# Global Routing

Presented By:
**Sridhar H Rangarajan**
**IBM STG India Enterprise Systems Development**

# Global Routing

Introduction

Terminology and Definitions

Optimization Goals

Representations of Routing Regions

The Global Routing Flow

Single-Net Routing

- Rectilinear Routing
- Global Routing in a Connectivity Graph
- Finding Shortest Paths with Dijkstra's Algorithm
- Finding Shortest Paths with A* Search

Full-Netlist Routing

- Routing by Integer Linear Programming
- Rip-Up and Reroute (RRR)

Modern Global Routing

- Pattern Routing
- Negotiated-Congestion Routing

# Introduction

Given a placement, a netlist and technology information,

- determine the necessary wiring, e.g., net topologies and specific routing segments, to connect these cells
- while respecting constraints, e.g., design rules and routing resource capacities, and
- optimizing routing objectives, e.g., minimizing total wirelength and maximizing timing slack.

# Introduction

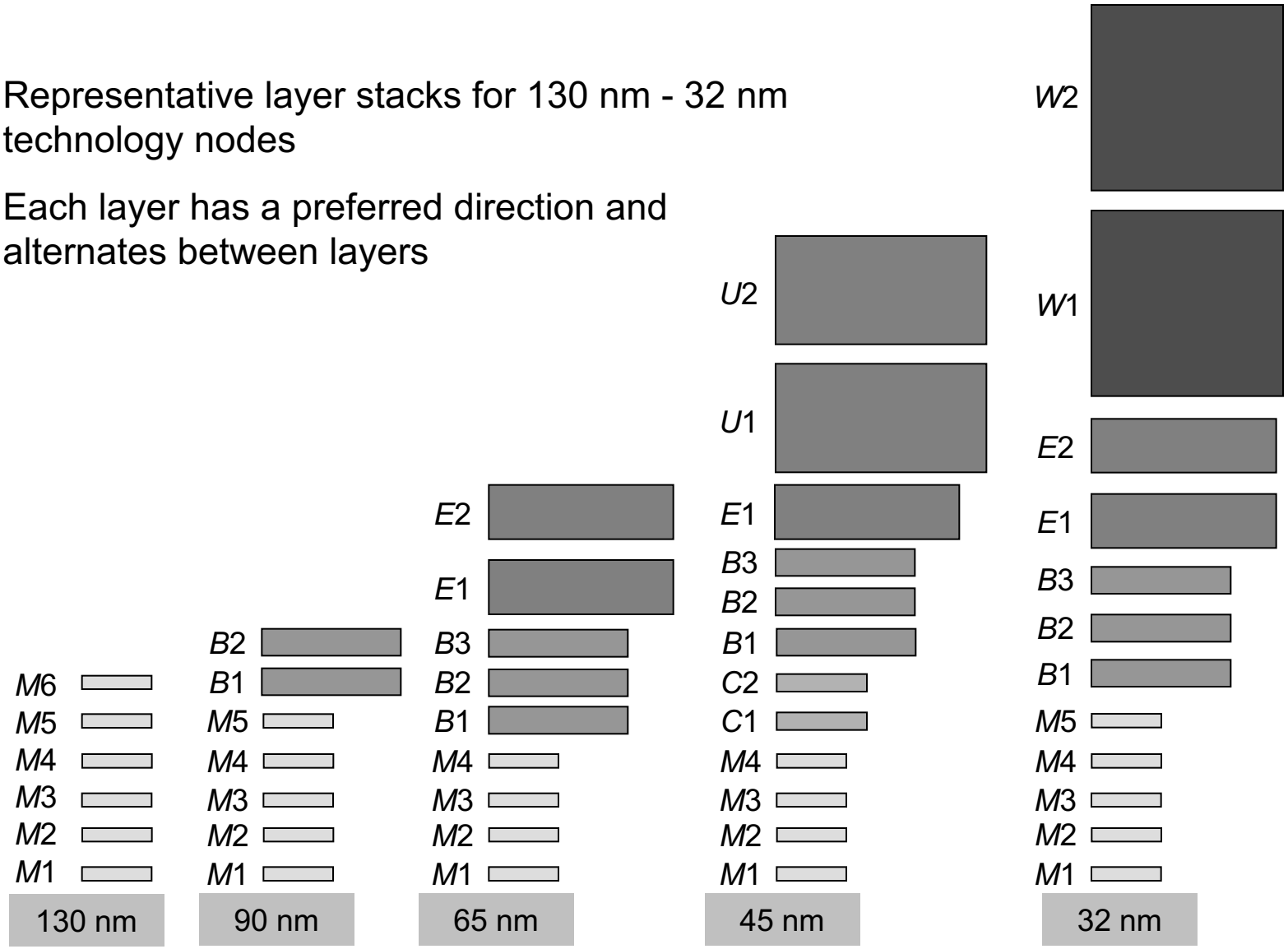
Terminology:

- Net: Set of two or more pins that have the same electric potential
- Netlist: Set of all nets.
- Congestion: Where the shortest routes of several nets are incompatible because they traverse the same tracks.
- Fixed-die routing: Chip outline and routing resources are fixed.
- Variable-die routing: New routing tracks can be added as needed.

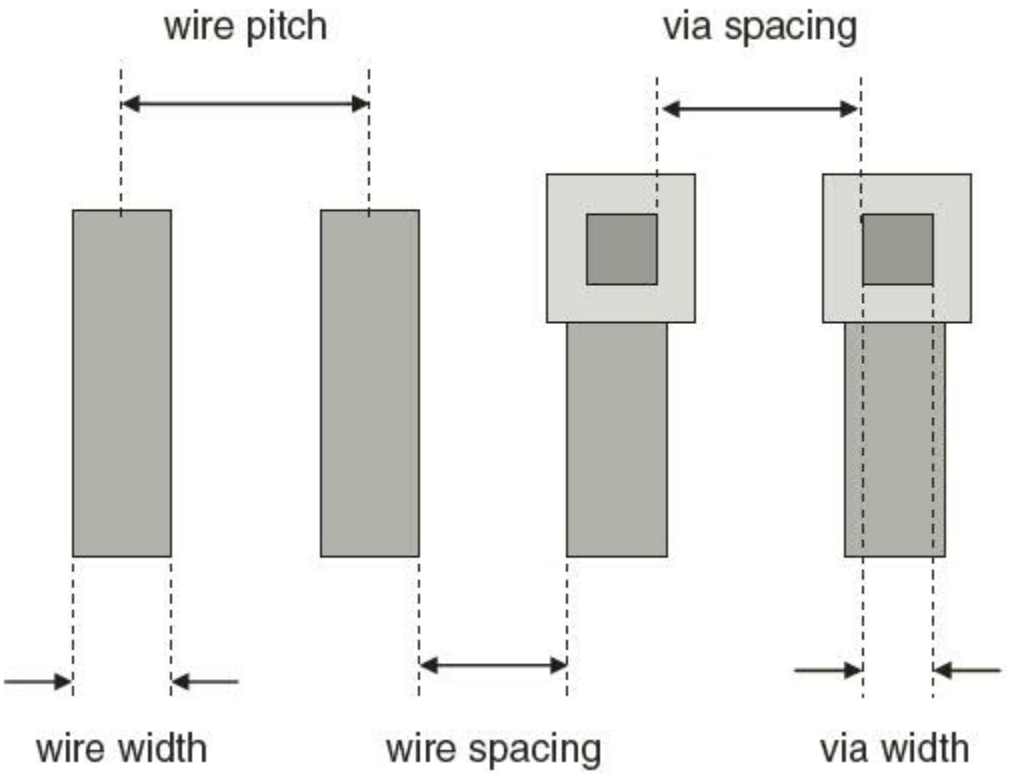
# Introduction

Representative layer stacks for 130 nm - 32 nm technology nodes

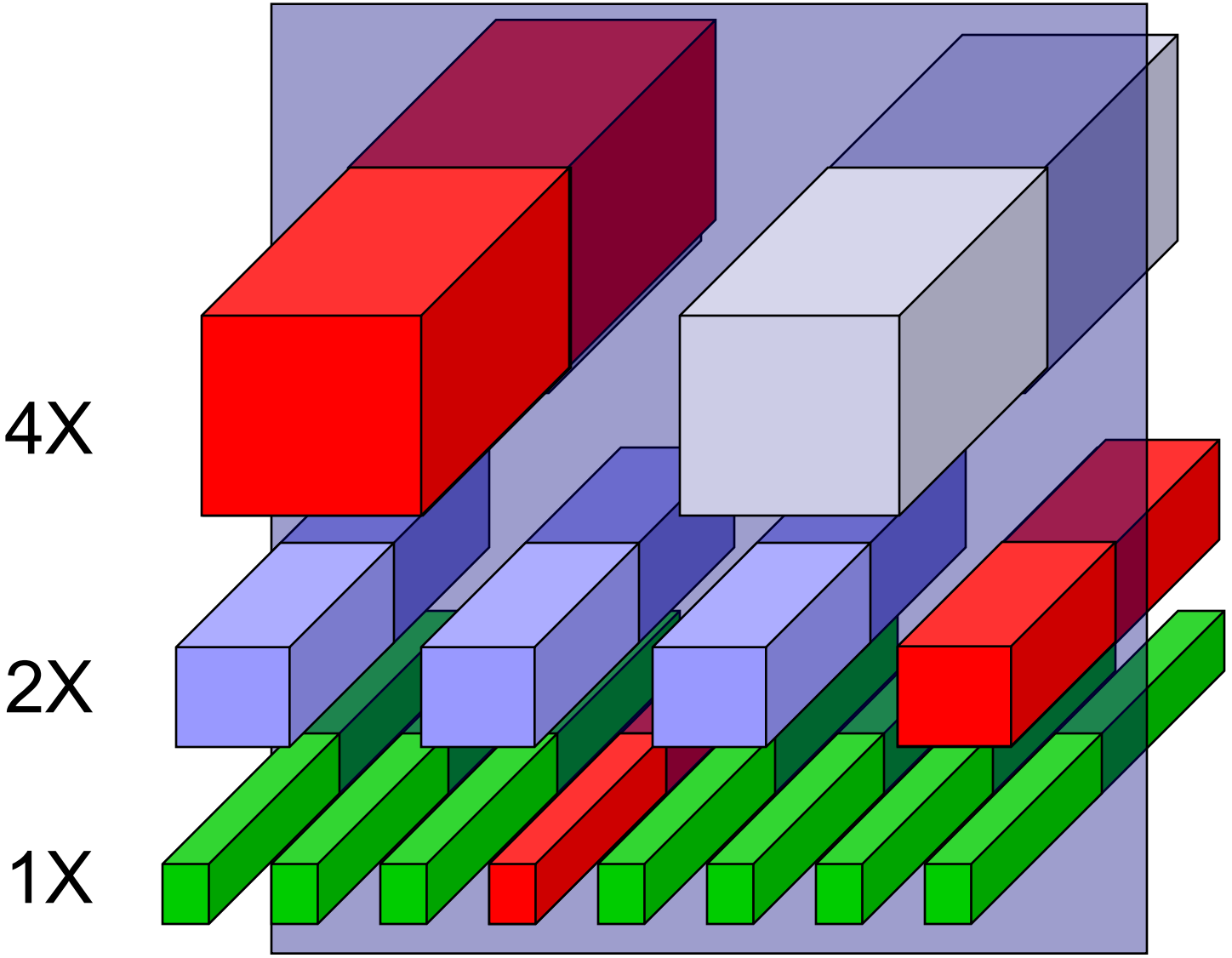
Each layer has a preferred direction and alternates between layers



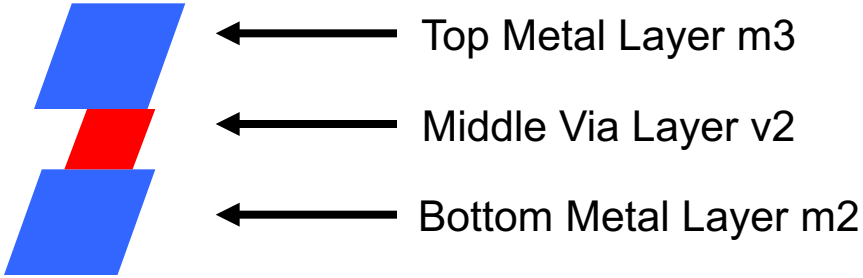
# Introduction - Pitch

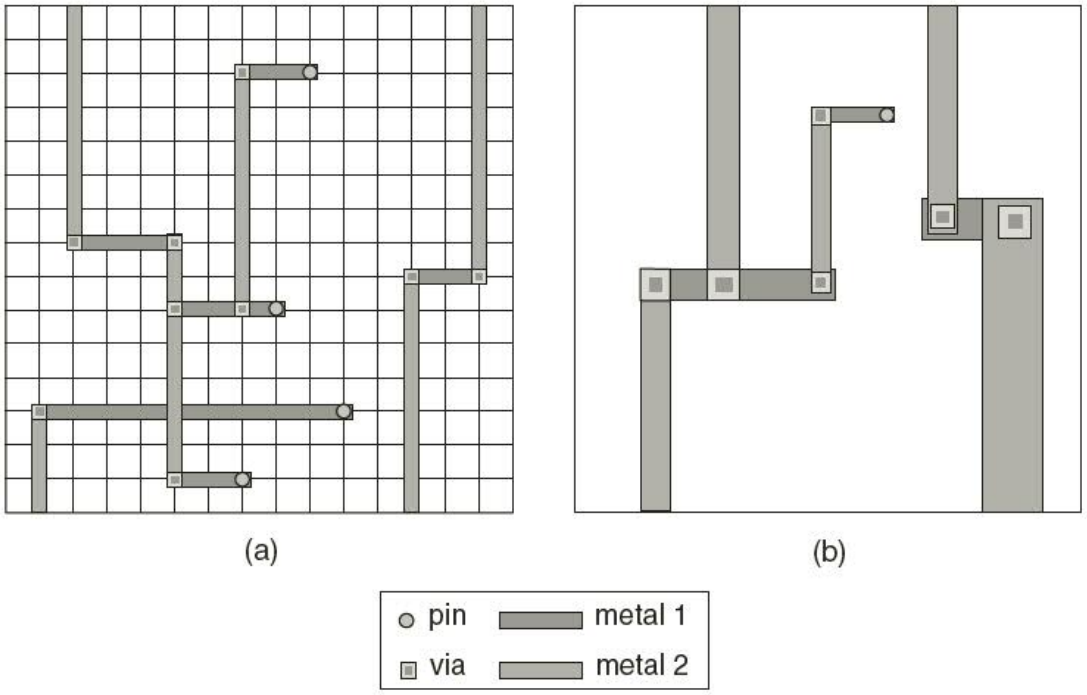


# Introduction – Different wire sizes



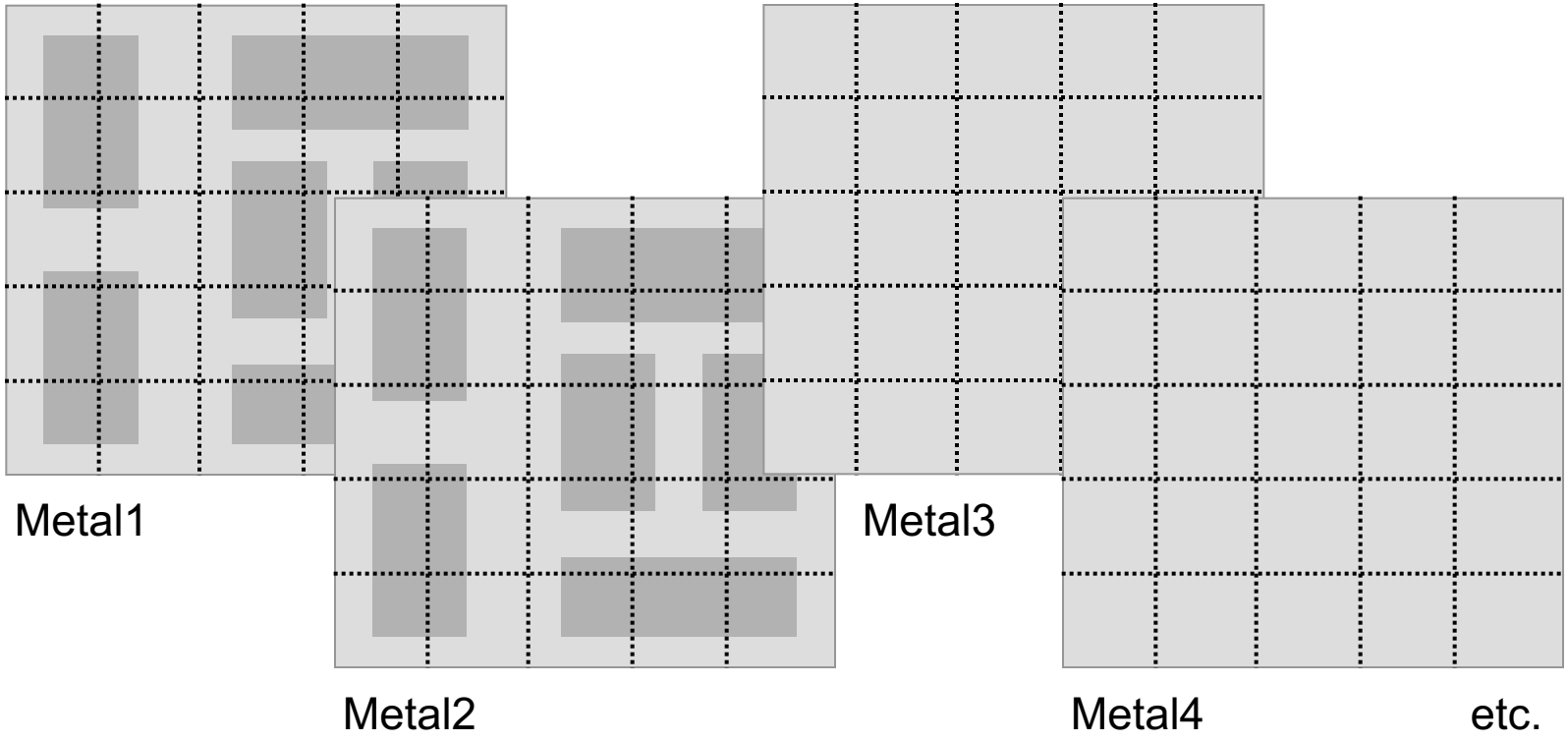
# Introduction - Via



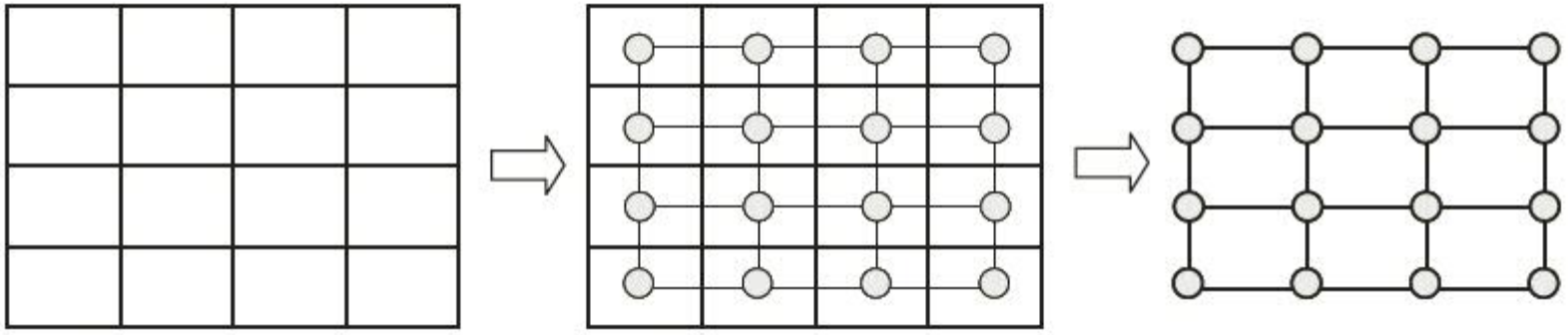



# Introduction - GCell

Gcells (Tiles) with macro cell layout

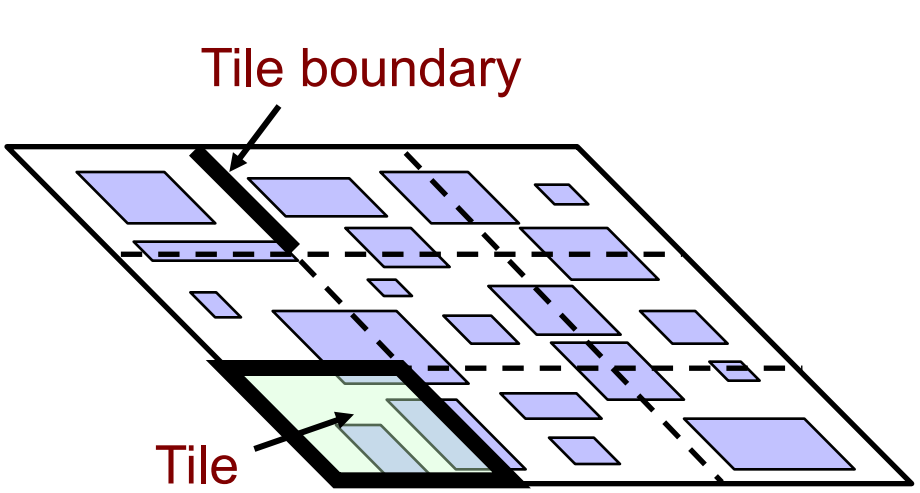


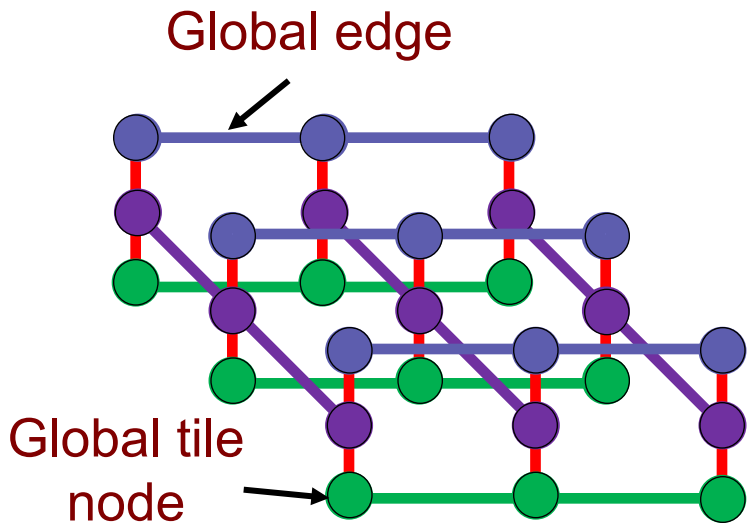
# Introduction - GCell



# Introduction - GCell

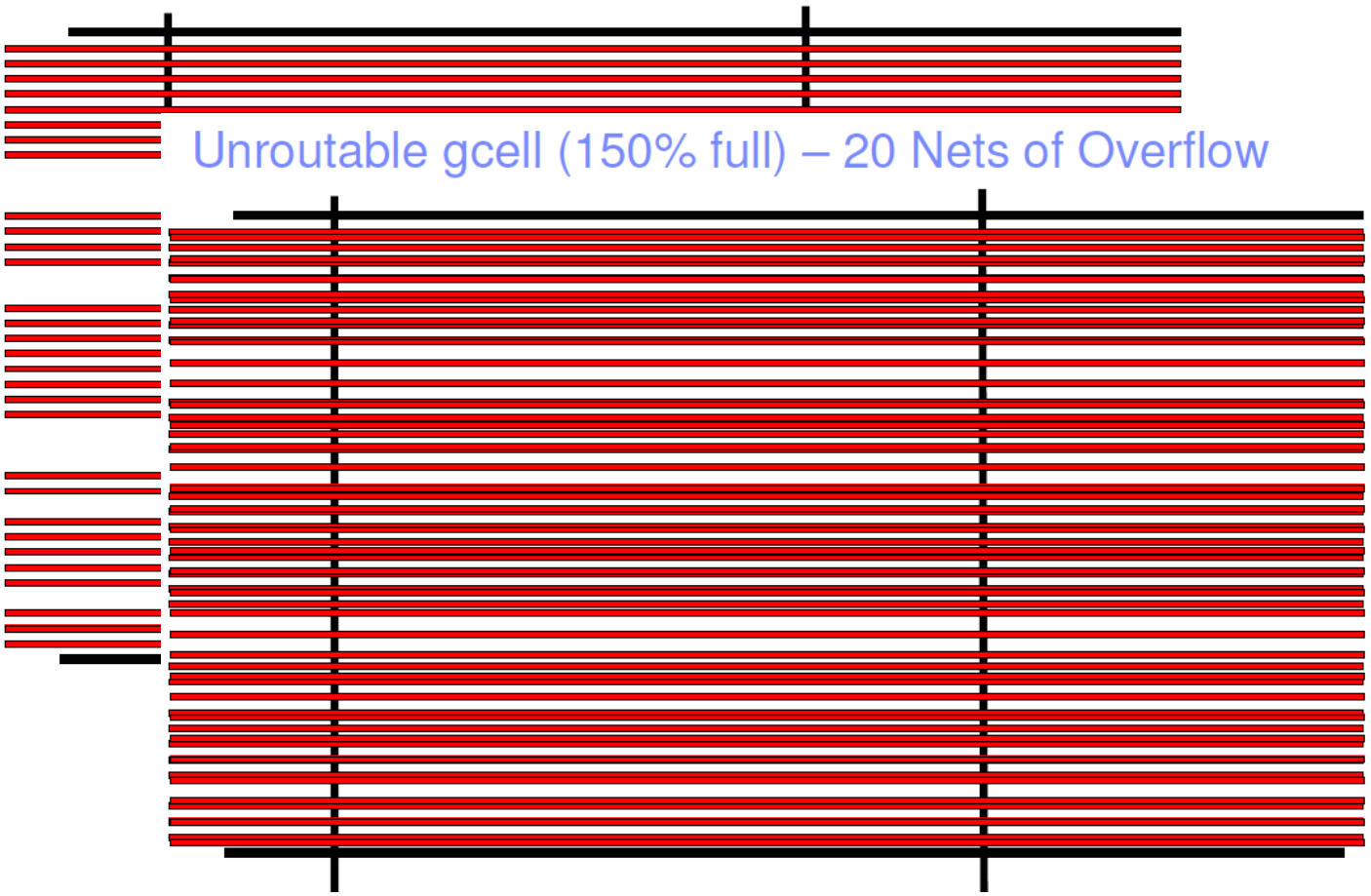
- Global tile node & global edge
- Overflow: the amount of routing demand that exceeds the given capacity



