



北京大学  
PEKING UNIVERSITY

# A Complete FPGA Placement and Routing Tutorial: Starting from OpenPARF Series

Jing Mai<sup>1</sup>, Jiarui Wang<sup>1</sup>, Zhixiong Di<sup>2</sup>, Guojie Luo<sup>1</sup>, Yun Liang<sup>1</sup>, Yibo Lin<sup>1</sup>

<sup>1</sup>Peking University

<sup>2</sup>Southeast Jiaotong University

[jingmai@pku.edu.cn](mailto:jingmai@pku.edu.cn)

June 10, 2024



1. Introduction
2. Preliminaries
3. OpenPARF 0.1: FPGA Placement considering SLICEL-SLICEM Heterogeneity and Clock Feasibility [Mai+, DAC'22]
4. OpenPARF 0.2: Timing-driven FPGA Placement [Mai+, TCAD'23]
5. OpenPARF 1.0: An Open-Source FPGA Placement and Routing Framework [Mai+, ASICON'23]
6. OpenPARF 3.0: Robust FPGA Placement Considering Cascaded Macros Groups and Fence Regions [Mai+, ISEDA'24]

# Introduction

# Modern FPGA Applications



Communication



Industrial IoT



Data Center



Automotive



Artifical Intelligence



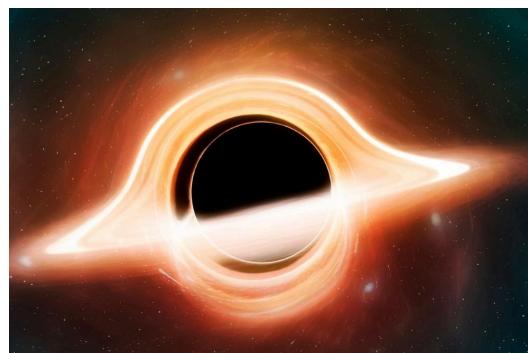
Emulation and Prototyping



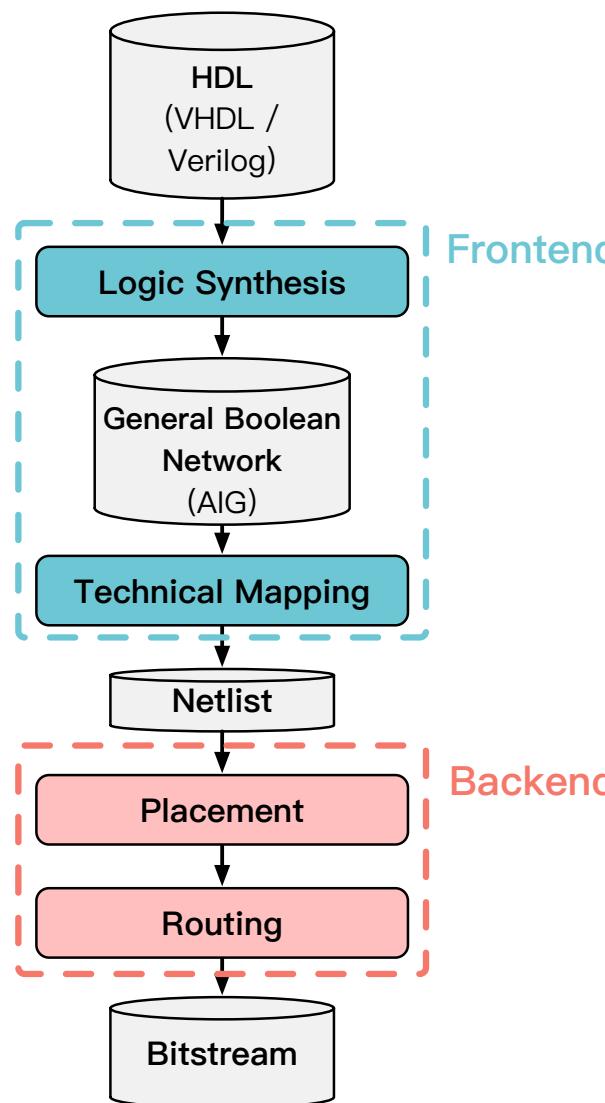
High-Frequency Trading



Cloud Computing



Scientific Computing



## HDL

- ▶ Hardware design is modeled in a *Hardware Description Language* (HDL)

## Frontend

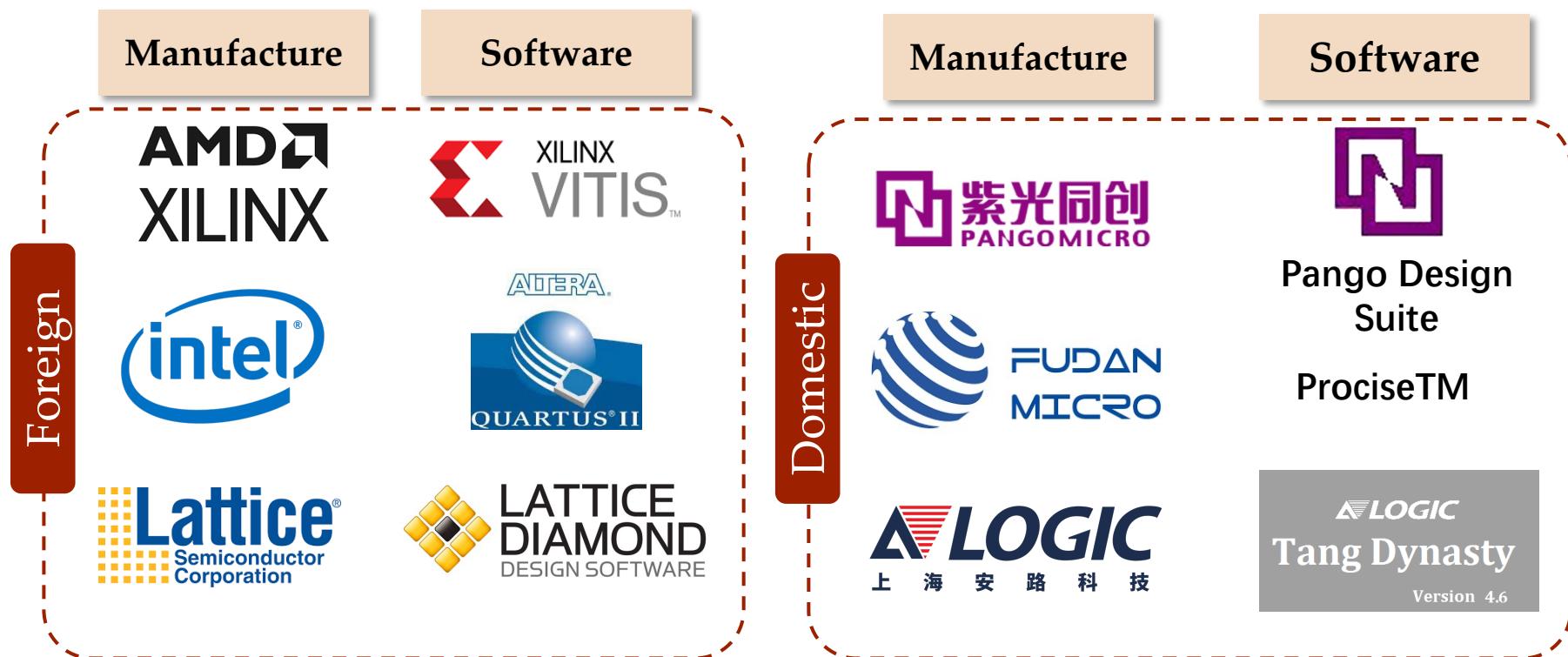
- ▶ A FPGA "compiler" (**synthesis** tool) translates the HDL into a general Boolean network, e.g, *And-Inverter Graph* (AIG)
- ▶ which is then **mapped** to a FPGA technology tailored **netlist**

## Backend

- ▶ The netlist components are **placed** on the FPGA layout
- ▶ and the connecting signals are **routed** through the interconnection network
- ▶ **bitstream** is finally generated for the FPGA configuration

## FPGA Hardware-Software Co-development

- Apart from some front-end processes, other stages require proprietary software tools from FPGA manufacturers.
- The level of software tools determines the scale and performance of the hardware to a great extent!
- **Placement and routing** is the core process of FPGA CAD backend.



# Preliminaries

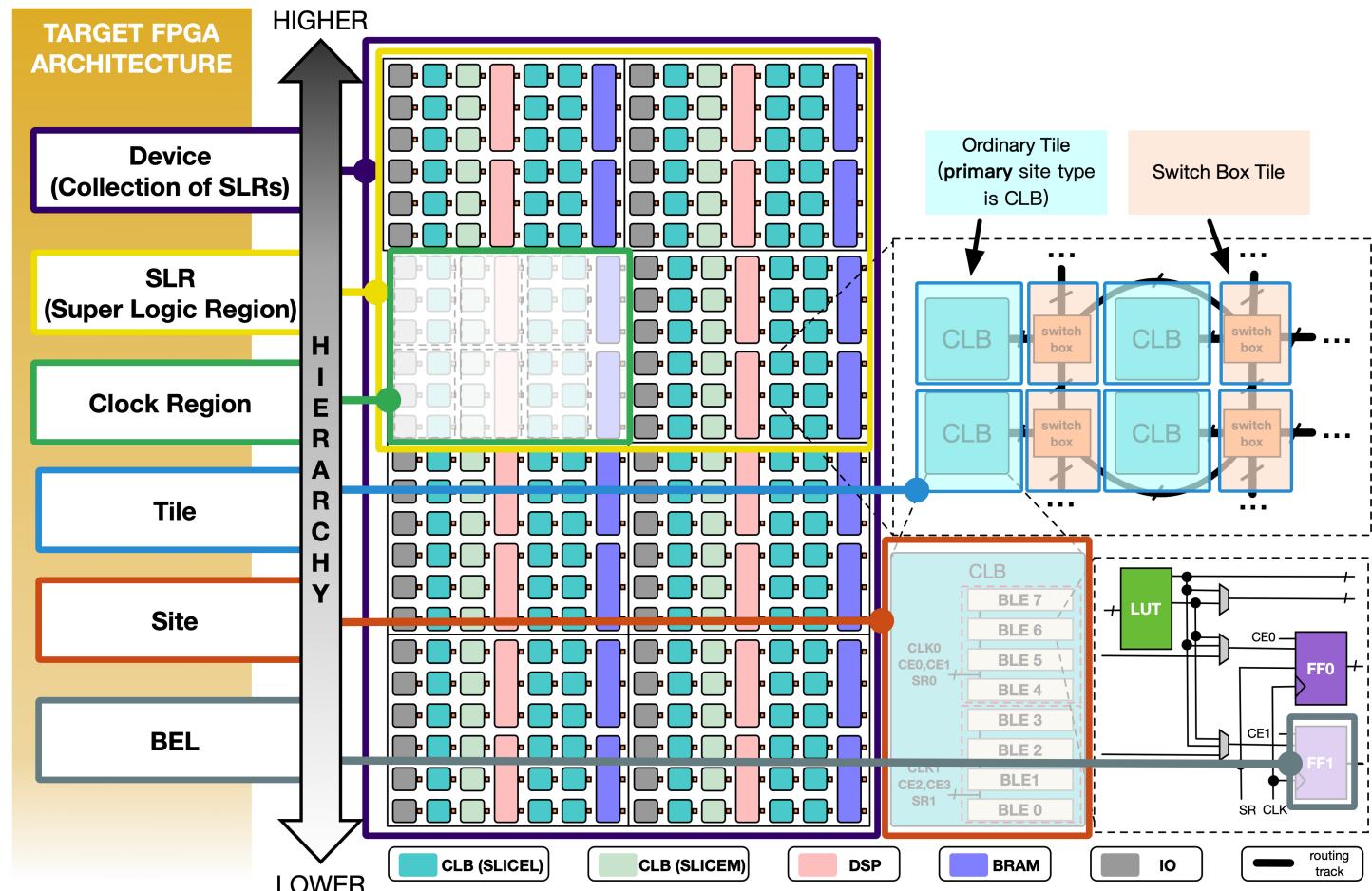
Before proceeding further, we must understand some basic knowledge about FPGA placement and routing:

1. *What is the architecture of modern FPGAs like?*
2. *What are the basic problems involved in FPGA placement and routing?*

# Modern FPGA Architecture

Take Xilinx UltraScale Series as an example...

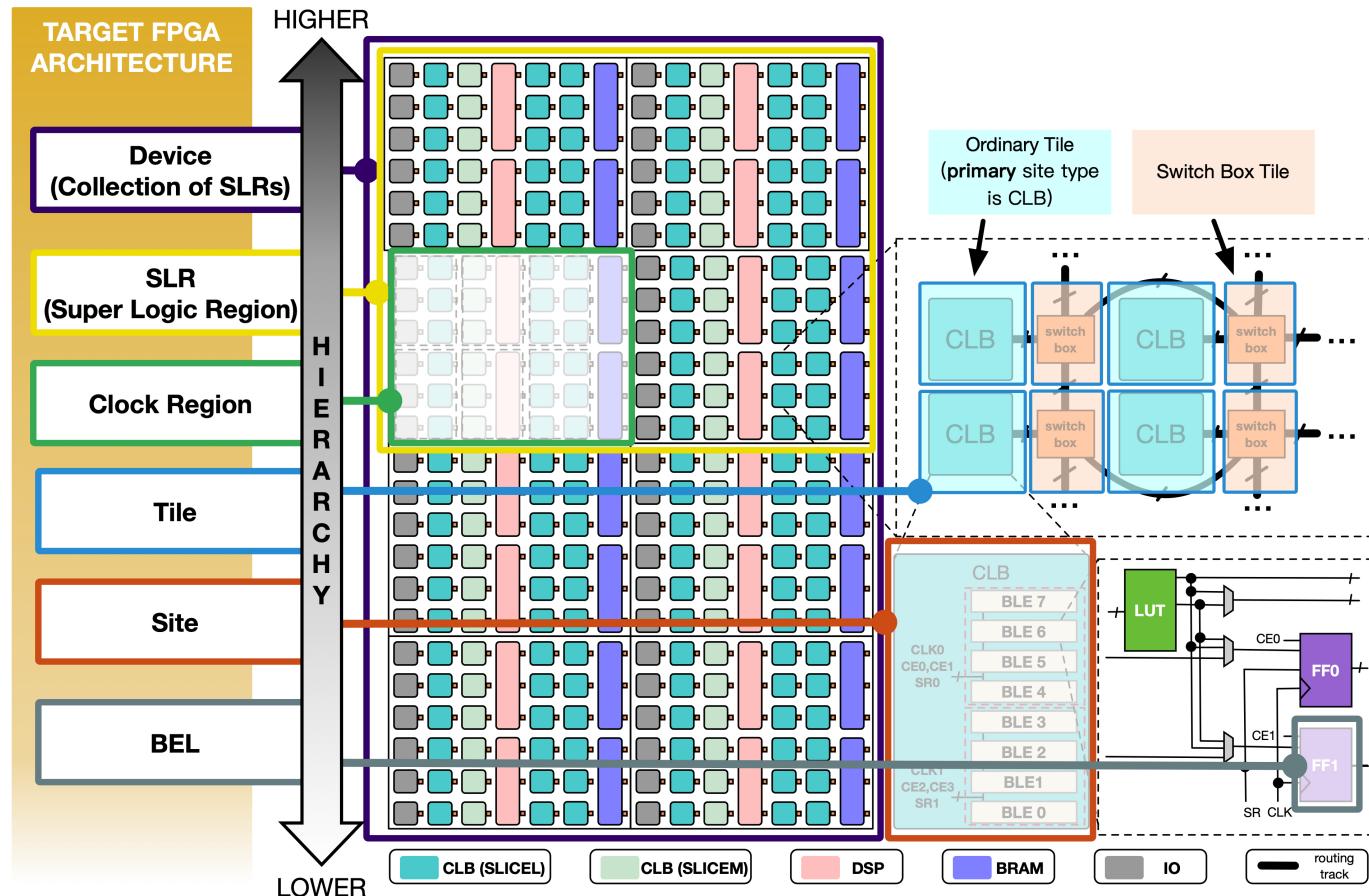
- ▶ Highly heterogeneous: hierarchical architecture, columnar distribution, and heterogeneous resources.
- ▶ Device, SLR, Clock Region, Tile, Site, BEL



# FPGA Hierarchical Architecture (I)

## Device

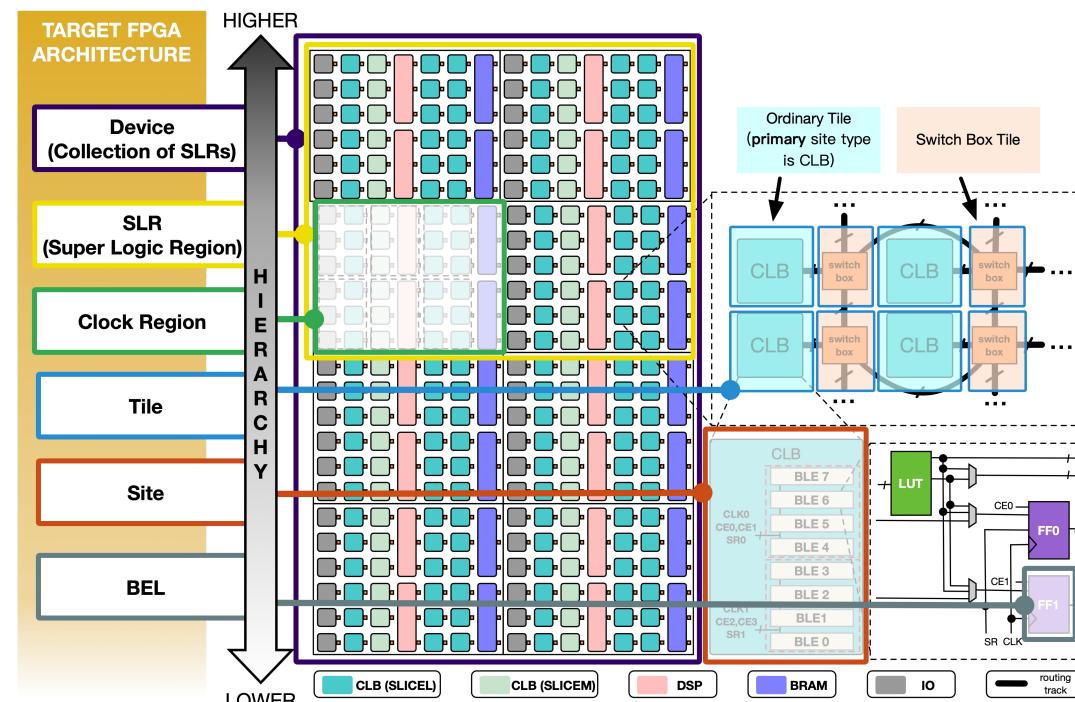
- In Xilinx FPGAs, the device represents the highest level of an FPGA chip, which includes one or more Super Logic Regions (SLRs).
- Multiple SLRs can be assembled through Through-Silicon Via (TSV) stacking technology or 2.5D interposer technology.



# FPGA Hierarchical Architecture (II)

## Super Logic Region (SLR, also known as die)

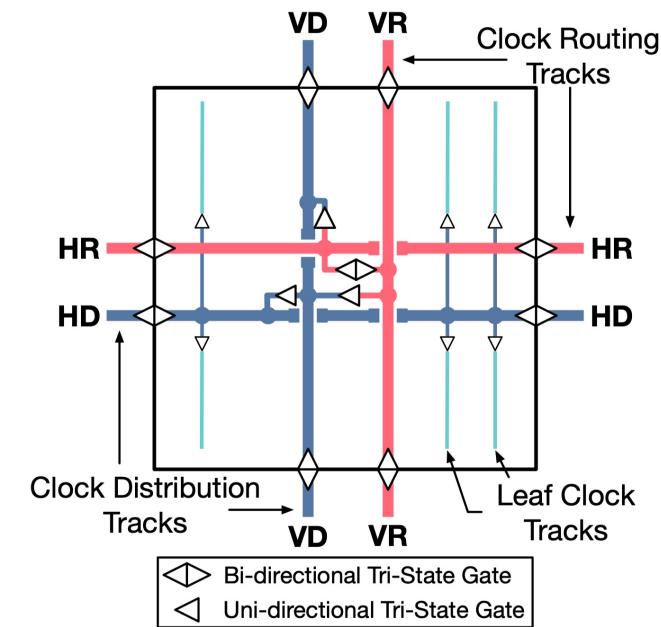
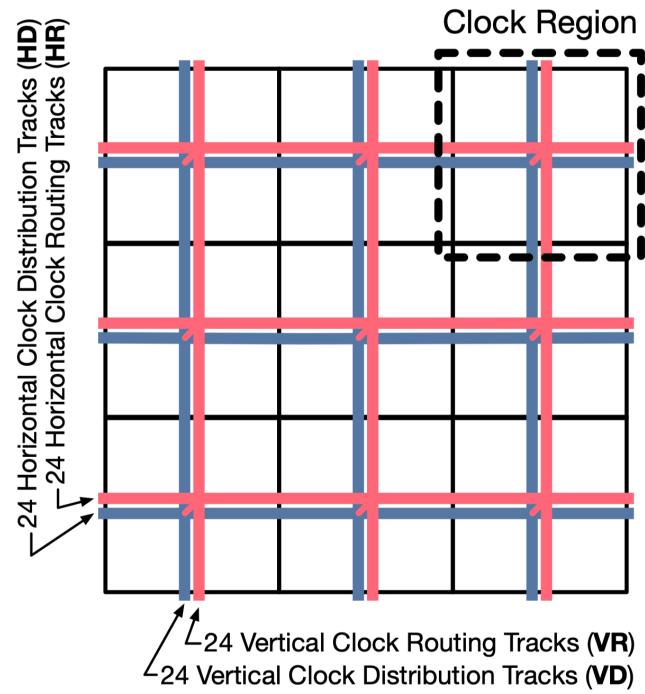
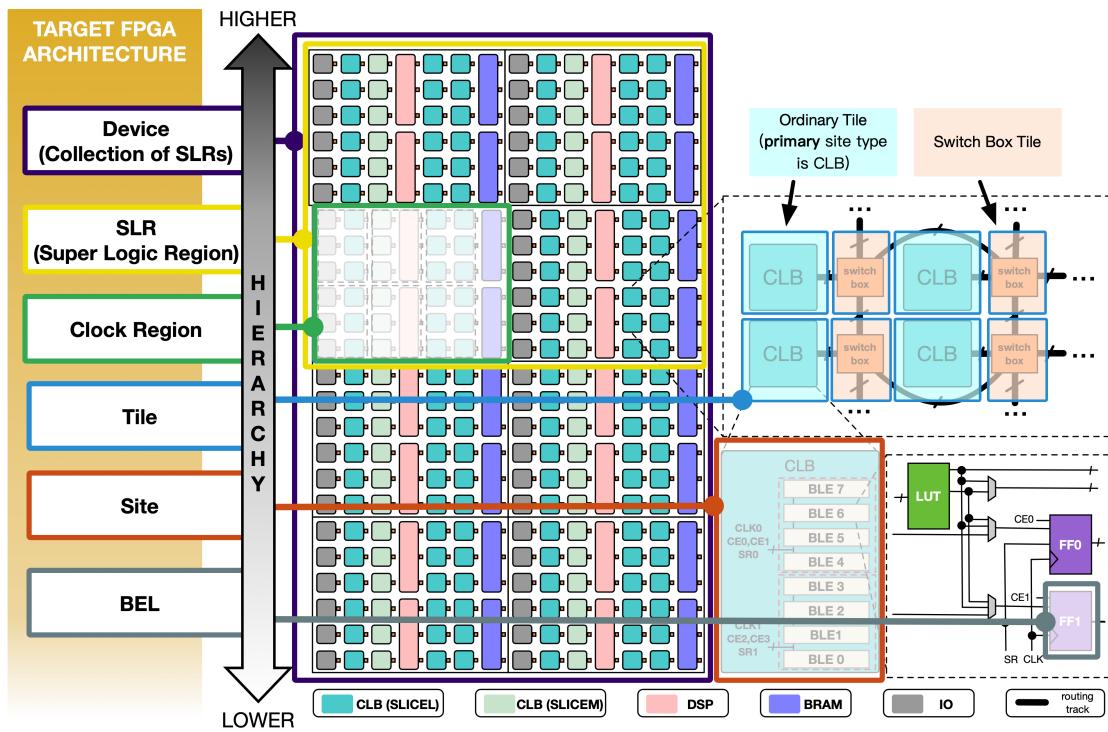
- ▶ SLRs are typically identical because each die is usually manufactured using the same masks.
- ▶ An SLR includes a two-dimensional array of Clock Regions (CRs).
- ▶ To facilitate signal propagation between SLRs, the UltraScale architecture employs special tiles at the edges of the Clock Regions where two SLRs meet.
- ▶ These tiles have dedicated flip-flop positions that help signals cross the SLR boundary. Interconnects that span across the SLR boundaries are known as **Super Long Lines (SLLs)**.
- ▶ Generally speaking, SLLs have a relatively large delay, so it is advisable to avoid signal crossings across SLR boundaries during placement and routing.



# FPGA Hierarchical Architecture (III)

## Clock Region (CR, also known as Fabric Sub Region)

- The clock region is a two-dimensional array of tiles that is related to the allocation of clock routing resources.
- In Xilinx FPGAs, clock signals are propagated to leaf cells along **clock routing tracks** and **clock distribution tracks**, and then **leaf clock tracks**.
- In the Xilinx UltraScale series, a clock region contains 24 horizontal clock routing tracks, 24 vertical clock routing tracks, 24 horizontal clock distribution tracks, and 24 vertical clock distribution tracks. Therefore, the number of different clock signal types within a clock region is at most 24 [Stephen+, ISPD'16] (*clock region constraint*).

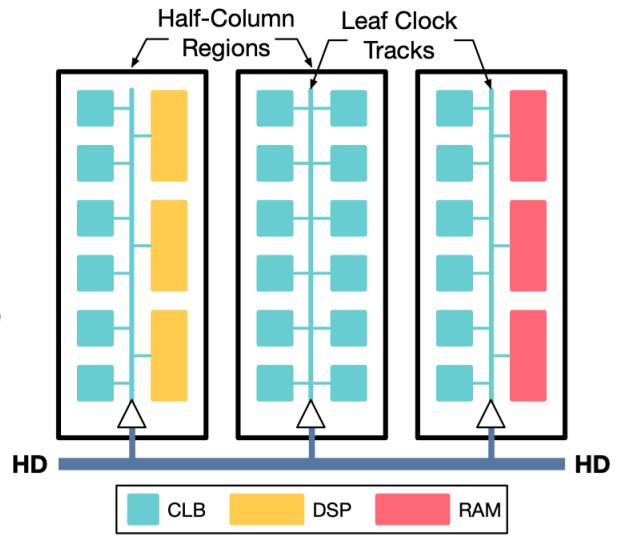
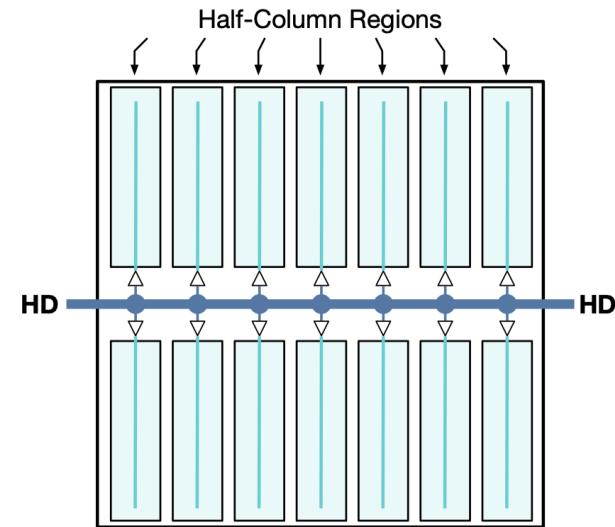
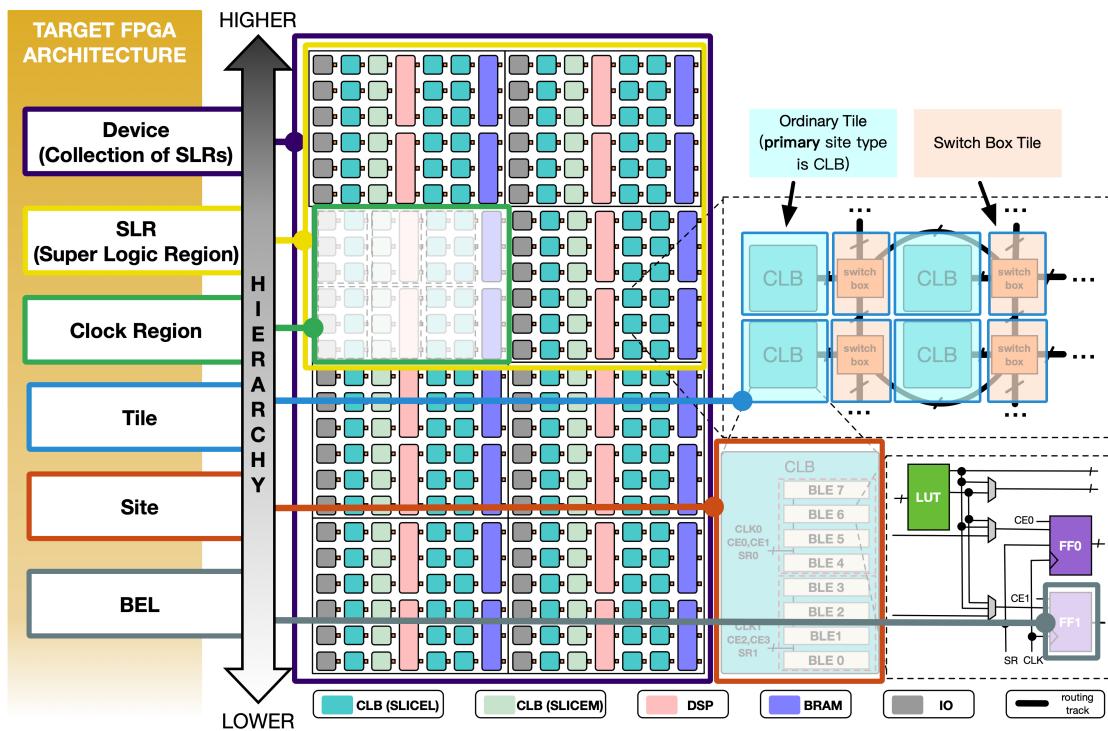


[Li+, TDAES'18]

# FPGA Hierarchical Architecture (III, cont'd)

## Clock Region (CR, also known as Fabric Sub Region)

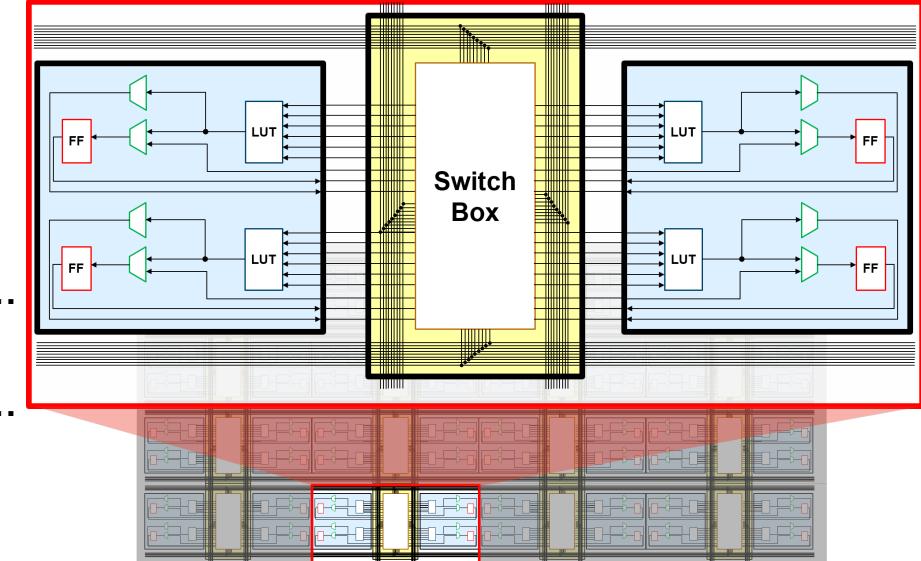
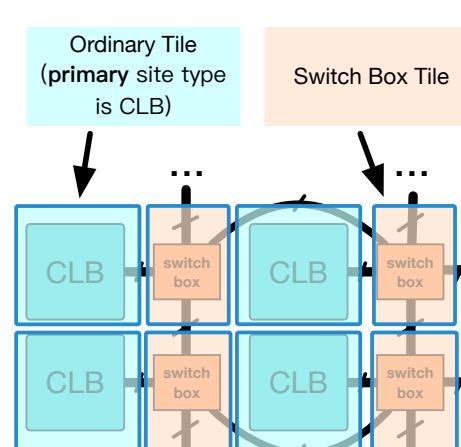
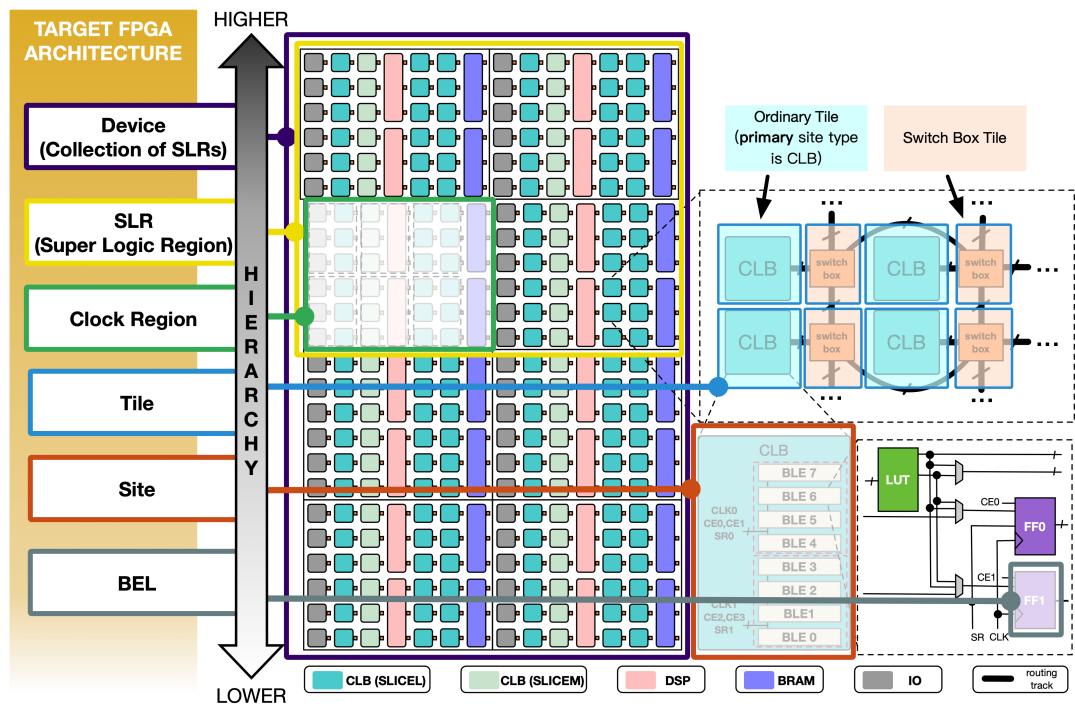
- The horizontal clock distribution tracks pass through the center of the clock region and transmit the signal in both the upper and lower directions to the left clock tracks.
- An area controlled by a left clock track is called a **half-column region**.
- Generally, the width of a half-column region is the width of two sites, and the height is half the height of a clock region.
- In the Xilinx UltraScale series, due to the limited number of left clock tracks, the number of clocks within a single half-column region does not exceed 12 [Stephen+, ISPD'16] (*half-column region constraint*).



