# ECE 260C, Spring 2025

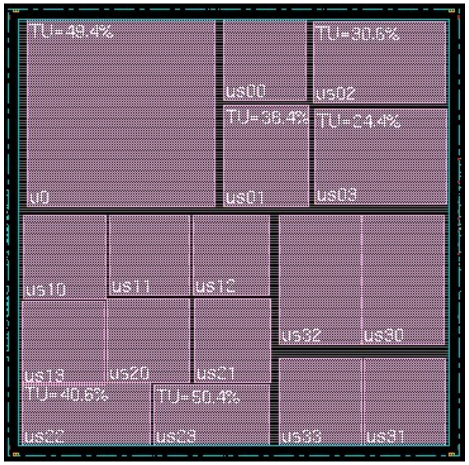
# Placement

Andrew B. Kahng

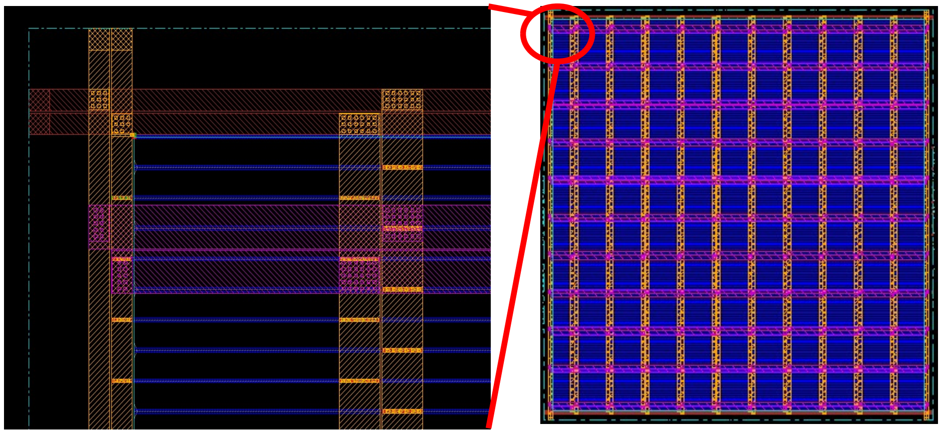
Thanks to: Andrew Kennings, Mingyu Woo, …

# Physical Design Flow Pictures (old ECE 260B slide)

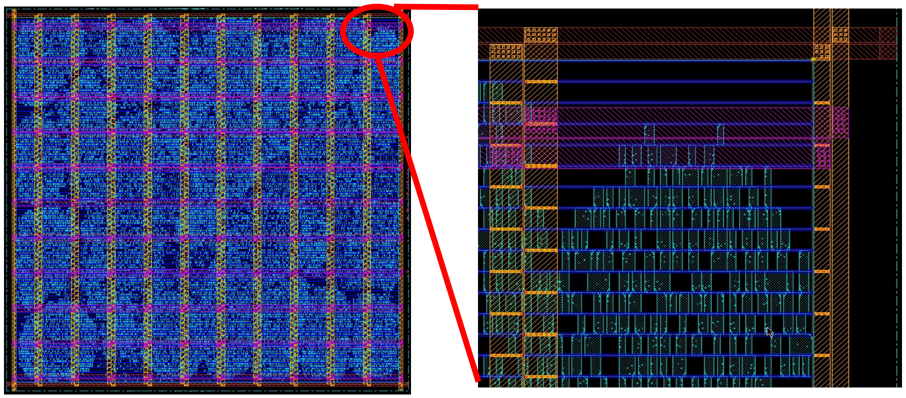
- Floorplanning



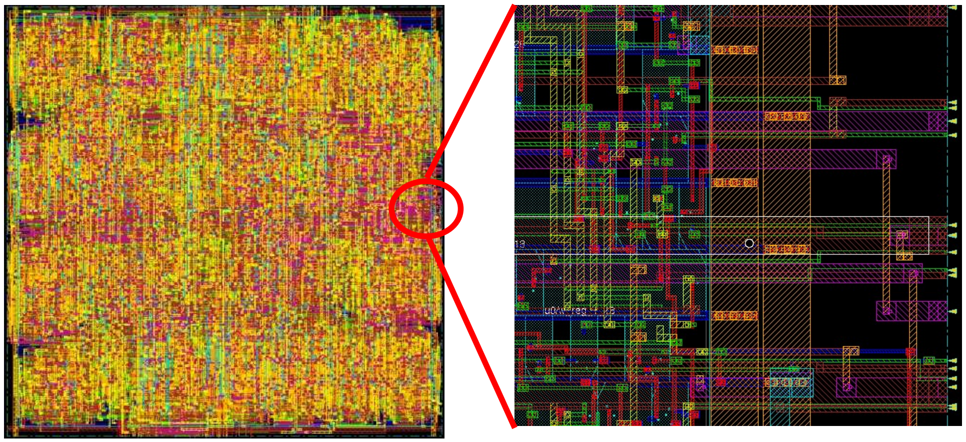
- Powerplanning



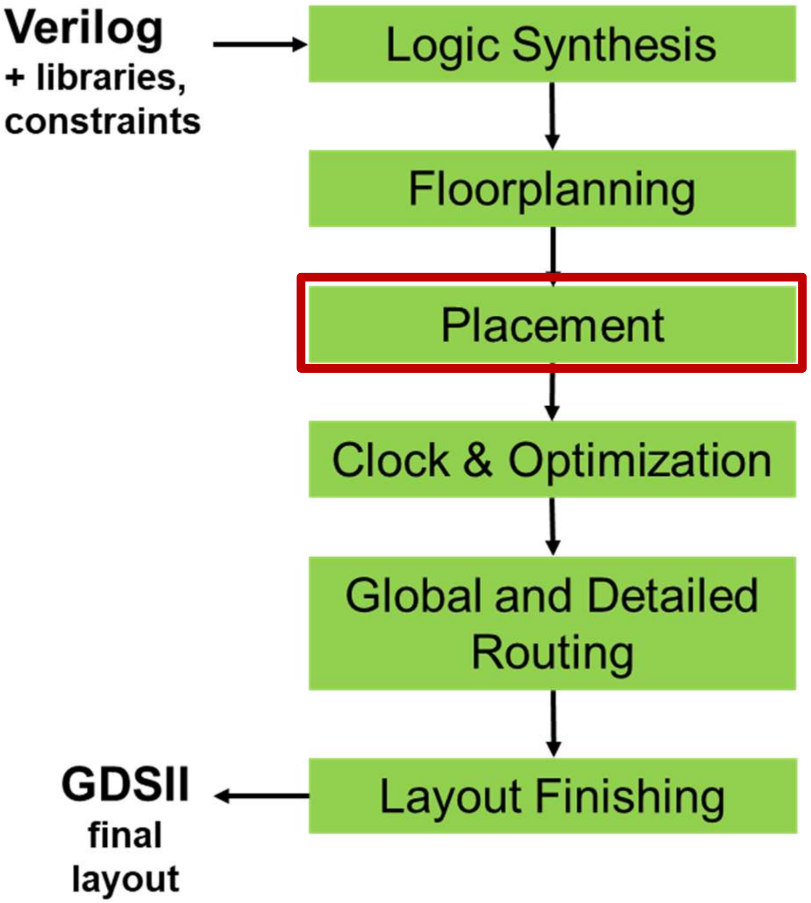
- Placement



- Routing



# Global Placement



**Verilog**
**+ libraries, constraints**

Logic Synthesis

Floorplanning

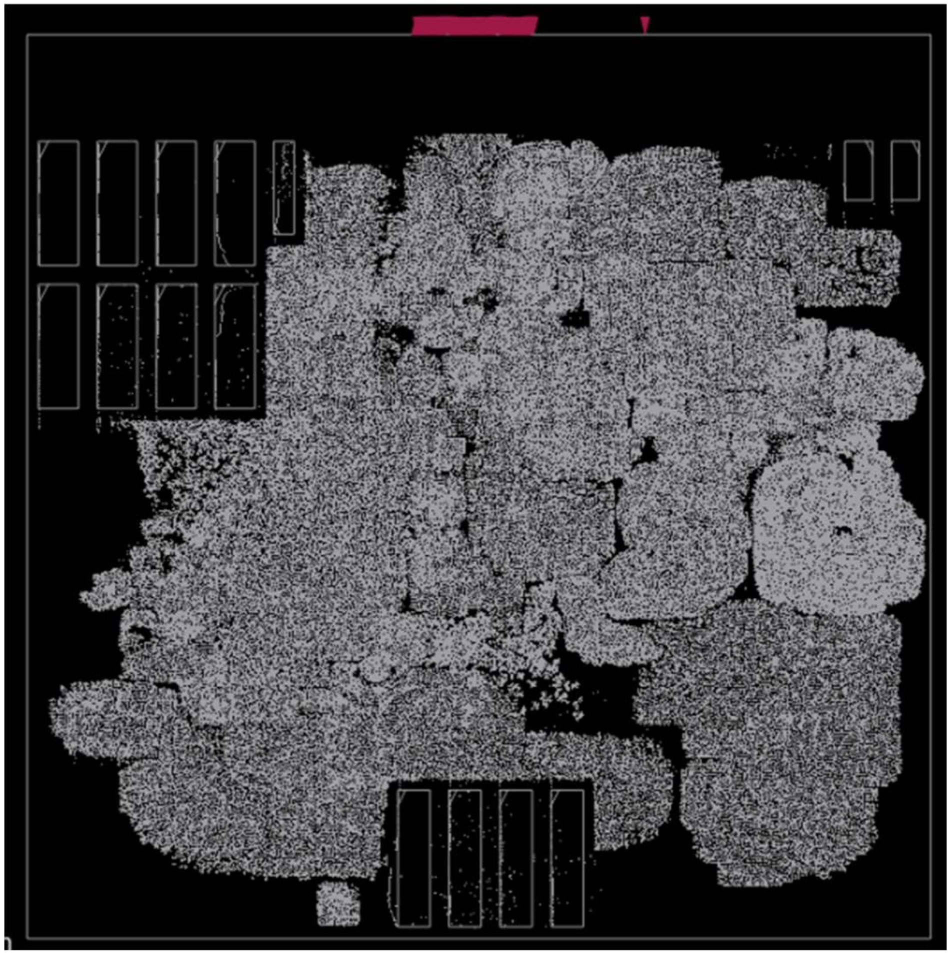
Placement

Clock & Optimization

Global and Detailed Routing

Layout Finishing

**GDSII**
**final layout**



# GPL

# RePlAce: Advancing Solution Quality and Routability Validation in Global Placement

**Chung-Kuan Cheng, Andrew B. Kahng, Ilgweon Kang and Lutong Wang**

(based on slides from Ilgweon Kang's Ph.D. defense)

paper

# Placement Overview

- ***Placement***
  - Determines the locations of standard cells and/or logic elements while addressing optimization objectives
  - Highly important physical design step in integrated circuit (IC) design flow
    - Directly impacts on timing closure, die utilization, routability, design turnaround time (TAT) → operating frequency, yield, power consumption, cost
- **Placement instances**
  - *Hypergraph* $G = (V, E)$
    - V:= a set of vertices, i.e., standard cells and macros
    - E:= a set of *hyperedges*, i.e., nets
- **Placement solution** $\boldsymbol{v = (x, y)}$, X- and Y-coordinates of all placeable vertices
  - *legal solution*?

1. Every instance should be settled in the placement region.
2. Every standard cell should be spaced within predefined rows.
3. No overlap is allowed between instances including both standard cells and macros.

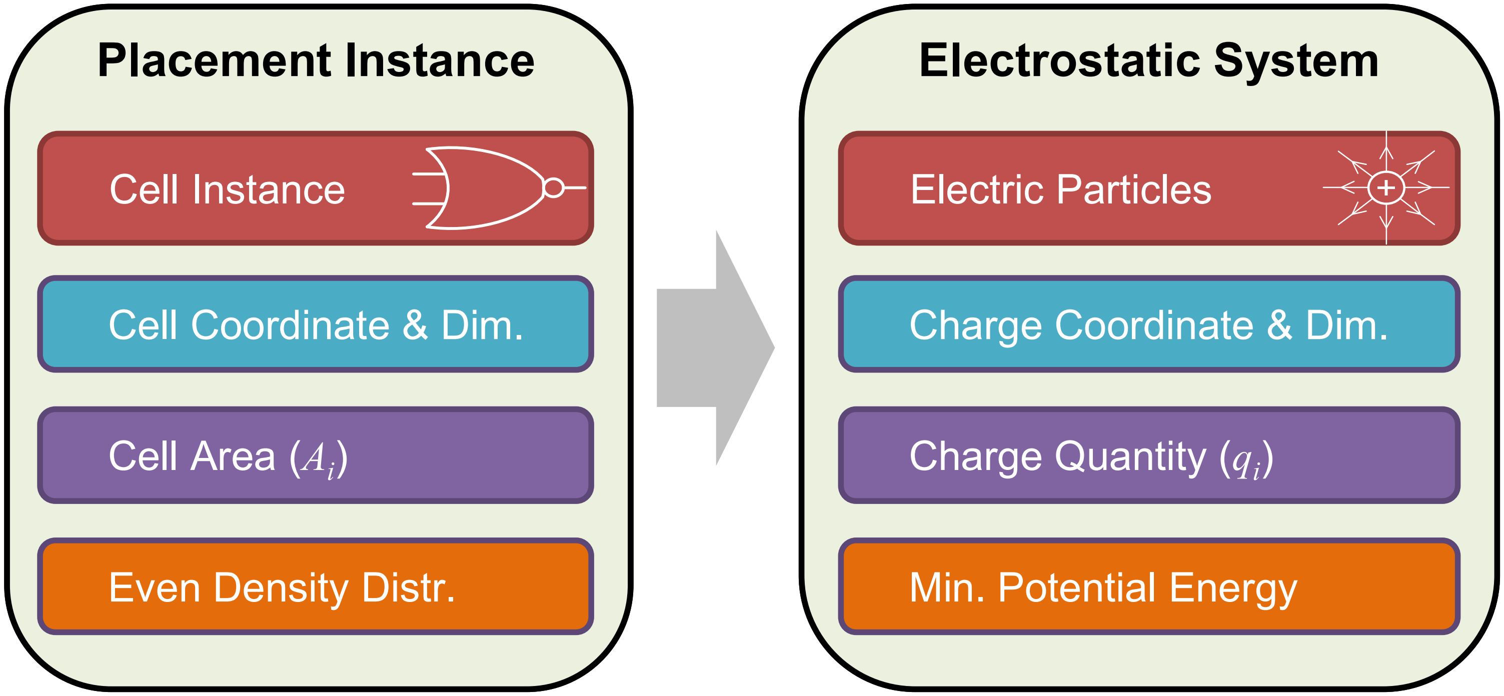
A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Springer, 2011

# Problem Formulation

- Placement Objective Function $f(v)$

$$\min_{\boldsymbol{v}} f(\boldsymbol{x},\boldsymbol{y}) = W(x,\boldsymbol{y}) + \sum_{b} \mathcal{V}_b\, D_b(x,y)$$

- ePlace: Electrostatics-based global-smooth density function



Placement Instance

Cell Instance

Cell Coordinate & Dim.

