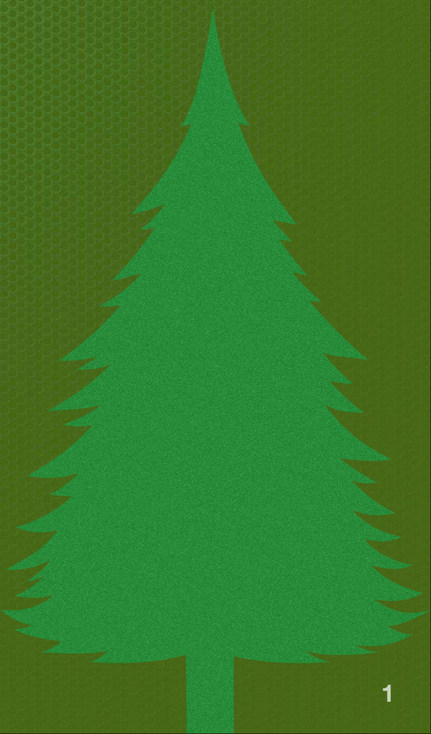
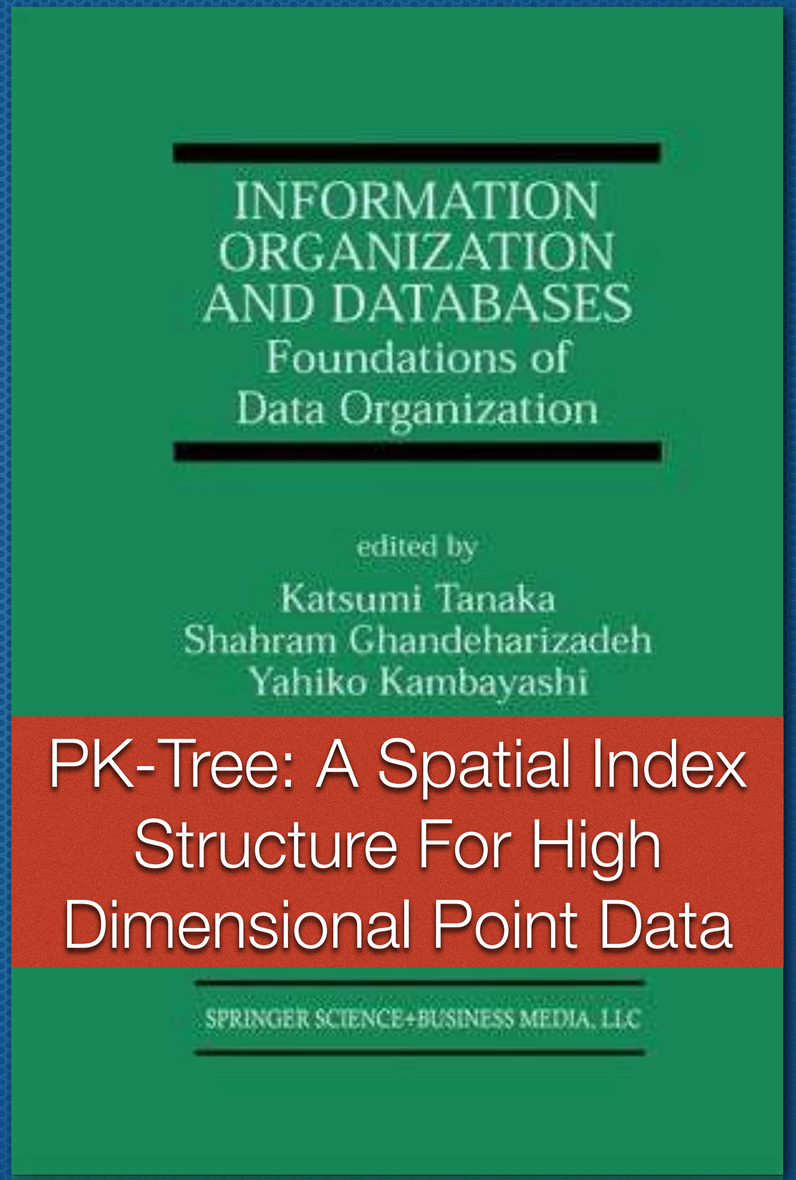
PK-Tree
(Pyramid K-instantiable Tree)

Israel Chaparro



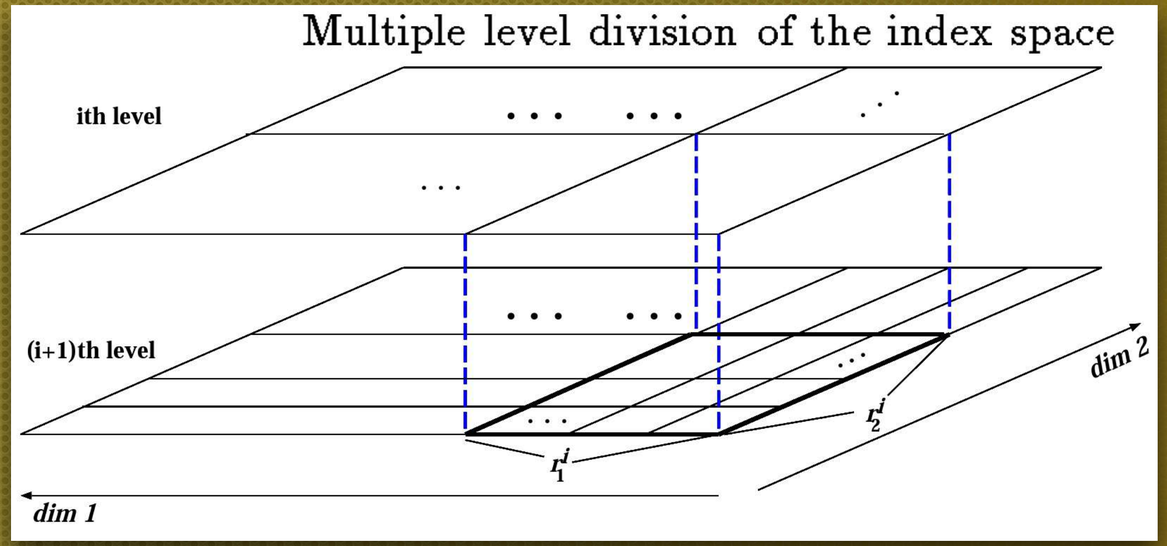
Introduction

- Wang, Yang, and Muntz (1998).
- Based on the application of a **grouping technique** "similar to bucketing".
- The key principle in the PK-tree is the use of a parameter k (k-instantiation) to stipulate the **minimum** number of objects or nodes that are grouped to form a node.



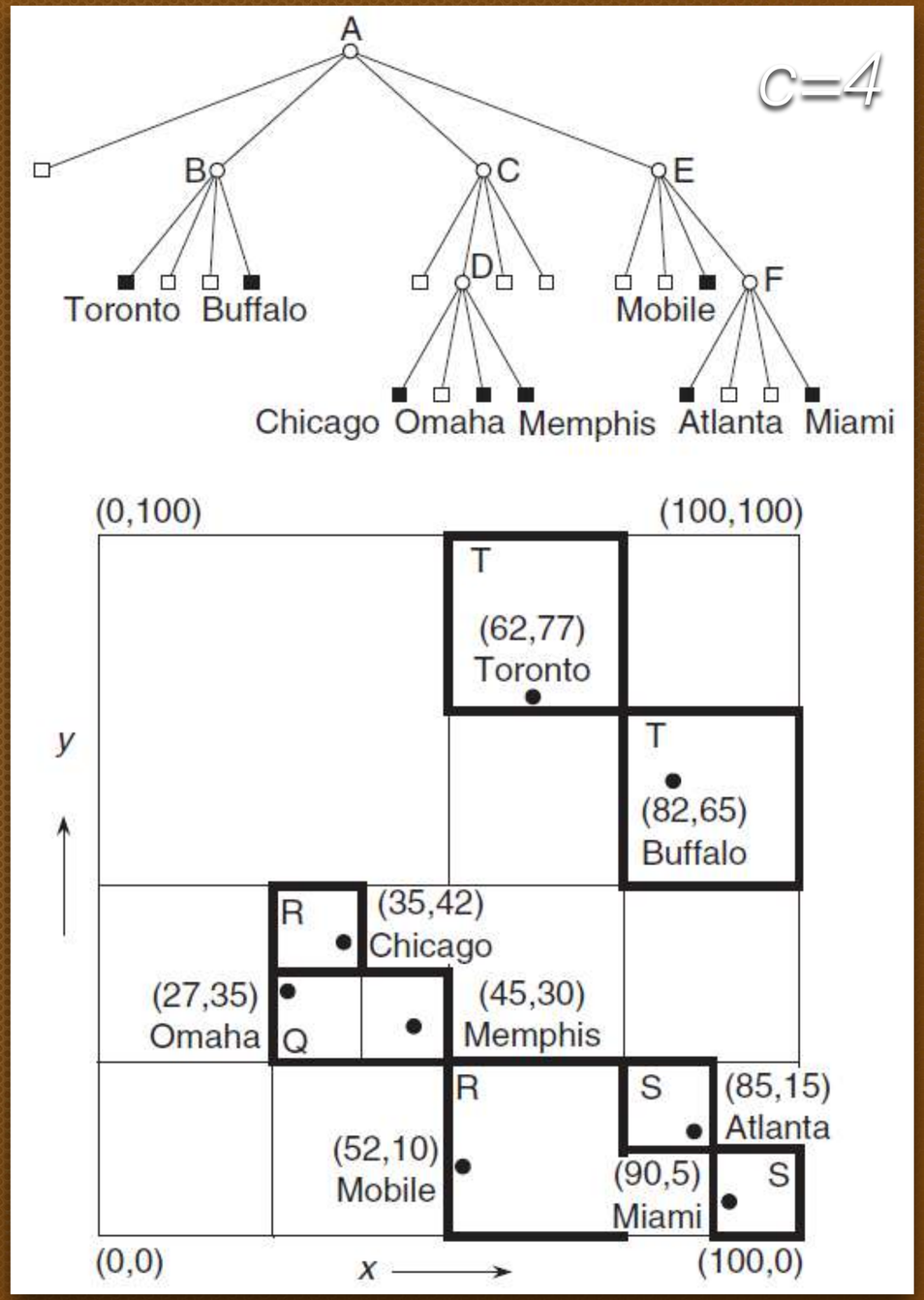
But currently...

- Performance for **highdimensional** data is often **unsatisfactory**.
- PK-tree is **little known**.
- Is **useful for lowdimensional data** like that found in applications such as **spatial** databases and **geographic** information systems (GIS).



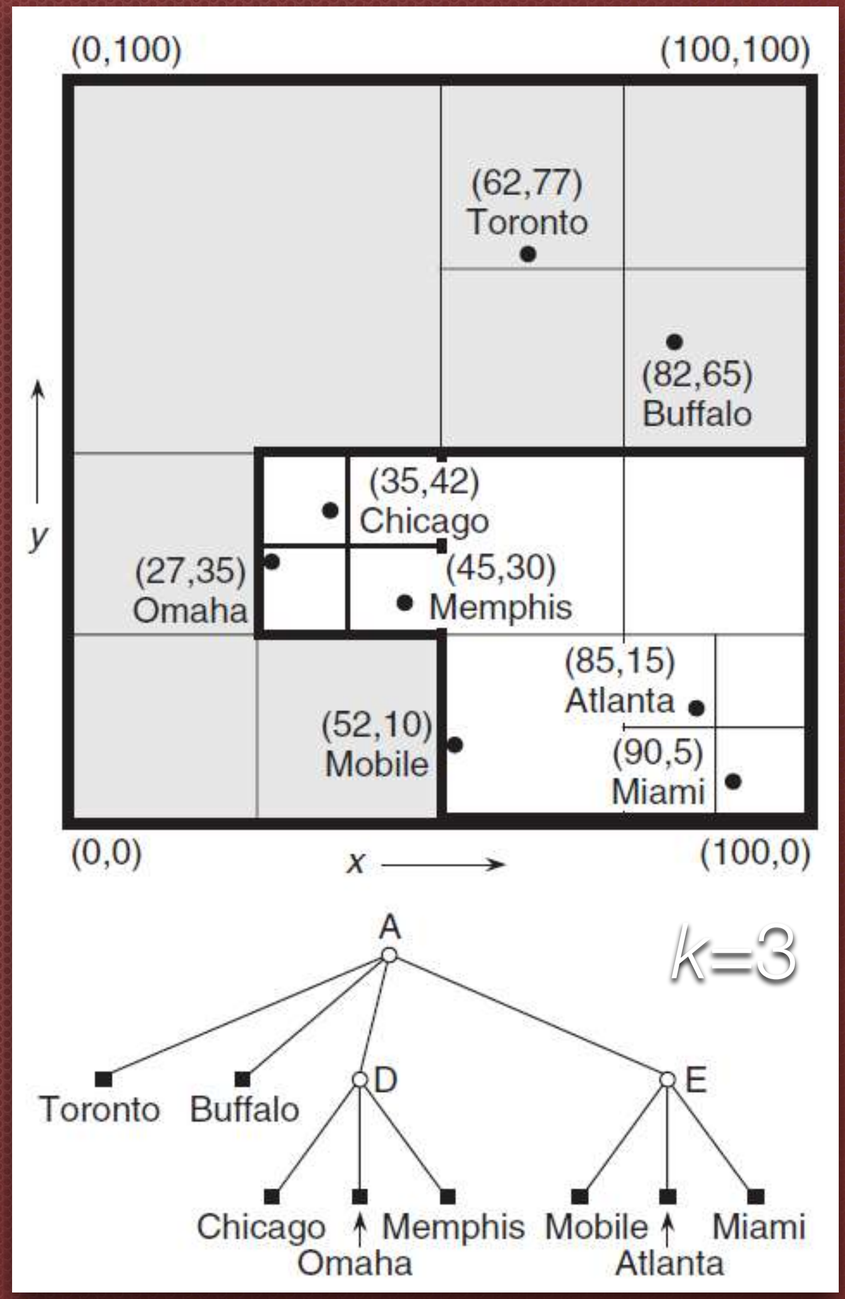
Recall:

- In a bucket methods, we **recursively** decompose the underlying space into **disjoint not identically sized** until the number of items in each block is **less than or equal to the bucket capacity**.
- Pk-tree **only instantiates non-leaf cells** with at least a certain number of nonempty children.



