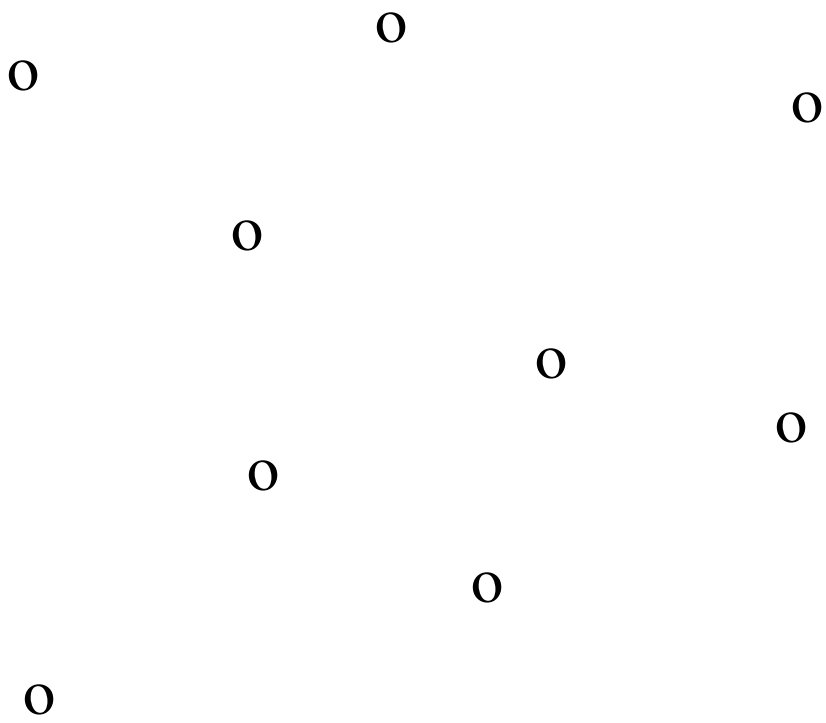
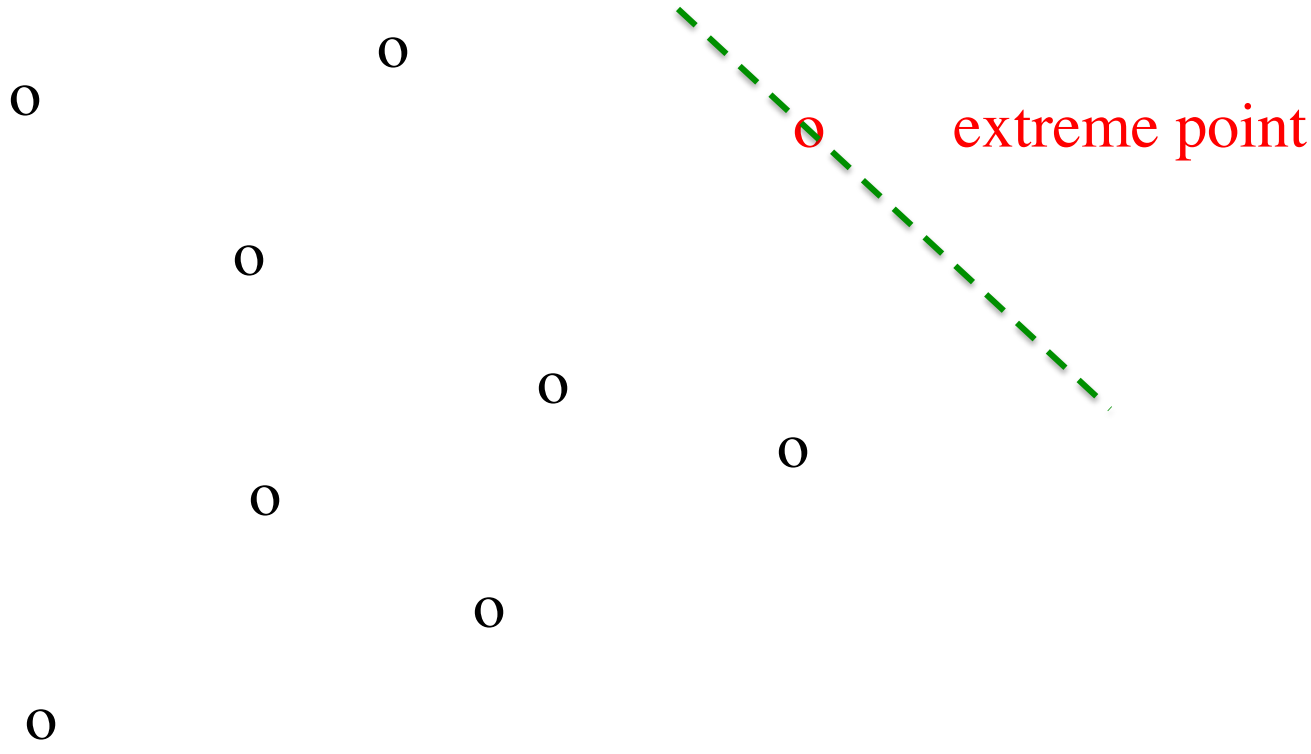
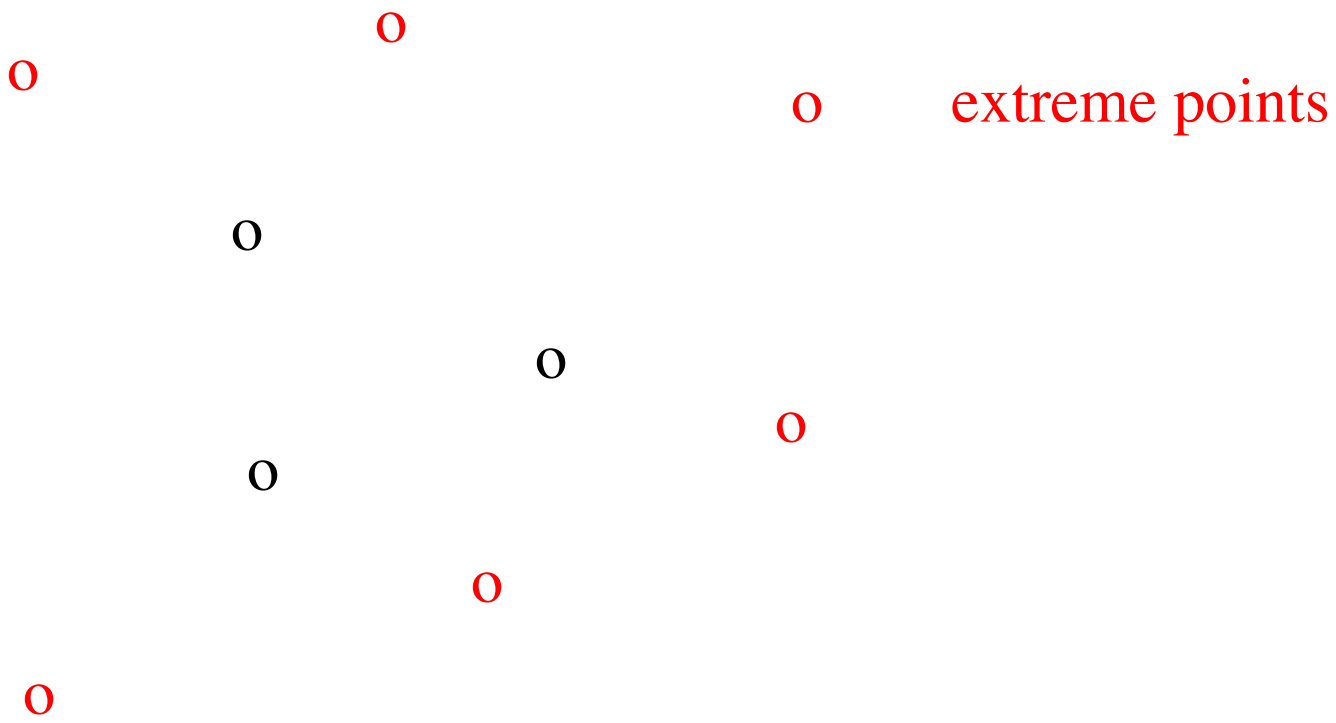
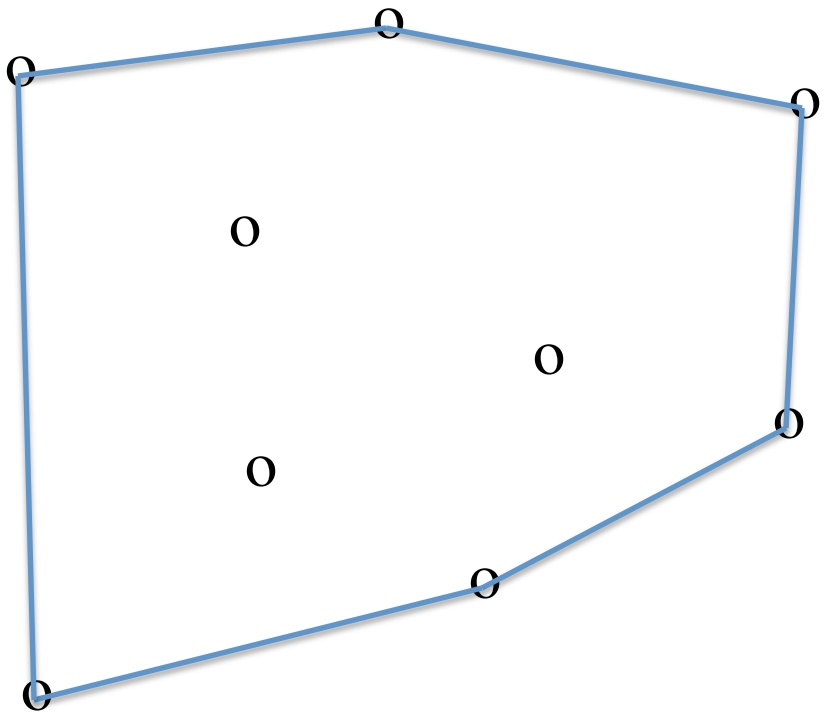
CS 420
Convex Hulls



