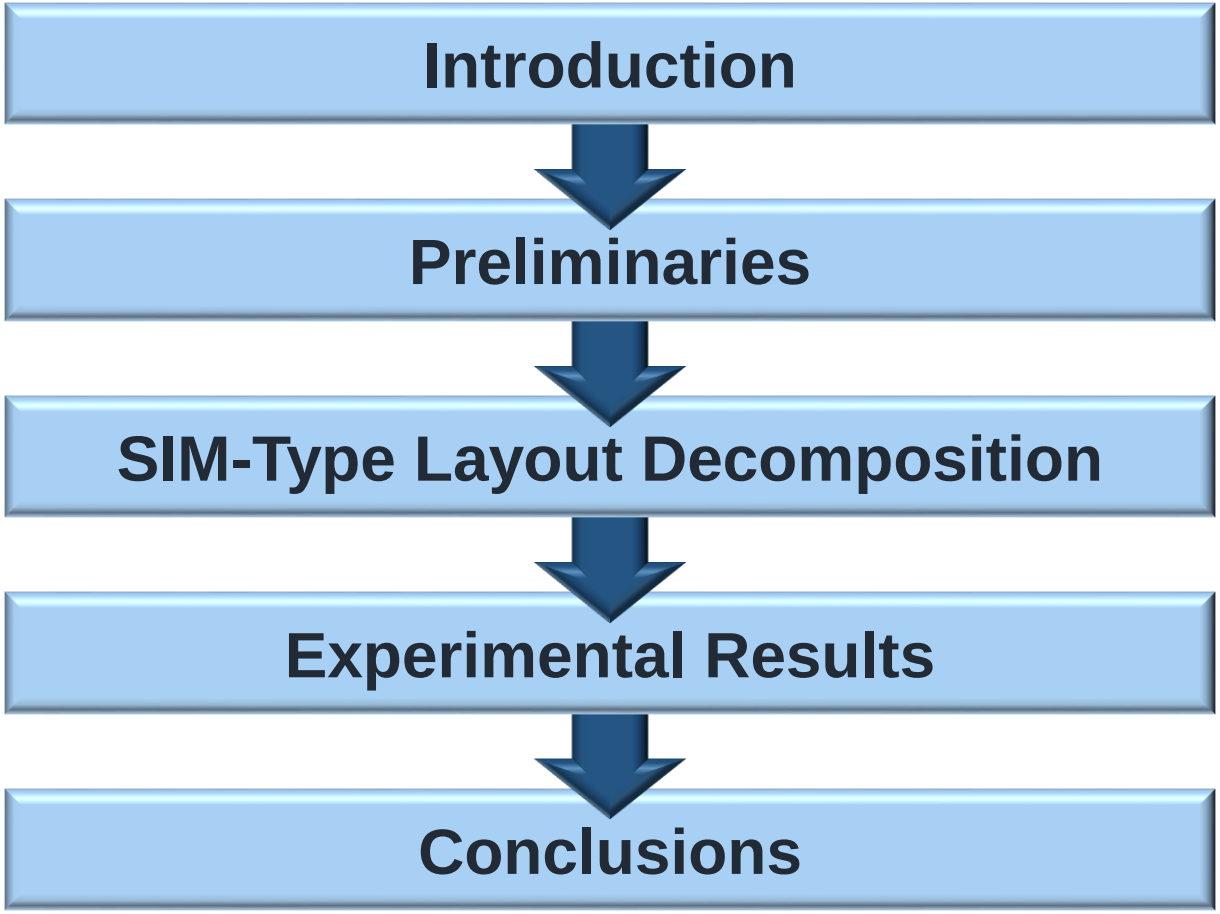
Layout Decomposition for Spacer-is-Metal (SIM) Self-Aligned Double Patterning

Shao-Yun Fang[1], Yi-Shu Tai[2], and Yao-Wen Chang[2]

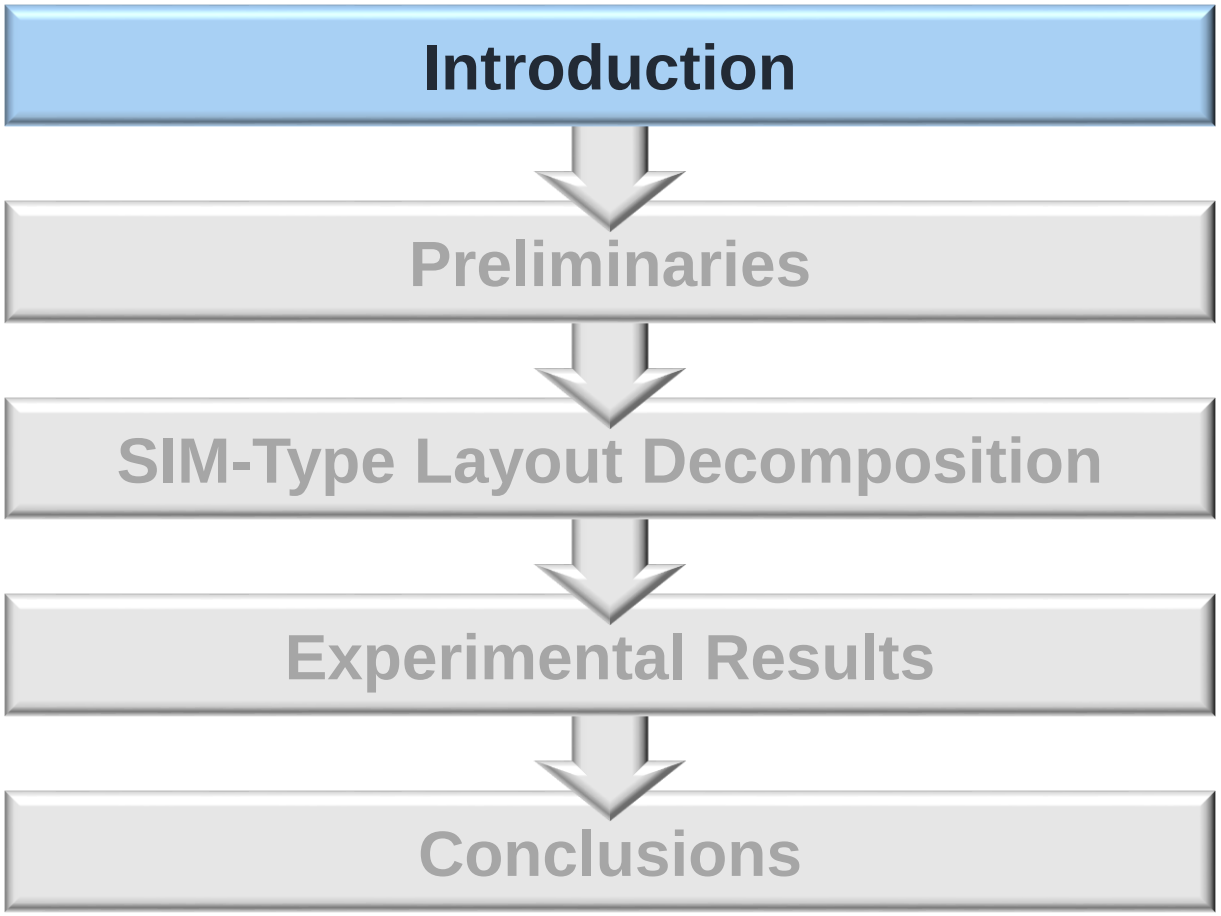
Jan 22, 2015

[1] Department of Electrical Engineering, National Taiwan University of Science and Technology
[2] Department of Electrical Engineering, National Taiwan University

Outline

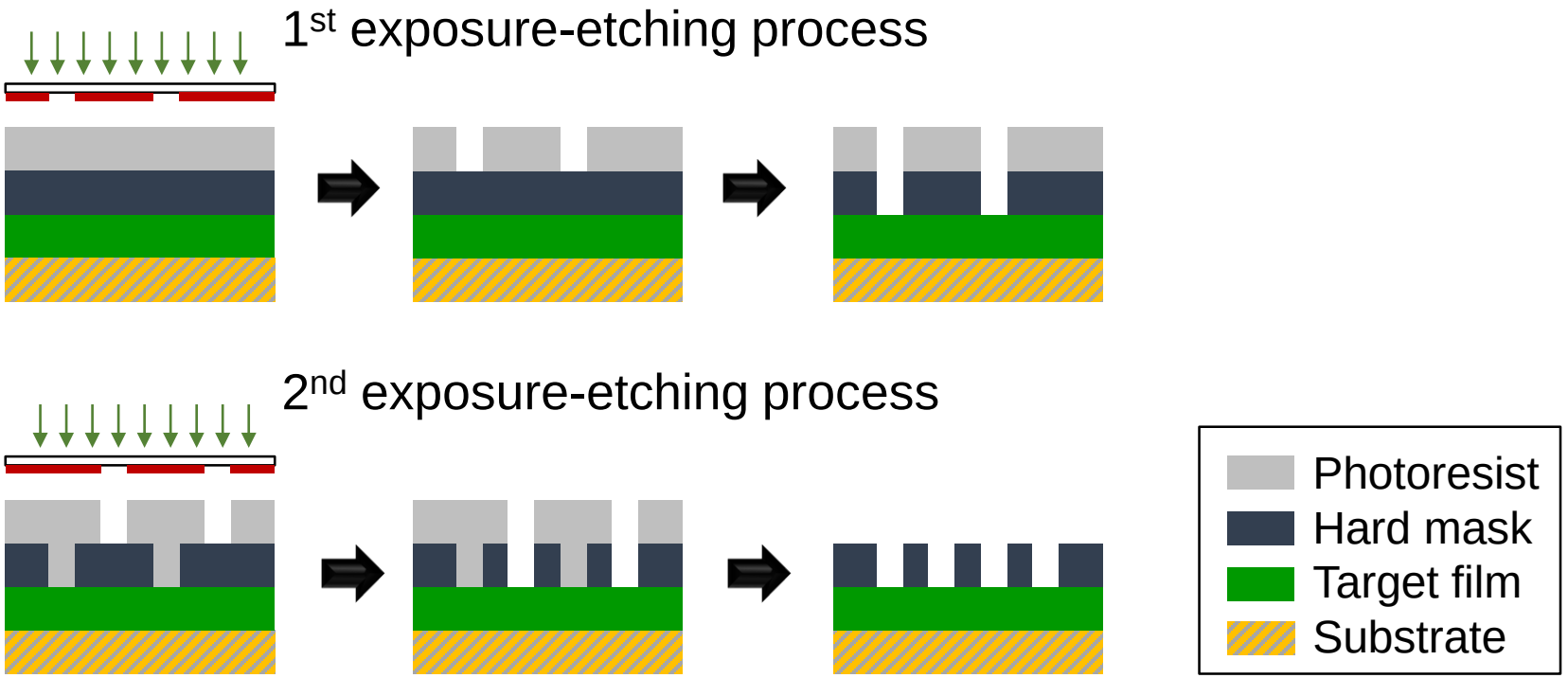

Introduction
Preliminaries
SIM-Type Layout Decomposition
Experimental Results
Conclusions

Outline


Introduction
Preliminaries
SIM-Type Layout Decomposition
Experimental Results
Conclusions

