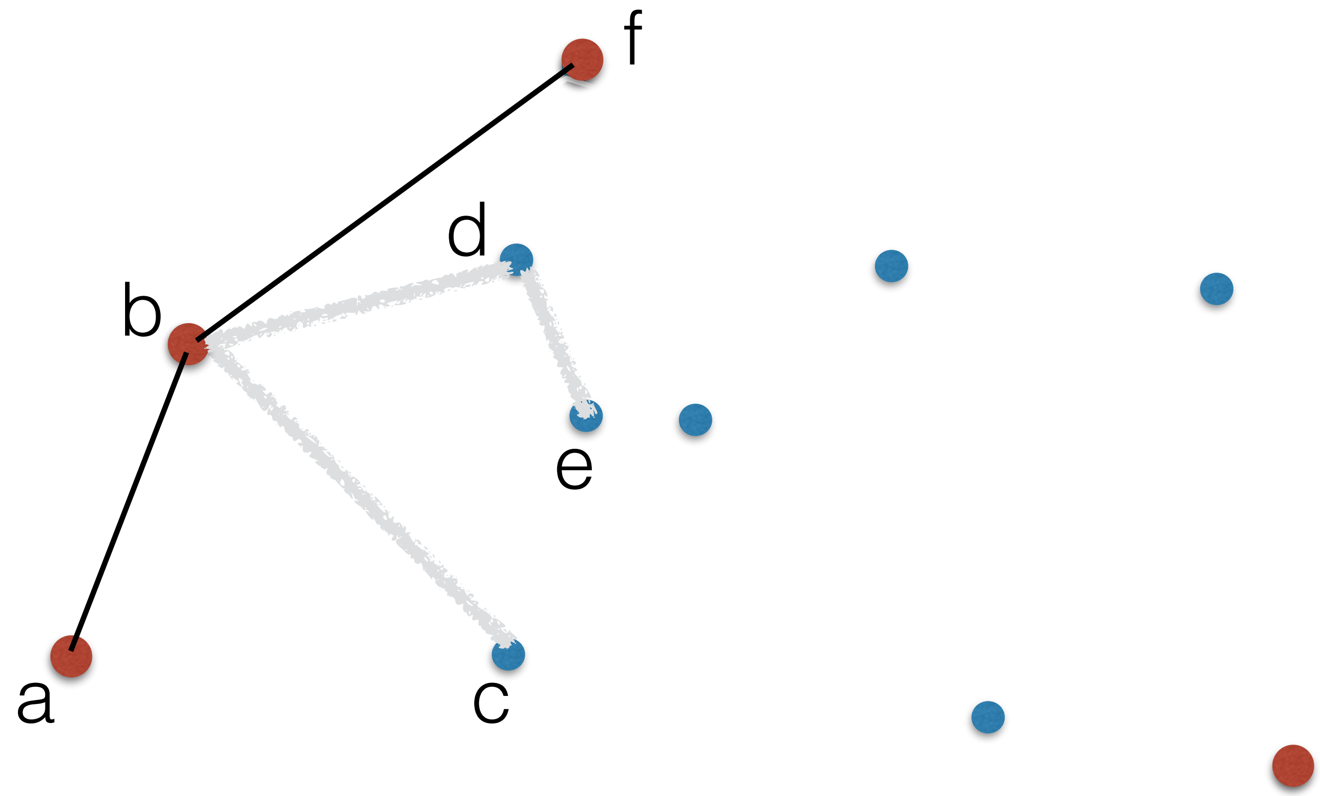
Andrew's Monotone Chain Algorithm (1979)

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



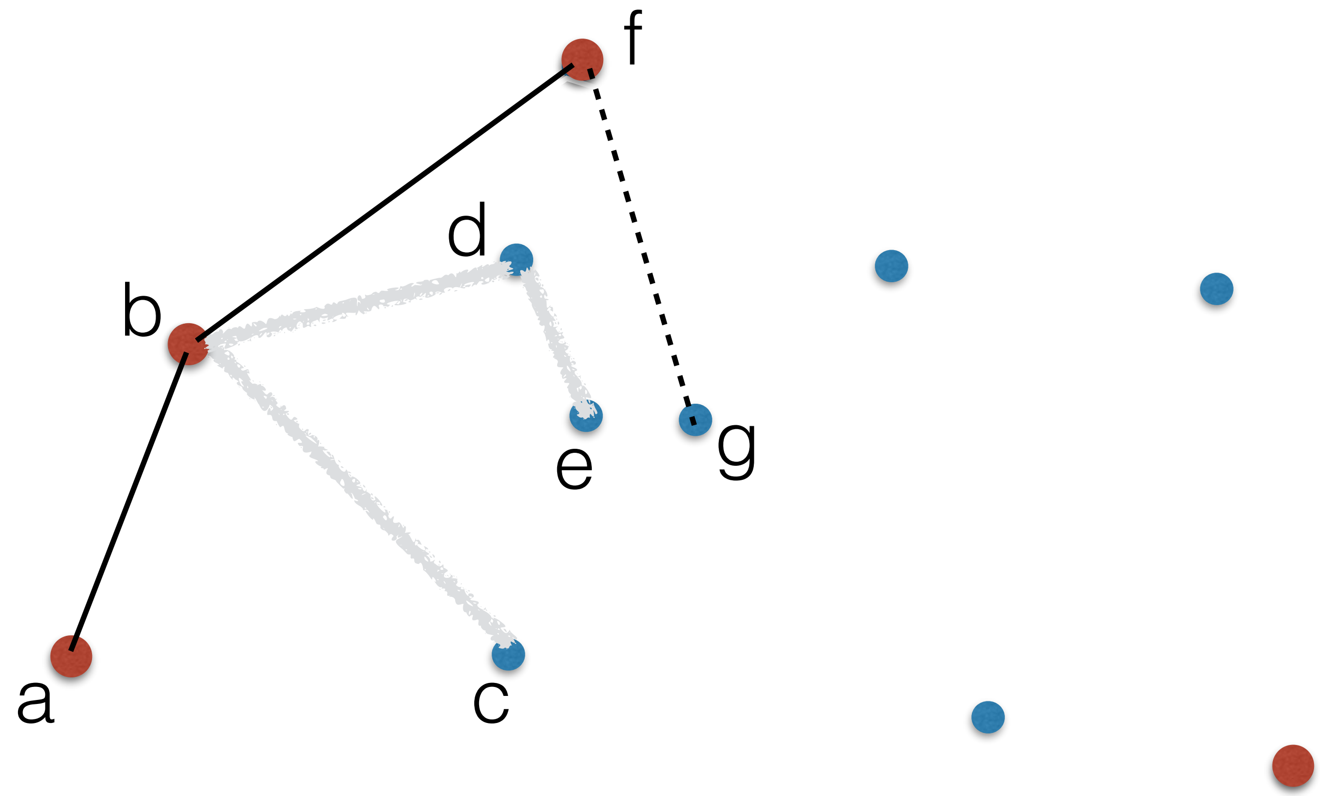
Andrew's Monotone Chain Algorithm (1979)

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



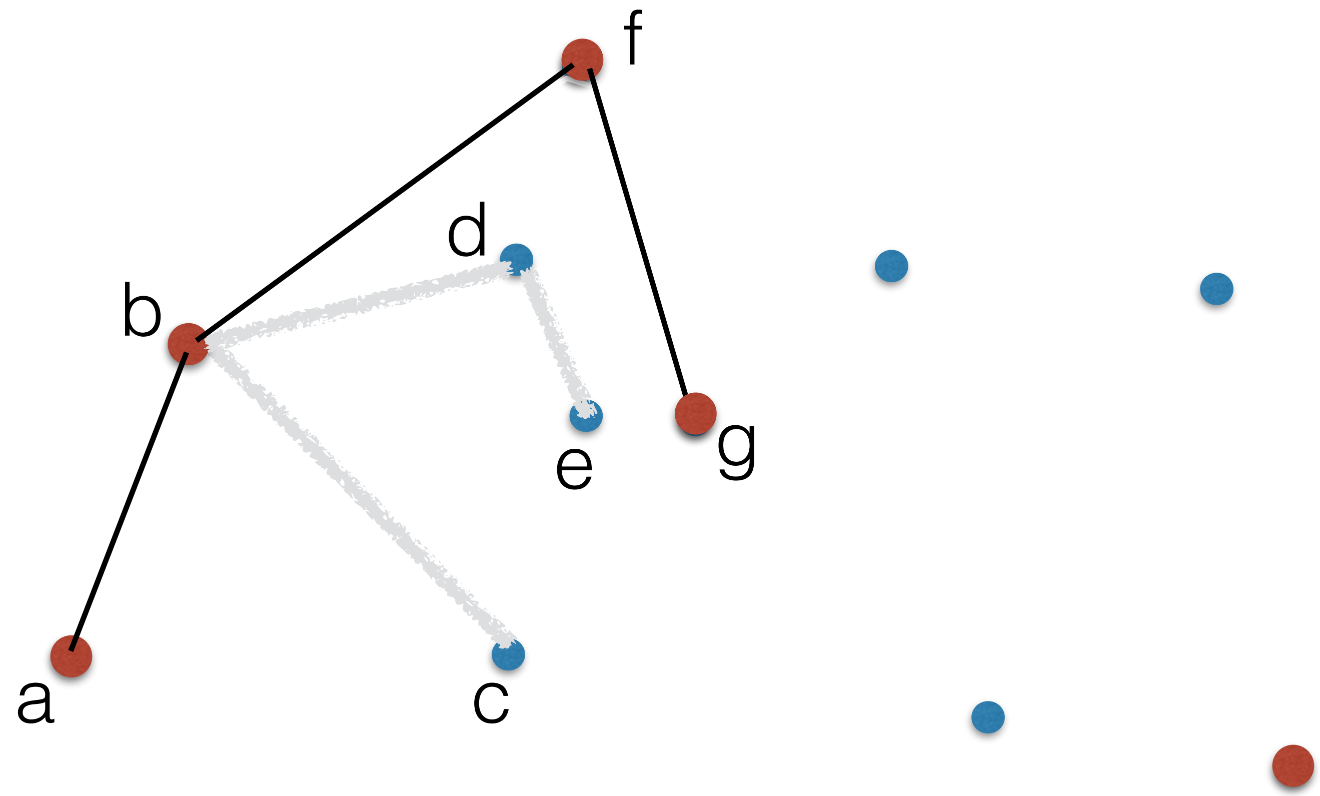
Andrew's Monotone Chain Algorithm (1979)

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



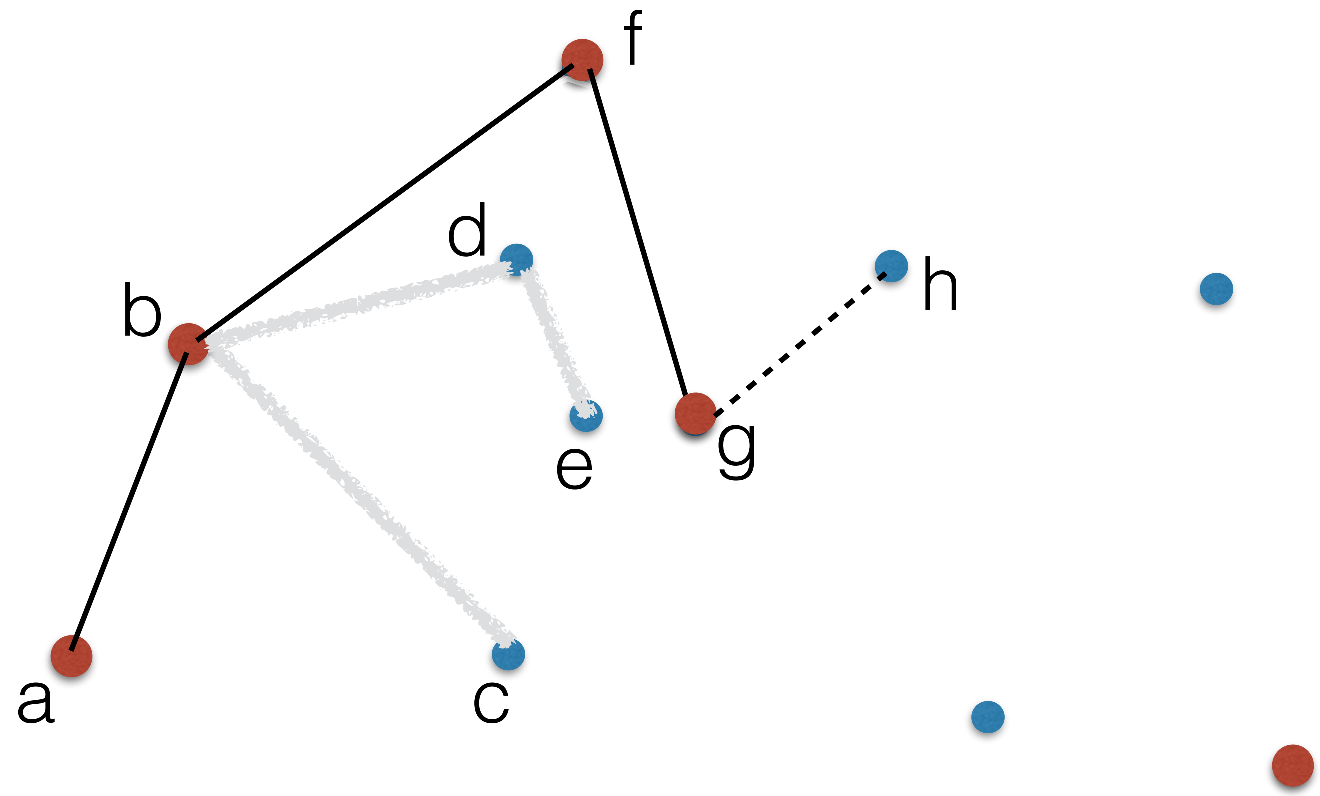
Andrew’s Monotone Chain Algorithm (1979)

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



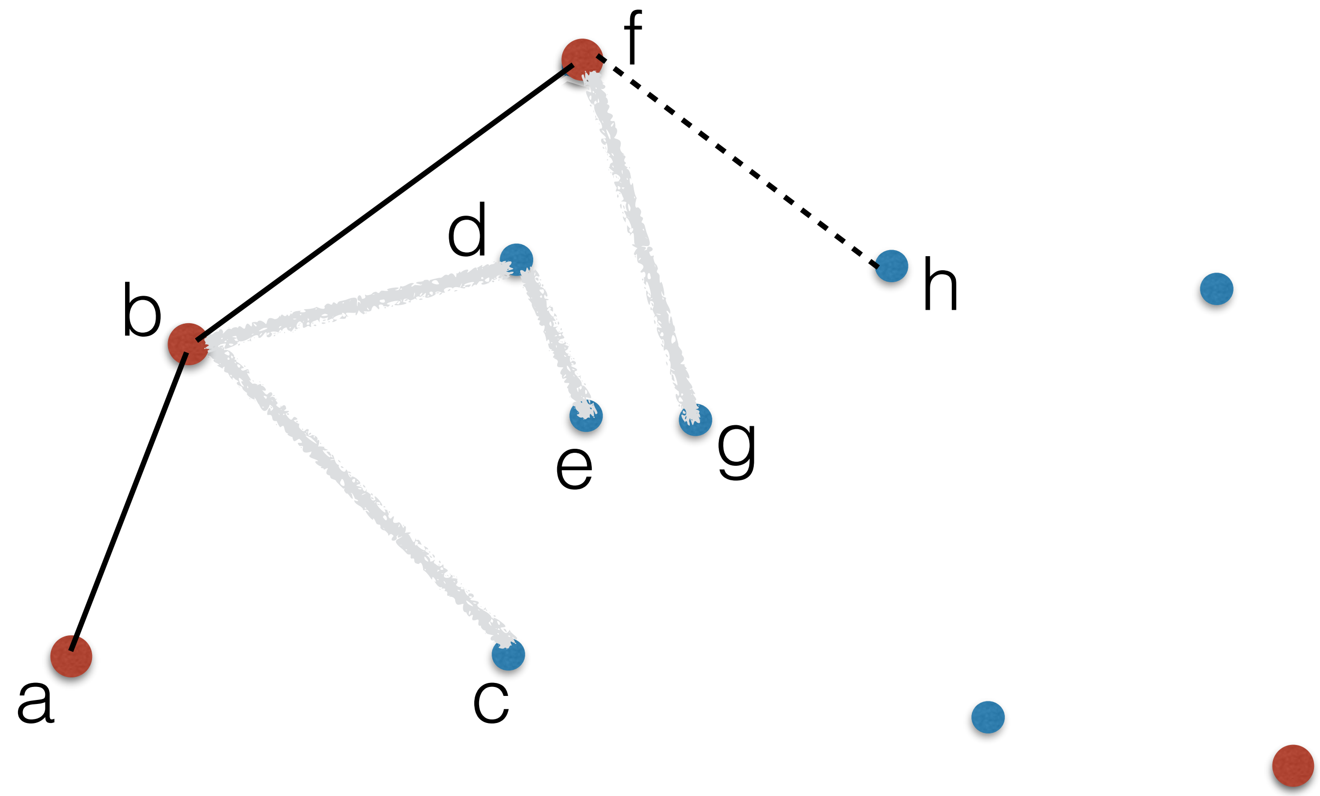
Andrew's Monotone Chain Algorithm (1979)

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



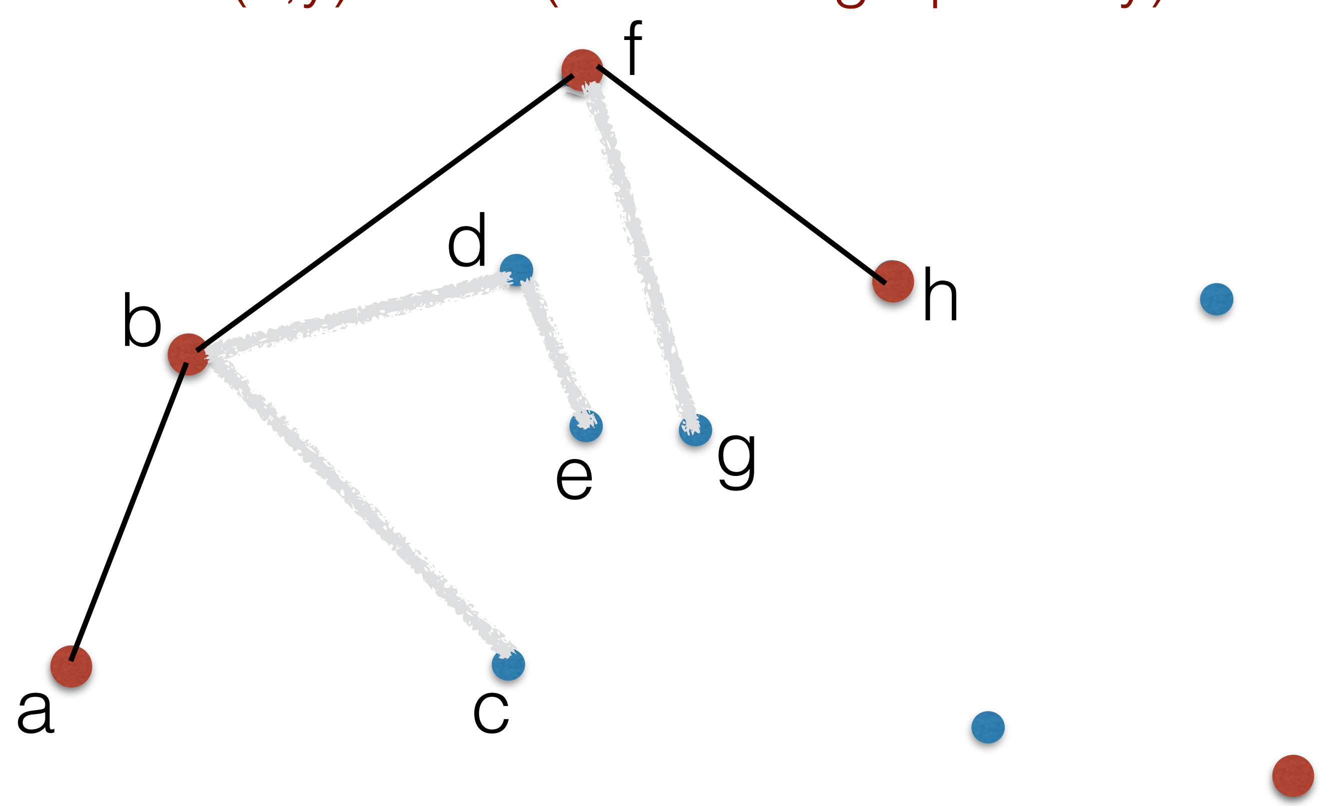
Andrew's Monotone Chain Algorithm (1979)

- 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	$O(nh)$	1973
Graham scan	$O(n \lg n)$	1972
Andrew monotone	$O(n \lg n)$	1979
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

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

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

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

- A lower bound for CH: What is the **worst-case running time** of the **best possible** CH algorithm?

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

Consider **all** possible CH algorithms A

Take the overall smallest worst-case running time

What is a lower bound?

- Lower bounds depend on the machine model.
  - The standard model is the decision tree (comparison) model.
  - Sorting lower bound in the decision tree model is $\Omega(n \lg n)$.

How do we prove lower bounds?

- Prove directly
  - Theorem: Any sorting algorithm that uses only comparisons uses at least $\Omega(n \lg n)$ comparisons in the worst case.
  - Proof: We saw this in Algorithms..
- Or via **reduction** from a problem known to have a lower bound
  - We'll use this to show that any algorithm for ConvexHull must have worst-case complexity $\Omega(n \lg n)$

Lower bounds by reduction

- We know that $\Omega(n \lg n) \leq$ Sorting
- If we could show that ConvexHull is at least as hard as Sorting

Sorting $\leq$ Convex hull

This would imply that ConvexHull is $\Omega(n \lg n)$

How do we show Sorting $\leq$ Convex hull ?

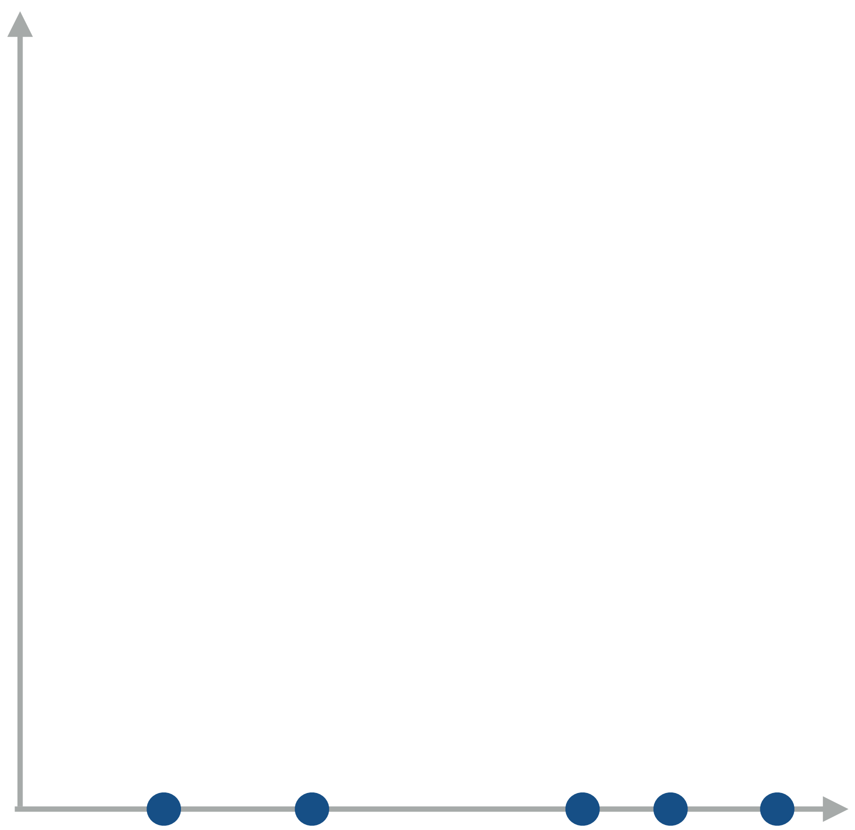
- We want to show that ConvexHull gives an upper bound to Sorting. This would be true if we could solve Sorting via ConvexHull.

