

DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

DUALITY AND APPLICATIONS OF ARRANGEMENTS

PETR FELKEL

FEL CTU PRAGUE

felkel@fel.cvut.cz

<https://cw.felk.cvut.cz/doku.php/courses/a4m39vg/start>

Based on [Berg], [Mount], and [Goswami]

Version from 5.1.2017

Talk overview

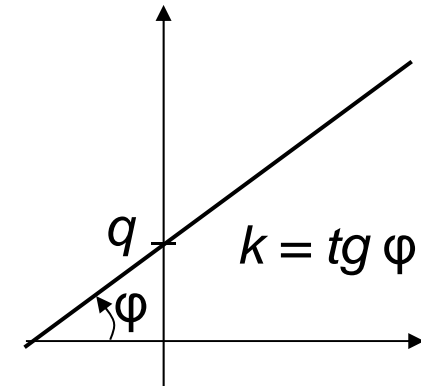
- Duality
 1. Points and lines
 2. Line segments
 3. Polar duality (different points and lines)
 4. Convex hull using duality
- Applications of duality and arrangements



1. Duality of lines and points in the plane

- Points and lines - both have 2 parameters:

- Points – coords x and y
- Lines – slope k and y -intercept q
 $y = kx + q$



- We can simply **map** points and lines 1:1
- Many mappings exist – it depends on the context



Why to use duality?

Some reasons why to use duality:

- Transforming a problem to dual plane may give a **new view on the problem**
- Looking from a different angle may **give the insight** needed to solve it
- Solution in dual space may be even simpler

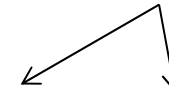


Definition of duality transformation D

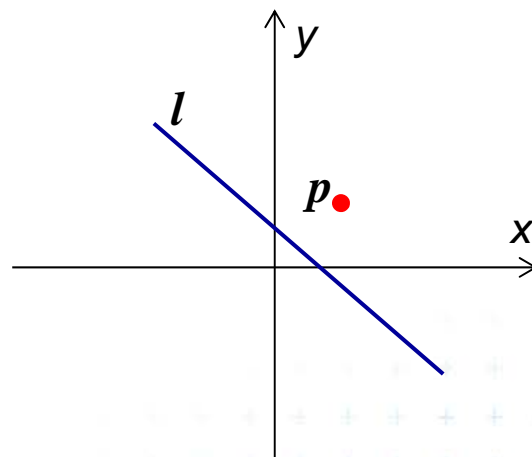
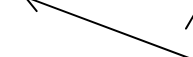
Let D be the duality transform:

- Point $p = [p_x, p_y]$ is transformed to line $D_p = p^* := (b = p_x a - p_y)$
- Line $l : (y = ax - b)$ is transformed to point $D_l = l^* := [a, b]$

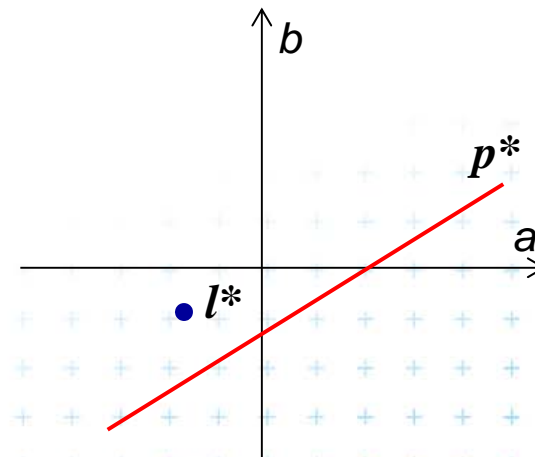
variables



constants



Primal plane (xy)



Dual plane (ab)



Example and more about duality D

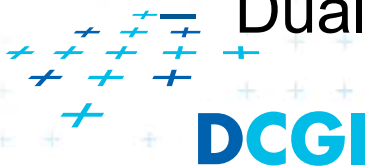
- Example:
line $y = 5x - 3$
can be represented as point $y^* = [5, 3]$

See the [applet]

- Duality D

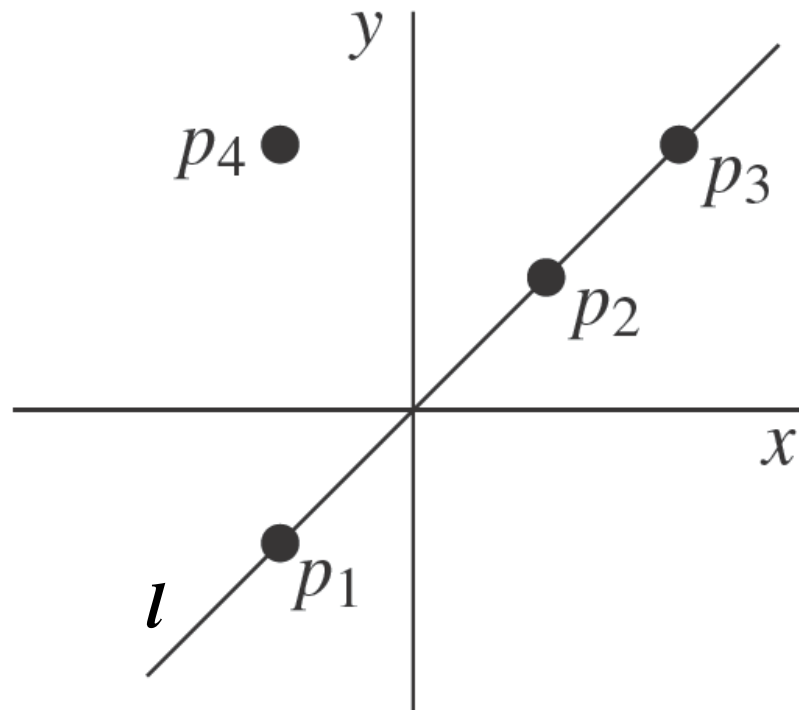
- is its own **inverse** $DD_p = p, DD_l = l$
- cannot represent **vertical lines**
=> Take vertical lines as special cases, use lexicographic order, or rotate the problem space slightly.

- Primal plane – plane with coordinates x, y
- Dual plane* – plane with coordinates a, b

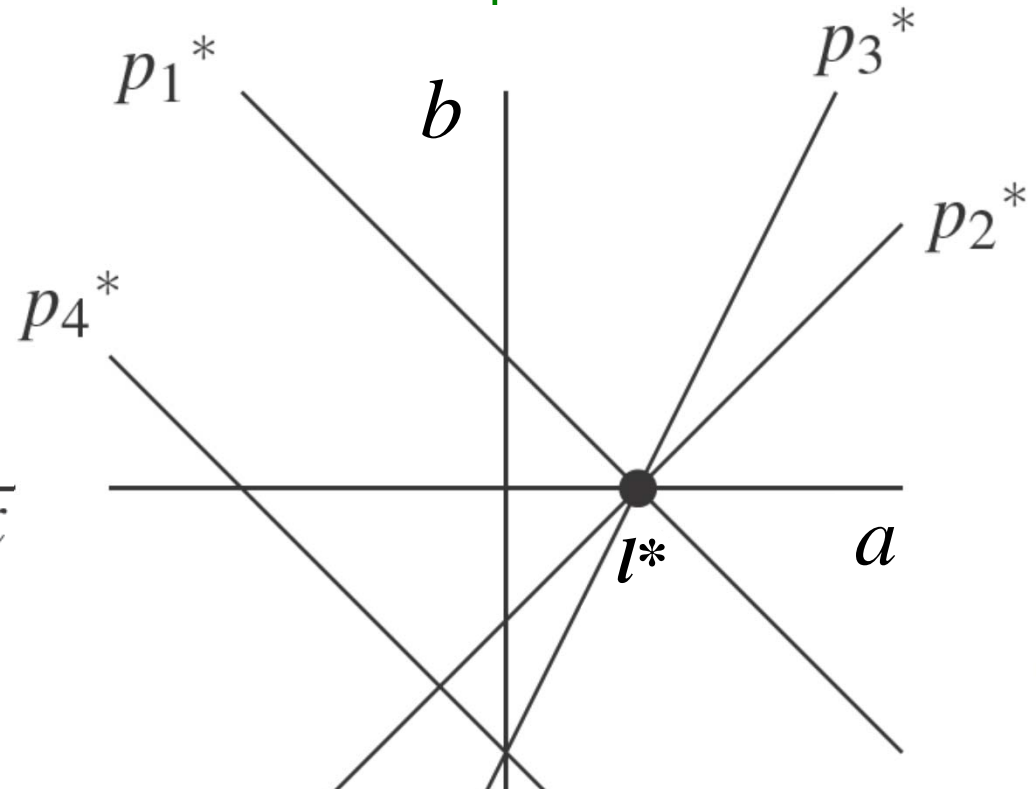


Duality of lines and points in the plane

Primal plane



Dual plane

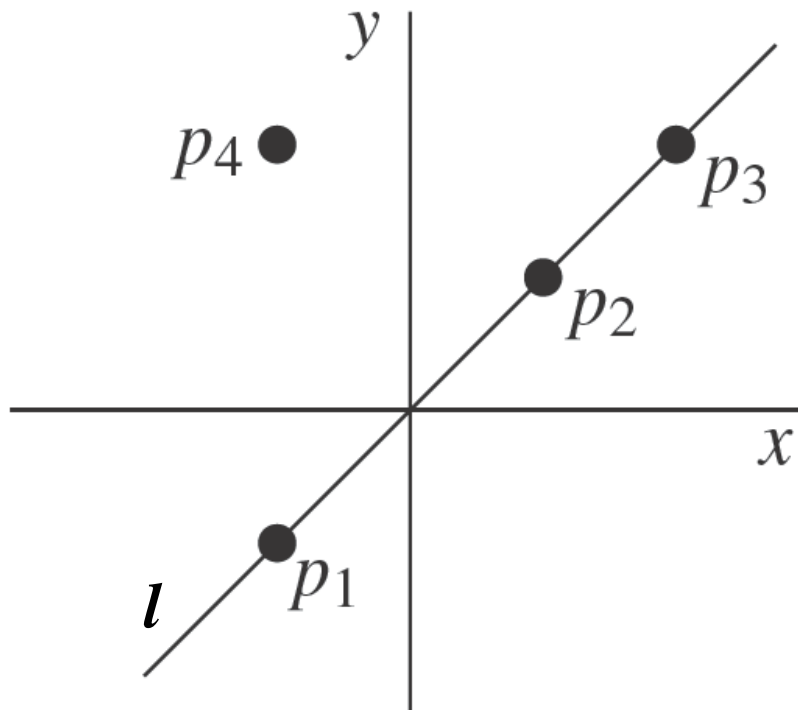


[Berg]



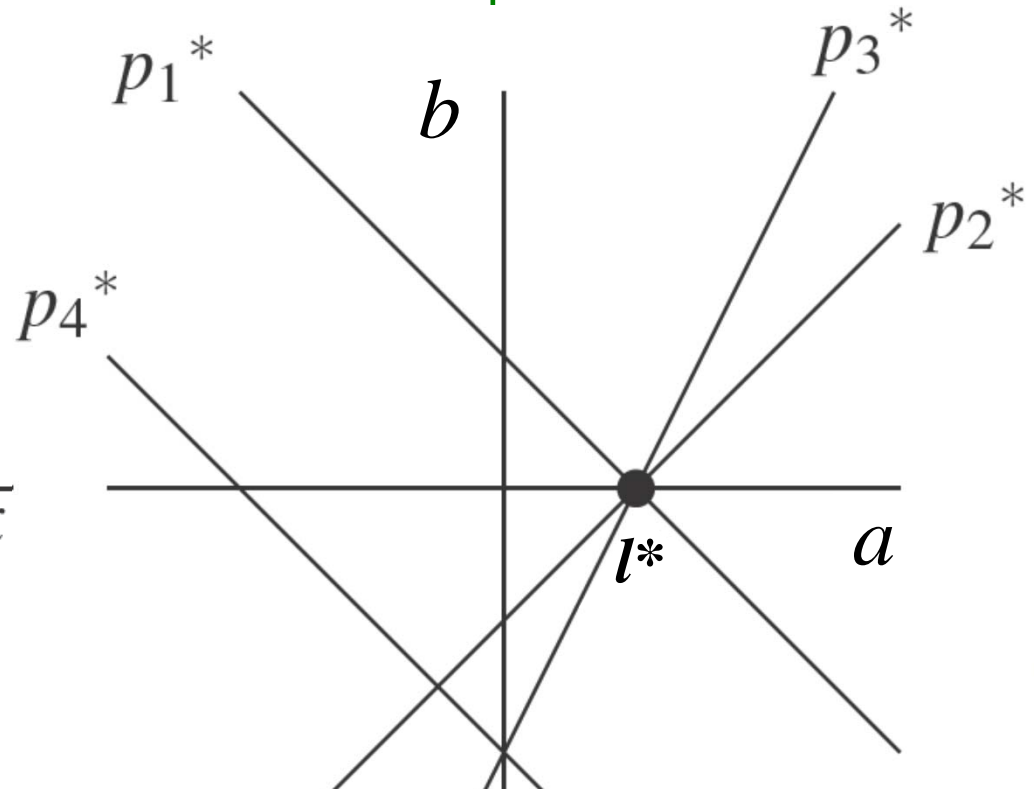
Duality of lines and points in the plane

Primal plane



point $p = [p_x, p_y]$

Dual plane

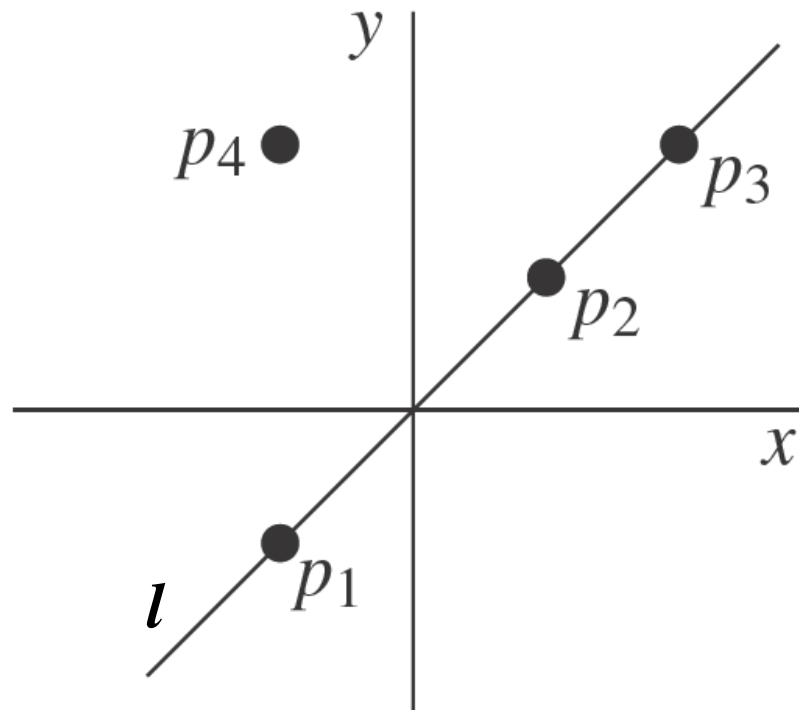


[Berg]



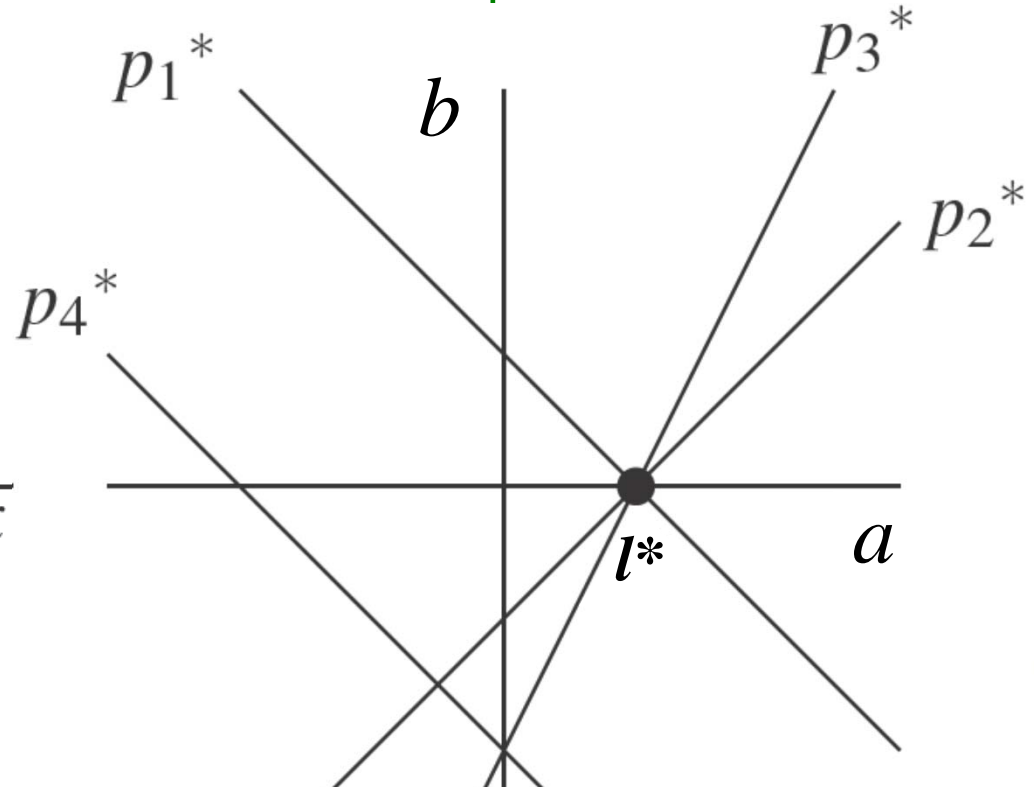
Duality of lines and points in the plane

Primal plane



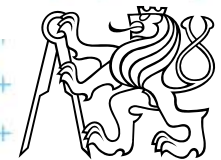
point $p = [p_x, p_y]$

Dual plane



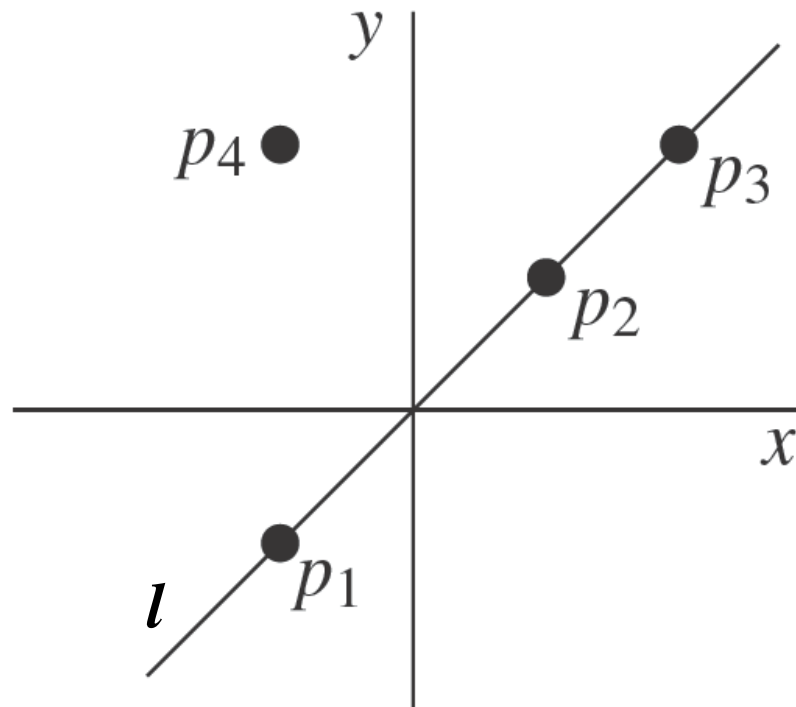
line $p^* := (b = p_x a - p_y)$

[Berg]



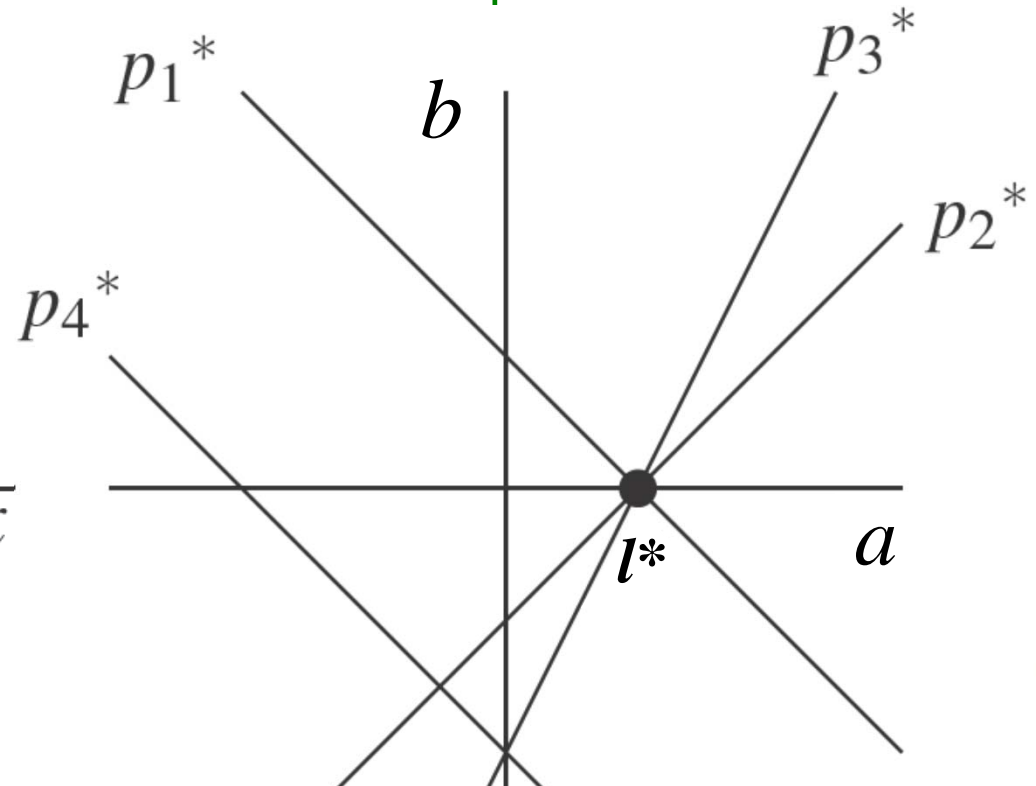
Duality of lines and points in the plane

Primal plane



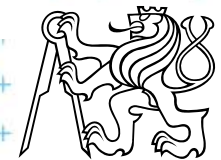
point $p = [p_x, p_y]$
 line $l := (y = ax + b)$

Dual plane



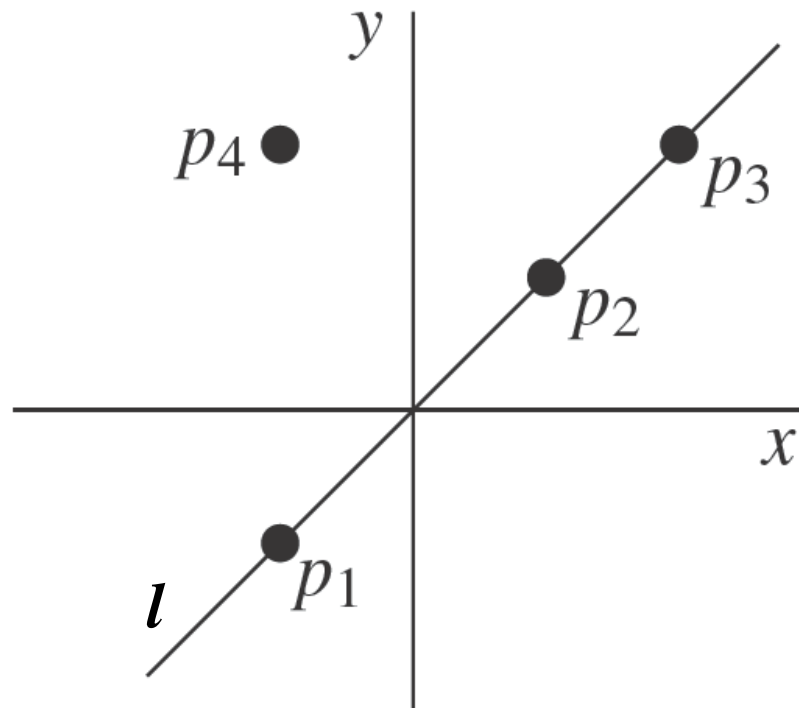
line $p^* := (b = p_x a - p_y)$

[Berg]



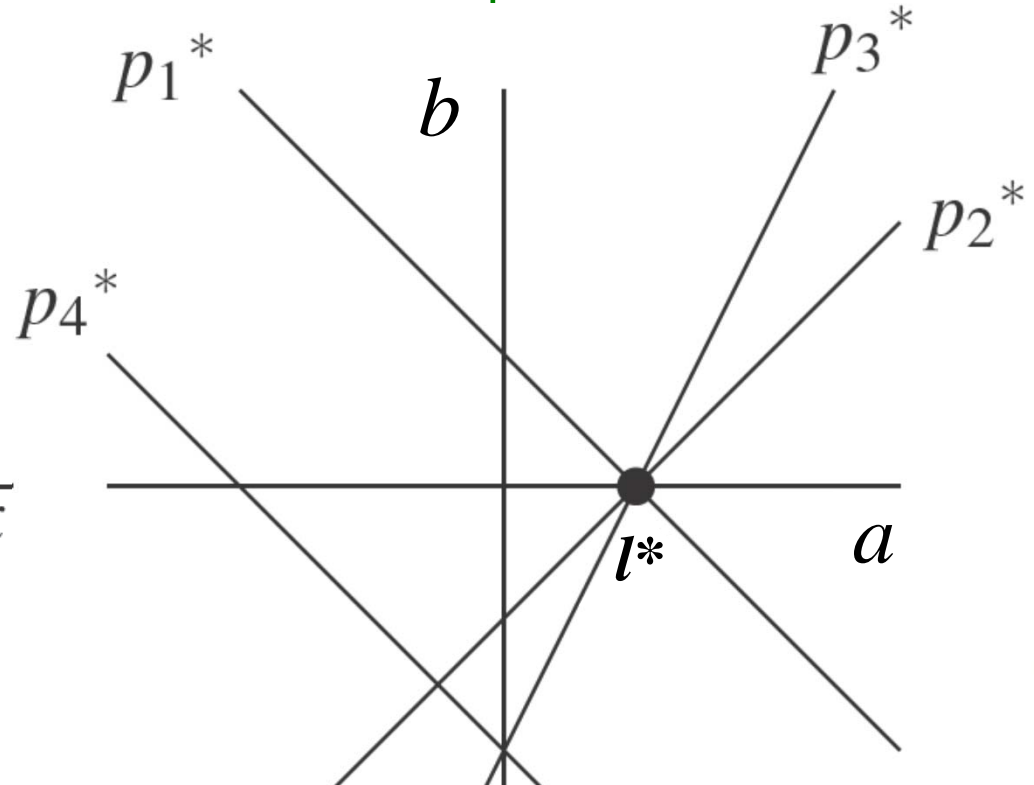
Duality of lines and points in the plane

Primal plane



point $p = [p_x, p_y]$
 line $l := (y = ax + b)$

Dual plane



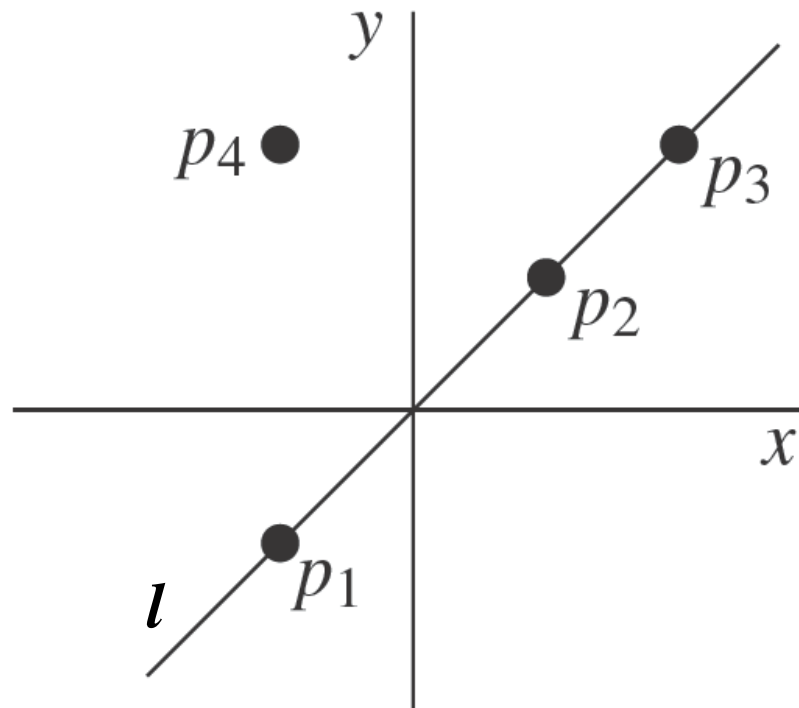
line $p^* := (b = p_x a - p_y)$
 Point $l^* = [a, -b]$

[Berg]



Duality of lines and points in the plane

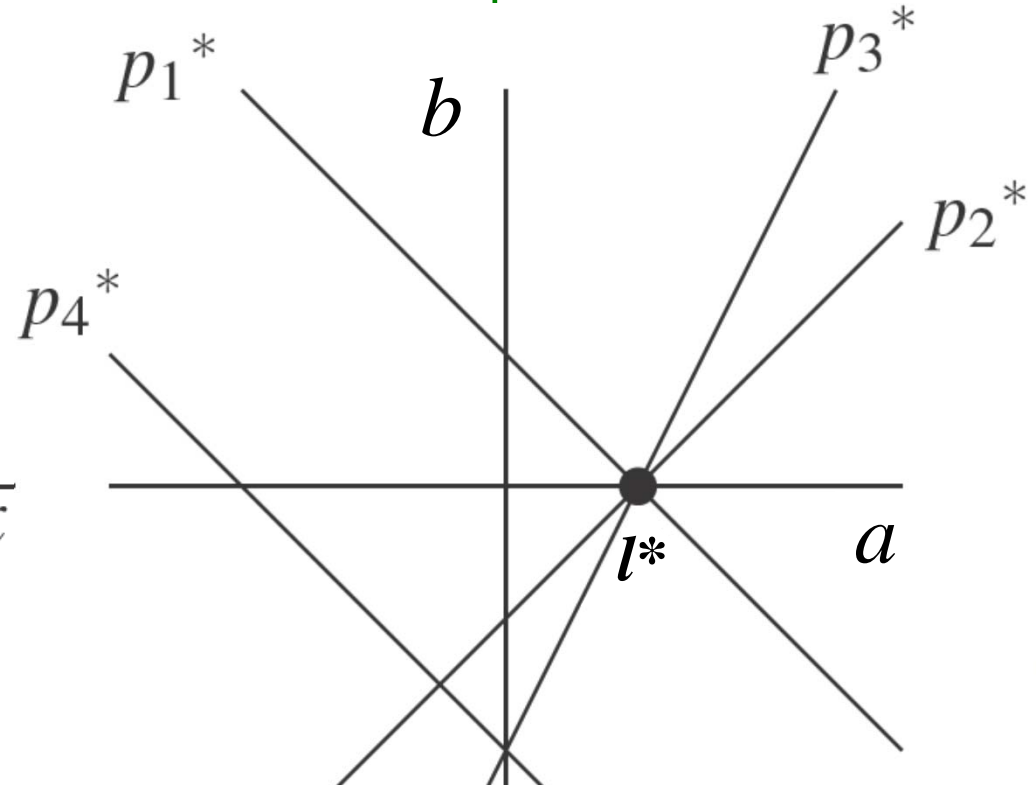
Primal plane



point $p = [p_x, p_y]$

line $l := (y = ax + b)$

Dual plane



line $p^* := (b = p_x a - p_y)$

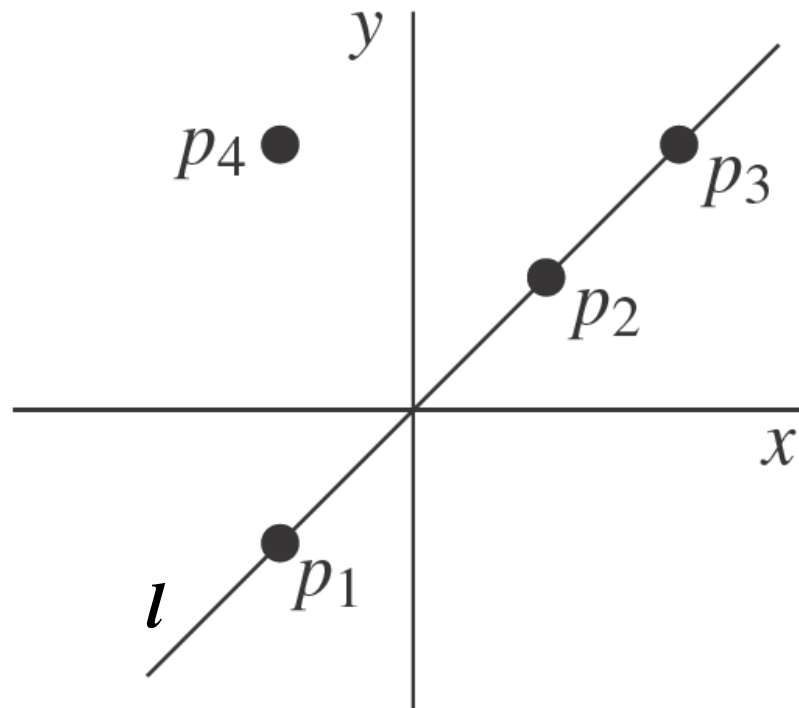
Point $l^* = [a, -b]$

[Berg]



Duality of lines and points in the plane

Primal plane

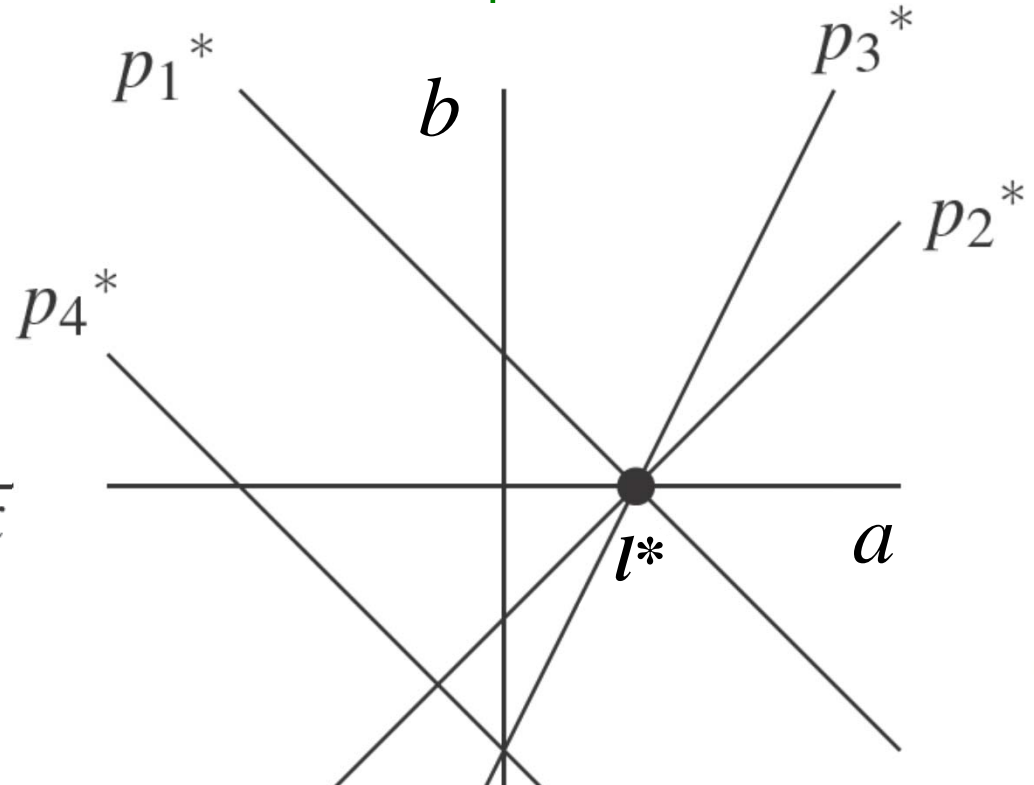


point $p = [p_x, p_y]$

~~line $l := (y = ax + b)$~~

line $l := (y = ax - b)$

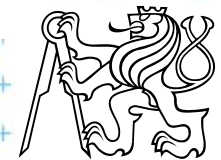
Dual plane



line $p^* := (b = p_x a - p_y)$

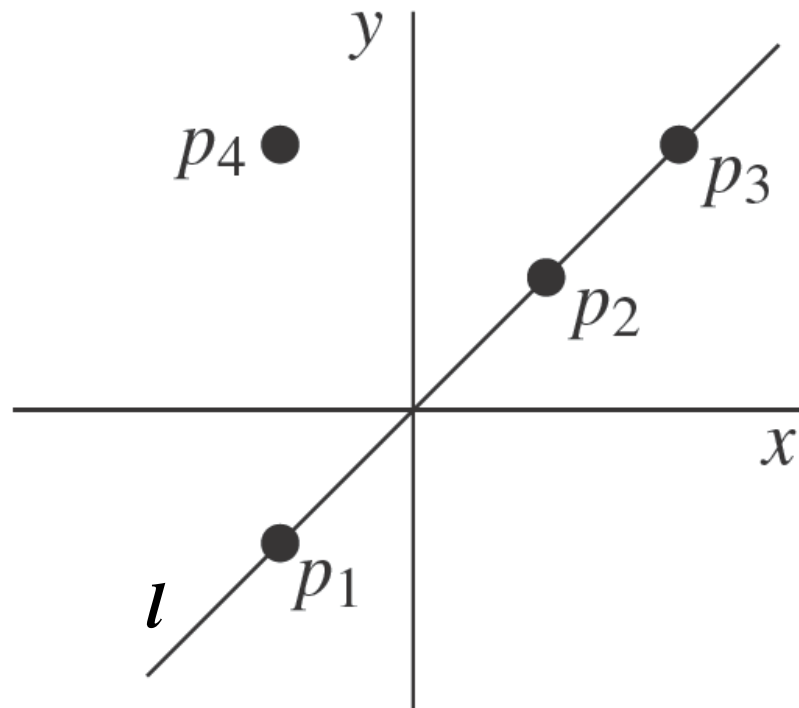
Point $l^* = [a, -b]$

[Berg]

