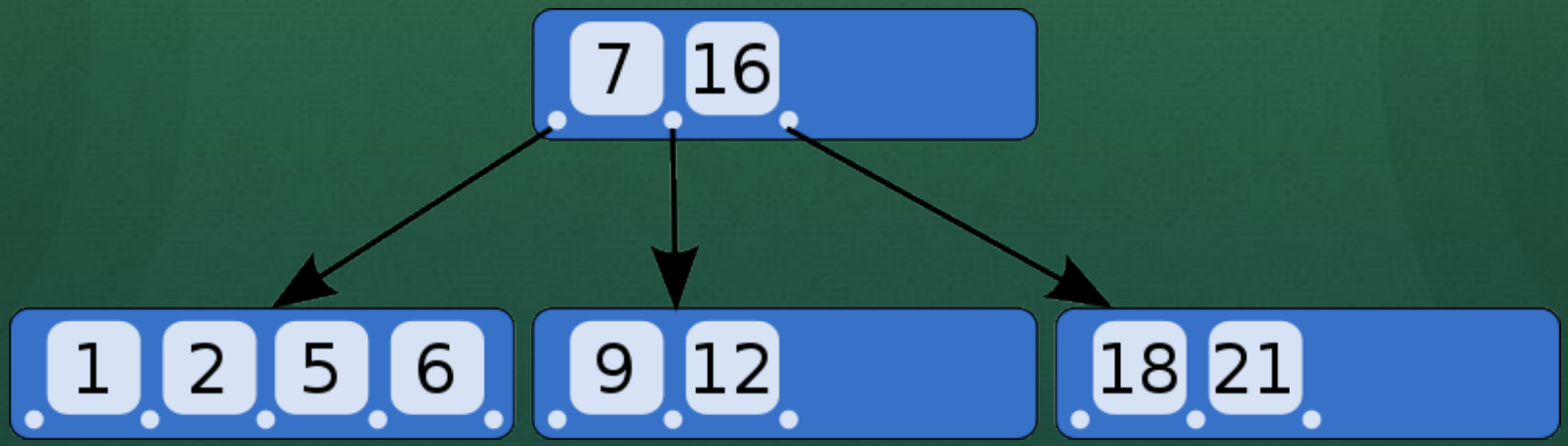

B-trees and R-trees
Neri Van Otten

Index Improves Performance of Search

- For N objects worst case Ω(n)
- Many index structure have O(log(N))
- In some applications O(1) is possible
- A good index structure has ability to collect similar data into same portion
- Some of the index structures that are widely used and some are more application or query type specific

B-tree

- Variable number of child node with predefined range
- Inserting or deleting nodes can cause a rebalance
- Keeps records in sorted order for traversing



Knuth's* defenition

1. Every node has at most n children.
2. Every node has at least n/2 children.
3. The root has at least two children if it is not a child node.
4. All leaf nodes are at the same level.
5. A non-Leaf node has n children contains n-1 keys.

*Knuth, Donald, "Sorting and Searching, The Art of Computer Programming", Addison-Wesley, ISBN 0-201-89685-0 , Vol. 3 (Second ed.), Section 6.2.4, Multiway Trees, pp. 481–491, 1998.

Properties of B-trees

- Height: best case log(mn), worst case log(m/2n)
- Searching similar to binary search tree with binary search used within a node
- Used in Apple's file system HFS+, Microsoft's NTFS and some Linux file systems such as btrfs and Ext4
- B-tree is efficient for the point query but not for range query and multi-dimensional data

R-tree

- Dynamic index structure for the spatial searching
- Let M be the maximum number of entries in one node and minimum number of entries in a node is $n \leq M/2$. R-tree satisfies following properties:

1. Each leaf node(unless it is root) has index records between n and M.
2. Each index record (I, tuple- identifier) in a leaf node. I is smallest rectangle represented by the indicated tuple and contains the n- dimensional data object.

