



# 計算幾何学

## Computational Geometry



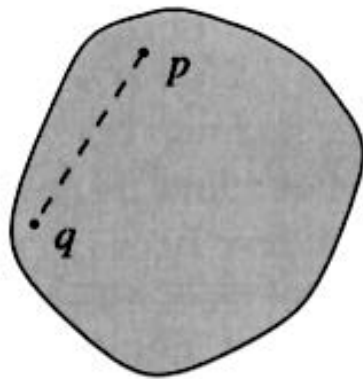
---

### 第三章 凸包

### Convex Hull

# 凸集合 (Convex Set)

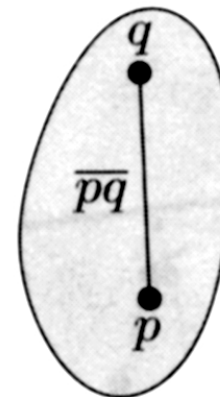
- 平面上に与えられる  $n$  個の点の集合  $S_n$  に対して、任意2点  $p, q$  の線分が  $S_n$  に完全に含まれる。



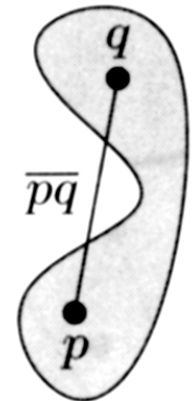
A convex set



A nonconvex set



凸



凸でない

# 凸包(Convex Hull)の定義と表現

- 平面上に与えられる $n$ 個の点 $p_i(i=1,2,\dots,n)$ の集合 $S_n$ に対し、 $S_n$ を含む最小の凸多角形 $\rightarrow S_n$ の凸包 $CH(S_n)$

- 実例

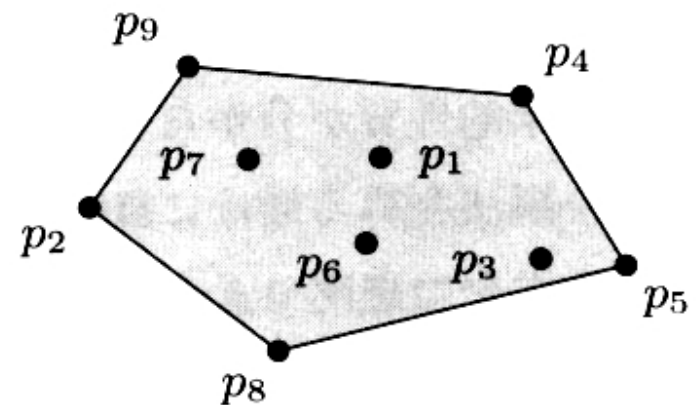
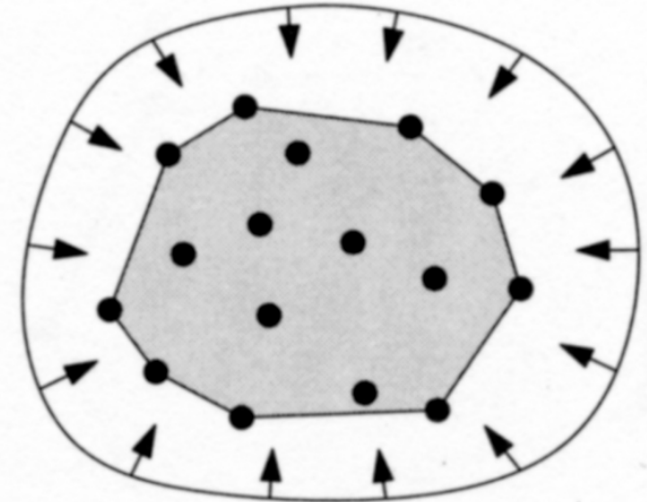
- 入力としての点集合

$$S_n = p_1, p_2, \dots, p_9$$

- 出力としての凸包

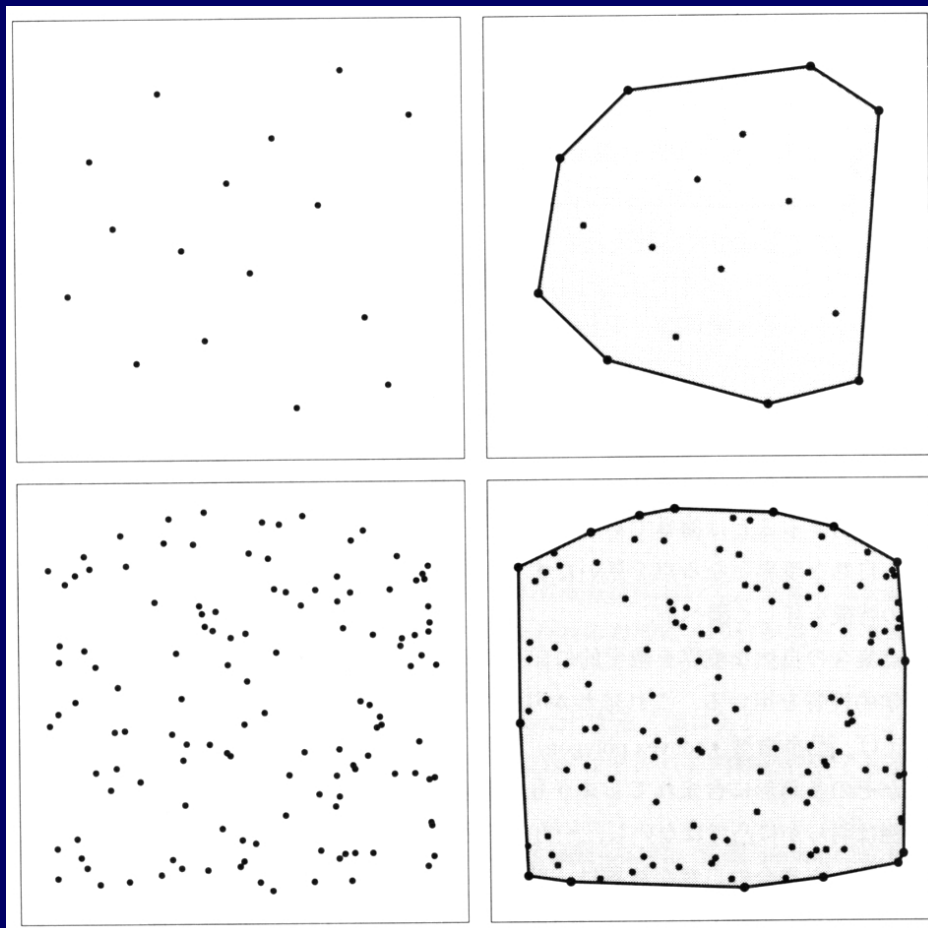
$$CH(S_n) = p_4, p_5, p_8, p_2, p_9$$

輪ゴムを掛けるようなもの



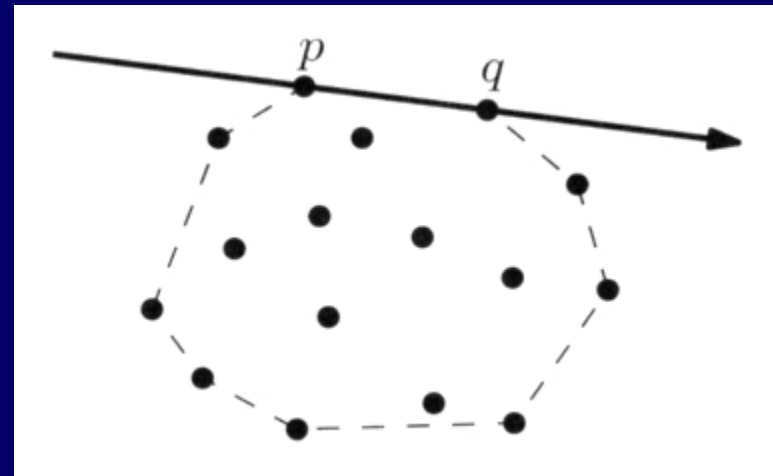
# 計算方法

- 直接法
- 包装法
- Graham走査法
- 逐次添加法
- 分割統治法
- Quick法
- 内点消去法  
(前処理)



# 直接法 (Direct method)

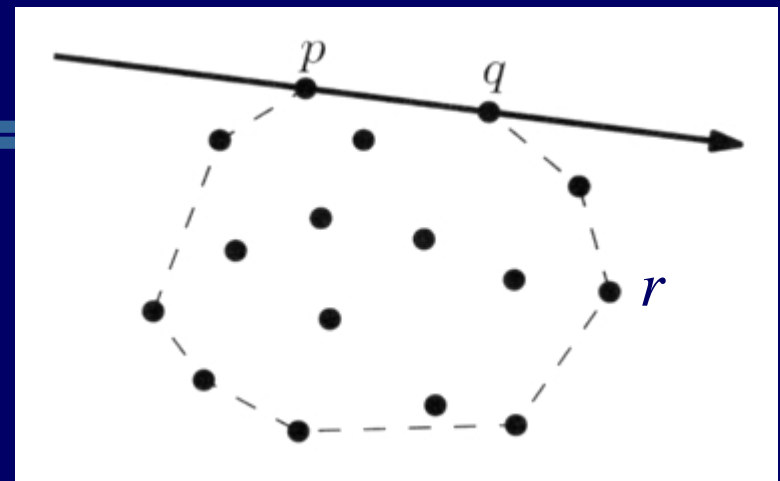
- 点の集合  $S_n$  から任意2点  $p, q$  を取り出し、その線分が凸包の多角形の辺であるかどうかを判定する
- 他の  $n-2$  個の点が全て  $p, q$  を通る直線の片側 (右又は左) にある時 → 線分  $pq$  が凸包の辺
- $n(n-1)/2$  組点対



# アルゴリズム

DirectConvexHull( $S_n$ )

- 入力: 平面上の点集合  $S_n$
  - 出力: **時計回り**の順に  $CH(S_n)$  の頂点を含むリスト  $\mathcal{L}$
1. 辺集合初期化  $E \leftarrow \emptyset$
  2. for 全ての点対  $p, q$  in  $S_n$ 、但し、 $p \neq q$
  3.     do  $valid \leftarrow \text{true}$
  4.         for  $p, q$  以外全ての点  $r$
  5.             do if  $r$  が  $\overrightarrow{pq}$  の **左側** にある
  6.                 then  $valid \leftarrow \text{false}$
  7.             if  $valid$  then  $\overrightarrow{pq}$  を  $E$  に追加
  8.  $E$  を時計回り順にして  $\mathcal{L}$  を構成する



