Data Structures:
Spatial Trees: K-D Tree, Tries

YoungWoon Cha
(Slide credits to Won Kim)
Spring 2022

K-Dimensional Trees

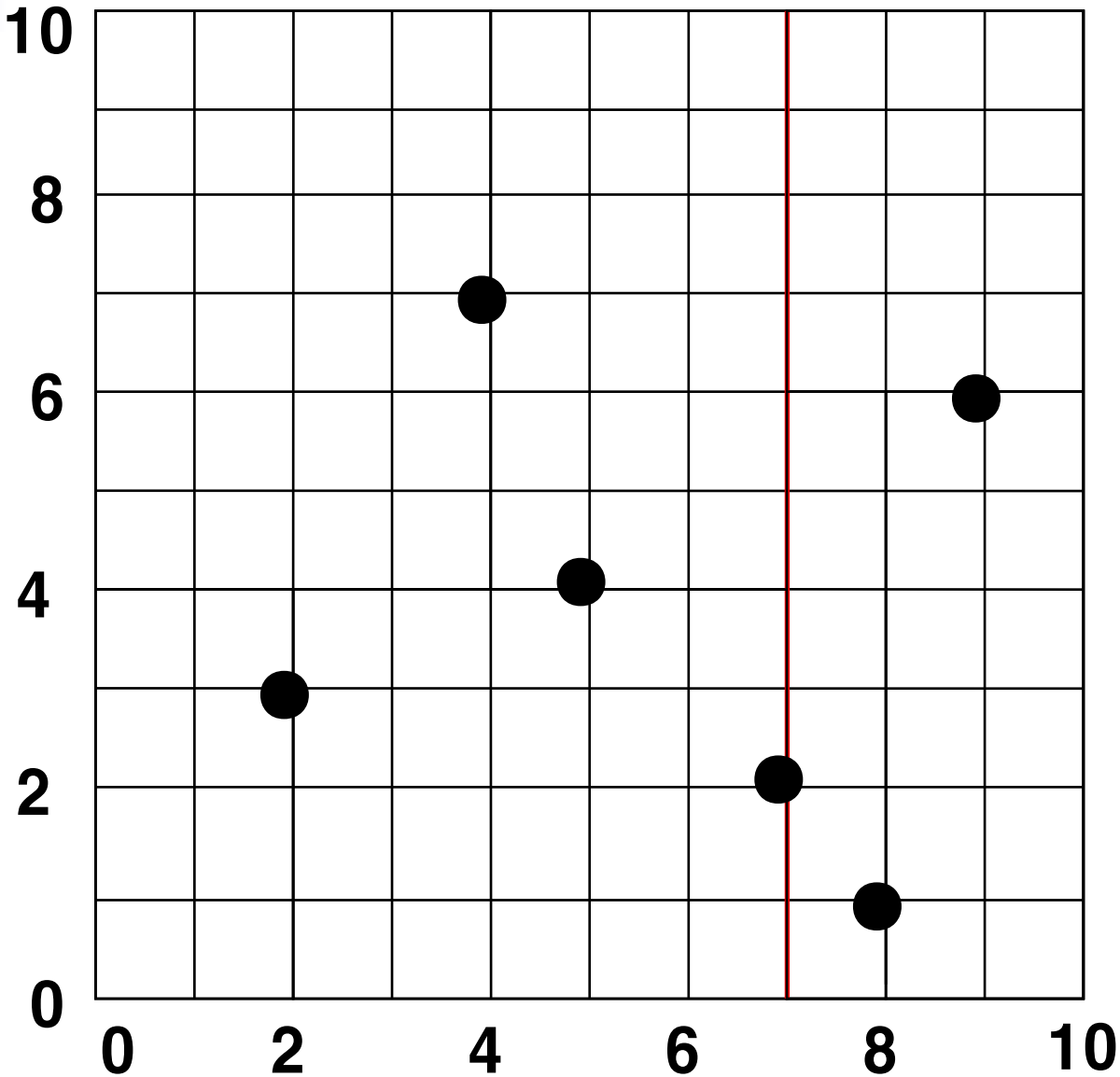
k-Dimensional Space

- 2-d
- 3-d
- 4-d
  - 3-d plus time
  - Person's (gender, age, education, ethnic origin)
- 5-d, 6-d, 7-d,…
  - Person's (gender, age, education, ethnic origin, religion, marital status, political party,…)

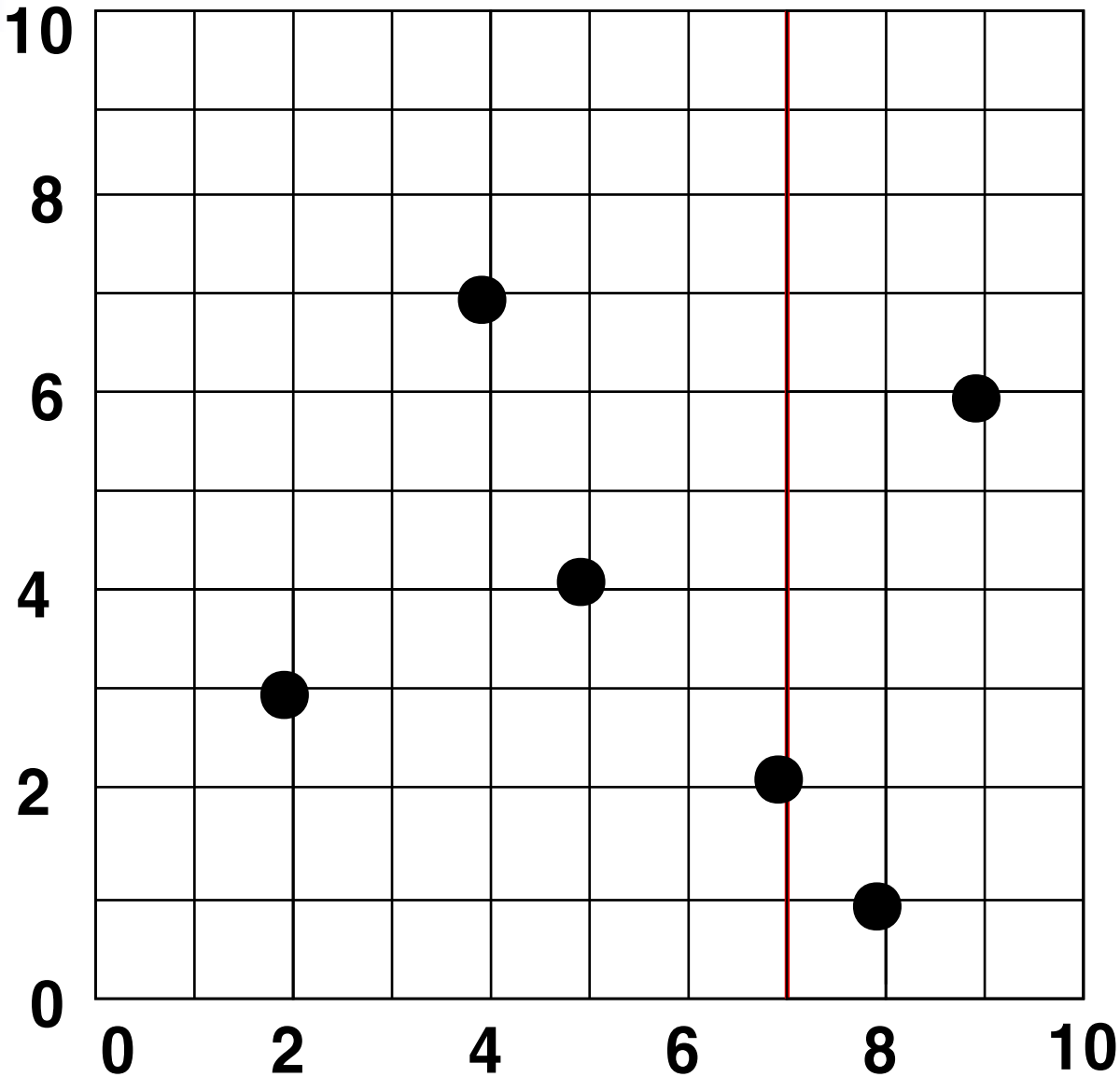
k-d Tree

- Store and search points in a k-dimensional space
  - k >= 2
- Point data (d1, d2, d3, d4,….)
  - ex. (5, 4), (70, 55, 37)

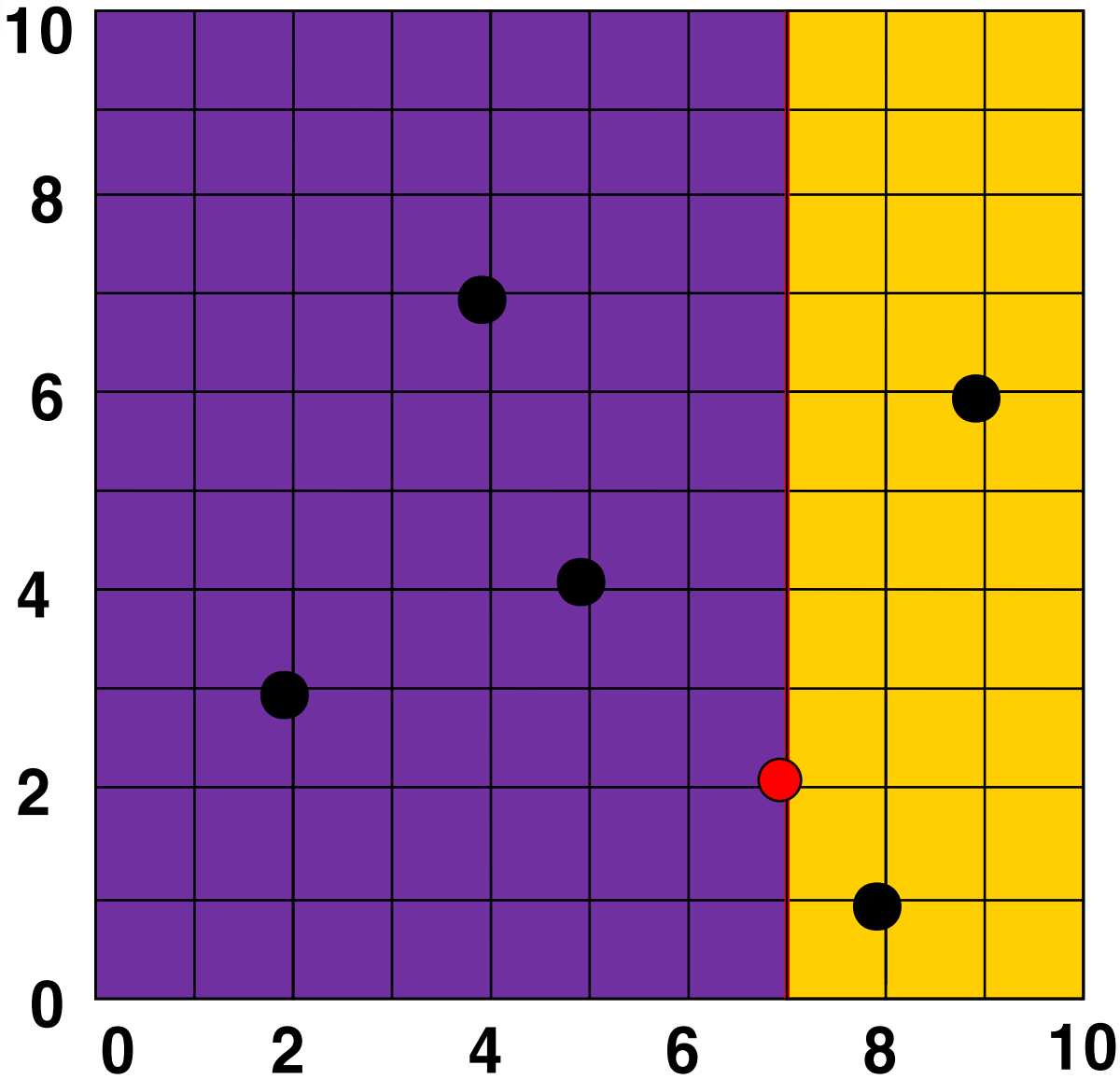
Successively Partition (Divide) a k-D Space Until No Partition Contains a Point



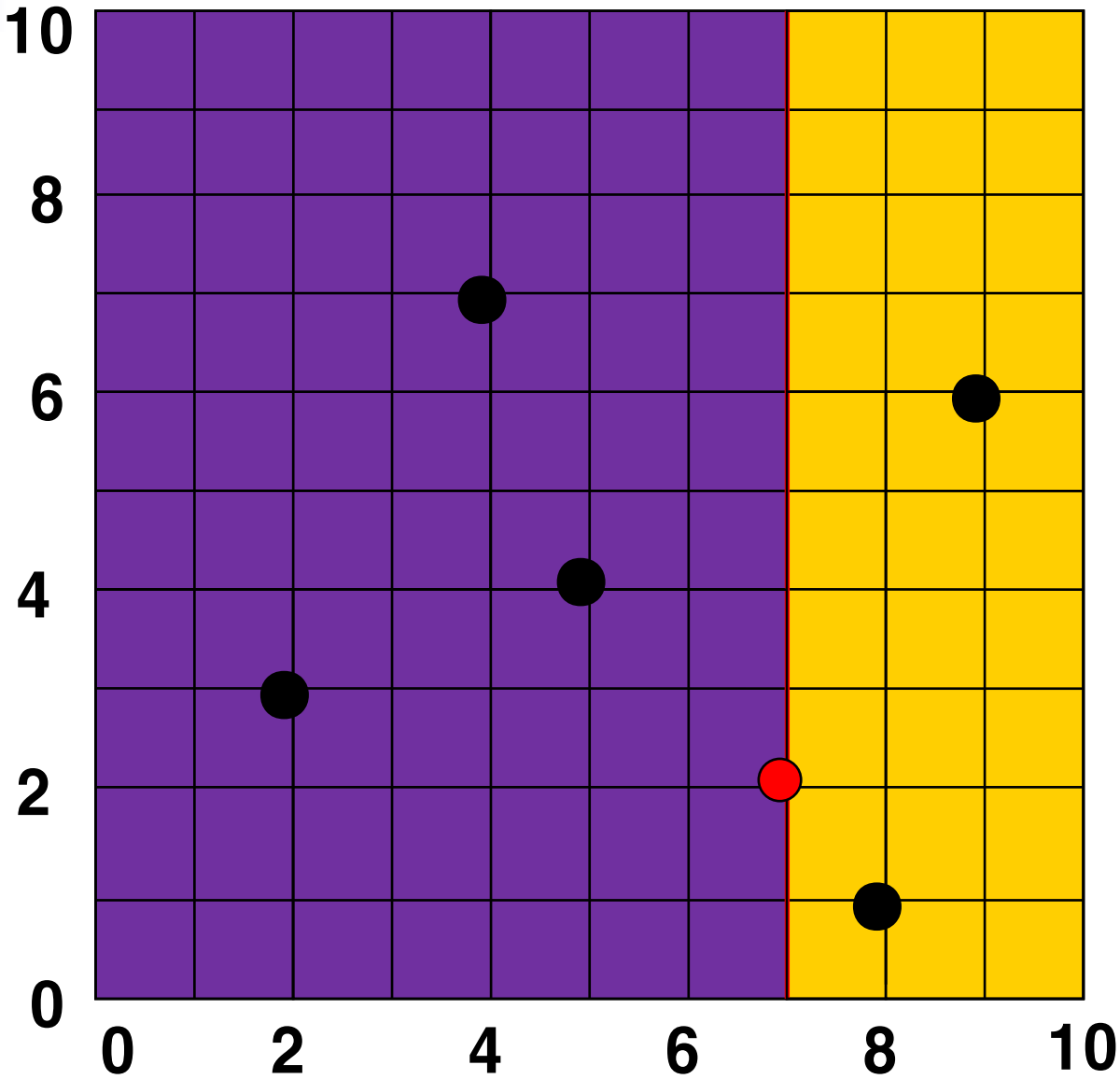
First, Select Point (7,2) and Partition the k-d Space along the Y Axis



