

# **ECE 5745 Complex Digital ASIC Design**

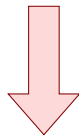
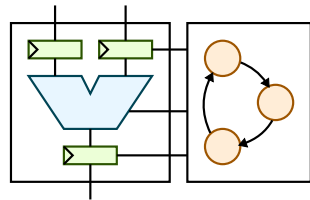
## **Topic 13: Physical Design Automation Algorithms**

Christopher Batten

School of Electrical and Computer Engineering  
Cornell University

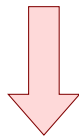
<http://www.csl.cornell.edu/courses/ece5745>

# Part 3: CAD Algorithms



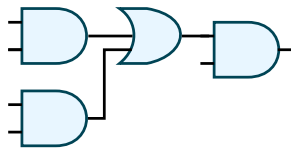
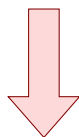
$$x = a'bc + a'bc'$$

$$y = b'c' + ab' + ac$$



$$x = a'b$$

$$y = b'c' + ac$$

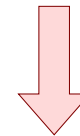
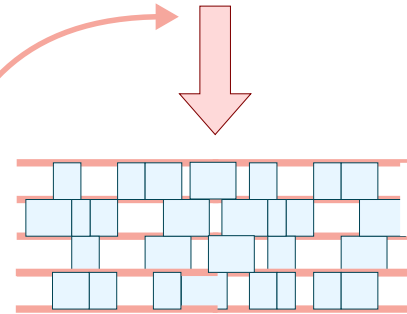


RTL to Logic  
Synthesis

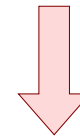
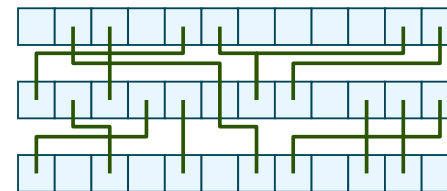
Technology  
Independent  
Synthesis

Technology  
Dependent  
Synthesis

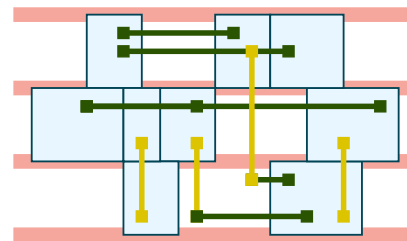
**Placement**



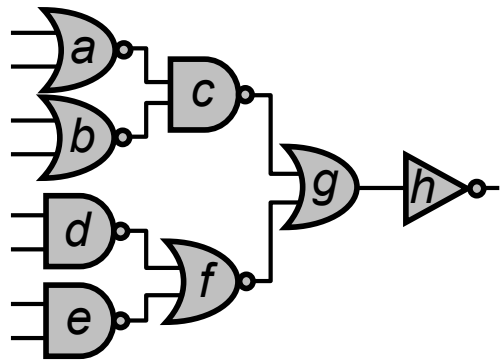
Global  
Routing



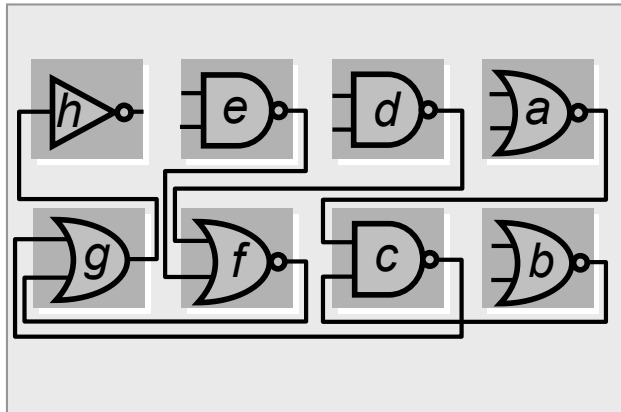
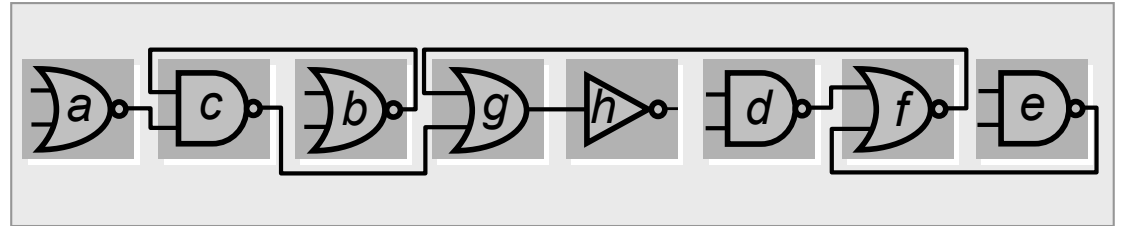
Detailed  
Routing



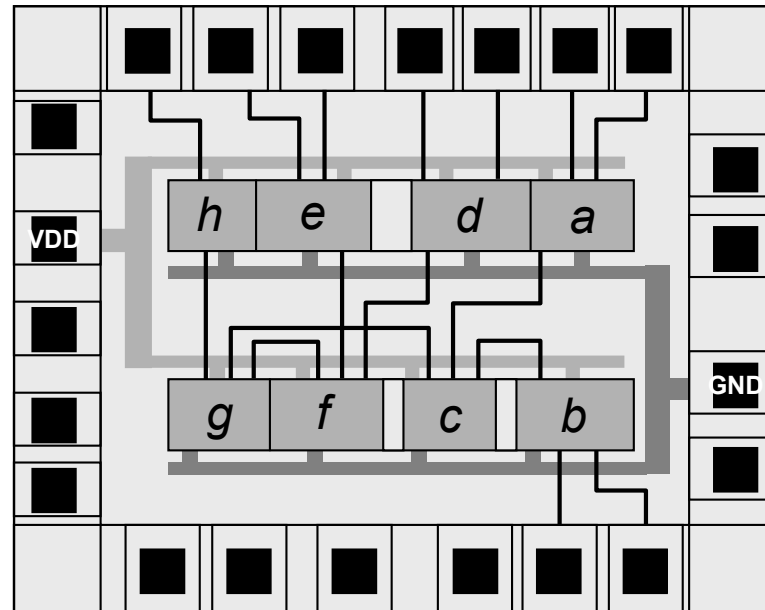
# Placement



Linear Placement



2D Placement

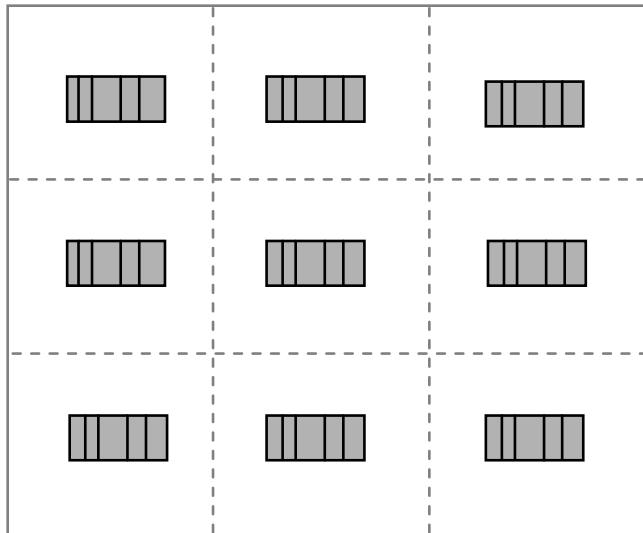


Placement and Routing with Standard Cells

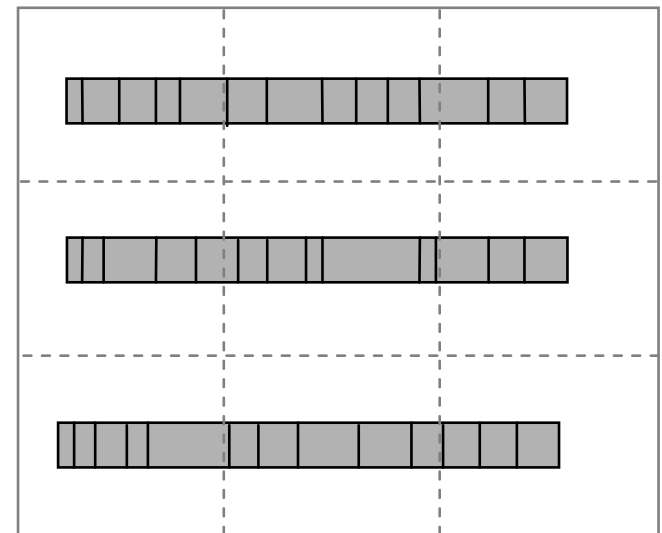
Adapted from [Khang'11]

# Global vs. Detailed Placement

Global Placement



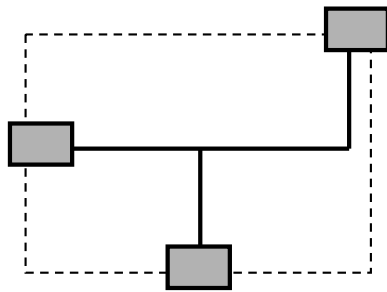
Detailed Placement



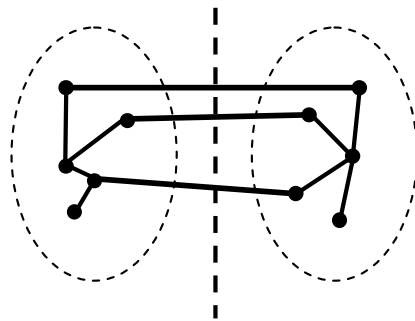
Adapted from [Khang'11]

# Placement Objective Functions

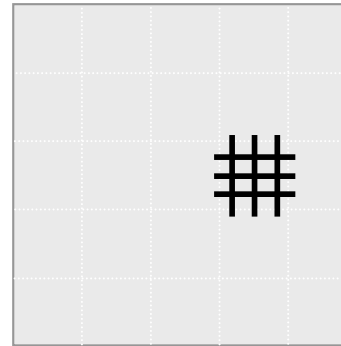
Total Wirelength



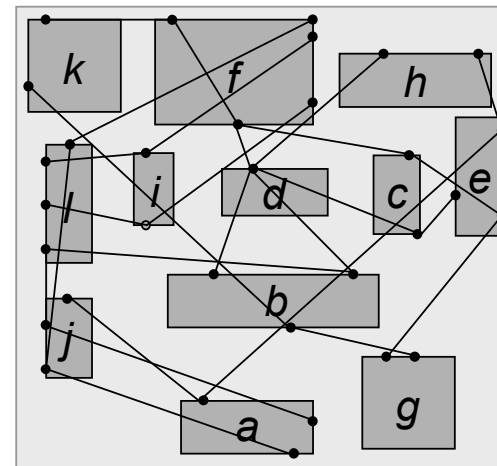
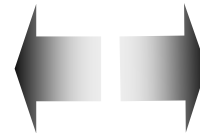
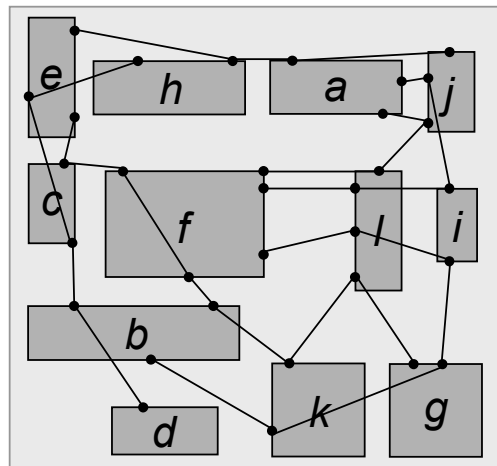
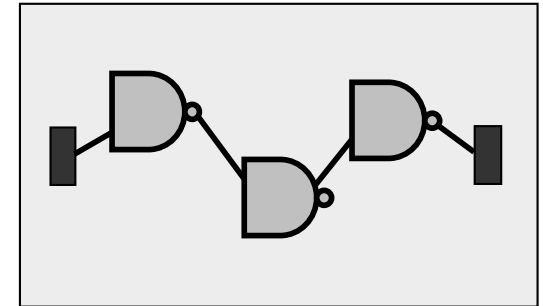
Number of Cut Nets



Wire Congestion

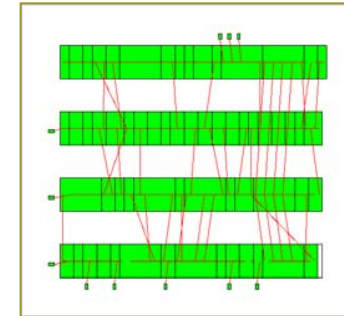
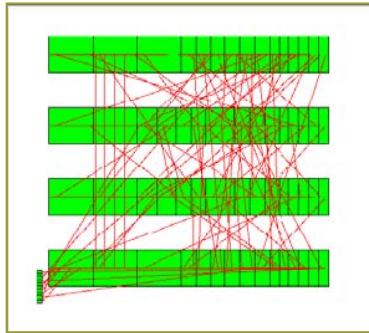


Signal Delay



Adapted from [Khang'11]

# Good vs. Bad Placement



**Bad placement causes routing congestion resulting in:**

- **Increases in circuit area (cost) and wiring**
- **Longer wires → more capacitance**
  - **Longer delay**
  - **Higher dynamic power dissipation**

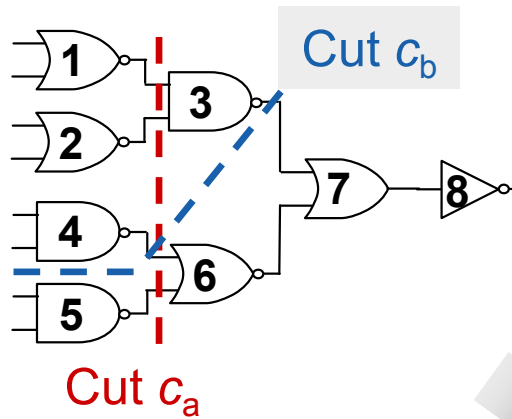
**Good placement**

- **Circuit area (cost) and wiring decreases**
- **Shorter wires → less capacitance**
  - **Shorter delay**
  - **Less dynamic power dissipation**

Adapted from [Devadas'06]

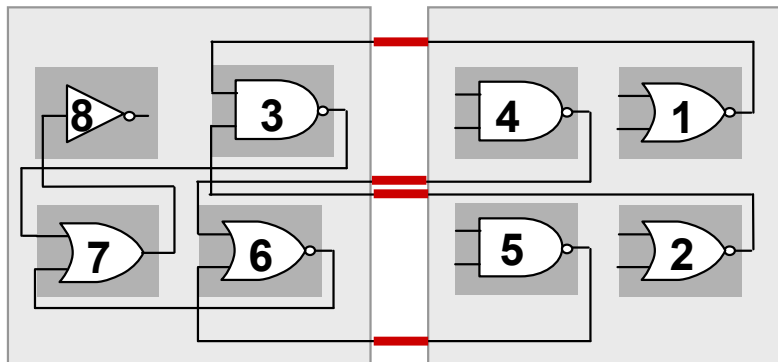
# Partitioning

Circuit:



Block A

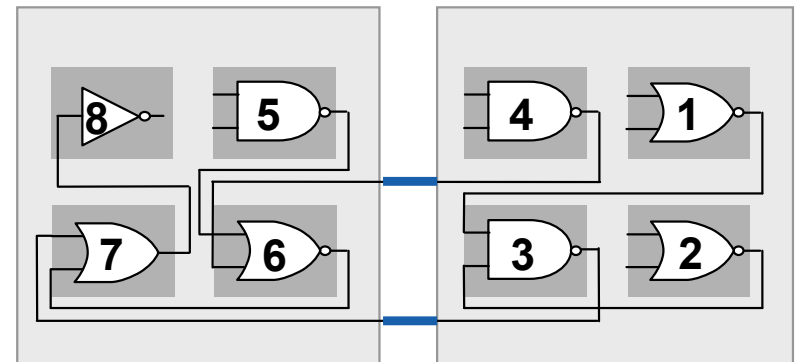
Block B



Cut  $c_a$ : four external connections

Block A

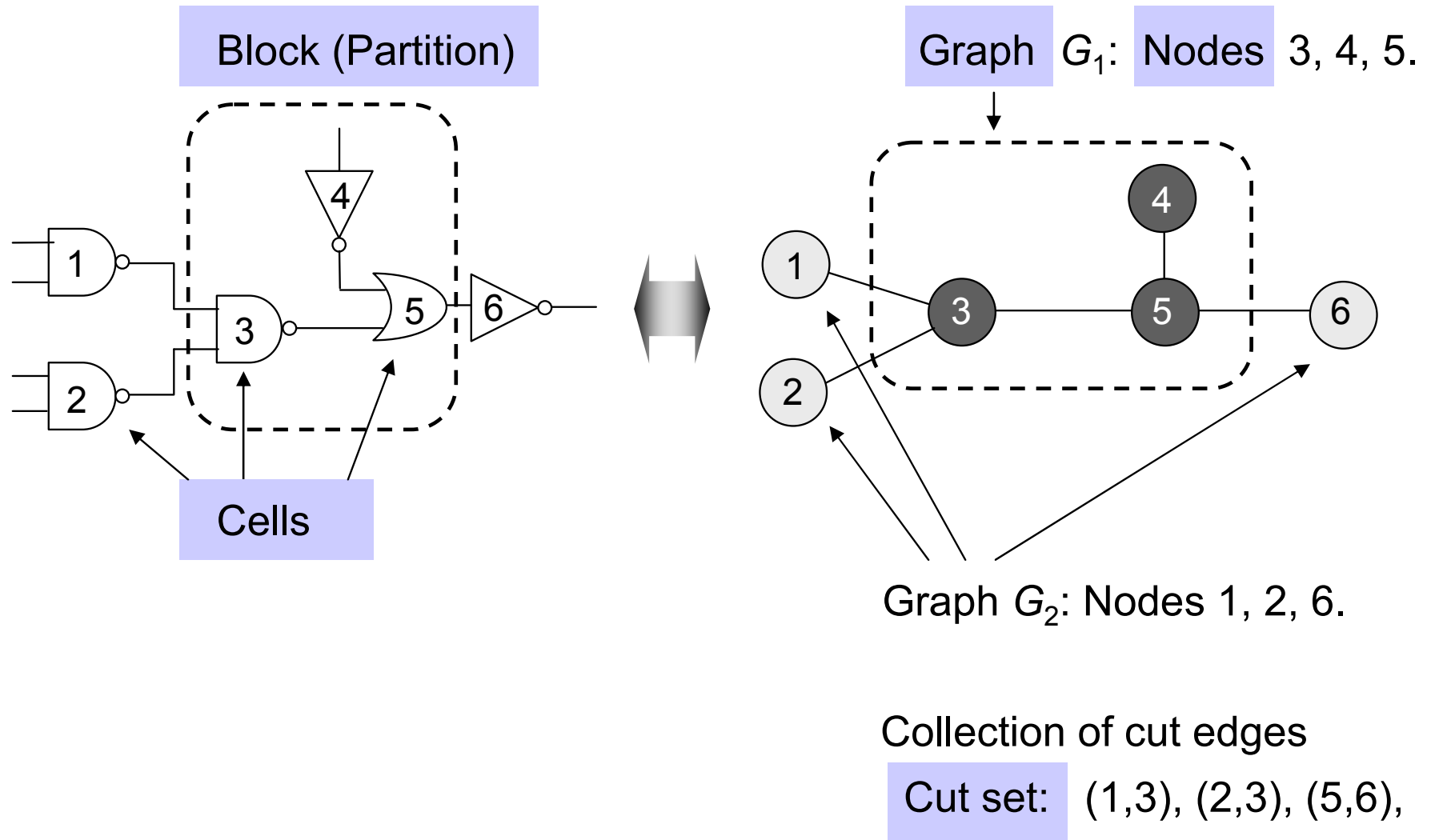
Block B



Cut  $c_b$ : two external connections

Adapted from [Khang'11]

# Partitioning Terminology



Adapted from [Khang'11]

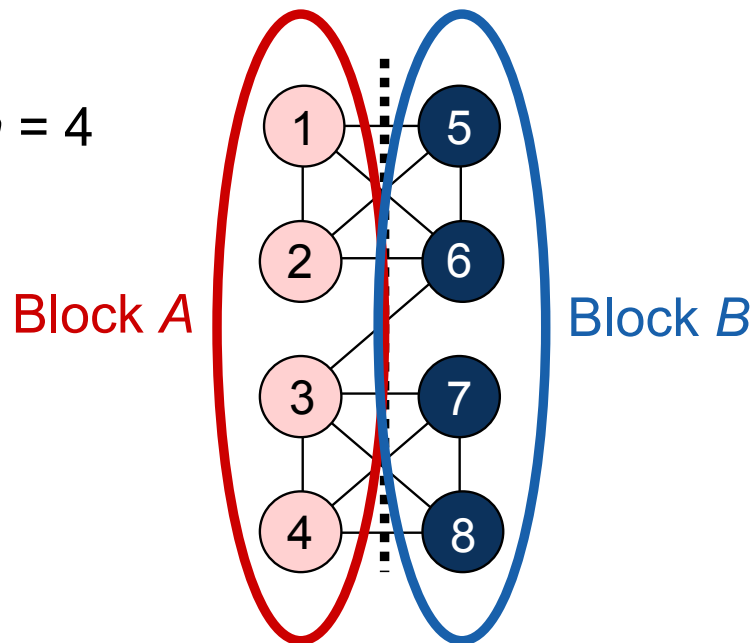


# Kernighan-Lin (KL) Algorithm

Given: A graph with  $2n$  nodes where each node has the same weight.

Goal: A partition (division) of the graph into two disjoint subsets  $A$  and  $B$  with minimum cut cost and  $|A| = |B| = n$ .

Example:  $n = 4$



Adapted from [Khang'11]

# Kernighan-Lin (KL) Algorithm Concepts

Cost  $D(v)$  of moving a node  $v$

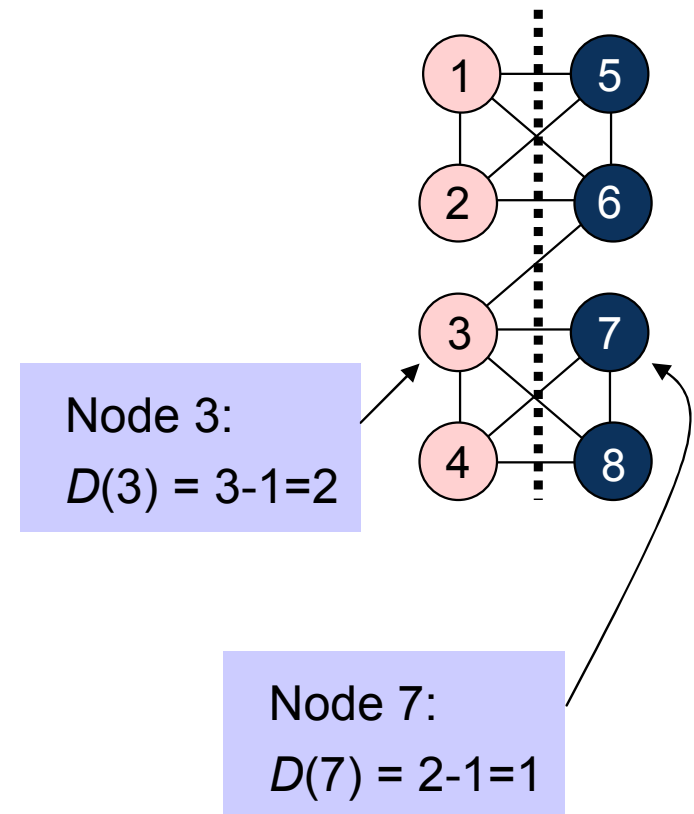
$$D(v) = |E_c(v)| - |E_{nc}(v)| ,$$

where

$E_c(v)$  is the set of  $v$ 's incident edges that are cut by the cut line, and

$E_{nc}(v)$  is the set of  $v$ 's incident edges that are not cut by the cut line.

High costs ( $D > 0$ ) indicate that the node should move, while low costs ( $D < 0$ ) indicate that the node should stay within the same partition.



Adapted from [Khang'11]

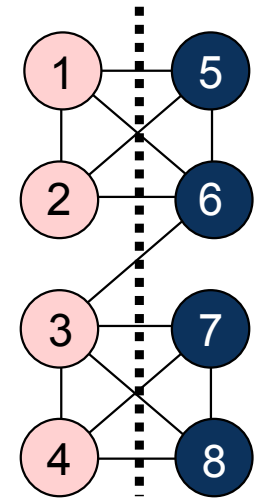
# Kernighan-Lin (KL) Algorithm Concepts

Gain of swapping a pair of nodes  $a$  and  $b$

$$\Delta g = D(a) + D(b) - 2 * c(a,b),$$

where

- $D(a)$ ,  $D(b)$  are the respective costs of nodes  $a$ ,  $b$
- $c(a,b)$  is the connection weight between  $a$  and  $b$ :  
If an edge exists between  $a$  and  $b$ ,  
then  $c(a,b) = \text{edge weight (here 1)}$ ,  
otherwise,  $c(a,b) = 0$ .



The gain  $\Delta g$  indicates how useful the swap between two nodes will be

The larger  $\Delta g$ , the more the total cut cost will be reduced

Adapted from [Khang'11]

# Kernighan-Lin (KL) Algorithm Concepts

Gain of swapping a pair of nodes  $a$  and  $b$

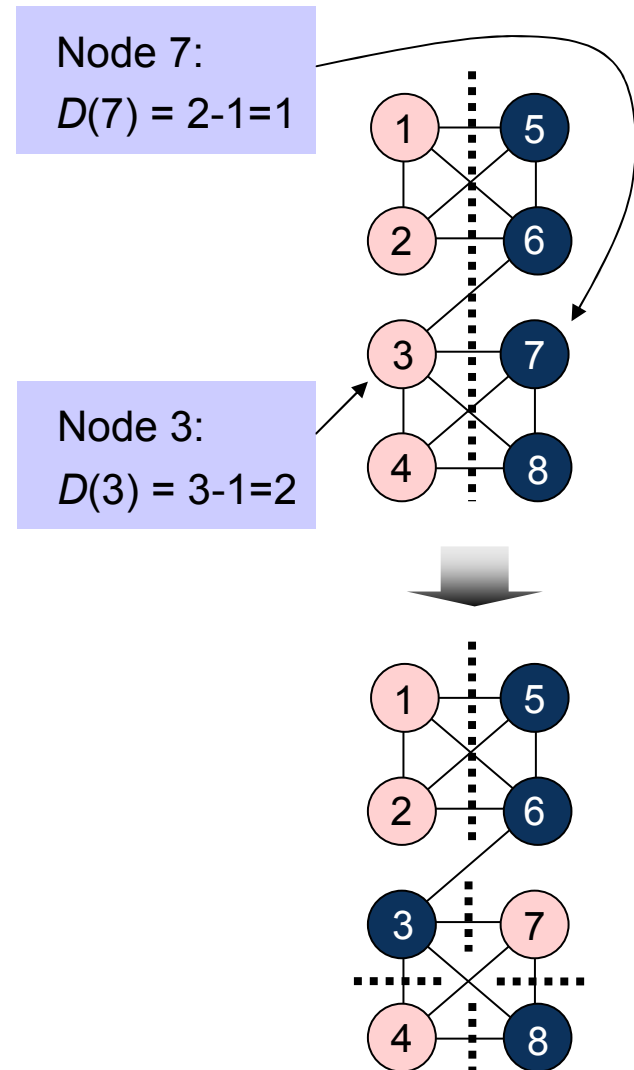
$$\Delta g = D(a) + D(b) - 2 \cdot c(a,b),$$

where

- $D(a)$ ,  $D(b)$  are the respective costs of nodes  $a$ ,  $b$
- $c(a,b)$  is the connection weight between  $a$  and  $b$ :  
If an edge exists between  $a$  and  $b$ ,  
then  $c(a,b) = \text{edge weight (here 1)}$ ,  
otherwise,  $c(a,b) = 0$ .

$$\Delta g(3,7) = D(3) + D(7) - 2 \cdot c(a,b) = 2 + 1 - 2 = 1$$

=> Swapping nodes 3 and 7 would reduce the cut size by 1



Adapted from [Khang'11]

# Kernighan-Lin (KL) Algorithm Concepts

Gain of swapping a pair of nodes  $a$  and  $b$

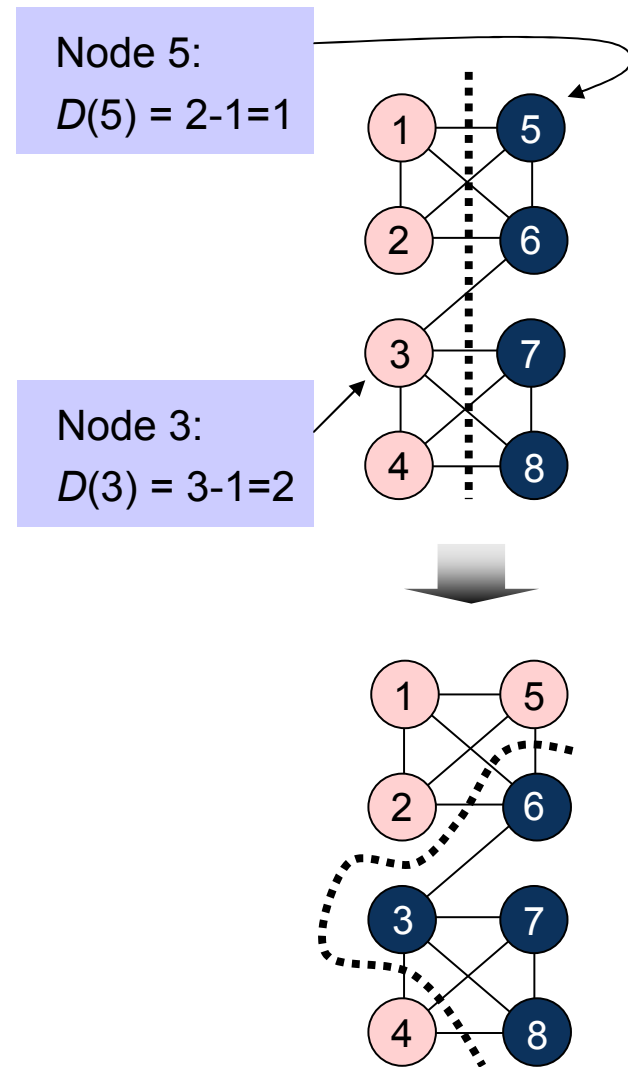
$$\Delta g = D(a) + D(b) - 2 * c(a,b),$$

where

- $D(a)$ ,  $D(b)$  are the respective costs of nodes  $a$ ,  $b$
- $c(a,b)$  is the connection weight between  $a$  and  $b$ :  
If an edge exists between  $a$  and  $b$ ,  
then  $c(a,b) =$  edge weight (here 1),  
otherwise,  $c(a,b) = 0$ .

