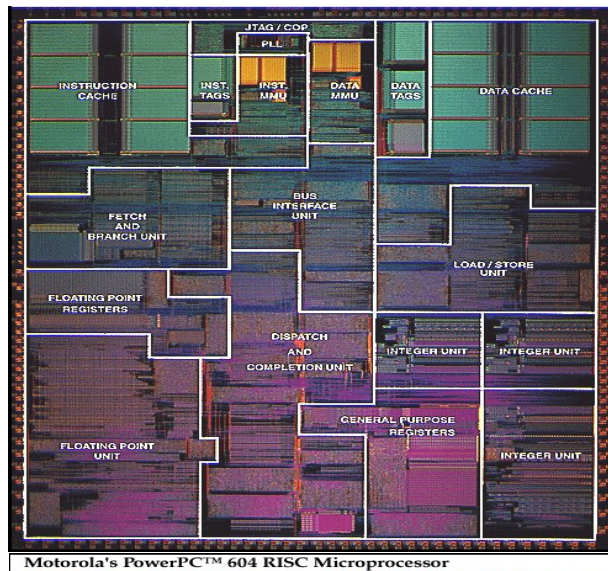
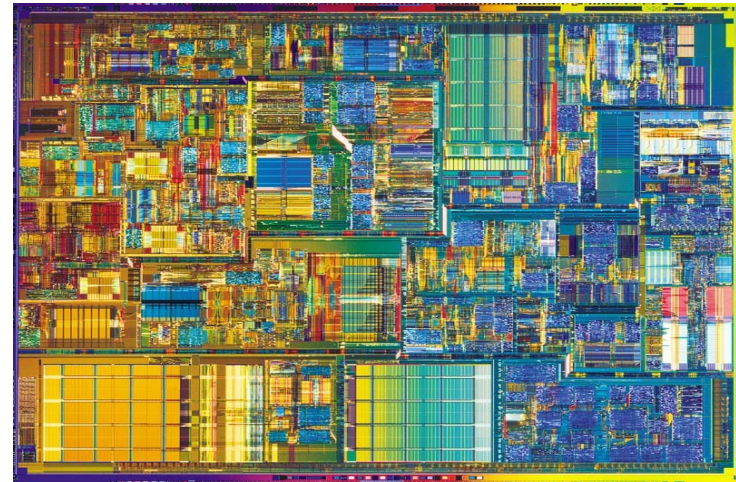


# Unit 5B: Floorplanning

- Course contents
  - Floorplan basics
  - Normalized Polish expression for slicing floorplans
  - B\*-trees for non-slicing floorplans
- Readings
  - Chapters 8 and 5.6



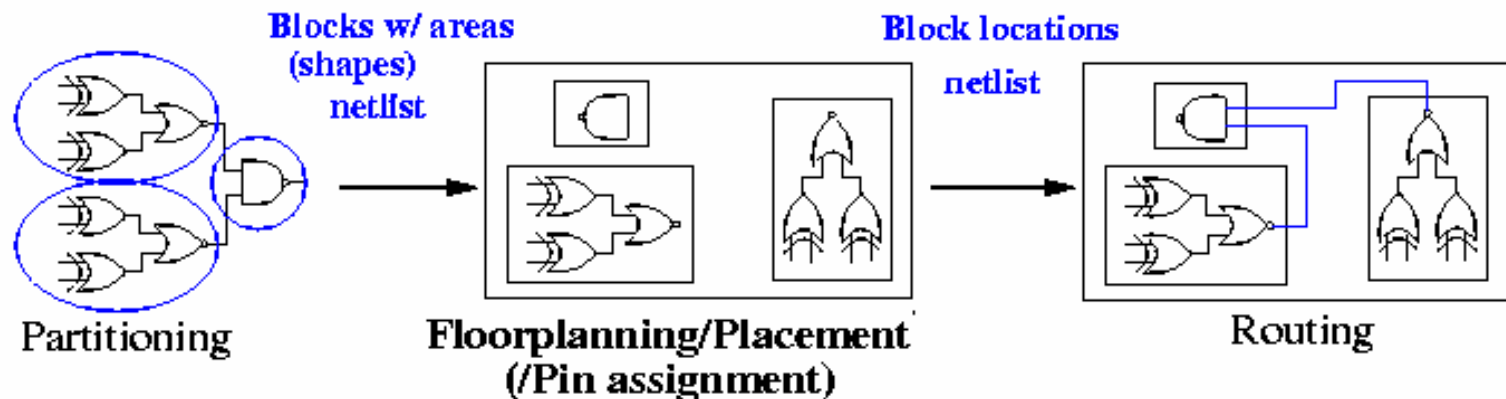
**PowerPC 604**



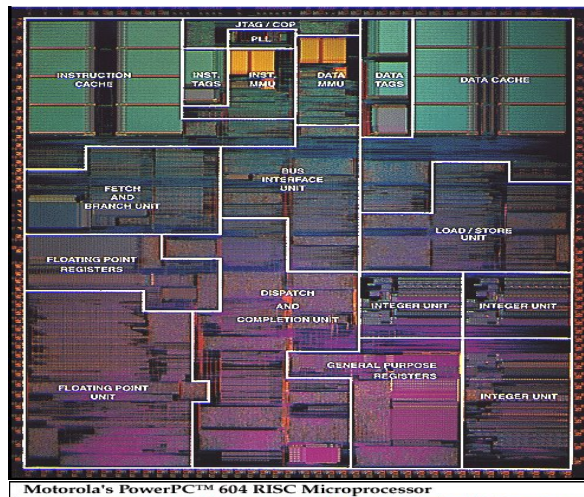
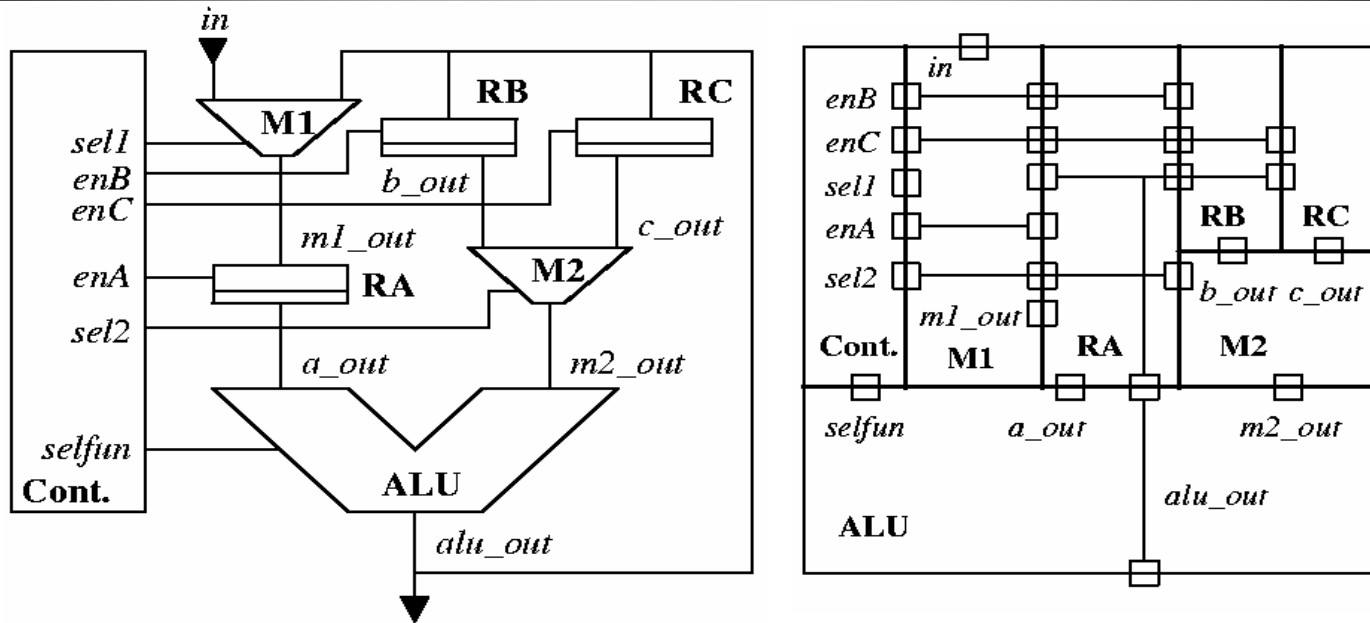
**Pentium 4**

# Floorplanning

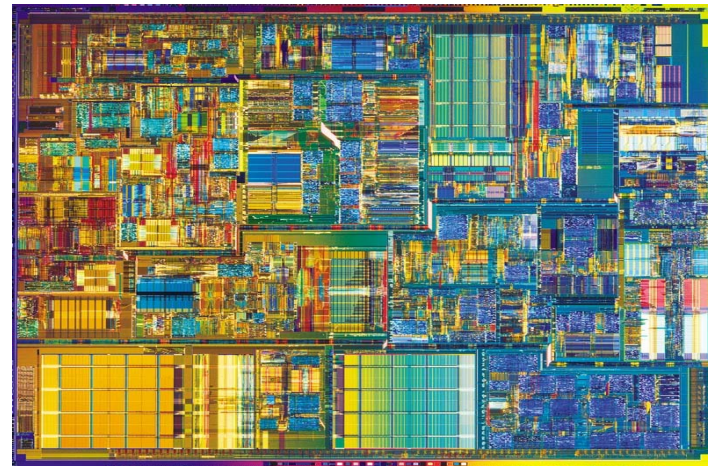
- Partitioning leads to
  - Blocks with well-defined **areas and shapes** (**rigid/hard** blocks).
  - Blocks with approximate areas and no particular shapes (**flexible/soft** blocks).
  - A **netlist** specifying connections between the blocks.
- Objectives
  - Find **locations** for all blocks.
  - Consider shapes of soft block and pin locations of all the blocks.



# Early Layout Decision Example



**PowerPC 604**



**Pentium 4**

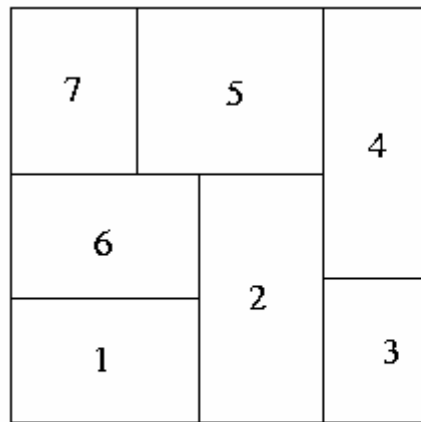
# Early Layout Decision Methodology

---

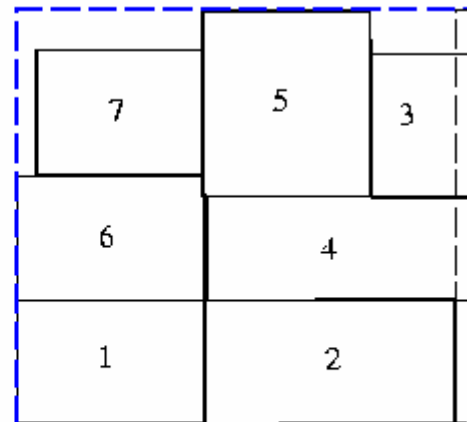
- An integrated circuit is essentially a two-dimensional medium; taking this aspect into account in early stages of the design helps in creating designs of good quality.
- Floorplanning gives early feedback: thinking of layout at early stages may suggest valuable architectural modifications; floorplanning also aids in estimating delay due to wiring.
- Floorplanning fits very well in a *top-down* design strategy, the *step-wise refinement* strategy also propagated in software design.
- Floorplanning assumes, however, *flexibility* in layout design, the existence of cells that can adapt their shapes and terminal locations to the environment.

# Floorplanning Problem

- Inputs to the floorplanning problem:
  - A set of blocks, hard or soft.
  - Pin locations of hard blocks.
  - A netlist.
- Objectives: minimize **area**, reduce **wirelength** for (critical) nets, maximize **routability** (minimize **congestion**), determine shapes of soft blocks, etc.