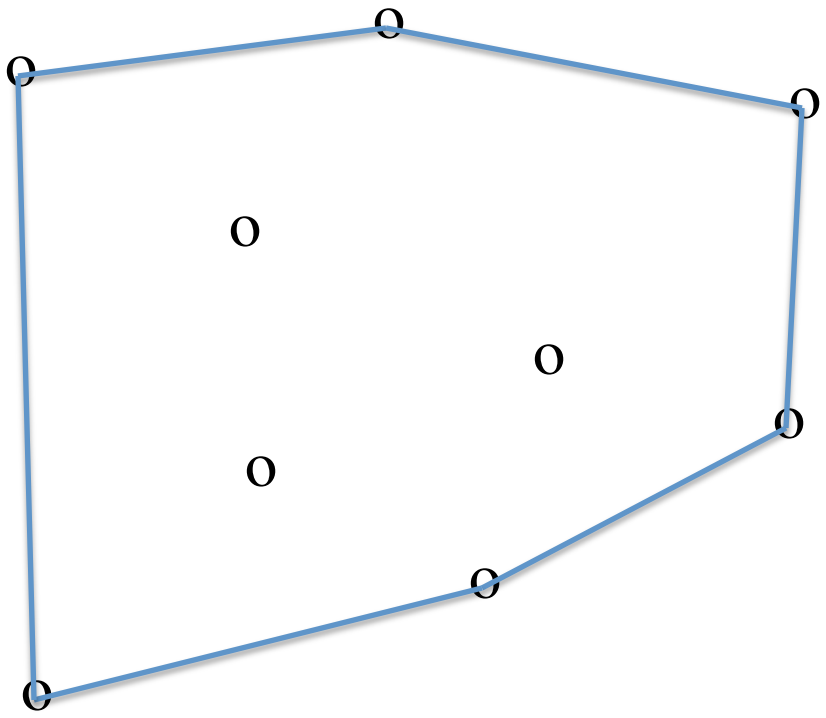
point set S


extreme point

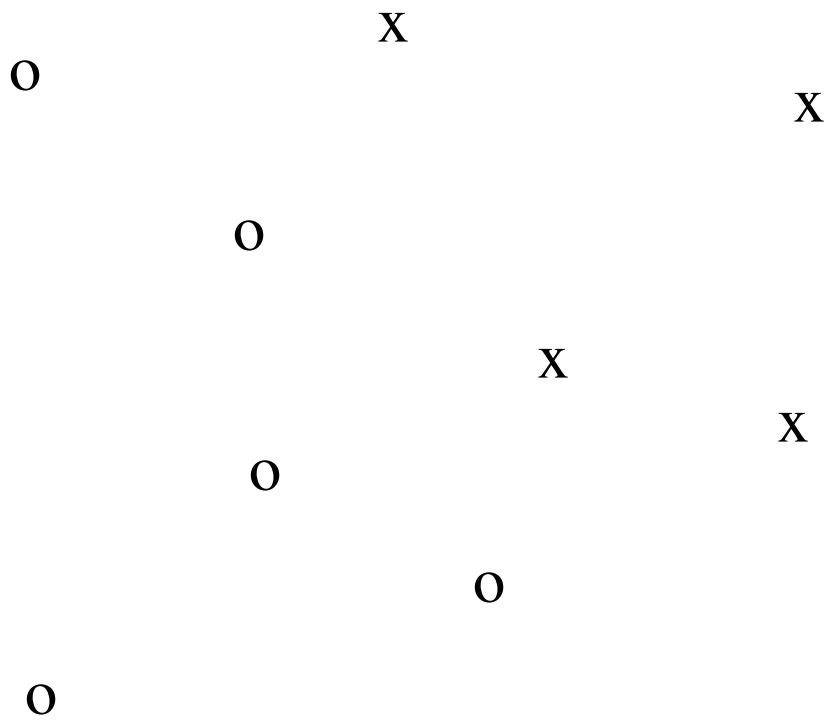

extreme points



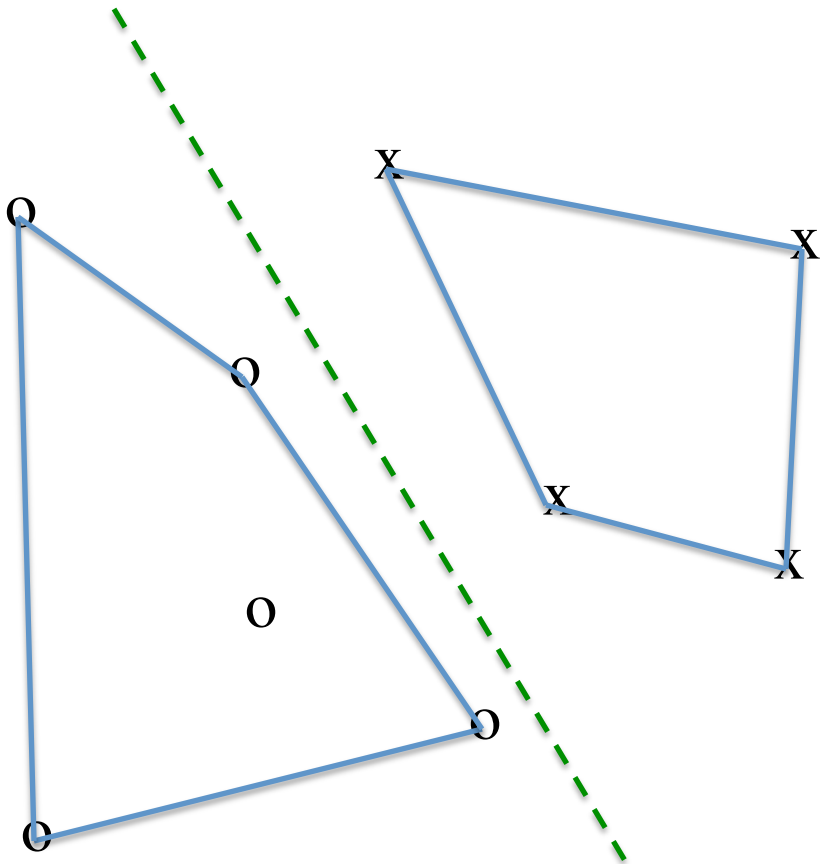
convex-hull =
polygon whose vertices
are extreme points



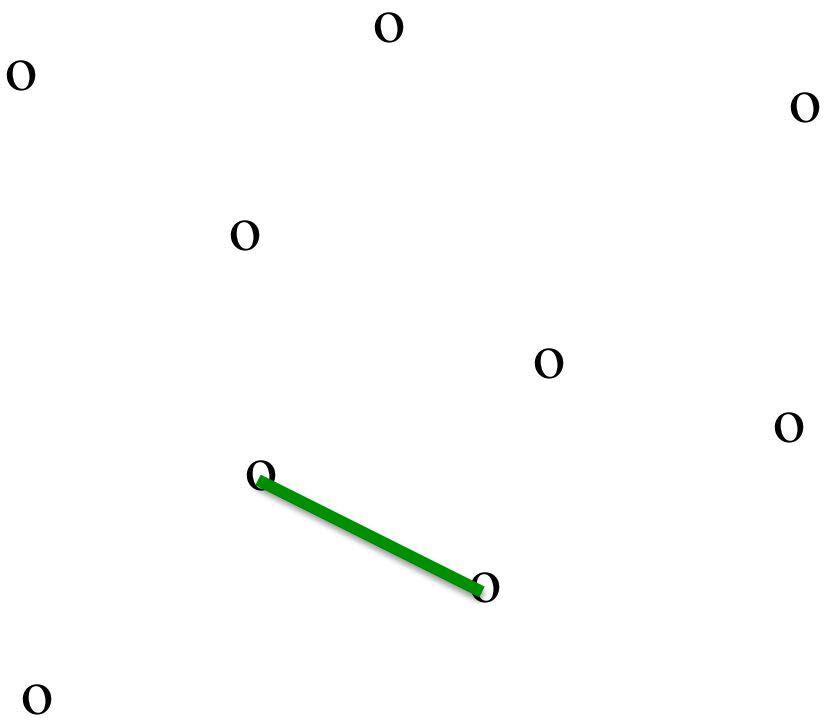
convex-hull :
shape approx



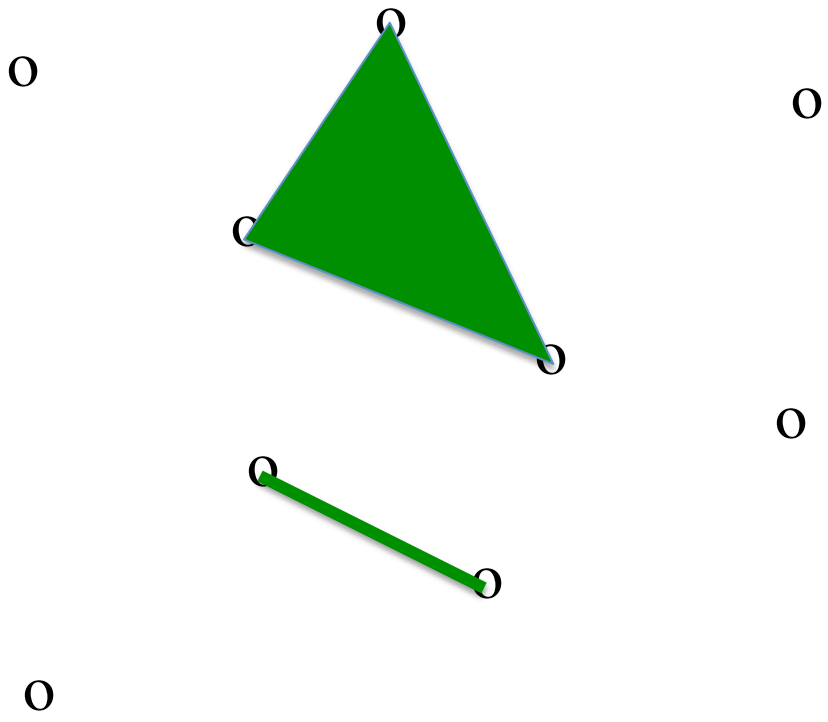
convex-hull :
shape approx
linear separability



