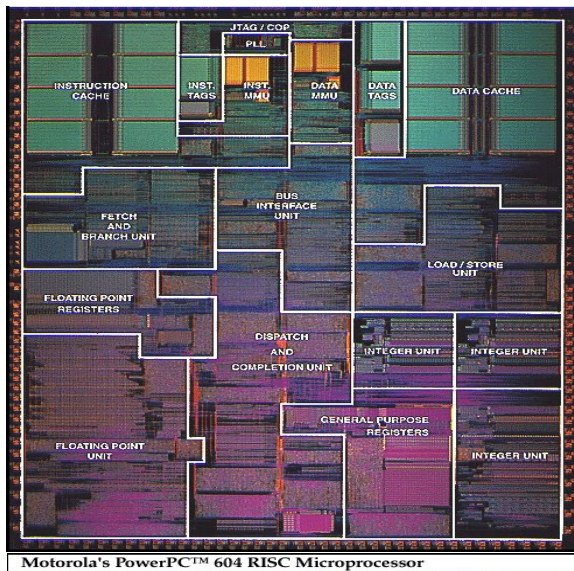
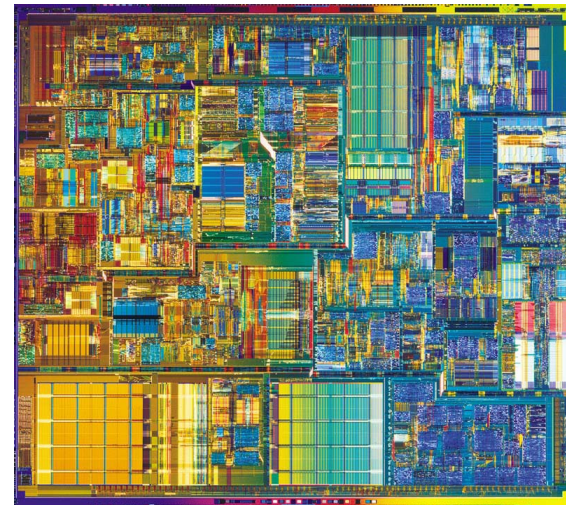


Unit 4: Floorplanning

- Course contents
 - Floorplanning basics
 - Normalized Polish expression for slicing floorplans
 - B*-trees for non-slicing floorplans
 - Sequence pair



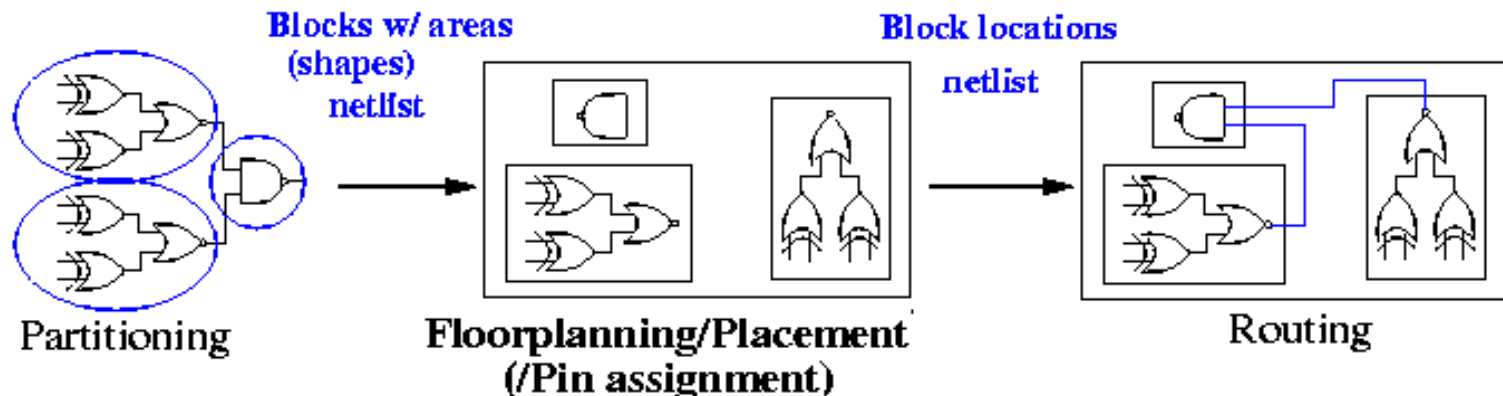
PowerPC 604



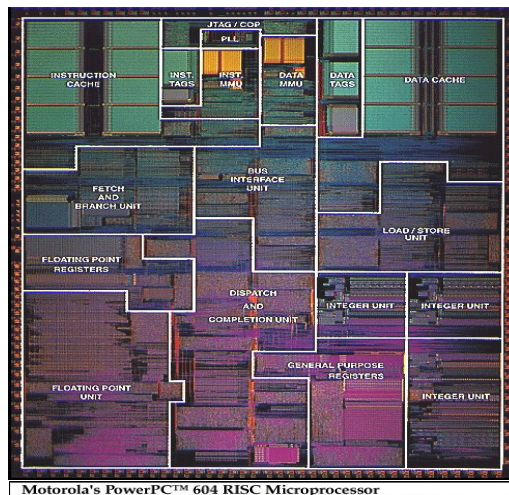
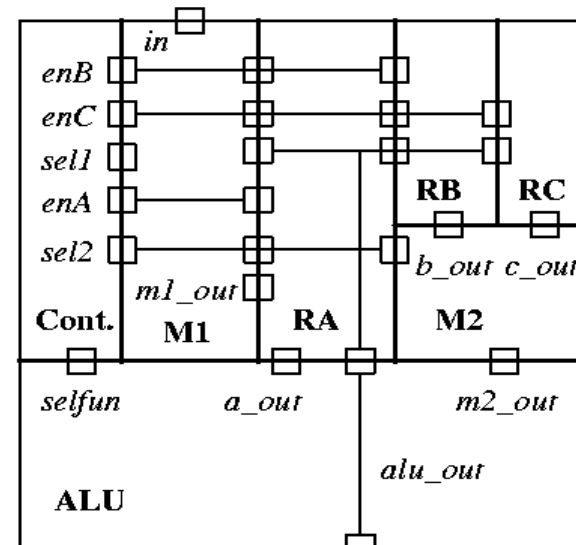
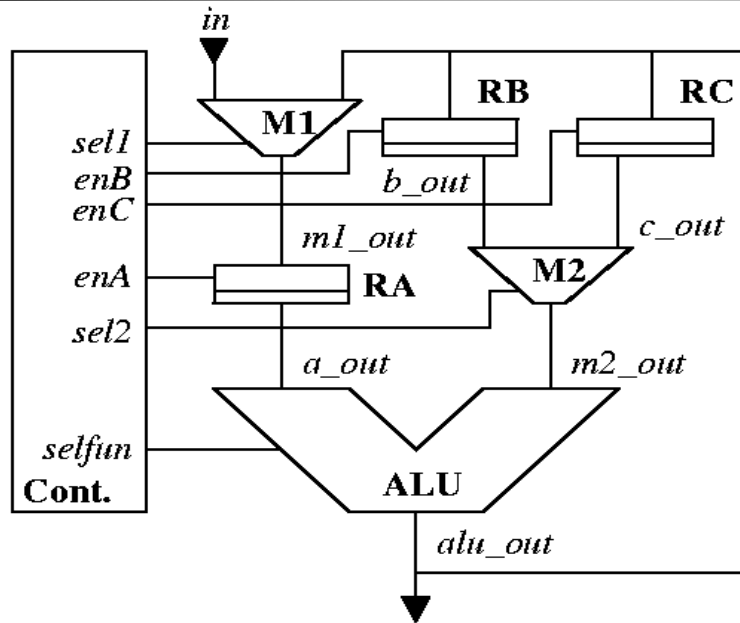
Pentium 4

Floorplanning

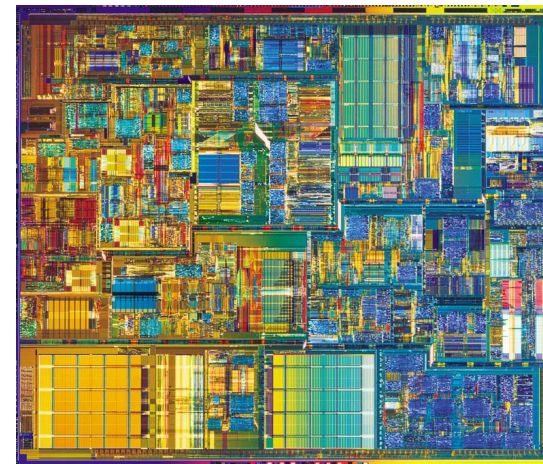
- Partitioning leads to
 - Blocks with well-defined **areas and shapes** (rigid/hard blocks).
 - Blocks with approximate areas and no particular shapes (flexible/soft blocks).
 - A **netlist** specifying connections between the blocks.
- Objectives
 - Find **locations** for all blocks.
 - Consider shapes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



Early Layout Decision Example



PowerPC 604



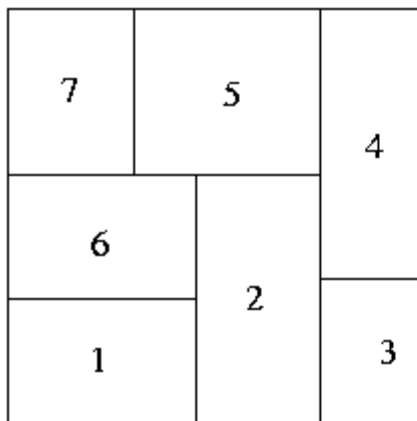
Pentium 4

Early Layout Decision Methodology

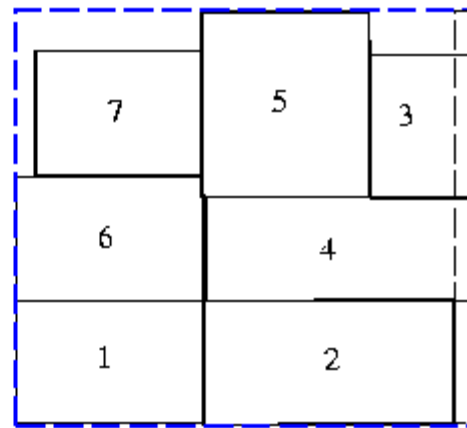
- An IC is a 2-D medium; considering the dimensions of blocks in early stages of the design helps to improve the quality.
- Floorplanning gives early feedback
 - Suggests valuable architectural modifications
 - Estimates the whole chip area
 - Estimates delay and congestion due to wiring
- Floorplanning fits very well in a *top-down* design strategy; the *step-wise refinement* strategy also propagated in software design.
- Floorplanning considers the *flexibility* in the shapes and terminal locations of blocks.

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.

