



THE UNIVERSITY OF  
MELBOURNE

# COMP90050 Advanced Database Systems

## Winter Semester, 2022

Lecturer: Farhana Choudhury (PhD)

Week 2 part 6

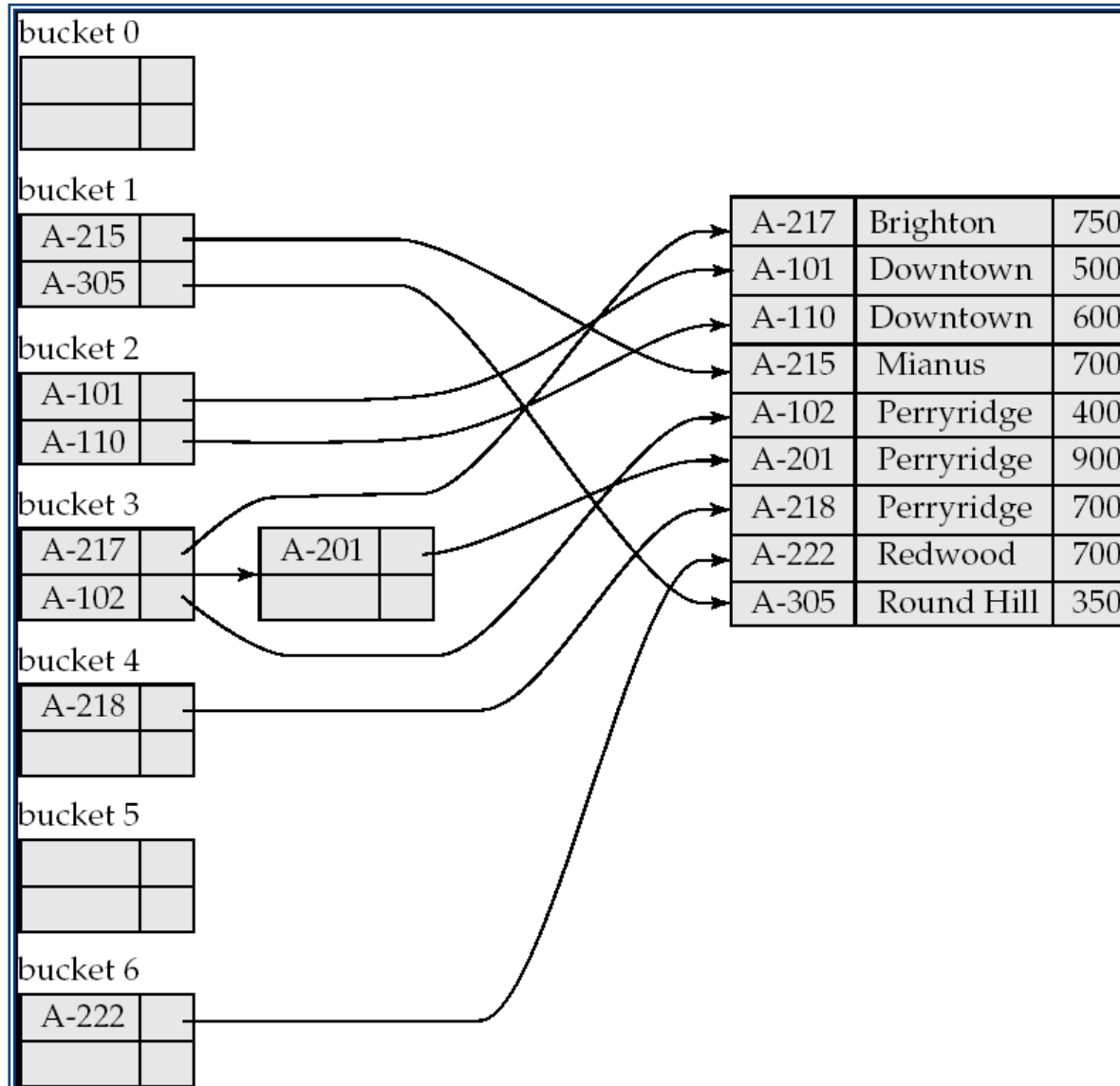




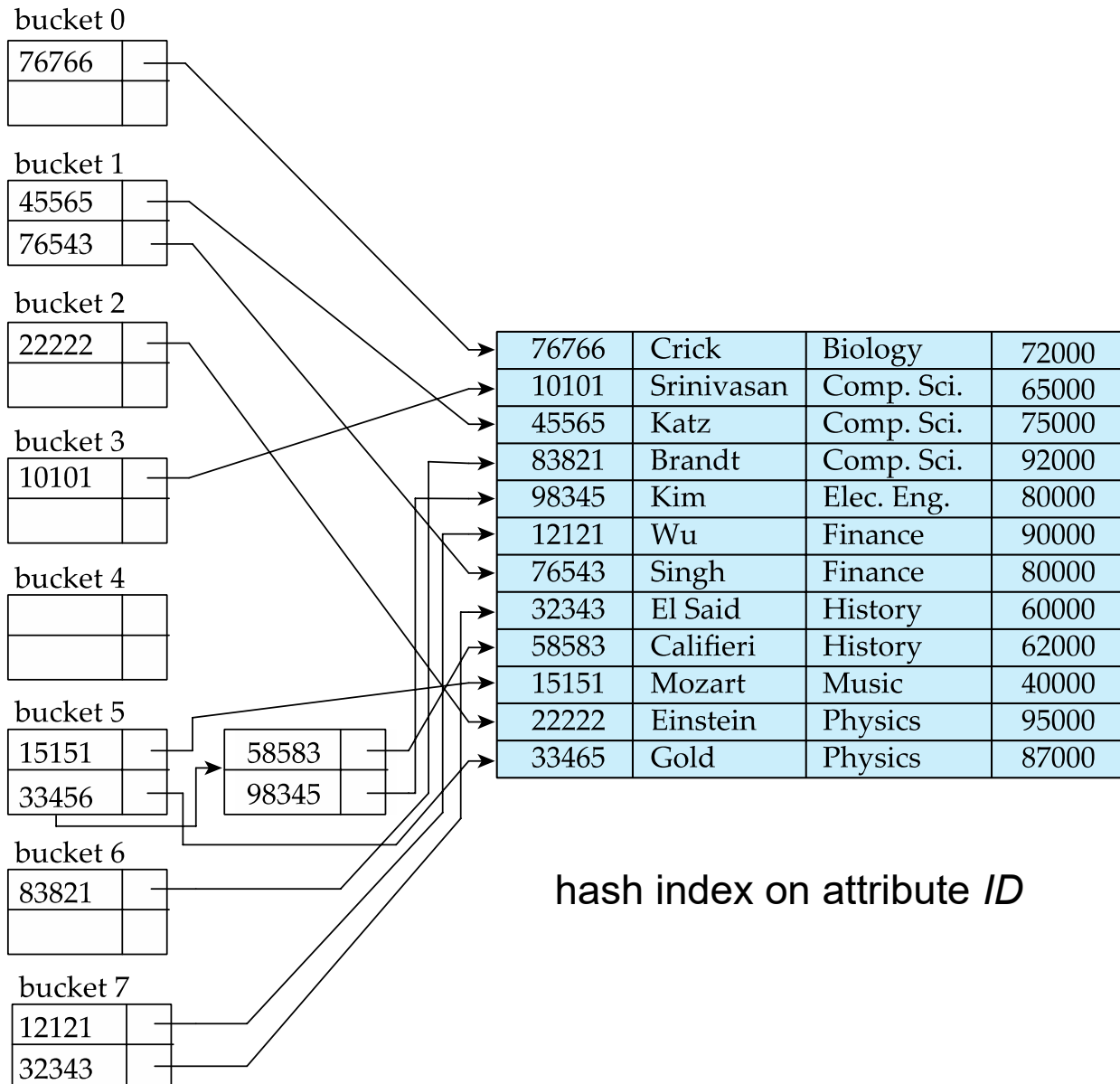
# Hash Indices

- A **hash index** organizes the search keys, with their associated record pointers, into a hash file structure. Order is not important.
- Hash indices are always secondary indices.
- Given a key the aim is to find the related record on file in one shot which is important.
- An ideal hash function is **uniform**, i.e., each bucket is assigned the same number of search-key values from the set of *all* possible values.
- Ideal hash function is **random**, so each bucket will have the same number of records assigned to it irrespective of the *actual distribution* of search-key values in the file.
- Typical hash functions perform computation on the internal binary representation of the search-key.

