# Object-based and Image-based Image and Object Representations


Hanan Samet

hjs@cs.umd.edu www.cs.umd.edu/˜hjs

Department of Computer Science

Center for Automation Research

Institute for Advanced Computer Studies

University of Maryland

College Park, MD 20742, USA

# Image and Object Representations

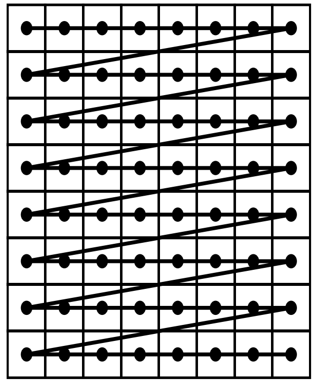
- Representation of spatial objects and their environment
- Usually decompose into collections of more primitive elements (termed *cells*) each of which has a location in space, a size, and a shape
  1. decompose objects into subobjects of varying shape
     - table consists of a flat top in the form of a rectangle and four legs in the form of rods whose lengths dominate their cross-sectional areas
     - object-based
  2. decompose environment into cells of uniform shape
     - image-based
- Queries:
  1. feature query: given an object, determine its constituent cells (i.e., their locations in space)
  2. location query: given a cell (i.e., location in space), determine the identity of object (or objects) of which it is a member as well as remaining constituent cells of object (or objects) via connected component labeling
- Interior-based vs: boundary-based representations

# Interior-based Representations: Unit-size Cells

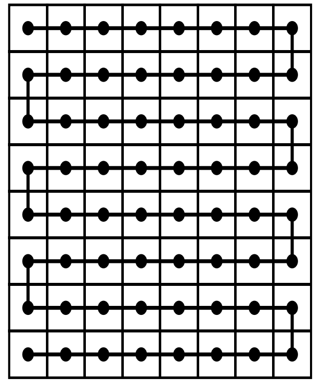
- Aggregate identically-valued cells by recording their locations in space
  1. explicit: if identities of the contiguous cells forming the object are hardwired into the representation
     - e.g., associates a set with each object $o$ that contains the location in space of each cell that comprises $o$
     - shortcoming is need to examine every cell to detect if particular cell is occupied by an object
     - resolve by storing an approximation with each object (e.g., bounding box)
     - object-based representation
  2. implicit: allocate an address $a$ in storage for each cell $c$ where an identifier is stored that indicates the identity of the object (or objects) of which $c$ is a member
     - must have a way of finding the address $a$ corresponding to $c$, taking into account that there is possibly a very large number of cells, and then retrieving $a$
     - known as an access structure
     - index on locations of space
     - image-based representation

# Ordering Space

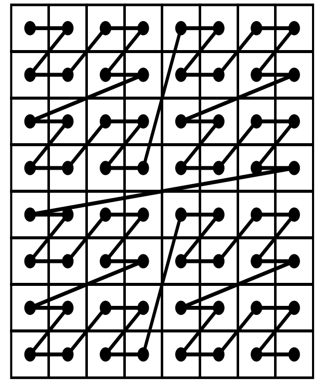
- Many ways of laying out the addresses corresponding to the locations in space of the cells each having its own mapping function
- Can use one of many possible space-filling curves
- Important to distinguish between *address* and *location* or *cell*
- *Address of a location or cell* ≡ physical location (e.g., in memory, on disk, etc.), if any, where some of the information associated with the *location* or *cell* is stored



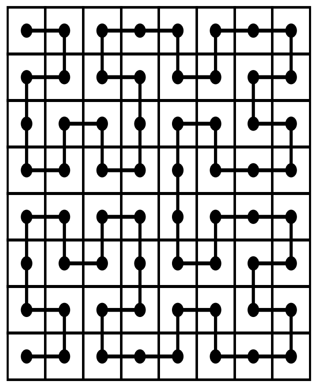
row order



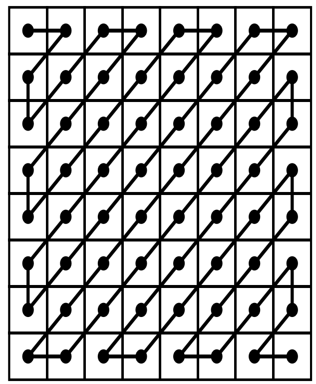
row-prime order



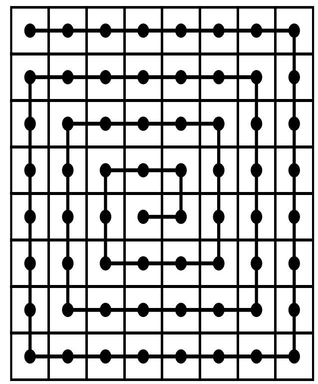
morton order



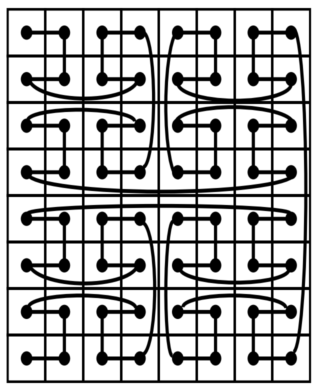
peano-hilbert order



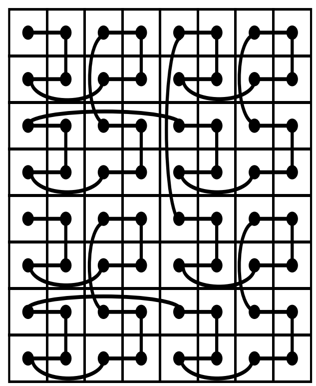
cantor-diagonal order



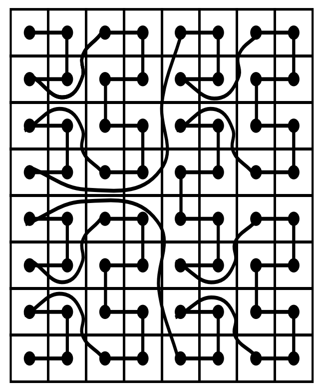
spiral order



gray code



double gray order



u order

# Array Access Structure

- Given a cell $c$ at a location $l$ in space, array enables us to calculate the address $a$ containing the identifier of the object associated with $c$
- Array is only a conceptual multidimensional structure (not a multidimensional physical entity in memory) in the sense that it is a mapping of the locations in space of the cells into sequential addresses in memory
  1. Actual addresses are obtained by the array access function, which is usually the mapping function for row order
  2. Ex: A1 A2 A3 A4 W1 W2 B1 B2 A5 A6 ...

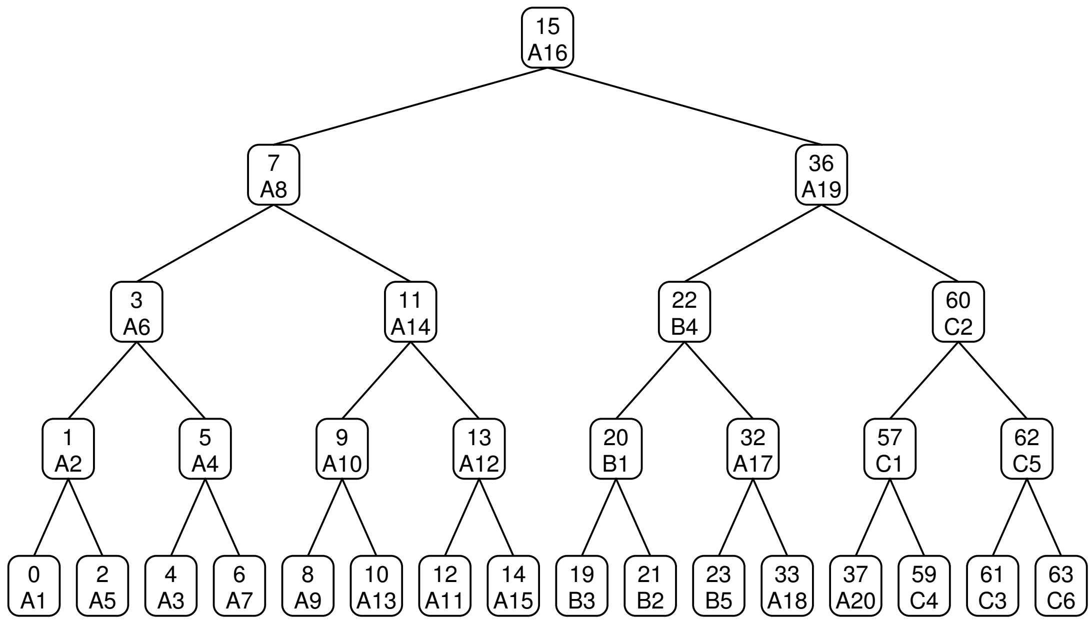
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0,0 A1 | 0,1 A2 | 0,2 A3 | 0,3 A4 | 0,4 W1 | 0,5 W2 | 0,6 B1 | 0,7 B2 |
| 1,0 A5 | 1,1 A6 | 1,2 A7 | 1,3 A8 | 1,4 W3 | 1,5 B3 | 1,6 B4 | 1,7 B5 |
| 2,0 A9 | 2,1 A10 | 2,2 A11 | 2,3 A12 | 2,4 W4 | 2,5 W5 | 2,6 W6 | 2,7 W7 |
| 3,0 A13 | 3,1 A14 | 3,2 A15 | 3,3 A16 | 3,4 W8 | 3,5 W9 | 3,6 W10 | 3,7 W11 |
| 4,0 A17 | 4,1 A18 | 4,2 A19 | 4,3 A20 | 4,4 W12 | 4,5 W13 | 4,6 W14 | 4,7 W15 |
| 5,0 W16 | 5,1 W17 | 5,2 W18 | 5,3 W19 | 5,4 W20 | 5,5 W21 | 5,6 W22 | 5,7 W23 |
| 6,0 W24 | 6,1 W25 | 6,2 W26 | 6,3 W27 | 6,4 W28 | 6,5 C1 | 6,6 C2 | 6,7 C3 |
| 7,0 W29 | 7,1 W30 | 7,2 W31 | 7,3 W32 | 7,4 W33 | 7,5 C4 | 7,6 C5 | 7,7 C6 |

- Random access data structure as can retrieve address associated with location in constant time
- Implicit representation as do not explicitly aggregate contiguous cells comprising objects

# Tree Access Structure

- Using array access structure is wasteful of storage as need an element for each cell regardless of whether the cell is associated with any of the objects
- Only keep track of nonempty cells
  1. use a multidimensional point data structure for nonempty cells (e.g., point quadtree, PR quadtree, etc.)
  2. order space (e.g., Morton order) and map into integers after which use a one-dimensional index such as a balanced binary tree

