Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Chair for
INFORMATICS I
Efficient Algorithms and
Knowledge-Based Systems

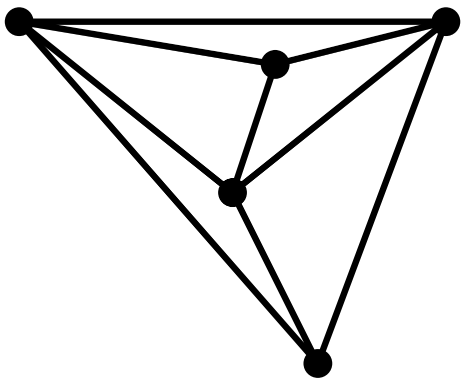
Institute for Informatics

1-Bend RAC Drawings of NIC-Planar Graphs in Quadratic Area

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

Beyond-Planar Graphs

Types of Drawings:

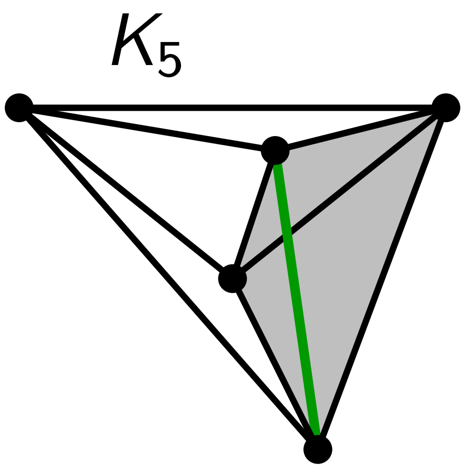


Planar: No crossings

Beyond-Planar Graphs

Types of Drawings:

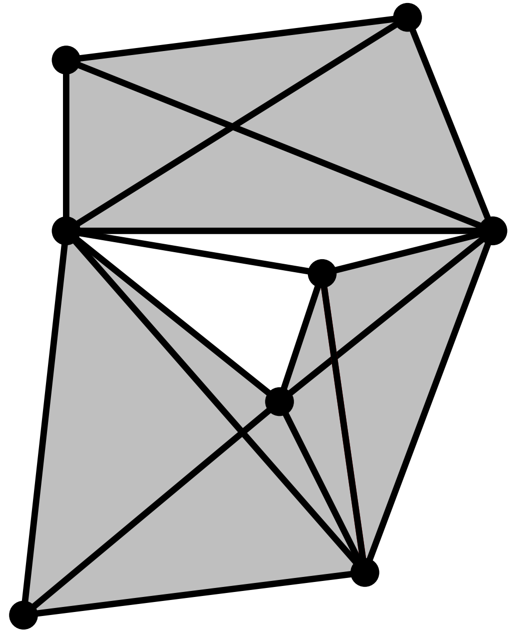
1-Planar: ≤ 1 crossings per edge



Planar: No crossings

Beyond-Planar Graphs

Types of Drawings:

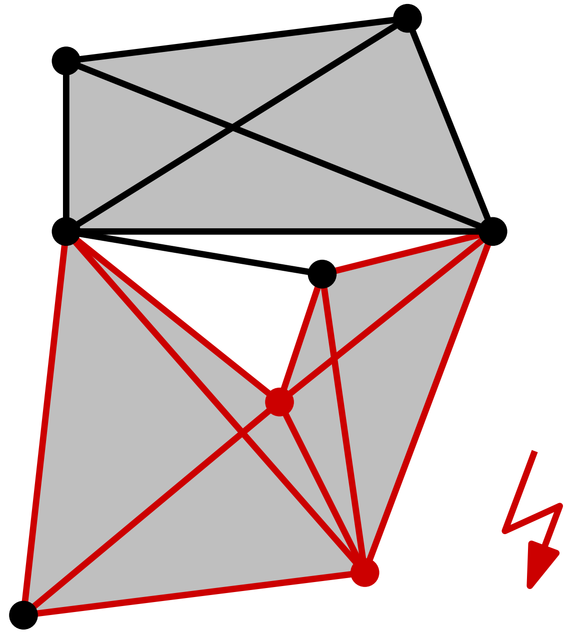


1-Planar: ≤ 1 crossings per edge

Planar: No crossings

Beyond-Planar Graphs

Types of Drawings:

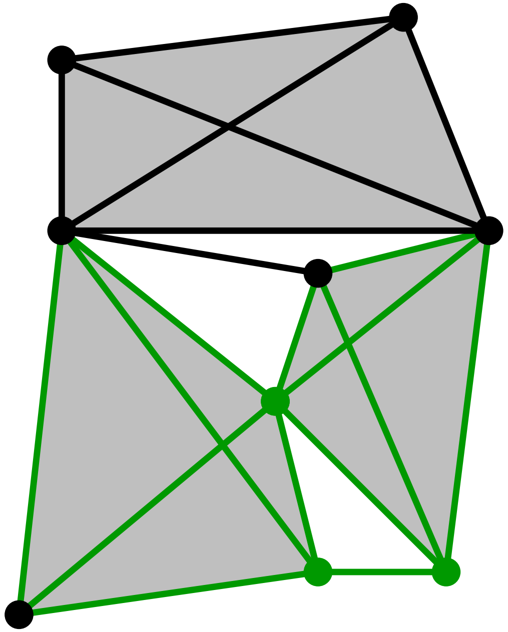


1-Planar: ≤ 1 crossings per edge

NIC-Planar: Two crossings share ≤ 1 vertices

Planar: No crossings

Beyond-Planar Graphs



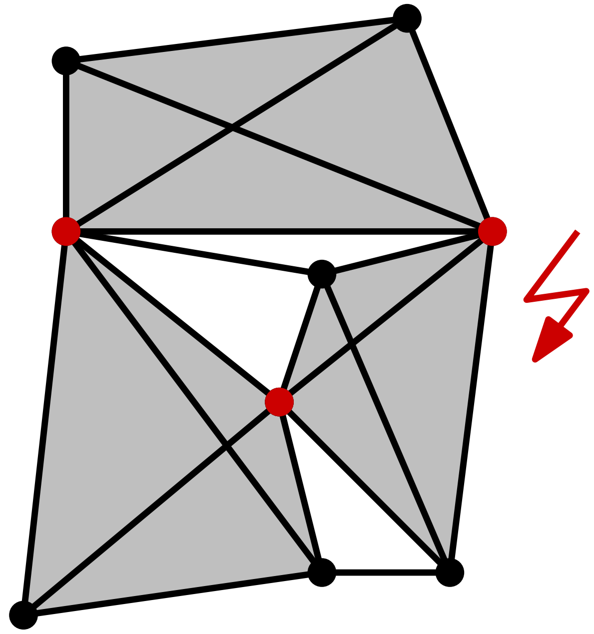
Types of Drawings:

1-Planar: ≤ 1 crossings per edge

NIC-Planar: Two crossings share ≤ 1 vertices

Planar: No crossings

Beyond-Planar Graphs



Types of Drawings:

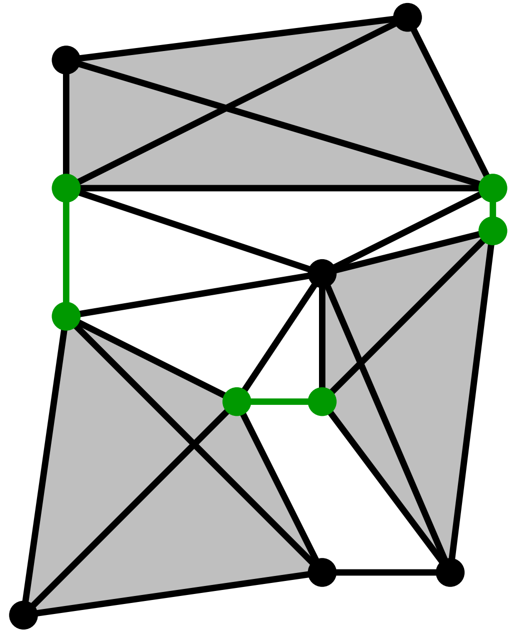
1-Planar: ≤ 1 crossings per edge

NIC-Planar: Two crossings share ≤ 1 vertices

IC-Planar: Two crossings share no vertices

Planar: No crossings

Beyond-Planar Graphs



Types of Drawings:

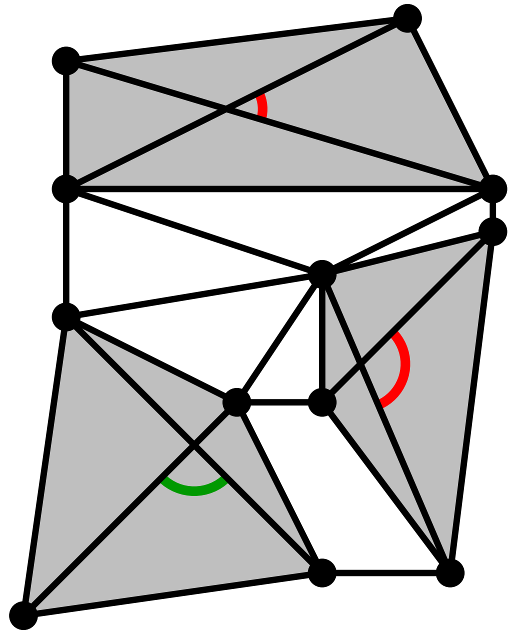
1-Planar: ≤ 1 crossings per edge

NIC-Planar: Two crossings share ≤ 1 vertices

IC-Planar: Two crossings share no vertices

Planar: No crossings

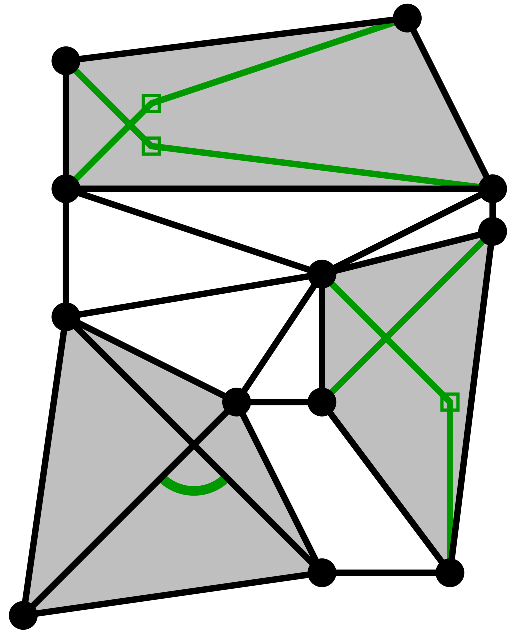
Beyond-Planar Graphs



Types of Drawings:

1-Planar:	≤ 1 crossings per edge
NIC-Planar:	Two crossings share ≤ 1 vertices
IC-Planar:	Two crossings share no vertices
Planar:	No crossings
RAC:	Right angle crossings

Beyond-Planar Graphs



Types of Drawings:

1-Planar: ≤ 1 crossings per edge

NIC-Planar: Two crossings share ≤ 1 vertices

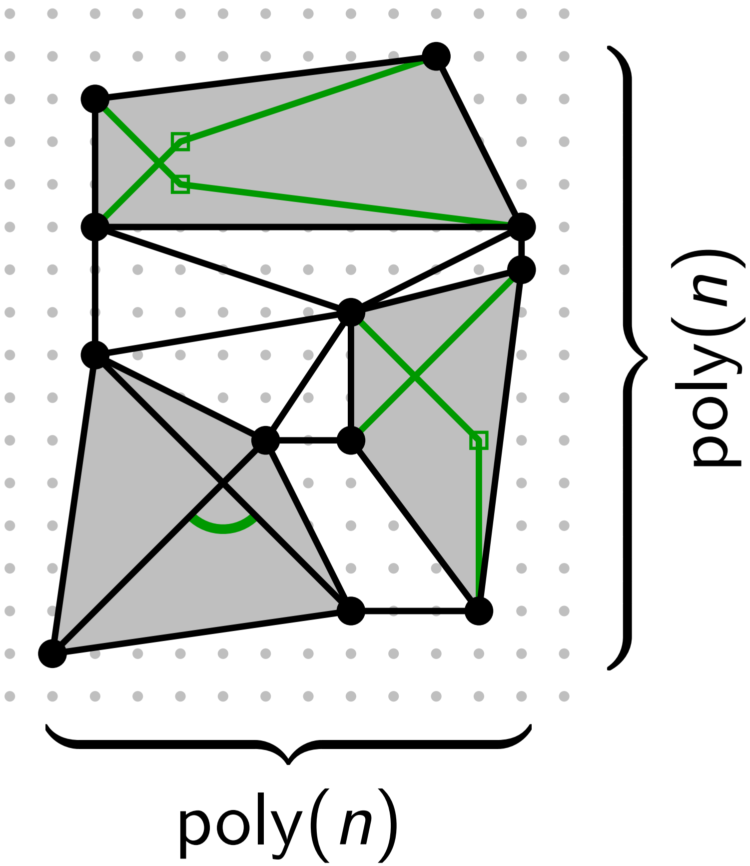
IC-Planar: Two crossings share no vertices

Planar: No crossings

RAC: Right angle crossings

RAC_1: with ≤ 1 bends per edge
RAC_0: with straight-line edges

Beyond-Planar Graphs



Types of Drawings:

1-Planar: ≤ 1 crossings per edge

NIC-Planar: Two crossings share ≤ 1 vertices

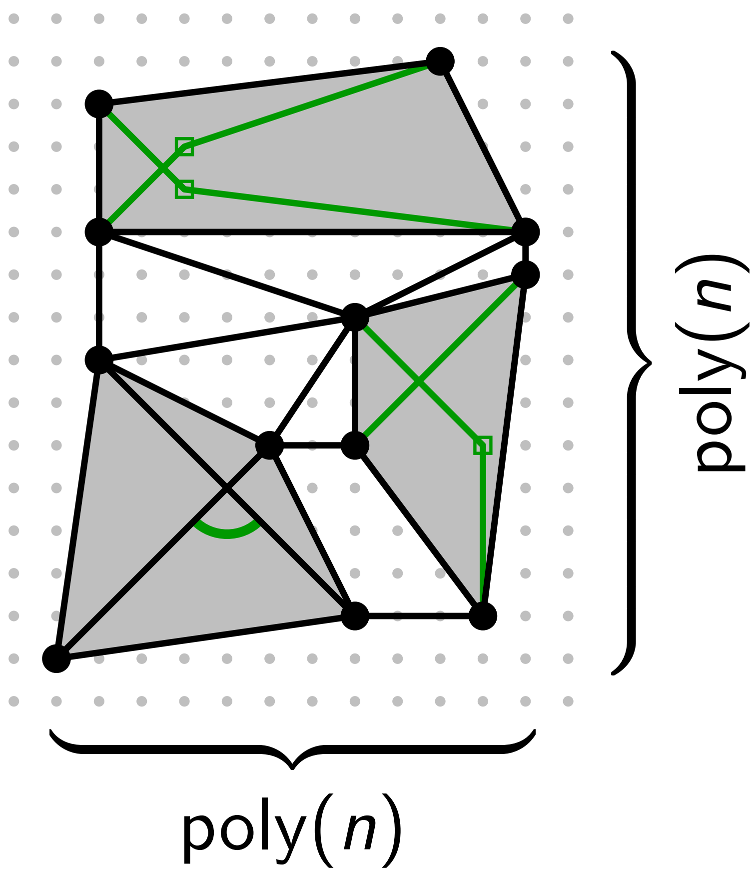
IC-Planar: Two crossings share no vertices

Planar: No crossings

RAC: Right angle crossings

- RAC_1: with ≤ 1 bends per edge
- RAC_0: with straight-line edges
- RAC^{poly}: in polynomial area

Beyond-Planar Graphs



Types of Drawings:

1-Planar: ≤ 1 crossings per edge

NIC-Planar: Two crossings share ≤ 1 vertices

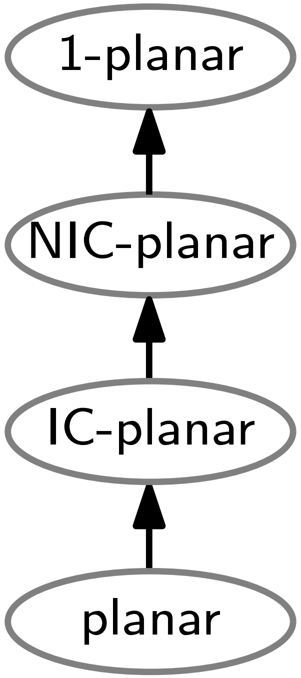
IC-Planar: Two crossings share no vertices

Planar: No crossings

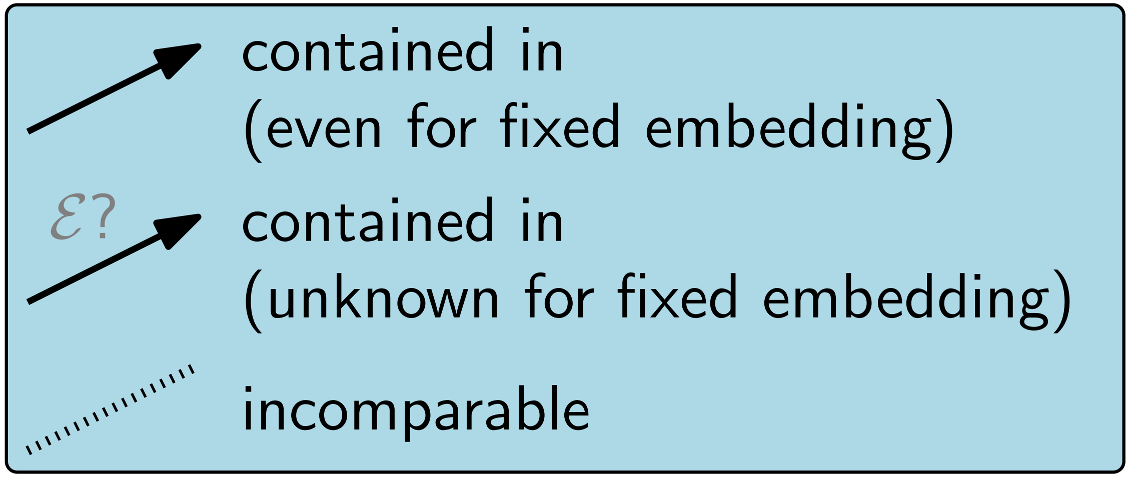
RAC: Right angle crossings

RAC_1: with ≤ 1 bends per edge
RAC_0: with straight-line edges
RAC^{poly}: in polynomial area

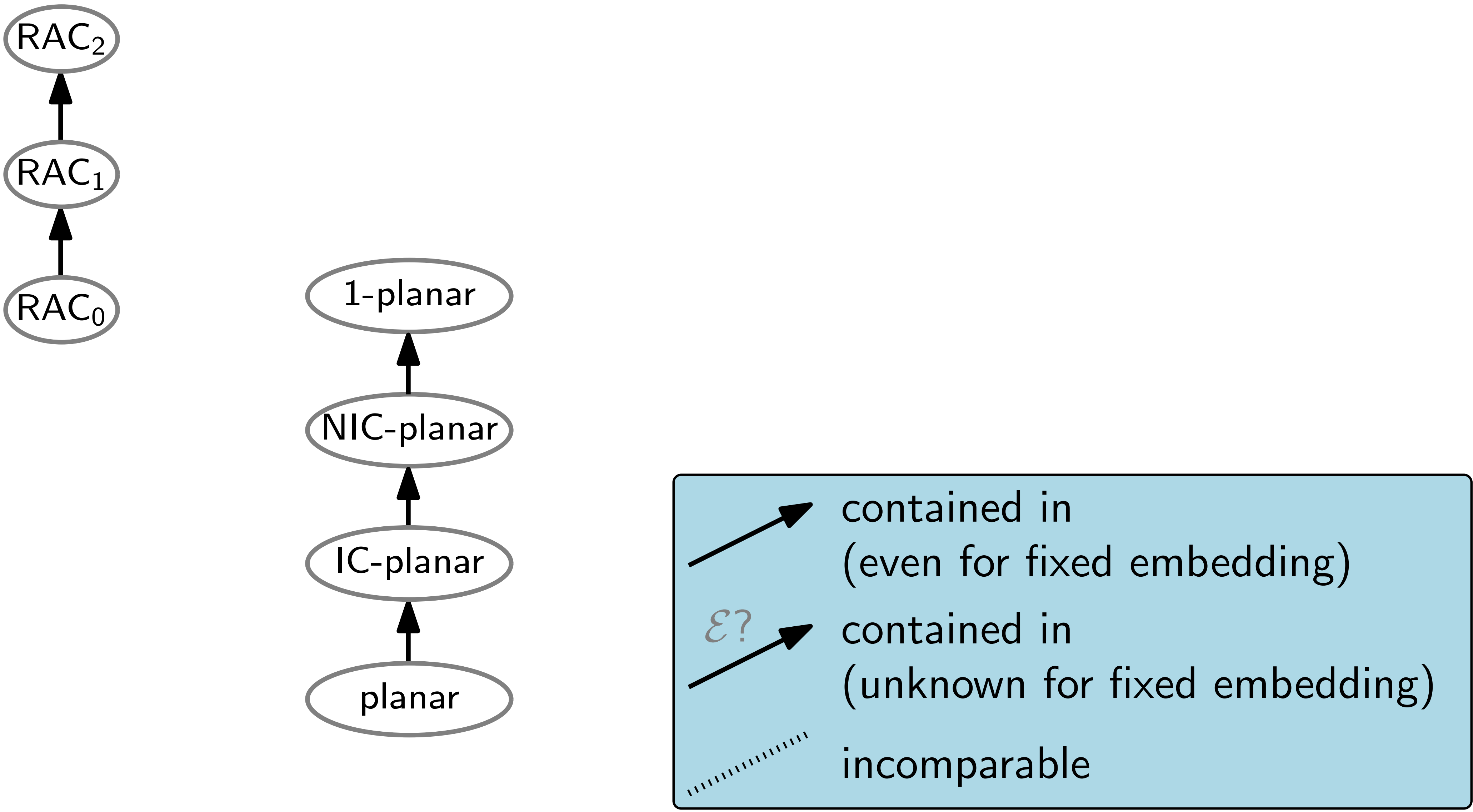
Related Work



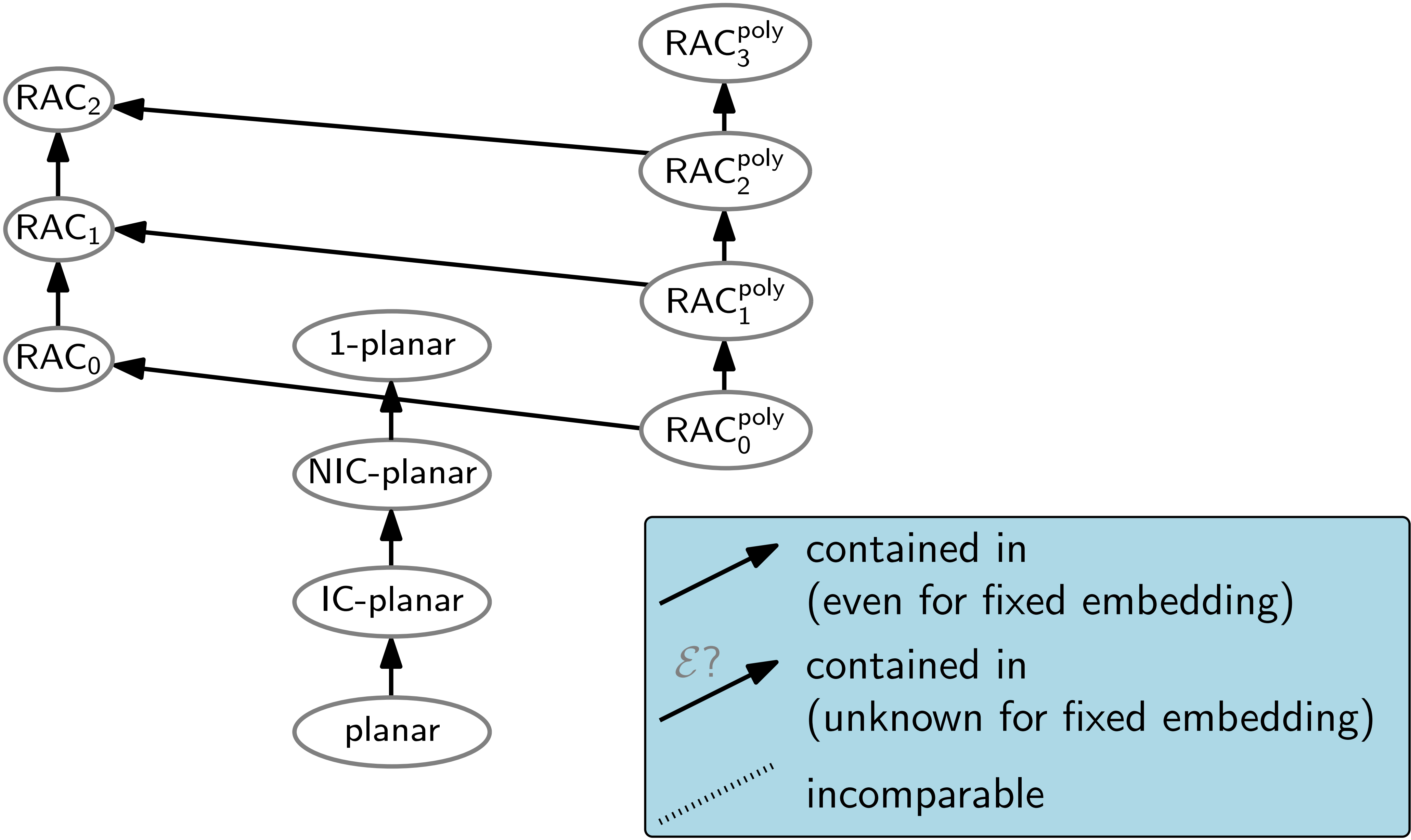
$= \{\text{graph } G \mid G \text{ has a NIC-planar drawing}\}$