# Introduction - GCell

- Total Overflow



# Introduction: General Routing Problem

Netlist:

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

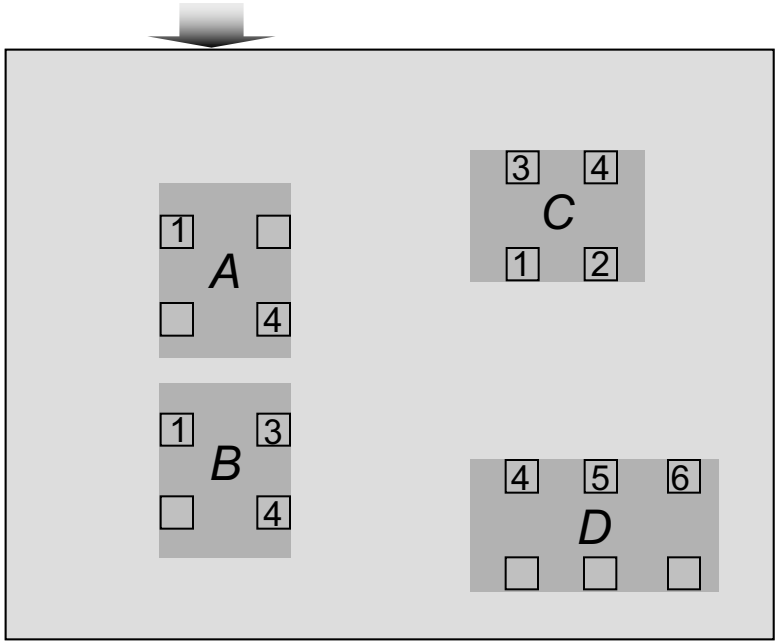
$N_2 = \{D_4, B_4, C_1, A_4\}$

$N_3 = \{C_2, D_5\}$

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

Technology Information
(Design Rules)

Placement result



# Introduction: General Routing Problem

Netlist:

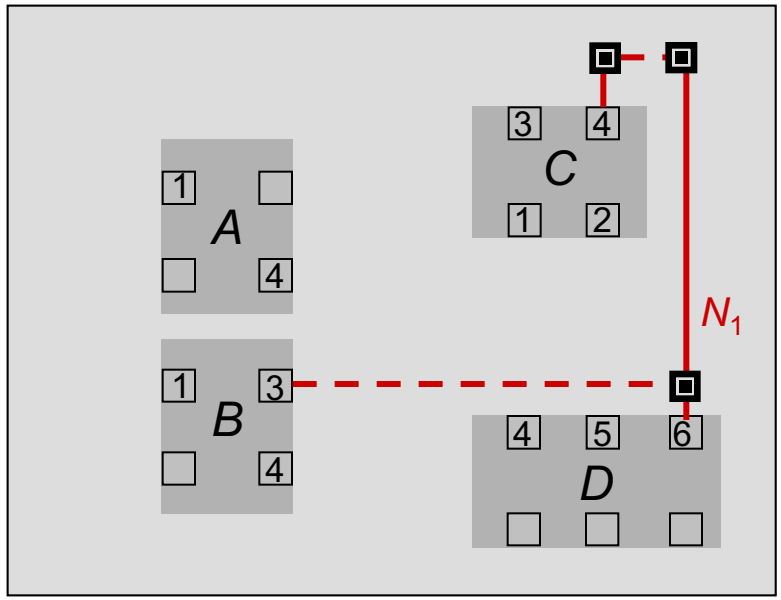
$N_1 = \{C_4, D_6, B_3\}$

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

$N_3 = \{C_2, D_5\}$

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

Technology Information
(Design Rules)



# Introduction: General Routing Problem

Netlist:

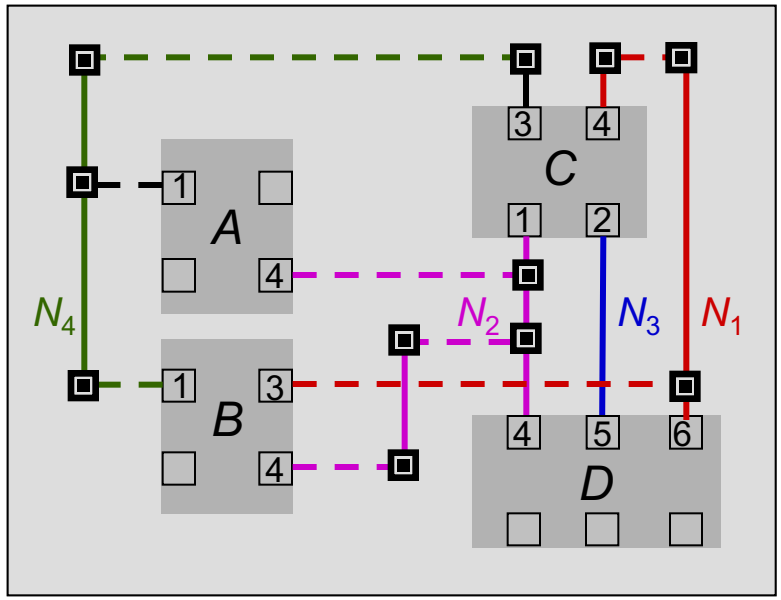
$N_1 = \{C_4, D_6, B_3\}$

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

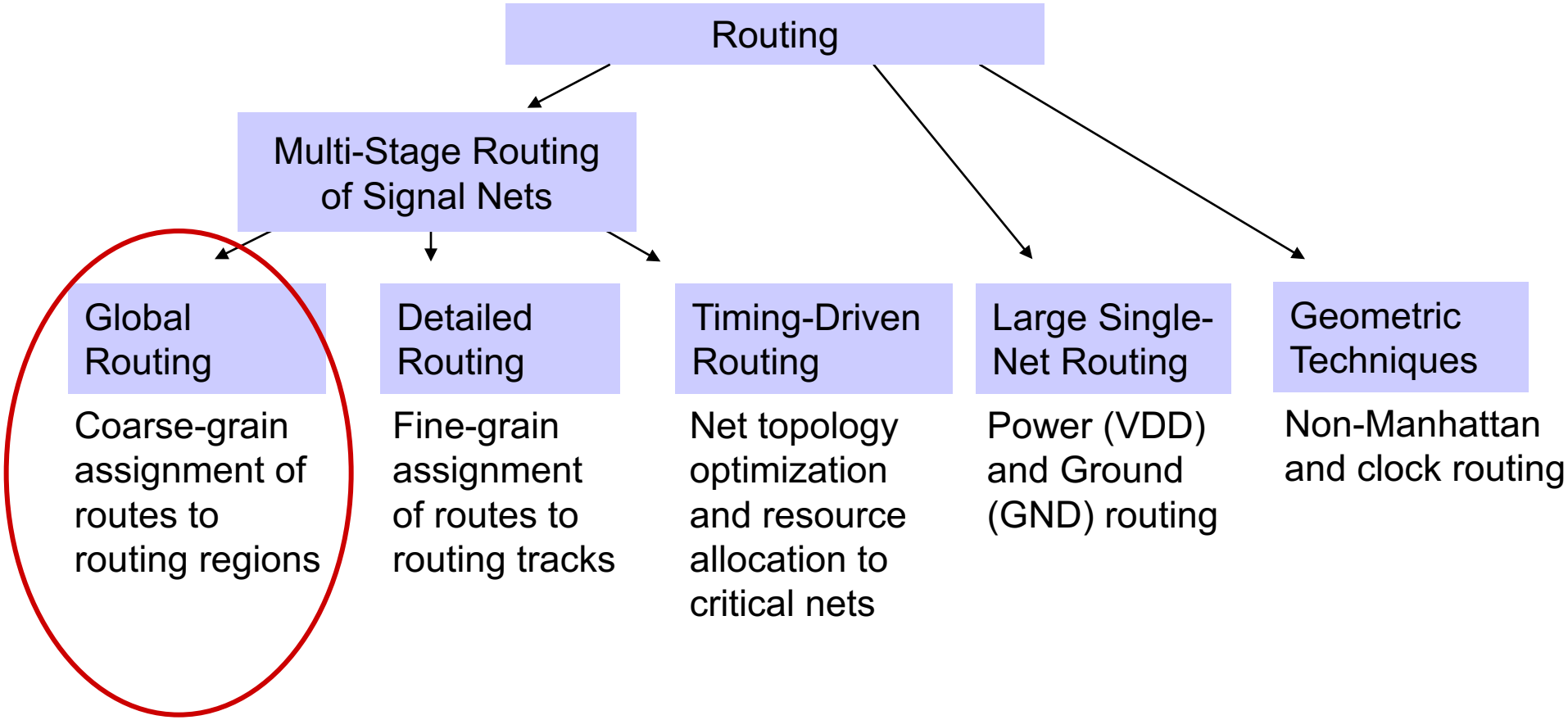
$N_3 = \{C_2, D_5\}$

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

Technology Information
(Design Rules)



# Global Routing



# Introduction

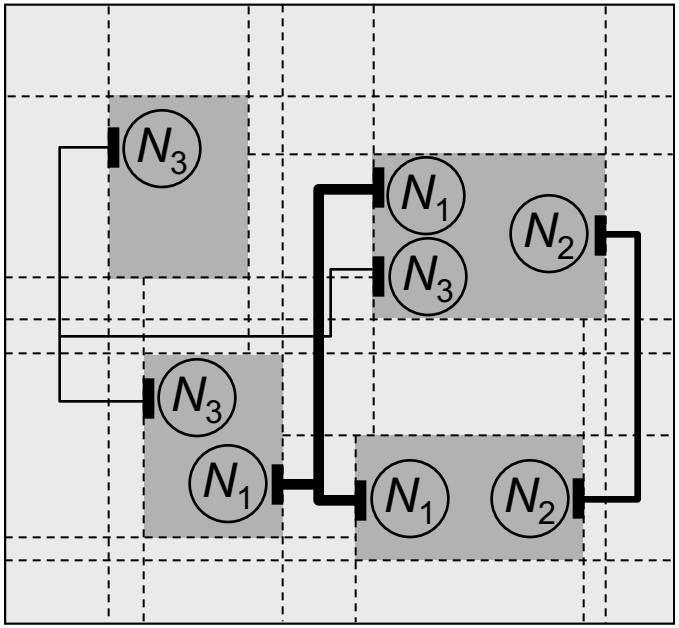
## Global Routing

- Wire segments are tentatively assigned (embedded) within the chip layout
- Chip area is represented by a coarse routing grid
- Available routing resources are represented by edges with capacities in a grid graph

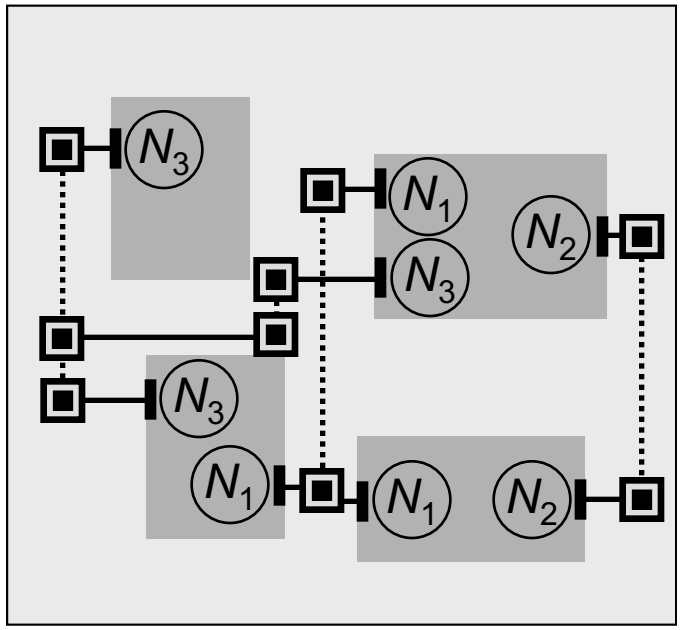
⇒ Nets are assigned to these routing resources

# Introduction

Global Routing



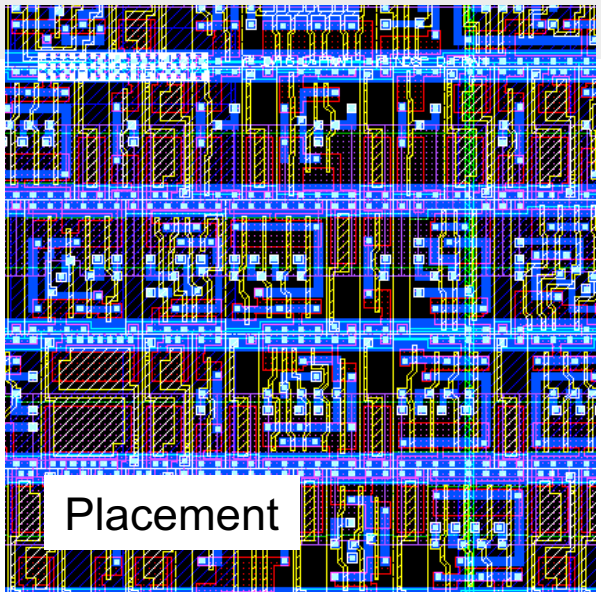

Detailed Routing

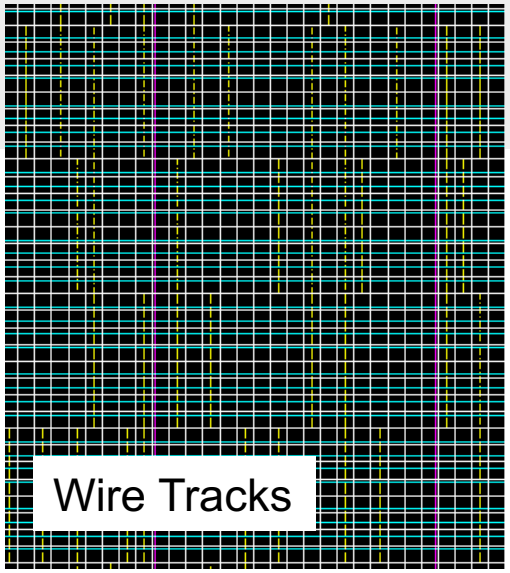


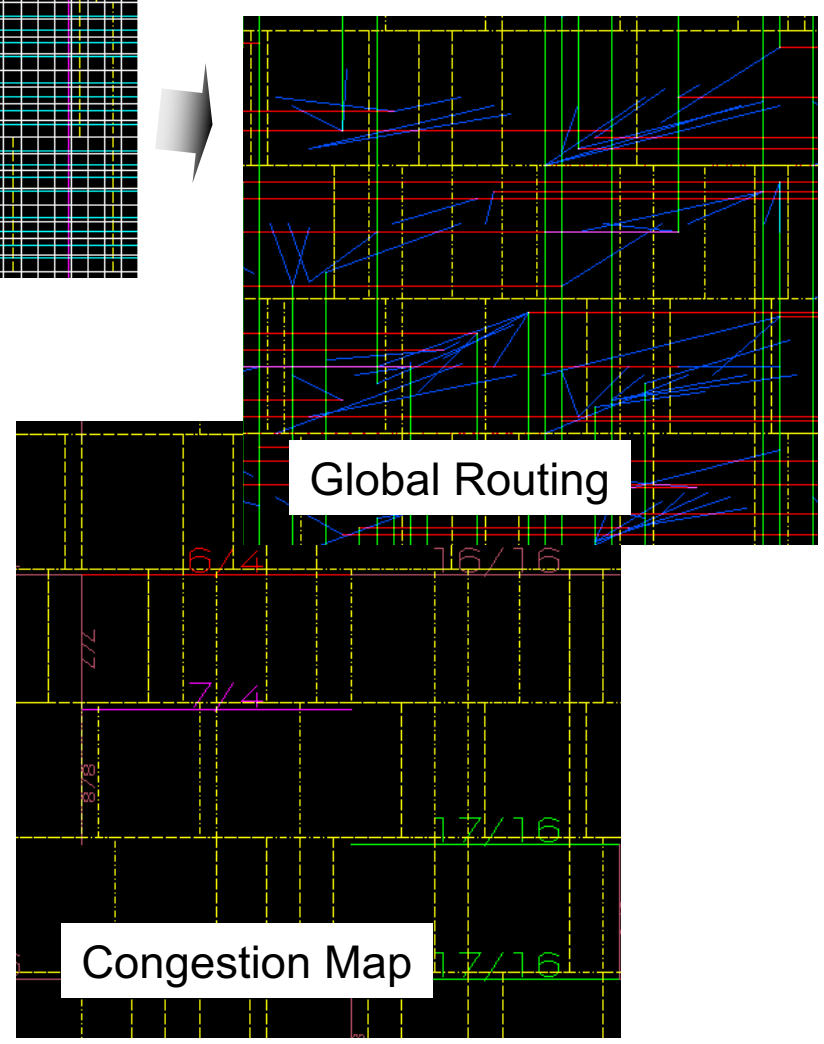

Horizontal Segment

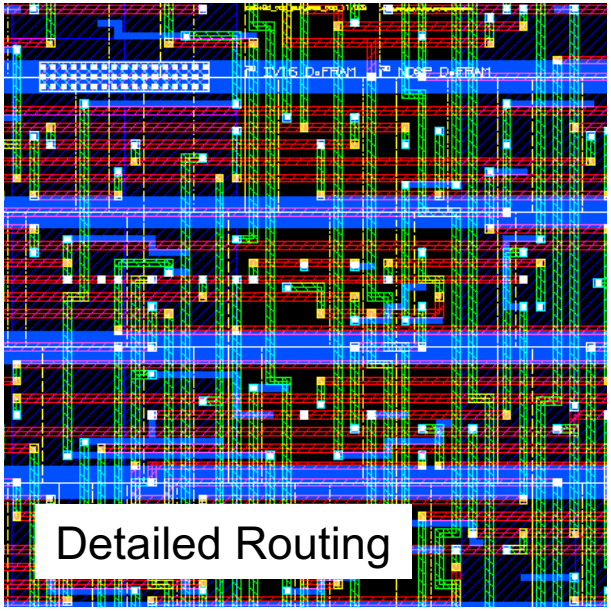
Vertical Segment

Via


Placement


Wire Tracks


Global Routing
6/4
16/16
7/7
7/4
8/8
17/16
17/16
Congestion Map


Detailed Routing

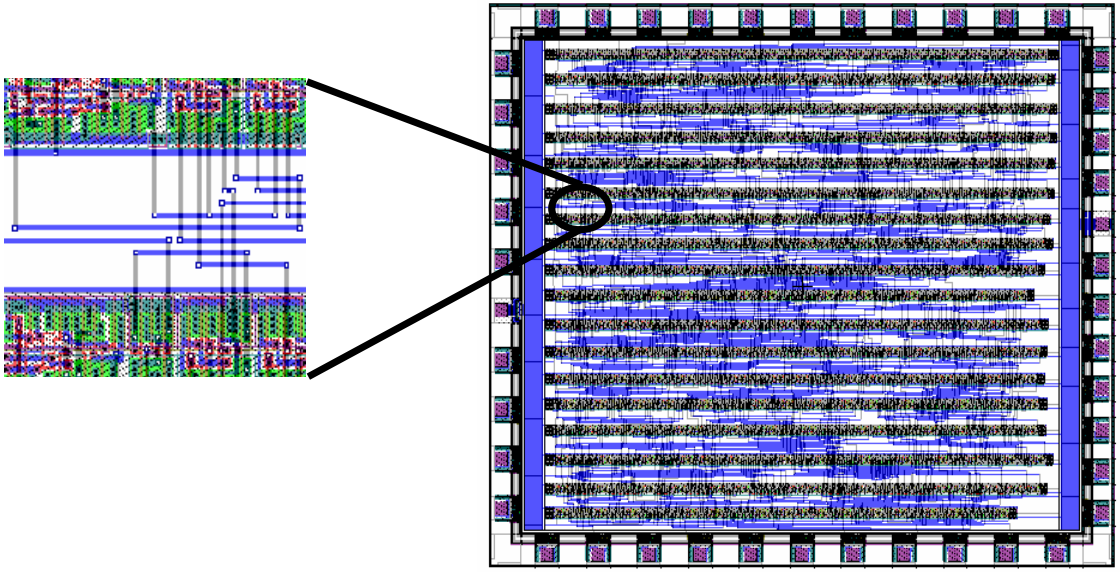
# Terminology and Definitions

- Routing Track: Horizontal wiring path
- Routing Column: Vertical wiring path
- Routing Region: Region that contains routing tracks or columns
- Uniform Routing Region: Evenly spaced horizontal/vertical grid
- Non-uniform Routing Region: Horizontal and vertical boundaries that are aligned to external pin connections or macro-cell boundaries resulting in routing regions that have differing sizes

# Terminology and Definitions

## Channel

Rectangular routing region with pins on two opposite sides

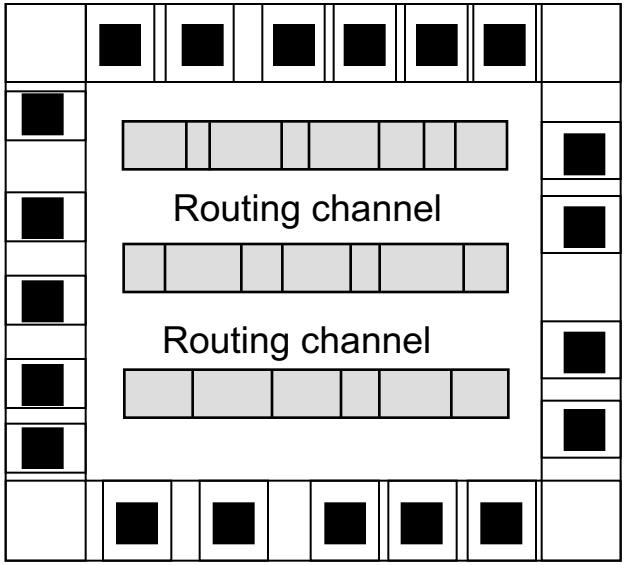


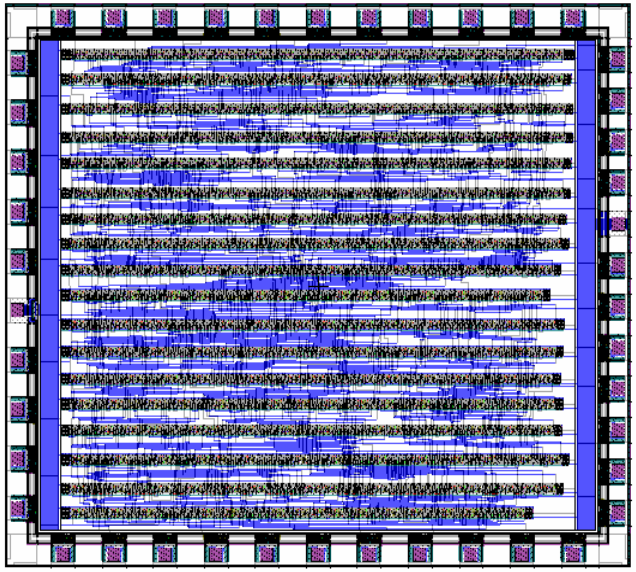
Standard cell layout (Two-layer routing)

# Terminology and Definitions

## Channel

Rectangular routing region with pins on two opposite sides



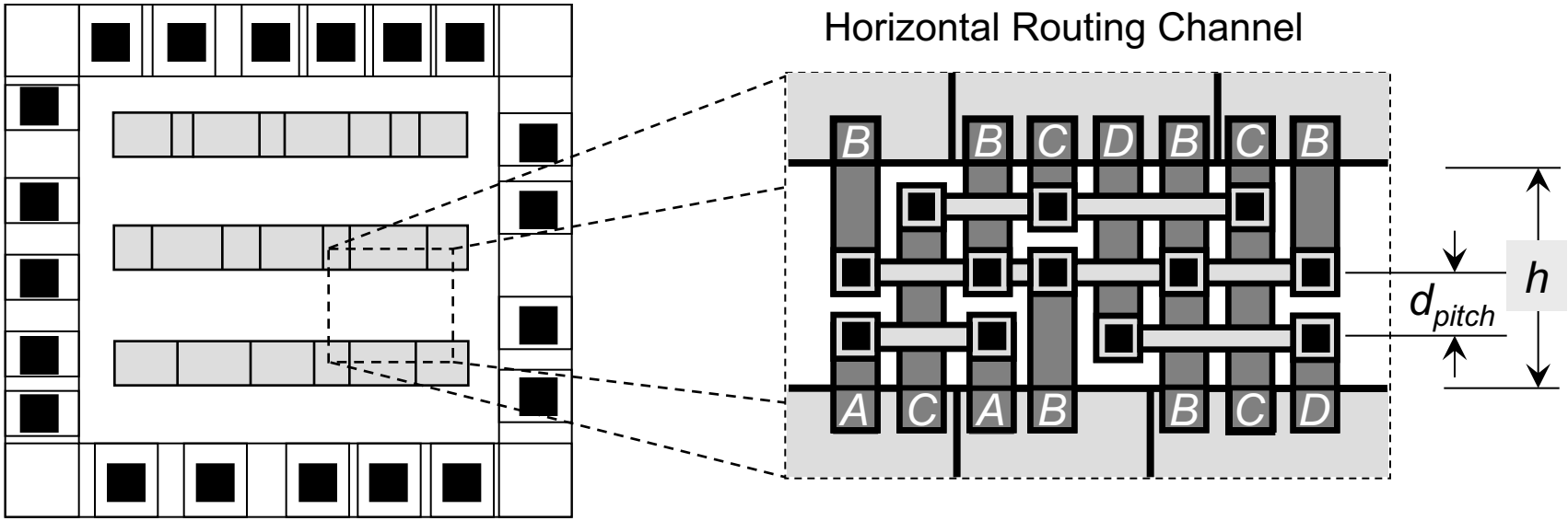



Standard cell layout (Two-layer routing)

