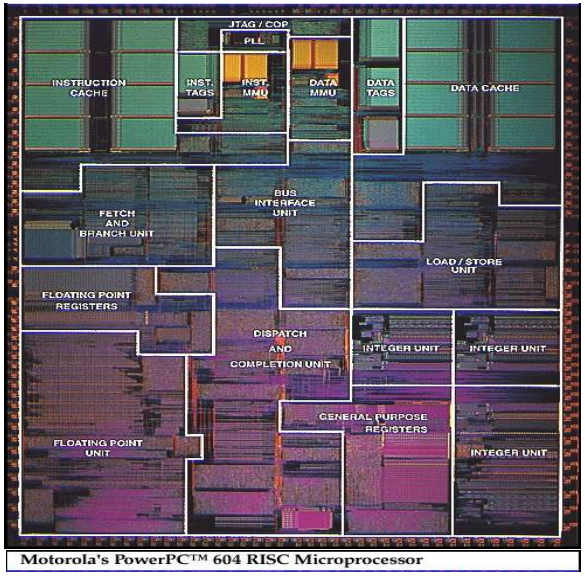
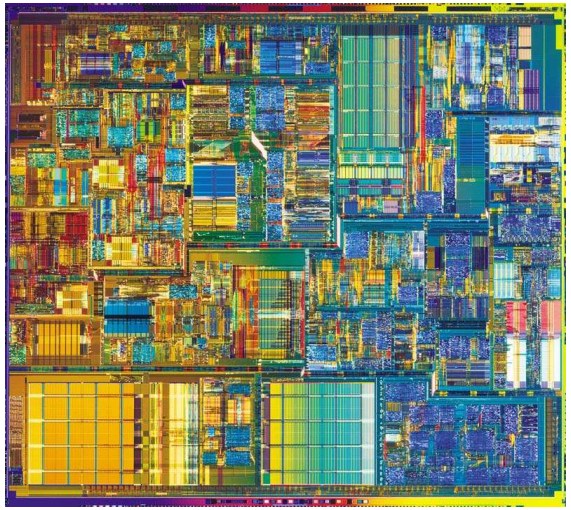
Unit 4: Floorplanning

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



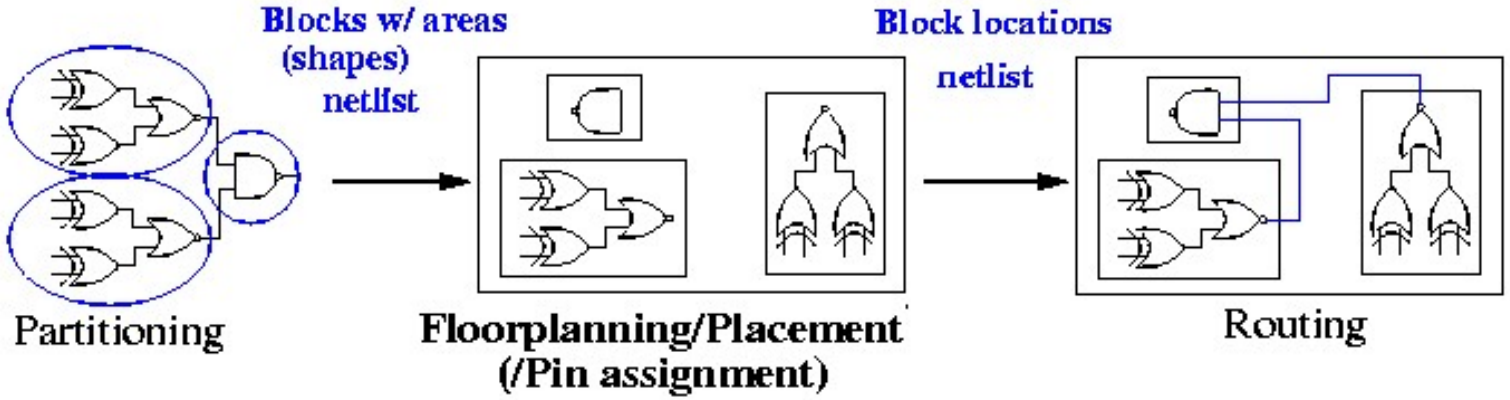
PowerPC 604



Pentium 4

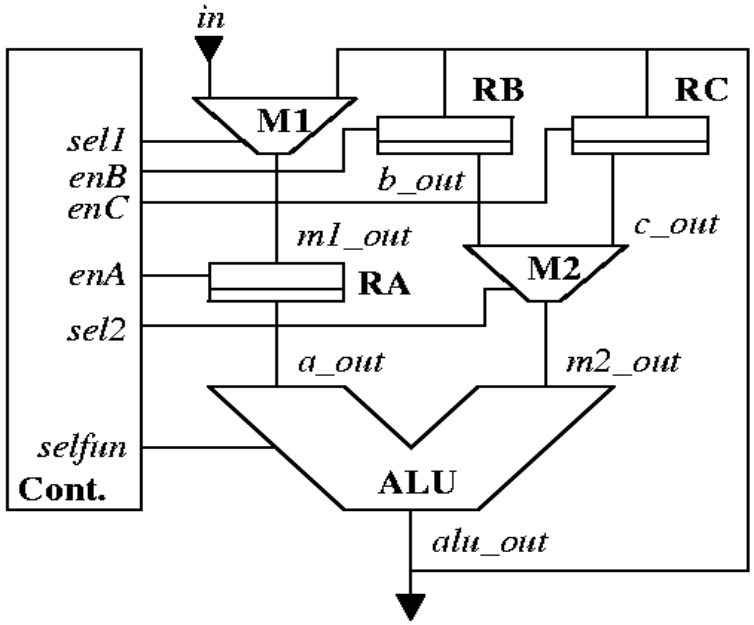
Floorplanning

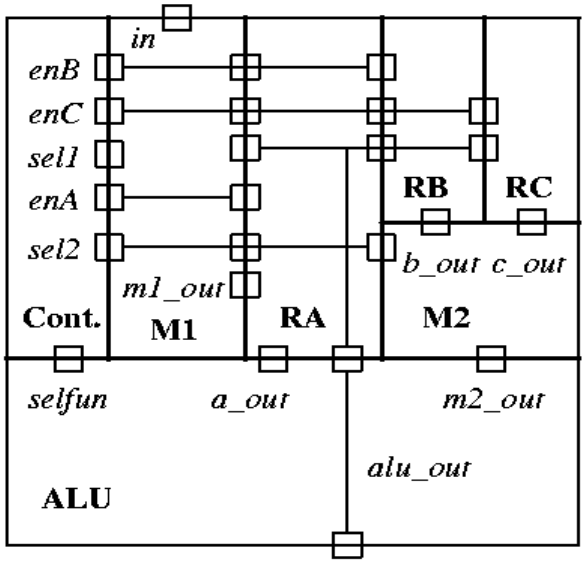
- Partitioning leads to
  - Blocks with well-defined **areas and shapes** (rigid/hard blocks).
  - Blocks with approximate areas and no particular shapes (flexible/soft blocks).
  - A **netlist** specifying connections between the blocks.
- Objectives
  - Find **locations** for all blocks.
  - Consider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

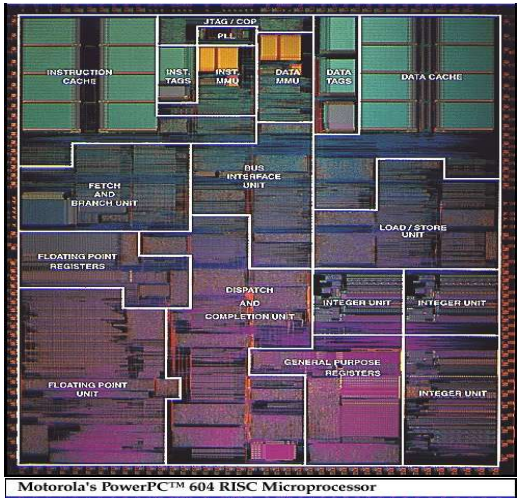


Slides Courtesy of Prof. H.-R. Jiang

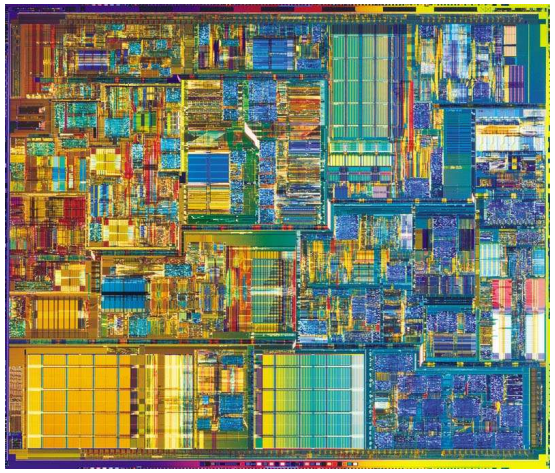
Early Layout Decision Example







PowerPC 604



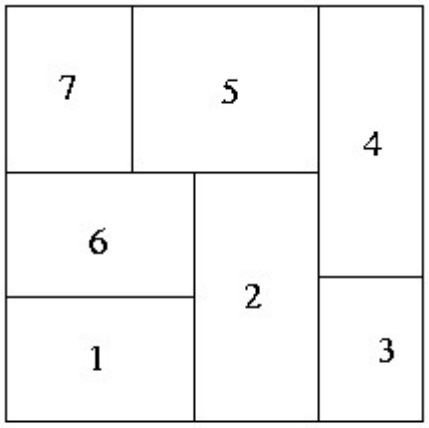
Pentium 4

Early Layout Decision Methodology

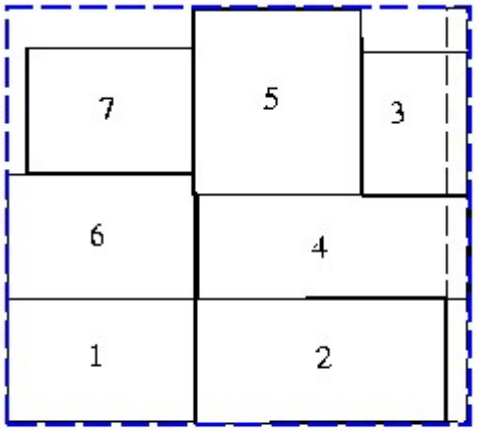
- An IC is a 2-D medium; considering the dimensions of blocks in early stages of the design helps to improve the quality.
- Floorplanning gives early feedback
  - Suggests valuable architectural modifications
  - Estimates the whole chip area
  - Estimates delay and congestion due to wiring
- Floorplanning 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.



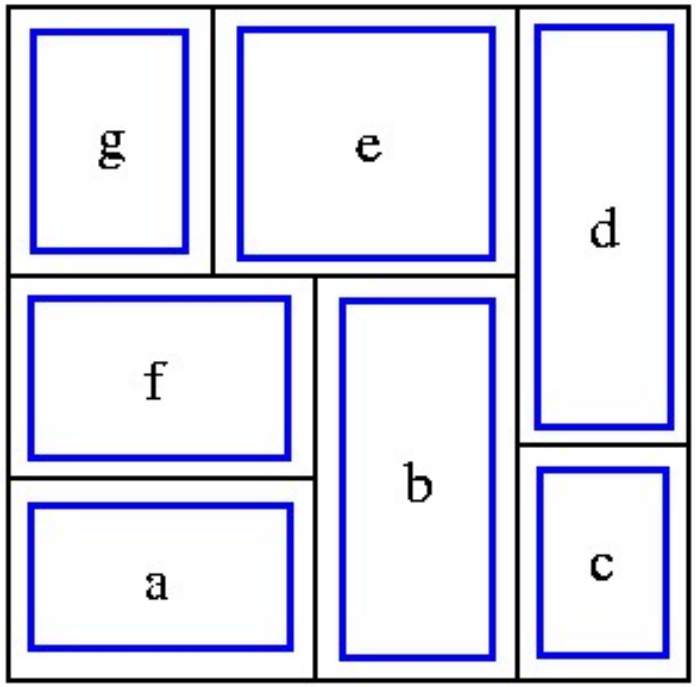
An optimal floorplan, in terms of area

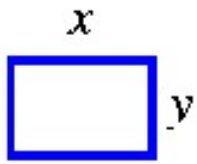


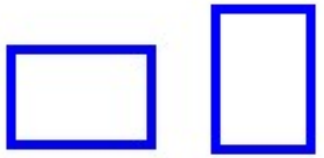
A non–optimal floorplan

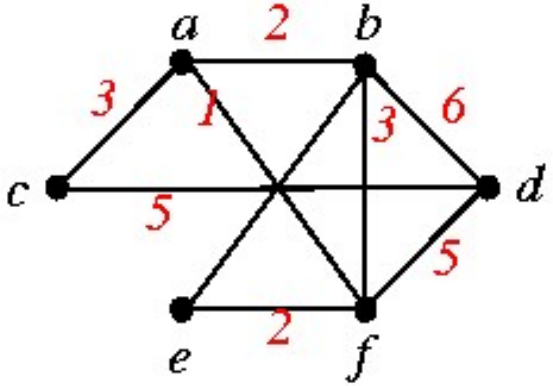
Slides Courtesy of Prof. H.-R. Jiang

Floorplan Design



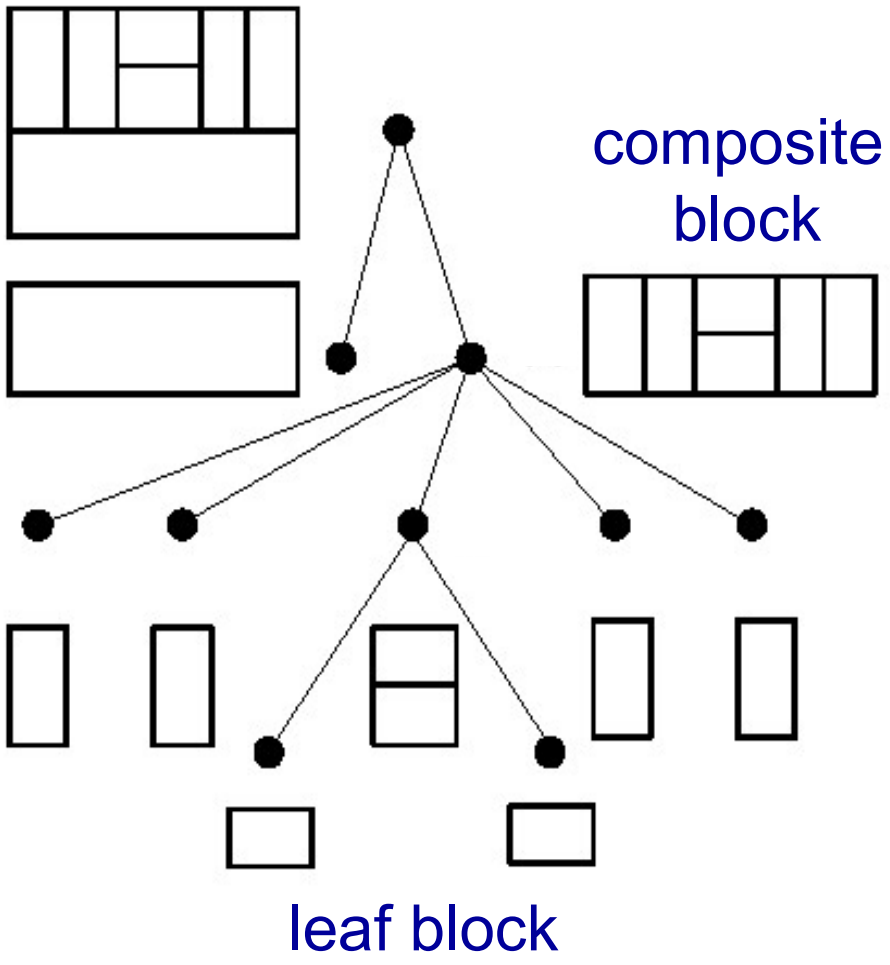
- *Modules:* 

- *Area: A=xy*
- *Aspect ratio: r <= y/x <= s*
- *Rotation:* 
- *Module connectivity*