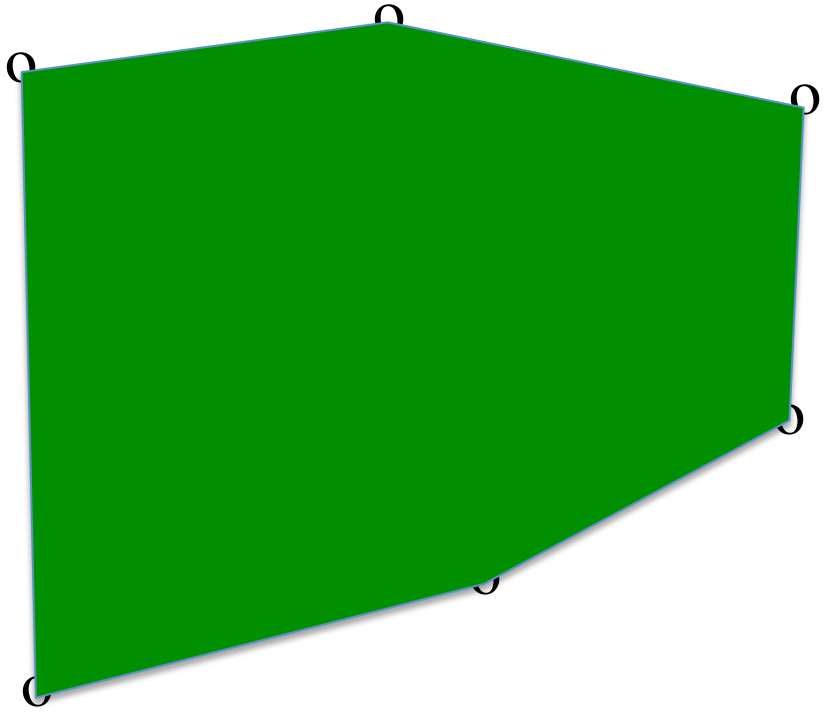
convex-hull :
shape approx
linear separability



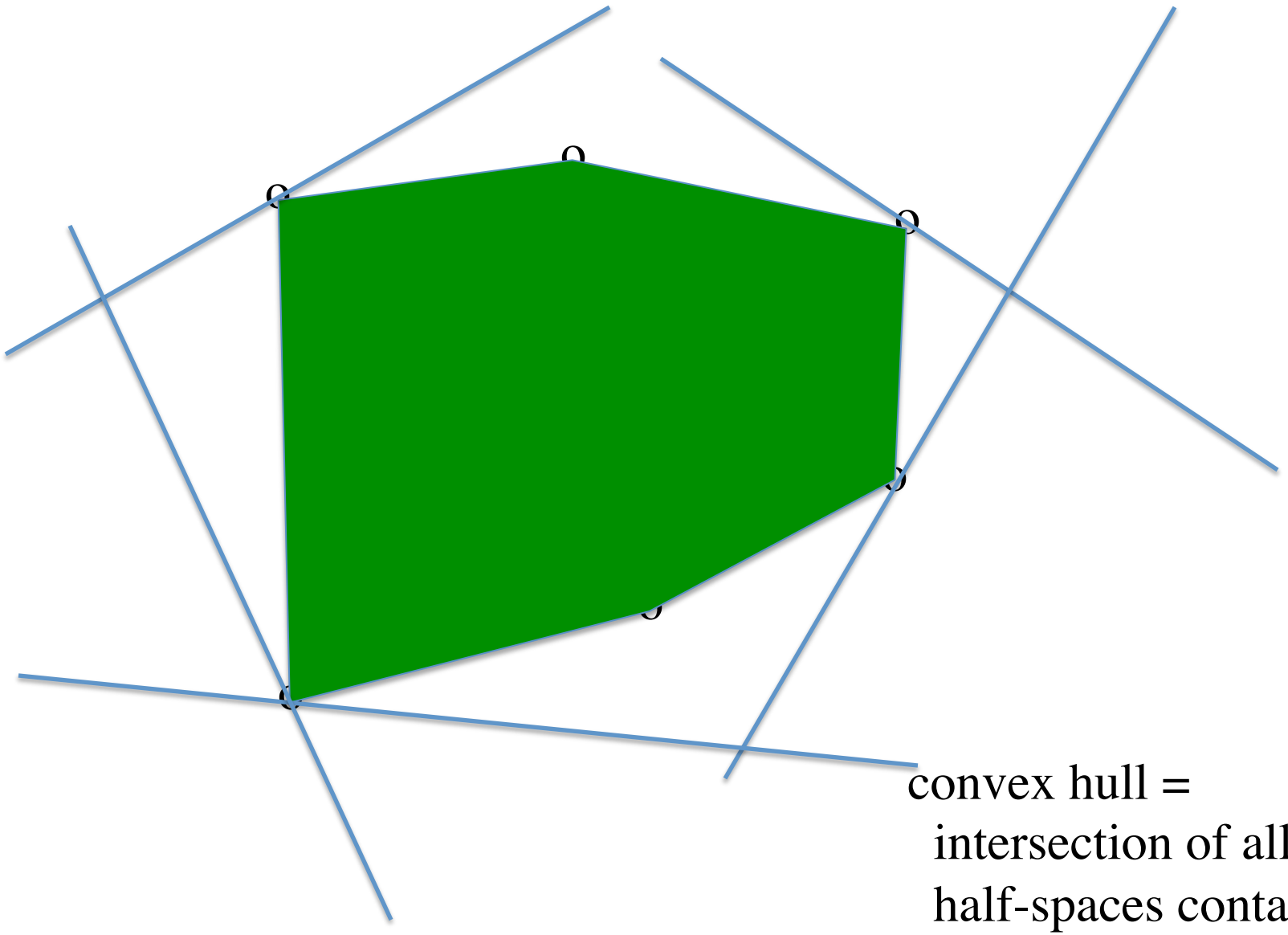
convex combination



convex combinations



convex hull =
union of all
convex combinations



convex hull =
intersection of all
half-spaces containing *S*

Convex hull CH(*S*)

- CH(*S*) is *smallest convex set* containing *S*.
- In $\mathbf{R}^2$, CH(*S*) is smallest area (or perimeter) convex polygon containing *S*.
- In $\mathbf{R}^2$, CH(*S*) is union of all triangles formed by triples of points in *S*.

Convex hull CH(*S*)

- CH(*S*) is *smallest convex set* containing *S*.
- In $\mathbf{R}^2$, CH(*S*) is smallest area (or perimeter) convex polygon containing *S*.
- In $\mathbf{R}^2$, CH(*S*) is union of all triangles formed by triples of points in *S*.
- None of these descriptions/properties yield efficient algorithms; at best $O(|S|^3)$.

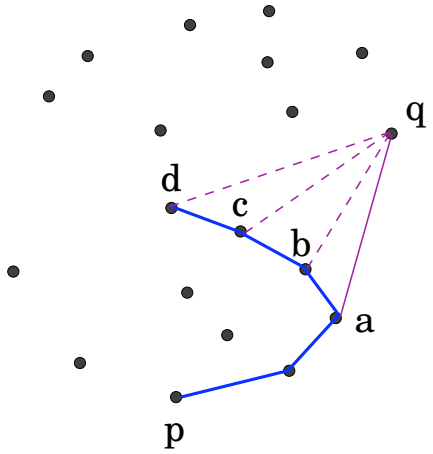
2-d convex hulls and sorting

- let $T_{sort}(n)$ and $T_{CH}(n)$ denote the worst case complexities of the sorting and convex hull problems (for input instances of size n)

2-d convex hulls and sorting

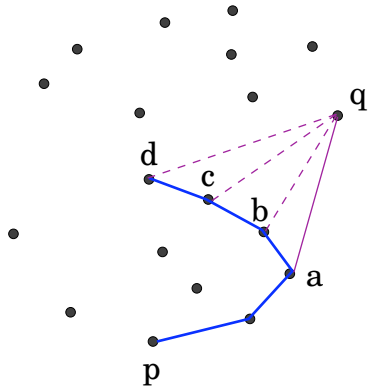
- let $T_{sort}(n)$ and $T_{CH}(n)$ denote the worst case complexities of the sorting and convex hull problems (for input instances of size n)
- we will show :
  - $T_{sort}(n) \leq T_{CH}(n) + \Theta(n)$
  - $T_{CH}(n) \leq T_{sort}(n) + \Theta(n)$

Graham Scan



1. **Sort by Y-order;** $p_1, p_2, \ldots, p_n$**.**
2. **Initialize. push** $(p_i, stack)$**,** $i = 1, 2$**.**
3. **for** $i = 3$ **to** n **do**
   while $\angle$ **next, top,** $p_i \neq$ **Left-Turn**
   pop $(stack)$
   push $(p_i, stack)$**.**
4. **return** $stack$**.**
5. **Invented by R. Graham '73. (Left and Right convex hull chains separately.)**

Analysis of Graham Scan



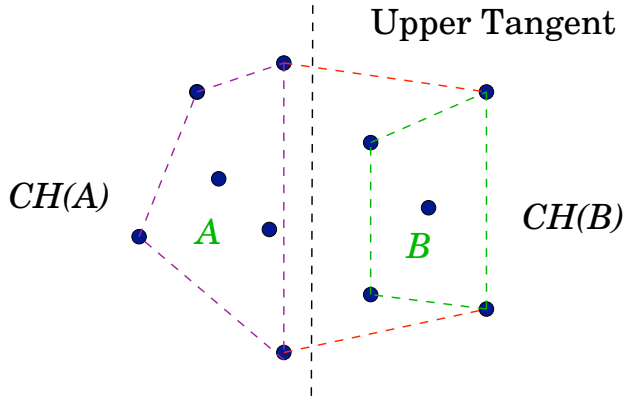
1. **Invariant: $\langle p_1, \ldots, top(stack) \rangle$ is convex. On termination, points in $stack$ are on CH.**

2. **Orientation Test:** $D = \begin{Vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{Vmatrix}$

   $\angle p, q, r$ is LEFT if $D > 0$, RIGHT if $D < 0$, and straight if $D = 0$.