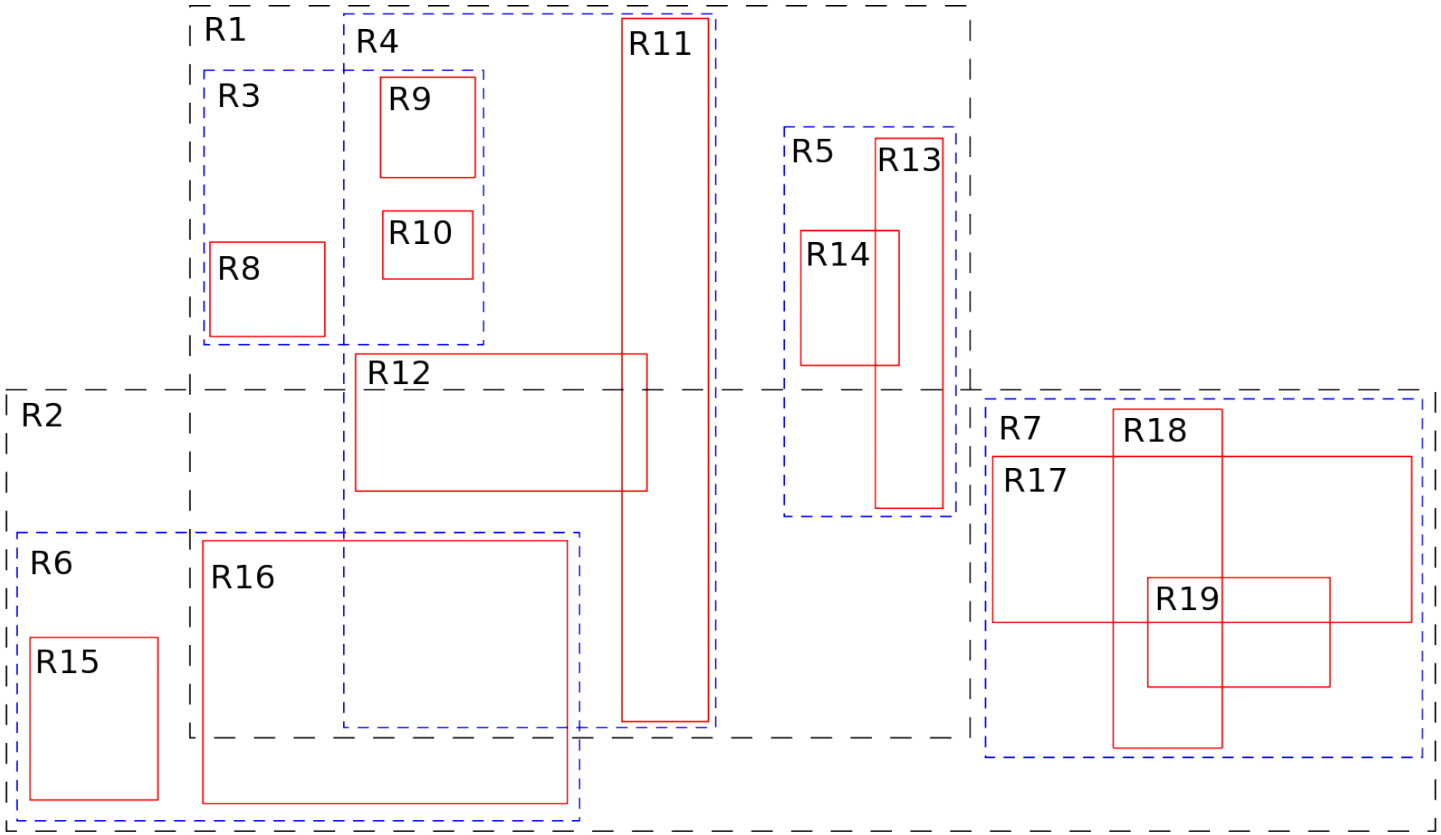
R-tree

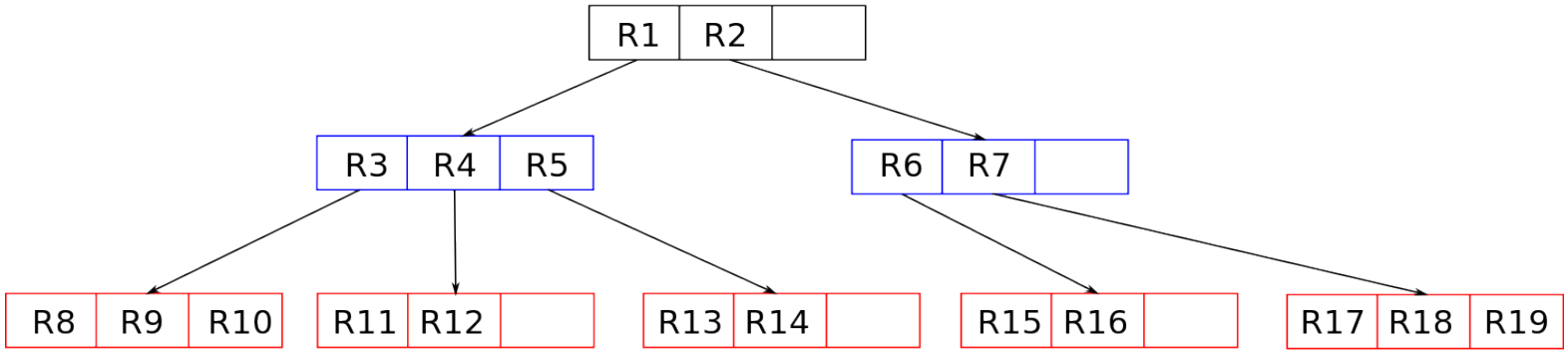
3. Each non-leaf (unless it is root) has children between n and M.

