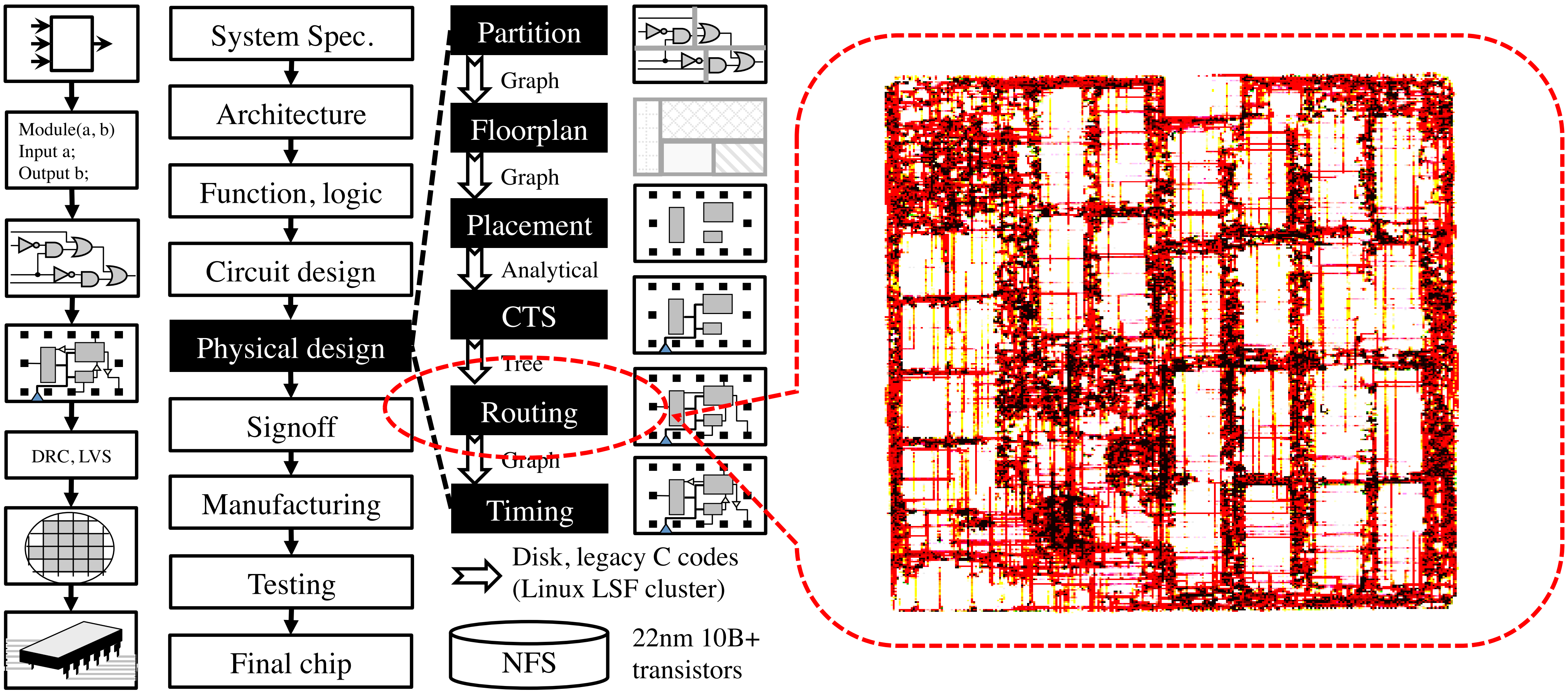
Lecture 17: Routing – II

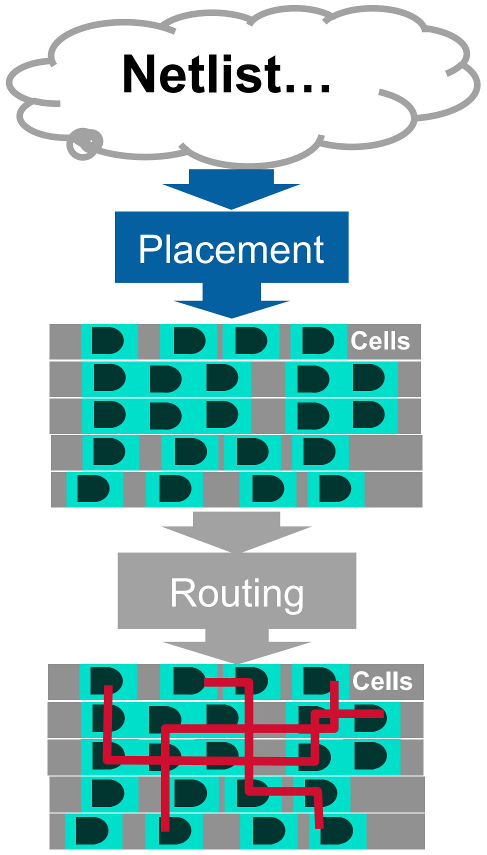
Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT

Recap: Routing

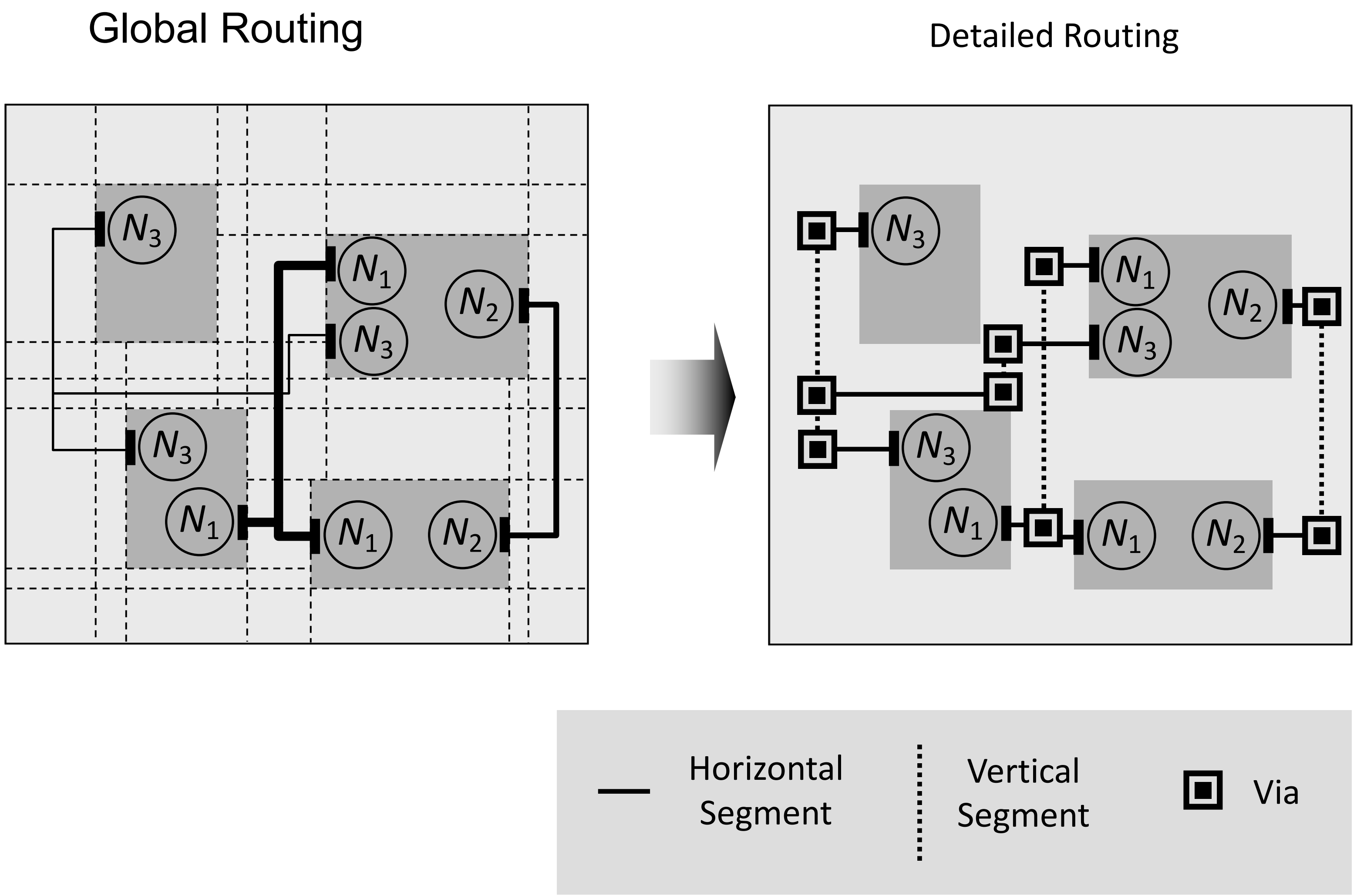


Recap: Challenges of Routing

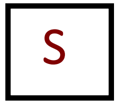
- **Scale**
  - Big chips have an enormous number (**millions**) of wires
  - Not every wire gets to take an “easy” path to connect its pins
  - **Must** connect them all--can’t afford to tweak many wires manually
- **Geometric complexity**
  - It used to be representing the layout was a simple “grid”
  - No longer true: at nanoscale, **geometry rules are complex** – makes routing hard
- **Electrical complexity**
  - It’s not enough to make sure you connect all the wires
  - Must ensure **delays** thru the wires are not too big
  - And wire-to-wire **interactions** (crosstalk) don’t mess up



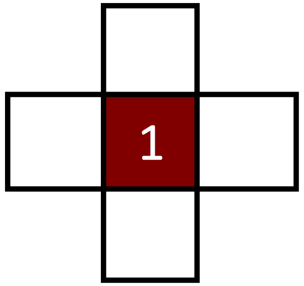
Recap: Global and Detailed Routing



Recap: Maze Router

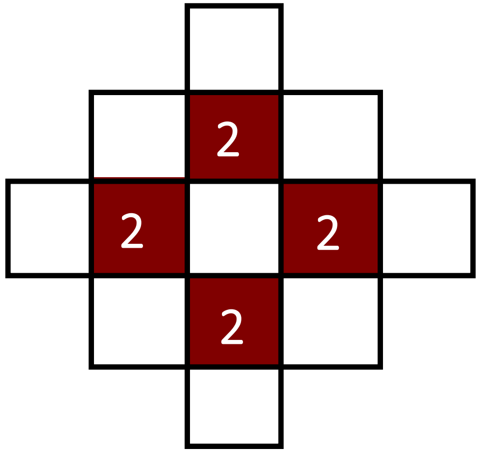


Start at the **source**



Find all new cells that are reachable at **pathlength 1**, ie, all paths that are just 1 unit in total length (just 1 cell) - mark all with this the pathlength

Repeat the expansion until the target is found!



Using the **pathlength 1** cells, find all new cells which are reachable at **pathlength 2**

Recap: Maze Router Walkthrough – I

3	2	3	4	5	6
2	S 1	2	3	4	5
3	2	3	4	5	6
4	3	4	5	6	7
5	4	5	6	T 7	
6	5	6	7		

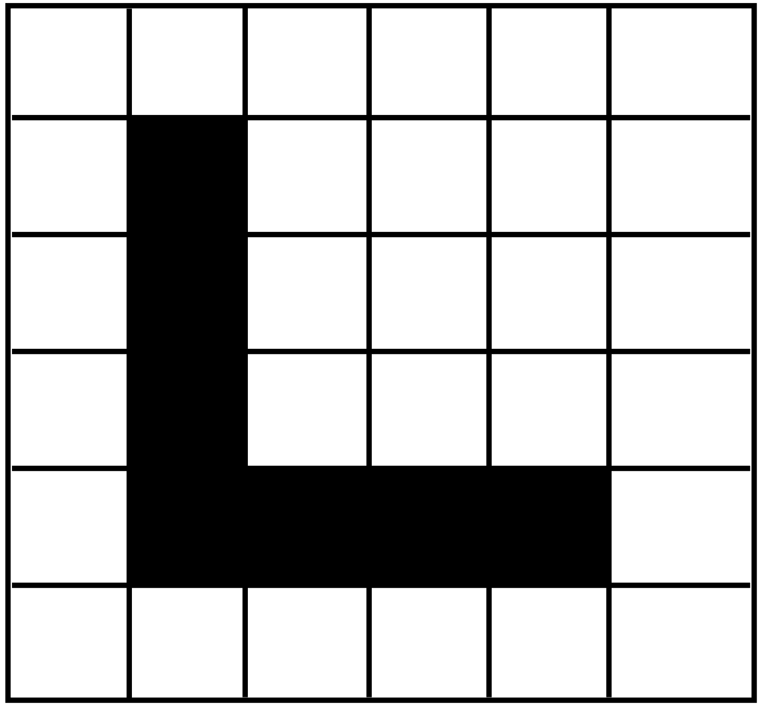
- **Strategy**
  - Expand **one cell at a time** until all the shortest paths from **S** to **T** are found.
  - Expansion creates a **wavefront** of paths that search broadly out from source cell until target is hit
  - Remember this!? We have done this using breadth-first-search (BFS) algorithm!

Recap: Maze Router Walkthrough – II

3	2	3	4	5	6
2	S 1	2	3	4	5
3	2	3	4	5	6
4	3	4	5	6	7
5	4	5	6	T 7	
6	5	6	7		

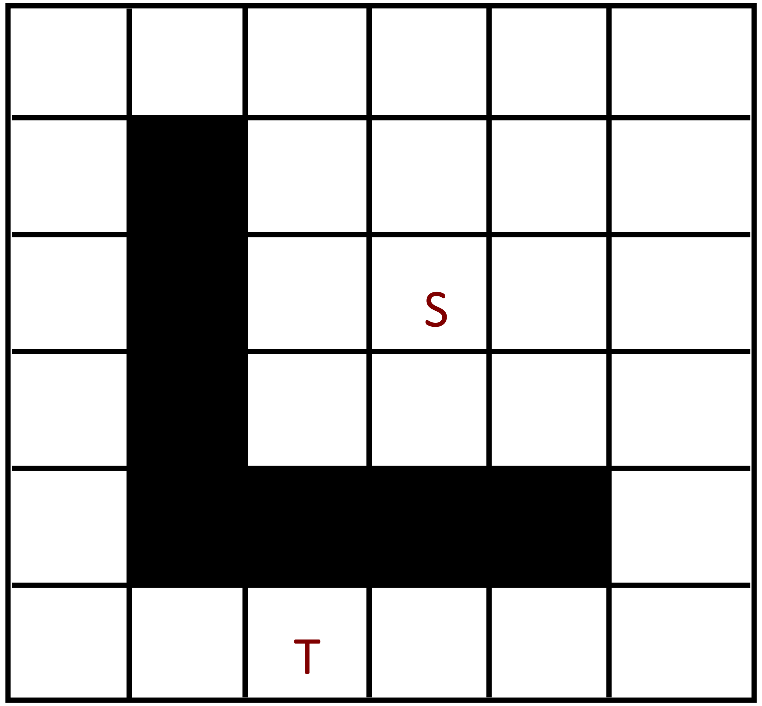
- **Now what? Backtrace**
  - Select a shortest-path (**any** shortest-path) from target back to source
  - Mark its cells so they cannot be used again – mark them as **obstacles** for later wires we want to route
  - Since there are many paths back, optimization information can be used to select the best one
  - Here, just follow the pathlengths in the cells **in descending order**

Recap: Maze Router Walkthrough – III



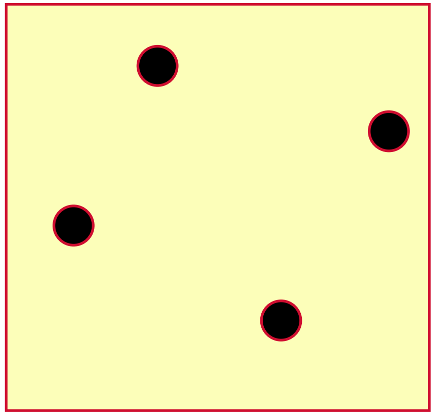
- **Now what?** **Clean-up**
  - Clean up the grid for the next net, leaving the **S** to **T** path as an **obstacle**
  - Now, ready to route the next net with the obstacles from the previously routed net in place in the grid

