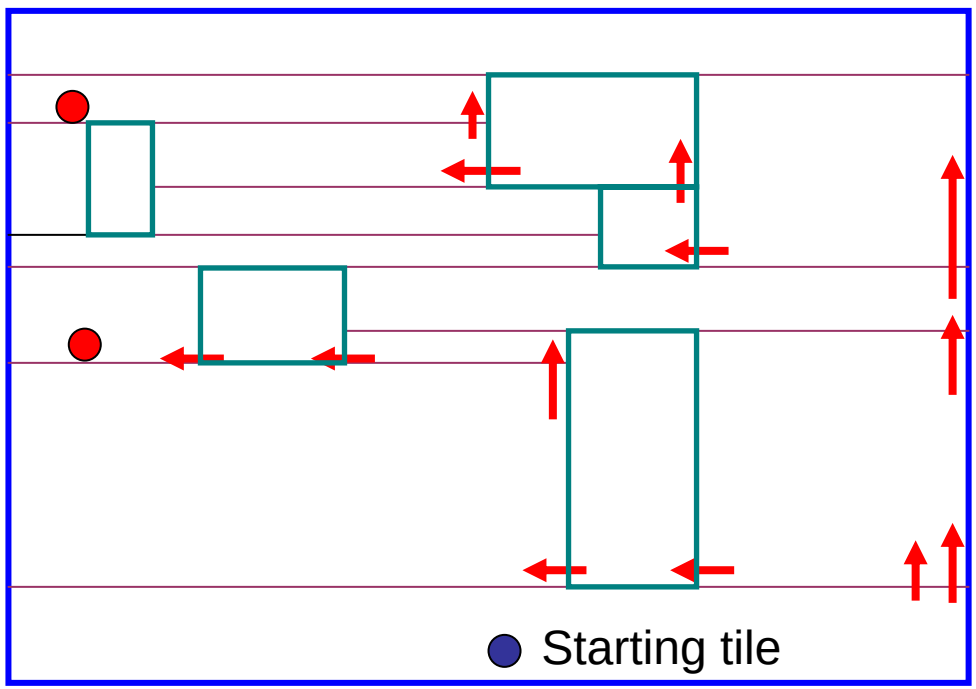
Unit 5G: Layout Editor

- Course contents:
  - Introduction
  - Components of a layout editor
  - Layout representation
  - Semi-automatic editing/verifying functions

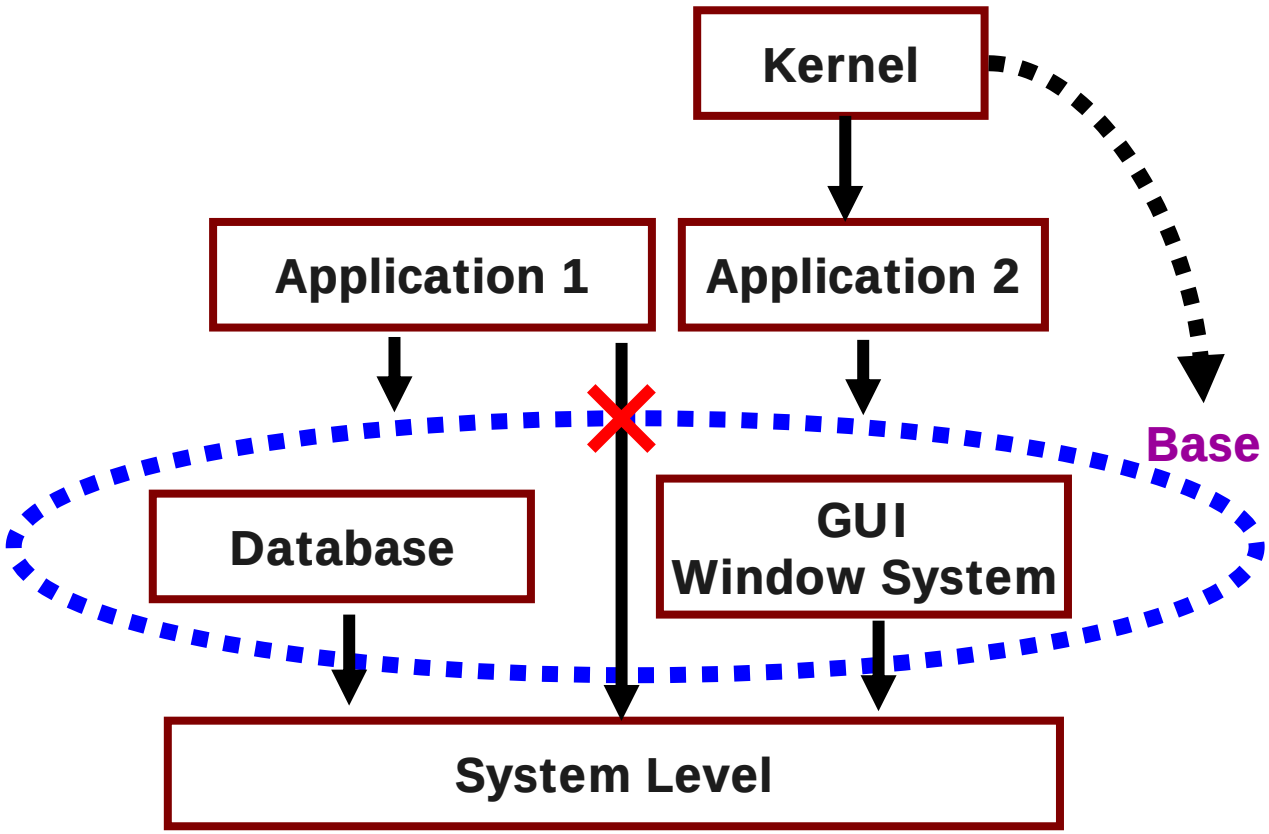


Introduction

- Layout editor is not only a graphic tool, but also a complex layout system that contains
  - Basic interactive geometry editing functions: creation, sizing, Boolean operation, rotation, and flipping.
  - Design hierarchy management.
  - Semi-auto editing/verifying functions: point-to-point routing, on-line design rule checking (DRC), transistor-chain creation, short locator, connection tracing, circuit extraction, and plowing.
- Layout editor
  - Integrates with the technology file containing all the design information, such as system resolution, design rules, and other template definitions for advanced semi-auto editing functions.
  - is important for full custom layout design as well as ECO editing.
  - Ex. Magic layout system by J. K. Ousterhout et al. 1984