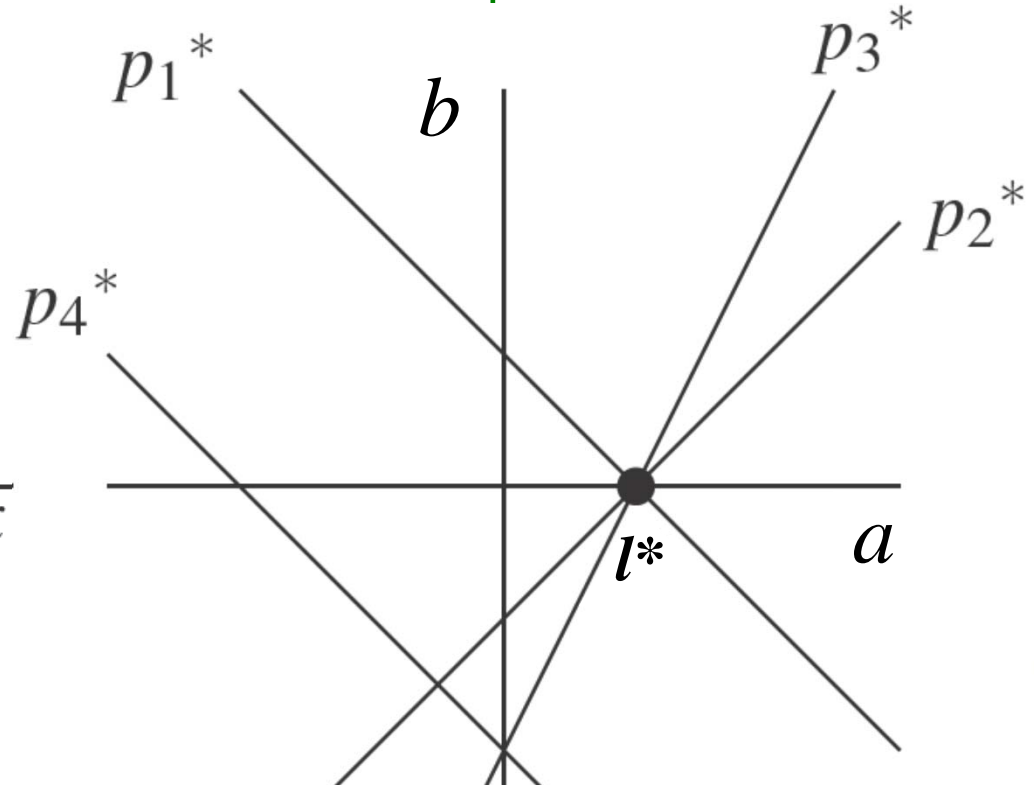


Duality of lines and points in the plane

Primal plane



Dual plane



point $p = [p_x, p_y]$

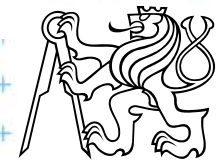
line $l := (y = ax + b)$

line $l := (y = ax - b)$

line $p^* := (b = p_x a - p_y)$

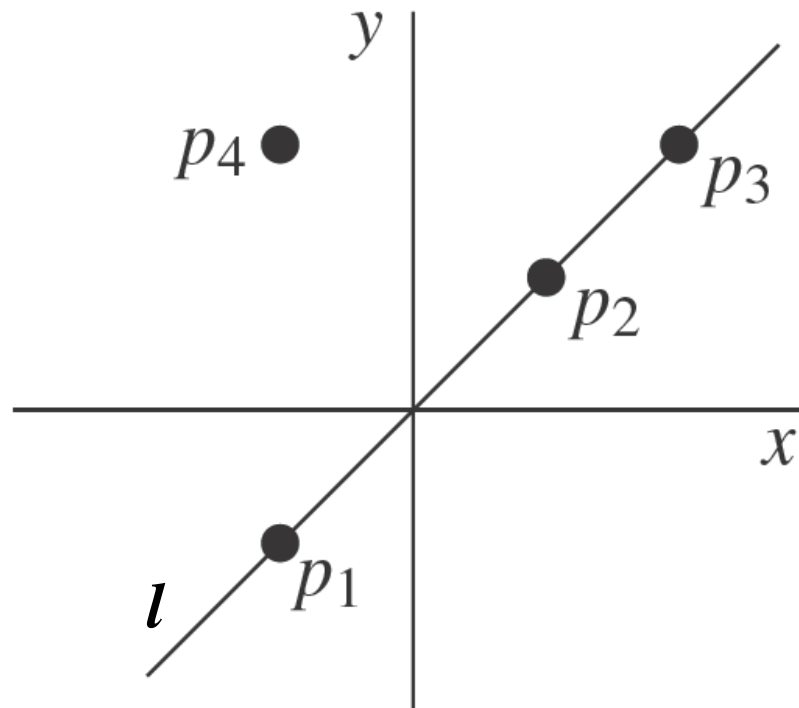
Point $l^* = [a, -b]$

[Berg]

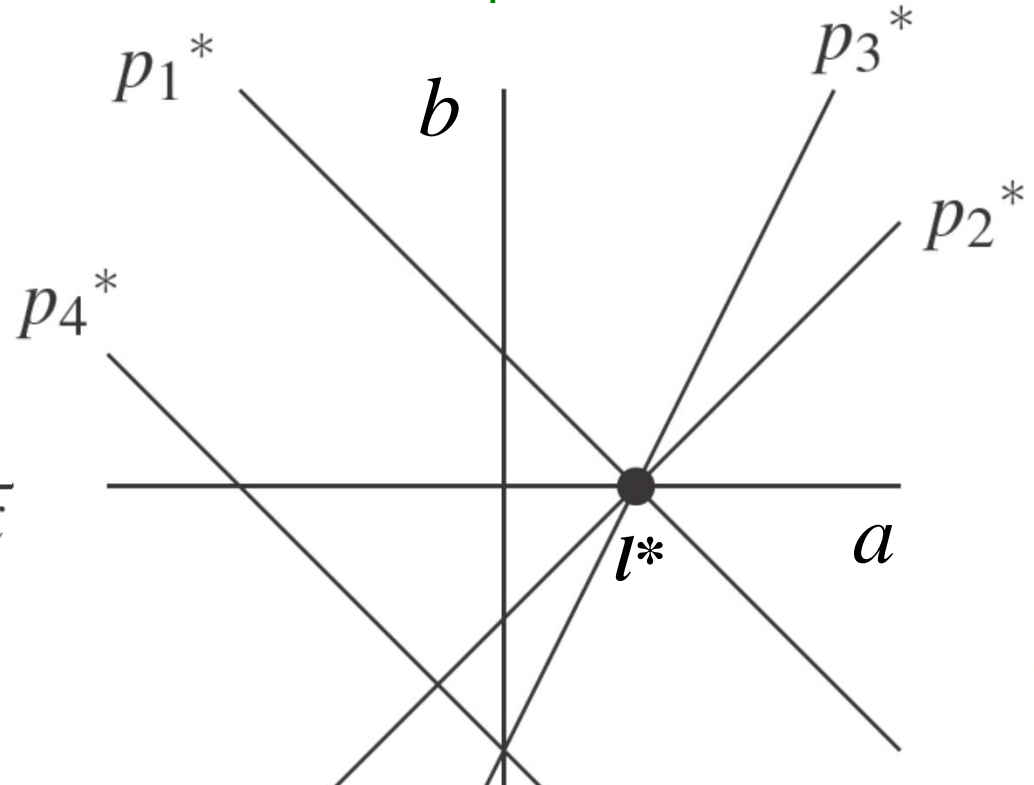


Duality of lines and points in the plane

Primal plane



Dual plane



point $p = [p_x, p_y]$

line $l := (y = ax + b)$

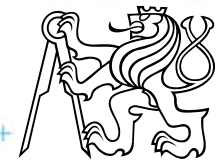
line $l := (y = ax - b)$

line $p^* := (b = p_x a - p_y)$

Point $l^* = [a, -b]$

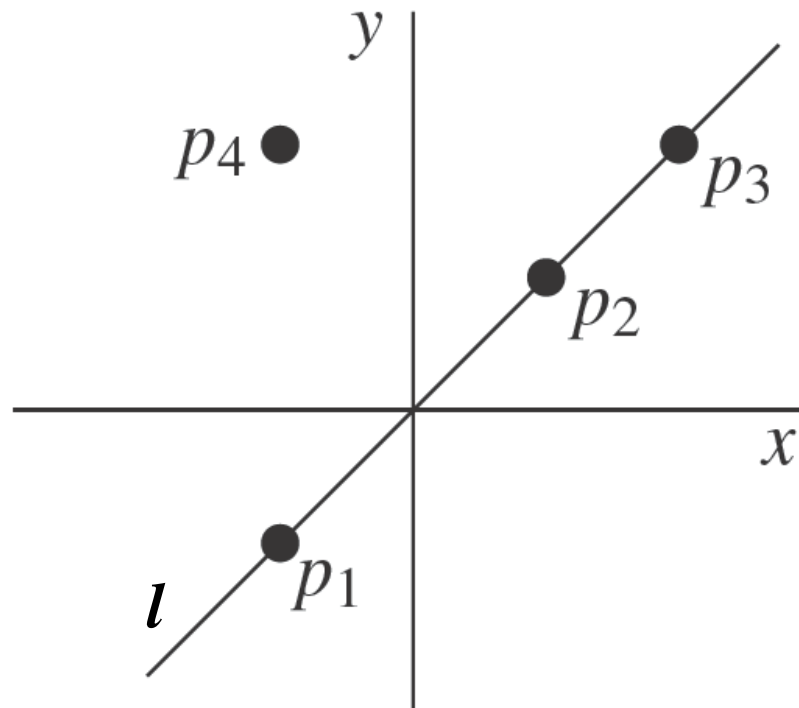
Point $l^* = [a, b]$

[Berg]

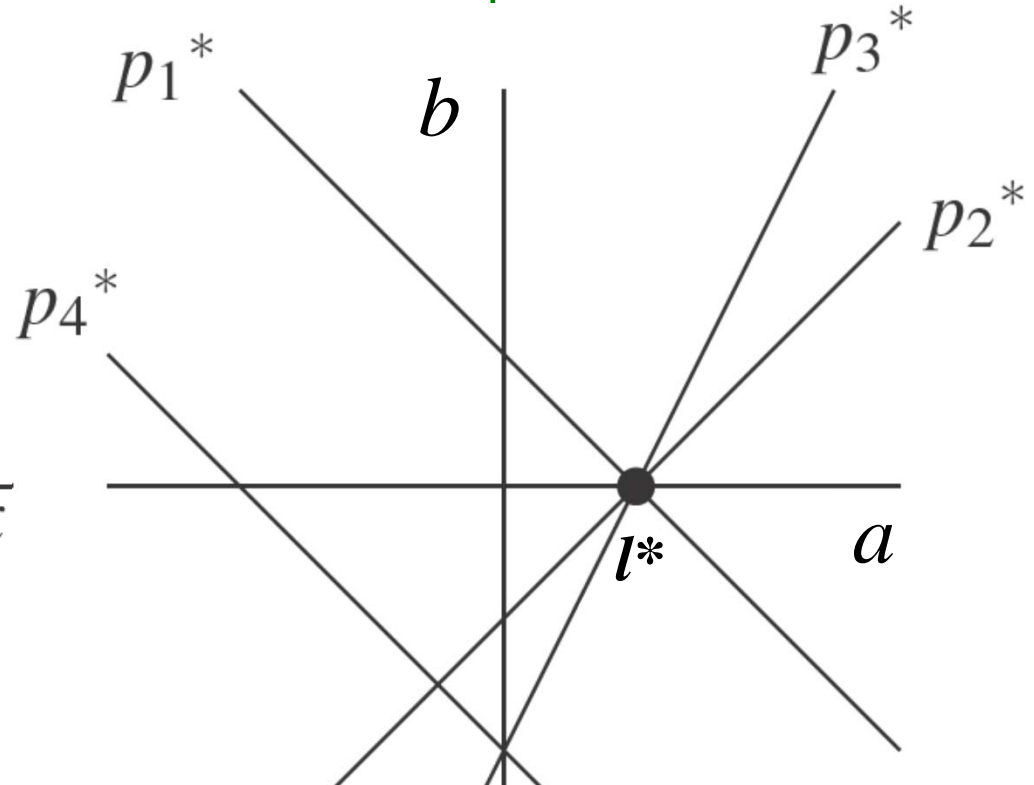


Duality of lines and points in the plane

Primal plane



Dual plane



point $p = [p_x, p_y]$

line $l := (y = ax + b)$

line $l := (y = ax - b)$

line $p^* := (b = p_x a - p_y)$

Point $l^* = [a, -b]$

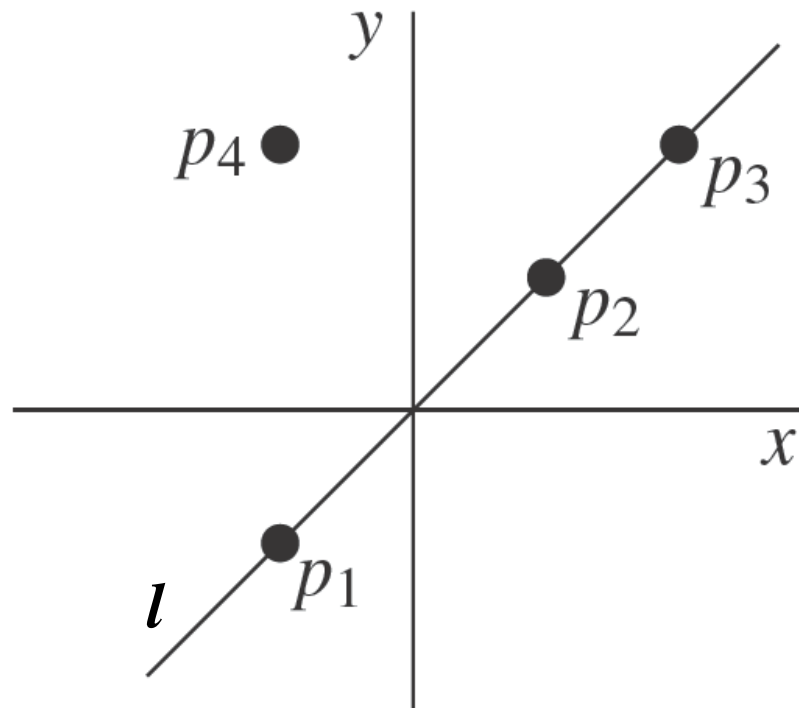
Point $l^* = [a, b]$

[Berg]

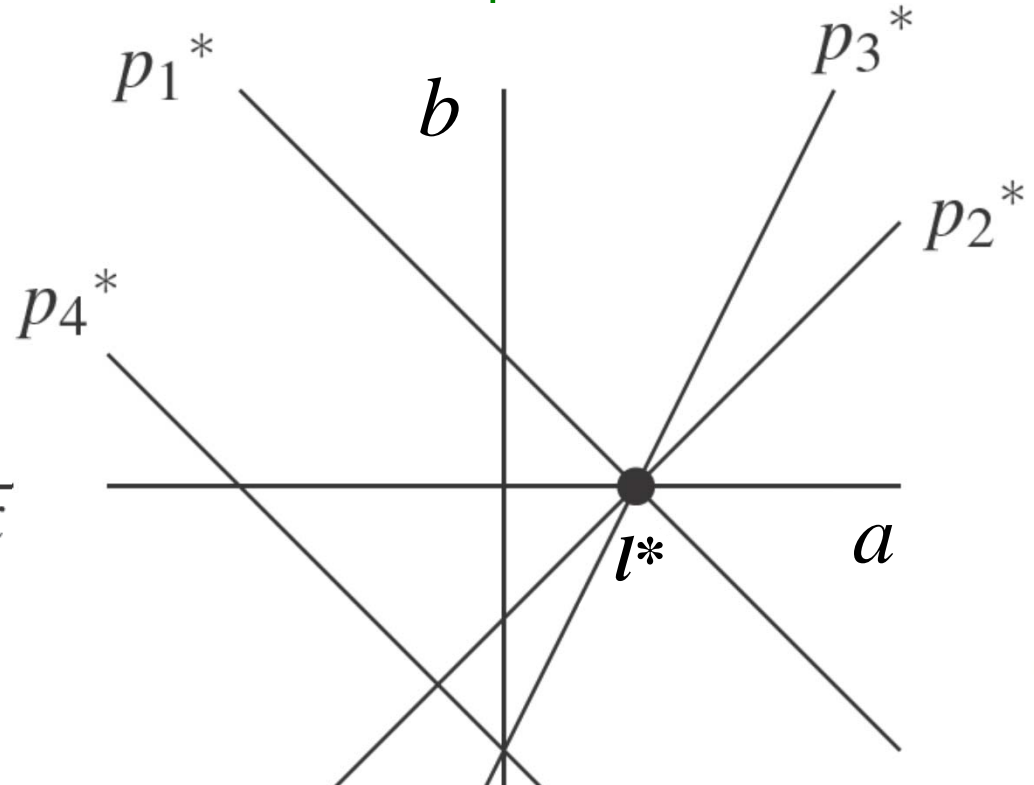


Duality of lines and points in the plane

Primal plane



Dual plane



point $p = [p_x, p_y]$

line $l := (y = ax + b)$

line $l := (y = ax - b)$

line $p^* := (b = p_x a - p_y)$

Point $l^* = [a, -b]$

Point $l^* = [a, b]$

[Berg]

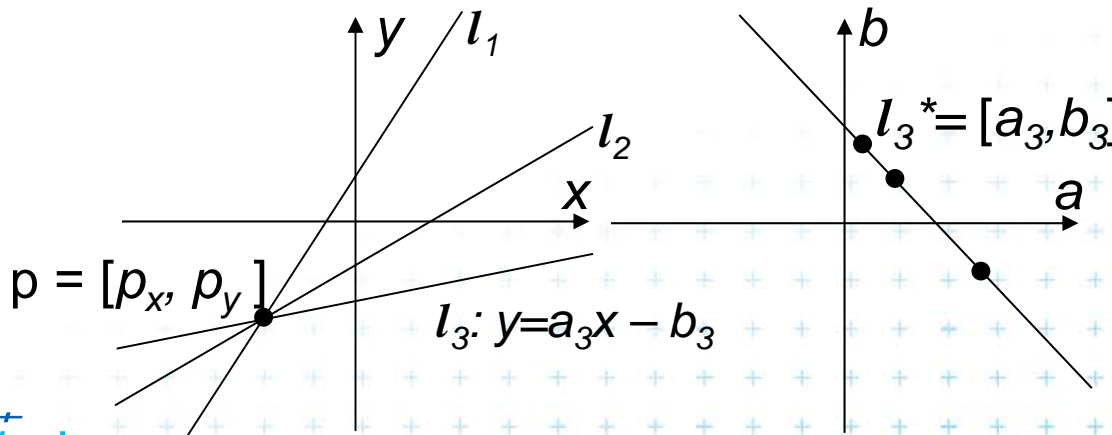
Same form => **It is convenient** to negate b in the line equation



Why is b negated in the line equation?

- In primal plane, consider
 - point $p = [p_x, p_y]$ and
 - set of non-vertical lines $l_i : y = a_i x - b_i$ passing through p satisfy the equation $p_y = a_i p_x - b_i$ (each line with different constants a_i, b_i)
- In dual plane, these lines transform to collinear points

$$\{ l_i^* = [a_i, b_i] : b_i = p_x a_i - p_y \}$$



Same form =>
 It is convenient to negate b in the line equation



If b not negated in the line equation...

Lines l_i have equation $l_i : y = a_i x - b_i$ OR $y = a_i x + b_i$

Passing through point $p = [p_x, p_y]$:

- With minus

- equation l_i : $p_y = a_i p_x - b_i$

- dual points $\{l_i^* = [a_i, b_i] : b_i = p_x a_i - p_y\}$... same form

- With plus

- equation l_i : $p_y = a_i p_x + b_i$

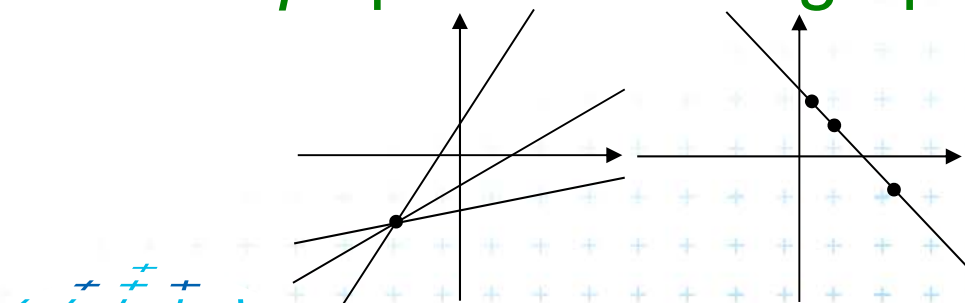
- dual $\{l_i^* = [a_i, b_i] : b_i = -p_x a_i + p_y\}$... different form



Properties of points and lines duality

Incidence is preserved

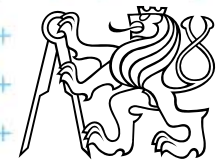
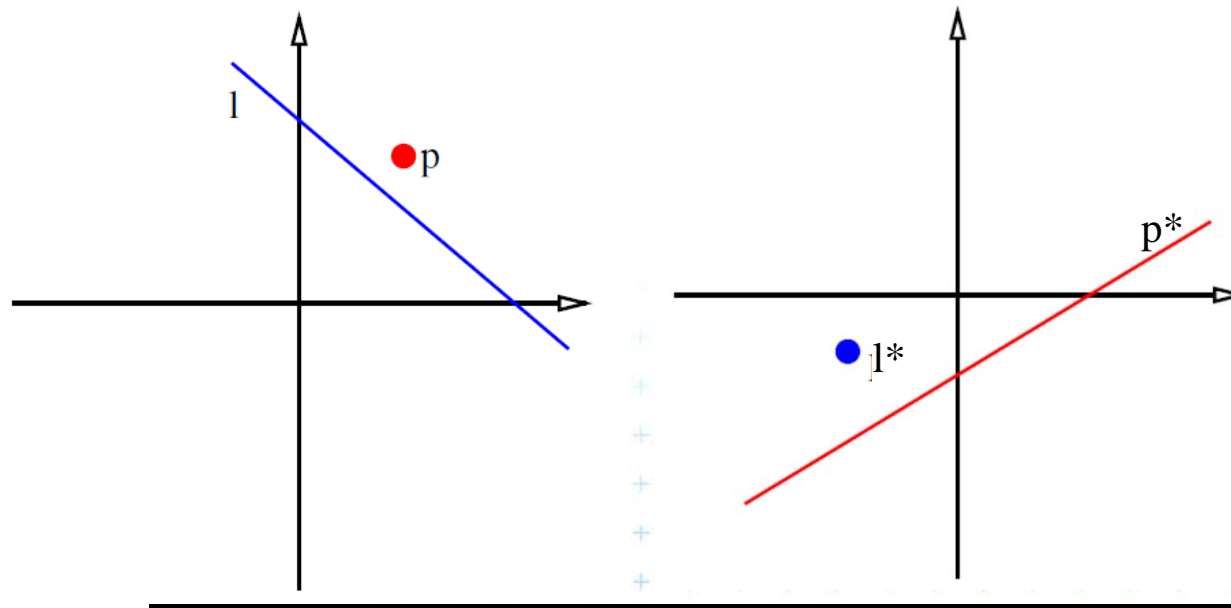
- Point p is incident to the line l in primal plane
iff
point l^* is incident to the line p^* in the dual plane.
- Lines l_1, l_2 intersects at point p
iff
line p^* passes through points l_1^*, l_2^* .



Properties of points and lines duality

But **order is reversed**

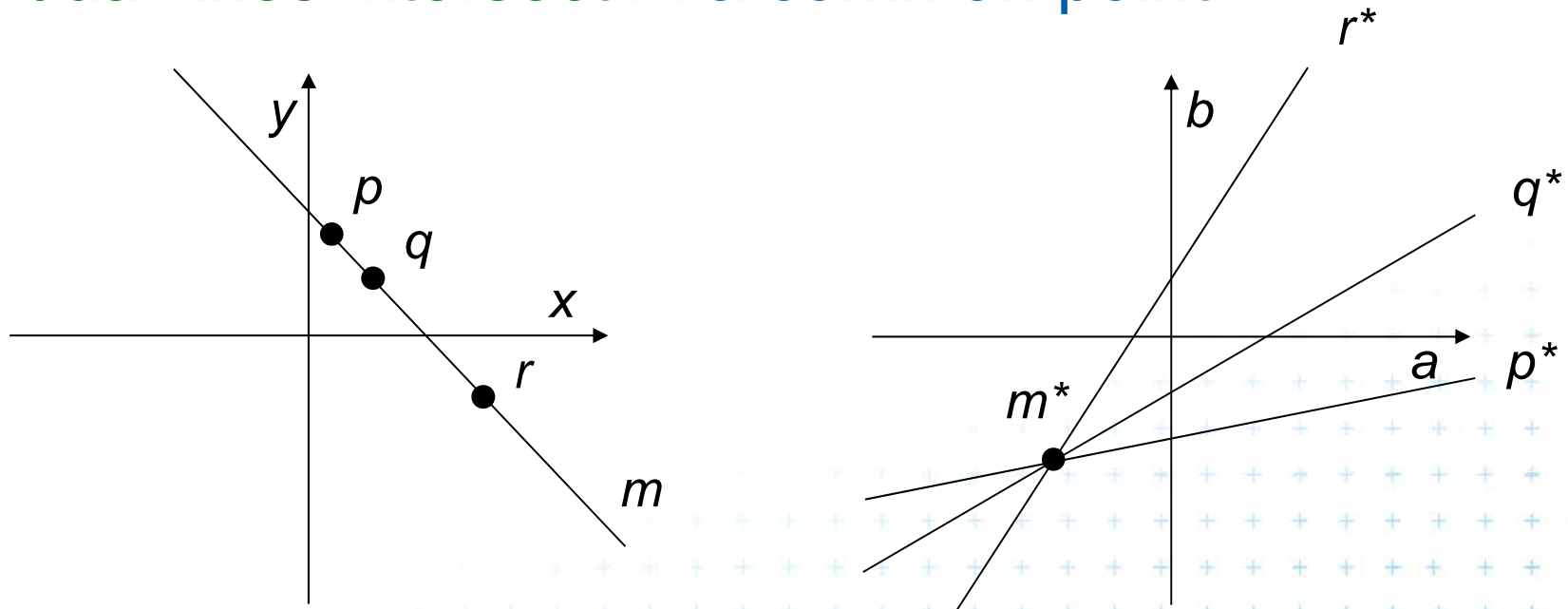
- Point p lies **above (below)** line l in the primal plane **iff** line p^* passes **below (above)** point l^* in the dual plane Or said order is preserved: ... **iff** Point l^* lies **above (below)** line p^*



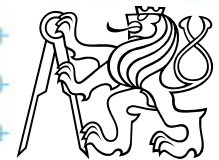
Properties of points and lines duality

Collinearity

- Points are **collinear** in the primal plane **iff** their dual lines intersect in a common point

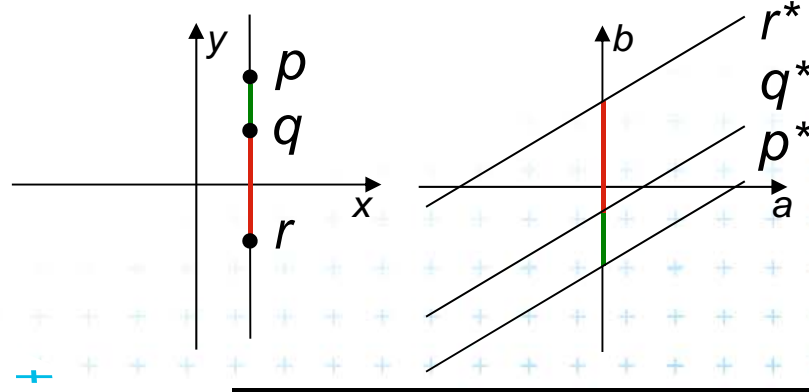


- This does not hold for points on vertical line



Handling of vertical lines

- Dual transform is undefined for vertical lines
 - Points with the same x coordinate dualize to lines with the same slope (parallel lines) and therefore
 - These dual lines do not intersect (as should for collinear points)
 - **Vertical line** through these points **does not dualize to an intersection point**
 - For detection of vertically collinear points use other method - $O(n)$ vertical lines $\rightarrow O(n^2)$ brute force 3|| lines s.



$\rightarrow O(n)$ after $O(n \log n)$ sorting by x

Vertical distances of such duals are "preserved". For $p_x = q_x$

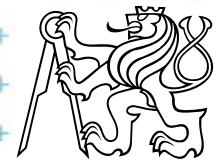
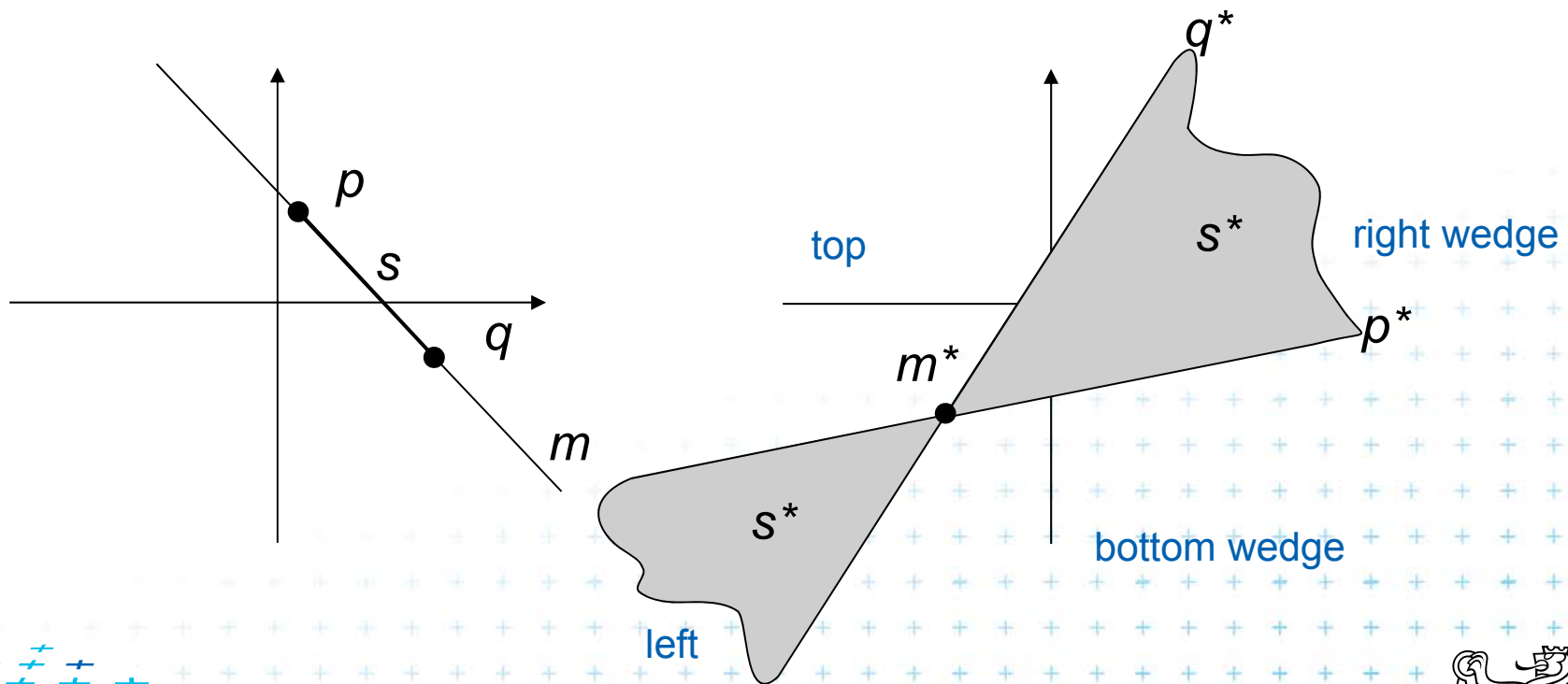
$$\text{vertDist}(q^*_b, p^*_b) = p_y - q_y$$



2. Duality of line segments

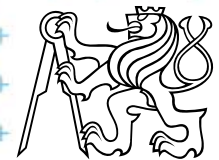
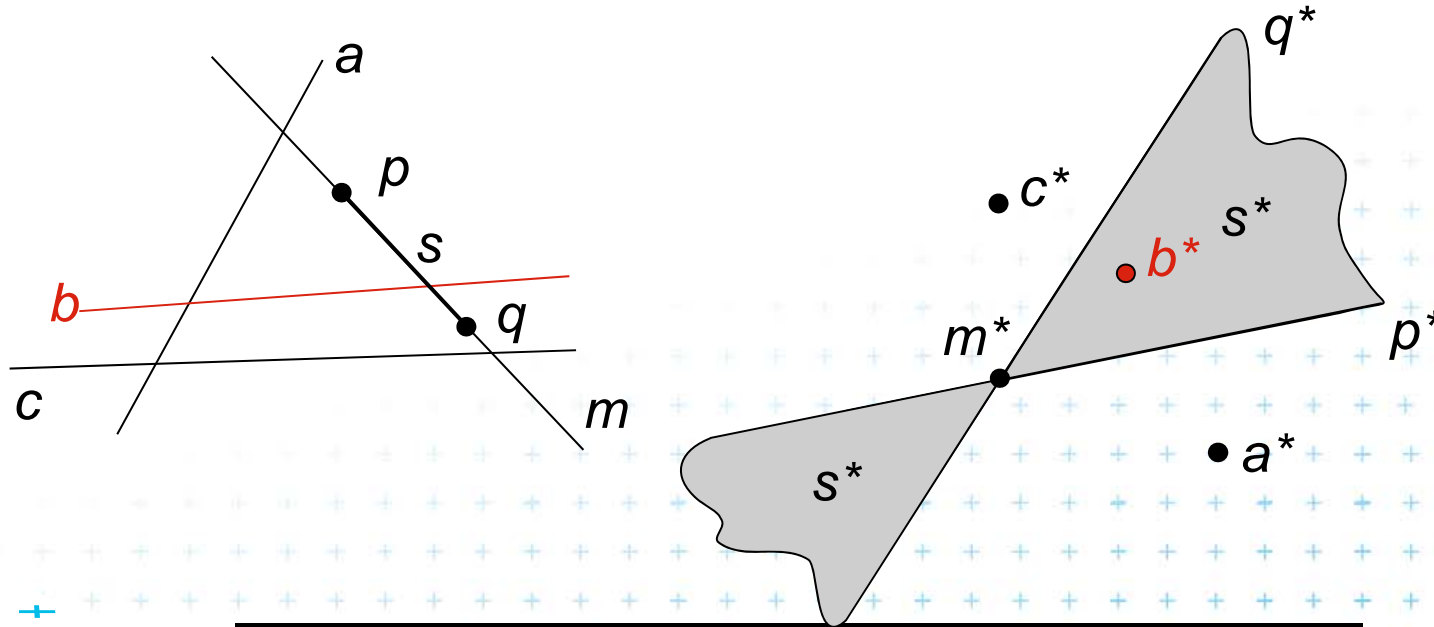
- Line segment s

- = set of collinear points $\xrightarrow{\text{dual}}$ set of lines passing one point
- union of these lines is a (left-right) **double wedge** s^*



Intersection of line and line segment

- Line b intersects line segment s
 - if point b^* lays in the double wedge s^* ,
i.e., between the duals p^*, q^* of segment endpoints p, q
 - point p lies above line b and q lies below line b
 - point b^* lies above line p^* and b^* lies below line q^*



3. Polar duality (Polarity)

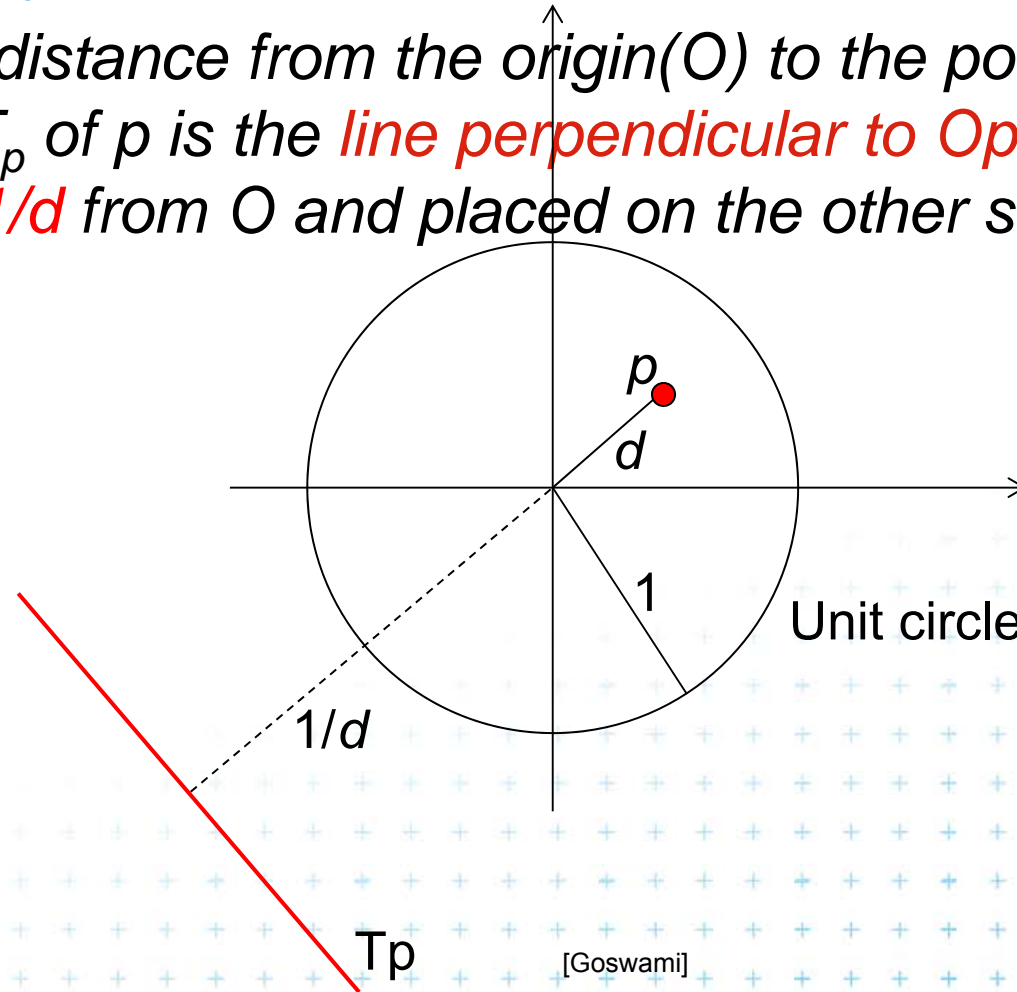
- Another example of **point-line duality**
- In 2D: Point $p = (p_x, p_y)$ in the primal plane corresponds to a line T_p with equation $ax + by = 1$ in the dual plane and vice versa $p_x x + p_y y = 1$
- In dD: Point p is taken as a radius-vector (starts in origin O). The **dot product** $(p \cdot x) = 1$ defines a **polar hyperplane** $p^* = T_p = \{ x \in R^d : (p \cdot x) = 1 \}$
- Used in theory of polytopes



Polar duality (Polarity)

- Geometrically in 2D, this means that

- if d is the distance from the origin(O) to the point p , the dual T_p of p is the **line perpendicular to Op** at distance $1/d$ from O and placed on the other side of O .