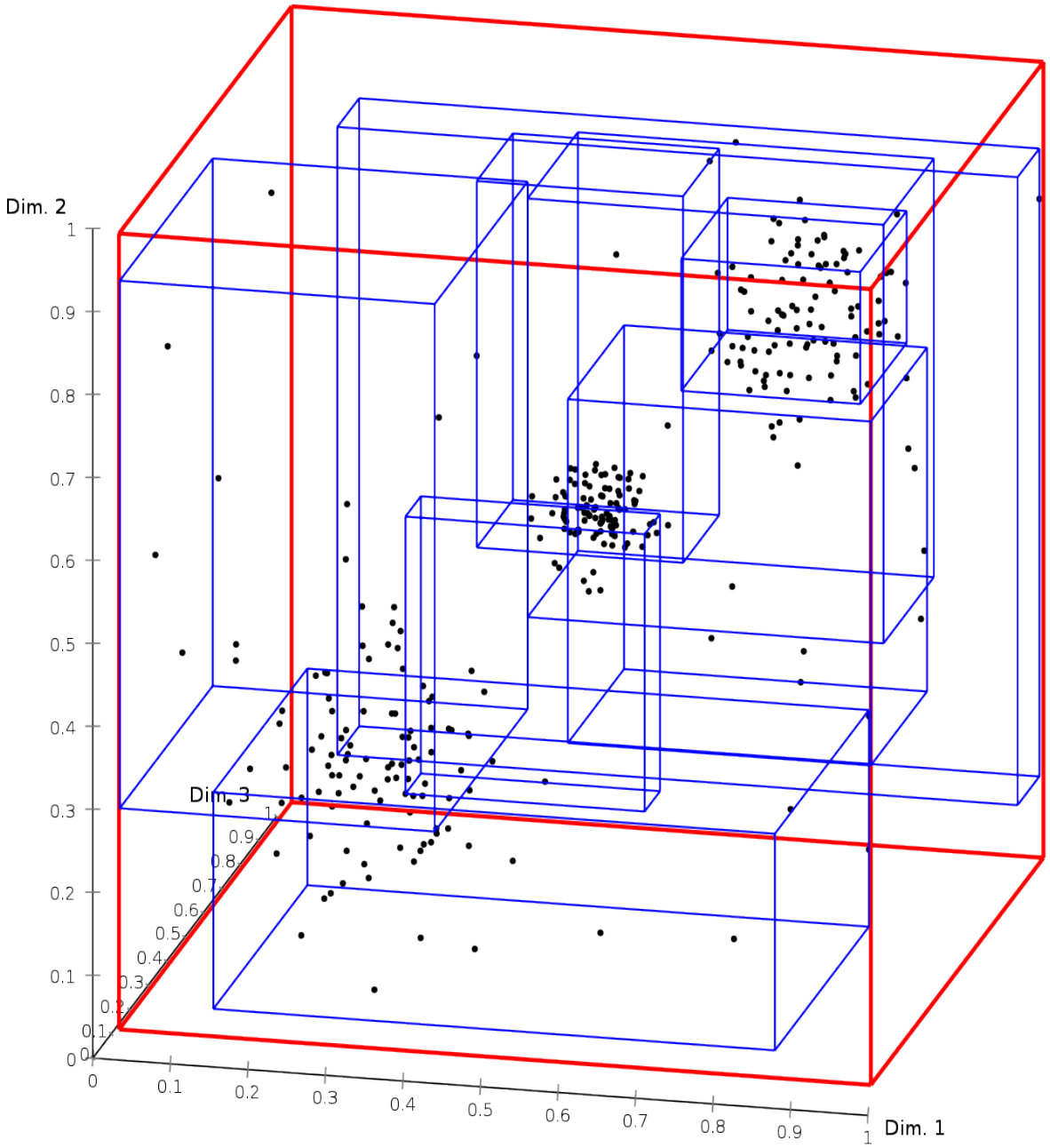
4. Each entry in non-leaf node (I, child pointer), I contain the rectangle in the child node is the smallest rectangle.