# Terminology and Definitions

## Capacity

Number of available routing tracks or columns



# Terminology and Definitions

## Capacity

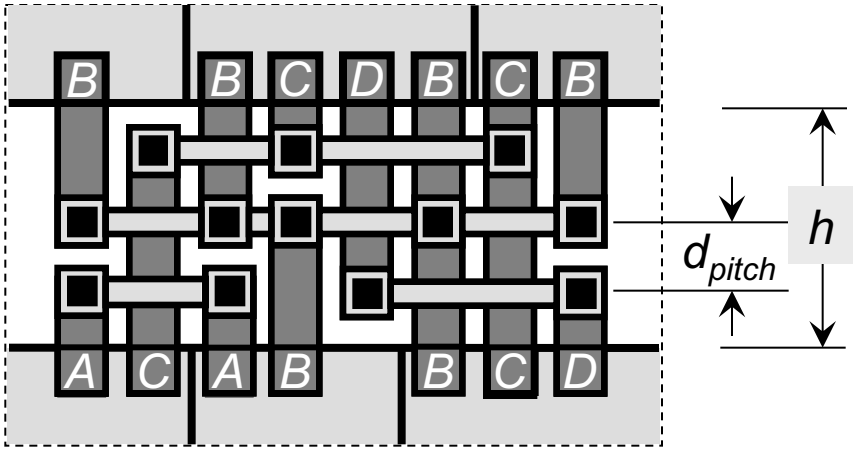
Number of available routing tracks or columns

- For single-layer routing, the capacity is the height *h* of the channel divided by the pitch $d_{pitch}$

- For multilayer routing, the capacity σ is the sum of the capacities of all layers.

$$\sigma(Layers) = \sum_{layer \in Layers} \left\lfloor \frac{h}{d_{pitch}(layer)} \right\rfloor$$

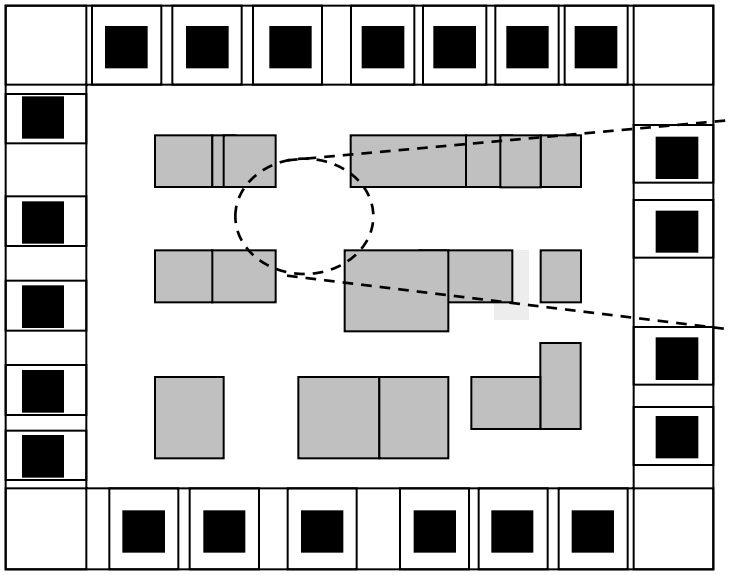
Horizontal Routing Channel

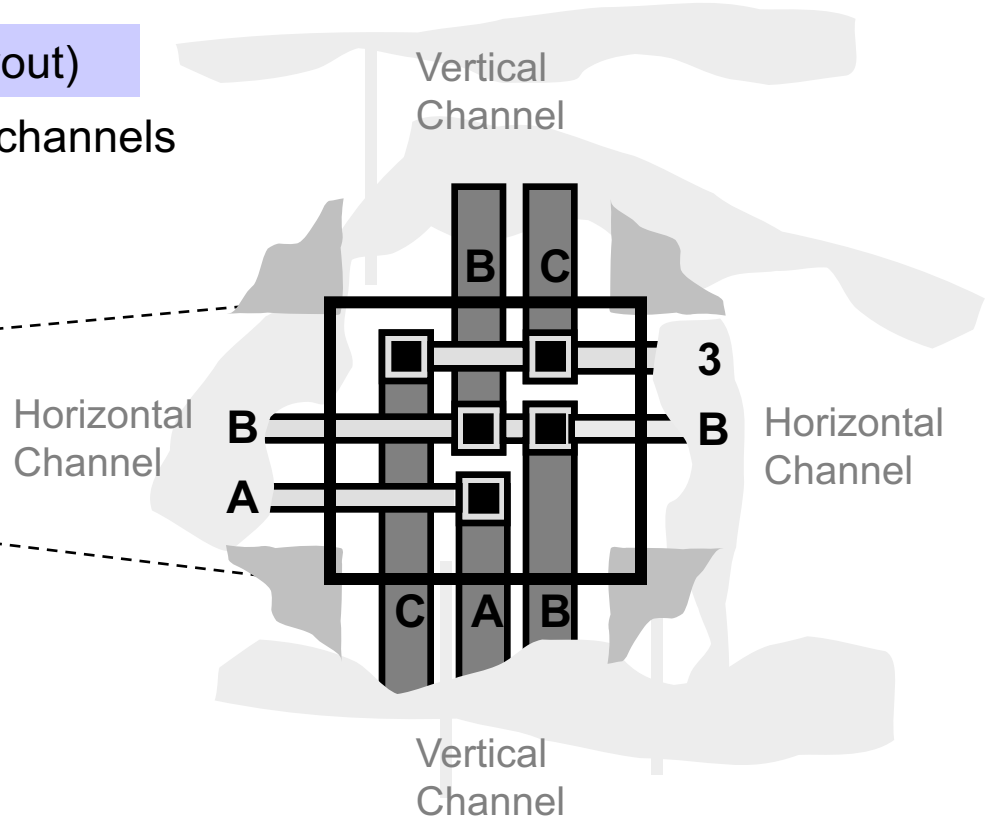


# Terminology and Definitions

Switchbox (Two-layer macro cell layout)

Intersection of horizontal and vertical channels

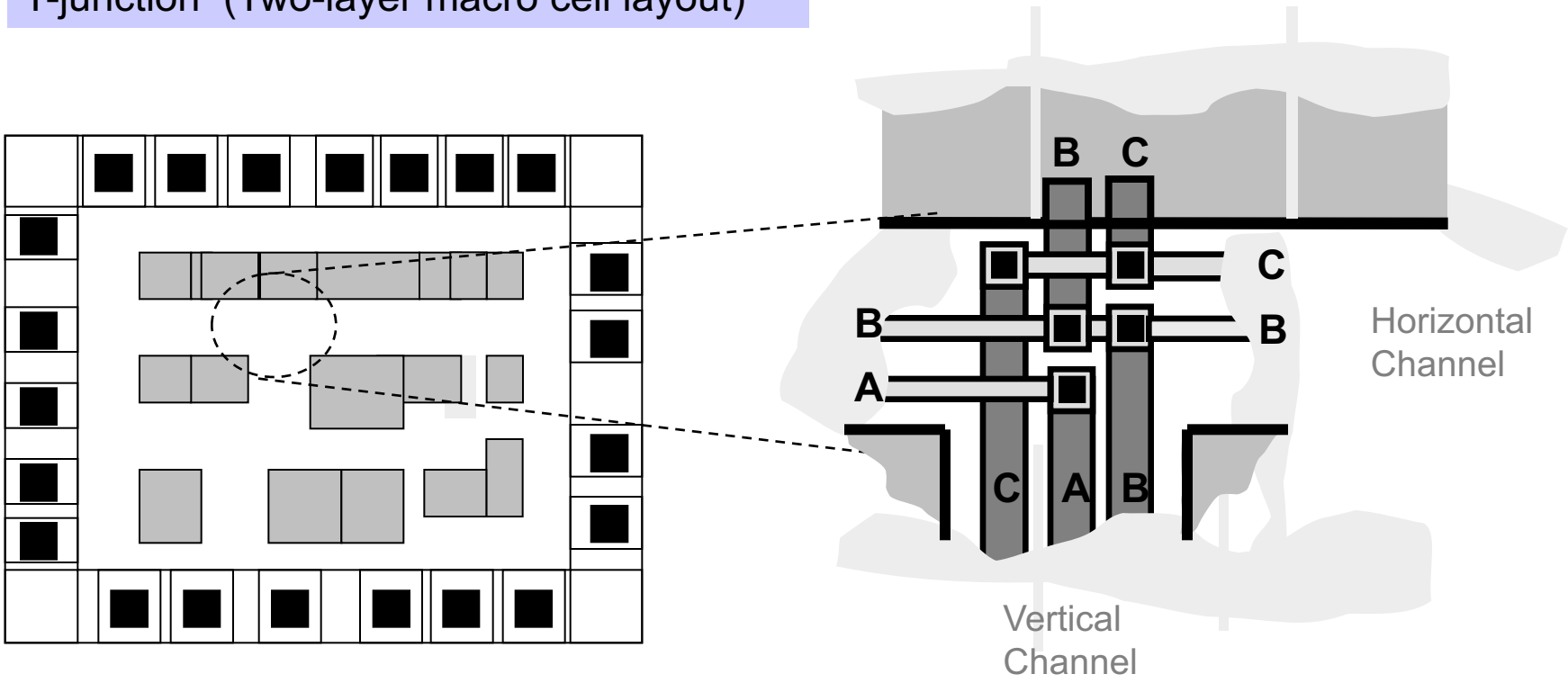





Horizontal channel is routed after vertical channel is routed

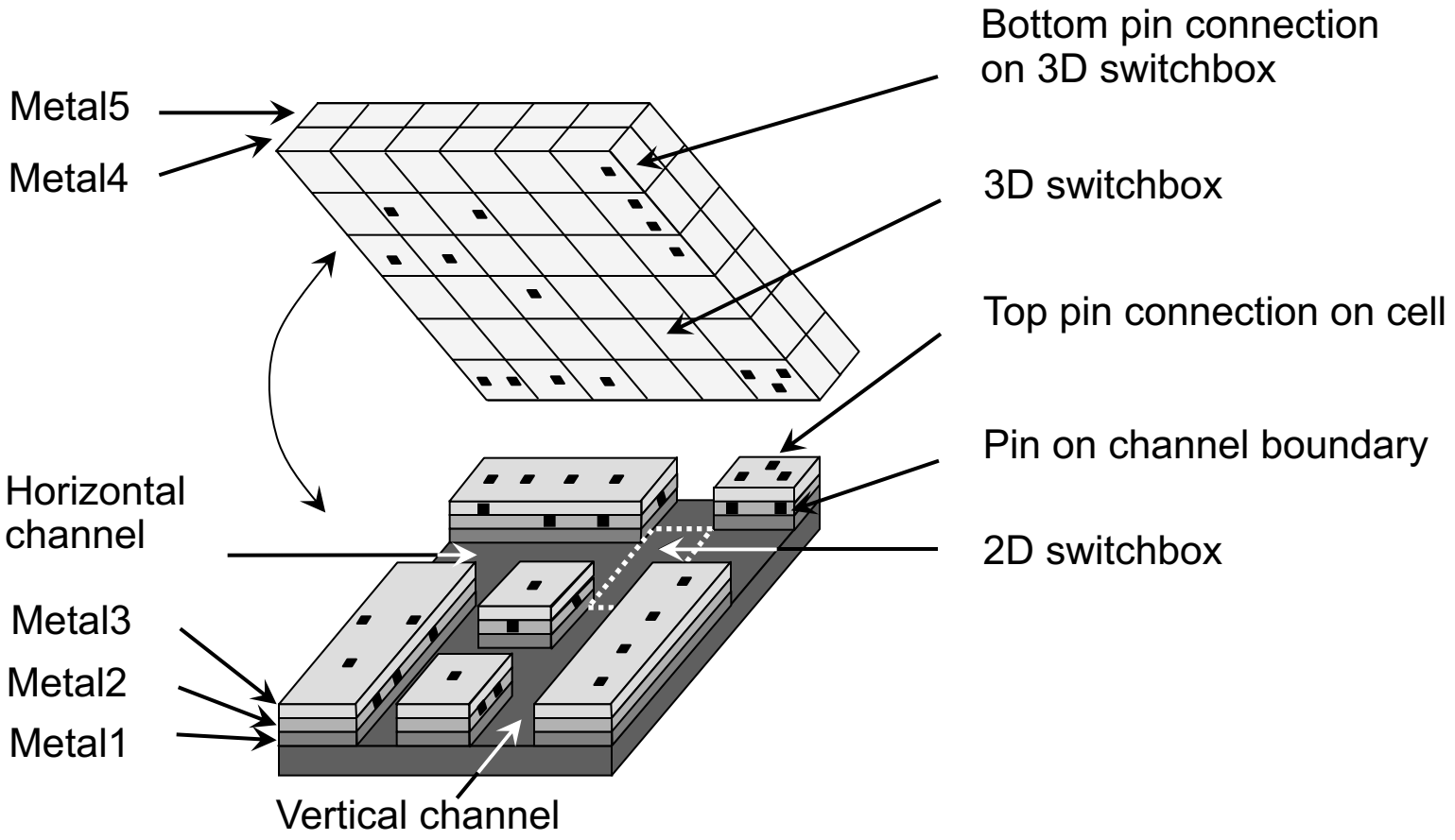
# Terminology and Definitions

T-junction (Two-layer macro cell layout)



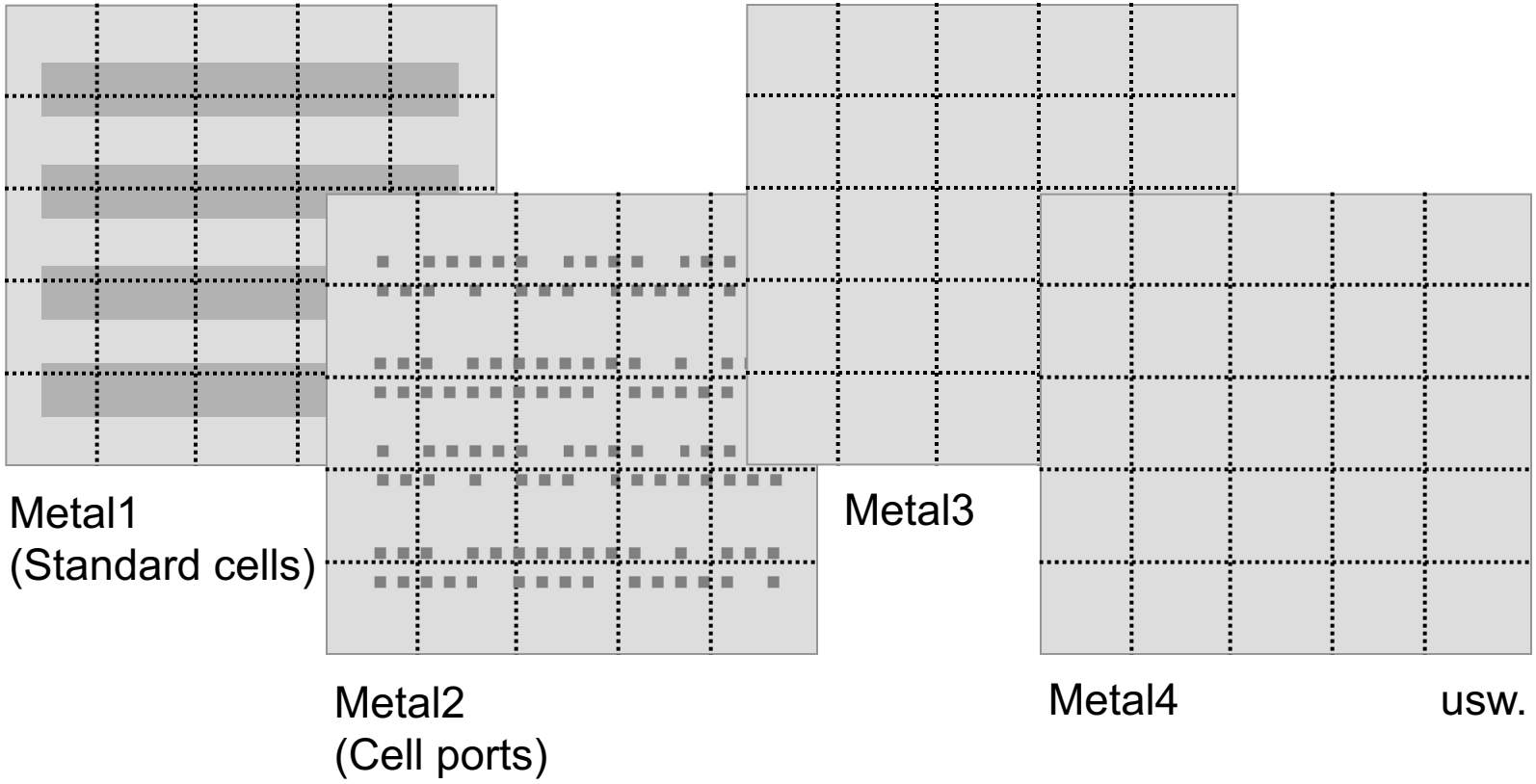
# Terminology and Definitions

## 2D and 3D Switchboxes



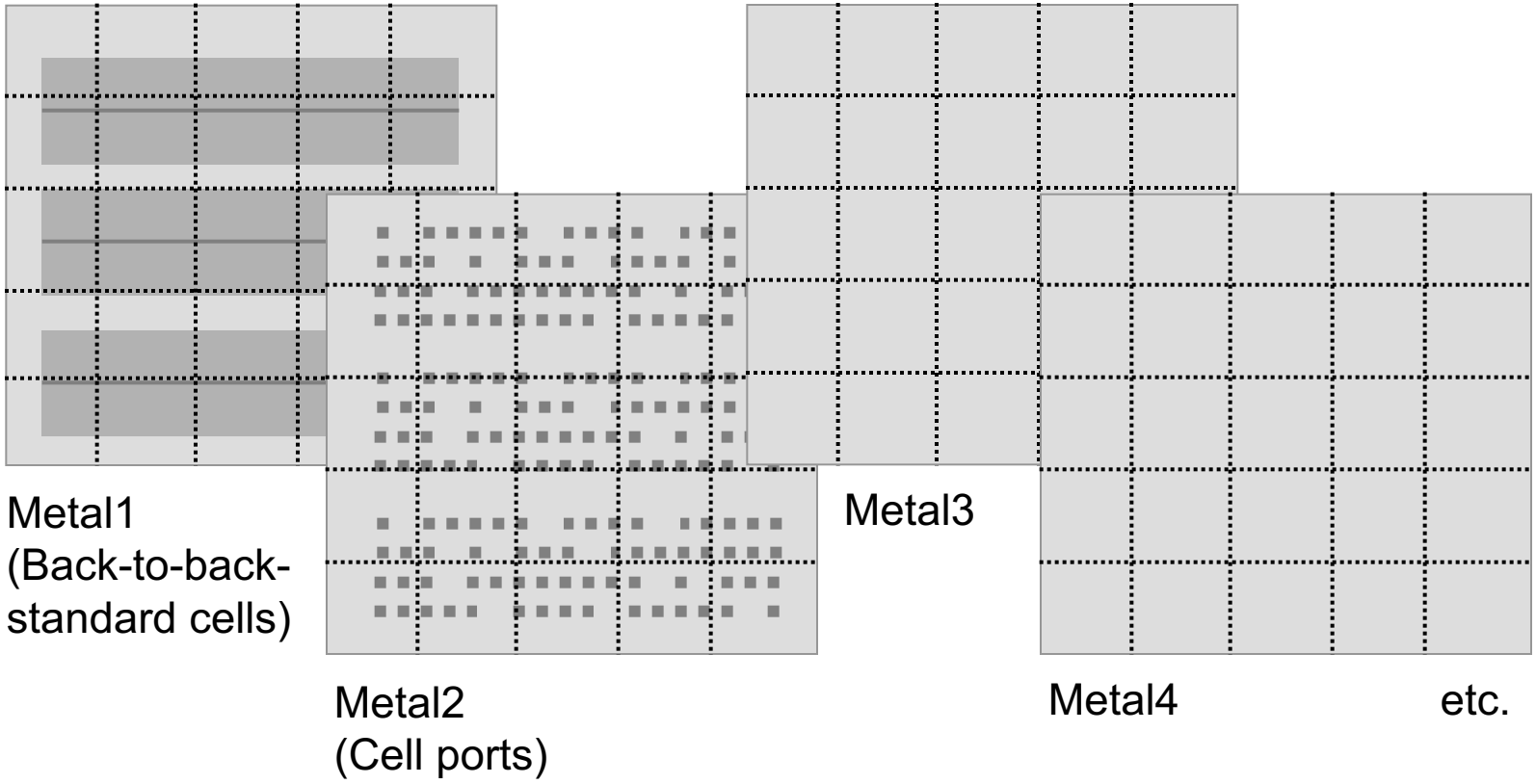
# Terminology and Definitions

Gcells (Tiles) with standard cells



# Terminology and Definitions

Gcells (Tiles) with standard cells (back-to-back)



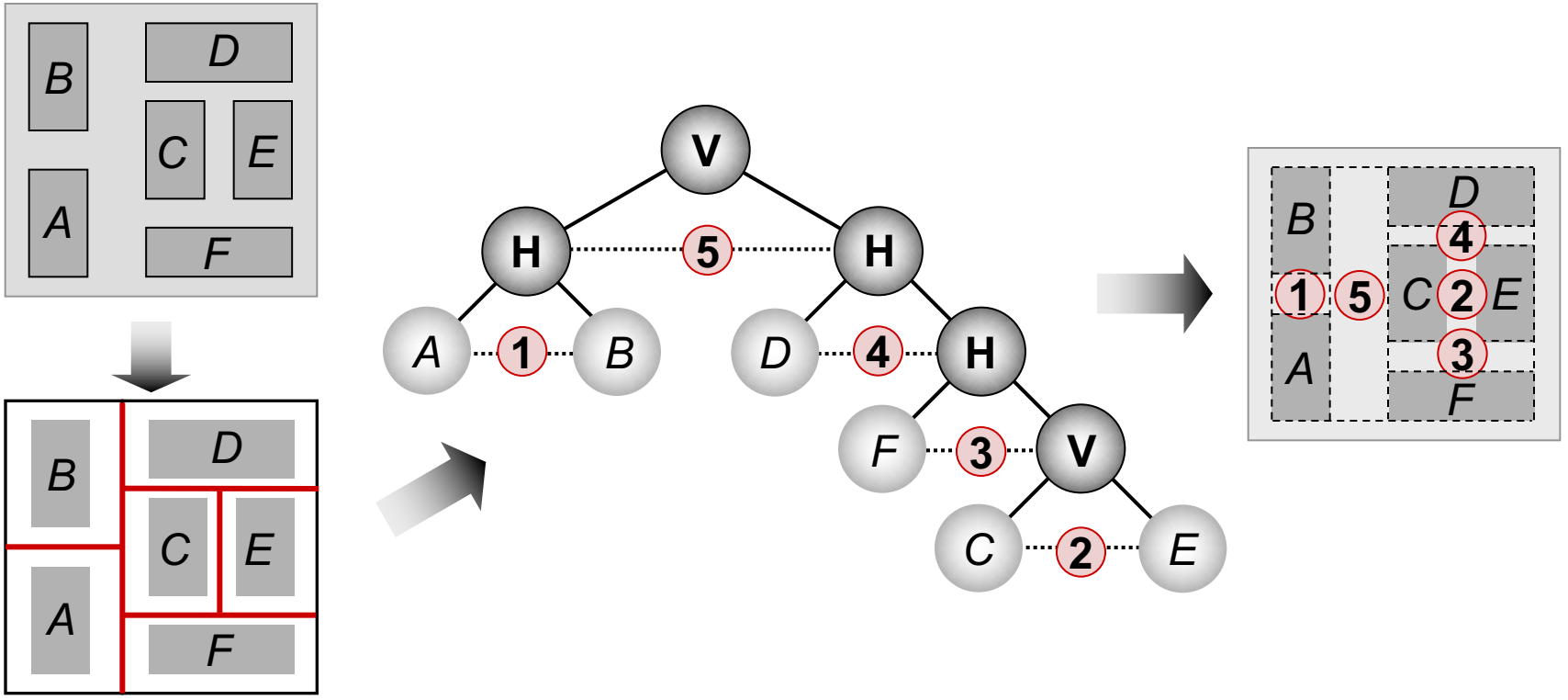
# Optimization Goals

- Global routing seeks to
  - determine whether a given placement is routable, and
  - determine a coarse routing for all nets within available routing regions
- Considers goals such as
  - minimizing total wirelength, and
  - reducing signal delays on critical nets