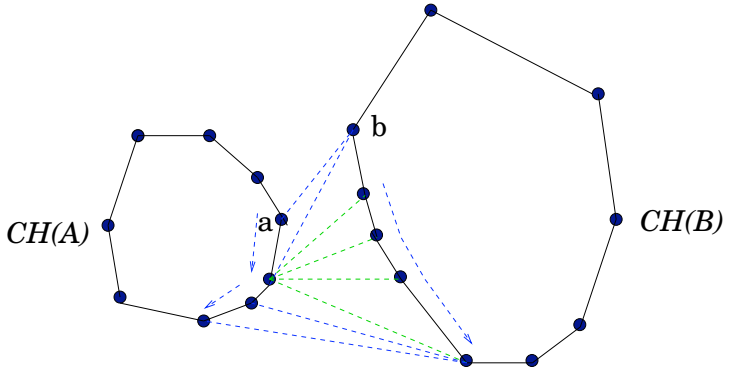
3. **After sorting, the scan takes $O(n)$ time: in each step, either a point is deleted, or added to stack.**

Divide and Conquer



- **Sort points by X-coordinates.**
- **Let A be the set of $n/2$ leftmost points, and B the set of $n/2$ rightmost points.**
- **Recursively compute $CH(A)$ and $CH(B)$.**
- **Merge $CH(A)$ and $CH(B)$ to obtain $CH(S)$.**

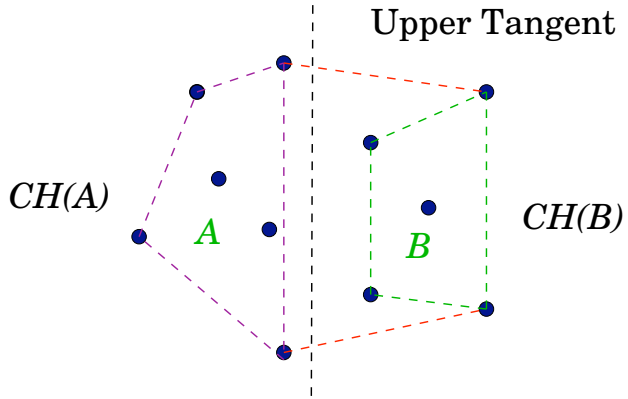
Merging Convex Hulls



Lower Tangent

- $a =$ **rightmost point of** $CH(A)$**.**
- $b =$ **leftmost point of** $CH(B)$**.**
- **while** ab **not lower tangent of** $CH(A)$ **and** $CH(B)$ **do**
  1. **while** ab **not lower tangent to** $CH(A)$ **set** $a = a - 1$ **(move** a **CW);**
  2. **while** ab **not lower tangent to** $CH(B)$ **set** $b = b + 1$ **(move** b **CCW);**
- **Return** ab

Analysis of D&C



- **Initial sorting takes $O(N \log N)$ time.**
- **Recurrence for divide and conquer**
  $T(N) = 2T(N/2) + O(N)$
- $O(N)$ **for merging (computing tangents).**
- **Recurrence solves to $T(N) = O(N \log N)$.**

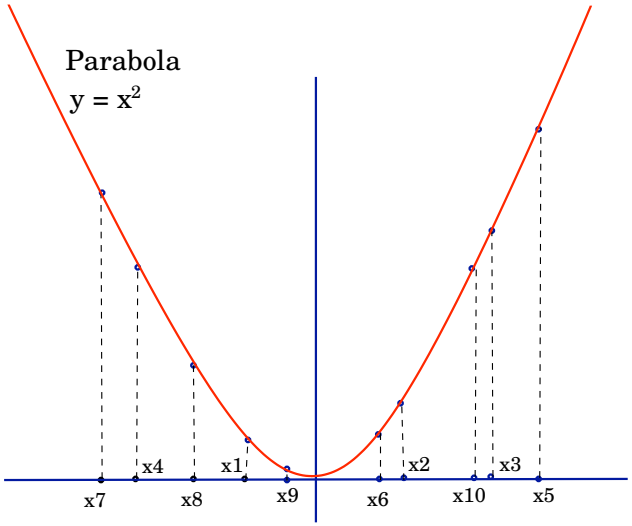
Can hulls be merged more efficiently?

What if hulls are not linearly separated?

Other sorting-inspired algorithms

Can hulls be constructed more efficiently?

Lower Bounds



- **Reduce sorting to convex hull.**
- **List of numbers to sort $\{x_1, x_2, \ldots, x_n\}$.**
- **Create point $p_i = (x_i, x_i^2)$, for each i.**
- **Convex hull of $\{p_1, p_2, \ldots, p_n\}$ has points in sorted x-order. $\Rightarrow$ CH of n points must take $\Omega(n \log n)$ in worst-case time.**

Other approaches…

- Convex hull algorithms that avoid sorting:
  - gift-wrapping (Jarvis) O(n h)
  - discard/filter interior points (QuickHull)

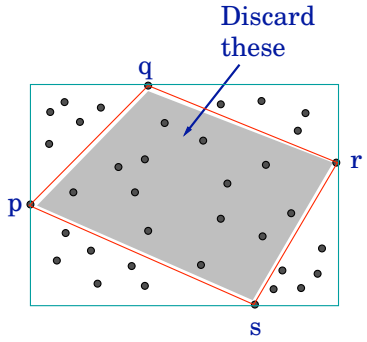
Ideas…

- More careful analysis of existing algorithms

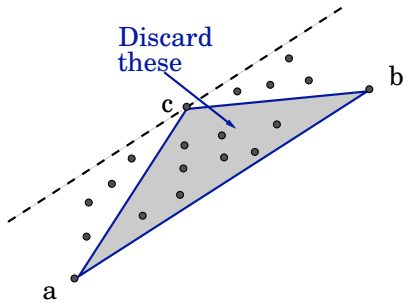
Ideas…

- More careful analysis of existing algorithms
- Try to discard non-extreme points quickly

Quick Hull Algorithm



Initialization



Recursive Elimination

1. **Form initial quadrilateral Q, with left, right, top, bottom. Discard points inside Q.**

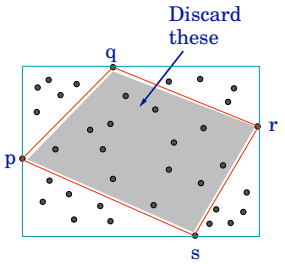
2. **Recursively, a convex polygon, with some points "outside" each edge.**

3. **For an edge ab, find the farthest outside point c. Discard points inside triangle abc.**

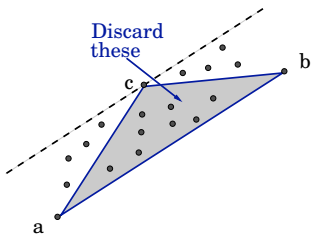
4. **Split remaining points into "outside" points for ac and bc.**

5. **Edge ab on CH when no point outside.**

Complexity of QuickHull



Initialization



Recursive Elimination

1. **Initial quadrilateral phase takes $O(n)$ time.**

2. **$T(n)$: time to solve the problem for an edge with n points outside.**

3. **Let n_1, n_2 be sizes of subproblems. Then,**

$$T(n) = \left\{ \begin{array}{ll} 1 & \textbf{if } n = 1 \\ n + T(n_1) + T(n_2) & \textbf{where } n_1 + n_2 \leq n \end{array} \right\}$$

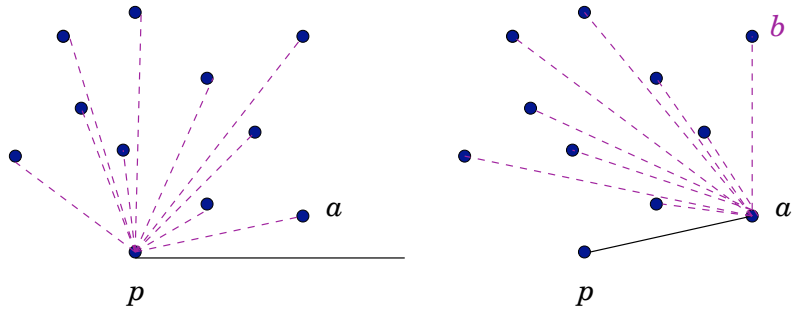
4. **Like QuickSort, this has expected running time $O(n \log n)$, but worst-case time $O(n^2)$.**

Ideas…

- More careful analysis of existing algorithms
- Try to discard non-extreme points quickly
- “wrap” around the extreme points

Efficient CH Algorithms

Gift Wrapping: [Jarvis '73; Chand-Kapur '70]

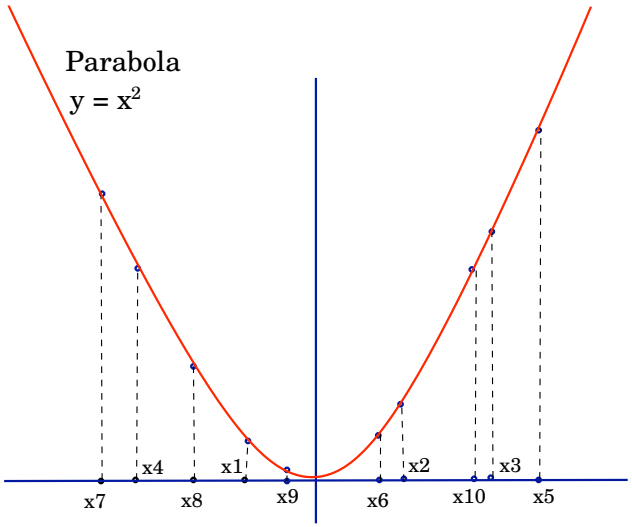


1. **Start with bottom point p.**
2. **Angularly sort all points around p.**
3. **Point a with smallest angle is on CH.**
4. **Repeat algorithm at a.**
5. **Complexity $O(Nh)$; $3 \leq h = |CH| \leq N$.**
   Worst case $O(N^2)$.

What is the complexity of finding 2-d convex hulls, in terms of *n* and *h*?