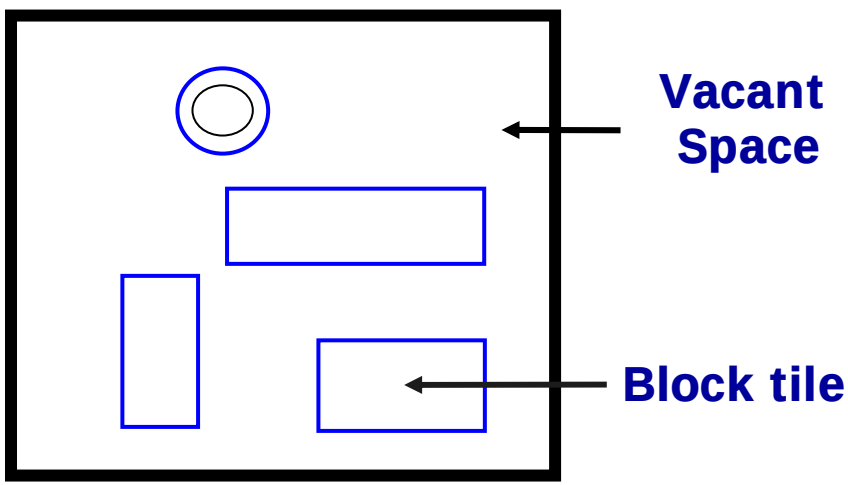
Introduction

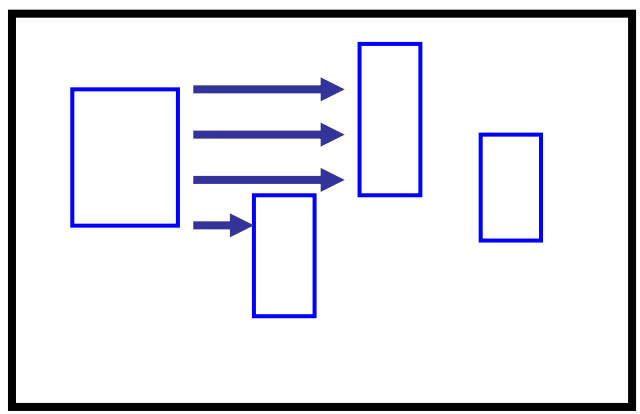
- Level hierarchy of a layout editor



Components of a Layout Editor

- Components
  - Operand: database entry
    - Flat: Dot, line, circle, rectangle, path, polygon, donut, etc.
    - Hierarchical: object, instance, cell, etc.
  - Operator: atomic operation
    - Point finding: find the block containing the point
    - Neighbor finding: find all blocks touching a given block
    - Block visibility



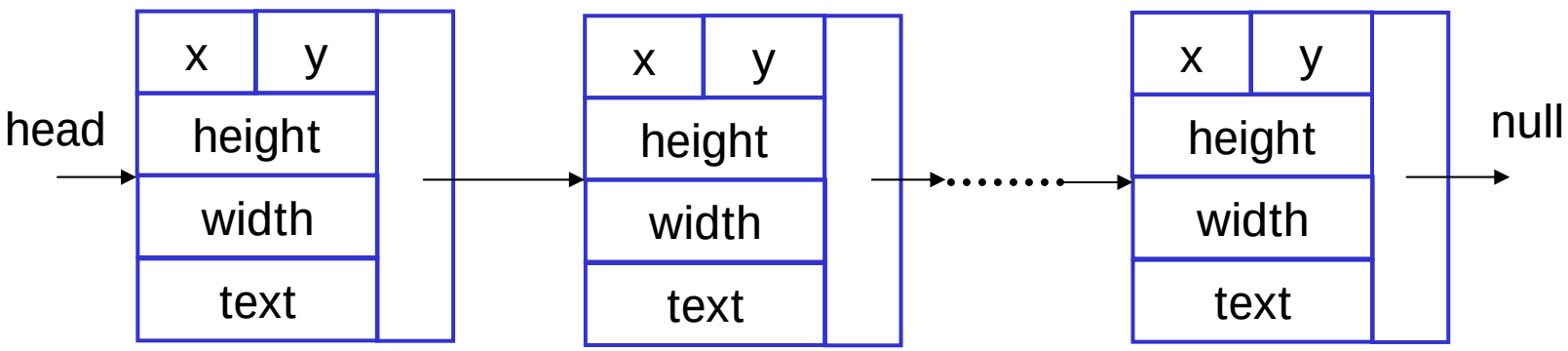


Components of a Layout Editor

- Components (continue)
  - Operator Area searching – check if there is any block in an area
    - (Directed) area enumeration – visit each tile intersecting with the area exactly once (in sorted order)
    - Block insertion
    - Block deletion
    - Plowing – move an object in one direction and then force associated objects to move
    - Compaction – plow the entire layout
      - One dimensional vs. two dimensional
    - Channel generation

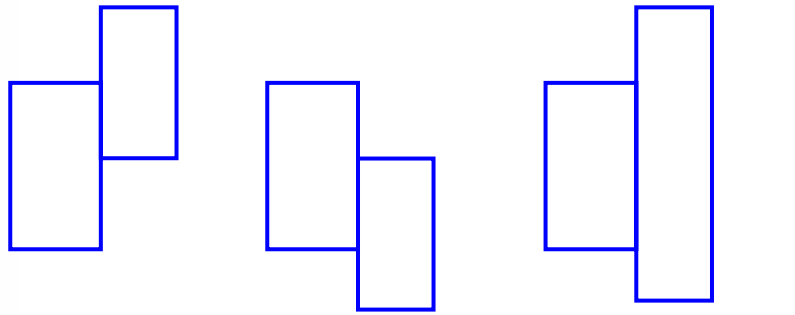
Layout Representation: Linked List

- Layout Representation: Linked list of blocks
  - Only suitable for a hierarchical system where each level contains few blocks
  - Space complexity $O(n)$, n is the number of blocks



Neighbor Finding Algorithm

```
Algorithm NEIGHBOR-FIND1(B, L)
begin
    neighbor-list = φ;
    x1 = B.x + B.width;
    y11 = B.y;
    y12 = B.y + B.height;
    for each block R ∈ L such that R ≠ B do
        y21 = R.y;
        y22 = R.y + R.height;
        if (x1 = R.x and (y11 ≤ y21 ≤ y12 or
                          y11 ≤ y22 ≤ y12 or
                          y21 < y11 < y12 < y22)) then
                INSERT(R, neighbor-list);
    return neighbor-list;
end.
```



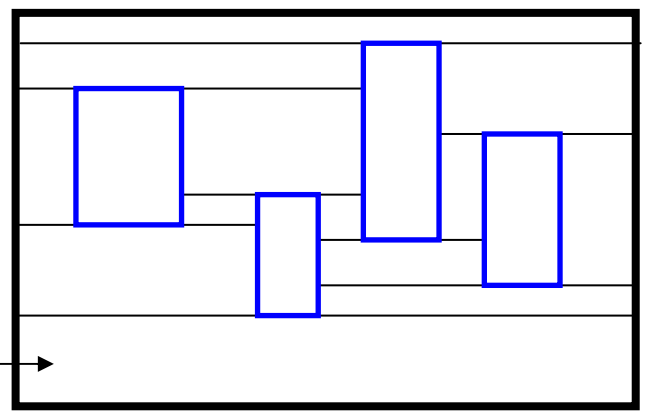
y11≤y21 ≤y12 **y11≤y22 ≤y12** **y21≤y11 ≤y12 ≤y22**

$y_{min} = \mathrm{MAX}(y_{11}, y_{21})$, $y_{max} = \mathrm{MIN}(y_{12}, y_{22})$

$y_{min} < y_{max} \rightarrow$ *overlap*

$y_{min} > y_{max} \rightarrow$ *non-overlap*

- Disadvantage: vacant space
- Modified linked list
  - maximal horizontal strips → vacant tiles