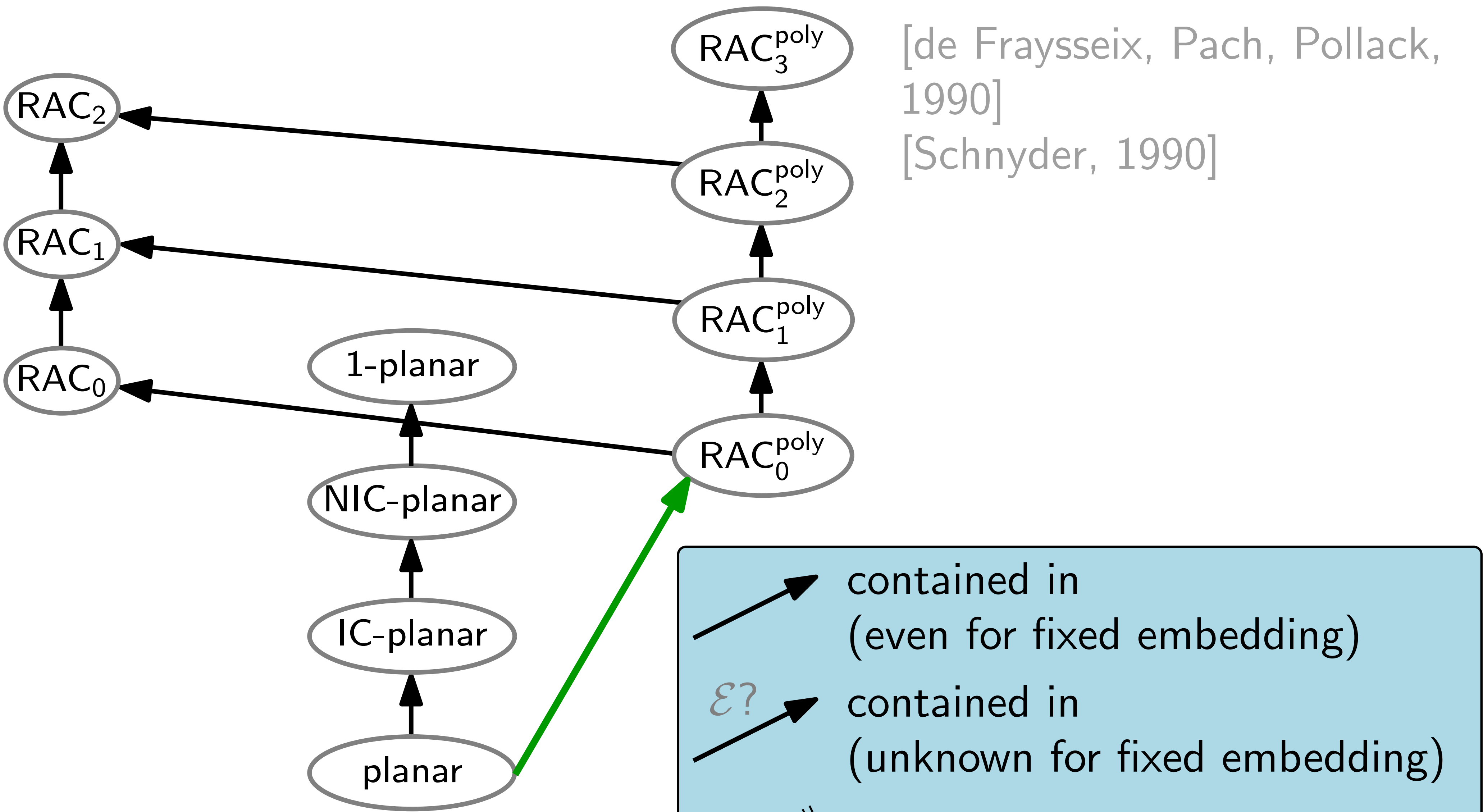
Related Work



Related Work

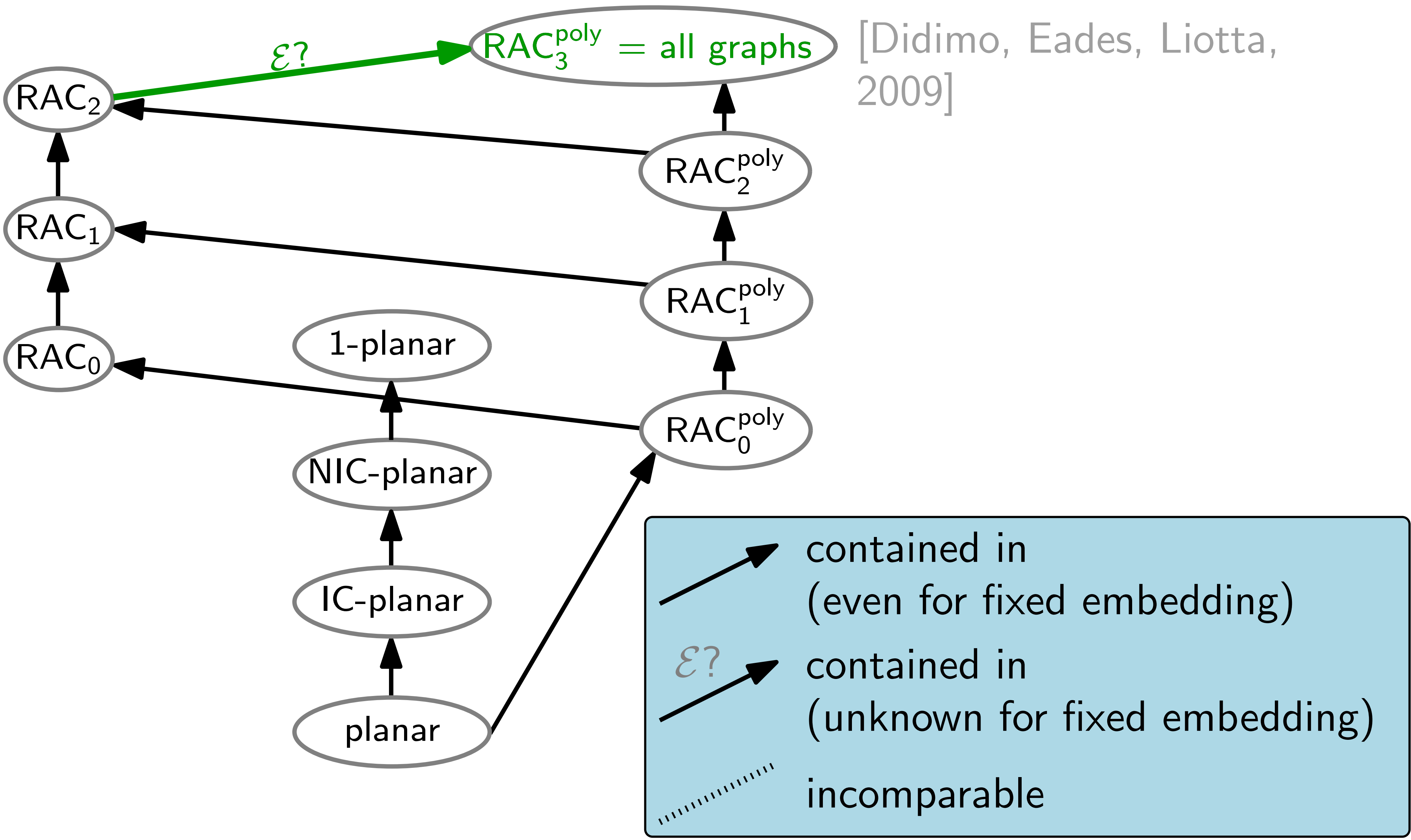


Related Work

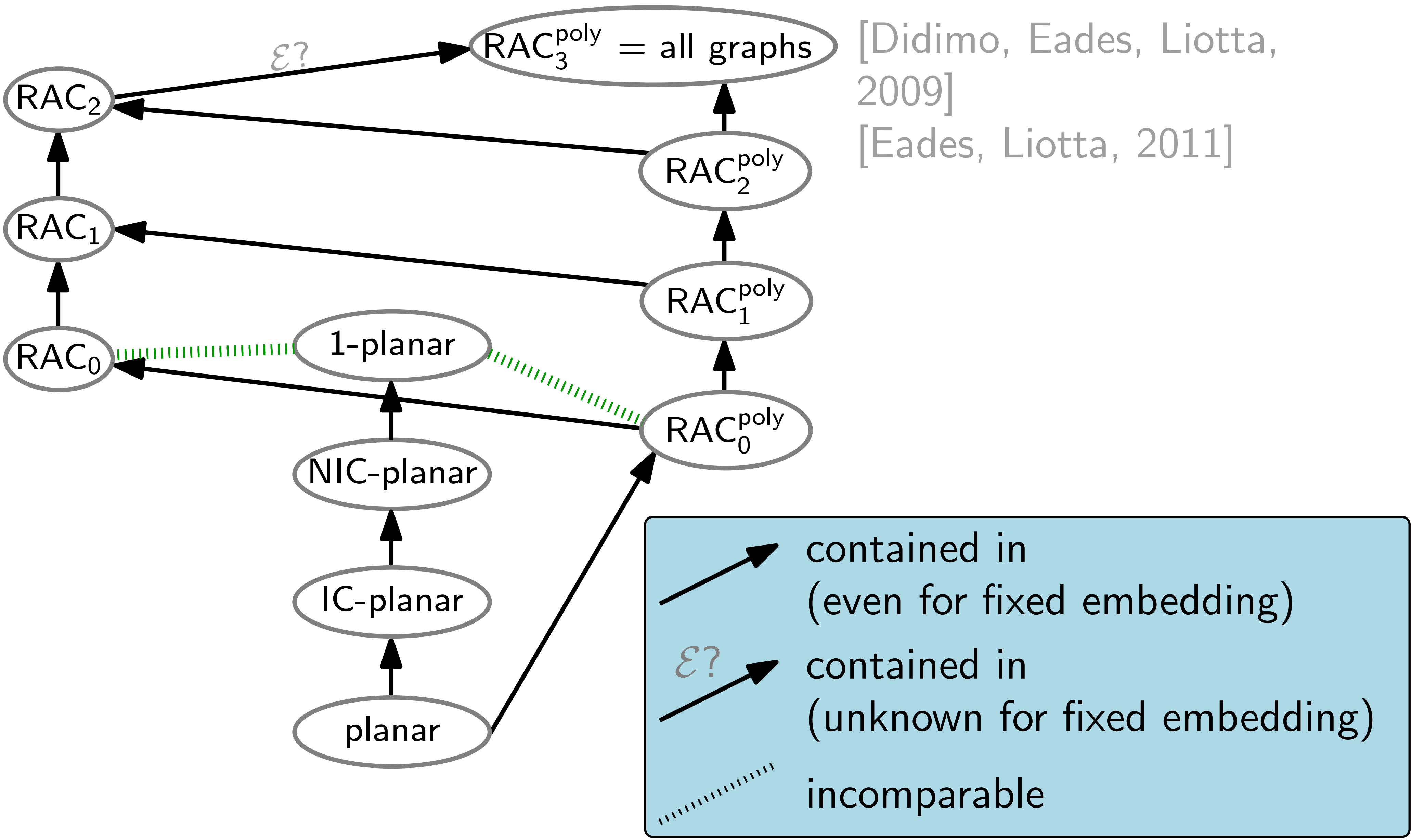


[de Fraysseix, Pach, Pollack, 1990]
[Schnyder, 1990]

Related Work

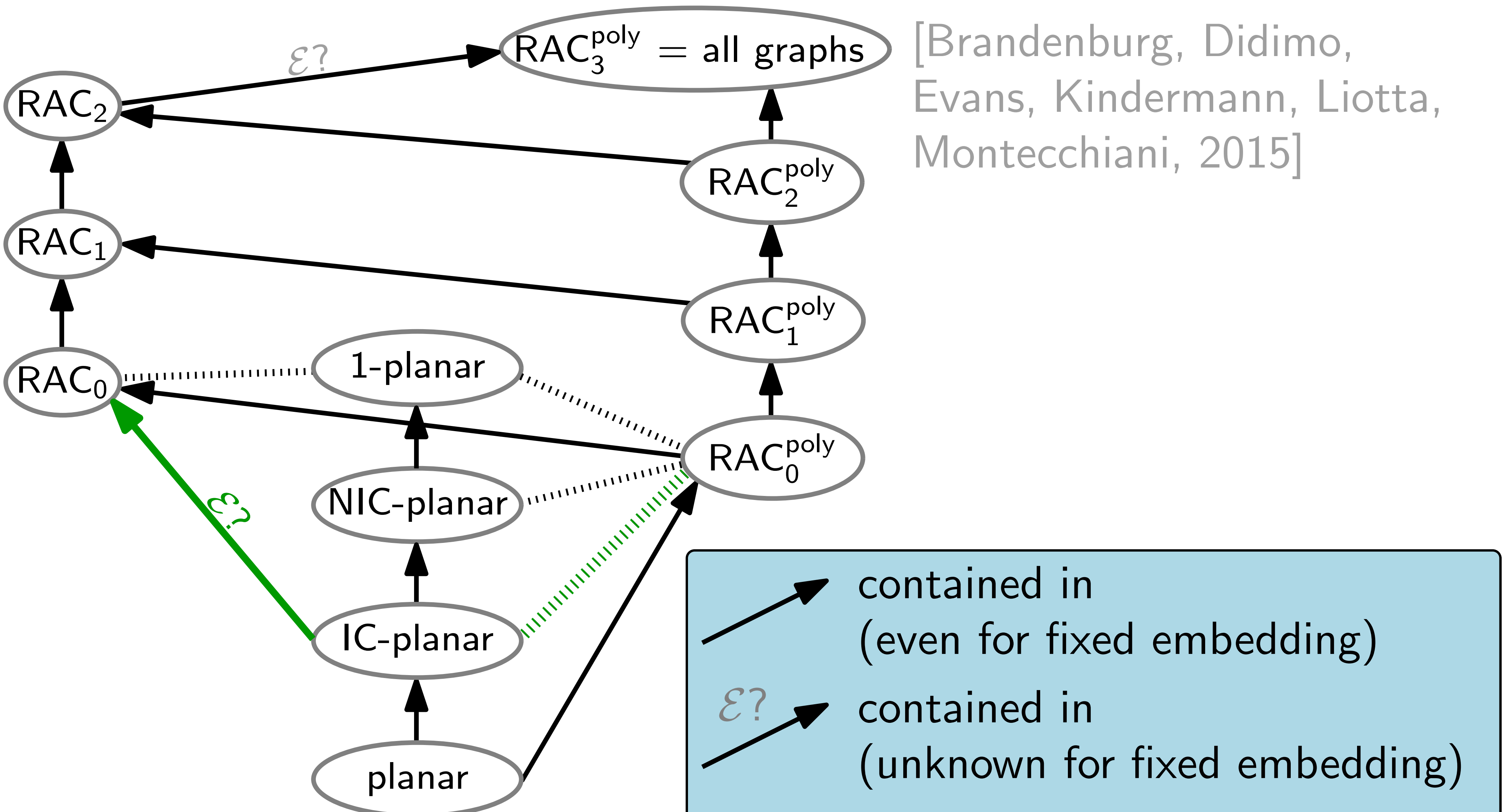


Related Work



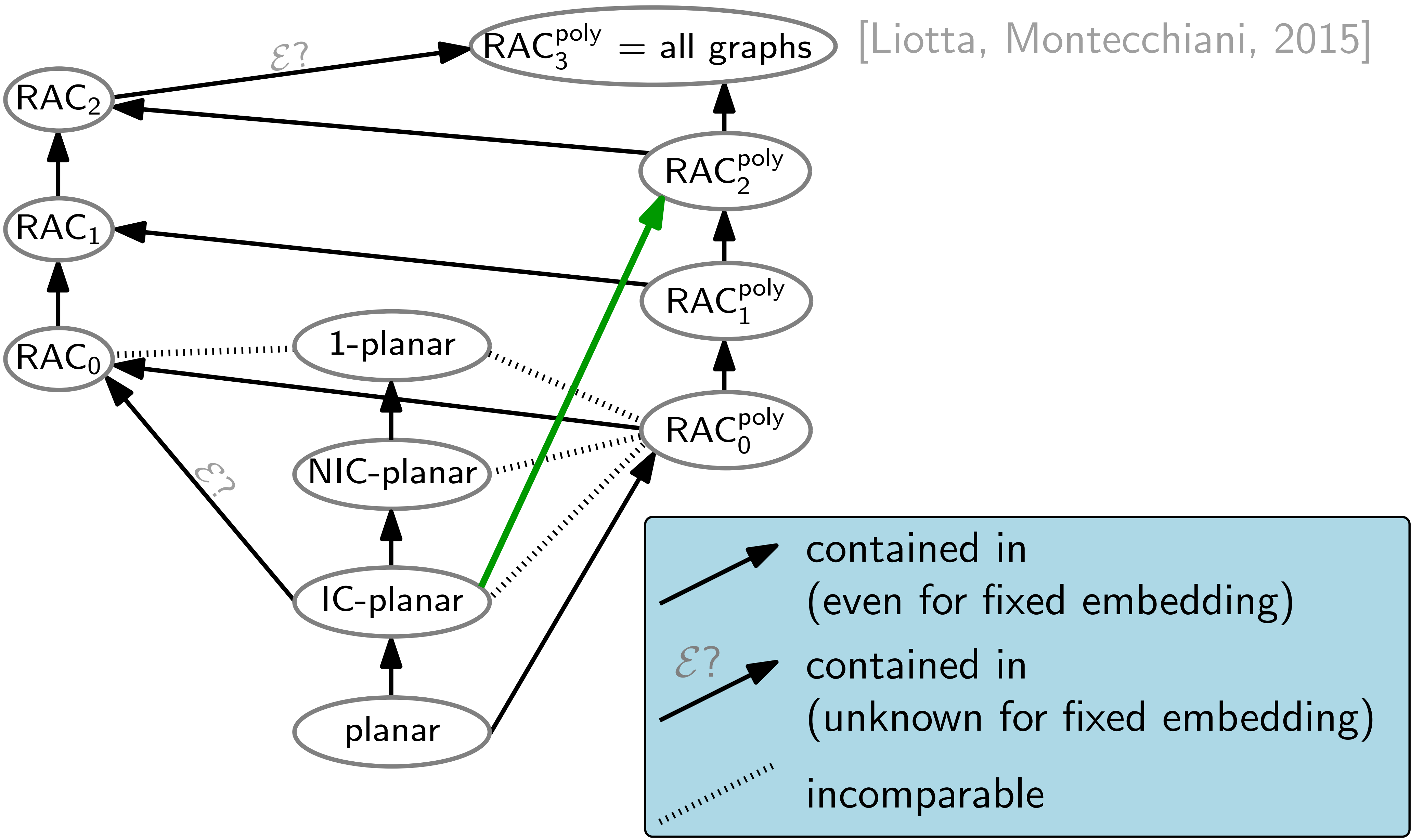
[Didimo, Eades, Liotta, 2009]
[Eades, Liotta, 2011]

Related Work

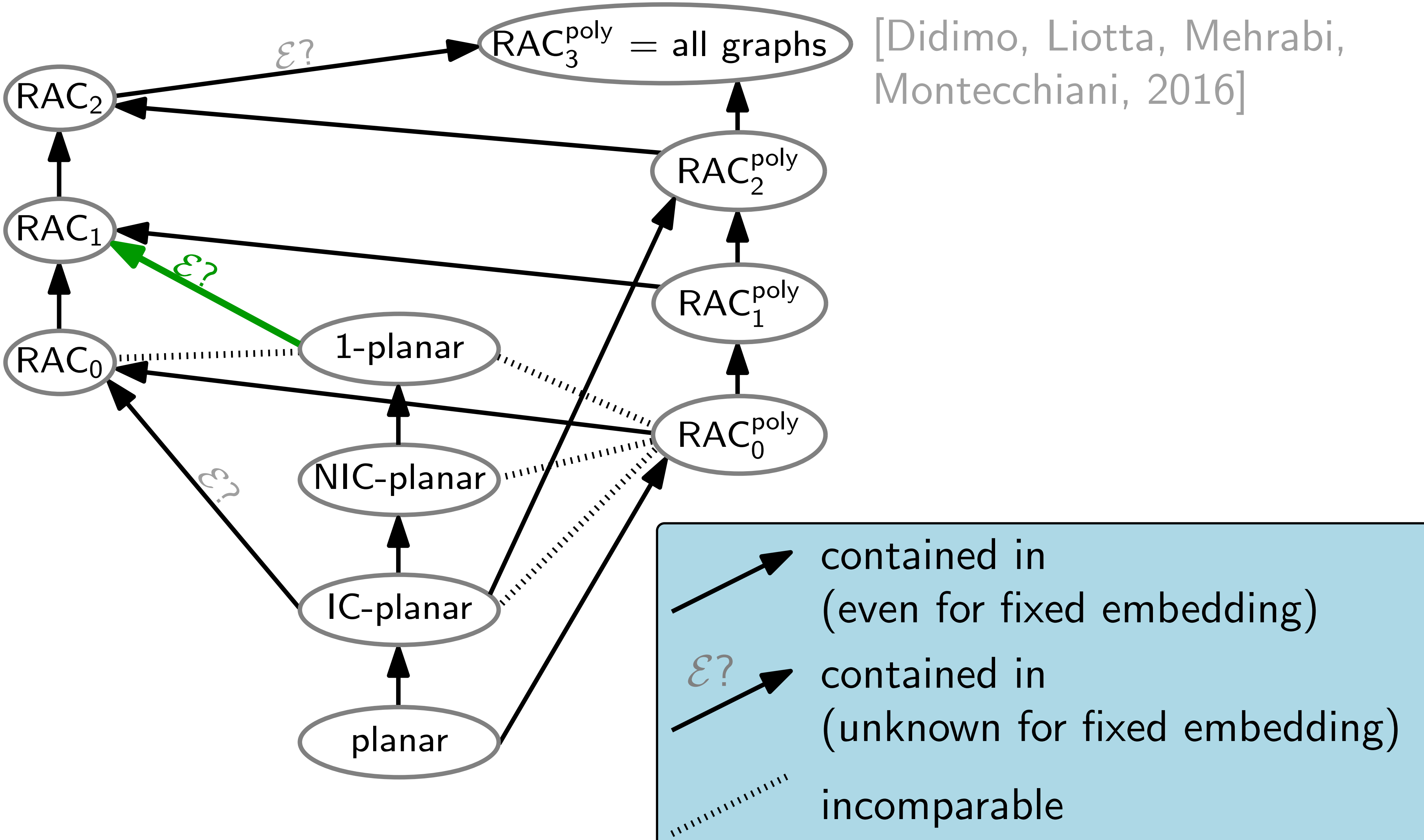


[Brandenburg, Didimo, Evans, Kindermann, Liotta, Montecchiani, 2015]

