Yan-Bin Jia, Iowa State University

Line Arrangements

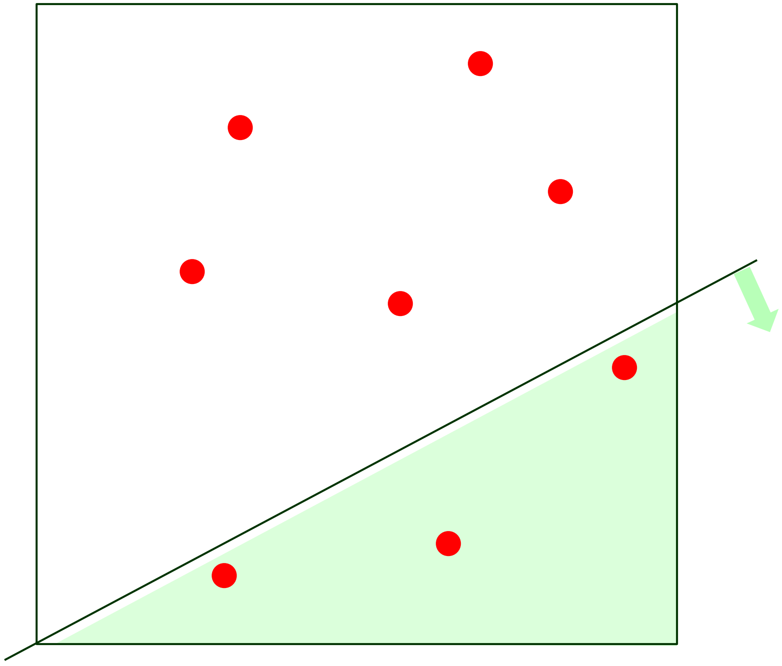
Outline:

I. Geometric complexity of a line arrangement

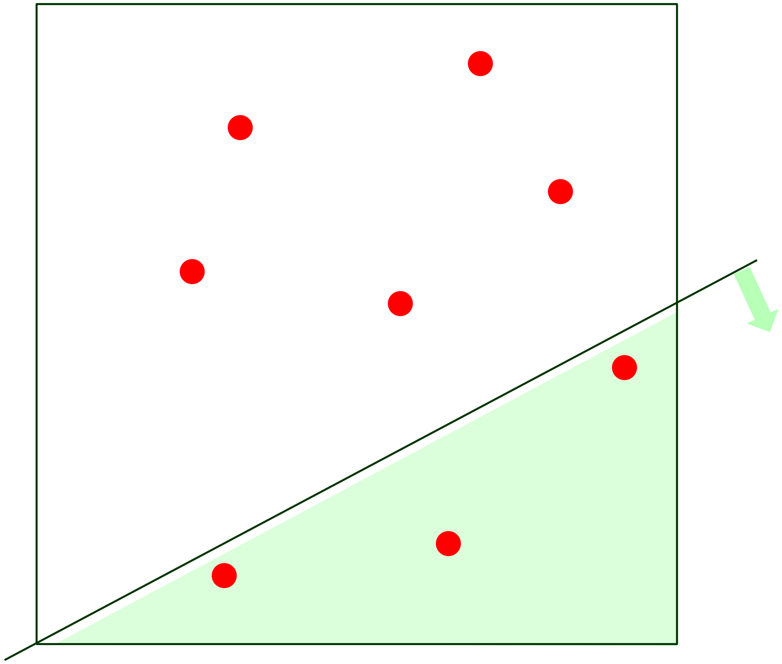
II. Incremental construction

III. Computation of the discrete discrepancy measure

The Discrepancy Problem

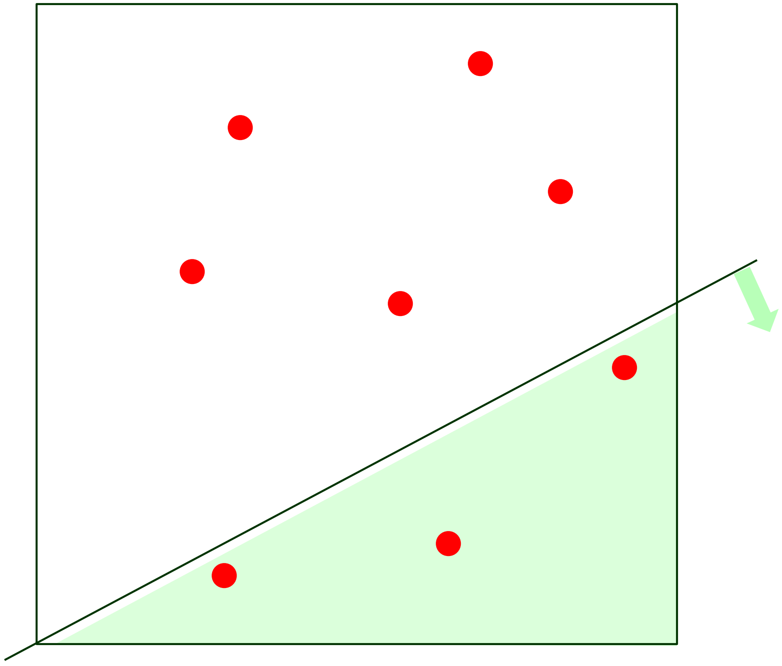


The Discrepancy Problem



Continuous measure: $\mu(h) = \frac{1}{4}$

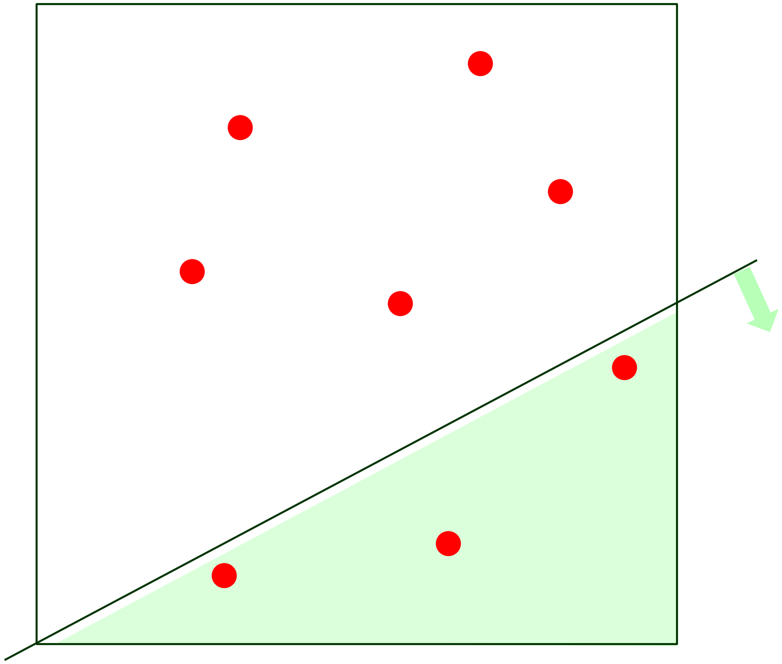
The Discrepancy Problem



Continuous measure: $\mu(h) = \frac{1}{4}$

Discrete measure: $\mu_s(h) = \frac{3}{8}$

The Discrepancy Problem

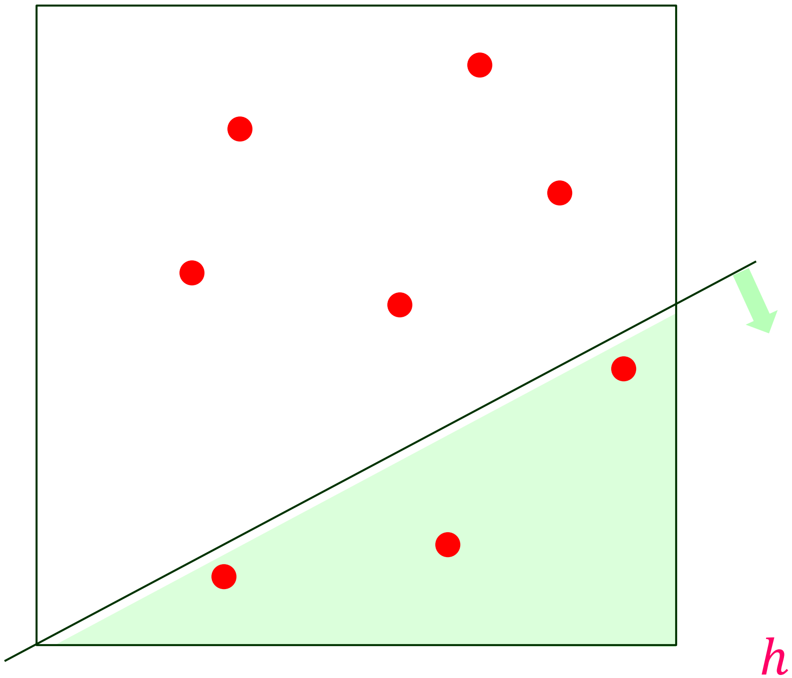


Continuous measure: $\mu(h) = \frac{1}{4}$

Discrete measure: $\mu_s(h) = \frac{3}{8}$

Maximize $|\mu_s(h) - \mu(h)|$

The Discrepancy Problem



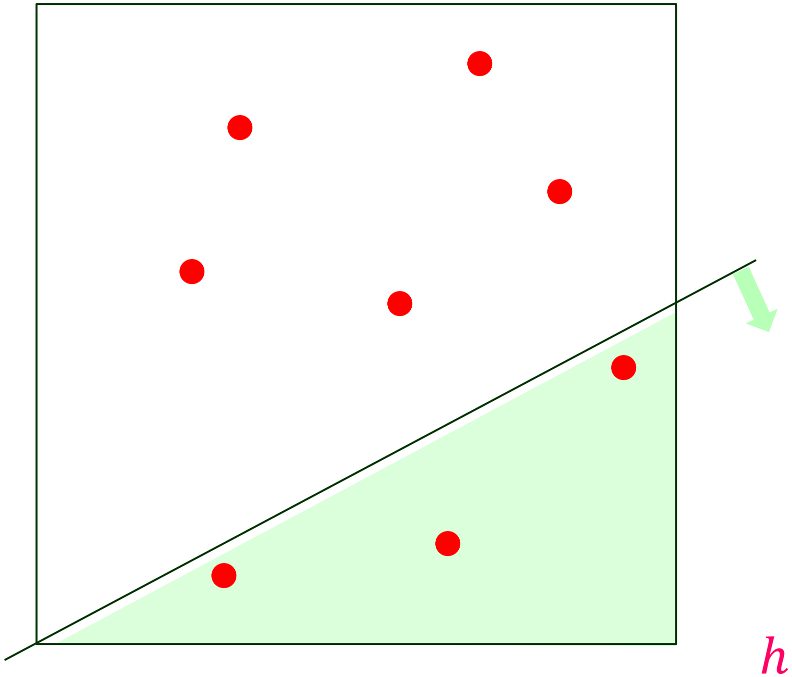
Continuous measure: $\mu(h) = \frac{1}{4}$

Discrete measure: $\mu_s(h) = \frac{3}{8}$

Maximize $|\mu_s(h) - \mu(h)|$

h must have ≥ 1 points on its boundary.

The Discrepancy Problem



Continuous measure: $\mu(h) = \dfrac{1}{4}$

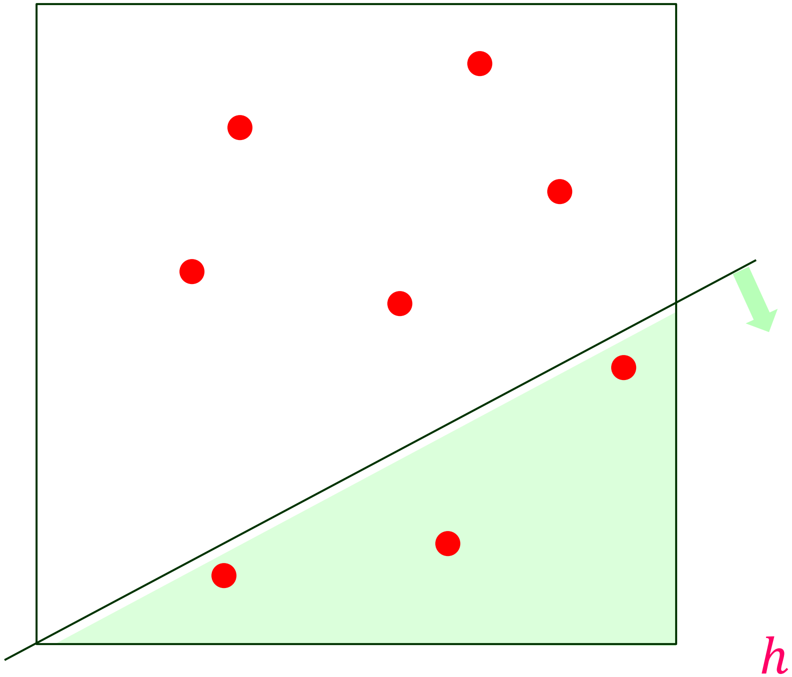
Discrete measure: $\mu_s(h) = \dfrac{3}{8}$

Maximize $|\mu_s(h) - \mu(h)|$

h must have ≥ 1 points on its boundary.

☆ exactly one point (Type i) $\Rightarrow$ brute-force method $O(n^2)$

The Discrepancy Problem



Continuous measure: $\mu(h) = \frac{1}{4}$

Discrete measure: $\mu_s(h) = \frac{3}{8}$

Maximize $|\mu_s(h) - \mu(h)|$

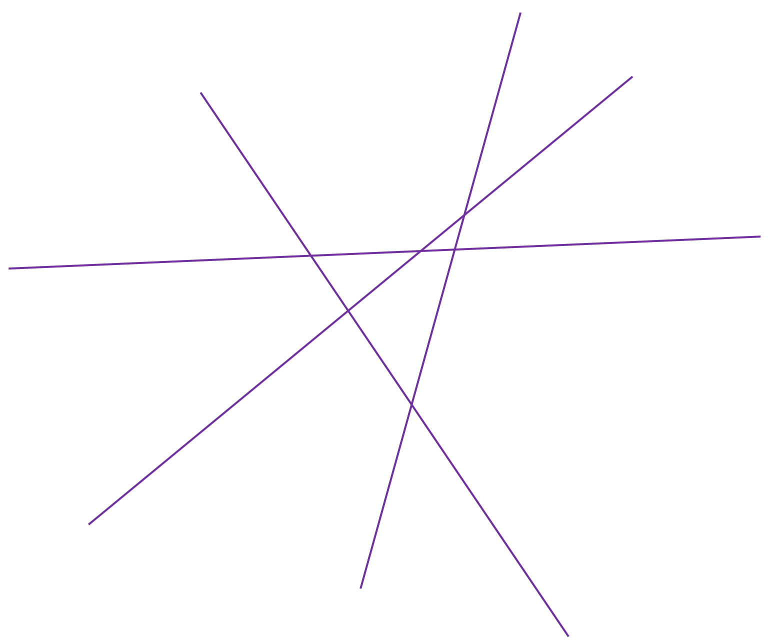
h must have ≥ 1 points on its boundary.

☆ exactly one point (Type i) $\Rightarrow$ brute-force method $O(n^2)$

☆ at least two points (Type ii) $\Rightarrow$ apply duality + line arrangement

I. Arrangement of Lines

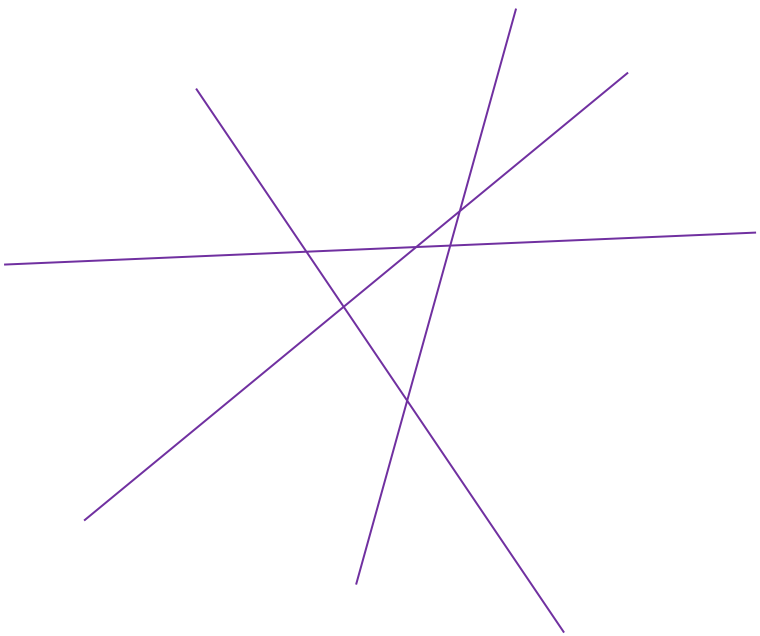
L: a set of n lines.



I. Arrangement of Lines

L: a set of n lines.

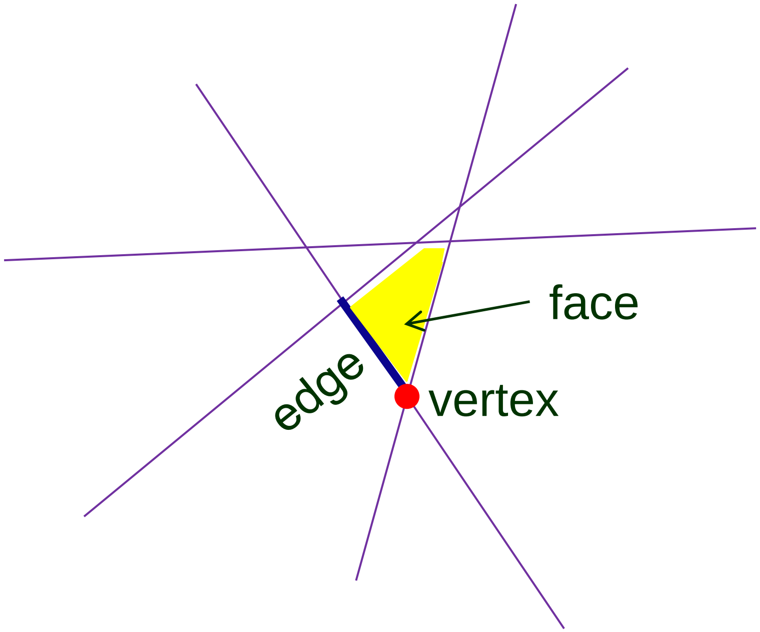
$A(L)$: planar subdivision induced by L.



I. Arrangement of Lines

L: a set of n lines.

$A(L)$: planar subdivision induced by L.

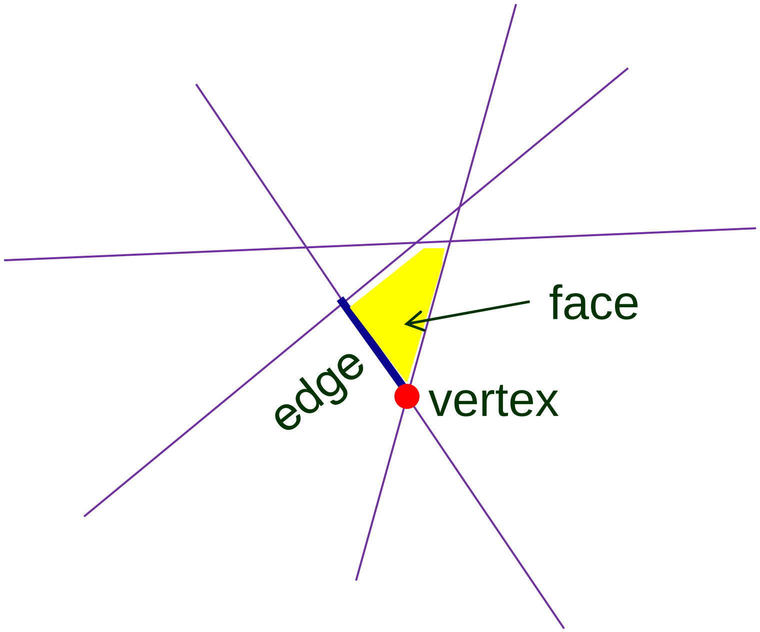


I. Arrangement of Lines

L: a set of n lines.

$A(L)$: planar subdivision induced by L.

- with unbounded edges and faces



I. Arrangement of Lines

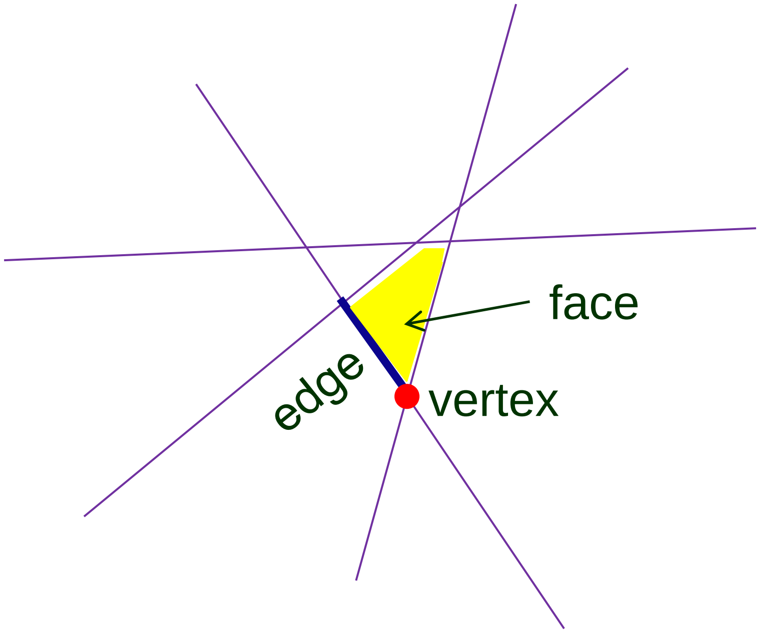
L: a set of n lines.

$A(L)$: planar subdivision induced by L.

- with unbounded edges and faces

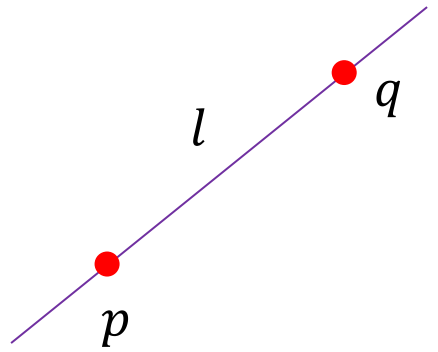
Simple arrangement if

- no three lines are concurrent;
- no two lines are parallel.

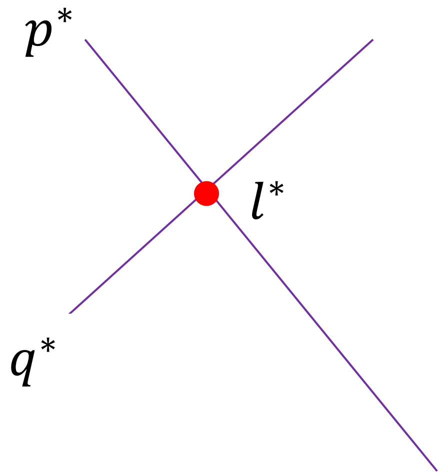


Reduction to Line Arrangment

Problem on points $\Longrightarrow$ problem on an arrangement of dual lines.



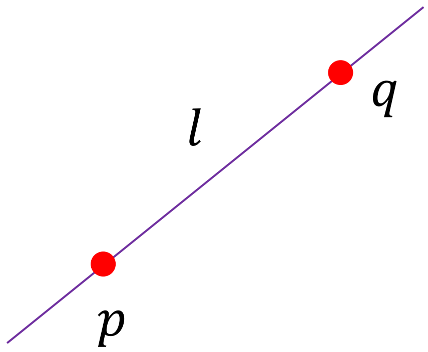
primal plane



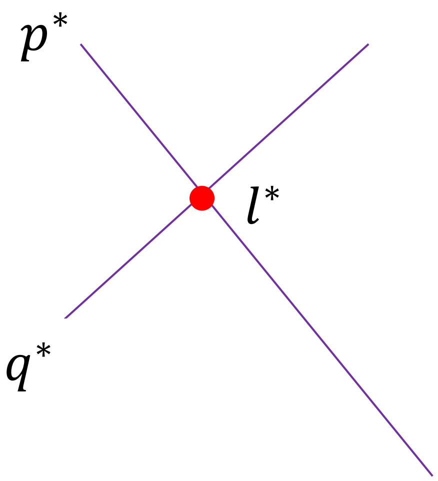
dual plane

Reduction to Line Arrangment

Problem on points $\Longrightarrow$ problem on an arrangement of dual lines.



primal plane

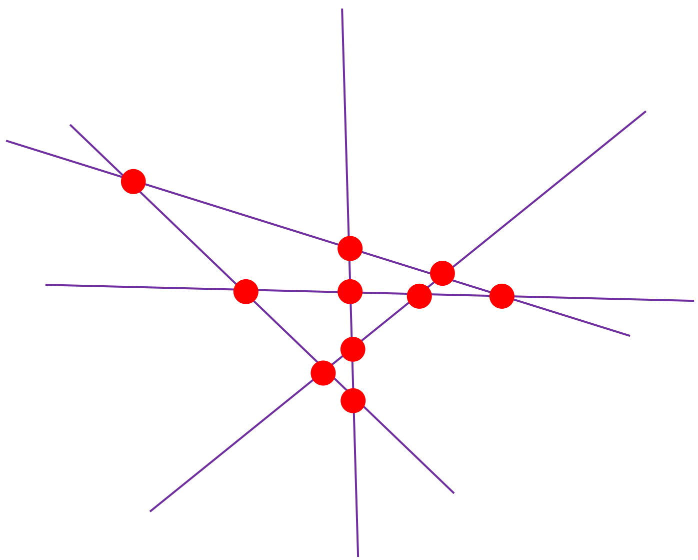


dual plane

Structure of a line arrangement is ***more apparent*** than that of a point set.

Combinatorial Complexity

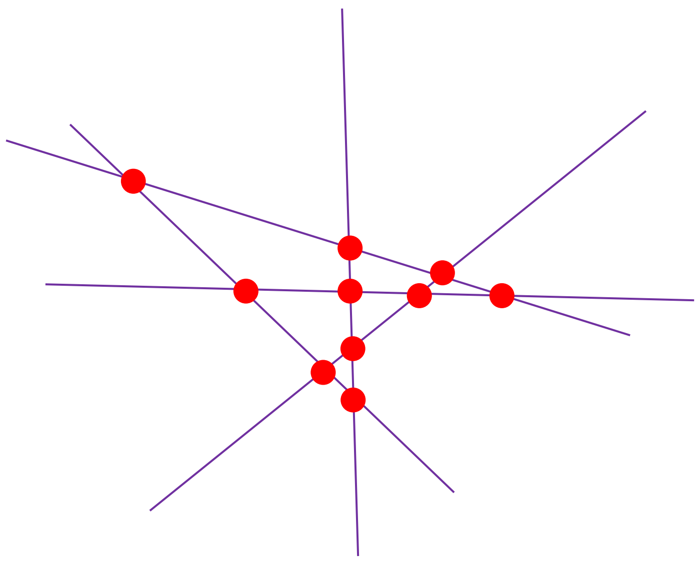
#vertices + #edges + #faces



Combinatorial Complexity

#vertices + #edges + #faces

Theorem #vertices $\leq \frac{n(n-1)}{2}$

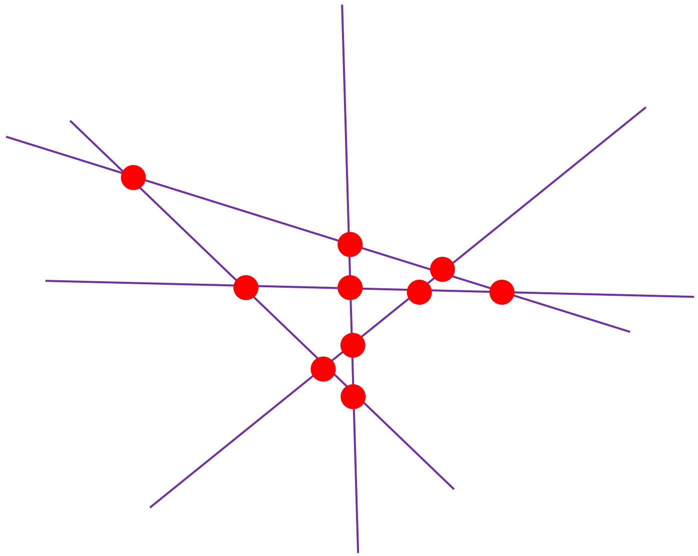


Combinatorial Complexity

#vertices + #edges + #faces

Theorem #vertices $\leq \frac{n(n-1)}{2}$

#edges $\leq n^2$



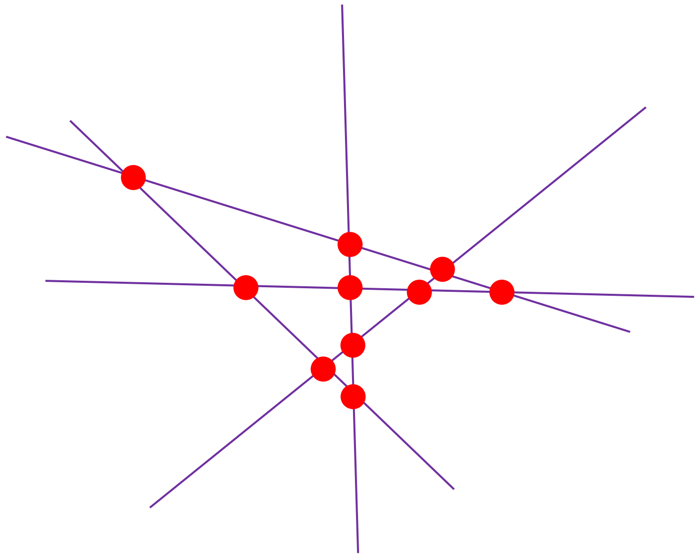
Combinatorial Complexity

#vertices + #edges + #faces

Theorem #vertices $\leq \frac{n(n-1)}{2}$

#edges $\leq n^2$

#faces $\leq \frac{n^2}{2} + \frac{n}{2} + 1$



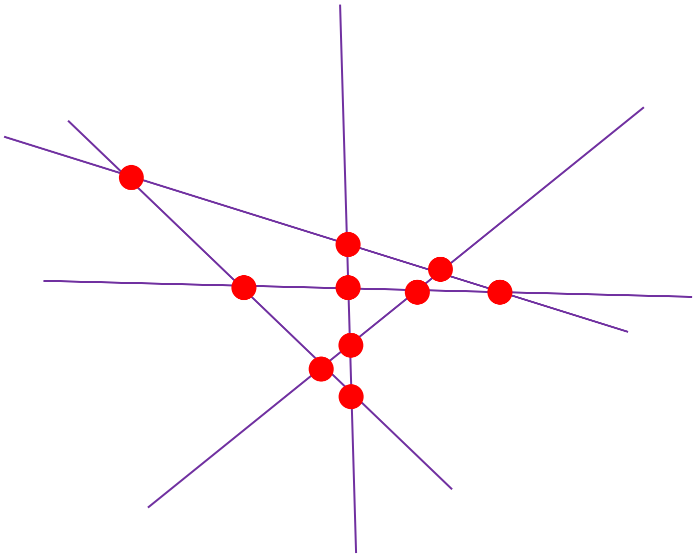
Combinatorial Complexity

#vertices + #edges + #faces

Theorem #vertices $\leq \frac{n(n-1)}{2}$

#edges $\leq n^2$

#faces $\leq \frac{n^2}{2} + \frac{n}{2} + 1$



Equality holds if and only if $A(L)$ is simple.

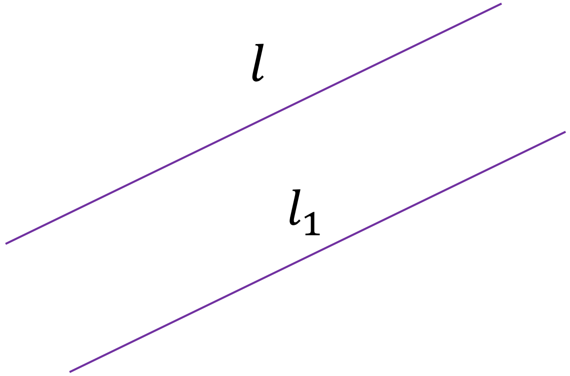
Proof of Complexity

We first show that #vertices, #edges, #faces are maximal when $A(L)$ is simple (no parallel or ≥ 3 concurrent lines) and not otherwise.

Proof of Complexity

We first show that #vertices, #edges, #faces are maximal when $A(L)$ is simple (no parallel or ≥ 3 concurrent lines) and not otherwise.

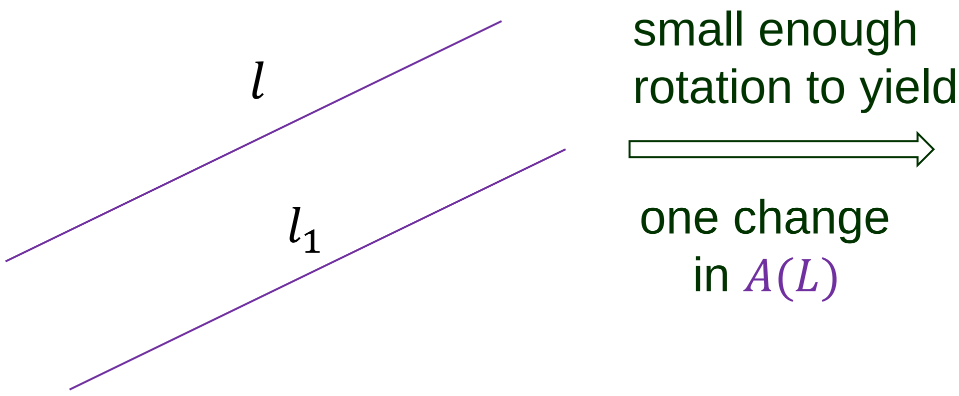
1) Let $l \in L$ be parallel to one or more lines.



Proof of Complexity

We first show that #vertices, #edges, #faces are maximal when $A(L)$ is simple (no parallel or ≥ 3 concurrent lines) and not otherwise.

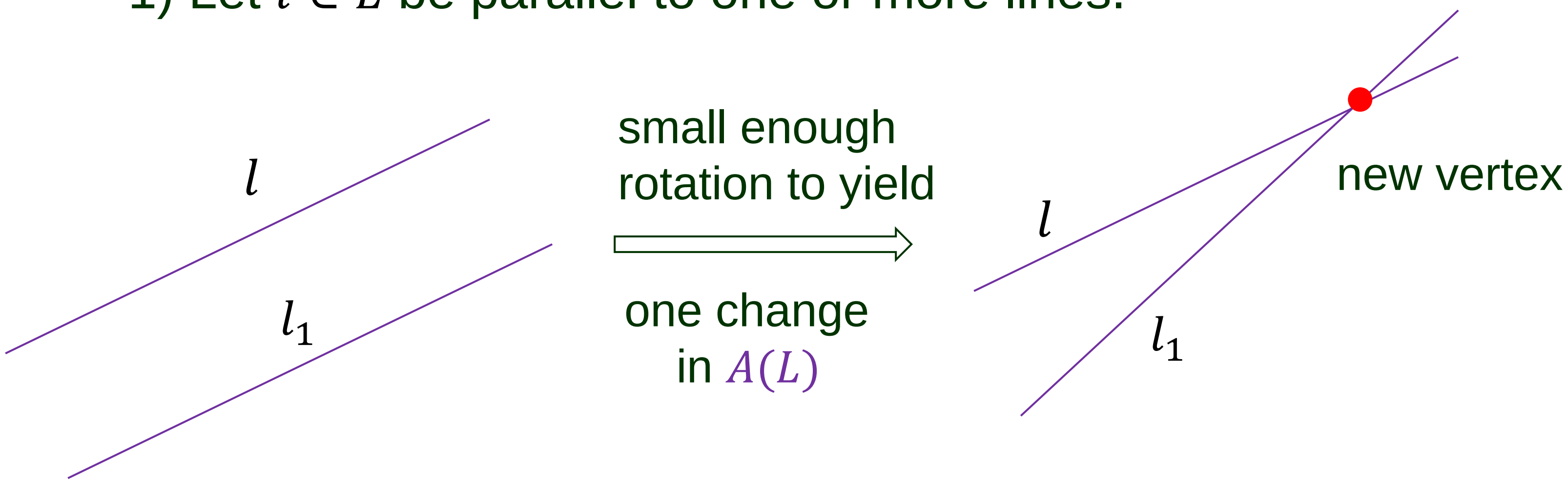
1) Let $l \in L$ be parallel to one or more lines.



Proof of Complexity

We first show that #vertices, #edges, #faces are maximal when $A(L)$ is simple (no parallel or ≥ 3 concurrent lines) and not otherwise.