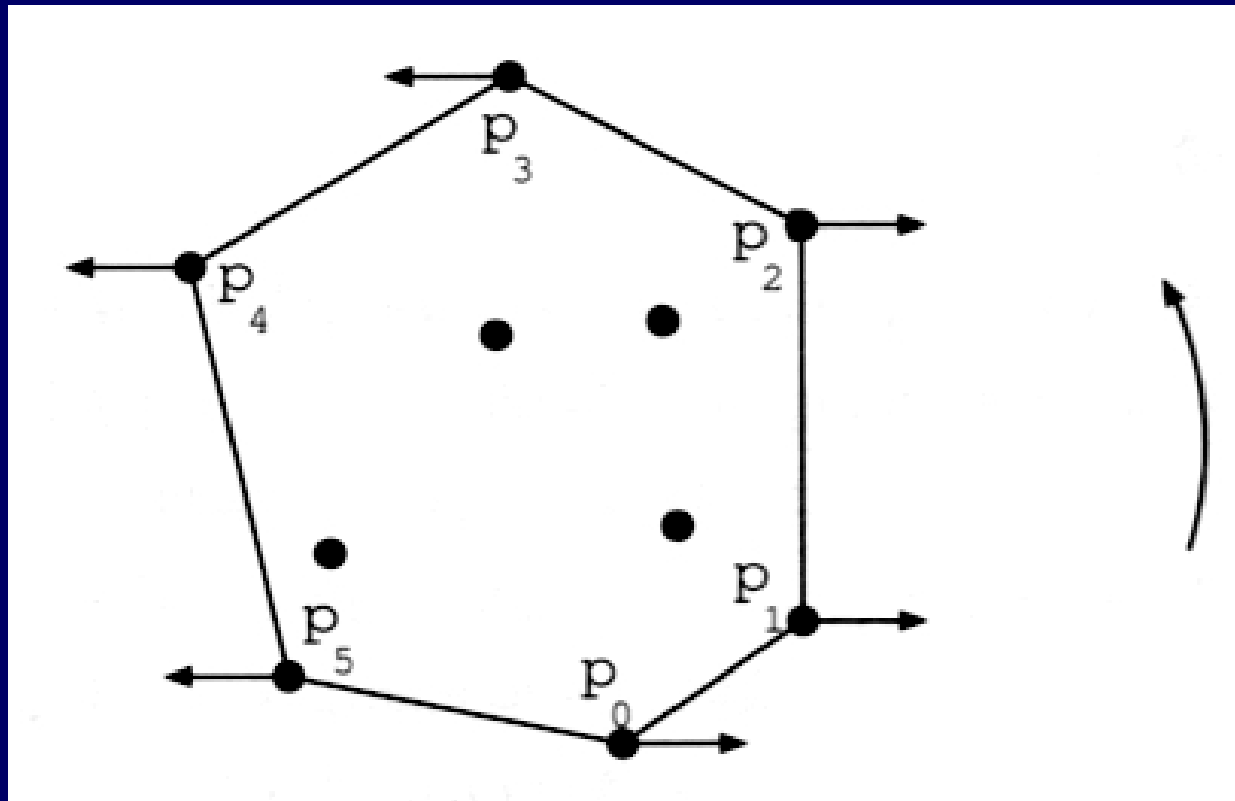
**逆時計回りの場合  
→右側**

# 計算量

- 点集合のデータ数 =  $n$
- 辺の組み合わせ数 =  $n(n-1)/2 = O(n^2)$
- 一つの辺に対して、 $n-2$ 個の三角形処理  
=  $O(n)$
- 総計算量 =  $O(n^3)$

# 包装法 (Wrapping method)

- 凸包の既存の頂点から出発して、順次に次の頂点を探索していく

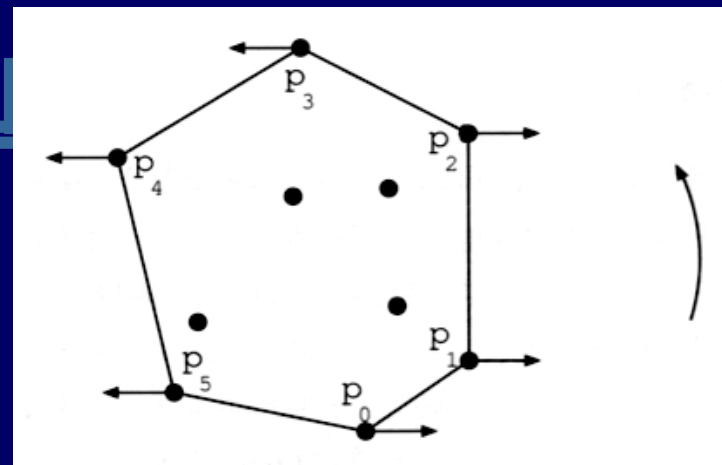




# アルゴリズム

WrappingConvexHull( $S_n$ )

- 入力: 平面上の点集合  $S_n$
  - 出力:  $CH(S_n)$  の頂点を含むリスト  $\mathcal{L}$
1. 最小の  $y$  座標を持つ点  $p_0$  を見つける
  2.  $\mathcal{L}$  に  $p_0$  を追加する
  3. for 全ての点  $q$  in  $S_n$ 、但し、 $q \notin \mathcal{L}$
  4.      $\mathcal{L}(\text{top})$  を原点として  $q$  との偏角を計算する
  5.     最小偏角を持つ点  $q_{\min}$  を探索する
  6.      $q_{\min}$  を  $\mathcal{L}$  に追加する
  7. if  $q_{\min} \neq p_0$  then goto step 3 else return  $\mathcal{L}$



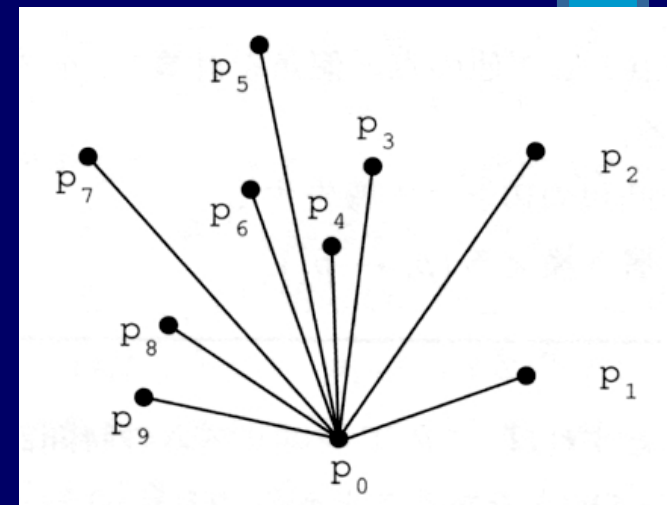


# 計算量

- 点集合のデータ数 =  $n$ 、頂点数 =  $h$
- 最初の頂点  $p_0$  を探索する計算量 =  $O(n)$
- 一つの頂点と残り未処理の点の最小偏角を探索する計算量 =  $O(n)$
- 総計算量 =  $O(n) + O(hn) = O(hn)$
- 最悪の場合  $\rightarrow h=n \rightarrow O(n^2)$

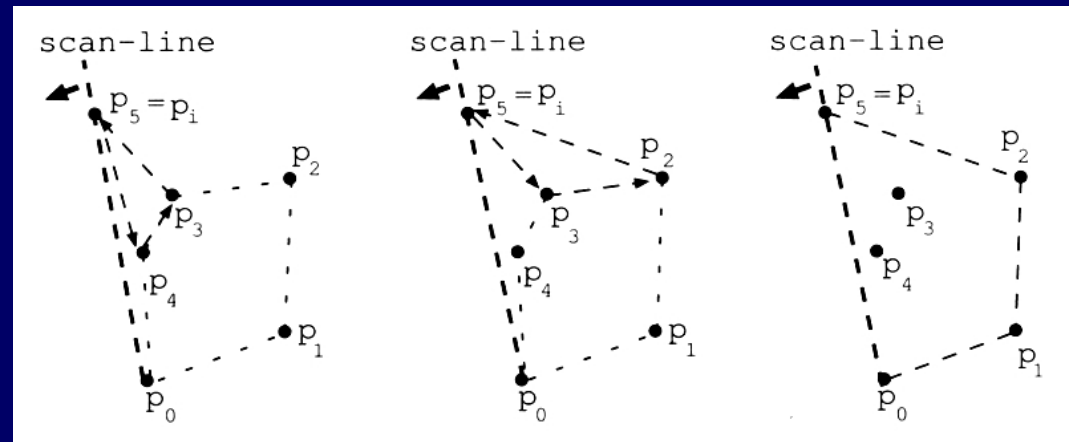
# Graham走査法(Graham scan)

- 最小の $y$ 座標を持つ点 $p_0$ を見つける
- $p_0$ と他の点で構成された線分の偏角を求める
- **偏角**の昇順で点集を並び替える
- $p_0$ と $p_1$ が凸包の頂点→スタック $S$ に挿入
- $p_2$ から各点を順次に調べる
- $p_i \neq$ 頂点→除去する
- How to...?



# Graham走査法 続き

- 点 $p_i$ を加える時、 $S[top-1]$ ,  $S[top]$ ,  $p_i$ から構成される三角形 $\neq$ 逆時計回り
- $S[top]$ を $S$ から取り出し、除去する
- 上記の三角形 $=$ 逆時計回りになるまで繰り返す
- $p_i$ を $S$ に入れる
- 例: $i=5$ の時





# アルゴリズム

## GrahamConvexHull( $S_n$ )

- 入力: 平面上の点集合  $S_n$
  - 出力:  $CH(S_n)$  の頂点を含むスタック  $S$
1. 最小の  $y$  座標を持つ点  $p_0$  を見つける
  2.  $p_0$  と他の点で構成された線分の偏角の昇順に並び替え、 $p_1, p_2, \dots, p_{n-1}$
  3.  $p_0, p_1 \rightarrow S$
  4. for  $i=2$  to  $n-1$  do
  5.     while  $\Delta(S[top-1], S[top], p_i) = \text{時計回り}$  do
  6.         pop( $S[top]$ )
  7.     push( $p_i$ )

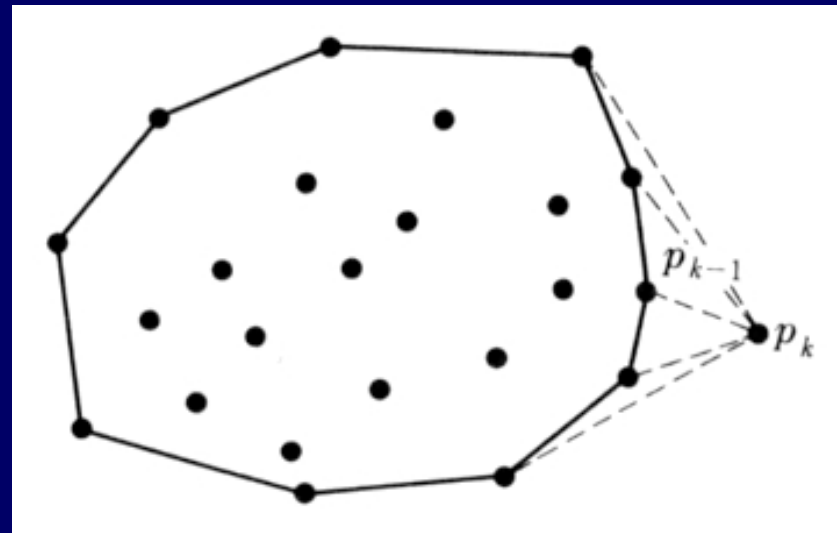


# 計算量

- 点集合のデータ数 =  $n$
- 最初の頂点  $p_0$  を探索する計算量 =  $O(n)$
- 偏角の並び替え計算量 =  $O(n \log n)$
- スタックへの点の追加と削除 =  $O(n)$
- 総計算量 =  $O(n) + O(n \log n) + O(n) = O(n \log n)$

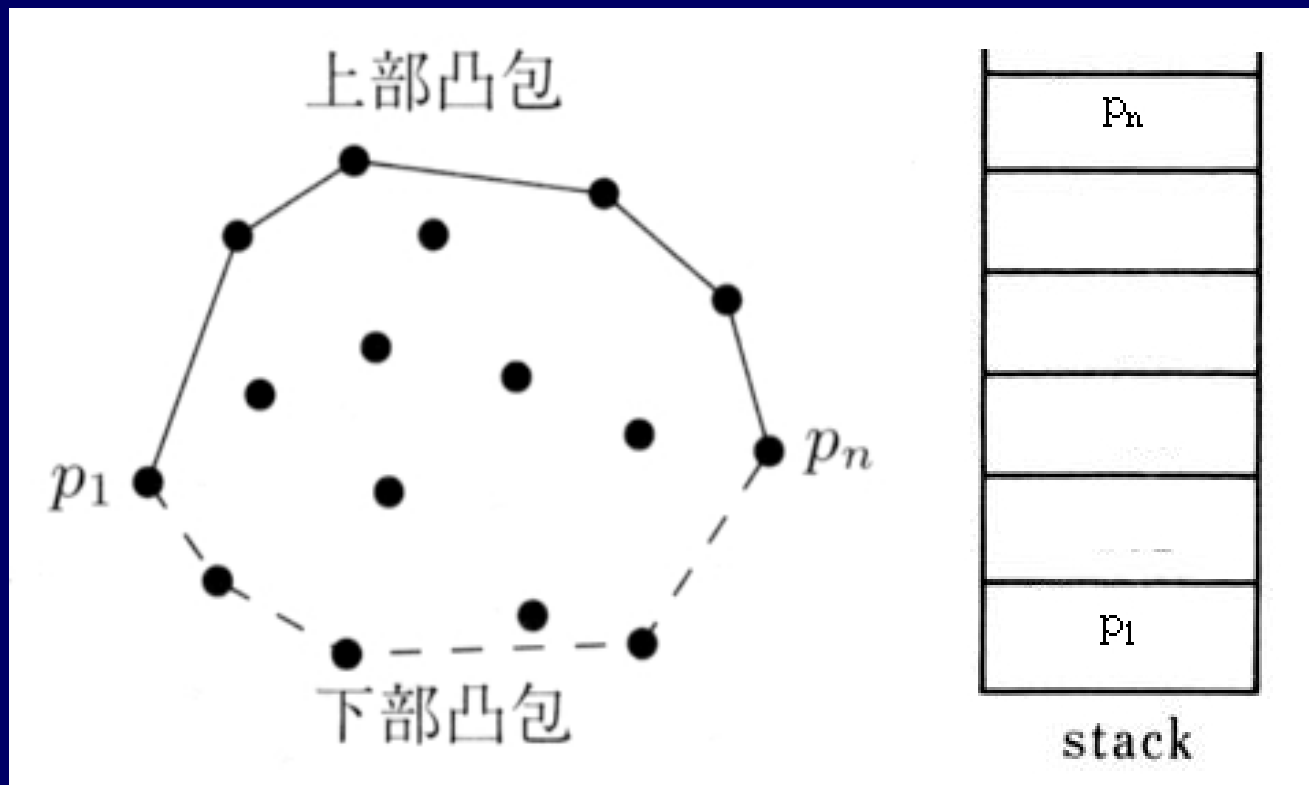
# 逐次添加法 (Incremental method)