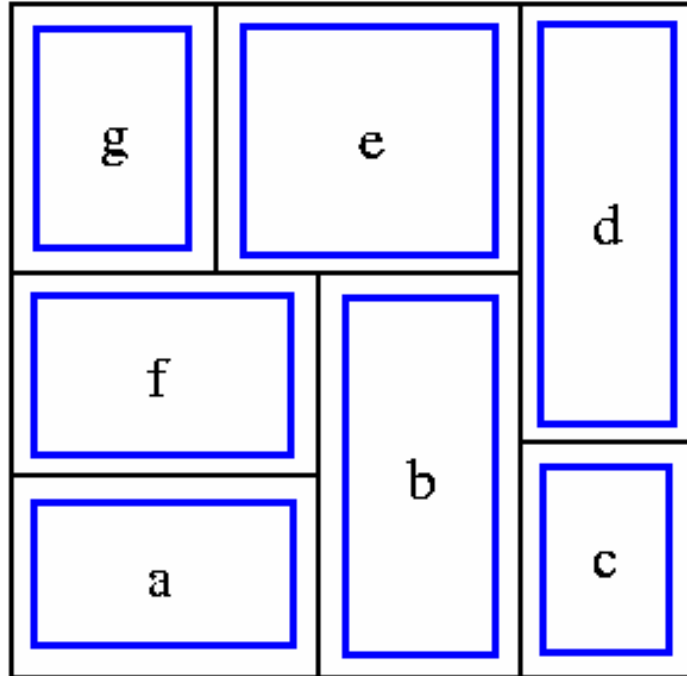





An optimal floorplan,  
in terms of area

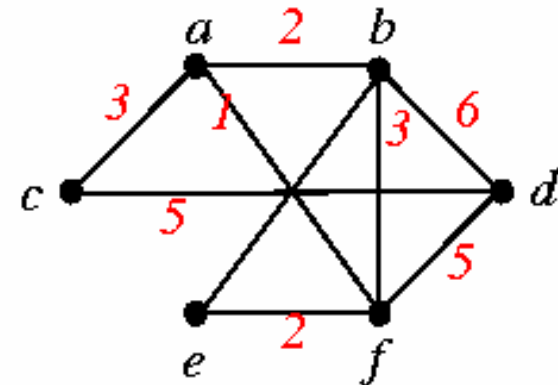


A non-optimal floorplan

# Floorplan Design



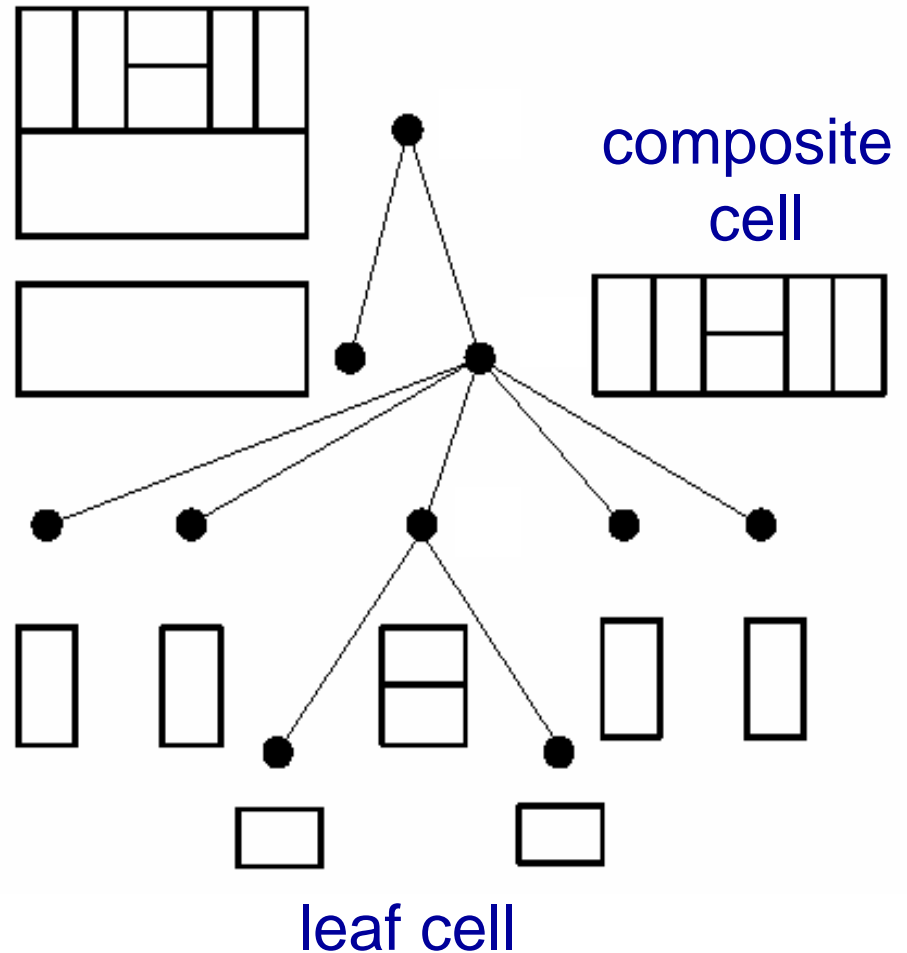
- *Modules:* 
- *Area:*  $A=xy$
- *Aspect ratio:*  $r \leq y/x \leq s$
- *Rotation:*  
- *Module connectivity*





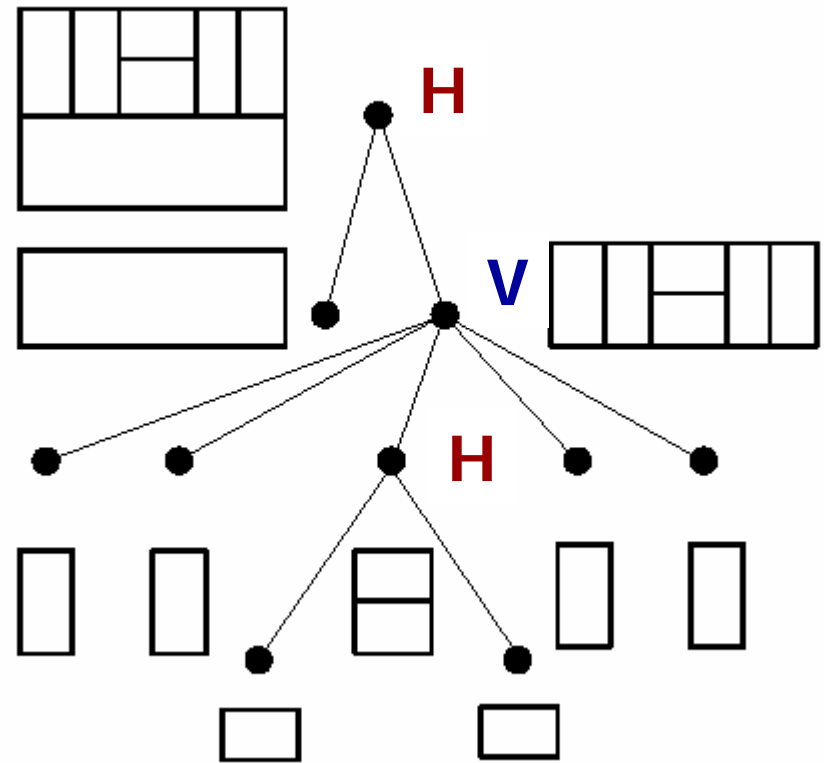
# Floorplanning Concepts

- **Leaf cell (block/module):** a cell at the lowest level of the hierarchy; it does not contain any other cell.
- **Composite cell (block/module):** a cell that is composed of either leaf cells or composite cells. The entire IC is the highest-level composite cell.



# Slicing Floorplan + Slicing Tree

- A composite cell's subcells are obtained by a horizontal or vertical *bisection* of the composite cell.
- Slicing floorplans can be represented by a **slicing tree**.
- In a slicing tree, all cells (except for the top-level cell) have a *parent*, and all composite cells have *children*.
- A slicing floorplan is also called a floorplan of **order 2**.



**H:** horizontal cut

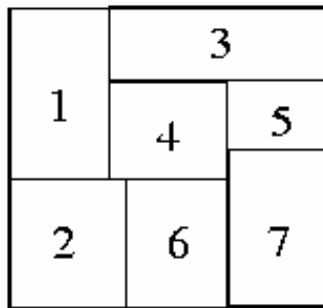
**V:** vertical cut

**different from the definitions in the text!!**

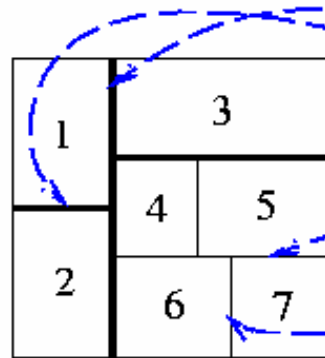


# Skewed Slicing Tree

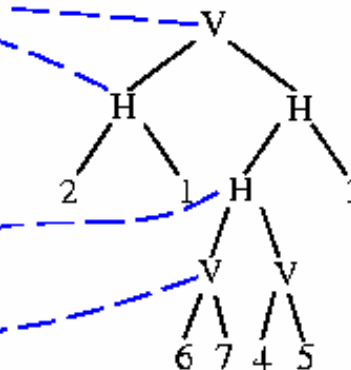
- **Rectangular dissection:** Subdivision of a given rectangle by a finite # of horizontal and vertical line segments into a finite # of non-overlapping rectangles.
- **Slicing structure:** a rectangular dissection that can be obtained by repetitively subdividing rectangles horizontally or vertically.
- **Slicing tree:** A binary tree, where each internal node represents a vertical cut line or horizontal cut line, and each leaf a basic rectangle.
- **Skewed slicing tree:** One in which no node and its **right** child are the same.



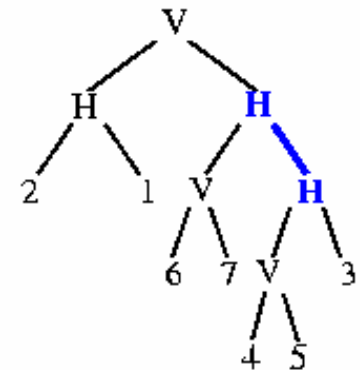
Non-slicing floorplan



Slicing floorplan



A slicing tree (skewed)



Another slicing tree (non-skewed)

# Slicing Floorplan Design by Simulated Annealing

---

- Related work
  - Wong & Liu, “A new algorithm for floorplan design,” DAC-86.
    - Considers slicing floorplans.
  - Wong & Liu, “Floorplan design for rectangular and L-shaped modules,” ICCAD'87.
    - Also considers L-shaped modules.
  - Wong, Leong, Liu, *Simulated Annealing for VLSI Design*, pp. 31--71, Kluwer Academic Publishers, 1988.
- Ingredients to simulated annealing
  - solution space?
  - neighborhood structure?
  - cost function?
  - annealing schedule?

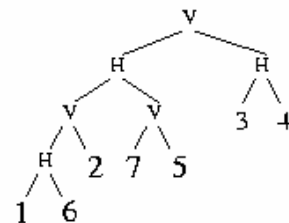
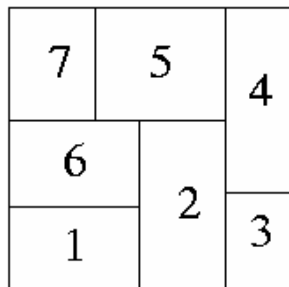
# Solution Representation

- An expression  $E = e_1 e_2 \dots e_{2n-1}$ , where  $e_i \in \{1, 2, \dots, n, H, V\}$ ,  $1 \leq i \leq 2n-1$ , is a **Polish expression** of length  $2n-1$  iff
  - every operand  $j$ ,  $1 \leq j \leq n$ , appears exactly once in  $E$ ;
  - (the balloting property)** for every subexpression  $E_i = e_1 \dots e_i$ ,  $1 \leq i \leq 2n-1$ , # operands  $>$  # operators.

1 6 H 3 5 V 2 H V 7 4 H V

# of operands = 4 ..... = 7  
 # of operators = 2 ..... = 5

- Polish expression  $\leftrightarrow$  Postorder traversal.
- $ijH$ : rectangle  $i$  on bottom of  $j$ ;  $ijV$ : rectangle  $i$  on the left of  $j$ .



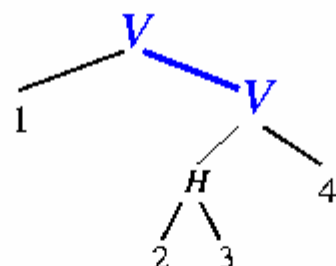
$E = 16H2V75VH34HV$

$E = 16 + 2 * 75 * + 34 + *$

*Postorder traversal of a tree!*

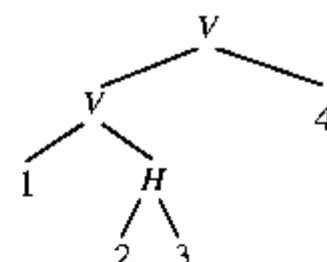
# Redundant Representations

1	3	4
	2	



$E = 123H4VV$

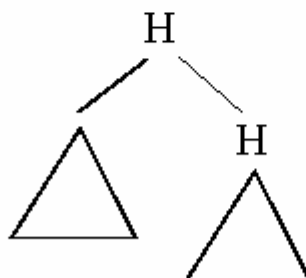
*non-skewed!*



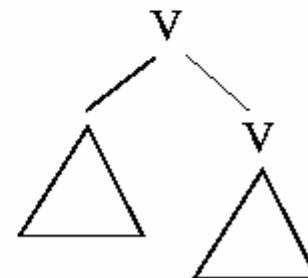
$E = 123HV4V$

*skewed!*

**Non-skewed  
cases**



..... HH .....