4. Convex hull using duality – definitions

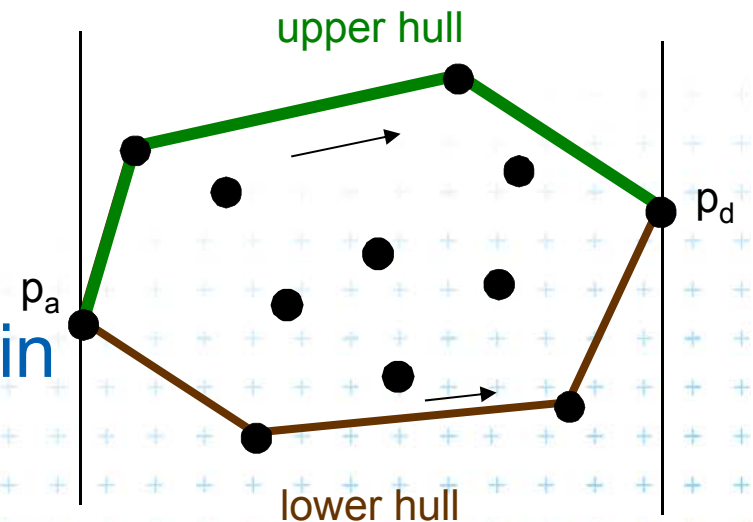
- An optimal algorithm
 - Let P be the given set of n points in the plane.
 - Let $p_a \in P$ be the point with smallest x-coordinate
 - Let $p_d \in P$ be the point with largest x-coordinate
- Both p_a and $p_d \in CH(P)$

Upper hull = CW polygonal chain

p_a, \dots, p_d along the hull

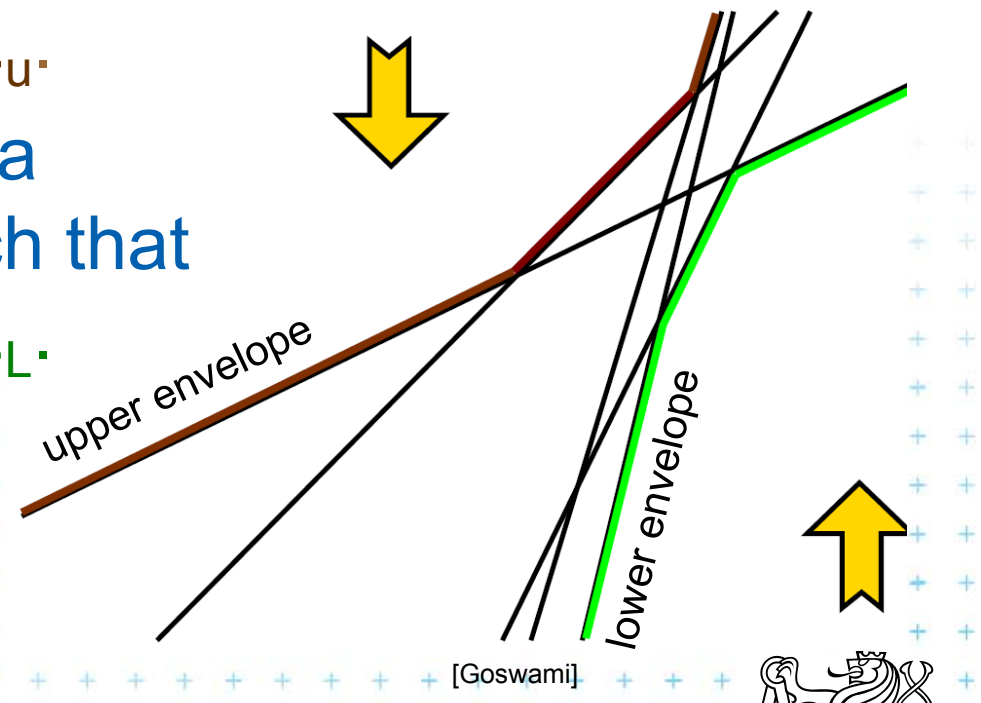
Lower hull = CCW polygonal chain

p_a, \dots, p_d along the hull

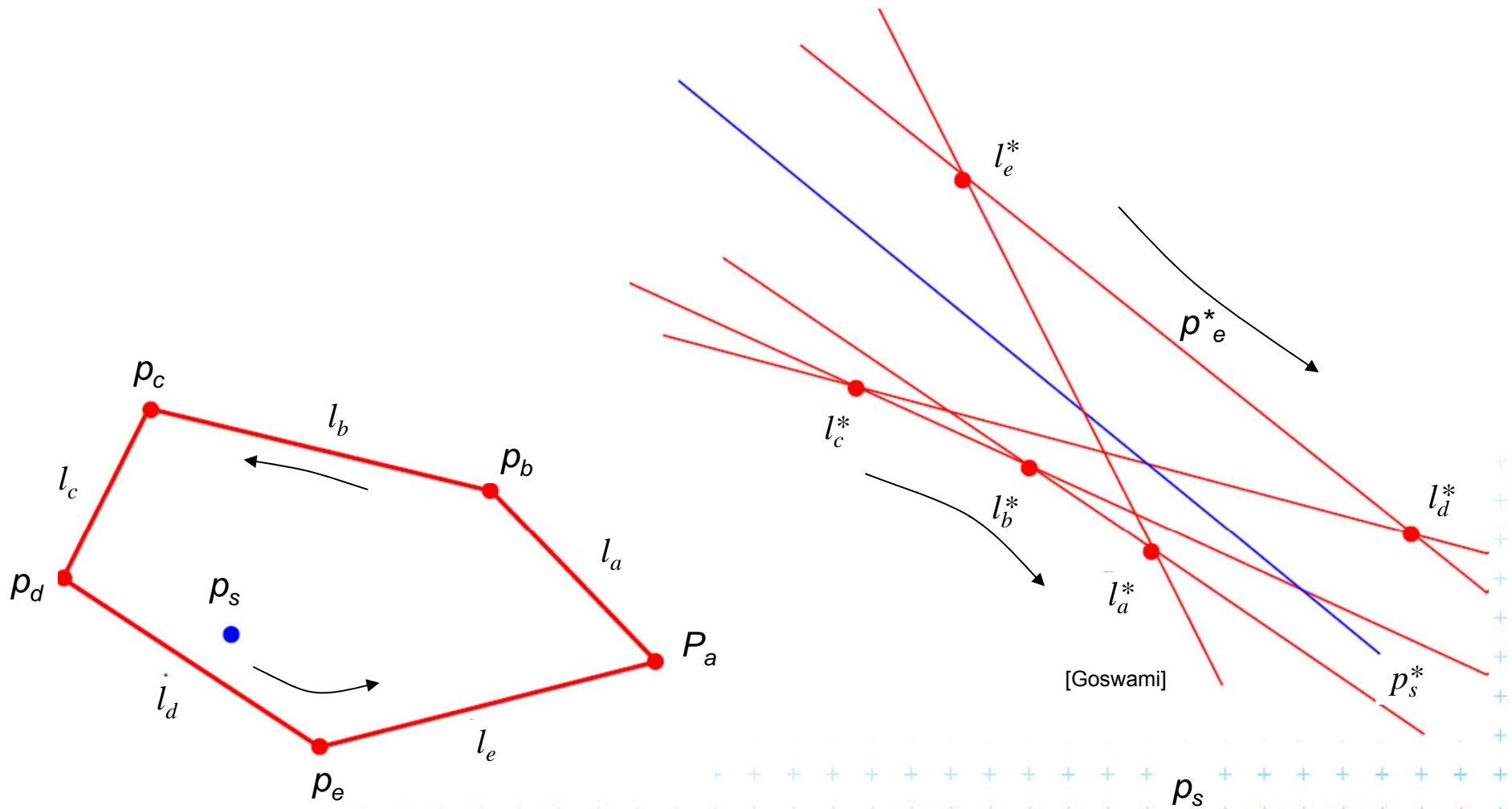


Definitions

- Let L be a set of lines in the plane
- The upper envelope is a polygonal chain E_u such that no line $l \in L$ is above E_u .
- The lower envelope is a polygonal chain E_L such that no line $l \in L$ is below E_L .

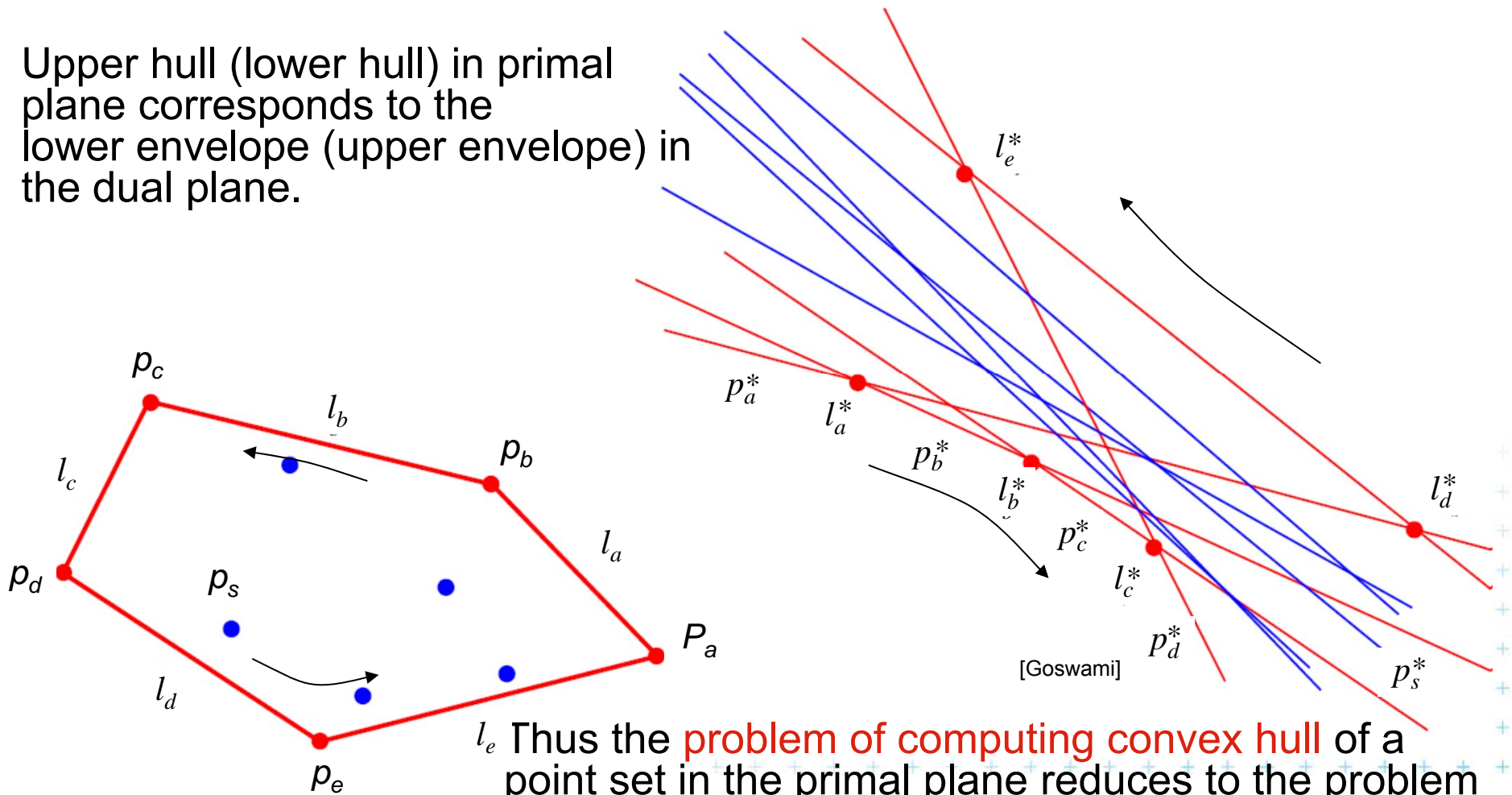


Connection between Hull and Envelope



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

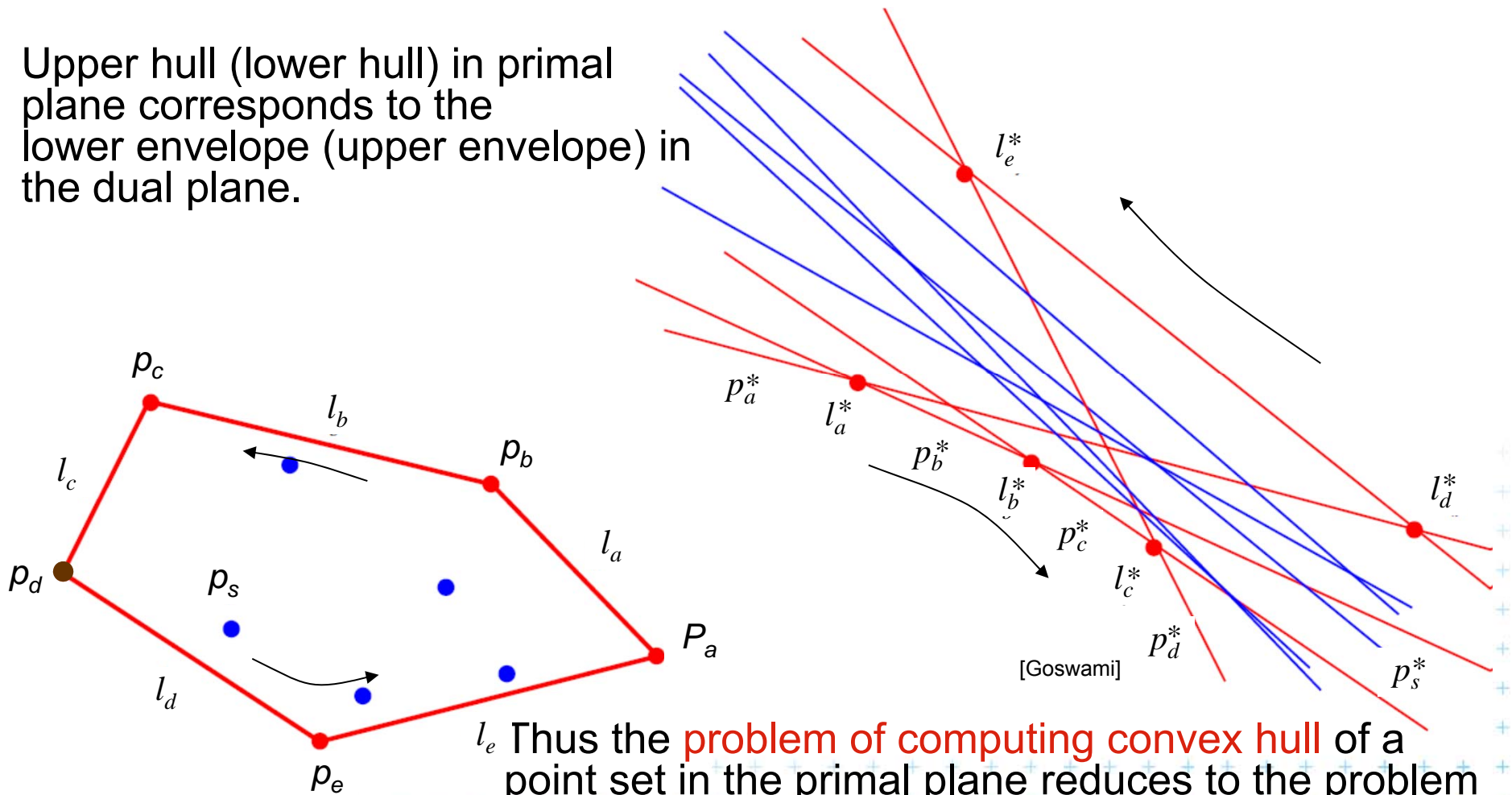


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

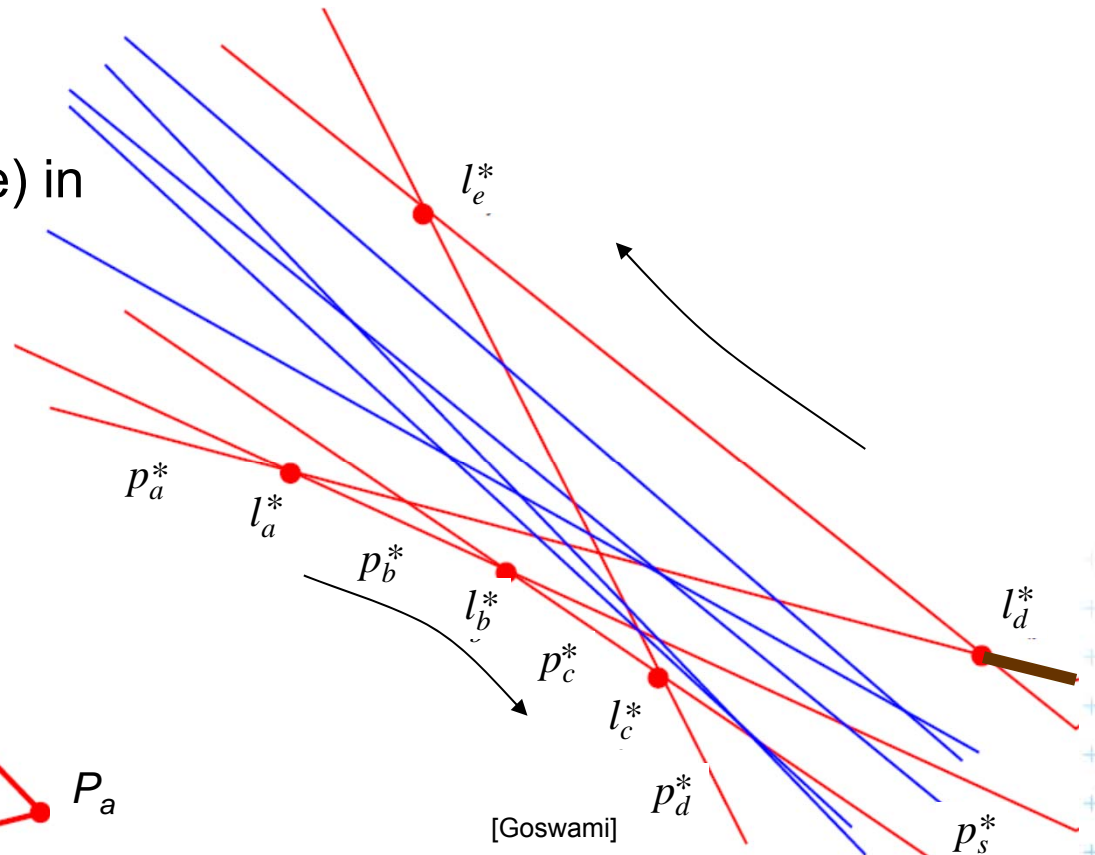
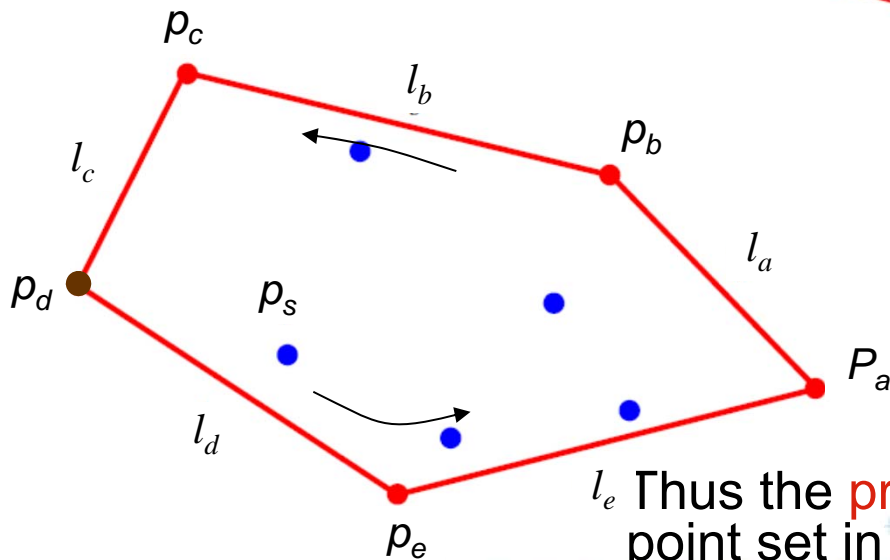


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

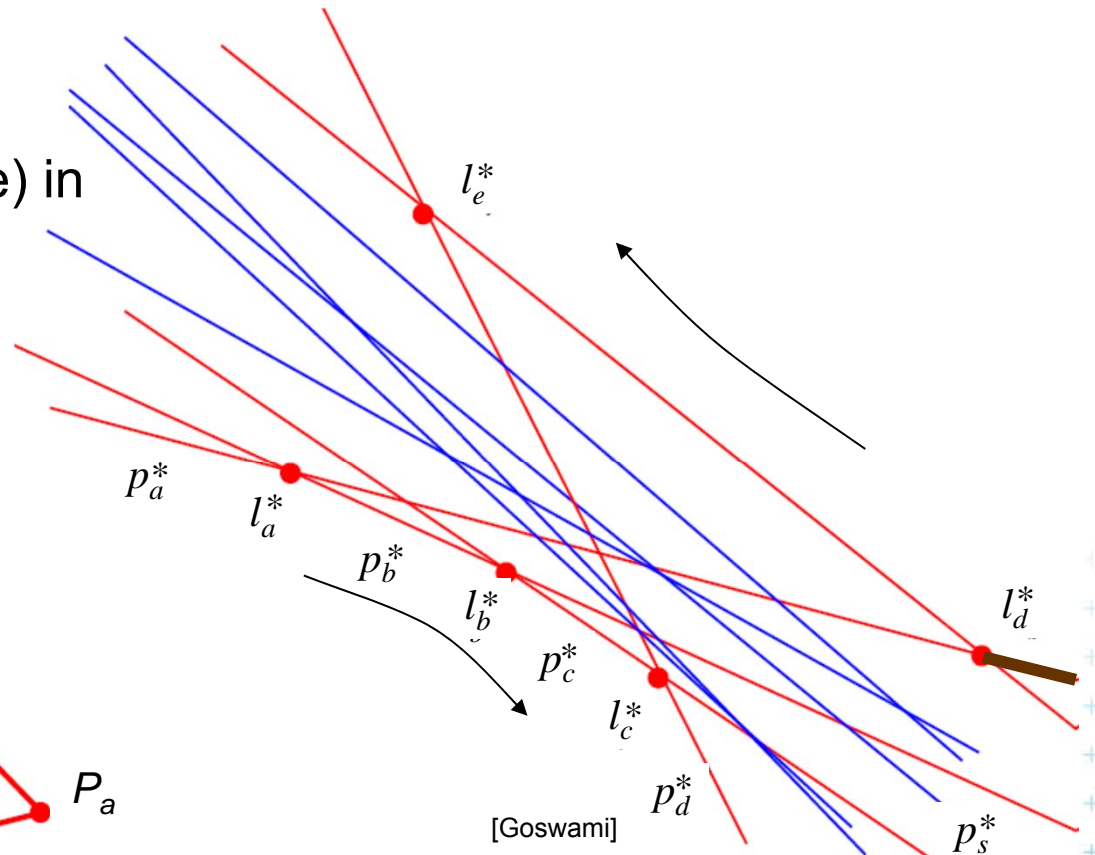
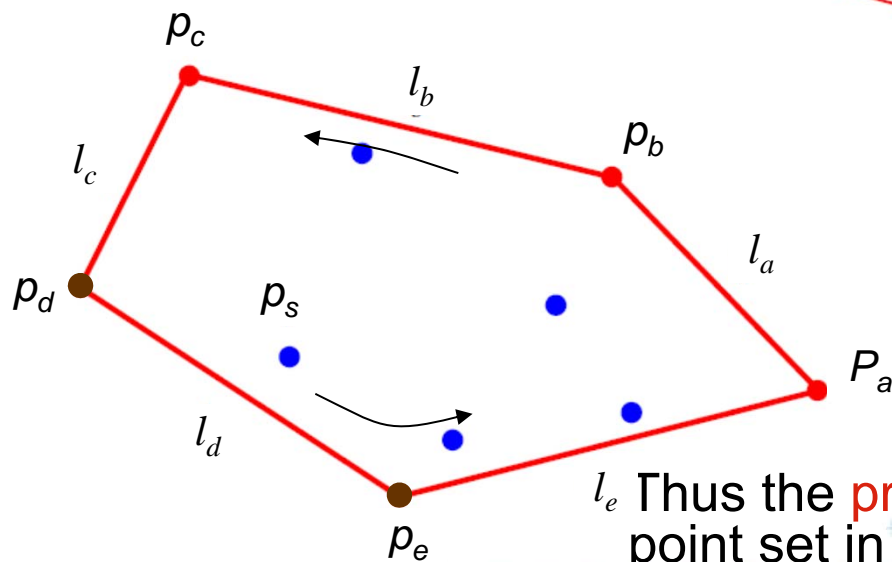


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

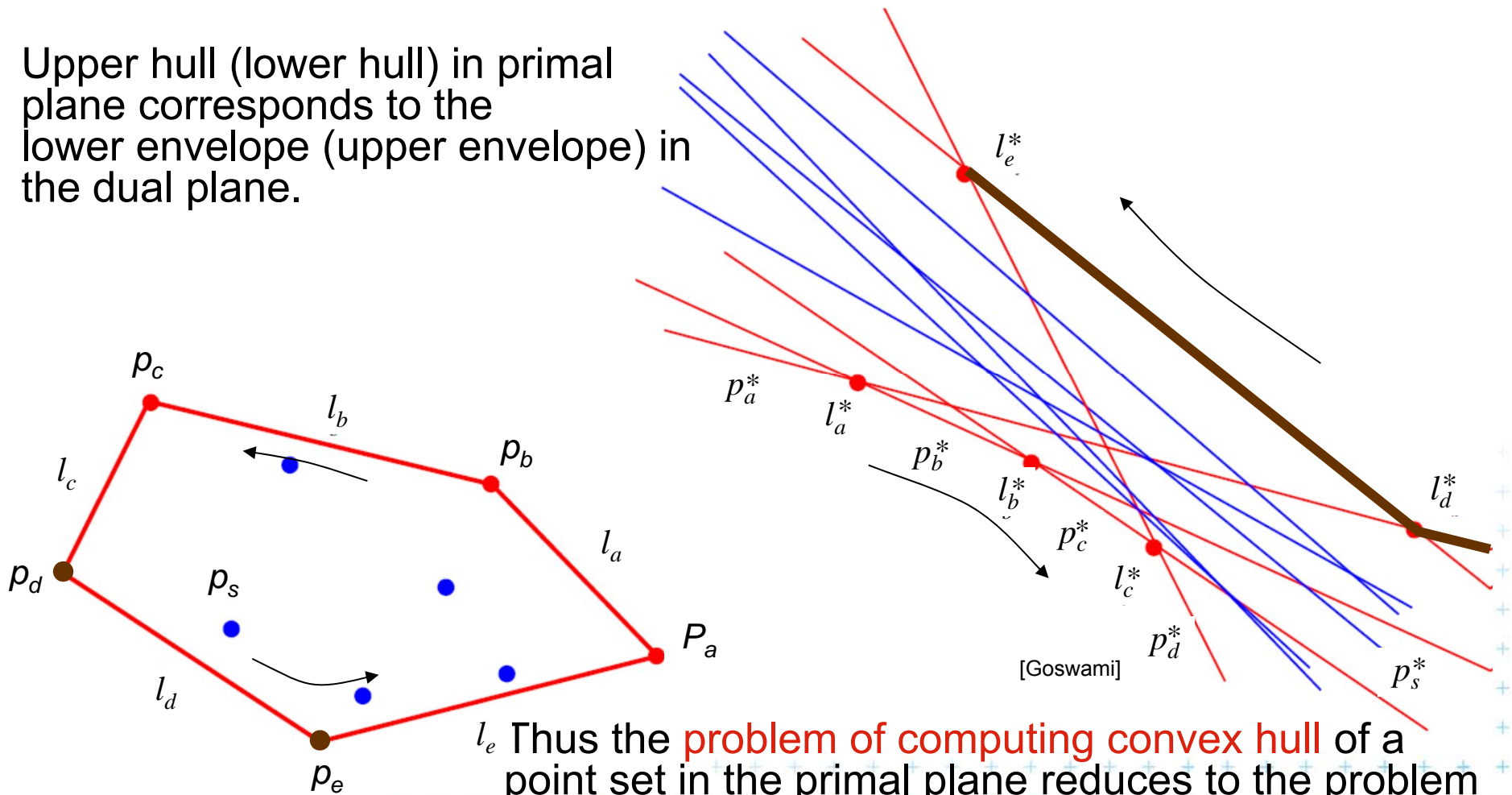


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



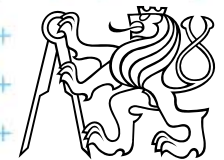
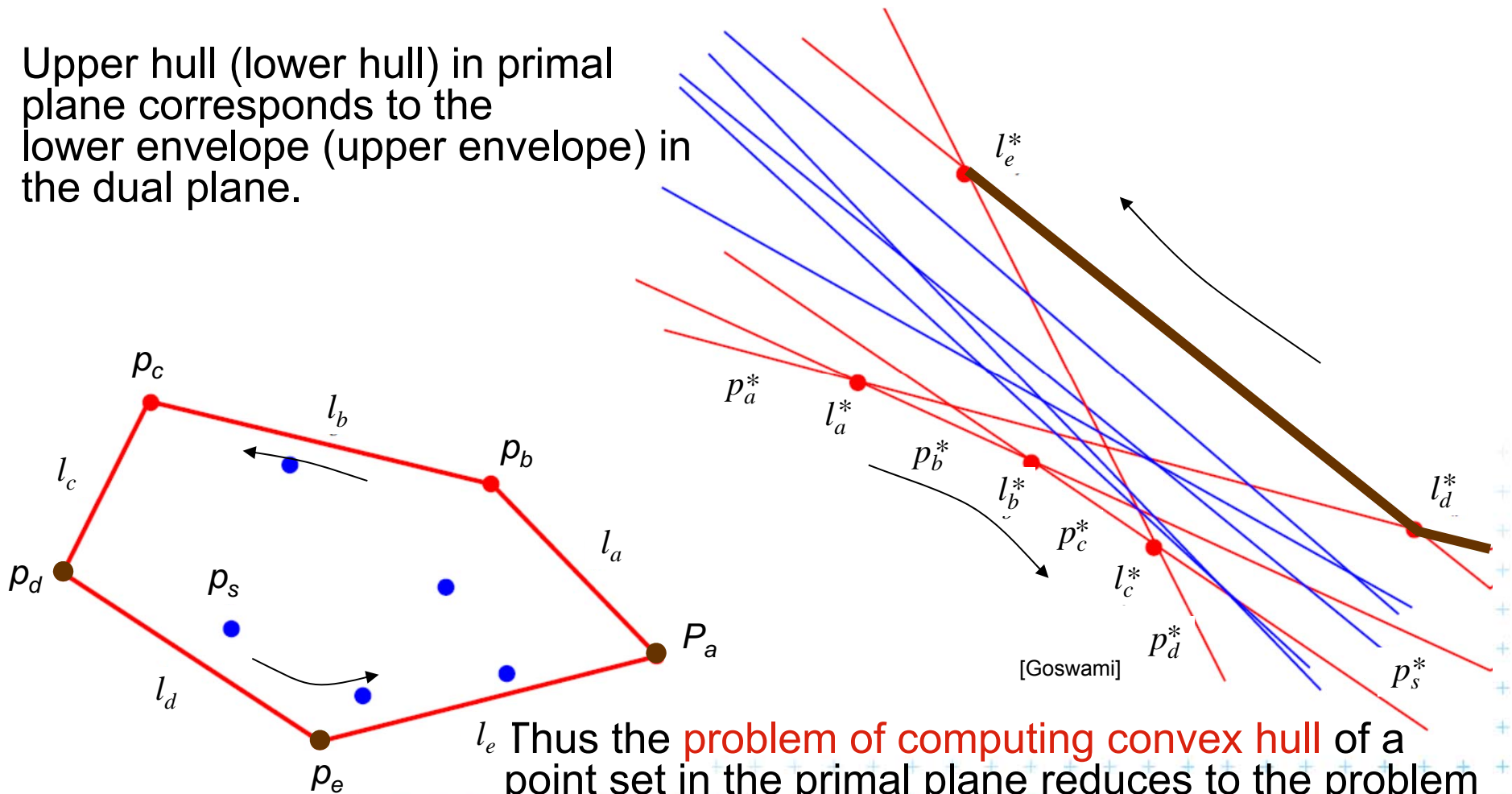
Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.



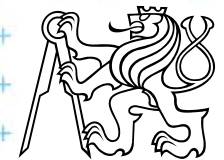
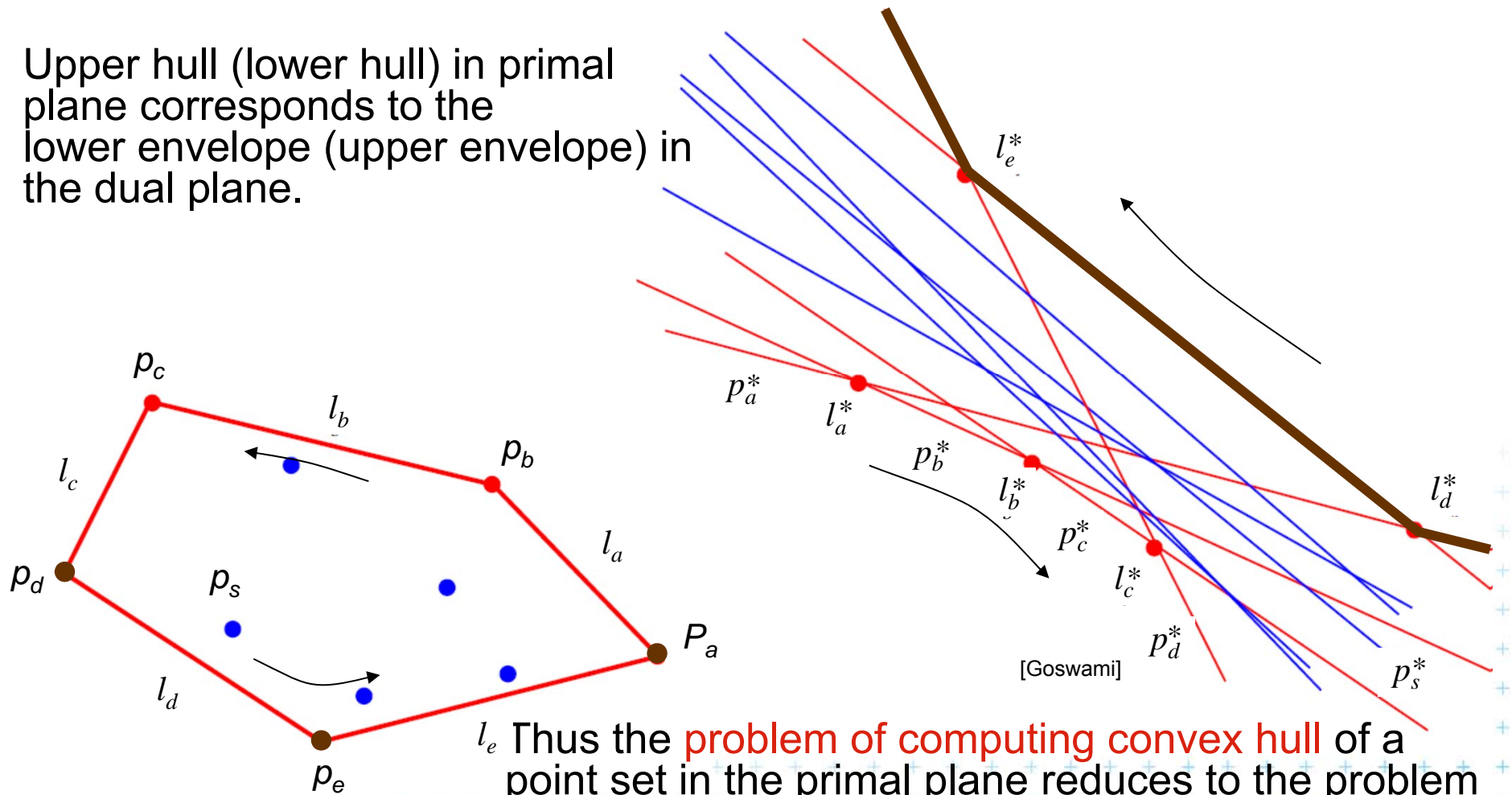
Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.



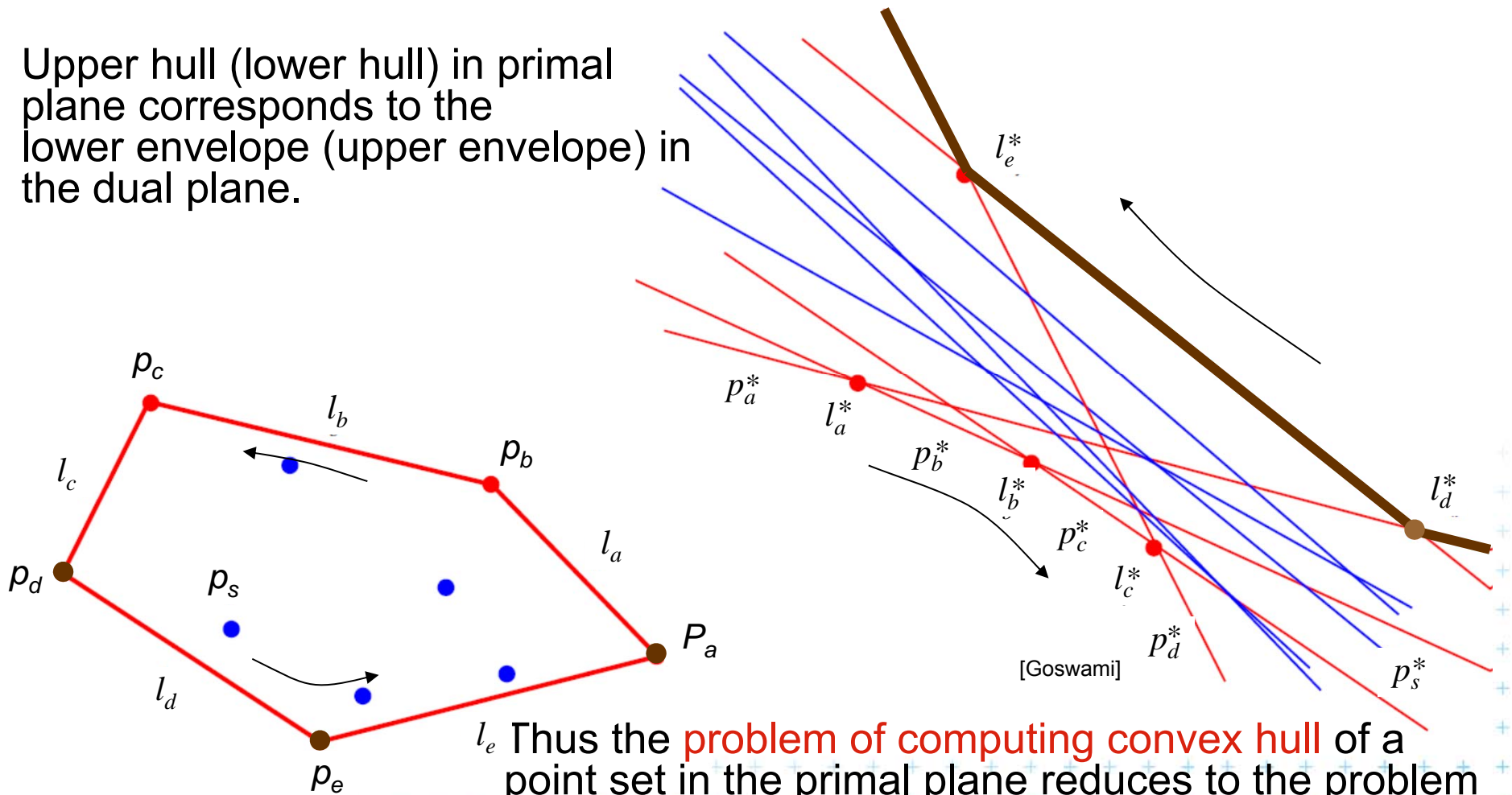
Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.