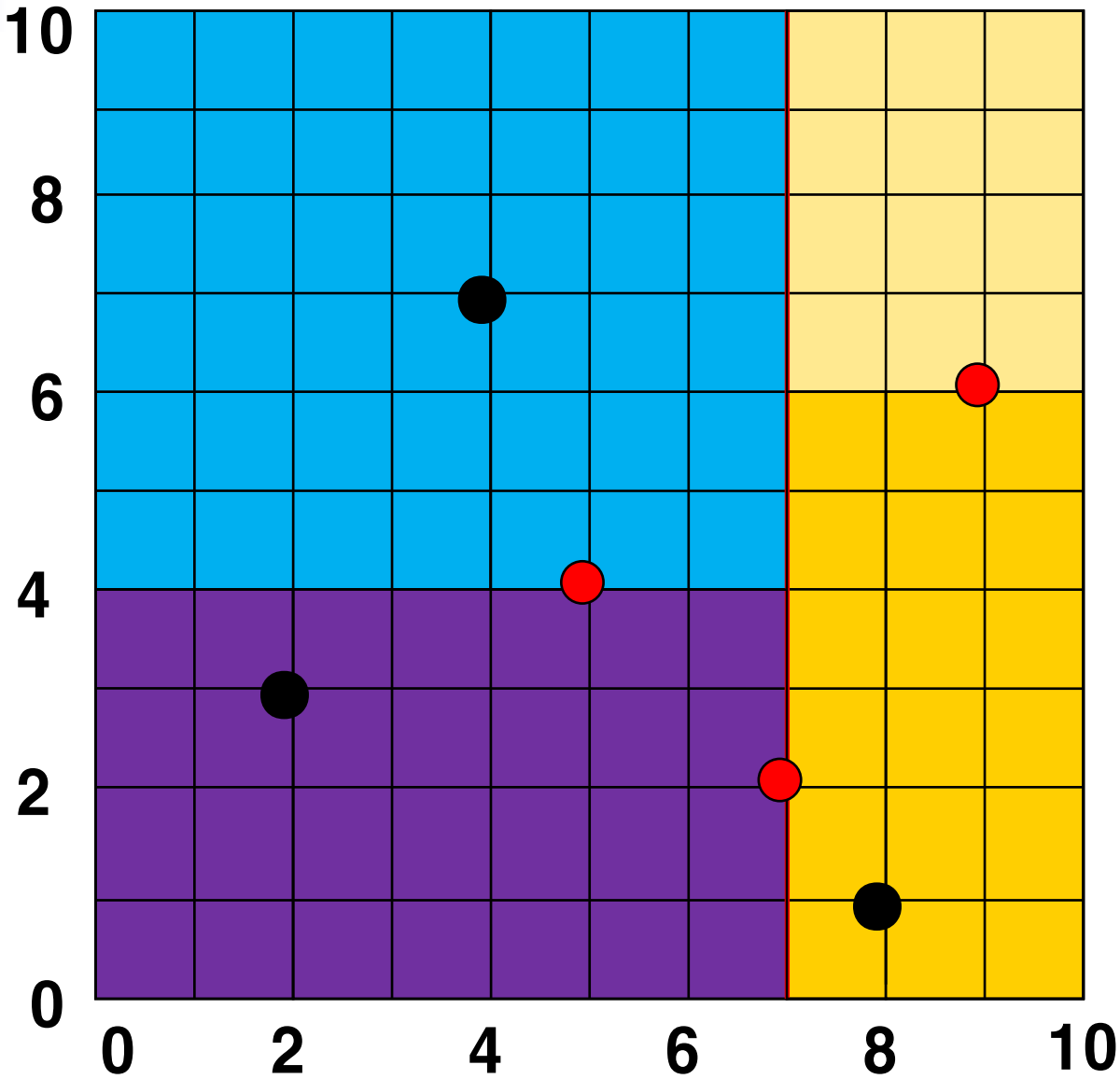
Result



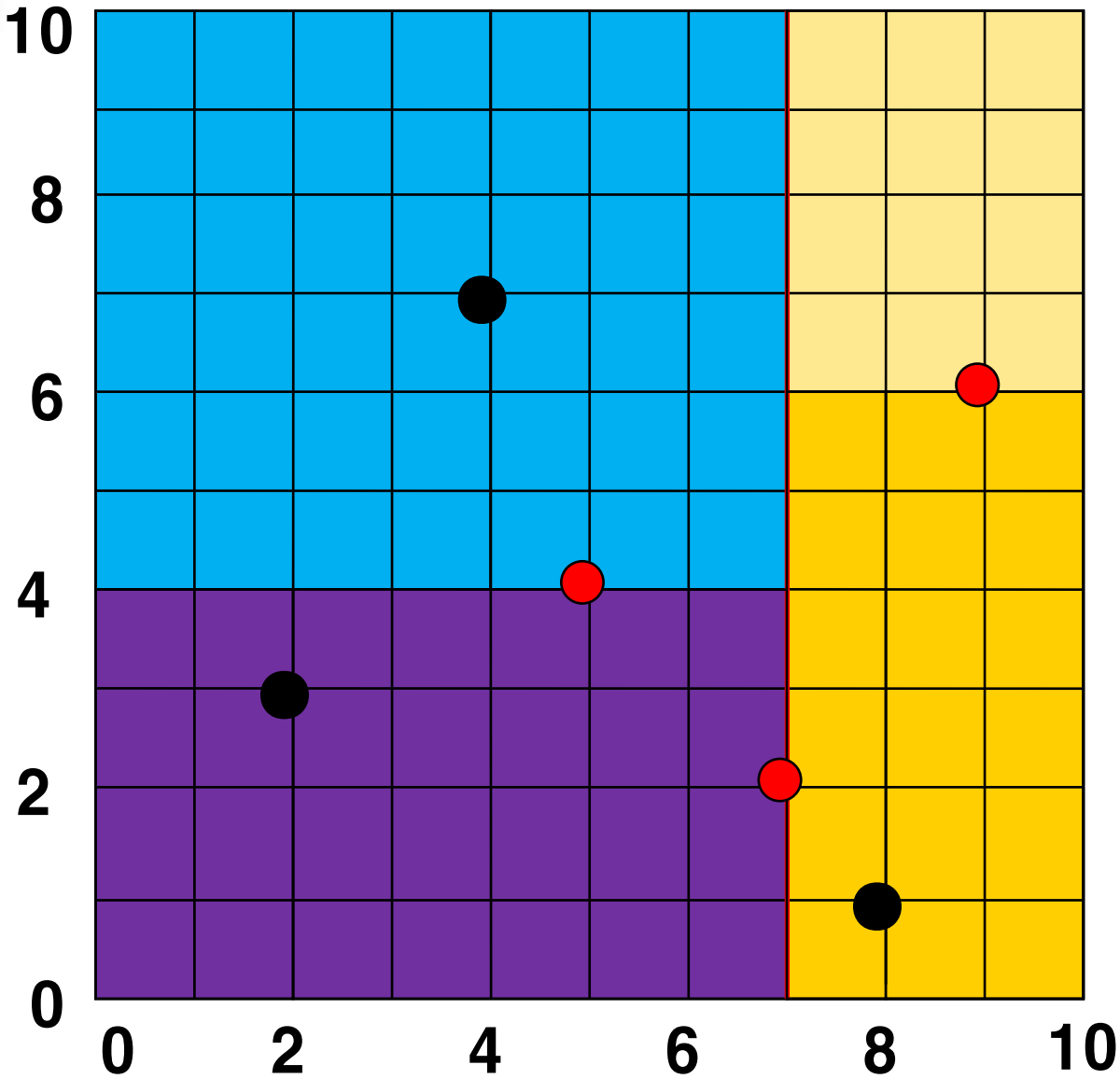
Second, Select Point (5,4) and (9,6) from Each k-d Space, and Partition Each k-d Space along the X Axis



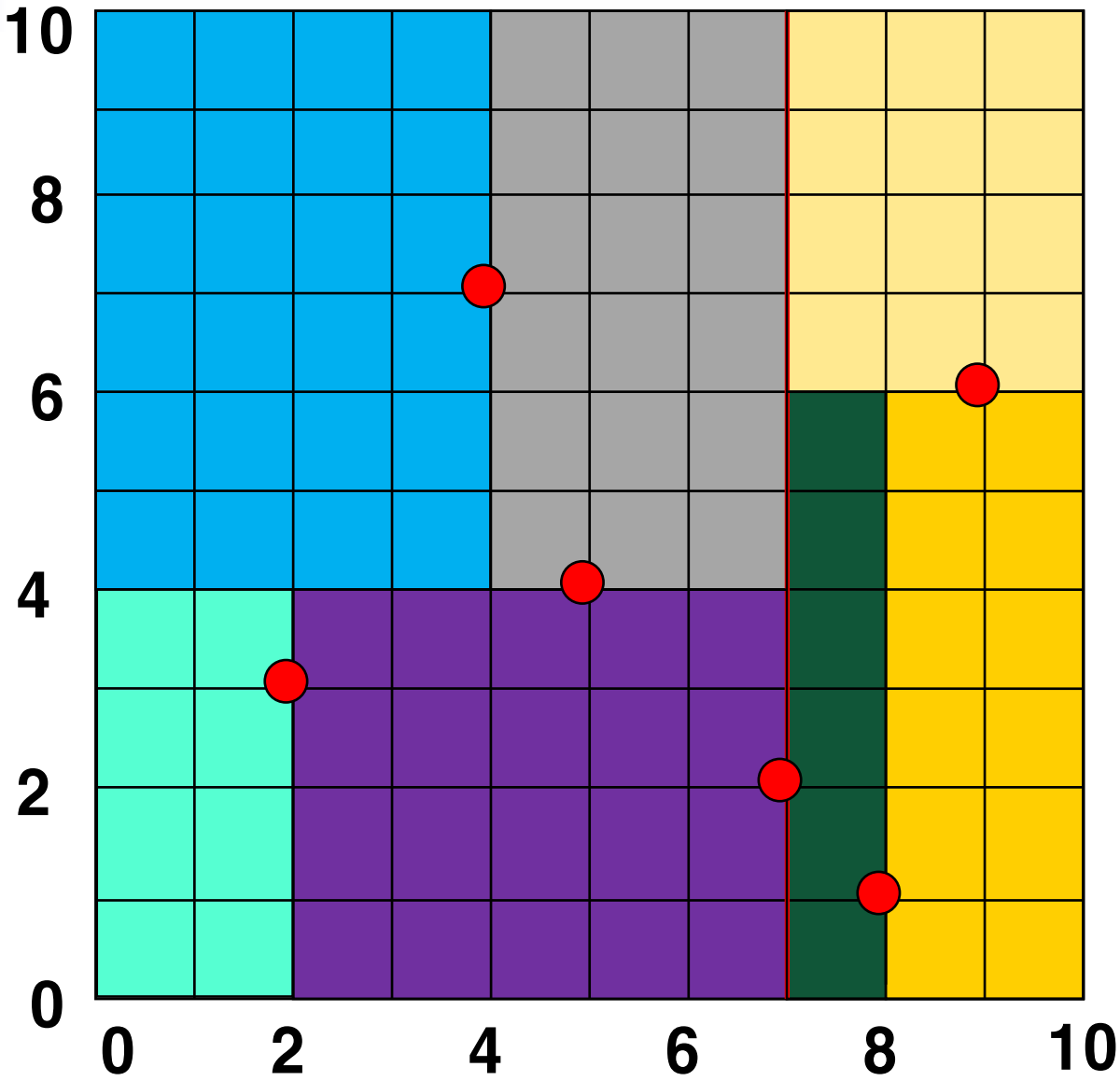
Result



Third, Select Point (2,3), (4,7), and (8,1) from Each k-d Space, and Partition Each k-d Space along the Y Axis

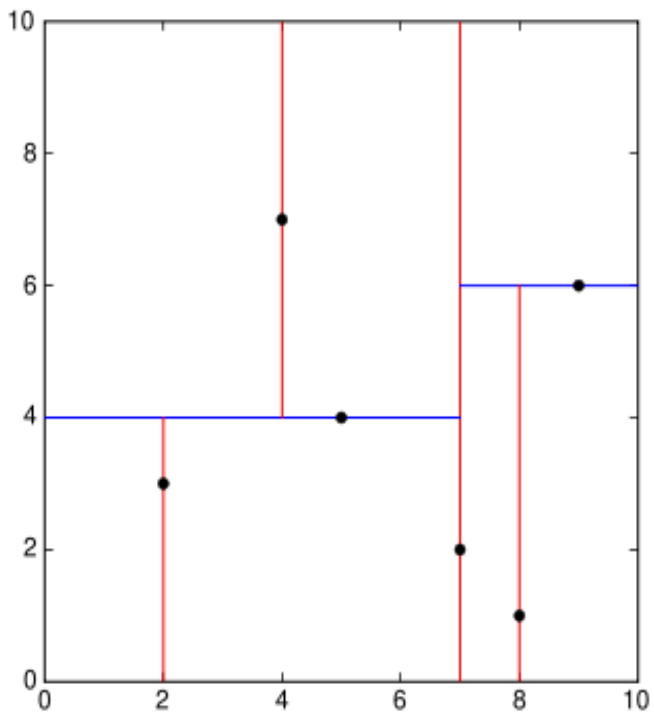


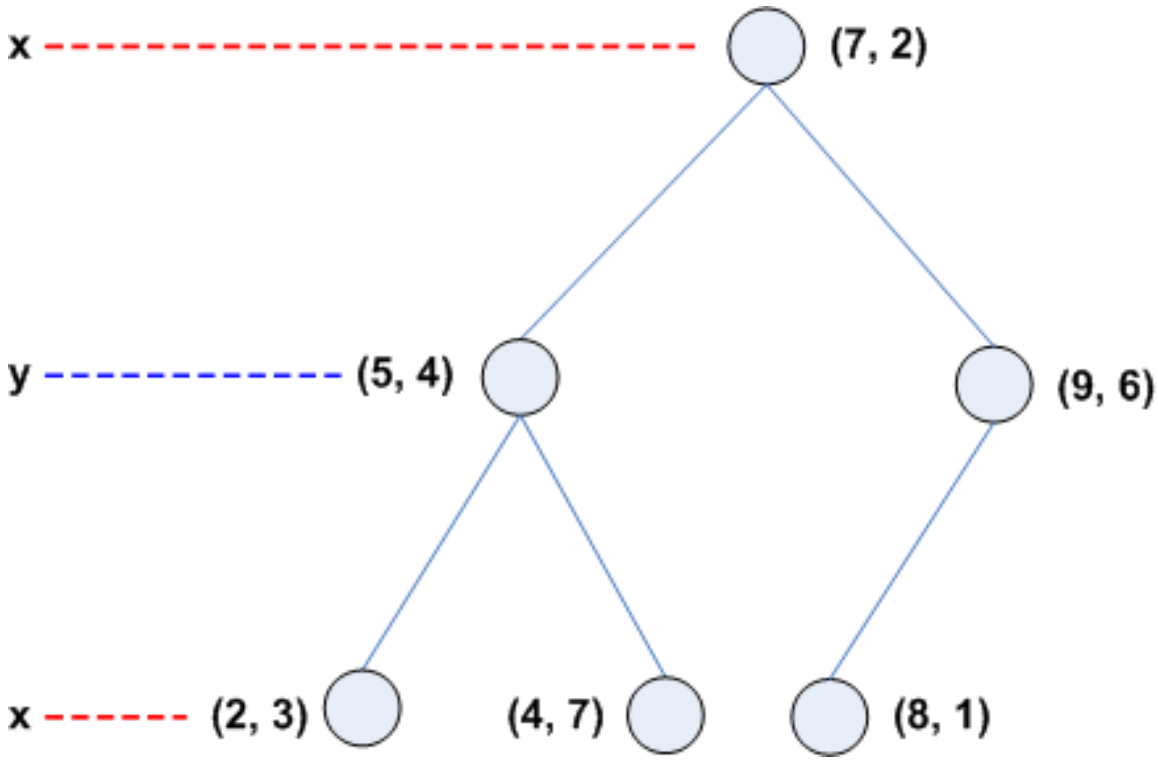
Result



Map Selected Points in a k-d Space to Nodes of a k-d Tree

How are these nodes selected? (We will see shortly…)





Constructing a k-d Tree

- In general, **static** construction with a set of given points in a k dimensional space.
- Each level of the tree represents a partitioning axis.
- The partitioning axis cycles through the k dimensions.
  - (e.g.) k=2 x,y,x,y,x,y,…
  - (e.g.) k=3 x,y,z,x,y,z,x,y,z,…
  - (e.g.) k=4 x,y,z,w,x,y,z,w,x,y,z,w,…
- Discriminator is the median key at each level of the k-d tree.
  - Median key is selected to distribute the data evenly on the tree

Computing the Median

- "Median"
  - equal number of values < and >
  - (e.g.) 4 7 9 10 11 15 30
- If the numbers are different,
  - Select the Largest of the Smaller Group or the Smallest of the Larger Group.
  - (e.g.) 4 7 10 11 15 30
    - 4 7 10 11 15 30
    - 4 7 10 11 15 30
- If there are multiple identical values in a median candidate?
  - Results in a skewed tree
  - (e.g.) 4 7 10 10 10 10 30
    - 4 7 10 10 10 10 30
- either "<= median" go to the left subtree
  - "> median" go to the right subtree
- or "< median" go to the left subtree
  - ">= median" go to the right subtree