Cell Area ($A_i$)

Even Density Distr.

Electrostatic System

Electric Particles

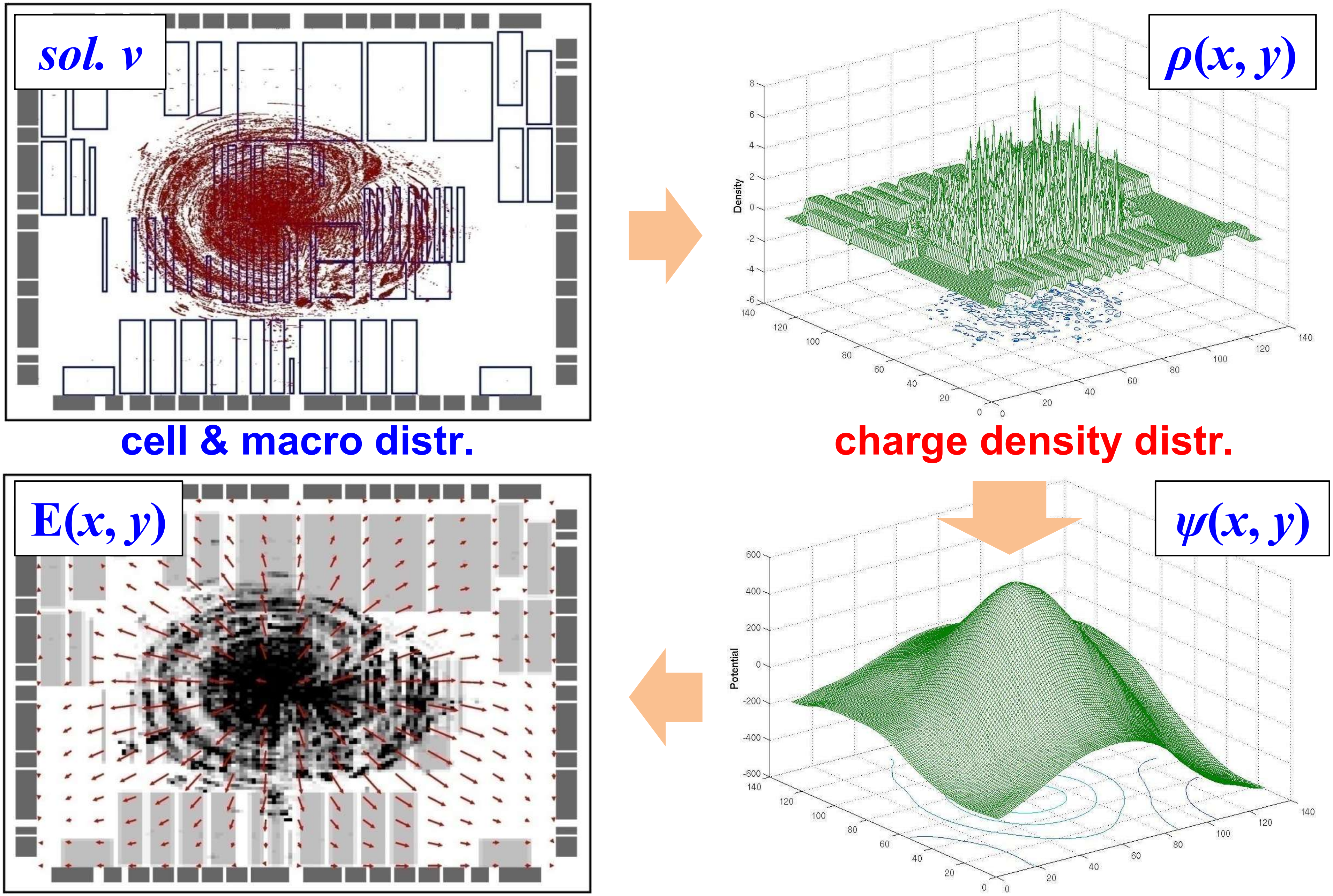
Charge Coordinate & Dim.

Charge Quantity ($q_i$)

Min. Potential Energy

[13] J. Lu, "Analytic VLSI Placement Using Electrostatic Analogy", Ph.D. Dissertation, UC San Diego, CA, 2014.

# ePlace:  In Each Iteration



*sol. v*

cell & macro distr.

$\rho(x, y)$

charge density distr.

$\psi(x, y)$

electric potential distr.

E$(x, y)$

electric field distr.

J. Lu, “Analytic VLSI Placement Using Electrostatic Analogy”, Ph.D. Dissertation, UC San Diego, CA, 2014.

# RePlAce

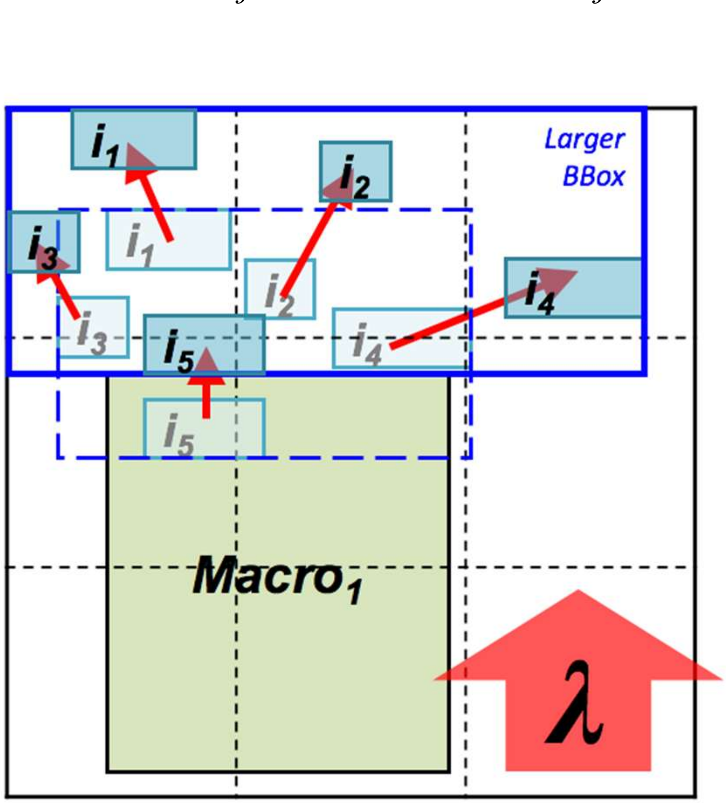
- Unified placement engine which solves multiple classes of academic benchmarks (mixed-size, fixed-macros, routability-driven, etc.
- Local Lagrange multiplier
  - Enables local smoothing w/awareness of local over-demanded bins
- Meta parameter tuning
  - Adjust step size of numerical method
- Routability optimization
  - Simple but effective metal layer-aware superlinear cell inflation
- Differences from the previous *ePlace 2.0*
  - Routability optimization techniques
  - Code optimizations for ~4X speedups
  - Various improvements to placement mechanisms
    - E.g., macro legalization using annealing; amount of rollback after fixing macro; handling of overflows; bin size determination and local smoothing; …

# RePlAce: Constraint-Driven Placement

Local Lagrangian multiplier $v_j$

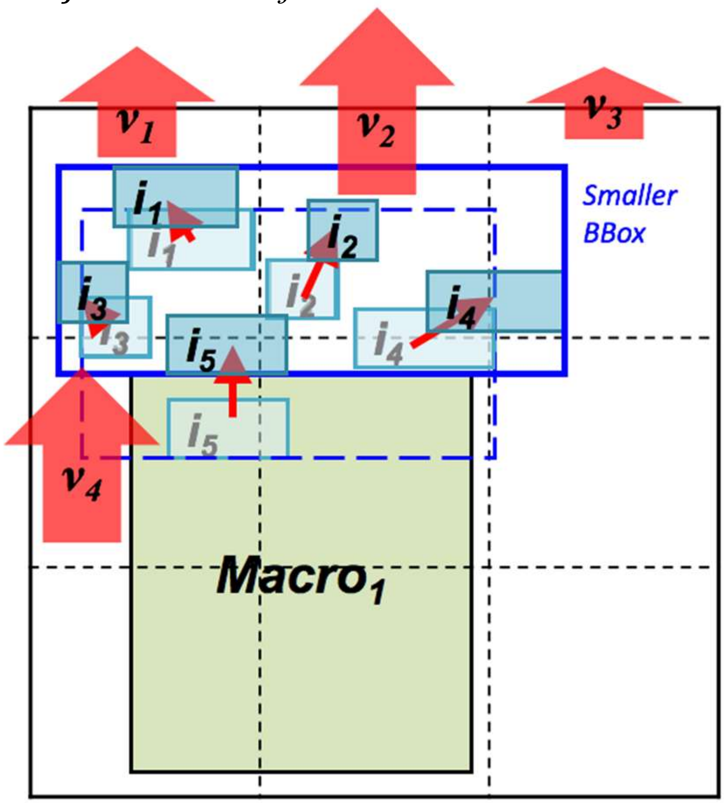
$$\nu_j = e^{\alpha(BinDemand_j - BinCapacity_j)}$$

- Local-Density Penalty Factor for each bin $b_j$
- $BinDemand_j$ = total cell area in bin $b_j$
- $BinCapacity_j$ = area of bin $b_j$
- $BinDemand_j$ - $BinCapacity_j$ = $overflow_j$ of bin $b_j$.



**Global $\lambda$ ↑, and Larger HPWL ↑**

[ePlace]



**Local $\nu_j$ ↑, and Smaller HPWL ↑**

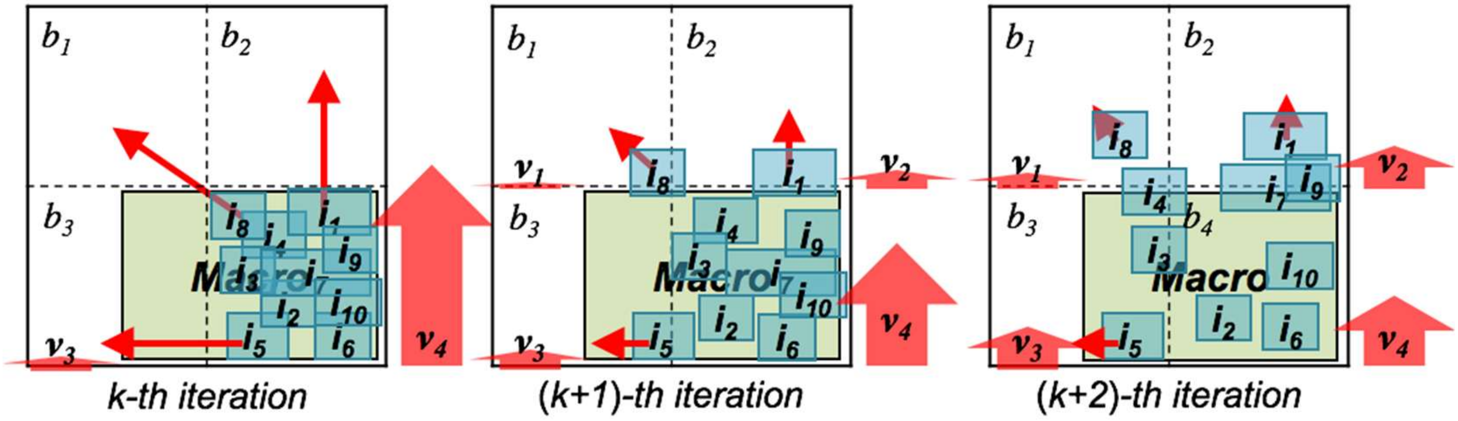
[RePlAce with Constraint-Driven]

# RePlAce:  Constraint-Driven Placement

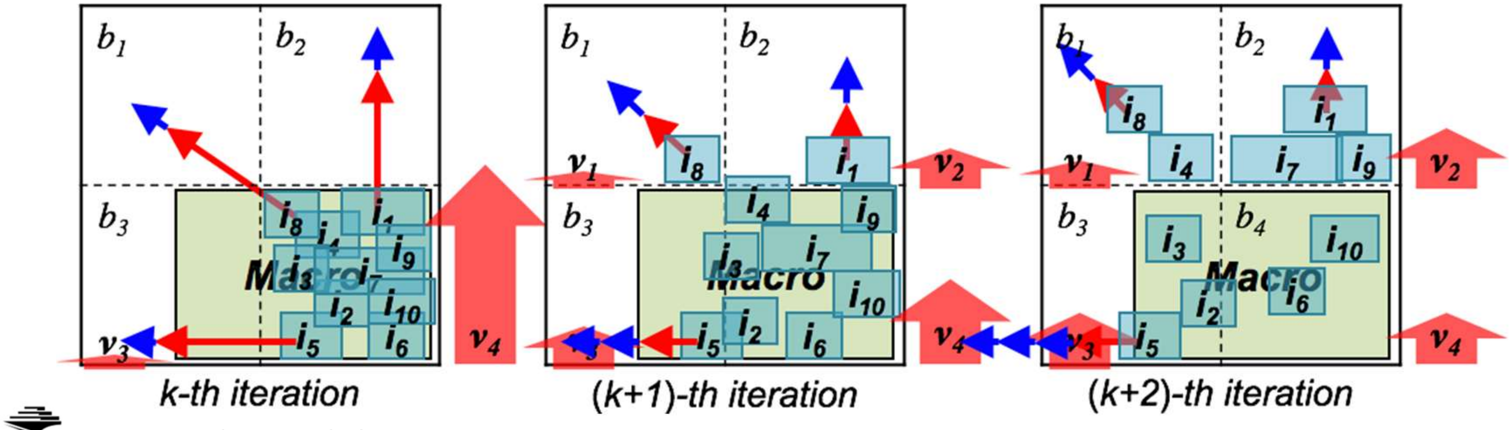
- Local-Density Cost Coefficient for each cell

$$\Delta_i^{iter+} = \Delta_i^{iter} + \beta \cdot \frac{\max(Overflow_j, 0)}{\sum_i A_i} \qquad i \in b_j.$$

  - The previously defined $v_j$ was insufficient for local smoothing
  - Cell $i$ from high-overflow bin losts a large amount of local-oriented repulsive force
  - Δi helps to memorizes previously obtained local force.