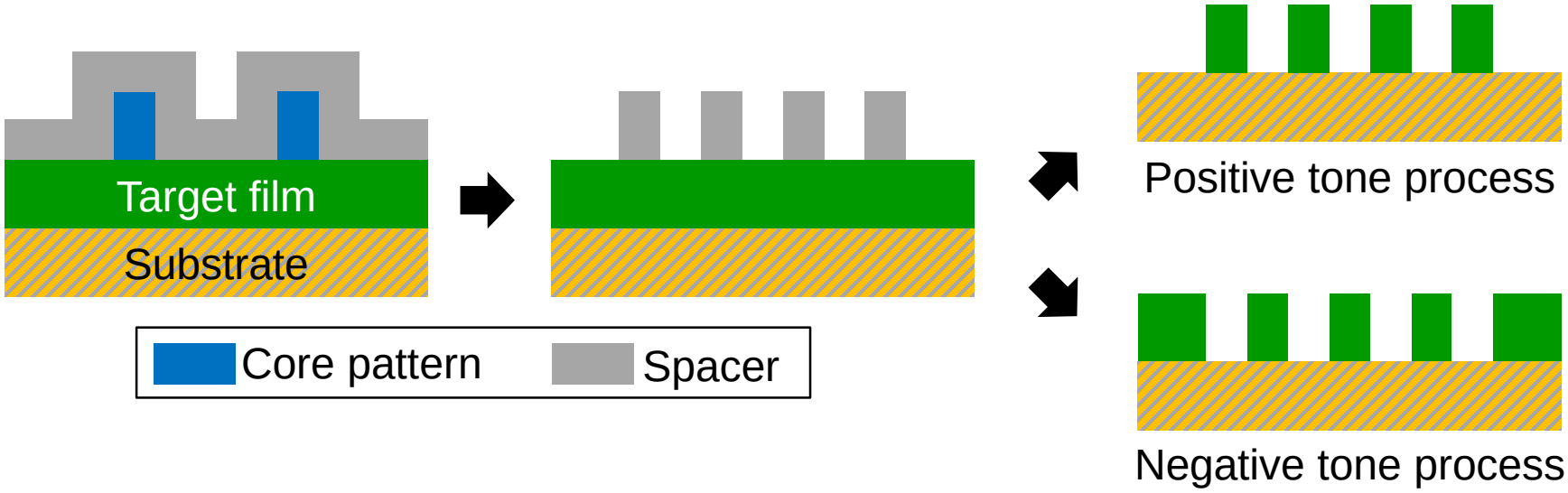
LELE Double Patterning Lithography

- Litho-etch-litho-etch (LELE) double patterning lithography (DPL) has been widely known and used in industry



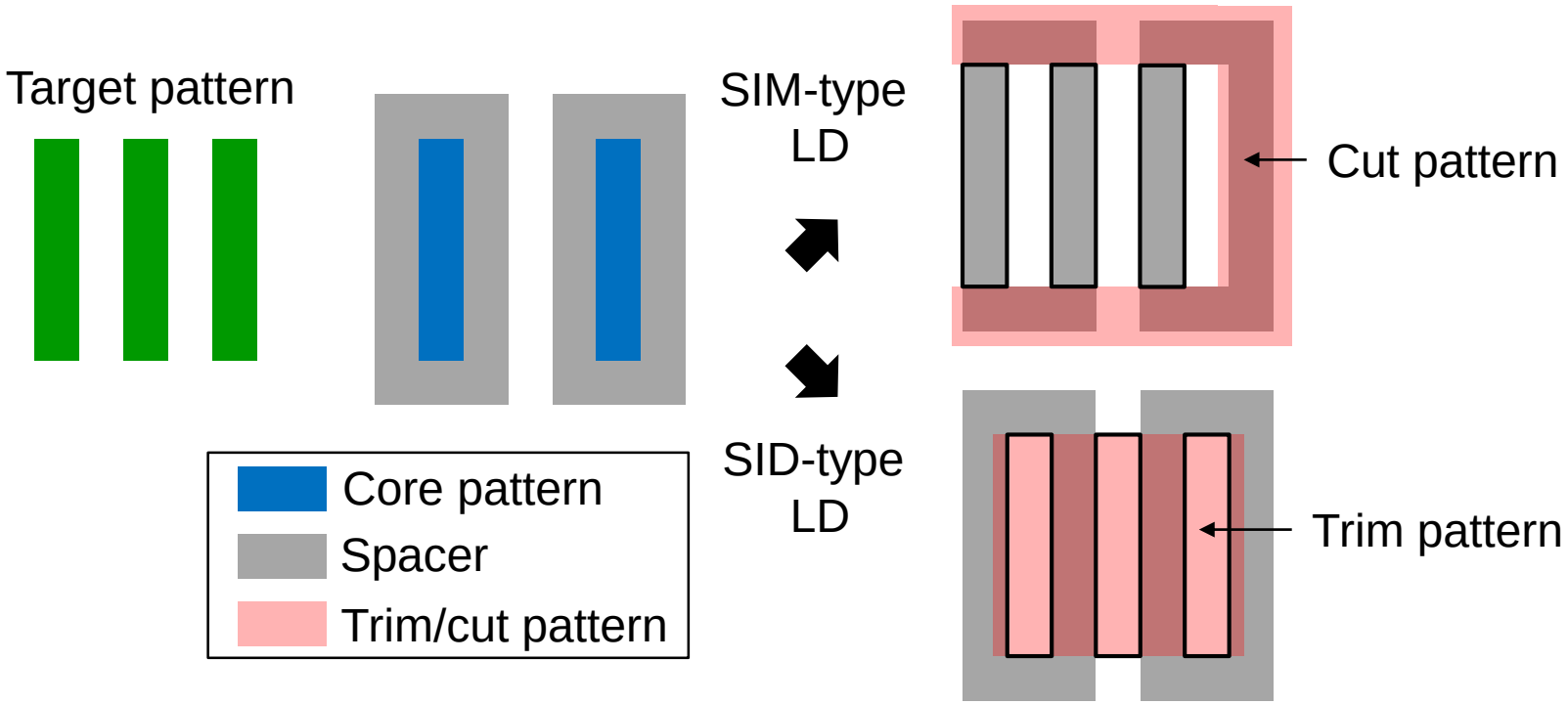
Self-Aligned Double Patterning (SADP)

- Self-aligned double patterning (SADP) becomes more popular due to the better overlay and critical dimension (CD) controllability
- Processes in SADP
  - Positive: spacers define lines
  - Negative: spacers define trenches

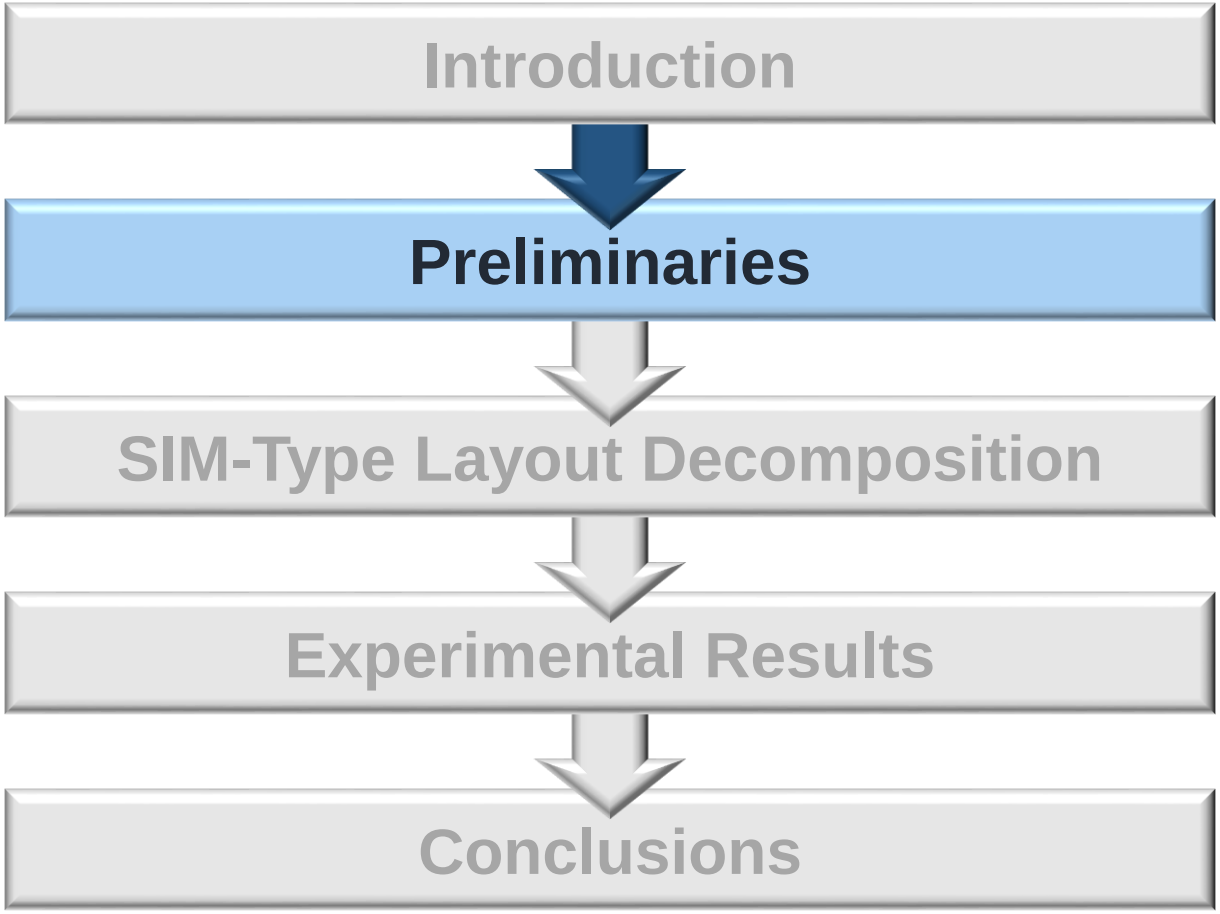


Layout Decomposition in SADP

- Layout decomposition (LD) in SADP
  - Spacer-is-metal (SIM): spacers define patterns
  - Spacer-is-dielectric (SID): spacers define spacings among patterns