Example (1/5)

set of points in 2-dimension

X	Y
2	3
4	7
5	4
7	2
8	1
9	6

select the median X value

Example (2/5)

x

7,2

select the median x value

2, 4, 5, 7, 8, 9

X <= 7

X	Y
2	3
4	7
5	4

X	Y
8	1
9	6

X > 7

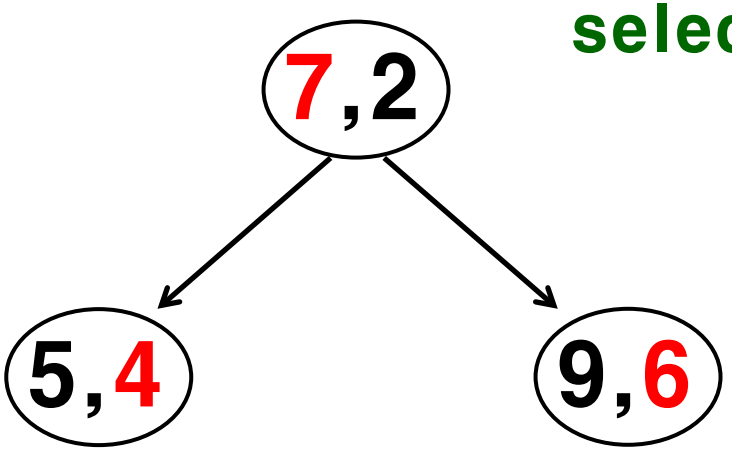
select the median Y value

select the median Y value

Example (3/5)

x

y



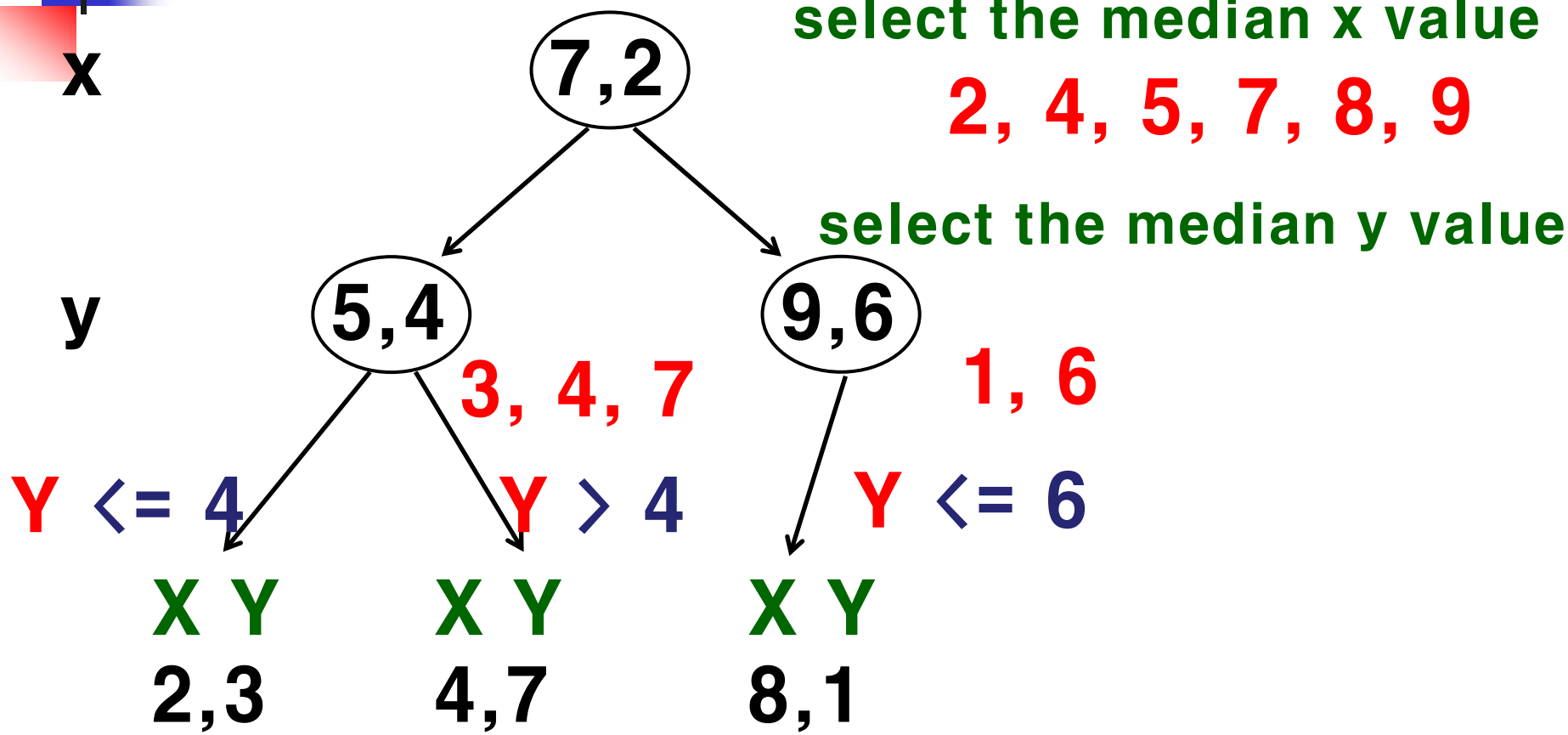
select the median x value

2, 4, 5, 7, 8, 9

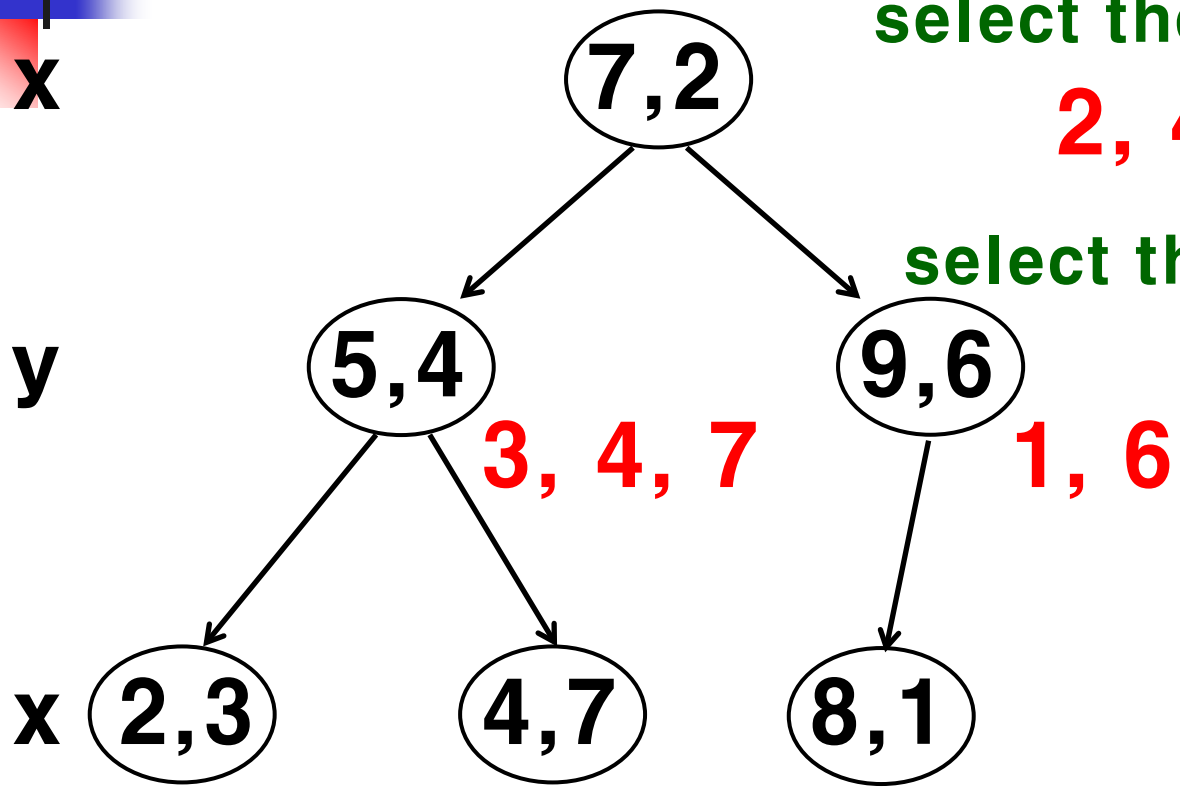
select the median y value

select the median y value

Example (4/5)


x
y
7,2
select the median x value
2, 4, 5, 7, 8, 9
select the median y value
5,4
9,6
3, 4, 7
1, 6
Y <= 4
Y > 4
Y <= 6
X Y
2,3
X Y
4,7
X Y
8,1

Example (5/5)



select the median x value

2, 4, 5, 7, 8, 9

select the median y value

select the median x value

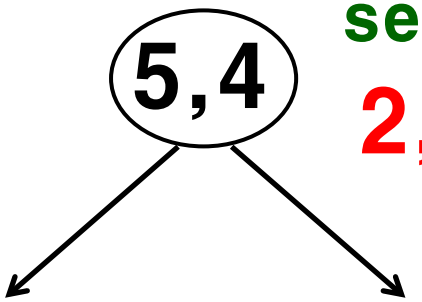
Example (1/6)

set of points in 2-dimension

X Y
2,3
5,1
5,2
5,4
5,6
5,7
9,8
10,6

Example 2 (2/6)

x



select the median x value

2,5,5,5,5,5,9,10

X	Y
2	3
5	1
5	2
5	4
5	6
5	7
9	8
10	6

Example 2 (3/6)

x

5,4

select the median x value

2,5,5,5,5,5,9,10

X	Y
2	3
5	1
5	2
5	6
5	7

X	Y
9	8
10	6

X	Y
2	3
5	1
5	2
5	4
5	6
5	7
9	8
10	6

Example 2 (4/6)

x

5,4

select the median x value

2,5,5,5,5,5,9,10

X	Y
5,	1
5,	2
2,	3
5,	6
5,	7

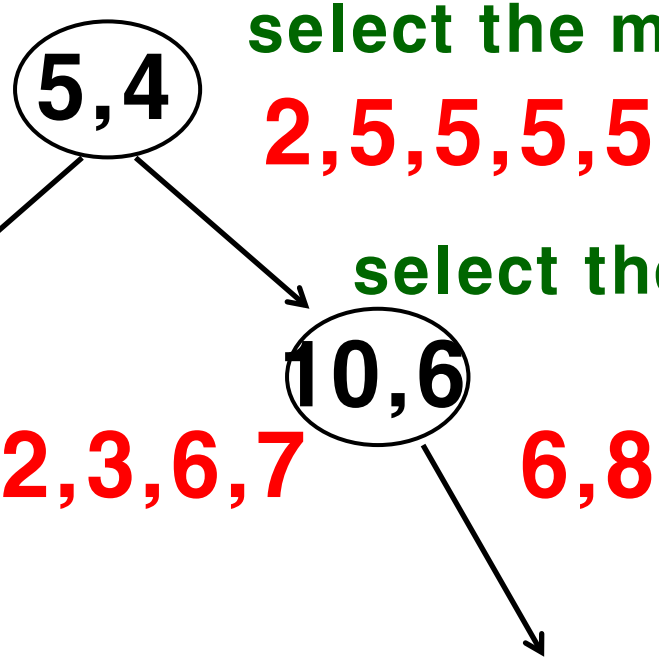
X	Y
10,	6
9,	8

X	Y
2,	3
5,	1
5,	2
5,	4
5,	6
5,	7
9,	8
10,	6

Example 2 (5/6)

x

y



select the median x value

2,5,5,5,5,5,9,10

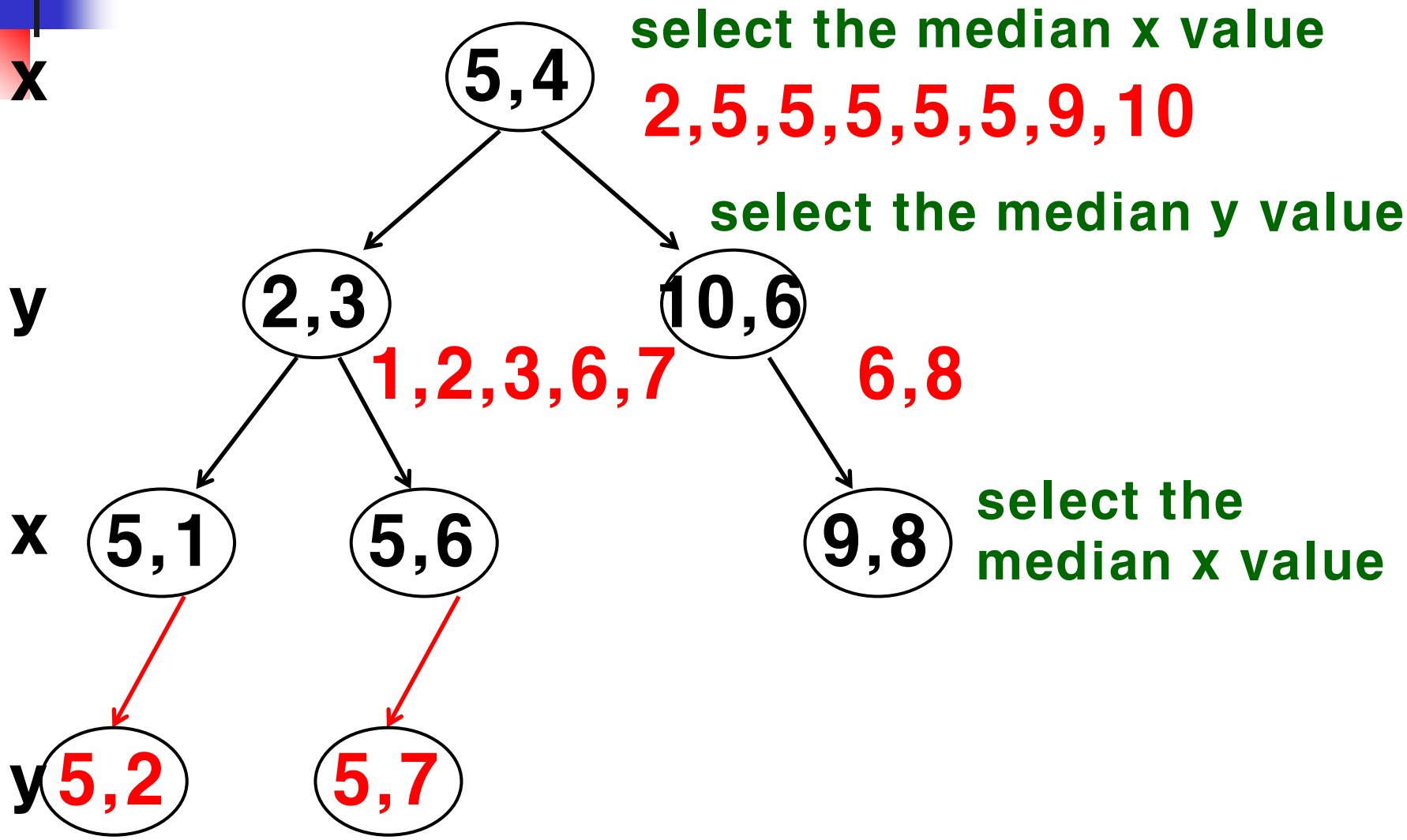
select the median y value

X	Y
5	1
5	2

X	Y
5	6
5	7

X	Y
9	8

Example 2 (6/6)

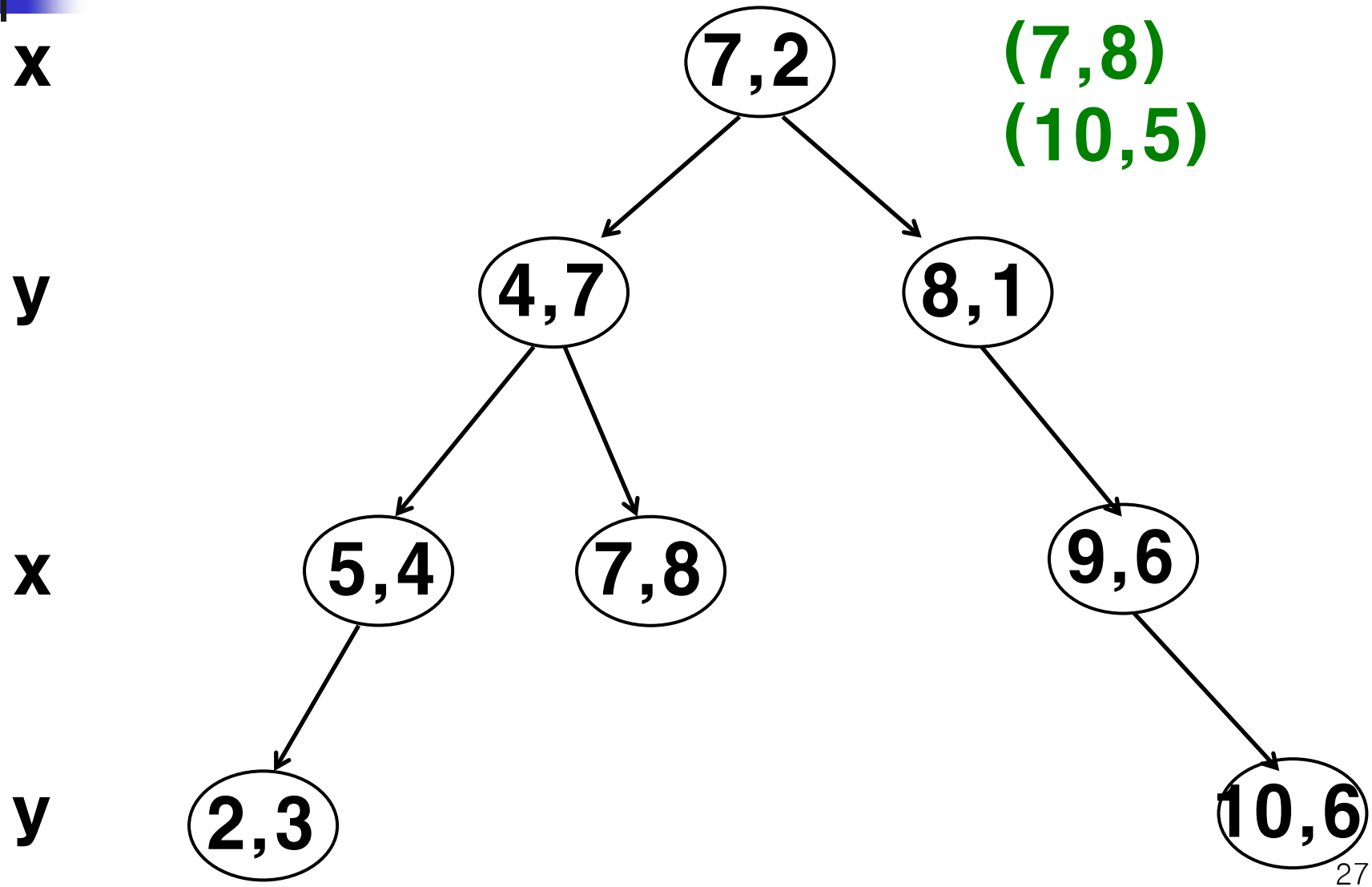

x
5,4
select the median x value
2,5,5,5,5,5,9,10
select the median y value
y
2,3
10,6
1,2,3,6,7
6,8
x
5,1
5,6
9,8
select the median x value
y
5,2
5,7