Recap: Maze Router Walkthrough – IV

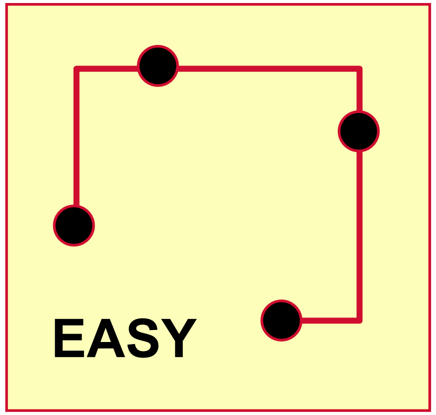


- **Also called "Blockages"**
  - Any cell you cannot use for a wire is a an **obstacle** or a **blockage**
  - There may be parts of the routing surface you just cannot use
  - But most importantly, you **label each newly routed net as a blockage**
  - Thus, all future nets must **route around** this blockage

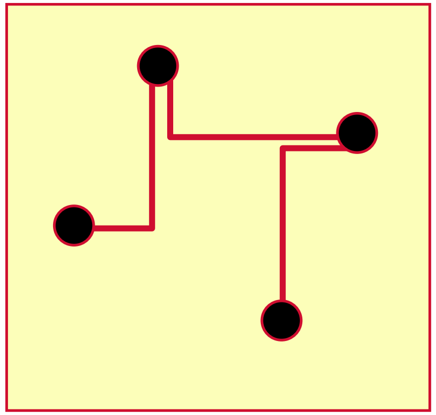
Recap: Steiner Tree Constructions



Pins to connect

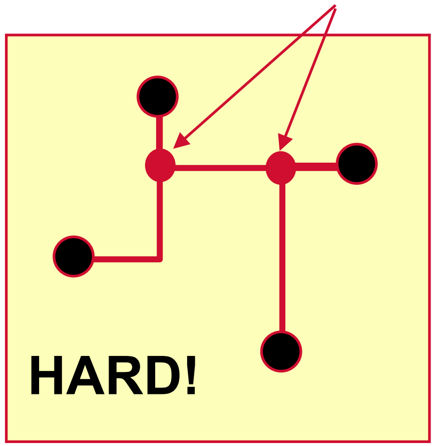


Route it so we guarantee each 2-point path is shortest; this is **Minimum Spanning Tree**



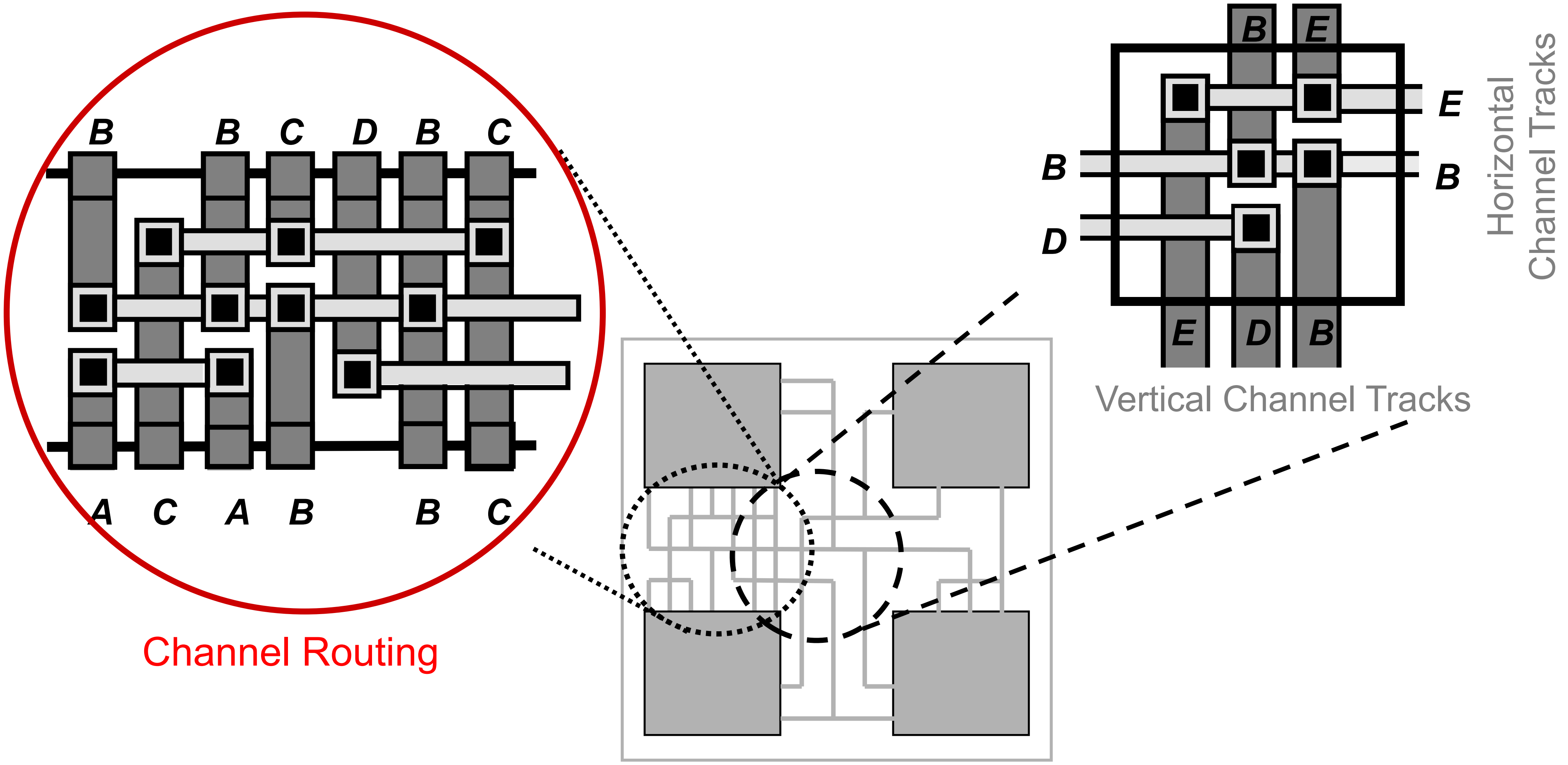
Redraw it--different orientations of 2-point paths

2 so-called "Steiner-points"