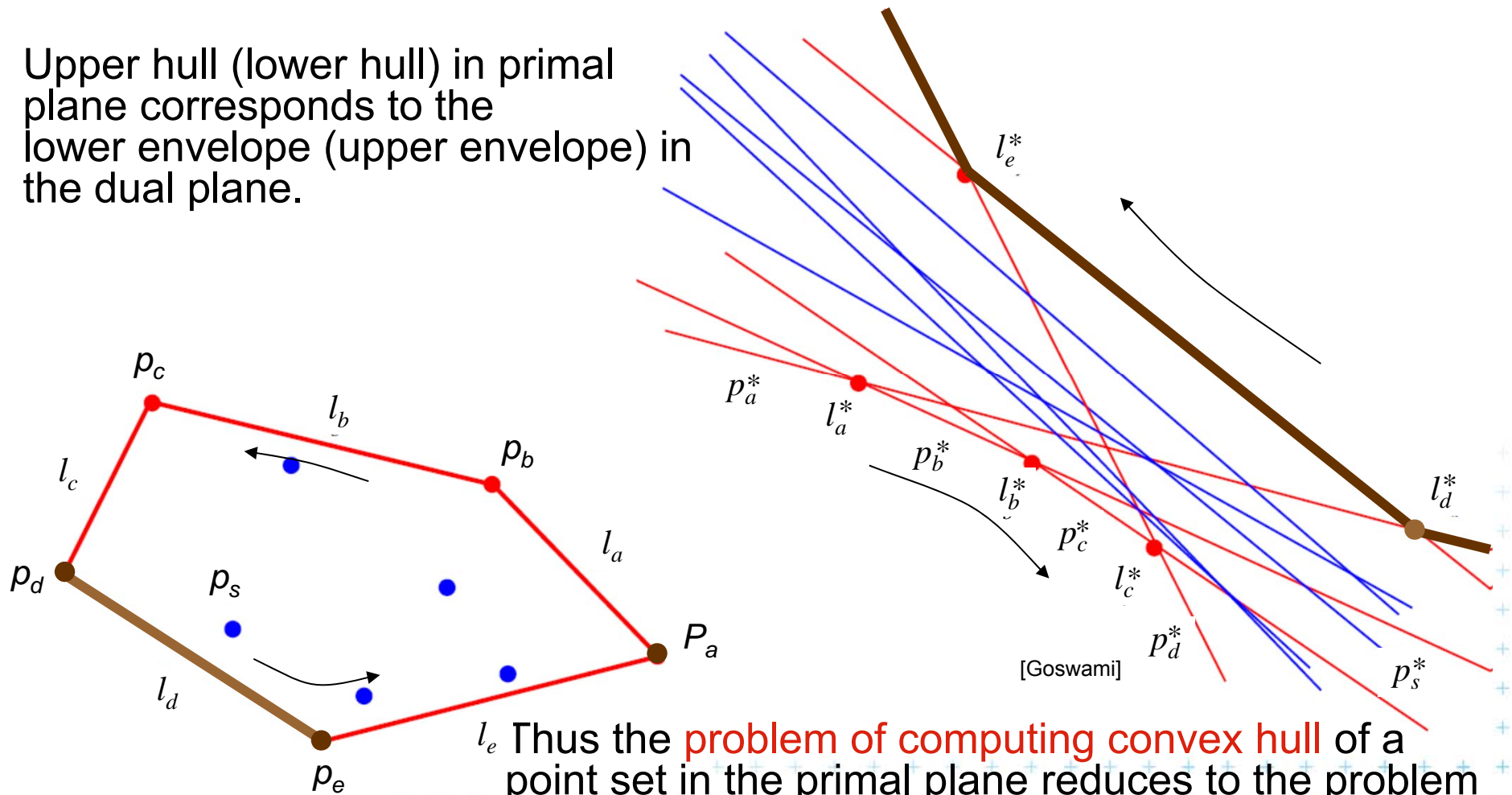
Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