Exercise

set of points in 3-dimension

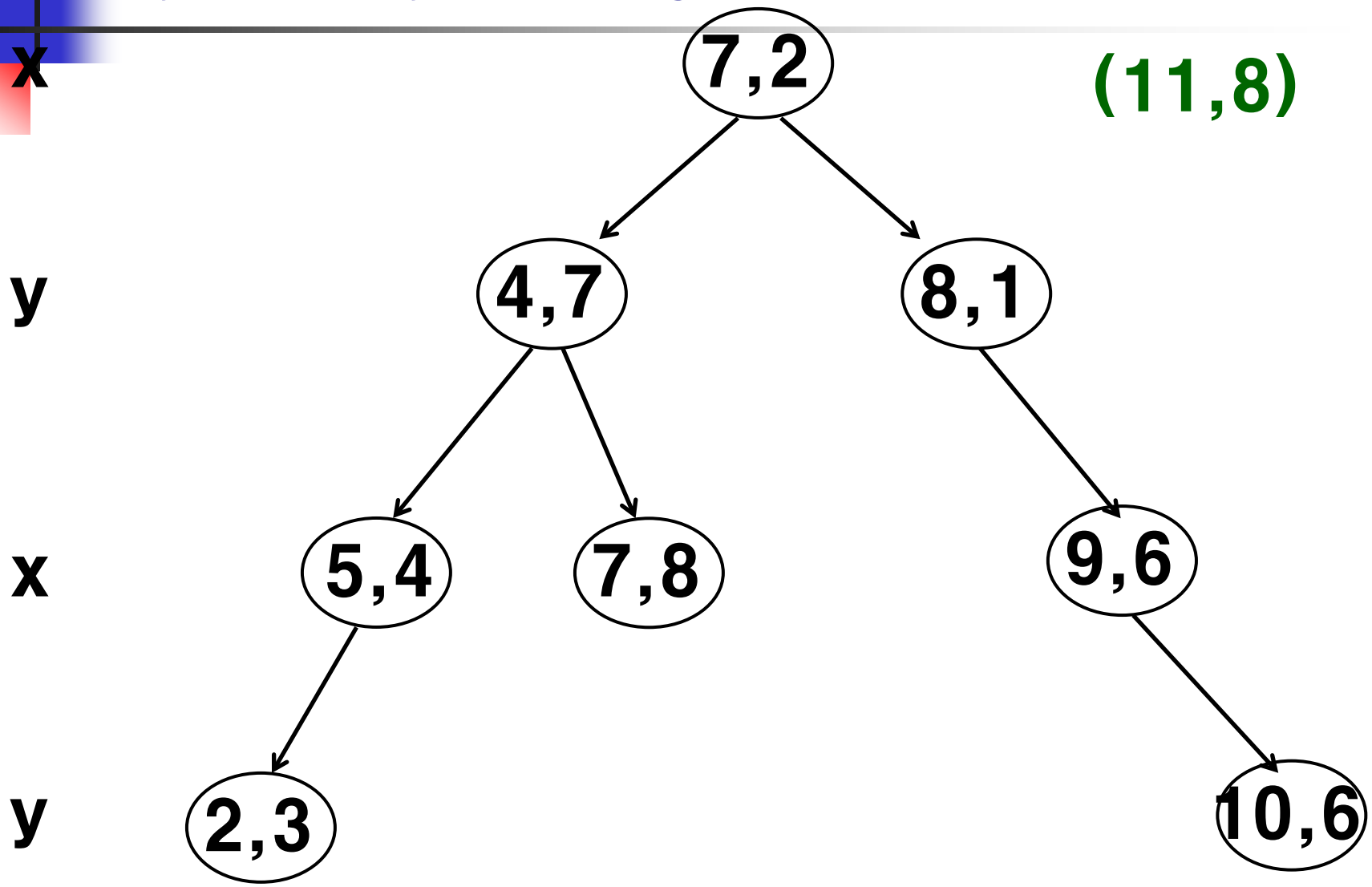
X	Y	Z
1	4	5
2	3	11
3	7	4
4	10	6
5	12	9
6	2	8
7	8	4
9	5	3
10	6	5
11	11	11
12	1	1

Searching a k-d Tree: (similar to constructing a k-d tree)

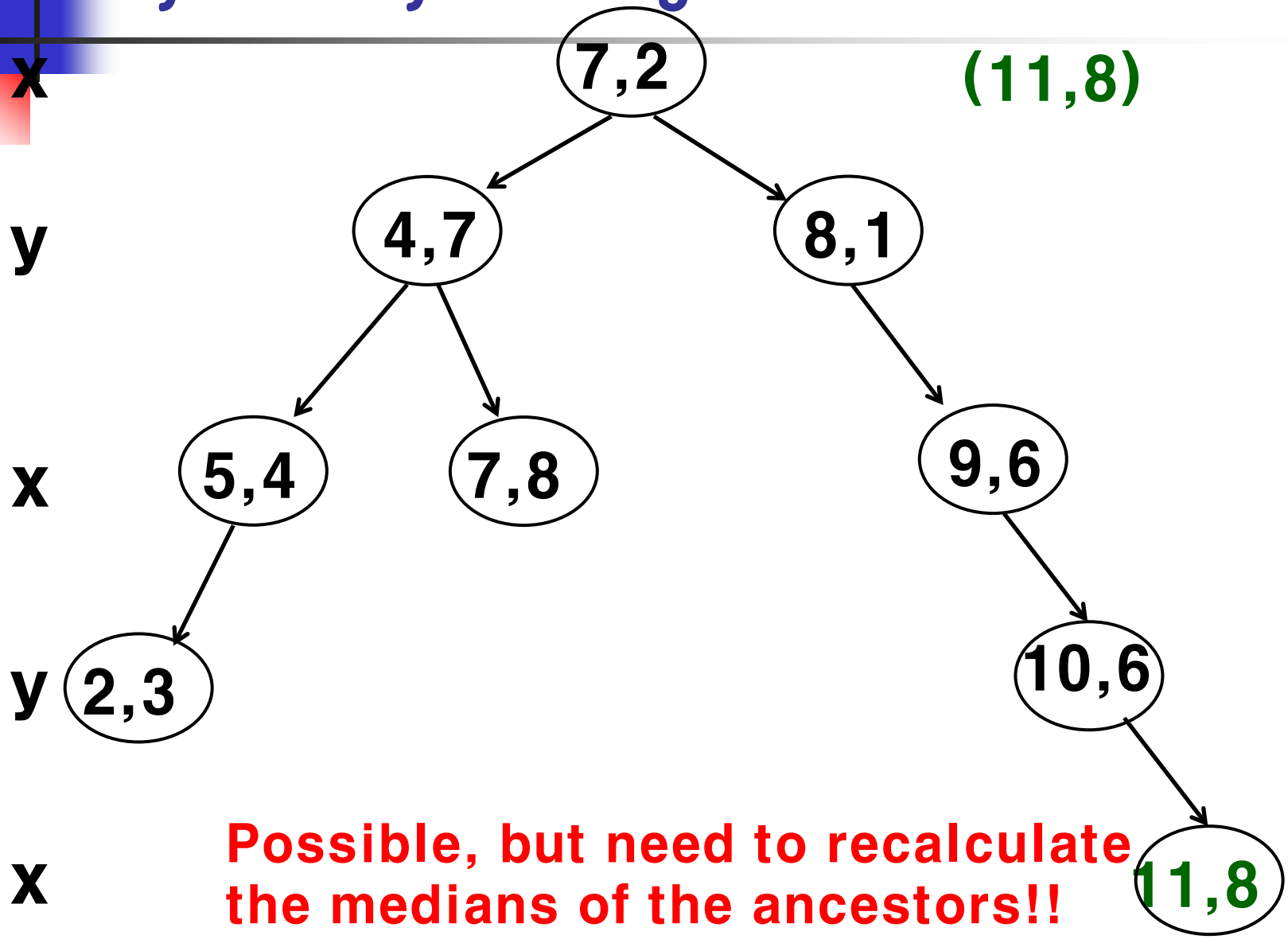


(7,8)
(10,5)

Dynamically Inserting a Node to a k-d Tree

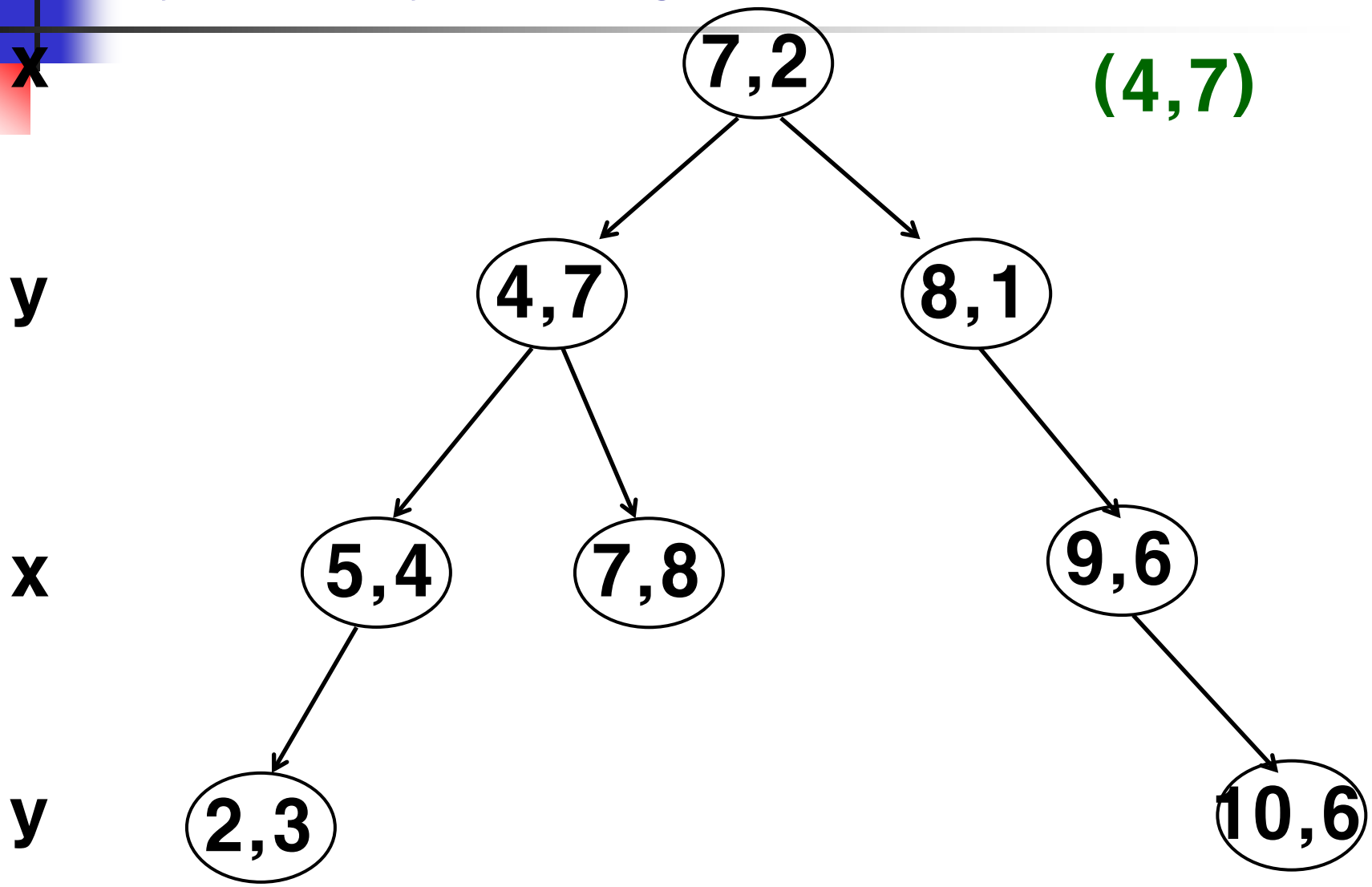


Dynamically Inserting a Node to a k-d Tree

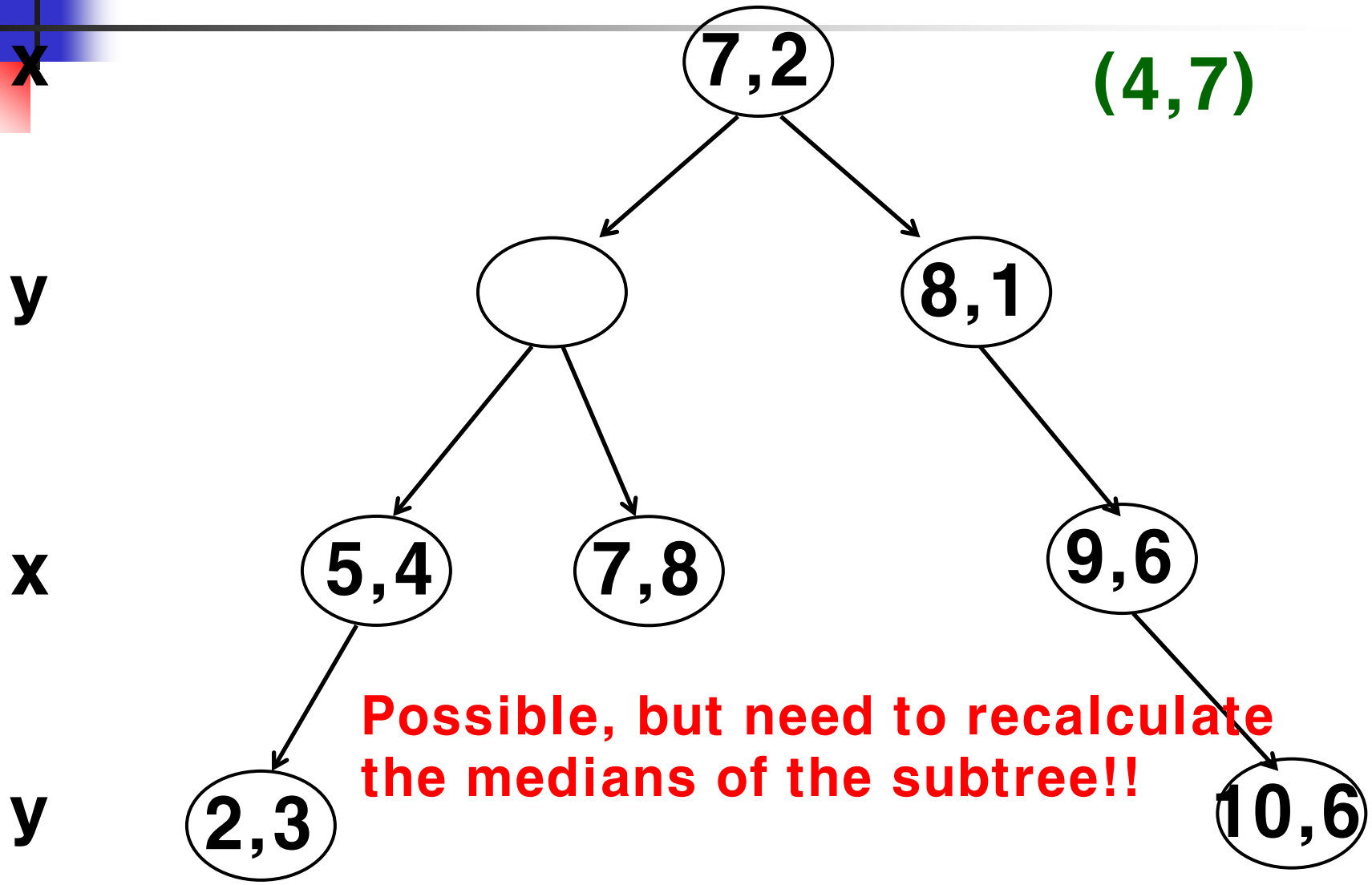


Possible, but need to recalculate the medians of the ancestors!!

Dynamically Deleting a Node from a k-d Tree

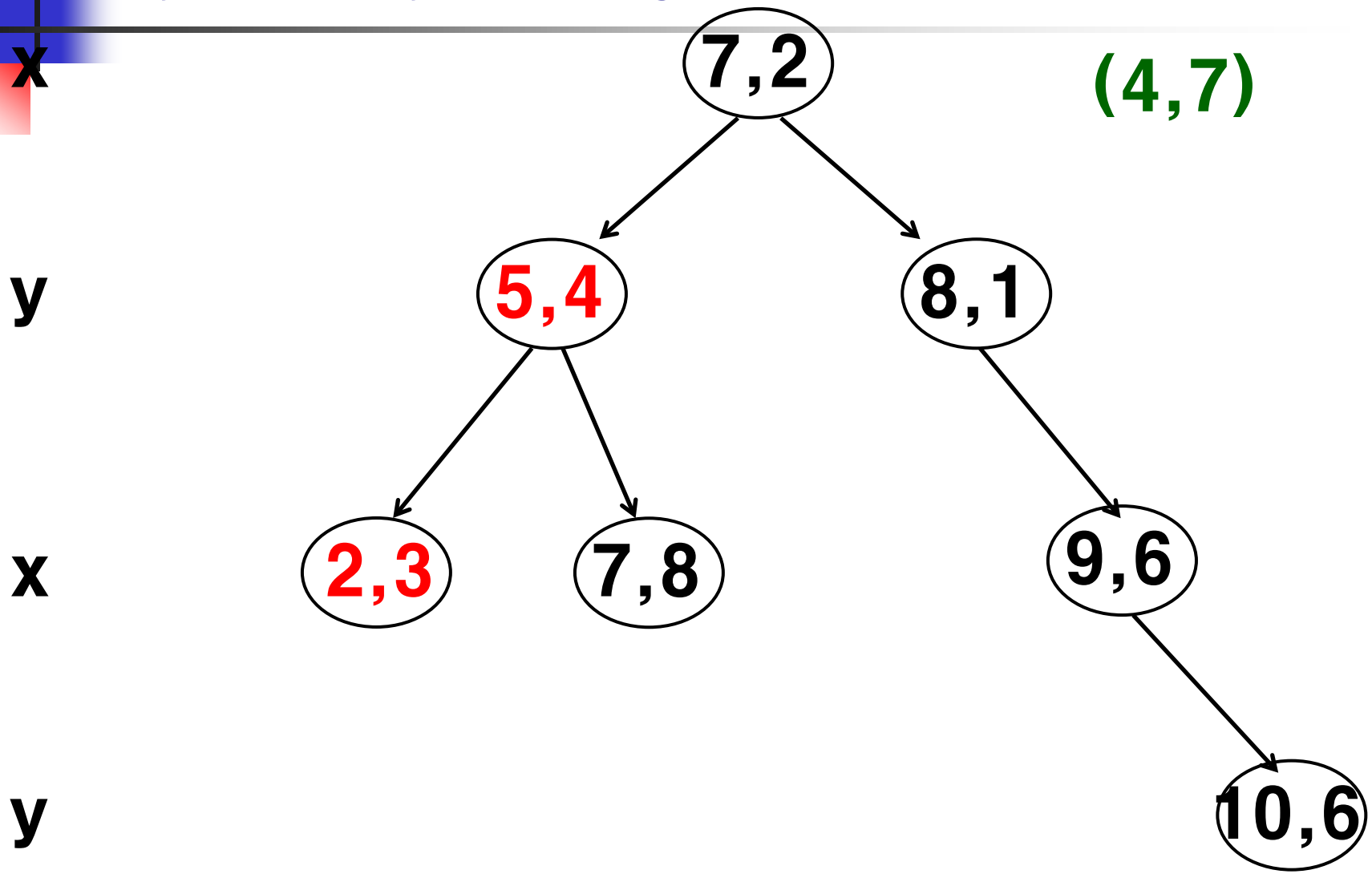


Dynamically Deleting a Node from a k-d Tree



Possible, but need to recalculate the medians of the subtree!!