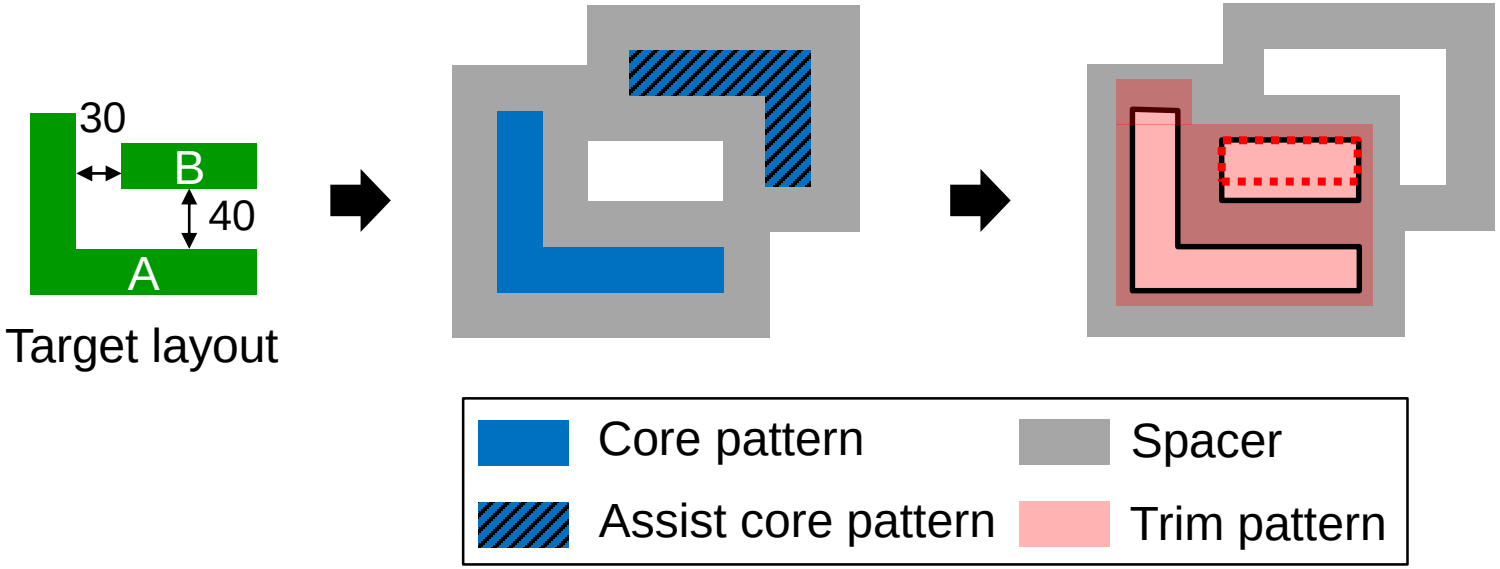


Outline


Introduction
Preliminaries
SIM-Type Layout Decomposition
Experimental Results
Conclusions

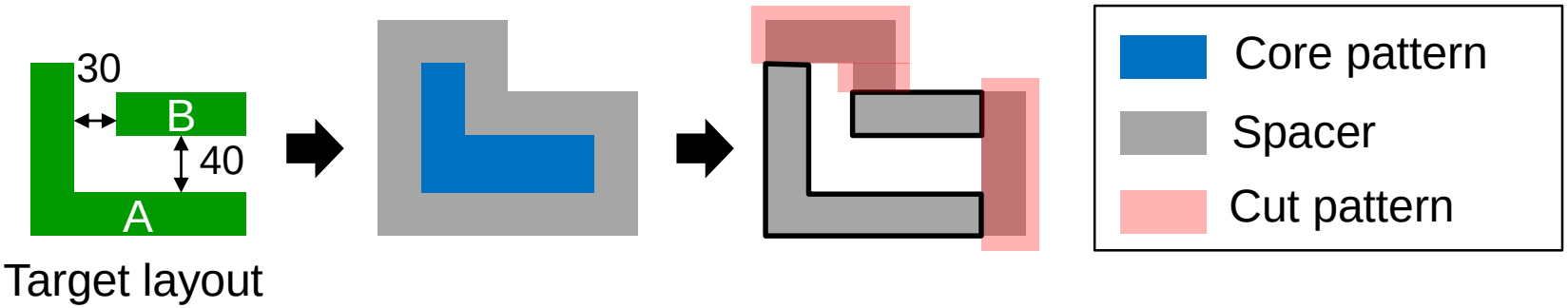
SIM-Type vs. SID-Type 2D LD

- SID-type 2D layout decomposition
  - Patterns are formed by either core patterns or assist core patterns
  - Patterns with arbitrary spacing values may be distorted after LD



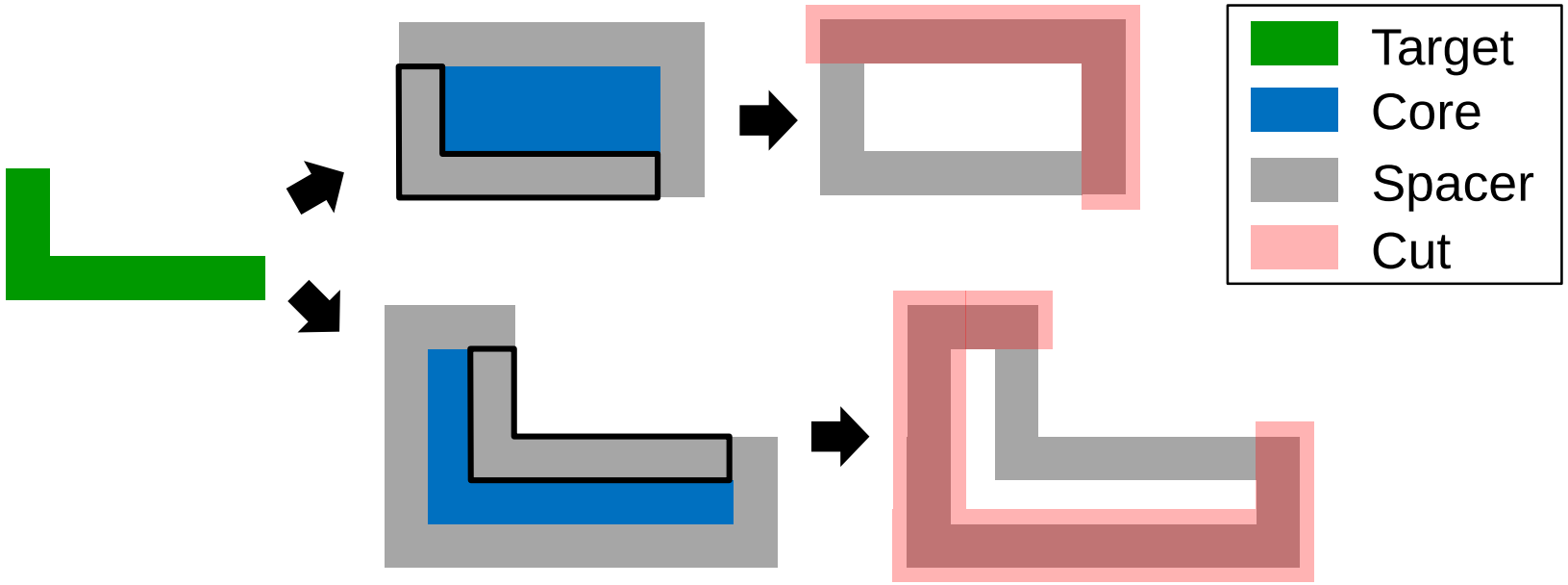
SIM-Type vs. SID-Type 2D LD

- SID-type 2D layout decomposition
  - Patterns are formed by either core patterns or assist core patterns
  - Patterns with arbitrary spacing values may be distorted after LD
- SIM-type 2D layout decomposition
  - Patterns are formed by spacers
  - May have better decomposability since non-uniform spacings can be controlled by core patterns



Cut Pattern Determination

- Cut patterns are used to remove spacers not covering layout patterns
- A cut pattern can be immediately determined as a core pattern is chosen



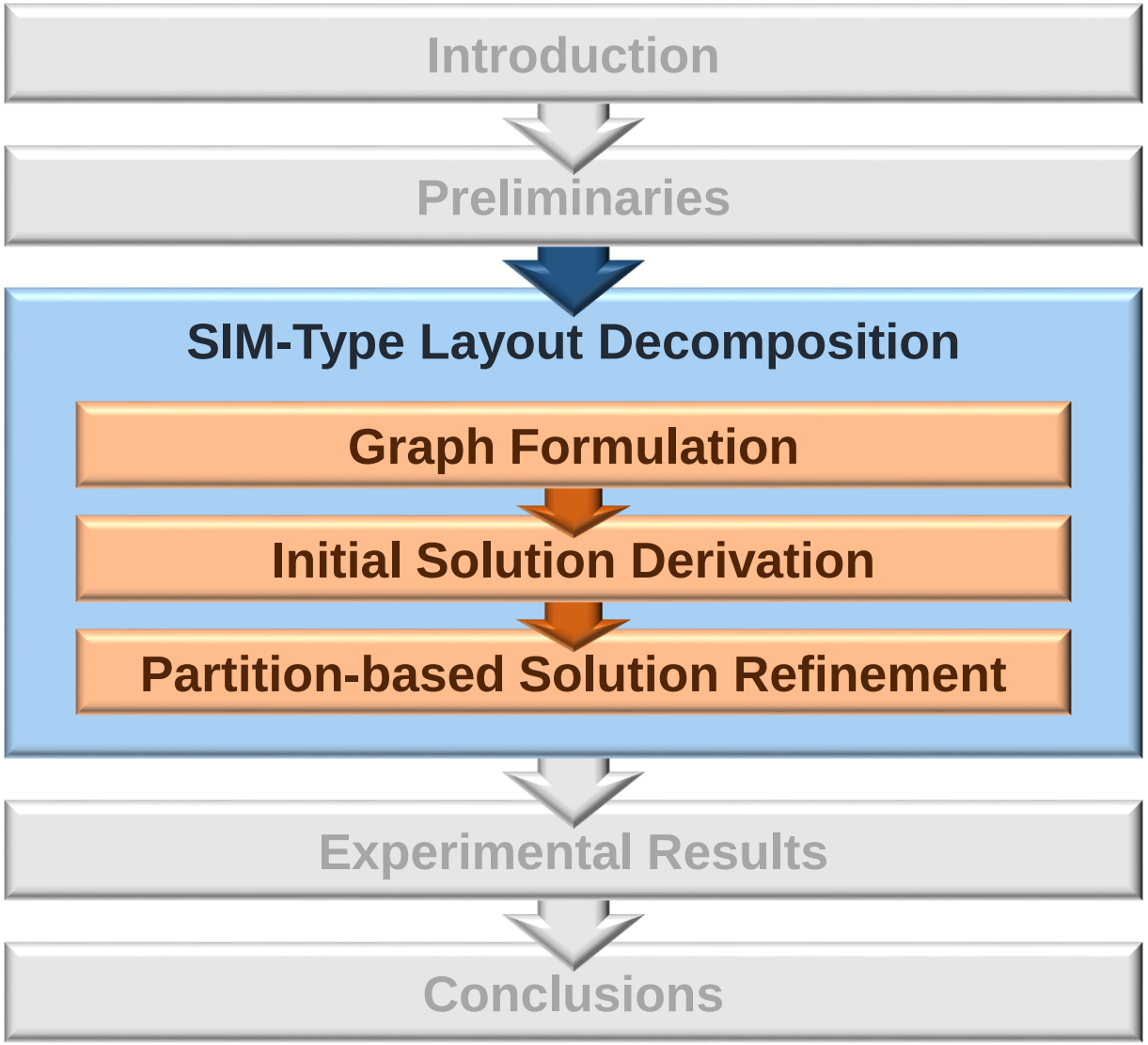
Previous Studies

- SID-type layout decomposition
  - Ban et al., “Flexible 2D layout decomposition framework for spacer-type double patterning lithography,” in *DAC-2011*
  - Zhang et al., “Self-aligned double patterning decomposition for overlay minimization and hot spot detection,” in *DAC-2011*
  - Xiao et al., “A polynomial time exact algorithm for self-aligned double patterning layout decomposition,” in *ISPD-2012*
- SIM-type layout decomposition
  - Zhang et al., “Effective decomposition algorithm for self-aligned double patterning lithography,” in *SPIE-2011*
    - A SAT-based algorithm (NP-complete)
    - Not applicable to indecomposable designs