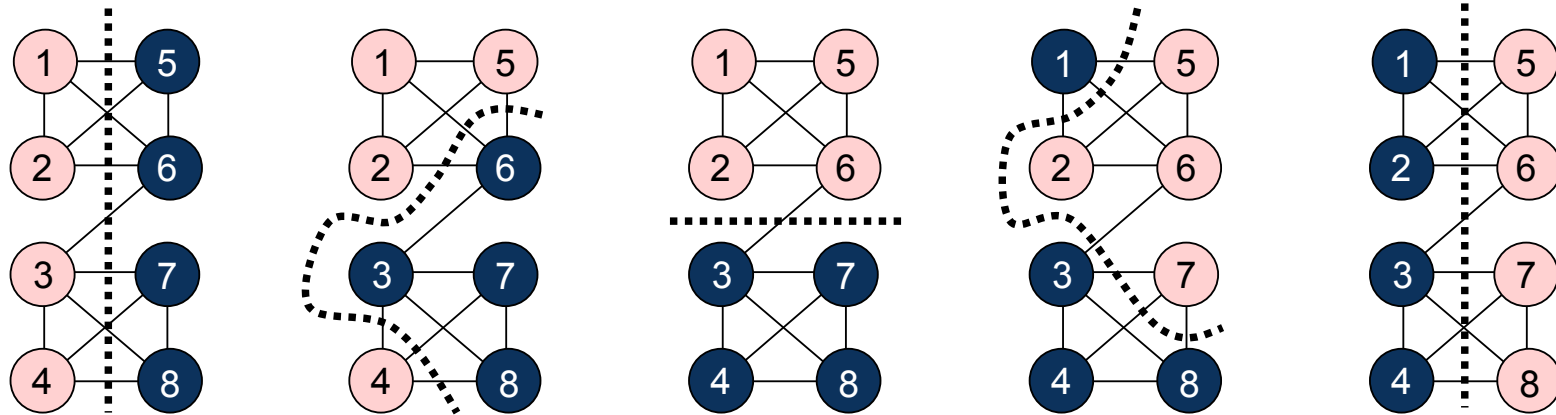
$$\Delta g (3,5) = D(3) + D(5) - 2 * c(a,b) = 2 + 1 - 0 = 3$$

=> Swapping nodes 3 and 5 would reduce the cut size by 3



Adapted from [Khang'11]

# Kernighan-Lin (KL) Algorithm Example



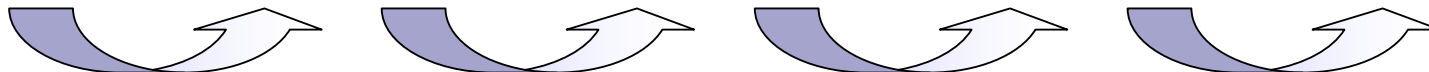
Cut cost: 9  
Not fixed:  
1,2,3,4,5,6,7,8

Cut cost: 6  
Not fixed:  
1,2,4,6,7,8

Cut cost: 1  
Not fixed:  
1,2,7,8

Cut cost: 7  
Not fixed:  
2,8

Cut cost: 9  
Not fixed:  
-



$D(1) = 1$      $D(5) = 1$   
 $D(2) = 1$      $D(6) = 2$   
 $D(3) = 2$      $D(7) = 1$   
 $D(4) = 1$      $D(8) = 1$

$\Delta g_1 = 2+1-0 = 3$   
**Swap (3,5)**  
 $G_1 = \Delta g_1 = 3$

$D(1) = -1$      $D(6) = 2$   
 $D(2) = -1$      $D(7) = -1$   
 $D(4) = 3$      $D(8) = -1$

$\Delta g_2 = 3+2-0 = 5$   
**Swap (4,6)**  
 $G_2 = G_1 + \Delta g_2 = 8$

$D(1) = -3$      $D(7) = -3$   
 $D(2) = -3$      $D(8) = -3$

$\Delta g_3 = -3-3-0 = -6$   
**Swap (1,7)**  
 $G_3 = G_2 + \Delta g_3 = 2$

$D(2) = -1$      $D(8) = -1$

$\Delta g_4 = -1-1-0 = -2$   
**Swap (2,8)**  
 $G_4 = G_3 + \Delta g_4 = 0$

Adapted from [Khang'11]

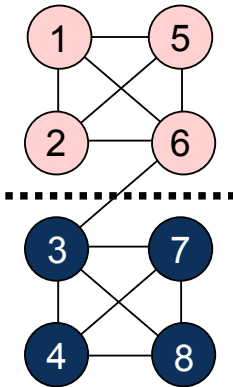
# Kernighan-Lin (KL) Algorithm Example

$D(1) = 1$ $D(2) = 1$ $D(3) = 2$ $D(4) = 1$ $D(5) = 1$ $D(6) = 2$ $D(7) = 1$ $D(8) = 1$ $\Delta g_1 = 2+1-0 = 3$ <b>Swap (3,5)</b> $G_1 = \Delta g_1 = 3$	$D(1) = -1$ $D(2) = -1$ $D(4) = 3$ $D(6) = 2$ $D(7) = -1$ $D(8) = -1$ $\Delta g_2 = 3+2-0 = 5$ <b>Swap (4,6)</b> $G_2 = G_1 + \Delta g_2 = 8$	$D(1) = -3$ $D(2) = -3$ $D(7) = -3$ $D(8) = -3$ $\Delta g_3 = -3-3-0 = -6$ <b>Swap (1,7)</b> $G_3 = G_2 + \Delta g_3 = 2$	$D(2) = -1$ $D(8) = -1$ $\Delta g_4 = -1-1-0 = -2$ <b>Swap (2,8)</b> $G_4 = G_3 + \Delta g_4 = 0$
---	---	---	---

Maximum positive gain  $G_m = 8$  with  $m = 2$ .

Since  $G_m > 0$ , the first  $m = 2$  swaps (3,5) and (4,6) are executed.

Since  $G_m > 0$ , more passes are needed until  $G_m \leq 0$ .



Adapted from [Khang'11]

# Min-Cut Placement

---

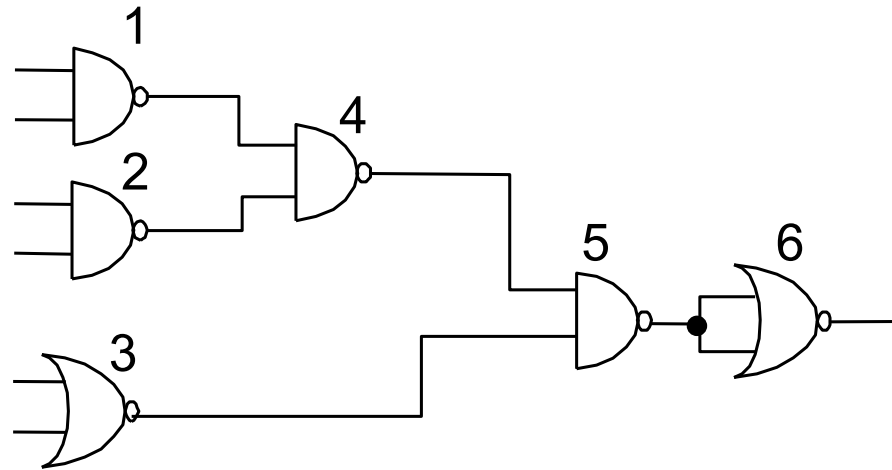
- ▶ Use partitioning algorithm (such as KL) to divide netlist into two regions
- ▶ Use partitioning algorithm to recursively divide the two partitions into two smaller regions (four regions total)
- ▶ Each cut heuristically minimizes the number of cut nets

Adapted from [Khang'11]

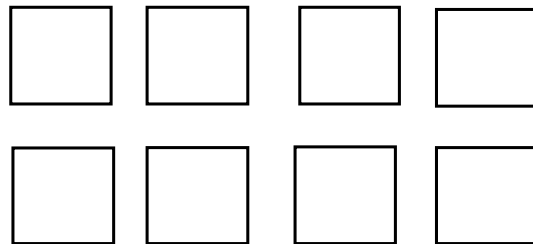


# Min-Cut Example

Given:

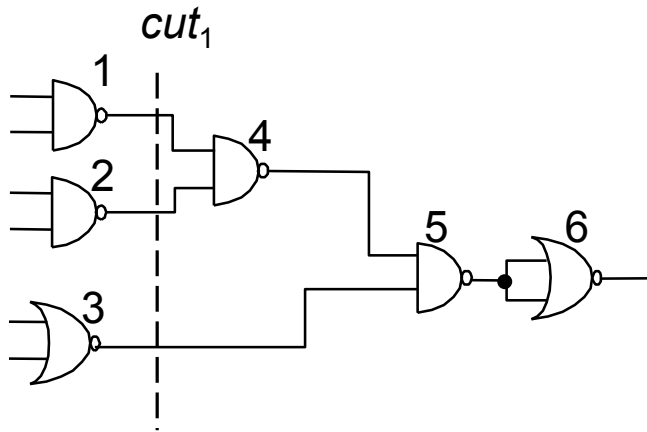


Task: 4 x 2 placement with minimum wirelength using alternative cutline directions and the KL algorithm

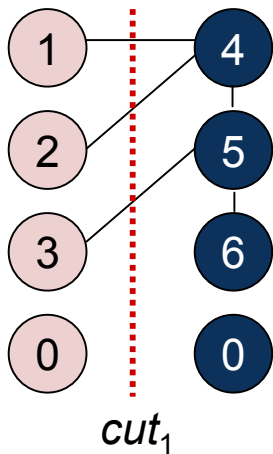


Adapted from [Khang'11]

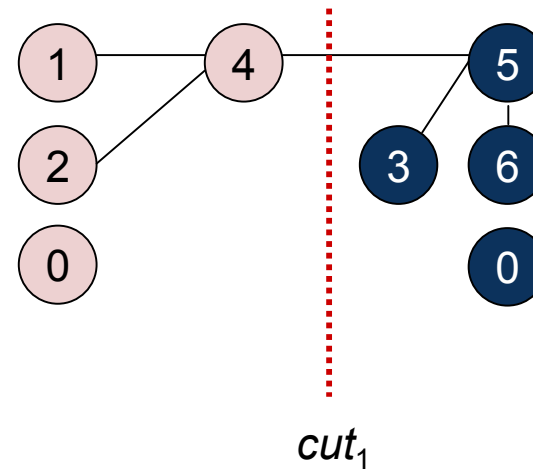
# Min-Cut Example



Vertical cut  $cut_1$ :  $L=\{1,2,3\}$ ,  $R=\{4,5,6\}$

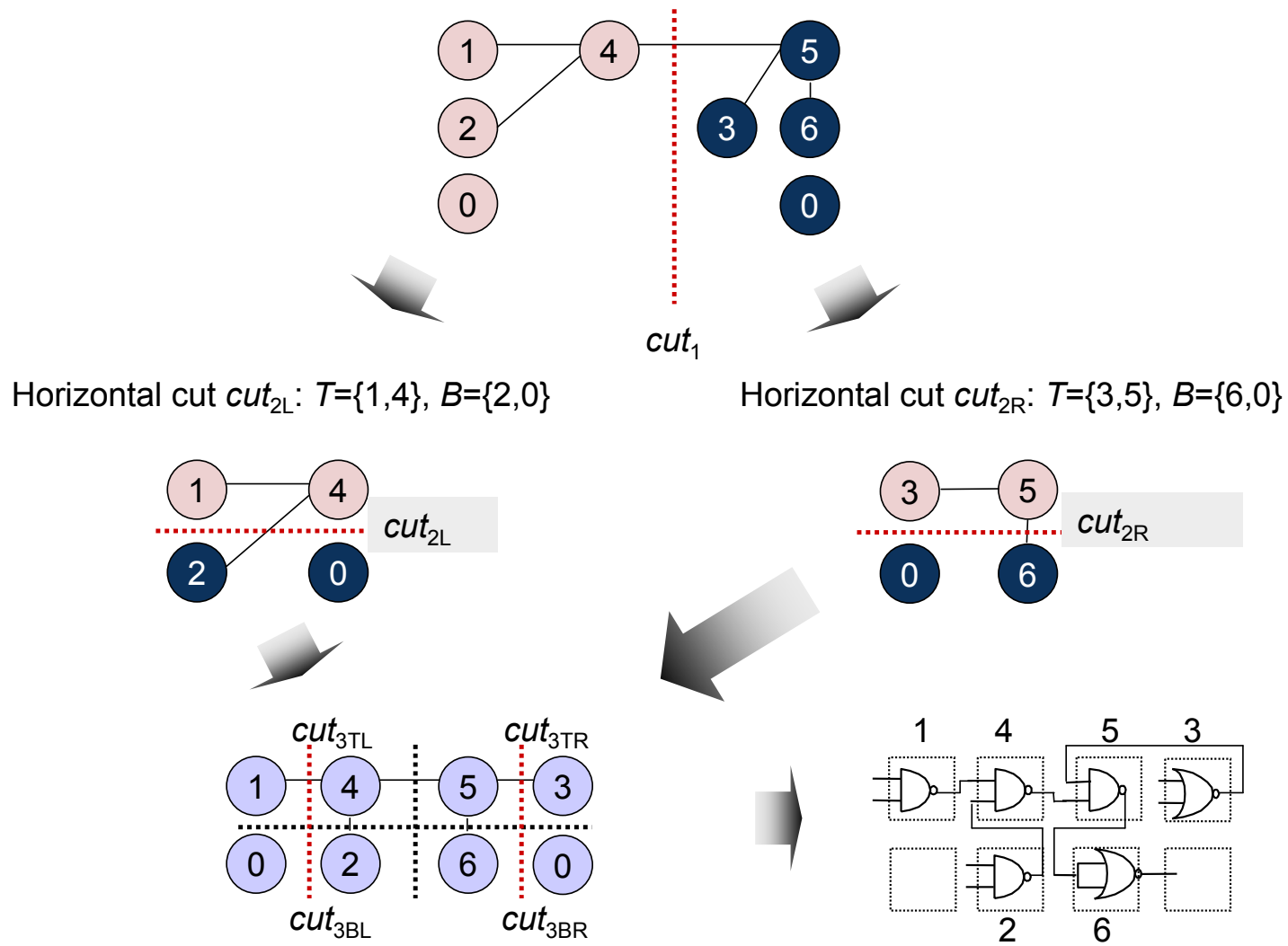


KL Algorithms



Adapted from [Khang'11]

# Min-Cut Example



Adapted from [Khang'11]

# Analytical Placement with Quadratic Placement Algorithm

---

- Objective function is quadratic; sum of (weighted) **squared Euclidean distance** represents placement objective function

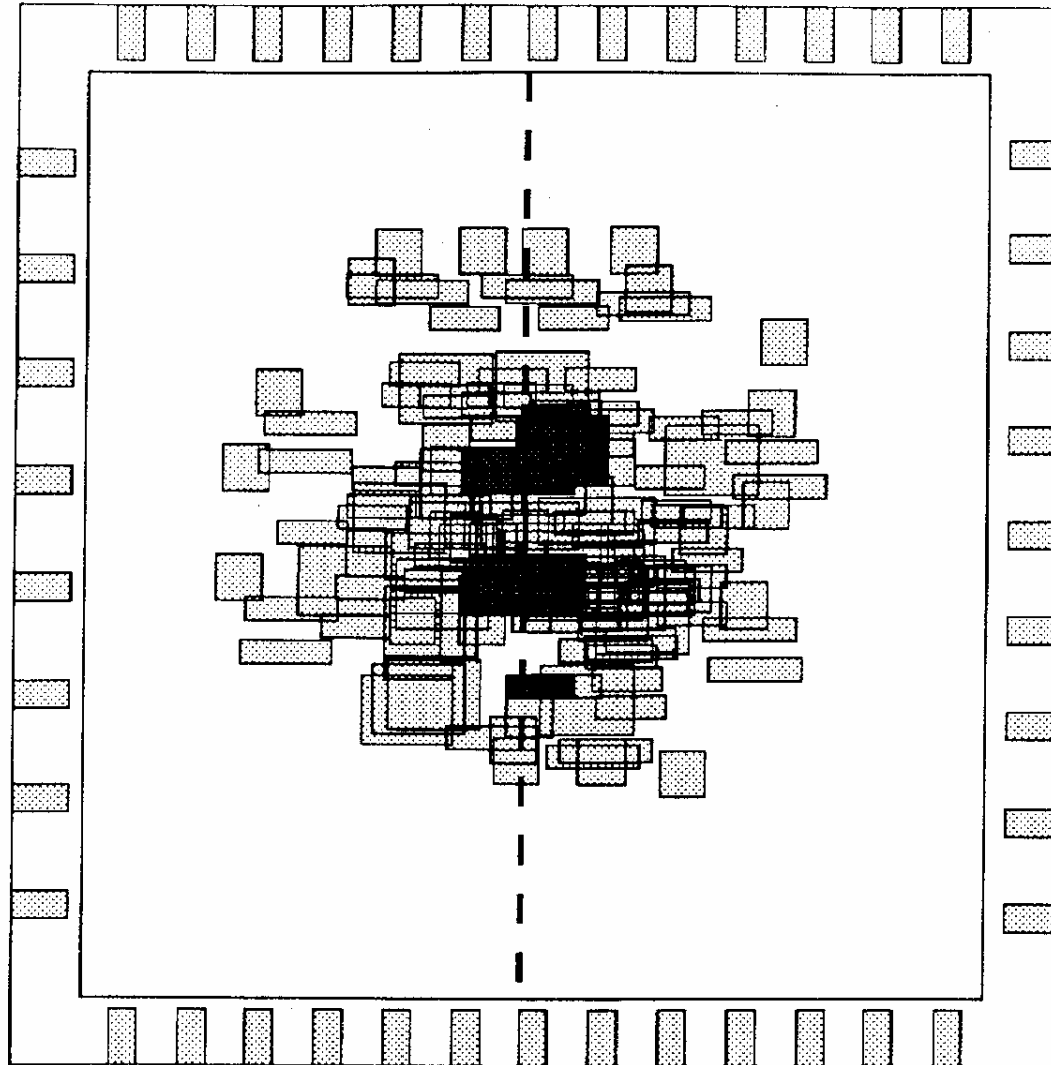
$$L(P) = \frac{1}{2} \sum_{i,j=1}^n c_{ij} \left( (x_i - x_j)^2 + (y_i - y_j)^2 \right)$$

where  $n$  is the total number of cells, and  $c(i,j)$  is the connection cost between cells  $i$  and  $j$ .