Dynamically Deleting a Node from a k-d Tree



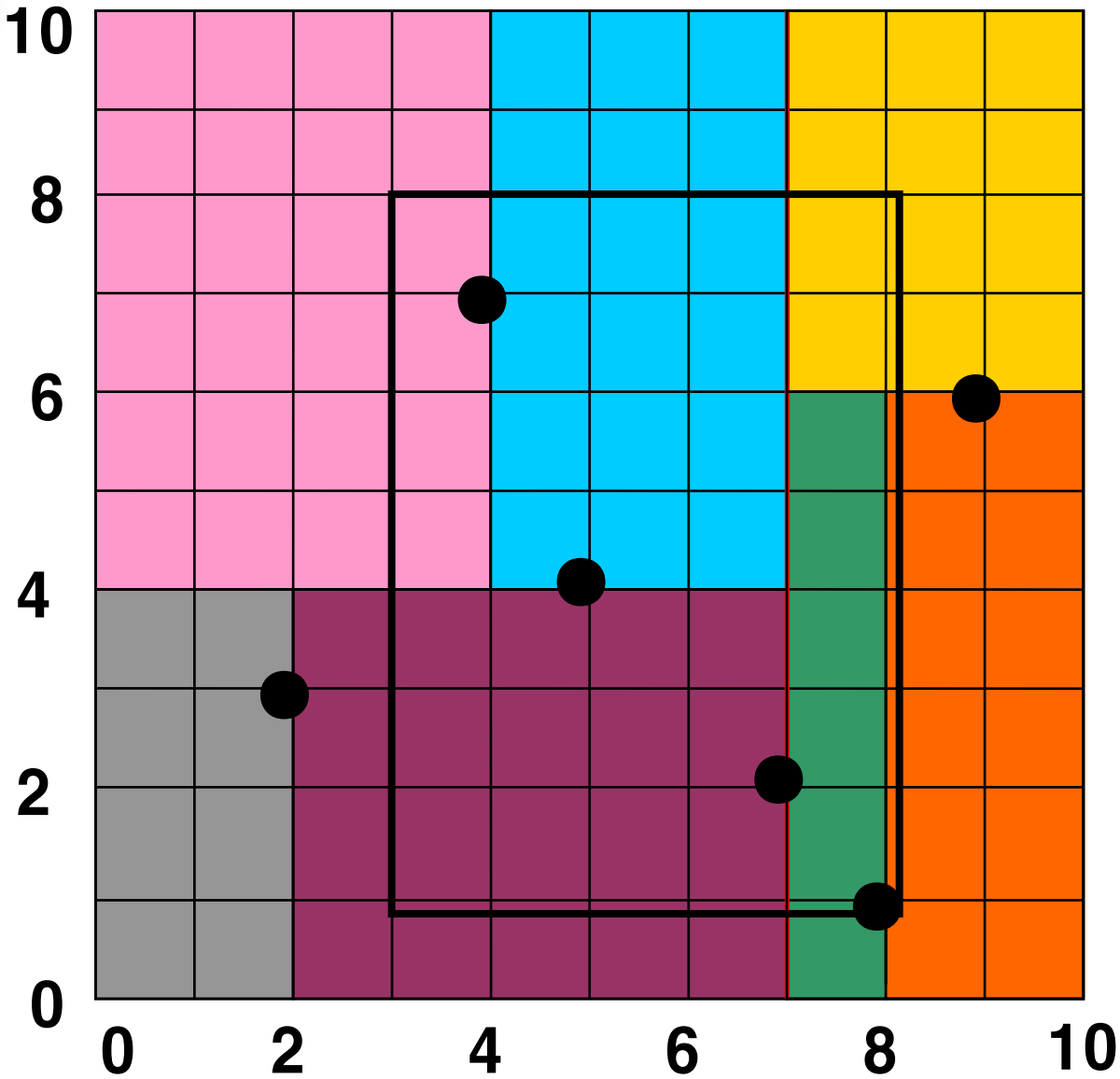
k-d Tree: Properties

- It is a binary search tree for k-d keys.
- It is not height-balanced
- It can be badly skewed if there are many identical median values at any level.
- It is a static tree for searches; (i.e.,) it is constructed once for repeated searches.
- Dynamic insertion and deletion of keys can require partial reorganizations of the tree.

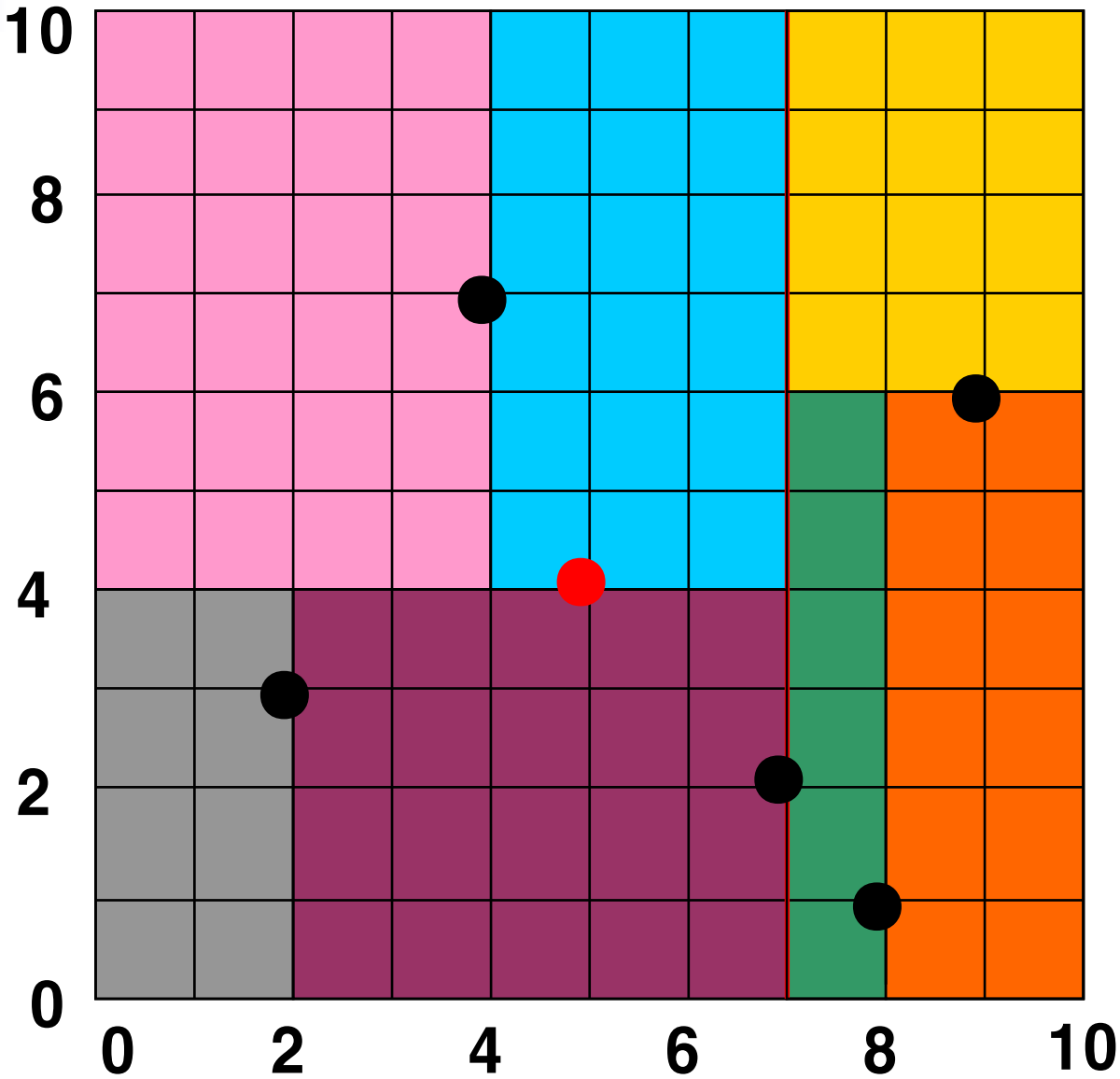
k-d Tree: Uses

- Point search
- Range (region) search
- Nearest neighbor search
- Partial key search

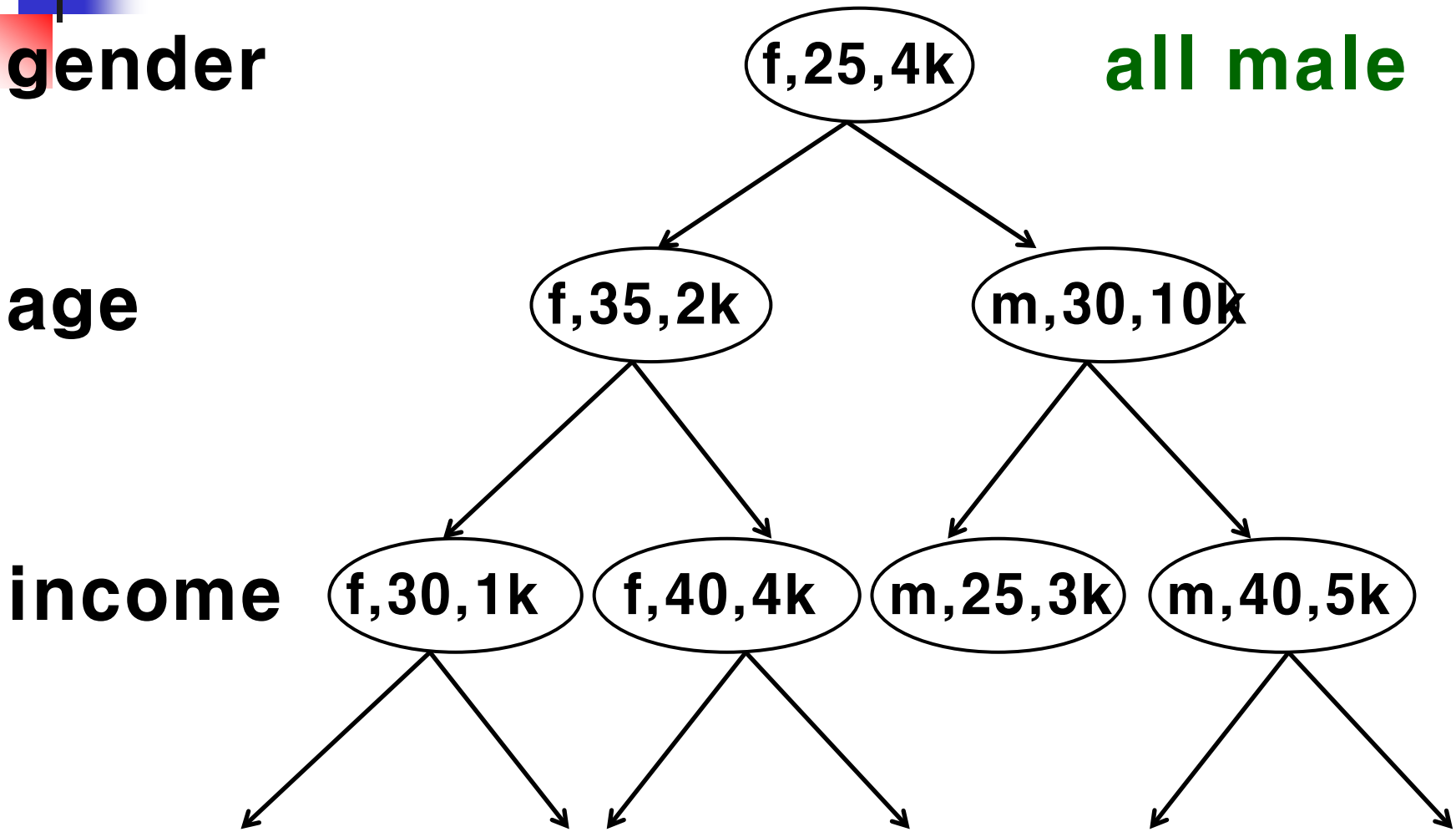
Example: Range Search



Example: Nearest Neighbor Search



Example: Partial Key Search



Tries

Binary Trie

- Retrieval
- A binary search tree
  - Not height-balanced
- Can be used as a replacement for hashing

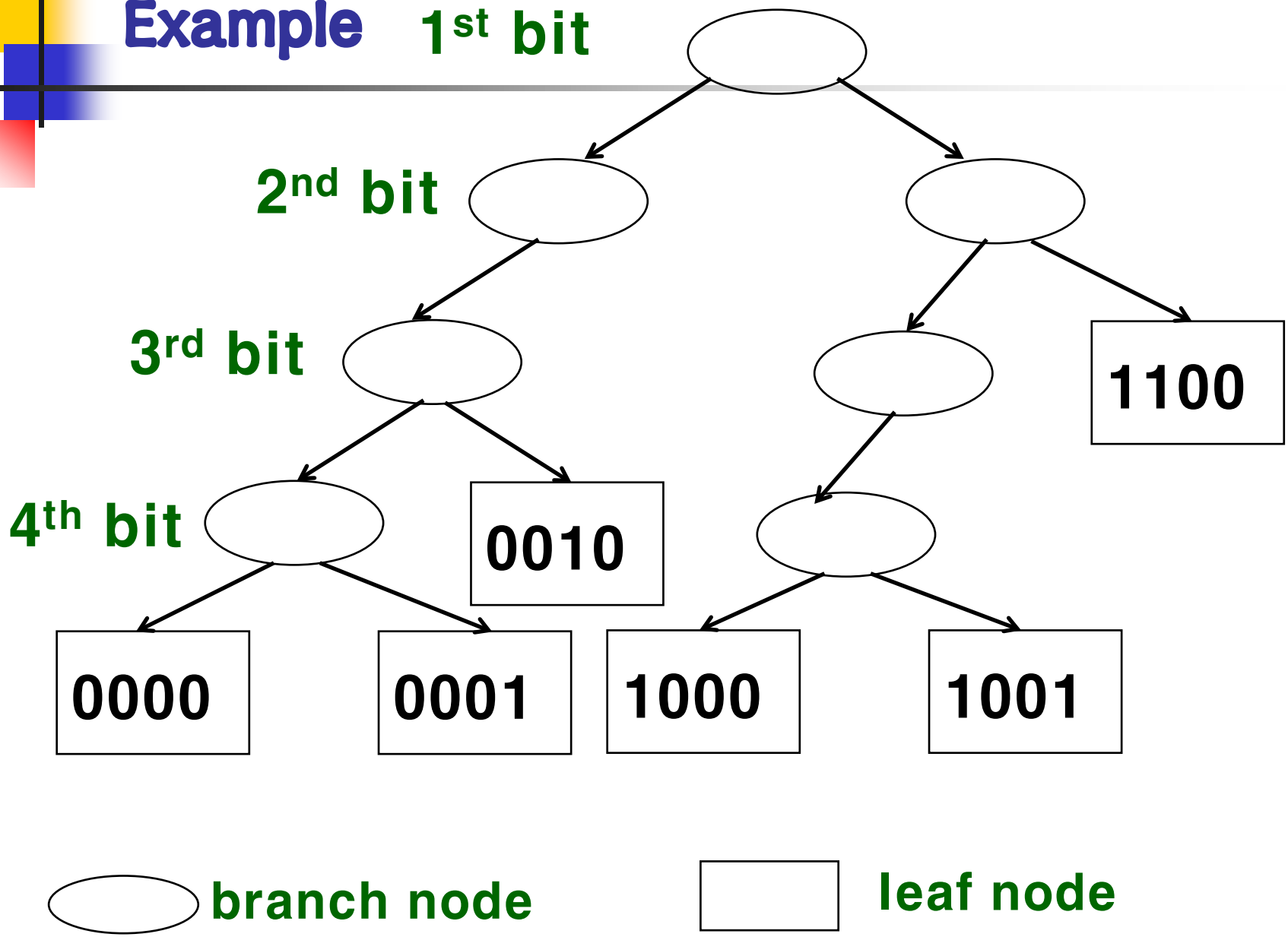
Binary Trie

- Two Types of Node
  - Branch (link, interior, non-leaf) node
    - left-child ptr, right-child ptr
    - no data
  - Element (data, leaf) node
    - contains the full key

Binary Trie: Search

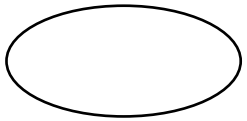
- For a Search Key k
  - At level i
    - if k's i^{th} bit is 0, move to the left subtree;
    - if it is 1, move to the right subtree.
  - At a leaf node, the search key is compared with the data stored in the node.
- End of Search
  - success: at a leaf node
  - failure: at a leaf node or a NULL pointer

Example


1st bit
2nd bit
3rd bit
4th bit
1100
0010
0000
0001
1000
1001
branch node
leaf node