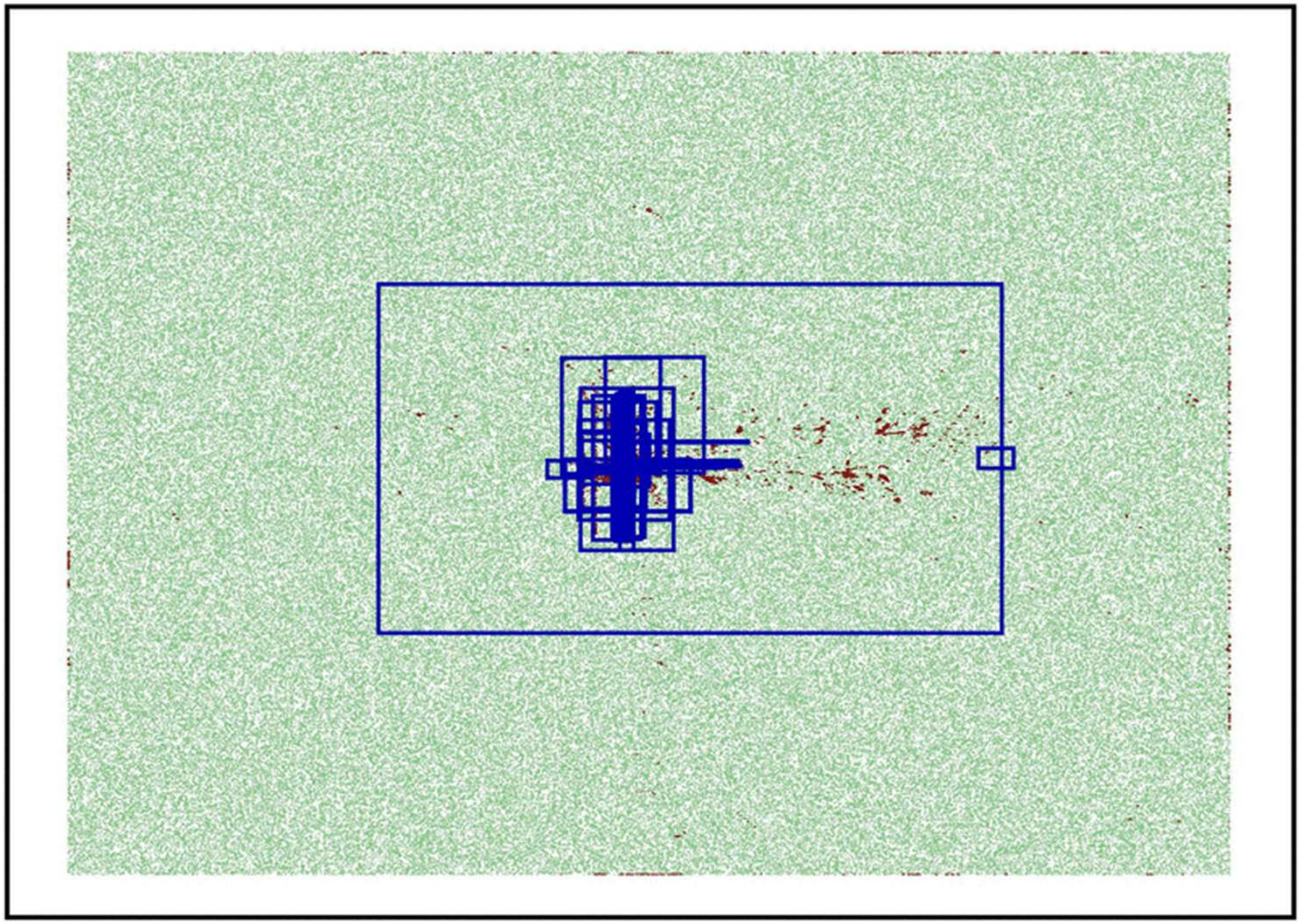
Local smoothing **without** local-density cost coefficient $\Delta_i$



Local smoothing **with** local-density cost coefficient $\Delta_i$

# RePlAce:  Constraint-Driven Placement

Global placement animation with local Lagrangian multiplier

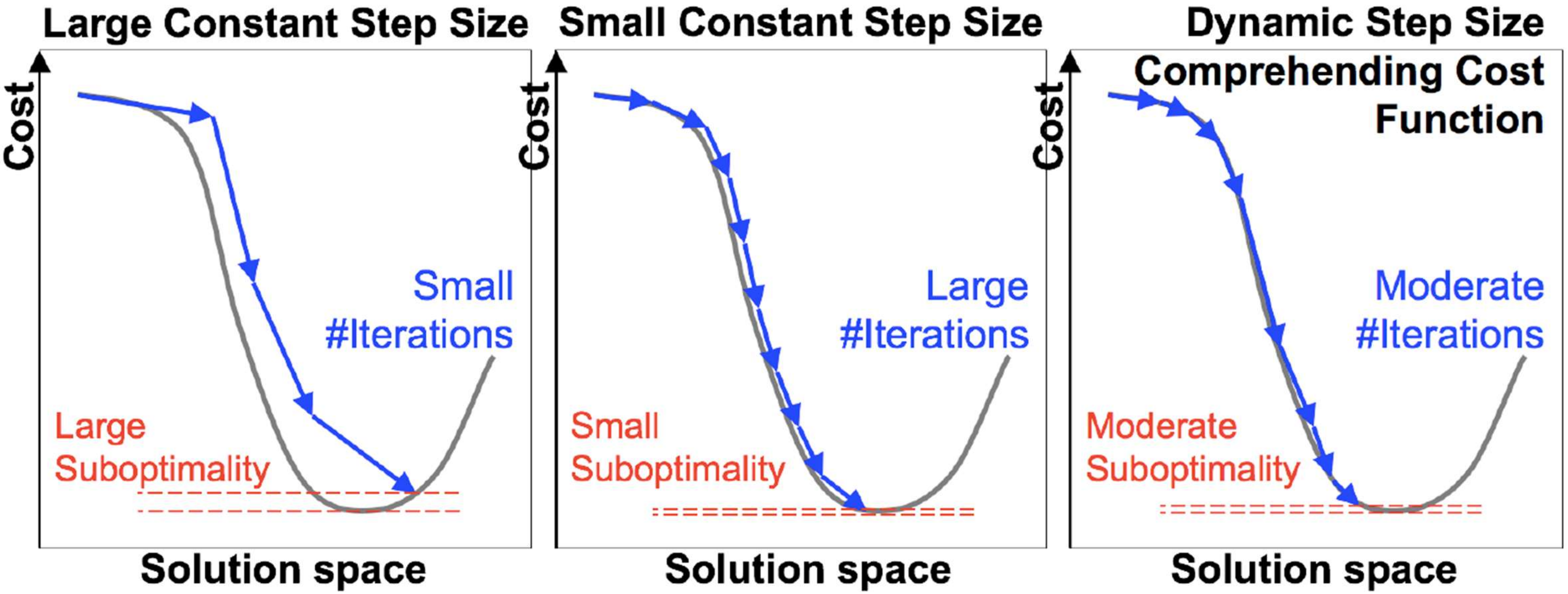


**NEWBLUE1**, [RePlAce with local Lagrangian multiplier, i.e., constraint-driven placement]
HPWL = 5.60E+7, #Iter = 762 (623+139), runtime = 9.9 min, target density = 100%

Comparison: ePlace 2.0 with local Lagrangian multiplier
HPWL = 5.71E+7, #Iter = 1078 (609 + 469), runtime = 42 min, target density = 100%

# RePlAce:  Dynamic Step Size

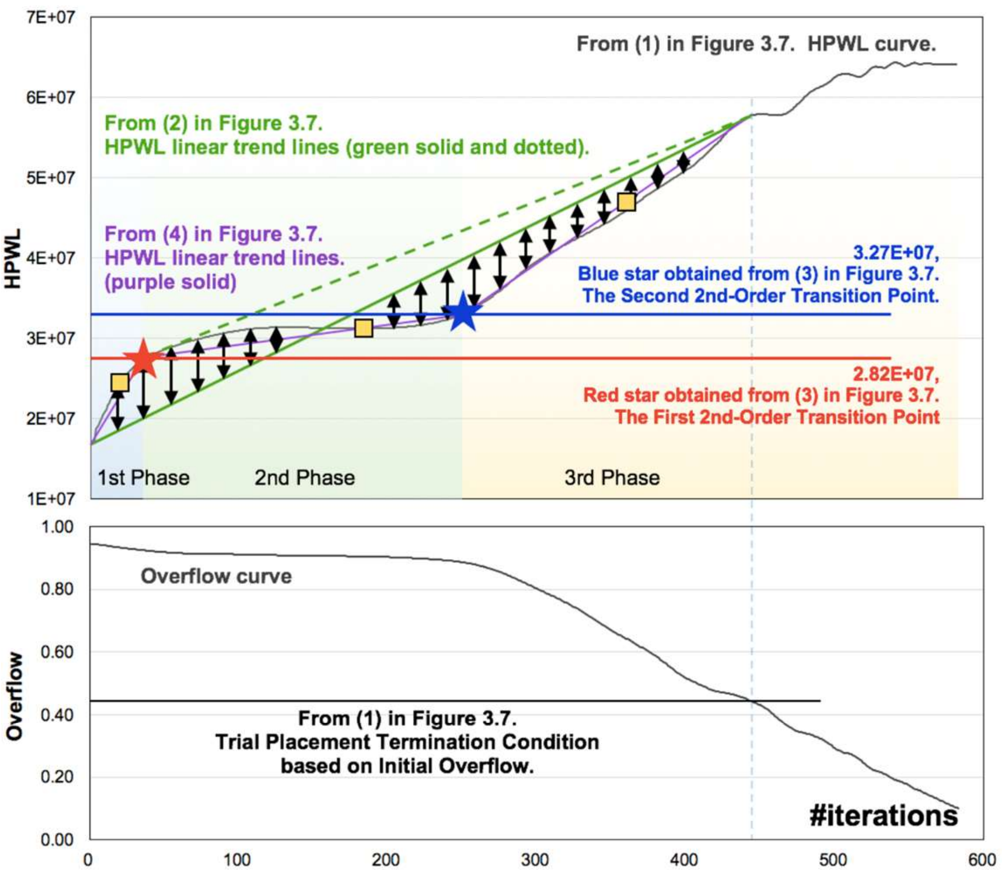
- General Idea of Dynamic Step Size



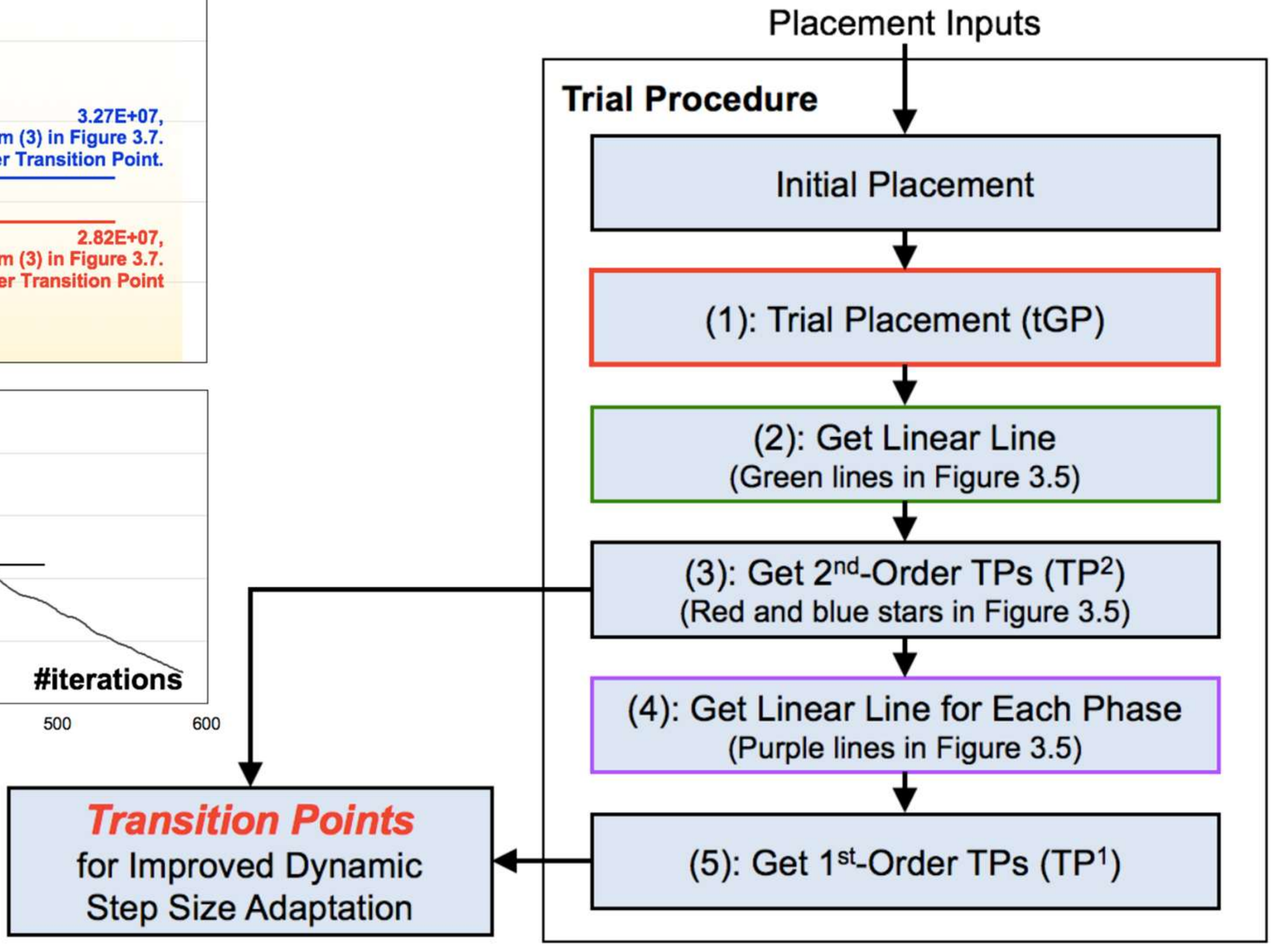
(a) (b) (c)

# RePlAce:  Improved Dynamic Step Size

Methodology to capture “transition points” on HPWL curve.