# Optimization Goals

Full-custom design

Layout is dominated by macro cells and routing regions are non-uniform



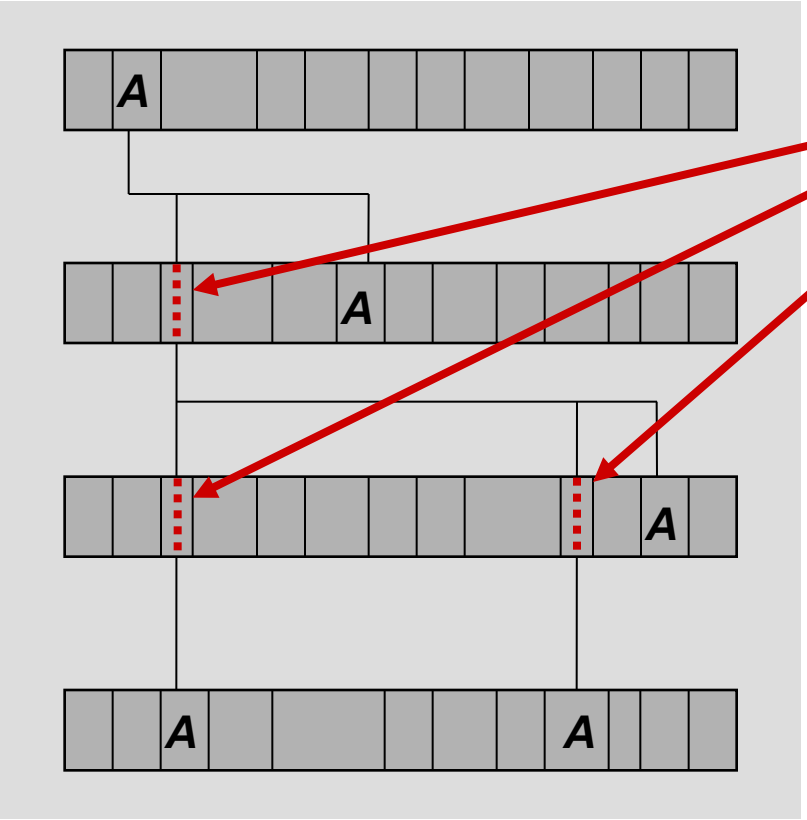

(1) Types of channels

(2) Channel ordering

# Optimization Goals

Standard-cell design

If number of metal layers is limited, feedthrough cells must be used to route across multiple cell rows



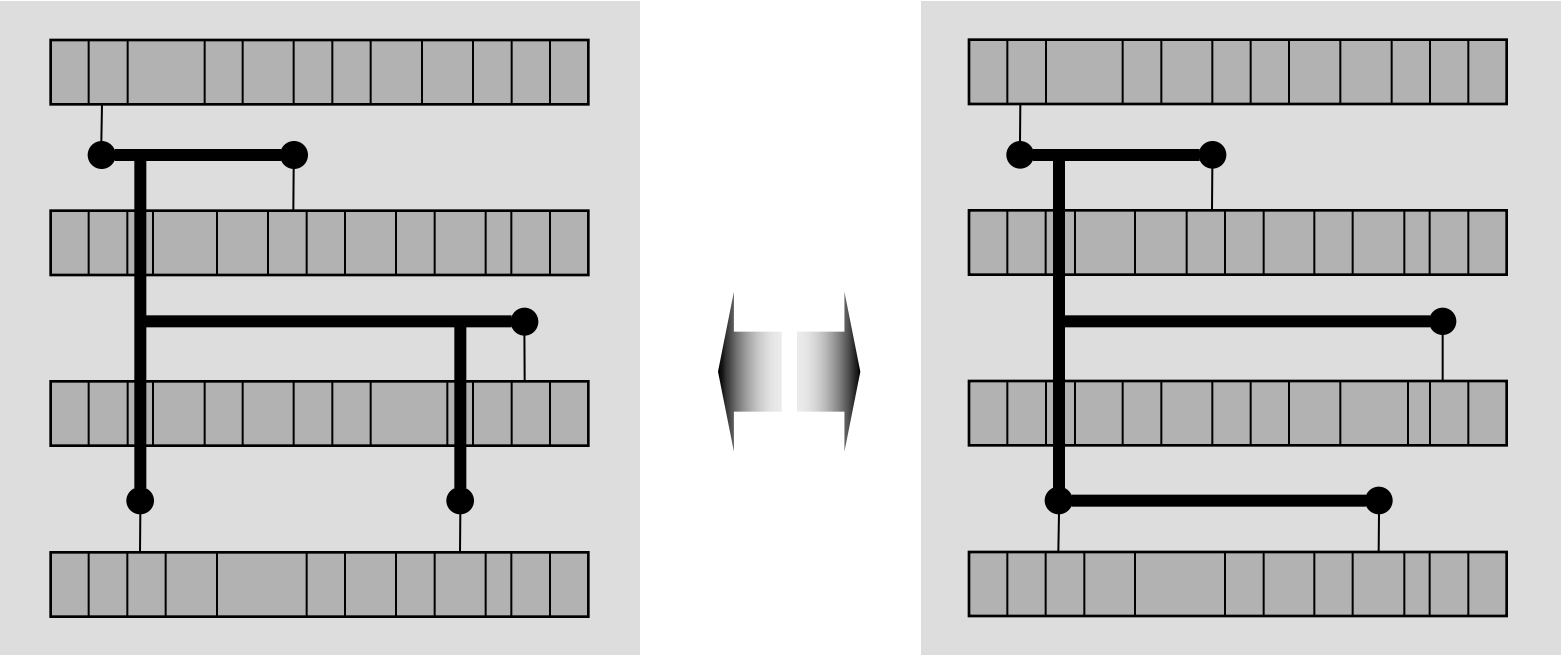

Feedthrough cells

Variable-die, standard cell design:

Total height =
ΣCell row heights +
All channel heights

# Optimization Goals

Standard-cell design



Steiner tree solution with minimal wirelength

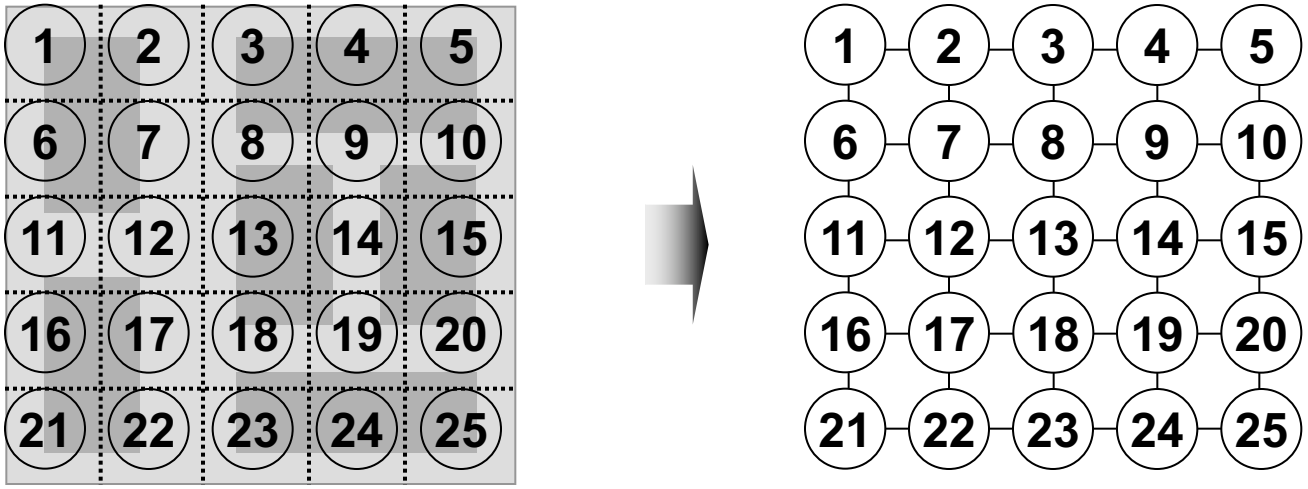
Steiner tree solution with fewest feedthrough cells

# Representations of Routing Regions

- Routing regions are represented using efficient data structures
- Routing context is captured using a graph, where
  - nodes represent routing regions and
  - edges represent adjoining regions
- Capacities are associated with both edges and nodes to represent available routing resources

# Representations of Routing Regions

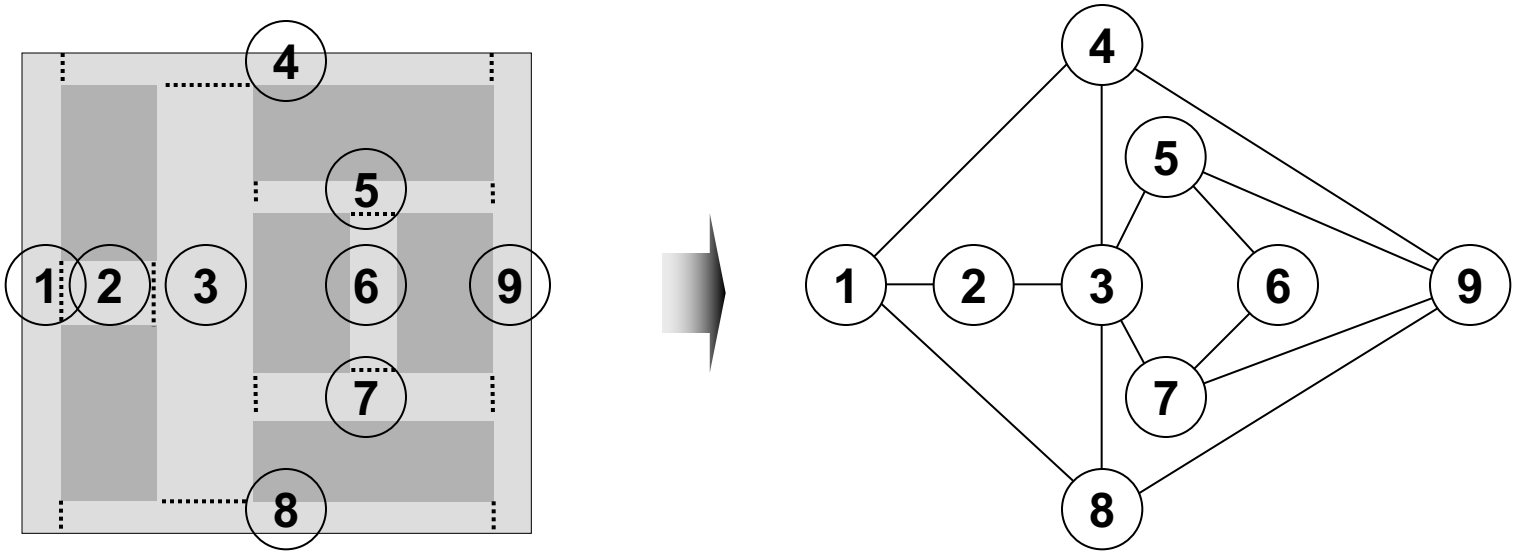
Grid graph model




*ggrid* = (*V*,*E*), where the nodes $v \in V$ represent the routing grid cells (*gcells*) and the edges represent connections of grid cell pairs ($v_i$,$v_j$)

# Representations of Routing Regions

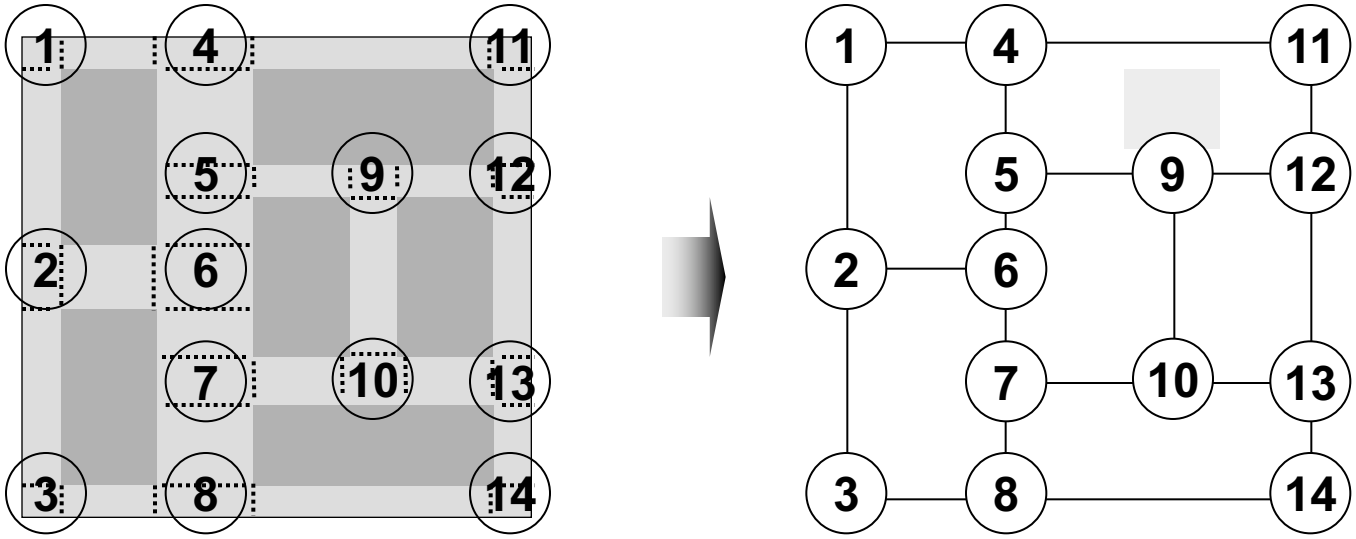
Channel connectivity graph




$G = (V,E)$, where the nodes $v \in V$ represent channels,
and the edges $E$ represent adjacencies of the channels

# Representations of Routing Regions

Switchbox connectivity graph

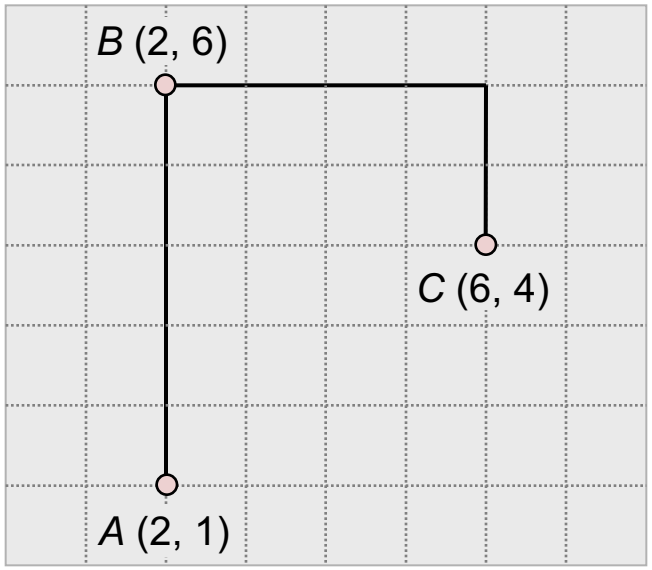


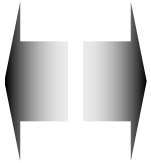

$G = (V, E)$, where the nodes $v \in V$ represent switchboxes
and an edge exists between two nodes if the corresponding switchboxes are on opposite sides of the same channel

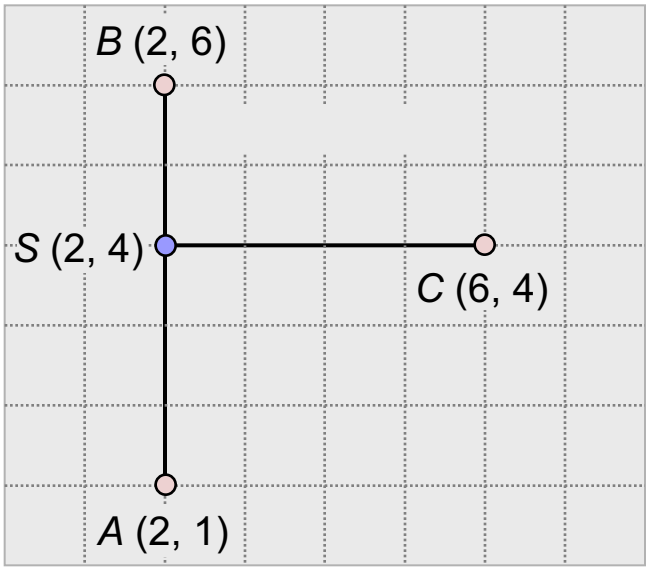
# The Global Routing Flow – General Idea

1. Defining the routing regions (Region definition)
   - Layout area is divided into routing regions
   - Nets can also be routed over standard cells
   - Regions, capacities and connections are represented by a graph
2. Mapping nets to the routing regions (Region assignment)
   - Each net of the design is assigned to one or several routing regions to connect all of its pins
   - Routing capacity, timing and congestion affect mapping

# Rectilinear Routing






Rectilinear minimum spanning tree (RMST)

Rectilinear Steiner minimum tree (RSMT)

# Rectilinear Routing

- An RMST can be computed in $O(p^2)$ time, where $p$ is the number of terminals in the net using methods such as Prim's Algorithm
- Prim's Algorithm builds an MST by starting with a single terminal and greedily adding least-cost edges to the partially-constructed tree
- Advanced computational-geometric techniques reduce the runtime to $O(p \log p)$

# Rectilinear Routing

**Characteristics of an RSMT**

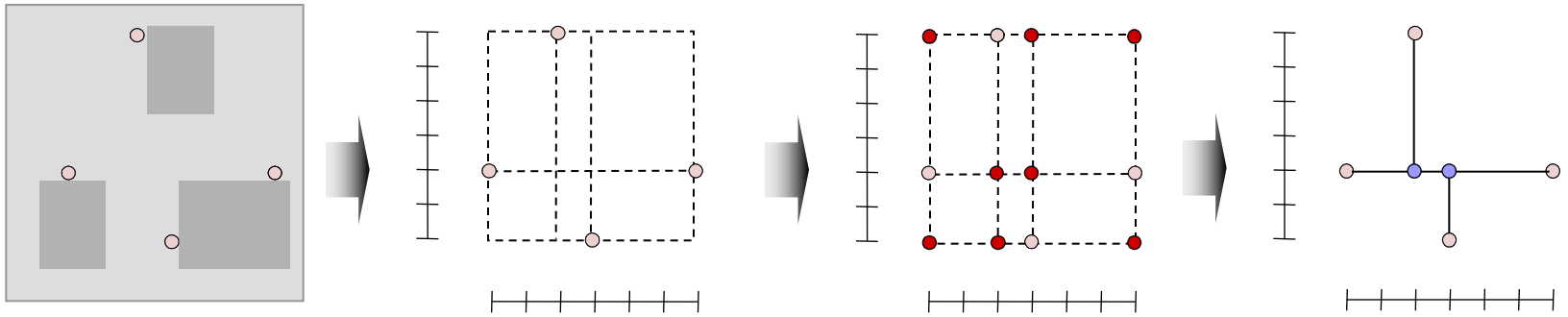
- An RSMT for a *p*-pin net has between 0 and *p* – 2 (inclusive) Steiner points
- The degree of any terminal pin is 1, 2, 3, or 4
- The degree of a Steiner point is either 3 or 4
- A RSMT is always enclosed in the minimum bounding box (MBB) of the net
- The total edge length $L_{RSMT}$ of the RSMT is at least half the perimeter of the minimum bounding box of the net: $L_{RSMT} \geq L_{MBB} / 2$

# Rectilinear Routing

## Hanan grid

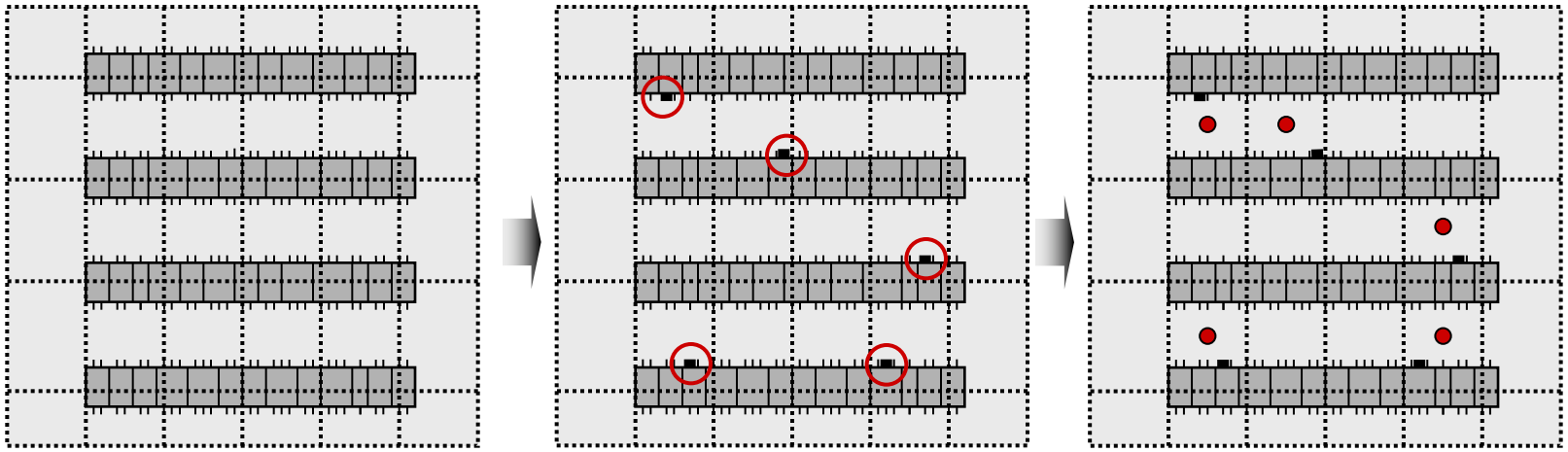
- Adding Steiner points to an RMST can significantly reduce the wirelength
- Maurice Hanan proved that for finding Steiner points, it suffices to consider only points located at the intersections of vertical and horizontal lines that pass through terminal pins
- The Hanan grid consists of the lines $x = x_p$, $y = y_p$ that pass through the location $(x_p,y_p)$ of each terminal pin $p$
- The Hanan grid contains at most $(n^2-n)$ candidate Steiner points ($n$ = number of pins), thereby greatly reducing the solution space for finding an RSMT

# Rectilinear Routing



Terminal pins  Intersection lines  Hanan points (•)  RSMT

# Rectilinear Routing

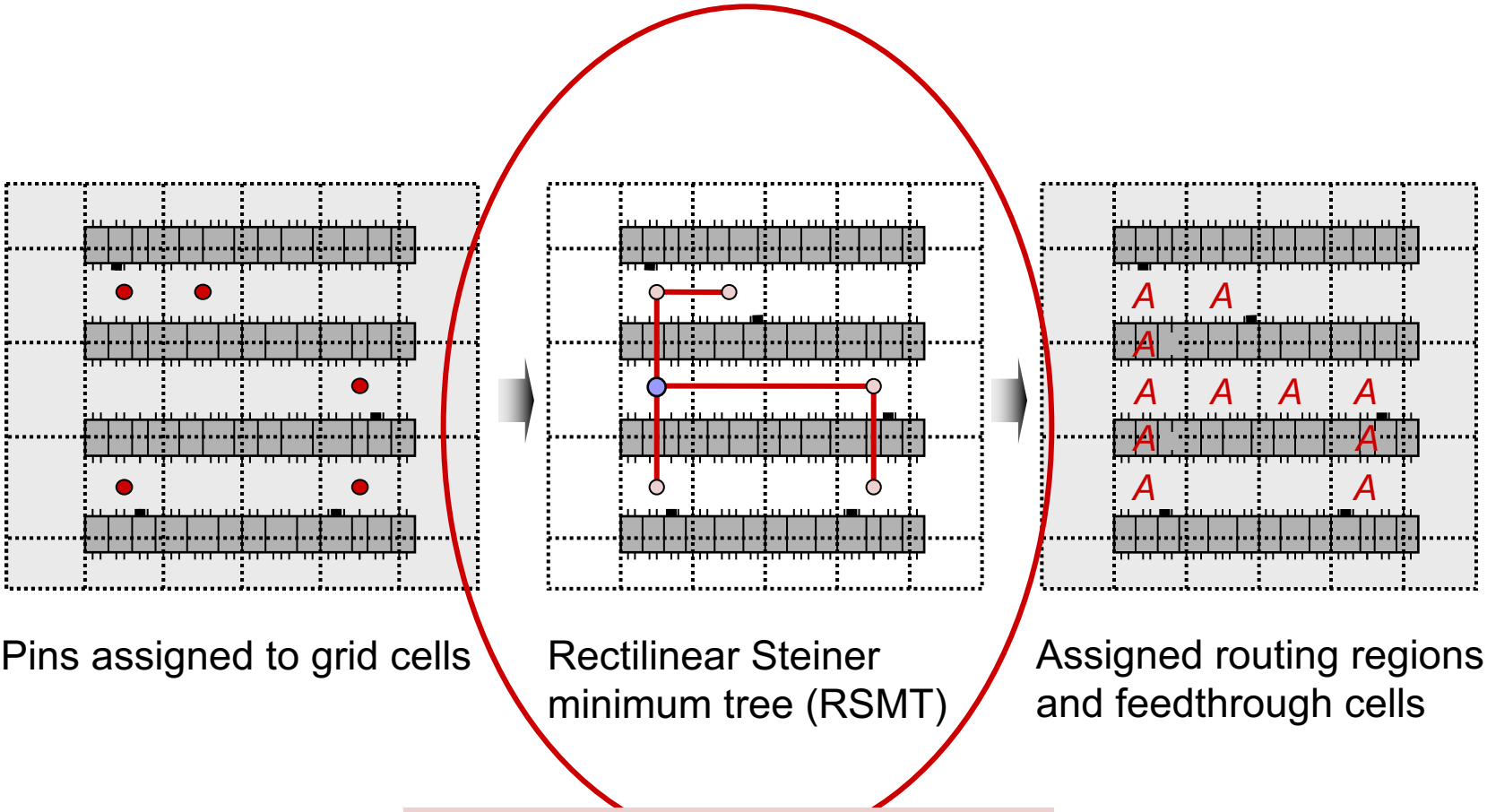


Definining routing regions

Pin connections

Pins assigned to grid cells

# Rectilinear Routing




Pins assigned to grid cells

Rectilinear Steiner minimum tree (RSMT)