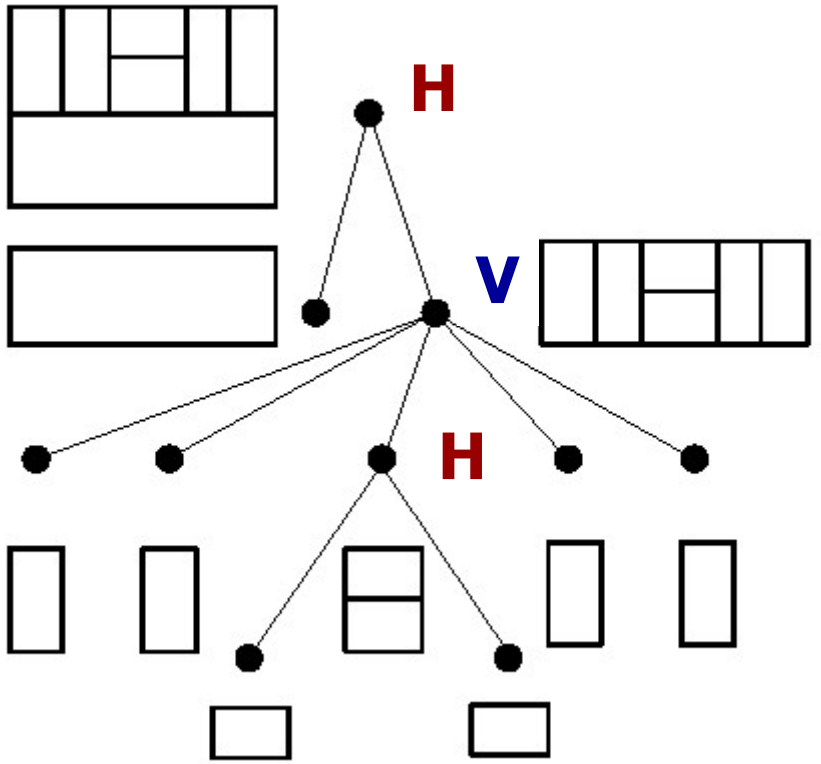
Floorplan Elements

- **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.



Slicing Floorplan + Slicing Tree

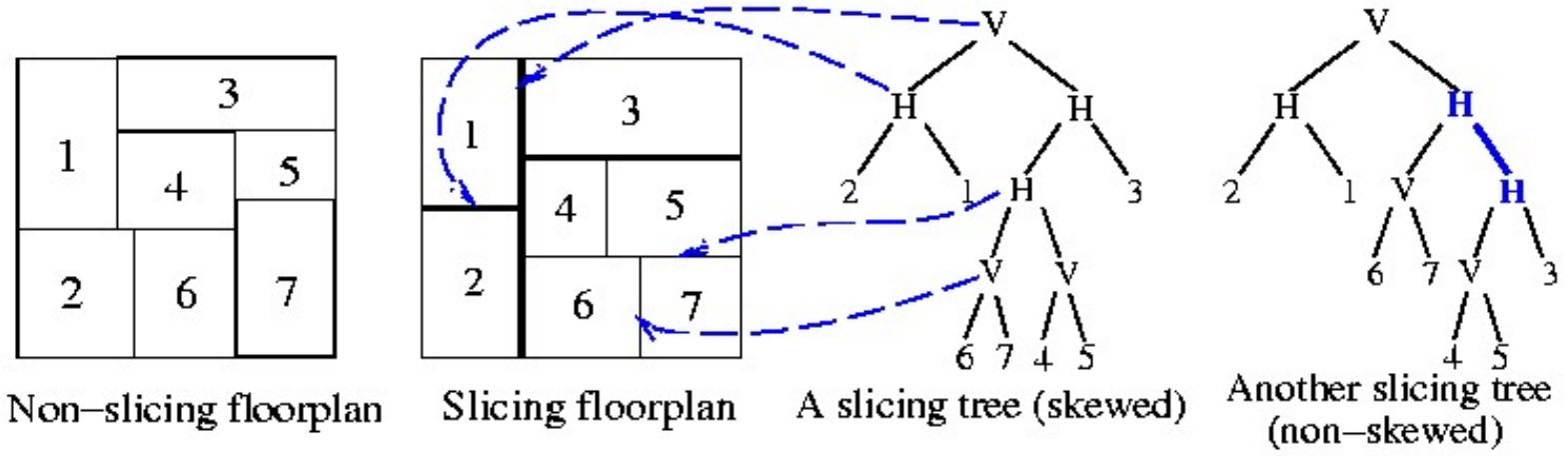
- A composite block's subblocks are obtained by a horizontal or vertical *bisection* of the composite block.
- Slicing floorplans can be represented by a **slicing tree**.
- In a slicing tree, all blocks (except for the top-level block) have a *parent*, and all composite blocks have *children*.
- A slicing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



H: horizontal cut
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.

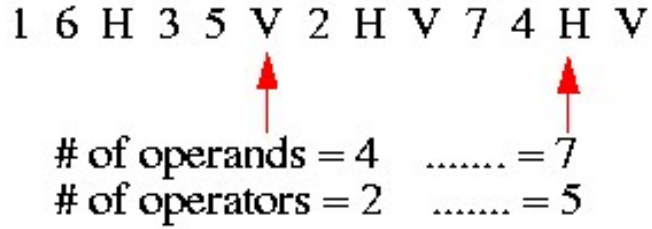


Slicing Floorplan Design by Simulated Annealing

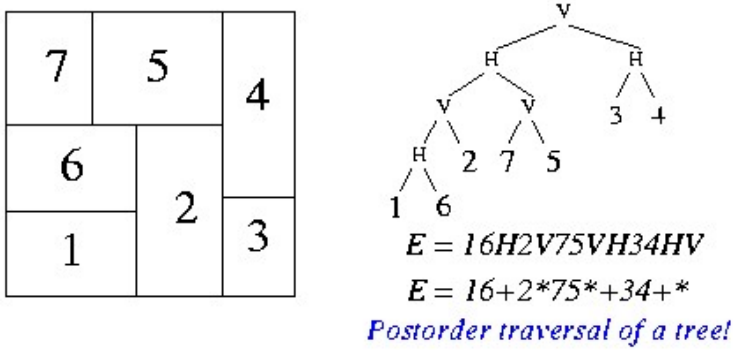
- Related work
  - Wong & Liu, “A new algorithm for floorplan design,” DAC-86.
  - Wong & Liu, “Floorplan design for rectangular and L-shaped modules,” ICCAD'87.
    - Also considers L-shaped modules.
  - Wong, Leong, Liu, *Simulated Annealing for VLSI Design*, pp. 31—71, Kluwer Academic Publishers,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

Solution Representation

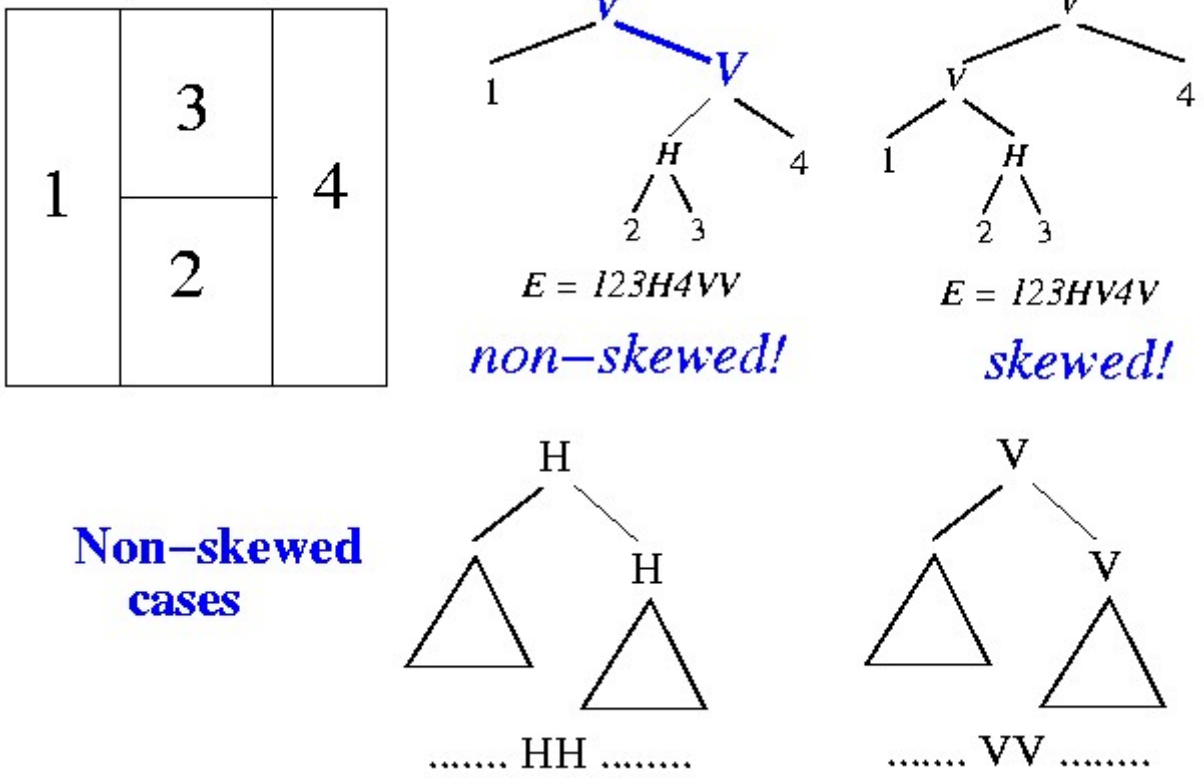
- An expression $E = e_1\ e_2 \ldots\ e_{2n-1}$, where $e_i \in \{1, 2, \ldots, n, H, V\}$, $1 \le i \le 2n\text{-}1$, is a **Polish expression** of length $2n$-1 iff
  1. every operand j, $1 \le j \le n$, appears exactly once in E;
  2. (**the balloting property**) for every subexpression $E_i = e_1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



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



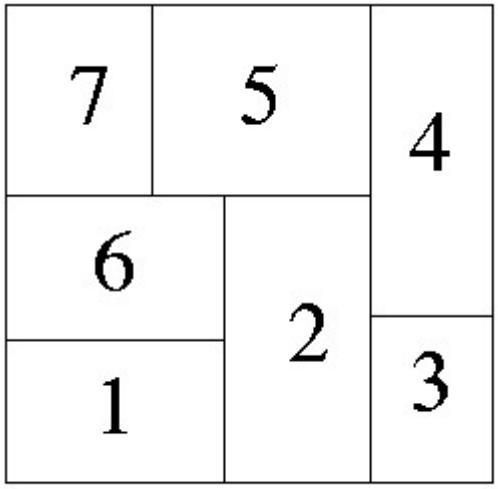
Redundant Representations

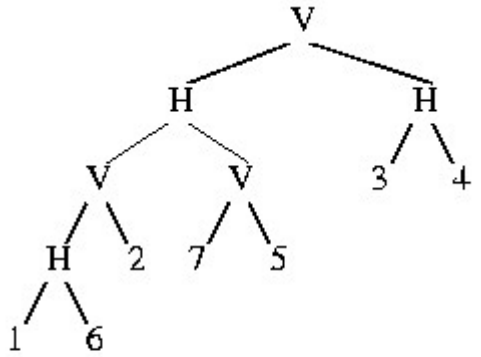


- **Question:** How to eliminate ambiguous representation?

Normalized Polish Expression

- A Polish expression $E = e_1\ e_2\ \ldots\ e_{2n-1}$ 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.