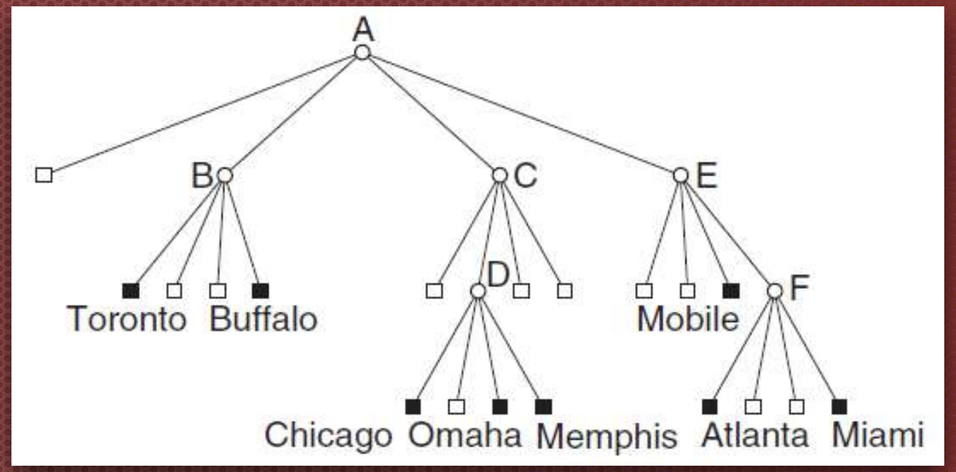
PK-Tree construction

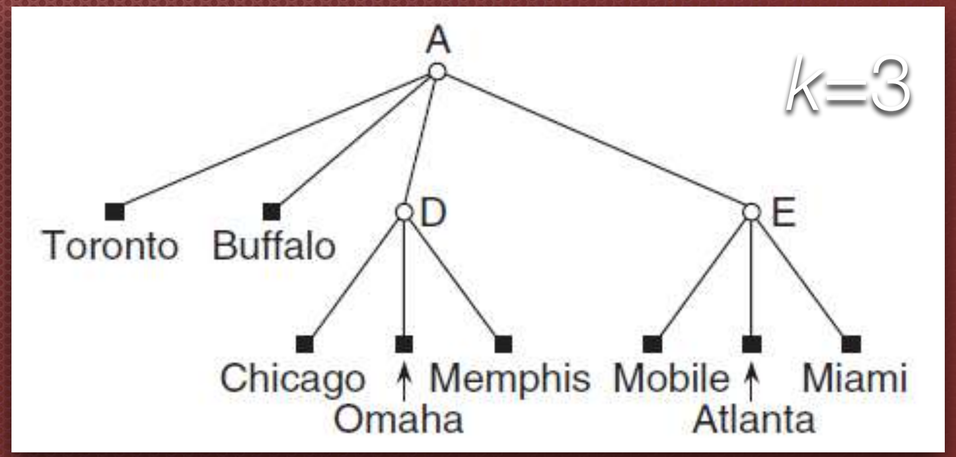
- 1. Creating a **PK-tree node for each node in the partition tree** (nodes and points).
- 2. **Grouping** the nodes in **bottom-up**, using the "**k-instantiation**" with ***k***.
- 3. At the deepest level, we **eliminate all point nodes** whose blocks **do not contain a point** and remove them from their parents.
- 4. For a directory node **a**, if **a** has less than **k** child nodes (nonempty), then we remove **a** and link its children to the parent of **a**. This process is terminated at the root.



PK-Tree construction

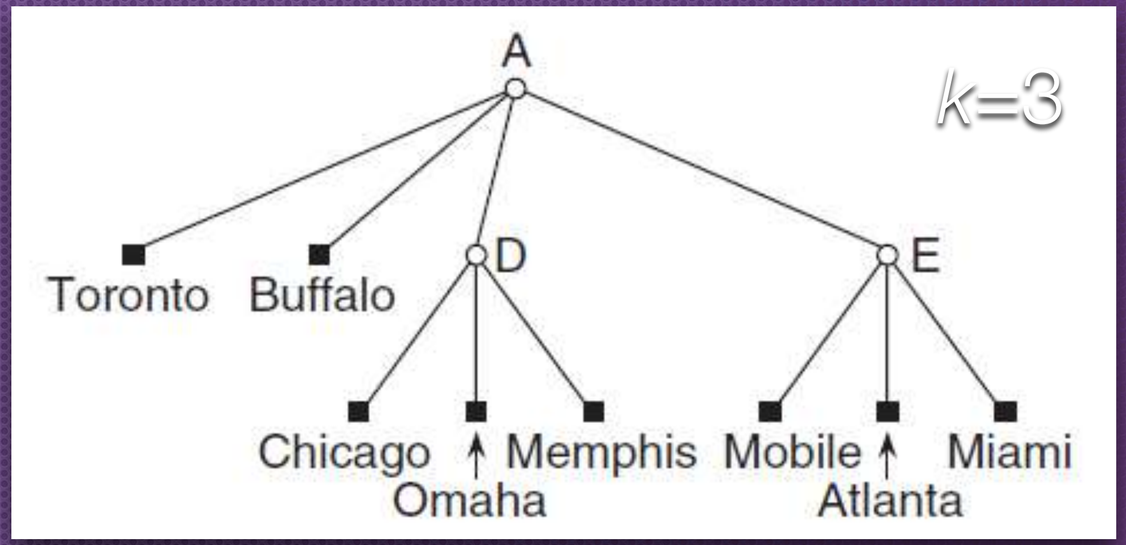
- 1. Creating a **PK-tree node for each node in the partition tree** (nodes and points).
- 2. **Grouping** the nodes in **bottom-up**, using the "**k-instantiation**" with ***k***.
- 3. At the deepest level, we **eliminate all point nodes** whose blocks **do not contain a point** and remove them from their parents.
- 4. Remove a directory node **a**, if has less than **k** child nodes and link its children to the parent of **a**. This process is terminated at the root.





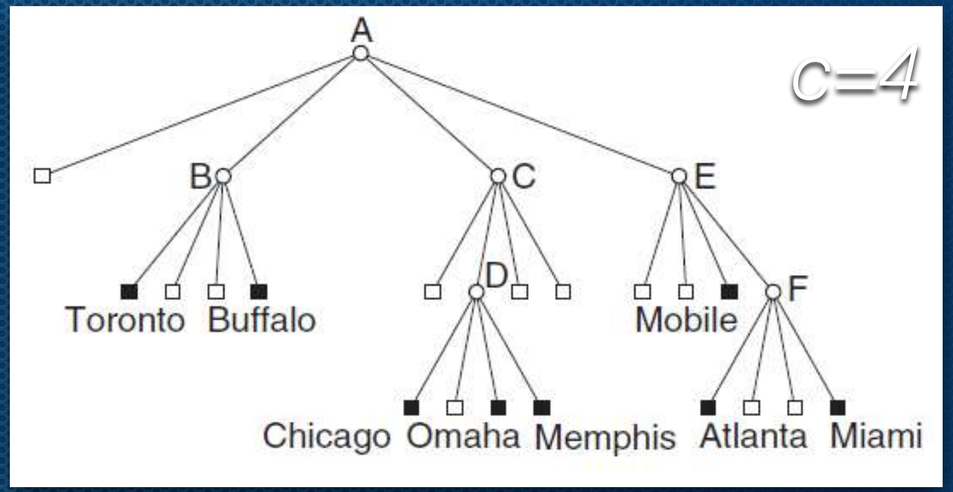
Definition

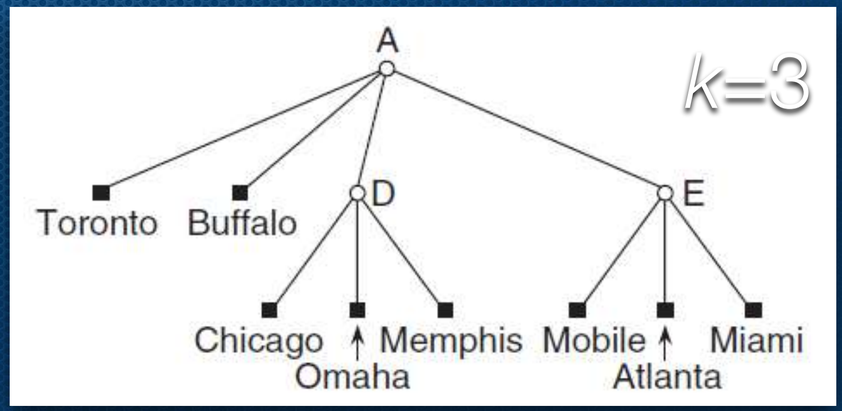
- 1. A **nonempty leaf** node in the partition tree is **k-instantiated**.
- 2. A nonleaf node **a** in the partition tree is k-instantiated if **a** directly contains **at least k** k-instantiated nodes or blocks.
- 3. The **root** node of the partition tree is **k-instantiated**.



Bucketing vs K-instantiation

Bucketing	K-instantiation
Different Concepts.	Similar Effects.
Maximum	Minimun Number of Children
Top-Down Process	Botton-Up Process.
Balanced (B+-Tree or R-Tree)	Not necessarily balanced.





Operations

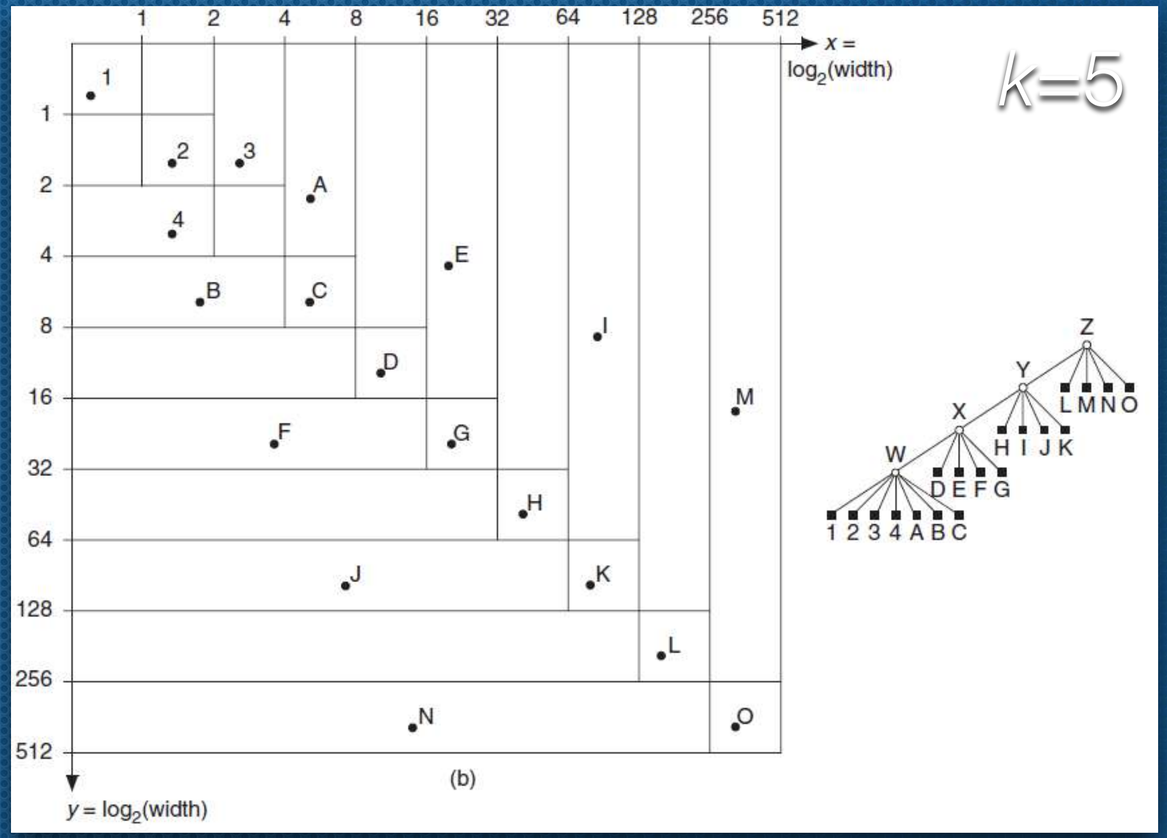
- Add or remove is a two-step process.
  - 1. Search for p:
    - Top-down manner.
    - Check if is possible perform the operation.
  - 2. Grouping process:
    - Button-up manner, to keep the PK-Tree definition.

Representation and Searching

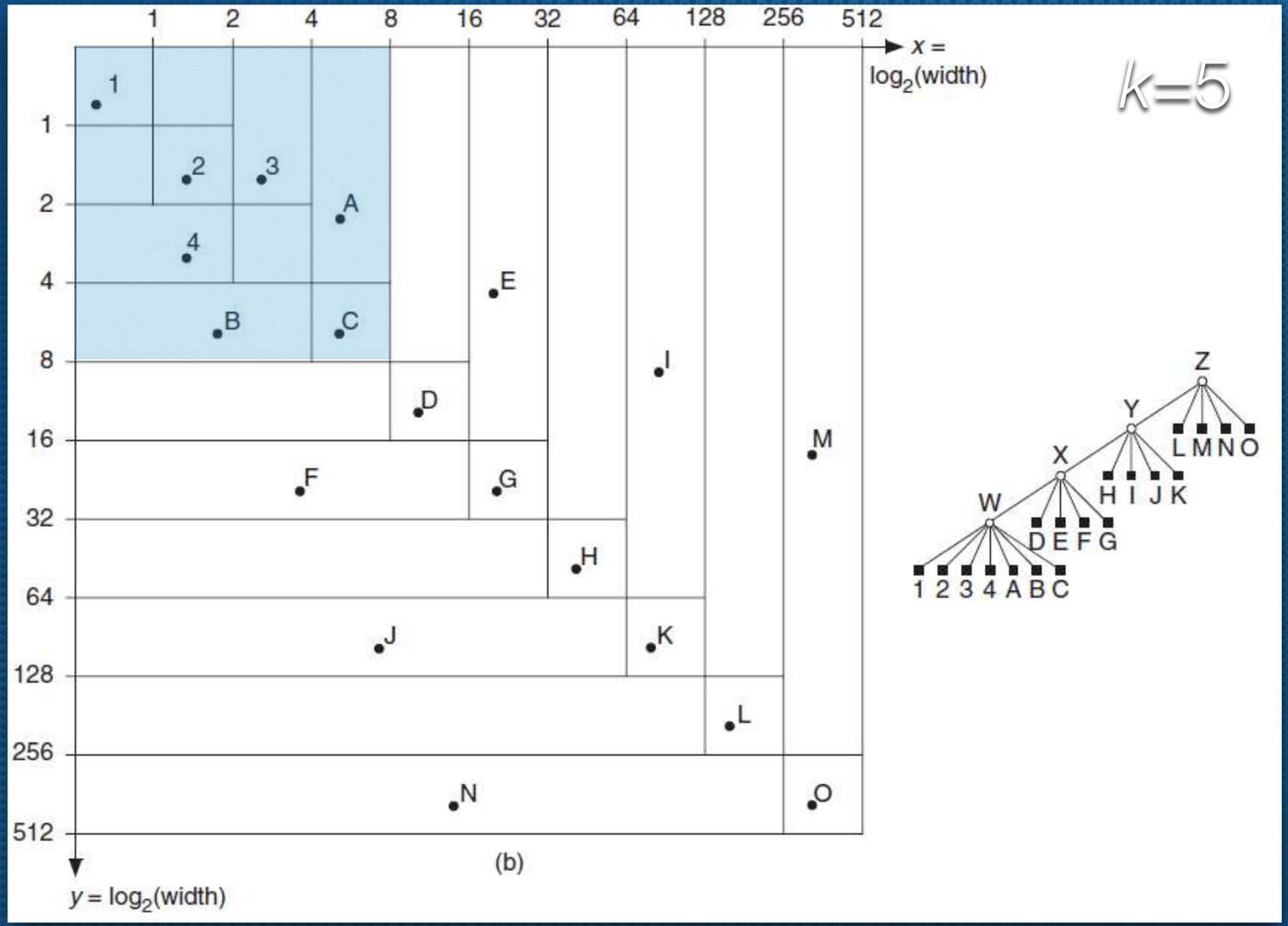
- The process is dependent on the **representation** of the points and the nodes.
- If two points fall into the same 1x1 cell, then they have the **same locational code**.