# Example



# Another example of Hash Index





# Bitmap Indices

- Records in a relation are assumed to be numbered sequentially from, say, 0
- Applicable on attributes that take on a relatively small number of distinct values
  - E.g. gender, country, state, ...
  - E.g. income-level (income broken up into a small number of levels such as 0-9999, 10000-19999, 20000-50000, 50000- infinity)
- A bitmap is simply an array of bits

# Example

- In its simplest form a bitmap index on an attribute has a bitmap for each value of the attribute
  - Bitmap has as many bits as records
  - In a bitmap for value  $v$ , the bit for a record is 1 if the record has the value  $v$  for the attribute, and is 0 otherwise
  - Used for business analysis, where rather than individual records say how much of one type exists is the query/important

Record Num	Name	State	Income_level
0	John	VIC	L1
1	Diana	NSW	L2
2	Xiaolu	WA	L1
3	Anil	VIC	L4
4	Peter	NSW	L3

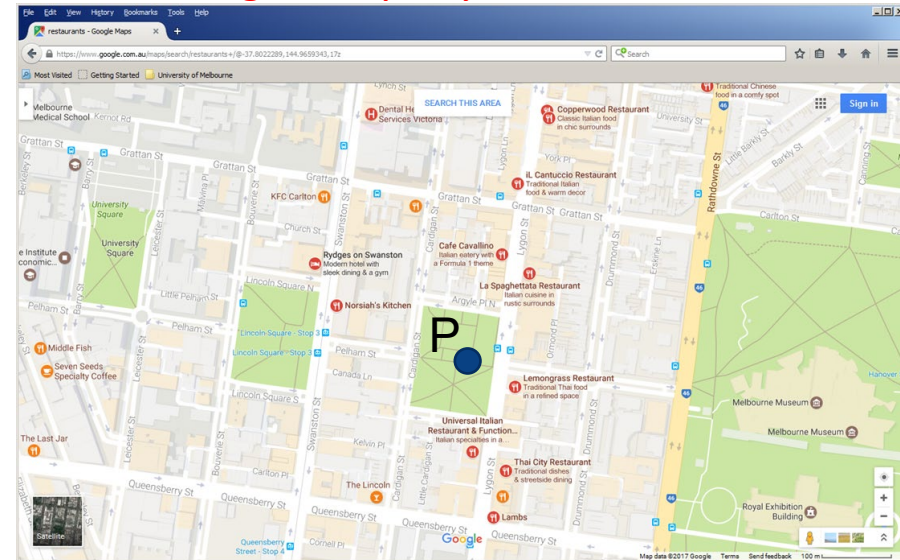
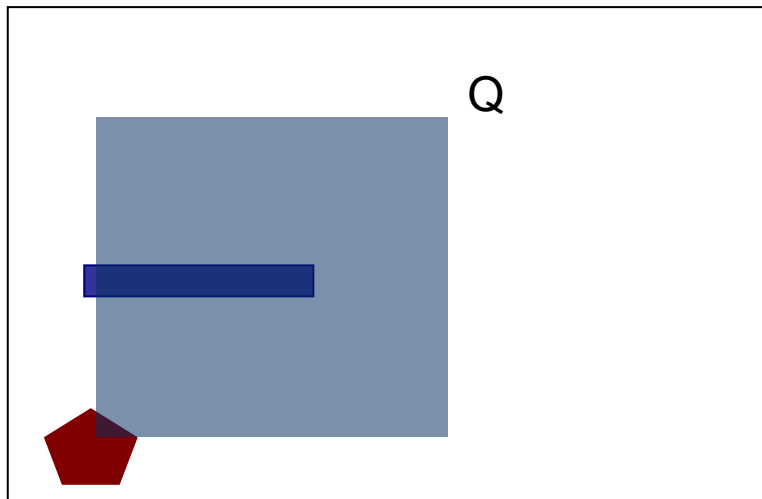
Bitmap for income level	Income_level
L1	1 0 1 0 0
L2	0 1 0 0 0
L3	0 0 0 0 1
L4	0 0 0 1 0
L5	0 0 0 0 0

# Indexing for Other Data Types

Unlike things we can access by names, ids, there is a lot of data that exists, and increasingly that requires special indexing

For example spatial data requires more **complex computations** for accessing data, e.g., intersections of objects in space

There is no trivial way to sort items which is a key issue, e.g., below is a common **range query** on a simple set of items and a **Nearest Neighbor query**

