

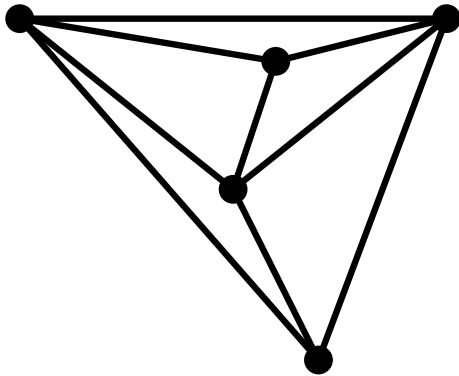
# Compact Drawings of 1-Planar Graphs with Right-Angle Crossings and Few Bends

Steven Chaplick, Fabian Lipp,  
Alexander Wolff, and **Johannes Zink**

# Introduction: Beyond-Planar Graphs

2

Types of drawings:



**Planar:**

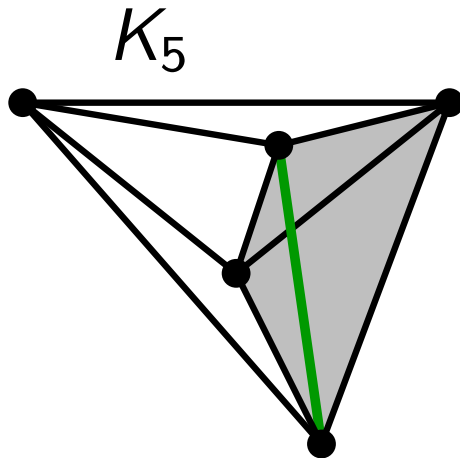
No crossings

# Introduction: Beyond-Planar Graphs

2

## Types of drawings:

**1-Planar:**  $\leq 1$  crossings per edge



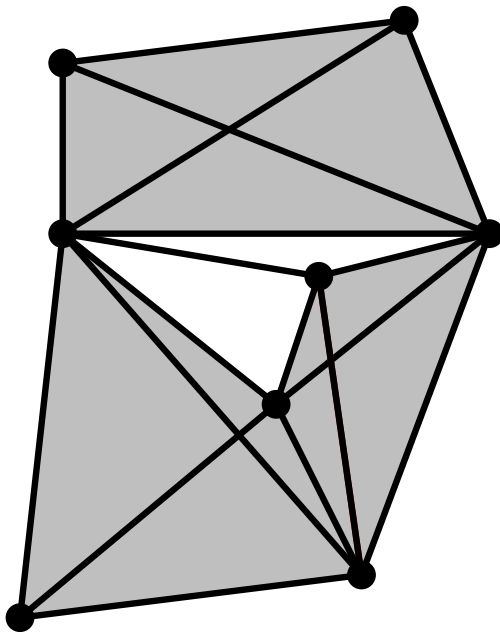
**Planar:** No crossings

# Introduction: Beyond-Planar Graphs

2

## Types of drawings:

**1-Planar:**  $\leq 1$  crossings per edge



**Planar:** No crossings

# Introduction: Beyond-Planar Graphs

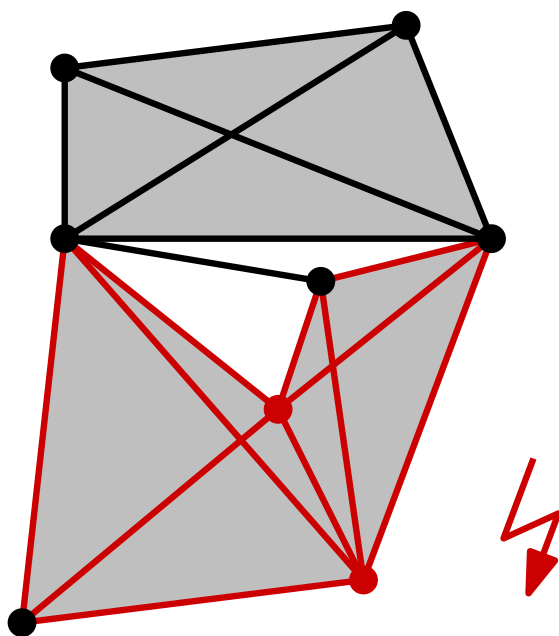
2

## Types of drawings:

**1-Planar:**  $\leq 1$  crossings per edge

**NIC-Planar:** Two crossings share  $\leq 1$  vertices

**Planar:** No crossings



# Introduction: Beyond-Planar Graphs

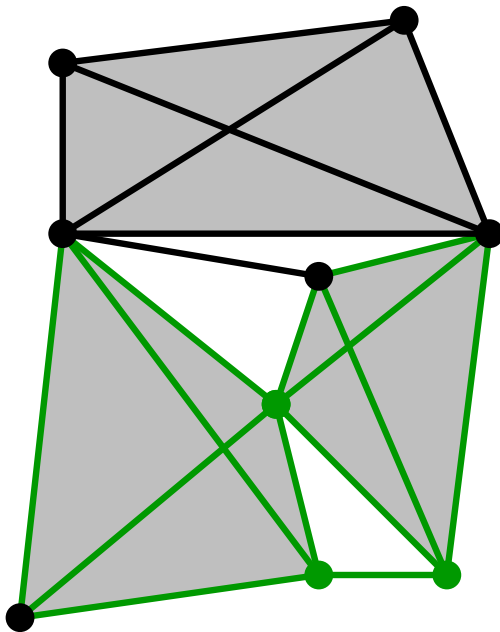
2

## Types of drawings:

**1-Planar:**  $\leq 1$  crossings per edge

**NIC-Planar:** Two crossings share  $\leq 1$  vertices

**Planar:** No crossings



# Introduction: Beyond-Planar Graphs

2

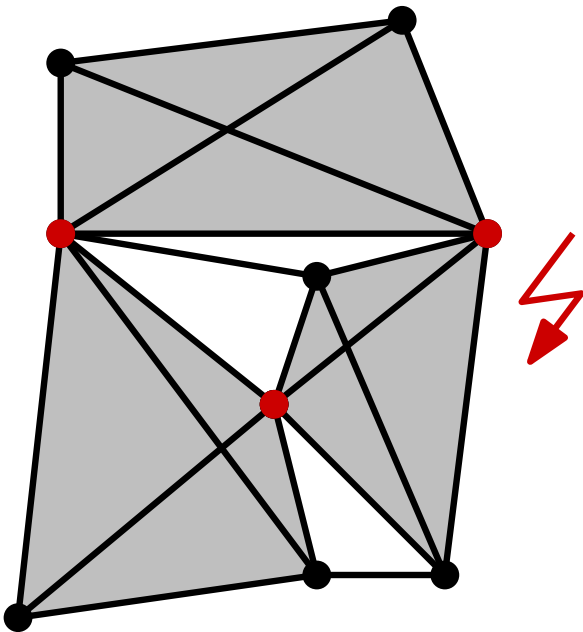
## Types of drawings:

**1-Planar:**  $\leq 1$  crossings per edge

**NIC-Planar:** Two crossings share  $\leq 1$  vertices

**IC-Planar:** Two crossings share no vertices

**Planar:** No crossings



# Introduction: Beyond-Planar Graphs

2

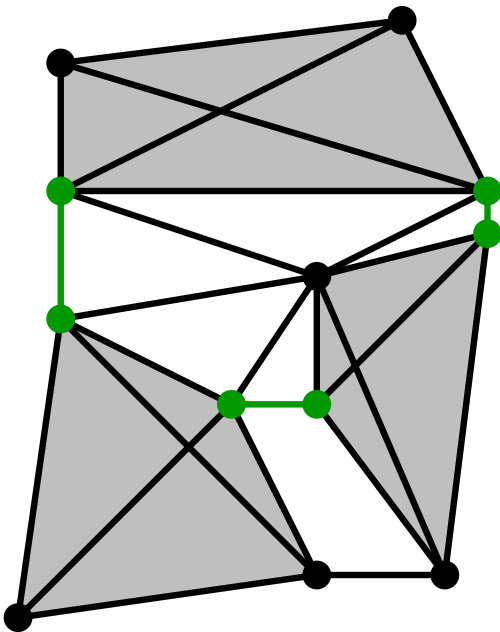
## Types of drawings:

**1-Planar:**  $\leq 1$  crossings per edge

**NIC-Planar:** Two crossings share  $\leq 1$  vertices

**IC-Planar:** Two crossings share no vertices

**Planar:** No crossings





# Introduction: Beyond-Planar Graphs

2

## Types of drawings:

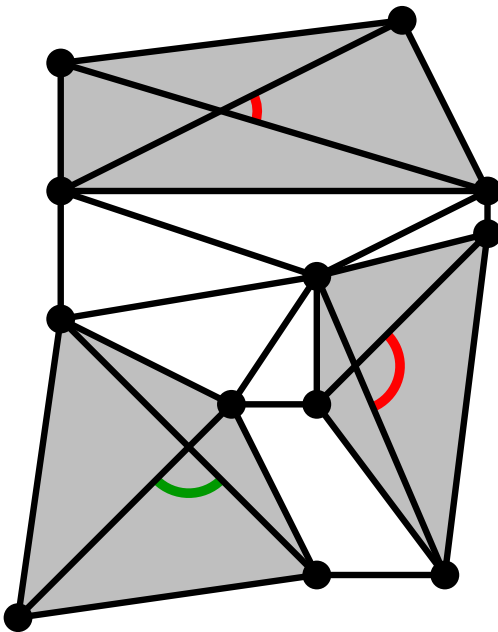
**1-Planar:**  $\leq 1$  crossings per edge

**NIC-Planar:** Two crossings share  $\leq 1$  vertices

**IC-Planar:** Two crossings share no vertices

**Planar:** No crossings

**RAC:** Right angle crossings



# Introduction: Beyond-Planar Graphs

2

## Types of drawings:

**1-Planar:**  $\leq 1$  crossings per edge

**NIC-Planar:** Two crossings share  $\leq 1$  vertices

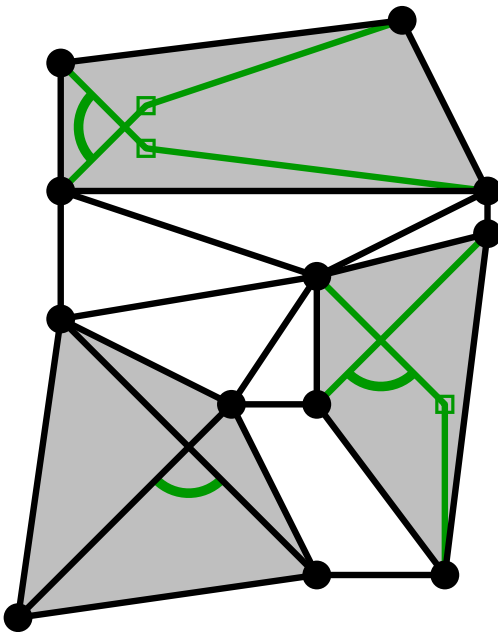
**IC-Planar:** Two crossings share no vertices

**Planar:** No crossings

**RAC:** Right angle crossings

$\text{RAC}_k$ : with  $\leq k$  bends per edge

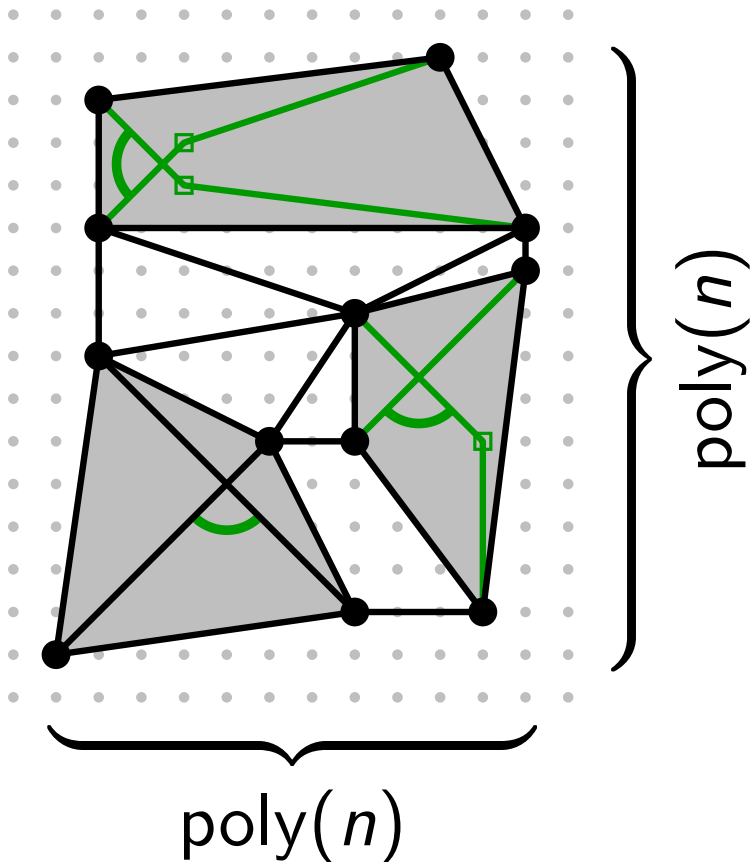
$\text{RAC}_0$ : with straight-line edges



# Introduction: Beyond-Planar Graphs

2

## Types of drawings:



**1-Planar:**  $\leq 1$  crossings per edge

**NIC-Planar:** Two crossings share  $\leq 1$  vertices

**IC-Planar:** Two crossings share no vertices

**Planar:** No crossings

**RAC:** Right angle crossings

$\text{RAC}_k$ : with  $\leq k$  bends per edge

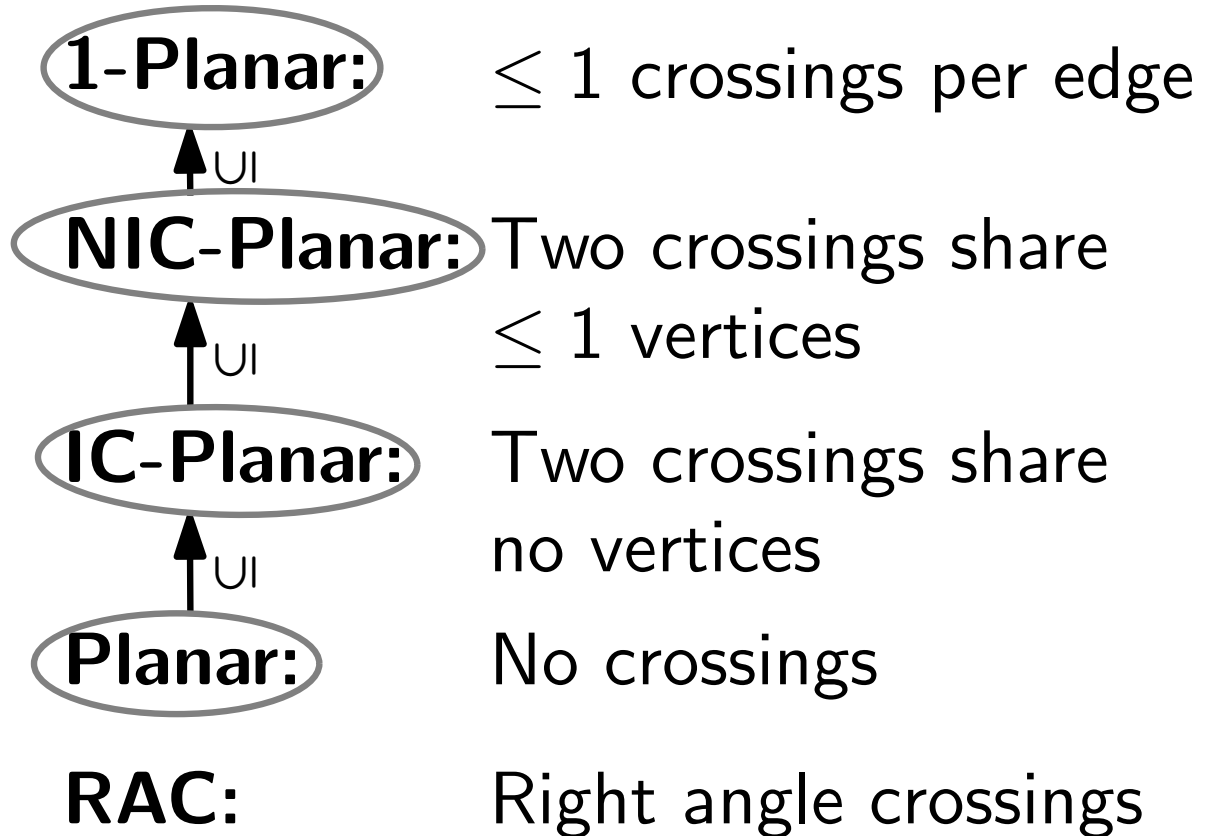
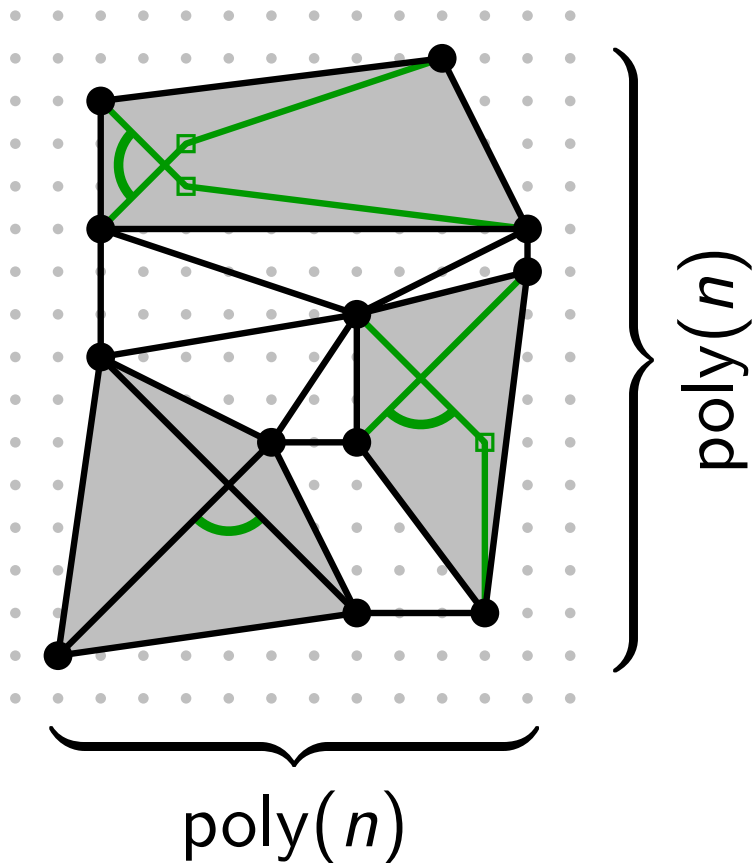
$\text{RAC}_0$ : with straight-line edges

$\text{RAC}^{\text{poly}}$ : in polynomial area

# Introduction: Beyond-Planar Graphs

2

## Types of drawings:



$\text{RAC}_k$ : with  $\leq k$  bends per edge

$\text{RAC}_0$ : with straight-line edges

$\text{RAC}^{\text{poly}}$ : in polynomial area

# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]

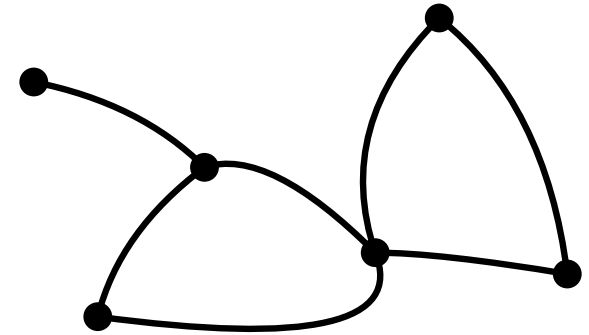
[Chrobak and Payne, 1995]

# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

**Idea:**



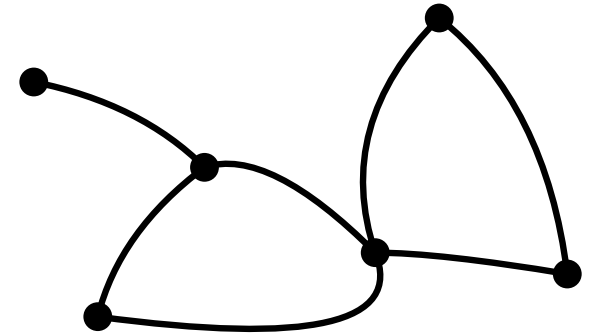
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.



# Introduction: The Shift Algorithm

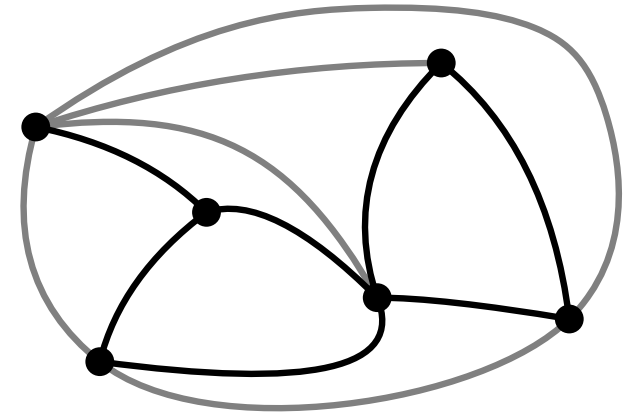
3

[de Fraysseix, Pach, and Pollack, 1990]

[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.





# Introduction: The Shift Algorithm

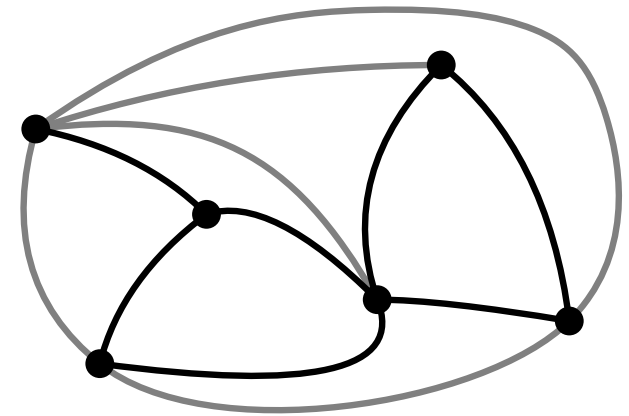
3

[de Fraysseix, Pach, and Pollack, 1990]

[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .



# Introduction: The Shift Algorithm

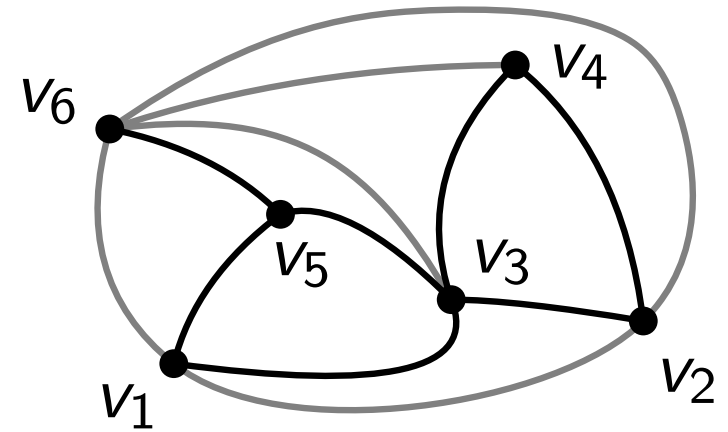
3

[de Fraysseix, Pach, and Pollack, 1990]

[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .



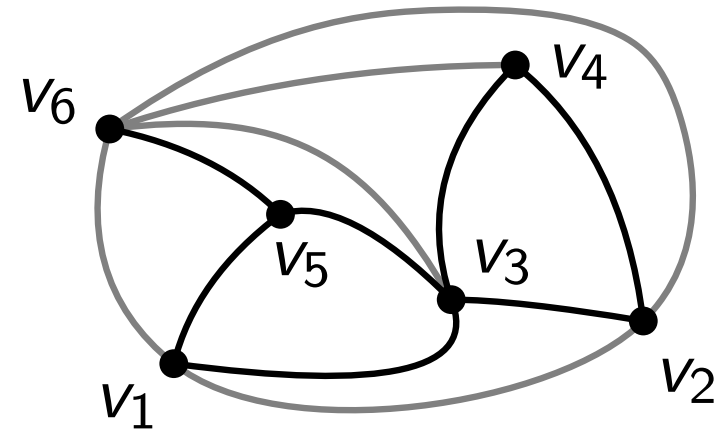
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
- Draw the graph:



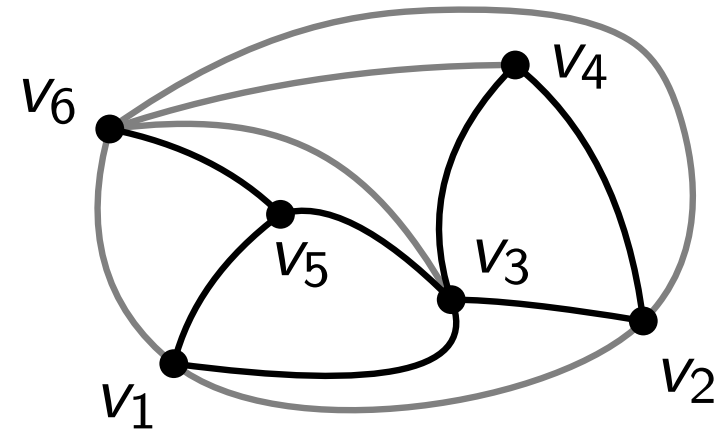
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
- Draw the graph:
  - Start with triangle  $v_1, v_2, v_3$ .



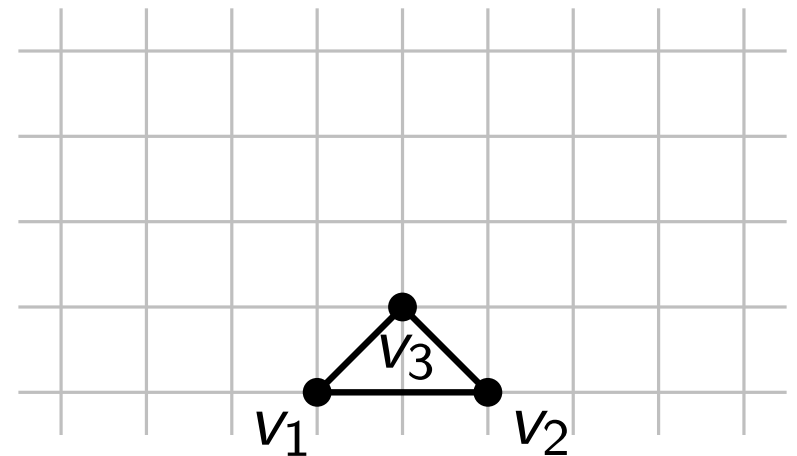
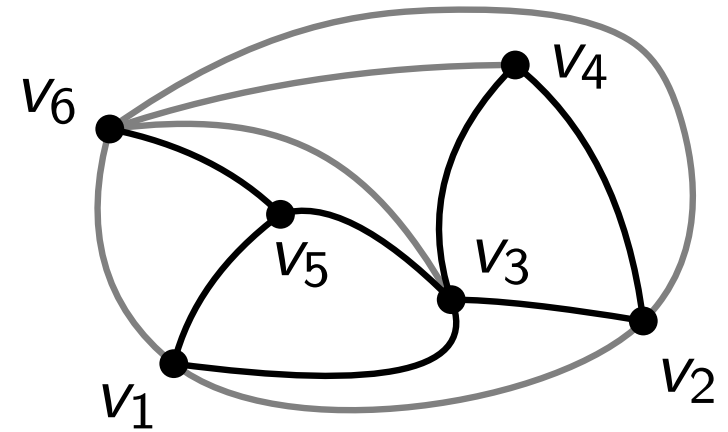
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
- Draw the graph:
  - Start with triangle  $v_1, v_2, v_3$ .



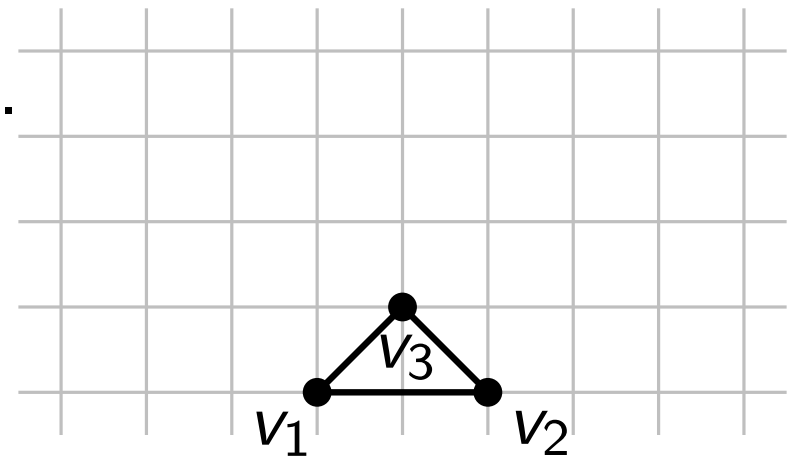
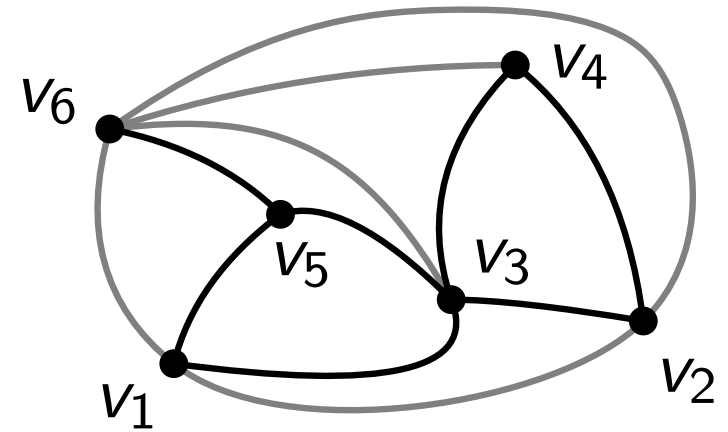
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
- Draw the graph:
  - Start with triangle  $v_1, v_2, v_3$ .
  - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .



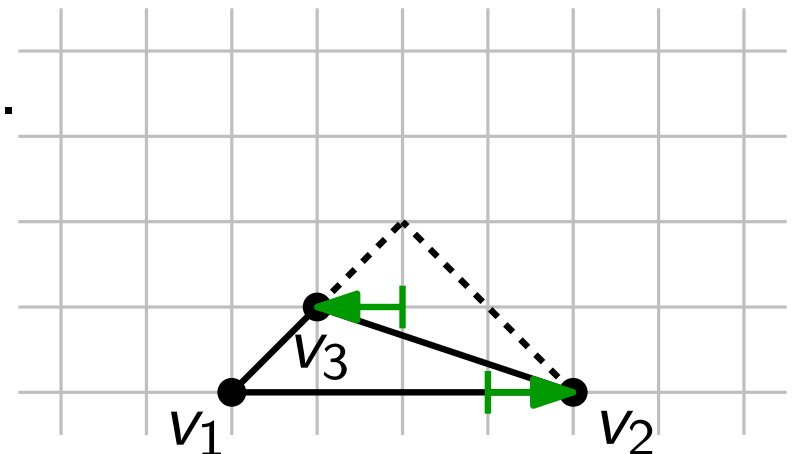
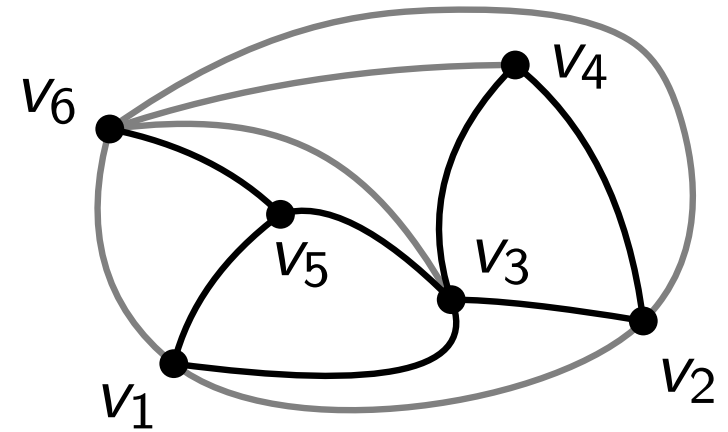
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
- Draw the graph:
  - Start with triangle  $v_1, v_2, v_3$ .
  - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .



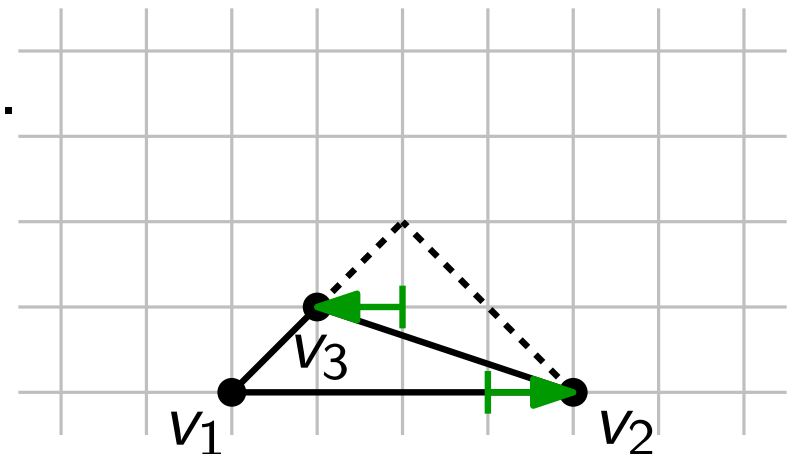
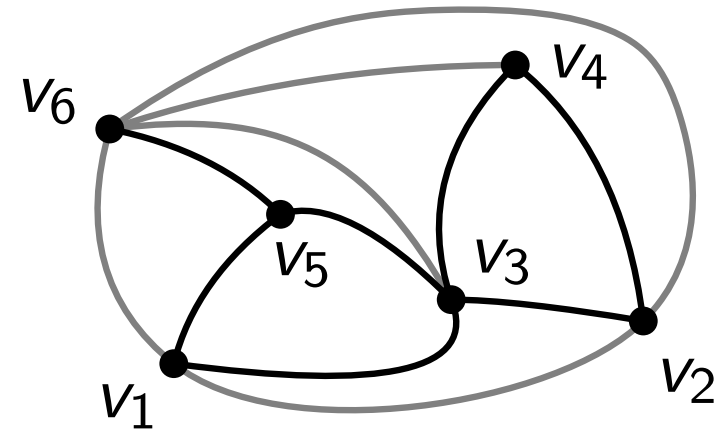
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
- Draw the graph:
  - Start with triangle  $v_1, v_2, v_3$ .
  - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
  - Add  $v_k$  to the outer face.