Insertion

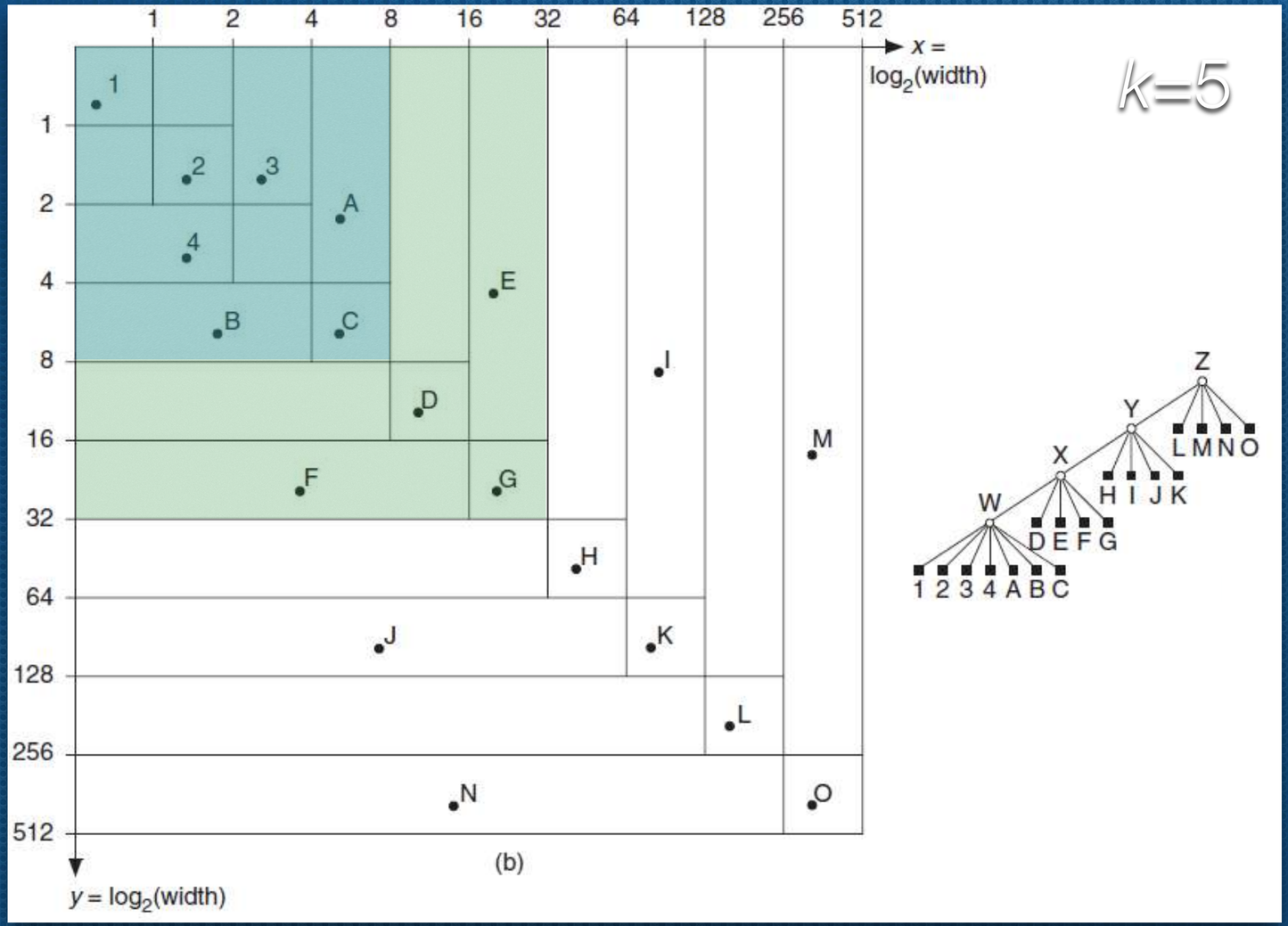
- Not need rebuild the structure from scratch.
- Once the position has been found, we continuosly check in a **bottom-up** manner for the applicability of **k-instantiation** until it is no longer applicable.



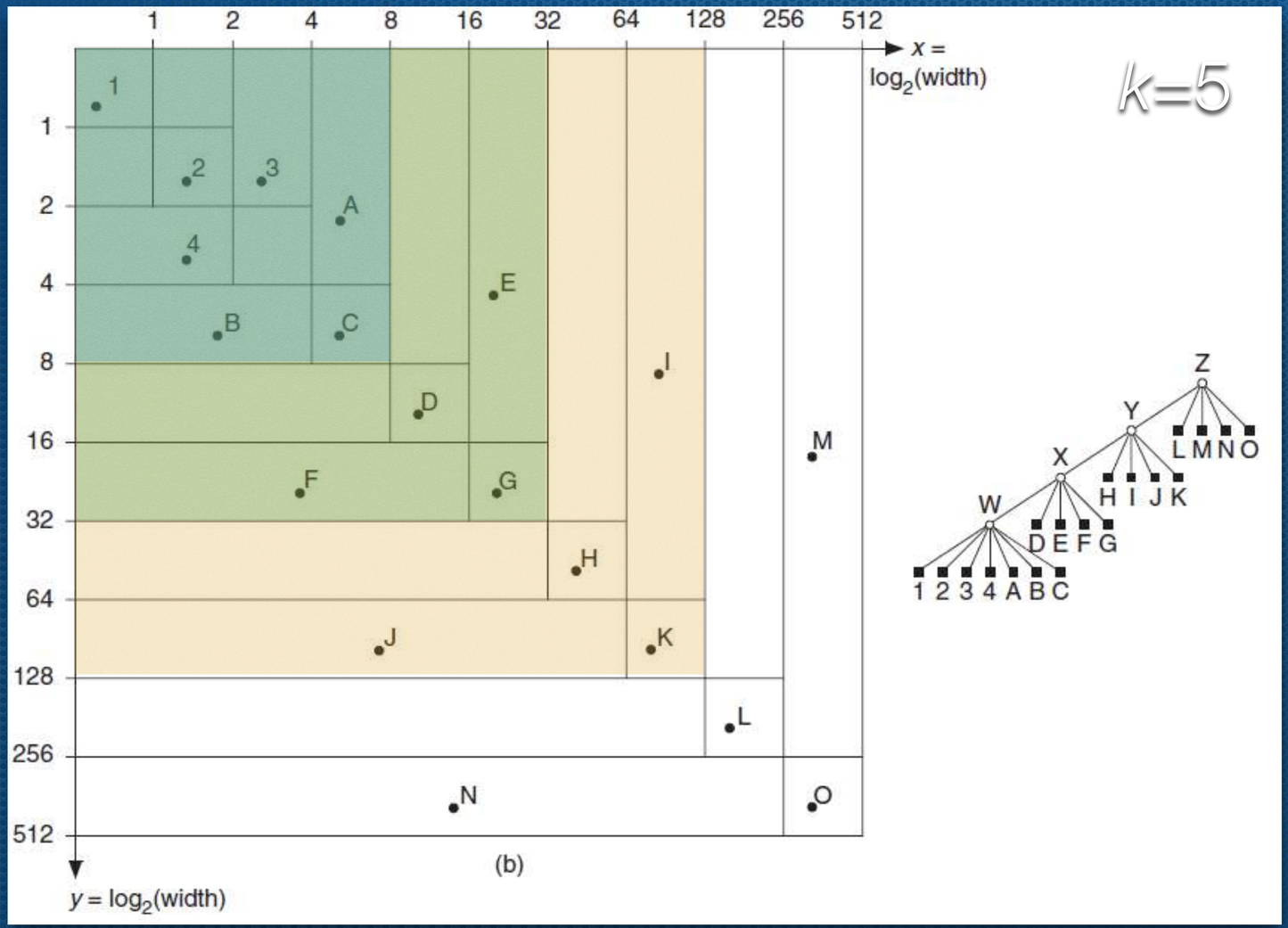
Insertion


1 2 4 8 16 32 64 128 256 512
x = log2(width)
k=5
1 2 4 8 16 32 64 128 256 512
y = log2(width)
(b)
1 2 3 4 A B C D E F G H I J K L M N O
W X Y Z

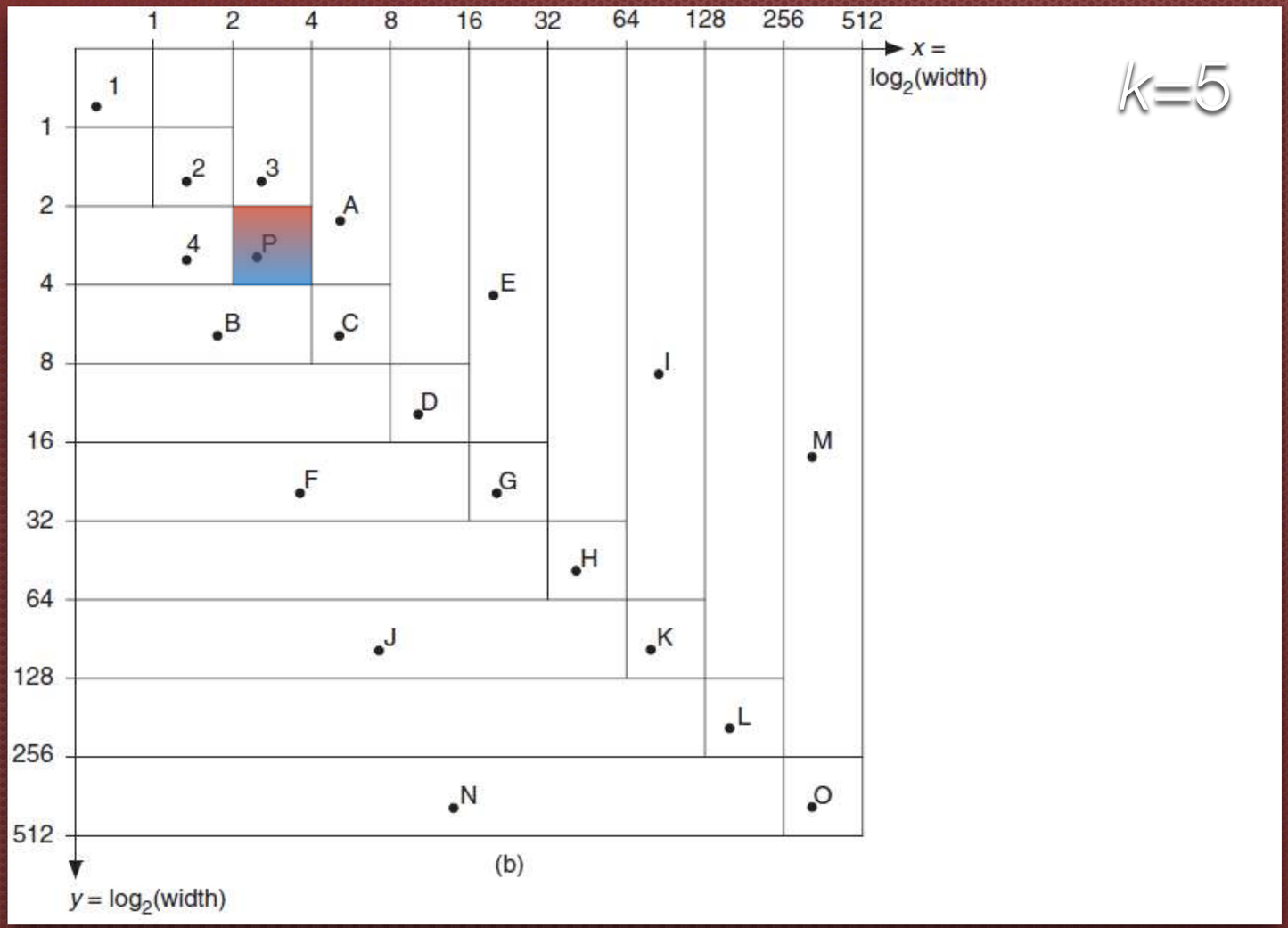
Insertion


1 2 4 8 16 32 64 128 256 512
x = log2(width)
k=5
1 2 4 8 16 32 64 128 256 512
y = log2(width)
(b)
1 2 3 4 A B C D E F G H I J K L M N O
W X Y Z