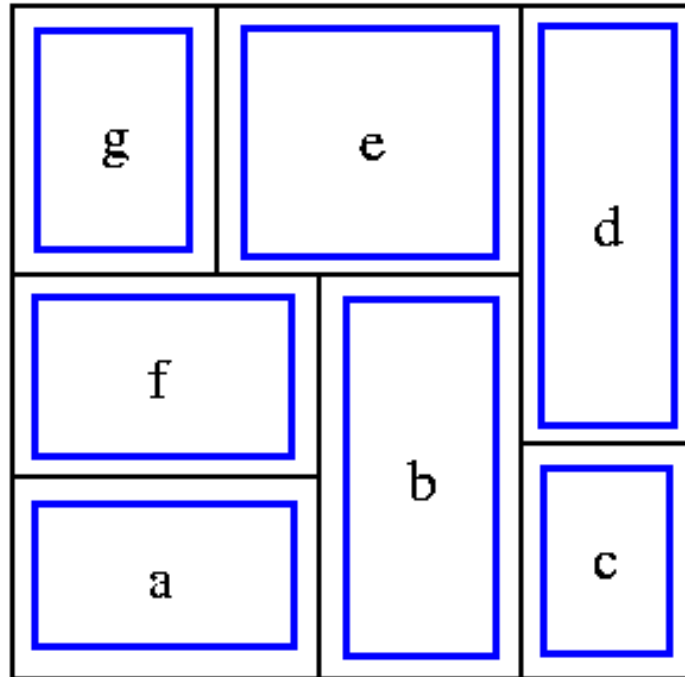


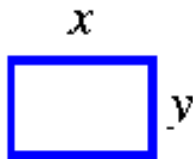


An optimal floorplan,
in terms of area

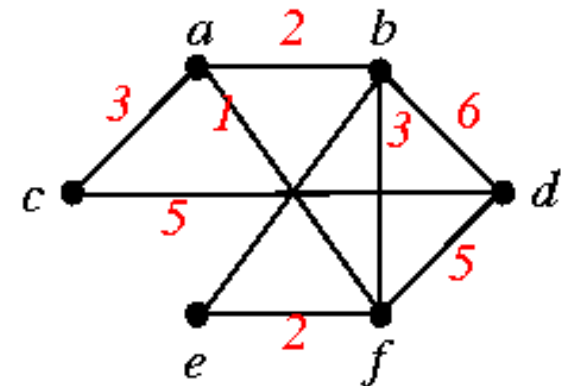


A non-optimal floorplan

Floorplan Design

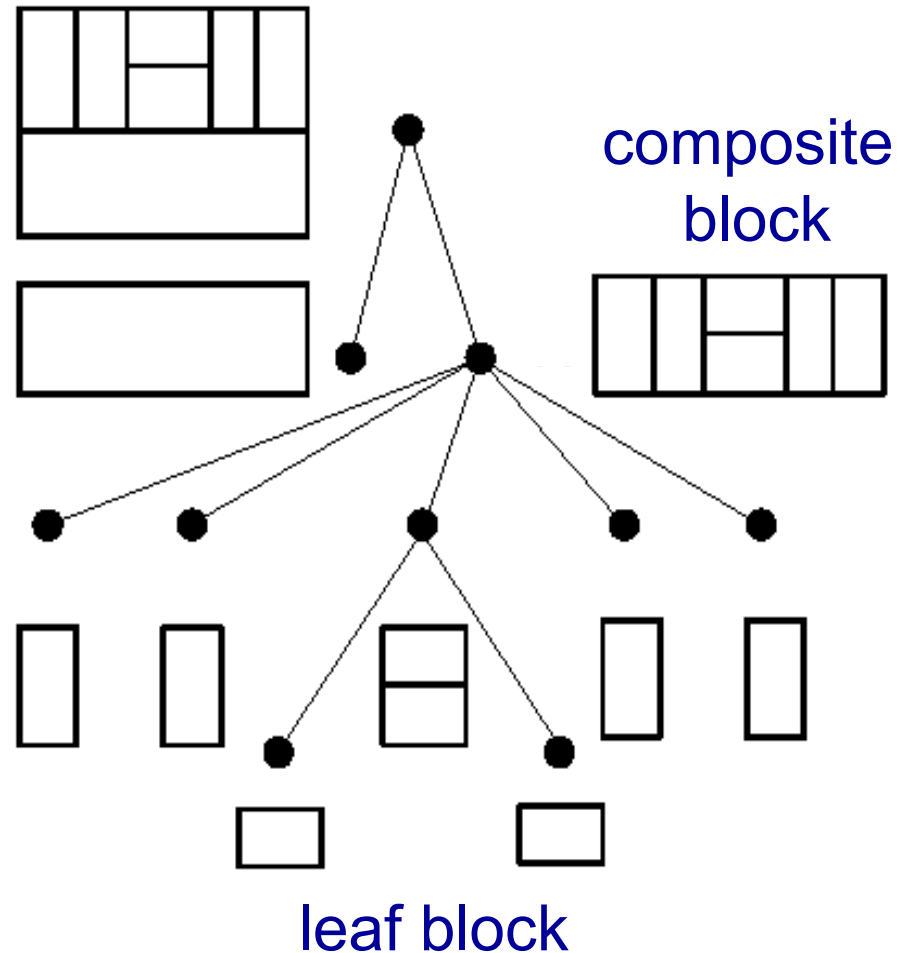


- *Modules:* 
- *Area:* $A=xy$
- *Aspect ratio:* $r \leq y/x \leq s$
- *Rotation:*  
- *Module connectivity*



Floorplan Elements

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

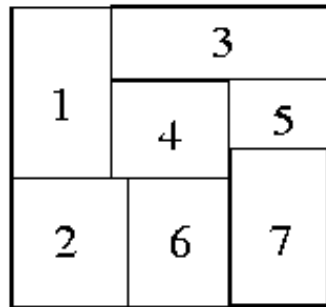


-

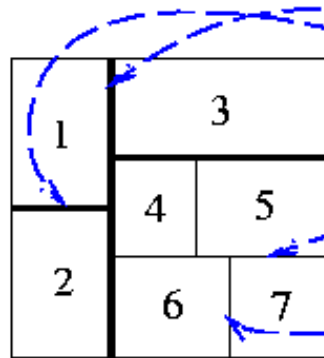
V: vertical cut

Skewed Slicing Tree

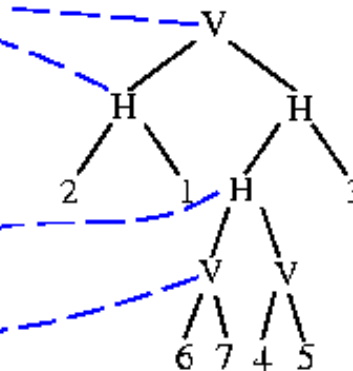
- **Rectangular dissection:** Subdivision of a given rectangle by a finite # of horizontal and vertical line segments into a finite # of non-overlapping rectangles.
- **Slicing structure:** a rectangular dissection that can be obtained by repetitively subdividing rectangles horizontally or vertically.
- **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.



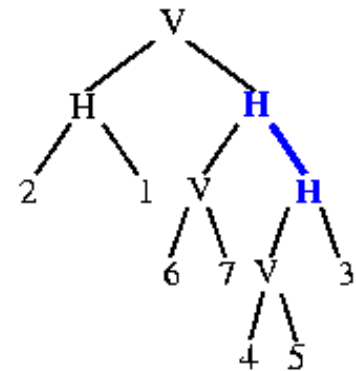
Non-slicing floorplan



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.
 - 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?

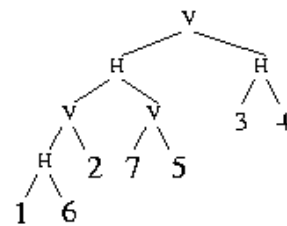
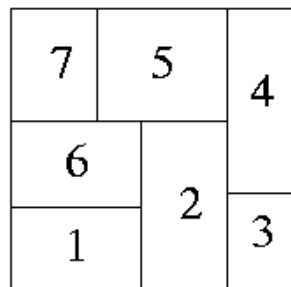
Solution Representation

- 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 .



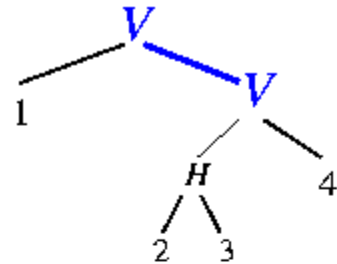
$E = 16H2V75VH34HV$

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

Postorder traversal of a tree!

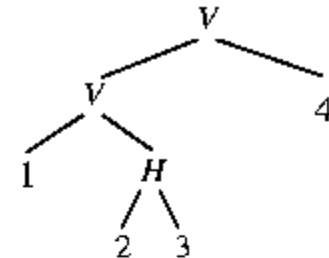
Redundant Representations

1	3	4
	2	



$E = 123H4VV$

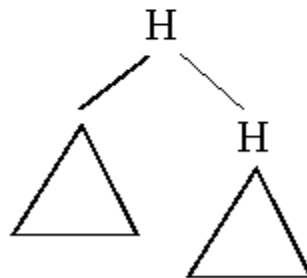
non-skewed!



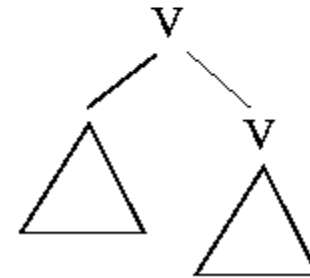
$E = 123HV4V$

skewed!

**Non-skewed
cases**



..... HH

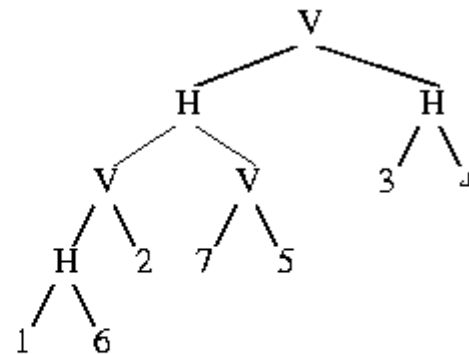
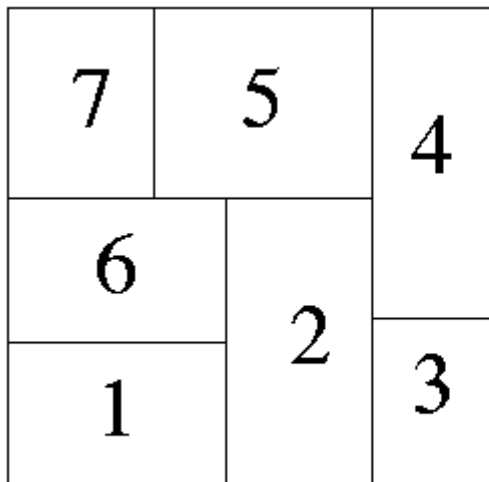


..... VV

- **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.

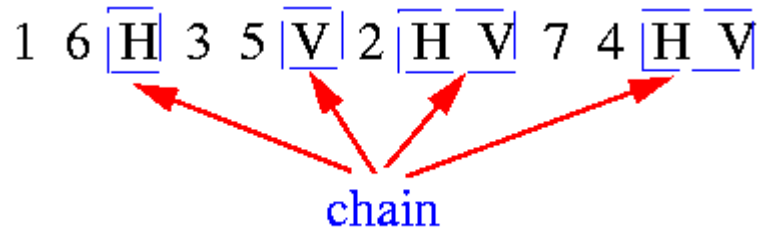


$E = 16H2V75VH34HV$

A normalized Polish expression

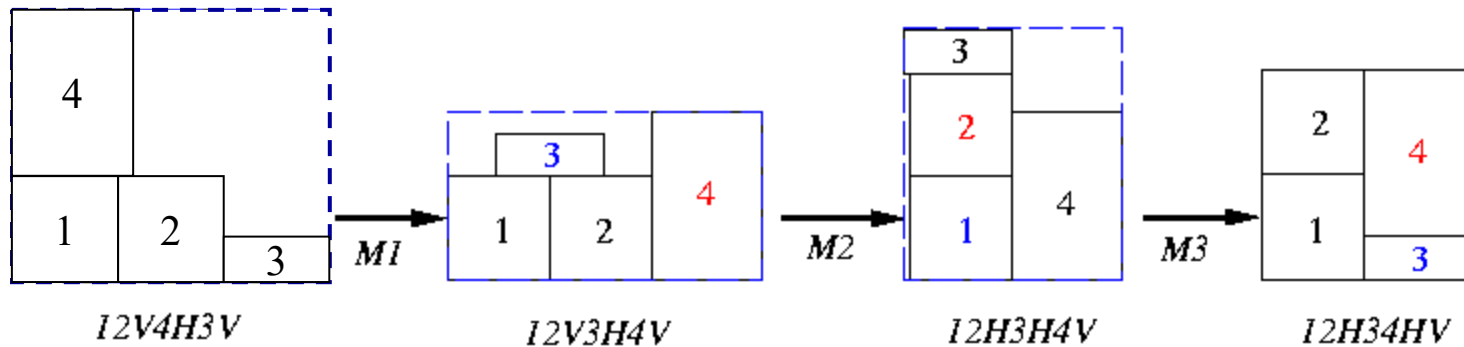
Neighborhood Structure

- **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 ($V = H$, $H = V$).
 - **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 *M3* 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 $E = e_1 e_2 \dots e_k$, $1 \leq k \leq 2n-1$