# FPGA Hierarchical Architecture (IV)

## Tile

- ▶ Tile is a physical concept of a rectangular space on the FPGA layout, the sizes of different types of tiles can vary.
- ▶ Inside a tile, there may be one site, multiple sites, or no sites at all.
- ▶ Specially, there is a special type of tile called a **switch box tile** which is composed of programmable interconnect points (PIPs) for routing.
- ▶ Its connections include interconnections between sites and switch boxes, as well as between different switch boxes.

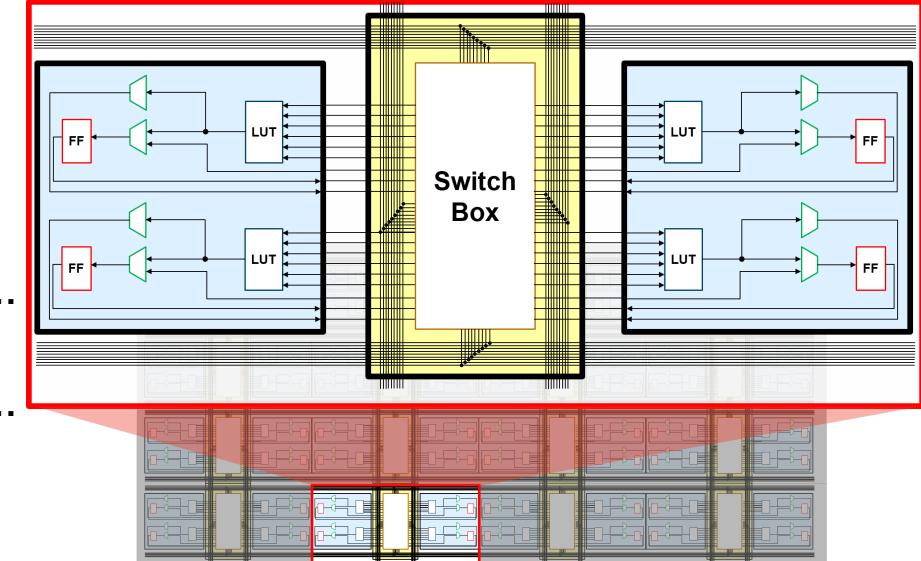
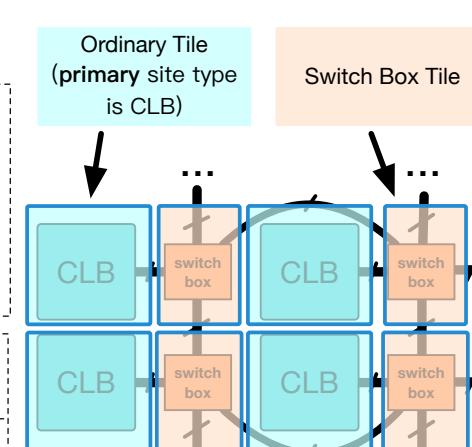
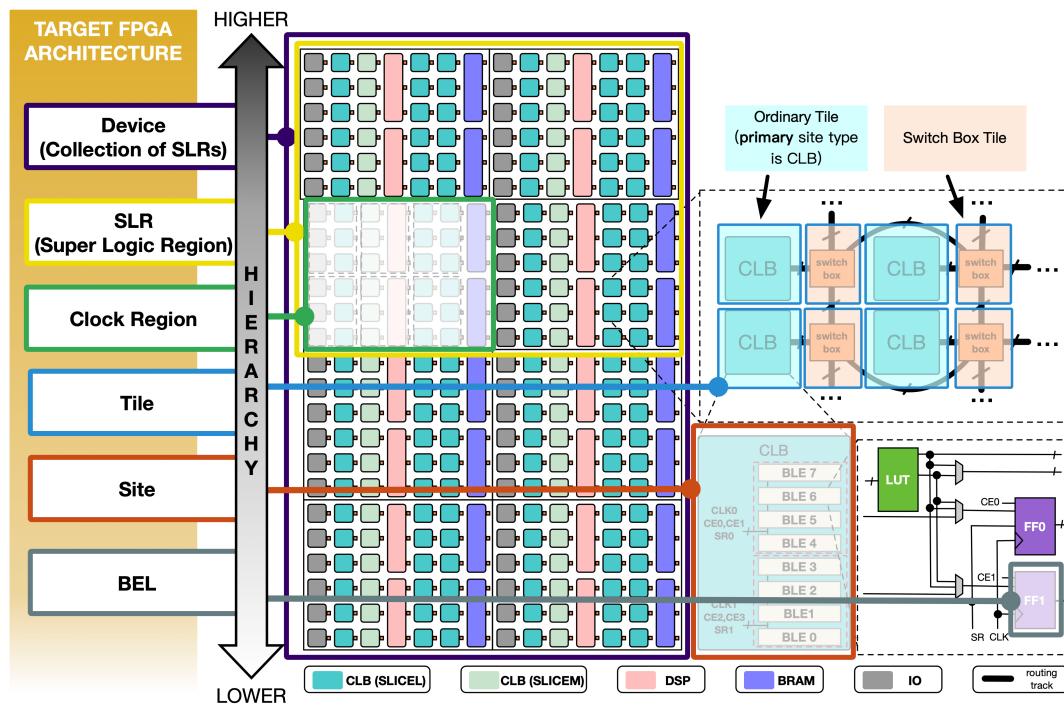


Close up view of replicated tiles of the logic array and interconnect  
[RapidWright, 2023]

# FPGA Hierarchical Architecture (IV, cont'd)

## Tile

- During routing, sites such as CLBs, DSPs, and BRAMs first connect to the adjacent switch box, and then connect to another site through horizontal and vertical interconnect tracks (**route channel**).
- These interconnect tracks can not only connect the switch boxes of adjacent tiles but also make cross-tile connections (interconnect tracks that make cross-tile connections are called **multiplexers**).
- Xilinx uses a traditional columnar approach for tile layout, which means that, with a few exceptions, all tiles in a column are of the same type, but tiles occupying the same row are usually of different types.

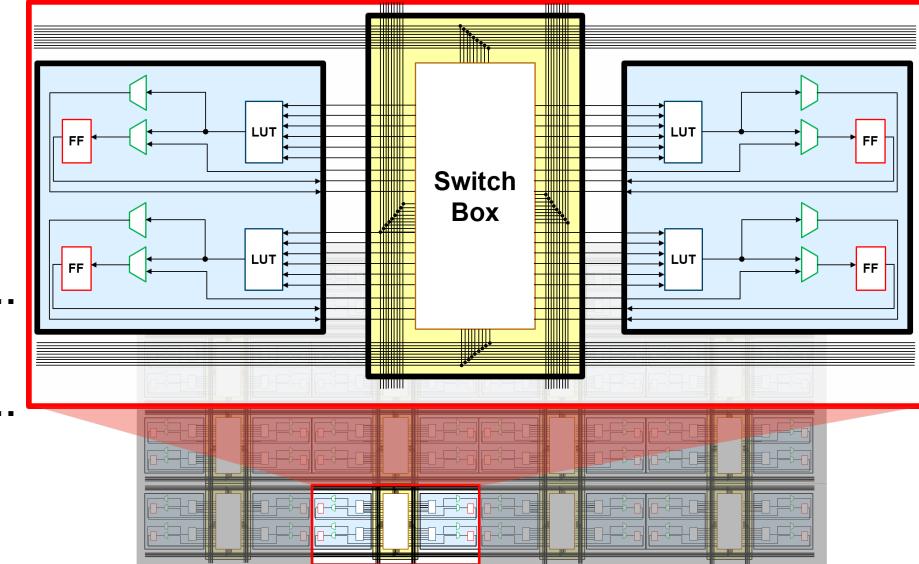
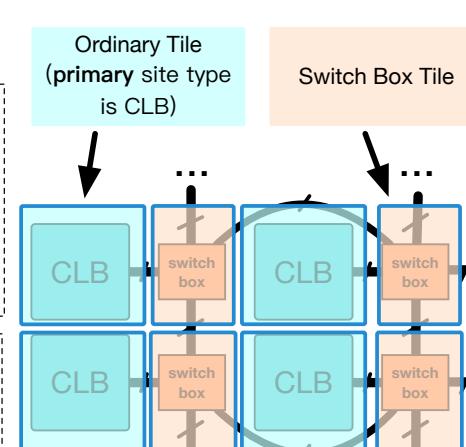
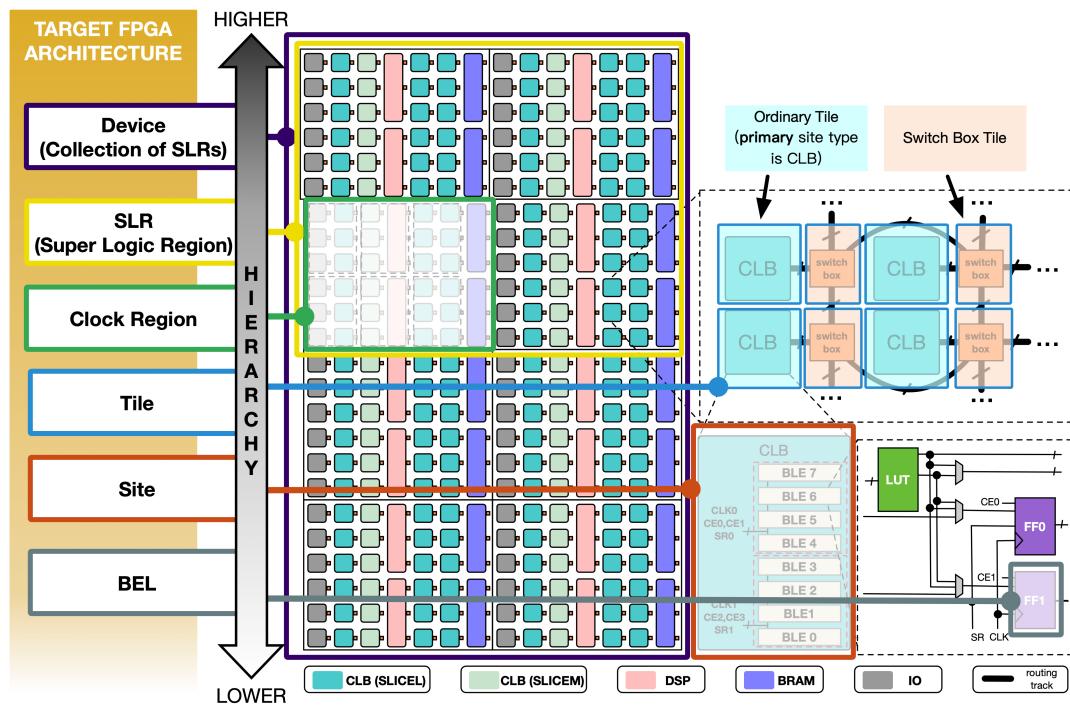


Close up view of replicated tiles of the logic array and interconnect  
[RapidWright, 2023]

# FPGA Hierarchical Architecture (V)

## Site

- During routing, sites such as CLBs, DSPs, and BRAMs first connect to the adjacent switch box, and then connect to another site through horizontal and vertical interconnect tracks (**route channel**).
- These interconnect tracks can not only connect the switch boxes of adjacent tiles but also make cross-tile connections (interconnect tracks that make cross-tile connections are called **multiplexers**).
- Xilinx uses a traditional columnar approach for tile layout, which means that, with a few exceptions, all tiles in a column are of the same type, but tiles occupying the same row are usually of different types.

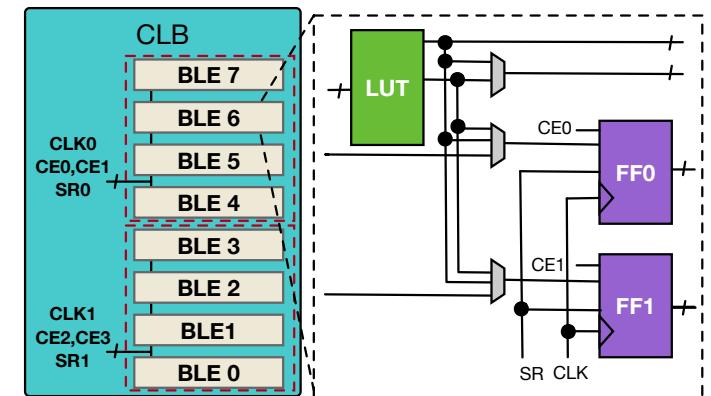
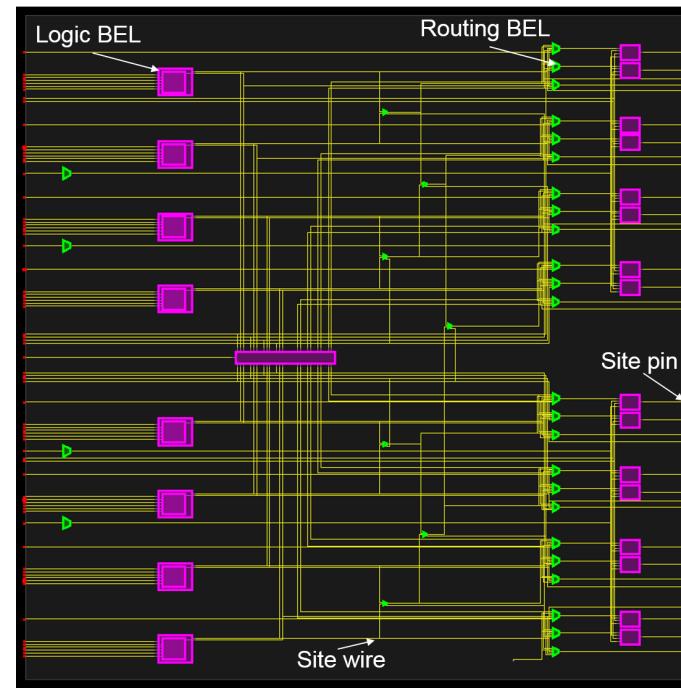
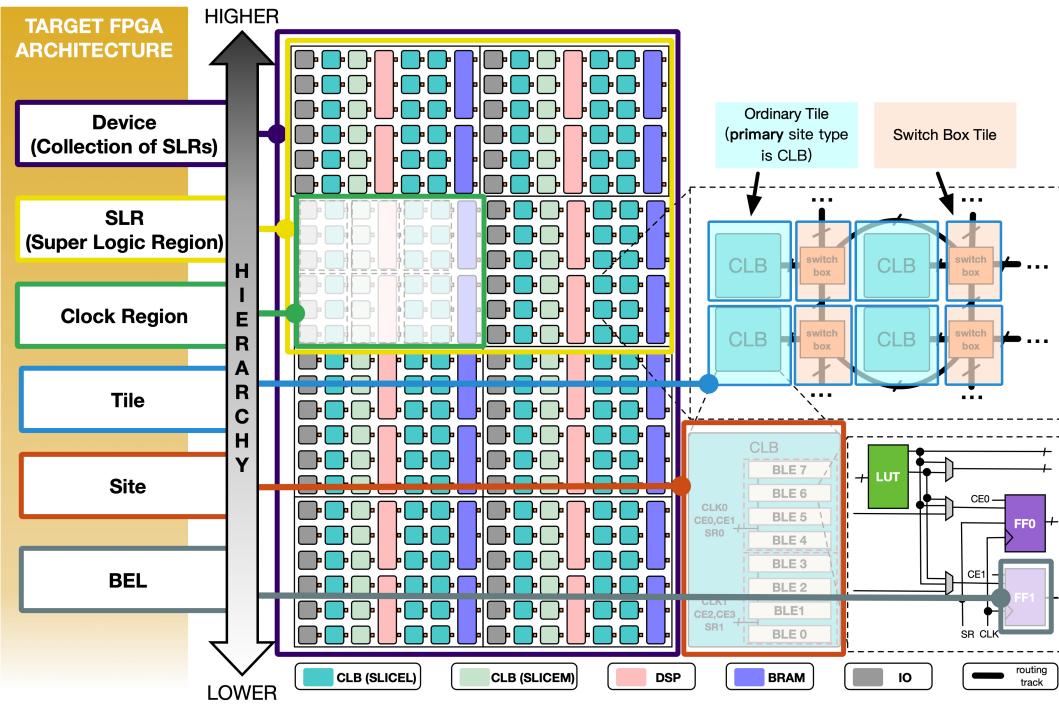


Close up view of replicated tiles of the logic array and interconnect  
[RapidWright, 2023]

# FPGA Hierarchical Architecture (V)

## Site

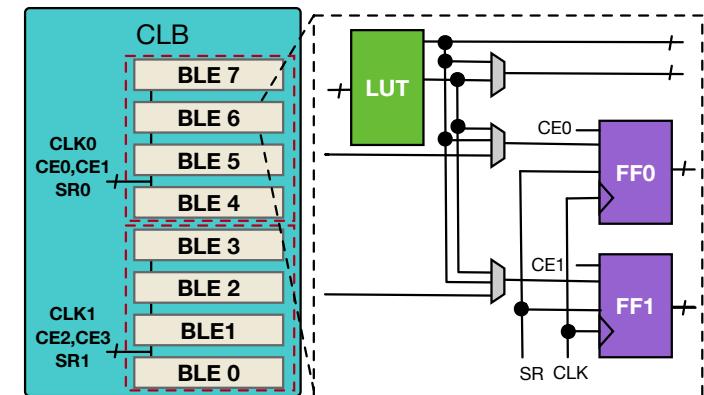
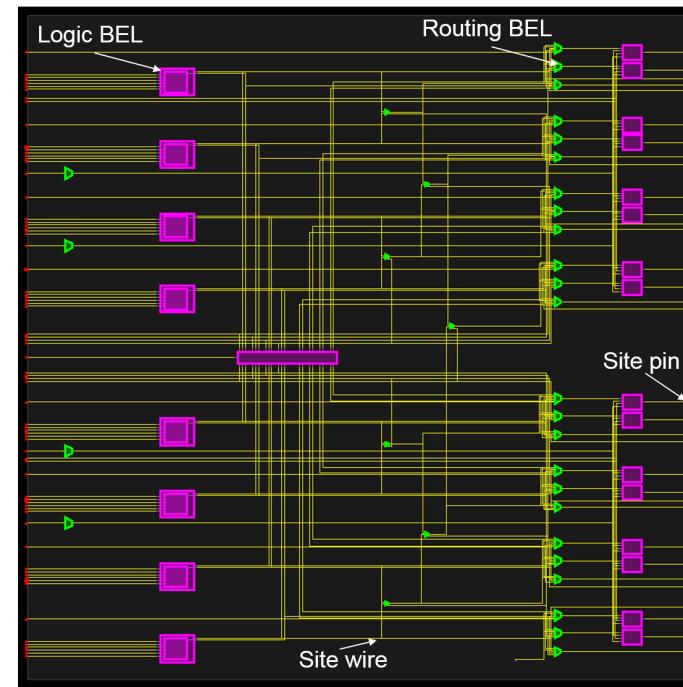
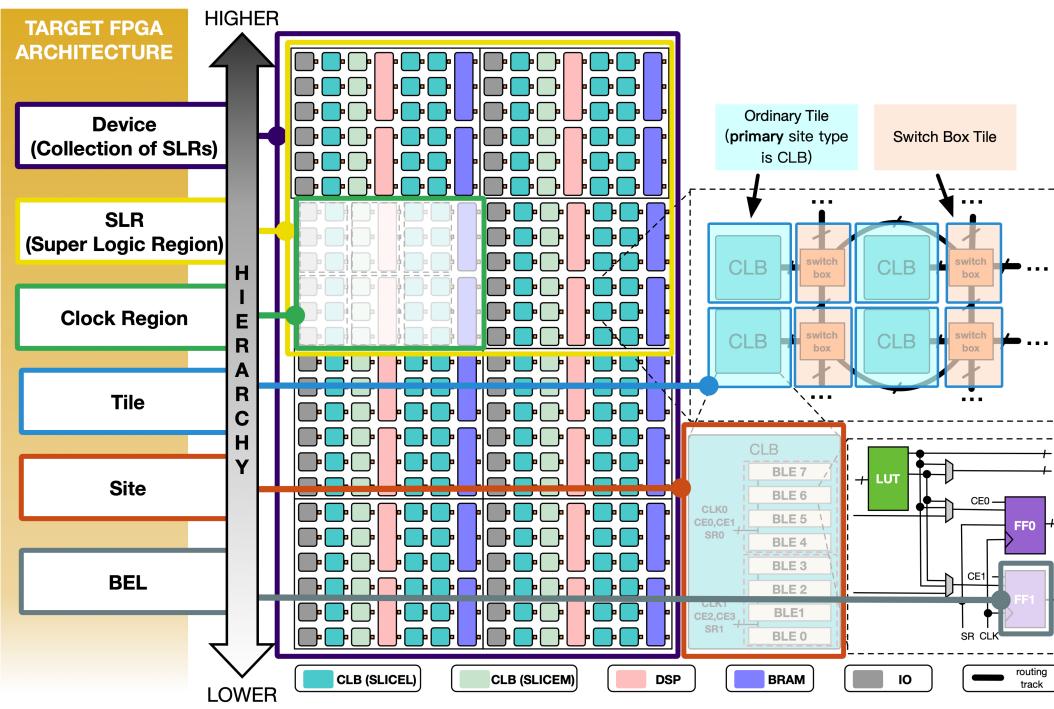
- ▶ A Site is composed of a set of interconnected BELs and their connections both internally and externally.
- ▶ The components of a Site include: 1) a set of interconnected BELs; 2) site pins for external input and output; 3) internal connections within the site, including wiring between BELs and between BELs and site pins.
- ▶ The main types of Sites include Configurable Logic Blocks (**CLBs**), Digital Signal Processing units (**DSPs**), Block RAMs (**BRAMs**), and Input/Output Blocks (**IOs**) for interfacing purposes.



# FPGA Hierarchical Architecture (V, cont'd)

## Site

- Among the different types of Sites, CLBs are the most numerous and widely distributed. CLBs can be divided into two heterogeneous types based on configurable logic: **SLICEL** and **SLICEM**.
- SLICEL site has 8 LUTs (Look-Up Tables), 16 flip-flops (FFs), 2 clock signals (CLK0, CLK1), 4 clock enable signals (CE0, CE1, CE2, CE3), and 2 set-reset signals (SR0, SR1).
- Depending on the combination of BELs and clock control signals, CLBs are further divided into 8 groups of Basic Logic Elements (BLEs), each consisting of one LUT and two adjacent FFs.

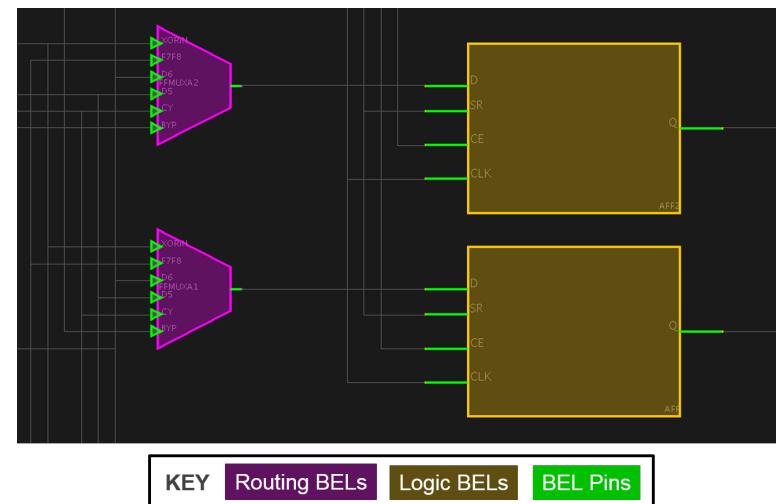
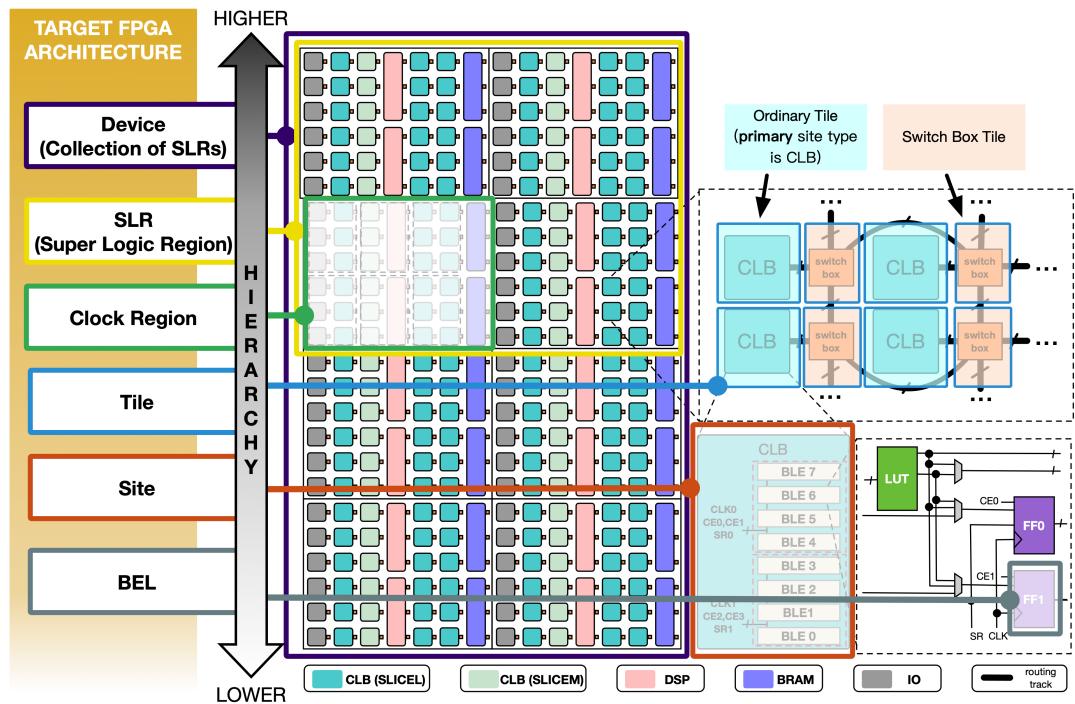


Close up view of SLICEL Site

# FPGA Hierarchical Architecture (VI)

## BEL (Basic Element of Logic)

- ▶ BEL is the fundamental building block in the architecture of an FPGA.
- ▶ There are two types of BELs in an FPGA: logic BELs and routing BELs.
- ▶ Logic BELs represent the smallest units of FPGA logic functions, including components such as Lookup Tables (LUTs) and Flip-Flops (FFs), Carry Chains (CARRYs).
- ▶ During the placement phase, the mapping between the leaf units in the netlist (non-leaf units do not represent implementable hardware but indicate the hierarchical structure) logic BELs is referred to as the "placement" of the units.

