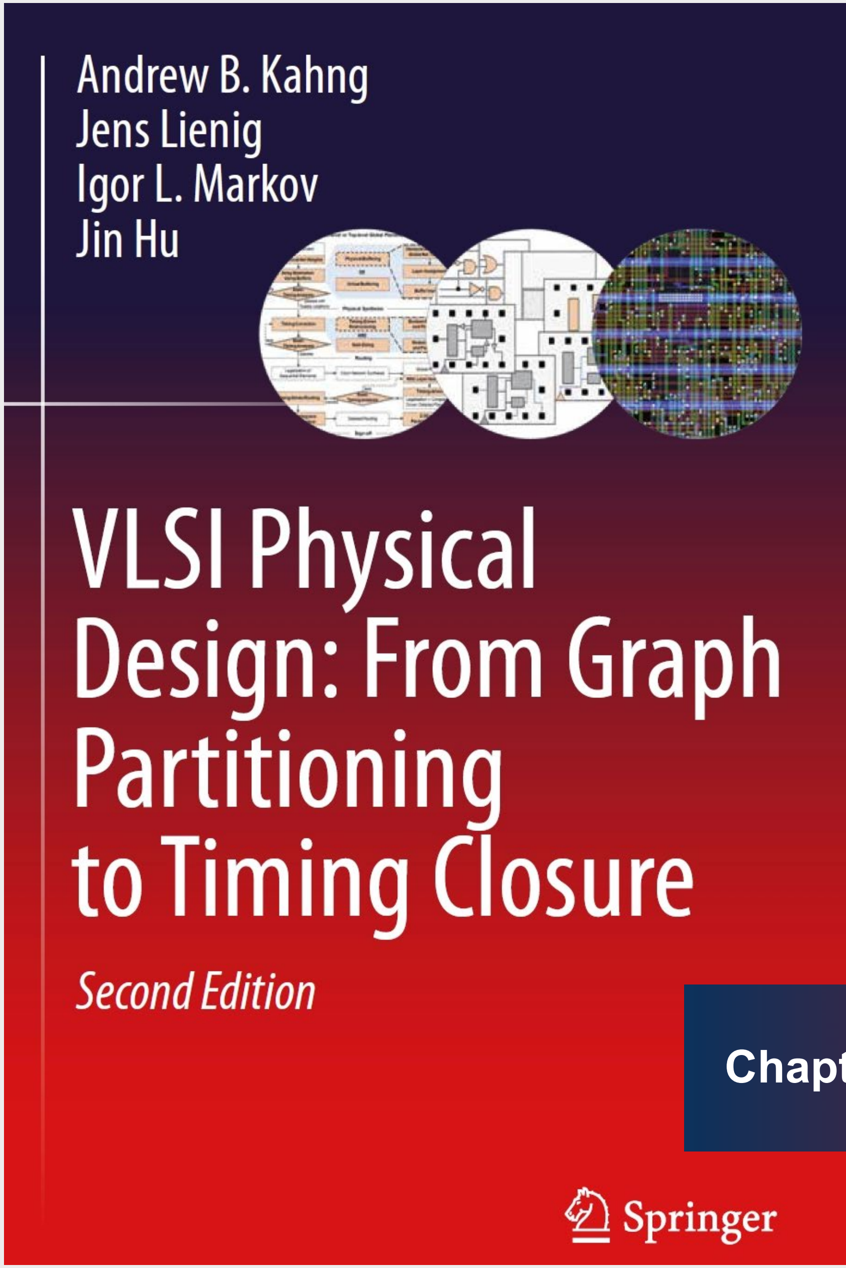
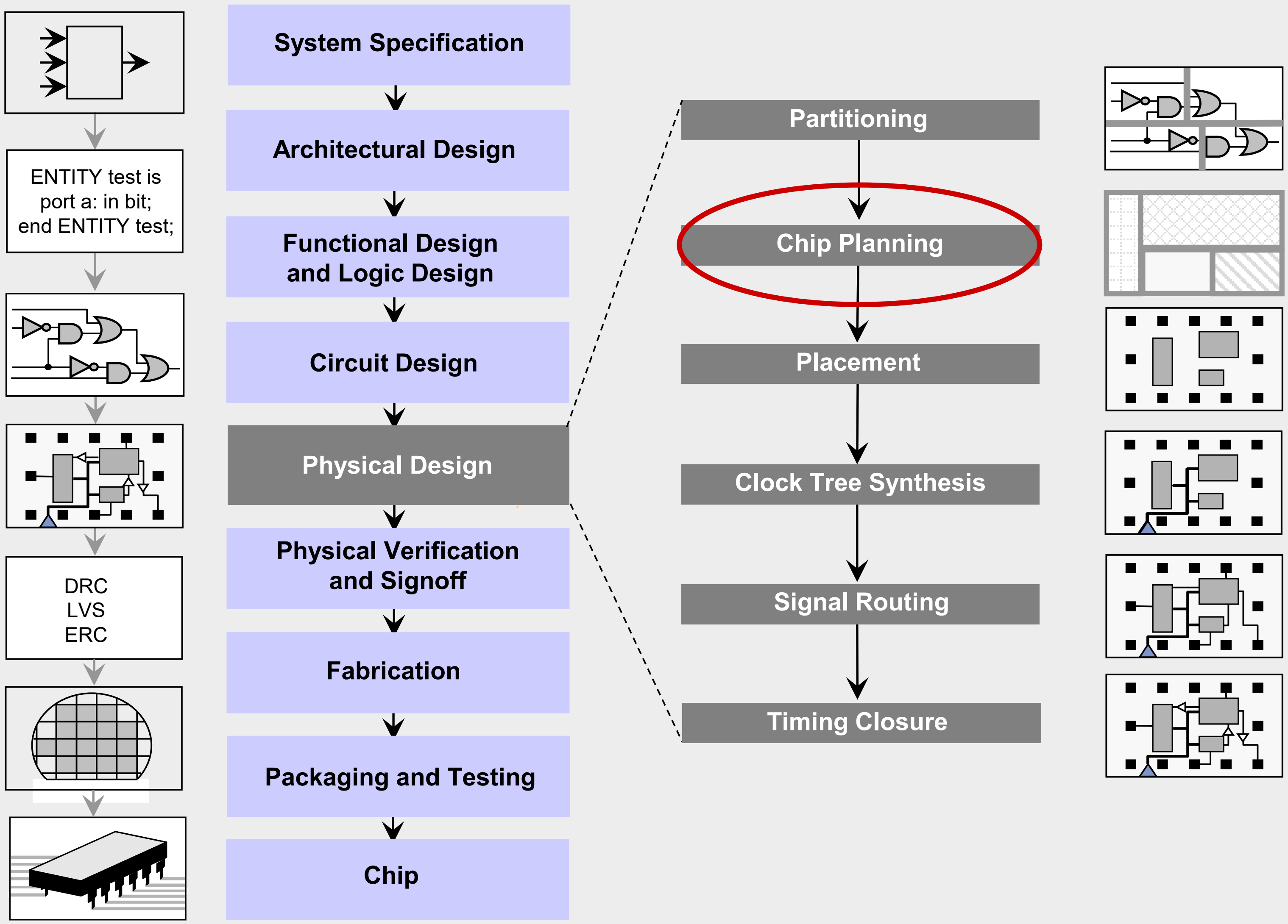


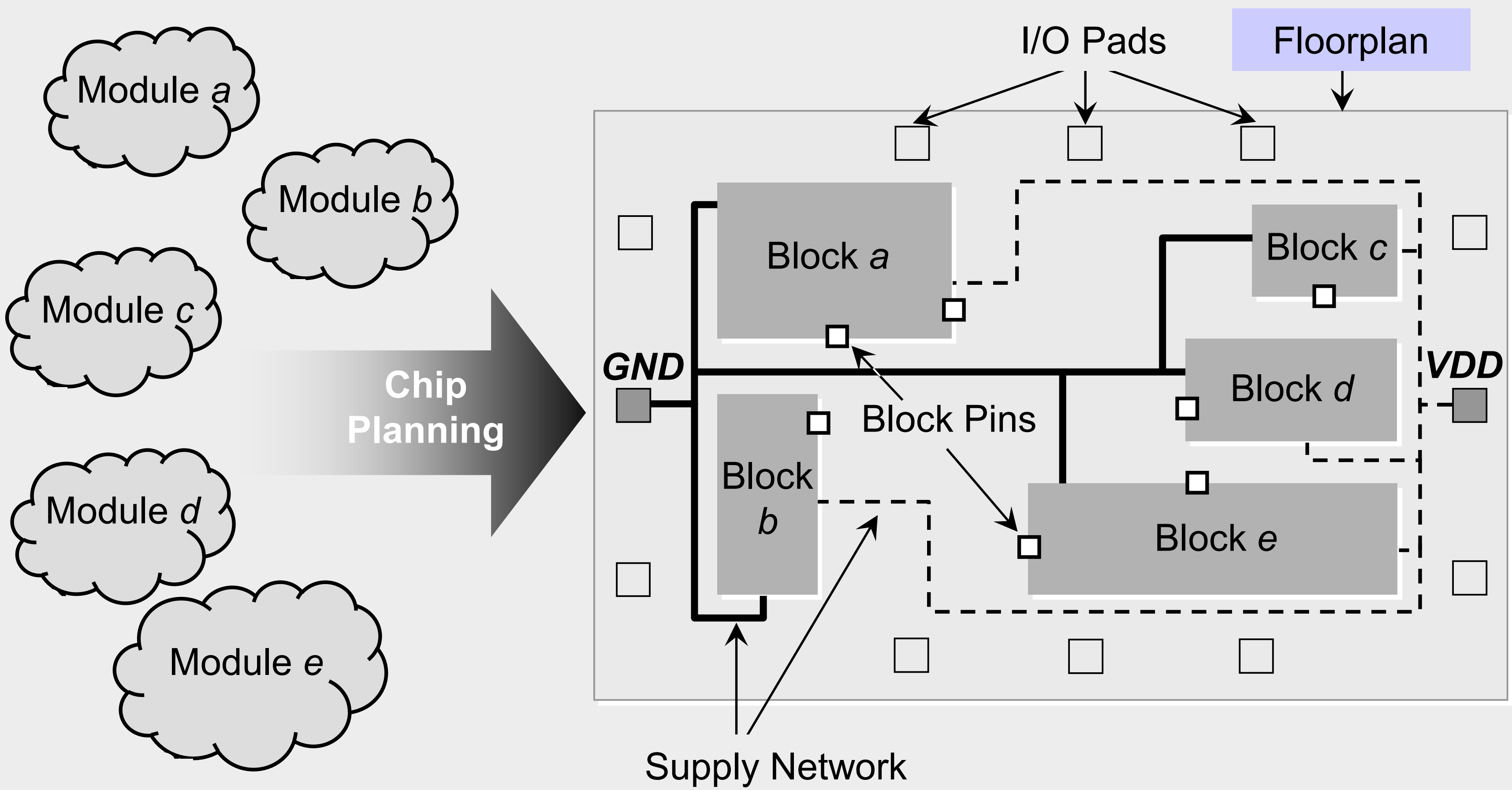
Chapter 3 – Chip Planning

Chapter 3 – Chip Planning

3.1 Introduction



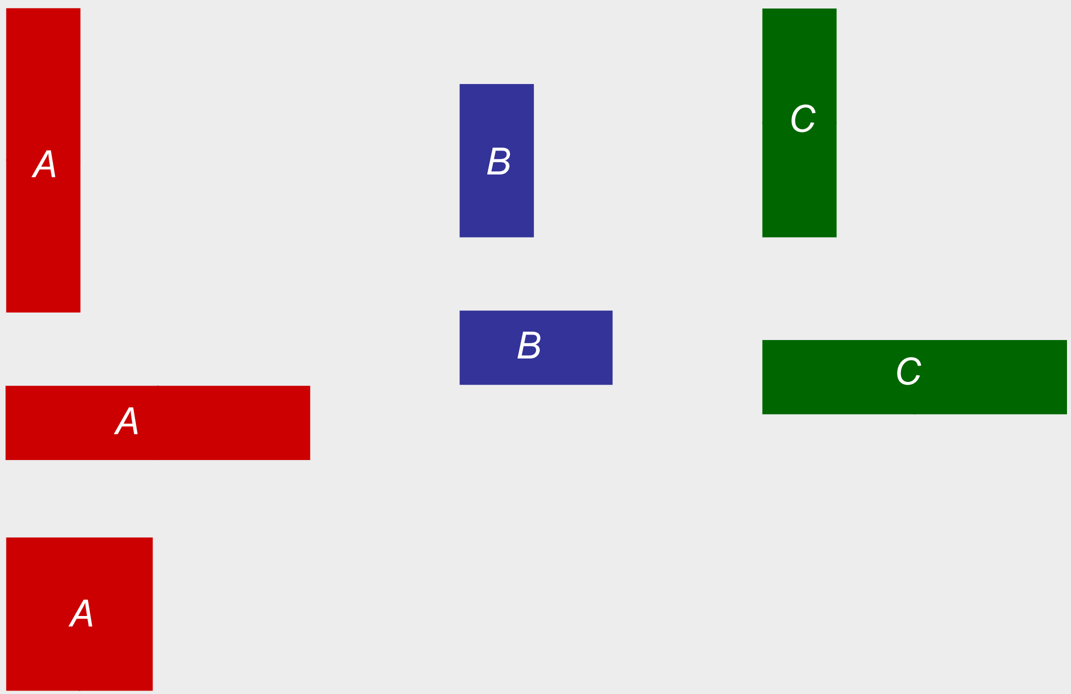
3.1 Introduction



3.1 Introduction

Example
Given: Three blocks with the following potential widths and heights
Block *A*: $w = 1, h = 4$ or $w = 4, h = 1$ or $w = 2, h = 2$
Block *B*: $w = 1, h = 2$ or $w = 2, h = 1$
Block *C*: $w = 1, h = 3$ or $w = 3, h = 1$

Task: Floorplan with minimum total area enclosed



3.1 Introduction

Example

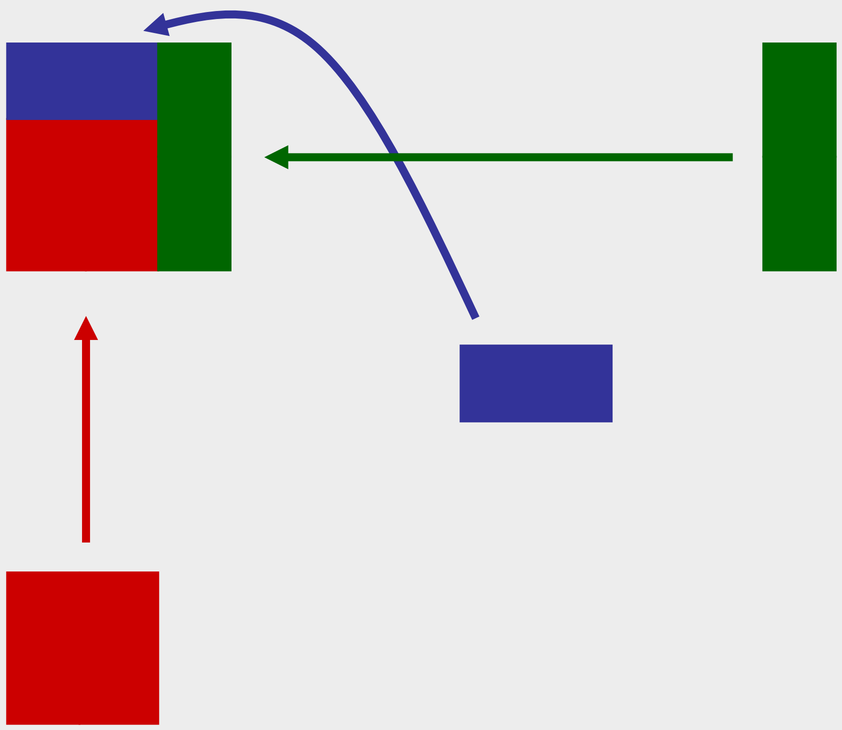
Given: Three blocks with the following potential widths and heights

Block A: w = 1, h = 4 or w = 4, h = 1 or w = 2, h = 2

Block B: w = 1, h = 2 or w = 2, h = 1

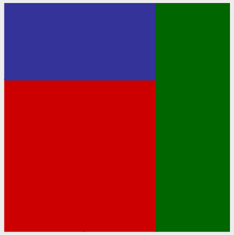
Block C: w = 1, h = 3 or w = 3, h = 1

Task: Floorplan with minimum total area enclosed



Example
Given: Three blocks with the following potential widths and heights
Block A: $w = 1$, $h = 4$ or $w = 4$, $h = 1$ or $w = 2$, $h = 2$
Block B: $w = 1$, $h = 2$ or $w = 2$, $h = 1$
Block C: $w = 1$, $h = 3$ or $w = 3$, $h = 1$

Task: Floorplan with minimum total area enclosed



Solution:
Aspect ratios
Block A with $w = 2$, $h = 2$; Block B with $w = 2$, $h = 1$; Block C with $w = 1$, $h = 3$

This floorplan has a global bounding box with minimum possible area (9 square units).

- Area and shape of the global bounding box
  - Global bounding box of a floorplan is the minimum axis-aligned rectangle that contains all floorplan blocks.
  - Area of the global bounding box represents the area of the top-level floorplan
  - Minimizing the area involves finding (x,y) locations, as well as shapes, of the individual blocks.
- Total wirelength
  - Long connections between blocks may increase signal propagation delays in the design.
- Combination of area $area(F)$ and total wirelength $L(F)$ of floorplan F
  - Minimize $\alpha \cdot area(F) + (1 - \alpha) \cdot L(F)$
    where the parameter $0 \leq \alpha \leq 1$ gives the relative importance between $area(F)$ and $L(F)$
- Signal delays
  - Static timing analysis is used to identify the interconnects that lie on critical paths.

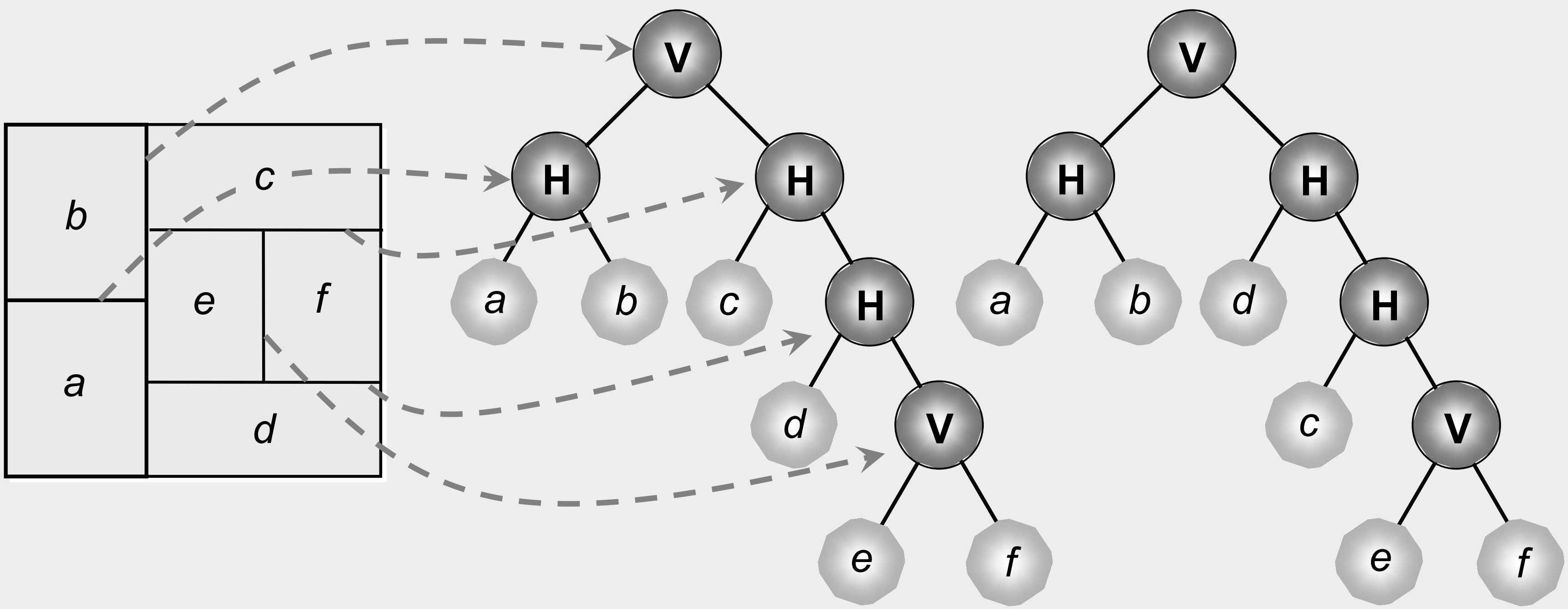
3.3 Terminology

- A rectangular dissection is a division of the chip area into a set of *blocks* or non-overlapping rectangles.