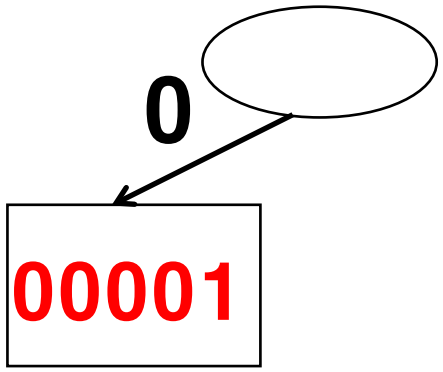
Building a Binary Trie

Insert 00001



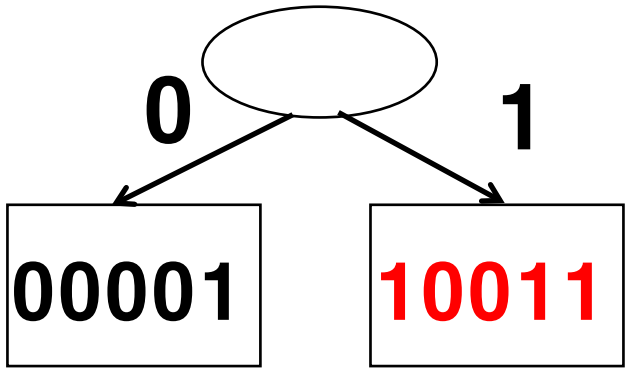
Building a Binary Trie

Insert 10011

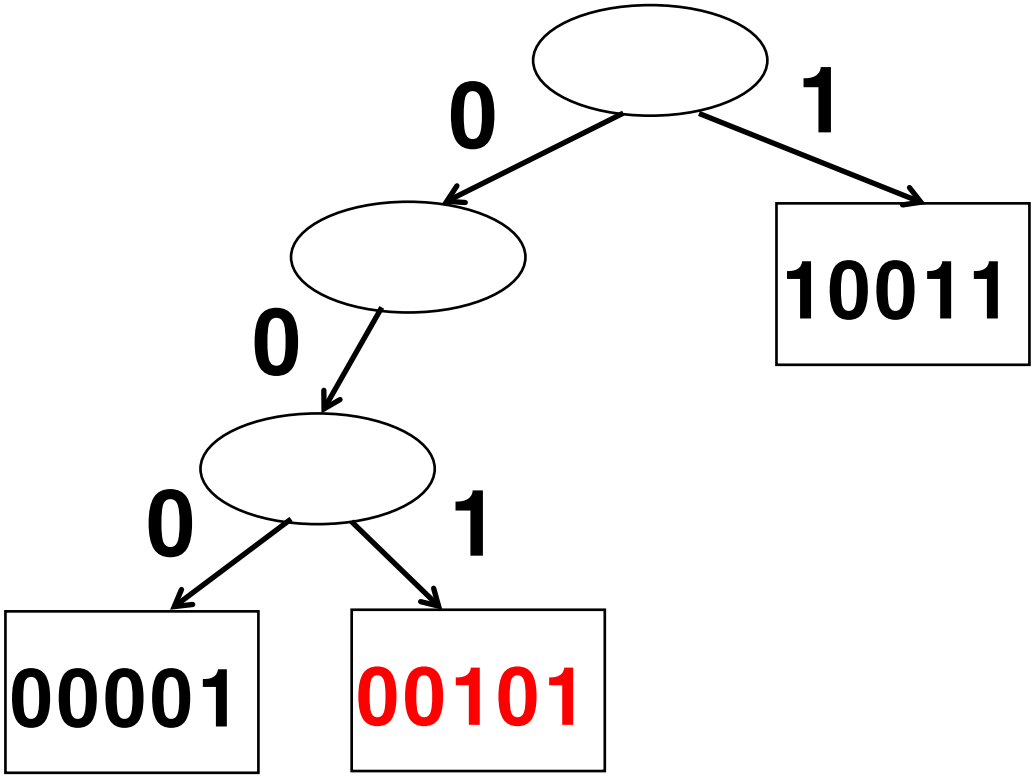


Building a Binary Trie

00101

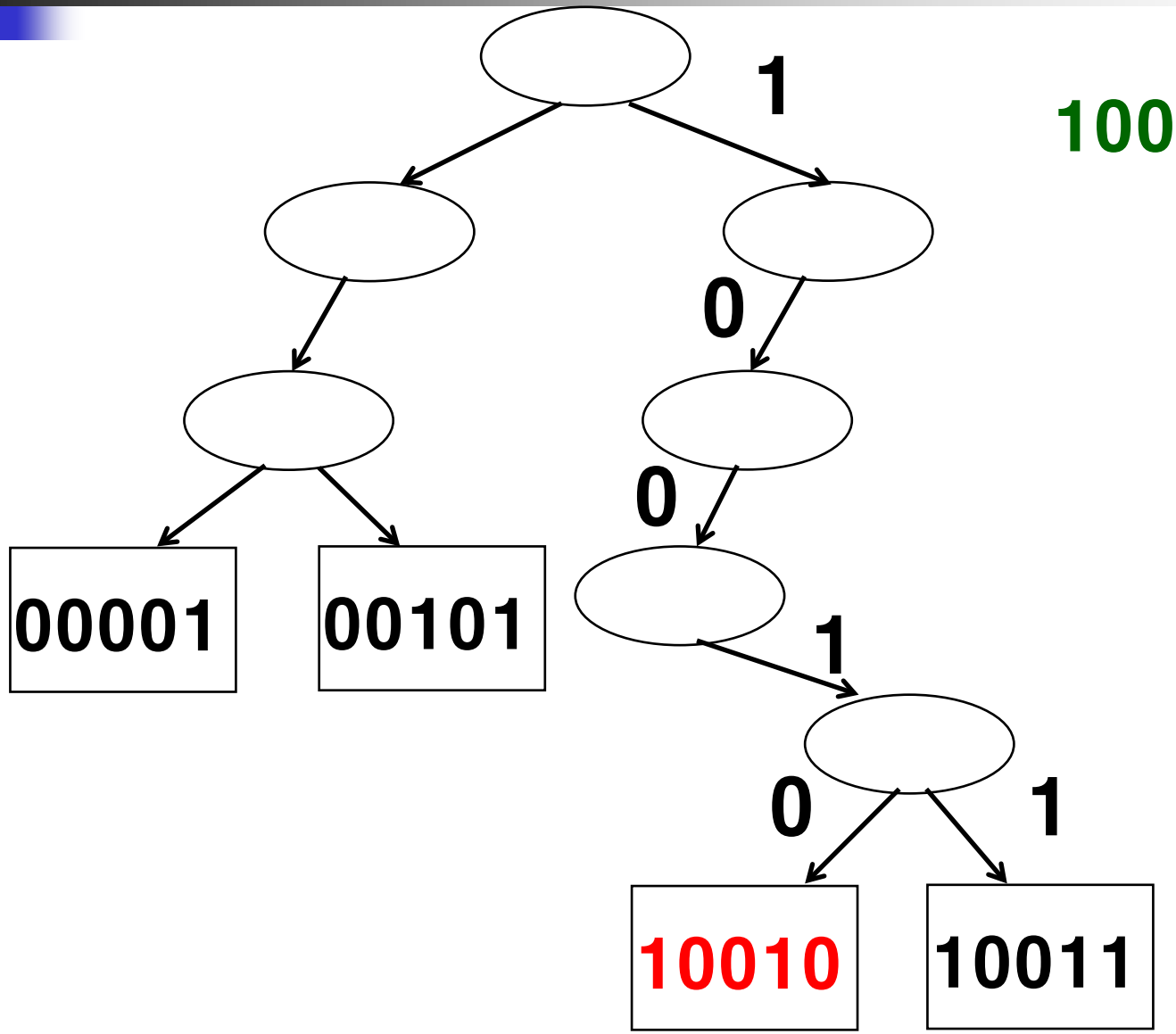


Building a Binary Trie – cont'd



10010

Building a Binary Trie – cont'd



10010

Exercise: Building a Binary Trie

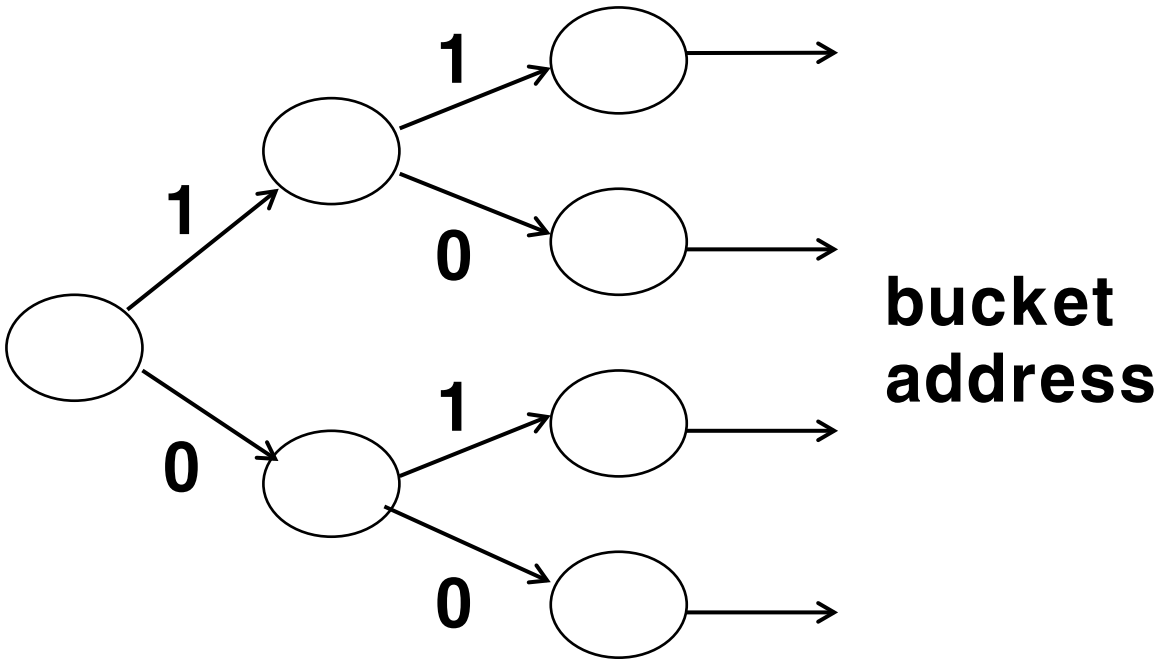
Insert 10010
10011
00001
00101

Binary Trie: Performance and Properties

- Avg. performance: o(k)
  - K is the maximum number of bits in the keys
- One comparison in a search
  - Only at the leaf node
- Efficient for exact key match and partial key match.
- Unbalanced tree
- The shape of the tree is independent of the order of insertions.

Binary Trie In Use

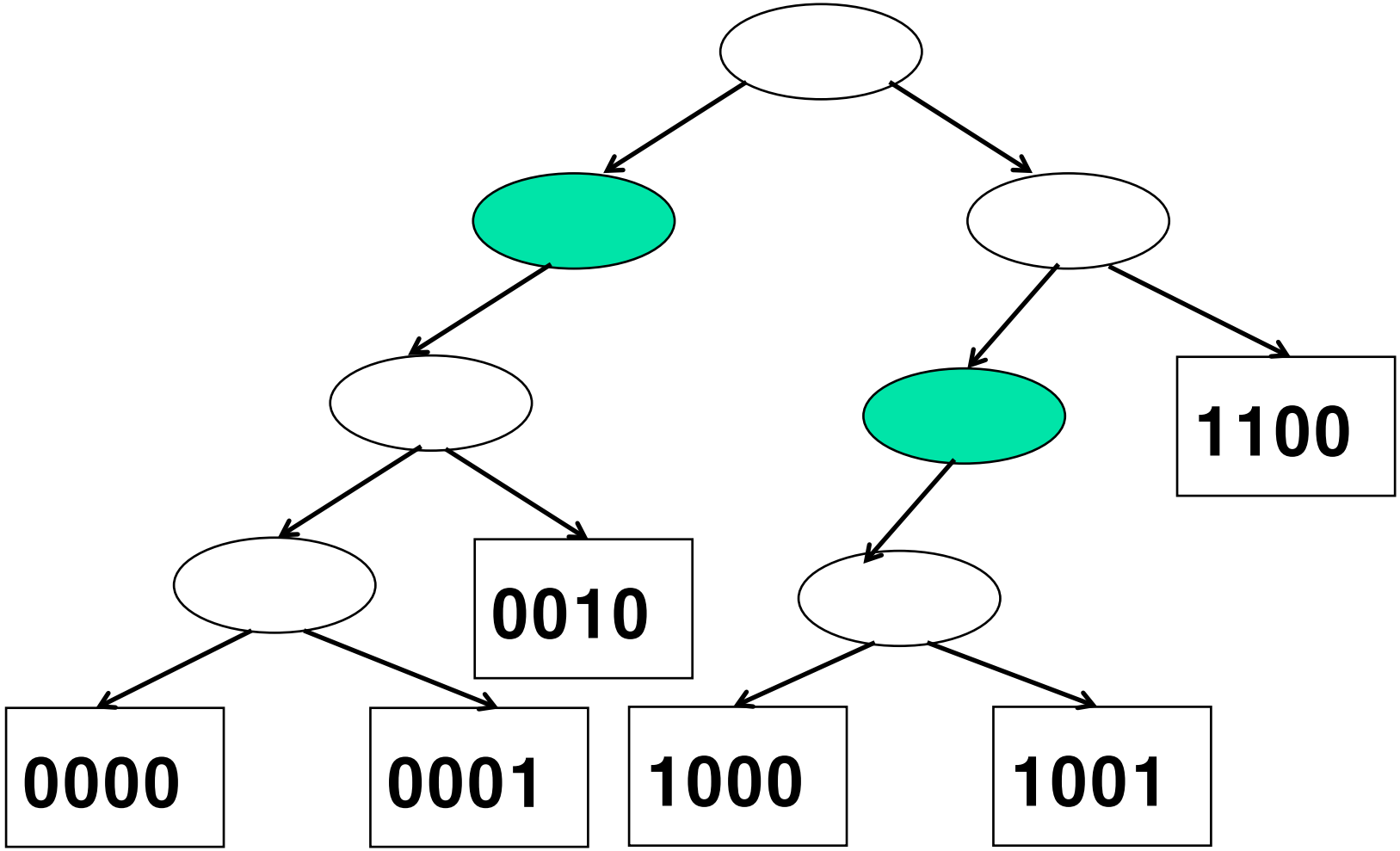
- Bucket directory in extendible hashing (to learn later)



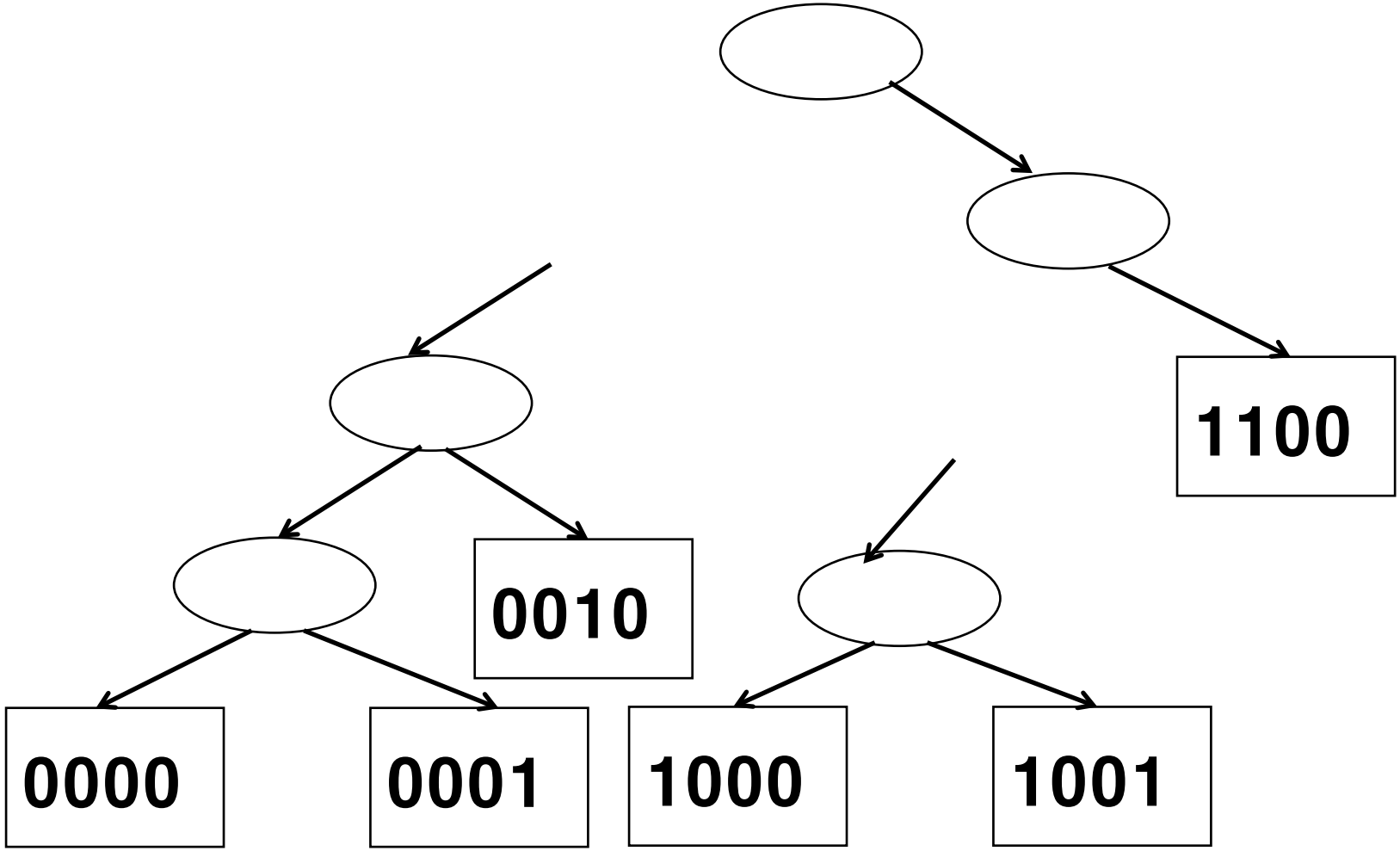
Compressed Binary Trie

- Branch nodes in a binary trie may be degree 1.
- Compressing a binary trie
  - Binary trie with degree-1 branch nodes eliminated.
  - Tree height is reduced.
- (On a compressed binary trie) each branch node is degree 2
- Branch node
  - left-child ptr, right-child ptr, bit-number