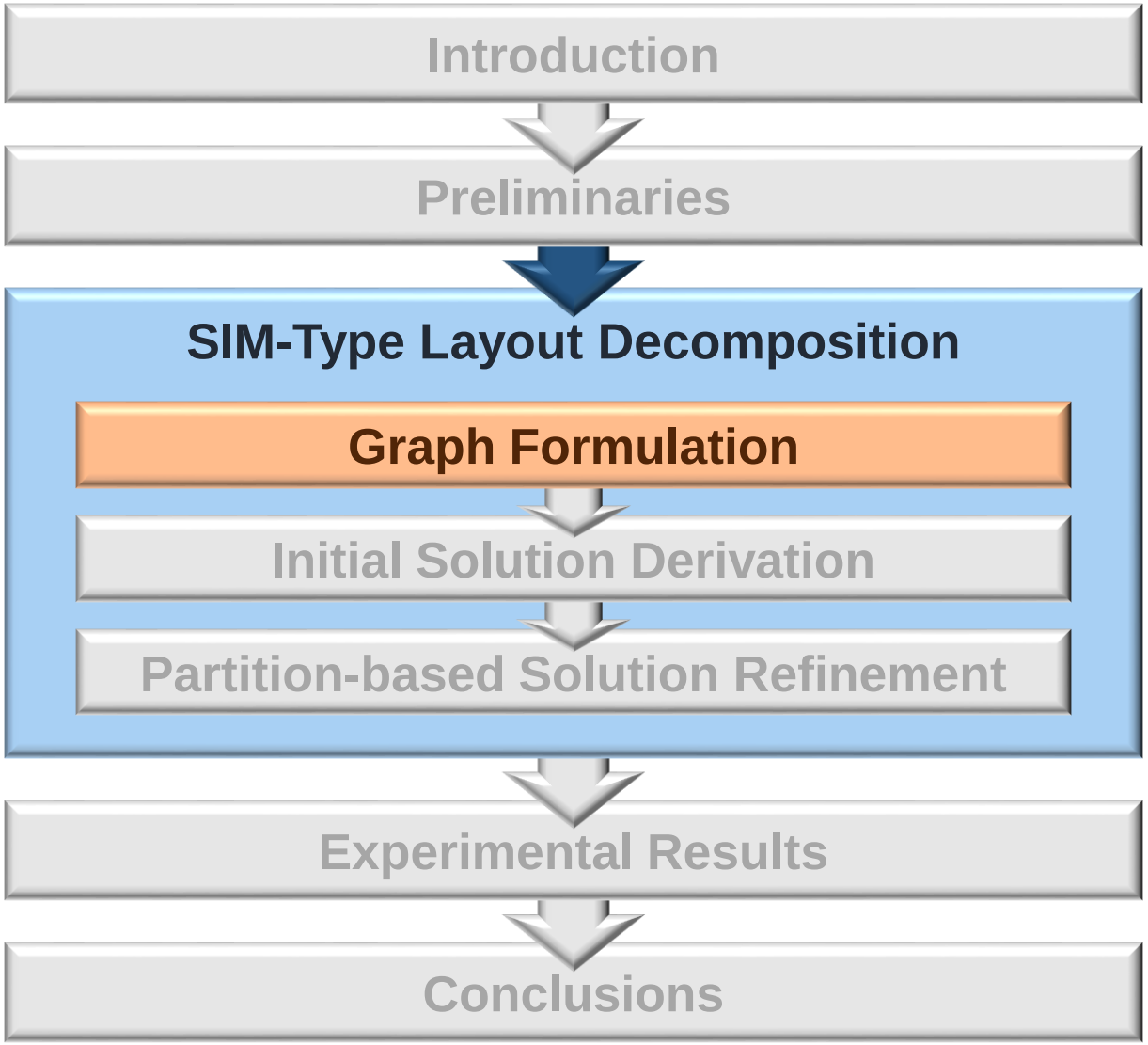
SIM-Type Layout Decomposition

- Terminologies
  - A core conflict: a pair of core patterns with distance $< dis_{core}$
  - A cut conflict: a pair of cut patterns with distance $< dis_{cut}$
- **Problem formulation**
  - **Given**: a photomask layout
  - **Objective**: a layout decomposition result with minimized #conflicts
    - A layout decomposition result = a core mask + a cut mask
    - #conflicts = #core conflicts + #cut conflicts
  - **Constraint**: the produced layout is exactly the same as the original one

Outline

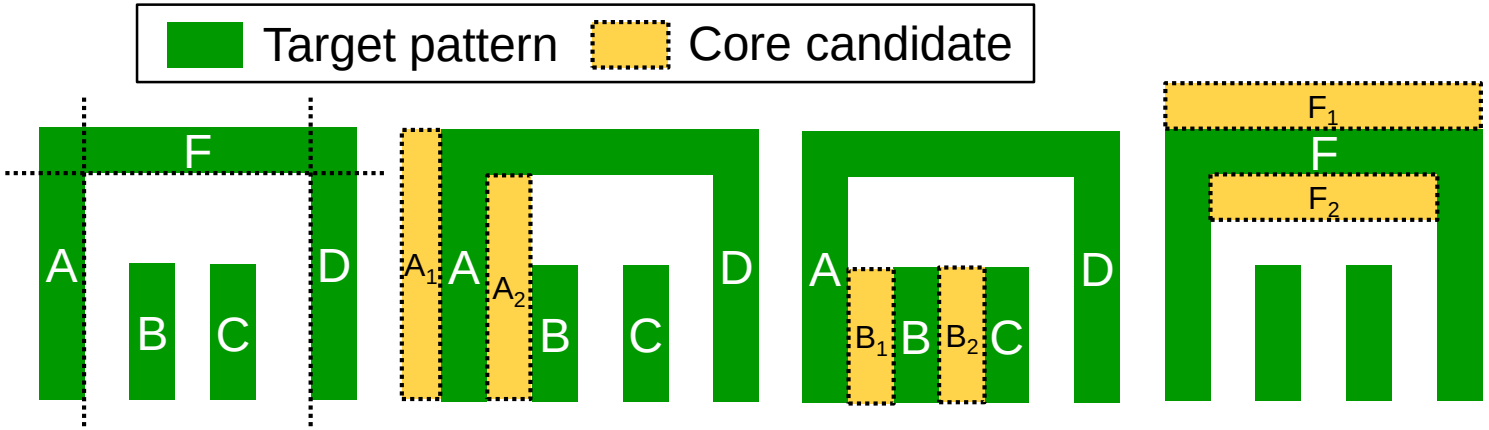

Introduction
Preliminaries
SIM-Type Layout Decomposition
Graph Formulation
Initial Solution Derivation
Partition-based Solution Refinement
Experimental Results
Conclusions

Outline


Introduction
Preliminaries
SIM-Type Layout Decomposition
Graph Formulation
Initial Solution Derivation
Partition-based Solution Refinement
Experimental Results
Conclusions

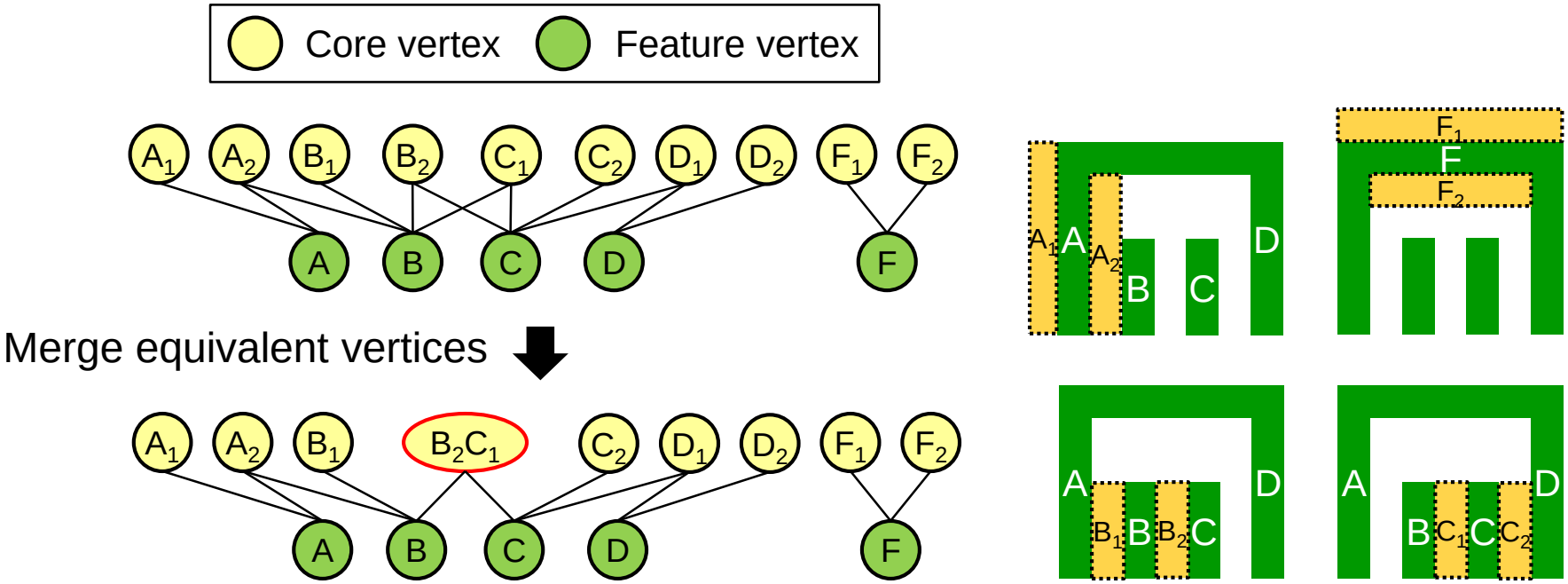
Core Candidate Identification

- Core candidates are identified for spacer deposition
  - Fragment polygons into rectangles (features)
  - Identify two core candidates for each feature



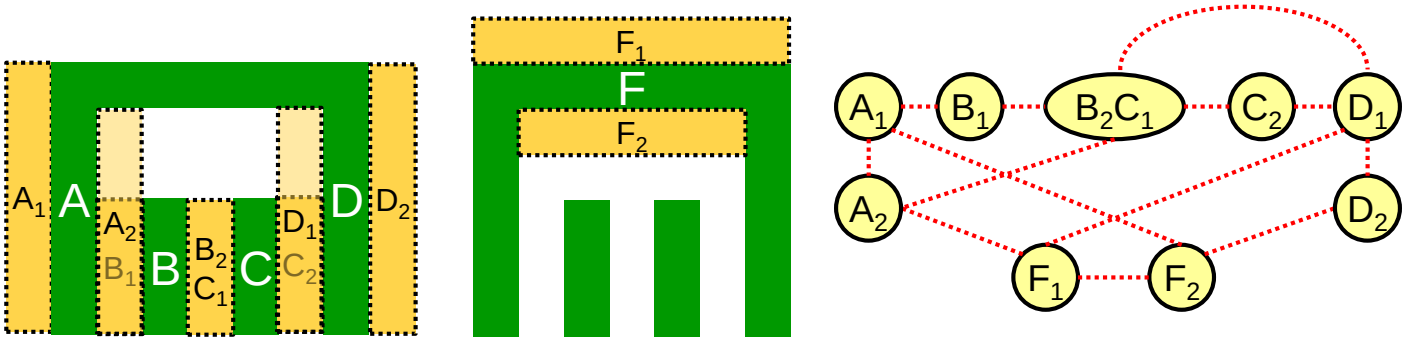
Core-Feature Matching Graph Construction