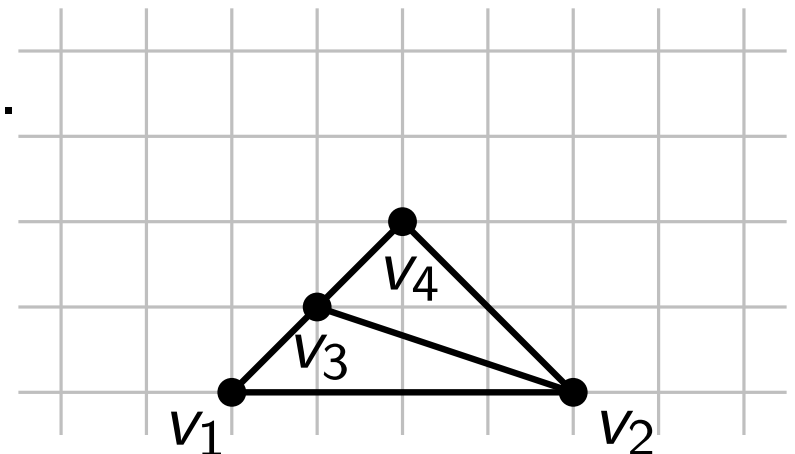
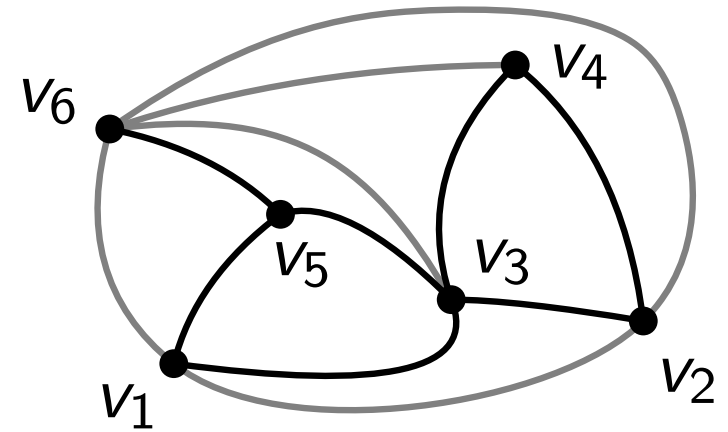
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
- Draw the graph:
  - Start with triangle  $v_1, v_2, v_3$ .
  - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
  - Add  $v_k$  to the outer face.



# Introduction: The Shift Algorithm

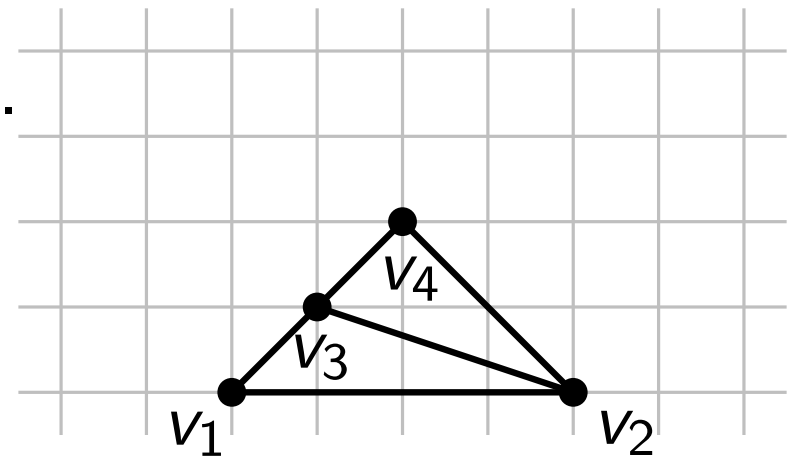
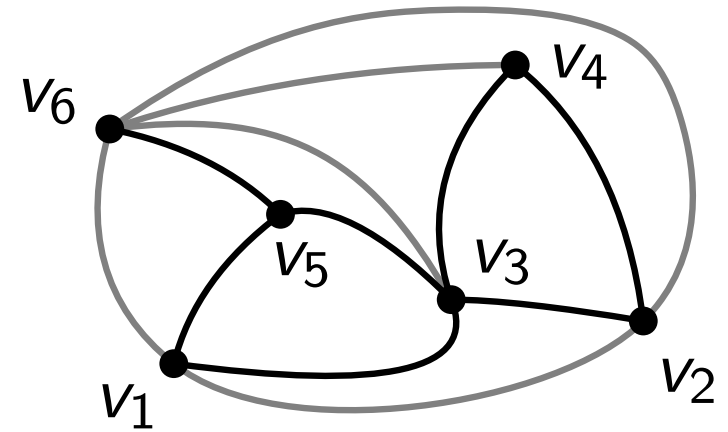
3

[de Fraysseix, Pach, and Pollack, 1990]

[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
  - Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
  - Draw the graph:
    - Start with triangle  $v_1, v_2, v_3$ .
    - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
    - Add  $v_k$  to the outer face.
- $\Rightarrow$  all slopes on outer face  $\pm 1$   
(except for  $v_1 v_2$ )



# Introduction: The Shift Algorithm

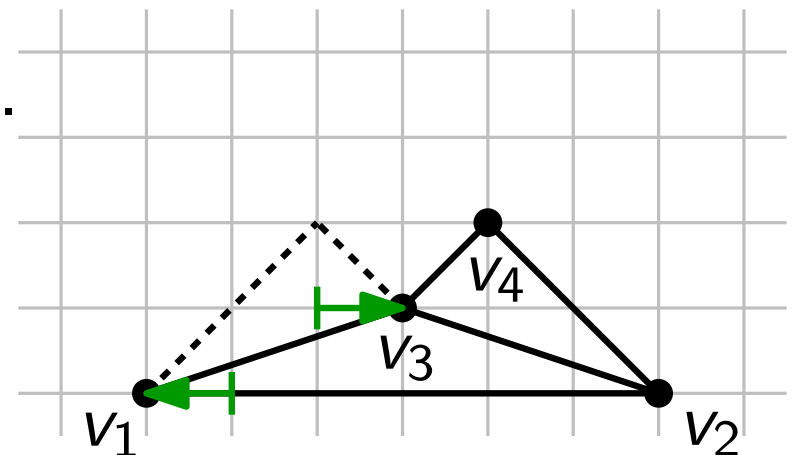
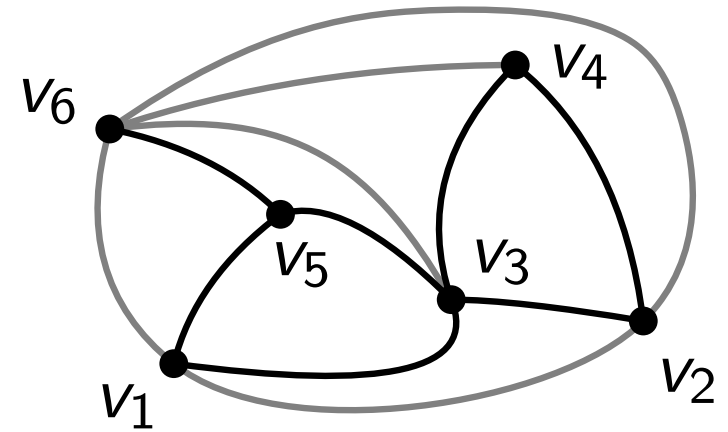
3

[de Fraysseix, Pach, and Pollack, 1990]

[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
  - Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
  - Draw the graph:
    - Start with triangle  $v_1, v_2, v_3$ .
    - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
    - Add  $v_k$  to the outer face.
- $\Rightarrow$  all slopes on outer face  $\pm 1$   
(except for  $v_1 v_2$ )



# Introduction: The Shift Algorithm

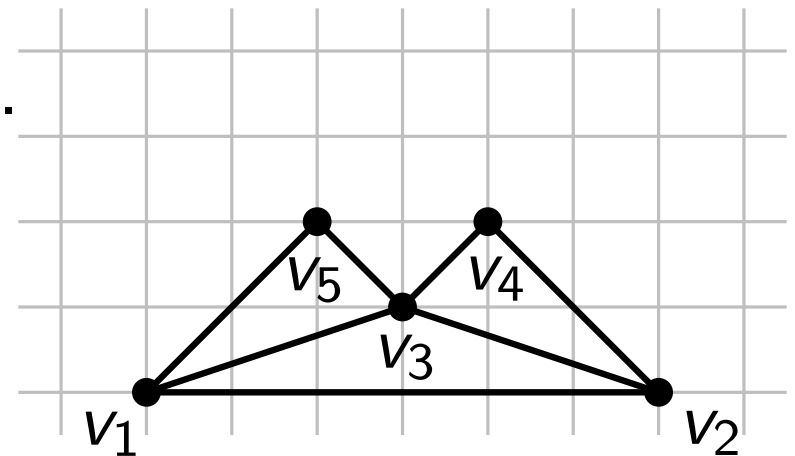
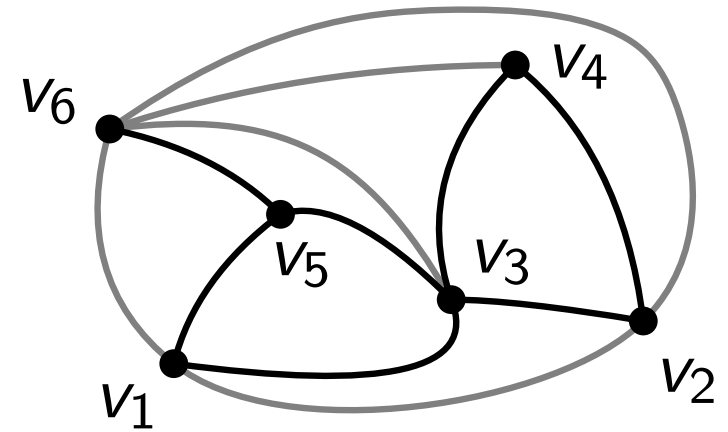
3

[de Fraysseix, Pach, and Pollack, 1990]

[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
  - Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
  - Draw the graph:
    - Start with triangle  $v_1, v_2, v_3$ .
    - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
    - Add  $v_k$  to the outer face.
- $\Rightarrow$  all slopes on outer face  $\pm 1$   
(except for  $v_1 v_2$ )



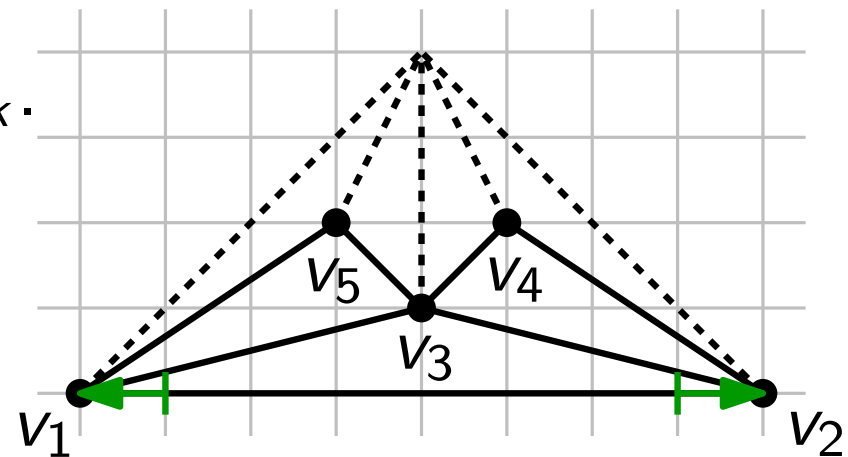
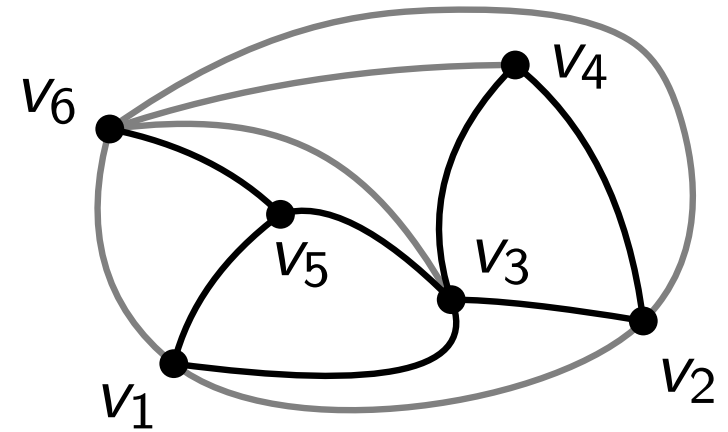
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
  - Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
  - Draw the graph:
    - Start with triangle  $v_1, v_2, v_3$ .
    - For  $v_k$ :
      - Shift first & last neighbor of  $v_k$ .
    - Add  $v_k$  to the outer face.
- $\Rightarrow$  all slopes on outer face  $\pm 1$   
(except for  $v_1 v_2$ )



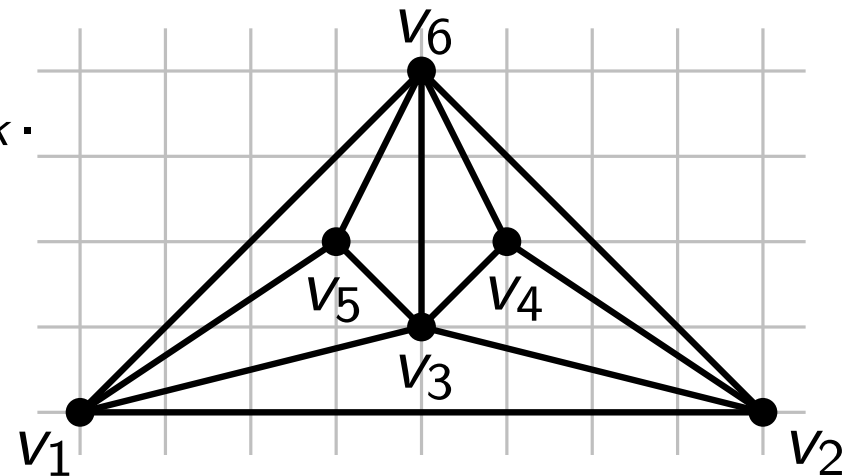
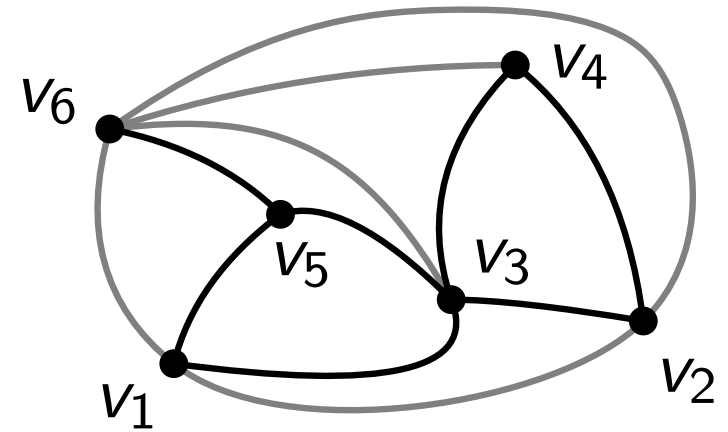
# Introduction: The Shift Algorithm

3

[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
  - Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
  - Draw the graph:
    - Start with triangle  $v_1, v_2, v_3$ .
    - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
    - Add  $v_k$  to the outer face.
- $\Rightarrow$  all slopes on outer face  $\pm 1$   
(except for  $v_1 v_2$ )



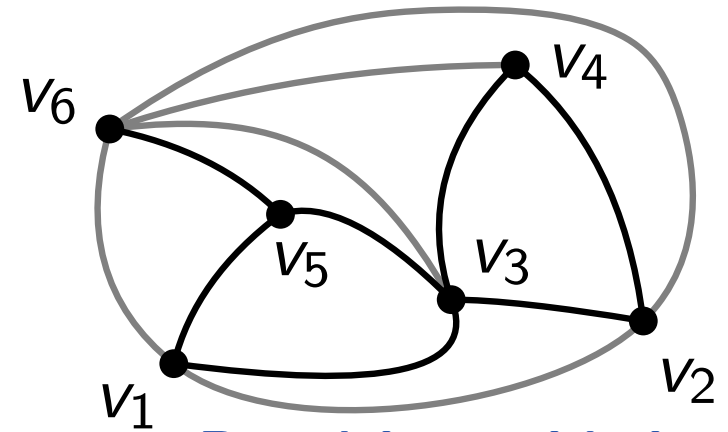
# Introduction: The Shift Algorithm

3

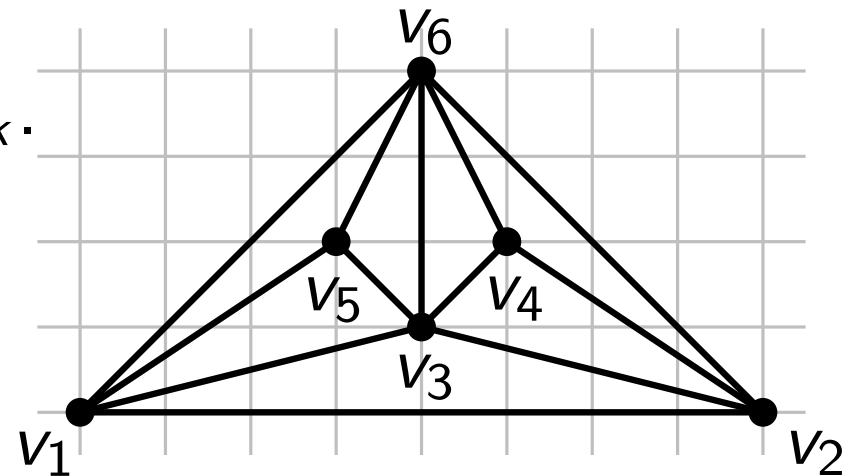
[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
  - Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
  - Draw the graph:
    - Start with triangle  $v_1, v_2, v_3$ .
    - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
    - Add  $v_k$  to the outer face.
- $\Rightarrow$  all slopes on outer face  $\pm 1$   
(except for  $v_1 v_2$ )



**Resulting grid size:**  
 $(2n - 4) \times (n - 2)$



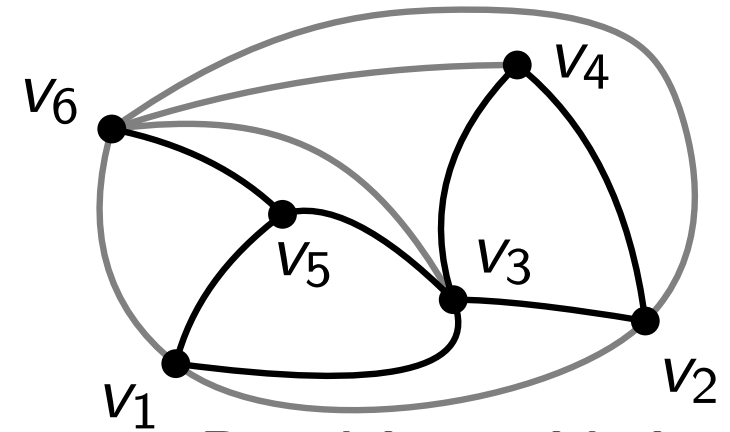
# Introduction: The Shift Algorithm

3

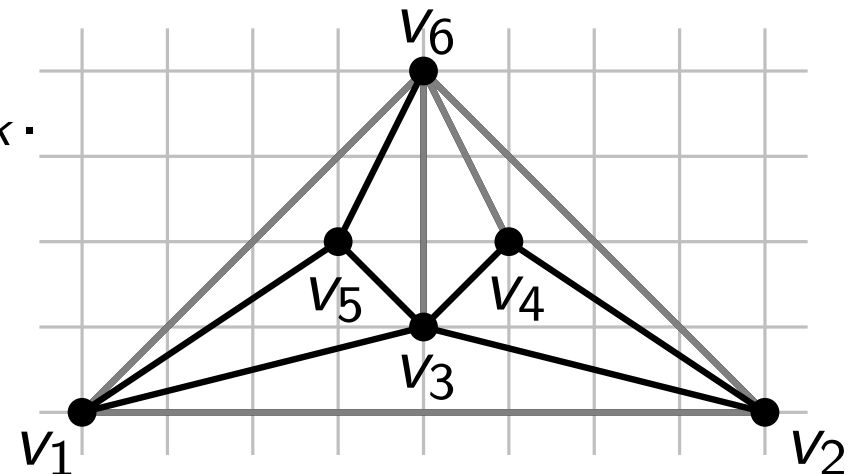
[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

- Triangulate given plane graph.
  - Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
  - Draw the graph:
    - Start with triangle  $v_1, v_2, v_3$ .
    - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
    - Add  $v_k$  to the outer face.
- $\Rightarrow$  all slopes on outer face  $\pm 1$   
(except for  $v_1 v_2$ )



**Resulting grid size:**  
 $(2n - 4) \times (n - 2)$





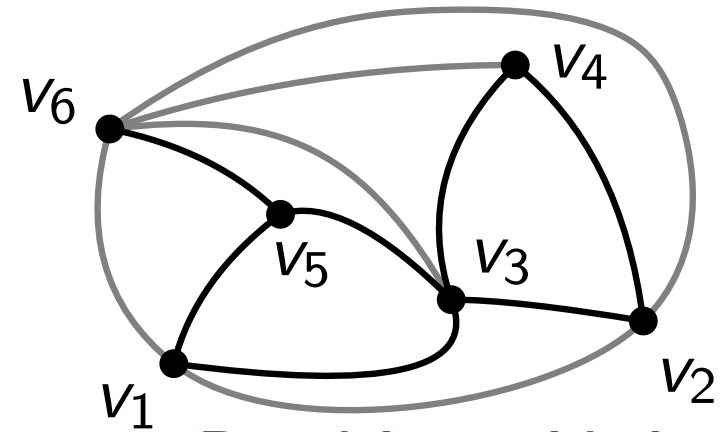
# Introduction: The Shift Algorithm

3

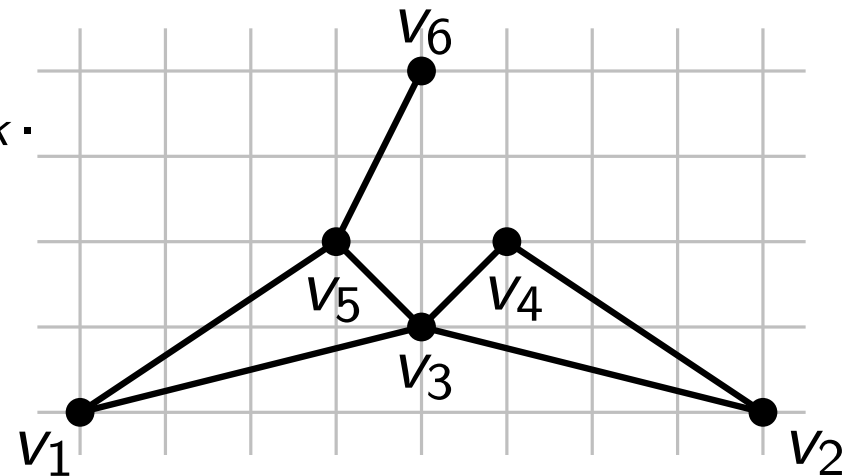
[de Fraysseix, Pach, and Pollack, 1990]  
[Chrobak and Payne, 1995]

## Idea:

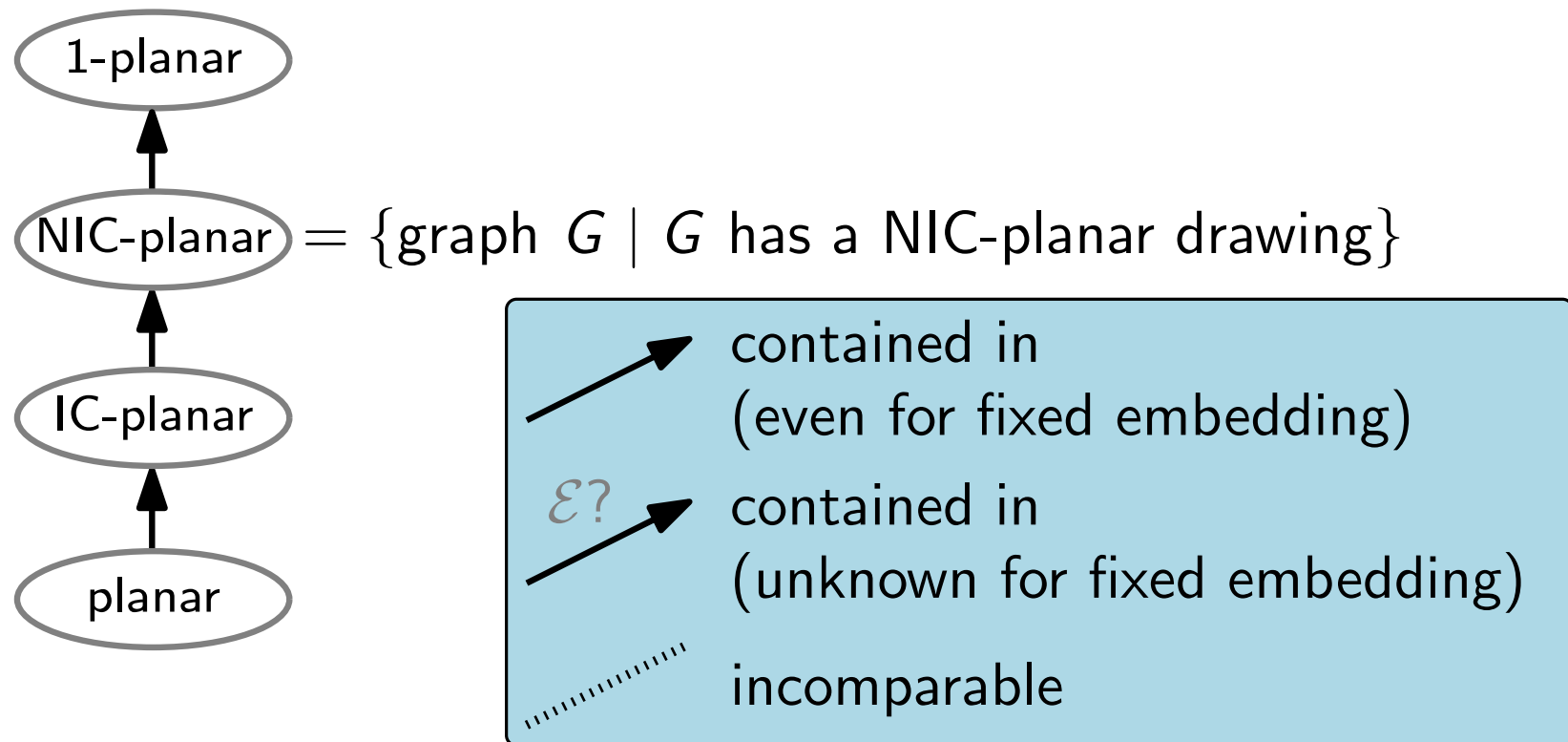
- Triangulate given plane graph.
  - Compute a canonical ordering of the vertices  $v_1, v_2, \dots, v_n$ .
  - Draw the graph:
    - Start with triangle  $v_1, v_2, v_3$ .
    - For  $v_k$ :  
Shift first & last neighbor of  $v_k$ .
    - Add  $v_k$  to the outer face.
- $\Rightarrow$  all slopes on outer face  $\pm 1$   
(except for  $v_1 v_2$ )



**Resulting grid size:**  
 $(2n - 4) \times (n - 2)$

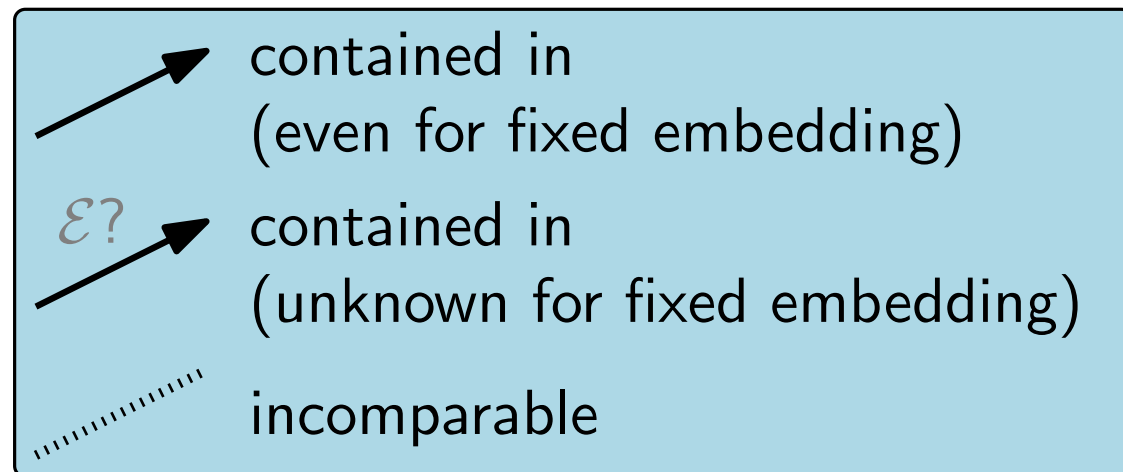
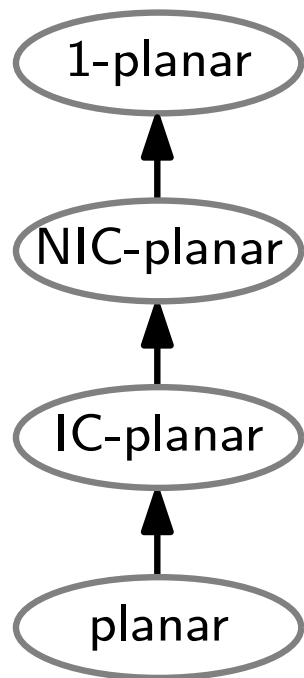
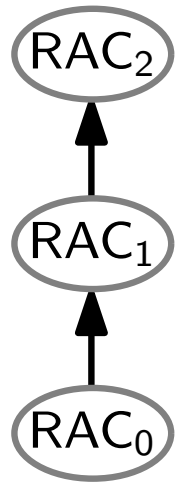