- Only two-point-connections
- Minimize objective function by equating its derivative to zero which reduces to solving a system of linear equations

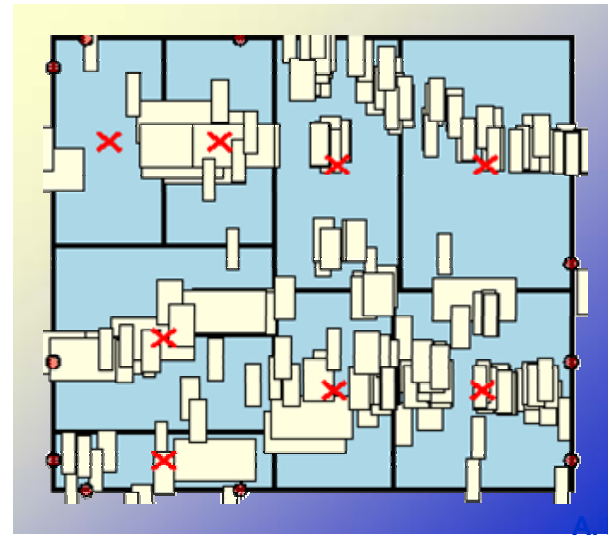
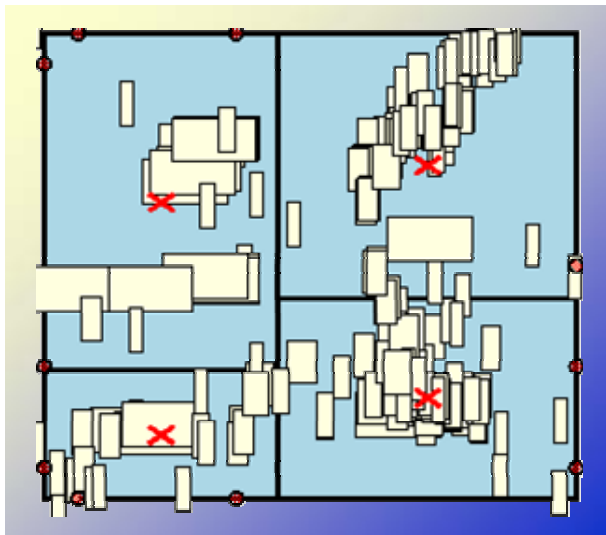
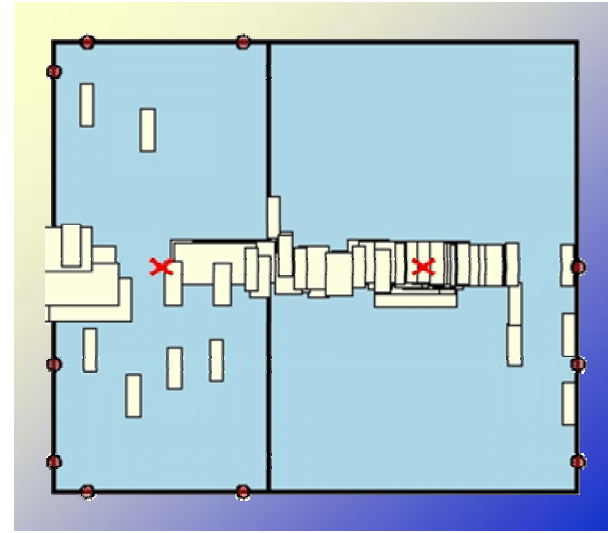
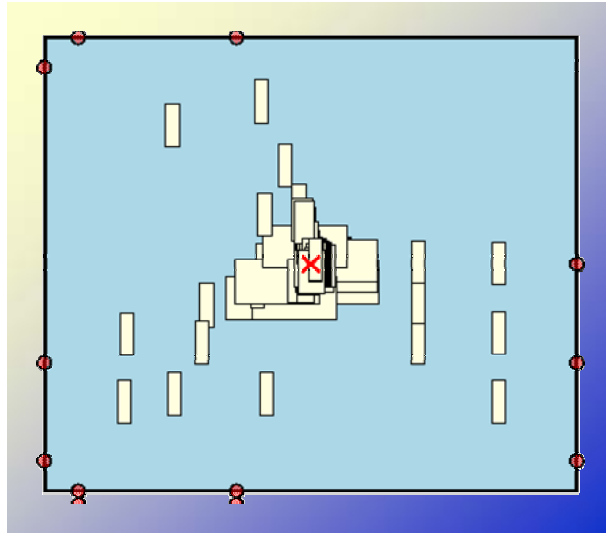
Adapted from [Khang'11]

# Minimizing Objective Function Clumps Cells Together



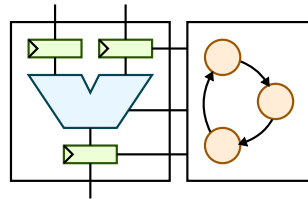
Adapted from [Devadas'06]

# Using Partitioning to Pull Cells Apart



Adapted from [Devadas'06,Kahng'11]

# Part 3: CAD Algorithms

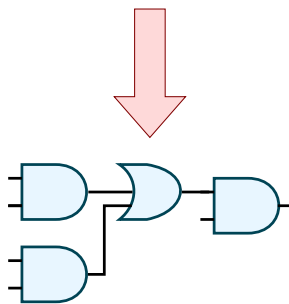


$$x = a'bc + a'bc'$$

$$y = b'c' + ab' + ac$$

$$x = a'b$$

$$y = b'c' + ac$$

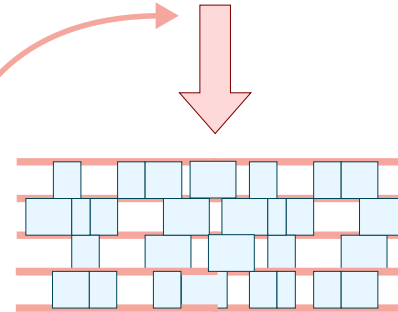


RTL to Logic  
Synthesis

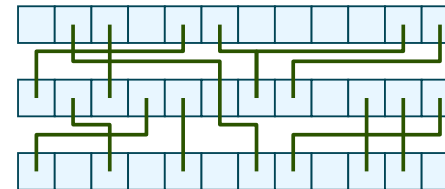
Technology  
Independent  
Synthesis

Technology  
Dependent  
Synthesis

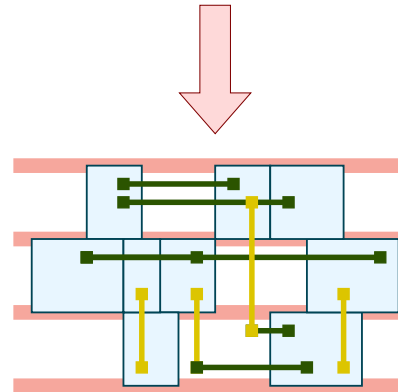
Placement



**Global  
Routing**



Detailed  
Routing



# General Routing Problem

Netlist:

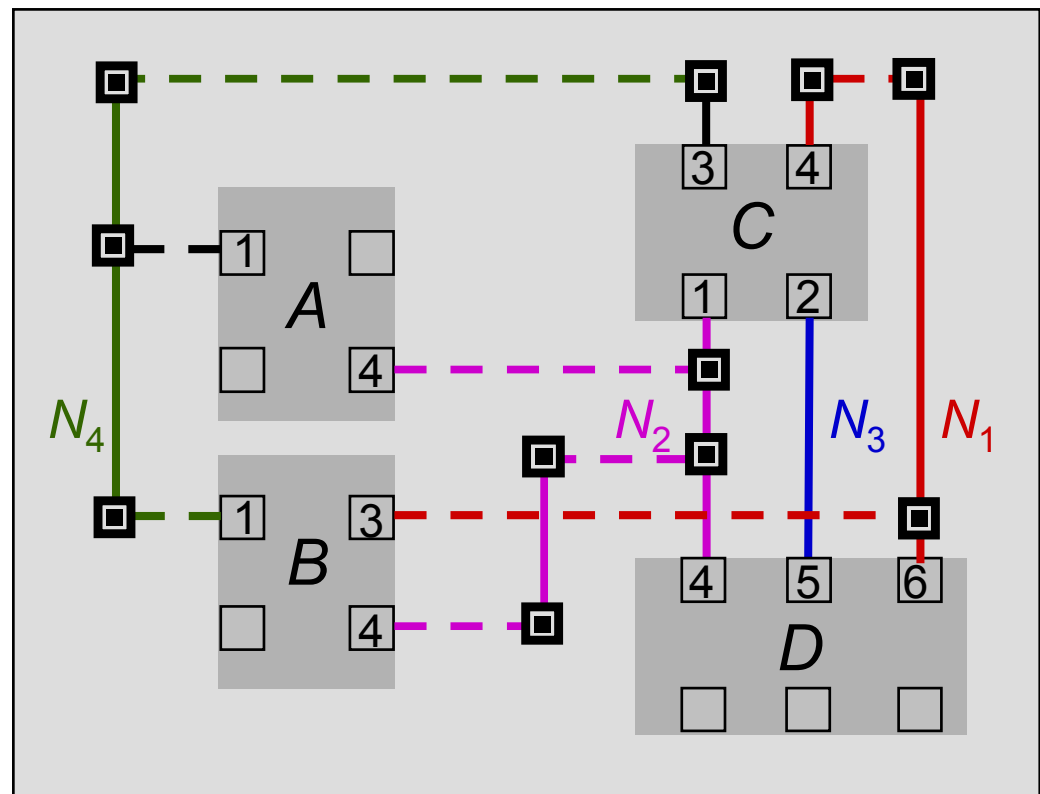
$$N_1 = \{C_4, D_6, B_3\}$$

$$N_2 = \{D_4, B_4, C_1, A_4\}$$

$$N_3 = \{C_2, D_5\}$$

$$N_4 = \{B_1, A_1, C_3\}$$

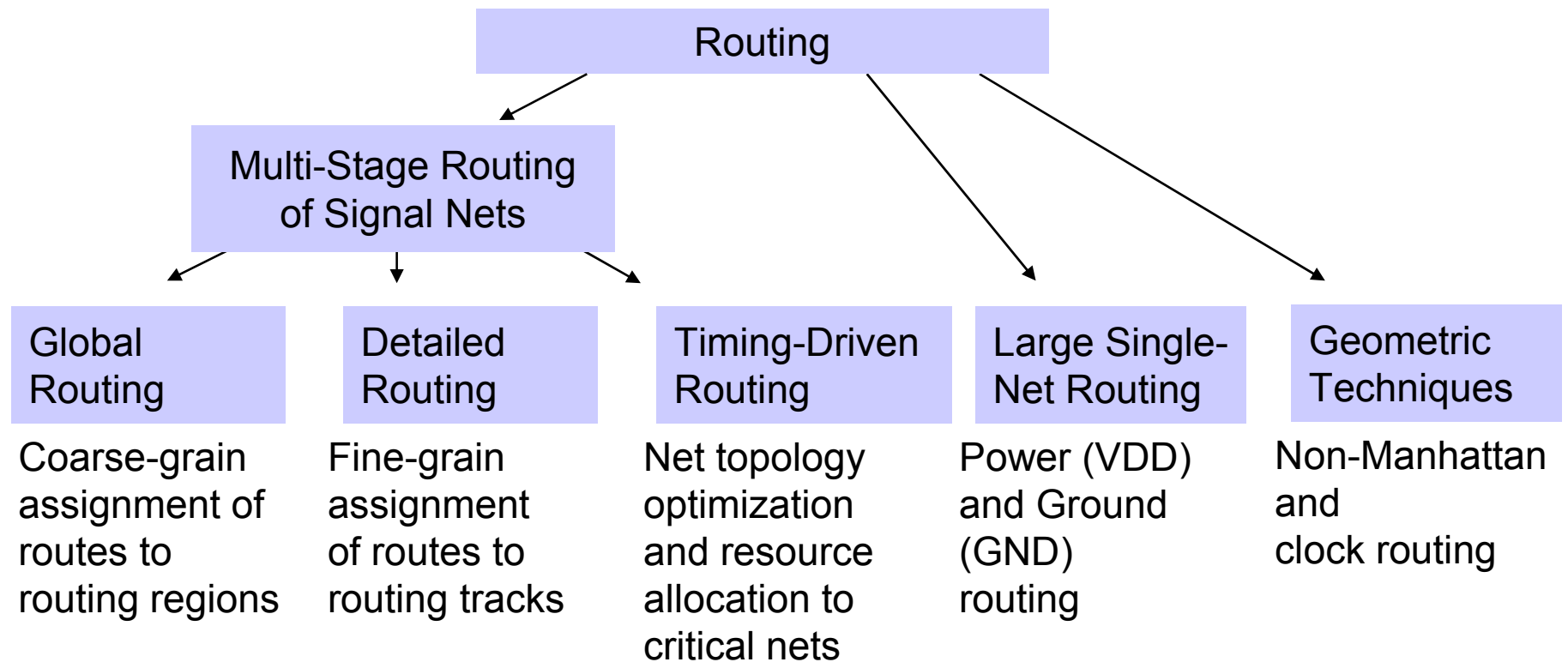
Technology Information  
(Design Rules)



Adapted from [Kahng'11]



# Routing Algorithm Taxonomy



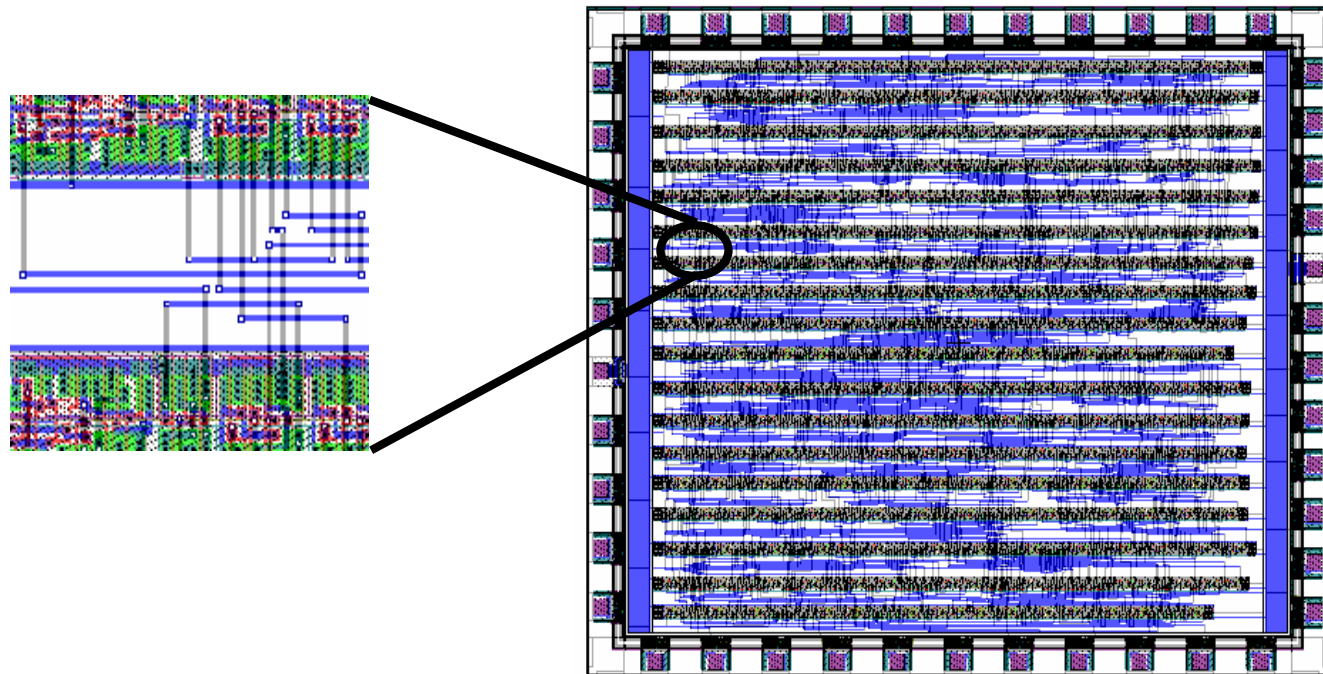
Adapted from [Kahng'11]