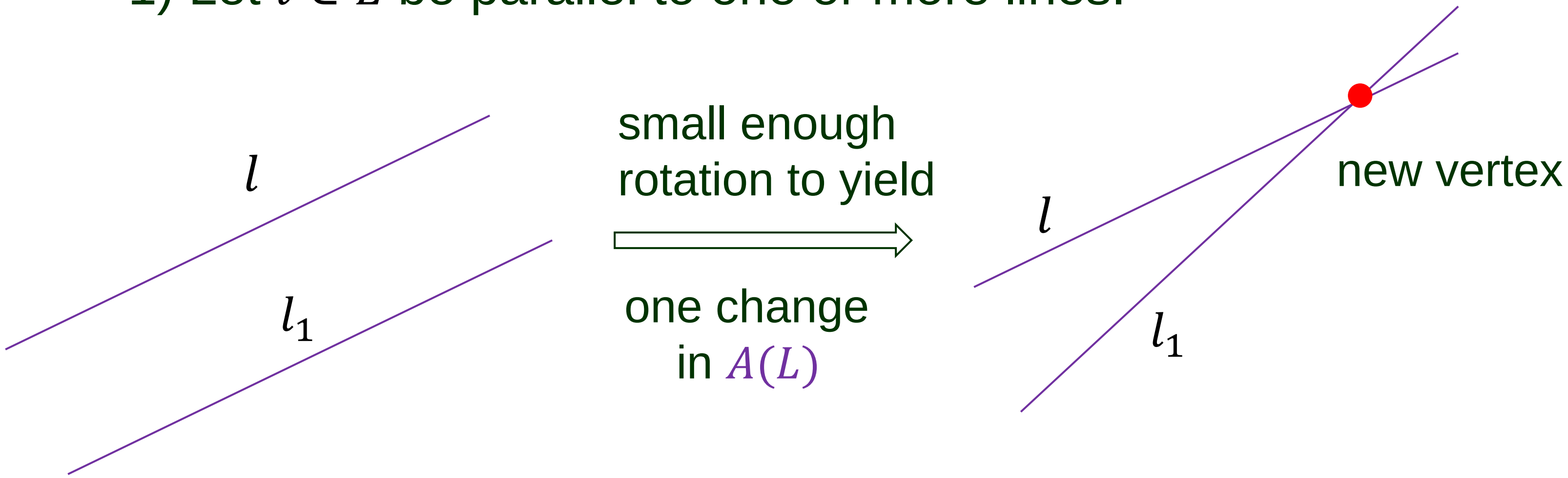
1) Let $l \in L$ be parallel to one or more lines.



Proof of Complexity

We first show that #vertices, #edges, #faces are maximal when $A(L)$ is simple (no parallel or ≥ 3 concurrent lines) and not otherwise.

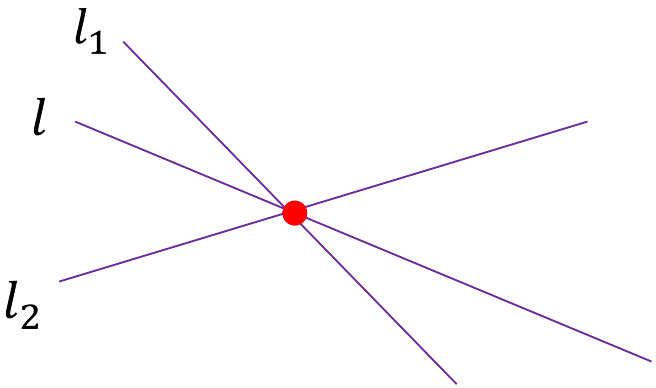
1) Let $l \in L$ be parallel to one or more lines.



Complexity increases in this case. (Hence such configuration cannot be maximal.)

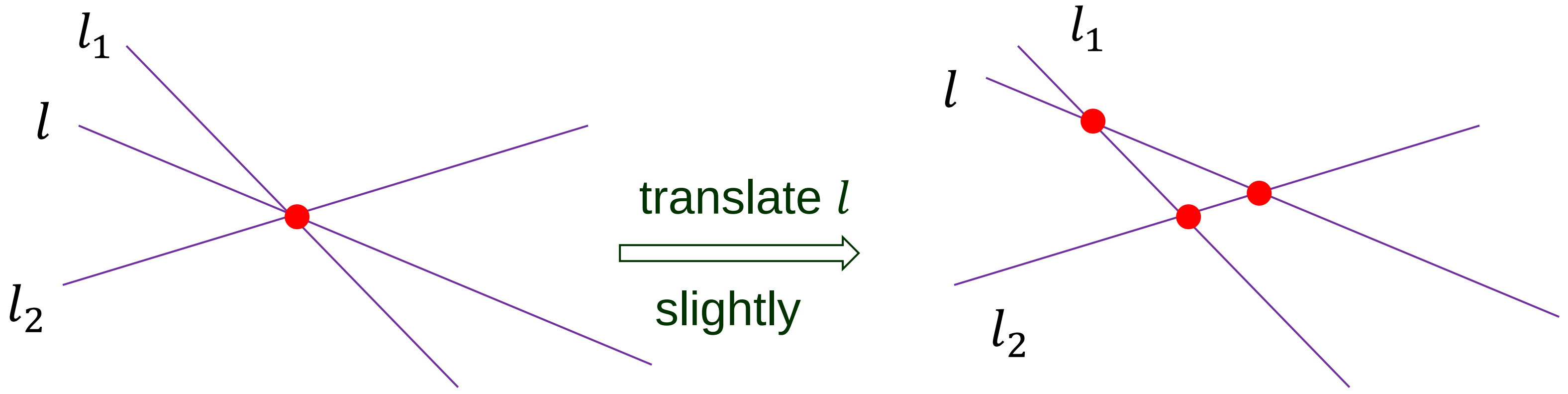
Proof (Cont'd)

2) Suppose l passes through a vertex.



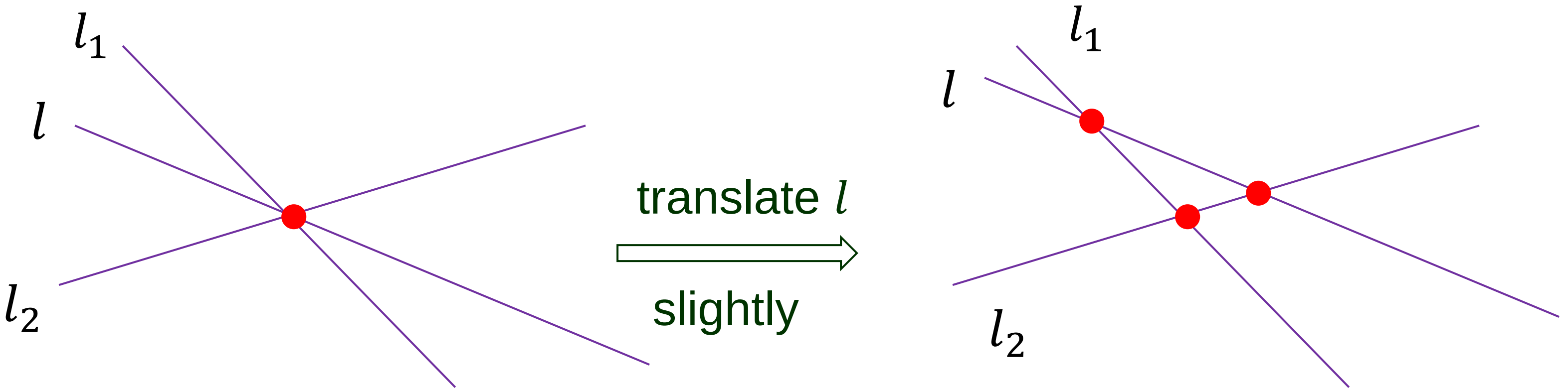
Proof (Cont'd)

2) Suppose l passes through a vertex.



Proof (Cont'd)

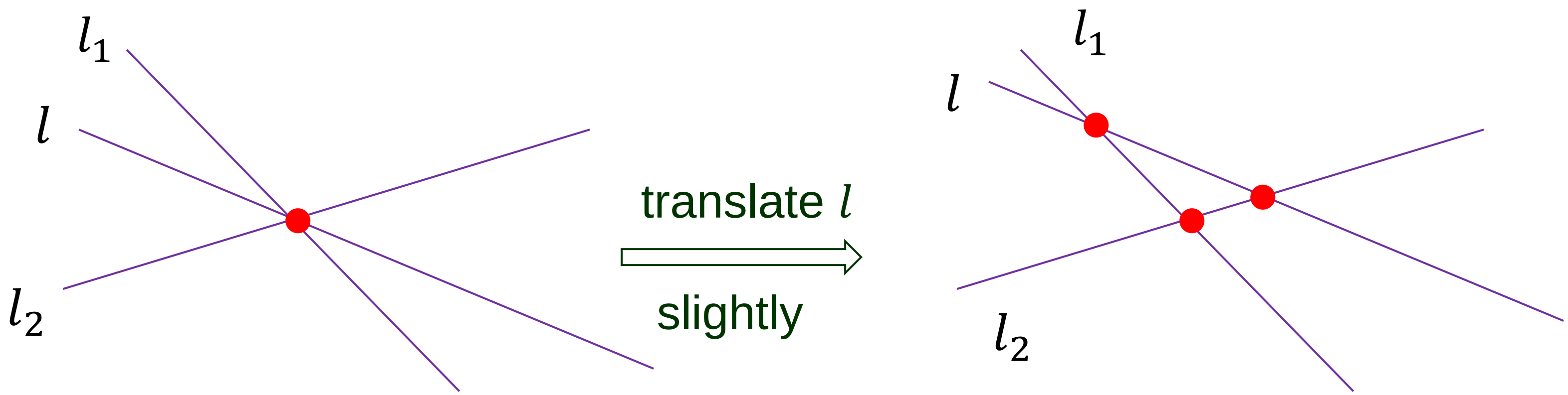
2) Suppose l passes through a vertex.



2 new vertices
3 new edges
1 new face

Proof (Cont'd)

2) Suppose l passes through a vertex.

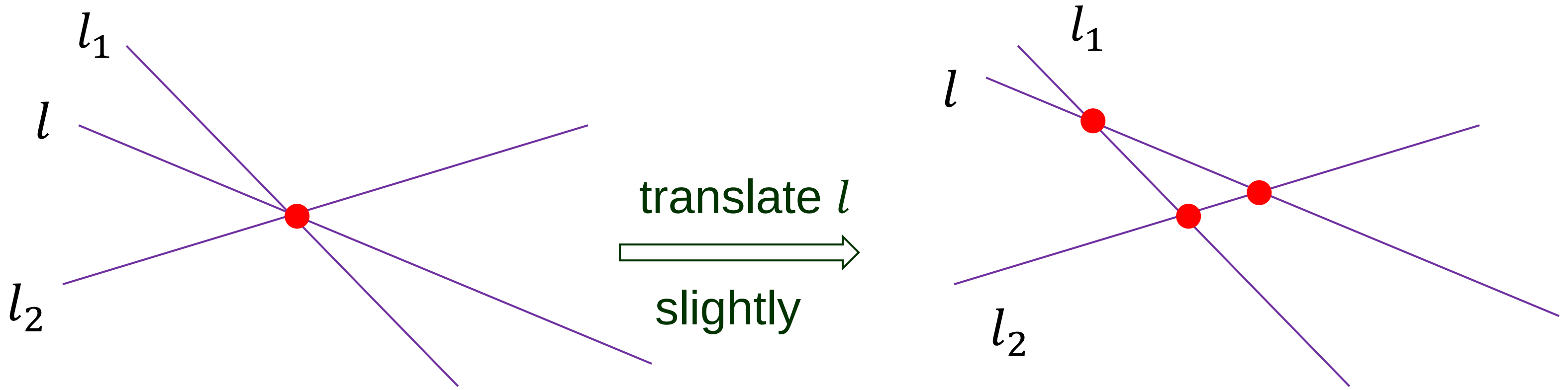


2 new vertices
3 new edges
1 new face

Since complexity increases, such configuration cannot be maximal either.

Proof (Cont'd)

2) Suppose l passes through a vertex.



2 new vertices
3 new edges
1 new face

Since complexity increases, such configuration cannot be maximal either.

The arrangement with maximal complexity must be simple.

Exact Size of a Simple Arrangement

$n_v =$ # vertices

$n_e =$ # edges

$n_f =$ # faces

Exact Size of a Simple Arrangement

$n_v =$ # vertices

$n_e =$ # edges

$n_f =$ # faces

Any pair of lines intersect.

Exact Size of a Simple Arrangement

$n_v =$ # vertices

$n_e =$ # edges

$n_f =$ # faces

Any pair of lines intersect. $\Longrightarrow$ $n_v = \binom{n}{2} = \frac{n(n-1)}{2}$

Exact Size of a Simple Arrangement

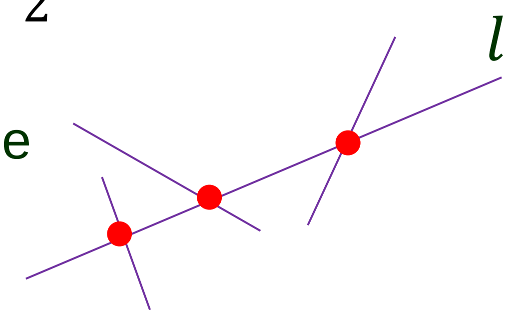
$n_v =$ # vertices

$n_e =$ # edges

$n_f =$ # faces

Any pair of lines intersect. $\Longrightarrow$ $n_v = \binom{n}{2} = \frac{n(n-1)}{2}$

#edges on one line = 1 + #intersections on the line



Exact Size of a Simple Arrangement

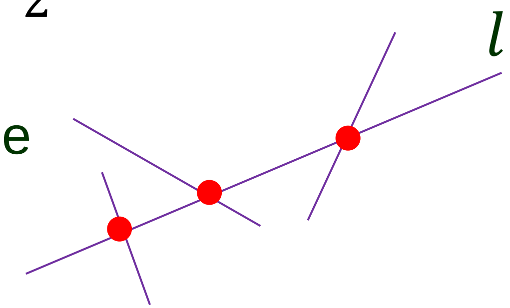
$n_v =$ # vertices

$n_e =$ # edges

$n_f =$ # faces

Any pair of lines intersect. $\Longrightarrow$ $n_v = \binom{n}{2} = \frac{n(n-1)}{2}$

#edges on one line = 1 + $\underbrace{\text{\#intersections on the line}}_{n-1}$



Exact Size of a Simple Arrangement

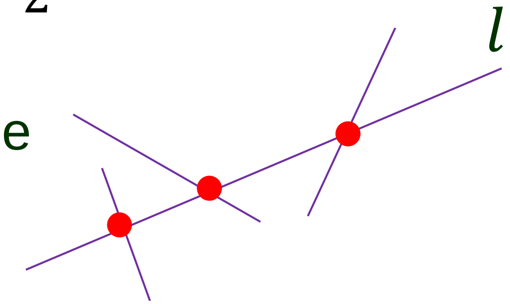
$n_v =$ # vertices

$n_e =$ # edges

$n_f =$ # faces

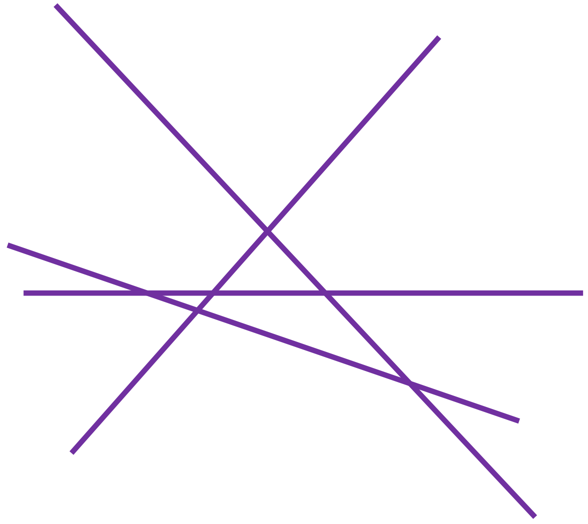
Any pair of lines intersect. $\Longrightarrow$ $n_v = \binom{n}{2} = \dfrac{n(n-1)}{2}$

#edges on one line = 1 + $\underbrace{\text{\#intersections on the line}}_{n-1}$

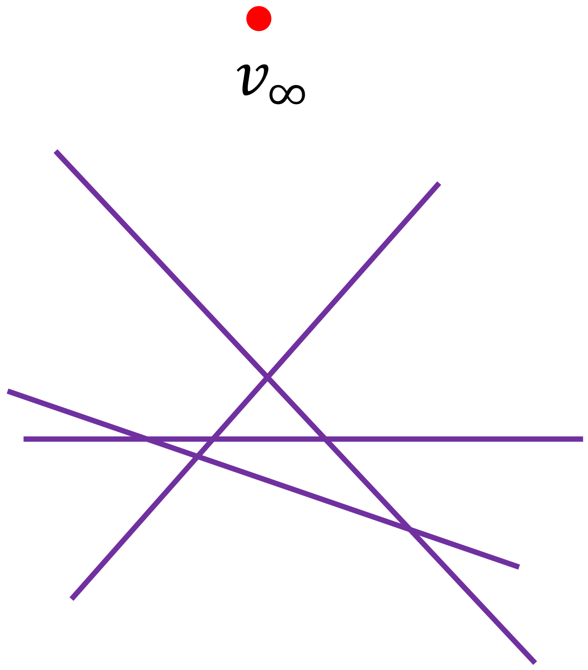


$$\Longrightarrow \quad n_e = n \cdot (1 + n - 1) = n^2$$

Number of Faces

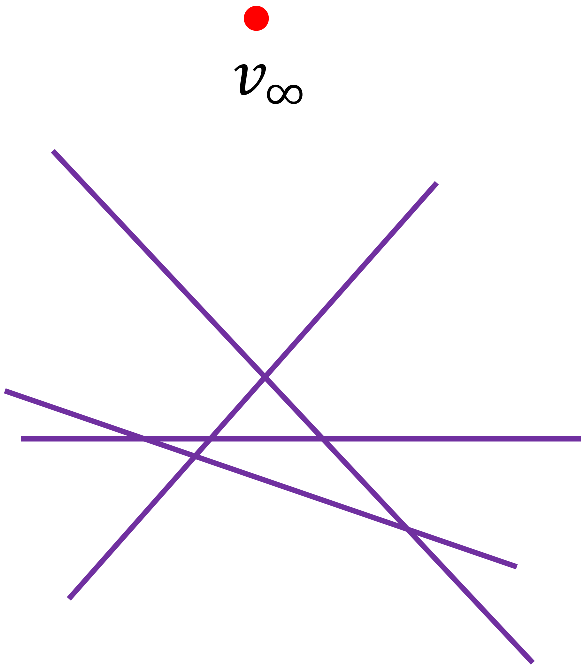


Number of Faces



1. Add a vertex v_∞ at infinity.

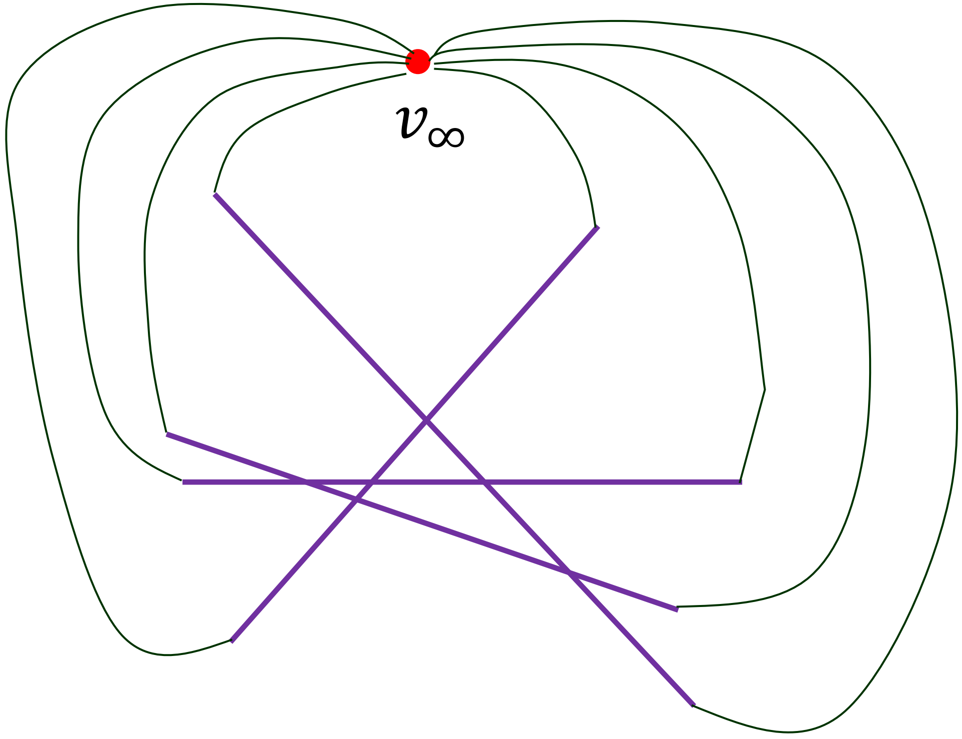
Number of Faces



1. Add a vertex v_∞ at infinity.

2. Extend (and bend) every half-infinite edge to v_∞.

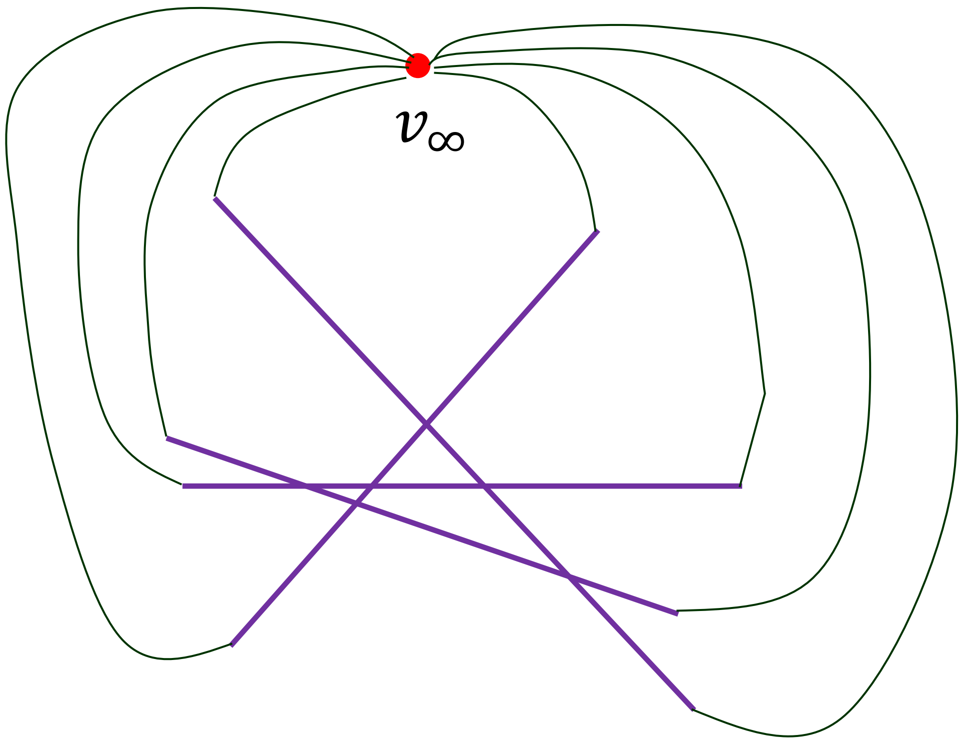
Number of Faces



1. Add a vertex v_∞ at infinity.

2. Extend (and bend) every half-infinite edge to v_∞.

Number of Faces

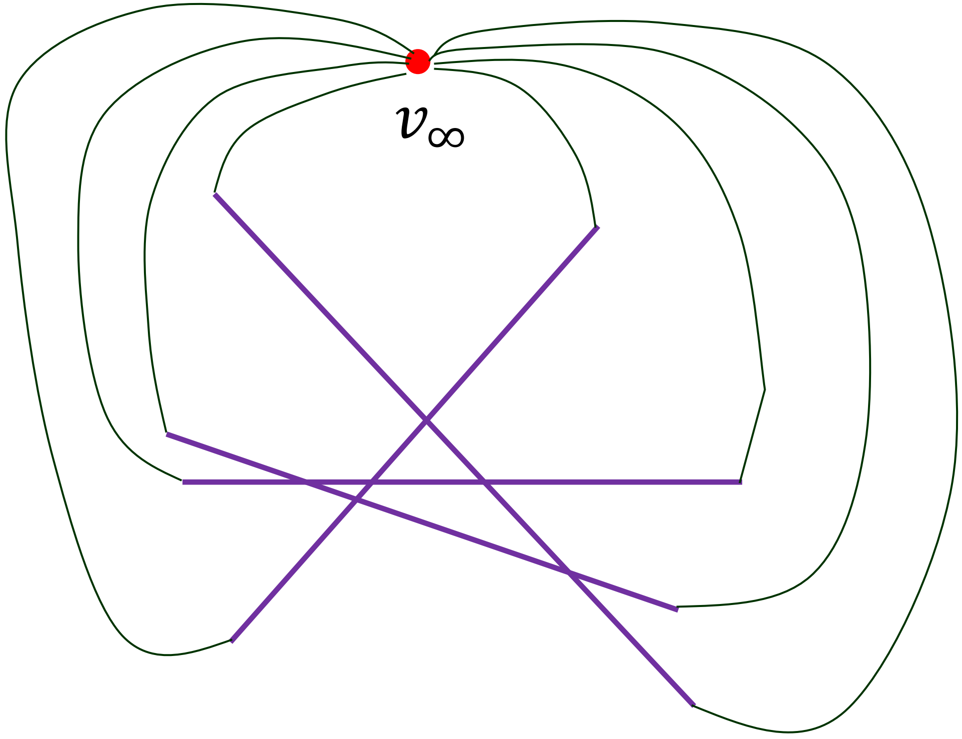


1. Add a vertex v_∞ at infinity.

2. Extend (and bend) every half-infinite edge to v_∞.

No edge crossing

Number of Faces



1. Add a vertex v_∞ at infinity.

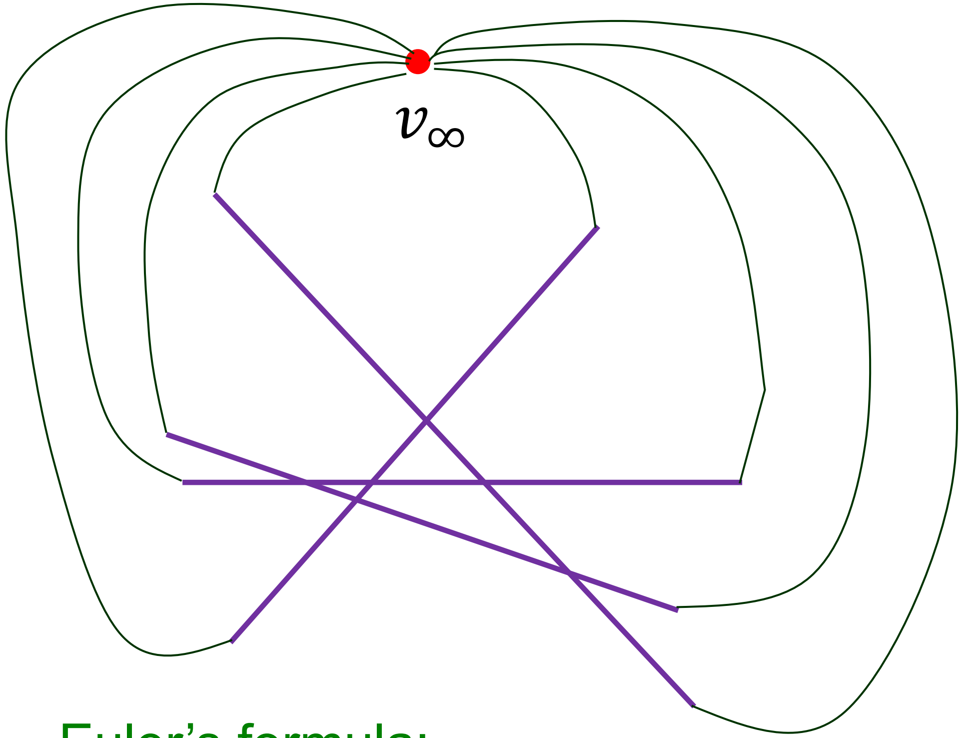
2. Extend (and bend) every half-infinite edge to v_∞.

No edge crossing

⇩

Planar graph

Number of Faces



1. Add a vertex v_∞ at infinity.

2. Extend (and bend) every half-infinite edge to v_∞.

No edge crossing

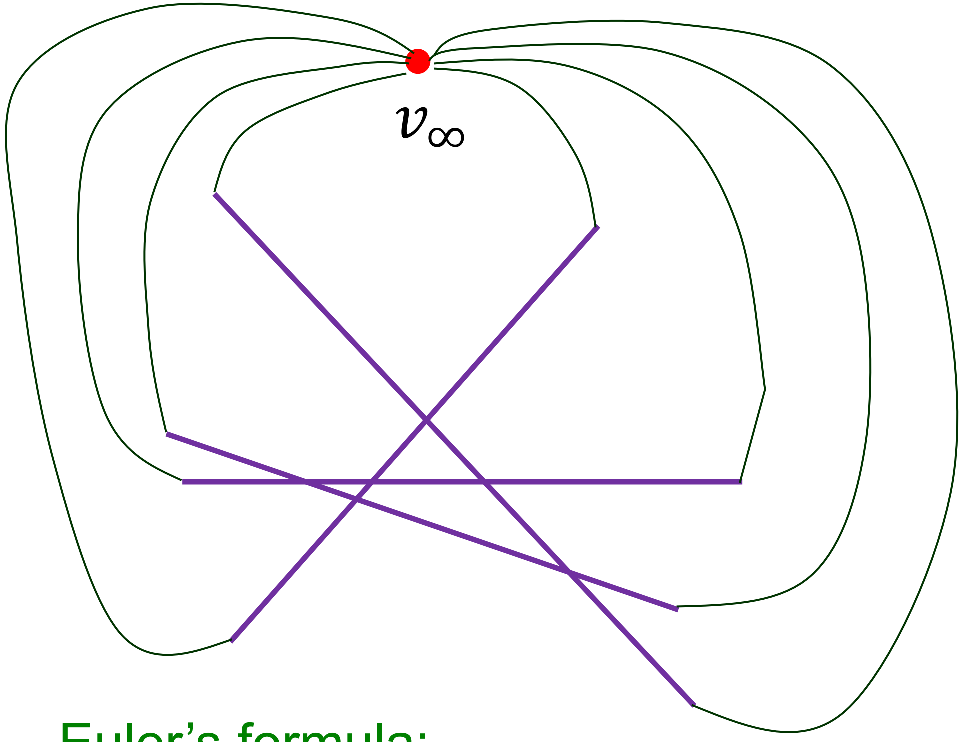
⇓

Planar graph

Euler's formula:

$$(n_v + 1) + n_f - n_e = 2$$

Number of Faces



1. Add a vertex v_∞ at infinity.

2. Extend (and bend) every half-infinite edge to v_∞.

No edge crossing

⇓

Planar graph

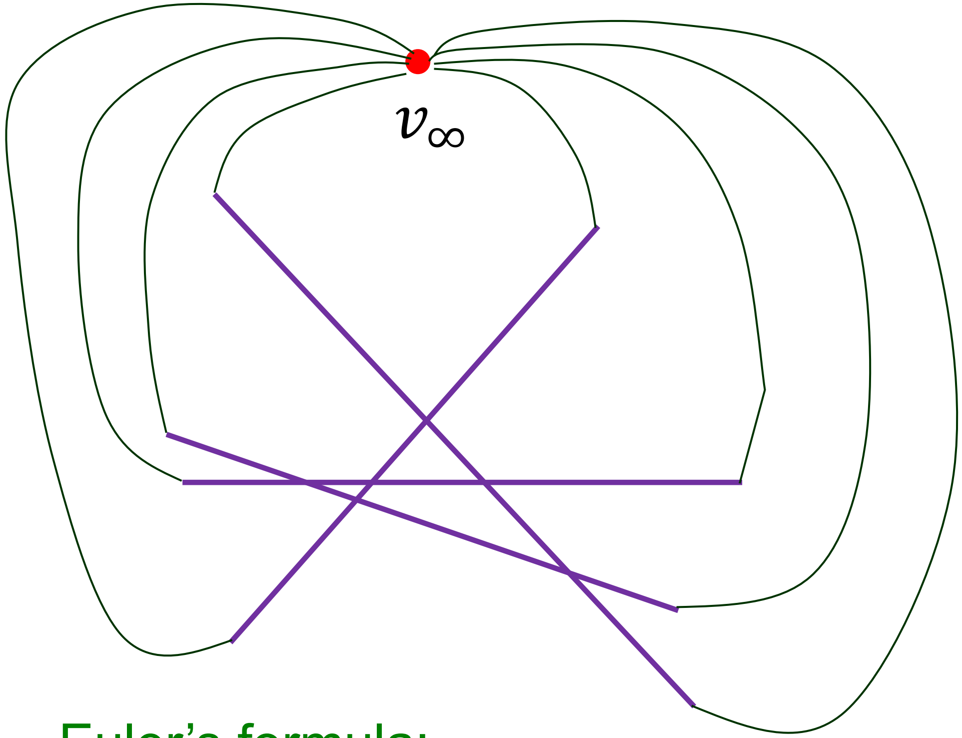
Euler's formula:

$$(n_v + 1) + n_f - n_e = 2$$

⇒

$$n_f = 2 - (n_v + 1) + n_e$$

Number of Faces



1. Add a vertex v_∞ at infinity.

2. Extend (and bend) every half-infinite edge to v_∞.

No edge crossing

⇓

Planar graph

Euler's formula:

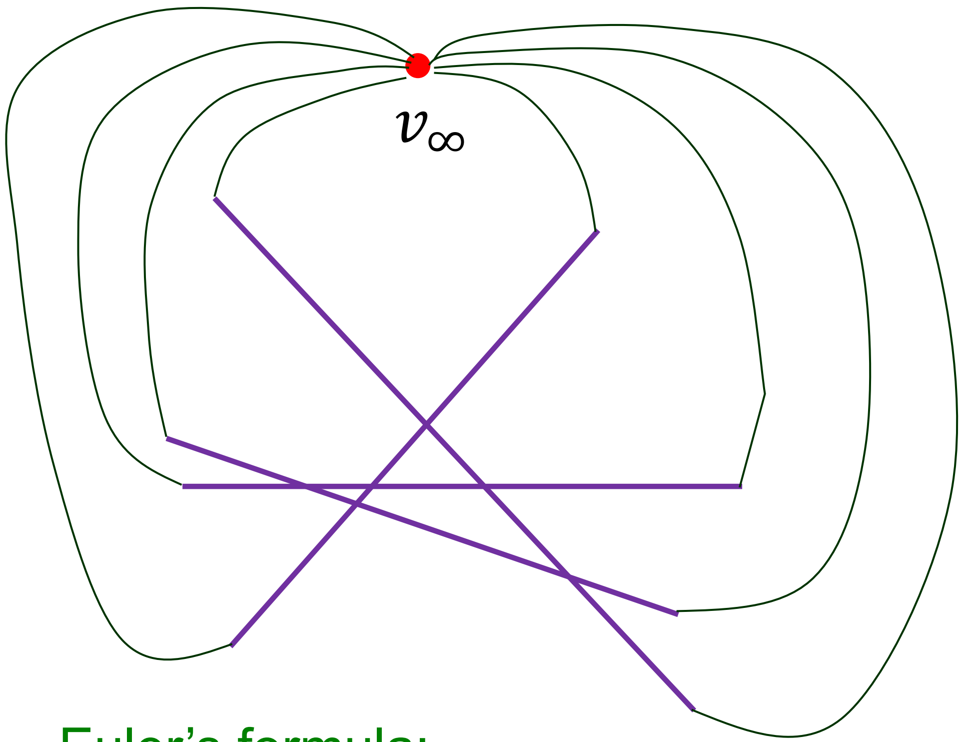
$$(n_v + 1) + n_f - n_e = 2$$

⇒

$$n_f = 2 - (n_v + 1) + n_e$$

$$= 2 - \left(\frac{n(n-1)}{2} + 1\right) + n^2$$

Number of Faces



1. Add a vertex v_∞ at infinity.

2. Extend (and bend) every half-infinite edge to v_∞.

No edge crossing

⇓

Planar graph

Euler's formula:

$$(n_v + 1) + n_f - n_e = 2$$

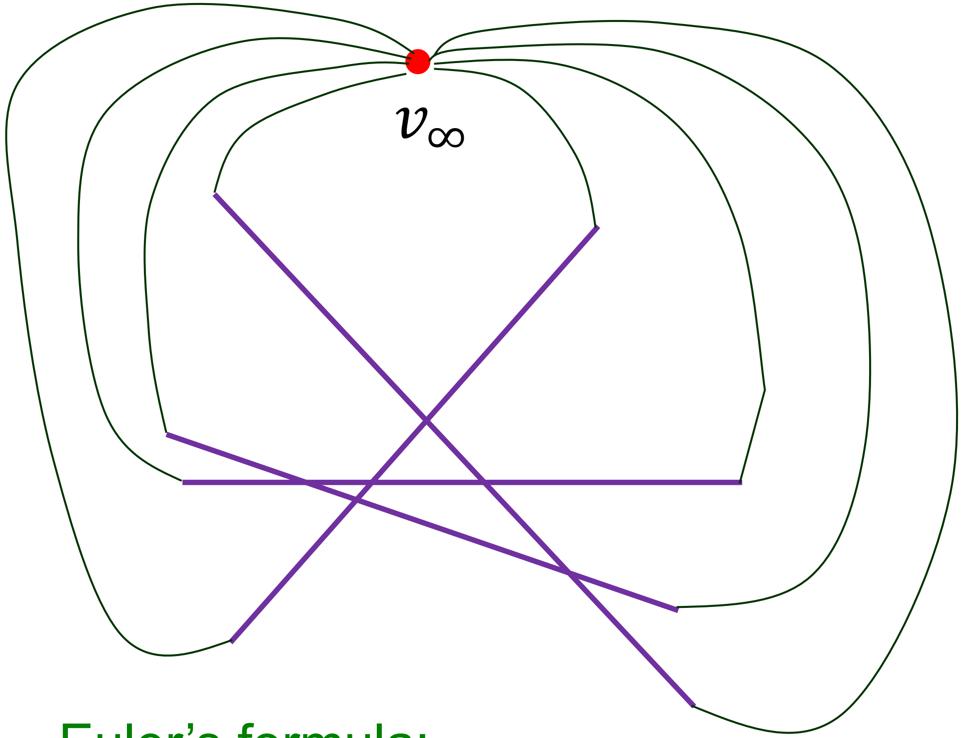
⇒

$$n_f = 2 - (n_v + 1) + n_e$$

$$= 2 - \left(\frac{n(n-1)}{2} + 1\right) + n^2$$

$$\boxed{= \frac{n^2}{2} + \frac{n}{2} + 1}$$

Number of Faces



1. Add a vertex v_∞ at infinity.

2. Extend (and bend) every half-infinite edge to v_∞.

No edge crossing

$\Downarrow$

Planar graph

Euler's formula:

$$(n_v + 1) + n_f - n_e = 2$$

$$\Rightarrow$$

$$n_f = 2 - (n_v + 1) + n_e$$

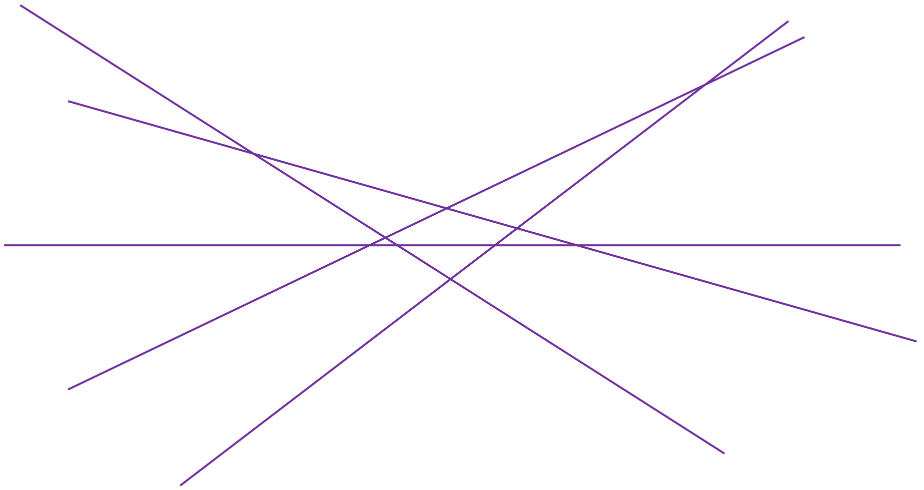
$$= 2 - \left(\frac{n(n-1)}{2} + 1\right) + n^2$$

$$\boxed{= \frac{n^2}{2} + \frac{n}{2} + 1}$$

□

II. Storage of Line Arrangement

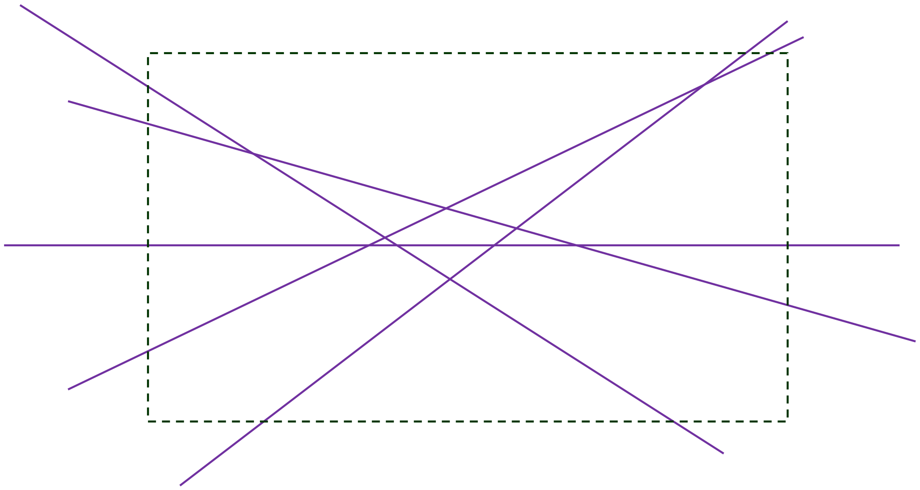
Doubly-connected edge list.



II. Storage of Line Arrangement

Doubly-connected edge list.

Add a bounding box to contain all vertices in interior.



Construction of DCEL for $A(L)$

Plane sweep?

$$O(n \log n + I \log n)$$

Construction of DCEL for $A(L)$

Plane sweep?

$$O(n \log n + I \log n)$$

pairwise intersection

Construction of DCEL for $A(L)$

Plane sweep?

$$O(n \log n + I \log n)$$

pairwise intersection $\Longrightarrow$ $I = n(n-1)/2$

Construction of DCEL for $A(L)$

Plane sweep?

$$O(n \log n + I \log n) = O(n^2 \log n)$$

pairwise intersection $\Rightarrow$ $I = n(n-1)/2$

Construction of DCEL for $A(L)$

Plane sweep?

$$O(n \log n + I \log n) = O(n^2 \log n)$$

pairwise intersection $\Rightarrow$ $I = n(n-1)/2$

Not optimal!

Incremental Algorithm

Preprocessing

- Compute the bounding box $B(L)$.

Incremental Algorithm

Preprocessing

- Compute the bounding box $B(L)$.
  - $n(n-1)/2$ intersections.

Incremental Algorithm

Preprocessing

- Compute the bounding box $B(L)$.
  - $n(n-1)/2$ intersections.
  - leftmost, rightmost, top, bottom intersections.

Incremental Algorithm

Preprocessing

- Compute the bounding box $B(L)$.
  - $n(n-1)/2$ intersections.
  - leftmost, rightmost, top, bottom intersections.
  - each line yielding two intersections with $B(L)$.

Incremental Algorithm

Preprocessing

- Compute the bounding box $B(L)$.
  - $n(n-1)/2$ intersections.
  - leftmost, rightmost, top, bottom intersections.
  - each line yielding two intersections with $B(L)$.
  - $\frac{n(n-1)}{2} + 2n + 4 = \frac{n^2+3n+8}{2}$ vertices in $B(L)$.

Incremental Algorithm

Preprocessing

- Compute the bounding box $B(L)$. $\Theta(n^2)$
  - $n(n-1)/2$ intersections.
  - leftmost, rightmost, top, bottom intersections.
  - each line yielding two intersections with $B(L)$.
  - $\frac{n(n-1)}{2} + 2n + 4 = \frac{n^2+3n+8}{2}$ vertices in $B(L)$.

Incremental Algorithm

Preprocessing

- Compute the bounding box $B(L)$. $\Theta(n^2)$
  - $n(n-1)/2$ intersections.
  - leftmost, rightmost, top, bottom intersections.
  - each line yielding two intersections with $B(L)$.
  - $\frac{n(n-1)}{2} + 2n + 4 \ = \frac{n^2+3n+8}{2}$ vertices in $B(L)$.

- Add lines $l_1, l_2, \ldots, l_n$ one by one.

Incremental Algorithm

Preprocessing

- Compute the bounding box $B(L)$. $\Theta(n^2)$
  - $n(n-1)/2$ intersections.
  - leftmost, rightmost, top, bottom intersections.
  - each line yielding two intersections with $B(L)$.
  - $\frac{n(n-1)}{2} + 2n + 4 = \frac{n^2+3n+8}{2}$ vertices in $B(L)$.

- Add lines $l_1, l_2, \ldots, l_n$ one by one.

  Update the DCEL after each addition.

Incremental Algorithm

Preprocessing

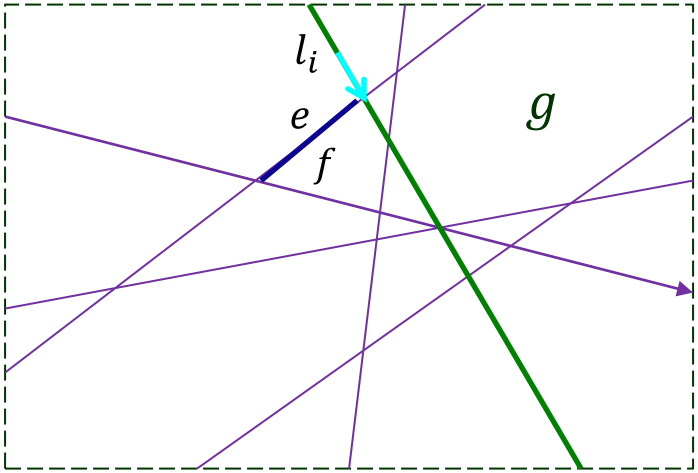
- Compute the bounding box $B(L)$. $\Theta(n^2)$
  - $n(n-1)/2$ intersections.
  - leftmost, rightmost, top, bottom intersections.
  - each line yielding two intersections with $B(L)$.
  - $\frac{n(n-1)}{2} + 2n + 4 \ = \frac{n^2+3n+8}{2}$ vertices in $B(L)$.

- Add lines $l_1, l_2, \ldots, l_n$ one by one.

  Update the DCEL after each addition.

Updating the Subdivision

A_i: subdivision due to $\{l_1, l_2, \ldots, l_i\}$

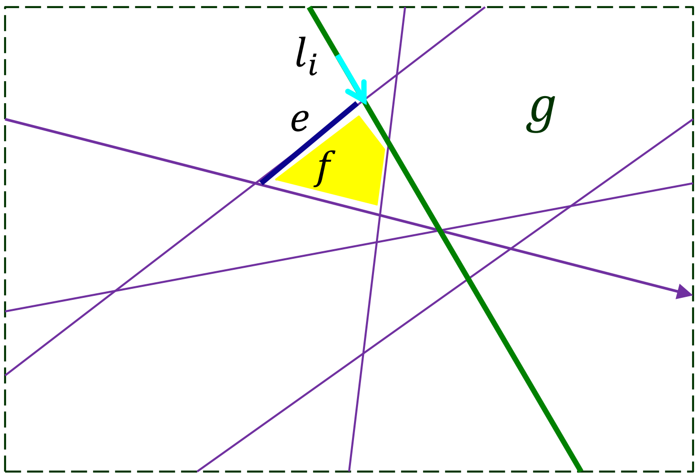


A_{i-1}

Case 1 Enter a face f through edge e.

Updating the Subdivision

A_i: subdivision due to $\{l_1, l_2, \ldots, l_i\}$

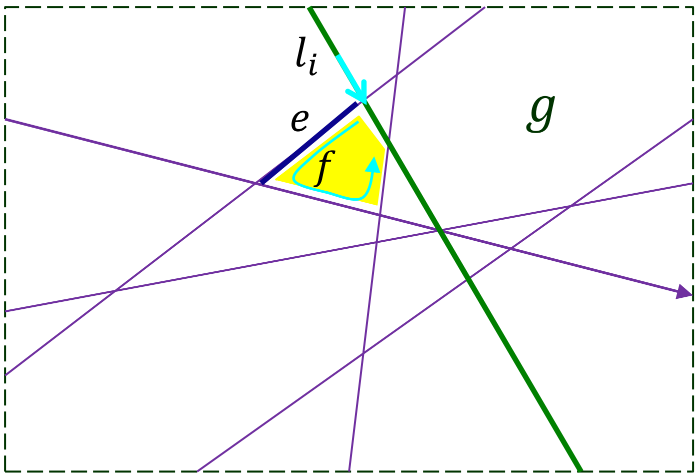


A_{i-1}

Case 1 Enter a face f through edge e.

Updating the Subdivision

A_i: subdivision due to $\{l_1, l_2, ..., l_i\}$



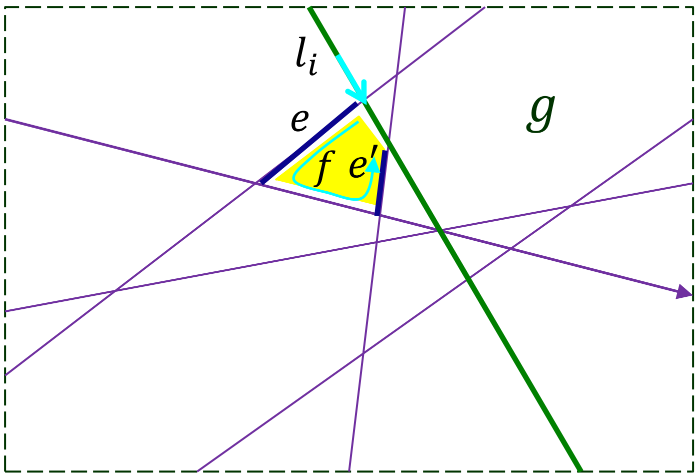
A_{i-1}

Case 1 Enter a face f through edge e.

- Walk along boundary of f using the the Next pointer.

Updating the Subdivision

A_i: subdivision due to $\{l_1, l_2, \ldots, l_i\}$



A_{i-1}

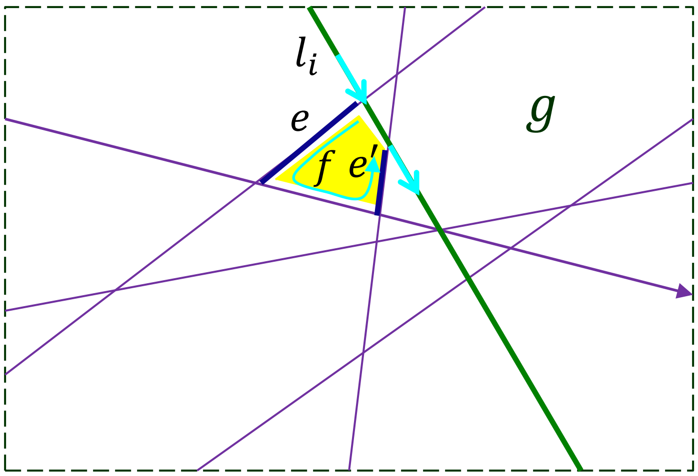
Case 1 Enter a face f through edge e.

- Walk along boundary of f using the the Next pointer.

- Find exit edge e'.

Updating the Subdivision

A_i: subdivision due to $\{l_1, l_2, \ldots, l_i\}$



A_{i-1}

Case 1 Enter a face f through edge e.

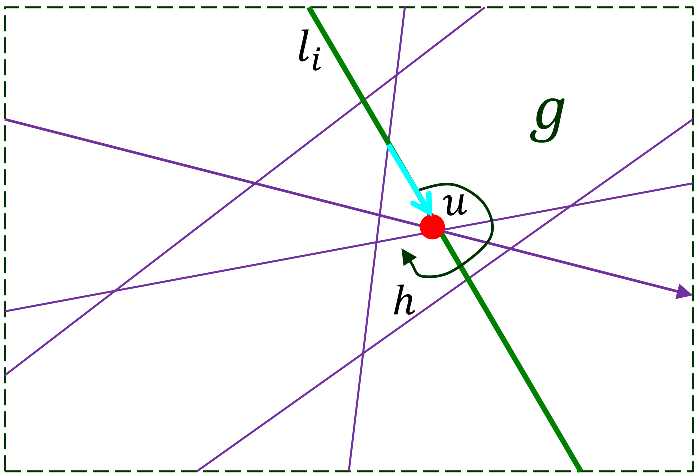
- Walk along boundary of f using the the Next pointer.

- Find exit edge e'.

- Use its Twin() pointer to enter face g.

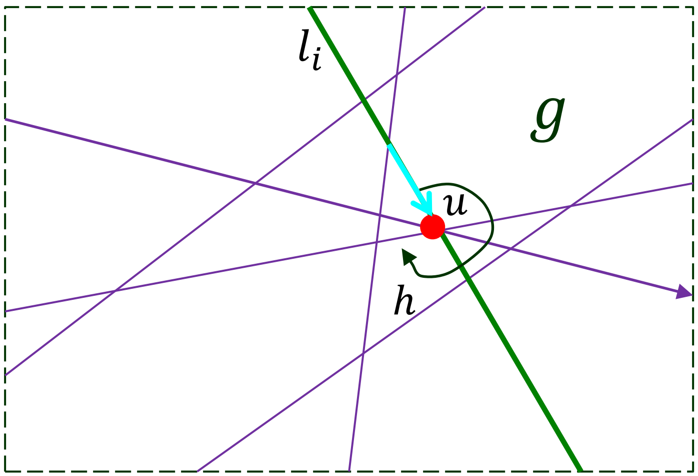
Updating the Subdivision

Case 2 Leave a face (g) through a vertex (u).



Updating the Subdivision

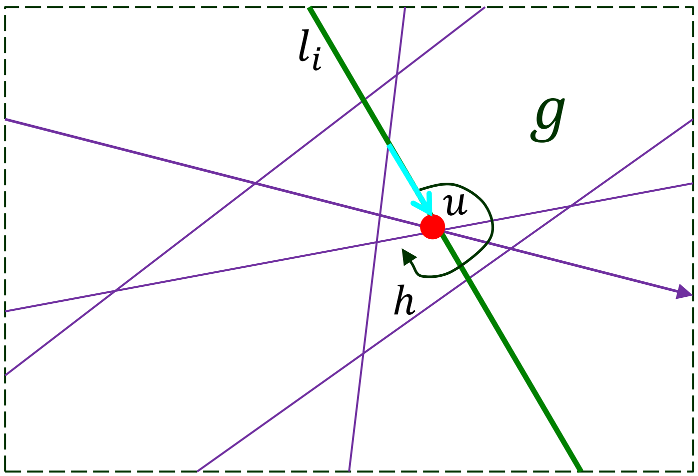
Case 2 Leave a face (g) through a vertex (u).



- walk around u to find the next face (h) intersected by l_i.

Updating the Subdivision

Case 2 Leave a face (g) through a vertex (u).

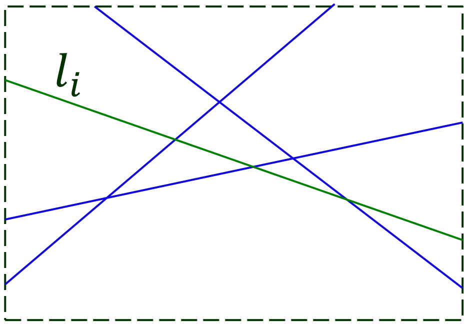


- walk around u to find the next face (h) intersected by l_i.

Alternatively use Next() and Twin() pointers.

First Edge of Intersection

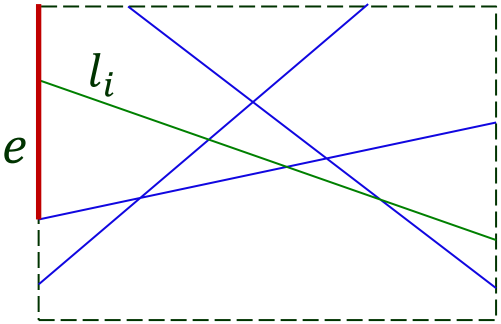
How to find the first edge (leftmost edge) intersected by l_i?



First Edge of Intersection

How to find the first edge (leftmost edge) intersected by l_i?

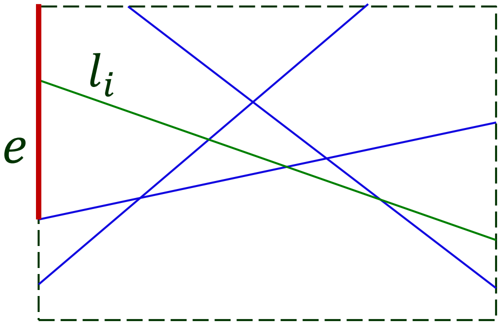
- It must be an edge on the bounding box $B(L)$.



First Edge of Intersection

How to find the first edge (leftmost edge) intersected by l_i?

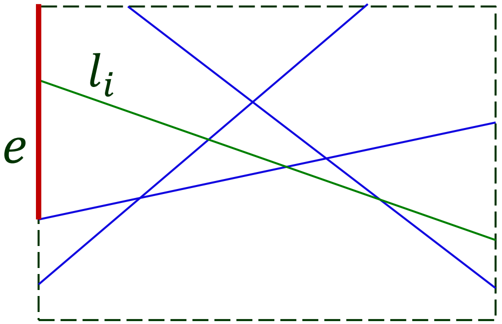
- It must be an edge on the bounding box $B(L)$.
- Just test all the edges on $B(L)$.



First Edge of Intersection

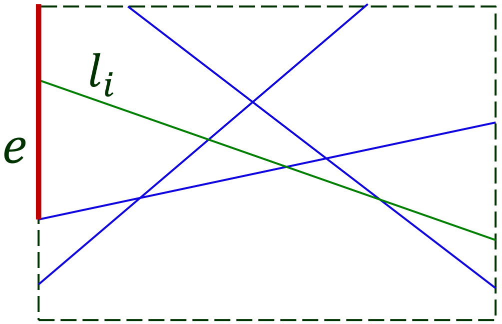
How to find the first edge (leftmost edge) intersected by l_i?

- It must be an edge on the bounding box $B(L)$.
- Just test all the edges on $B(L)$.
- In case l_i is vertical, locate the bottom intersection to start off traversal.



First Edge of Intersection

How to find the first edge (leftmost edge) intersected by l_i?

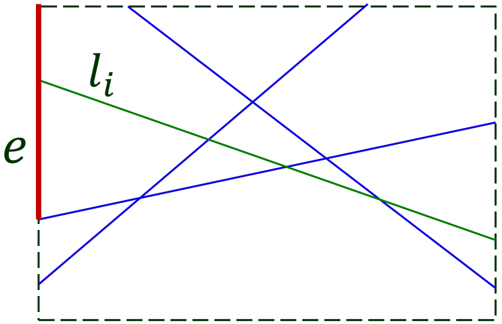


- It must be an edge on the bounding box $B(L)$.
- Just test all the edges on $B(L)$.
- In case l_i is vertical, locate the bottom intersection to start off traversal.

A_{i-1} has $2(i-1)+4$ edges on $B(L)$ since each edge intersects it twice.

First Edge of Intersection

How to find the first edge (leftmost edge) intersected by l_i?

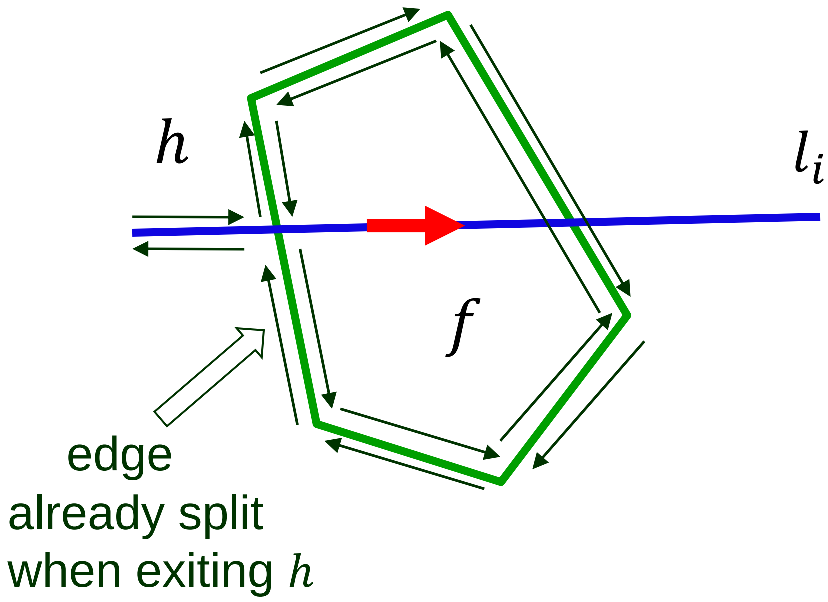


- It must be an edge on the bounding box $B(L)$.
- Just test all the edges on $B(L)$.
- In case l_i is vertical, locate the bottom intersection to start off traversal.

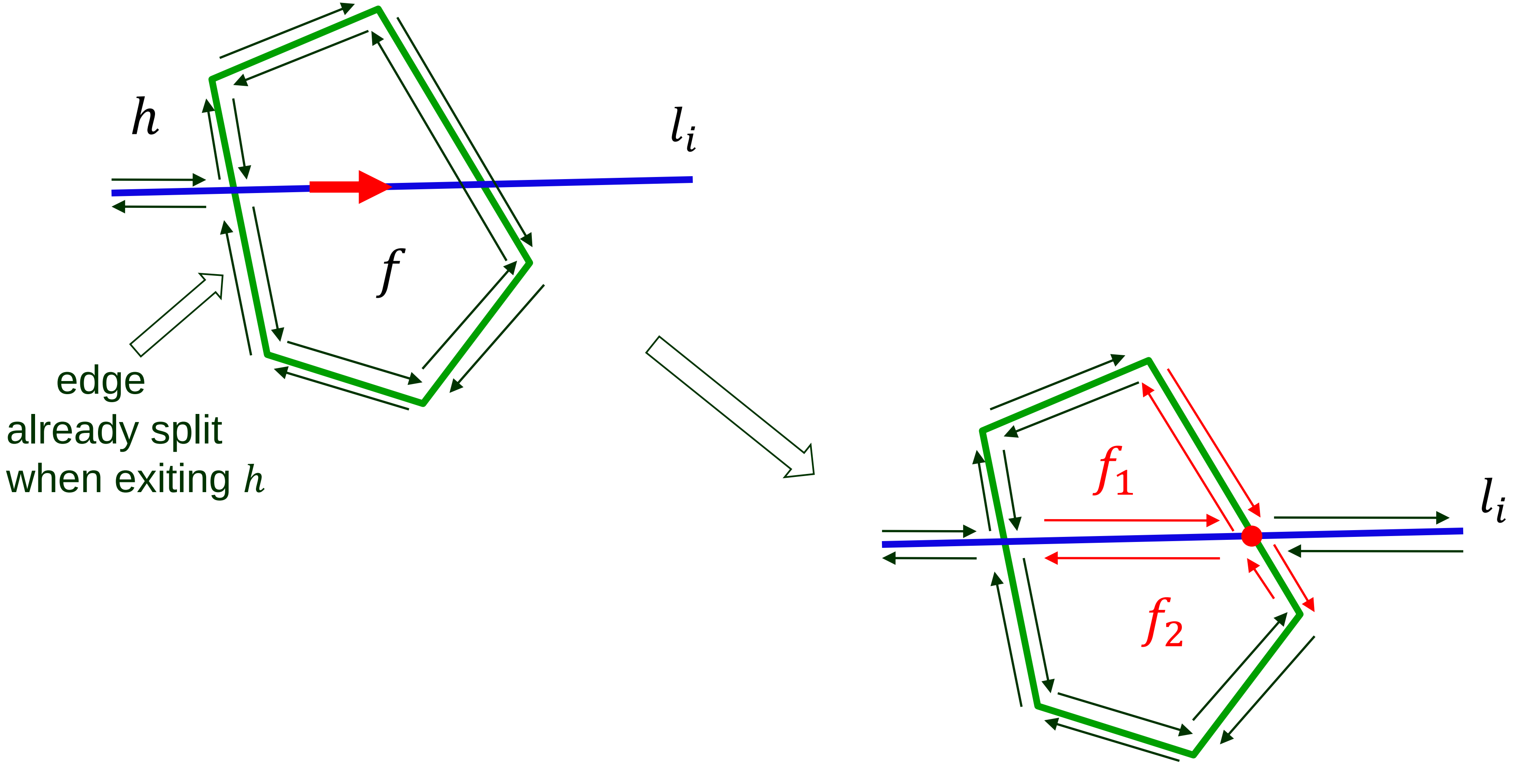
A_{i-1} has $2(i-1)+4$ edges on $B(L)$ since each edge intersects it twice.

First intersection edge can be found in $O(i)$ time.

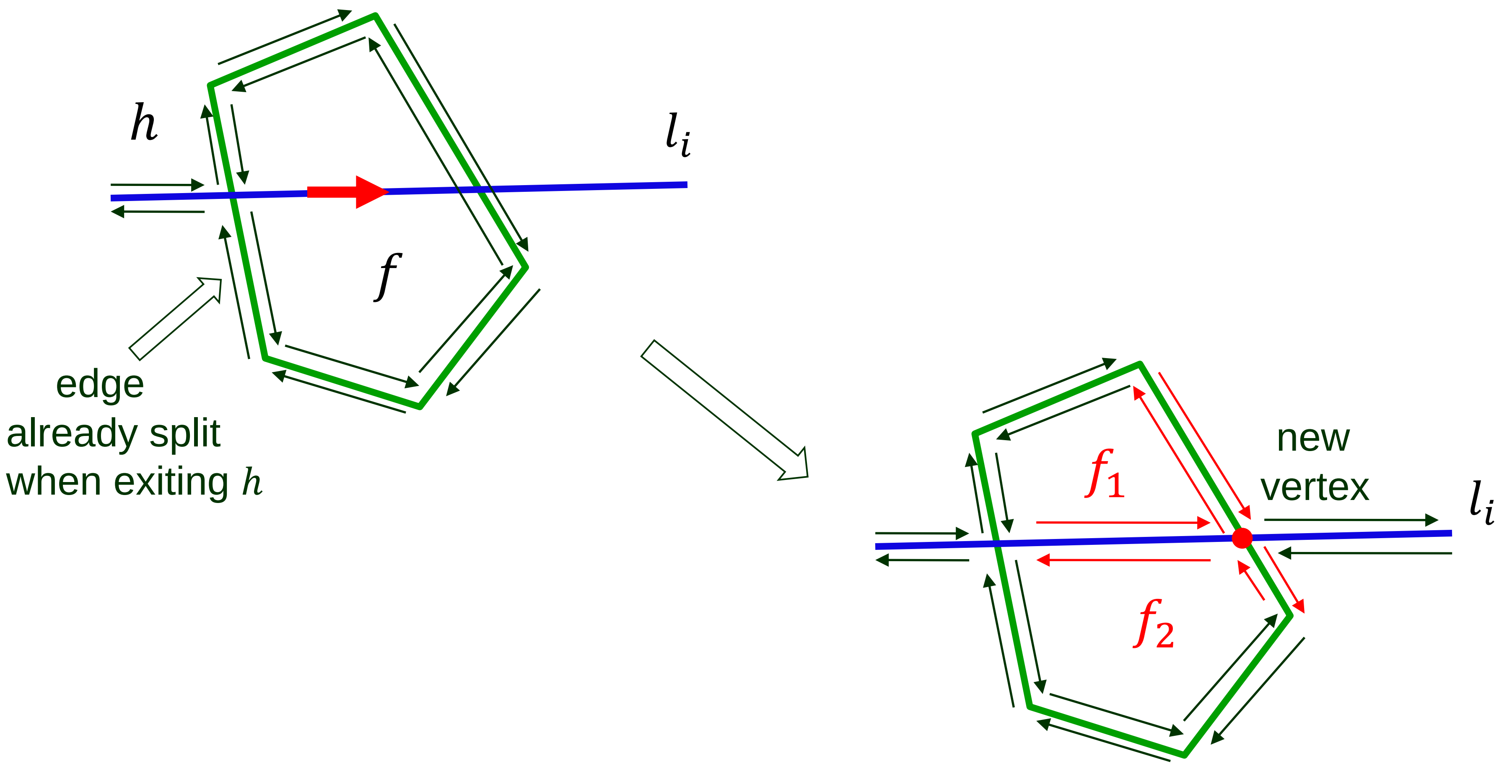
Splitting a Face



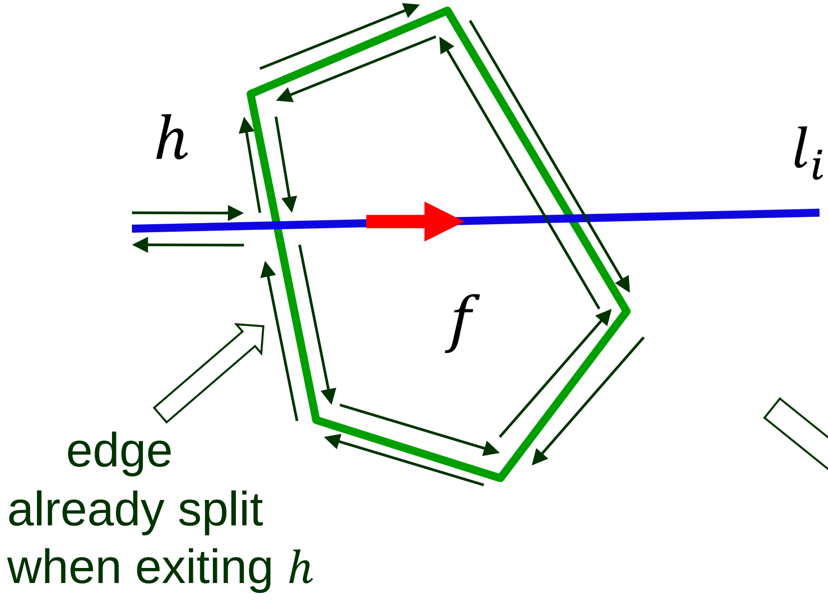
Splitting a Face



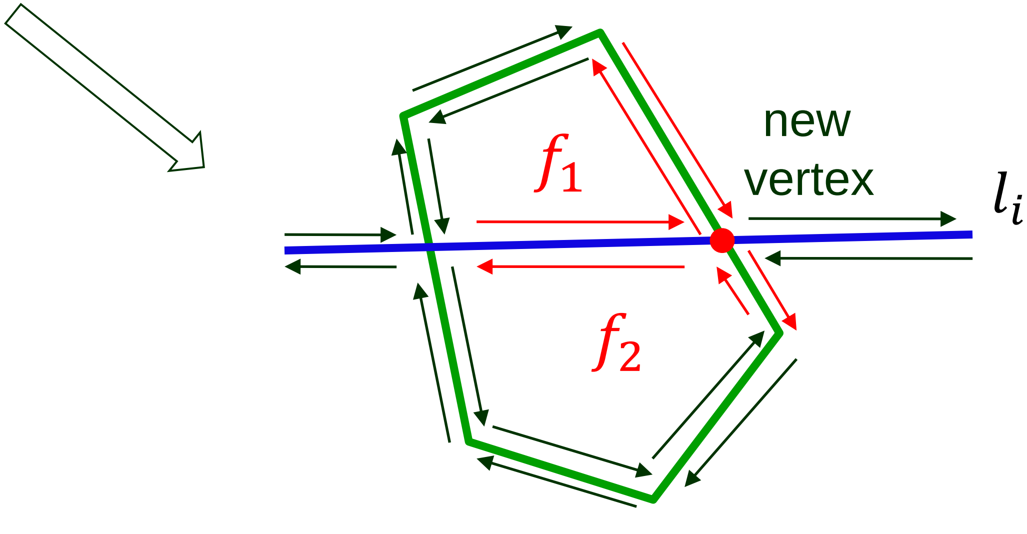
Splitting a Face



Splitting a Face



2 new faces
1 new vertex
6 new half-edges



The Algorithm

```
ConstructArrangement(L)

1.  Compute a bounding box B(L) to contain all vertices of A(L)// O(n^2)
2.  Initialize DCEL for B(L)  // O(1)
3.  for i ← 1 to n
4.      do e ← edge on B(L) that first intersects with l_i
5.         f ← bounded face incident on e      // O(i)
6.         while f is bounded
7.             split f
8.             f ← next intersected face
```

Face Splitting

Split faces in A_{i-1} intersected by l_i.