Illegal M3 Moves

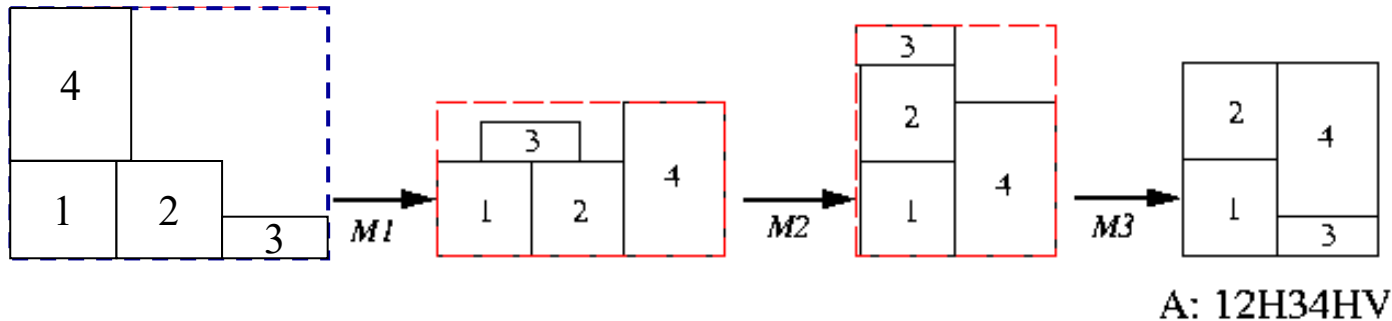
- Assume that *M3* swaps the operand e_i with the operator e_{i+1} , $1 \leq i \leq k-1$.

		e_1	e_2	...	e_{i-1}	e_i	e_{i+1}
						operand	operator
before	#operands				n	n+1	n+1
	#operators				n-1	n-1	n
swap						operator	operand
after	#operands				n	n	n+1
	#operators				n-1	n	n

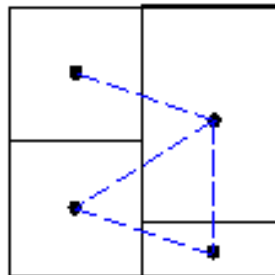
Violate!
Condition:
 $i=2n=2N_{i+1}$

Cost Function

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

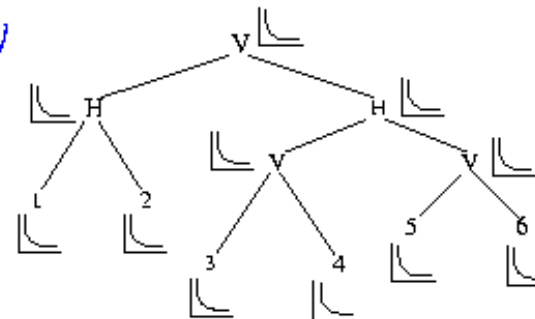
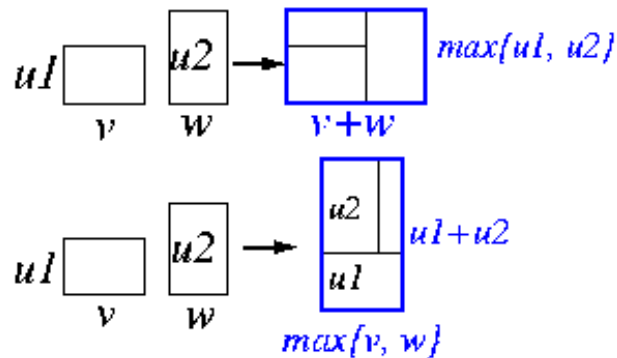
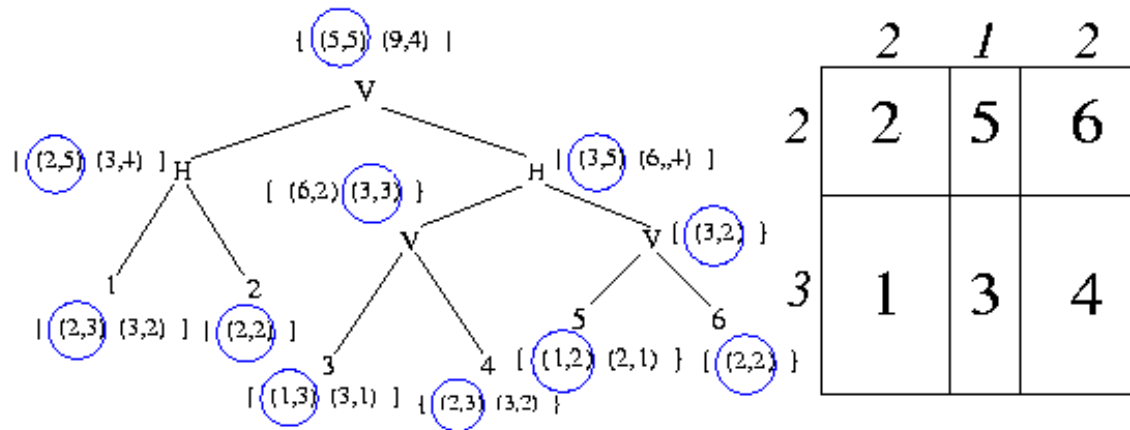


- $W = \sum_{ij} c_{ij} d_{ij}$.
 - c_{ij} : # of connections between blocks i and j .
 - d_{ij} : center-to-center distance between basic rectangles i and j .



Area Computation for Hard Blocks

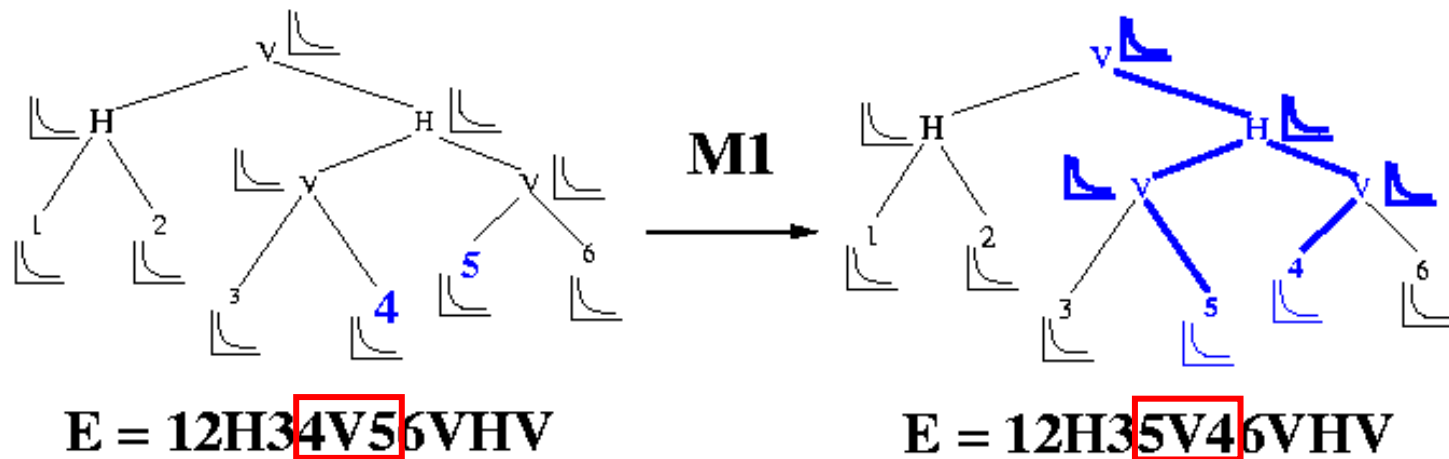
- Allow rotation



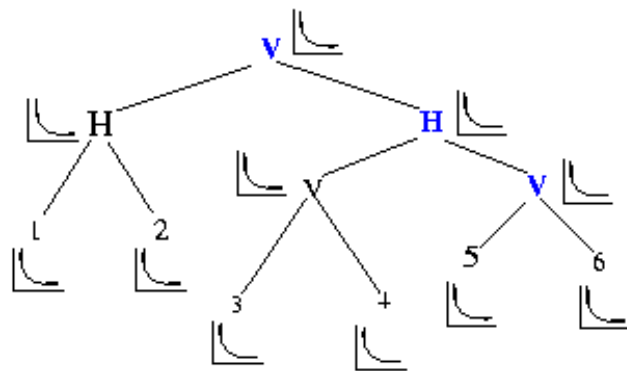
- Wiring cost?
 - Center-to-center interconnection length

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.

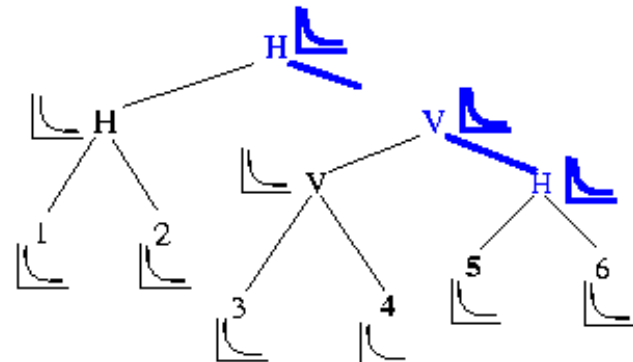


Incremental Computation of Cost Function (cont'd)

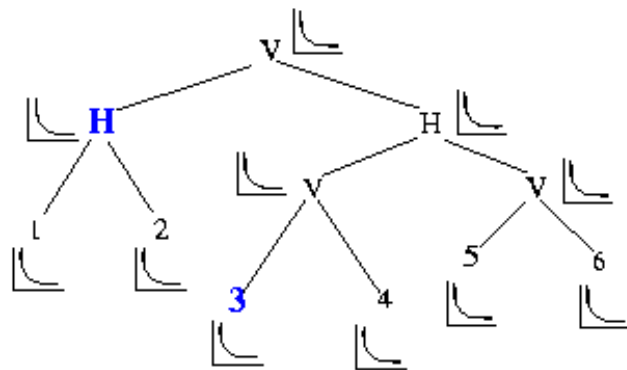


E = 12H34V56VHV

M2

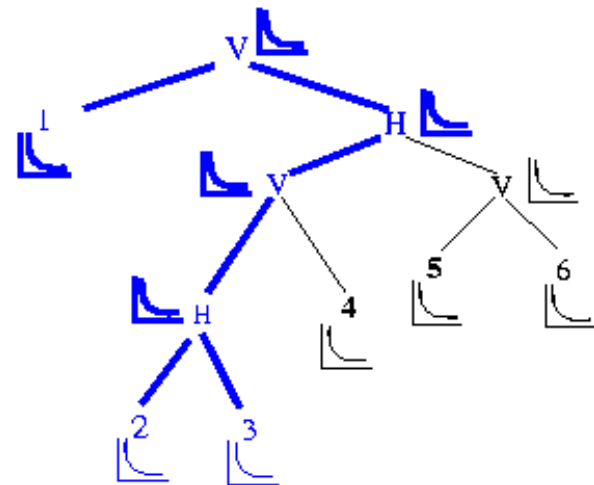


E = 12H34V56HVH



E = 12H34V56VHV

M3



E = 123H4V56VHV

Annealing Schedule

- Initial solution: 12V3V ... 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, ε, r, k)