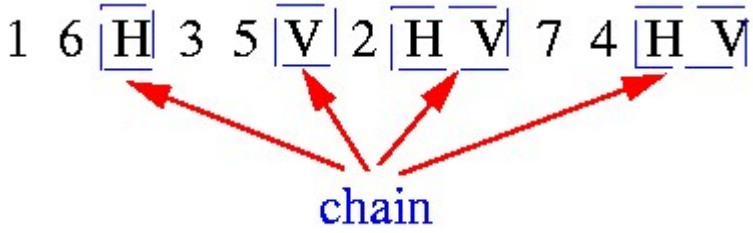


E = 16H2V75VH34HV

A normalized Polish expression

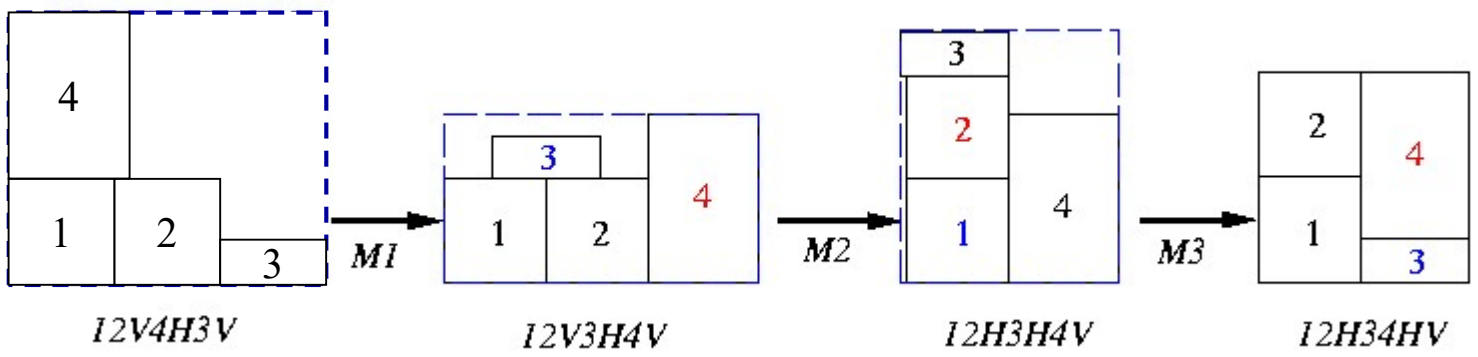
Neighborhood Structure

- **Chain:** *HVHVH* … or *VHVHV* …



- **Adjacent:** 1 and 6 are adjacent operands; 2 and 7 are adjacent operands; 5 and *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}, $1 \le i \le k\text{-}1$. Then, the swap will not violate the balloting property iff $2N_{i+1} < i$.
  - N_k: # of operators in the Polish expression $E = e_1\ e_2\ \ldots\ e_k$, $1 \le k \le 2n\text{-}1$

Illegal M3 Moves

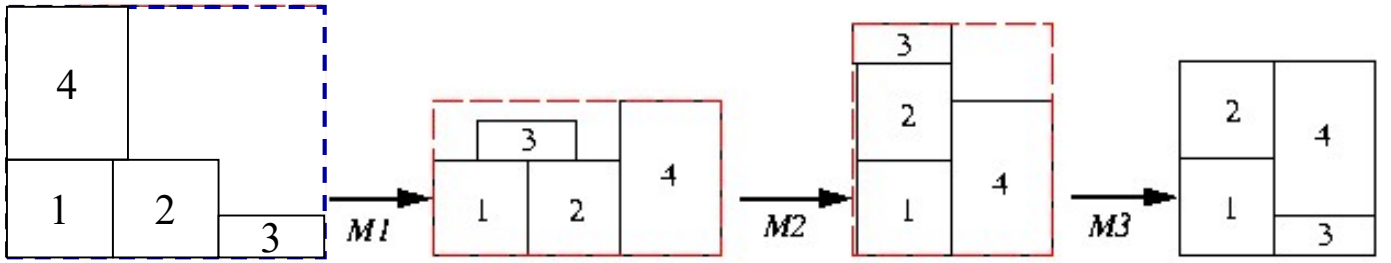
- Assume that *M3* swaps the operand e_i with the operator e_{i+1}, $1 \le i \le k\text{-}1$.

		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:
$i=2n=2N_{i+1}$

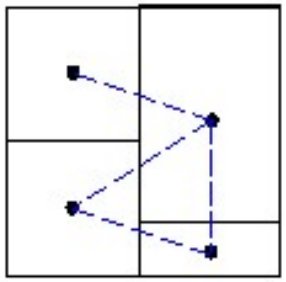
Cost Function

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



A: 12H34HV

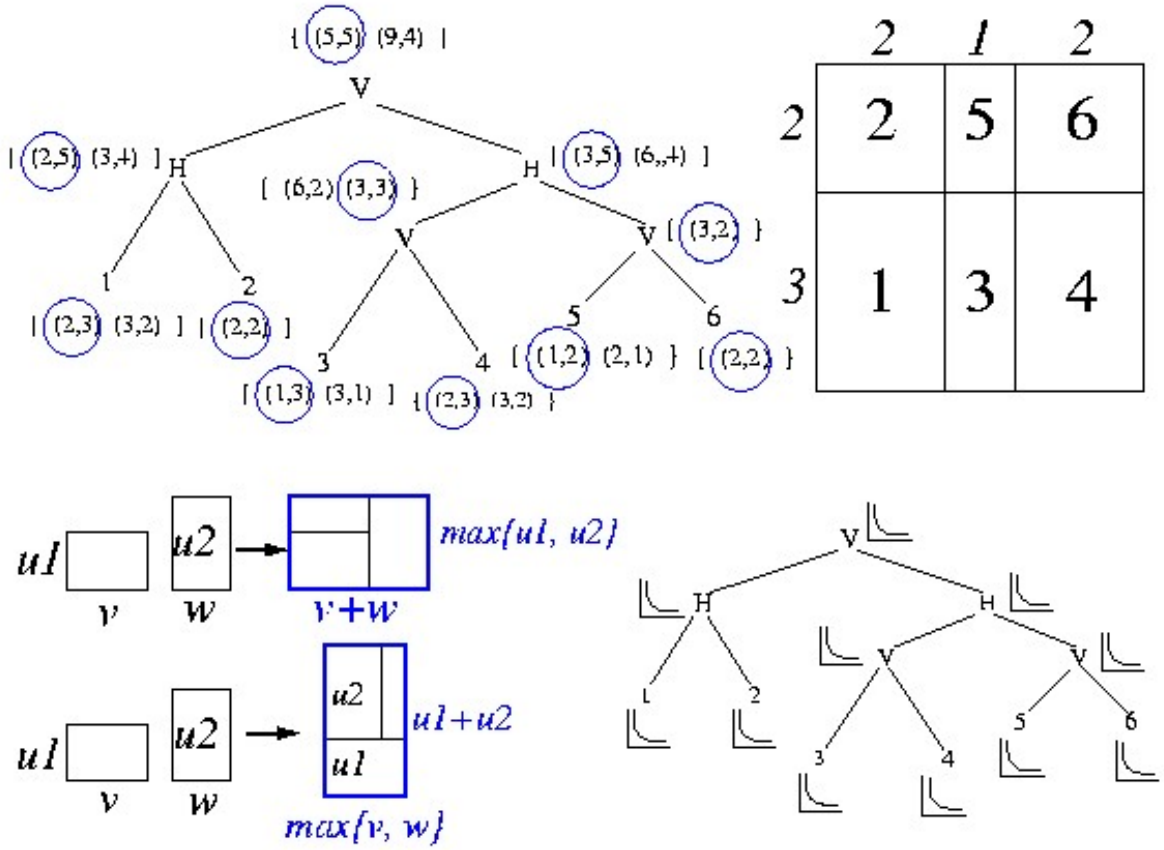
- $W = \sum_{ij} c_{ij}\, d_{ij}$.
  - c_{ij}: # of connections between blocks i and j.
  - d_{ij}: center-to-center distance between basic rectangles i and j.



Slides Courtesy of Prof. H.-R. Jiang

Area Computation for Hard Blocks

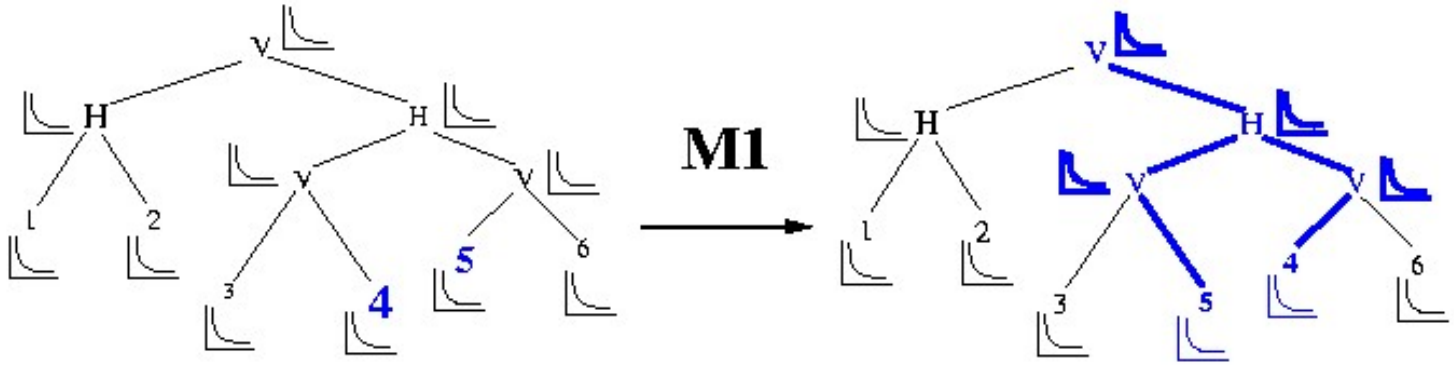
- Allow rotation



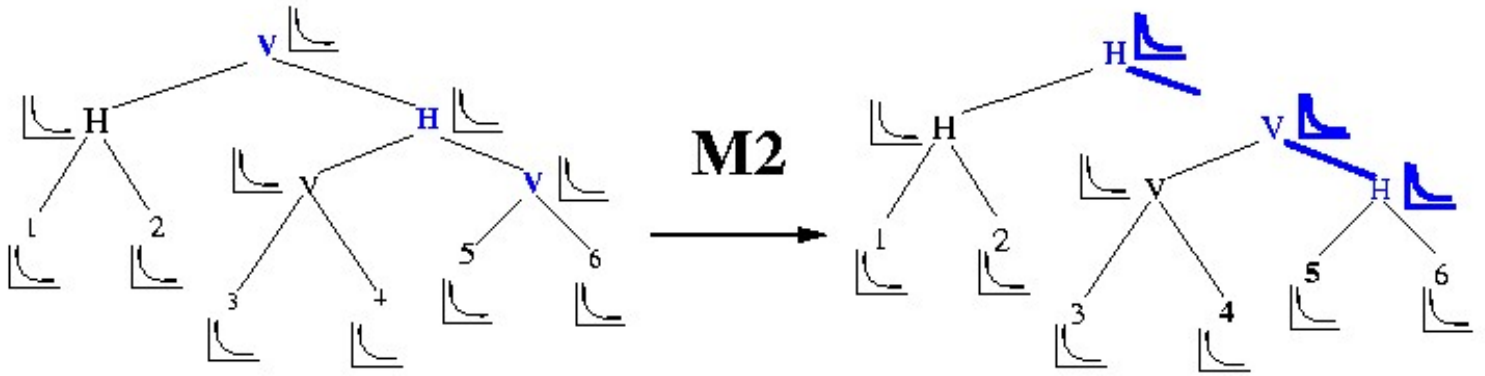
- Wiring cost?
  - Center-to-center interconnection length

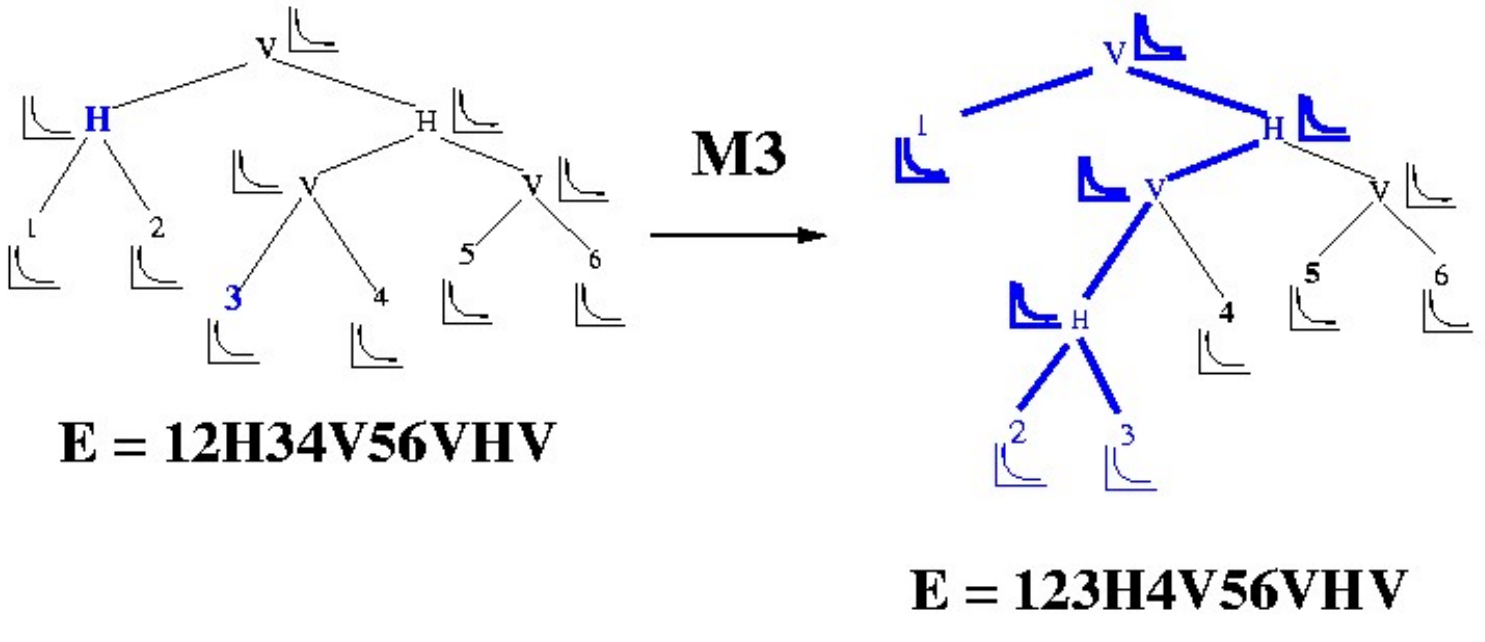
Incremental Computation of Cost Function

- 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.



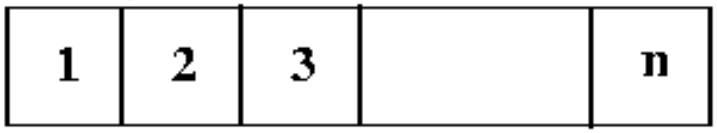
Incremental Computation of Cost Function (cont'd)





Annealing Schedule

- Initial solution: 12*V*3*V* … *nV*.



- $T_i = r^i T_0$, i = 1, 2, 3, …; 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.