Related Work

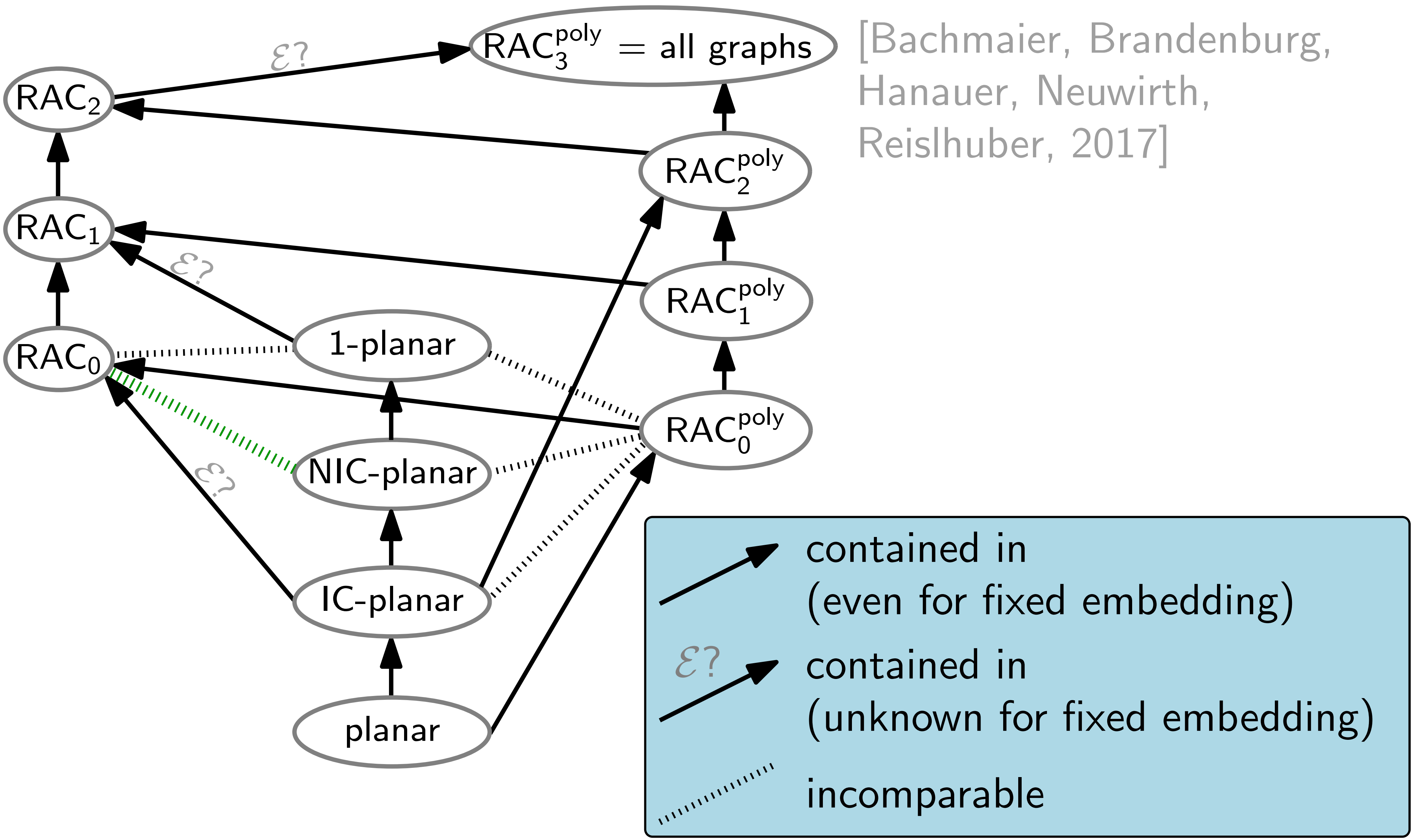


Related Work



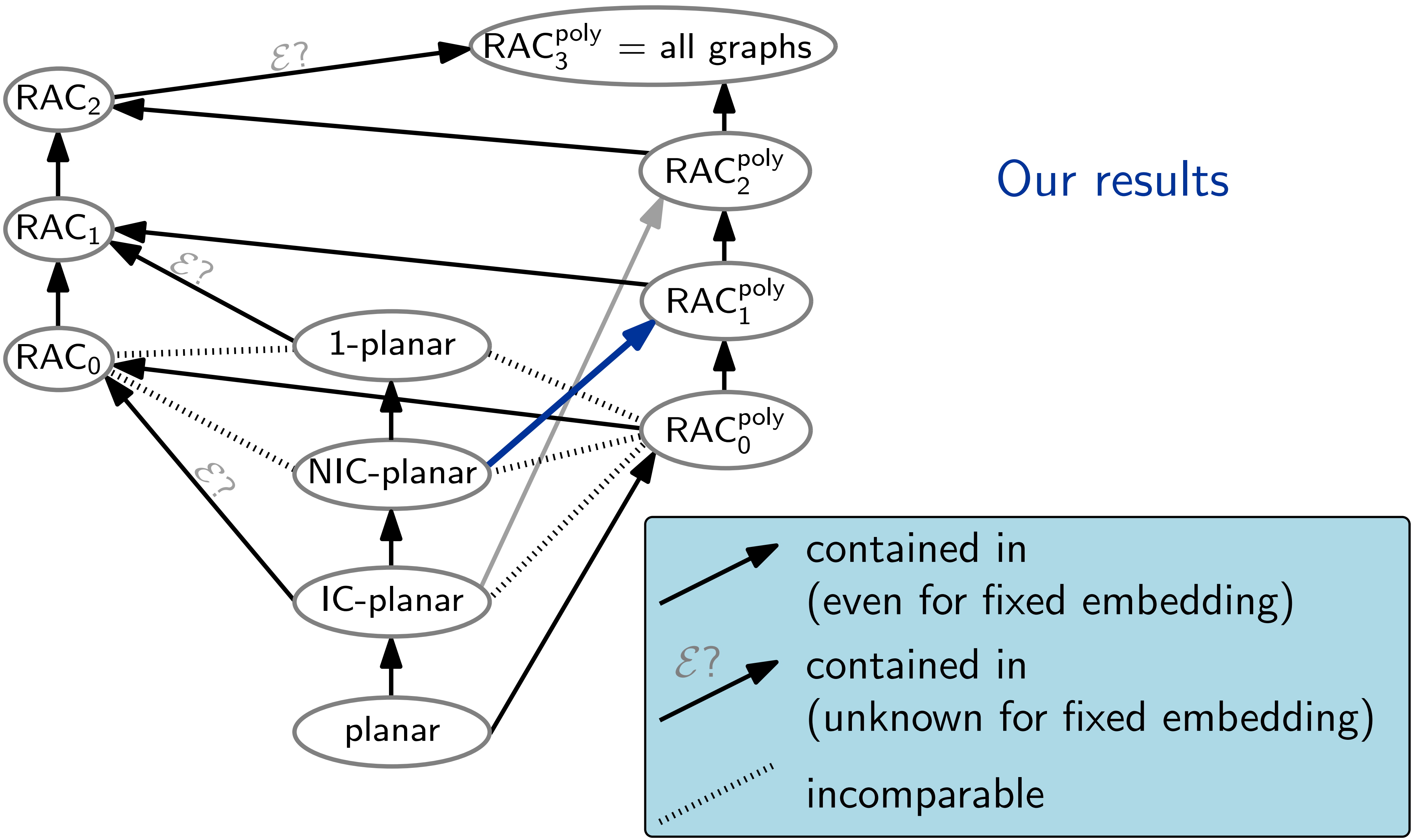
[Didimo, Liotta, Mehrabi, Montecchiani, 2016]

Related Work

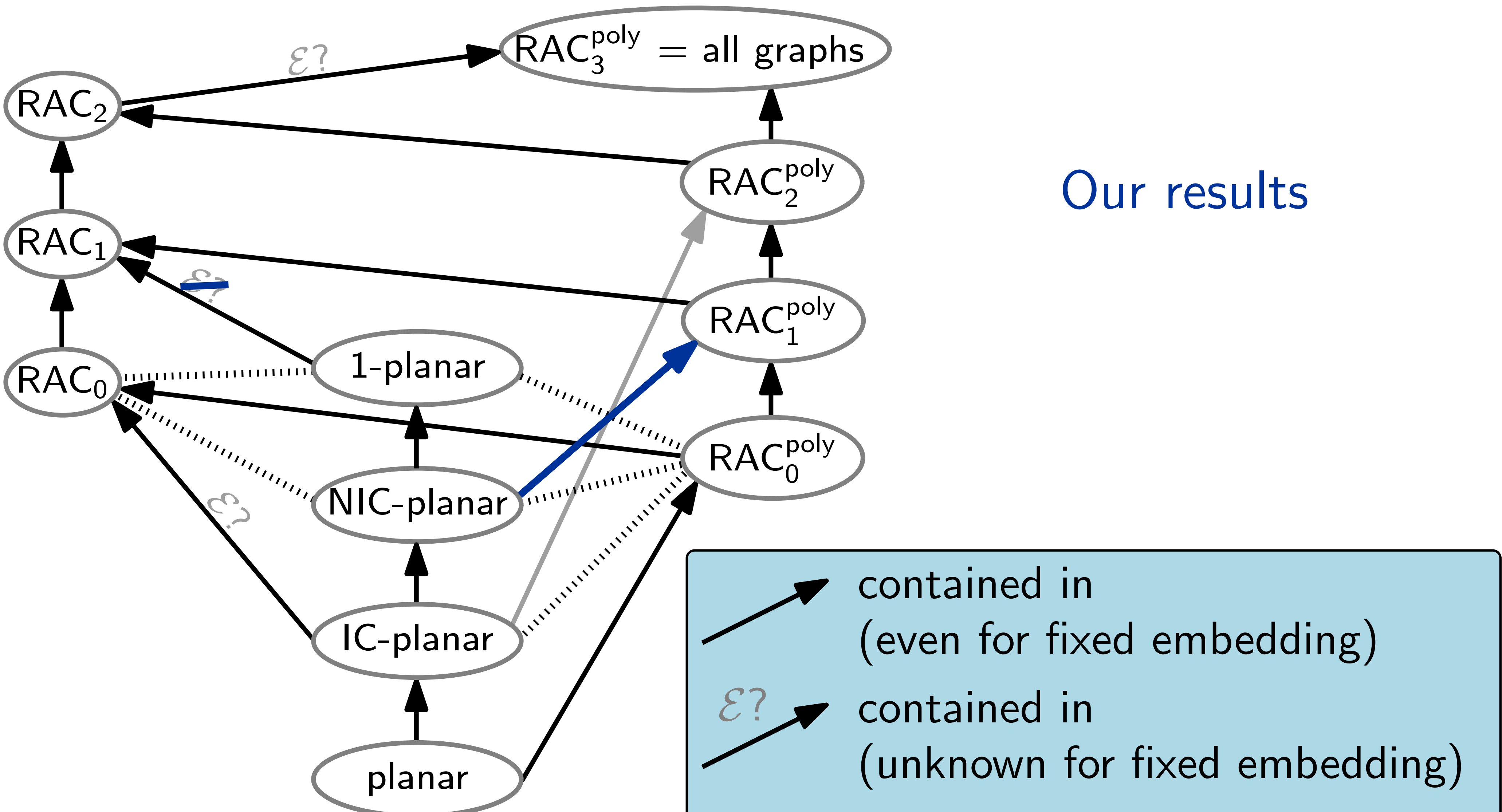


[Bachmaier, Brandenburg, Hanauer, Neuwirth, Reislhuber, 2017]

Related Work

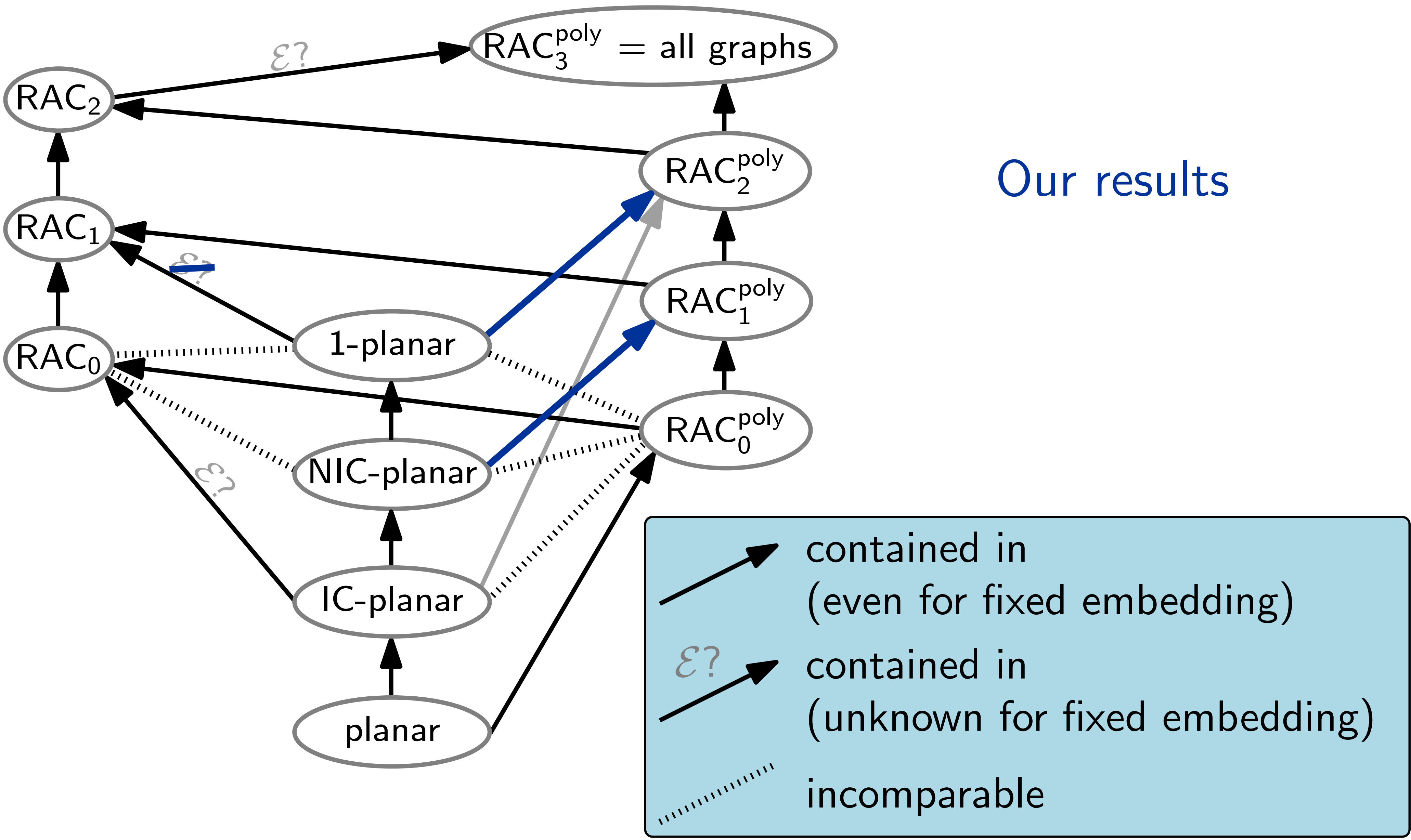


Related Work



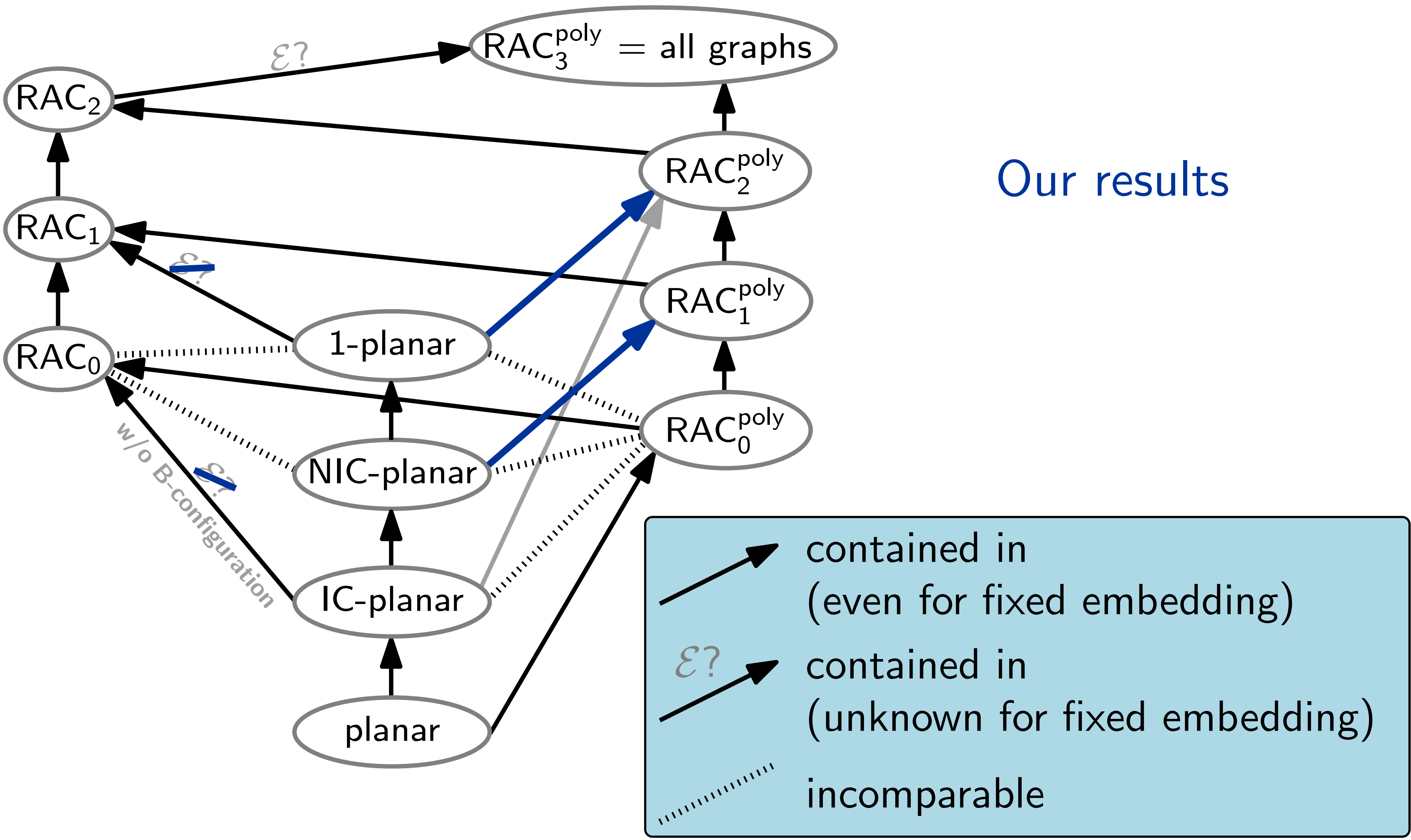
Our results

Related Work

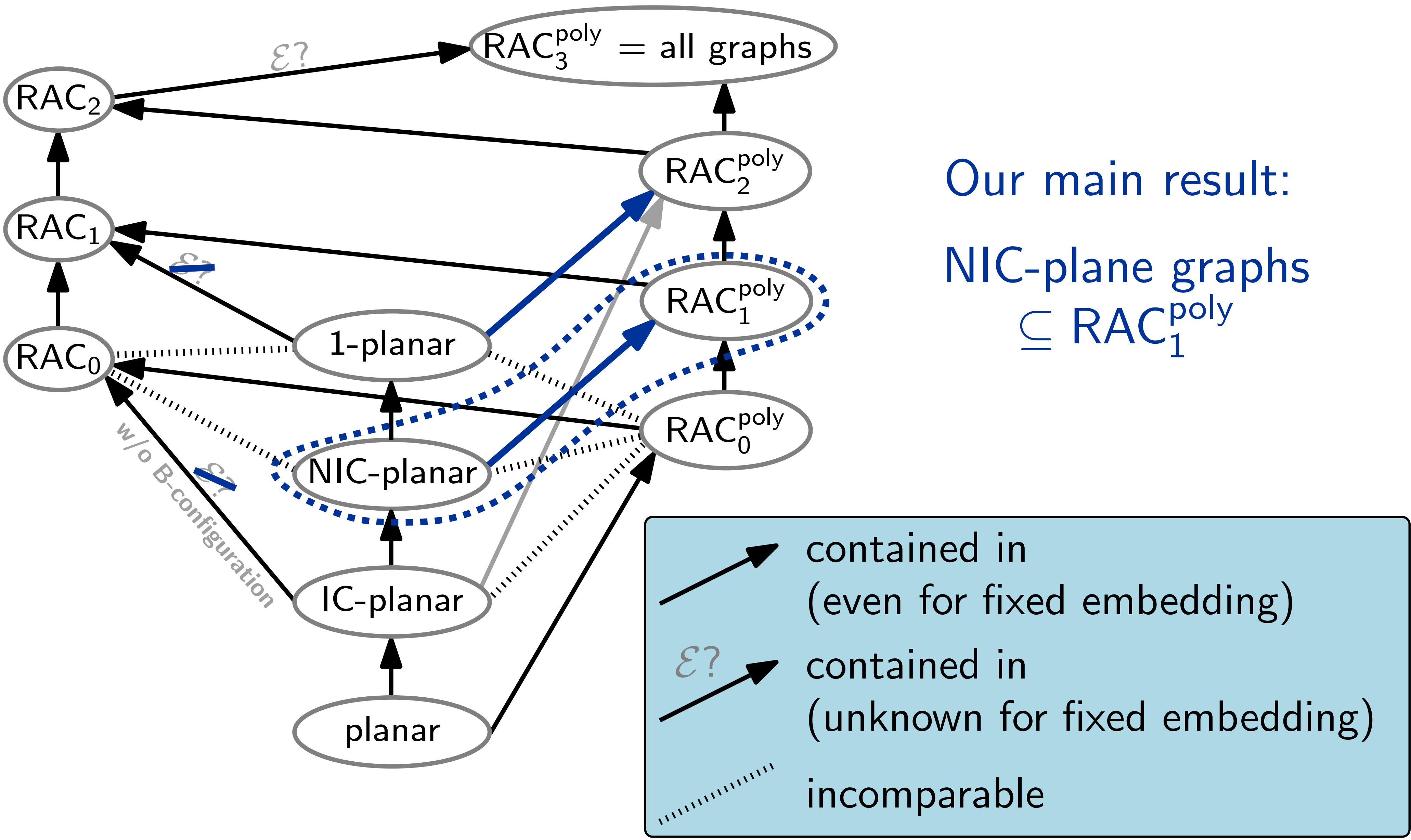


Our results

Related Work



Related Work



Our main result:

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

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

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

Algorithm
in $O(n)$ **time:**

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

Algorithm
in $O(n)$ time:

Input:
NIC-plane graph $(G, \mathcal{E})$ with n vertices

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

Algorithm
in $O(n)$ time:

Graph G with a NIC-planar embedding $\mathcal{E}$

Input:
NIC-plane graph $(G, \mathcal{E})$ with n vertices

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

Algorithm
in $O(n)$ time:

Graph G with a NIC-planar embedding $\mathcal{E}$

Input:
NIC-plane graph $(G, \mathcal{E})$ with n vertices

Output:
1-bend RAC drawing Γ of G according to $\mathcal{E}$
Every vertex, bend point, and crossing point of Γ lies on a grid of size $O(n) \times O(n)$

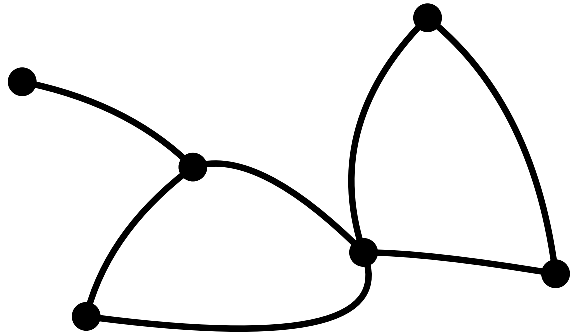
The Shift Algorithm

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

The Shift Algorithm

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

Idea:

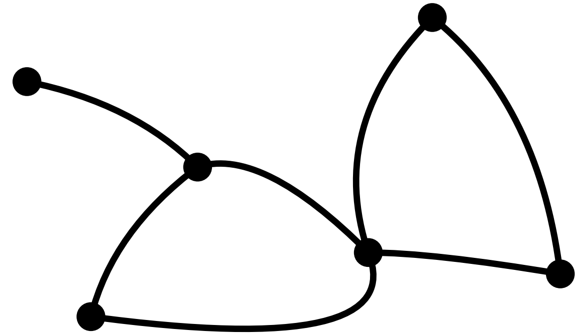


The Shift Algorithm

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

Idea:

- Triangulate given plane graph.

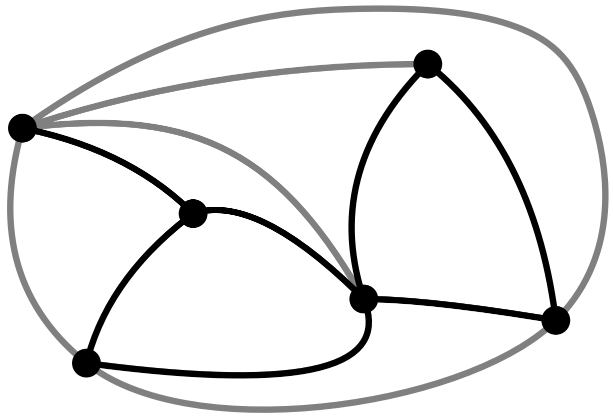


The Shift Algorithm

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

Idea:

- Triangulate given plane graph.