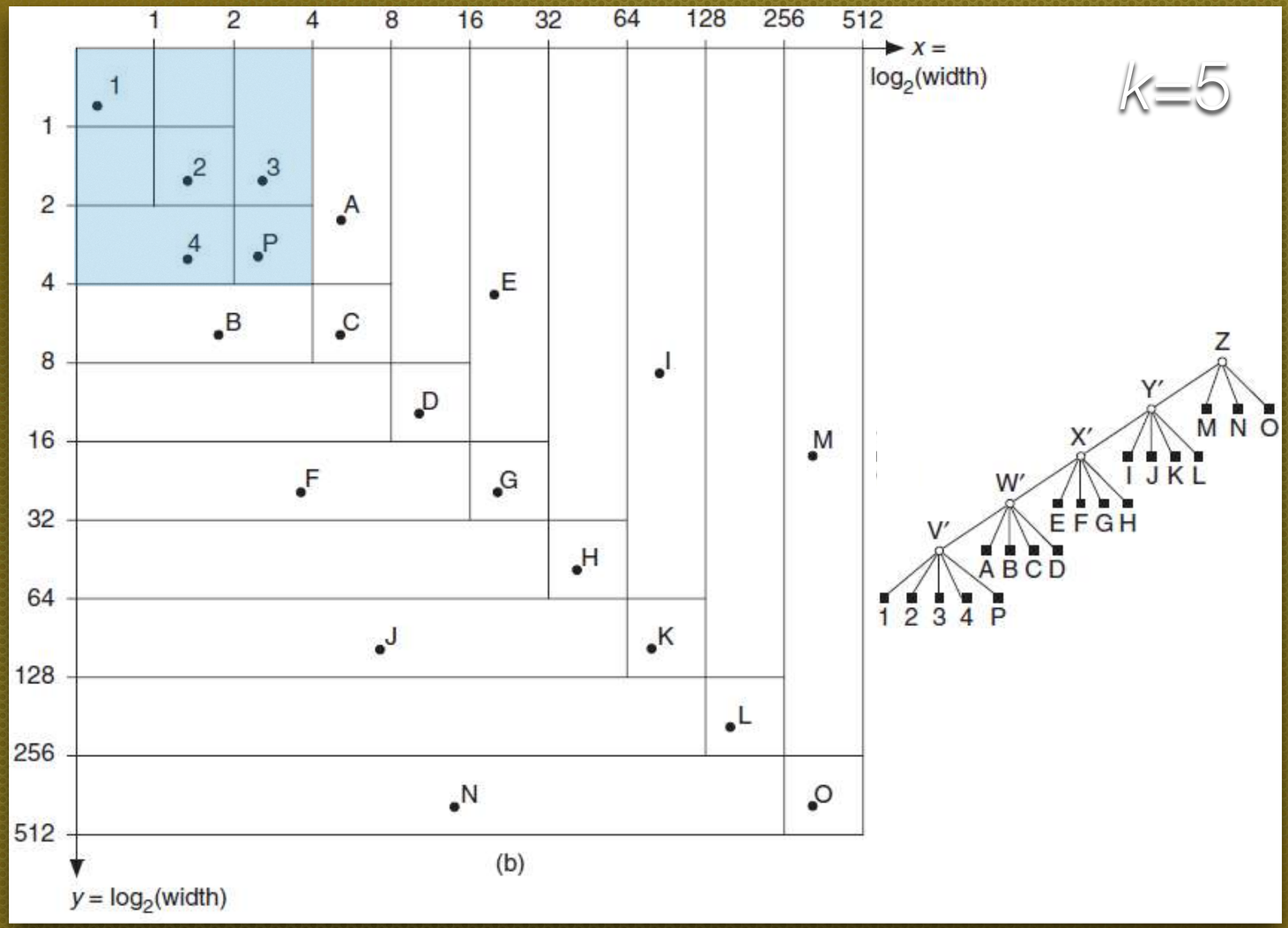
Insertion


1 2 4 8 16 32 64 128 256 512
x = log2(width)
k=5
1 2 4 8 16 32 64 128 256 512
y = log2(width)
(b)
1 2 3 4 A B C D E F G H I J K L M N O
W X Y Z

Insertion

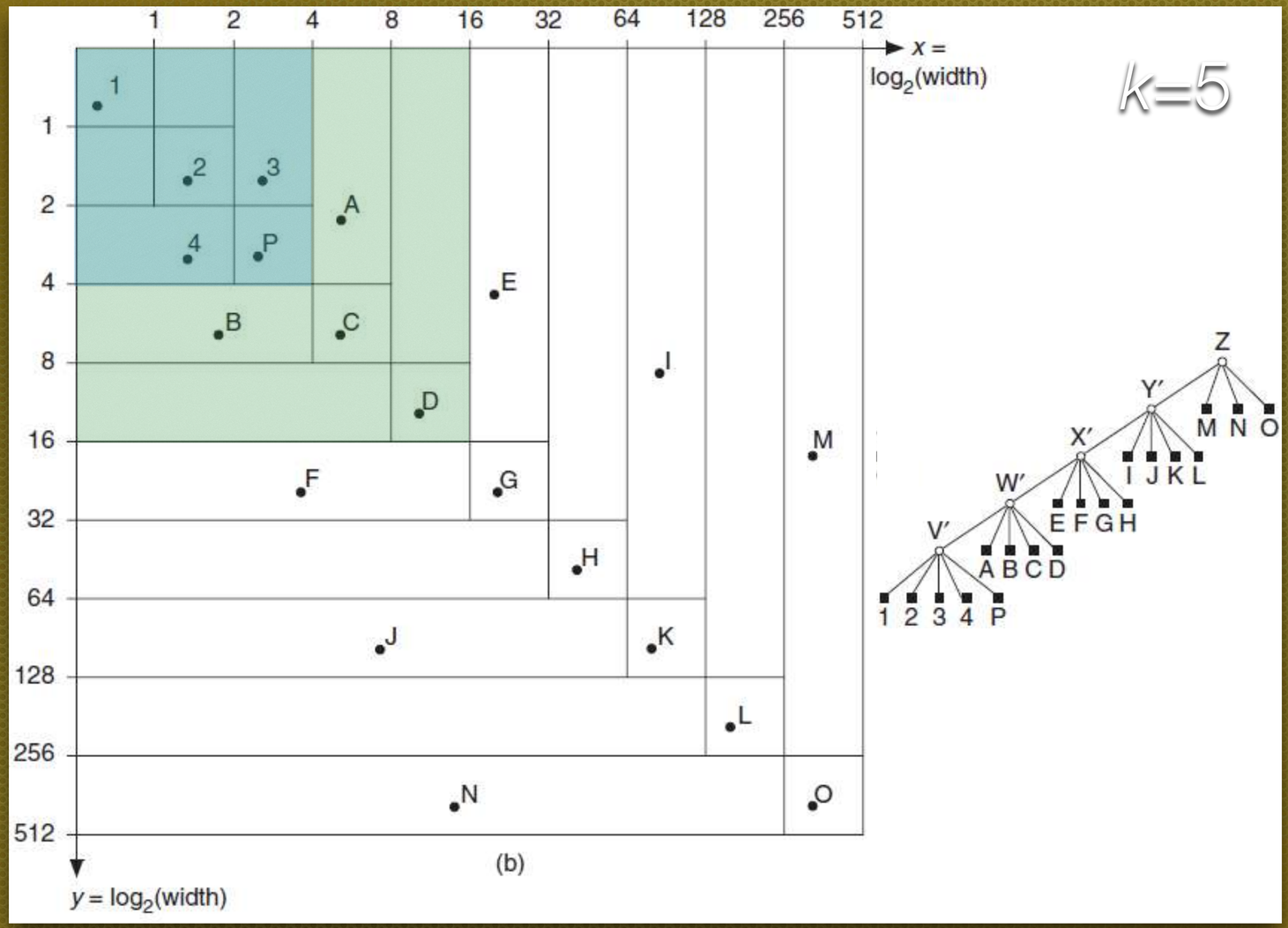

1
2
4
8
16
32
64
128
256
512
x = log2(width)
y = log2(width)
k=5
1
2
3
4
A
P
B
C
D
E
F
G
H
I
J
K
L
M
N
O
(b)

Insertion



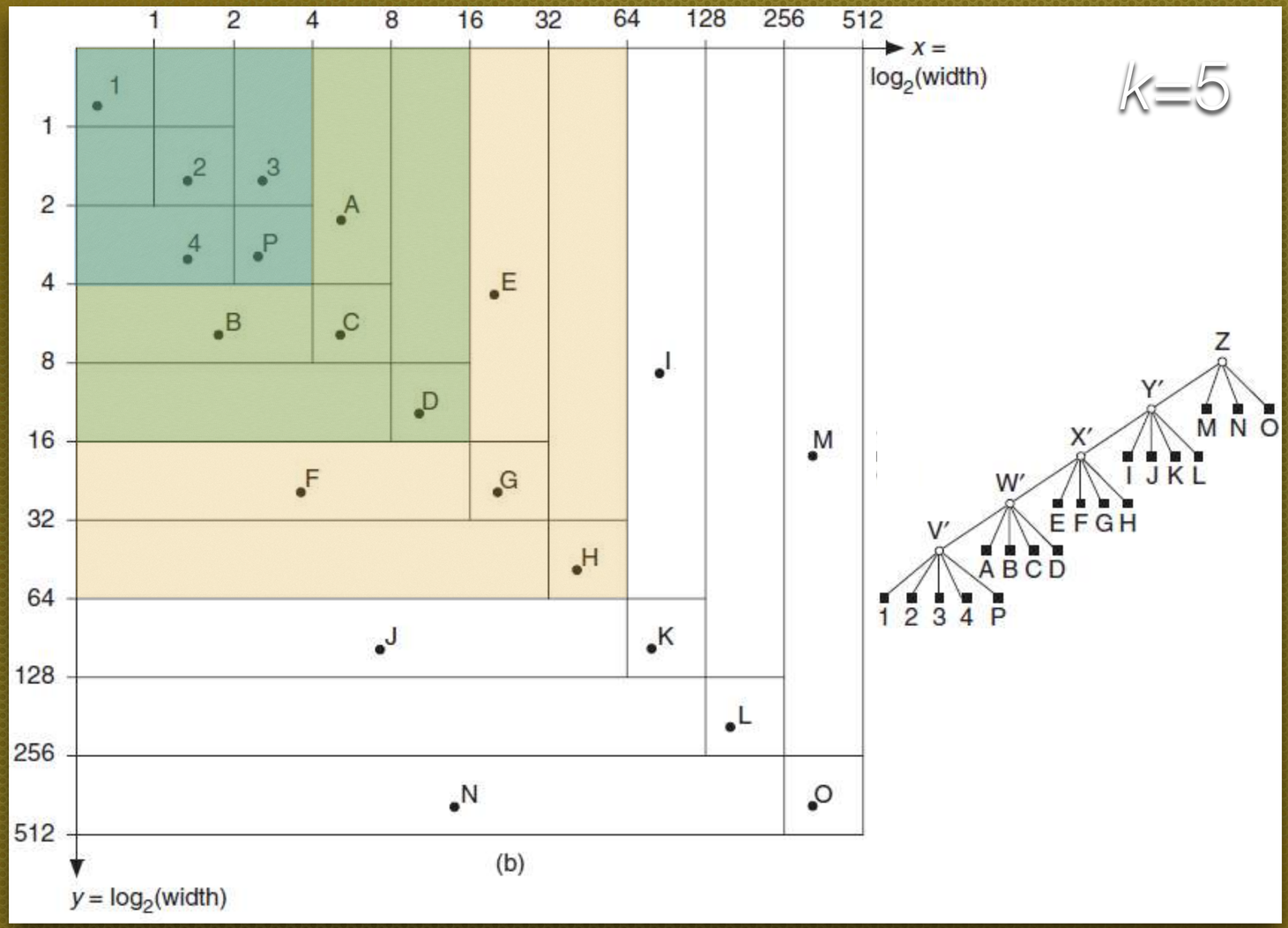
FORM-NODE-FOR-MINEN-CLOSING-BLOCK-WITH-K-ITEMS

Insertion



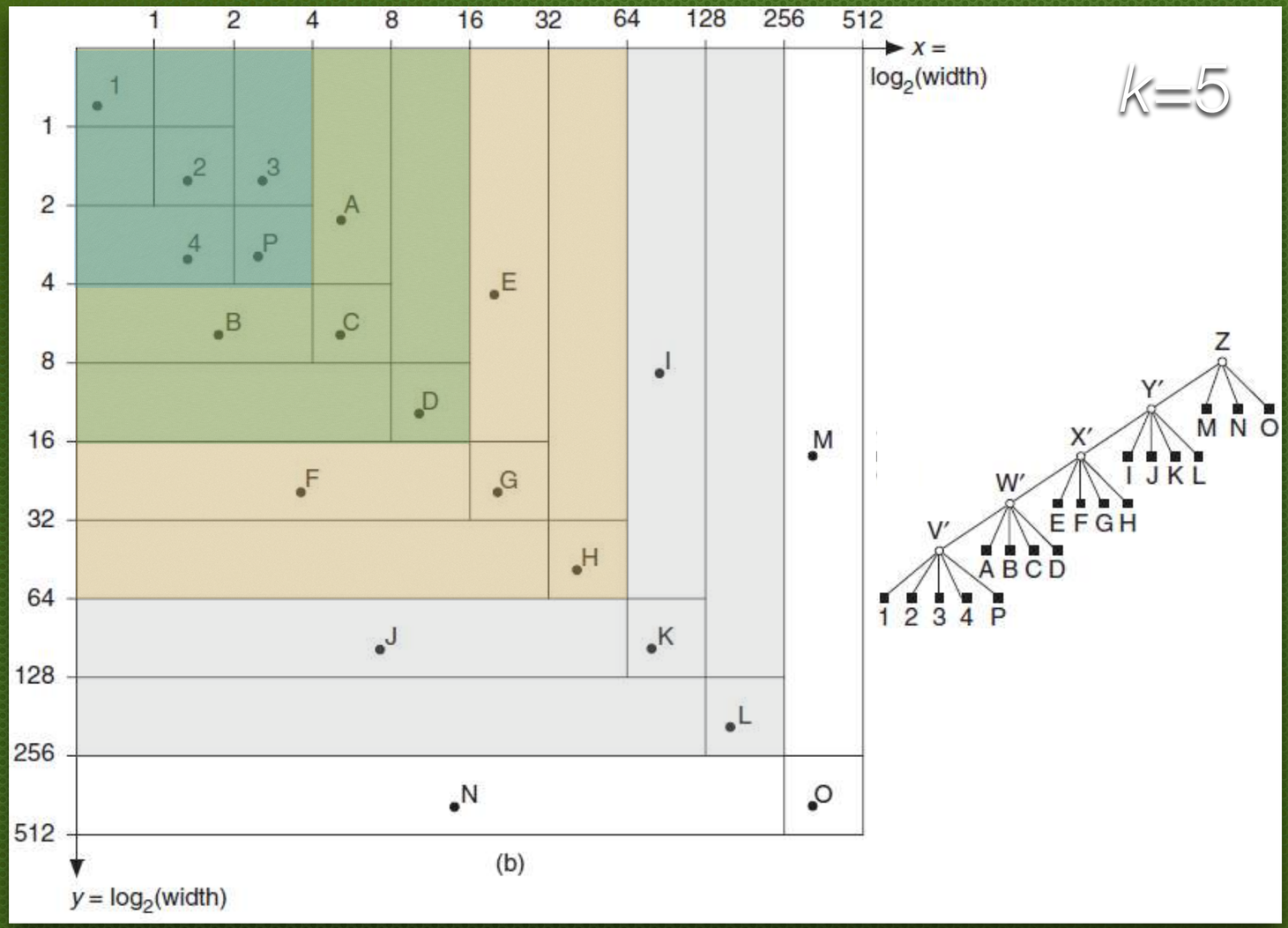
FORM-NODE-FOR-MINEN-CLOSING-BLOCK-WITH-K-ITEMS