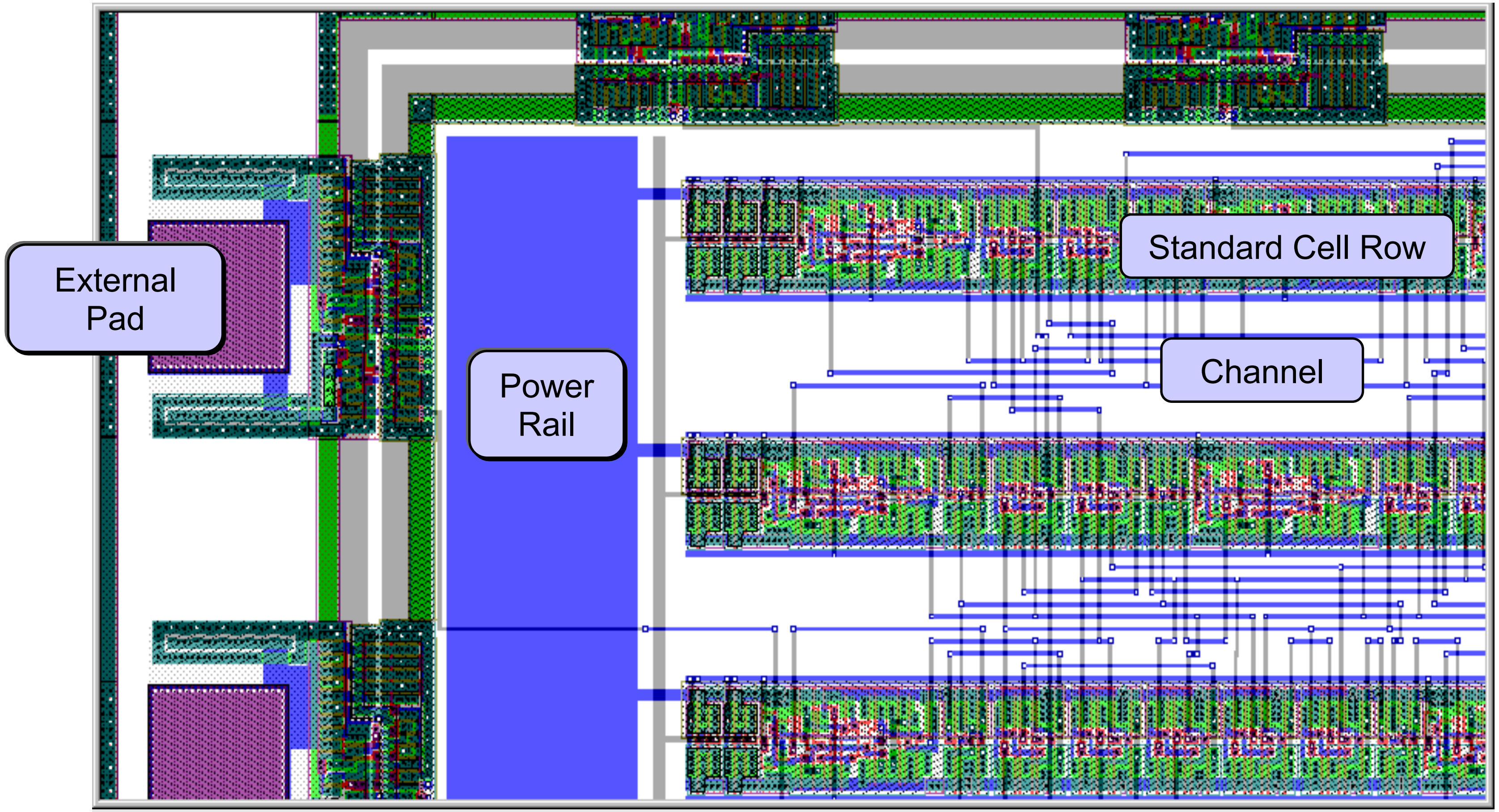


Now we can see the better (shorter) Steiner tree

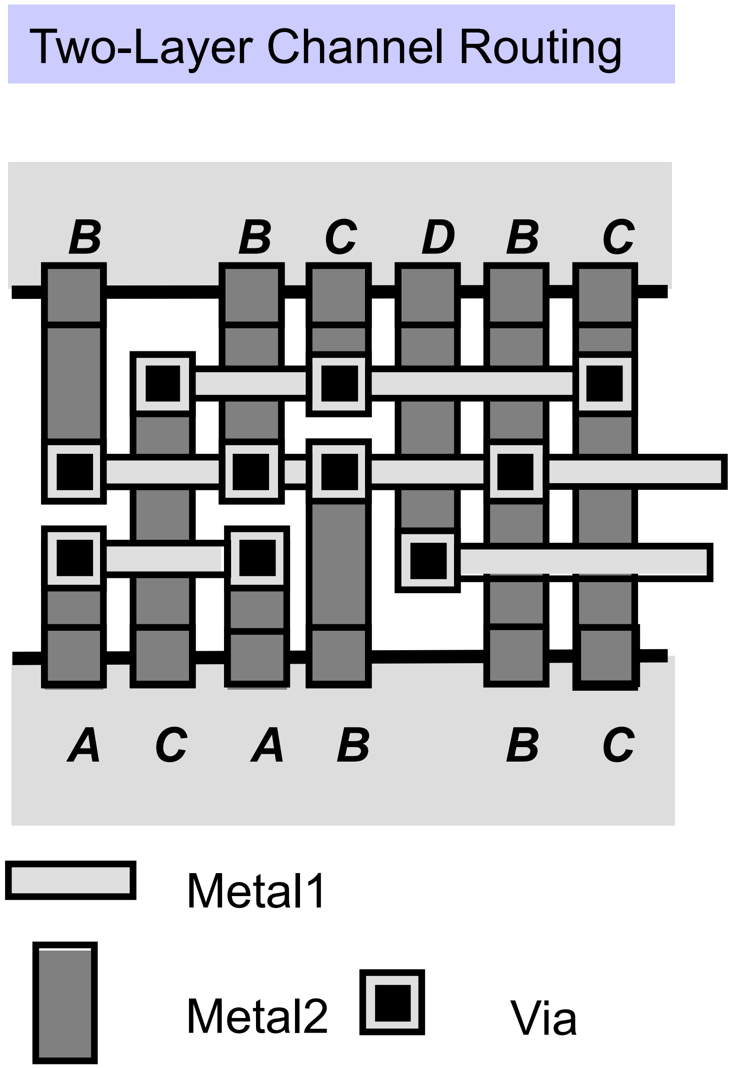
Channel and Switchbox Routing

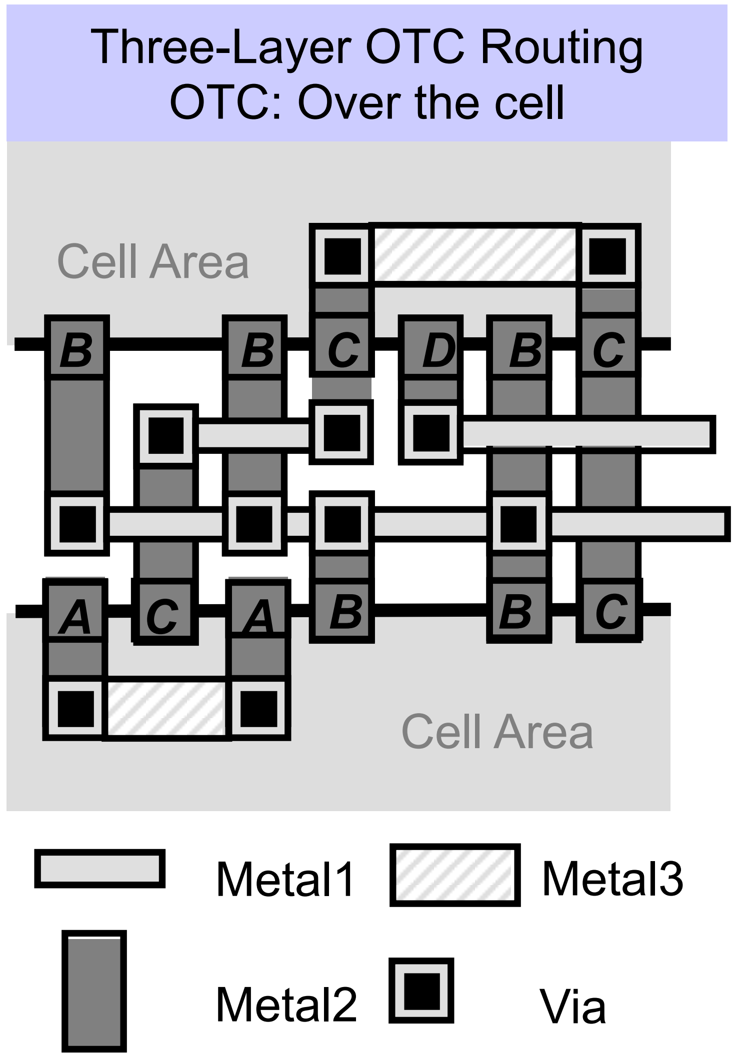


Channel Routing Terminology – I

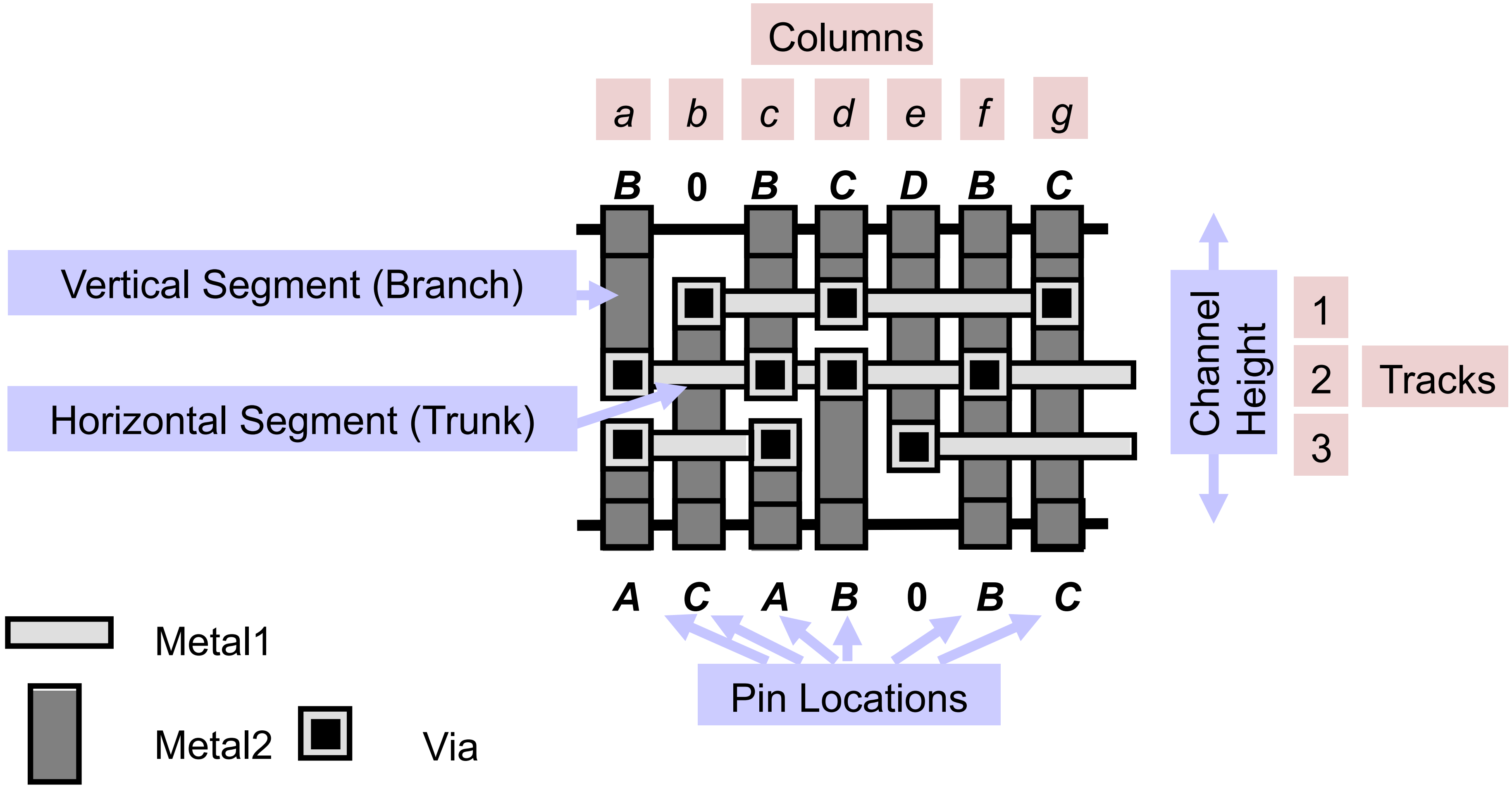


Channel Routing Terminology – II



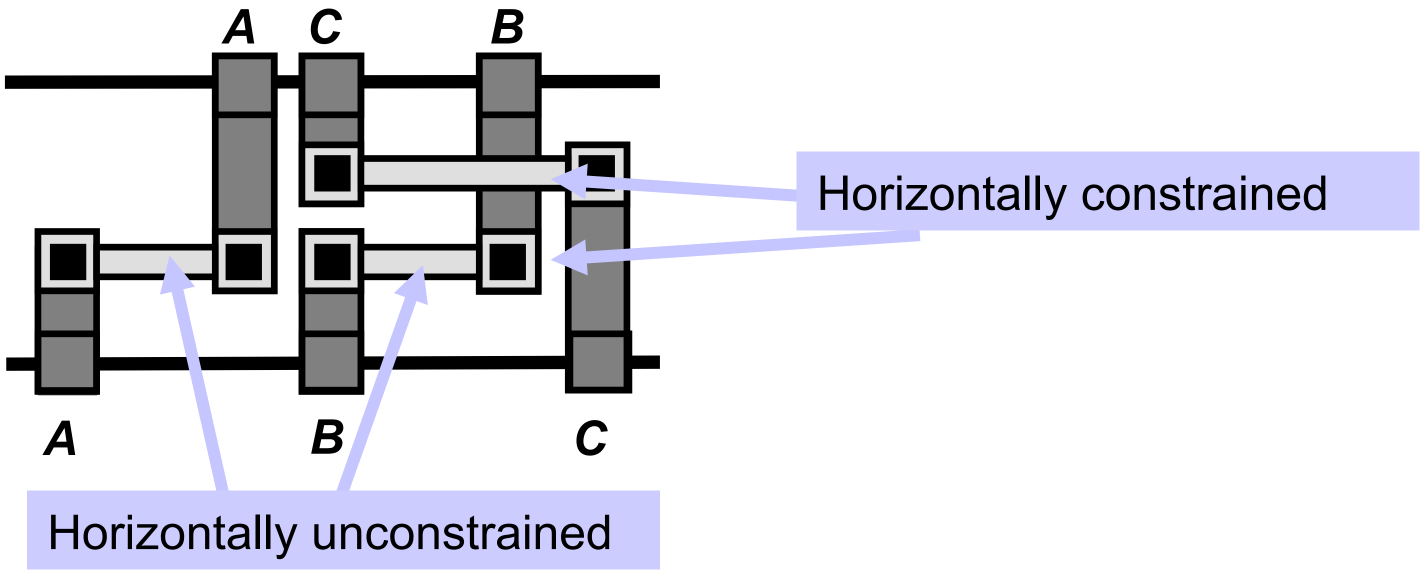


Channel Routing Terminology – III



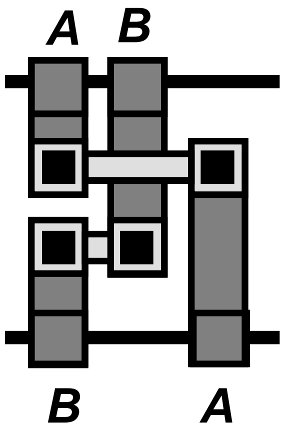
Channel Routing Terminology – IV

- A horizontal constraint exists between two nets if their horizontal segments overlap

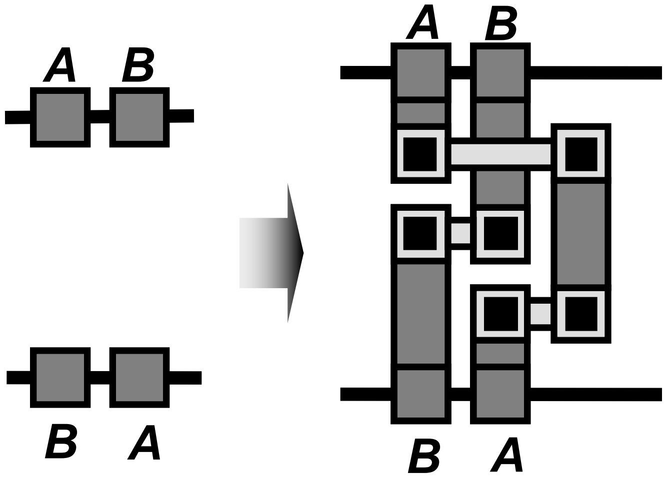


Channel Routing Terminology – V

- 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 (route back)



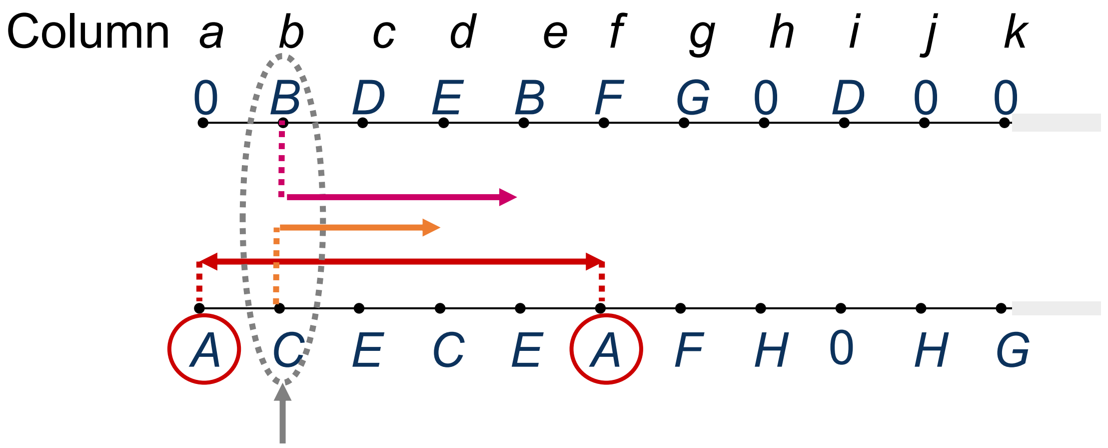
Vertically constrained with a vertical conflict (route back)

Horizontal and Vertical Constraint Graph

- The relative positions of nets in a channel routing instance can be modeled by horizontal and vertical constraint graphs

- These graphs are used to
  1. initially predict the minimum number of tracks that are required
  2. detect potential routing conflicts

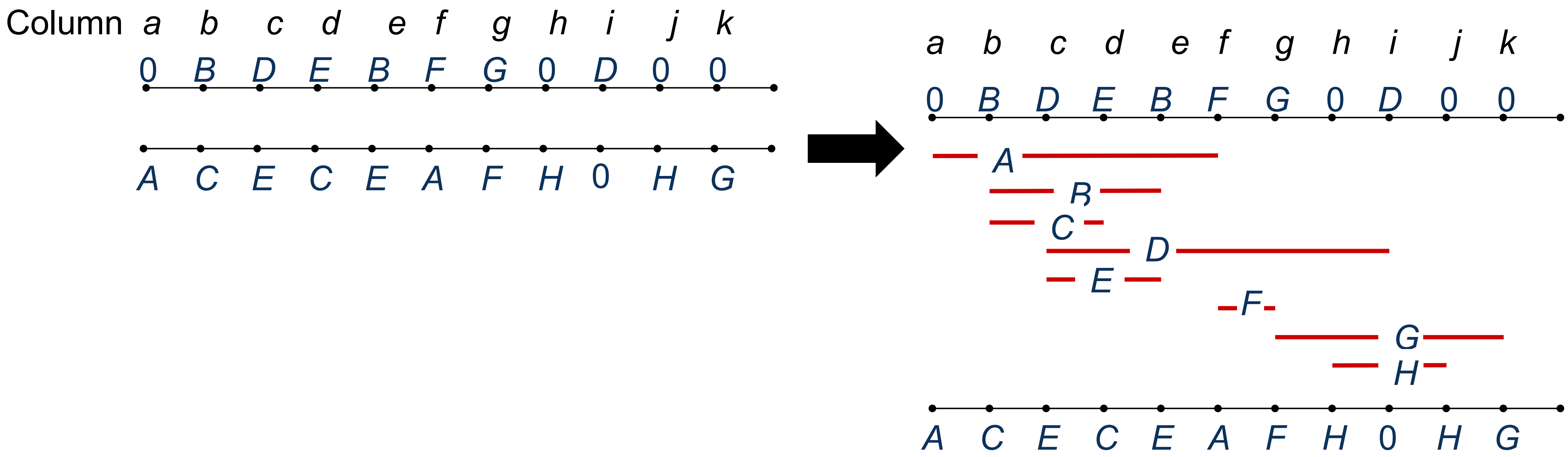
Horizontal and Vertical Constraint Graph



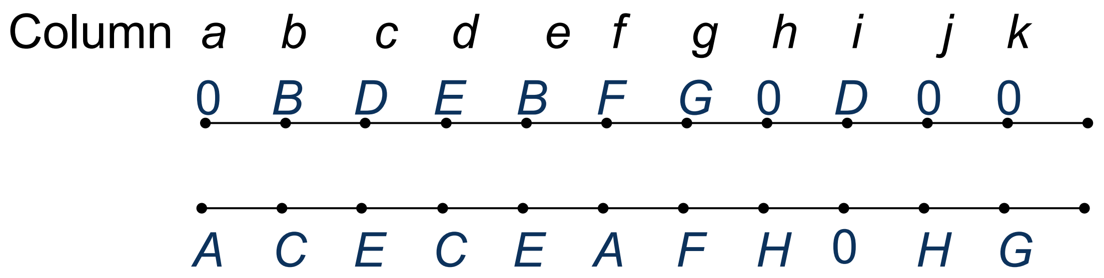
$S(b) = \{A, B, C\}$ – **lower bound of # tracks = 3**

- 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 colum *col*; lower bound of the channel height is given by maximum cardinality of any $S(col)$