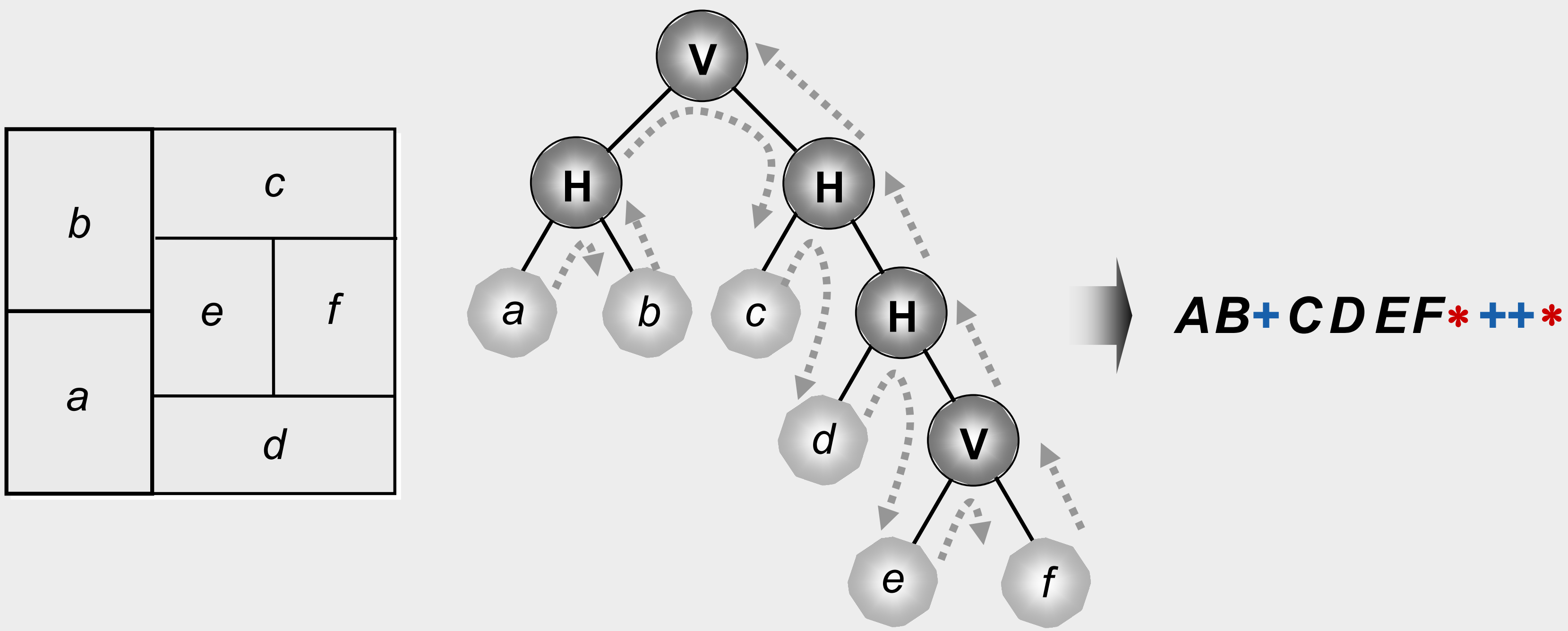
- A slicing floorplan is a rectangular dissection
  - Obtained by repeatedly dividing each rectangle, starting with the entire chip area, into two smaller rectangles
  - Horizontal or vertical cut line.

- A slicing tree or slicing floorplan tree is a binary tree with k leaves and k – 1 internal nodes
  - Each leaf represents a block
  - Each internal node represents a horizontal or vertical cut line.

Slicing floorplan and two possible corresponding slicing trees

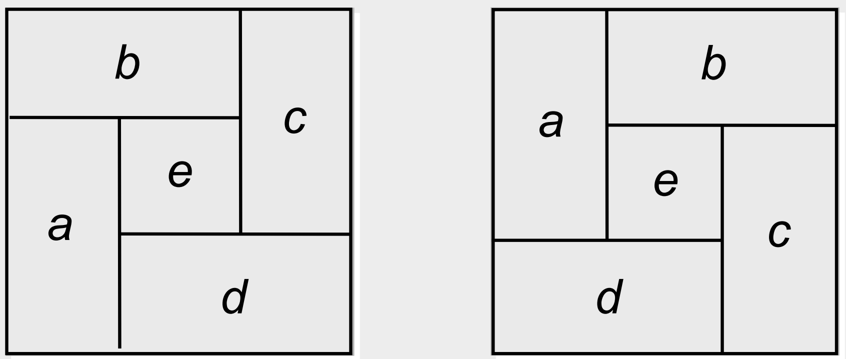


Polish expression

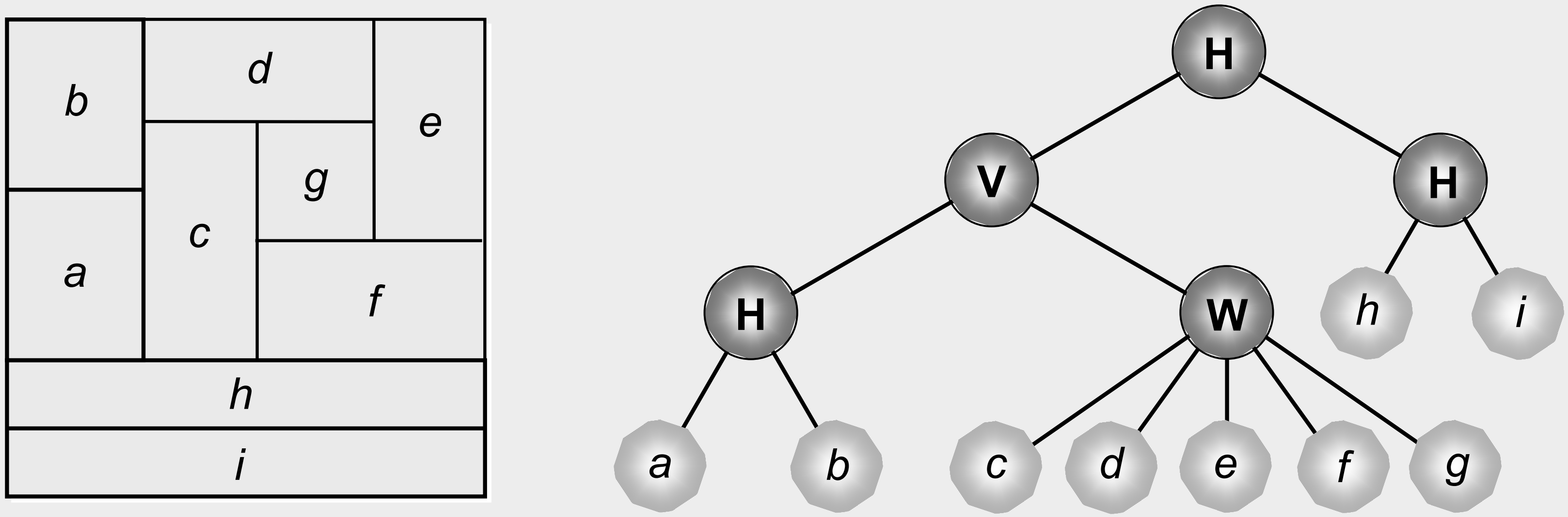


- Bottom up: **V** → * and **H** → +
- Length 2*n*-1 (*n* = Number of leaves of the slicing tree)

Non-slicing floorplans (wheels)



Floorplan tree: Tree that represents a hierarchical floorplan



H Horizontal division (objects to the top and bottom)

V Vertical division (objects to the left and right)

W Wheel (4 objects cycled around a center object)

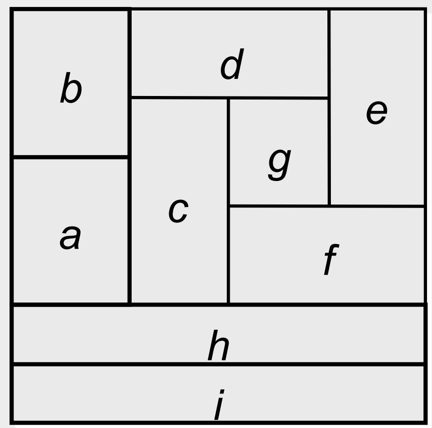
- In a vertical constraint graph (VCG), node weights represent the heights of the corresponding blocks.
  - Two nodes v_i and v_j, with corresponding blocks m_i and m_j, are connected with a directed edge from v_i to v_j if m_i is below m_j.

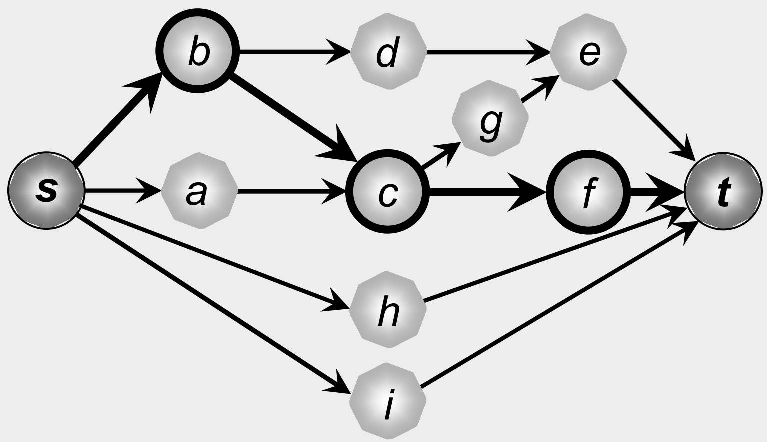
- In a horizontal constraint graph (HCG), node weights represent the widths of the corresponding blocks.
  - Two nodes v_i and v_j, with corresponding blocks m_i and m_j, are connected with a directed edge from v_i to v_j if m_i is to the left of m_j.

- The longest path(s) in the VCG / HCG correspond(s) to the minimum vertical / horizontal floorplan span required to pack the blocks (floorplan height / width).

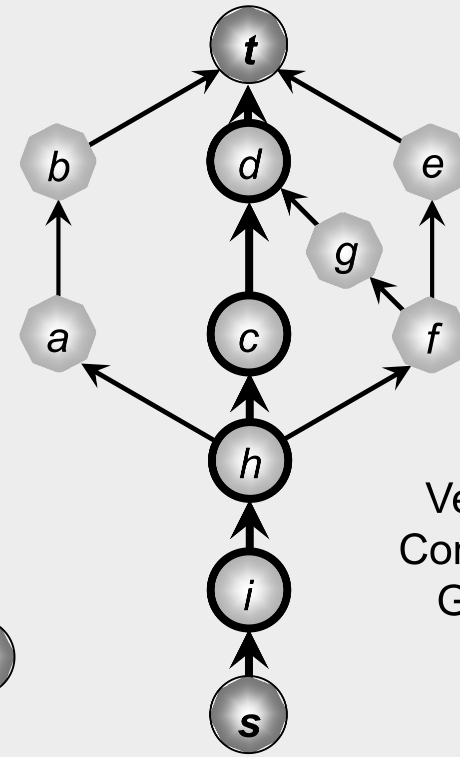
- A constraint-graph pair is a floorplan representation that consists of two directed graphs – *vertical constraint graph* and *horizontal constraint graph* – which capture the relations between block positions.

Constraint graphs





Horizontal Constraint Graph

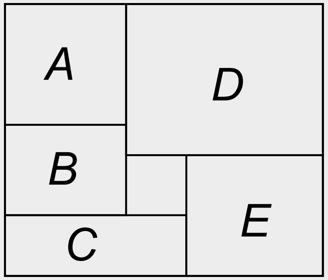


Vertical Constraint Graph

Sequence pair

- Two permutations represent geometric relations between every pair of blocks

- Example: (*ABDCE*, *CBAED*)



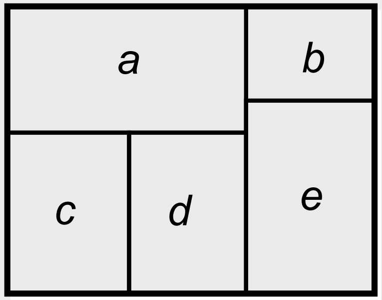
- Horizontal and vertical relations between blocks *A* and *B*:

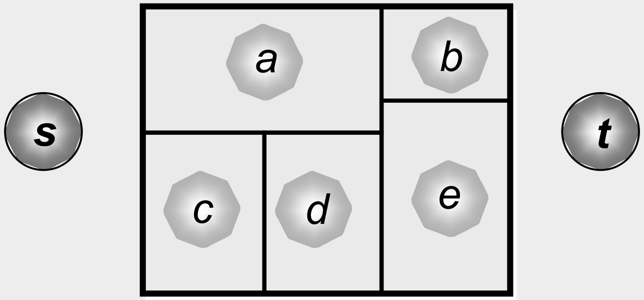
  (… *A* … *B* … , … *A* … *B* …) → *A* is left of *B*
  (… *A* … *B* … , … *B* … *A* …) → *A* is above *B*
  (… *B* … *A* … , … *A* … *B* …) → *A* is below *B*
  (… *B* … *A* … , … *B* … *A* …) → *A* is right of *B*

3.4 Floorplan Representations

3.4.1 Floorplan to a Constraint-Graph Pair

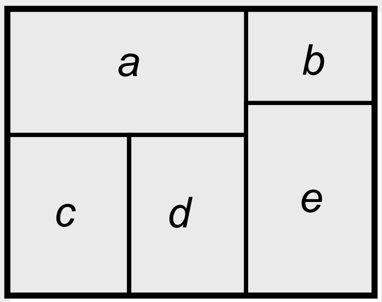
- Create nodes for every block
- In addition, create a source node and a sink one

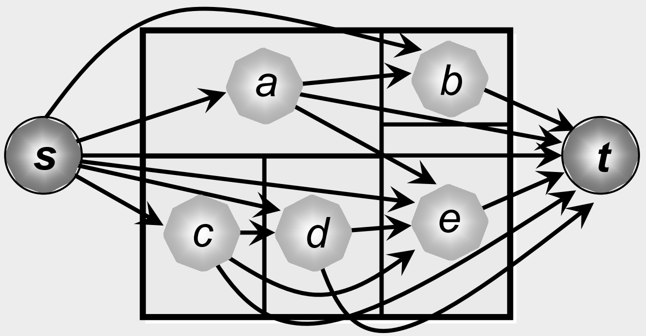




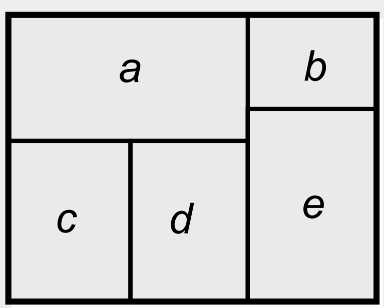
3.4.1 Floorplan to a Constraint-Graph Pair

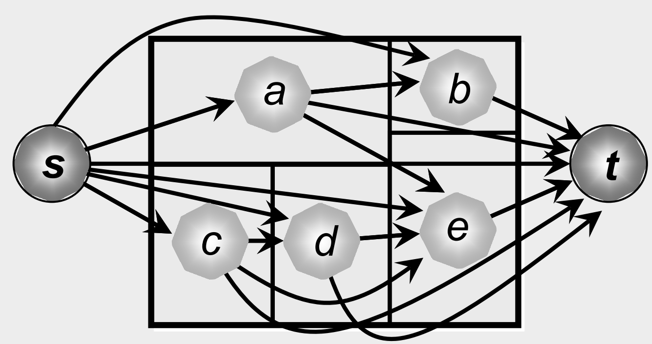
- Create nodes for every block.
- In addition, create a source node and a sink one.
- Add a directed edge (*A*,*B*) if Block *A* is below/left of Block *B*. (HCG)





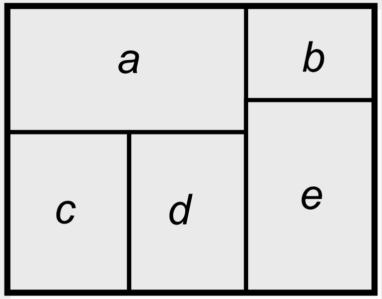
- Create nodes for every block.
- In addition, create a source node and a sink one.
- Add a directed edge (*A*,*B*) if Block *A* is below/left of Block *B*. (HCG)
- Remove the redundant edges
  that can be derived from other edges by transitivity.





3.4.2 Floorplan to a Sequence Pair

- Given two blocks *A* and *B* with
  - Locations: $A = (x_A, y_A)$ and $B = (x_B, y_B)$
  - Dimensions: $A = (w_A, h_A)$ and $B = (w_B, h_B)$
- If $x_A + w_A \leq x_B$ and $!(y_A + h_A \leq y_B$ or $y_B + h_B \leq y_A)$, then *A* is **left of** *B*
- If $y_A + h_A \leq y_B$ and $!(x_A + w_A \leq x_B$ or $x_B + w_B \leq x_A)$, then *A* is **below** *B*



$$S_+ : < acdbe >$$

$$S_- : < cdaeb >$$

3.4.3 Sequence Pair to a Floorplan

- Start with the bottom left corner
- Define a *weighted sequence* as a sequence of blocks based on width
  - Each block B has its own width $w(B)$

- Old (traditional) algorithm: find the longest path through edges ($O(n^2)$)
- Newer approach: find the *longest common subsequence* (*LCS*)
  - Given two weighted sequences S_1 and S_2, the $LCS(S_1,S_2)$ is the longest sequence found in both S_1 and S_2
  - The length of $LCS(S_1,S_2)$ is the sum of weights

- For block placement:
  - $LCS(S_+,S_-)$ returns the x-coordinates of all blocks
  - $LCS(S_+^R,S_-)$ returns the y-coordinates of all blocks (S_+^R is the reverse of S_+)
  - The length of $LCS(S_+,S_-)$ and $LCS(S_+^R,S_-)$ is the width and height, respectively

3.4.3 Sequence Pair to a Floorplan

Algorithm: Longest Common Subsequence (LCS)

Input: sequences S_1 and S_2, weights of n blocks *weights*

Output: positions of each block *positions*, total span L

```
for (i = 1 to n)                                      // initialization
  block_order[S2[i]] = i
  lengths[i] = 0
for (i = 1 to n)
  block = S1[i]                                       // current block
  index = block_order[block]
  positions[block] = lengths[index]                   // compute block position
  t_span = positions[block] + weights[block]          // finds length of sequence
                                                      // from beginning to block
  for (j = index to n)                                // update total length
    if (t_span > lengths[j]) lengths[j] = t_span
    else break
L = lengths[n]                                        // total length is stored here
```

3.4.3 Sequence Pair to a Floorplan

Example: S_1 = <*acdbe*>, S_2 = <*cdaeb*>,
widths[*a b c d e*] = [8 4 4 4 4], *heights*[*a b c d e*] = [4 2 5 5 6]

Find *x*-coordinates – go by S_1's order:

Initial: *block_order*[*a b c d e*] = [3 5 1 2 4], *lengths* = [0 0 0 0 0]

Iteration 1 – block = *a*, index = 3:
positions[*a*] = *lengths*[3] = 0, *t_span* = 8, *lengths* = [0 0 **8 8 8**]

Iteration 2 – block = *c*, index = 1:
positions[*c*] = *lengths*[1] = 0, *t_span* = 4, *lengths* = [**4 4 8 8 8**]

Iteration 3 – block = *d*, index = 2:
positions[*d*] = *lengths*[2] = 4, *t_span* = 8, *lengths* = [4 **8 8 8 8**]

Iteration 4 – block = *b*, index = 5:
positions[*b*] = *lengths*[5] = 8, *t_span* = 12, *lengths* = [4 8 8 8 **12**]