- Construct a core-feature matching graph G_M
  - A feature vertex u: a feature
  - A core vertex v: a core candidate
  - An edge (u, v): v is a core candidate of u
- Merge equivalent vertices to reduce graph complexity



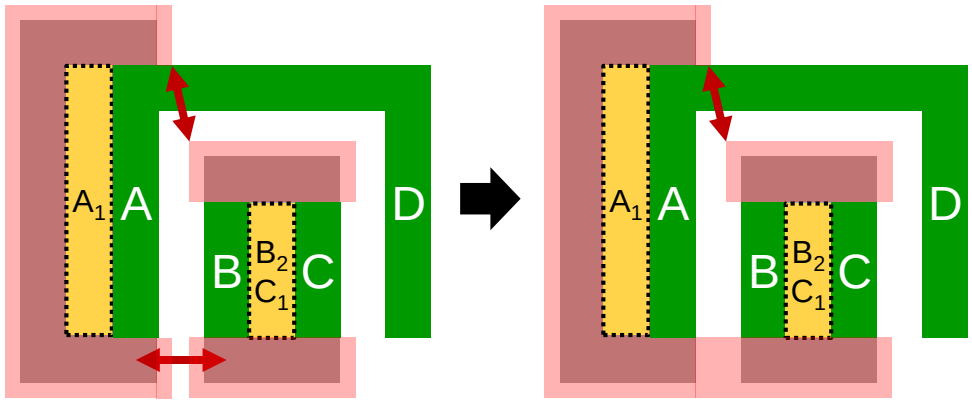
Conflict Graph Construction

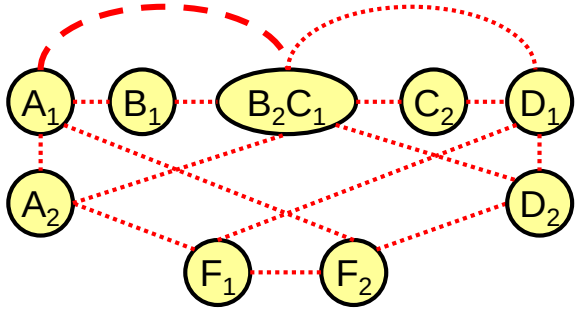
- Inspect conflicts among core candidates
  - The distance between two core candidates $< dis_{core}$
  - The two core candidates cannot be merged
- Construct a conflict graph G_C
  - A vertex v: a core candidate
  - An edge (v_i, v_j): v_i and v_j are conflicting



Conflict Graph Construction (cont'd)

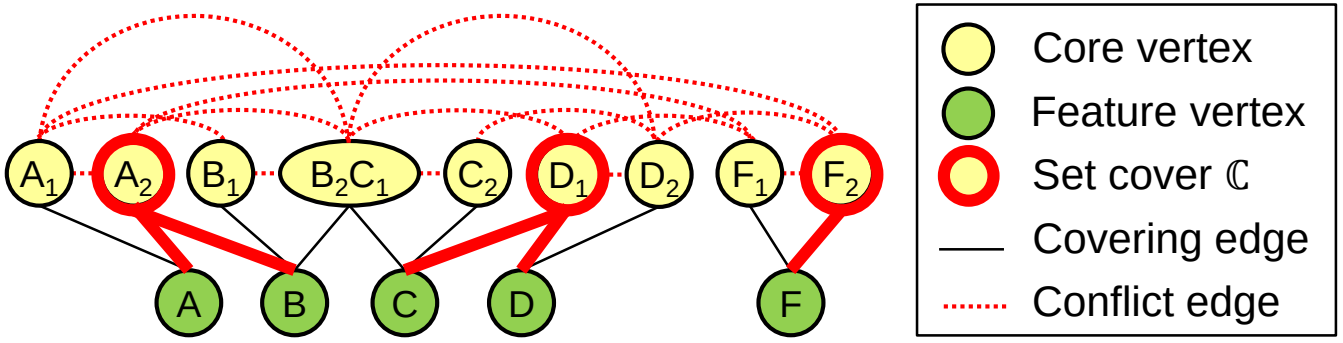
- Inspect conflicts among cut patterns
  - The distance between two cut patterns $< dis_{cut}$
  - The two cut patterns cannot be merged
- Update the conflict graph G_C
  - A vertex v: a core candidate
  - An edge (v_i, v_j): v_i and v_j are conflict or two corresponding cut patterns are conflict



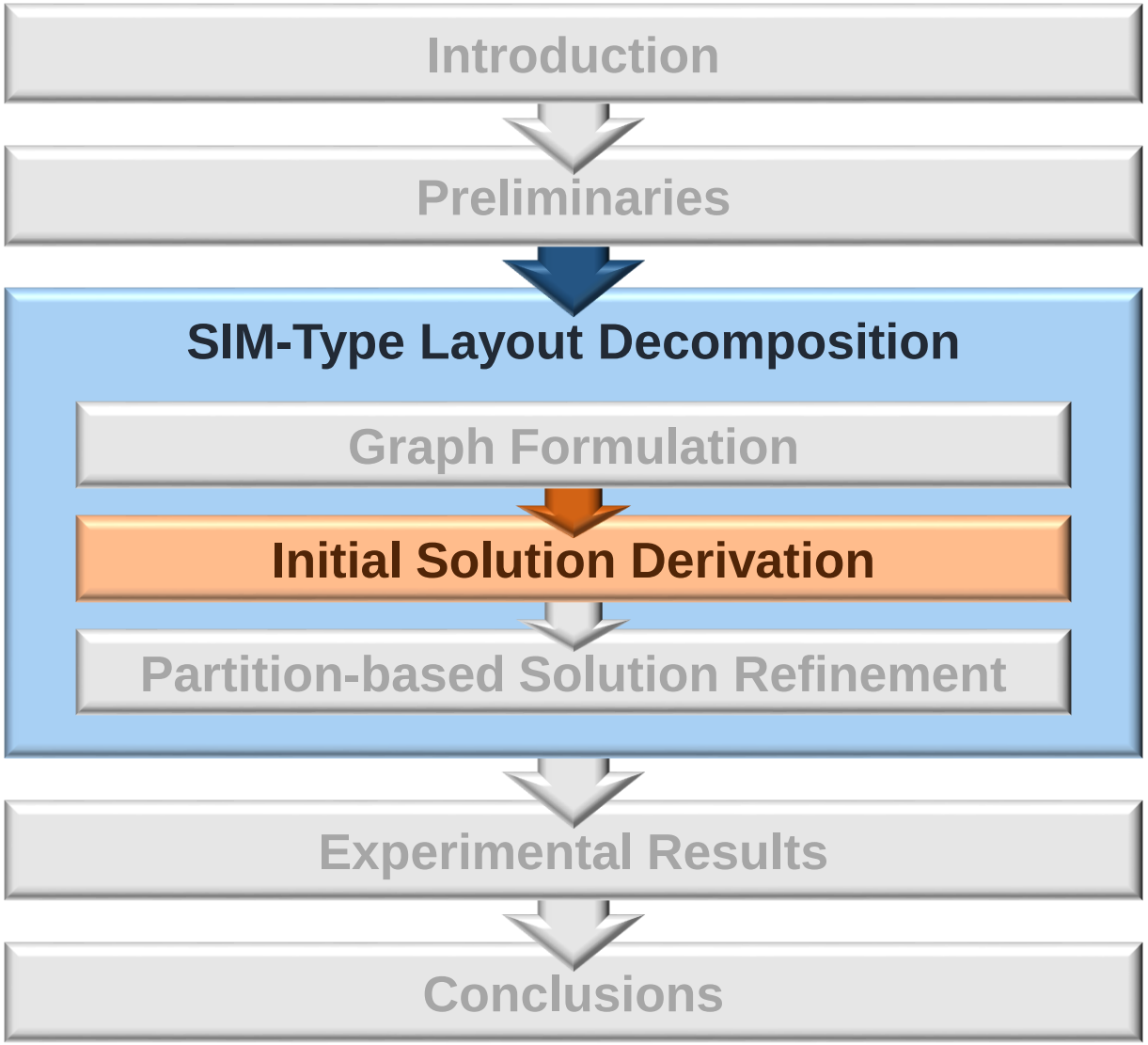


Graph Formulation

- Construct the combination graph G of a core-feature matching graph G_M and a conflict graph G_C
- **Problem** ***Constrained Set Covering Problem (SCP)***
  - **Given:** a combination graph
  - **Objective:** a set cover $\mathbb{C}$ with the minimized #conflicts
    - A set cover $\mathbb{C}$: a set of core vertices
    - #conflicts: #conflict edges within $\mathbb{C}$
  - **Constraint:** all feature vertices are covered by $\mathbb{C}$



Outline


Introduction
Preliminaries
SIM-Type Layout Decomposition
Graph Formulation
Initial Solution Derivation
Partition-based Solution Refinement
Experimental Results
Conclusions

Initial Solution Derivation

- Greedy principle: select the most ***cost-effective*** core vertex into the set cover $\mathbb{C}$ at a time
  - $covering(v_i)$: #uncovered feature vertices covered by v_i
  - $conflict(v_i)$: increased #conflict edges due to the addition of v_i into $\mathbb{C}$
  - $effectiveness(v_i) = covering(v_i) - \alpha \cdot conflict(v_i)$
- **Algorithm** ***Greedy Algorithm for Constrained SCP***