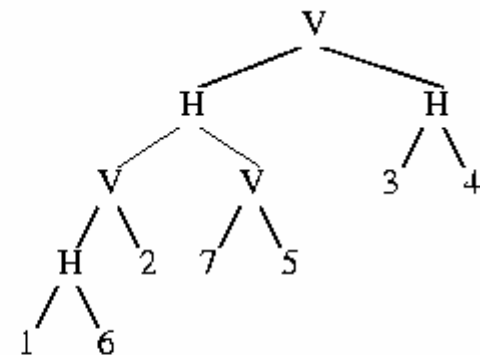
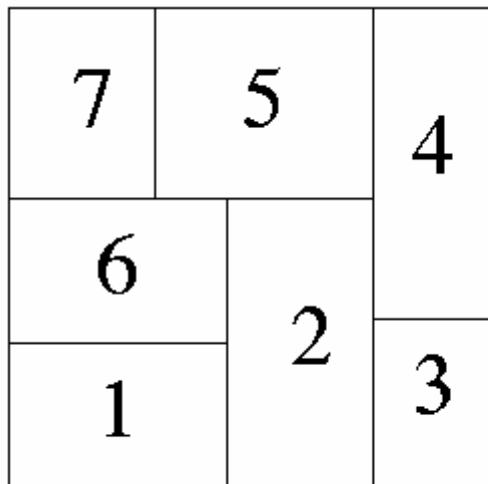


..... VV .....

- **Question:** How to eliminate ambiguous representation?

# Normalized Polish Expression

- A Polish expression  $E = e_1 e_2 \dots e_{2n-1}$  is called **normalized** iff  $E$  has no consecutive operators of the same type ( $H$  or  $V$ ).
- Given a **normalized Polish expression**, we can construct a **unique** rectangular slicing structure.

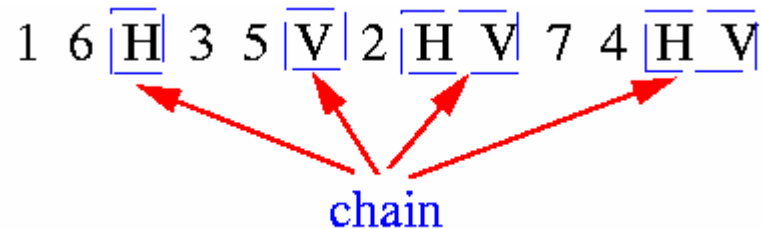


$E = 16H2V75VH34HV$

A normalized Polish expression

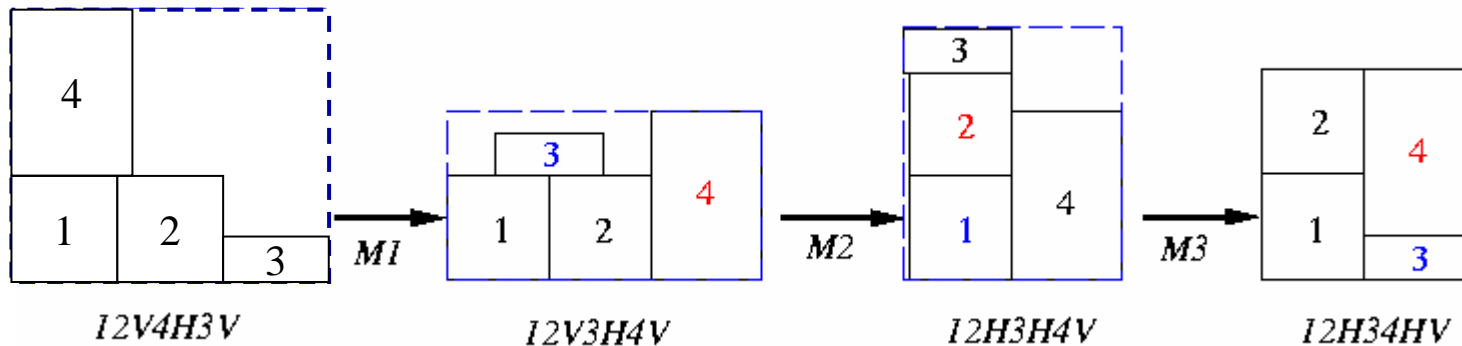
# Neighborhood Structure

- **Chain:**  $HVHVH \dots$  or  $VHVHV \dots$



- **Adjacent:** 1 and 6 are adjacent operands; 2 and 7 are adjacent operands; 5 and  $\overline{V}$  are adjacent operand and operator.
- 3 types of moves:
  - **M1 (Operand Swap):** Swap two adjacent operands.
  - **M2 (Chain Invert):** Complement some chain ( $\overline{V} \equiv H$ ,  $H \equiv \overline{V}$ ).
  - **M3 (Operator/Operand Swap):** Swap two adjacent operand and operator.

# Effects of Perturbation

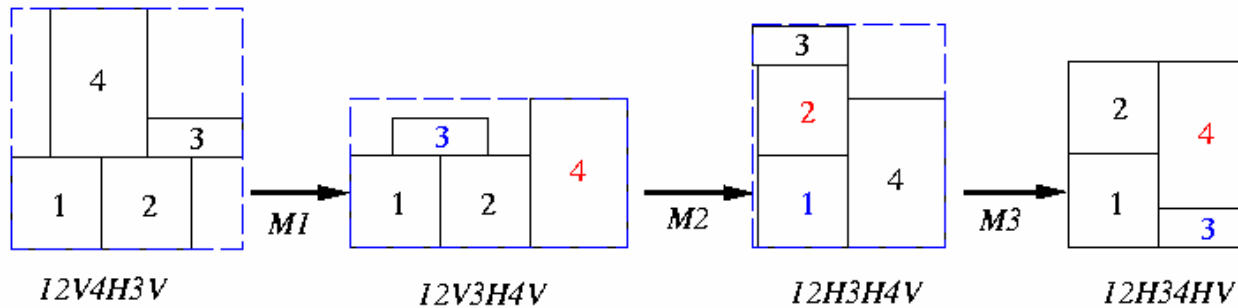


- **Question:** The balloting property holds during the moves?
  - *M1* and *M2* moves are OK.
  - **Check the *M3* moves!** Reject “illegal” *M3* moves.
- **Check *M3* moves:** Assume that the *M3* move swaps the operand  $e_i$  with the operator  $e_{i+1}$ ,  $1 \leq i \leq k-1$ . Then, the swap will not violate the balloting property iff  $2N_{i+1} < i$ .
  - $N_k$ : # of operators in the Polish expression  $E = e_1 e_2 \dots e_k$ ,  $1 \leq k \leq 2n-1$

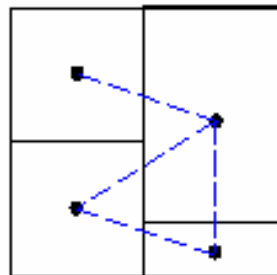


# Cost Function

- $\phi = A + \lambda W$ .
  - $A$ : area of the smallest rectangle
  - $W$ : overall wiring length
  - $\lambda$  : user-specified parameter

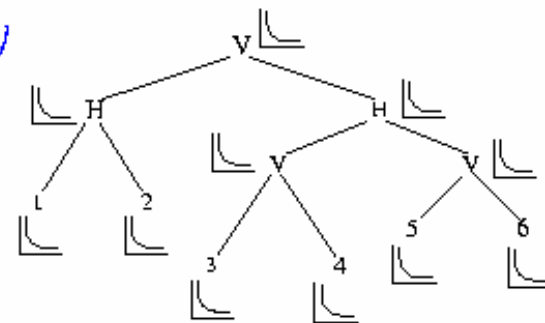
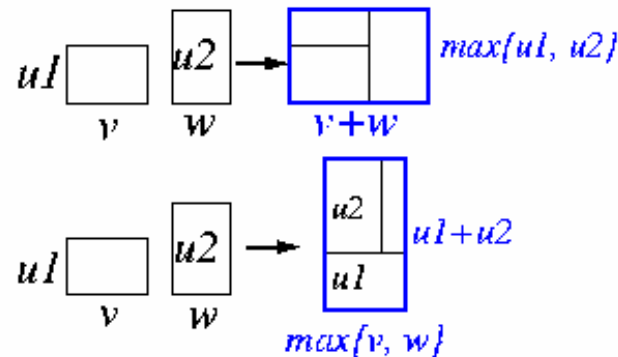
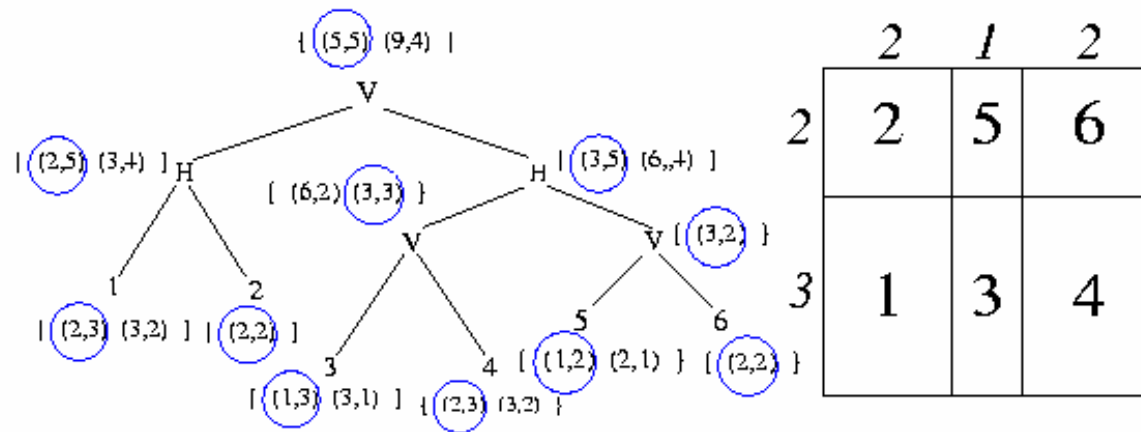


- $W = \sum_{ij} c_{ij} d_{ij}$ .
  - $c_{ij}$ : # of connections between blocks  $i$  and  $j$ .
  - $d_{ij}$ : center-to-center distance between basic rectangles  $i$  and  $j$ .



# Area Computation for Hard Blocks

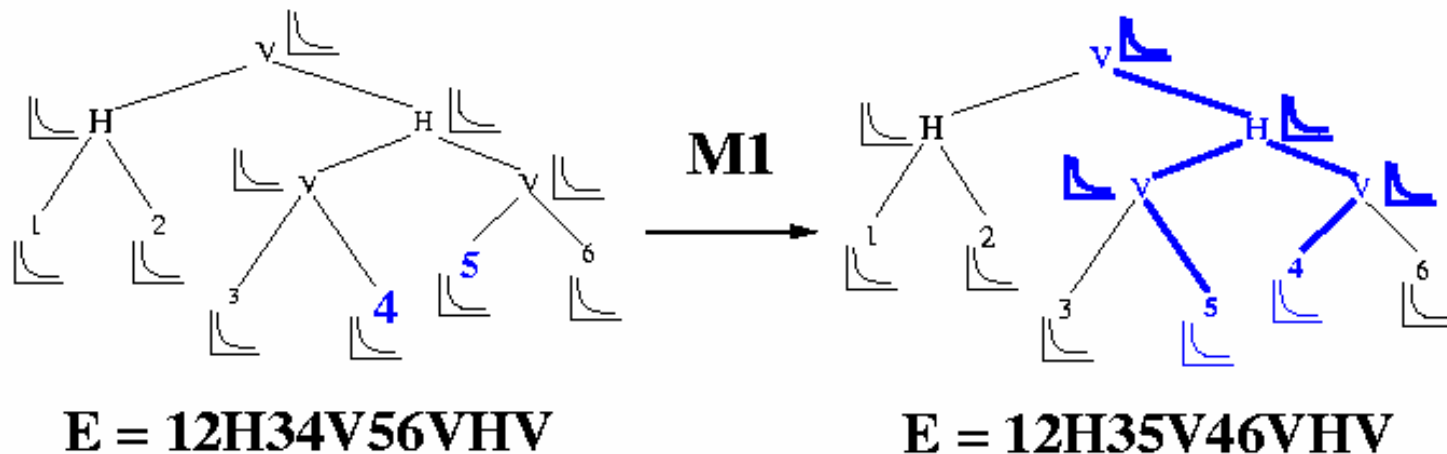
- Allow rotation



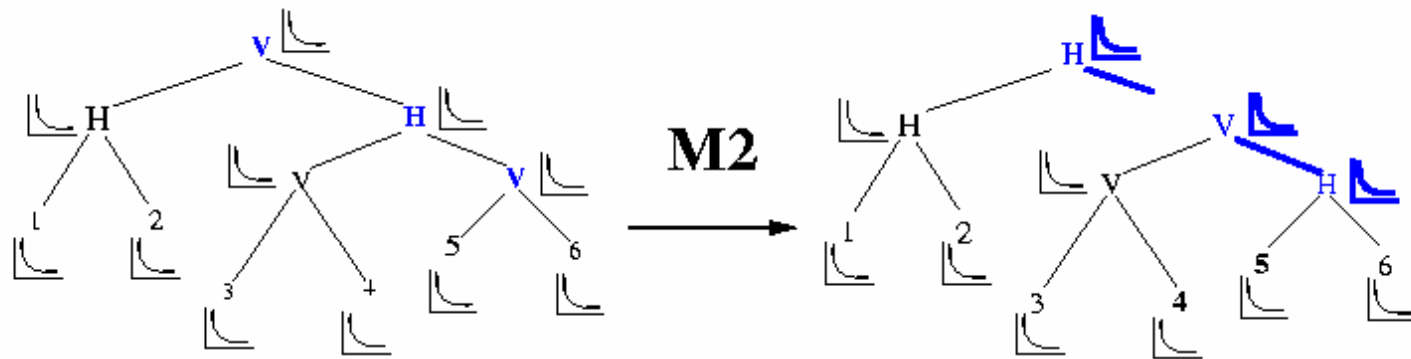
- Wiring cost?
  - Center-to-center interconnection length

# Incremental Computation of Cost Function

- Each move leads to only a minor modification of the Polish expression.
- At most **two paths** of the slicing tree need to be updated for each move.