Horizontal and Vertical Constraint Graph



Horizontal and Vertical Constraint Graph



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

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

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

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

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

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

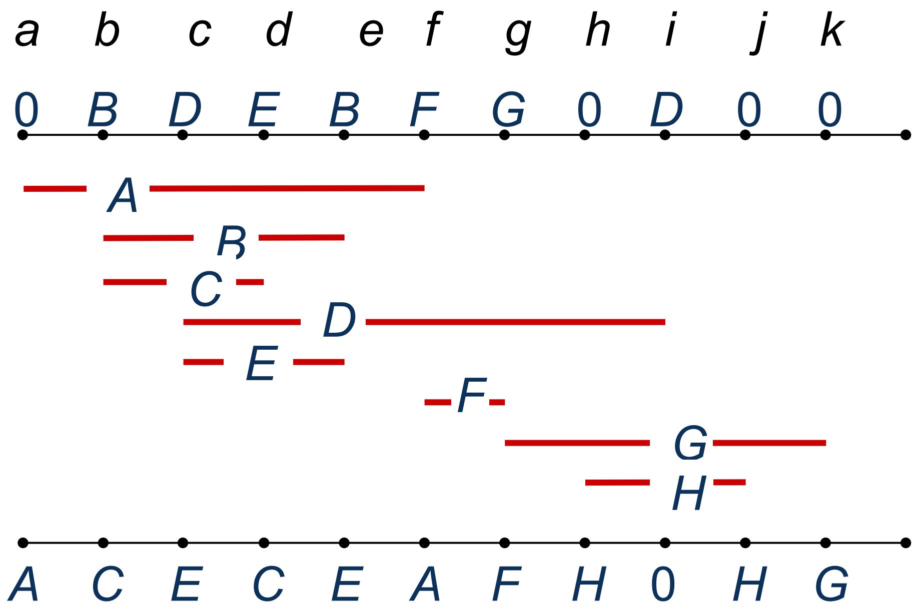
$S(g) = \{D,F,G\}$

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

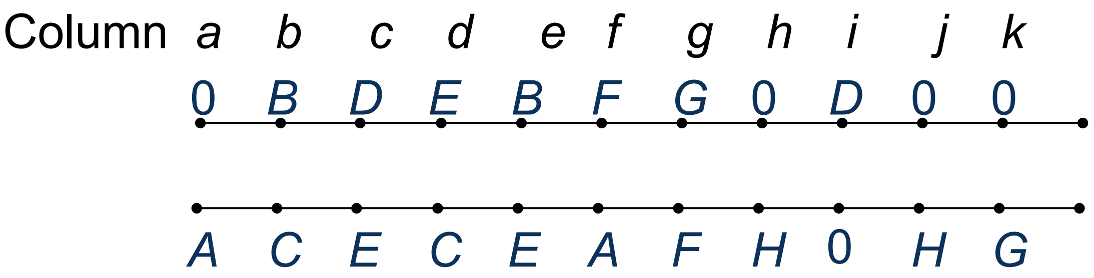
$S(i) = \{D,G,H\}$

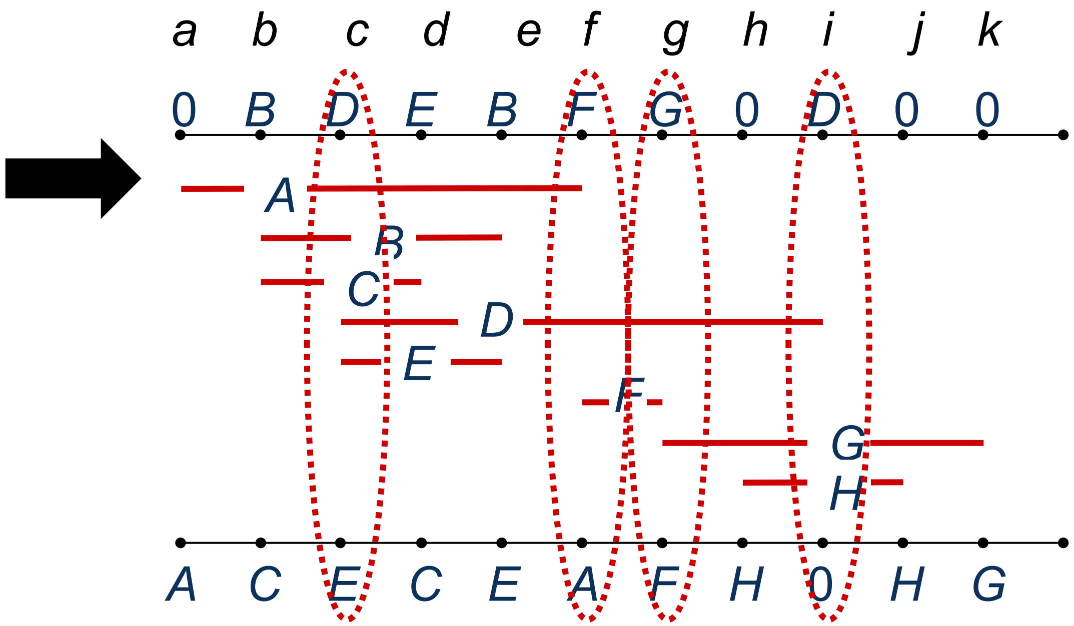
$S(j) = \{G,H\}$

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



Horizontal and Vertical Constraint Graph





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

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

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

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

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

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

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

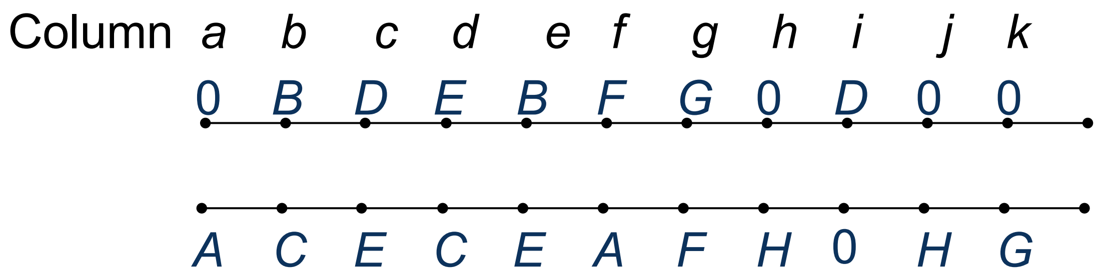
$S(h) = \{D,G,H\}$

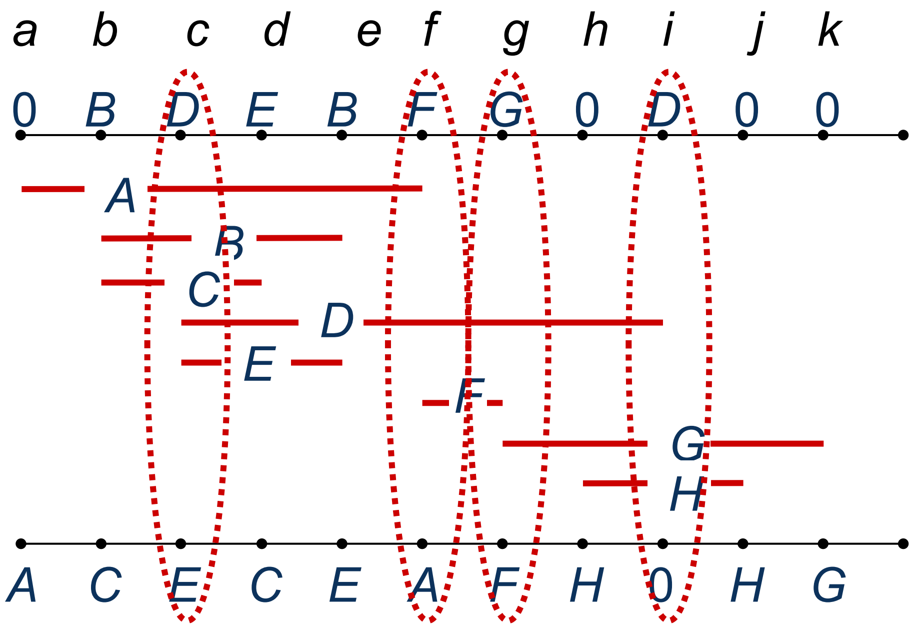
$S(i) = \{D,G,H\}$

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

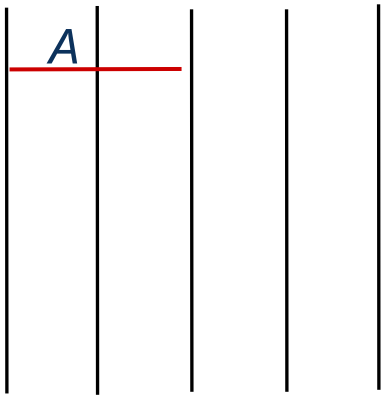
$S(k) = \{G\}$

Zone Representation: {A}





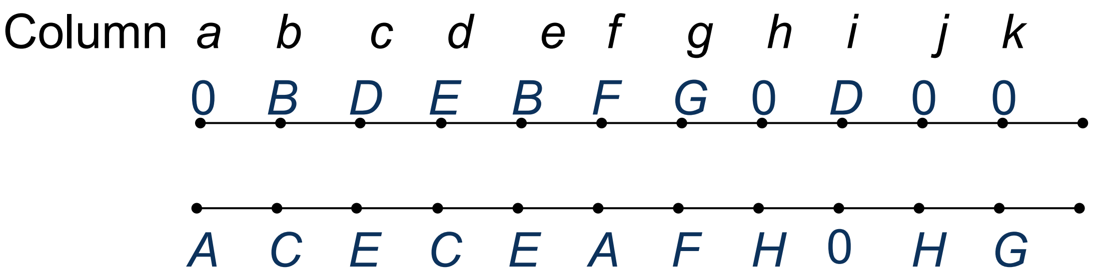
$S(a) = \{A\}$
$S(b) = \{A,B,C\}$
$S(c) = \{A,B,C,D,E\}$
$S(d) = \{A,B,C,D,E\}$
$S(e) = \{A,B,D,E\}$
$S(f) = \{A,D,F\}$
$S(g) = \{D,F,G\}$
$S(h) = \{D,G,H\}$
$S(i) = \{D,G,H\}$
$S(j) = \{G,H\}$
$S(k) = \{G\}$



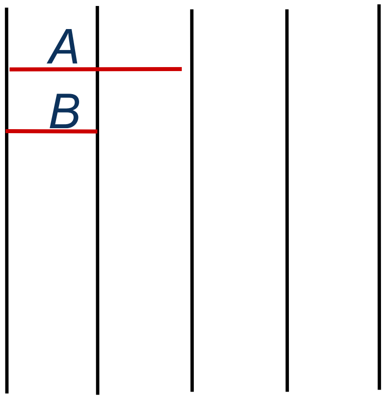
Zone representation

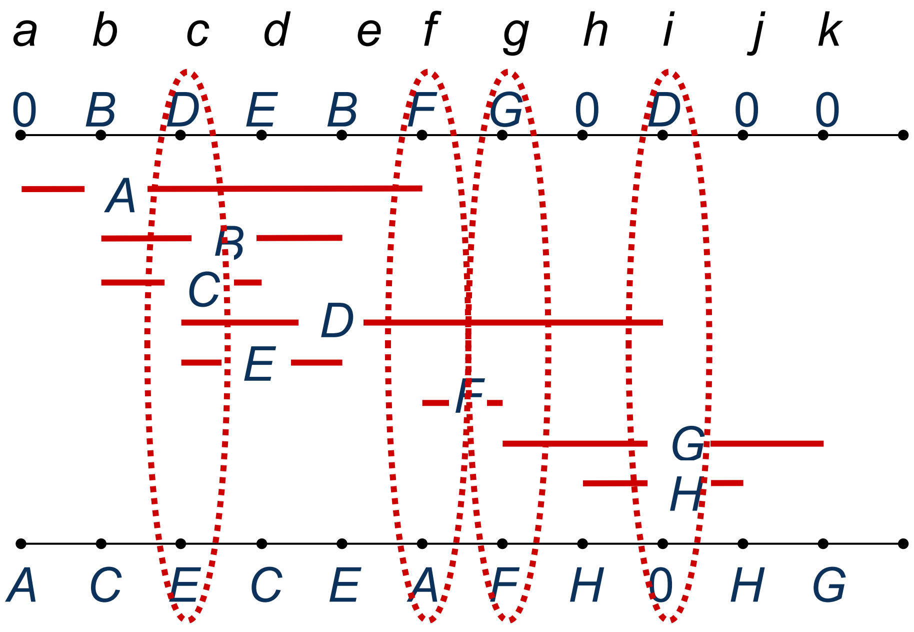
Lower bound on the number of tracks = 5

Zone Representation: {A, B}



$S(a) = \{A\}$
$S(b) = \{A,B,C\}$
$S(c) = \{A,B,C,D,E\}$
$S(d) = \{A,B,C,D,E\}$
$S(e) = \{A,B,D,E\}$
$S(f) = \{A,D,F\}$
$S(g) = \{D,F,G\}$
$S(h) = \{D,G,H\}$
$S(i) = \{D,G,H\}$
$S(j) = \{G,H\}$
$S(k) = \{G\}$

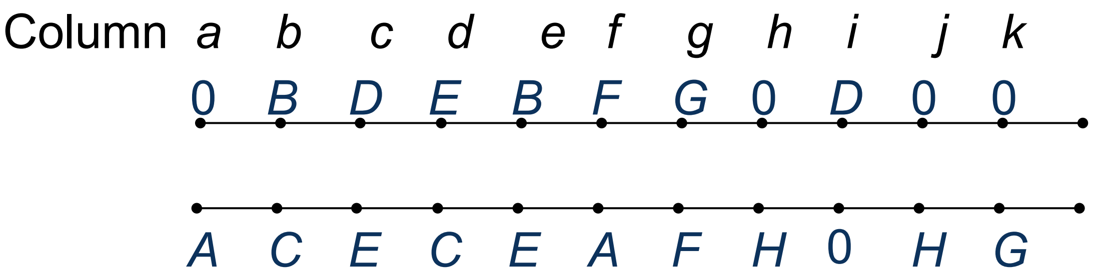




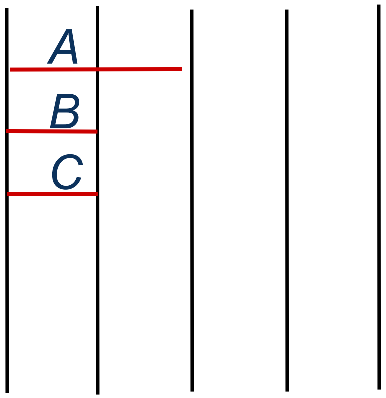
Zone representation

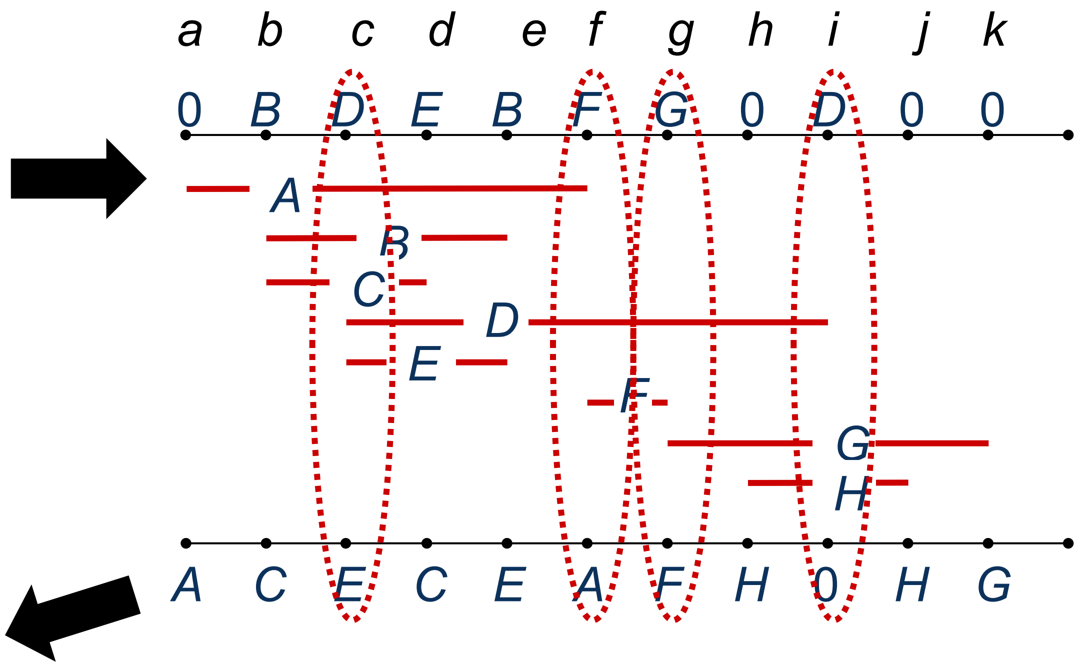
Lower bound on the number of tracks = 5

Zone Representation: {A, B, C}



$S(a) = \{A\}$
$S(b) = \{A,B,C\}$
$S(c) = \{A,B,C,D,E\}$
$S(d) = \{A,B,C,D,E\}$
$S(e) = \{A,B,D,E\}$
$S(f) = \{A,D,F\}$
$S(g) = \{D,F,G\}$
$S(h) = \{D,G,H\}$
$S(i) = \{D,G,H\}$
$S(j) = \{G,H\}$
$S(k) = \{G\}$

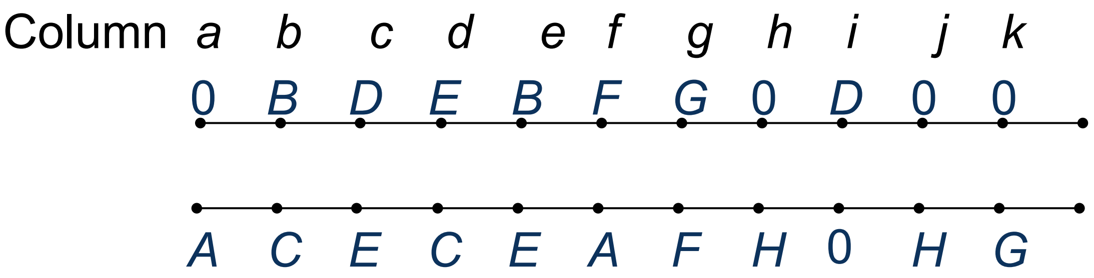


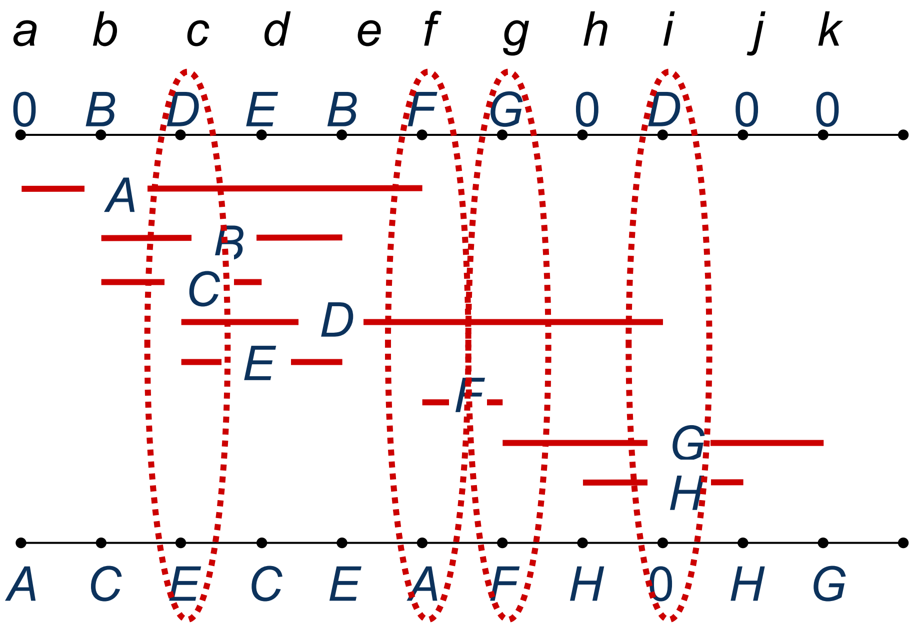


Zone representation

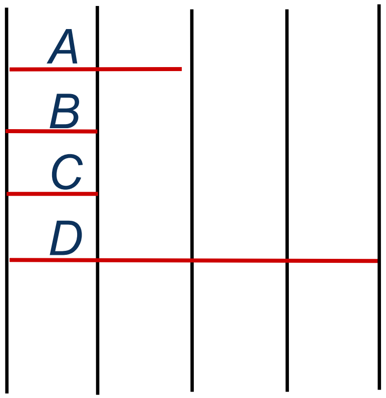
Lower bound on the number of tracks = 5

Zone Representation: {A, B, C, D}





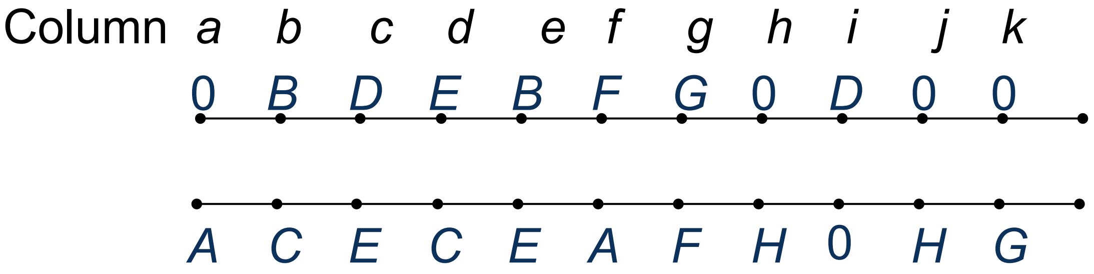
$S(a) = \{A\}$
$S(b) = \{A,B,C\}$
$S(c) = \{A,B,C,D,E\}$
$S(d) = \{A,B,C,D,E\}$
$S(e) = \{A,B,D,E\}$
$S(f) = \{A,D,F\}$
$S(g) = \{D,F,G\}$
$S(h) = \{D,G,H\}$
$S(i) = \{D,G,H\}$
$S(j) = \{G,H\}$
$S(k) = \{G\}$