Iteration 5 – block = *e*, index = 4:
positions[*e*] = *lengths*[4] = 8, *t_span* = 12, *lengths* = [4 8 8 **12 12**]

positions[*a b c d e*] = [0 8 0 4 8], *total_width* = *lengths*[*n* = 5] = 12

Example: S_1 = <*acdbe*>, S_2 = <*cdaeb*>,
widths[*a b c d e*] = [8 4 4 4 4], *heights*[*a b c d e*] = [4 2 5 5 6]

Find *y*-coordinates – go by S_1^R's order:

Initial: *block_order*[*a b c d e*] = [3 5 1 2 4], *lengths* = [0 0 0 0 0]

Iteration 1 – block = *e*, index = 4:
positions[*e*] = *lengths*[4] = 0, *t_span* = 6, *lengths* = [0 0 0 **6 6**]

Iteration 2 – block = *b*, index = 5:
positions[*b*] = *lengths*[5] = 6, *t_span* = 9, *lengths* = [0 0 0 6 **9**]

Iteration 3 – block = *d*, index = 2:
positions[*d*] = *lengths*[2] = 0, *t_span* = 5, *lengths* = [0 **5 5 6 9**]

Iteration 4 – block = *c*, index = 1:
positions[*c*] = *lengths*[1] = 0, *t_span* = 5, *lengths* = [**5 5 5 6 9**]

Iteration 5 – block = *a*, index = 3:
positions[*a*] = *lengths*[3] = 5, *t_span* = 9, *lengths* = [5 5 **9 9 9**]

positions[*a b c d e*] = [5 6 0 0 0], *total_height* = *lengths*[*n* = 5] = 9

3.5 Floorplanning Algorithms

Common Goals

- To minimize the total length of interconnect, subject to an upper bound on the floorplan area

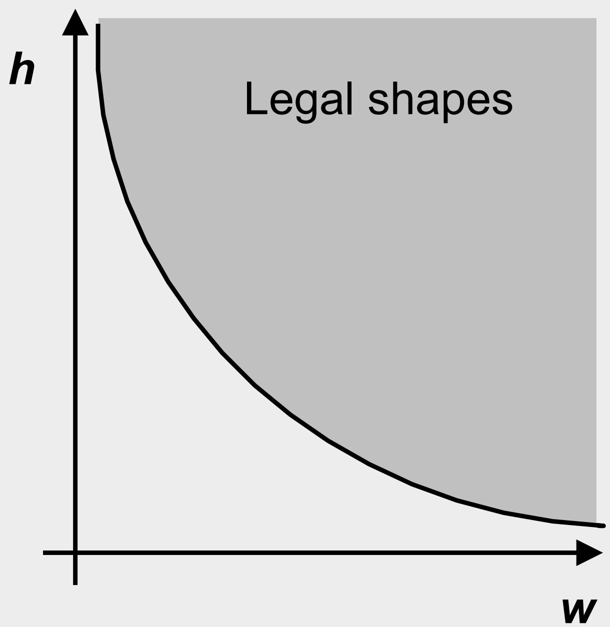
 or

- To simultaneously optimize both wire length and area

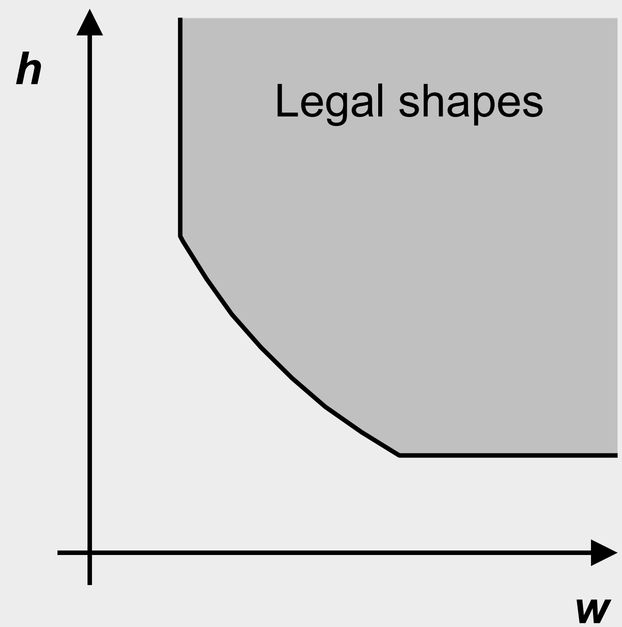
Otten, R.: Efficient Floorplan Optimization. Int. Conf. on Computer Design, 499-502, 1983

3.5.1 Floorplan Sizing

Shape functions



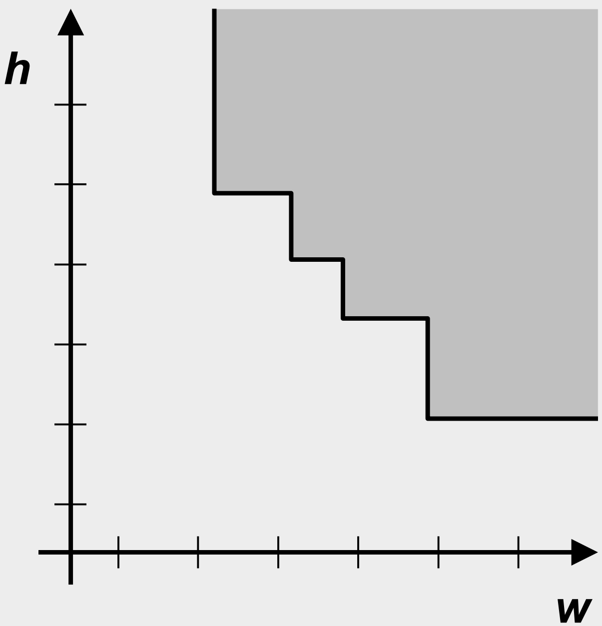
$h * w \geq A$



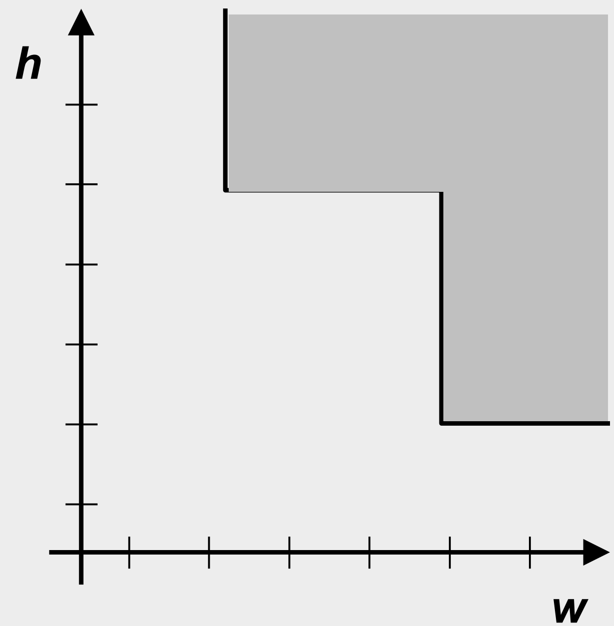
Block with minimum width and height restrictions

Otten, R.: Efficient Floorplan Optimization. Int. Conf. on Computer Design, 499-502, 1983

Shape functions



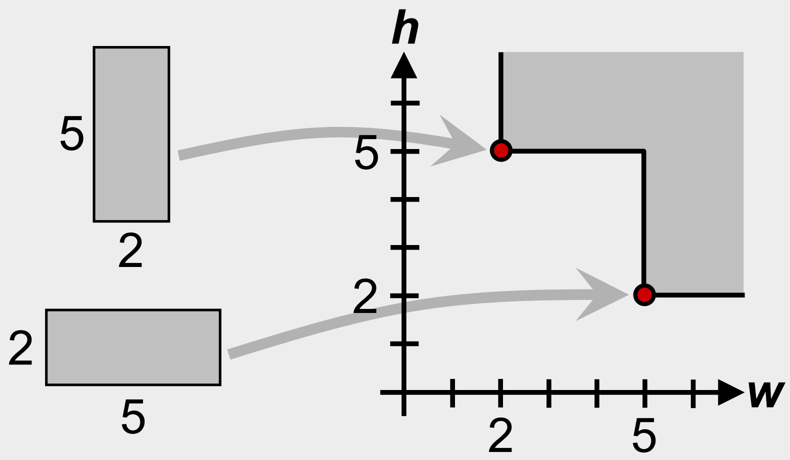
Discrete (h,w) values



Hard library block

Otten, R.: Efficient Floorplan Optimization. Int. Conf. on Computer Design, 499-502, 1983

Corner points



Algorithm

This algorithm finds the **minimum floorplan area** for a given slicing floorplan in polynomial time. For non-slicing floorplans, the problem is NP-hard.

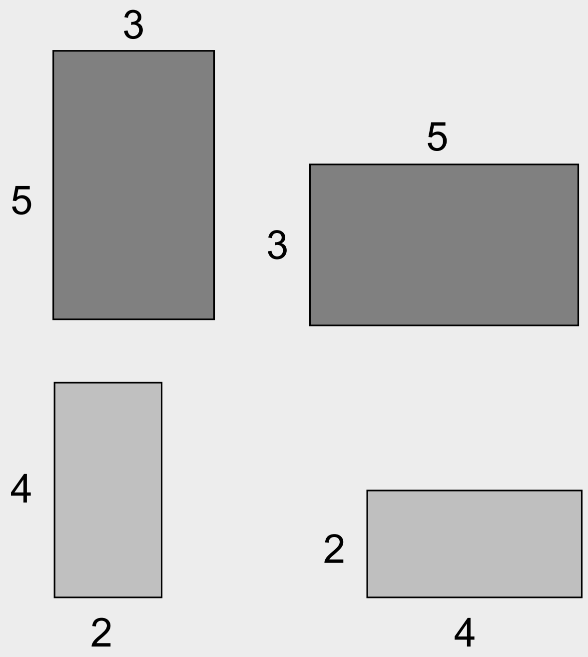
- Construct the shape functions of all individual blocks
- Bottom up: Determine the shape function of the top-level floorplan from the shape functions of the individual blocks
- Top down: From the corner point that corresponds to the minimum top-level floorplan area, trace back to each block's shape function to find that block's dimensions and location.

3.5.1 Floorplan Sizing – Example

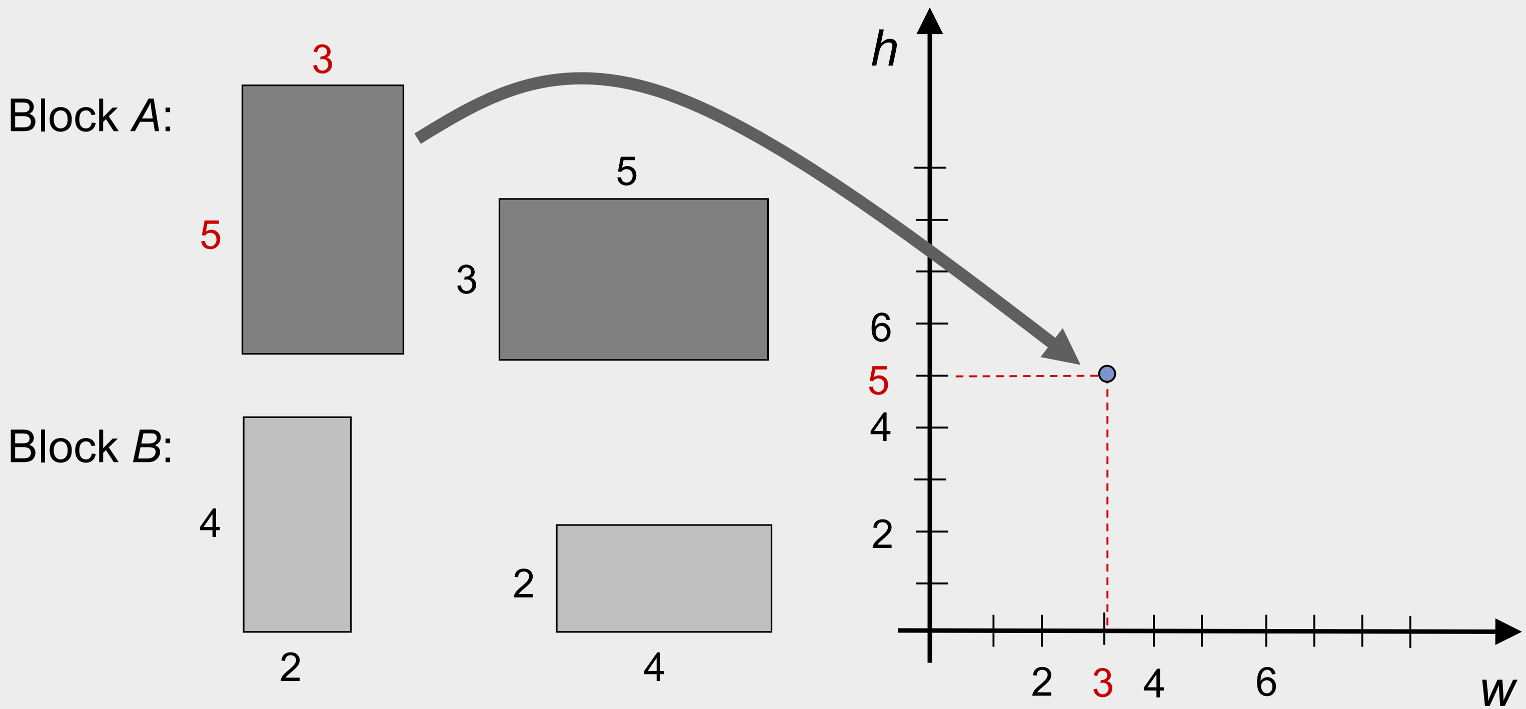
Step 1: Construct the shape functions of the blocks

Block *A*:

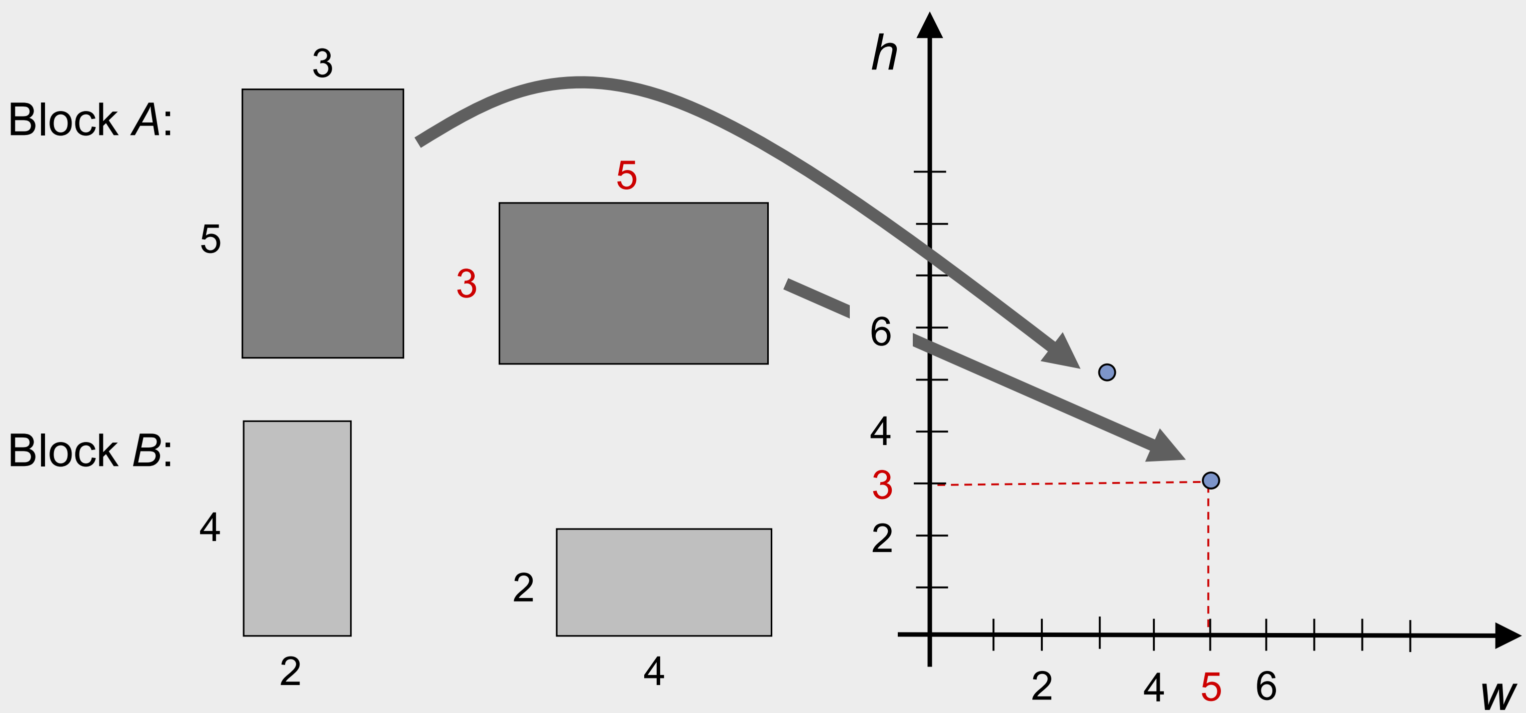
Block *B*:



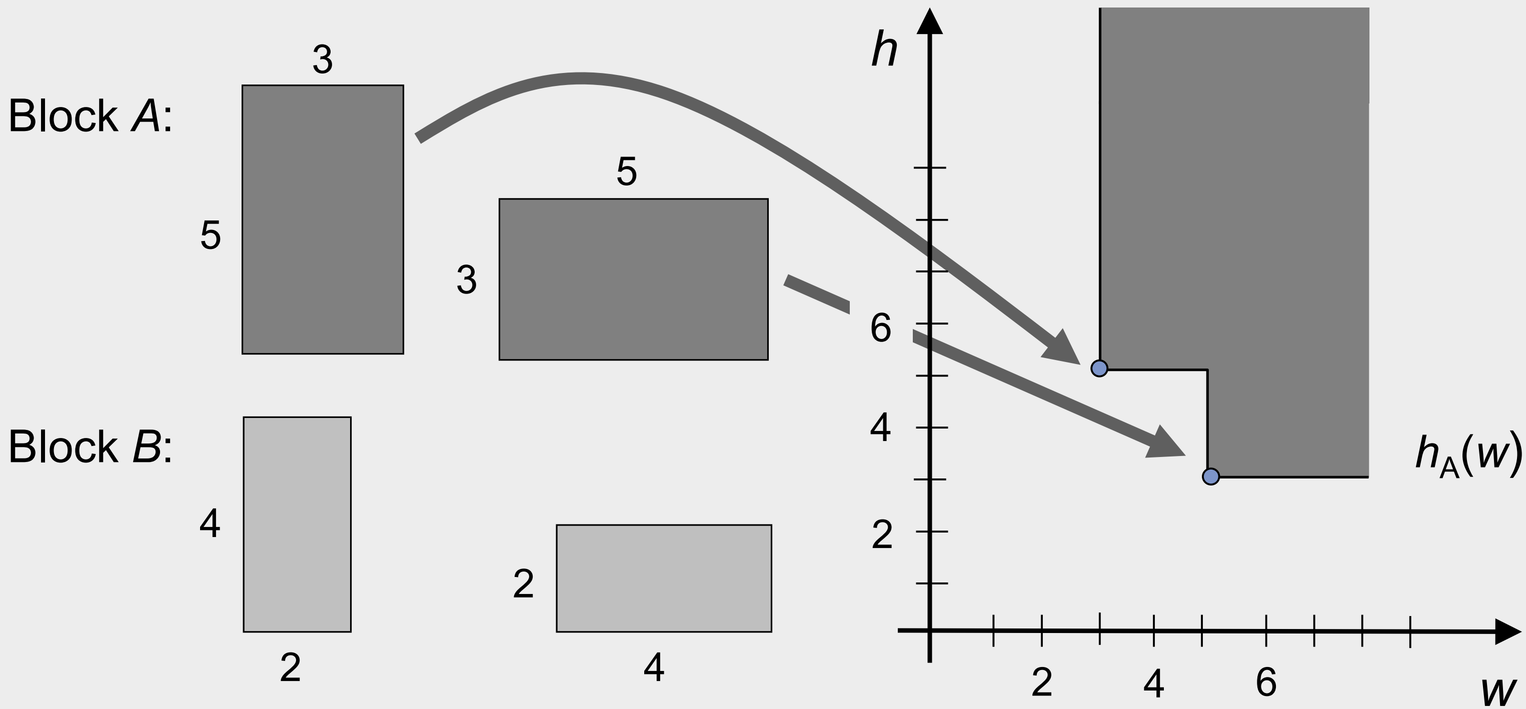
Step 1: Construct the shape functions of the blocks