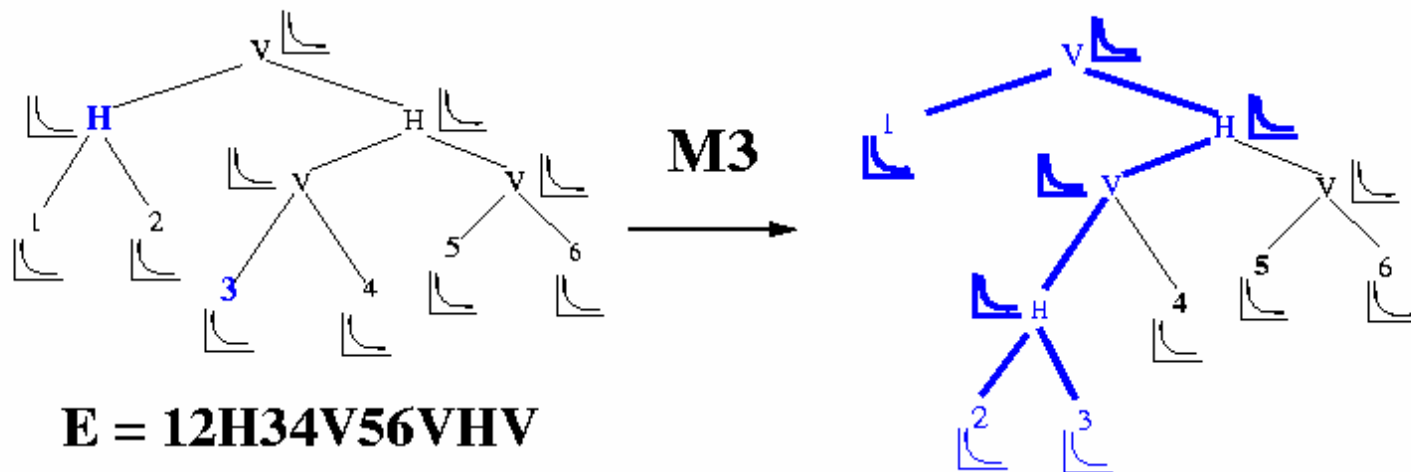


# Incremental Computation of Cost Function (cont'd)



**E = 12H34V56VHV**

**E = 12H34V56HVV**



**E = 12H34V56VHV**

**E = 123H4V56VHV**

# Annealing Schedule

---

- Initial solution:  $12V3V \dots nV$ .

1	2	3		n
---	---	---	--	---

- $T_i = r^i T_0$ ,  $i = 1, 2, 3, \dots$ ;  $r = 0.85$ .
- At each temperature, try  $kn$  moves ( $k = 5-10$ ).
- Terminate the annealing process if
  - # of accepted moves  $< 5\%$ ,
  - temperature is low enough, or
  - run out of time.

# Algorithm: Wong-Liu ( $P, \varepsilon, r, k$ )

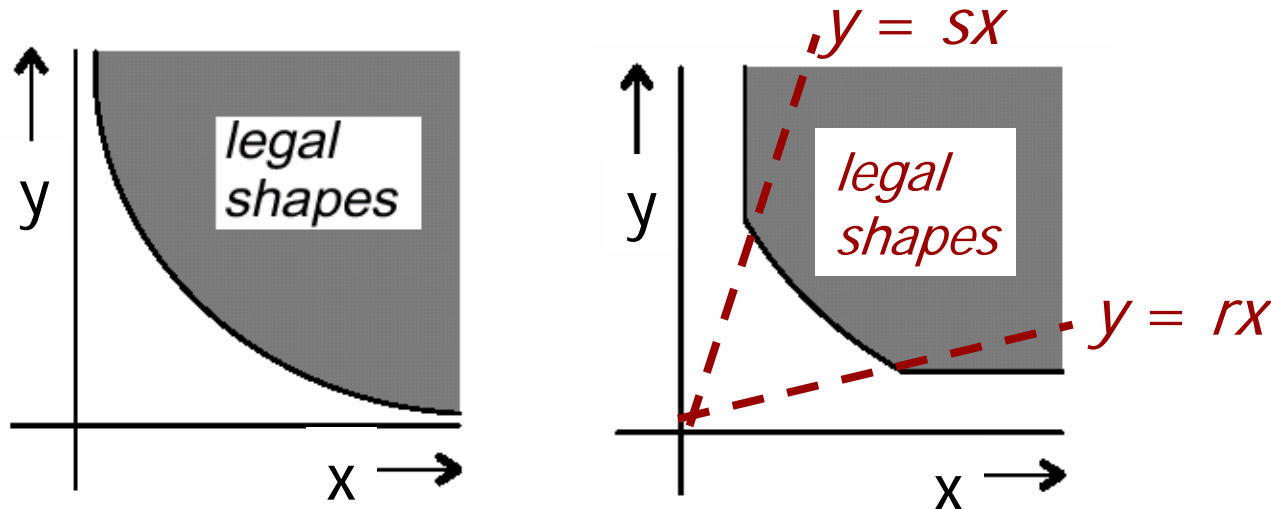
```

1 begin
2  $E \leftarrow 12V3V4V \dots nV$ ; /* initial solution */
3  $Best \leftarrow E$ ;  $T_0 \leftarrow \frac{\Delta_{avg}}{\ln(P)}$ ;  $M \leftarrow MT \leftarrow uphill \leftarrow 0$ ;  $N = kn$ ;
4 repeat
5    $MT \leftarrow uphill$ ; reject  $\leftarrow 0$ ;
6   repeat
7     SelectMove( $M$ );
8     Case  $M$  of
9        $M_1$ : Select two adjacent operands  $e_i$  and  $e_j$ ;  $NE \leftarrow \text{Swap}(E, e_i, e_j)$ ;
10       $M_2$ : Select a nonzero length chain  $C$ ;  $NE \leftarrow \text{Complement}(E, C)$ ;
11       $M_3$ : done  $\leftarrow \text{FALSE}$ ;
12      while not (done) do
13        Select two adjacent operand  $e_i$  and operator  $e_{i+1}$ ;
14        if ( $e_{i-1} \neq e_{i+1}$ ) and ( $2 N_{i+1} < i$ ) then done  $\leftarrow \text{TRUE}$ ;
13'      Select two adjacent operator  $e_i$  and operand  $e_{i+1}$ ;
14'      if ( $e_i \neq e_{i+2}$ ) then done  $\leftarrow \text{TRUE}$ ;
15       $NE \leftarrow \text{Swap}(E, e_i, e_{i+1})$ ;
16       $MT \leftarrow MT+1$ ;  $\Delta cost \leftarrow \text{cost}(NE) - \text{cost}(E)$ ;
17      if ( $\Delta cost \leq 0$ ) or ( $\text{Random}() < e^{-\frac{\Delta cost}{T}}$ )
18      then
19        if ( $\Delta cost > 0$ ) then uphill  $\leftarrow uphill + 1$ ;
20         $E \leftarrow NE$ ;
21        if  $\text{cost}(E) < \text{cost}(best)$  then best  $\leftarrow E$ ;
22      else reject  $\leftarrow reject + 1$ ;
23    until (uphill  $> N$ ) or ( $MT > 2N$ );
24     $T \leftarrow rT$ ; /* reduce temperature */
25  until (reject/ $MT > 0.95$ ) or ( $T < \varepsilon$ ) or OutOfTime;
26 end

```

# Shape Curve

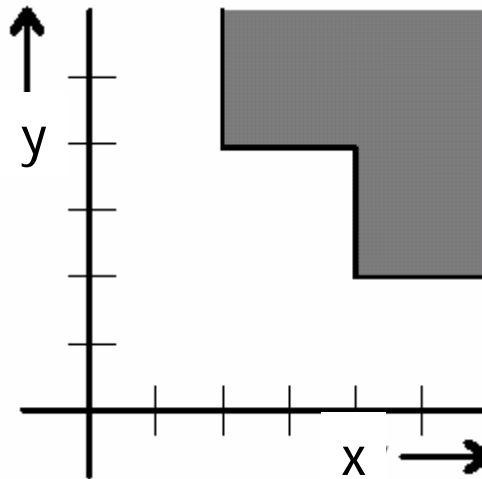
- Flexible cells imply that cells can have different aspect ratios.
- The relation between the width  $x$  and the height  $y$  is:  $xy = A$ , or  $y = A/x$ . The shape function is a hyperbola.
- Very thin cells are not interesting and often not feasible to design. The shape function is a combination of a hyperbola and two straight lines.
  - Aspect ratio:  $r \leq y/x \leq s$ .





## Shape Curve (cont'd)

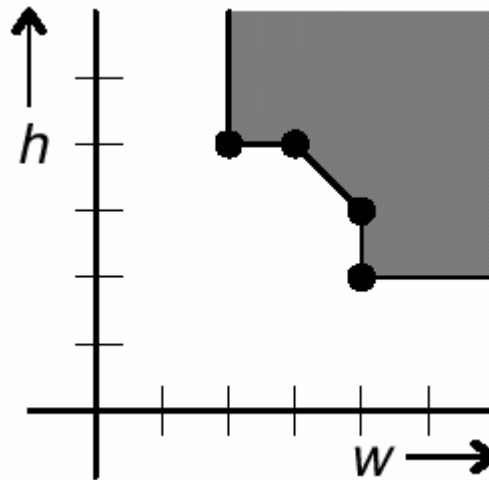
- Leaf cells are built from discrete transistors: it is not realistic to assume that the shape function follows the hyperbola continuously.
- In an extreme case, a cell is rigid: it can only be rotated and mirrored during floorplanning or placement.



The shape function of a  $2 \times 4$  inset cell.

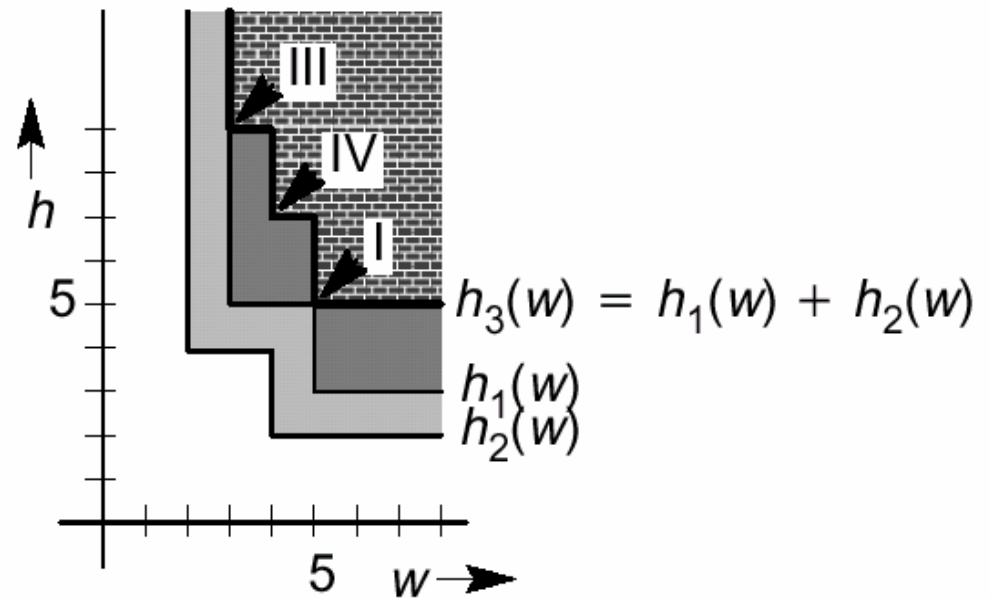
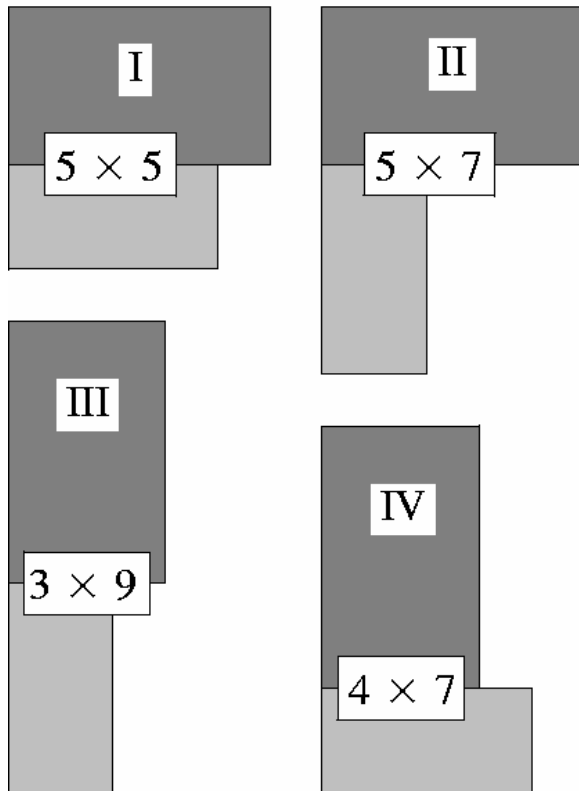
## Shape Curve (cont'd)

- In general, a *piecewise linear* function can be used to approximate any shape function.
- The points where the function changes its direction, are called the *corner (break) points* of the piecewise linear function.



