Sr. No.	Module	Detailed Contents	Hrs
1	**Business Intelligence-**	Introduction and overview of BI-Effective and timely decisions, Data Information and knowledge, BI Architecture, Ethics and BI. BI Applications- Balanced score card, Fraud detection, Telecommunication Industry, Banking and finance, Market segmentation.	**06**
2	**Prediction methods and models for BI**	Data preparation, Prediction methods-Mathematical method, Distance methods, Logic method, heuristic method-local optimization technique, stochastic hill climber, evaluation of models	**06**
3	**BI using Data Warehousing**	Introduction to DW, DW architecture, ETL Process, Top-down and bottom-up approaches, characteristics and benefits of data mart, Difference between OLAP and OLTP. Dimensional analysis- Define cubes. Drill- down and roll- up – slice and dice or rotation, OLAP models- ROLAP and MOLAP. Define Schemas- Star, snowflake and fact constellations.	**08**
4	**Data Mining and Preprocessing**	Data mining- definition and functionalities, KDD Process, Data Cleaning: - Missing values, Noisy data, data integration and transformations. Data Reduction: - Data cube aggregation, dimensionality reduction- data compression, Numerosity reduction- discretization and concept hierarchy.	**06**
5	**Associations and Correlation**	Association rule mining:-support and confidence and frequent item sets, market basket analysis, Apriori algorithm, Incremental ARM, Associative classification- Rule Mining.	**06**
6	**Classification and Prediction**	Introduction, Classification methods:-Decision Tree- ID3, CART, Bayesian classification- Baye'stheorem(Naïve Bayesian classification),Linear and nonlinear regression.	**08**
7	**Clustering**	Introduction, categorization of Major, Clustering Methods:- partitioning methods- K-Means. Hierarchical- Agglomerative and divisive methods, Model- based- Expectation and Maximization.	**08**
8	**Web mining and Text**	Text data analysis and Information retrieval, text retrieval methods, dimensionality reduction for text	**04**

Outline

- Clustering
- Types of clustering
  - Hierarchical → Agglomerative algorithm
  - Partition →K-means clustering algorithm
    →Nearest neighbor algorithm
  - Clustering large databases → BIRCH

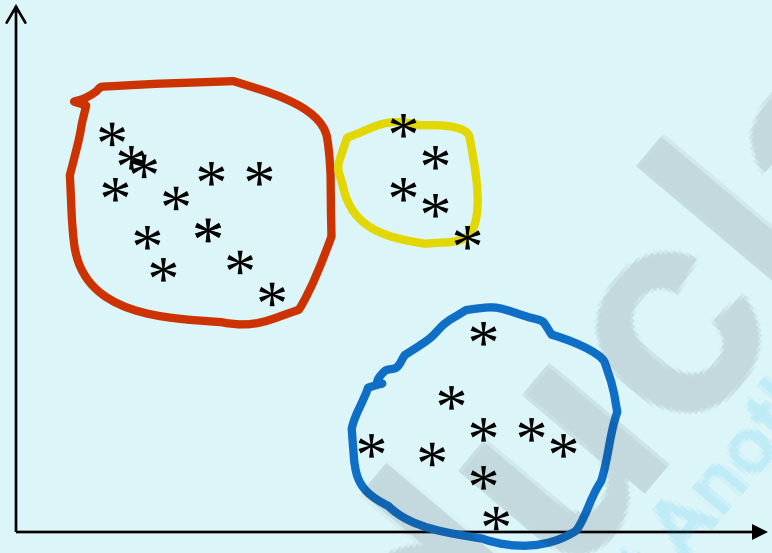
Clustering: Basic Idea

- Groups similar data together into clusters
- Clustering is accomplished by determining the similarity among the data on predefined attributes.
- Most similar data are grouped into clusters
- Partitioning or segmentation

*Clustering is **unsupervised***, because

- No previous categorization known
- Totally data driven
- Classes are formed after examining the data

Clustering : Example



- A good clustering method will produce high quality clusters

Similarity?

- Groups of similar customers
  - Similar demographics
  - Similar buying behavior
  - Similar health
- Similar products
  - Similar cost
  - Similar function
  - Similar store
  - …
- Similarity usually is domain/problem specific

Clustering (Contd.)

- The output of a clustering algorithm consists of a summarized representation of each cluster.
- The type of summarized representation depends strongly on the type & shape of clusters the algorithm computes.

Type of Clustering Algorithms

- A **Hierarchical** clustering algorithm generates a tree of clusters.
- It is categorized as Agglomerative and Divisive.
- Agglomerative start with each object in an individual cluster and nearby clusters are repeatedly merged resulting in larger and larger clusters until some stopping criterion is met or all objects are merged into a single large cluster with highest level of hierarchy.
- Divisive approach starts by putting all data objects into one cluster and repeatedly perform splitting which results in smaller and smaller clusters until a stopping criterion is met or each cluster has only one object in it.