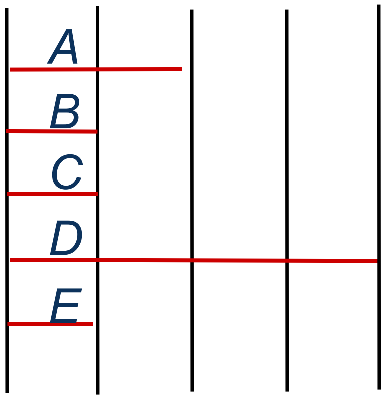
Zone representation

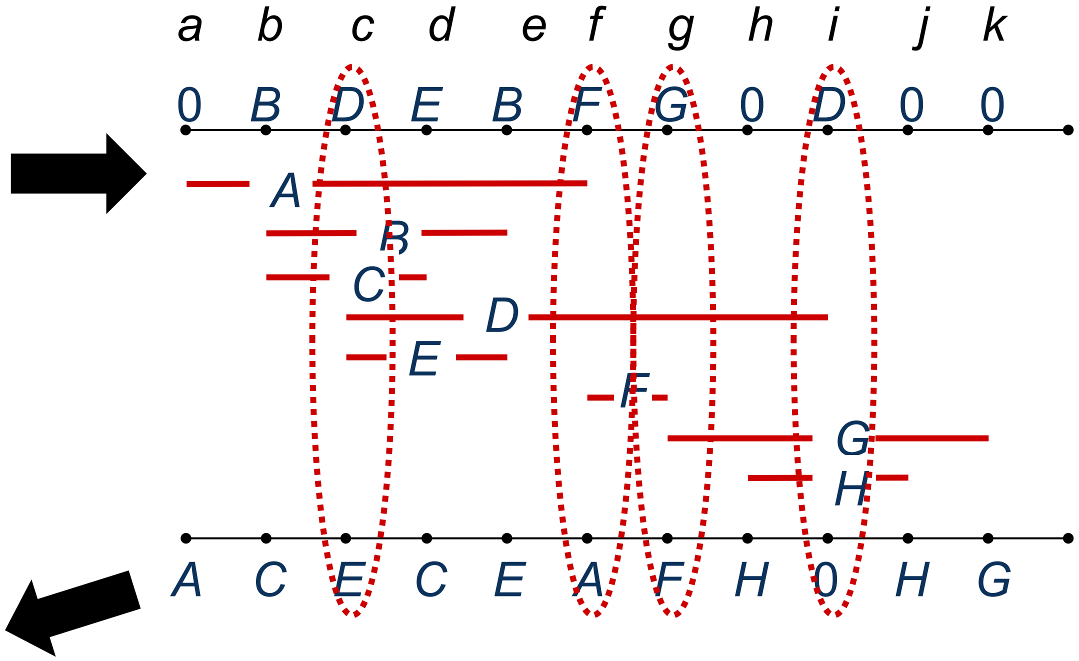
Lower bound on the number of tracks = 5

Zone Representation: {A, B, C, D, E}



$S(a) = \{A\}$
$S(b) = \{A,B,C\}$
$S(c) = \{A,B,C,D,E\}$
$S(d) = \{A,B,C,D,E\}$
$S(e) = \{A,B,D,E\}$
$S(f) = \{A,D,F\}$
$S(g) = \{D,F,G\}$
$S(h) = \{D,G,H\}$
$S(i) = \{D,G,H\}$
$S(j) = \{G,H\}$
$S(k) = \{G\}$

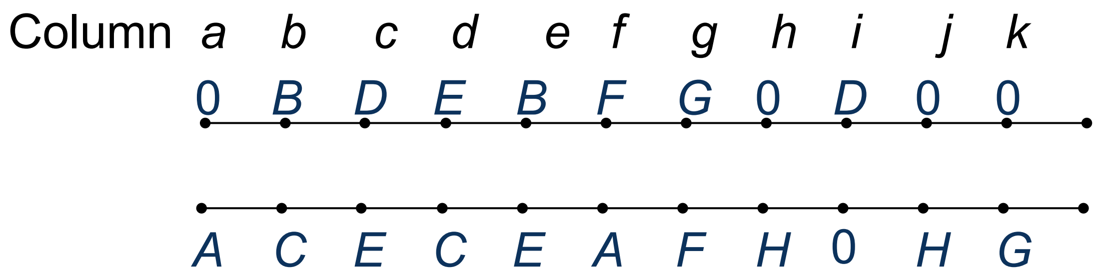


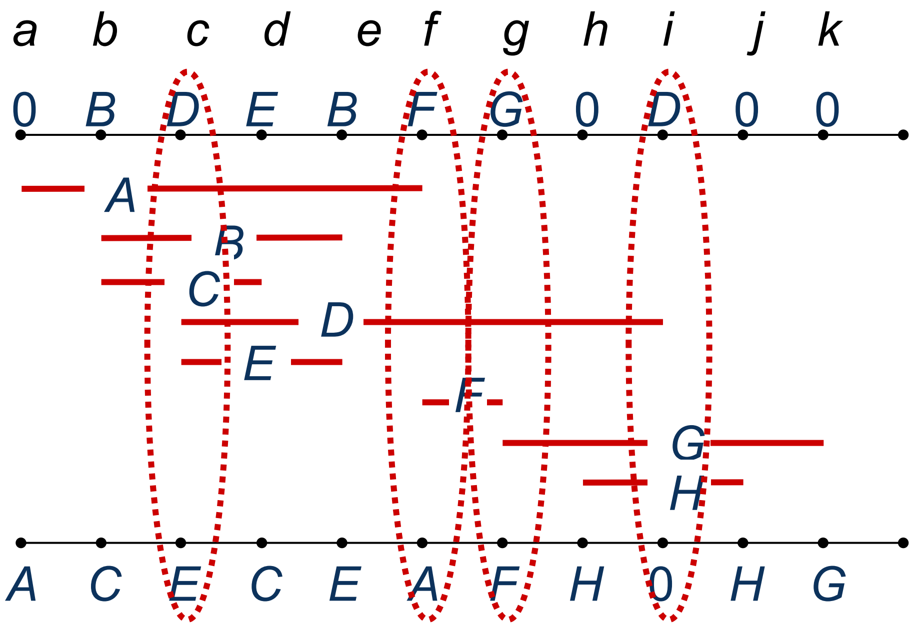


Zone representation

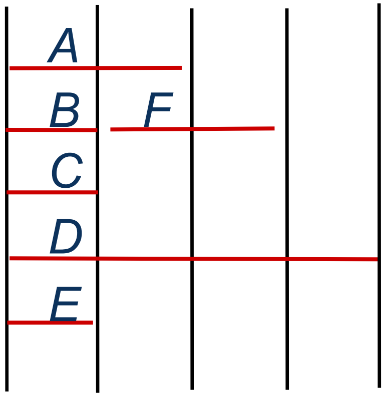
Lower bound on the number of tracks = 5

Zone Representation: {A, B, C, D, E, F}





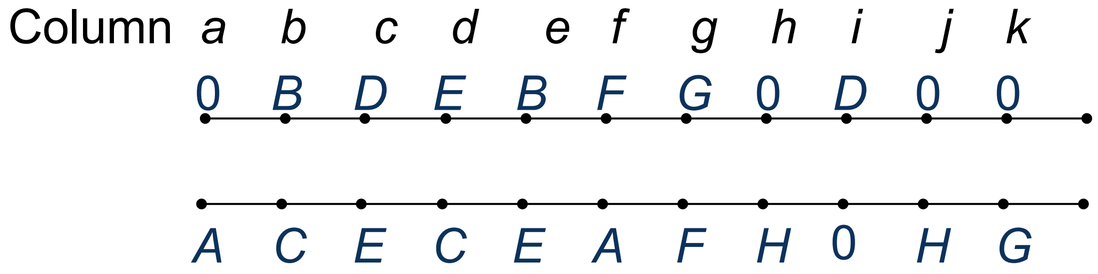
S(*a*) = {*A*}
S(*b*) = {*A*,*B*,*C*}
S(*c*) = {*A*,*B*,*C*,*D*,*E*}
S(*d*) = {*A*,*B*,*C*,*D*,*E*}
S(*e*) = {*A*,*B*,*D*,*E*}
S(*f*) = {*A*,*D*,*F*}
S(*g*) = {*D*,*F*,*G*}
S(*h*) = {*D*,*G*,*H*}
S(*i*) = {*D*,*G*,*H*}
S(*j*) = {*G*,*H*}
S(*k*) = {*G*}

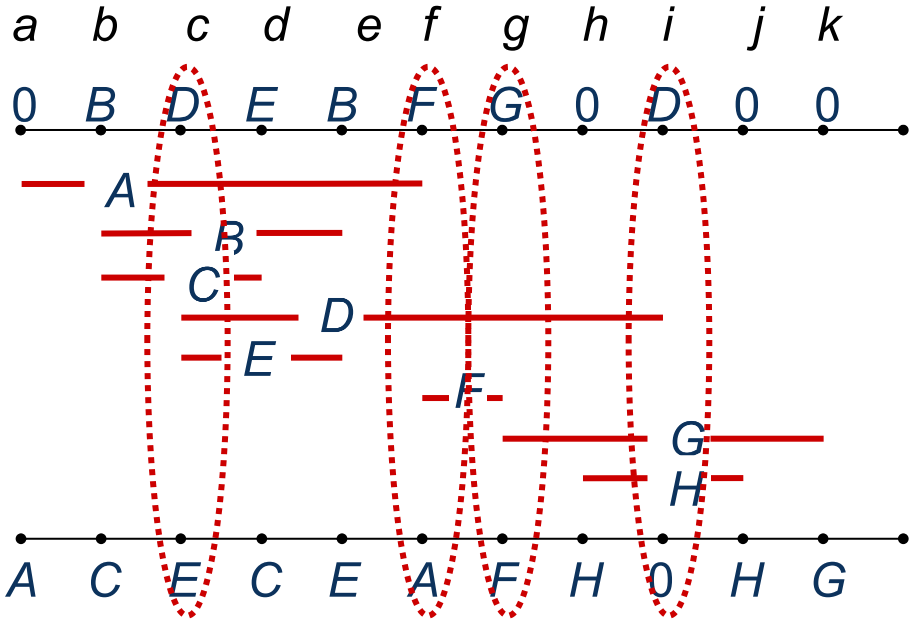


Zone representation

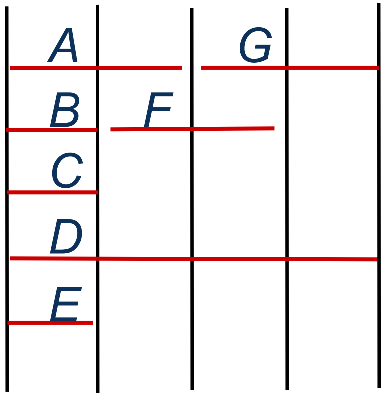
Lower bound on the number of tracks = 5

Zone Representation: {A, B, C, D, E, F, G}





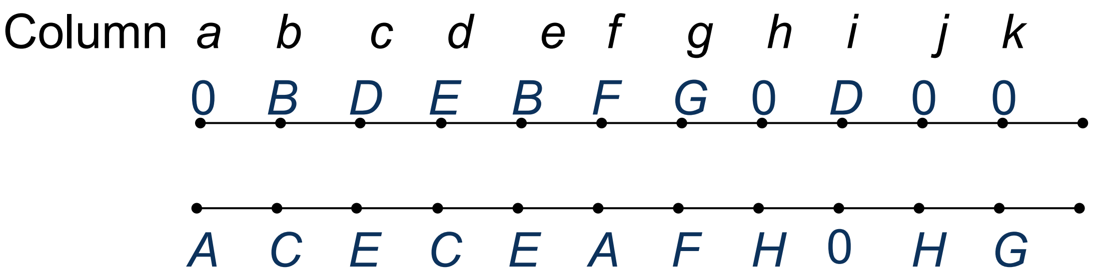
S(*a*) = {*A*}
S(*b*) = {*A*,*B*,*C*}
S(*c*) = {*A*,*B*,*C*,*D*,*E*}
S(*d*) = {*A*,*B*,*C*,*D*,*E*}
S(*e*) = {*A*,*B*,*D*,*E*}
S(*f*) = {*A*,*D*,*F*}
S(*g*) = {*D*,*F*,*G*}
S(*h*) = {*D*,*G*,*H*}
S(*i*) = {*D*,*G*,*H*}
S(*j*) = {*G*,*H*}
S(*k*) = {*G*}



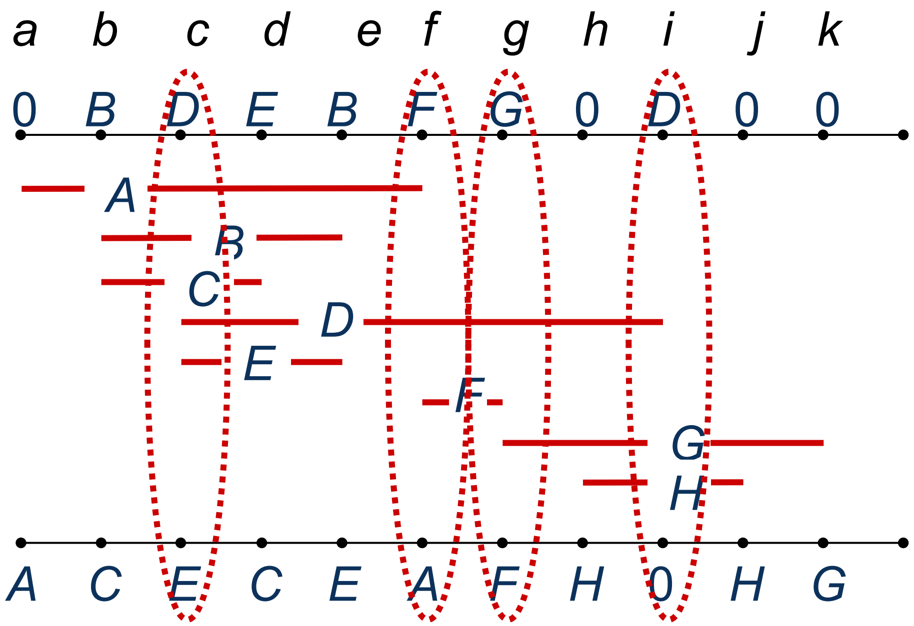
Zone representation

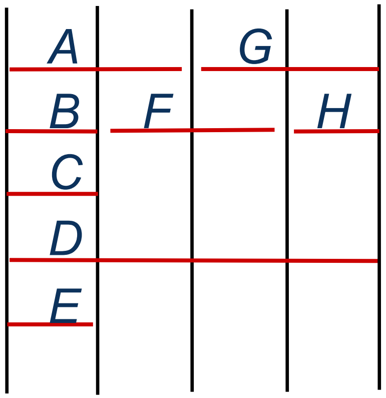
Lower bound on the number of tracks = 5

Zone Representation: {A, B, C, D, E, F, G, H}



$S(a) = \{A\}$
$S(b) = \{A,B,C\}$
$S(c) = \{A,B,C,D,E\}$
$S(d) = \{A,B,C,D,E\}$
$S(e) = \{A,B,D,E\}$
$S(f) = \{A,D,F\}$
$S(g) = \{D,F,G\}$
$S(h) = \{D,G,H\}$
$S(i) = \{D,G,H\}$
$S(j) = \{G,H\}$
$S(k) = \{G\}$