Assigned routing regions and feedthrough cells

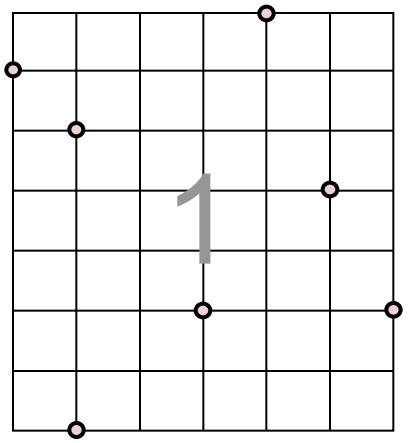
Sequential Steiner Tree Heuristic

# Rectilinear Routing

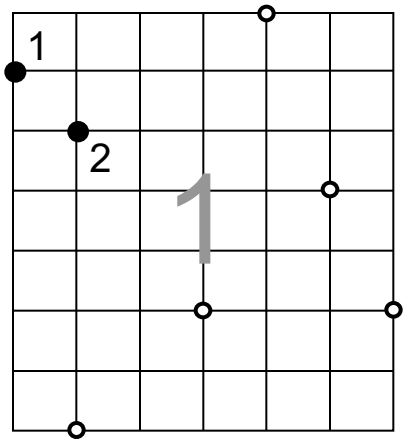
**A Sequential Steiner Tree Heuristic**

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

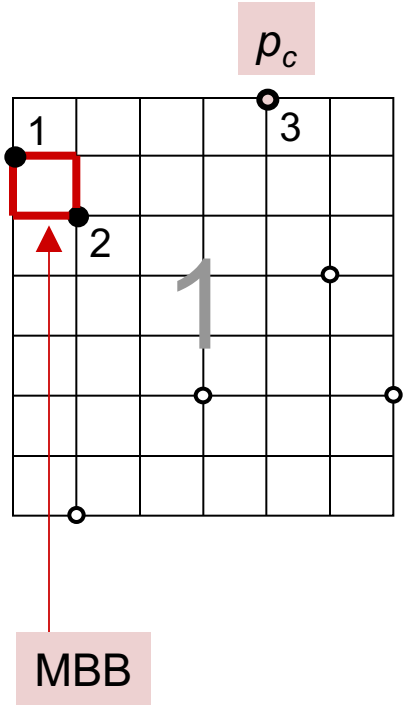
# Rectilinear Routing: Example Sequential Steiner Tree Heuristic



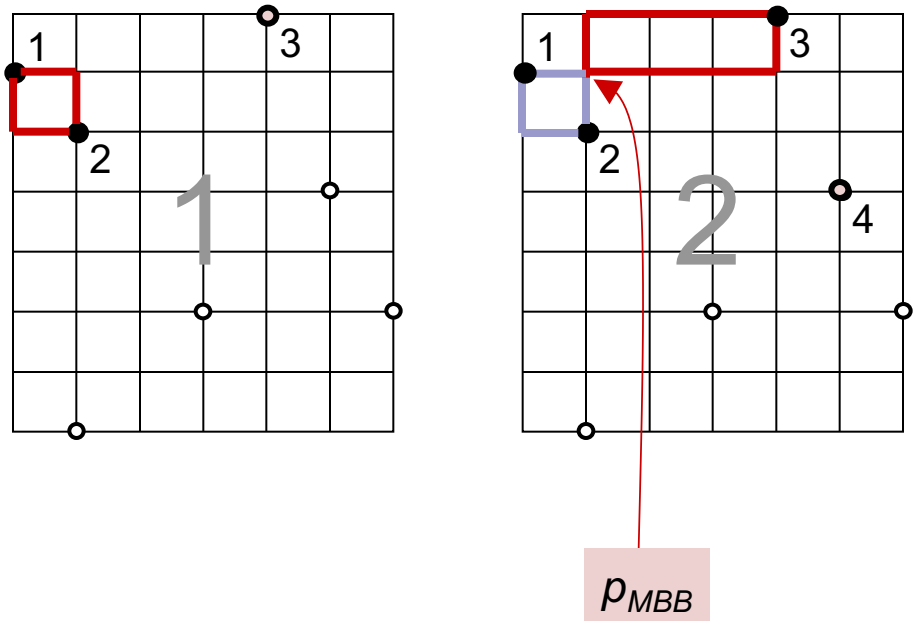
# Rectilinear Routing: Example Sequential Steiner Tree Heuristic



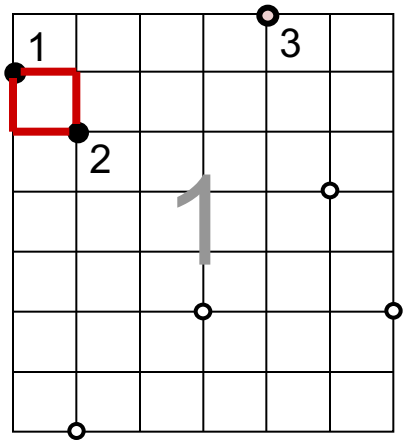
# Rectilinear Routing: Example Sequential Steiner Tree Heuristic

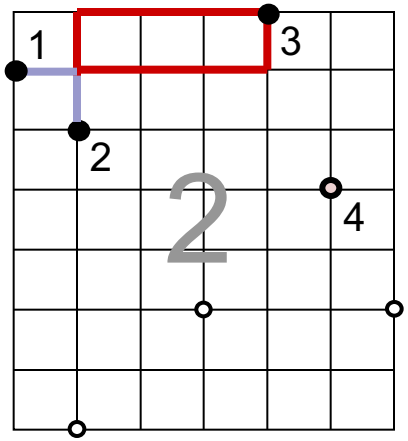


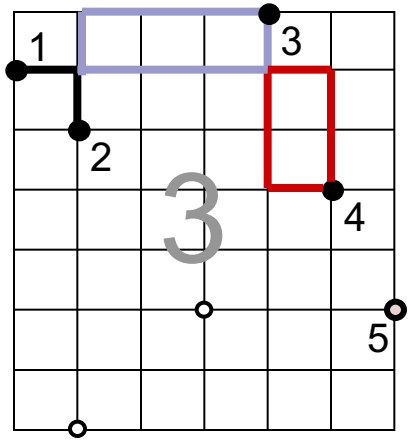
# Rectilinear Routing: Example Sequential Steiner Tree Heuristic



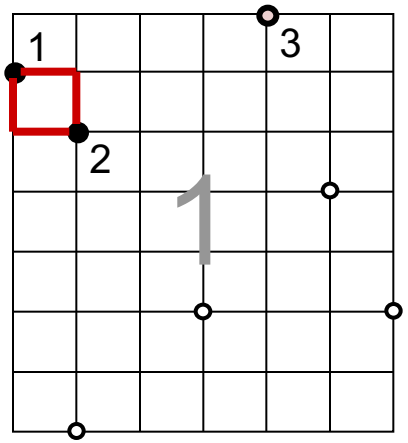
# Rectilinear Routing: Example Sequential Steiner Tree Heuristic

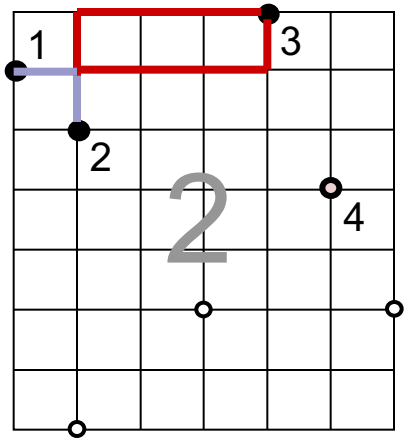


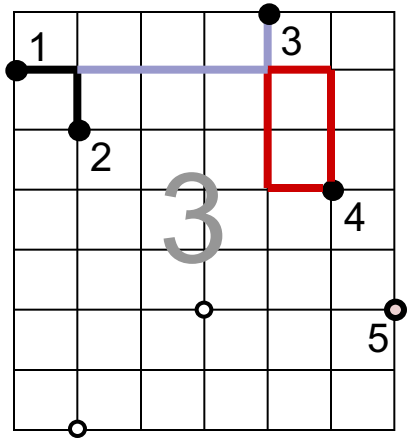



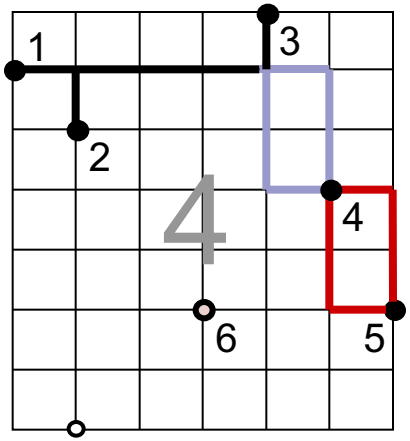



# Rectilinear Routing: Example Sequential Steiner Tree Heuristic

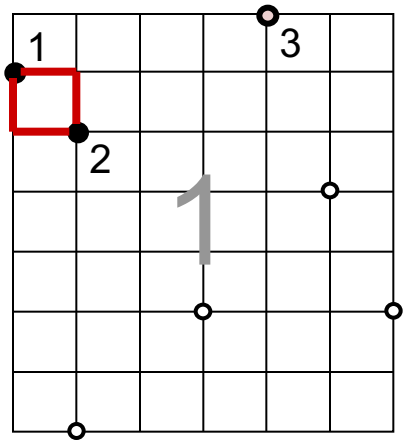


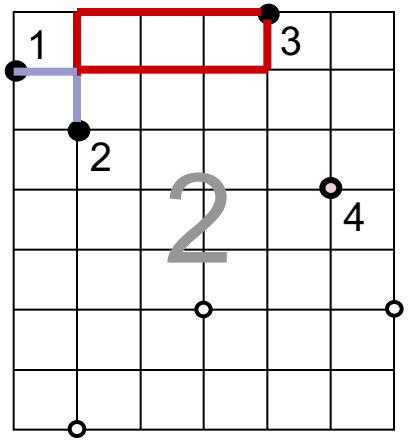



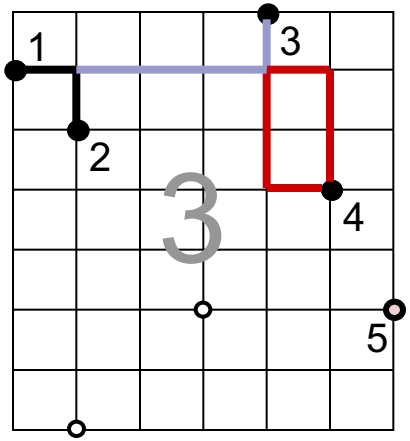



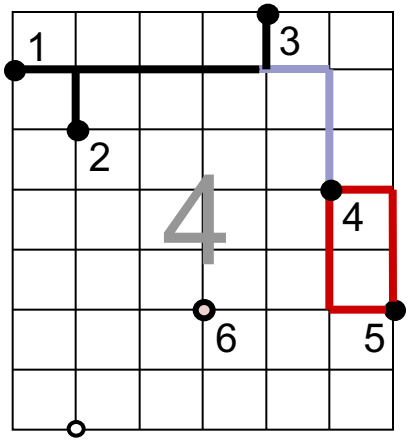



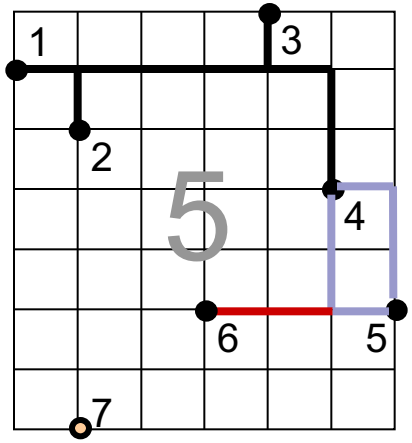
# Rectilinear Routing: Example Sequential Steiner Tree Heuristic



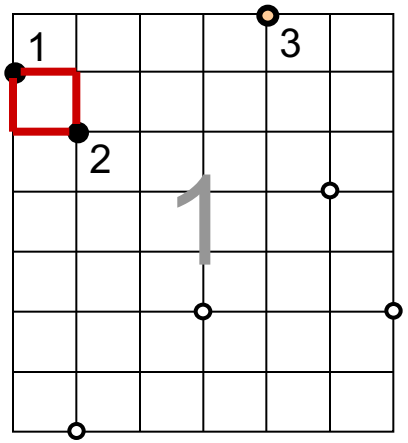



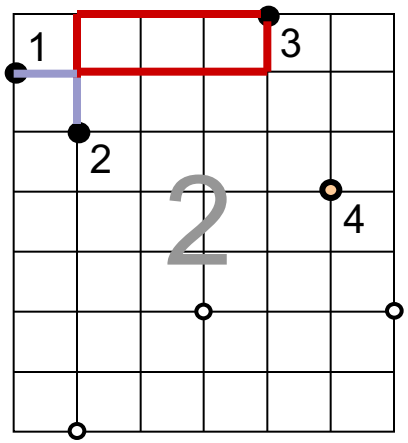



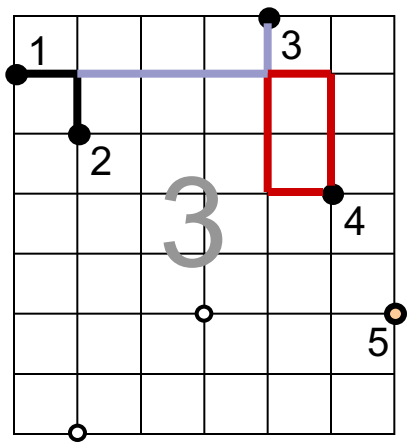



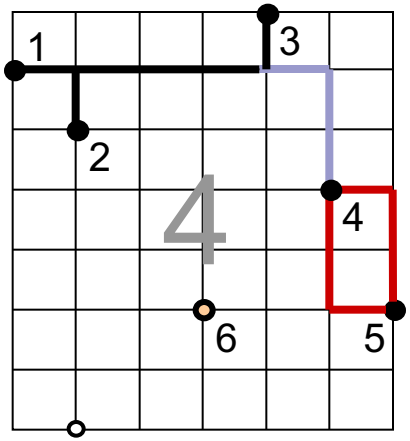



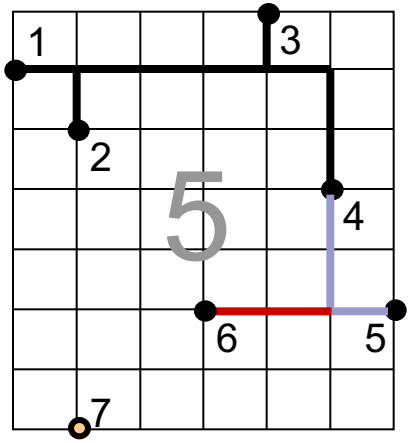
# Rectilinear Routing: Example Sequential Steiner Tree Heuristic

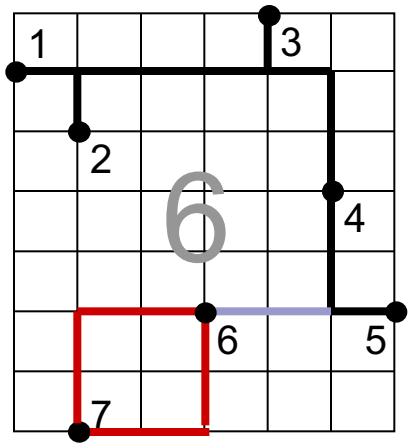


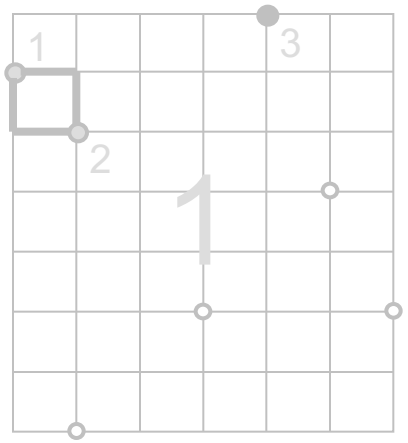



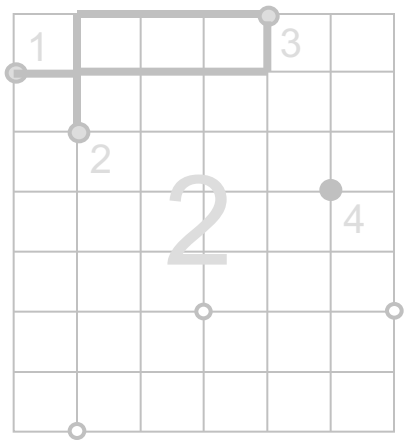



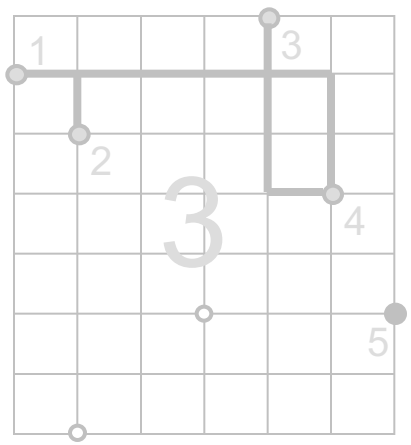



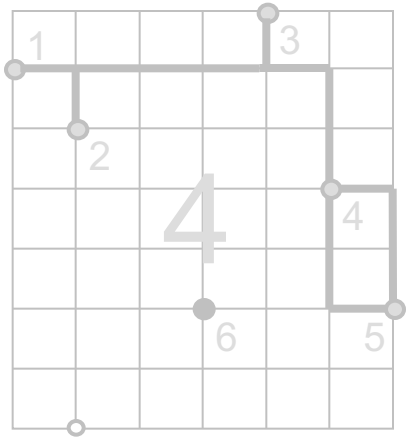



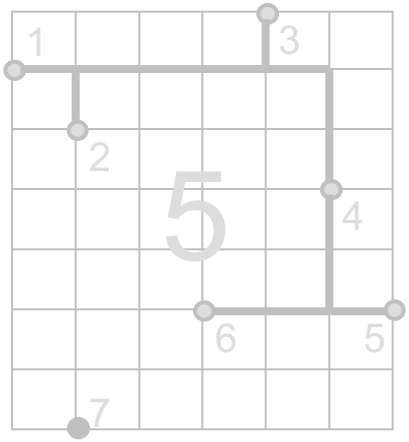
# Rectilinear Routing: Example Sequential Steiner Tree Heuristic

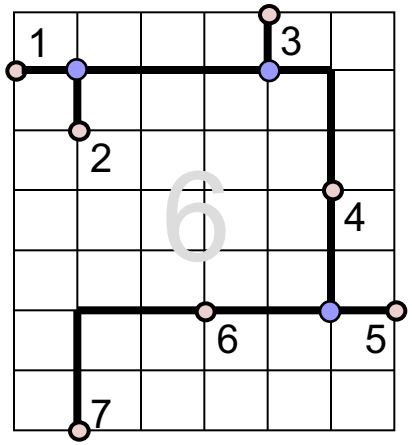


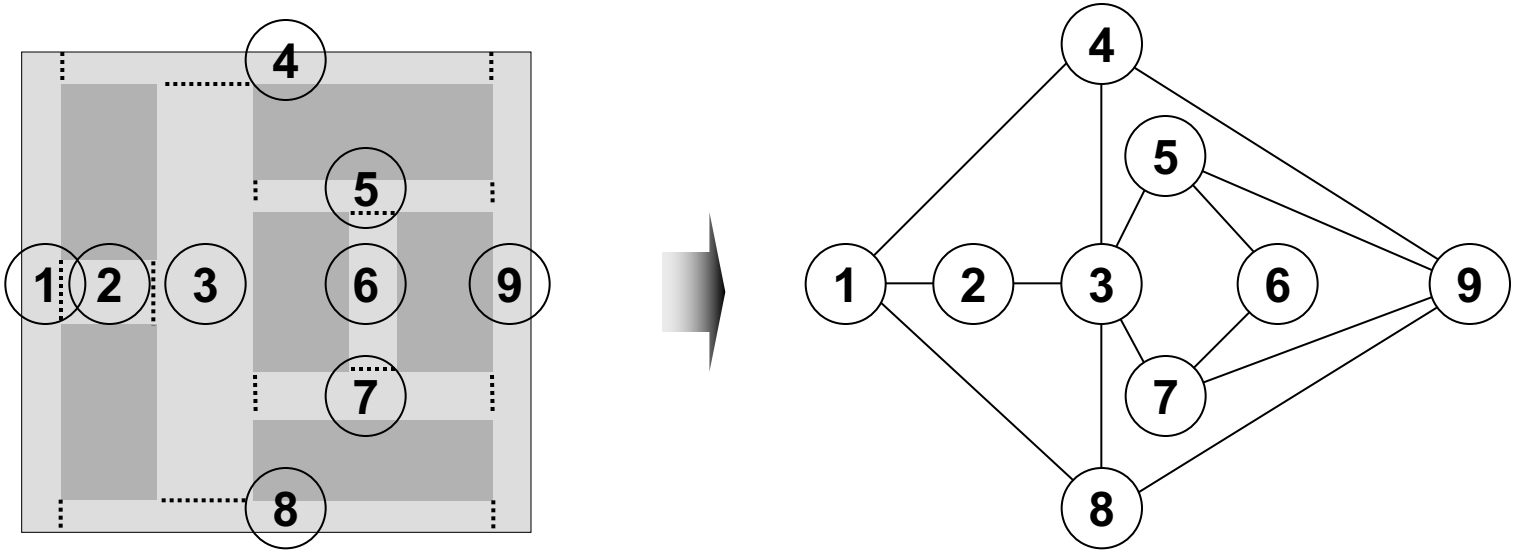



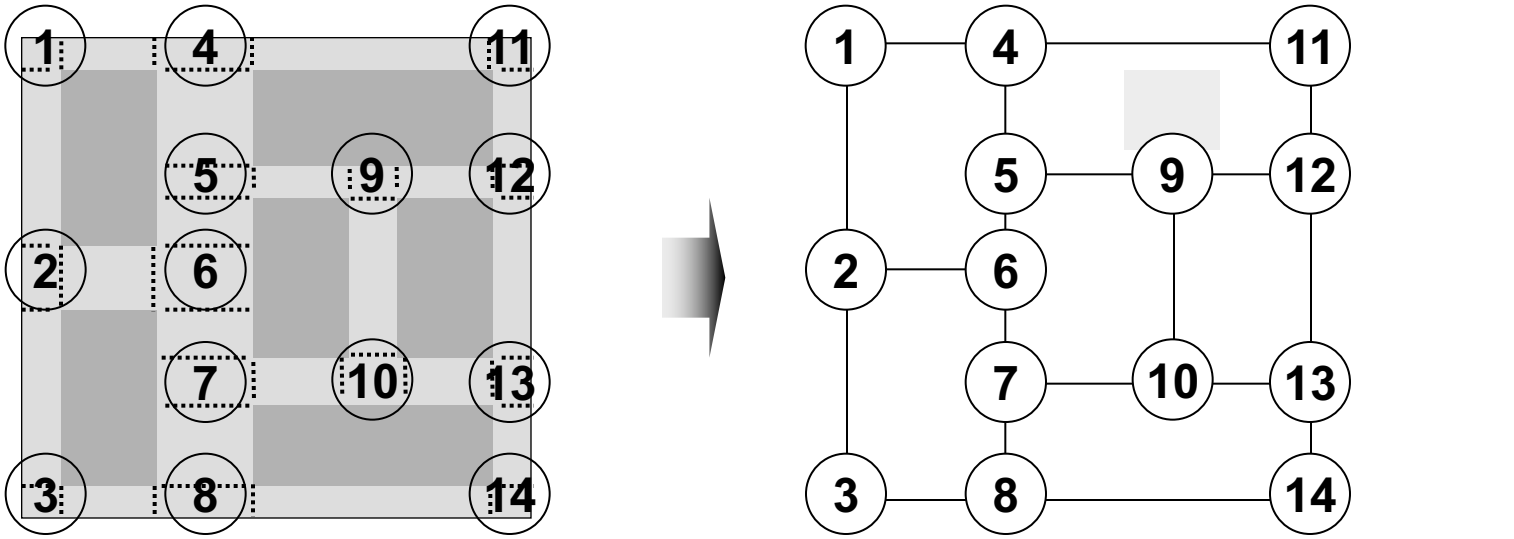



# Global Routing in a Connectivity Graph

Channel connectivity graph

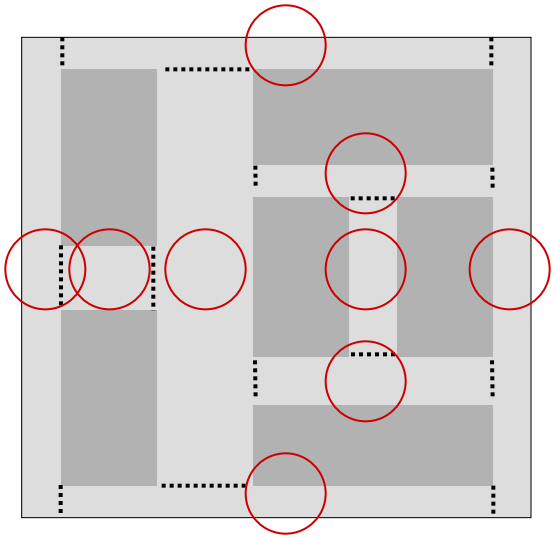



Switchbox connectivity graph

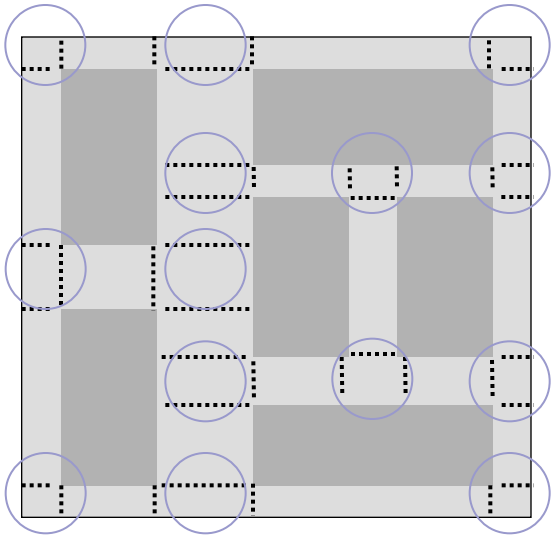


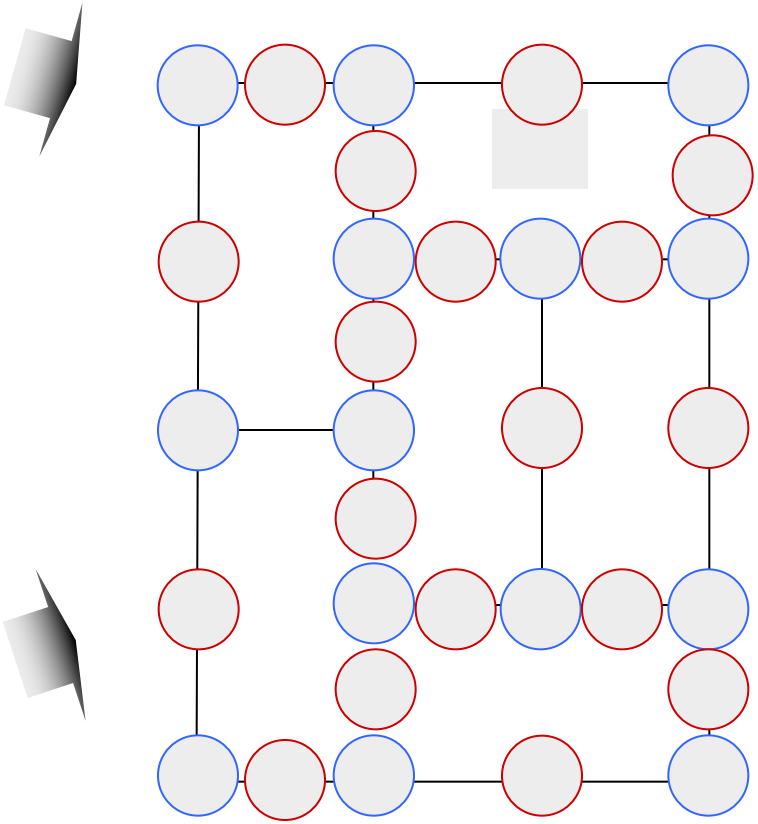
# Global Routing in a Connectivity Graph