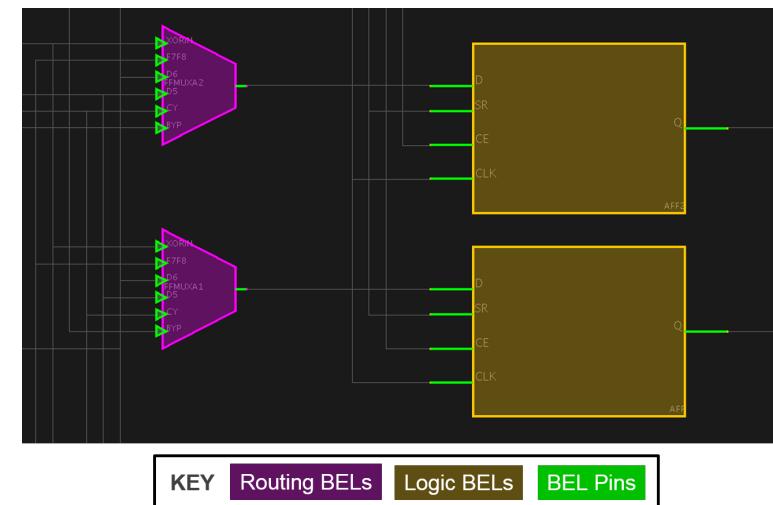
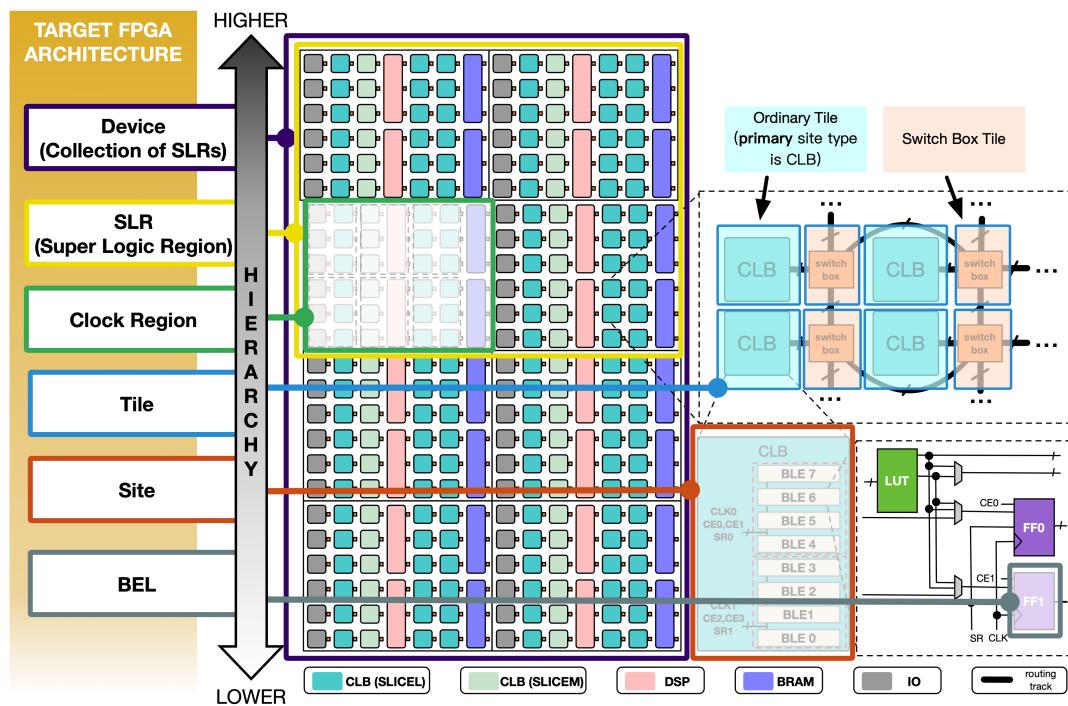


Two routing muxes (routing BELs) and two flip flops (logic BELs)  
 [RapidWright, 2023]

# FPGA Hierarchical Architecture (VI, cont'd)

## BEL (Basic Element of Logic)

- ▶ Therefore, when running the Vivado command `place_design`, what is actually happening is the mapping of all leaf units in the netlist to compatible and legal BELs.
- ▶ Certain logic BELs (such as DSPs, BRAMs, and CARRYs) are also required to be placed vertically in sequence within continuous sites (*cascaded constraints*).
- ▶ Routing BELs are programmable routing multiplexers used to route signals between BELs.
- ▶ Routing BELs cannot be mapped by logical units in the netlist; they are solely used for routing.



Two routing muxes (routing BELs) and two flip flops (logic BELs)  
 [RapidWright, 2023]

# FPGA Architecture & Its Challenges



The modern FPGAs come with advanced technologies along with more challenges.

## Massive Design

- ▶ Millions of cells in a single FPGA design
- ▶ Millions routing tracks for FPGA Routing [Wang+, ASP-DAC' 23]

## Highly Heterogeneity

- ▶ Various cells types (LUT, FF, DSP, BRAM, CARRY, Distributed RAM, Shift Register, etc)
- ▶ SLICEL-SLICEM Heterogeneity [Mai+, DAC'22]

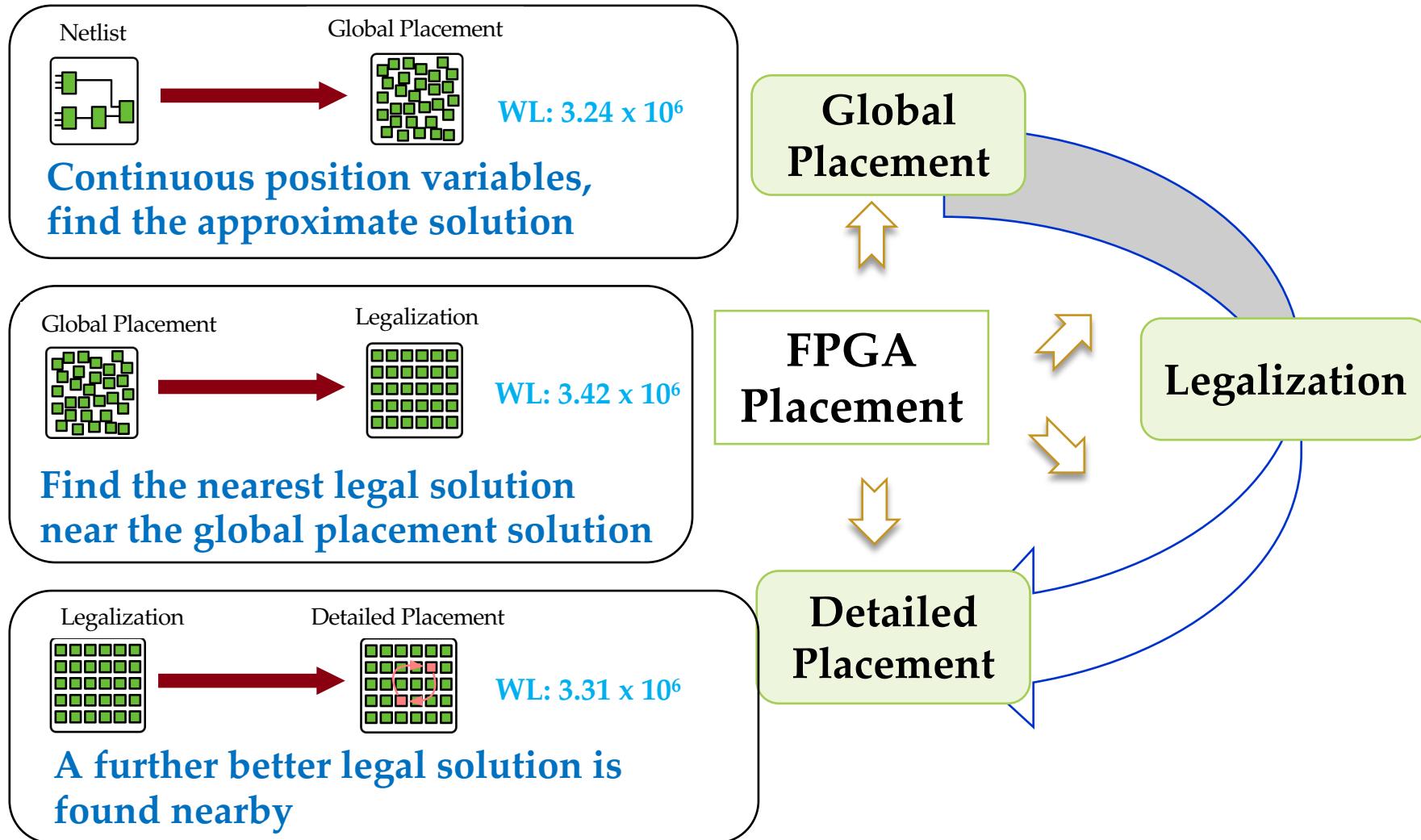
## Highly Discrete

- ▶ Clock region constraint & half-column region constraints [Li+, FPGA'19]
- ▶ Cascaded constraint [Mai+, ISEDA'24]
- ▶ Timing constraint [Mai+, TCAD'23]
- ▶ Fence Region Constraint [Mai+, ISEDA'24]
- ▶ ...

# Basic Problems in FPGA Placement



FPGA placement is essentially a combinatorial optimization problem, which is too complicated to solve directly, so it is decomposed into three basic problems.



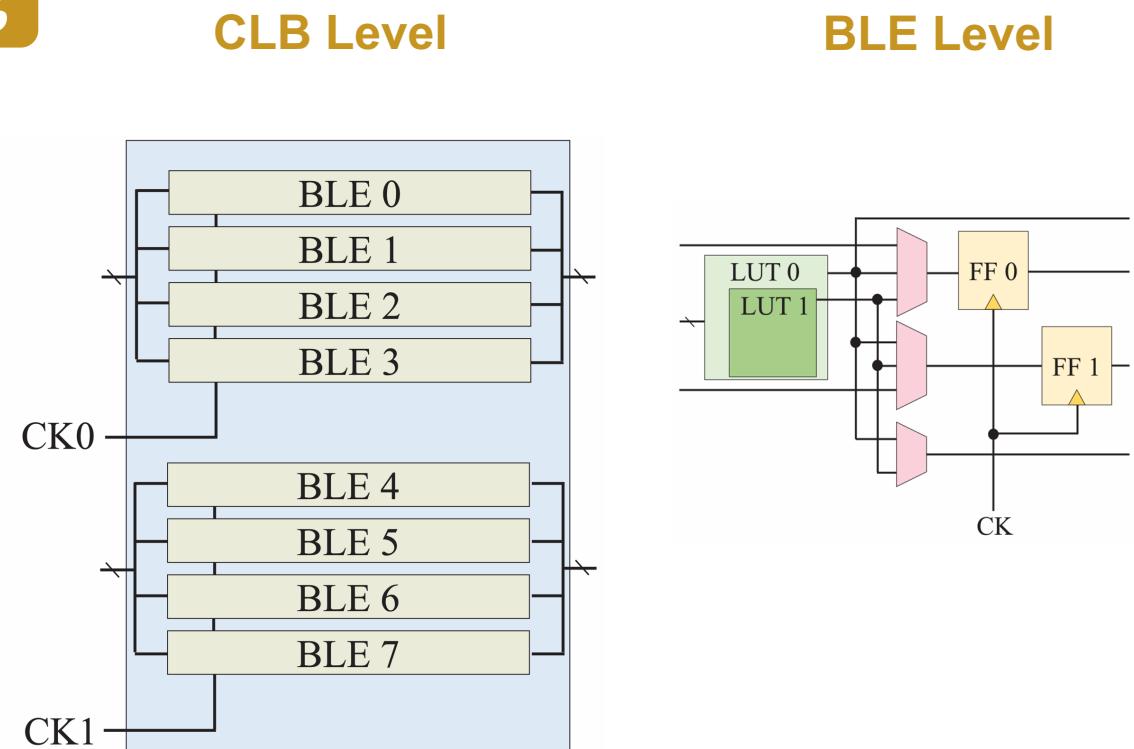
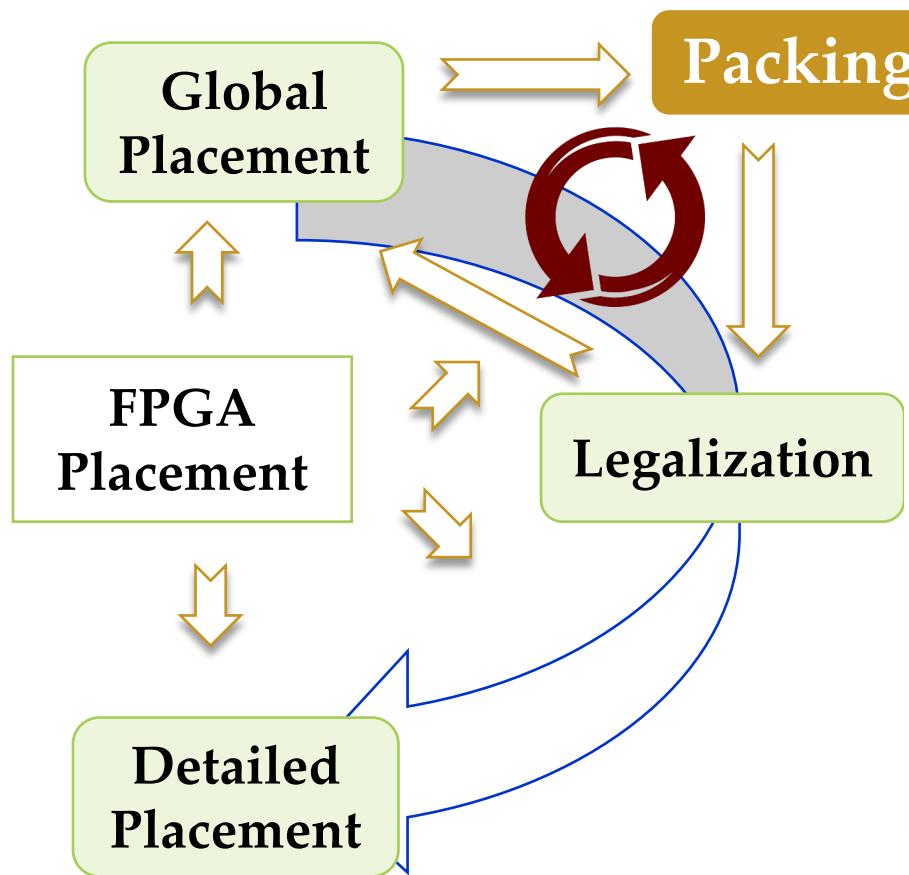
# Basic Problems in FPGA Placement (cont'd)

An FPGA-specific FPGA problem: packing

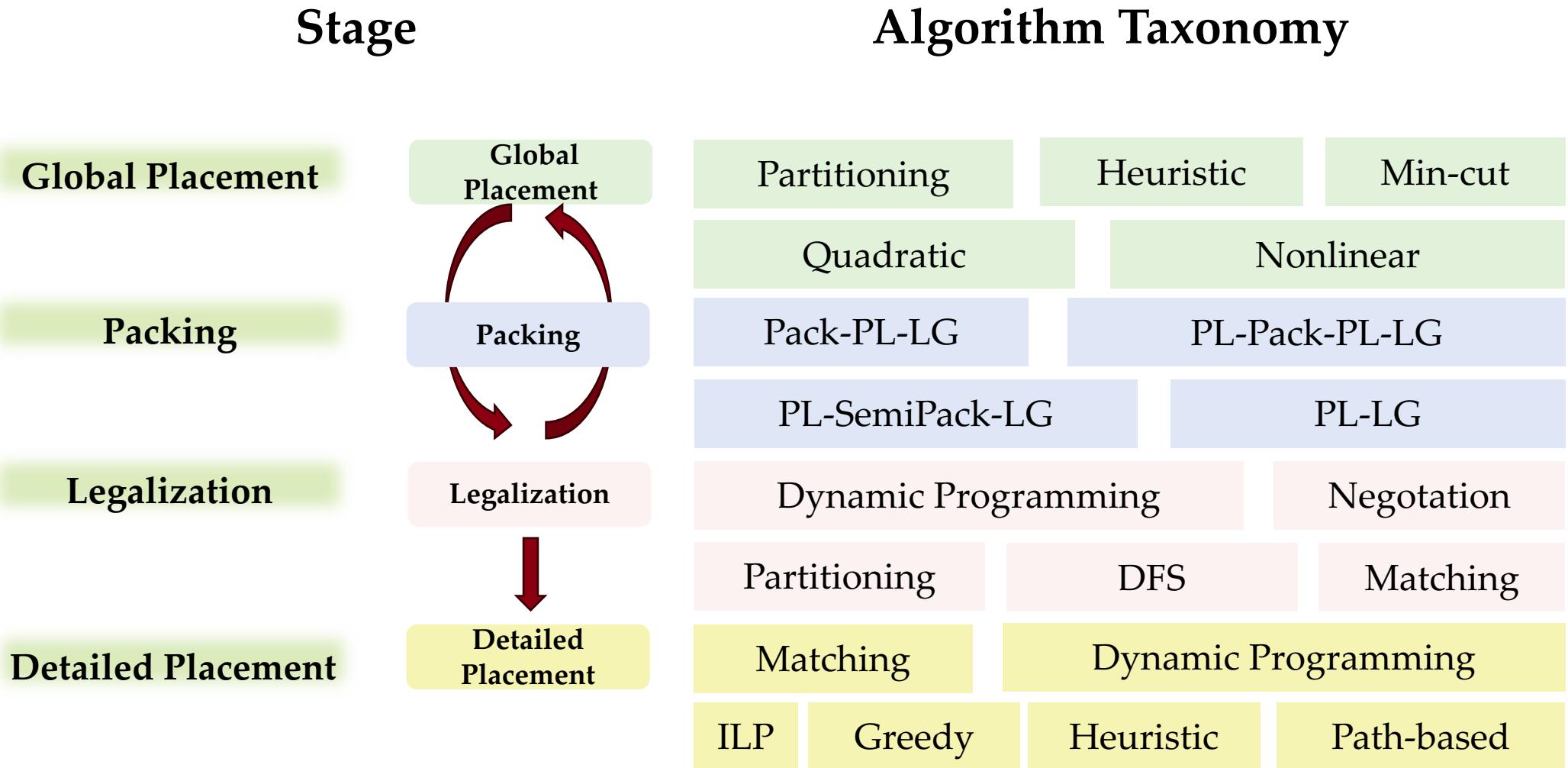
**Problem sources** FPGA hierarchical architecture, i.e., the combination rules of LUTs and FFs within the CLB

**Relationship with basic problems** Alternate hierarchical global placement and legalization

**Packing Level** CLB Level, BLE Level



# FPGA Placement Algorithm Taxonomy



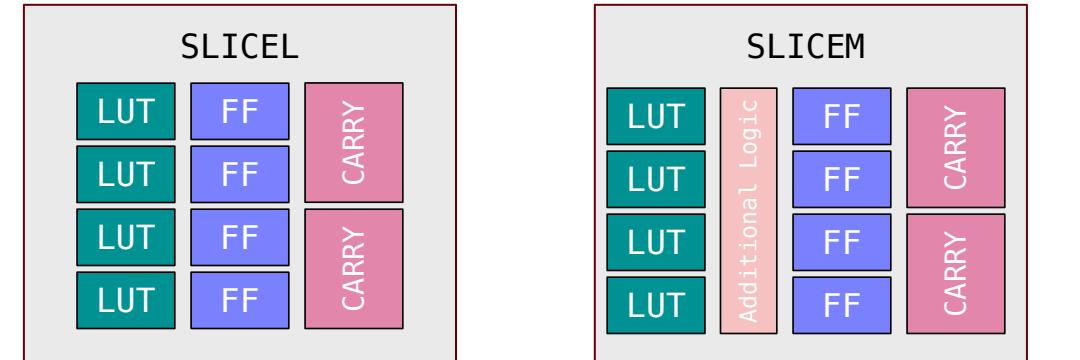
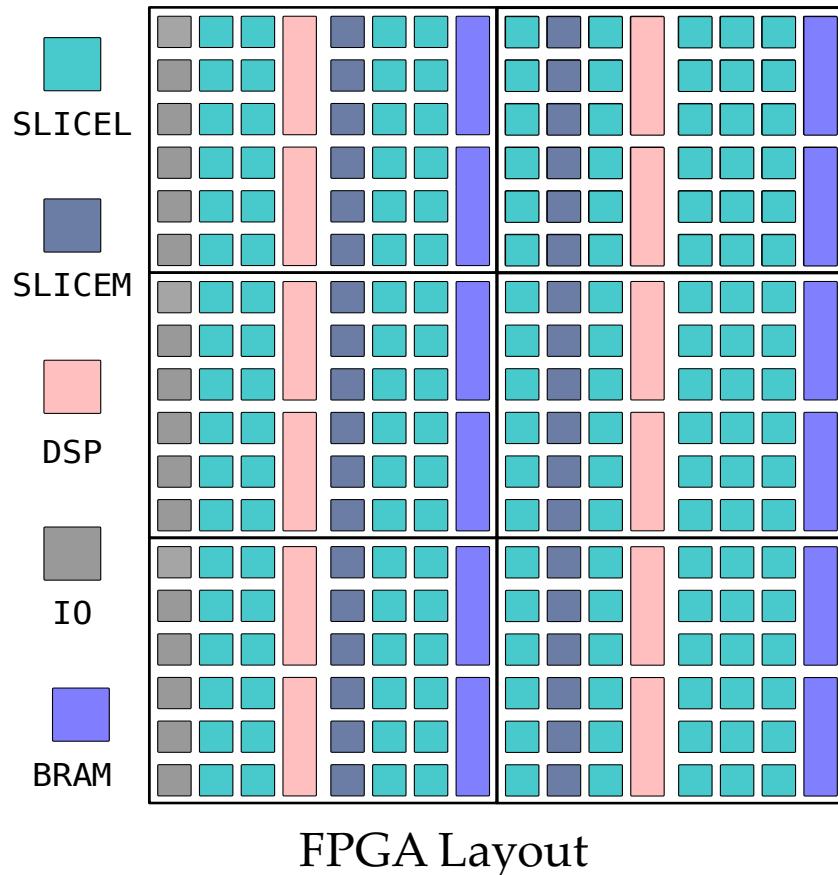
# Other Published SOTA FPGA Placers

Placer	Clock Constraints	LUT, FF, BRAM, DSP Supported	CARRY, SHIFT, distributed RAM Supported	GPU-Acceleration	Algorithm Category
RippleFPGA	✗	✓	✗	✗	Quadratic
GPlace	✗	✓	✗	✗	Quadratic
UTPlaceF	✗	✓	✗	✗	Quadratic
elfPlace	✗	✓	✗	✓	Nonlinear
GPlace 3.0	✗	✓	✗	✗	Quadratic
RippleFPGA Clock-Aware	✓	✓	✗	✗	Quadratic
UTPlaceF 2.0 & 2.X	✓	✓	✗	✗	Quadratic
NTUfPlace	✓	✓	✗	✗	Nonlinear
Ours	✓	✓	✓	✓	Nonlinear

# **OpenPARF 0.1: SLICEL-SLICEM Heterogeneity, Clock Feasibility**

# Two Categories of CLBs: SLICEL and SLICEM

- ▶ A Configurable Logic Block (CLB) is further categorized to **SLICEL** and **SLICEM** with asymmetric compatibility.
- ▶ On the basis of SLICEL, SLICEM adds additional logic signals, so it can be configured not only as LUT logic units with logical functions but also as distributed memory units (**Distributed RAM**) and shift registers (**Shift Register**) with storage functions.

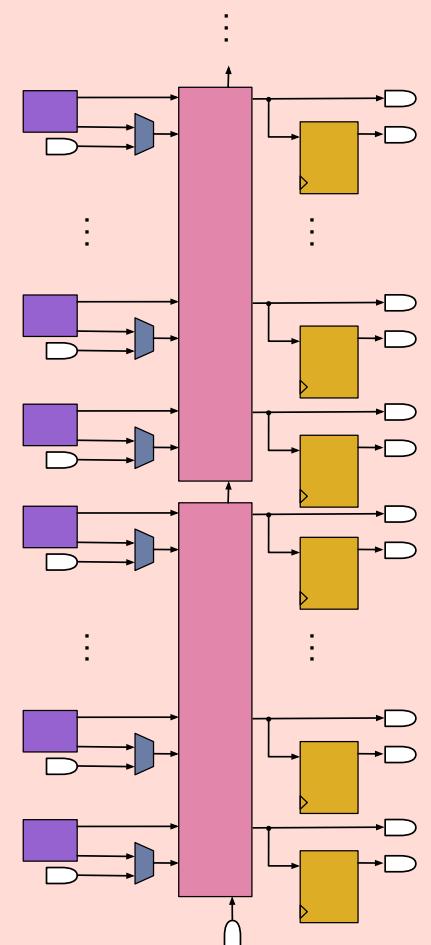
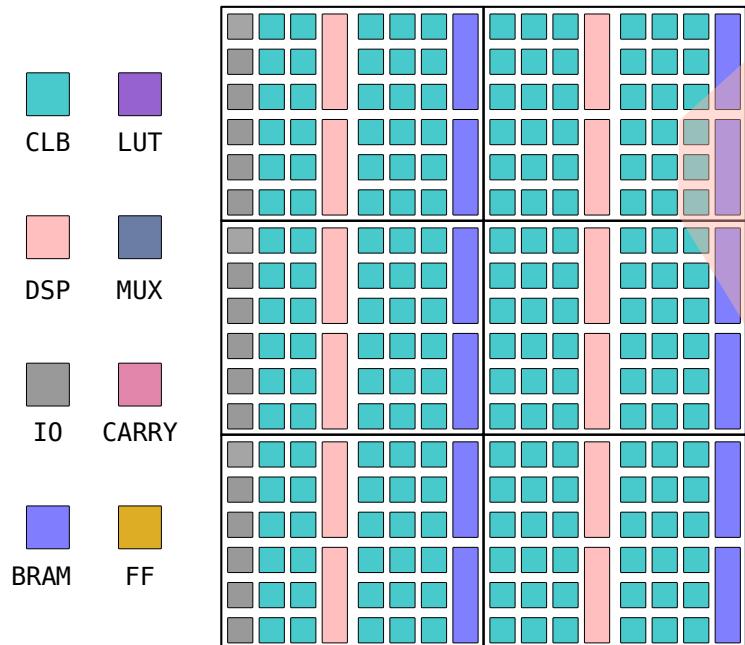


CLB Slice	LUTs	FFs	CARRY	Distributed RAM	SHIFT
SLICEL	8	16	1	NA	NA
SLICEM	8	16	1	512 bits	512 bits

Logic Resource on One CLB

# Cascaded CARRYs

Cascaded CARRYs and the immediate LUTs and FFs should be placed in consecutive CLBs in the same column.



# Distributed RAM or SHIFT as SLICEM

## LUT-like cells

- ▶ Regular LUT, distributed RAM & shift register (SHIFT)

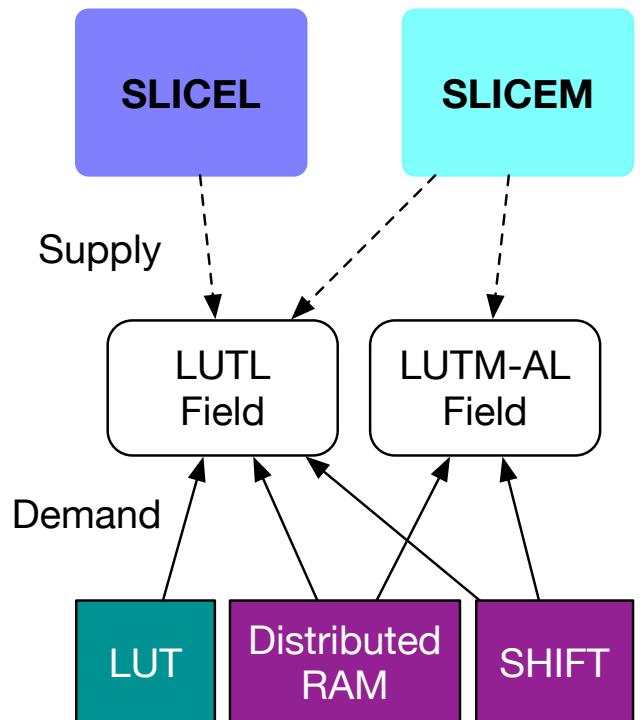
$$\text{SLICEL} = \text{LUT} \times 8 + \text{CARRY} \times 1 + \text{FF} \times 16$$

- ▶ LUT can only be used for logic (not memory)

$$\text{SLICEM} = (\text{LUT} / \text{distributed RAM} / \text{SHIFT}) \times 8 + \text{CARRY} \times 1 + \text{FF} \times 16$$

- ▶ LUT can only be used for logic and memory function
- ▶ No mixing between LUTs, distributed RAMs, and SHIFTs in a SLICEM is allowed

**CLB Slices**



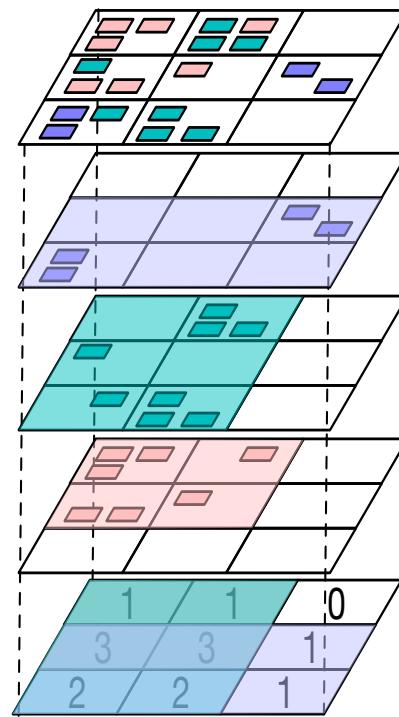
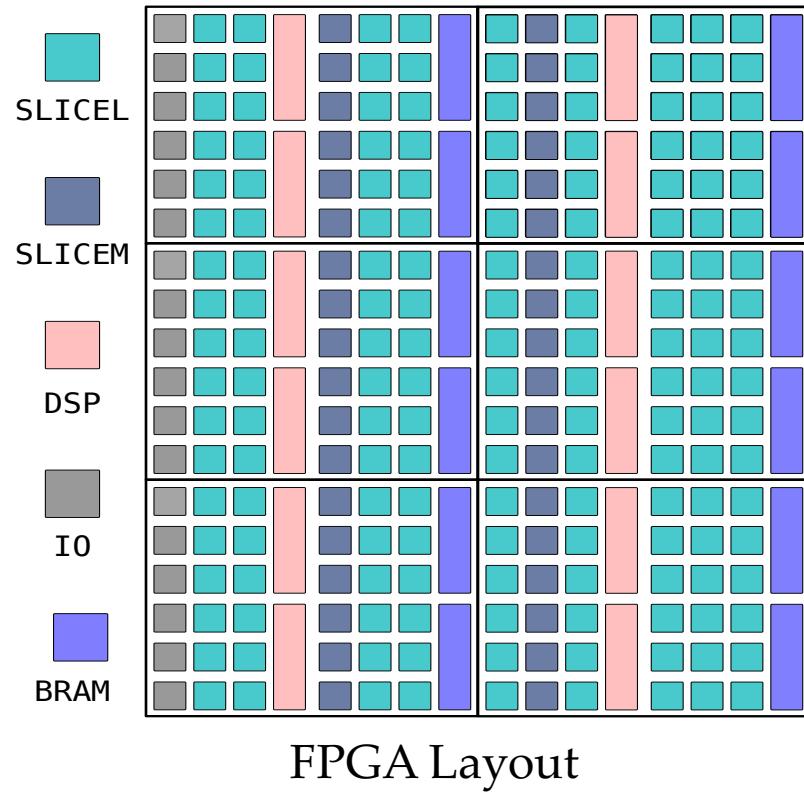
# Clock Constraints

## Clock Region Constraint

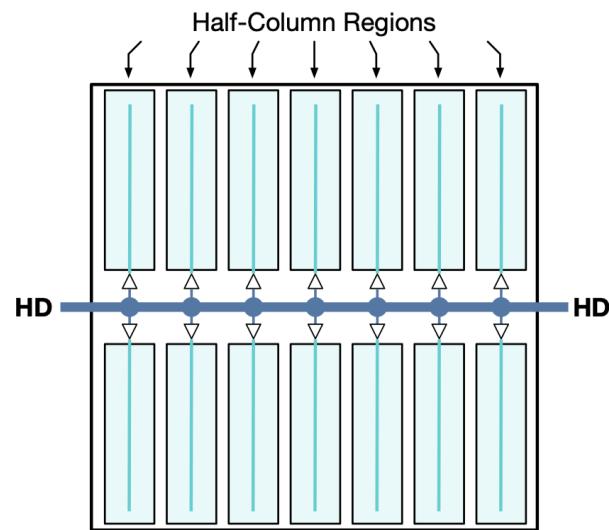
- The clock demand of each clock region is at most 24.