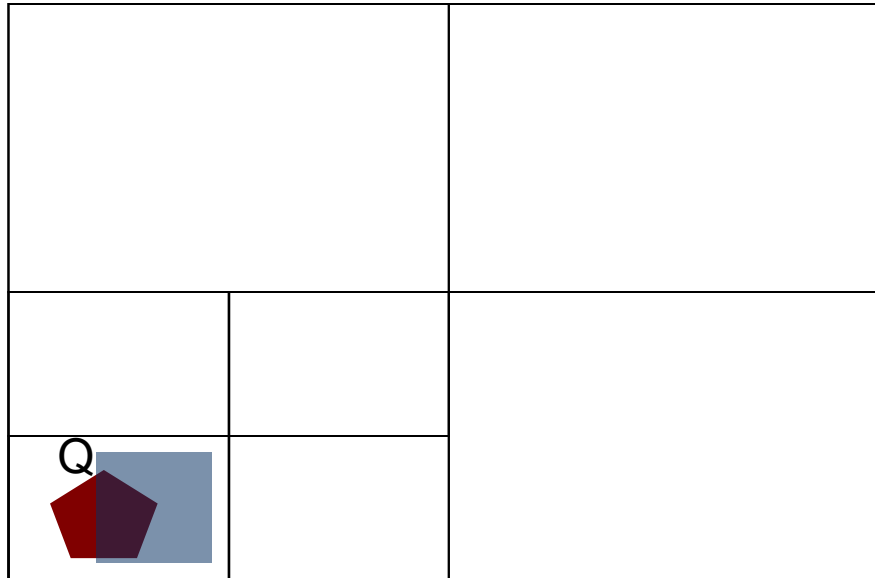


# How do we create buckets then?

In 2-d space, similar to B+trees to **exponentially reduce** the number of calculations by a repetitive division of space, novel indices were invented

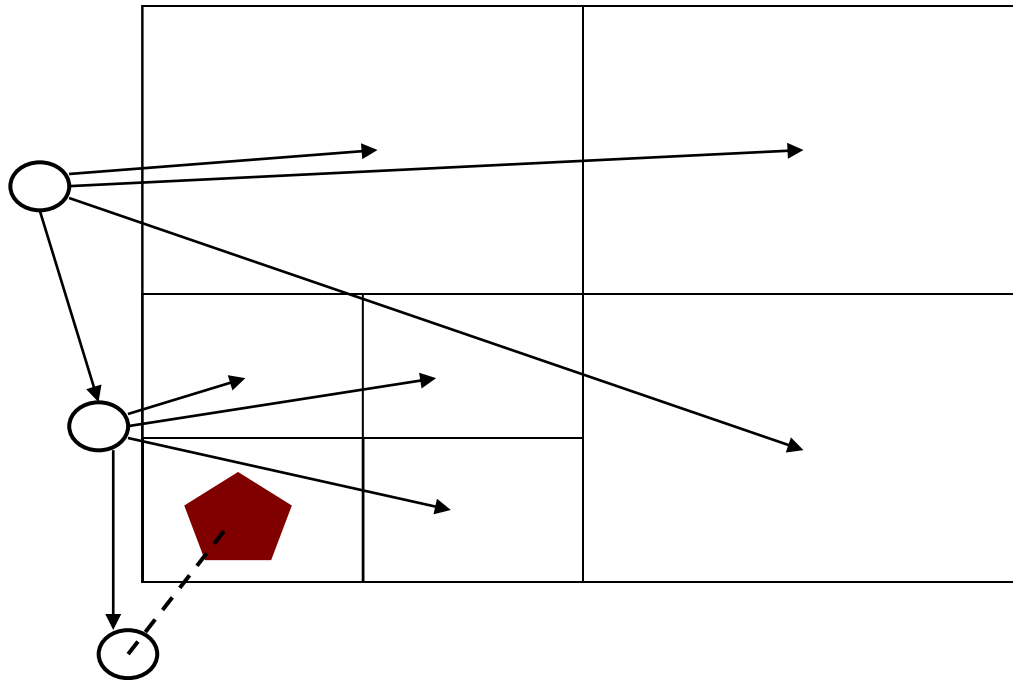
They are in use in Oracle Spatial and other comparable DBMS extensions