- We'll show that we can use ConvexHull to Sort: Let P be a set of values that need to be sorted. We'll show that 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 ConvexHull(P')
  • use the convex hull to infer sorted order of P
```

Sorting via ConvexHull

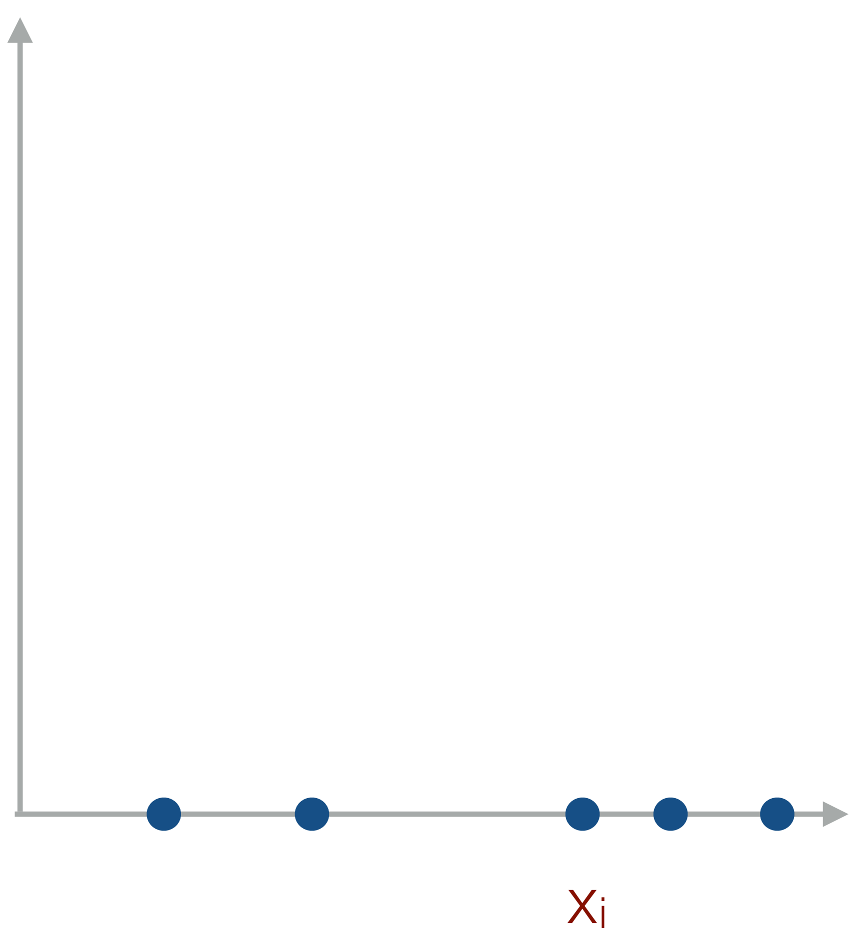
- Let P: set of values x_1, x_2, …x_n. to sort

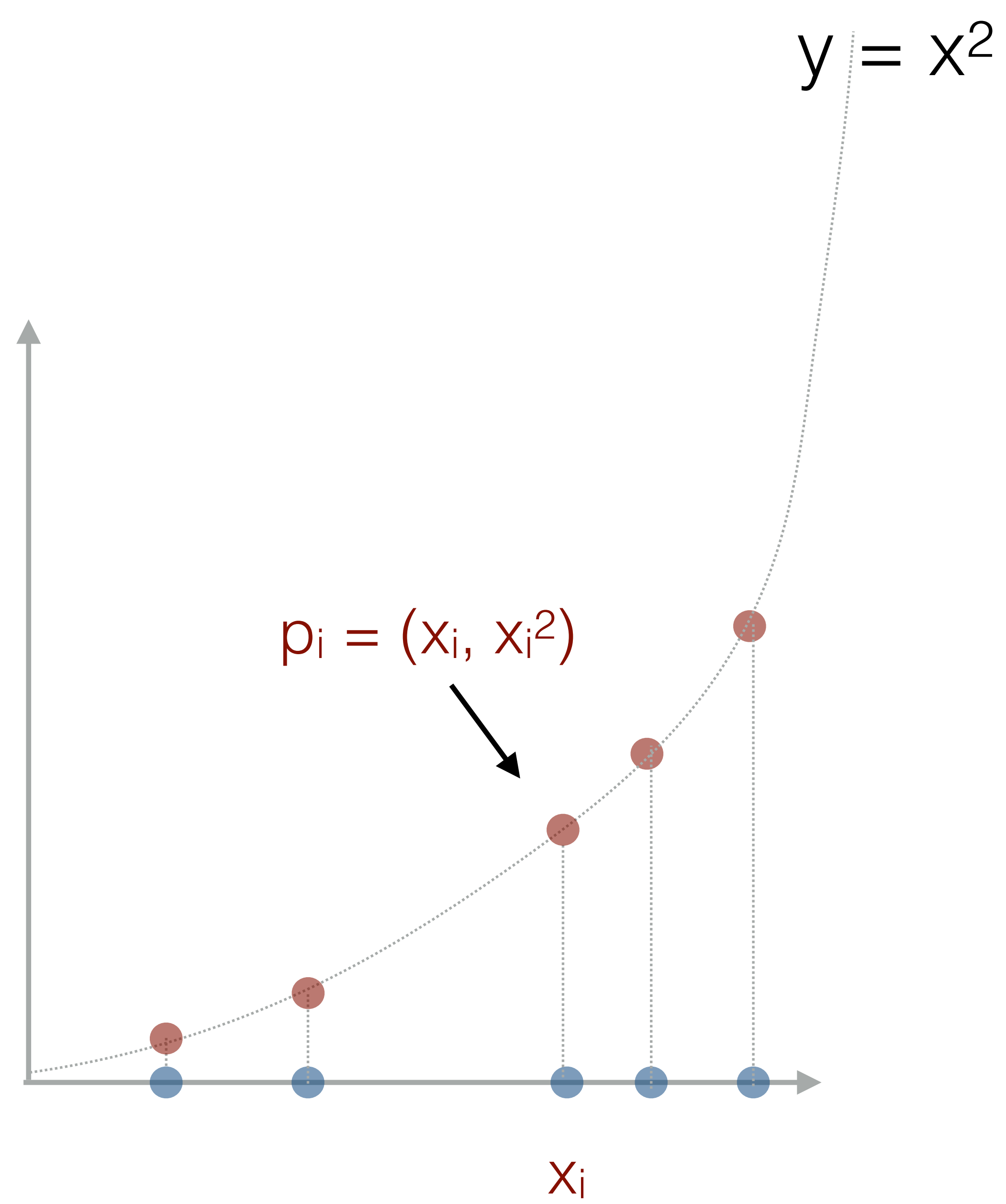


Our goal is to argue that there exists some instance of a convex hull problem that sorts our numbers.

Sorting via ConvexHull

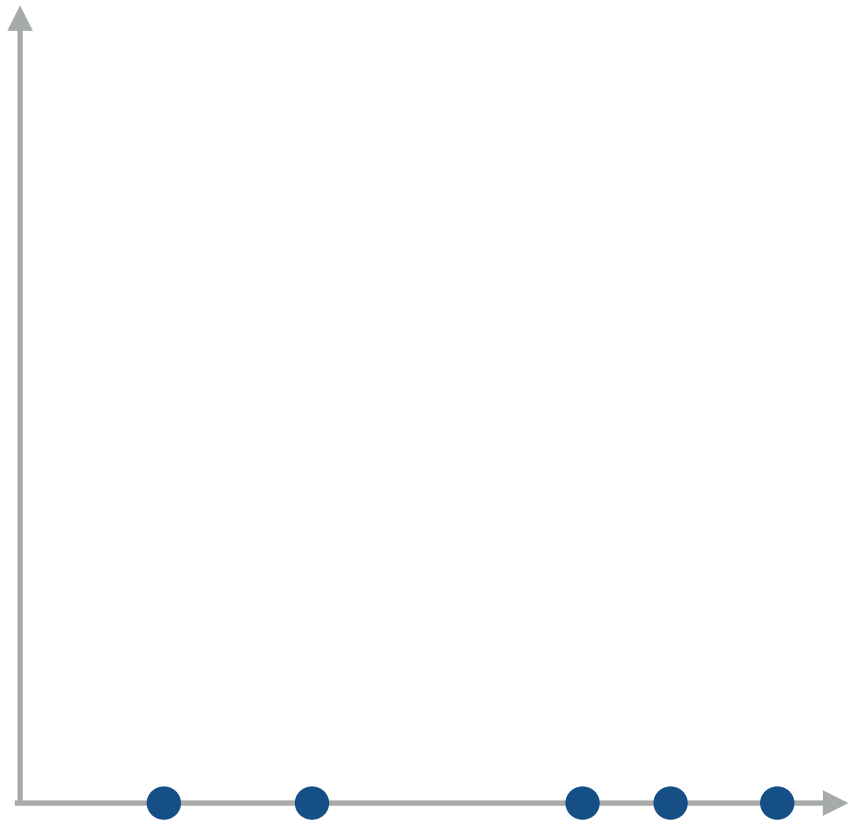
- Let P: set of values x_1, x_2, …x_n. to sort
- Let P’: set points { $p_i = (x_i, x_i^2)$}



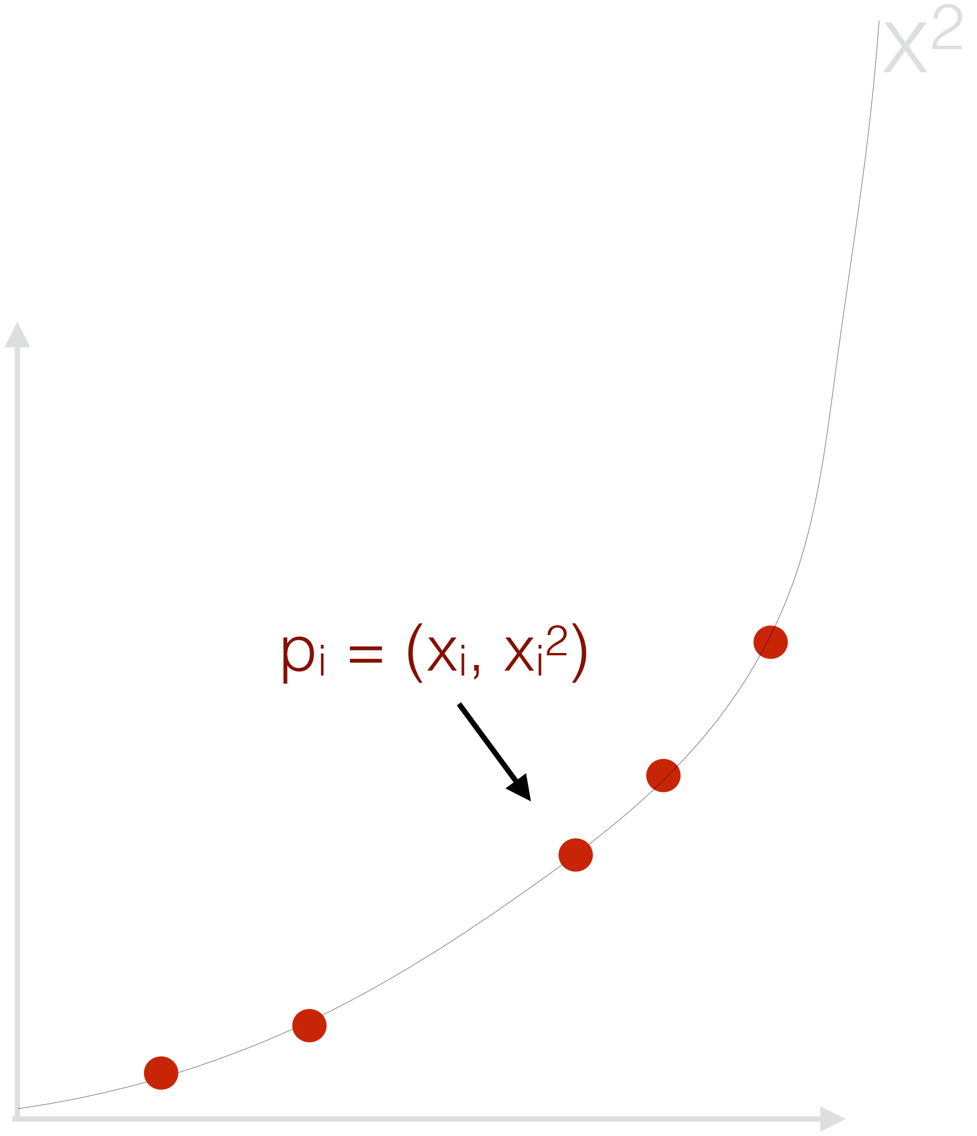


Sorting via ConvexHull

- Let P: set of values $x_1, x_2, \ldots x_n$. to sort



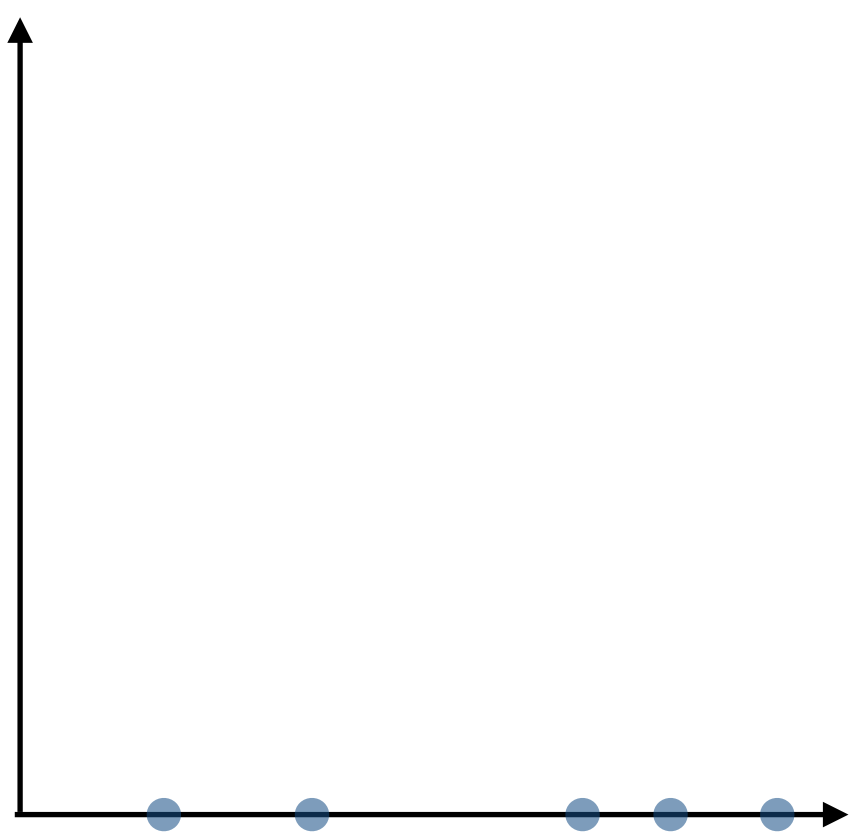
- Let P': set points $\{ p_i = (x_i, x_i^2)\}$

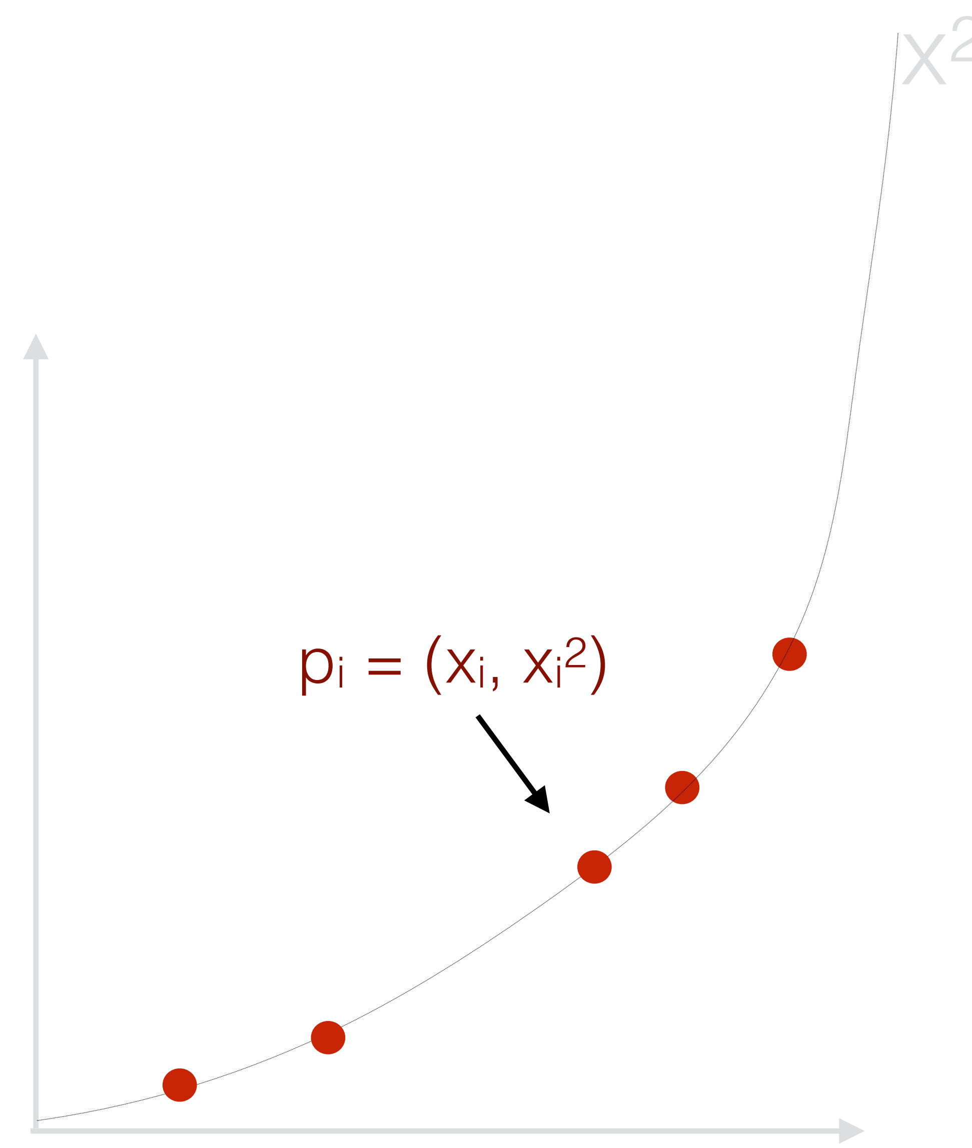


Sorting via ConvexHull

- Let P: set of values x_1, x_2, …x_n. to sort

- Let P': set points { $p_i = (x_i, x_i^2)$}
- Run CH(P') to find their convex hull



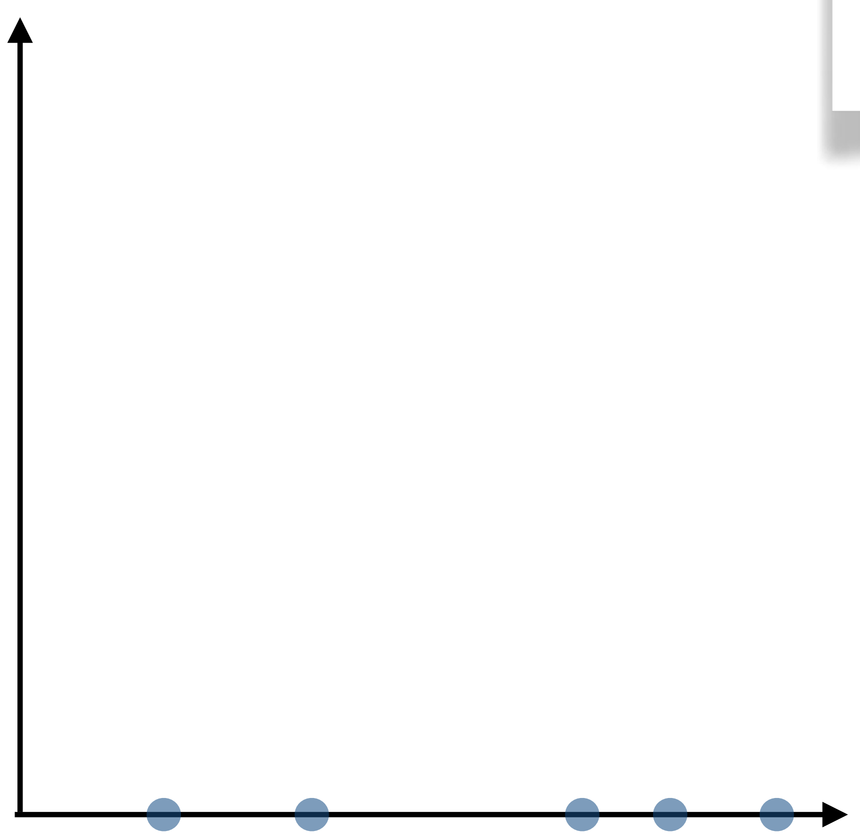


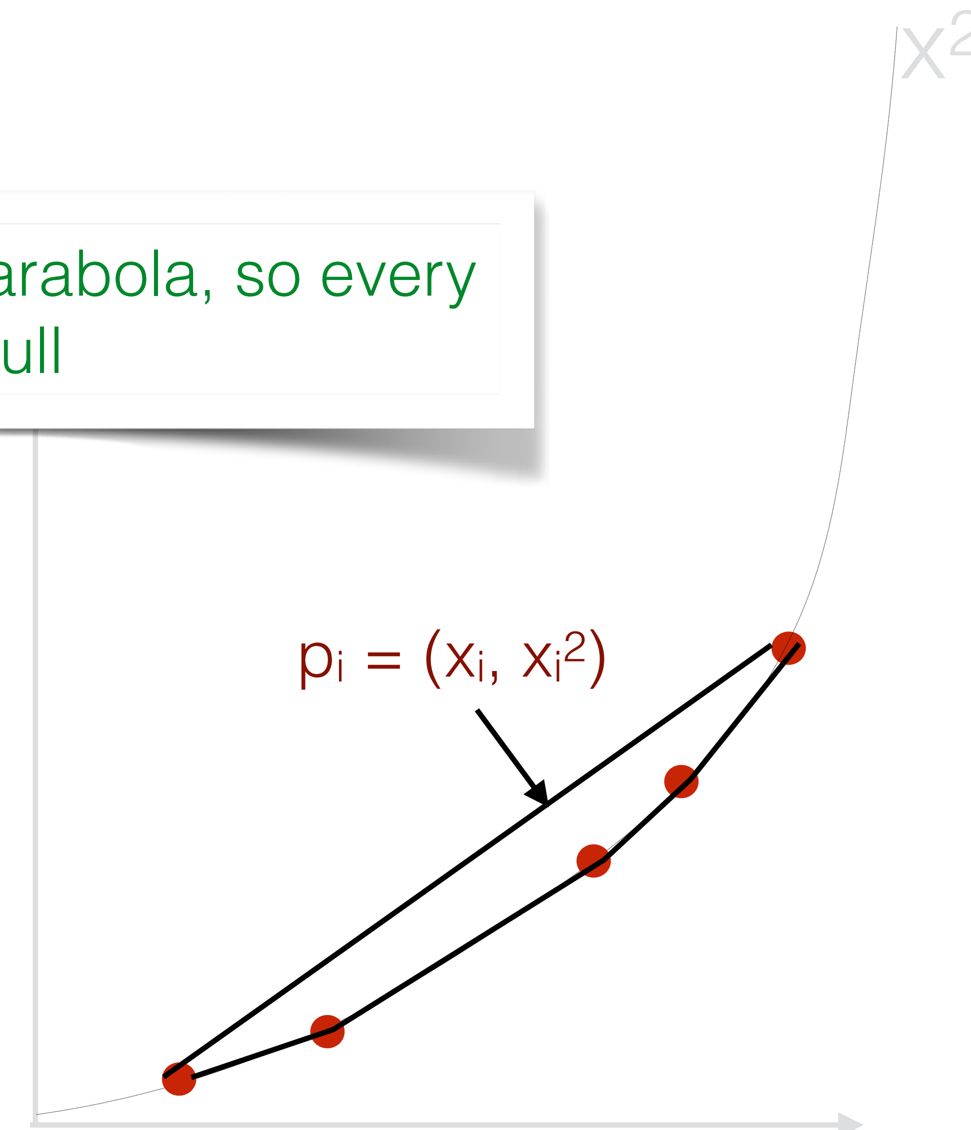
Sorting via ConvexHull

- Let P: set of values $x_1, x_2, \ldots x_n$. to sort

- Let P': set points { $p_i = (x_i, x_i^2)$}
- Run CH(P') to find their convex hull

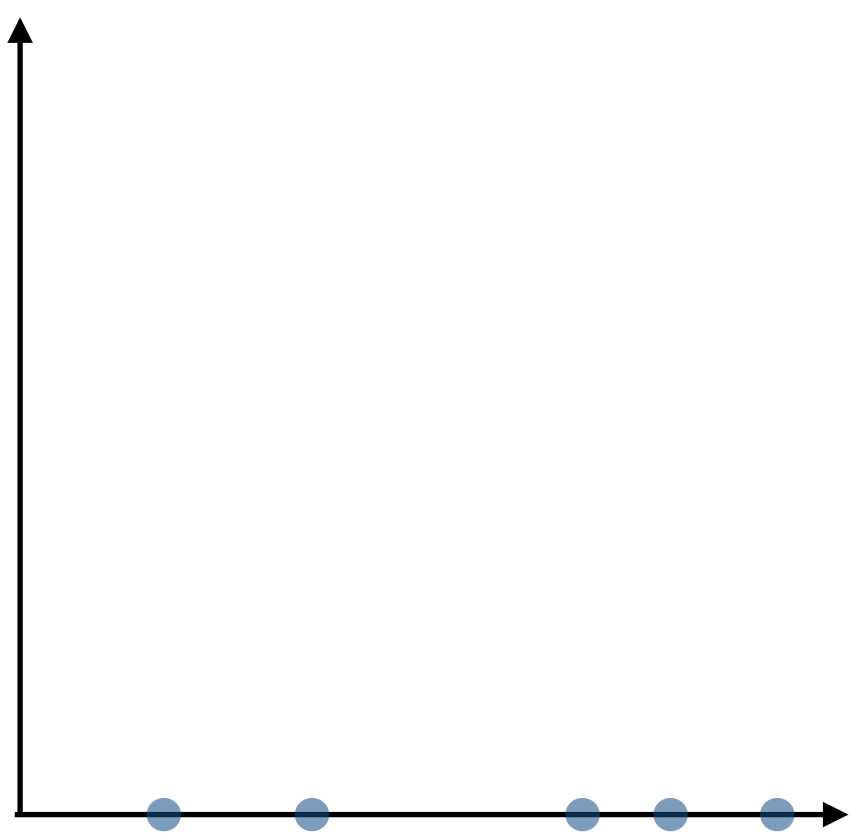
- They fall on a parabola, so every point is on the hull



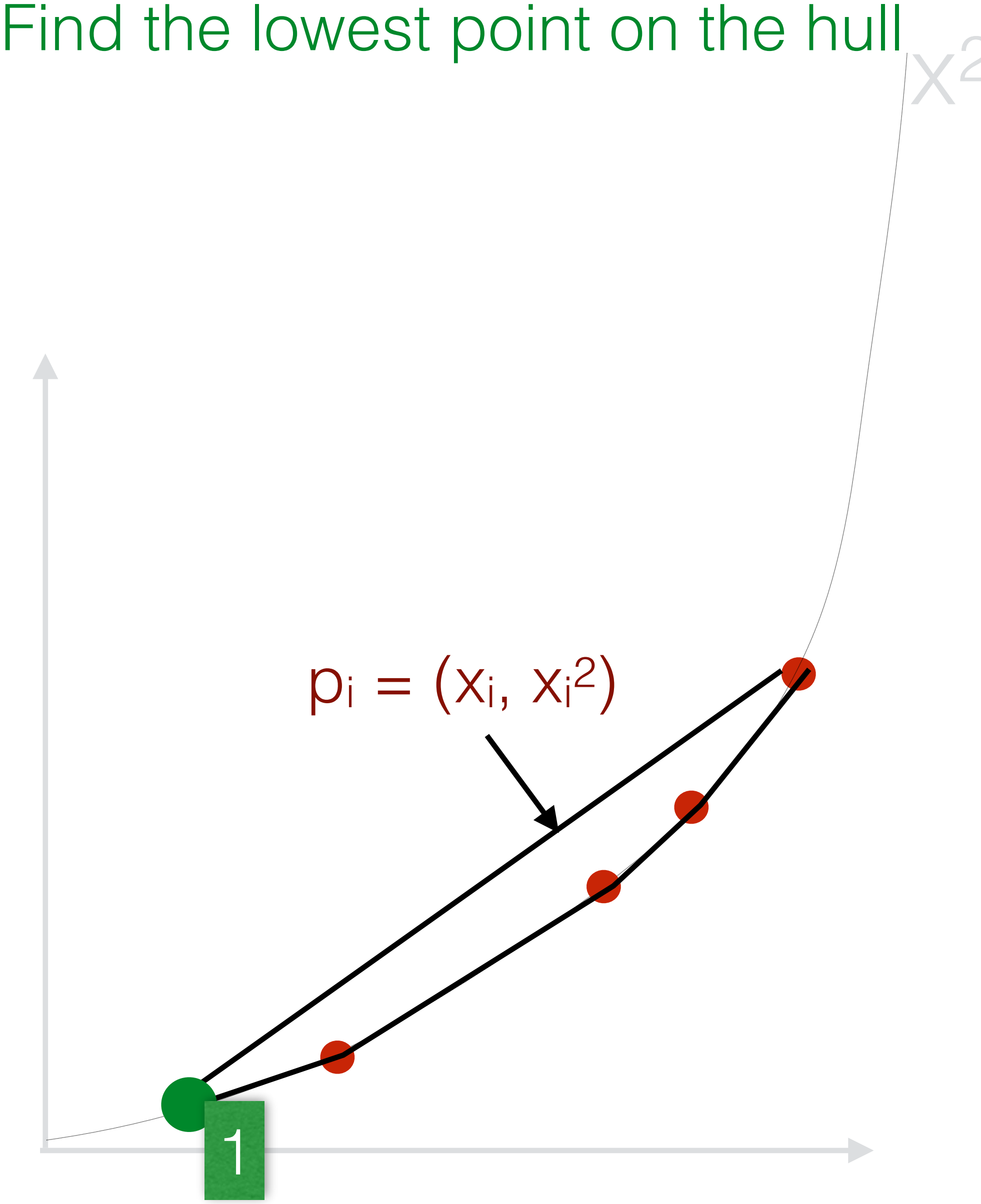


Sorting via ConvexHull

- Let P: set of values $x_1, x_2, \ldots x_n$ to sort

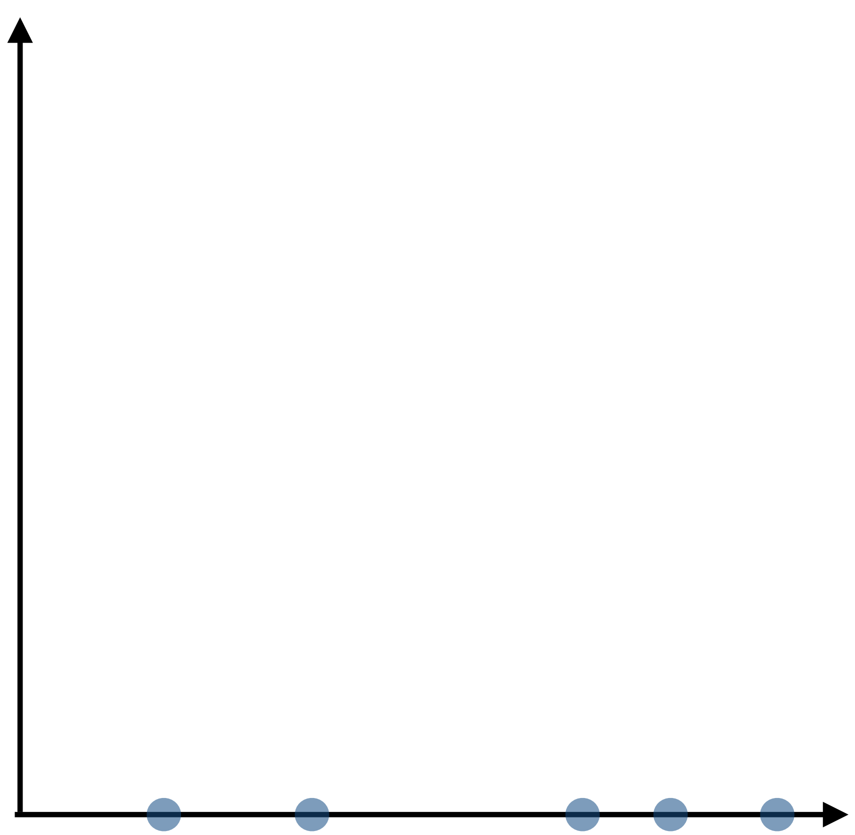


- Let P': set points $\{ p_i = (x_i, x_i^2)\}$
- Run CH(P') to find their convex hull