- Lower bound of $\Omega(n \lg n)$
- Jarvis' algorithm is $O(nh)$, beats the lower bound when *h* is small

Lower Bounds



- **Reduce sorting to convex hull.**
- **List of numbers to sort $\{x_1, x_2, \ldots, x_n\}$.**
- **Create point $p_i = (x_i, x_i^2)$, for each i.**
- **Convex hull of $\{p_1, p_2, \ldots, p_n\}$ has points in sorted x-order. $\Rightarrow$ CH of n points must take $\Omega(n \log n)$ in worst-case time.**
- **More refined lower bound is $\Omega(n \log h)$. LB holds even for identifying the CH vertices.**

Output-Sensitive CH

1. **Kirkpatrick-Seidel (1986) describe an $O(n \log h)$ worst-case algorithm. Always optimal—linear when $h = O(1)$ and $O(n \log n)$ when $h = \Omega(n)$.**

2. **T. Chan (1996) achieved the same result with a much simpler algorithm.**

3. **Remarkably, Chan's algorithm combines two slower algorithms (Jarvis and Graham) to get the faster algorithm.**

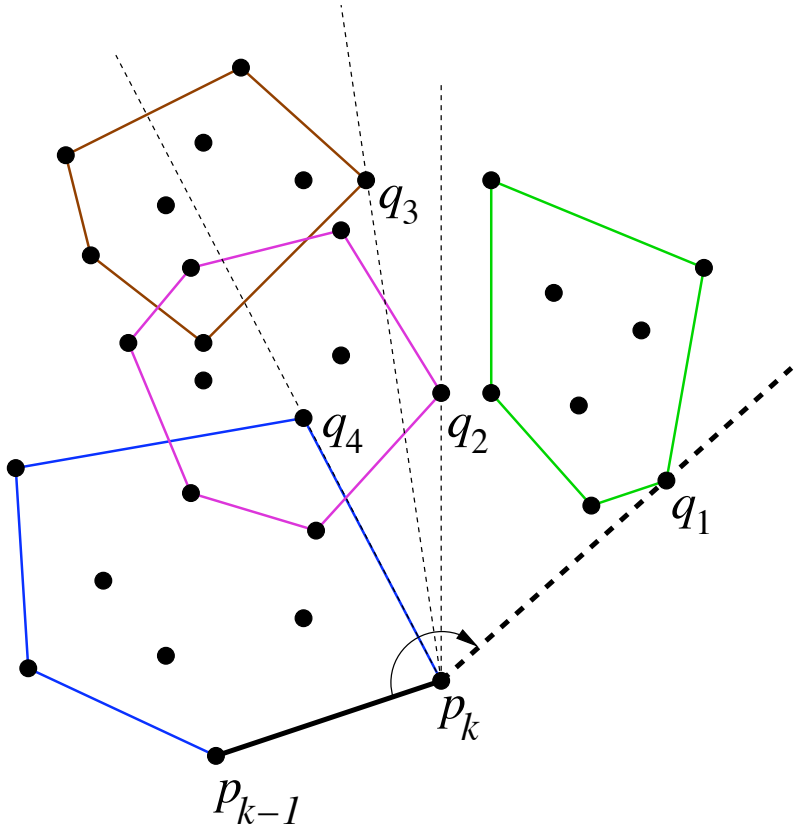
4. **Key idea of Chan is as follows.**

   (a) **Partition the n points into groups of size m; number of groups is $r = \lceil n/m \rceil$.**
   (b) **Compute hull of each group with Graham's scan.**
   (c) **Next, run Jarvis on the groups.**

Chan's Algorithm

1. **The algorithm requires knowledge of CH size h.**
2. **Use m as proxy for h. For the moment, assume we know $m = h$.**
3. **Partition P into r groups of m each.**
4. **Compute $\text{Hull}(P_i)$ using Graham scan, $i = 1, 2, \ldots, r$.**
5. **$p_0 = (-\infty, 0)$; p_1 bottom point of P.**
6. **For $k = 1$ to m do**
   - **Find $q_i \in P_i$ that maximizes the angle $\angle p_{k-1} p_k q_i$.**
   - **Let p_{k+1} be the point among q_i that maximizes the angle $\angle p_{k-1} p_k q$.**
   - **If $p_{k+1} = p_1$ then return $\langle p_1, \ldots, p_k \rangle$.**
7. **Return "m was too small, try again."**

Illustration

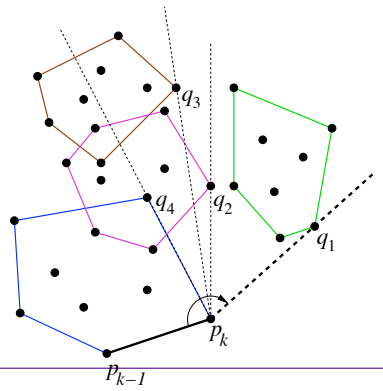


Time Complexity

- **Graham Scan:** $O(rm \log m) = O(n \log m)$.
- **Finding tangent from a point to a convex hull in** $O(\log n)$ **time.**
- **Cost of Jarvis on** r **convex hulls: Each step takes** $O(r \log m)$ **time; total** $O(hr \log m) = ((hn/m) \log m)$ **time.**
- **Thus, total complexity**

$$O\left(\left(n + \frac{hn}{m}\right) \log m\right)$$

- **If** $m = h$**, this gives** $O(n \log h)$ **bound.**
- **Problem: We don't know** h.



Finishing Chan

Hull(P)

- **for** $t = 1, 2, \ldots$ **do**

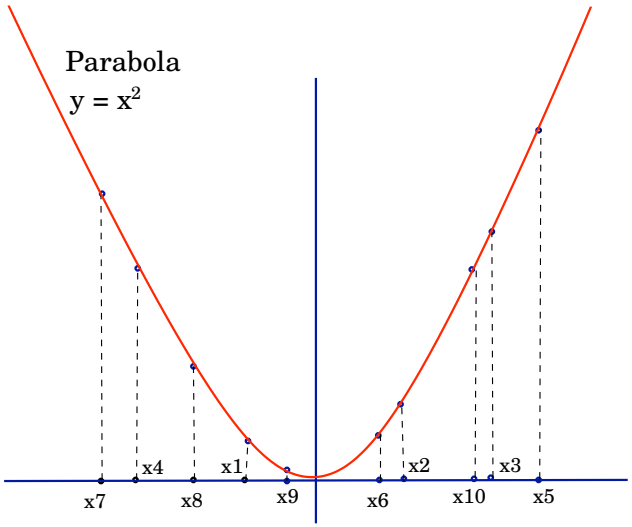
  1. **Let** $m = \min(2^{2^t}, n)$.
  2. **Run Chan with** m**, output to** L.
  3. **If** $L \neq$ **"try again" then return** L.

1. **Iteration** t **takes time** $O(n \log 2^{2^t}) = O(n2^t)$.
2. **Max value of** $t = \log\log h$**, since we succeed as soon as** $2^{2^t} > h$.
3. **Running time (ignoring constant factors)**

$$\sum_{t=1}^{\lg\lg h} n2^t = n \sum_{t=1}^{\lg\lg h} 2^t \leq n2^{1+\lg\lg h} = 2n \lg h$$

4. **2D convex hull computed in** $O(n \log h)$ **time.**

Lower Bounds



- **Reduce sorting to convex hull.**
- **List of numbers to sort $\{x_1, x_2, \ldots, x_n\}$.**
- **Create point $p_i = (x_i, x_i^2)$, for each i.**
- **Convex hull of $\{p_1, p_2, \ldots, p_n\}$ has points in sorted x-order. $\Rightarrow$ CH of n points must take $\Omega(n \log n)$ in worst-case time.**
- **More refined lower bound is $\Omega(n \log h)$. LB holds even for identifying the CH vertices.**

Lower bounds revisited

- recall limitations of the reduction from sorting argument
    - need a stronger model than pairwise comparisons

Lower bounds revisited

- recall limitations of the reduction from sorting argument
  - need a stronger model than pairwise comparisons
    - algebraic decision trees

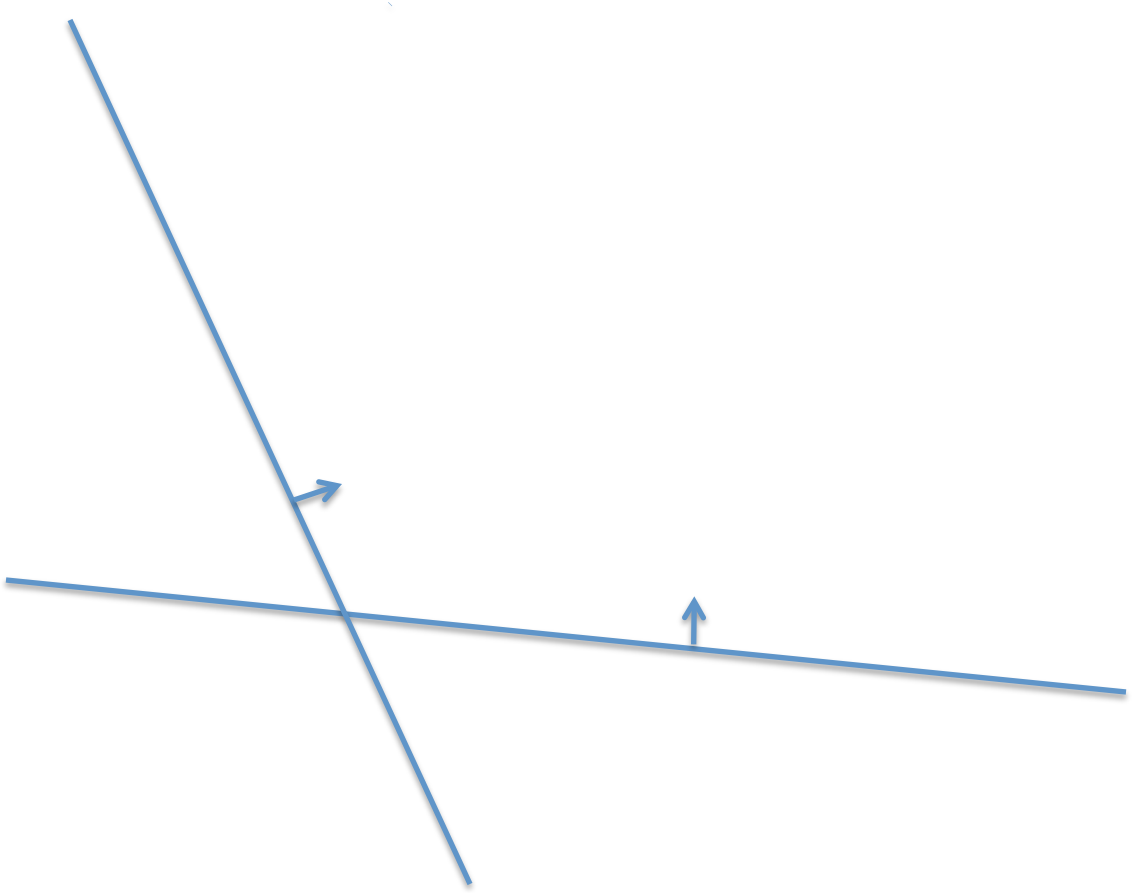
Lower bounds revisited

- recall limitations of the reduction from sorting argument
  - need a stronger model than pairwise comparisons
    - algebraic decision trees
  - applies to strong version of CH problem (requires ordered output)
  - does not explain output-size sensitivity (dependence on h)