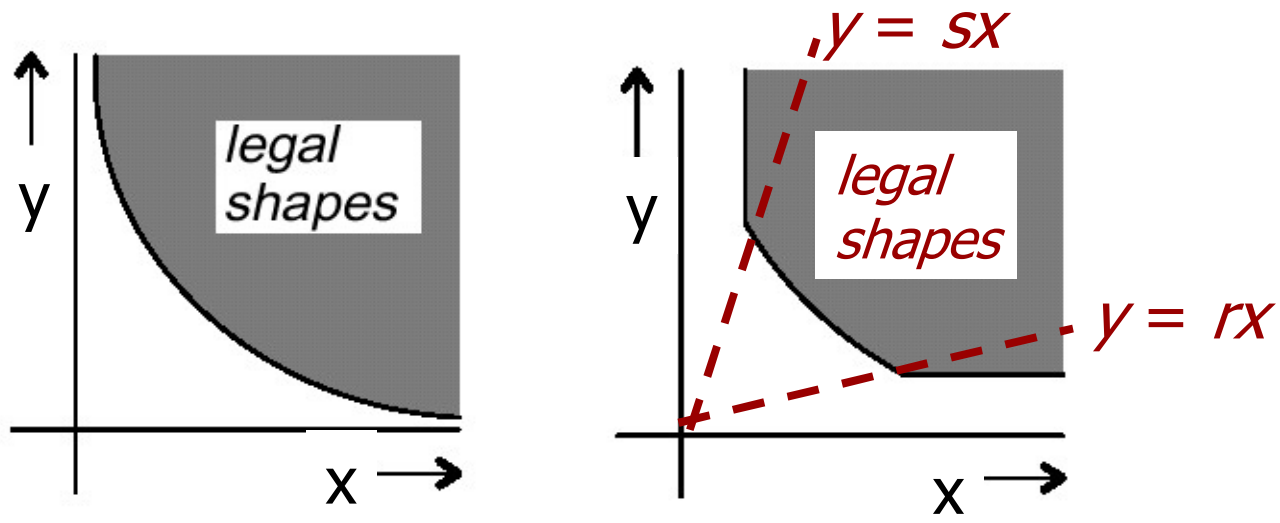
Algorithm: Wong-Liu (P, ε, r, k)

```
1 begin
2 E ← 12V3V4V … nV; /* initial solution */
3 Best ← E; T0 ← Δavg / ln(P) ; M ← MT ← uphill ← 0; N = kn;
4 repeat
5   MT ← uphill ← reject ← 0;
6   repeat
7     SelectMove(M);
8     Case M of
9     M1: Select two adjacent operands ei and ej; NE ← Swap(E, ei, ej);
10    M2: Select a nonzero length chain C; NE ← Complement(E, C);
11    M3: done ← FALSE;
12       while not (done) do
13           Select two adjacent operand ei and operator ei+1;
14           if (ei-1 ≠ ei+1)  and (2 Ni+1 < i) then done ← TRUE;
13'          Select two adjacent operator ei and operand ei+1;
14'          if (ei≠ ei+2) then done ← TRUE;
15       NE ← Swap(E, ei, ei+1);
16    MT ← MT+1; Δcost ← cost(NE) - cost(E);
17    if (Δcost ≤ 0) or (Random < e^(-Δcost/T))
18     then
19        if (Δcost > 0) then uphill ← uphill + 1;
20        E ← NE;
21        if  cost(E) < cost(best) then best ← E;
22     else reject ← reject + 1;
23   until (uphill > N) or (MT > 2N);
24   T ← rT; /* reduce temperature */
25 until (reject/MT > 0.95) or (T < ε) or OutOfTime;
26 end
```


Slides Courtesy of Prof. H.-R. Jiang

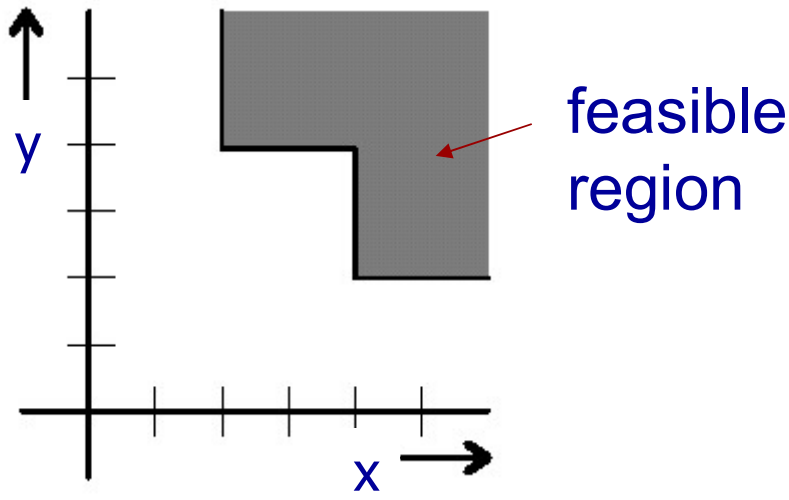
Shape Curve for Floorplan Sizing

- 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: $r \le y/x \le s$.



Shape Curve

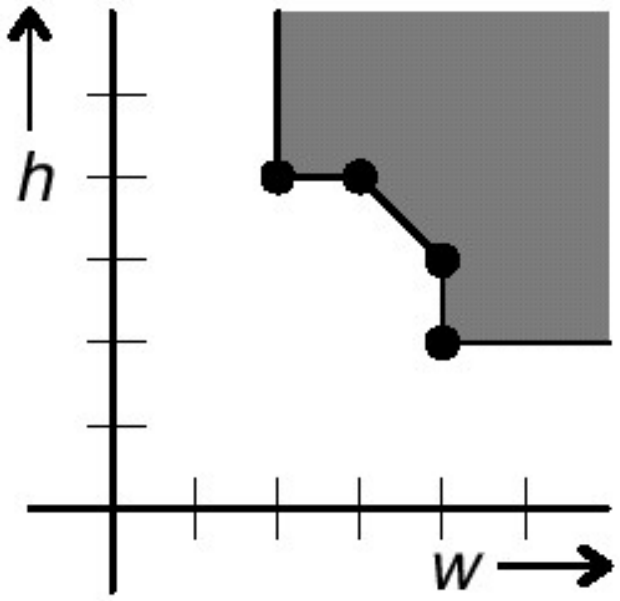
- Since a basic block is built from discrete transistors, it is not realistic to assume that the shape function follows the hyperbola continuously.
- In an extreme case, a block is rigid/hard: it can only be rotated and mirrored during floorplanning or placement.



The shape curve of a 2×4 hard block.

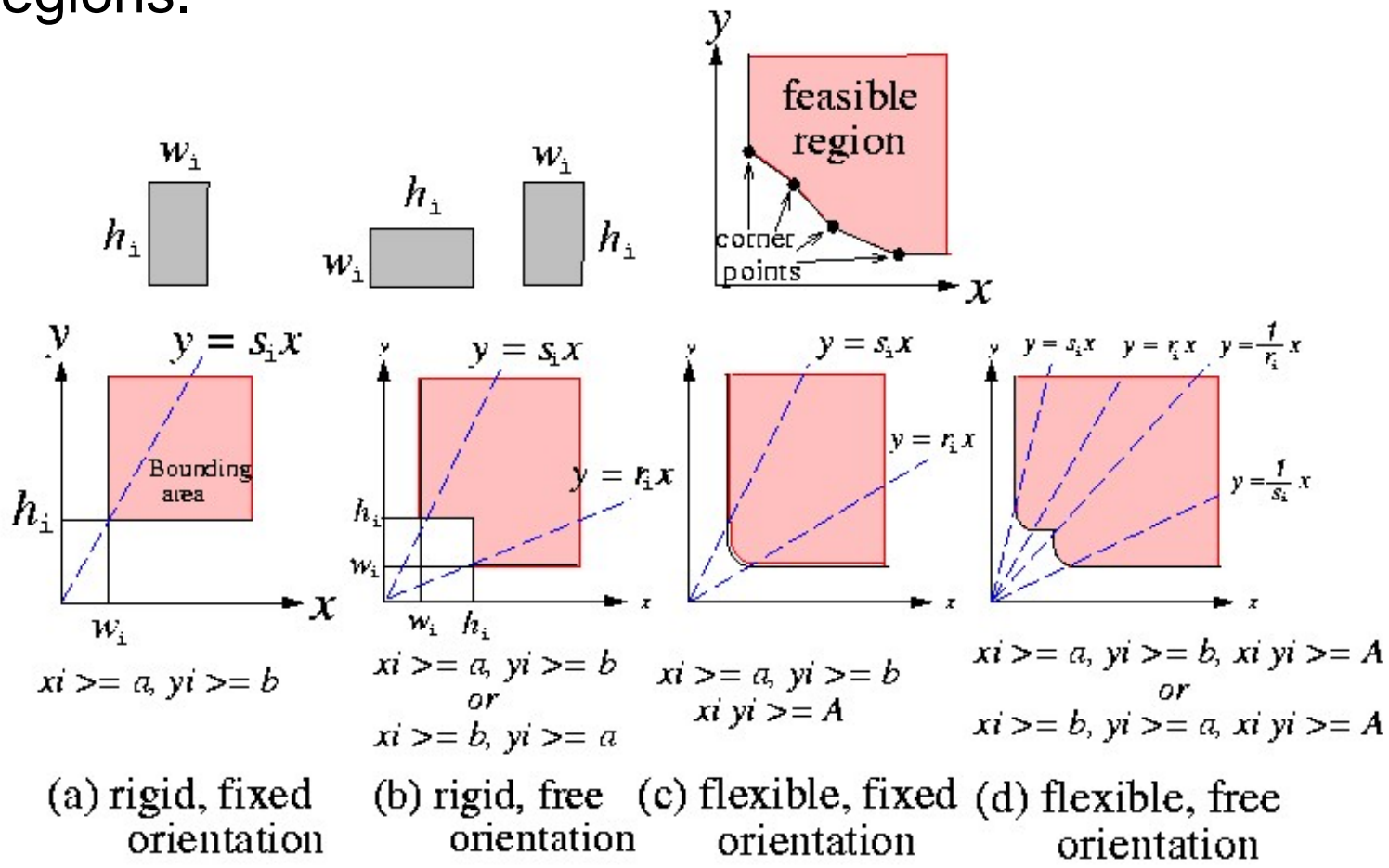
Shape Curve (cont'd)

- 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

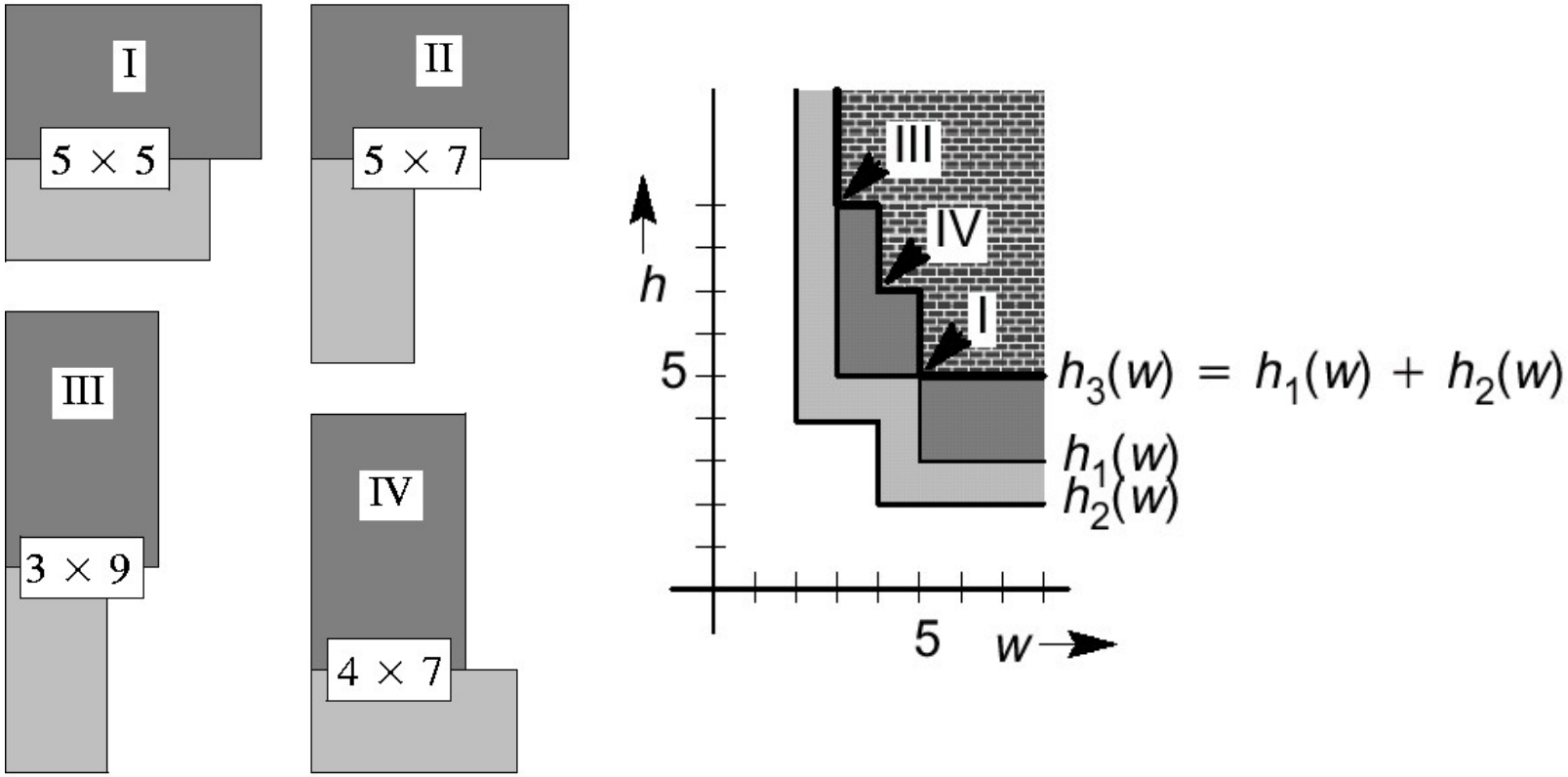
- Shape curves correspond to different kinds of constraints where the shaded areas are feasible regions.