- 点集を $x$ 座標で昇順に並び替える
- 最初の3点で凸包を構成する
- 残りの点を1点ずつ加えながら、凸包を更新していく
- 新しい点 $p_k$ を加える時に、 $p_k$  から凸包への2本接線(?)を求め、凸包の辺とする。接線に挟まれる凸包の境界線を取り除く



# 上・下側境界線

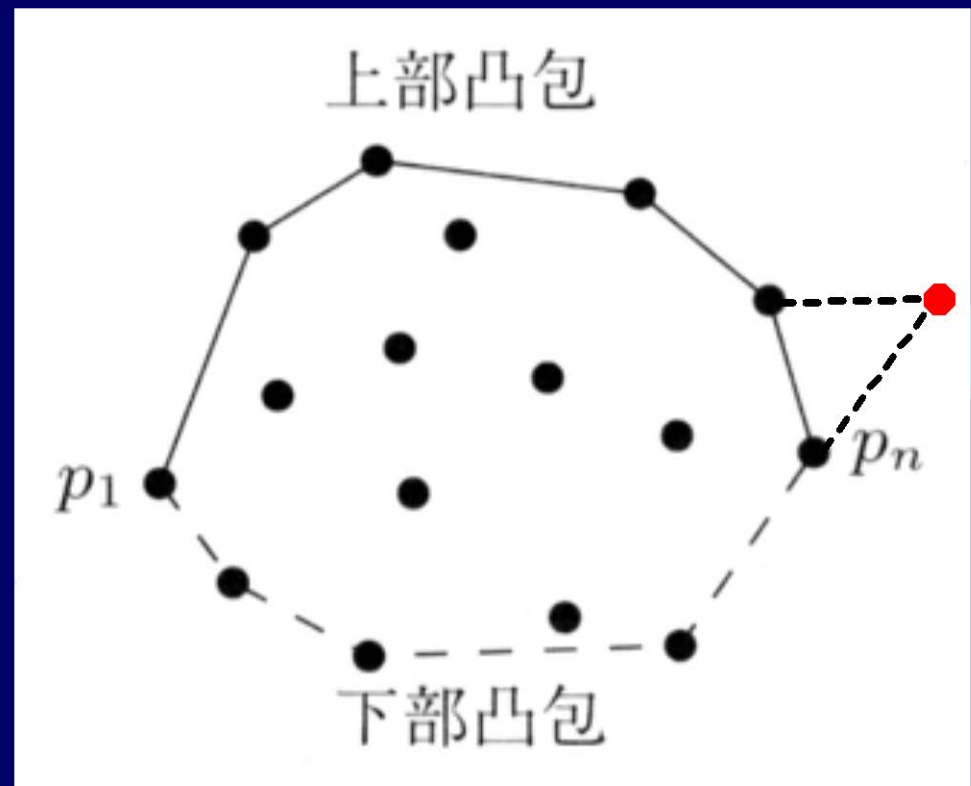
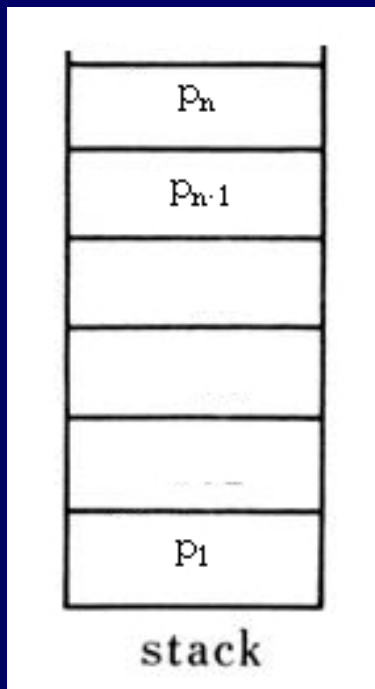
- $p_1$ と $p_n$ 必ず存在する
- $p_1$ と $p_n$ は凸包の境界線を上・下側境界線に分ける





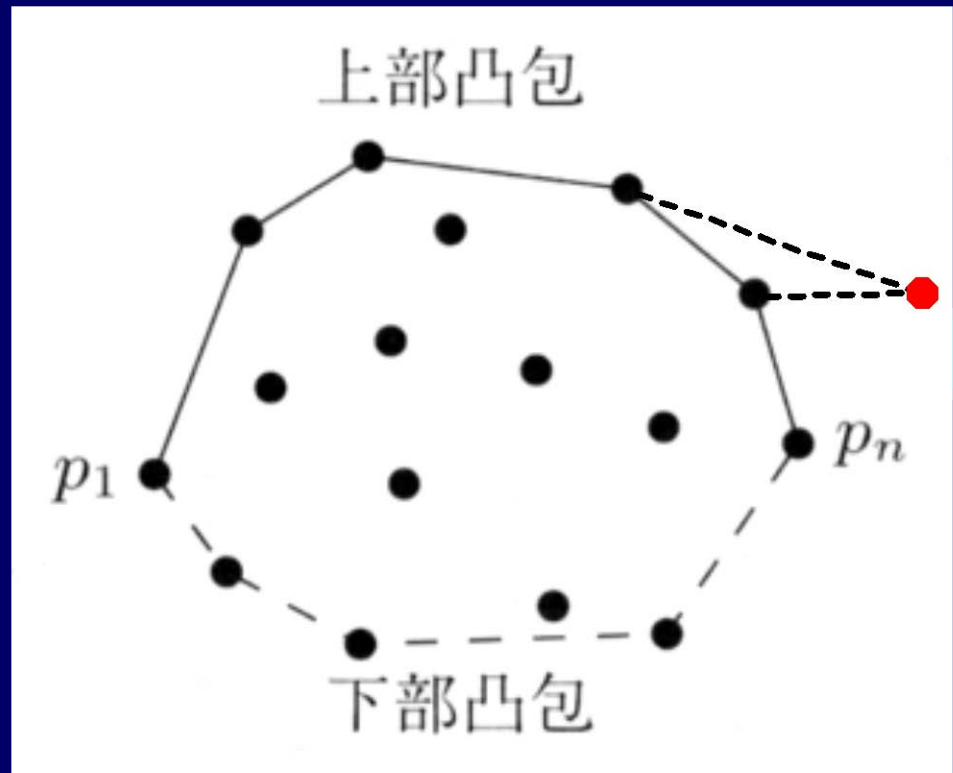
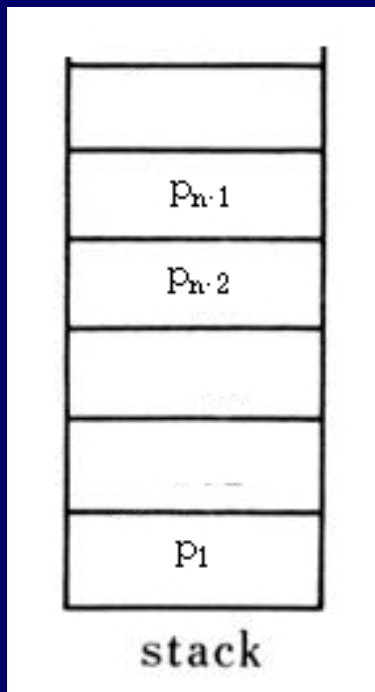
# 接線の求め方

- $\Delta(p_{n+1}, S[top], S[top-1]) = \text{逆時計回りまで}$



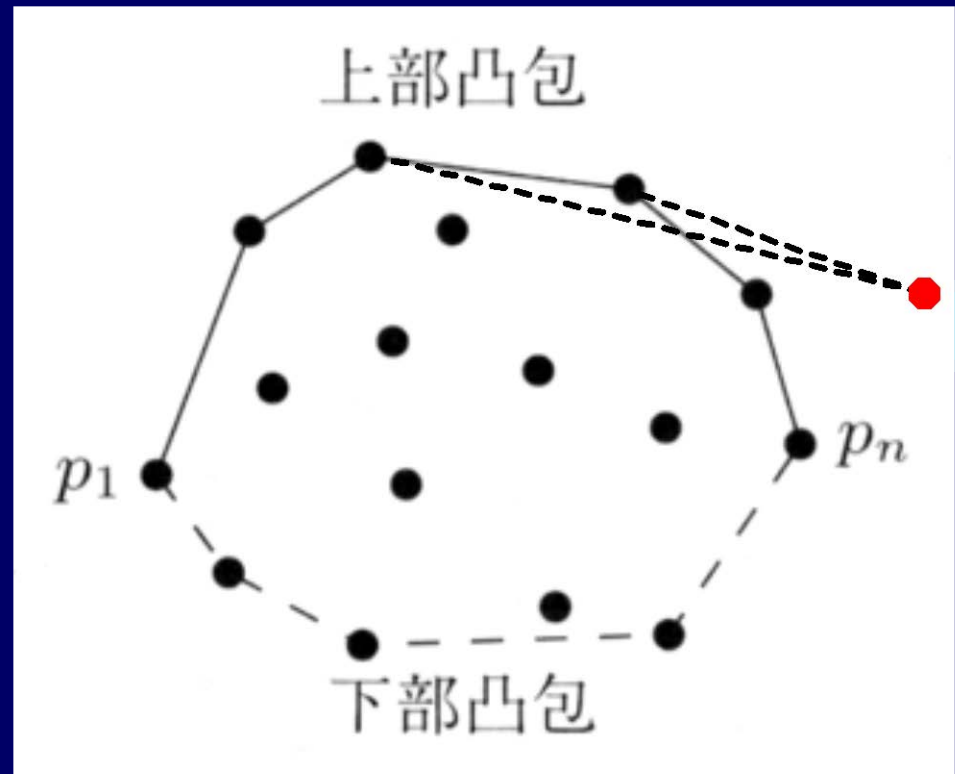
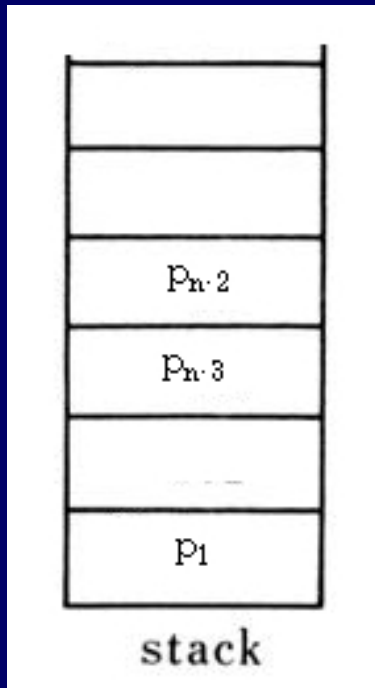
# 接線の求め方