Insertion

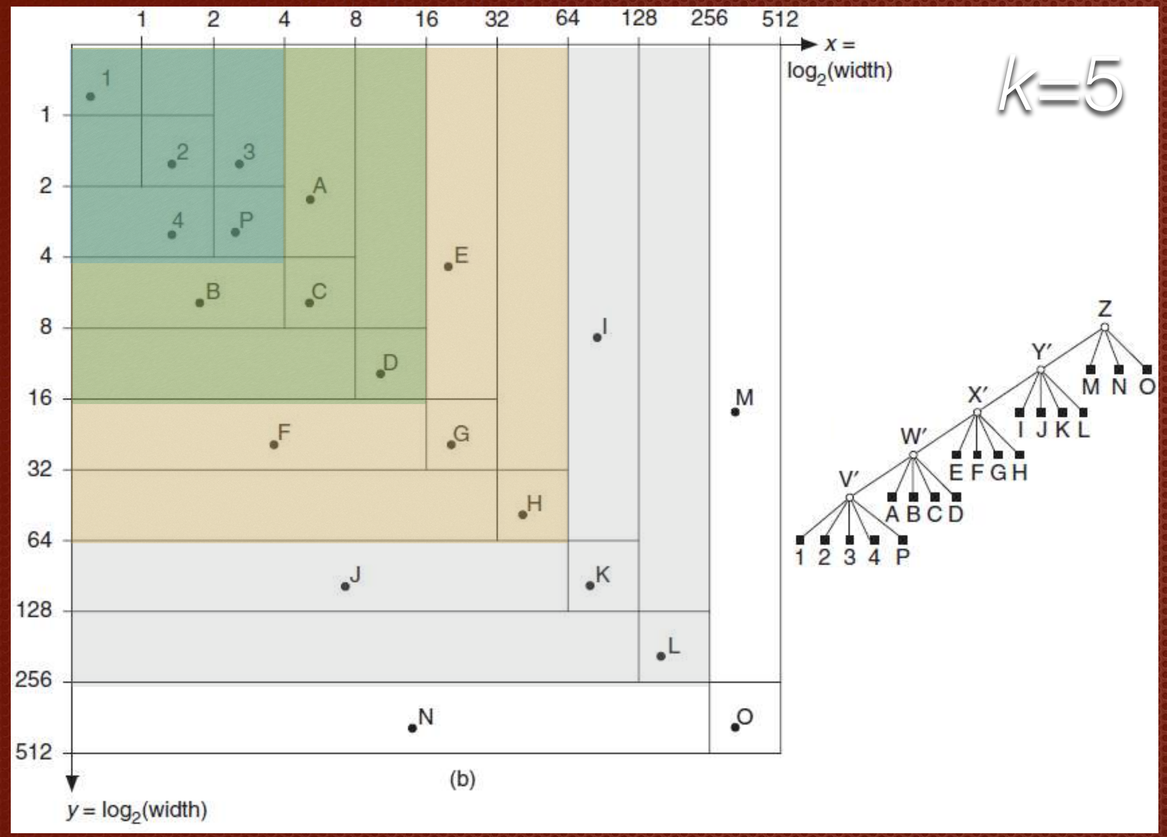


FORM-NODE-FOR-MINEN-CLOSING-BLOCK-WITH-K-ITEMS

Insertion

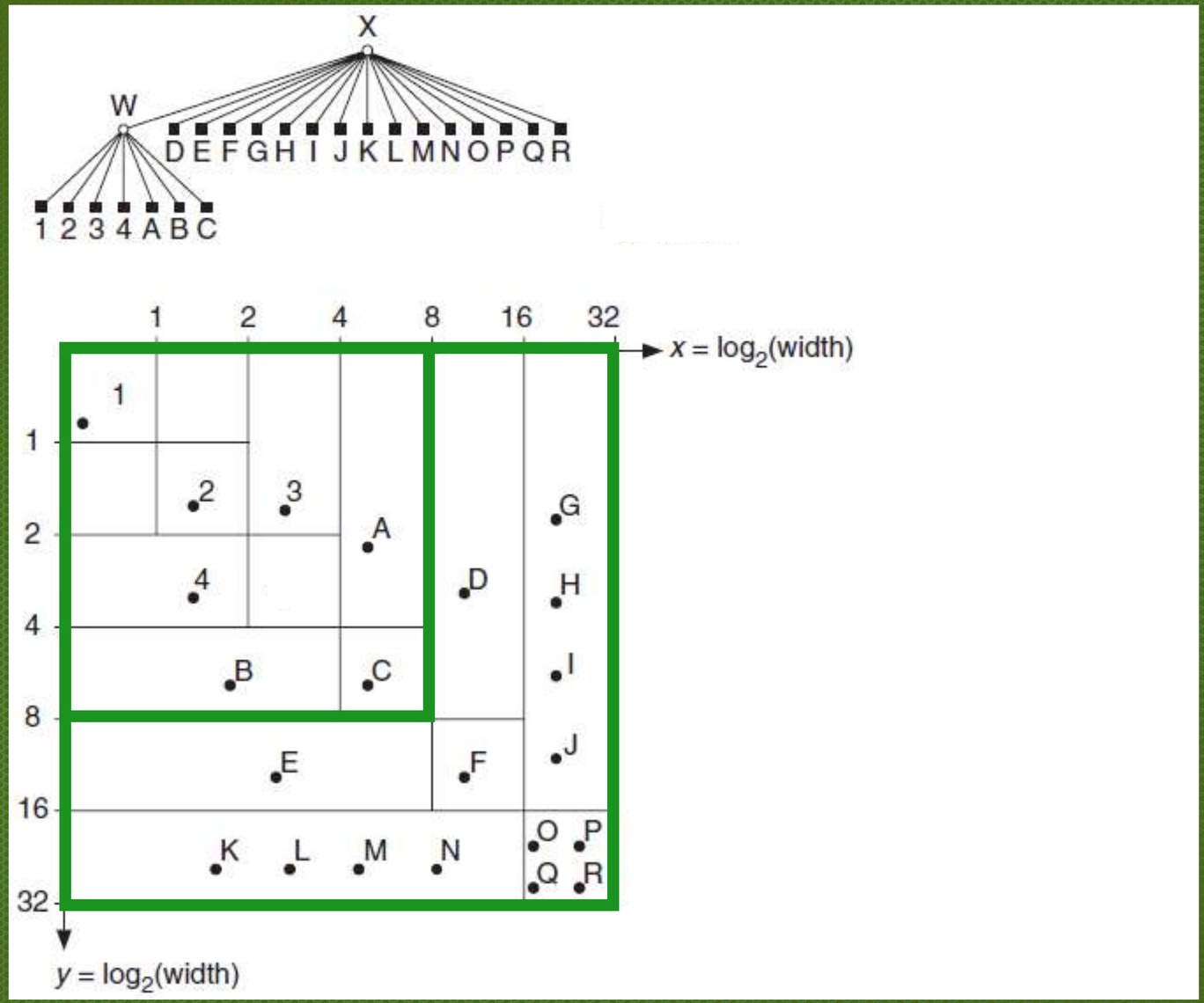

1 2 4 8 16 32 64 128 256 512
x = log2(width)
k=5
1 2 4 8 16 32 64 128 256 512
y = log2(width)
(b)
1 2 3 4 P A B C D E F G H I J K L M N O
V′ W′ X′ Y′ Z

PKinsert Pseudocode:



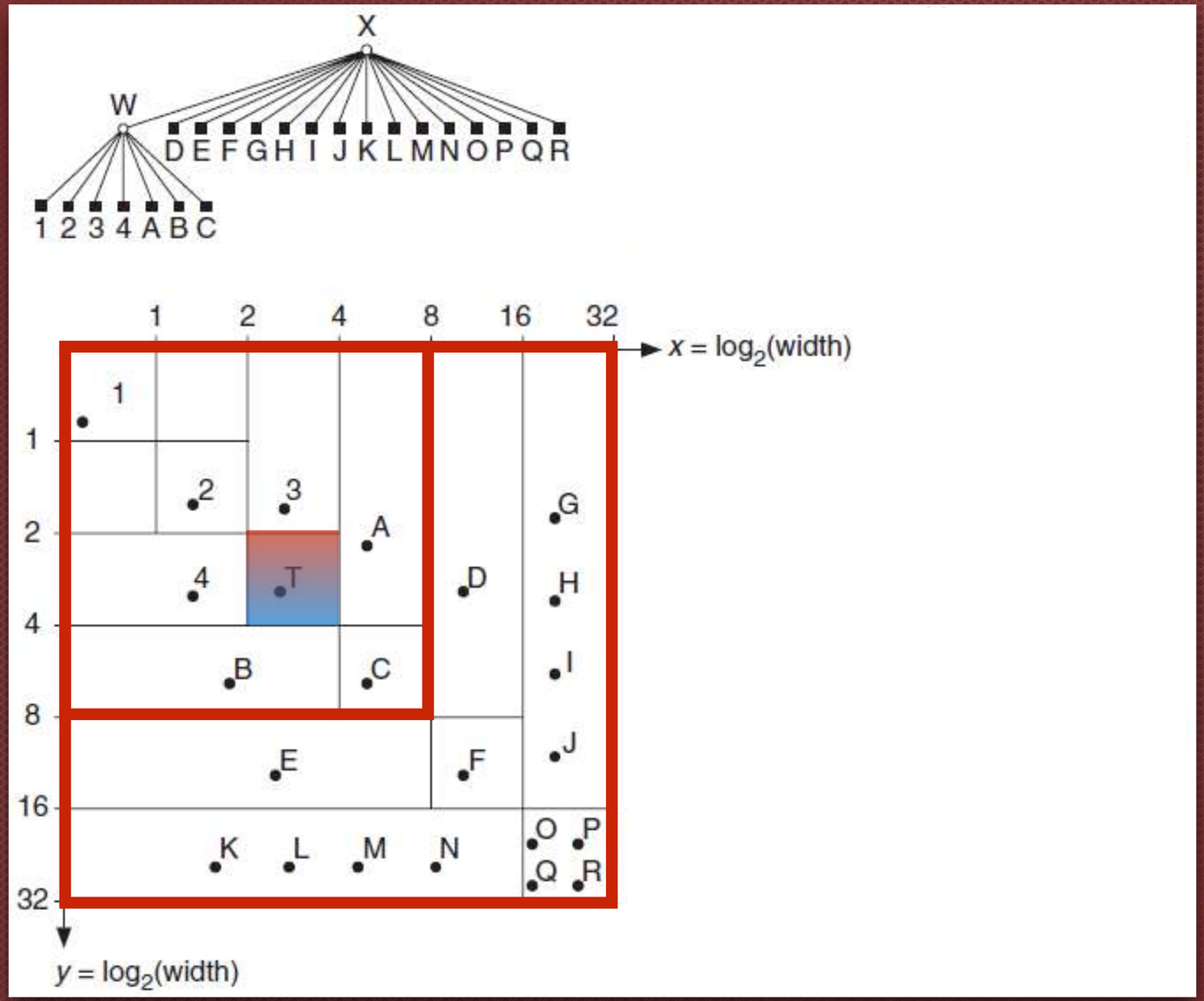

```
1  recursive node procedure PKINSERT(P, Q, F, K)
2  /* Attempt to insert point P in the PK-tree with parameter K rooted at node
       Q whose father is F. An empty PK-tree still has a nonempty root node
       with no entries in it. */
3  value pointer point P
4  value pointer node Q, F
5  value integer K
6  pointer node D, S, T
7  S ← FINDCHILD(P, Q)
8  D ←if ISNULL(S) then MAKENODEANDADDASCHILD(P, Q)
9        else PKINSERT(P, S, Q, K)
10       endif
11 T ← FORMNODEFORMINENCLOSINGBLOCKWITHKITEMS(D, Q, K)
12 if T = Q then
13    return Q /* Can exit completely as no k-instantiation has occurred */
14 else /* Propagate k-instantiation */
15    REMOVECHILDSOFNEWNODEFROMFATHER(T, Q)
16    INSERTNODE(T, Q)
17    if not ISNULL(F) and COUNTCHILDS(Q) < K then
18       /* propagate k-deinstantiation */
19       INSERTCHILDSOFNODE(Q, F)
20       REMOVENODEFROMFATHER(Q, F)
21       RETURNTOAVAIL(Q)
22       return T
23    else return Q /* Can exit completely as no further k-instantiation can */
24    endif         /* take place since no k-deinstantiation has occurred */
25 endif
```