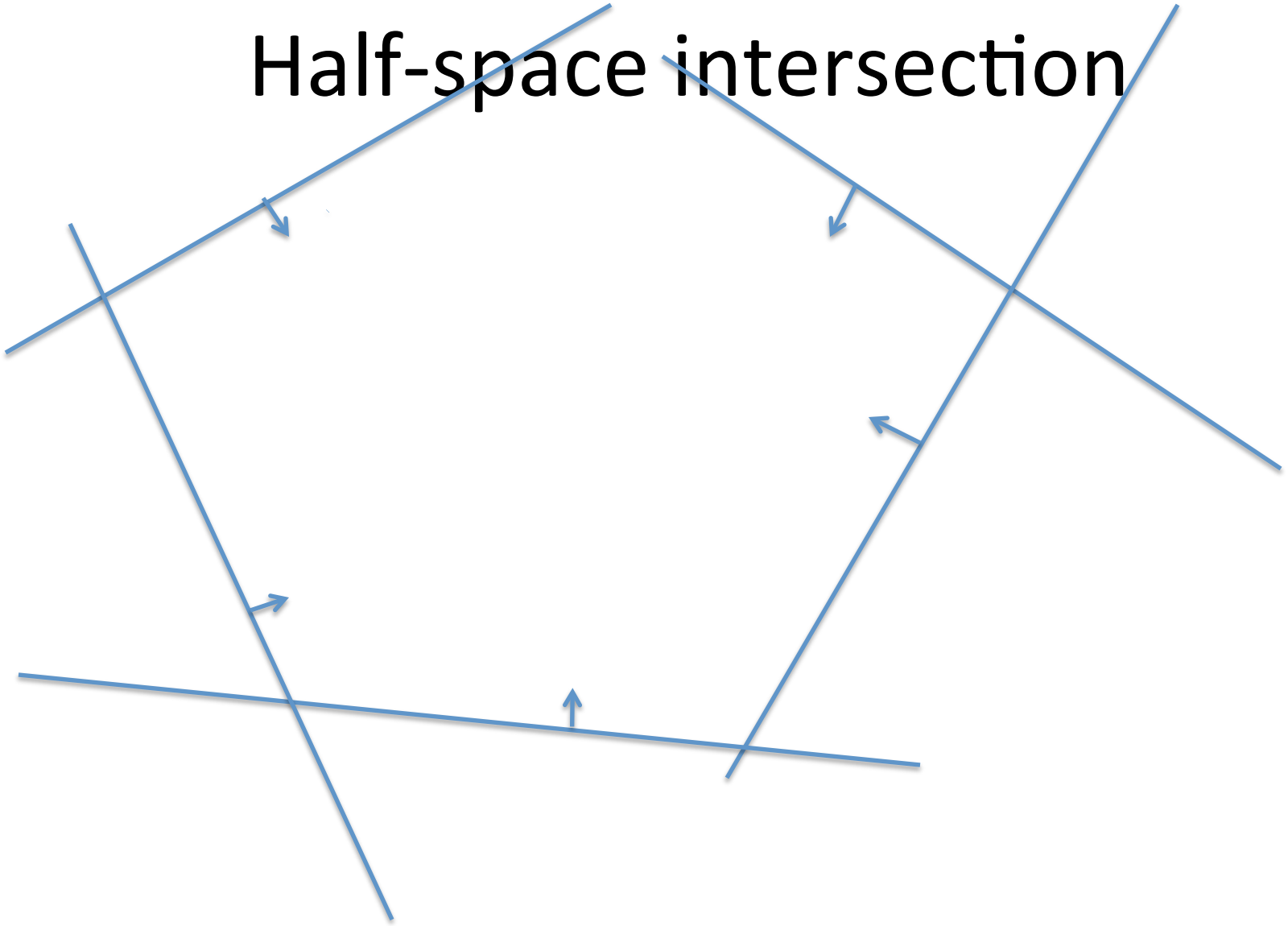
Lower bounds revisited

- recall limitations of the reduction from sorting argument
  - need a stronger model than pairwise comparisons
    - algebraic decision trees
  - applies to strong version of CH problem (requires ordered output)
  - does not explain output-size sensitivity (dependence on h)
    - formulate decision problems as point-classification problems

Half-space intersection

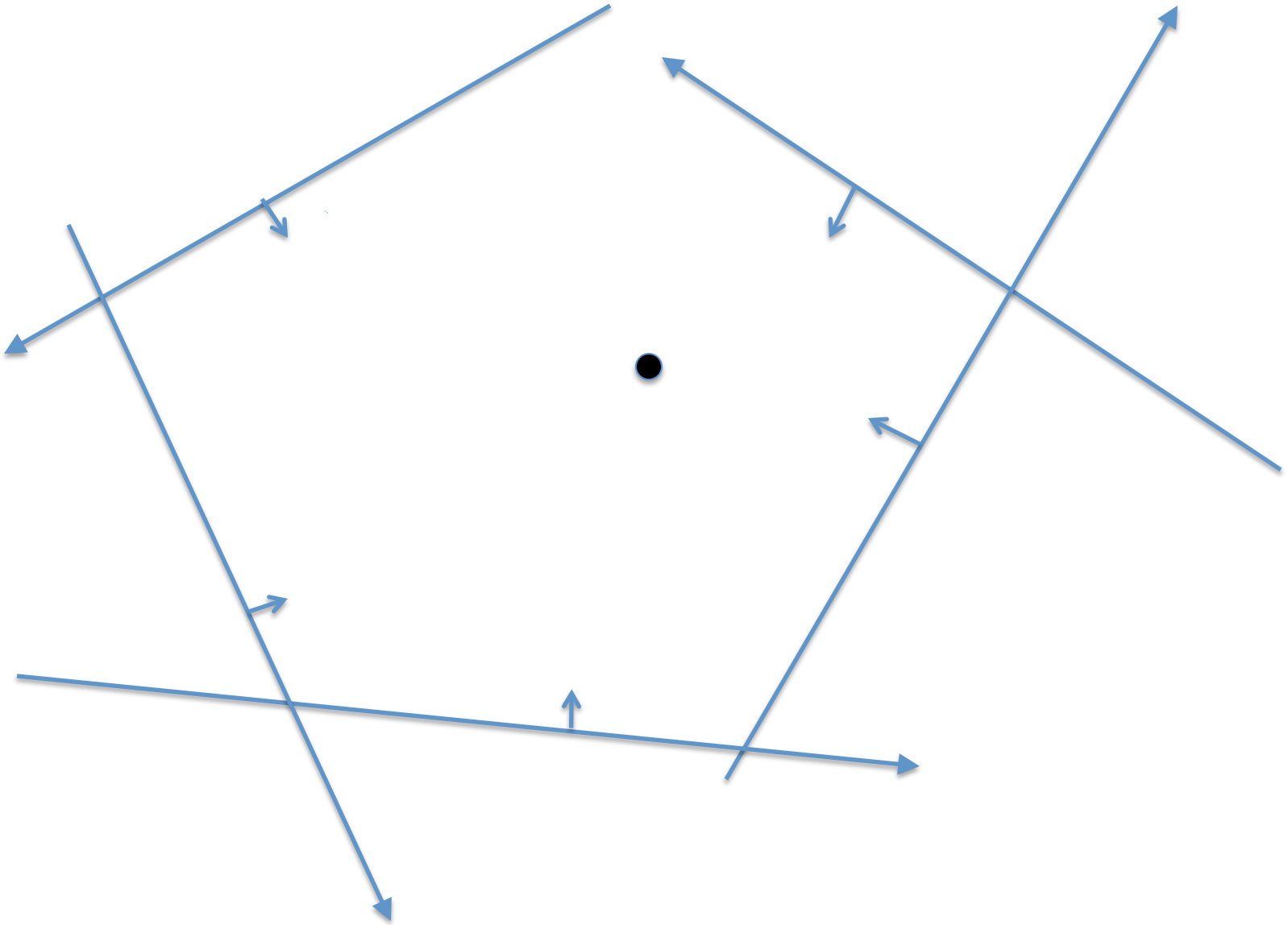


Half-space intersection



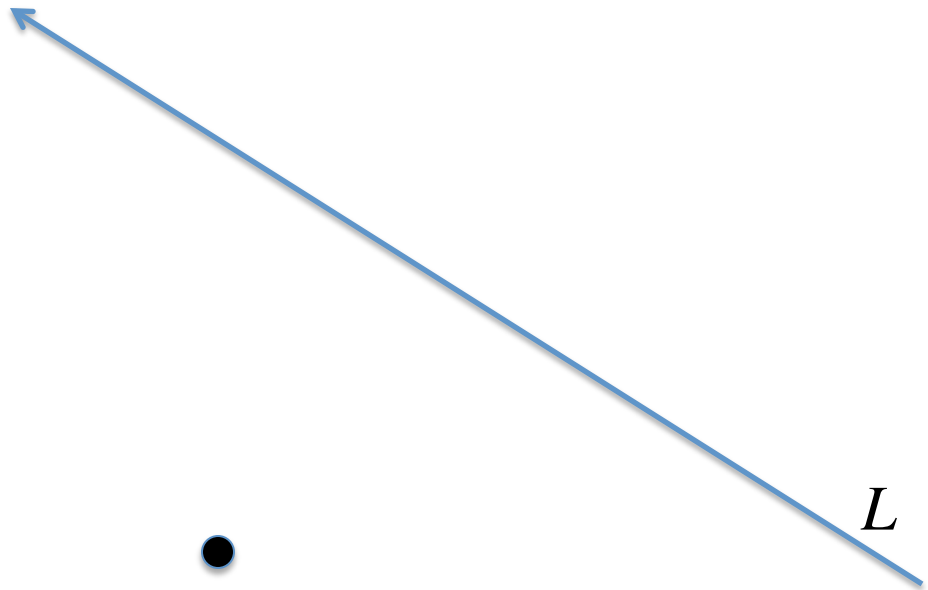
Half-space intersection

- suppose we have a *witness* to the non-emptiness of the intersection—may as well be the origin
- such half-spaces are defined by *oriented* lines (directed so that origin lies to the left)