▲ HPWL curve of ADAPTEC1 (MMS) from the trial placement procedure and the estimated transition points (TPs)


Placement Inputs

Trial Procedure

Initial Placement

(1): Trial Placement (tGP)

(2): Get Linear Line
(Green lines in Figure 3.5)

(3): Get 2nd-Order TPs (TP²)
(Red and blue stars in Figure 3.5)

(4): Get Linear Line for Each Phase
(Purple lines in Figure 3.5)

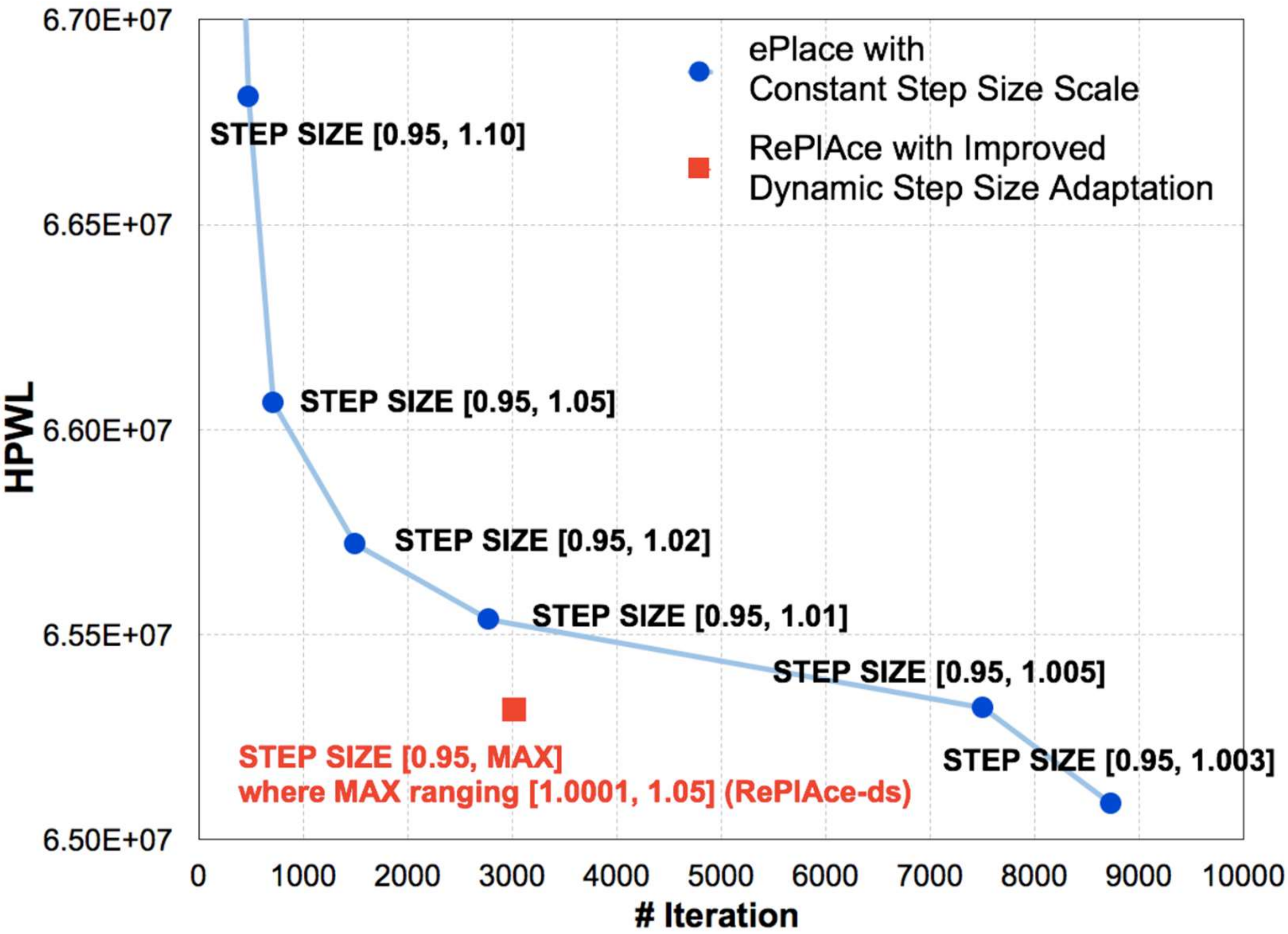
(5): Get 1st-Order TPs (TP¹)

*Transition Points*
for Improved Dynamic
Step Size Adaptation

▲ Flowchart of trial placement procedure. The red rectangle indicates nonlinear optimization using Nesterov’s method. The actual placement procedure follows this tGP procedure.

# RePlAce:  Improved Dynamic Step Size

- Solution quality in terms of the final HPWL
  - ePlace vs. RePlAce-ds (ADAPTEC1)
    - RePlAce-ds achieves a dominating runtime and solution quality (red square)
    - Below [0.95, 1.003], e.g., [0.95, 1.002], [0.95, 1.001], etc., runs did not converge



# RePlAce: Routability-Driven Placement

- Simple but effective routability optimization
  - A layer-aware cell inflation technique
  - Integrate the official global router NCTU-GR [17] of the DAC-2012 [18] and ICCAD-2012 [19] benchmark suites for congestion estimation.
  - Superlinear cell inflation technique to mitigate global routing congestion during global placement.
  - We further include a post-placement optimization by [20]
    - Following the strategy of recent leading works [21] [22]

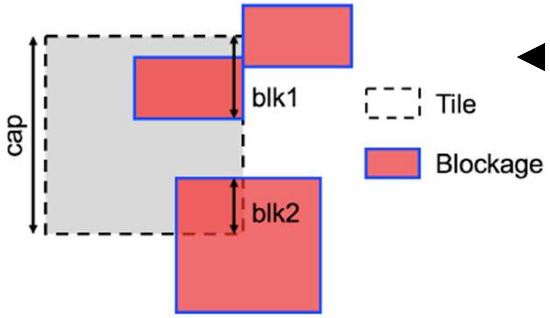
[17] NCTU-GR, http://people.cs.nctu.edu.tw/˜whliu/NCTU-GR.htm
[18] N. Viswanathan, C. J. Alpert, C. N. Sze, Z. Li and Y. Wei, “The DAC 2012 Routability-driven Placement Contest and Benchmark Suite”, *Proc. DAC*, 2012, pp. 774-782.
[19] N. Viswanathan, C. J. Alpert, C. N. Sze, Z. Li and Y. Wei, “ICCAD-2012 CAD Contest in Design Hierarchy Aware Routability-Driven Placement and Benchmark Suite”, *Proc. ICCAD*, 2012, pp. 345-348.
[20] W.-H. Liu, C.-K. Koh and Y.-L. Li, “Optimization of Placement Solutions for Routability”, *Proc. DAC*, 2013, pp. 1-9.
[21] X. He, T. Huang, L. Xiao, H. Tian and E. F. Y. Young, “Ripple: A Robust and Effective Routability-Driven Placer”, *IEEE Trans. on CAD* 32(10) (2013), pp. 1546-1556.
[22] X. He, Y. Wang, Y. Guo and E. F. Y. Young, “Ripple 2.0: Improved Movement of Cells in Routability-Driven Placement”, *ACM Trans. on DAES* 22(1) (2016), pp. 10:1-10:26.

# RePlAce:  Routability-Driven Placement

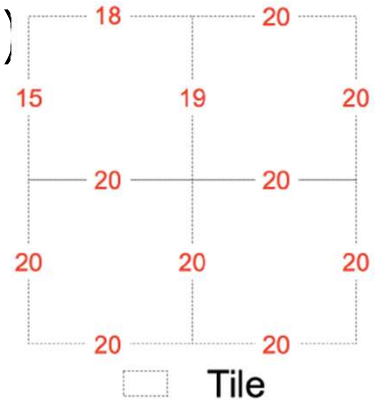
- Metal layer-aware superlinear cell inflation

$$infl\_ratio = \max_{all\ e,ml}\left(\left(\frac{demand_{e,ml}+blk_{e,ml}}{cap_{e,ml}}\right)^{\gamma_{super}}, 2.5\right)$$

  - $e$ = one of the four edges of a given global routing tile
  - $ml$ = a specific metal layer
  - $\gamma_{super}$ = 2.33 (empirically determined)



◀ Blockage calculation. For the vertical edge on the right, $blk = blk1 + blk2$. Note the union of blocked capacity for the upper two blockages.

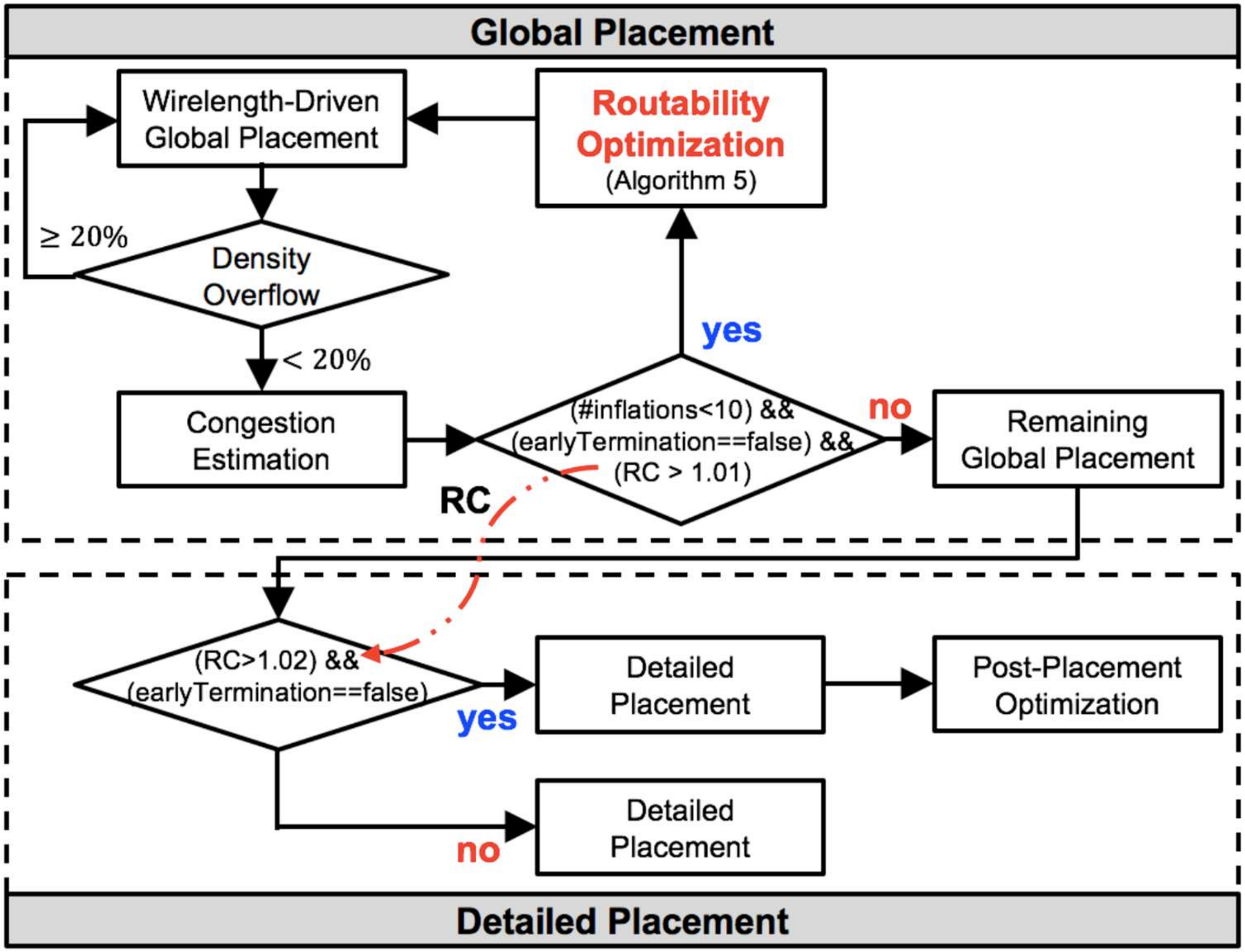


◀ Routing demand calculation: the upper-left tile has a horizontal routing demand of $\max(15,19)$ = 19

- Considers the total available whitespace
  - Starting from 90% die utilization,
  - We limit 'the maximum cell-inflated area' ≤ 10%
  - If exceed, then perform 'inflation ratio adjustment'
    - Divides the inflation ratio for each tile by the inflation ratio of the least-congested tile that has a ratio greater than one

# RePlAce: Routability-Driven Placement

- IEEE Trans. on CAD paper: routability optimization flow

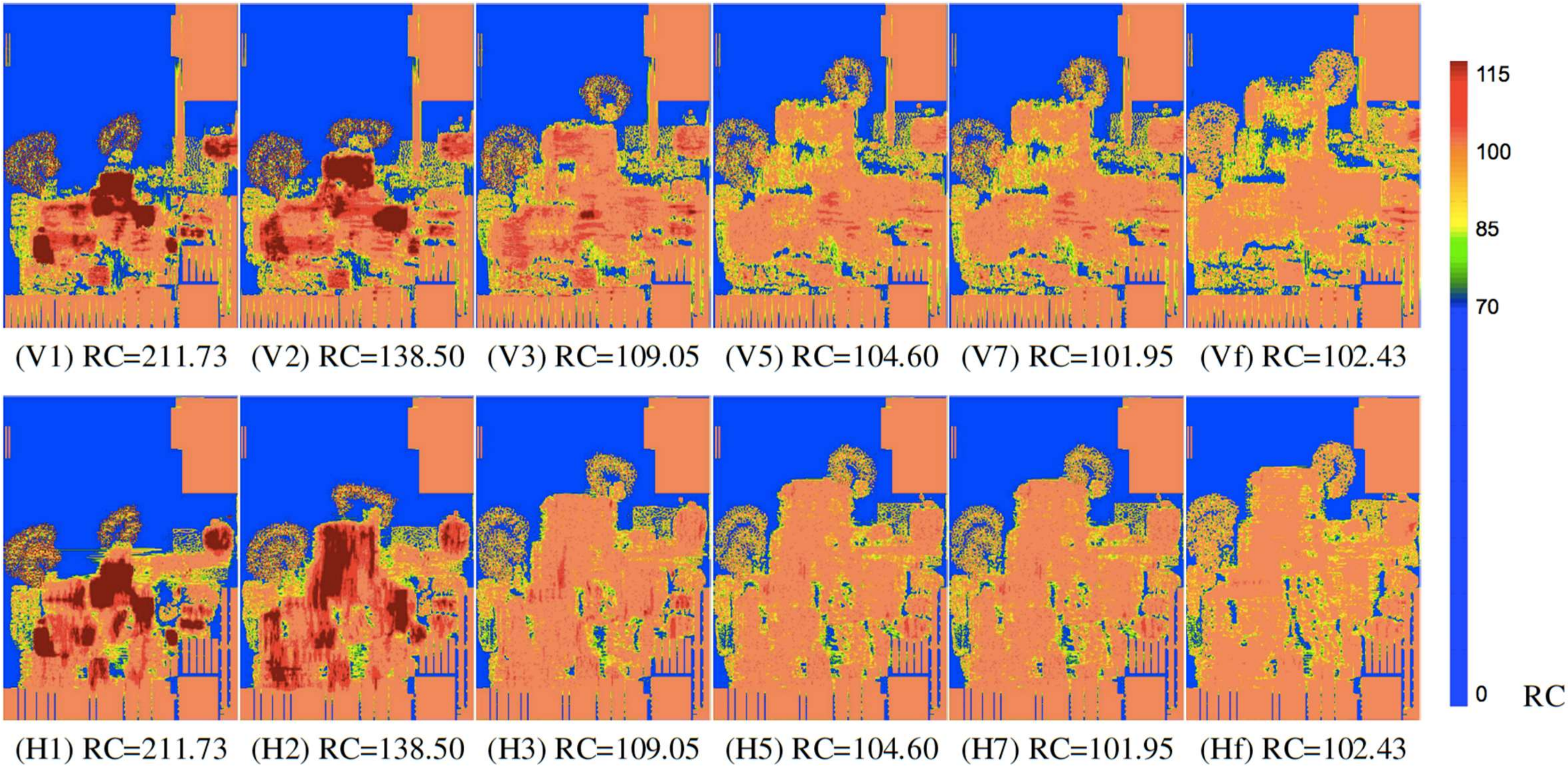

Global Placement
Wirelength-Driven Global Placement
Routability Optimization (Algorithm 5)
≥ 20%
Density Overflow
< 20%
yes
Congestion Estimation
(#inflations<10) && (earlyTermination==false) && (RC > 1.01)
no
Remaining Global Placement
RC
(RC>1.02) && (earlyTermination==false)
yes
Detailed Placement
Post-Placement Optimization
no
Detailed Placement
Detailed Placement

- Uses the detailed placer from *NTUplace3*

T.-C. Chen, Z.-W. Jiang, T.-C. Hsu, H.-C. Chen and Y.-W. Chang, “NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs with Preplaced Blocks and Density Constraints”, *IEEE Trans. on CAD* 27(7) (2008), pp. 1228-1240.