Understanding the K-Instantiation



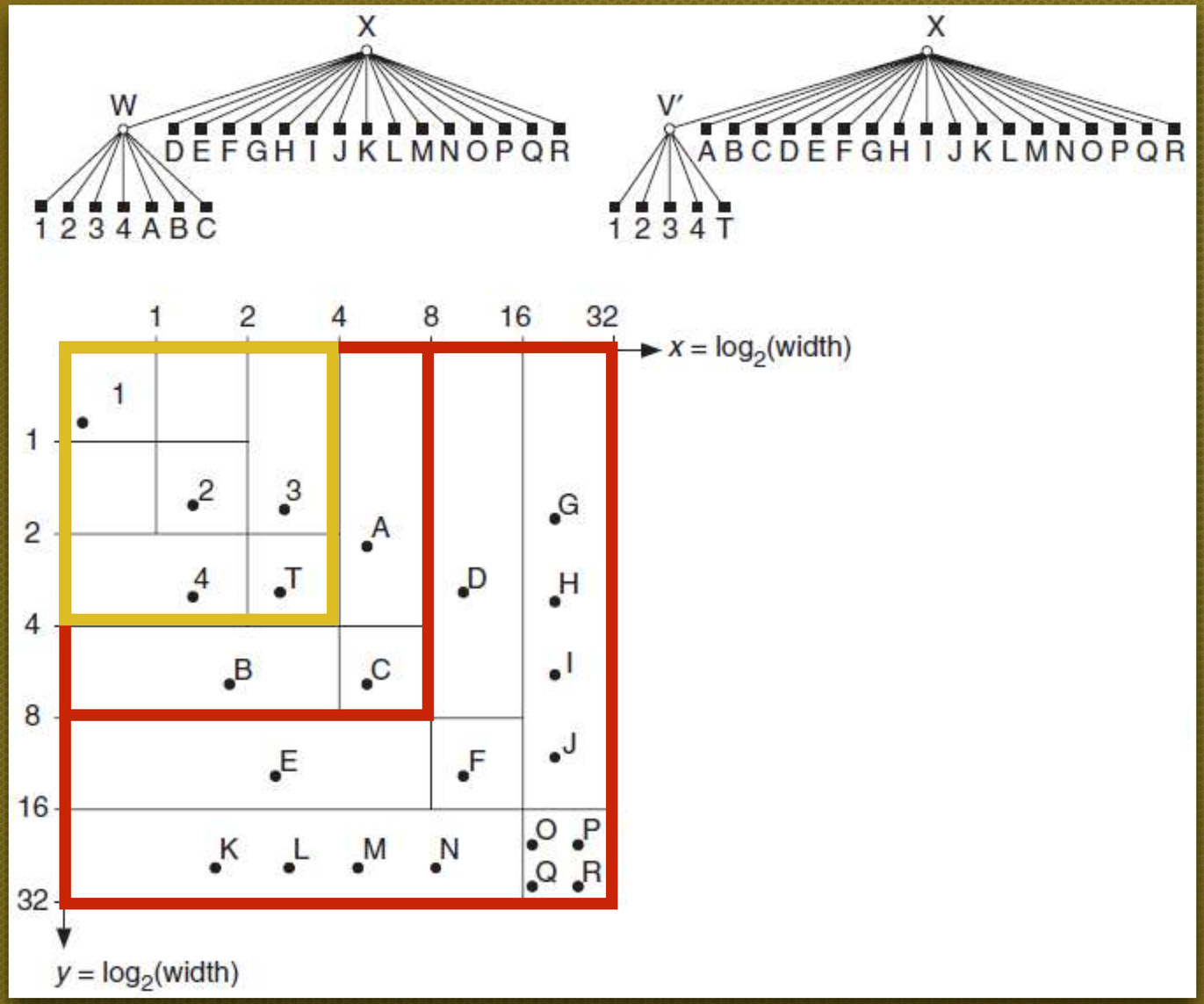
k=5

Understanding the K-Instantiation



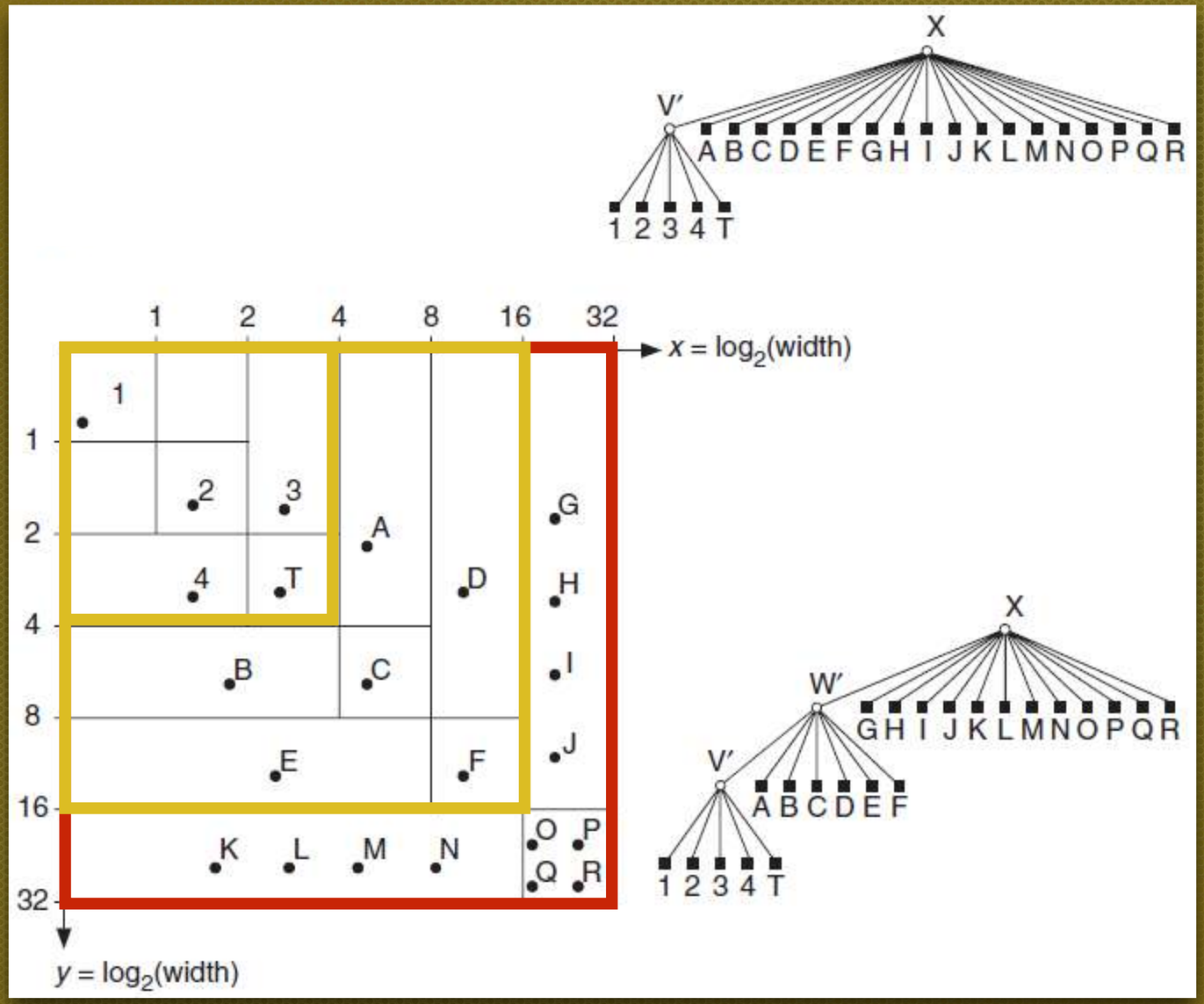
k=5

Understanding the K-Instantiation



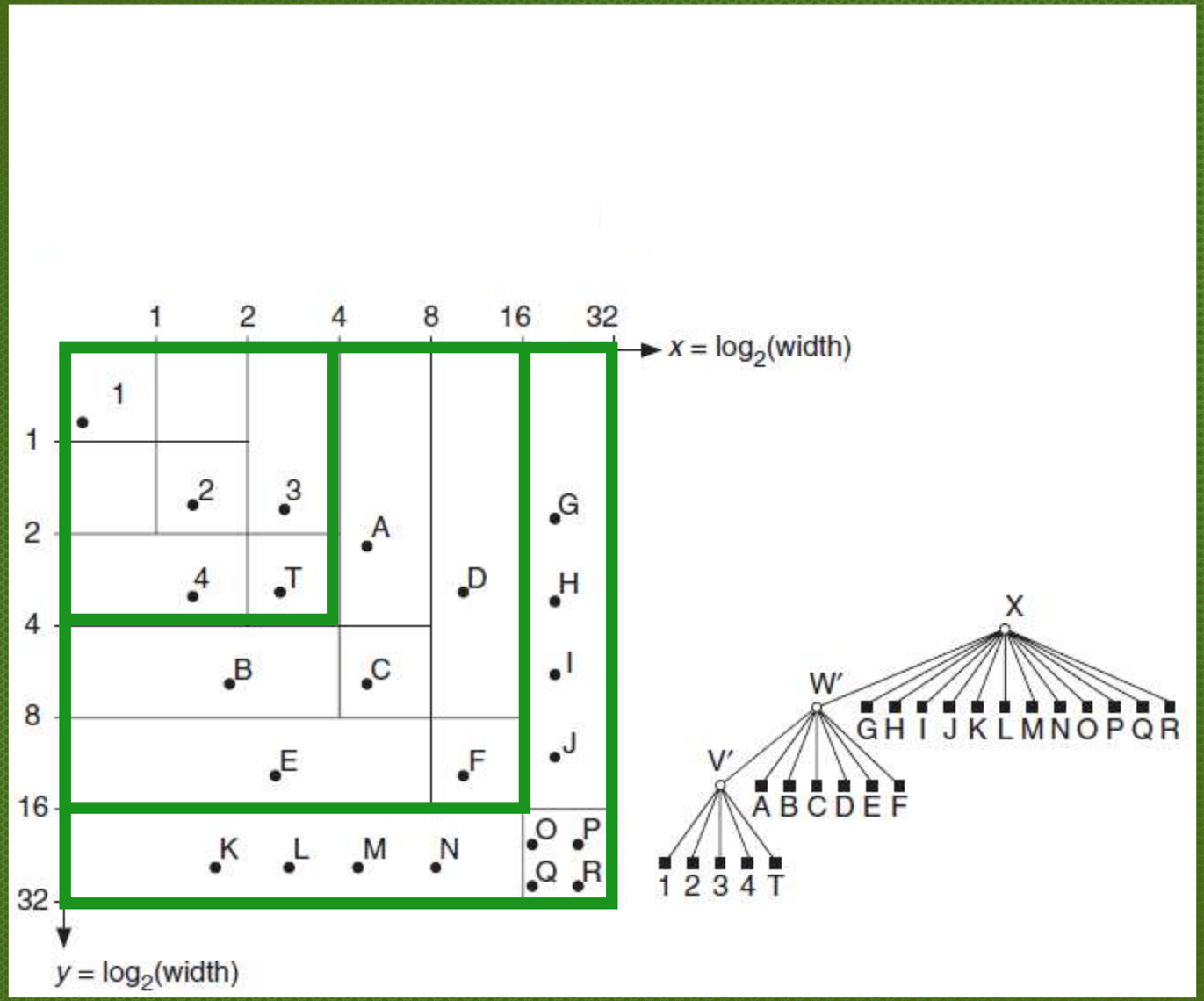
k=5

Understanding the K-Instantiation



k=5

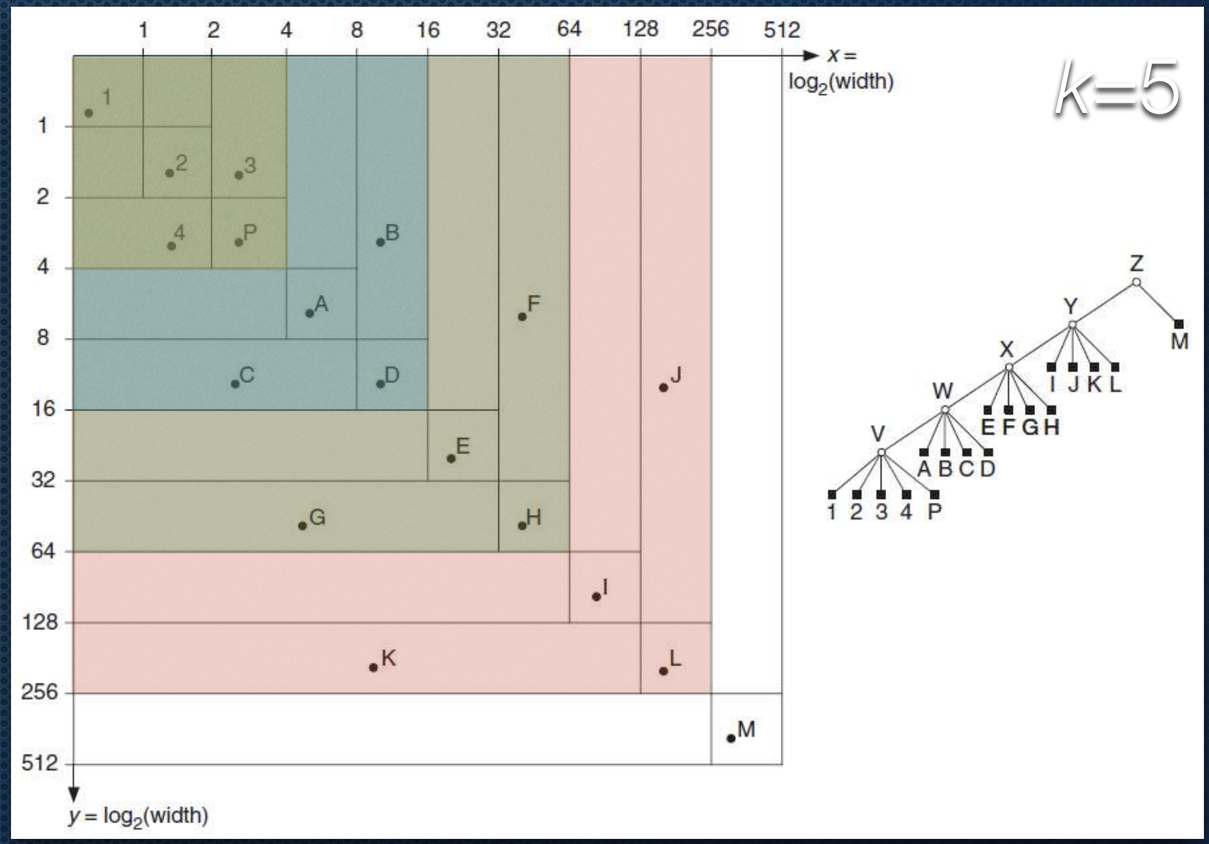
Understanding the K-Instantiation



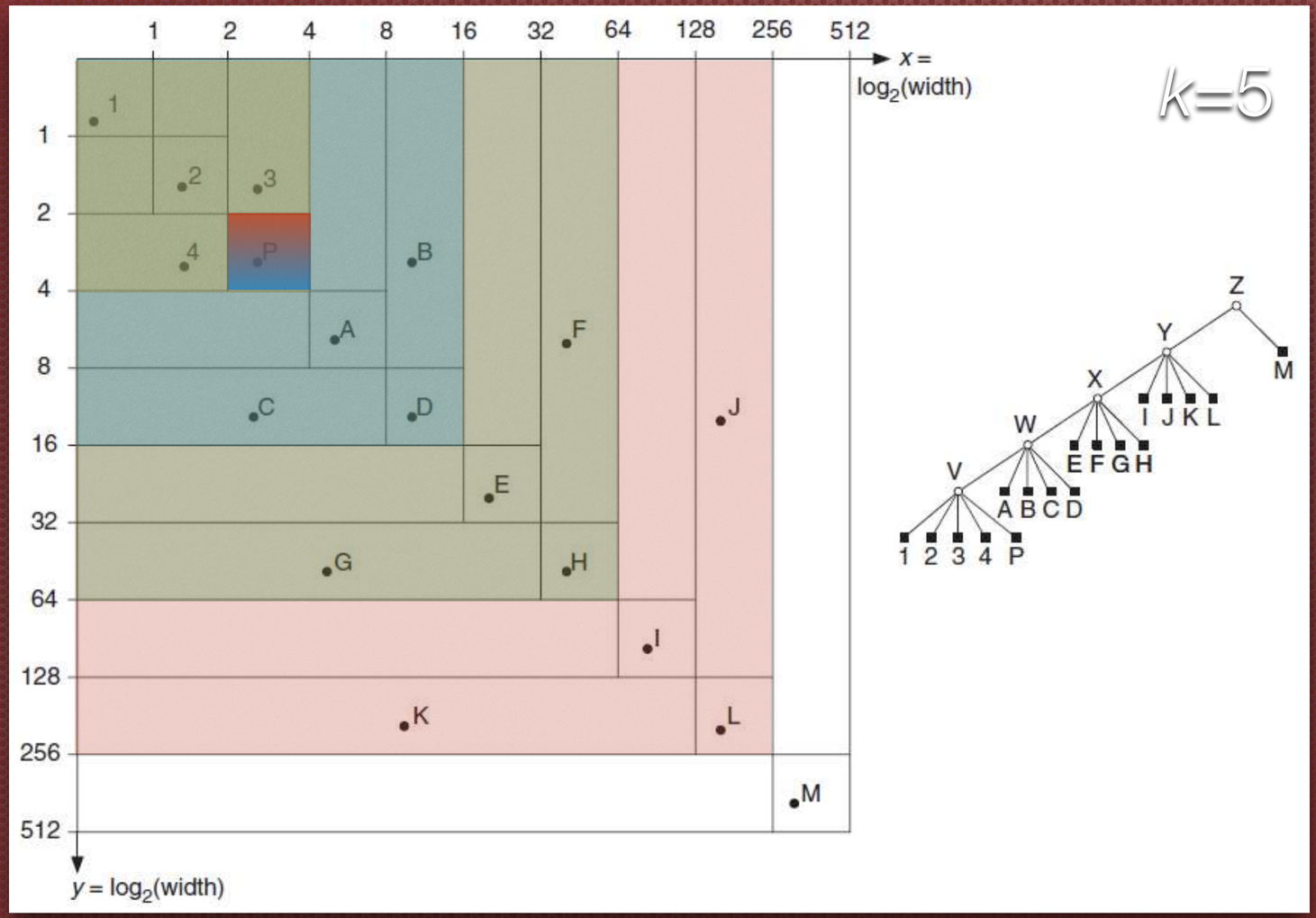
k=5

Deletion

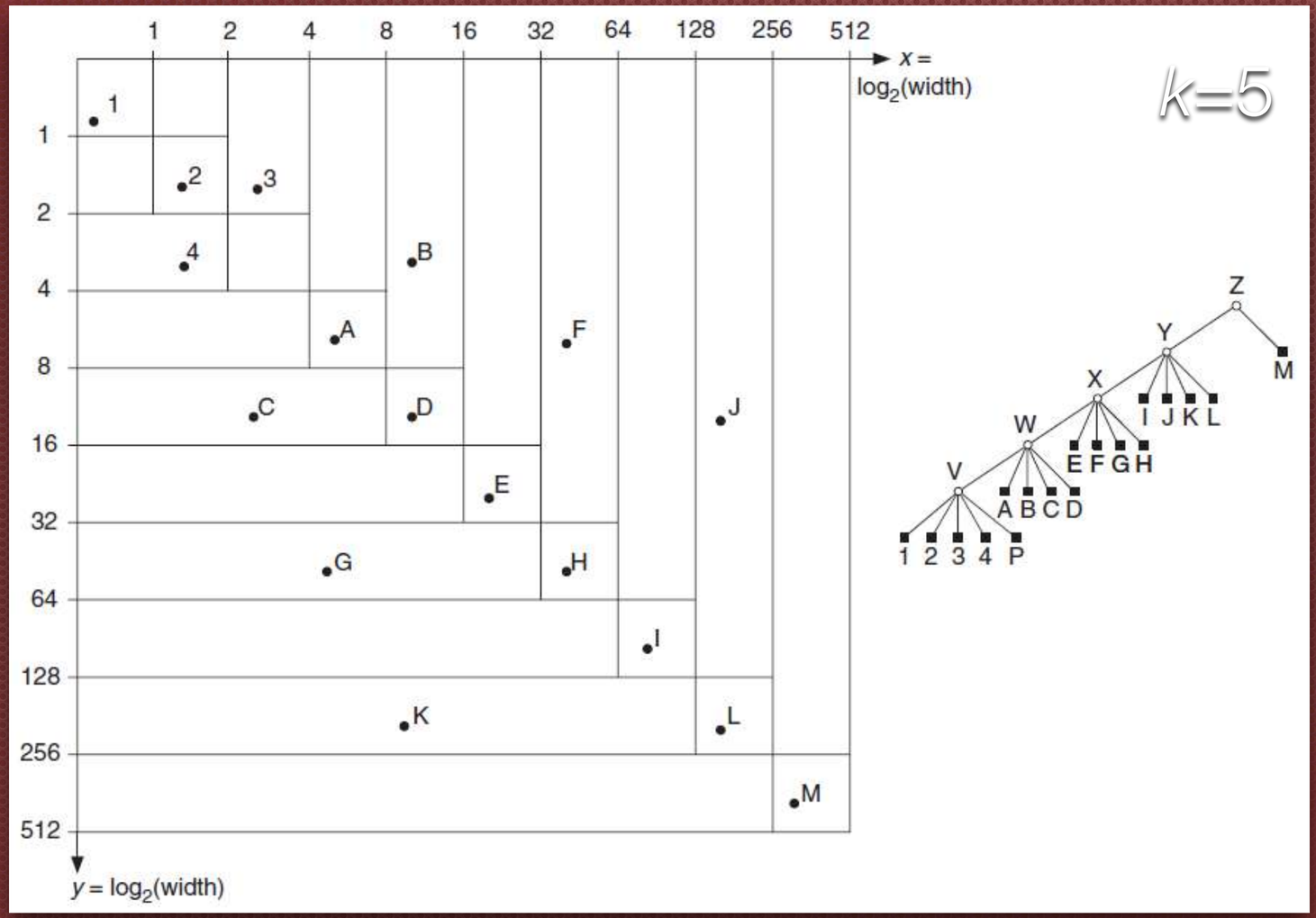
- Similar to insertion.
- May result in the removal of Directory-Nodes.
- We continuosly check in a bottom-up manner for the applicability of k-instantiation until it is no longer applicable.



