Detail Routing

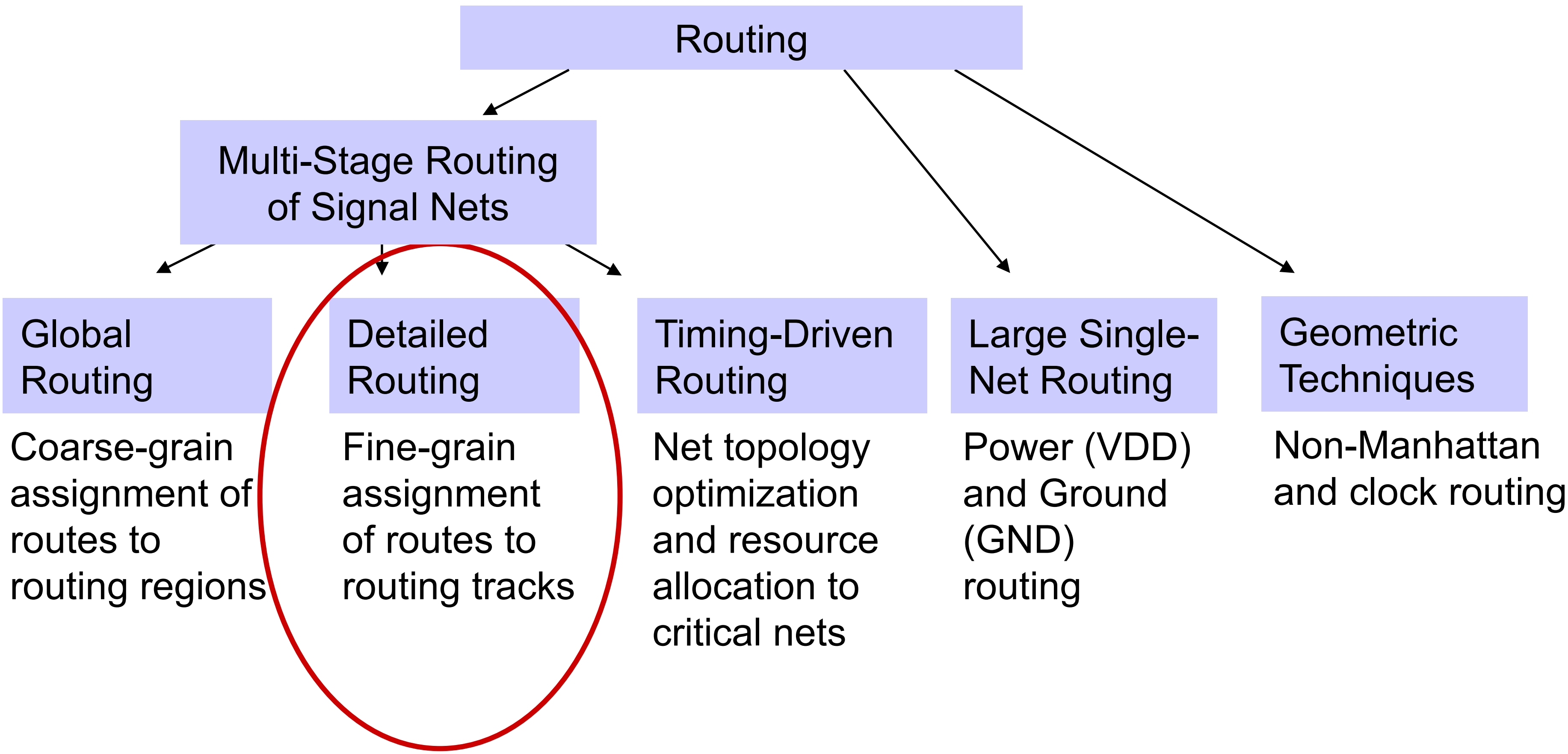
Presented By:

Sridhar H Rangarajan

IBM STG India Enterprise Systems Development

Detailed Routing

- Terminology
- Horizontal and Vertical Constraint Graphs
  - Horizontal Constraint Graphs
  - Vertical Constraint Graphs
- Channel Routing Algorithms
  - Left-Edge Algorithm
  - Dogleg Routing
- Switchbox Routing
  - Terminology
  - Switchbox Routing Algorithms
- Over-the-Cell Routing Algorithms
  - OTC Routing Methodology
  - OTC Routing Algorithms
- Modern Challenges in Detailed Routing


Routing
Multi-Stage Routing of Signal Nets
Global Routing
Coarse-grain assignment of routes to routing regions
Detailed Routing
Fine-grain assignment of routes to routing tracks
Timing-Driven Routing
Net topology optimization and resource allocation to critical nets
Large Single-Net Routing
Power (VDD) and Ground (GND) routing
Geometric Techniques
Non-Manhattan and clock routing

Detailed Routing

- The objective of detailed routing is to assign route segments of signal nets to specific routing tracks, vias, and metal layers in a manner consistent with given global routes of those nets

- Similar to global routing
  - Use physical wires to do connections
  - Estimating the wire resistance and capacitance, which determines whether the design meets timing requirements

- Detailed routing techniques are applied within routing regions, such as
  - Channels
  - switchboxes, and
  - global routing cells

- Detailed routers must account for
  - manufacturing rules and
  - the impact of manufacturing faults

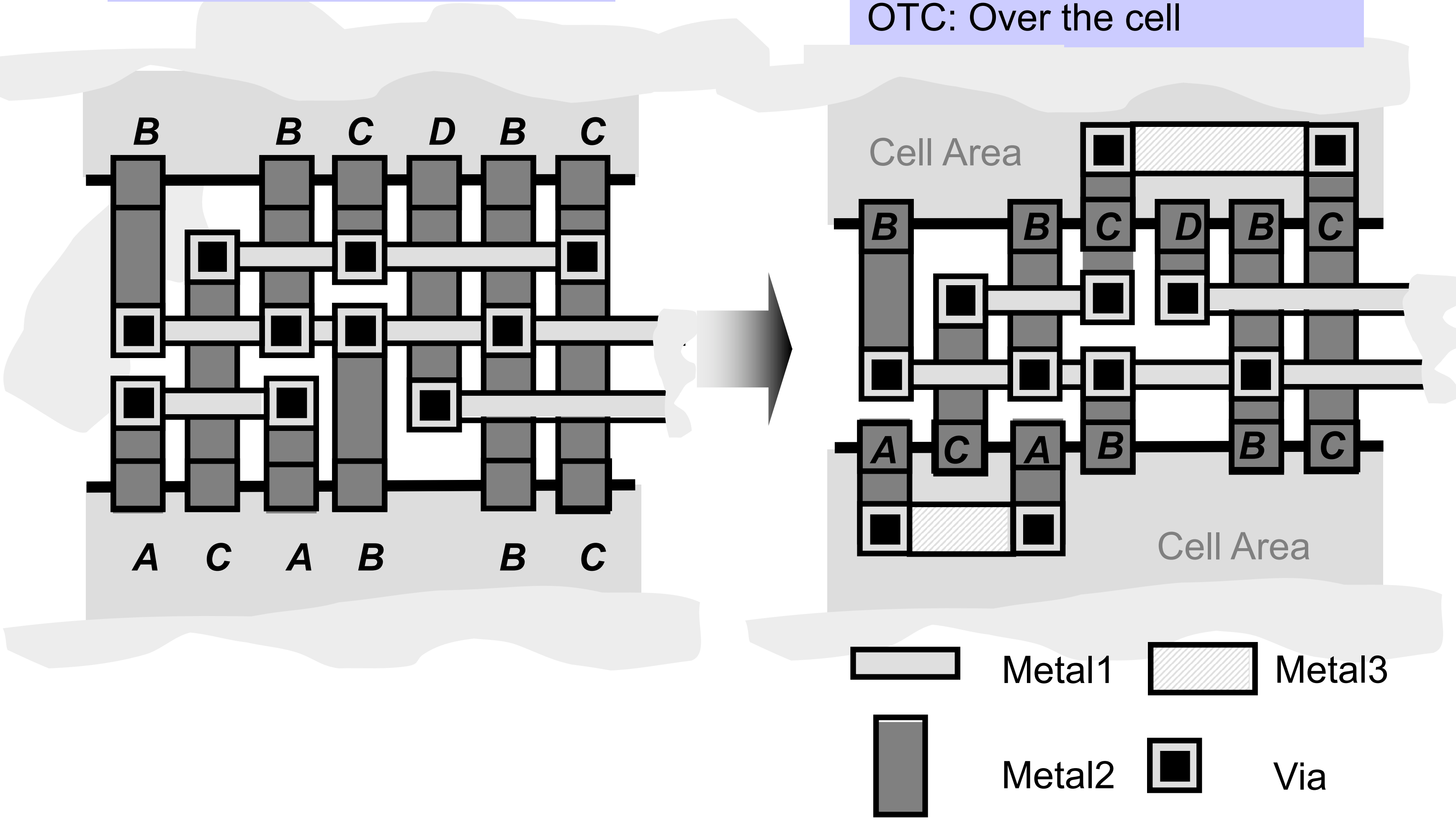
Detailed Routing

- Detailed Routing Stages
  - Assign routing tracks
  - Perform entire routing – no open connection left
  - Search and repair – resolving all the physical design rules
  - Perform optimizations, e.g. add redundant vias (reduce resistivity, better yield)

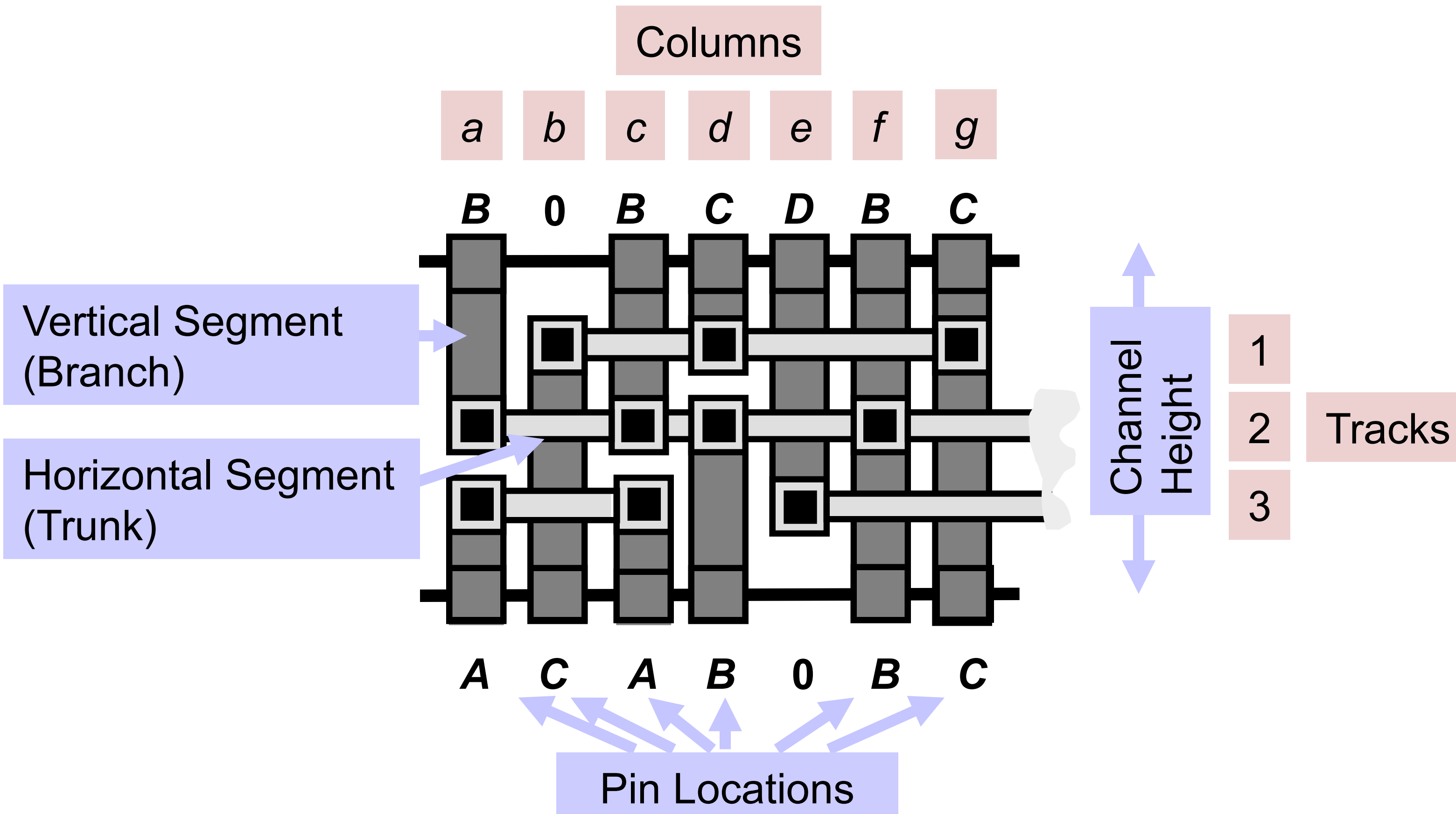
Terminology

Two-Layer Channel Routing

Three-Layer OTC Routing
OTC: Over the cell



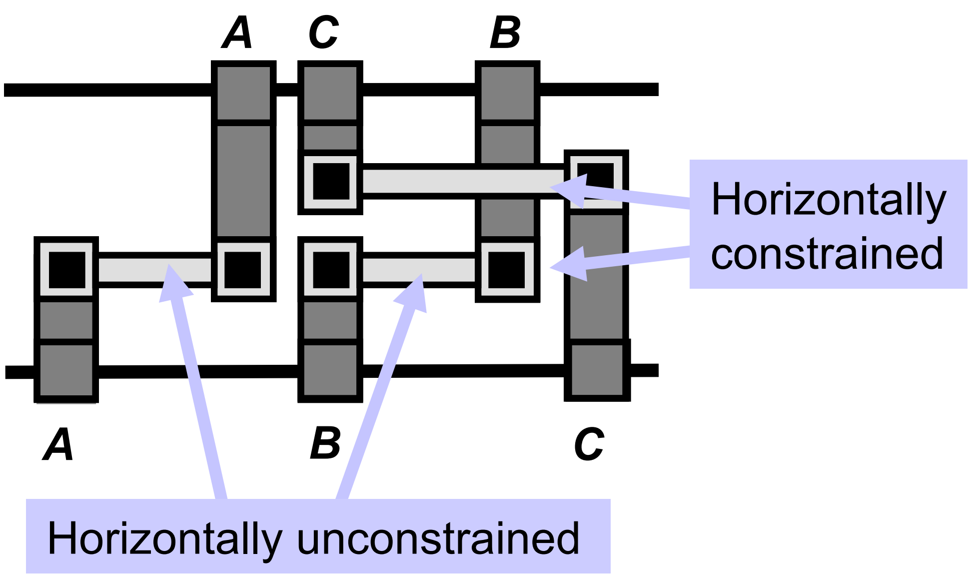
Terminology



Terminology

Horizontal Constraint

- Assumption: one layer for horizontal routing
- A horizontal constraint exists between two nets if their horizontal segments overlap