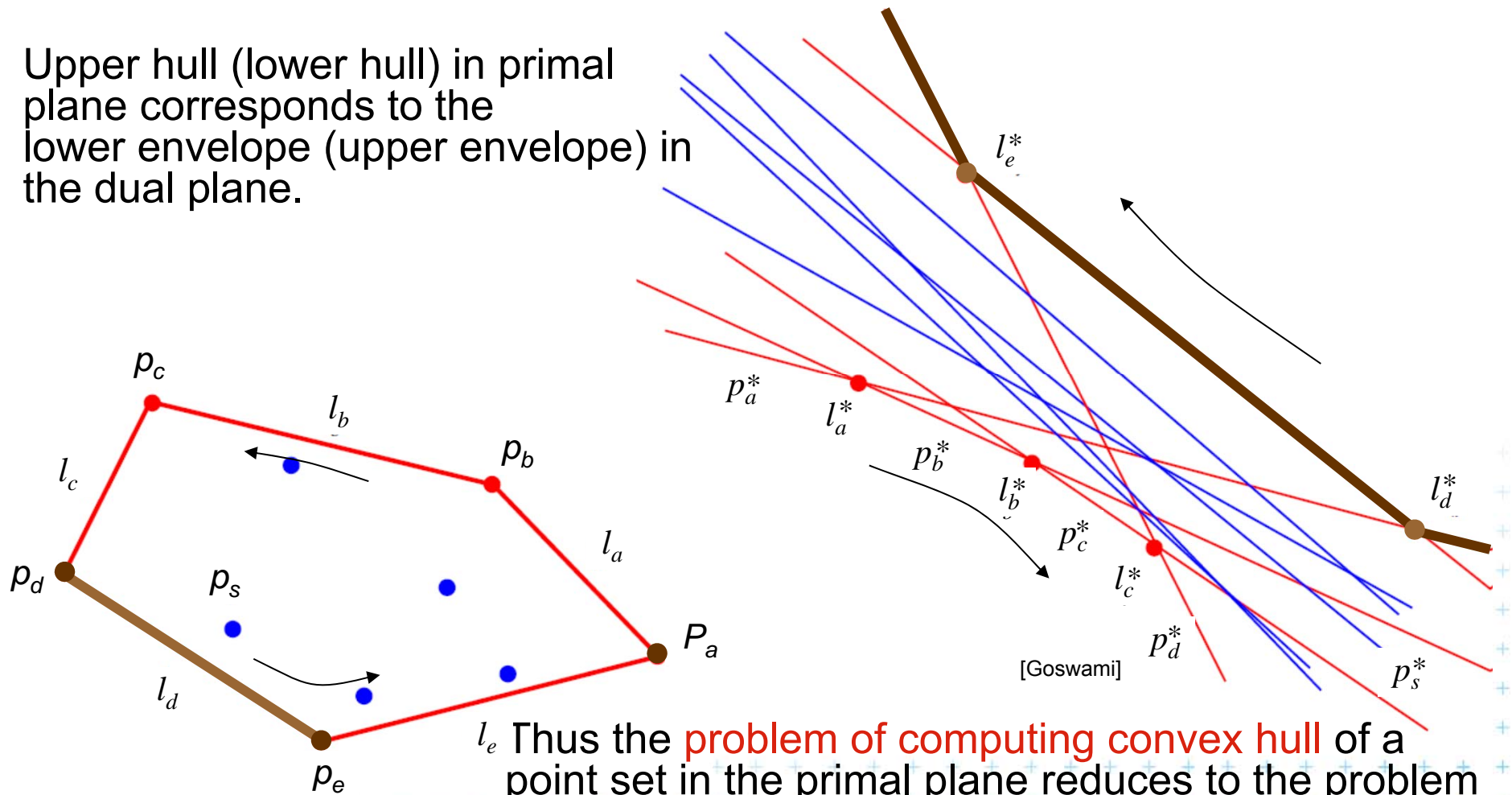


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

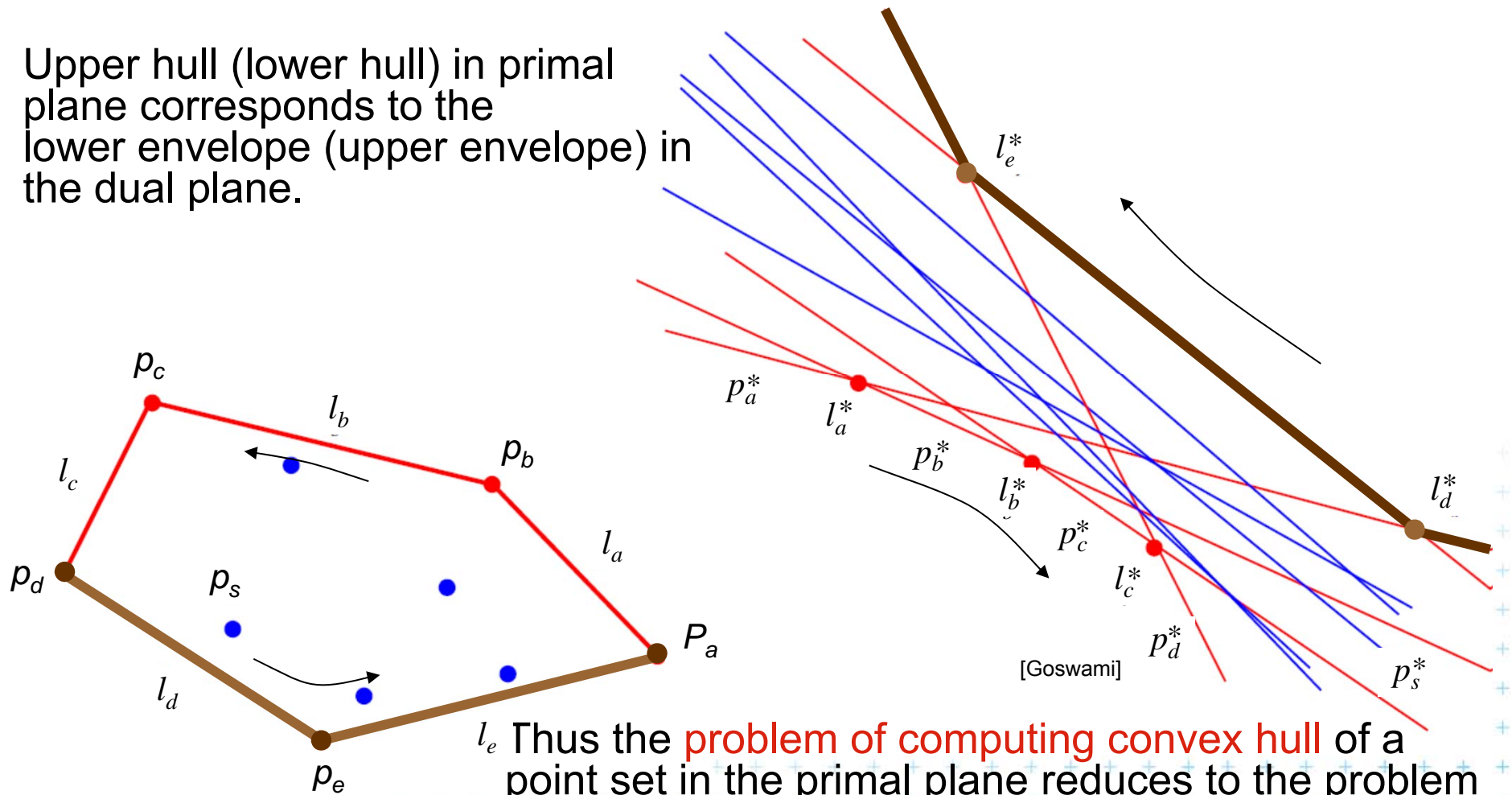


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

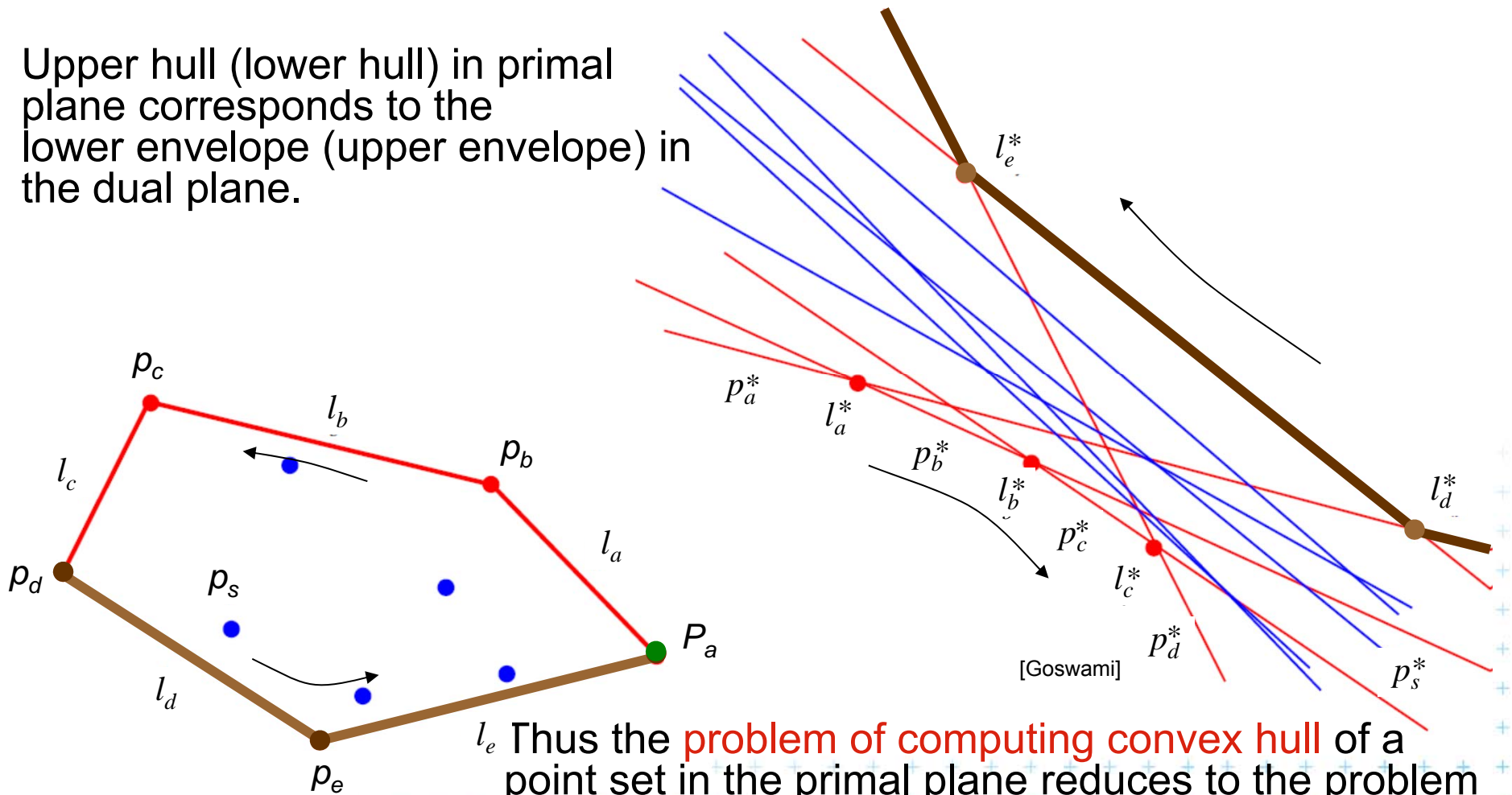


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

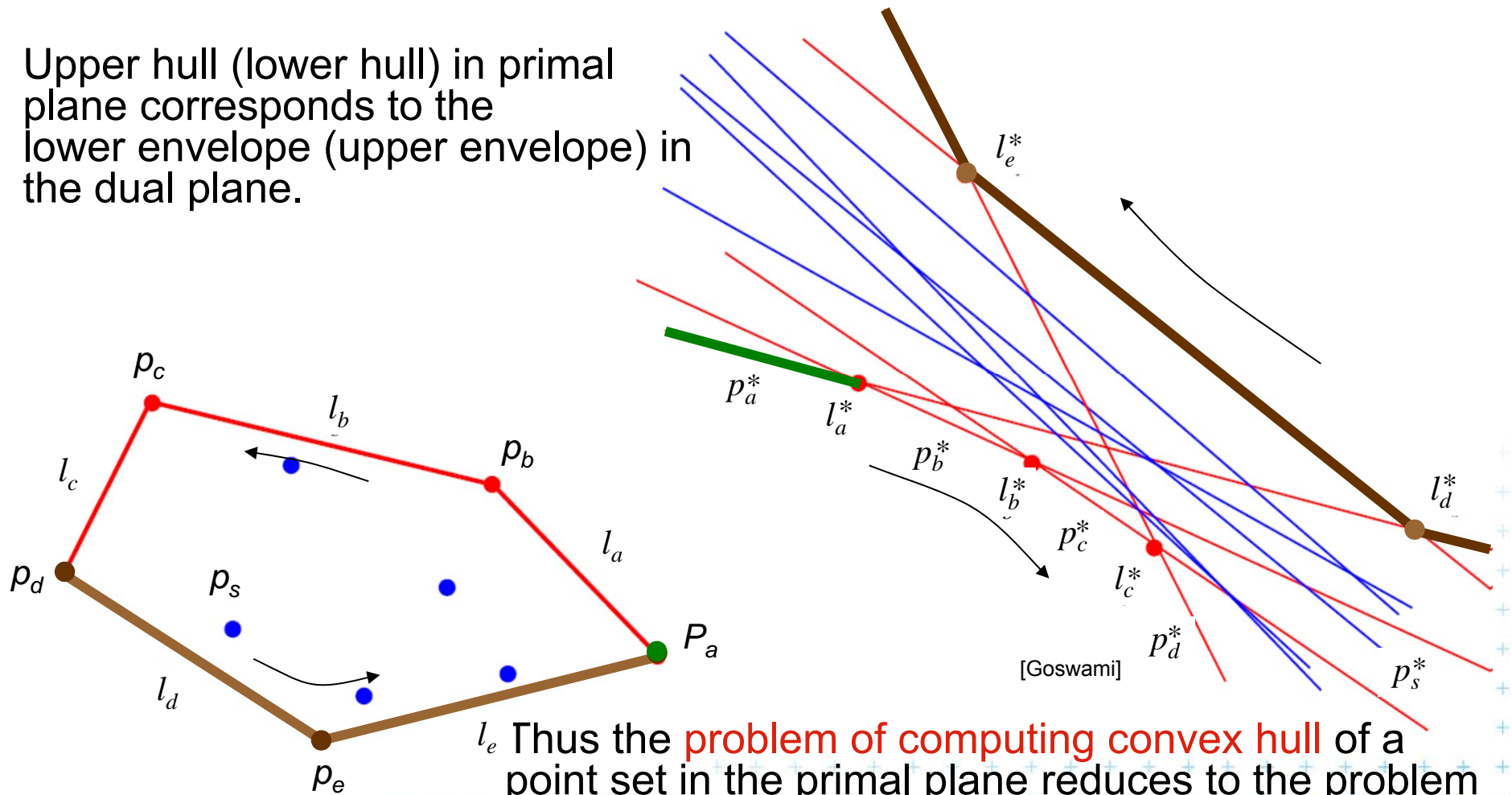


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

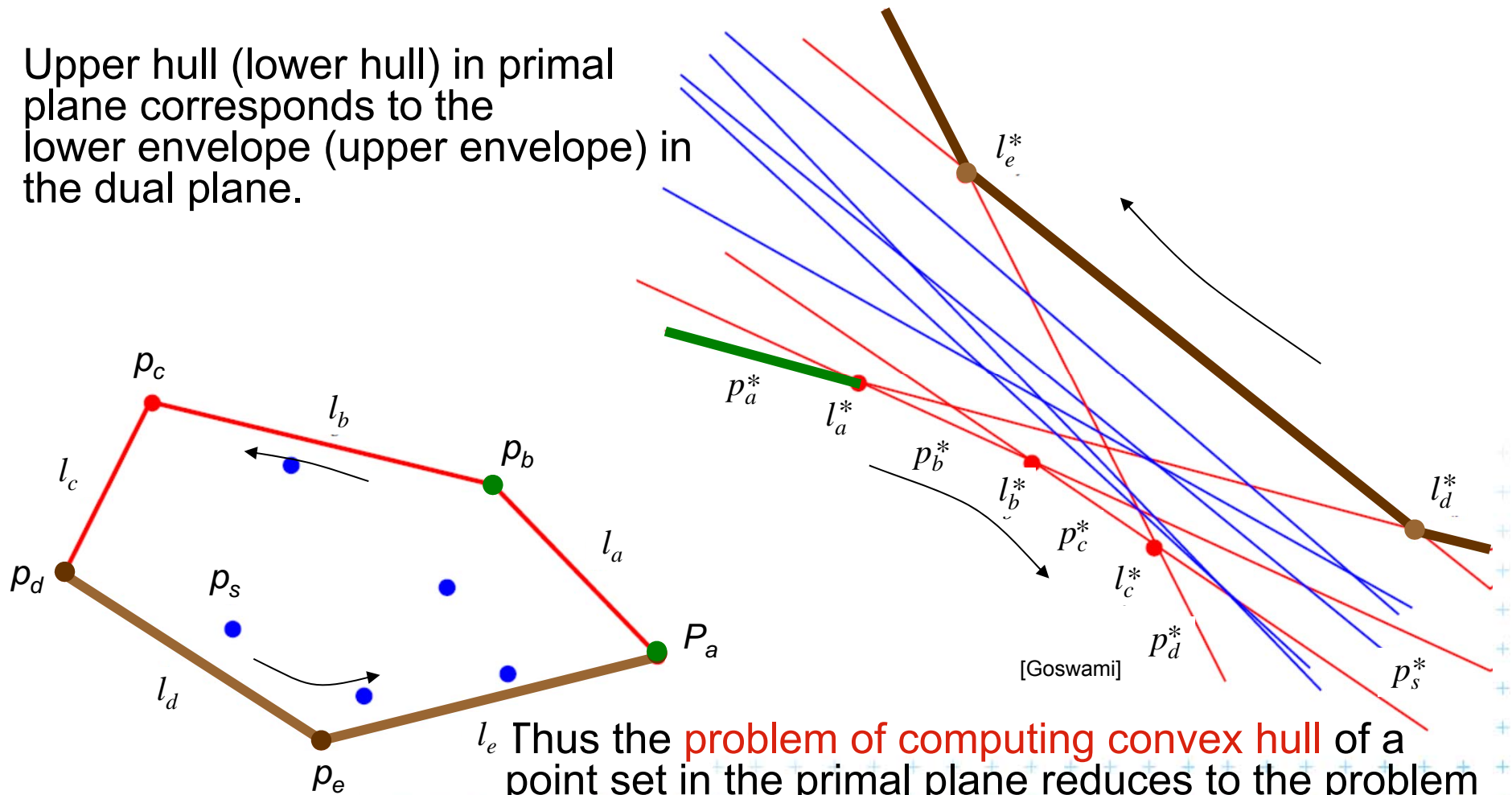


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

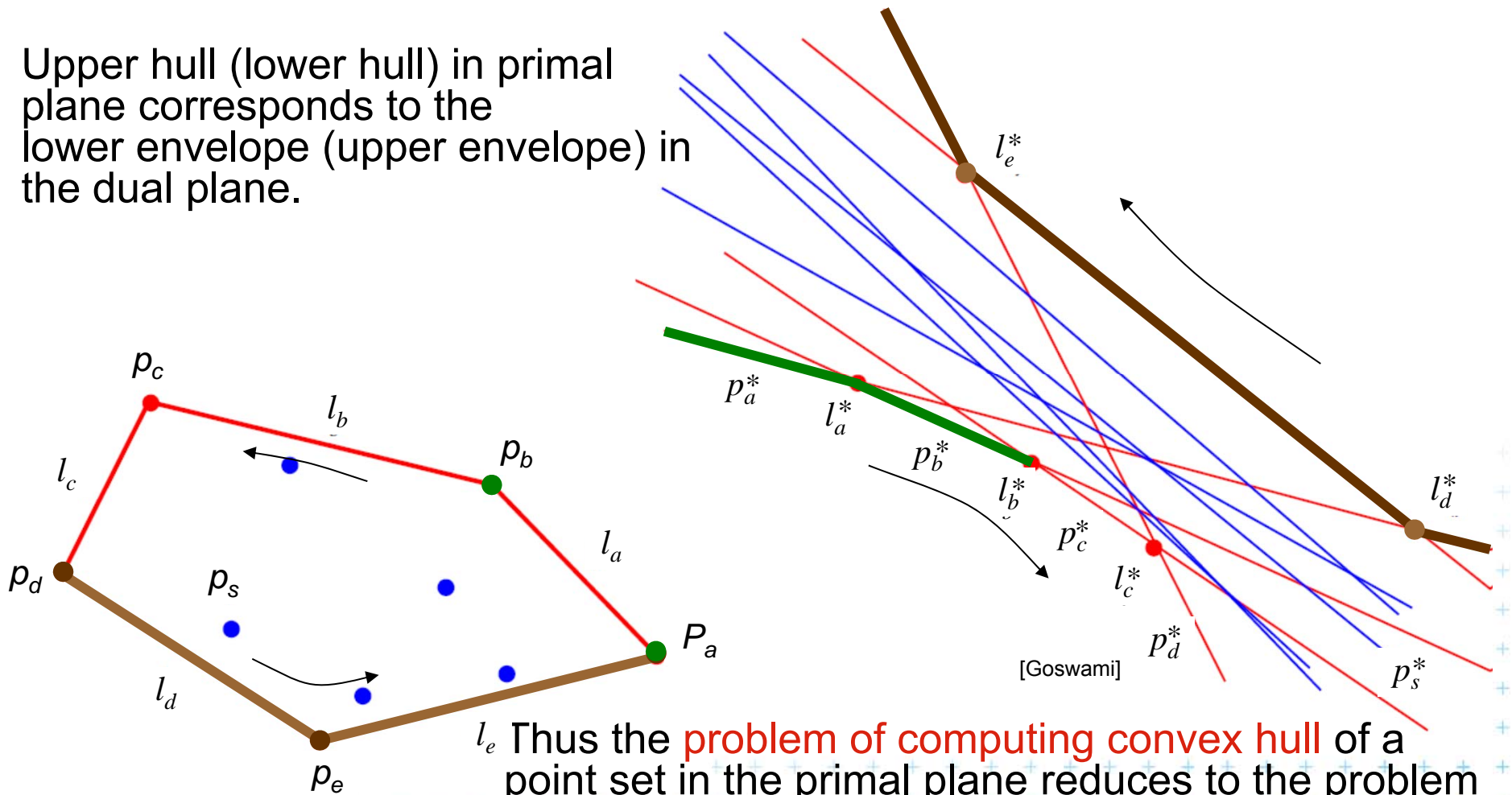


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

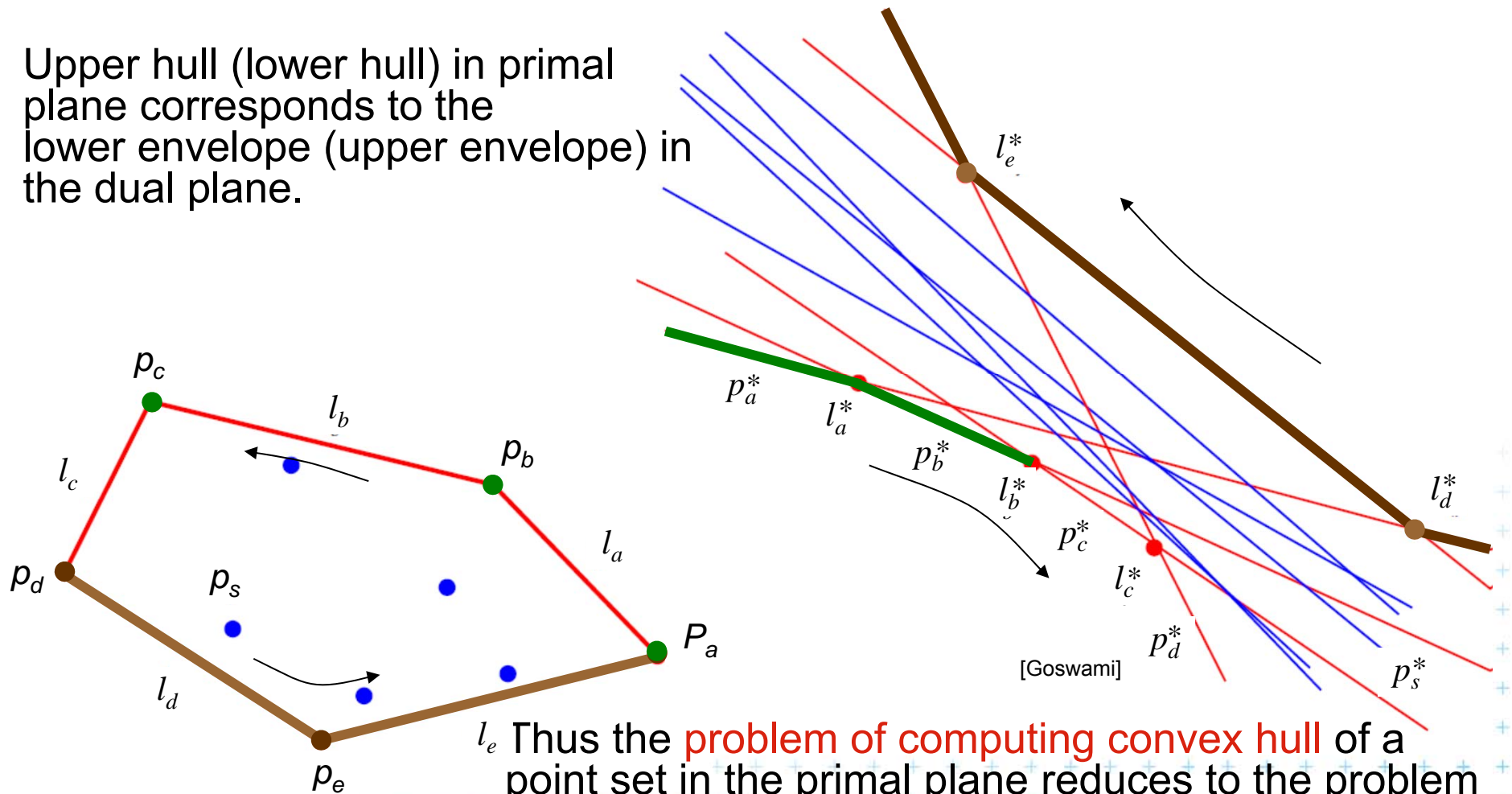


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

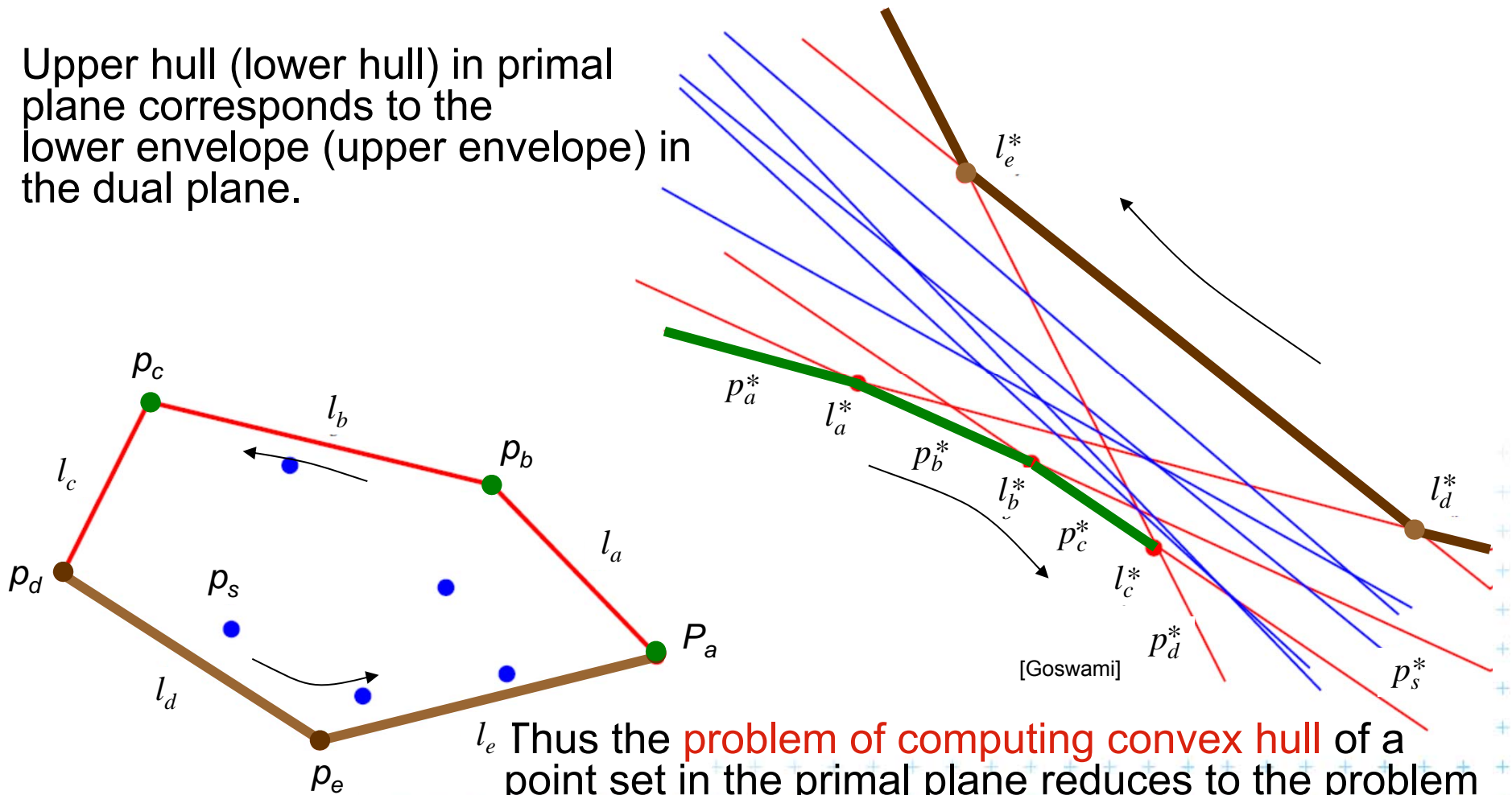


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

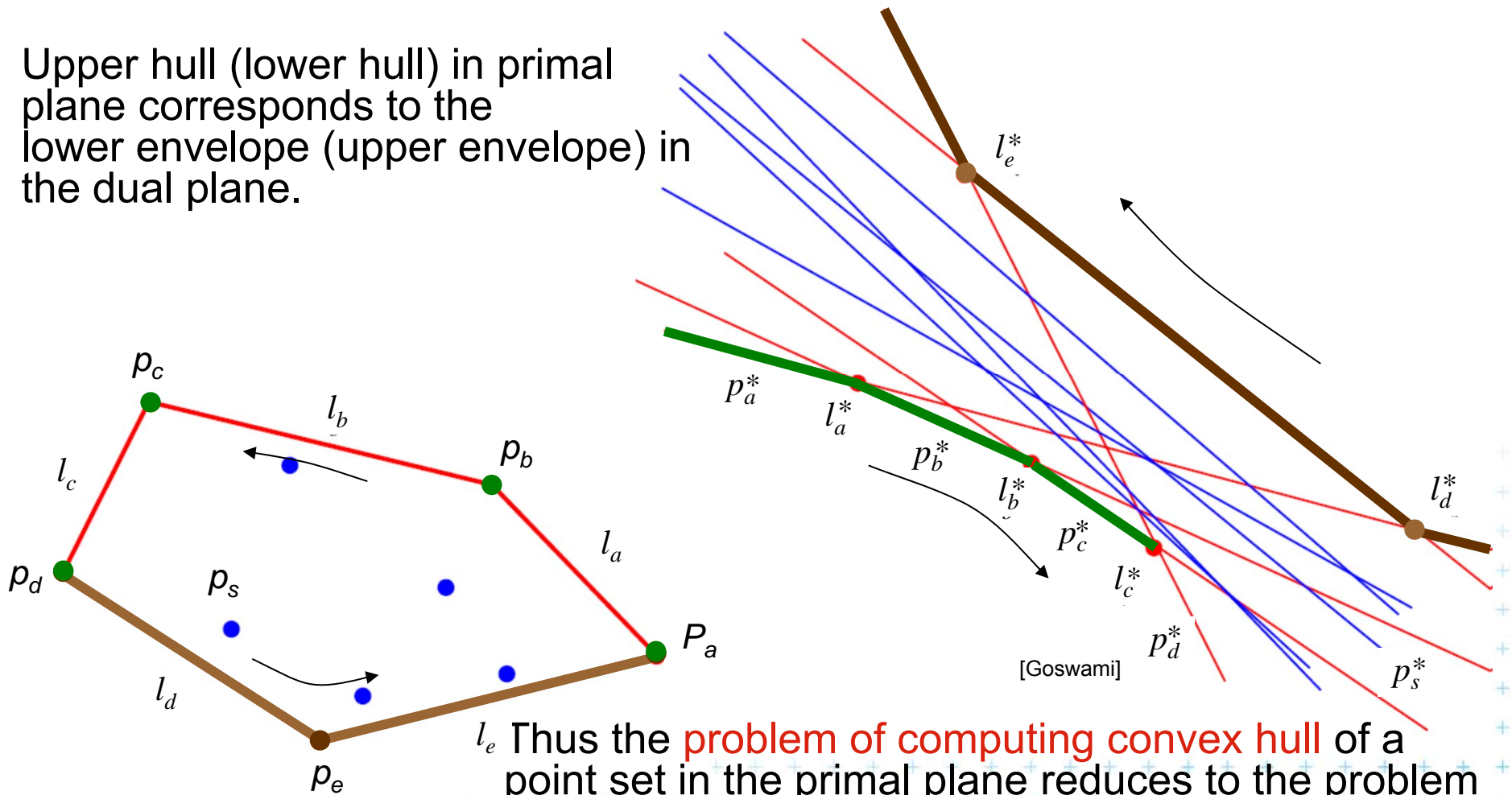


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

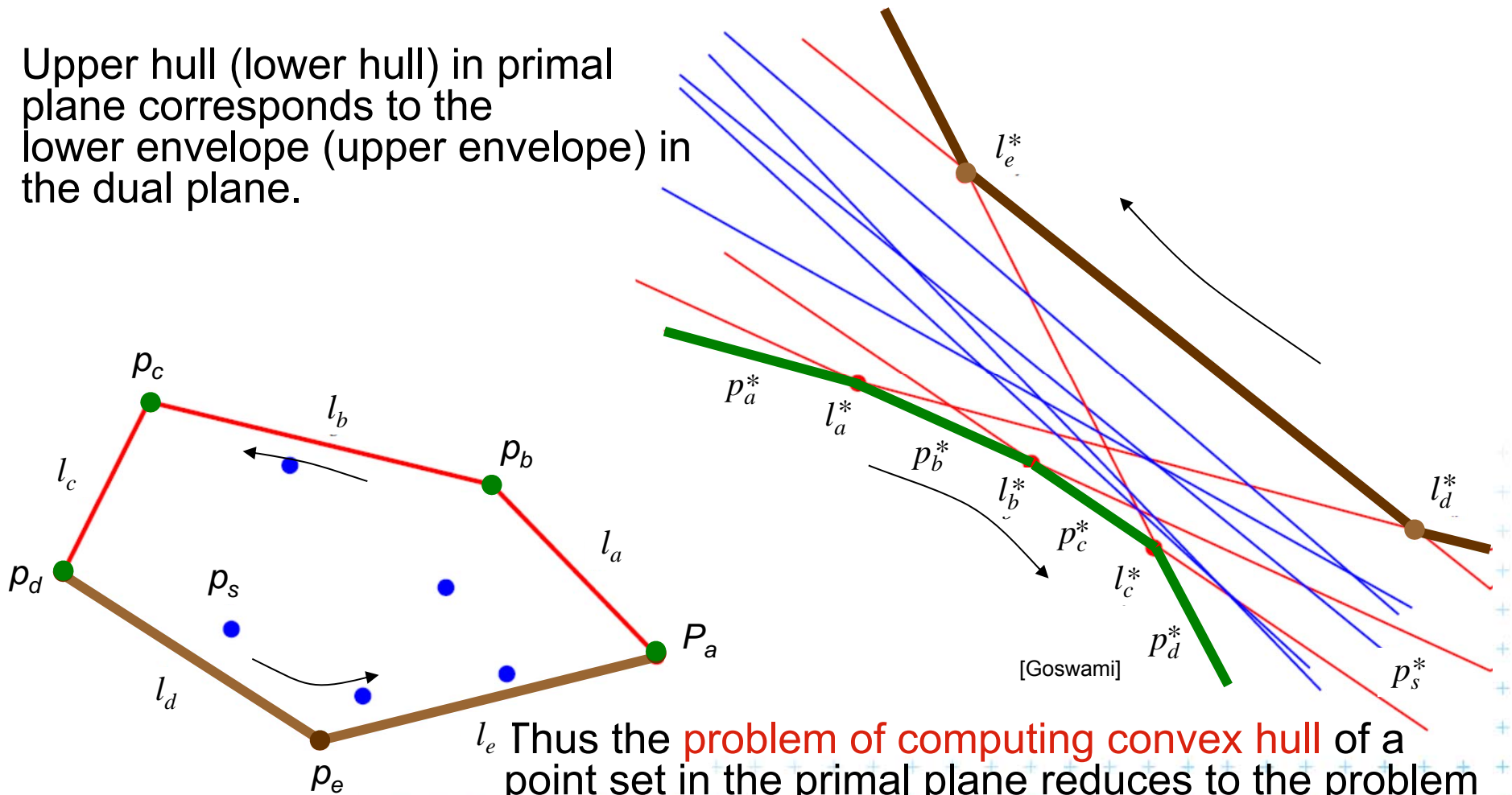


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

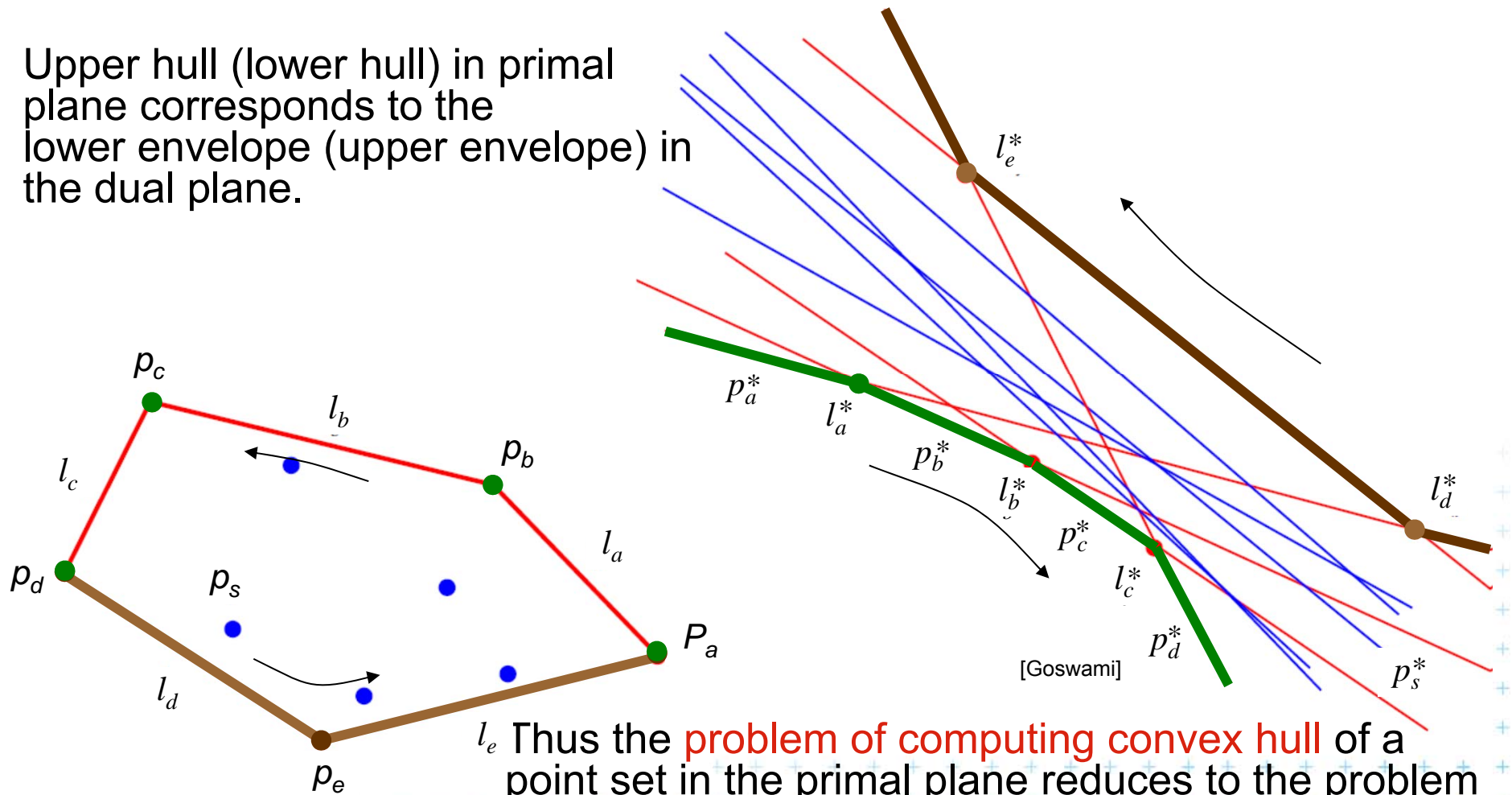


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

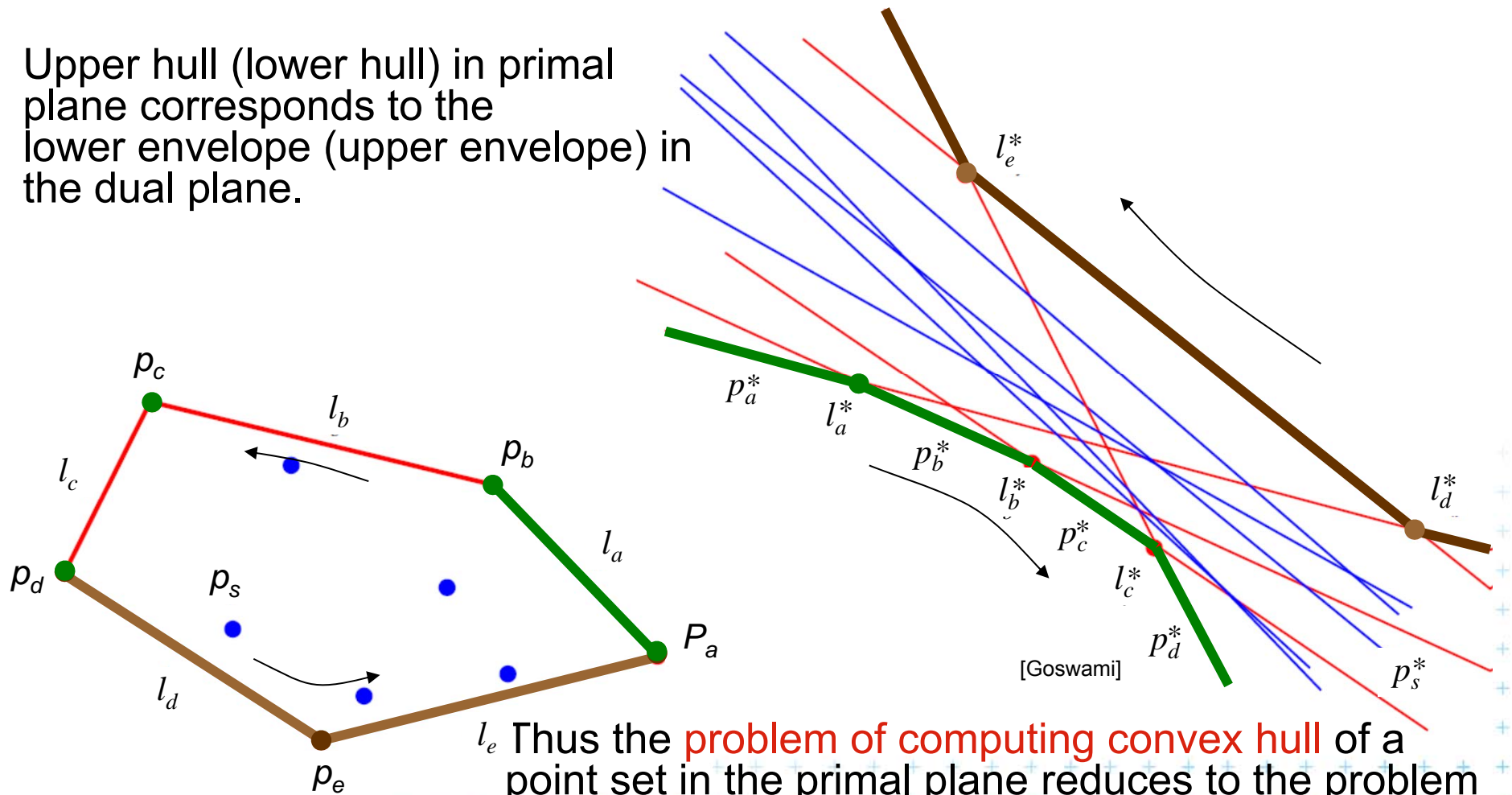


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

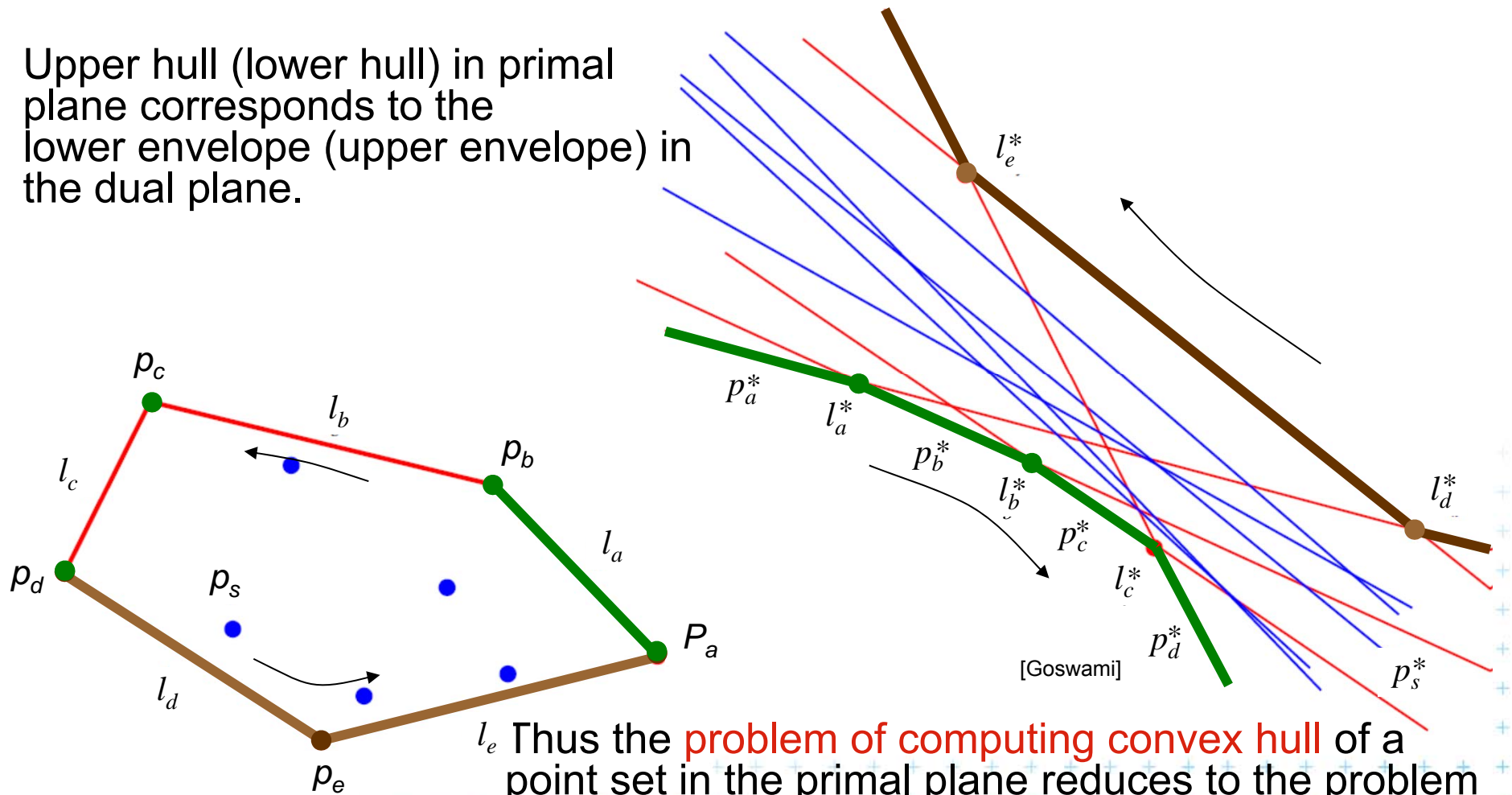


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

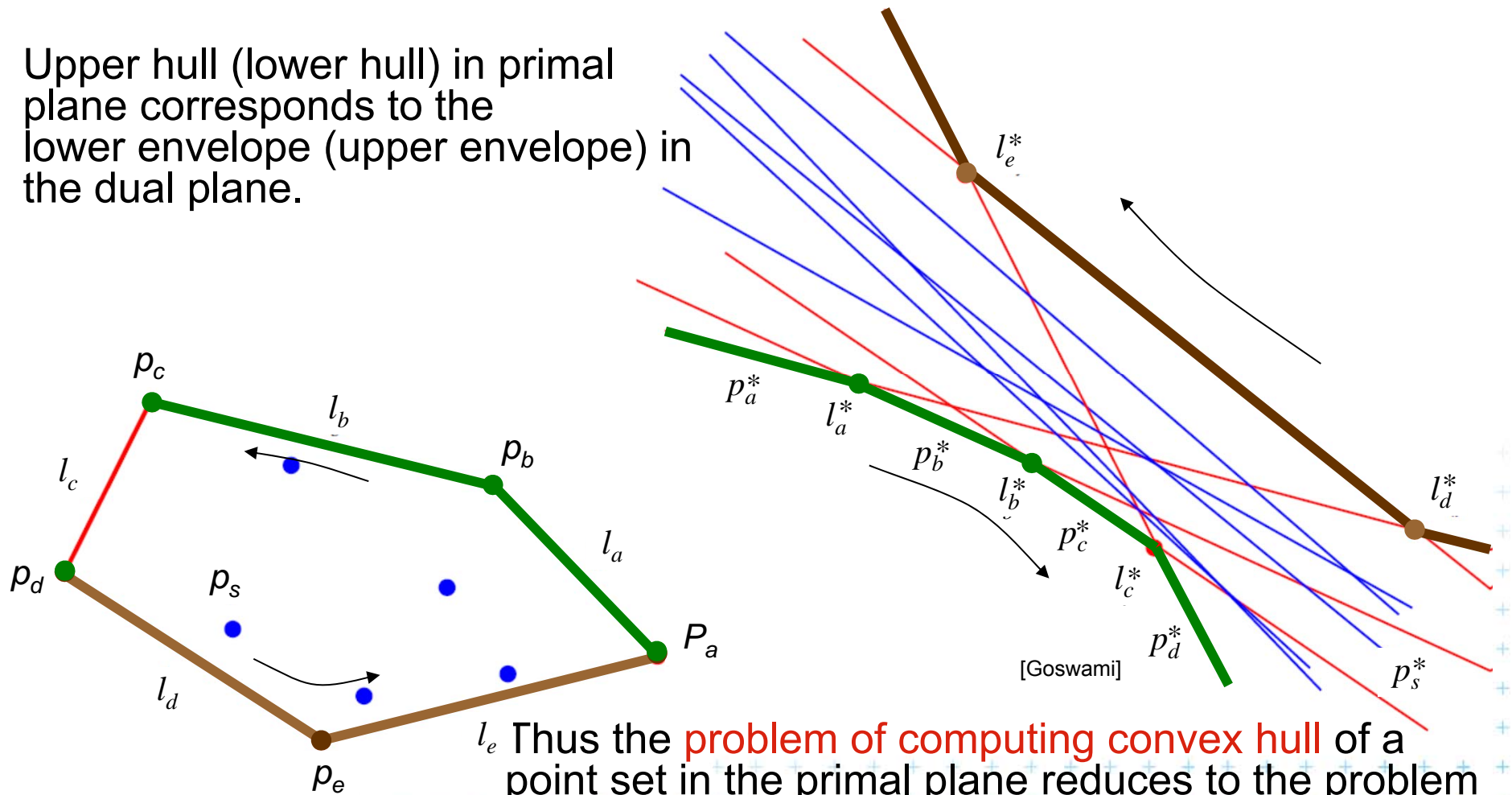


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

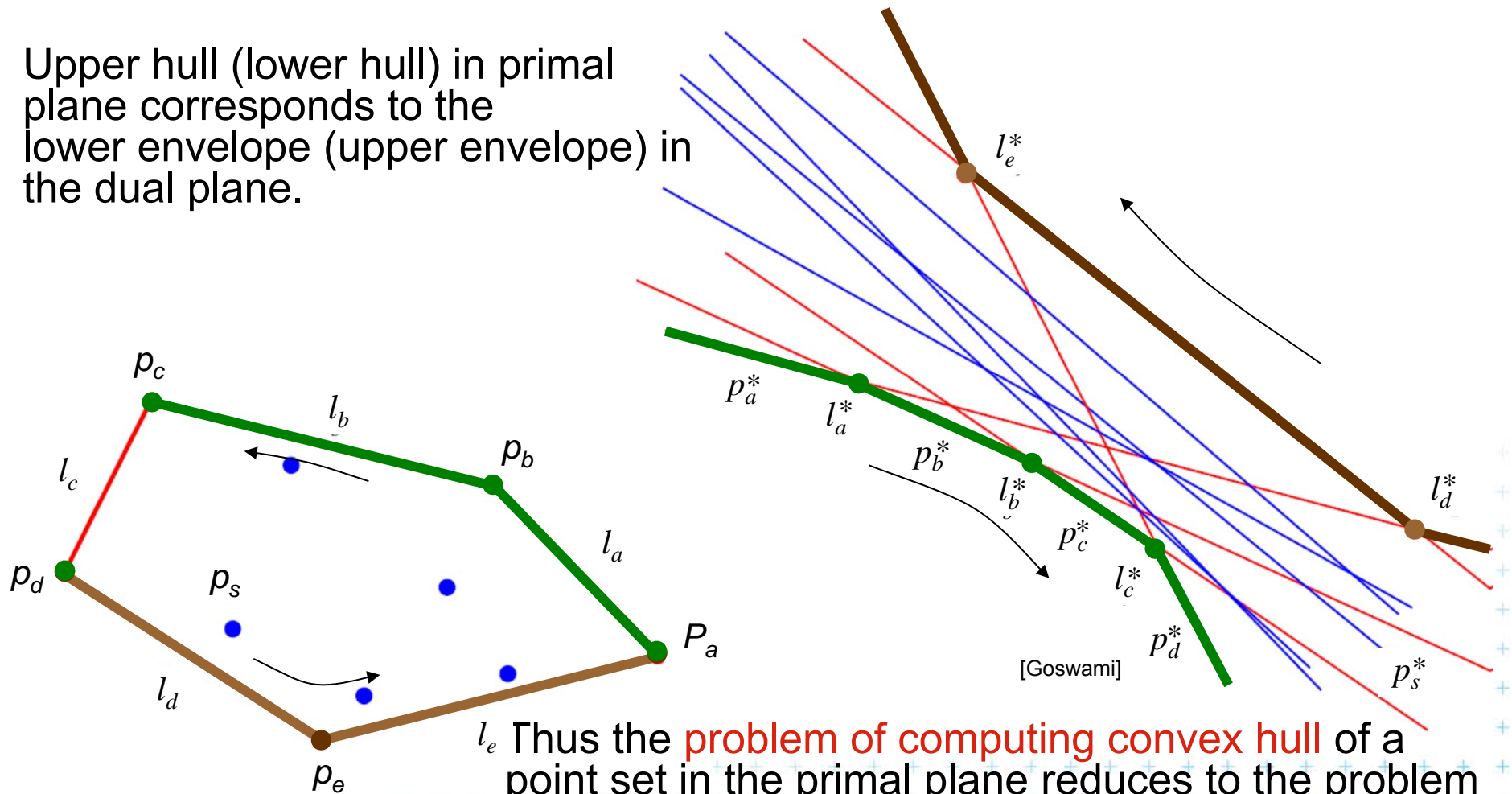


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.

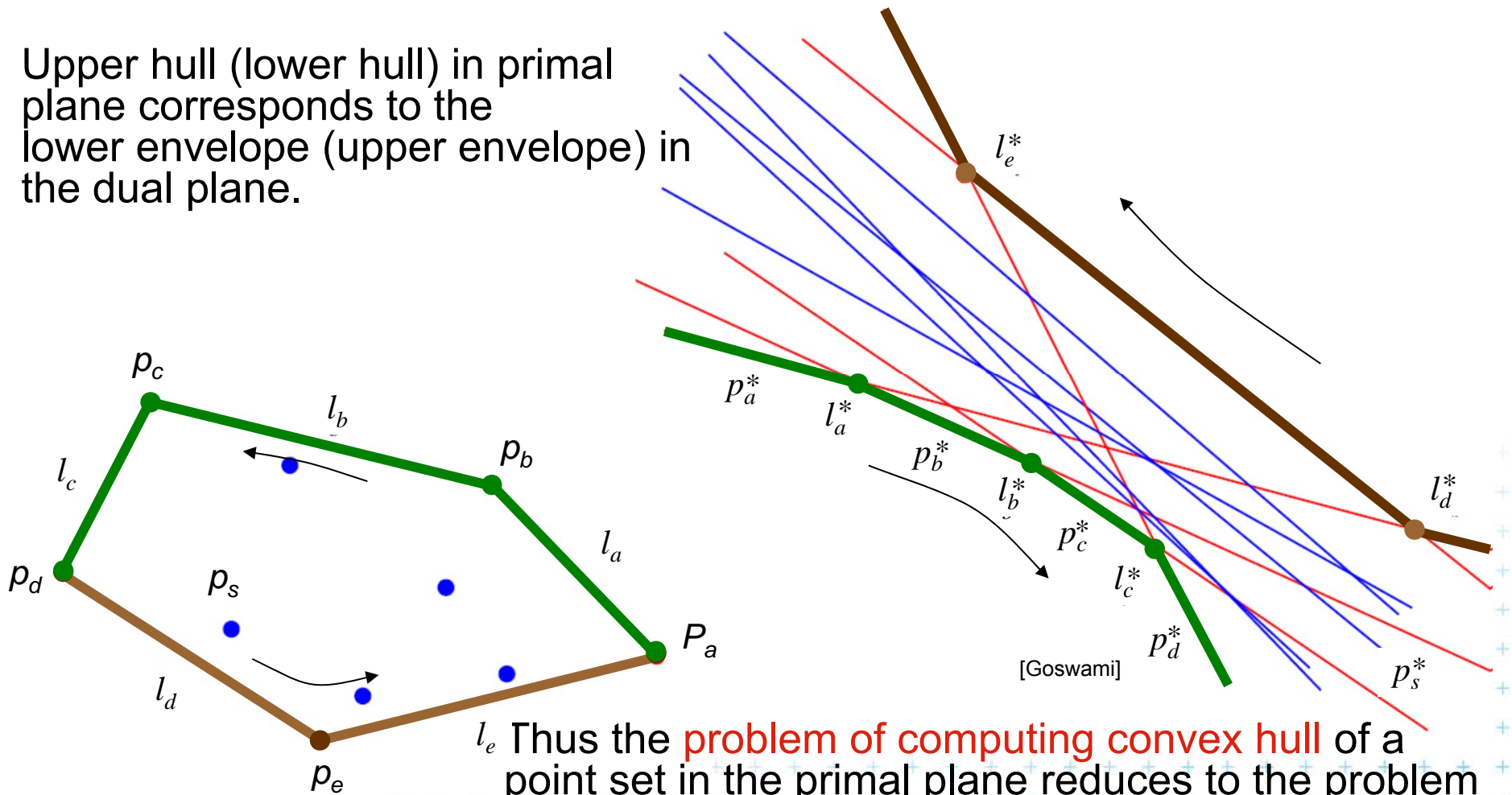


Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



Connection between Hull and Envelope

Upper hull (lower hull) in primal plane corresponds to the lower envelope (upper envelope) in the dual plane.



Thus the **problem of computing convex hull** of a point set in the primal plane reduces to the problem of computing **upper and lower envelopes** of the line set in the dual plane.



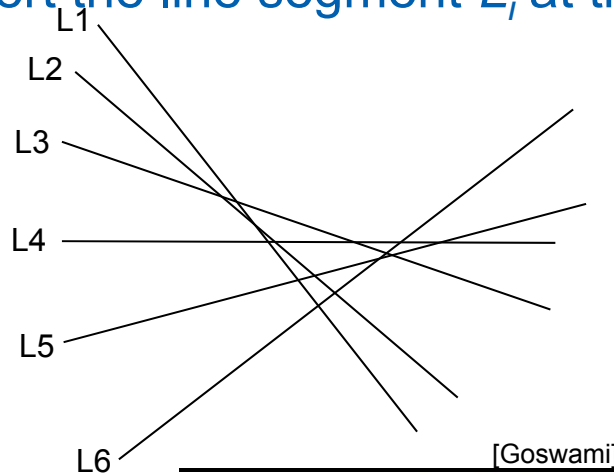
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



[Goswami]



DCGI

Felkel: Computational geometry

(22 / 38)



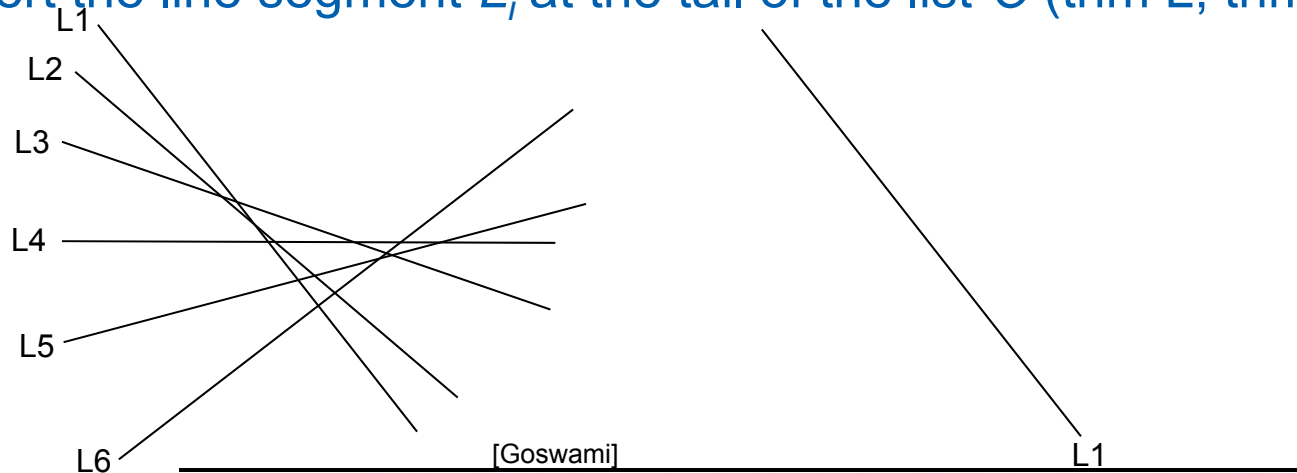
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



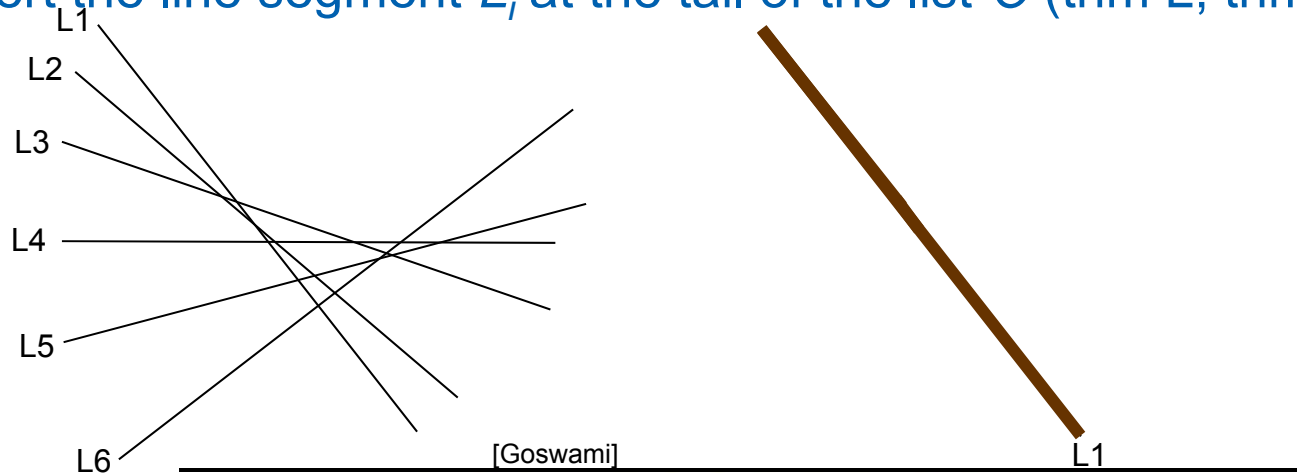
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



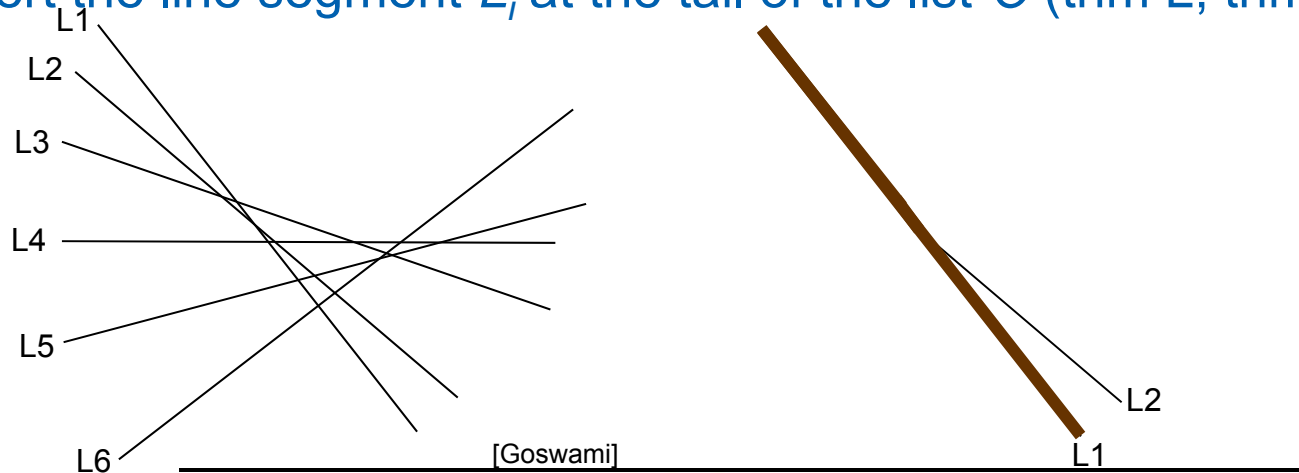
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



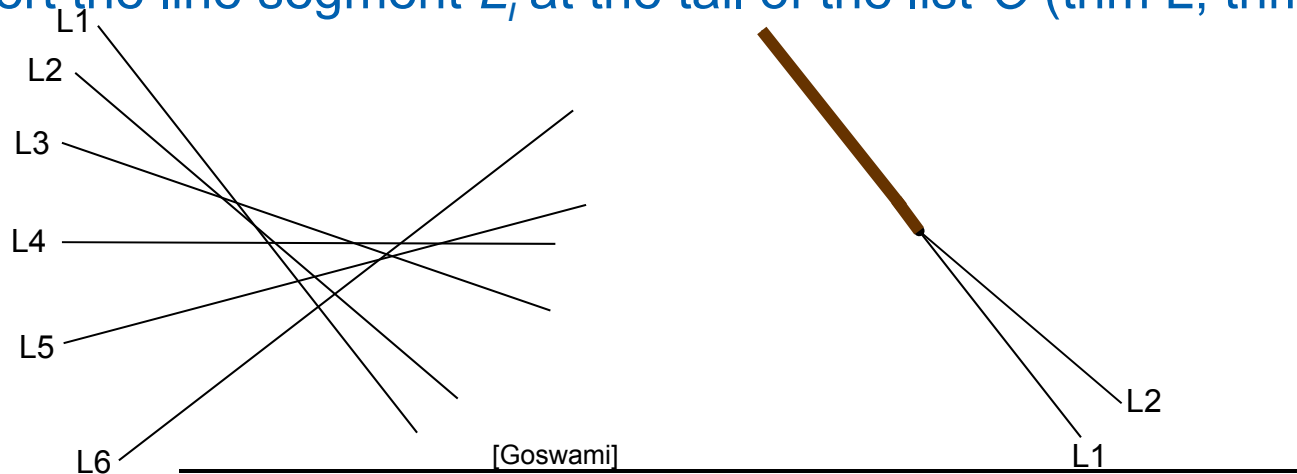
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



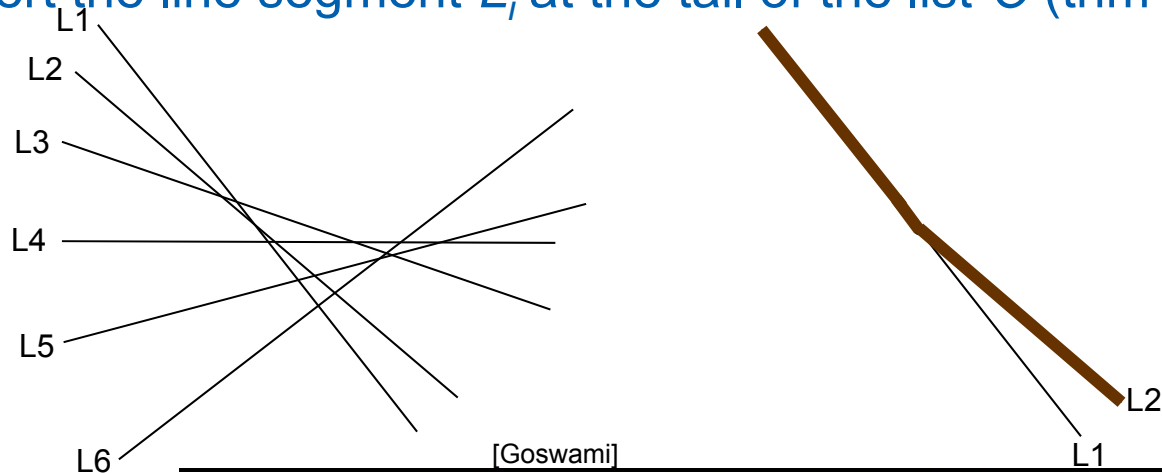
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



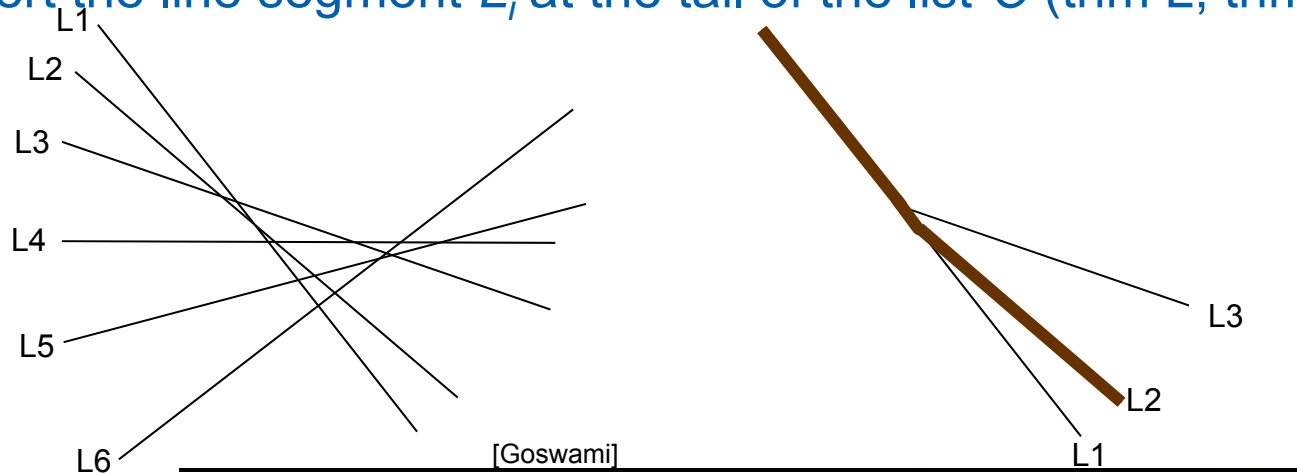
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. for $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. while(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



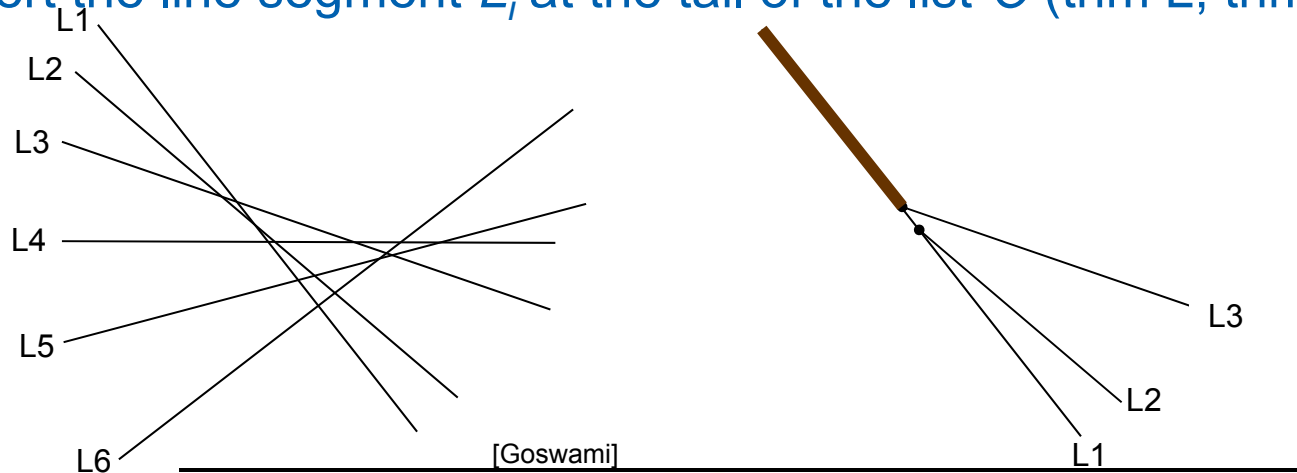
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



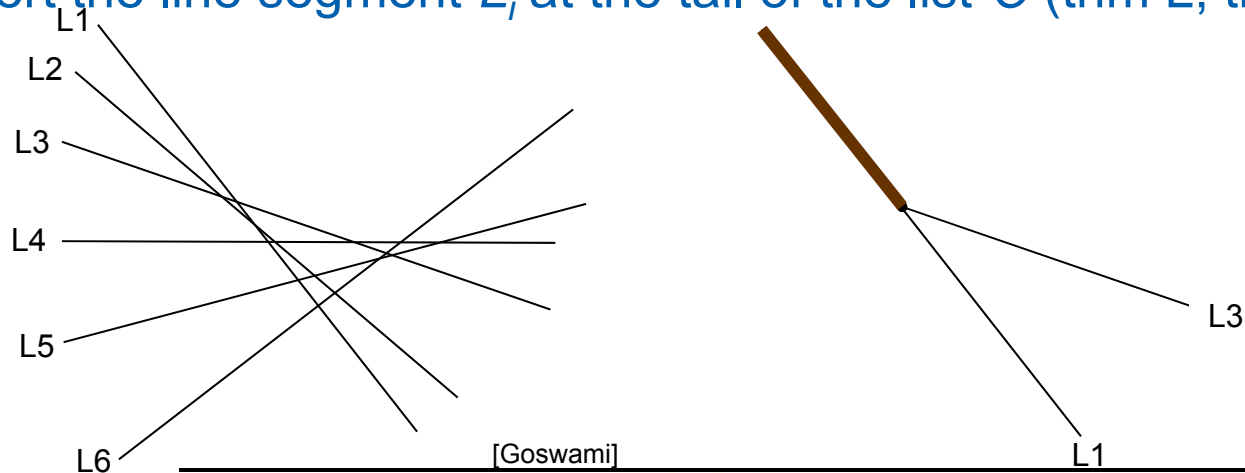
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



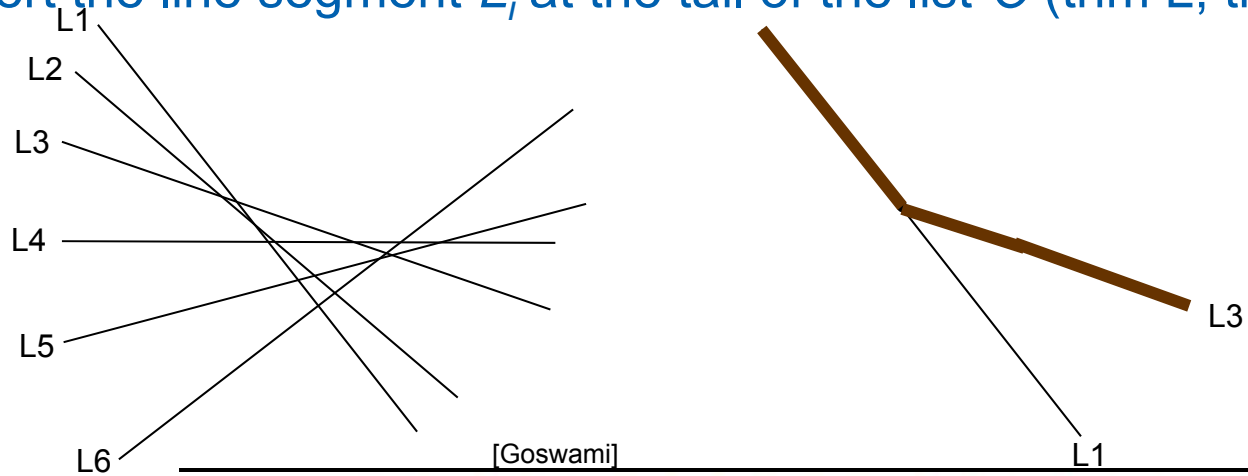
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