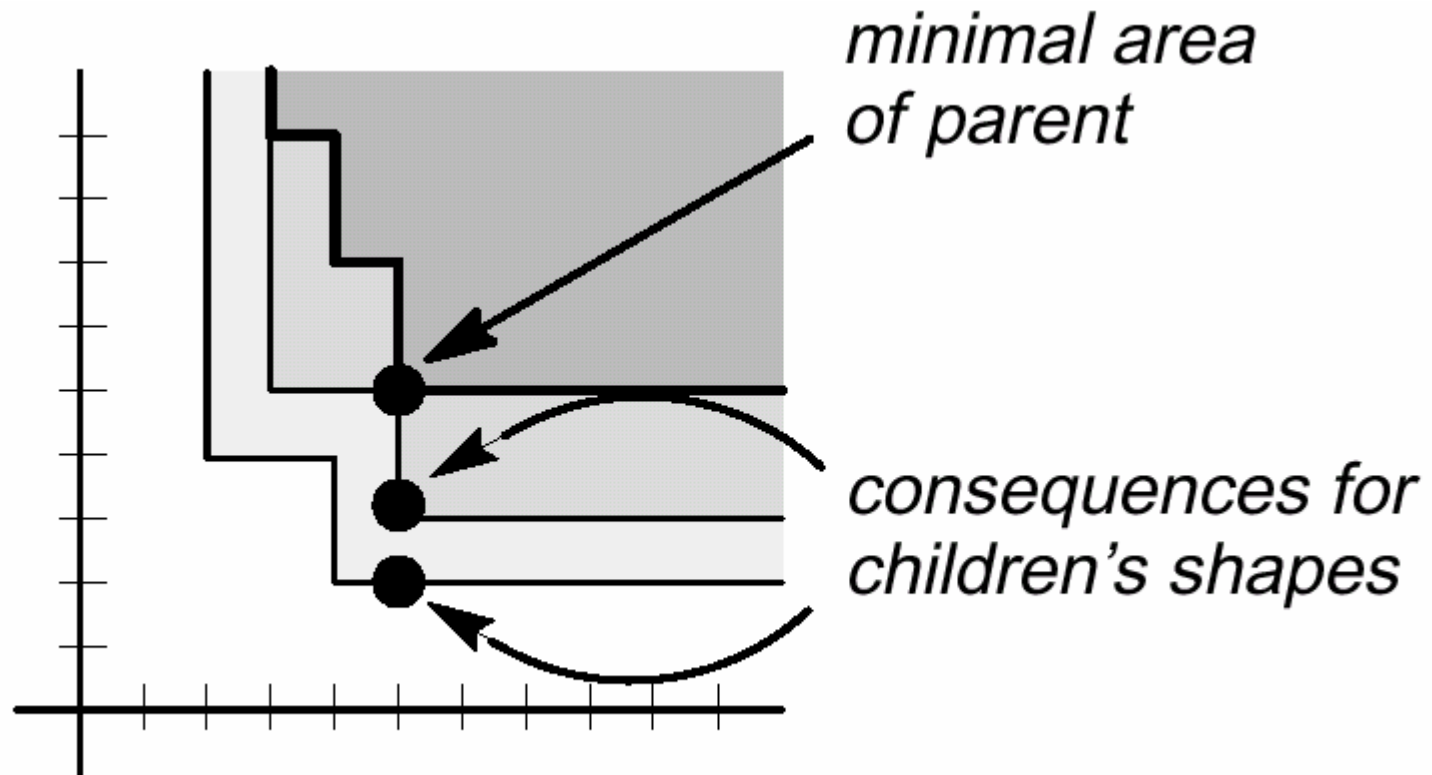
# Addition for Vertical Abutment

- Composition by vertical abutment  $\Rightarrow$  the addition of shape functions.



# Deriving Shapes of Children

- A choice for the minimal shape of composite cell fixes the shapes of the shapes of its children cells.



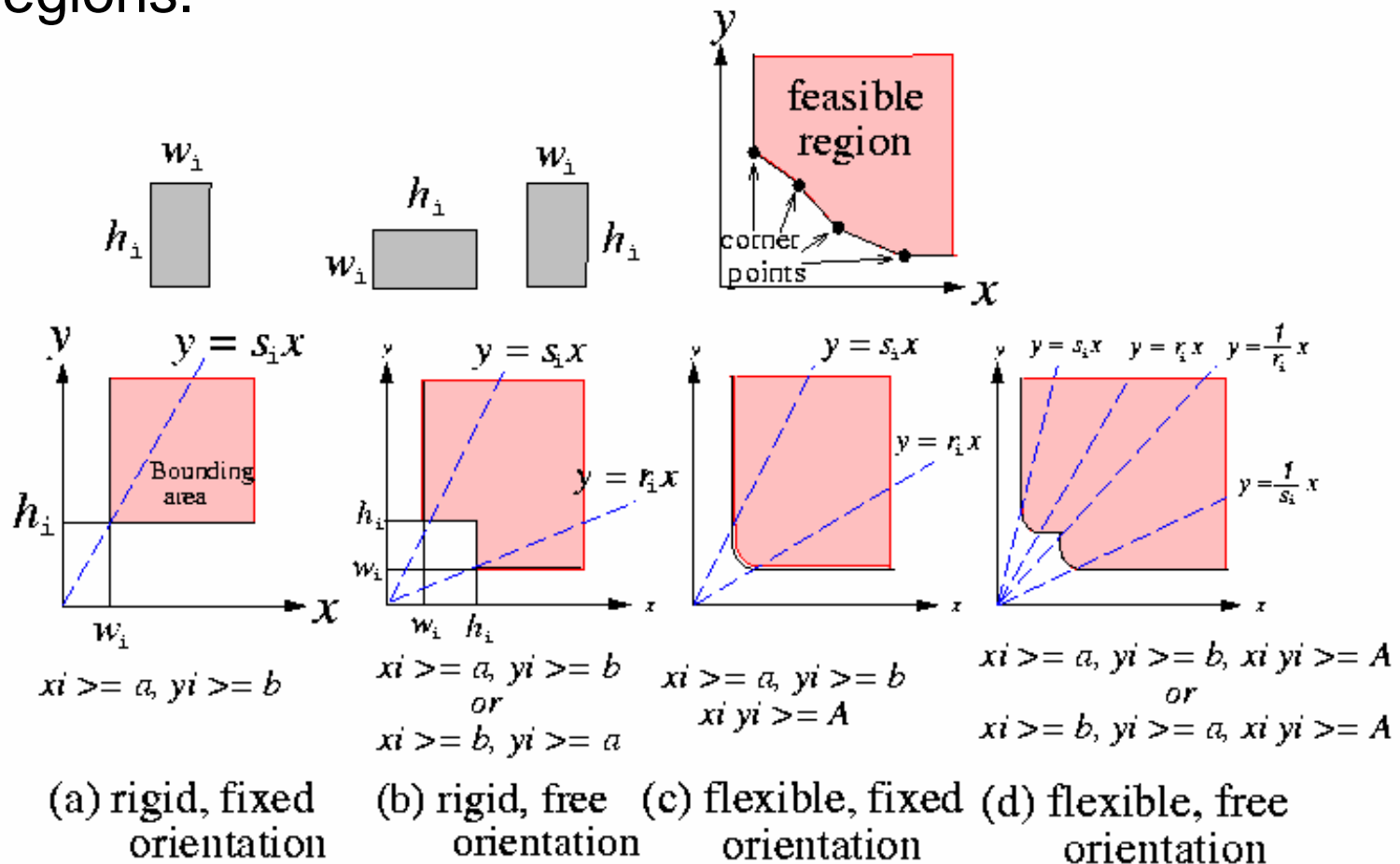
# Sizing Algorithm for Slicing Floorplans

---

- The shape functions of all leaf cells are given as piecewise linear functions.
- Traverse the slicing tree in order to compute the shape functions of all composite cells (bottom-up composition).
- Choose the desired shape of the top-level cell; as the shape function is piecewise linear, only the break points of the function need to be evaluated, when looking for the minimal area.
- Propagate the consequences of the choice down to the leaf cells (top-down propagation).
- The sizing algorithm runs in polynomial time for slicing floorplans
  - NP-complete for non-slicing floorplans

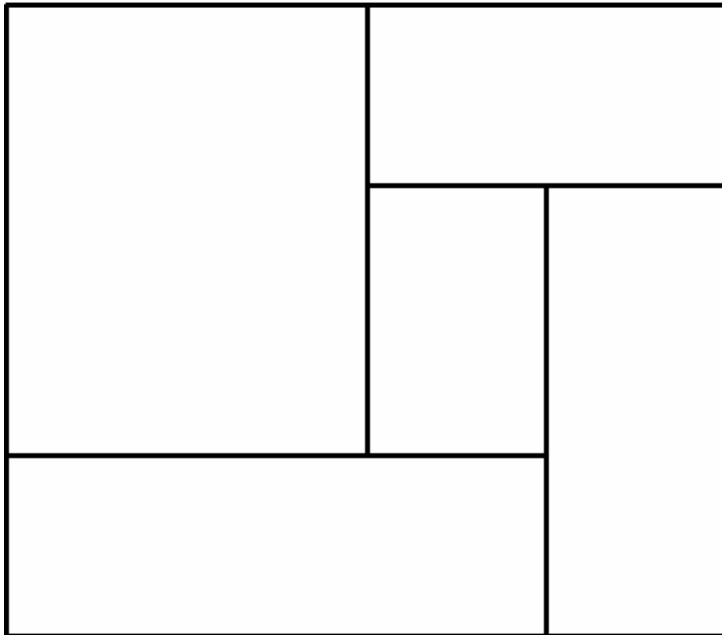
# Feasible Implementations

- Shape curves correspond to different kinds of constraints where the shaded areas are feasible regions.



# Wheel or Spiral Floorplan

---

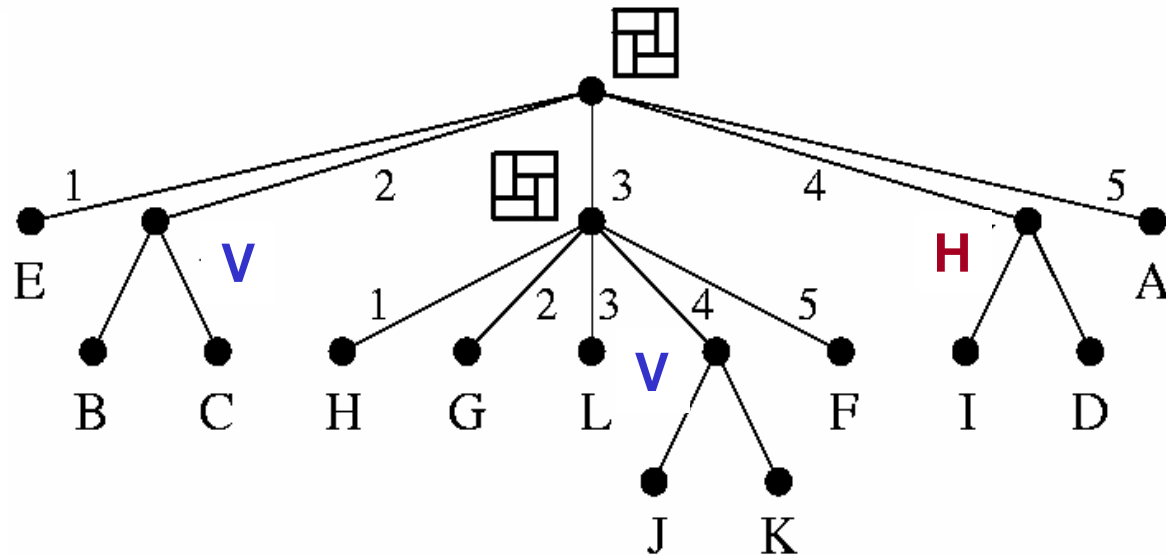
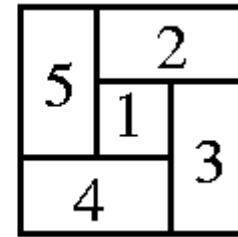
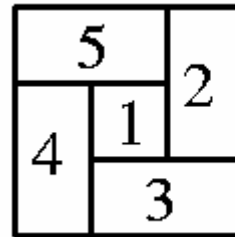
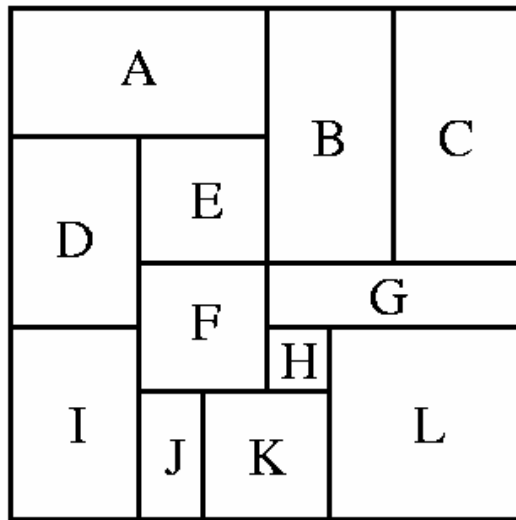


- This floorplan is not slicing!
- **Wheel** is the smallest non-slicing floorplans.