Face Splitting

Split faces in A_{i-1} intersected by l_i.

- ☆ $A(L)$ simple

Face Splitting

Split faces in A_{i-1} intersected by l_i.

- ☆ $A(L)$ simple
  - Splitting a face f and finding the next intersected face

    $O(\text{complexity of } f)$

Face Splitting

Split faces in A_{i-1} intersected by l_i.

☆ $A(L)$ simple

- Splitting a face f and finding the next intersected face

  O(complexity of f)

  ⇩

- Insertion of l_i takes time linear in total complexity of faces intersected by the line.

Face Splitting

Split faces in A_{i-1} intersected by l_i.

☆ $A(L)$ simple

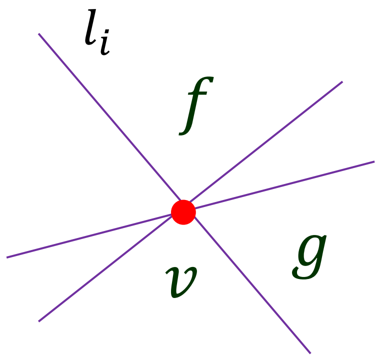
- Splitting a face f and finding the next intersected face

  O(complexity of f)

  ⇩

- Insertion of l_i takes time linear in total complexity of faces intersected by the line.

☆ $A(L)$ not simple



Face Splitting

Split faces in A_{i-1} intersected by l_i.

☆ $A(L)$ simple

- Splitting a face f and finding the next intersected face

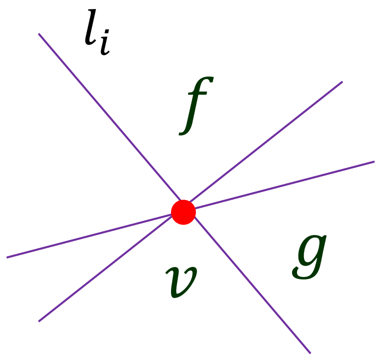
  O(complexity of f)

  ⇩

- Insertion of l_i takes time linear in total complexity of faces intersected by the line.

☆ $A(L)$ not simple

- l_i may leave f through a vertex v where ≥ 3 lines including l_i intersect.



Face Splitting

Split faces in A_{i-1} intersected by l_i.

☆ $A(L)$ simple

- Splitting a face f and finding the next intersected face

  O(complexity of f)

  ⇩

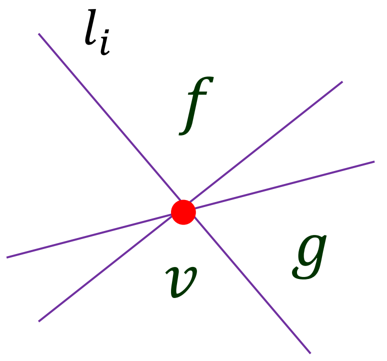
- Insertion of l_i takes time linear in total complexity of faces intersected by the line.

☆ $A(L)$ not simple

- l_i may leave f through a vertex v where ≥ 3 lines including l_i intersect.

  ⇩

- Walk around v to find the next face (g) to split, scanning over edges that bound faces intersected by l_i.



Face Splitting

Split faces in A_{i-1} intersected by l_i.

Total complexity?

☆ $A(L)$ simple

- Splitting a face f and finding the next intersected face

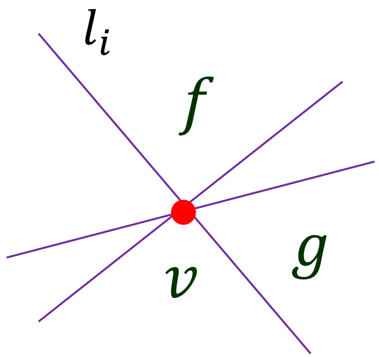
  O(complexity of f)

- Insertion of l_i takes time linear in total complexity of faces intersected by the line.

☆ $A(L)$ not simple

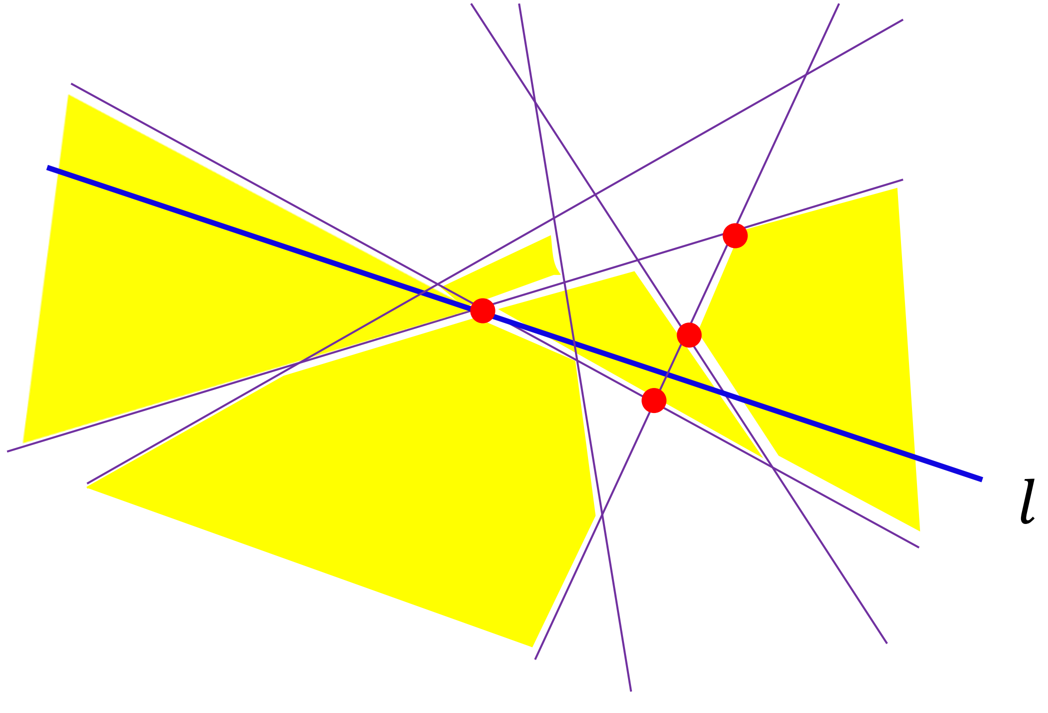
- l_i may leave f through a vertex v where ≥ 3 lines including l_i intersect.

- Walk around v to find the next face (g) to split, scanning over edges that bound faces intersected by l_i.



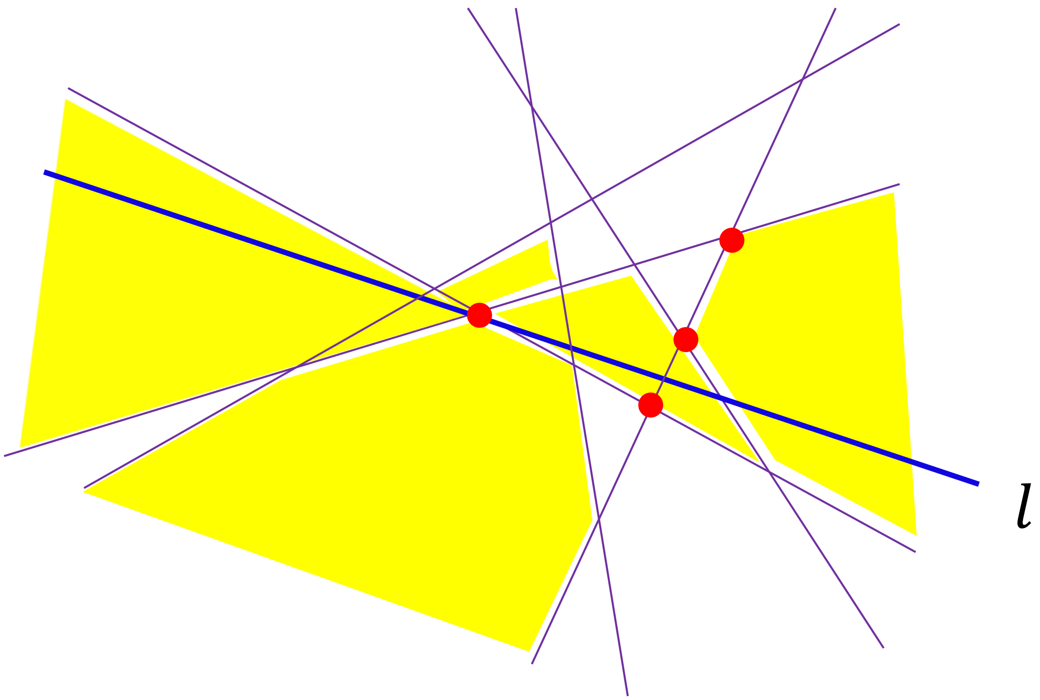
Zone

$Z(l)$ = { f | f is a face of $A(L)$ intersected by l }



Zone

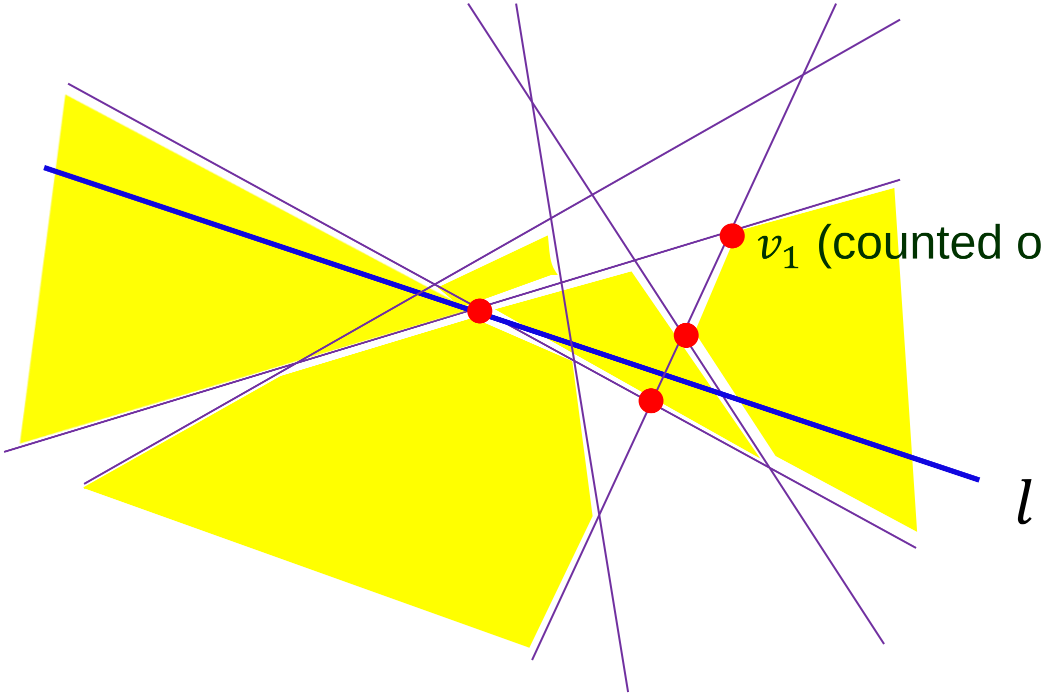
$Z(l)$ = { f | f is a face of $A(L)$ intersected by l }



Complexity of Z = total complexity of faces (#vertices + #edges).

Zone

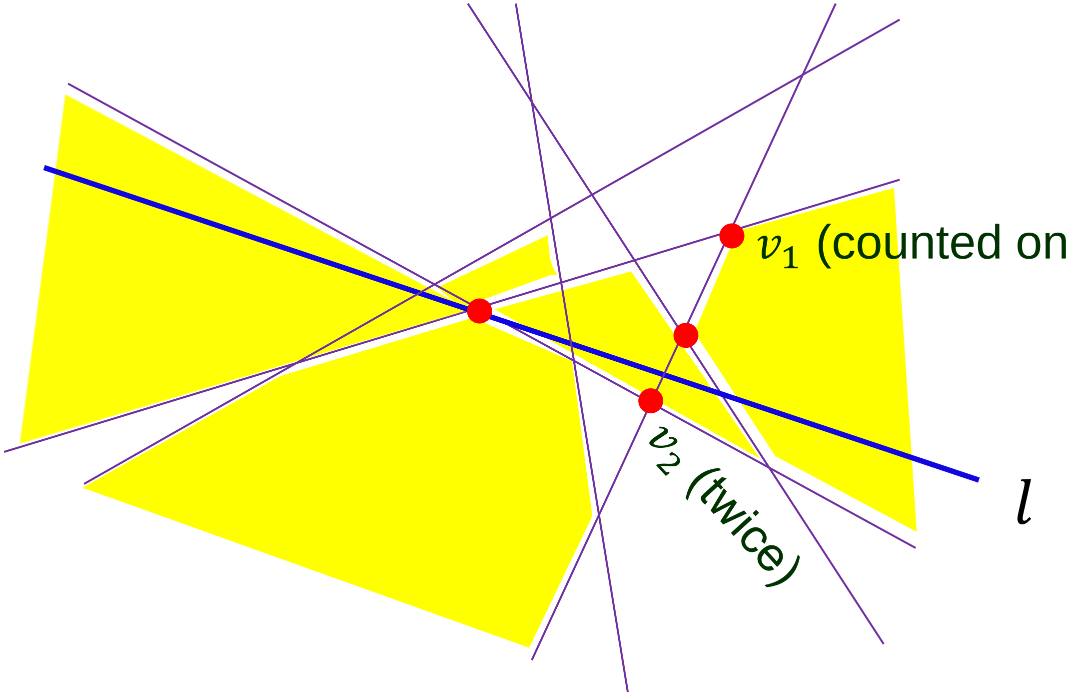
$Z(l)$ = { f | f is a face of $A(L)$ intersected by l }



Complexity of Z = total complexity of faces (#vertices + #edges).

Zone

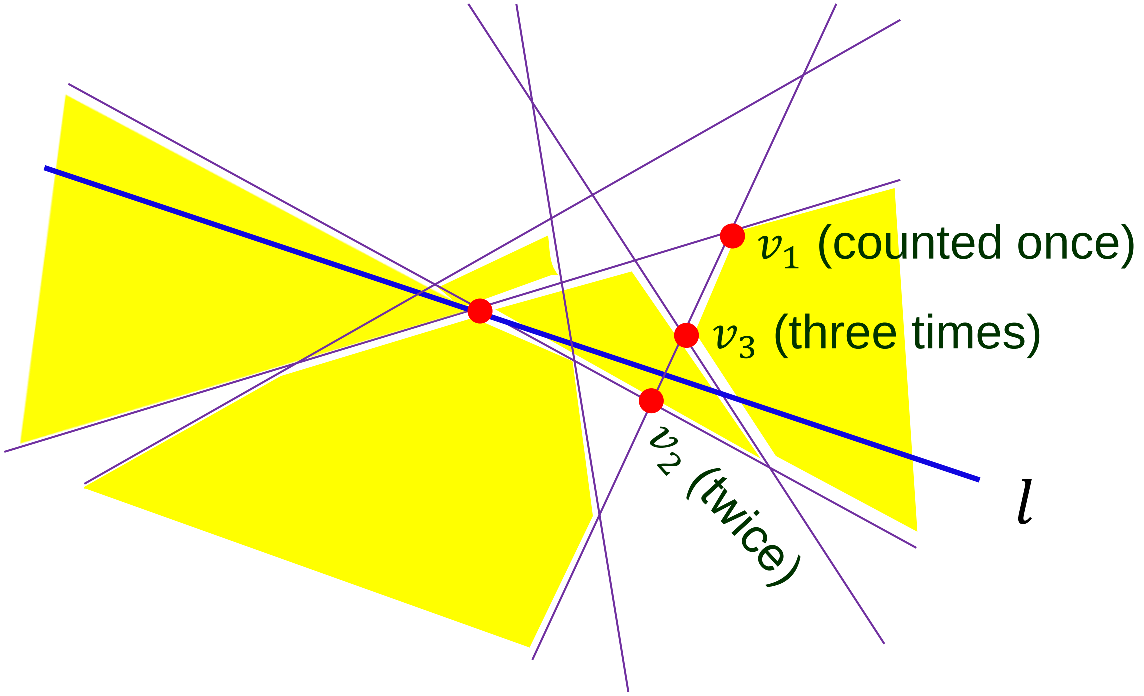
$Z(l)$ = { f | f is a face of $A(L)$ intersected by l }



Complexity of Z = total complexity of faces (#vertices + #edges).

Zone

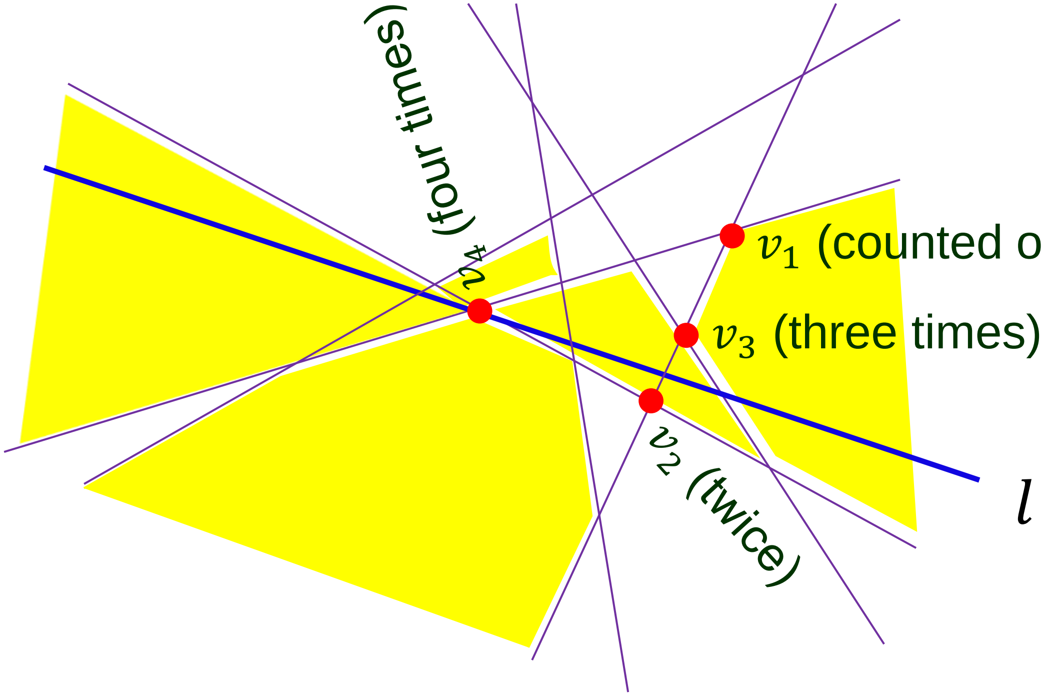
$Z(l) = \{\ f \mid f$ is a face of $A(L)$ intersected by $l\ \}$



Complexity of Z = total complexity of faces (#vertices + #edges).

Zone

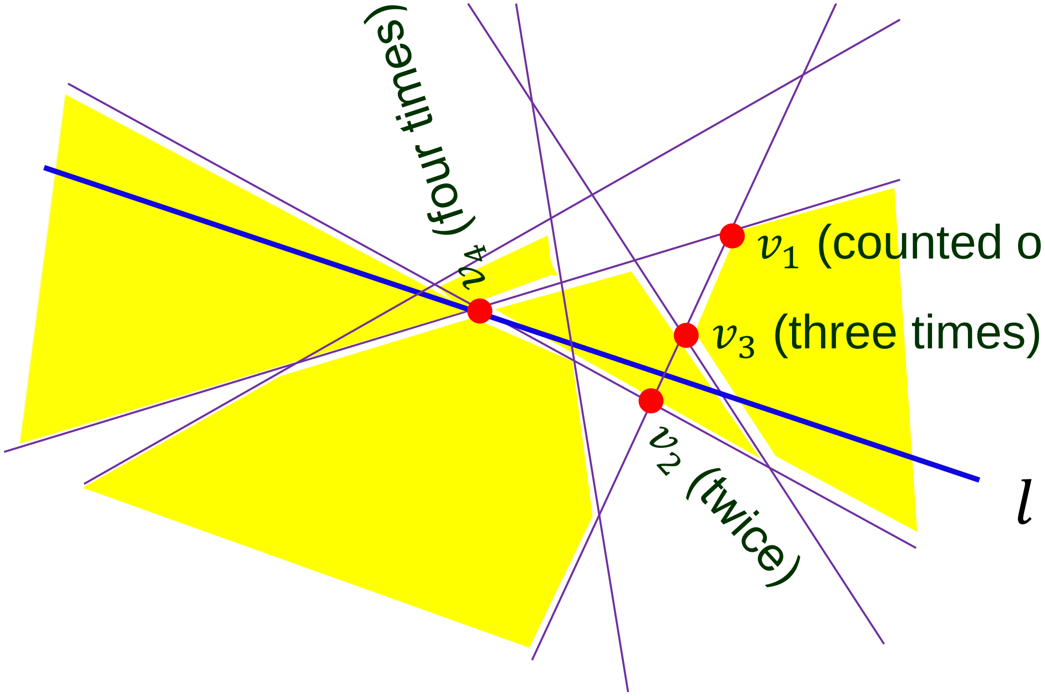
$Z(l) = \{\ f \mid f$ is a face of $A(L)$ intersected by $l\ \}$



Complexity of Z = total complexity of faces (#vertices + #edges).

Zone

$Z(l)$ = { f | f is a face of $A(L)$ intersected by l }



Complexity of Z = total complexity of faces (#vertices + #edges).

A vertex may be counted ≤ 4 times.

Time of Arrangement Construction

Zone Theorem Complexity of zone in an arrangement of m lines is $O(m)$.

Proof By induction.

Time of Arrangement Construction

Zone Theorem Complexity of zone in an arrangement of m lines is $O(m)$.

Proof By induction.

Thus, insertion of one line takes $O(i)$ time.

Time of Arrangement Construction

Zone Theorem Complexity of zone in an arrangement of m lines is $O(m)$.

Proof By induction.

Thus, insertion of one line takes $O(i)$ time.

Time to insert all lines, and thus to construct line arrangement:

Time of Arrangement Construction

Zone Theorem Complexity of zone in an arrangement of m lines is $O(m)$.

Proof By induction.

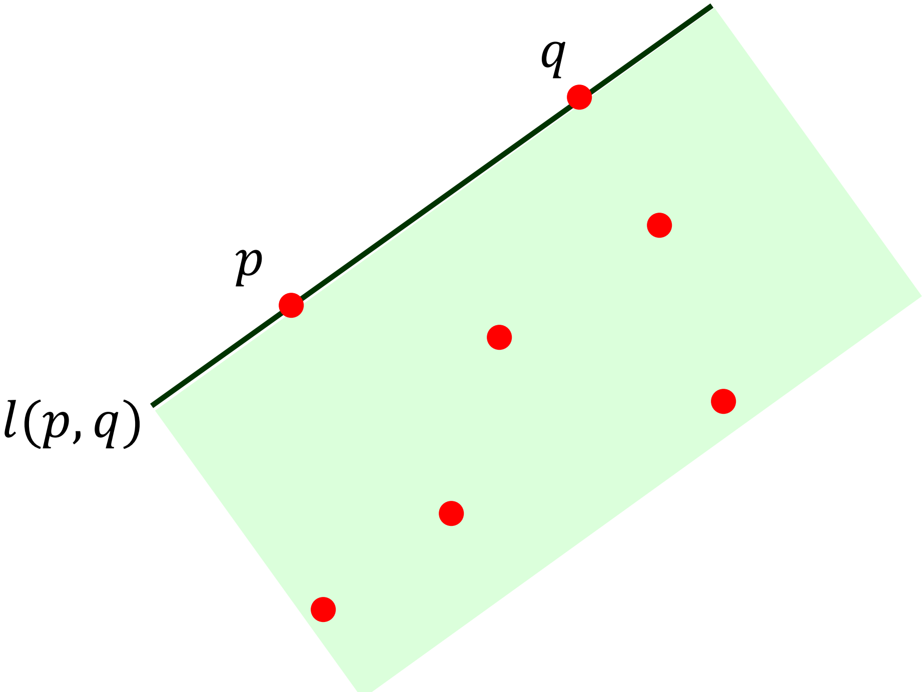
Thus, insertion of one line takes $O(i)$ time.

Time to insert all lines, and thus to construct line arrangement:

$$\sum_{i=1}^{n} O(i) = O(n^2)$$

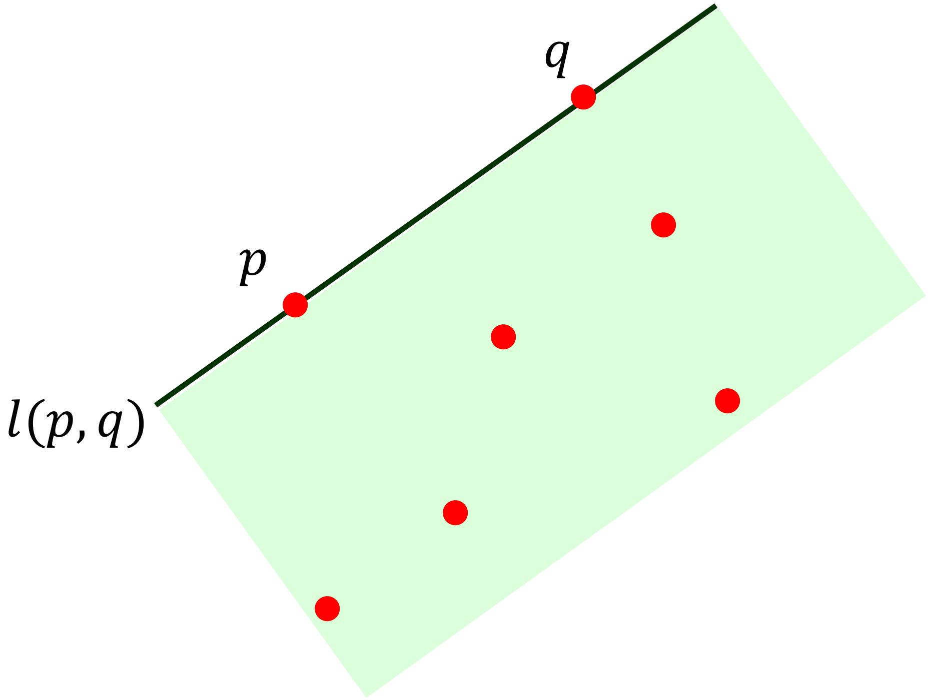
III. Discrete Measure

Primal plane



III. Discrete Measure

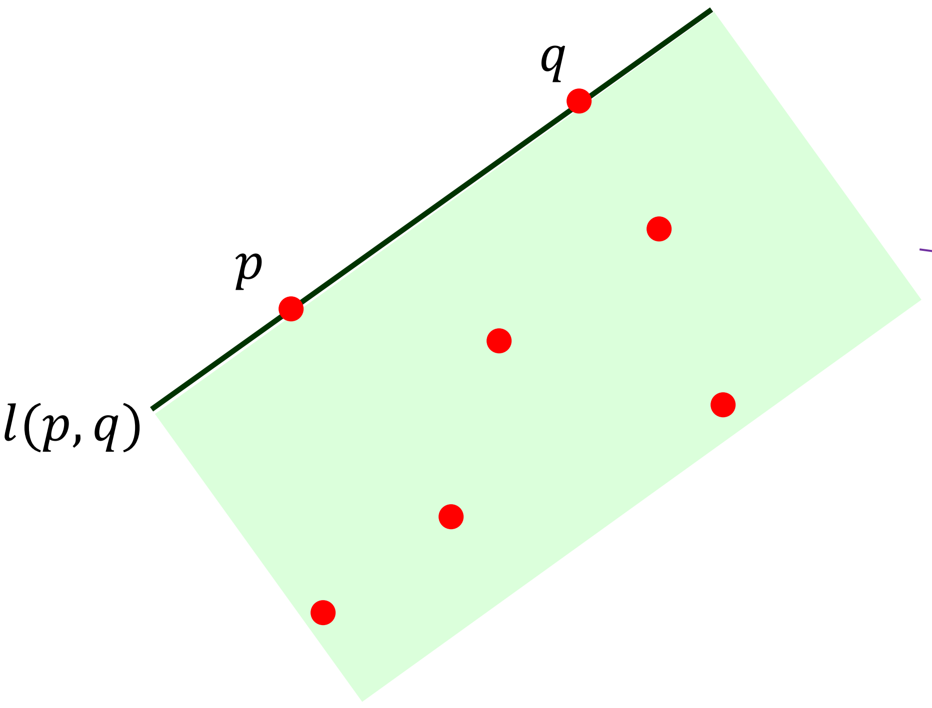
Primal plane



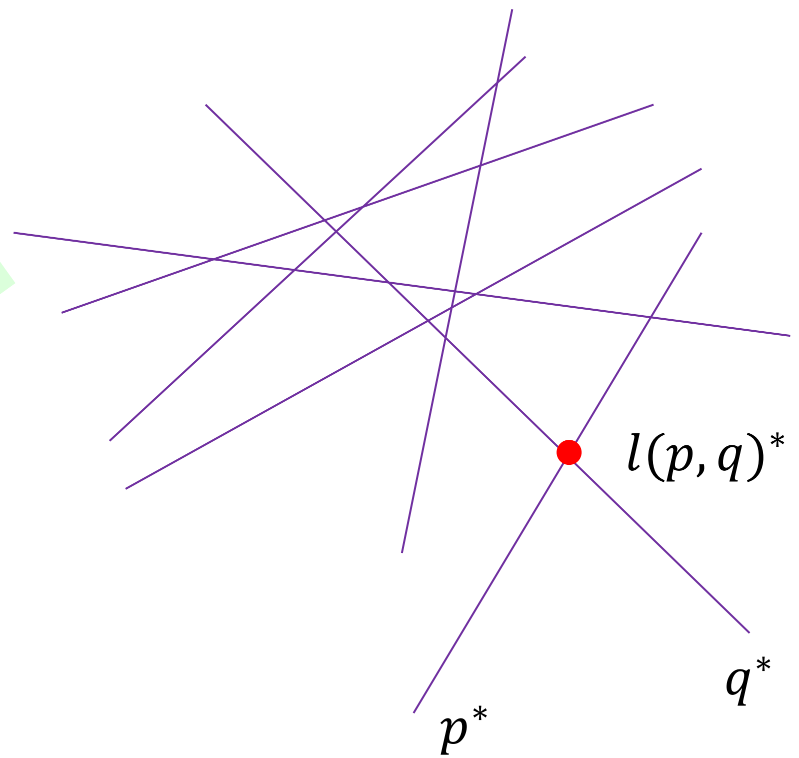
points below $l(p,q)$

III. Discrete Measure

Primal plane



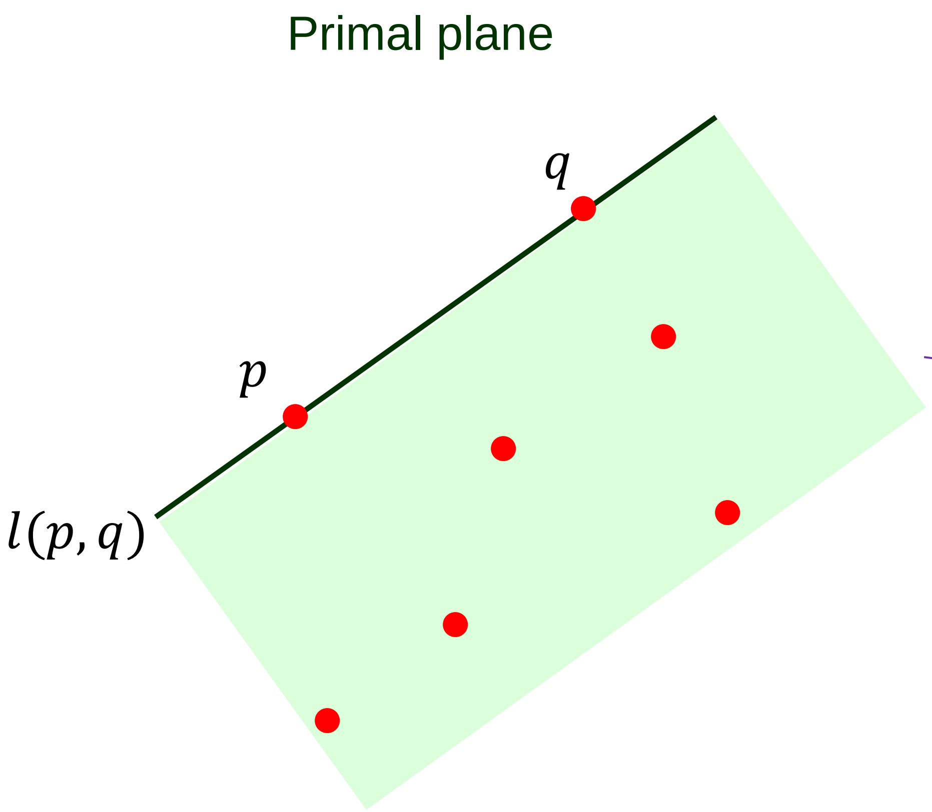
Dual plane

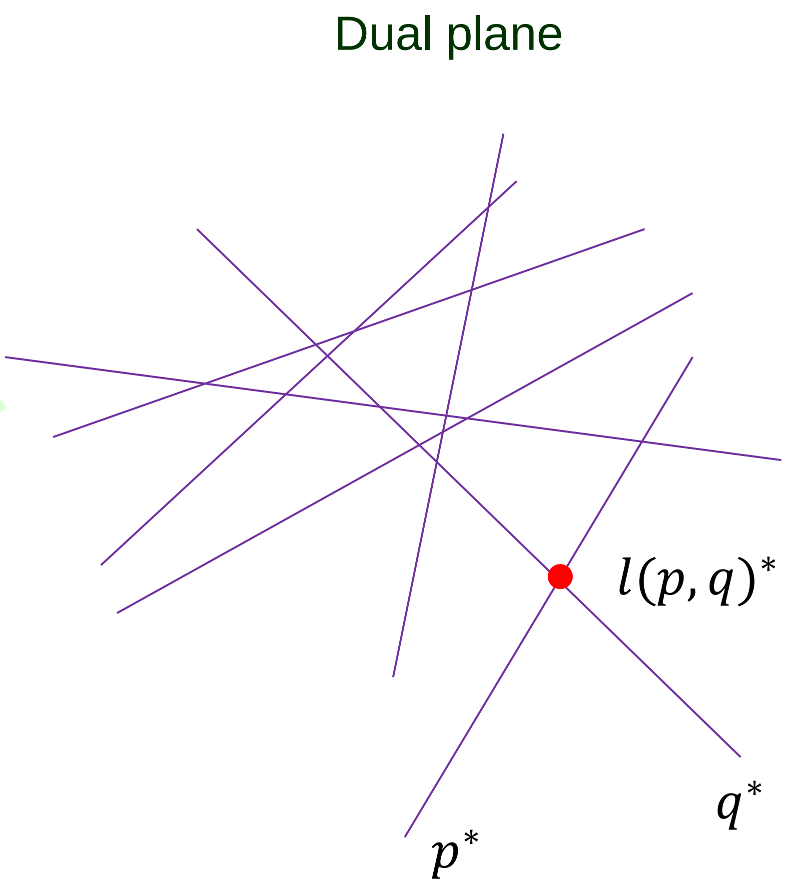


points below $l(p,q)$

$\Leftrightarrow$ lines strictly above dual point $l(p,q)^*$

III. Discrete Measure





points below $l(p,q)$

$\Leftrightarrow$ lines strictly above dual point $l(p,q)^*$

Efficient algorithm exists!

How to Use Duality?

A set S of n sample points

How to Use Duality?

A set S of n sample points

$\Longrightarrow$ A set S^* of n (dual) lines

How to Use Duality?

A set S of n sample points

$\Longrightarrow$ A set S^* of n (dual) lines

A line through $\geq\ 2$ sample points

How to Use Duality?

A set S of n sample points

$\Longrightarrow$ A set S^* of n (dual) lines

A line through ≥ 2 sample points

$\Longrightarrow$ A vertex in the arrangement $A(S^*)$

How to Use Duality?

A set S of n sample points

$\Longrightarrow$ A set S^* of n (dual) lines

A line through ≥ 2 sample points

$\Longrightarrow$ A vertex in the arrangement $A(S^*)$

Because p lies above l iff l^* lies above p^*, the discrepancy problem reduces to the following:

How to Use Duality?

A set S of n sample points

$\Longrightarrow$ A set S^* of n (dual) lines

A line through $\geq$ 2 sample points

$\Longrightarrow$ A vertex in the arrangement $A(S^*)$

Because p lies above l iff l^* lies above p^*, the discrepancy problem reduces to the following:

Problem For every vertex in $A(S^*)$, compute the numbers of lines above it, passing through it, and below it.