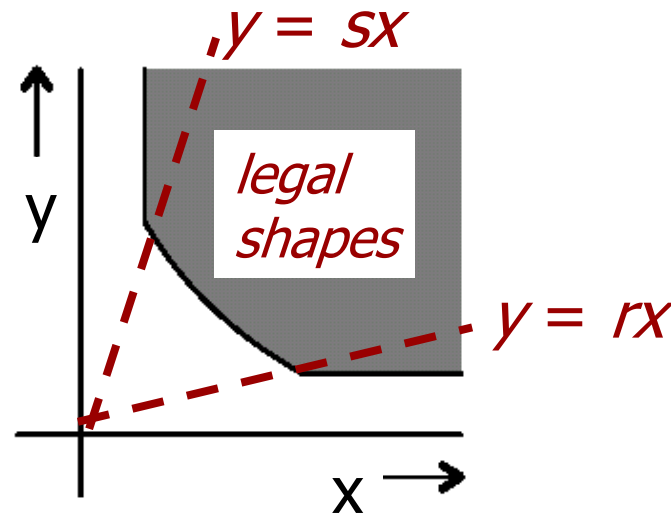
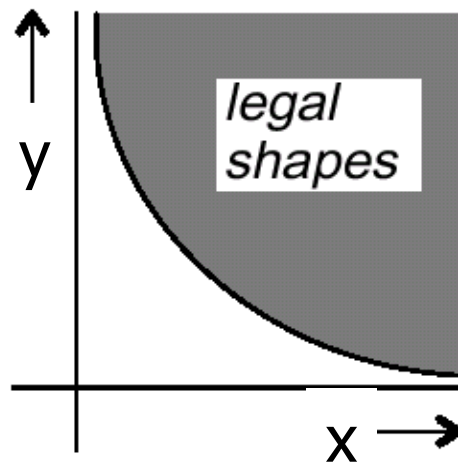
```

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    $MT \leftarrow uphill \leftarrow reject \leftarrow 0$ ;
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  $\leftarrow \text{TRUE}$ ;
15        Select two adjacent operator  $e_i$  and operand  $e_{i+1}$ ;
16        if ( $e_i \neq e_{i+2}$ ) then done  $\leftarrow \text{TRUE}$ ;
17         $NE \leftarrow \text{Swap}(E, e_i, e_{i+1})$ ;
18         $MT \leftarrow MT+1$ ;  $\Delta cost \leftarrow cost(NE) - cost(E)$ ;
19        if ( $\Delta cost \leq 0$ ) or ( $\text{Random} < \frac{-\Delta cost}{e^T}$ )
20          then
21            if ( $\Delta cost > 0$ ) then  $uphill \leftarrow uphill + 1$ ;
22             $E \leftarrow NE$ ;
23            if  $cost(E) < cost(best)$  then  $best \leftarrow E$ ;
24            else  $reject \leftarrow reject + 1$ ;
25          until ( $uphill > N$ ) or ( $MT > 2N$ );
26         $T \leftarrow rT$ ; /* reduce temperature */
27      until ( $reject/MT > 0.95$ ) or ( $T < \varepsilon$ ) or OutOfTime;
28 end

```

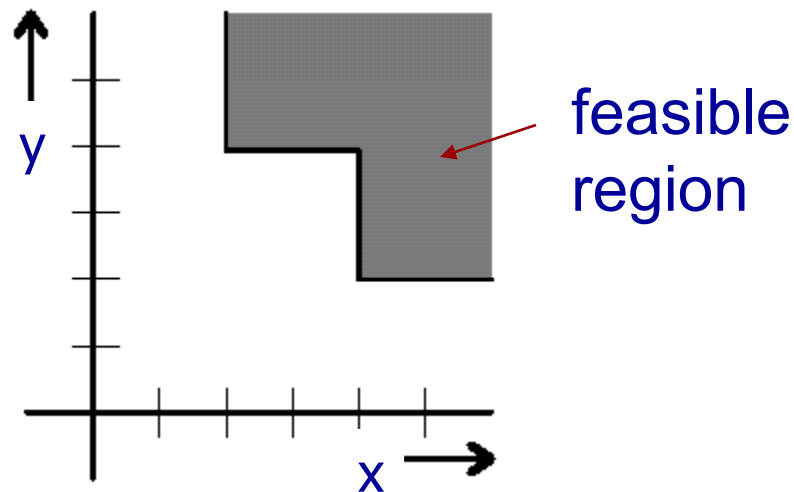
Shape Curve for Floorplan Sizing

- A soft (flexible) blocks b can have different aspect ratios, but is with a fixed area A .
- The shape function of b is a hyperbola: $xy = A$, or $y = A/x$, for width x and height y .
- Very thin blocks are often not interesting and feasible to design; add two straight lines for the constraints on aspect ratios.
 - Aspect ratio: $r \leq y/x \leq s$.



Shape Curve

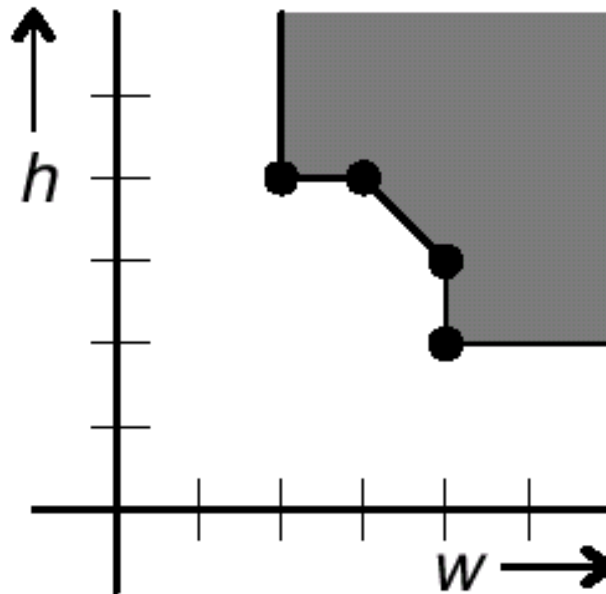
- Since a basic block is built from discrete transistors, it is not realistic to assume that the shape function follows the hyperbola continuously.
- In an extreme case, a block is rigid/hard: it can only be rotated and mirrored during floorplanning or placement.



The shape curve of a 2×4 hard block.

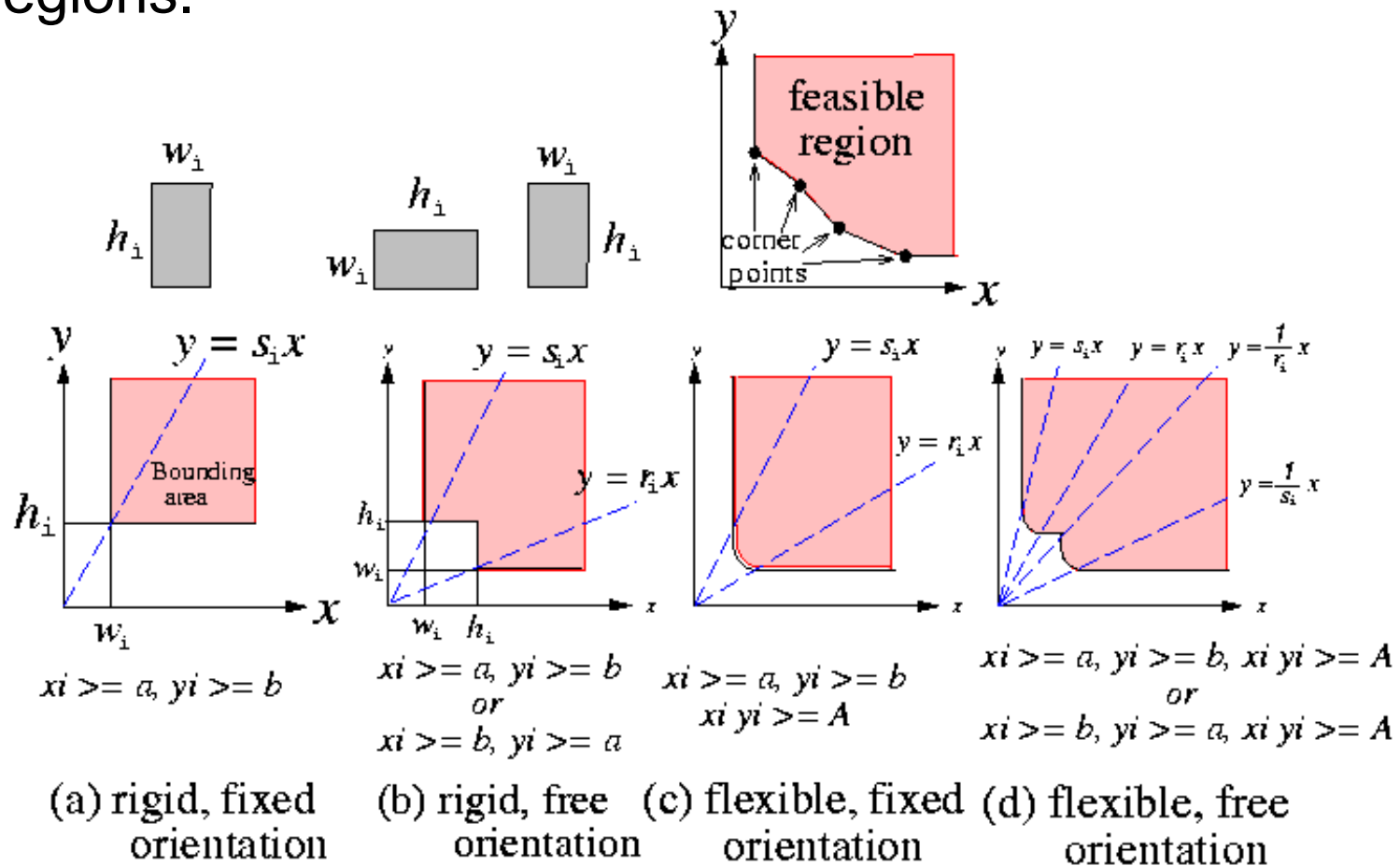
Shape Curve (cont'd)

- 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.