# Introduction: Related Work



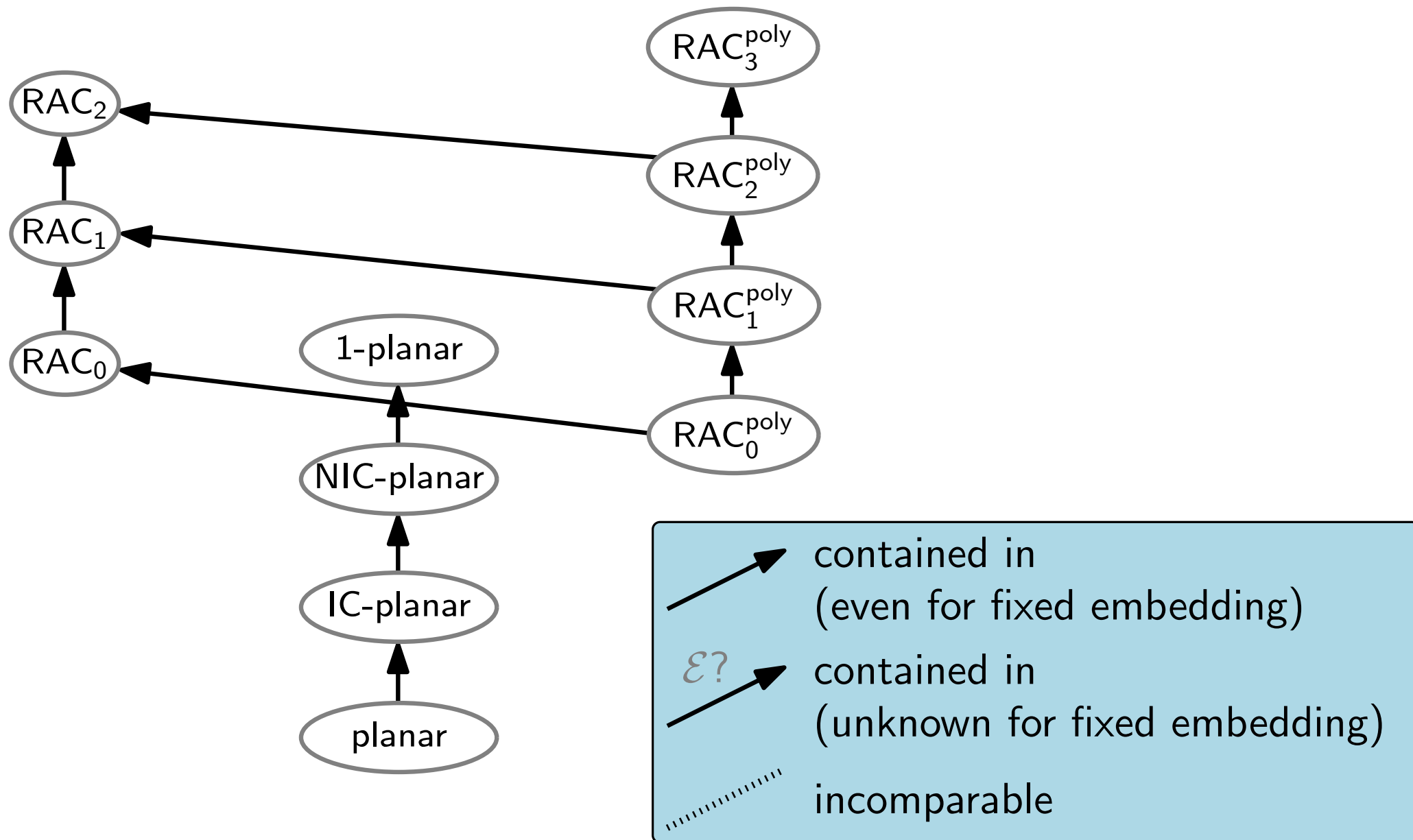
# Introduction: Related Work

4



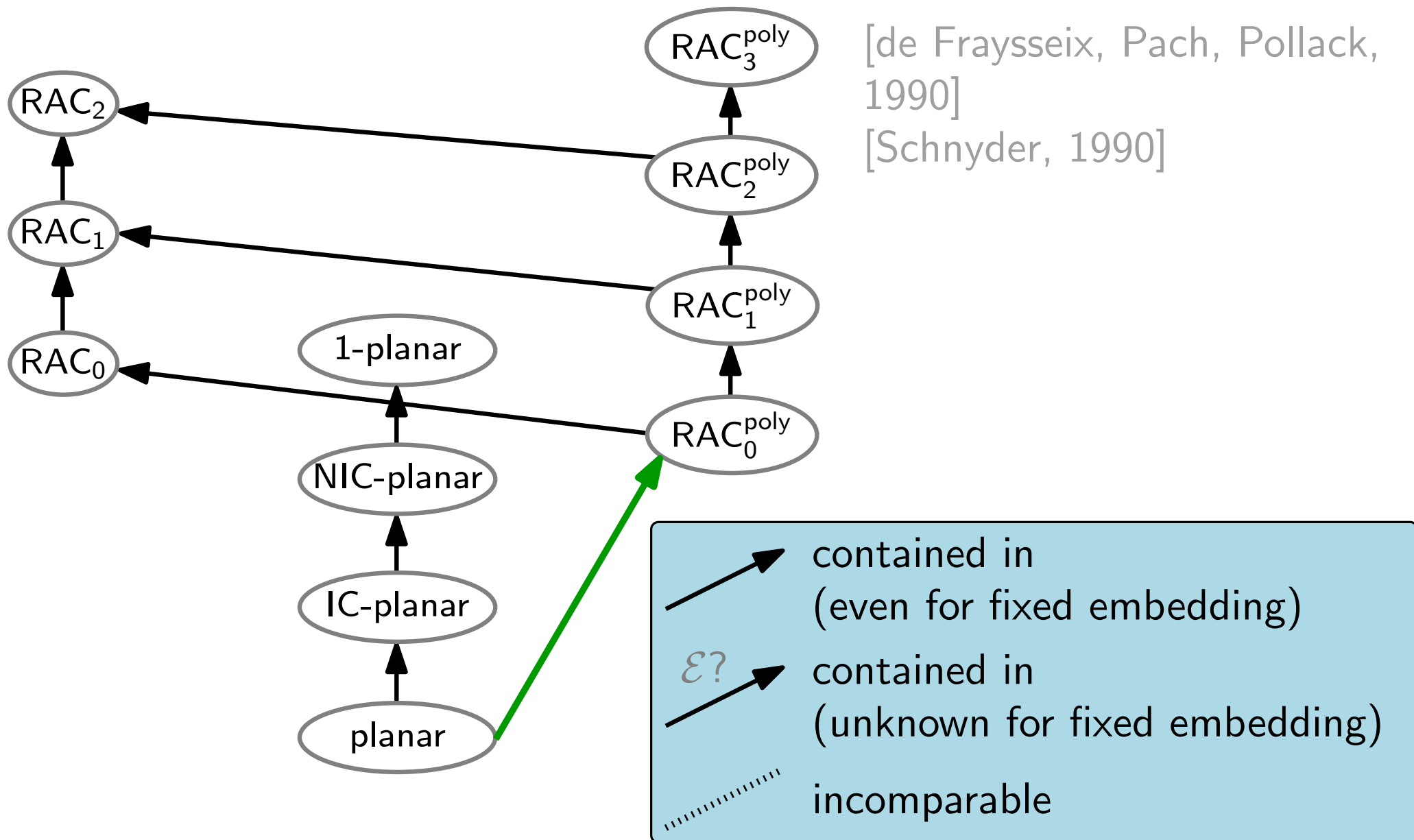
# Introduction: Related Work

4



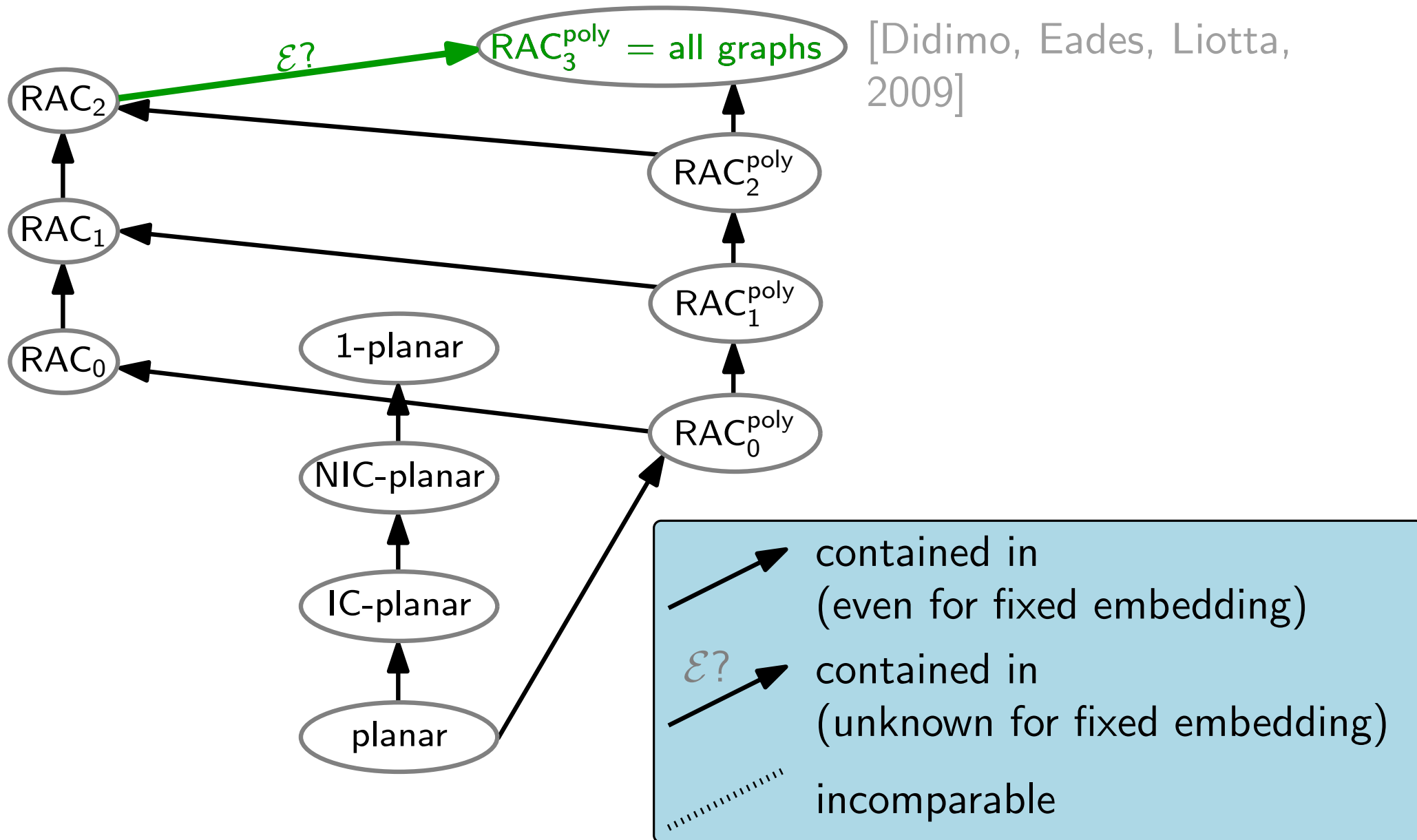
# Introduction: Related Work

4



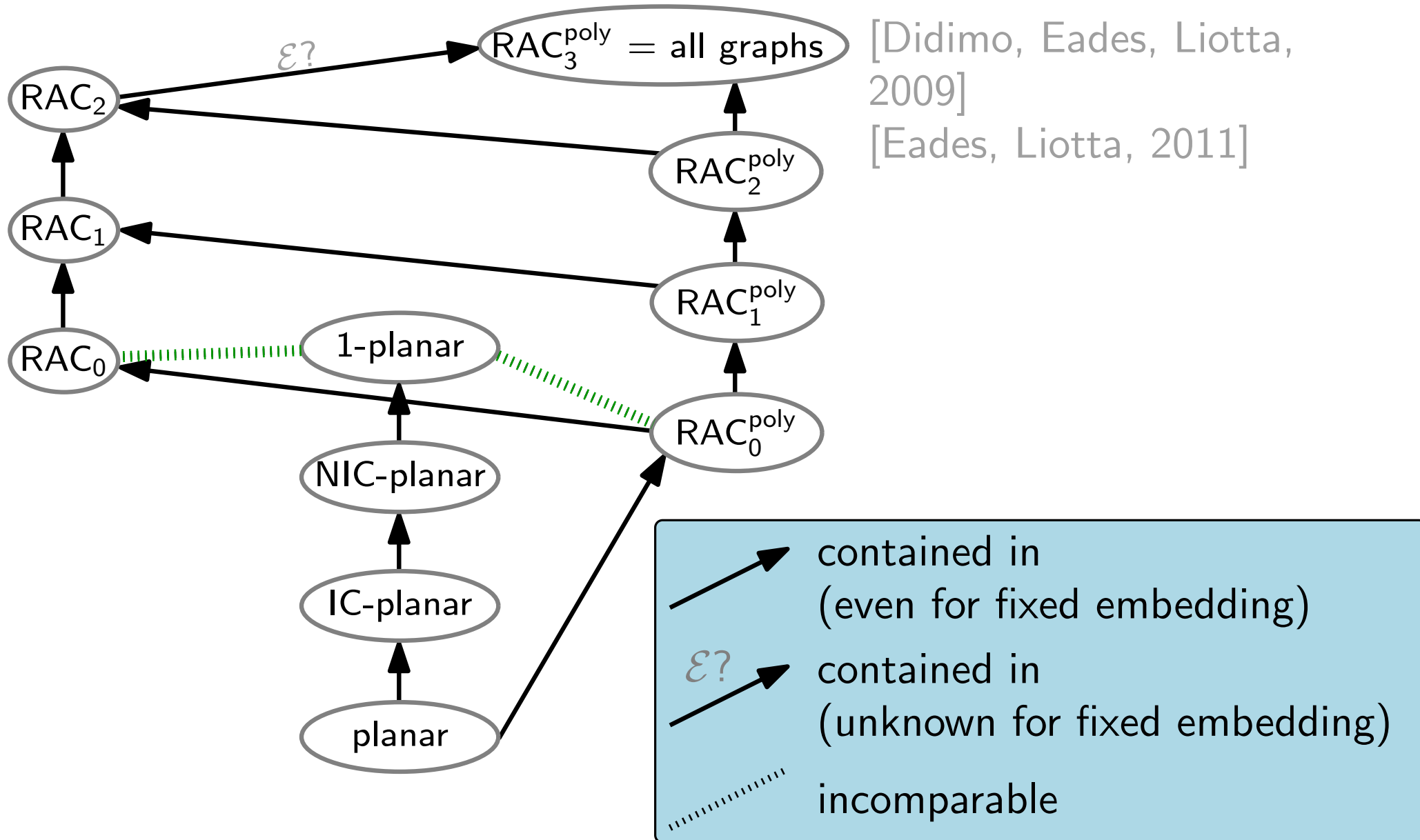
# Introduction: Related Work

4



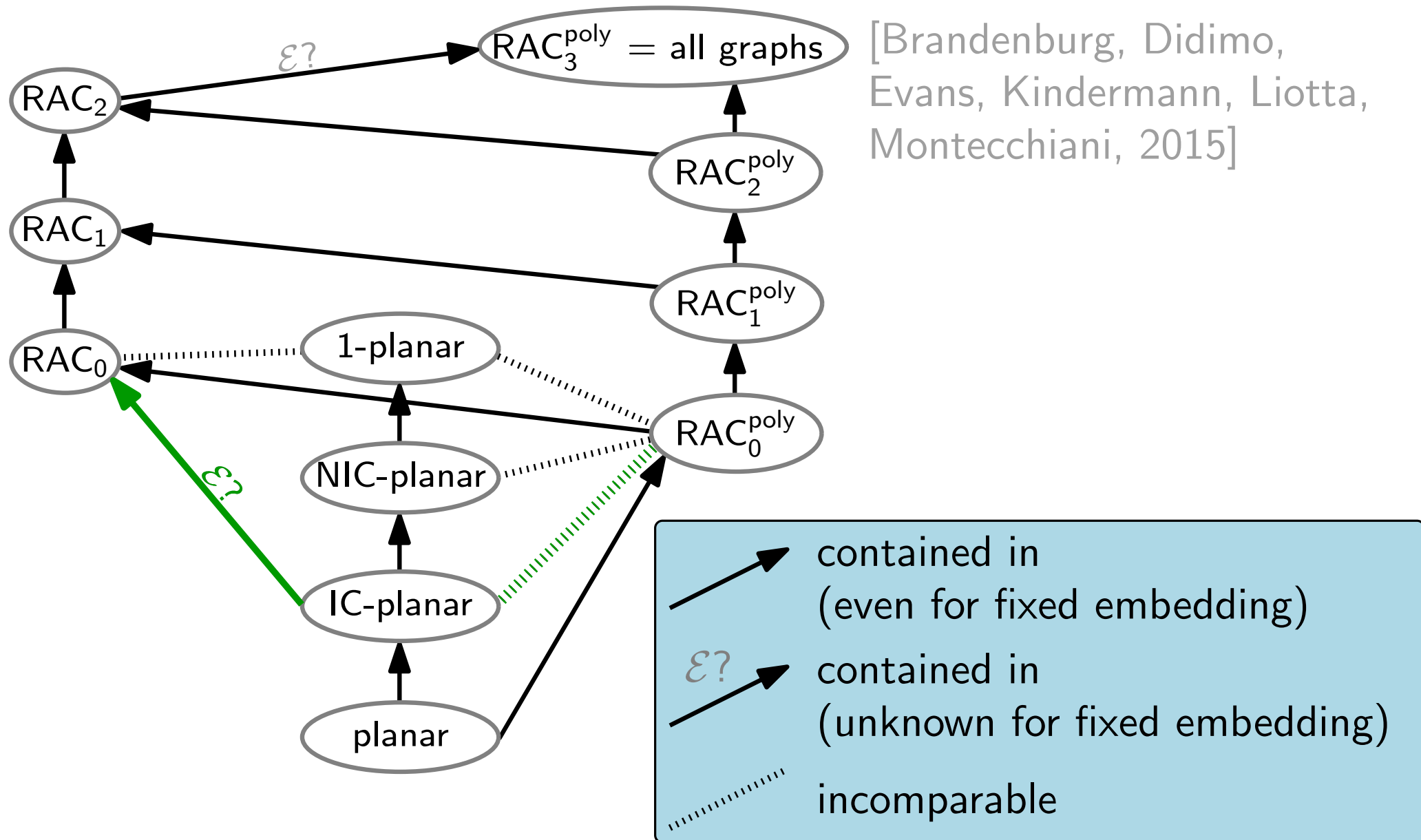
# Introduction: Related Work

4



# Introduction: Related Work

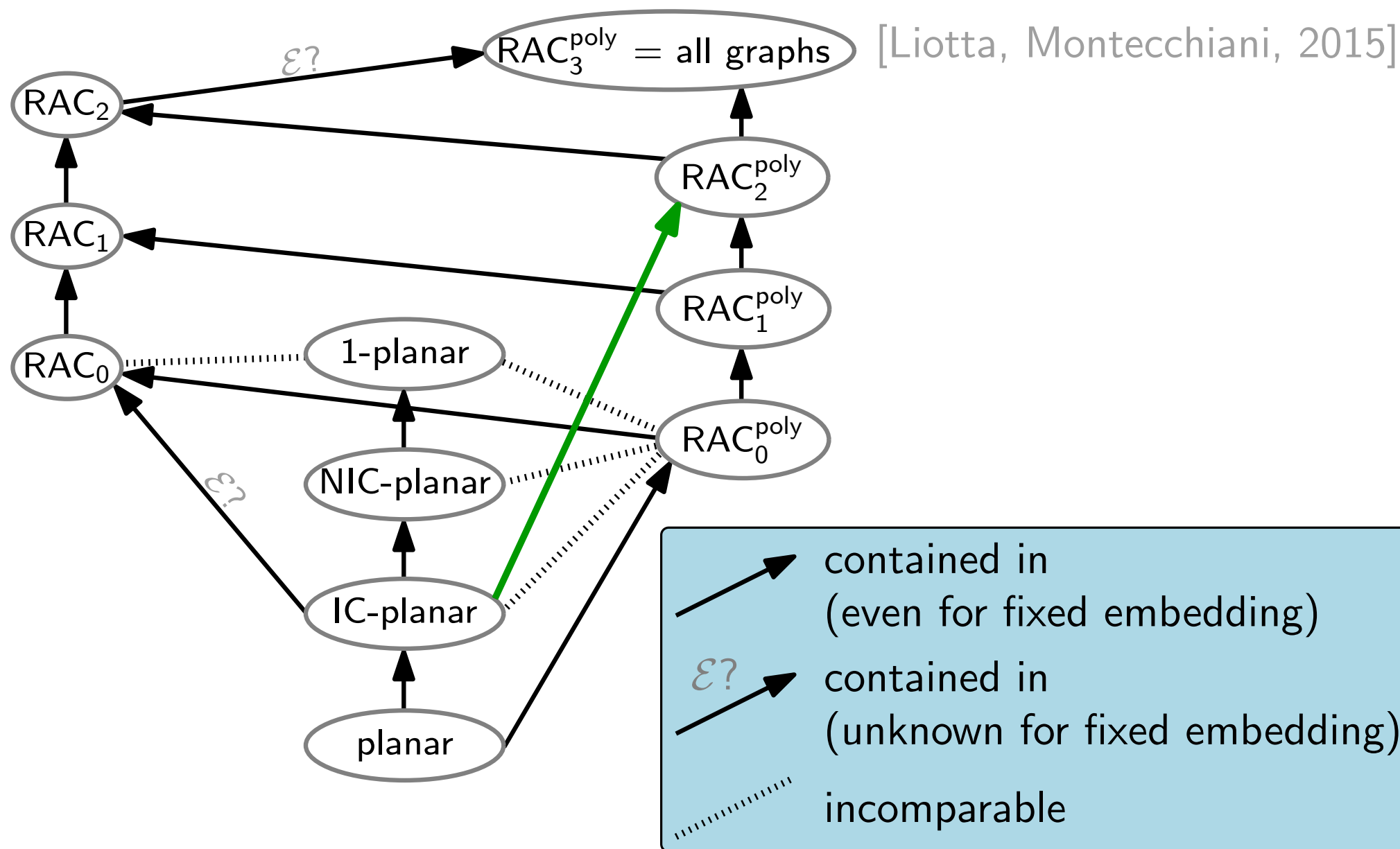
4





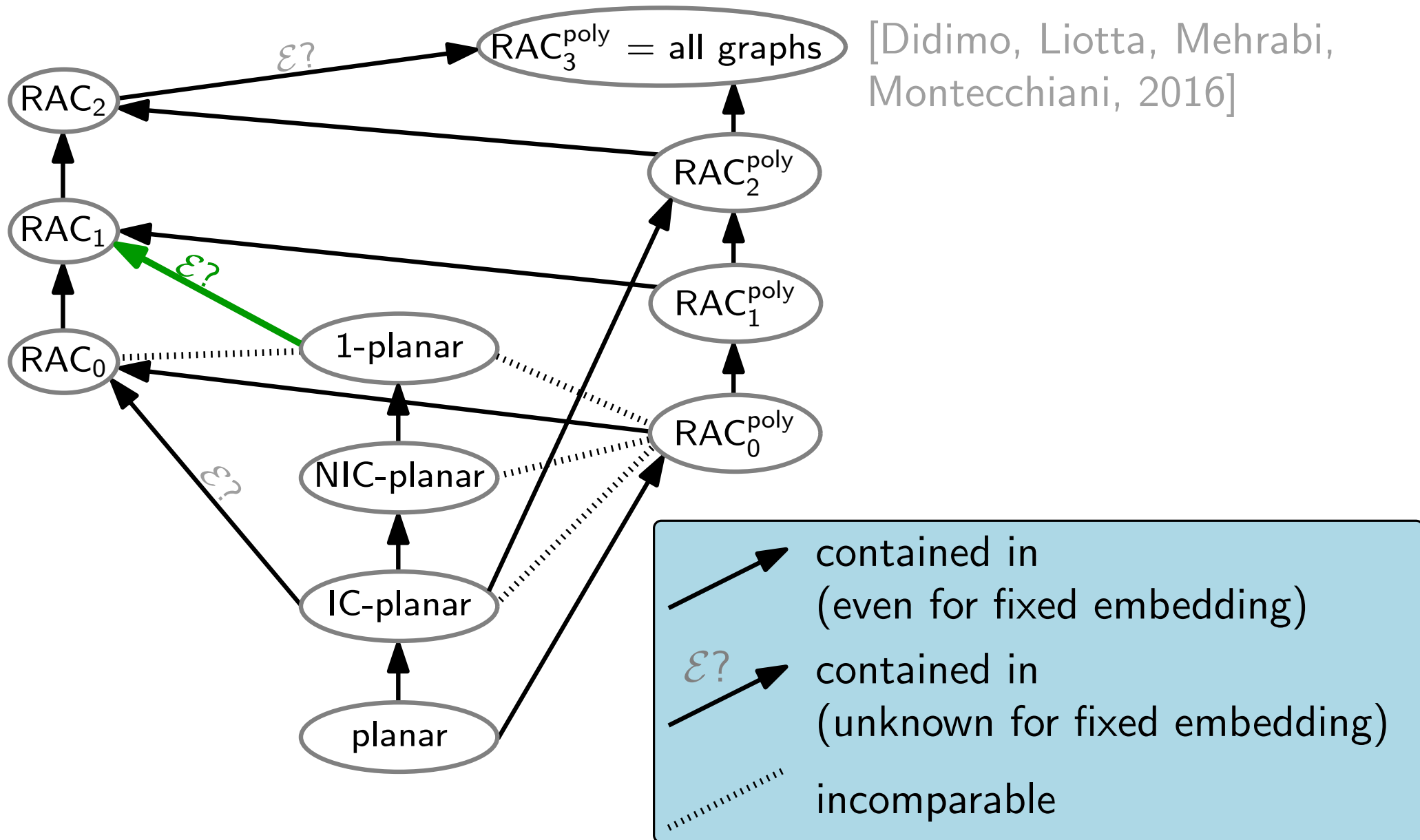
# Introduction: Related Work

4



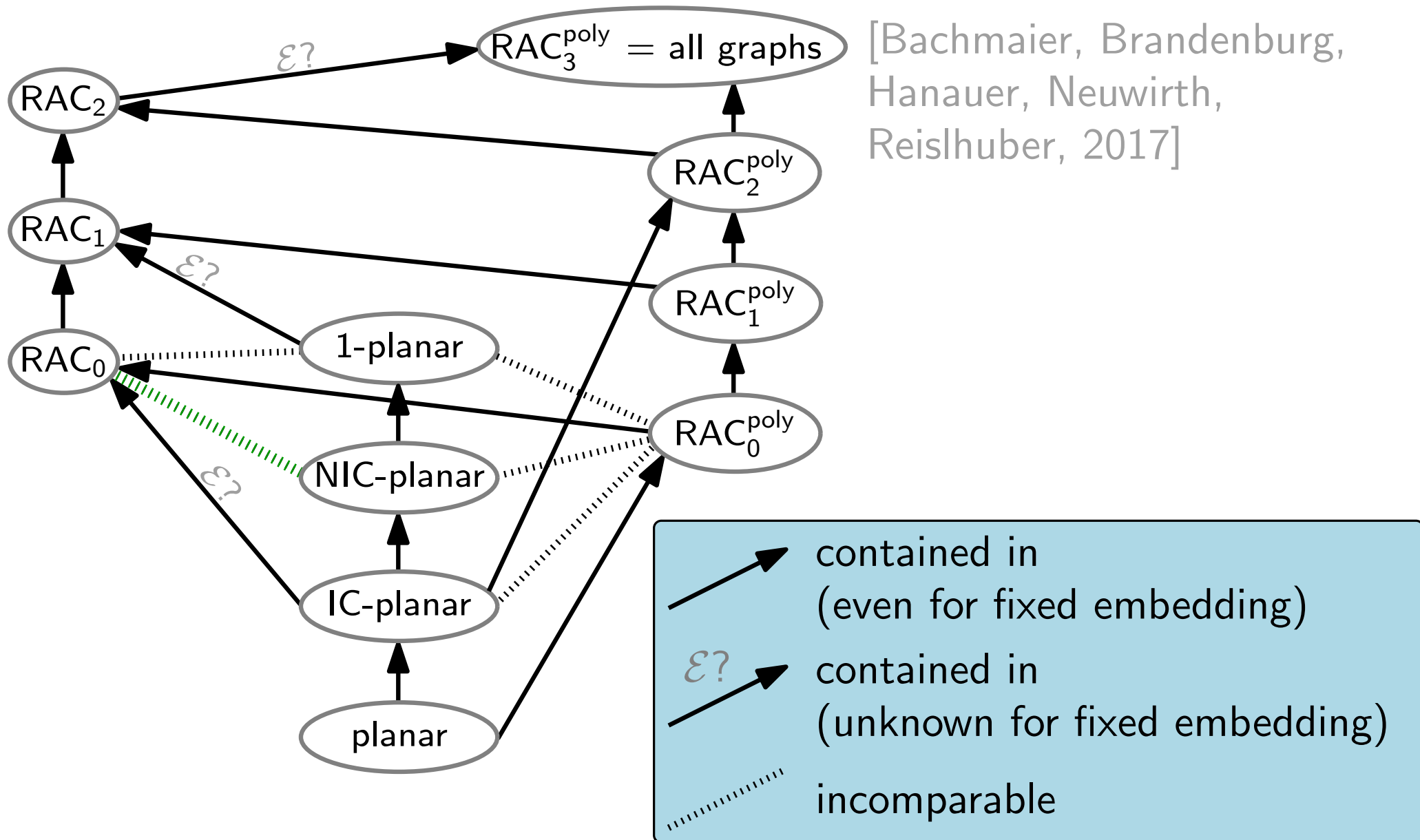
# Introduction: Related Work

4



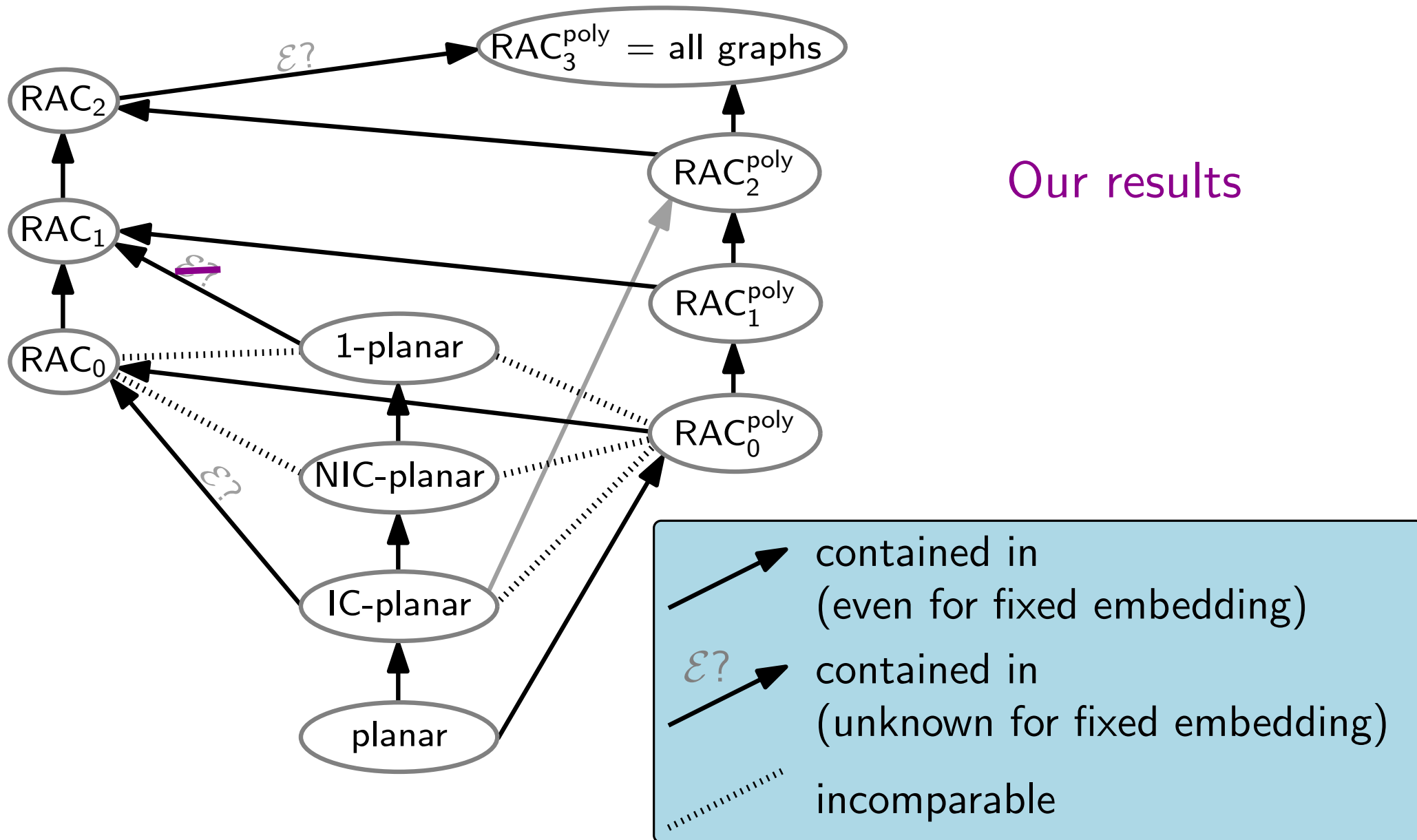
# Introduction: Related Work

4



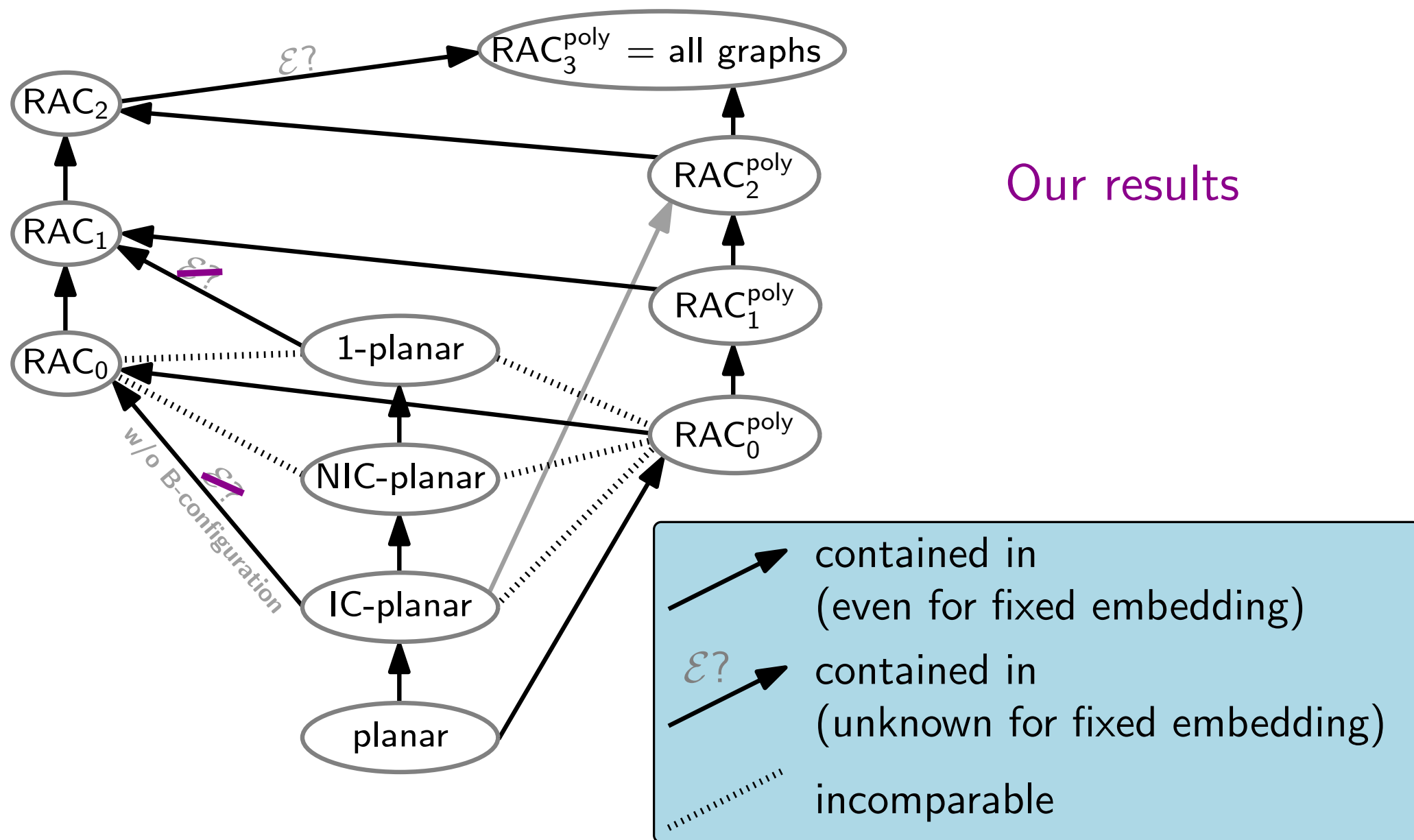
# Introduction: Related Work

4



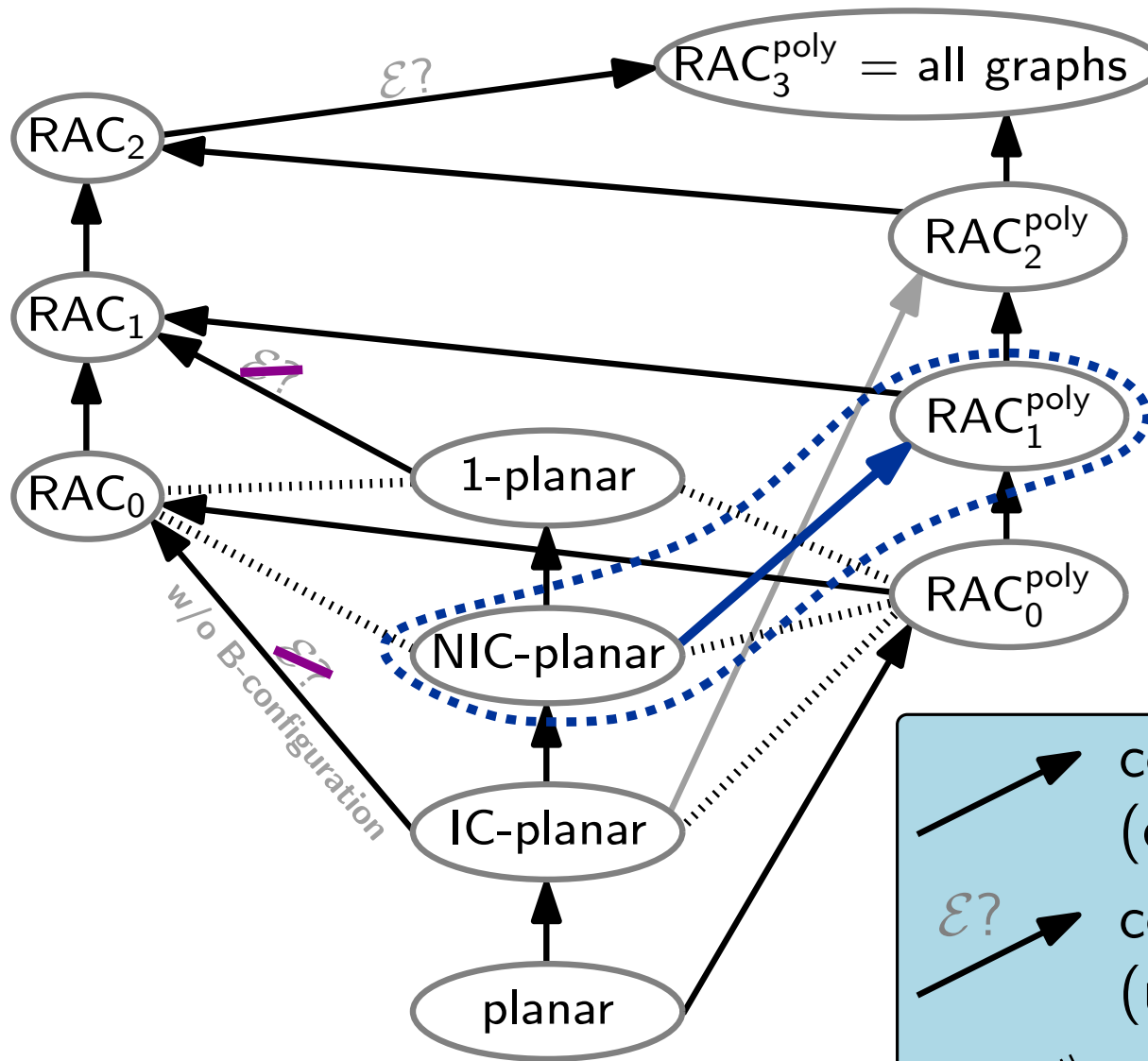
# Introduction: Related Work

4



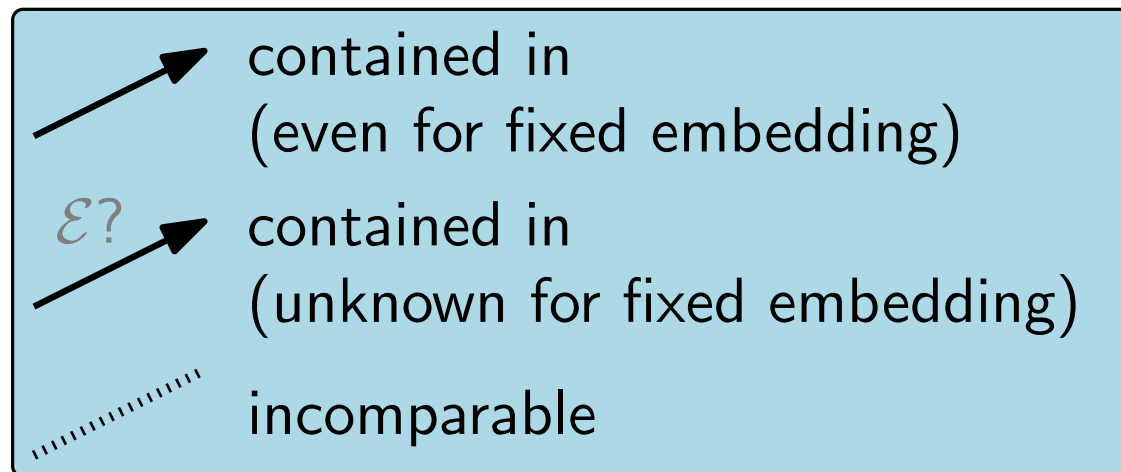
# Introduction: Related Work

4



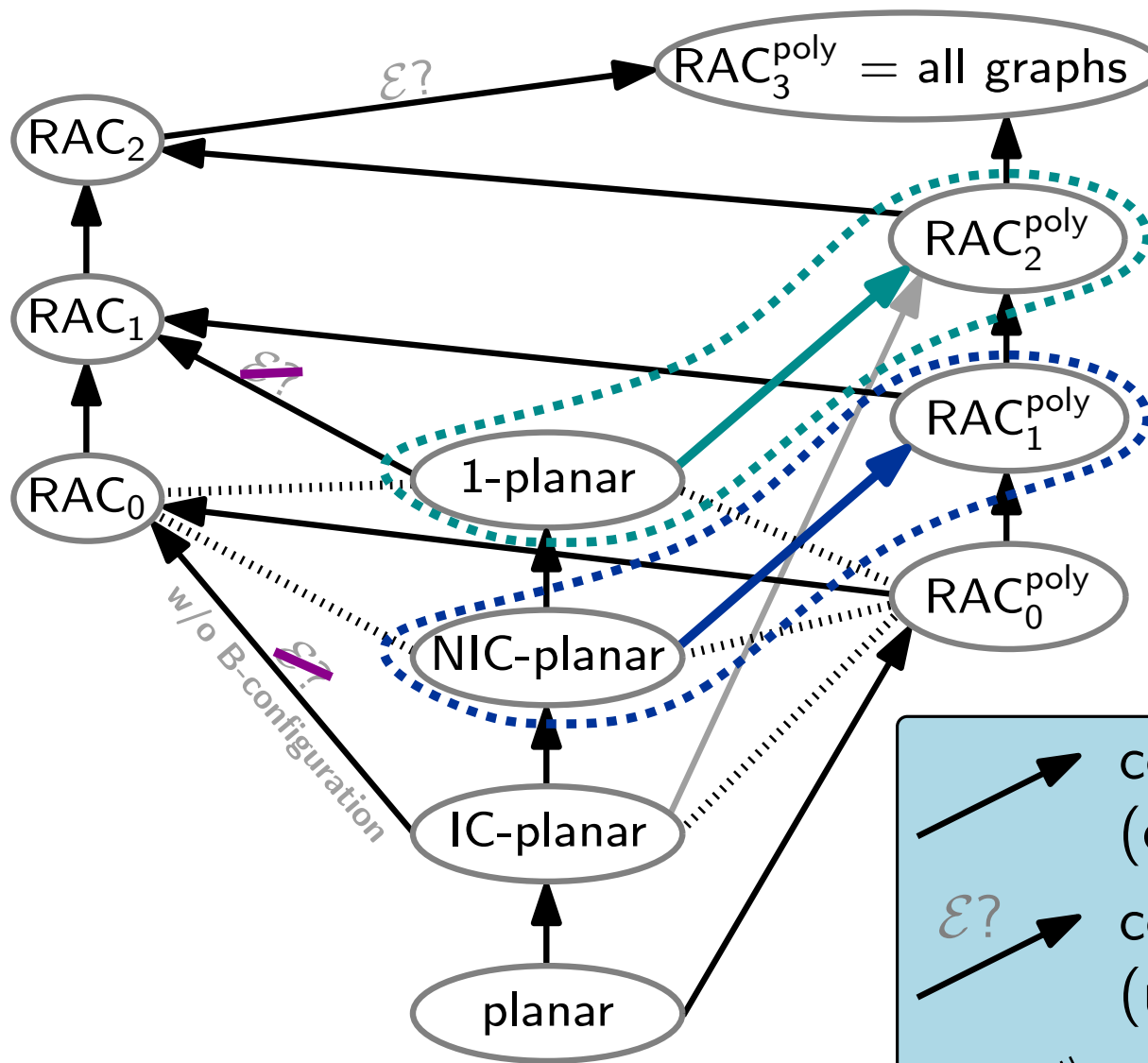
Our first main result:

NIC-plane graphs  
 $\subseteq \text{RAC}_1^{\text{poly}}$



# Introduction: Related Work

4

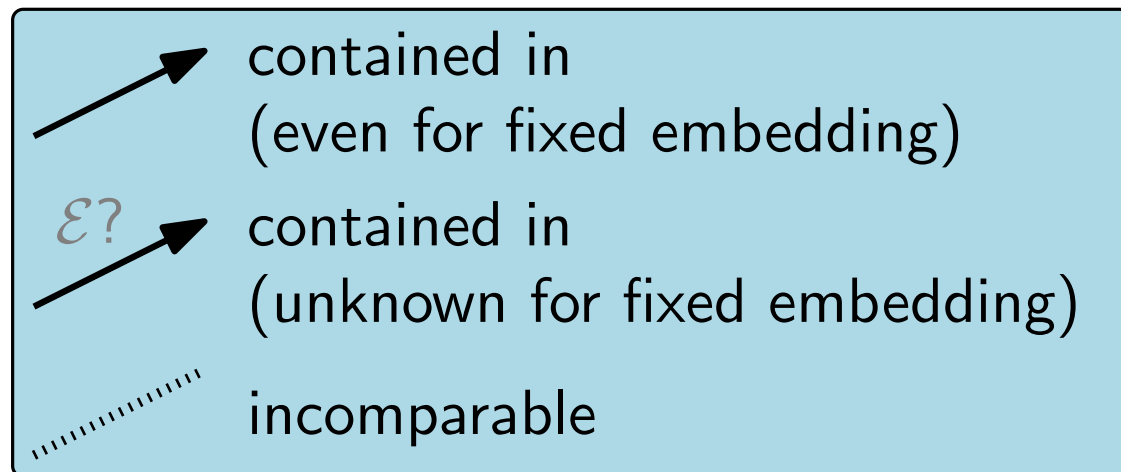


Our second main result:

$$\text{1-plane graphs} \subseteq \text{RAC}_2^{\text{poly}}$$

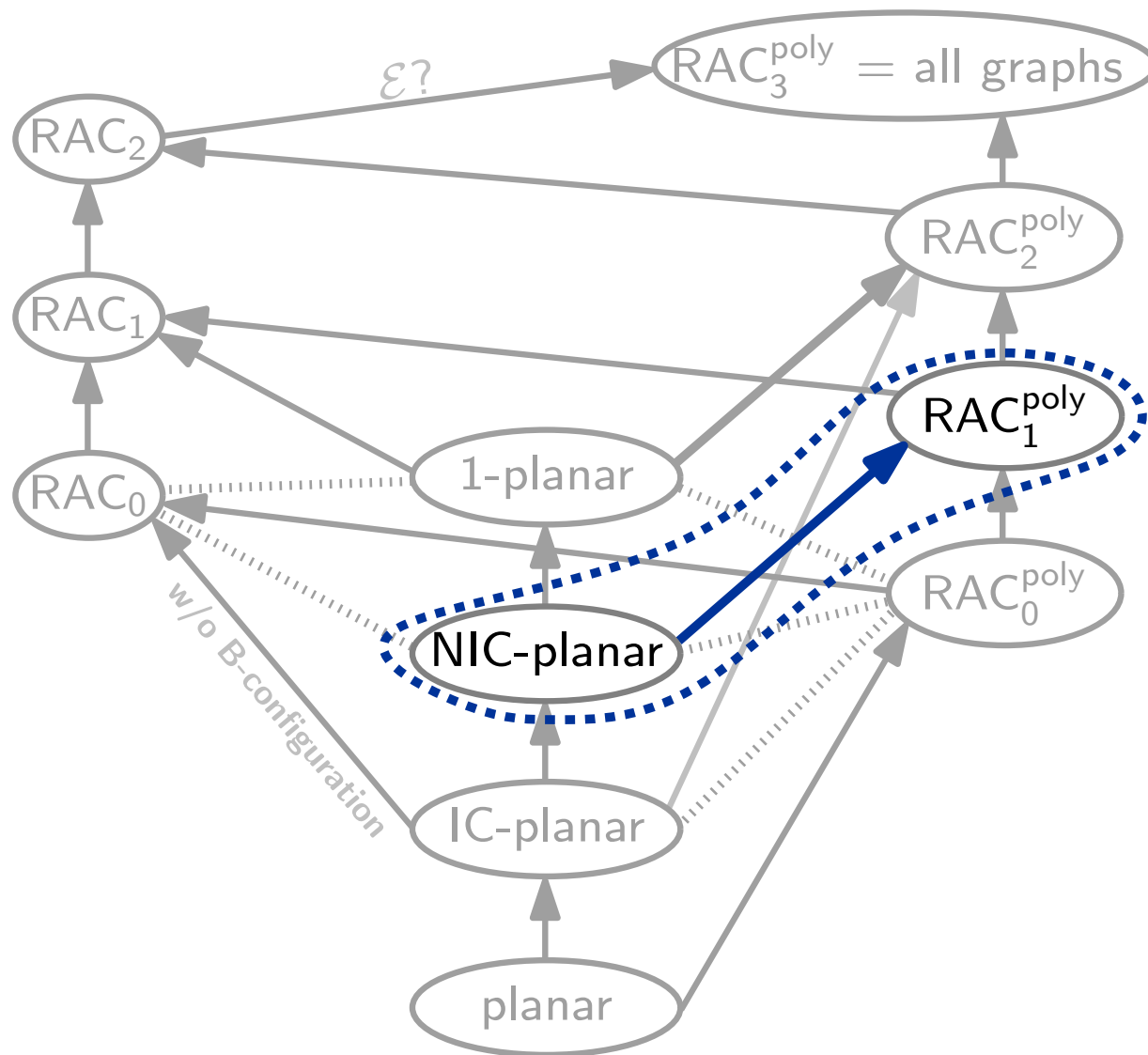
Our first main result:

$$\text{NIC-plane graphs} \subseteq \text{RAC}_1^{\text{poly}}$$



# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

5





# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

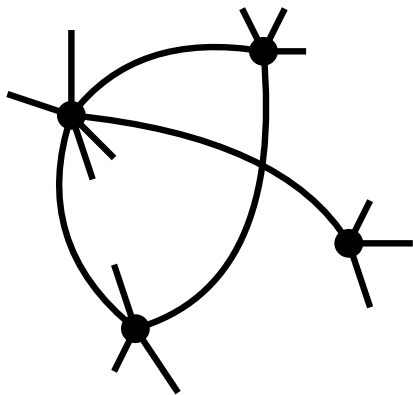
**Approach that nearly works:**

# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

**Approach that nearly works:**



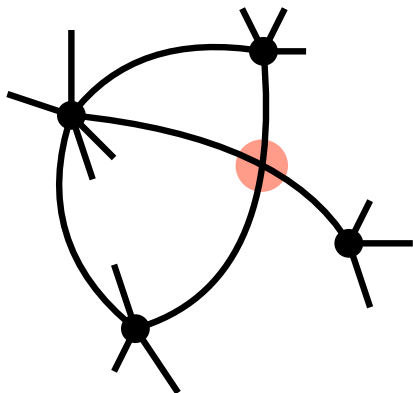
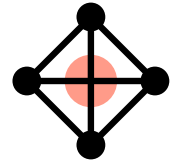
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each **crossing** by a so called *empty kite*:



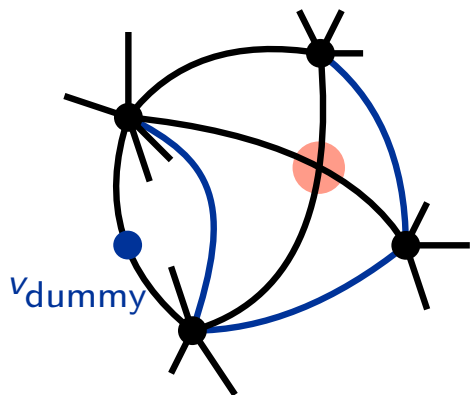
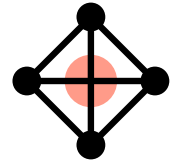
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each **crossing** by a so called *empty kite*:



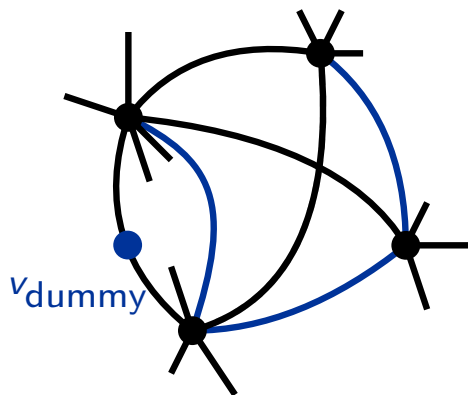
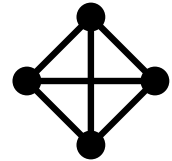
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*:
- Replace each pair of crossing edges by a single edge



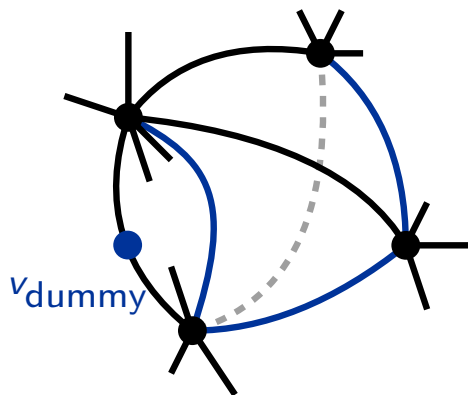
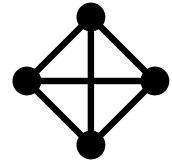
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*:
- Replace each pair of crossing edges by a single edge

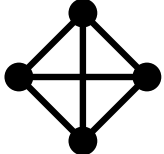


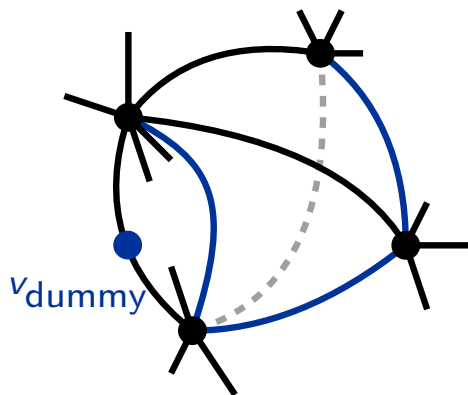
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm



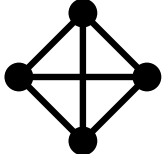


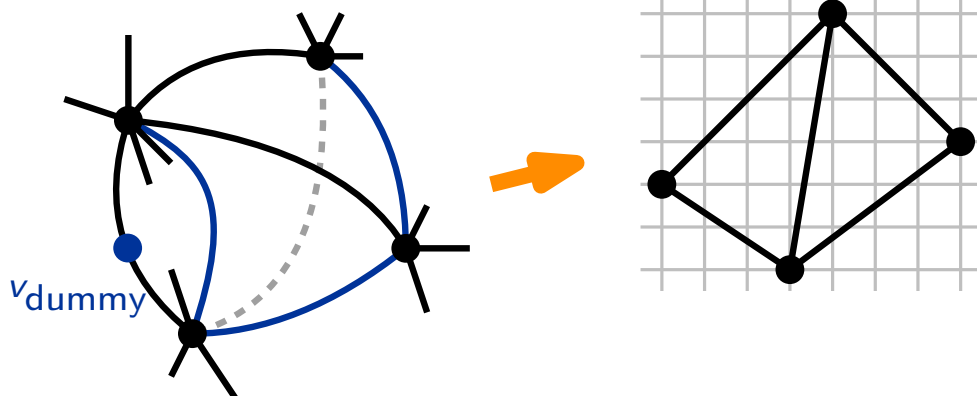
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm

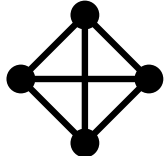


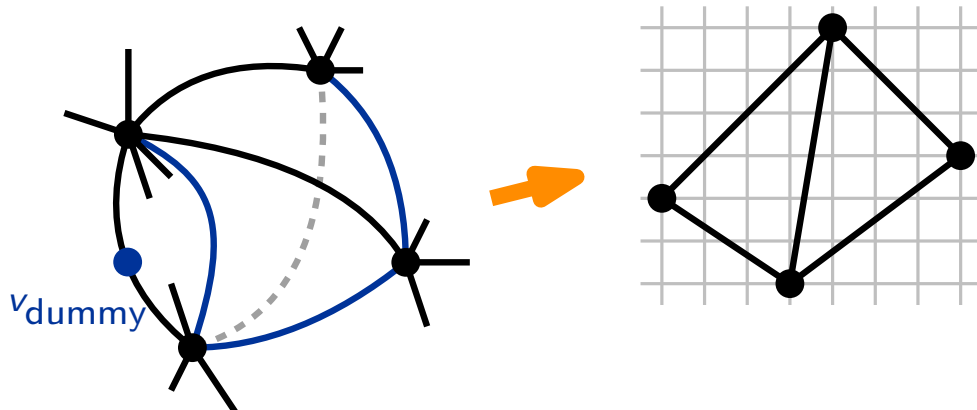
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

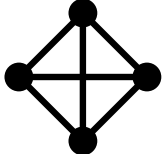


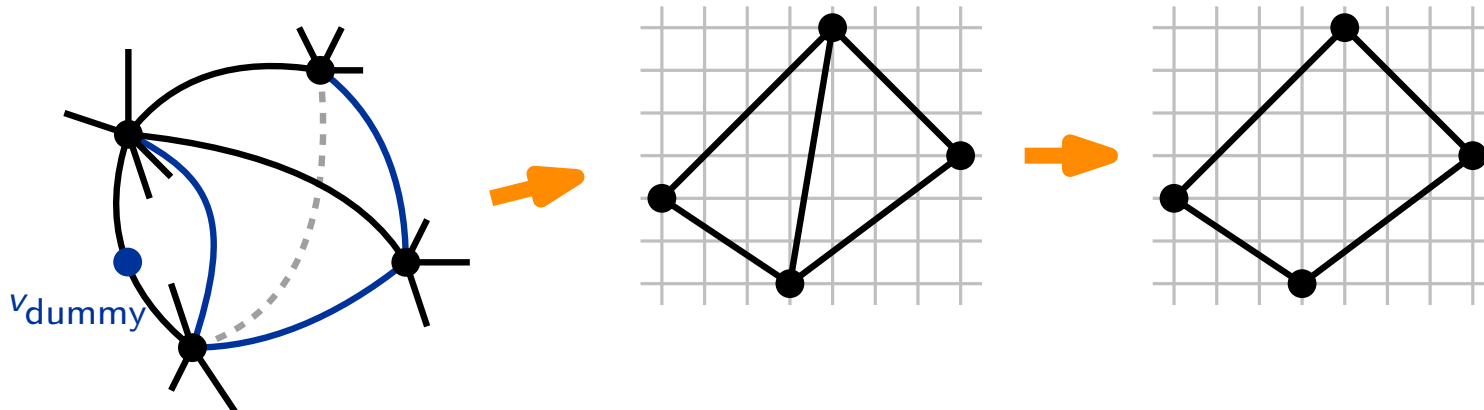
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

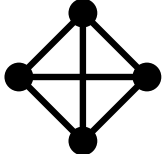


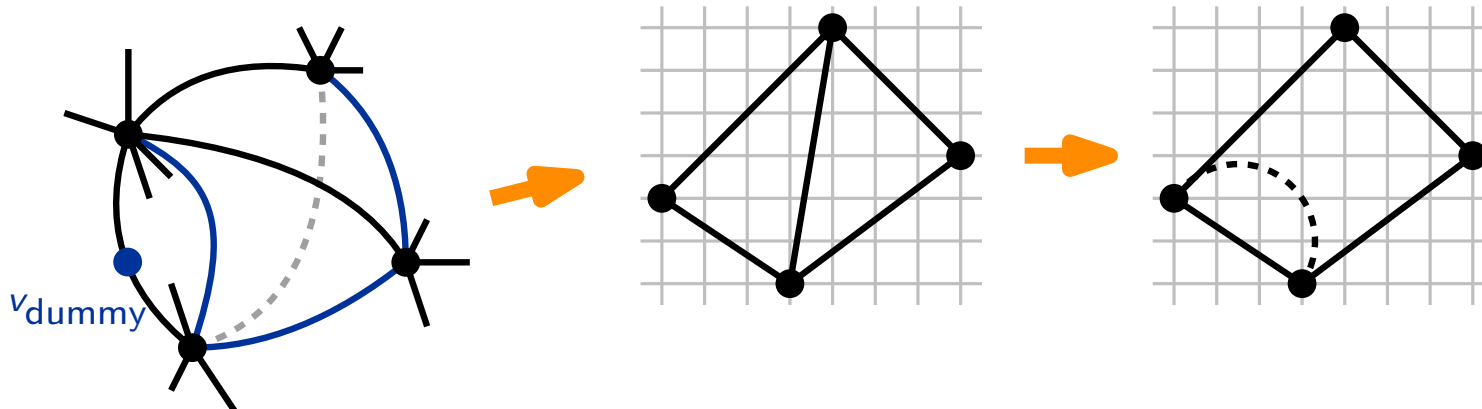
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

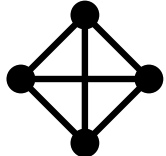


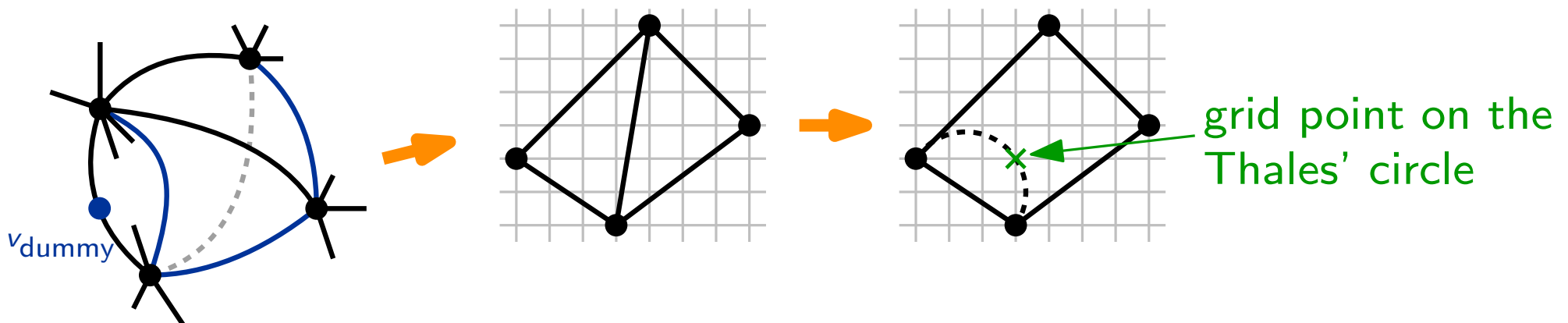
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

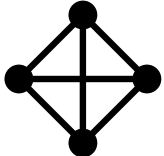


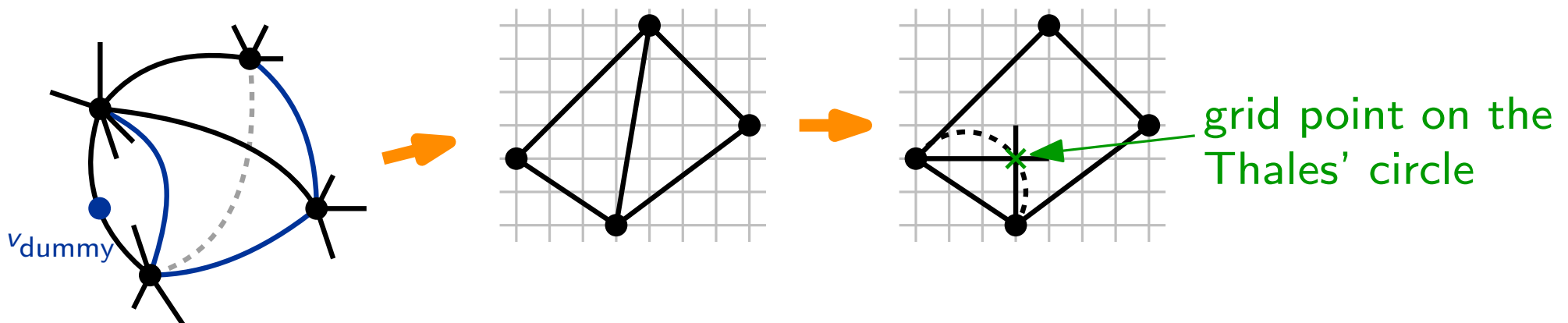
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

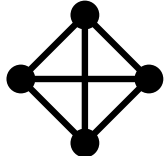


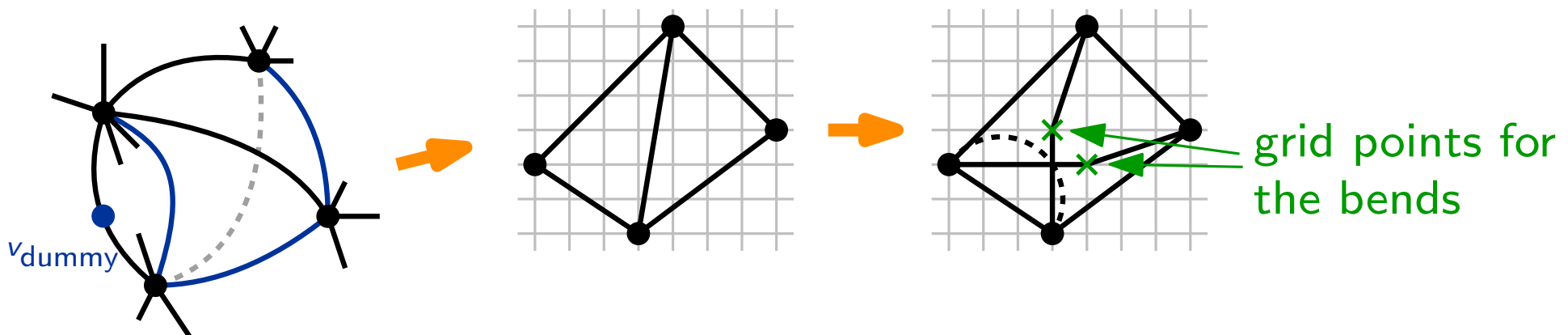
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

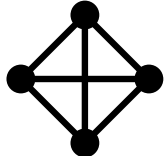


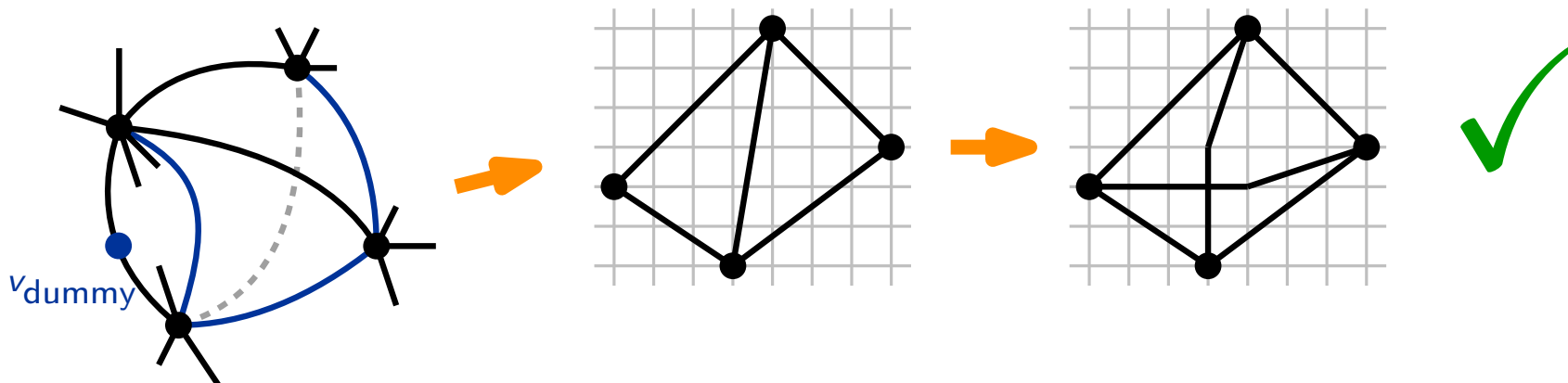
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)



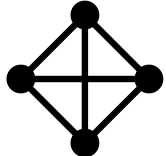


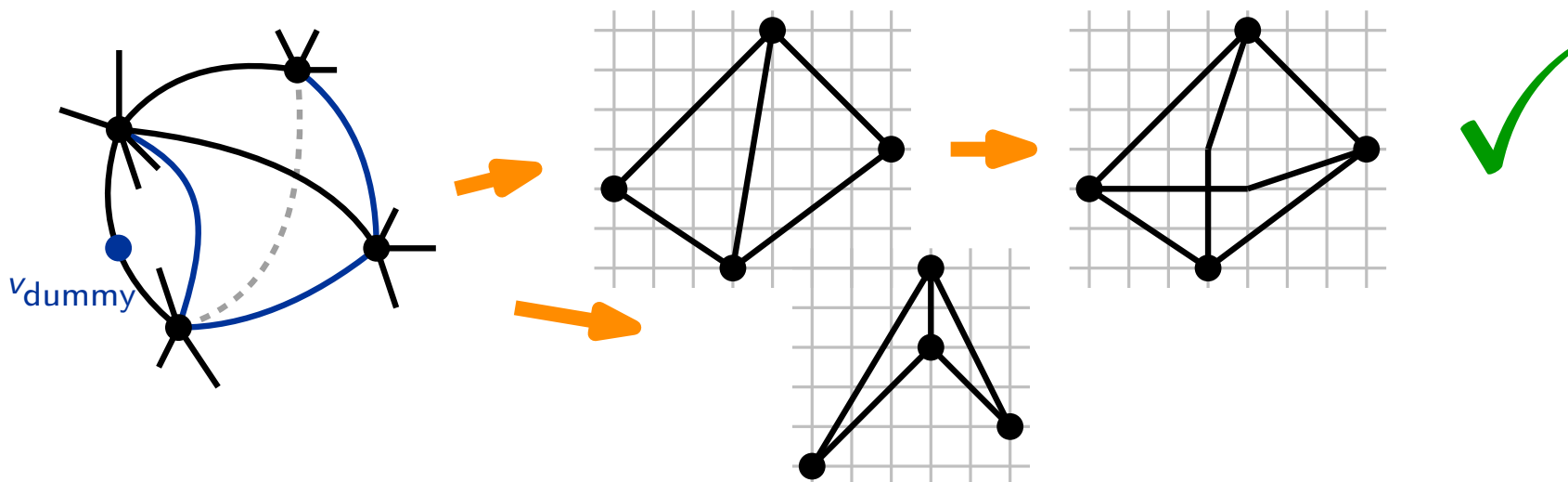
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

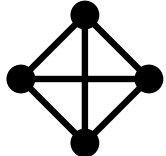


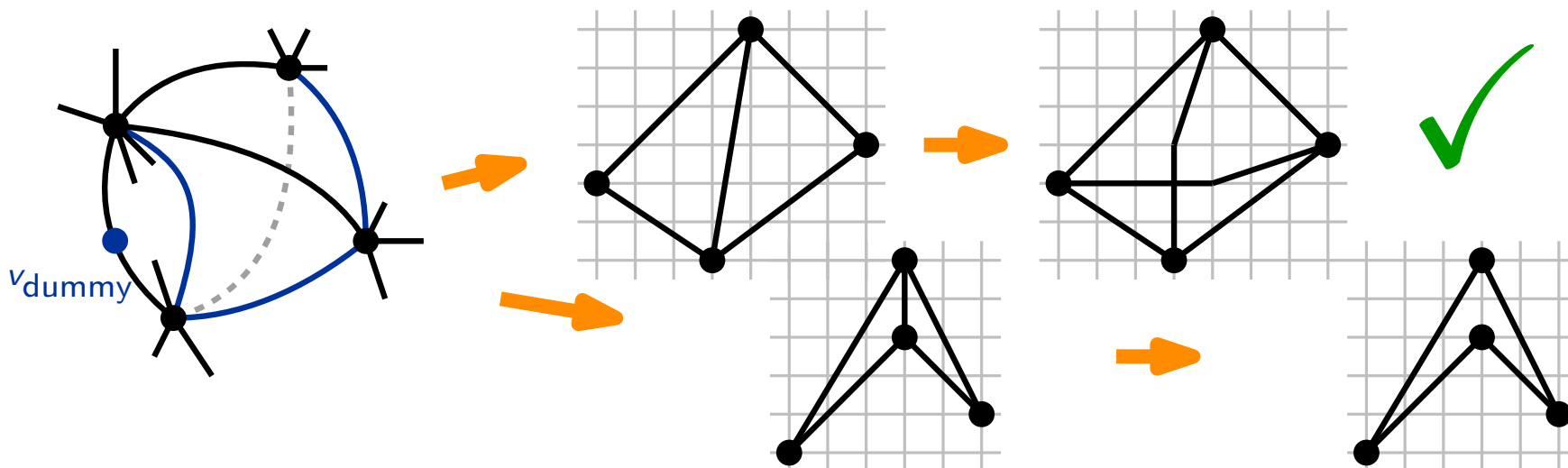
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