Reduction

$n_a =$ # lines above a vertex

$n_b =$ # lines below the vertex

$n_o =$ # lines through the vertex

Reduction

$n_a =$ # lines above a vertex

$n_b =$ # lines below the vertex

$n_o =$ # lines through the vertex

- ☆ Sufficient to compute 2 of 3 numbers (with sum n).

Reduction

$n_a =$ # lines above a vertex

$n_b =$ # lines below the vertex

$n_o =$ # lines through the vertex

- ☆ Sufficient to compute 2 of 3 numbers (with sum n).
- ☆ n_o is known from DCEL.

Reduction

$n_a =$ # lines above a vertex

$n_b =$ # lines below the vertex

$n_o =$ # lines through the vertex

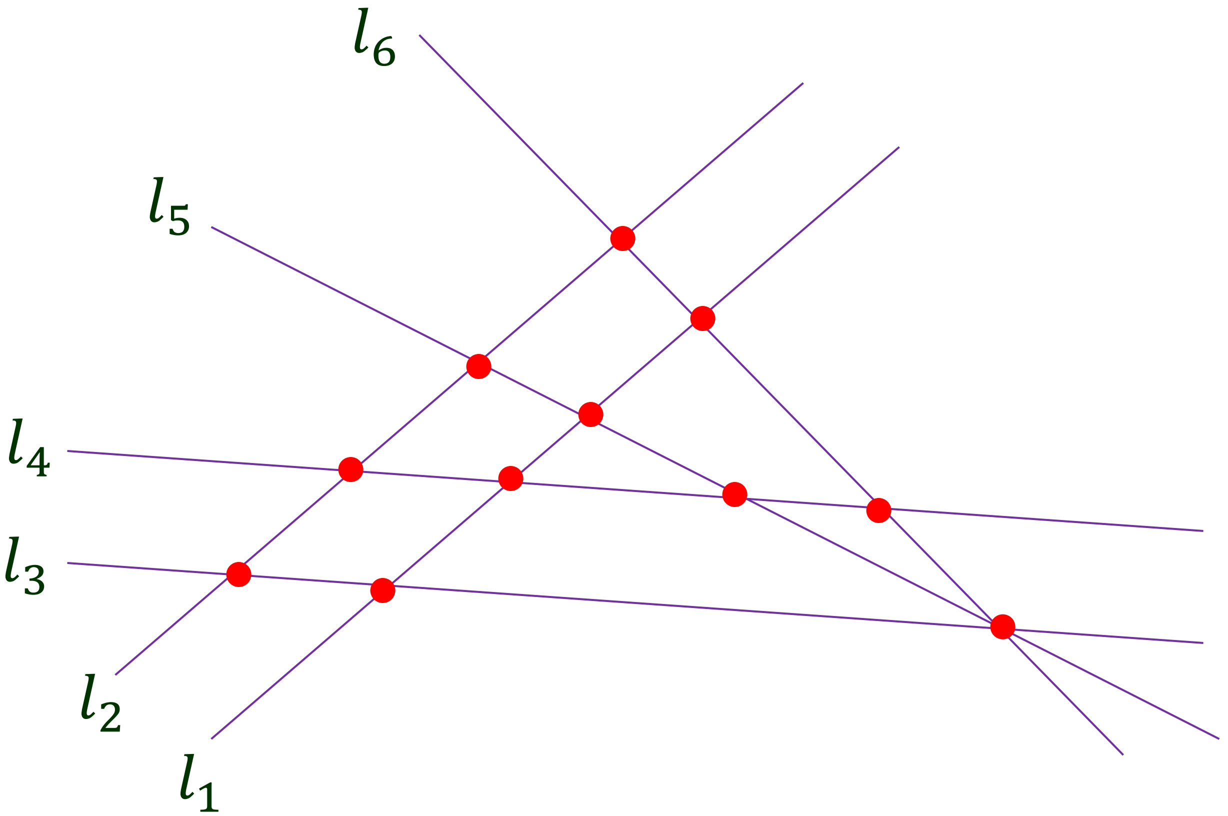
- ☆ Sufficient to compute 2 of 3 numbers (with sum n).
- ☆ n_o is known from DCEL.

Need only compute n_a of every vertex in $A(S^*)$.

Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

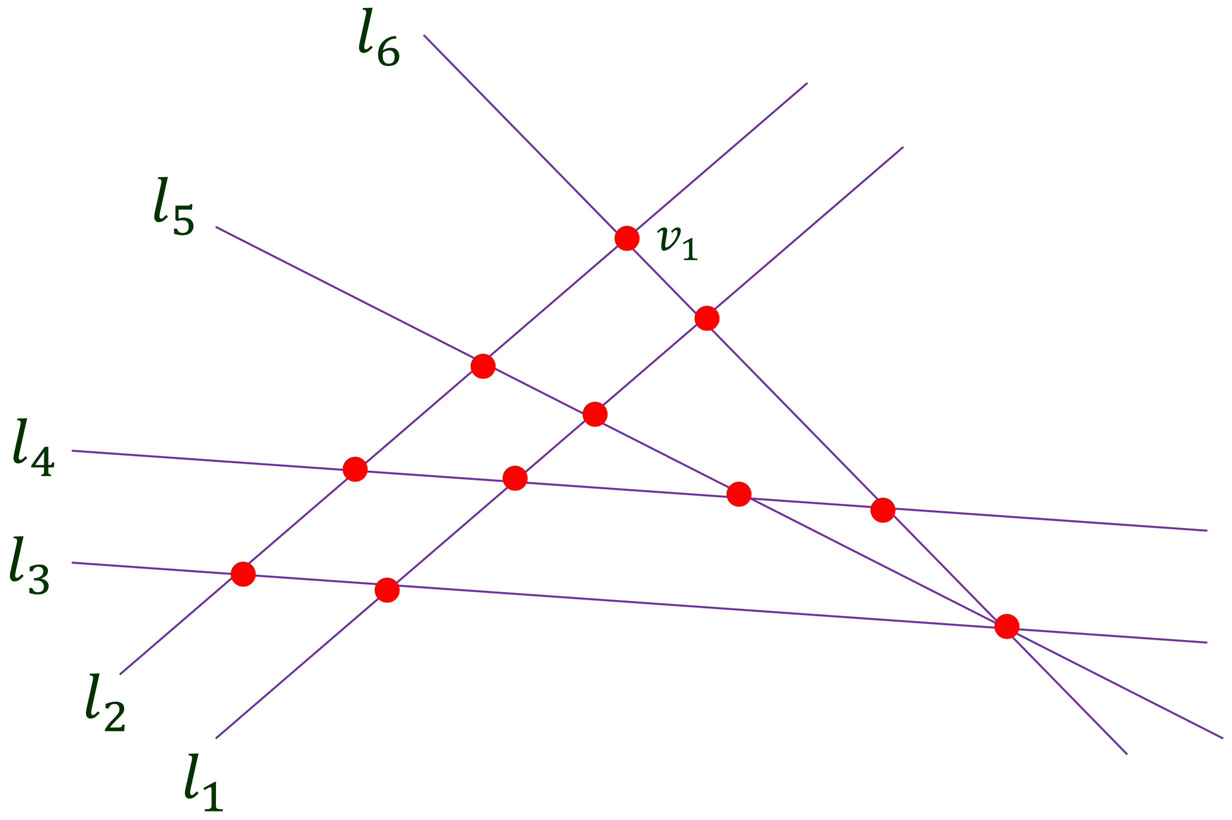
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

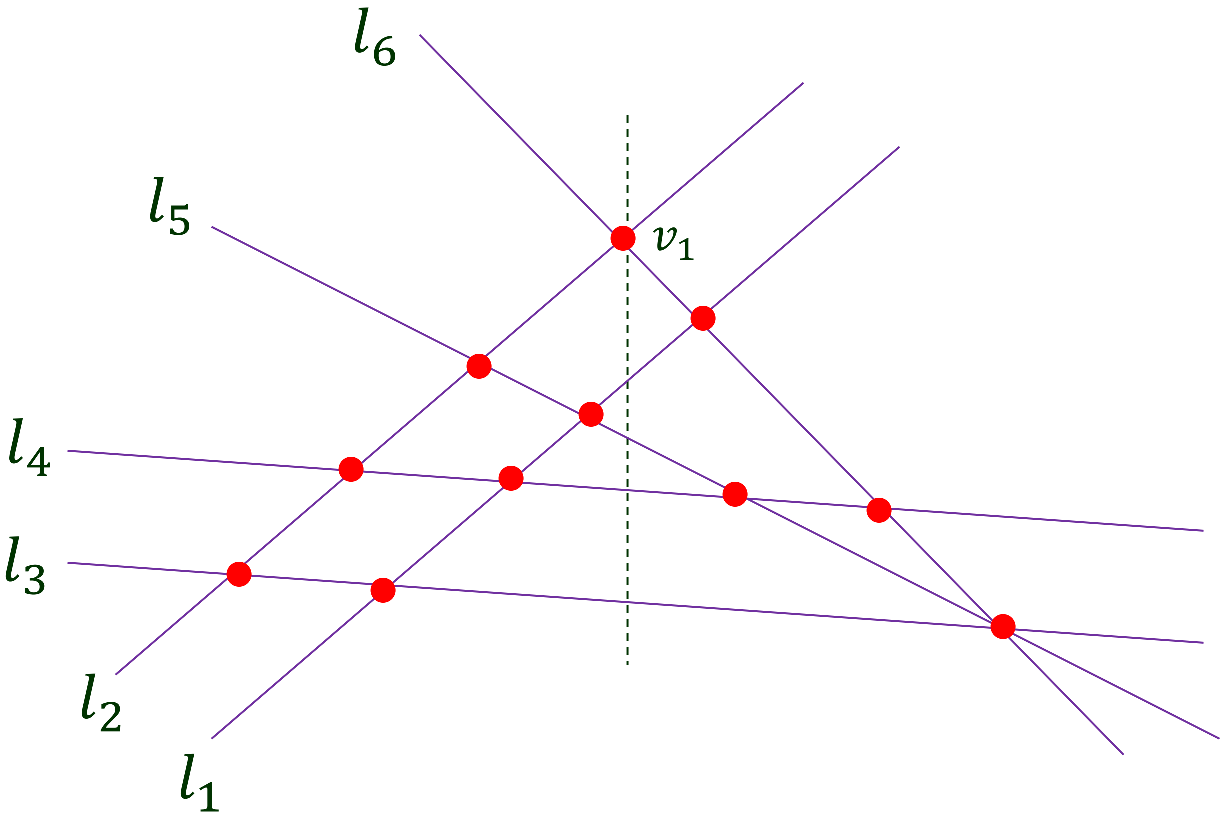
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

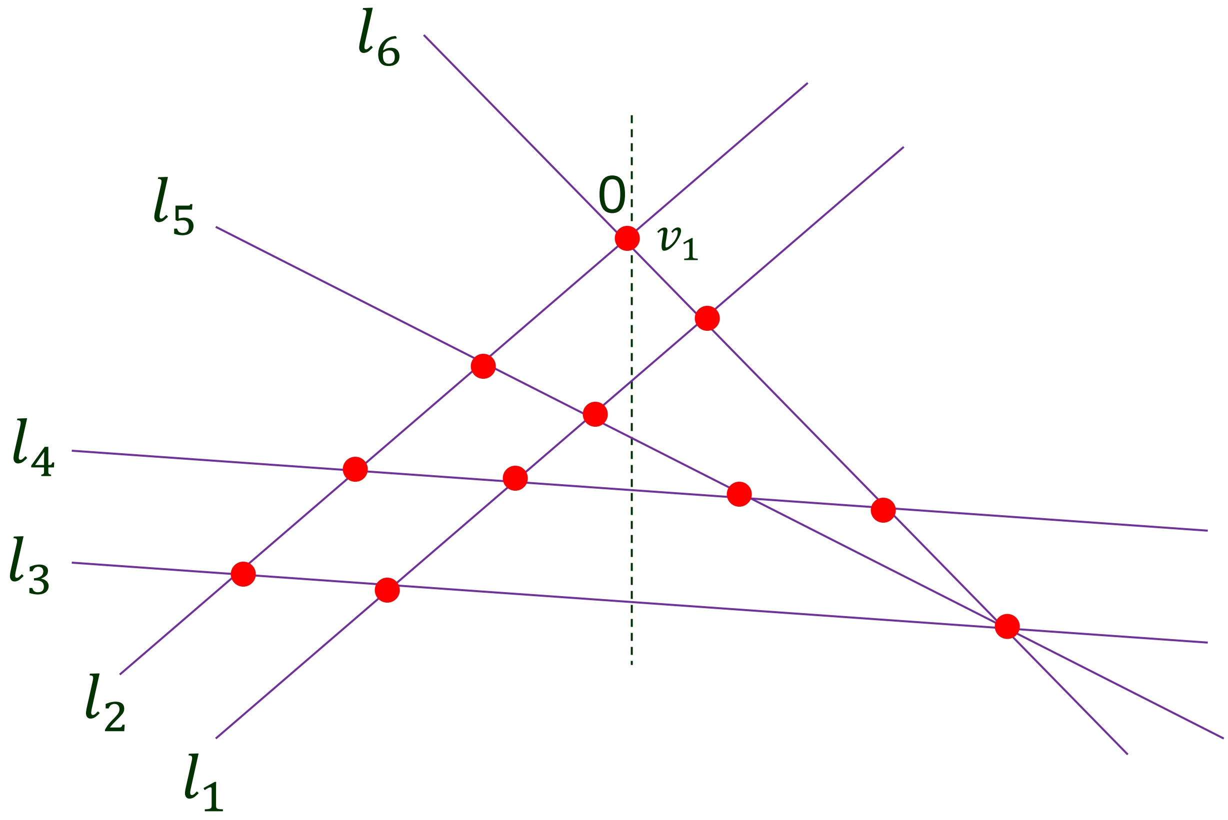
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

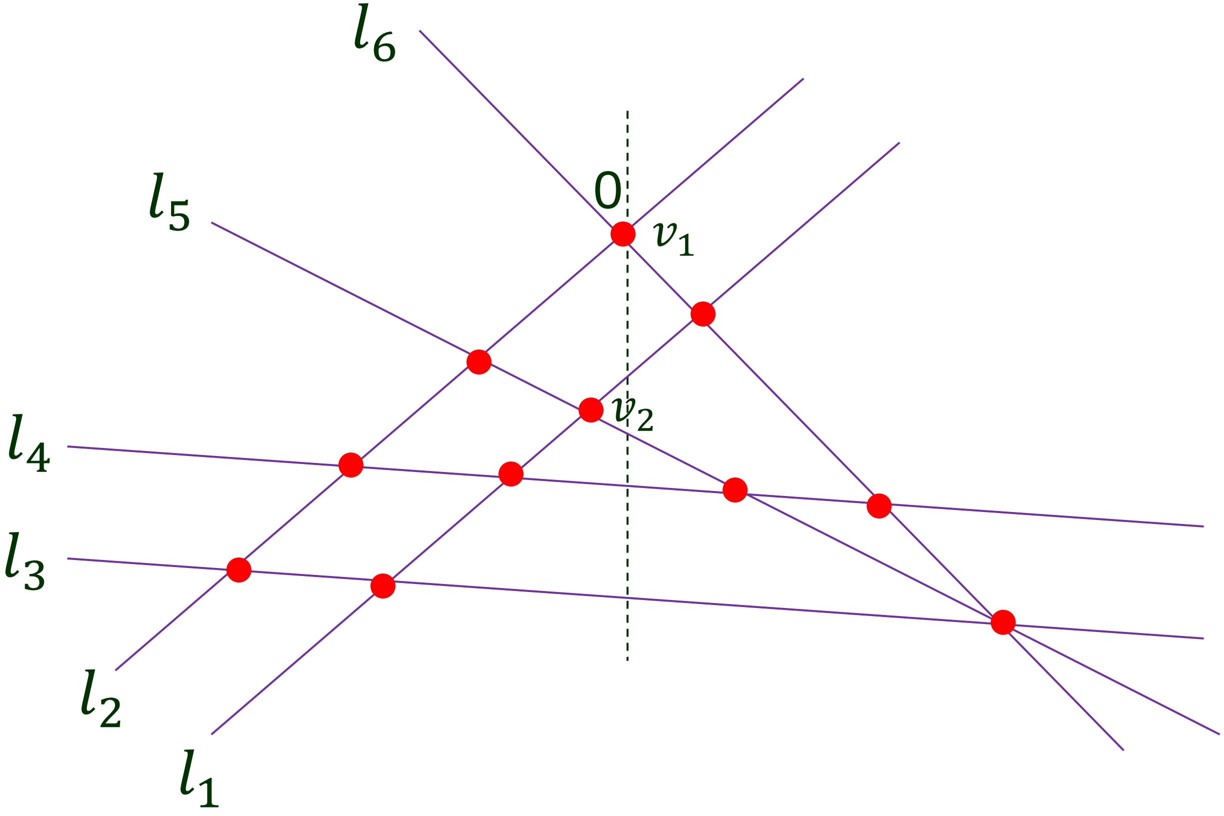
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

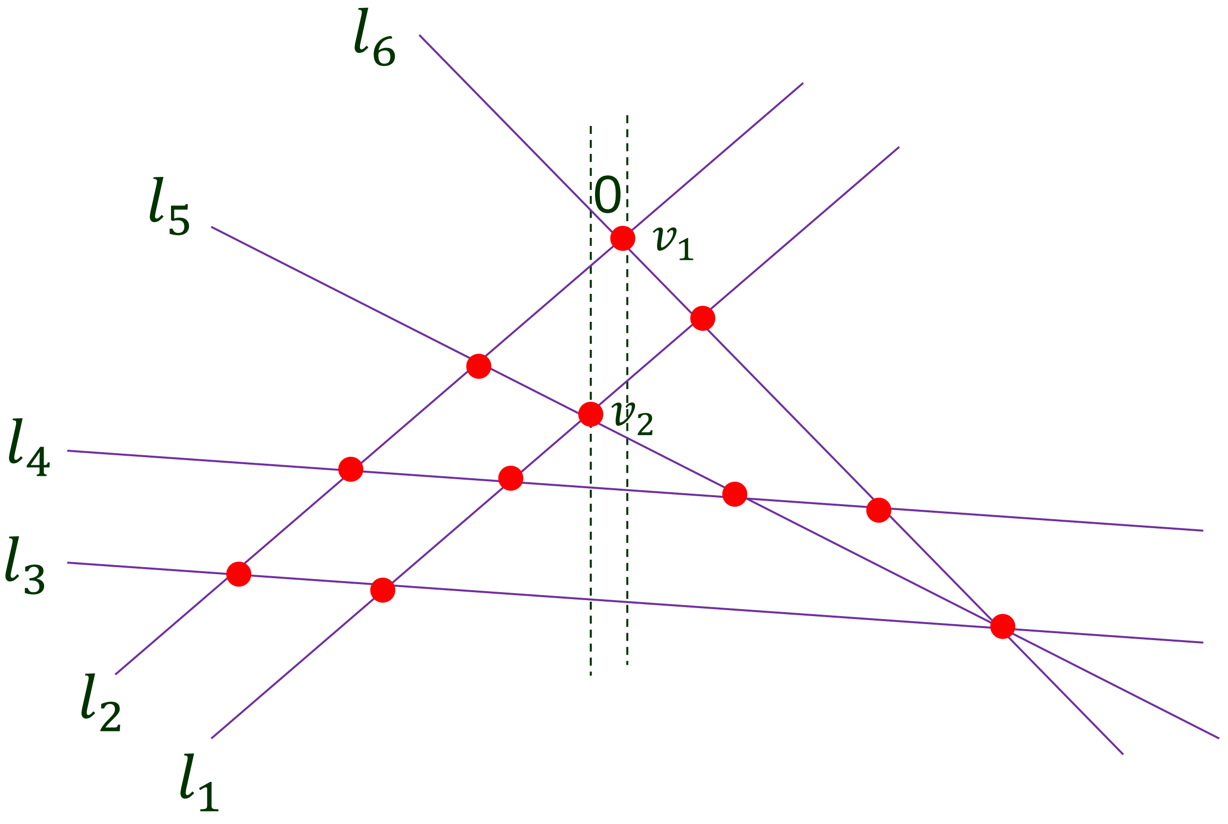
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

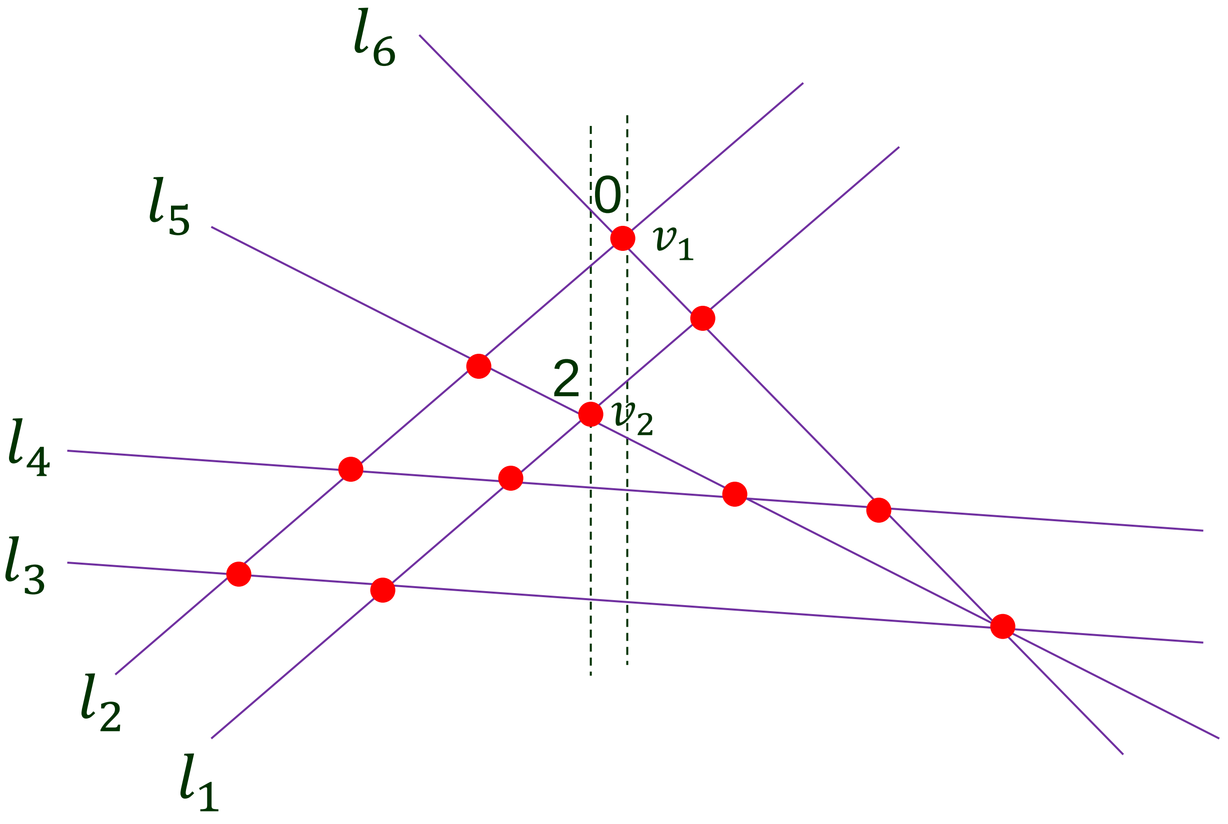
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

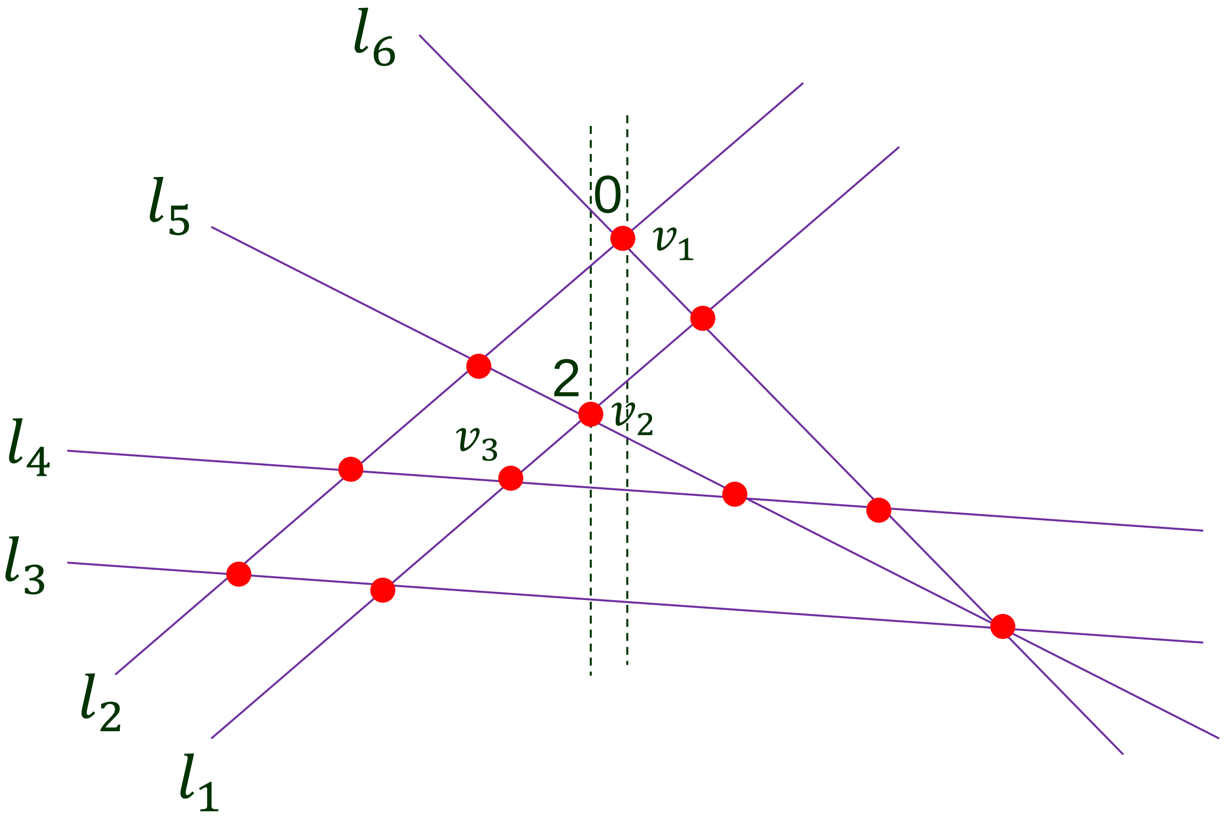
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

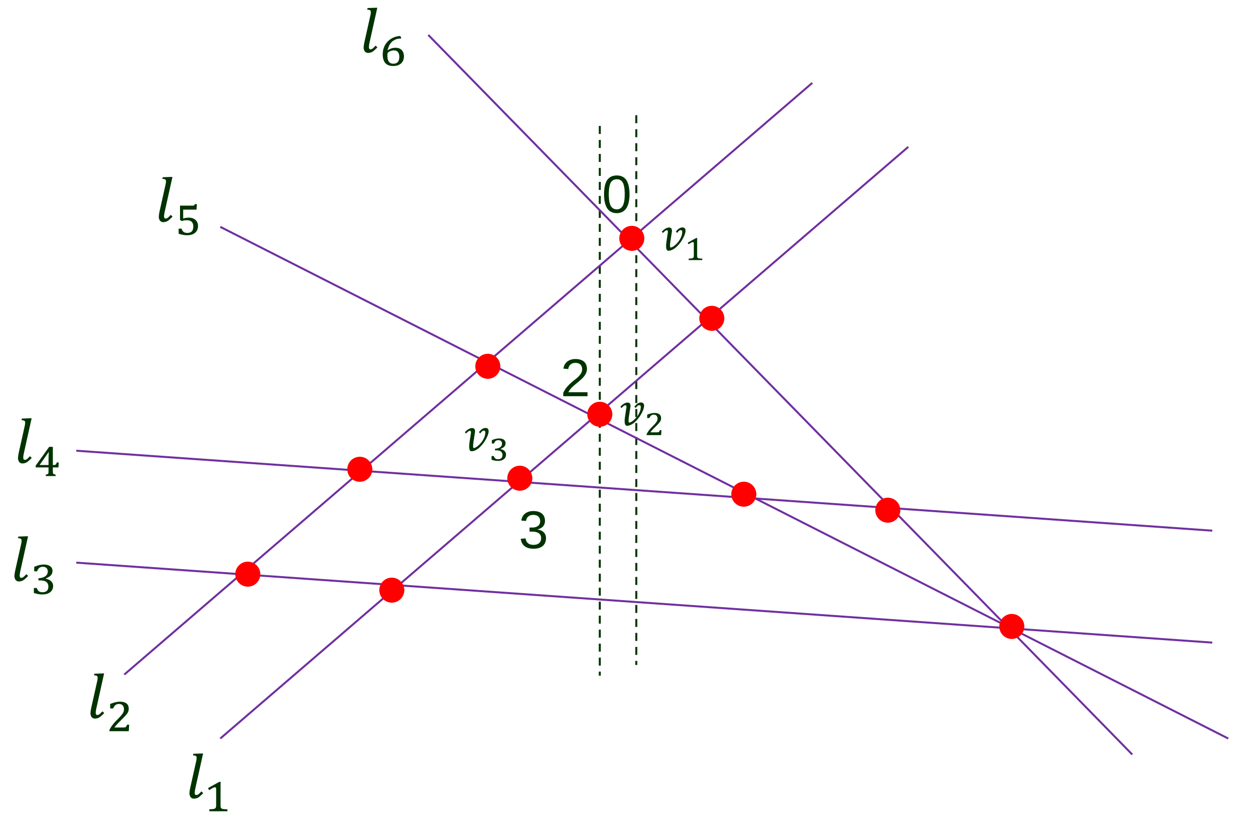
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

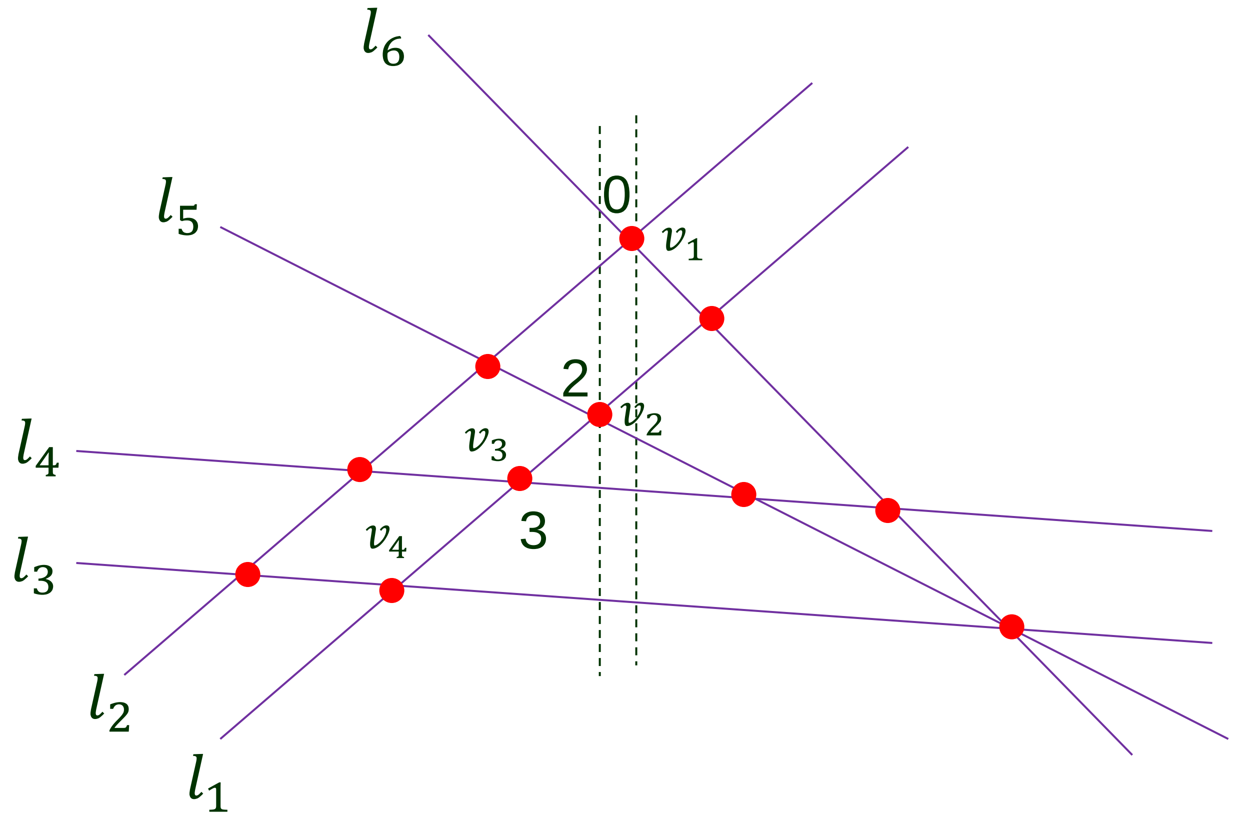
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

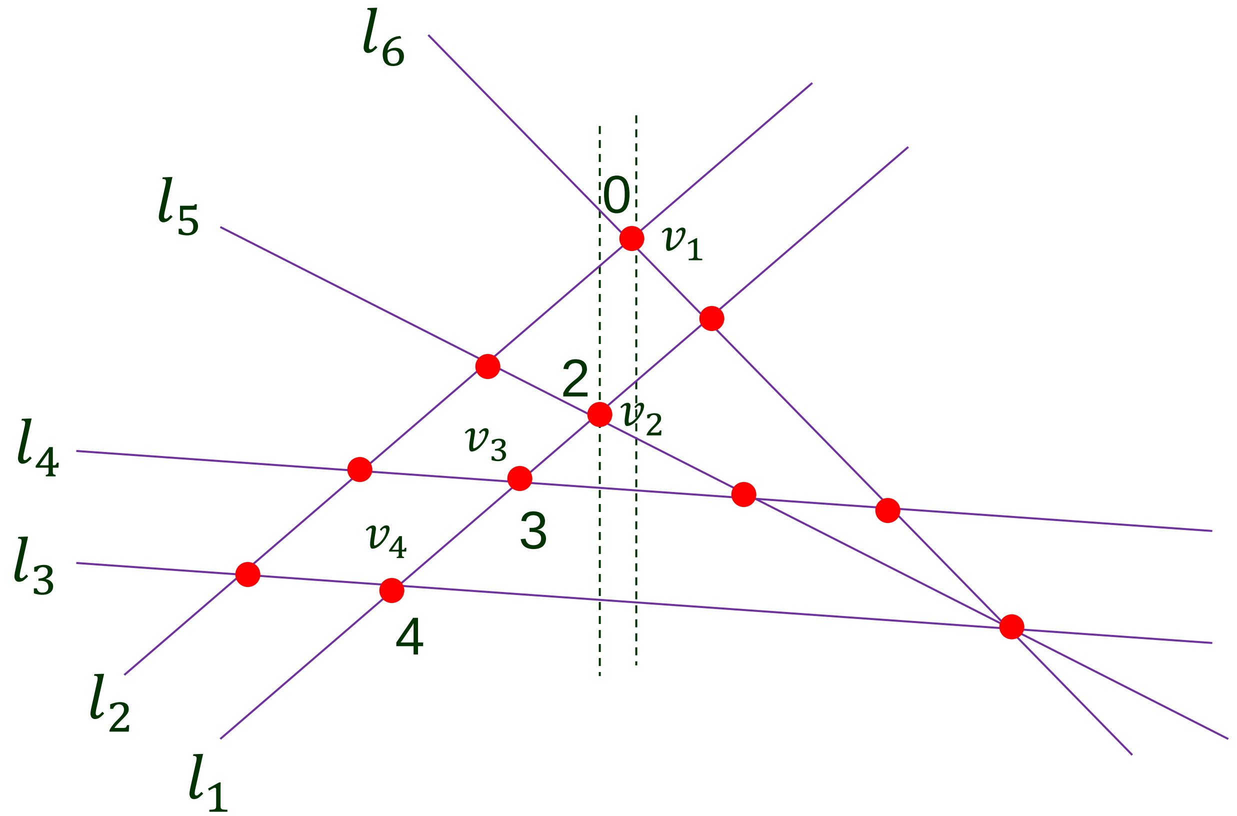
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

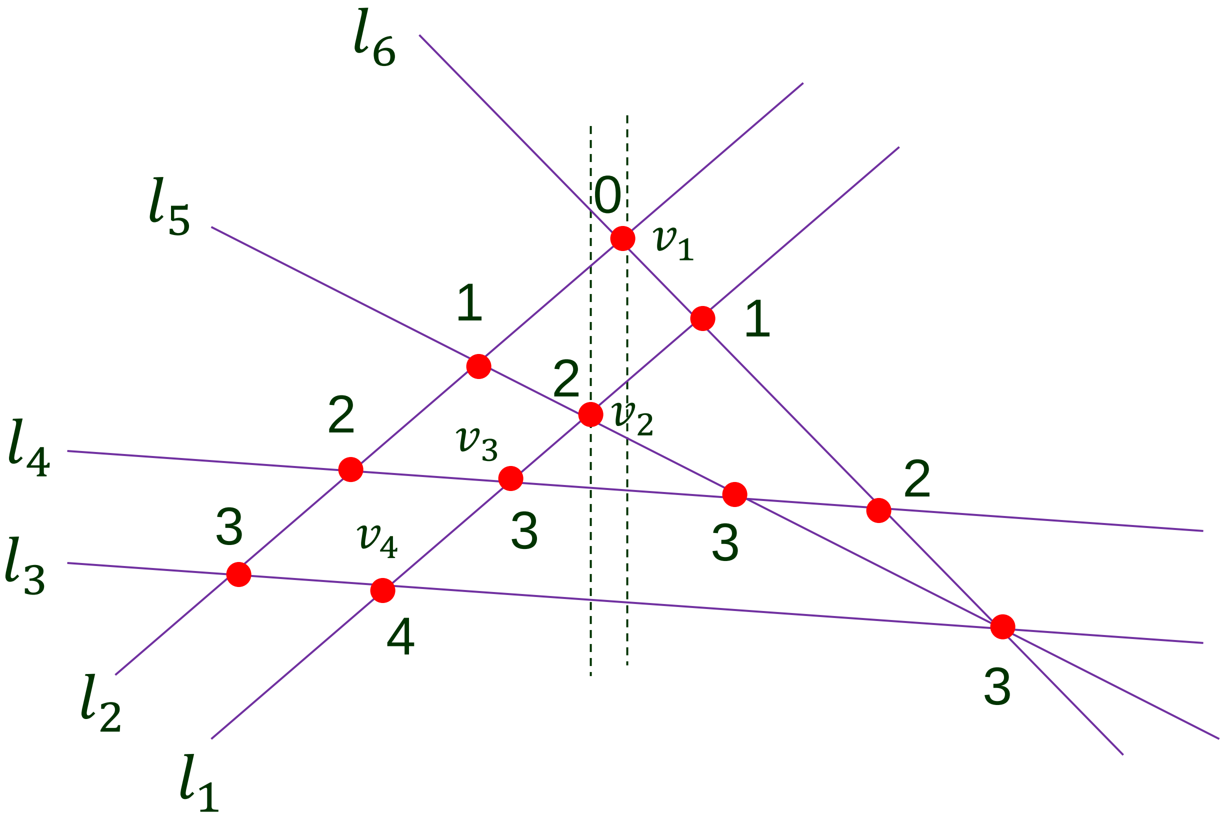
A line l is *above* a point p if its intersection with the vertical line through p is above p.