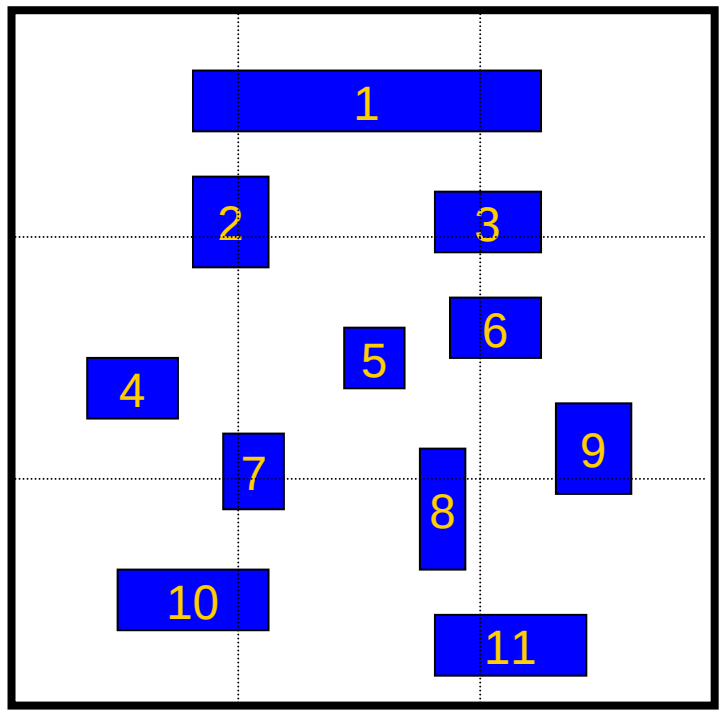
Vacant tile

Layout Representation: Bin-Based Method

- Layout is superimposed by virtual grids (row,col)
- Space complexity $O(bn)$, b is the number of bins
- Easily degenerated to the linked list: all blocks are in a bin
- Worse performance for neighbor finding, area searching, area enumeration
  - Worst-case complexity: $O(b + n)$
- Sensitive to time-space tradeoff

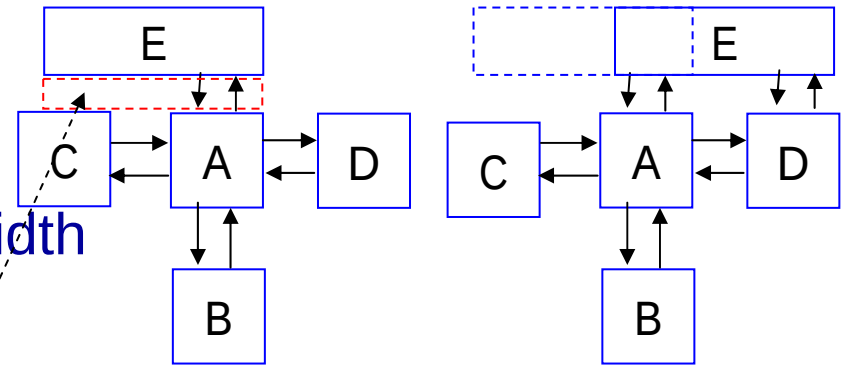
Neighbor Finding Algorithm

Algorithm NEIGHBOR-FIND2(A, $\mathcal{B}$)
begin
    $neighbor\text{-}list = \phi$;
    $x1 = A.x + A.width$;
    $y11 = A.y$;
    $y12 = A.y + A.height$;
    let $B' \subseteq \mathcal{B}$ be set of bins which contain A;
    for all bins $X \in B'$ **do**
        for each block $R \in X$ **do**
            $y21 = R.y$;
            $y22 = R.y + R.height$;
            if ($x1 = R.x$ **and** ($y11 \leq y21 \leq y12$ **or**
                $y11 \leq y22 \leq y12$ **or**
                $y21 < y11 < y12 < y22$)) **then**
                INSERT($R, neighbor\text{-}list$);
    return $neighbor\text{-}list$;
end.

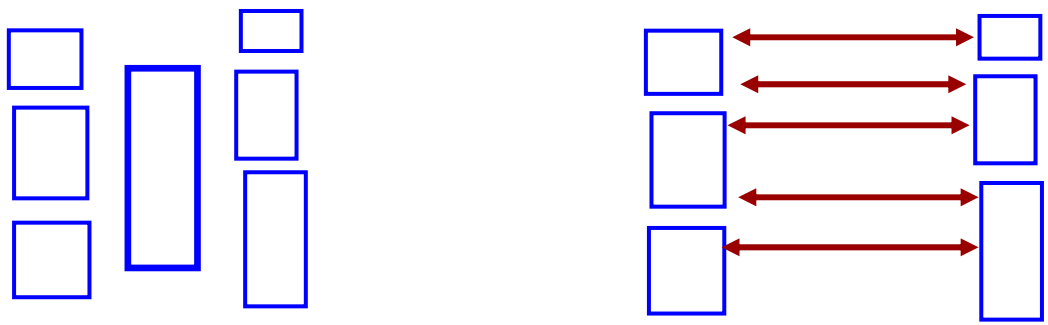


Layout Representation: Neighbor Pointers

- Keep neighboring info
- Space complexity $O(n^2)$
- Block representation
  - Upper left corner, length, width
- Easy for plowing operation
- Difficult to generate channels
- Difficult for updating operations
- $O(n^2)$ for a plowing operation which modifies the neighbors of all blocks
- $O(n)$ for block insertion and block deletion

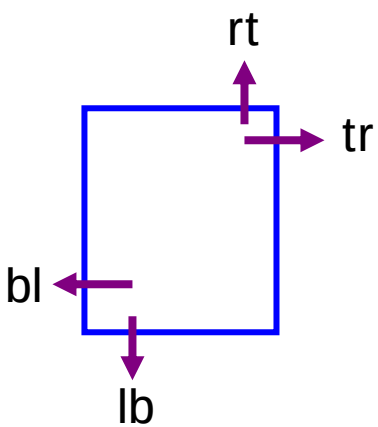


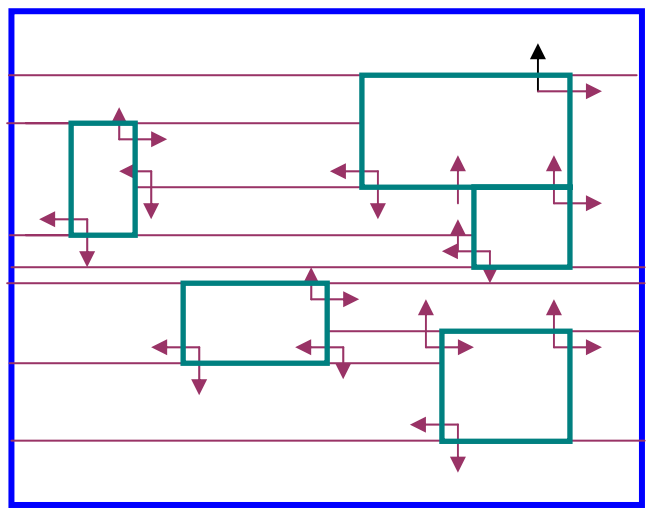
Neighbor pointer update



Layout Representation: Corner Stitching

- The first data structure to represent both block tiles and vacant (space) tiles
- Keep four neighboring pointers
- Rapid stretching, compaction, neighbor-finding and channel finding
- Also efficient block insertion and deletion
- Need more memory space