- Find the lowest point on the hull

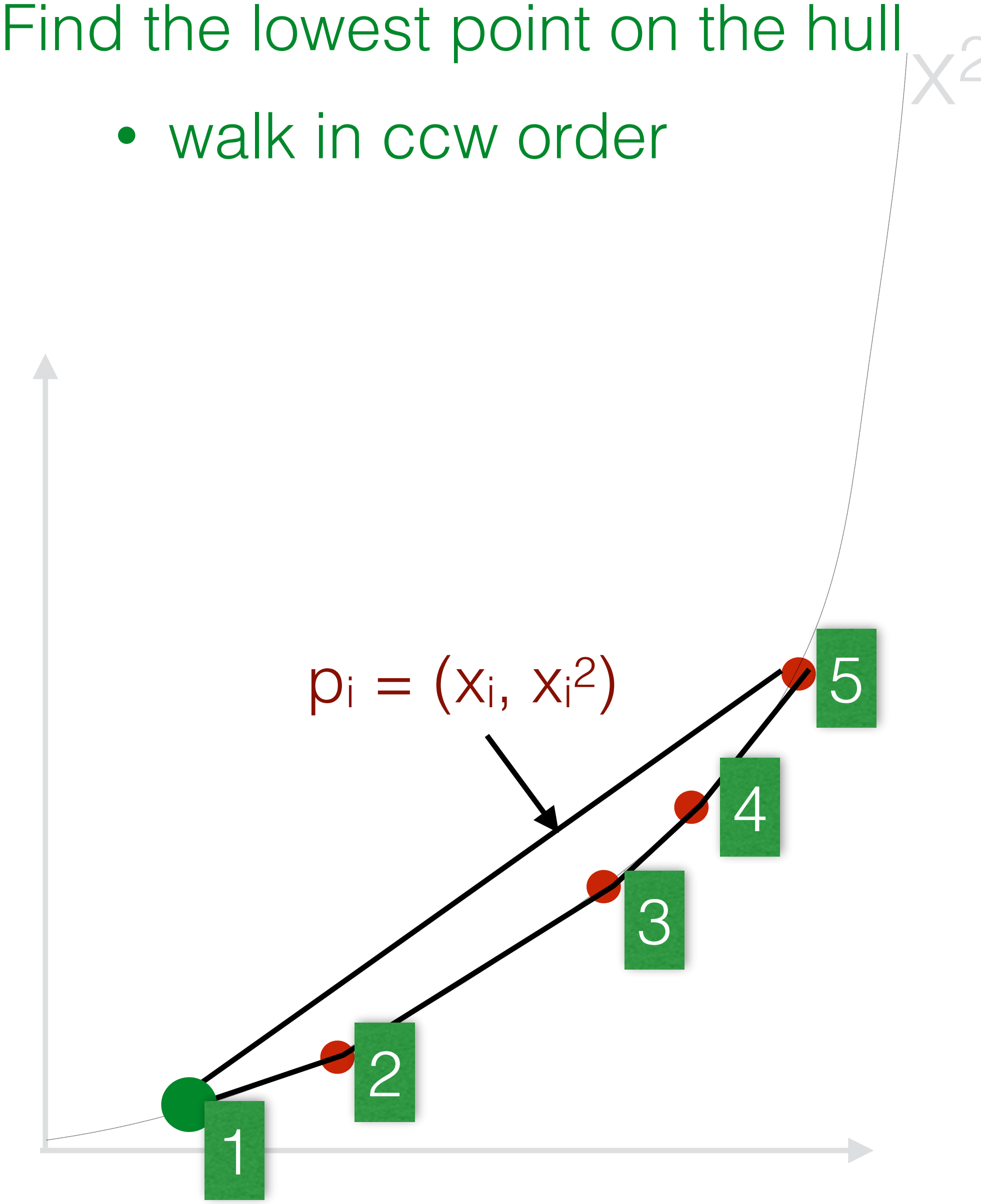


Sorting via ConvexHull

- Let P: set of values $x_1, x_2, \ldots x_n$. to sort

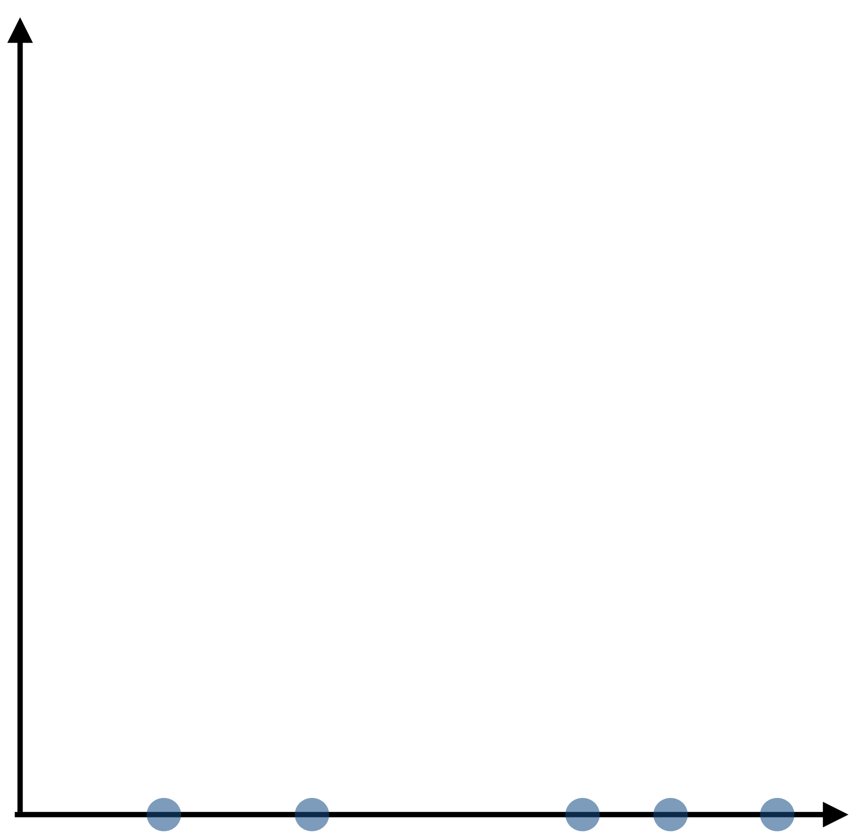


- Let P': set points { $p_i = (x_i, x_i^2)$}
- Run CH(P') to find their convex hull
- Find the lowest point on the hull
  - walk in ccw order



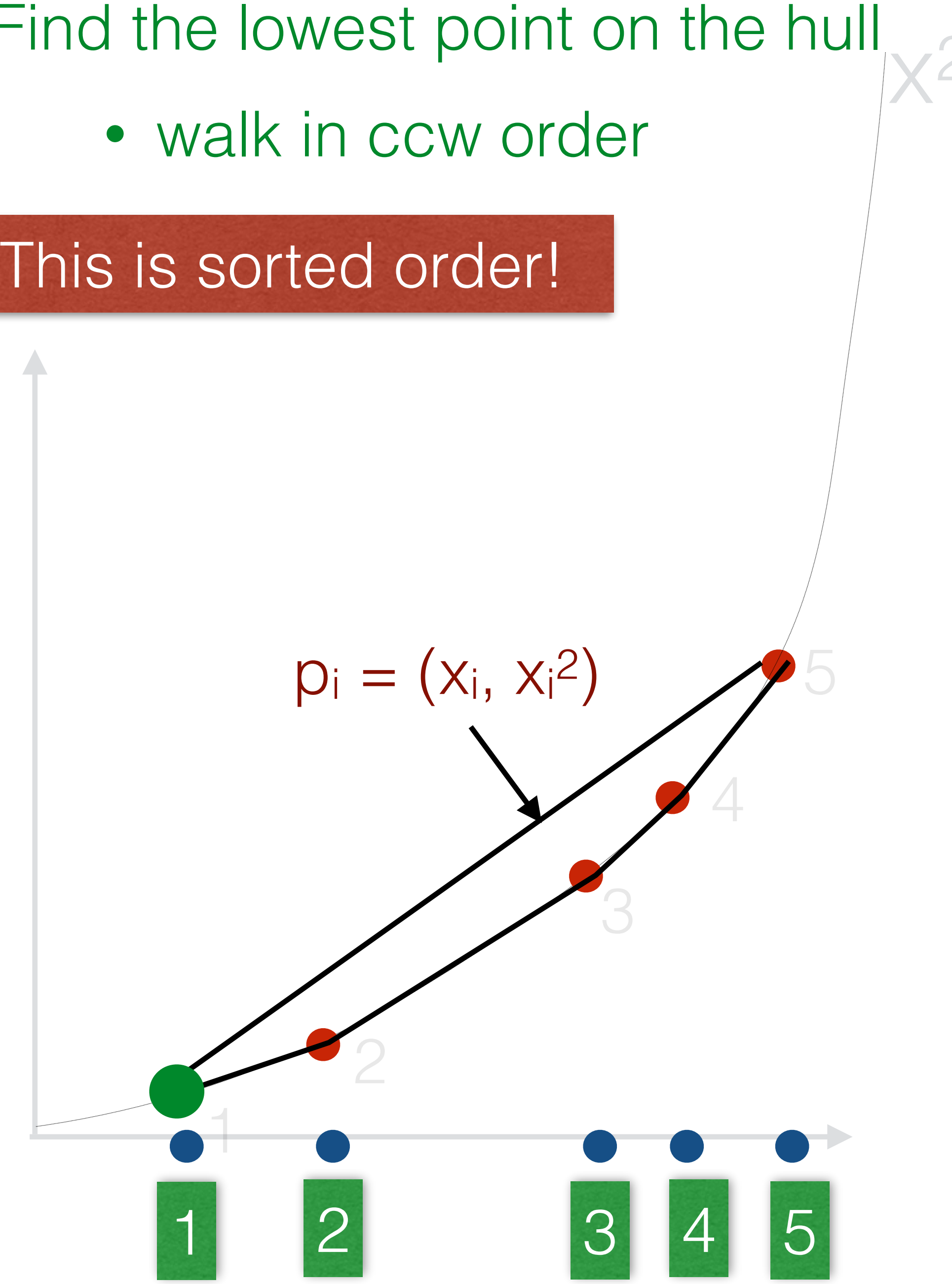
Sorting via ConvexHull

- Let P: set of values $x_1, x_2, \ldots x_n$ to sort



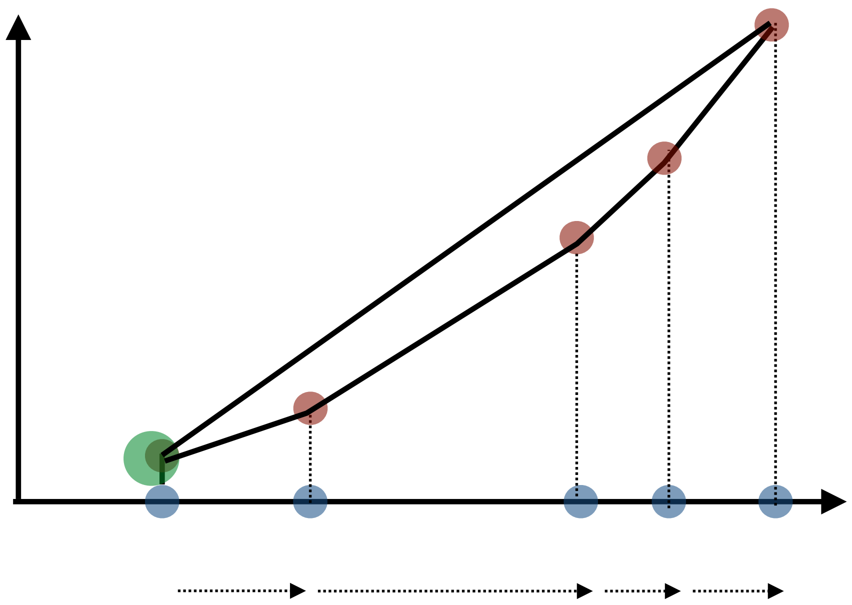
- Let P': set points { $p_i = (x_i, x_i^2)$ }
- Run CH(P') to find their convex hull
- Find the lowest point on the hull
  - walk in ccw order

This is sorted order!



Sorting via ConvexHull

- Input: set of points $x_1, x_2, \ldots x_n$
  - 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!



Analysis: runs in O(CH(n)) + O(n)

- This shows that CH is an upper bound for sorting, or Sorting $\leq$ ConvexHull
- If we could find the CH faster than $\Theta(n \lg n)$, we could use it to sort faster than $\Theta(n \lg n)$, which is impossible!

Summary

sorting $\xrightarrow{O(n)}$ convex hull

reduces to
(or: solves via)

sorting is $\Omega(n \lg n)$

We show that we can use ConvexHull to Sort: Let P be a set of values that need to be sorted. We'll show that there exists some instance of the CH problem that sorts P, and we can build this instance in O(n) time

Sort (n) = O(n) + O(Convex Hull(n))

CH must be $\Omega(n \lg n)$

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 quite recently that a convex hull algorithm, even if it does not produce the boundary in order, still needs $\Omega(n \lg n)$ in the worst case