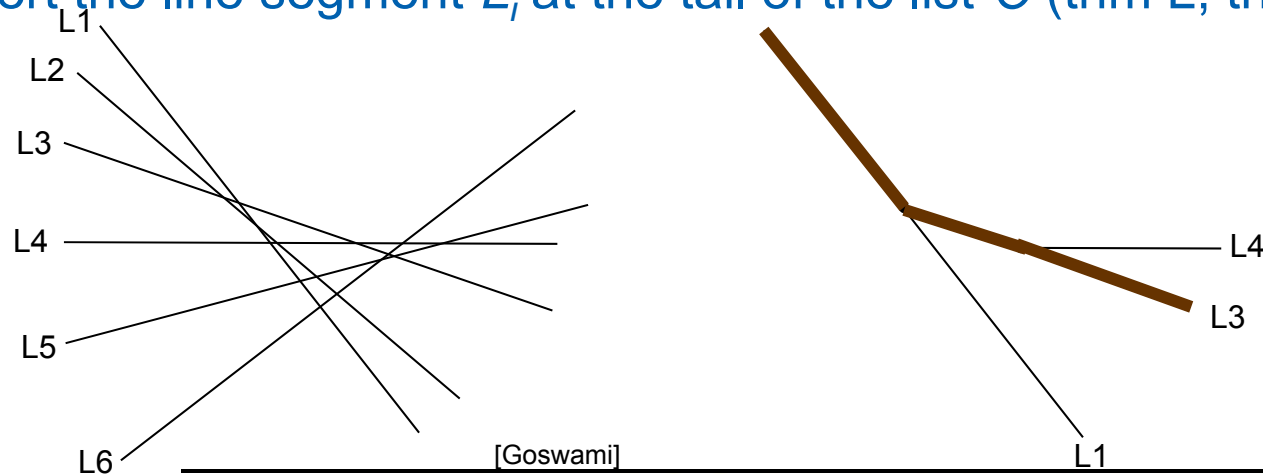
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



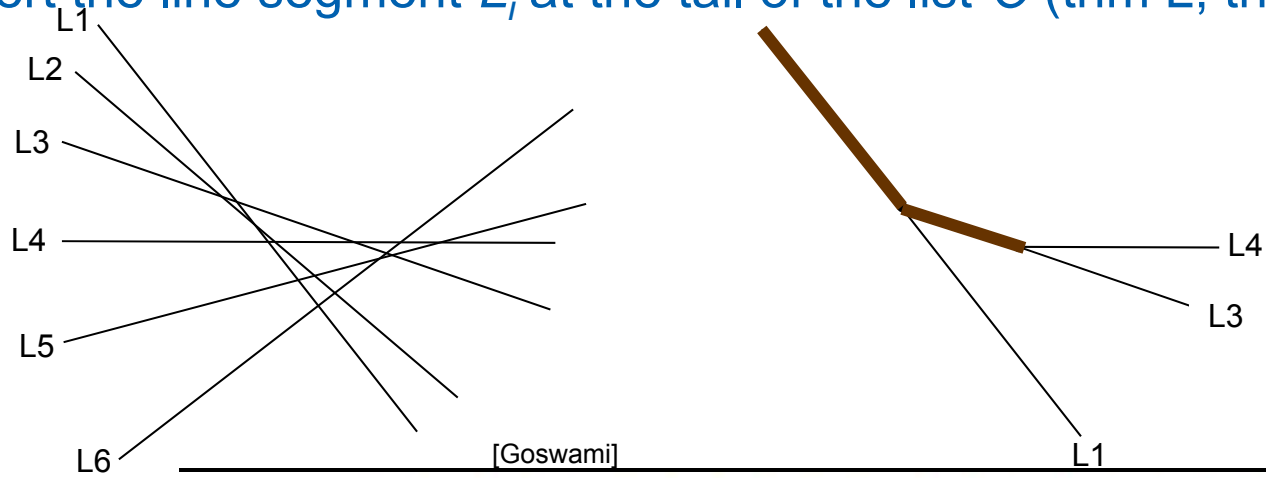
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments,
// except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



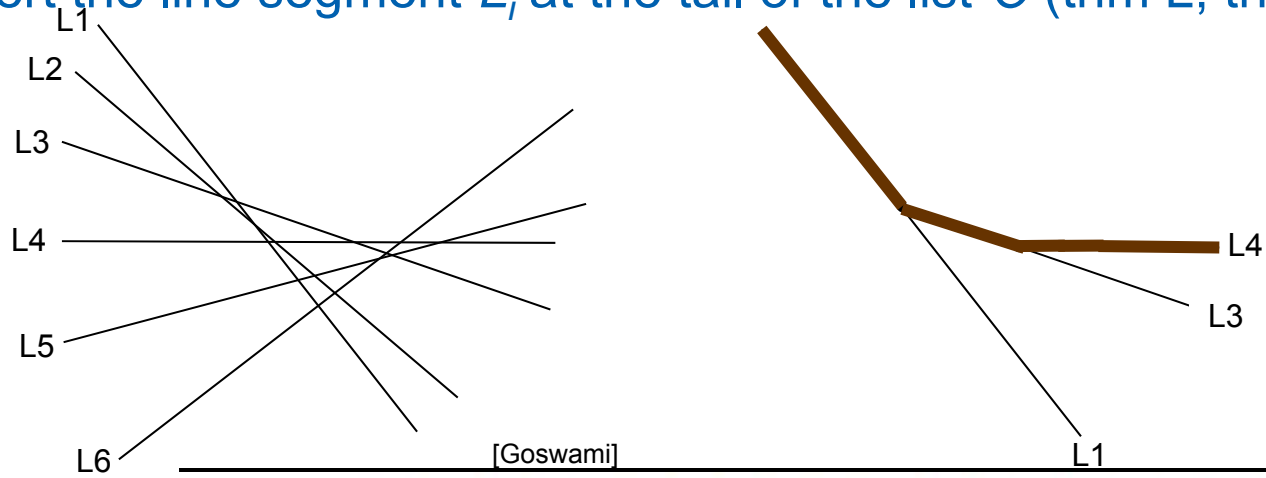
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments,
// except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



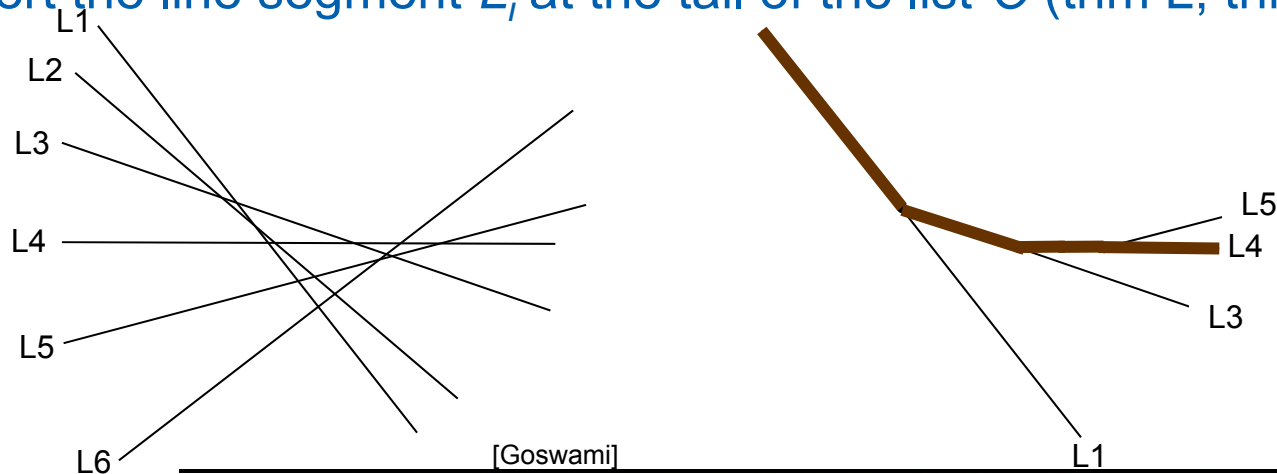
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. for $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. while(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



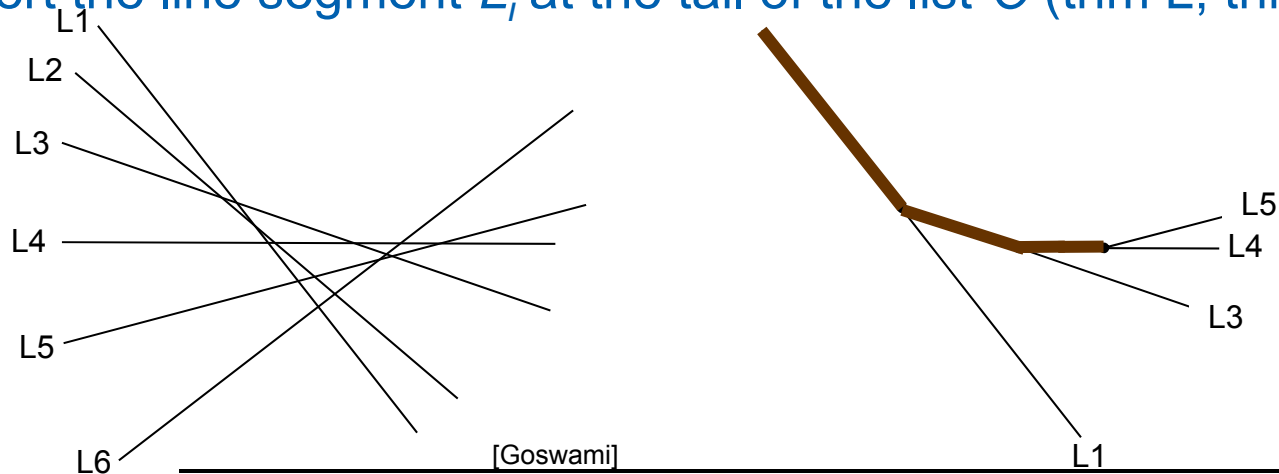
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments,
// except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



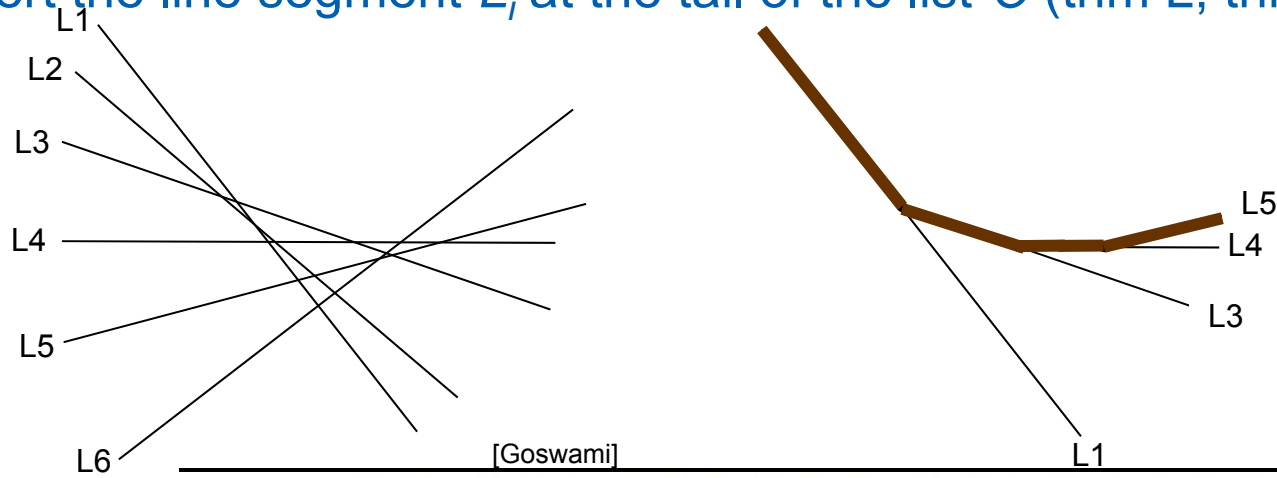
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. for $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. while(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



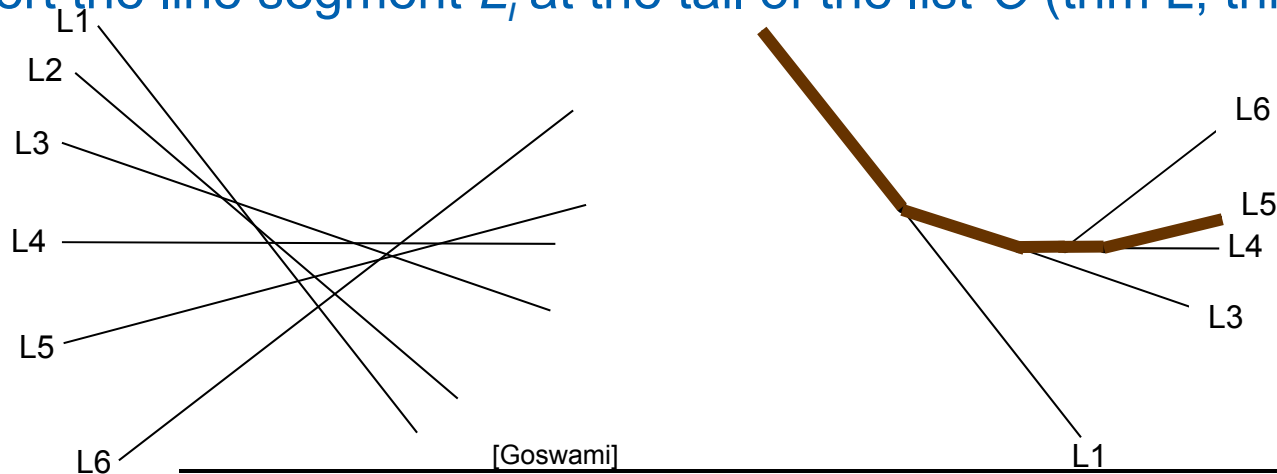
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. for $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. while(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



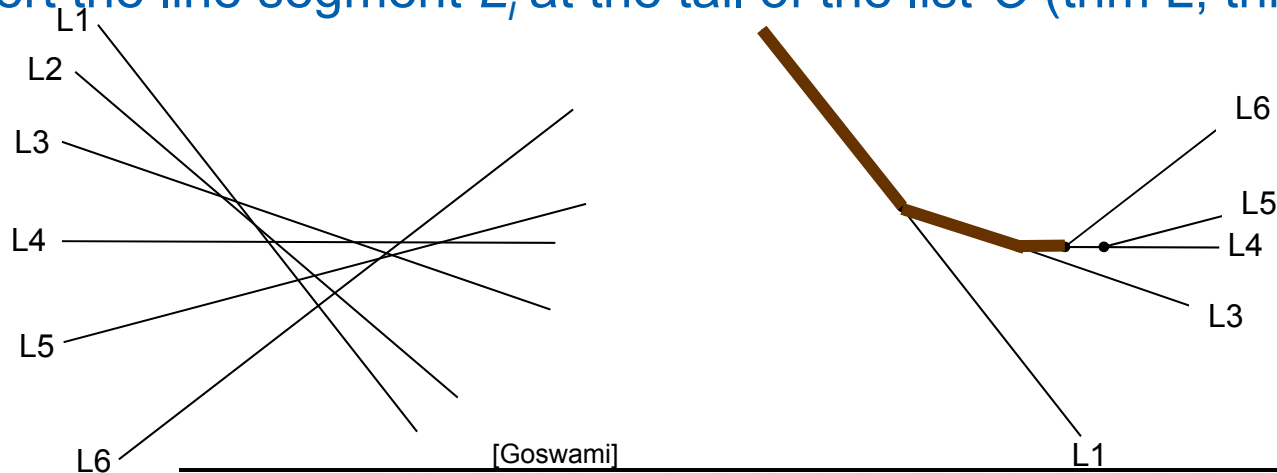
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



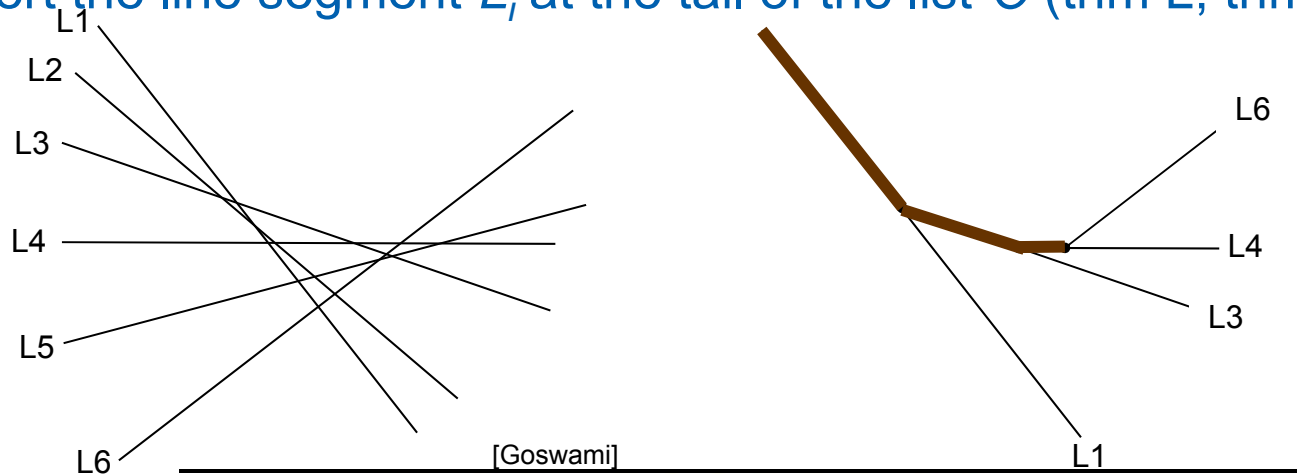
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



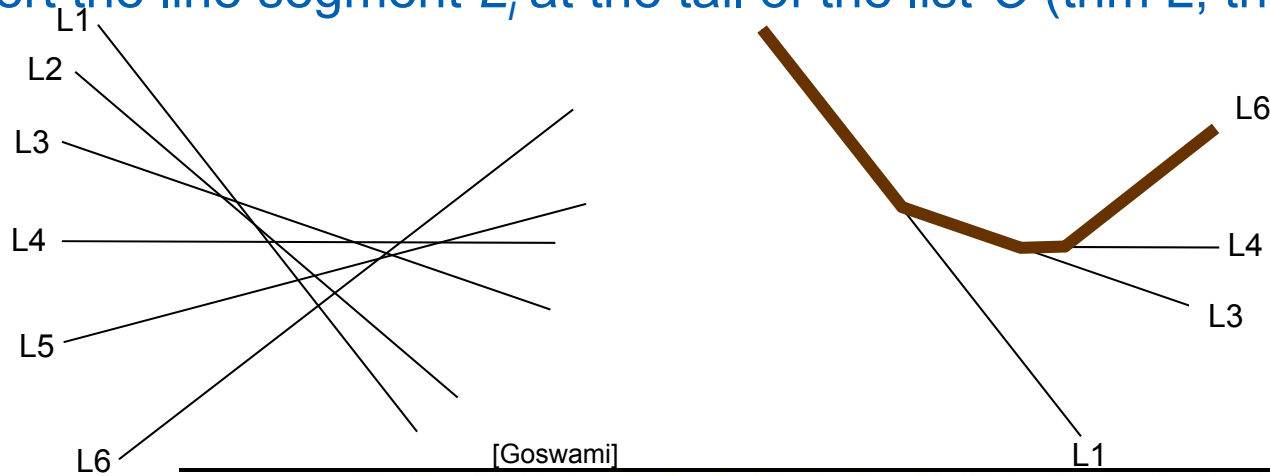
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



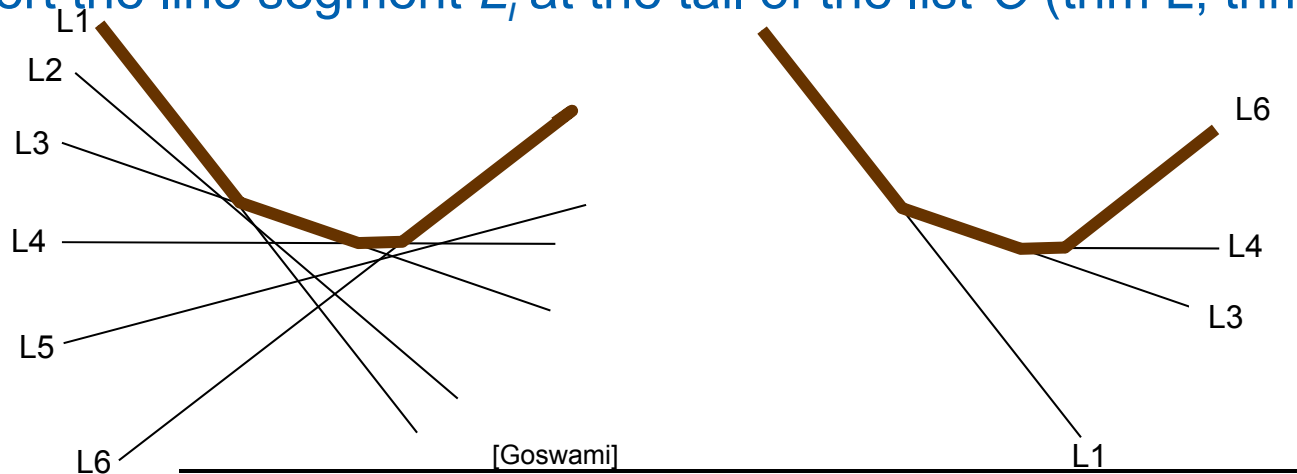
Upper envelope algorithm

UpperEnvelope(L)

Set of lines L sorted by **increasing** order of slopes (-90° to 90°)

Polygonal chain O representing the upper hull

1. $O = L_1$ // the only complete line in O
2. **for** $i = 2$ to n
3. $L =$ last entry in O // O contains half-lines, or line segments, // except of complete line L_1
4. **while**(the line segment L does not intersect line L_i)
5. remove L from O and replace L with its predecessor // L_2, L_5
6. insert the line segment L_i at the tail of the list O (trim L , trim L_i)



Convex hull via upper and lower envelope

■ Upper envelope complexity

- After sorting n lines by their slopes in $O(n \log n)$ time, the upper envelope can be obtained in $O(n)$ time
- Proof: It may check more than one line segment when inserting a new line, but those ones checked are all removed except the last one.
($O(n)$ insertions, max $O(n)$ removals
=> $O(n)$ all steps. Average step $O(1)$ amortized time)

■ Convex hull complexity

- Given a set P of n points in the plane, $\text{CH}(P)$ can be computed in $O(n \log n)$ time using $O(n)$ space.



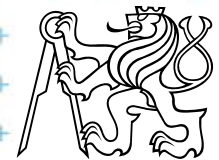
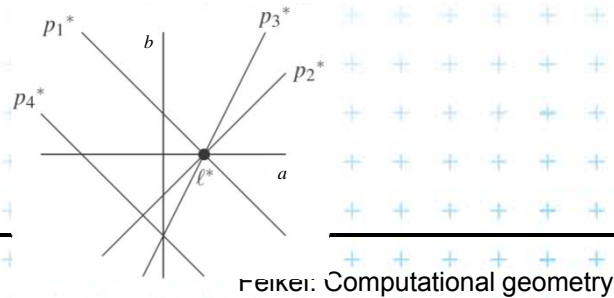
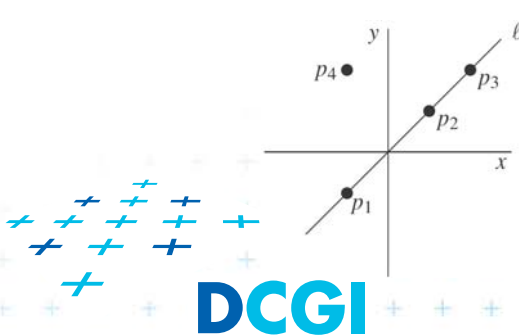
Applications of line arrangement

Examples of applications – solved in $O(n^2)$ and
↙ $O(n^2)$ space by constructing a line arrangement or
 $O(n)$ space through topological plain sweep.

a) General position test:

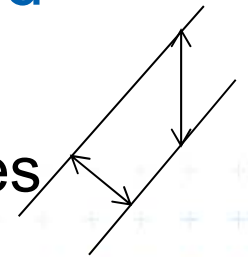
Given a set of n points in the plane, determine whether any three are collinear.

- Construct an arrangement in dual plane
- Report intersections of more than 2 lines



b) Minimum k-corridor

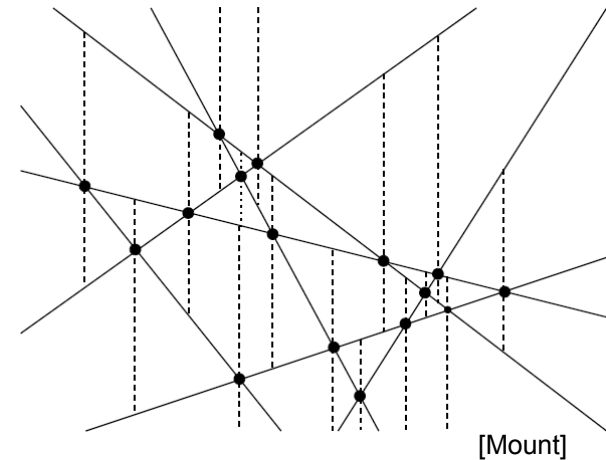
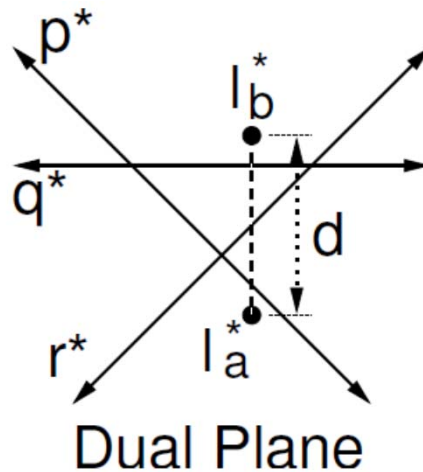
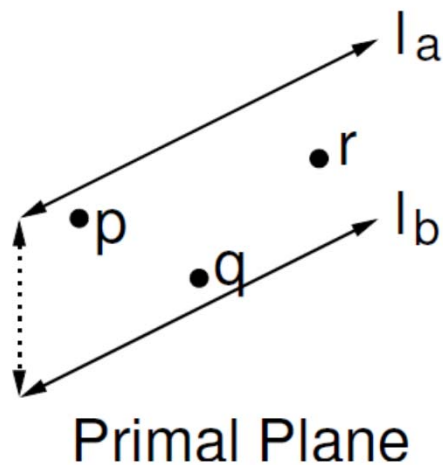
- Given a set of n points, and an integer $k \in [1 : n]$, determine the **narrowest pair of parallel lines** that **enclose at least k points** of the set.
- The distance between the lines can be defined
 - either as the **vertical distance** between the lines
 - or as the **perpendicular distance** between the lines
- Simplifications
 - Assume $k = 3$ and **no 3 points are collinear**
=> narrowest corridor - contains exactly 3 points
- has width > 0
 - No 2 points have the same x coordinate (avoid I duals)



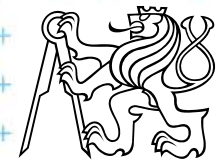
vertical



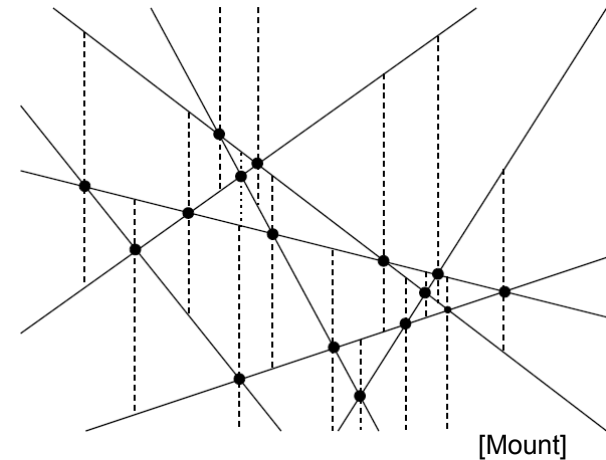
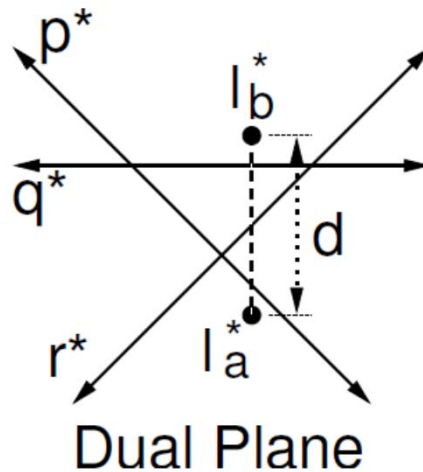
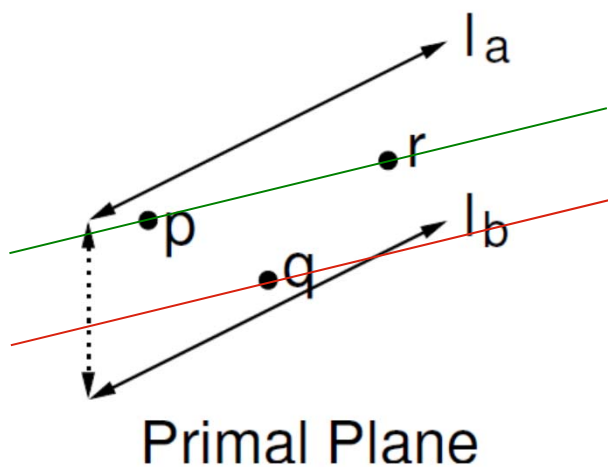
b) Minimum k-corridor



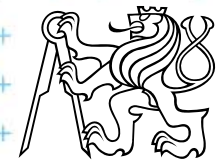
- **Vertical distance** of $l_a, l_b = (-)$ distance of l_a^*, l_b^*
- Nearest lines – one passes 2 vertices, e.g., p & r
- In dual plane are represented as intersection $p^* \times r^*$
- Find nearest 3-stabber similarly as trapezoidal map
- $O(n^2)$ time and $O(n)$ space – topological line sweep



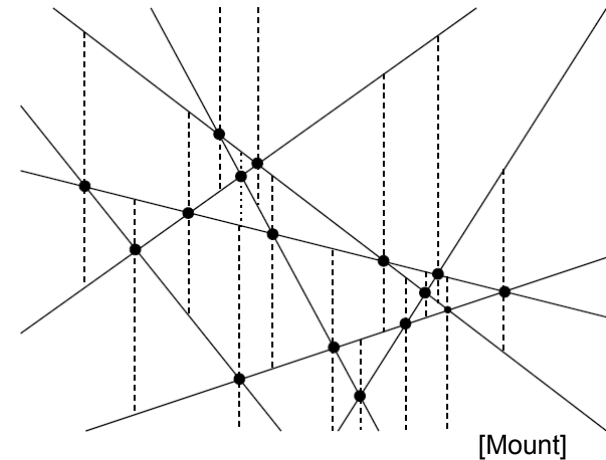
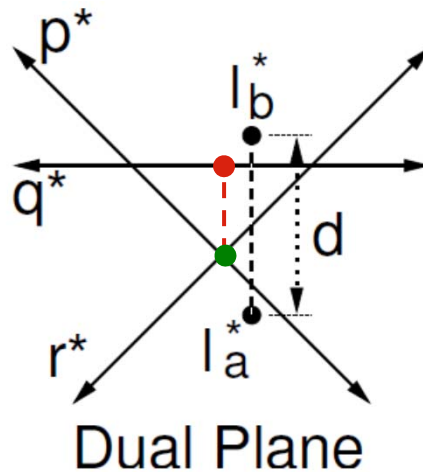
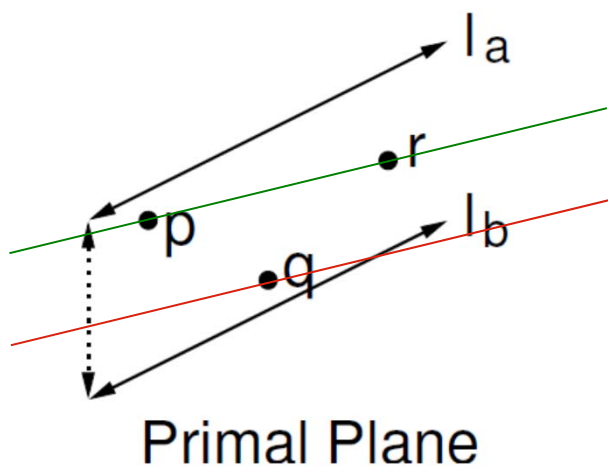
b) Minimum k-corridor



- **Vertical distance** of $l_a, l_b = (-)$ distance of l_a^*, l_b^*
- Nearest lines – one passes 2 vertices, e.g., p & r
- In dual plane are represented as intersection $p^* \times r^*$
- Find nearest 3-stabber similarly as trapezoidal map
- $O(n^2)$ time and $O(n)$ space – topological line sweep



b) Minimum k-corridor



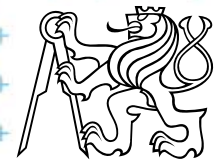
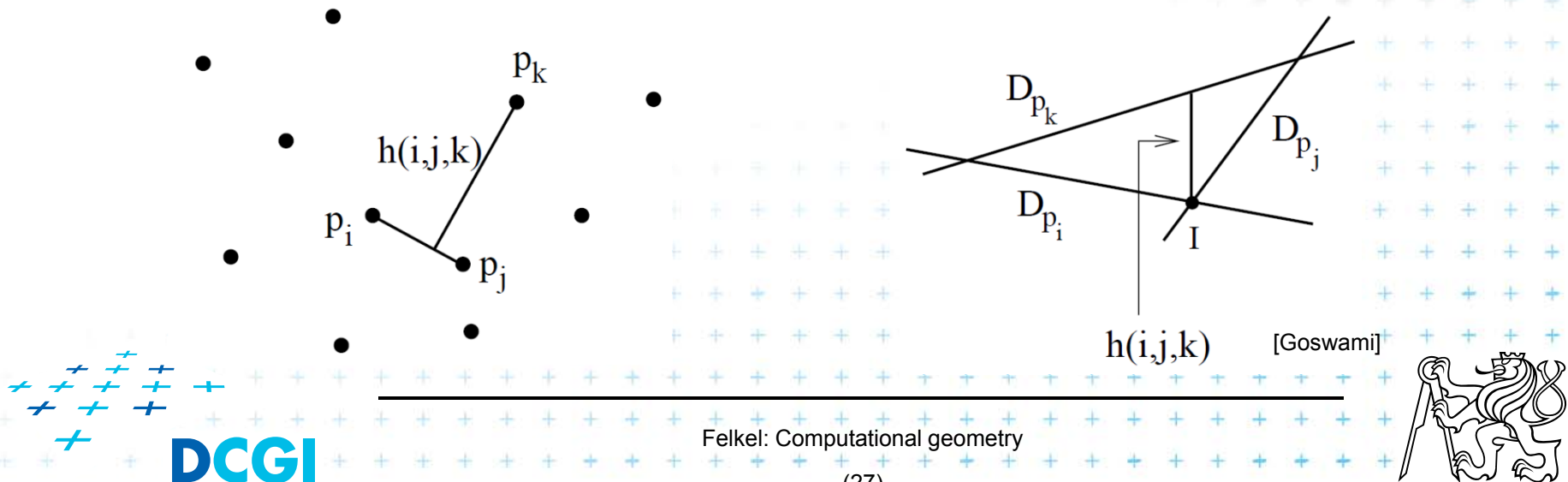
- Vertical distance of $l_a, l_b = (-)$ distance of l_a^*, l_b^*
- Nearest lines – one passes 2 vertices, e.g., p & r
- In dual plane are represented as intersection $p^* \times r^*$
- Find nearest 3-stabber similarly as trapezoidal map
- $O(n^2)$ time and $O(n)$ space – topological line sweep



c) Minimum area triangle

[Goswami]

- Given a set of n points in the plane, determine the minimum area triangle whose vertices are selected from these points
- Construct “trapezoids” as in the nearest corridor
- Minimize perpendicular distances (converted from vertical) multiplied by the distance from p_i to p_j



d) Sorting all angular sequences – naïve

- Natural application of duality and arrangements
- Important for **visibility graph** computation
- Set of n points in the plane
- For **each point** perform an **CCW angular sweep**
- Naïve: for each point compute angles to remaining $n - 1$ points and sort them
- $\Rightarrow O(n \log n)$ time per point
- $O(n^2 \log n)$ time overall
- Arrangements can get rid of $O(\log n)$ factor