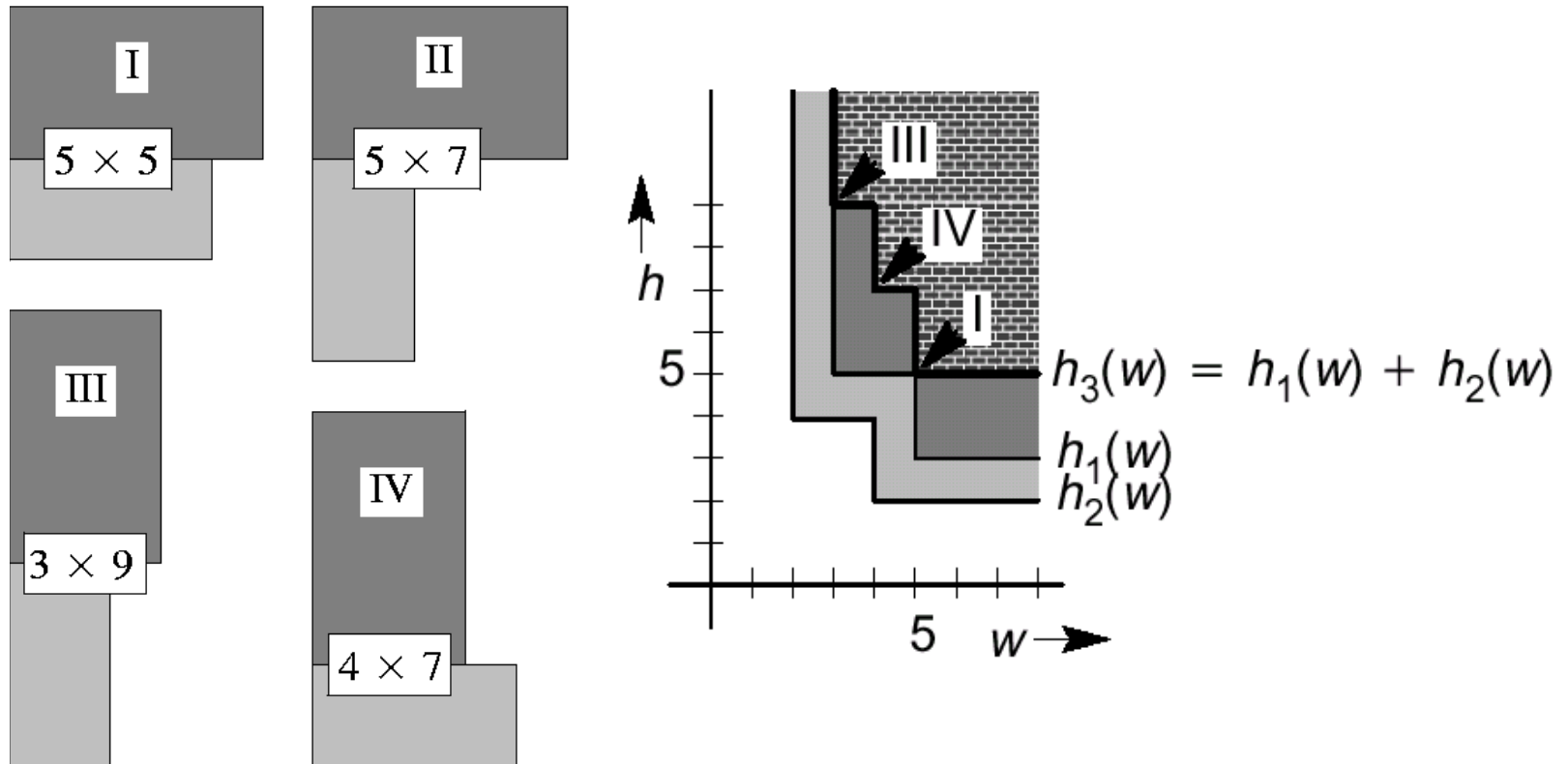
Feasible Implementations

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



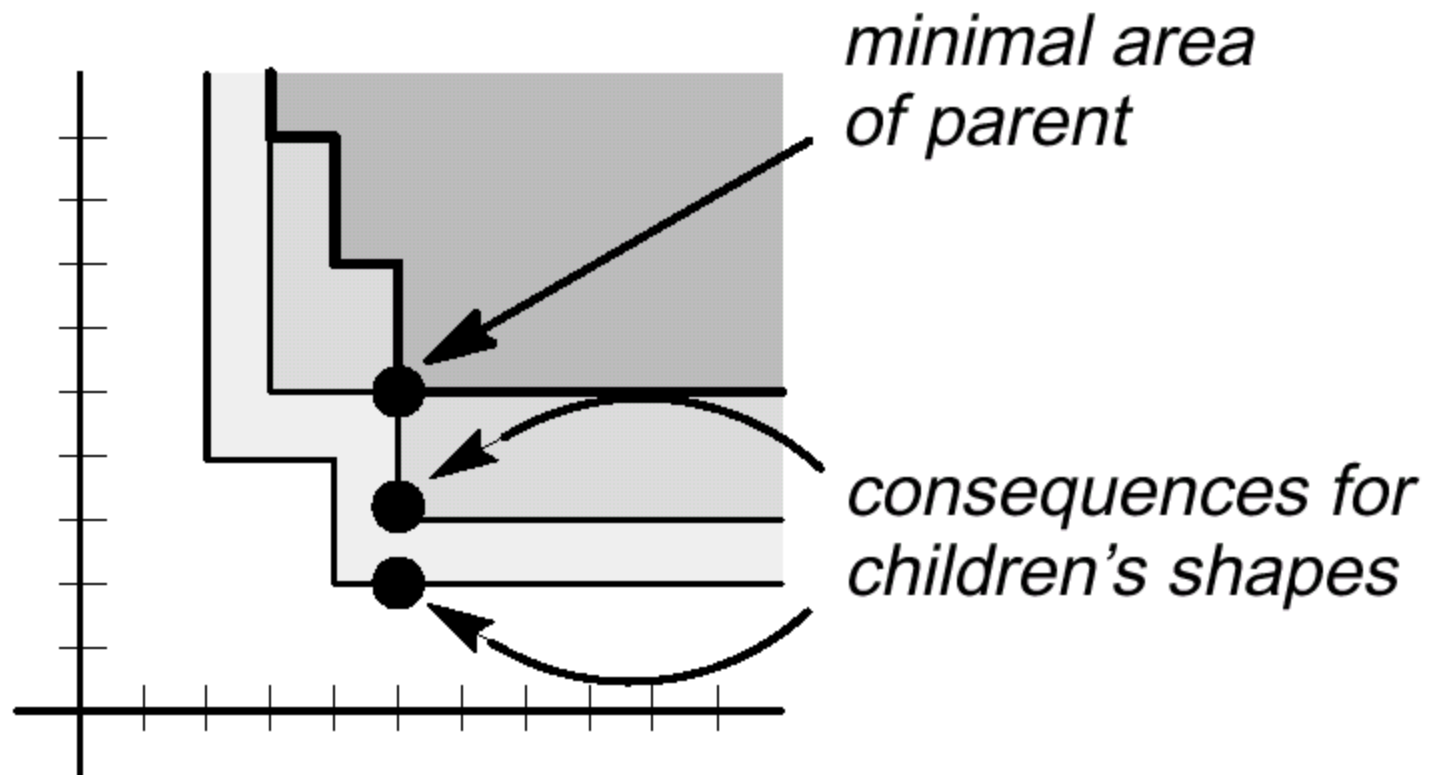
Vertical Abutment

- Composition by vertical abutment (horizontal cut) \Rightarrow the addition of shape functions.



Deriving Shapes of Children

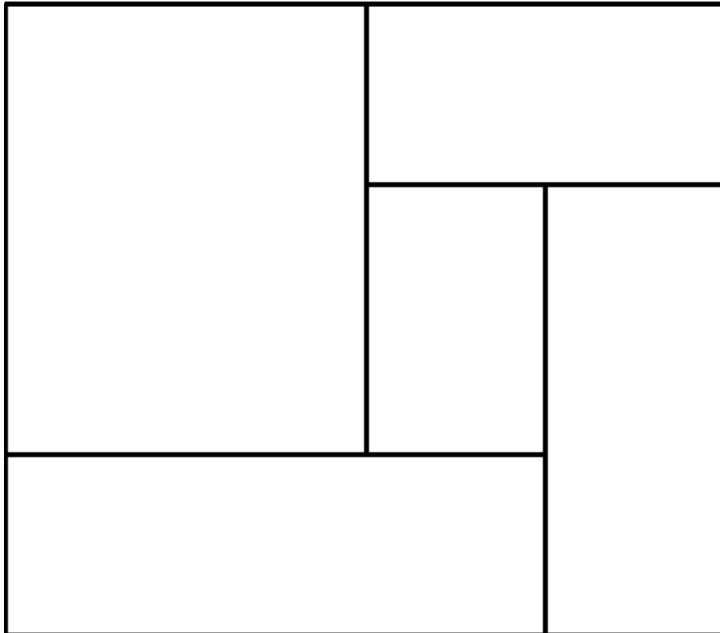
- A choice for the minimal shape of a composite block fixes the shapes of the shapes of its children blocks.



Slicing Floorplan Sizing

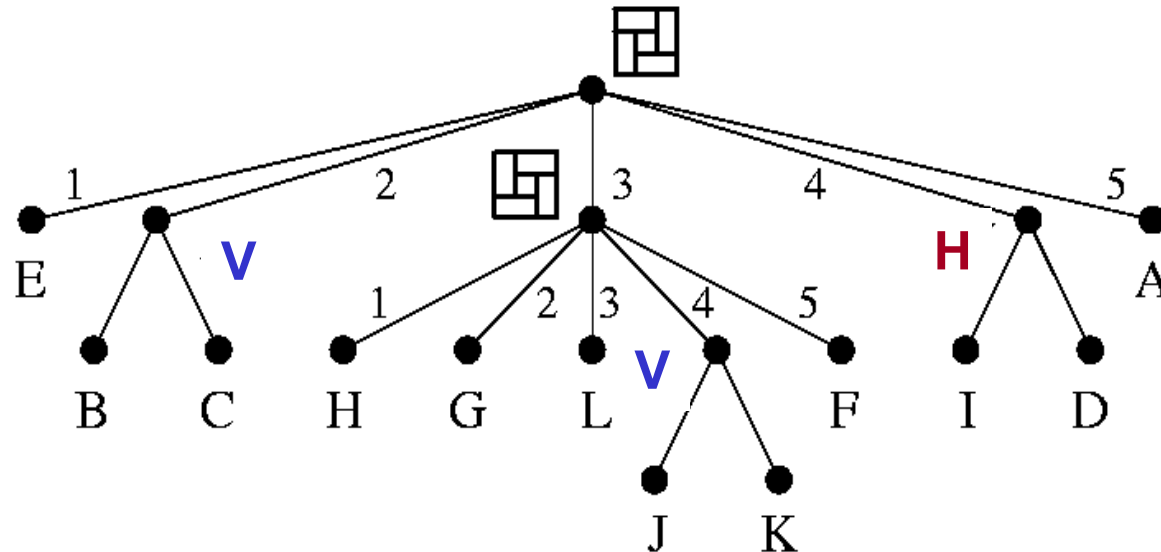
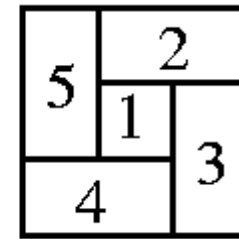
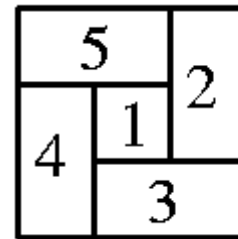
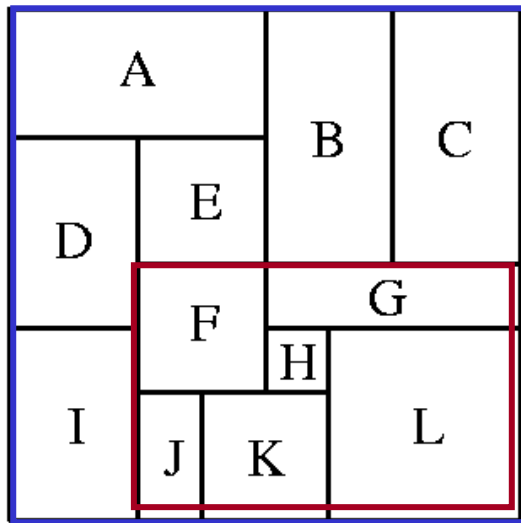
- The shape functions of all leaf blocks are given as piecewise linear functions.
- Traverse the slicing tree to compute the shape functions of all composite blocks (**bottom-up composition**).
- Choose the desired shape of the top-level block; as the shape function is piecewise linear only the corner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

Wheel or Spiral Floorplan



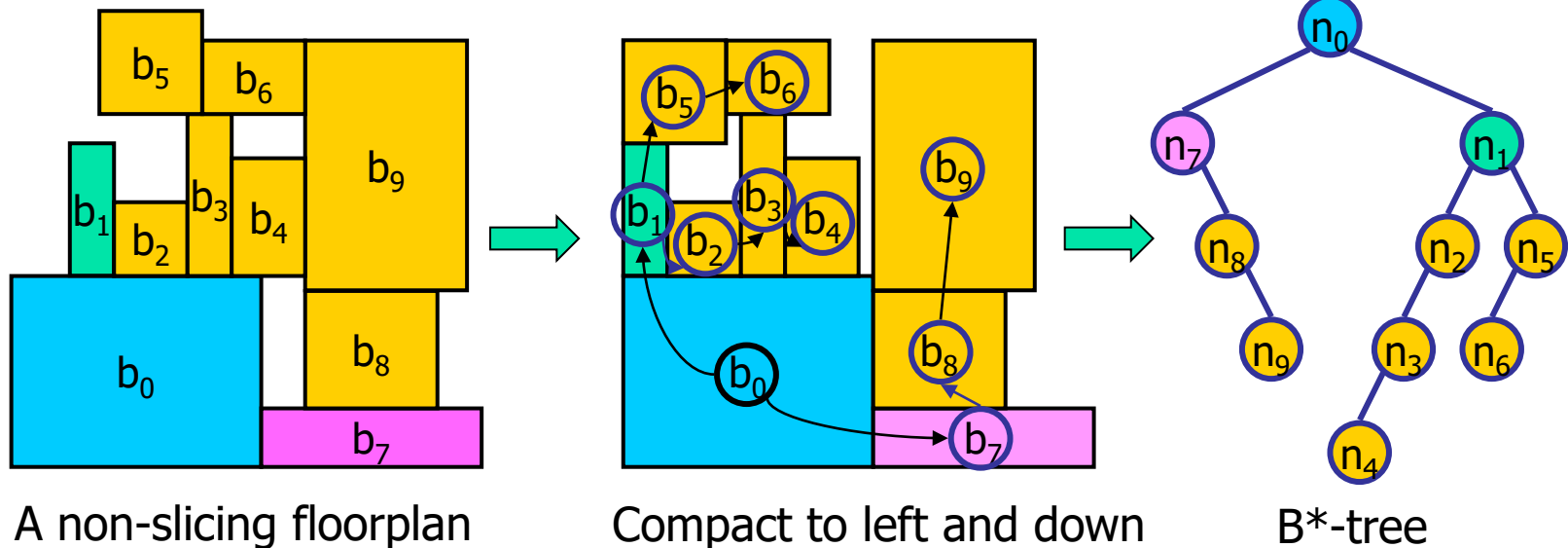
- This floorplan is not slicing!
- **Wheel** is the smallest non-slicing floorplans.
- Limiting floorplans to those that have the slicing property can facilitate floorplanning algorithms.
- Taking the shape of a wheel floorplan and its mirror image as the basis of operators leads to hierarchical descriptions of *order 5*.