- 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

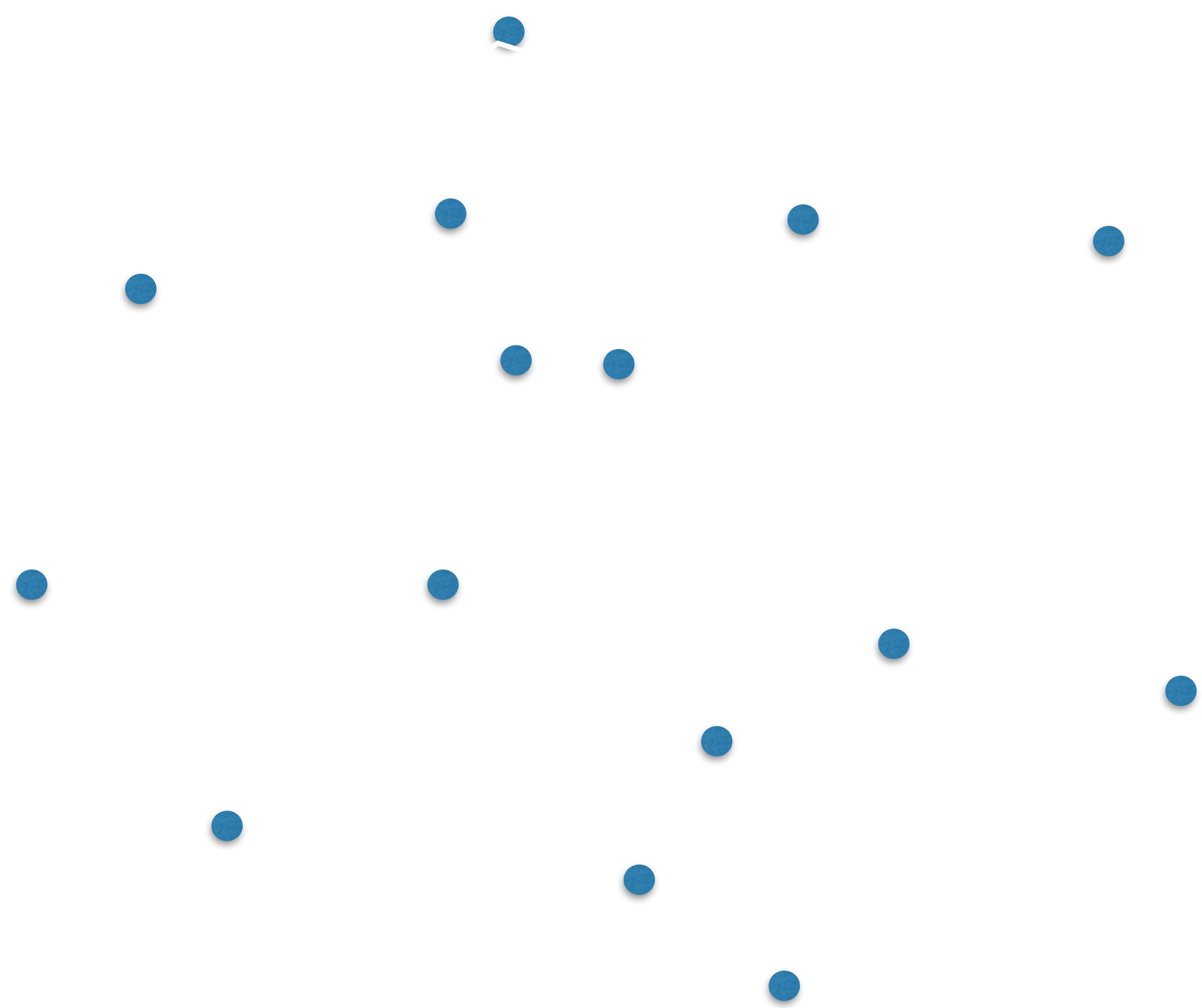
Incremental algorithms

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

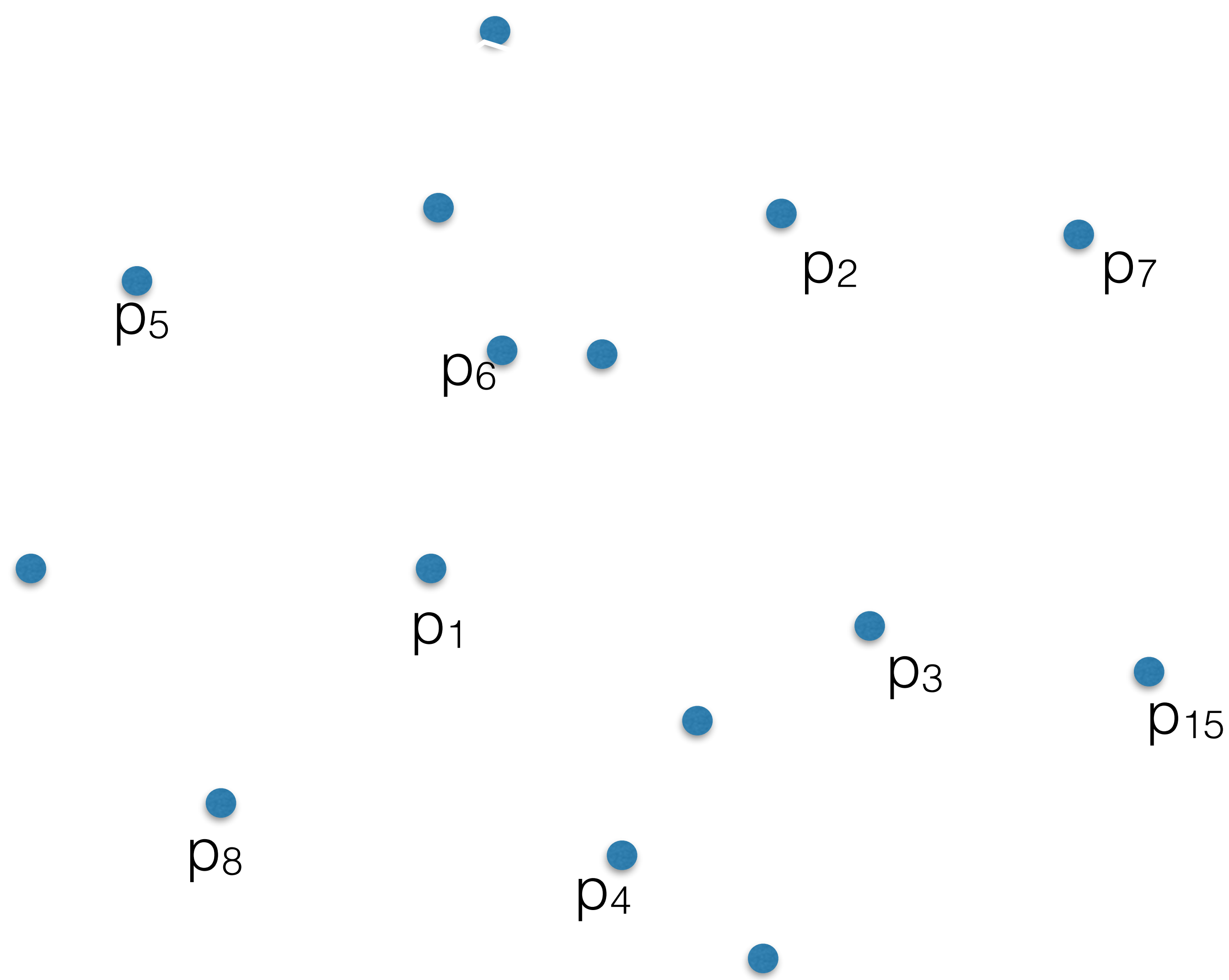
Incremental algo for CH

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



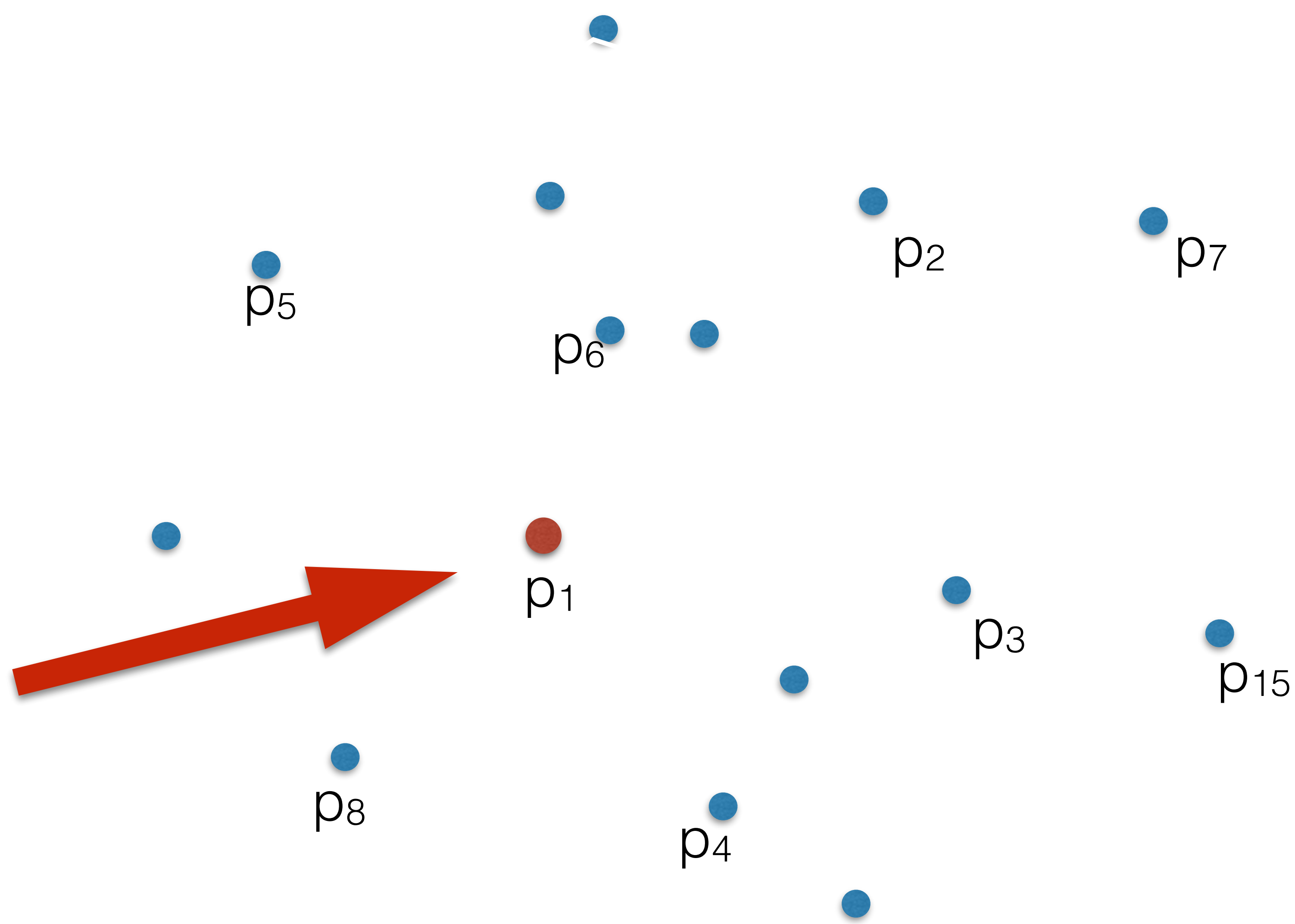
Incremental algo for CH

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



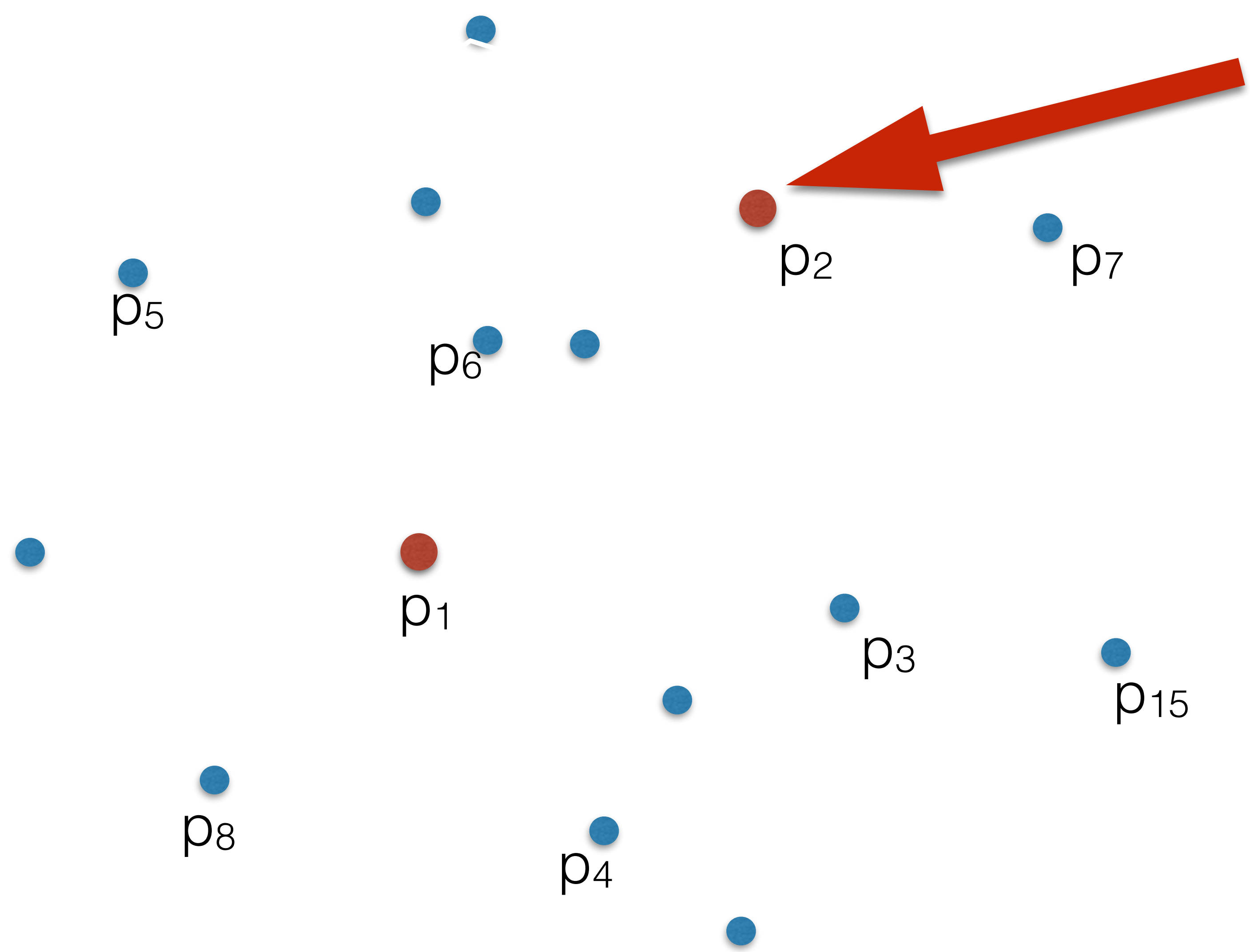
Incremental algo for CH

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



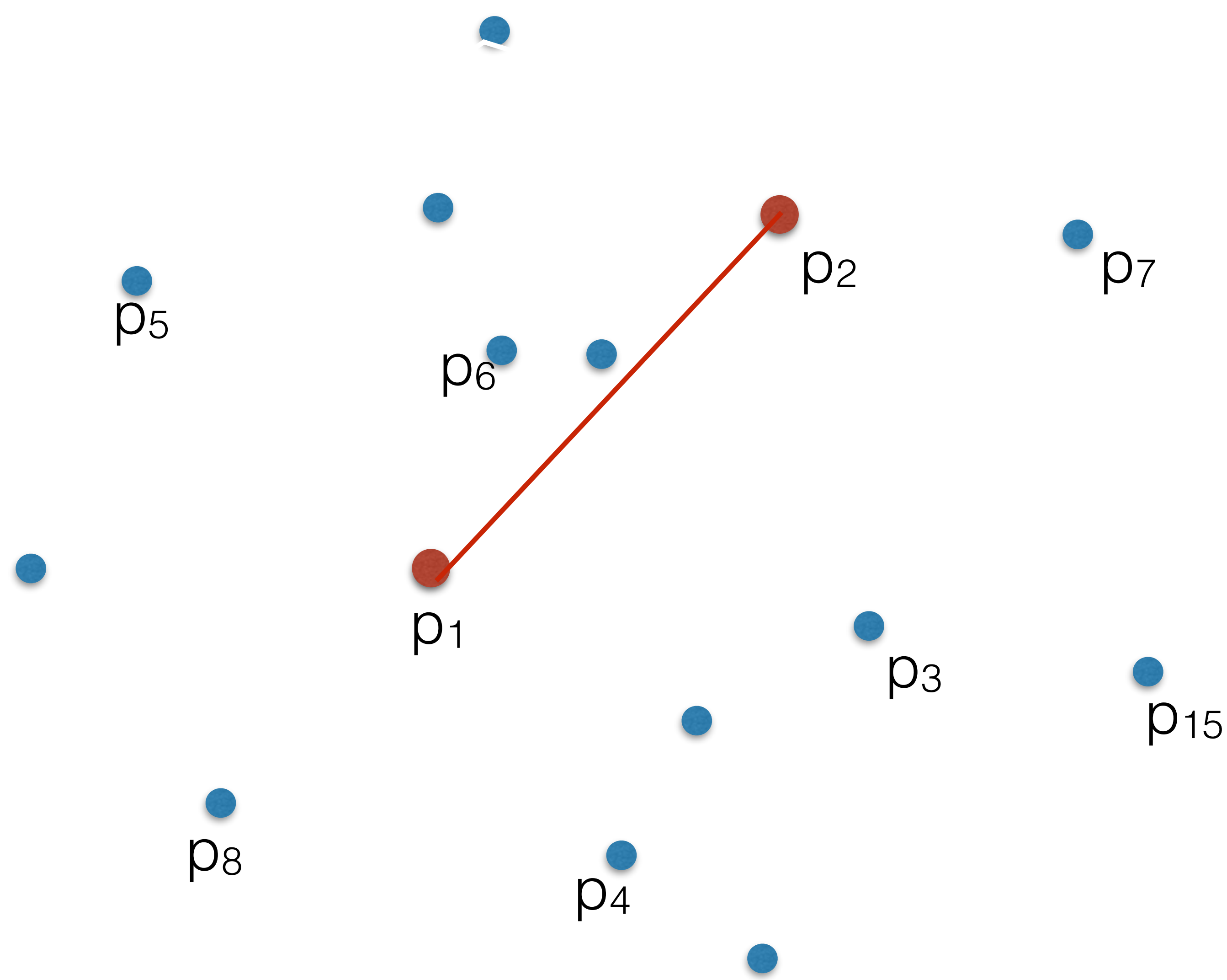
Incremental algo for CH

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



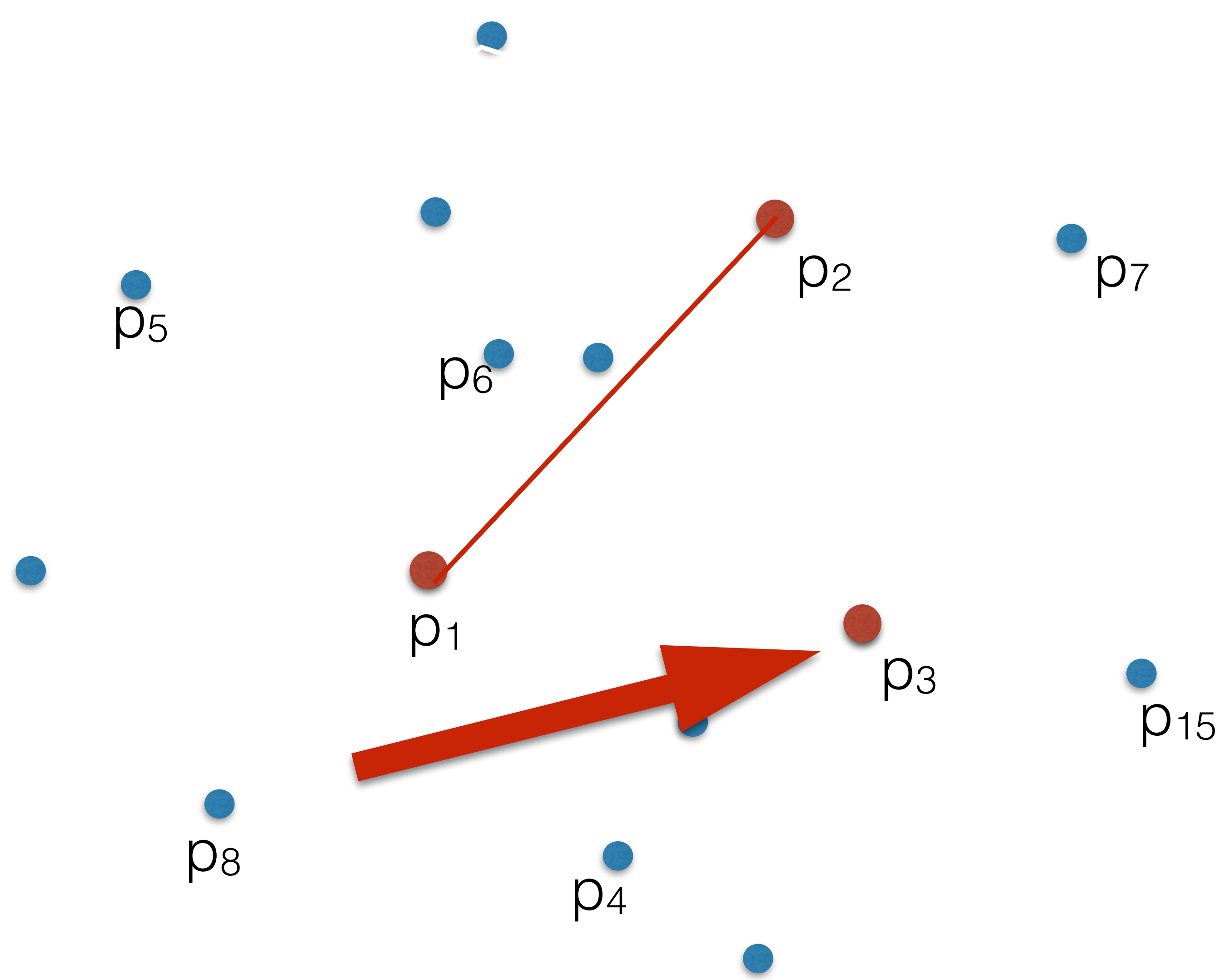
Incremental algo for CH

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



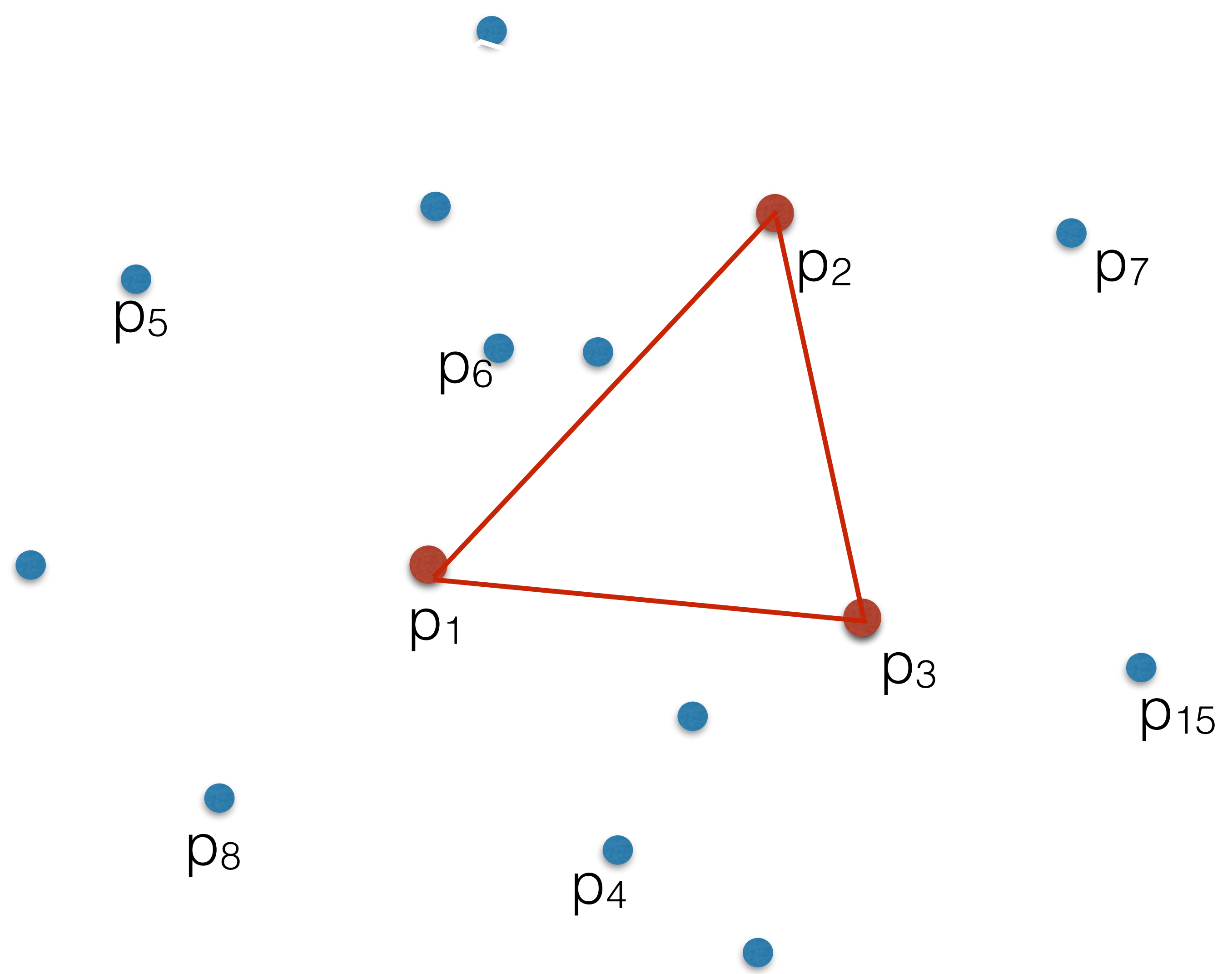
Incremental algo for CH

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



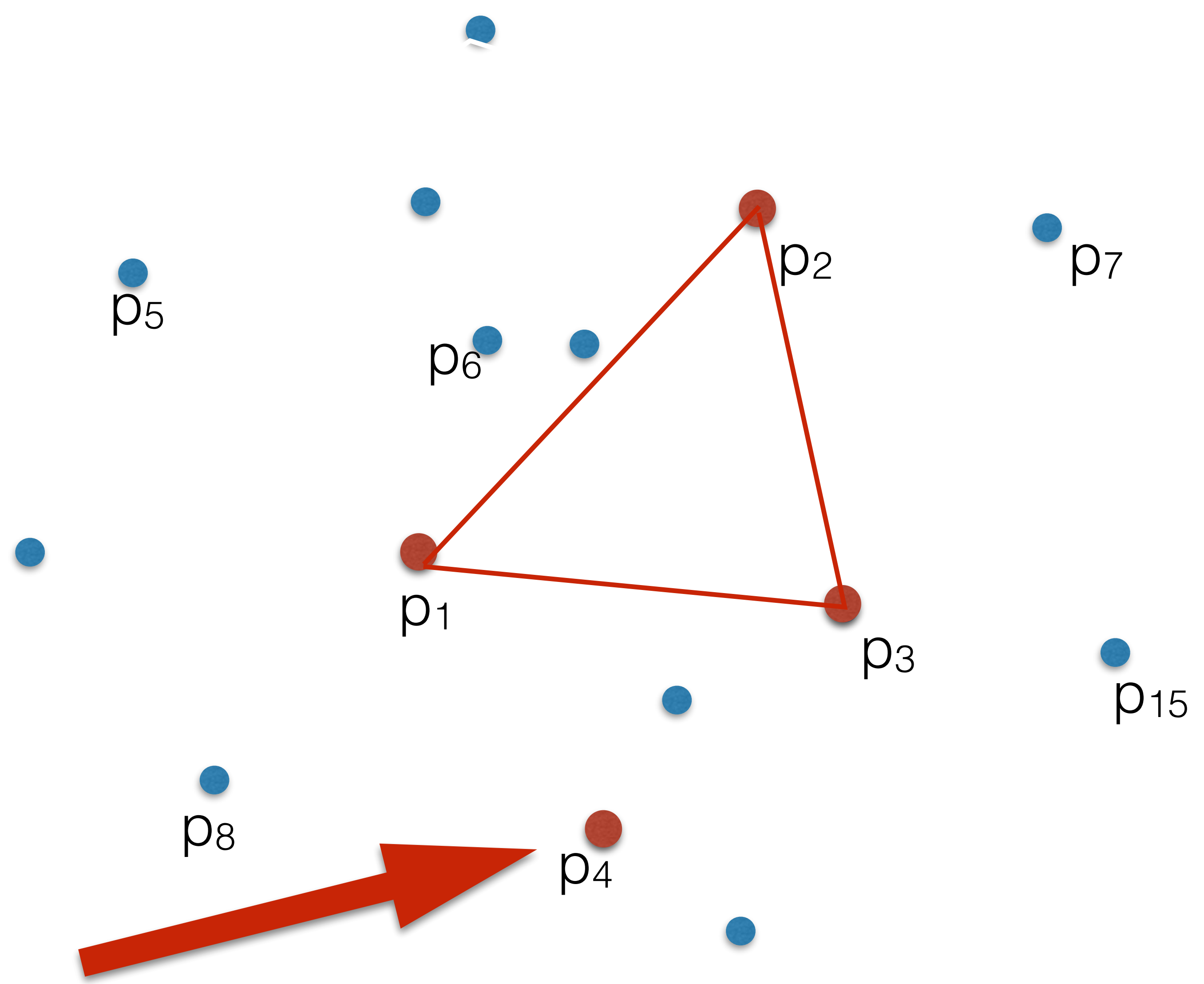
Incremental algo for CH

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



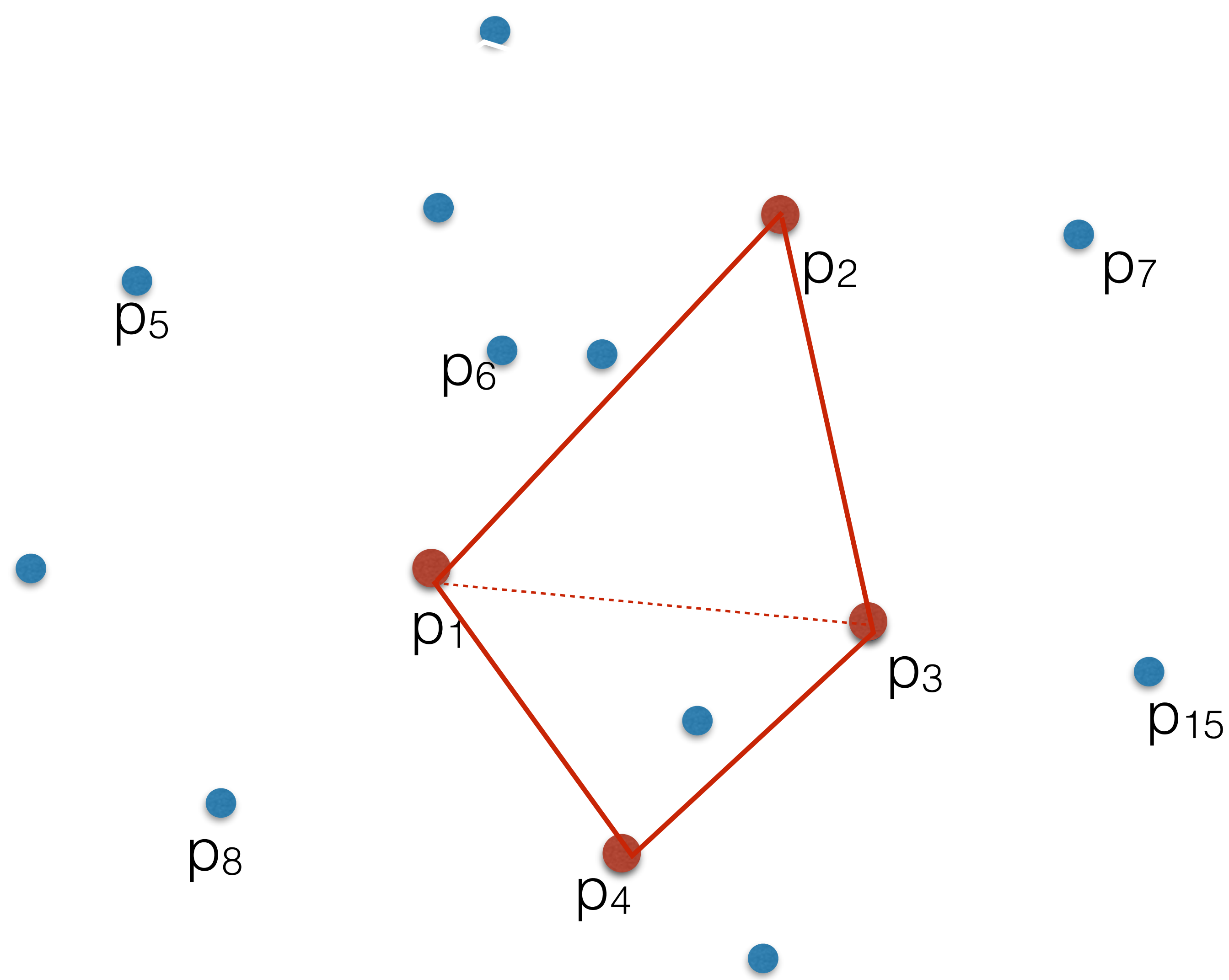
Incremental algo for CH

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



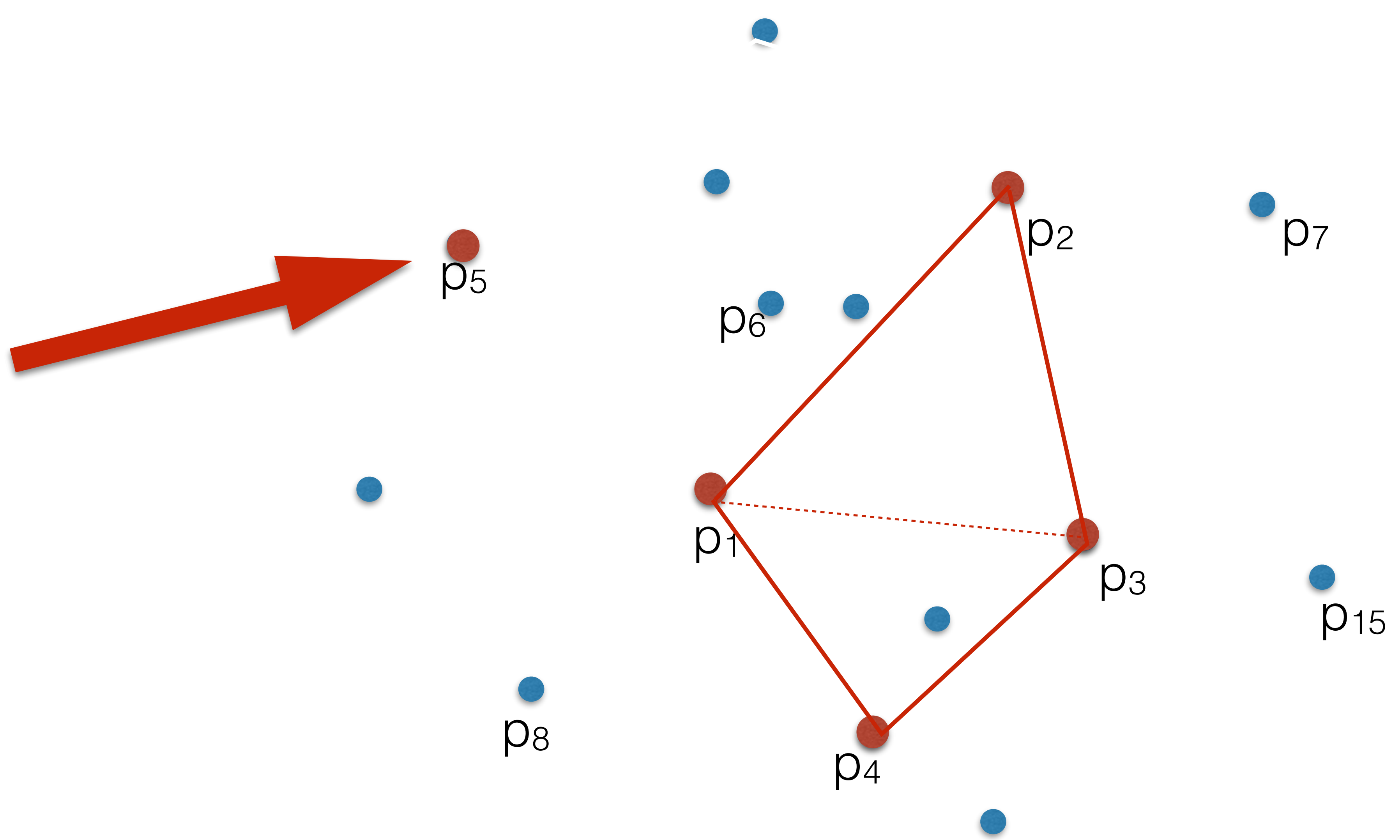
Incremental algo for CH

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



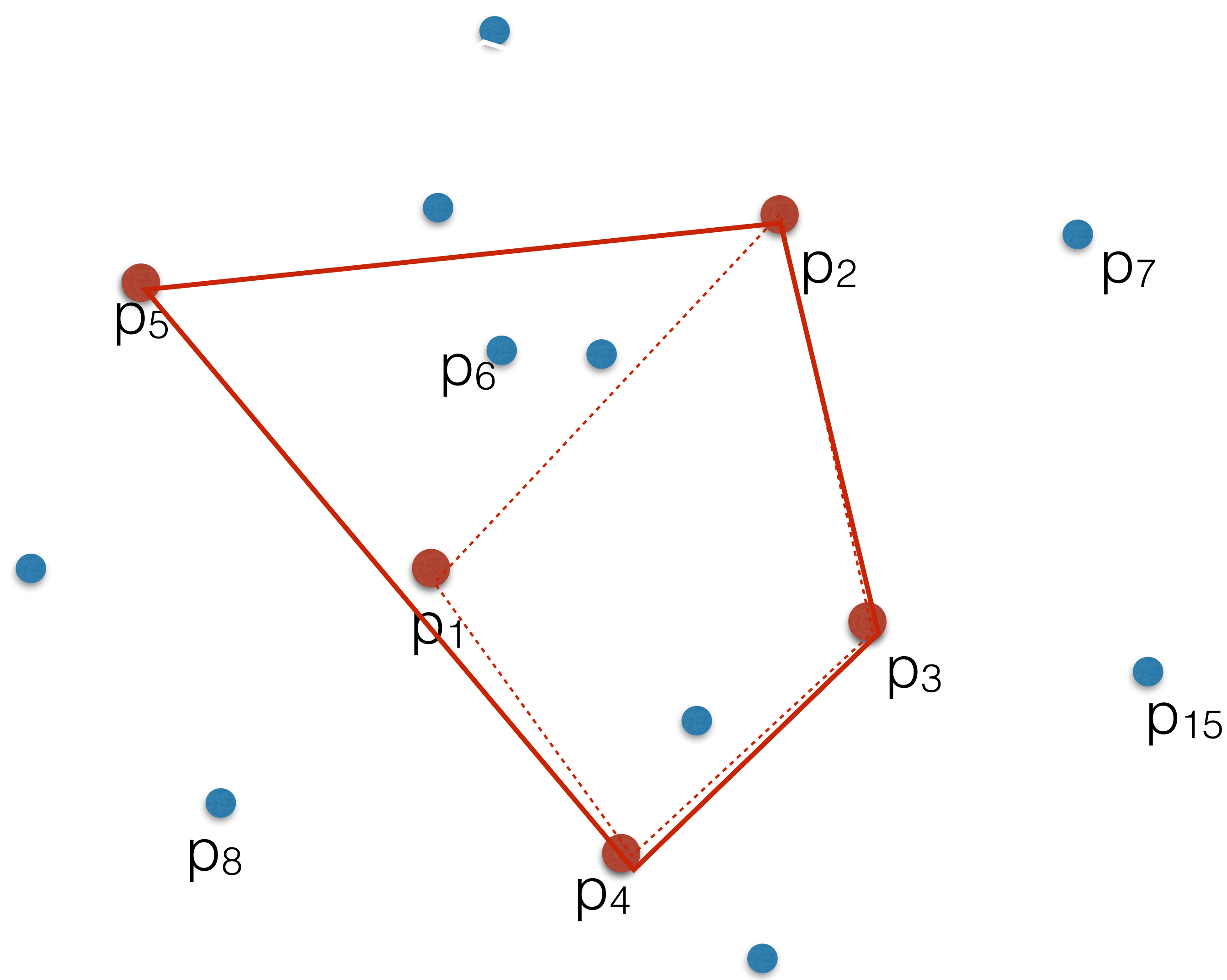
Incremental algo for CH

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



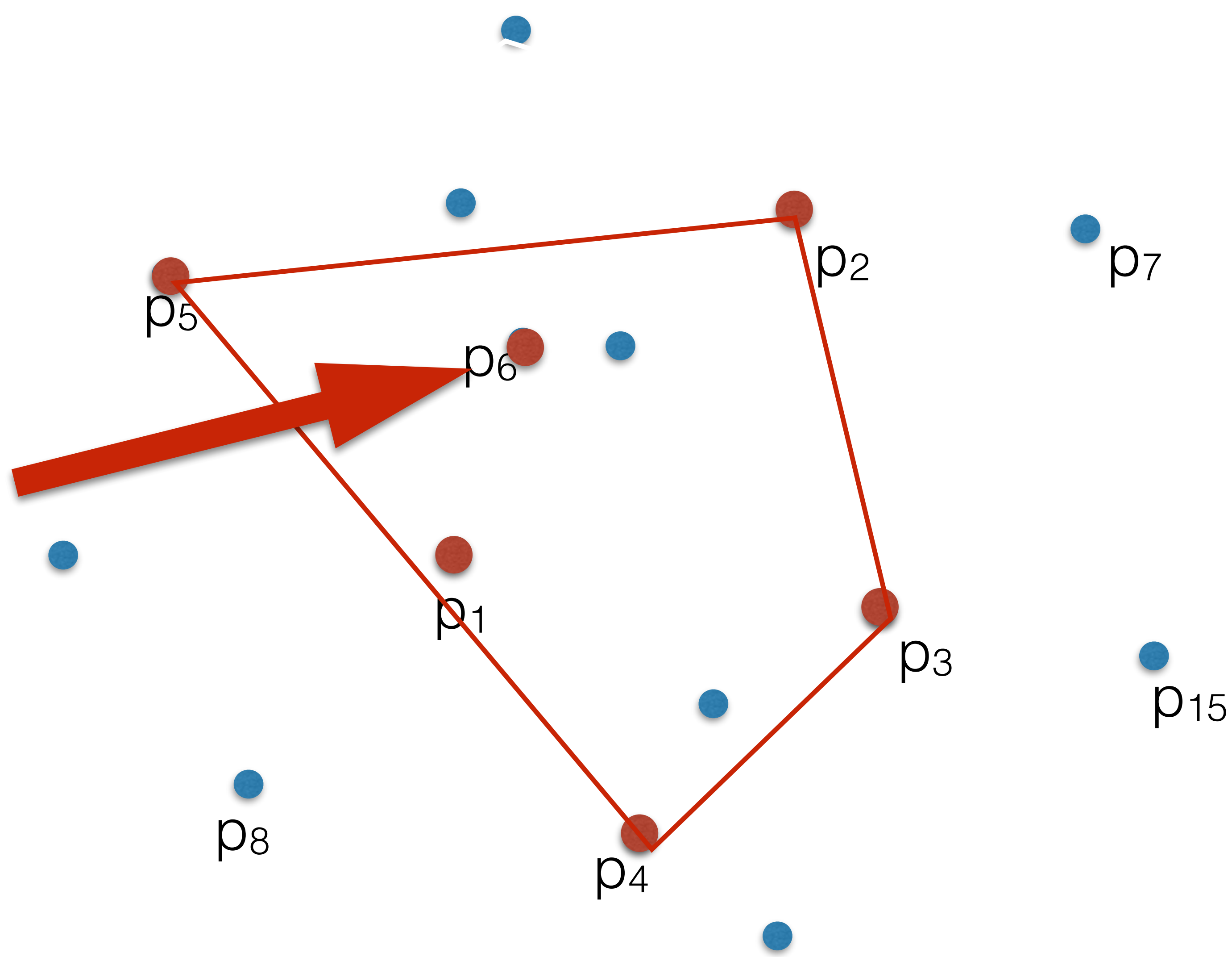
Incremental algo for CH

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



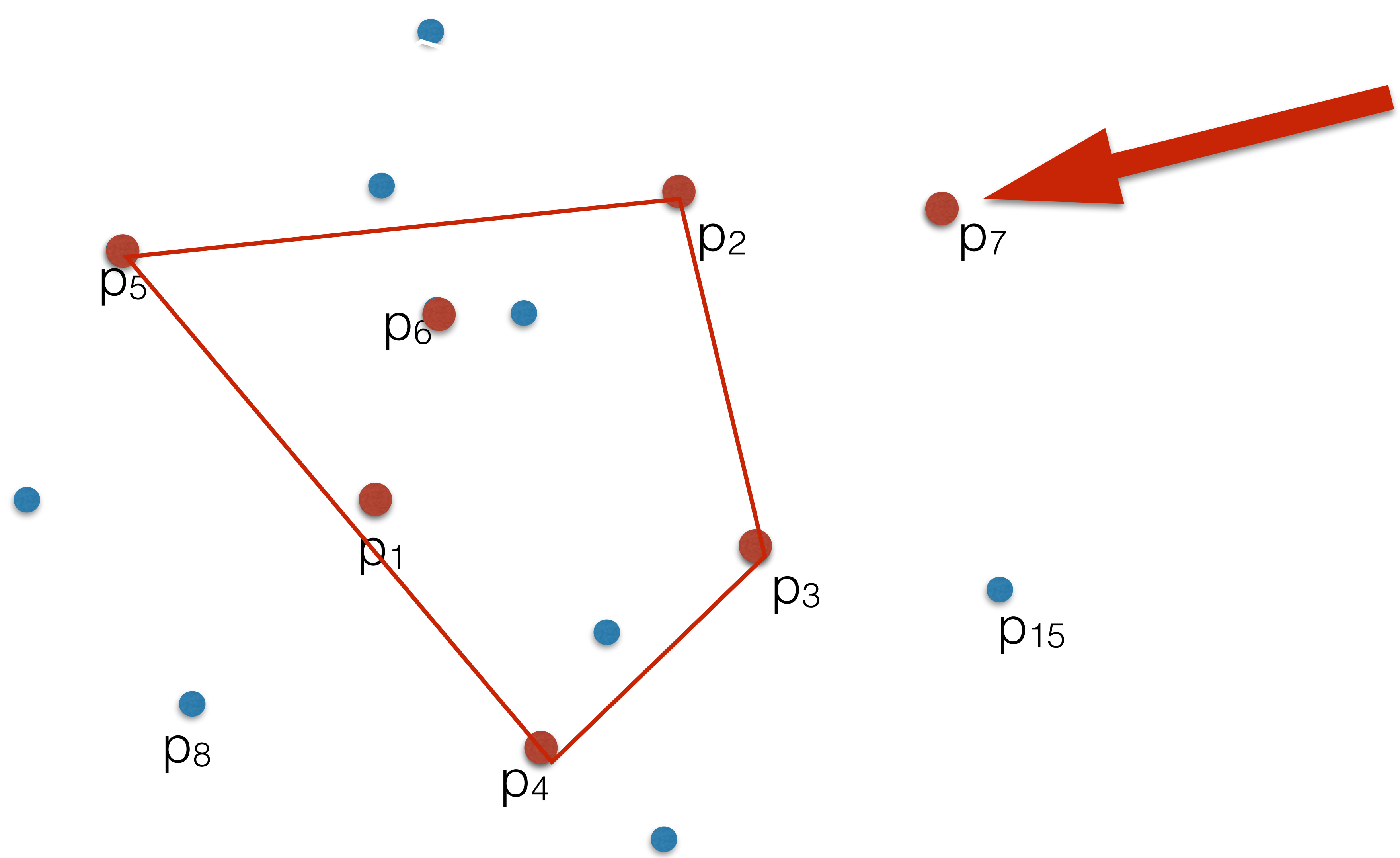
Incremental algo for CH

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