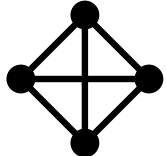


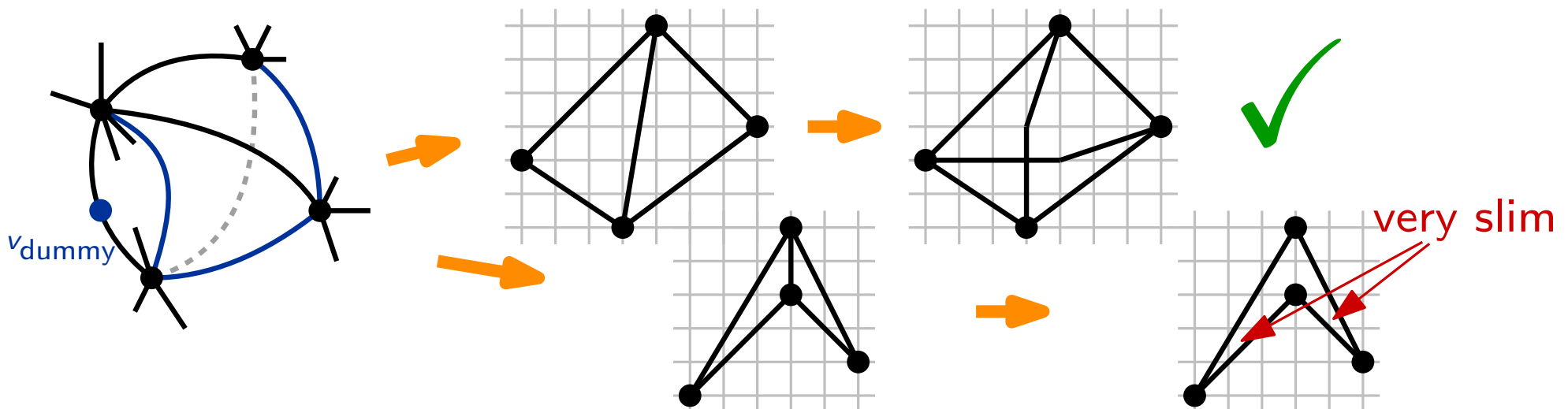
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

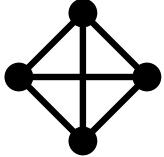


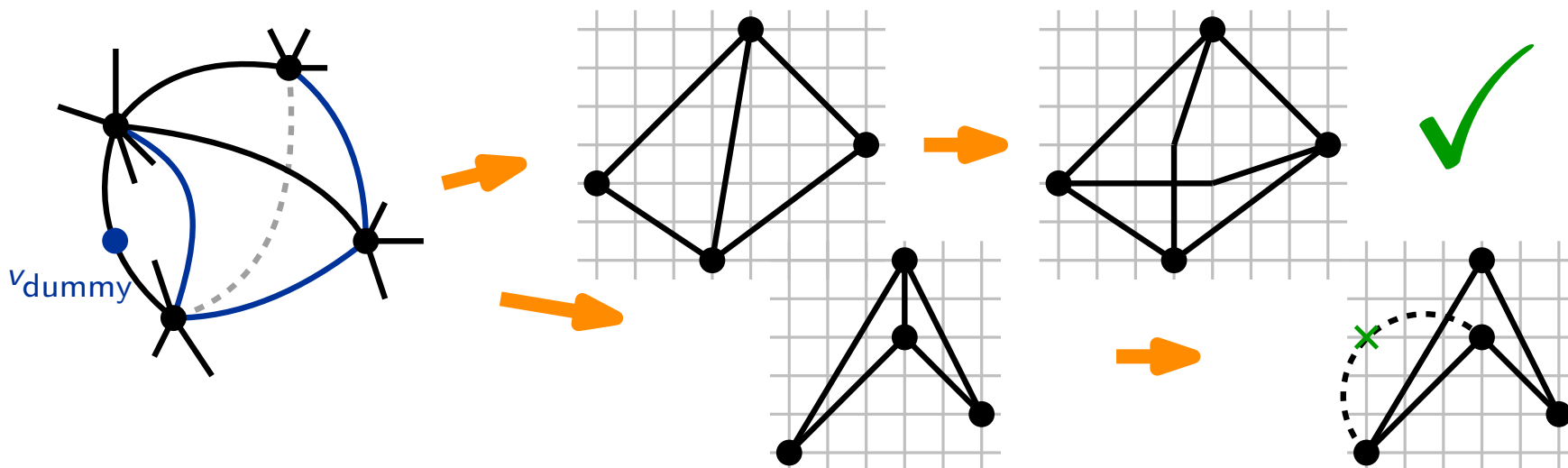
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

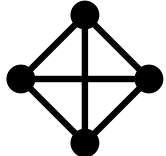


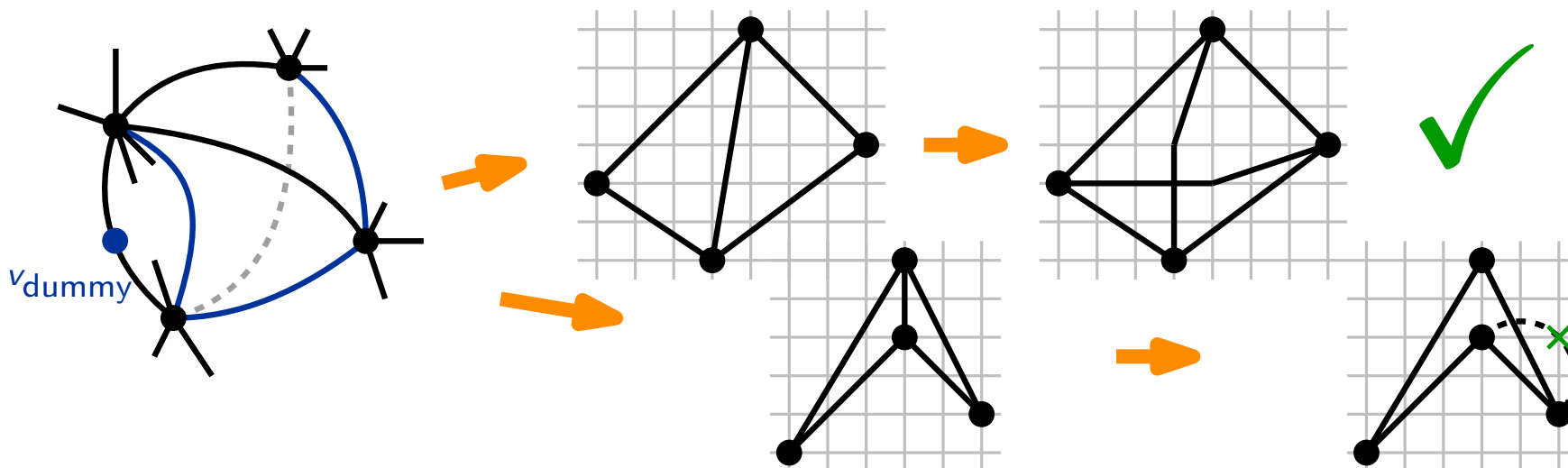
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

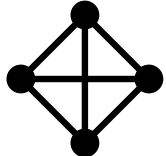


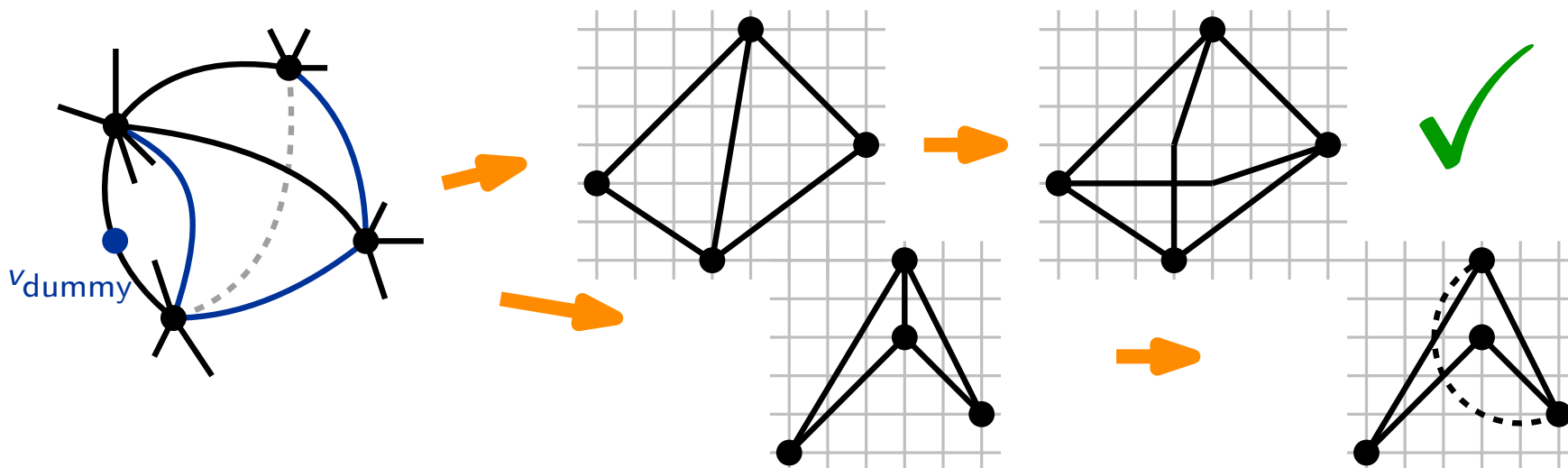
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

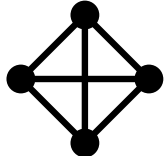


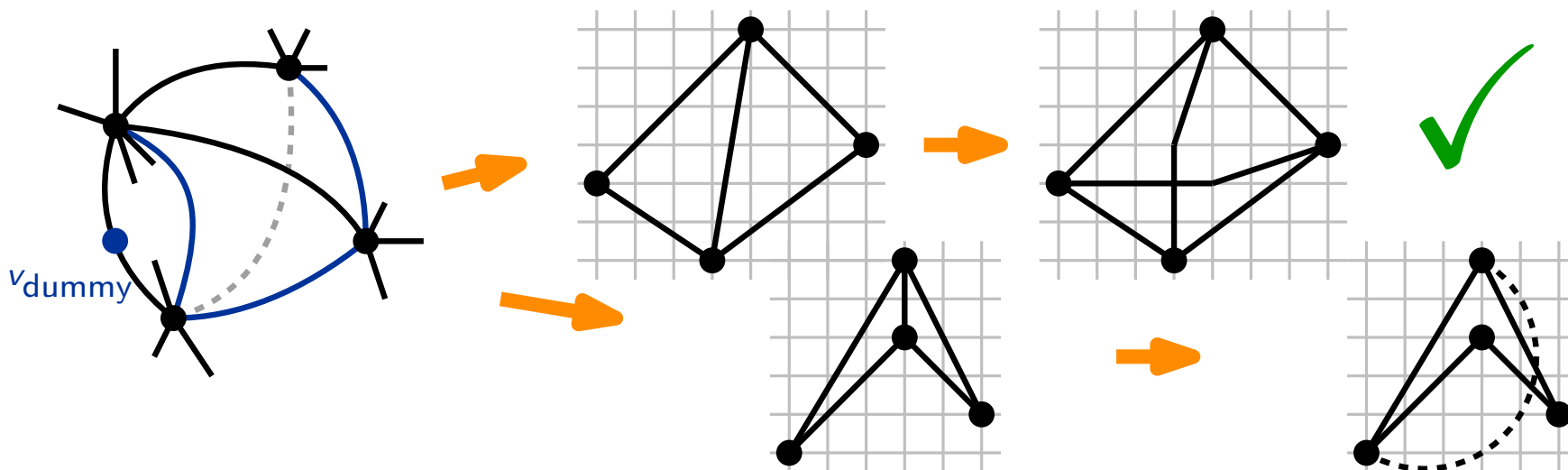
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)

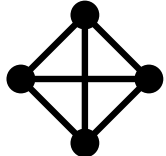


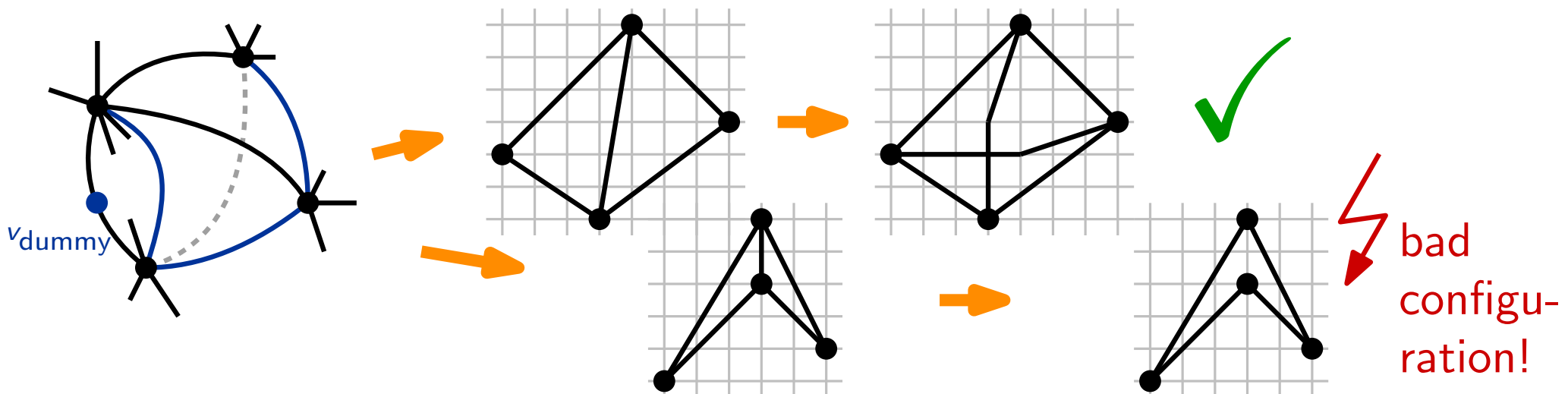
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

6

- Input: a NIC-plane graph

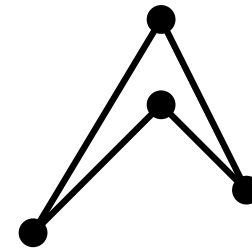
## Approach that nearly works:

- Enclose each crossing by a so called *empty kite*: 
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends on the grid)





Result 1: NIC-Plane Graphs  $\subseteq \text{RAC}_1^{\text{poly}}$



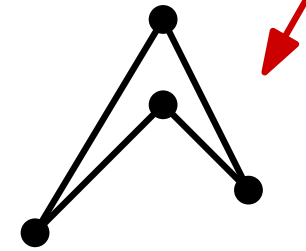
bad  
configu-  
ration!

# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

7

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.



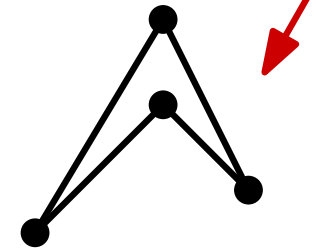
bad  
configu-  
ration!

# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

7

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs).



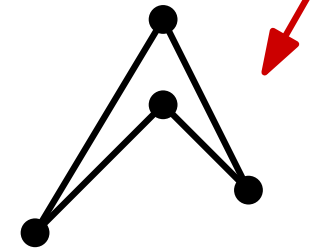
bad  
configu-  
ration!

# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

7

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



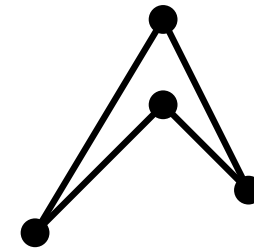
bad  
configu-  
ration!

# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

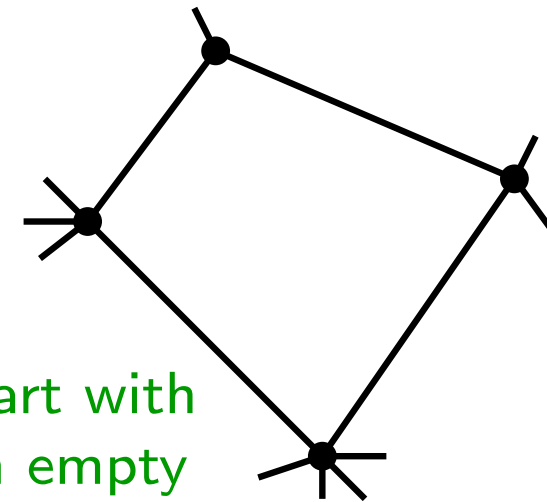
7

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



bad  
configu-  
ration!

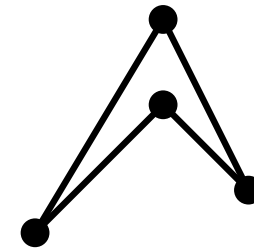


start with  
an empty  
quadrangle

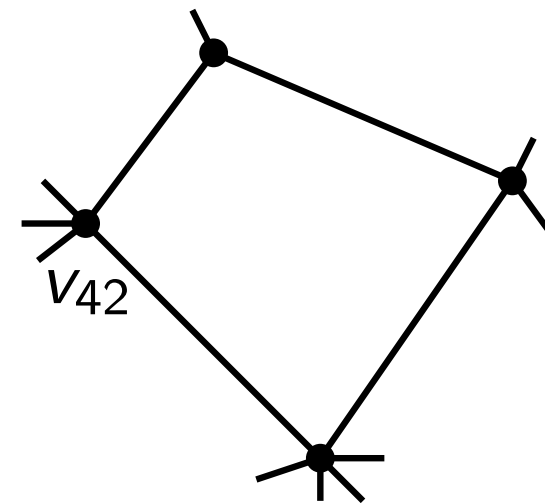
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



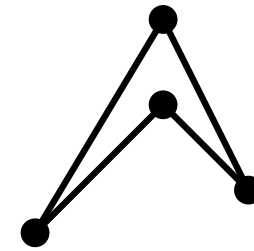
bad  
configu-  
ration!



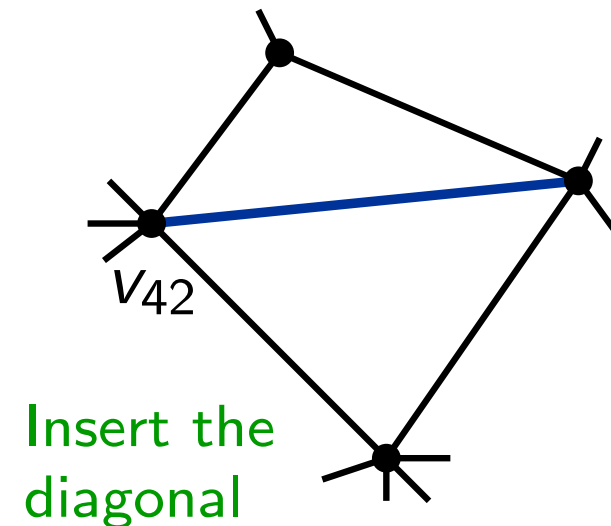
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



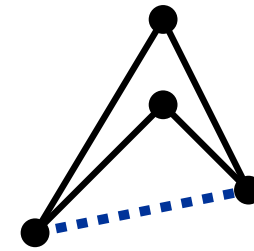
bad  
configu-  
ration!



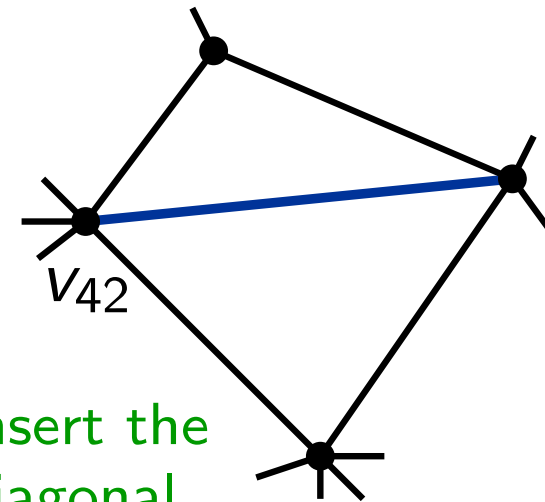
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sadas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



bad  
configu-  
ration!

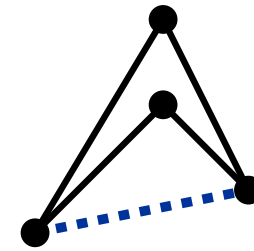




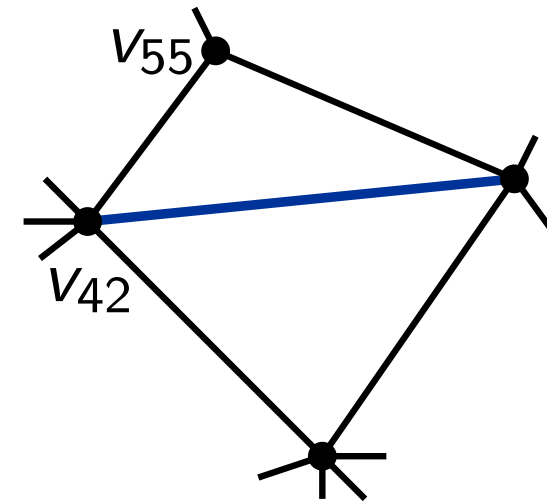
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



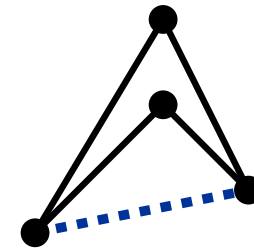
bad  
configu-  
ration!



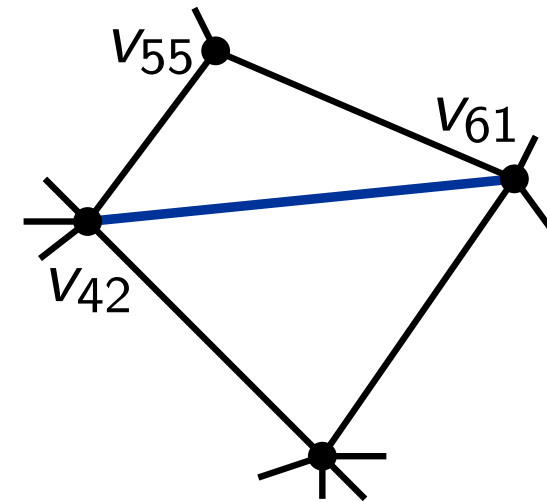
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



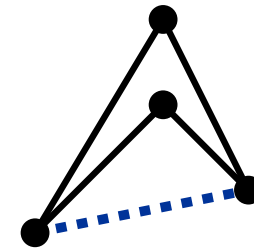
bad  
configu-  
ration!



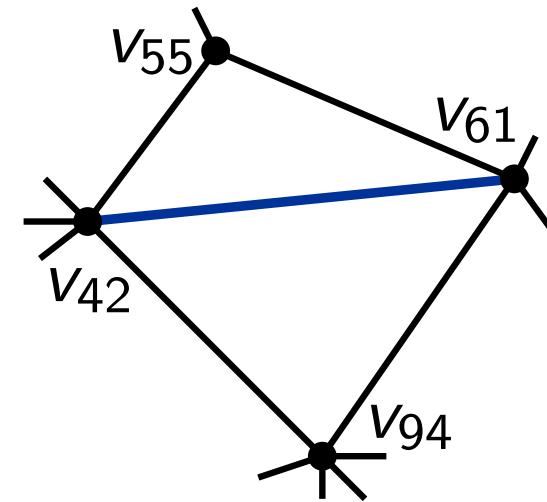
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



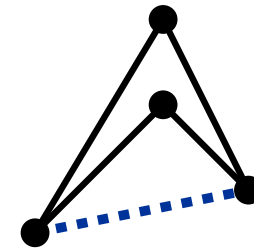
bad  
configu-  
ration!



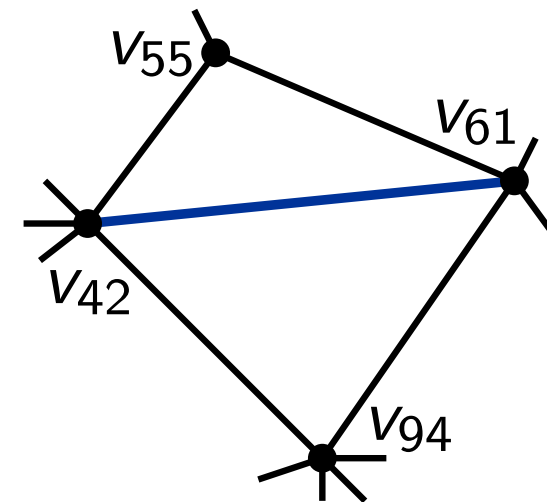
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



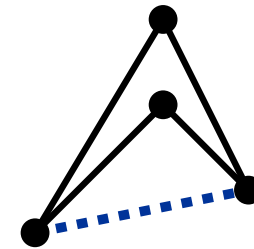
bad  
configu-  
ration!



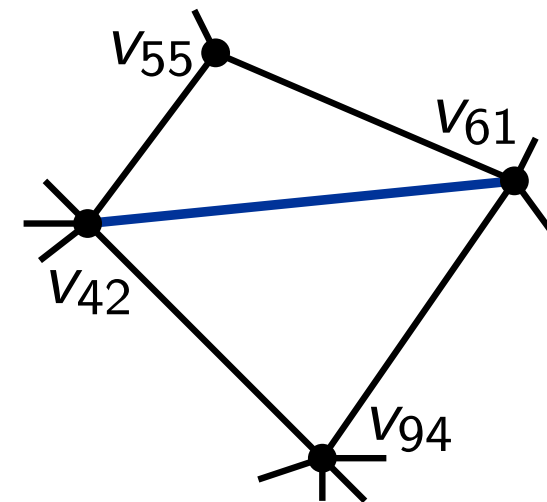
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$ 7

## Solution:

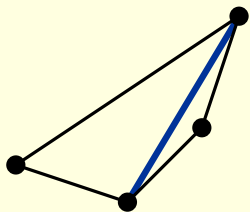
- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



bad  
configu-  
ration!



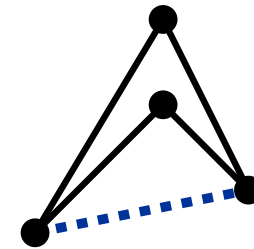
Case 1



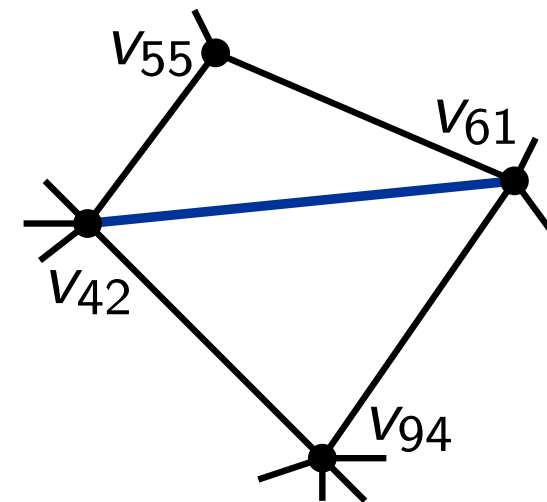
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

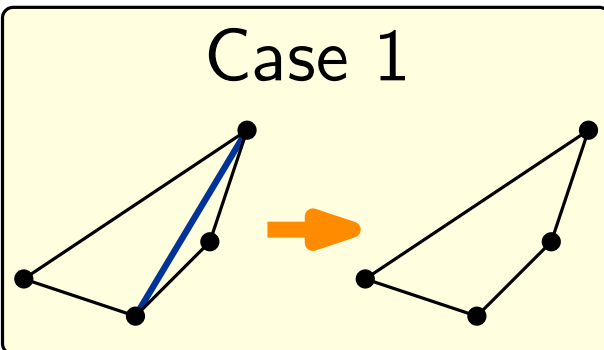
- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



bad  
configu-  
ration!



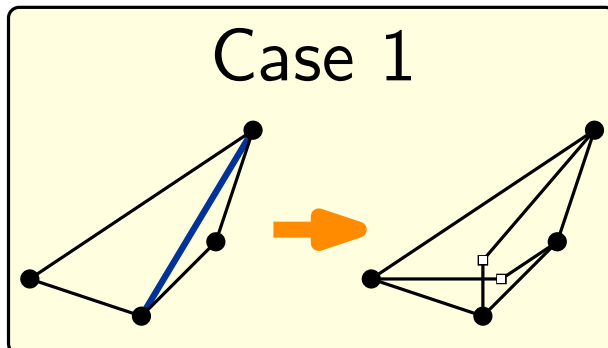
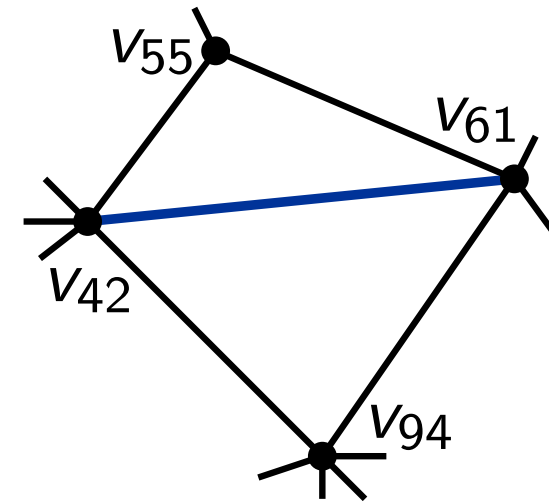
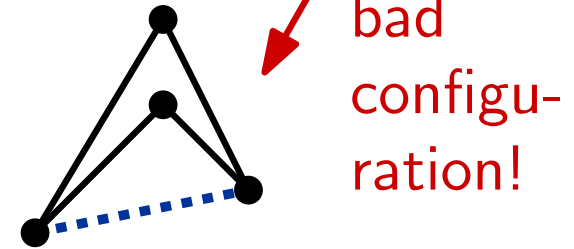
Case 1



# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$ 7

## Solution:

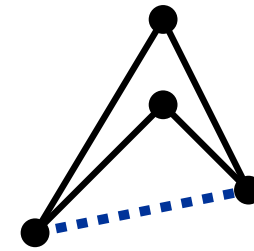
- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



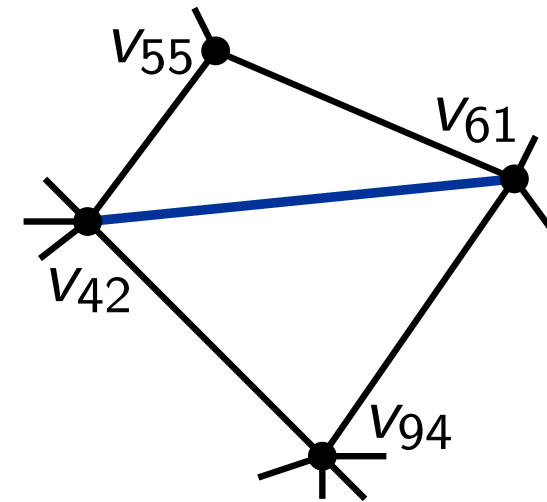
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

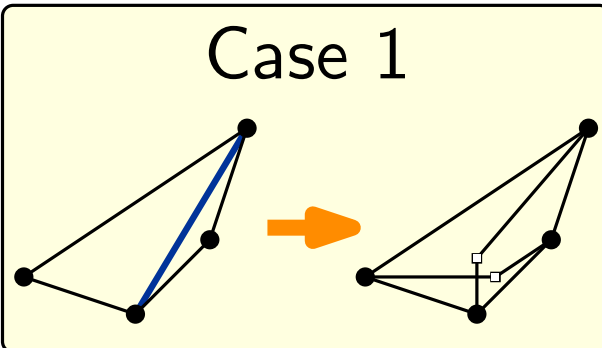
- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sadas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



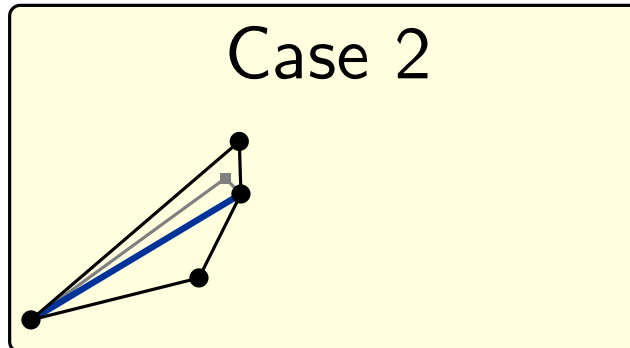
bad  
configu-  
ration!



Case 1



Case 2

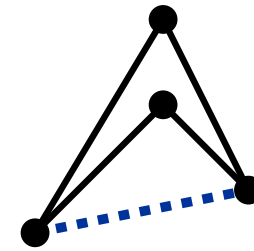




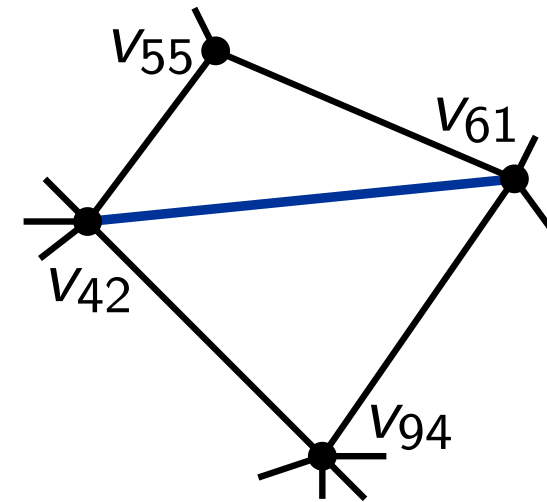
# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

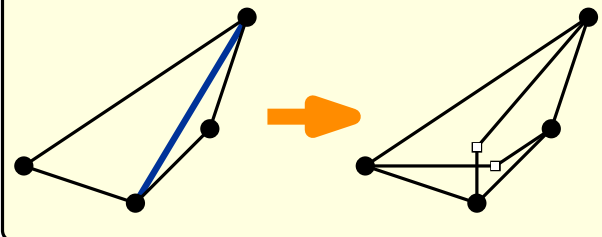
- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sadas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



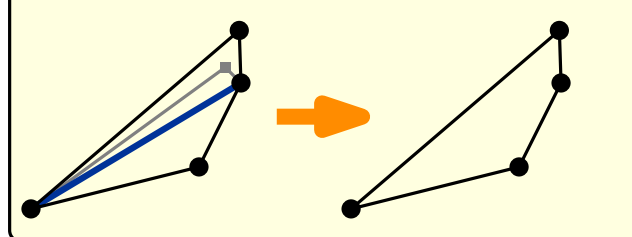
bad  
configu-  
ration!