Step 1: Construct the shape functions of the blocks



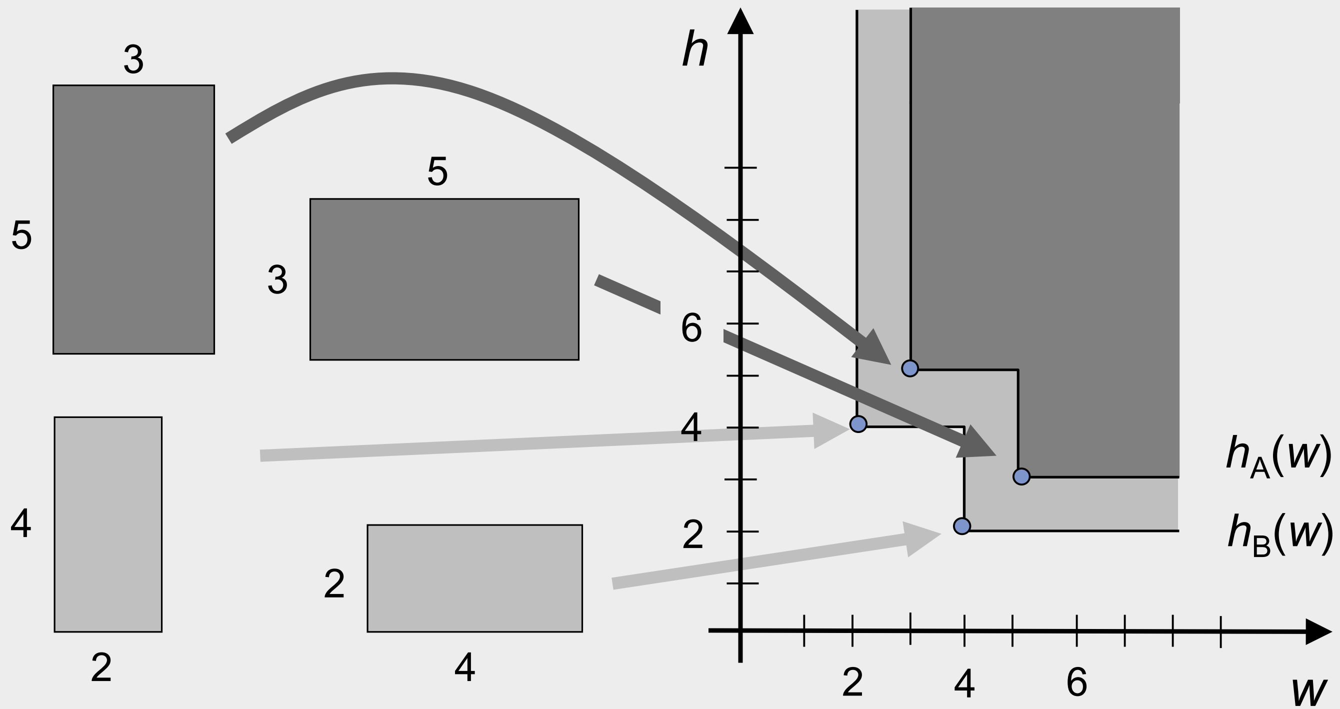
Step 1: Construct the shape functions of the blocks



Step 1: Construct the shape functions of the blocks

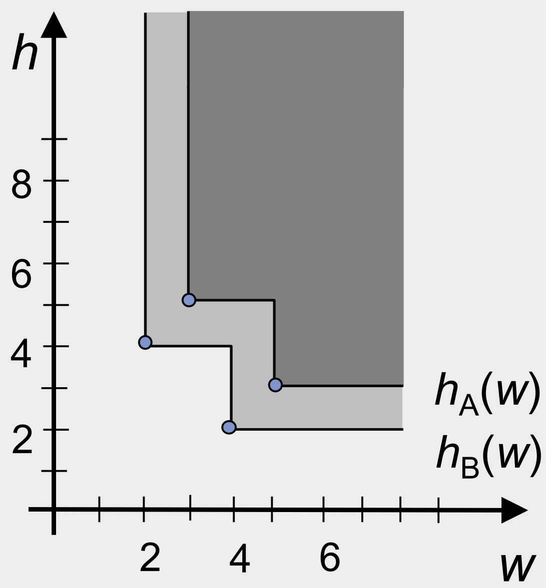
Block *A*:

Block *B*:

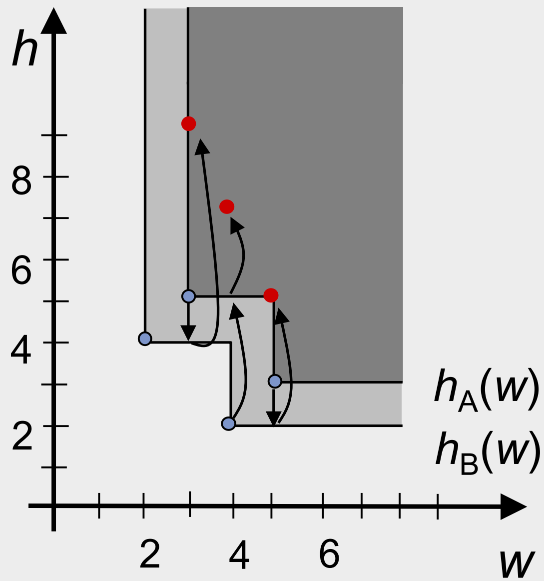


3.5.1 Floorplan Sizing – Example

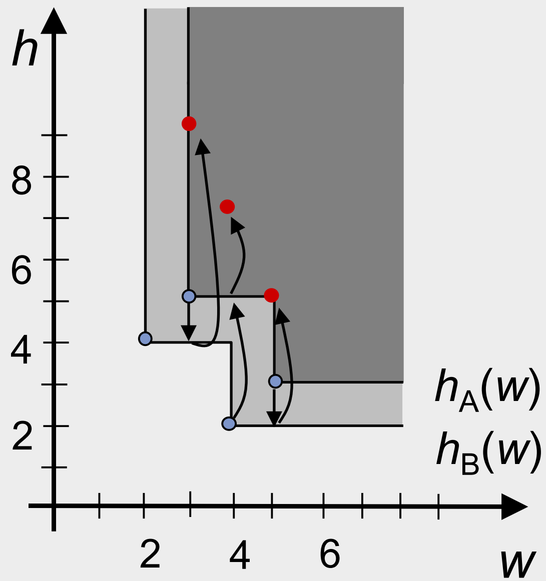
Step 2: Determine the shape function of the top-level floorplan (vertical)

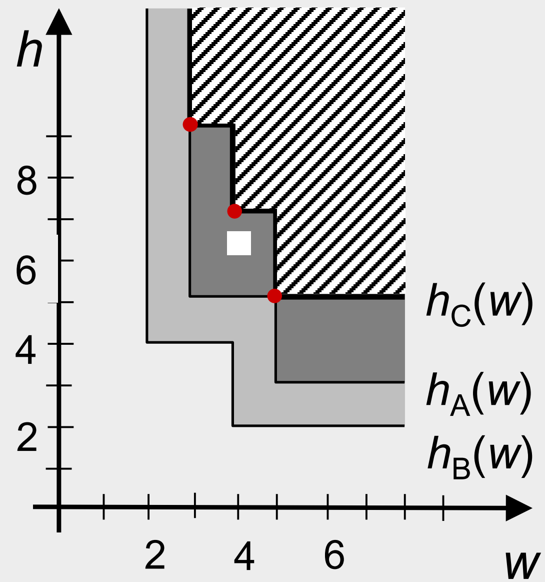


Step 2: Determine the shape function of the top-level floorplan (vertical)

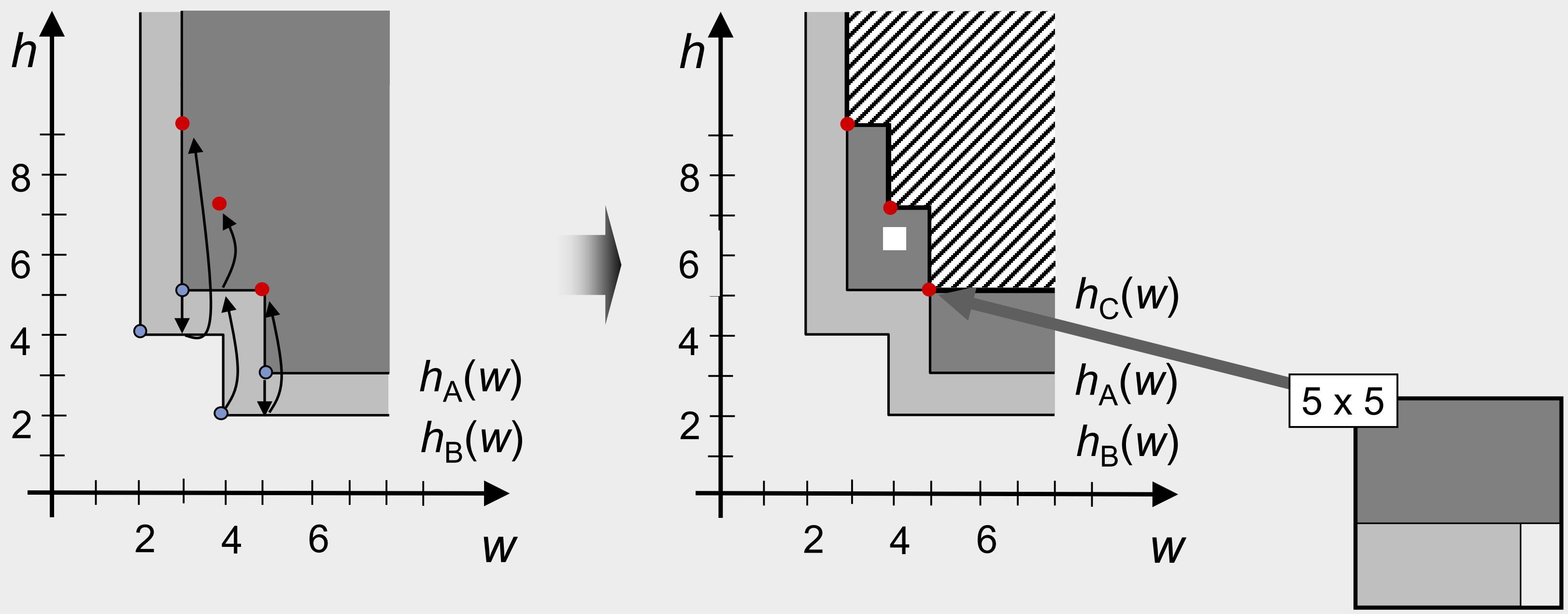


Step 2: Determine the shape function of the top-level floorplan (vertical)

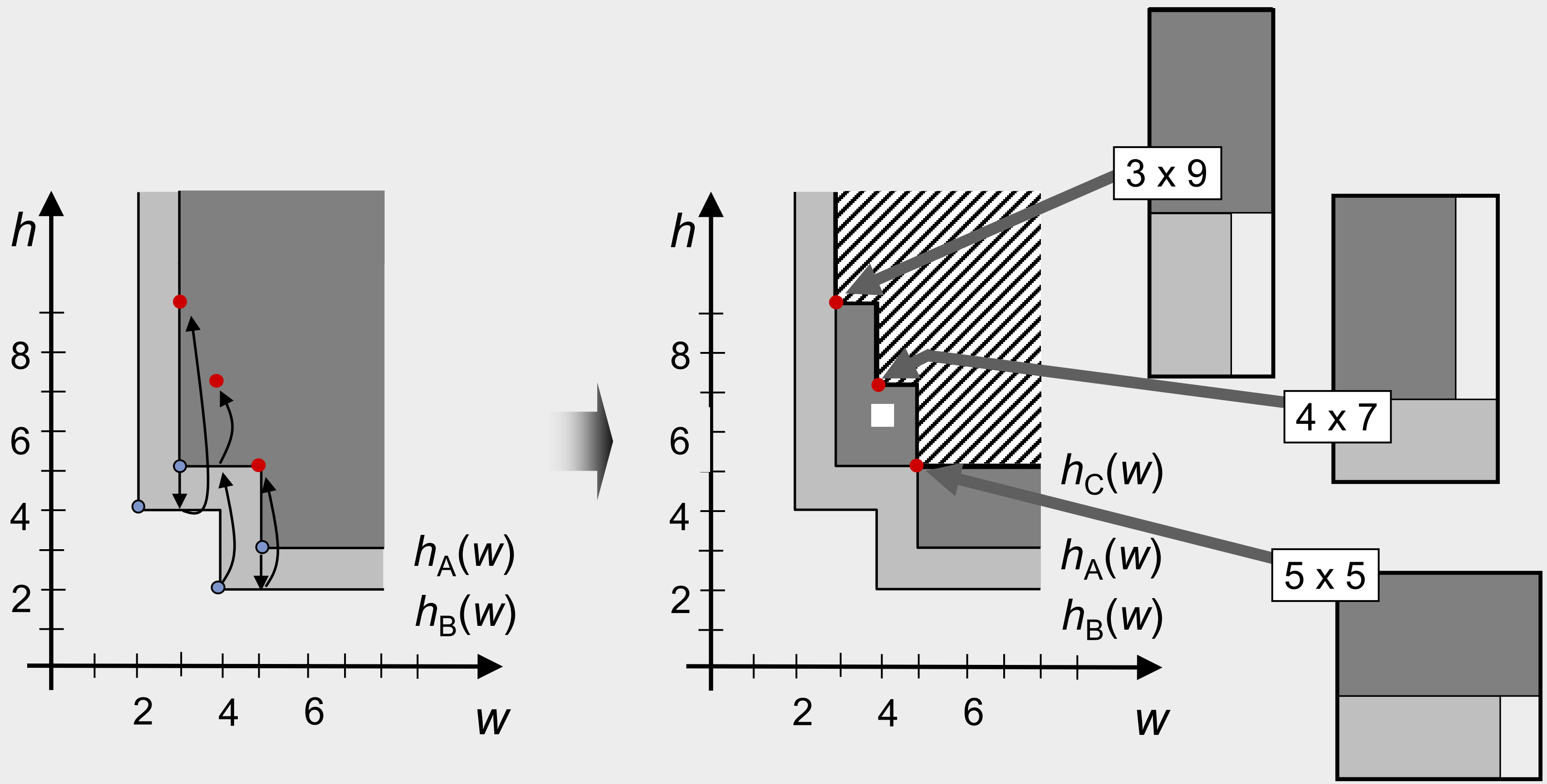




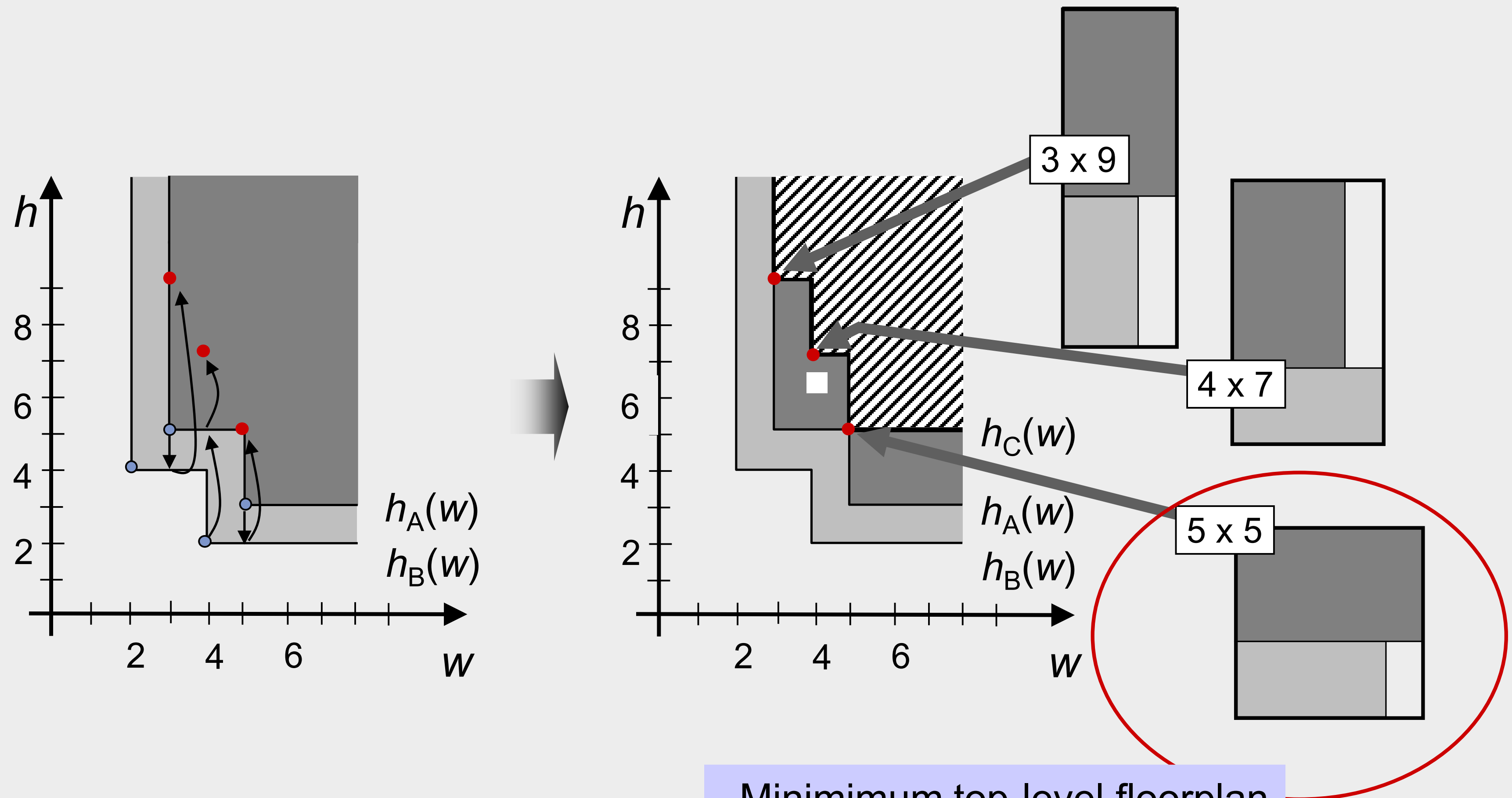
Step 2: Determine the shape function of the top-level floorplan (vertical)



Step 2: Determine the shape function of the top-level floorplan (vertical)



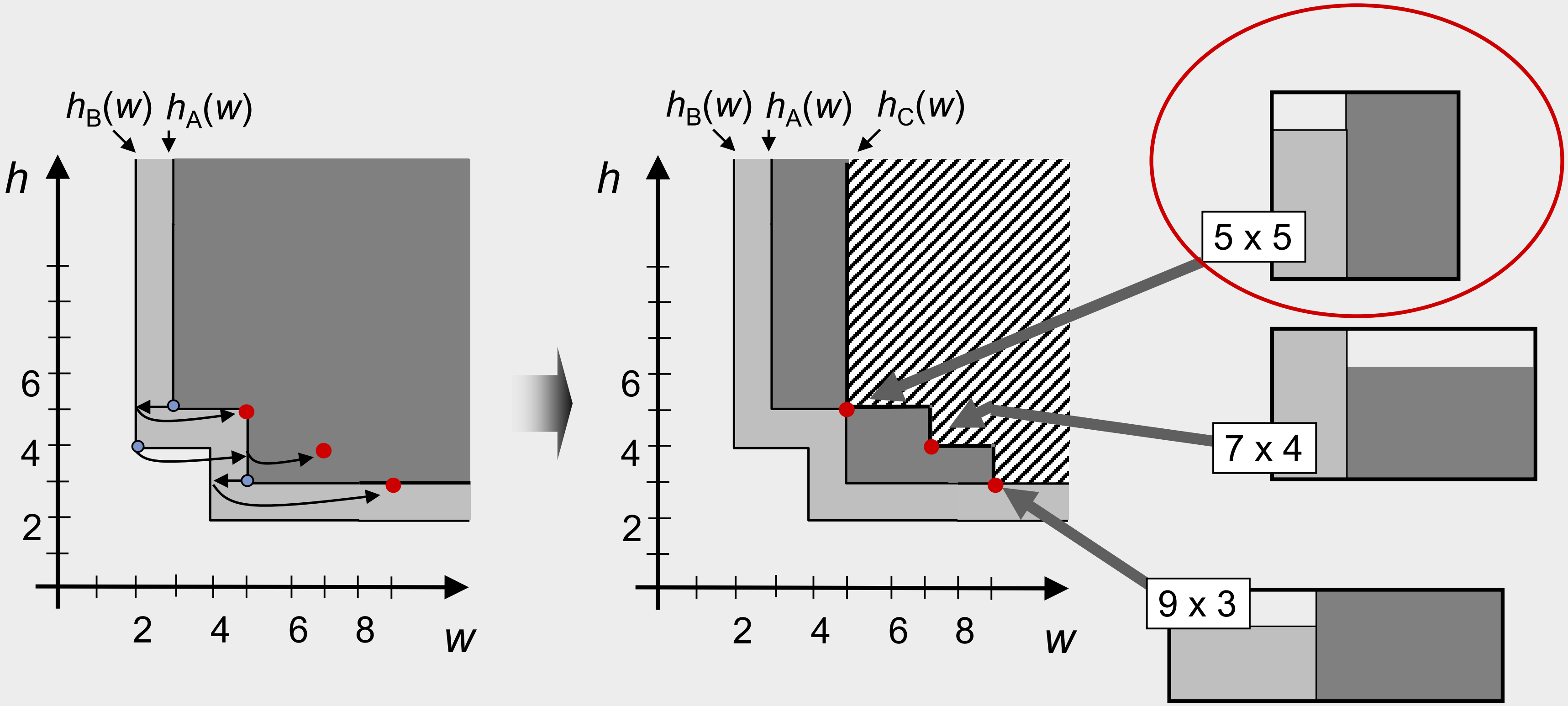
Step 2: Determine the shape function of the top-level floorplan (vertical)