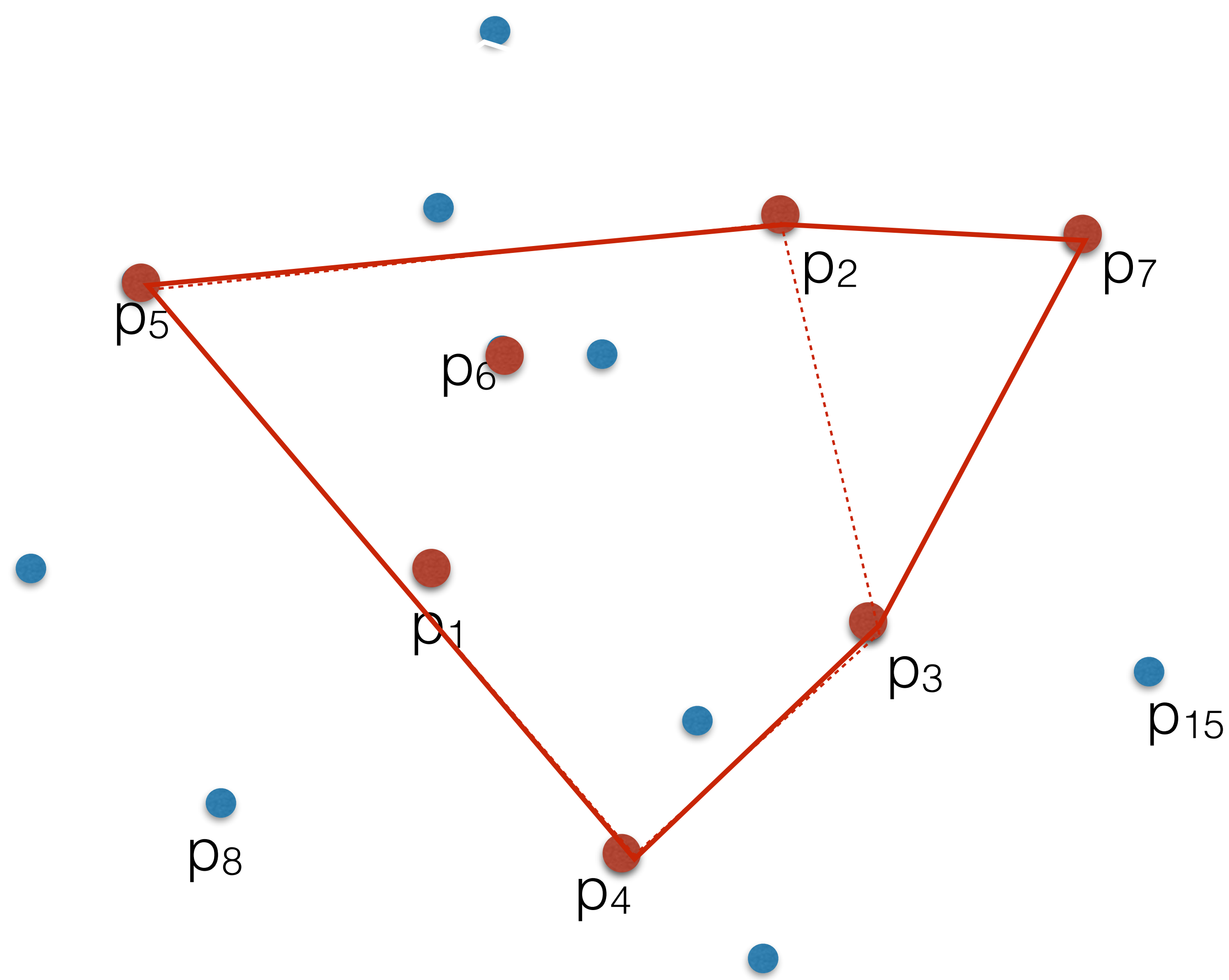
Incremental algo for CH

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



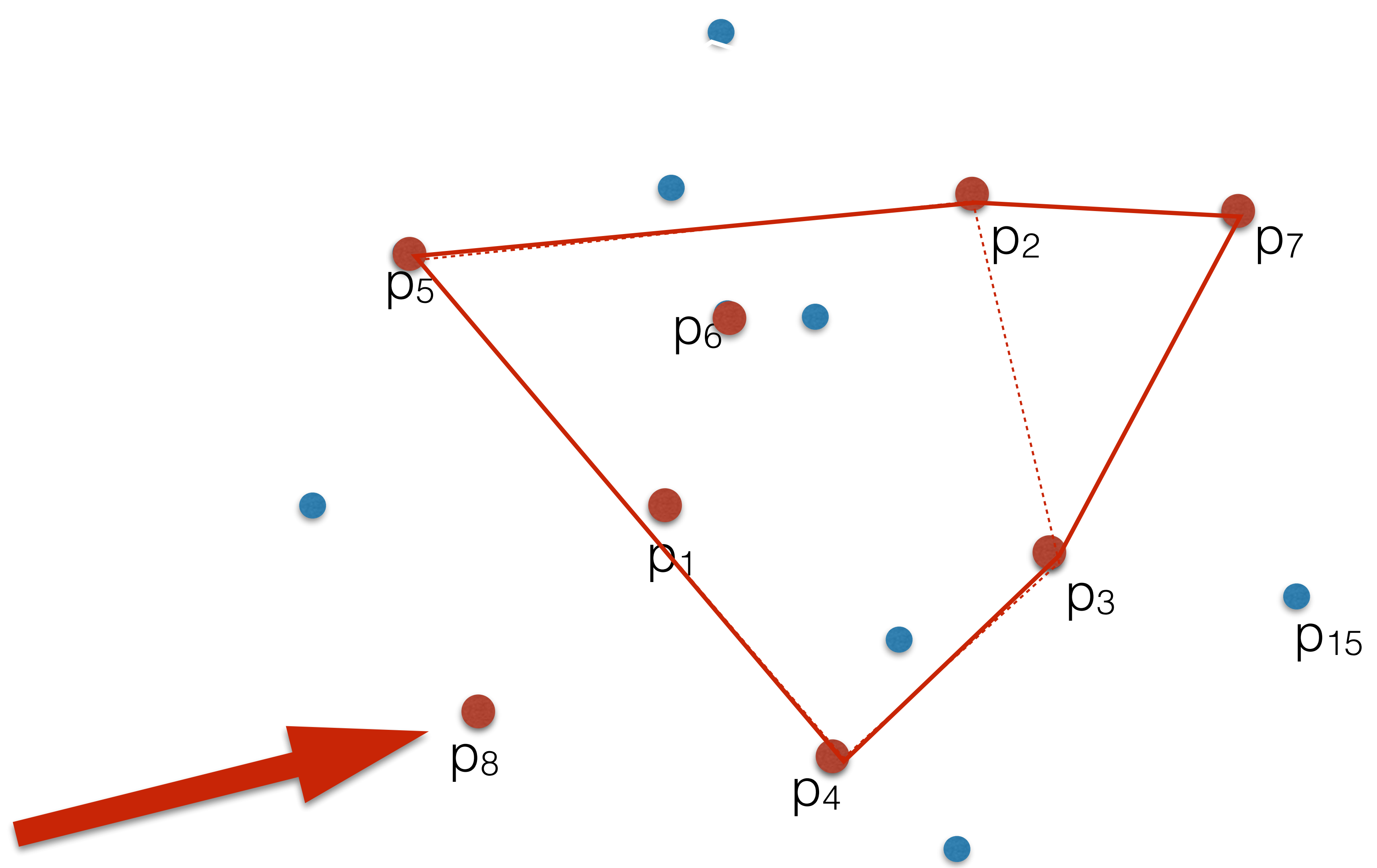
Incremental algo for CH

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



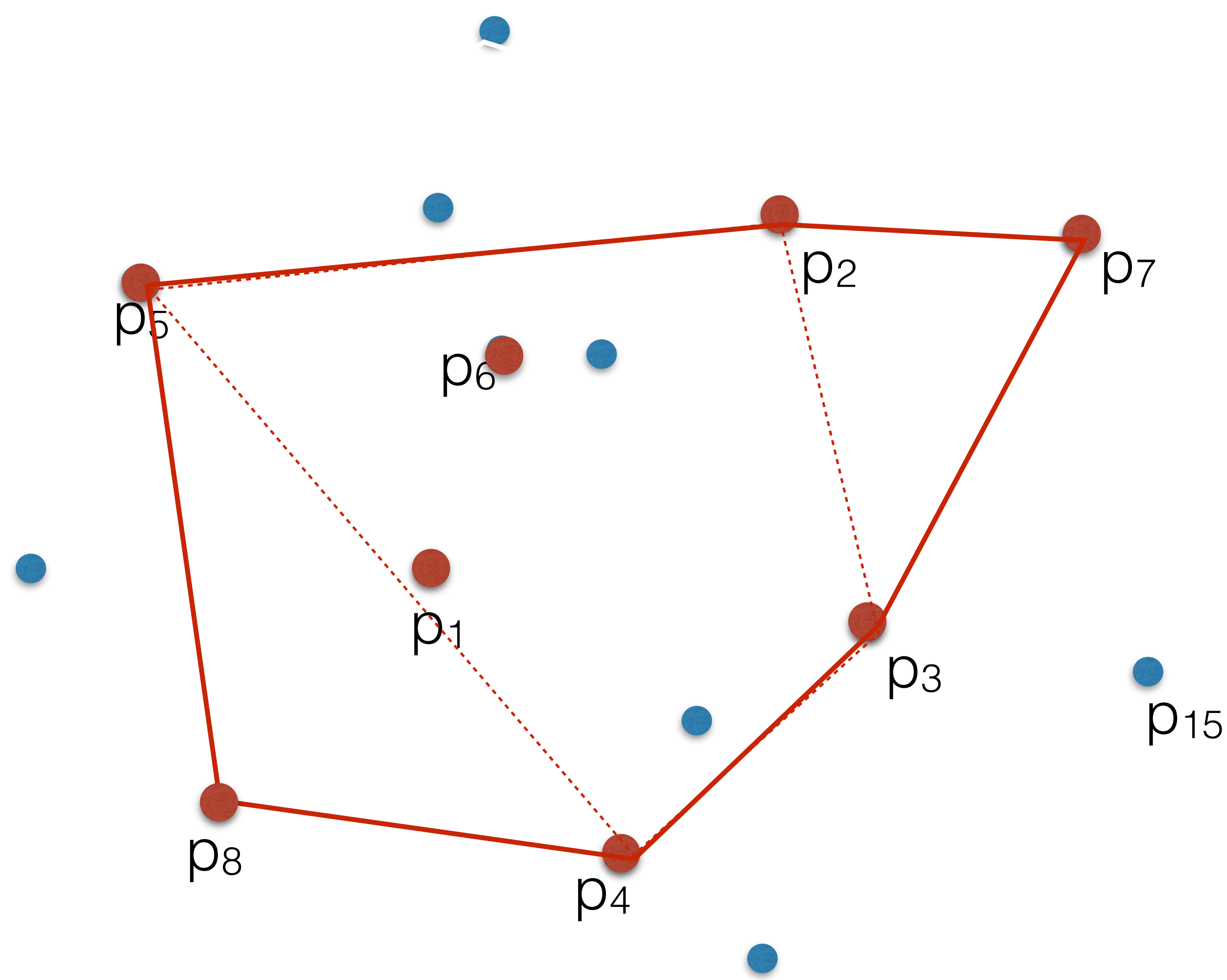
Incremental algo for CH

- 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

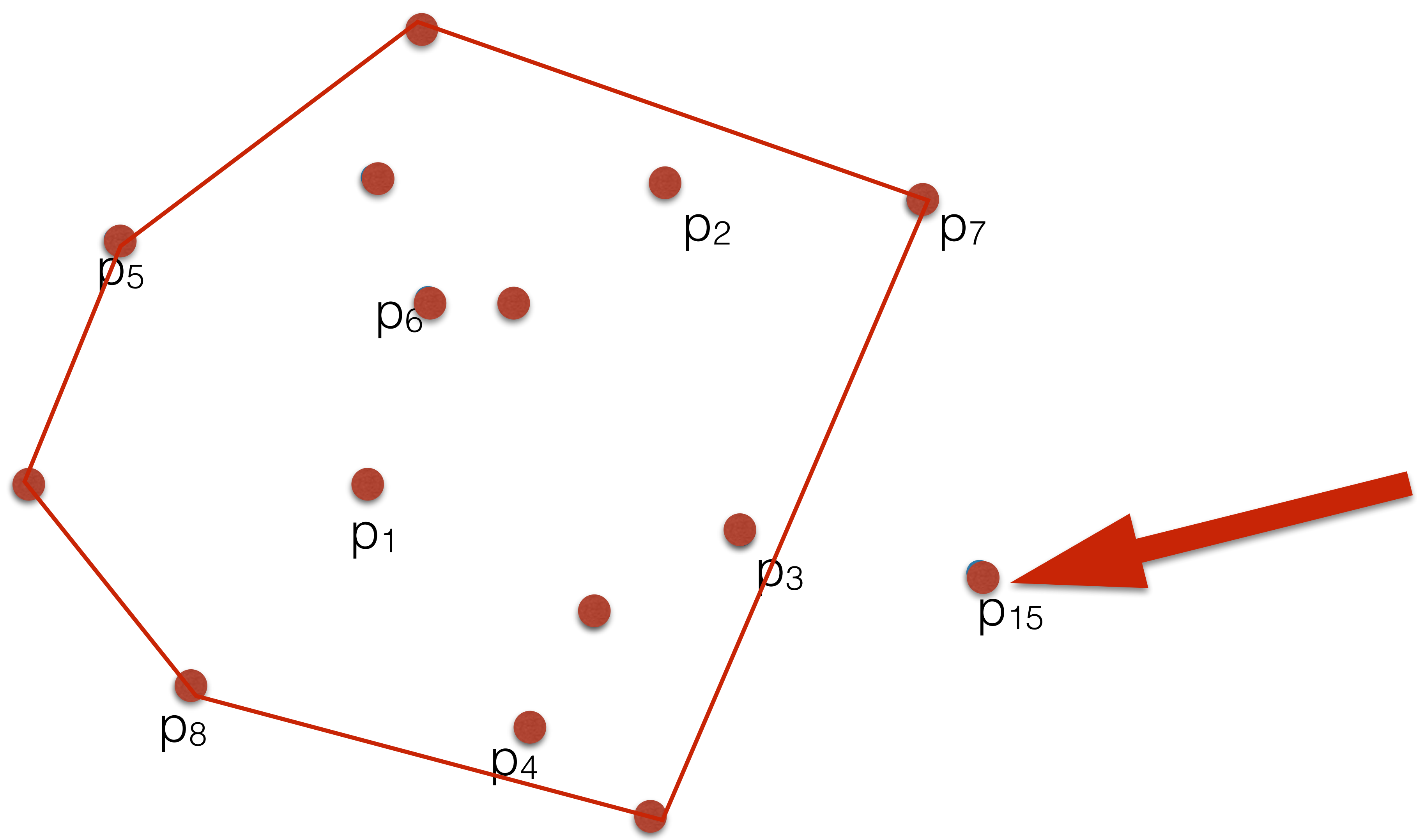
- CH = {}
- for i=1 to n
    - //CH represents the CH of $p_1..p_{i-1}$
    - update CH to represent the CH of $p_1..p_i$



Incremental algo for CH

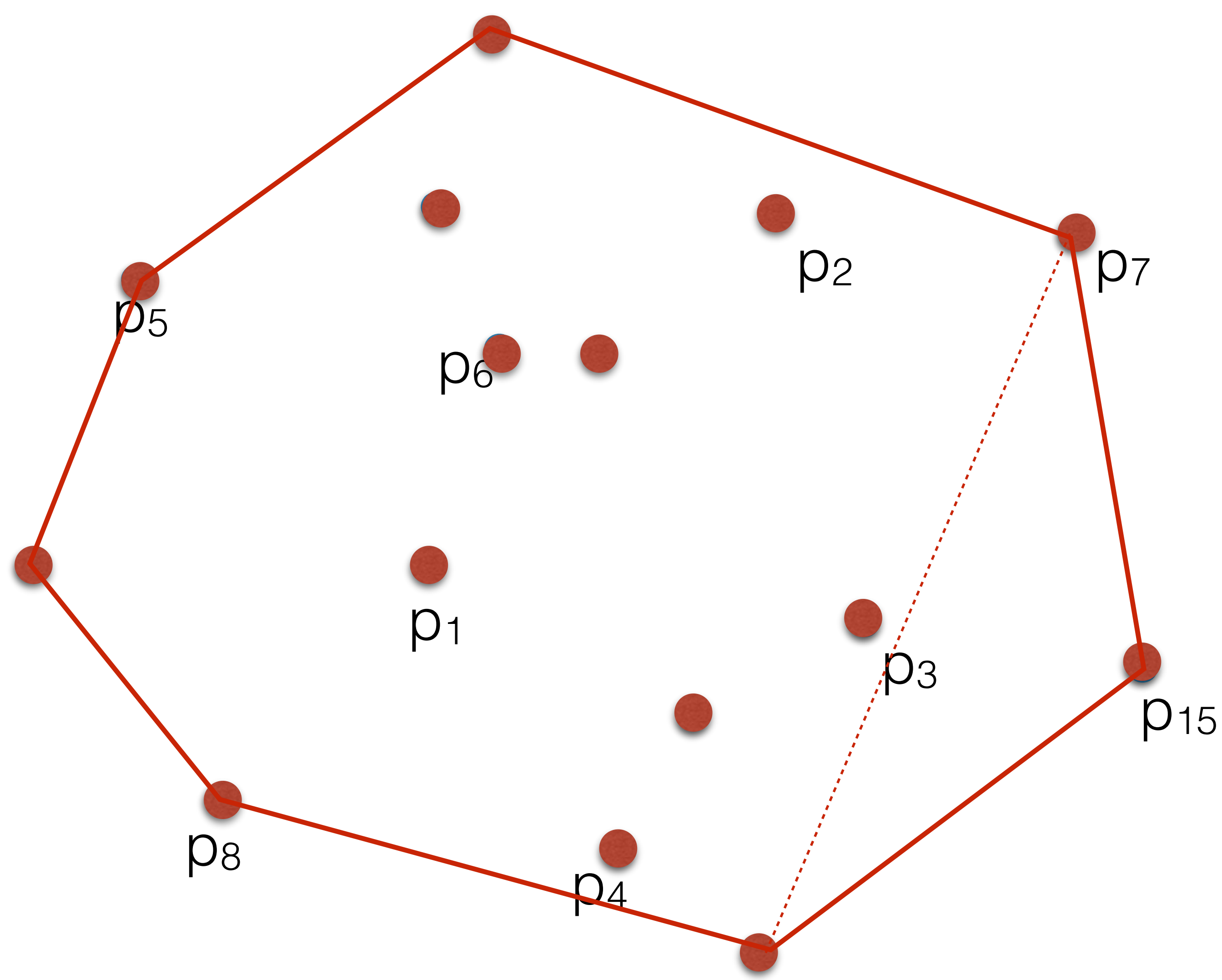
- CH = {}
- for i=1 to n
  - //CH represents the CH of $p_1..p_{i-1}$
  - update CH to represent the CH of $p_1..p_i$

and so on



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
    - CASE 1: p is in polygon
    - CASE 2: p outside polygon

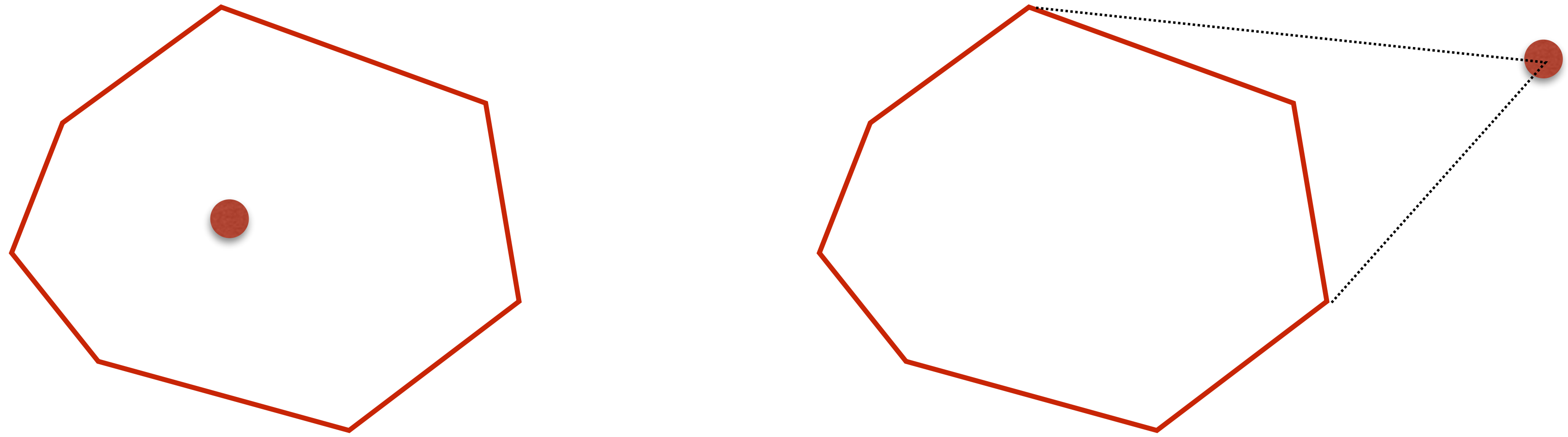
How do you handle each case?

- Class work: Pick a set of points, simulate the incremental approach, and try to answer the question: how do you handle each case?

Incremental algo for CH

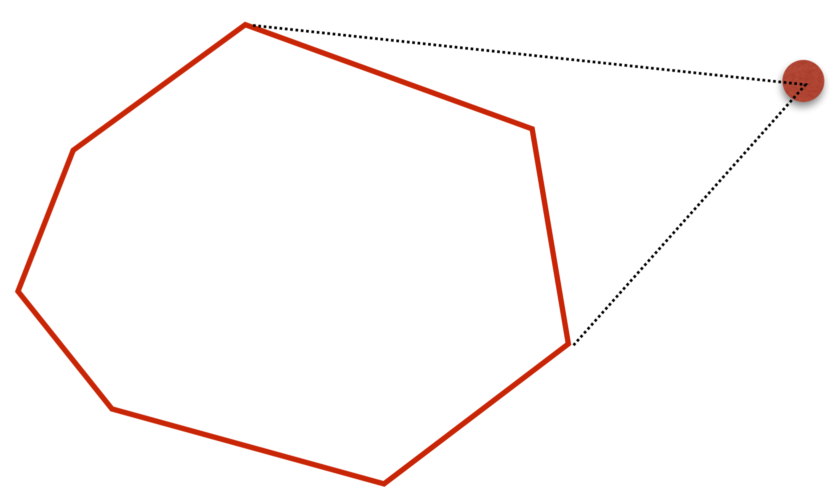
- CH = {}
- for i=1 to n
    - //CH represents the CH of $p_1..p_{i-1}$
    - update CH to represent the CH of $p_1..p_i$

- 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

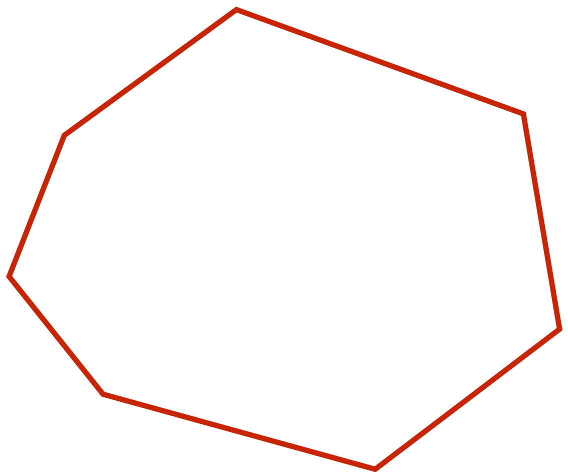
- Issues to solve
  - What's a good representation for a (convex) polygon?
  - We need a point-in-convex-polygon test
  - How to handle CASE 2 ?



Representing a polygon

A polygon is represented as a list of vertices in boundary order.

(the convention is counter-clockwise order)

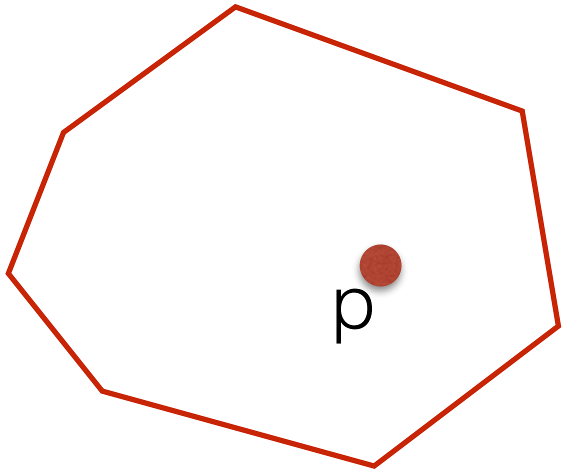