# Terminology: Channel

## Channel

Rectangular routing region with pins on two opposite sides



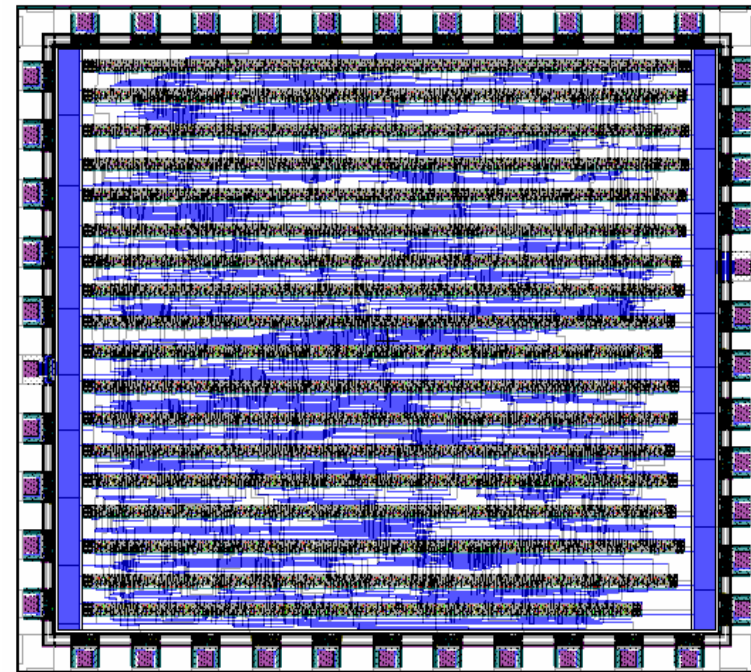
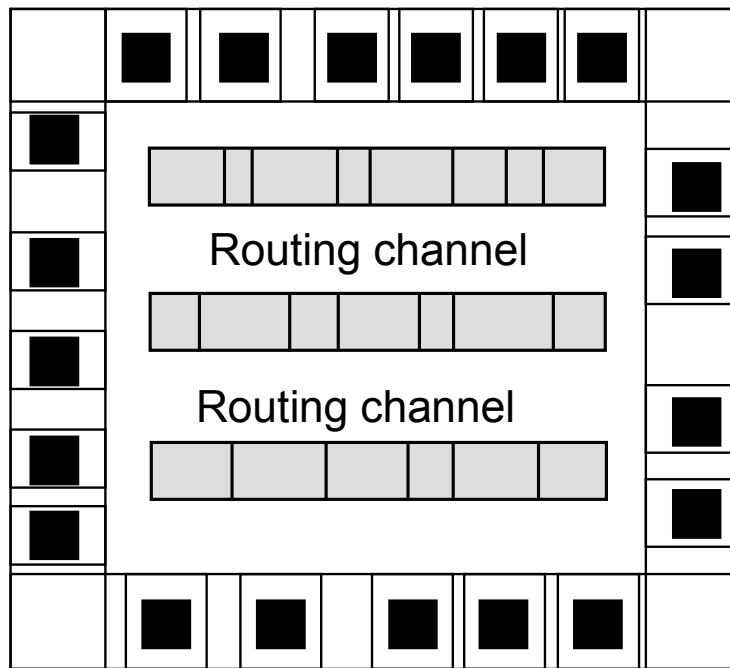
Standard cell layout (Two-layer routing)

Adapted from [Kahng'11]

# Terminology: Channel

## Channel

Rectangular routing region with pins on two opposite sides



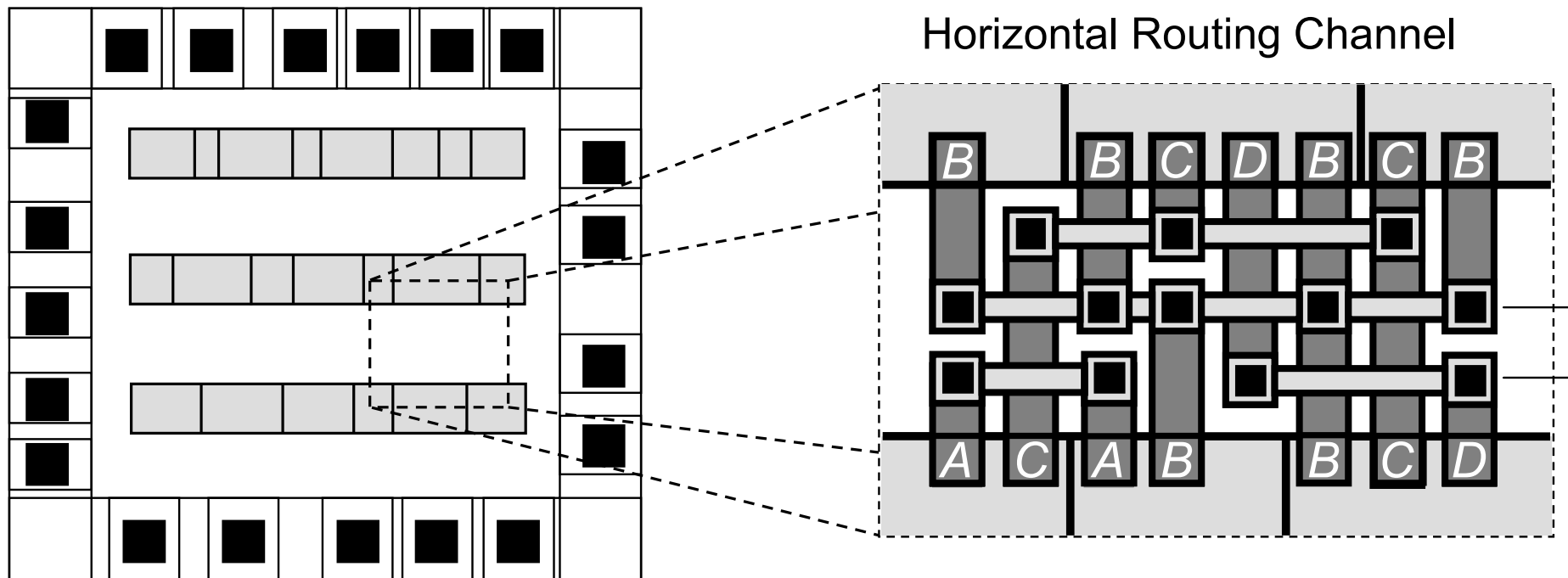
Standard cell layout (Two-layer routing)

Adapted from [Kahng'11]

# Terminology: Capacity

## Capacity

Number of available routing tracks or columns

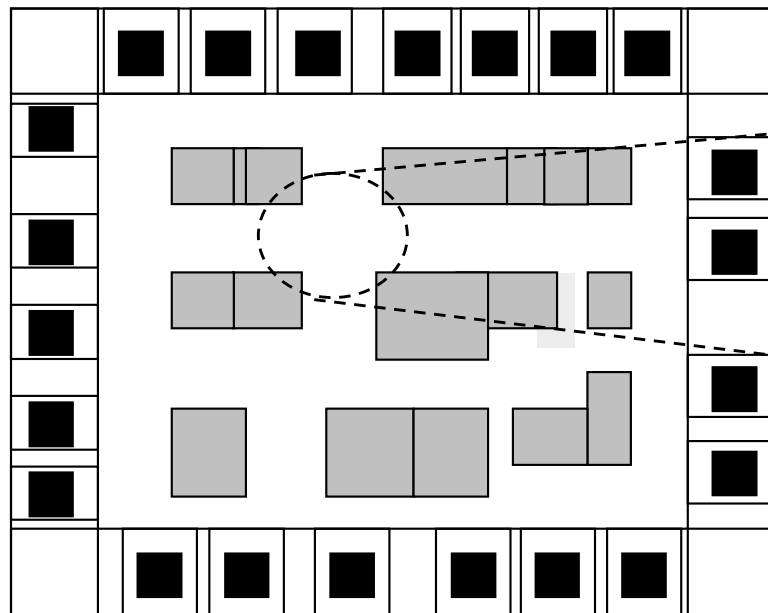


Adapted from [Kahng'11]

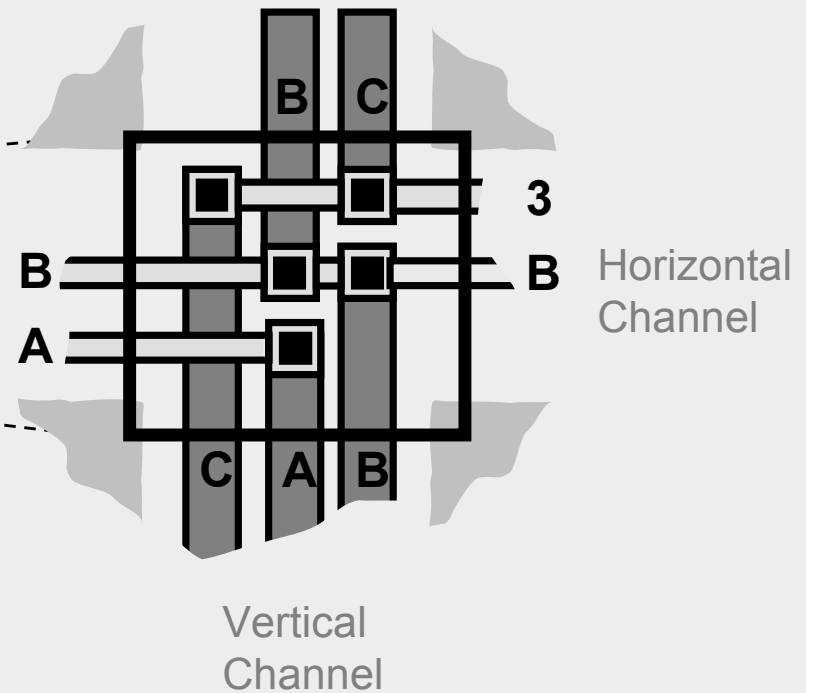
# Terminology: Switchbox

## Switchbox (Two-layer macro cell layout)

Intersection of horizontal and vertical channels



Horizontal Channel

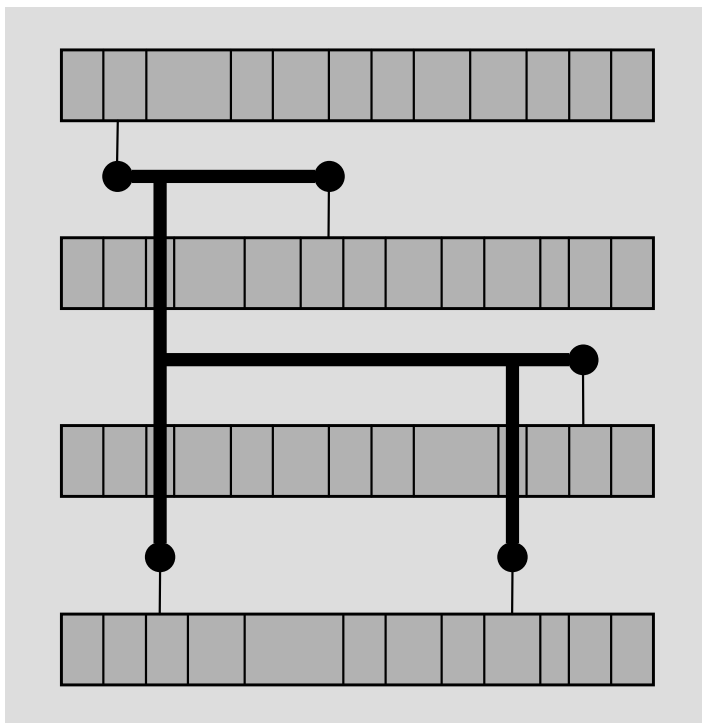


Adapted from [Kahng'11]

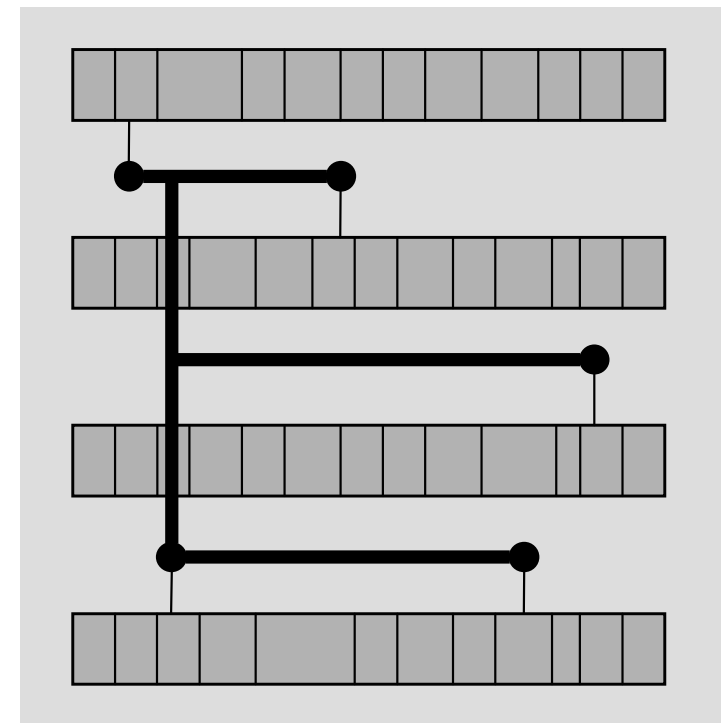
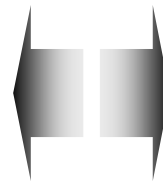


# Optimizing Wirelength vs. Number of Feedthroughs

## Standard-cell design



Steiner tree solution with minimal wirelength

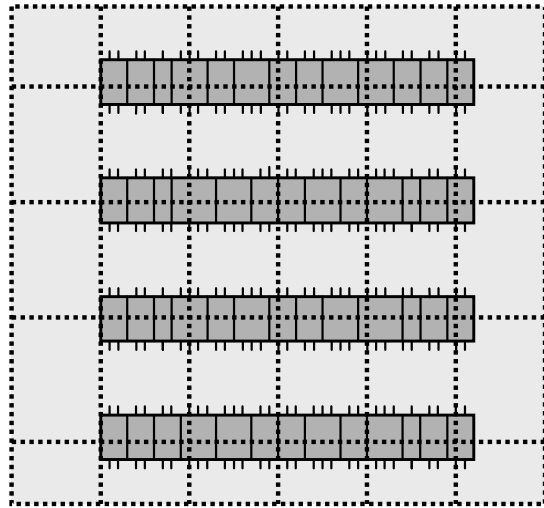


Steiner tree solution with fewest feedthrough cells

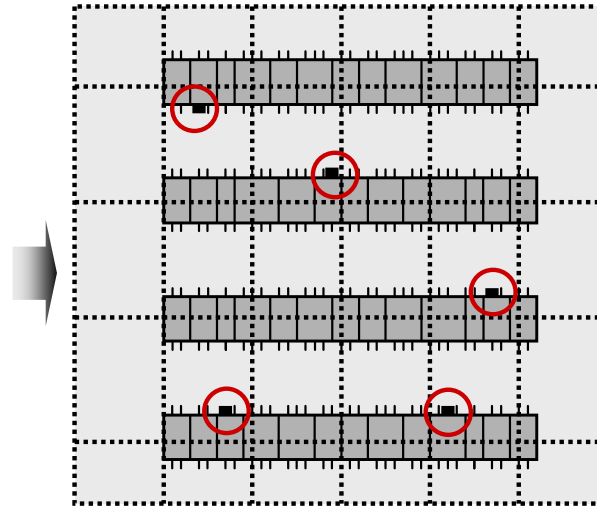
Adapted from [Kahng'11]



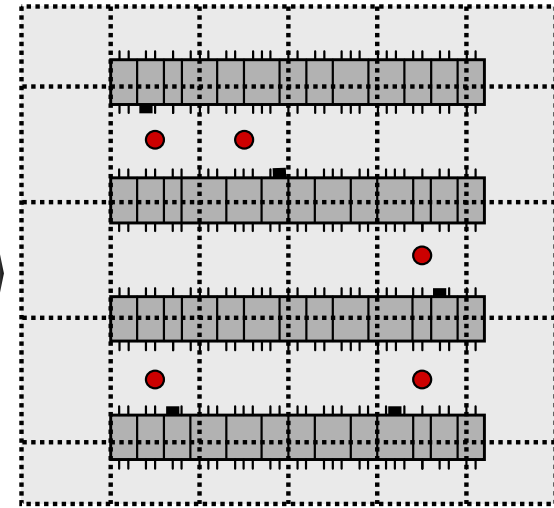
# Rectilinear Routing



Defining routing regions



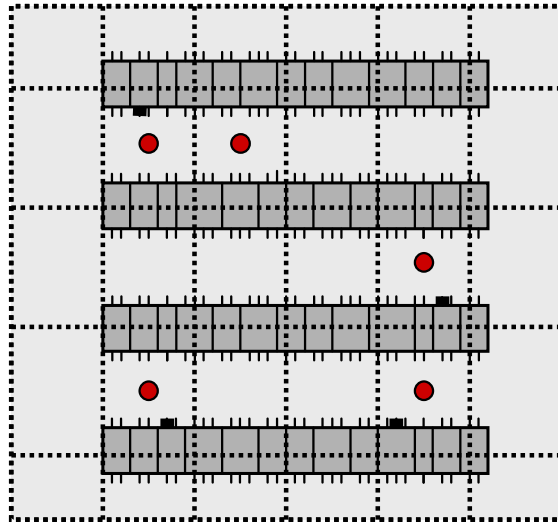
Pin connections



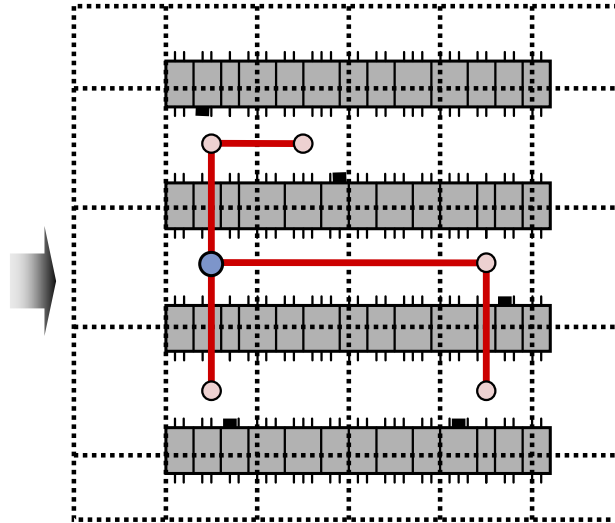
Pins assigned to grid cells

Adapted from [Kahng'11]