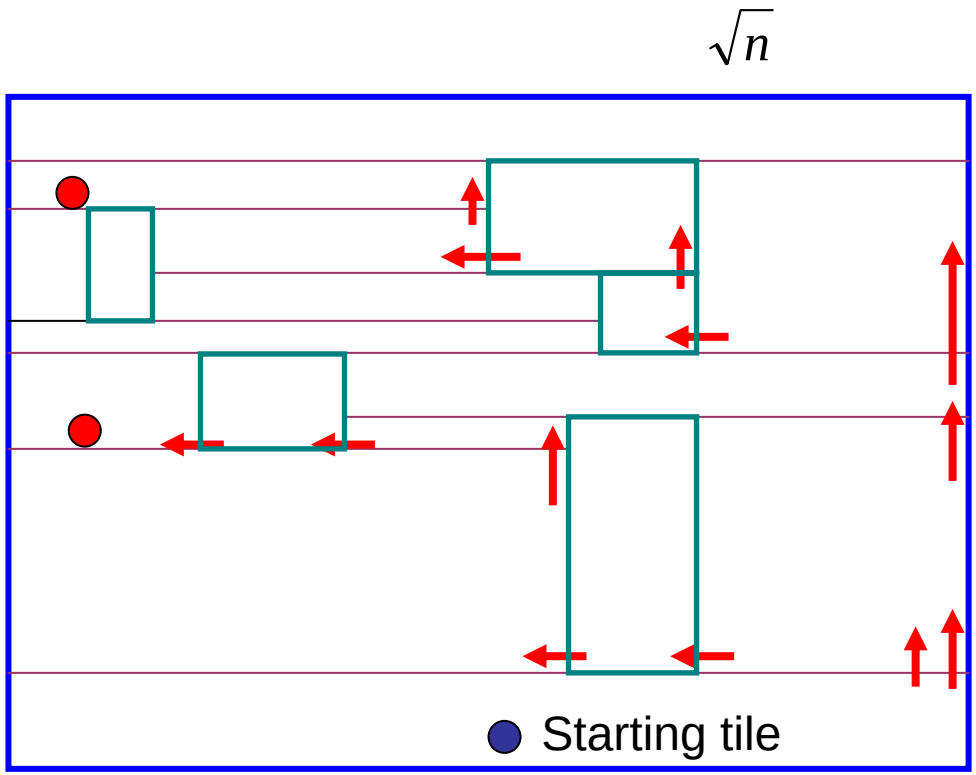




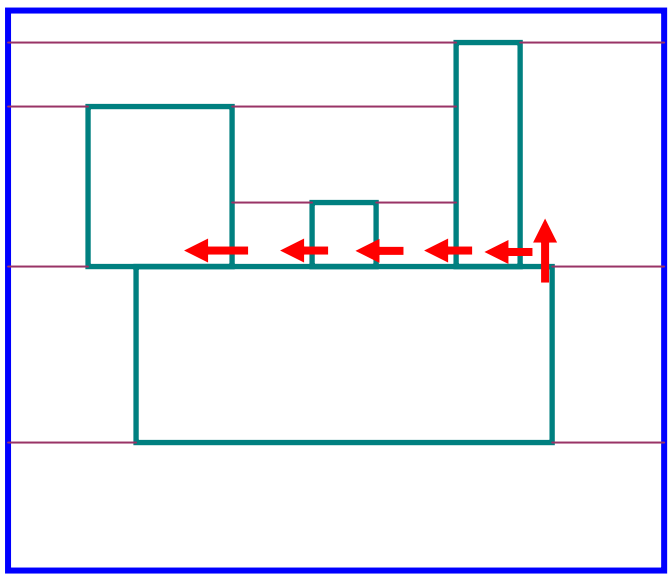
Semi-auto Editing/Verifying Functions

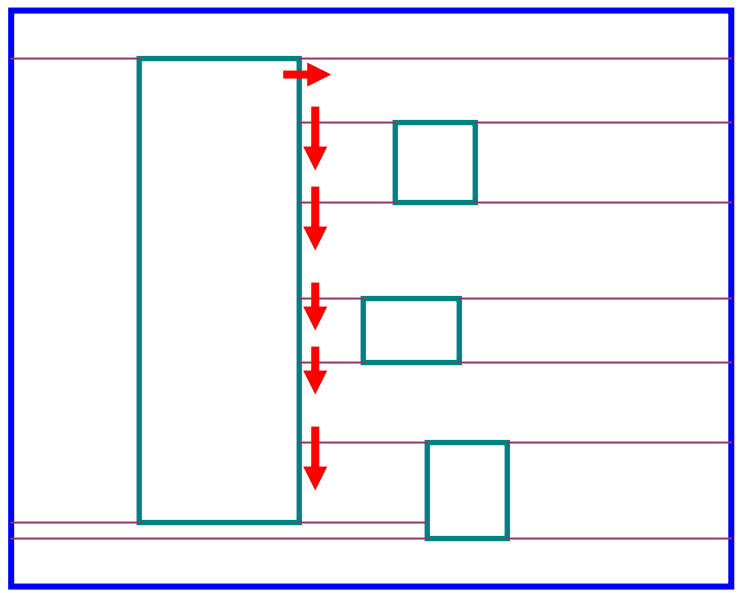
- Graphic editing
  - **Point finding, neighboring finding, area searching, block enumeration, block insertion/deletion.**
- Semi-auto editing function
  - point-to-point routing, transistor-chain creation, and **plowing**.
- Semi-auto verifying function
  - **on-line design rule checking (DRC),** short locator, connection tracing, and **circuit extraction.**
- Corner stitching is a good data structure to implement these functions.
  - The highlighted items in red will be illustrated shortly.