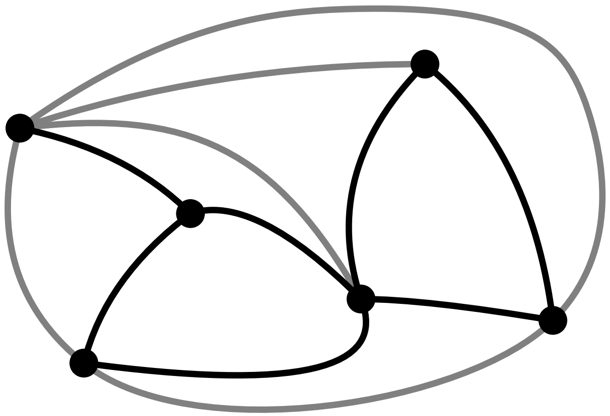


The Shift Algorithm

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

Idea:

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

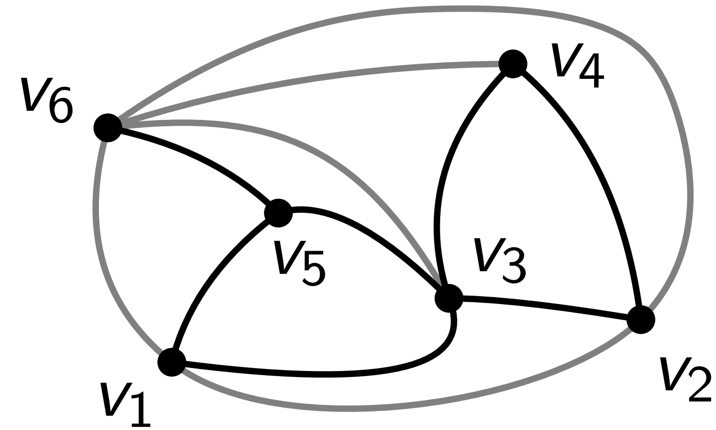


The Shift Algorithm

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

Idea:

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

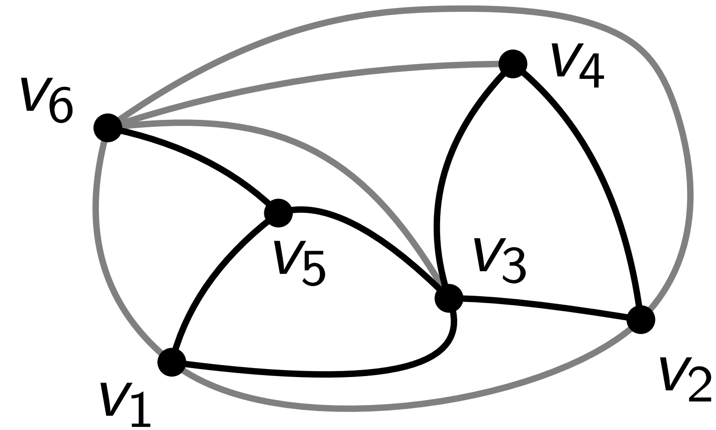


The Shift Algorithm

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

Idea:

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

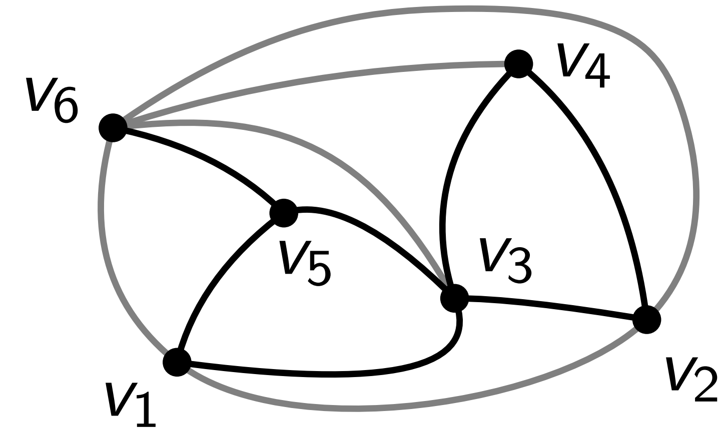


The Shift Algorithm

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

Idea:

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

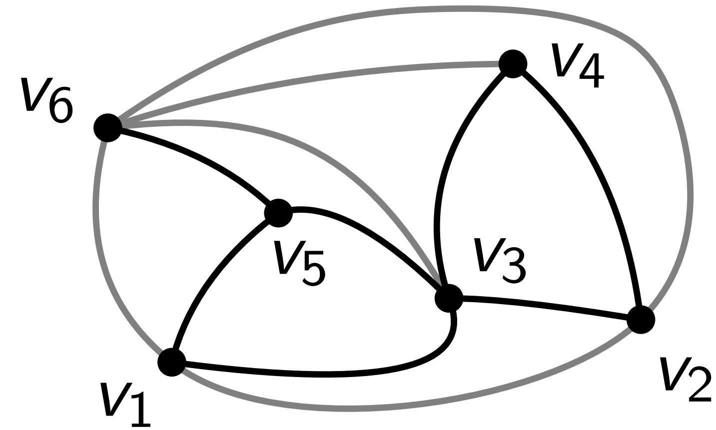


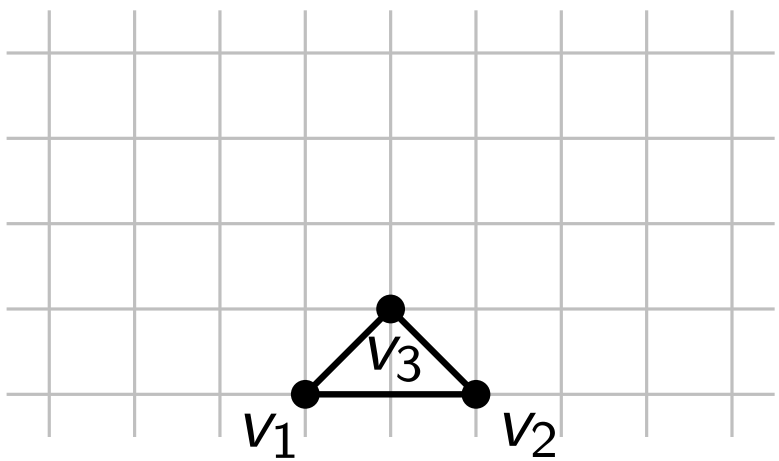
The Shift Algorithm

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

Idea:

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



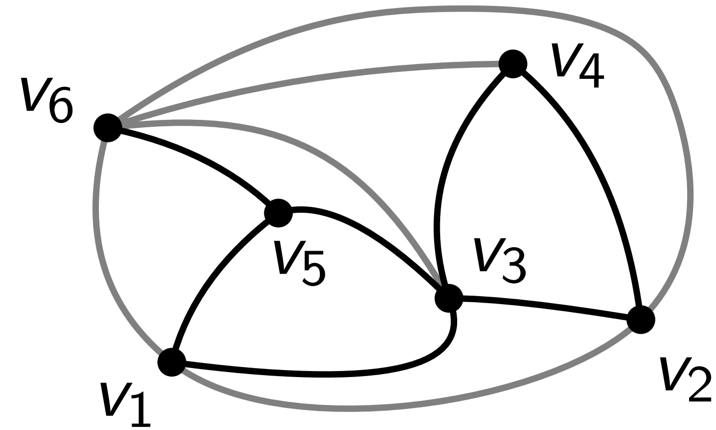


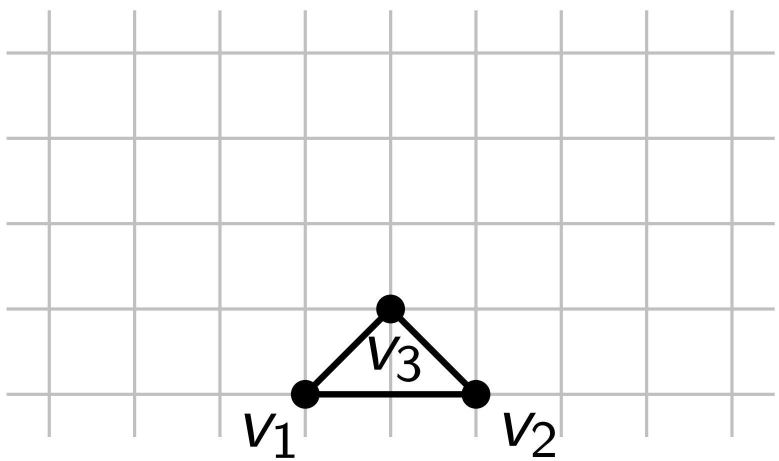
The Shift Algorithm

[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.



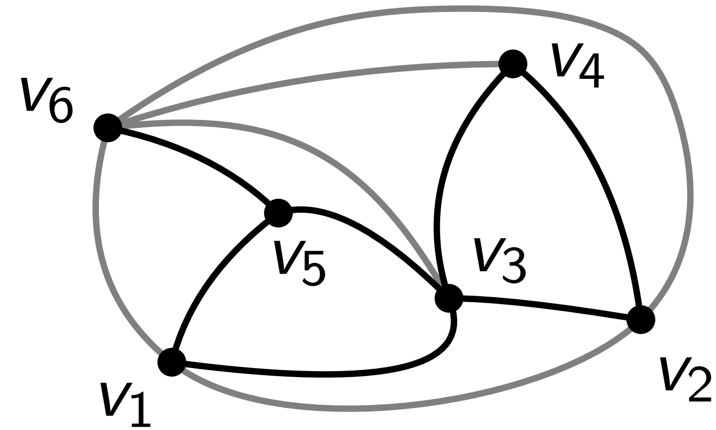


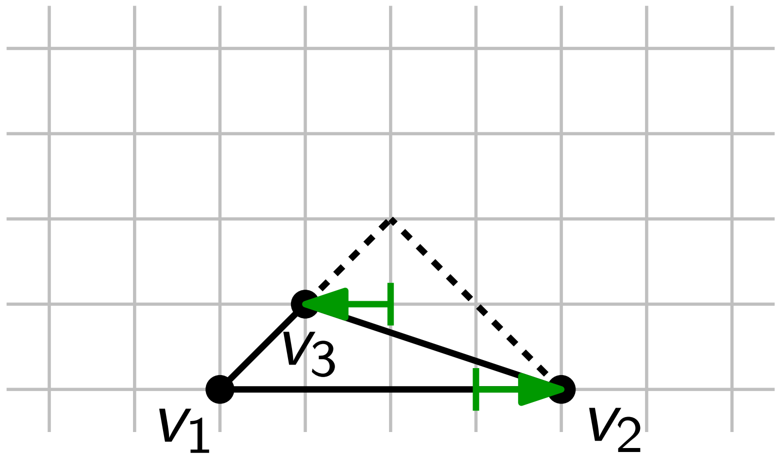
The Shift Algorithm

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

Idea:

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



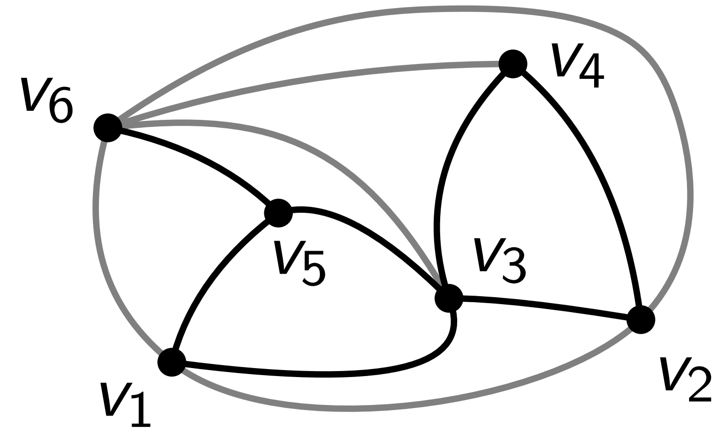


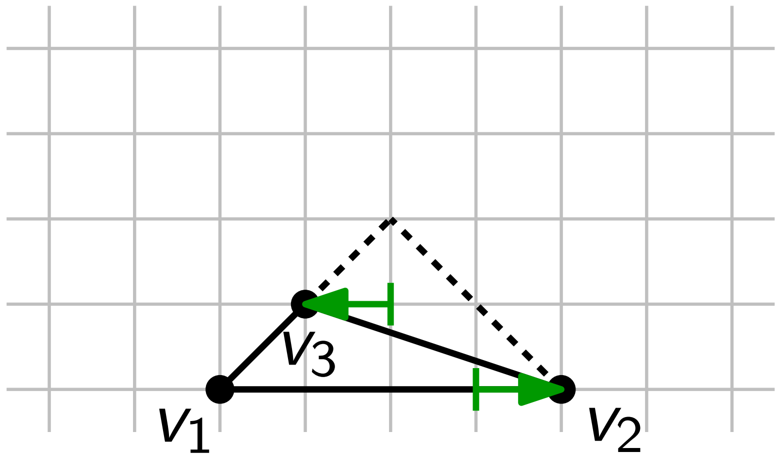
The Shift Algorithm

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

Idea:

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



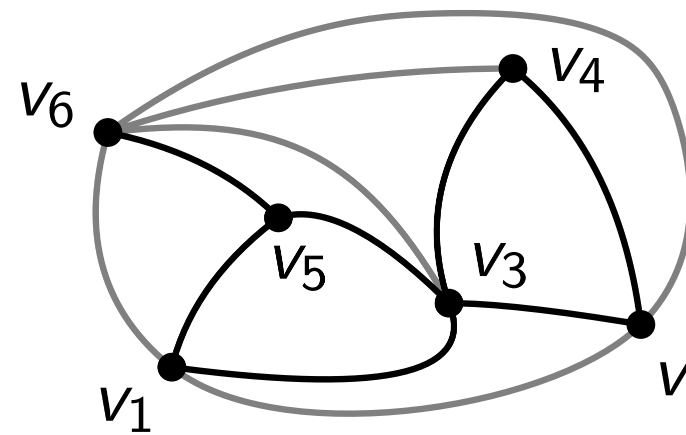


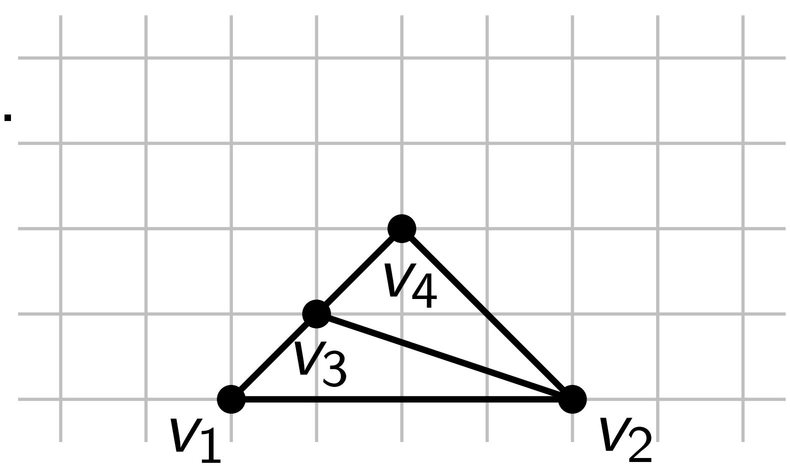
The Shift Algorithm

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

Idea:

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





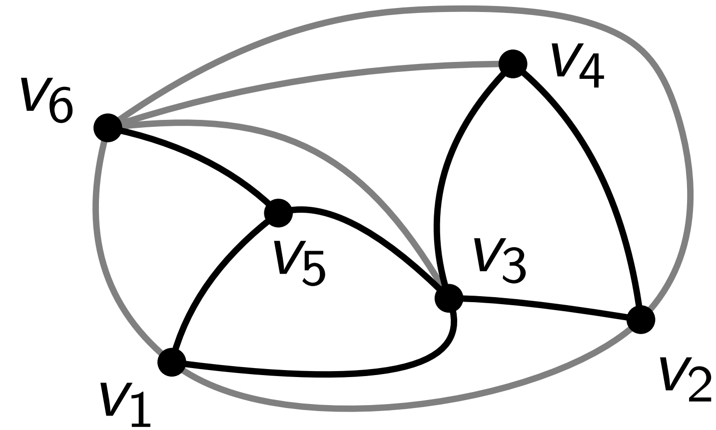
The Shift Algorithm

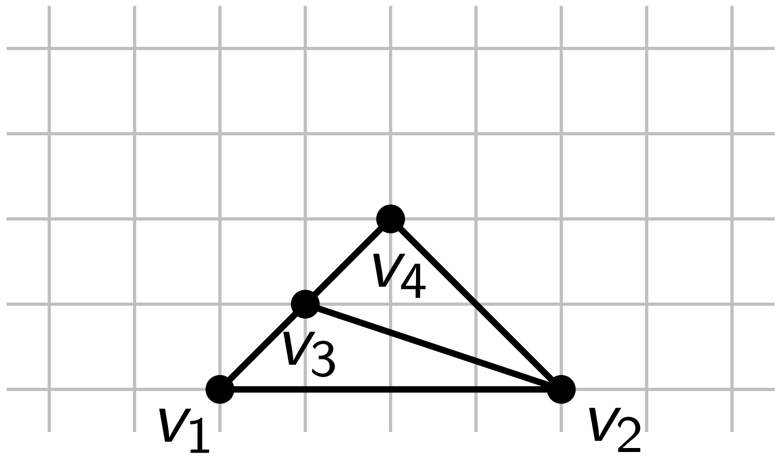
[de Fraysseix, Pach, and Pollack, 1990]
[Chrobak and Payne, 1995]

Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices $v_1, v_2, \ldots, v_n$.
- 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 ± 1 (except for $v_1 v_2$)





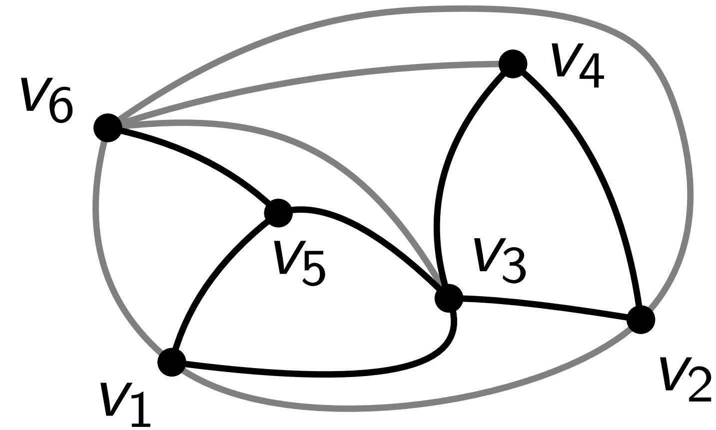
The Shift Algorithm

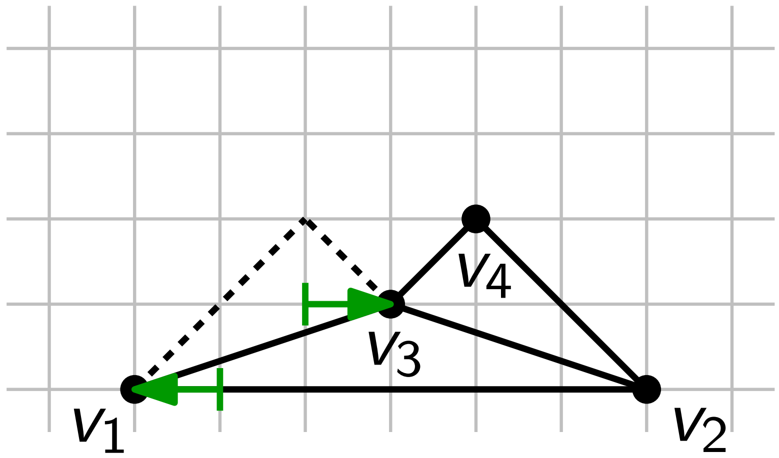
[de Fraysseix, Pach, and Pollack, 1990]
[Chrobak and Payne, 1995]

Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices $v_1, v_2, \ldots, v_n$.
- 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 ± 1 (except for $v_1 v_2$)





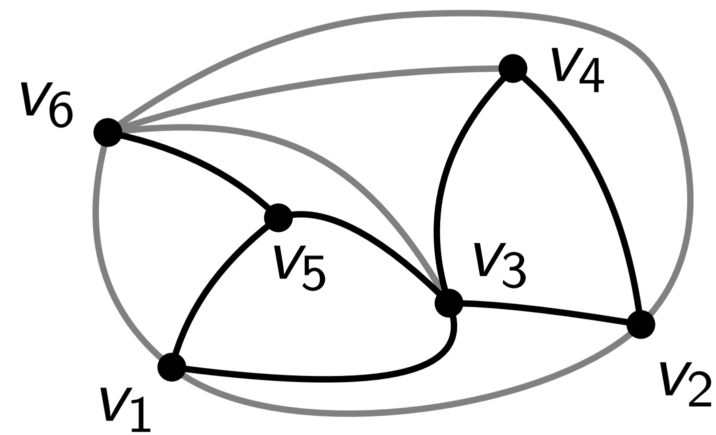
The Shift Algorithm

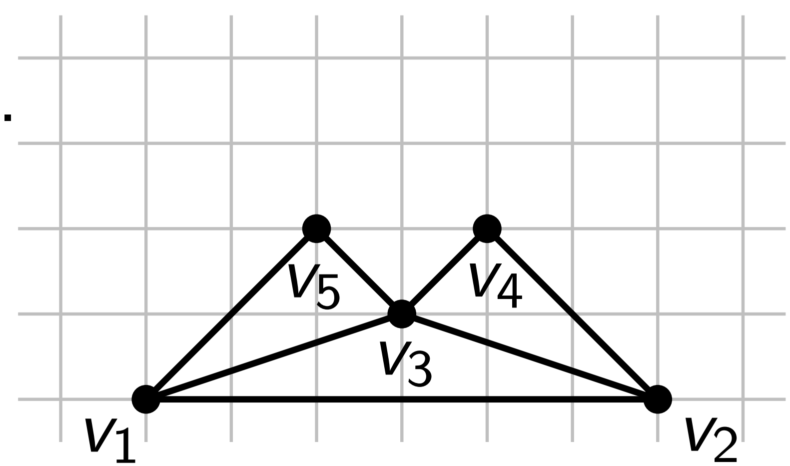
[de Fraysseix, Pach, and Pollack, 1990]
[Chrobak and Payne, 1995]

Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices $v_1, v_2, \ldots, v_n$.
- 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 ± 1
  (except for $v_1 v_2$)





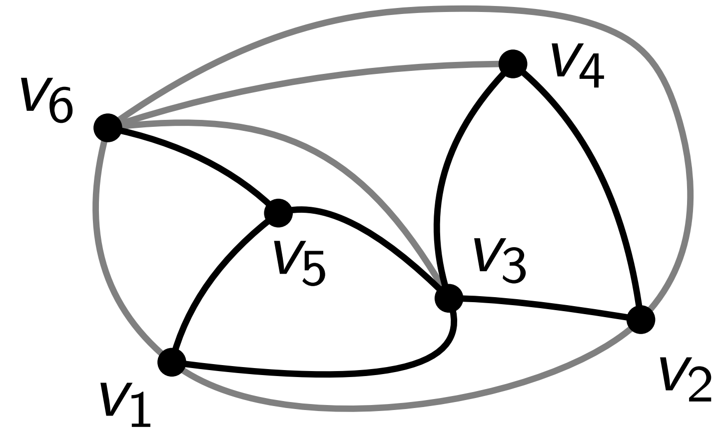
The Shift Algorithm

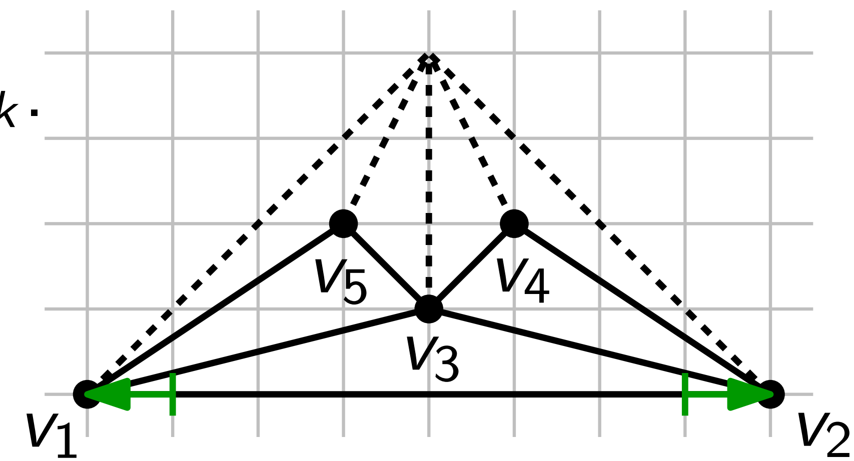
[de Fraysseix, Pach, and Pollack, 1990]
[Chrobak and Payne, 1995]

Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices $v_1, v_2, \ldots, v_n$.
- 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 ± 1 (except for $v_1 v_2$)





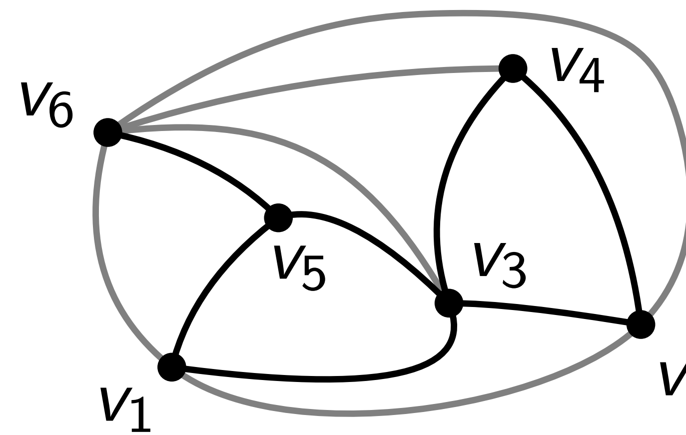
The Shift Algorithm

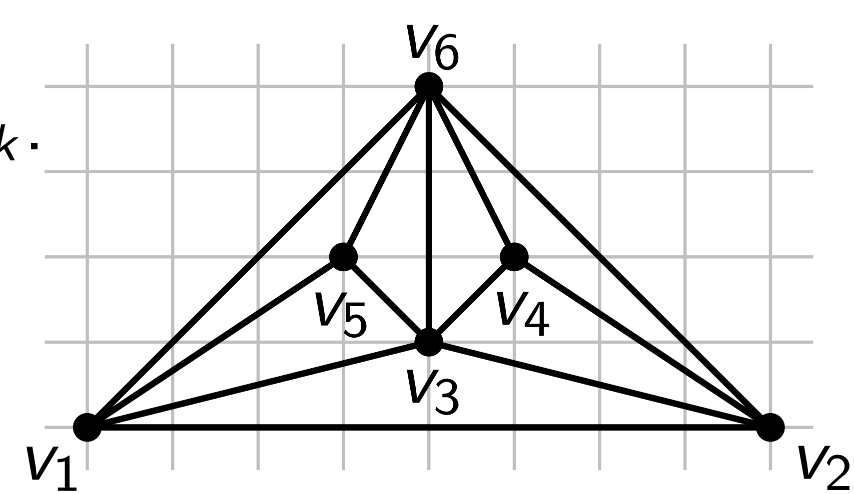
[de Fraysseix, Pach, and Pollack, 1990]
[Chrobak and Payne, 1995]

Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices $v_1, v_2, \ldots, v_n$.
- 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 ± 1 (except for $v_1 v_2$)