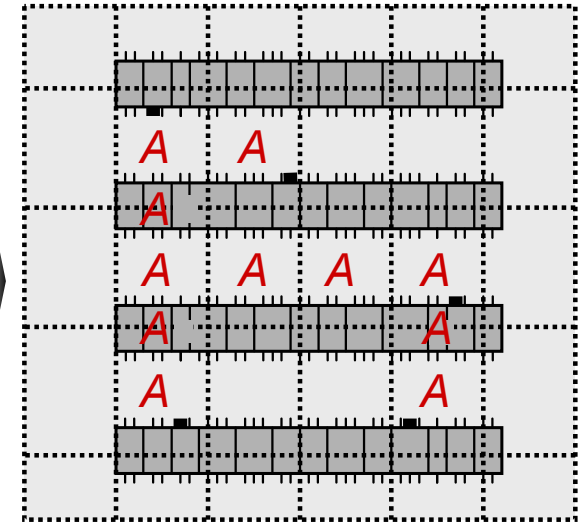
# Rectilinear Routing



Pins assigned to grid cells



Rectilinear Steiner minimum tree (RSMT)



Assigned routing regions and feedthrough cells

Adapted from [Kahng'11]

# Heuristic Sequential Steiner Tree Algorithm

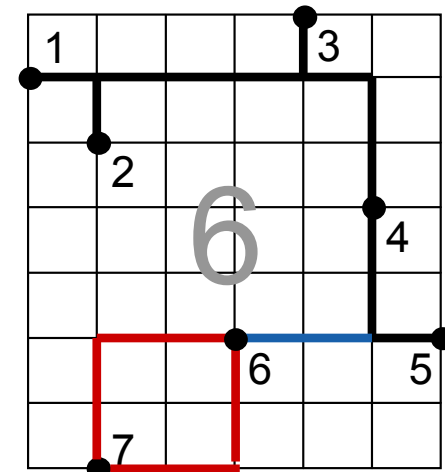
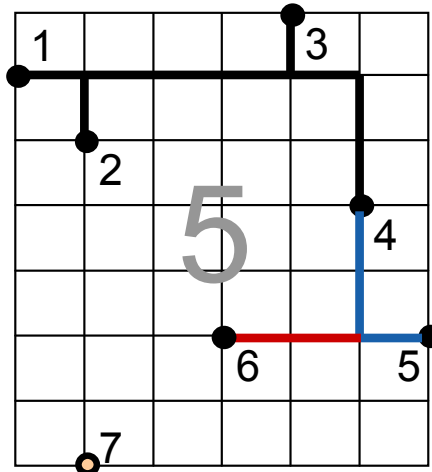
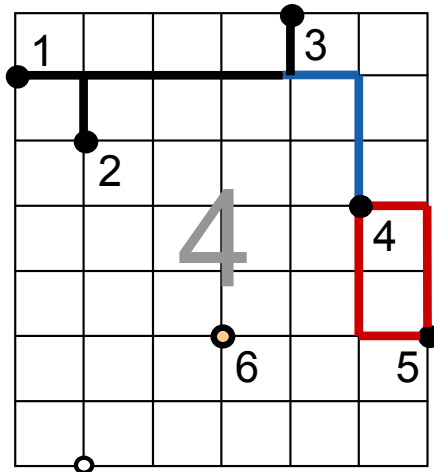
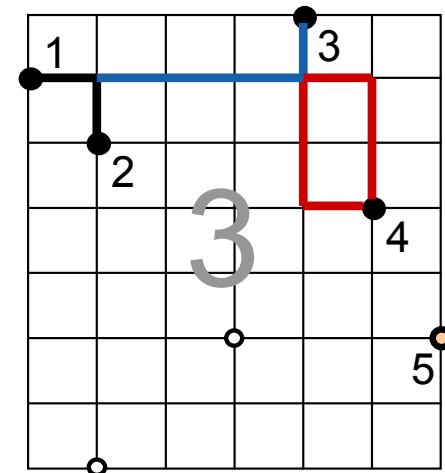
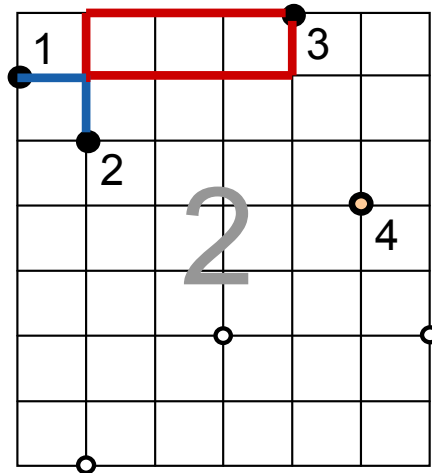
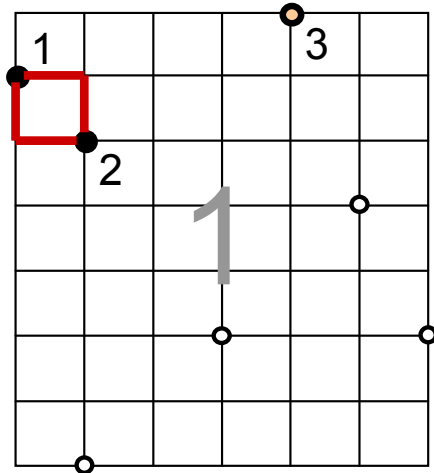
---

1. Find the closest (in terms of rectilinear distance) pin pair, construct their minimum bounding box (MBB)
2. Find the closest point pair ( $p_{MBB}, p_C$ ) between any point  $p_{MBB}$  on the MBB and  $p_C$  from the set of pins to consider
3. Construct the MBB of  $p_{MBB}$  and  $p_C$
4. Add the  $L$ -shape that  $p_{MBB}$  lies on to  $T$  (deleting the other  $L$ -shape). If  $p_{MBB}$  is a pin, then add any  $L$ -shape of the MBB to  $T$ .
5. Goto step 2 until the set of pins to consider is empty



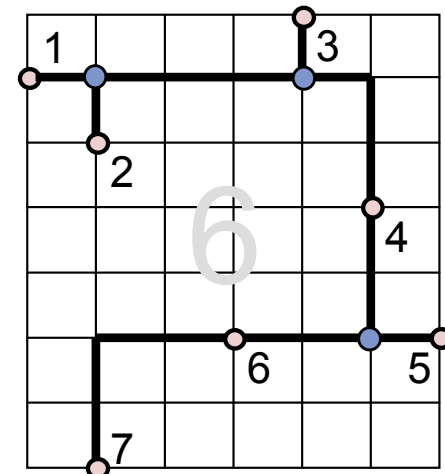
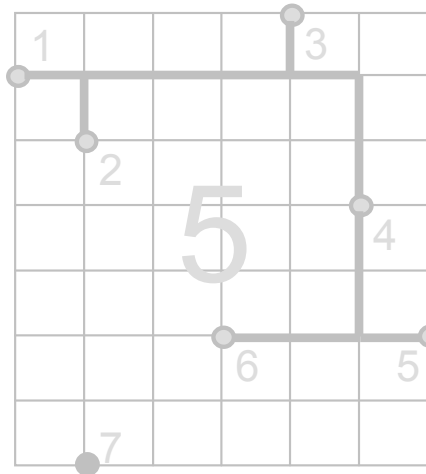
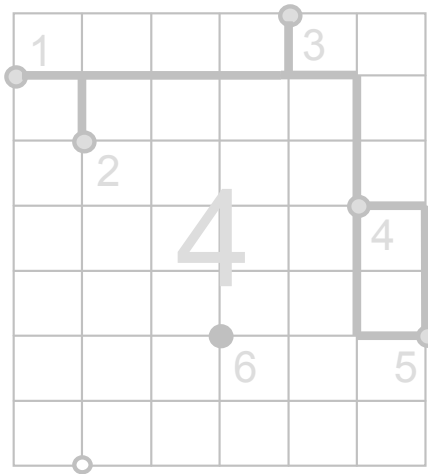
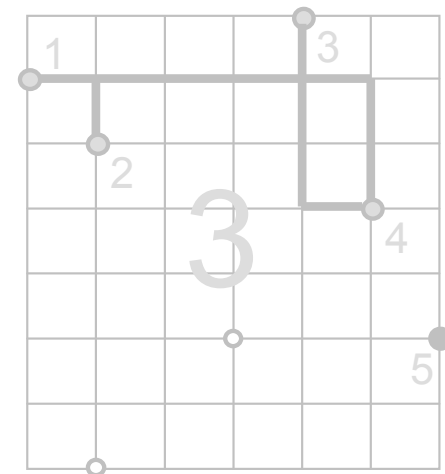
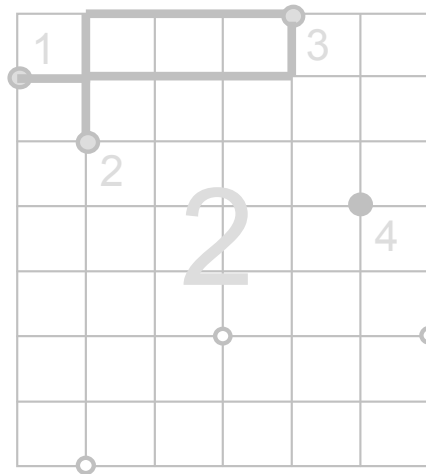
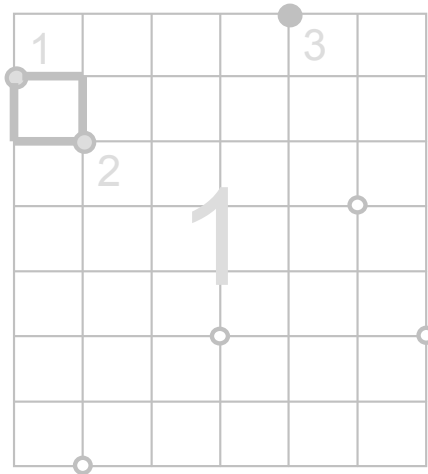
Adapted from [Kahng'11]

# Heuristic Sequential Steiner Tree Example



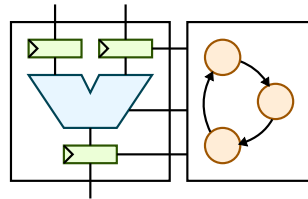
Adapted from [Kahng'11]

# Heuristic Sequential Steiner Tree Example



Adapted from [Kahng'11]

# Part 3: CAD Algorithms

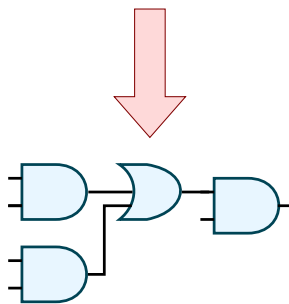


$$x = a'bc + a'bc'$$

$$y = b'c' + ab' + ac$$

$$x = a'b$$

$$y = b'c' + ac$$

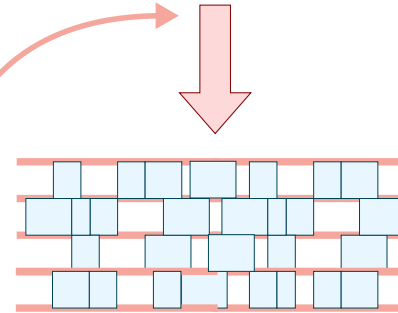


RTL to Logic  
Synthesis

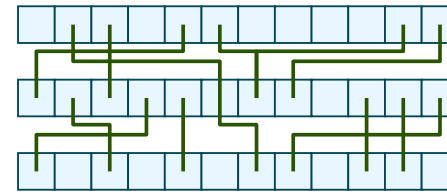
Technology  
Independent  
Synthesis

Technology  
Dependent  
Synthesis

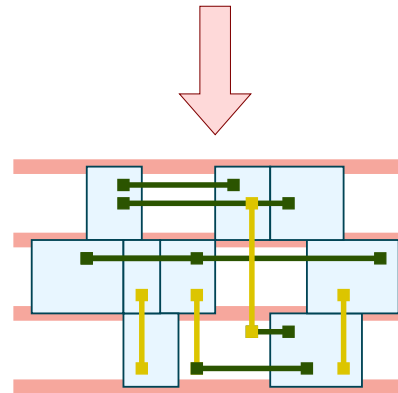
Placement



Global  
Routing

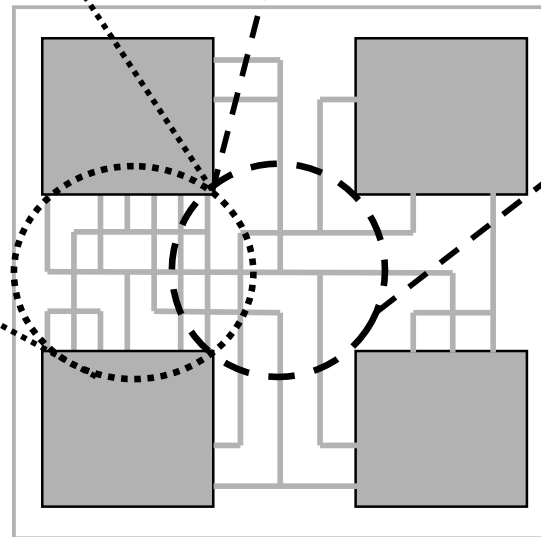
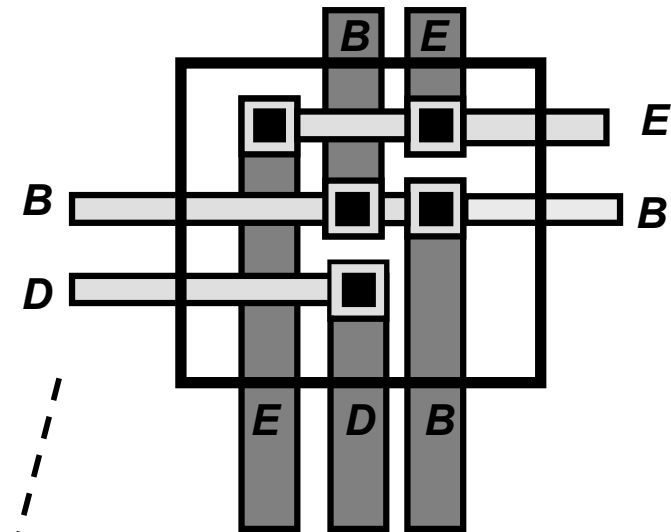
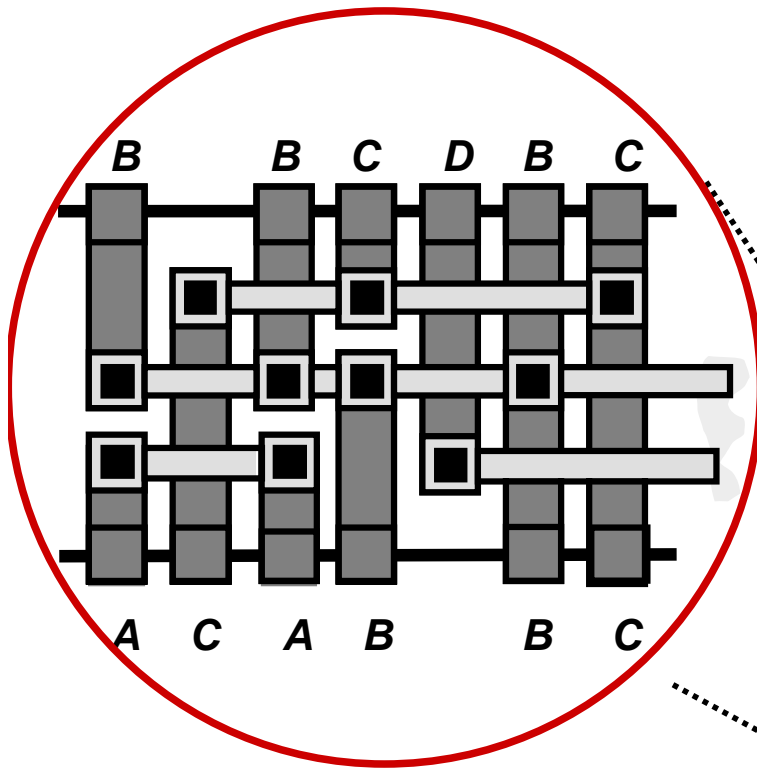