Eliminating Degree 1 Branch Nodes

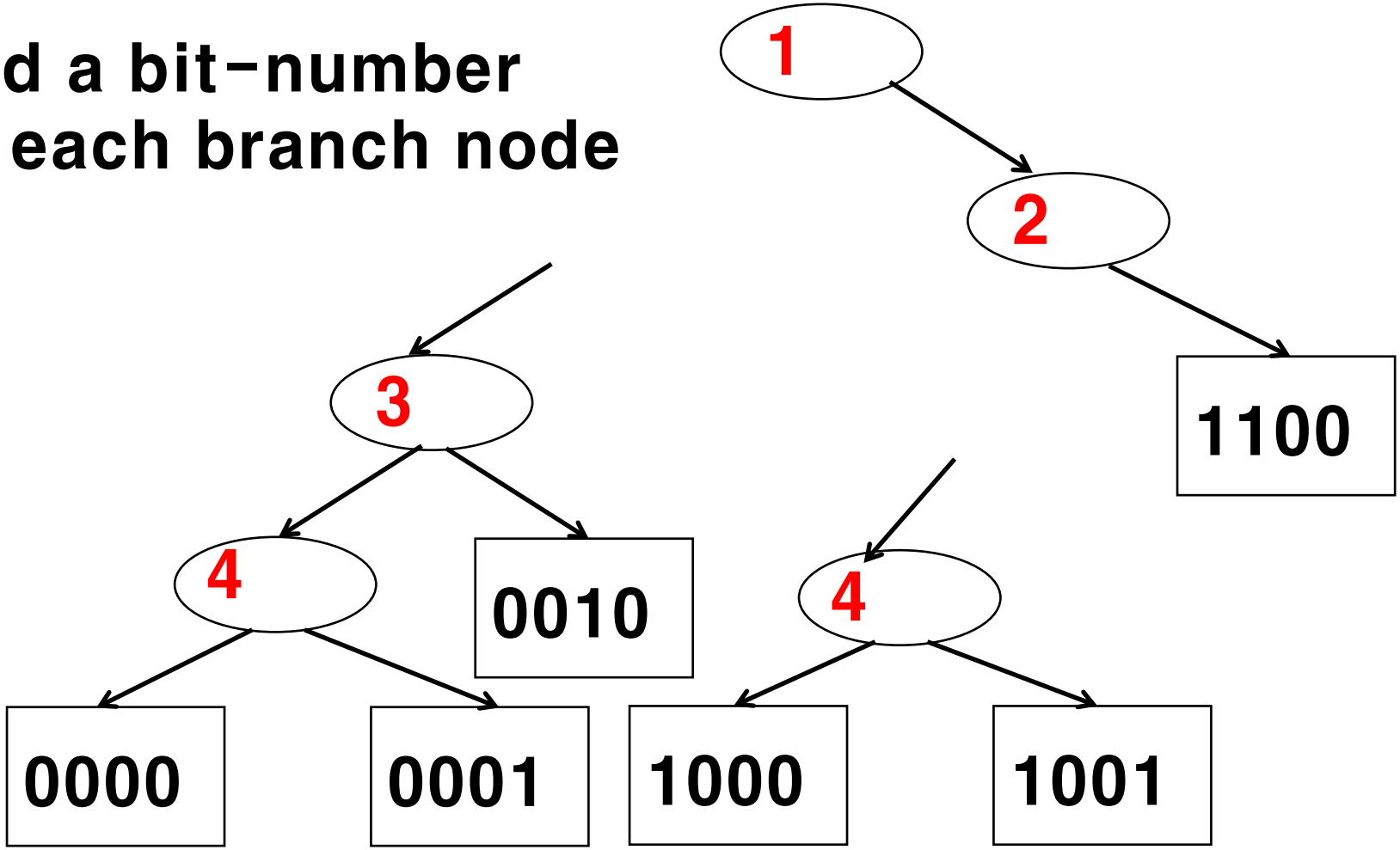


Eliminating Degree 1 Branch Nodes (cont'd)

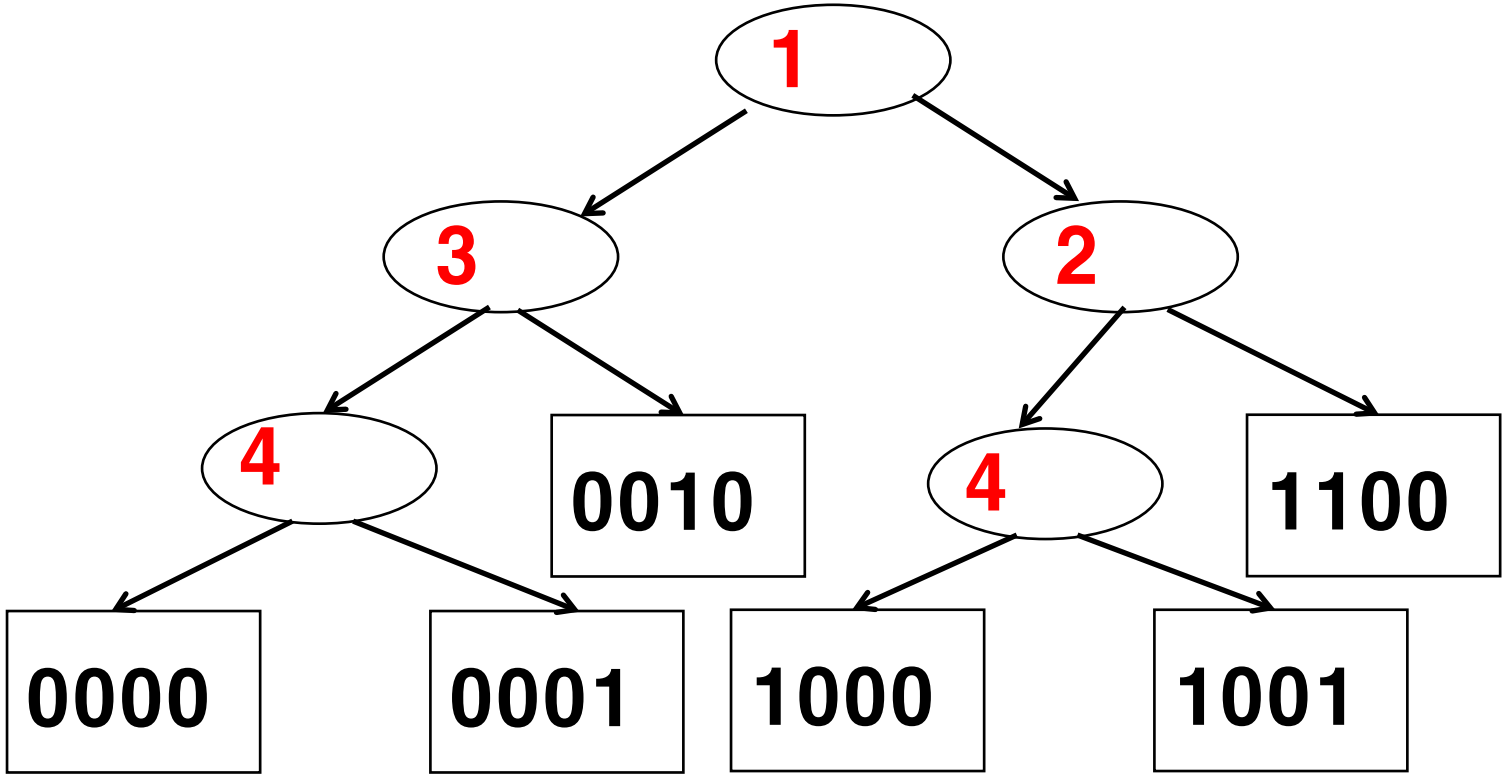


Eliminating Degree 1 Branch Nodes (cont'd)

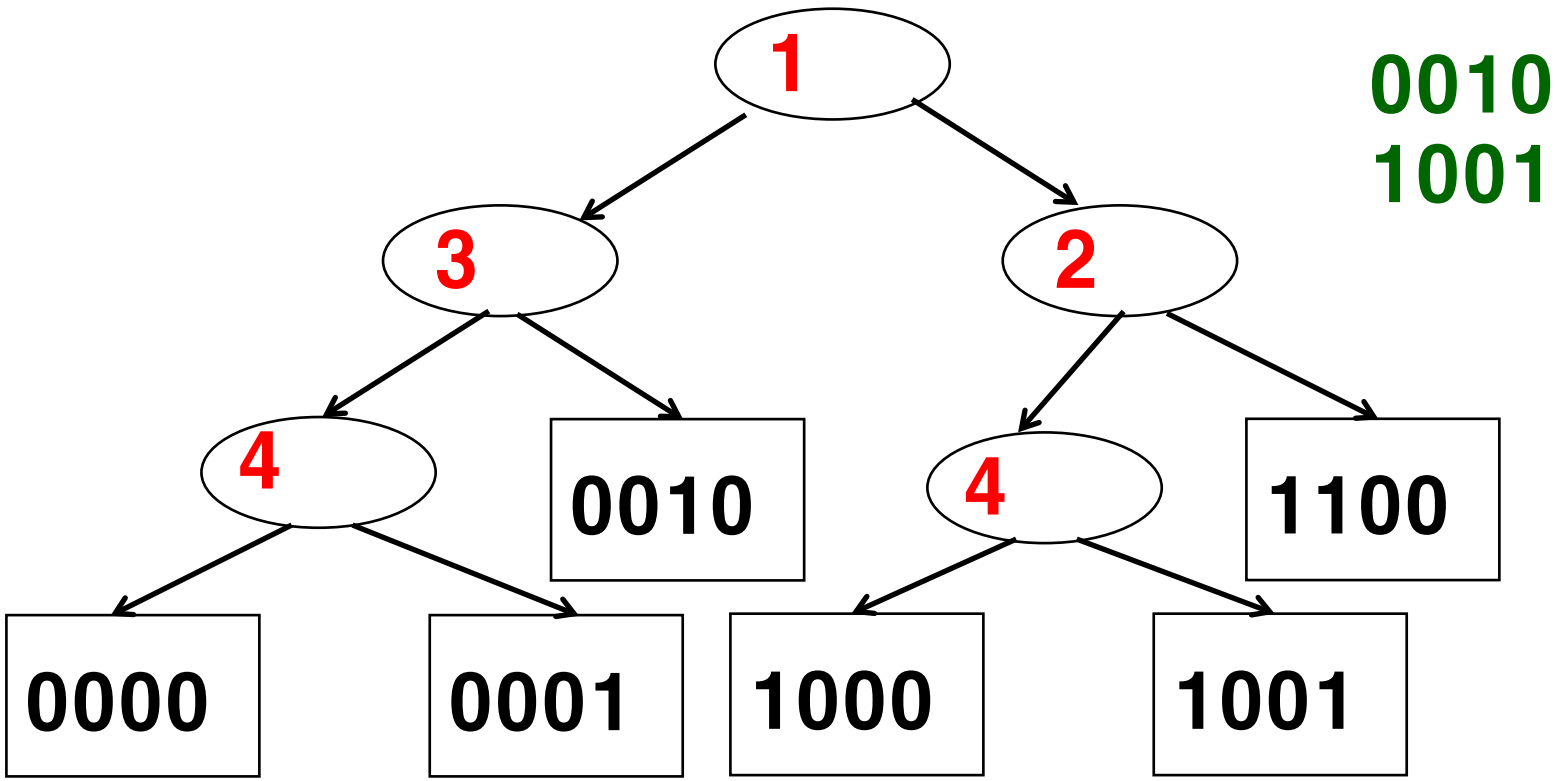
add a bit-number
to each branch node



Eliminating Degree 1 Branch Nodes (cont'd)



Searching a Compressed Binary Trie

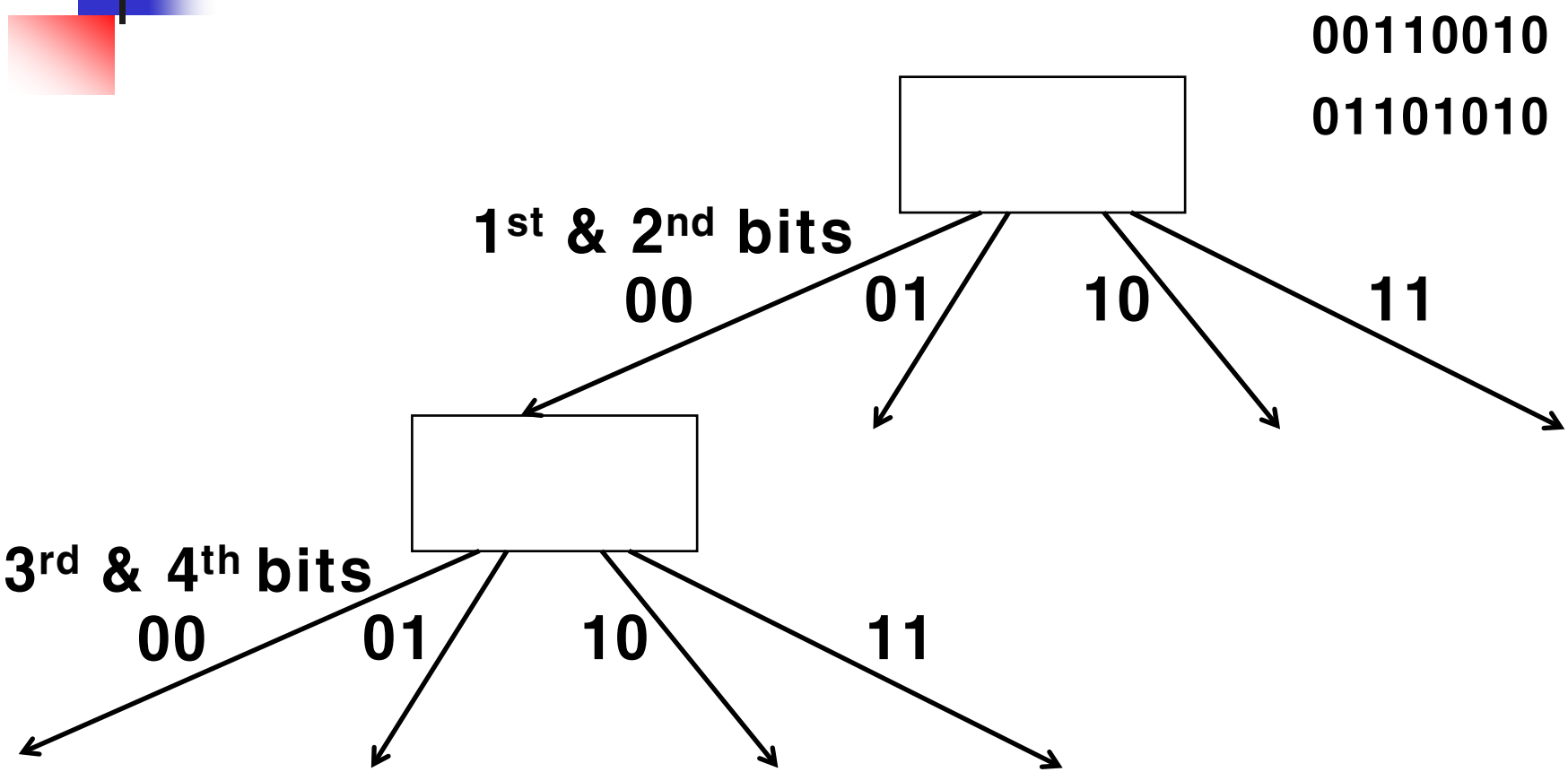


Multi-Way Tries

Multi-Way Trie

- Trie of degree m >= 2
- Tree height is reduced (vs. Binary trie)
- Branching is based on a portion of the key.
  - May be a bit combination or digit or an alphabet