## Half Column Constraint

- The number of clock nets within the half column is at most 12.



Illustrations of  
the clock demand  
of a clock region.

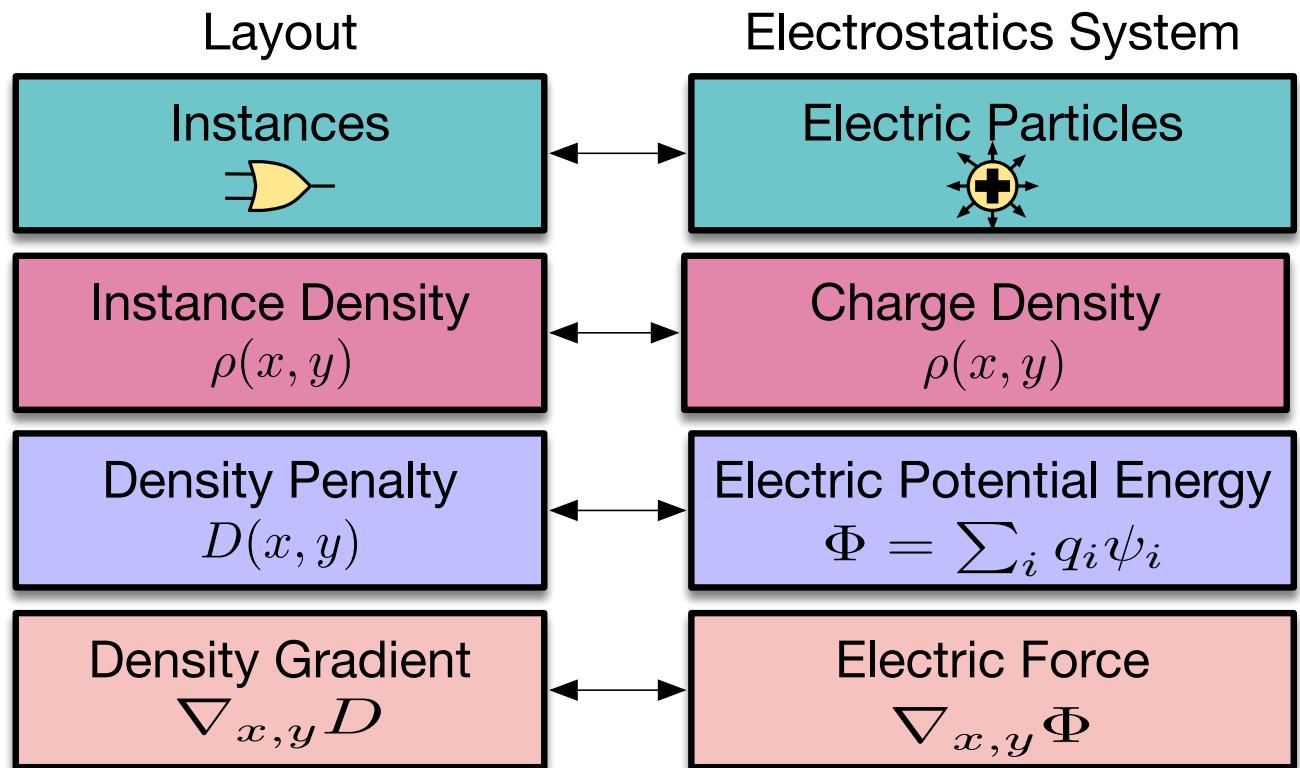


# Proposed Algorithm

# Multi-Electrostatics based Placement



Analogy between placement of a single resource type and an electrostatic system like ePlace [Lu+, TCAD'15]



# Multi-Electrostatics based Placement in elfPlace

Resource Type Set  $\mathcal{T}$

$$\mathcal{T} = \{LUT, FF, DSP, BRAM, IO\}.$$

Multi-Electrostatics-based placement

Separate electrostatic system for each resource type [elfPlace, Meng+, TCAD'21]

$$\begin{aligned} & \min_{\mathbf{x}, \mathbf{y}} \tilde{W}(\mathbf{x}, \mathbf{y}), \\ & \text{s.t. } \Phi_s(\mathbf{x}, \mathbf{y}) = 0, \forall s \in \mathcal{T} \end{aligned}$$

Wirelength

Weighted-average WL model with smoothness control [Hsu+, TCAD'13]

Modified augmented Lagrangian formulation [Zhu+, DAC'18]

$$\begin{aligned} & \min_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}) = \tilde{W}(\mathbf{x}, \mathbf{y}) + \sum_{s \in S} \lambda_s \mathcal{D}_s, \\ & \text{s.t. } \mathcal{D}_s = \Phi_s + \frac{\mu}{2} \mathcal{P}_s \Phi_s^2, \forall s \in \mathcal{T} \end{aligned}$$

Density weight preconditioner  $\mathcal{P}_s$

$$\mathcal{P}_s = 1 / \Phi_s^{(0)}$$

# Multi-Electrostatics based Placement in OpenPARF 0.1



## Multi-Electrostatics-based placement

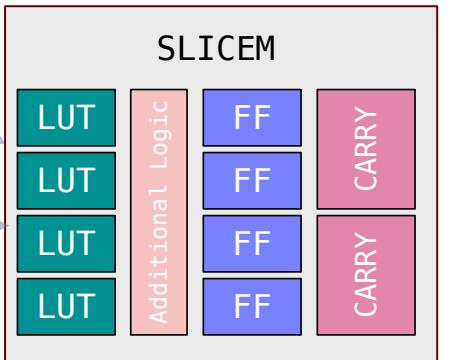
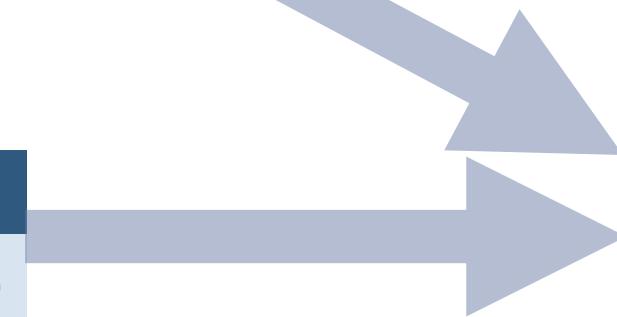
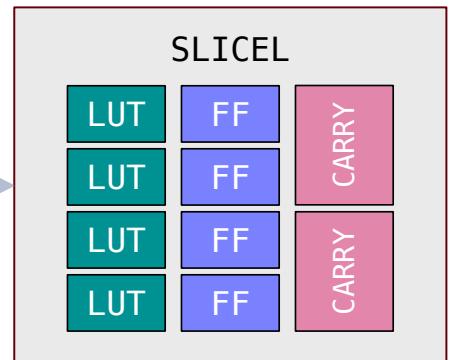
$$\min_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}) = \tilde{W}(\mathbf{x}, \mathbf{y}) + \sum_{s \in S} \lambda_s \mathcal{D}_s + \eta \Gamma(\mathbf{x}, \mathbf{y}),$$

$$s.t. \mathcal{D}_s = \Phi_s + \frac{\mu}{2} \mathcal{P}_s \Phi_s^2, \forall s \in S,$$

*Cascaded CARRY Constraints,*

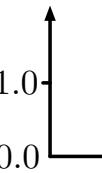
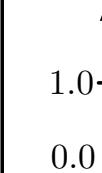
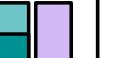
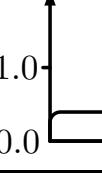
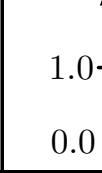
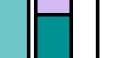
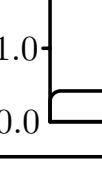
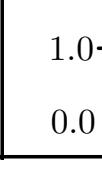
## SLICEL-SLICEM-aware Electrostatics Density Model

$$S = \{S_{LUTL}^D, S_{LUTM-AL}^D, S_{FF}^D, S_{DSP}^D, S_{BRAM}^D, S_{IO}^D\}.$$



# Special Field Mechanism

Fields LUT and LUTM-AL form an asymmetric combination mechanism together.

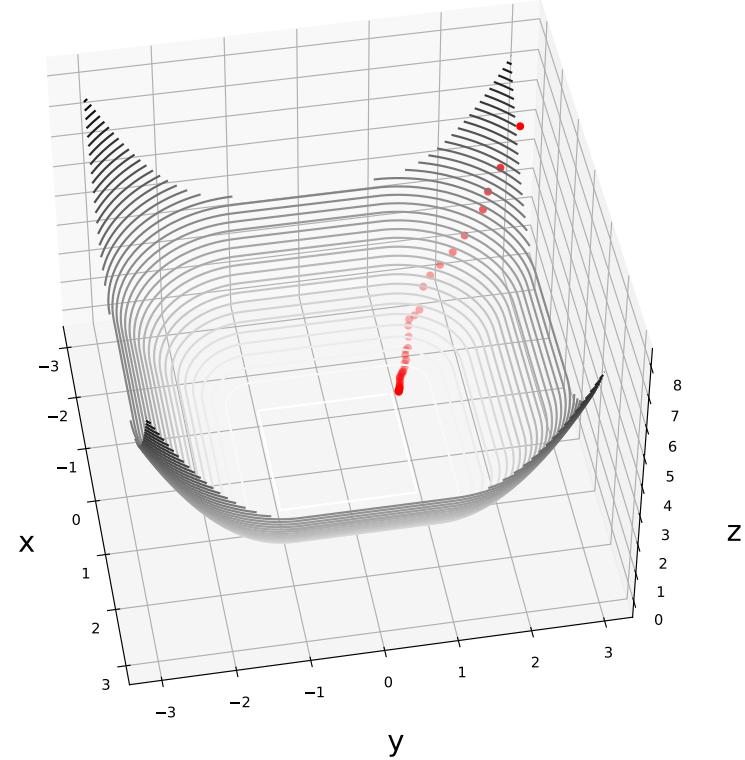
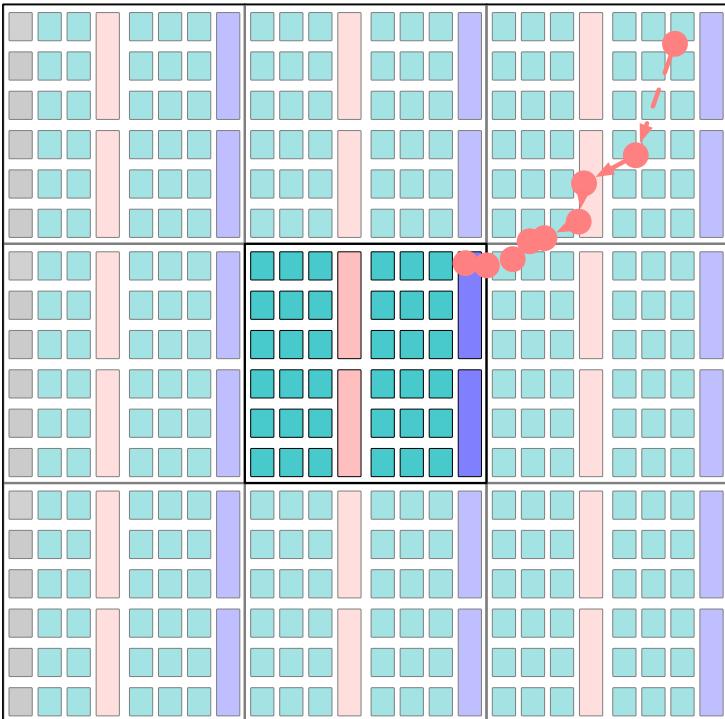
	LUTL Field		LUTM-AL Field		Two Fields	
	$\rho_{LUT}$	$\Phi_{LUT}$	$\rho_{LUTM\_AL}$	$\Phi_{LUTM\_AL}$	$\Phi_{LUT} + \Phi_{LUTM\_AL}$	
<b>Initial</b>			-		-	-
<b>Solution I</b>			Low		Low	Low
<b>Solution II</b>			Low		High	High



# Two-stage Clock Network Planning

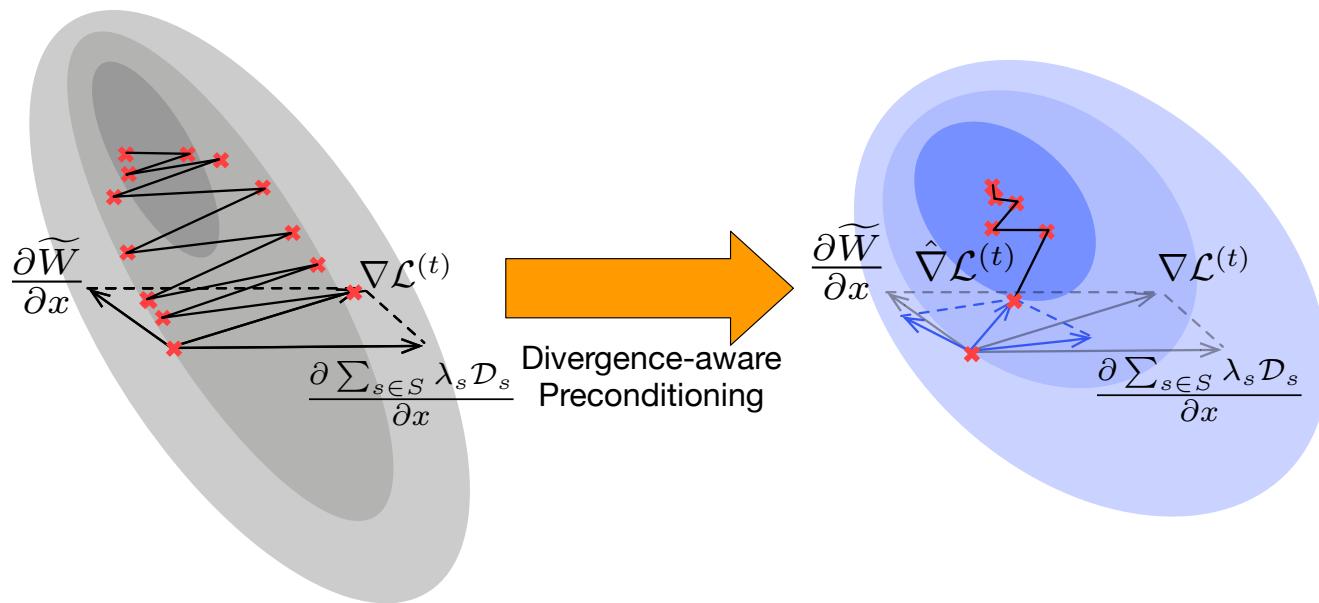
- ▶ Instance-to-clock region mapping using the branch-and-bound method [UTPlaceF 2.0, Li+, TDAES'18]
- ▶ Bowl-like and differentiable “gravitational” attraction terms for optimization

$$\min_{x,y} \mathcal{L}(x, y; \lambda, \mathcal{A}, \eta) = \widetilde{W}(x, y) + \sum_{s \in S} \lambda_s \mathcal{D}_s + \eta \Gamma(x, y)$$



# Divergence-aware Preconditioning

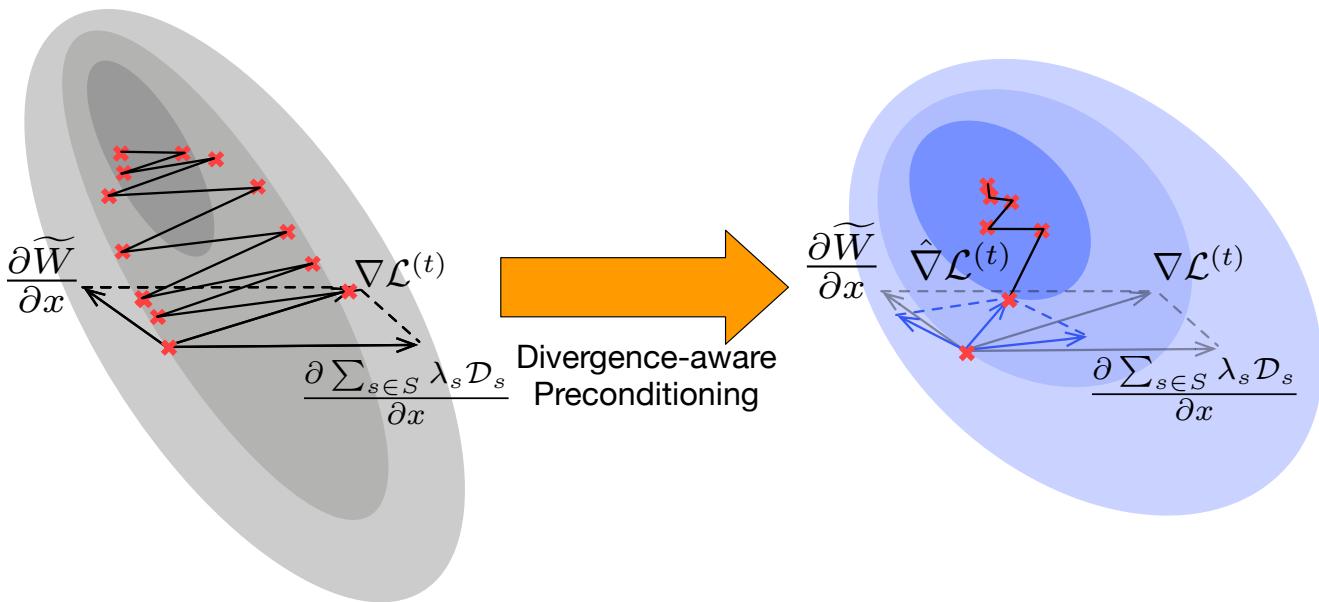
- ▶ Stabilizes convergence by preconditioning the gradient  $\nabla \mathcal{L}^{(t)}$  by  $\nabla \mathcal{L}^{(t)} \odot \mathcal{P}^{(t)}$
- ▶ Improve elfPlace's approaches using the divergence-aware weighting method
- ▶ `elfPlace`:  $\mathcal{P}_i^{(t)} \sim \max\left(1, \left[\mathcal{P}_i^W + \lambda_s^{(t)} \mathcal{A}_i^s\right]^{-1}\right), \forall i \in \mathcal{V}_s, \quad \mathcal{P}_i^W = \frac{\partial^2 \tilde{W}(\mathbf{x}, \mathbf{y})}{\partial x_i^2} \sim \sum_{e \in E_i} \frac{w_e}{|e| - 1}, \forall i \in \mathcal{V}$
- ▶ Ours:  $\mathcal{P}_i^{(t)} \sim \max\left(1, \left[\mathcal{P}_i^W + \sum_{s \in S} \alpha_s^{(t)} \lambda_s^{(t)} \mathcal{A}_i^s\right]^{-1}\right)$



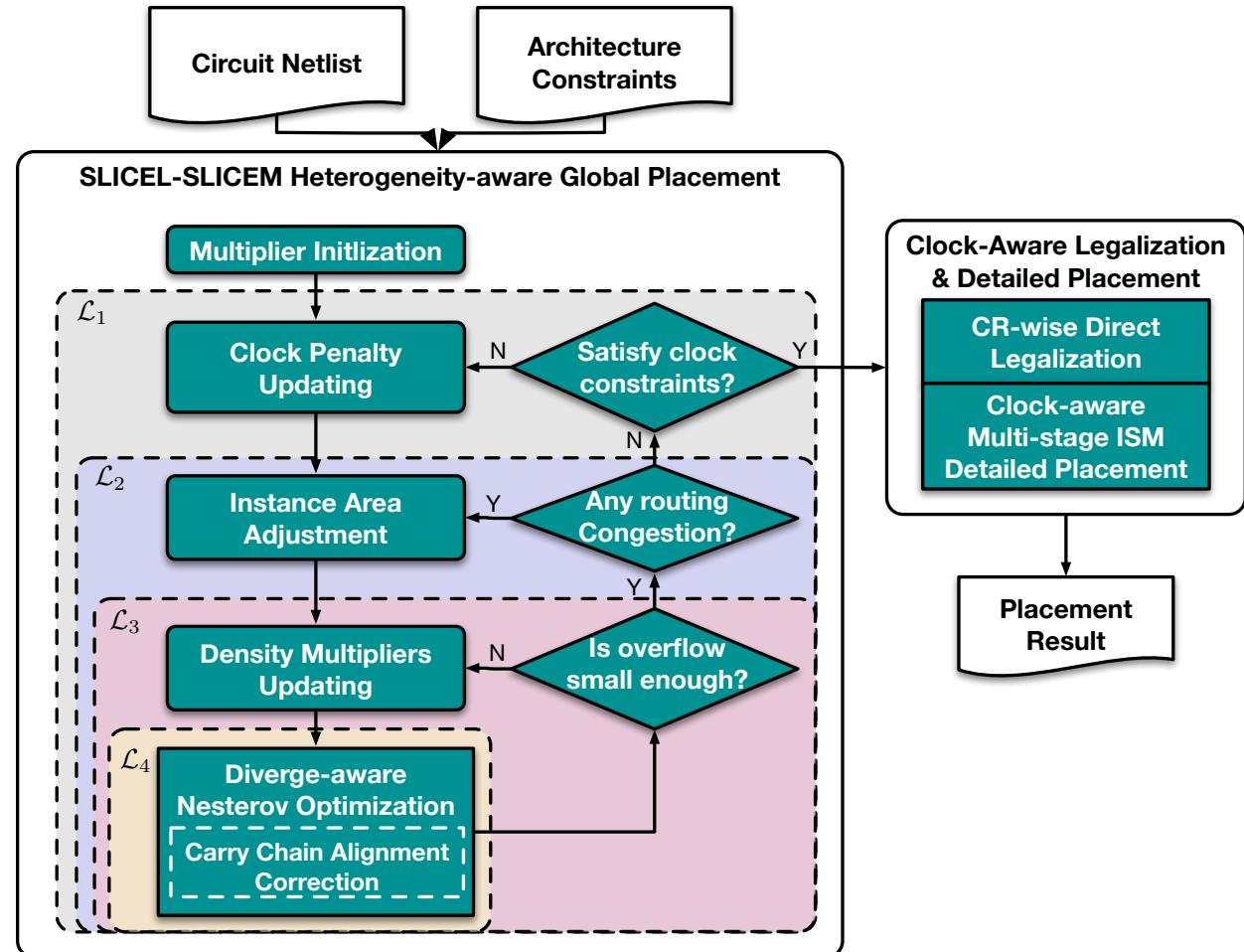
# Divergence-aware Preconditioning

- ▶  $\alpha_s^{(t)}$  balances the ratio of the gradient norms from density and wirelength
- ▶ The fast increasing of this ratio indicates the divergence

$$\begin{aligned}\vartheta_s^{(t)} &= \max\left(1, \frac{\nabla \mathcal{D}_s}{\sum_{i \in \mathcal{V}_s^r} |\partial \tilde{W}/\partial x_i|}\right), \forall s \in S, \\ \bar{\mathcal{P}}_s^W &= \frac{\sum_{i \in \mathcal{V}_s^r} \mathcal{P}_i^W}{|\mathcal{V}_s^r|}, \forall s \in S, \\ \boldsymbol{\alpha}^{(t)} &= \boldsymbol{\vartheta}^{(t)} \odot \bar{\mathcal{P}}^W,\end{aligned}$$



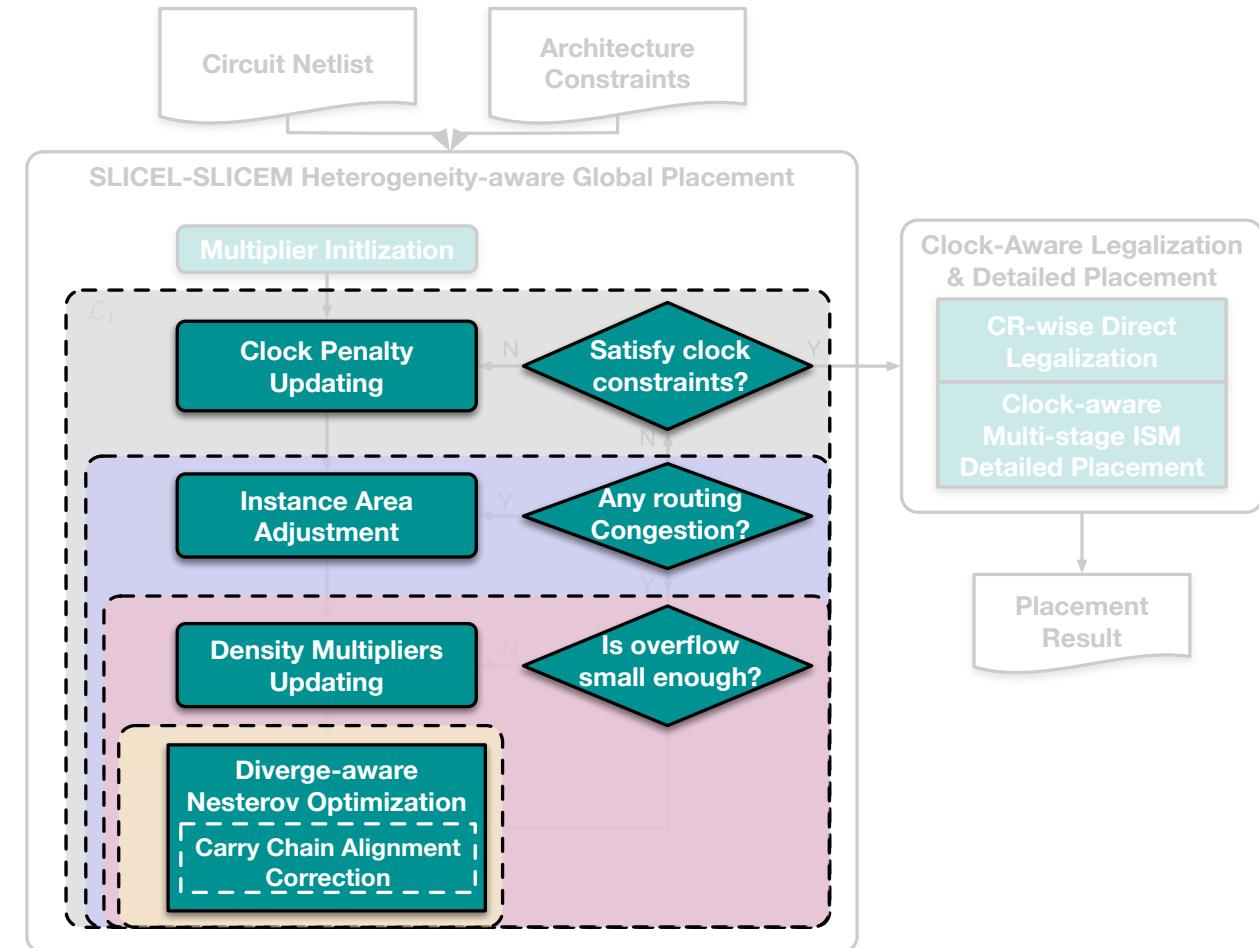
# Overall Flow: Nested Optimization Framework



# Overall Flow: Nested Optimization Framework

## Clock Opt.

- ▶  $\mathcal{L}_1 = \max_{\eta} \mathcal{L}_2(\eta)$
- ▶ Two-stage clock network planning



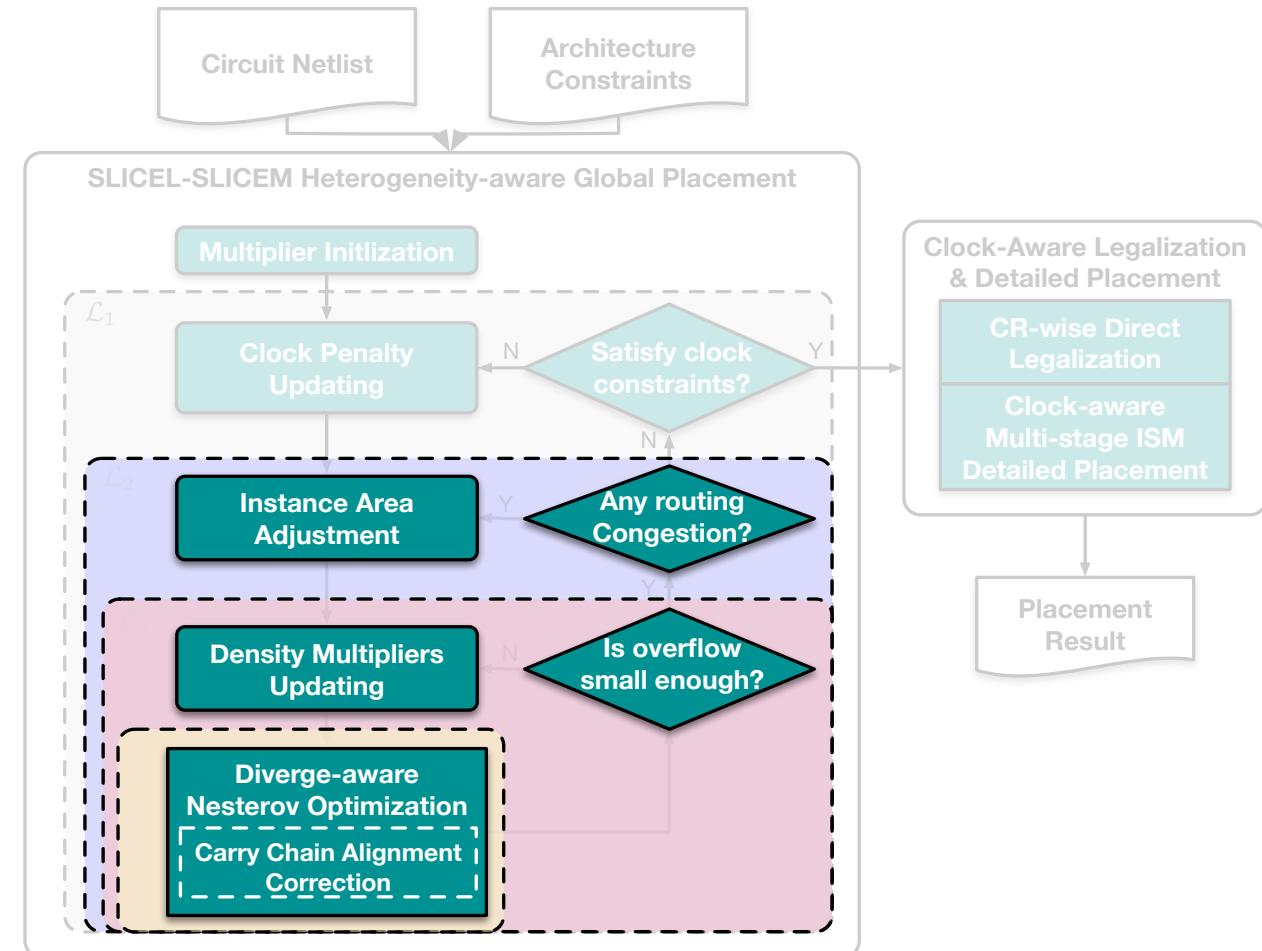
# Overall Flow: Nested Optimization Framework