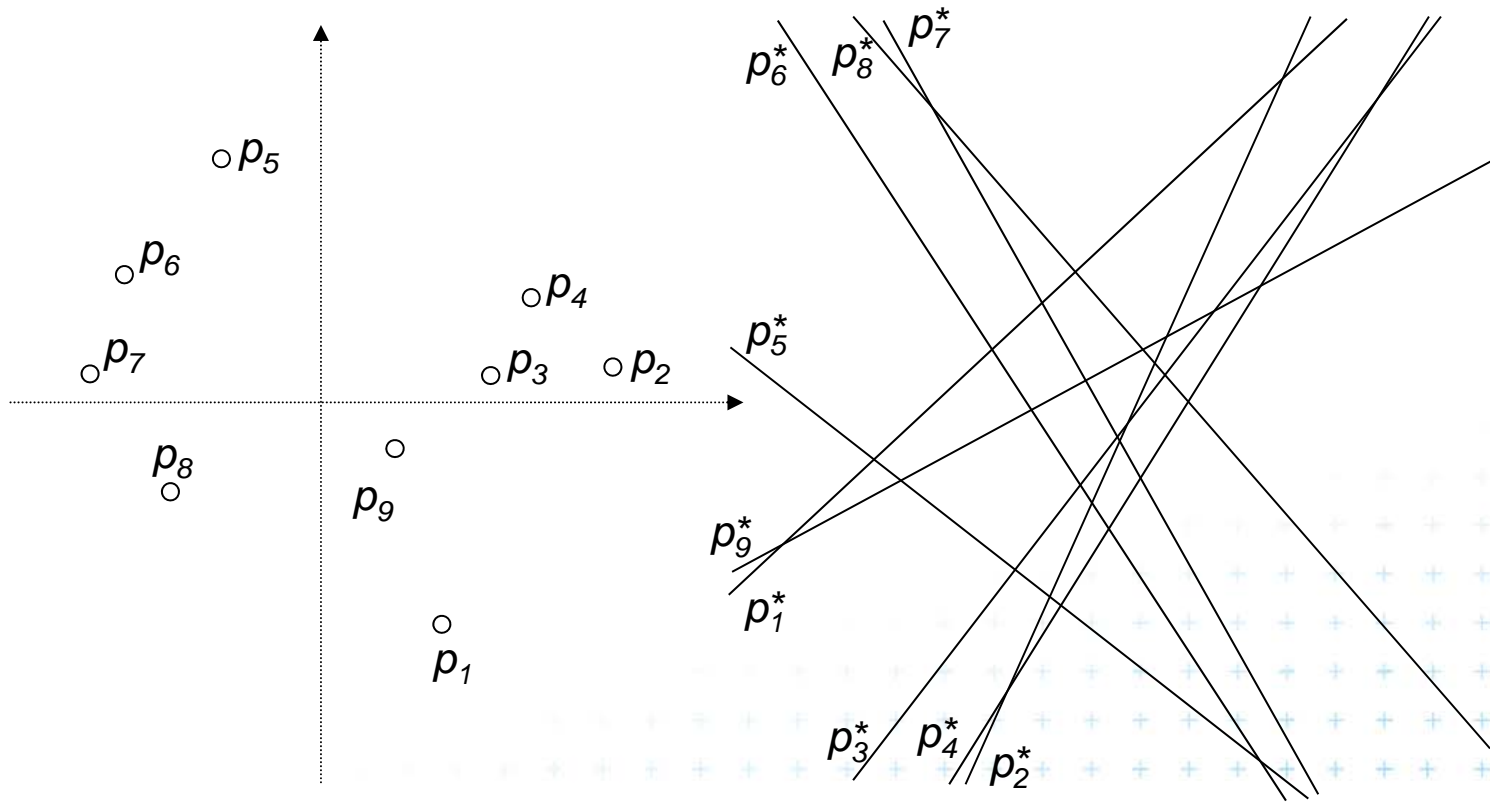


d) Sorting all angular sequences – optimal

- For point p_i
 - Dual of point p_i is line p_i^*
 - Line p_i^* intersects other dual lines in **order of slope**
(angles from -90° to 90°) (180°)
 - We need **order of angles around p_i**
(angles from -90° to 270°) (360°)
 - Split points in primal plane by vertical line through p_i
 - First, report intersections of points **right of p_i**
 - Second, report the intersections of points **left of p_i**
 - Once the arrangement is constructed:
 $O(n)$ time for point, **$O(n^2)$ time for all n points**



d) Angular sequence around p_9

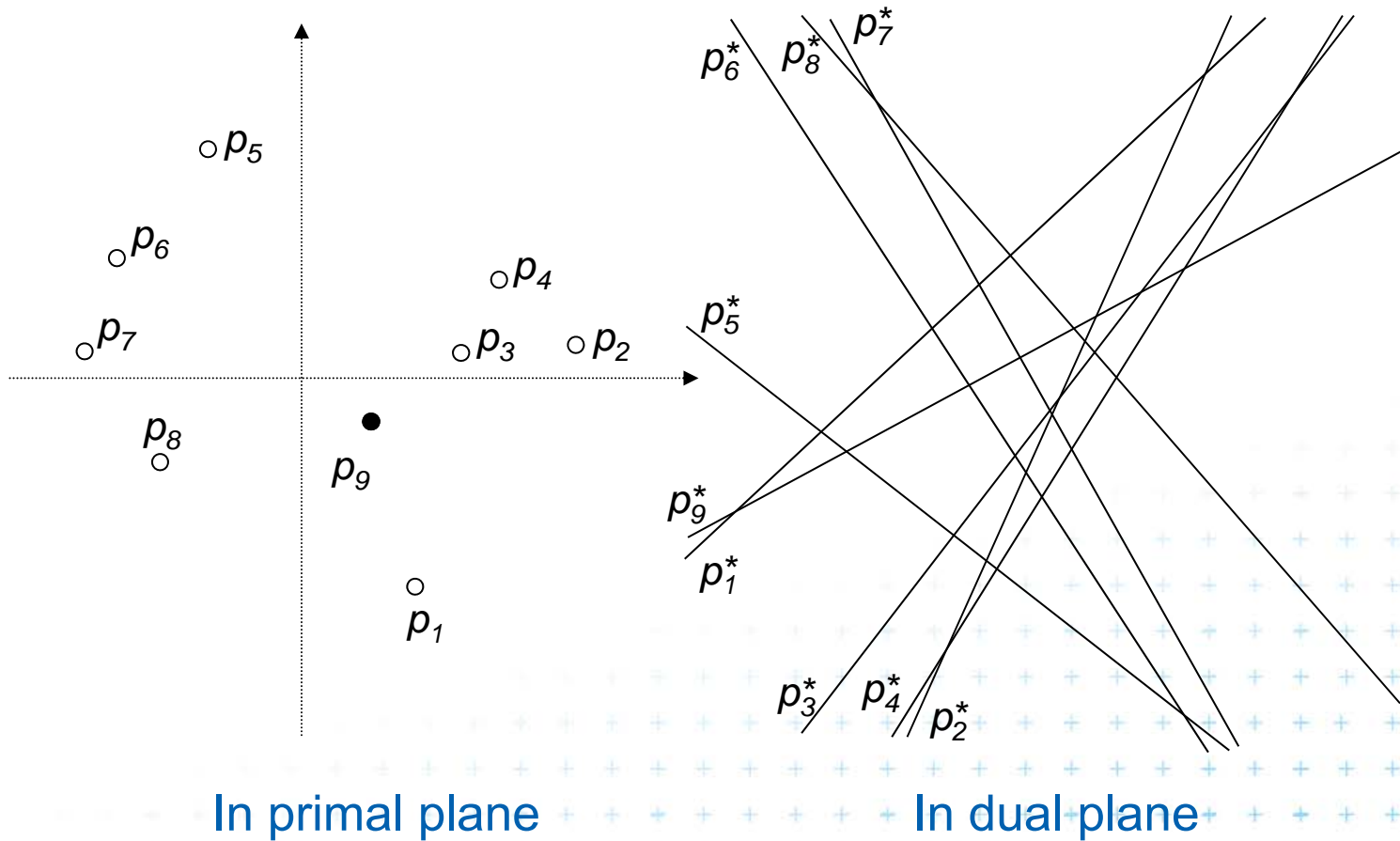


In primal plane

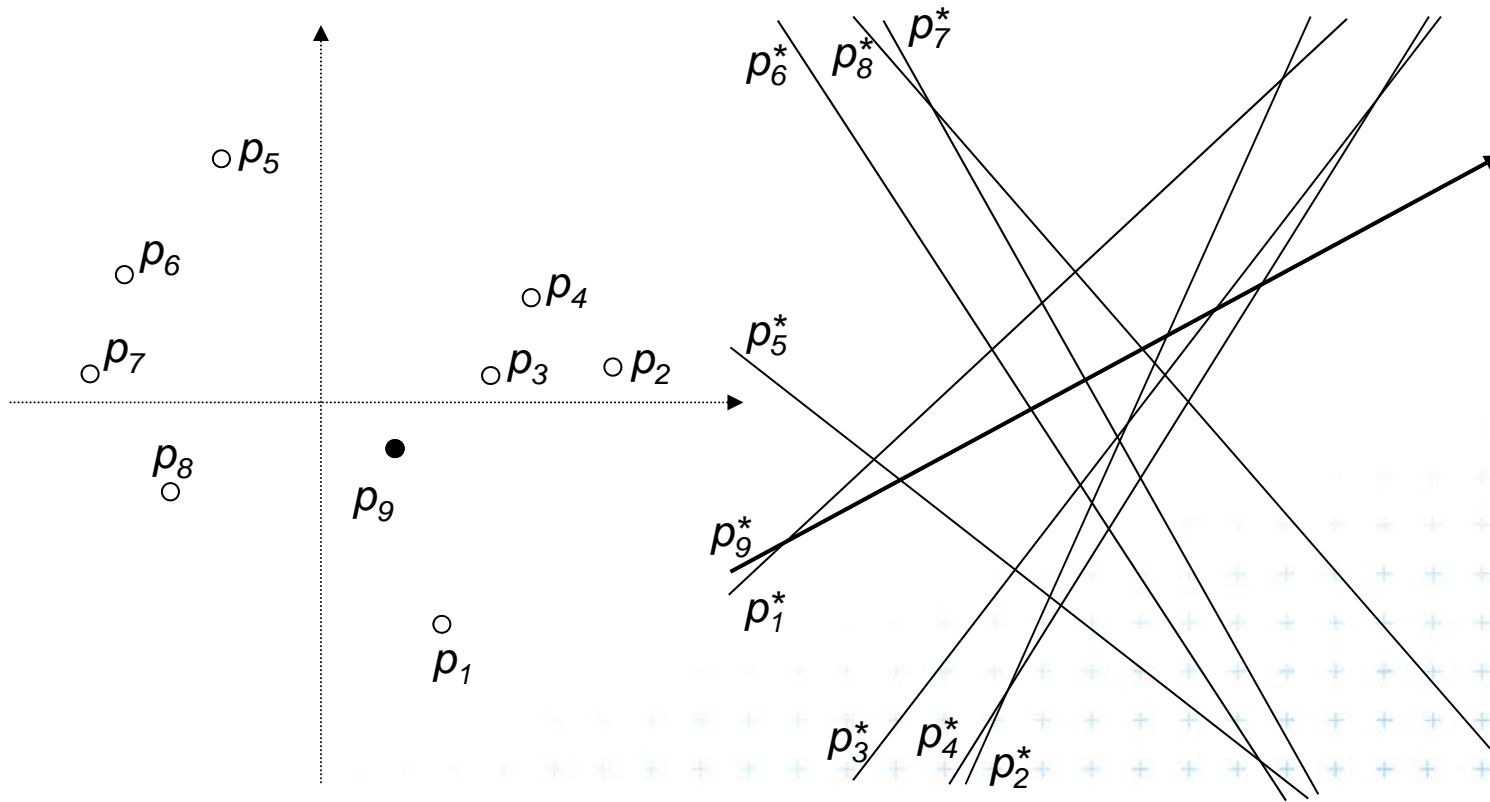
In dual plane



d) Angular sequence around p_9



d) Angular sequence around p_9

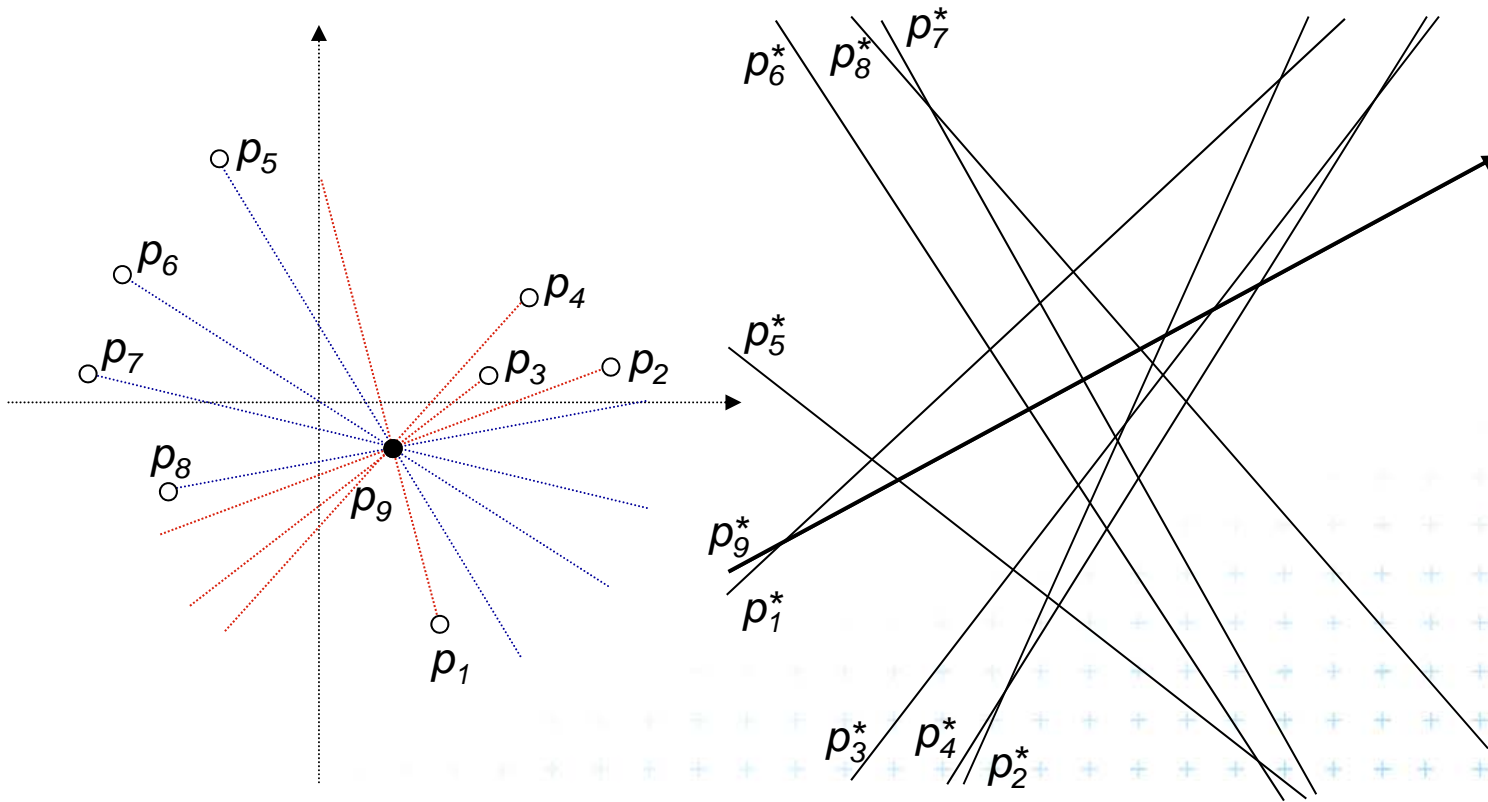


In primal plane

In dual plane

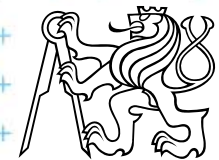


d) Angular sequence around p_9

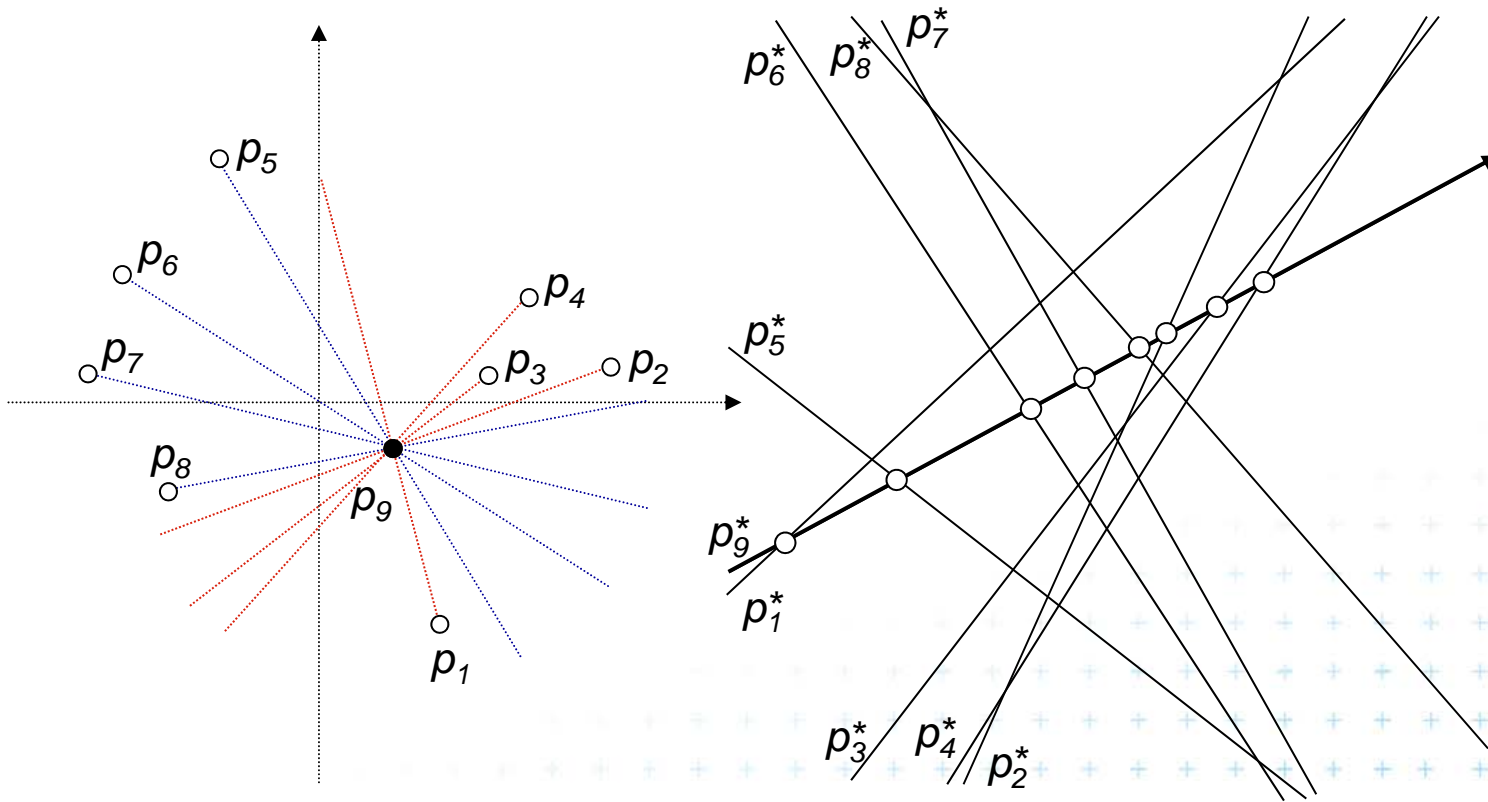


In primal plane

In dual plane



d) Angular sequence around p_9

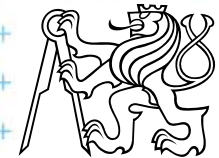
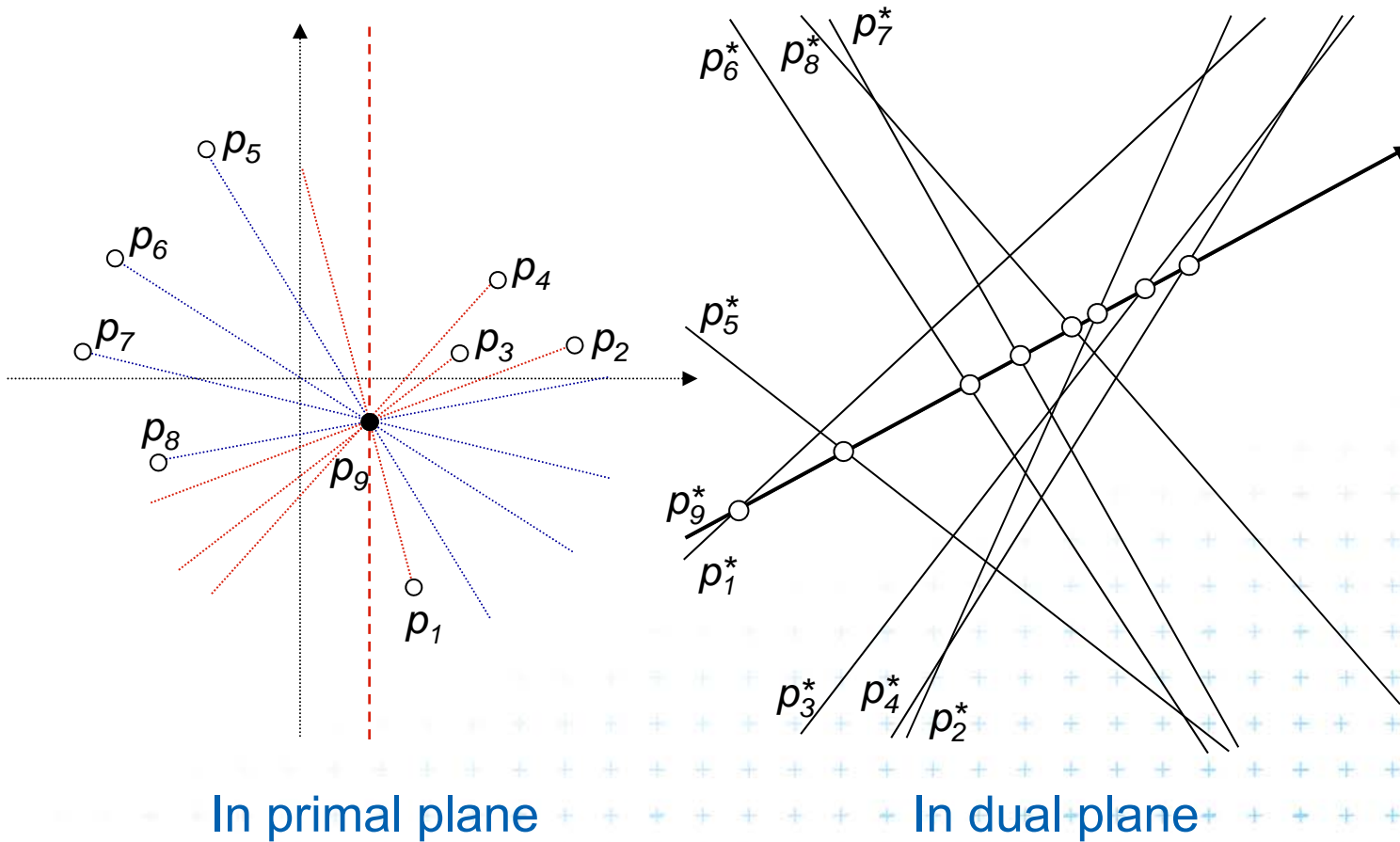


In primal plane

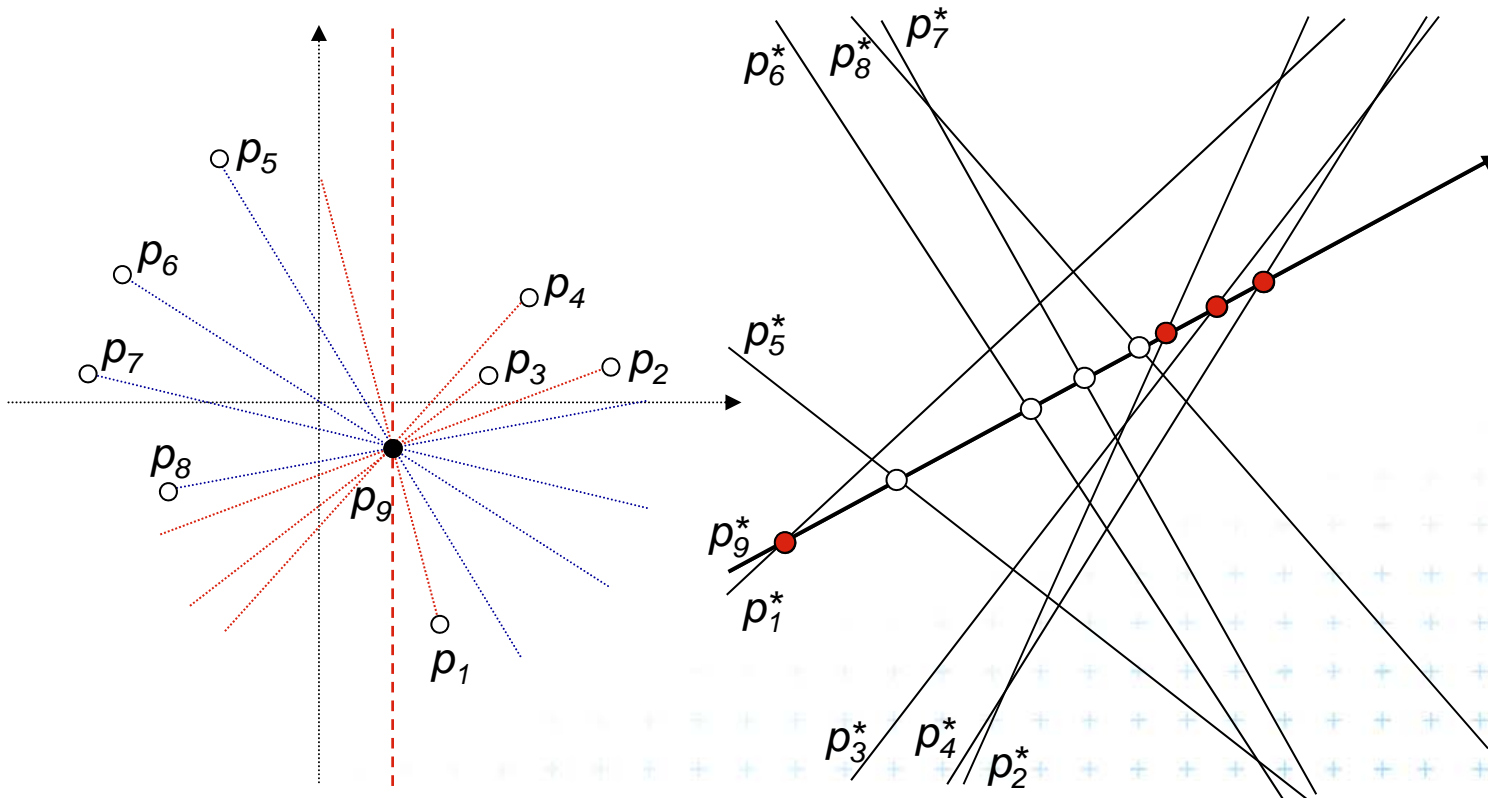
In dual plane



d) Angular sequence around p_9

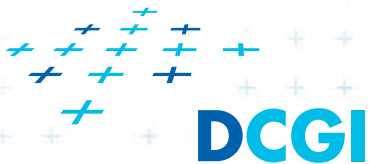


d) Angular sequence around p_9

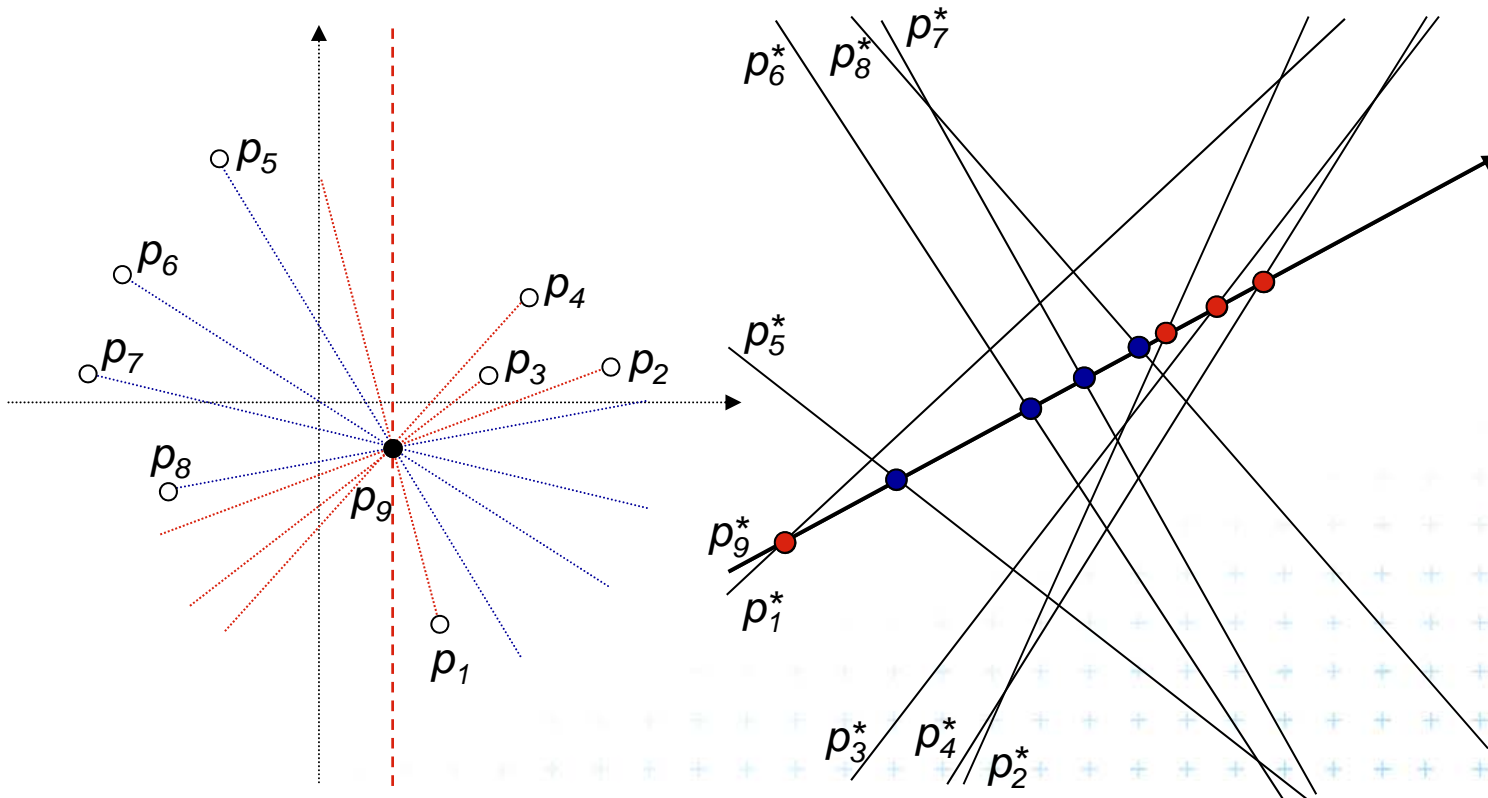


In primal plane

In dual plane



d) Angular sequence around p_9

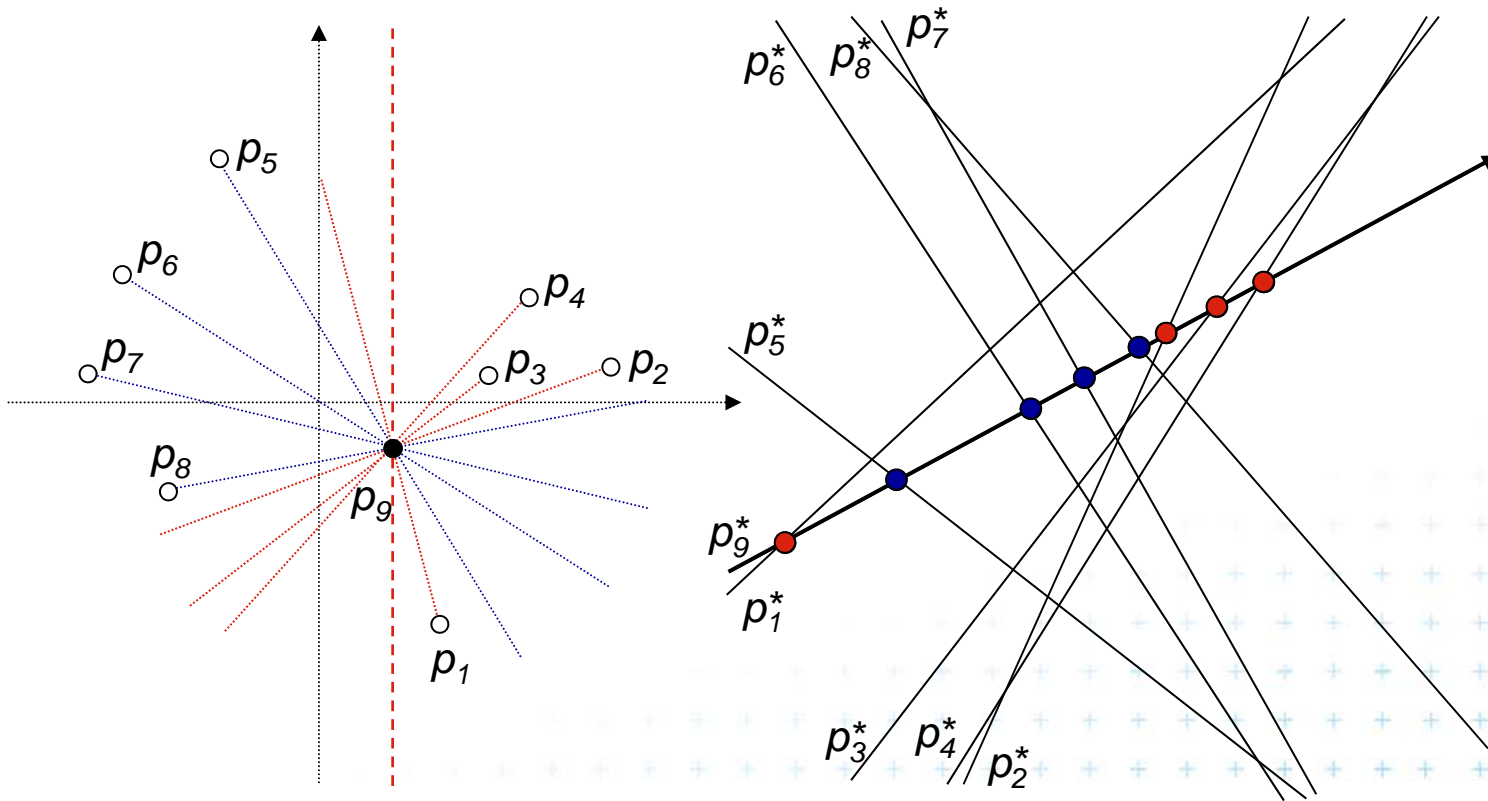


In primal plane

In dual plane



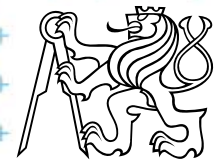
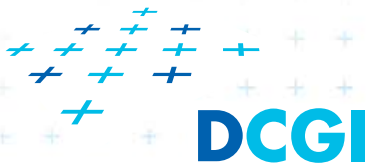
d) Angular sequence around p_9



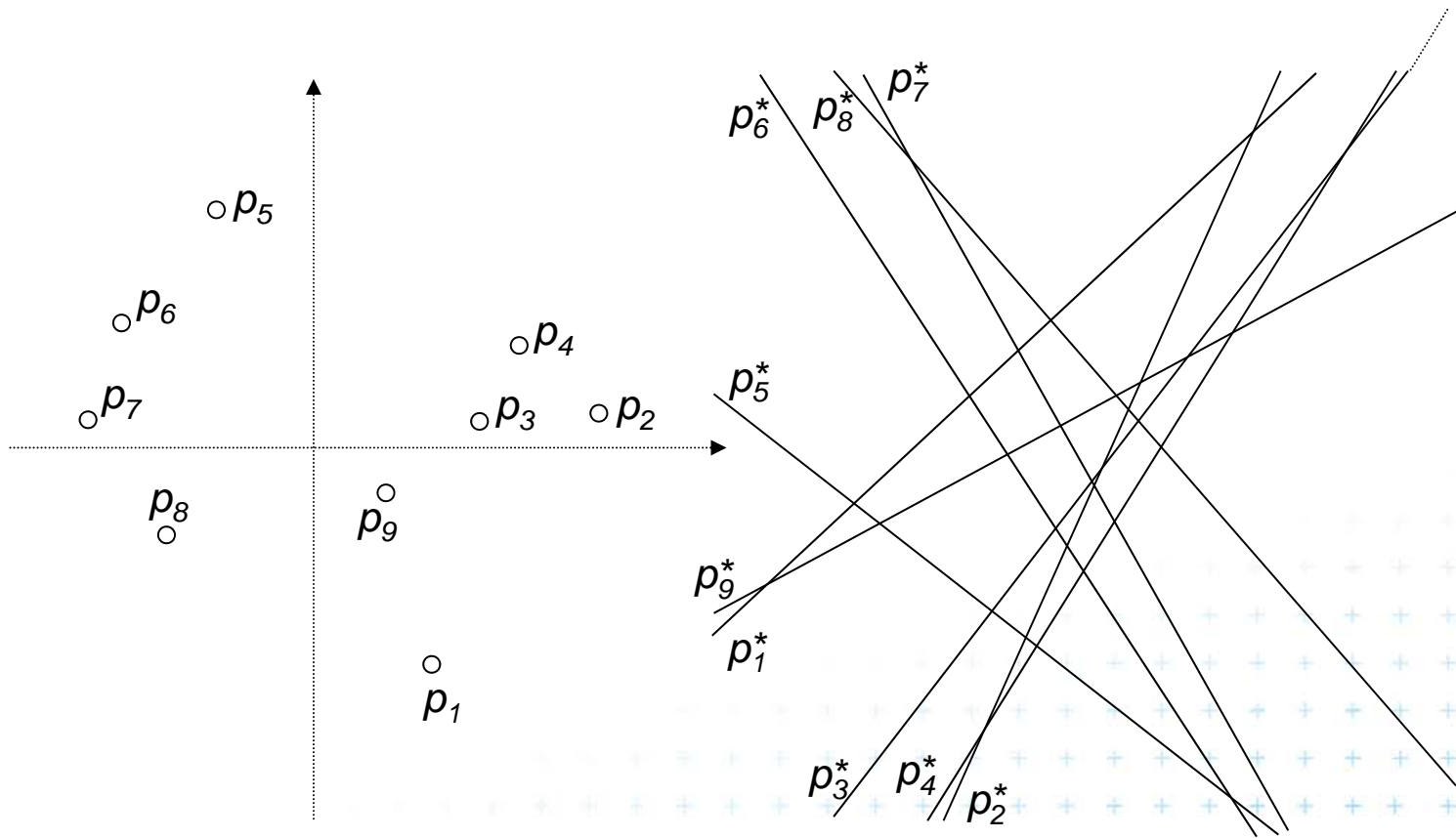
In primal plane

In dual plane

Point order around p_9 : $p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8$