Channel connectivity graph



Switchbox connectivity graph

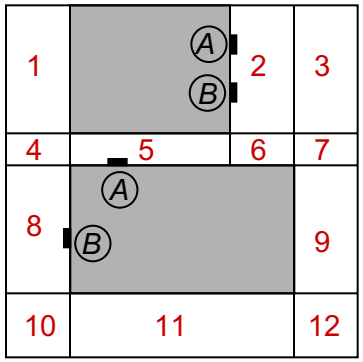




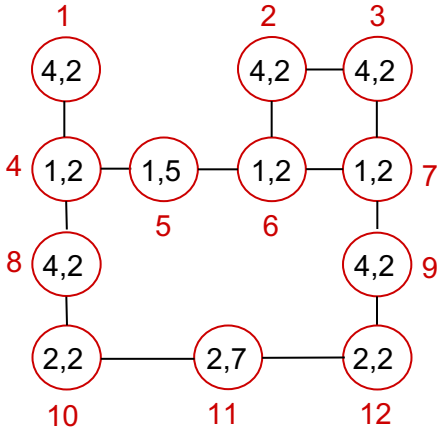
# Global Routing in a Connectivity Graph

- Combines switchboxes and channels, handles non-rectangular block shapes
- Suitable for full-custom design and multi-chip modules

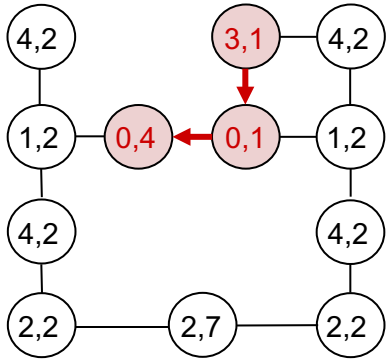
Overview:




Routing regions



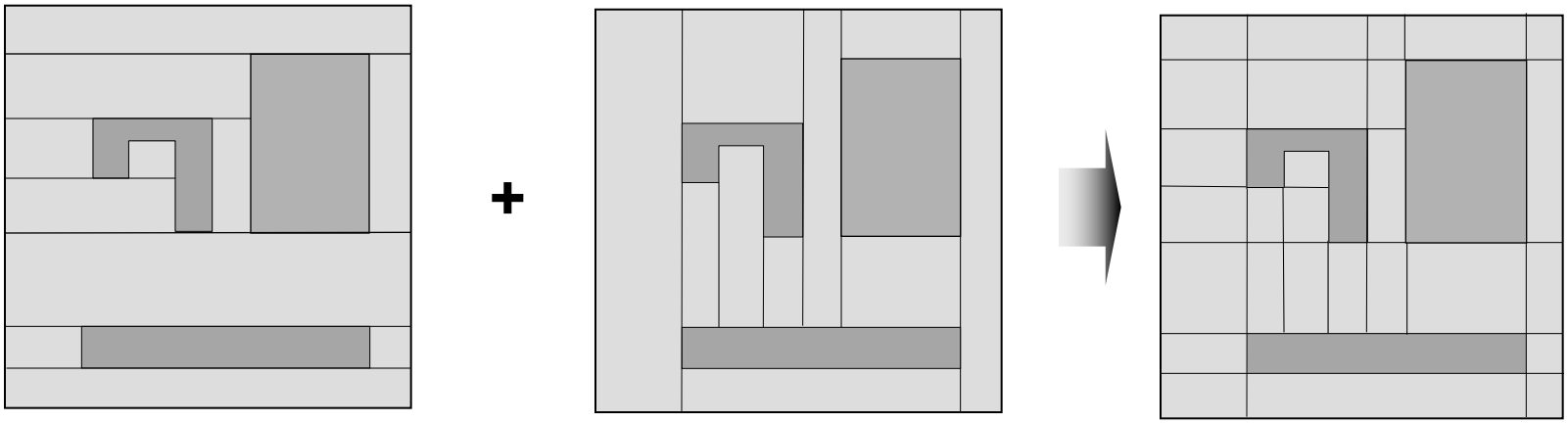

Graph representation




Graph-based path search

# Global Routing in a Connectivity Graph

**Defining the routing regions**

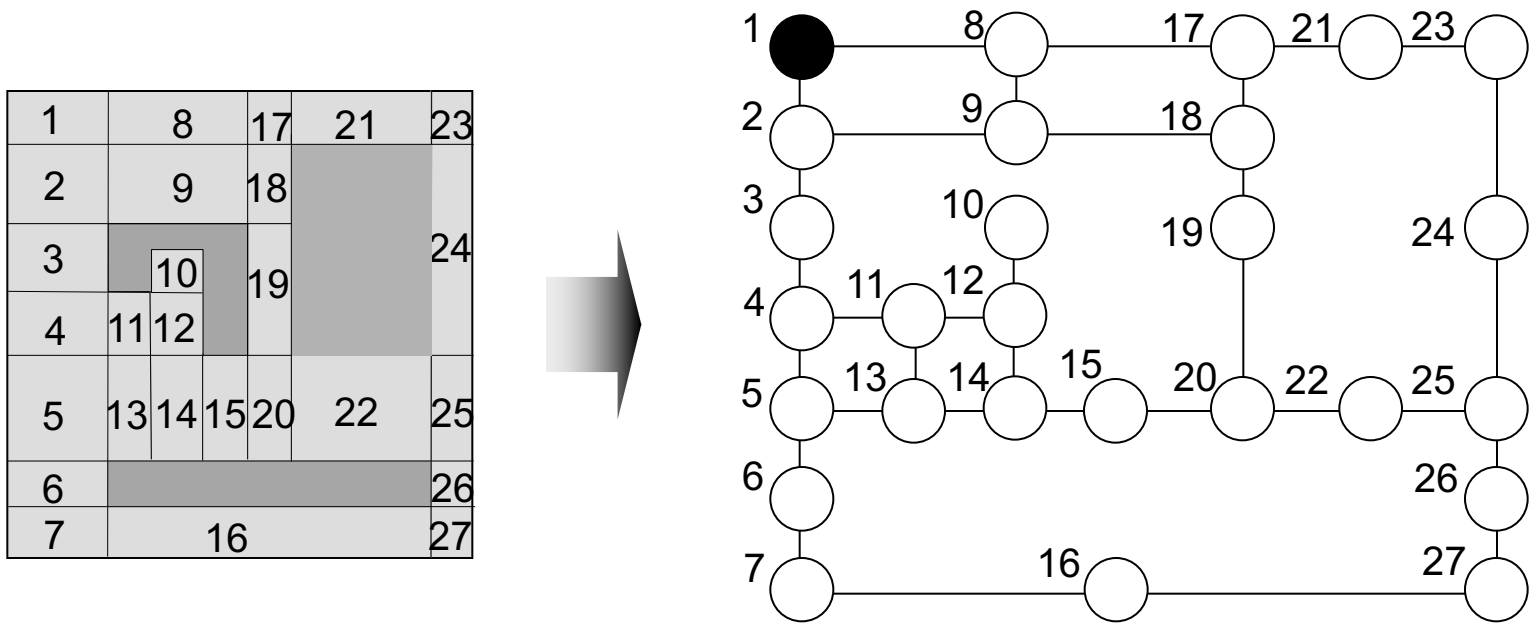


Horizontal macro-cell edges

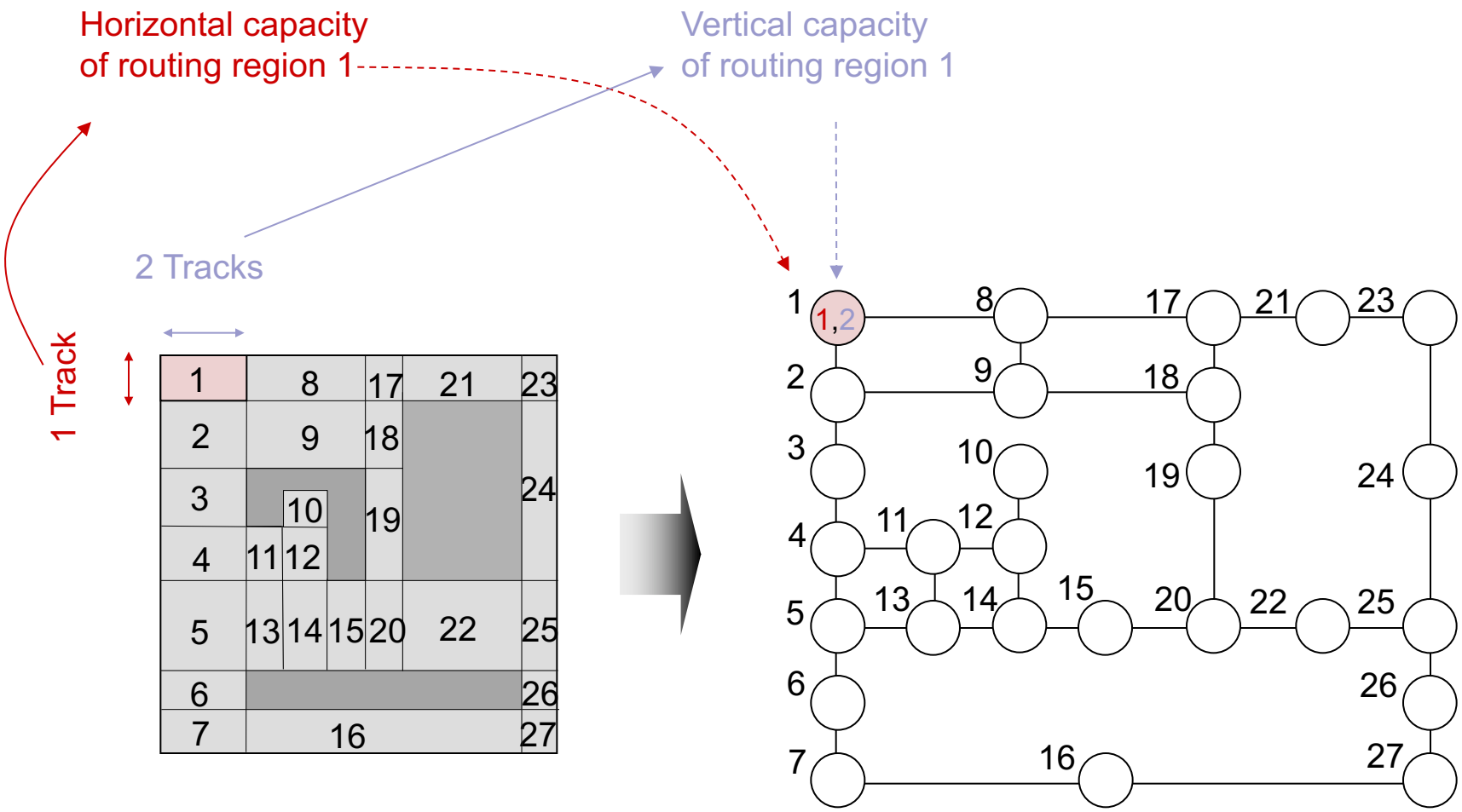
Vertical macro-cell edges

# Global Routing in a Connectivity Graph

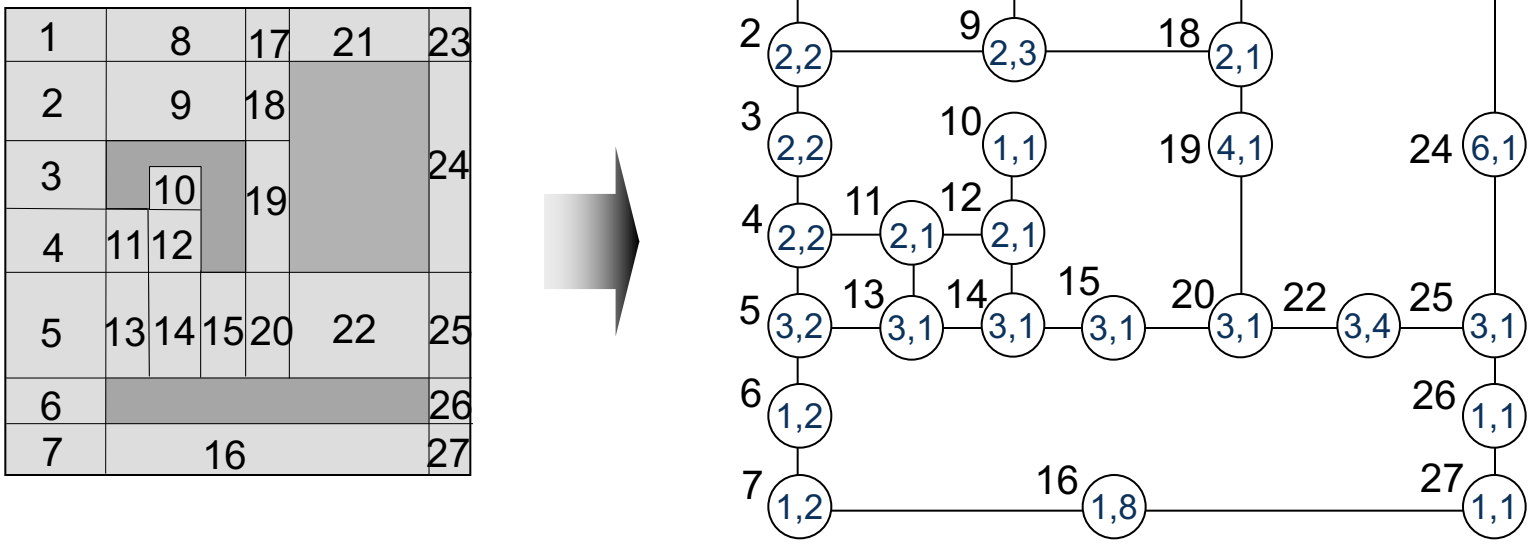
**Defining the connectivity graph**



# Global Routing in a Connectivity Graph



# Global Routing in a Connectivity Graph



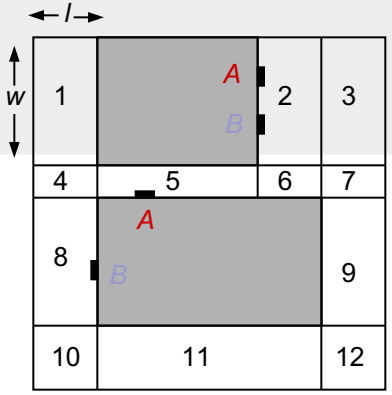
# Global Routing in a Connectivity Graph

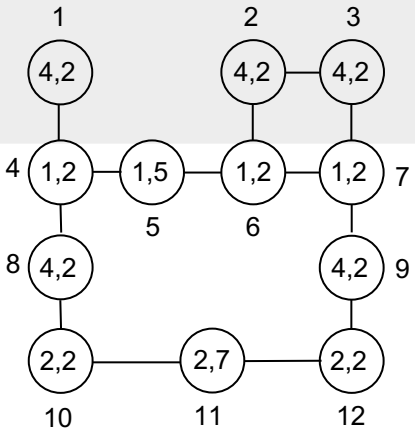
**Algorithm Overview**

1. Define routing regions
2. Define connectivity graph
3. Determine net ordering
4. Consider each net
   a) Free corresponding tracks for net's pins
   b) Decompose net into two-pin subnets
   c) Find shortest path for subnet connectivity graph
   d) If no shortest path exists, do not route, otherwise, assign subnet to the nodes of shortest path and update routing capacities
5. If there are unrouted nets, goto Step 5, otherwise END