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 A1 | 1 A2 | 4 A3 | 5 A4 | 16 W1 | 17 W2 | 20 B1 | 21 B2 |
| 2 A5 | 3 A6 | 6 A7 | 7 A8 | 18 W3 | 19 B3 | 22 B4 | 23 B5 |
| 8 A9 | 9 A10 | 12 A11 | 13 A12 | 24 W4 | 25 W5 | 28 W6 | 29 W7 |
| 10 A13 | 11 A14 | 14 A15 | 15 A16 | 26 W8 | 27 W9 | 30 W10 | 31 W11 |
| 32 A17 | 33 A18 | 36 A19 | 37 A20 | 48 W12 | 49 W13 | 52 W14 | 53 W15 |
| 34 W16 | 35 W17 | 38 W18 | 39 W19 | 50 W20 | 51 W21 | 54 W22 | 55 W23 |
| 40 W24 | 41 W25 | 44 W26 | 45 W27 | 56 W28 | 57 C1 | 60 C2 | 61 C3 |
| 42 W29 | 43 W30 | 46 W31 | 47 W32 | 58 W33 | 59 C4 | 62 C5 | 63 C6 |

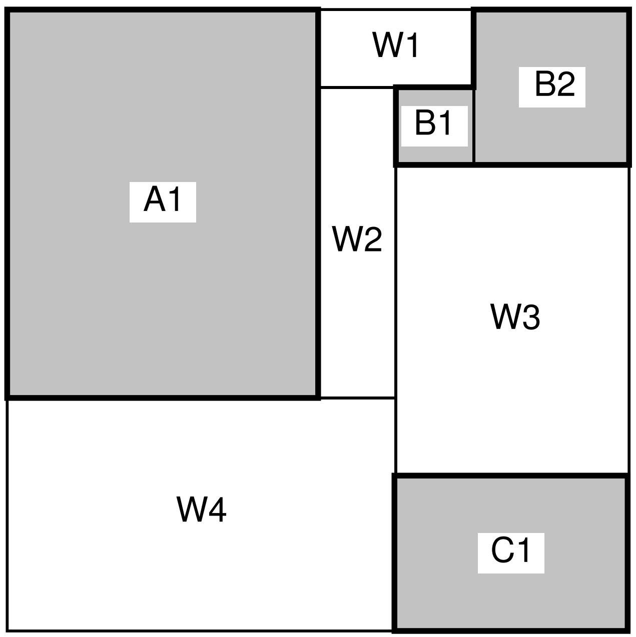


# Interior-based Representations: Blocks

- Decomposition is usually recursive and occurs in stages
- Often, results of stages form a containment hierarchy:
  1. stage $i$: decompose block $b$ into a set of blocks $b_j$ that span the same space
  2. stage $i+1$: decompose blocks $b_j$ using same decomposition rule
- Some decomposition rules restrict the possible sizes and shapes of the blocks as well as their placement in space
  1. all blocks at a particular stage are congruent
  2. similar blocks at all stages (e.g., rectangles and isoceles triangles)
  3. all but one side of a block are unit-sized (e.g, runlength encoding)
  4. all sides of a block are of equal size (e.g., squares and equilateral triangles)
  5. all sides of each block are powers of two
  6. etc.

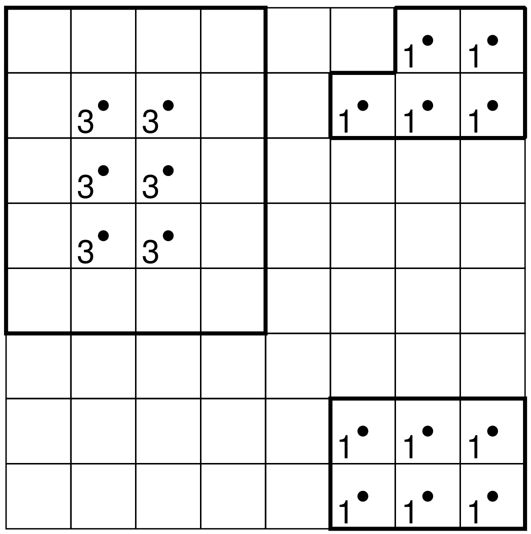
# Arbitrarily-sized Rectangular Blocks

- Non-uniform block size means that instead of associating a set with each object $o$ that contains the location in space of each cell that comprises $o$, we need to associate with each object $o$ the location in space and size of each block that comprises $o$
- Goal is to associate objects with minumum number of blocks
  - analogous to bin packing
  - NP-complete problem
- Ex: record coordinate values of upper-left corner of each block and sizes of their various sides

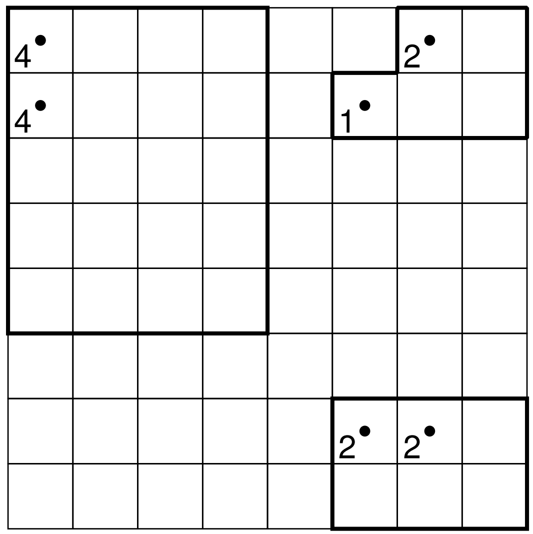


# Medial Axis Transformation (MAT)

- For each unit-size cell $c$ in object $o$, find largest square $s_c$ of width $w_c$ (i.e., radius $w_c/2$) centered at $c$ that is contained in $o$
- $s_c$ is a *maximal block* if it is not contained in the largest square $s_{c'}$ of any other cell $c'$ in $o$
- Each object is completely specified by the maximal blocks, their widths, and the unit-size cells at their centers — that is, the set of triples $(c, s_c, w_c)$ — termed a *medial axis transformation (MAT)*




MAT

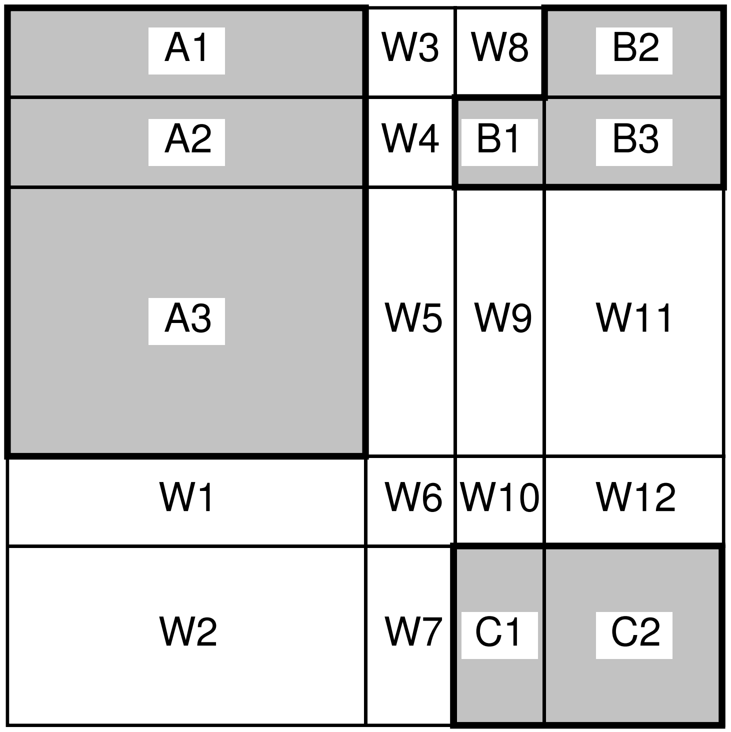



CMAT

- Can save space by anchoring MAT at a corner (CMAT)
- Rooted in skeletons where block width is omitted

# Irregular Grids

- Grid of irregular-sized blocks
- Linear scales aid in accessing column and row of entry array access structure corresponding to a block containing location $l$
- Used in implementation of grid file representation for point data




Irregular Grid

| | | | |
|---|---|---|---|
| A1 | W3 | W8 | B2 |
| A2 | W4 | B1 | B3 |
| A3 | W5 | W9 | W11 |
| W1 | W6 | W10 | W12 |
| W2 | W7 | C1 | C2 |

Grid Directory

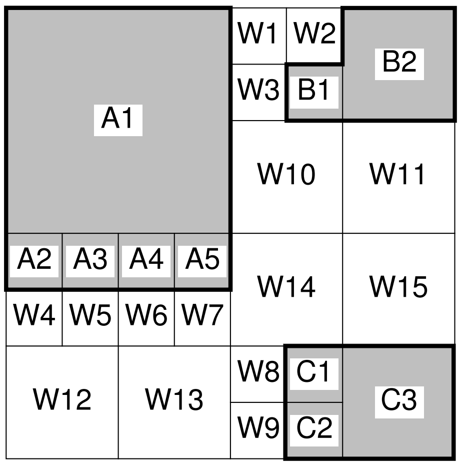
| array column | x range |
|---|---|
| 1 | [0,4) |
| 2 | [4,5) |
| 3 | [5,6) |
| 4 | [6,8) |

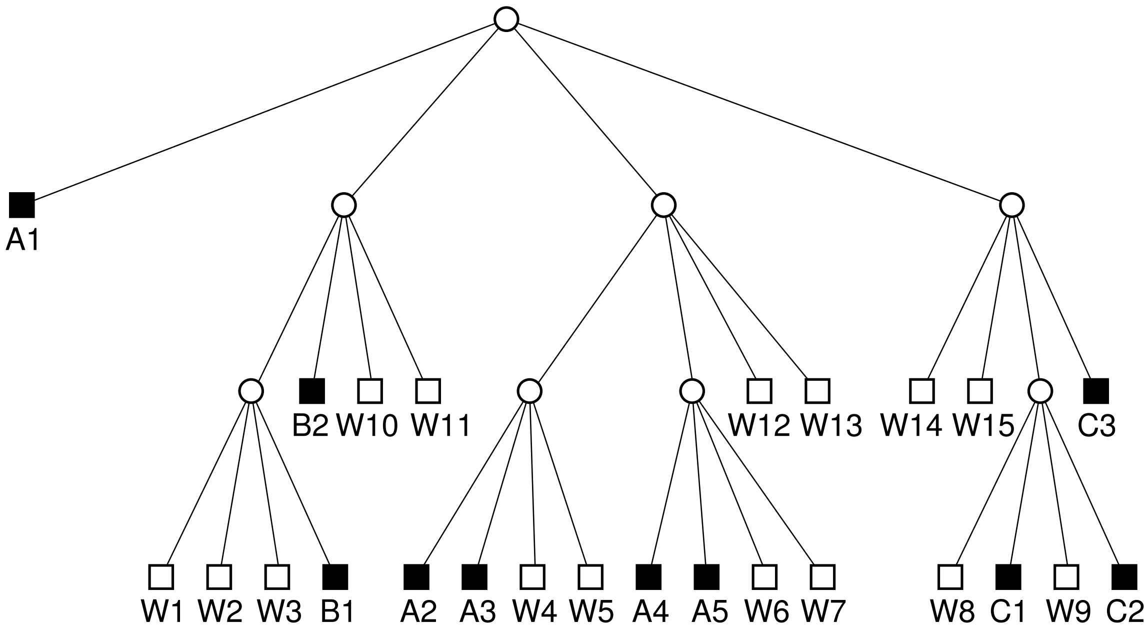
| array row | y range |
|---|---|
| 1 | [0,1) |
| 2 | [1,2) |
| 3 | [2,5) |
| 4 | [5,6) |
| 5 | [6,8) |