Slides Courtesy of Prof. H.-R. Jiang

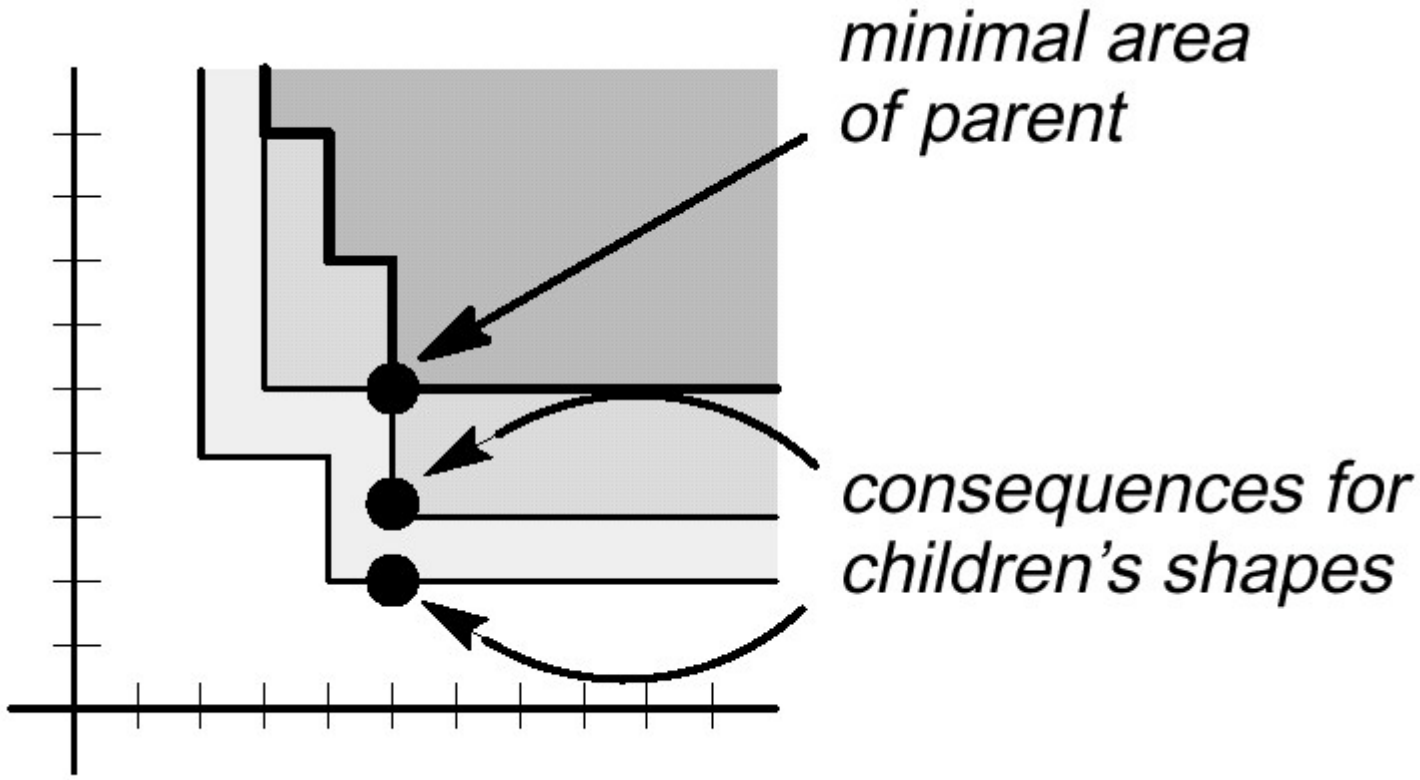
Vertical Abutment

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



Deriving Shapes of Children

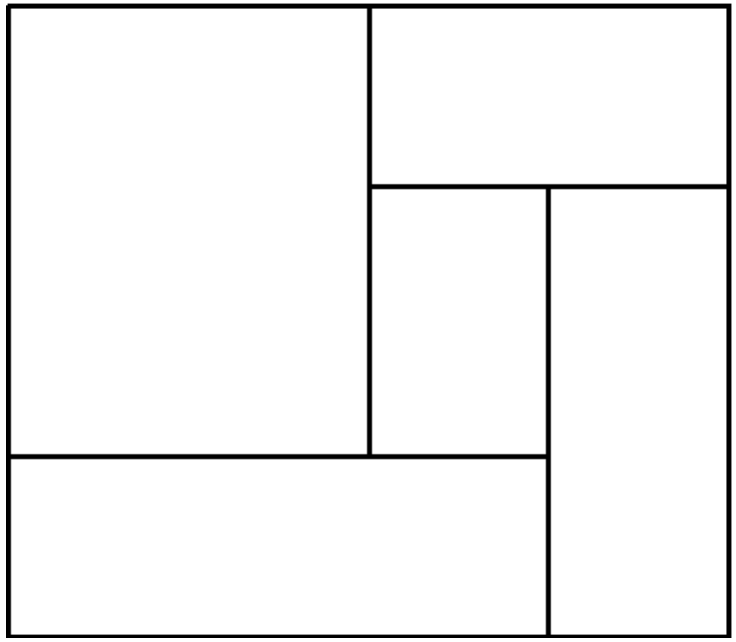
- A choice for the minimal shape of a composite block fixes the shapes of the shapes of its children blocks.



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

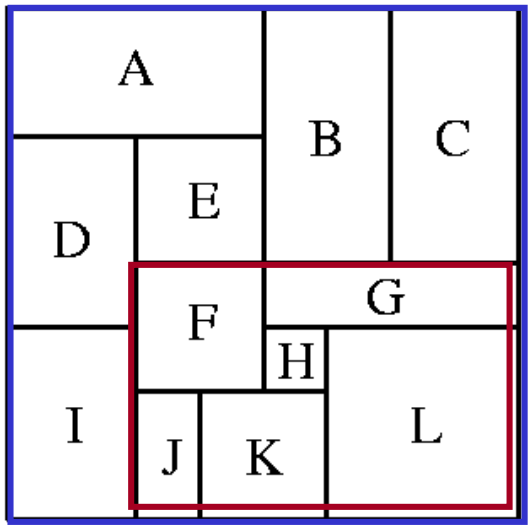
Wheel or Spiral Floorplan

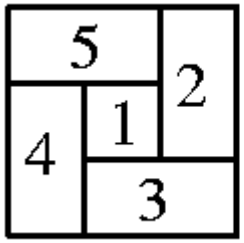


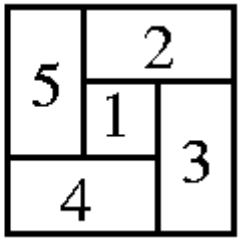
- This floorplan is not slicing!
- **Wheel** is the smallest non-slicing floorplans.

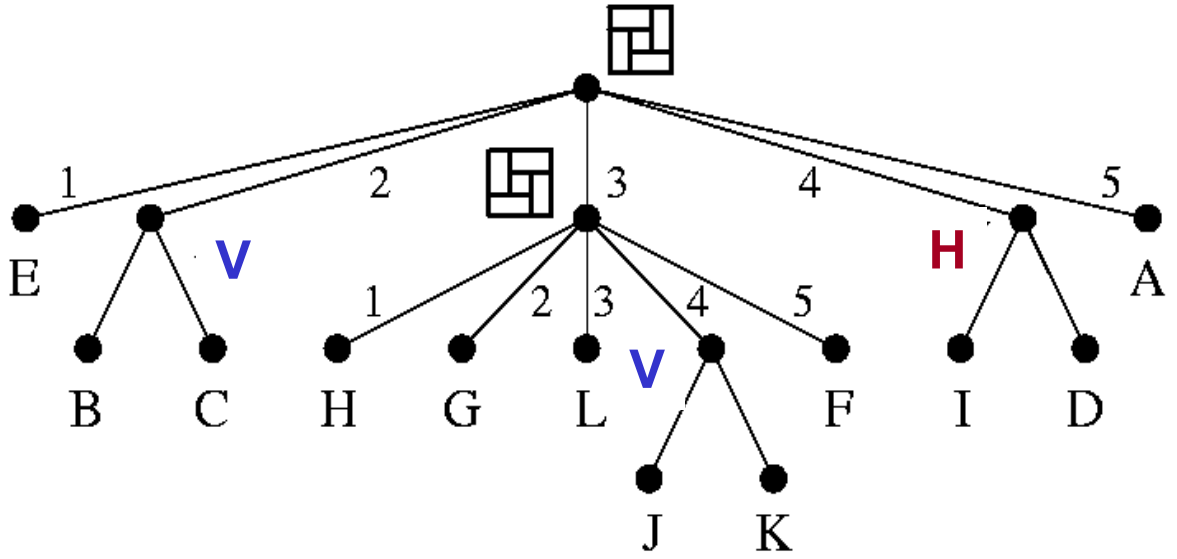
- Limiting floorplans to those that have the slicing property can facilitate floorplanning algorithms.
- Taking the shape of a wheel floorplan and its mirror image as the basis of operators leads to hierarchical descriptions of *order 5*.

Order-5 Floorplan Examples





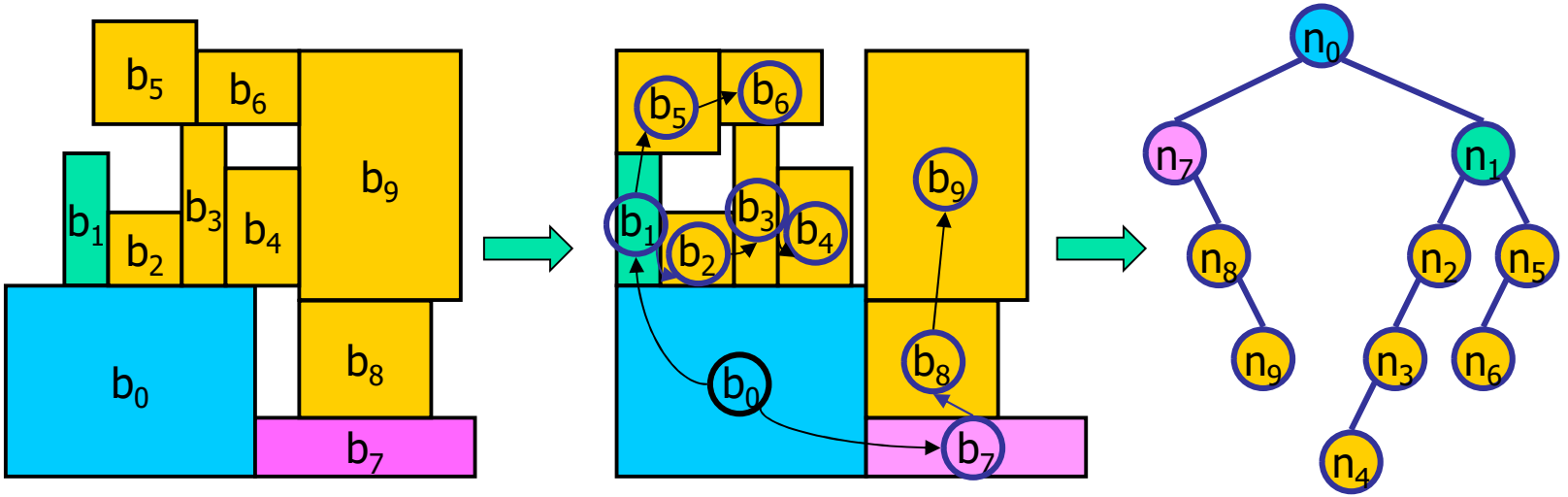




Slides Courtesy of Prof. H.-R. Jiang

B*-Tree: Compacted Floorplan Representation

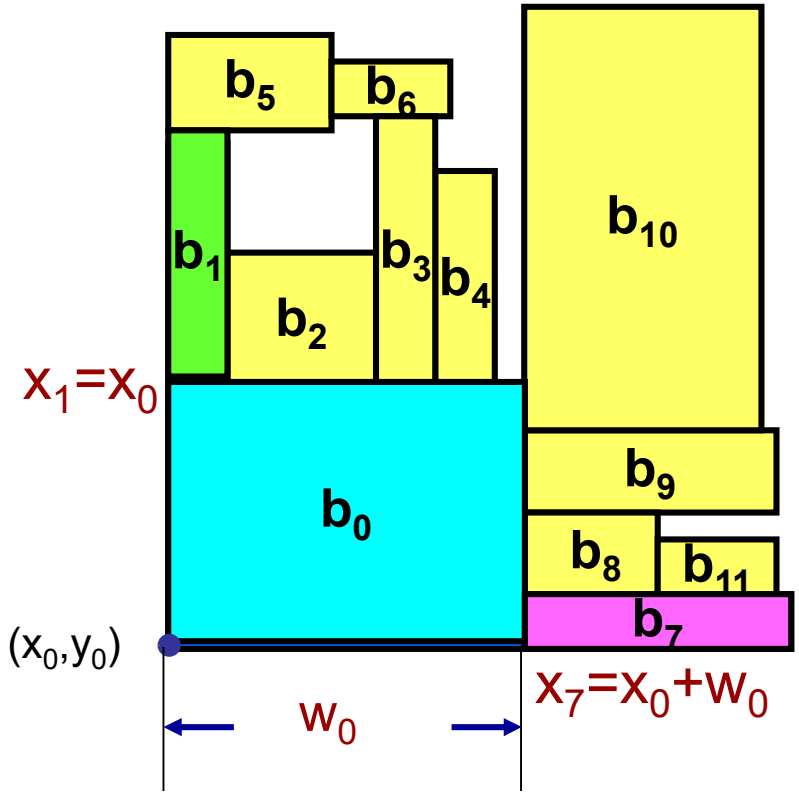
- Chang et. al., "B-tree: A new representation for non-slicing floorplans," DAC-2k.
  - Compact modules to left and bottom.
  - Construct an ordered binary tree (B*-tree).
    - Left child: the lowest, adjacent block on the right ($x_j = x_i + w_i$).
    - Right child: the first block above, with the same x-coordinate ($x_j = x_i$).