- Limiting floorplans to those that have the slicing property is reasonable: it certainly facilitates floorplanning algorithms.
- Taking the shape of a wheel floorplan and its mirror image as the basis of operators leads to hierarchical descriptions of *order 5*.

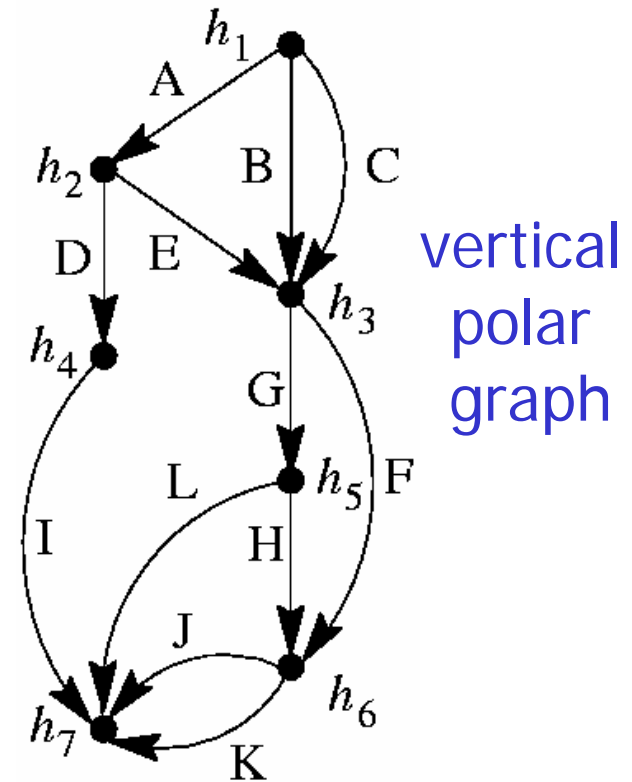
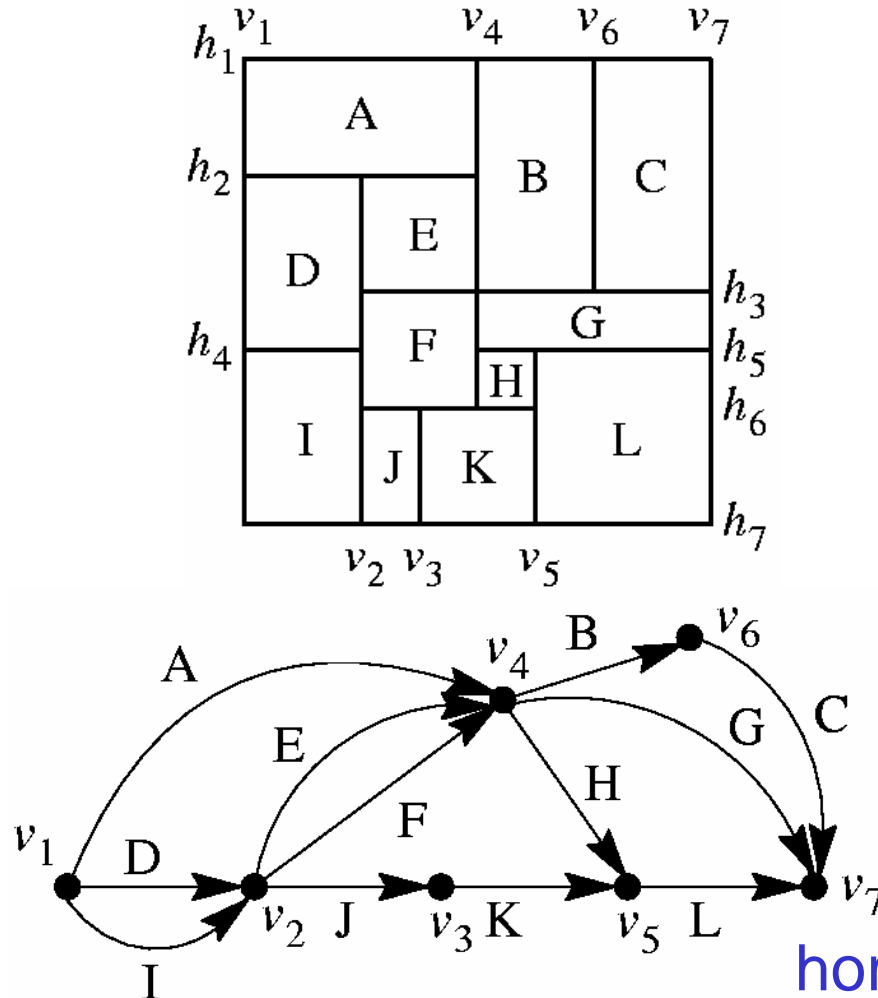


# Order-5 Floorplan Examples



# General Floorplan Representation: Polar Graphs

- vertex: channel segment
- edge (weight): cell/block/module (dimension).

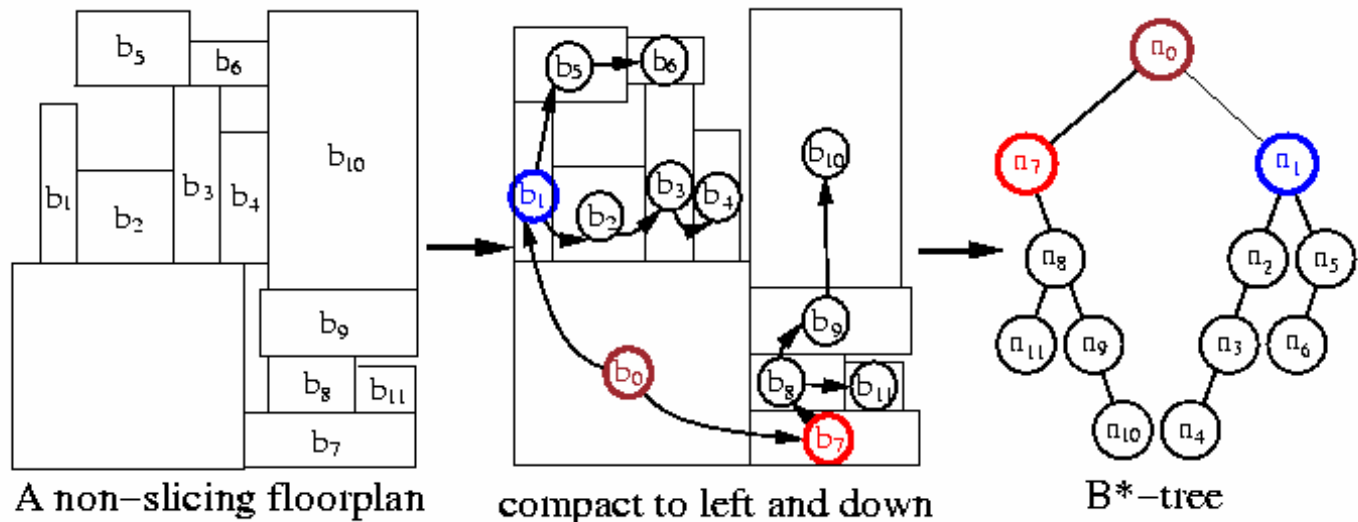


vertical  
polar  
graph

horizontal polar graph

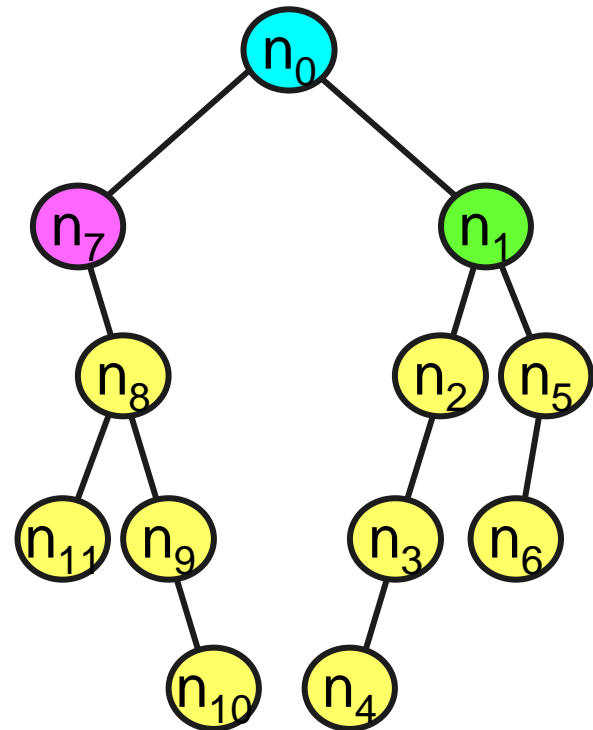
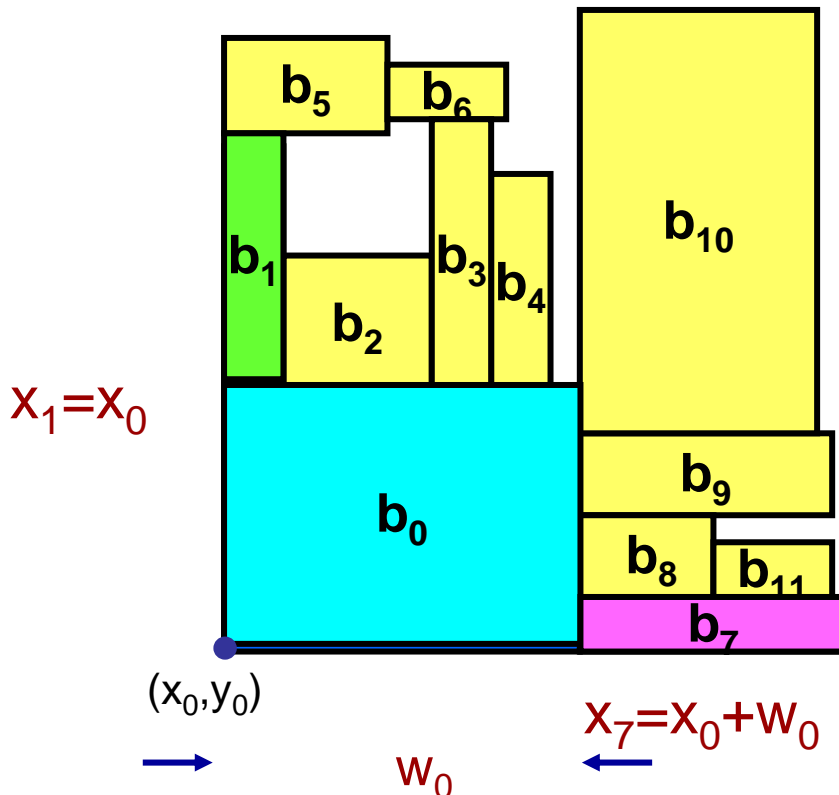
# B\*-Tree for Compacted Non-Slicing Floorplans

- Chang et. al., “B\*-tree: A new representation for non-slicing floorplans,” DAC-2k.
  1. Compact modules to left and bottom (O-tree).
  2. Construct an ordered binary tree (B\*-tree) (see the paper for more rigid rules!!)
    - Left child: the lowest, adjacent block on the right ( $x_j = x_i + w_j$ ).
    - Right child: the first block above, with the same x-coordinate ( $x_j = x_i$ ).
- 1-to-1 correspondence between a compacted non-slicing floorplan and its induced B\*-tree.