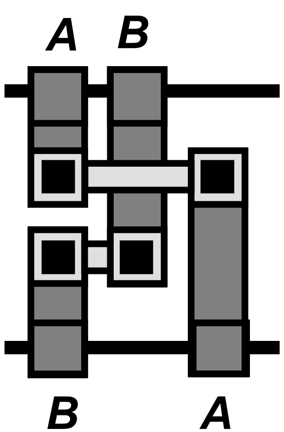


Terminology

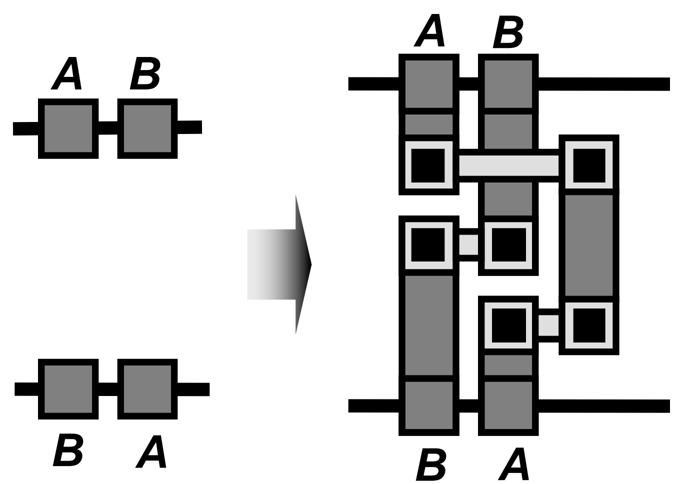
Vertical Constraint

- A vertical constraint exists between two nets if they have pins in the same column

⇒ The vertical segment coming from the top must “stop” before overlapping with the vertical segment coming from the bottom in the same column



Vertically constrained without conflict

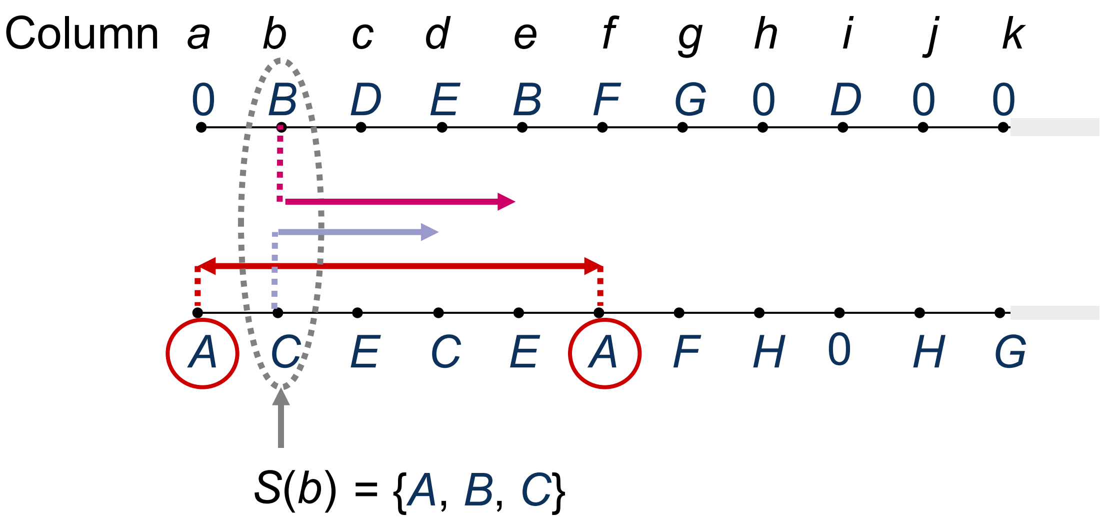


Vertically constrained with a vertical conflict

Horizontal and Vertical Constraint Graphs

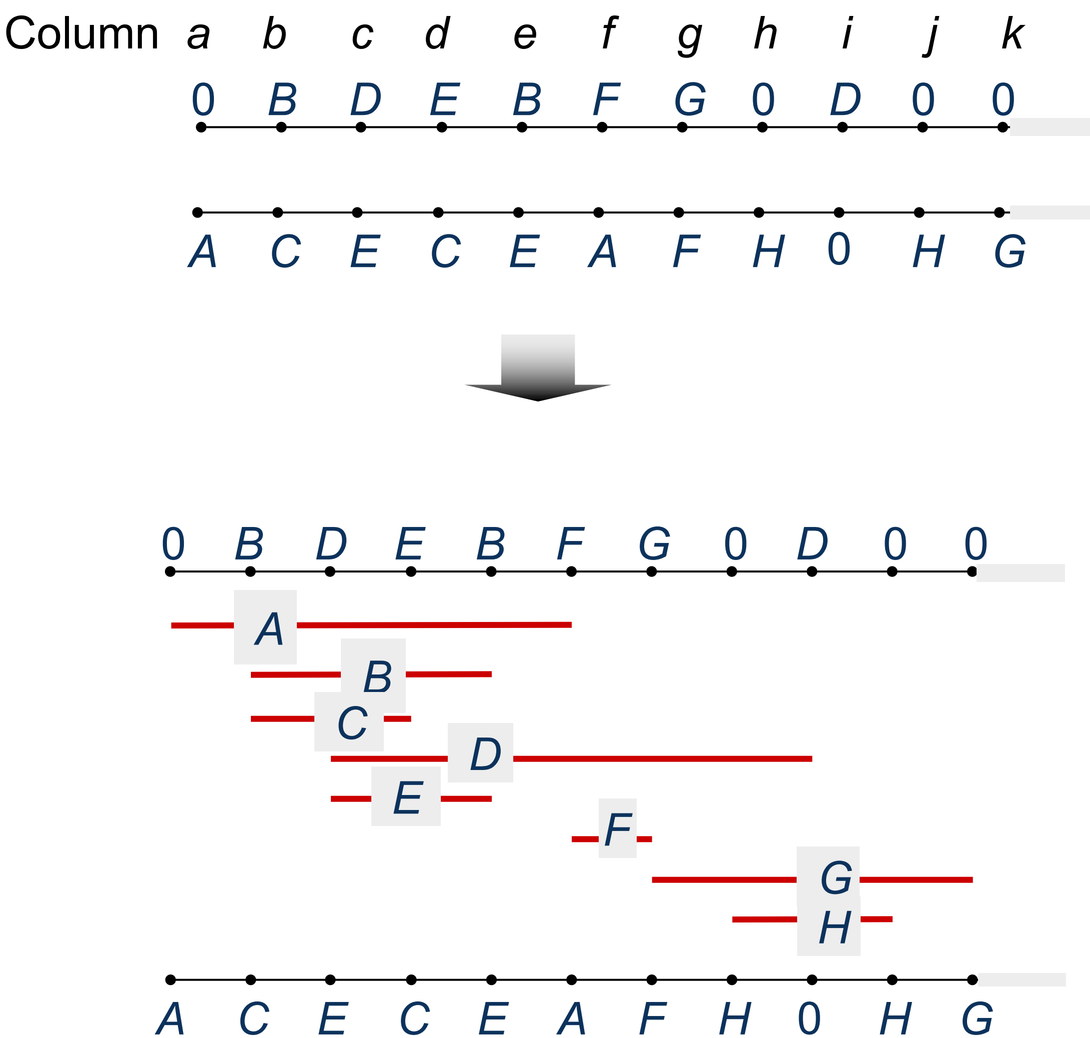
- The relative positions of nets in a channel routing instance can be modeled by horizontal and vertical constraint graphs
- These graphs are used to
  - initially predict the minimum number of tracks that are required
  - detect potential routing conflicts

Horizontal Constraint Graphs

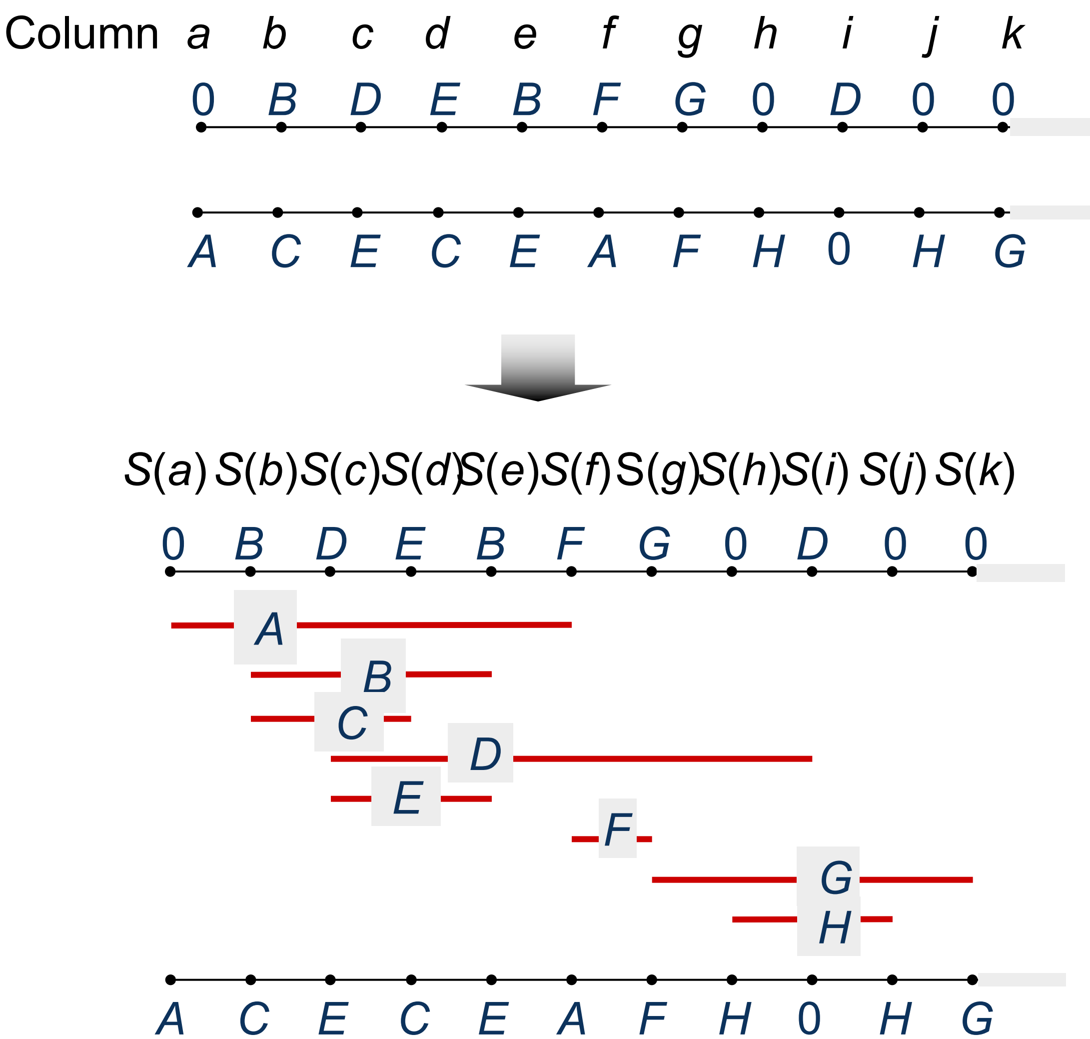


- Let *S*(*col*) denote the set of nets that pass through column *col*
- *S*(*col*) contains all nets that either (1) are connected to a pin in column *col* or (2) have pin connections to both the left and right of *col*
- Since horizontal segments cannot overlap, each net in *S*(*col*) must be assigned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