5. The root node (unless it is leaf) has at least two children.

6. All leaves appear on the same level.


R1
R4
R11
R3
R9
R5
R13
R10
R14
R8
R12
R2
R7
R18
R17
R6
R16
R19
R15


R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19


Dim. 2
Dim. 3
Dim. 1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Properties of R-trees

- Searching and insertion is similar to B-tree
- R-tree gives best performance when 30-40% full as complex balancing is required for spatial data

Variants of B-tree

- B+-tree: B+-tree is similar to the B-tree the difference is all records are stored at leaf level and only keys stored in non-leaf nodes.
- B+-tree is use in relational database system for metadata indexing and used for the data stored in RAM.
- B*-tree balance more internal neighbor nodes in order to keep internal nodes more densely packed. When both nodes are full they split into three, single node gets full then it shares with the next node

Variants of B-tree

- UB-tree is for storing and retrieving multidimensional data. The algorithm provided for the range search in the multidimensional point data is exponential with dimension so not feasible.
- H-tree is used for directory indexing. It has constant depth of one or two levels and do not require balancing, use a hash of a file name. It is use in Linux file system ext3 and ext4.

Variants of B-tree

- ST2B-tree: A Self-Tunable Spatio-Temporal B+-tree Index for Moving Objects. It is built on B+-tree without change in insertion and deletion algorithms.
- Compact B+-tree uses vacant space of siblings before the overflow happens in a node, reducing the amount of split operations.

	Query type	Data type	Complexity	Application
B-tree	Point query	Linear data	O(log n)	HFS+, NTFS , btrfs and Ext4
B+-tree	Point query	Linear data	O(log n)	IBM DB2, Microsoft MySql, Oracle 8, Sybase ASE
B*-tree	Point query	Linear data	O(log n) use space more efficiently than B+-tree	HFS and Reiser4 file systems
UB-tree	Point/ range query	Linear data, multidimensional data	O(log n) but not for multidimensional data	Multidimensional range search.
H-tree	Point query	Point query	O(log n) utilize space more efficiently	Ext3, ext4 Linux file systems.
ST2B-tree	Point query	Multidimensional data	Better for moving object data	Multidimensional data
Compact B-tree	Point query	Linear data	O(log n) use space more efficiently	In place of B-tree.