- $\Delta(p_{n+1}, S[top], S[top-1]) = \text{逆時計回りまで}$



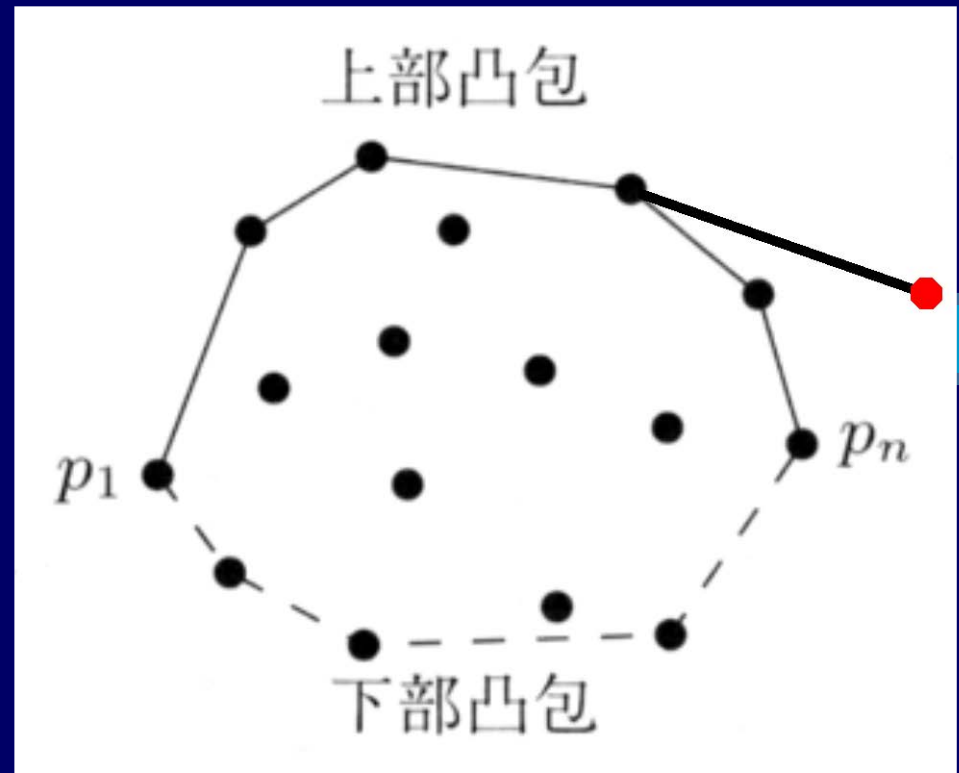
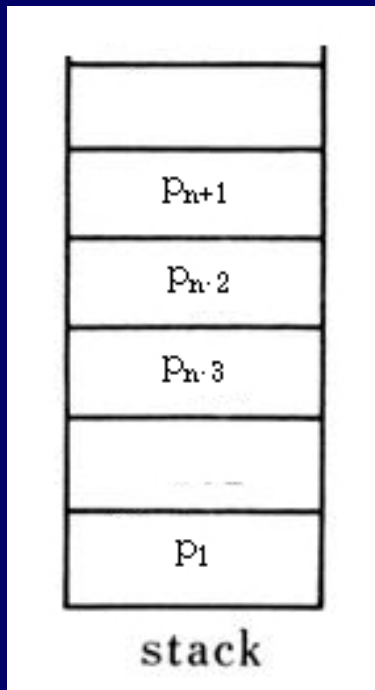
# 接線の求め方

- $\Delta(p_{n+1}, S[top], S[top-1]) = \text{逆時計回りまで}$



# 接線の求め方

- $\Delta(p_{n+1}, S[top], S[top-1]) = \text{逆時計回りまで}$





# アルゴリズム

## IncrementalConvexHull( $S_n$ )

- 入力: 平面上の点集合 $S_n$
- 出力:  $CH(S_n)$ の頂点を含むスタック $S$ 
  1. 点集を $x$ 座標で昇順に並び替える
  2. 最初の3点 $p_0, p_1, p_2$ で凸包を構成する
  3.  $p_0, p_1, p_2 \rightarrow S$
  4. for  $i=3$  to  $n-1$  do
  5.     while  $\Delta(p_i, S[top], S[top-1]) = \text{時計回り}$  do
  6.         pop( $S[top]$ )
  7.         push( $p_i$ )



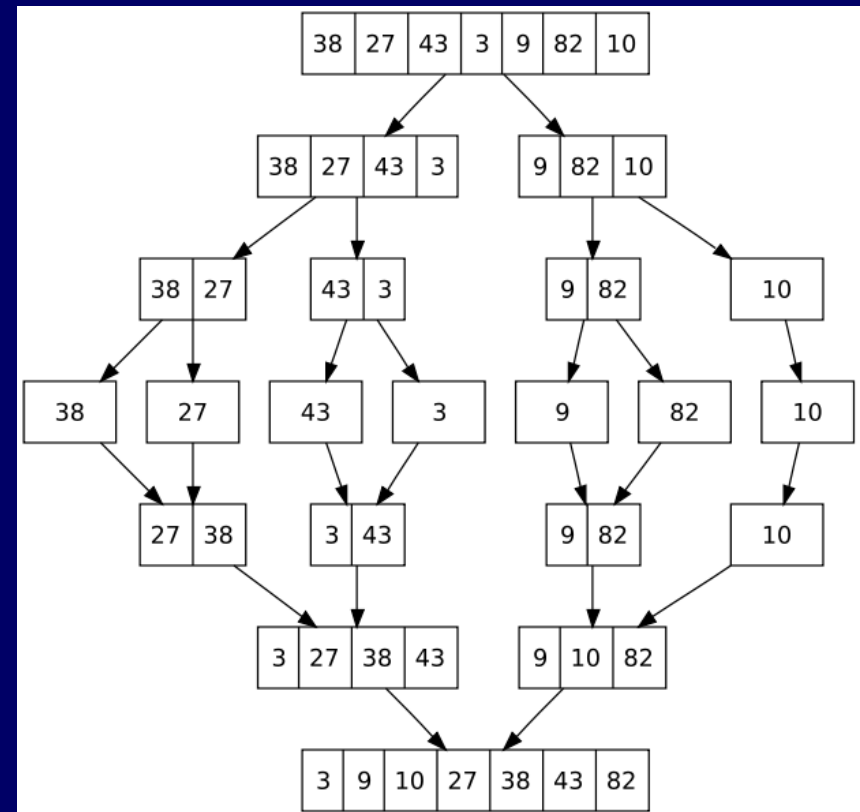
# 計算量

- 点集合のデータ数 =  $n$
- 点集合の並び替え =  $O(n \log n)$
- スタックへの点の追加と削除 =  $O(n)$
- 総計算量 =  $O(n \log n) + O(n) = O(n \log n)$

# 分割統治法 (Divide and conquer)

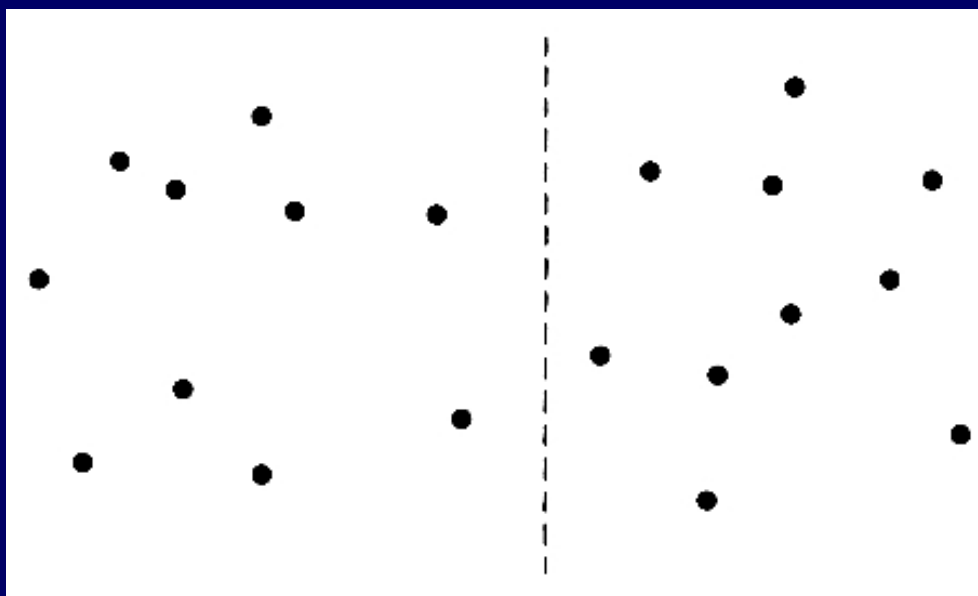
## 基本的な考え方

1. 小さい問題に分割する  
(点集合の2等分)
2. 再帰的に各部分を解く
3. 各解答を統合して、元の問題に対する解答を得る



# 点集合の2等分

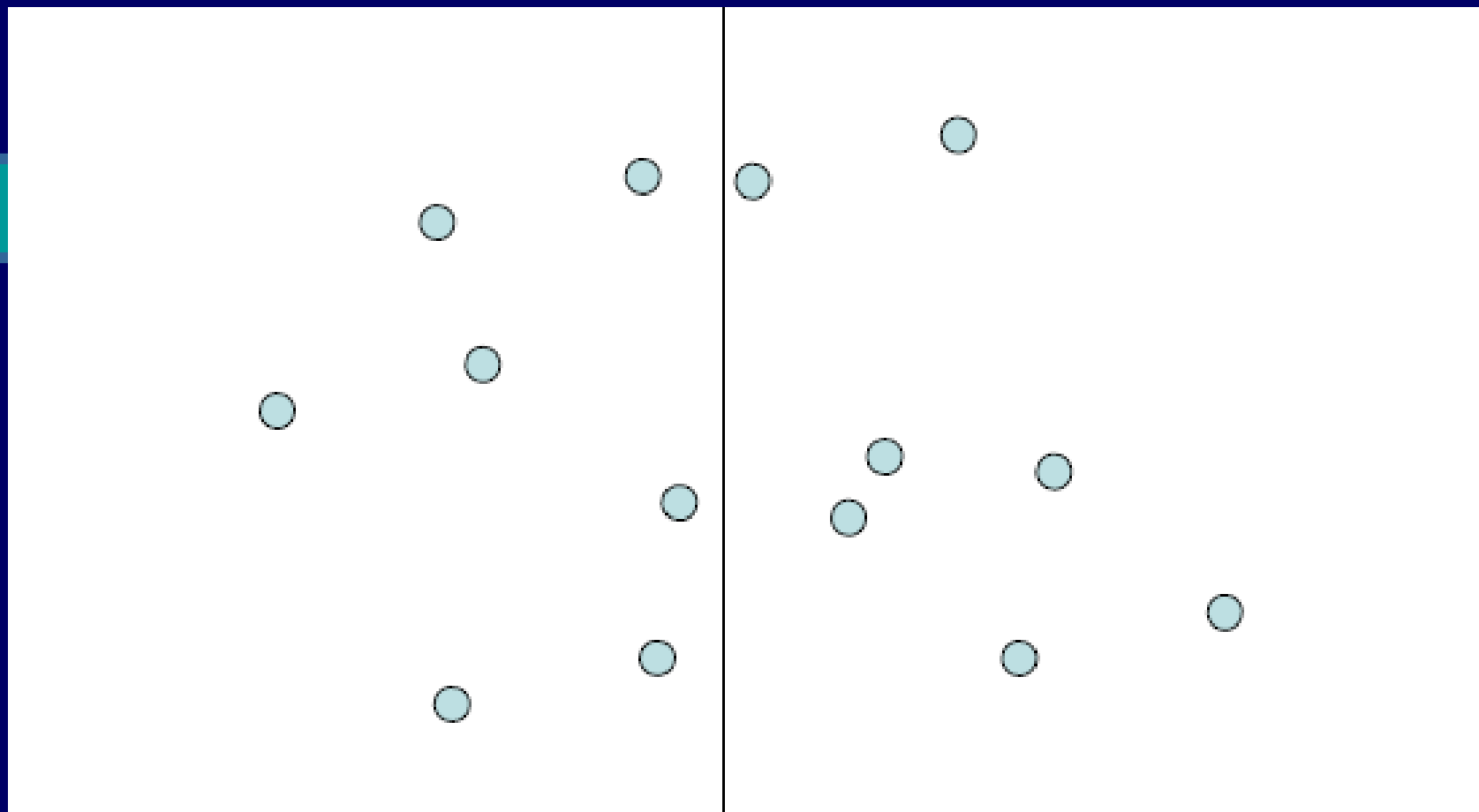
- 点集の $x$ 座標の昇順で2分割する
- 左右の部分で再帰的に凸包を求める
- 2つの凸包の共通外接線を求める
- 全体の点集合の凸包を構成する