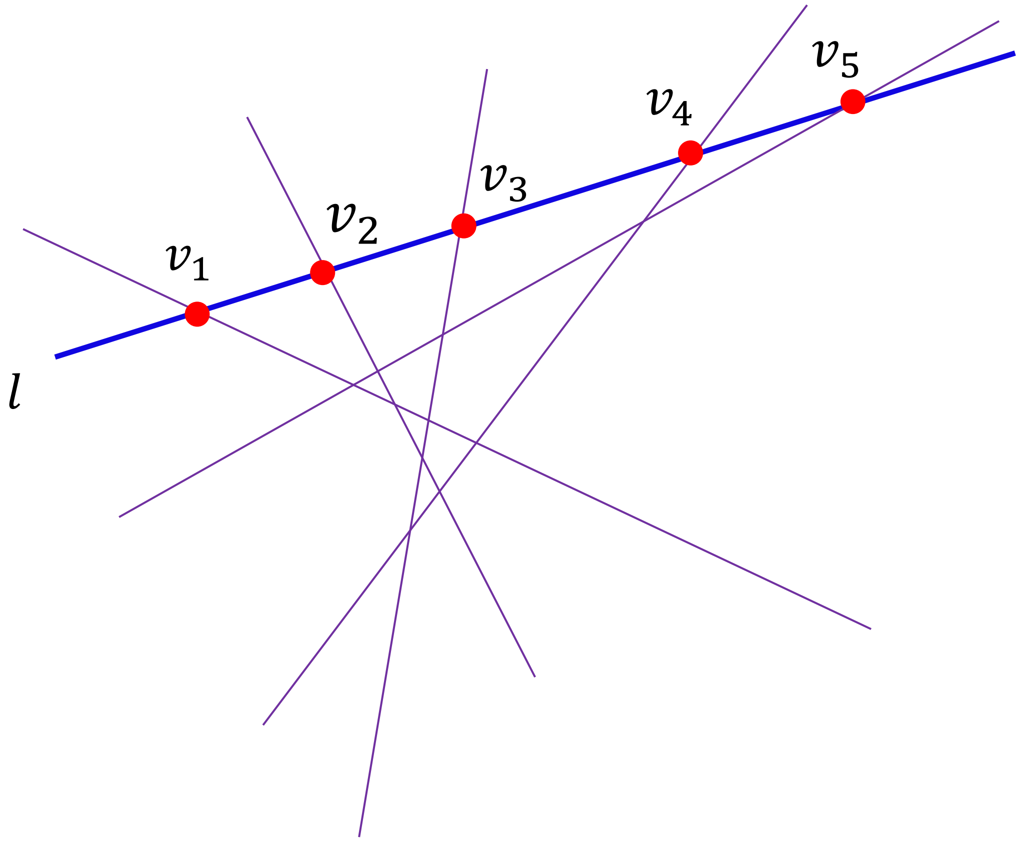
Levels of Vertices in an Arrangement

level of a point = # lines strictly above it.

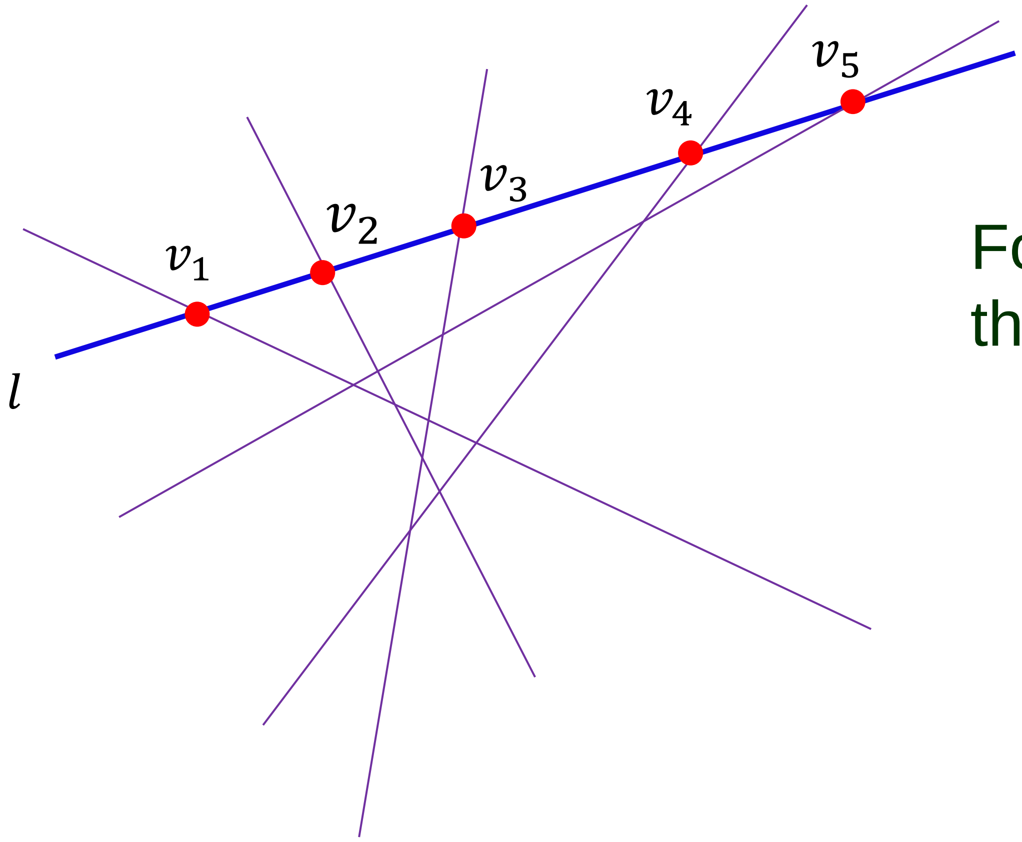
A line l is *above* a point p if its intersection with the vertical line through p is above p.



Counting Levels of Vertices

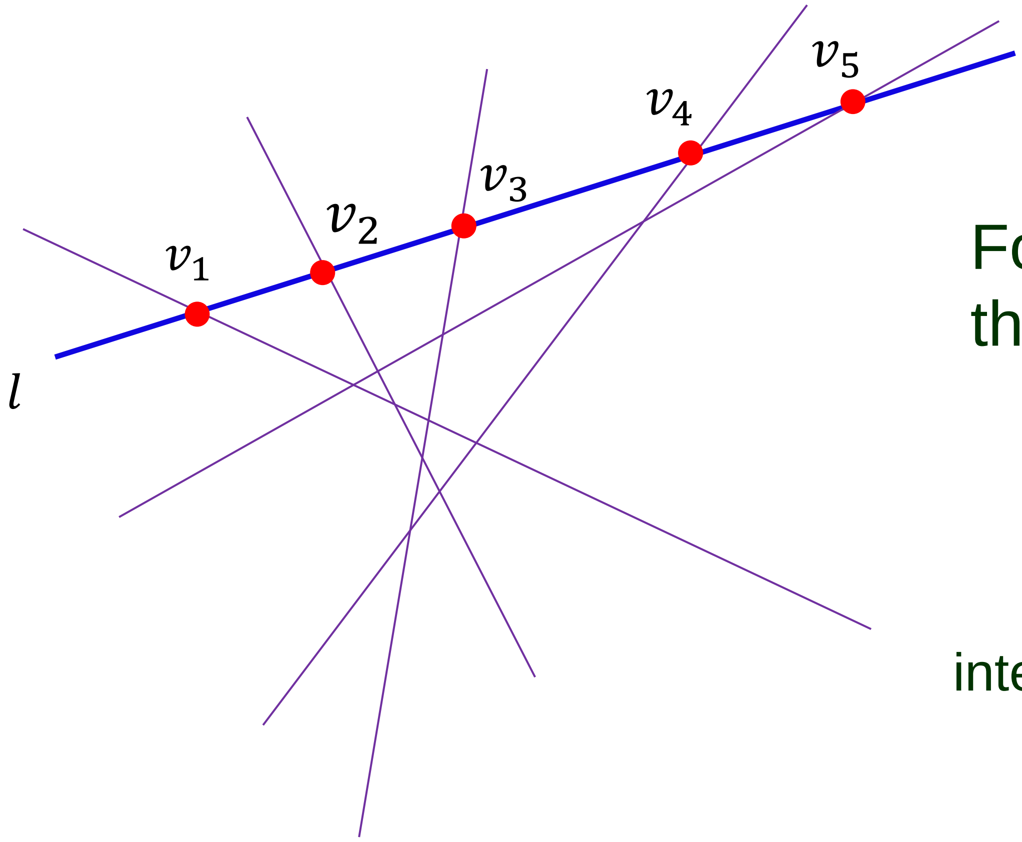


Counting Levels of Vertices



For each line l, compute the levels of vertices on it.

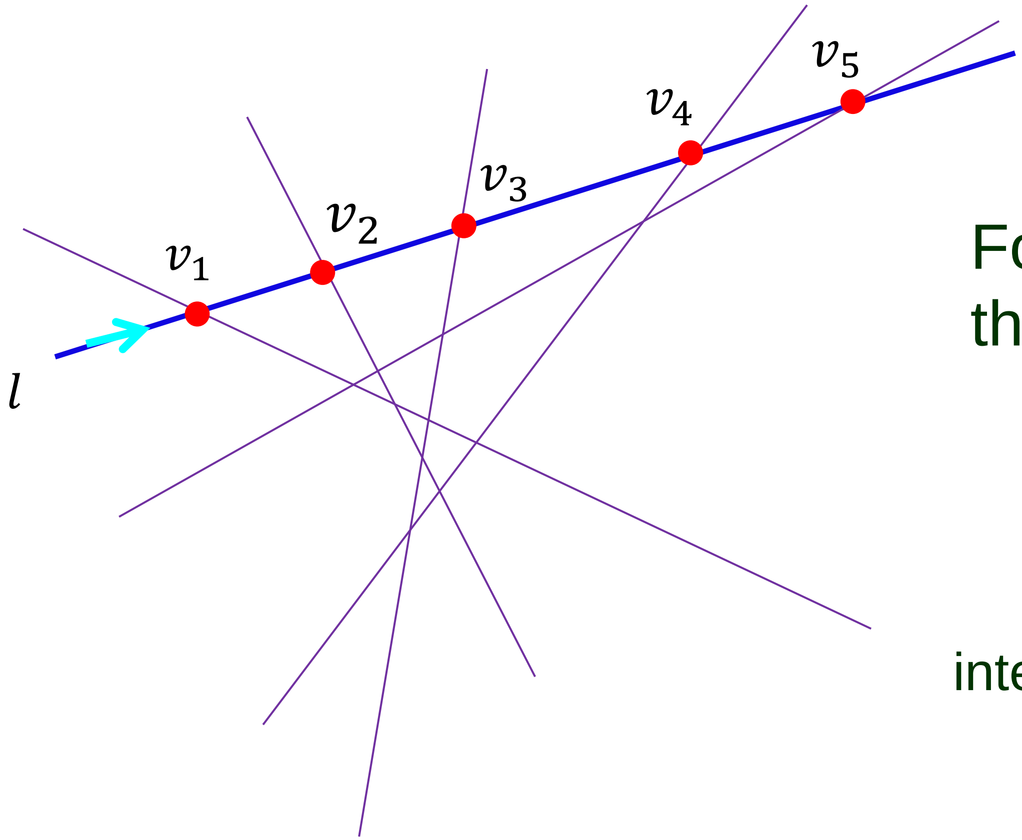
Counting Levels of Vertices



For each line l, compute the levels of vertices on it.

$$\underbrace{v_1, v_2, \ldots, v_{n-1}}_{\text{intersections with } n-1 \text{ other lines}}$$

Counting Levels of Vertices

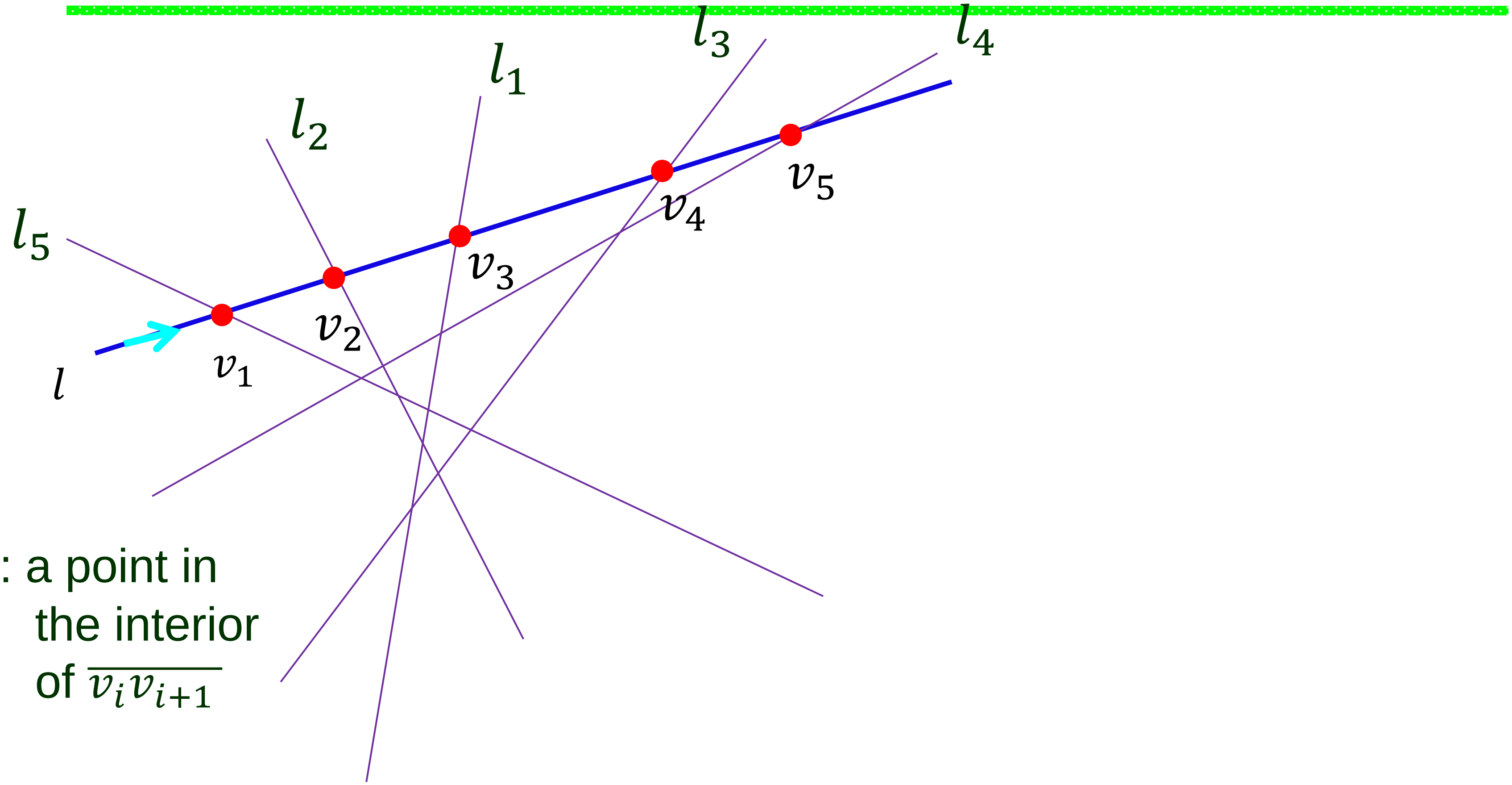


For each line l, compute the levels of vertices on it.

$$\underbrace{v_1, v_2, \ldots, v_{n-1}}_{\text{intersections with } n-1 \text{ other lines}}$$

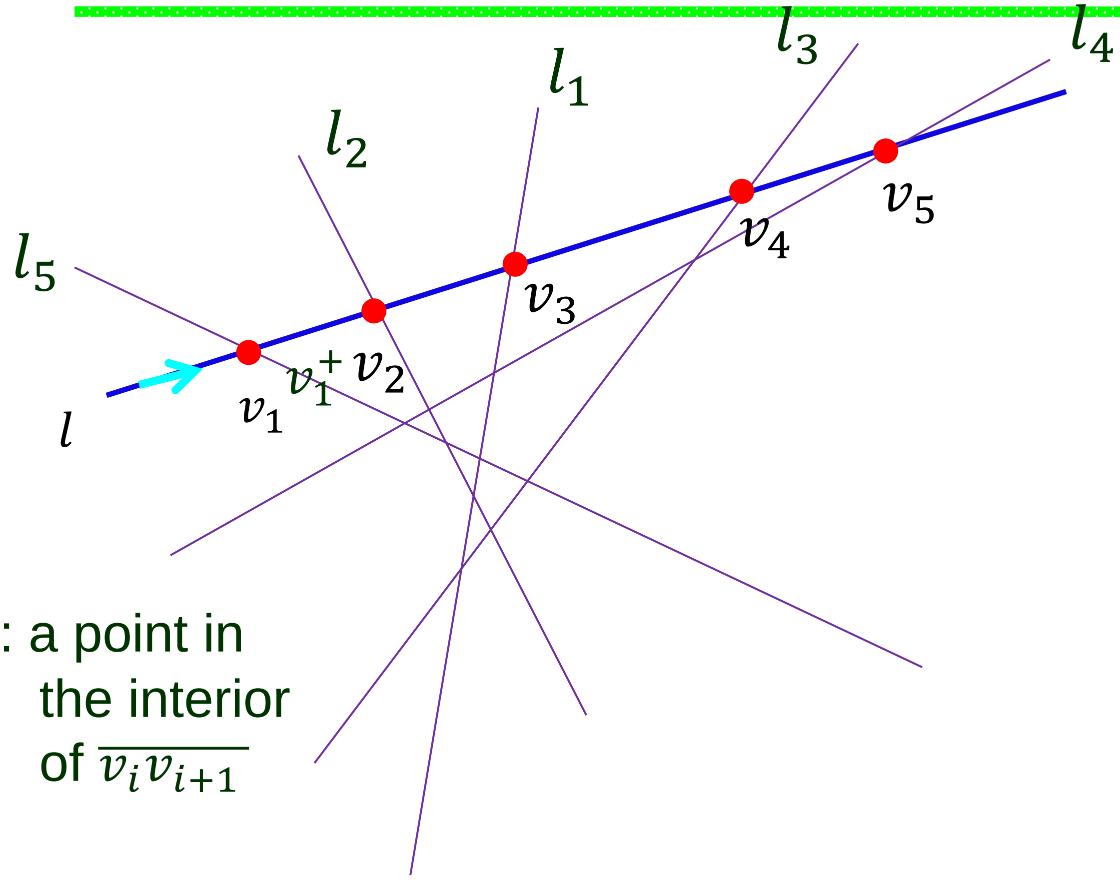
Strategy: walk along l from left to right.

Counting Levels of Vertices



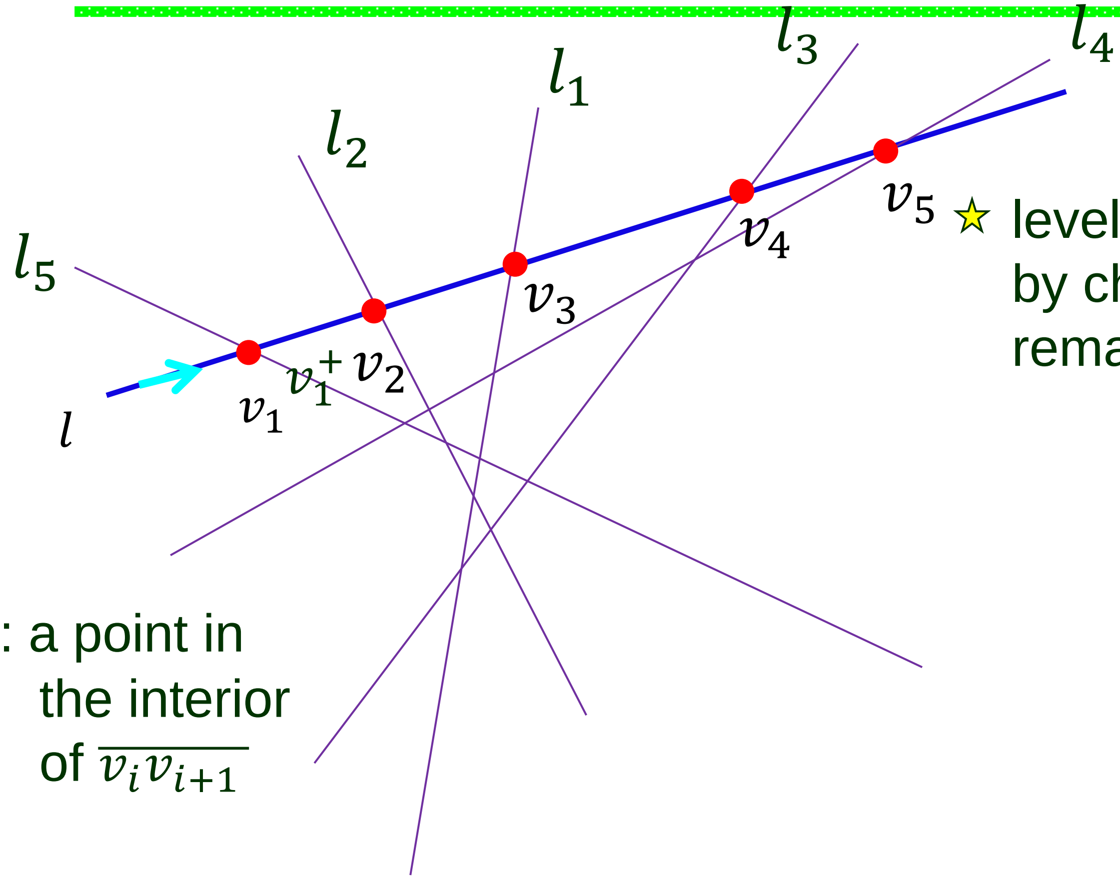
v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

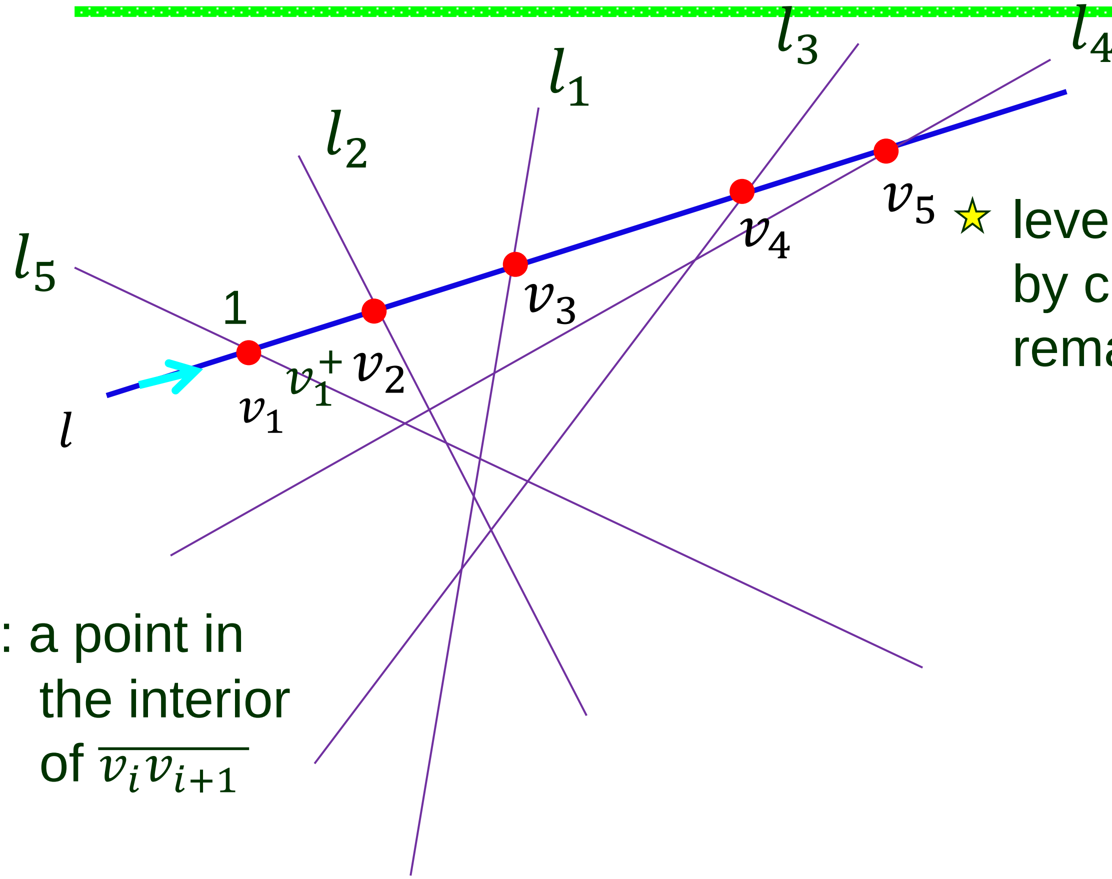
Counting Levels of Vertices



★ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

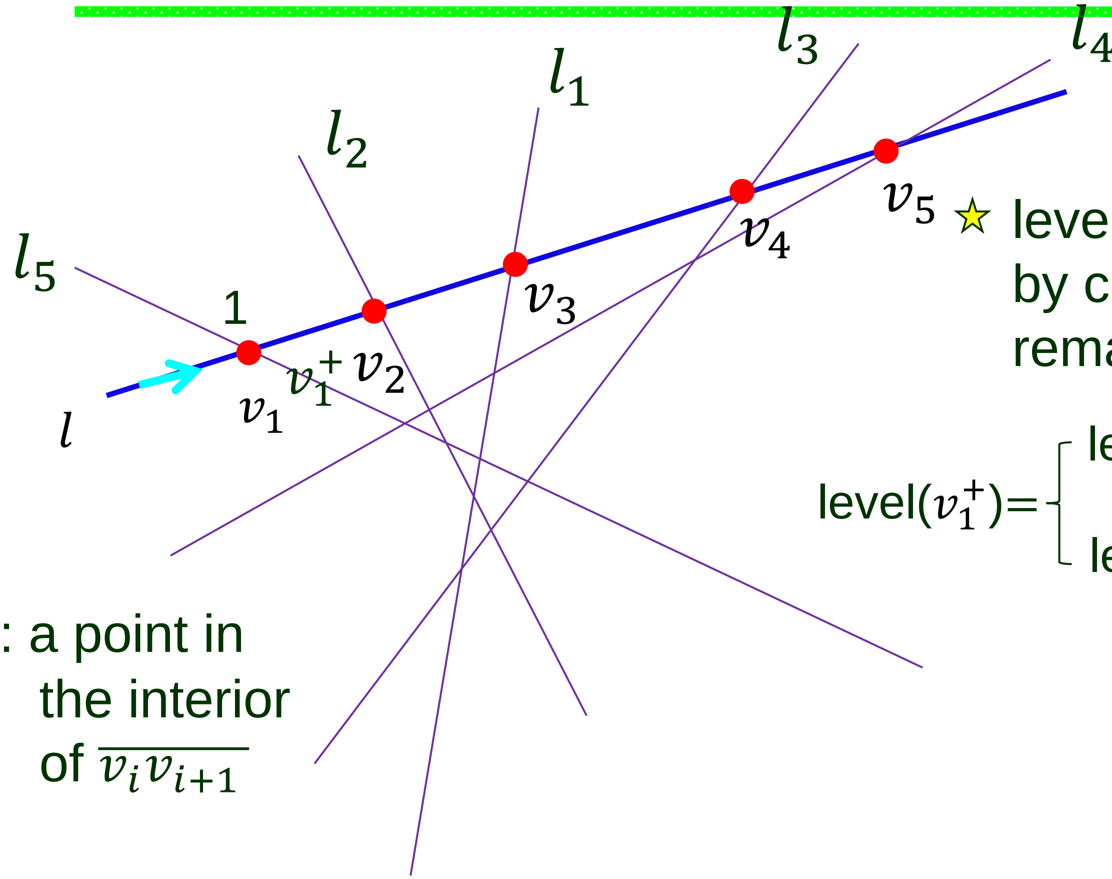
Counting Levels of Vertices



★ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

Counting Levels of Vertices

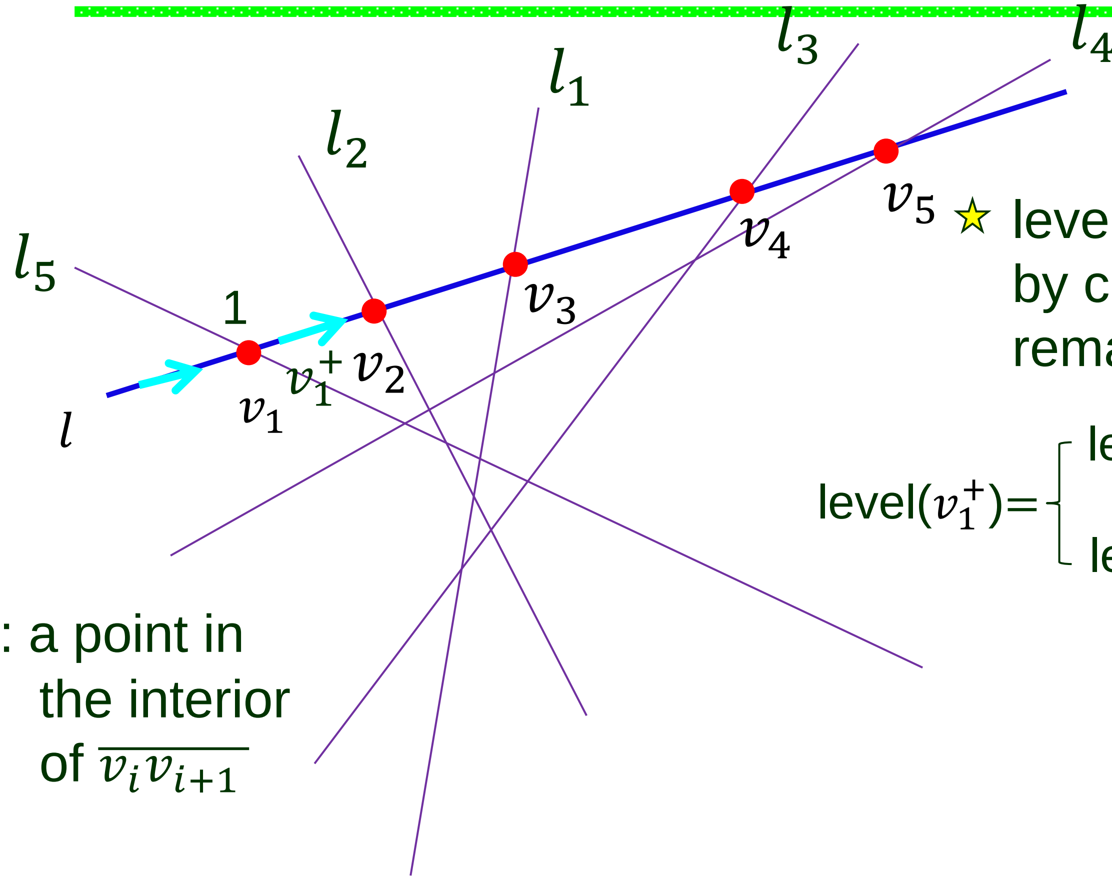


☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

Counting Levels of Vertices

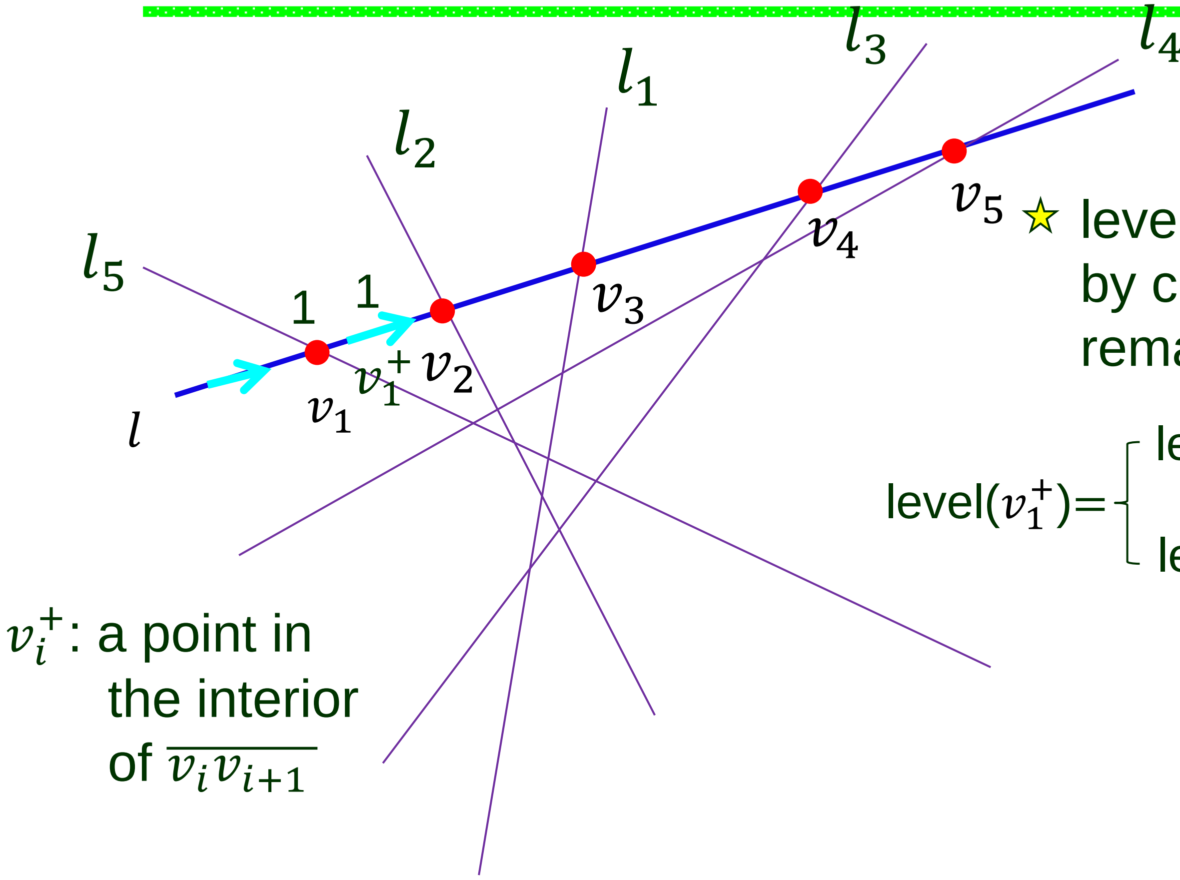


☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

Counting Levels of Vertices



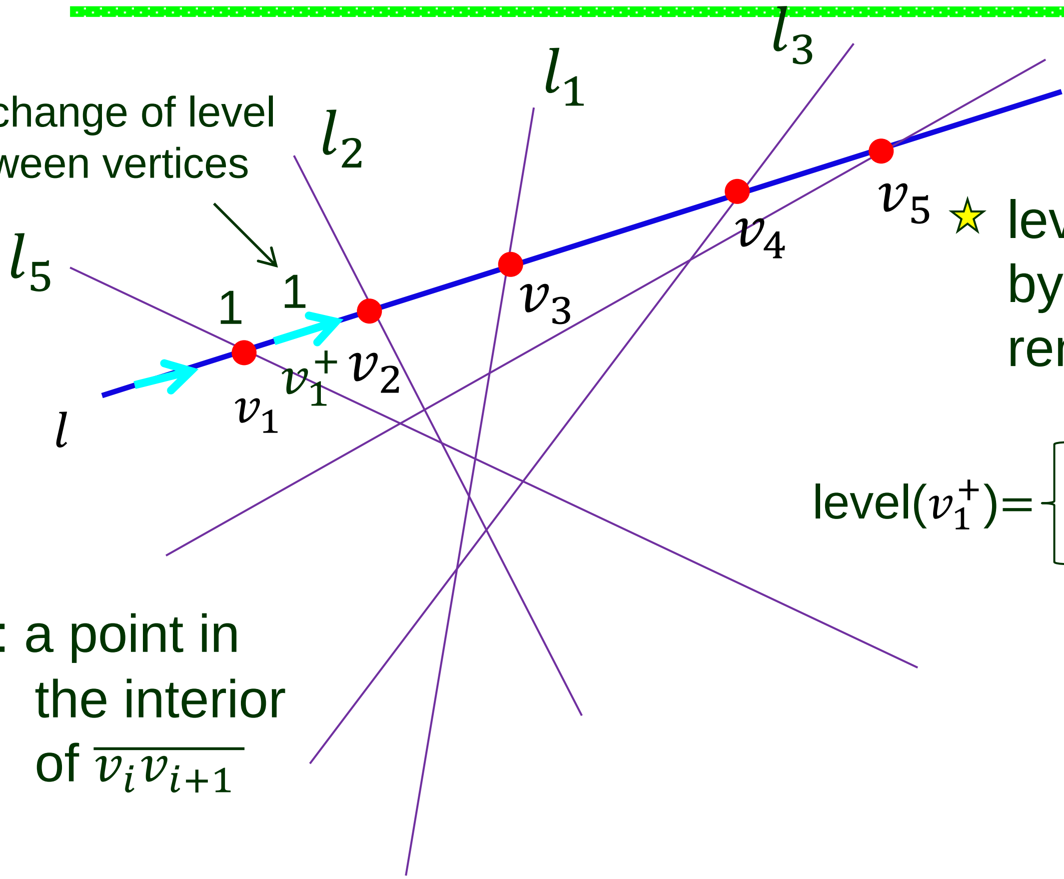
☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