Minimimum top-level floorplan with vertical composition

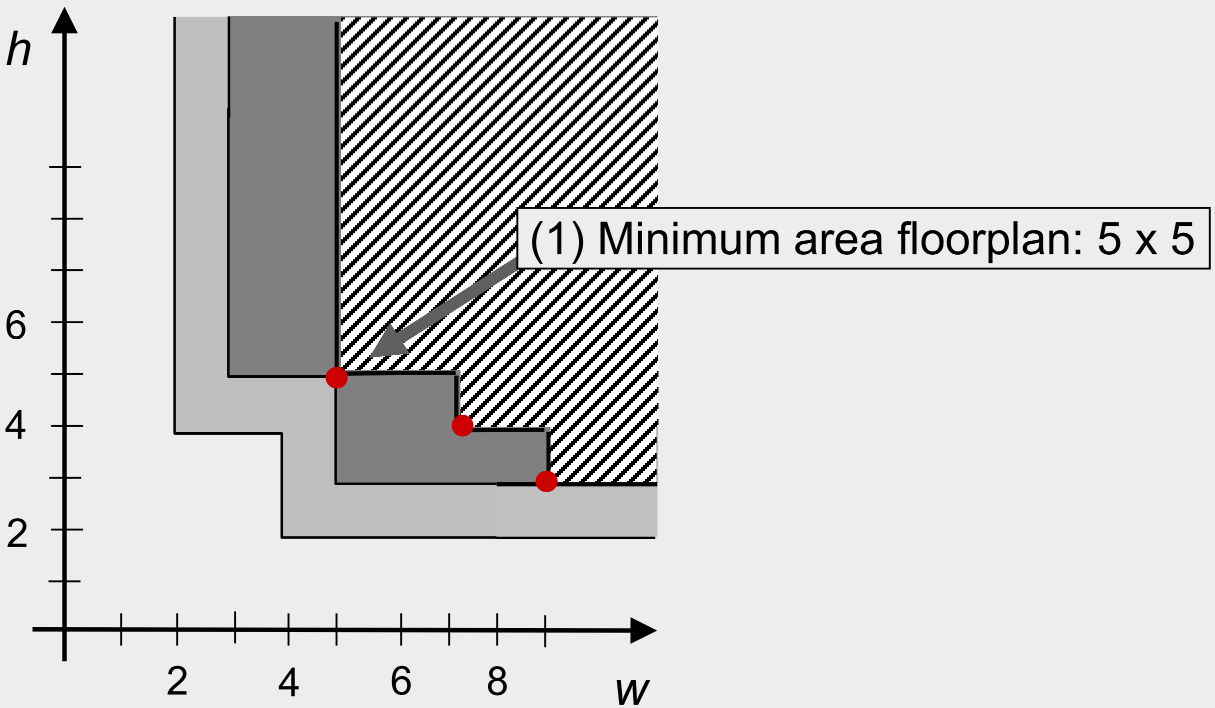
3.5.1 Floorplan Sizing – Example

Step 2: Determine the shape function of the top-level floorplan (horizontal)



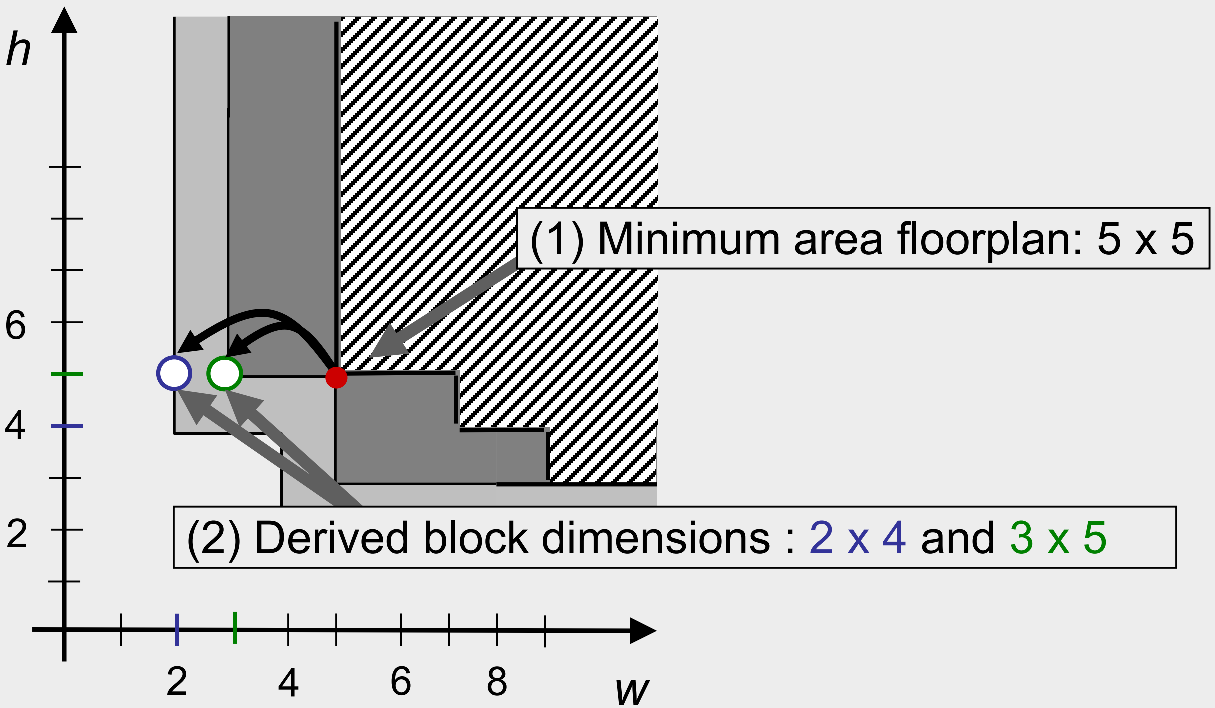
Minimimum top-level floorplan with horizontal composition

Step 3: Find the individual blocks' dimensions and locations



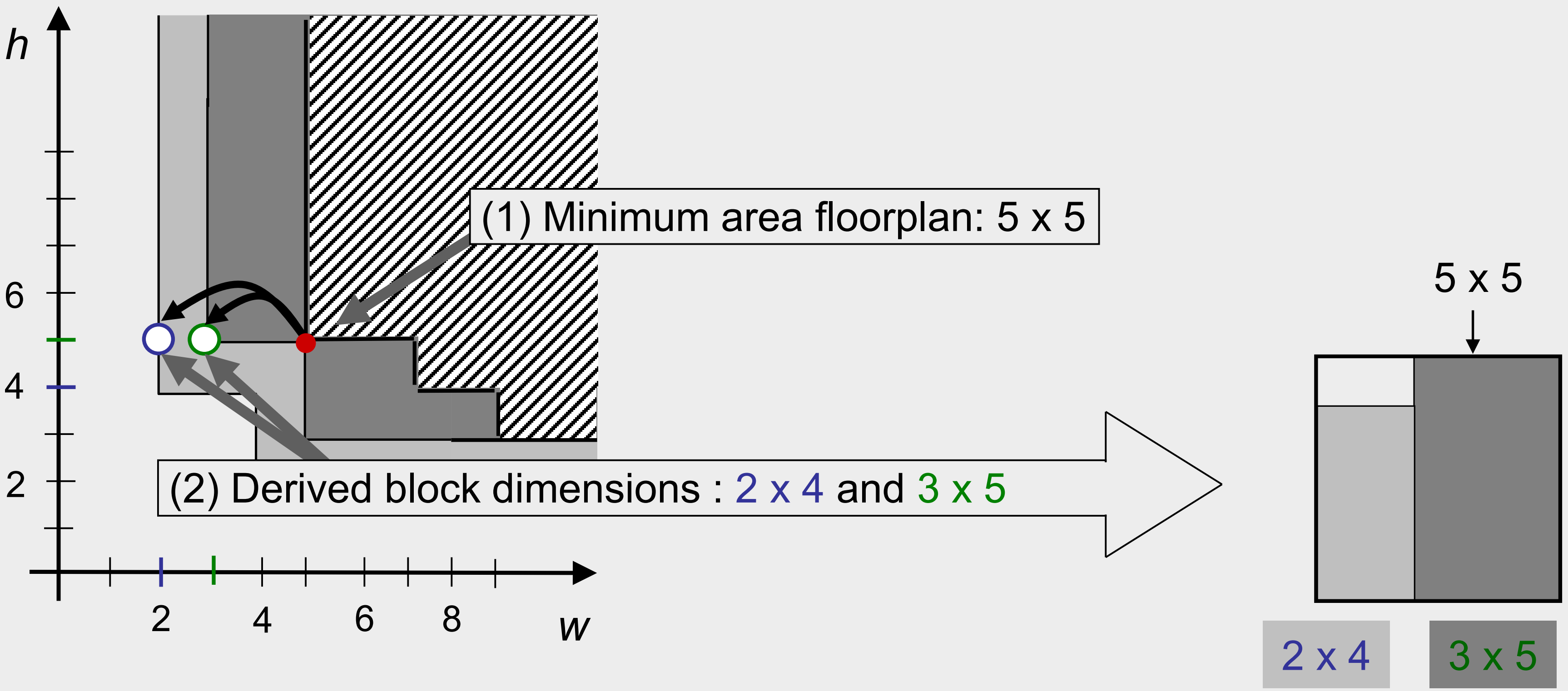
Horizontal composition

Step 3: Find the individual blocks' dimensions and locations



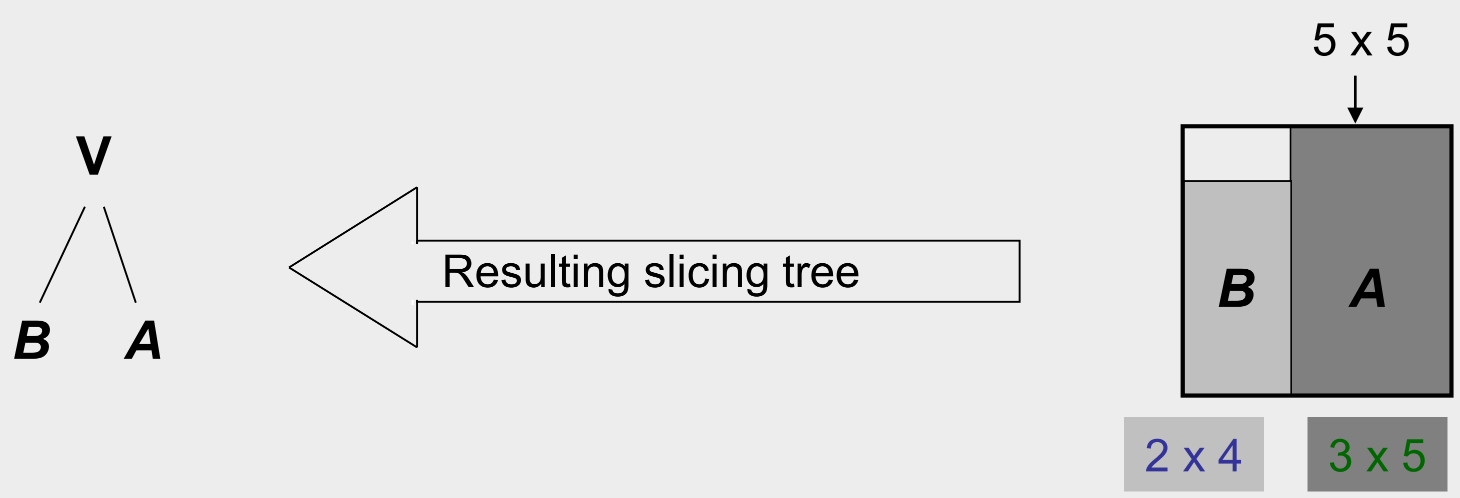
Horizontal composition

Step 3: Find the individual blocks' dimensions and locations

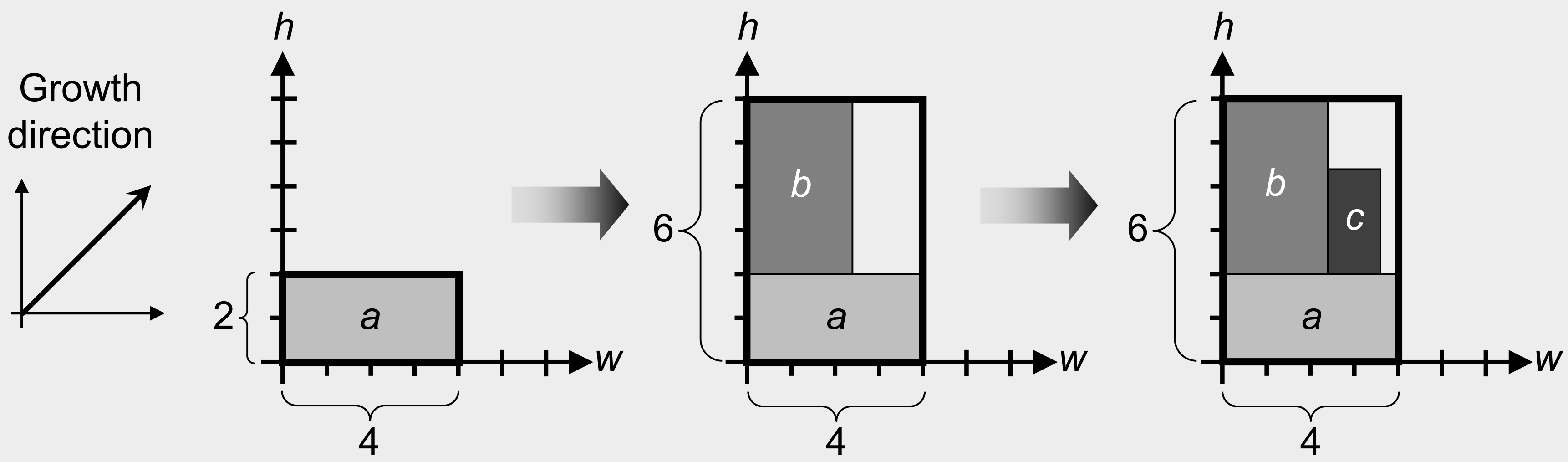


Horizontal composition

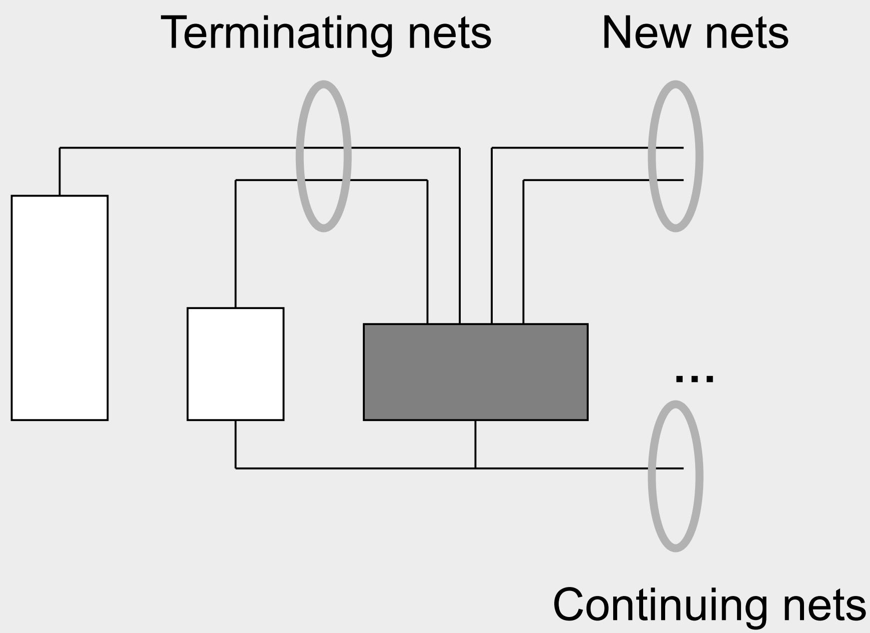
3.5.1 Floorplan Sizing – Example



- Iteratively add blocks to the cluster until all blocks are assigned
- Only the different orientations of the blocks instead of the shape / aspect ratio are taken into account
- Linear ordering to minimize total wirelength of connections between blocks



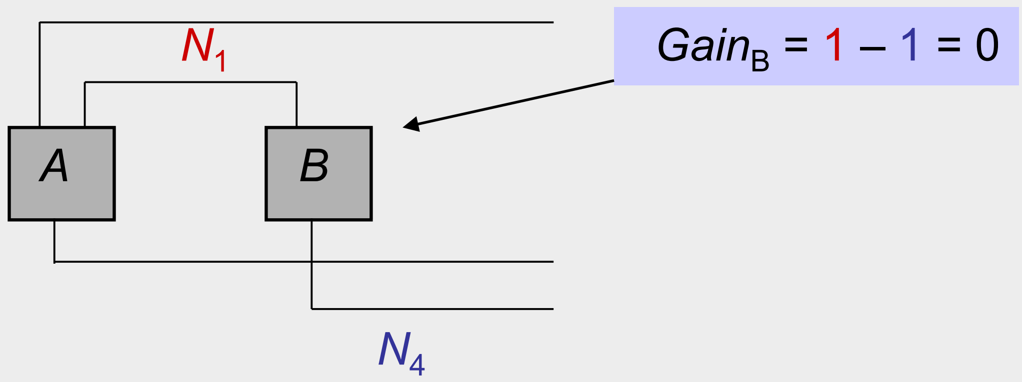
- New nets have no pins on any block from the partially-constructed ordering
- Terminating nets have no other incident blocks that are unplaced
- Continuing nets have at least one pin on a block from the partially-constructed ordering and at least one pin on an unordered block



3.5.2 Cluster Growth – Linear Ordering

- Gain of each block *m* is calculated:

  $Gain_m$ = (Number of terminating nets of *m*) – (New nets of *m*)