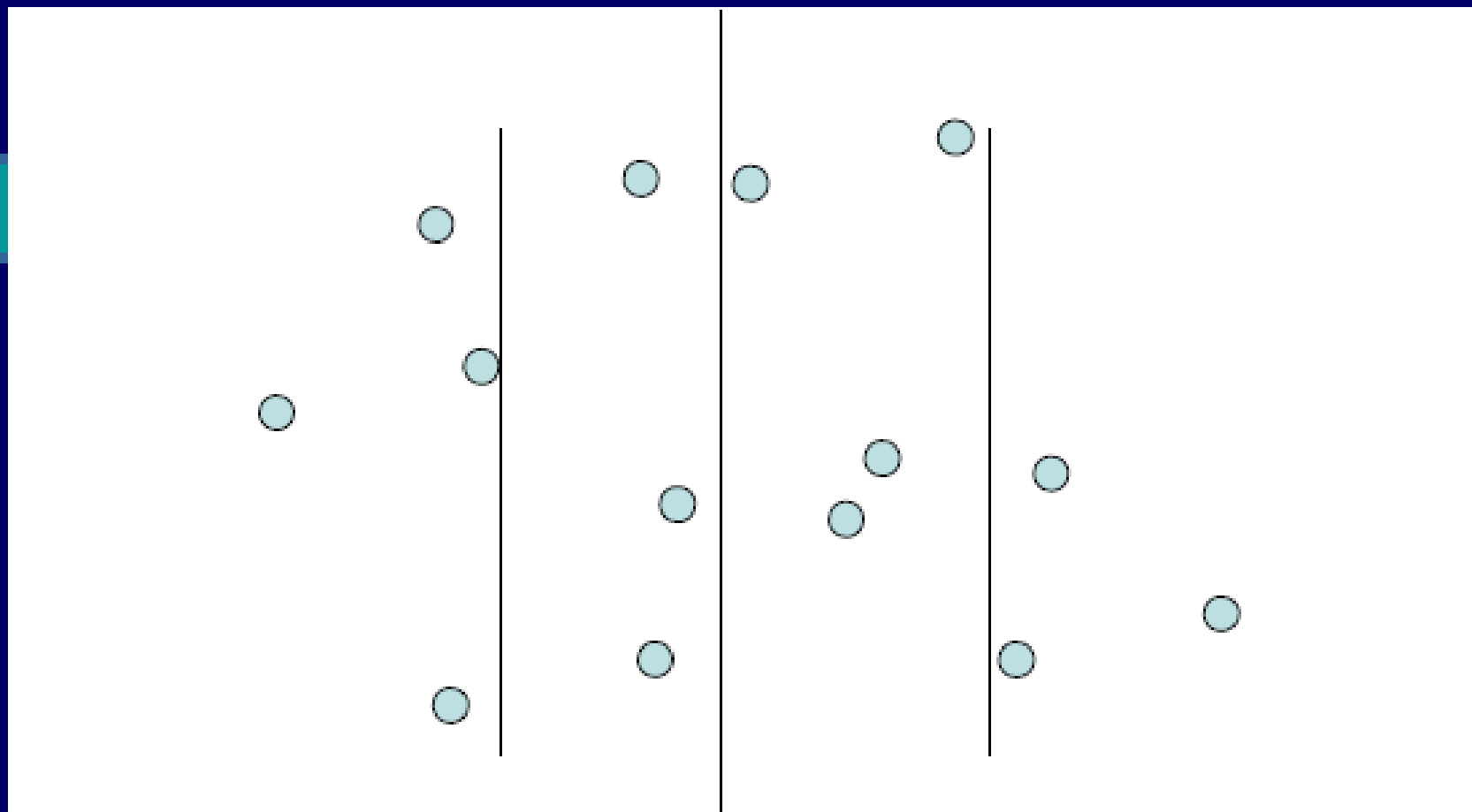


# 分割統治法



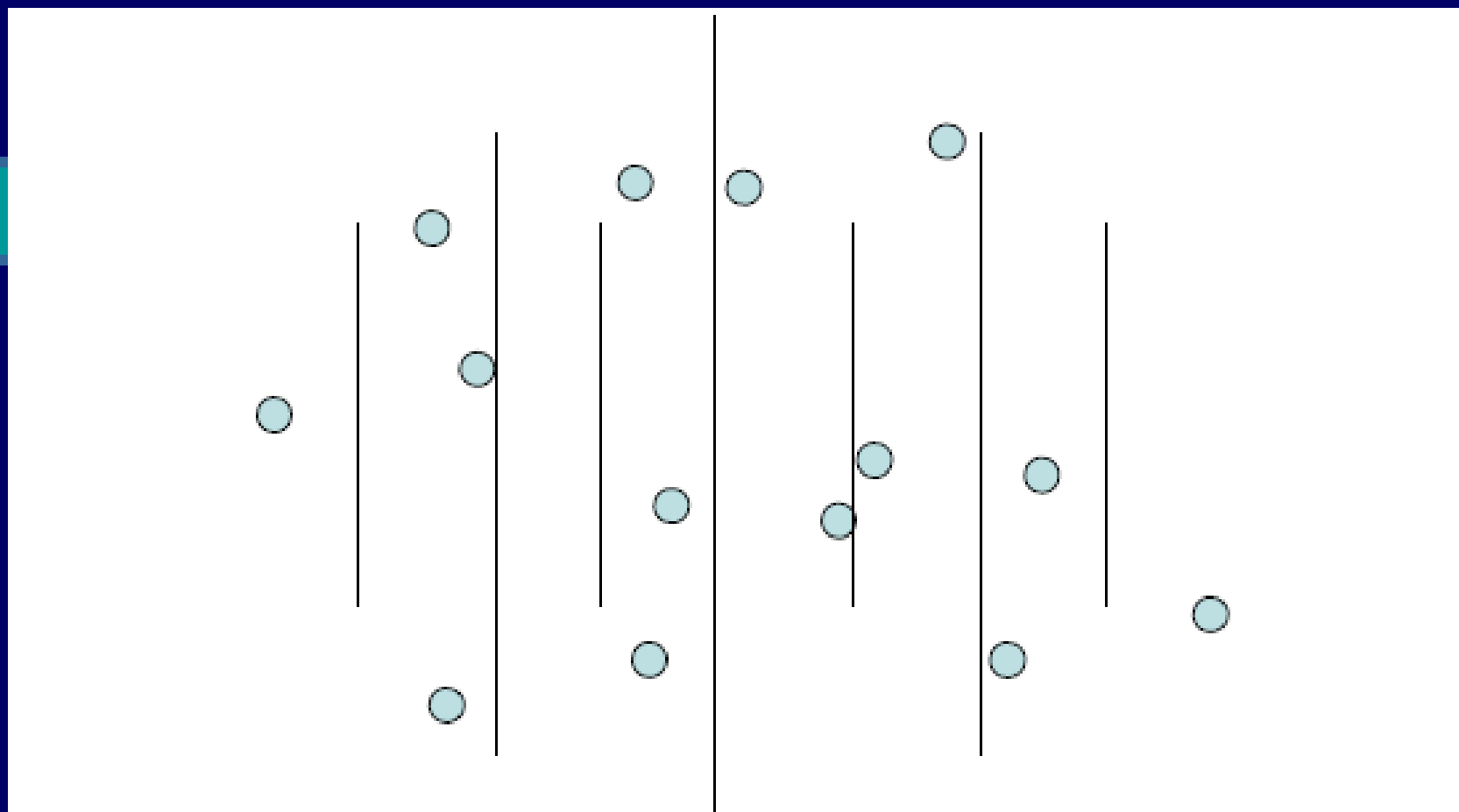


# 分割統治法



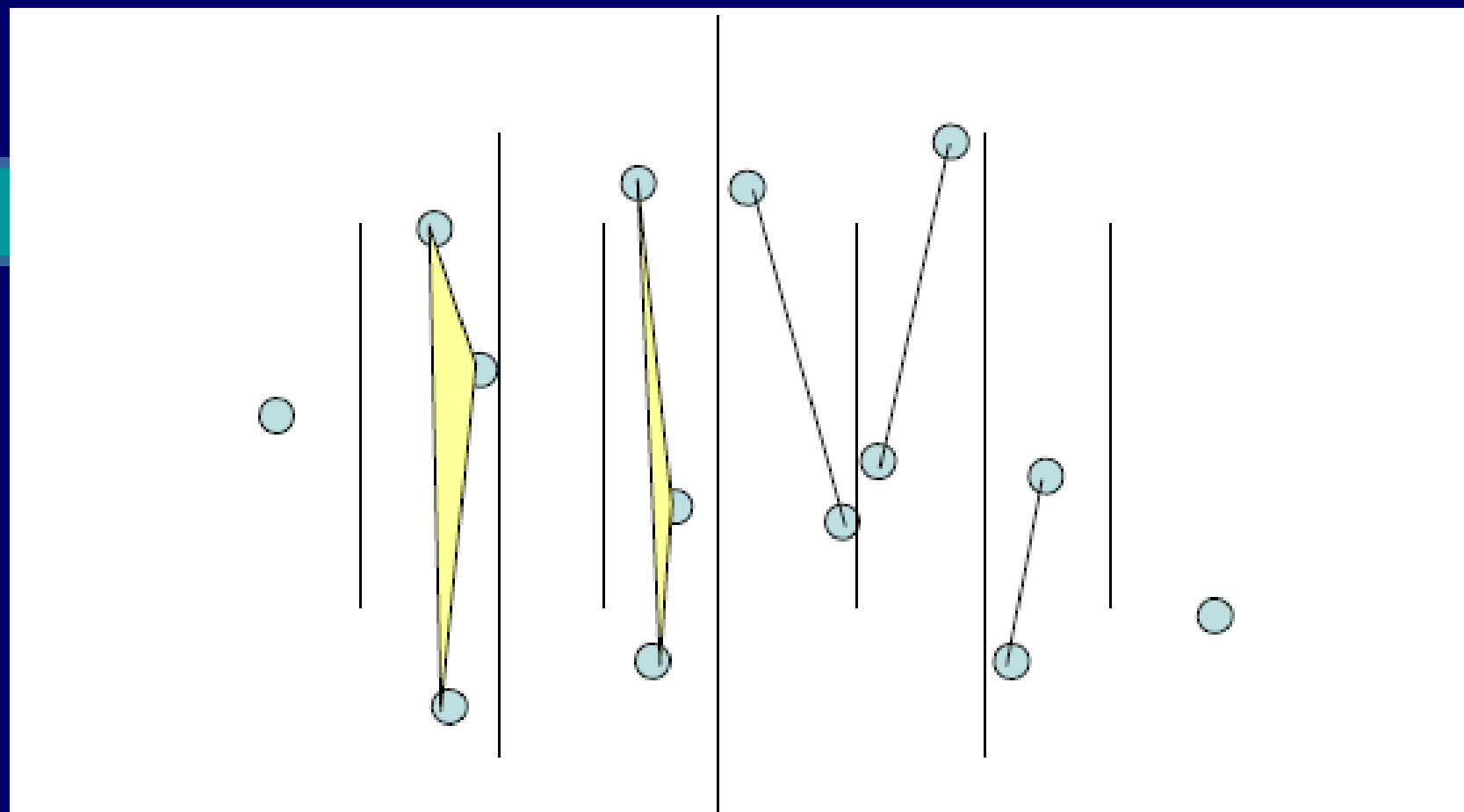


# 分割統治法



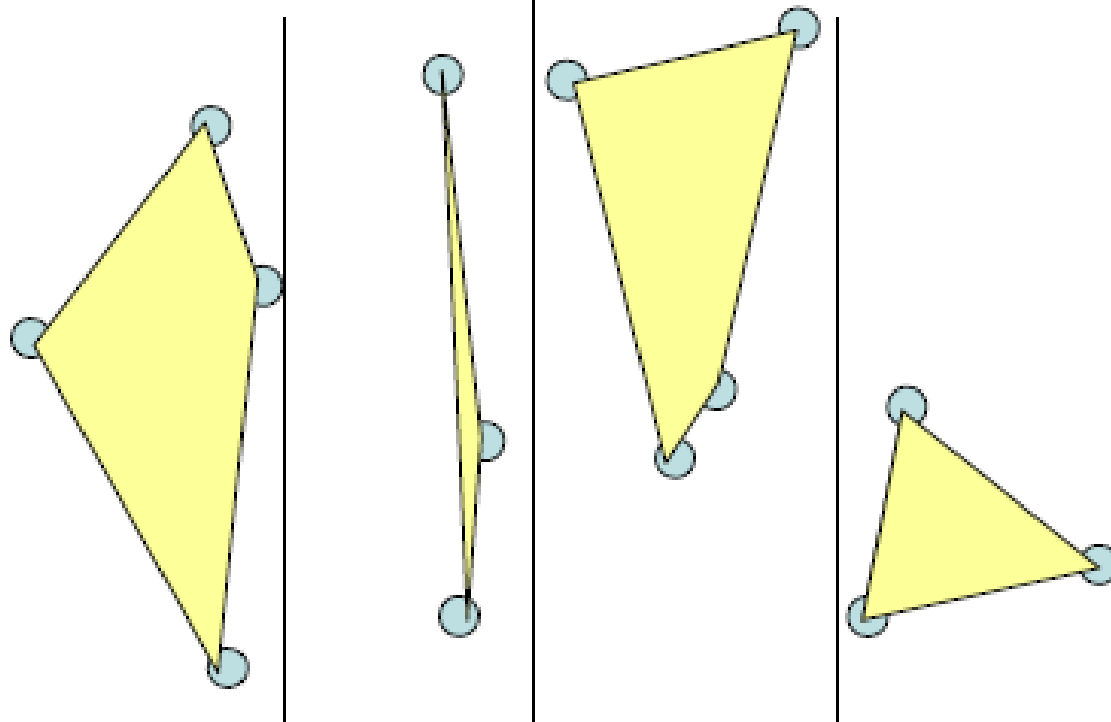


# 分割統治法



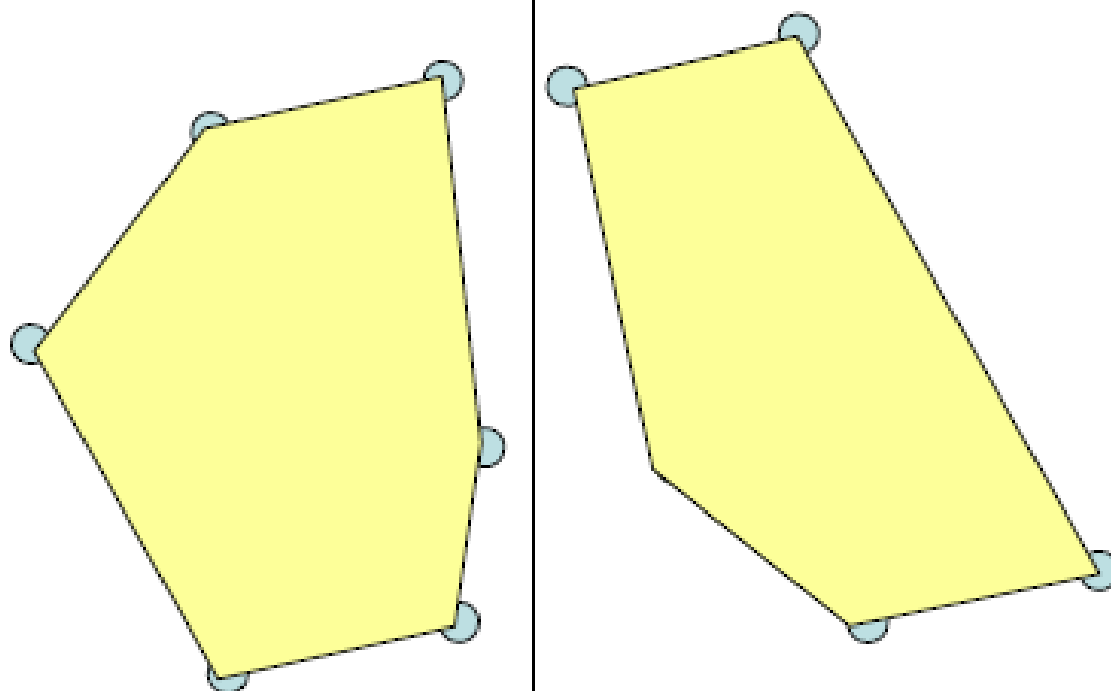


# 分割統治法



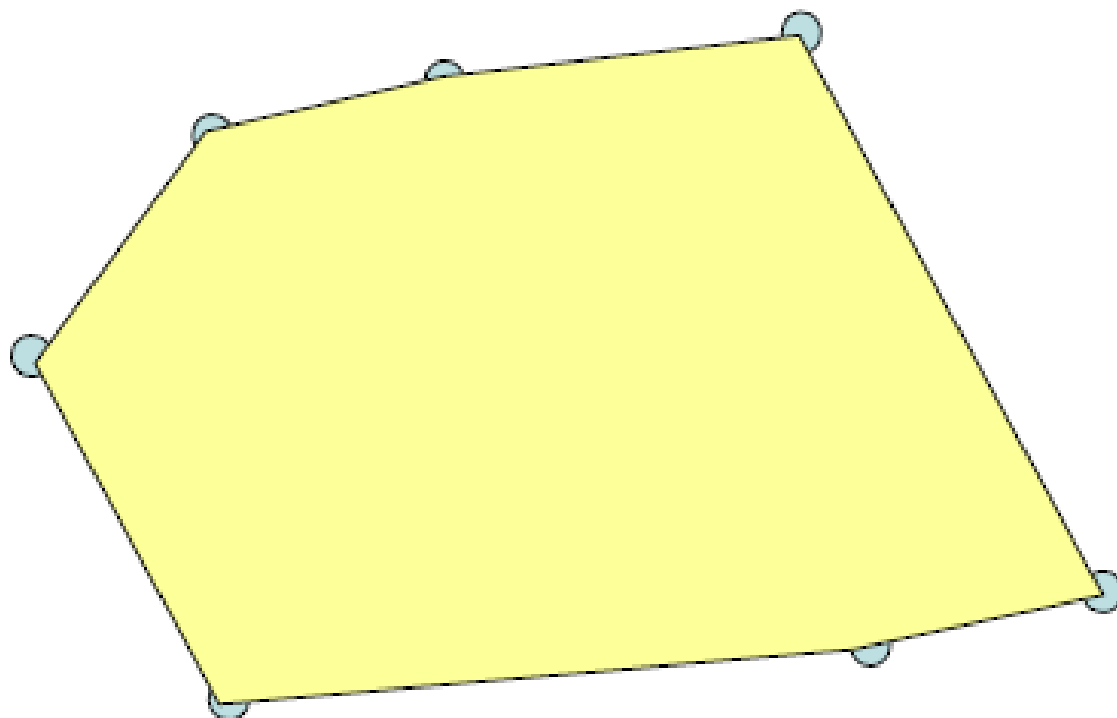


# 分割統治法





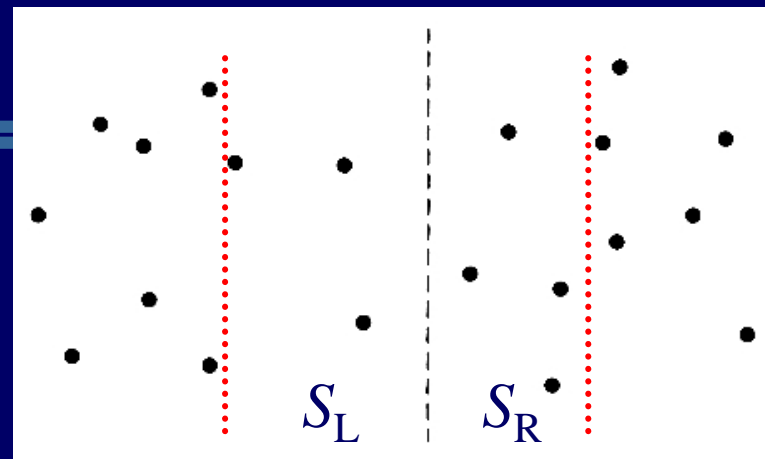
# 分割統治法



# アルゴリズム

D&CConvexHull( $S_n$ )

- 入力: 平面上の点集合 $S_n$