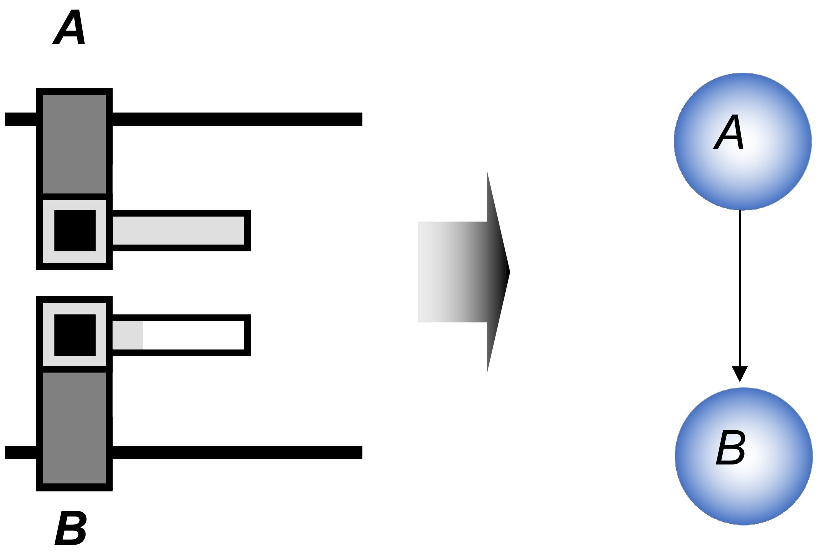


Zone representation

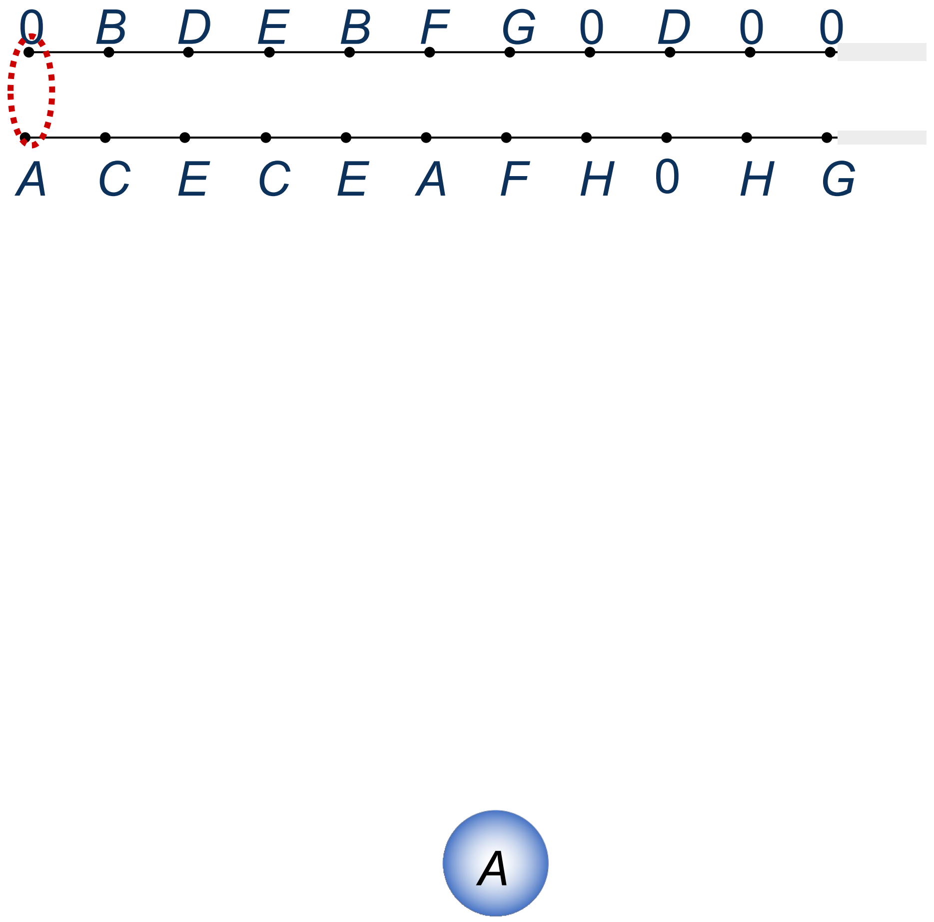
Lower bound on the number of tracks = 5

Vertical Constraint Graph

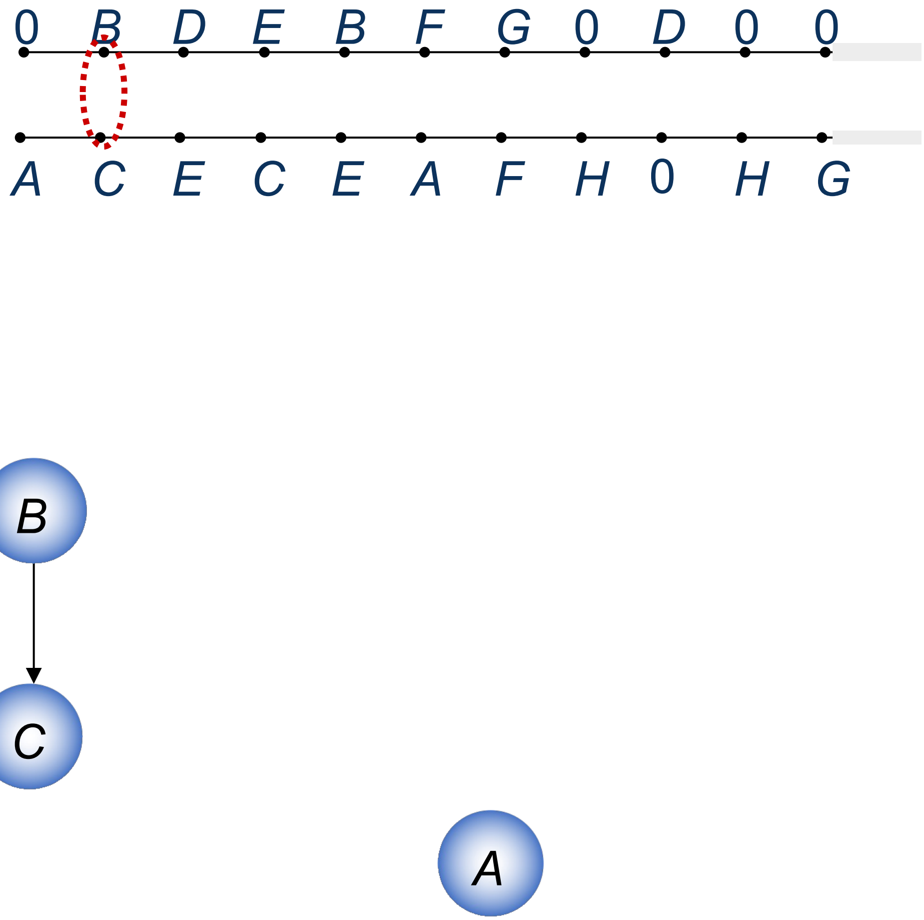
- A directed edge $e(i,j)$ connects nodes i and j if the horizontal segment of net i must be located above net j



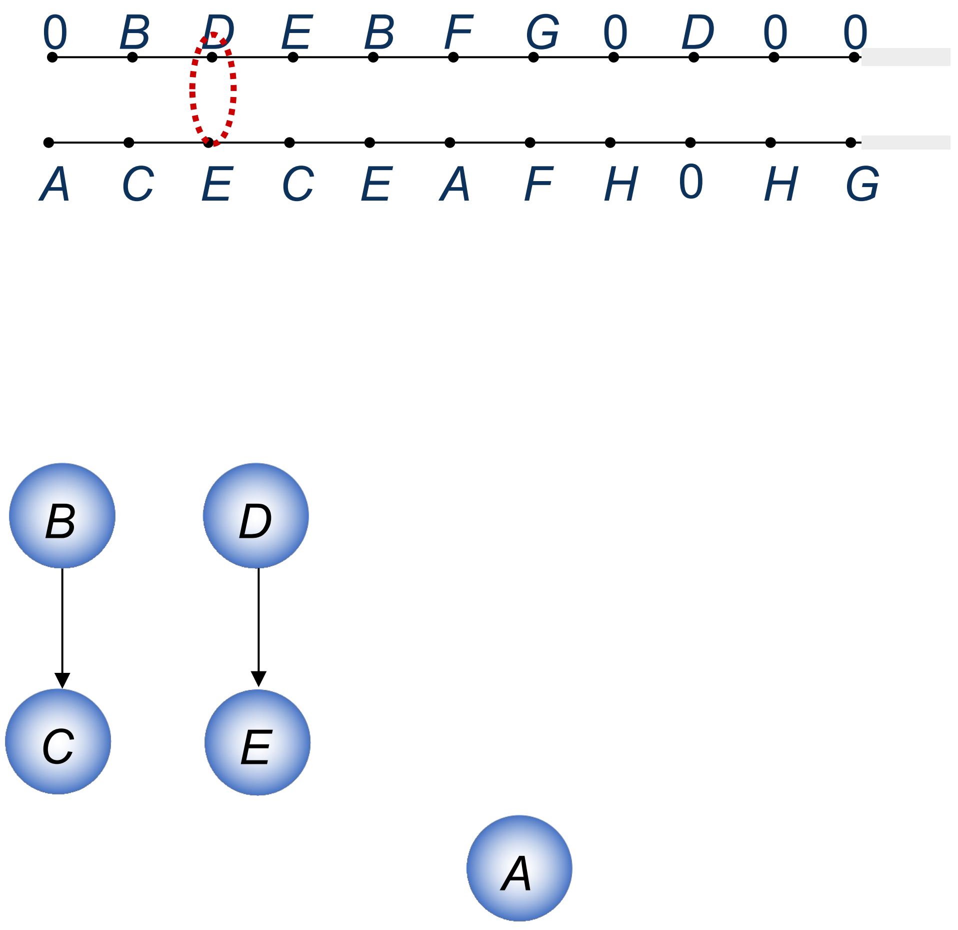
Vertical Constraint Graph



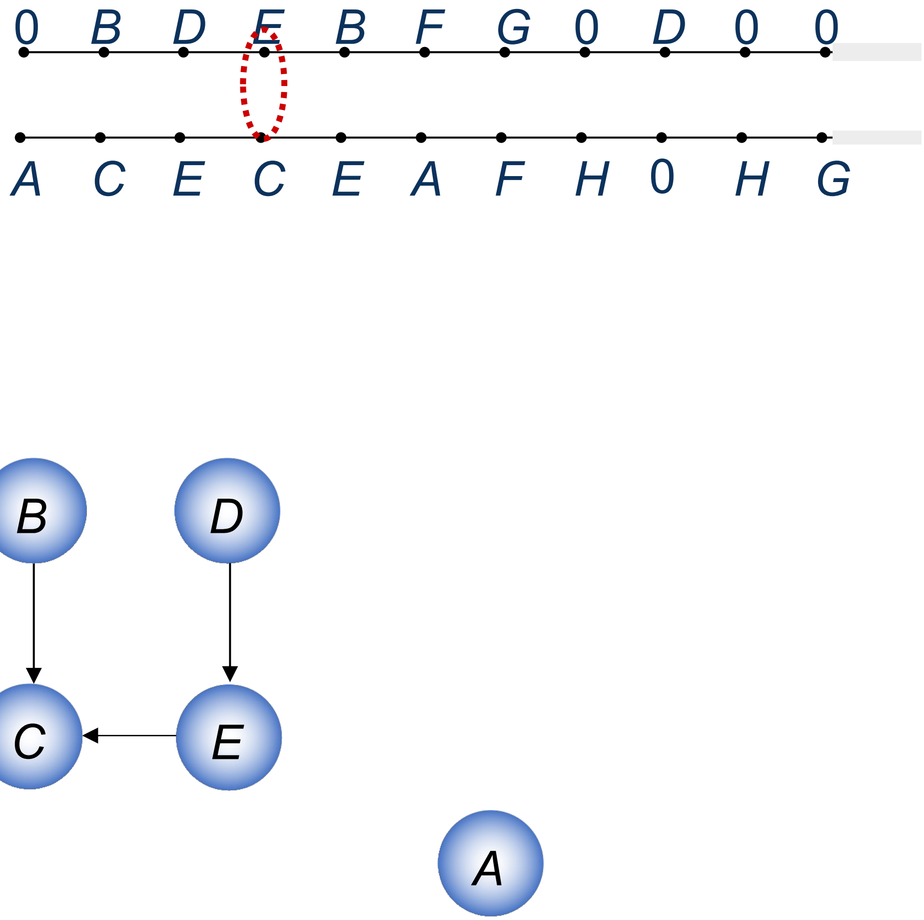
Vertical Constraint Graph



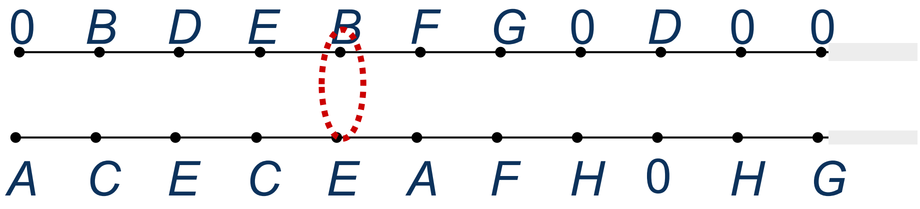
Vertical Constraint Graph

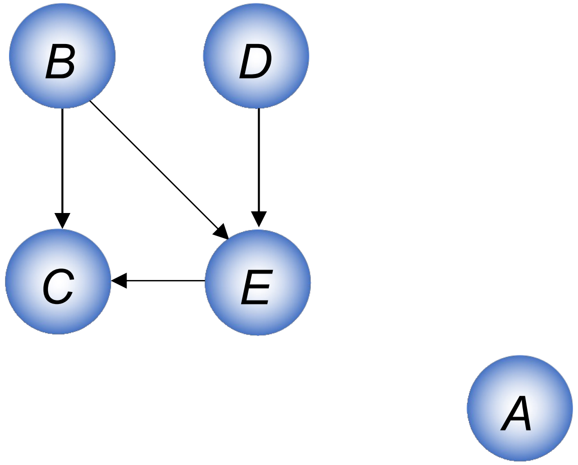


Vertical Constraint Graph



Vertical Constraint Graph

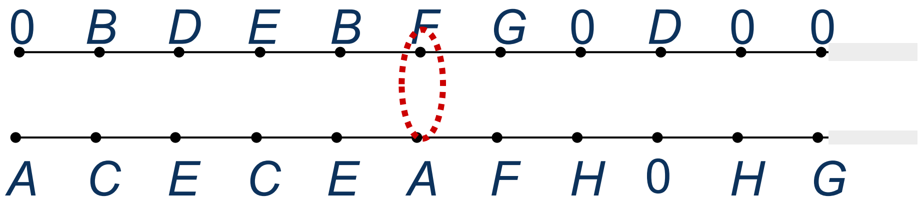


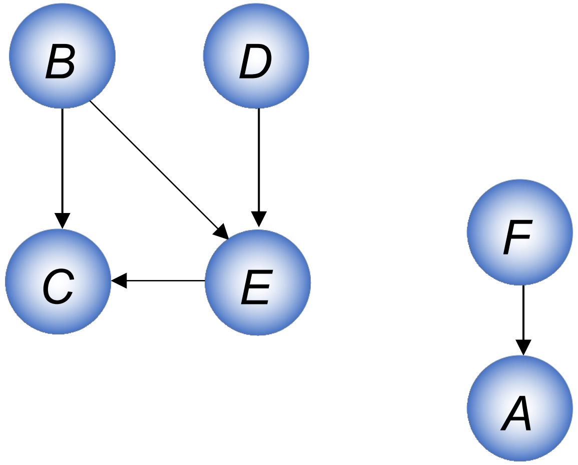


Vertical Constraint Graph (VCG)

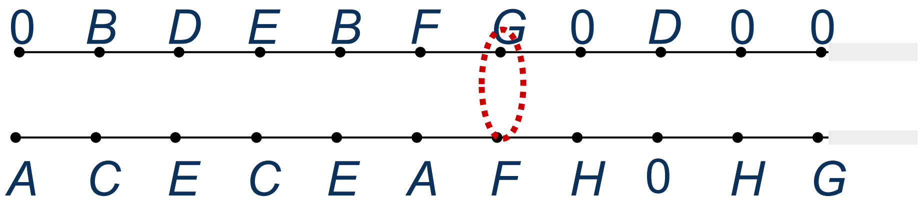
Note: an edge that can be derived by transitivity is not included, such as edge (*B*,*C*)