  Step 1 Compute $effectiveness(v_i)$ for each core vertex v_i

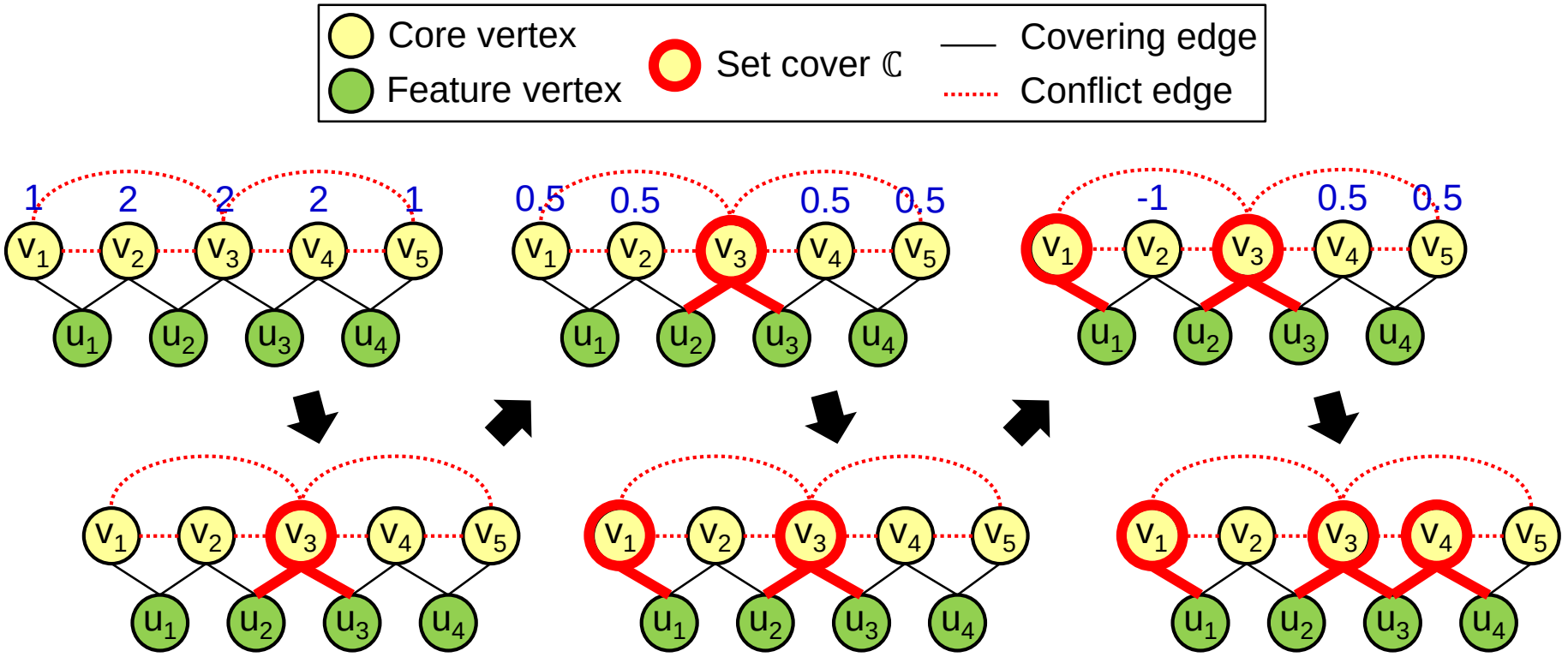
  Step 2 Add v_j with the largest $effectiveness(v_j)$ into $\mathbb{C}$

  Step 3 Update $covering(v_i)$ and $conflict(v_i)$ for each $v_i \in \bar{\mathbb{C}}$

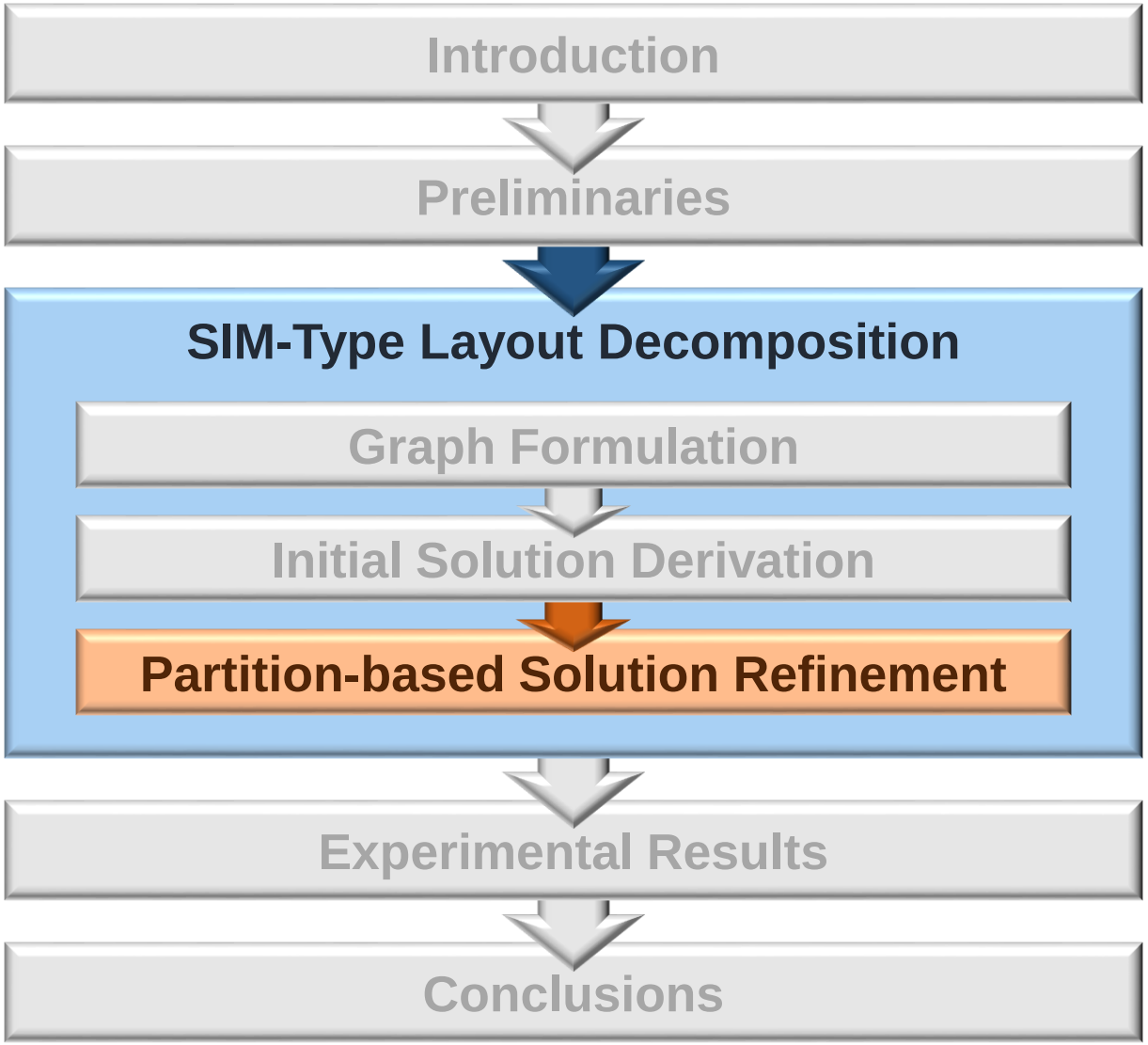
  Repeat Steps 1 to 3 until all feature vertices are covered

Example of the Greedy Algorithm

- $effectiveness(v_i) = covering(v_i) - 0.5 \cdot conflict(v_i)$

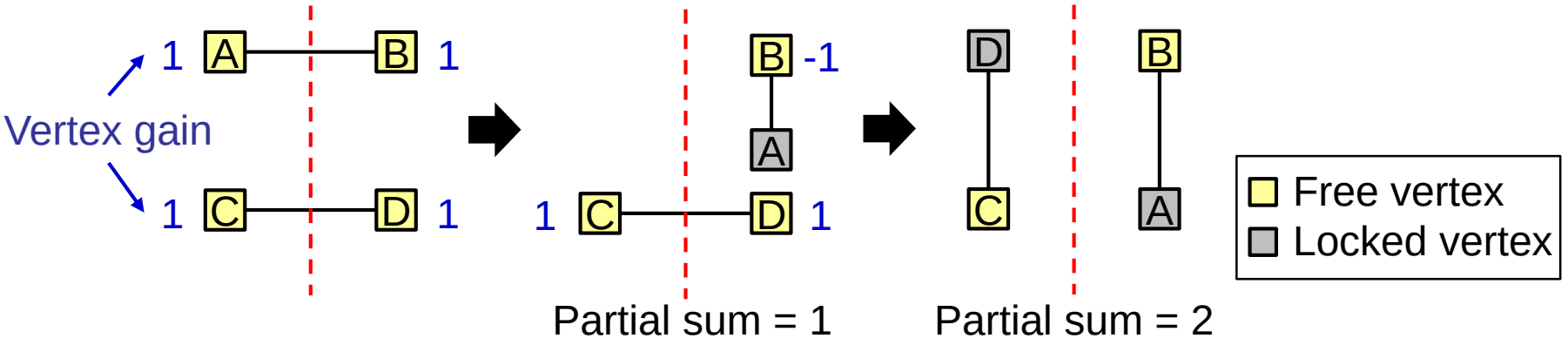


Outline


Introduction
Preliminaries
SIM-Type Layout Decomposition
Graph Formulation
Initial Solution Derivation
Partition-based Solution Refinement
Experimental Results
Conclusions

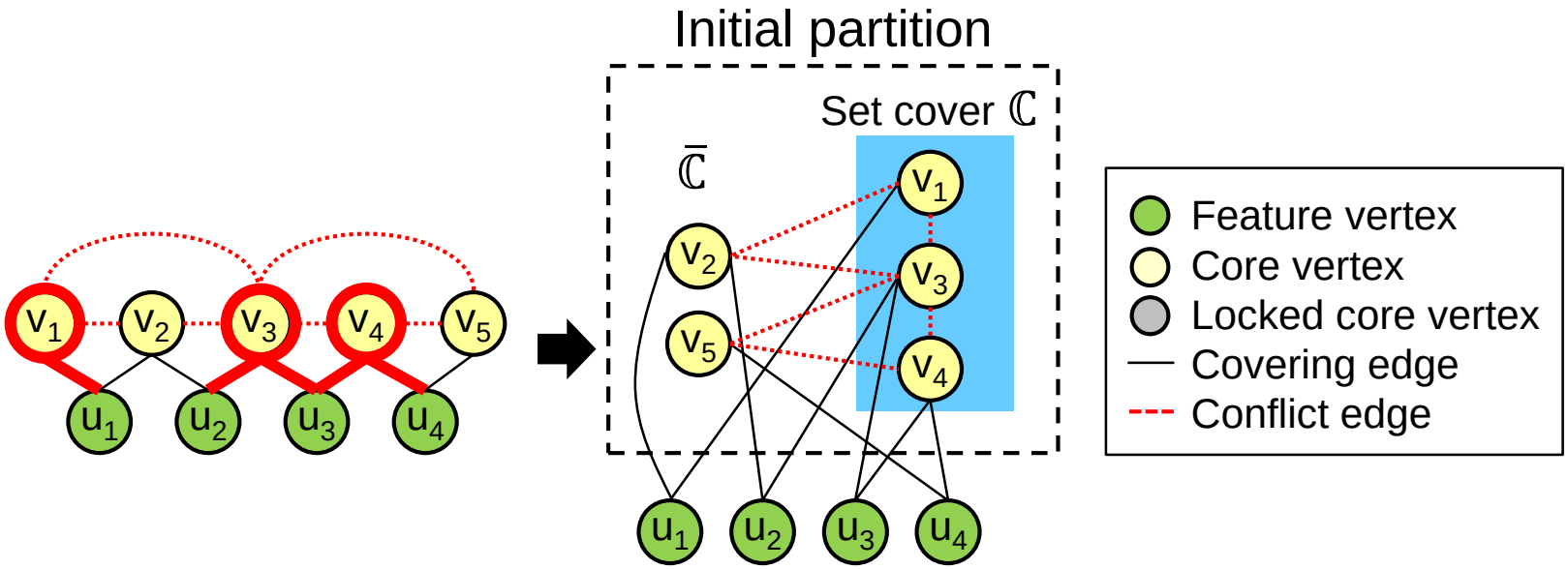
F-M Heuristic

- The solution refinement algorithm is based on the **F-M partitioning heuristic** [Fiduccia and Mattheyses, DAC'82]
  - Objective: minimize #cut edges in a partition
  - In each step, move a vertex with the largest gain and lock it
    - $gain(v) = -\Delta edge\_cut(v)$
  - In each iteration, choose the partition with the maximum partial sum of gains as the refinement solution