**Detailed  
Routing**



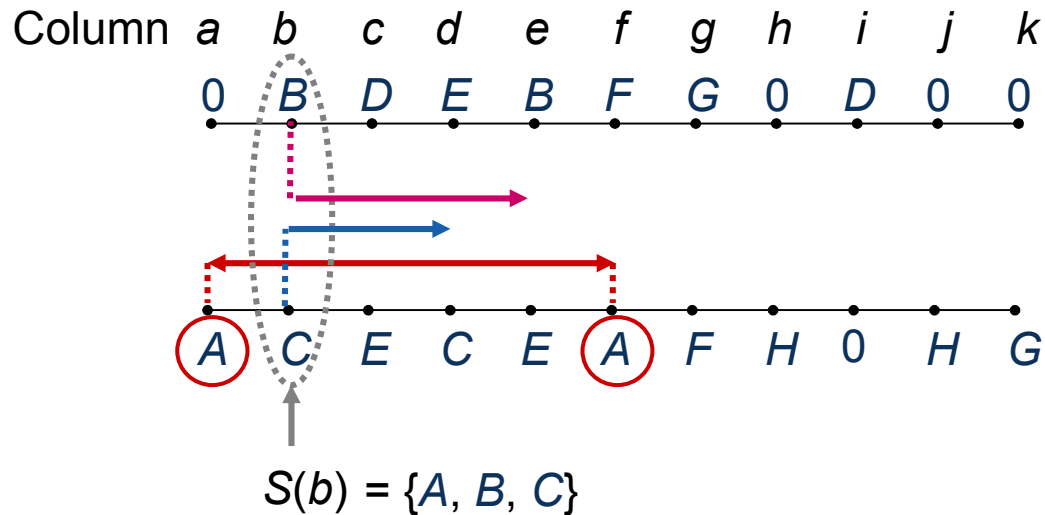
# Detailed Routing

Channel and Switchbox Routing



Adapted from [Khang'11]

# Horizontal Constraint Graphs

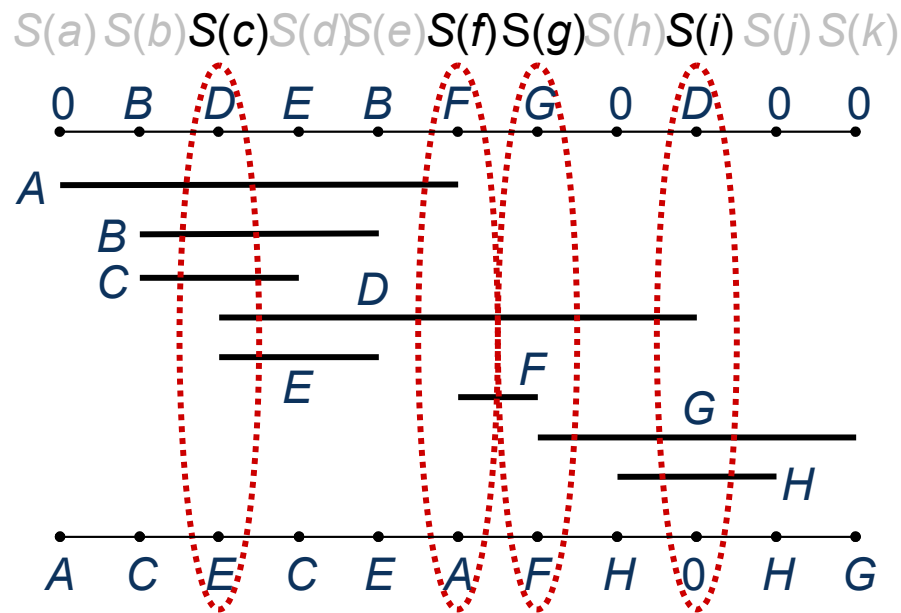
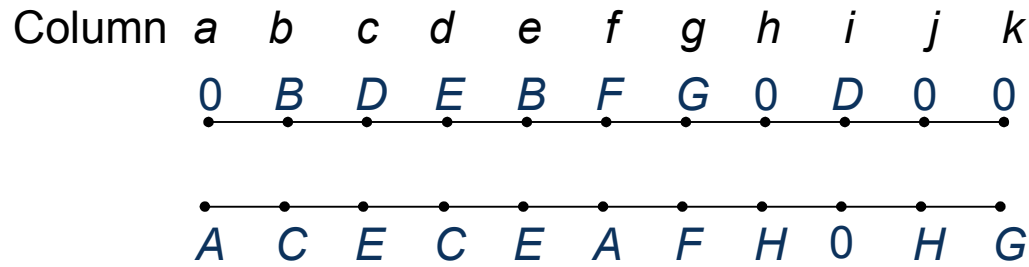


- Let  $S(col)$  denote the set of nets that pass through column  $col$
- $S(col)$  contains all nets that either (1) are connected to a pin in column  $col$  or (2) have pin connections to both the left and right of  $col$
- Since horizontal segments cannot overlap, each net in  $S(col)$  must be assigned to a different track in column  $col$
- $S(col)$  represents the lower bound on the number of tracks in column  $col$ ; lower bound of the channel height is given by maximum cardinality of any  $S(col)$

Adapted from [Khang'11]



# Horizontal Constraint Graph Example



Focus on columns  
which are not subsets  
of any other column

$$S(a) = \{A\}$$

$$S(b) = \{A, B, C\}$$

$$S(c) = \{A, B, C, D, E\}$$

$$S(d) = \{A, B, C, D, E\}$$

$$S(e) = \{A, B, D, E\}$$

$$S(f) = \{A, D, F\}$$

$$S(g) = \{D, F, G\}$$

$$S(h) = \{D, G, H\}$$

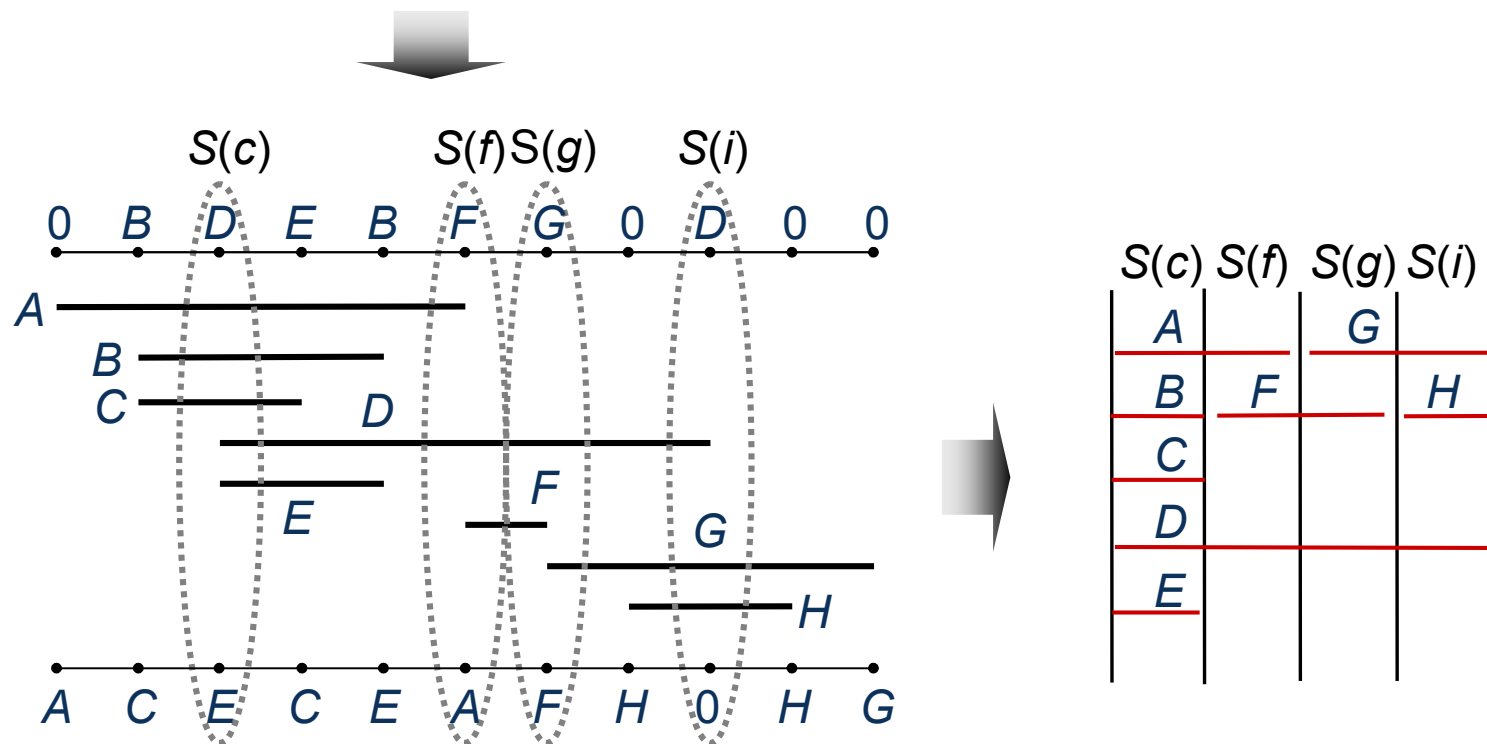
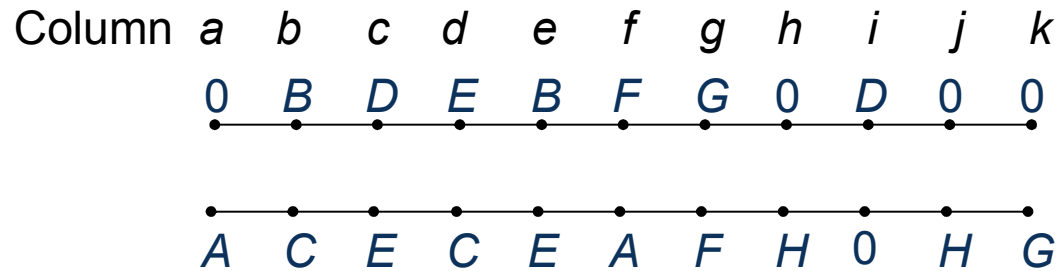
$$S(i) = \{D, G, H\}$$

$$S(j) = \{G, H\}$$

$$S(k) = \{G\}$$

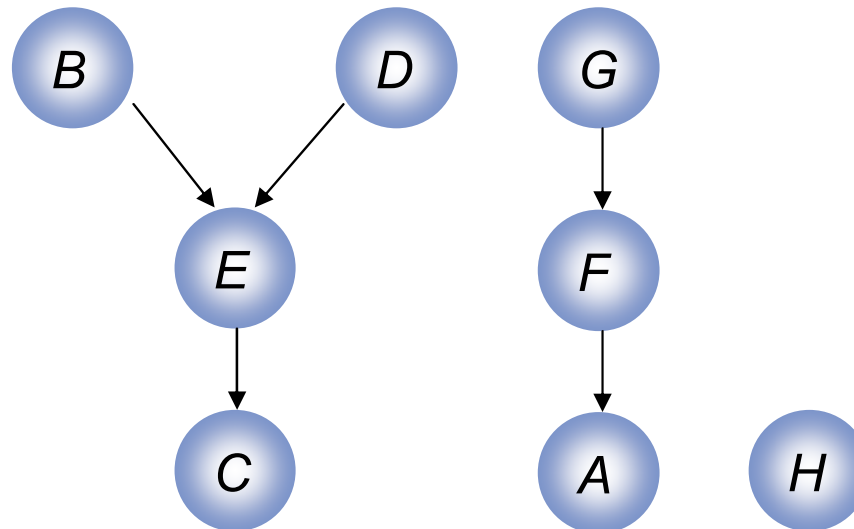
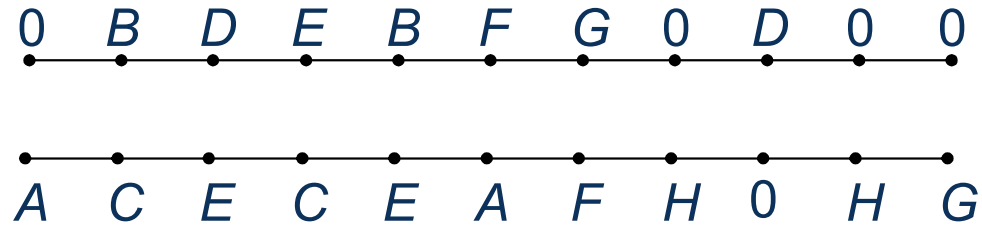
Adapted from [Khang'11]

# Horizontal Constraint Graph Example



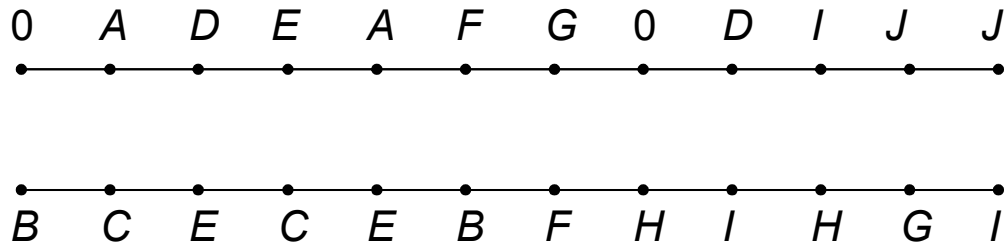
Adapted from [Khang'11]

# Vertical Constraint Graphs

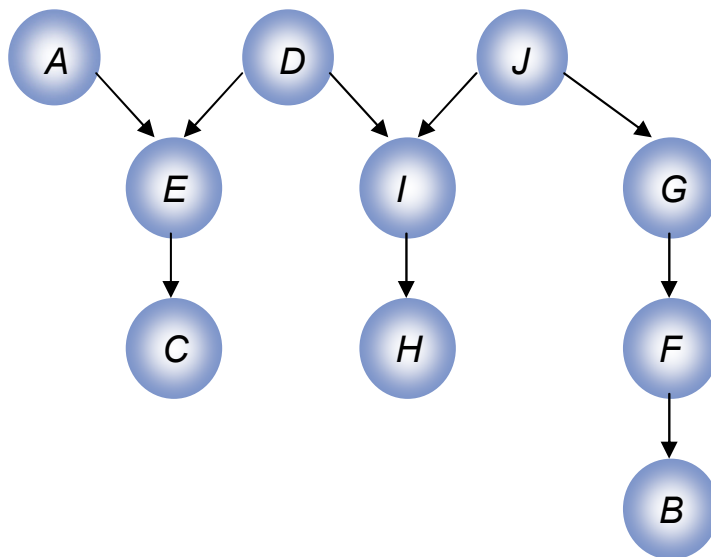


Adapted from [Khang'11]

# Left-Edge Algorithm Example



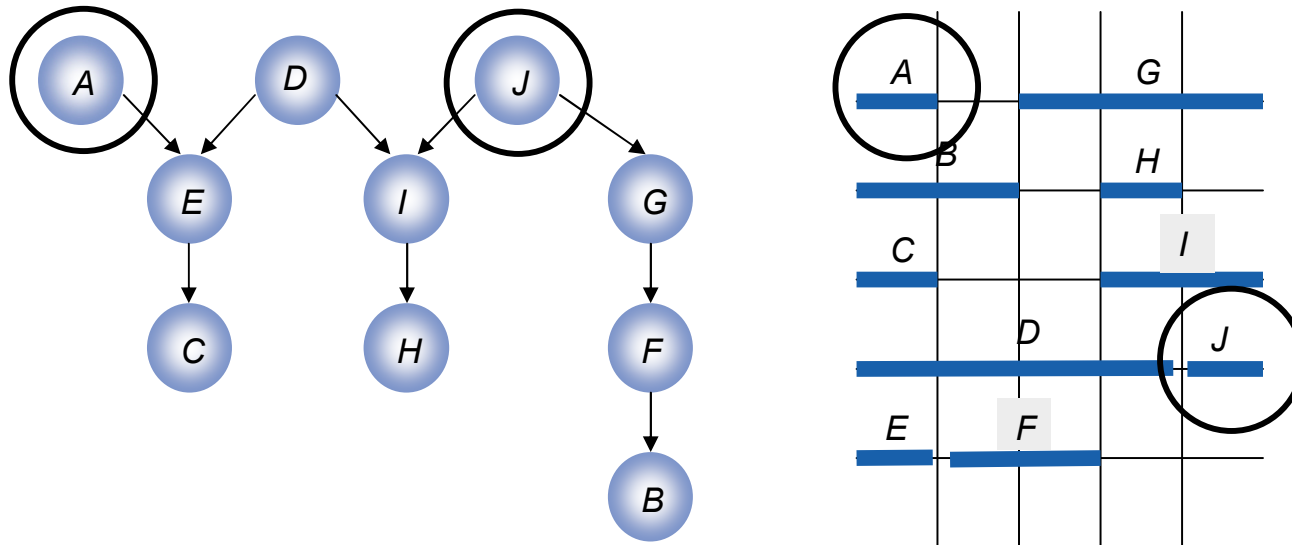
1. Generate VCG and zone representation



A			G	
	B		H	
C			I	
		D		J
E		F		

Adapted from [Khang'11]