Horizontal Constraint Graphs



Horizontal Constraint Graphs



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

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

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

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

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

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

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

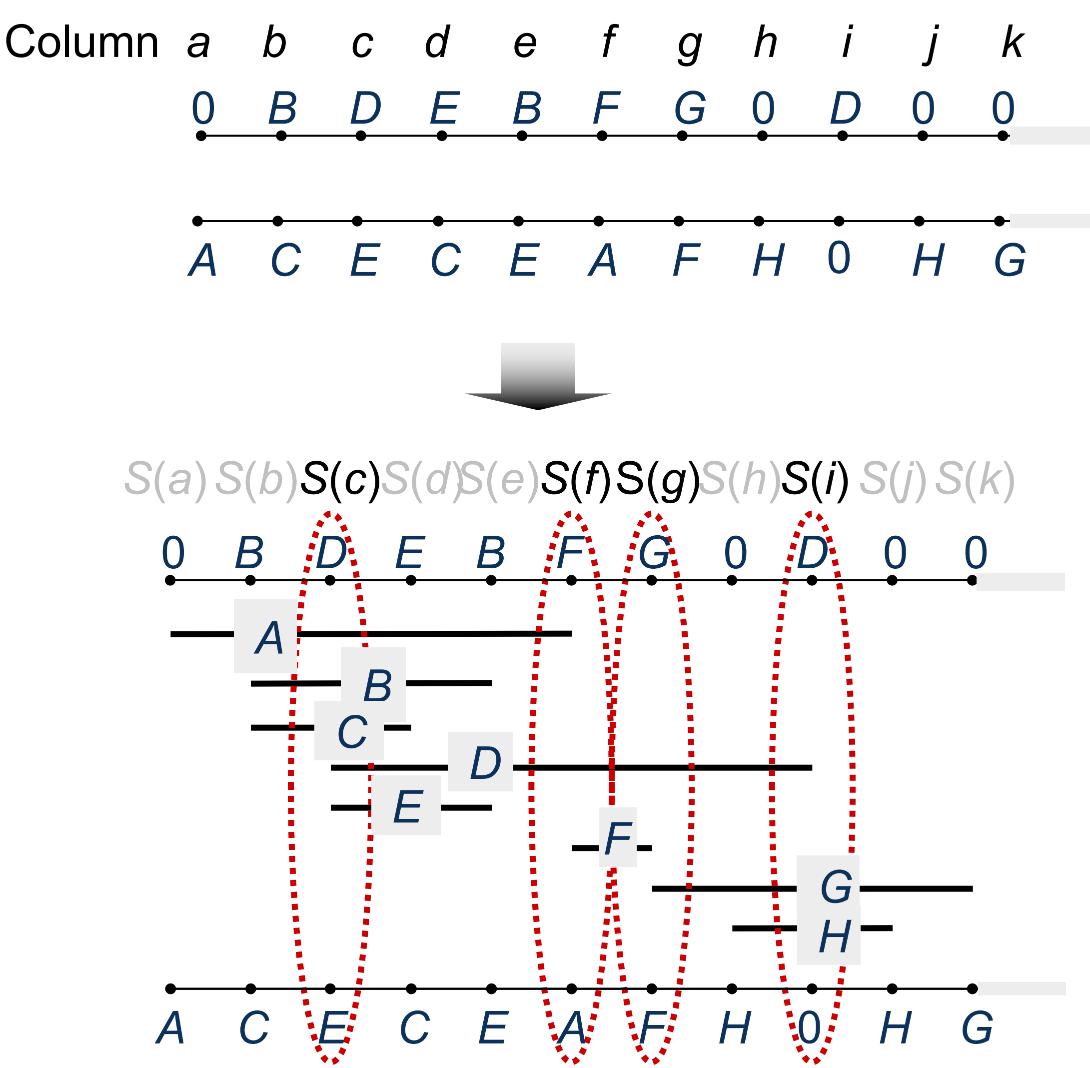
$S(h) = \{D,G,H\}$

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

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

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

Horizontal Constraint Graphs



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

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

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

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

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

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

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

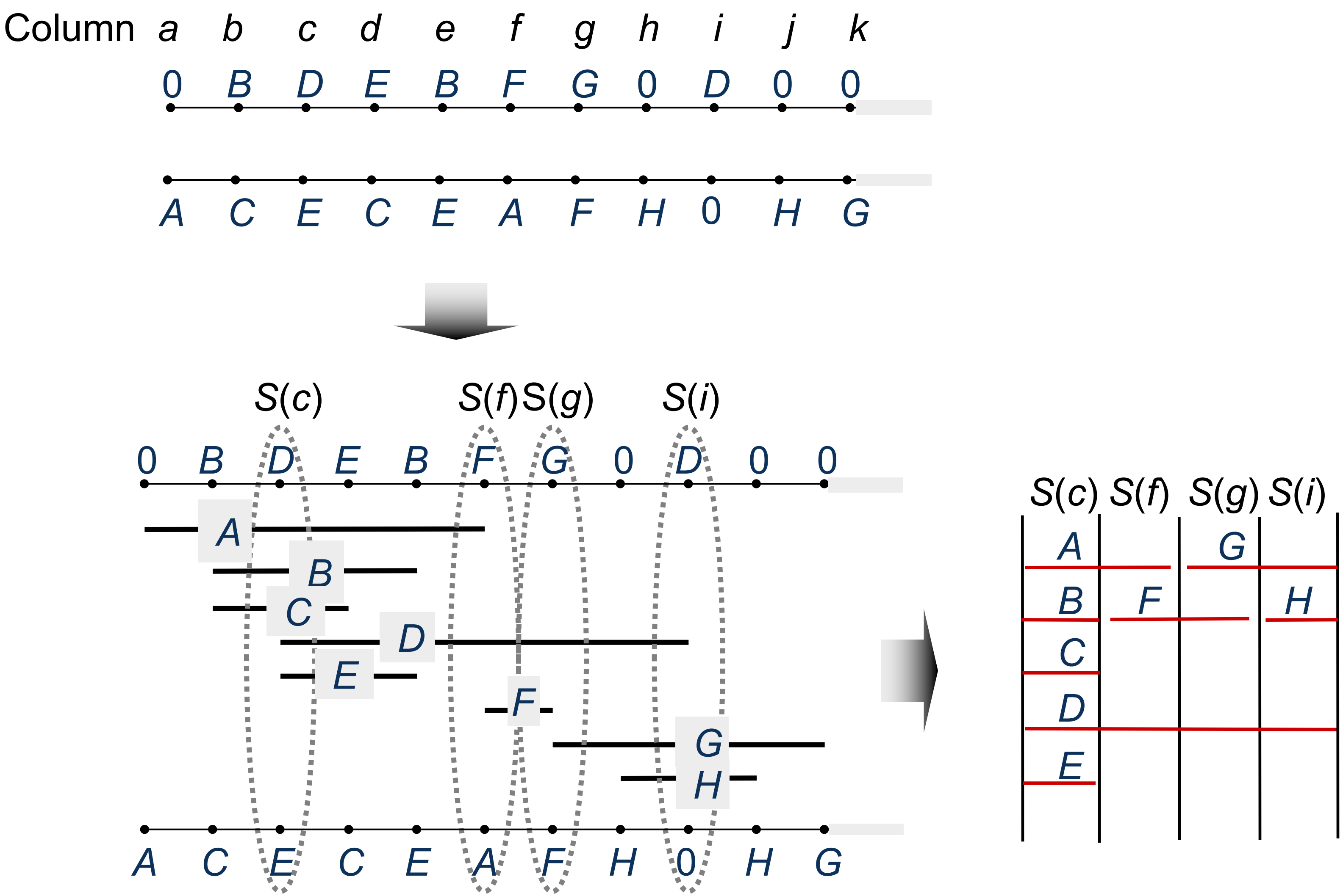
$S(h) = \{D,G,H\}$

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

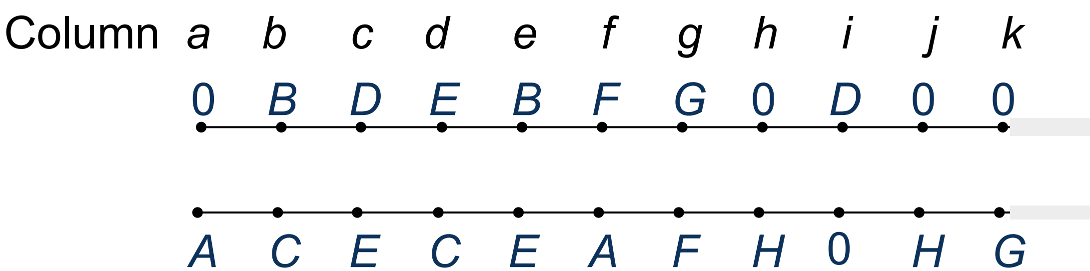
$S(j) = \{G,H\}$

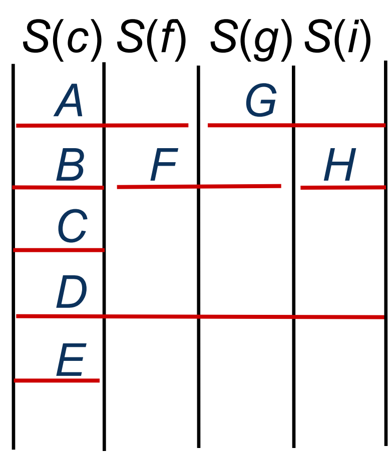
$S(k) = \{G\}$

Horizontal Constraint Graphs



Horizontal Constraint Graphs



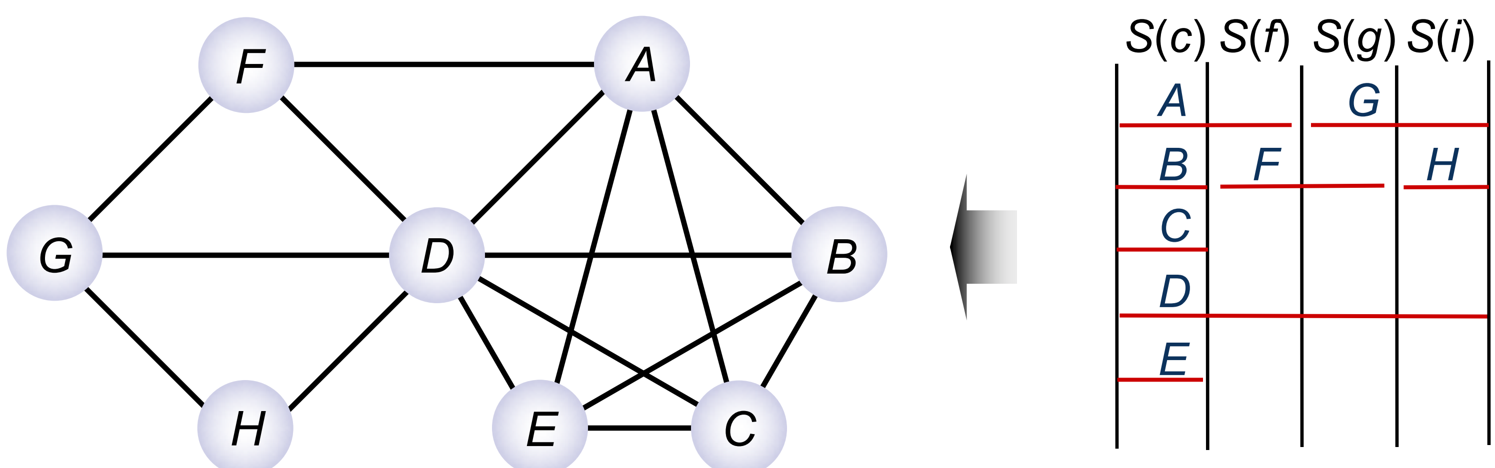


Lower bound on the number of tracks = 5

Horizontal Constraint Graphs

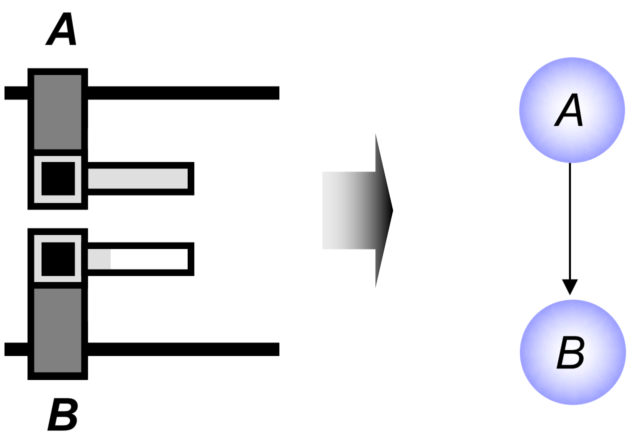
Column	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*
	0	*B*	*D*	*E*	*B*	*F*	*G*	0	*D*	0	0
	A	*C*	*E*	*C*	*E*	*A*	*F*	H	0	*H*	*G*

Graphical Representation

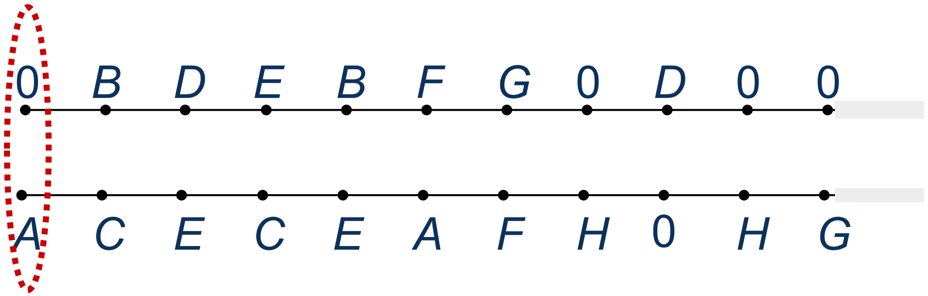


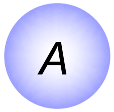
Vertical Constraint Graphs

- A directed edge $e(i,j) \in E$ connects nodes *i* and *j* if the horizontal segment of net *i* must be located above net *j*