Linear Scales

# Region Quadtree

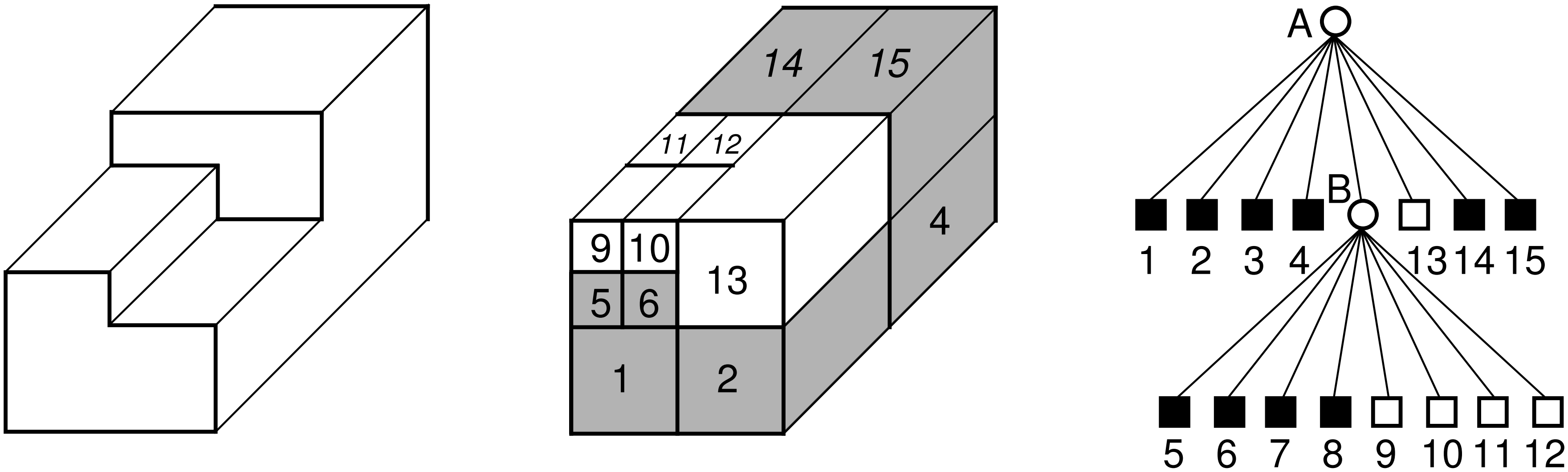
- Recursively decompose into rectangular congruent blocks at each level
  1. all blocks have sides that are powers of two
  2. restricted positions
     - block $b$ of size $2^s \times 2^s$ has upper left corner $(i, j)$ such that $a \bmod 2^s = 0$ and $b \bmod 2^s = 0$
- Key idea is decomposition of space into blocks
- Tree access structure for logarithmic retrieval of blocks but other access structures are possible (e.g., B$^+$-tree for point data)



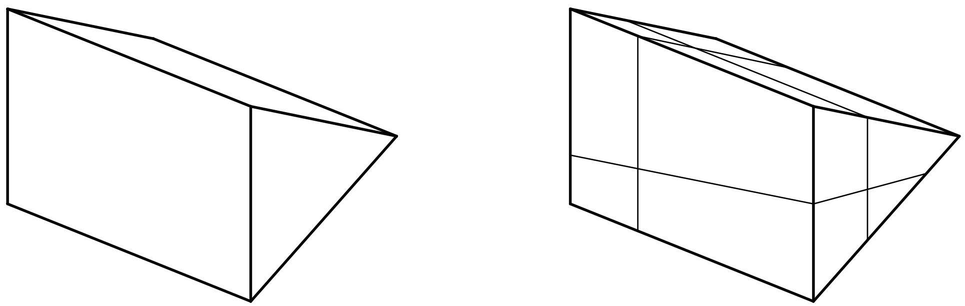



# Octree

- Three-dimensional region octree

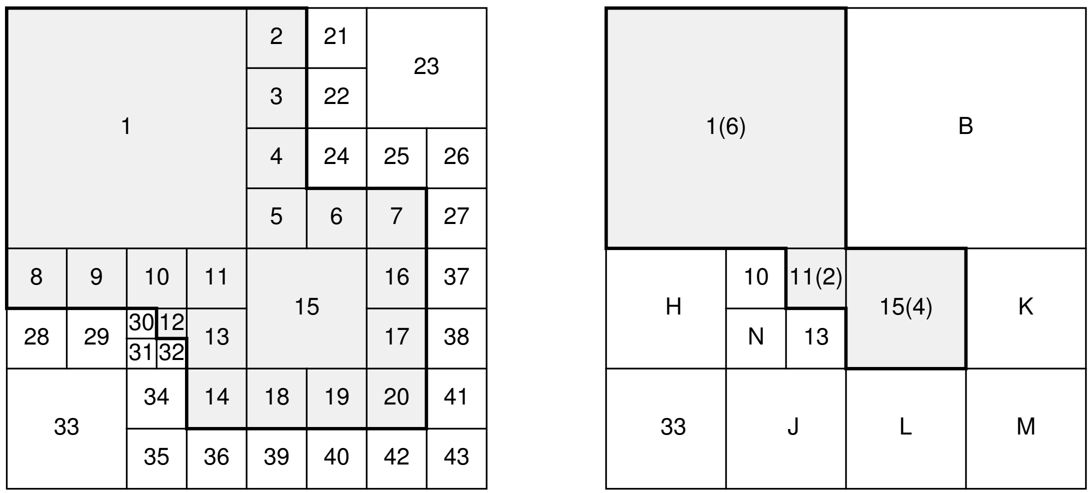



- PM octree
  1. for polyhedra
  2. Split if more than one face in a block unless all faces are adjacent to the same edge or are incident at the same vertex



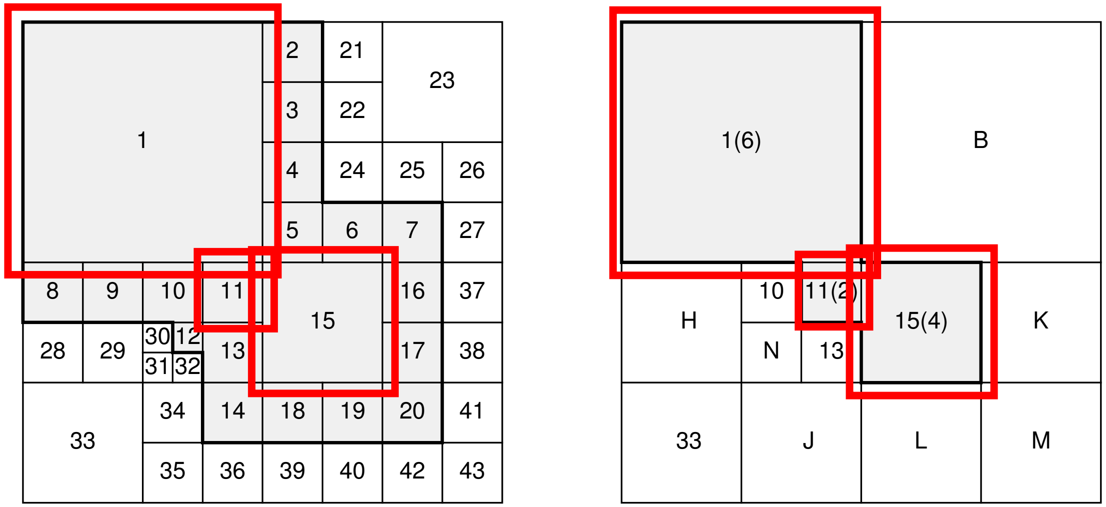
# Quadtree Medial Axis Transform (QMAT)

- Region quadtree is a special case of medial axis transformation (MAT)
  1. widths of maximal blocks of MAT are restricted to be powers of 2
  2. positions of their centers are constrained to be centers of blocks obtained by a recursive decomposition of the underlying space into four congruent disjoint blocks
- Adapt MAT to region quadtree by only constraining positions of centers of blocks but not their sizes and result in quadtree medial axis transform (QMAT)
  1. quadtree skeleton: blocks comprising QMAT and is unique
  2. maximal blocks in QMAT are not disjoint



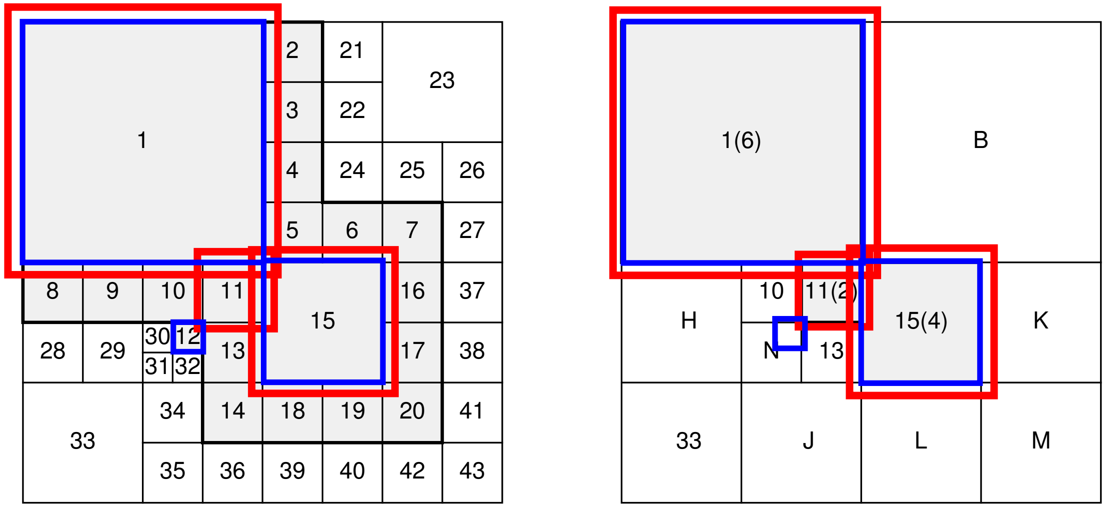
# Quadtree Medial Axis Transform (QMAT)