Counting Levels of Vertices

no change of level between vertices

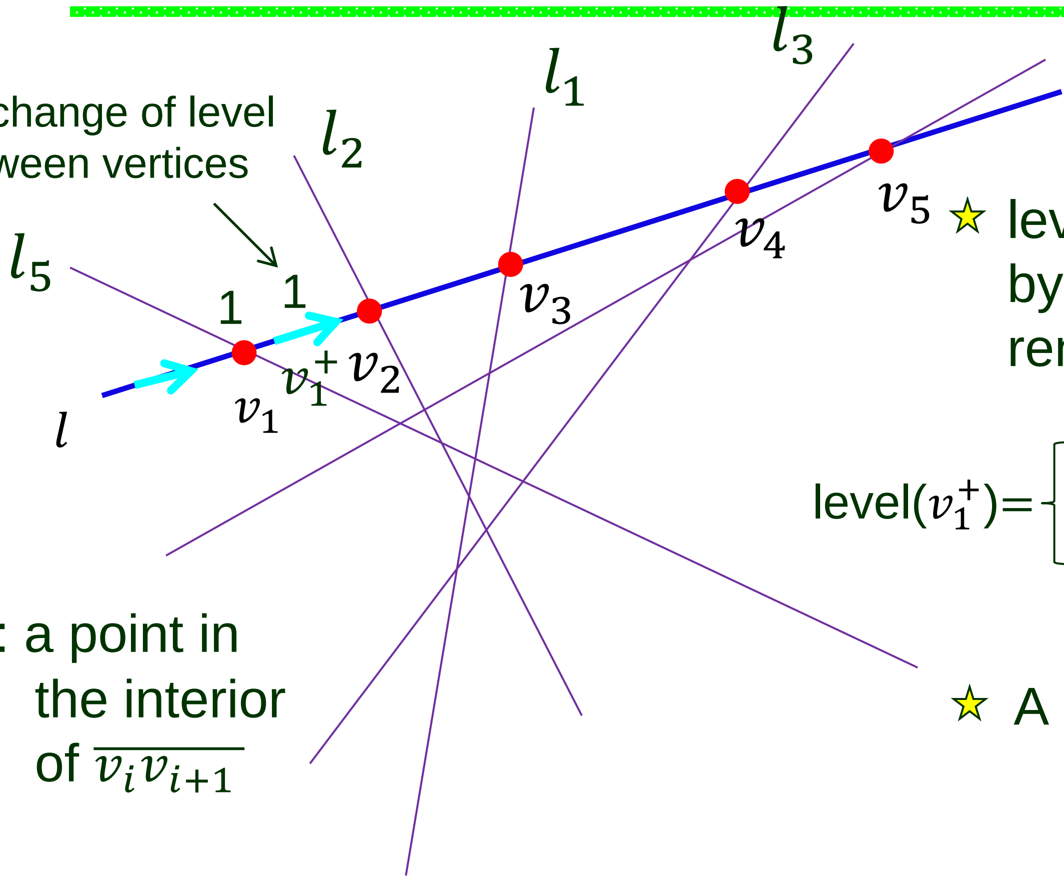


☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1) + 1 & \text{if the crossing line comes from below} \end{cases}$$

v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

Counting Levels of Vertices



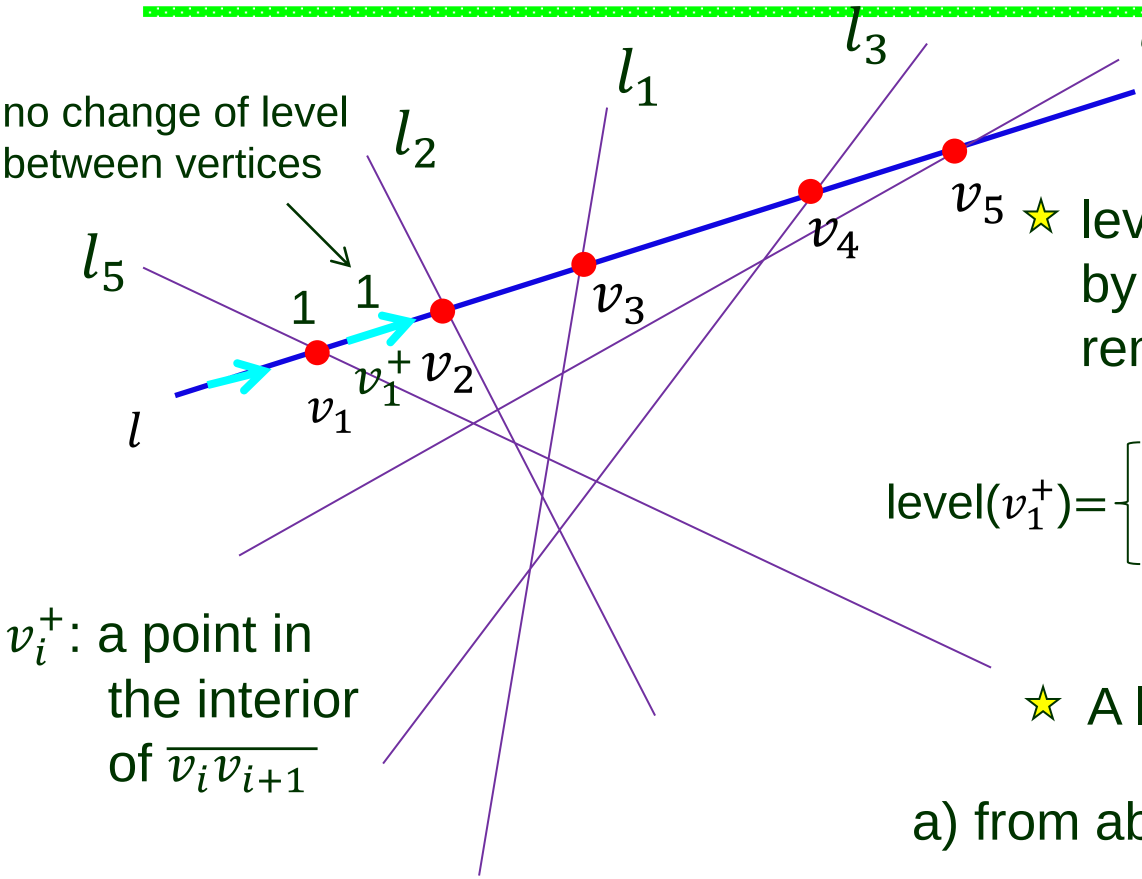
v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

★ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

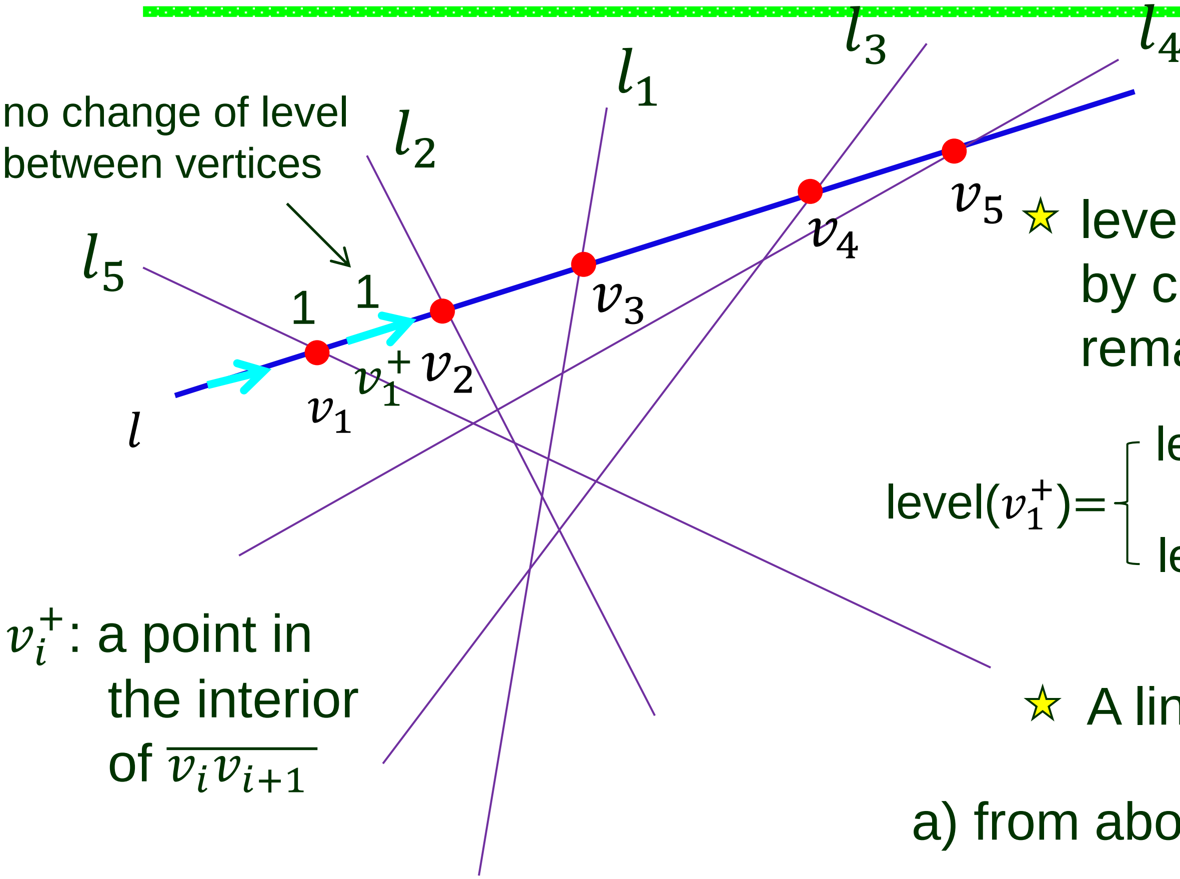
$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

★ A line crossing v_i, $i > 1$, is coming

a) from above

b) from below

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

★ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

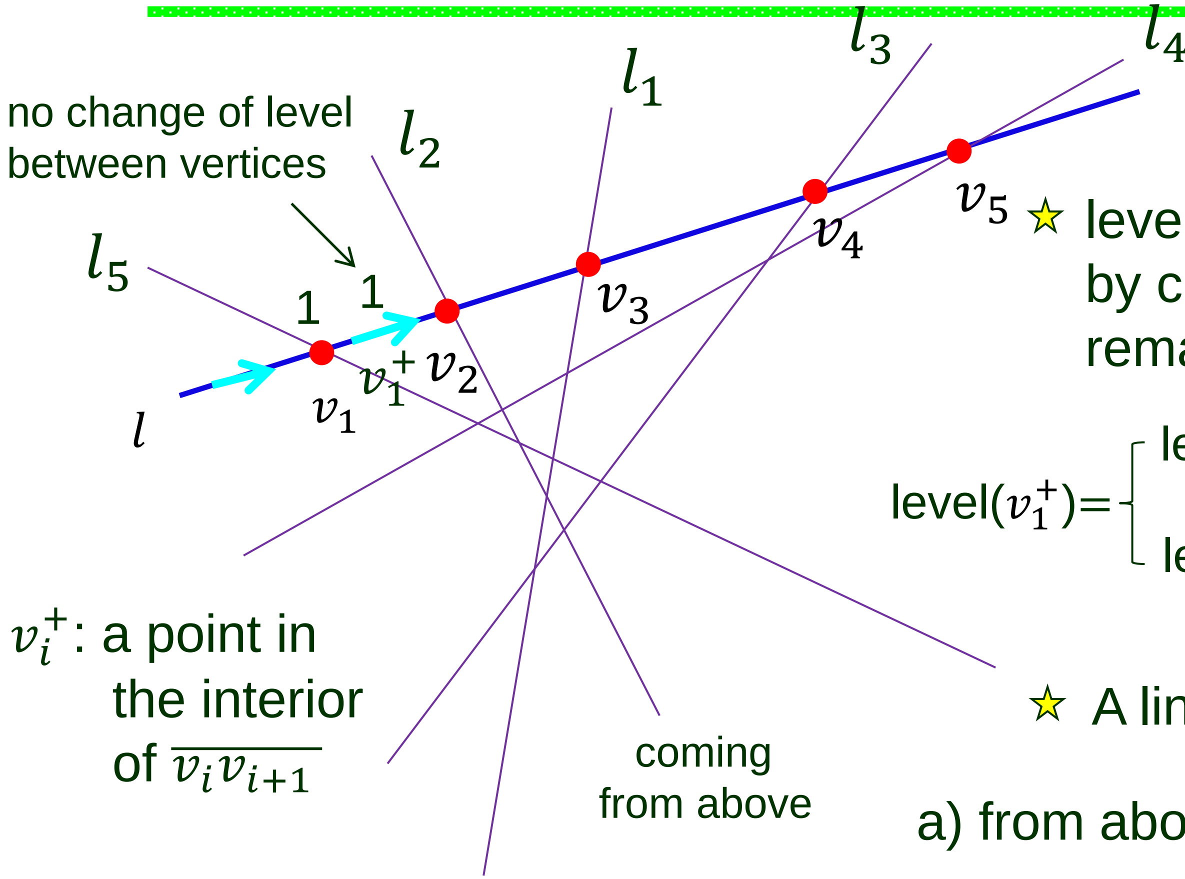
★ A line crossing v_i, $i > 1$, is coming

a) from above

$$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$$
$$\text{level}(v_i^+) = \text{level}(v_i)$$

b) from below

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

level(v_1^+)= $\begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$

☆ A line crossing v_i, $i > 1$, is coming

a) from above

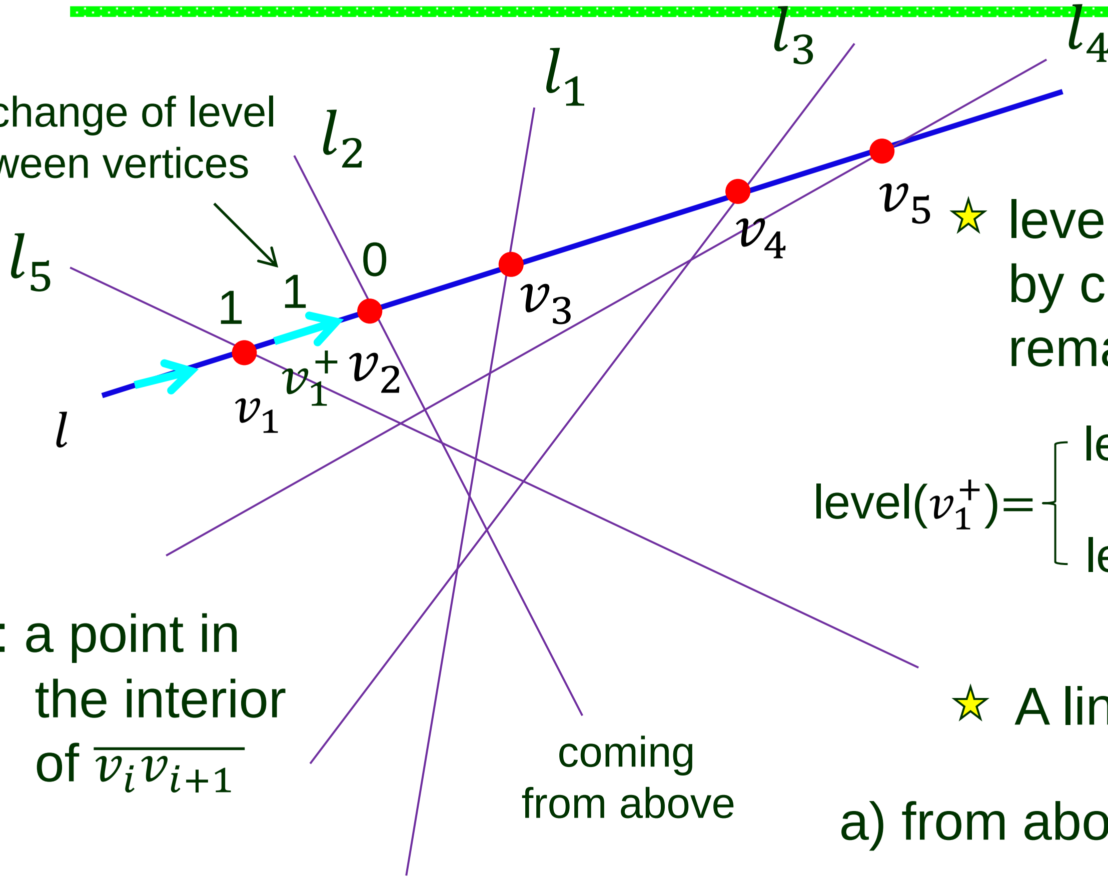
level(v_i) = level$(v_{i-1}^+) - 1$

level(v_i^+) = level(v_i)

b) from below

Counting Levels of Vertices

no change of level between vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

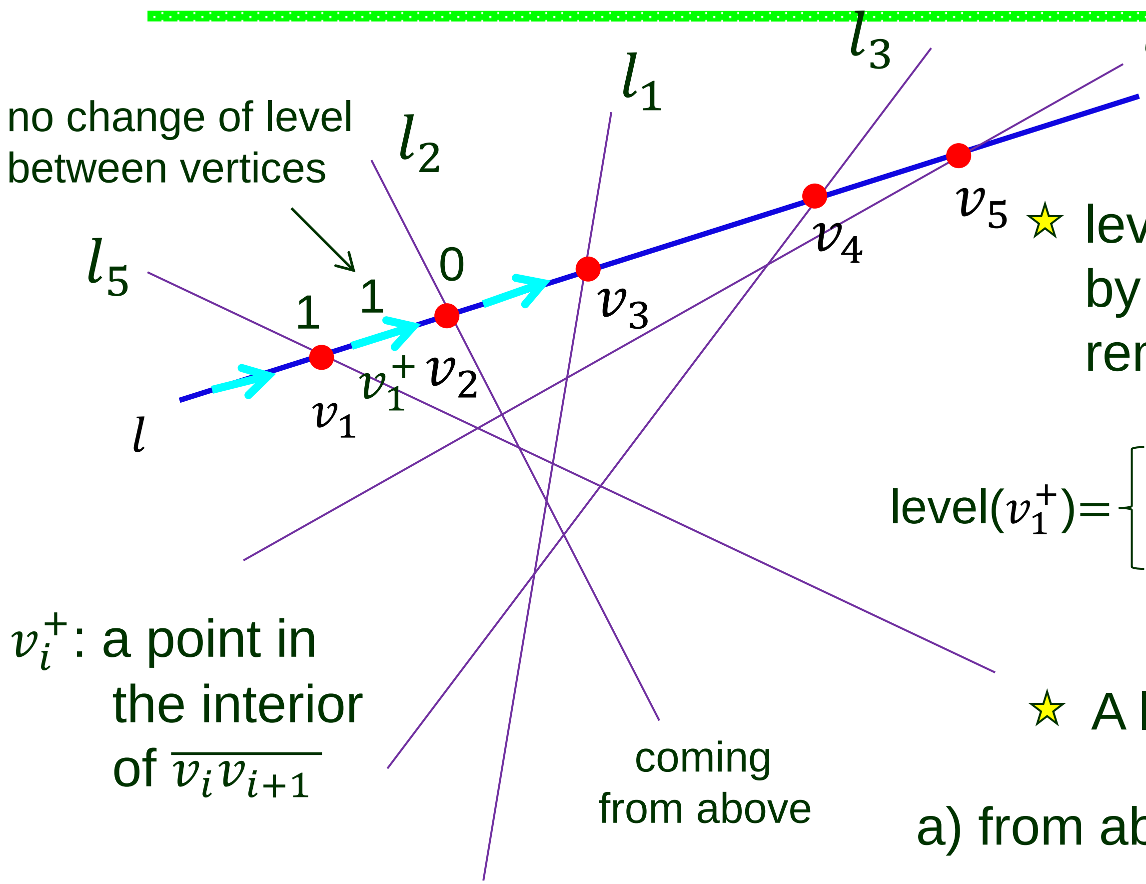
a) from above

$$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$$

$$\text{level}(v_i^+) = \text{level}(v_i)$$

b) from below

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

level(v_1^+) = $\begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$

☆ A line crossing v_i, $i > 1$, is coming

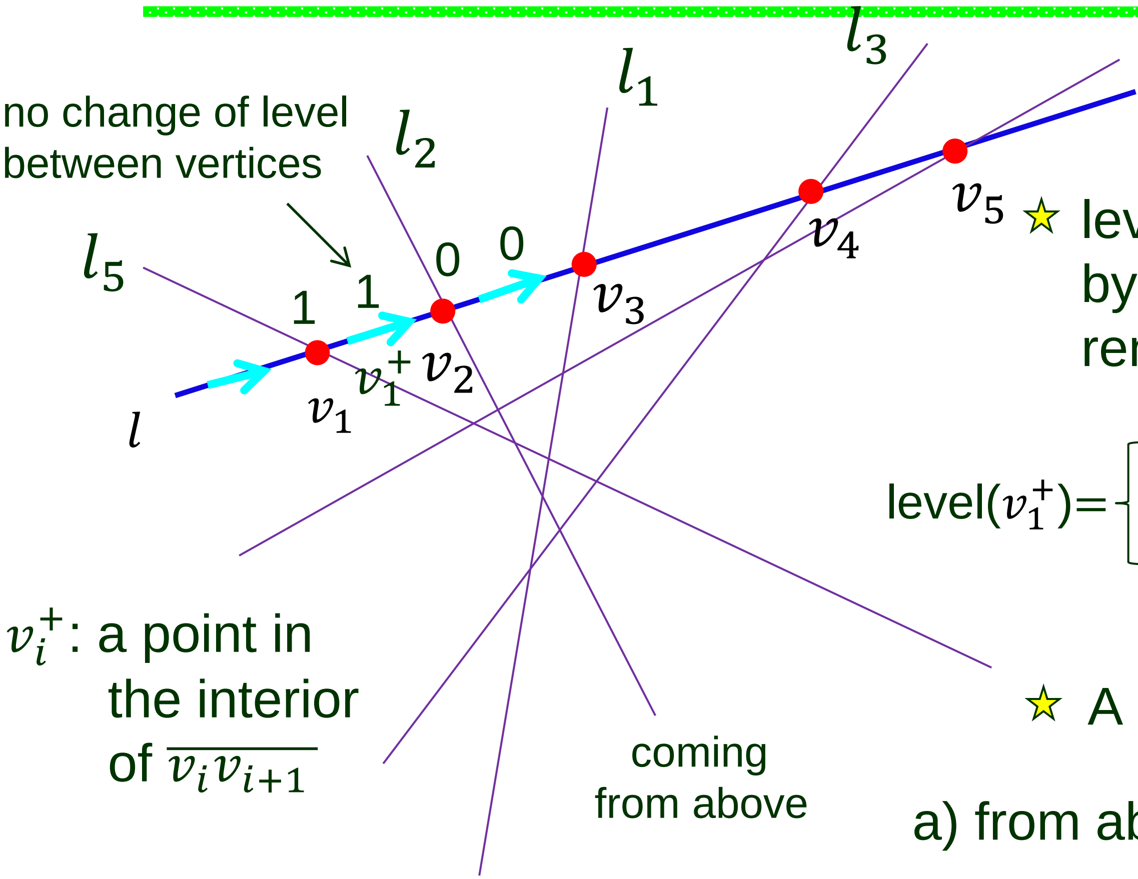
a) from above

level(v_i) = level(v_{i-1}^+) − 1

level(v_i^+) = level(v_i)

b) from below

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1) + 1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

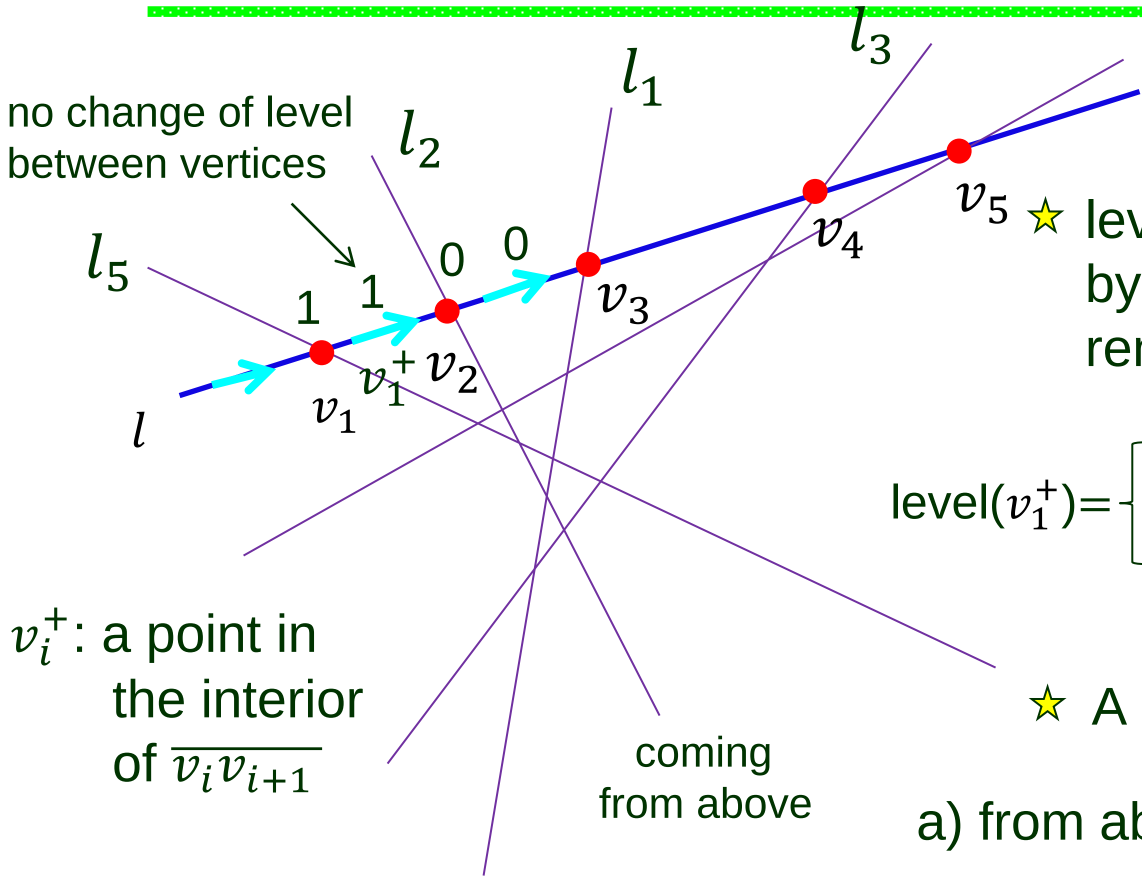
a) from above

$$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$$

$$\text{level}(v_i^+) = \text{level}(v_i)$$

b) from below

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

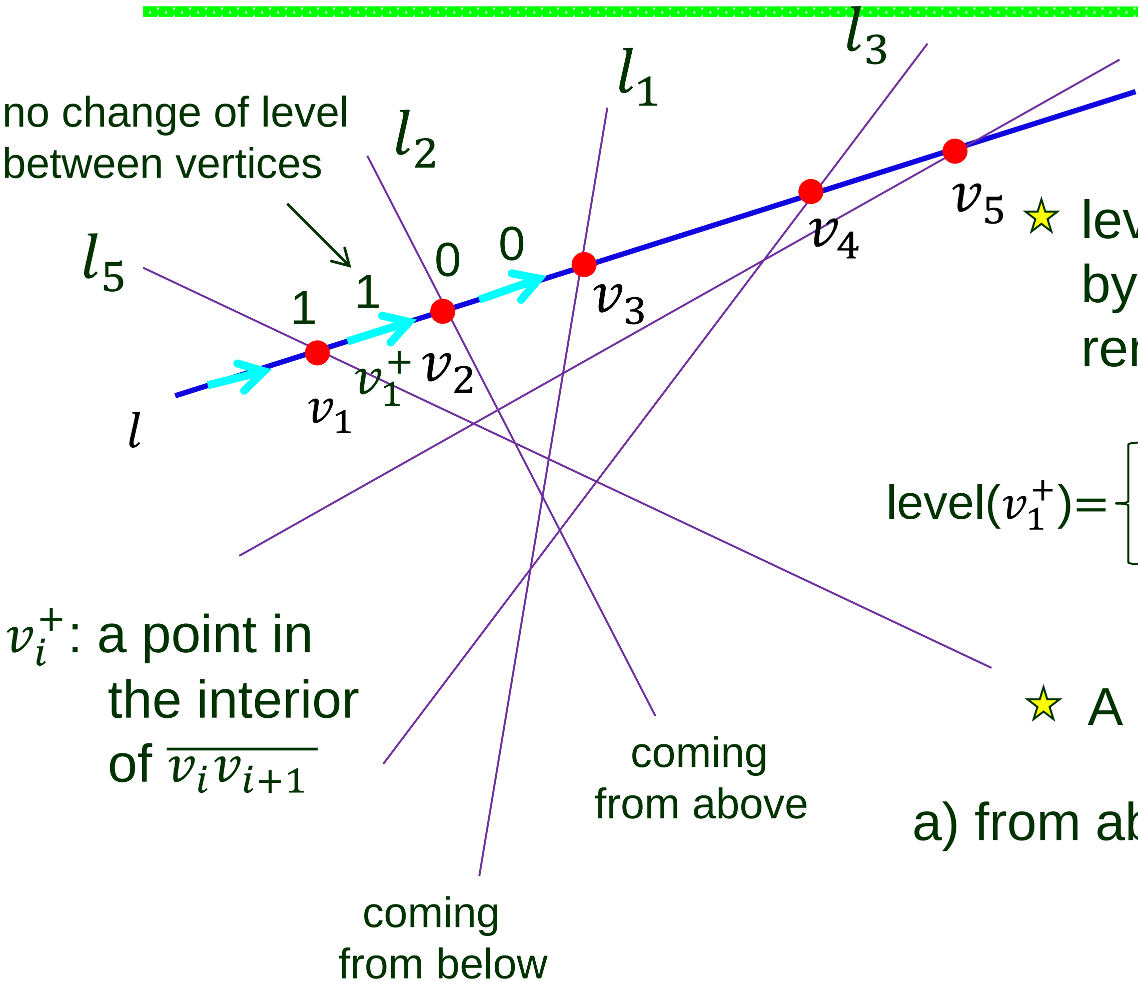
a) from above

$$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$$
$$\text{level}(v_i^+) = \text{level}(v_i)$$

b) from below

$$\text{level}(v_i) = \text{level}(v_{i-1}^+)$$
$$\text{level}(v_i^+) = \text{level}(v_i) + 1$$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

- ☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

- ☆ A line crossing v_i, $i > 1$, is coming

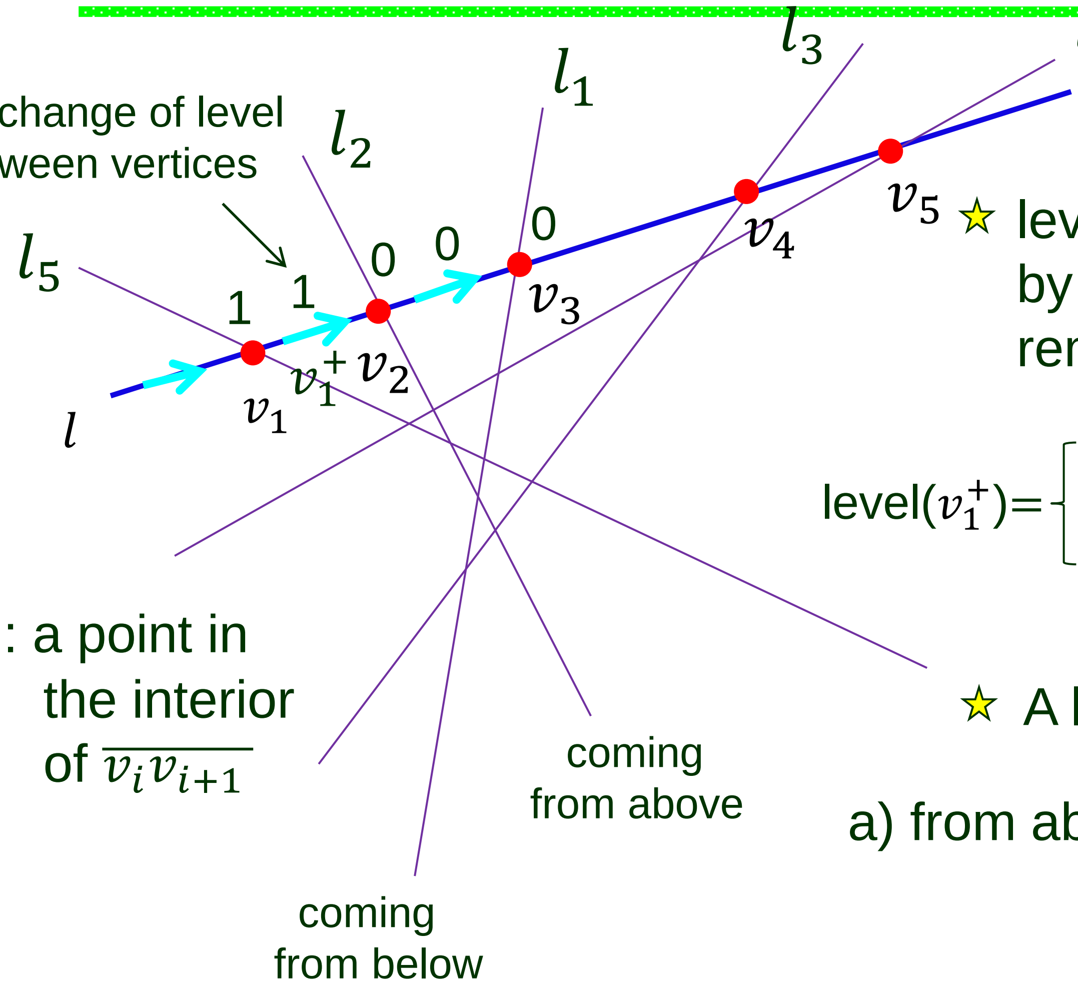
a) from above

$$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$$
$$\text{level}(v_i^+) = \text{level}(v_i)$$

b) from below

$$\text{level}(v_i) = \text{level}(v_{i-1}^+)$$
$$\text{level}(v_i^+) = \text{level}(v_i) + 1$$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

a) from above

$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$

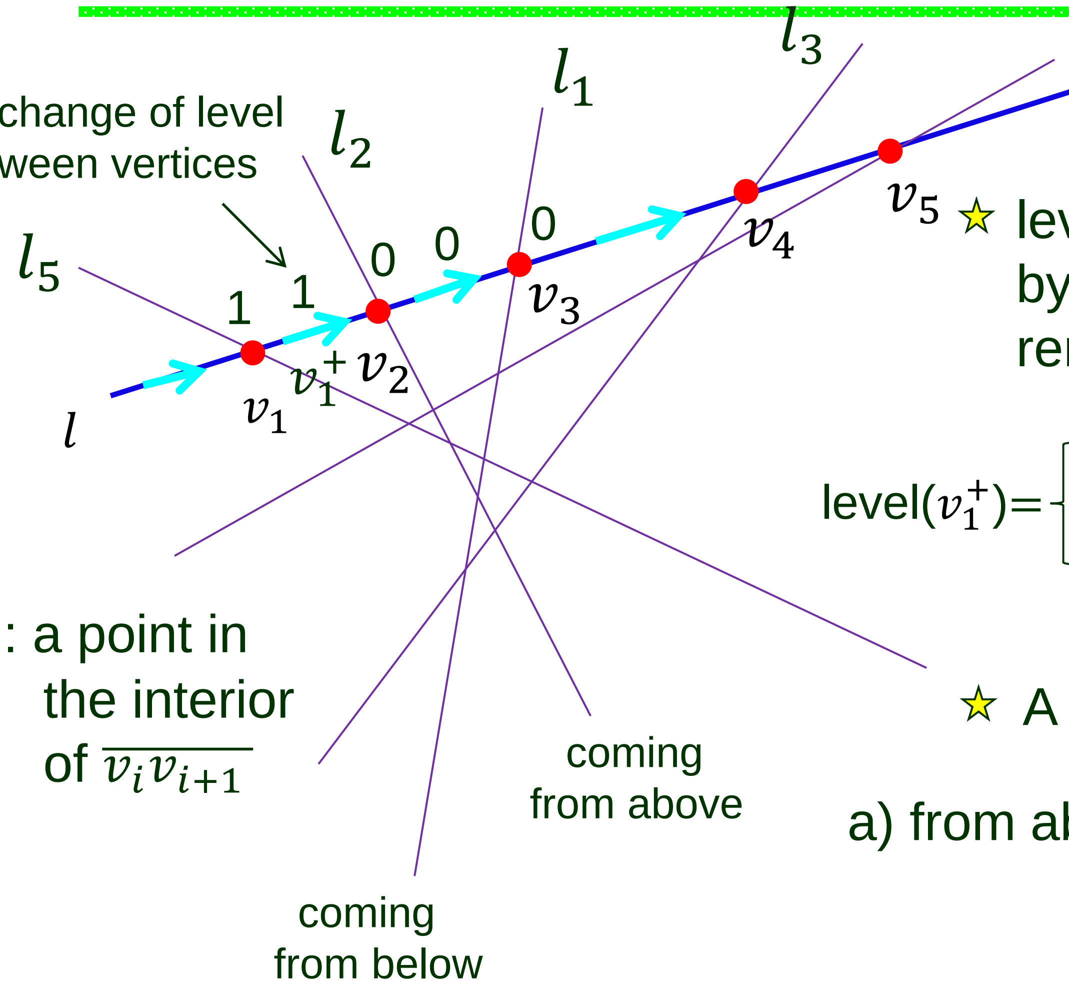
$\text{level}(v_i^+) = \text{level}(v_i)$

b) from below

$\text{level}(v_i) = \text{level}(v_{i-1}^+)$

$\text{level}(v_i^+) = \text{level}(v_i) + 1$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

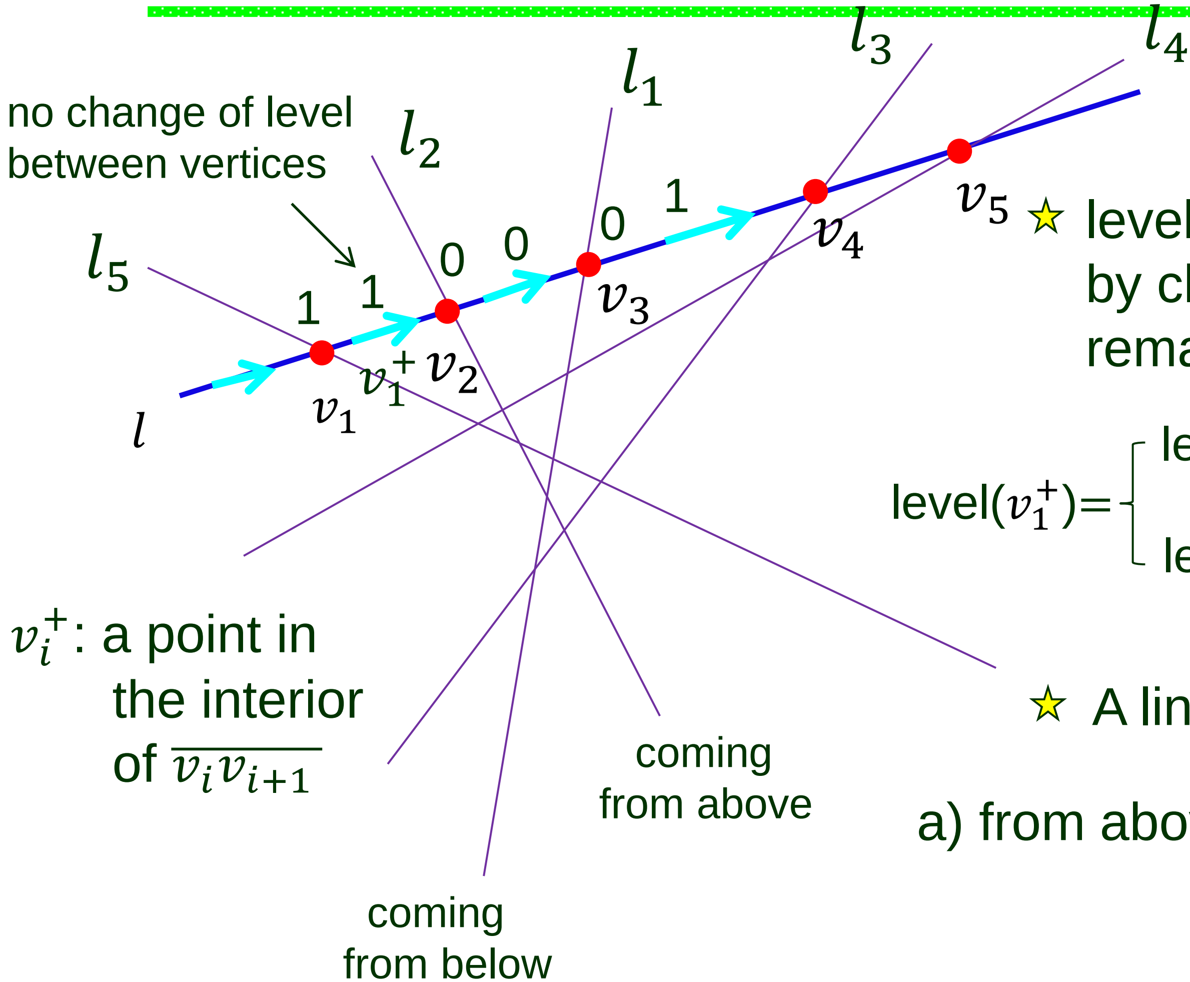
a) from above

$$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$$
$$\text{level}(v_i^+) = \text{level}(v_i)$$

b) from below

$$\text{level}(v_i) = \text{level}(v_{i-1}^+)$$
$$\text{level}(v_i^+) = \text{level}(v_i) + 1$$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

level(v_1^+)= $\begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$

☆ A line crossing v_i, $i > 1$, is coming

a) from above

level(v_i) = level$(v_{i-1}^+) - 1$

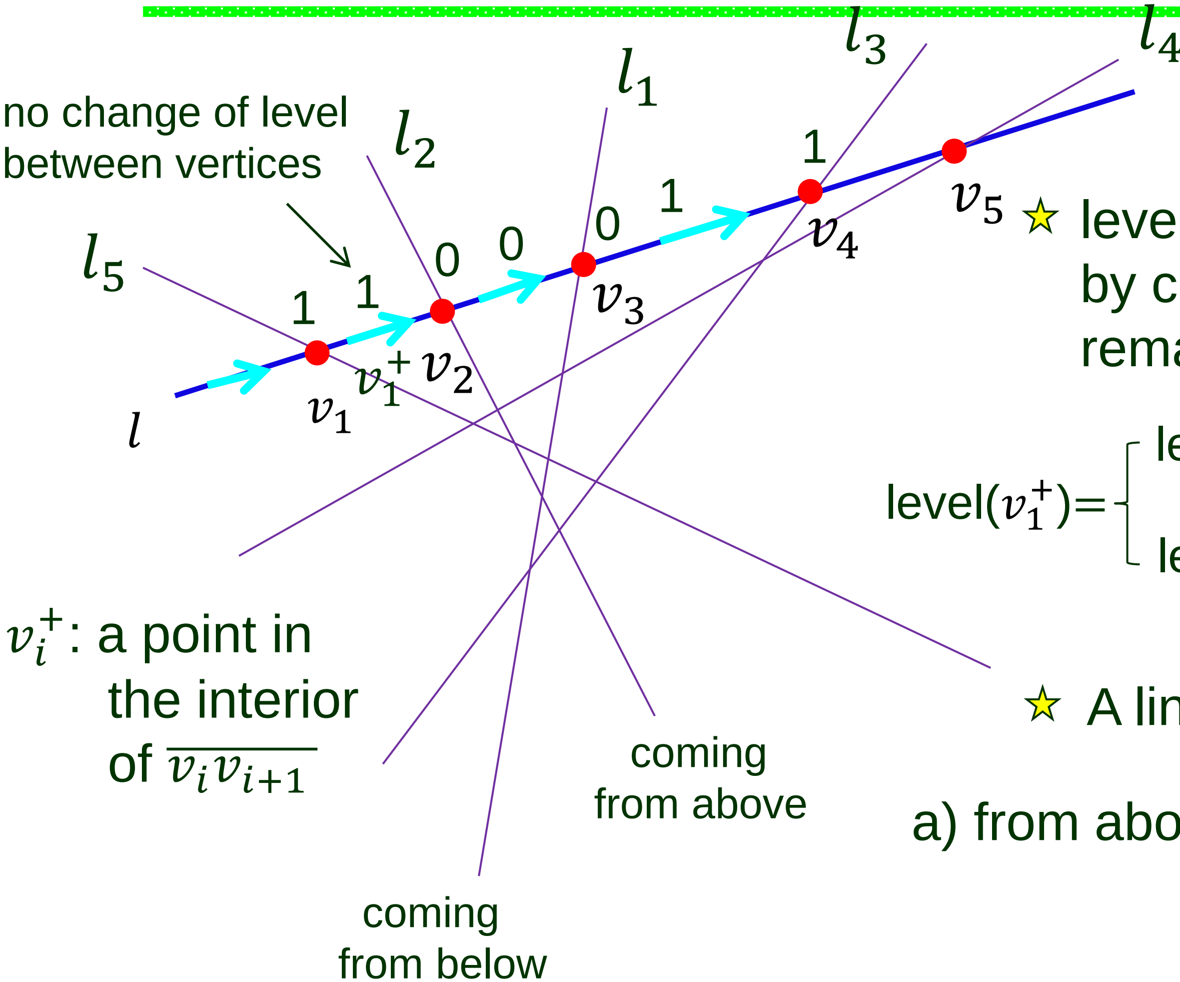
level(v_i^+) = level(v_i)

b) from below

level(v_i) = level(v_{i-1}^+)

level(v_i^+) = level$(v_i) + 1$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

a) from above

$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$

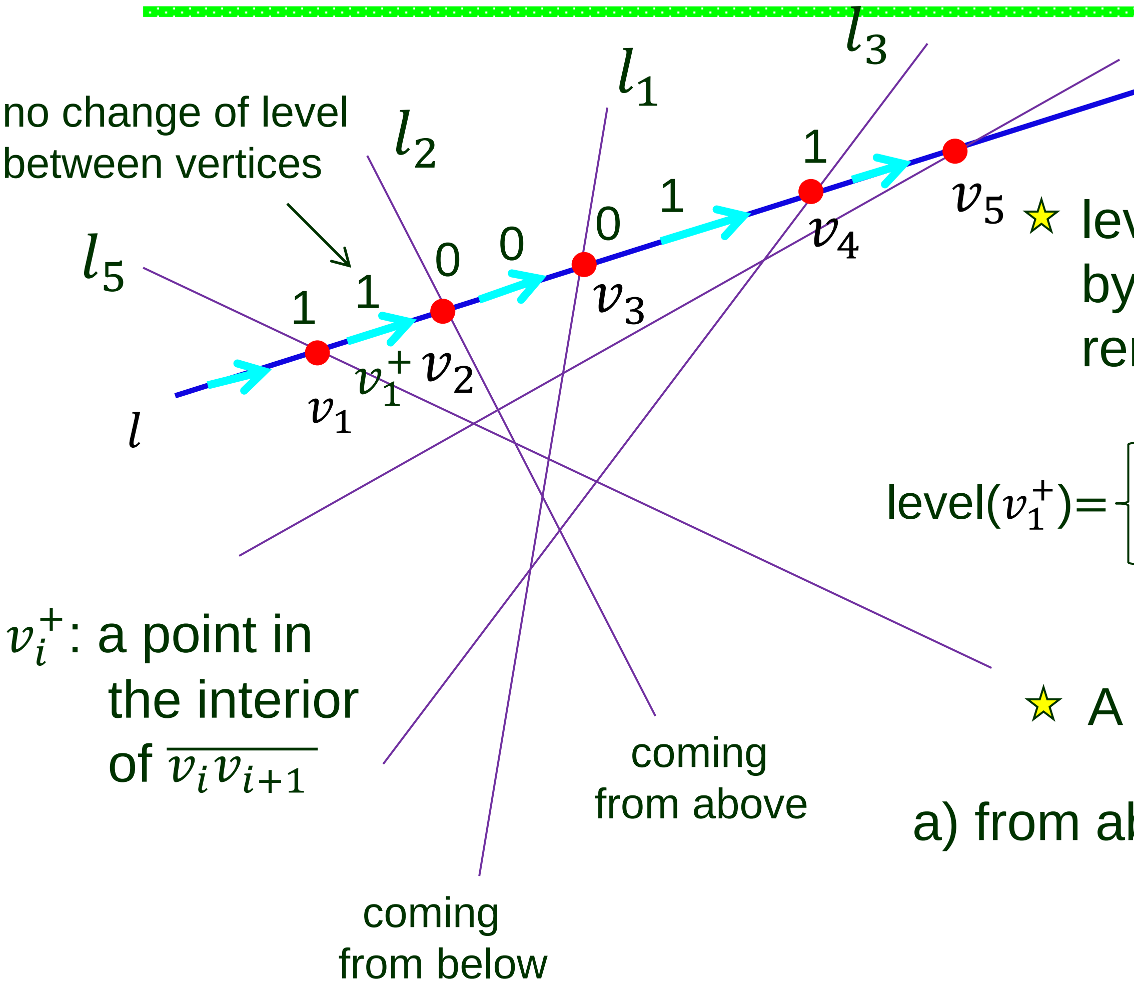
$\text{level}(v_i^+) = \text{level}(v_i)$

b) from below

$\text{level}(v_i) = \text{level}(v_{i-1}^+)$

$\text{level}(v_i^+) = \text{level}(v_i) + 1$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1) + 1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

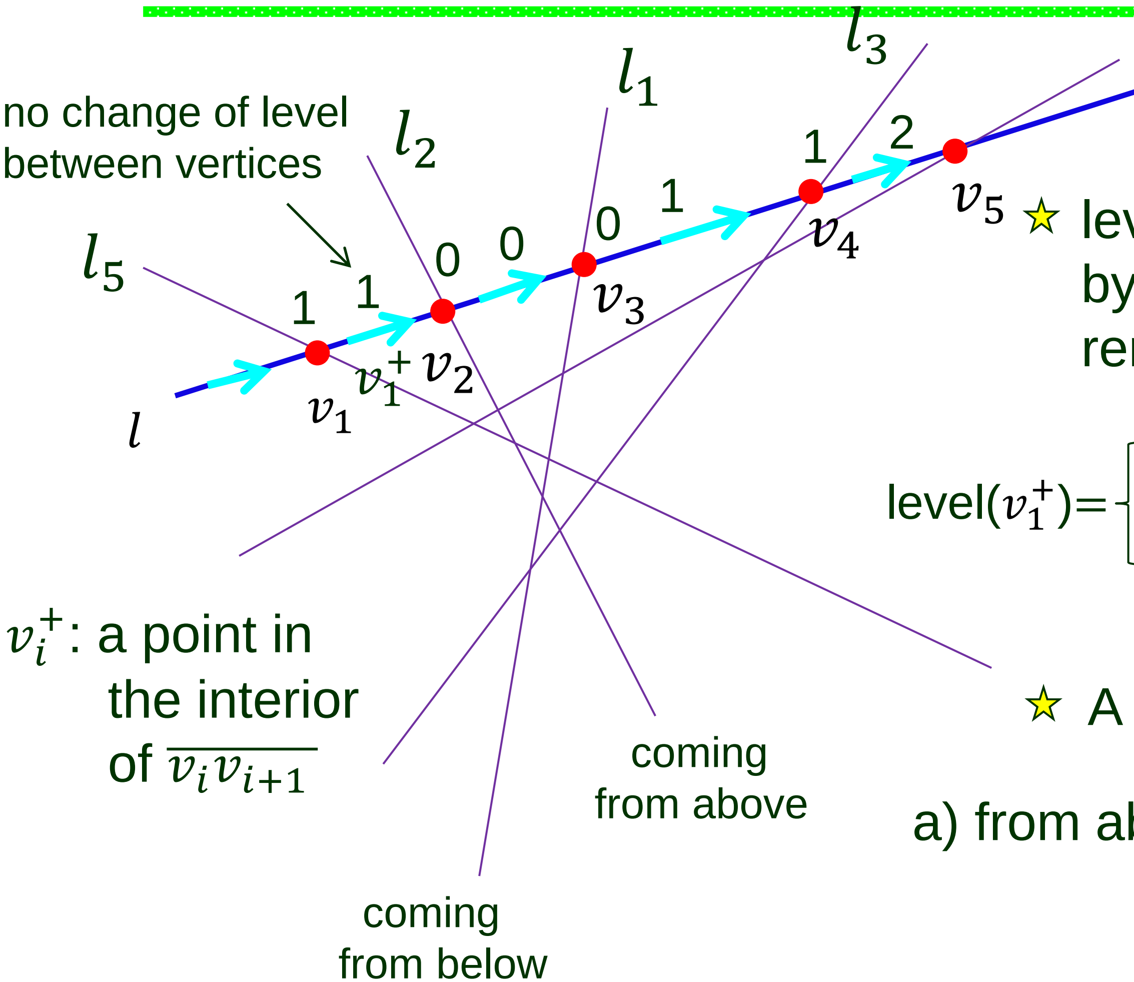
a) from above

$$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$$
$$\text{level}(v_i^+) = \text{level}(v_i)$$

b) from below

$$\text{level}(v_i) = \text{level}(v_{i-1}^+)$$
$$\text{level}(v_i^+) = \text{level}(v_i) + 1$$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1) + 1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

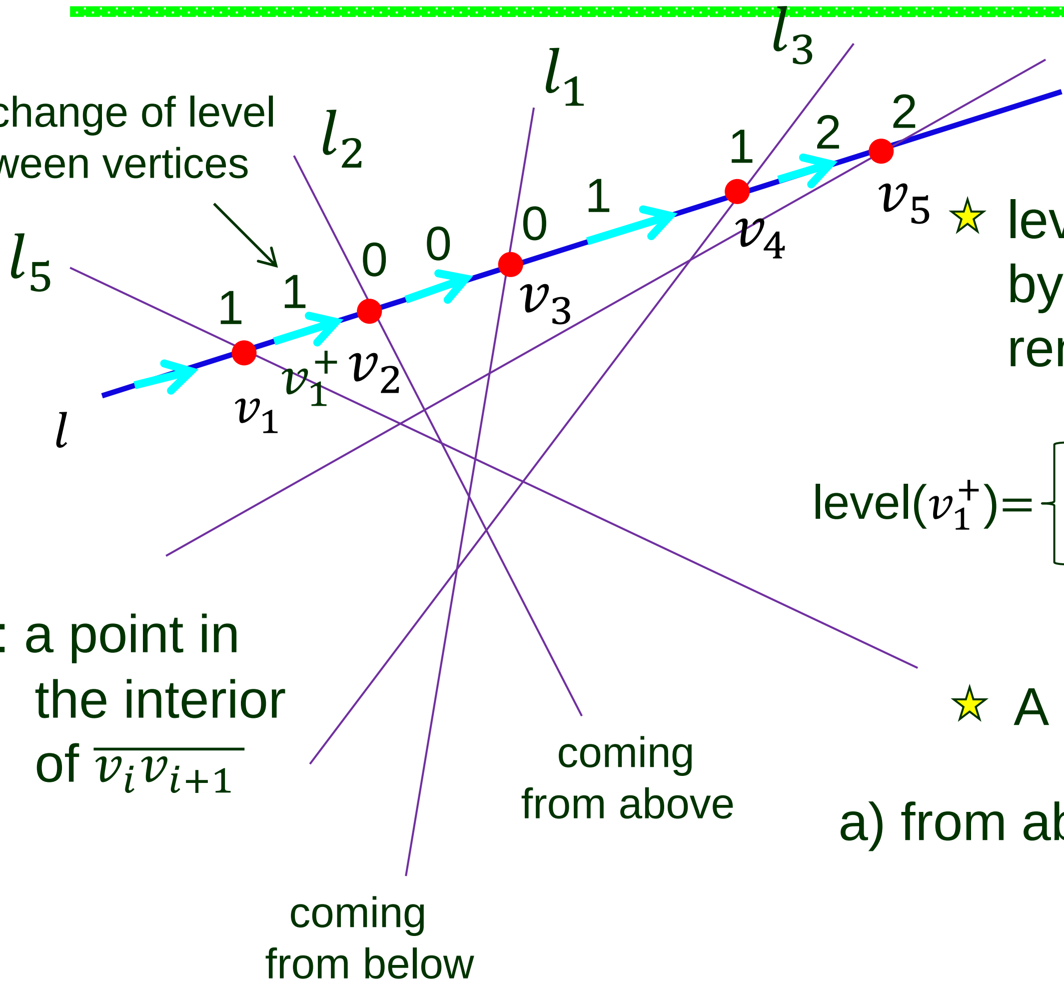
a) from above

$$\text{level}(v_i) = \text{level}(v_{i-1}^+) - 1$$
$$\text{level}(v_i^+) = \text{level}(v_i)$$

b) from below

$$\text{level}(v_i) = \text{level}(v_{i-1}^+)$$
$$\text{level}(v_i^+) = \text{level}(v_i) + 1$$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

level(v_1^+)= $\begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$

☆ A line crossing v_i, $i > 1$, is coming

a) from above

level(v_i) = level$(v_{i-1}^+) - 1$

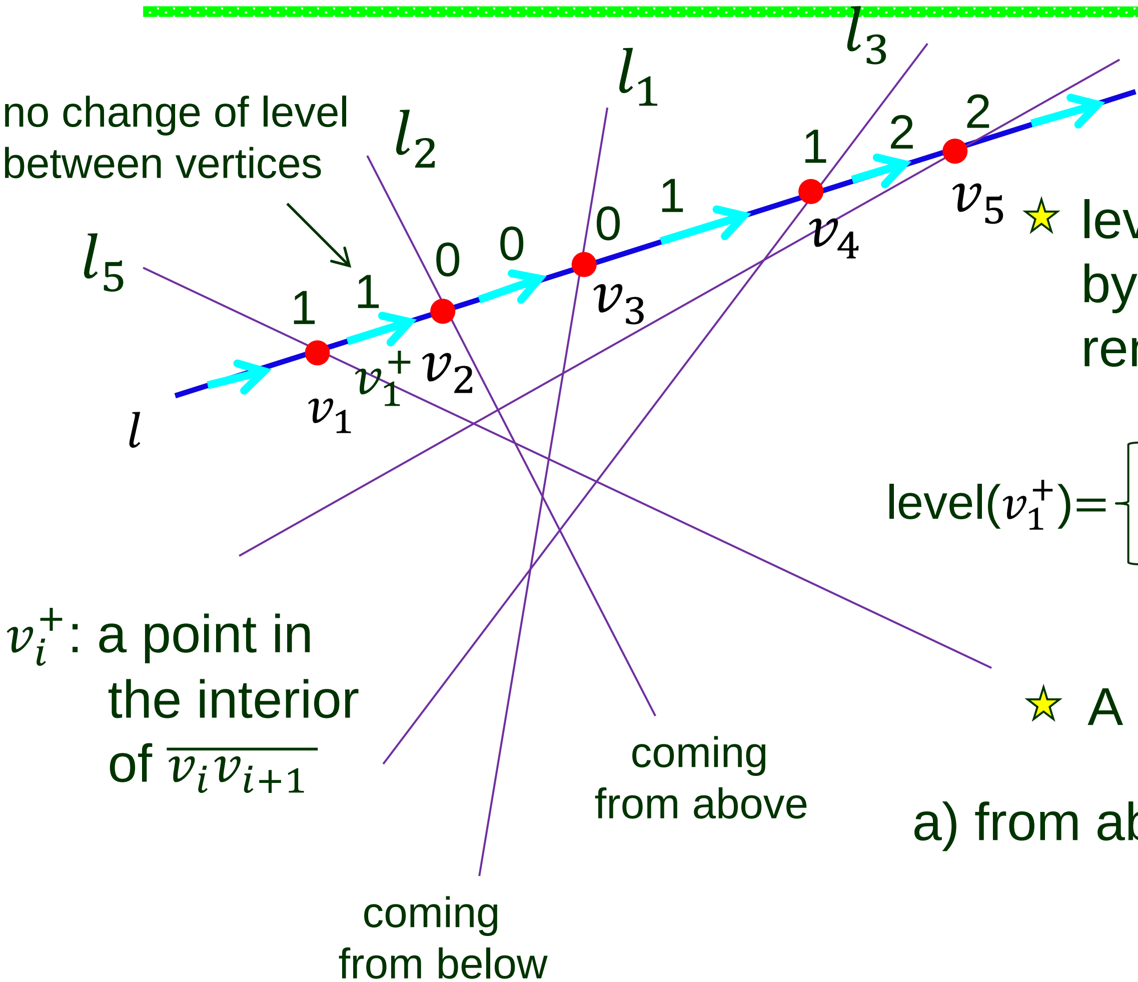
level(v_i^+) = level(v_i)

b) from below

level(v_i) = level(v_{i-1}^+)

level(v_i^+) = level$(v_i) + 1$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

★ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

★ A line crossing v_i, $i > 1$, is coming

a) from above

level(v_i) = level$(v_{i-1}^+) - 1$

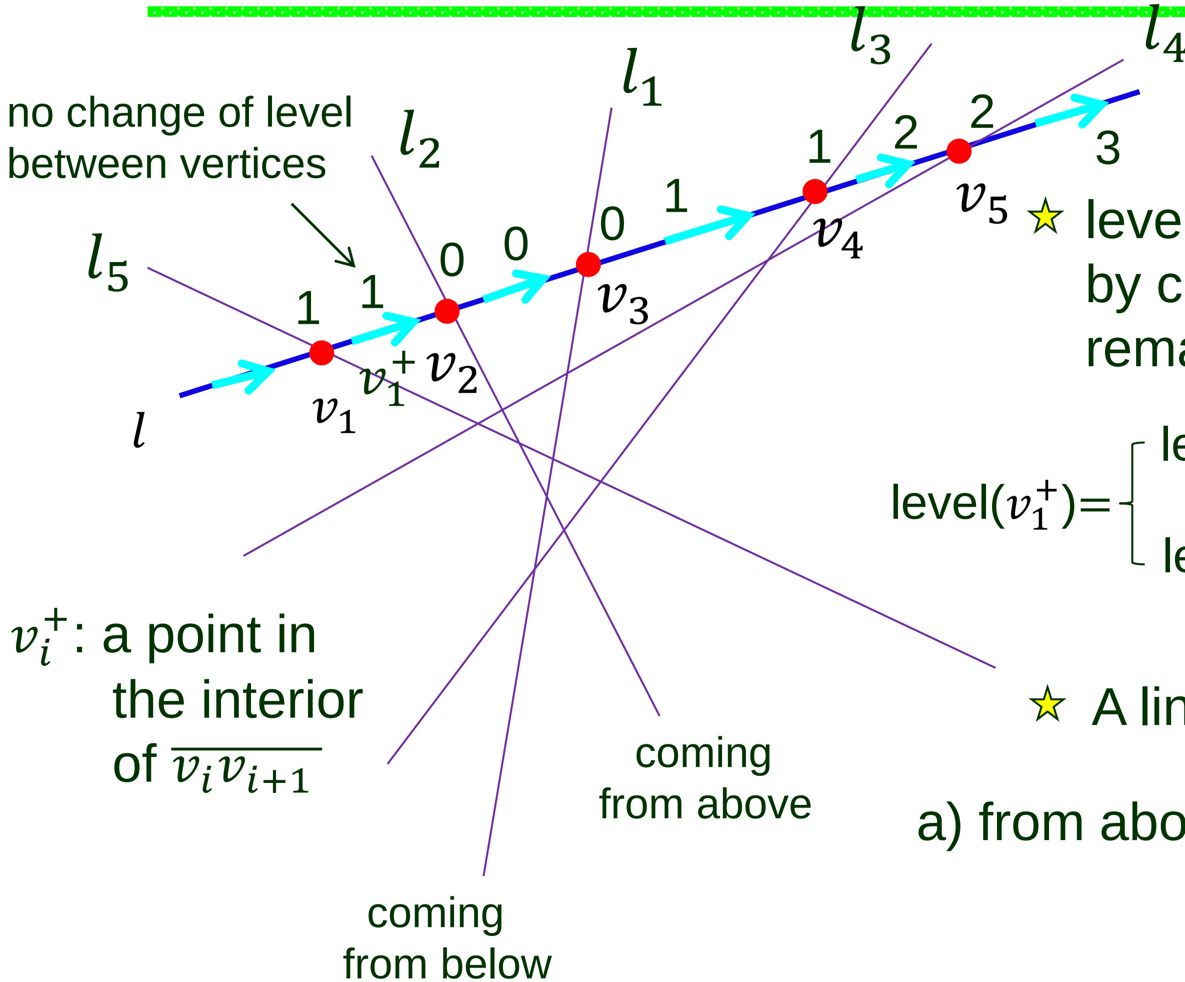
level(v_i^+) = level(v_i)

b) from below

level(v_i) = level(v_{i-1}^+)

level(v_i^+) = level$(v_i) + 1$

Counting Levels of Vertices



v_i^+: a point in the interior of $\overline{v_i v_{i+1}}$

☆ level(v_1) determined in $O(n)$ time by checking how many of the $n-1$ remaining lines are above v_1.

$$\text{level}(v_1^+) = \begin{cases} \text{level}(v_1) & \text{if the line crossing } v_1 \text{ comes from above} \\ \text{level}(v_1)+1 & \text{if the crossing line comes from below} \end{cases}$$

☆ A line crossing v_i, $i > 1$, is coming

a) from above

level(v_i) = level(v_{i-1}^+) − 1

level(v_i^+) = level(v_i)

b) from below

level(v_i) = level(v_{i-1}^+)

level(v_i^+) = level(v_i) + 1

Running Times

Levels of vertices along a line computable in $O(n)$ time.

Levels of all vertices in a line arrangement can be computed in $O(n^2)$ time.

Discrete Measures & Degeneracy

Discrete measures of all type ii) half-planes are computable in $O(n^2)$ time if no two points are on the same vertical line.

Discrete Measures & Degeneracy

Discrete measures of all type ii) half-planes are computable in $O(n^2)$ time if no two points are on the same vertical line.

What if two or more points are on the same vertical line l?

Discrete Measures & Degeneracy

Discrete measures of all type ii) half-planes are computable in $O(n^2)$ time if no two points are on the same vertical line.

What if two or more points are on the same vertical line l?

- ♣ l^* *undefined* and thus does *not* show up as an intersection in the dual plane.

Discrete Measures & Degeneracy

Discrete measures of all type ii) half-planes are computable in $O(n^2)$ time if no two points are on the same vertical line.

What if two or more points are on the same vertical line l?

- ♣ l^* *undefined* and thus does *not* show up as an intersection in the dual plane.

- ♦ For every vertical line through ≥ 2 points, determine the discrete measure of the corresponding half-plane.

Discrete Measures & Degeneracy

Discrete measures of all type ii) half-planes are computable in $O(n^2)$ time if no two points are on the same vertical line.

What if two or more points are on the same vertical line l?

- ♣ l^* *undefined* and thus does *not* show up as an intersection in the dual plane.

- ♦ For every vertical line through ≥ 2 points, determine the discrete measure of the corresponding half-plane.
- ♦ Only $O(n)$ such vertical lines.

Discrete Measures & Degeneracy

Discrete measures of all type ii) half-planes are computable in $O(n^2)$ time if no two points are on the same vertical line.

What if two or more points are on the same vertical line l?

- ♣ l^* *undefined* and thus does *not* show up as an intersection in the dual plane.

- ♦ For every vertical line through ≥ 2 points, determine the discrete measure of the corresponding half-plane.
- ♦ Only $O(n)$ such vertical lines.

$\Downarrow$

Their discrete measures can be computed in $O(n^2)$ time.

Summary

Maximum discrepancy due to half-planes:

$$\max_h \Delta_S(h) = |\mu(h) - \mu_S(h)|$$

$$\Downarrow$$

The boundary line of the maximizing h must pass either i) one point or ii) ≥ 2 points.

Summary

Maximum discrepancy due to half-planes:

$$\max_h \Delta_S(h) = |\mu(h) - \mu_S(h)|$$

$$\Downarrow$$

The boundary line of the maximizing h must pass either i) one point or ii) ≥ 2 points.

- Discrepancies of the n type i) candidates for h can be computed in $O(n^2)$ time by using calculus to find extrema of the continuous measure $\mu(h)$. (Discrete measure each takes time $O(n)$.)

Summary

Maximum discrepancy due to half-planes:

$$\max_h \Delta_S(h) = |\mu(h) - \mu_S(h)|$$

$$\Downarrow$$

The boundary line of the maximizing h must pass either i) one point or ii) ≥ 2 points.

- Discrepancies of the n type i) candidates for h can be computed in $O(n^2)$ time by using calculus to find extrema of the continuous measure $\mu(h)$. (Discrete measure each takes time $O(n)$.)
- For $O(n^2)$ type ii) candidates, effort is on the discrete measure $\mu_S(h)$.
  - Deal with all vertical lines passing through ≥ 2 points ($O(n^2)$).
  - Use duality to compute $\mu_S(h)$ for all the non-vertical lines through ≥ 2 points ($O(n^2)$).

Summary

Maximum discrepancy due to half-planes:

$$\max_h \Delta_S(h) = |\mu(h) - \mu_S(h)|$$

$$\Downarrow$$

The boundary line of the maximizing h must pass either i) one point or ii) ≥ 2 points.

- Discrepancies of the n type i) candidates for h can be computed in $O(n^2)$ time by using calculus to find extrema of the continuous measure $\mu(h)$. (Discrete measure each takes time $O(n)$.)
- For $O(n^2)$ type ii) candidates, effort is on the discrete measure $\mu_S(h)$.
  - Deal with all vertical lines passing through ≥ 2 points ($O(n^2)$).
  - Use duality to compute $\mu_S(h)$ for all the non-vertical lines through ≥ 2 points ($O(n^2)$).

Total time: $O(n^2)$