## Clock Opt.

- ▶  $\mathcal{L}_1 = \max_{\eta} \mathcal{L}_2(\eta)$
- ▶ Two-stage clock network planning

## Routability Opt.

- ▶  $\mathcal{L}_2(\eta) = \max_{\mathcal{A}} \mathcal{L}_3(\mathcal{A}, \eta)$
- ▶ Area adjustment scheme



# Overall Flow: Nested Optimization Framework

## Clock Opt.

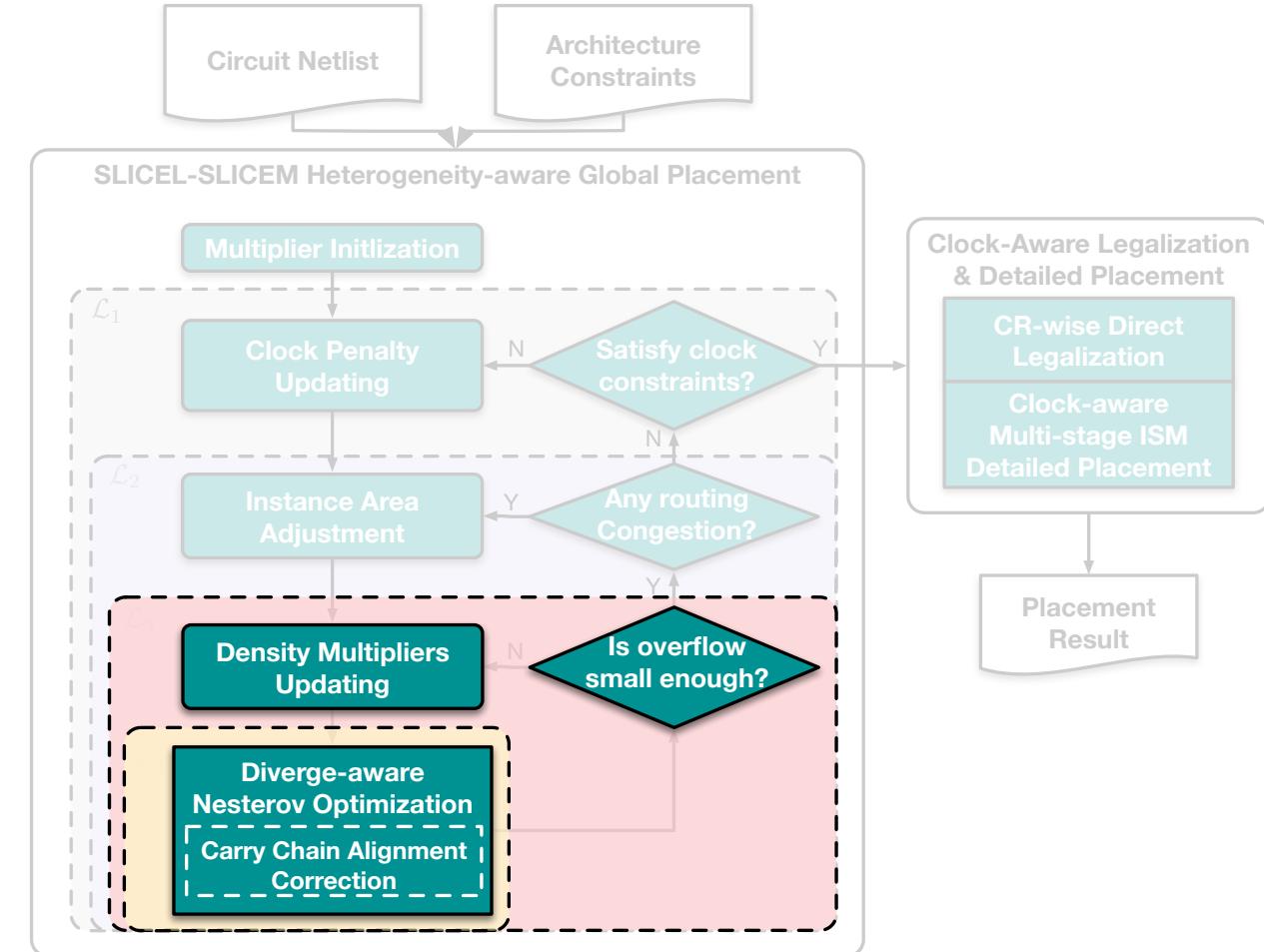
- ▶  $\mathcal{L}_1 = \max_{\eta} \mathcal{L}_2(\eta)$
- ▶ Two-stage clock network planning

## Routability Opt.

- ▶  $\mathcal{L}_2(\eta) = \max_{\mathcal{A}} \mathcal{L}_3(\mathcal{A}, \eta)$
- ▶ Area adjustment scheme

## Wirelength Opt.

- ▶  $\mathcal{L}_3(\mathcal{A}, \eta) = \max_{\lambda} \mathcal{L}_4(\lambda, \mathcal{A}, \eta)$
- ▶ Weighted Average (WA) wirelength



# Overall Flow: Nested Optimization Framework

## Clock Opt.

- ▶  $\mathcal{L}_1 = \max_{\eta} \mathcal{L}_2(\eta)$
- ▶ Two-stage clock network planning

## Routability Opt.

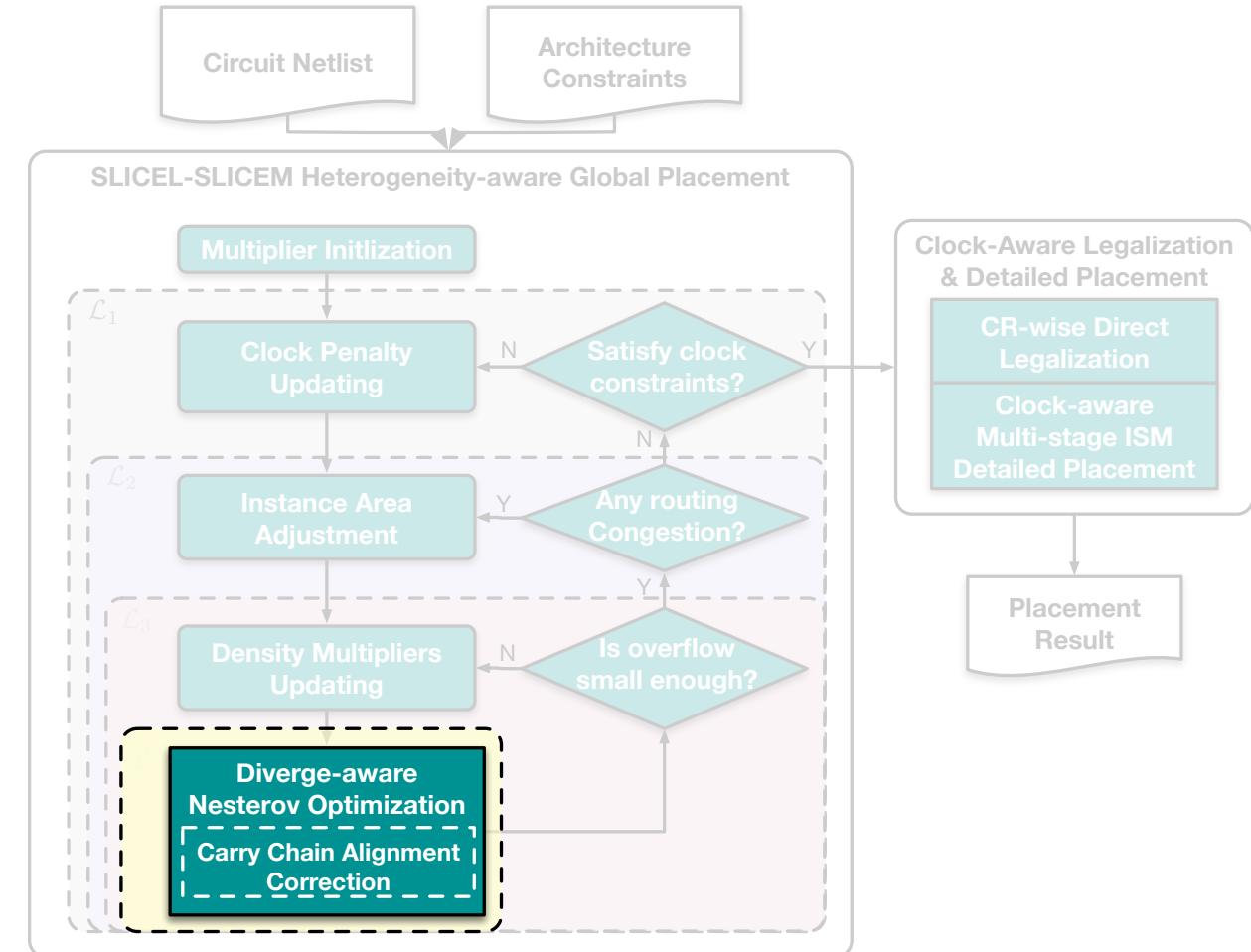
- ▶  $\mathcal{L}_2(\eta) = \max_{\mathcal{A}} \mathcal{L}_3(\mathcal{A}, \eta)$
- ▶ Area adjustment scheme

## Wirelength Opt.

- ▶  $\mathcal{L}_3(\mathcal{A}, \eta) = \max_{\lambda} \mathcal{L}_4(\lambda, \mathcal{A}, \eta)$
- ▶ Weighted Average (WA) wirelength

## Subproblem Opt.

- ▶  $\mathcal{L}_4(\lambda, \mathcal{A}, \eta) = \min_{x,y} \mathcal{L}(x, y, \lambda, \mathcal{A}, \eta)$
- ▶ Divergence-aware Preconditioning
- ▶ CARRY alignment after each descending step



# **OpenPARF 0.2: Timing-driven FPGA Placement**



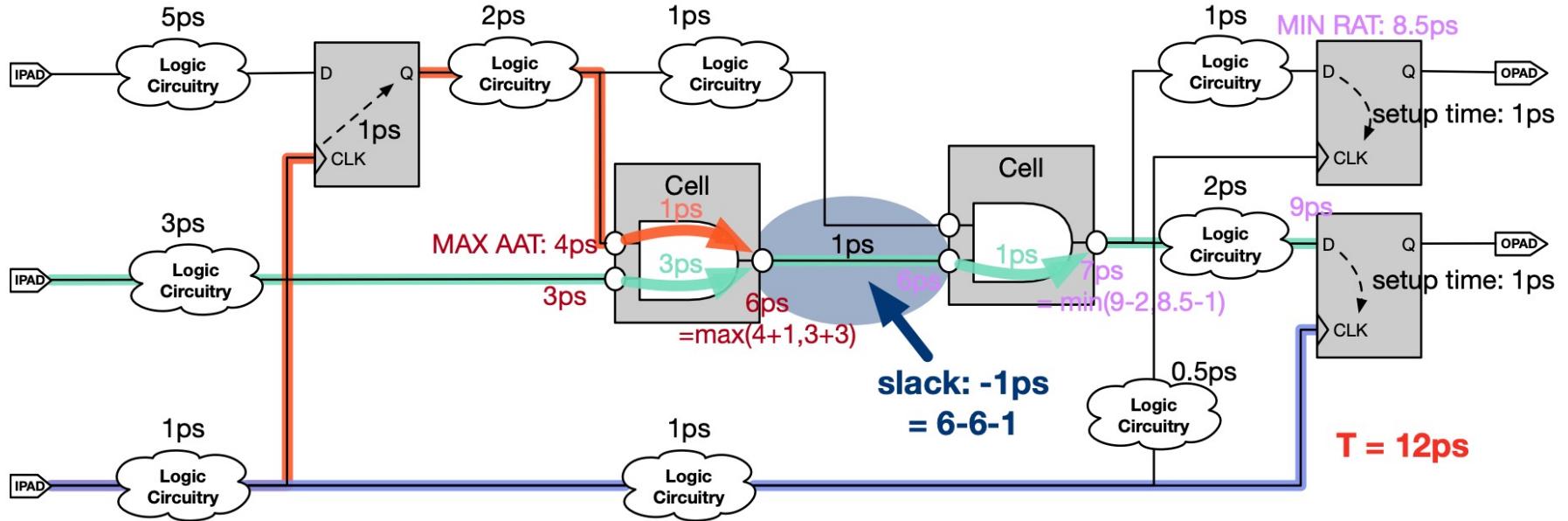
## Multi-electrostatics-based FPGA Placement

$$\begin{aligned}
 \min_{\mathbf{x}, \mathbf{y}} \quad & \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\lambda}, \mathcal{A}, \eta, \boldsymbol{\omega}) = \boxed{\tilde{\mathcal{T}}_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y})} + \sum_{s \in S} \lambda_s \mathcal{D}_s \\
 & + \eta \Gamma(\mathbf{x}, \mathbf{y}), \\
 \mathcal{D}_s = & \Phi_s + \frac{1}{2} \mathcal{C}_s \Phi_s^2, \quad \forall s \in S,
 \end{aligned}$$

## Timing-Criticality-based Weighting Method

$$\tilde{\mathcal{T}}_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y}) = \sum_{e \in E} \omega_e \cdot \widetilde{W}(e)$$

# Static Timing Analysis



Actual Arrival Time & Required Arrival Time

$$T_{ATT}(v_i) = \max_{v_j \in fanin(v_i)} T_{ATT}(v_j) + e_{j,i}$$

$$T_{ATT}(v_i) = \min_{v_j \in fanout(v_i)} T_{RAT}(v_j) - e_{i,j}$$

Timing Slack

$$s_{i,j} = T_{RAT}(v_j) - T_{ATT}(v_i) - e_{i,j},$$

# Timing-Criticality-based Weighting Method

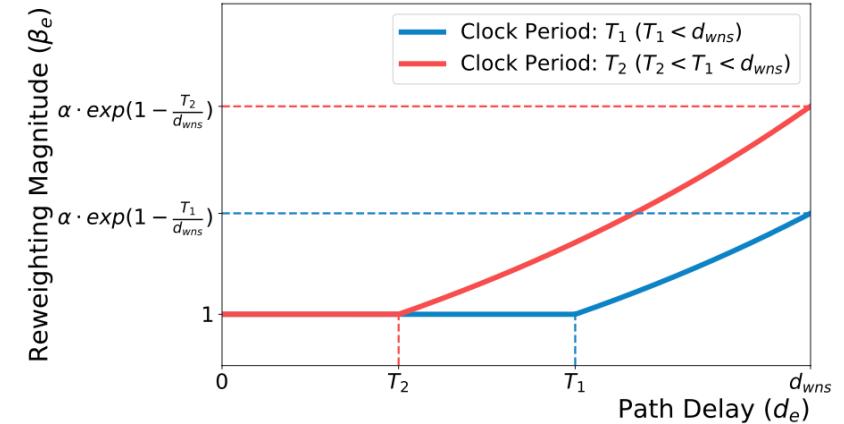
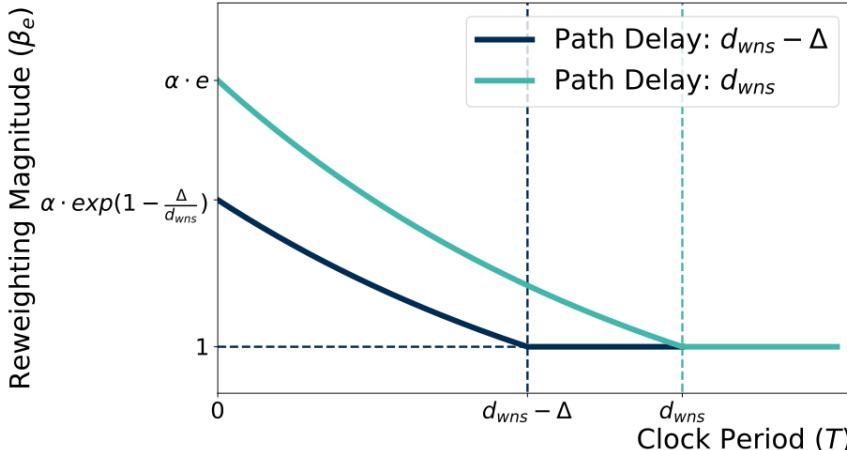
Timing criticality

$$c_e = \frac{\min(0, s_e)}{\min(0, s_{wns}) - T} \in [0, 1],$$

Timing criticality-based net weight

$$\beta_e = \alpha \cdot \max(1, \exp(c_e))$$

$$\omega'_e \leftarrow \omega_e \cdot \beta_e$$



# Nested Optimization Framework

Modified augmented Lagrangian formulation [Zhu+, DAC'18]

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{y}} \quad & \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\lambda}, \mathcal{A}, \eta, \boldsymbol{\omega}) = \tilde{\mathcal{T}}_{\boldsymbol{\omega}}(\mathbf{x}, \mathbf{y}) + \sum_{s \in S} \lambda_s \mathcal{D}_s \\ & + \eta \Gamma(\mathbf{x}, \mathbf{y}), \\ \mathcal{D}_s = & \Phi_s + \frac{1}{2} \mathcal{C}_s \Phi_s^2, \quad \forall s \in S, \end{aligned}$$

## Nested Optimization Framework

$$\text{Timing Opt.: } \mathcal{L}_1 = \max_{\boldsymbol{\omega}} \mathcal{L}_2(\boldsymbol{\omega}), \quad (5a)$$

$$\text{Clock Opt.: } \mathcal{L}_2(\boldsymbol{\omega}) = \max_{\eta} \mathcal{L}_3(\eta, \boldsymbol{\omega}), \quad (5b)$$

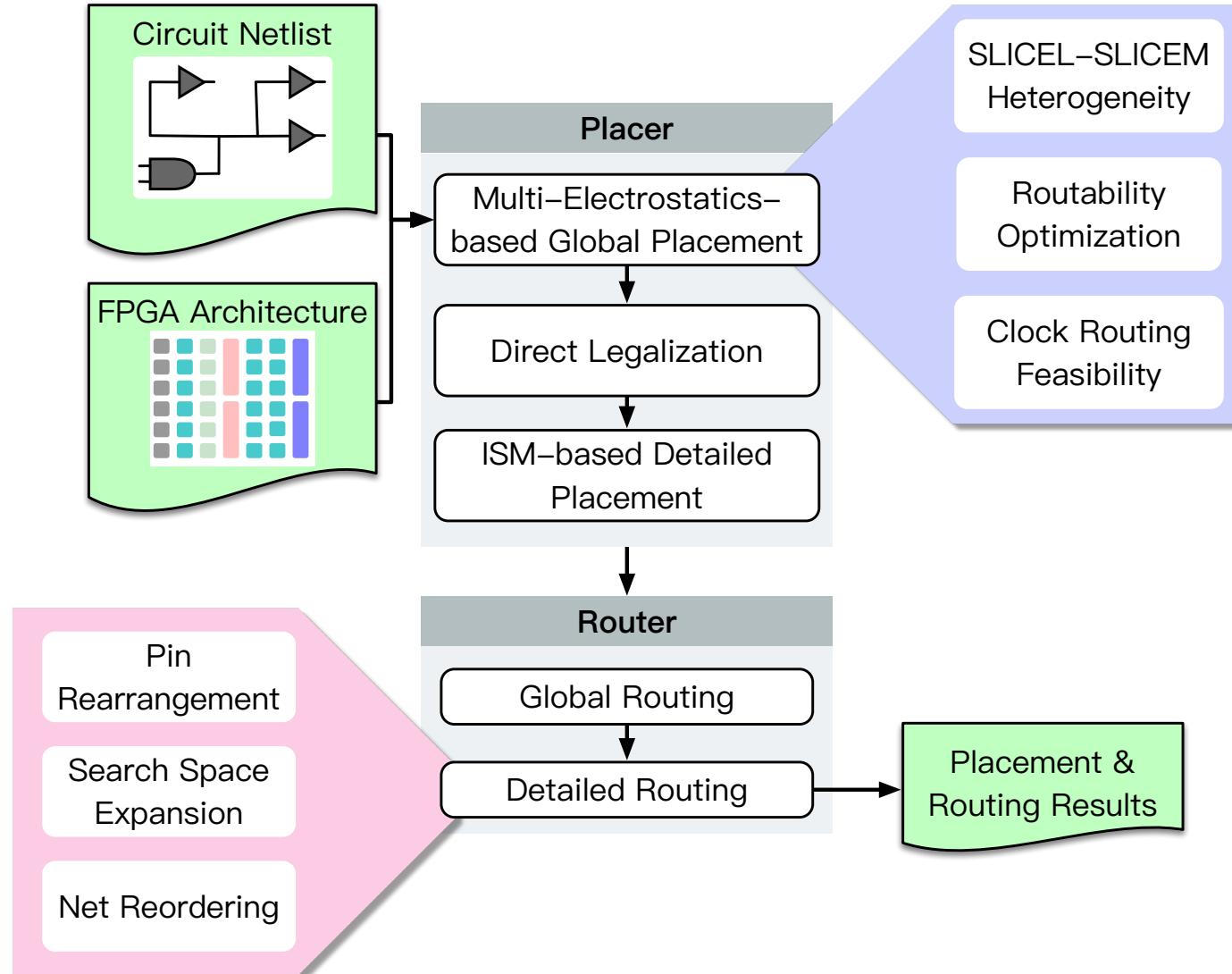
$$\text{Routability Opt.: } \mathcal{L}_3(\eta, \boldsymbol{\omega}) = \max_{\mathcal{A}} \mathcal{L}_4(\mathcal{A}, \eta, \boldsymbol{\omega}), \quad (5c)$$

$$\text{Wirelength Opt.: } \mathcal{L}_4(\mathcal{A}, \eta, \boldsymbol{\omega}) = \max_{\boldsymbol{\lambda}} \mathcal{L}_5(\boldsymbol{\lambda}, \mathcal{A}, \eta, \boldsymbol{\omega}), \quad (5d)$$

$$\text{Subproblem: } \mathcal{L}_5(\boldsymbol{\lambda}, \mathcal{A}, \eta, \boldsymbol{\omega}) = \min_{\mathbf{x}, \mathbf{y}} \mathcal{L}(\mathbf{x}, \mathbf{y}; \boldsymbol{\lambda}, \mathcal{A}, \eta, \boldsymbol{\omega}), \quad (5e)$$

# OpenPARF 1.0: Open-source FPGA Placement and routing Framework

# OpenPARF 1.0 Overview



# The OpenPARF Framework



## Code Components

### ▶ **openparf**

The core placement and routing tool

### ▶ **openparf.ops**

A collection of operators that allow the implementation of various PR algorithms

### ▶ **openparf.placement**

A set of APIs for performing placement tasks

### ▶ **openparf.routing**

A set of APIs for performing routing tasks

### ▶ **openparf.py\_utils**

Provides other utility functions for Python convenience

README.md

## OpenPARF

💡 OpenPARF is an open-source FPGA placement and routing framework build upon the deep learning toolkit [PyTorch](#). It is designed to be flexible, efficient, and extensible.

- [OpenPARF](#)

- [More About OpenPARF](#)
  - [A Multi-Electrostatic-based FPGA P&R Framework](#)
  - [Reference Flow](#)
  - [Demo](#)

- [Prerequisites](#)

- [Build from Source](#)
  - [Install Dependencies](#)
  - [Install Gurobi \(Optional\)](#)
- [Build with Docker](#)
  - [Docker Image](#)
    - [Using pre-built images](#)
    - [Building the image yourself](#)
  - [Running the Docker Image](#)
  - [Entering the Docker Container](#)

- [Build and Install OpenPARF](#)

- [Get the OpenPARF Source](#)
- [Install OpenPARF](#)
- [Adjust Build Options \(Optional\)](#)

- [Getting Started](#)

- [ISPD 2016/2017 Benchmarks](#)
  - [Obtaining Benchmarks](#)
  - [Linking Benchmarks](#)
  - [Running the Benchmarks](#)
  - [Adjust Benchmark Options \(Optional\)](#)
- [More Advanced Usages](#)
  - [Running Benchmarks in Batches](#)
  - [Vivado Flow for Placement Evaluation](#)

- [Resources](#)

- [Releases and Contributing](#)

- [The Team](#)

- [Publications](#)

- [License](#)

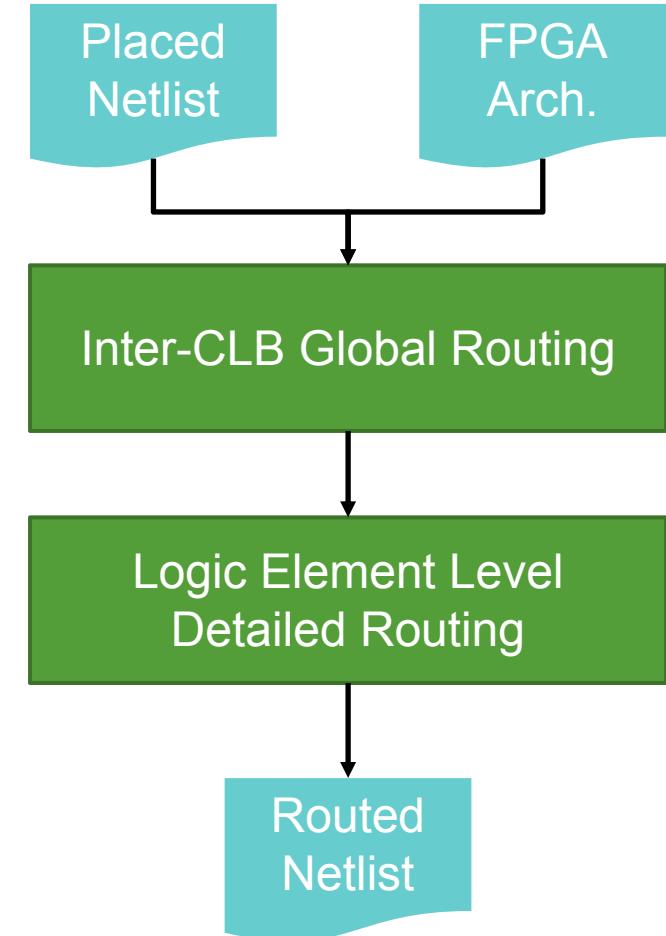
# Highlighted Features

## State-of-the art P&R Algorithms

- ▶ SOTA multi-electrostatics-based global placement
- ▶ SOTA two CLB-level FPGA routing

## More P&R Constraints

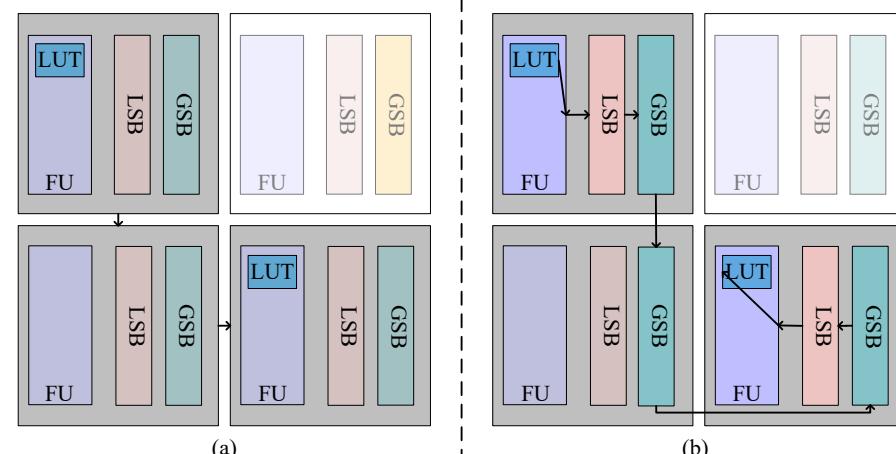
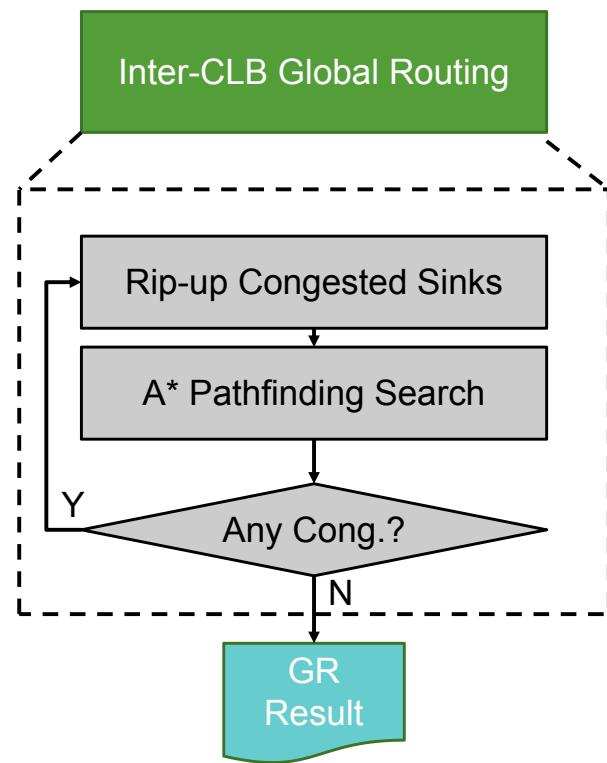
- ▶ SLICEL-SLICEM heterogeneity
- ▶ Routability Optimization
- ▶ Clock Routing Feasibility



# Two-stage CLB-level Routing

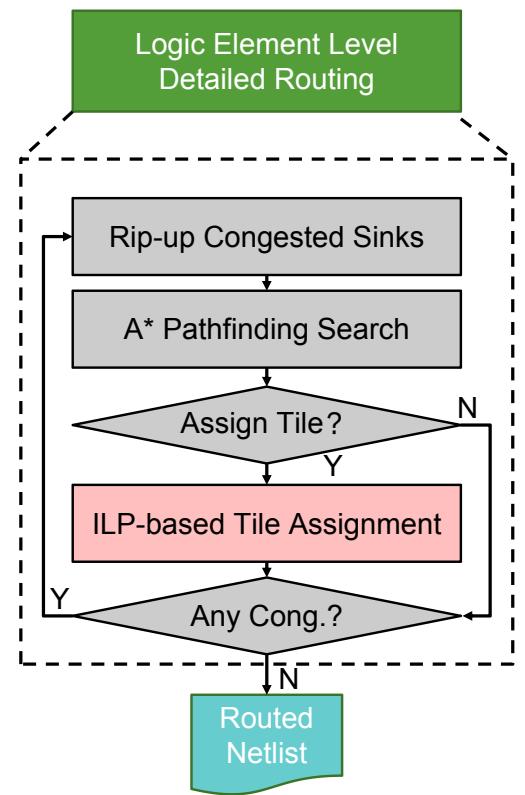
## Inter-CLB level gouting routing

- ▶ Coarse-grained routing graph
- ▶ Provide inter-CLB routing topology



## Logic element level detailed routing

- ▶ Fine-grained routing graph
- ▶ Generate final routing results
- ▶ Proposed ILP-based tile assignment to remedy congestion