- Region quadtree is a special case of medial axis transformation (MAT)
  1. widths of maximal blocks of MAT are restricted to be powers of 2
  2. positions of their centers are constrained to be centers of blocks obtained by a recursive decomposition of the underlying space into four congruent disjoint blocks
- Adapt MAT to region quadtree by only constraining positions of centers of blocks but not their sizes and result in quadtree medial axis transform (QMAT)
  1. quadtree skeleton: blocks comprising QMAT and is unique
  2. maximal blocks in QMAT are not disjoint
  3. Ex: QMAT consists of blocks 1, 11, and 15



# Quadtree Medial Axis Transform (QMAT)

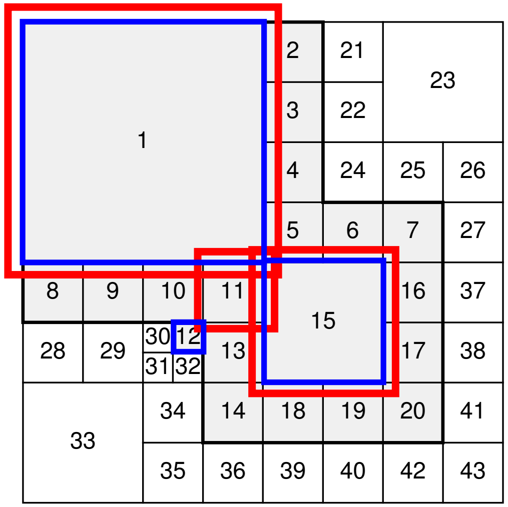
- Region quadtree is a special case of medial axis transformation (MAT)
  1. widths of maximal blocks of MAT are restricted to be powers of 2
  2. positions of their centers are constrained to be centers of blocks obtained by a recursive decomposition of the underlying space into four congruent disjoint blocks
- Adapt MAT to region quadtree by only constraining positions of centers of blocks but not their sizes and result in quadtree medial axis transform (QMAT)
  1. quadtree skeleton: blocks comprising QMAT and is unique
  2. maximal blocks in QMAT are not disjoint
  3. Ex: QMAT consists of blocks 1, 11, and 15

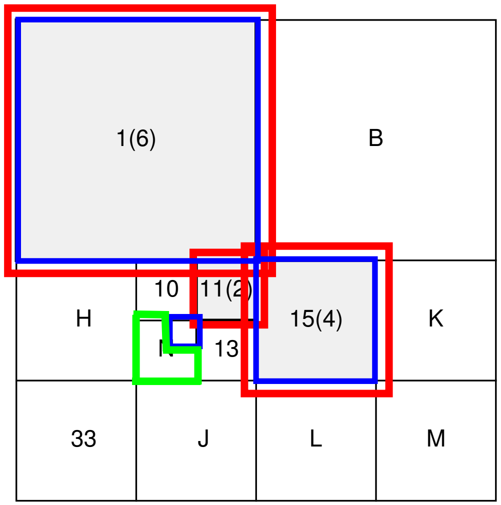



- Maximal block requirement precludes alternative QMAT (blocks 1,12,15)

# Quadtree Medial Axis Transform (QMAT)

- Region quadtree is a special case of medial axis transformation (MAT)
  1. widths of maximal blocks of MAT are restricted to be powers of 2
  2. positions of their centers are constrained to be centers of blocks obtained by a recursive decomposition of the underlying space into four congruent disjoint blocks
- Adapt MAT to region quadtree by only constraining positions of centers of blocks but not their sizes and result in quadtree medial axis transform (QMAT)
  1. quadtree skeleton: blocks comprising QMAT and is unique
  2. maximal blocks in QMAT are not disjoint
  3. Ex: QMAT consists of blocks 1, 11, and 15

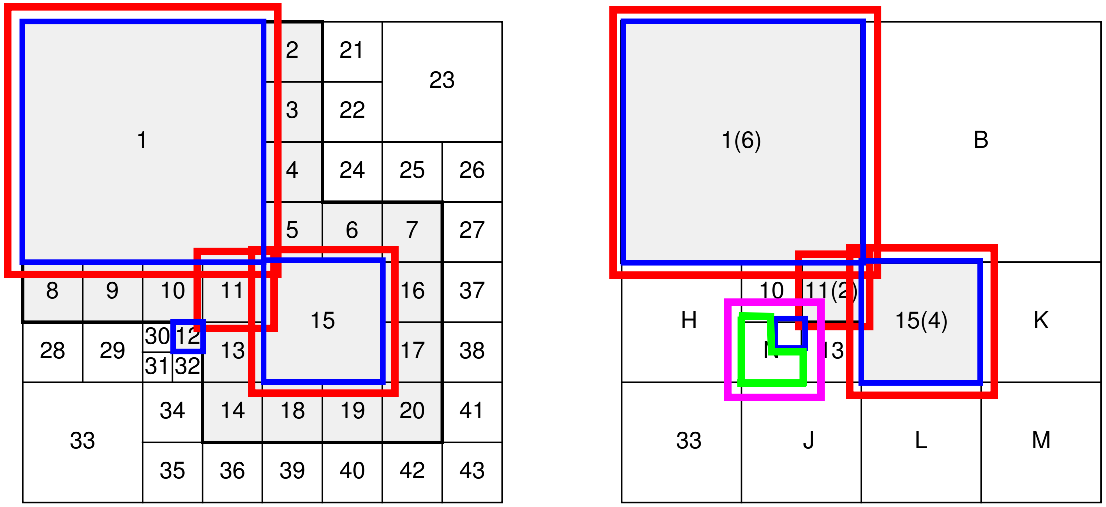



- Maximal block requirement precludes alternative QMAT (blocks 1,12,15)
- alternative has more empty blocks in QMAT as 12 is deeper

# Quadtree Medial Axis Transform (QMAT)

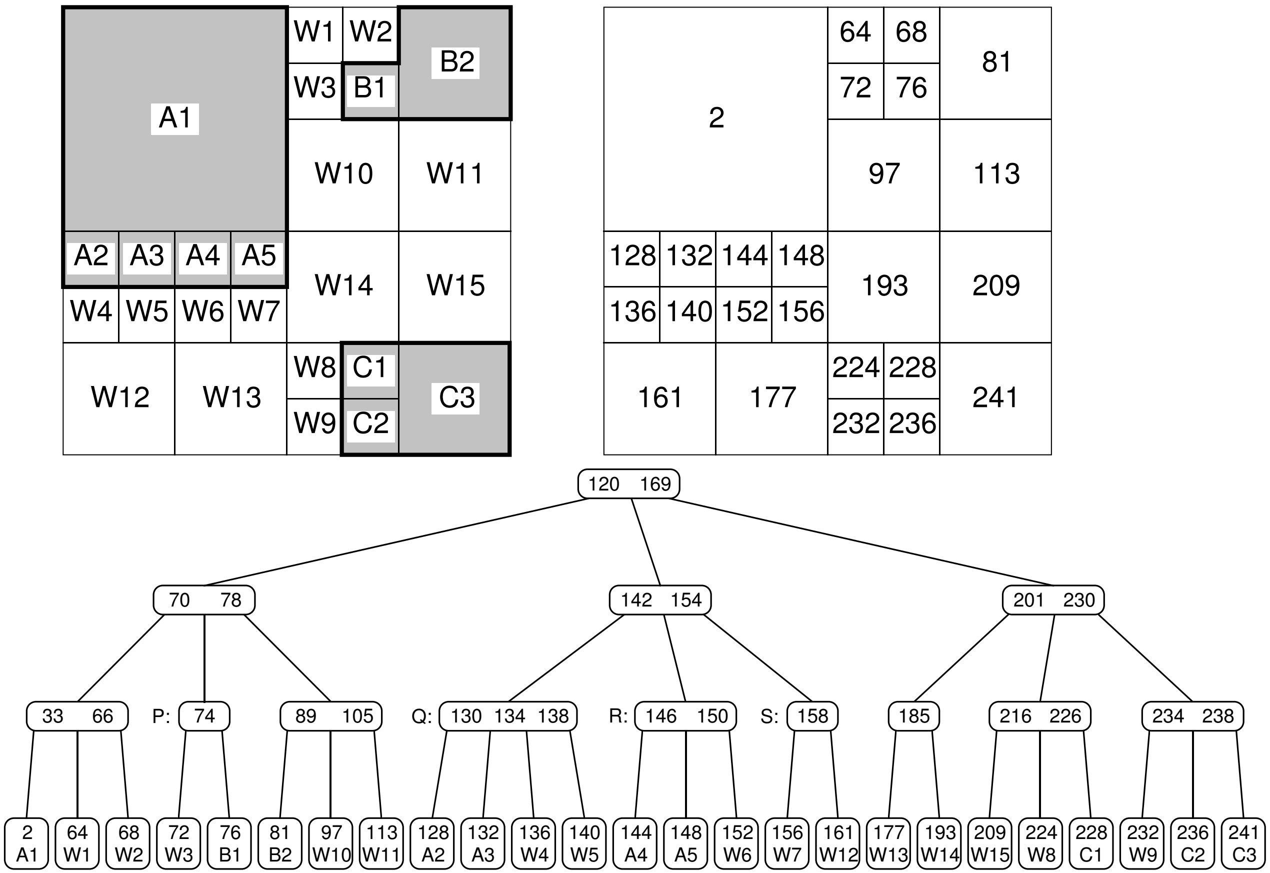
- Region quadtree is a special case of medial axis transformation (MAT)
  1. widths of maximal blocks of MAT are restricted to be powers of 2
  2. positions of their centers are constrained to be centers of blocks obtained by a recursive decomposition of the underlying space into four congruent disjoint blocks
- Adapt MAT to region quadtree by only constraining positions of centers of blocks but not their sizes and result in quadtree medial axis transform (QMAT)
  1. quadtree skeleton: blocks comprising QMAT and is unique
  2. maximal blocks in QMAT are not disjoint
  3. Ex: QMAT consists of blocks 1, 11, and 15




- Maximal block requirement precludes alternative QMAT (blocks 1,12,15)
- alternative has more empty blocks in QMAT as 12 is deeper than 11