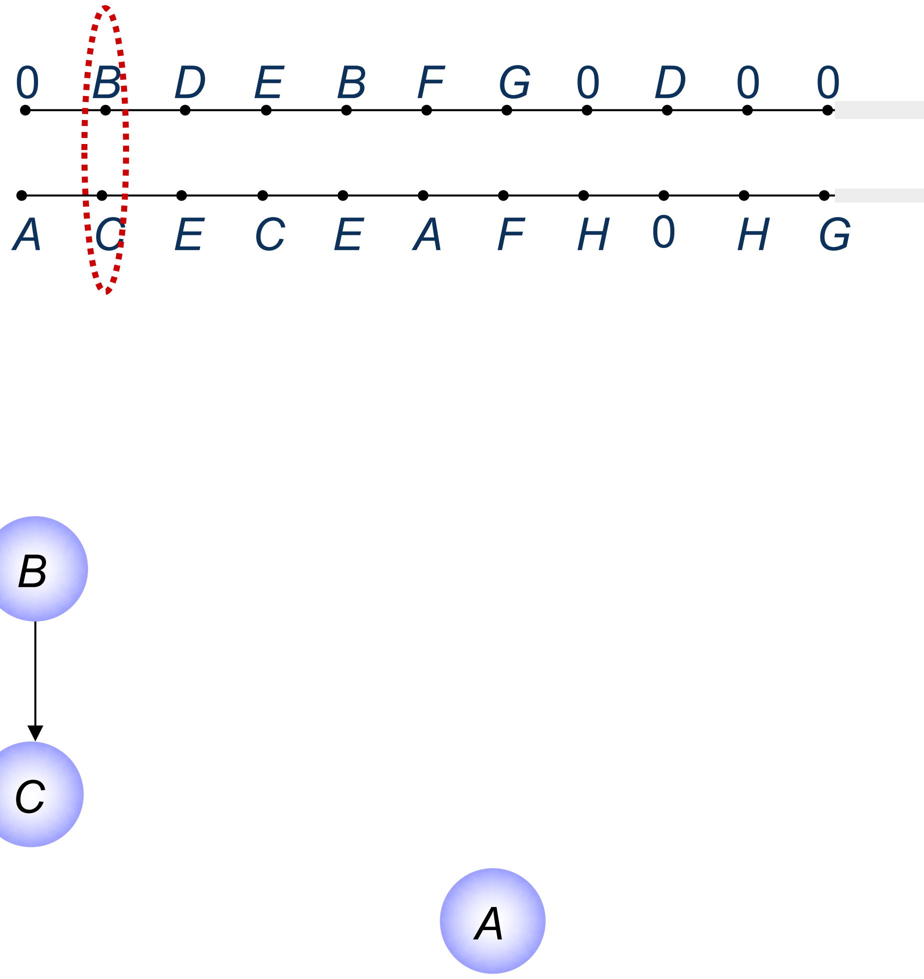


Vertical Constraint Graphs

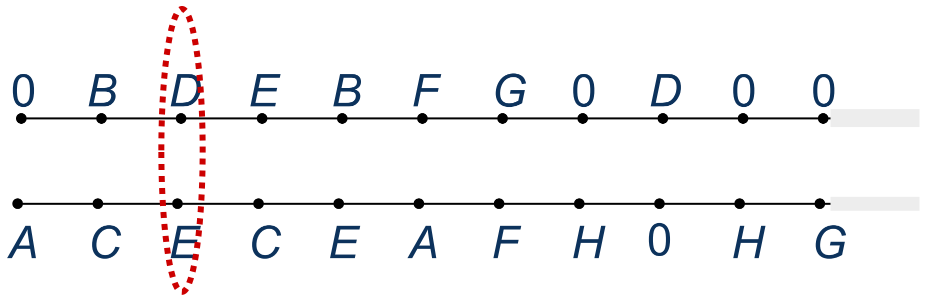


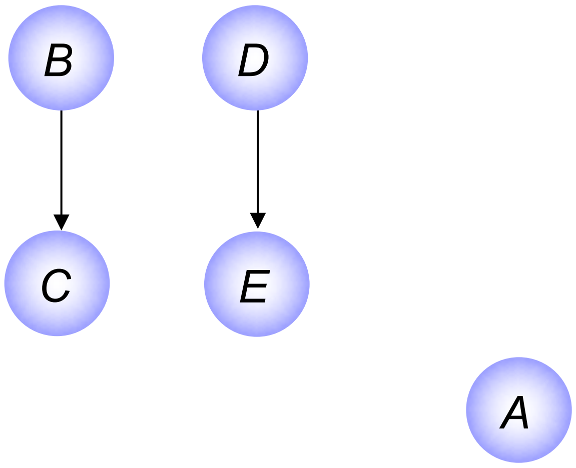


Vertical Constraint Graphs

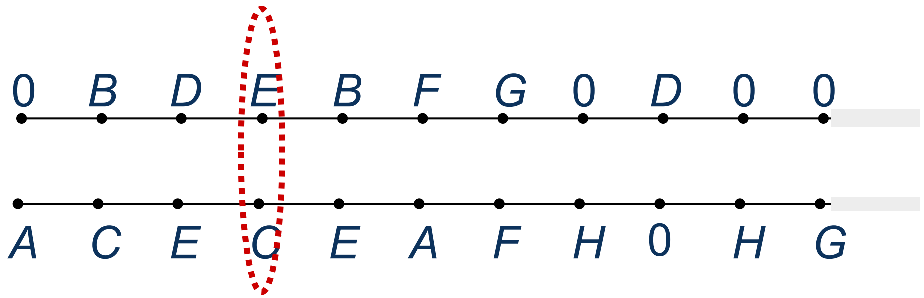


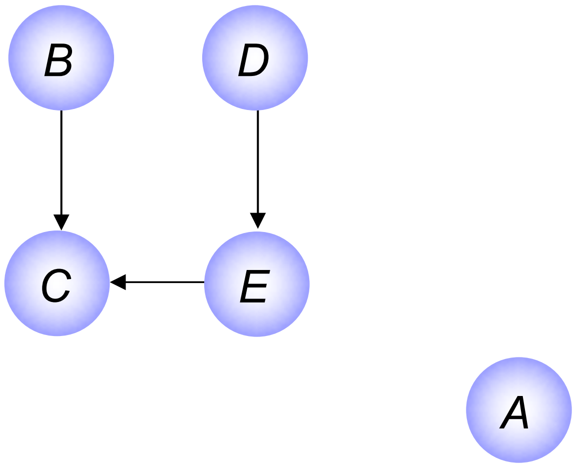
Vertical Constraint Graphs



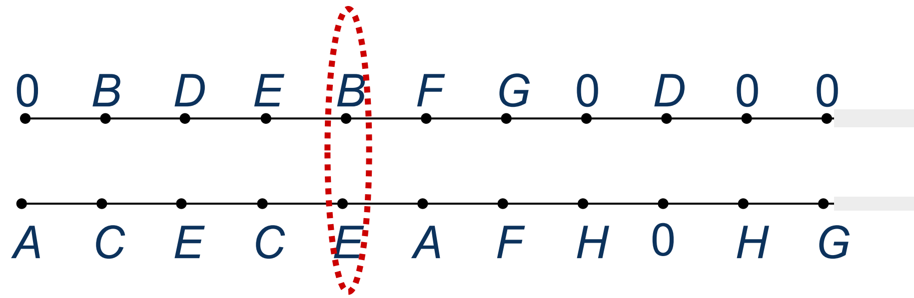


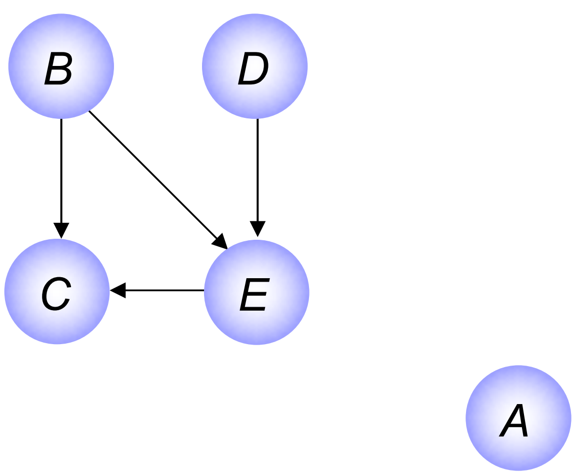
Vertical Constraint Graphs





Vertical Constraint Graphs

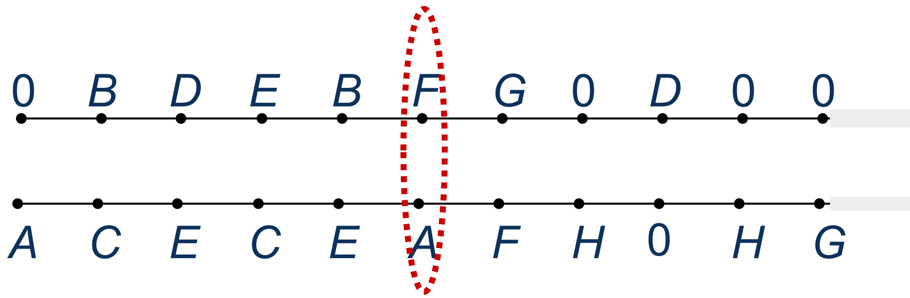


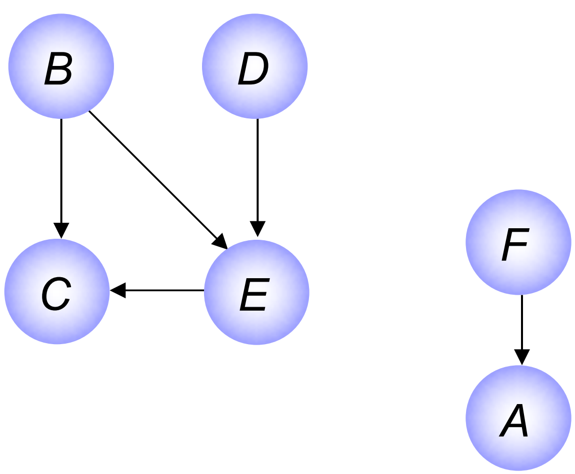


Vertical Constraint Graph (VCG)

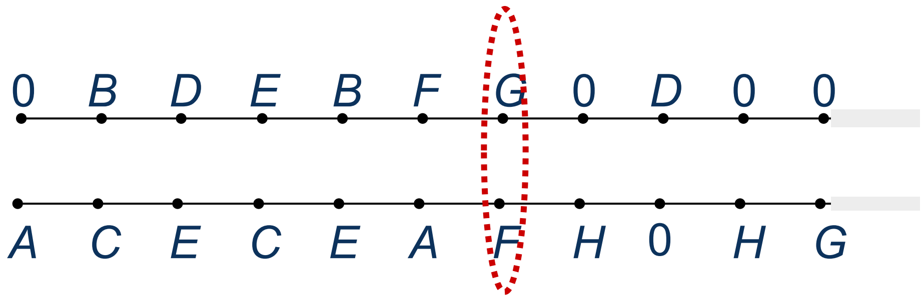
Note: an edge that can be derived by transitivity is not included, such as edge (*B*,*C*)