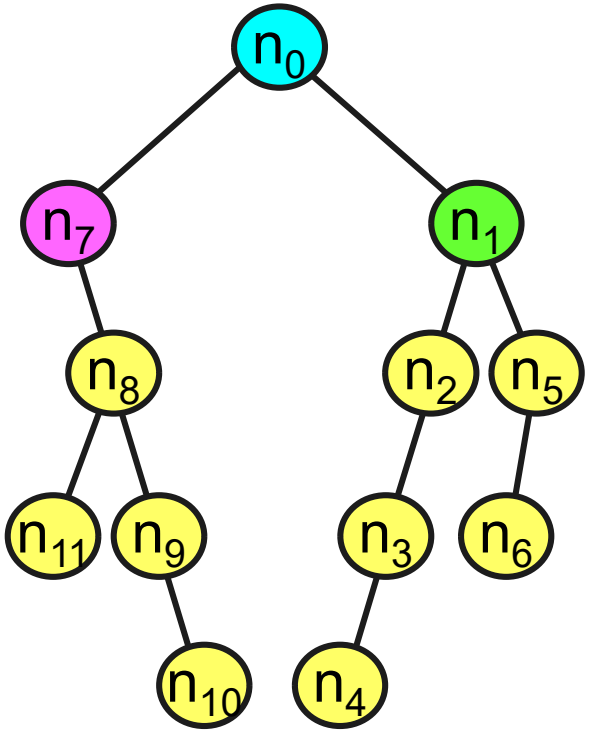


A non-slicing floorplan Compact to left and down B*-tree

B*-tree Packing

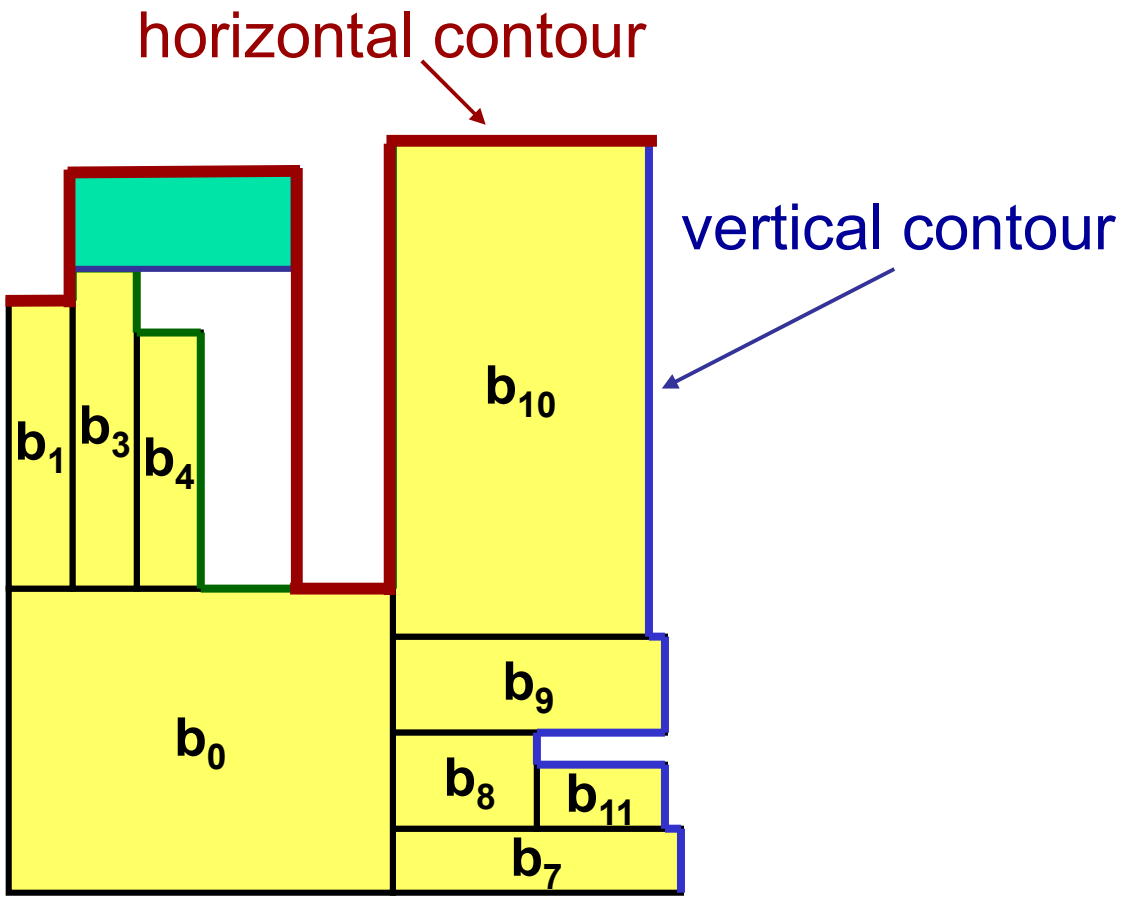
- x-coordinates can be determined by the tree structure.
  - Left child: the lowest, adjacent block on the right ($x_j = x_i + w_i$).
  - Right child: the first block above, with the same *x*-coordinate ($x_j = x_i$).
- y-coordinates?





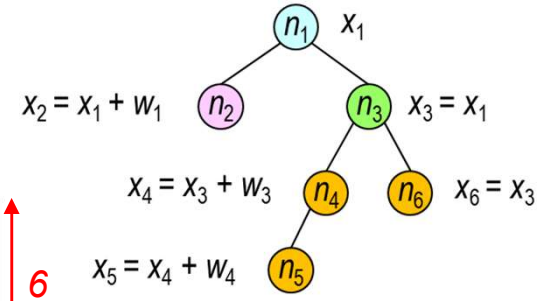
Computing y-coordinates

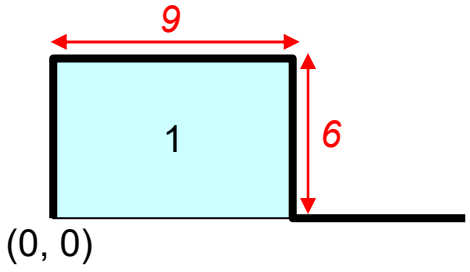
- Reduce the complexity of computing a y-coordinate to amortized O(1) time.



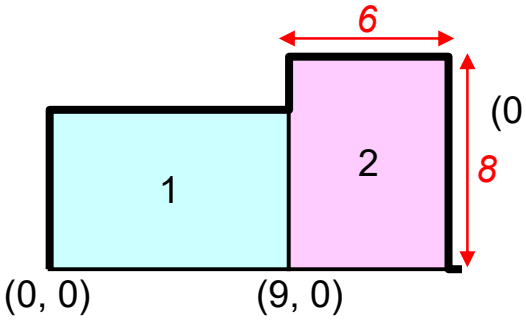
Slides Courtesy of Prof. H.-R. Jiang

Contour Data Structure

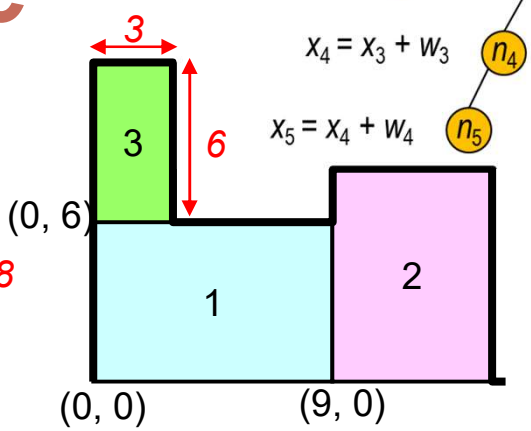




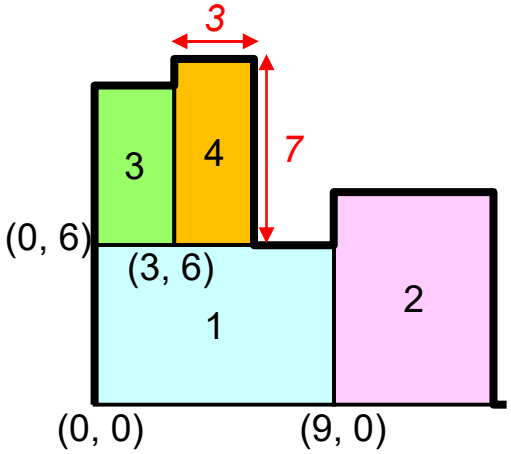
C = <(0,0), **(0,6), (9,6), (9,0),** (∞,0)>



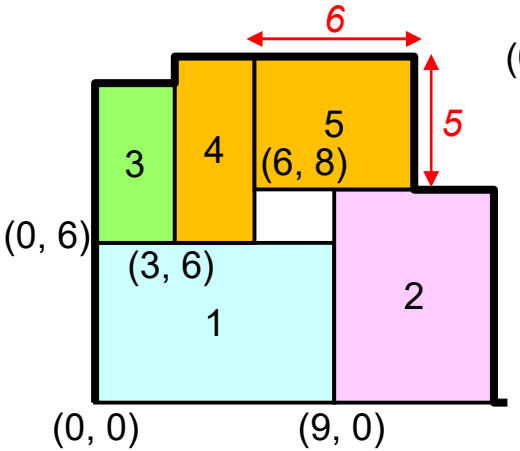
C = <(0,0), (0,6), (9,6), **(9,8), (15,8), (15,0),** (∞,0)>



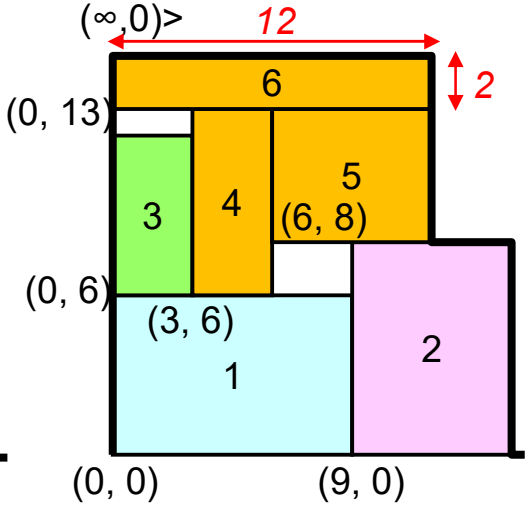
C = <(0,0), **(0,12), (3,12), (3,6),** (9,6), (9,8), (15,8), (15,0), (∞,0)>



C = <(0,0), (0,12), (3,12), **(3,13), (6,13), (6,6),** (9,6), (9,8), (15,8), (15,0), (∞,0)>



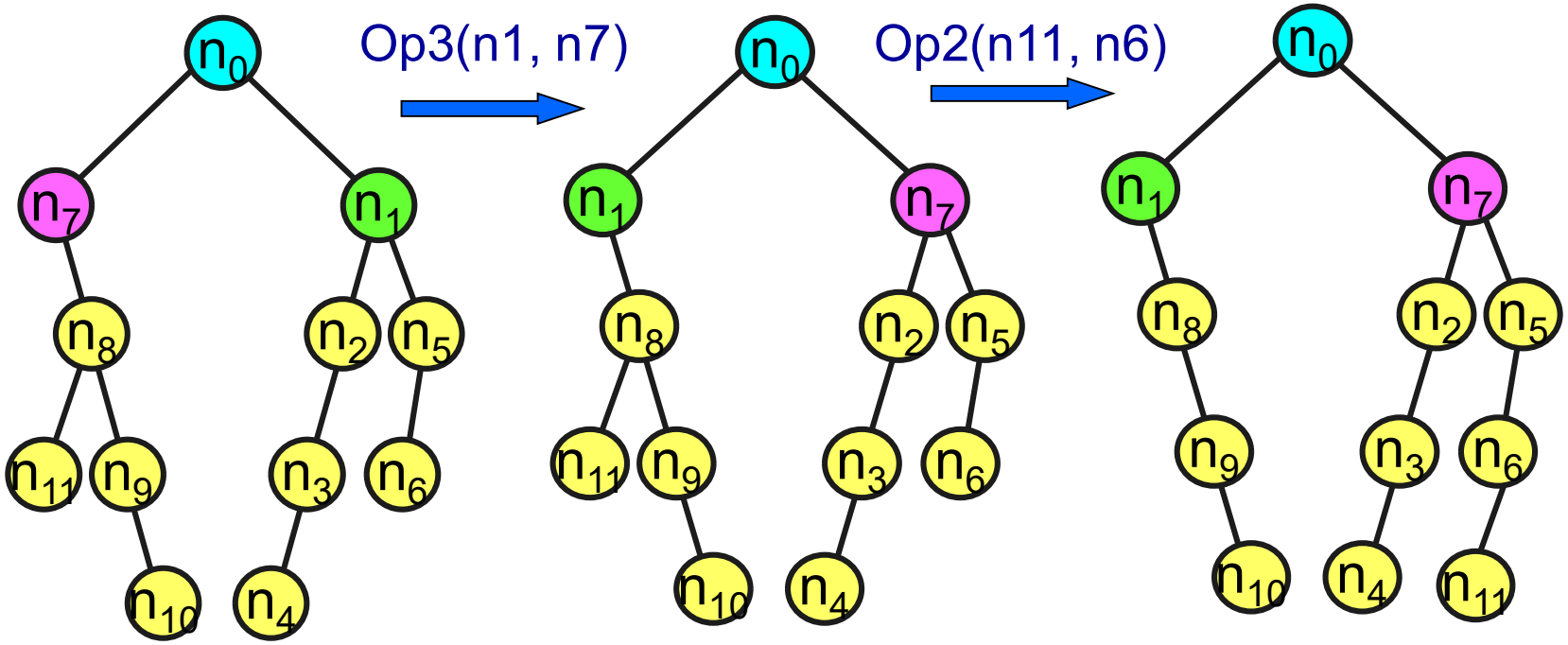
C = <(0,0), (0,12), (3,12), (3,13), **(6,13), (12,13), (12,8),** (15,8), (15,0), (∞,0)>



C = <(0,0), **(0,15), (12,15), (12,13),** (12,8), (15,8), (15,0), (∞,0)>

B*-Tree Perturbation

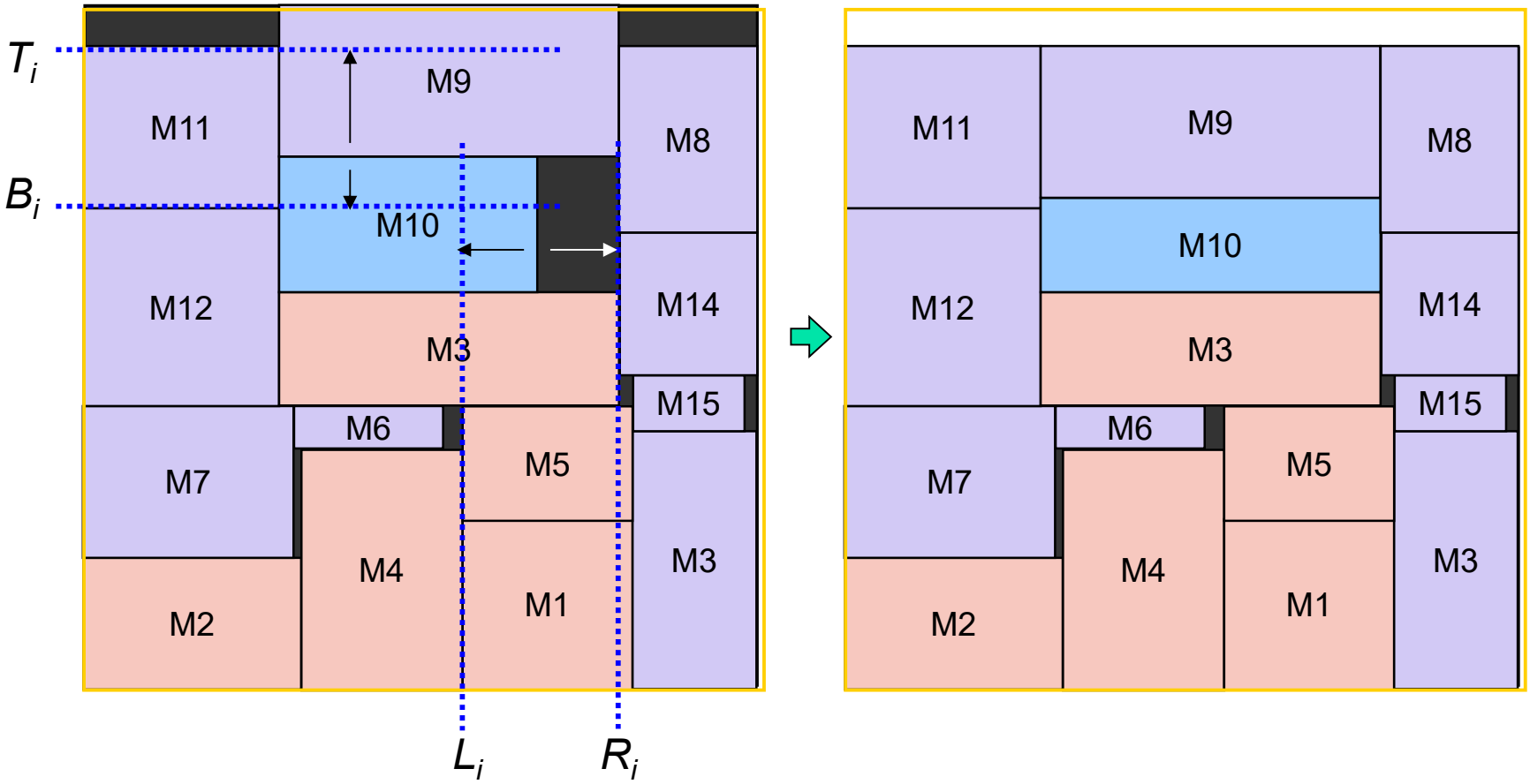
- Op1: rotate a macro
- Op2: delete & insert
- Op3: swap 2 nodes
- Op4: resize a soft macro