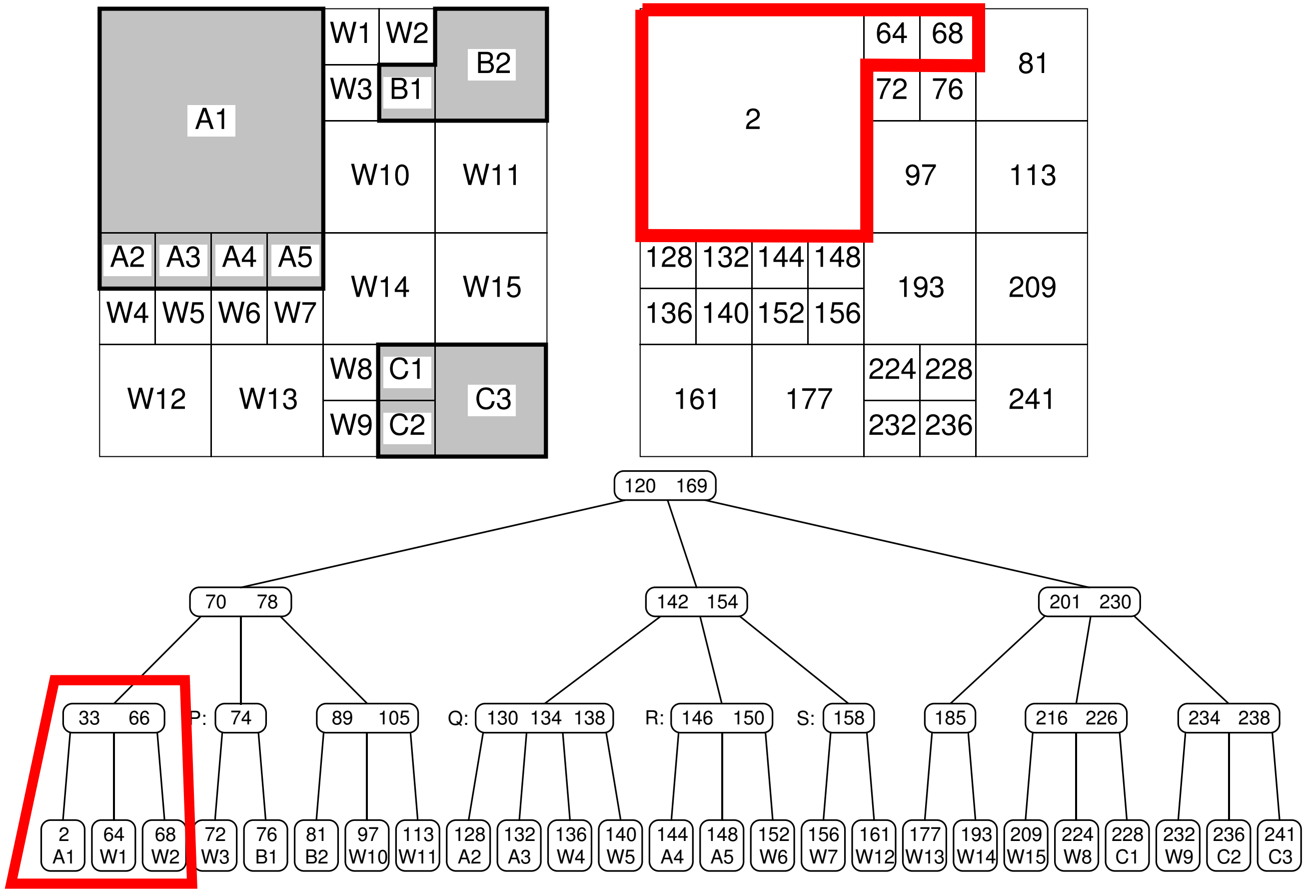
# Pointerless Quadtree Representation ($B^+$-tree)

- Locational code: bit interleaving (Morton order) with $y$ more significant than $x$ (2 times 3 bits) on left, and size ranging from 0 to 3 (2 bits) on right



# Pointerless Quadtree Representation (B$^+$-tree)

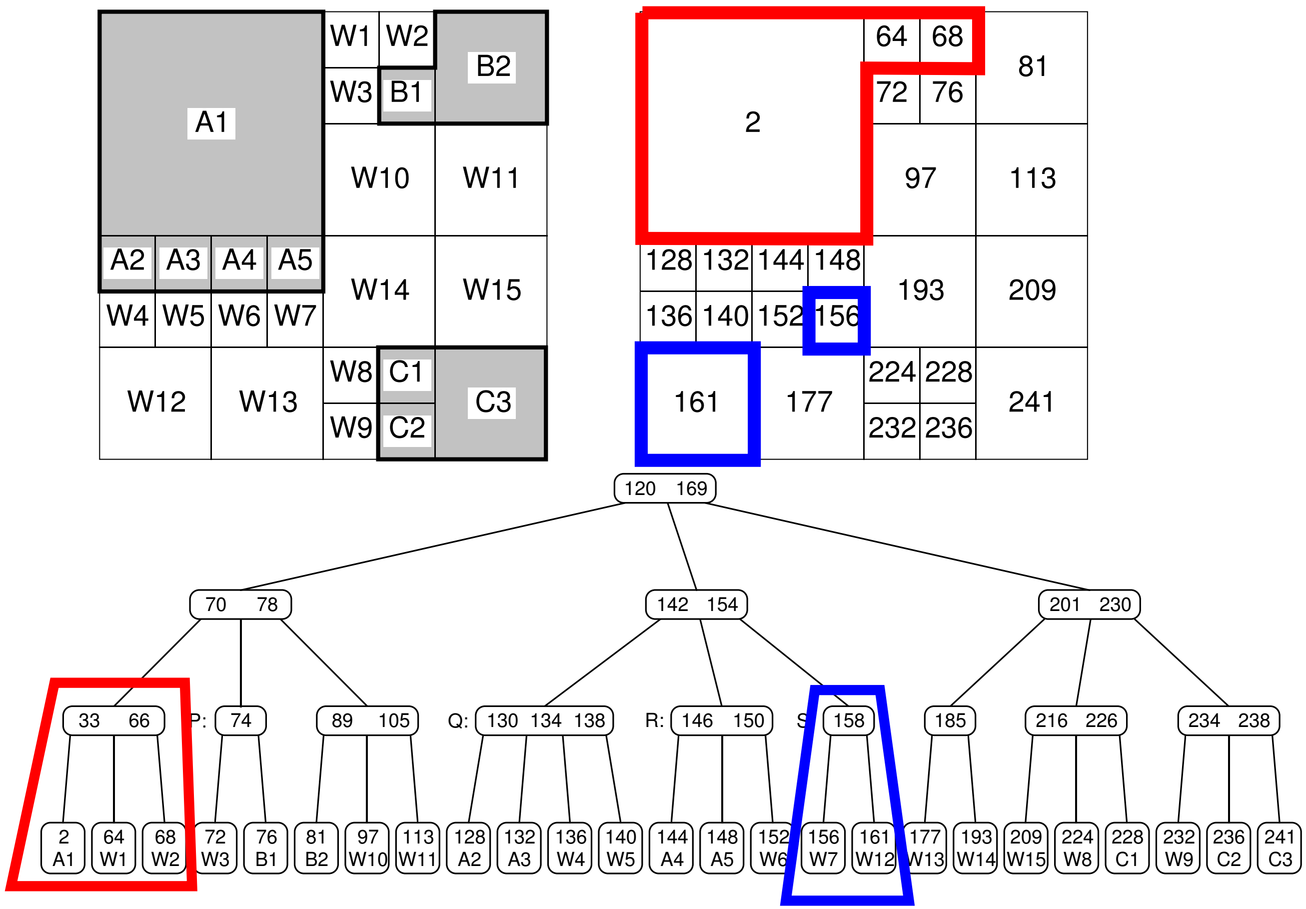
- Locational code: bit interleaving (Morton order) with $y$ more significant than $x$ (2 times 3 bits) on left, and size ranging from 0 to 3 (2 bits) on right




- Drawback: blocks corresponding to nonleaf nodes are not necessarily square

# Pointerless Quadtree Representation ($B^{+}$-tree)

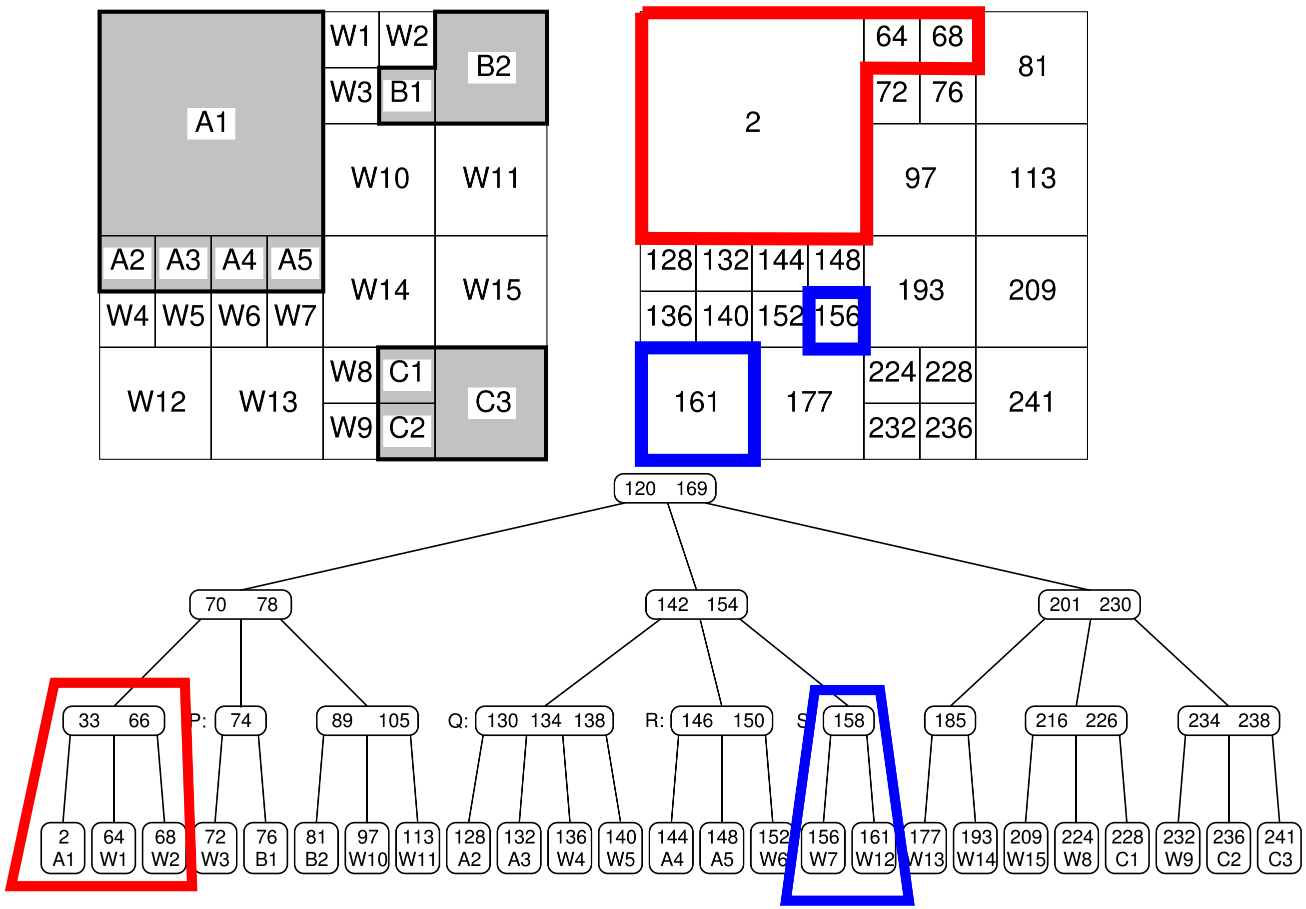
- Locational code: bit interleaving (Morton order) with $y$ more significant than $x$ (2 times 3 bits) on left, and size ranging from 0 to 3 (2 bits) on right