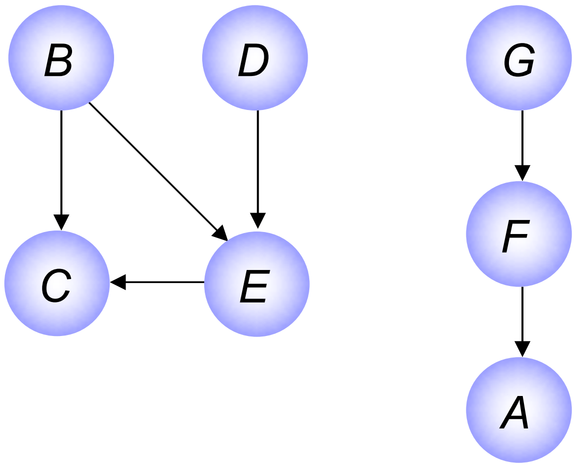
Vertical Constraint Graphs



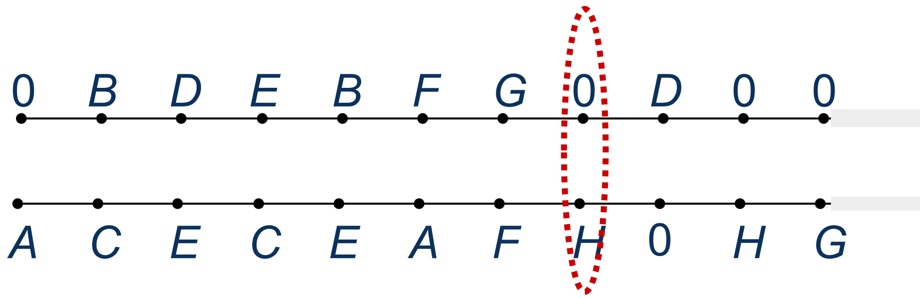


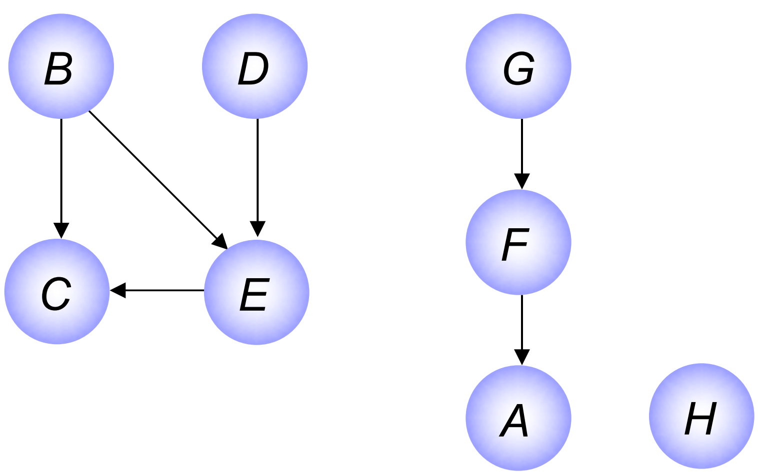
Vertical Constraint Graphs



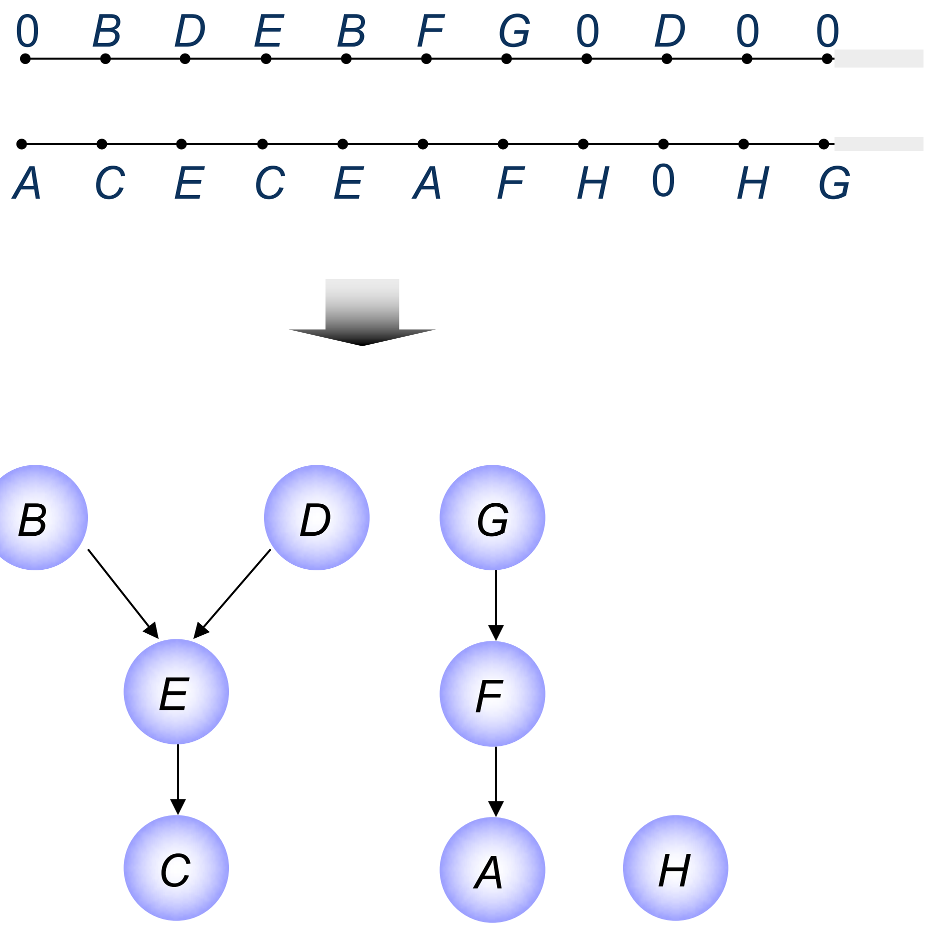


Vertical Constraint Graphs

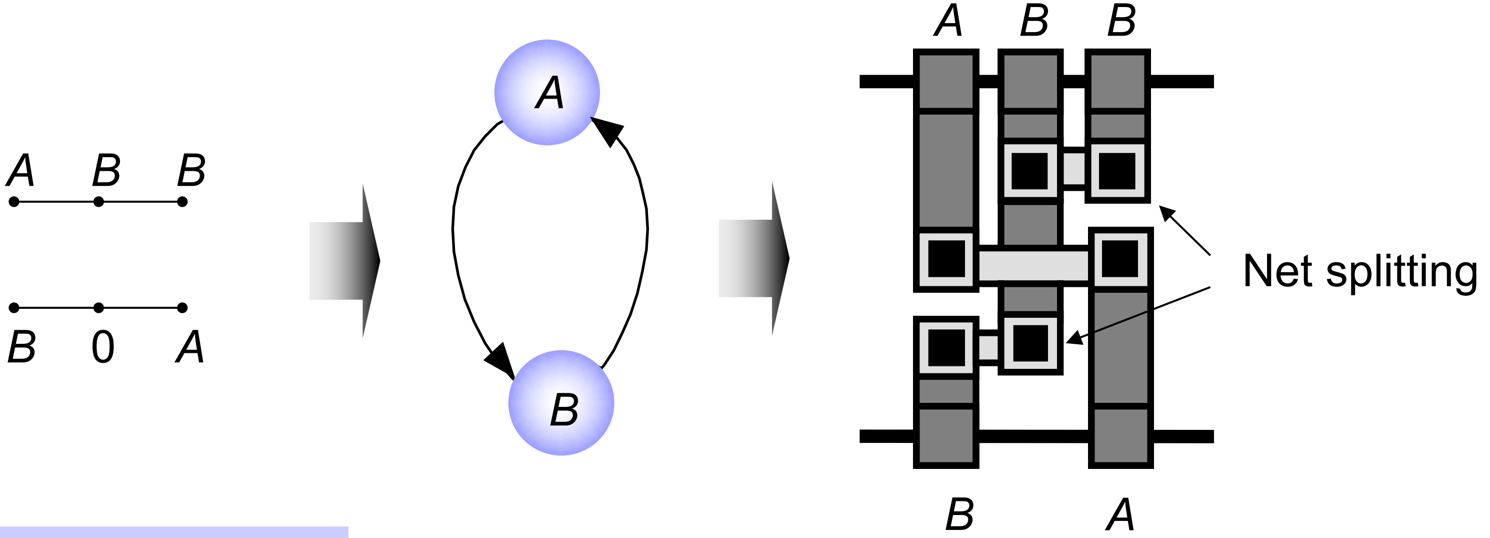




Vertical Constraint Graphs



Vertical Constraint Graphs



Cyclic conflict

Left-Edge Algorithm

- Based on the VCG and the zone representation, greedily maximizes the usage of each track
  - VCG: assignment order of nets to tracks
  - Zone representation: determines which nets may share the same track
- Each net uses only one horizontal segment (trunk)

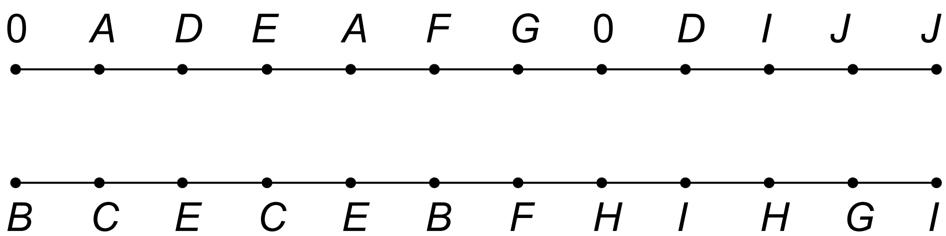
Left-Edge Algorithm

Input: channel routing instance *CR*

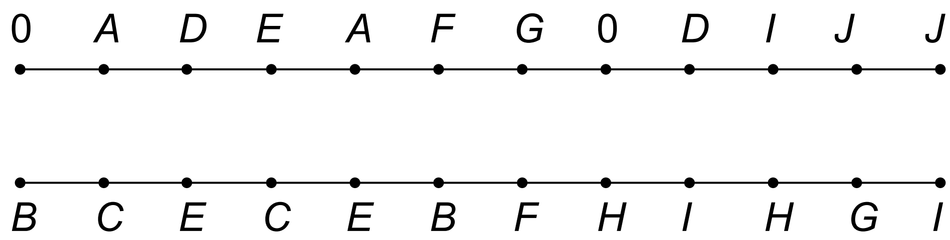
Output: track assignments for each net

```
curr_track = 1                              // start with topmost track
nets_unassigned = Netlist
while (nets_unassigned != Ø)                // while nets still unassigned
   VCG = VCG(CR)                            // generate VCG and zone
   ZR = ZONE_REP(CR)                        //   representation
   SORT(nets_unassigned,start column)       // find left-to-right ordering
                                            //   of all unassigned nets
   for (i =1 to |nets_unassigned|)
      curr_net = nets_unassigned[i]
      if (PARENTS(curr_net) == Ø &&         // if curr_net has no parent
         (TRY_ASSIGN(curr_net,curr_track))  //   and does not cause
                                            //   conflicts on curr_track,
         ASSIGN(curr_net,curr_track)        //   assign curr_net
         REMOVE(nets_unassigned,curr_net)
   curr_track = curr_track + 1              // consider next track
```

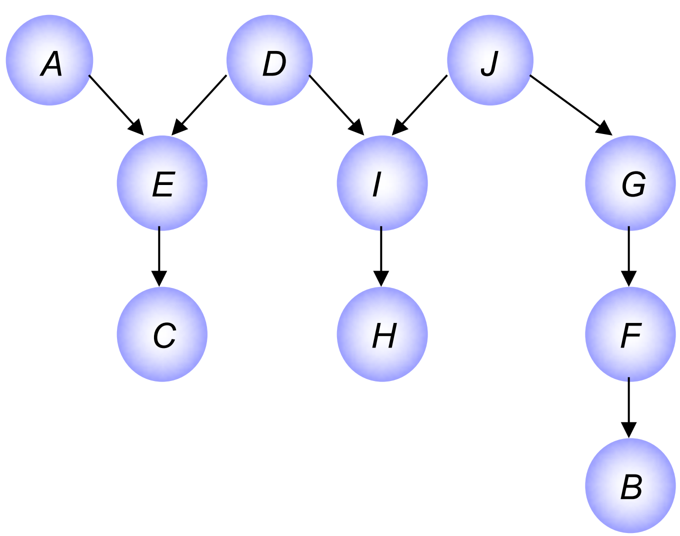
Left-Edge Algorithm – Example

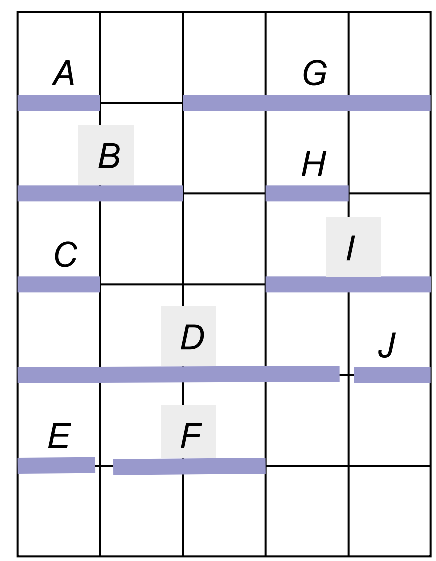


Left-Edge Algorithm – Example

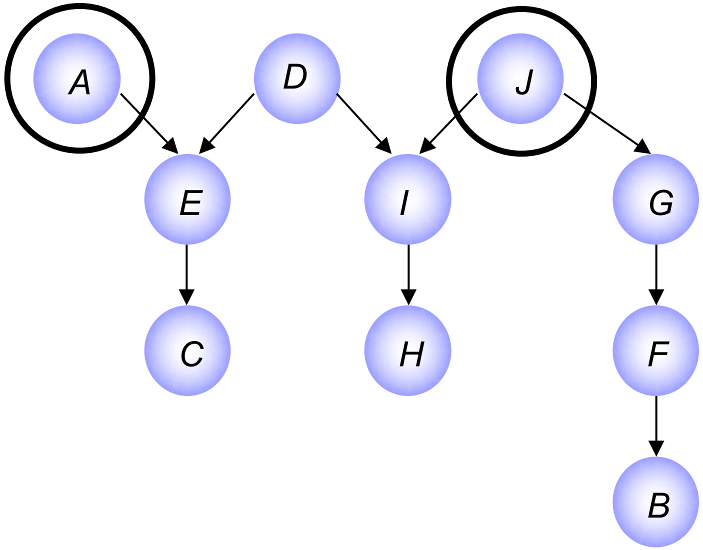


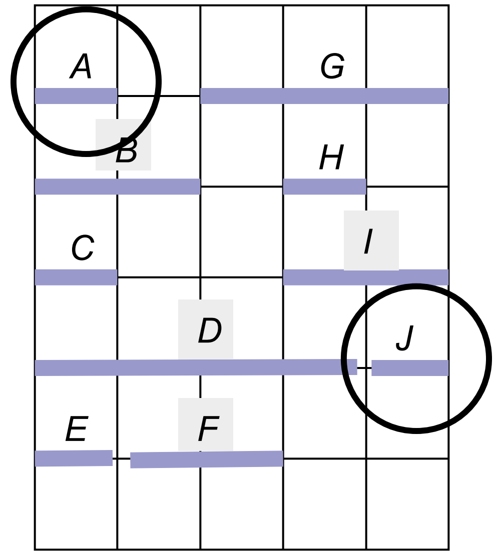
1. Generate VCG and zone representation





Left-Edge Algorithm – Example





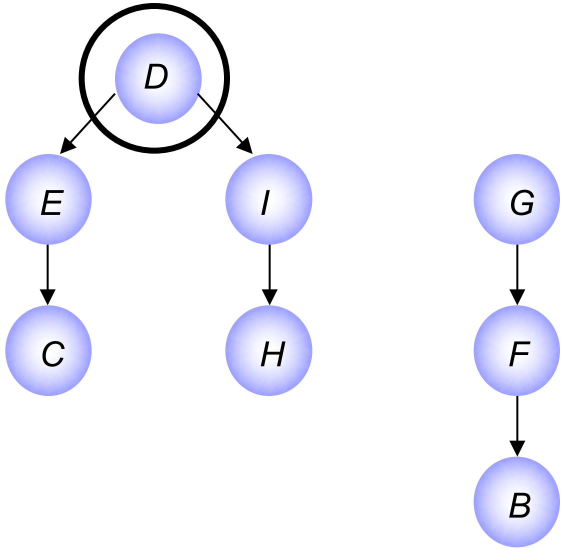
2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track* assign *curr_net*