Order-5 Floorplan Examples



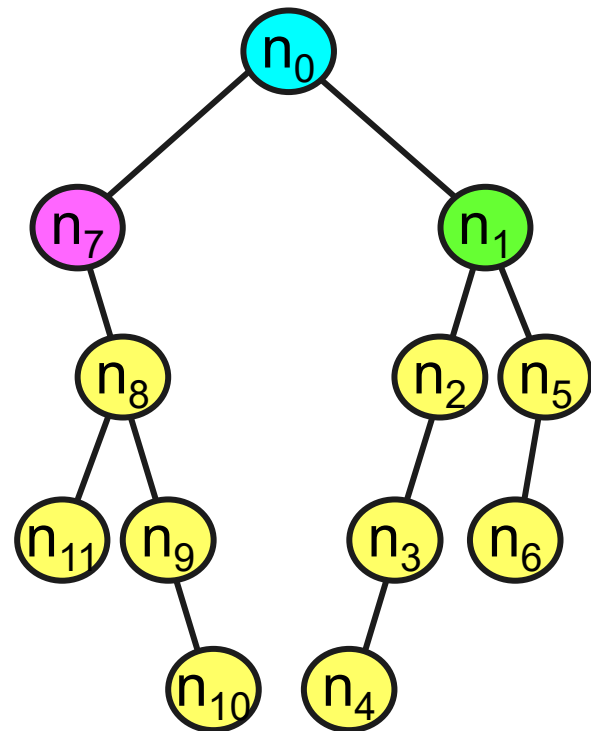
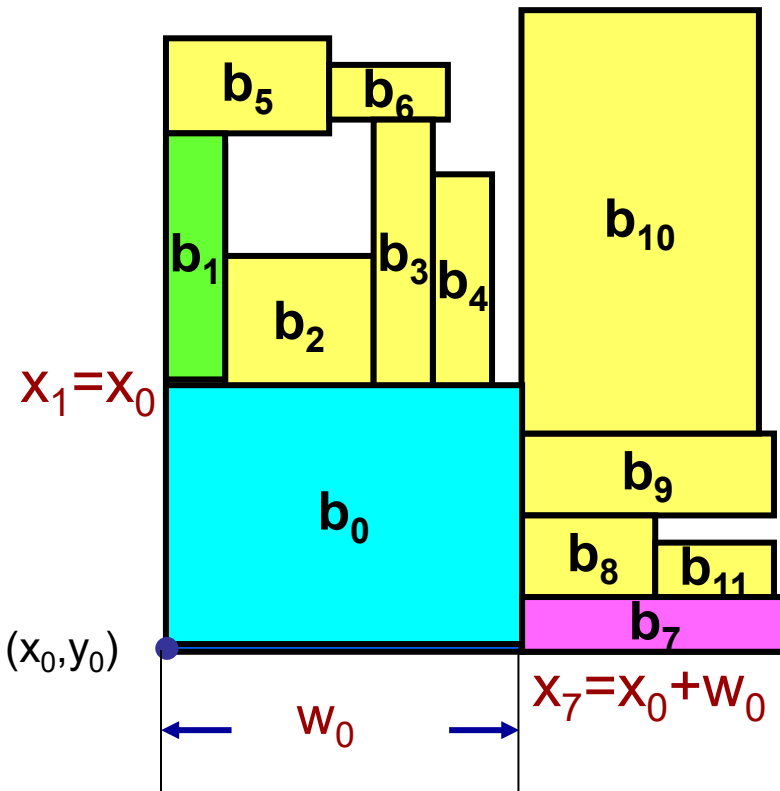
B*-Tree: Compacted Floorplan Representation

- Chang et. al., “B-tree: A new representation for non-slicing floorplans,” DAC-2k.
 - Compact modules to left and bottom.
 - Construct an ordered binary tree (B*-tree).
 - 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 ($x_j = x_i$).



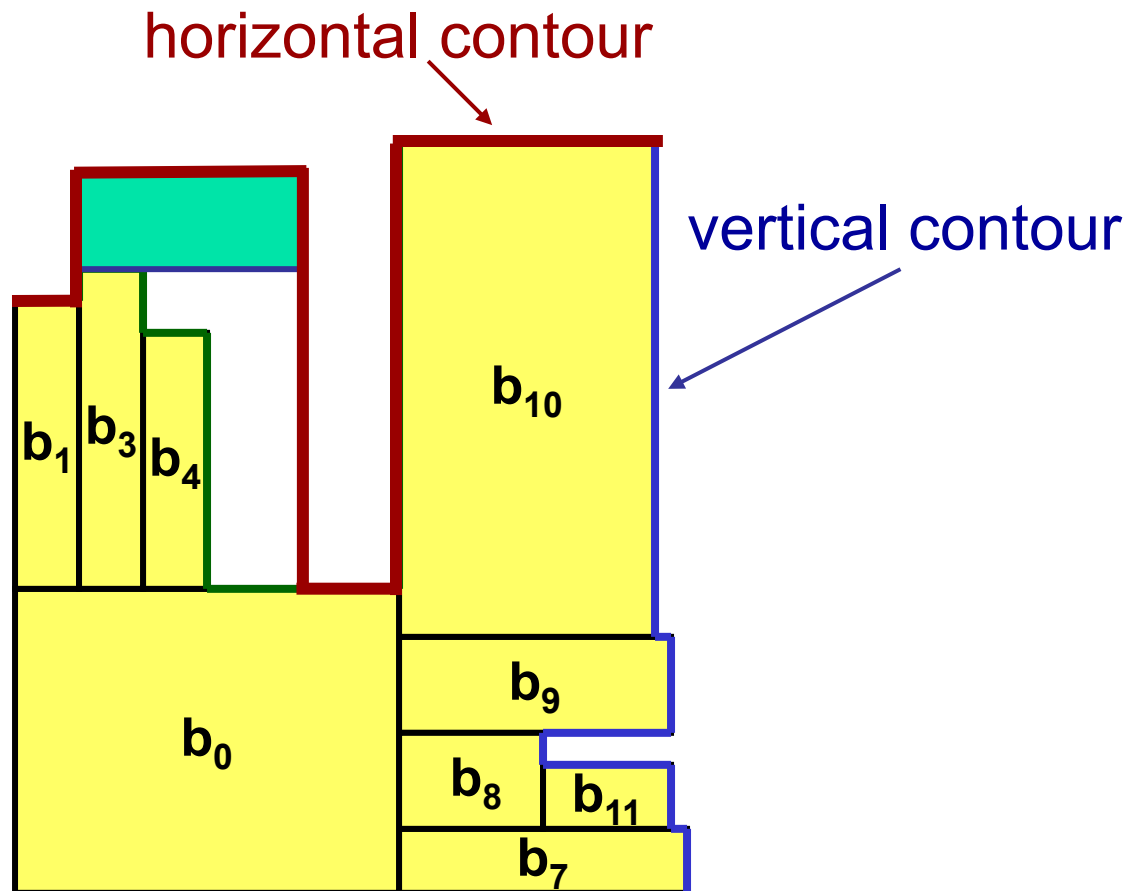
B*-tree Packing

- 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?

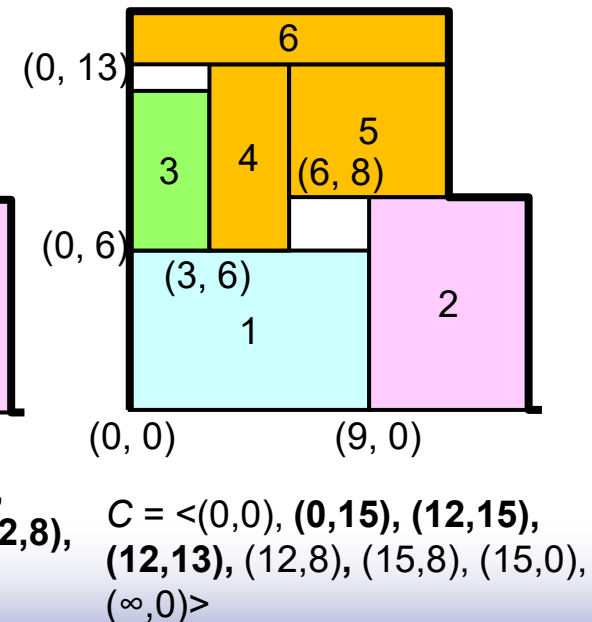
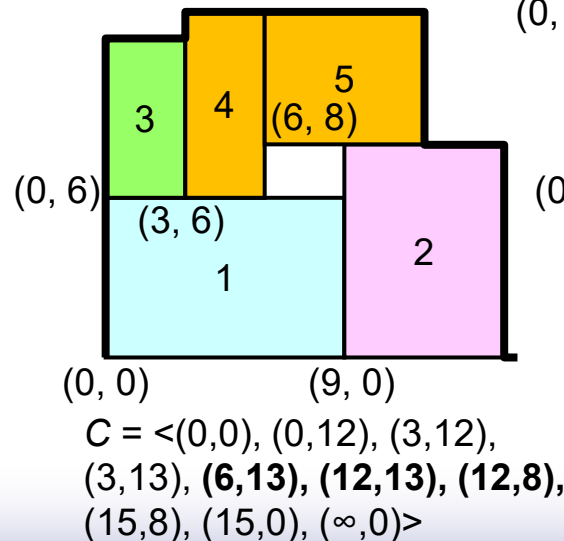
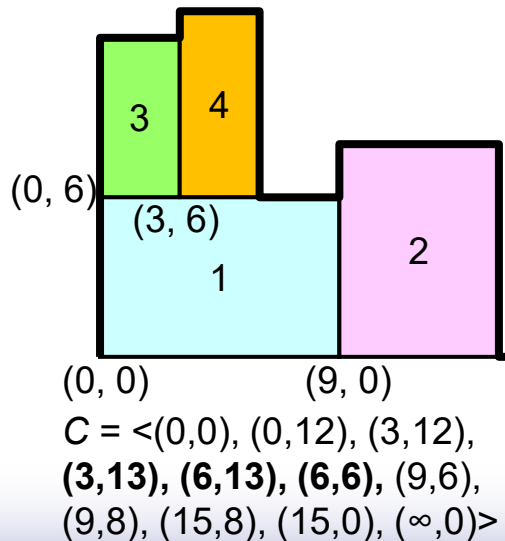
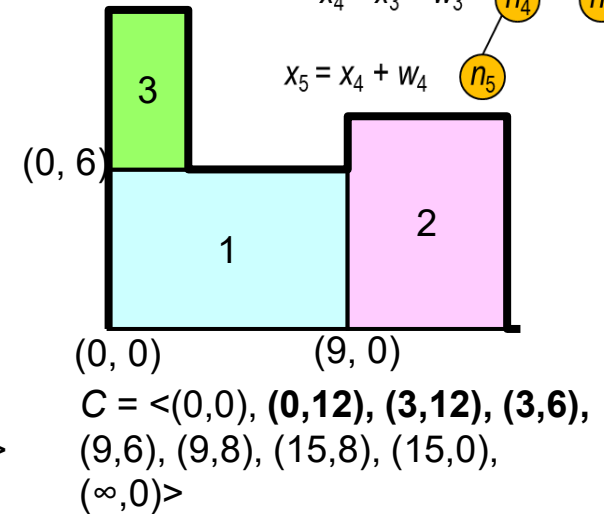
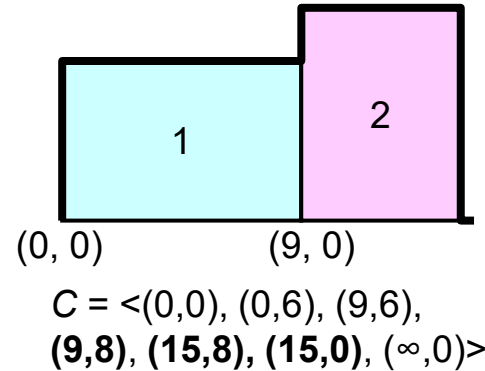
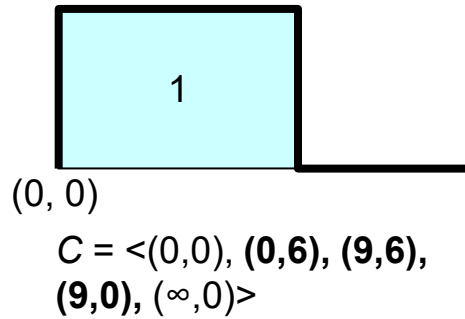
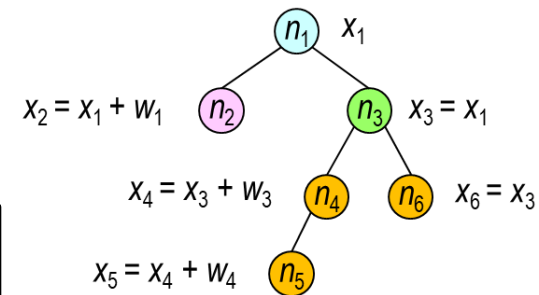