# RePlAce: Routability-Driven Placement

- Global routing overflow (SUPERBLUE12) during routability-driven global placement procedure

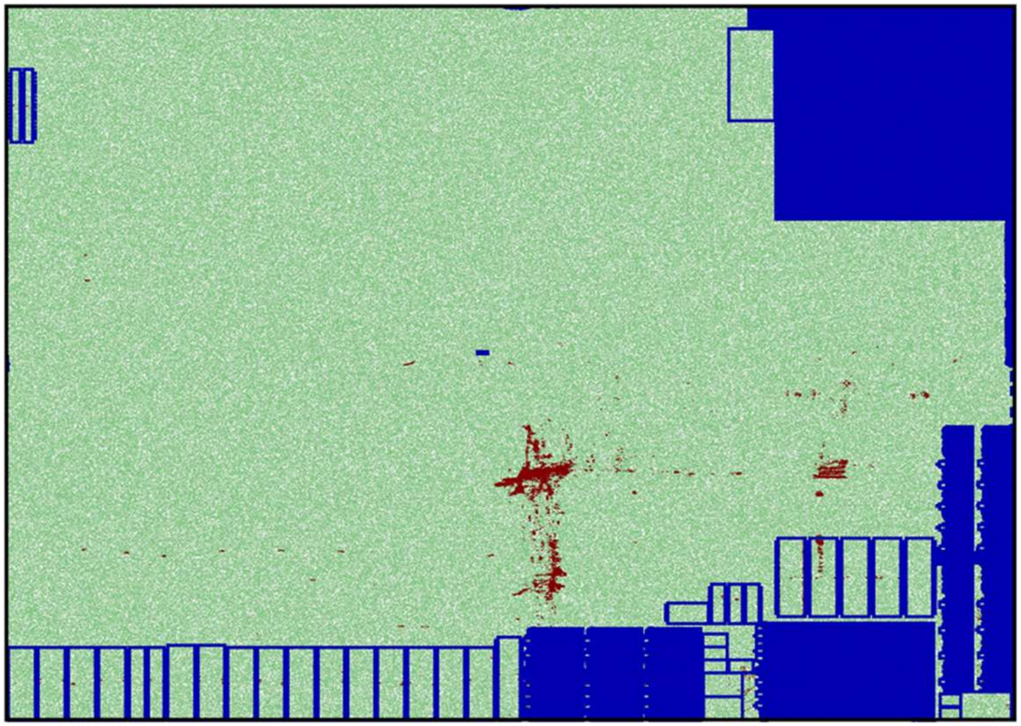


(V1) RC=211.73 (V2) RC=138.50 (V3) RC=109.05 (V5) RC=104.60 (V7) RC=101.95 (Vf) RC=102.43

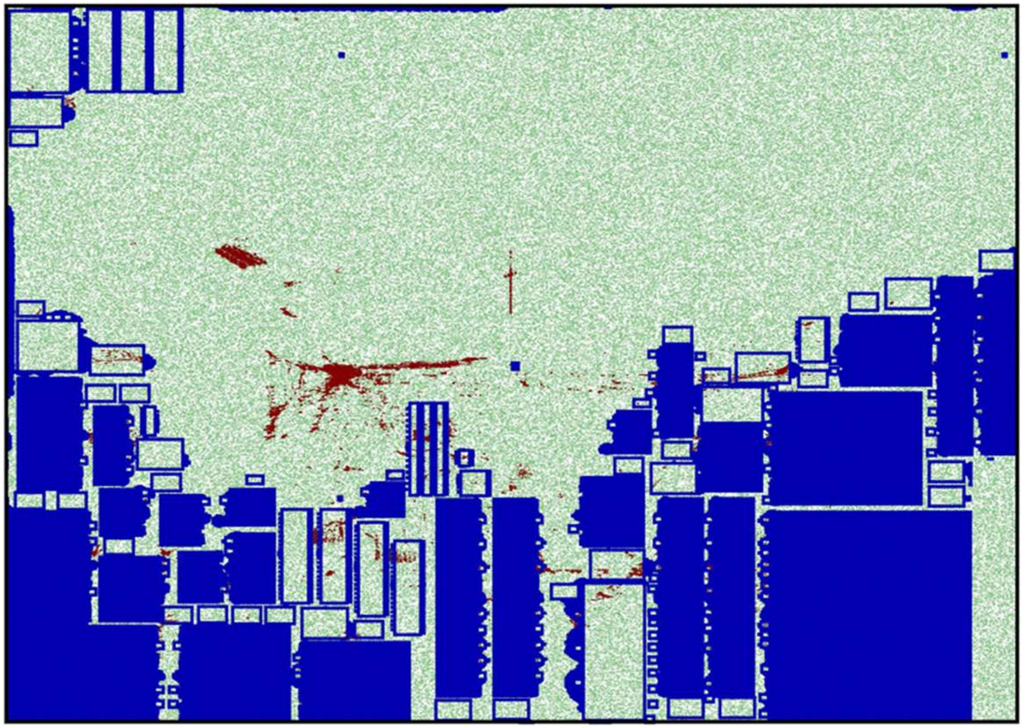
(H1) RC=211.73 (H2) RC=138.50 (H3) RC=109.05 (H5) RC=104.60 (H7) RC=101.95 (Hf) RC=102.43

# RePlAce:  Routability-Driven Placement

- Global placement animation with our routability optimization
  - Keeps the inflated cell size by the end of global placement
  - Red-cell zone is keep growing along with the routability optimization



SUPERBLUE12 (DAC-2012)
The initial RC value = 211.73
The final RC value = 102.43



SUPERBLUE18 (ICCAD-2012)
The initial RC value = 135.23
The final RC value = 102.10

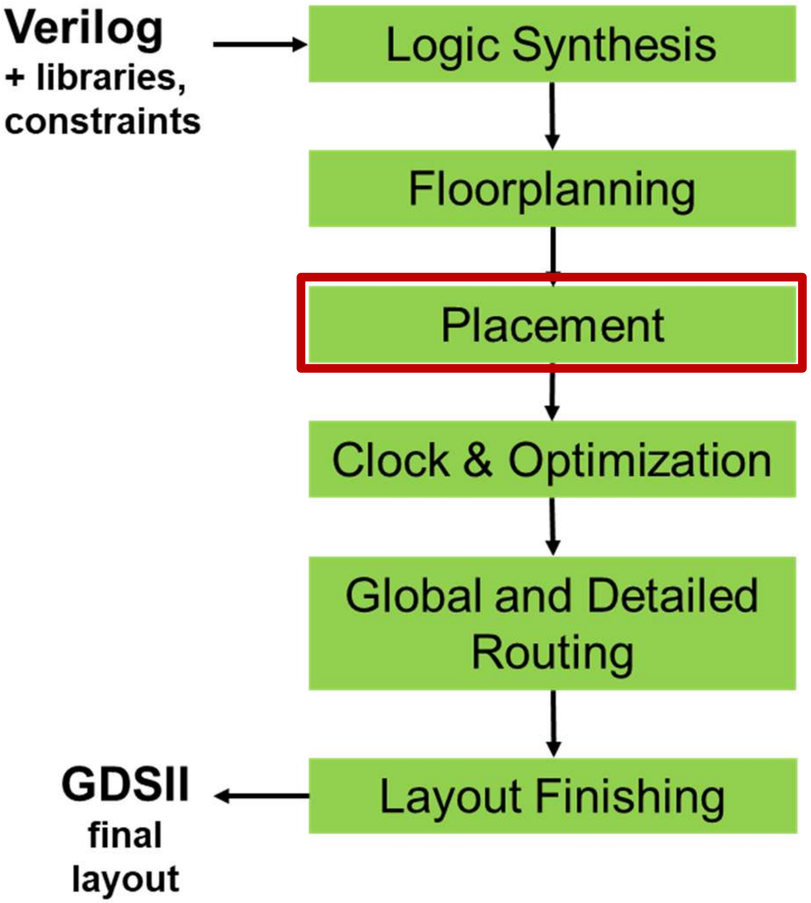
# RePlAce (2019) Summary

- Advancing solution quality and routability validation in GPL
  - Local density function
    - Local smoothing comprehending local over-demanded bins
  - Improved dynamic step size adaptation
  - Routability optimization
    - Simple but effective metal layer-aware superlinear cell inflation technique
- Superior solution quality for range of problem types
  - Standard cell placement: average HPWL reduction of 2.00% over best previous ISPD benchmark results
  - Mixed-size placement: average HPWL reduction of 2.73% over the best previous MMS benchmark results
  - Routability-driven placement: average 8.50% to 9.59% scaled HPWL reduction over previous leading academic placers for DAC-2012 and ICCAD-2012 benchmark suites

**Notes**

- Formulation is here (calculate demand / capacity) and here
- Dynamic step size updating is here
- Entry point for routability-driven is here
- Calculation of cell inflation for routability-driven placement is here and here