Case 1



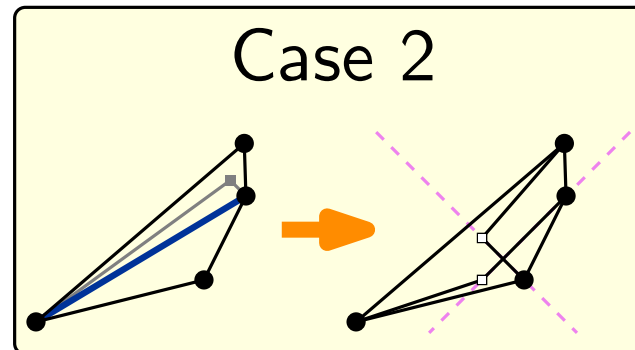
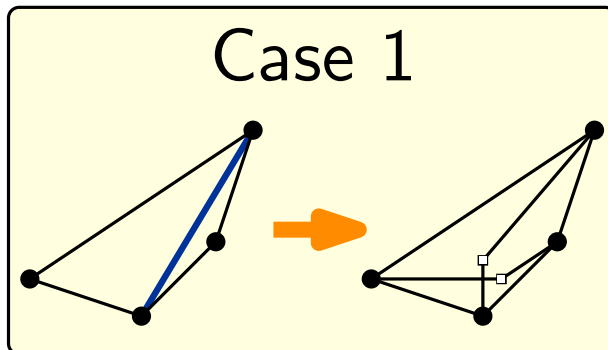
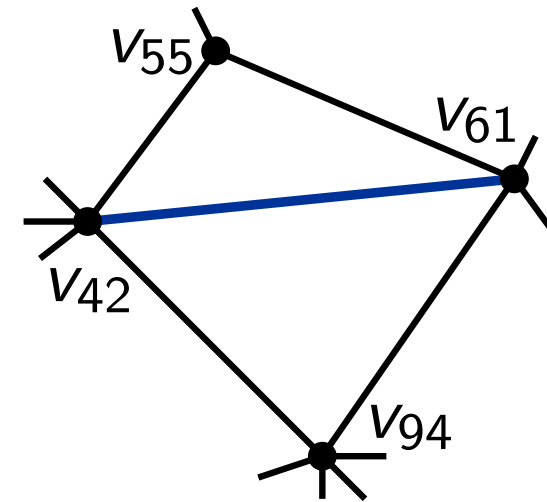
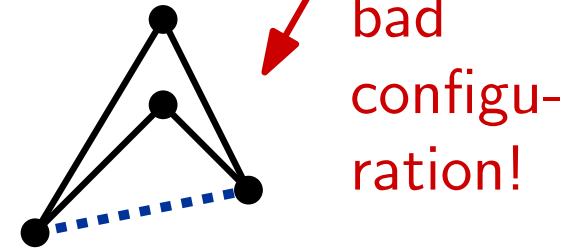
Case 2



# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:

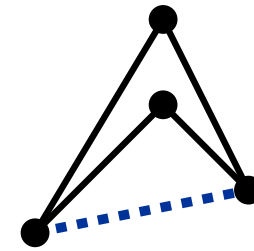


# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

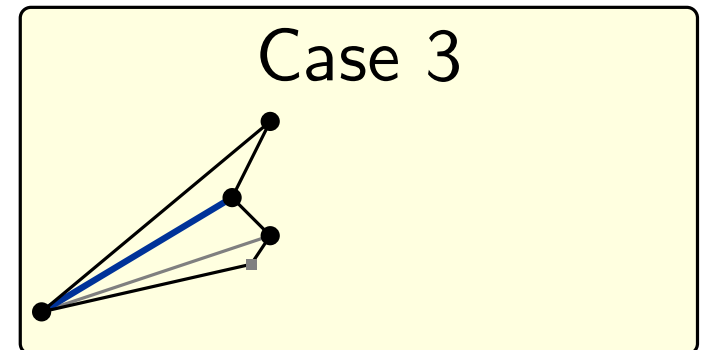
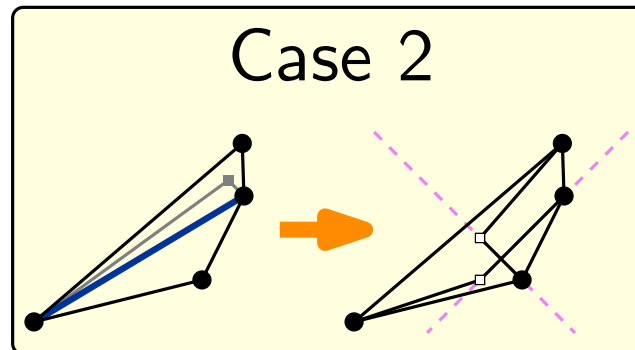
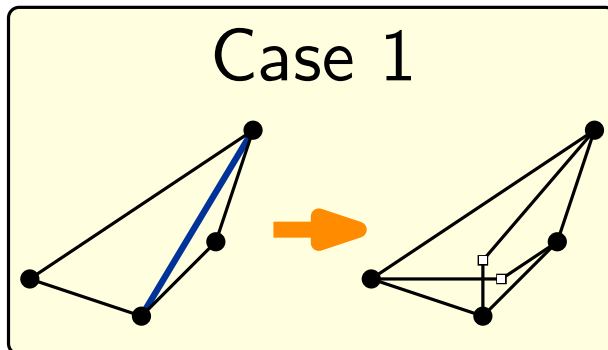
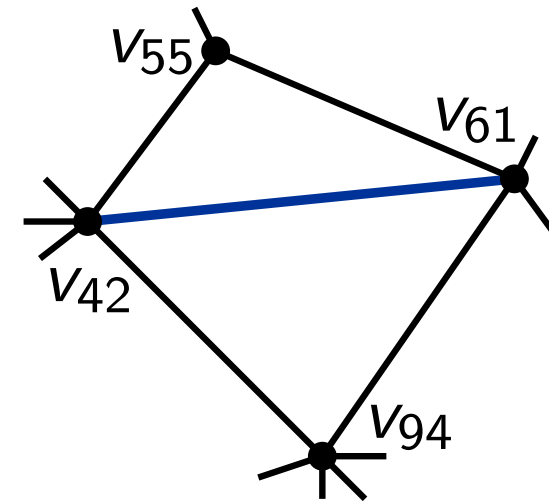
7

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



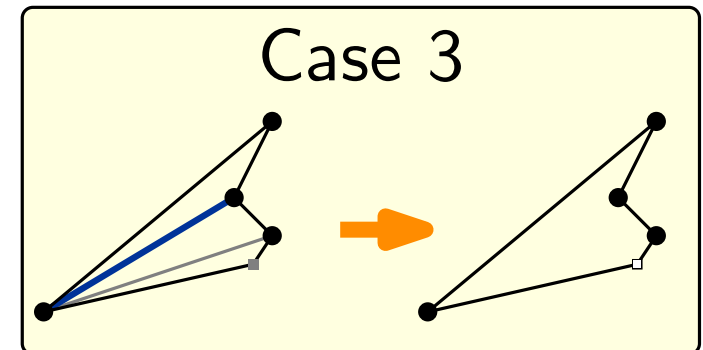
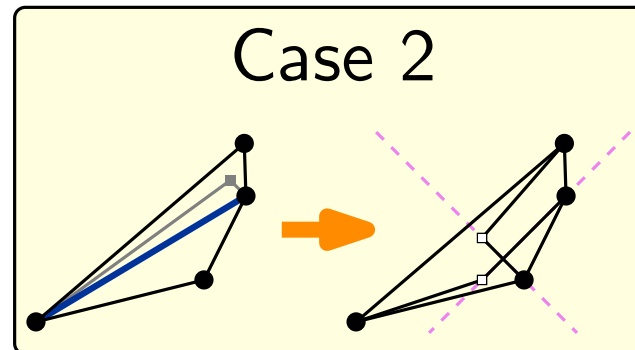
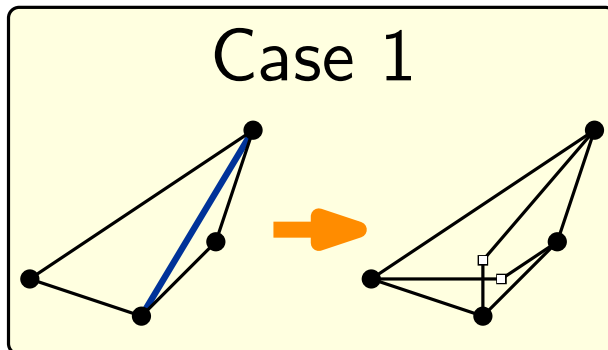
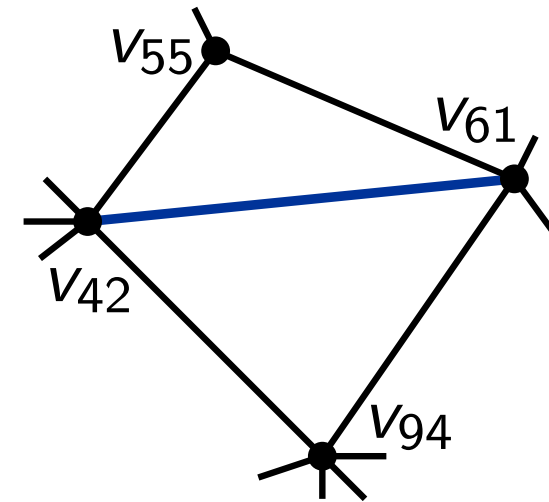
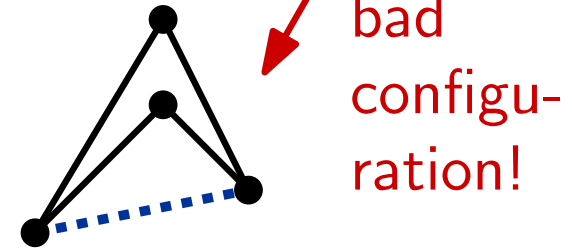
bad  
configu-  
ration!



# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

## Solution:

- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:

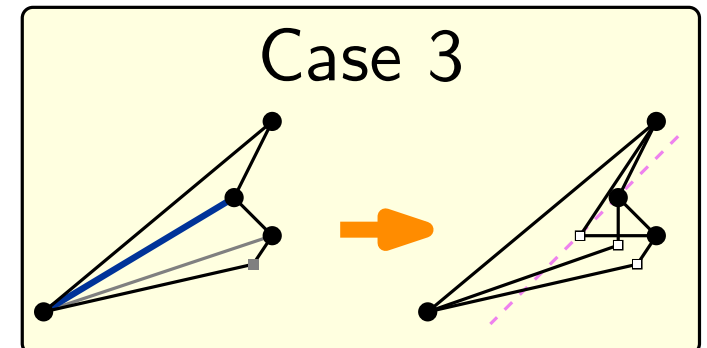
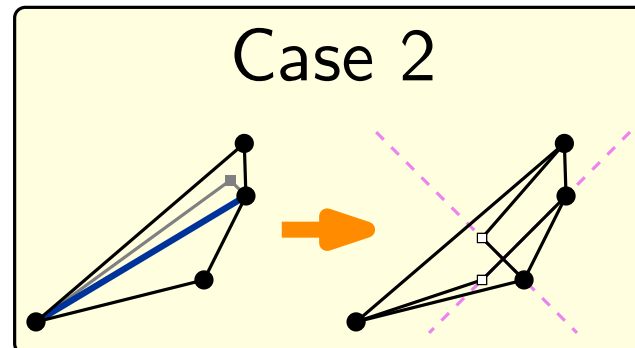
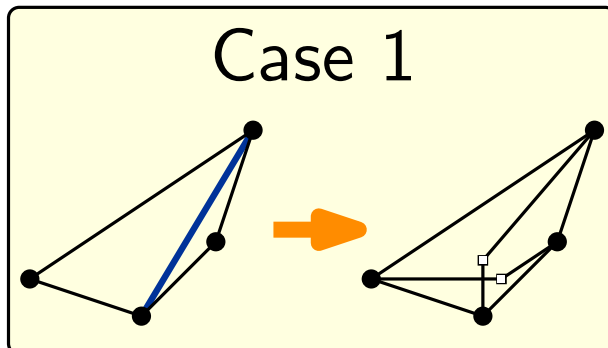
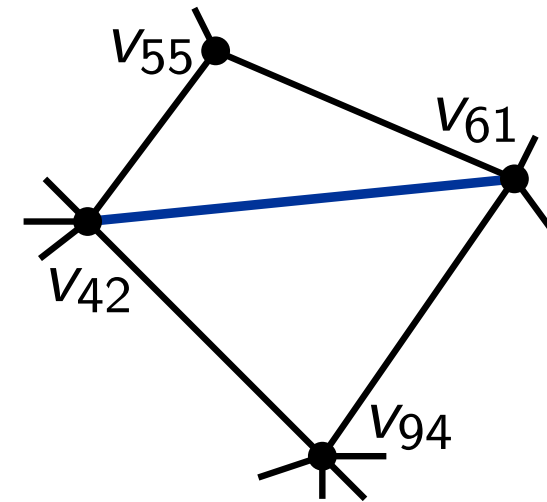
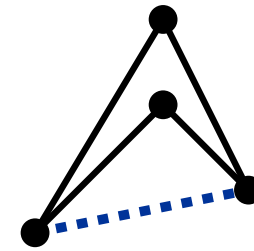


# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

7

## Solution:

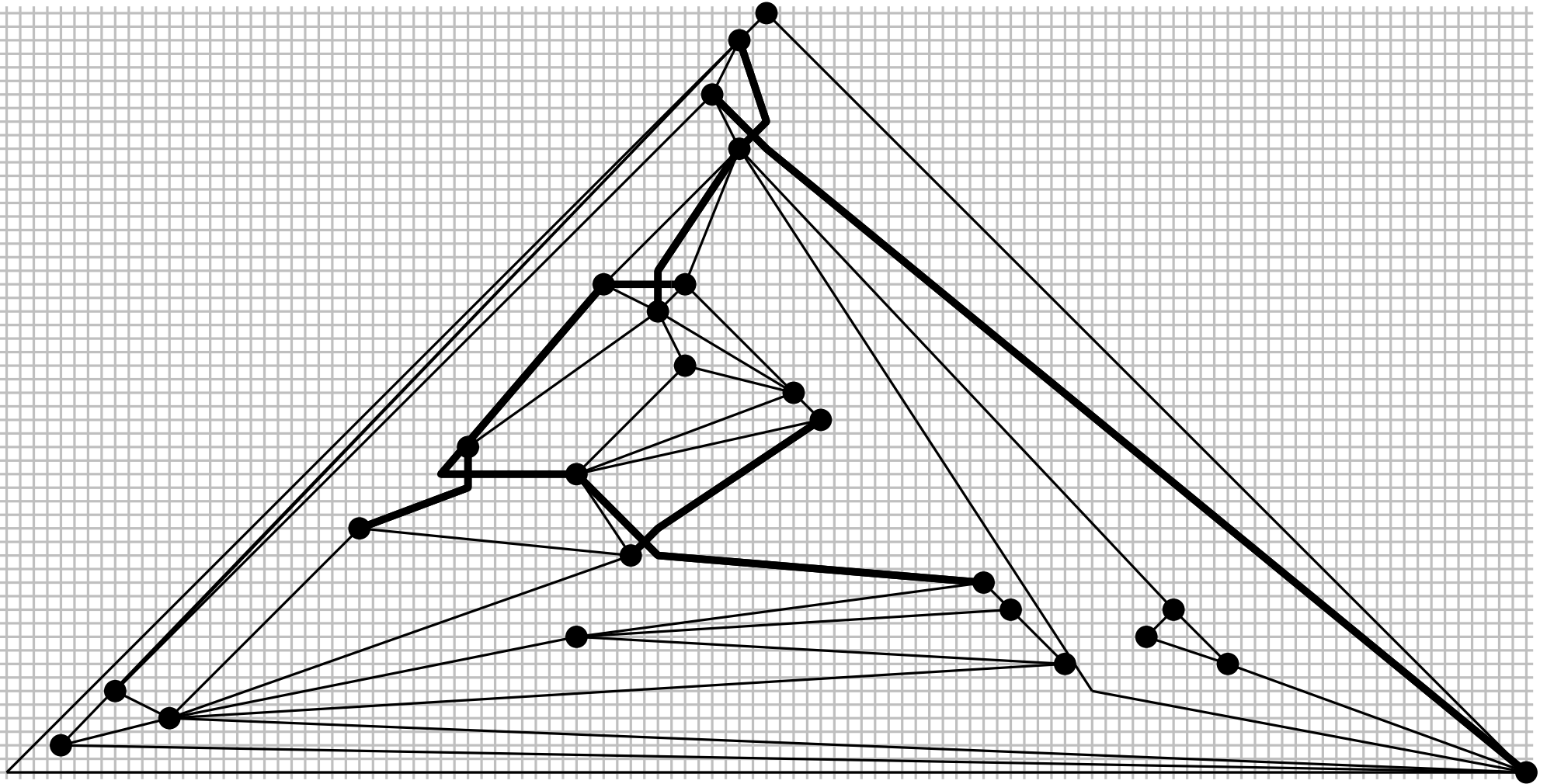
- Make the first vertex in the quadrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sadas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

8

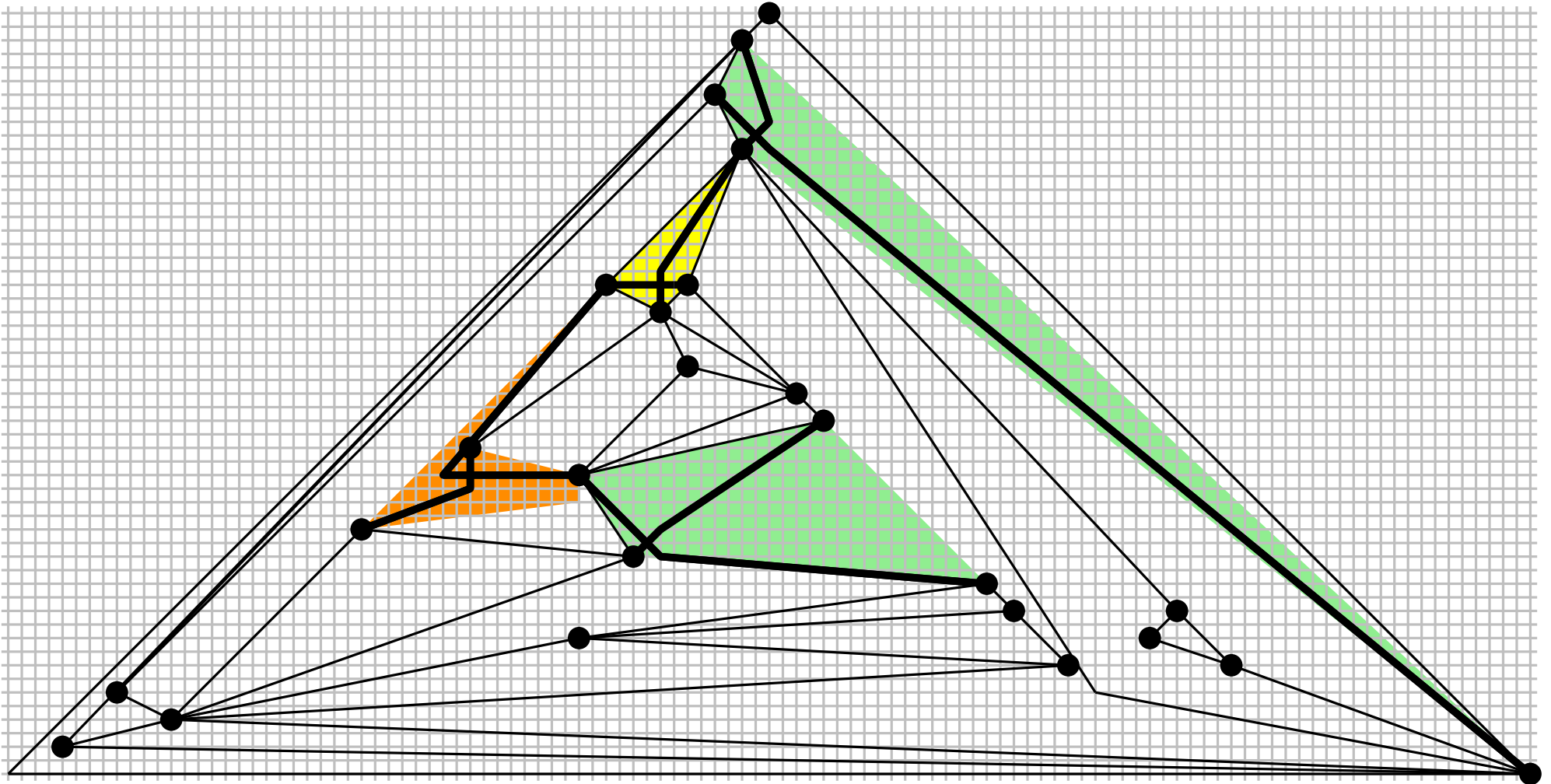
**Full example:**



# Result 1: NIC-Plane Graphs $\subseteq \text{RAC}_1^{\text{poly}}$

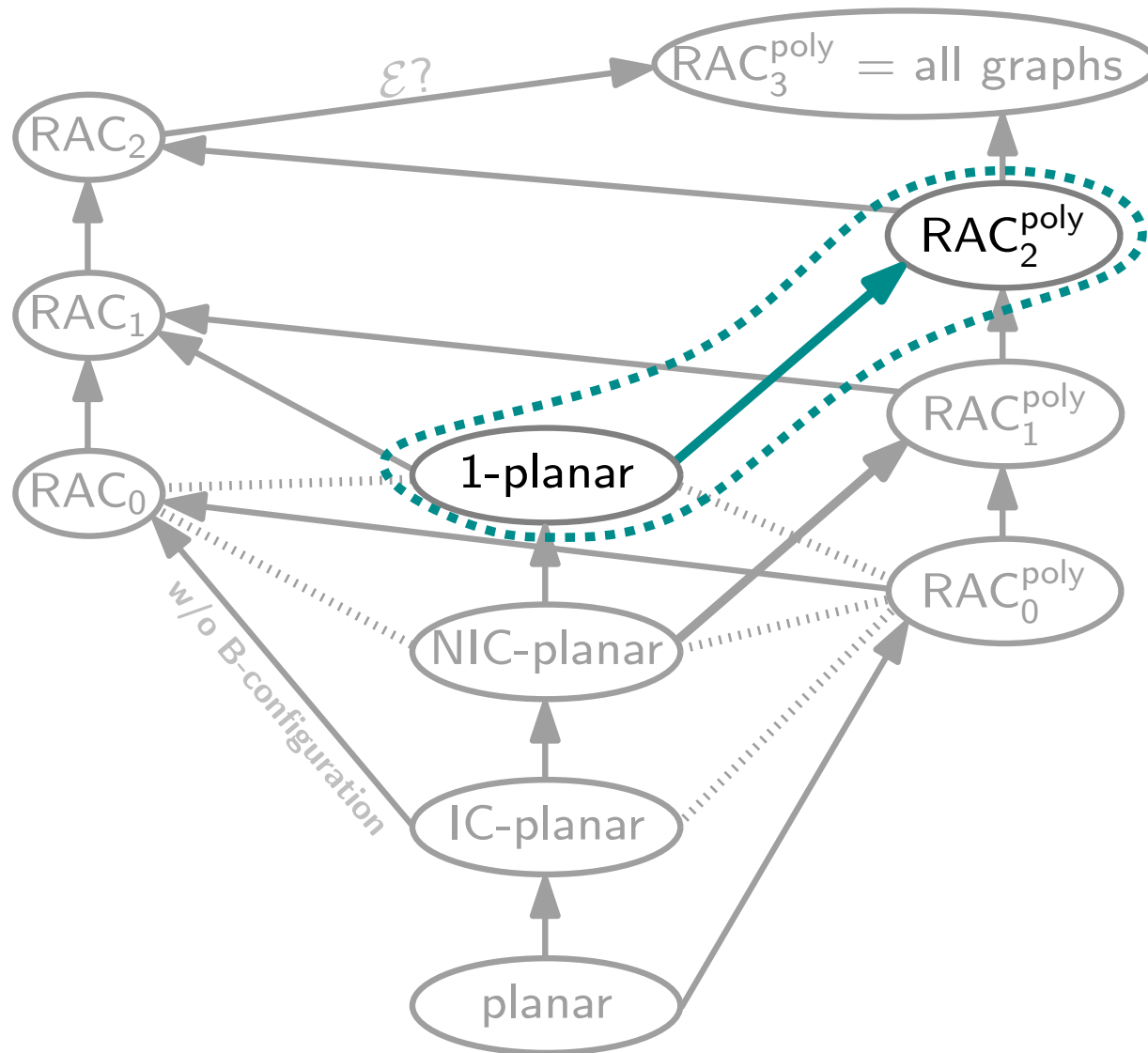
8

Full example:



# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

9





# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

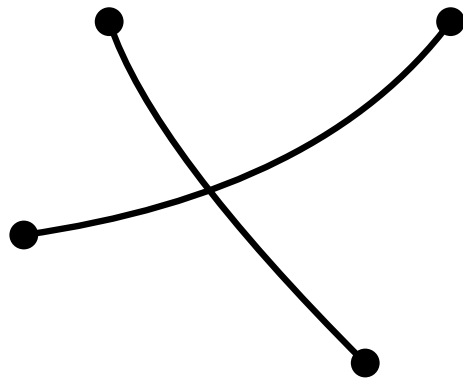
**Preprocessing:**

# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

**Preprocessing:**



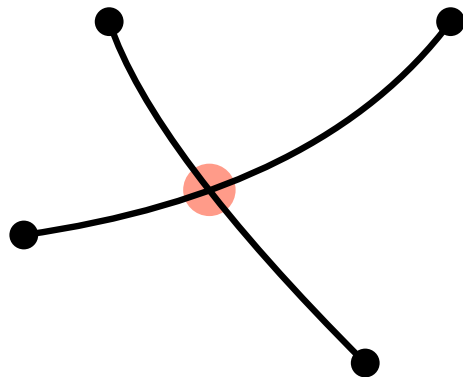
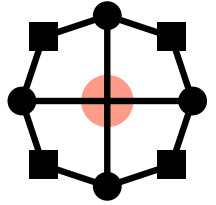
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

## Preprocessing:

- Enclose each **crossing** by a so called *subdivided kite*:



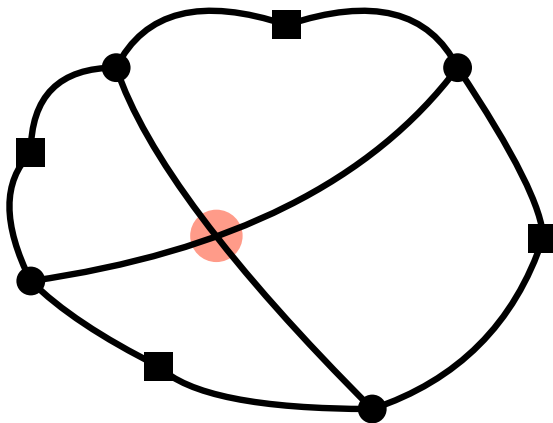
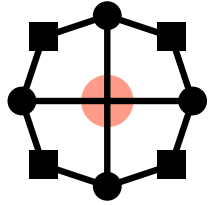
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

## Preprocessing:

- Enclose each **crossing** by a so called *subdivided kite*:



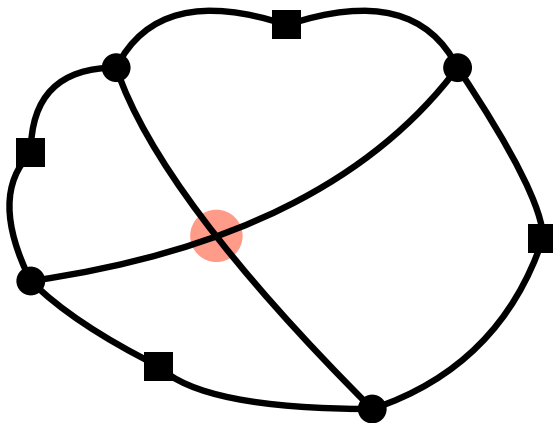
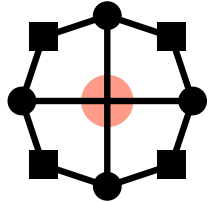
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

## Preprocessing:

- Enclose each **crossing** by a so called *subdivided kite*:
- Planarize the graph by replacing each crossing by a vertex



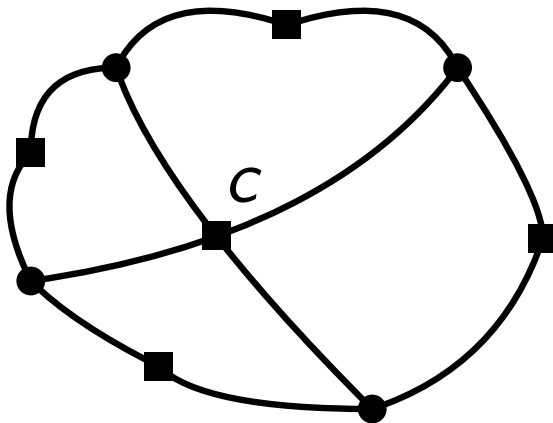
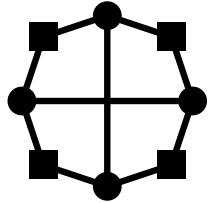
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

## Preprocessing:

- Enclose each crossing by a so called *subdivided kite*:
- Planarize the graph by replacing each crossing by a vertex



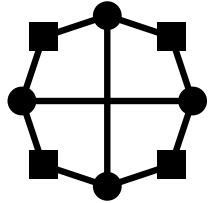
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

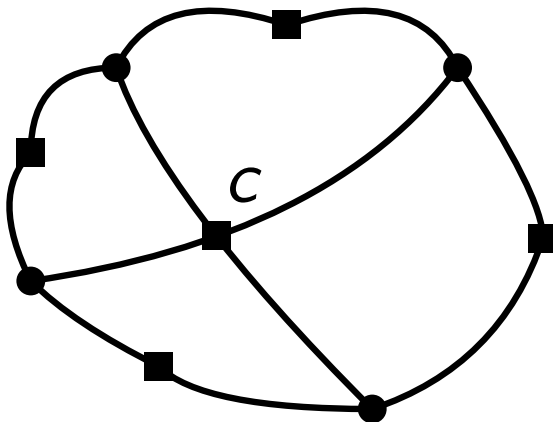
## Preprocessing:

- Enclose each crossing by a so called *subdivided kite*:
- Planarize the graph by replacing each crossing by a vertex



## Drawing phase:

- Draw the obtained plane graph using the Shift Algorithm





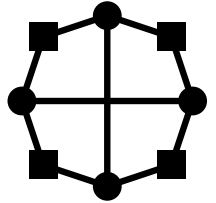
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

10

- Input: a 1-plane graph

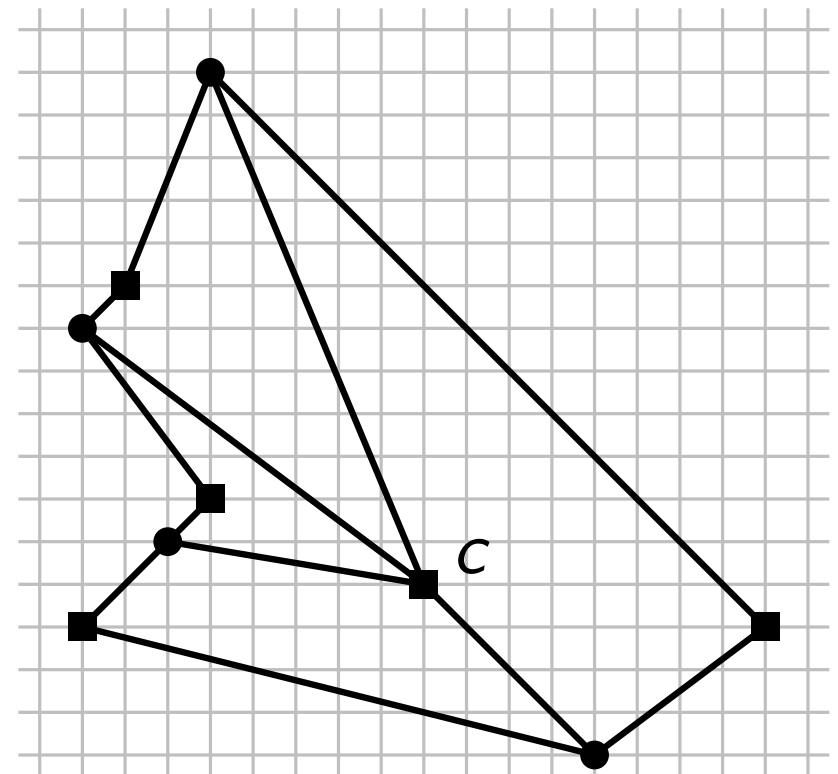
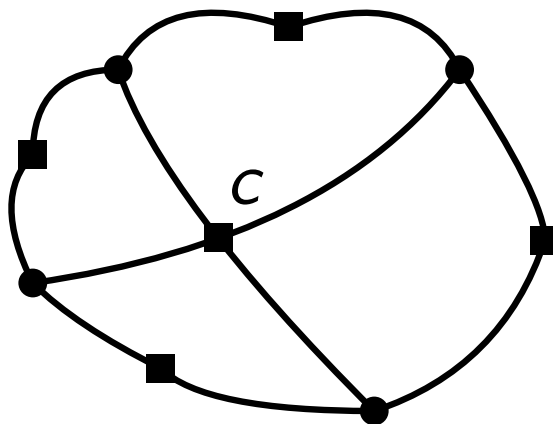
## Preprocessing:

- Enclose each crossing by a so called *subdivided kite*:
- Planarize the graph by replacing each crossing by a vertex



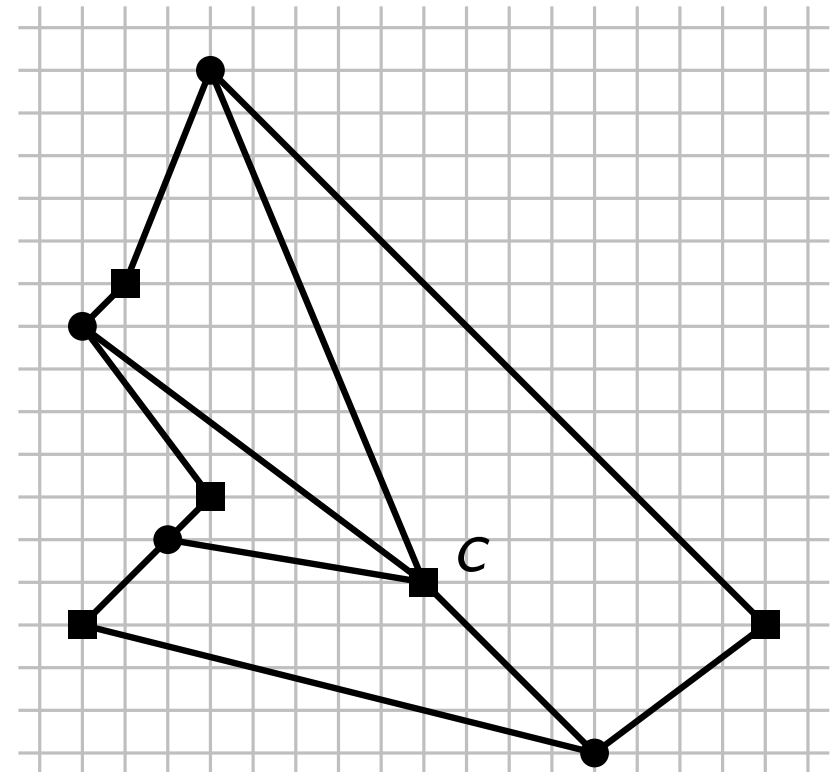
## Drawing phase:

- Draw the obtained plane graph using the Shift Algorithm



# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

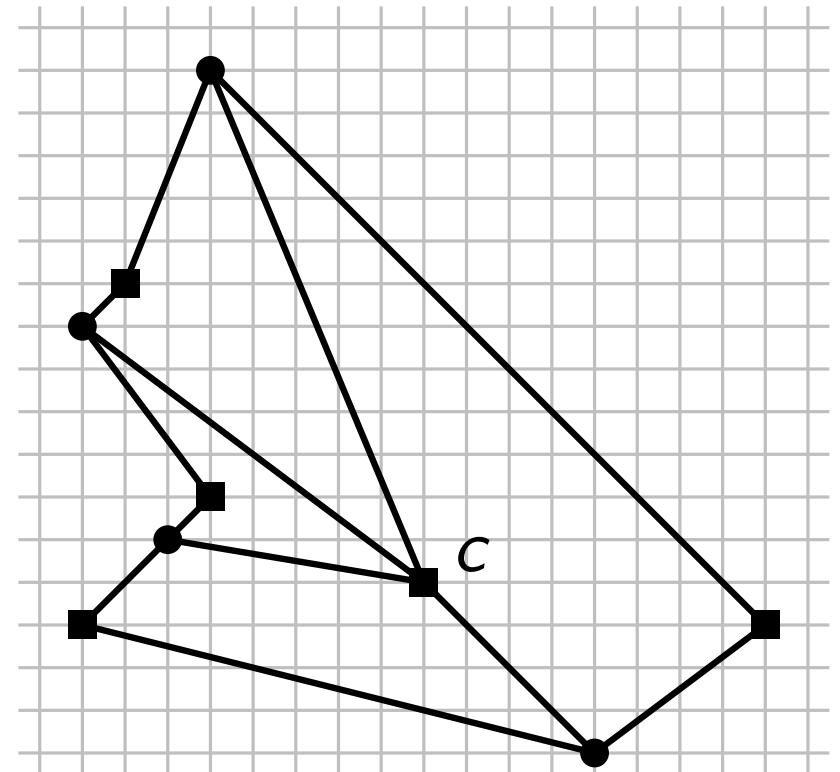
11



# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

**Postprocessing (obtaining crossings at right angles):**

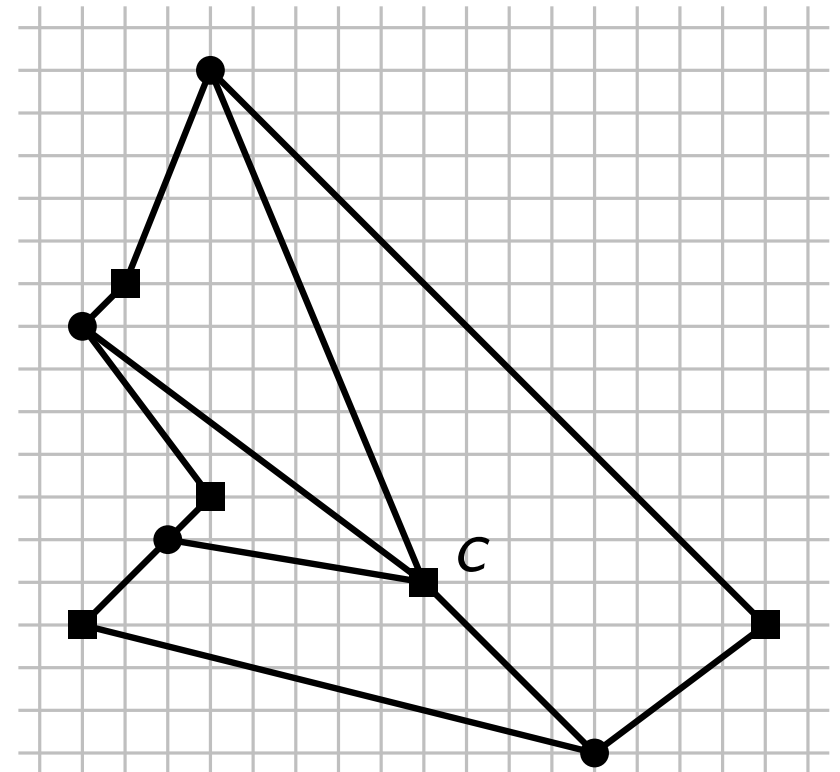


## Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

**Postprocessing (obtaining crossings at right angles):**

- Consider the four axis-parallel half-lines originating at  $c$

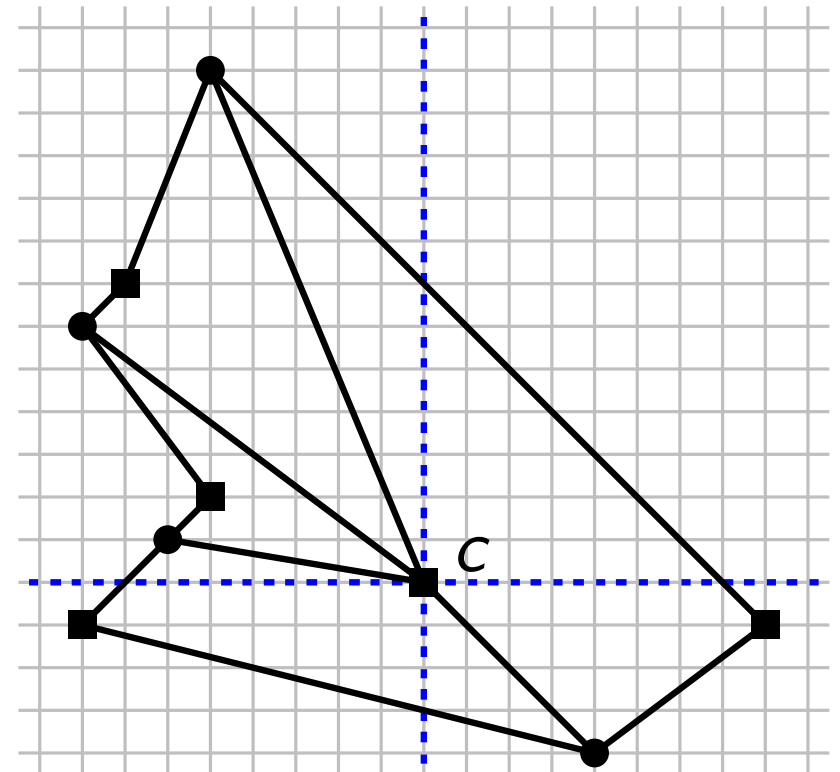


## Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

**Postprocessing (obtaining crossings at right angles):**

- Consider the four axis-parallel half-lines originating at  $c$

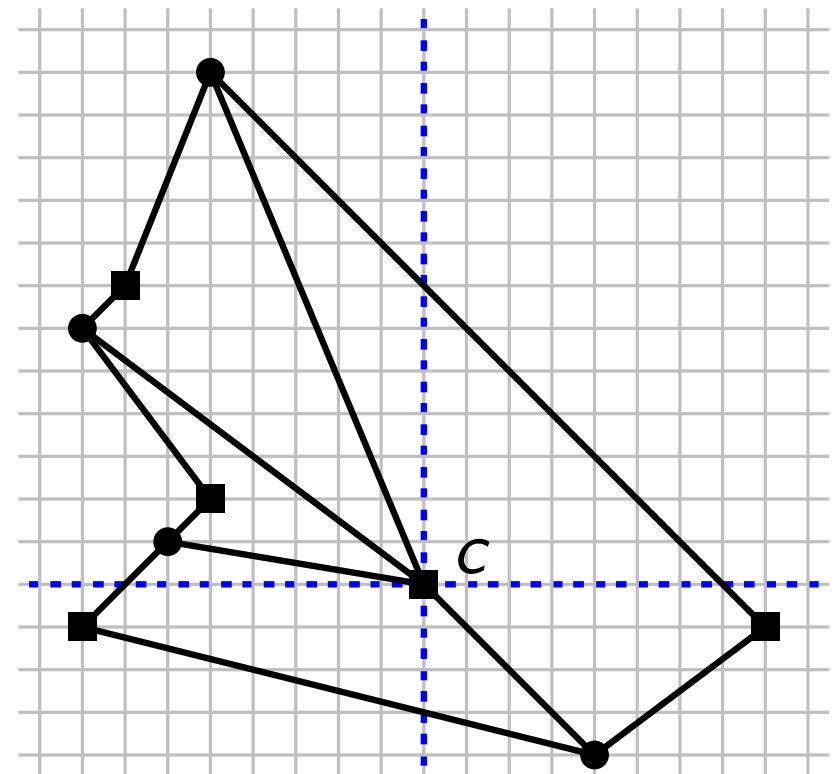


## Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

### Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines

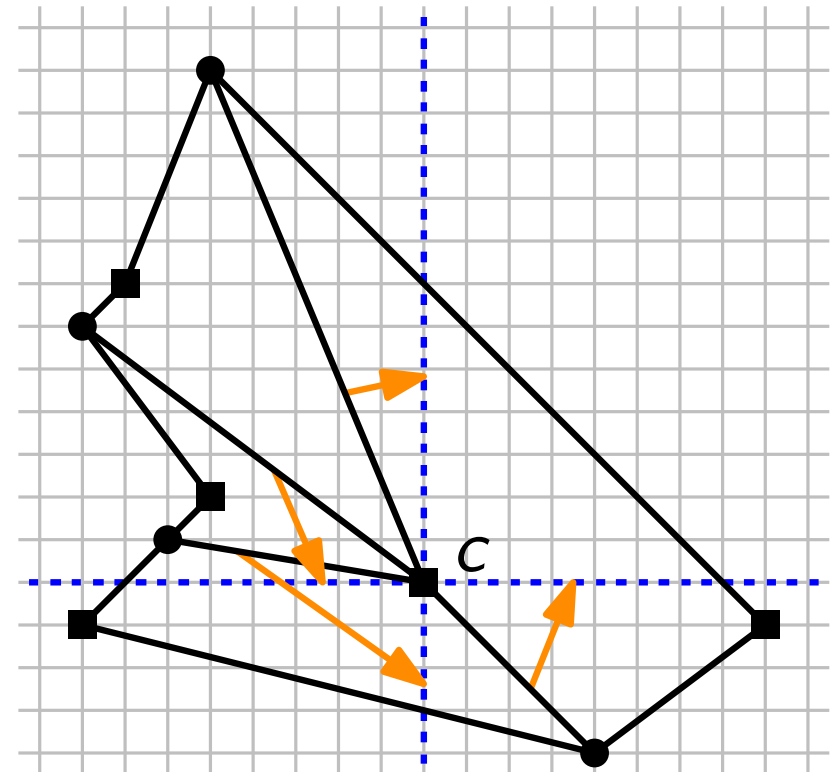


## Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

### Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines

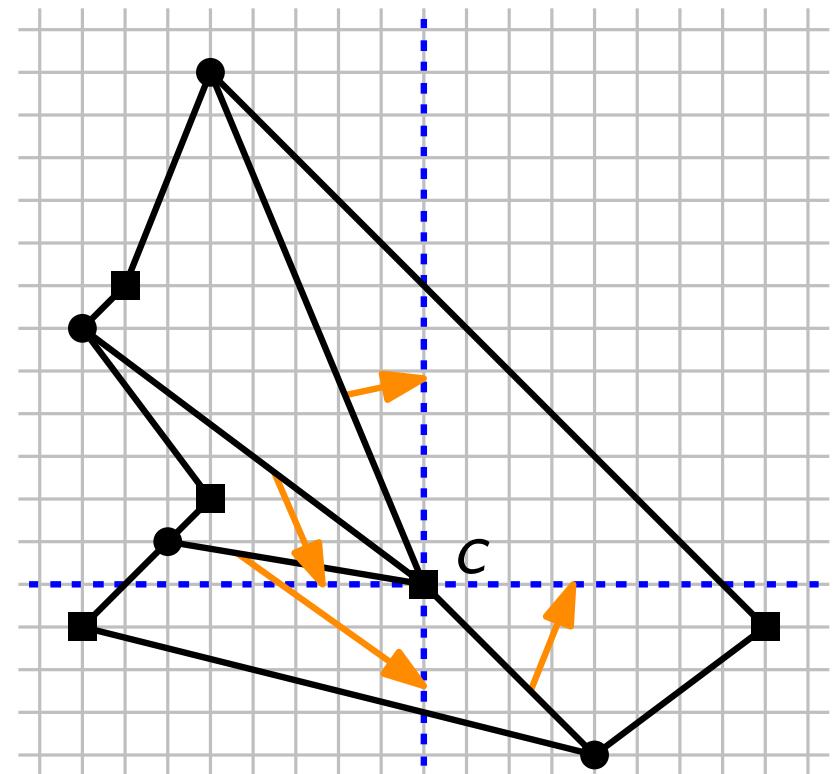


## Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

### Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:



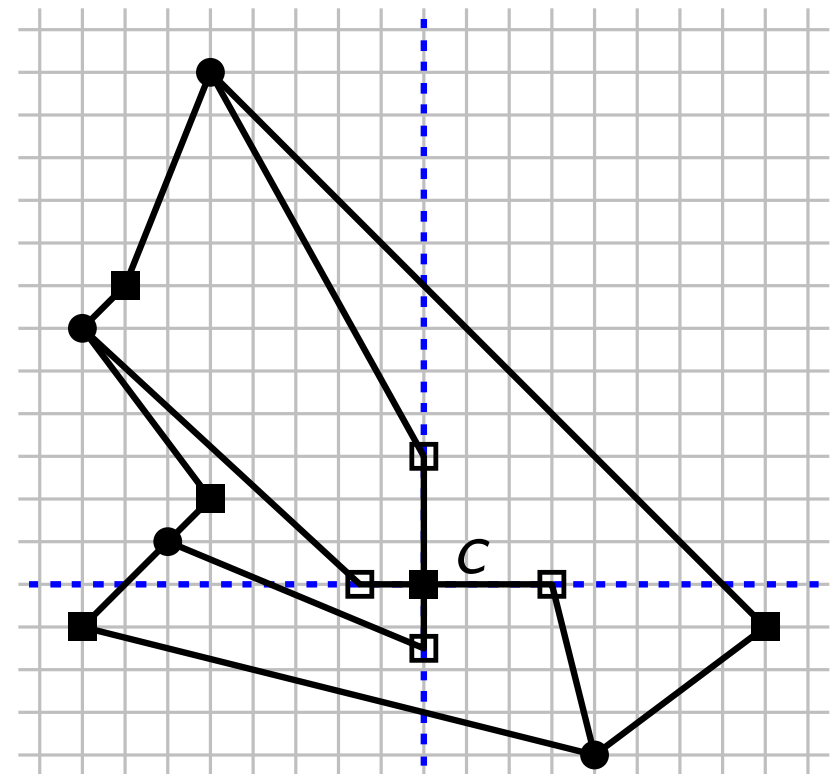


## Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

### Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:

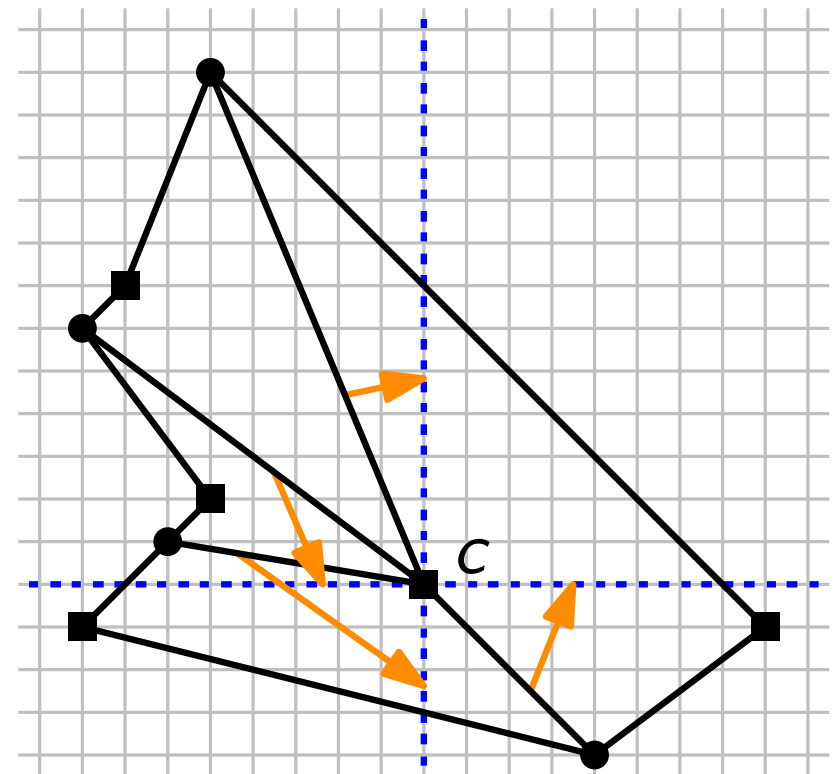


## Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

### Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:



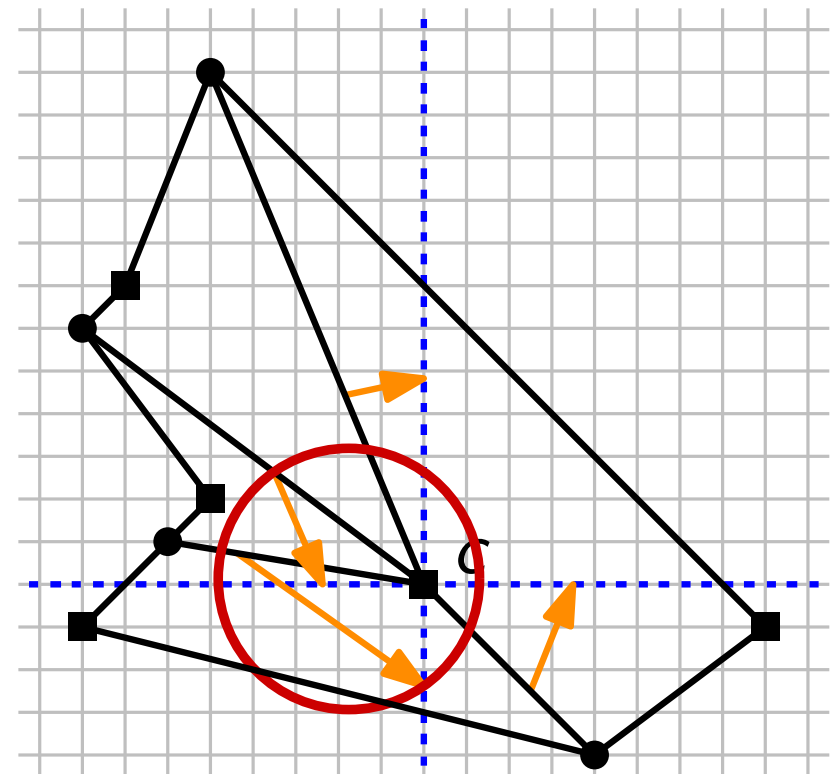
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:

Be careful:  
One assignment  
might depend on  
another one



# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

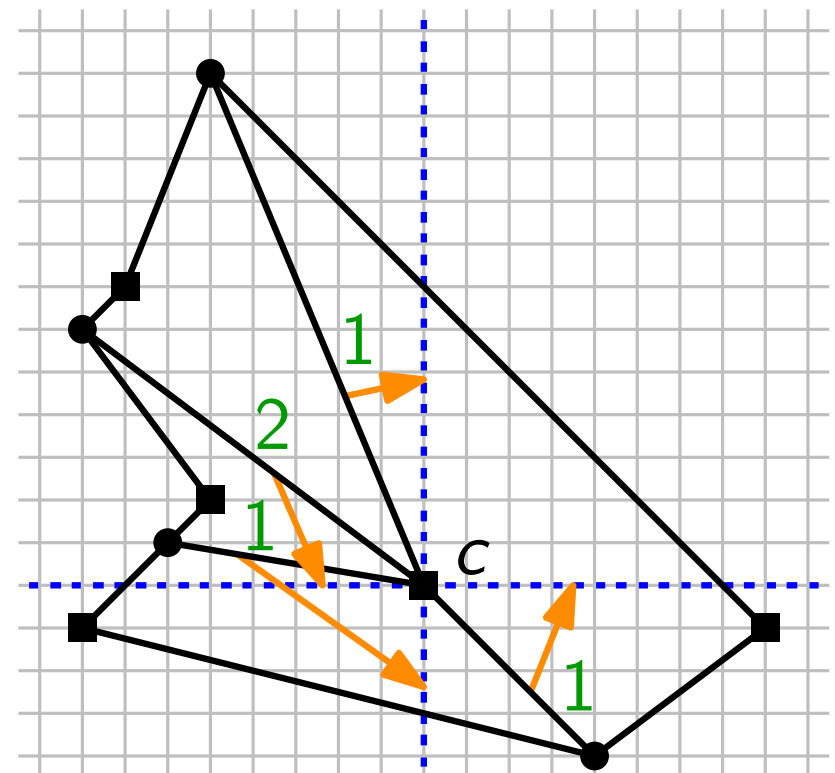
11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:

Be careful:  
One assignment  
might depend on  
another one

Solution:  
re-draw the  
independent  
ones first



# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:

Be careful:

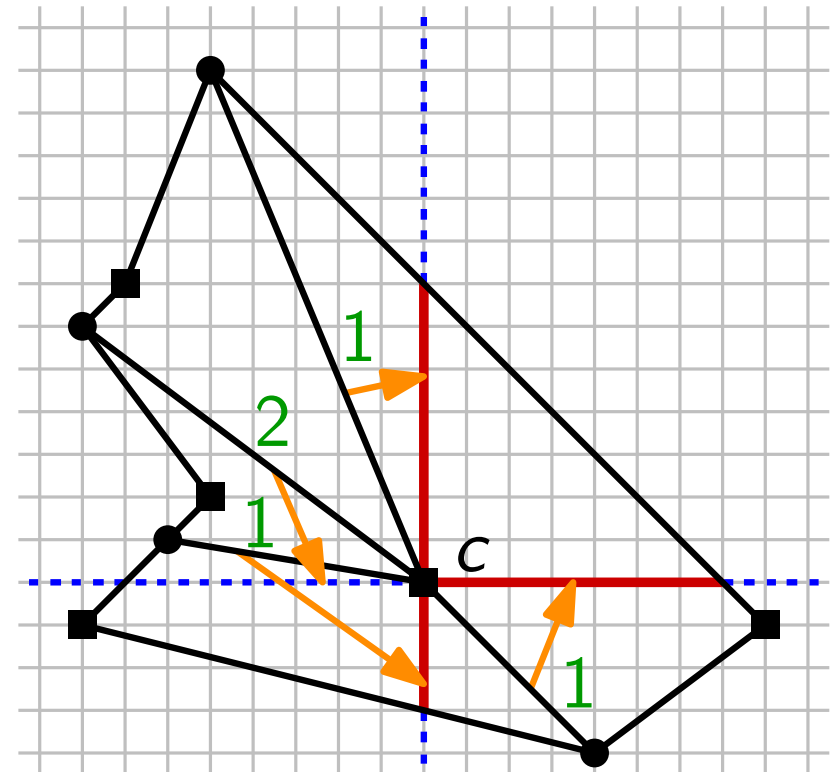
One assignment  
might depend on  
another one

Solution:

re-draw the  
independent  
ones first

Be careful:

There might be  
no grid points to  
bend the edges



# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:

Be careful:

One assignment might depend on another one

Be careful:

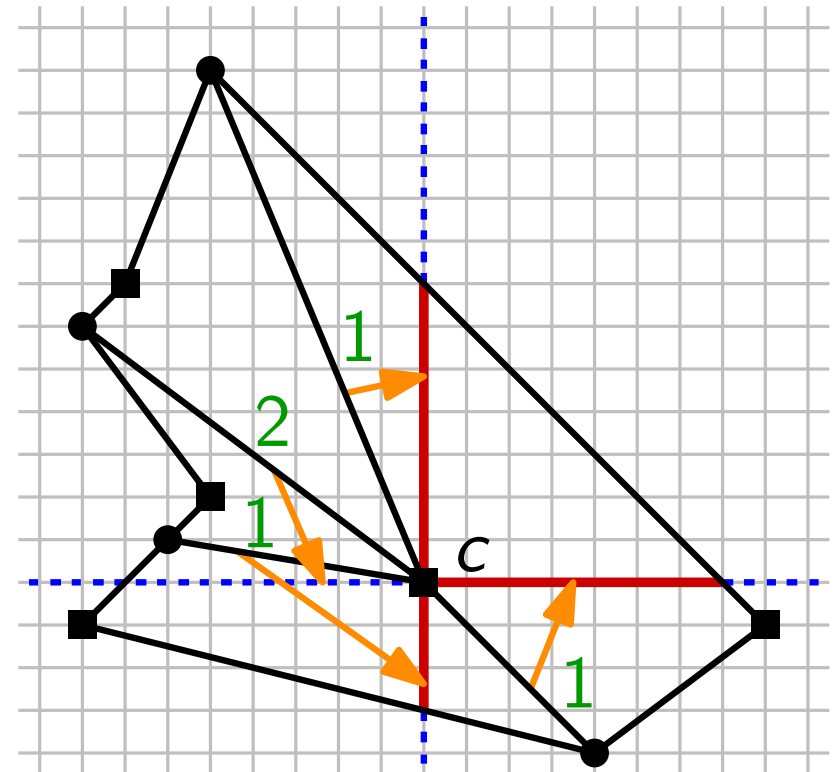
There might be no grid points to bend the edges

Solution:

re-draw the independent ones first

Solution:

make the grid sufficiently fine



# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:

grid size:  
 $O(n) \times O(n)$

Be careful:

One assignment  
might depend on  
another one

Solution:

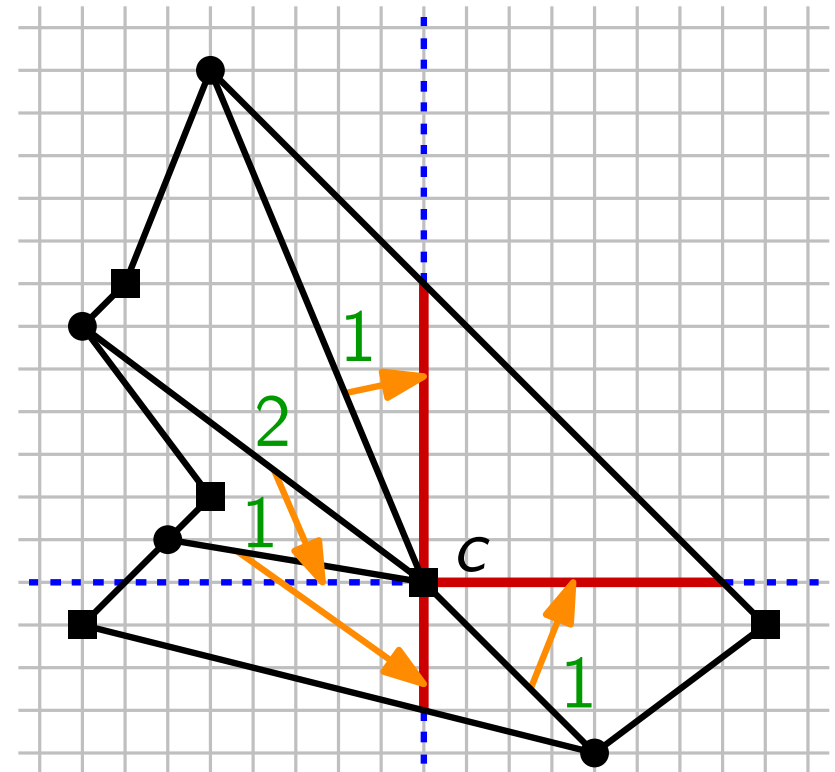
re-draw the  
independent  
ones first

Be careful:

There might be  
no grid points to  
bend the edges

Solution:

make the grid  
sufficiently  
fine



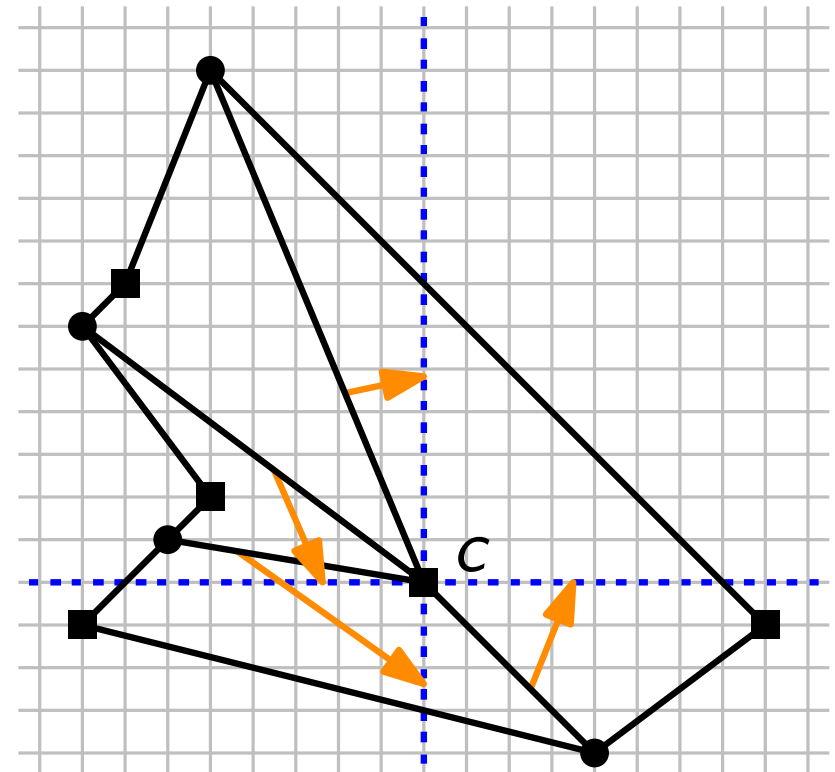
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$

grid size:  
 $O(n) \times O(n)$





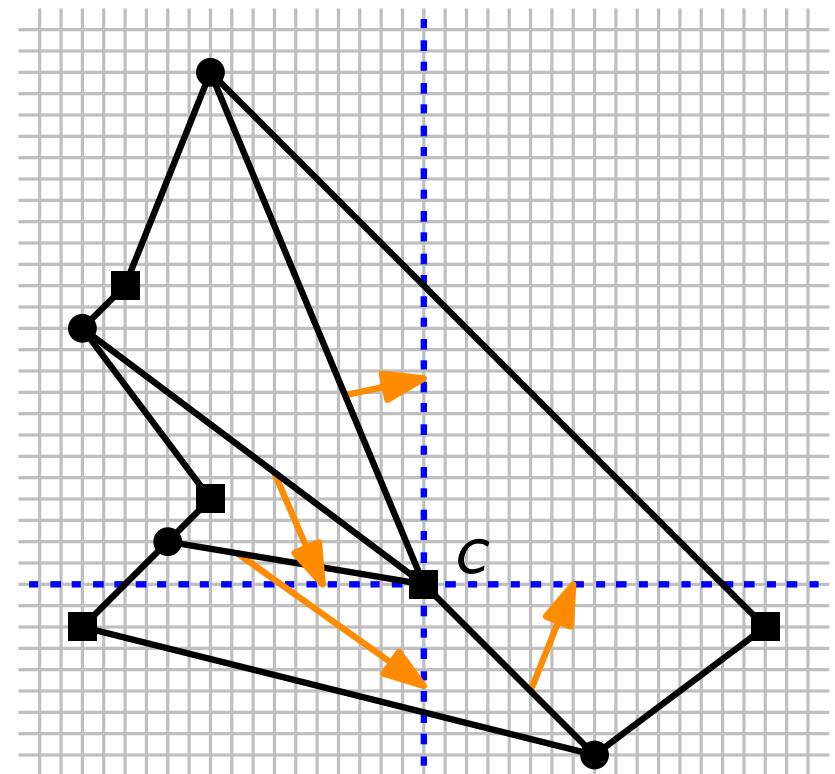
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$

grid size:  
 $O(n^2) \times O(n^2)$



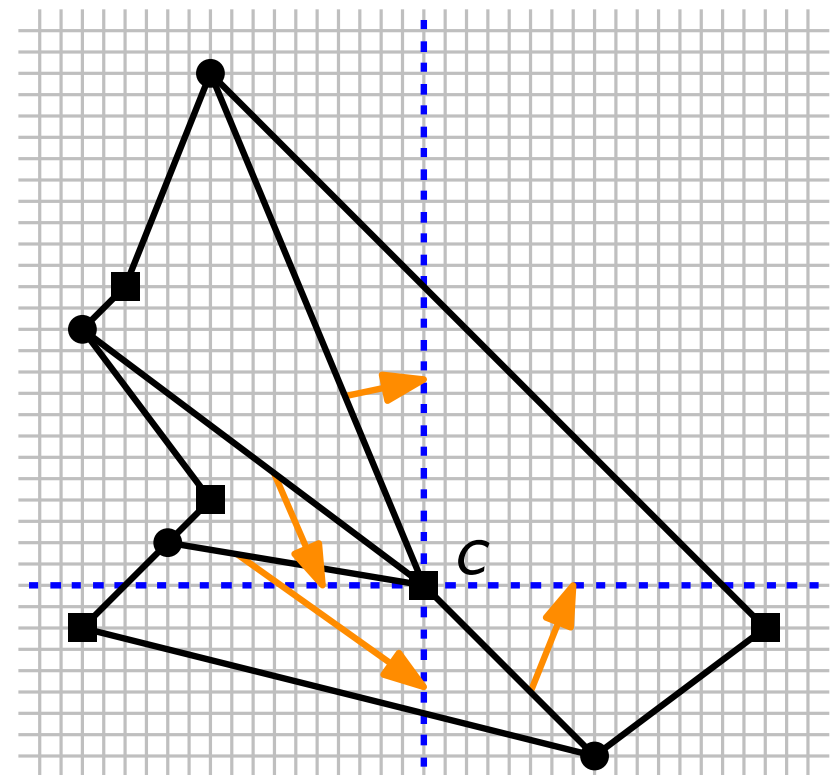
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$
  - 2. Re-draw independent edges

grid size:  
 $O(n^2) \times O(n^2)$



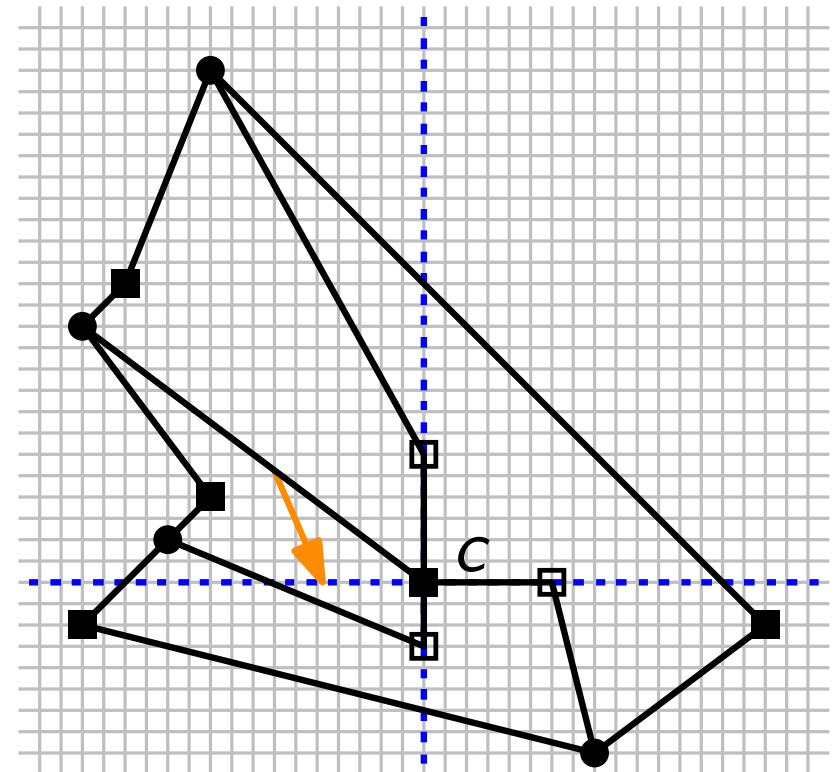
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$
  - 2. Re-draw independent edges

grid size:  
 $O(n^2) \times O(n^2)$

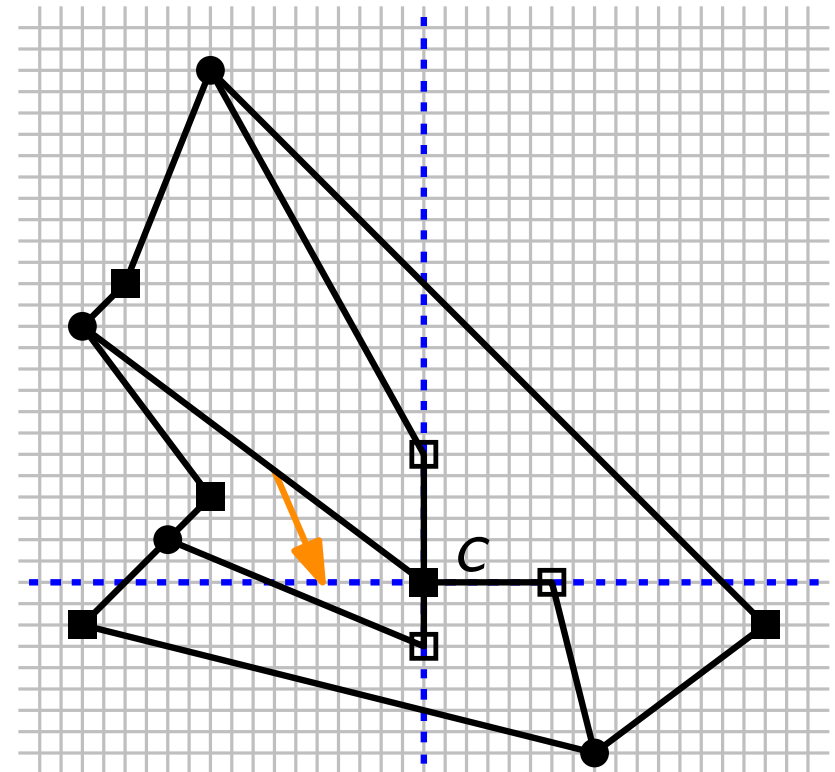


# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:  
grid size:  $O(n^2) \times O(n^2)$ 
  1. Refine the grid by  $\tilde{n} \in O(n)$
  2. Re-draw independent edges
  3. Refine the grid by  $\tilde{n}$  again