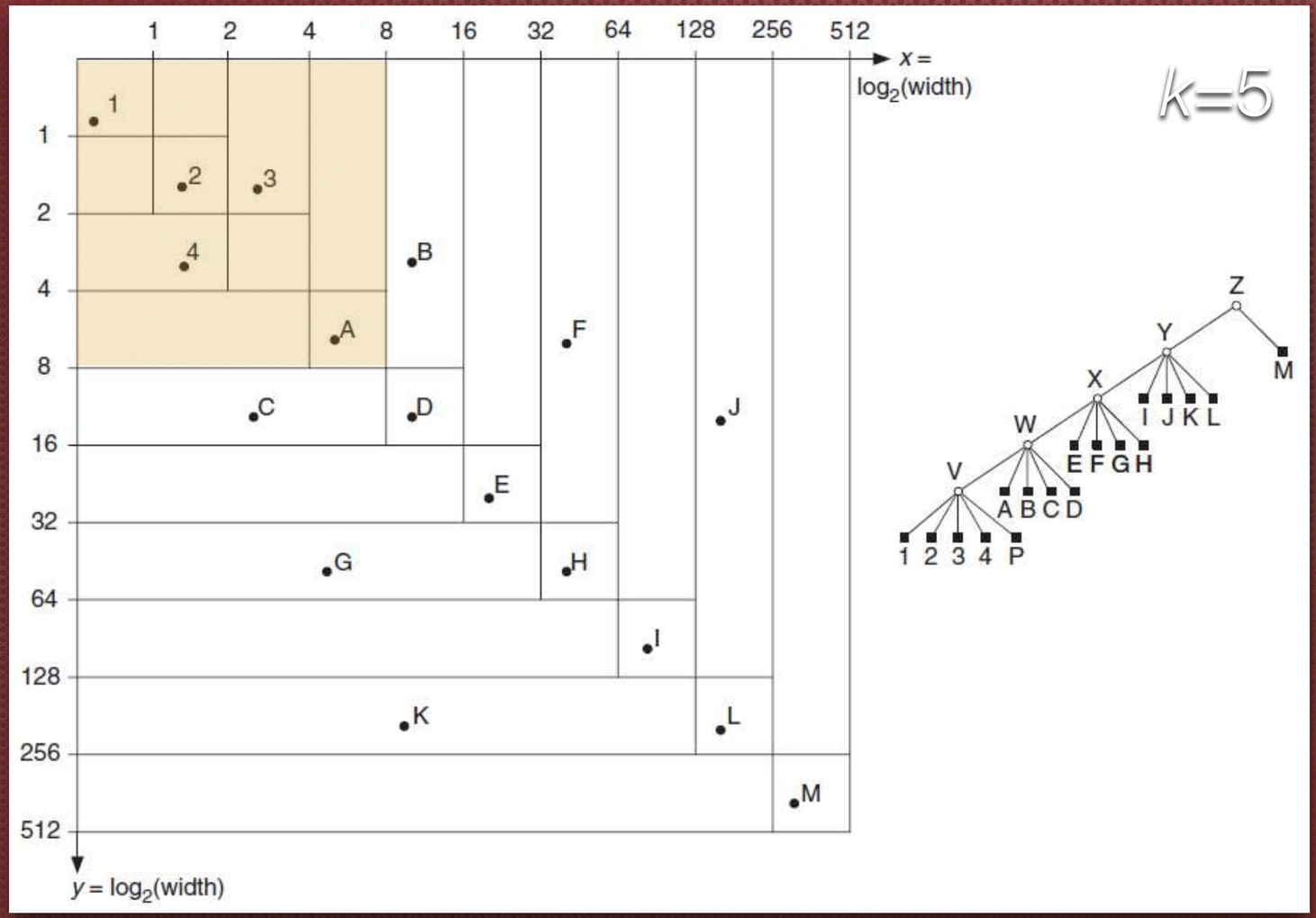
Deletion


1 2 4 8 16 32 64 128 256 512
x = log2(width)
1 2 4 8 16 32 64 128 256 512
y = log2(width)
1 2 3 4 P A B C D E F G H I J K L M
k=5
Z Y M X I J K L W E F G H V A B C D 1 2 3 4 P

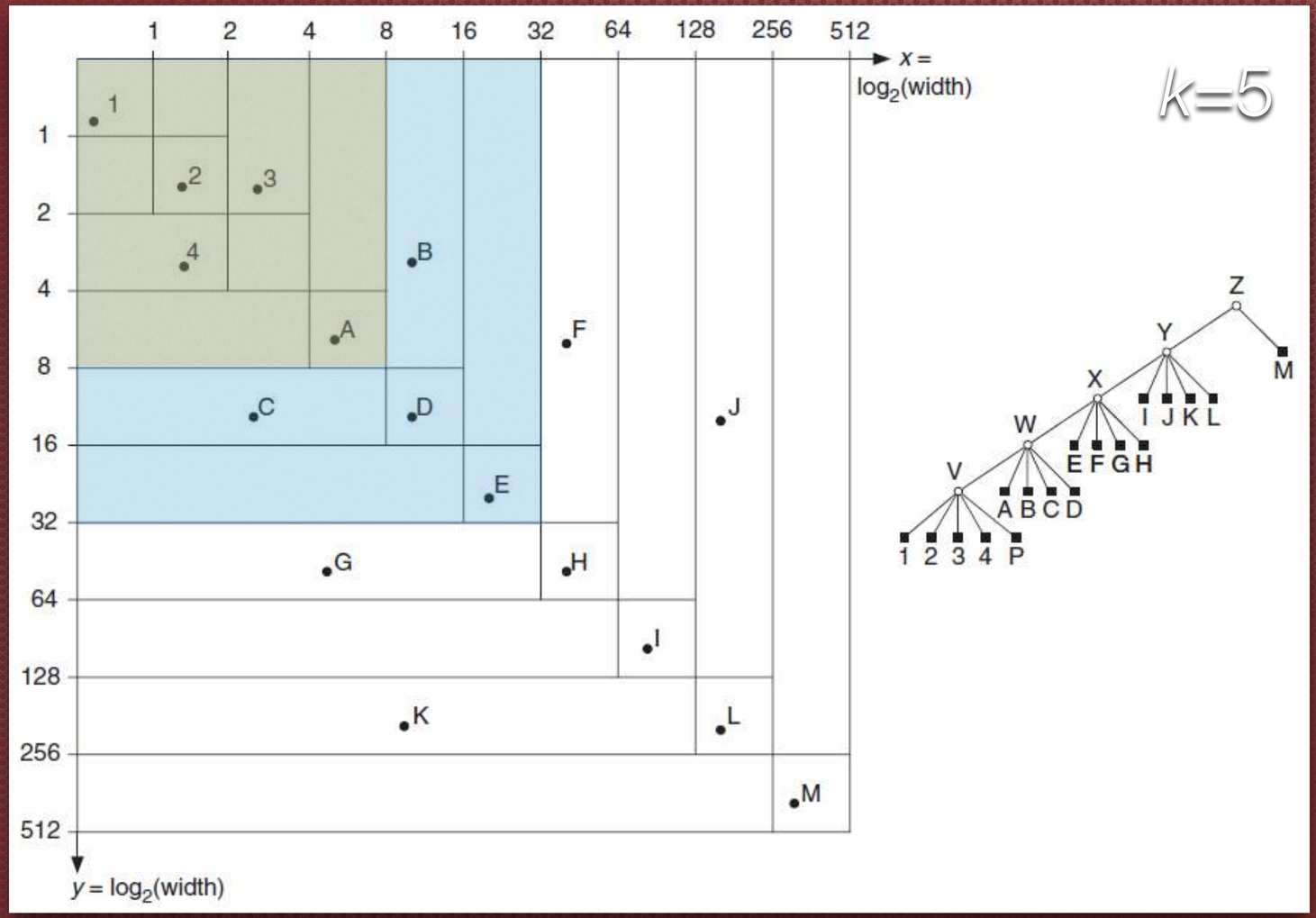
Deletion


1 2 4 8 16 32 64 128 256 512
x = log2(width)
1 2 4 8 16 32 64 128 256 512
y = log2(width)
1 2 3 4 A B C D E F G H I J K L M
k=5
Z Y M X I J K L W E F G H V A B C D 1 2 3 4 P

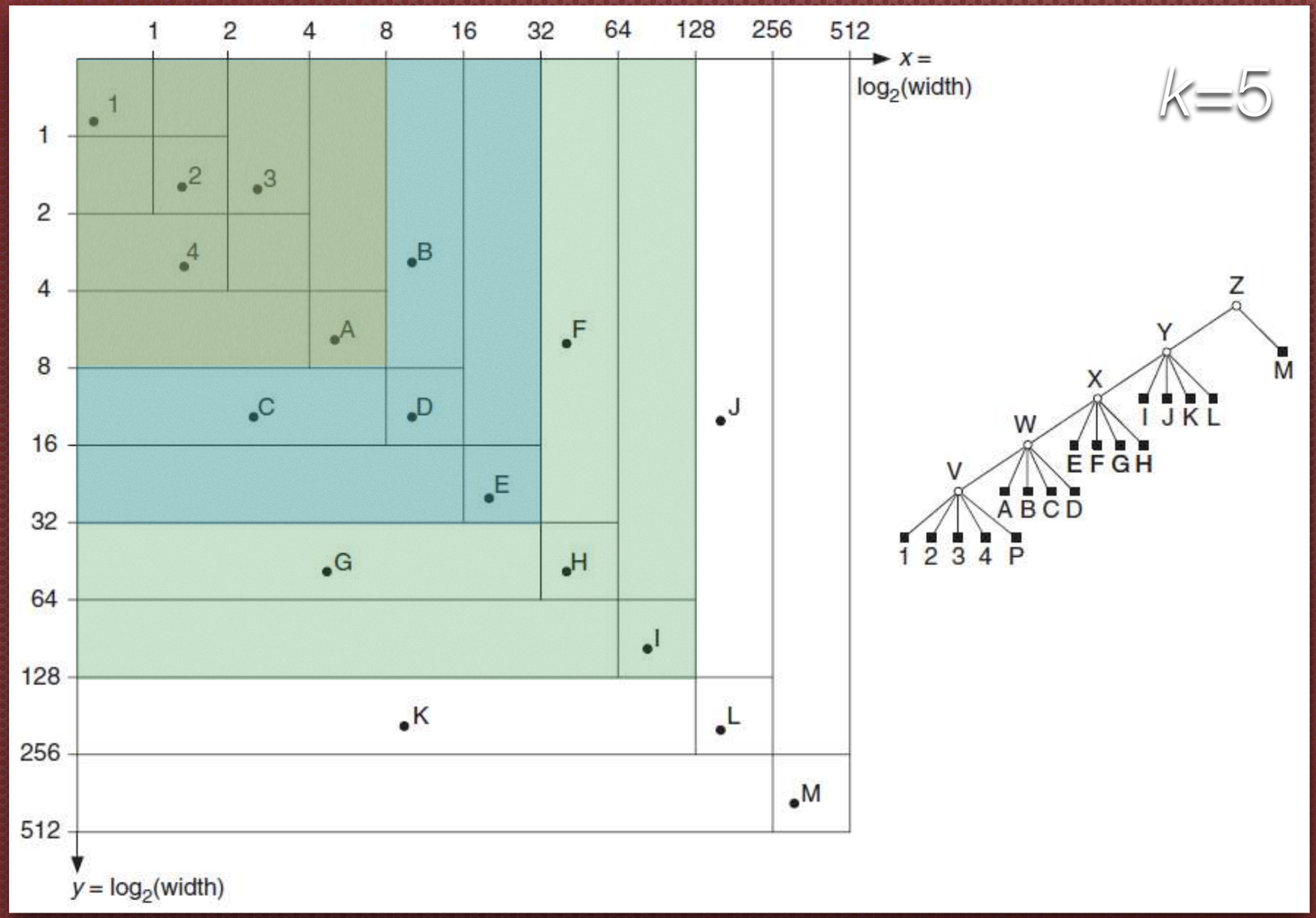
Deletion


1 2 4 8 16 32 64 128 256 512
x = log2(width)
1 2 4 8 16 32 64 128 256 512
y = log2(width)
1 2 3 4 A B C D E F G H I J K L M
k=5
Z Y M X I J K L W E F G H V A B C D 1 2 3 4 P