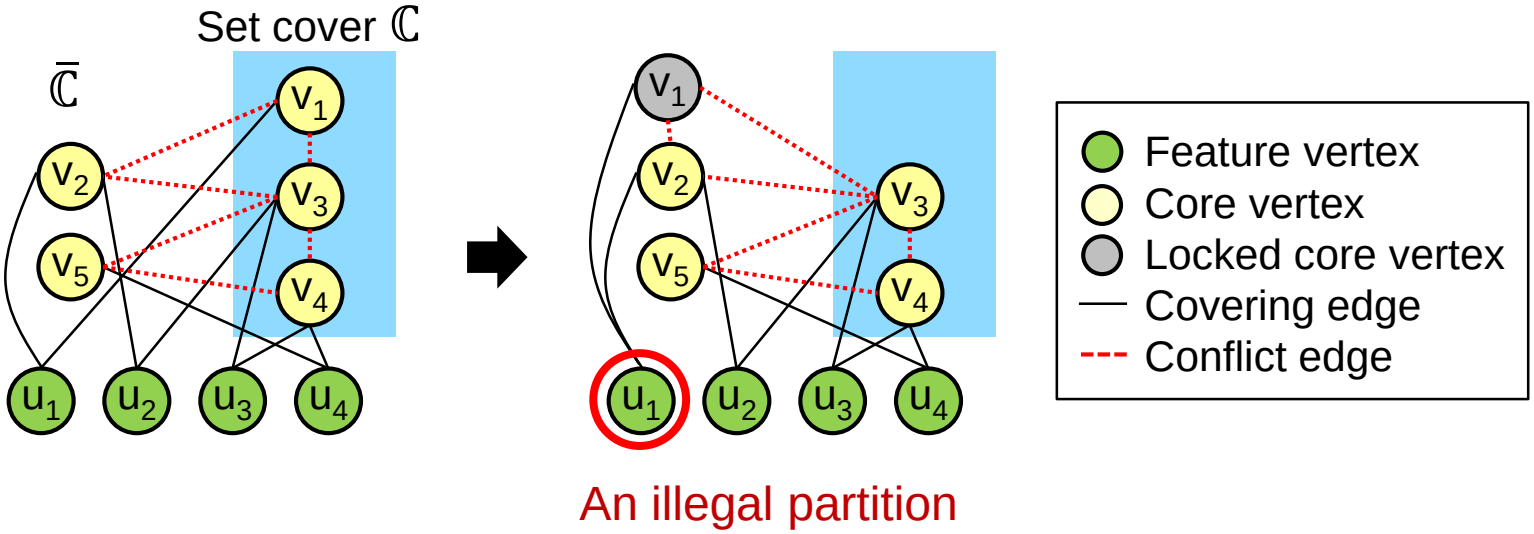
Partition-Based Solution Refinement

- Solution refinement for the constrained SCP
  - **Initial partition:** the initial solution from the greedy algorithm
  - **Objective:** minimize #conflicts in the set cover $\mathbb{C}$ of a partition
  - **Constraint:** the refinement solution (a partition with the maximum partial sum) must be legal in each iteration
    - A legal partition: all feature vertices are covered by the corresponding set cover $\mathbb{C}$



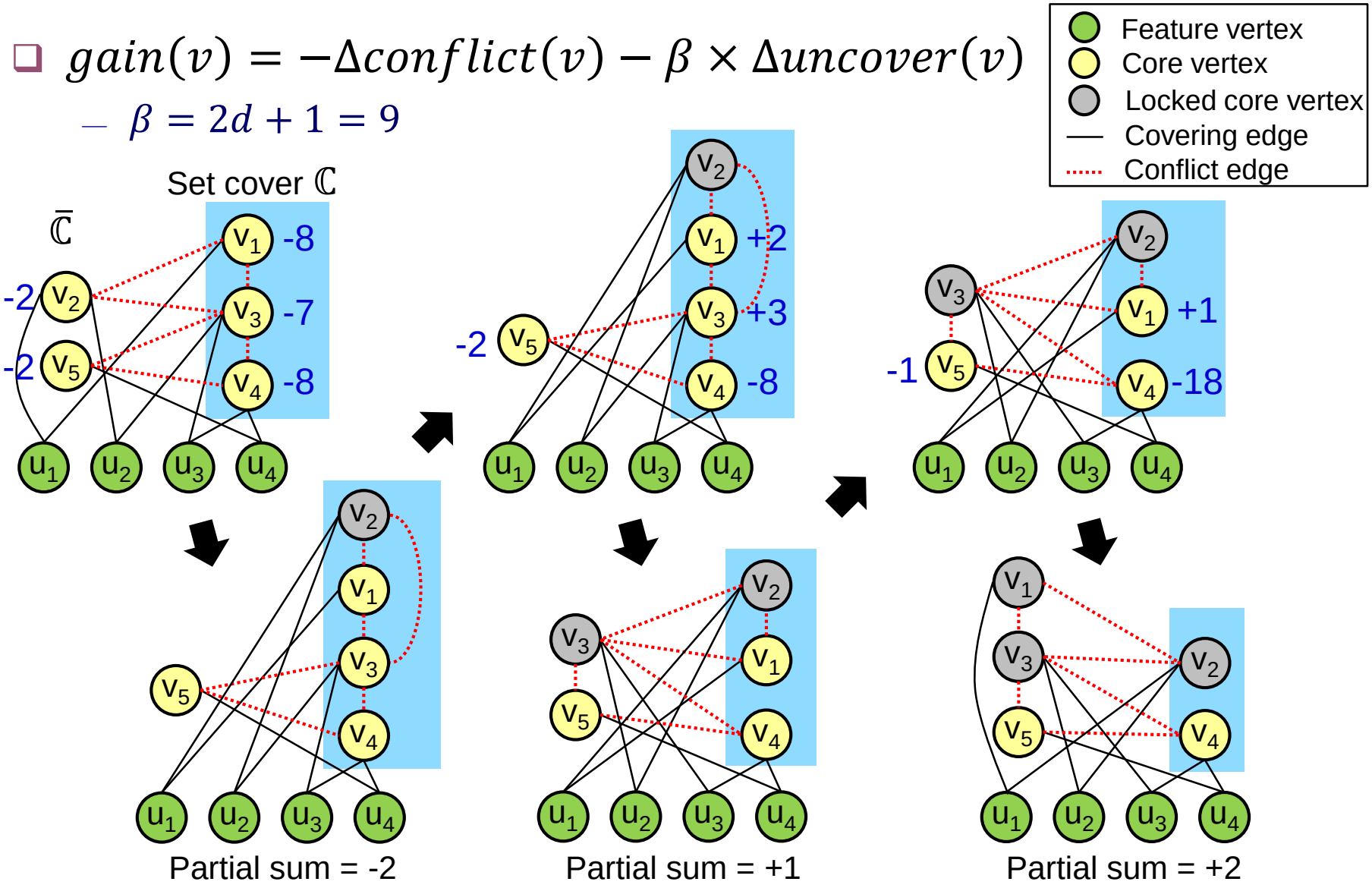
Partition-Based Solution Refinement (cont'd)

- Solution legalization by gain setting
  - **Vertex gain:** $gain(v) = -\Delta conflict(v) - \beta \times \Delta uncover(v)$
    - $\Delta conflict(v)$: variation of #conflicts due to the movement of v
    - $\Delta uncover(v)$: variation of #uncovered feature vertices
  - **Theorem** By setting $\beta > 2d$, the partition-based algorithm can always find a legal refinement solution in each iteration
    - d: maximum #conflict edges incident to a core vertex

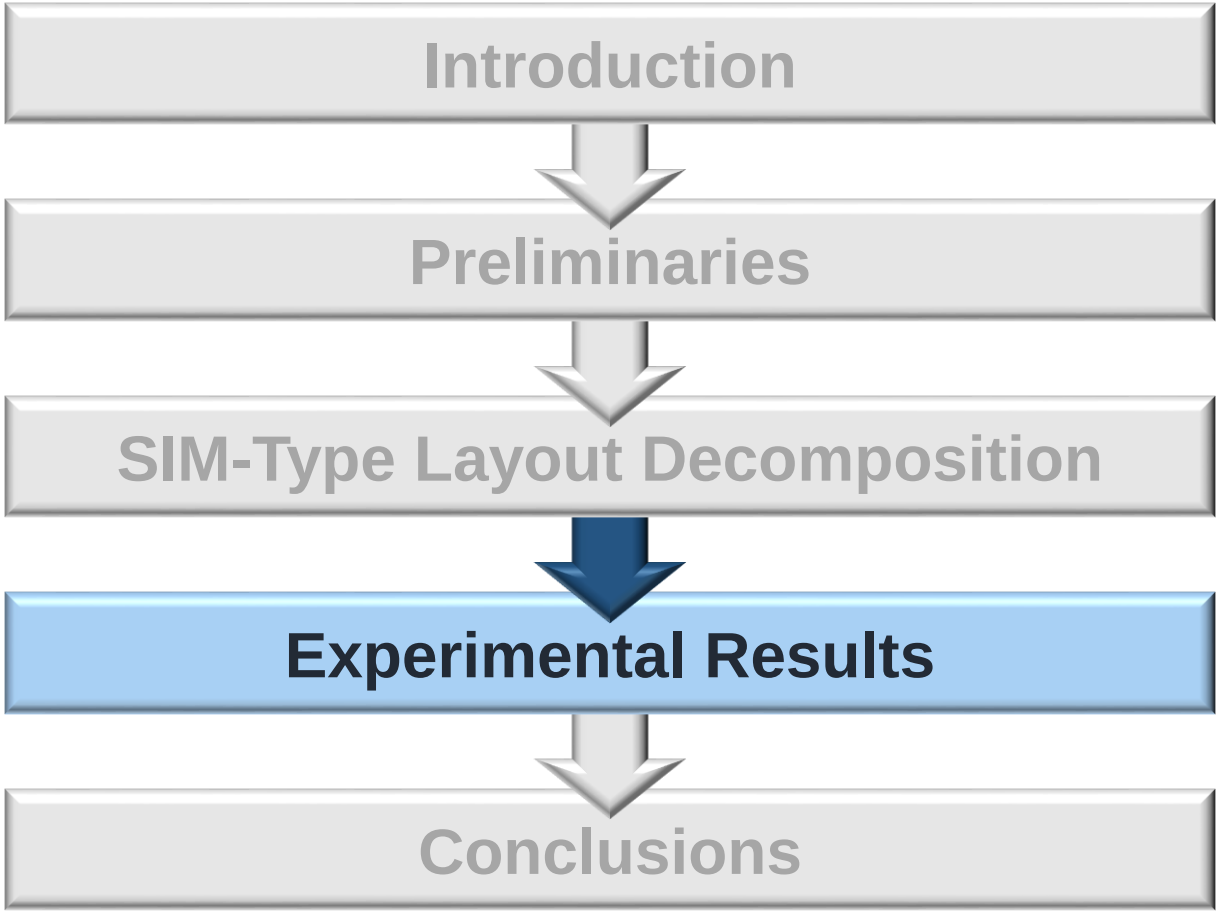


Example of Solution Refinement

- $gain(v) = -\Delta conflict(v) - \beta \times \Delta uncover(v)$
  - $\beta = 2d + 1 = 9$