  - 出力:  $CH(S_n)$ の頂点を含むスタック $S$
1. 点集合 $S_n$ を $x$ 座標で昇順に並び替える
  2.  $x$ 座標の中央値を用いて左右の部分点集合 $S_L$ と $S_R$ に3点の小さい点集合まで分割する
  3.  $S_L$ と $S_R$ に対する凸包を再帰的に求める
  4. 上部・下部外接線を求め(?), 2つの凸包を統合する

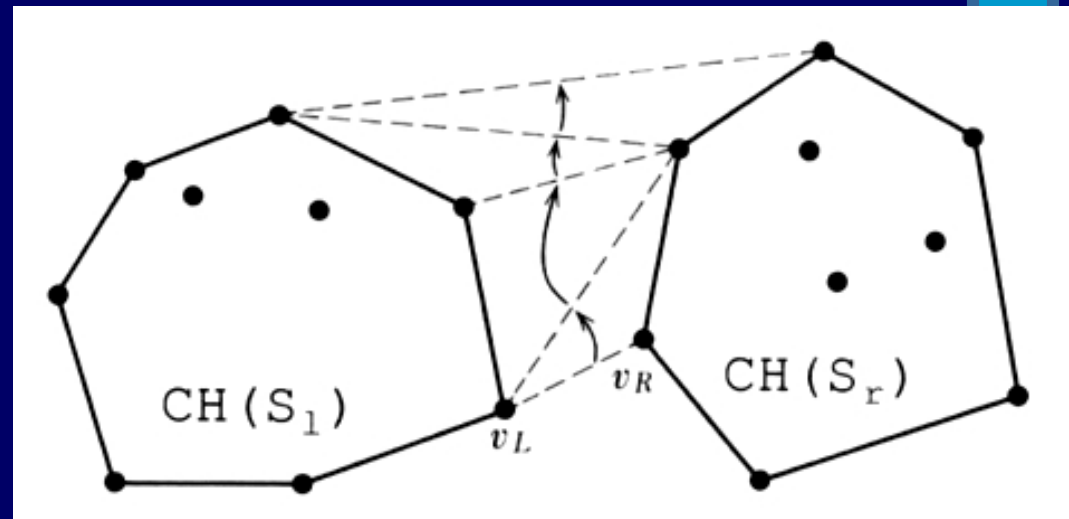




# 上部外接線の計算

- 最も右と最も左の頂点を選ぶ
- while  $\Delta(S_L[top], S_R[top], S_R[top-1]) = \text{逆時計回り}$
- $\text{pop}(S_R[top])$
- while  $\Delta(S_R[top], S_L[top], S_L[top-1]) = \text{時計回り}$
- $\text{pop}(S_L[top])$