The following class of data structures is one such index: **Quadtrees**

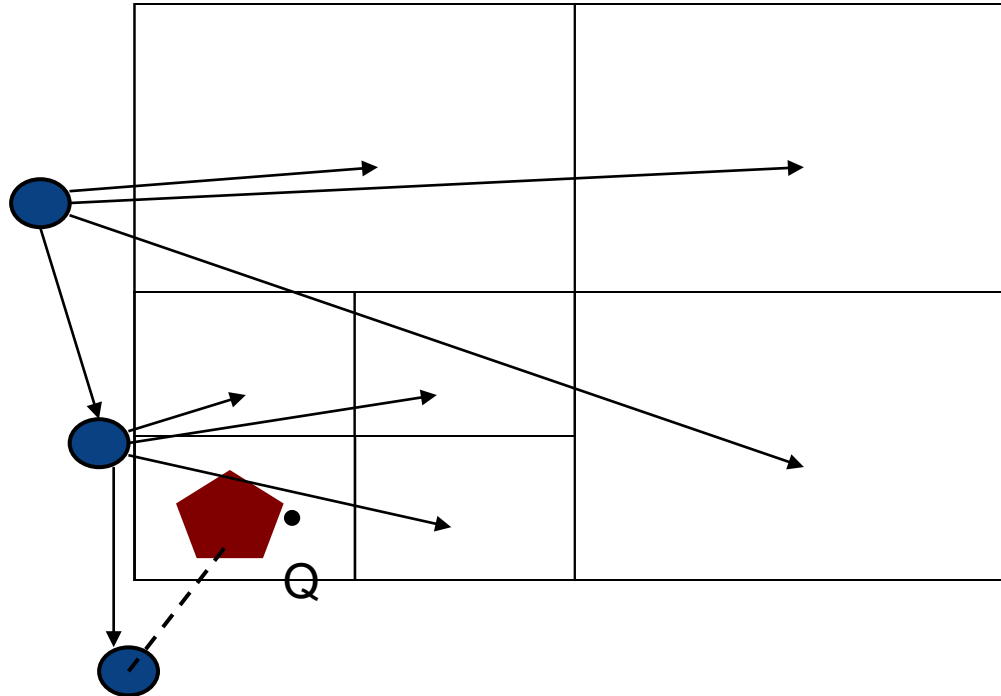




# But hold on where is the tree?



# How to run a NN query?



Basically the same approach, as any other tree does, e.g., Best First Search

Just order access with respect to distance to point

# More on Quadrees

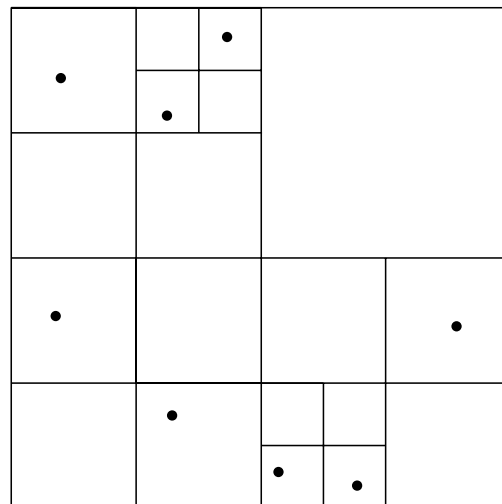
Each **node** of a quadtree is associated with a rectangular region of **space**; the top node is associated with the entire target space.

Each **division** happens with respect to a rule based on data type.

Each non-leaf **nodes** divides its region into four equal sized quadrants

Thus each such node has four child nodes corresponding to the four quadrants and **division continues recursively until a stopping condition**

Example: Leaf nodes have between zero and some fixed maximum number of points (set to 1 in example below)





# R-Trees

**R-trees** are an N-dimensional extension of B<sup>+</sup>-trees, useful for indexing sets of rectangles and other polygons.

Supported in many modern database systems, along with variants like R<sup>+</sup>-trees and R\*-trees.

**Basic idea:** generalize the notion of a one-dimensional interval associated with each B<sup>+</sup>-tree node to an N-dimensional interval, that is, an N-dimensional rectangle.

Will consider only the two-dimensional case ( $N = 2$ )

- generalization for  $N > 2$  is straightforward, although R-trees work well only for relatively small  $N$