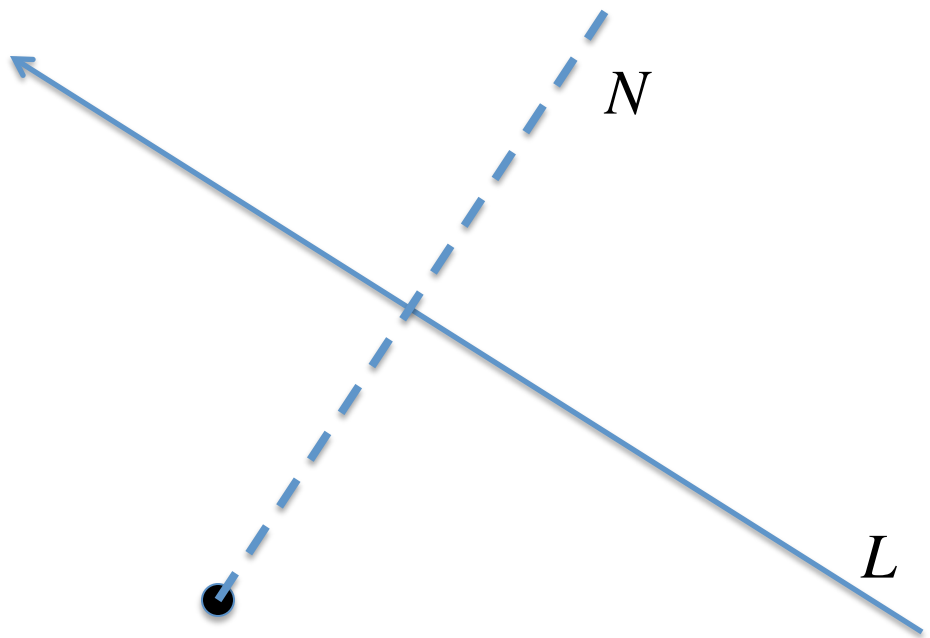


Polarity transform

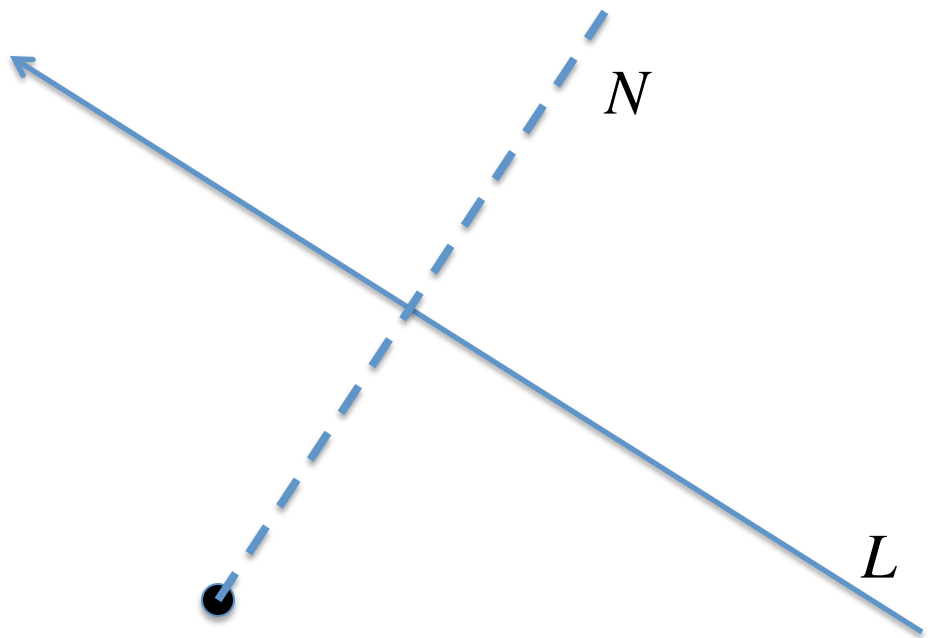
- an arbitrary line *L* that *avoids* the origin has the an equation of the form: $ax + by - 1 = 0$
- view as directed line where points to *left* (respectively *right*) make $ax + by - 1$ *negative* (respectively, *positive*)