# Global Placement Zoom-In



**Verilog**
**+ libraries, constraints**

Logic Synthesis

Floorplanning

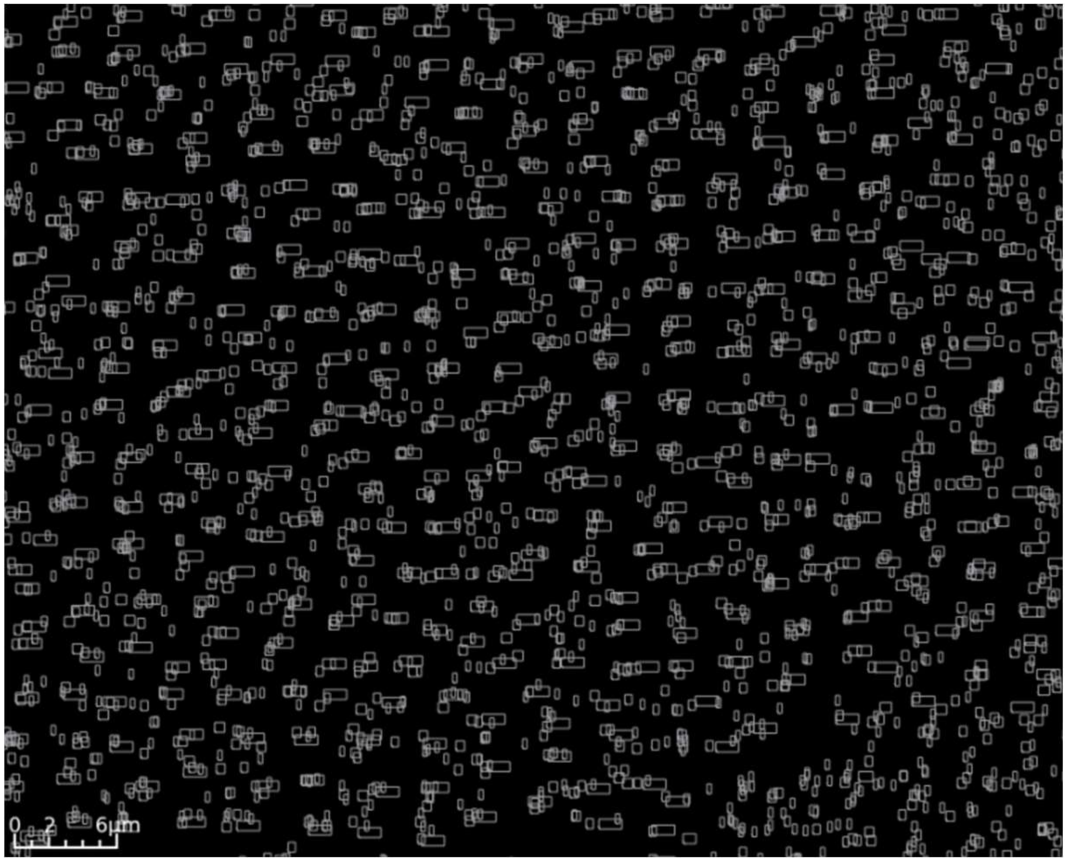
Placement

Clock & Optimization

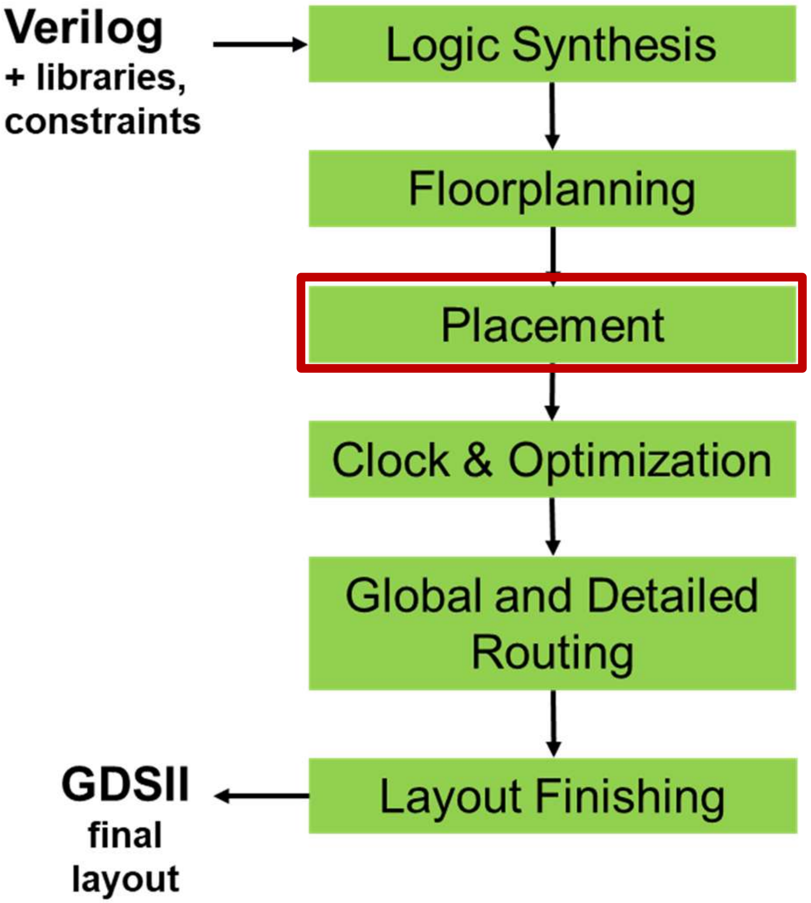
Global and Detailed Routing

Layout Finishing

**GDSII**
**final layout**



# (Legalized) Detailed Placement Zoom-In



Verilog

+ libraries, constraints

Logic Synthesis

Floorplanning

Placement

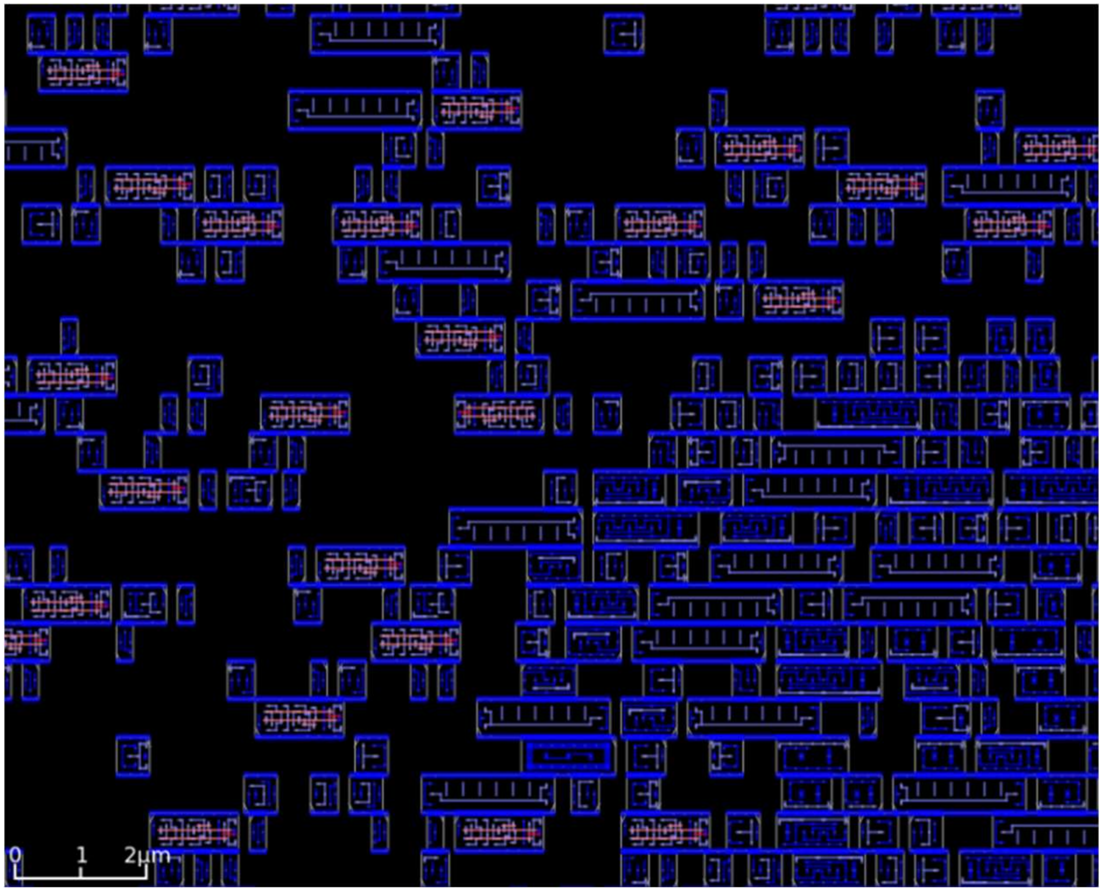
Clock & Optimization

Global and Detailed Routing

Layout Finishing

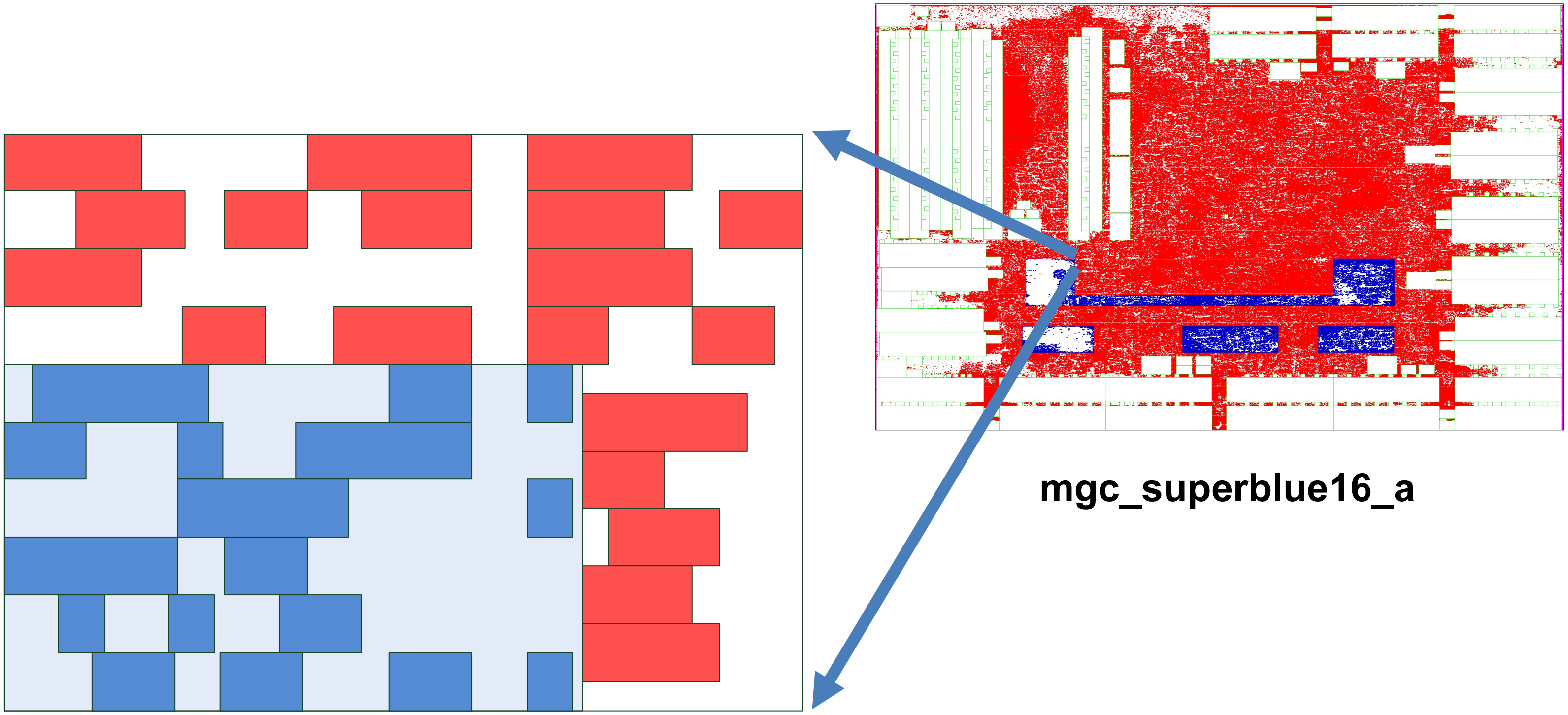
GDSII

final layout



# Region constraints

- **Cells assigned to a region have to be placed inside the boundary of the region.**



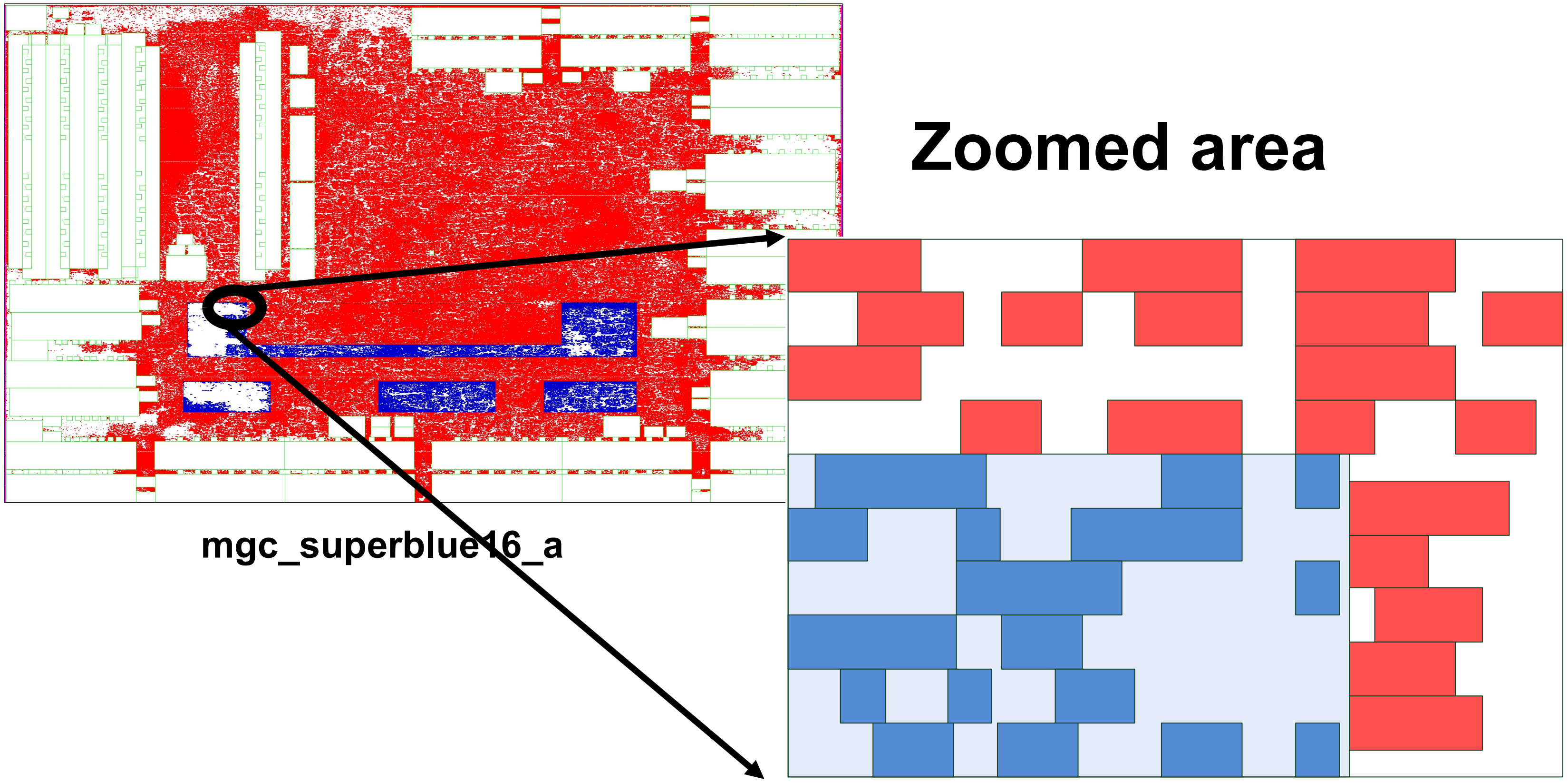
**mgc_superblue16_a**

**Zoomed area containing a region constraint**

Thanks: Nima Darav, AMD

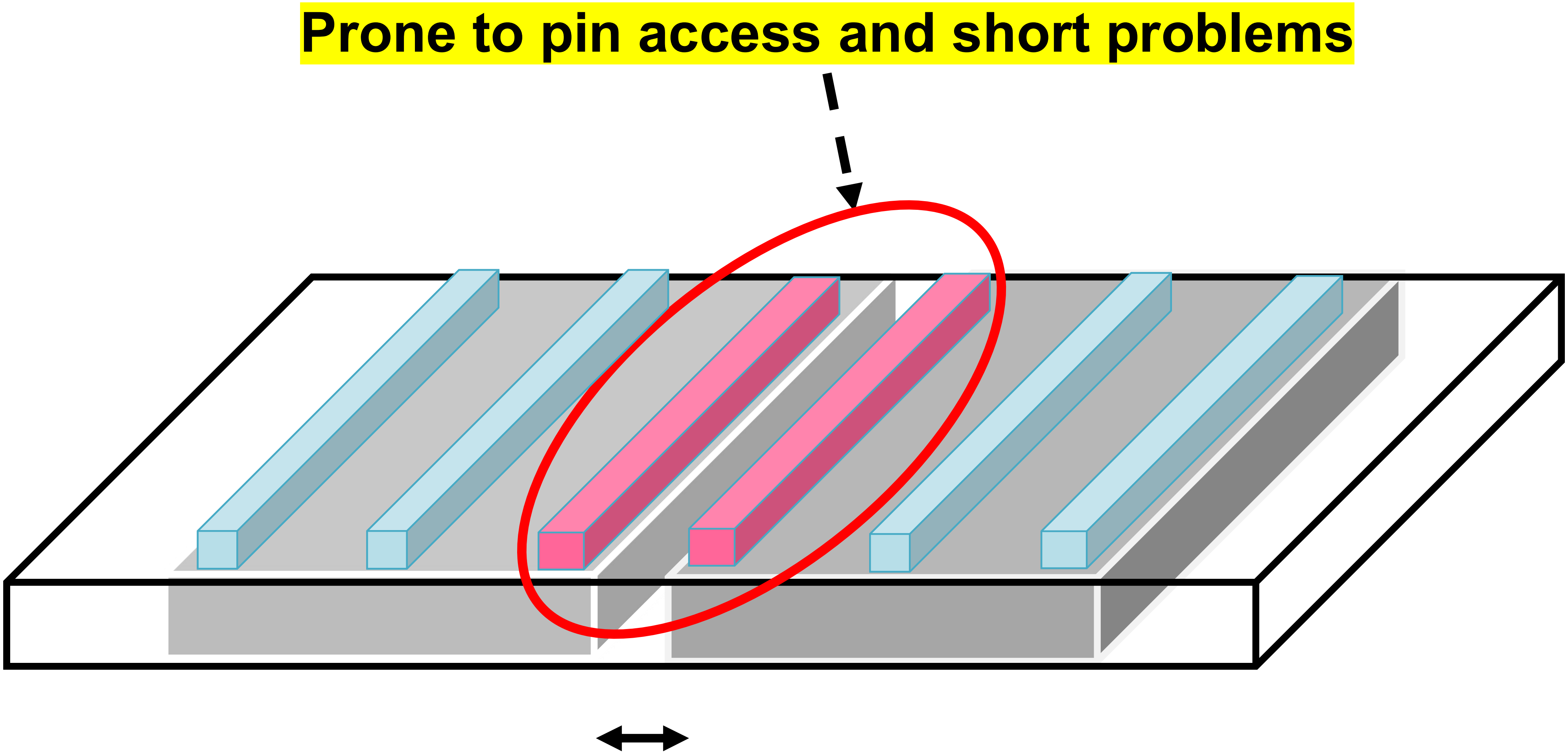
# Fence region constraints

- Cells assigned to a fence region have to be placed inside the boundary of the region while other cells need to be placed outside.