```
typedef struct _polygon{
      int k; //number of vertices
      Point* vertices; //the vertices, ccw in boundary order
  } Polygon;

  or

Vector<Point>           //note: the vertices, ccw in boundary order
```

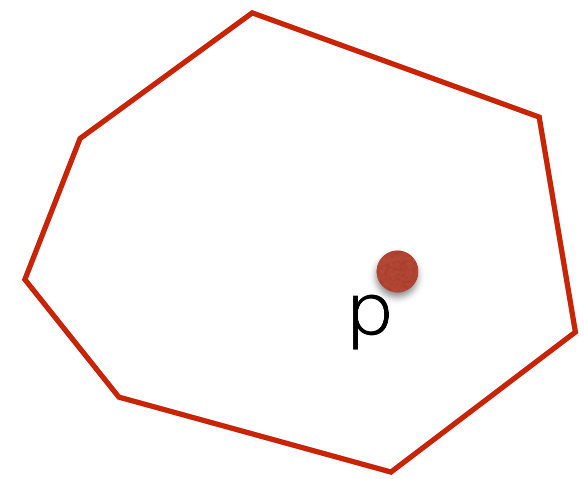
Point in convex polygon




```
//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?

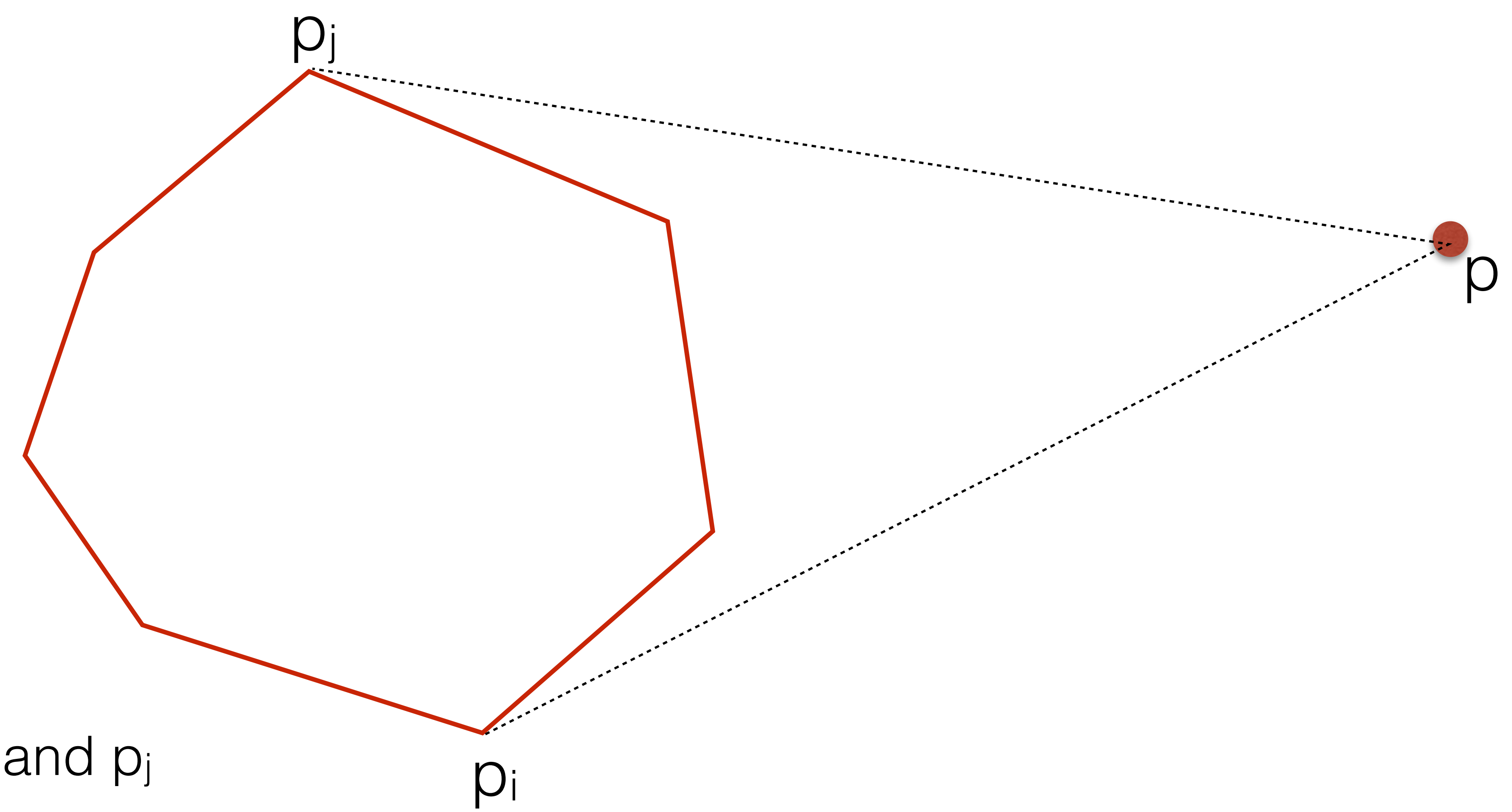
Point in convex polygon




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

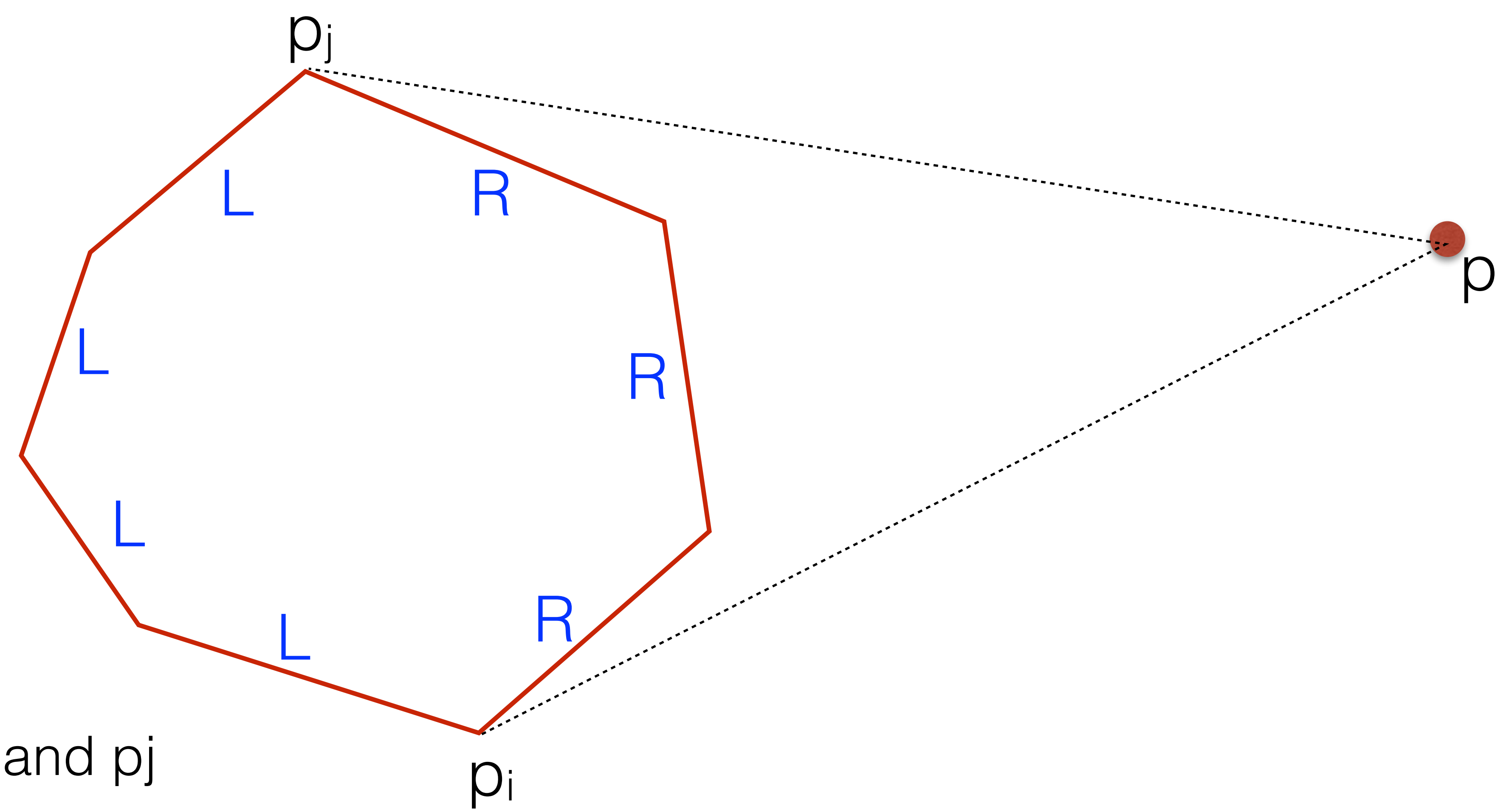


We want to find p_i and p_j

IDEAS?

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

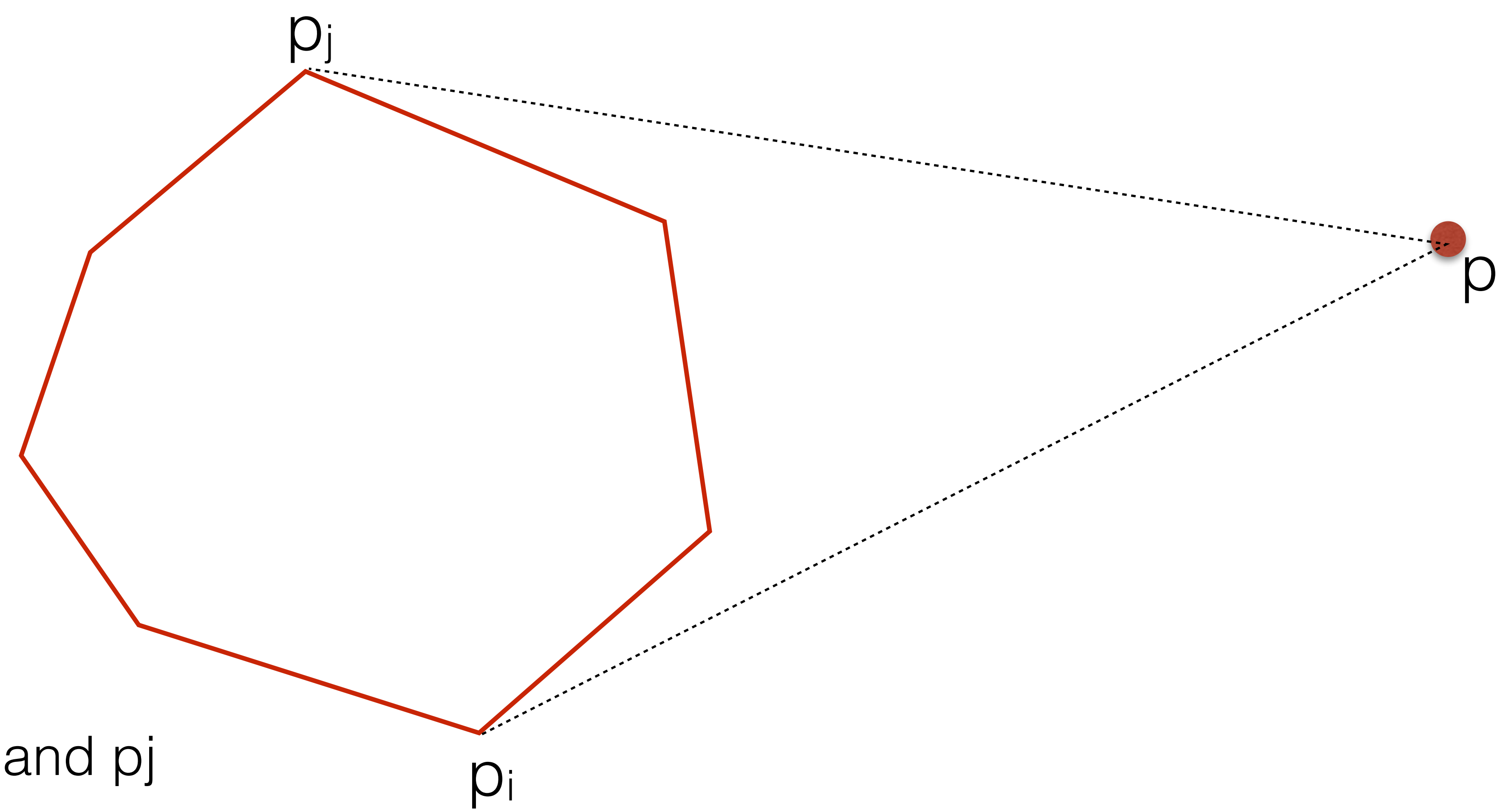
Case 2:



We want to find pi and pj

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

Case 2:



We want to find pi and pj

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

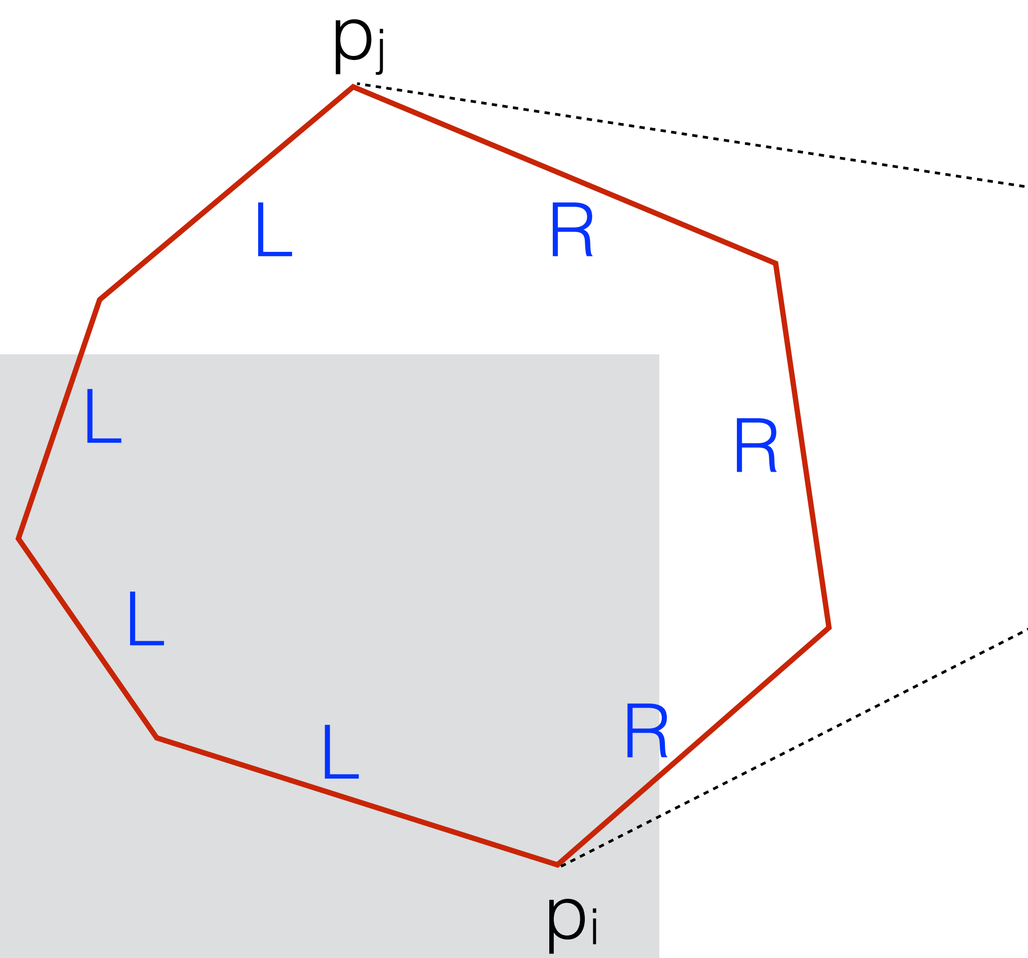
Hint: Check the orientation of p wrt the edges of the polygon.

Finding tangent points

Input: point p outside H

polygon H = [p_0, p_1,..., 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



Putting it all together

Incremental CH

- H = [p1, p2, p3]
- for i=4 to n do
  - //add p_i to H
  - if point_in_polygon(pi, H)
    - //do nothing
  - else
    - find p_k the tangent point where orientation changes from L to R
    - find p_j the tangent point where orientation changes from R to L
    - cut out the part from p_k to p_j in H (note: pk not necessarily before pj in the vertex array of H. view H as wrapping around)

Incremental CH

- H = [p1, p2, p3]
- for i=4 to n do
  - //add p_i to H
  - if point_in_polygon(pi, H)
    - //do nothing
  - else
    - find p_k the tangent point where orientation changes from L to R
    - find p_j the tangent point where orientation changes from R to L
    - cut out the part from p_k to p_j in H (note: pk not necessarily before pj in the vertex array of H. view H as wrapping around)

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 CH

- H = [p1, p2, p3]
- for i=4 to n do
  - //add p_i to H
  - if point_in_polygon(pi, H)
    - //do nothing
  - else
    - find p_k the tangent point where orientation changes from L to R
    - find p_j the tangent point where orientation changes from R to L
    - cut out the part from p_k to p_j in H (note: pk not necessarily before pj in the vertex array of H. view H as wrapping around)

Simulate the algorithm on a couple of examples. Think how p_i could come before p_j in H or the other way around.

Analysis:

Incremental CH

- H = [p1, p2, p3]
- for i=4 to n do
  - //add p_i to H
  - if point_in_polygon(pi, H) — O(i)
    - //do nothing
  - else
    - find p_k 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
    - cut out the part from p_k to p_j in H (note: pk not necessarily before pj in the vertex array of H. view H as wrapping around)

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

Incremental CH

- Improvement: pre-sort the points by their x-coordinates and add them in this order. What happens?

Incremental CH

- Improvement: pre-sort the points by their x-coordinates and add them in this order. What happens?
  - point p_i is to the right of p_{i-1}, so it will be outside CH{p_1, p_2, …, pi-1}
  - No need to check!

- pre-sort the points by their x-coordinates. Let H = [p1, p2, p3]
- for i=4 to n do
  - ~~//add p_i to H~~
  - ~~if point_in_polygon(pi, H)~~
    - ~~//do nothing~~
  - ~~else~~
    - find p_k the tangent point where orientation changes from L to R
    - find p_j the tangent point where orientation changes from R to L
    - cut out the part from p_k to p_j in H

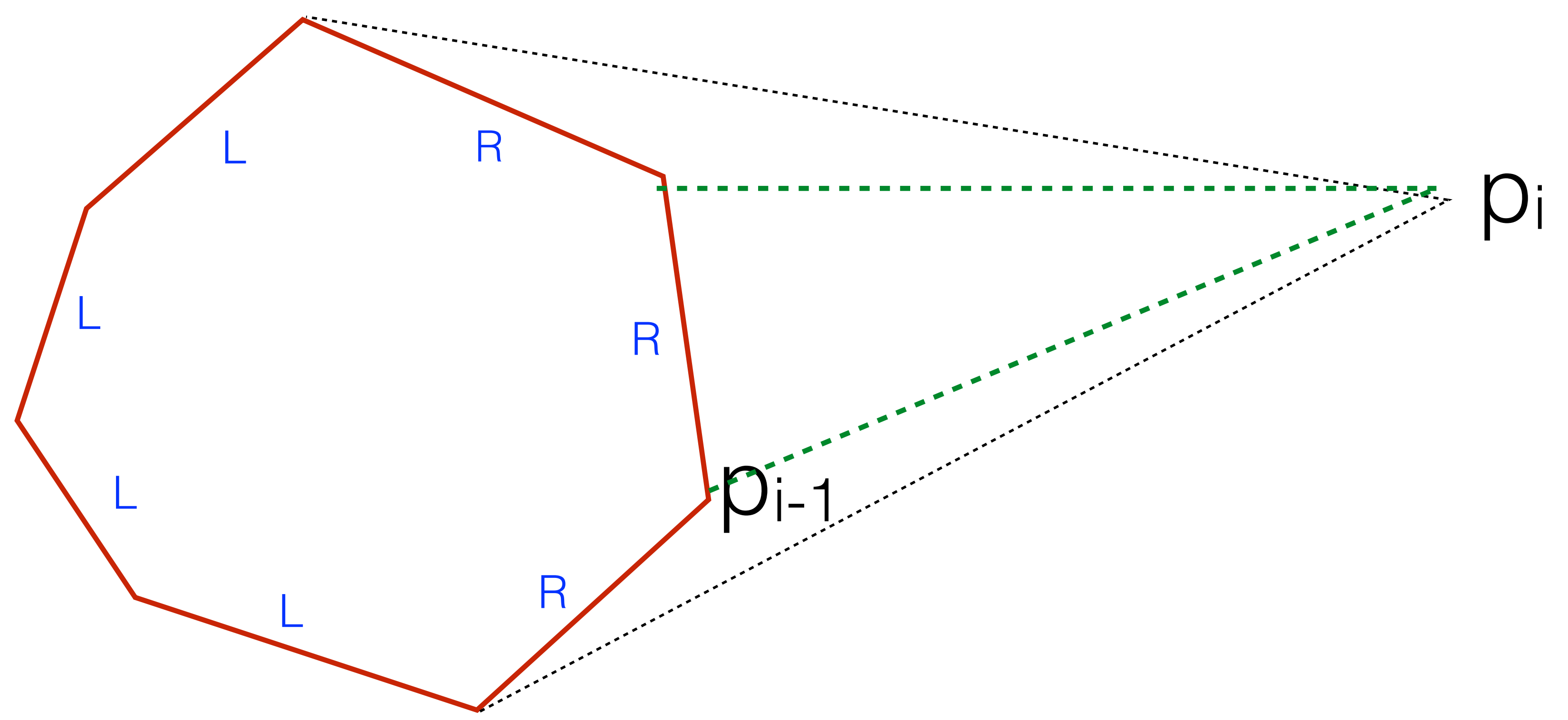
Incremental CH

- Improvement: pre-sort the points by their x-coordinates and add them in this order. What happens?
  - point p_i is to the right of p_{i-1}, so it will be outside CH{p_1, p_2, …, pi-1}
  - No need to check!

- pre-sort the points by their x-coordinates. Let H = [p1, p2, p3]
- for i=4 to n do
  - find p_k the tangent point where orientation changes from L to R
  - find p_j the tangent point where orientation changes from R to L
  - cut out the part from p_k to p_j in H

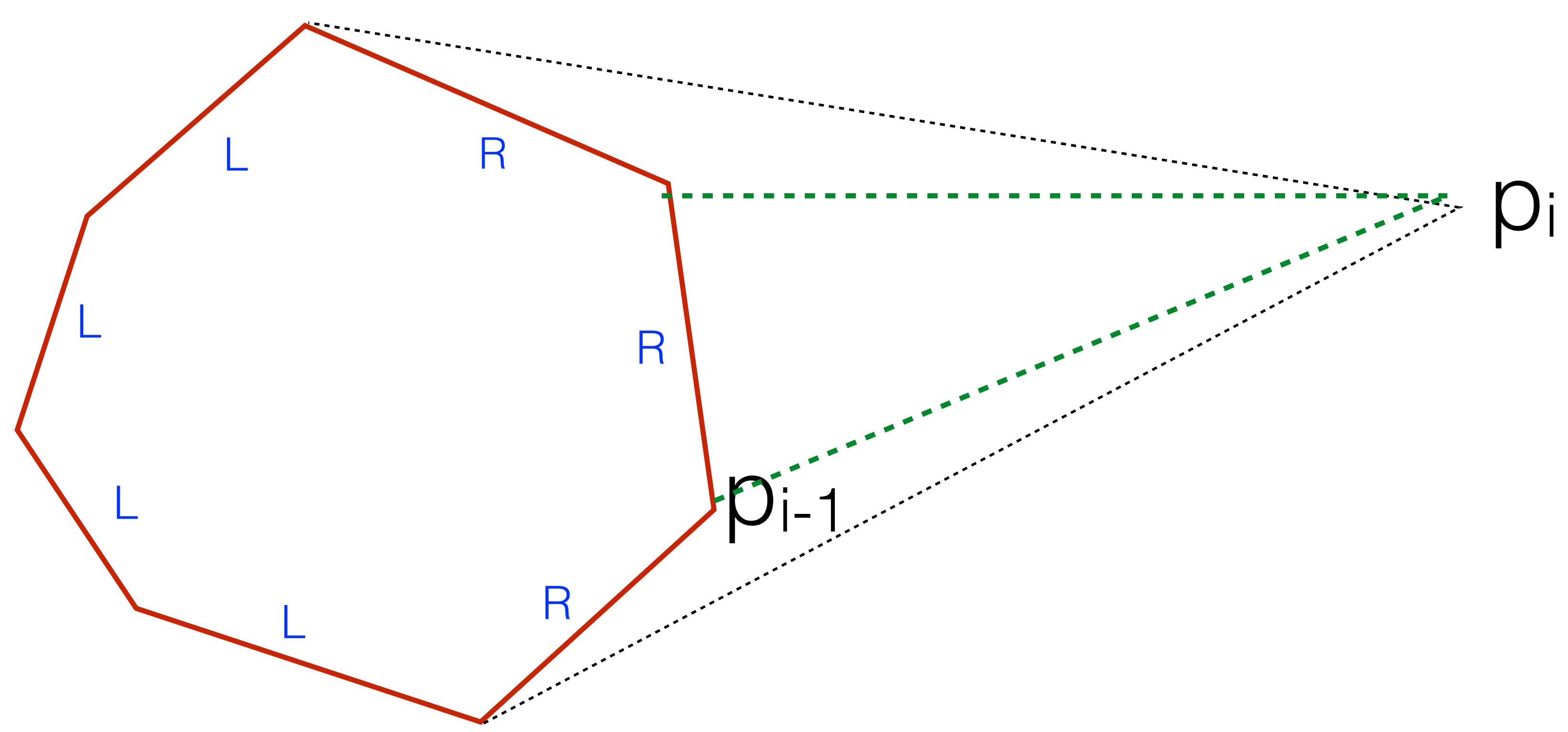
How do we make this run in O(n) once sorted?