Outline

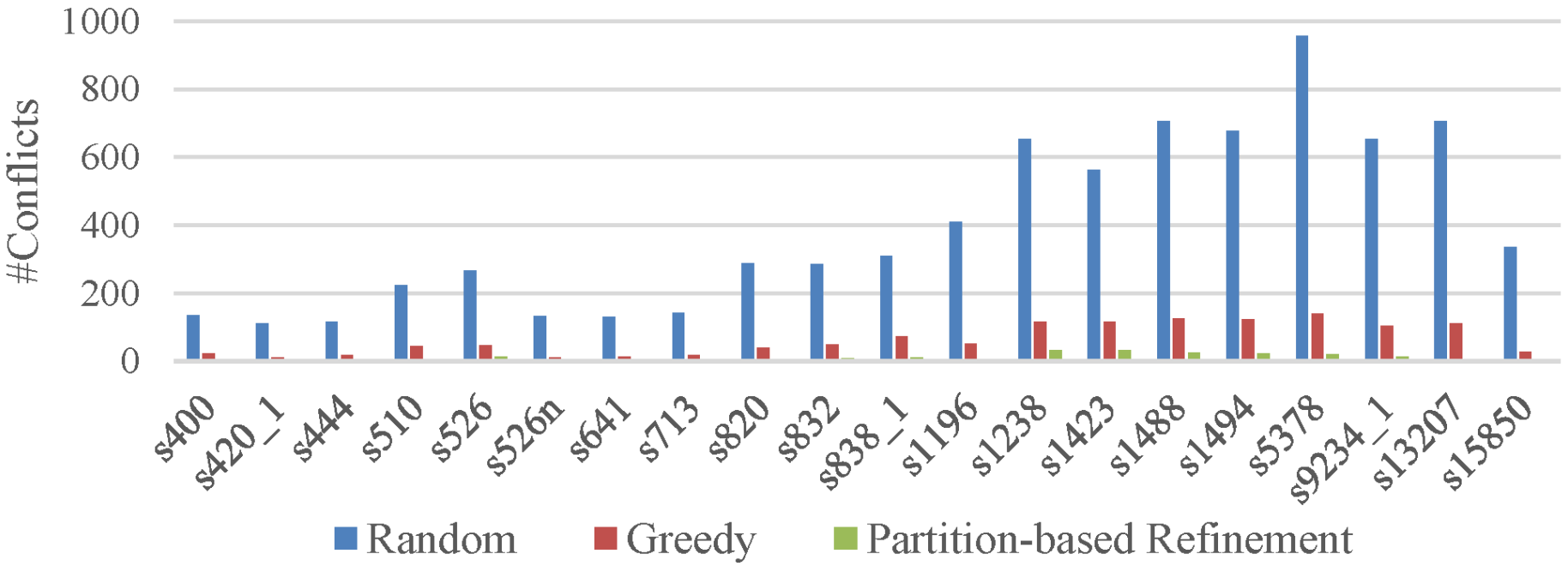

Introduction
Preliminaries
SIM-Type Layout Decomposition
Experimental Results
Conclusions

Experimental Setup

- Platform
  - C++ programming language
  - 2.13 GHz Linux workstation with 48 GB memory
- Benchmark
  - ISCAS-89 circuits
  - Routing wires in Metal-2, 2D gridless layouts, uniform wire width
  - Indecomposable layouts (previous study [Zhang et al., SPIE'11] is not applicable)
- Comparison
  - A random algorithm for the constrained SCP was implemented
  - Core vertices covering some uncovered feature vertices are randomly and sequentially selected

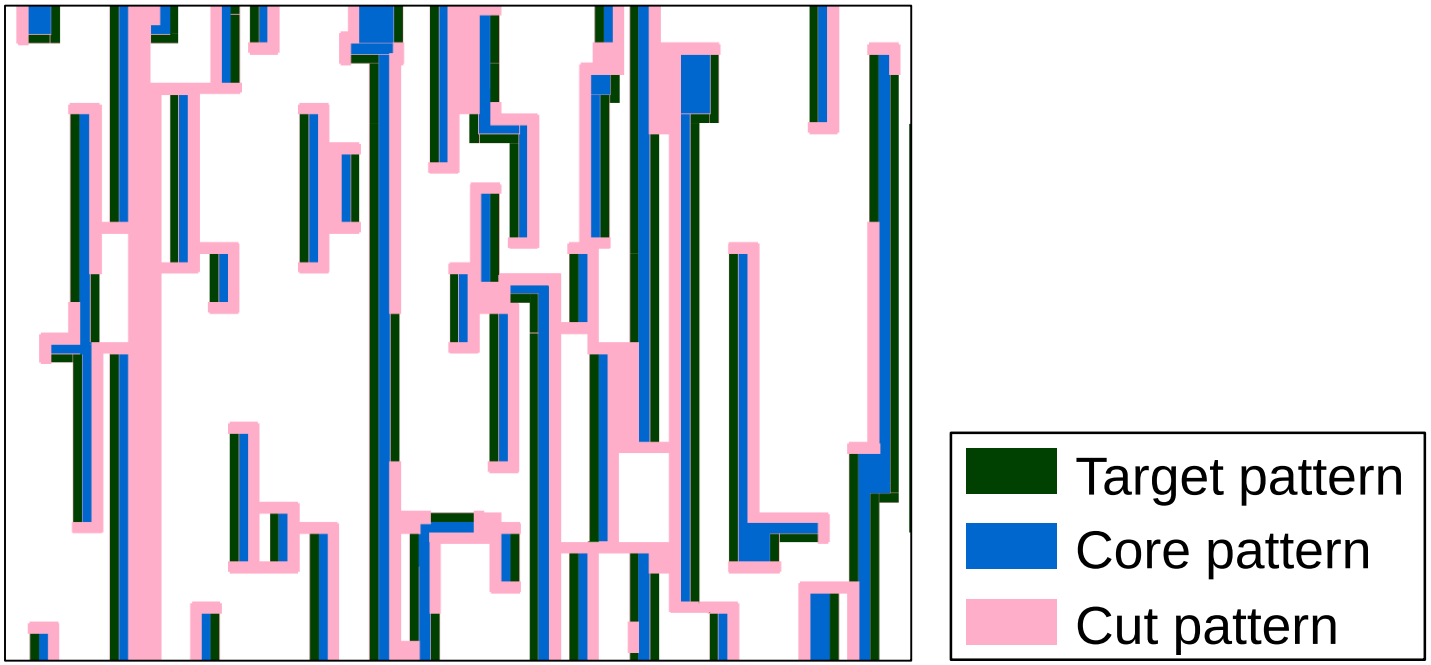
Experimental Results and Conclusion

- The greedy heuristic generates 7X fewer conflicts than the random method
- The partition-based solution refinement further reduce 84.7% #conflicts
- Our algorithms can effectively derive an SIM-type layout decomposition solution with desired numbers of core conflicts.

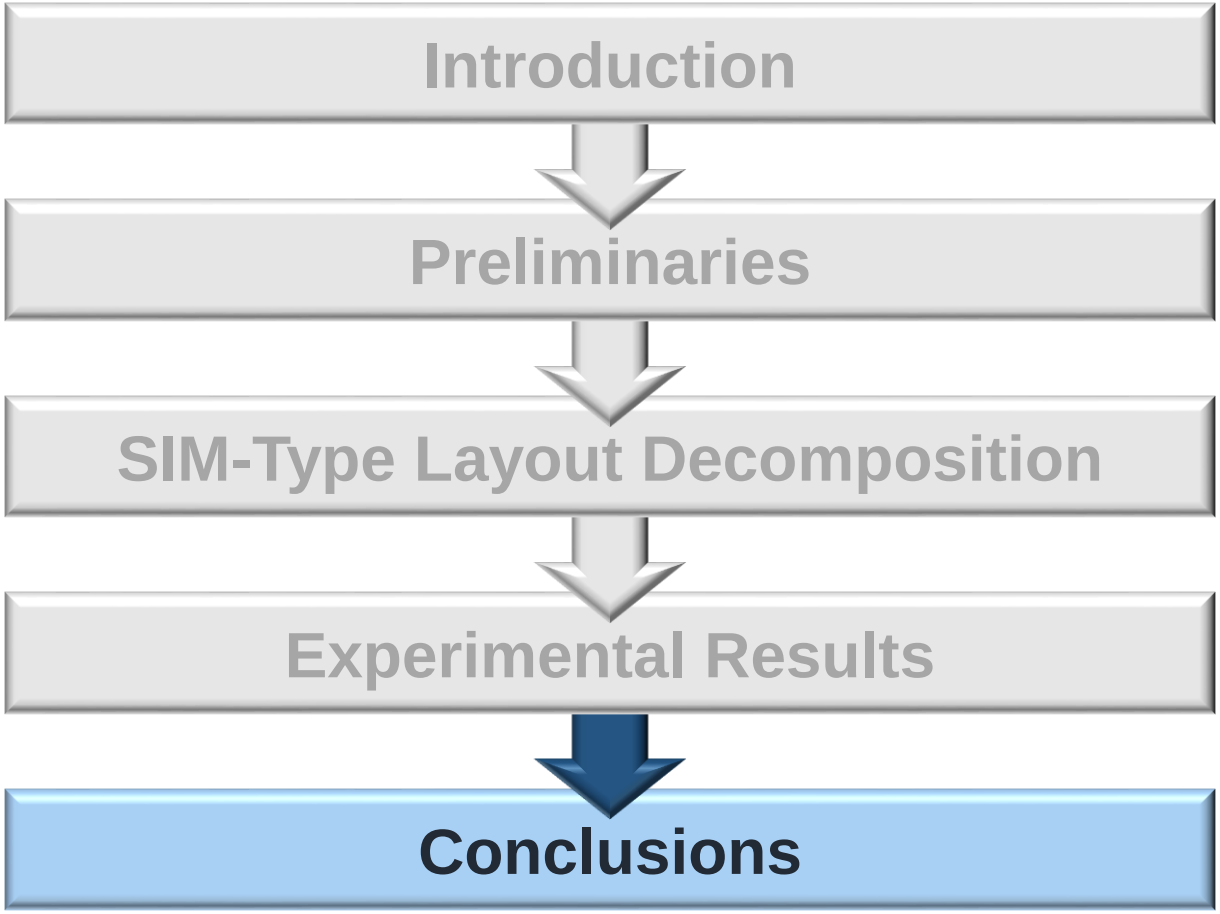


Layout Decomposition Result

- A local view of the layout decomposition result of s5378



Outline


Introduction
Preliminaries
SIM-Type Layout Decomposition
Experimental Results
Conclusions

Conclusions

- This paper proposes an efficient graph-based SIM-type layout decomposition algorithm for SADP
- The decomposition problem is transformed into a constrained SCP and solved with efficient heuristic algorithms
- Experimental results have shown that our algorithms can effectively derive good SIM-type layout decomposition solutions