Corner Stitching: Point finding



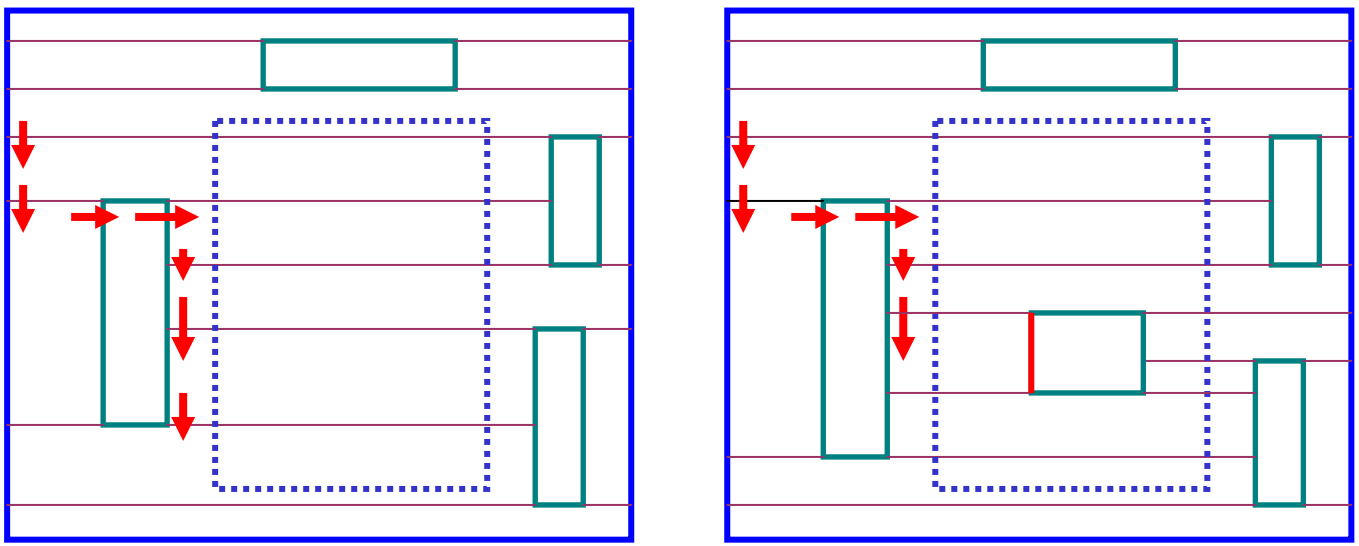
Corner Stitching: Neighbor Finding





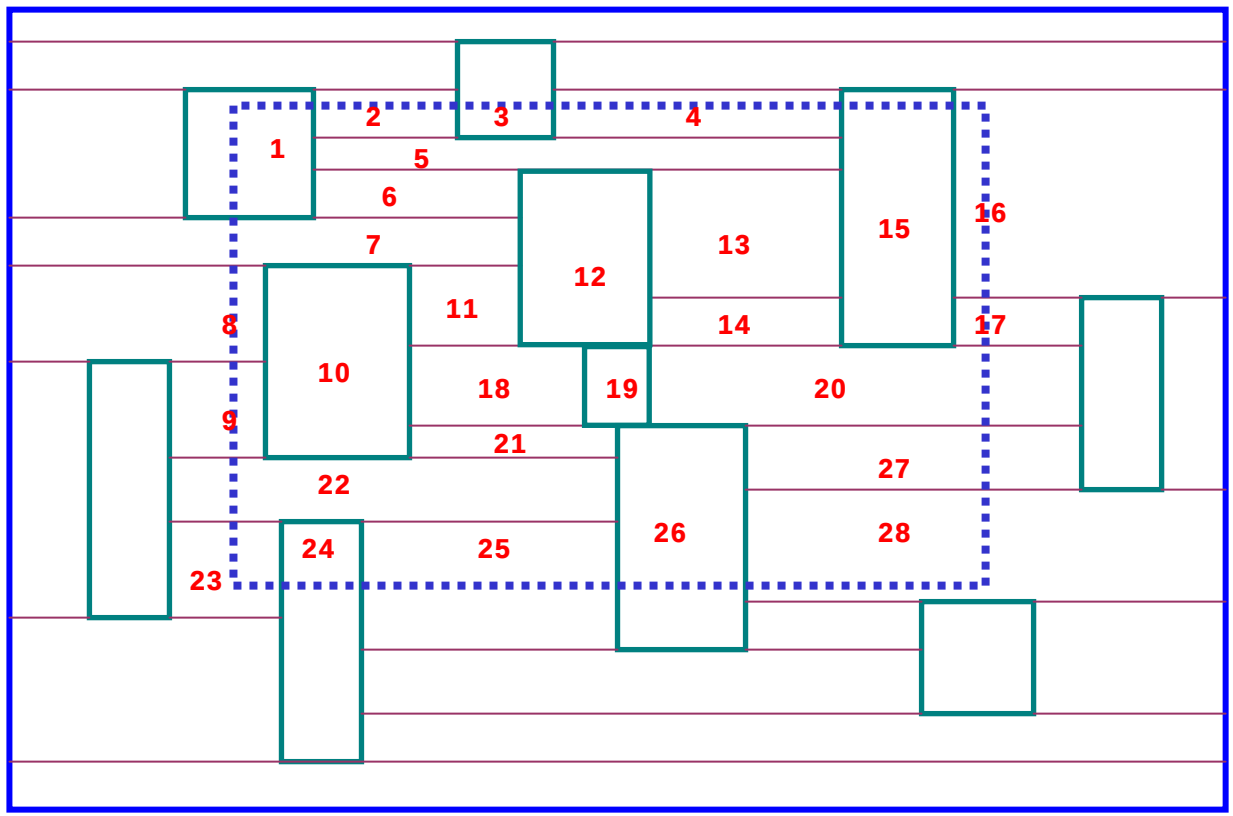
Corner Stitching: Area Searching

- Area Searching: search if there is block tile inside an area
  - First do point location
  - Check the right side of each tile containing the left edge of the searching area