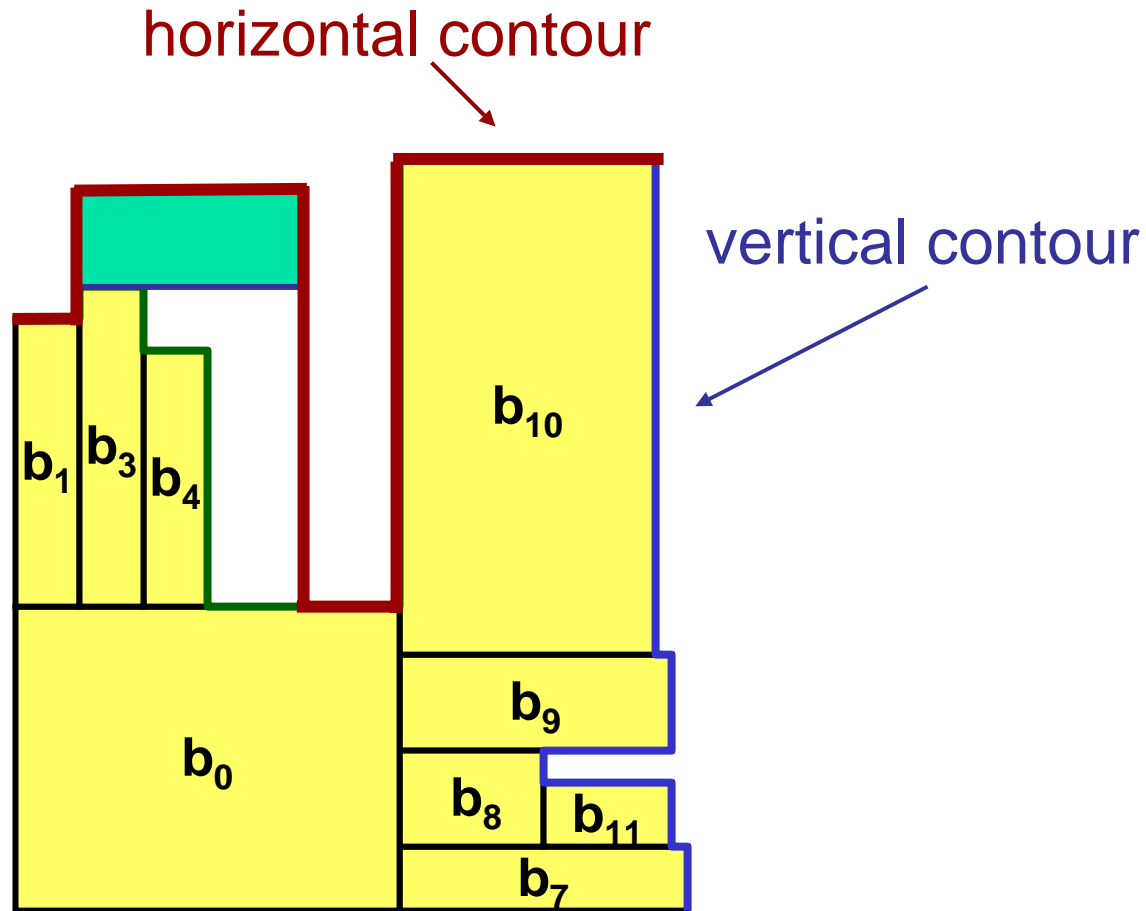
# B\*-tree Packing

- x-coordinates can be determined by the tree structure.
  - Left child: the lowest, adjacent block on the right ( $x_j = x_i + w_i$ ).
  - Right child: the first block above, with the same x-coordinate ( $x_j = x_i$ ).
- y-coordinates?



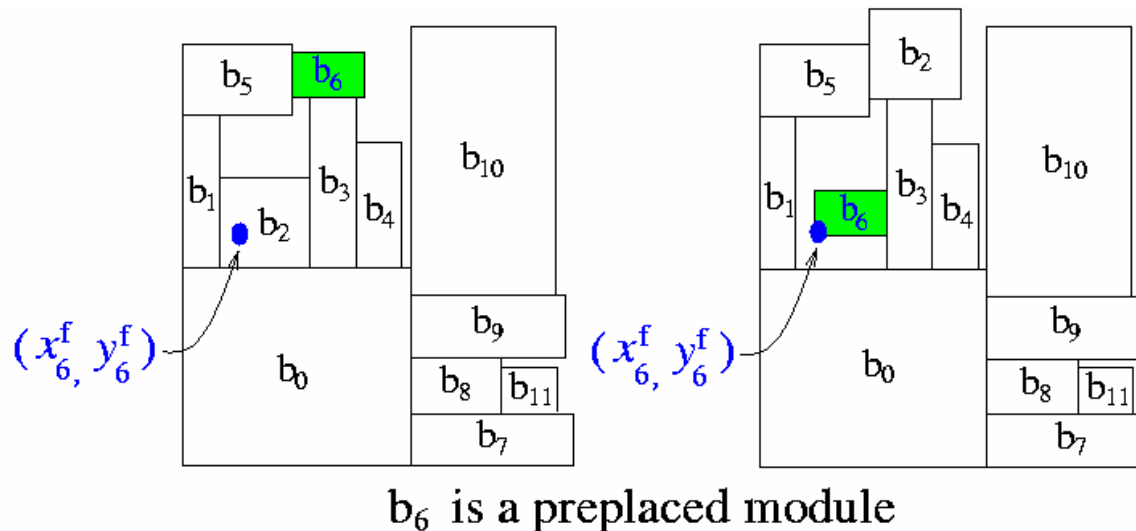
# Computing y-coordinates

- Reduce the complexity of computing a y-coordinate to amortized  $O(1)$  time.



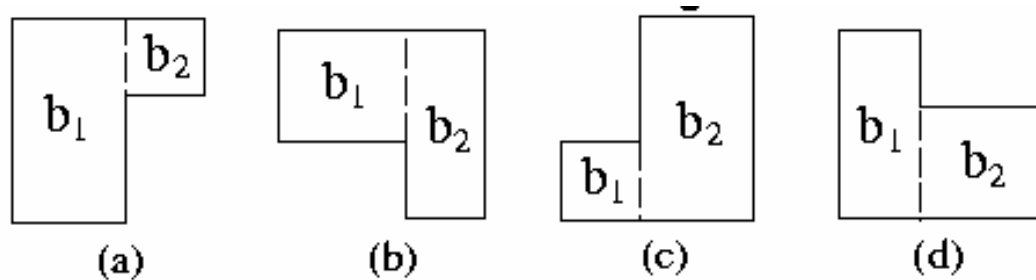
# Coping with Pre-placed Modules

- If there are modules ahead or lower than  $b_i$  so that  $b_i$  cannot be placed at its fixed position  $(x_i^f, y_i^f)$ , exchange  $b_i$  with the module in  $D_i = \{b_j \mid (x_j, y_j) \leq (x_i^f, y_i^f)\}$  that is most close to  $(x_i^f, y_i^f)$ .
- Incremental area cost update is possible.
  - E.g., the positions of  $b_0, b_7, b_8, b_{11}, b_9, b_{10}$ , and  $b_1$  (before  $b_2$  in the DFS order of  $T$ ) remain unchanged after the exchange since they are in front of  $b_2$  in the DFS order.

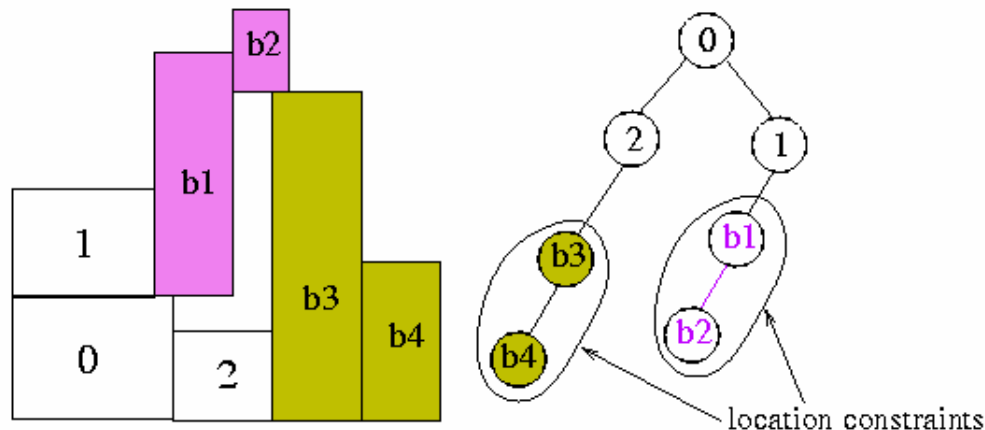


# Coping with Rectilinear Modules

- Partition a rectilinear module into rectangular sub-modules.



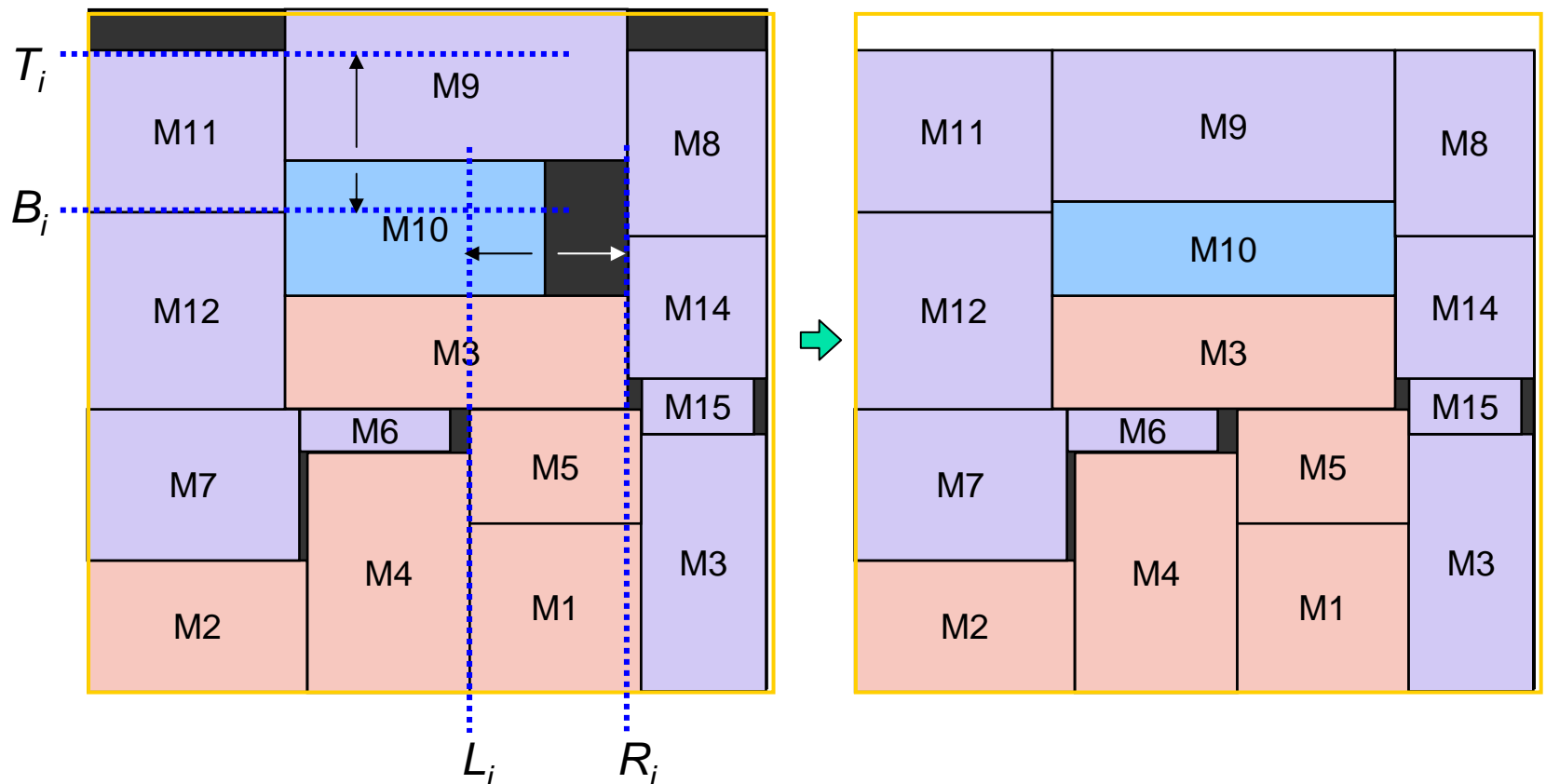
- Keep **location constraints** for the sub-modules.
  - E.g., Keep the right sub-module as the left child in the B\*-tree.
- Align sub-modules, if necessary.
- Treat the sub-modules of a module as a whole during processing.