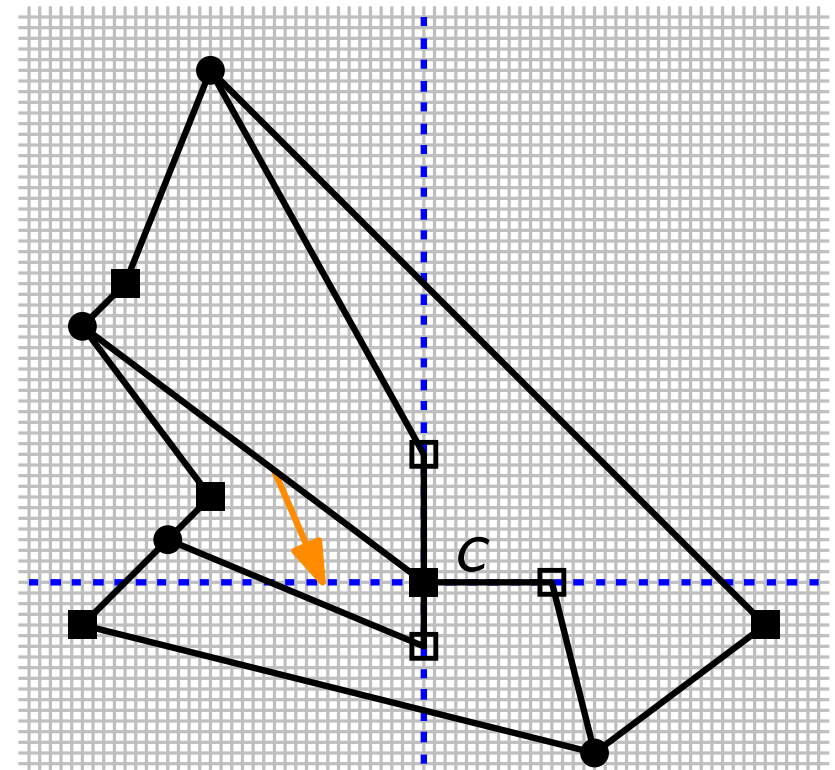
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$
  - 2. Re-draw independent edges
  - 3. Refine the grid by  $\tilde{n}$  again

grid size:  
 $O(n^3) \times O(n^3)$



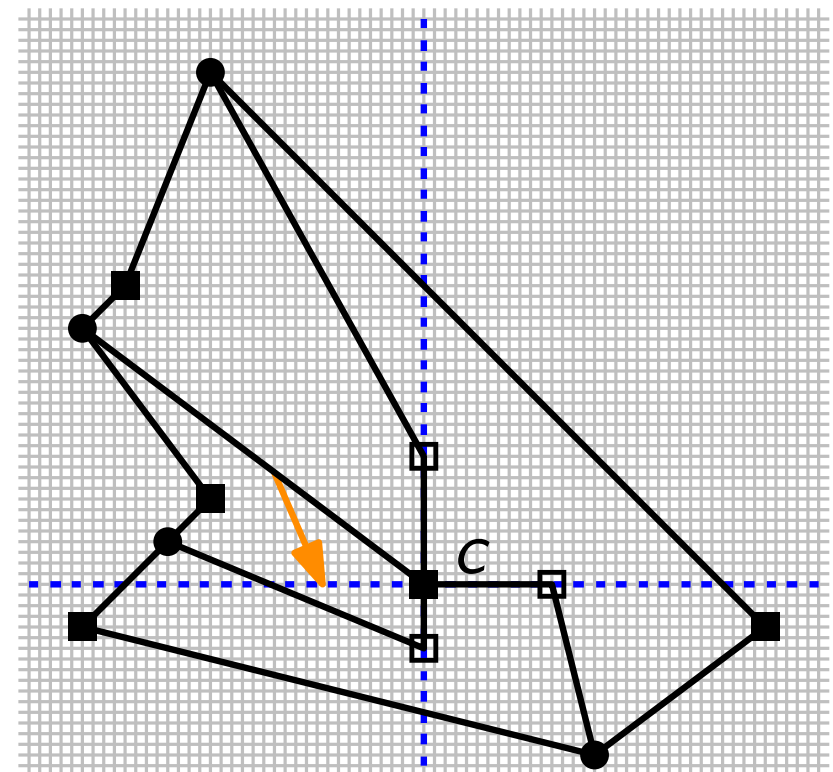
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$
  - 2. Re-draw independent edges
  - 3. Refine the grid by  $\tilde{n}$  again
  - 4. Re-draw dependent edges

grid size:  
 $O(n^3) \times O(n^3)$



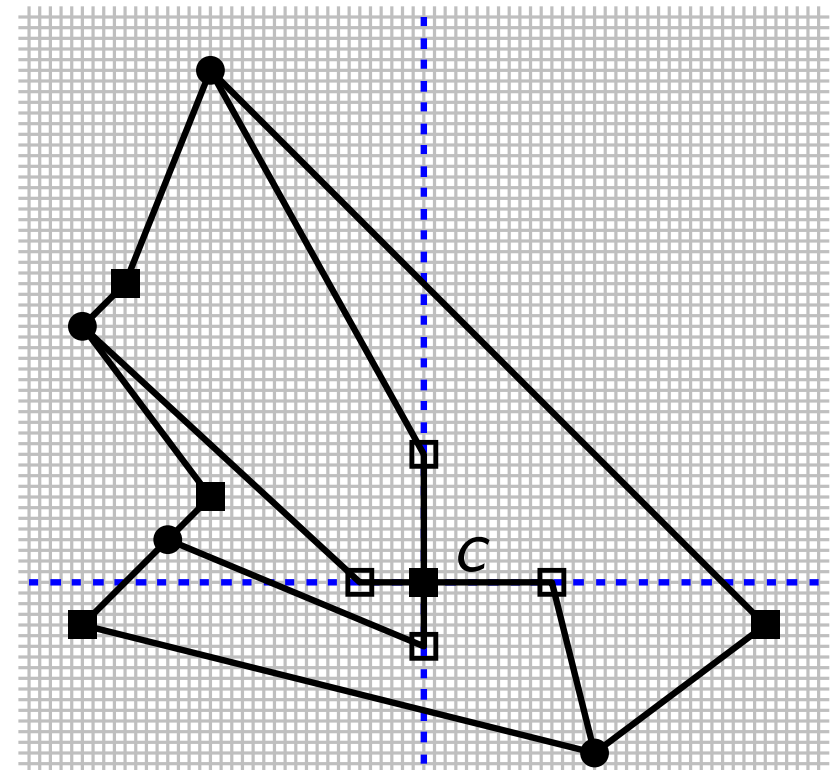
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$
  - 2. Re-draw independent edges
  - 3. Refine the grid by  $\tilde{n}$  again
  - 4. Re-draw dependent edges

grid size:  
 $O(n^3) \times O(n^3)$



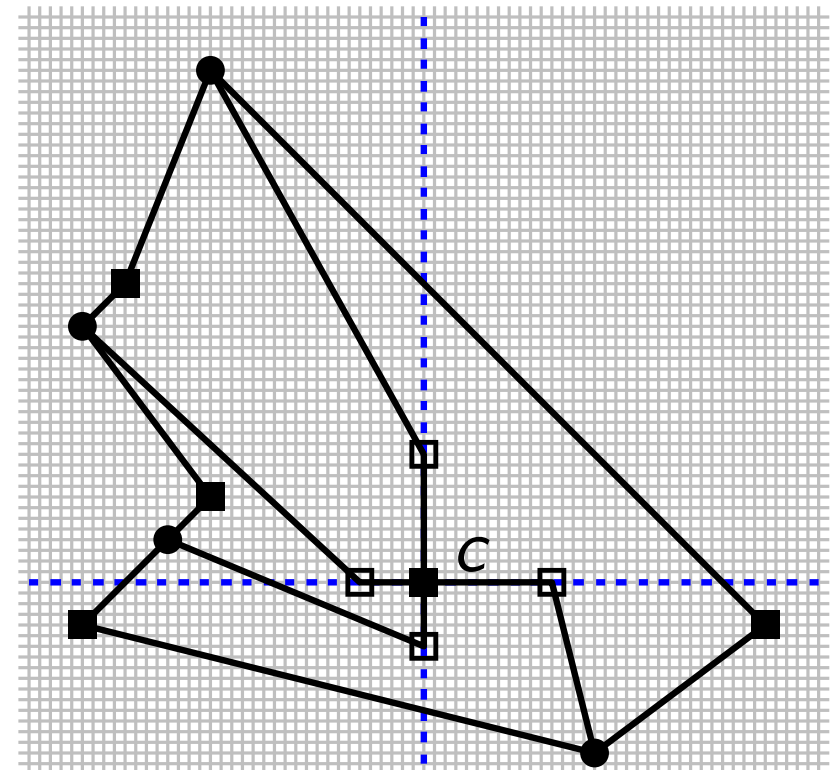
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$
  - 2. Re-draw independent edges
  - 3. Refine the grid by  $\tilde{n}$  again
  - 4. Re-draw dependent edges
- Remove the dummy objects

grid size:  
 $O(n^3) \times O(n^3)$





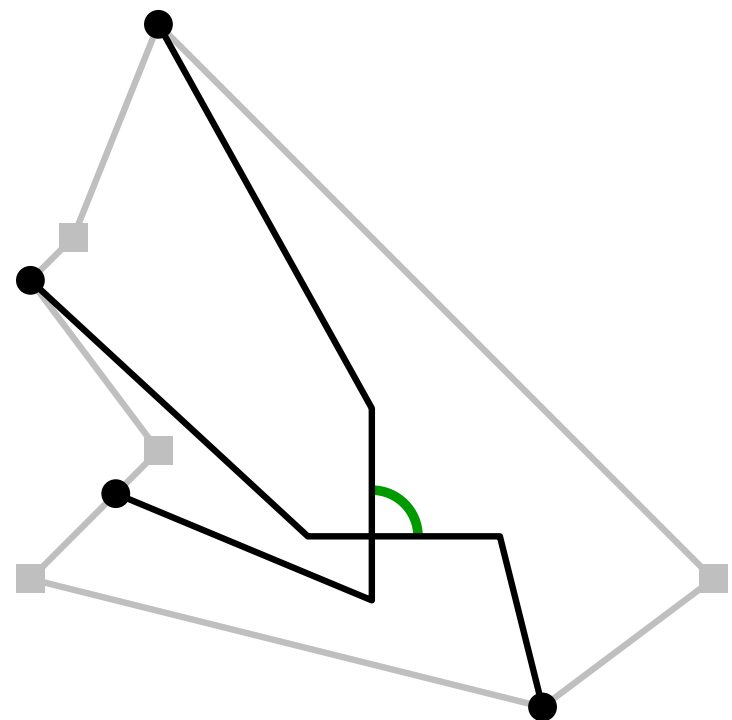
# Result 2: 1-Plane Graphs $\subseteq \text{RAC}_2^{\text{poly}}$

11

## Postprocessing (obtaining crossings at right angles):

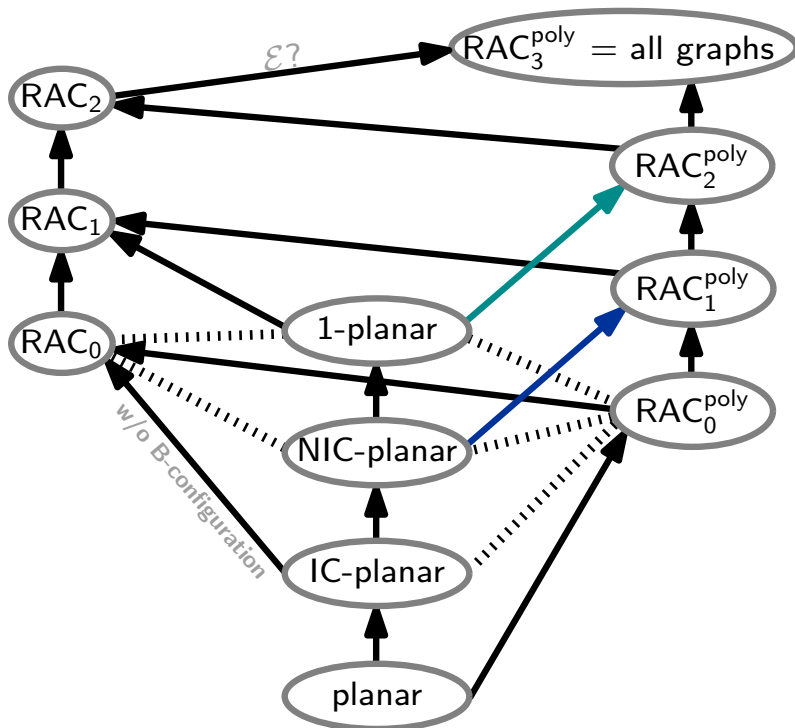
- Consider the four axis-parallel half-lines originating at  $c$
- Assign the four edges being incident to  $c$  to these half-lines
- Bend these edges at their assigned half-lines:
  - 1. Refine the grid by  $\tilde{n} \in O(n)$
  - 2. Re-draw independent edges
  - 3. Refine the grid by  $\tilde{n}$  again
  - 4. Re-draw dependent edges
- Remove the dummy objects

grid size:  
 $O(n^3) \times O(n^3)$



# Summary and Open Questions

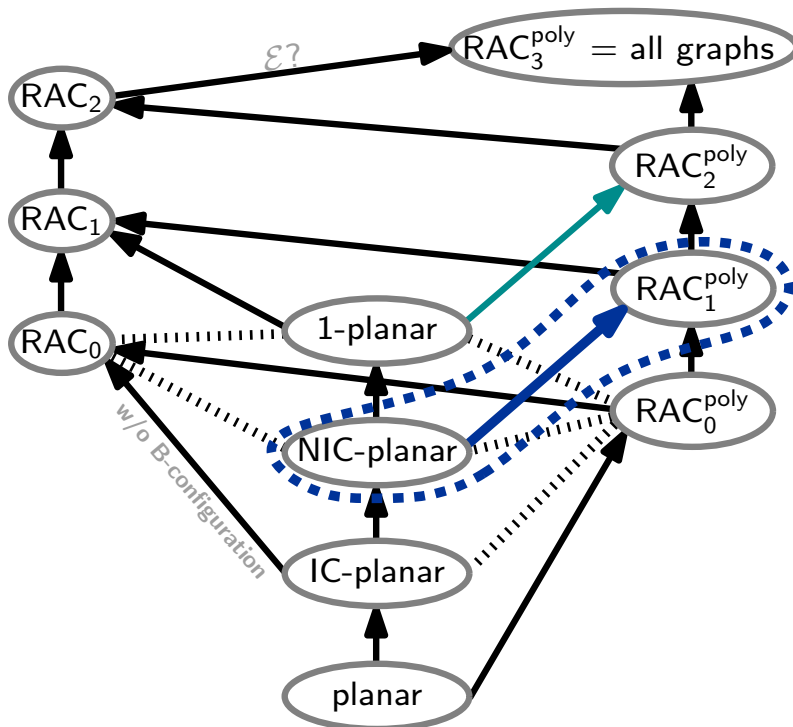
12



# Summary and Open Questions

12

NIC-plane  
 $\subseteq \text{RAC}_1^{\text{poly}}$

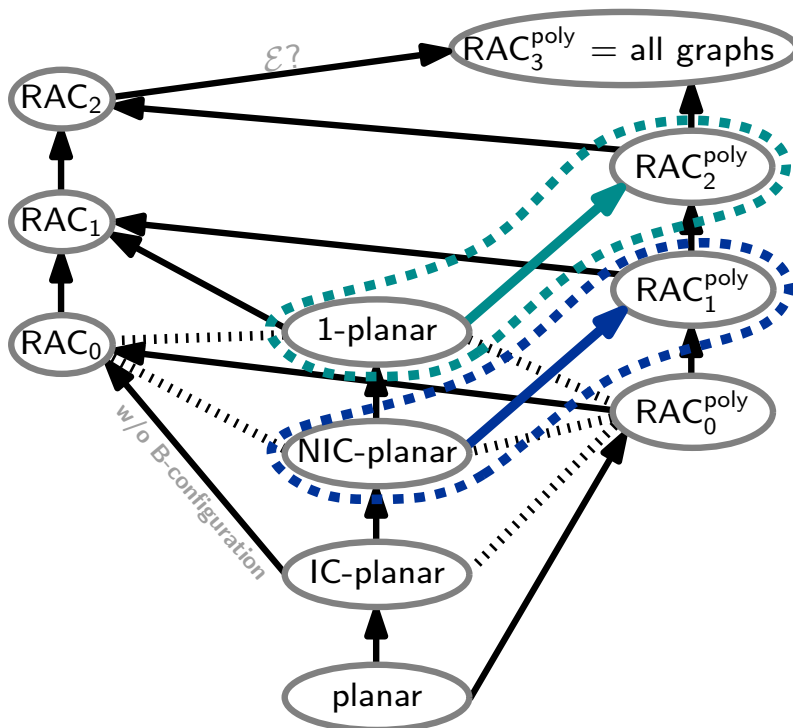


# Summary and Open Questions

12

NIC-plane  
 $\subseteq \text{RAC}_1^{\text{poly}}$

1-plane  
 $\subseteq \text{RAC}_2^{\text{poly}}$



# Summary and Open Questions

12

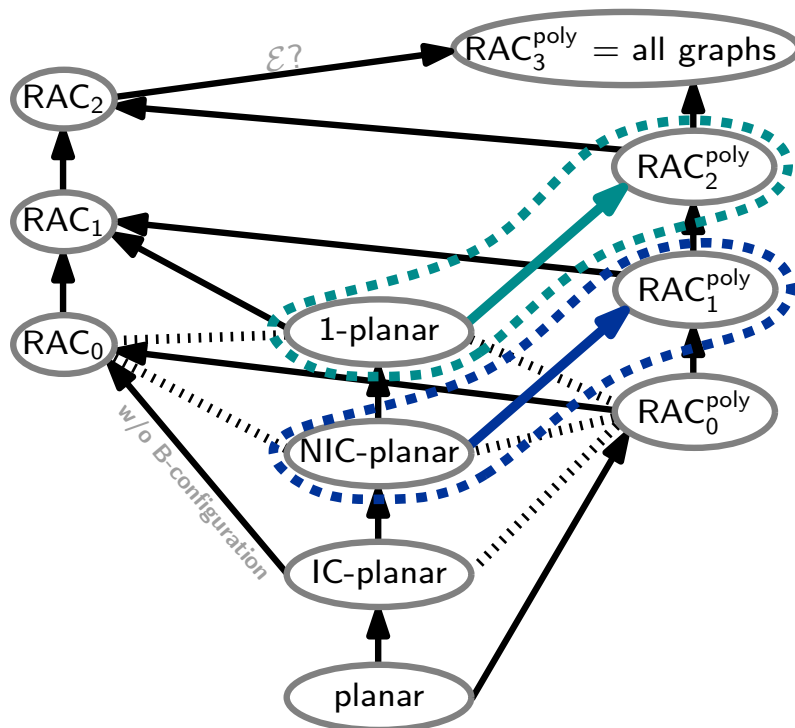
NIC-plane  
 $\subseteq \text{RAC}_1^{\text{poly}}$

1-plane  
 $\subseteq \text{RAC}_2^{\text{poly}}$

Preserves embedding

Yes

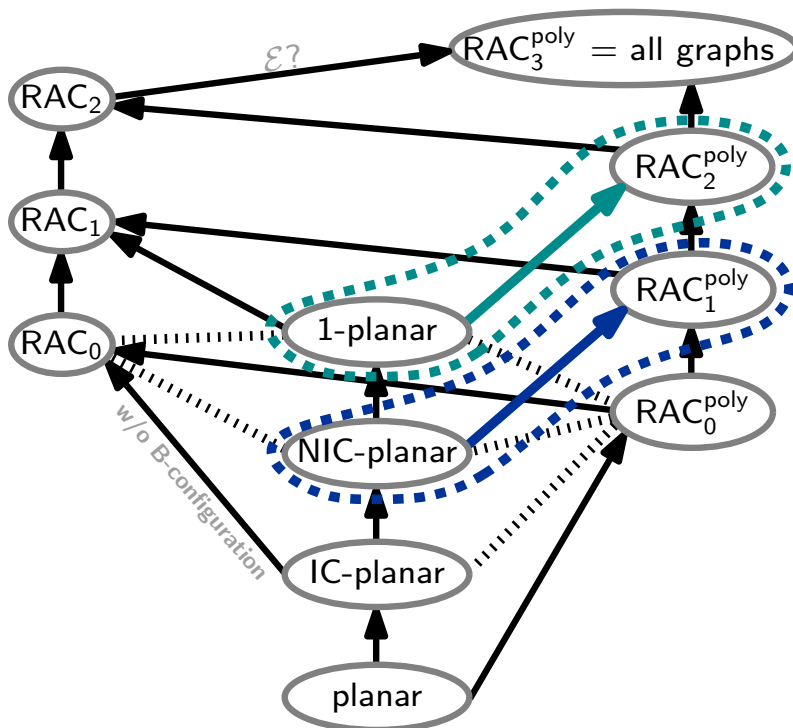
Yes



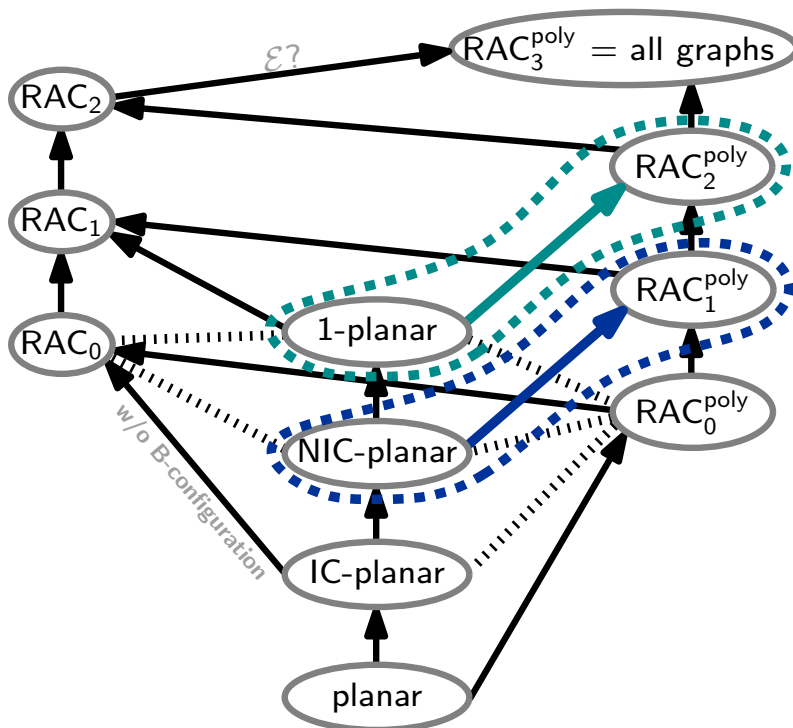
# Summary and Open Questions

12

	NIC-plane $\subseteq \text{RAC}_1^{\text{poly}}$	1-plane $\subseteq \text{RAC}_2^{\text{poly}}$
Preserves embedding	Yes	Yes
Runtime	$O(n)$	$O(n)$



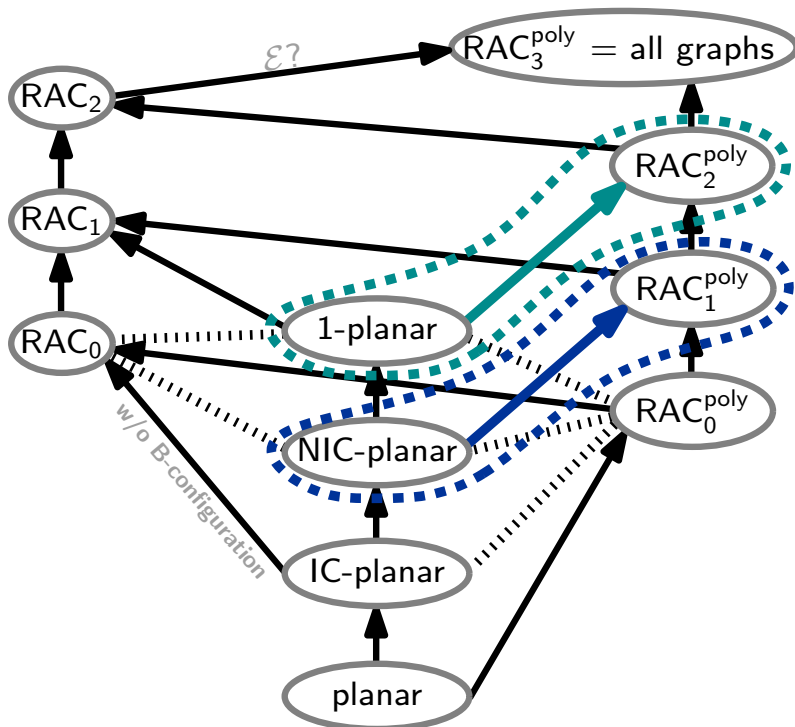
	NIC-plane $\subseteq \text{RAC}_1^{\text{poly}}$	1-plane $\subseteq \text{RAC}_2^{\text{poly}}$
Preserves embedding	Yes	Yes
Runtime	$O(n)$	$O(n)$
Bends per edge	$\leq 1$	$\leq 2$



# Summary and Open Questions

12

	NIC-plane $\subseteq \text{RAC}_1^{\text{poly}}$	1-plane $\subseteq \text{RAC}_2^{\text{poly}}$
Preserves embedding	Yes	Yes
Runtime	$O(n)$	$O(n)$
Bends per edge	$\leq 1$	$\leq 2$
Grid size	$O(n) \times O(n)$	$O(n^3) \times O(n^3)$

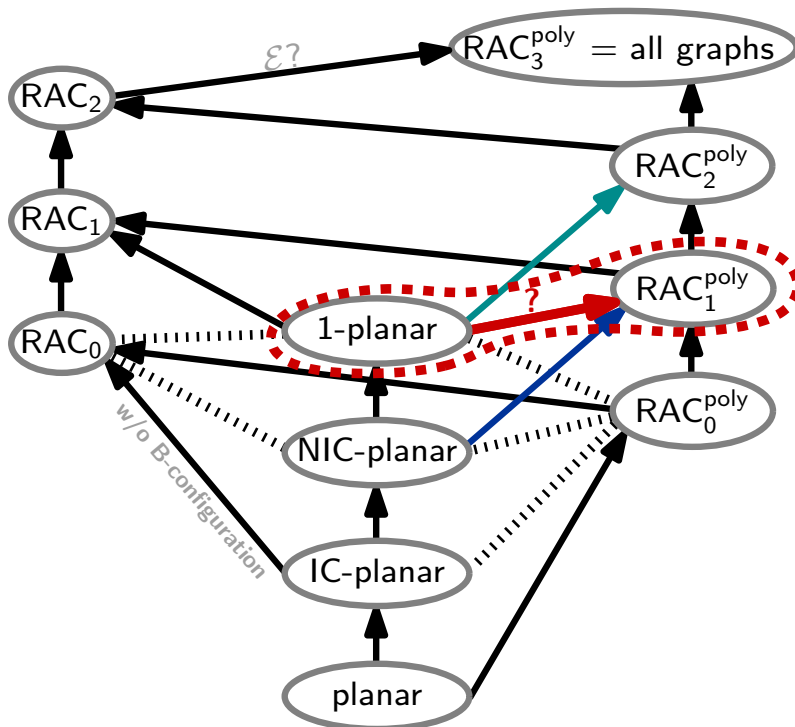




# Summary and Open Questions

12

	NIC-plane $\subseteq \text{RAC}_1^{\text{poly}}$	1-plane $\subseteq \text{RAC}_2^{\text{poly}}$
Preserves embedding	Yes	Yes
Runtime	$O(n)$	$O(n)$
Bends per edge	$\leq 1$	$\leq 2$
Grid size	$O(n) \times O(n)$	$O(n^3) \times O(n^3)$

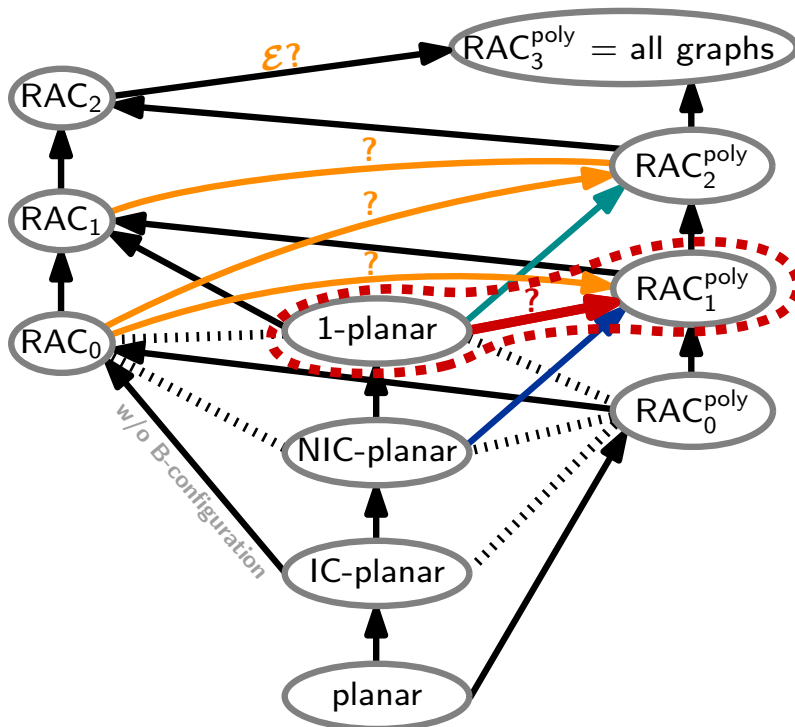


Open question:  
1-planar  $\subseteq \text{RAC}_1^{\text{poly}}$  ?

# Summary and Open Questions

12

	NIC-plane $\subseteq \text{RAC}_1^{\text{poly}}$	1-plane $\subseteq \text{RAC}_2^{\text{poly}}$
Preserves embedding	Yes	Yes
Runtime	$O(n)$	$O(n)$
Bends per edge	$\leq 1$	$\leq 2$
Grid size	$O(n) \times O(n)$	$O(n^3) \times O(n^3)$



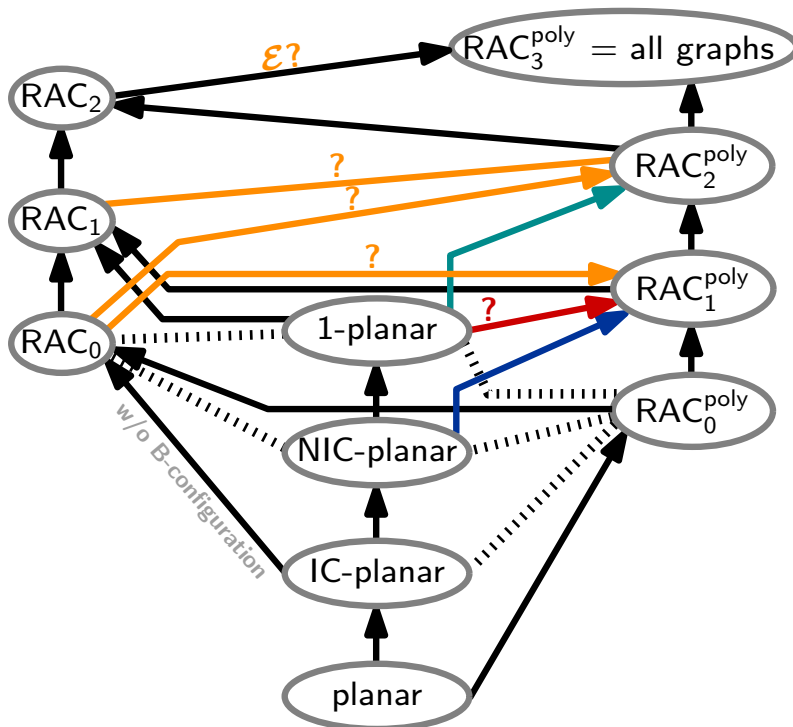
More open questions

Open question:  
 $1\text{-planar} \subseteq \text{RAC}_1^{\text{poly}} ?$

# Summary and Open Questions

12

	NIC-plane $\subseteq \text{RAC}_1^{\text{poly}}$	1-plane $\subseteq \text{RAC}_2^{\text{poly}}$
Preserves embedding	Yes	Yes
Runtime	$O(n)$	$O(n)$
Bends per edge	$\leq 1$	$\leq 2$
Grid size	$O(n) \times O(n)$	$O(n^3) \times O(n^3)$



More open questions

Open question:  
 $1\text{-planar} \subseteq \text{RAC}_1^{\text{poly}} ?$