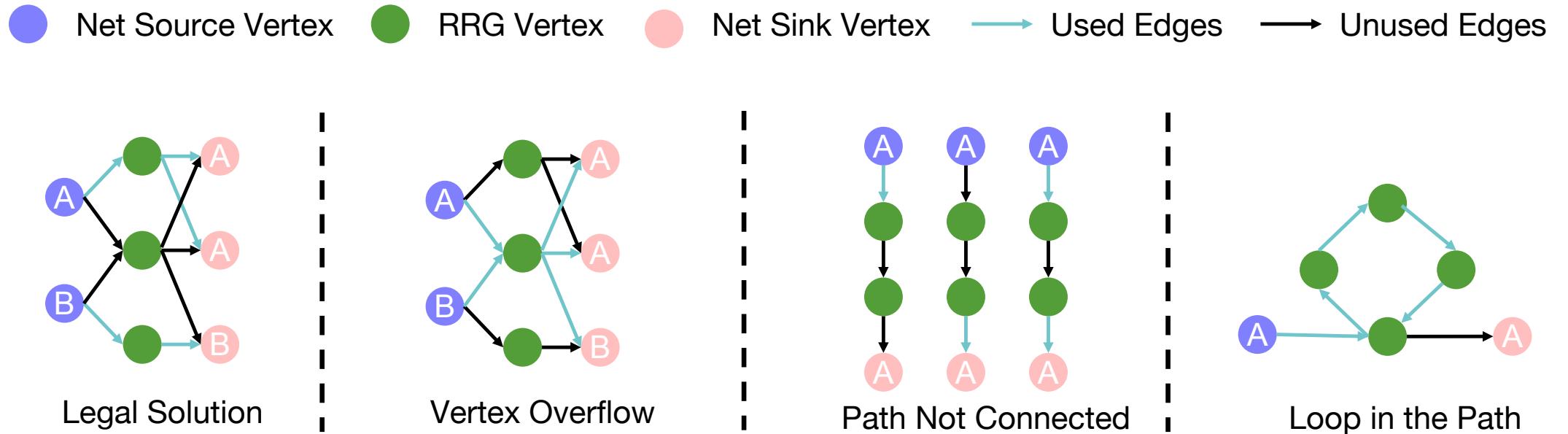


# ILP-based Tile Assignment (I)

## Problem Formulation

Route multiple nets inside a tile and its neighbor tile concurrently

- ▶ No overflow vertices
- ▶ Paths must be connected
- ▶ No loop in the paths



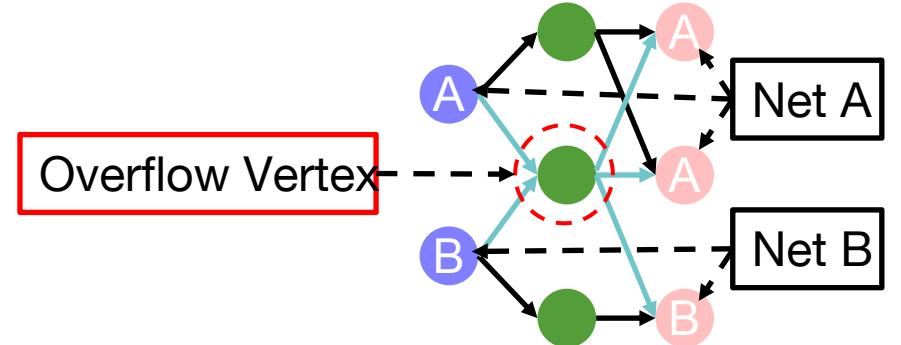
# ILP-based Tile Assignment (II)



## Integer Linear Programming (ILP) Modeling

### 1. No overflowed vertex

$$\sum_{e,j} R_{e,j} \leq \text{cap}(v), e \in \text{FI}(v)$$



# ILP-based Tile Assignment (II)



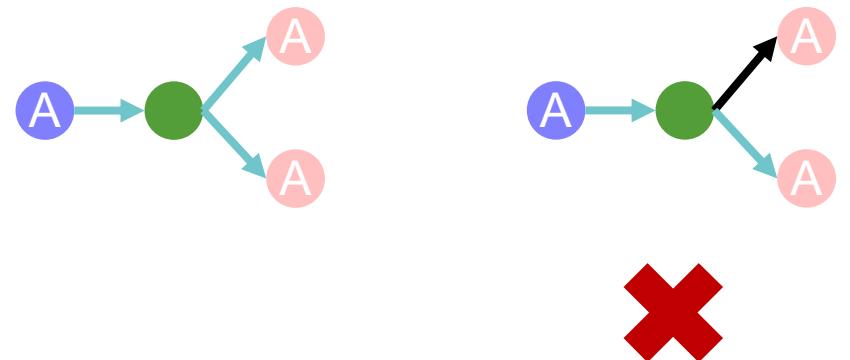
## Integer Linear Programming (ILP) Modeling

1. No overflowed vertex

$$\sum_{e,j} R_{e,j} \leq \text{cap}(v), e \in \text{FI}(v)$$

2. Each sink of each net is routed

$$S_{e,j,k} \leq R_{e,j}, k \in \text{SINK}(j)$$



# ILP-based Tile Assignment (II)



## Integer Linear Programming (ILP) Modeling

1. No overflowed vertex

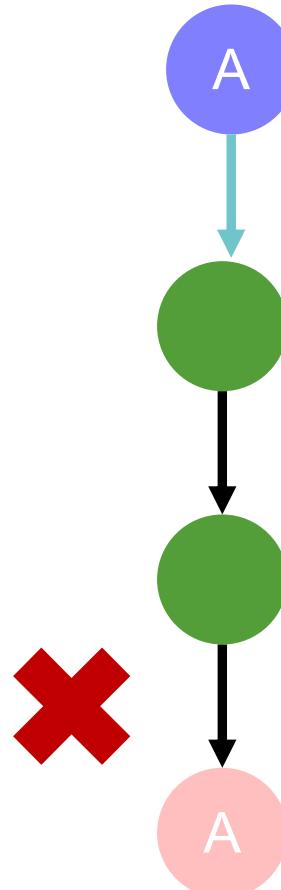
$$\sum_{e,j} R_{e,j} \leq \text{cap}(v), e \in \text{FI}(v)$$

2. Each sink of each net is routed

$$S_{e,j,k} \leq R_{e,j}, k \in \text{SINK}(j)$$

3. The signal is sent from the source pin of each net

$$\sum_{e,j,k} S_{e,j,k} = 1, e \in \text{FO}(v), v = \text{SOURCE}(j), \forall k \in \text{SINK}(j)$$



# ILP-based Tile Assignment (II)



## Integer Linear Programming (ILP) Modeling

1. No overflowed vertex

$$\sum_{e,j} R_{e,j} \leq \text{cap}(v), e \in \text{FI}(v)$$

2. Each sink of each net is routed

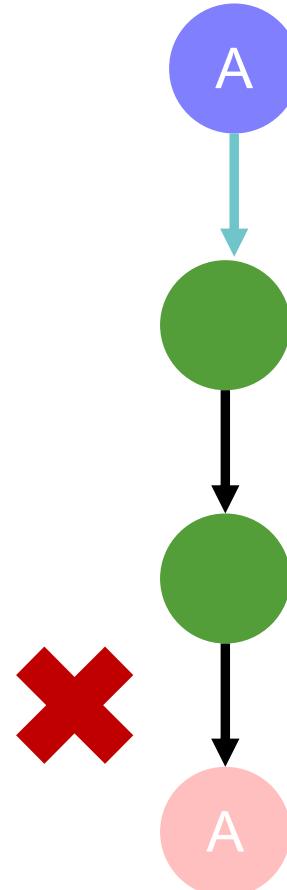
$$S_{e,j,k} \leq R_{e,j}, k \in \text{SINK}(j)$$

3. The signal is sent from the source pin of each net

$$\sum_{e,j,k} S_{e,j,k} = 1, e \in \text{FO}(v), v = \text{SOURCE}(j), \forall k \in \text{SINK}(j)$$

4. The signal is received at each sink pin of each net

$$\sum_{e,j,k} S_{e,j,k} = 1, e \in \text{FI}(v), v = \text{SINK}(j, k)$$



# ILP-based Tile Assignment (II)



## Integer Linear Programming (ILP) Modeling

1. No overflowed vertex

$$\sum_{e,j} R_{e,j} \leq \text{cap}(v), e \in \text{FI}(v)$$

2. Each sink of each net is routed

$$S_{e,j,k} \leq R_{e,j}, k \in \text{SINK}(j)$$

3. The signal is sent from the source pin of each net

$$\sum_{e,j,k} S_{e,j,k} = 1, e \in \text{FO}(v), v = \text{SOURCE}(j), \forall k \in \text{SINK}(j)$$

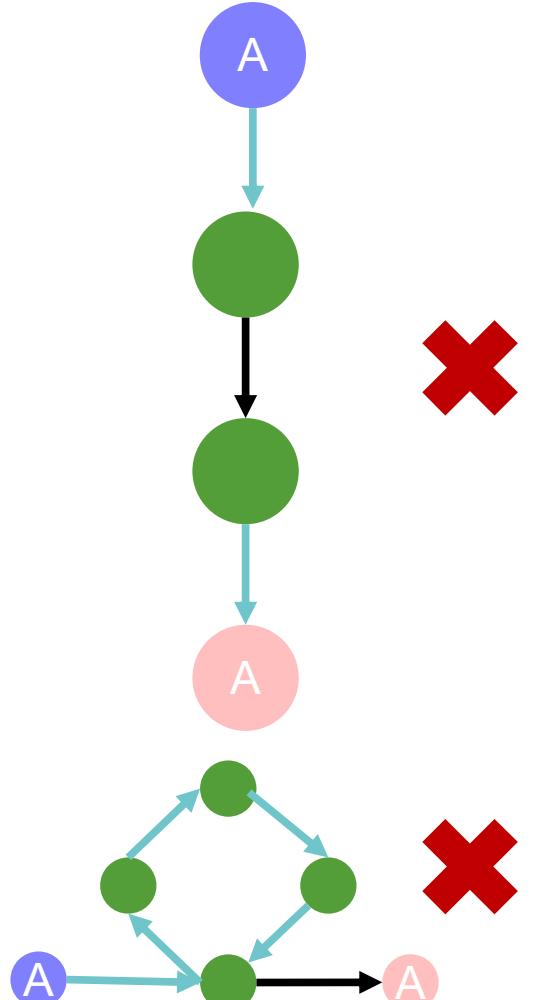
4. The signal is received at each sink pin of each net

$$\sum_{e,j,k} S_{e,j,k} = 1, e \in \text{FI}(v), v = \text{SINK}(j, k)$$

5. There is a path from source pin to each sink pin and no loop

$$\sum_{e_{in}} S_{e_{in},j,k} = \sum_{e_{out}} S_{e_{out},j,k},$$

$$e_{in} \in \text{FI}(v), e_{out} \in \text{FO}(v), v \neq \text{SOURCE}(j), v \notin \text{SINK}(j)$$



# Experimental Results

# Experimental Setup (I)



## Implementation

- ▶ C++ & Python
- ▶ Build upon Pytorch for agile gradient computation

## Machine

- ▶ Intel(R) Xeon(R) Gold 6230 CPUs (2.10 GHz, 40 cores)
- ▶ 512GB RAM
- ▶ One NVIDIA RTX 2080Ti GPU

# Experimental Setup (II)

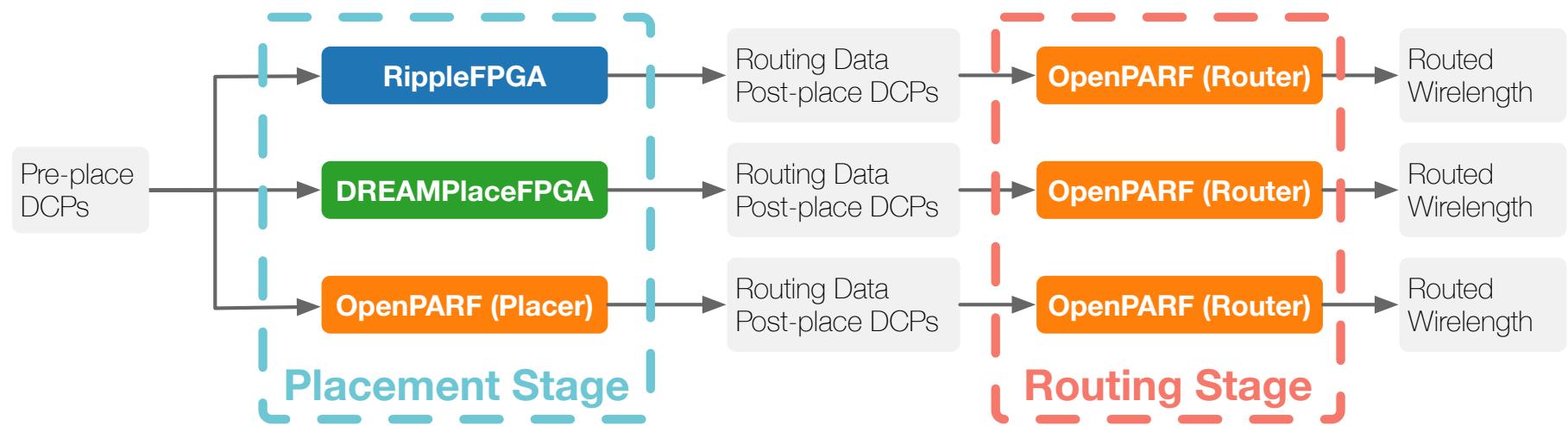
## Benchmark Suite

- ▶ ISPD 2016 Routability-driven FPGA Placement Contest [Yang+, ISPD'16]
- ▶ ISPD 2017 Clock-aware FPGA Placement Contest [Yang+, ISPD'17]
- ▶ SLICEL-SLICEM Structure-Aware Industrial Benchmarks

## Placers for Comparison

- ▶ RippleFPGA [Chen+, TCAD'18]
- ▶ DREAMPlaceFPGA [Rajarathnam+, ISPD'23]

## Evaluation Flow

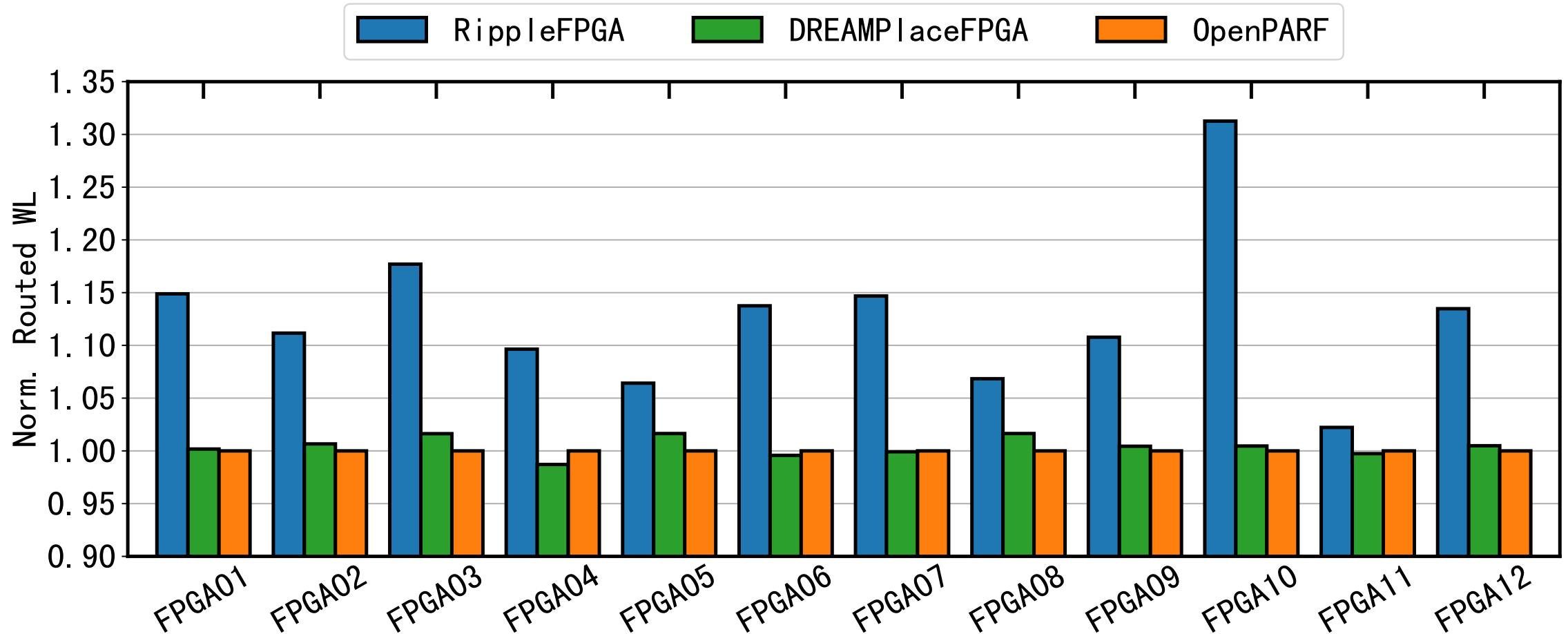


# ISPD 2016 Routability-Driven FPGA Placement Contest



## Routed Wirelength Comparison on ISPD2016

- ▶ 12.7% better than RippleFPGA
- ▶ 0.4% better than DREAMPlaceFPGA



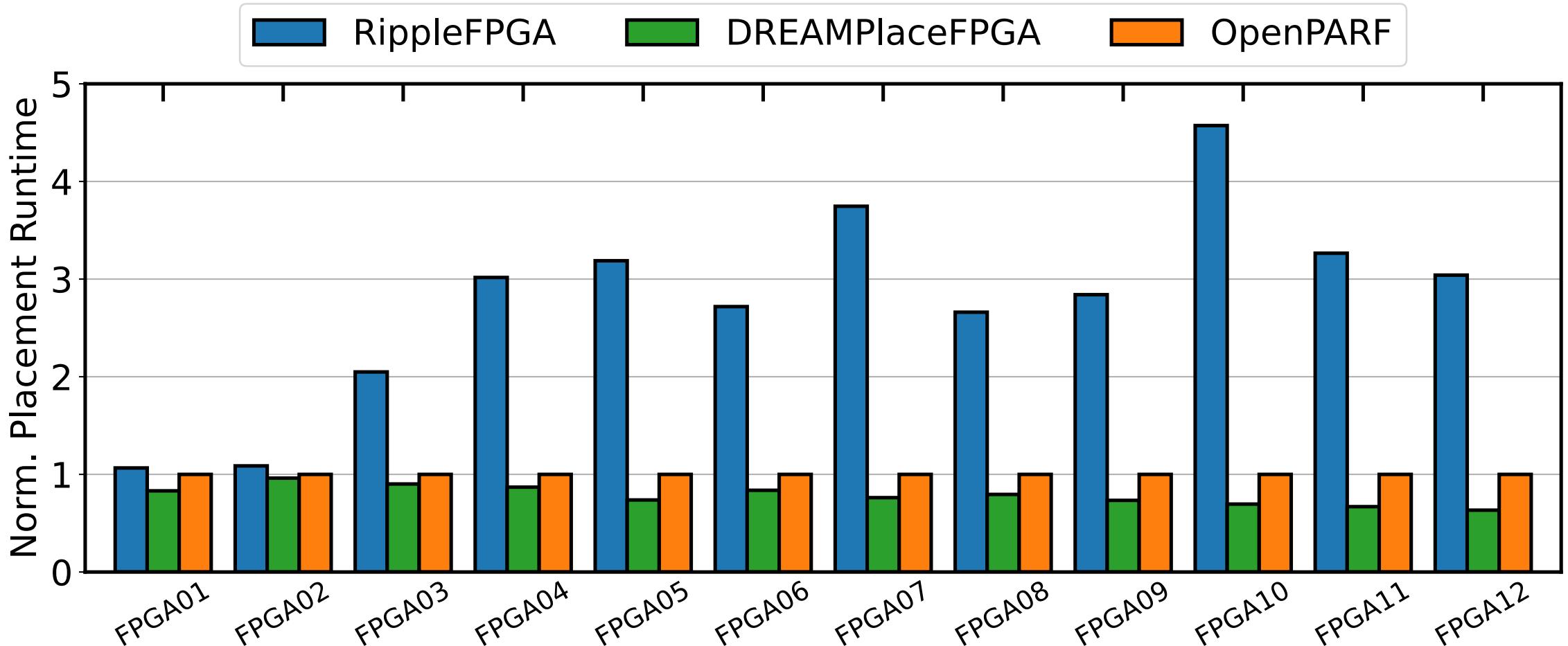
# ISPD 2016 Routability-Driven FPGA Placement Contest



## Placement Runtime Comparison on ISPD2016

► 2.771x faster than RippleFPGA

► 1.272x slower than DREAMPlaceFPGA

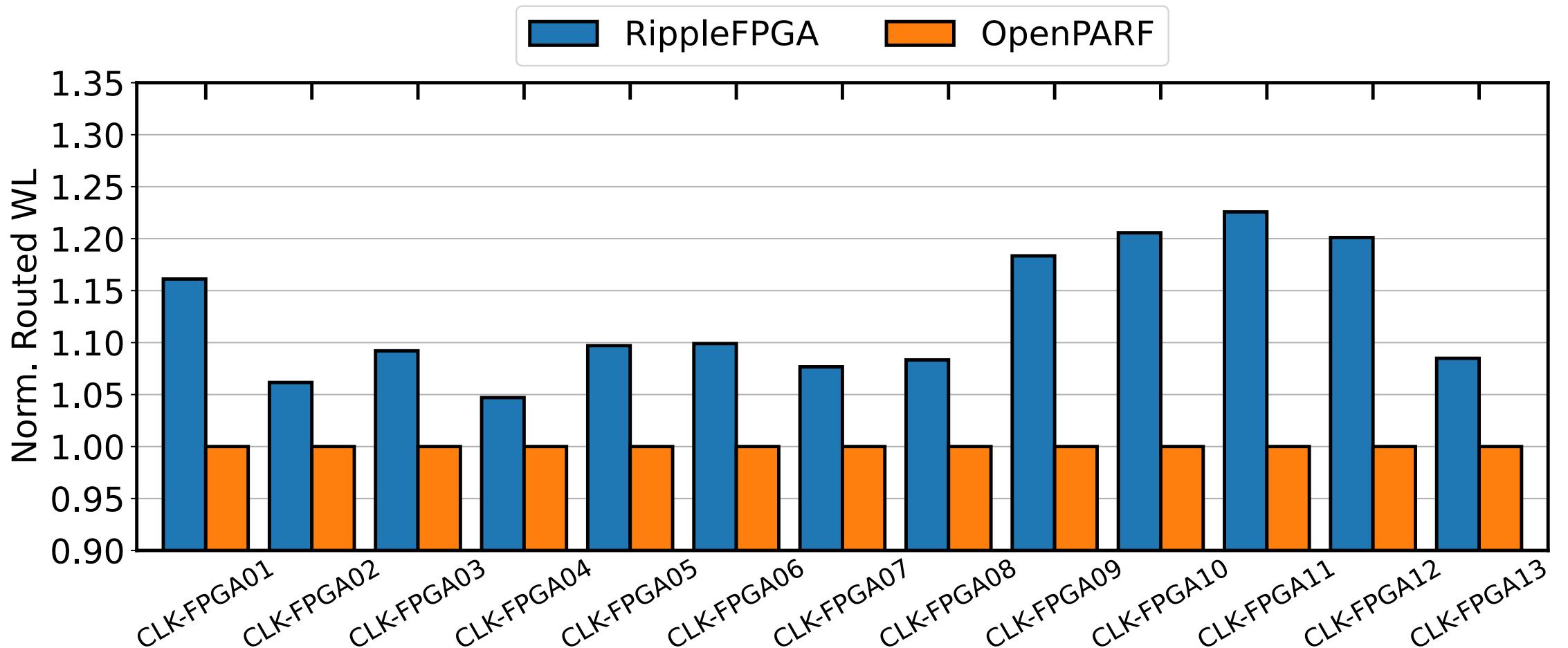


# ISPD 2016 Routability-Driven FPGA Placement Contest



## Routed Wirelength Comparison on ISPD2017

- ▶ 12.8% better than RippleFPGA

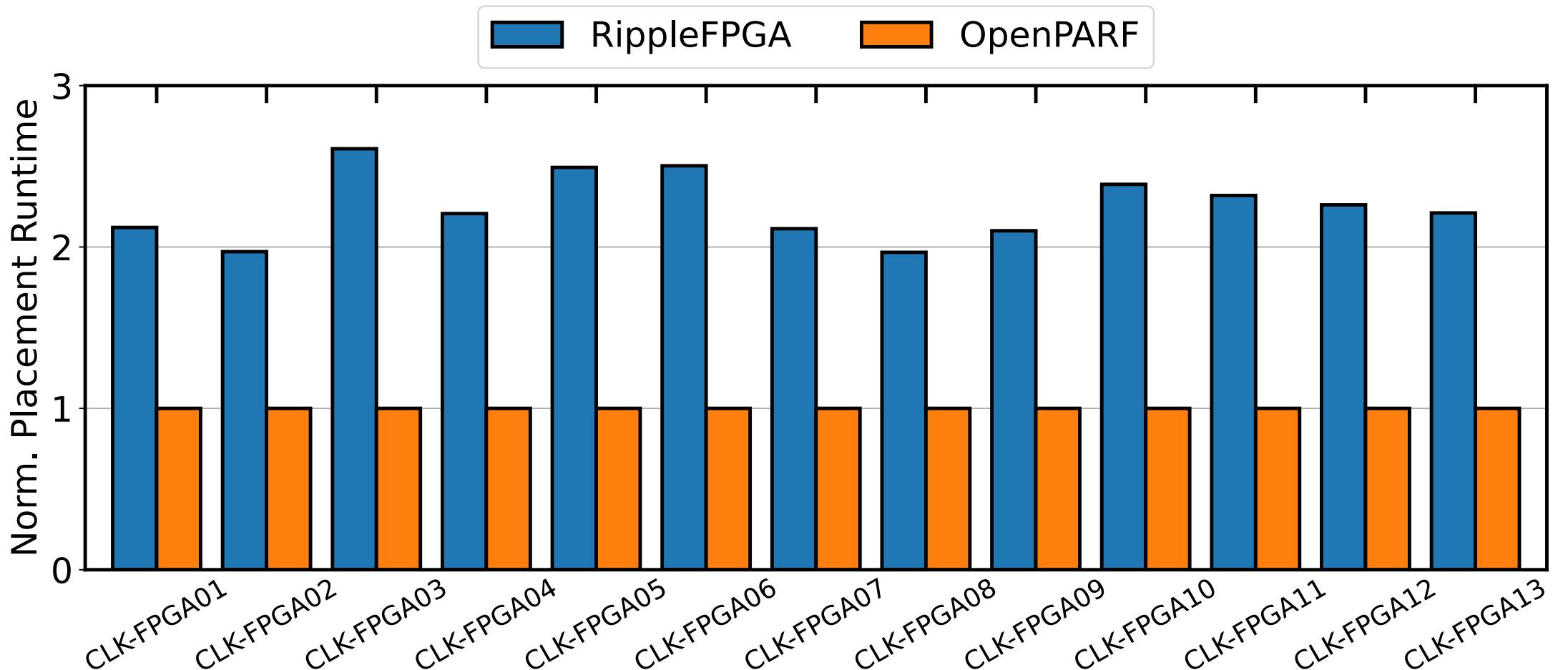


# ISPD 2016 Routability-Driven FPGA Placement Contest



## Placement Runtime Comparison on ISPD2017

- ▶ 2.251x faster than RippleFPGA



# The OpenPARF 3.0 Framework: Fence Regions, Cascaded Macros

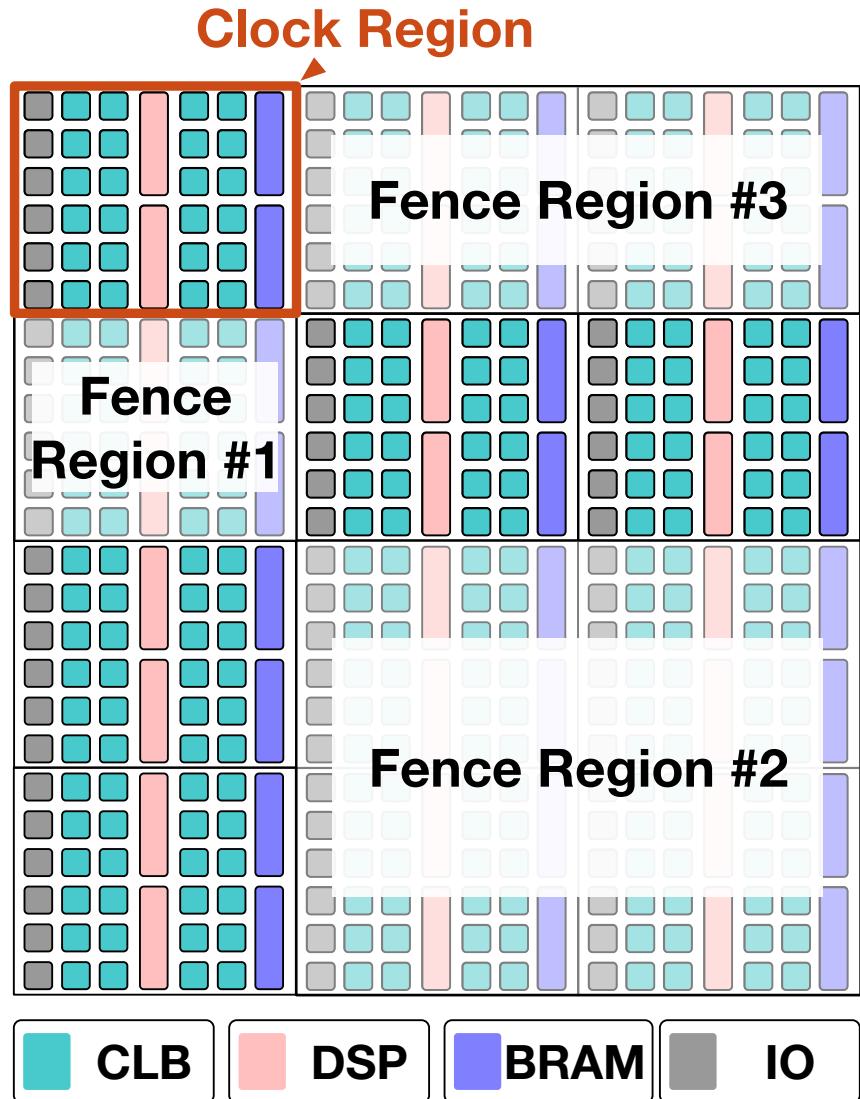
# Various Cell Types & Fence Region Constraint

## Various Cell Types

- ▶ Heterogenous resources & columnar distribution
- ▶ LUT & FF (standard cell, placed in CLB)
- ▶ DSP, BRAM (Macro)
- ▶ IO, etc.
- ▶ Clock Region

## Fence Region Constraint

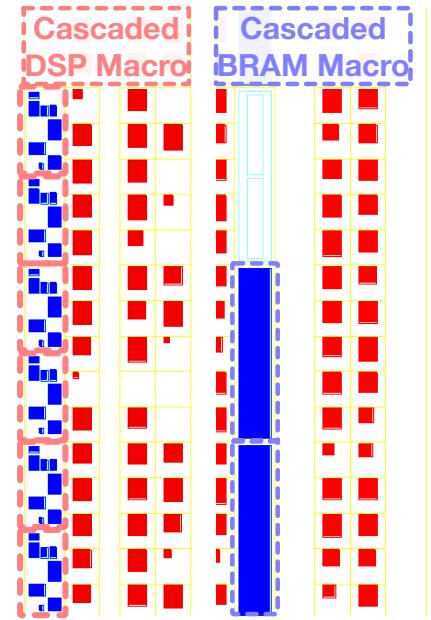
- ▶ Cells s.t. the constraint must be placed within the region
- ▶ Stem from clock regions or can be user-defined