$L: \; ax+by-1 = 0$

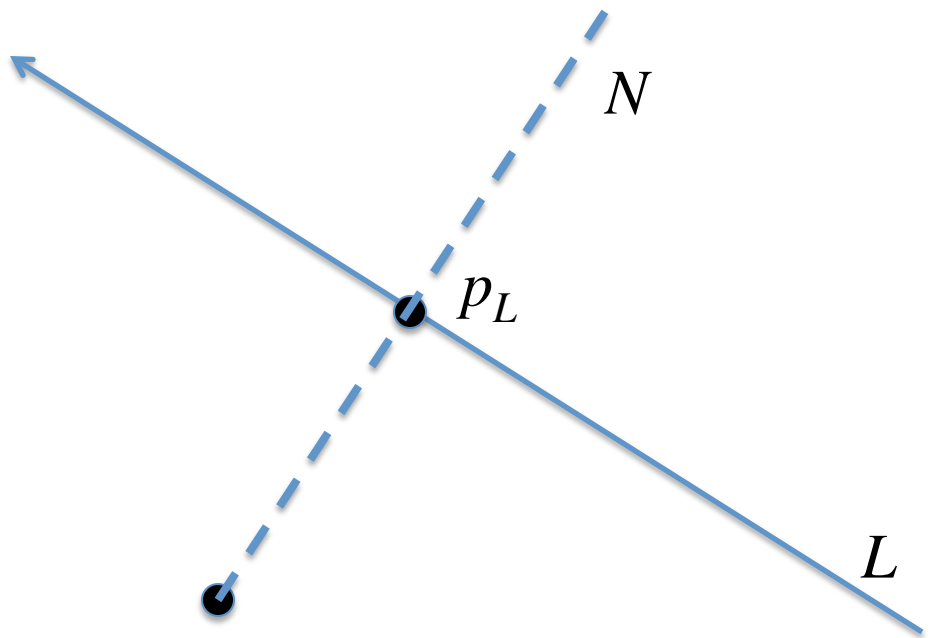


$L:\ ax+by-1=0$



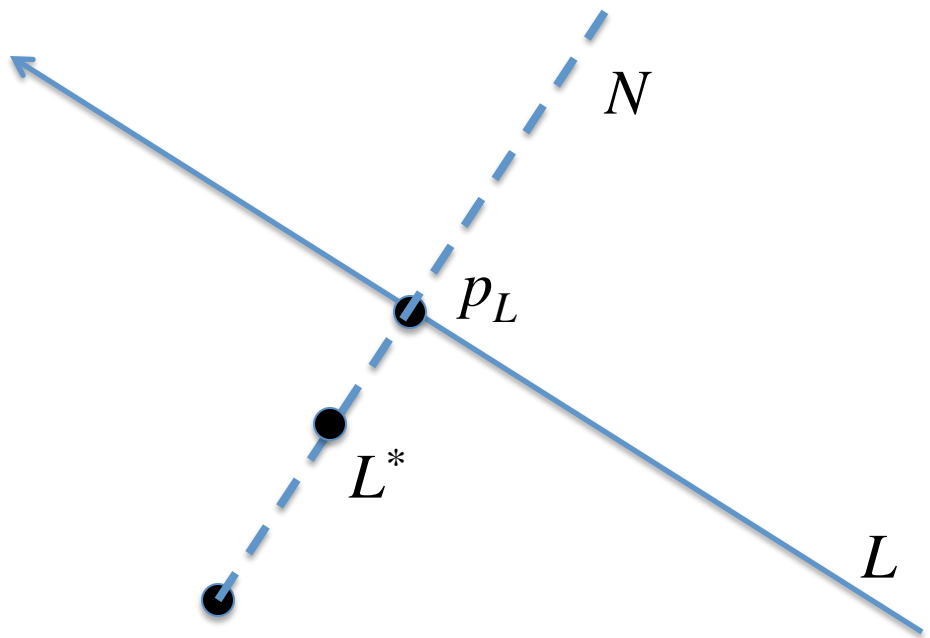
$L: \; ax+by-1 = 0$

$N: \; bx-ay = 0$



L: $ax+by-1 = 0$

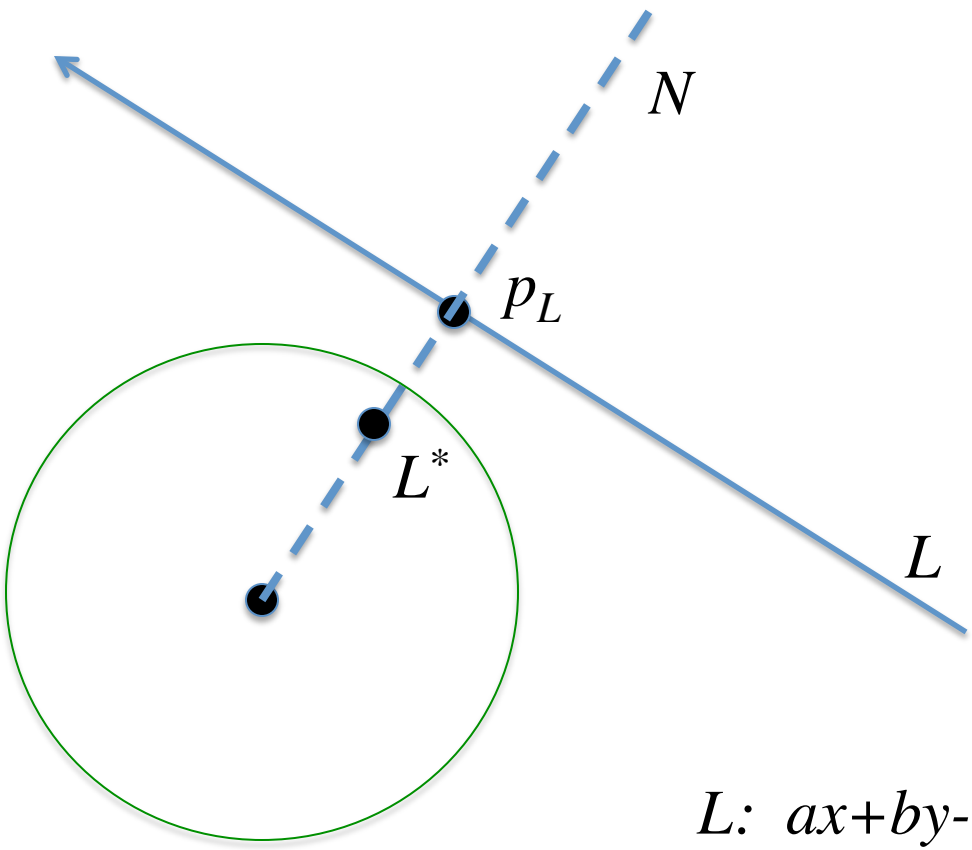
N: $bx-ay = 0$



L: $ax+by-1 = 0$

N: $bx-ay = 0$

L^*: (a,b)

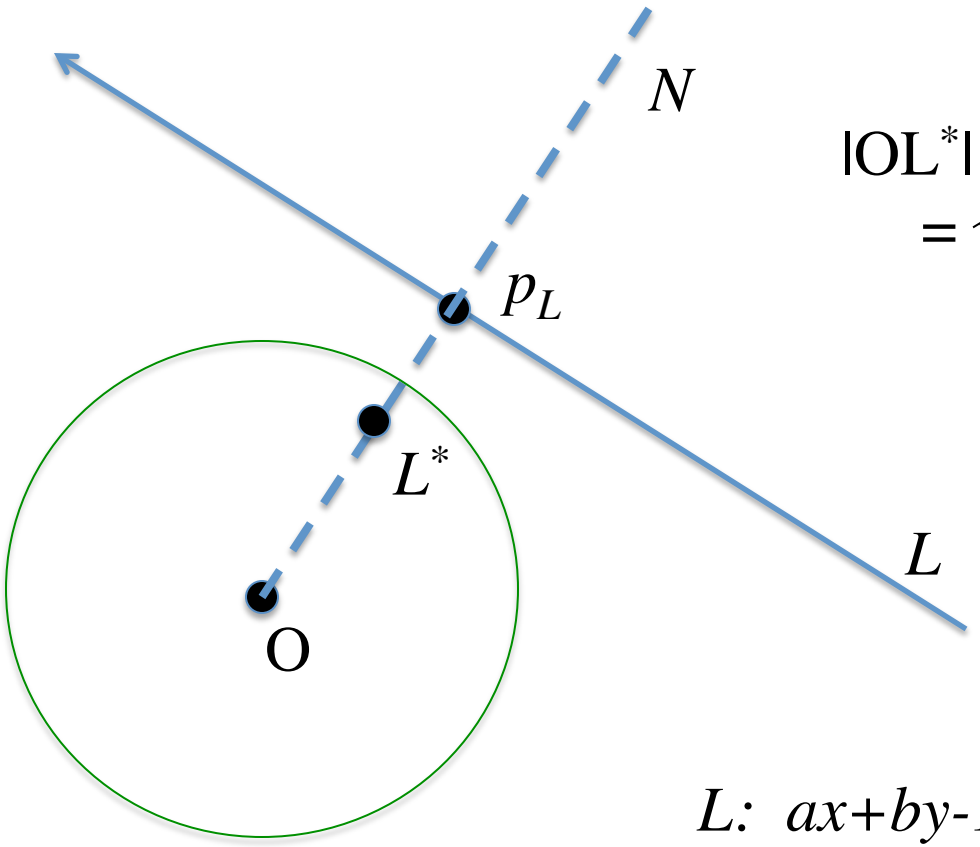


unit circle

L: $ax+by-1 = 0$

N: $bx-ay = 0$

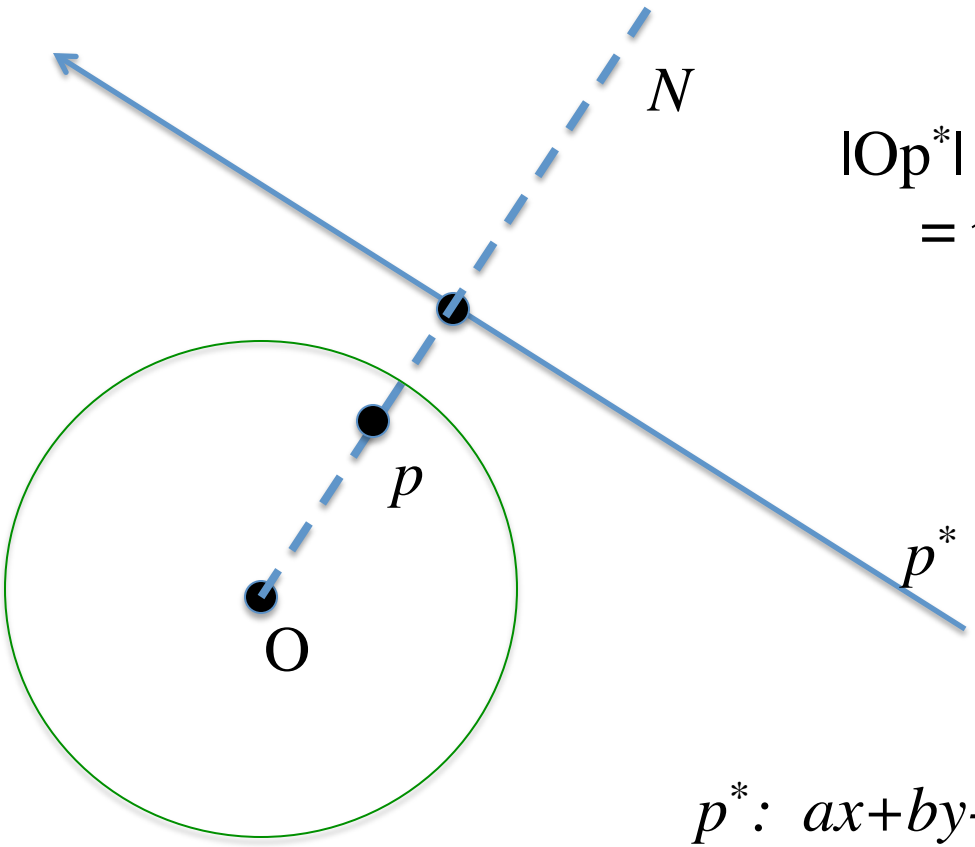
L^*: (a,b)



$|OL^*| = 1 / |OL|$
$= \sqrt{(a^2 + b^2)}$

unit circle

L: ax+by-1 = 0
N: bx-ay = 0
L^*: (a,b)



$|Op^*| = 1 / |Op|$
$= \sqrt{(a^2 + b^2)}$

unit circle

p^*: $ax+by-1 = 0$
N: $bx-ay = 0$
p: (a,b)

Properties

- [self inverse] $(p^*)^* = p$

Properties

- [self inverse] $(p^*)^* = p$ (and $(L^*)^* = L$)

Properties

- [self inverse] $(p^*)^* = p$
- [incidence preserving] if p belongs to L, then L^* belongs to p^*

Properties

- [self inverse] $(p^*)^* = p$
- [incidence preserving] if p belongs to L, then L^* belongs to p^*
- [sidedness reversing] if p lies to the left (right) of L, then L^* lies to the left (right) of p^*

Properties

- [self inverse] $(p^*)^* = p$
- [incidence preserving] if p belongs to L, then L^* belongs to p^*
- [sidedness reversing] if p lies to the left (right) of L, then L^* lies to the left (right) of p^*
- the line joining points p_1 and p_2 is the dual of the point formed by the intersection of the lines p_1^* and p_2^*

Equivalent problems

- [half-space intersection] finding all points that lie to the left of the (primal) lines defining the half-spaces
- [convex hull] finding all lines that lie to the right of all of the (dual) points
- in both cases a succinct description is a polygon (polytope); the boundary order is preserved under duality.

Loose ends from earlier discussions

- half-space intersection problem
  - how do we find a point in the common intersection (if it exists) in general?

Loose ends from earlier discussions

- half-space intersection problem
  - how do we find a point in the common intersection (if it exists) in general?
    - LP feasibility

Loose ends from earlier discussions

- half-space intersection problem
  - how do we find a point in the common intersection (if it exists) in general?
    - LP feasibility
- the marriage-before-conquest convex hull algorithm
  - need to find an (upper) bridge between opposite partitions. How do we do this efficiently?

Loose ends from earlier discussions

- half-space intersection problem
  - how do we find a point in the common intersection (if it exists) in general?
    - LP feasibility
- the marriage-before-conquest convex hull algorithm
  - need to find an (upper) bridge between opposite partitions. How do we do this efficiently?
    - a 2 variable (2-d) linear programming problem

Low-dimensional linear programming

- [2-d] a deterministic linear time algorithm
  - view as bridge-finding; candidate elimination
  - general LP formulation (Megiddo)
  - linear-time algorithms in higher dimensions

Low-dimensional linear programming (cont.)

- an incremental approach
  - in 1-d
  - in 2-d
    - (worst-case) analysis of deterministic implementation
    - (expected-case) analysis of randomized implementation

Low-dimensional linear programming (cont.)

- extensions to higher dimensions
  - Meggido's approach
  - randomized incremental approach

Applications

- 1-center problem…
  - “pinned” subproblems
    - uniqueness of solutions
    - reductions
- other LP-type problems