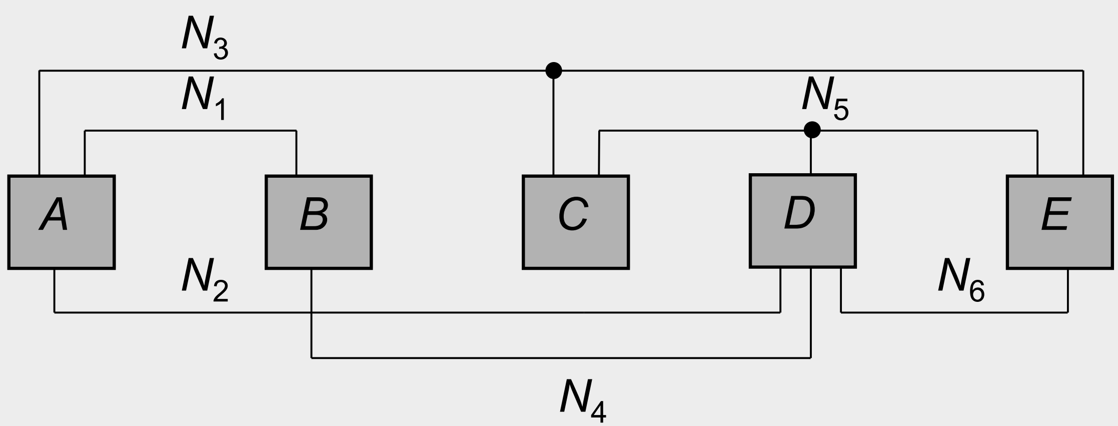


- The block with the maximum gain is selected to be placed next

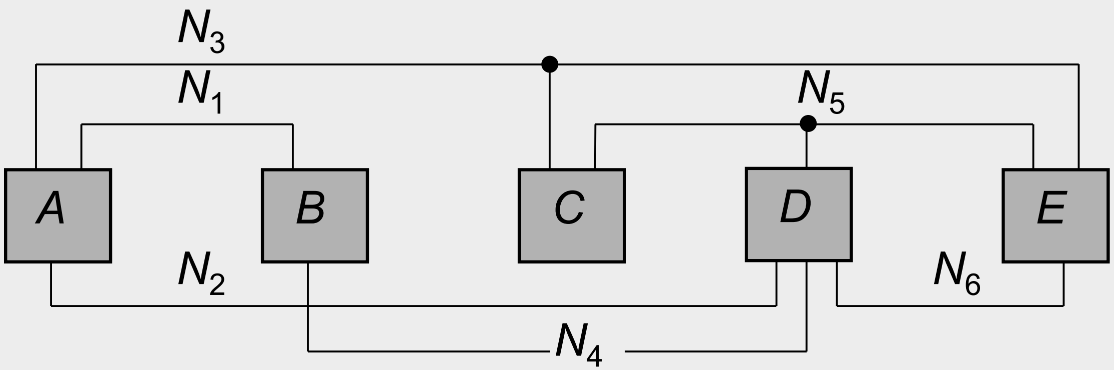
3.5.2 Cluster Growth – Linear Ordering (Example)

Given:

- Netlist with five blocks *A, B, C, D, E* and six nets
  $N_1 = \{A, B\}$
  $N_2 = \{A, D\}$
  $N_3 = \{A, C, E\}$
  $N_4 = \{B, D\}$
  $N_5 = \{C, D, E\}$
  $N_6 = \{D, E\}$
- Initial block: *A*



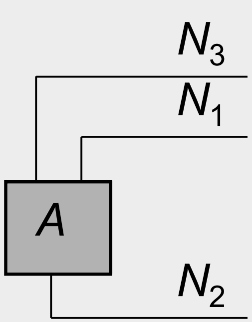
Task: Linear ordering with minimum netlength

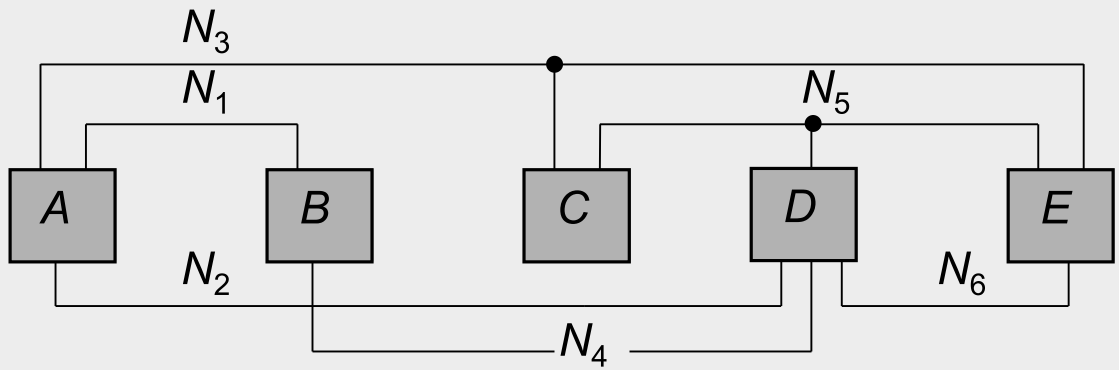


Iteration #	Block	New Nets	Terminating Nets	Gain	Continuing Nets
0	***A***	N_1, N_2, N_3	--	-3	--

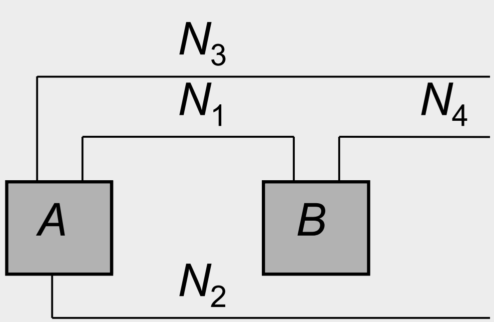
Initial block

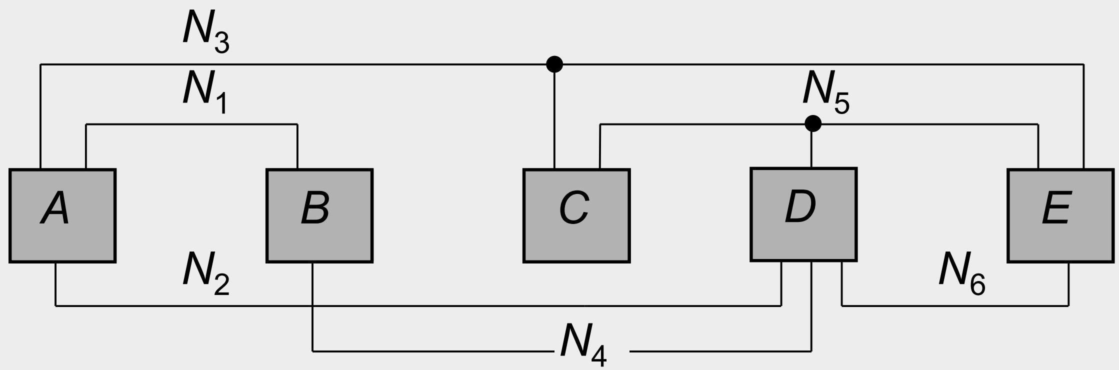
$Gain_A$ = (Number of terminating nets of A) – (New nets of A)



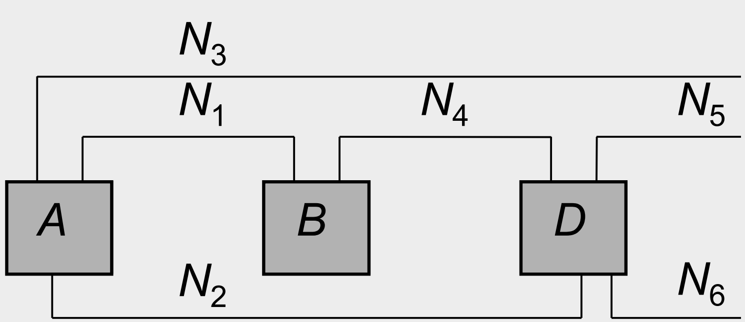


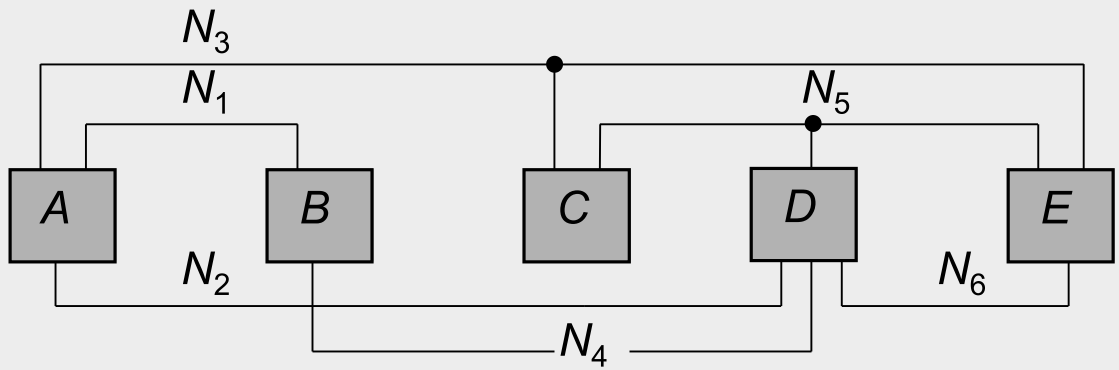
Iteration #	Block	New Nets	Terminating Nets	Gain	Continuing Nets
0	**A**	N_1,N_2,N_3	--	-3	--
1	**B**	N_4	N_1	0	--
	C	N_5	--	-1	N_3
	D	N_4,N_5,N_6	N_2	-2	--
	E	N_5,N_6	--	-2	N_3





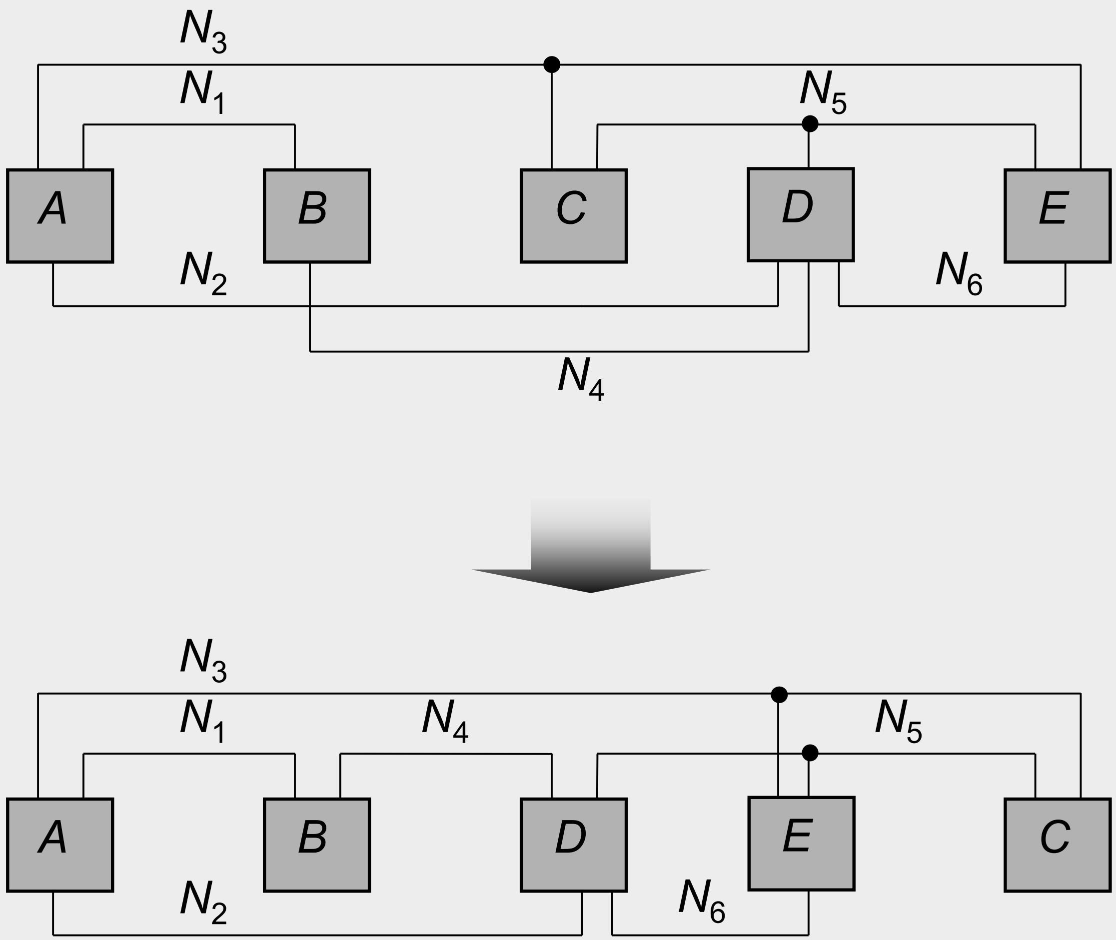
Iteration #	Block	New Nets	Terminating Nets	Gain	Continuing Nets
0	**A**	N_1,N_2,N_3	--	-3	--
1	**B**	N_4	N_1	0	--
	C	N_5	--	-1	N_3
	D	N_4,N_5,N_6	N_2	-2	--
	E	N_5,N_6	--	-2	N_3
2	C	N_5	--	-1	N_3
	D	N_5,N_6	N_2,N_4	0	--
	E	N_5,N_6	--	-2	N_3





Iteration #	Block	New Nets	Terminating Nets	Gain	Continuing Nets
0	***A***	N_1,N_2,N_3	--	-3	--
1	***B***	N_4	N_1	0	--
	C	N_5	--	-1	N_3
	D	N_4,N_5,N_6	N_2	-2	--
	E	N_5,N_6	--	-2	N_3
2	*C*	N_5	--	-1	N_3
	D	N_5,N_6	N_2,N_4	0	--
	E	N_5,N_6	--	-2	N_3
3	*C*	--	--	0	N_3,N_5
	E	--	N_6	1	N_3,N_5
4	***C***	--	N_3,N_5	2	--

3.5.2 Cluster Growth – Linear Ordering (Example)



3.5.2 Cluster Growth – Algorithm

Input: set of all blocks *M*, cost function *C*

Output: optimized floorplan *F* based on *C*

```
F = Ø
order = LINEAR_ORDERING(M)                    // generate linear ordering
for (i = 1 to |order|)
    curr_block = order[i]
    ADD_TO_FLOORPLAN(F,curr_block,C)          // find location and orientation
                                              // of curr_block that causes
                                              // smallest increase based on
                                              // C while obeying constraints
```

3.5.2 Cluster Growth

Analysis

- The objective is to minimize the total wirelength of connections blocks
- Though this produces mediocre solutions, the algorithm is easy to implement and fast.
- Can be used to find the initial floorplan solutions for iterative algorithms such as *simulated annealing*.

3.5.3 Simulated Annealing

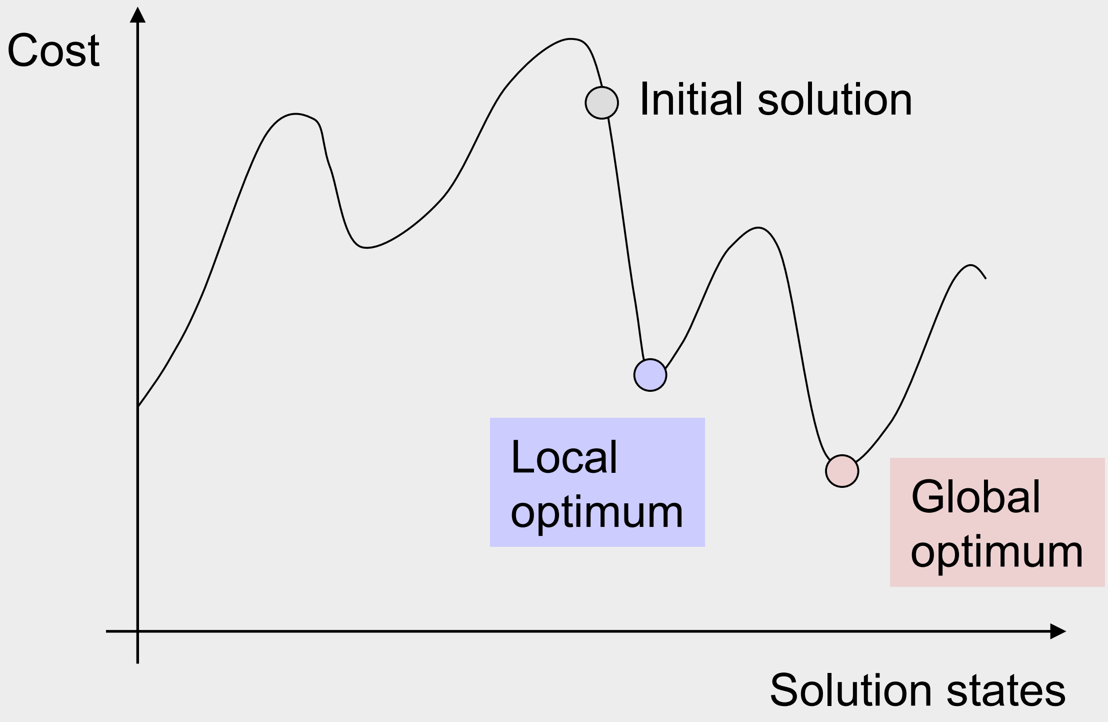
Introduction

- Simulated Annealing (SA) algorithms are iterative in nature.

- Begins with an initial (arbitrary) solution and seeks to incrementally improve the objective function.

- During each iteration, a local neighborhood of the current solution is considered. A new candidate solution is formed by a small perturbation of the current solution.

- Unlike greedy algorithms, SA algorithms can accept candidate solutions with higher cost.


Cost
Initial solution
Local optimum
Global optimum
Solution states

3.5.3 Simulated Annealing

What is annealing?

- Definition (from material science): controlled cooling process of high-temperature materials to modify their properties.
- Cooling changes material structure from being highly randomized (chaotic) to being structured (stable).
- The way that atoms settle in low-temperature state is probabilistic in nature.

- Slower cooling has a higher probability of achieving a perfect lattice with minimum-energy
  - Cooling process occurs in steps
  - Atoms need enough time to try different structures
  - Sometimes, atoms may move across larger distances and create (intermediate) higher-energy states
  - Probability of the accepting higher-energy states decreases with temperature

Simulated Annealing

- Generate an initial solution S_{init}, and evaluate its cost.
- Generate a new solution S_{new} by performing a random walk
- S_{new} is accepted or rejected based on the temperature T
  - Higher T means a higher probability to accept S_{new} if $COST(S_{new}) > COST(S_{init})$
  - T slowly decreases to form the final solution
- Boltzmann acceptance criterion, where r is a random number [0,1)

$$e^{\frac{COST(S_{init}) - COST(S_{new})}{T}} > r$$

Simulated Annealing

- Generate an initial solution and evaluate its cost
- Generate a new solution by performing a random walk
- Solution is accepted or rejected based on a temperature parameter T
- Higher T indicates higher probability to accept a solution with higher cost
- T slowly decreases to form the finalized solution.

- Boltzmann acceptance criterion:

$$e^{-\frac{cost(curr_{sol})-cost(next_{sol})}{T}} > r$$

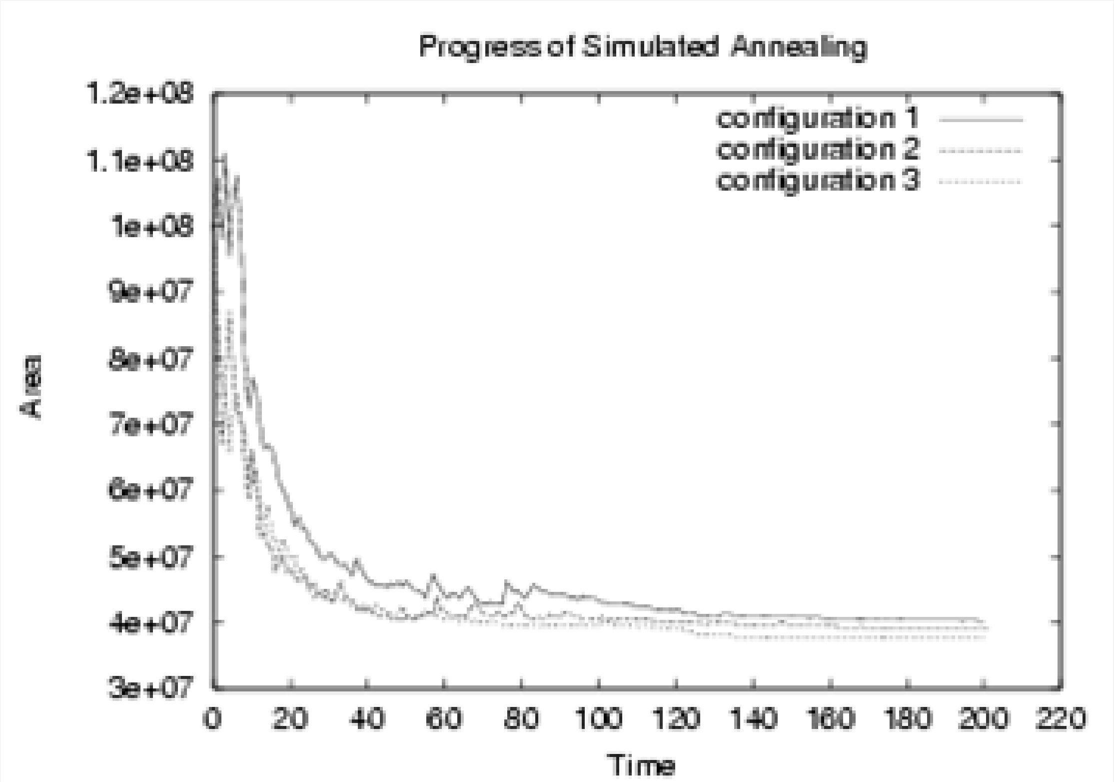
$curr_{sol}$: current solution

$next_{sol}$: new solution after perturbation

T: current temperature

r: random number between[0,1) from normal distr.