# Cascaded Macros

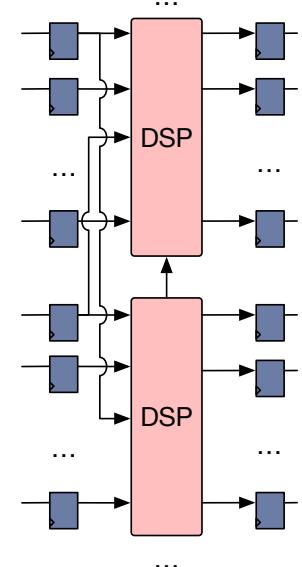
## Cascaded Macro (CM)

- ▶ Cells must be placed in continuous columnated sites in a prescribed sequence
- ▶ Constraints on two macro types
- ▶ Cascaded DSP Macros
- ▶ Cascaded BRAM macros



## Cascaded Macro Group

- ▶ *definition:* cascaded macros + the standard cells closely connected to them
- ▶ e.g., I/O signals of cascaded DSP macros are often connected to numerous FFs (~100)



**FPGA macros significantly impact the overall routability of the design!**

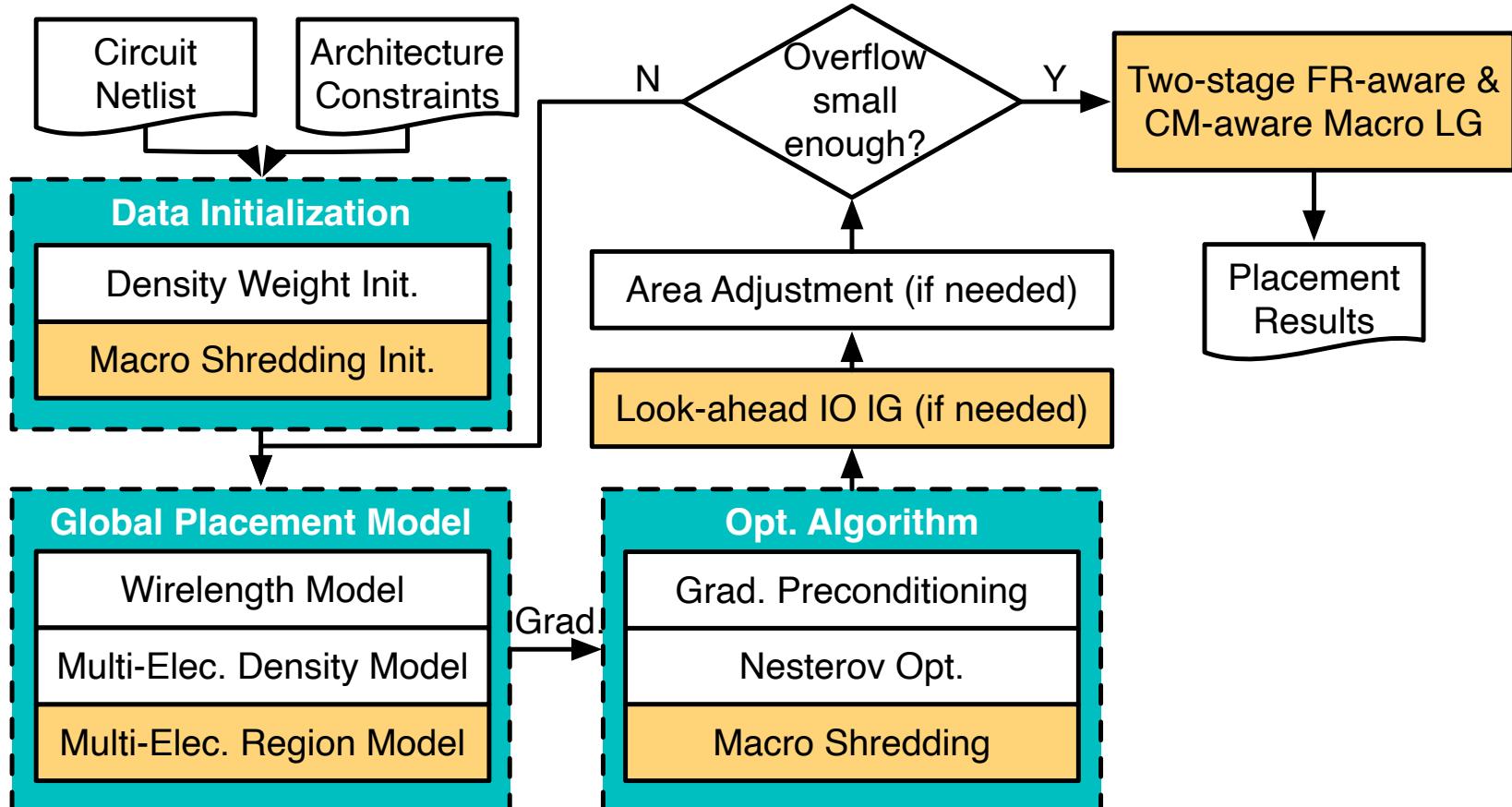
# Contribution of OpenPARF 3.0



We propose **OpenPARF 3.0**, a robust FPGA macro placer considering both fence region constraints and cascaded macro constraints.

- ▶ We propose a novel **multi-electrostatics region model** accompanied by a footprint compression technique to effectively handle the discontinuity in the fence region.
- ▶ We propose a **divergence-aware density weight scheduling** scheme which can effectively addresses robustness issues.
- ▶ We propose a **cascaded macro shredding technique** to address the imbalance issue of cascaded macro sizes.
- ▶ Experiments demonstrate that OpenPARF 3.0 can achieve **13.3-49.1%** overall score improvement as well as **1.81-3.18×** speedup compared with Xilinx Vivado 2021.1 and other SOTA academic FPGA macro placers.

# Overall Flow of OpenPARF 3.0



# Global Placement Model



Resource Type Set  $\mathcal{T}$

$$\mathcal{T} = \{LUT, FF, DSP, BRAM, IO\}.$$

Multi-Electrostatics-based Global Placement Model

consider fence region constraints and cascaded macro constraints,

$$\begin{aligned} & \min_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}) = \tilde{W}(\mathbf{x}, \mathbf{y}) + \sum_{s \in S} \lambda_s \mathcal{D}_s, \\ & \text{s.t. } \mathcal{D}_s = \Phi_s + \frac{\mu}{2} \mathcal{P}_s \Phi_s^2, \forall s \in S^D \cup \mathcal{S}^R, \\ & \quad \text{Cascaded macro Constraints,} \end{aligned}$$

$S^D$  : the multi-electrostatics density model as OpenPARF [Mai+, ASICON'23],

$$S_D = \{S_{LUT}^D, S_{FF}^D, S_{DSP}^D, S_{BRAM}^D, S_{IO}^D\}.$$

Density weight preconditioner  $\mathcal{P}_s$

$$\mathcal{P}_s = 1 / \Phi_s^{(0)}$$

Multi-Electrostatics-based Region Model

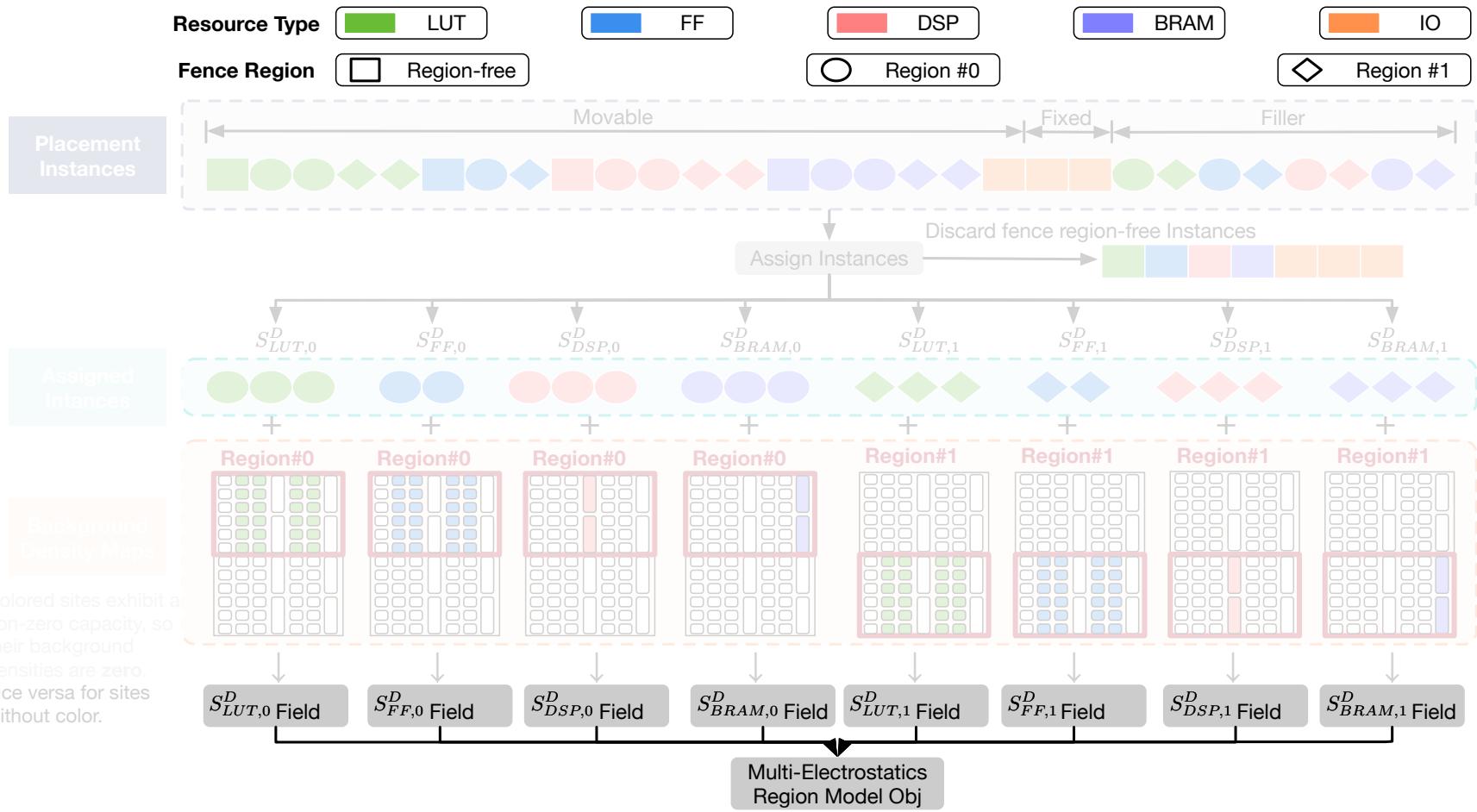
$\mathcal{S}_R$  : our proposed multi-electrostatics region model to resolve fence region constraints.

# Multi-Electrostatics Region Model (I)

## Multi-Electrostatics-based Region Model

For each resource type  $t \in \mathcal{T}$  within each fence region  $f \in \mathcal{F}$ , we construct an electrostatics system  $S_{t,f}^R$ .

In other words, we construct  $|\mathcal{S}^R| = |\mathcal{T}||\mathcal{F}|$  more potential energy terms as the density objective.



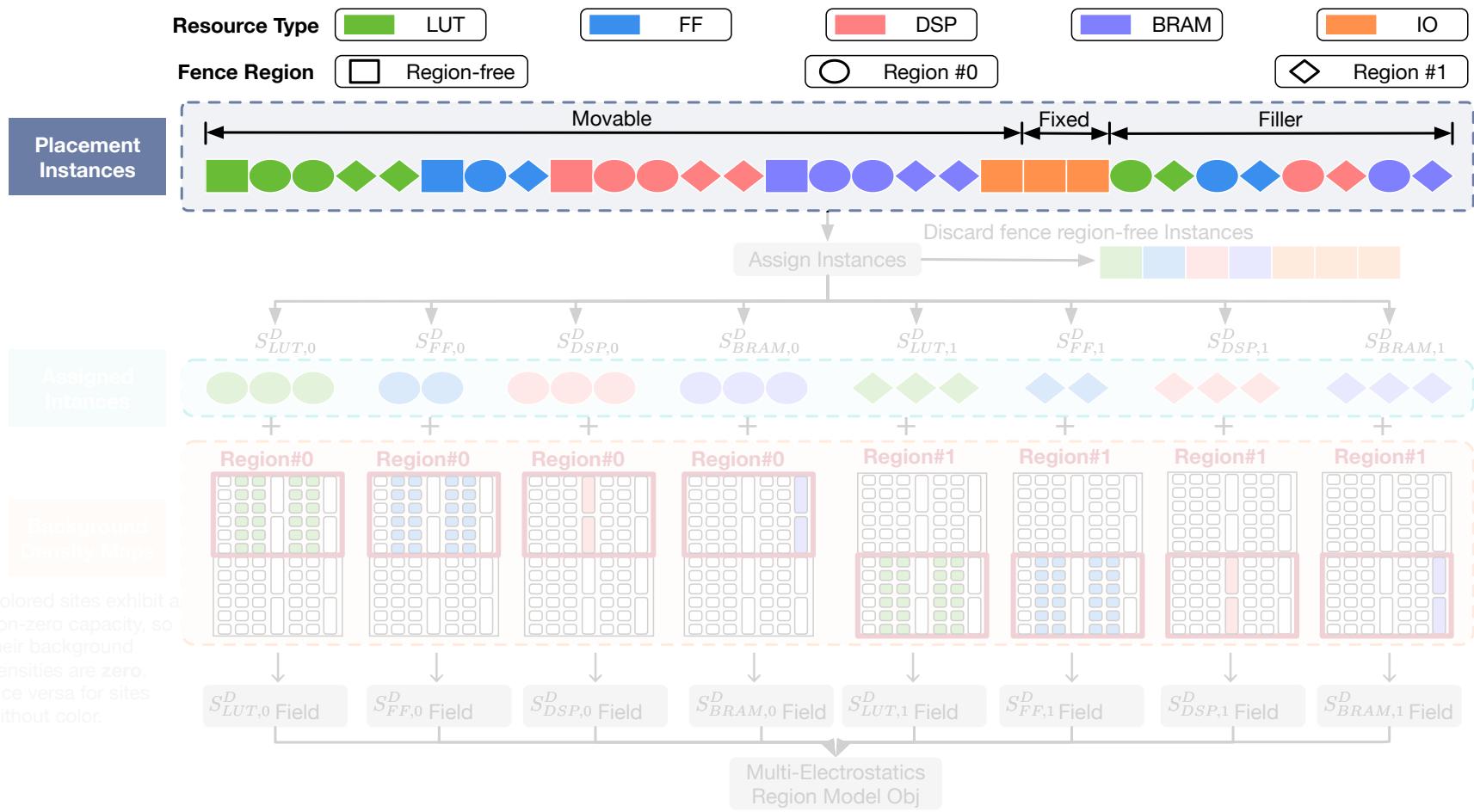
# Multi-Electrostatics Region Model (II)



## 1. Placement Instances

All placement instances are firstly categorized as movable instances, fixed instances, and fillers.

Fillers are artificially created for the purpose of target density control for each electrostatics system.

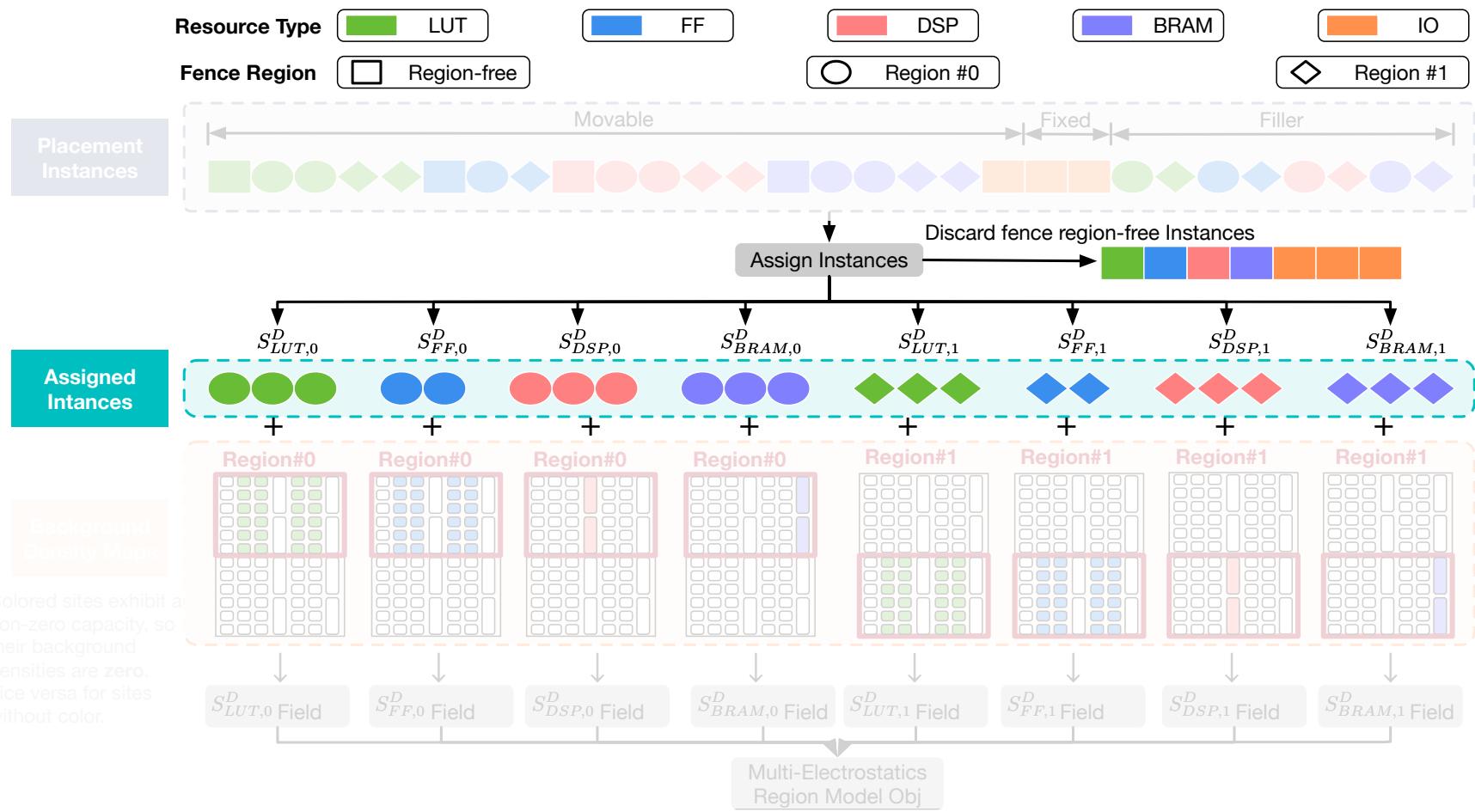


# Multi-Electrostatics Region Model (III)

## 2. Assigned Instances

The placement instances are then assigned to electrostatics systems based on 1) their resource type and 2) the constraints imposed by the fence region they subject to.

Region-free Instances will be discarded.

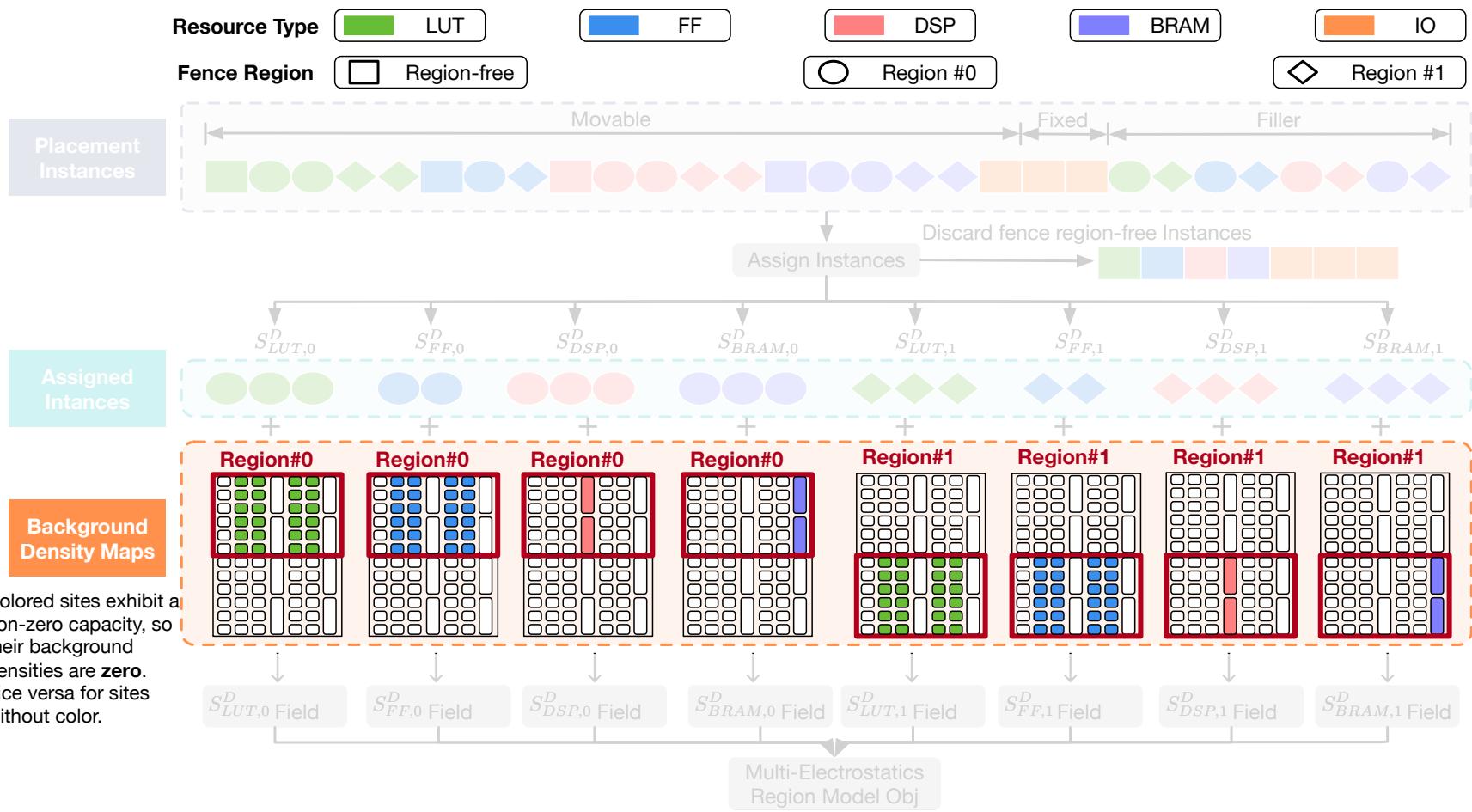


# Multi-Electrostatics Region Model (IV)

## 3. Background Density Maps

For sites with capacity, the background density is set to zero, indicating the permissibility of placing instances on those sites.

For sites without capacity, the background density is set to the non-zero target density.



# Footprint Compression Technique

Memory allocation is required to store the sizes of placement instances under different electrostatics systems.

Space Complexity of previous approach:  $O(\#nodes \times \#eSystems)$

## Three Observations

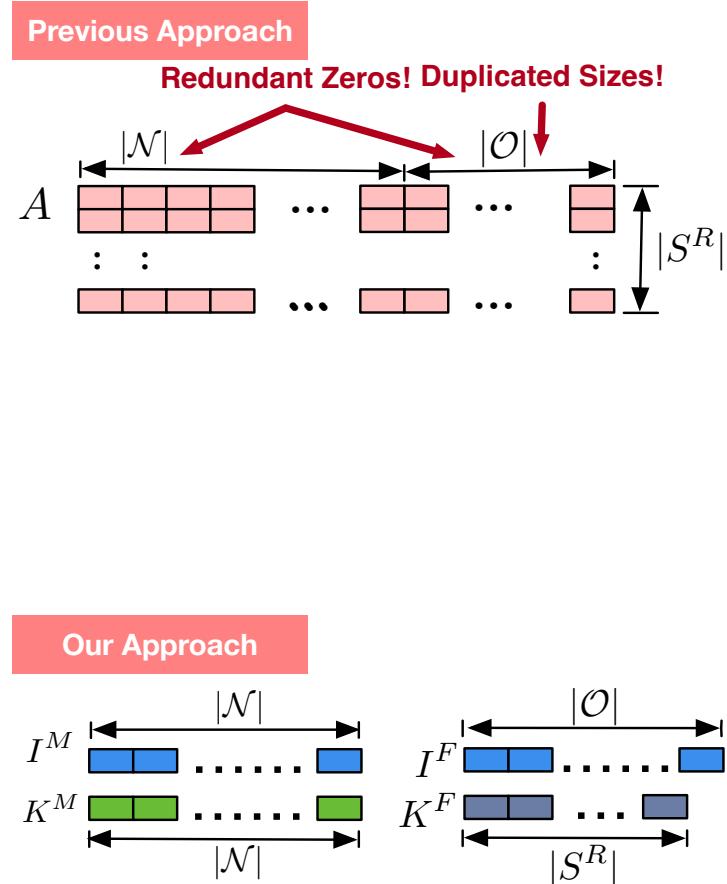
- ▶ Each movable and fixed instance subject to at most one fence region constraint.
- ▶ Each filler is exclusively associated with one electrostatics system.
- ▶ Filler sizes within the same electrostatics system are consistent.

## Footprint Compression Technique

- ▶ For movable and fixed instances
  - $I_i^M$  : the electrostatics system to which instance  $i$  is assigned
  - $K_i^M$  : the size of the instance
- ▶ For fillers
  - $I_i^F$  : the electrostatics system to which instance  $i$  is assigned
  - $K_j^F$  : the size of the filler within electrostatics system

Space Complexity of our approach:

$$O(\#nodes + \#eSystems)$$



Reduce the GPU memory usage from 21.4GB to 3.9GB on case Design\_142 with 22 fence regions!

# Divergence-aware Density Weight Scheduling

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{y}} L(\mathbf{x}, \mathbf{y}) &= \tilde{W}(\mathbf{x}, \mathbf{y}) + \sum_{s \in S} \lambda_s \mathcal{D}_s, \\ \text{s.t. } \mathcal{D}_s &= \Phi_s + \frac{\mu}{2} \mathcal{P}_s \Phi_s^2, \forall s \in S_D \cup S_R \end{aligned}$$

The update strategy for density weight  $\boldsymbol{\lambda}$  also plays a crucial role in the result quality and optimization stability.  
**Two additional auxiliary variables**

$$\begin{aligned} w_s^{(t+1)} &= \sum_{i \in \mathcal{N}_s} |\partial \widetilde{W}^{(t)} / \partial x_i|_1, \quad \forall s \in S \\ d_s^{(t+1)} &= |\nabla \mathcal{D}_s^{(t)}|_1, \quad \forall s \in S \end{aligned}$$

where  $\mathcal{N}_s$  represents all the nodes within the electrostatics system  $s$ .

## Divergence-aware weight $\boldsymbol{\theta}$

$\mathbf{u}^{(t)}$  and  $\mathbf{v}^{(t)}$  follow the annotation from [DREAMPlace 3.0, Gu+, ICCAD'20]

$$\begin{aligned} \boldsymbol{\theta}^{(t+1)} &= \max \left( 1, \mathbf{d}^{(t+1)} / \mathbf{w}^{(t+1)} \right) / \boldsymbol{\lambda}^{(t)} \\ \boldsymbol{\lambda}^{(t+1)} &= \min(\mathbf{u}^{(t+1)} \odot \mathbf{v}^{(t+1)}, \gamma / \boldsymbol{\theta}^{(t+1)}) \end{aligned}$$

When the optimization comes to the final stage, the large density gradient ( $|\mathbf{d}| \gg |\mathbf{w}|$ ) is likely to cause the optimization to diverge.

$\boldsymbol{\theta}$  can effectively govern the growth rate of  $\lambda_s$  not to surpass a upper bound  $\frac{\gamma}{d_s^{(t+1)} / w_s^{(t+1)}}$ .

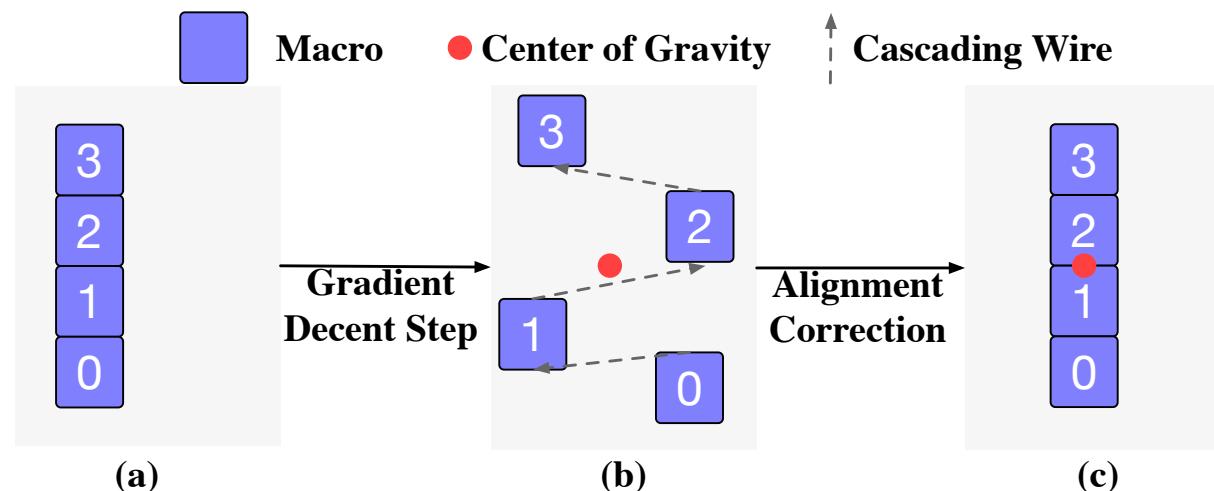
# Cascaded Macro Shredding Technique

## Pesudo Net Weighting within Cascaded Macro Group

Enlarge the weight of nets within the cascaded macro groups by a factor of two.

### Macro Shredding

1. substitute large cascaded structures with multiple individual macros.
2. shred the cascaded macros into individual macros and update the placement
3. At the end of each iteration, arrange the shredded macros in a columnar shape based on the horizontal coordinates of their center of gravity.