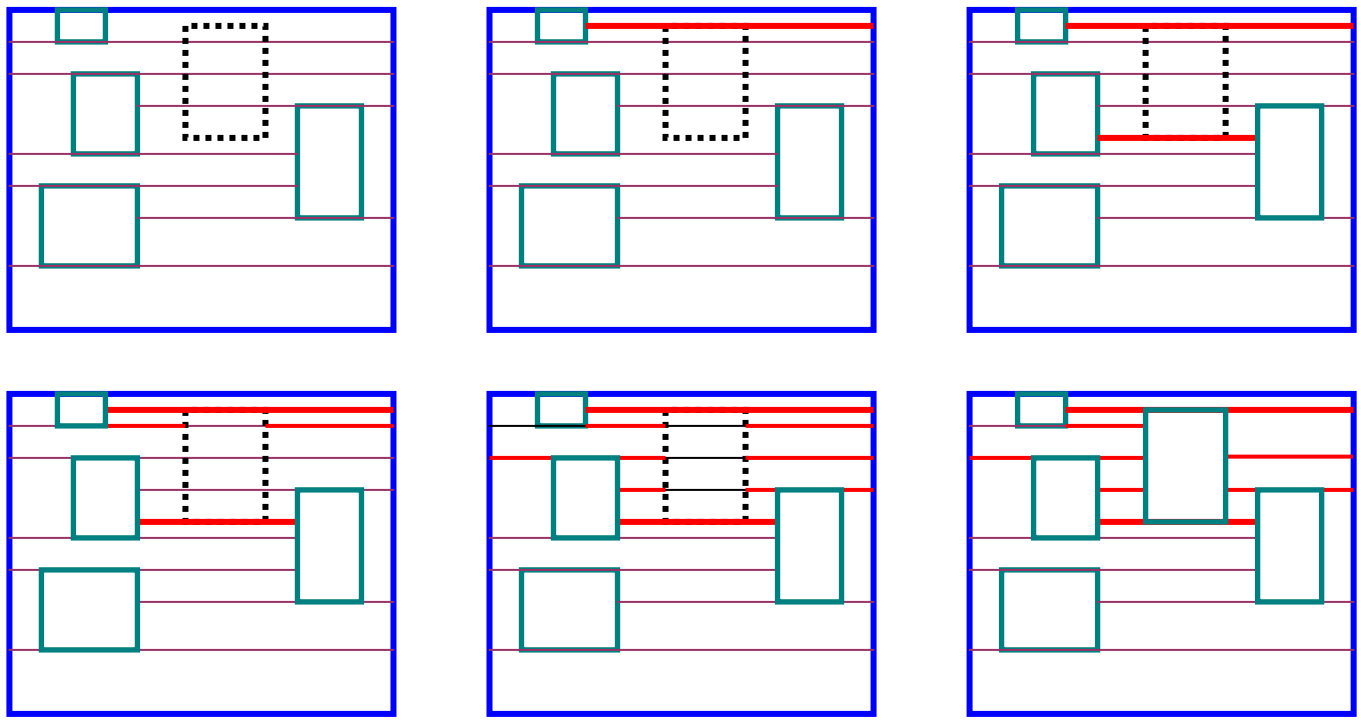


Corner Stitching: Tile Enumeration

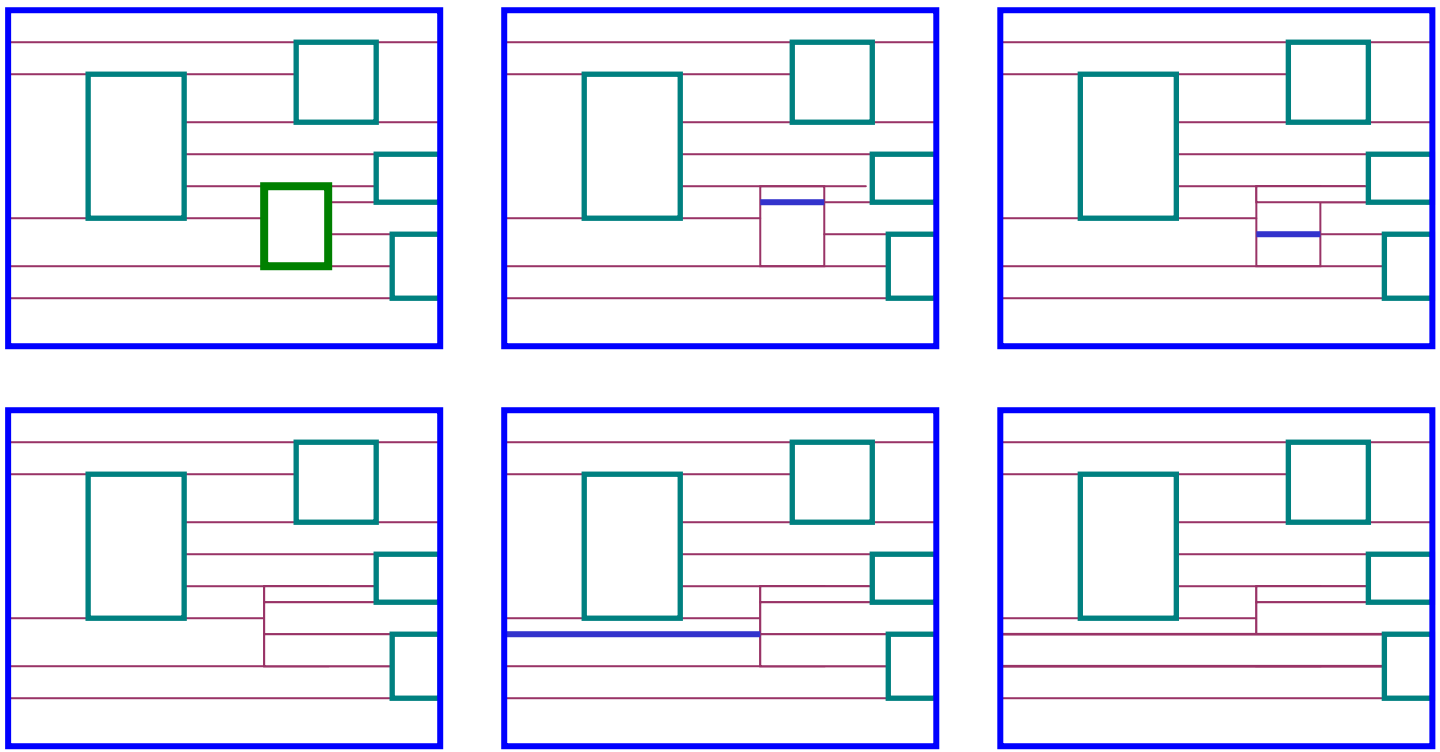
- Enumerate all tiles: first DFS, and then BFS