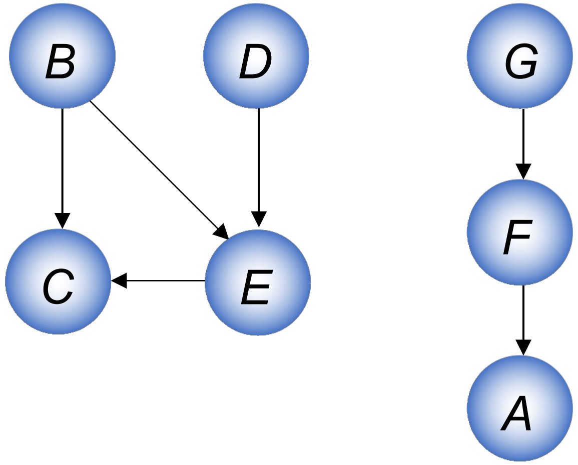
Vertical Constraint Graph



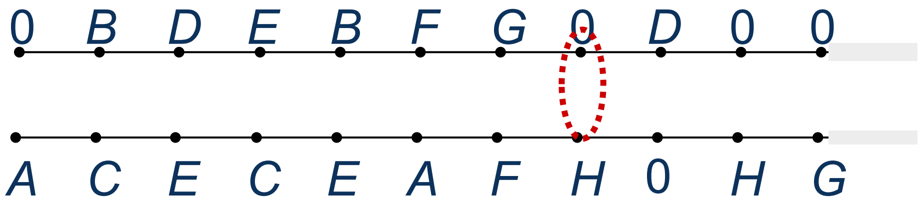


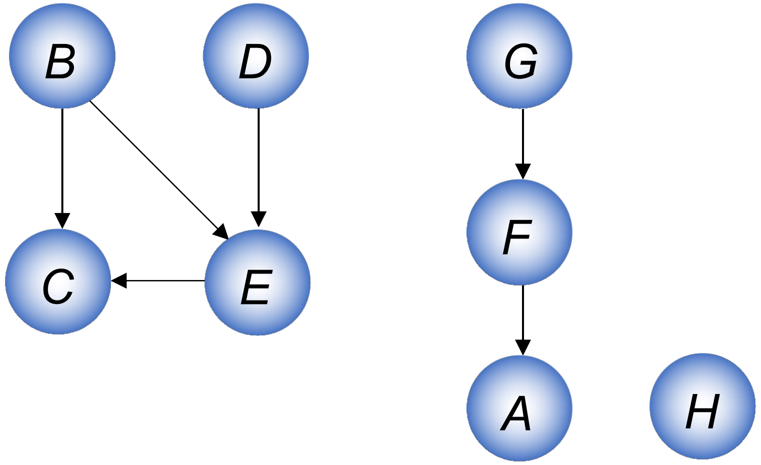
Vertical Constraint Graph



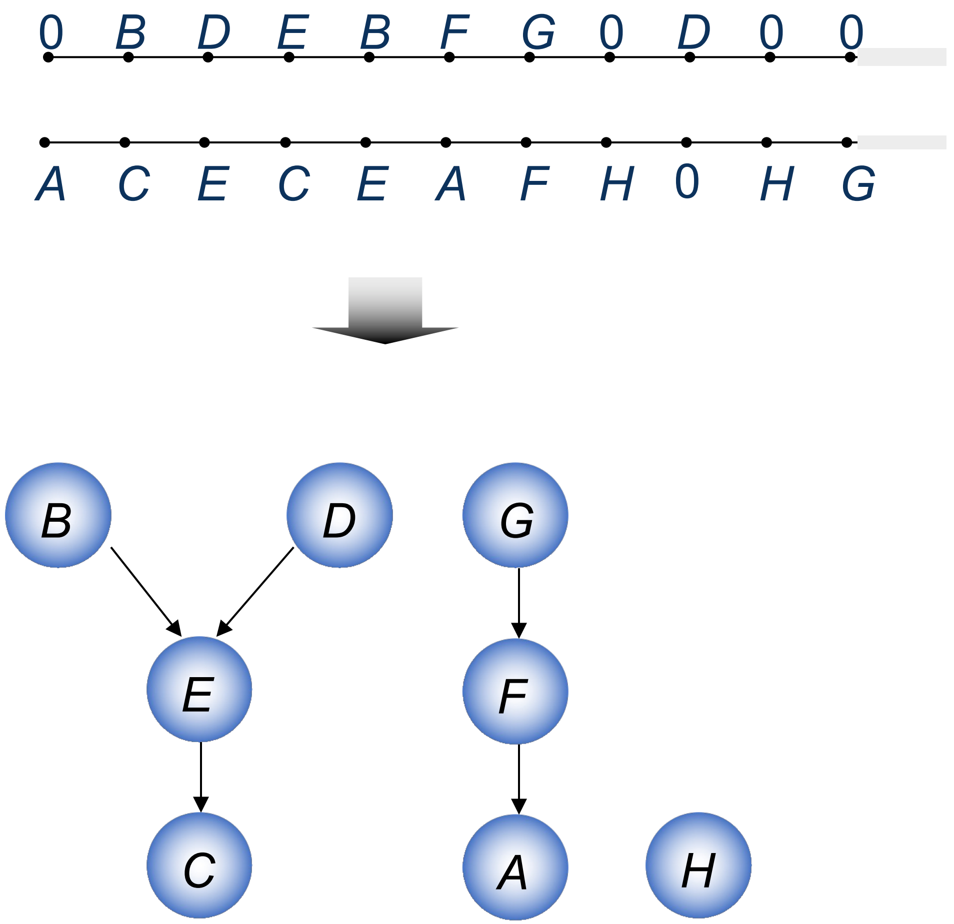


Vertical Constraint Graph

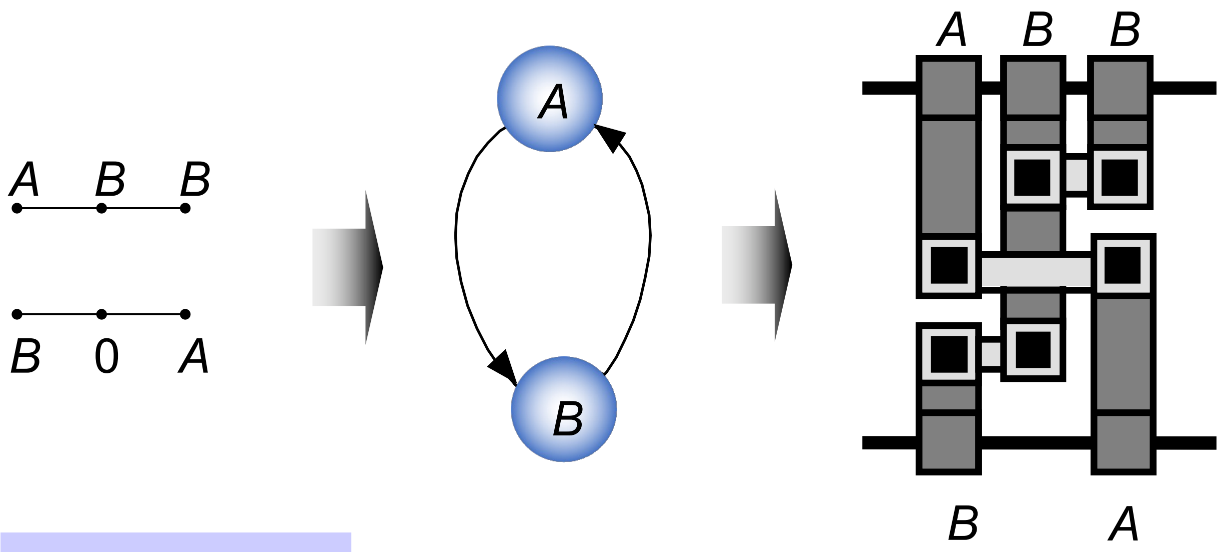




Vertical Constraint Graph – Final



Is Vertical Constrain Graph Acyclic?



Cyclic conflict

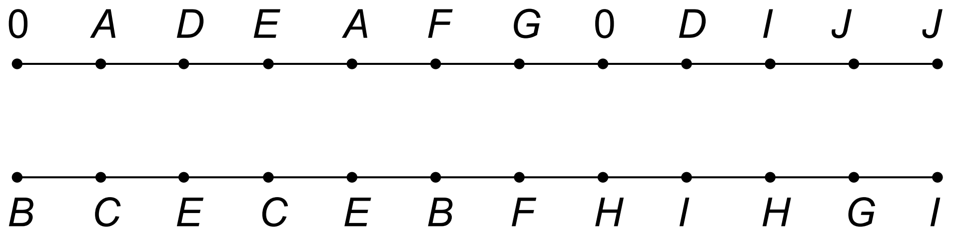
Left-Edge Channel Routing Algorithm

- **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**

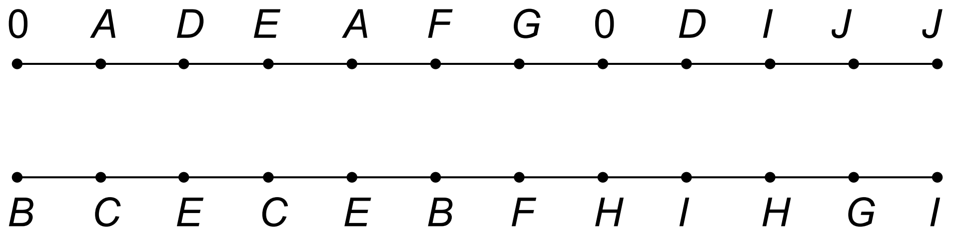
Left-Edge Algorithm: Pseudo Code

```
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
```

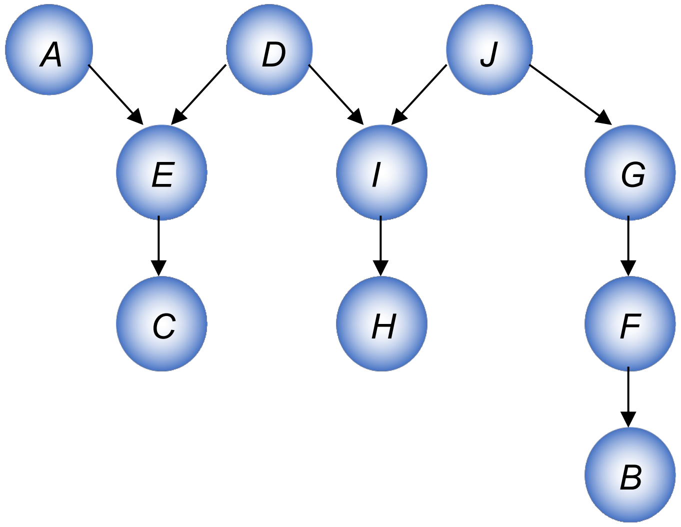
Walkthrough – I

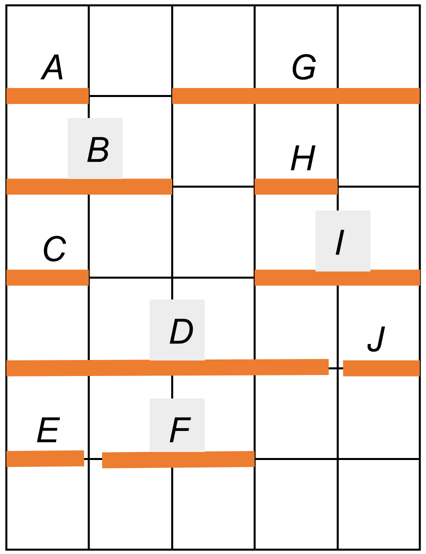


Walkthrough – II

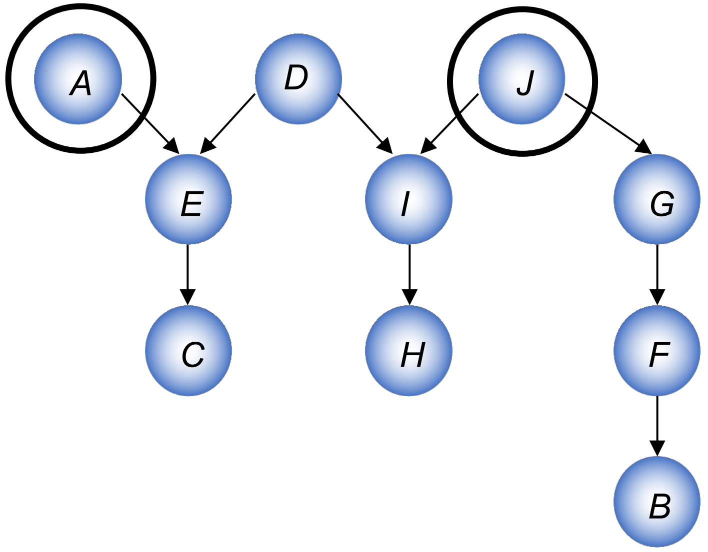


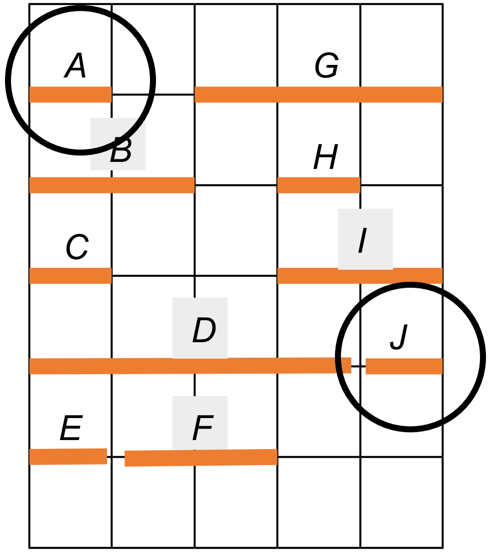
1. Generate VCG and zone representation





Walkthrough – III





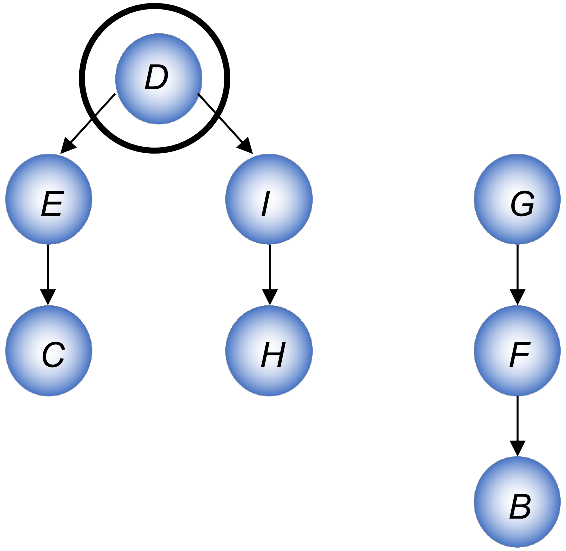
2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track* assign *curr_net*