Example

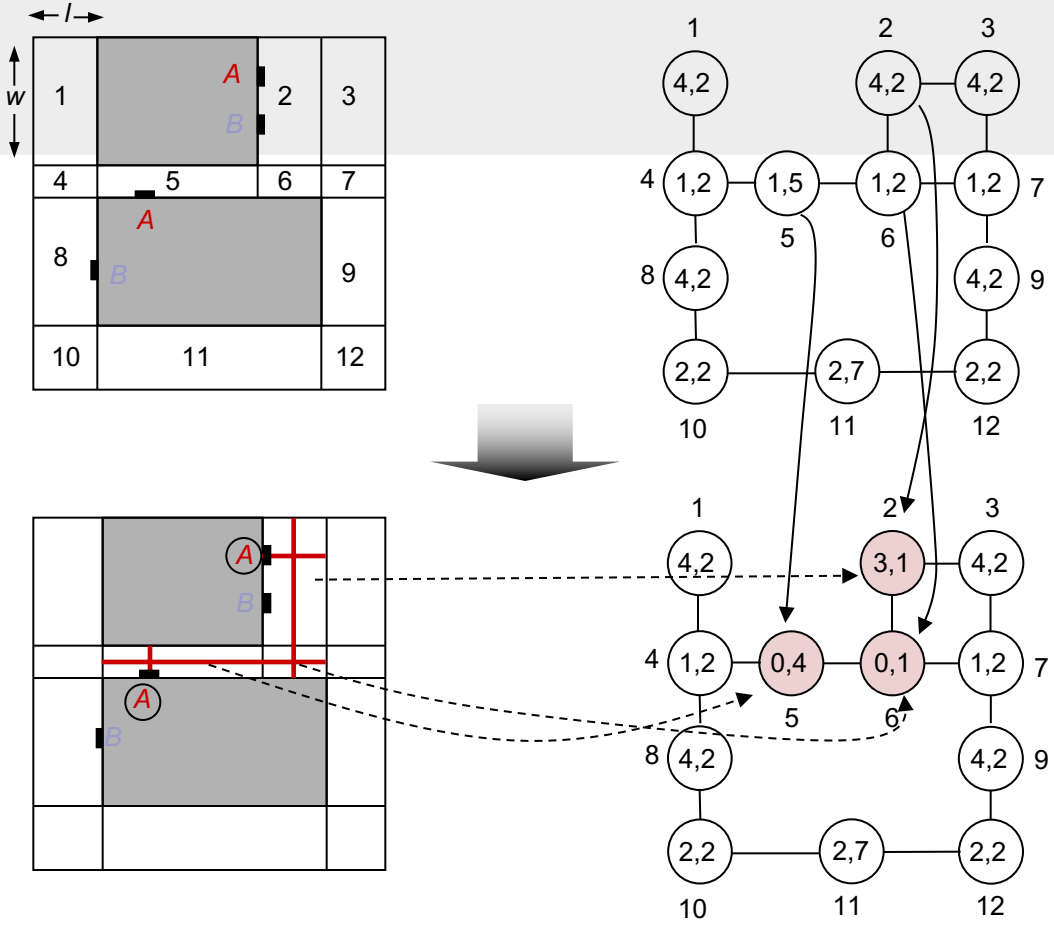
Global routing
of the nets A-A and B-B



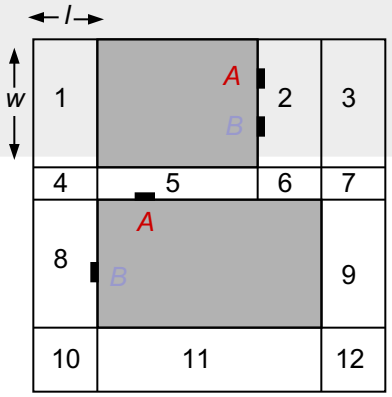
## Example

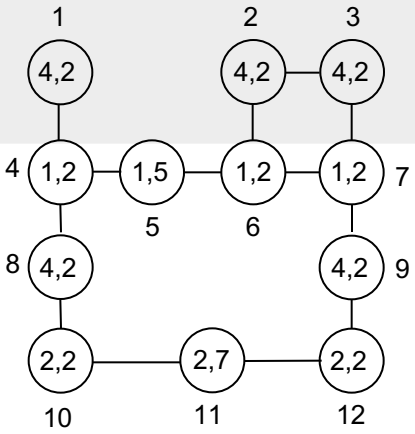
Global routing
of the nets A-A and B-B

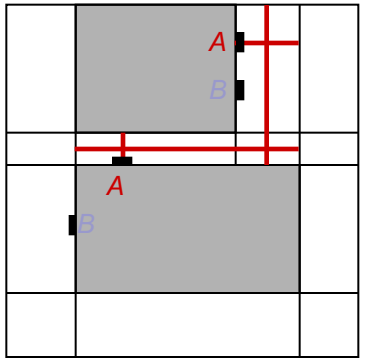


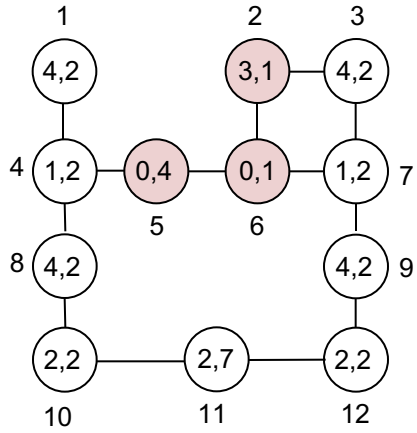
Example

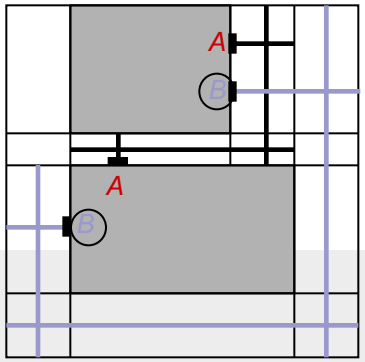
Global routing
of the nets A-A and B-B

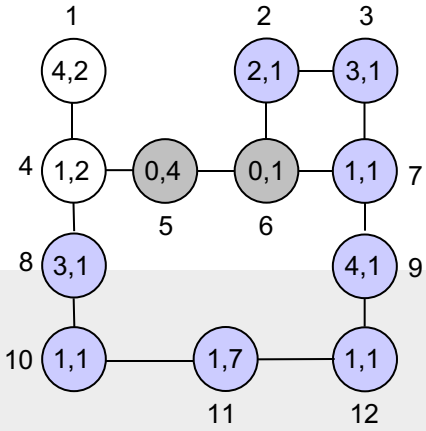


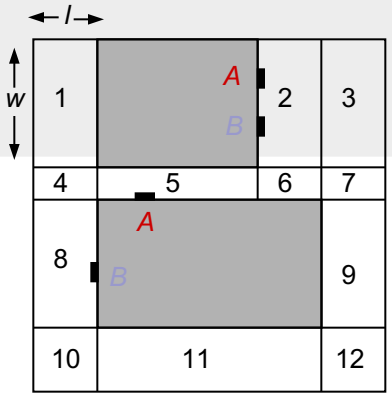



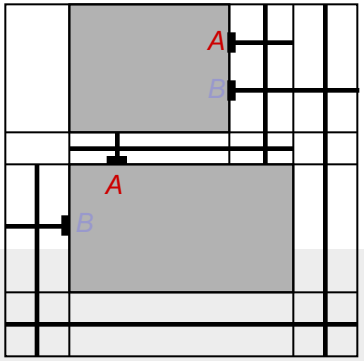



Example

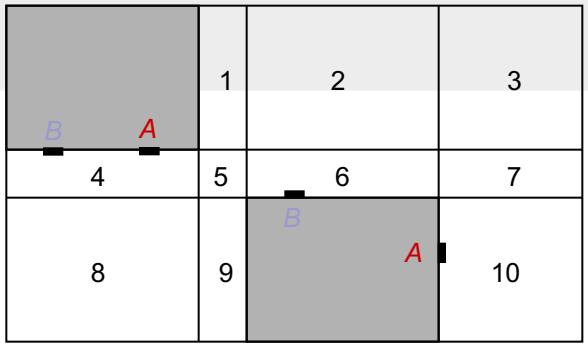
Global routing
of the nets A-A and B-B

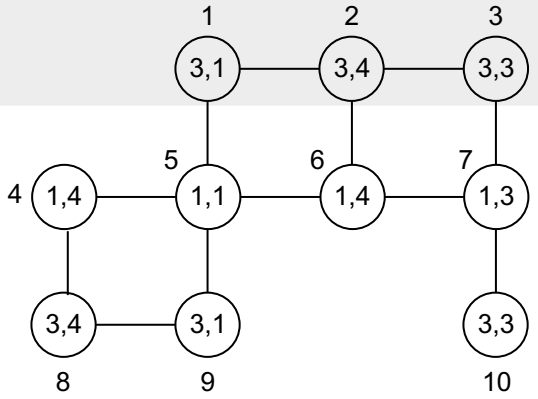


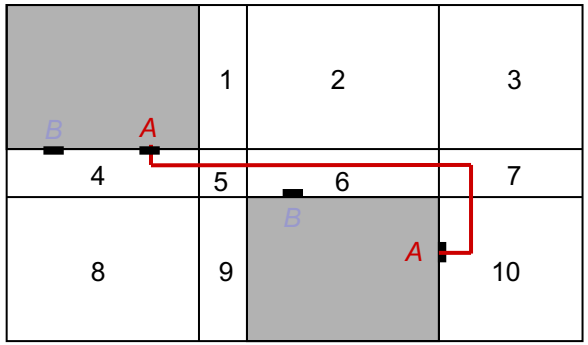



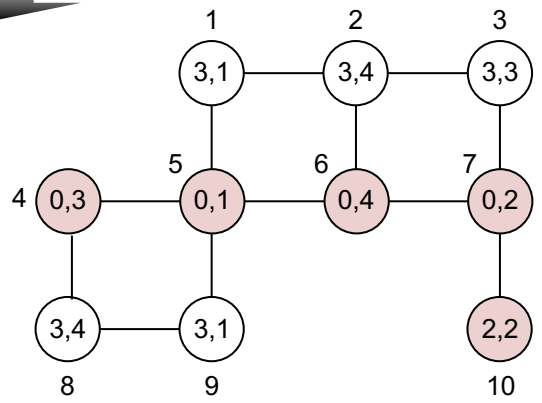
Example

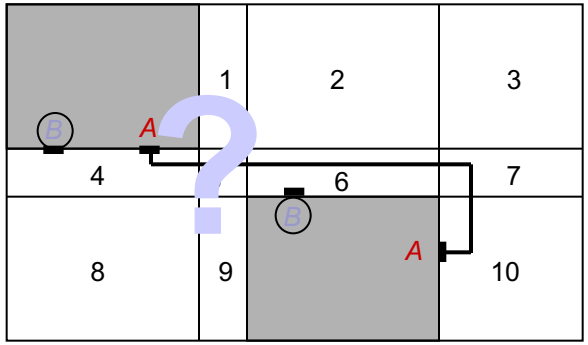
Determine routability of a placement

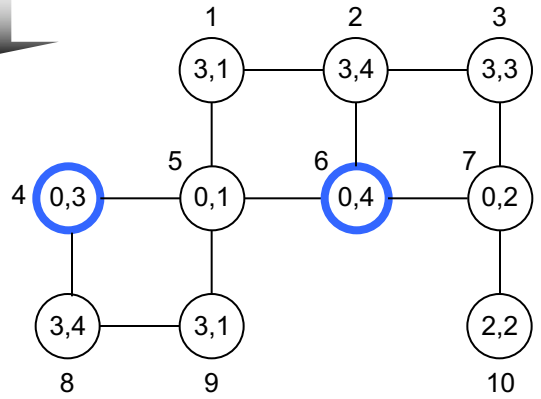


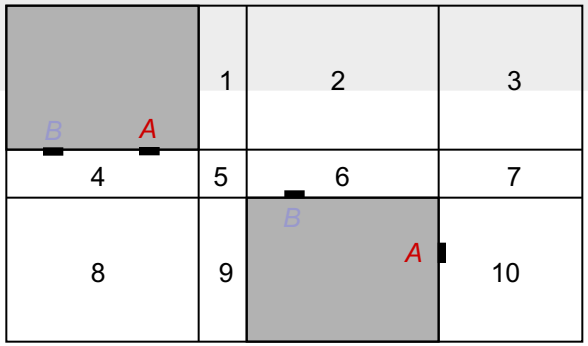



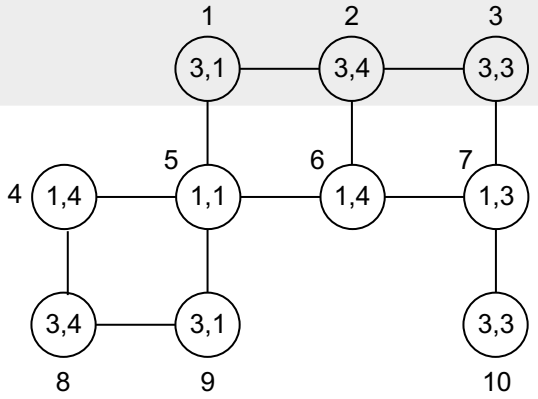



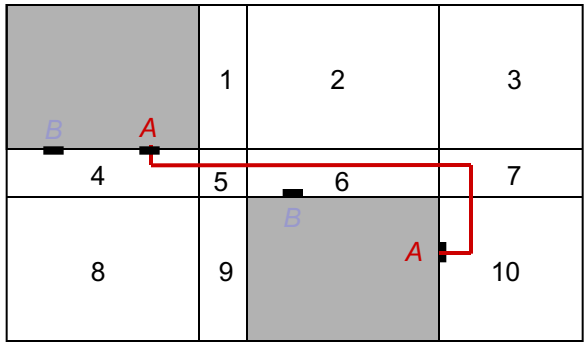



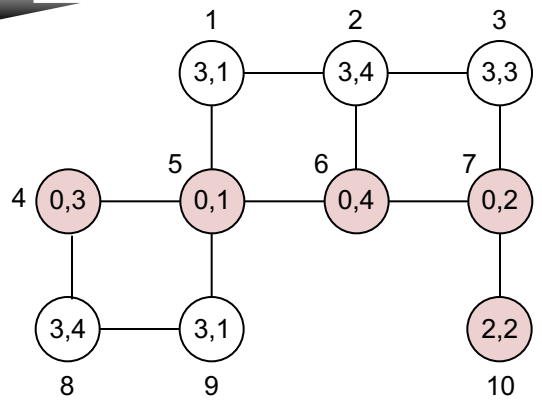
Example

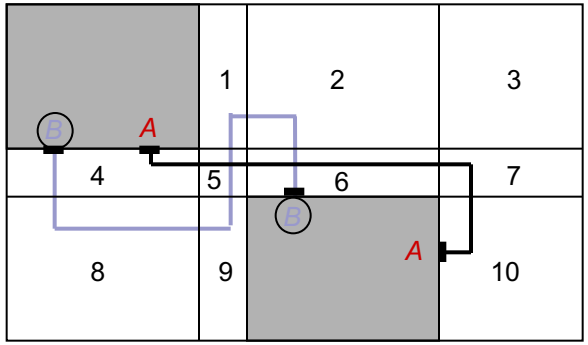
Determine routability of a placement

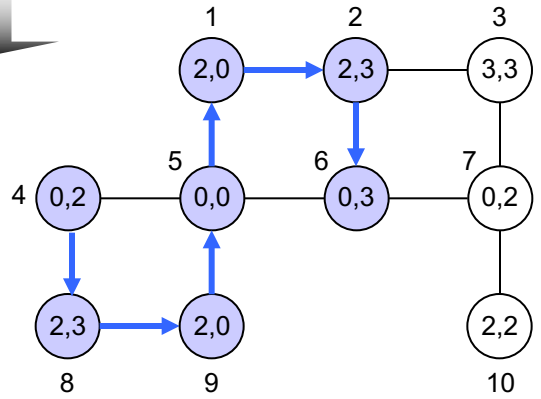


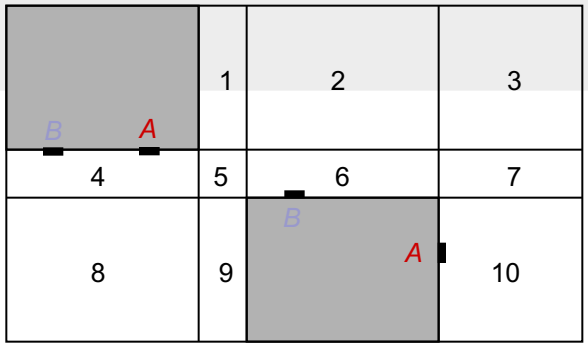



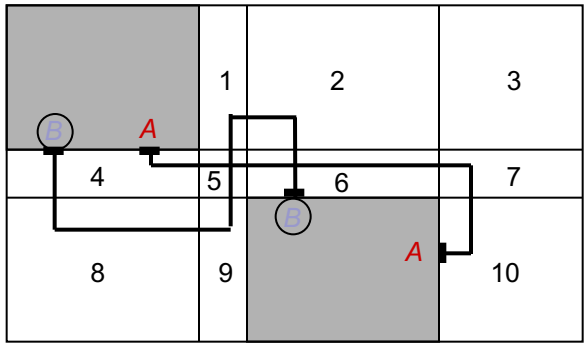



Example

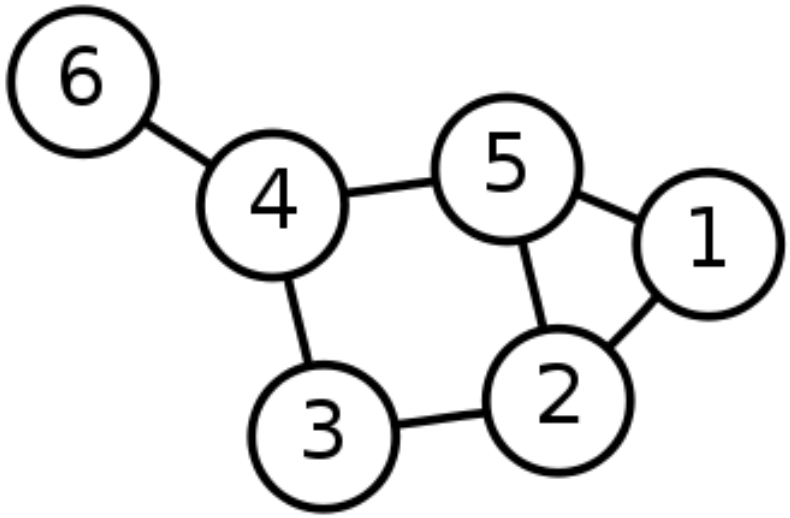
Determine
routability
of a placement



# Single-Source Shortest Path Problem

**Single-Source Shortest Path Problem** - The problem of finding shortest paths from a source vertex *v* to all other vertices in the graph.



# Dijkstra's algorithm

**Dijkstra's algorithm** - is a solution to the single-source shortest path problem in graph theory.

Works on both directed and undirected graphs. However, all edges must have nonnegative weights.

Approach: Greedy

Input: Weighted graph G={E,V} and source vertex $v \in V$, such that all edge weights are nonnegative

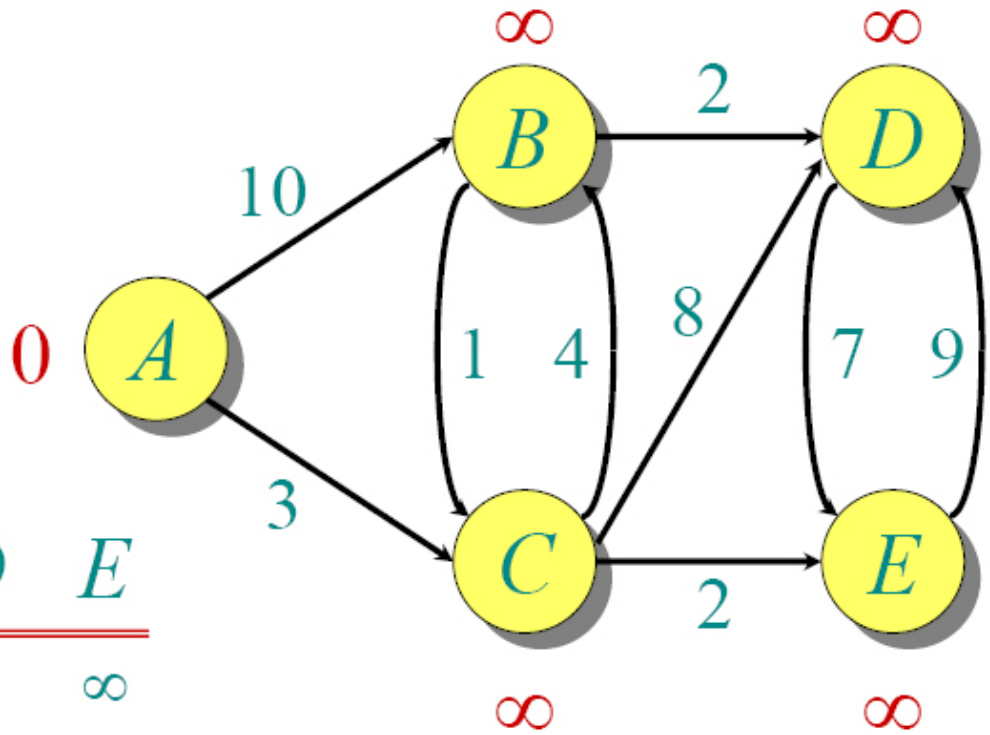
Output: Lengths of shortest paths (or the shortest paths themselves) from a given source vertex $v \in V$ to all other vertices

# Dijkstra's algorithm - Pseudocode

```
dist[s] ←0                              (distance to source vertex is zero)
for  all v ∈ V–{s}
       do  dist[v] ←∞                   (set all other distances to infinity)
S←∅                                     (S, the set of visited vertices is initially empty)
Q←V                                     (Q, the queue initially contains all
vertices)
while Q ≠∅                              (while the queue is not empty)
do   u ← mindistance(Q,dist)            (select the element of Q with the min.
distance)
     S←S ∪ {u}                          (add u to list of visited vertices)
      for all v ∈ neighbors[u]
            do  if   dist[v] > dist[u] + w(u, v)          (if new shortest path found)
                  then      d[v] ←d[u] + w(u, v)      (set new value of shortest path)
                       (if desired, add traceback code)
return dist
```