Incremental CH



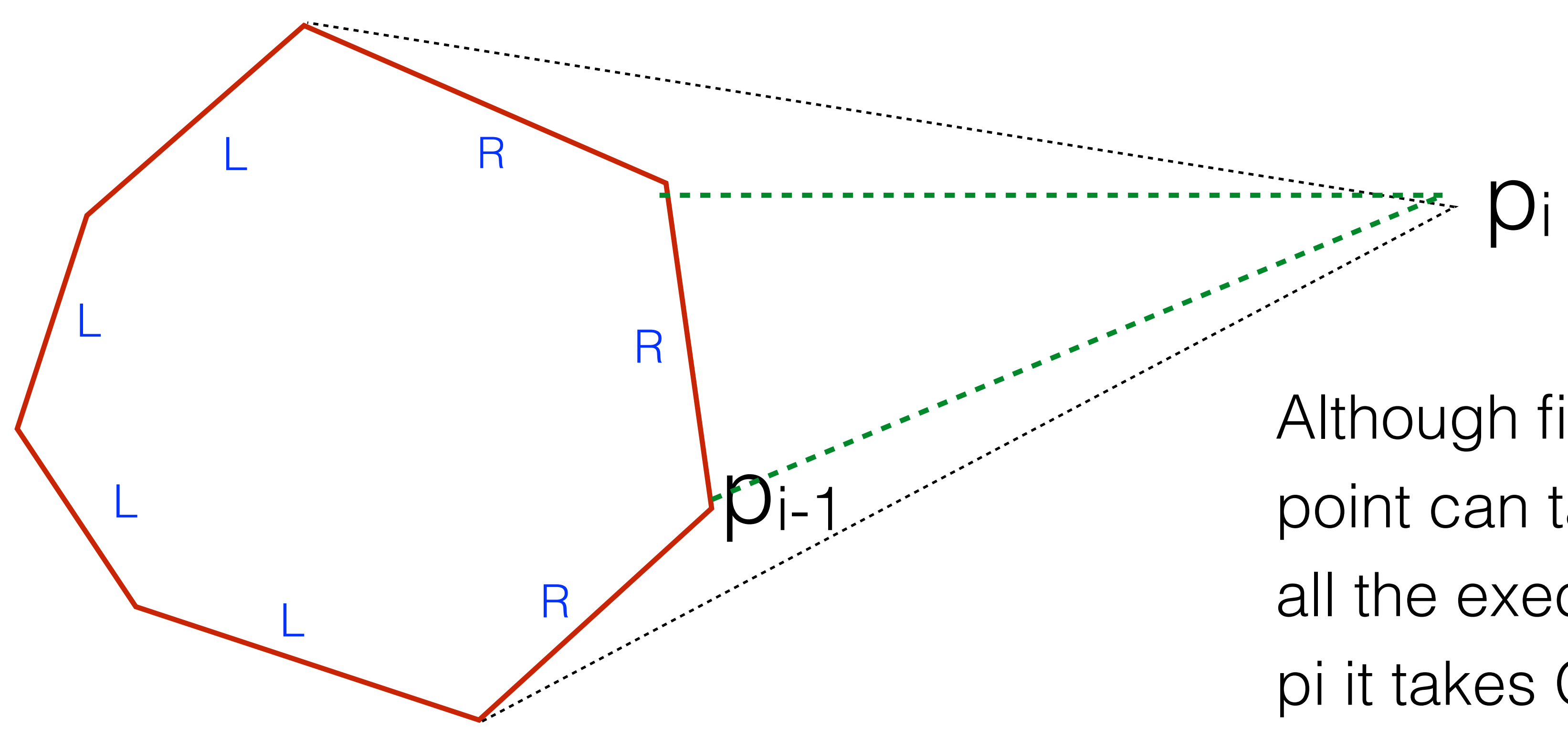
Finding tangent points of pi to the hull H of {p1, p2, …, pi-1}

- find vertex p_{i-1} on H
- v = p_{i-1}
- while point pi lies to the right of (v, succ(v)): v = succ(v)
- v is the upper tangent point
- find lower tangent point analogously



Finding tangent points of pi to the hull H of {p1, p2, …, pi-1}

- find vertex p_{i-1} on H
- v = p_{i-1}
- while point pi lies to the right of (v, succ(v)): v = succ(v)
- v is the upper tangent point
- find lower tangent point analogously



Although finding a tangent point can take O(n), over all the executions over all pi it takes O(n)

Theorem: Incremental CH (in 2D) takes $O(n \lg n)$ to sort the points followed by $O(n)$ to construct the convex hull.