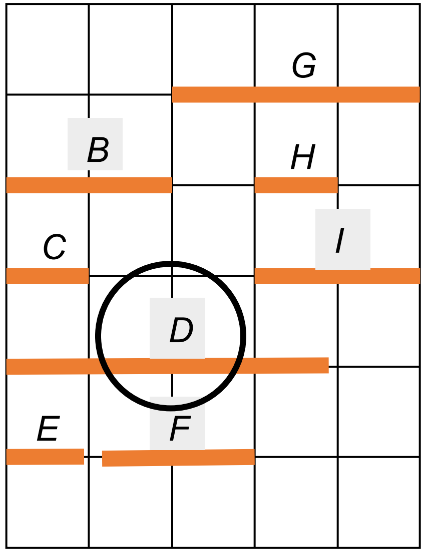
curr_track = 1: Net *A*	Net *J*

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

Walkthrough – IV

Walkthrough – V



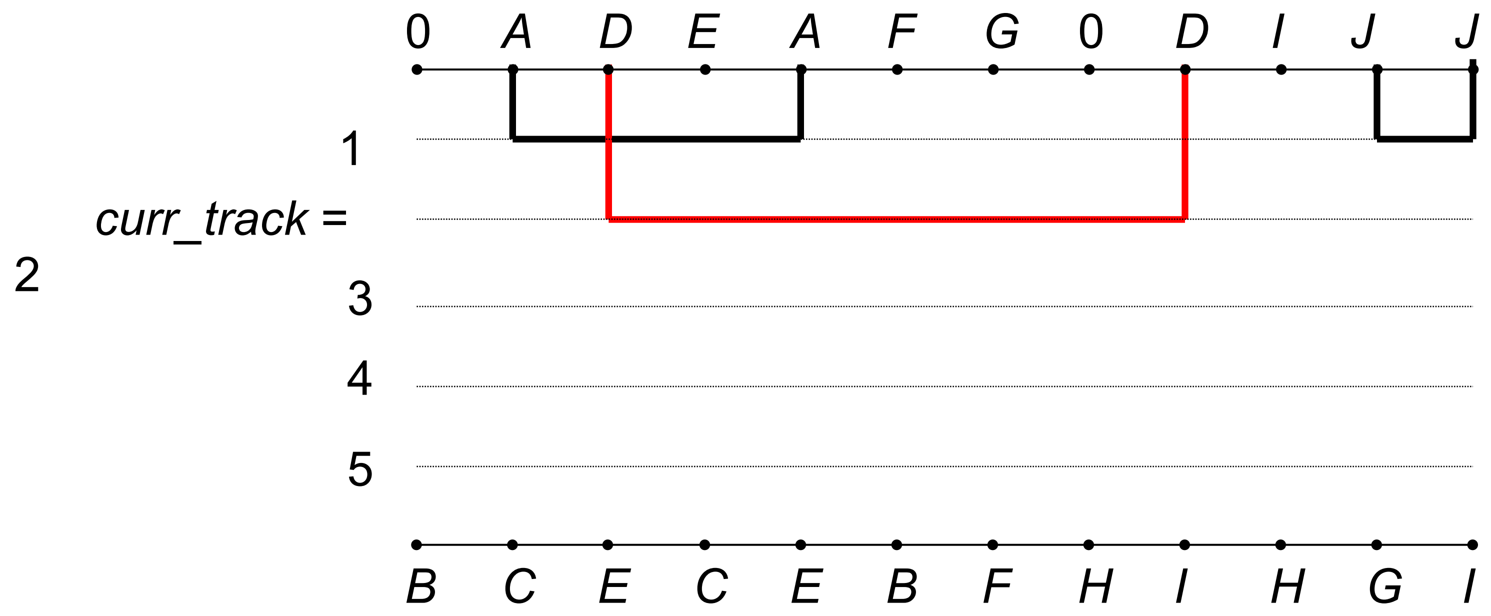


2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track*
   assign *curr_net*

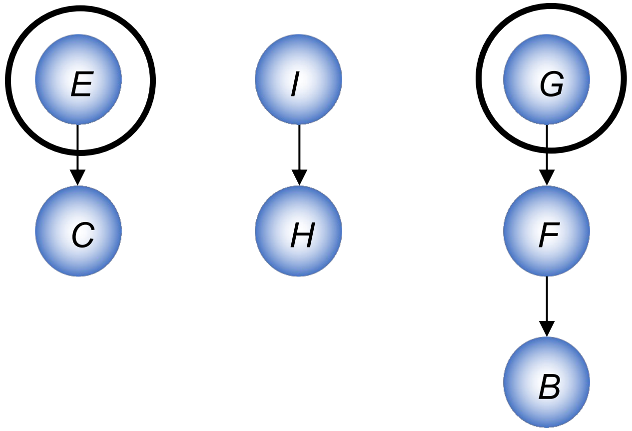
curr_track = 2: Net *D*

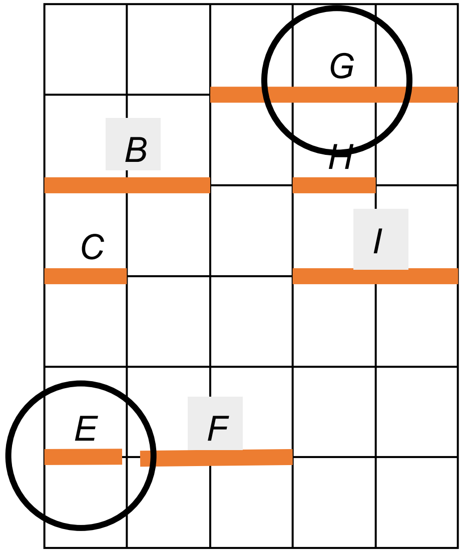
4. Delete placed nets (*D*) in VCG and zone representation

Walkthrough – VI



Walkthrough – VII



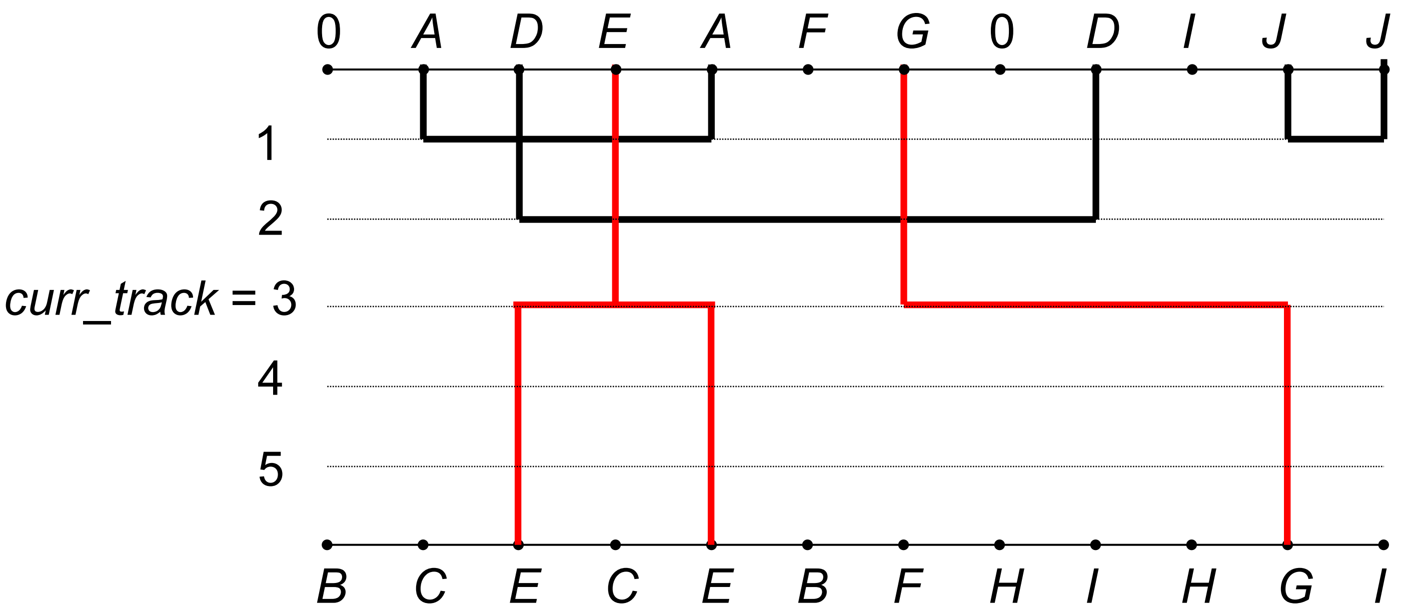


2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track*
   assign *curr_net*

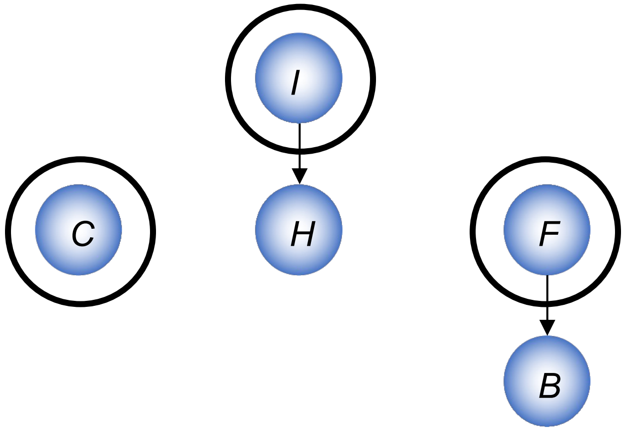
curr_track = 3:   Net *E*   Net *G*

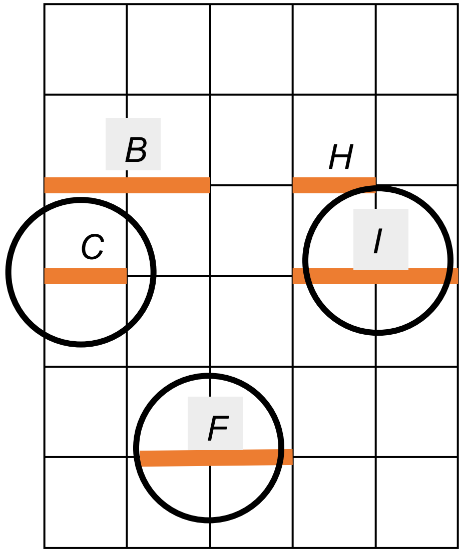
4. Delete placed nets (*E*, *G*) in VCG and zone representation

Walkthrough – VIII



Walkthrough – IX



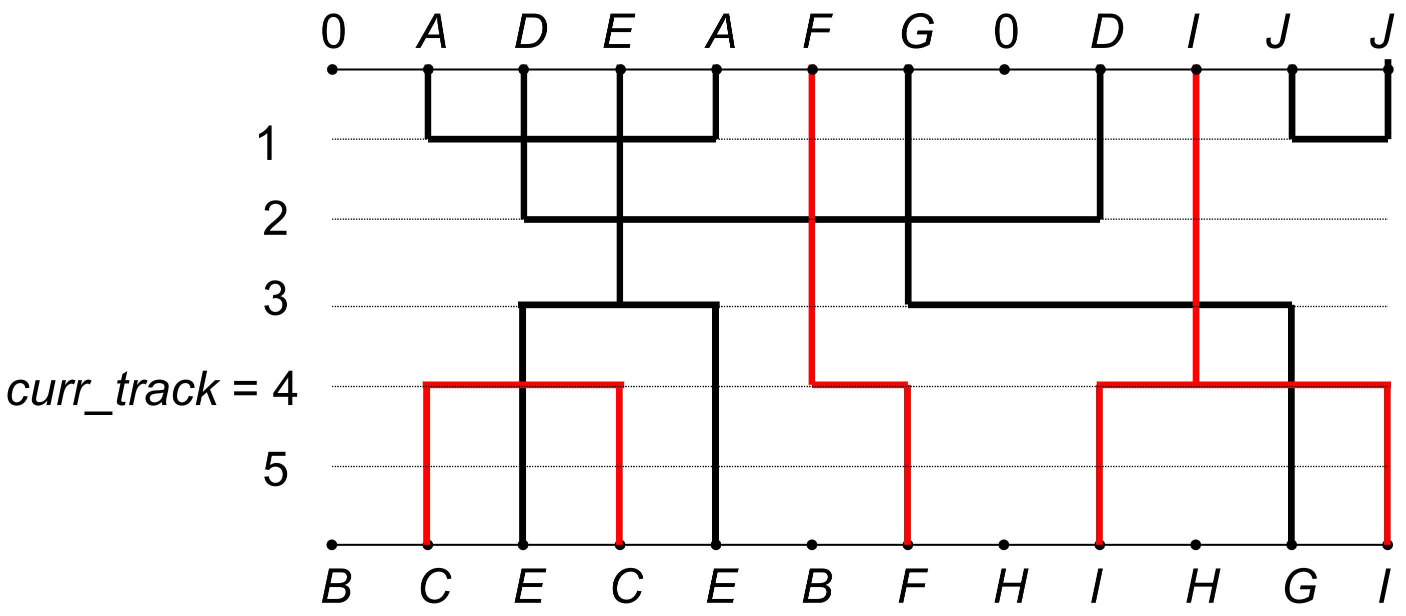


2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track* assign *curr_net*

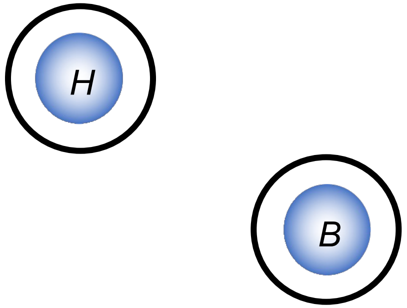
curr_track = 4: Net *C* | Net *F* | Net *I*

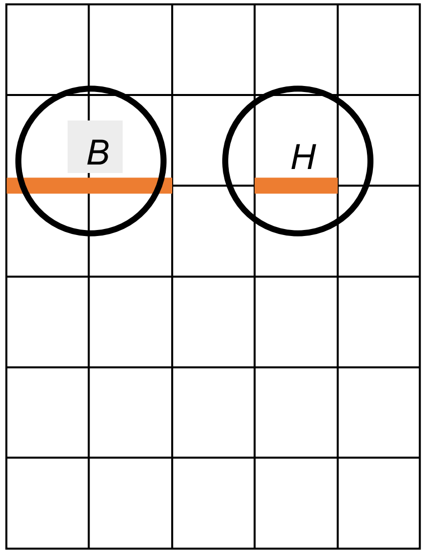
4. Delete placed nets (*C*, *F*, *I*) in VCG and zone representation

Walkthrough – X



Walkthrough – XI



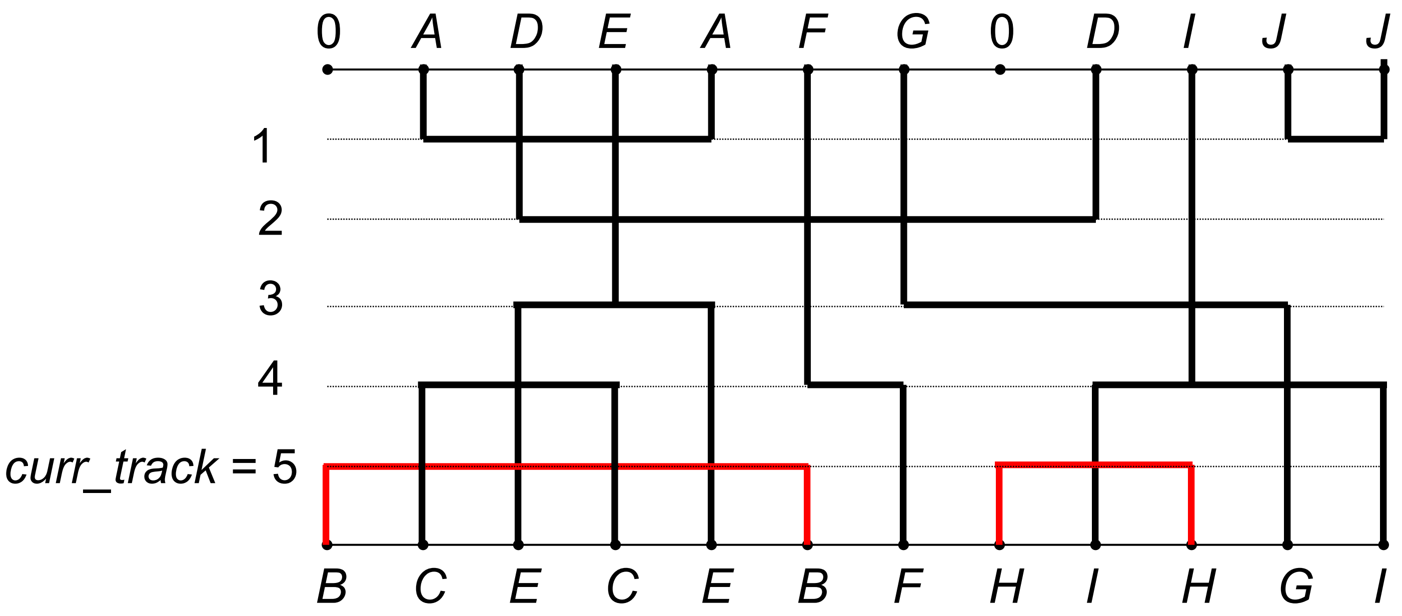


2. Consider next track
3. Find left-to-right ordering of all unassigned nets
   If *curr_net* has no parents and does not cause conflicts on *curr_track*
   assign *curr_net*

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

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

Walkthrough – XII



Routing result

Summary

- **We have discussed channel routing**
  - Horizontal constraint graph
  - Vertical constraint graph
- **We have discussed left-edge routing algorithm**
  - Vertical constraint graph & zone representation
  - Greedy assignment to maximize the utilization
  - Assign a new track whenever needed