# Left-Edge Algorithm Example



2. Consider next track
3. Find left-to-right ordering of all unassigned nets  
If *curr\_net* has no parents and does not cause conflicts on *curr\_track*  
assign *curr\_net*

*curr\_track* = 1: Net A Net J

4. Delete placed nets (A, J) in VCG and zone representation

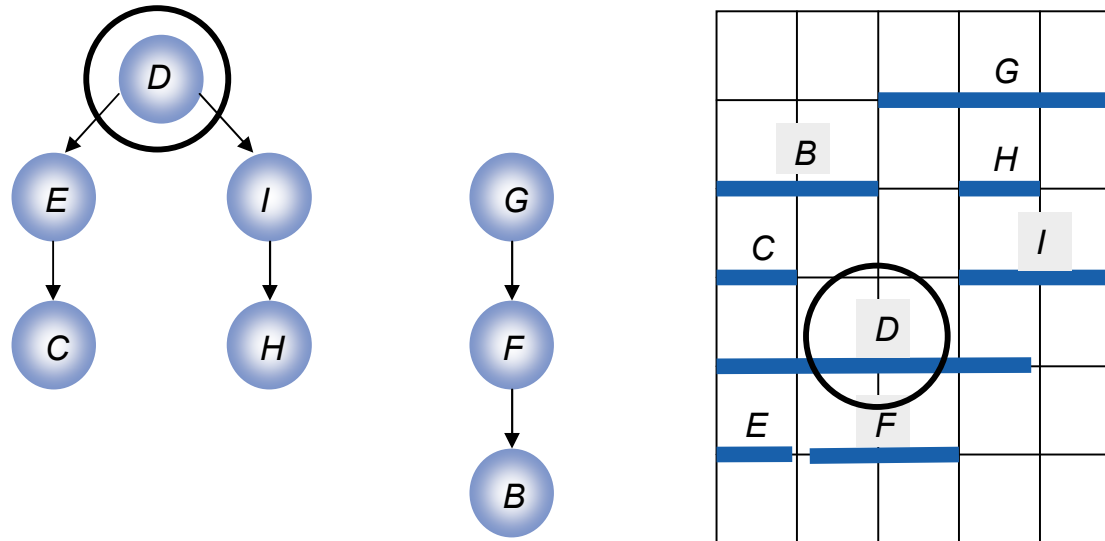
Adapted from [Khang'11]

# Left-Edge Algorithm Example



Adapted from [Khang'11]

# Left-Edge Algorithm Example



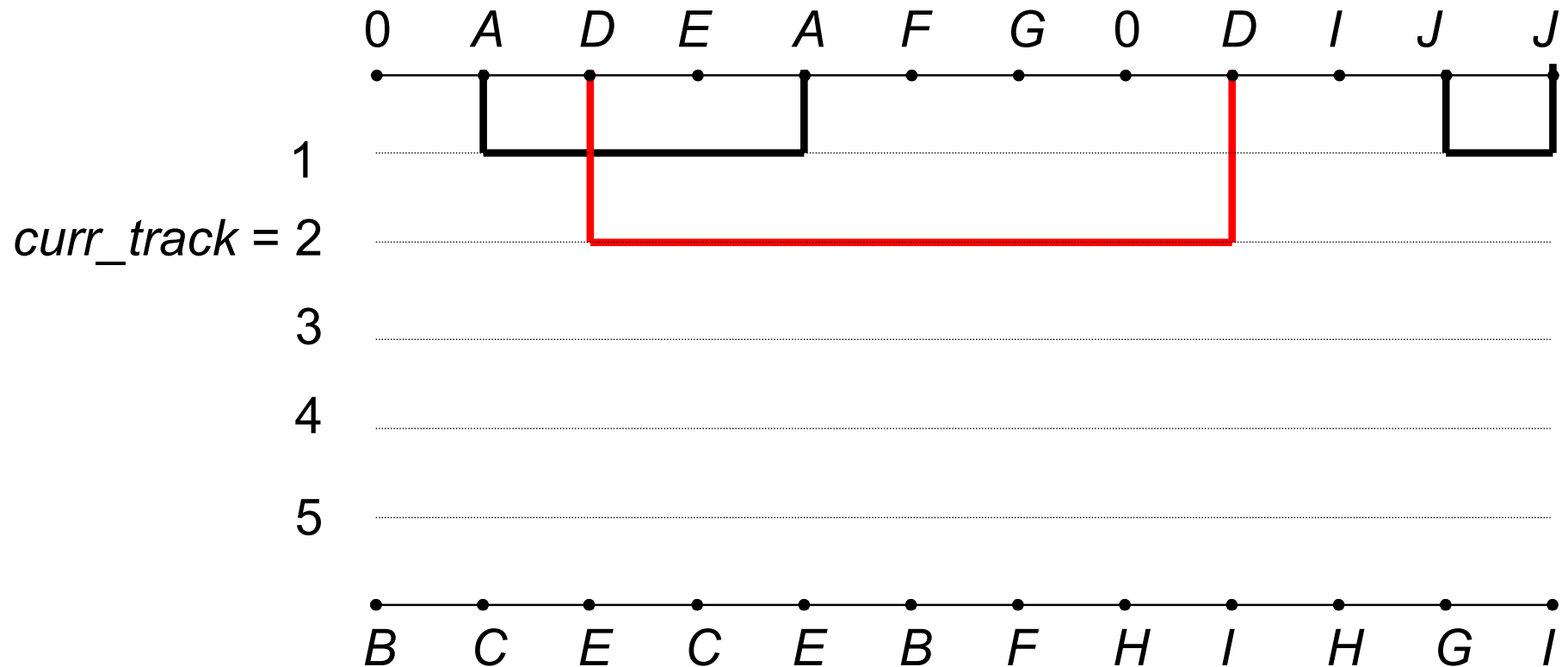
2. Consider next track
3. Find left-to-right ordering of all unassigned nets  
If *curr\_net* has no parents and does not cause conflicts on *curr\_track* assign *curr\_net*

*curr\_track* = 2: Net D

4. Delete placed nets (*D*) in VCG and zone representation

Adapted from [Khang'11]

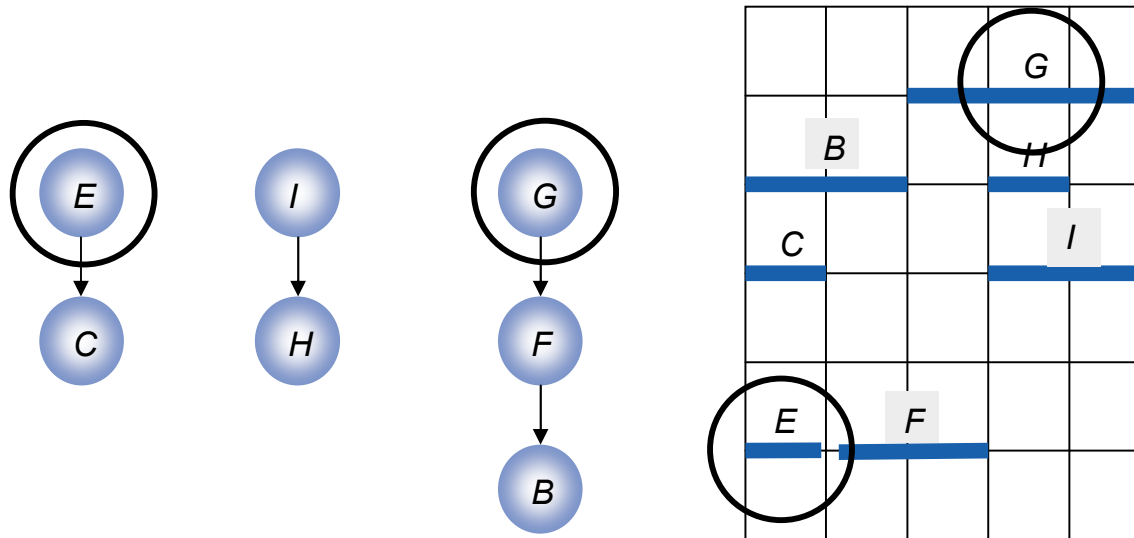
# Left-Edge Algorithm Example



Adapted from [Khang'11]



# Left-Edge Algorithm Example



2. Consider next track
3. Find left-to-right ordering of all unassigned nets  
If *curr\_net* has no parents and does not cause conflicts on *curr\_track*  
assign *curr\_net*

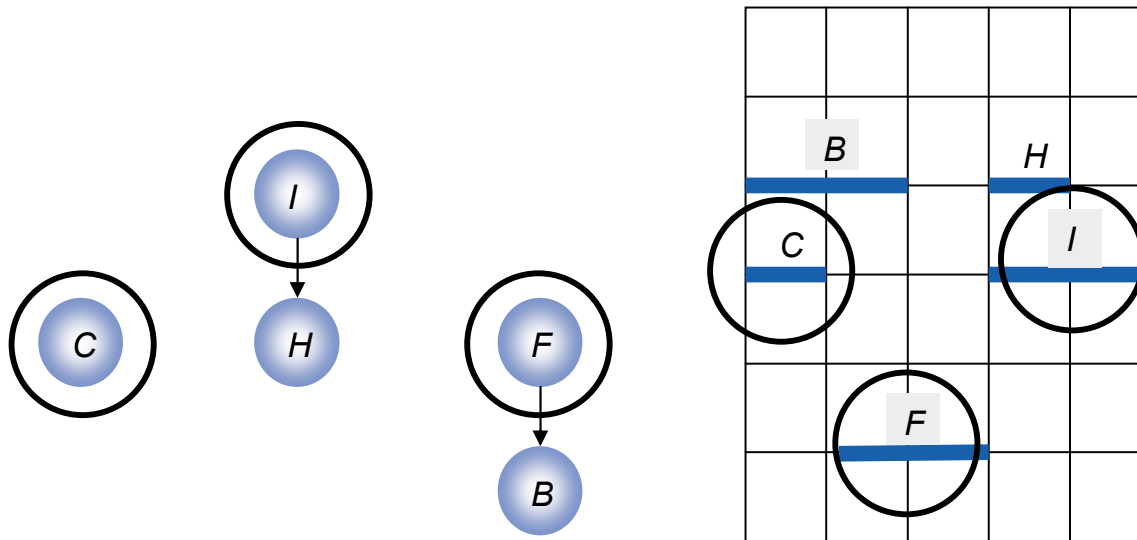
*curr\_track* = 3: Net E Net G

4. Delete placed nets (**E**, **G**) in VCG and zone representation

Adapted from [Khang'11]



# Left-Edge Algorithm Example



2. Consider next track
3. Find left-to-right ordering of all unassigned nets  
If *curr\_net* has no parents and does not cause conflicts on *curr\_track* assign *curr\_net*

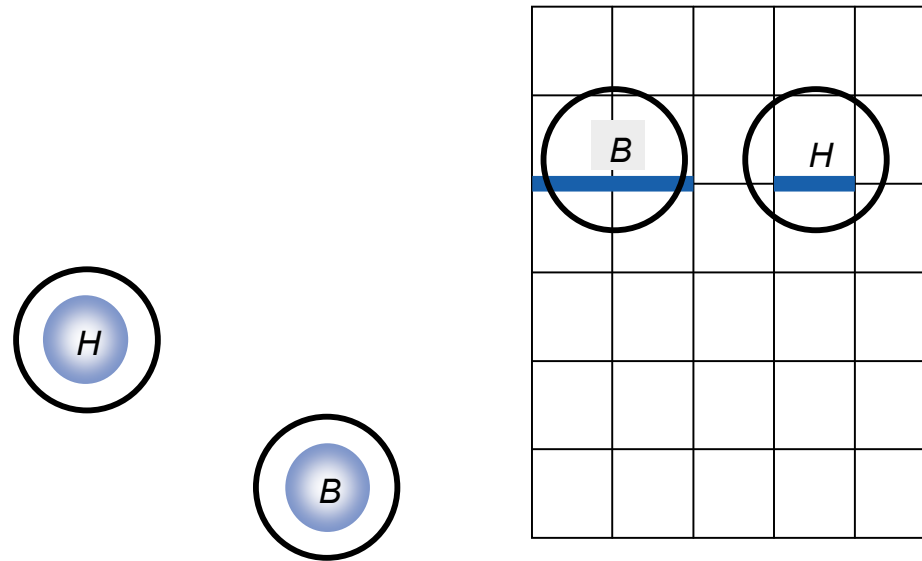
*curr\_track* = 4: Net C Net F Net I

4. Delete placed nets (C, F, I) in VCG and zone representation

Adapted from [Khang'11]



# Left-Edge Algorithm Example



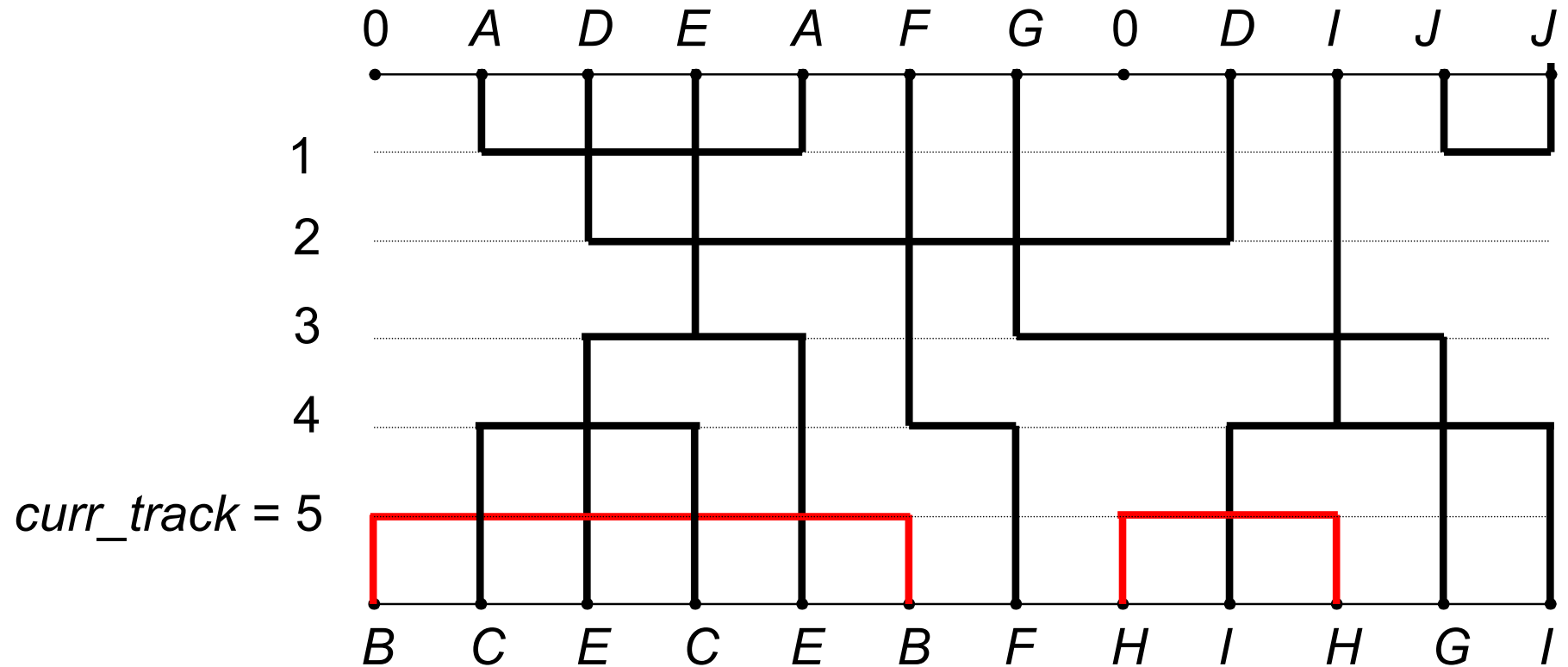
2. Consider next track
3. Find left-to-right ordering of all unassigned nets  
If *curr\_net* has no parents and does not cause conflicts on *curr\_track*  
assign *curr\_net*

*curr\_track* = 5: Net *B* Net *H*

4. Delete placed nets (*B*, *H*) in VCG and zone representation

Adapted from [Khang'11]

# Left-Edge Algorithm Example



Adapted from [Khang'11]

# Acknowledgments

---

- ▶ [Devadas'06] S. Devadas, "VLSI CAD Flow: Logic Synthesis, Placement, and Routing," MIT 6.375 Complex Digital Systems Guest Lecture Slides, 2006.
- ▶ [Kahng'11] A.B. Kahng, J. Liening, I.L. Markov, and J. Hu. Companion Slides for "VLSI Physical Design: From Graph Partitioning to Timing Closure," Springer 2011.