Thanks: Nima Darav, AMD

# Edge spacing constraints



Prone to pin access and short problems

Two cells are too close to each other

Thanks: Nima Darav, AMD

# DPL

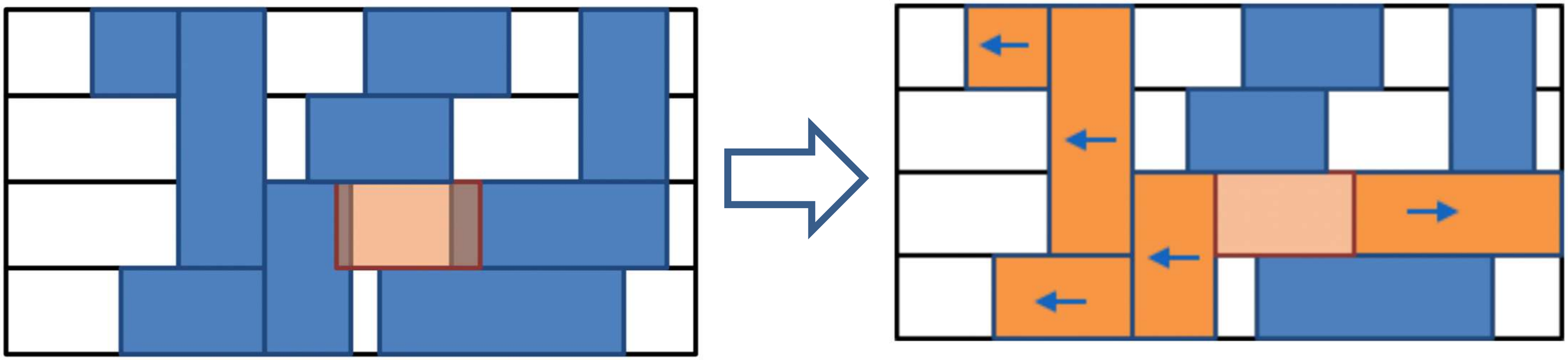
# Legalization and detailed placement

- Global placement must be legalized
  - Cell locations typically do not align with power rails
  - Small cell overlaps due to incremental changes, such as cell resizing or buffer insertion
- Legalization seeks to find legal, non-overlapping placements for all placeable modules
- Legalization can be improved by detailed placement techniques, such as
  - Swapping neighboring cells to reduce wirelength
  - Sliding cells to unused space
- Software implementations of legalization and detailed placement are often **bundled** (dpl)

Thanks: Sang-Gi Do, Samsung

# Mixed-height standard cell legalization

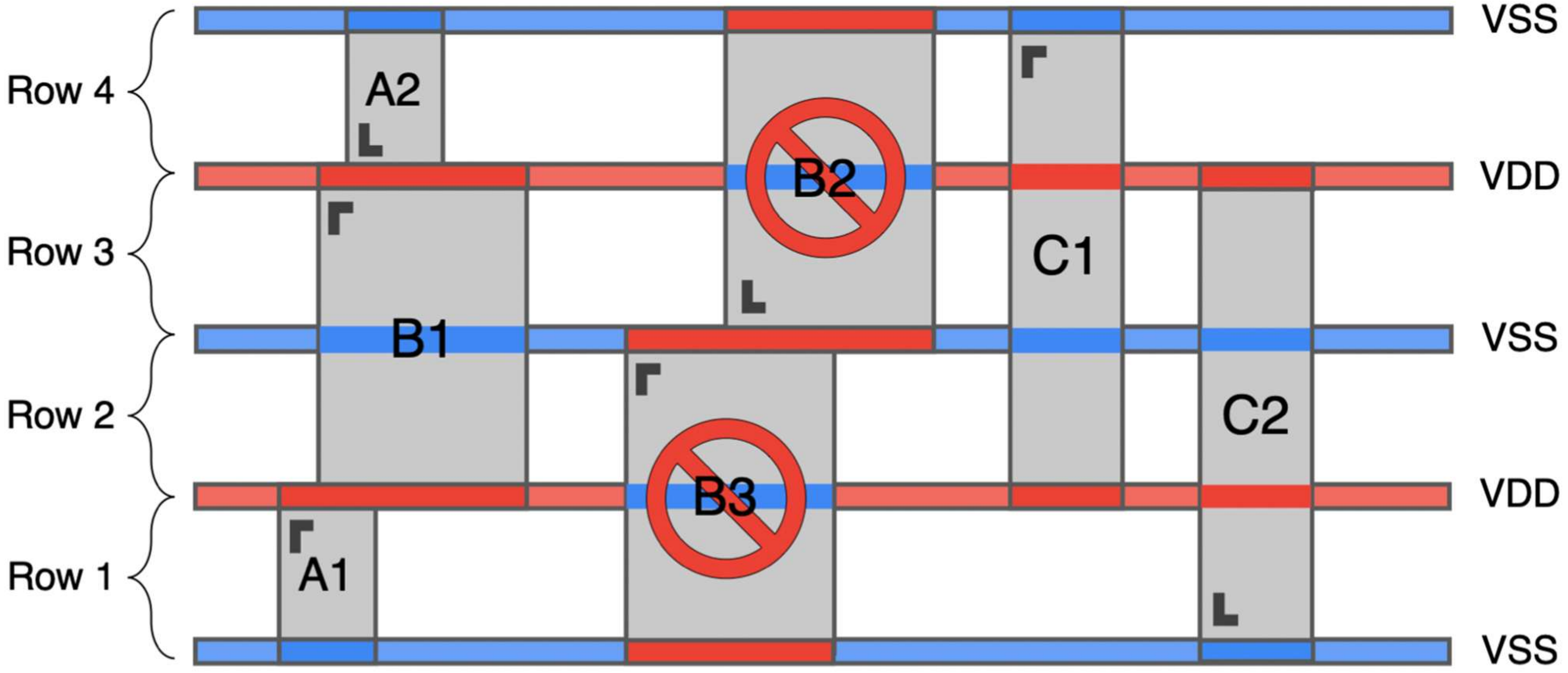
- **Legalization:** seek **legal** placement + remove **overlaps**



- **Mixed-height** standard cells: standard cells that span different row heights ← single/double/triple/quadruple
- **Mixed-height** standard cell legalization is **challenging**
  - Potential overlaps in **vertical** (+ horizontal) directions
  - Horizontal movement of a multi-height cell disrupts placement across all **rows** it overlaps
- Presence of fence regions aggravate the problem

Thanks: Sang-Gi Do, Samsung

# An example of power rails violation

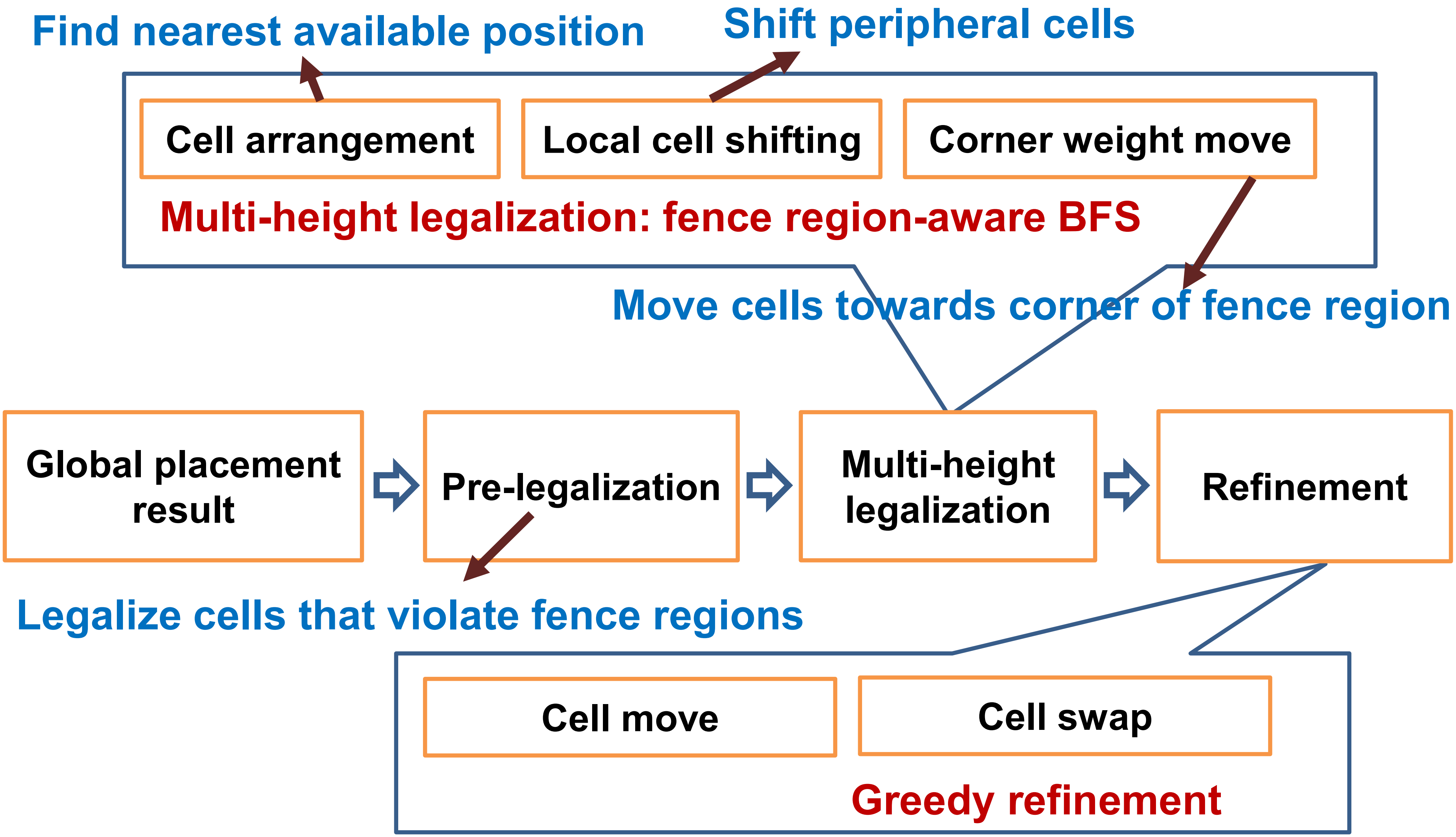


- Red bars: power (VDD); blue bars: ground (VSS)
- A1, A2: single height standard cells; B1, B2: double height standard cells; C1, C2: triple height standard cells
- B2 and B3 **violate power rails alignment**

Thanks: Sang-Gi Do, Samsung

# Overall framework in dpl

- Fence region-aware (FRA) BFS algorithm (paper)