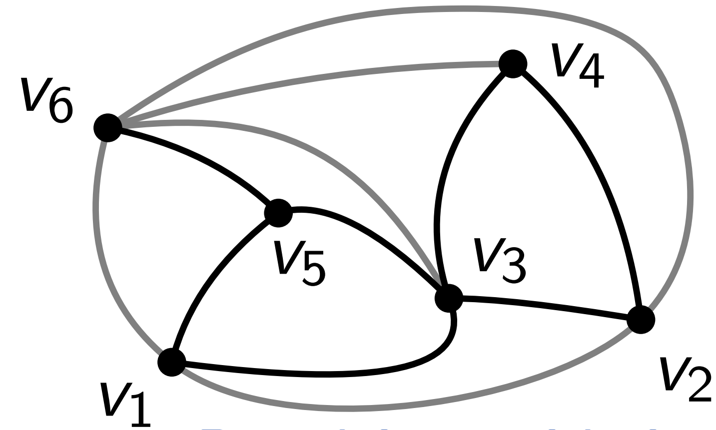
The Shift Algorithm

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

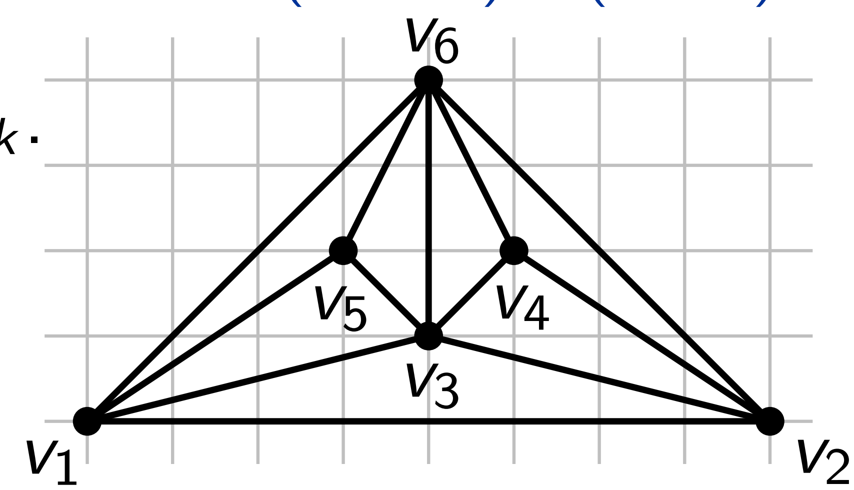
Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices $v_1, v_2, \ldots, v_n$.
- 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 ± 1 (except for $v_1 v_2$)



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



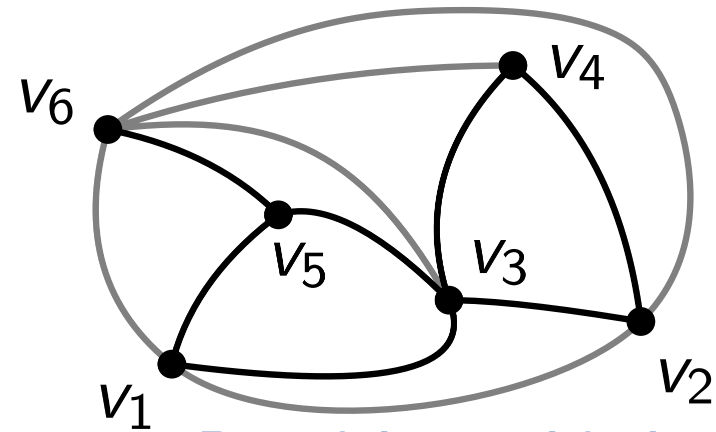
The Shift Algorithm

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

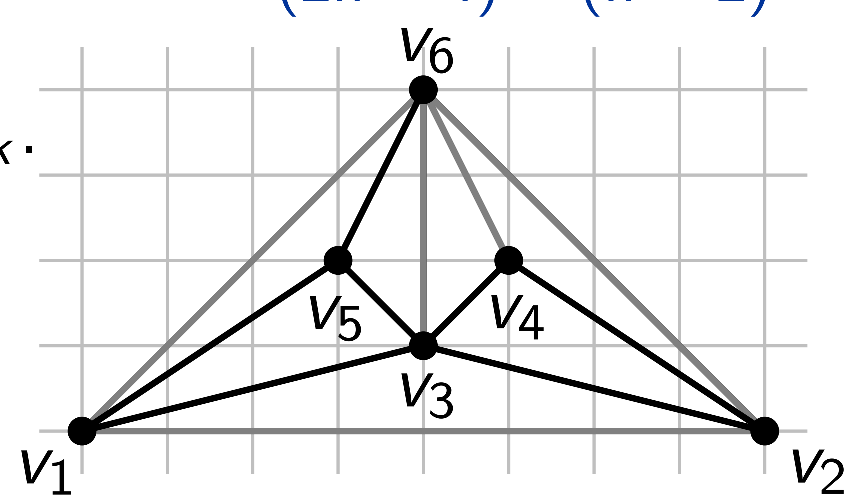
Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices $v_1, v_2, \ldots, v_n$.
- 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 ± 1 (except for $v_1 v_2$)



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



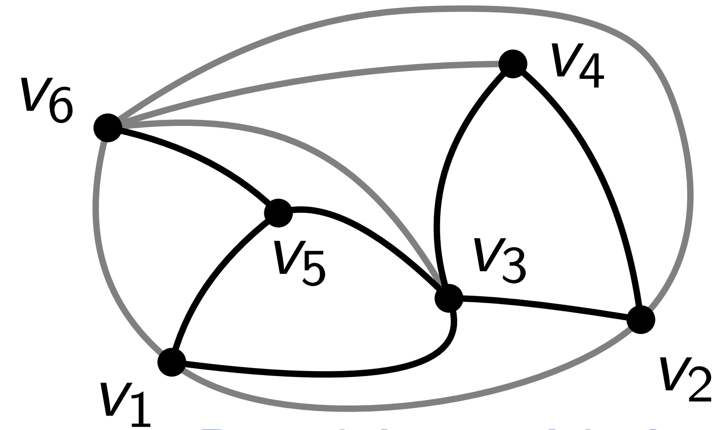
The Shift Algorithm

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

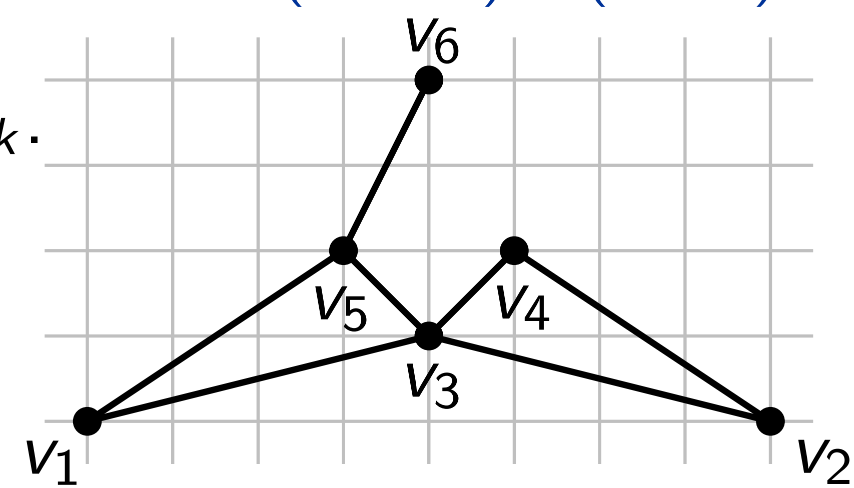
Idea:

- Triangulate given plane graph.
- Compute a canonical ordering of the vertices $v_1, v_2, \ldots, v_n$.
- 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 ± 1 (except for $v_1 v_2$)



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



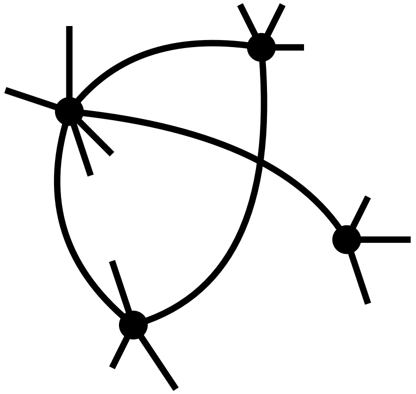
Approach that Nearly Works

Approach that Nearly Works

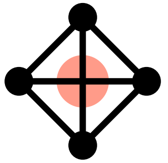
- Input: a NIC-plane graph

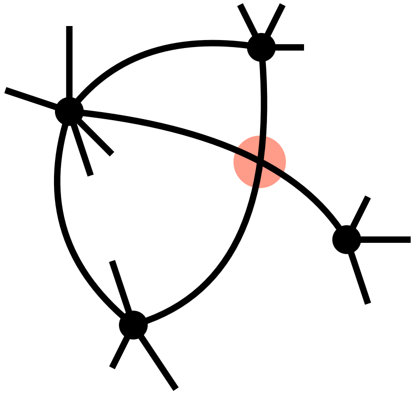
Approach that Nearly Works

- Input: a NIC-plane graph

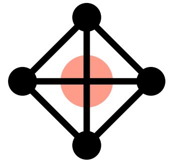


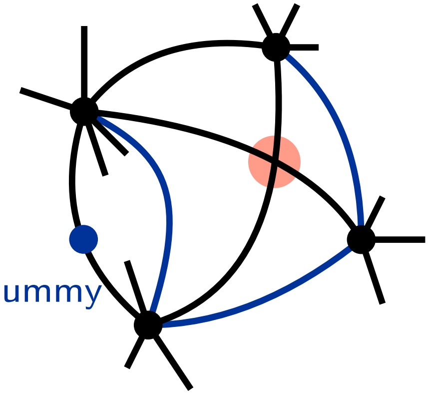
Approach that Nearly Works

- Input: a NIC-plane graph
- Enclose each crossing by a so called *empty kite* ()

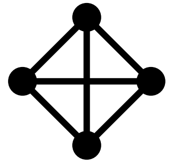


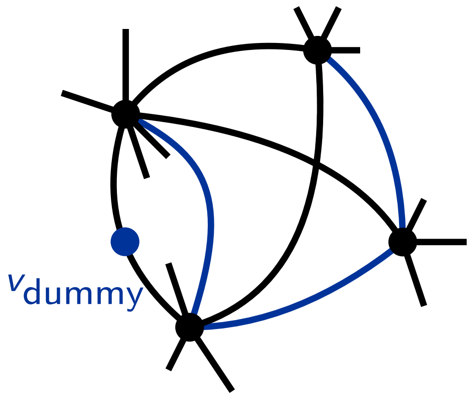
Approach that Nearly Works

- Input: a NIC-plane graph
- Enclose each crossing by a so called *empty kite* ()

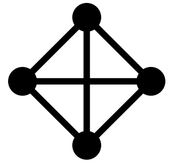


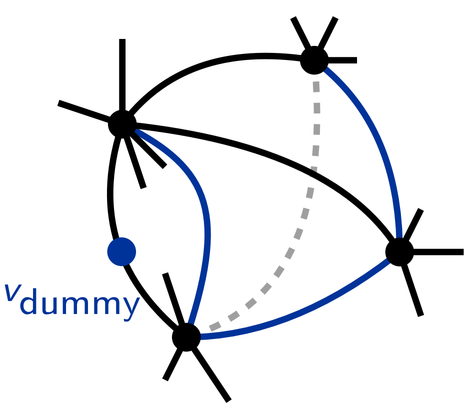
Approach that Nearly Works

- Input: a NIC-plane graph
- Enclose each crossing by a so called *empty kite* ()
- Replace each pair of crossing edges by a single edge

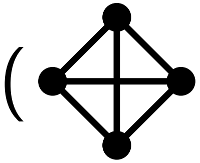


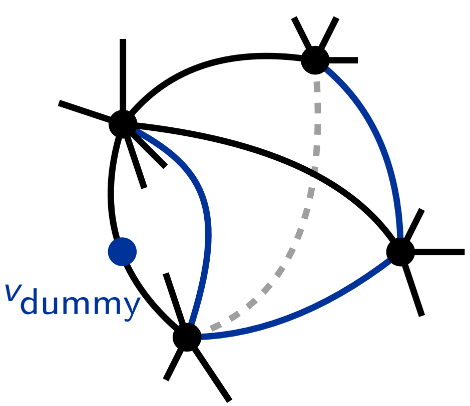
Approach that Nearly Works

- Input: a NIC-plane graph
- Enclose each crossing by a so called *empty kite* ()
- Replace each pair of crossing edges by a single edge

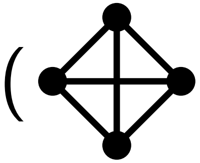


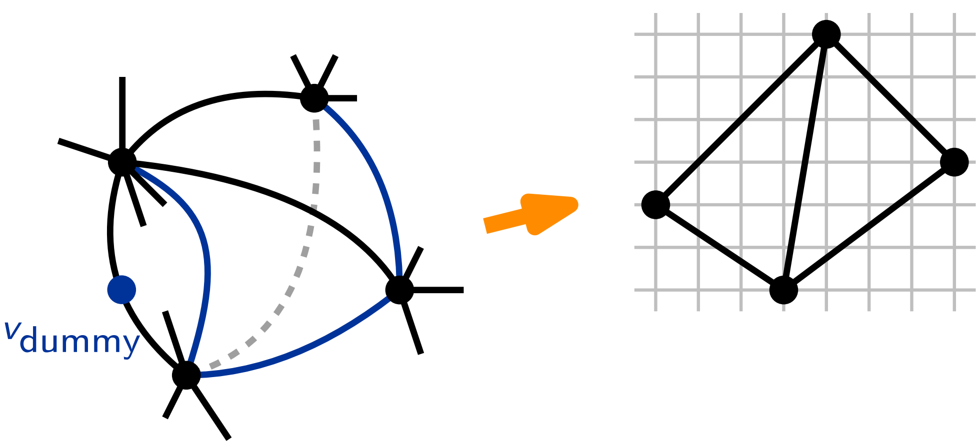
Approach that Nearly Works

- Input: a NIC-plane graph
- 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

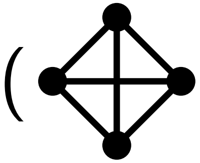


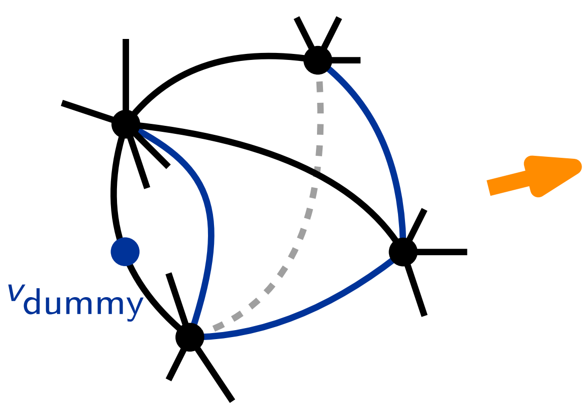
Approach that Nearly Works

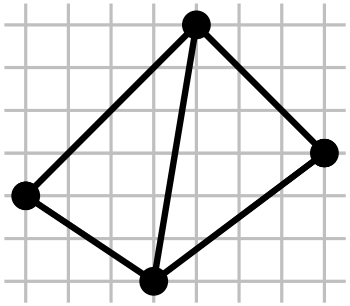
- Input: a NIC-plane graph
- Enclose each crossing by a so called *empty kite* ()
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm



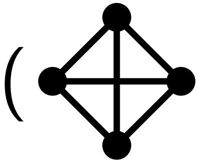
Approach that Nearly Works

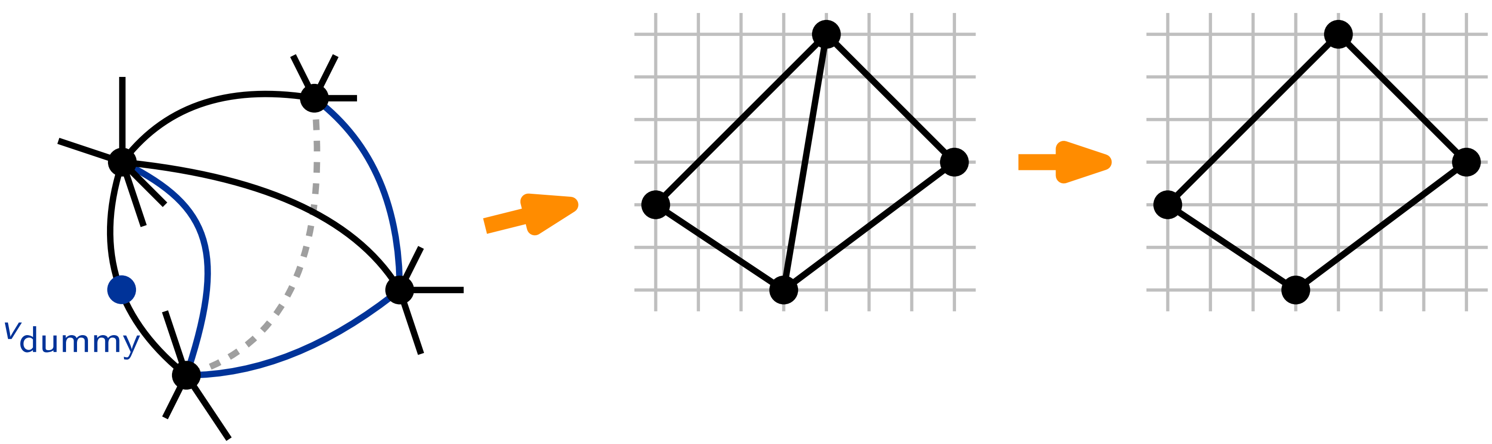
- Input: a NIC-plane graph
- Enclose each crossing by a so called *empty kite* ()
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends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



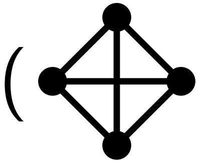


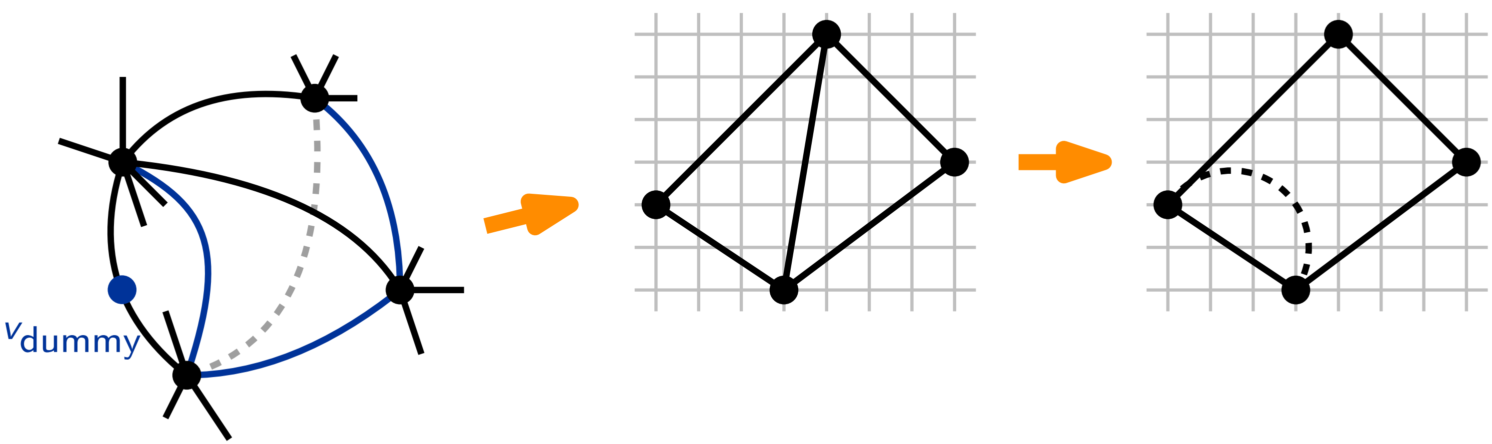
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

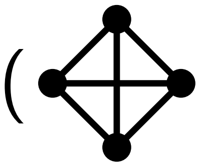


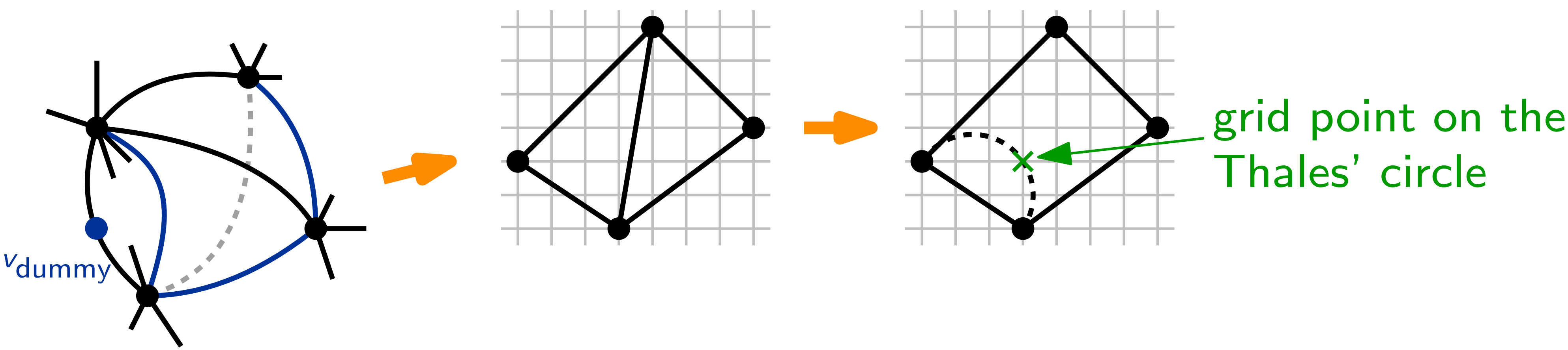
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

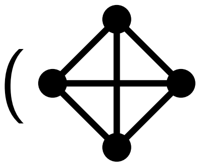


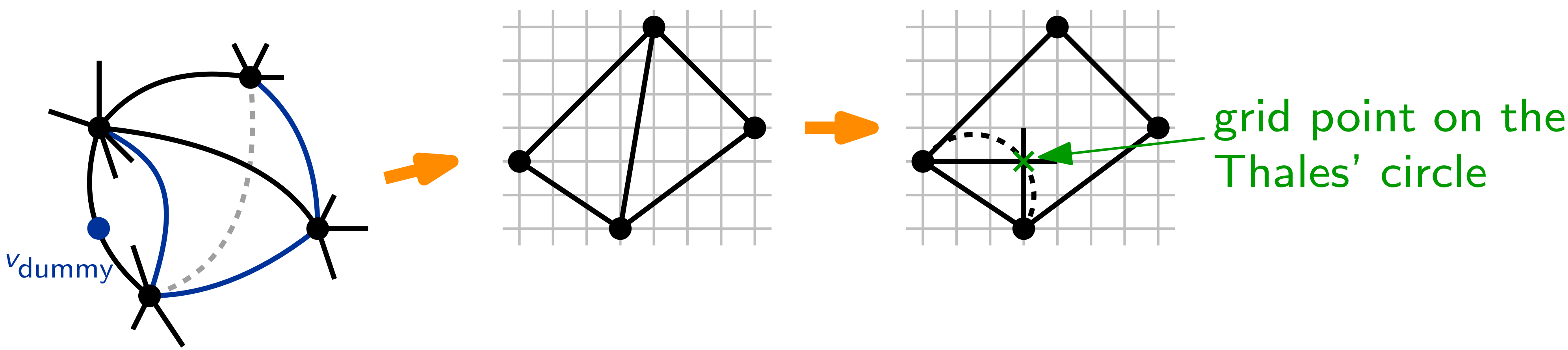
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

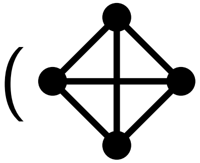


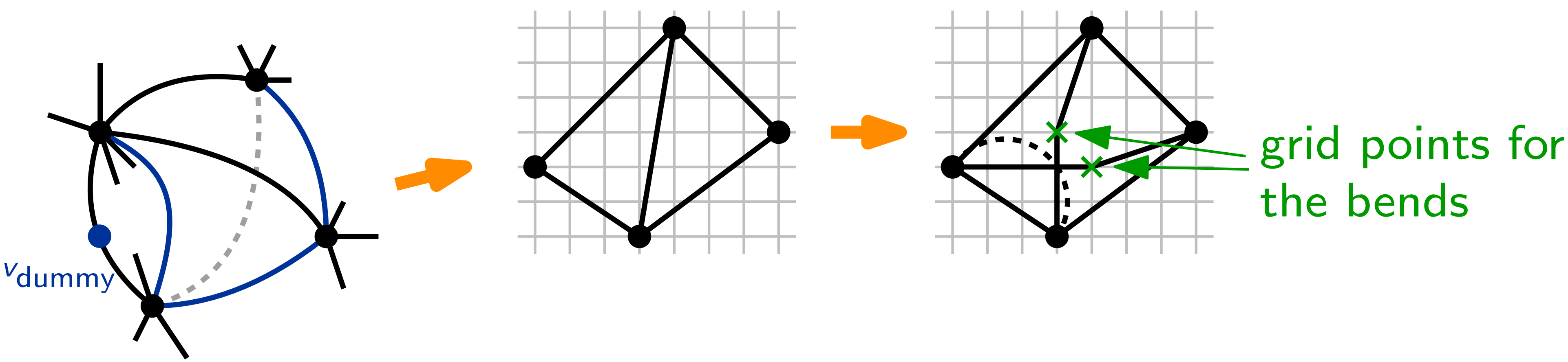
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