Deletion

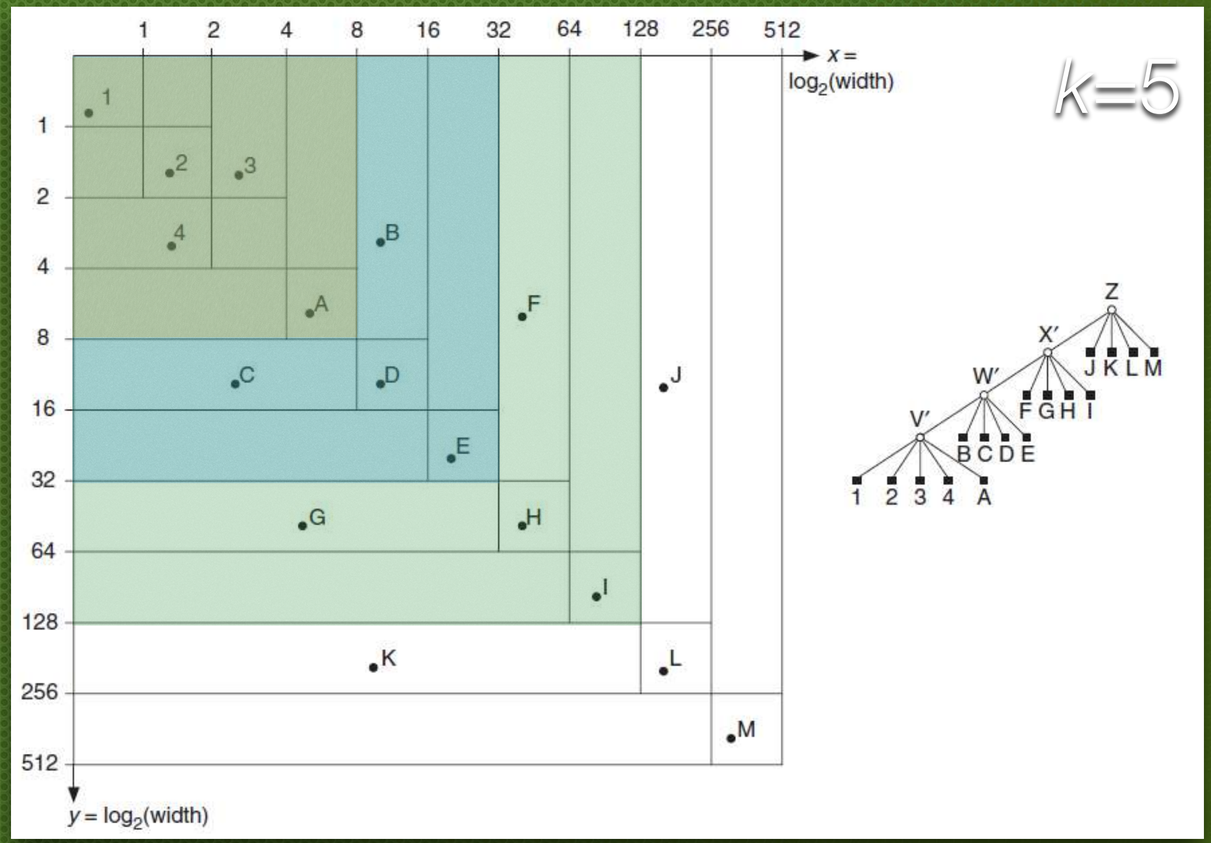

1 2 4 8 16 32 64 128 256 512
x = log2(width)
y = log2(width)
1 2 3 4 A B C D E F G H I J K L M
k=5
Z Y M X I J K L W E F G H V A B C D 1 2 3 4 P

Deletion

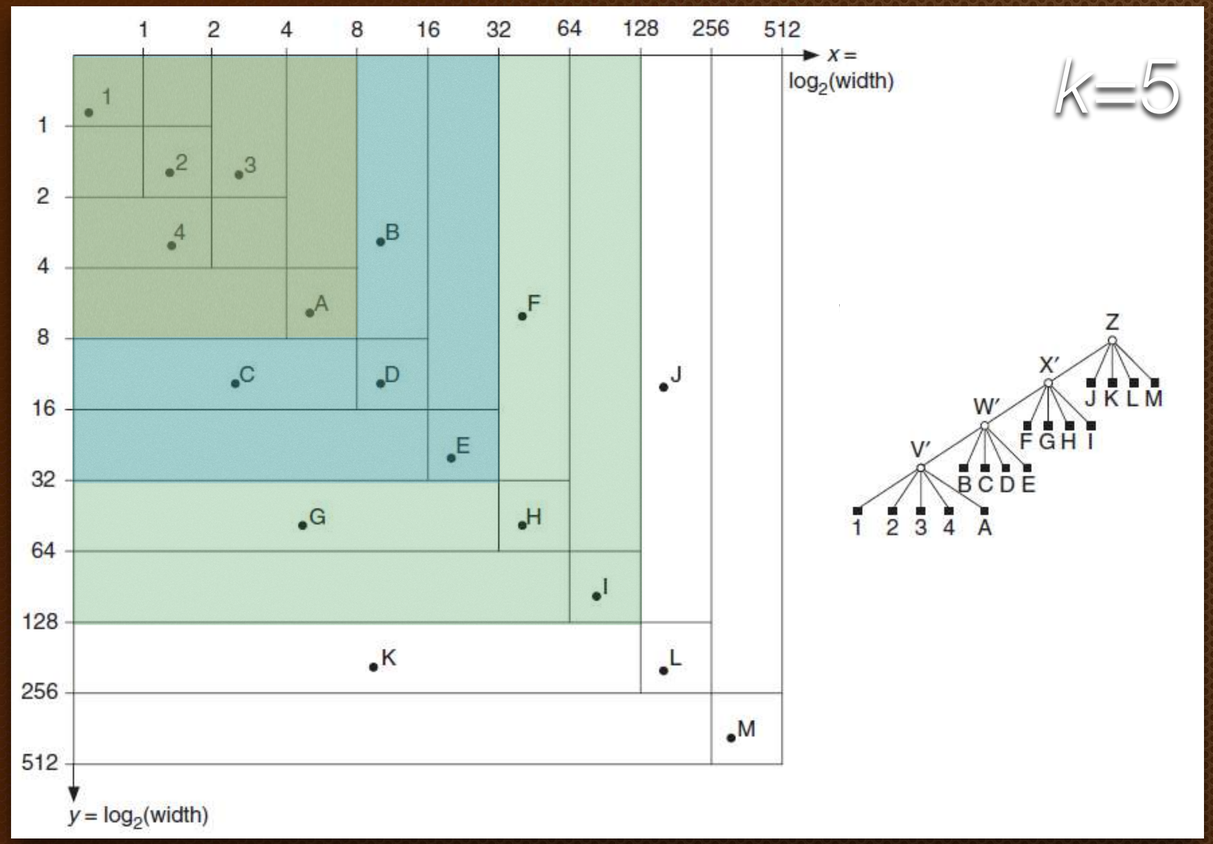

1 2 4 8 16 32 64 128 256 512
x = log2(width)
1 2 4 8 16 32 64 128 256 512
y = log2(width)
1 2 3 4 A B C D E F G H I J K L M P
k=5

Deletion

- Similar to insertion.
- May result in the removal of Directory-Nodes.
- We continuosly check in a bottom-up manner for the applicability of k-instantiation until it is no longer applicable.



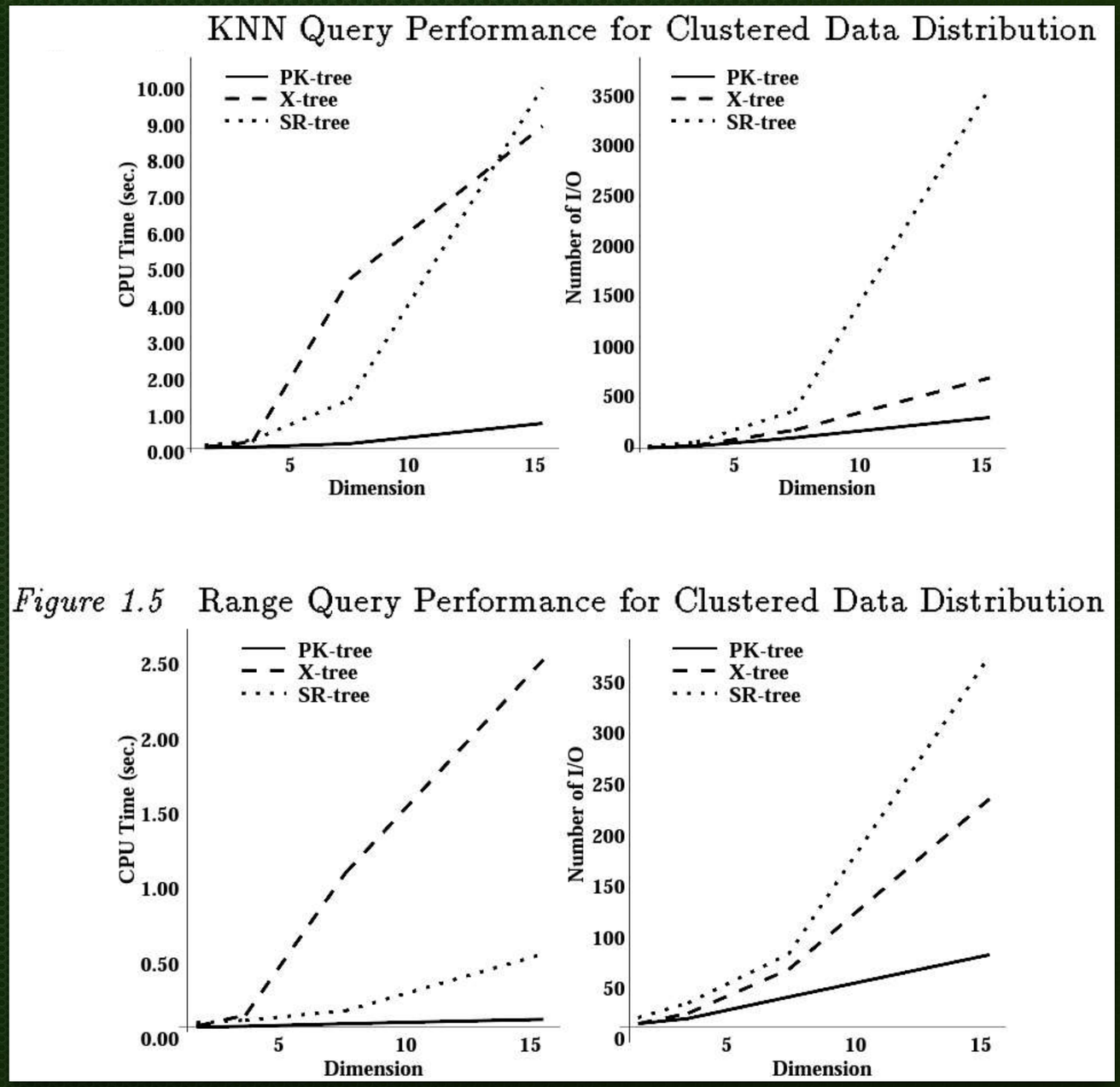
PKdelete Pseudocode:



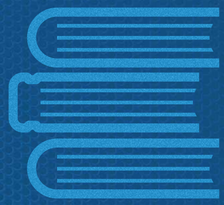

```
1   recursive node procedure PKDELETE(P, Q, F, K)
2   /* Attempt to delete node P from the PK-tree with parameter K rooted at
       node Q whose father is F. An empty PK-tree still has a nonempty root
       node with no entries in it. */
3   value pointer point P
4   value pointer node Q, F
5   value integer K
6   pointer node D, S, T
7   S ← FINDCHILD(P, Q)
8   if ISNULL(S) then
9      return Q /* P is not in the PK-tree and can exit completely */
10  elseif ISPOINT(S) then
11     /* S must be point P as, otherwise, FINDCHILD is NIL */
12     REMOVENODEFROMFATHER(S, Q)
13     RETURNTOAVAIL(S)
14     T ← CHOOSECHILD(Q)
15  else
16     D ← PKDELETE(P, S, Q)
17     T ← FORMNODEFORMINENCLOSINGBLOCKWITHKITEMS(D, Q, K)
18     if T = Q then
19        return Q /* Can exit completely as no k-instantiation has occurred */
20     else /* Propagate k-instantiation */
21        REMOVECHILDSOFNEWNODEFROMFATHER(T, Q)
22        INSERTNODE(T, Q)
23     endif
24  endif
25  if not ISNULL(F) and COUNTCHILDS(Q) < K then
26     /* propagate k-deinstantiation */
27     INSERTCHILDSOFNODE(Q, F)
28     REMOVENODEFROMFATHER(Q, F)
29     RETURNTOAVAIL(Q)
30     return T
31  else          /* Can exit completely as no further k-instantiation can */
32     return Q /* take place since no k-deinstantiation has occurred */
33  endif
```

Final words

- Maximum depth of a PK-tree is O(N).
- Expected search O(logN).
- Merge (k-instantiation) and split (k-deinstantiation).
- Number of **splitting** and **mergings** operations needed may be as large as the maximum depth of the tree.
- PK-Tree **no distinction** is made between data and directory nodes.



Bibliography



- Foundations of Multidimensional and Metric Data Structures, Hanan Samet, 2006.
- PK-Tree: A Spatial Index Structure for High Dimensional Point Data, Wei WangJiong YangRichard Muntz, 2000.


Thanks.