# Experimental Results

# Experimental Setup (I)



## Implementation

- ▶ C++ & Python
- ▶ Build upon OpenPARF [Mai+, ASICON'23] for agile development with GPU acceleration

## Machine

- ▶ Intel(R) Xeon(R) Silver 4210R CPU (2.40 GHz, 10 cores)
- ▶ 512GB RAM
- ▶ One NVIDIA RTX 2080Ti GPU

## Benchmark Suite

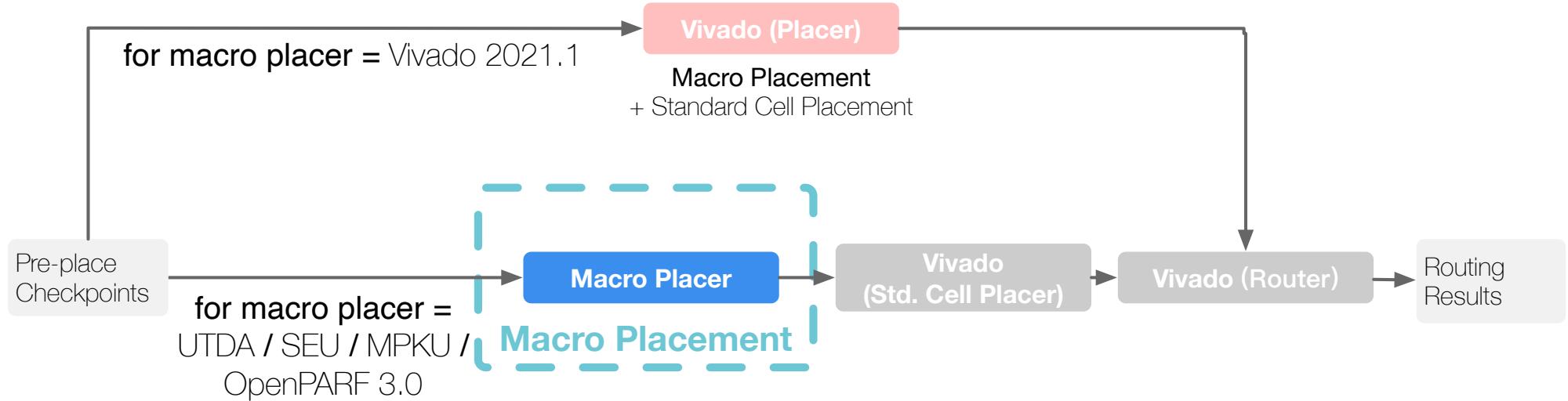
- ▶ MLCAD 2023 FPGA Macro Placement Contest [Bustany+, MLCAD'23]

## Macro Placers for Comparison

- ▶ Vivado 2021.1: a representative FPGA placement commercial tool
- ▶ UTDA, SEU, and MPKU: top three winners in MLCAD 2023 FPGA Macro Placement Contest [Bustany+, MLCAD'23]

# Experimental Setup (II)

## Evaluation Flow



## Evaluation Metrics

Metrics	Indication
<i>Score</i>	Overall evaluation of placement runtime, routing runtime, and routability [Bustany+, MLCAD'23]
<i>Routability</i>	Short, long, and global congestion
<i>#ripup</i>	Detailed routing overflow
$RT_{mp1}$	Macro placement runtime

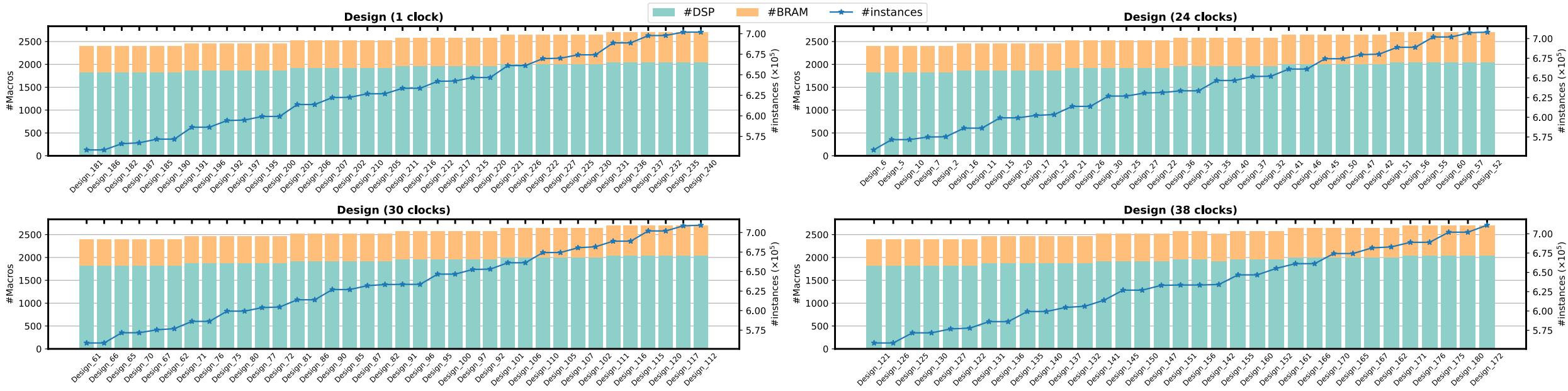
# Experimental Setup (III)

## Benchmark Statistics

- ▶ 16nm single-die UltraScale+ xcvu3p
- ▶ 140 synthetically generated designs
- ▶ Varying levels of difficulty
  - #clocks (1, 24, 30, 38)
  - Rent's Exponent (0.65, 0.67, 0.7, 0.72)

Designs	Statistics
Number	140
#Instances	558K-711K
#DSP+#BRAM	2.4K-2.7K
LUT (%)	70%-84%
FF (%)	38%-47%
BRAM (%)	80%-90%
DSP (%)	80%-90%
Rent <sup>5</sup>	0.65-0.72
#Regions	0-22
#Instances within Regions	0-285K
Instances within Regions (%)	0%-44.29%
Cascaded DSP Macros Size	{2, 5, 7, 10, 60} ×
Cascaded BRAM Macros Size	{2, 5, 7, 10, 30} ×

## Benchmark Overview

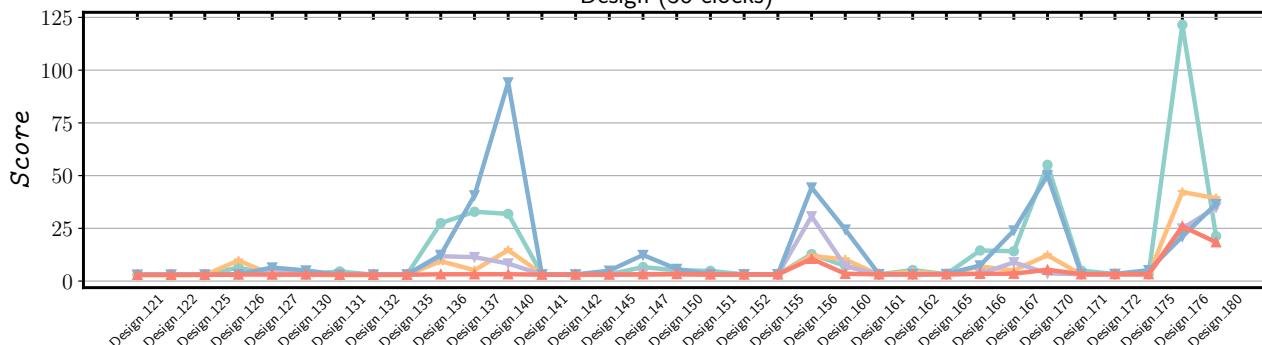
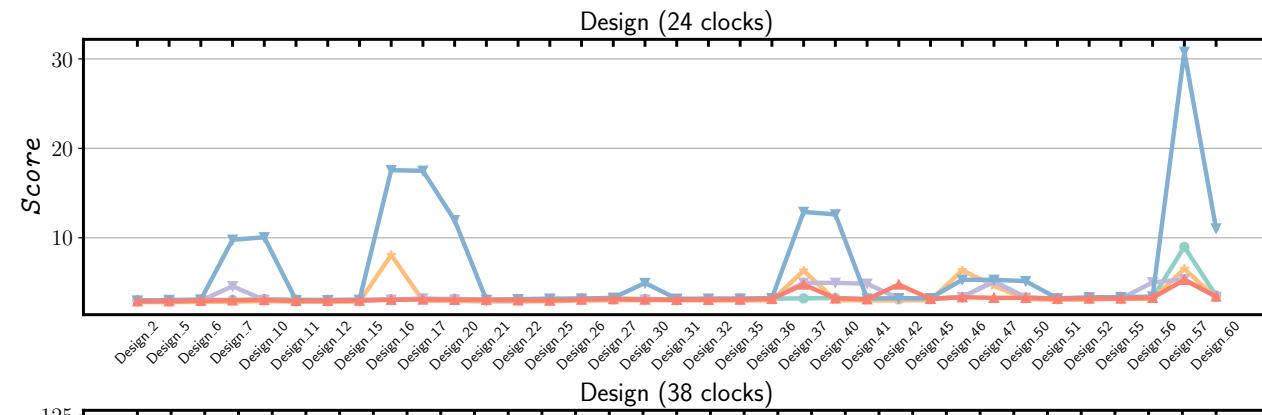
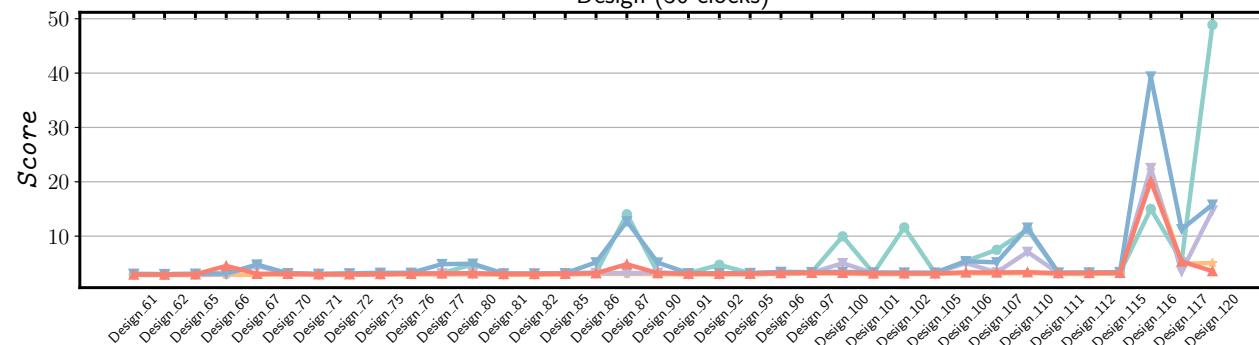
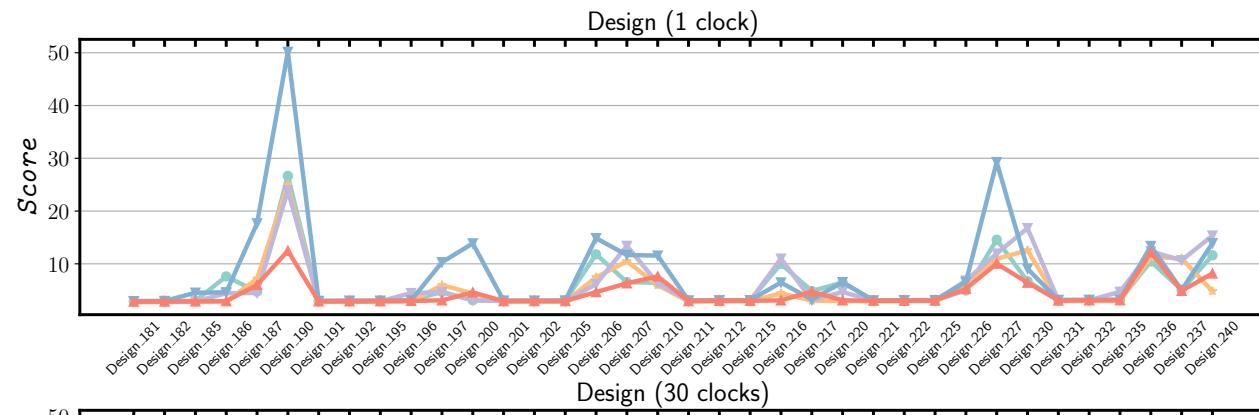


# Comparison with State-of-the-Art Placers

## Overall Score Comparison on MLCAD 2023

- ▶ 27.8% better than Vivado 2021.1
- ▶ 13.3% better than SEU

- ▶ 6.9% better than UTDA
- ▶ 49.1% better than MPKU

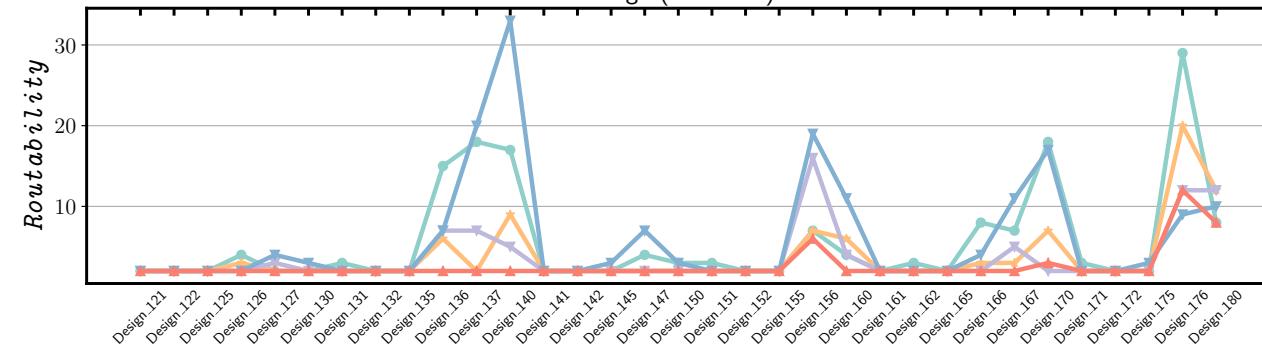
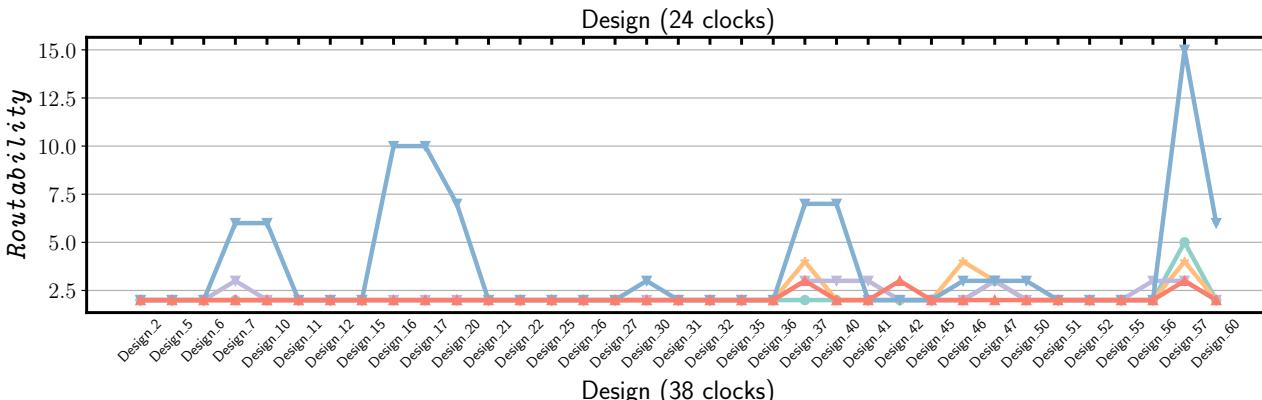
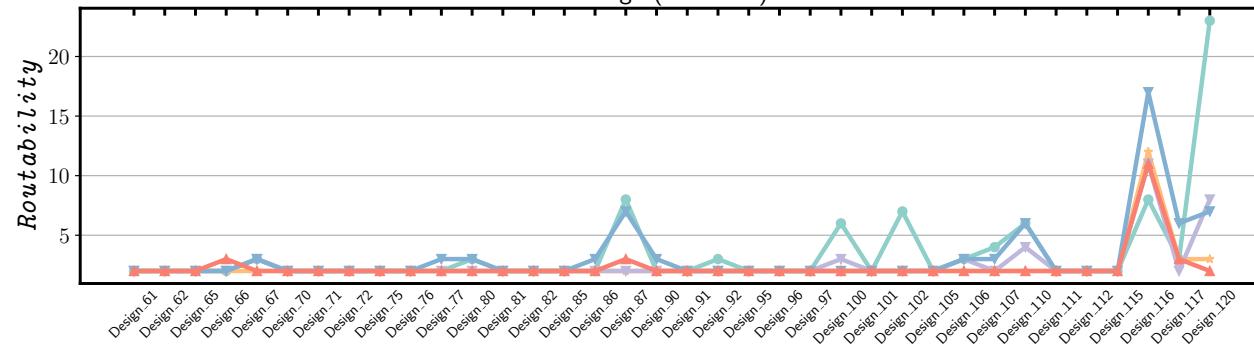
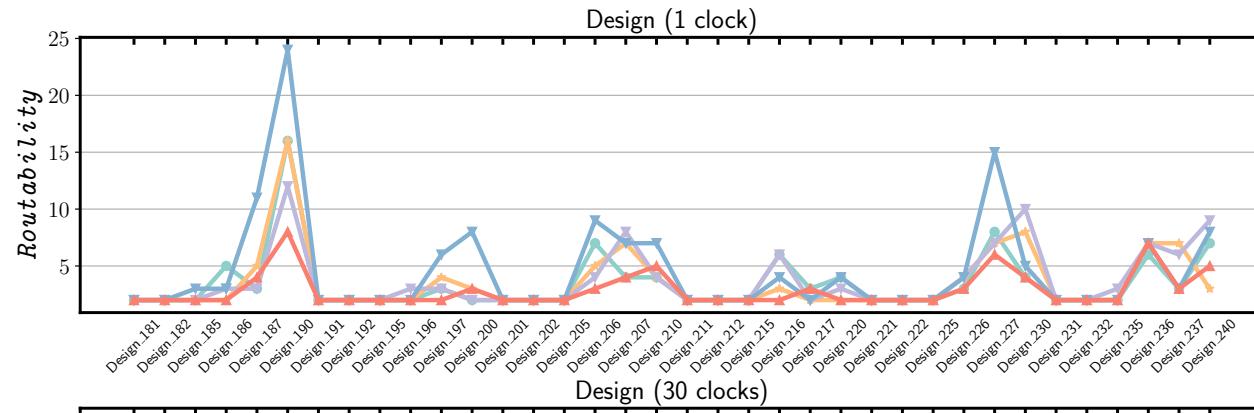


# Comparison with State-of-the-Art Placers

## Routability Comparison on MLCAD 2023

- ▶ 22.8% better than Vivado 2021.1
- ▶ 12.1% better than SEU

- ▶ 8.7% better than UTDA
- ▶ 39.1% better than MPKU

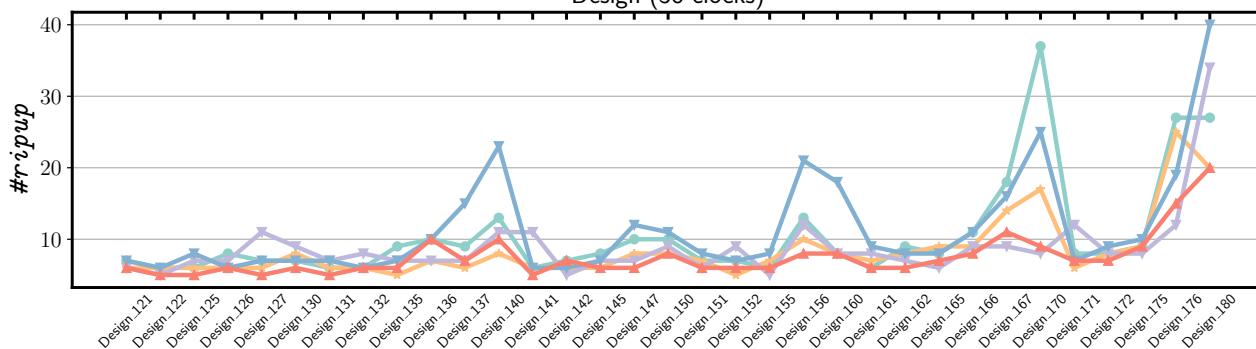
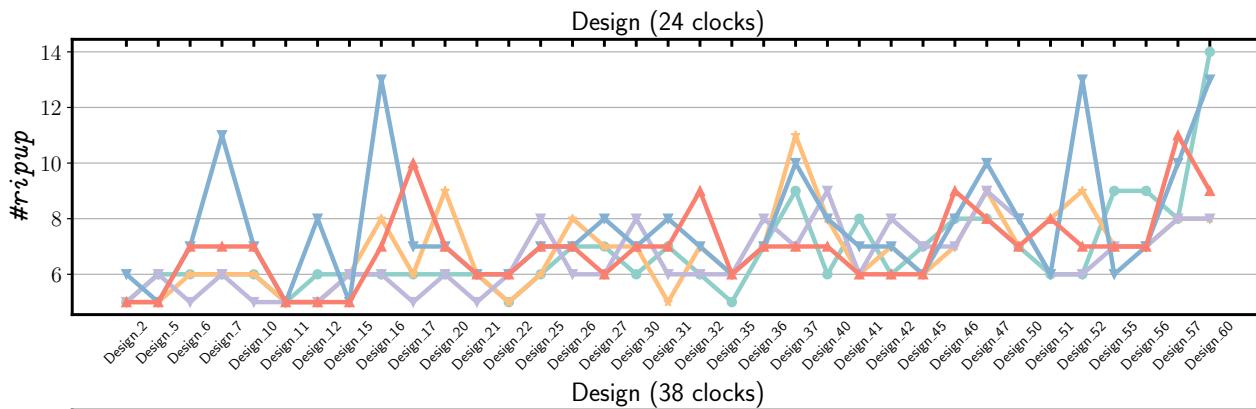
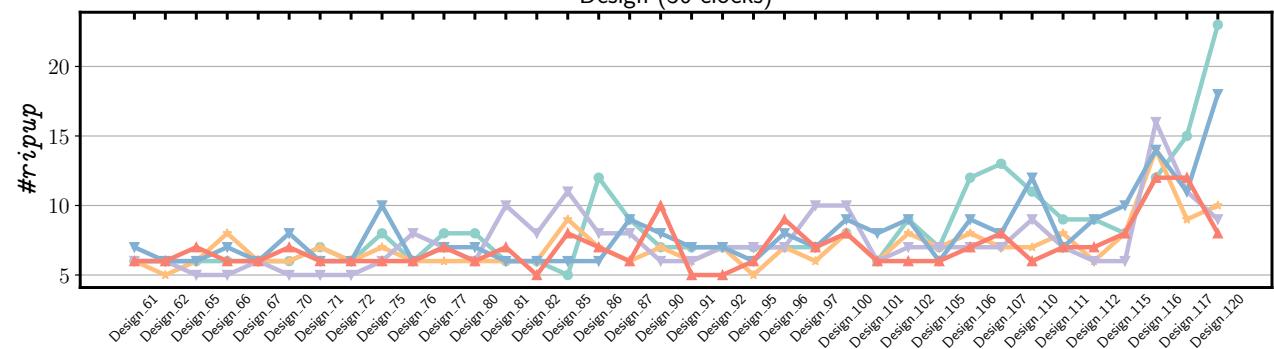
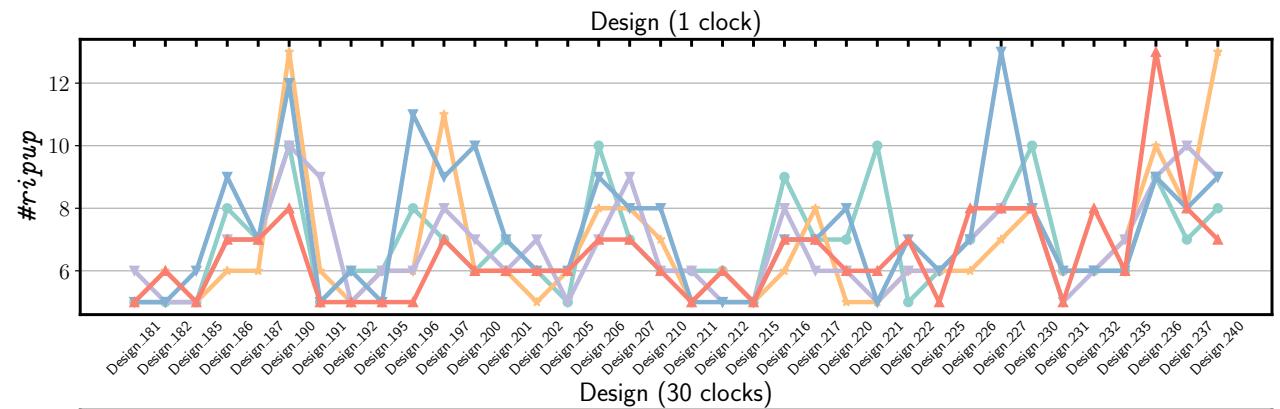


# Comparison with State-of-the-Art Placers

## Overall $\#ripup$ (detailed routing congestion) Comparison on MLCAD 2023

- ▶ 10.8% better than Vivado 2021.1
- ▶ 4.0% better than SEU

- ▶ 2.7% better than UTDA
- ▶ 16.8% better than MPKU

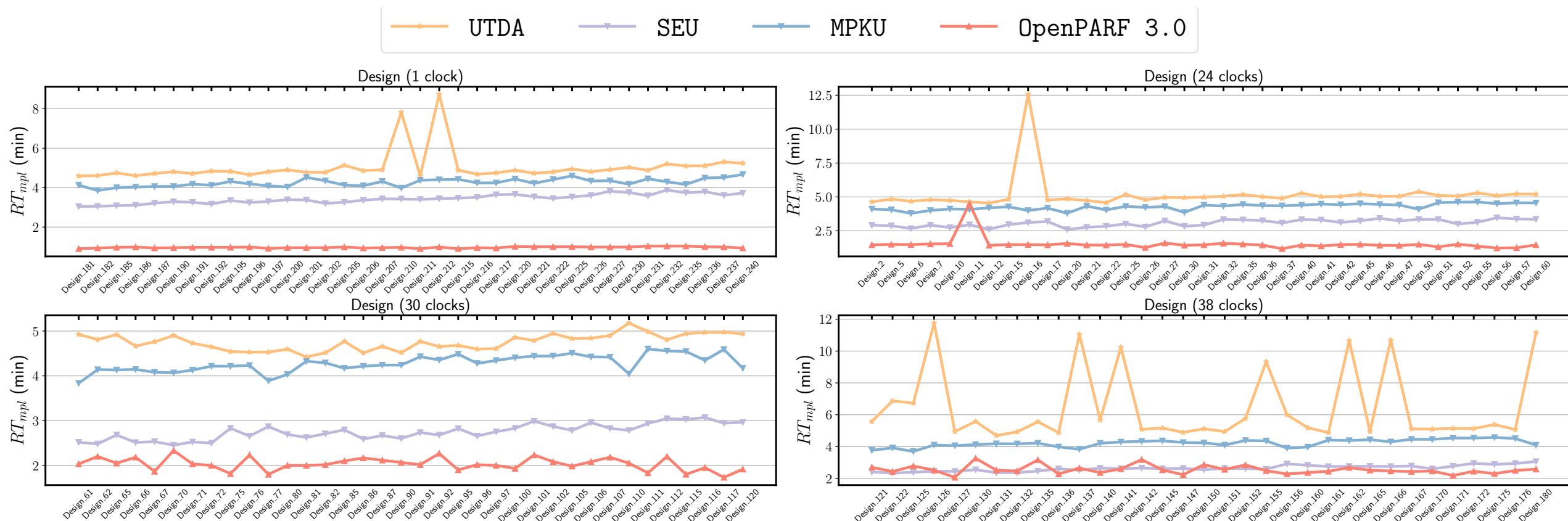


# Comparison with State-of-the-Art Placers

## Macro Placement Runtime $RT_{mpl}$ Comparison on MLCAD 2023

- ▶ **3.180X faster** UTDA
- ▶ **2.599X faster than** MPKU

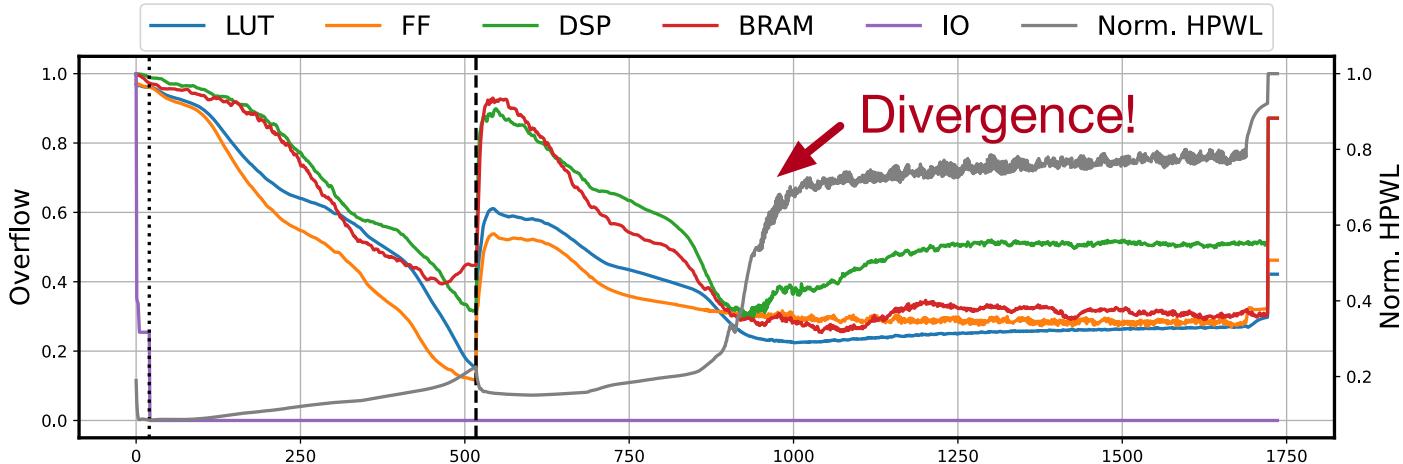
- ▶ **1.808X faster than** SEU



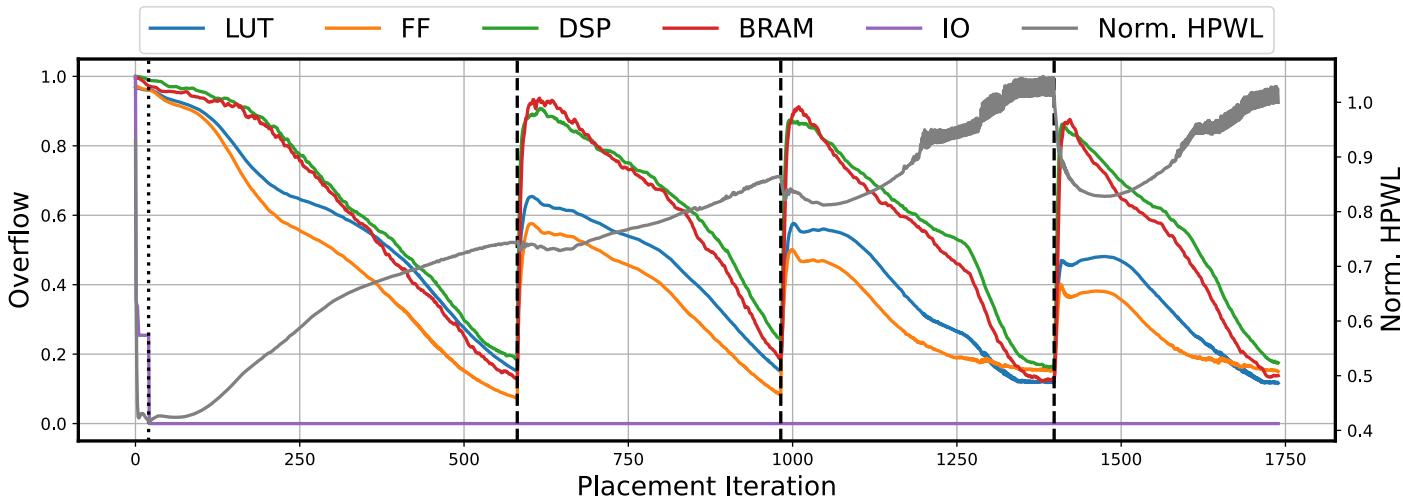
# Robustness Evaluation

## Overflows and HPWL Comparison on the Design\_156

- ▶ The density weight updating method in [Gu+, ICCAD'20]



- ▶ Our proposed divergence-aware density weight scheduling



**THANK YOU!**