*Bounding boxes of children of a node are allowed to overlap*

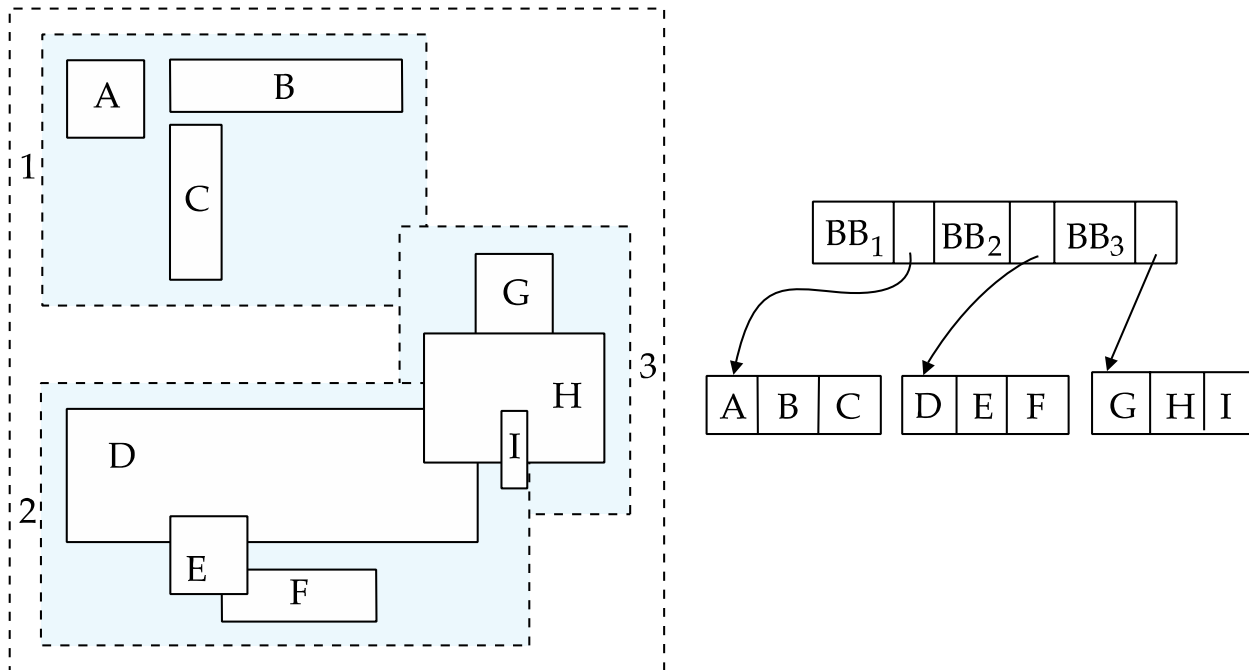
Note: A **bounding box** of a node is a minimum sized rectangle that contains all the rectangles/polygons associated with the node

# Example R-Tree

A set of rectangles (solid line) and the bounding boxes (dashed line) of the nodes of an R-tree for the rectangles.

The R-tree is shown on the right.

The clustering of objects are based on a rule.





# Search in R-Trees

To find data items intersecting a given query point/region, do the following, starting from the root node:


- If the node is a leaf node, output the data items whose keys intersect the given query point/region.
- Else, for each child of the current node whose bounding box intersects the query point/region, recursively search the child.

Can be very inefficient in worst case since multiple paths may need to be searched due to overlaps, but works acceptably in practice.

# Using indexes

- Some indexes are automatically created by the DBMS
  - For UNIQUE constraint, DBMS creates a nonclustered index.
  - For PRIMARY KEY, DBMS creates a clustered index
- You can create indexes on any relation (or view)

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	





# Index Definition in SQL

Create an index

```
create index <index-name> on <relation-name>  
(<attribute-list>)
```

To drop an index

```
drop index <index-name>
```

Most database systems allow specification of type of index, and clustering





# Index Definition in SQL

1. Create a clustered index on a table

```
CREATE CLUSTERED INDEX index1 ON table1 (column1);
```

2. Create a non-clustered index with a unique constraints

```
CREATE UNIQUE INDEX index1 ON table1 (column1 DESC,  
column2 ASC, column3 DESC);
```

(A unique index is one in which no two rows are permitted to have the same index key value)



# Index Definition in SQL

Specialized indexes

## **Filtered index**

```
CREATE INDEX index1 ON table1 (column1)
```

```
WHERE Year > '2010';
```

(A filtered index is an optimized nonclustered index, suited for queries that select a small percentage of rows from a table. It uses a filter predicate to index a portion of the data in the table)

## **Spatial index**

```
CREATE SPATIAL INDEX index_name ON  
table_name(Geometry_type_col_name) WITH ( BOUNDING_BOX = ( 0,  
0, 500, 200 ) );
```



# What you need to do now?

- There is no point going through all index types for all data types
  - There is hundreds of them
  - Even each type would have many subtypes
    - ▶ E.g., MX-CIF quadtree is one quadtree type among many
  - Same with R-trees, etc
  - Same with many other index types
- Given a data set, when uploading to the DBMS
  - Find the potential query types
  - Research what indices that particular DBMS would have for that data type
  - Research for what queries you would better do on what index
  - Create index if you have large data
  - Monitor performance
  - Tune or create other indices
  - Your DMBS will have a version of the “create index” SQL statement