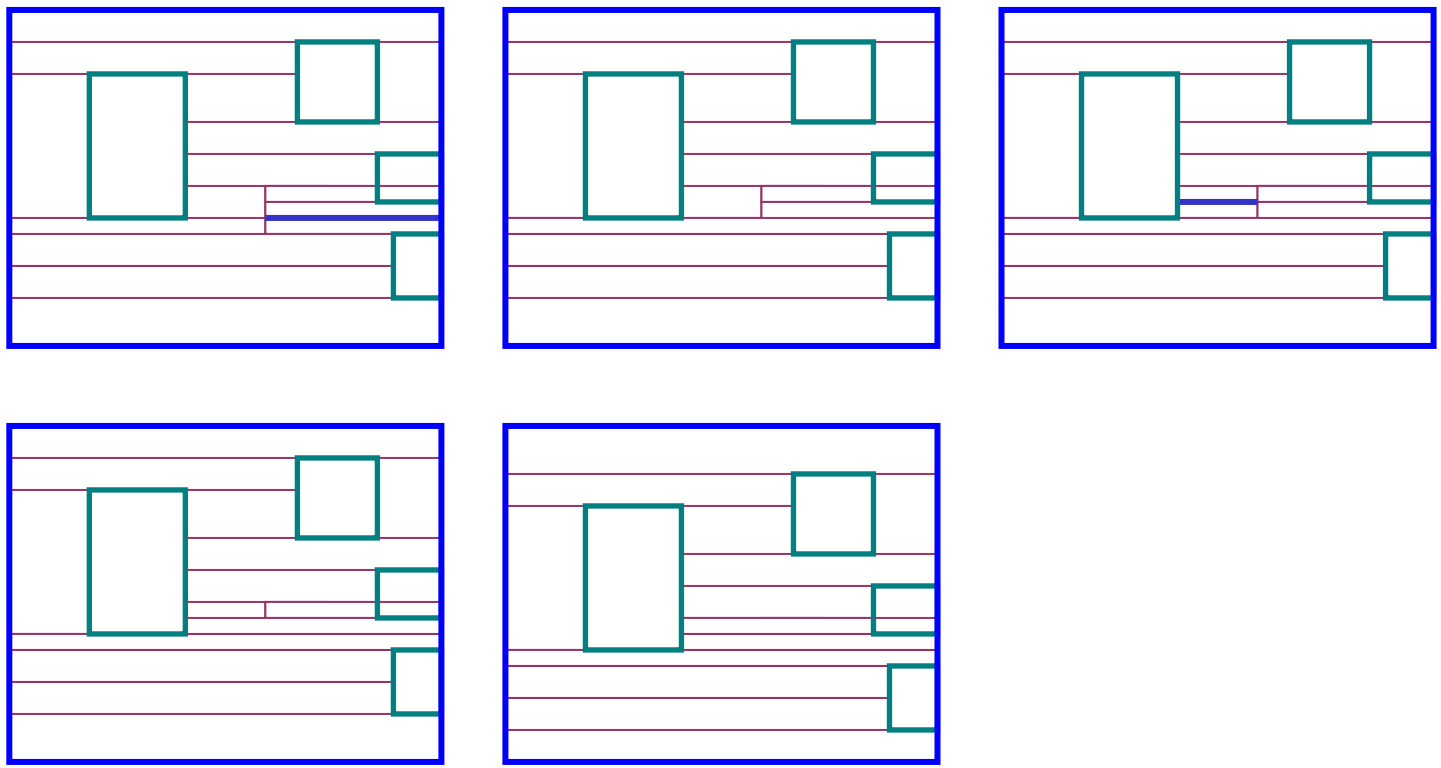
Corner Stitching: Block Insertion



Corner Stitching: Block Deletion

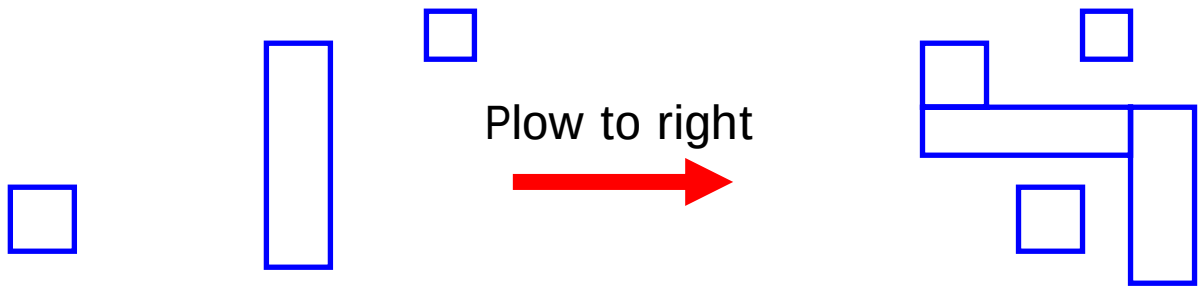


Corner Stitching: Block Deletion (cont'd)