Find nearest available position

Shift peripheral cells

Cell arrangement

Local cell shifting

Corner weight move

Multi-height legalization: fence region-aware BFS

Move cells towards corner of fence region

Global placement result

Pre-legalization

Multi-height legalization

Refinement

Legalize cells that violate fence regions

Cell move

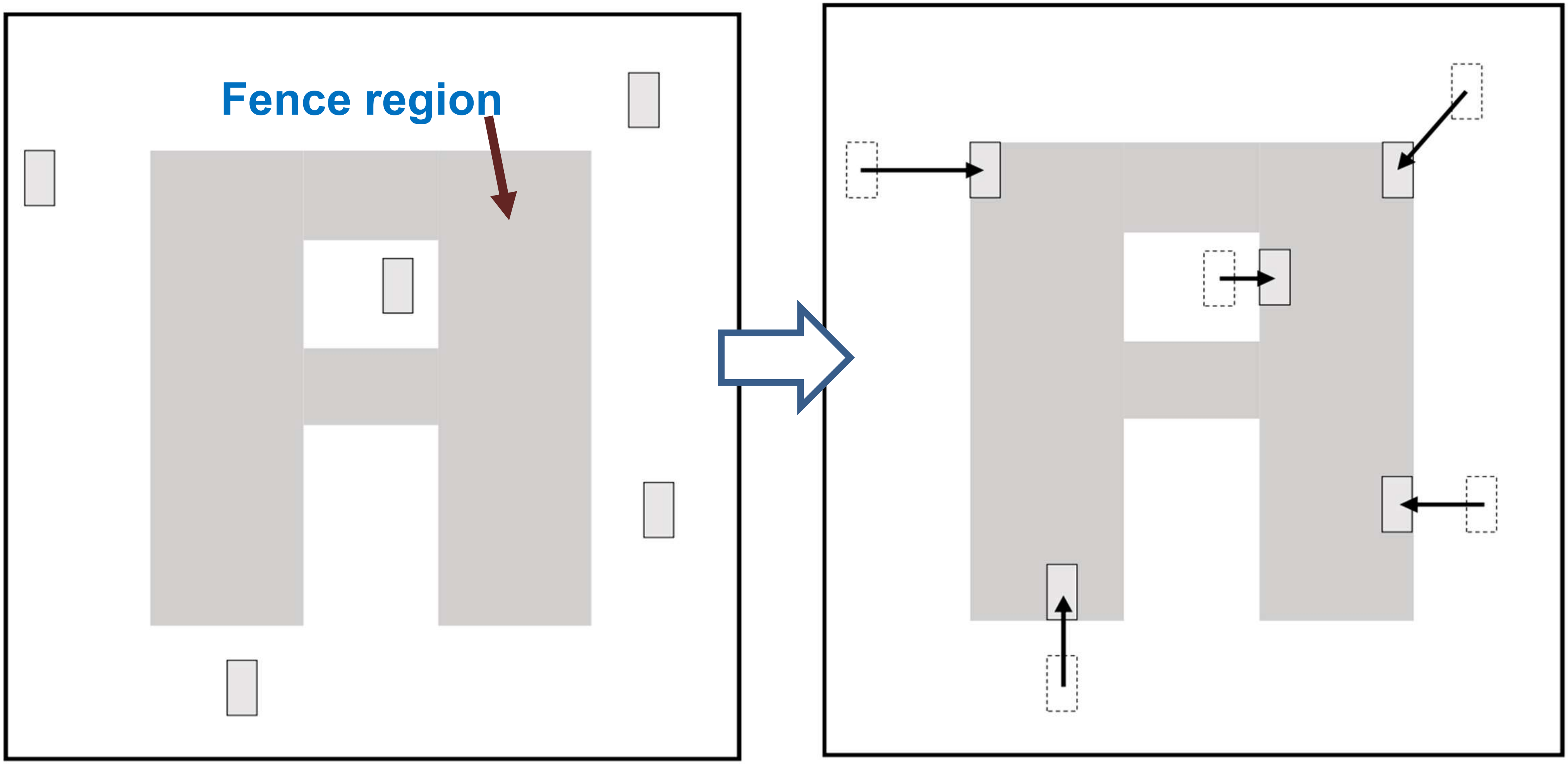
Cell swap

Greedy refinement

Thanks: Sang-Gi Do, Samsung

# Pre-legalization – an example

- Grey cells outside fence region are pushed into the fence region



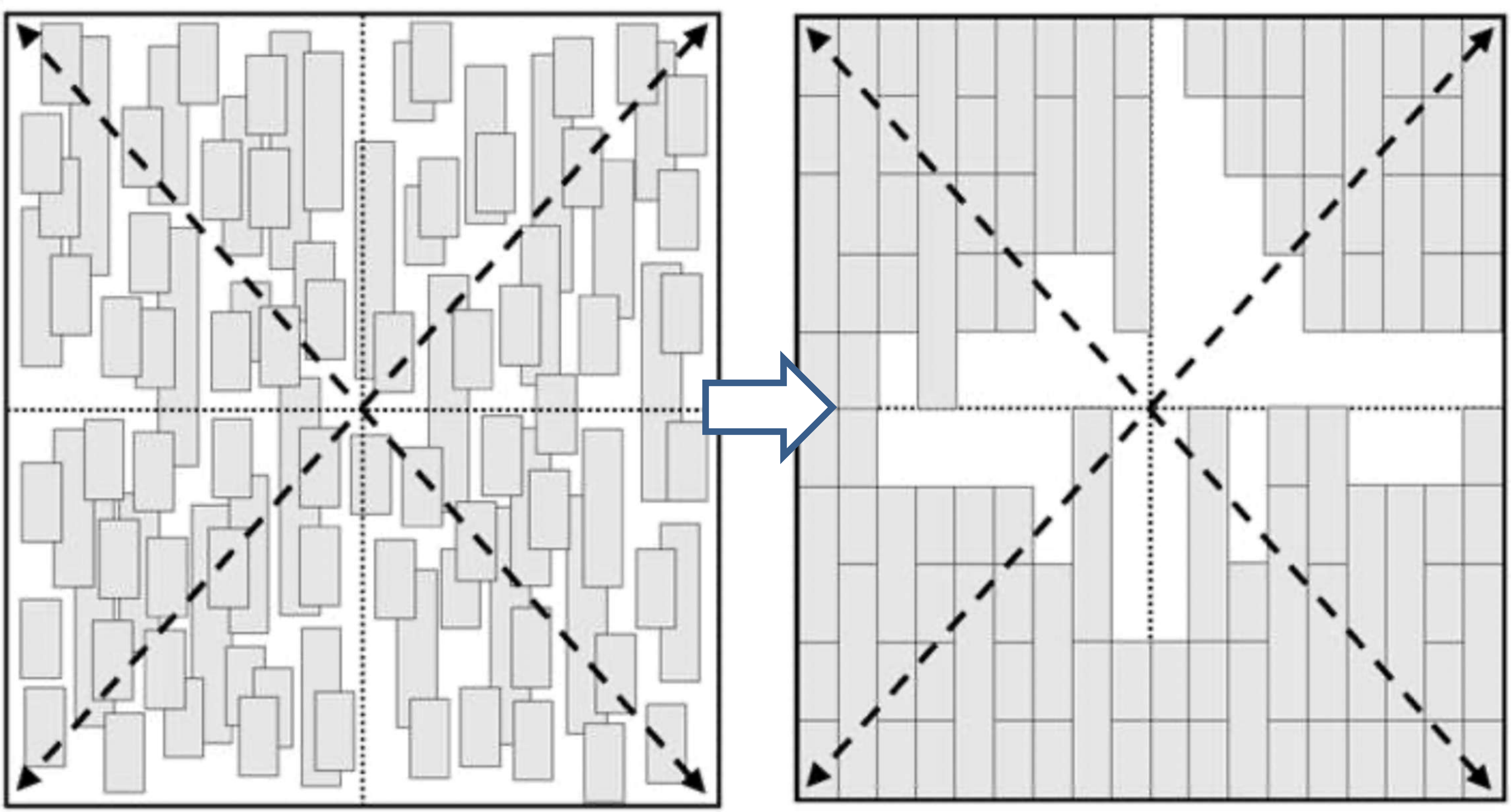
**Before pre-legalization**

**After pre-legalization**

Thanks: Sang-Gi Do, Samsung

# Corner weight move – an example

- Push standard cells towards the corners of the fence region



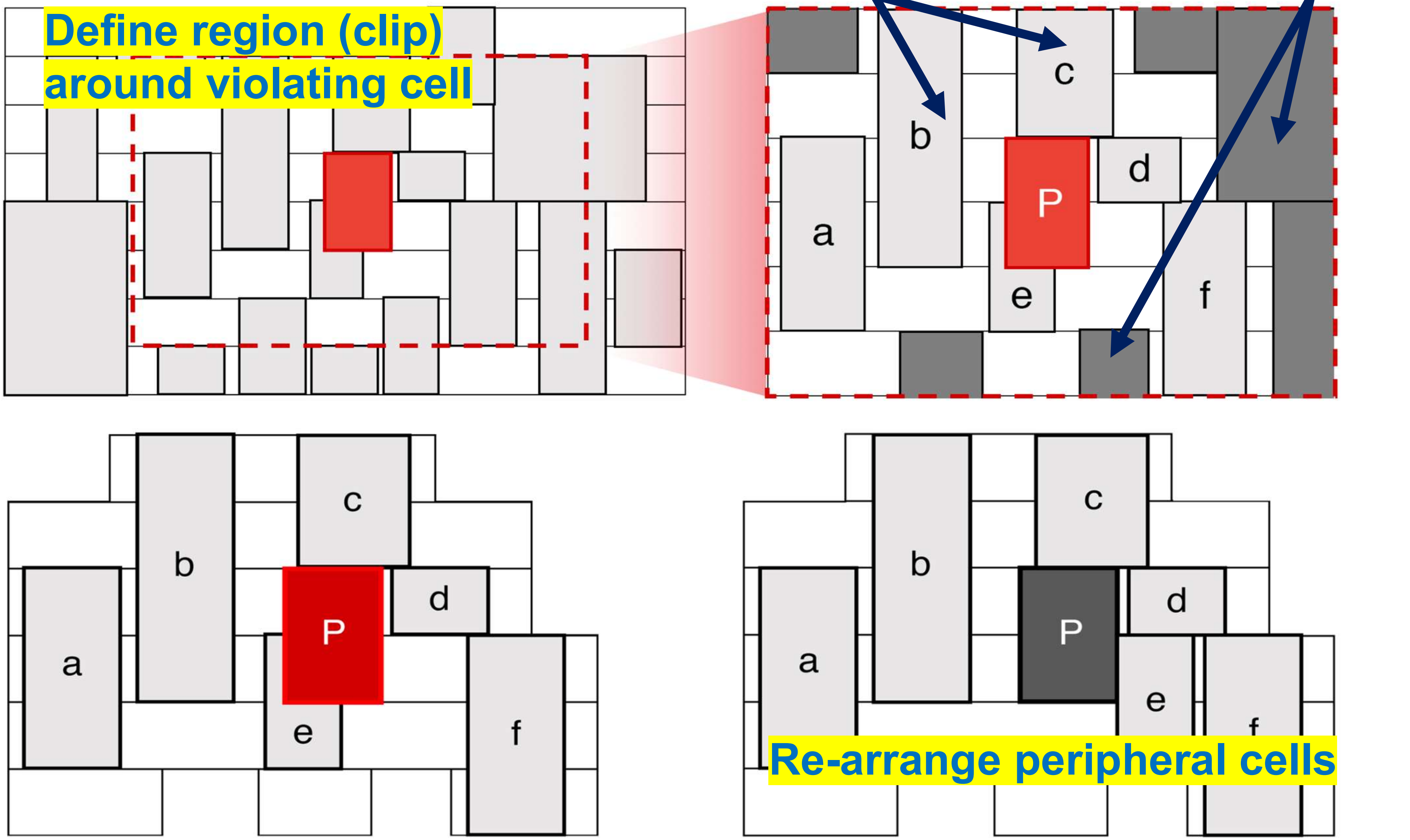
**Before corner weight moves**

**After corner weight moves**

Thanks: Sang-Gi Do, Samsung

# Local shifting – an example

- Set a region around violating cell + rearrange peripheral cells to resolve violation



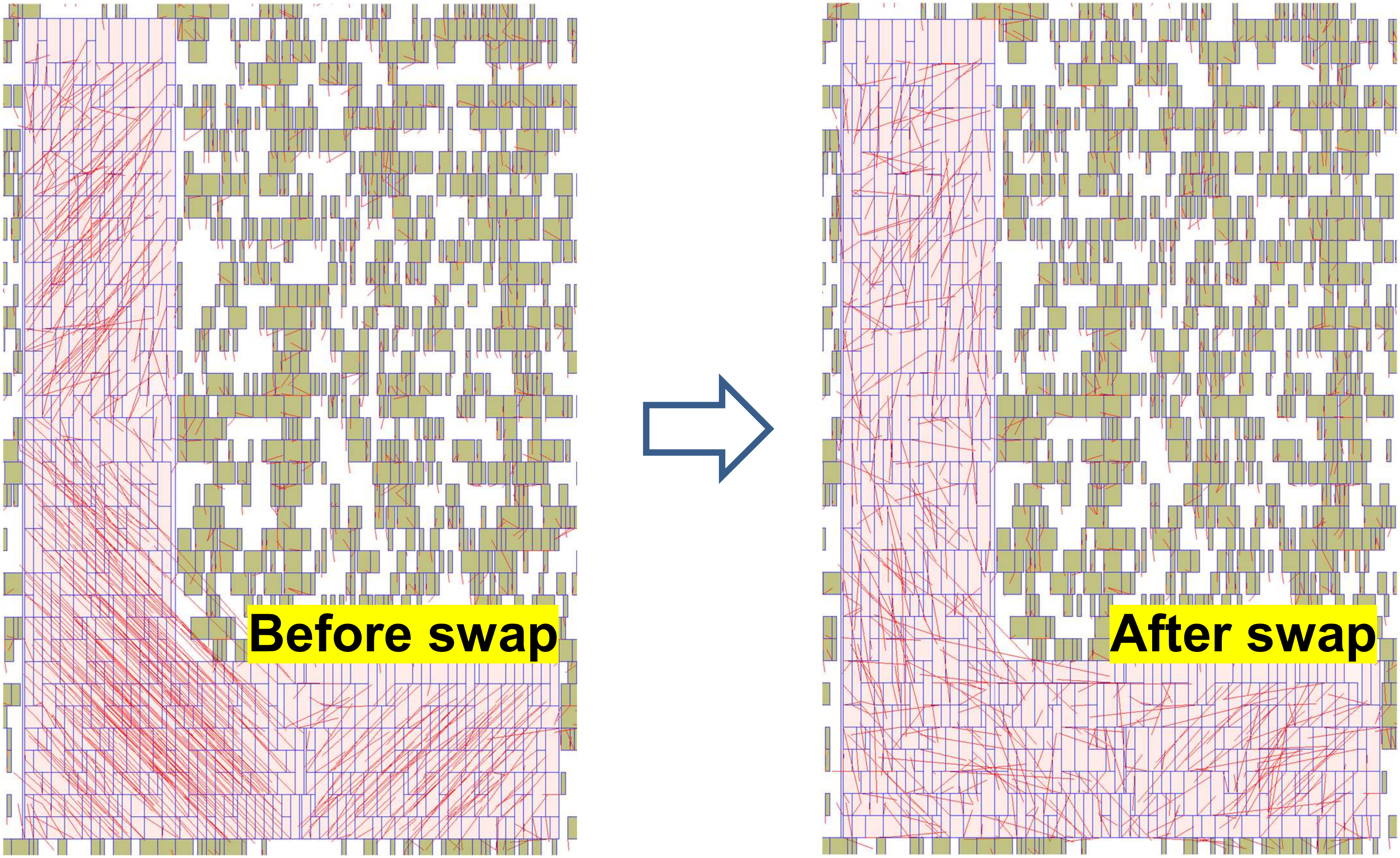
Allowed cells

Disallowed cells

Define region (clip) around violating cell

Re-arrange peripheral cells

# Cell swaps – an example



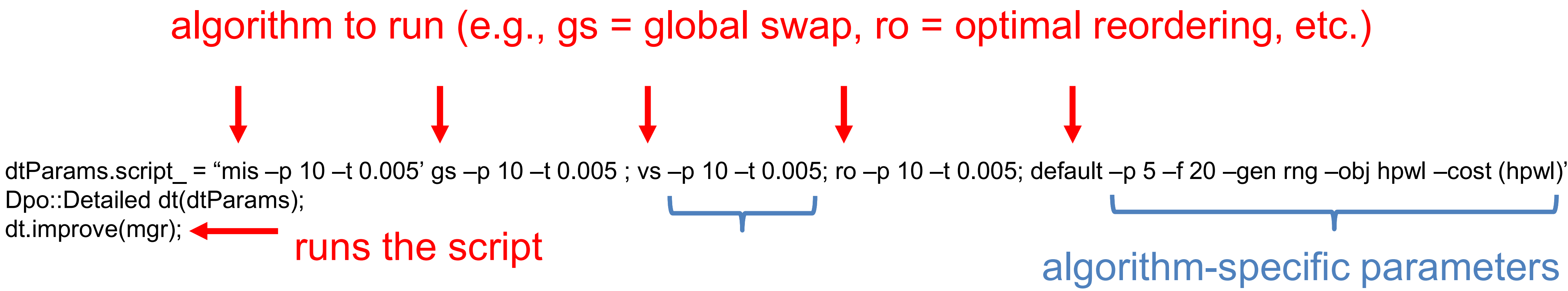
**Benchmark : des_perf_b_md2. Cell count : 112644.**
**Max cell row : 4 Fence Utilization : 96.2%**

Thanks: Sang-Gi Do, Samsung

# DPO

# DPO in a nutshell (1/2)

- DPO runs a detailed placement “script” that can perform a sequence of different optimization algorithms (see Optdp.cpp).
- Implemented algorithms (variations of what’s been described in literature as there are many incarnations of the different algorithms): Maximum independent set matching [1], Optimal reordering [2], Median improvement (global and vertical swapping) [3], greedy improvement [4]
- Sample script string:



algorithm to run (e.g., gs = global swap, ro = optimal reordering, etc.)

```
dtParams.script_ = “mis –p 10 –t 0.005’ gs –p 10 –t 0.005 ; vs –p 10 –t 0.005; ro –p 10 –t 0.005; default –p 5 –f 20 –gen rng –obj hpwl –cost (hpwl)”
Dpo::Detailed dt(dtParams);
dt.improve(mgr);
```

runs the script

algorithm-specific parameters

Thanks: Andrew Kennings, Altera

# DPO in a nutshell (2/2)

- Most algorithms can optimize hpwl or displacement (other objectives such as timing, congestion, etc.) are not well accounted for which is a “consequence” of the algorithm.
- The greedy algorithm is actually very flexible; it uses the idea of “move generators” and “cost objects” that can propose any set of cell movements (cells assigned to new locations) which are then evaluated by the “cost objects” to form a cost function.
  - Proposed moves are accepted or rejected based on cost and move generators and cost objects track the current state of the placement (for incremental computation).
  - Need to be careful to support the required object pieces to add something; e.g., routines need accept() and reject() methods.
- DPO is largely not great with multi-height cells
  - E.g., greedy algorithm can only do simple swaps and moves and is not good at “shifting other cells out of the way”.

Thanks: Andrew Kennings, Altera

# Some References

1. K. Doll, F. M. Johannes, and K. J. Antreich. "Iterative placement improvement by network flow methods", IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems, 13(10):1189-1200, 1994
2. A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal partitioners and end-case placers for standard-cell layout", *Proc. ACM Intl. Symp. on Physical Design*, April 1999, pp. 90-96.
3. Min Pan, Natarajan Viswanathan and Chris Chu, "An efficient and effective detailed placement algorithm", IEEE/ACM International Conference on Computer-Aided Design, pages 48-55, 2005.
4. Andrew A. Kennings, Nima Karimpour Darav, Laleh Behjat "Detailed placement accounting for technology constraints VLSI-SoC 2014:1-6

# BACKUP