Computing y-coordinates

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

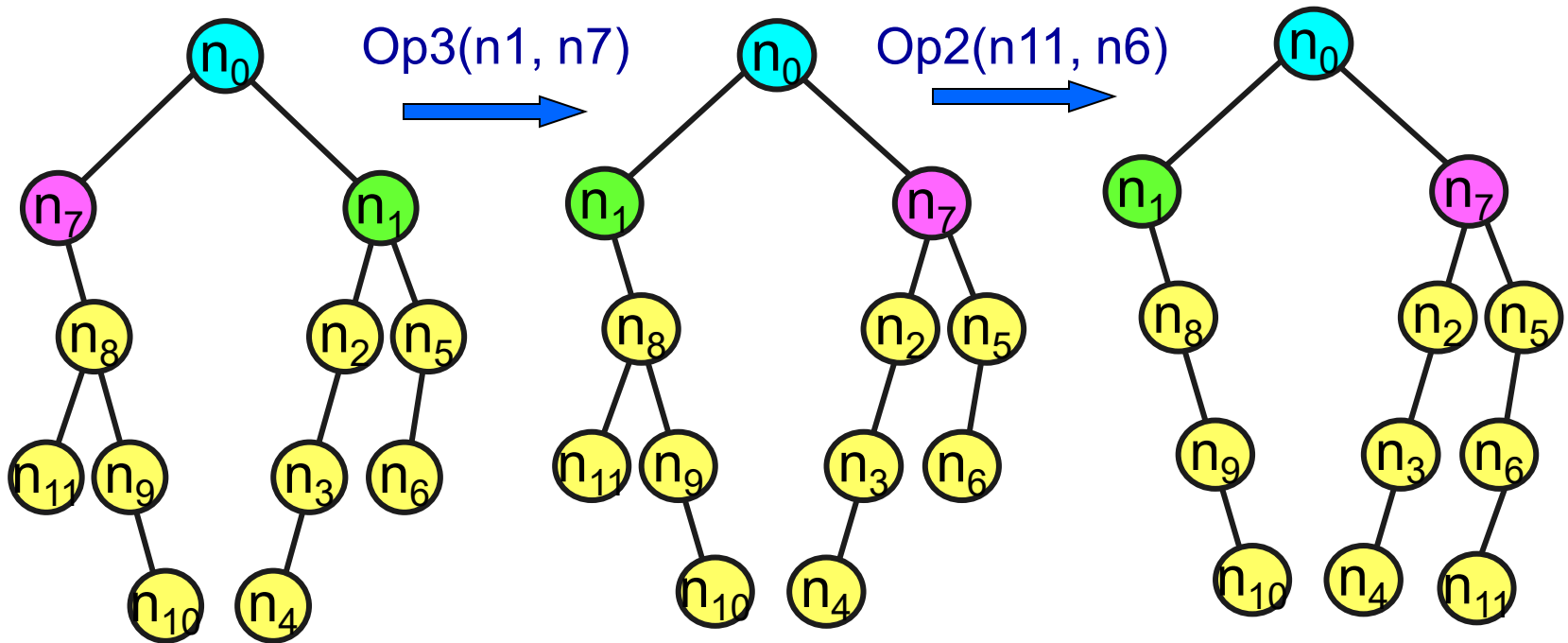


Contour Data Structure



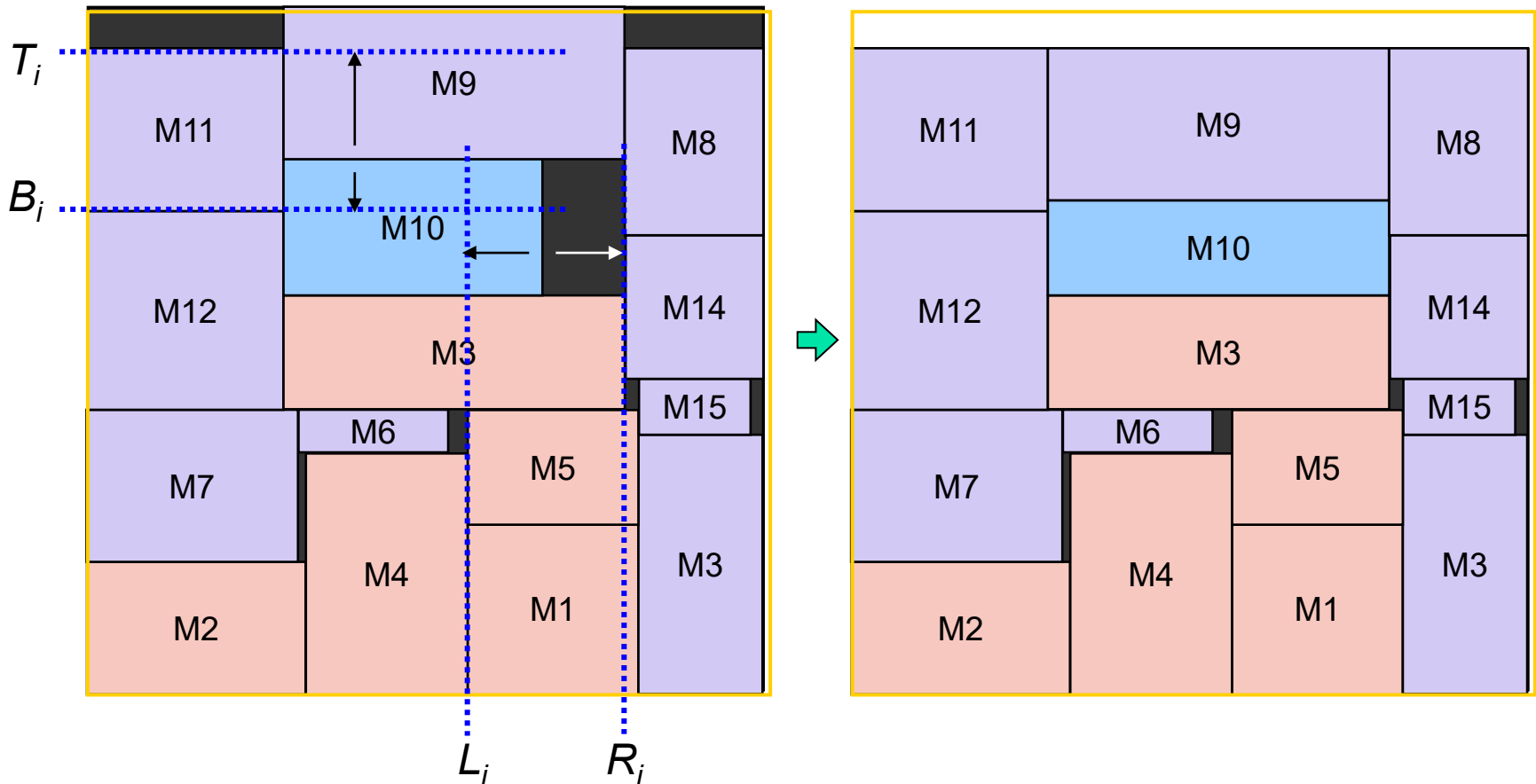
B*-Tree Perturbation

- Op1: rotate a macro
- Op2: delete & insert
- Op3: swap 2 nodes
- Op4: resize a soft macro



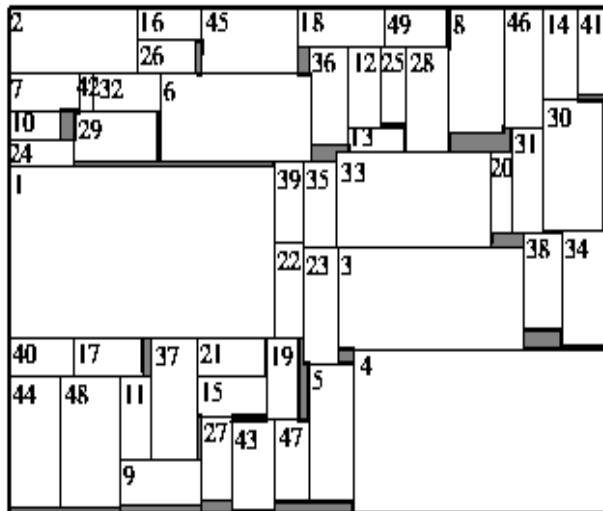
Simple Floorplan Sizing

- Key: Line up with adjacent modules
- Advantage: fast and reasonably effective

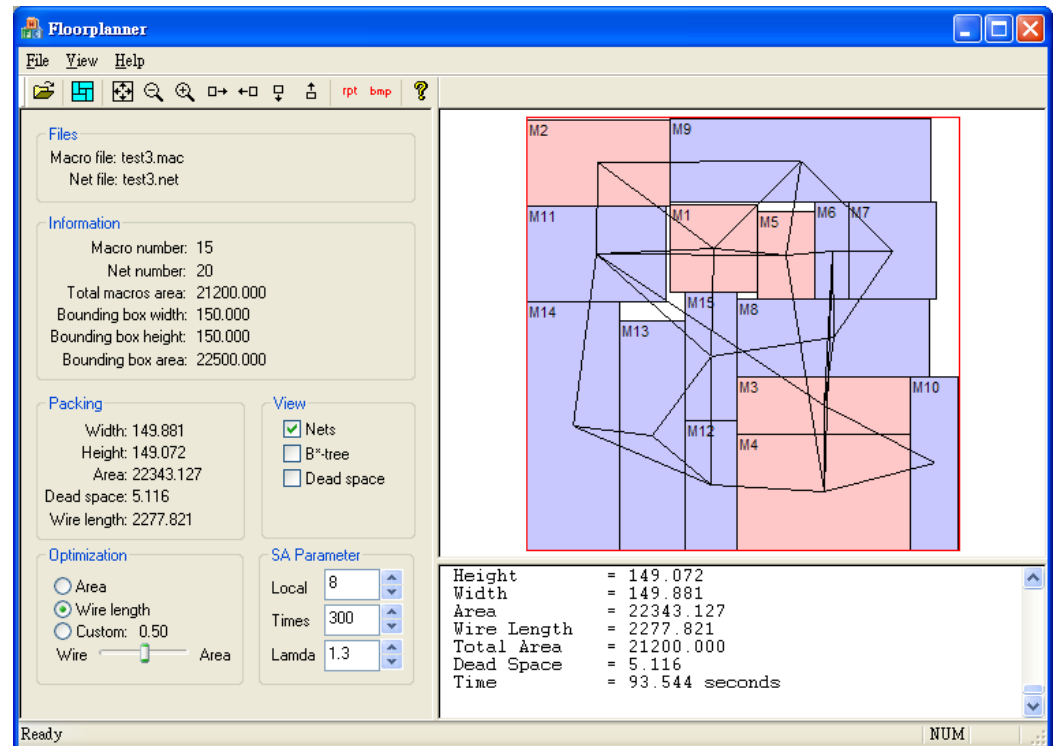


B*-tree Based Floorplanner

- ami49: Area = 36.74 mm^2 ; dead space = 3.53%; CPU time = 0.25 min on SUN Ultra 60 (optimum = 35.445 mm^2).