Slides Courtesy of Prof. H.-R. Jiang

Simple Floorplan Sizing

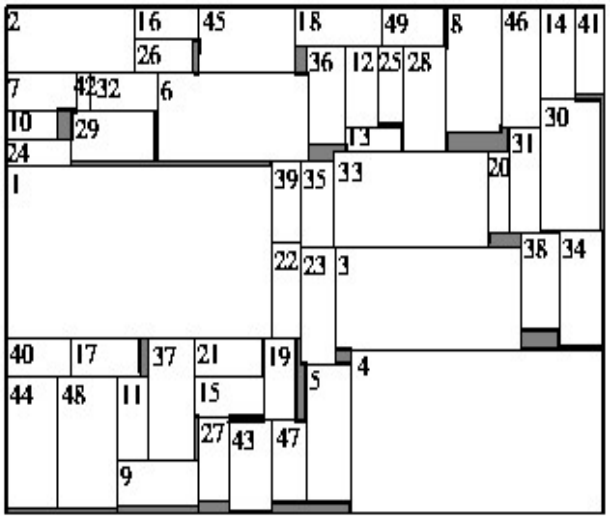
- Key: Line up with adjacent modules
- Advantage: fast and reasonably effective

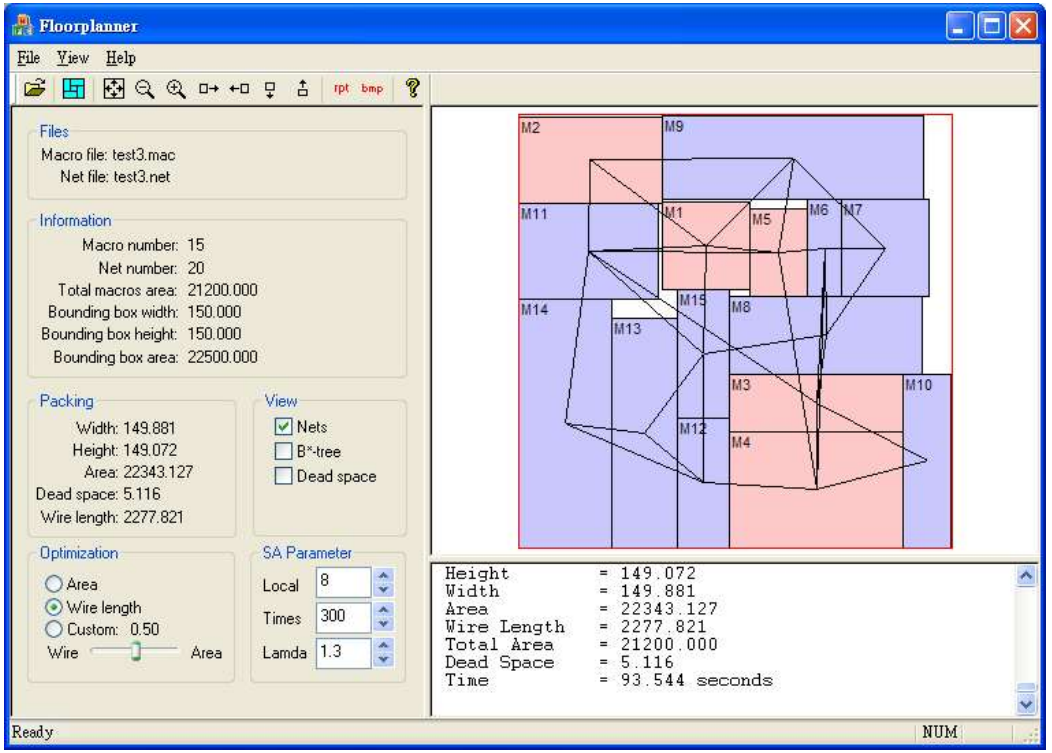


Slides Courtesy of Prof. H.-R. Jiang

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).

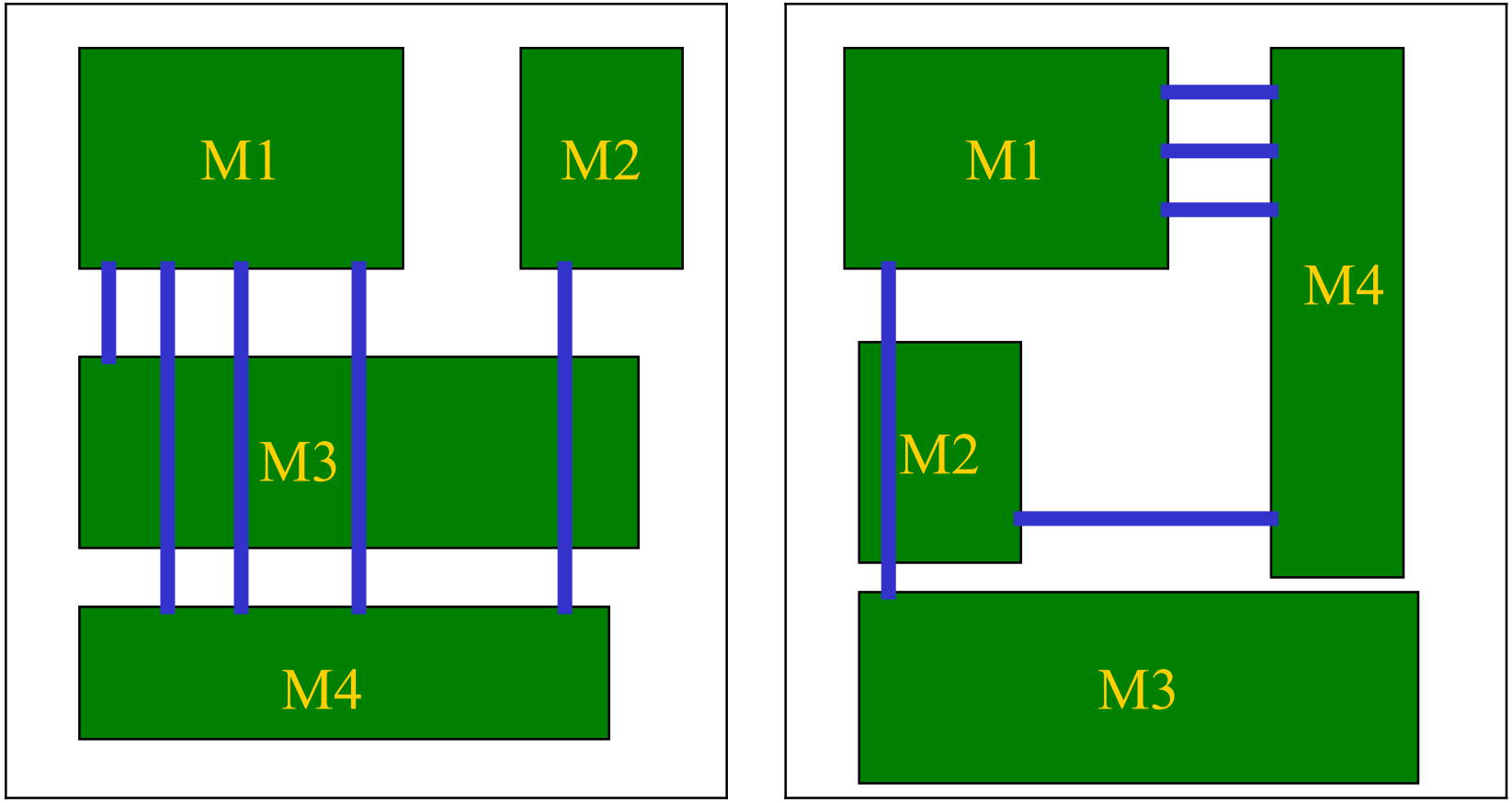




Courtesy T.-C. Chen

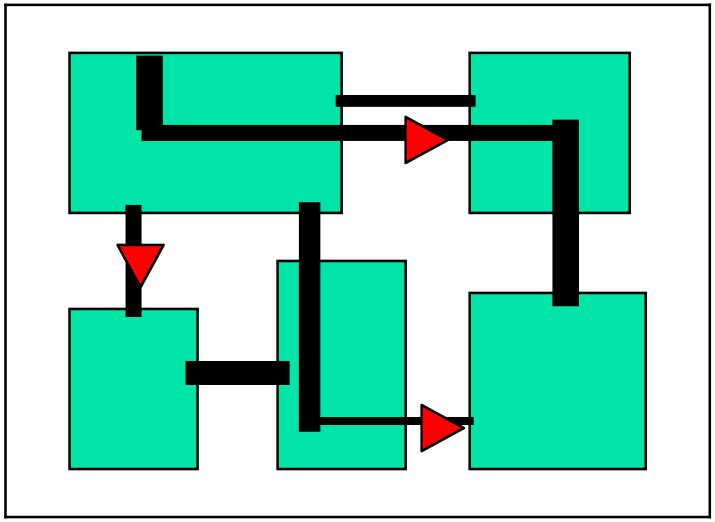
Interconnect-Centric Floorplanning

- Floorplanning greatly influences interconnect structure

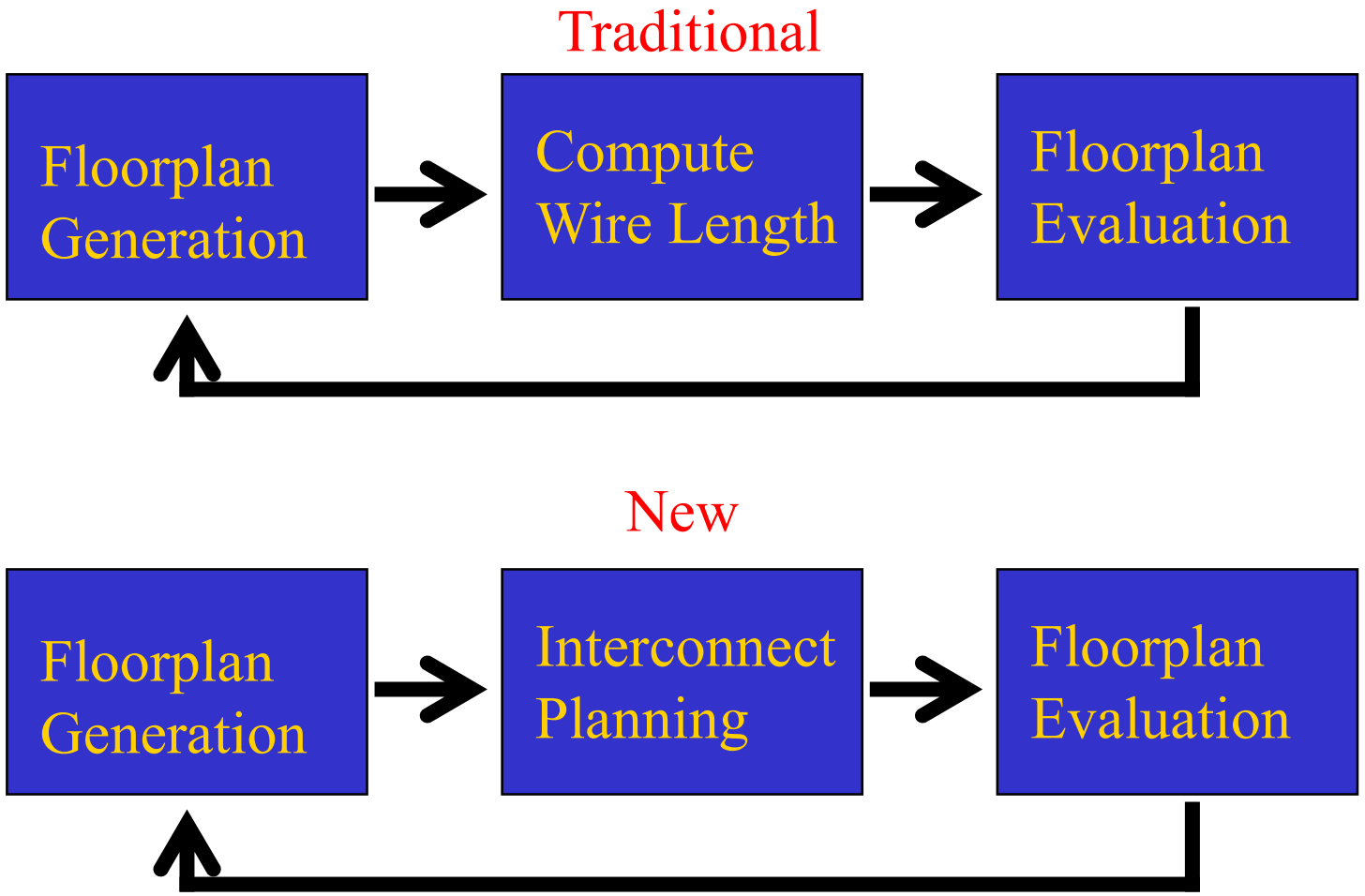


Interconnect Planning

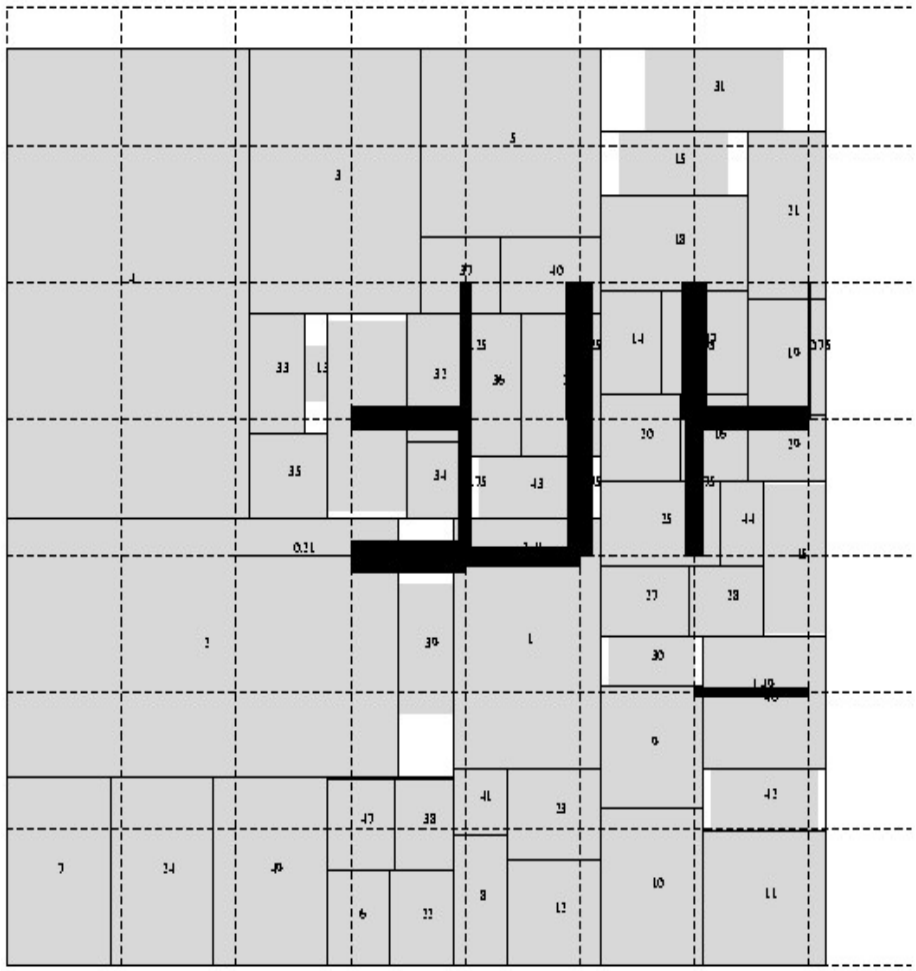
- Pin assignment and routing of global interconnects
- Buffer insertion and sizing
  - Buffer block planning
- Wire sizing