# Dijkstra Example

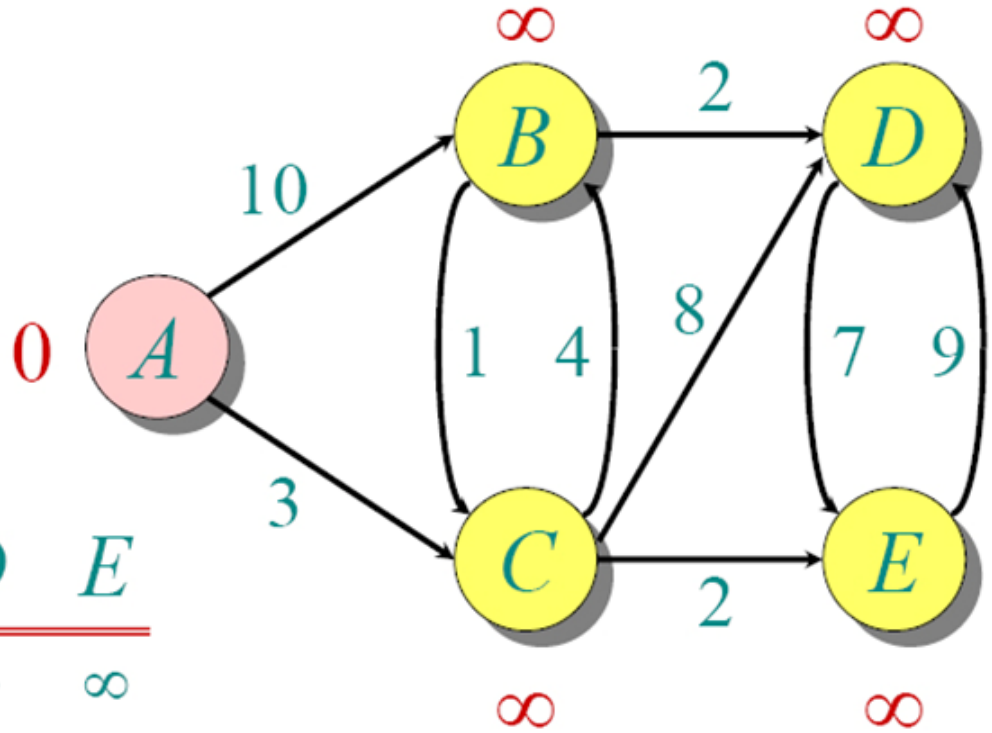
**Initialize:**




| $Q$: | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |

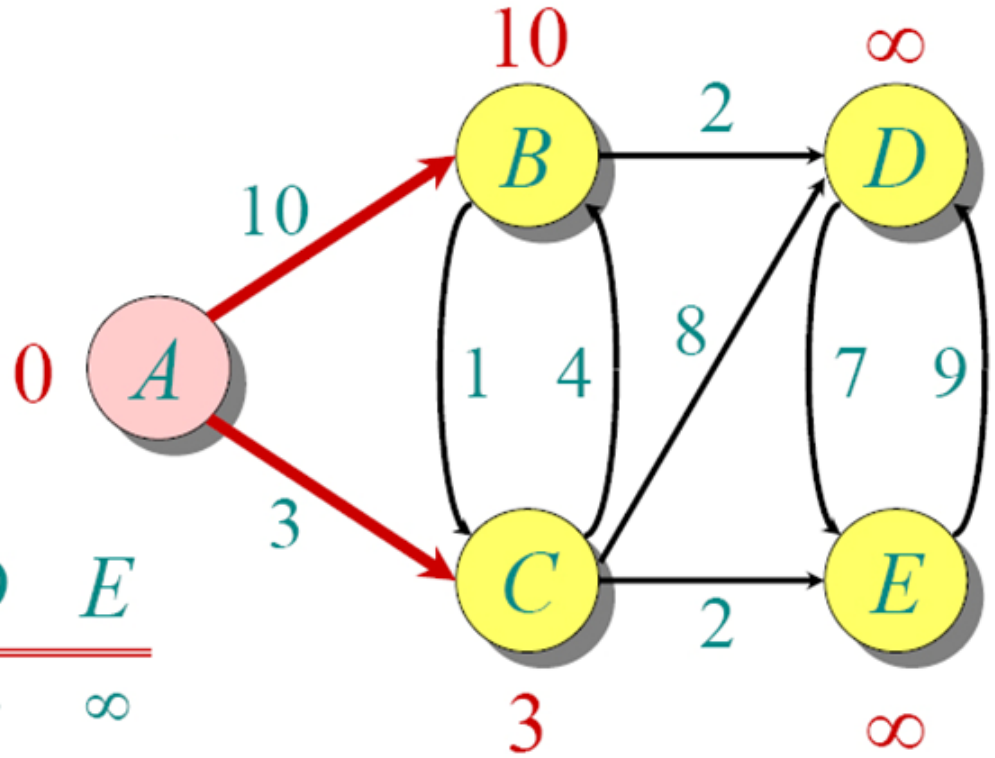
$S$: {}

# Dijkstra Example




| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |

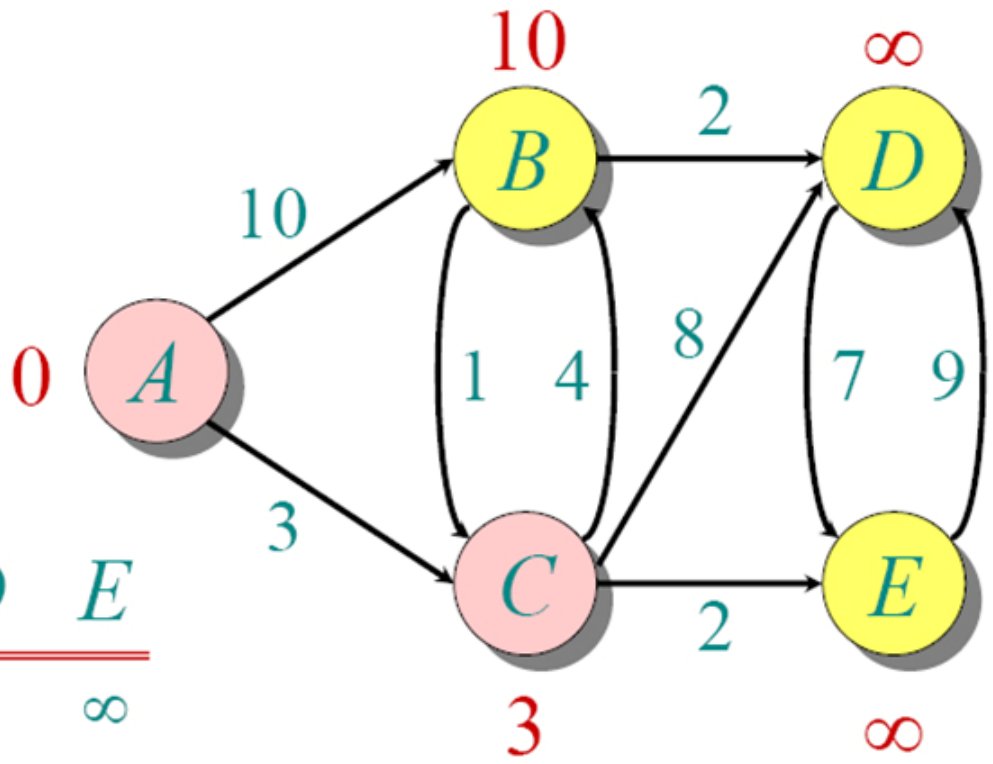
# Dijkstra Example




| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |

$S: \{ A \}$

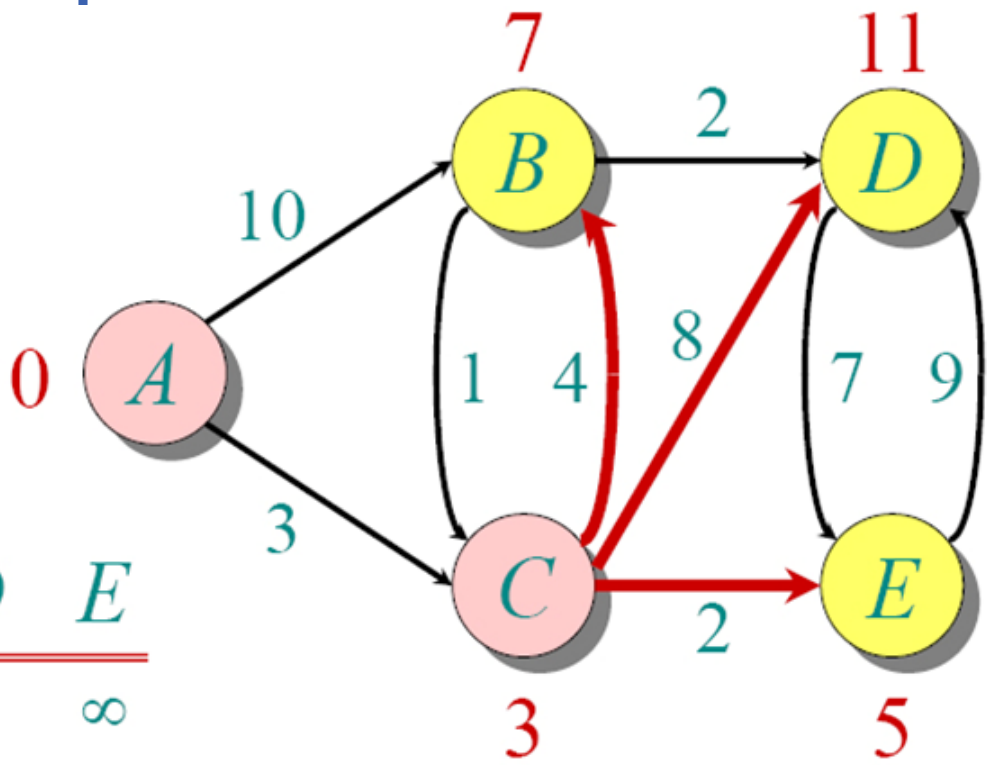
# Dijkstra Example




| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |

$S: \{ A, C \}$

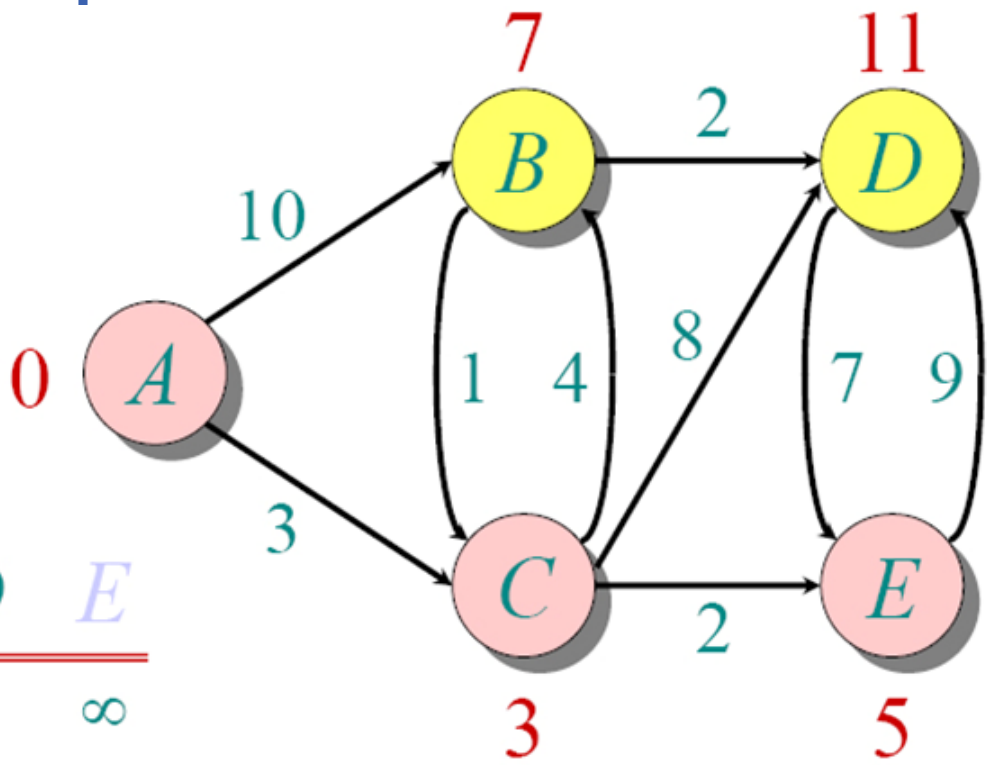
# Dijkstra Example




| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |

$S: \{ A, C \}$

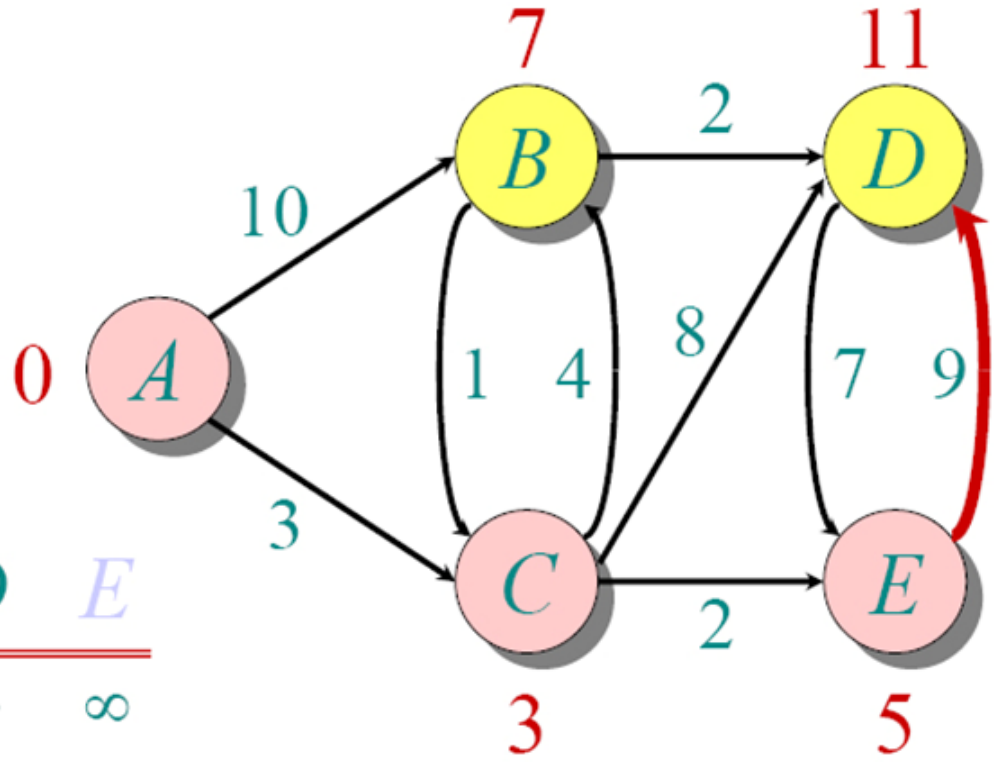
# Dijkstra Example




| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |

$S: \{ A, C, E \}$

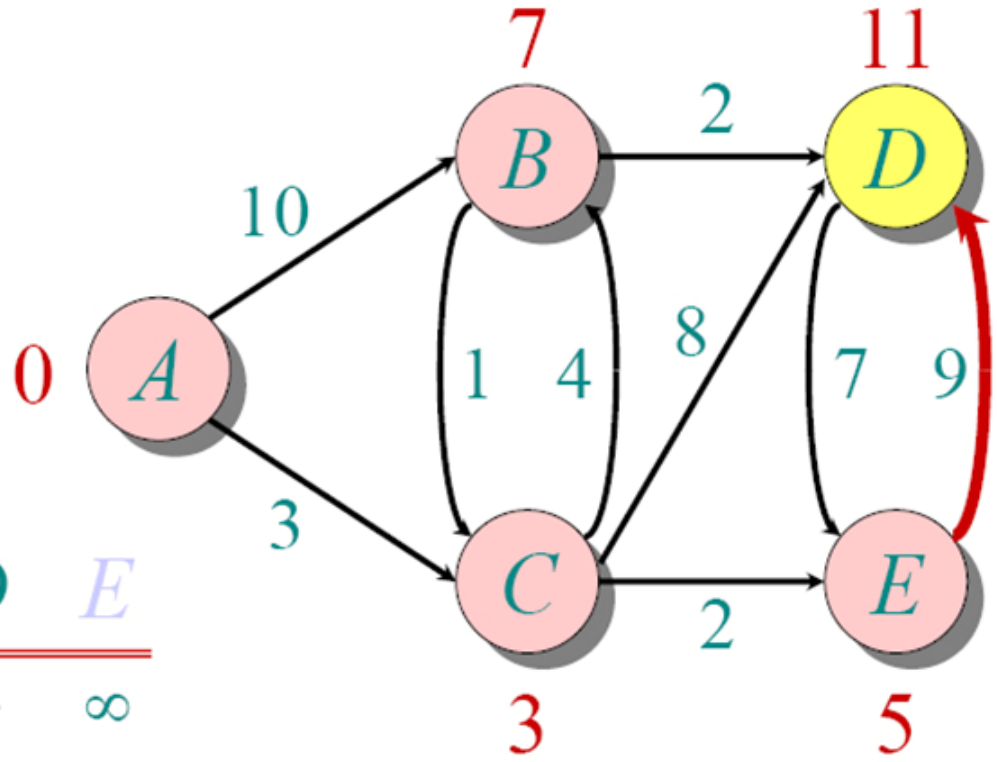
# Dijkstra Example




| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |
| | | 7 | | 11 | |

$S: \{ A, C, E \}$

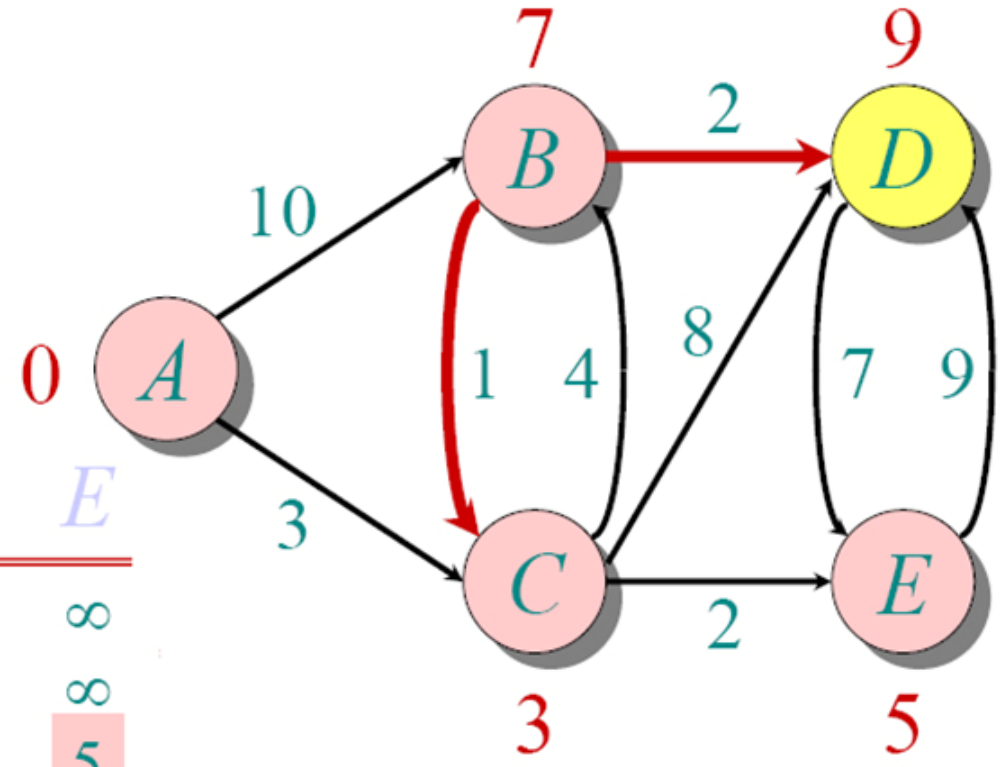
# Dijkstra Example




| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |
| | | 7 | | 11 | |

$S$: { $A$, $C$, $E$, $B$ }

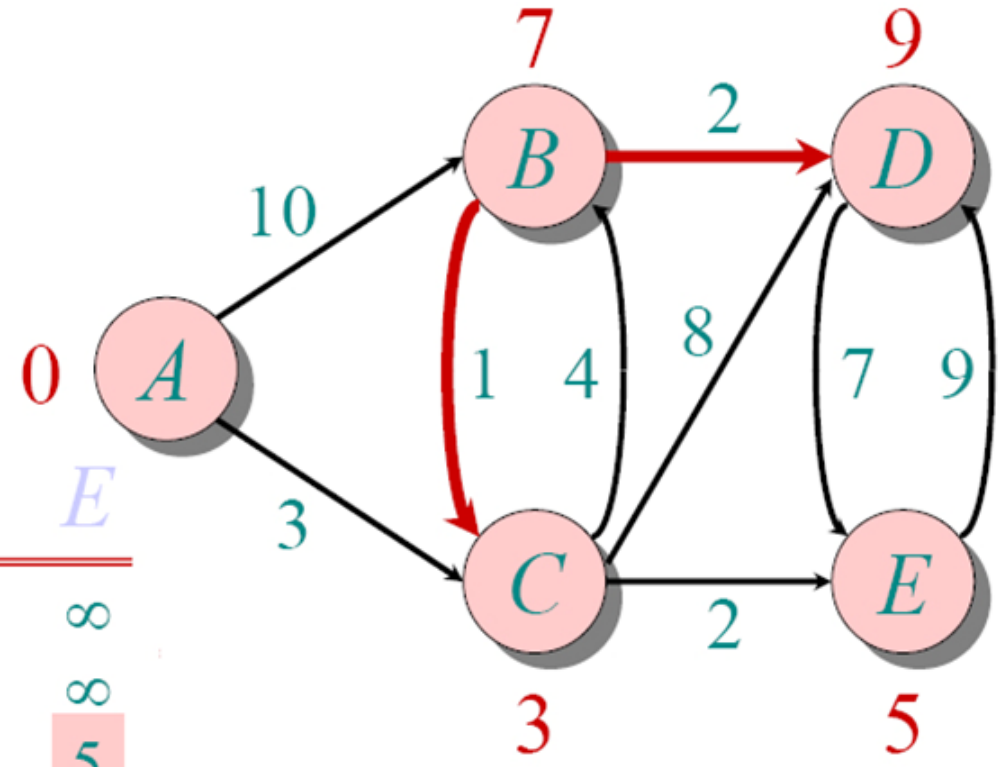
# Dijkstra Example




| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |
| | | 7 | | 11 | |
| | | | | 9 | |

$S$: { $A$, $C$, $E$, $B$ }

# Dijkstra Example

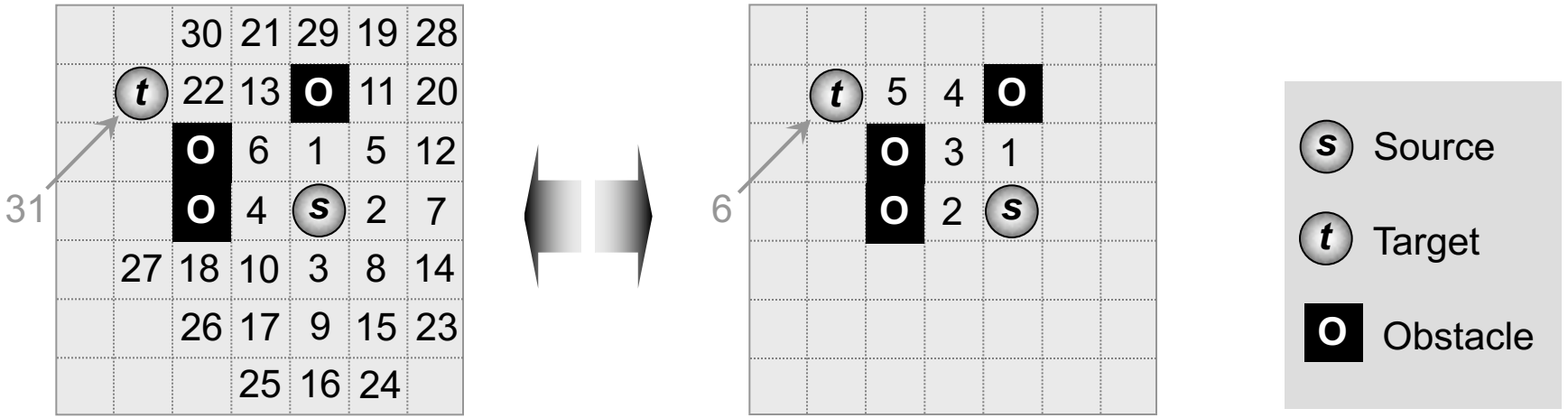



| Q: | A | B | C | D | E |
|---|---|---|---|---|---|
| | 0 | ∞ | ∞ | ∞ | ∞ |
| | | 10 | 3 | ∞ | ∞ |
| | | 7 | | 11 | 5 |
| | | 7 | | 11 | |
| | | | | 9 | |

$S$: { $A$, $C$, $E$, $B$, $D$ }

# Finding Shortest Paths with A* Search

- A* search operates similarly to Dijkstra's algorithm, but extends the cost function to include an estimated distance from the current node to the target
- Expands only the most promising nodes; its best-first search strategy eliminates a large portion of the solution space

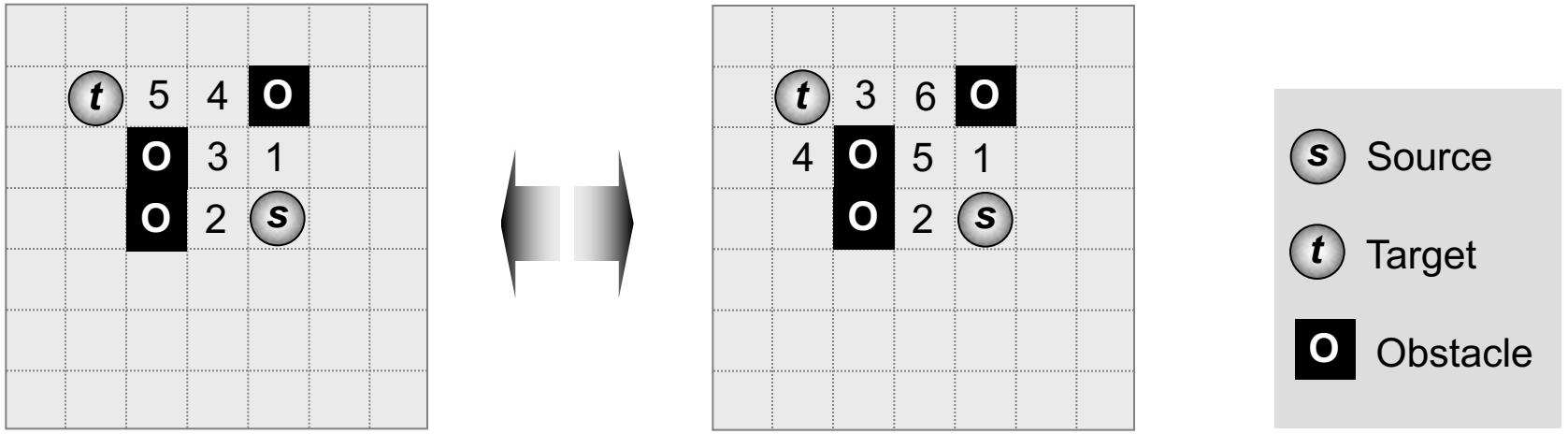



Dijkstra's algorithm
(exploring 31 nodes)

A* search
(exploring 6 nodes)

# Finding Shortest Paths with A* Search

- Bidirectional A* search: nodes are expanded from both the source and target until the two expansion regions intersect
- Number of nodes considered can be reduced




Unidirectional A* search

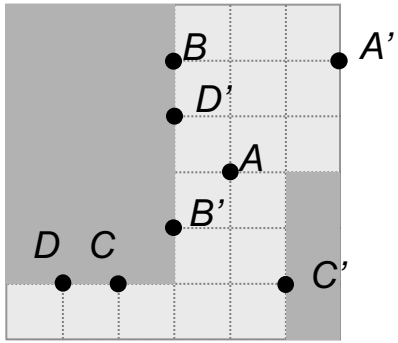
Bidirectional A* search

# Full-Netlist Routing

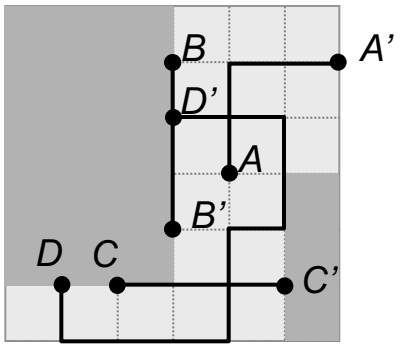
- Global routers must properly match nets with routing resources, without oversubscribing resources in any part of the chip
- Signal nets are either routed
  - simultaneously, e.g., by integer linear programming, or
  - sequentially, e.g., one net at a time
- When certain nets cause resource contention or overflow for routing edges, sequential routing requires multiple iterations: rip-up and reroute

# Rip-Up and Reroute (RRR)

- Rip-up and reroute (RRR) framework: focuses on hard-to-route nets
- Idea: allow temporary violations, so that all nets are routed, but then iteratively remove some nets (rip-up), and route them differently (reroute)

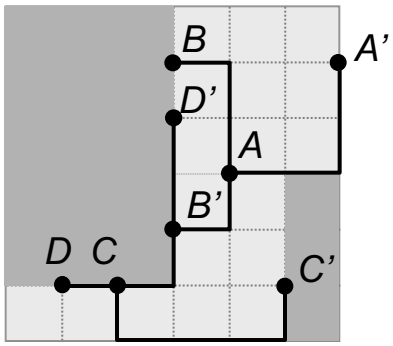



Routing without allowing violations




WL = 21

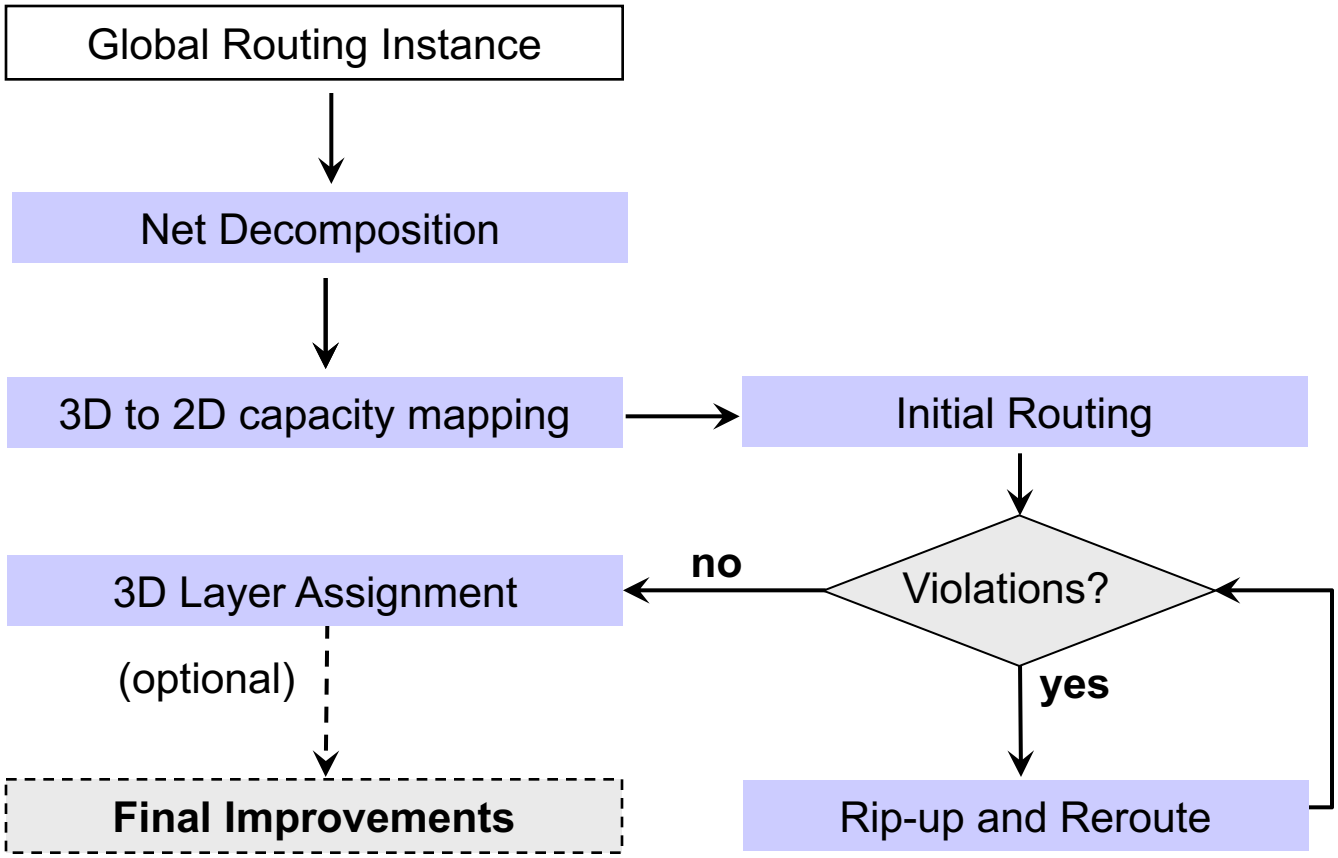
Routing with allowing violations and RRR




WL = 19

# Modern Global Routing

- General flow for modern global routers, where each router uses a unique set of optimizations:



# Modern Global Routing

- Negotiated-Congestion Routing
  - Each edge *e* is assigned a cost value *cost*(*e*) that reflects the demand for edge *e*
  - A segment from net *net* that is routed through *e* pays a cost of *cost*(*e*)
  - Total cost of *net* is the sum of *cost*(*e*) values taken over all edges used by *net*:

$$cost(net) = \sum_{e \in net} cost(e)$$

  - The edge cost *cost*(*e*) is increased according to the *edge congestion* φ(*e*), defined as the total number of nets passing through *e* divided by the capacity of *e*:

$$\varphi(e) = \frac{\eta(e)}{\sigma(e)}$$

  - A higher *cost*(*e*) value discourages nets from using *e* and implicitly encourages nets to seek out other, less used edges
  - ⇒ Iterative routing approaches (Dijkstra's algorithm, A* search, etc.) find routes with minimum cost while respecting edge capacities

# Summary

- Input: netlist, placement, obstacles + (usually) routing grid
- Partitions the routing region (chip or block) into global routing cells (gcells)
- Considers the locations of cells within a region as identical
- Plans routes as sequences of gcells
- Minimizes total length of routes and, possibly, routed congestion
- May fail if routing resources are insufficient
  - Variable-die can expand the routing area, so can't usually fail
  - Fixed-die is more common today (cannot resize a block in a larger chip)
- Interpreting failures in global routing
  - Failure with many violations => must restructure the netlist and/or redo global placement
  - Failure with few violations => detailed routing may be able to fix the problems

# Summary

- Usually ~50% of the nets are two-pin nets, ~25% have three pins, ~12.5% have four, etc.
  - Two-pin nets can be routed as L-shapes or using maze search (in a connectivity graph of the routing regions)
  - Three-pin nets usually have 0 or 1 branching point
  - Larger nets are more difficult to handle
- Pattern routing
  - For each net, considers only a small number of shapes (L, Z, U, T, E)
  - Very fast, but misses many opportunities
  - Good for initial routing, sometimes is sufficient
- Routing pin-to-pin connections
  - Breadth-first-search (when costs are uniform)
  - Dijkstra's algorithm (non-uniform costs)
  - A*-search (non-uniform costs and/or using additional distance information)