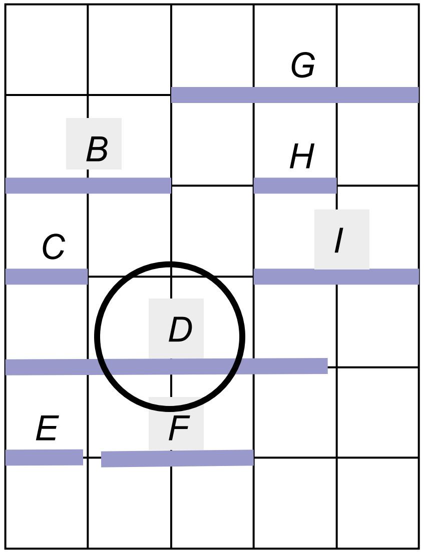
curr_track = 1: Net *A* Net *J*

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

Left-Edge Algorithm – Example

Left-Edge Algorithm – Example



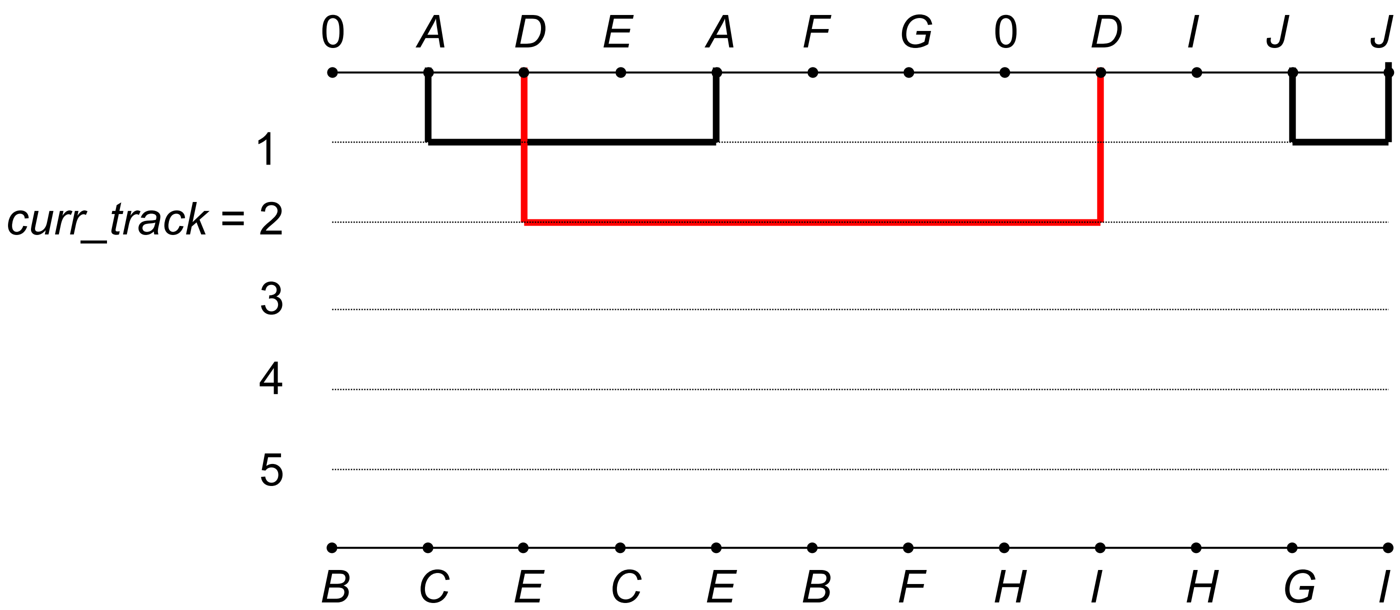


2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track* assign *curr_net*

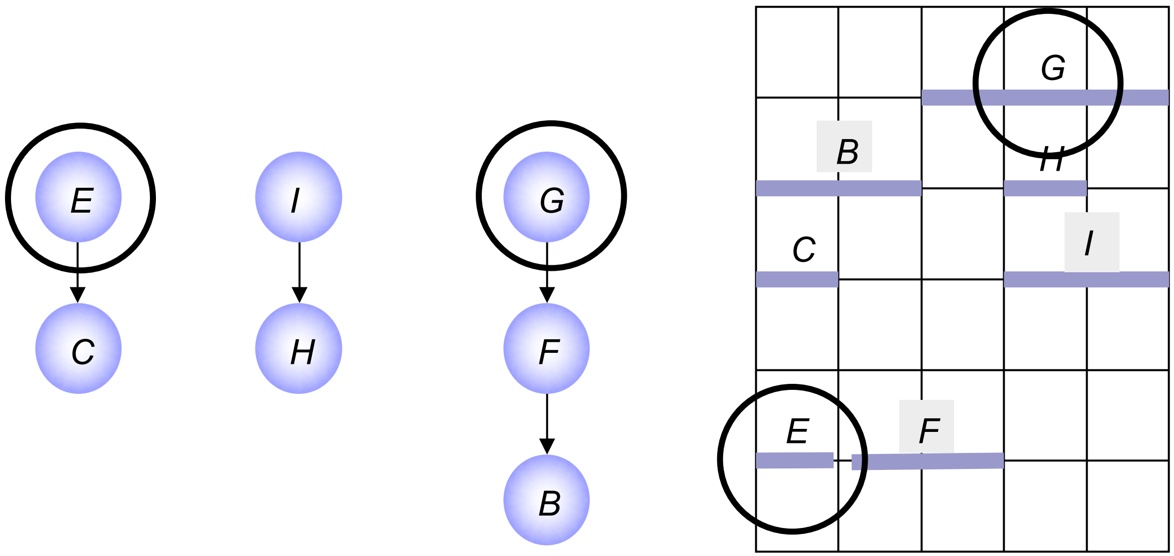
curr_track = 2: Net *D*

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

Left-Edge Algorithm – Example



Left-Edge Algorithm – Example

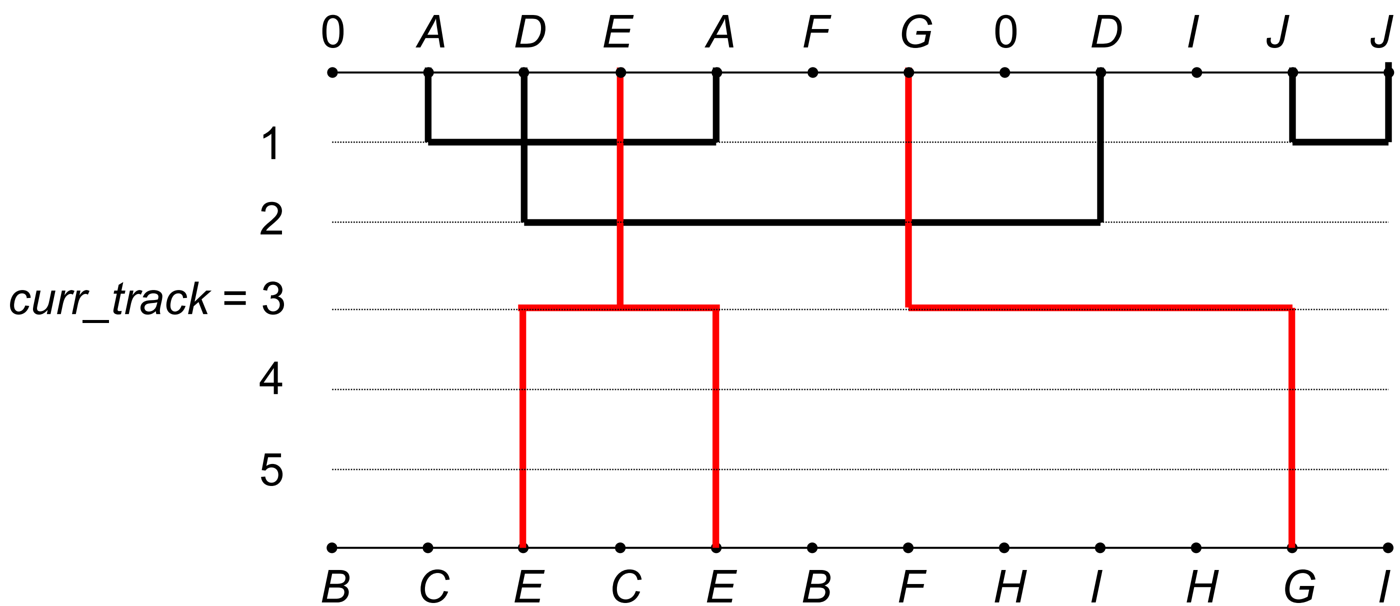


2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track*
   assign *curr_net*

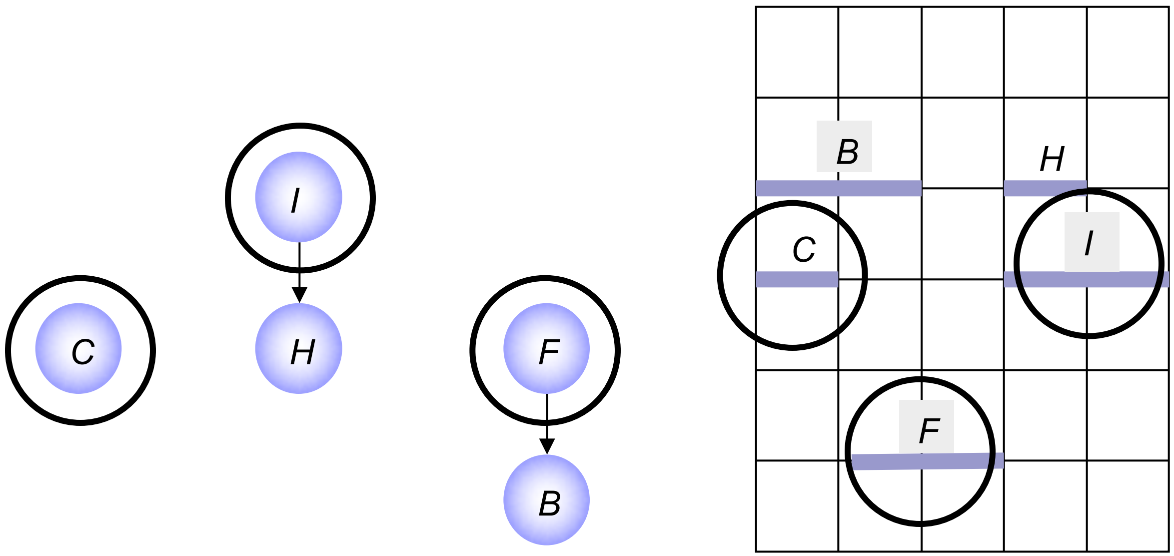
curr_track = 3: | Net *E* | Net *G*

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

6.3.1 Left-Edge Algorithm – Example



Left-Edge Algorithm – Example

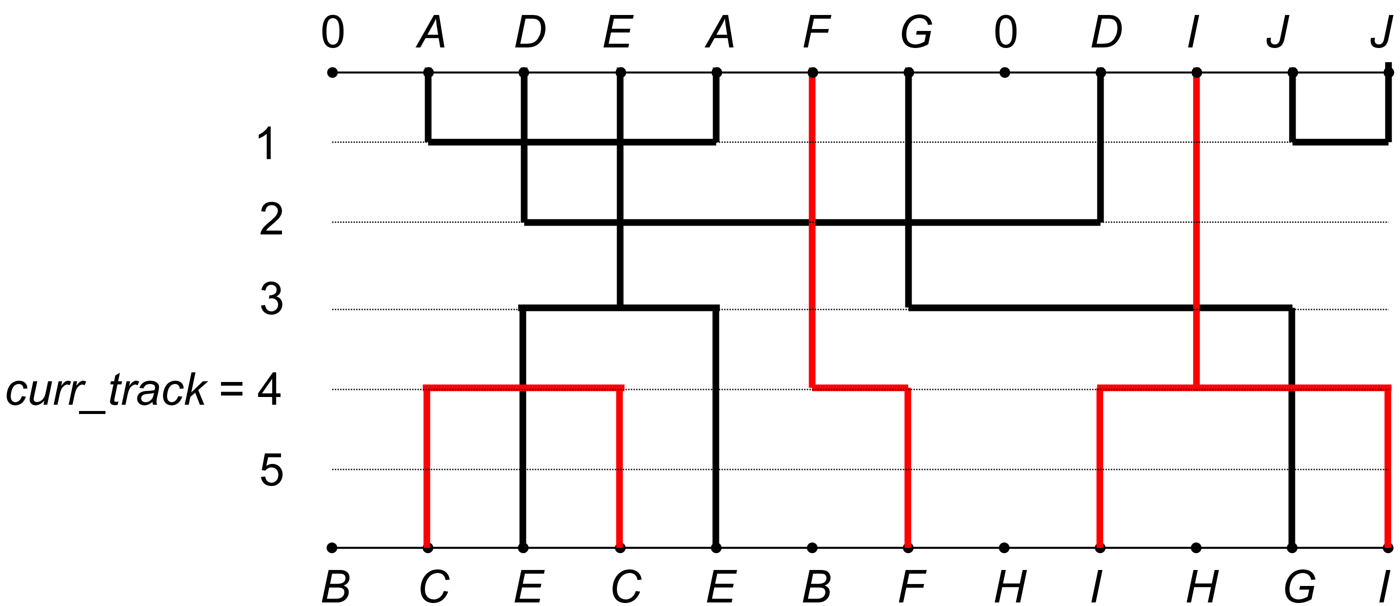


2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track*
   assign *curr_net*

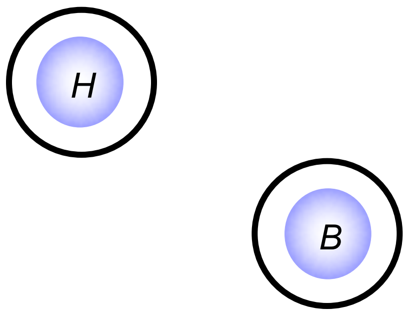
curr_track = 4:	Net *C*	Net *F*	Net *I*

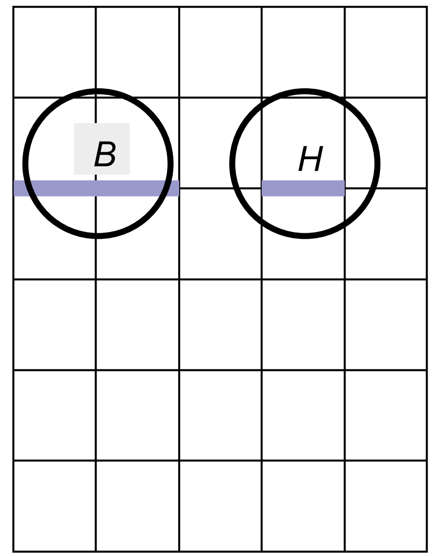
4. Delete placed nets (*C*, *F*, *I*) in VCG and zone representation