- Drawback: blocks corresponding to nonleaf nodes are not necessarily square nor contiguous

# Pointerless Quadtree Representation ($B^+$-tree)

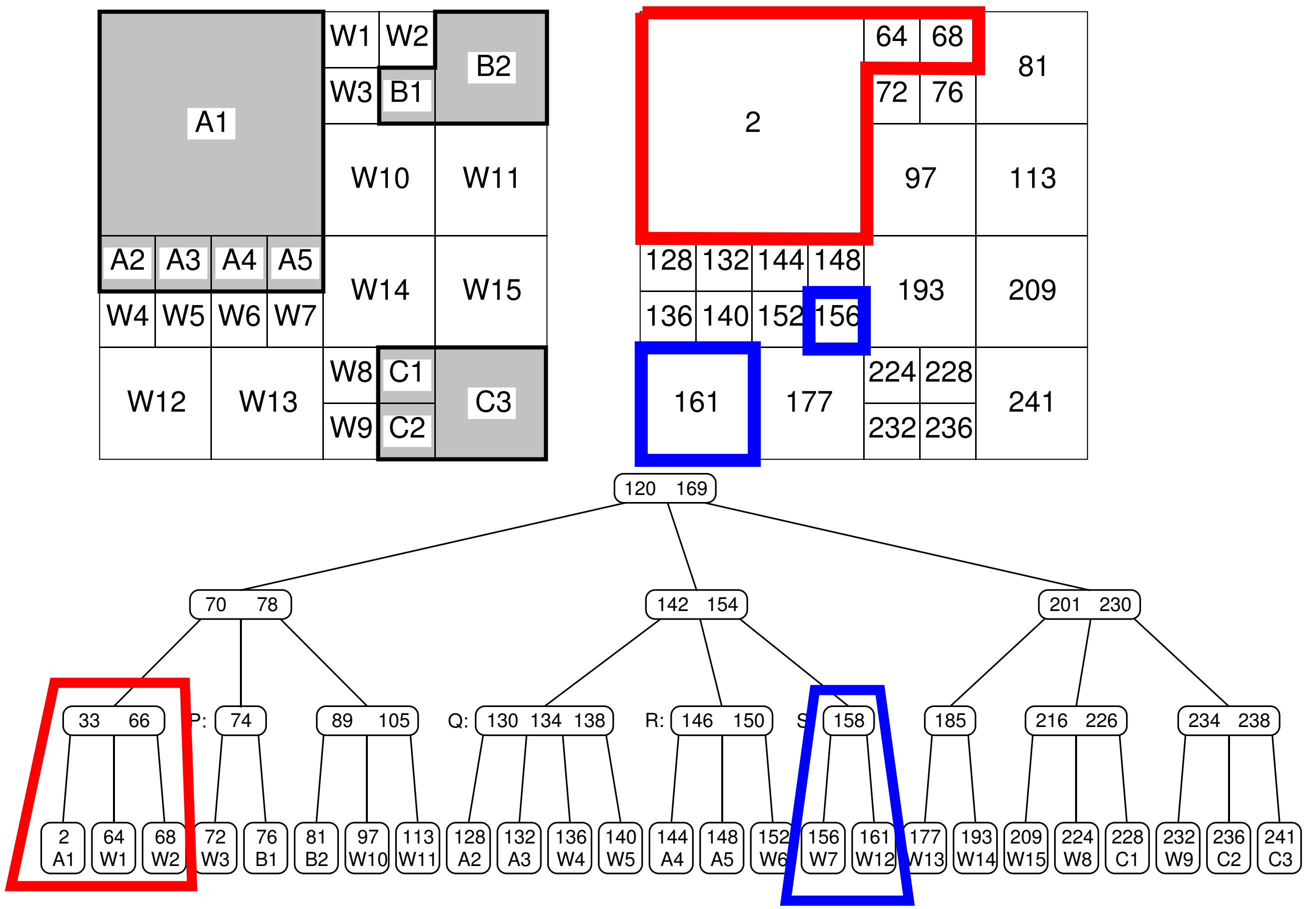
- Locational code: bit interleaving (Morton order) with $y$ more significant than $x$ (2 times 3 bits) on left, and size ranging from 0 to 3 (2 bits) on right




- Drawback: blocks corresponding to nonleaf nodes are not necessarily square nor contiguous (but see PK-tree where the price is the loss of balance)

# Pointerless Quadtree Representation (B$^+$-tree)

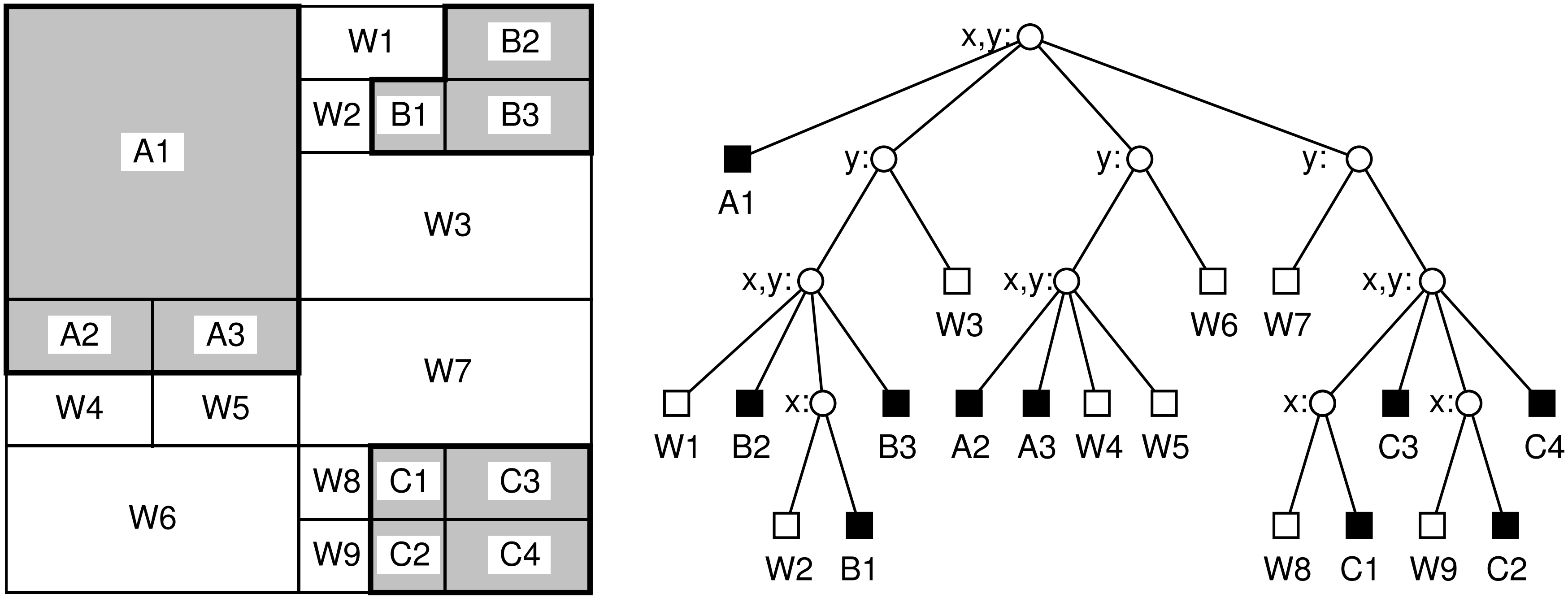
- Locational code: bit interleaving (Morton order) with $y$ more significant than $x$ (2 times 3 bits) on left, and size ranging from 0 to 3 (2 bits) on right




- Drawback: blocks corresponding to nonleaf nodes are not necessarily square nor contiguous (but see PK-tree where the price is the loss of balance)
- Can also store locational codes of BLACK leaf nodes and infer WHITE nodes

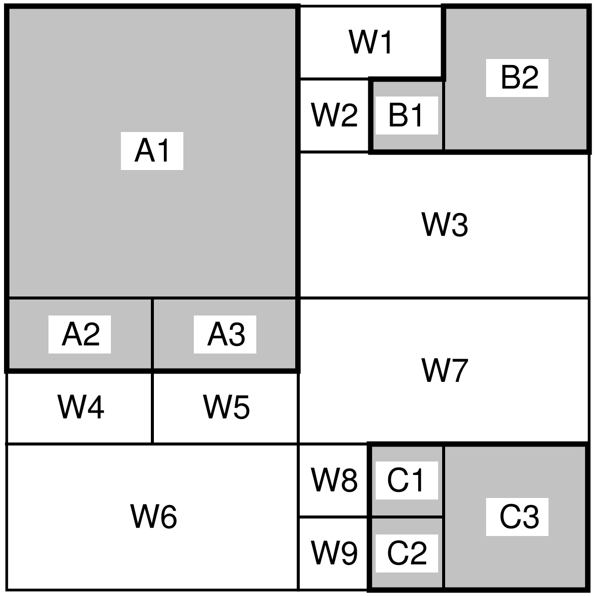
# ATree

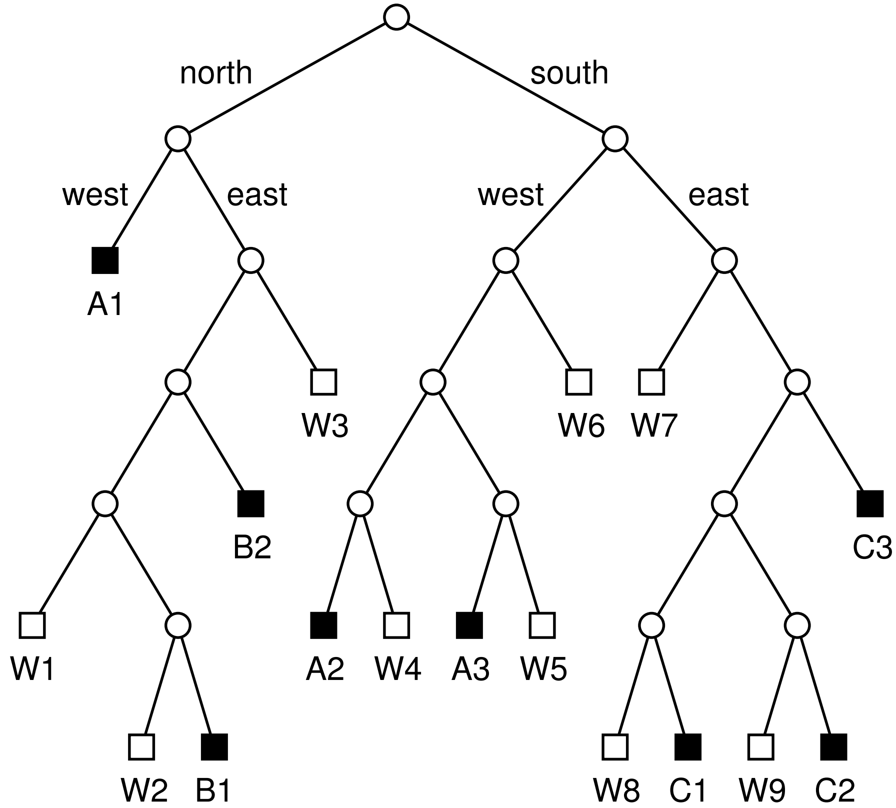
- Loosen requirement that $2^d$ congruent child blocks for each node at each level
- Replace by partitioning all blocks at the same subdivision stage (i.e., depth) $i$ into $2^{c_i}$ $(1 \leq c_i \leq d)$ congruent blocks
- Need to record identity of partition axes at each nonleaf node