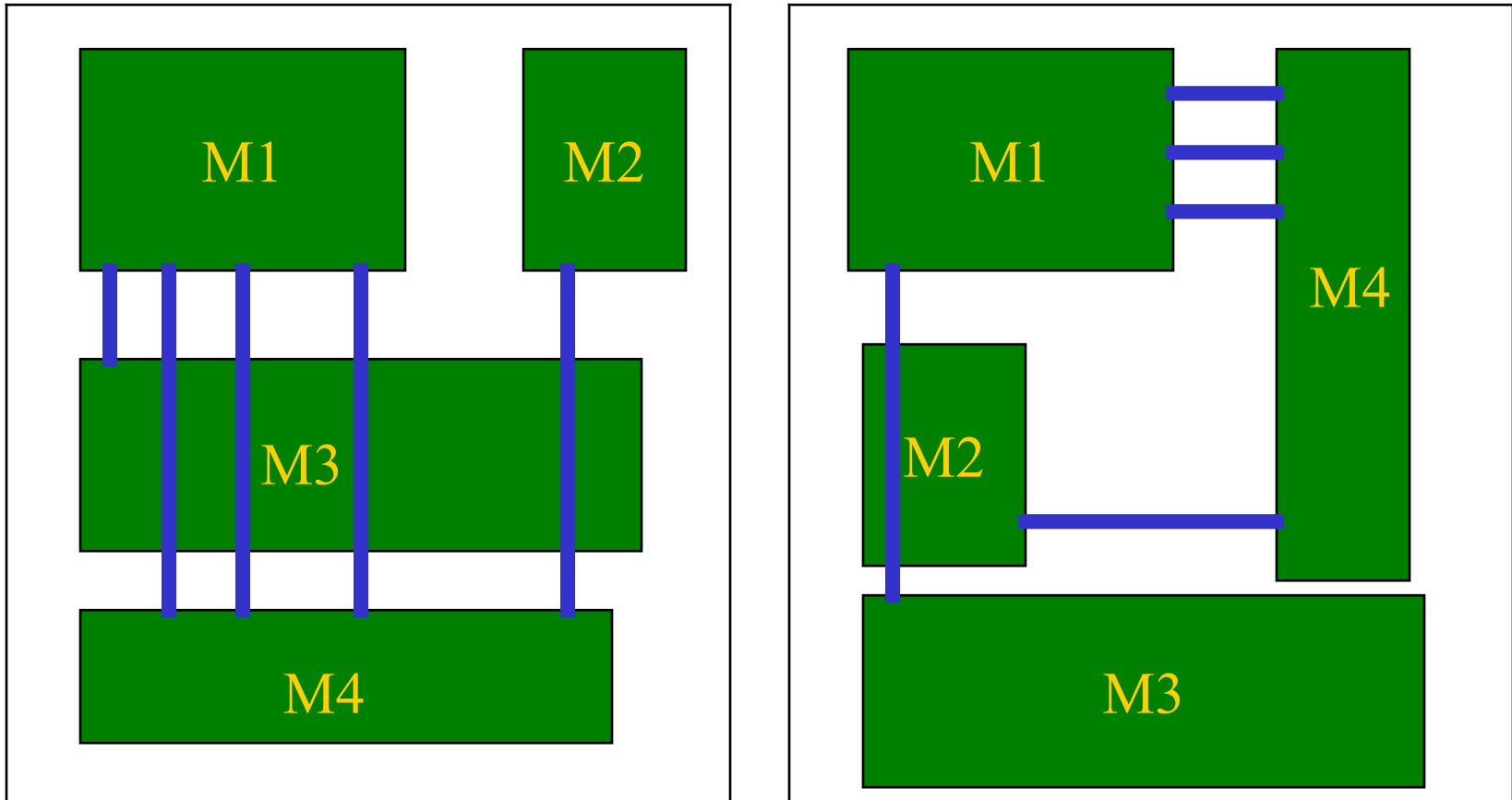
ami49



Courtesy T.-C. Chen

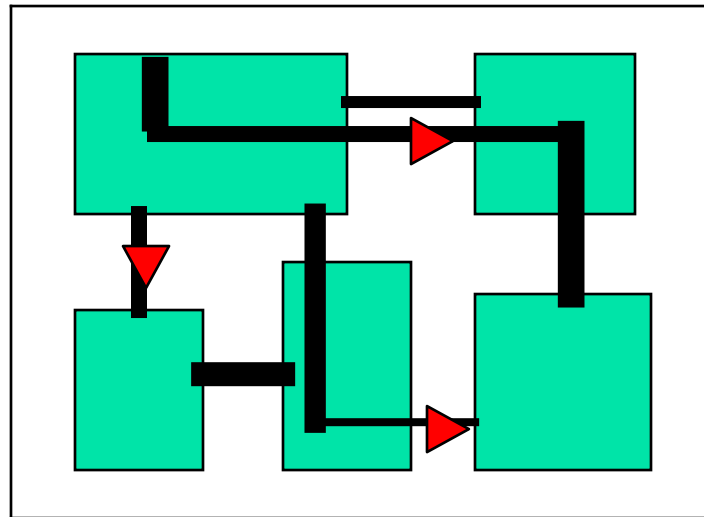
Interconnect-Centric Floorplanning

- Floorplanning greatly influences interconnect structure

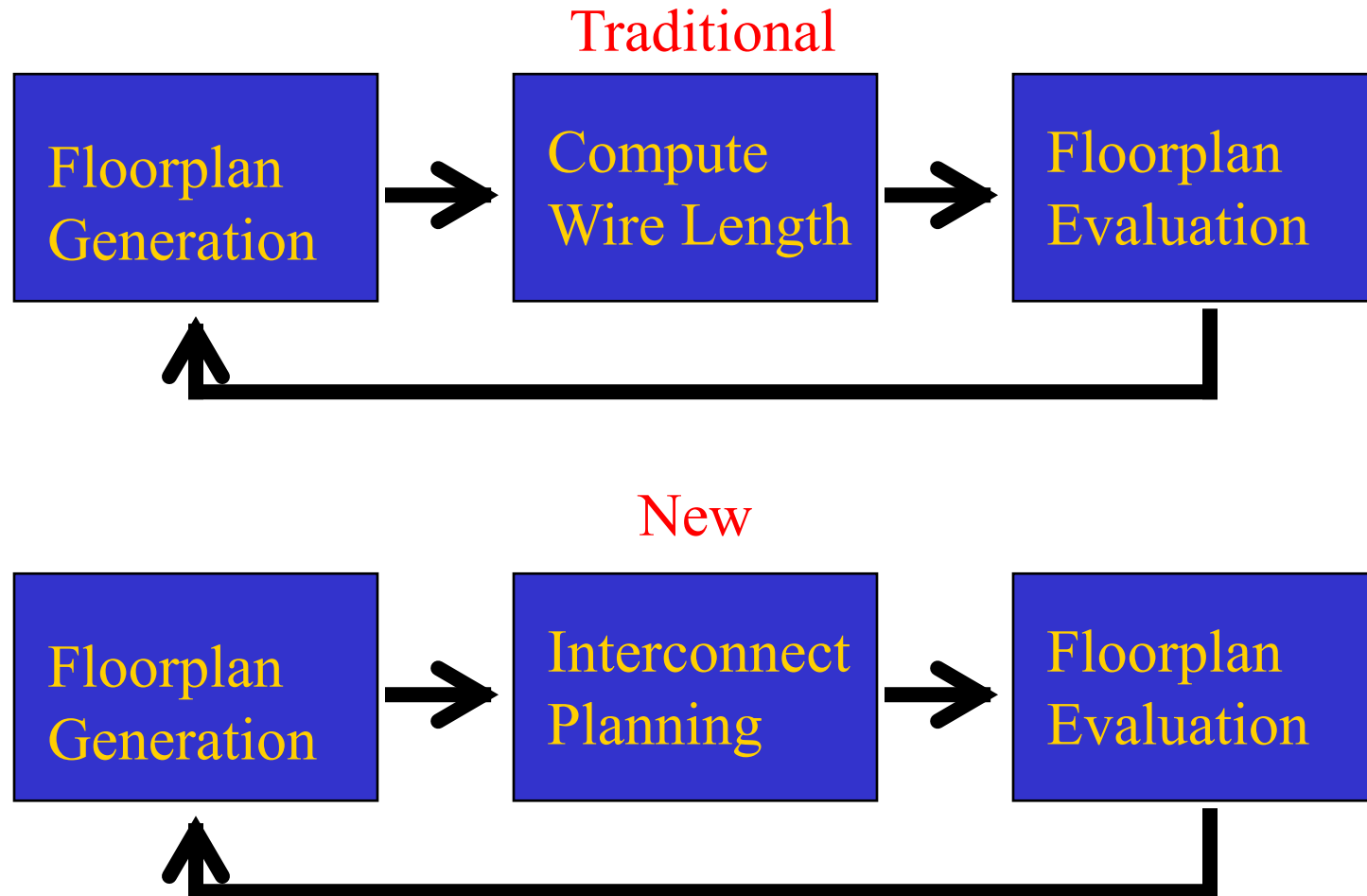


Interconnect Planning

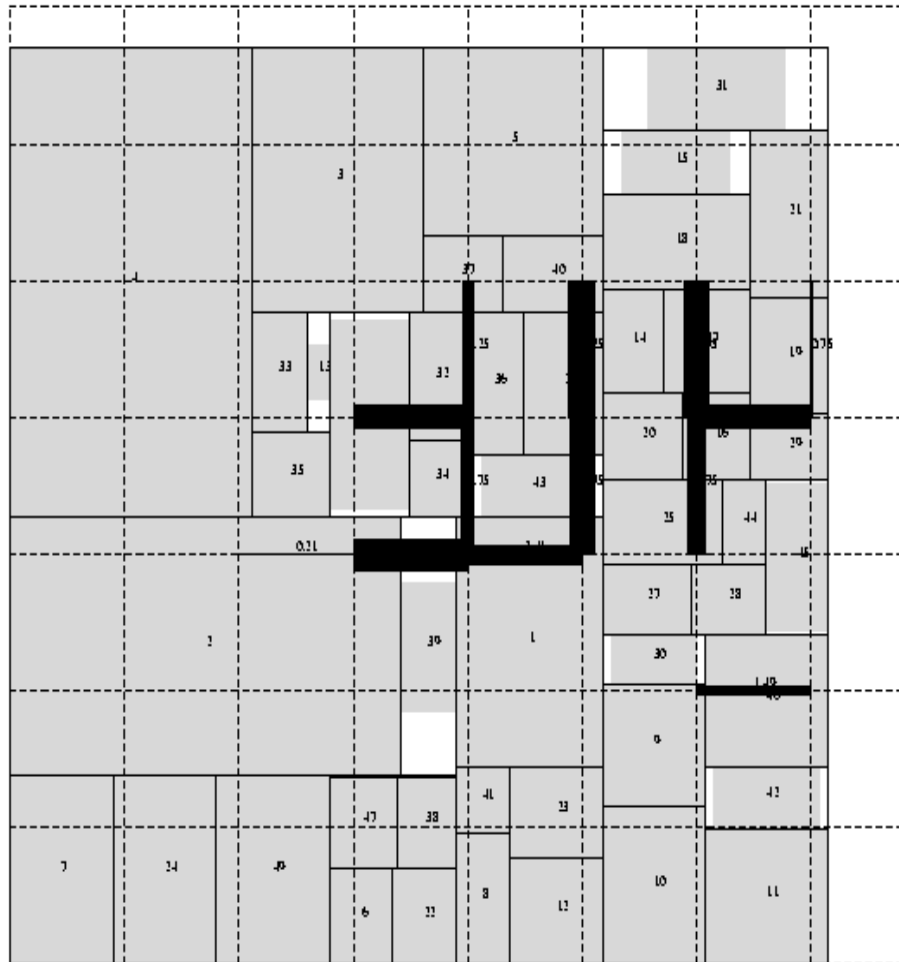
- Pin assignment and routing of global interconnects
- Buffer insertion and sizing
 - Buffer block planning
- Wire sizing



Floorplanning and Interconnect Planning



Interconnect-Centric Floorplanning: ami49 (1/2)



Interconnect-Centric Floorplanning: ami49 (2/2)