Left-Edge Algorithm – Example



Left-Edge Algorithm – Example



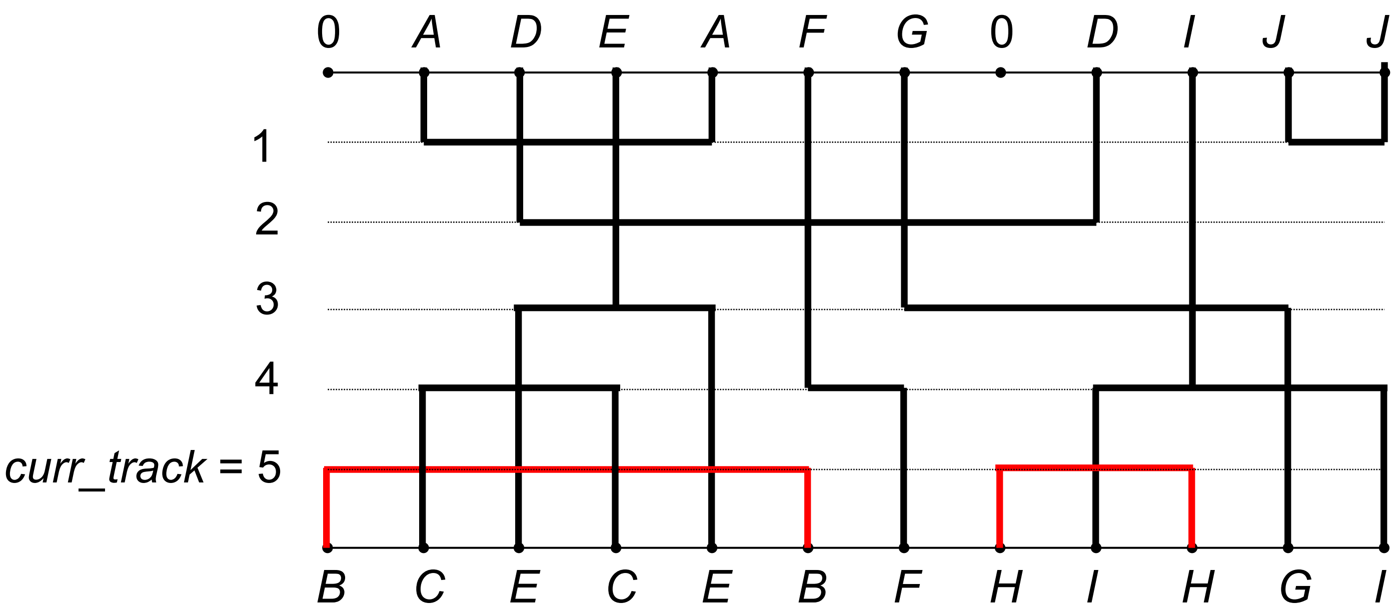


2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track* assign *curr_net*

curr_track = 5: Net *B* Net *H*

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

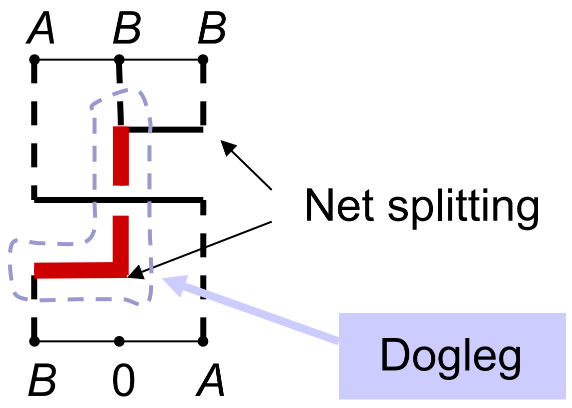
Left-Edge Algorithm – Example



Routing result

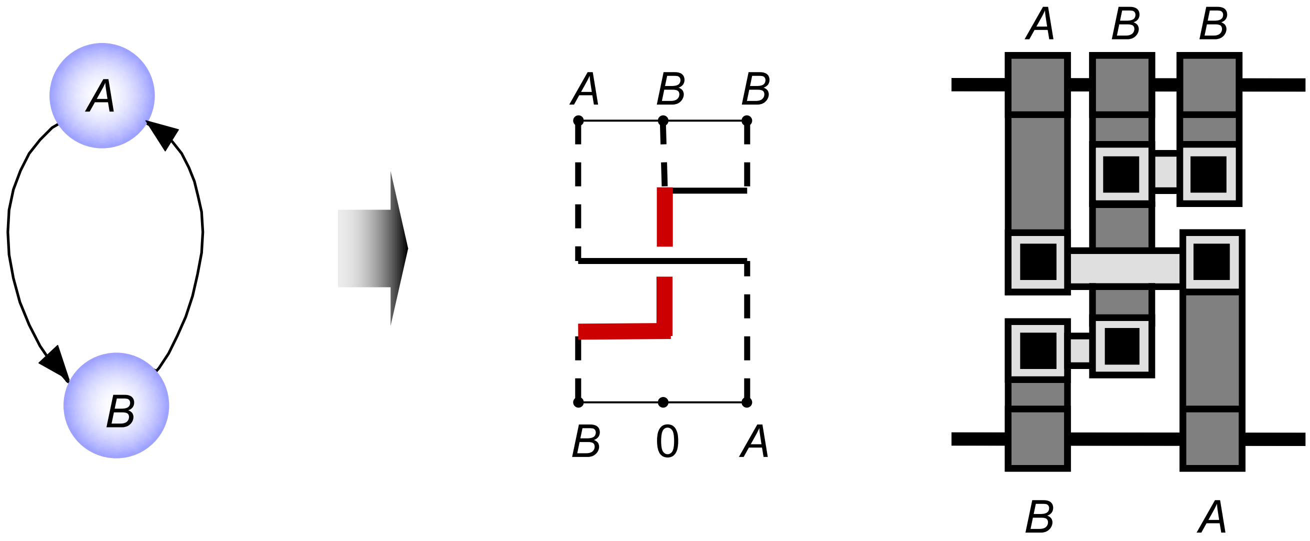
Dogleg Routing

- Improving left-edge algorithm by net splitting
- Two advantages:
  - Alleviates conflicts in VCG
  - Number of tracks can often be reduced



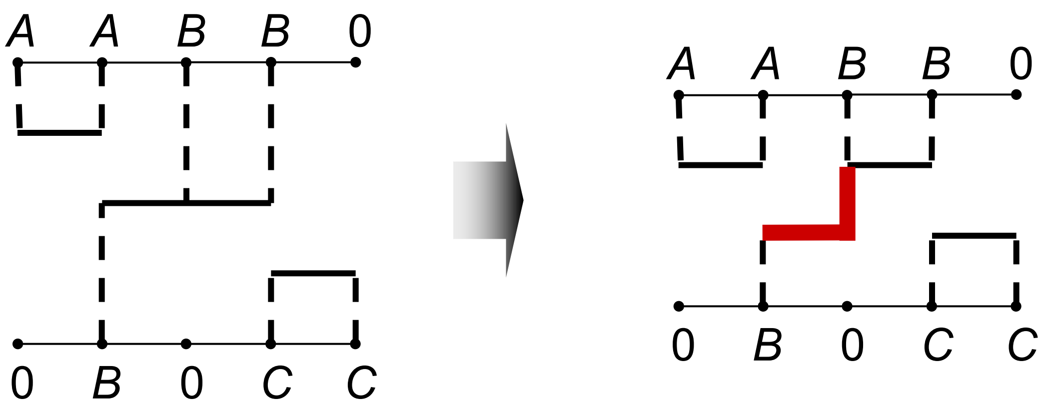
Dogleg Routing

Conflict alleviation using a dogleg



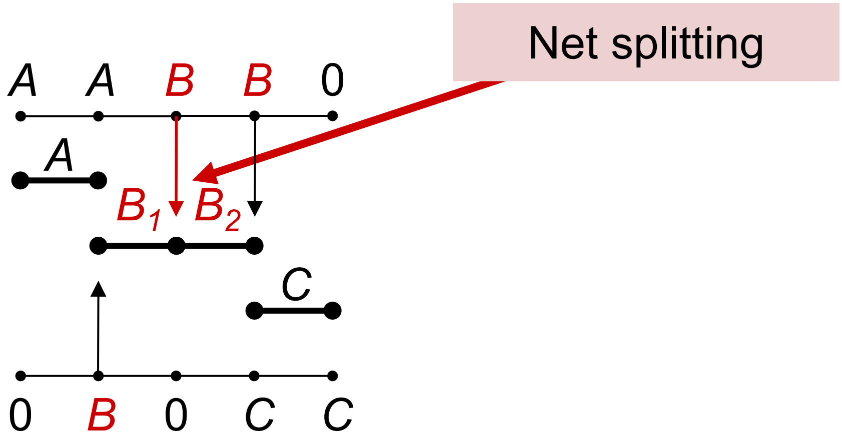
Dogleg Routing

Track reduction using a dogleg

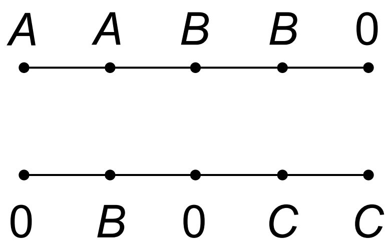


Dogleg Routing

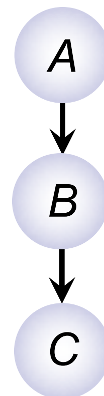
- Splitting *p*-pin nets (*p* > 2) into *p* -1 horizontal segments
- Net splitting occurs only in columns that contain a pin of the given net
- After net splitting, the algorithm follows the left-edge algorithm



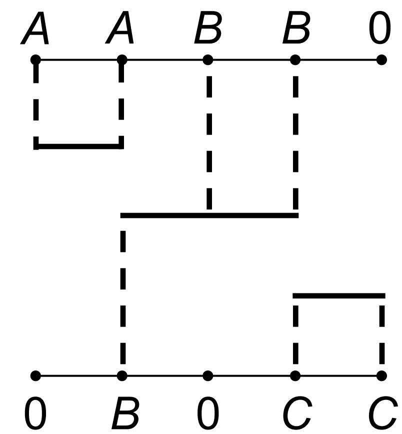
Dogleg Routing



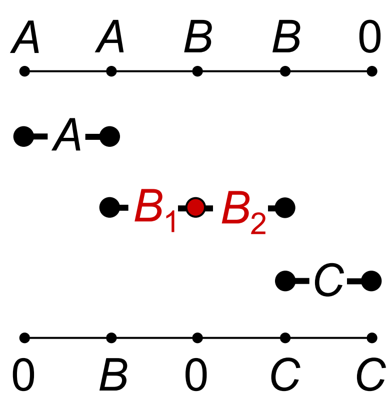
Channel routing problem



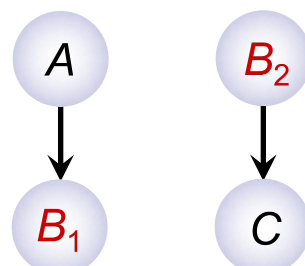
VCG without net splitting



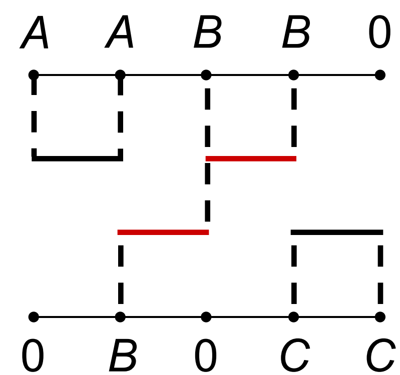
Channel routing solution



Net splitting



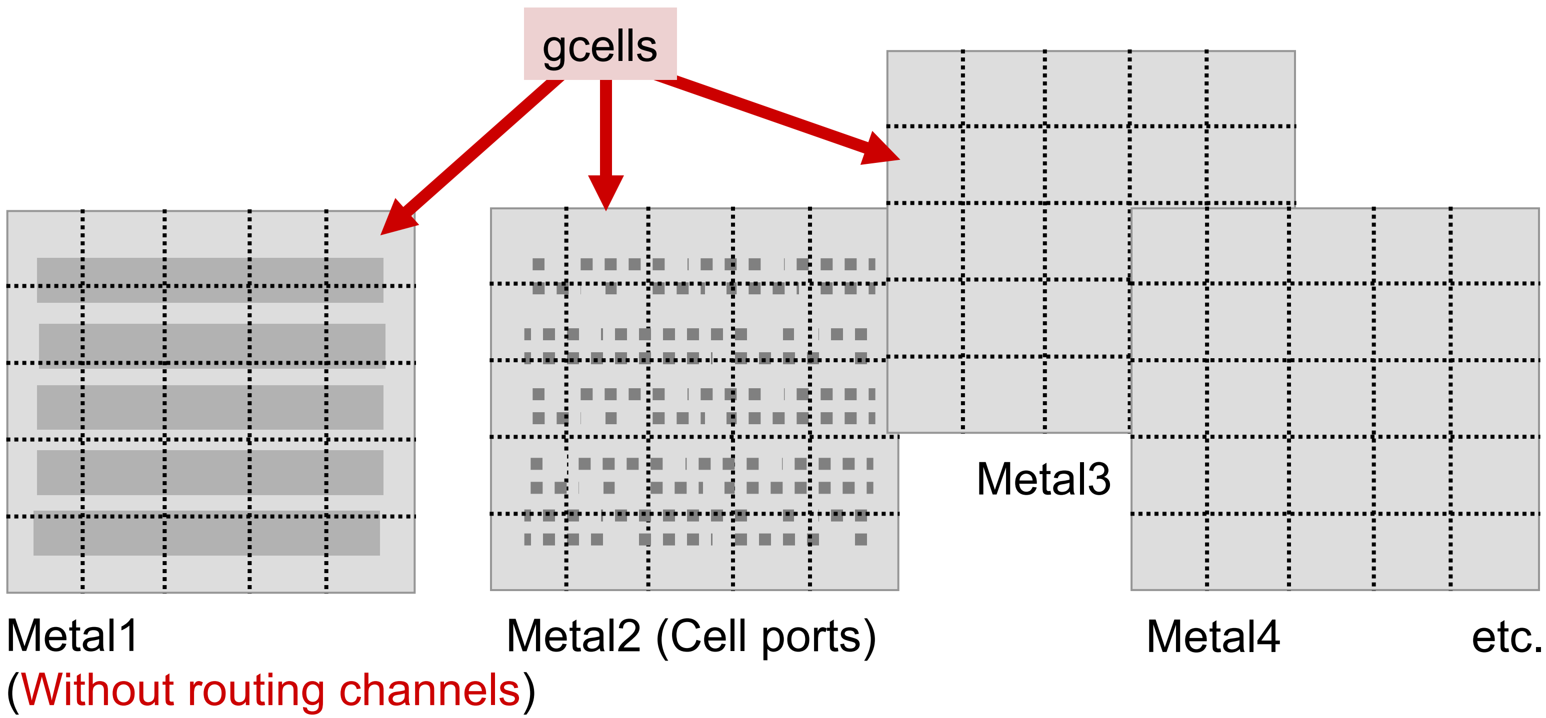
VCG with net splitting



Channel routing solution

Over-the-Cell Routing Algorithms

- Standard cells are placed back-to-back or without routing channels
- Metal layers are usually represented by a coarse routing grid made up of global routing cells (gcells)