# Bintree

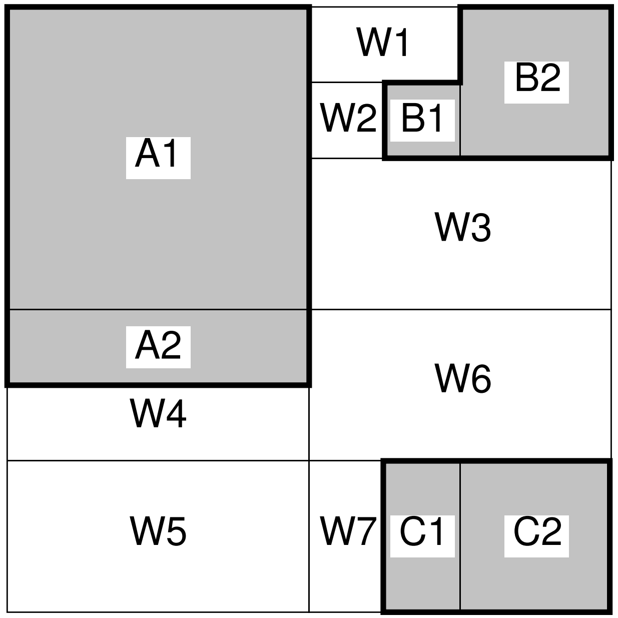
- Regular decomposition k-d tree
- Cycle through attributes

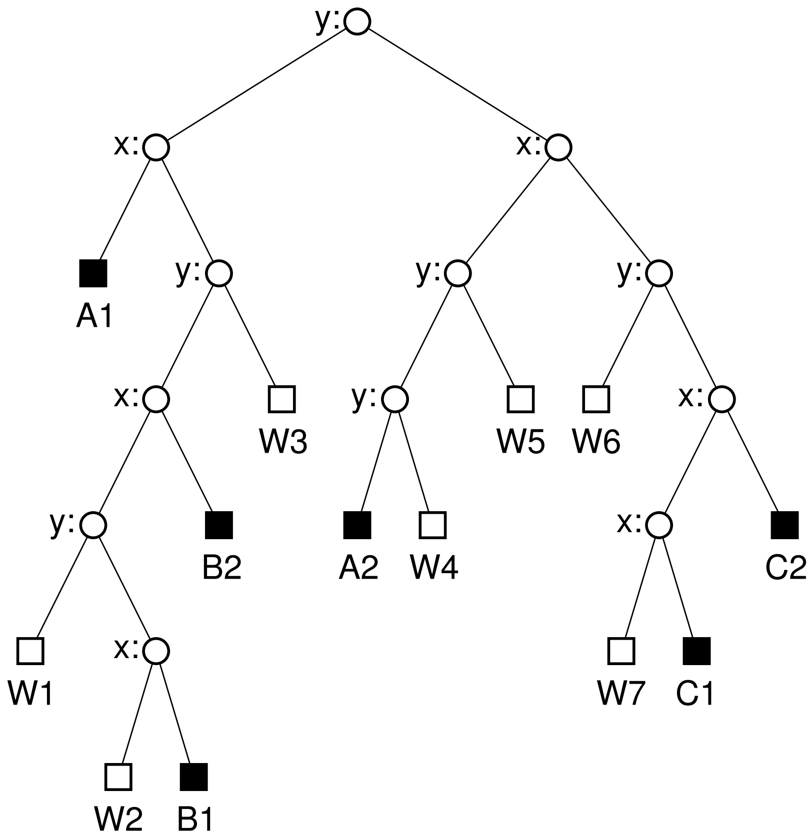


# Generalized Bintree

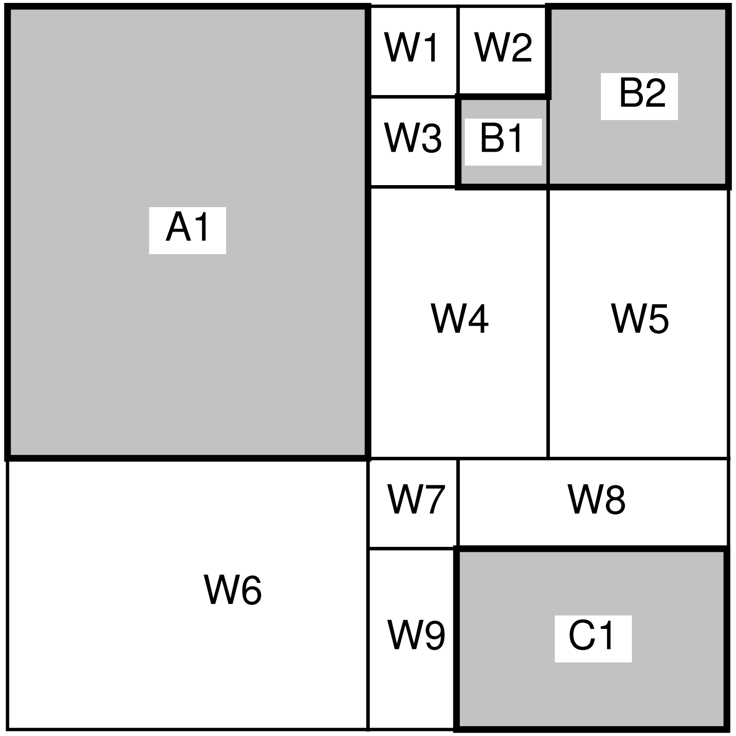
- Regular decomposition k-d tree but no need to cycle through attributes
- Need to record identity of partition axis at each nonleaf node

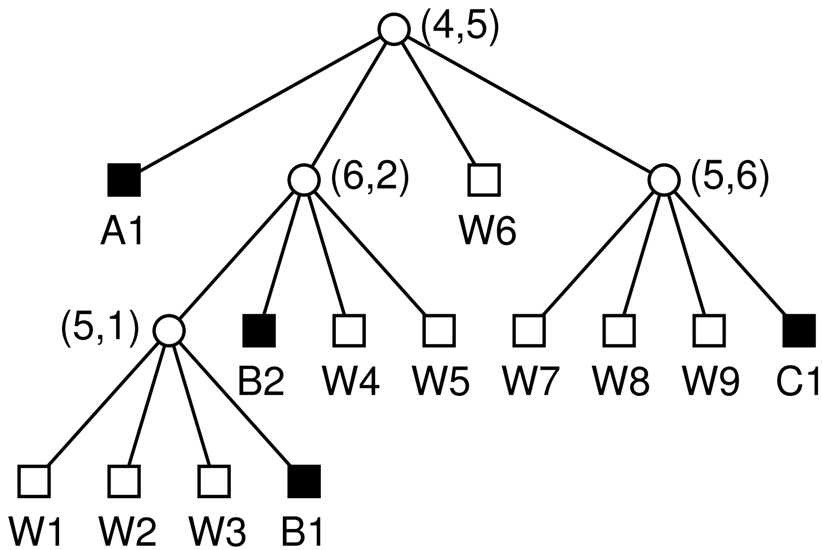


# Adaptation of Point Quadtree for Regions

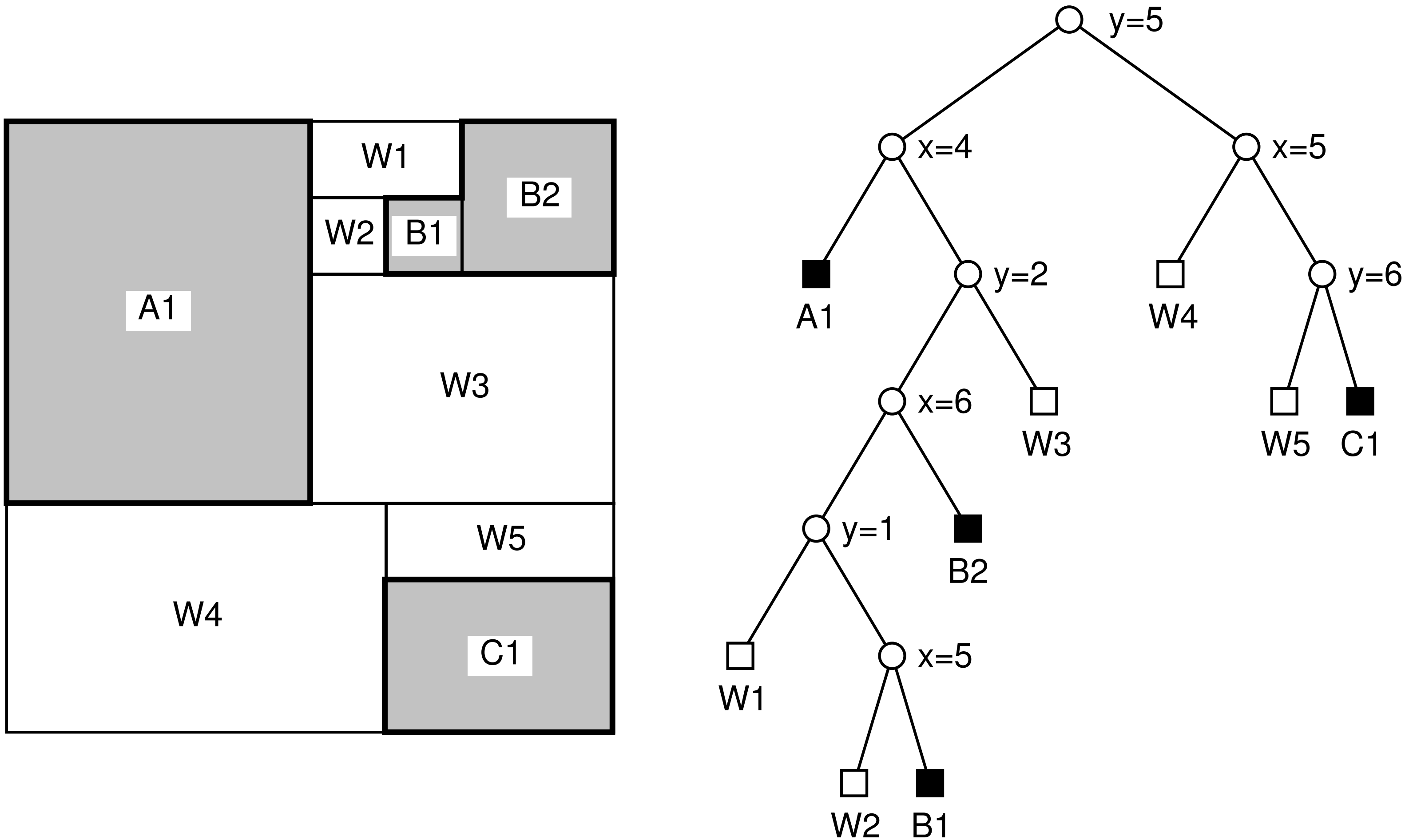
- No need for regular decomposition
- Need to record partition point at each nonleaf node