d) Angular sequences around p_3

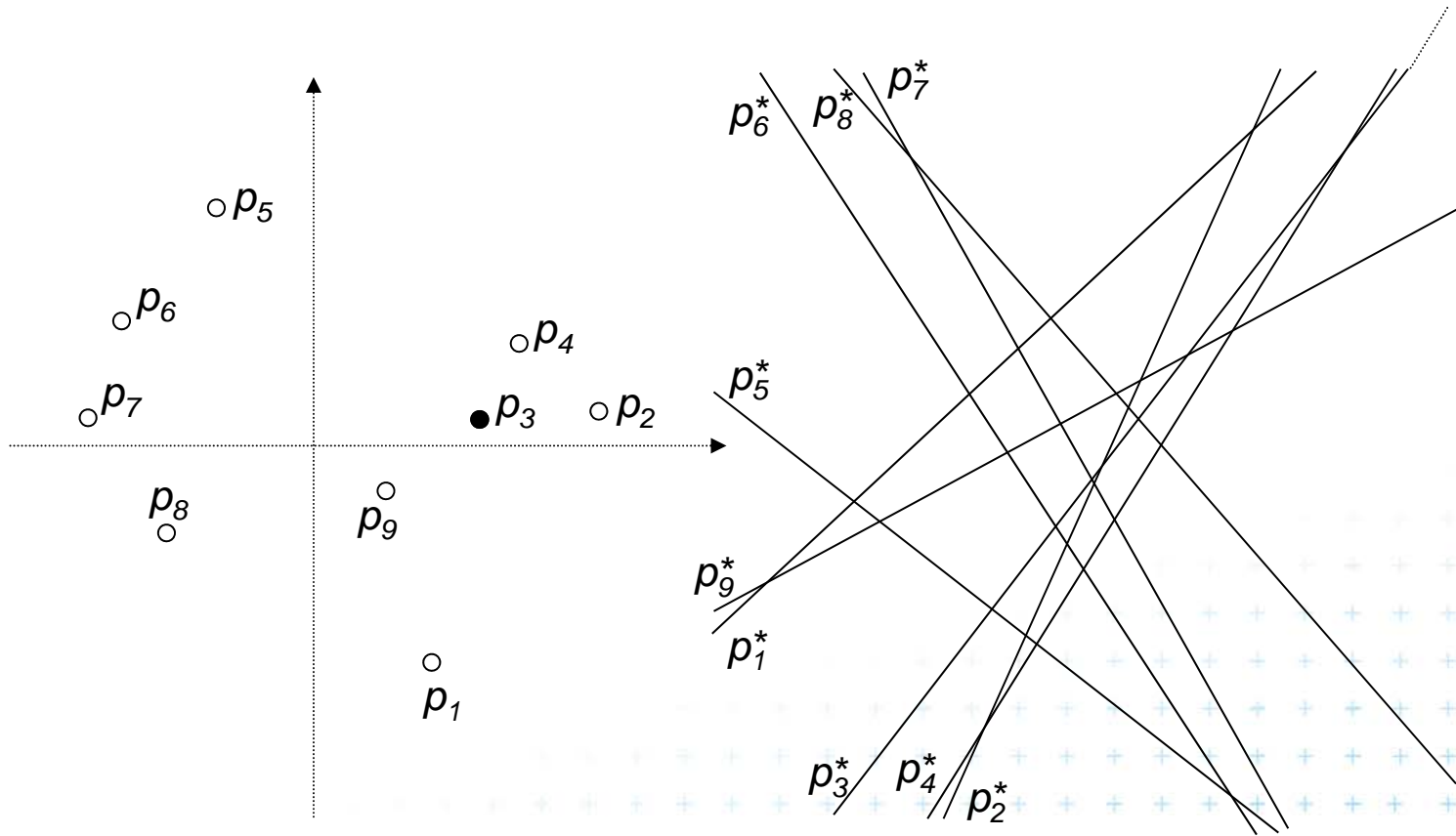


In primal plane

In dual plane



d) Angular sequences around p_3

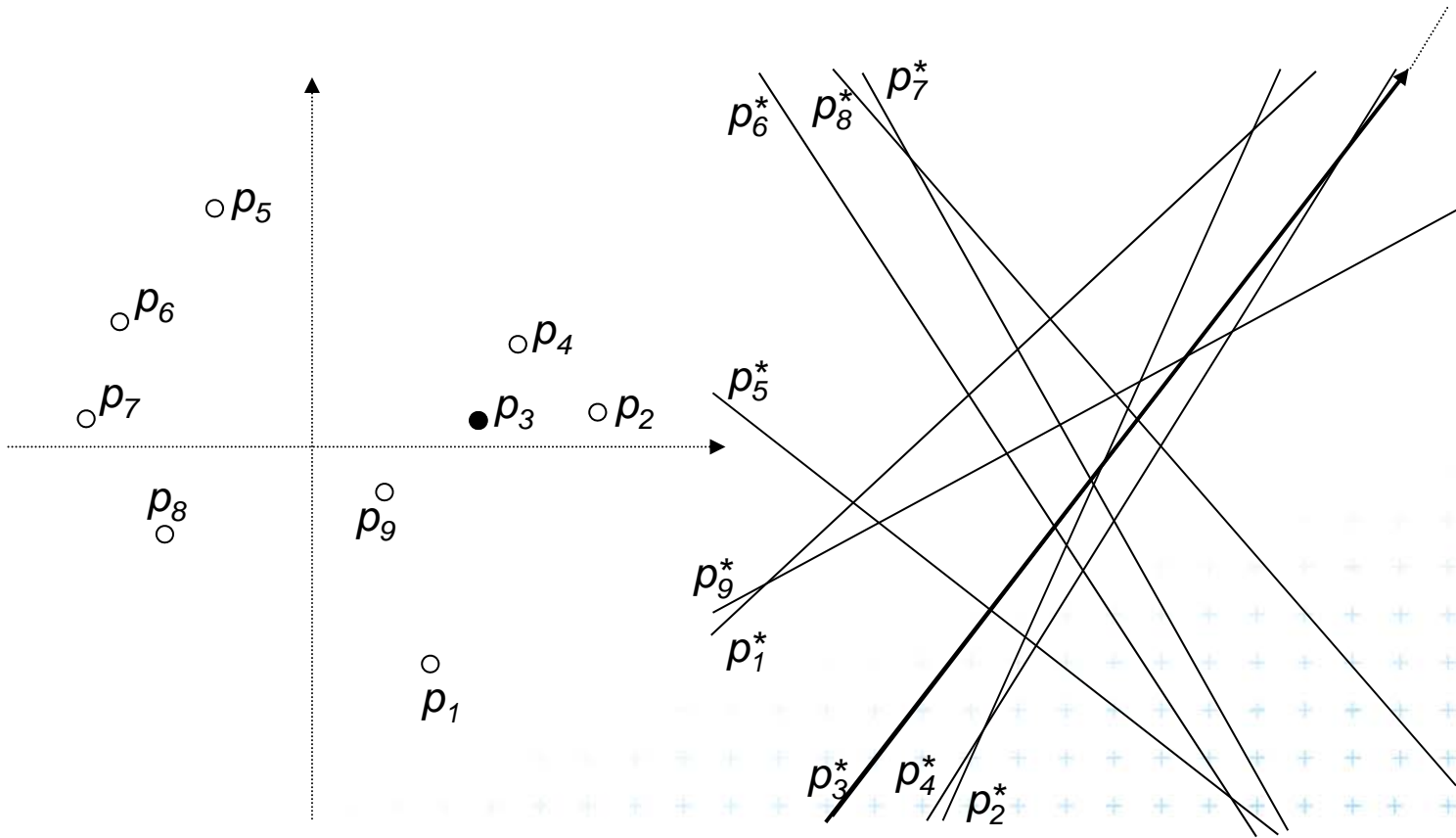


In primal plane

In dual plane



d) Angular sequences around p_3

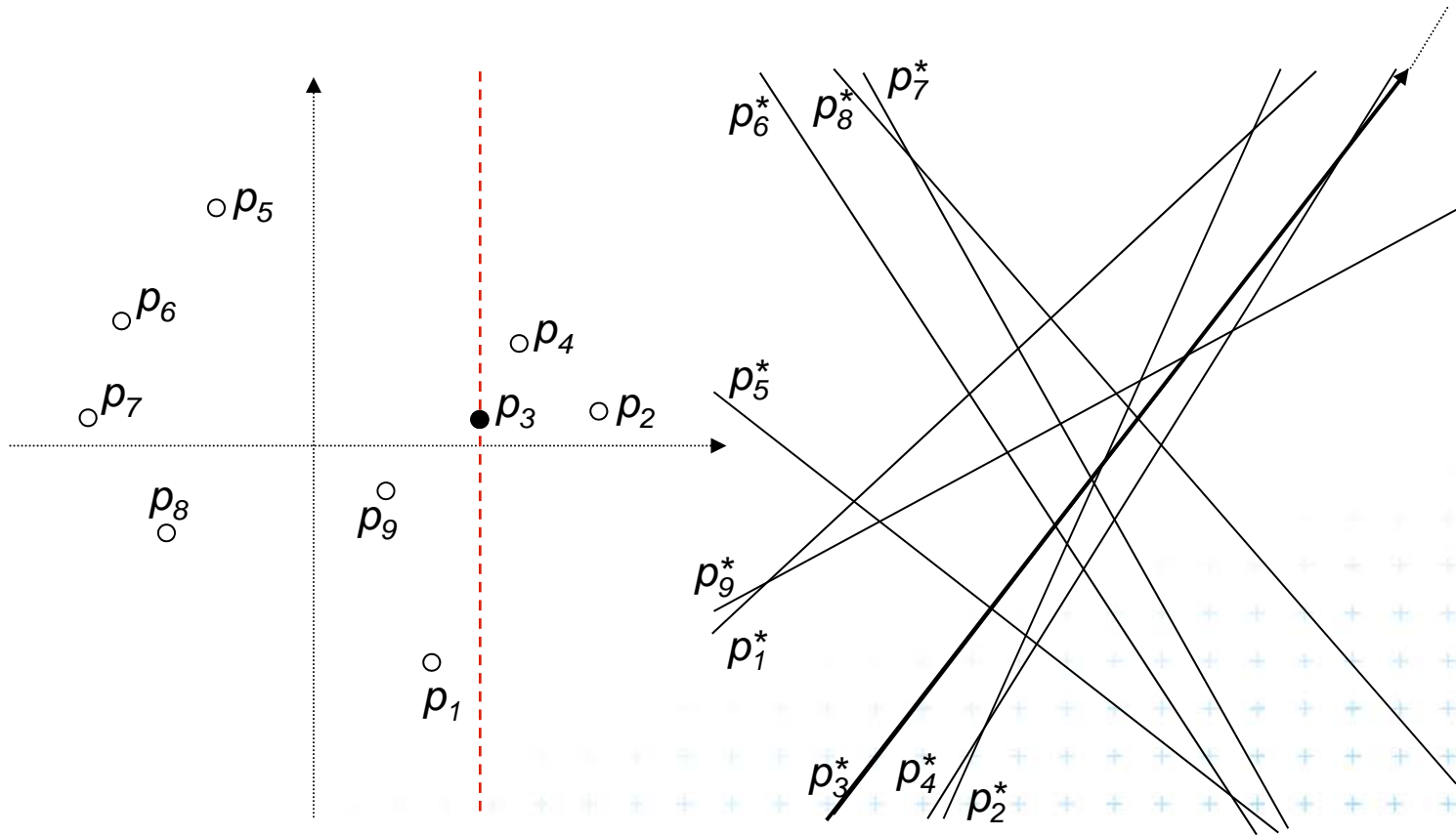


In primal plane

In dual plane



d) Angular sequences around p_3

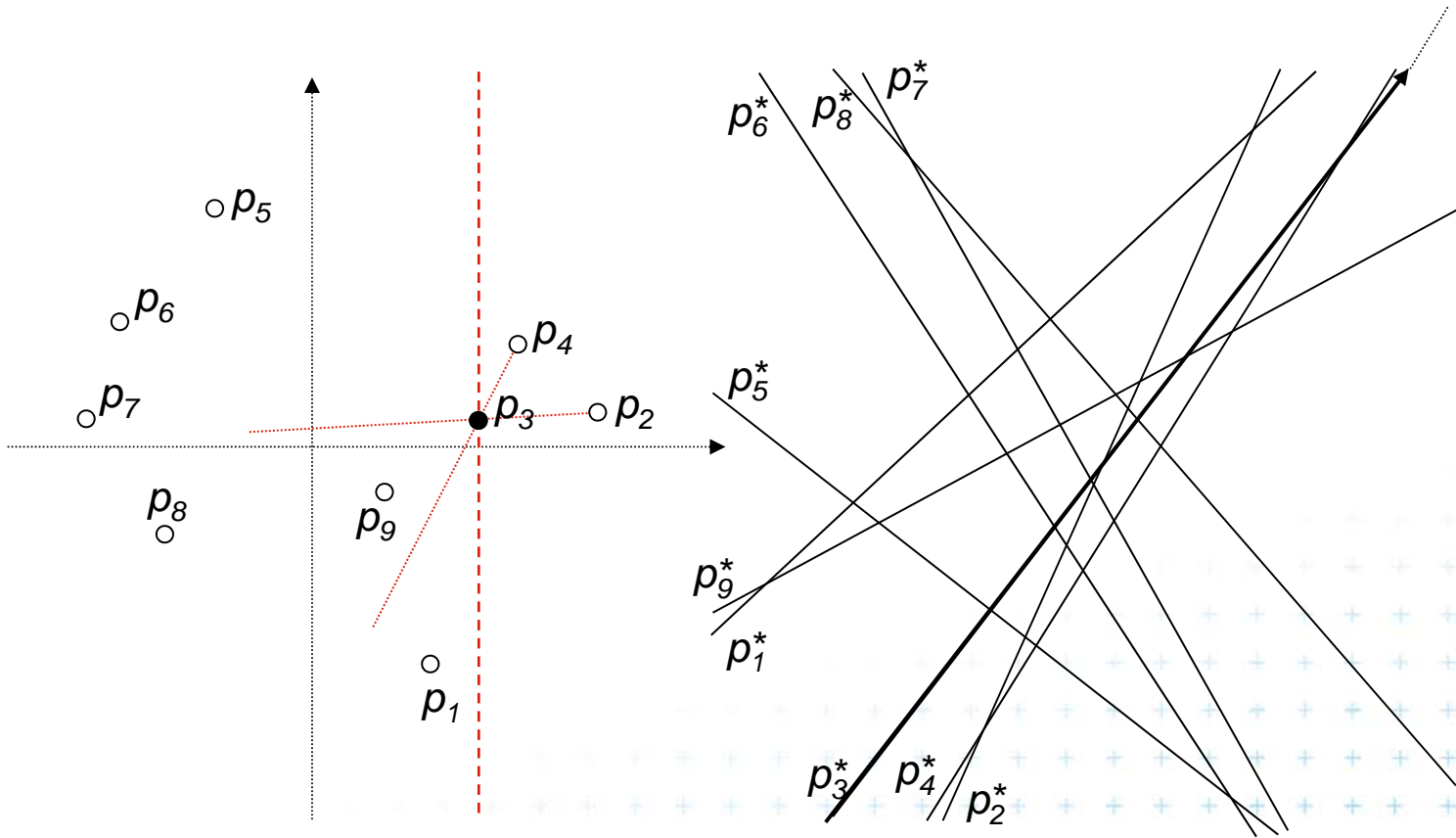


In primal plane

In dual plane

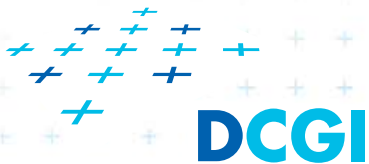


d) Angular sequences around p_3

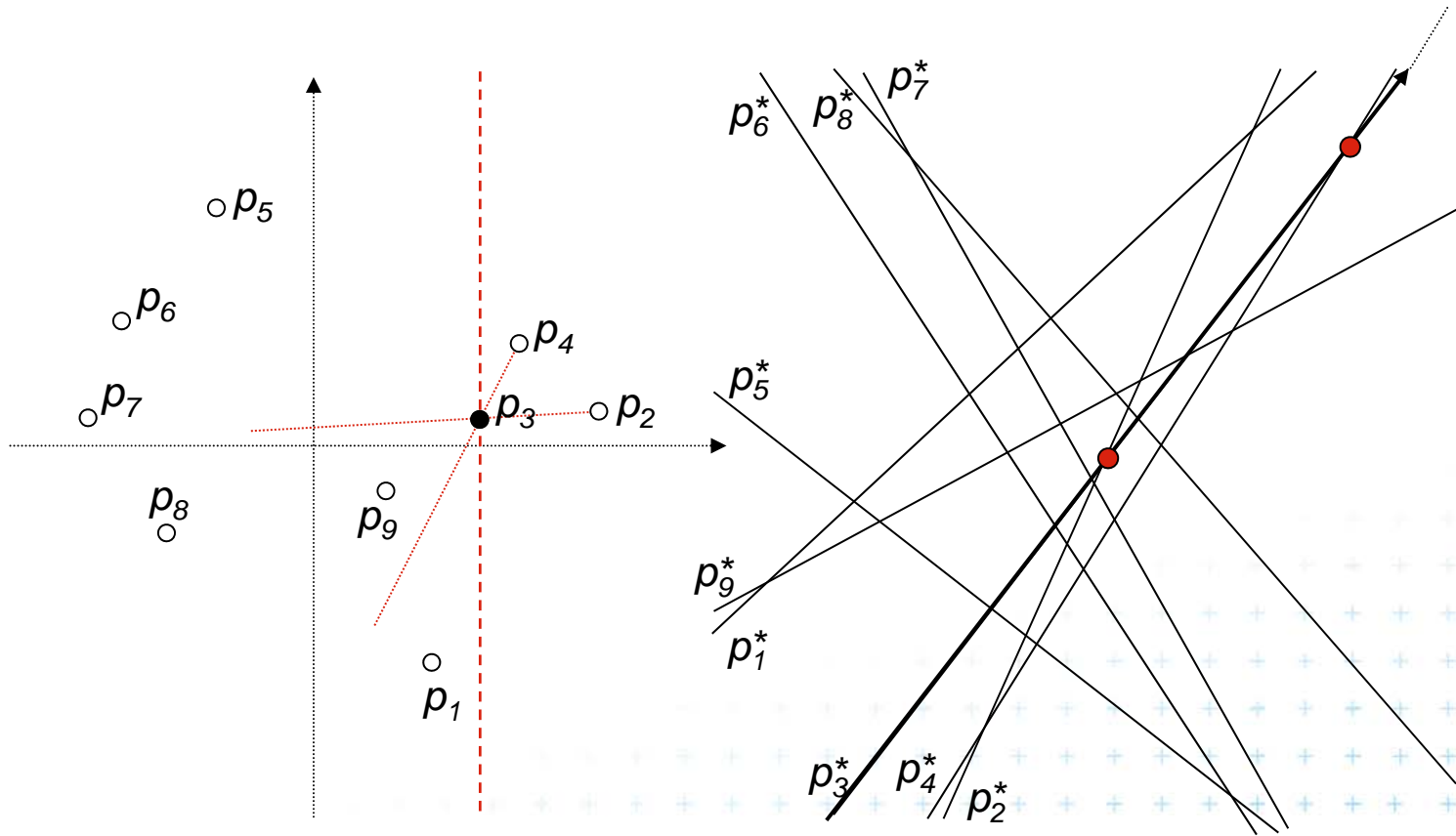


In primal plane

In dual plane

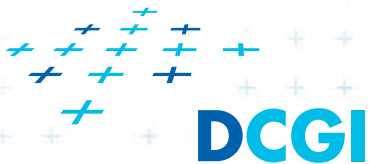


d) Angular sequences around p_3

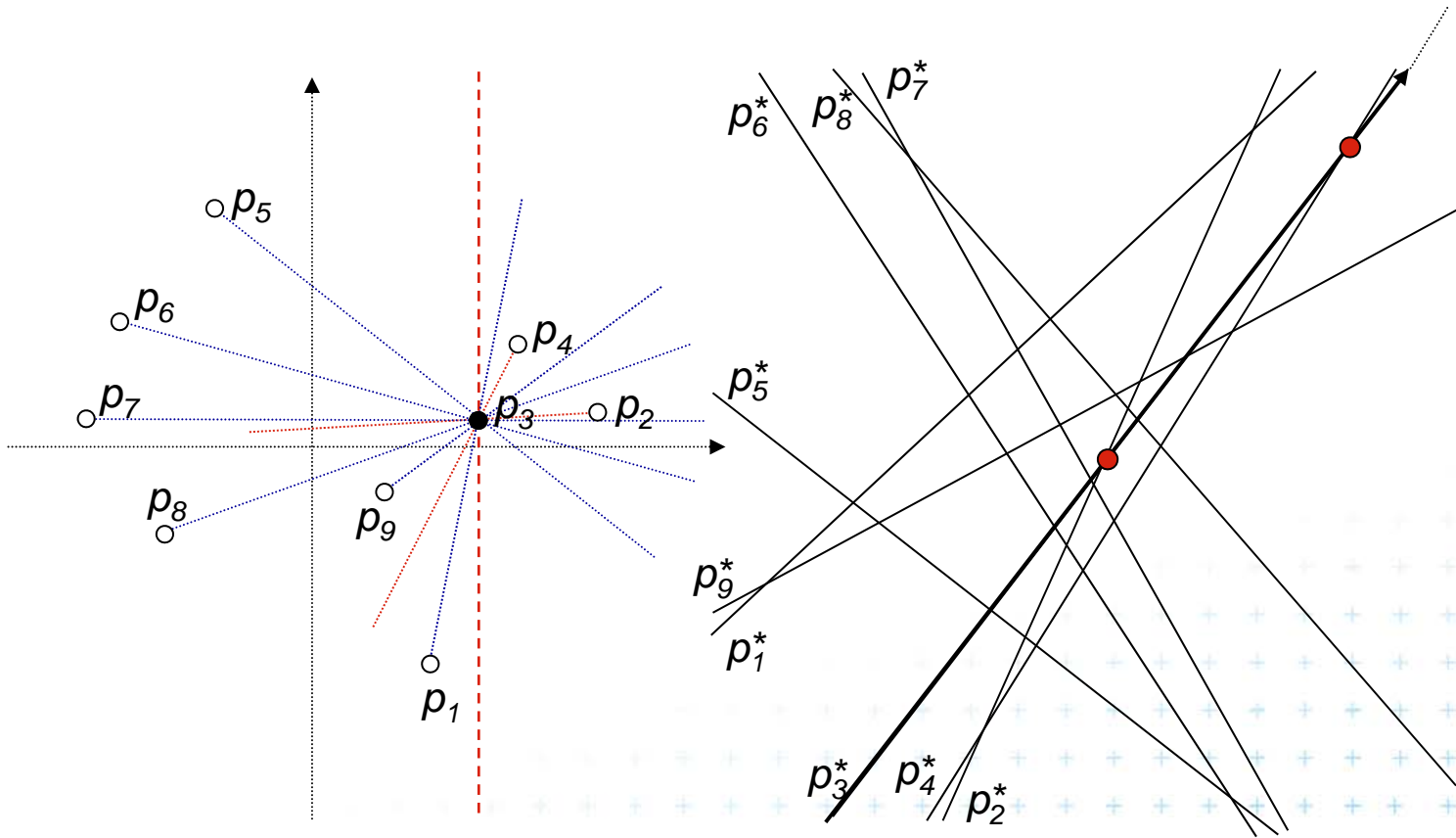


In primal plane

In dual plane



d) Angular sequences around p_3

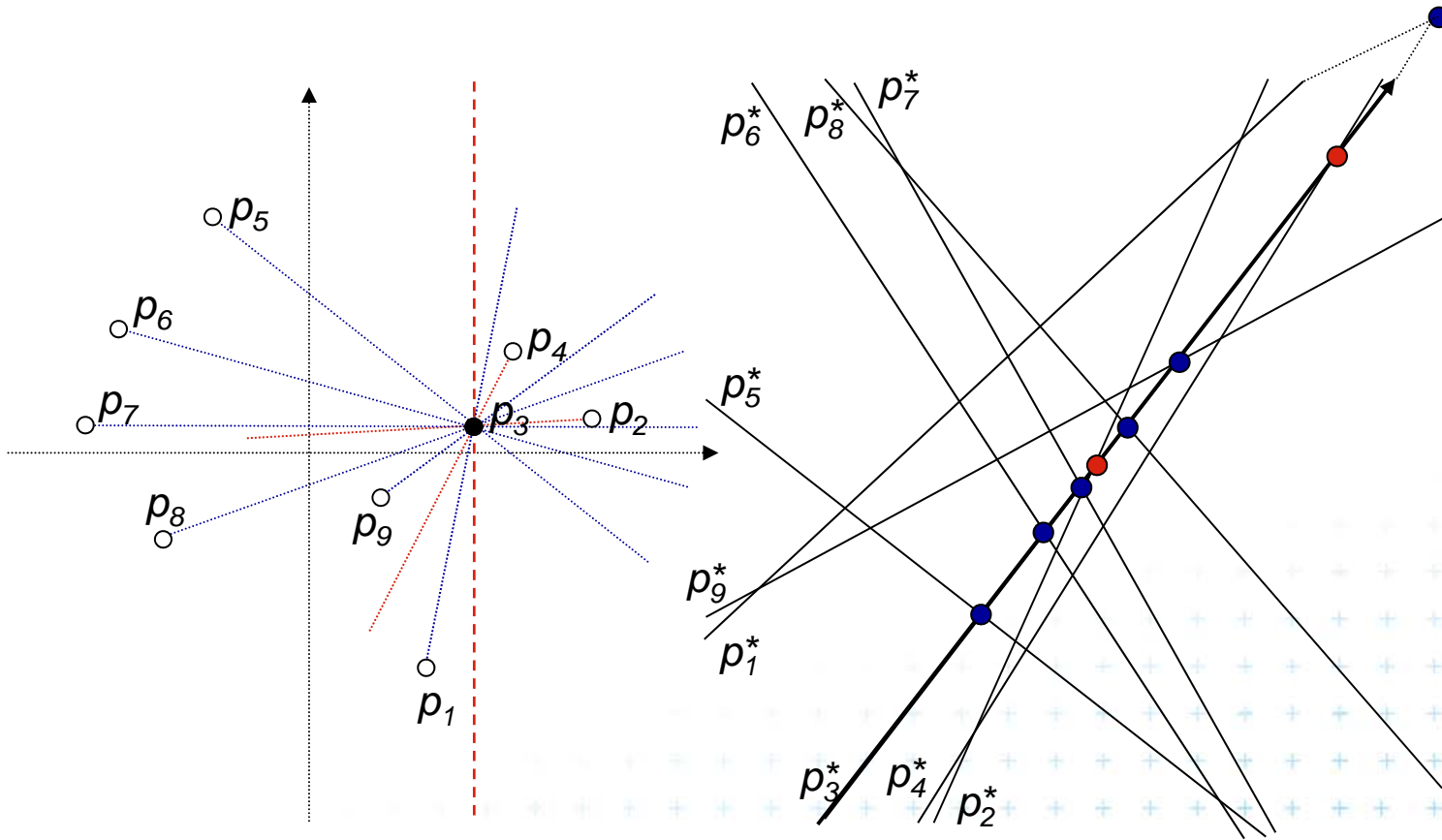


In primal plane

In dual plane

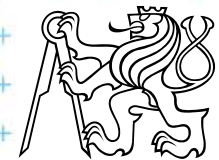


d) Angular sequences around p_3

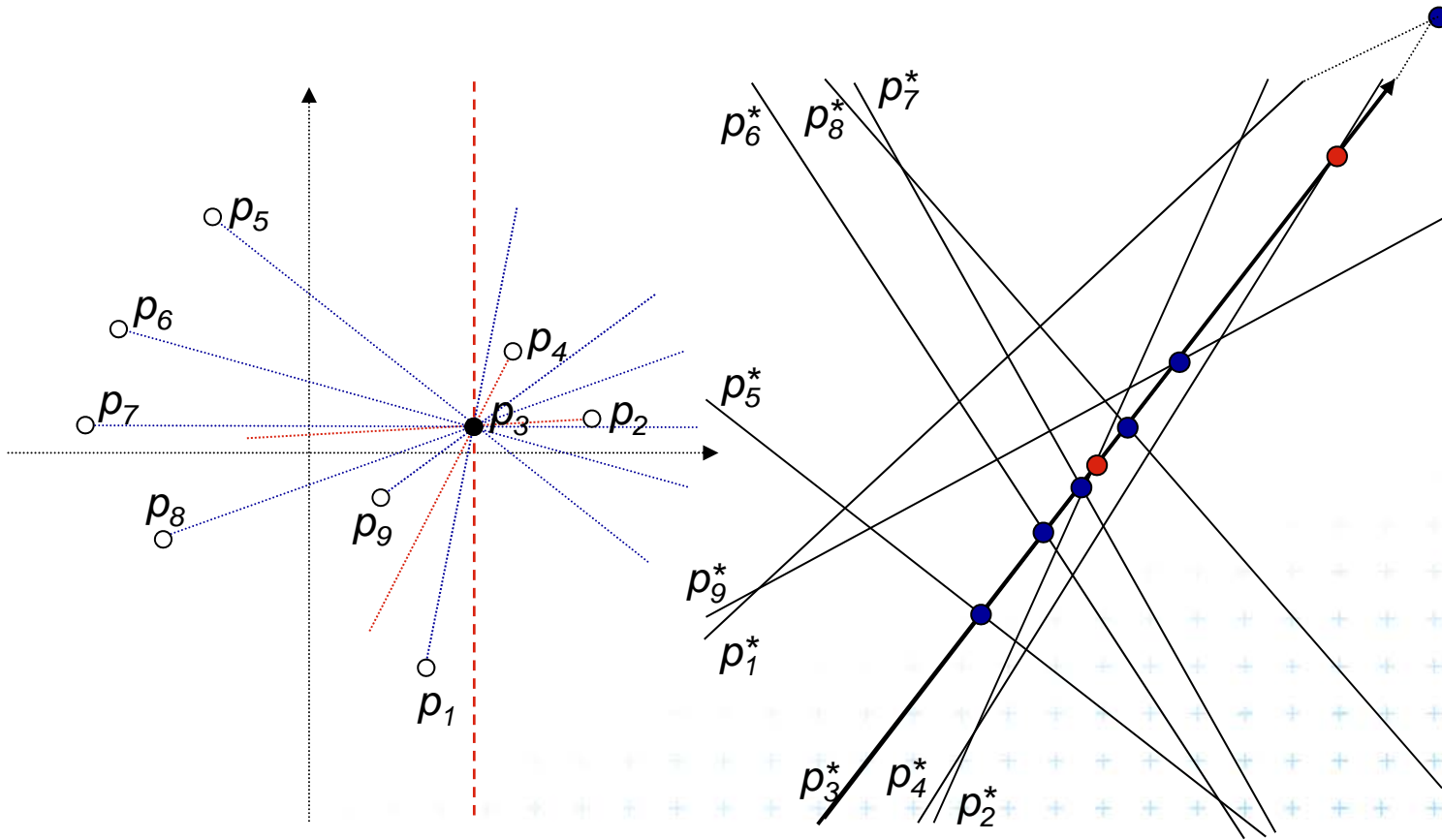


In primal plane

In dual plane



d) Angular sequences around p_3



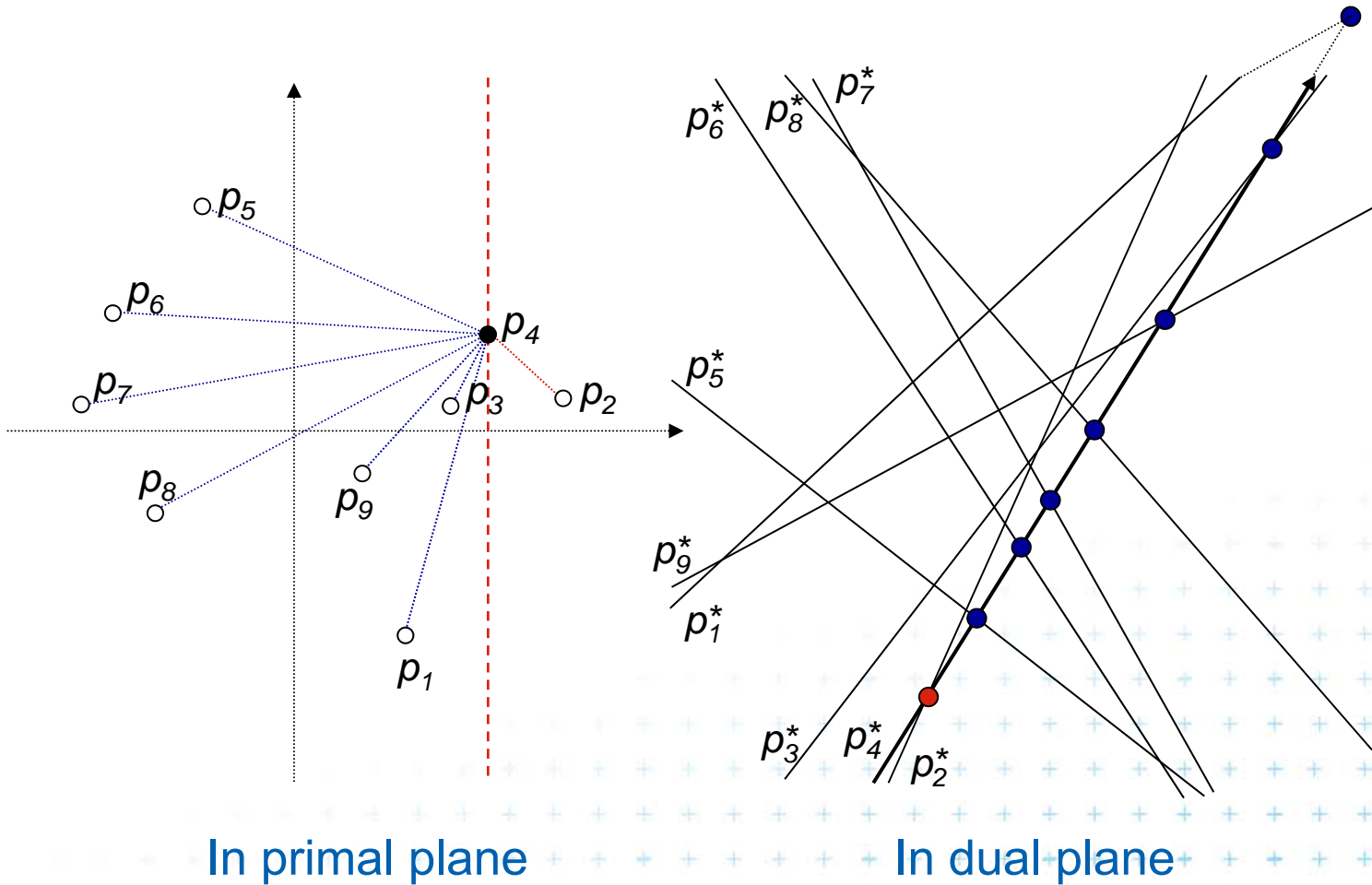
In primal plane

In dual plane

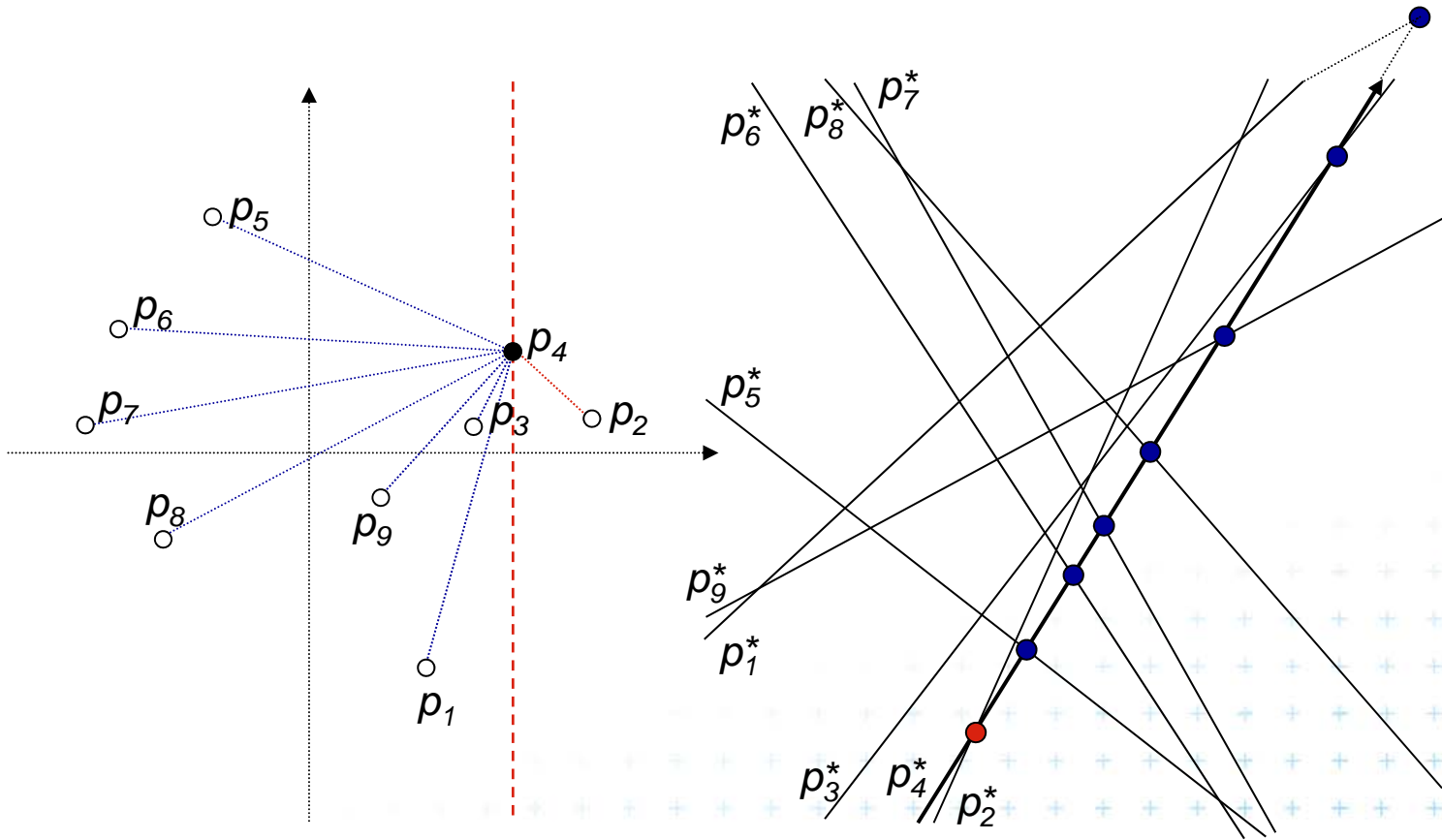
Point order around p_3 : $p_2, p_4, p_5, p_6, p_7, p_8, p_3, p_1$



d) Angular sequences around p_4



d) Angular sequences around p_4



In primal plane

In dual plane

Point order around p_4 : $p_2, p_5, p_6, p_7, p_8, p_9, p_3, p_1$



e) More applications of line arrangement

Visibility graph

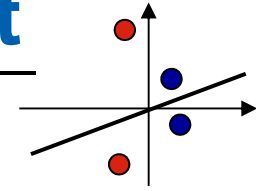
Given a set of n non-intersecting line segments, compute the *visibility graph*, whose vertices are the endpoints of the segments, and whose edges are pairs of visible endpoints (use angular sequences).

Maximum stabbing line

Given a set of n line segments in the plane, compute the line that stabs (intersects) the maximum number of these line segments.



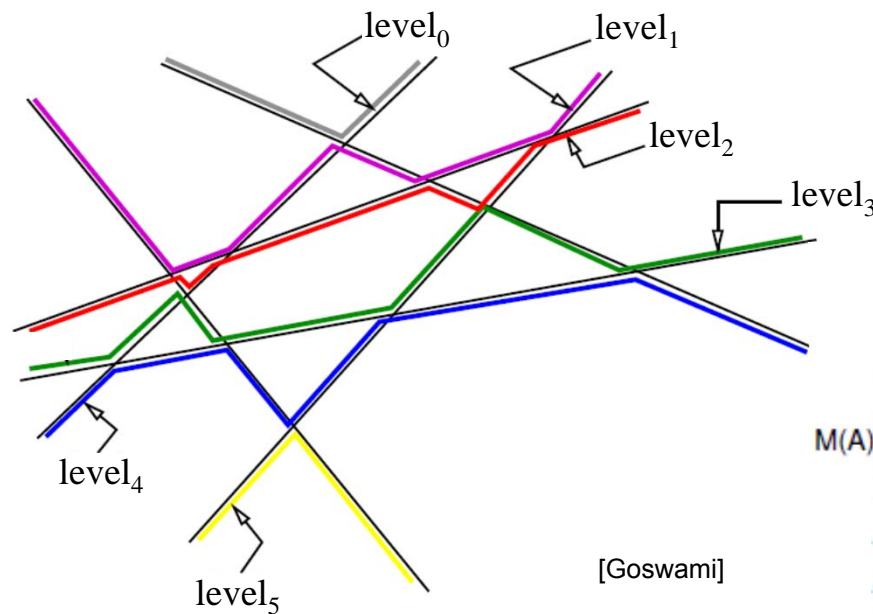
More applications of line arrangement



Ham-Sandwich cut

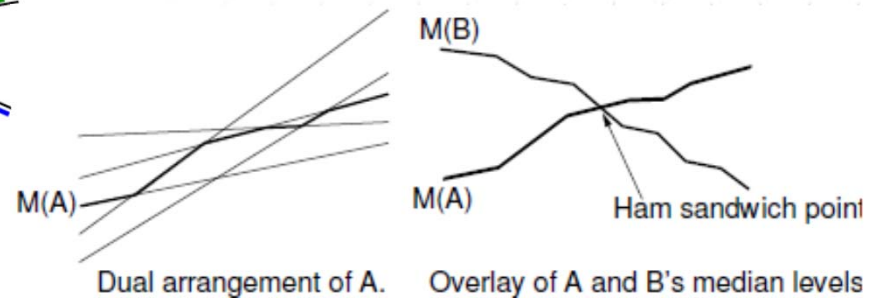
Given two sets of points, n red and m blue points compute a **single line that simultaneously bisects both sets**

Principle – intersect middle levels of arrangements



[Goswami]

Point at k -th level L_k has
at most k lines above and
at most $n - k - 1$ lines below



Dual arrangement of A.

Overlay of A and B's median levels

[Mount]



DCGI



References

- [Berg] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars:
Computational Geometry: Algorithms and Applications, Springer-Verlag,
3rd rev. ed. 2008. 386 pages, 370 fig. ISBN: 978-3-540-77973-5,
Chapters 8., <http://www.cs.uu.nl/geobook/>
- [Mount] David Mount, - CMSC 754: Computational Geometry, Lecture Notes for
Spring 2007, University of Maryland, Lectures 8,15,16,31, and 32.
<http://www.cs.umd.edu/class/spring2007/cmsc754/lectures.shtml>
- [applet] Allen K. L. Miu: Duality Demo
<http://nms.lcs.mit.edu/~aklmiu/6.838/dual/>
- [Goswami] Partha P. Goswami: Duality Transformation and its Application to
Computational Geometry, University of Calcutta, India
<http://www.tcs.tifr.res.in/~igga/lectureslides/partha-lec-iisc-jul09.pdf>