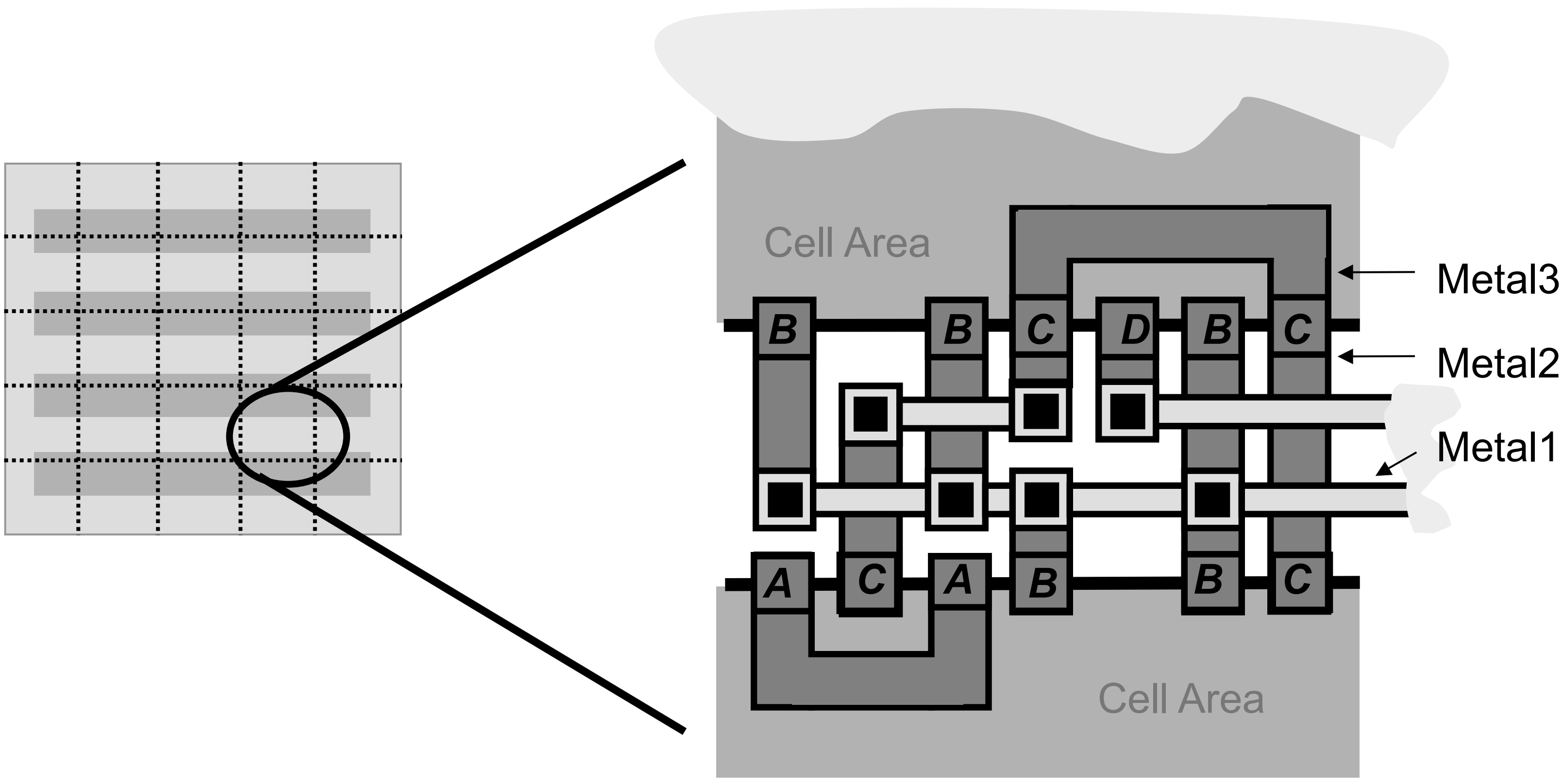
Over-the-Cell Routing Algorithms

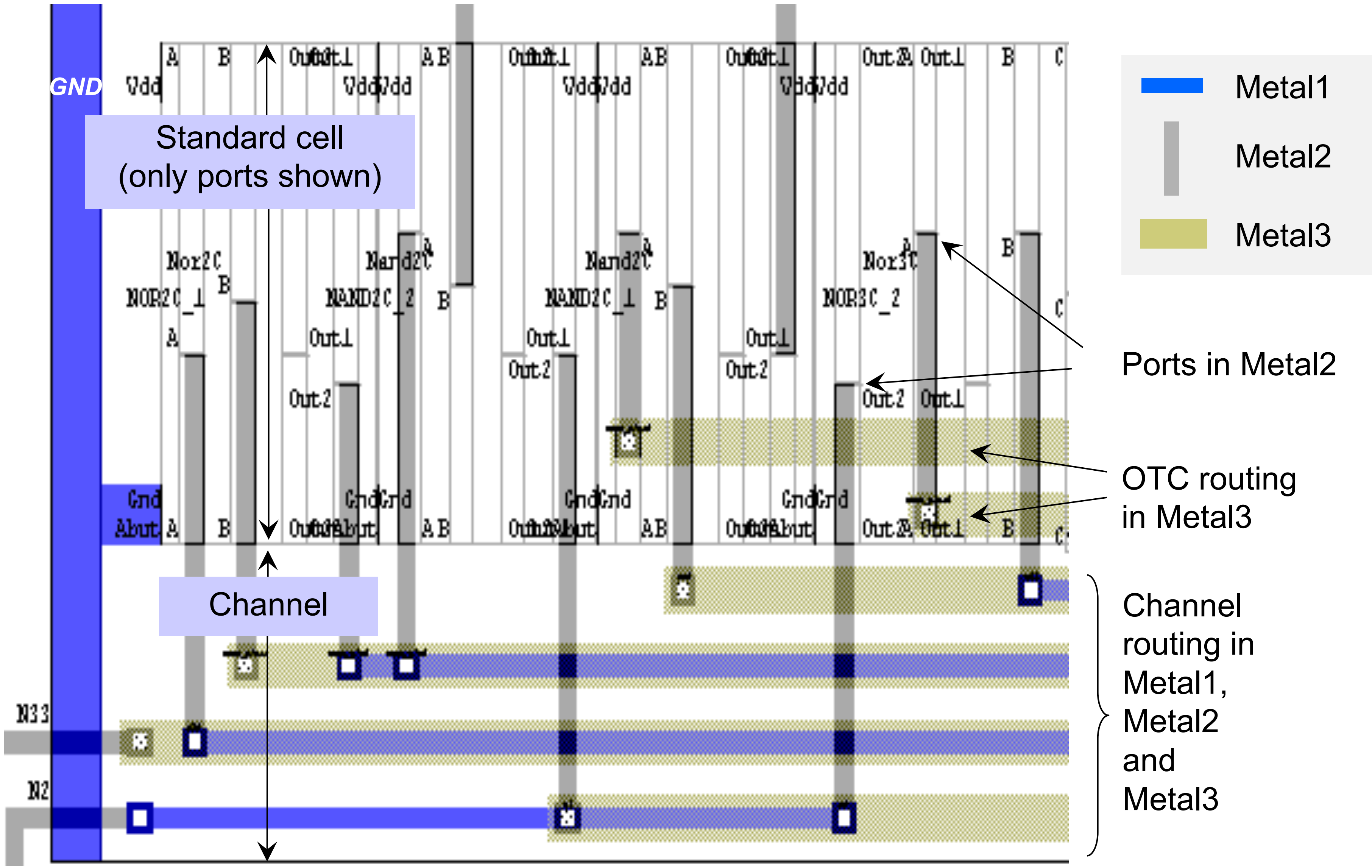
- Standard cells are placed back-to-back or without routing channels
- Metal layers are usually represented by a coarse routing grid made up of global routing cells (gcells)
- Layers that are not obstructed by standard cells are typically used for over-the-cell (OTC) routing
- Nets are globally routed using gcells and then detail-routed

Over-the-Cell Routing Algorithms

Three-layer approach

- Metal3 is used for over-the-cell (OTC) routing




GND
Standard cell
(only ports shown)
Channel
Metal1
Metal2
Metal3
Ports in Metal2
OTC routing
in Metal3
Channel
routing in
Metal1,
Metal2
and
Metal3
NOR2C_1
NAND2C_2
NAND2C_1
NOR3C_2
N33
N2

Modern Challenges in Detailed Routing

- Manufacturers today use different configurations of metal layers and widths to accommodate high-performance designs
- Detailed routing is becoming more challenging, for example:
  - Vias connecting wires of different widths inevitably block additional routing resources on the layer with the smaller wire pitch
  - Advanced lithography techniques used in manufacturing require stricter enforcement of preferred routing direction on each layer

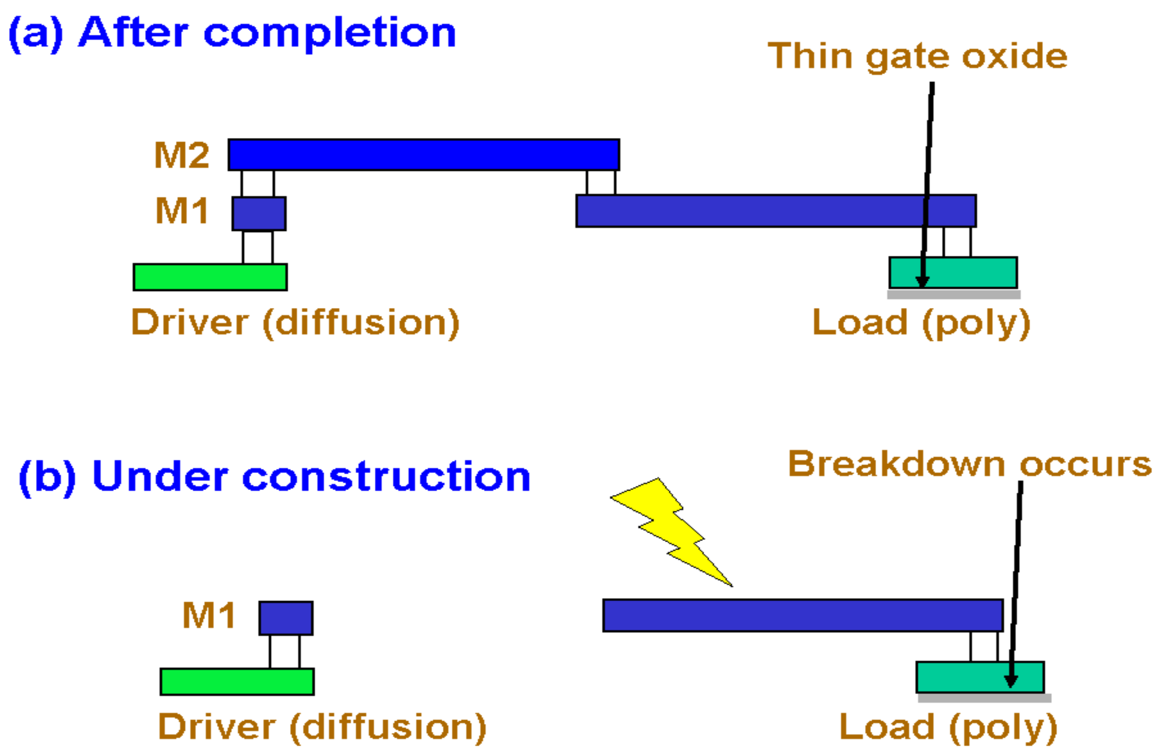
Modern Challenges in Detailed Routing

- Semiconductor manufacturing yield is a key concern in detailed routing
  - Redundant vias and wiring segments as backups (via doubling and non-tree routing)
  - Manufacturability constraints (design rules) become more restrictive
  - Forbidden pitch rules prohibit routing wires at certain distances apart, but allows smaller or greater spacings

- Detailed routers must account for manufacturing rules and the impact of manufacturing faults
  - Via defects: via doubling during or after detailed routing
  - Interconnect defects: add redundant wires to already routed nets
  - Antenna-induced defects: detailed routers limit the ratio of metal to gate area on each metal layer

Modern Challenges in Detailed Routing

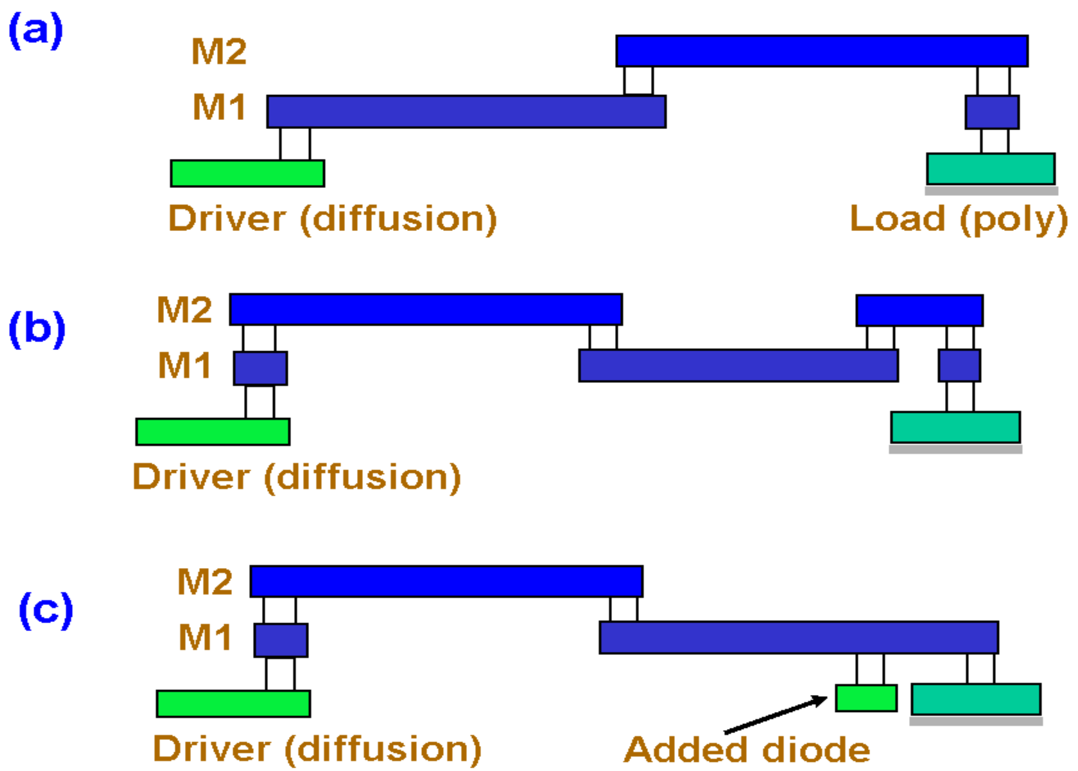
Antenna Effect



Source: http://en.wikipedia.org/wiki/Antenna_effect

Modern Challenges in Detailed Routing

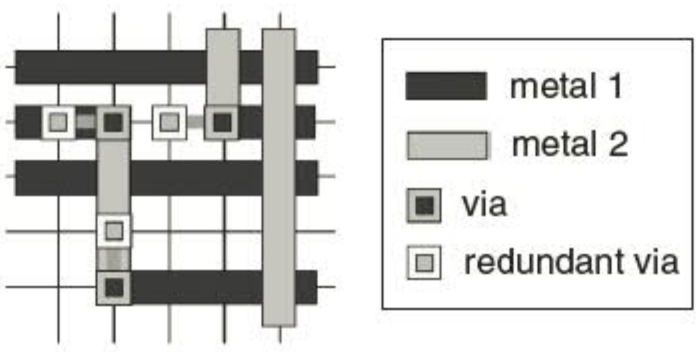
Antenna Effect Fix



Source: `http://en.wikipedia.org/wiki/Antenna_effect`

Modern challenges in Detail routing

- Redundant Via
  - Via open defect is one of the major cause of failure.
  - Can happen due to random defect, cut misalignment, electro-migration etc.
  - This significantly reduces yield and in some cases performance



Summary

- Detailed routing is invoked after global routing
- Usually takes about as much time as global routing
  - For heavily congested designs can take much longer
- Generates specific track assignments for each connection
  - Tries to follow "suggestions" made by global routing, but may alter them if necessary
  - A small number of failed global routed (disconnected, overcapacity) can be tolerated
- More affected by technology & manufacturing constraints than global routing
  - Must satisfy design rules

Summary – Modern Challenges

- Variable-pitch wire stacks
  - Not addressed in the literature until 2008
- Satisfying more complex design rules
  - Min spacing between wires and devices
  - Forbidden pitch rules
  - Antenna rules
- Soft rules
  - Do not need to be satisfied
  - Can improve yield by decreasing the probability of defects
- Redundant vias
  - In case some vias are poorly manufactured
- Redundant wires
  - In case some wires get disconnected