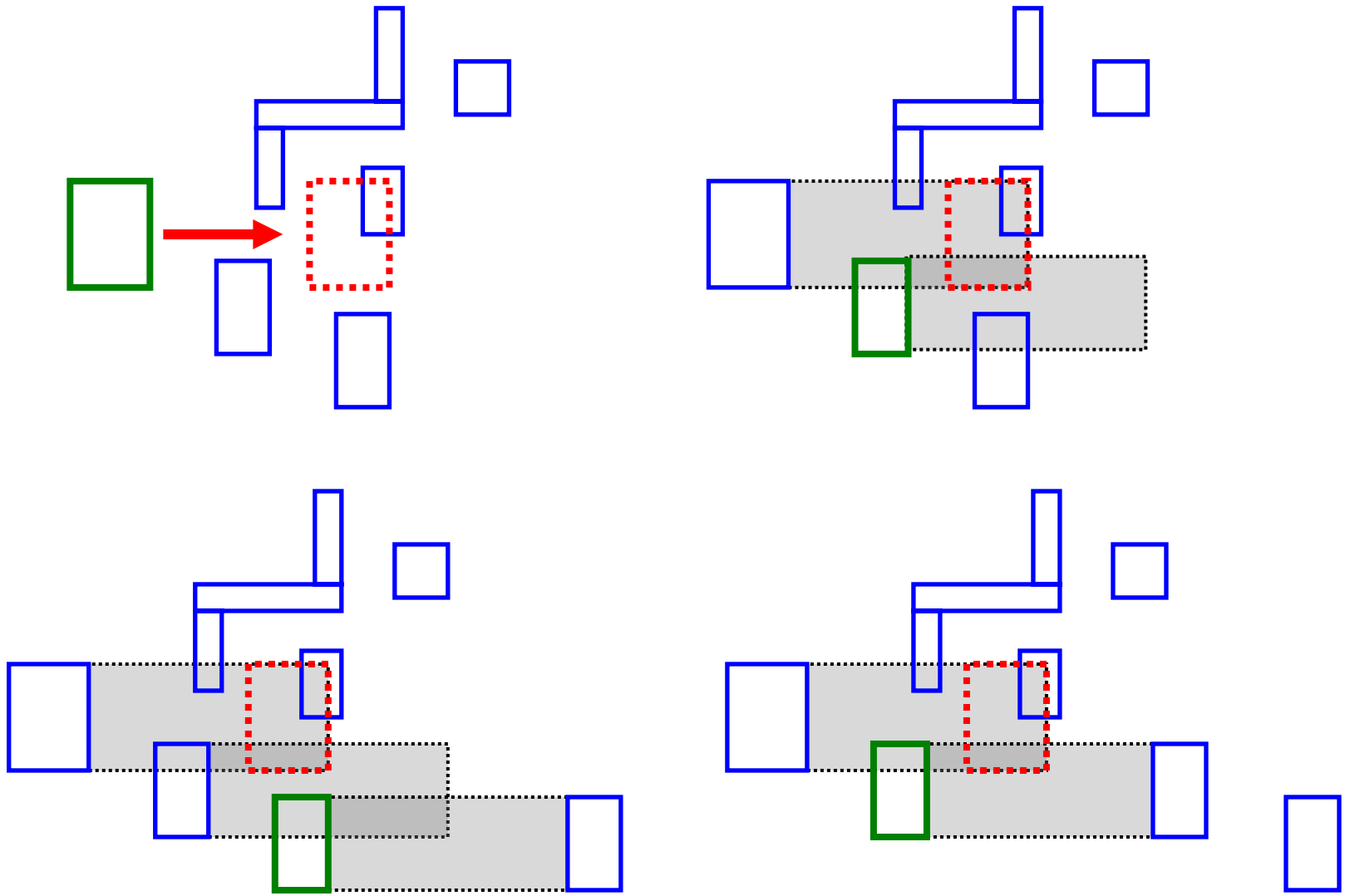


Corner Stitching: Plow

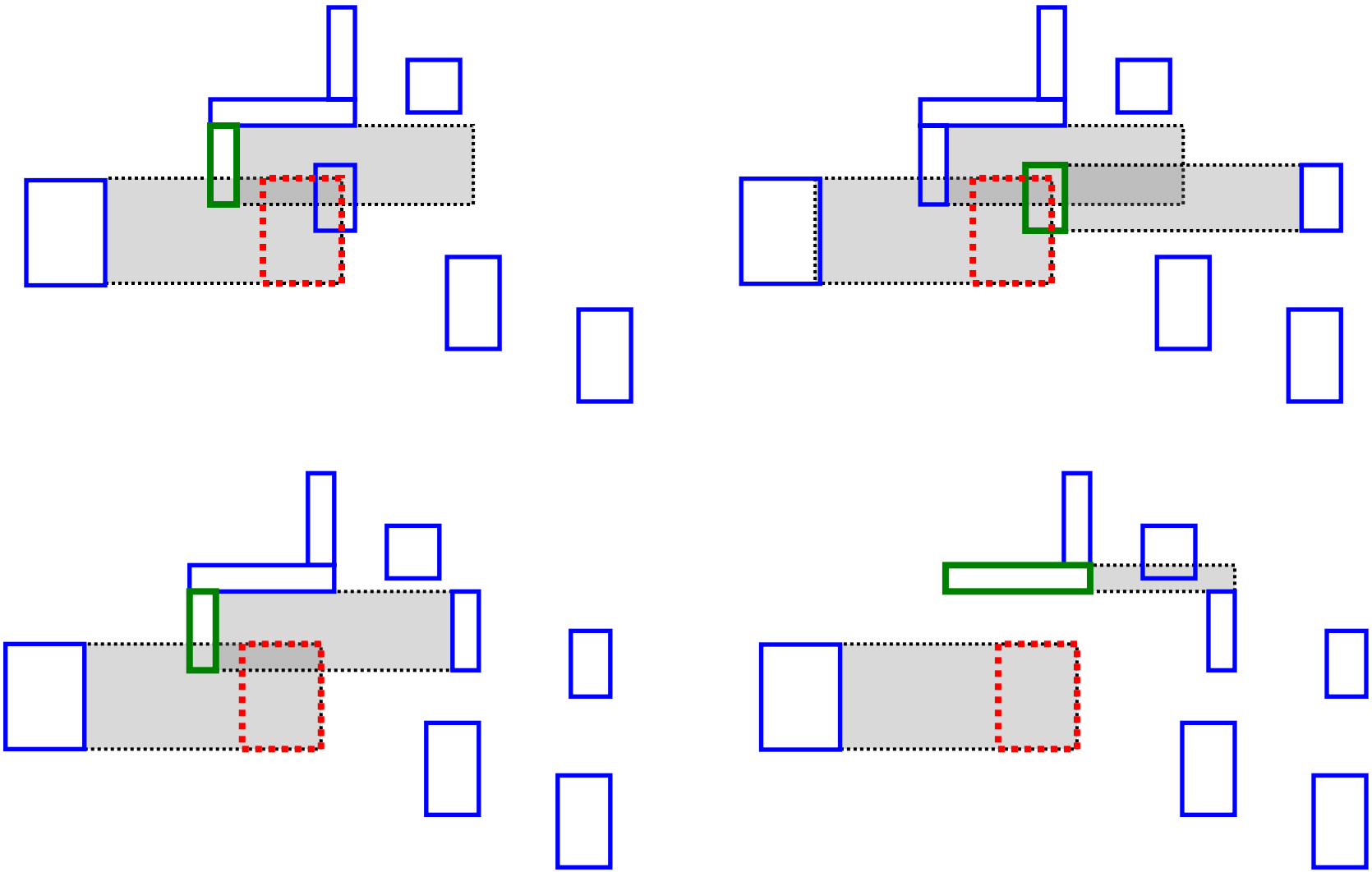
- Plow: recursive area search and compression



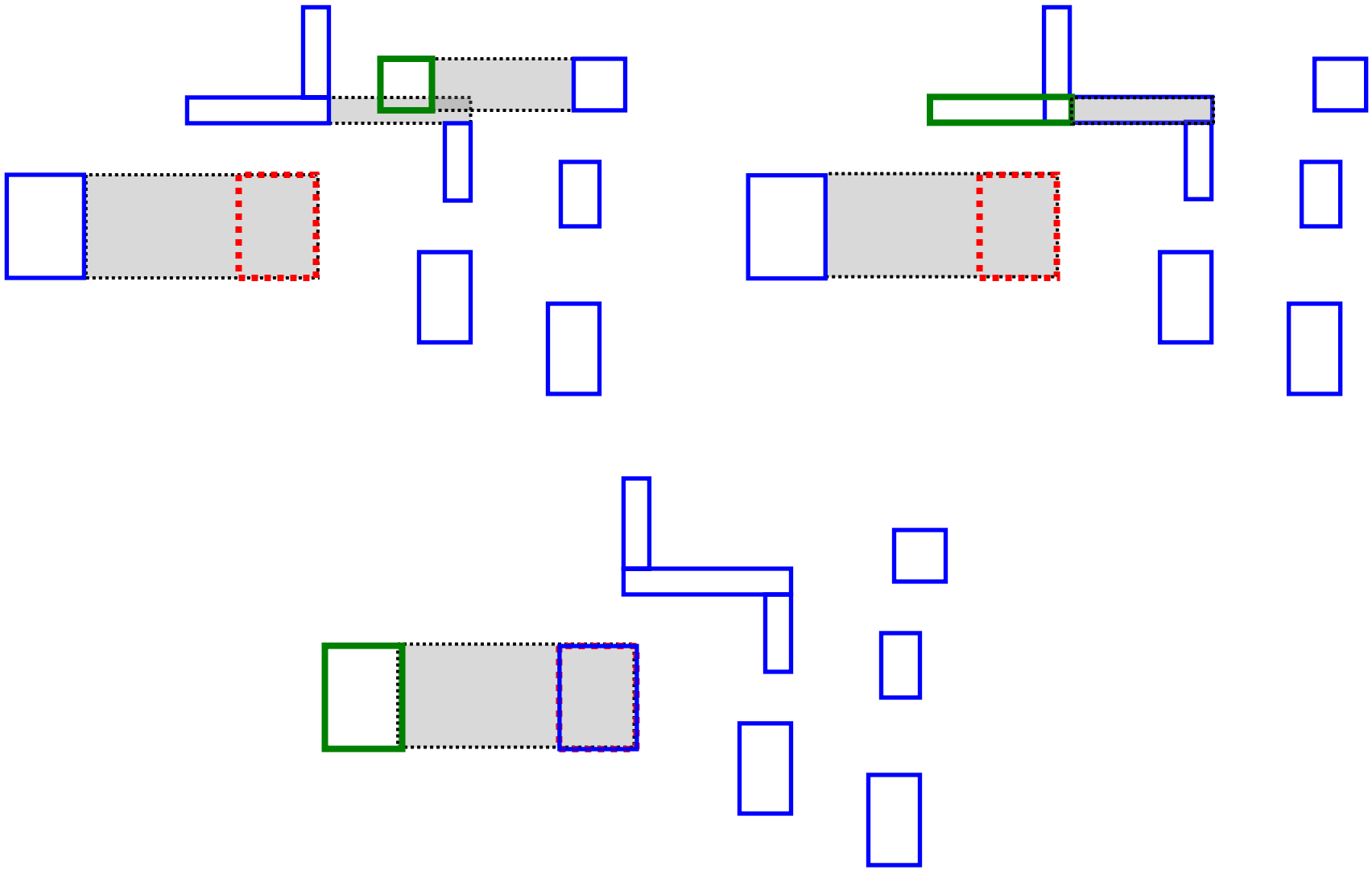
Plowing Example



Plowing Example (cont'd)



Plowing Example (cont'd)



Corner Stitching: On-Line Extraction

- On-Line Extraction: use geometry Boolean operation