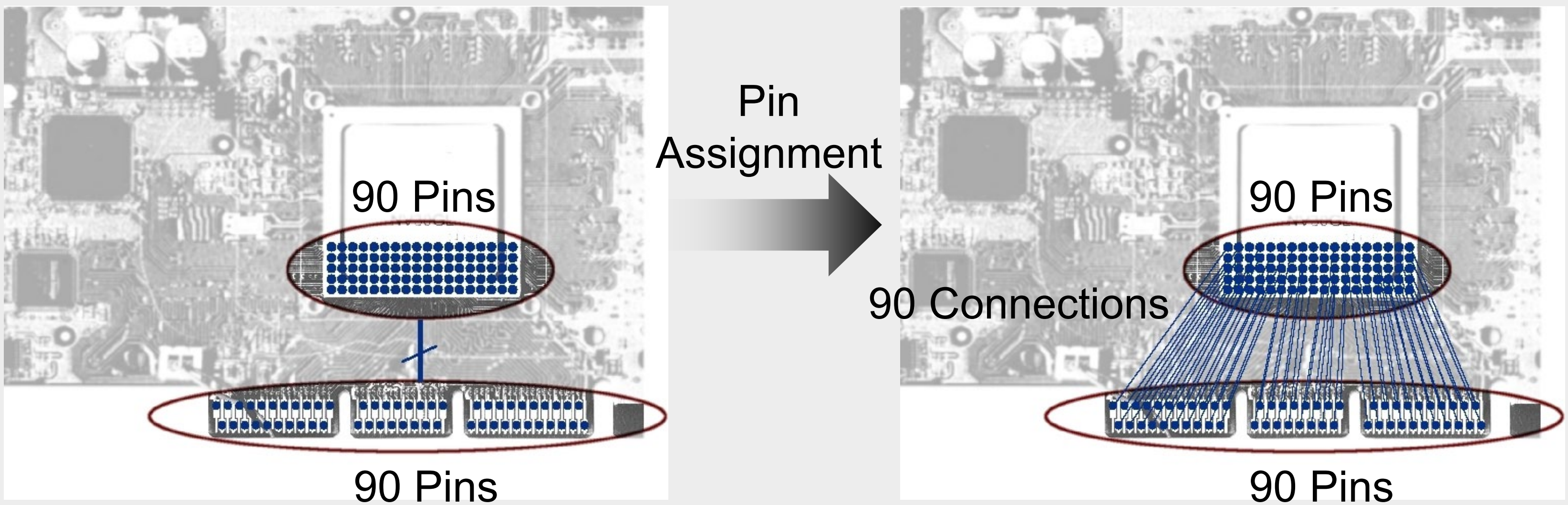
3.5.3 Simulated Annealing – Algorithm



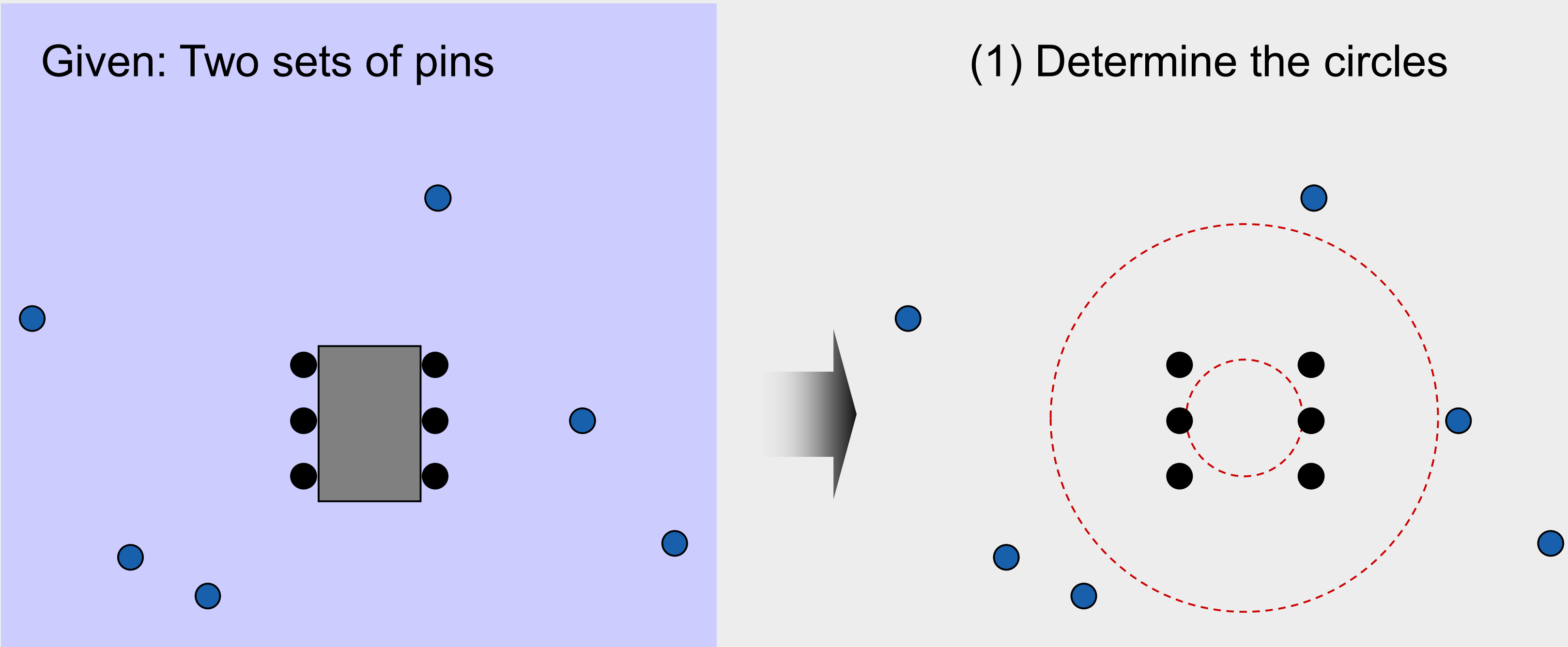
Input: initial solution *init_sol*
Output: optimized new solution *curr_sol*

```
T = T_0                                         // initialization
i = 0
curr_sol = init_sol
curr_cost = COST(curr_sol)
while (T > T_min)
    while (stopping criterion is not met)
     i = i + 1
     (a_i,b_i) = SELECT_PAIR(curr_sol)          // select two objects to perturb
     trial_sol = TRY_MOVE(a_i,b_i)              // try small local change
     trial_cost = COST(trial_sol)
     Δcost = trial_cost – curr_cost
     if (Δcost < 0)                             // if there is improvement,
               curr_cost = trial_cost           //  update the cost and
               curr_sol = MOVE(a_i,b_i)         //  execute the move
     else
               r = RANDOM(0,1)                  // random number [0,1]
               if (r < e^(–Δcost/T))            // if it meets threshold,
                    curr_cost = trial_cost      //   update the cost and
                    curr_sol = MOVE(a_i,b_i)    //   execute the move
     T = α · T                                  // 0 < α < 1, T reduction
```

During pin assignment, all nets (signals) are assigned to unique pin locations such that the overall design performance is optimized.

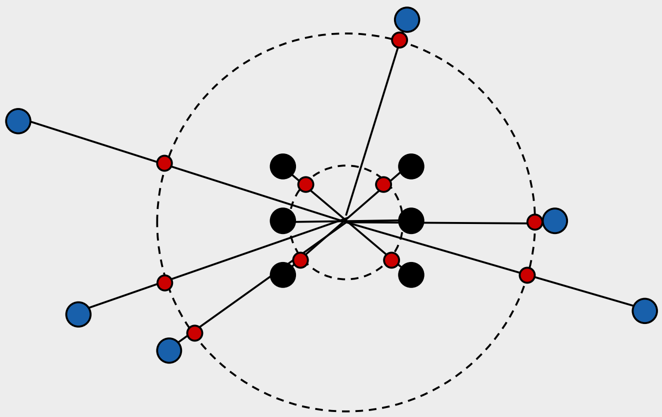


Pin Assignment – Example



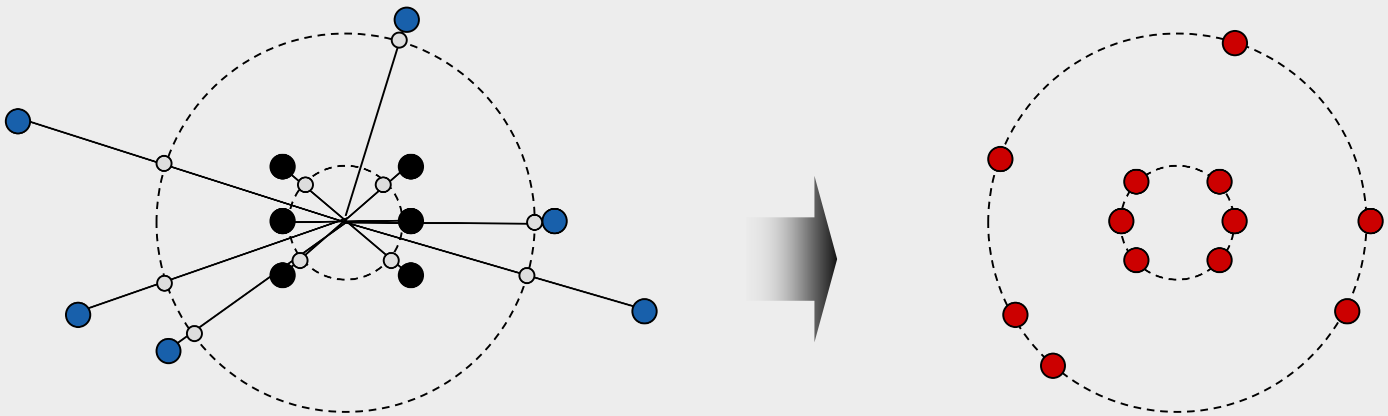
Koren, N. L.: Pin Assignment in Automated Printed Circuit Boards

(2) Determine the points



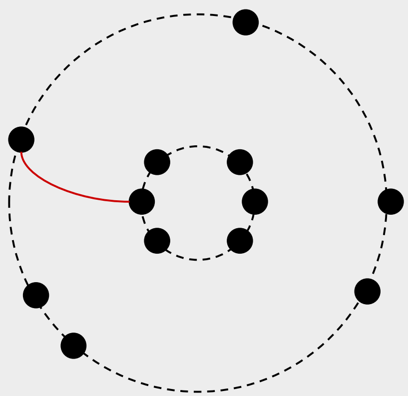
Koren, N. L.: Pin Assignment in Automated Printed Circuit Boards

(2) Determine the points

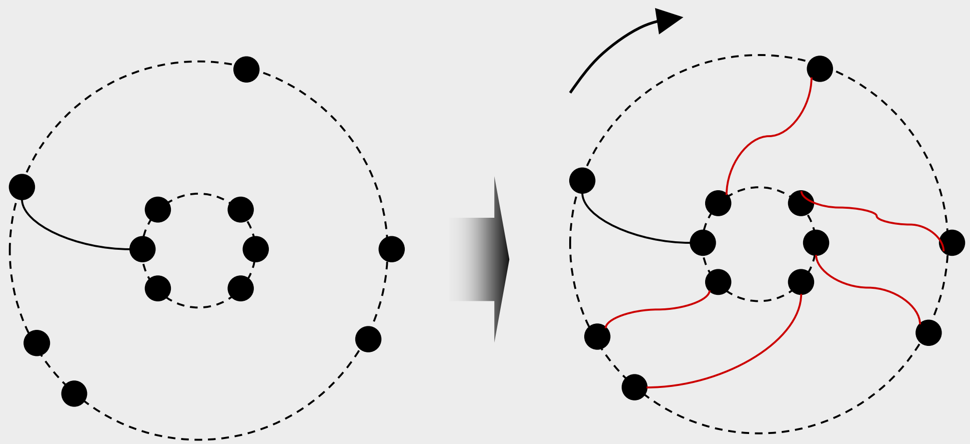


Koren, N. L.: Pin Assignment in Automated Printed Circuit Boards

(3) Determine initial mapping

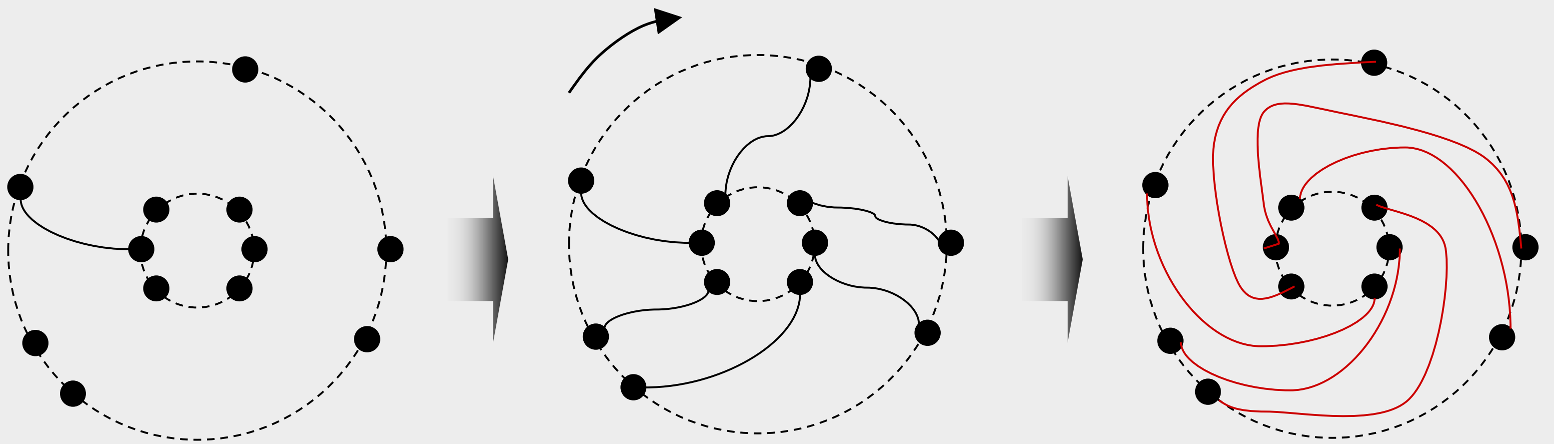


(3) Determine initial mapping and (4) optimize the mapping (complete rotation)



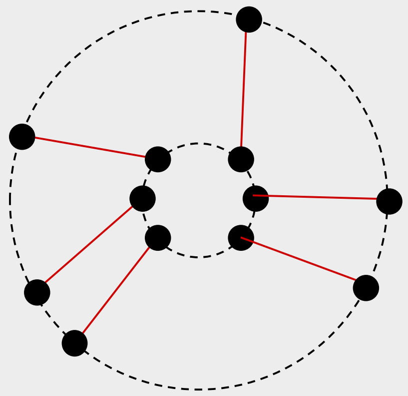
Koren, N. L.: Pin Assignment in Automated Printed Circuit Boards

(3) Determine initial mapping and (4) optimize the mapping (complete rotation)



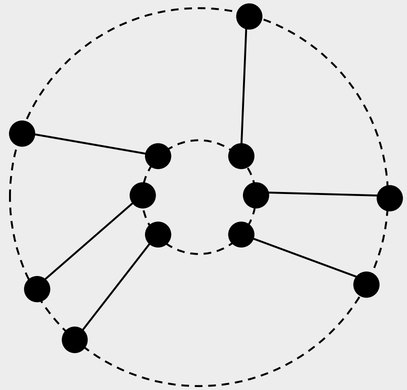
Koren, N. L.: Pin Assignment in Automated Printed Circuit Boards

(4) Best mapping (shortest Euclidean distance)