Variants of R-tree

- R+-tree differs from R-trees slightly. Nodes are not guaranteed to be at least half filled. Entries of internal node do not overlap and object id may be stored in more than one leaf node.
- R+-tree searching follows single path fewer nodes are visited than R-tree. But data are duplicated over many leaf node structure of R+-tree can be larger than R-tree.

Variants of R-tree

- R*-tree outperform the traditional R-tree in query processing and performance. The costly operation is reinsertion but it reduce the split operation. Storage utilization is higher than variants of R-tree but implementation cost is higher than R-tree.
- M-tree is fully parametric on a distance function and triangle inequality for efficient queries. It has overlap of regions and no strategy to avid overlap.

Variants of R-tree

- X-tree prevents overlapping of bounding boxes, which is a problem in high dimension data. Nodes not split will result into super-nodes and in some extreme cases these trees will linearize.
- Hilbert R-tree is used for indexing of object like line, curve, 3-D object and high dimension future based parametric objects.

Variants of R-tree

- Bloom filter base R-tree (BR-tree) in which bloom filter is integrated to R-tree node. BR-tree is basically R-tree structure for supporting dynamic indexing. In it each node maintains range index to indicate attribute of existing item.
- QR+-tree is hybrid structure of Quad tree (Q-tree) and R-tree. First rough level partition of index space using Q-tree and then use R-tree to index space object.

	Query type	Data type	Complexity	Application
R-tree	Range query	Multidimensional data	Not utilize space more efficiently, not have worst case time complexity	Real world application like navigation system etc.
R+-tree	Range query	Multidimensional data	Non overlapping data utilize space efficiently than R-tree	Multidimensional data object
R*-tree	Point/ Range query	Spatial data, multidimensional data	Implementation cost is more than other R-tree variants but robust in data distribution than other ugly structures.	Application with data in form of points and rectangles
X-tree	Range query	Multidimensional data, High dimensional data	In some extreme cases tree become linear and time complexity O(n)	High dimension data

	Query type	Data type	Complexity	Application
M-tree	Range query, k-NN query	Multidimensional data	Not require periodic reorganization, time is less in construction	k-NN query, application use multidimensional (spatial) access methods
Hilbert R-tree	Range query	Multidimensional data	Search cost give 28% saving above R*-tree.	Cartography, Computer Aided Design(CAD), computer vision and robotics etc
BR-tree	Point query, range query, cover query, bound query	Linear data, Multidimensional data	O(≤ log n)	Application require all four type of query and also use in distributed environment
QR+-tree	Range query	Large scale spatial data	No redundant information make query processing more efficient.	Large scale GIS database

ST2B-trees: Su Chen, Beng Chin Ooi, Kian-Lee Tan, M. A. Nascimento, “ST2B-tree: A Self-Tunable Spatio- Temporal B+-tree Index for Moving Objects”, Proc. ACM SIGMOD international conference on Management of data, 2008.

Compact B+-tree, QR+-tree: Mao Huaqing, Bian Fuling, “Design and Implementation of QR+Tree Index Algorithms”, International Conference on Digital Object Identifier, pp. 5987 - 5990, 2007.

X-tree: Stefan Berchtold, D. A. Keim, Hans-Peter Kriegel, "The X-Tree: An Index Structure for High-Dimensional Data", Proc. 22th International Conference on Very Large Data Bases, pp. 28–39, 1996.

M-tree: Paolo Ciaccia, Marco Patella, Pavel Zezula, "M-tree An Efficient Access Method for Similarity Search in Metric Spaces", Proc. 13th International Conference on Very Large Data Bases, 1997.

Hilbert R-tree: I. Kamel, C. Faloutsos, “Hilbert R-tree: An improved R- tree using fractals”, Proc. 20th International Conference on Very Large Data Bases, pp. 500–509, 1994.

Bloom filter based R-tree: Yu Hua, Bin Xiao, Jianping Wang, “BR-Tree: A Scalable Prototype for supporting Multiple Queries of Multidimensional Data”, IEEE Transactions on Computers, vol. 58, Issue 12, pp. 1585–1598, 2009.