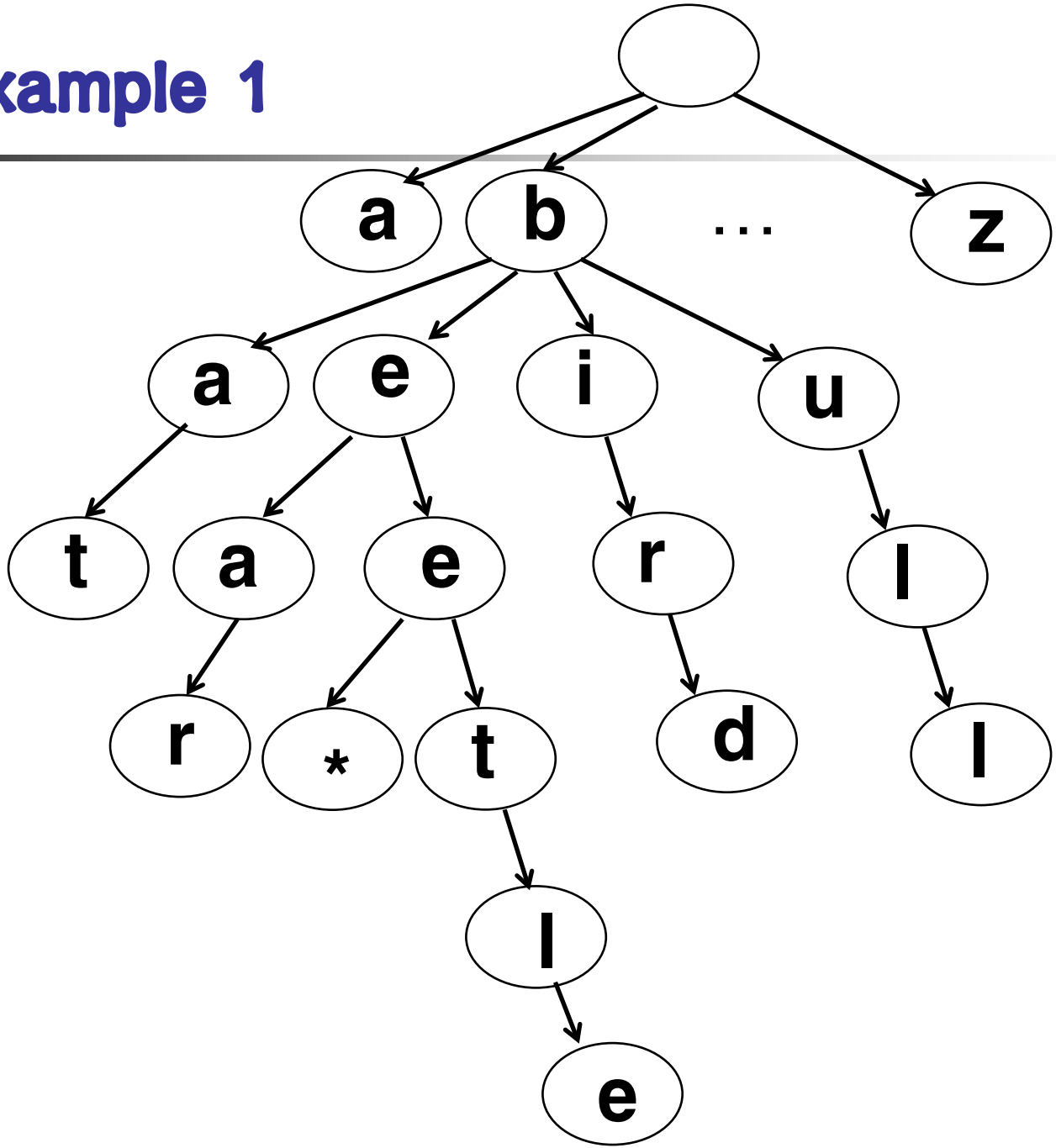
A 4-Way Trie


00110010
01101010
1st & 2nd bits
00
01
10
11
3rd & 4th bits
00
01
10
11

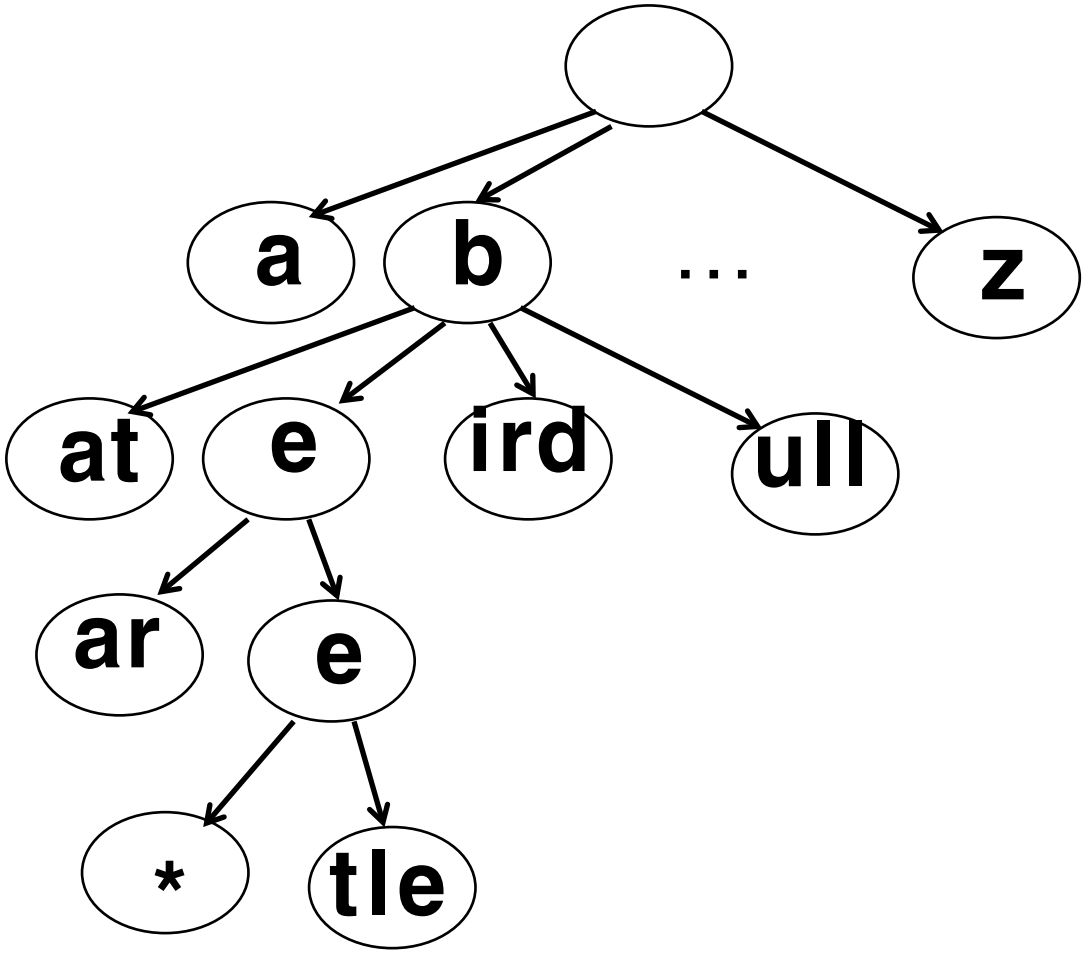
Multi-Way Trie: Use Example

- Stores a string over the alphabet across a chain of nodes.
- Usage
  - Storing a dictionary of words
  - Auto-complete dictionary, spell-checking and hyphenation software
- Keys at any level are the alphabet.
  - M = 26 + 1
- Keys are not stored in the branch nodes.
- Final portion of the keys are stored in the leaf nodes.

Example 1

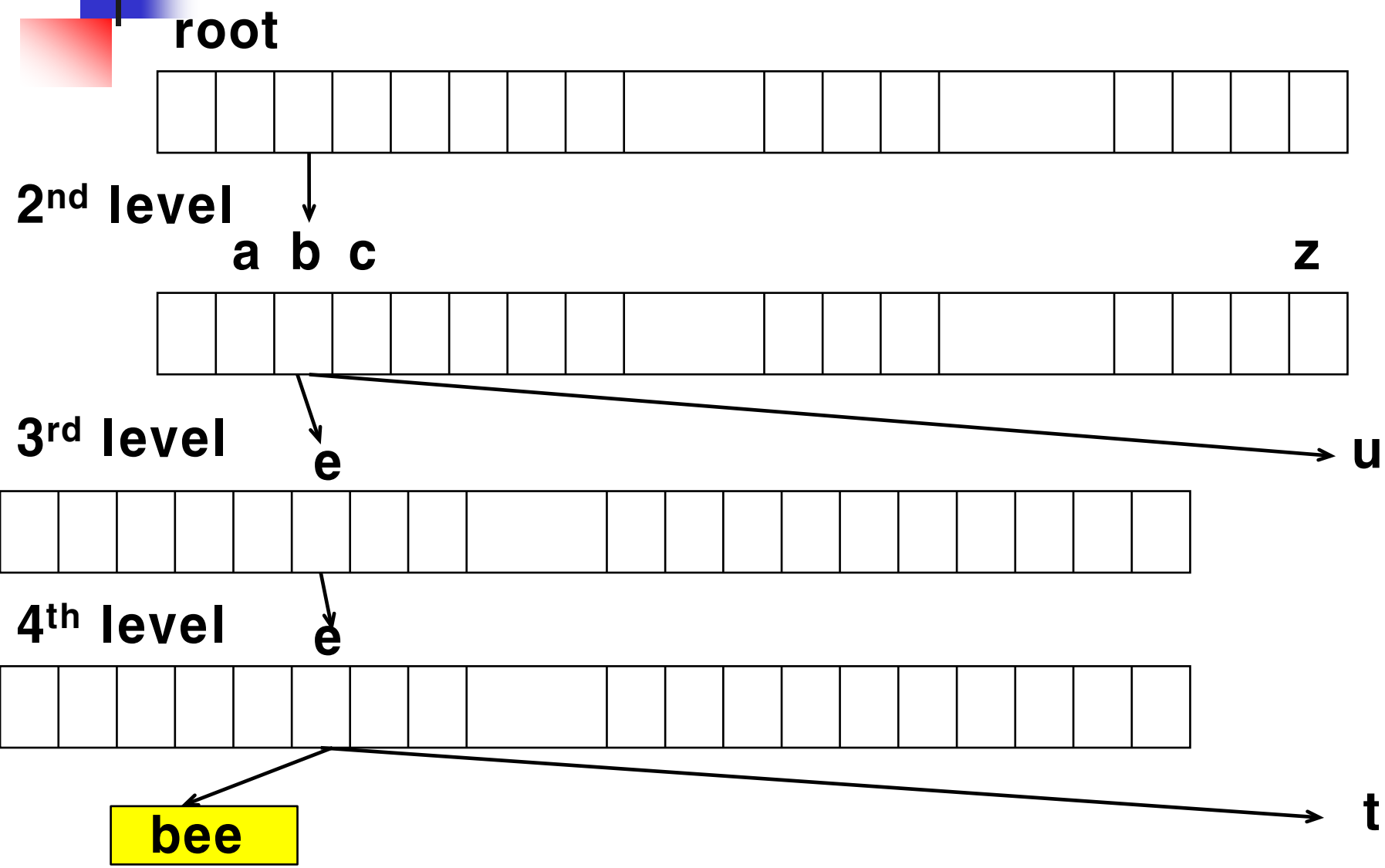


Example 2



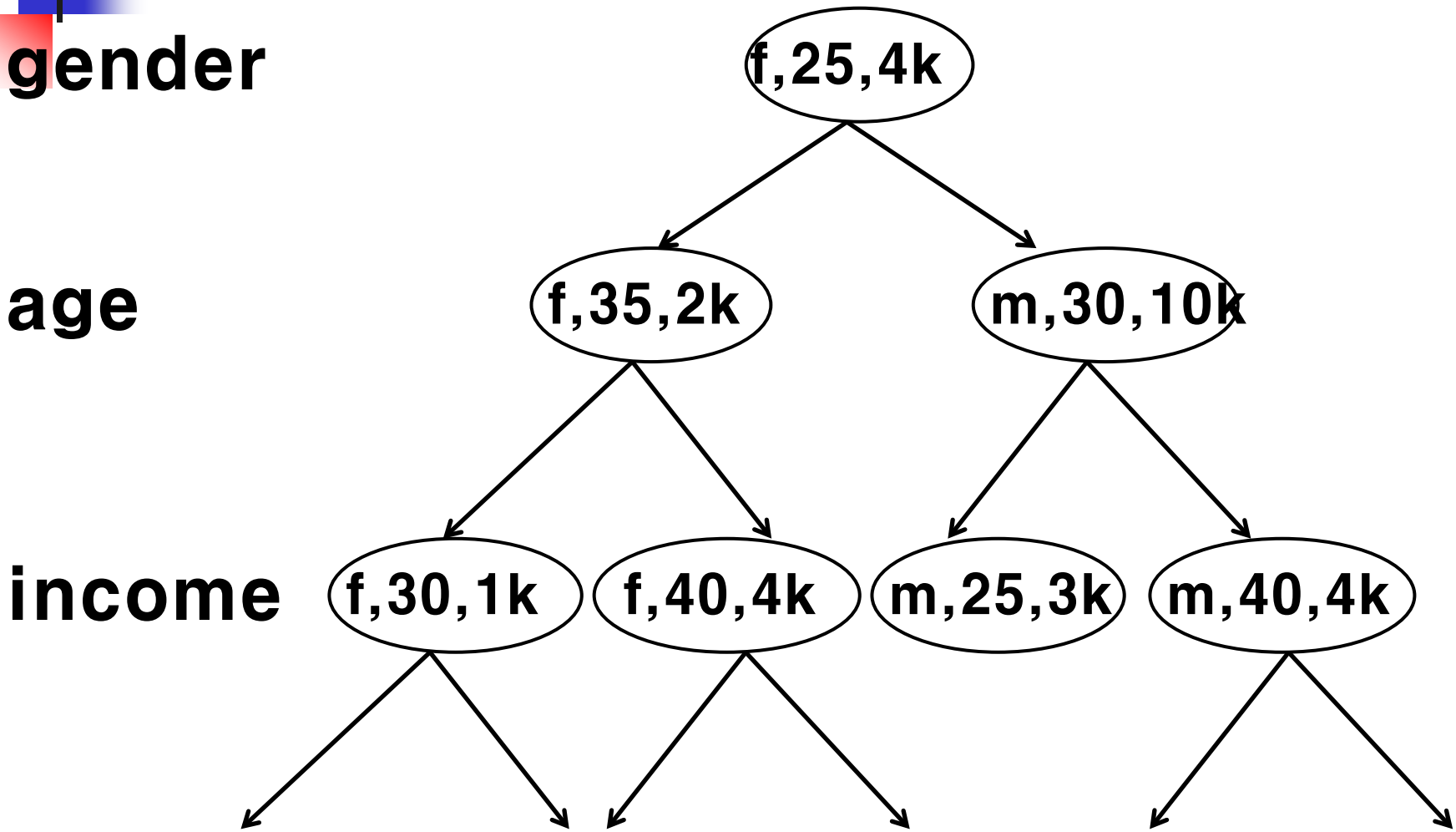
Implementation

bee beetle

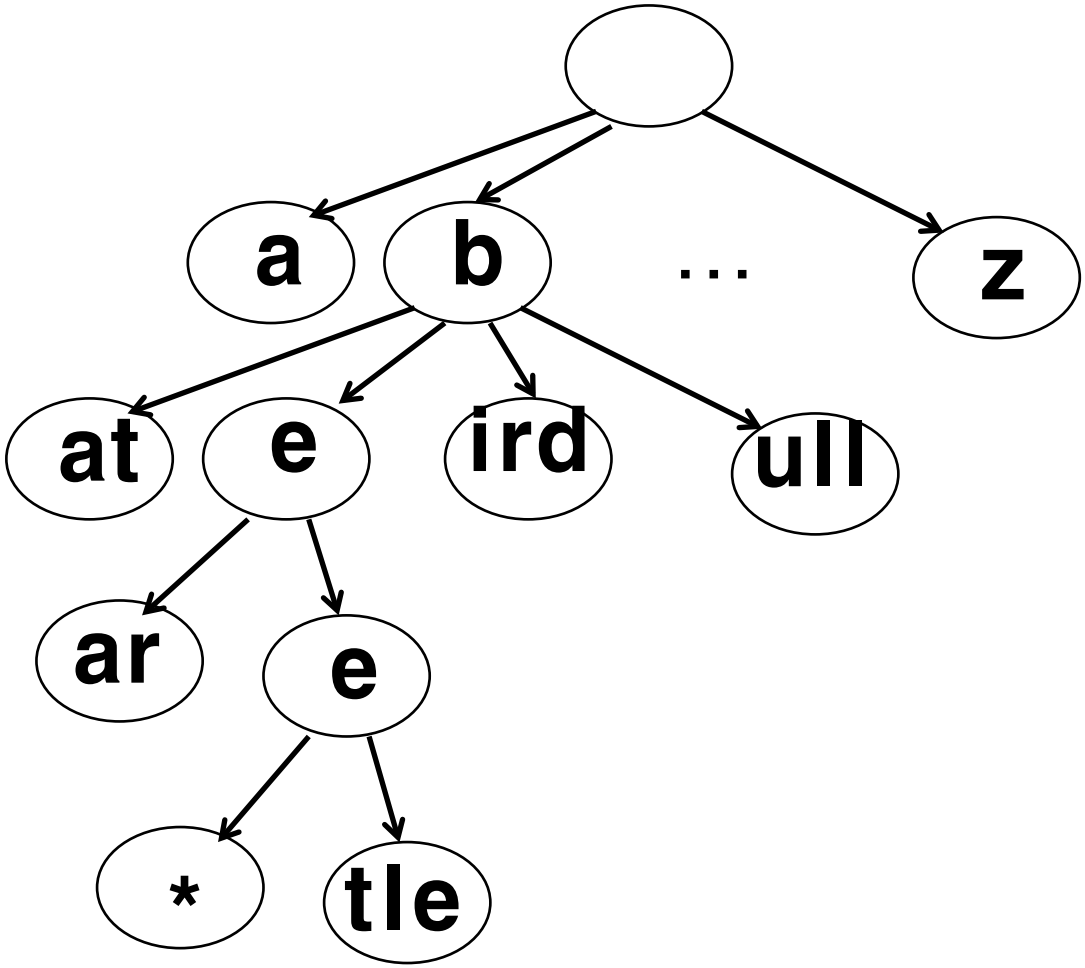


Data Structures for Partial Key Search

k-d Tree: Partial Key Search



M-Way Trie



Reprised

- Binary Search Trees
  - AVL tree, binary trie
  - k-d tree, T-tree
- Multi-Way Search Trees
  - m-way trie, quad tree, oct tree
- Height-Balanced Search Trees
  - AVL tree, T-tree, red-black tree
- Perfectly Height-Balanced Search Trees
  - 2-3 tree, (B tree, R-tree)
- Spatial Search Trees
  - quad tree, oct tree, k-d tree
- Trees that Support Partial Key Searches
  - k-d trees, tries (binary, m-way)

Search Performance: Best, Avg, Worst (1)

- Array
  - O(1), O(n), O(n)
- Binary Search
  - O(1), O($\log_2$ n), O($\log_2$ n)
- Binary Search Tree
  - O(1), O($\log_2$ n), O(n)
- AVL, Red-Black (Binary Search) Tree
  - O(1), O($\log_2$ n), O($\log_2$ n)
- Quad Tree
  - O(1), O($\log_4$ n), O(n)
- k-d Tree
  - O(1), O($\log_2$ n), O(n)
- T-Tree (with m-element array)
  - O(1), O($\log_2$ n/m), O($\log_2$ n/m)

Search Performance: Best, Avg, Worst (2)

- Binary Trie (with k-bit keys)
  - O(1), O(k), O(k)
- M-Way Trie (with k-digit keys)
  - O(1), O(k), O(k)

End of Lecture