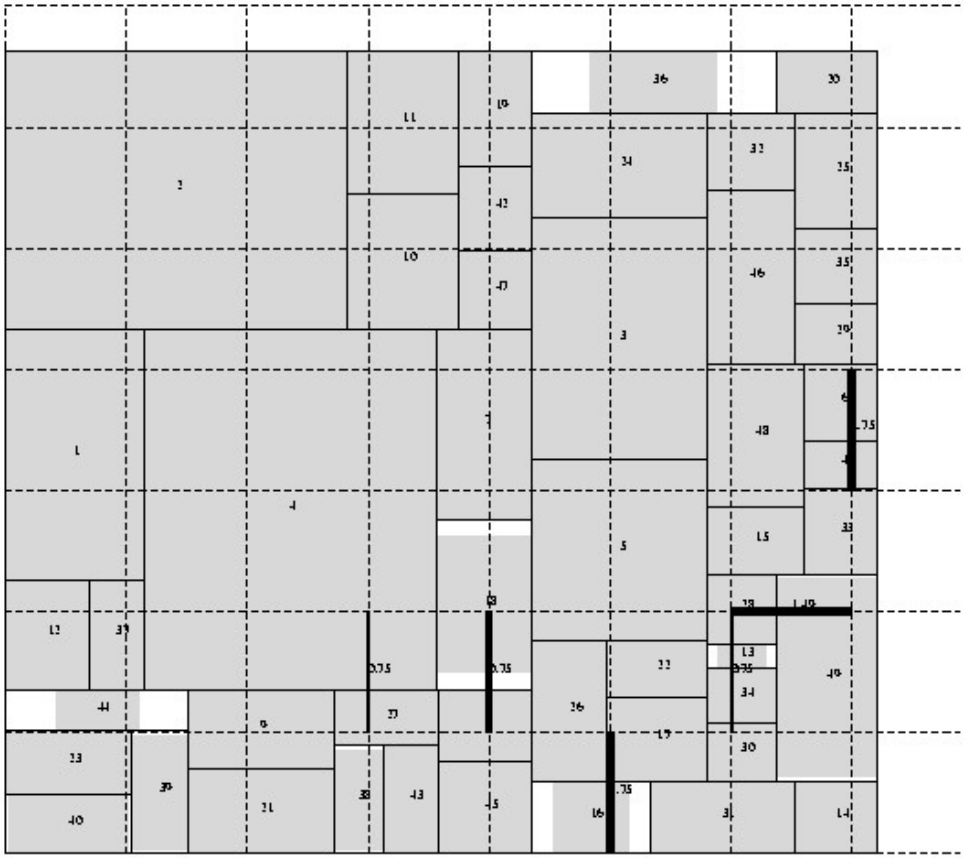
Floorplanning and Interconnect Planning



Interconnect-Centric Floorplanning: ami49 (1/2)

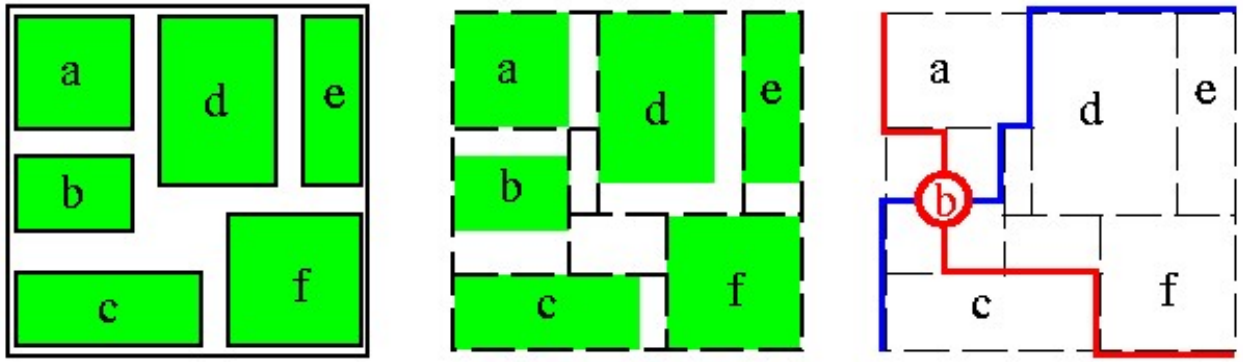


Interconnect-Centric Floorplanning: ami49 (2/2)



Sequence Pair (SP)

- Murata, Fujiyoshi, Nakatake, Kajitani, “Rectangle-Packing Based Module Placement,” ICCAD-95.
- Represent a packing by a pair of module-name sequences (e.g., (*abdecf*, *cbfade*)).
  - Solution space: $(n!)^2$
- Correspond all pairs of the sequences to a P-admissible solution space.
- Search in the P-admissible solution space (by simulated annealing).
  - Swap two nodes only in a sequence
  - Swap two nodes in both sequences

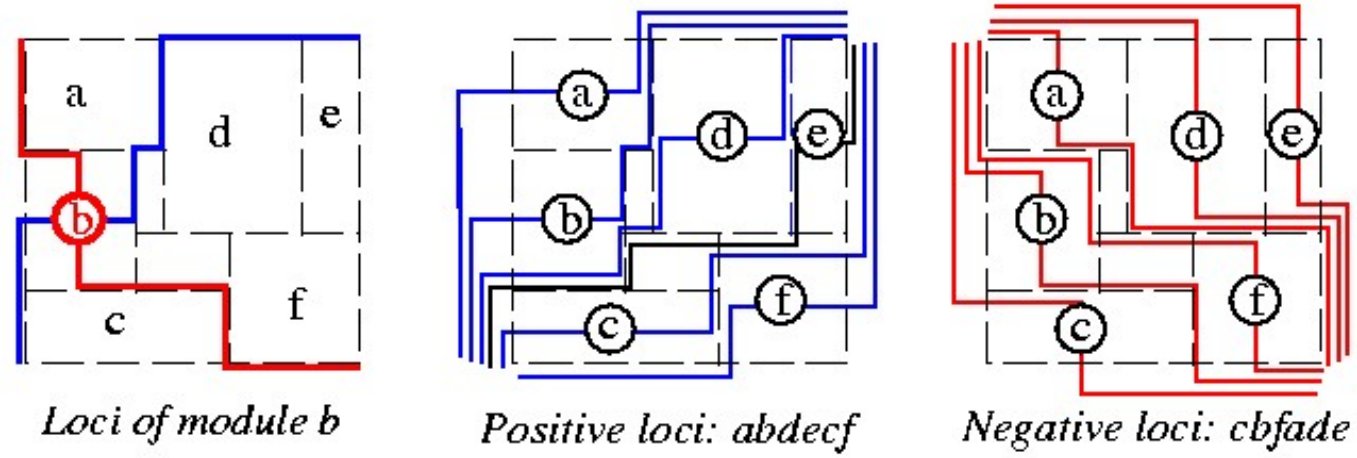


A floorplan Loci of module b

Slides Courtesy of Prof. H.-R. Jiang

Relative Module Positions

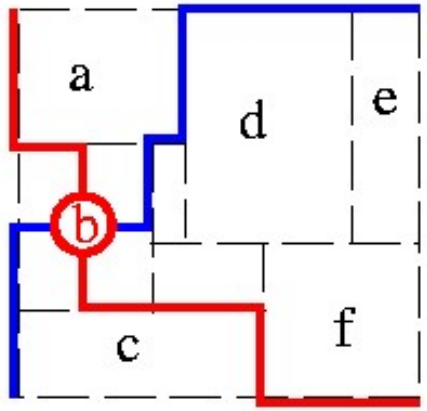
- A floorplan is a partition of a chip into **rooms**, each containing at most one block.
- **Locus** (right-up, left-down, up-left, down-right)
  1. Take a non-empty room.
  2. Start at the center of the room, walk in two alternating directions to hit the sides of rooms.
  3. Continue until to reach a corner of the chip.
- **Positive locus** Γ_+**:** Union of right-up locus and left-down locus.
- **Negative locus** Γ_-**:** Union of up-left locus and down-right locus.



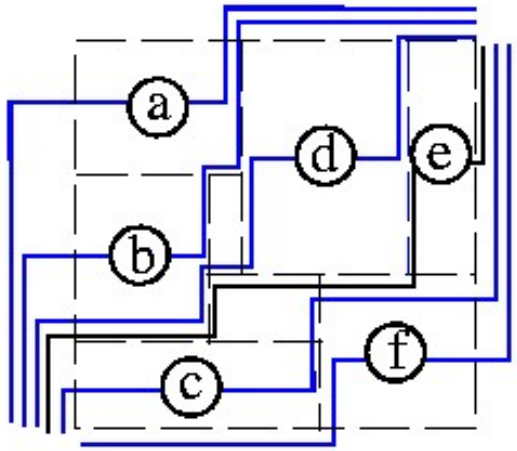
Loci of module b *Positive loci: abdecf* *Negative loci: cbfade*

Geometrical Information

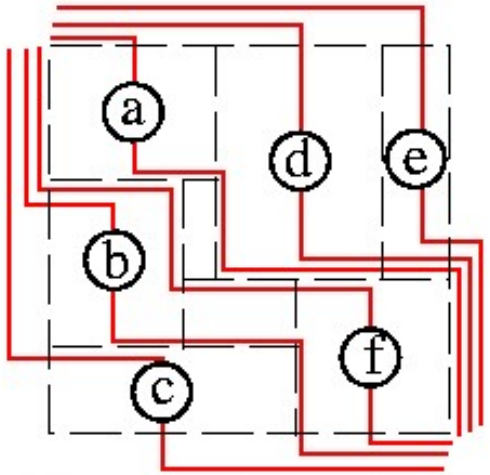
- No pair of positive (negative) loci cross each other, i.e., **loci are linearly ordered**.
- SP uses two sequences (Γ_+, Γ_-) to represent a floorplan.
  - **H-constraint:** (..a..b.., ..a..b..) iff a is on the left of b
  - **V-constraint:** (..a..b..,..b..a..) iff b is below a



Loci of module b



Positive loci: abdecf

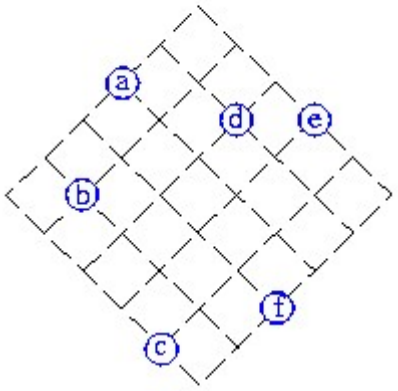


Negative loci: cbfade

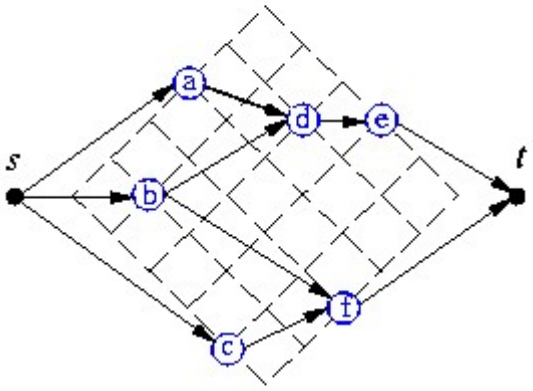
(Γ_+, Γ_-) = (abdecf, cbfade)

(Γ_+, Γ_-)-Packing

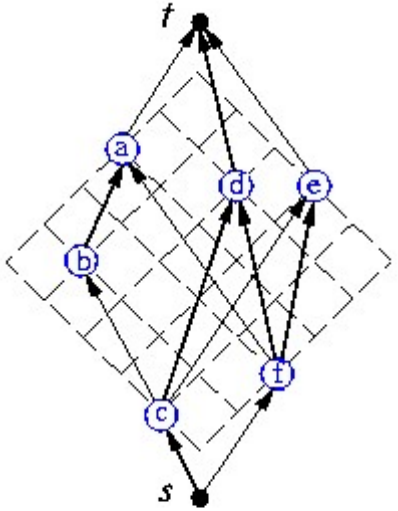
- For every SP (Γ_+, Γ_-), there is a (Γ_+, Γ_-) packing.
- **Horizontal constraint graph** $G_H(V, E)$ (similarly for $G_V(V, E)$):
  - V: source s, sink t, n vertices for modules.
  - E: (s, x) and (x, t) for each module x, and (x, y) iff x must be left to y.
  - **Vertex weight:** 0 for s and t, **width** of module x for the other vertices.



Packing for sequence pair: (ubdecf, cbfude)



Horizontal constraint graph (Transitive edges are not shown)



Vertical constraint graph (Transitive edges are not shown)