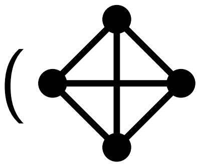


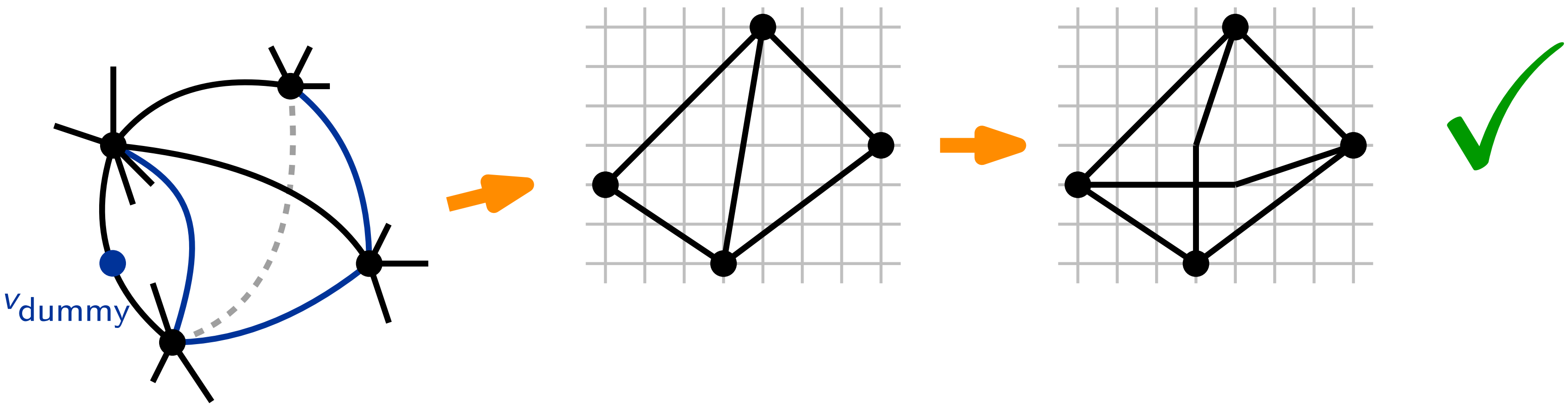
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

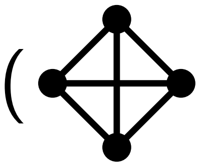


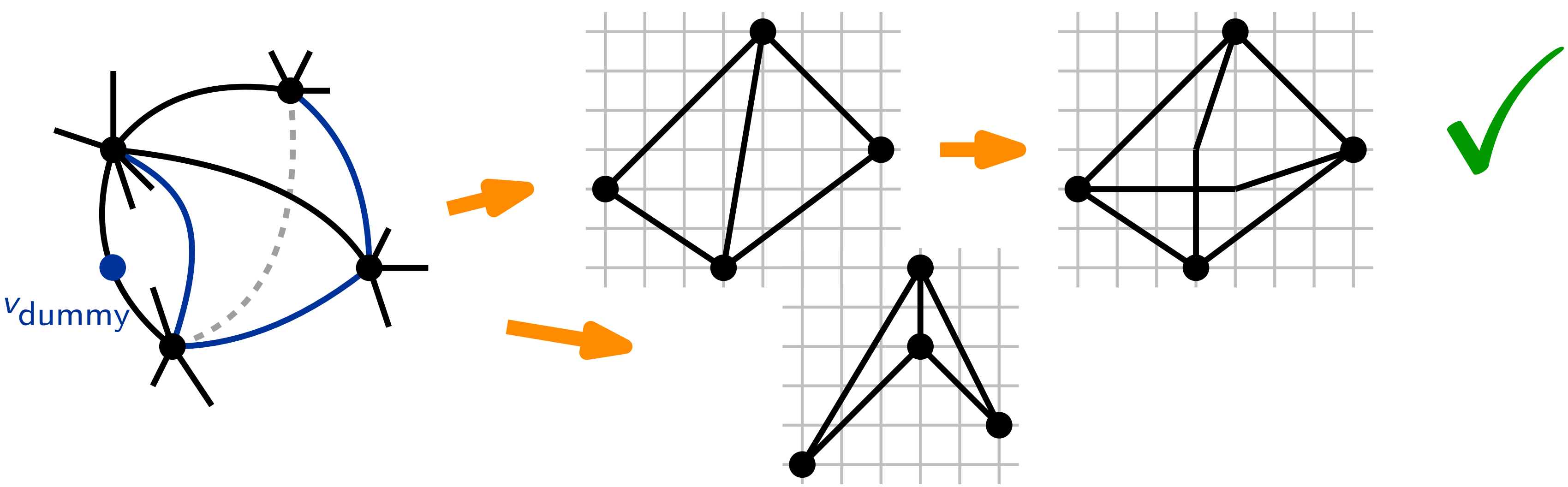
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

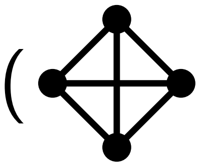


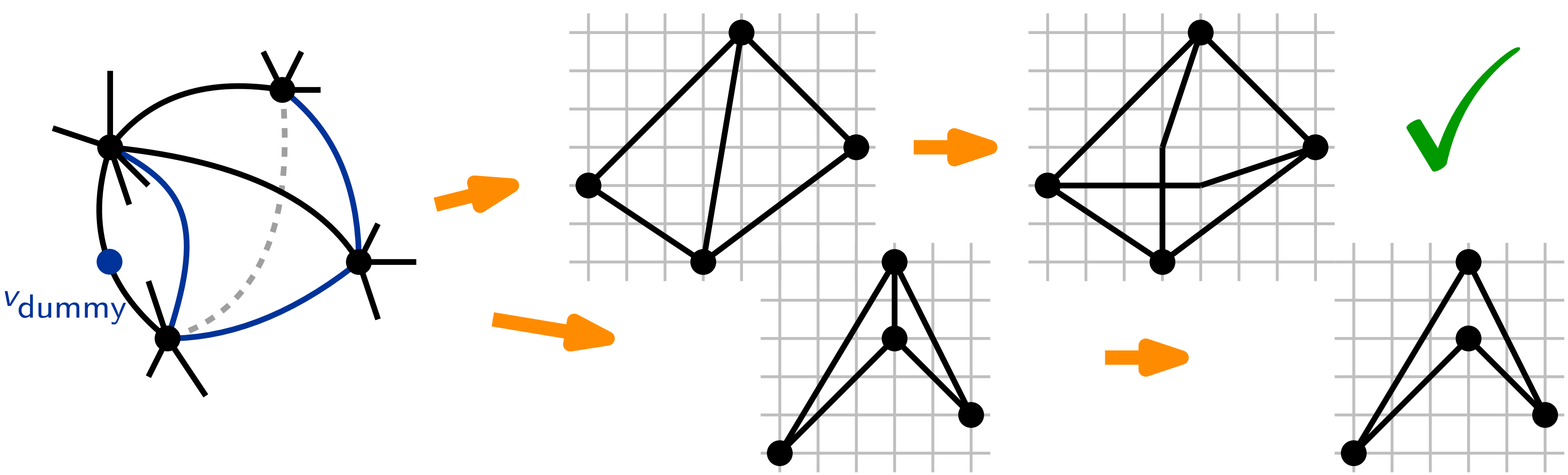
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

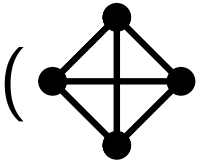


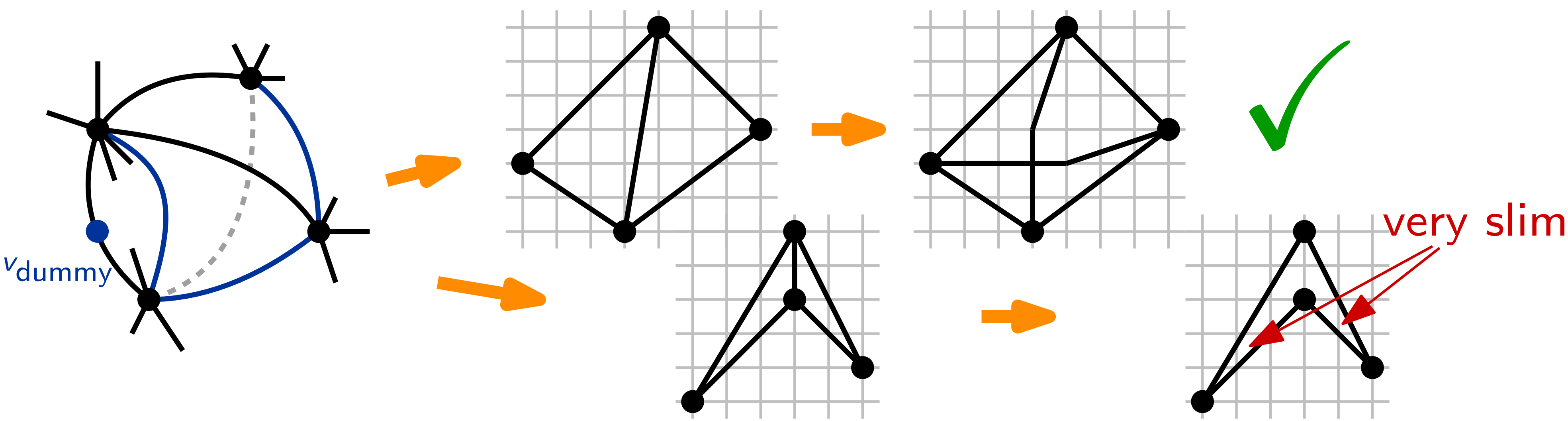
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

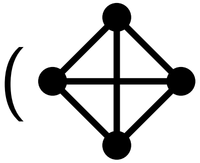


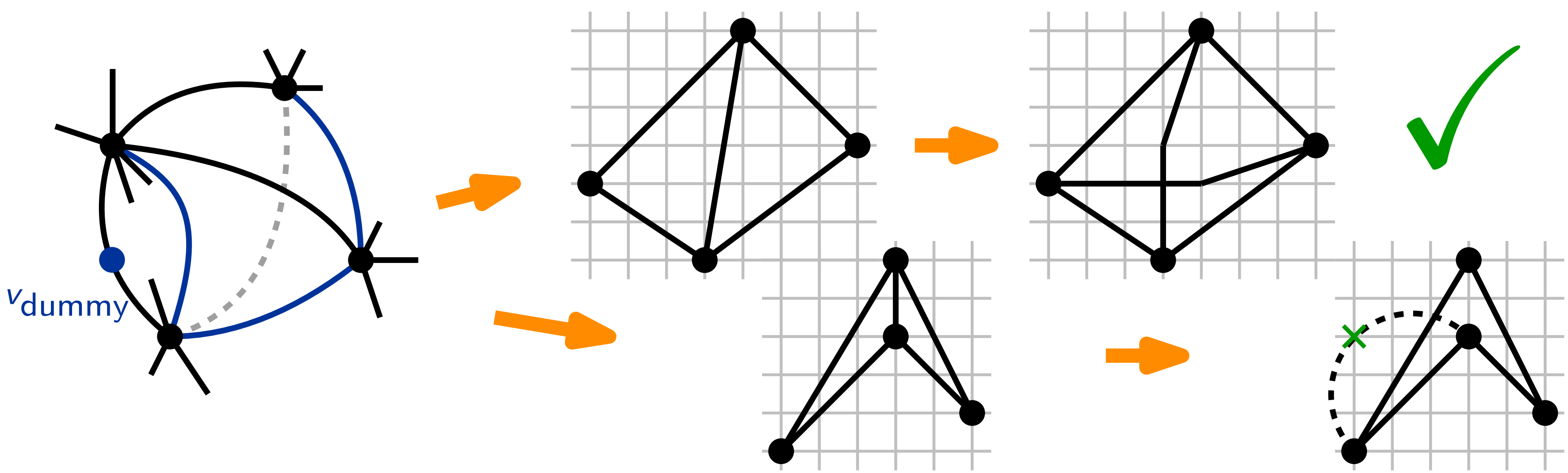
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

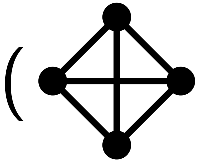


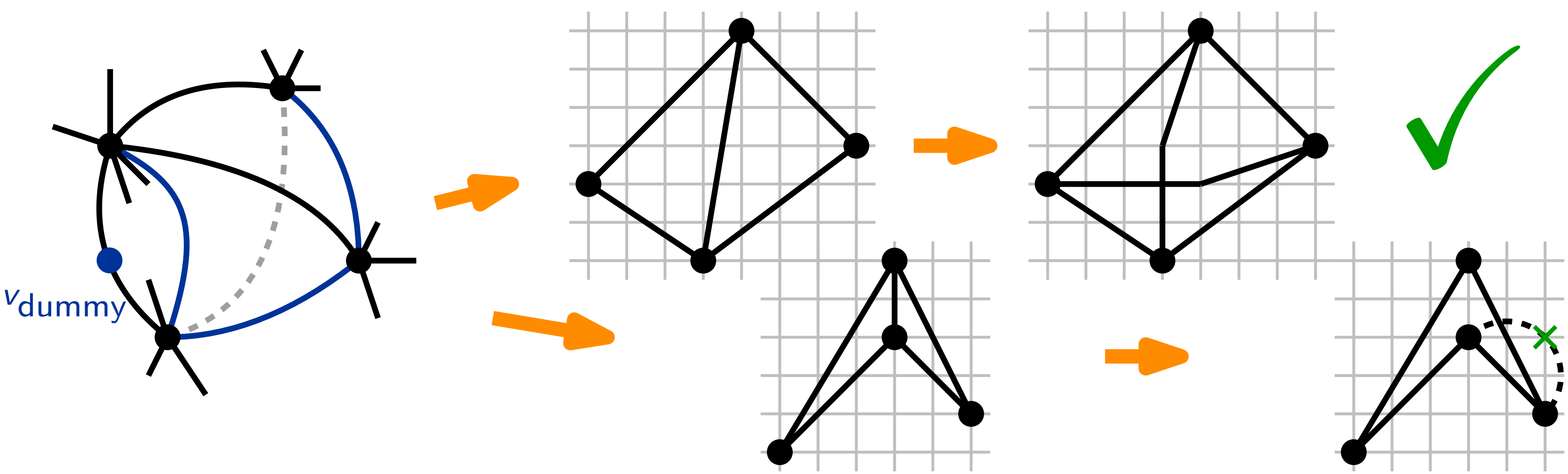
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

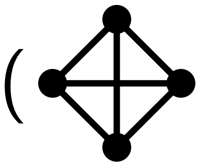


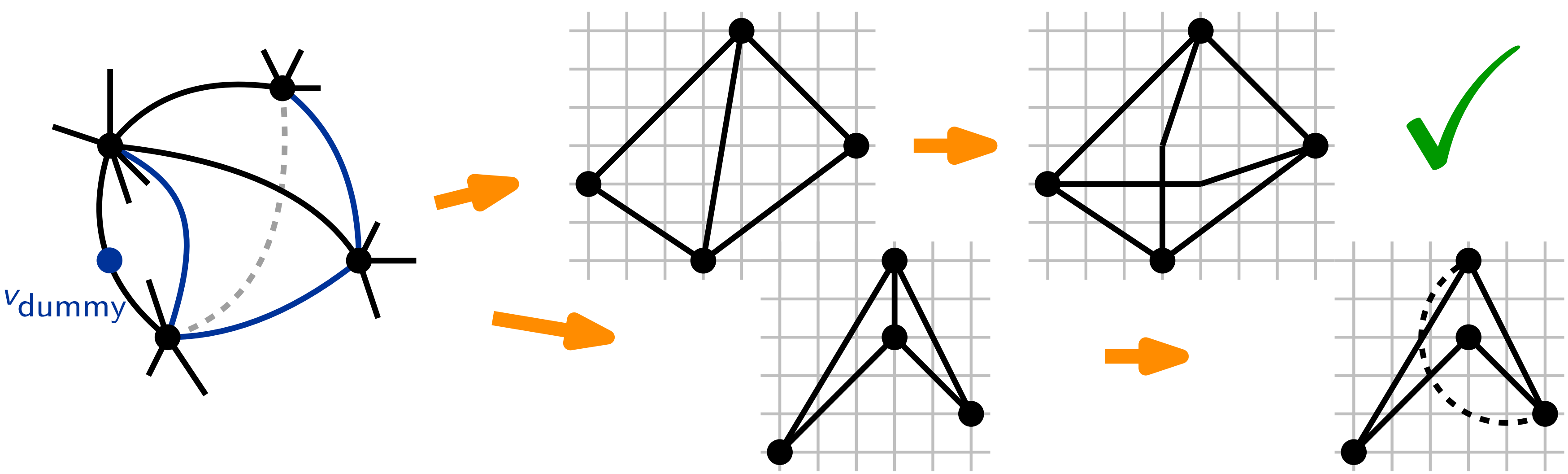
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

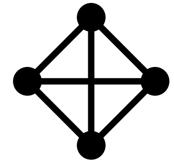


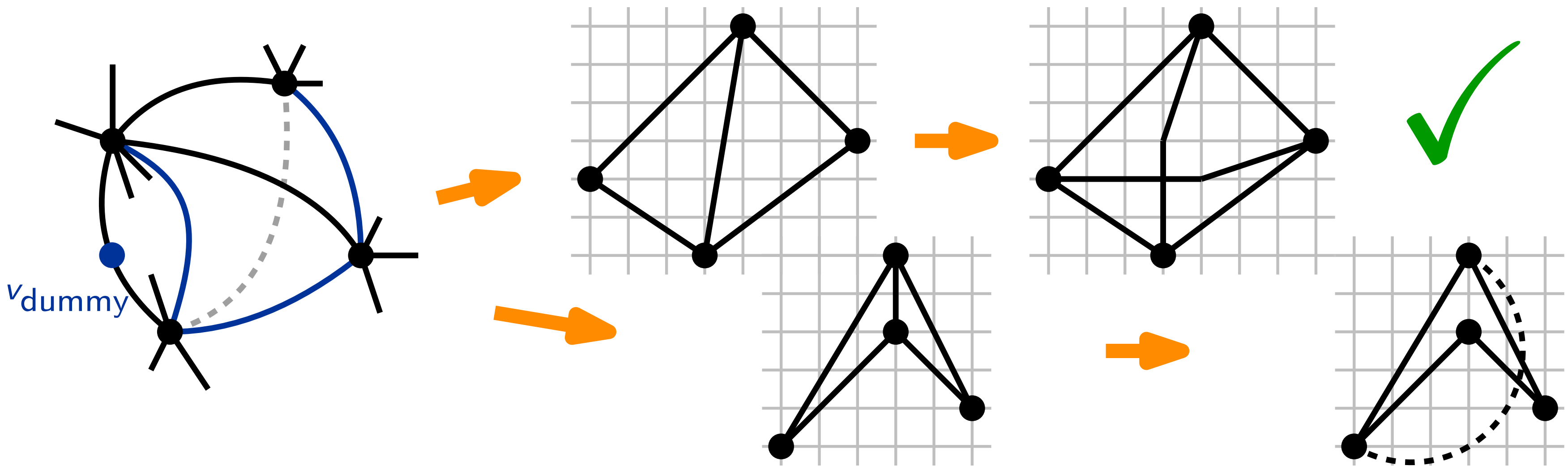
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

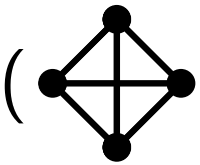


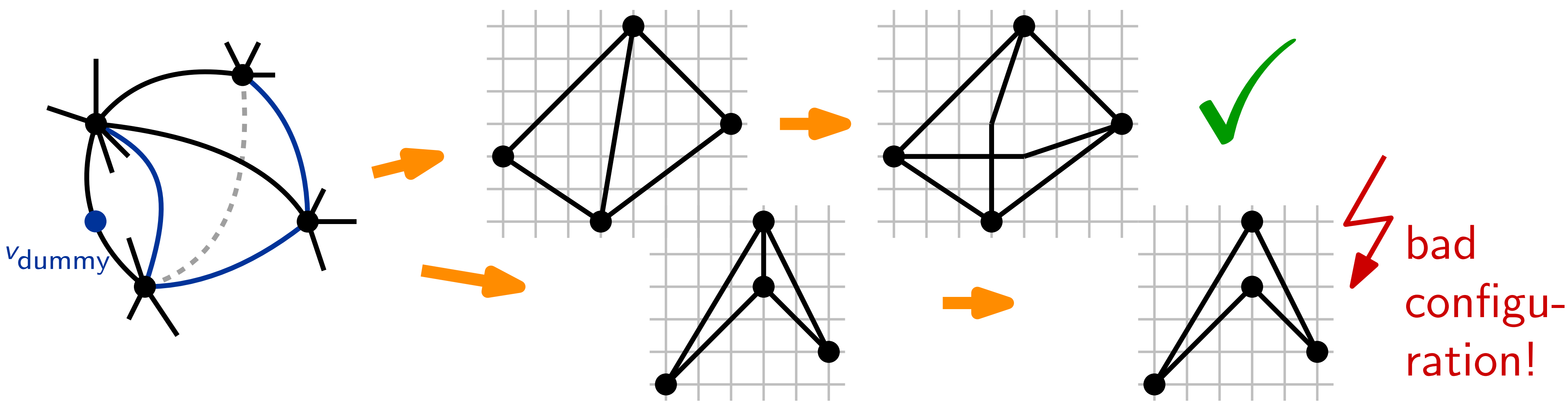
Approach that Nearly Works

- Input: a NIC-plane graph
- 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)

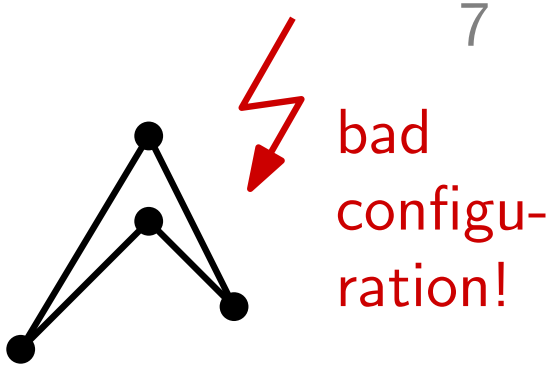


Approach that Nearly Works

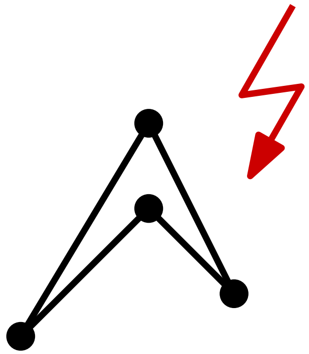
- Input: a NIC-plane graph
- Enclose each crossing by a so called *empty kite* ()
- Replace each pair of crossing edges by a single edge
- Draw the obtained plane graph with the Shift Algorithm
- Manually reinsert the removed edges with 1 bend so that they cross in a right angle (crossings and bends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



Our Algorithm



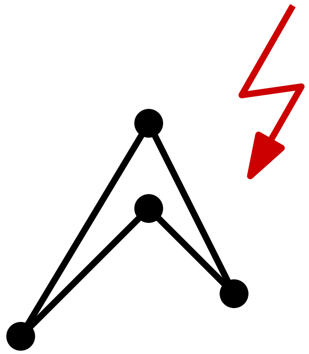
Our Algorithm



Solution:

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

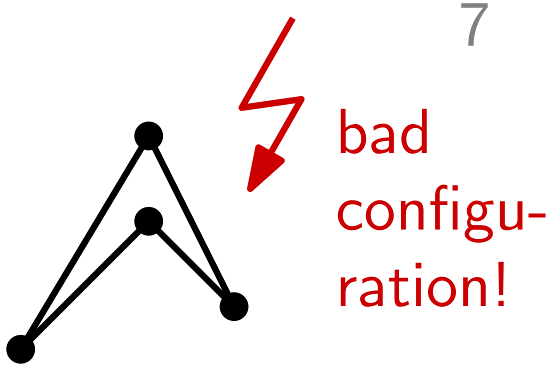
Our Algorithm



Solution:

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

Our Algorithm



Solution:

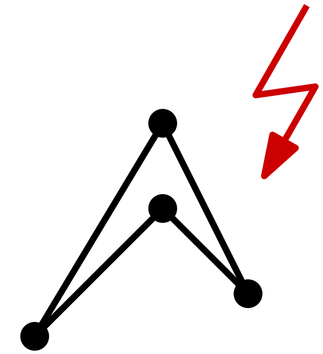
- Make the first vertex in the qudrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.

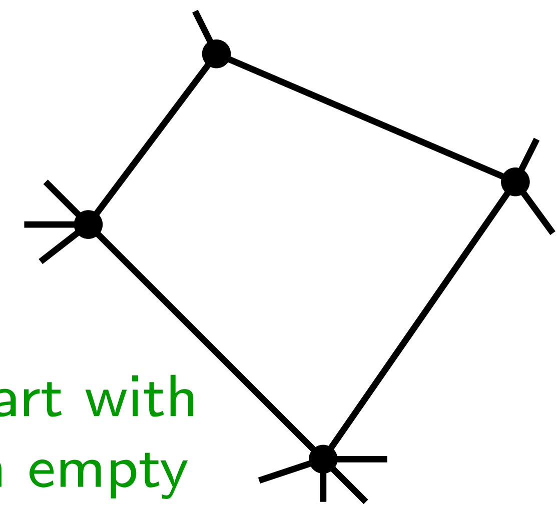
Our Algorithm

Solution:

- Make the first vertex in the qudrangle (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.