- 逐次添加法？
- 下部外接線？



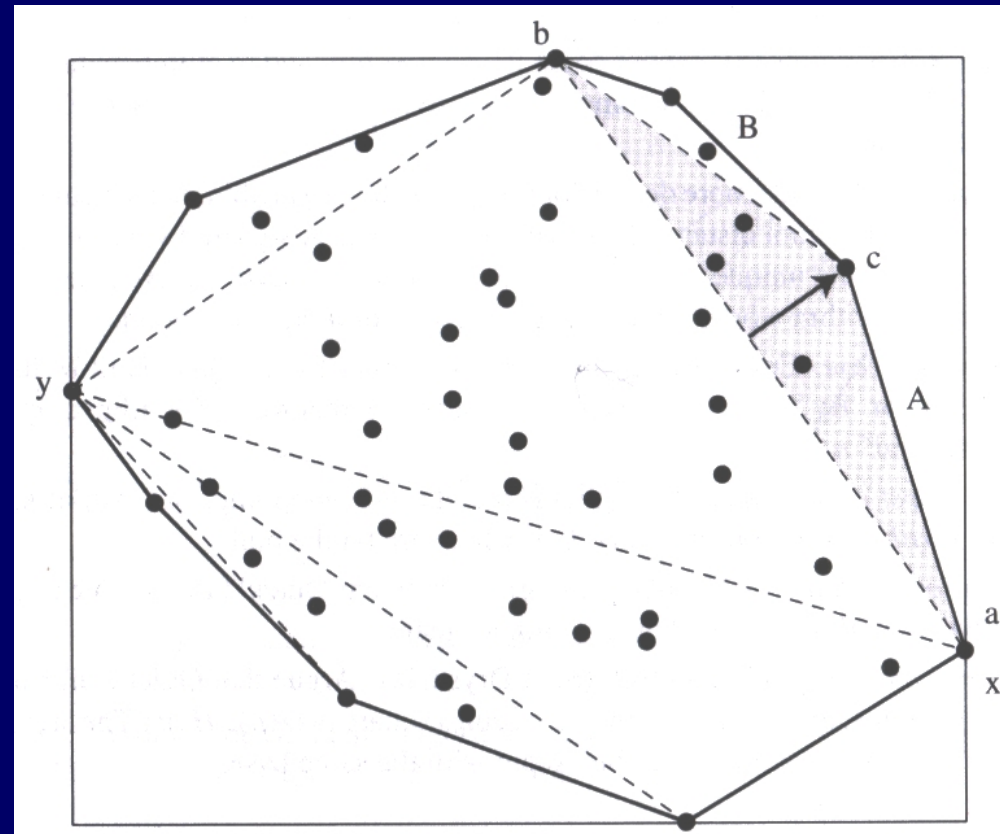


# 計算量

- 点集合のデータ数 =  $n$
- 点集合の並び替え =  $O(n \log n)$
- 接線を求める計算量 =  $O(n)$
- 再帰計算  $T(n) = 2T(n/2) + O(n)$
- $T(n) = 2T(n/2) + bn, T(1) = b$
- $T(n) = 2(2T(n/2^2) + bn/2) + bn$
- $= 2^2T(n/2^2) + 2bn$
- $\dots = 2^i T(n/2^i) + ibn$
- $n = 2^i$ 、即ち  $i = \log n$  の時、 $T(n) = nb + nb \log n$
- 総計算量 =  $O(n \log n)$

# Quick法(Quick method)

- Similar to Quick Sort
- 基本的な考え方
  - 2つ極点を見つける
    - 右端最低点 $x$
    - 左端最高点 $y$
  - Upper hullとLower hullに分けられる
  - 線分 $xy$ との距離が一番遠い極点 $b$ を見つける
  - 線分 $ab$ の右側の極点 $c$ を見つける
  - そして、再帰的に線分 $ac$ , 線分 $cb$ の極点を見つける



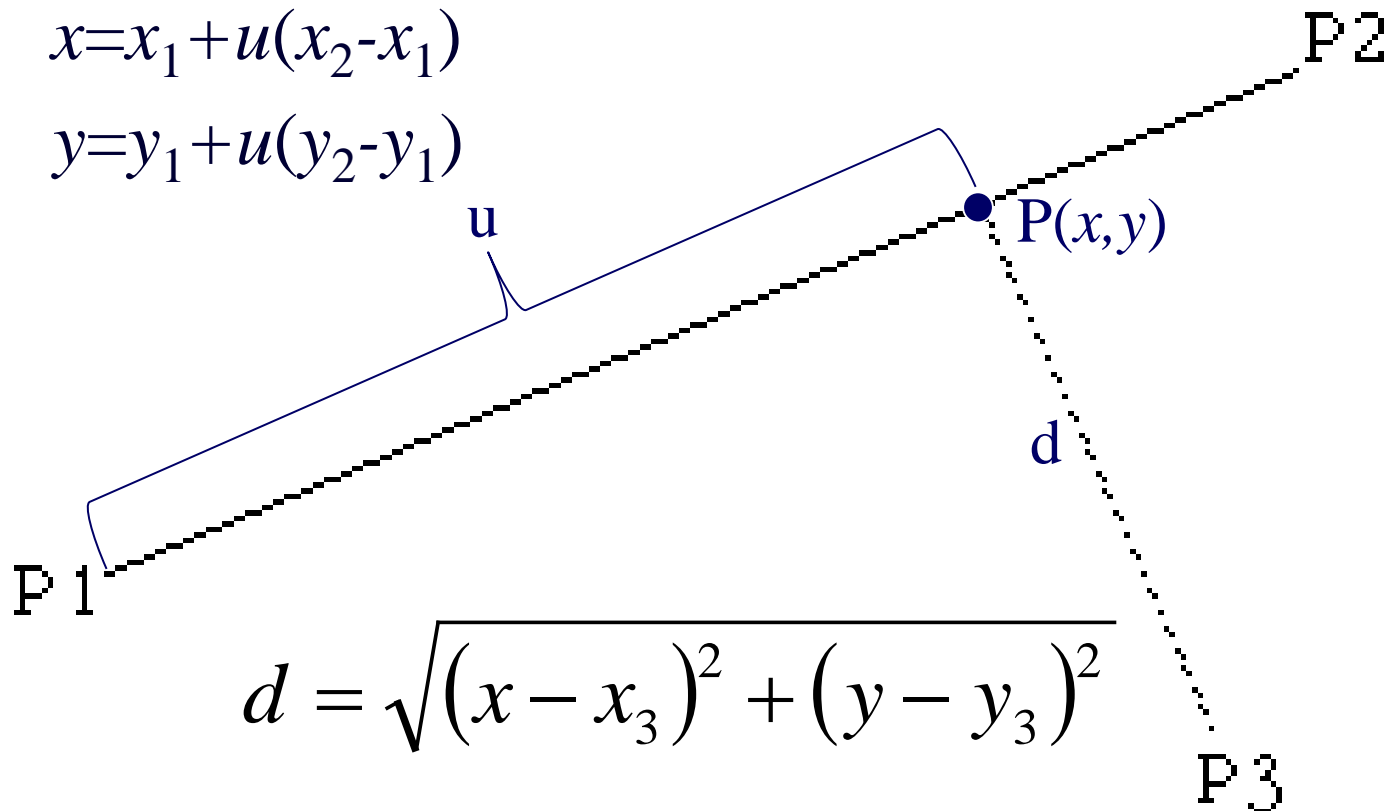


# 点と線分間の距離

$$u = \frac{(x_3 - x_1)(x_2 - x_1) + (y_3 - y_1)(y_2 - y_1)}{\|p_2 - p_1\|^2}$$

$$x = x_1 + u(x_2 - x_1)$$

$$y = y_1 + u(y_2 - y_1)$$



$$d = \sqrt{(x - x_3)^2 + (y - y_3)^2}$$

# アルゴリズム

## QuickConvexHull( $S_n$ )

- 入力: 平面上の点集合 $S_n$
  - 出力:  $CH(S_n)$ の頂点を含むリスト $L$
1. 2つの極点、右端最低点 $x$ , 左端最高点 $y$ を見つける
  2.  $S_n$ を上部点集 $S_u$ と下部点集 $S_l$ に分ける
  3.  $L = (x) + \text{QuickHull}(x, y, S_u) + (y) + \text{QuickHull}(y, x, S_l)$

## Function QuickHull( $a, b, S$ )

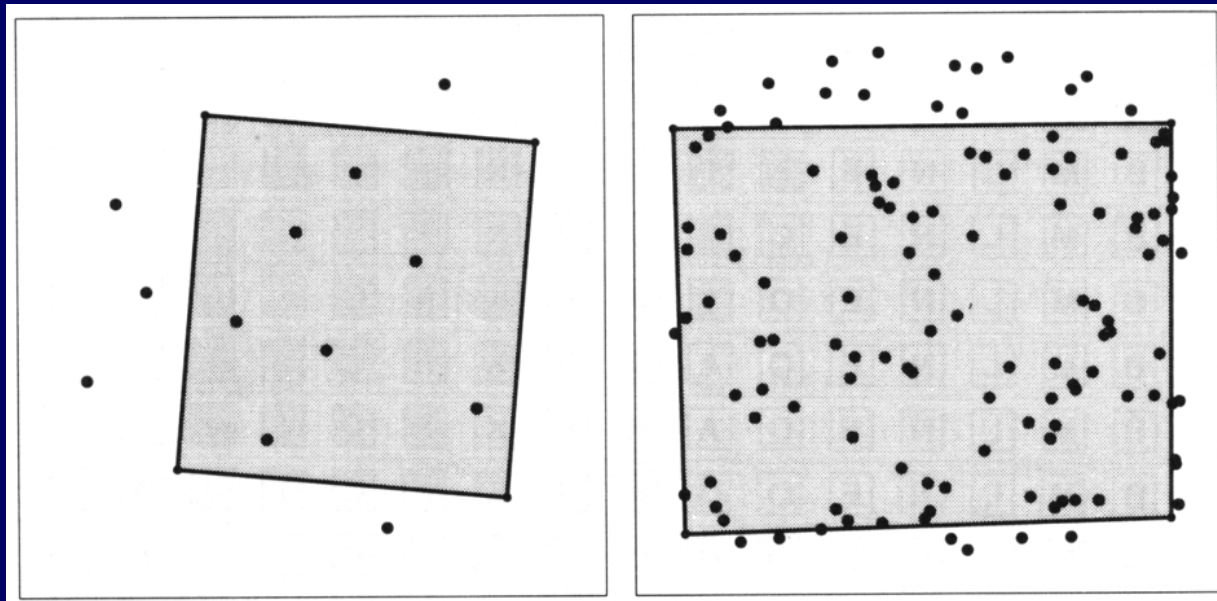
1. if  $S = \text{empty}$  then return;
2. else
3.  $c \leftarrow$ 線分 $ab$ から最も離れる点
4.  $A \leftarrow$ 線分 $ac$ の右側の点集
5.  $B \leftarrow$ 線分 $cb$ の右側の点集
6. return QuickHull( $a, c, A$ )+(c)+QuickHull( $c, b, B$ )

# 計算量

- 点集合のデータ数 =  $n$
- 極点の探索 =  $O(n)$
- 再帰計算  $T(n) = T(n_1) + T(n_2) + O(n)$   
但し、 $n_1 + n_2 \leq n$
- 仮に  $n_1 \approx n_2$ ,  $T(n) = 2T(n/2) + O(n)$
- 総計算量 =  $O(n \log n)$