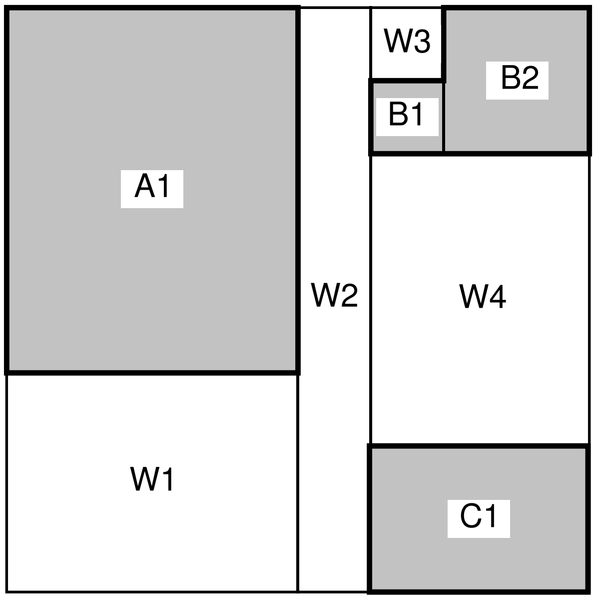
# Adaptation of K-d Tree for Regions

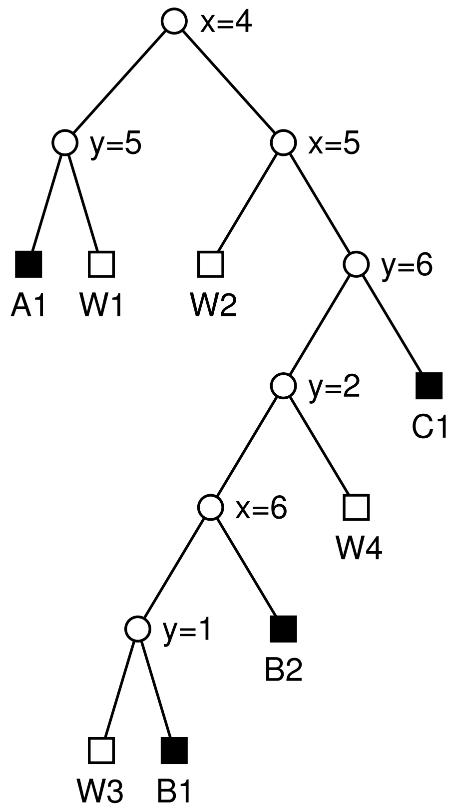
- No need for regular decomposition
- Need to record partition point at each nonleaf node



# Adaptation of Generalized K-d Tree for Regions

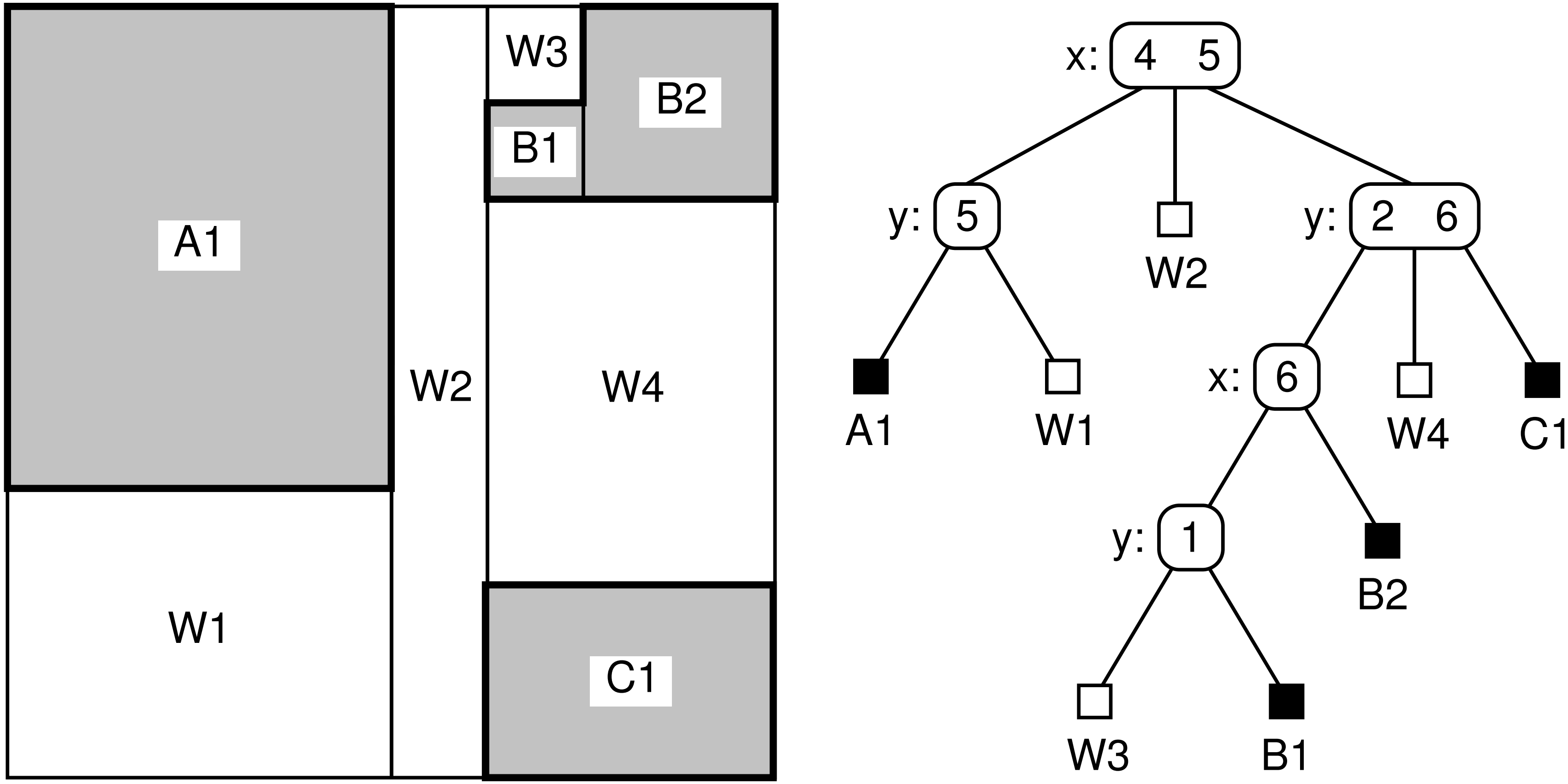
- No need for regular decomposition
- No need to cycle through attributes
- Need to record partition point and axis at each nonleaf node



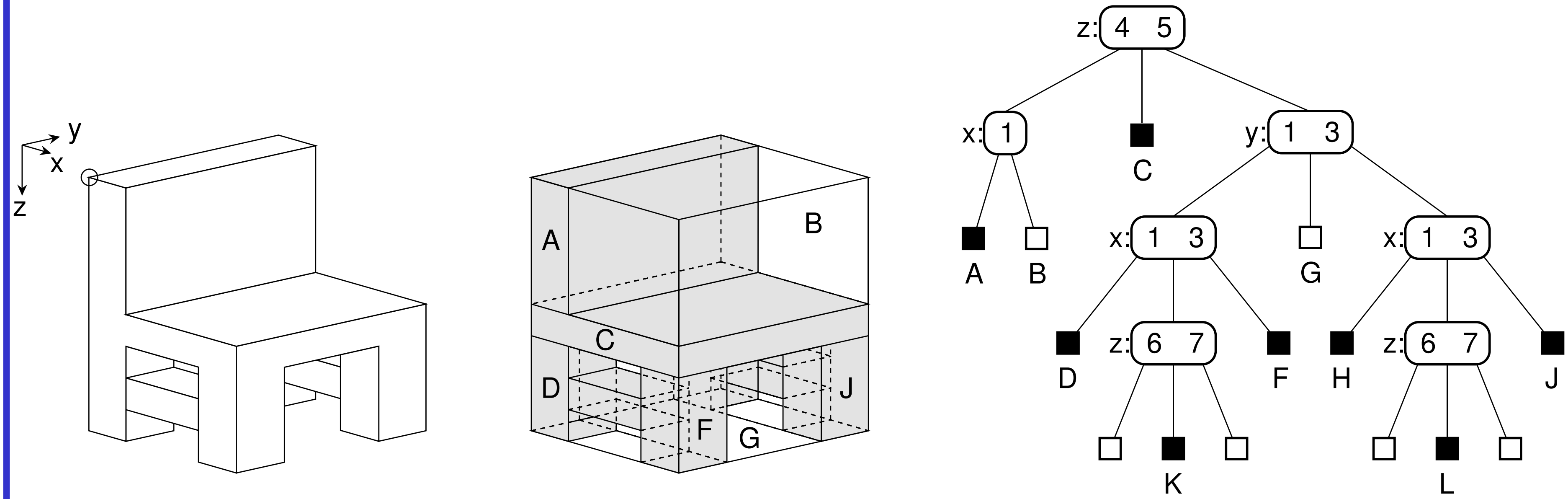
# X-Y Tree and Puzzle Tree

- Split into two or more parts at each partition step
- Implies no two successive partitions along the same attribute as they are combined
- Implies cycle through attributes in two dimensions



# Three-Dimensional X-Y Tree and Puzzle Tree

- No longer require cycling through dimensions as this results in losing some perceptually appealing block combinations



# Comparison of Bintree with X-Y Tree and Puzzle Tree

- Much more decomposition in bintree