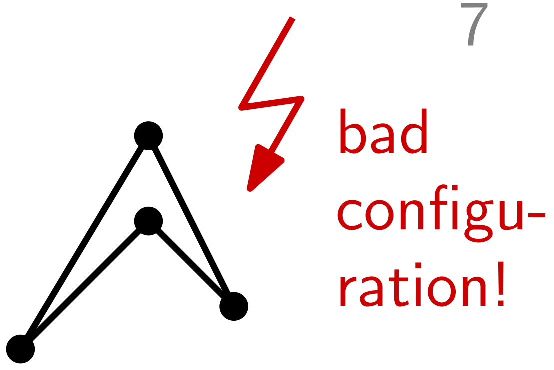


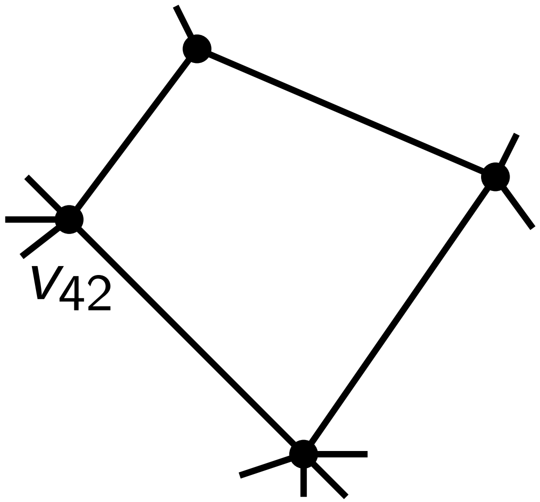
Our Algorithm

Solution:

- Make the first vertex in the qudrangle (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.

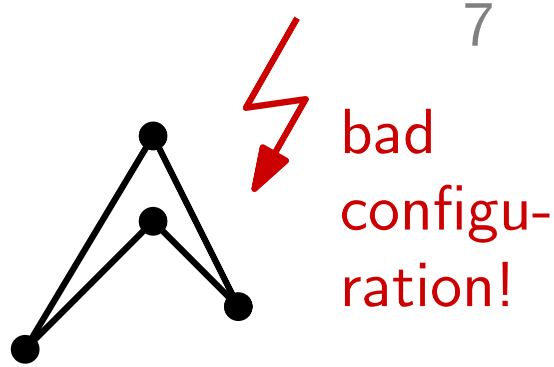


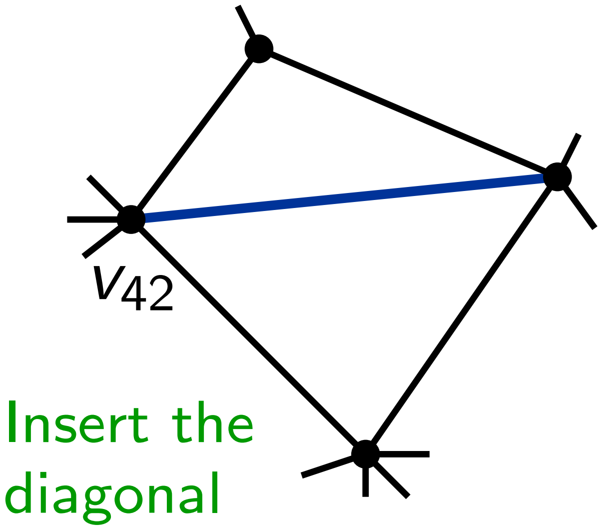


Our Algorithm

Solution:

- Make the first vertex in the qudrangle (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.

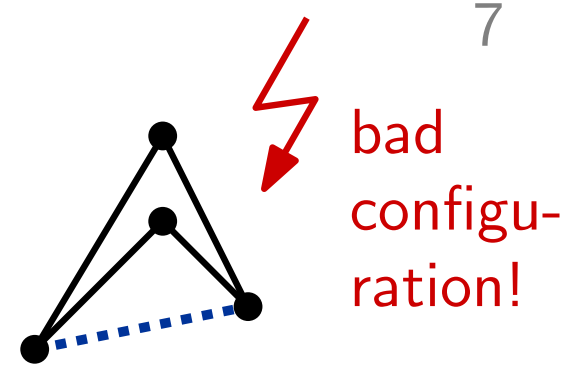


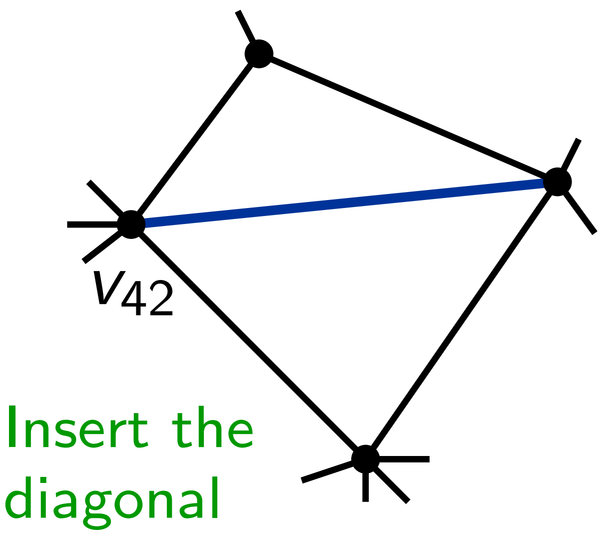


Our Algorithm

Solution:

- Make the first vertex in the qudrangle (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.



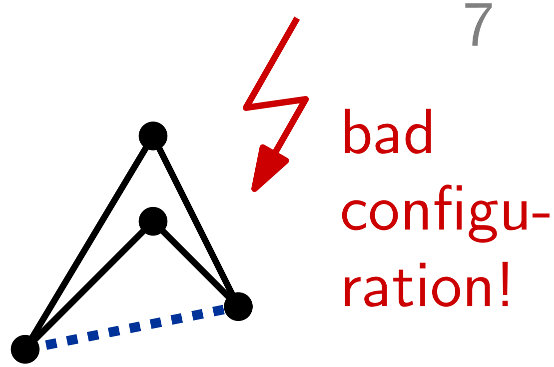


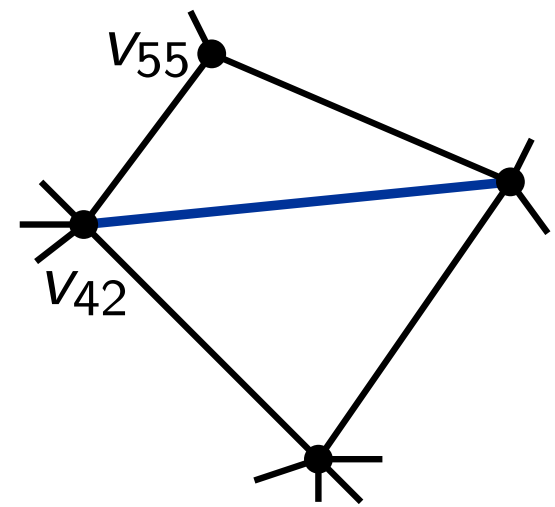
Our Algorithm

Solution:

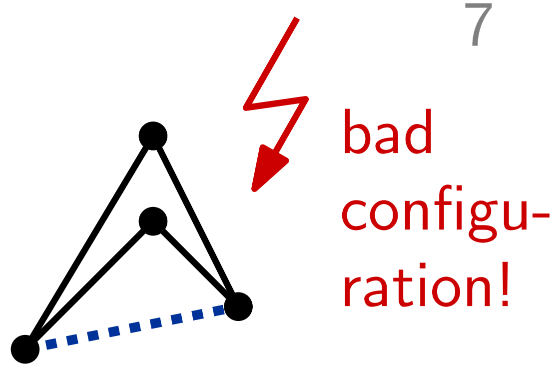
- Make the first vertex in the qudrangle (regarding the canonical ordering) adjacent to the other three vertices.

- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.



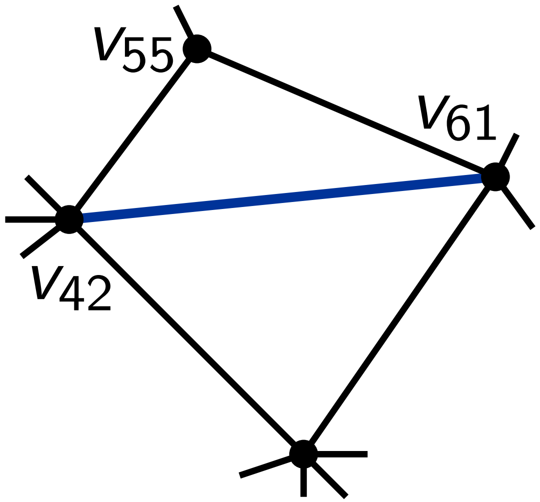


Our Algorithm



Solution:

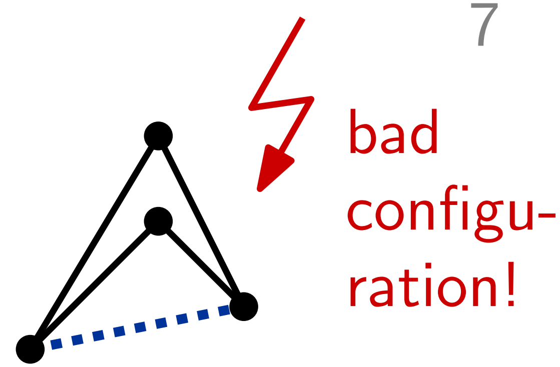
- Make the first vertex in the qudrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.

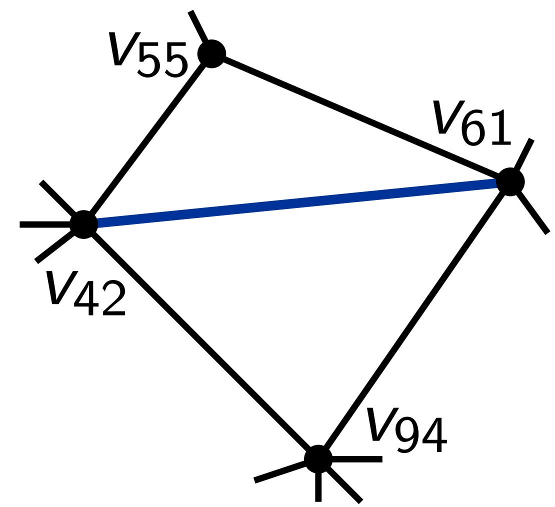


Our Algorithm

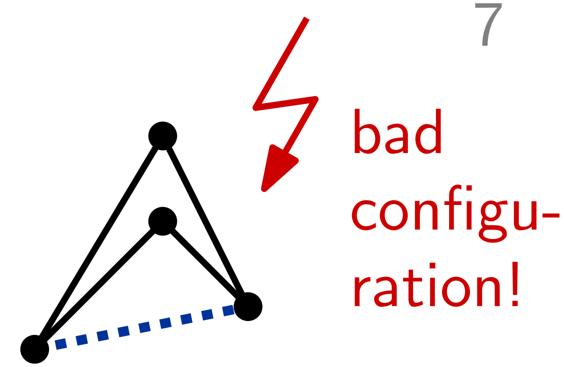
Solution:

- Make the first vertex in the qudrangle (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.



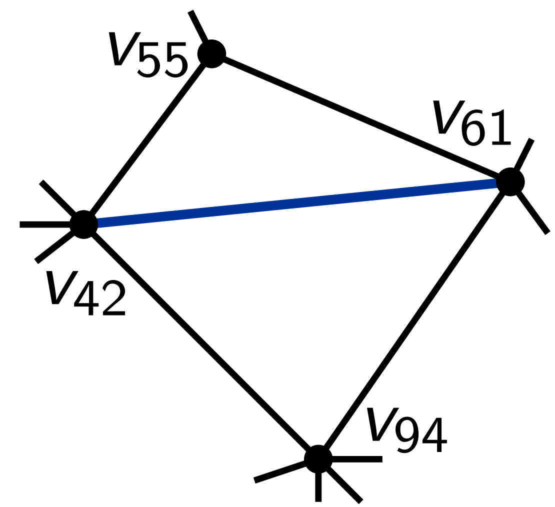


Our Algorithm



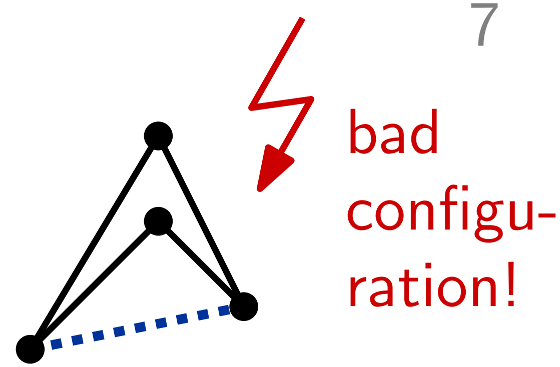
Solution:

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



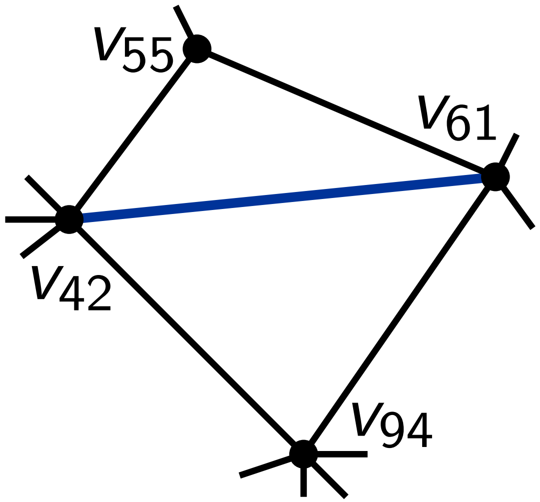
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three "good" cases can appear:

Our Algorithm



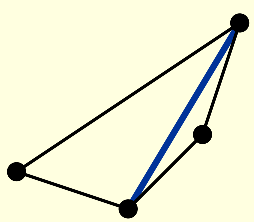
Solution:

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

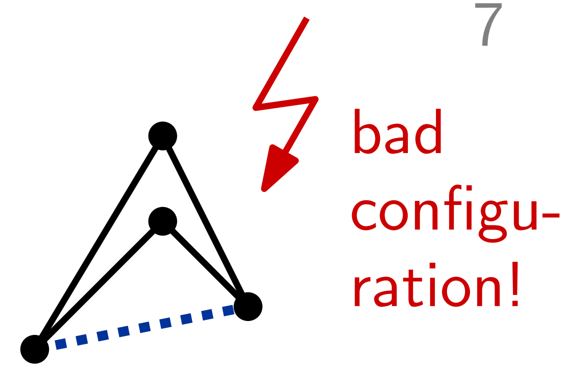


- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three "good" cases can appear:

Case 1

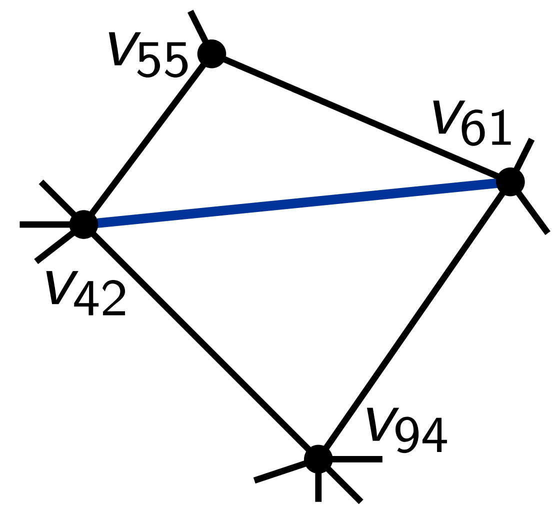


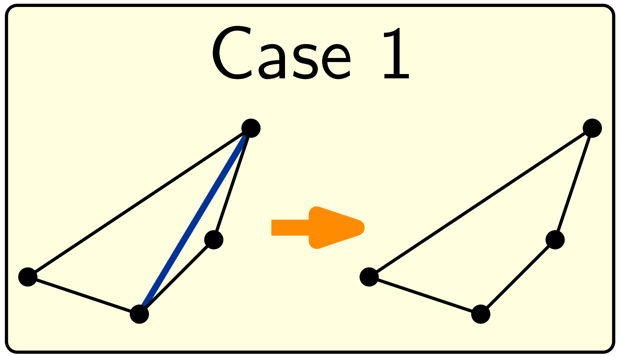
Our Algorithm



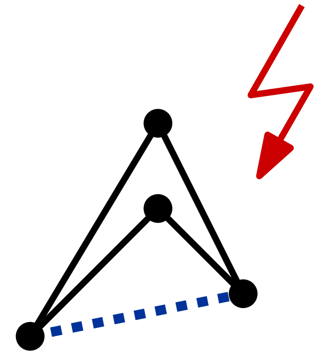
Solution:

- Make the first vertex in the qudrangle (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:



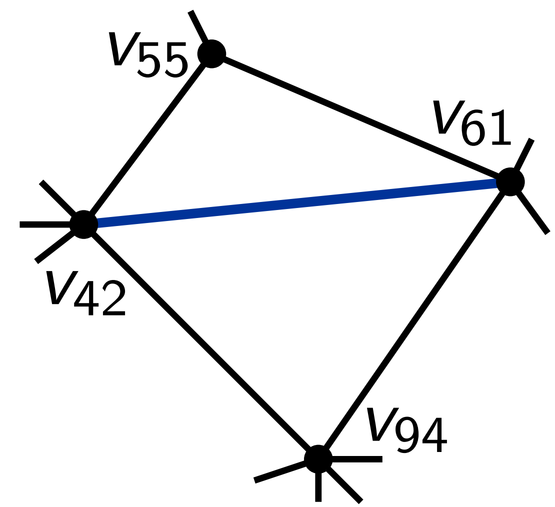


Our Algorithm

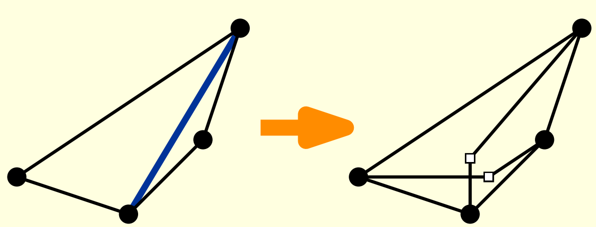


Solution:

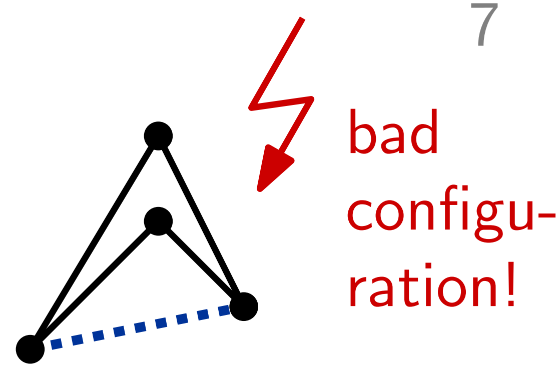
- Make the first vertex in the qudrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:



Case 1

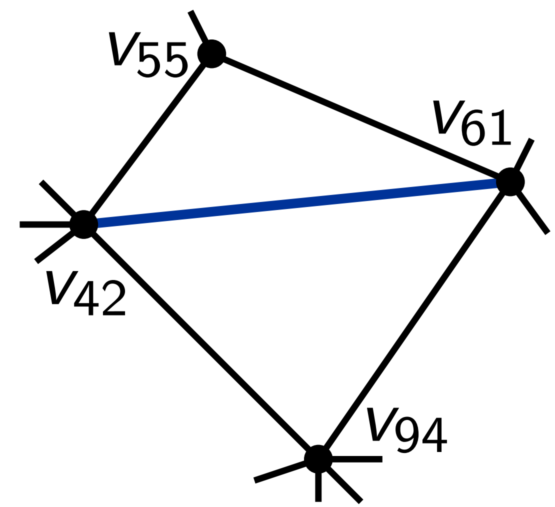


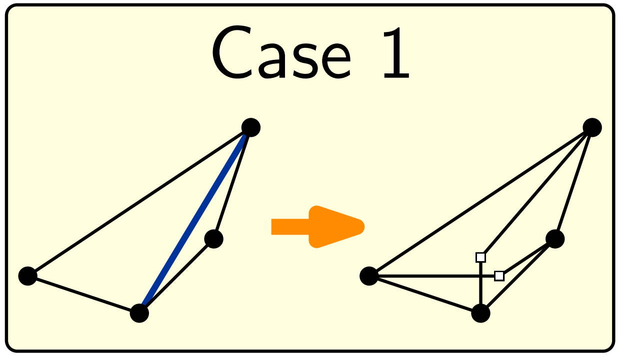
Our Algorithm

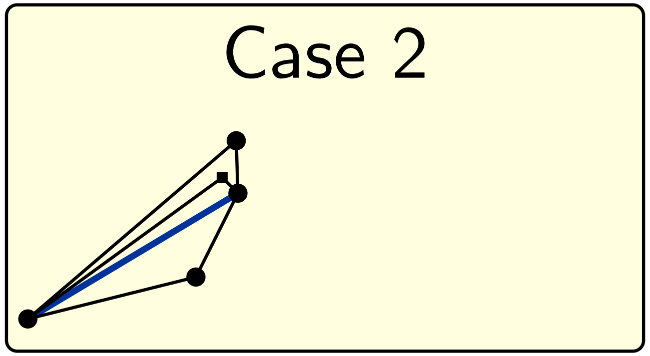


Solution:

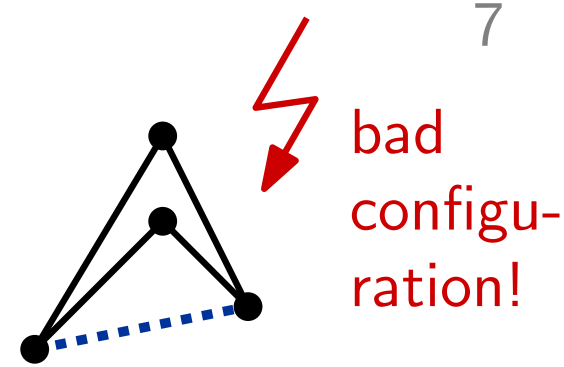
- Make the first vertex in the qudrangle (regarding the canonical ordering) adjacent to the other three vertices.
- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:





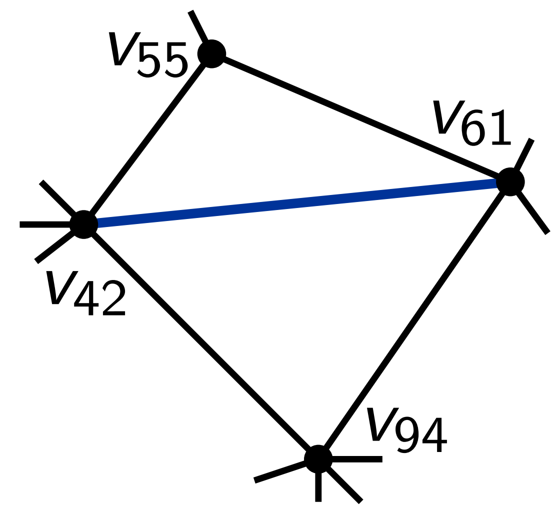


Our Algorithm



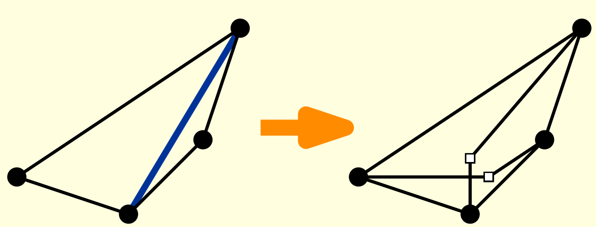
Solution:

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

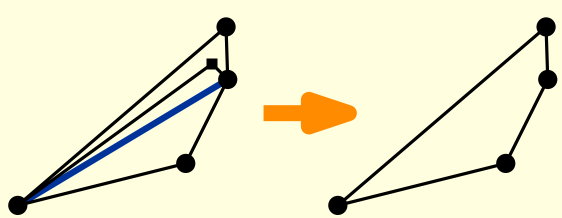


- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three “good” cases can appear:

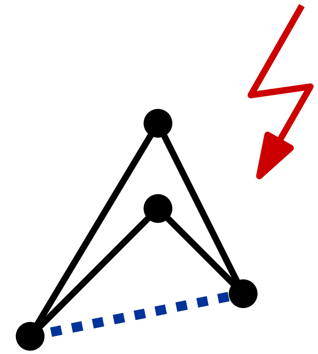
Case 1



Case 2

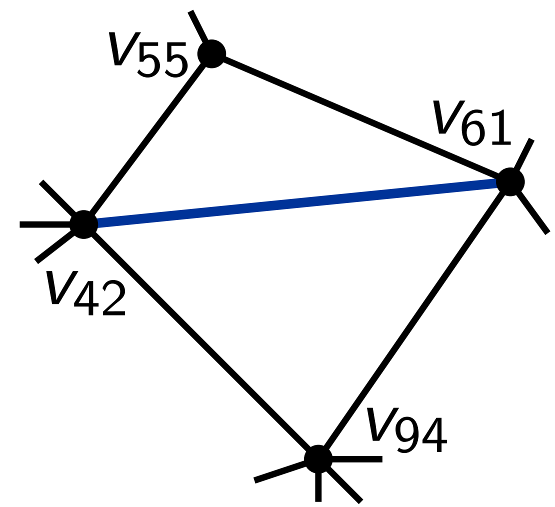


Our Algorithm



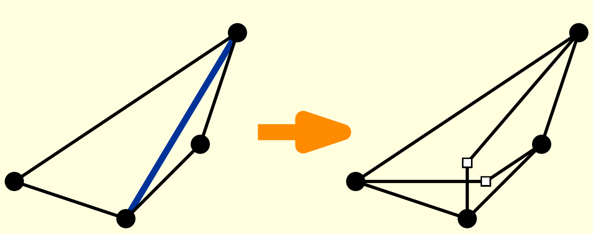
Solution:

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

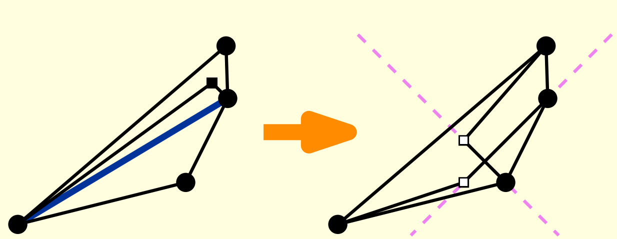


- Use the algorithm by Harel and Sardas (Shift Algorithm for biconnected graphs). It builds a canonical ordering bottom-up instead of top-down.
- Now only three "good" cases can appear:

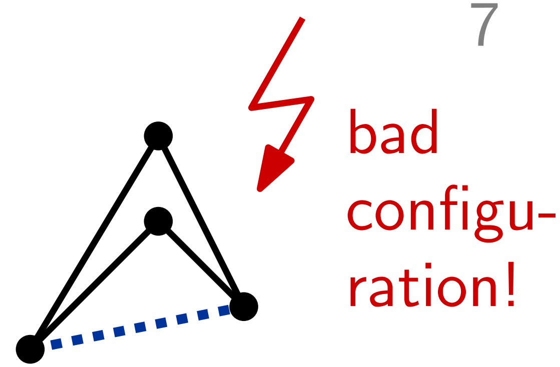
Case 1



Case 2

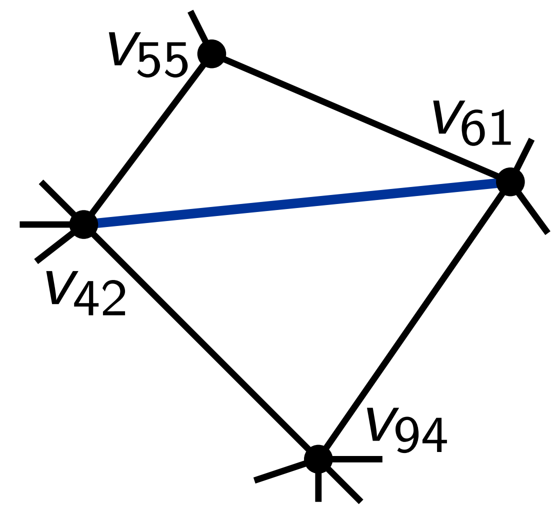


Our Algorithm

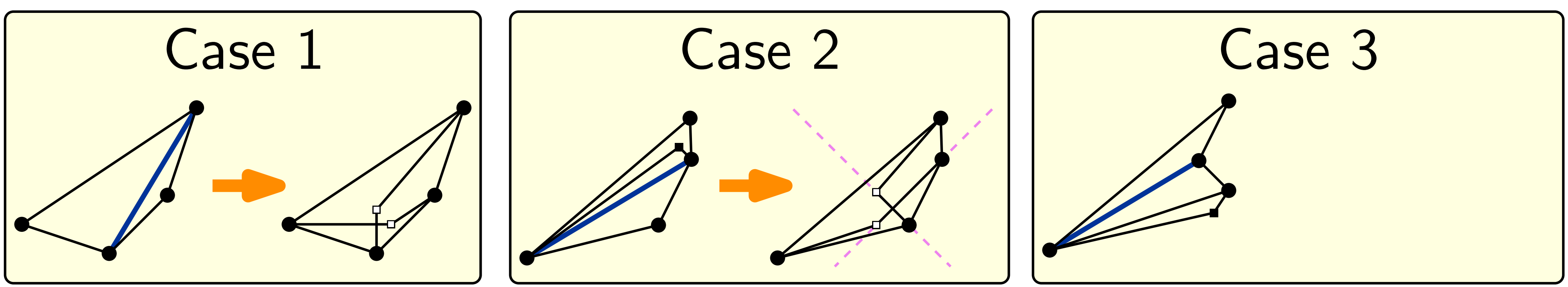


Solution:

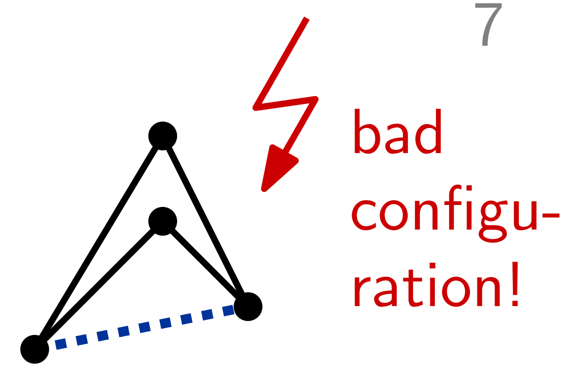
- Make the first vertex in the qudrangle (regarding the canonical ordering) adjacent to the other three vertices.



- 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:

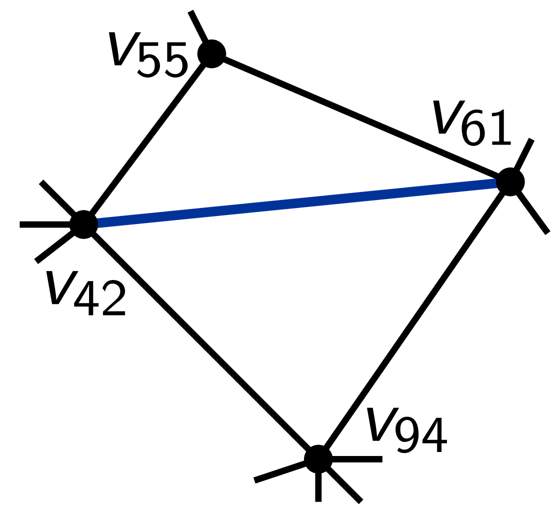


Our Algorithm

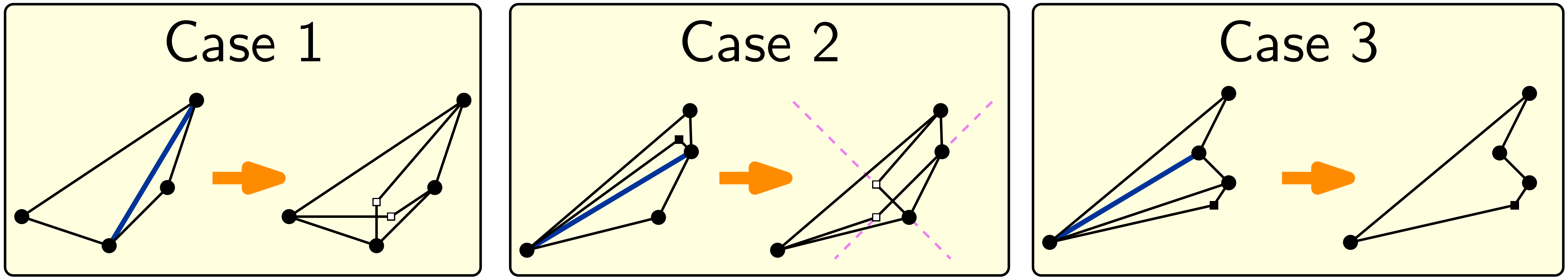


Solution:

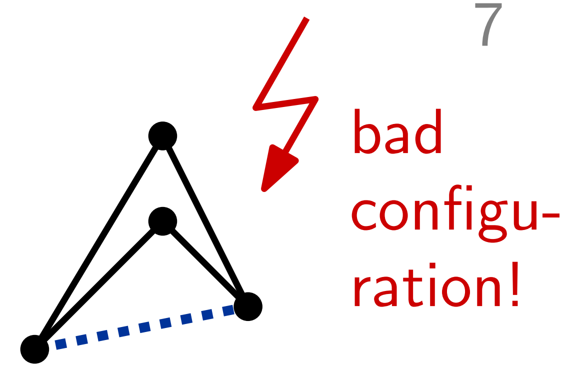
- Make the first vertex in the qudrangle (regarding the canonical ordering) adjacent to the other three vertices.



- 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:

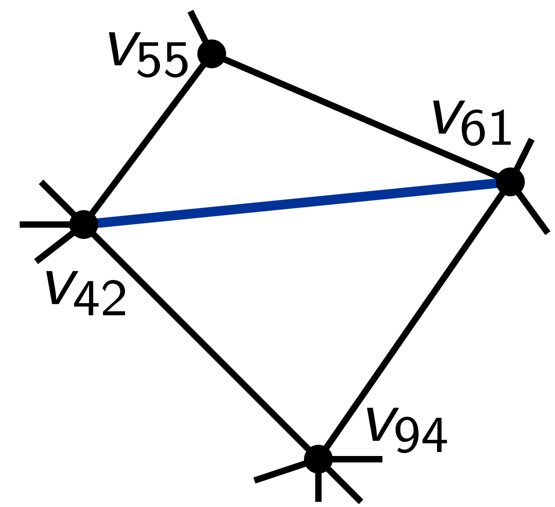


Our Algorithm

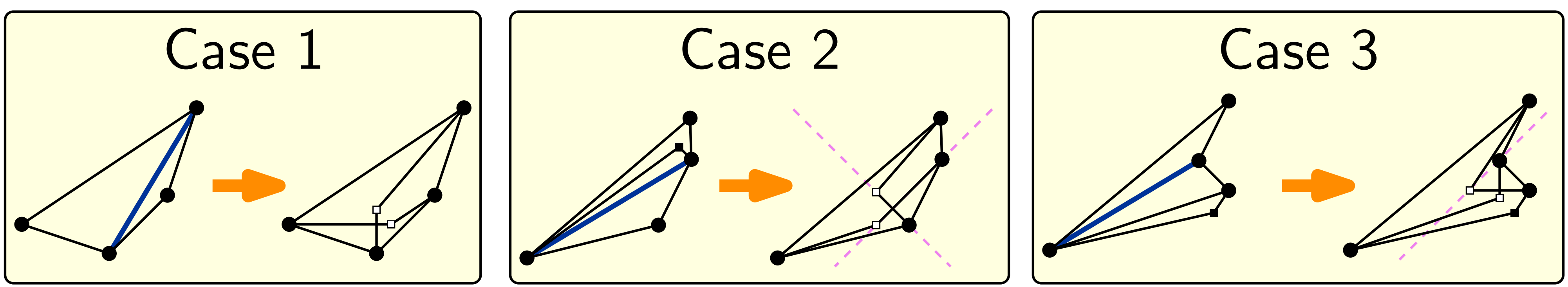


Solution:

- Make the first vertex in the qudrangle (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:



Summary

Summary

- Runs in $O(n)$ time.

Summary

- Runs in $O(n)$ time.
- Resulting drawing is NIC-planar RAC with ≤ 1 bend per edge.

Summary

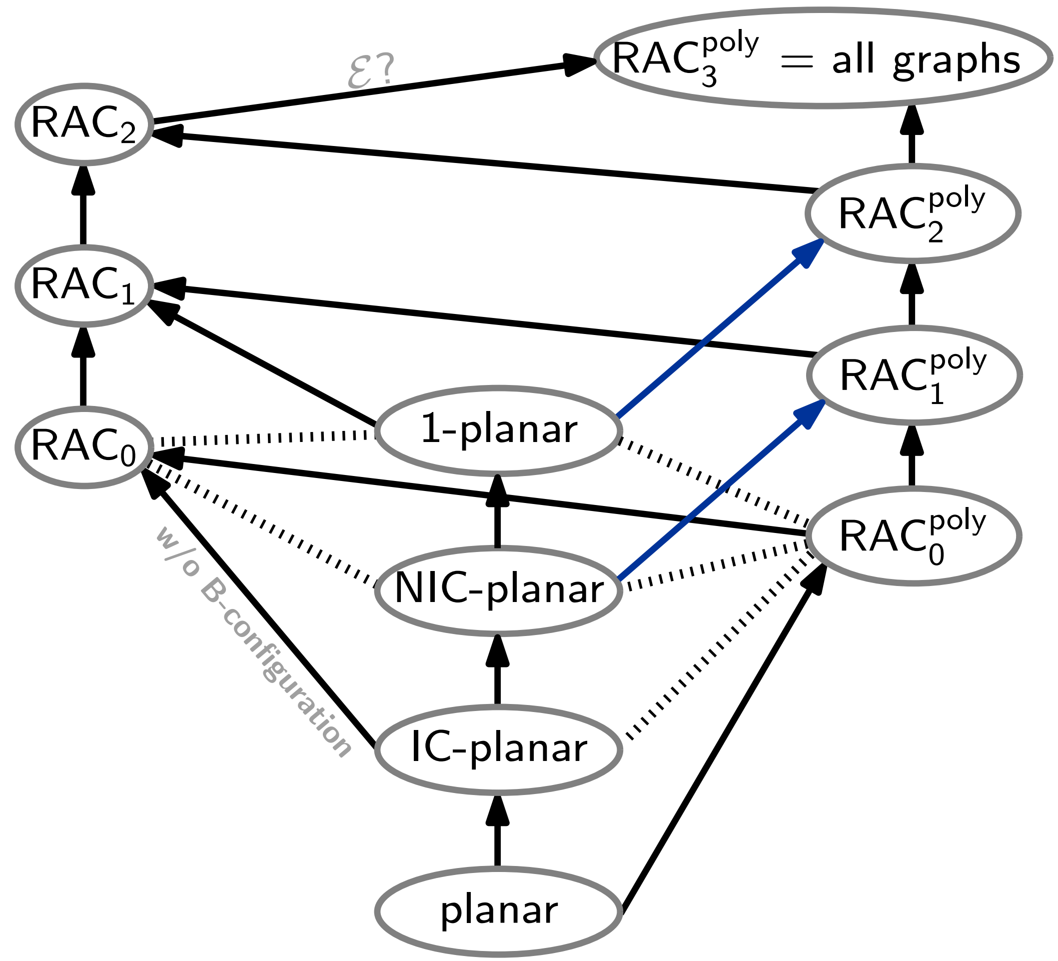
- Runs in $O(n)$ time.
- Resulting drawing is NIC-planar RAC with ≤ 1 bend per edge.
- Grid of size at most $(16n - 32) \times (8n - 16)$.

Summary

- Runs in $O(n)$ time.
- Resulting drawing is NIC-planar RAC with ≤ 1 bend per edge.
- Grid of size at most $(16n - 32) \times (8n - 16)$.
- Needs NIC-planar embedding as input; this embedding is preserved.

Summary

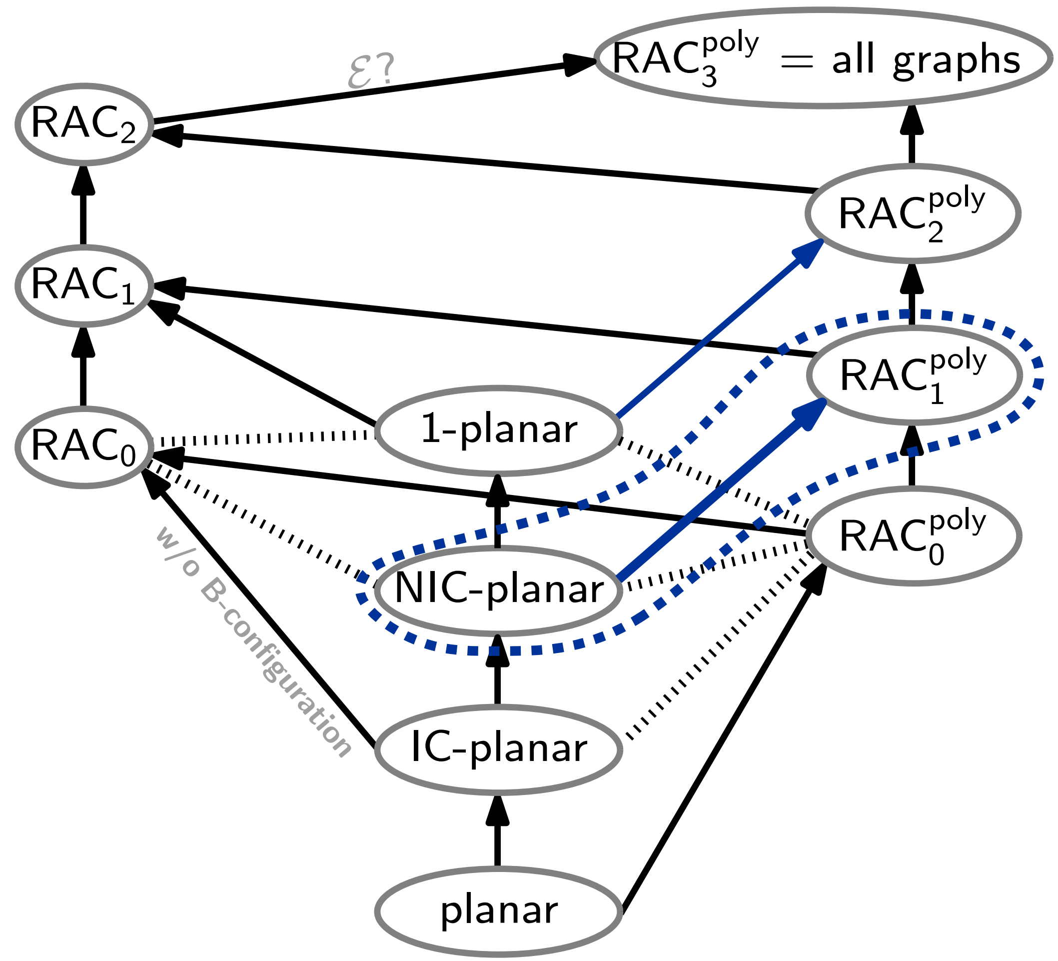
- Runs in $O(n)$ time.
- Resulting drawing is NIC-planar RAC with ≤ 1 bend per edge.
- Grid of size at most $(16n - 32) \times (8n - 16)$.
- Needs NIC-planar embedding as input; this embedding is preserved.



Summary

- Runs in $O(n)$ time.
- Resulting drawing is NIC-planar RAC with ≤ 1 bend per edge.
- Grid of size at most $(16n - 32) \times (8n - 16)$.
- Needs NIC-planar embedding as input; this embedding is preserved.

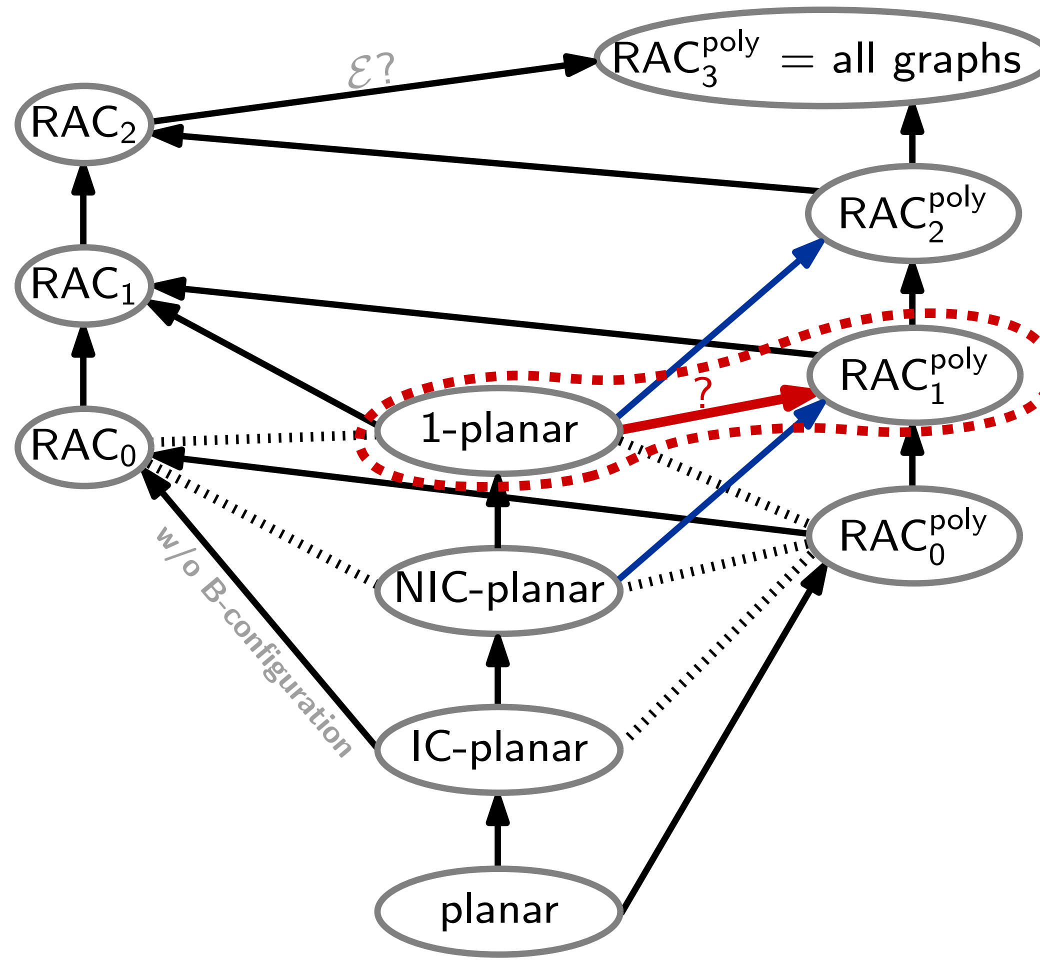
Our main result:
NIC-plane graphs $\subseteq$ RAC_1^{poly}



Summary

- Runs in $O(n)$ time.
- Resulting drawing is NIC-planar RAC with ≤ 1 bend per edge.
- Grid of size at most $(16n - 32) \times (8n - 16)$.
- Needs NIC-planar embedding as input; this embedding is preserved.

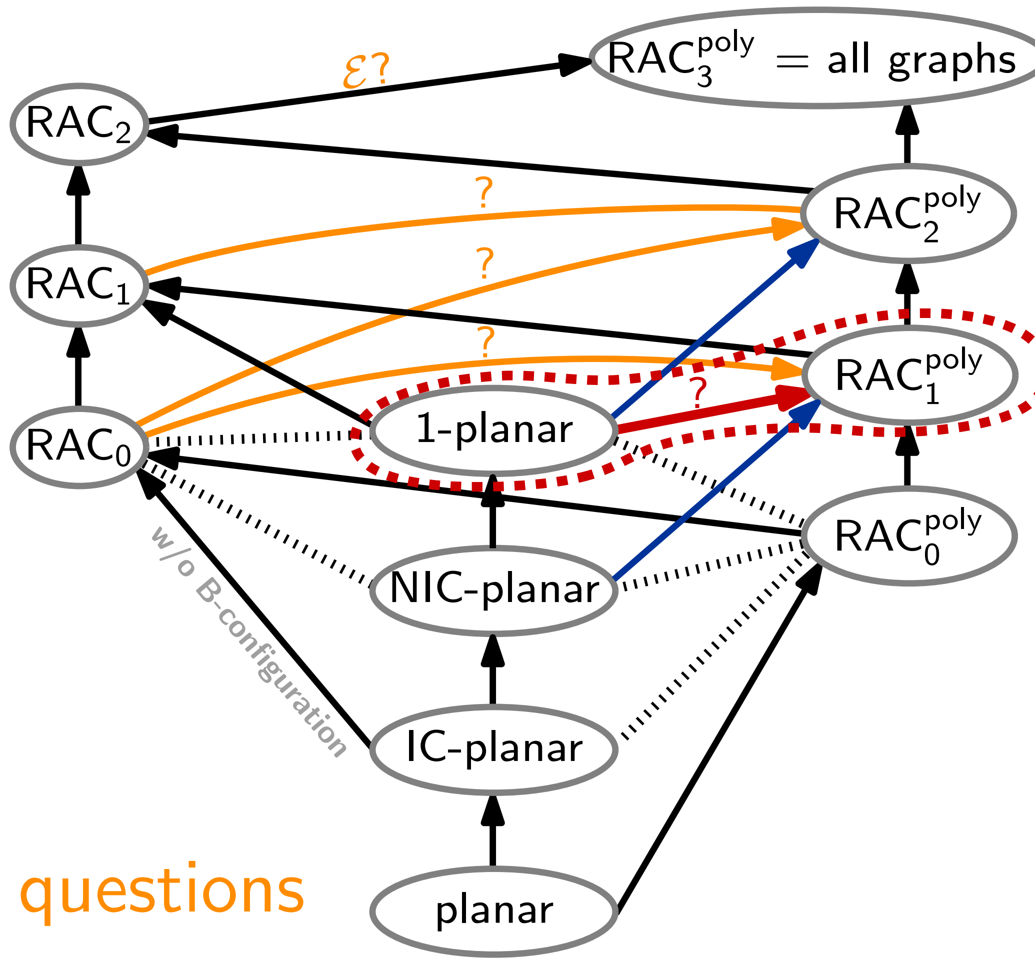
Open question:
1-planar graphs $\subseteq \text{RAC}_1^{\text{poly}}$?



Summary

- Runs in $O(n)$ time.
- Resulting drawing is NIC-planar RAC with ≤ 1 bend per edge.
- Grid of size at most $(16n - 32) \times (8n - 16)$.
- Needs NIC-planar embedding as input; this embedding is preserved.

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



More open questions