A divide-and-conquer algorithm for CH

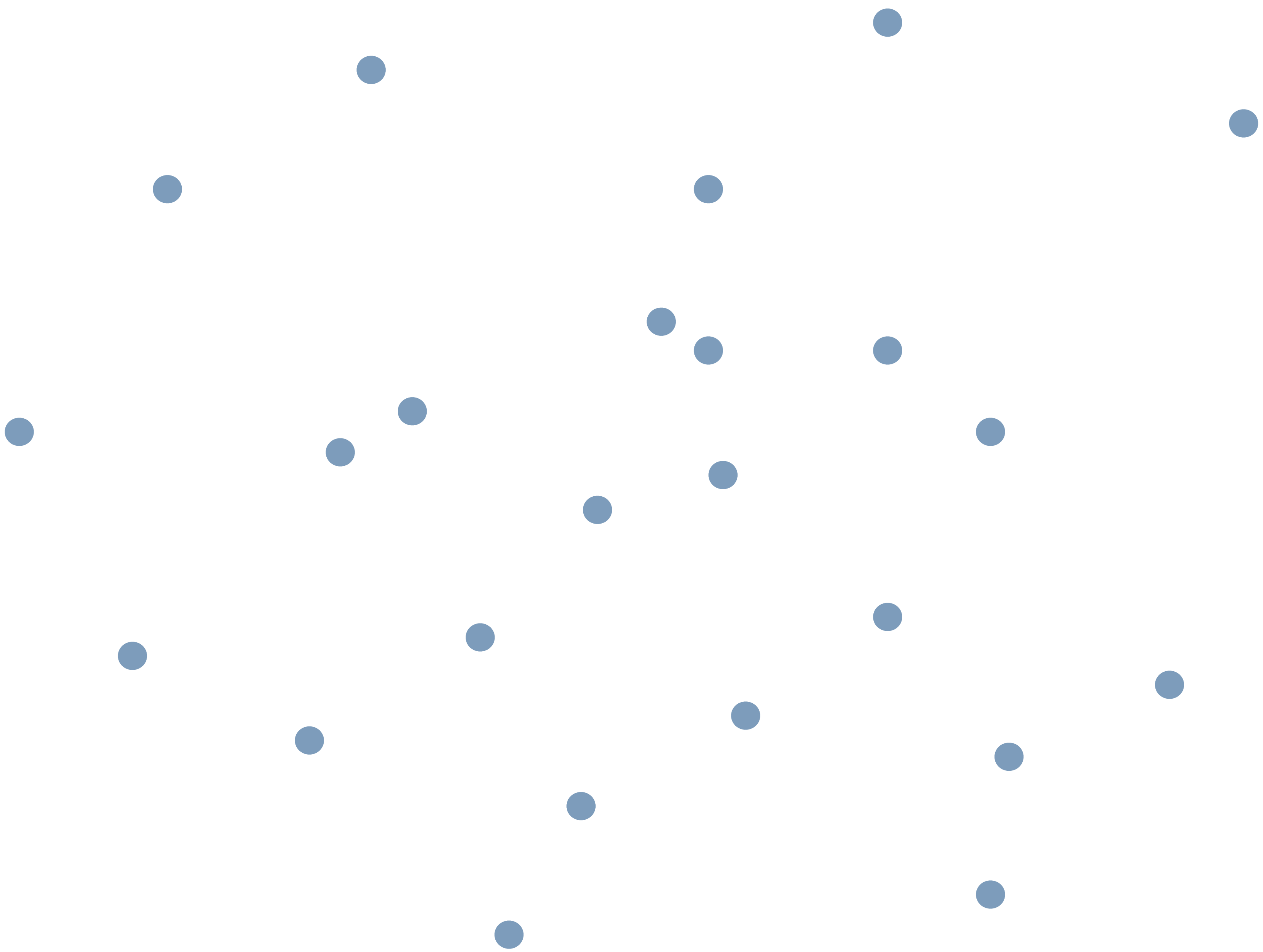
Divide-and-conquer

```
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: $T(n) = 2T(n/2) + O($merge phase$)$

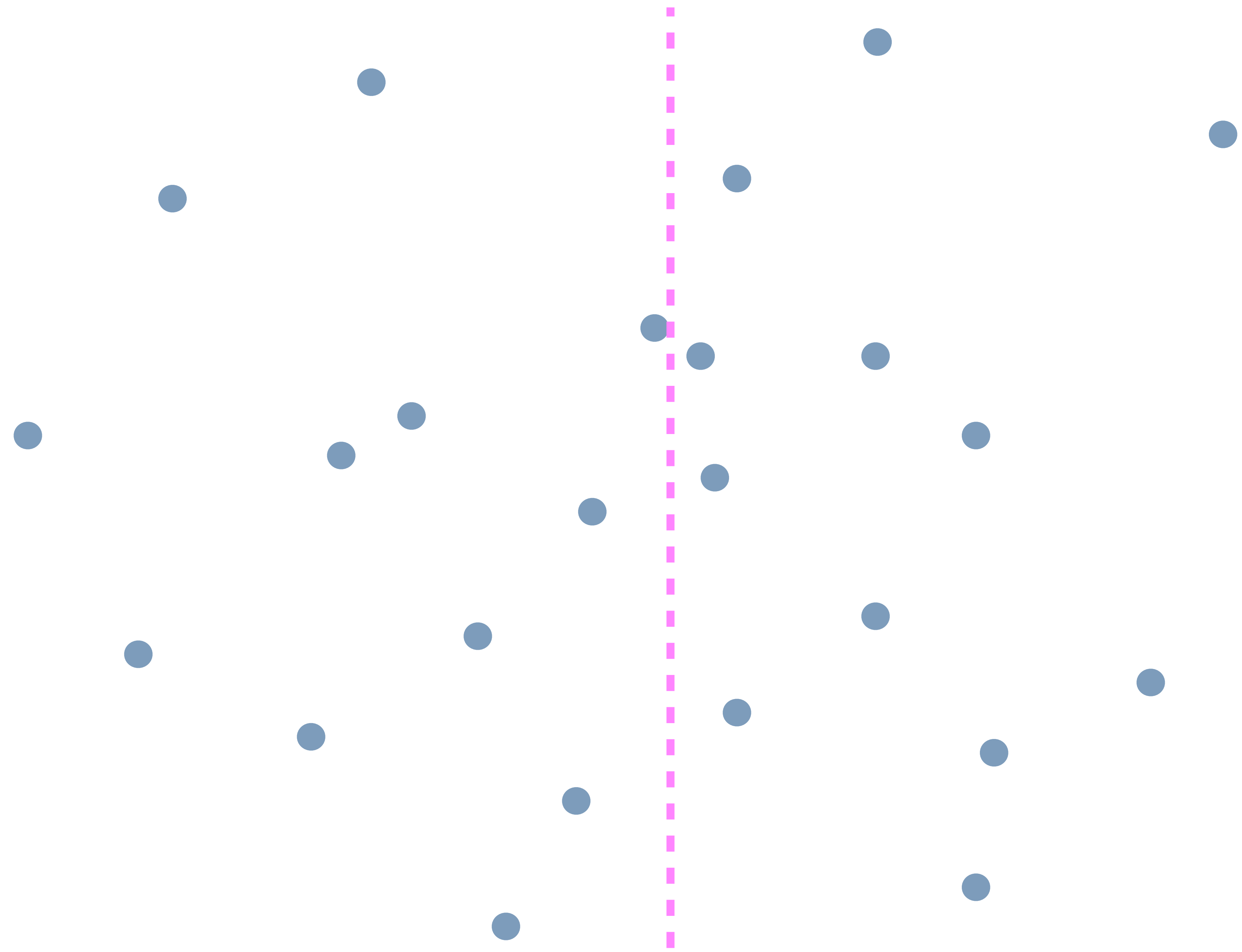
- if merge phase is **$O(n)$:** $T(n) = 2T(n/2) + O(n)$ => $O(n \lg n)$

CH via divide-and-conquer



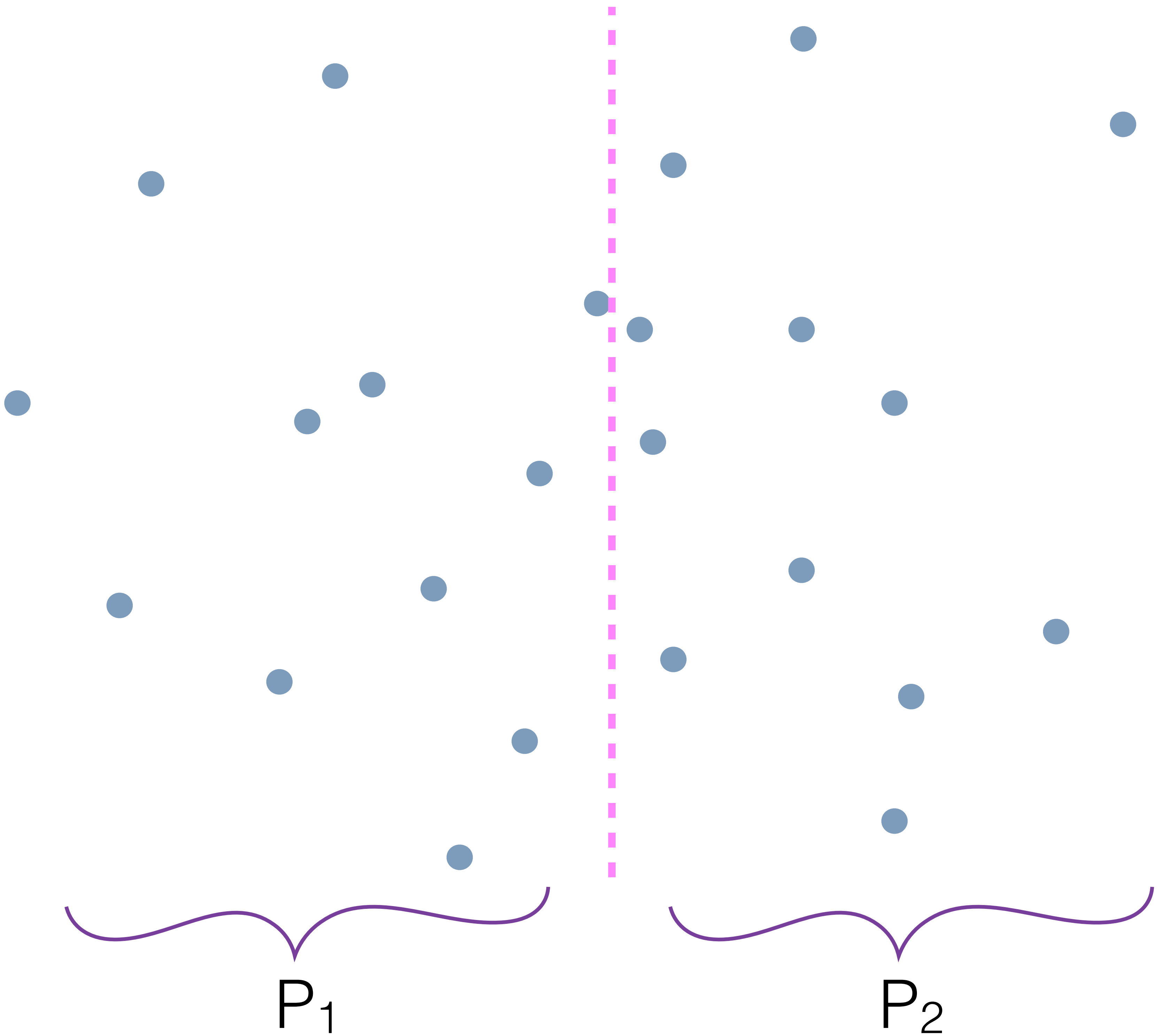
CH via divide-and-conquer

- find vertical line that splits P in half



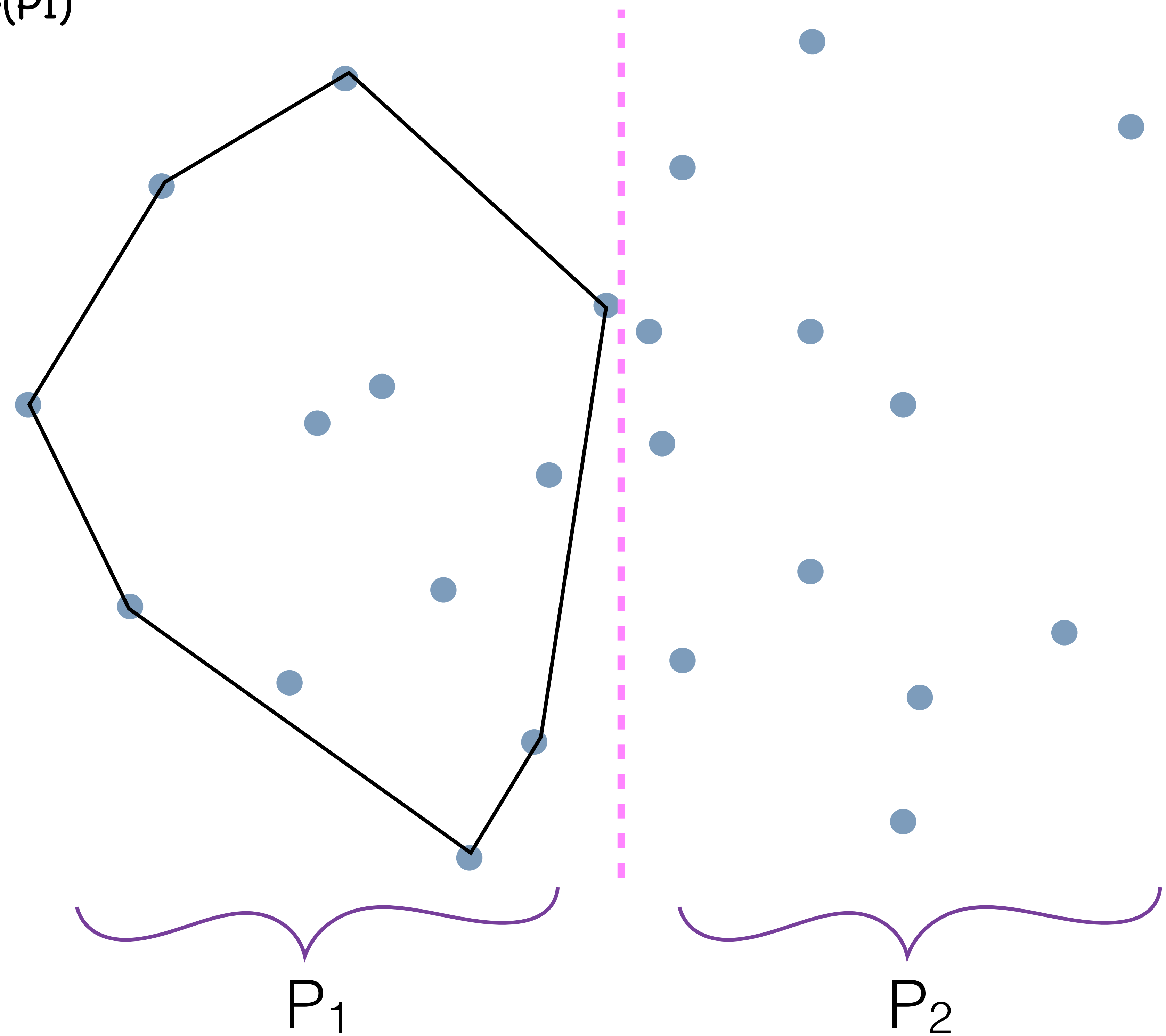
CH via divide-and-conquer

- find vertical line that splits P in half
- let P1, P2 = set of points to the left/right of line



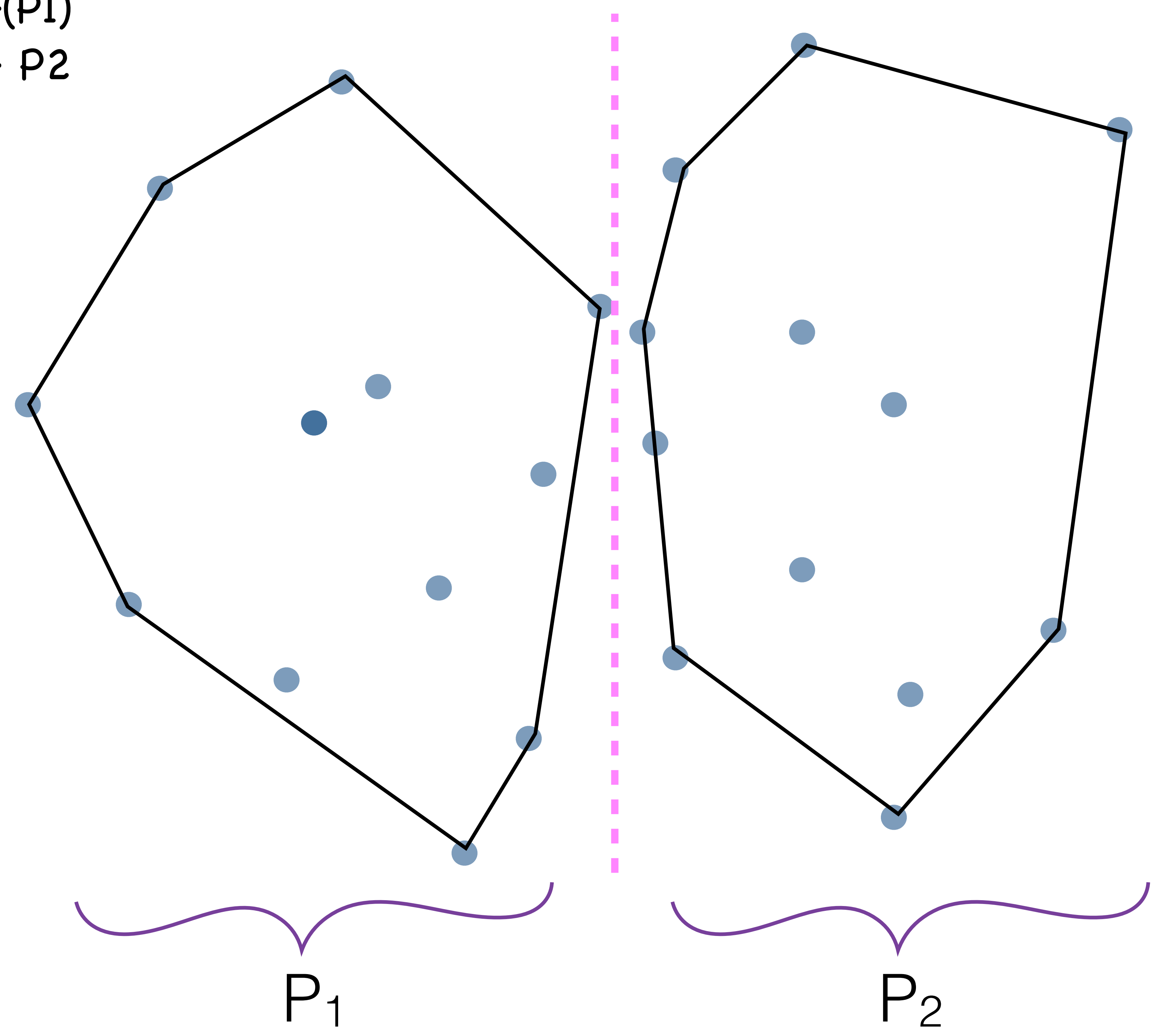
CH via divide-and-conquer

- find vertical line that splits P in half
- let P1, P2 = set of points to the left/right of line
- recursively find CH(P1)



CH via divide-and-conquer

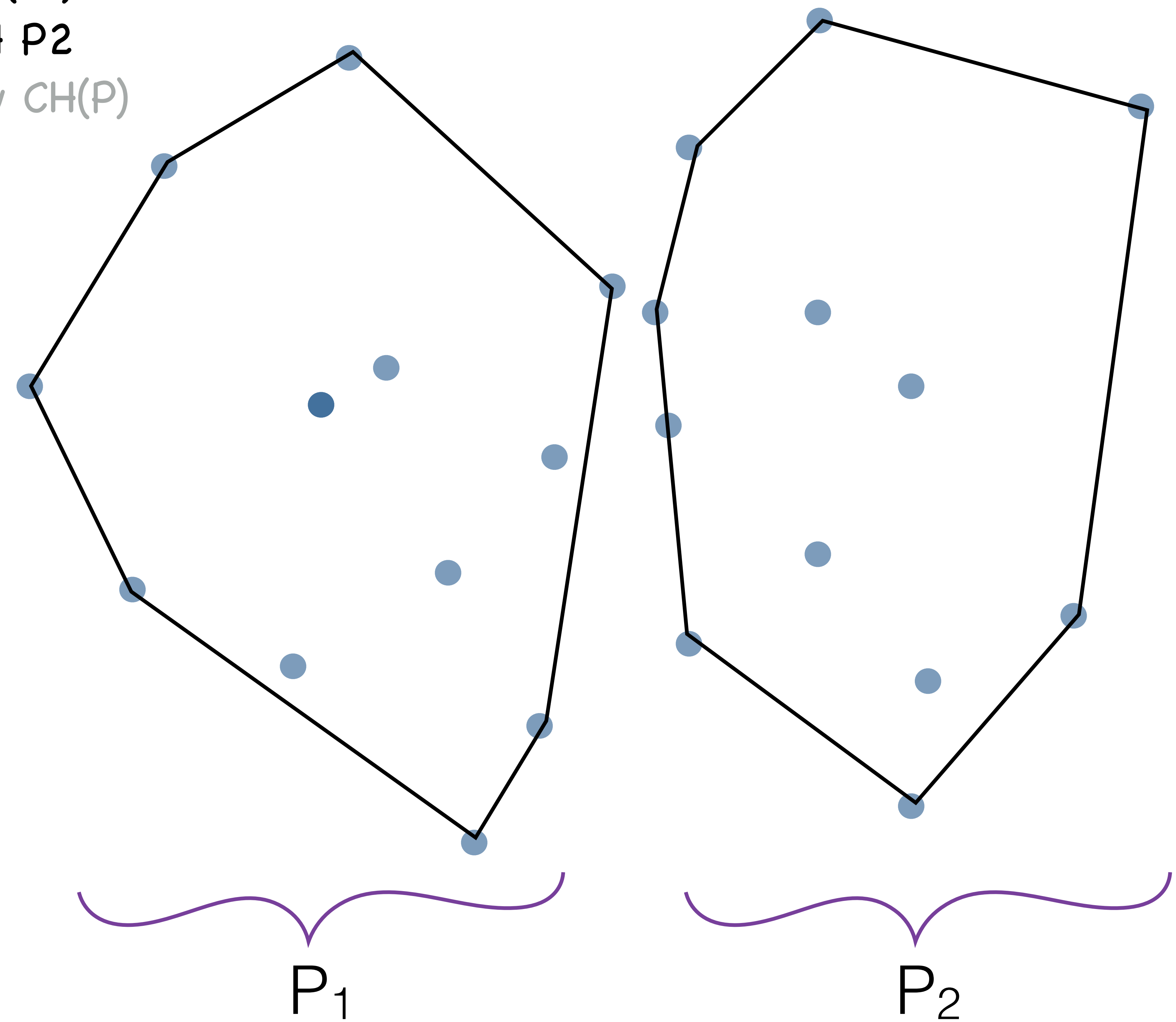
- find vertical line that splits P in half
- let P1, P2 = set of points to the left/right of line
- recursively find CH(P1)
- recursively find CH P2



CH via divide-and-conquer

- find vertical line that splits P in half
- let P1, P2 = set of points to the left/right of line
- recursively find CH(P1)
- recursively find CH P2

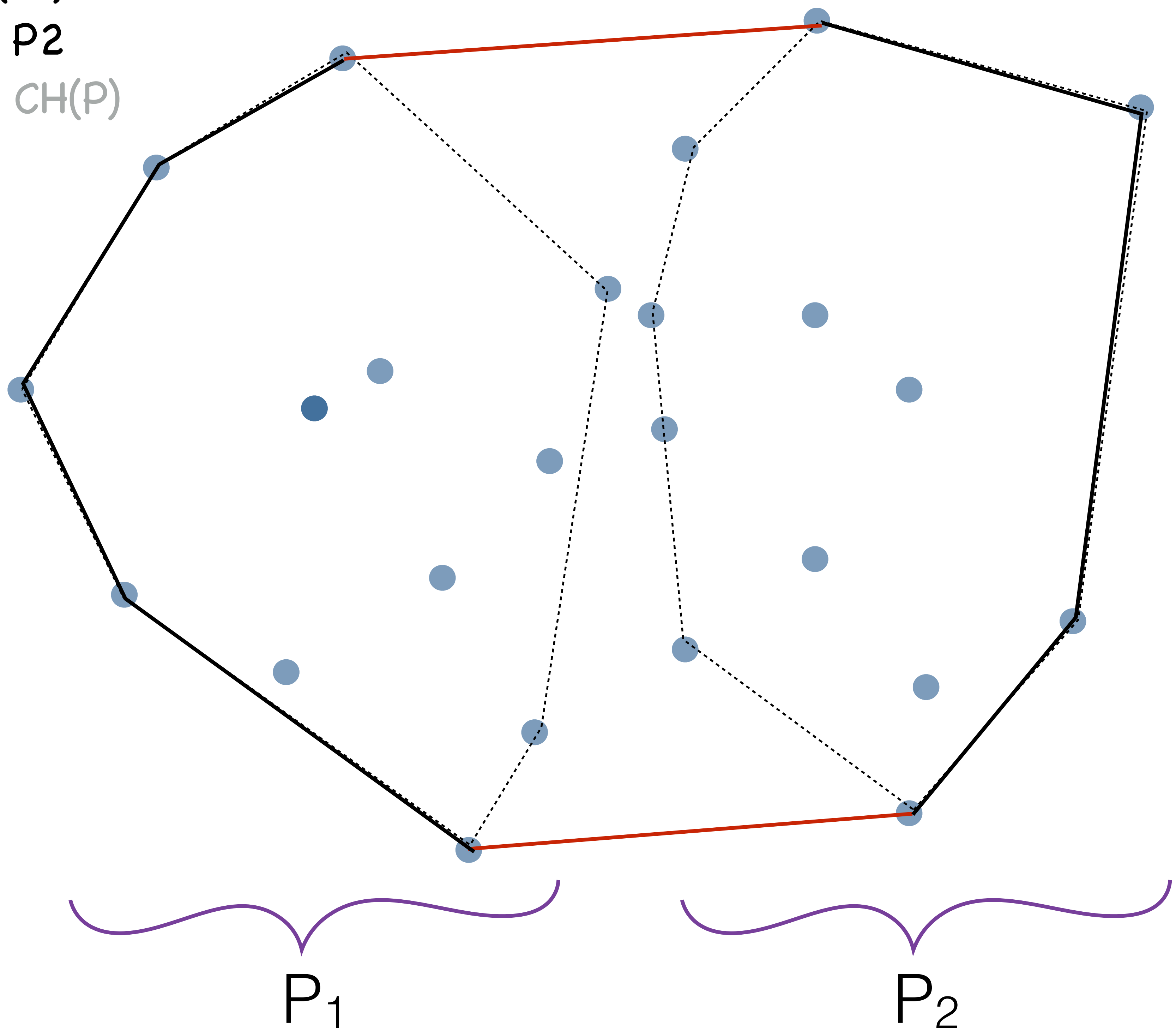
//now get somehow CH(P)



CH via divide-and-conquer

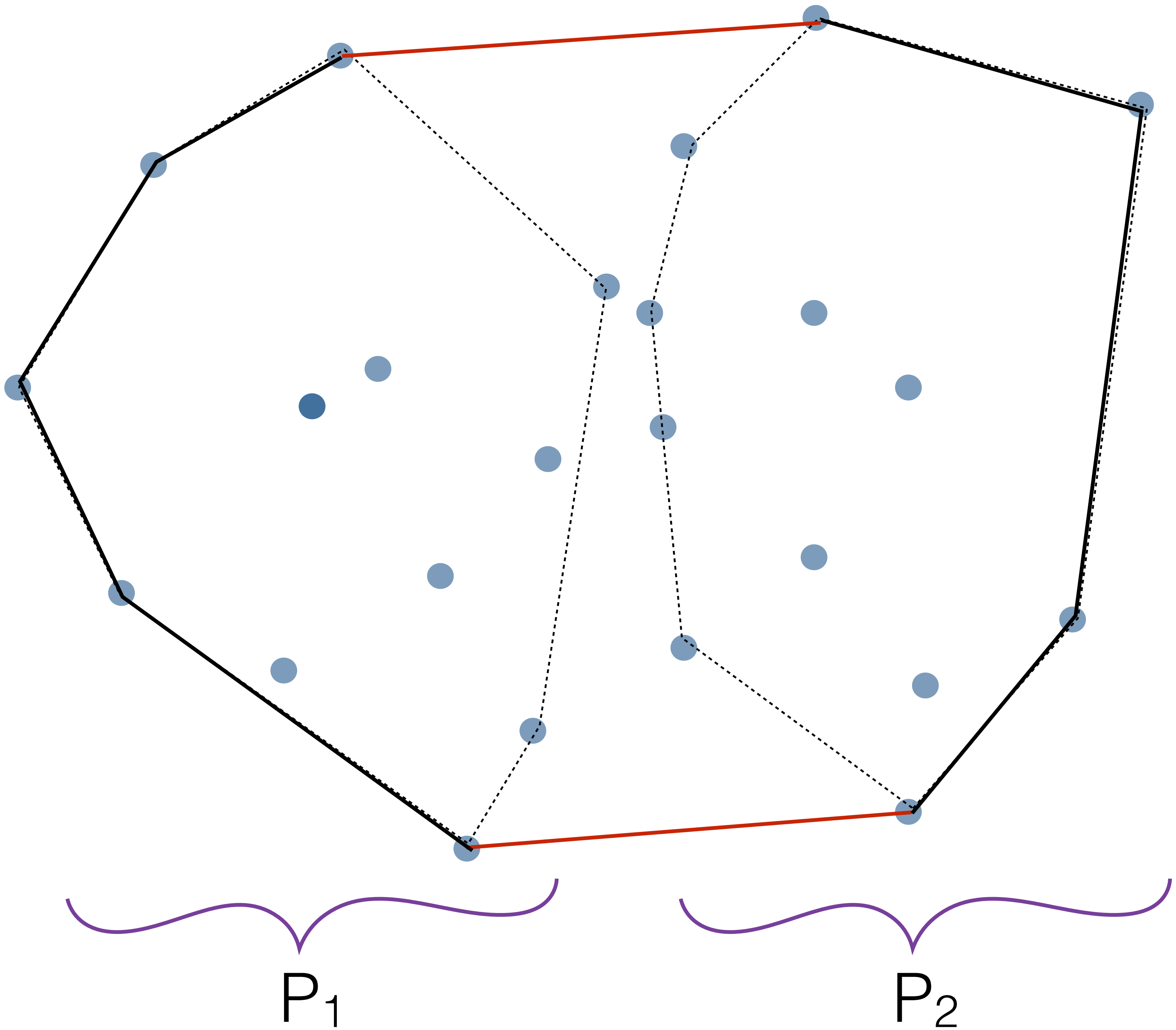
- find vertical line that splits P in half
- let P1, P2 = set of points to the left/right of line
- recursively find CH(P1)
- recursively find CH P2

//now get somehow CH(P)



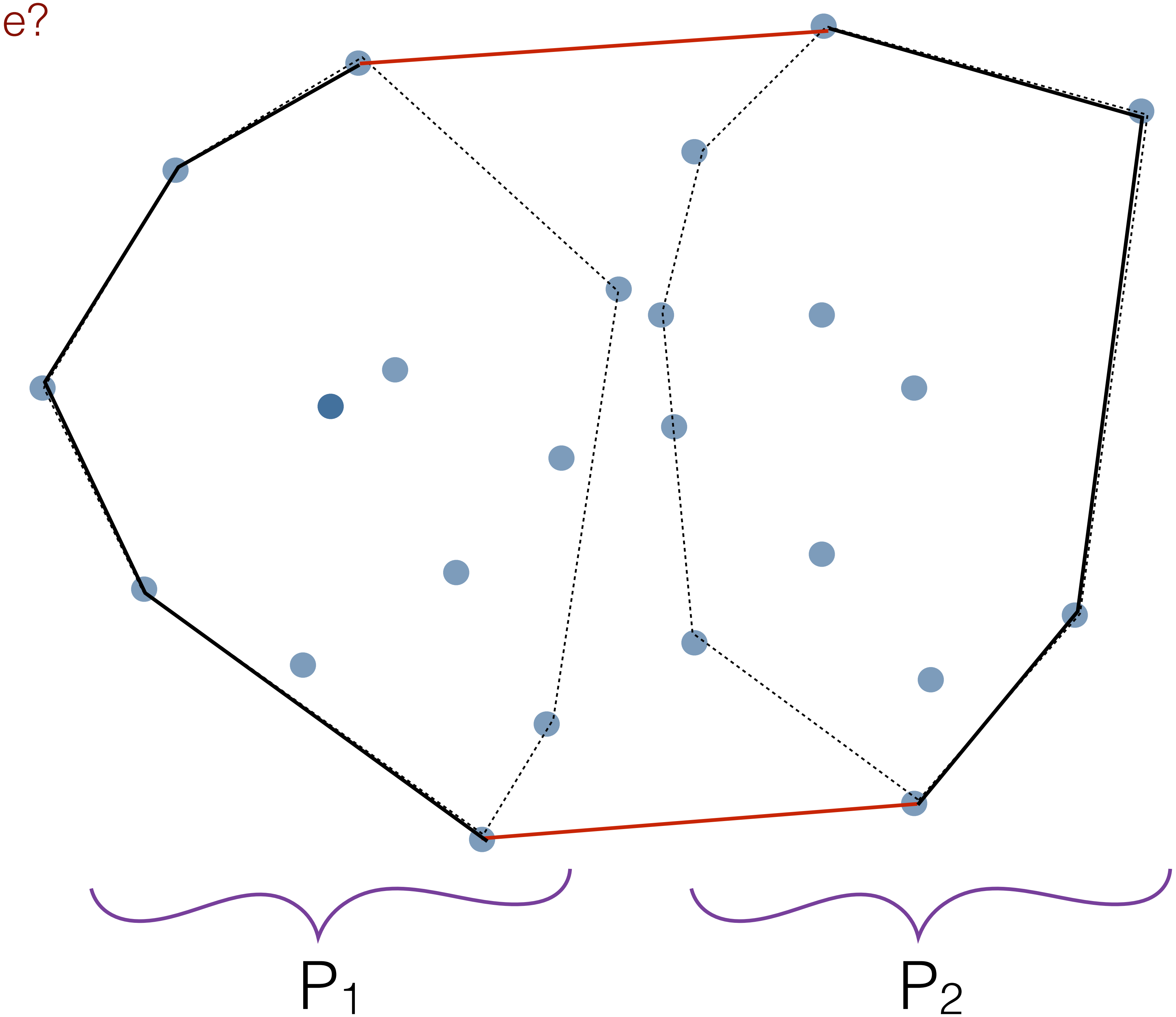
Merging two hulls..in linear time

- Need to find the two “tangents” (bridges?)



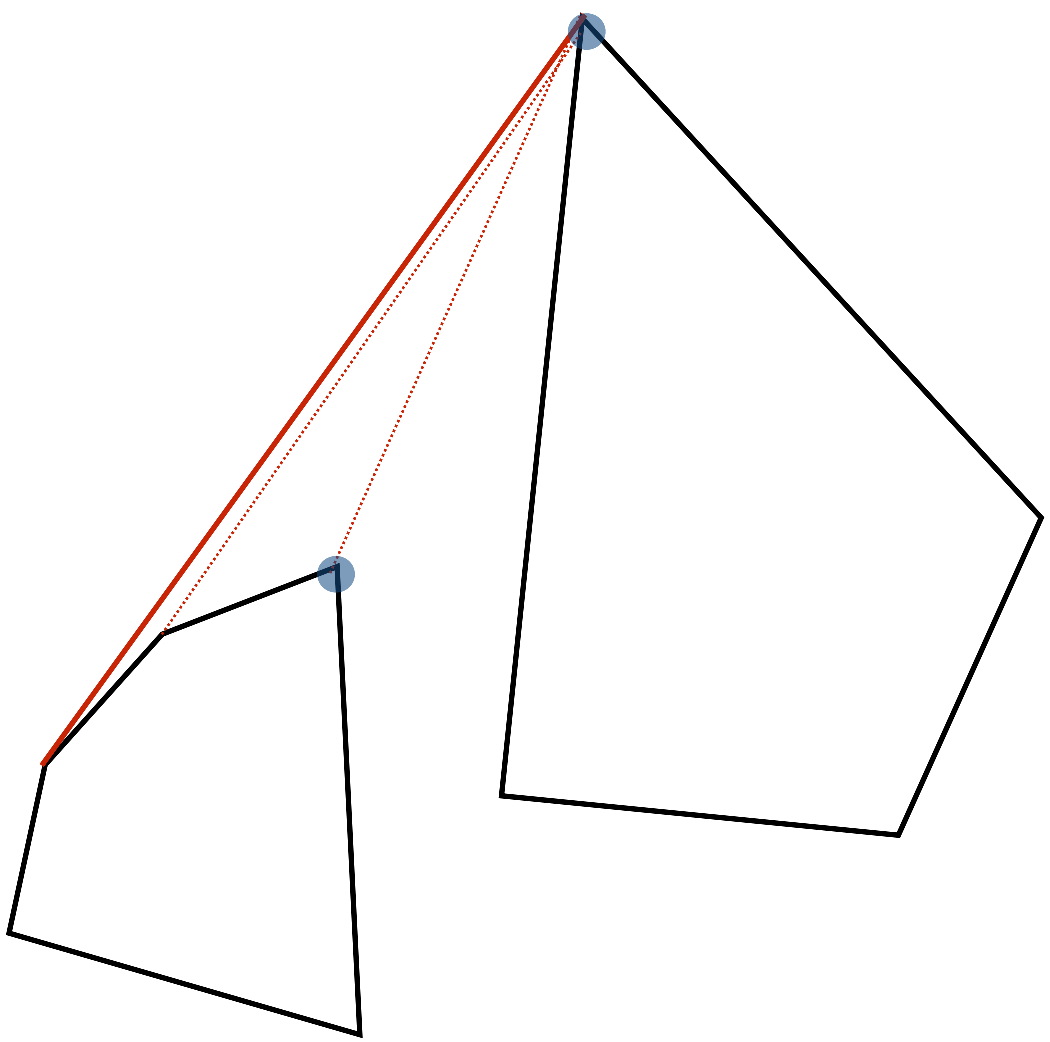
Merging two hulls..in linear time

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

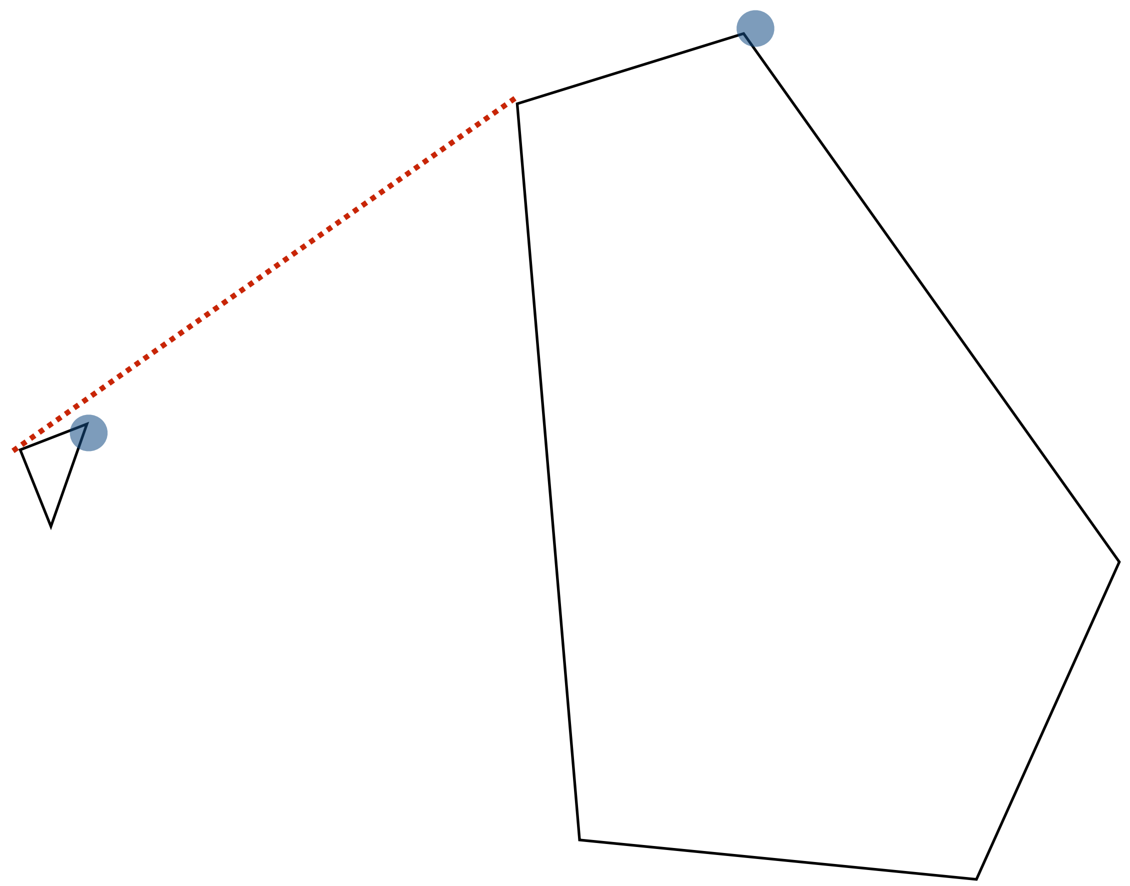


- Is the upper tangent guaranteed to connect the **top** points in P_1 and P_2 ?

Not necessarily…



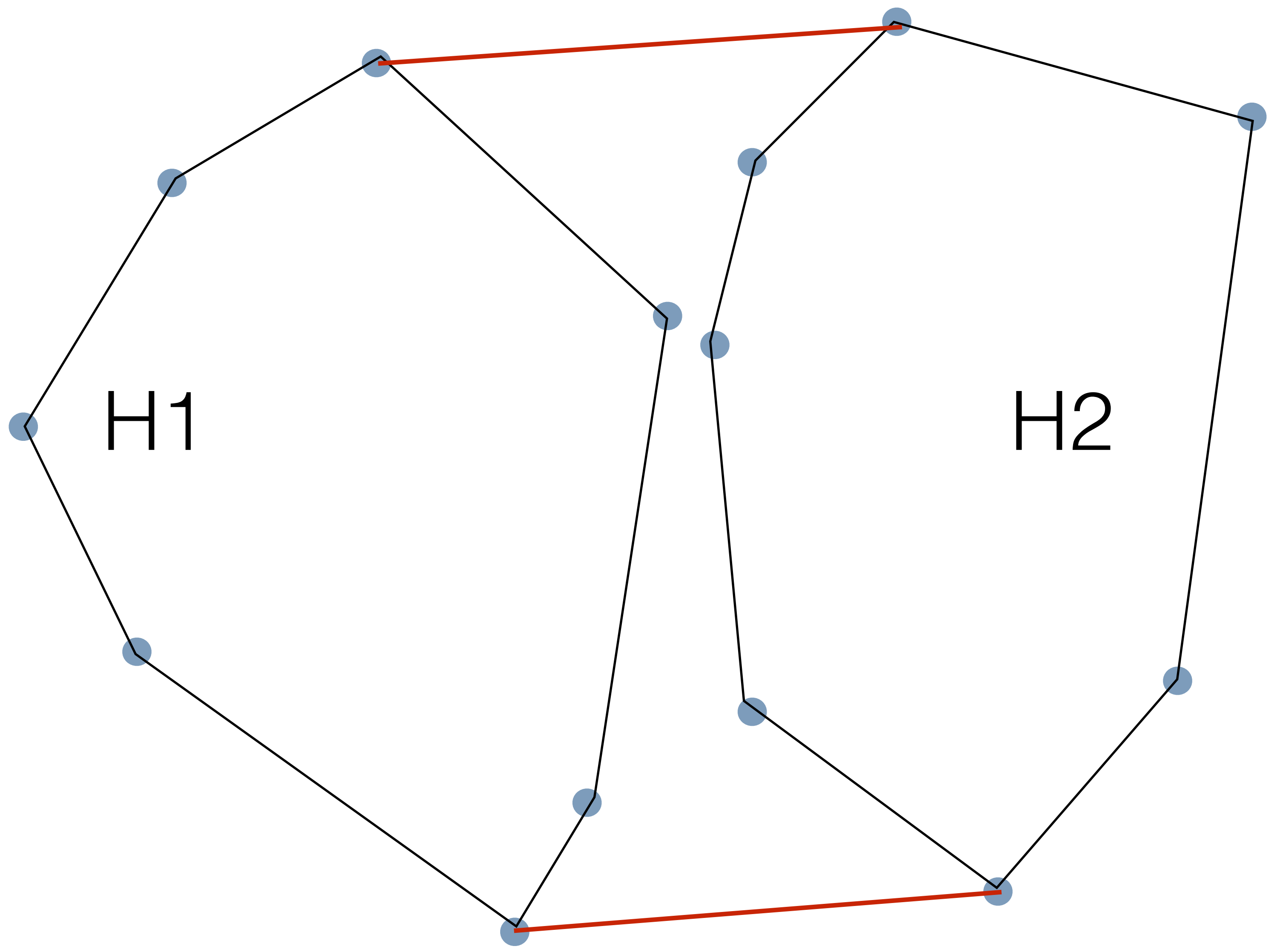
The top-most point overall is on the CH, but not necessarily on the upper tangent



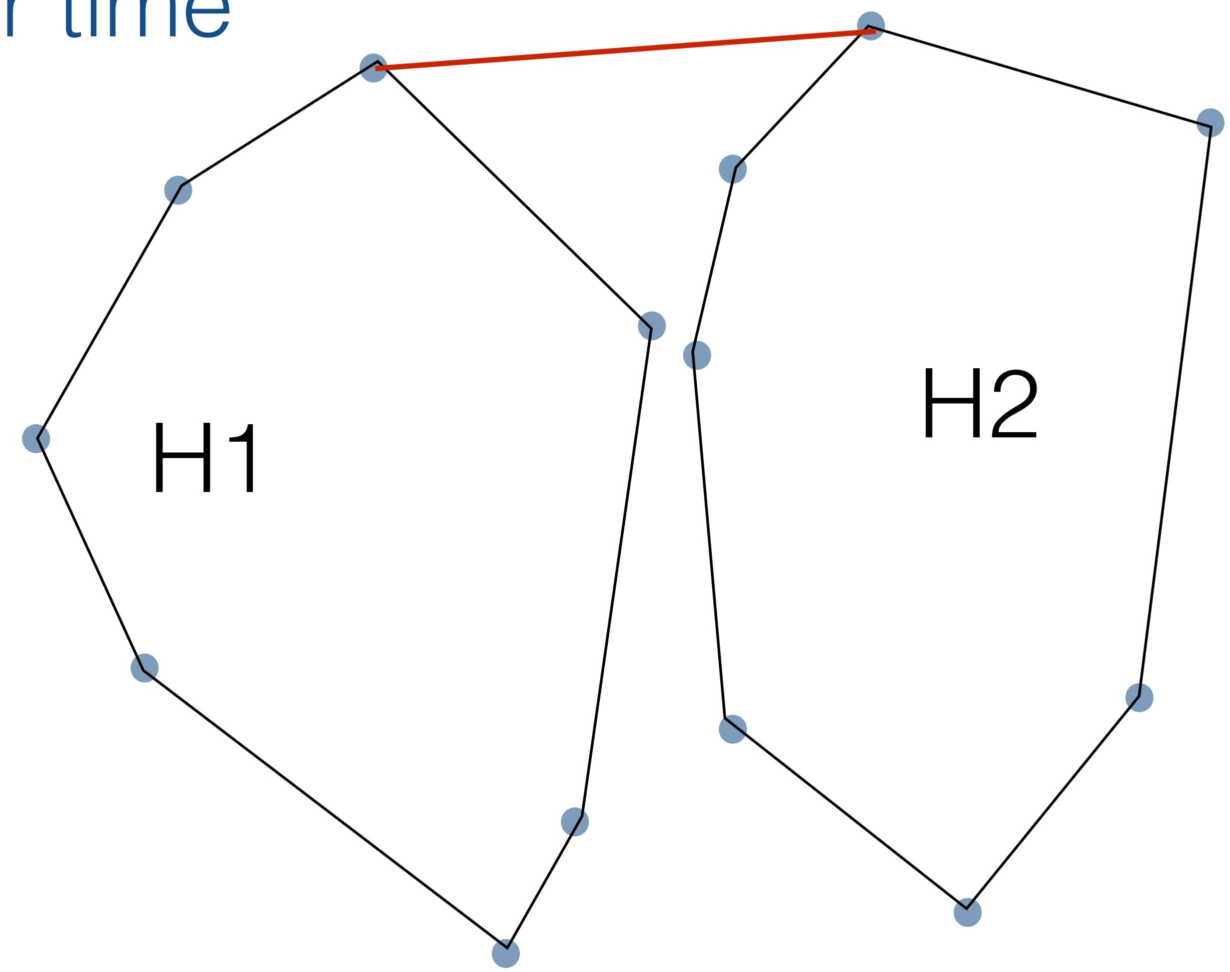
Merging two hulls..in linear time

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

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



Merging two hulls..in linear time



- To find the upper bridge:
  - let P1, P2 = set of points to the left/right of line
  - start with a = right most point of P1, b = left most point of P2
  - while one of succ(a) and pred(b) lies above line ab do:
    - if succ(a) lies above ab then set a = succ(a)
    - else : set b = pred(b)
  - return ab as the upper bridge

Theorem: D&C CH (in 2D) takes $O(n \lg n)$

CH via divide-and-conquer

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