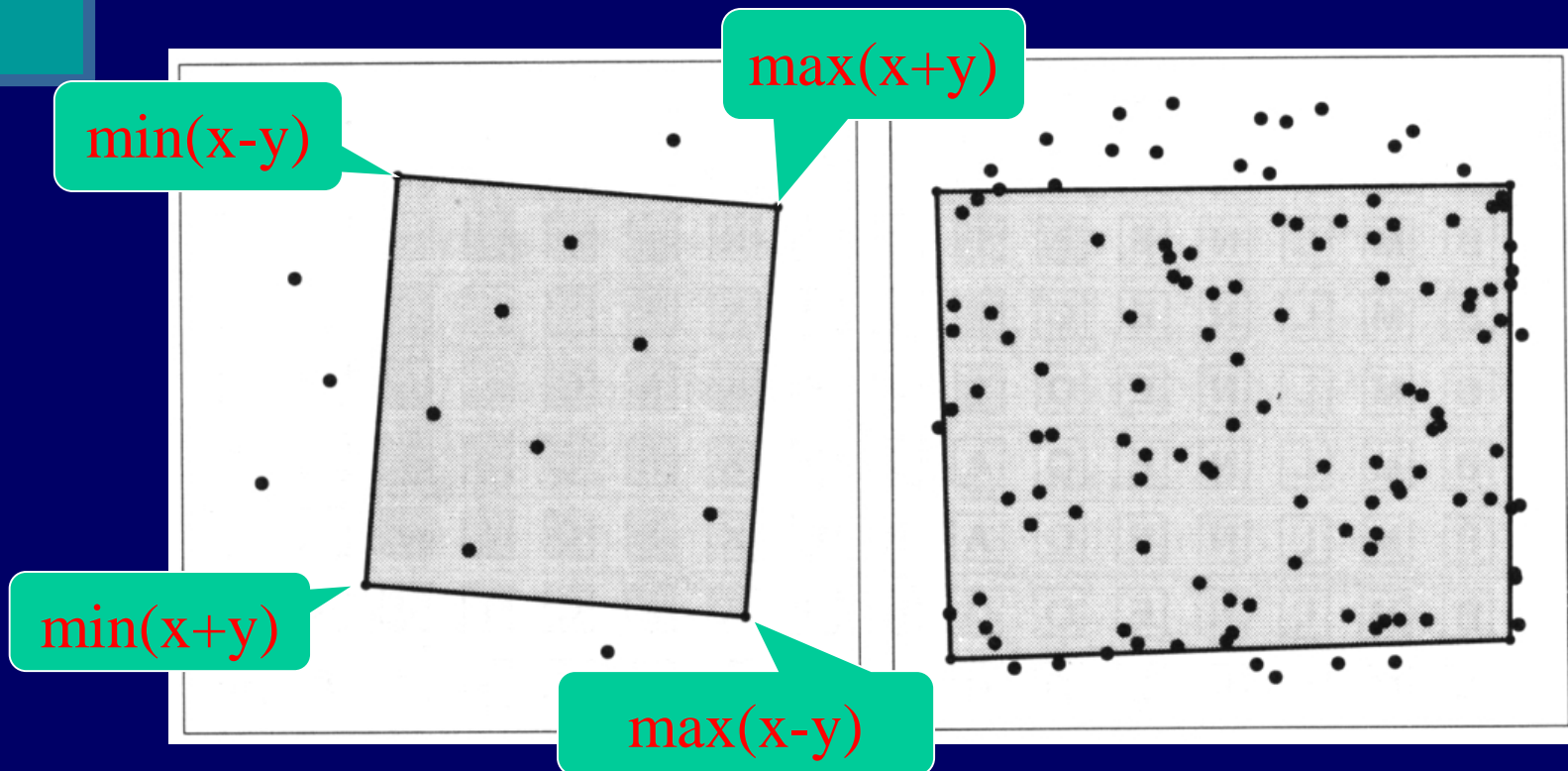
# 内点消去法 (Inner points elimination)

- ある四角形の内部にあり、明らかに凸包頂点ではない点集を消去すると、考慮すべき点数↓↓！
- 四角形頂点の選択(?)と内点の判断(?)



# 四角形頂点の選択

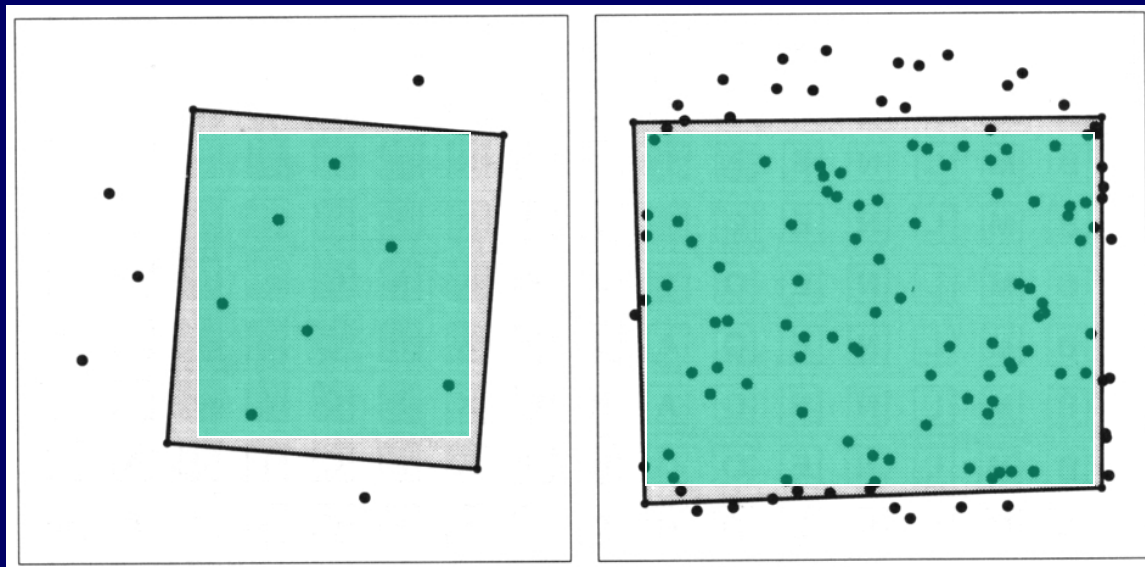
- 点集の分布に応じて選択できれば良い
- 均等分布であれば、下記の4点を選ぶ





# 内点の判断

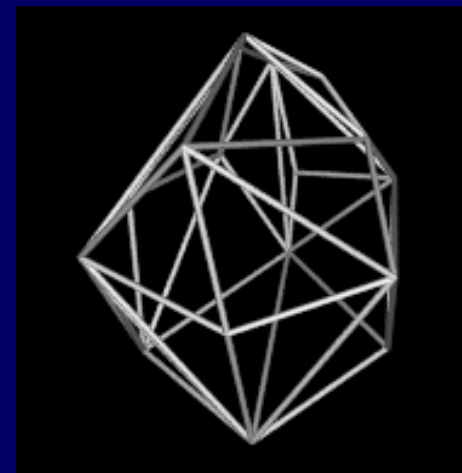
- 各点が四角形の内部に入っているかどうかを判断
- 座標軸に平行な長方形を用いれば、四角形頂点の座標のみから、内点を判断可能→直交領域探索



# 計算量

- 点集合のデータ数 =  $n$
- 点集の並び替え =  $O(n \log n)$
- 点の位置判断(内外) =  $O(n)$
- 総計算量 =  $O(n \log n)$

# 応用



## ■ 経路選択

- 路幅、高度制限のある道路、トンネルを通過できるかどうか
- 障害物を回避できるかどうか

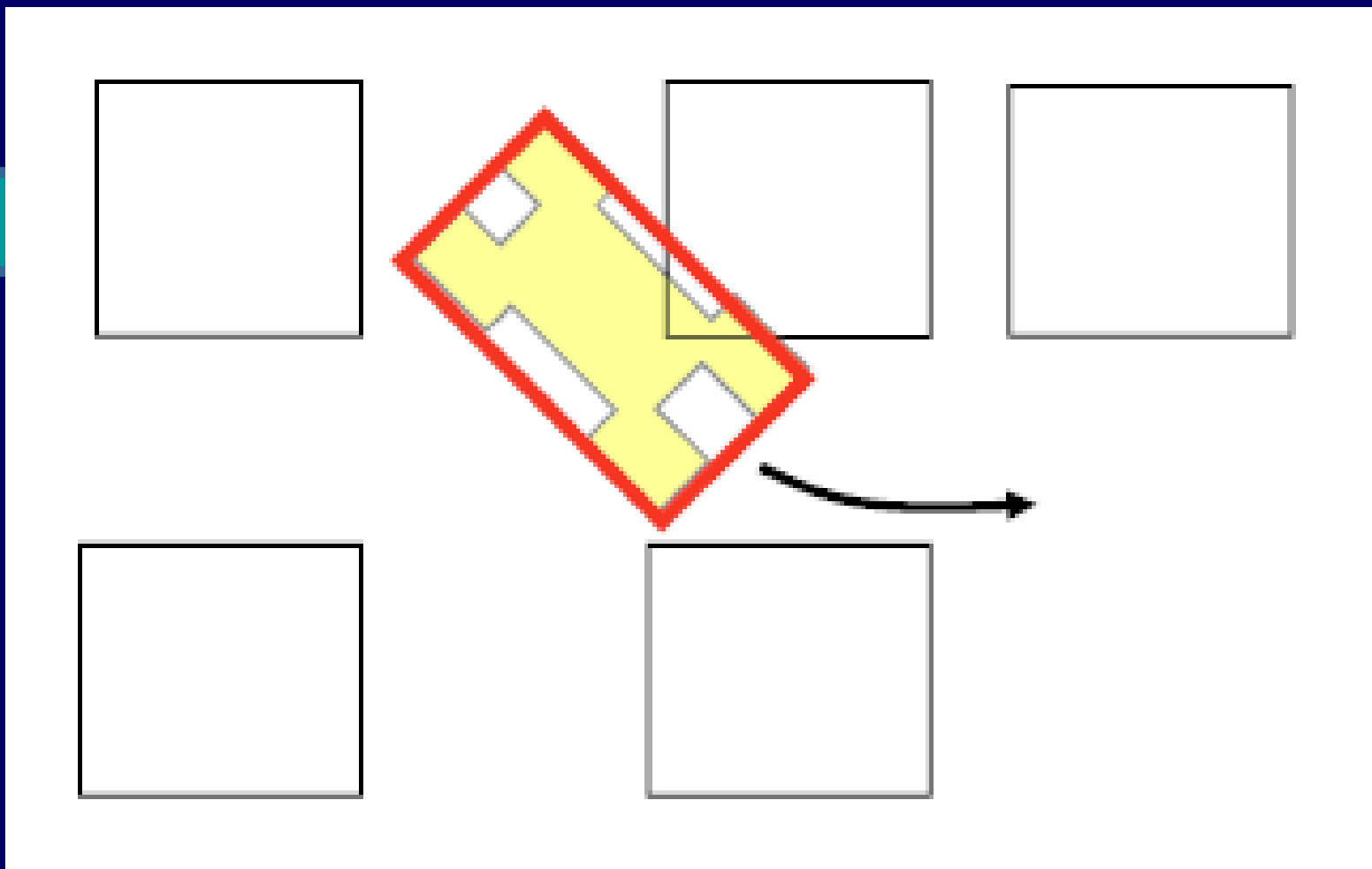
## ■ 最小面積、最小体積

- シリコンチップ(回路基板)、コンテナの選択

## ■ 自動分類

- 各クラスターの最小範囲、類別の境界定め

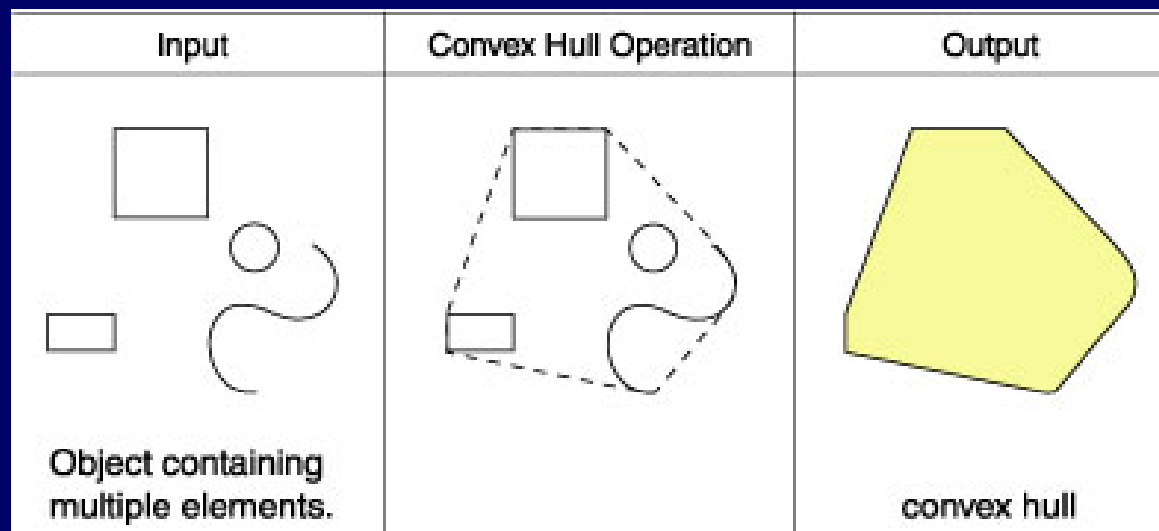
# ロボット衝突判定



# MapInfo

## ■ Convex-Hull Function Example

Returns the minimum boundary around a spatial object without the boundary being concave. The result is the spatial object representing the convex hull of the perimeter



# ダンサー凸包



会津大学