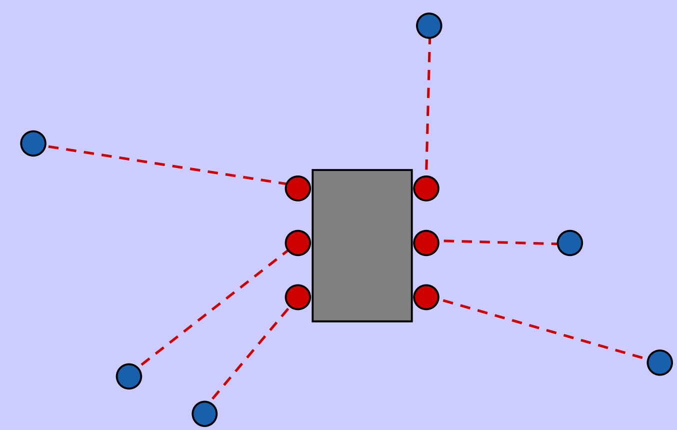


Koren, N. L.: Pin Assignment in Automated Printed Circuit Boards

(4) Best mapping

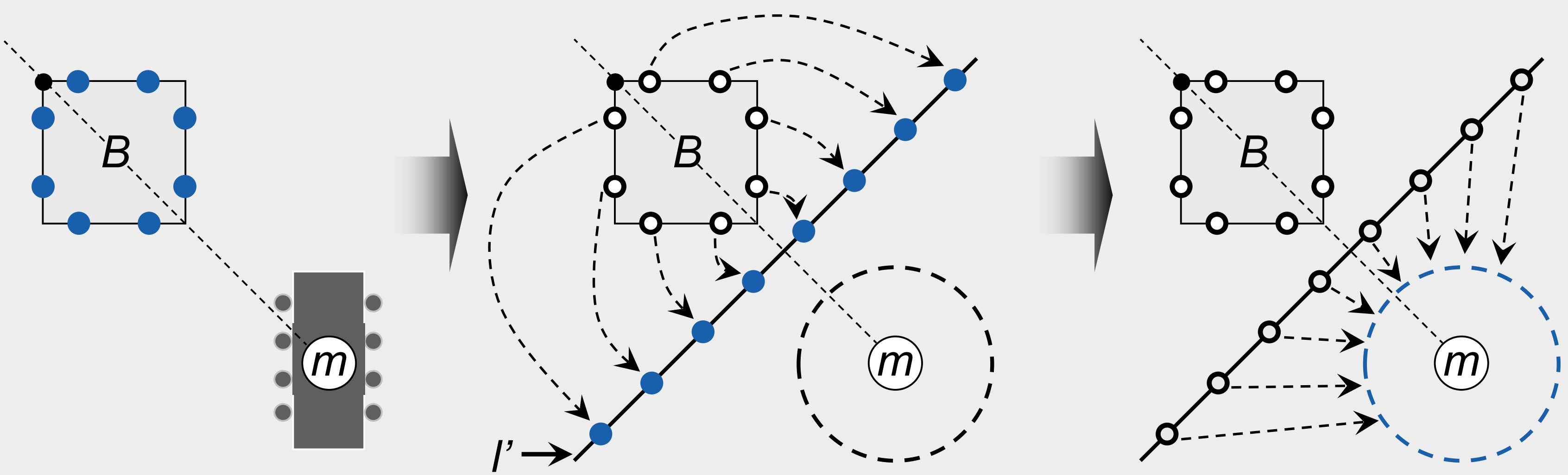


Final pin assignment



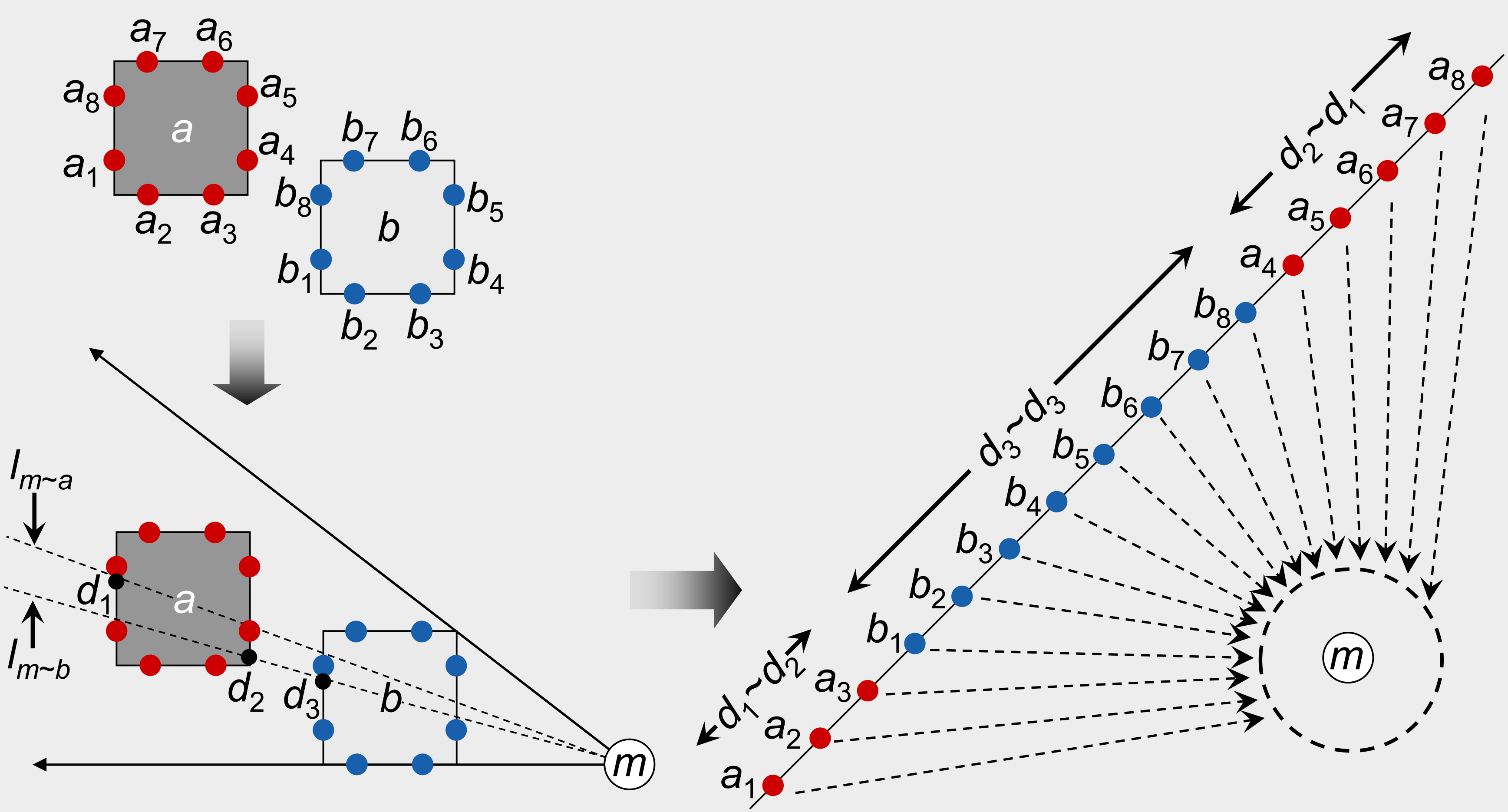
Koren, N. L.: Pin Assignment in Automated Printed Circuit Boards

Pin assignment to an external block *B*



H. N. Brady, "An Approach to Topological Pin Assignment", IEEE Trans. on CAD 3(3) (1984), pp. 250-255

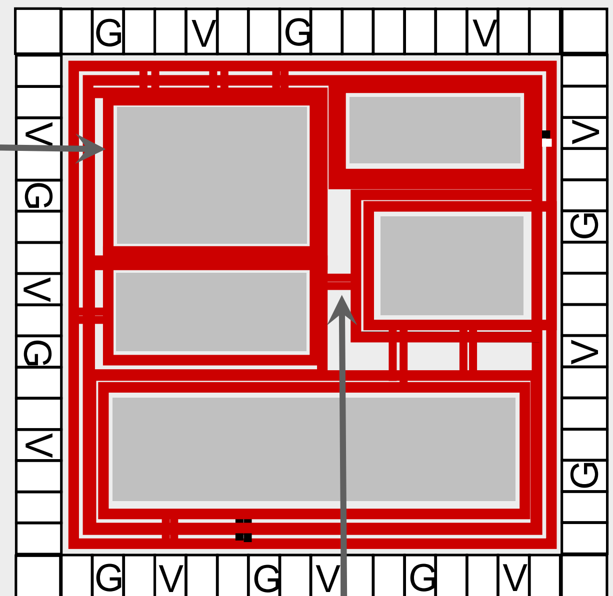
Pin assignment to two external blocks A and *B*



H. N. Brady, "An Approach to Topological Pin Assignment", IEEE Trans. on CAD 3(3) (1984), pp. 250-255

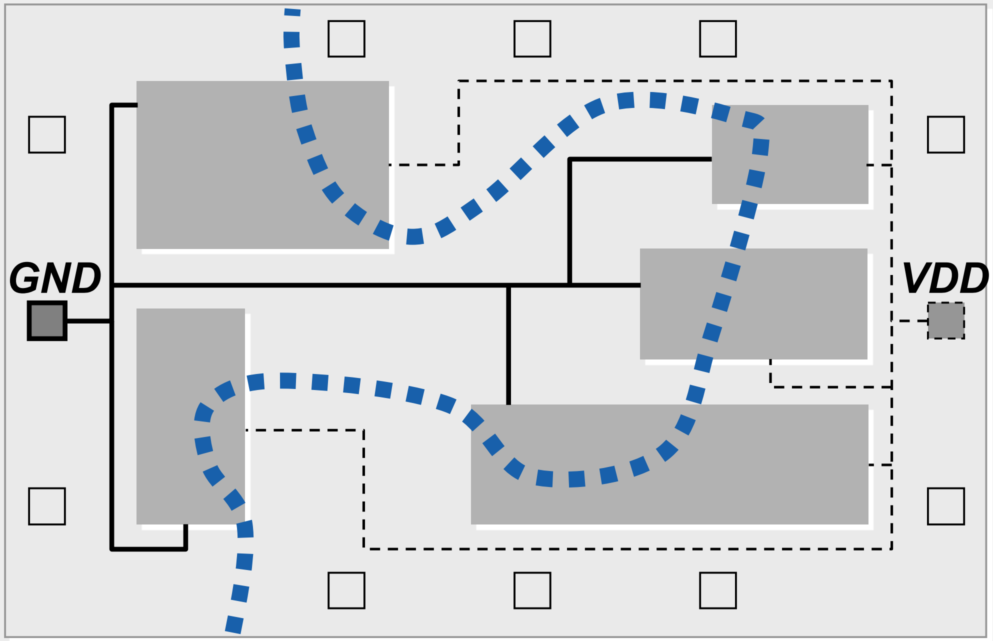
Power-ground distribution for a chip floorplan

Power and ground rings per block or abutted blocks



Trunks connect rings to each other or to top-level power ring

Planar routing



Hamiltonian path

Planar routing

Step 1: Planarize the topology of the nets

- As both power and ground nets must be routed on one layer, the design should be split using the Hamiltonian path

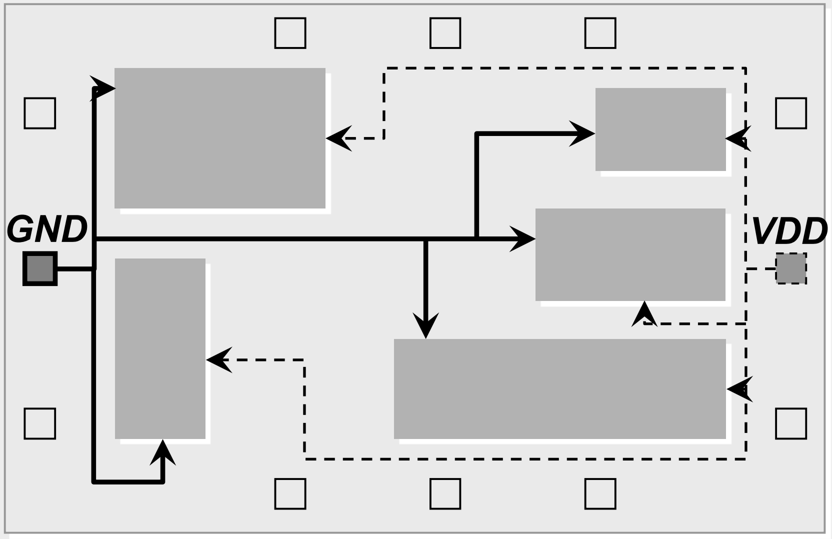
Step 2: Layer assignment

- Net segments are assigned to appropriate routing layers

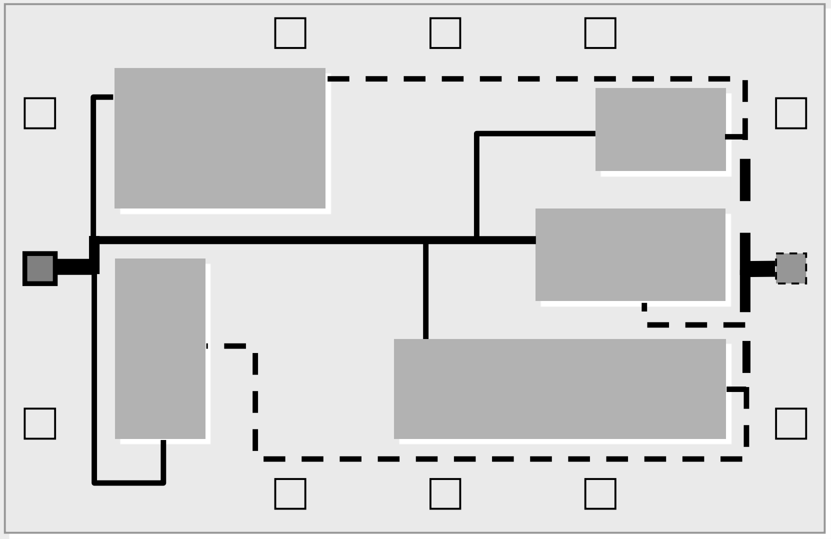
Step 3: Determining the widths of the net segments

- A segment’s width is determined from the sum of the currents from all the cells to which it connects

Planar routing



Generating topology
of the two supply nets



Adjusting widths of the segments
with regard to their current loads

3.7 Power and Ground Routing

Mesh routing

Step 1: Creating a ring

- A ring is constructed to surround the entire core area of the chip, and possibly individual blocks.

Step 2: Connecting I/O pads to the ring

Step 3: Creating a mesh

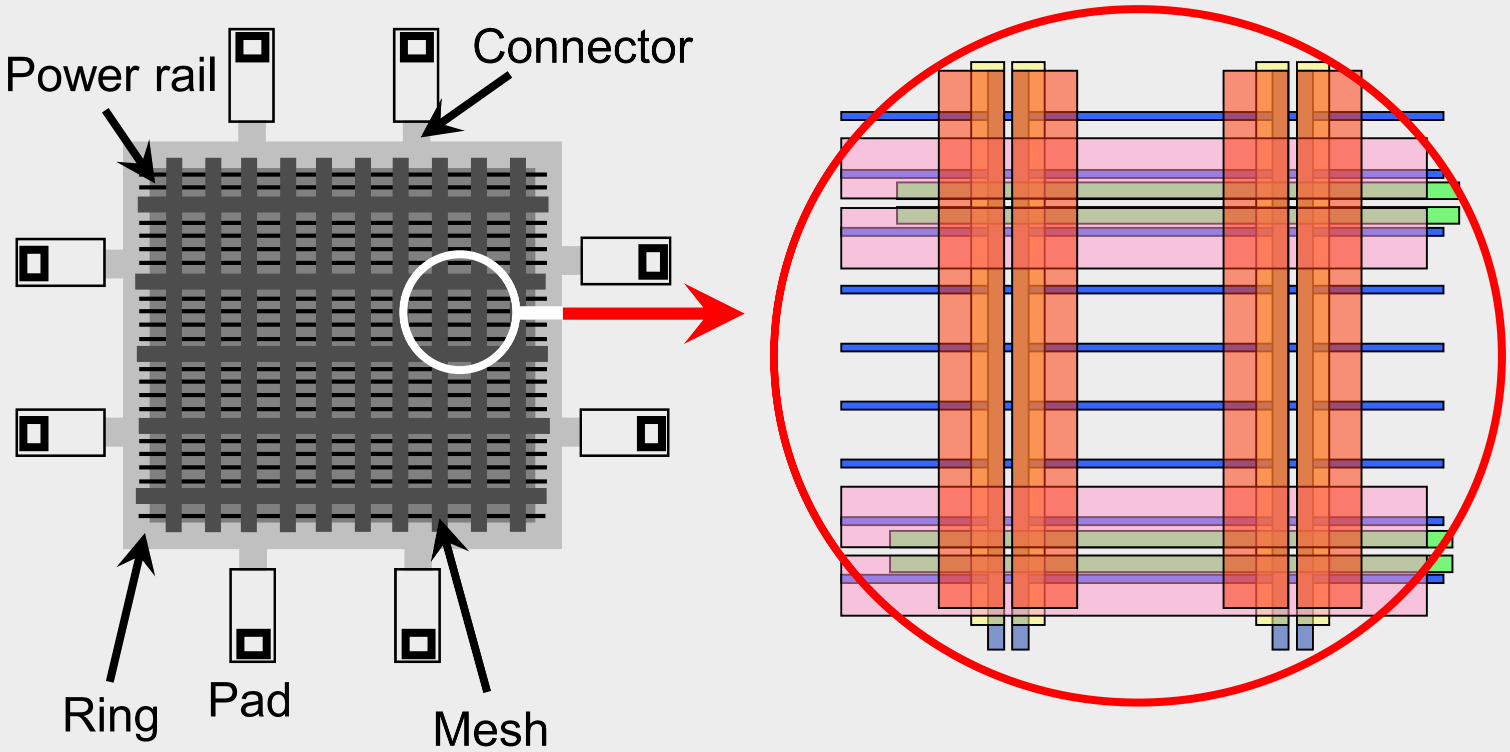
- A power mesh consists of a set of stripes at defined pitches on two or more layers

Step 4: Creating Metal1 rails

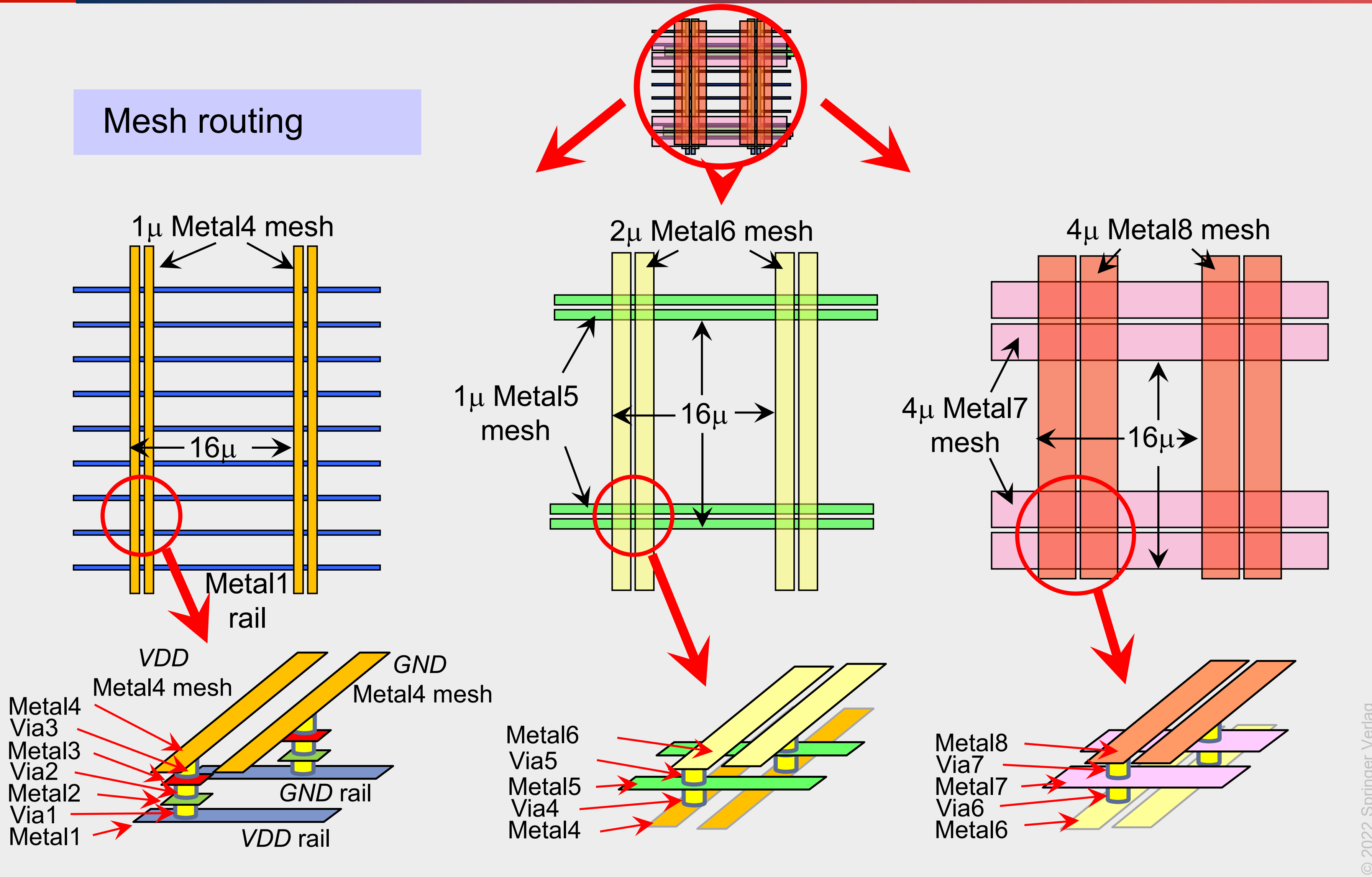
- Power mesh consists of a set of stripes at defined pitches on two or more layers

Step 5: Connecting the Metal1 rails to the mesh

Mesh routing



Mesh routing



M1-to-M4 connection **M4-to-M6 connection** **M6-to-M8 connection**

Summary of Chapter 3 – Objectives and Terminology

- Traditional floorplanning
  – Assumes area estimates for top-level circuit modules
  – Determines shapes and locations of circuit modules
  – Minimizes chip area and length of global interconnect
- Additional aspects
  – Assigning/placing I/O pads
  – Defining channels between blocks for routing and buffering
  – Design of power and ground networks
  – Estimation and optimization of chip timing and routing congestion
- Fixed-outline floorplanning
  – Chip size is fixed, focus on interconnect optimization
  – Can be applied to individual chip partitions (hierarchically)
- Structure and types of floorplans
  – Slicing versus non-slicing, the wheels
  – Hierarchical
  – Packed
  – Zero-deadspace

Summary of Chapter 3 – Data Structures for Floorplanning

- Slicing trees and Polish expressions
  - Evaluating a floorplan represented by a Polish expression
- Horizontal and vertical constraint graphs
  - A data structure to capture (non-slicing) floorplans
  - Longest paths determine floorplan dimensions
- Sequence pair
  - An array-based data structure that captures the information
  - contained in H+V constraint graphs
  - Makes constraint graphs unnecessary in practice
- Floorplan sizing
  - Shape-function arithmetic
  - An algorithm for slicing floorplans

Summary of Chapter 3 – Algorithms for Floorplanning

- Cluster growth
  - Simple, fast and intuitive
  - Not competitive in practice
- Simulated annealing
  - Stochastic optimization with hill-climbing
  - Many details required for high-quality implementation (e.g., temperature schedule)
  - Difficult to debug, fairly slow
  - Competitive in practice
- Pin assignment
  - Peripheral I/Os versus area-array I/Os
  - Given "ideal locations", project them onto perimeter and shift around, while preserving initial ordering
- Power and ground routing
  - Planar routing in channels between blocks
  - Can form rings around blocks to increase current supplied and to improve reliability
  - Mesh routing