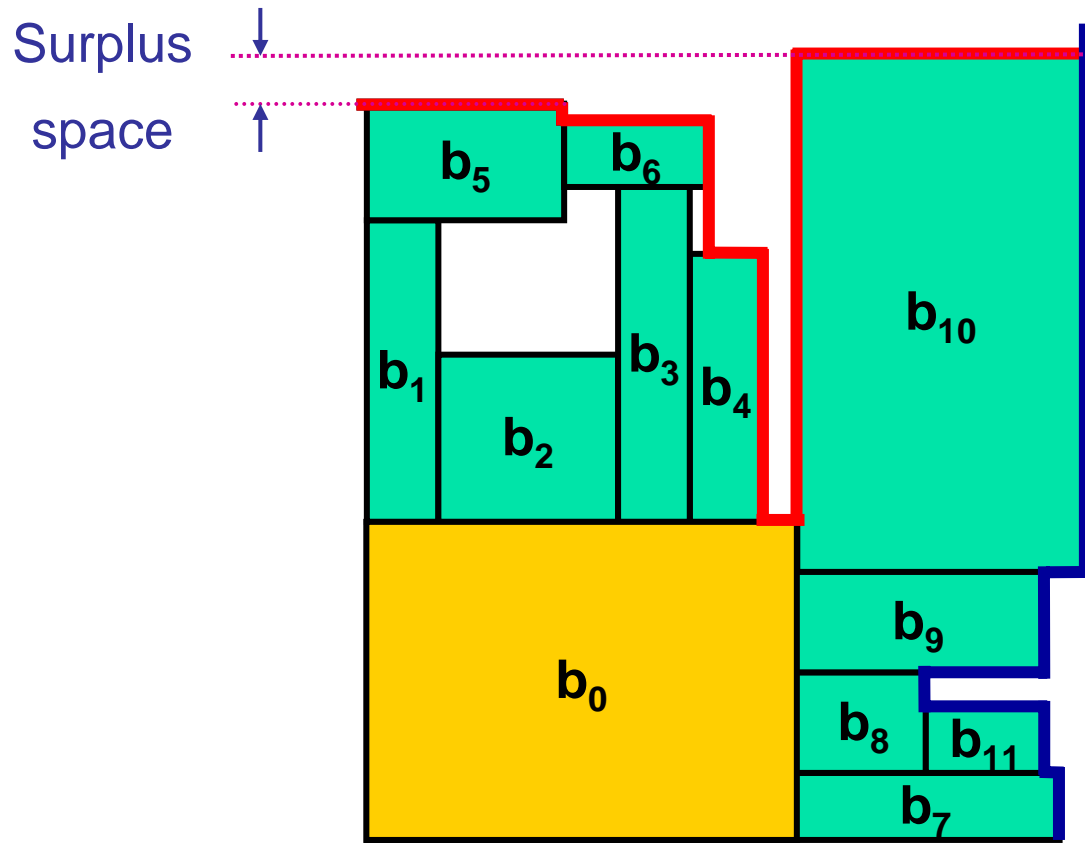
# An Easy Way to Cope with Soft Modules

- Change the shape of a soft module to align with adjacent modules.
- Process soft modules one by one.



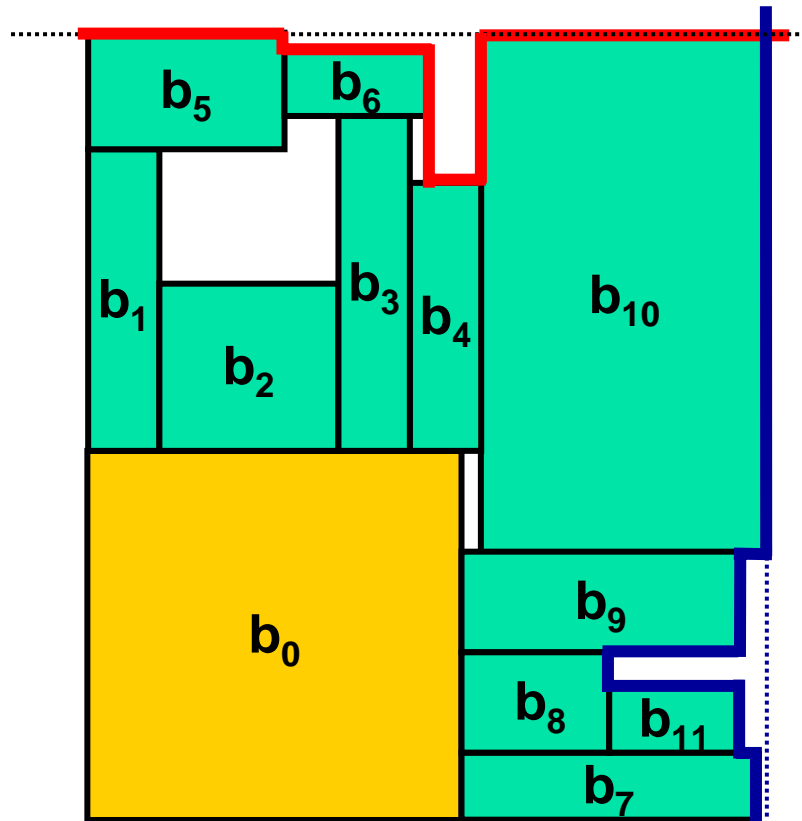
# More on Soft Modules

- Step1: Change the shape of the inserted soft module.
- Step2: Change the shapes of other soft modules.



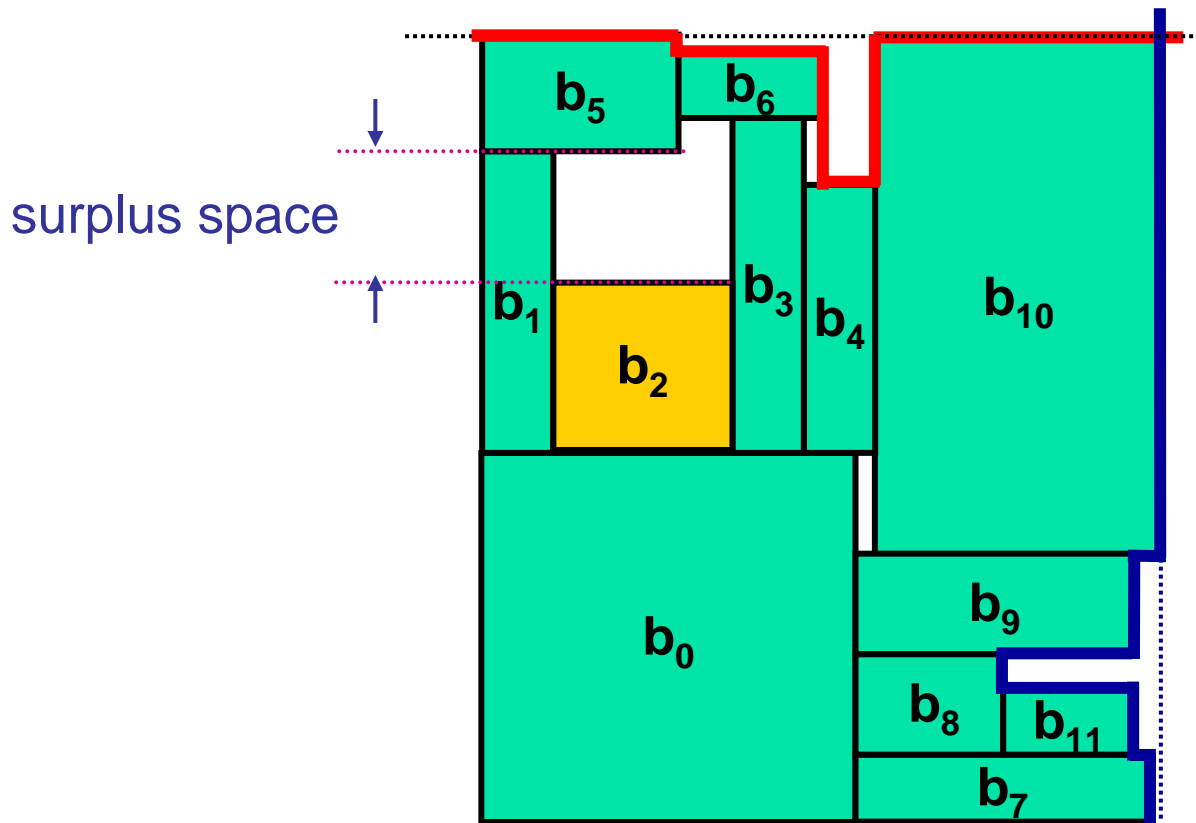
# More on Soft Modules

- Step1: Change the shape of the inserted soft module
- Step2: Change the shape of other soft modules



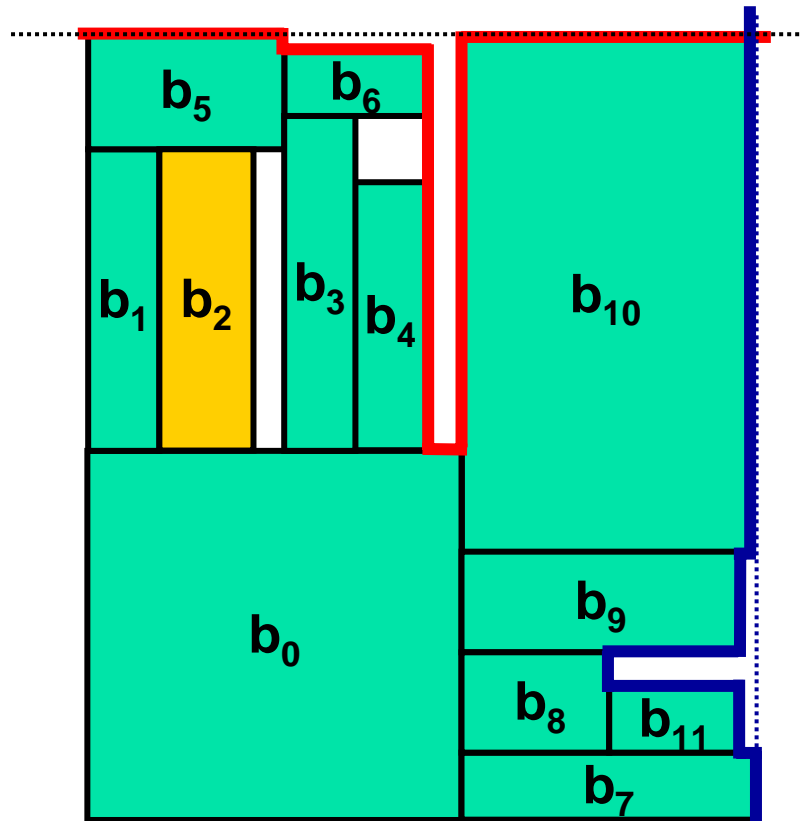
# More on Soft Modules

- Step1: Change the shape of the inserted soft module
- Step2: Change the shapes of other soft modules



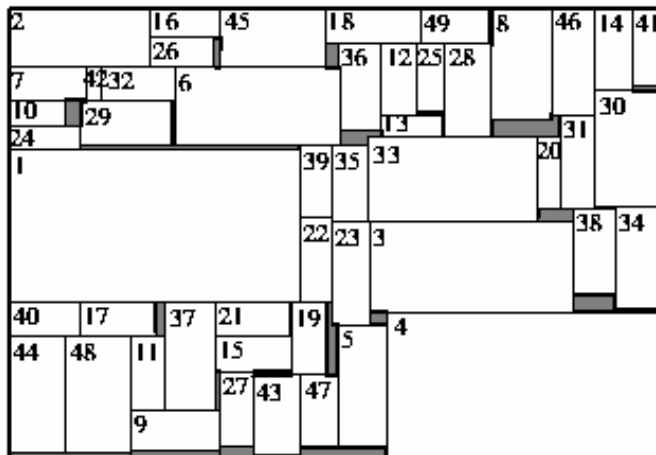
# More on Soft Modules

- Step1: Change the shape of the inserted soft module
- Step2: Change the shape of other soft modules

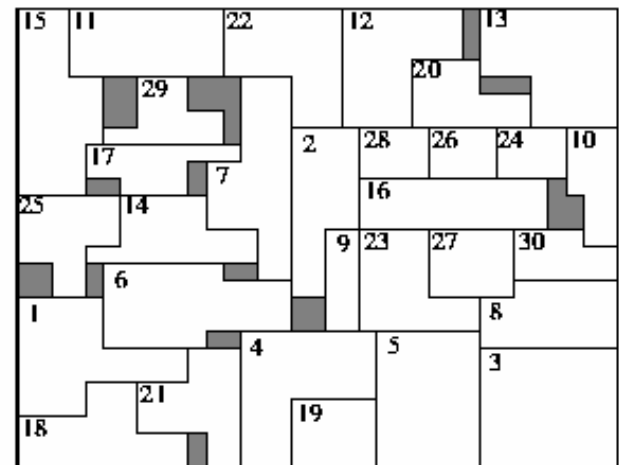


# Perturbations & Solutions

- Perturbing B\*-trees in simulated annealing
  - Op1: Rotate a module.
  - Op2: Flip a module.
  - Op3: Move a module to another place.
  - Op4: Swap two modules.
- ami49: Area =  $36.74 \text{ mm}^2$ ; dead space = 3.53%; CPU time = 0.25 min on SUN Ultra 60 (optimum =  $35.445 \text{ mm}^2$ ).



ami49



Rectangular, L-, and T-shaped modules