Type of Clustering Algorithms

- A **Partitional** clustering algorithm partitions the data into k groups such that some criterion that evaluates the clustering quality is optimized.
- The number of clusters k is a parameter whose value is specified by the user.

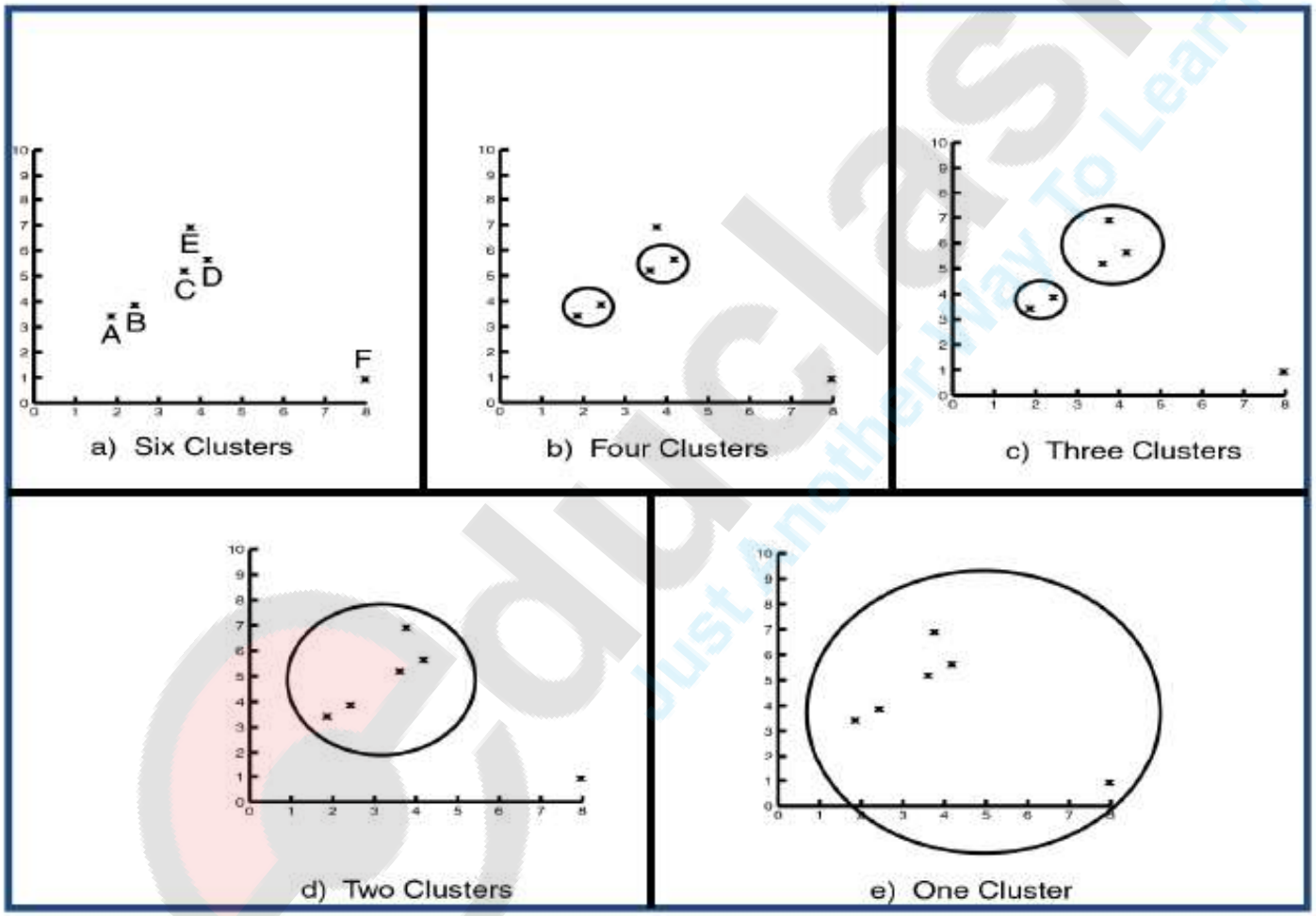
Agglomerative algorithm

1. **Bottom-up approach:**
2. Initially each item x1, . . . , xn is in its own cluster C1, . . . ,Cn.
3. Repeat until there is only one cluster left:

   Merge the nearest clusters, say Ci and Cj .

- The result is a cluster tree. One can cut the tree at any level to produce different clustering.

Levels of Clustering



Agglomerative algorithm

1. Allocate each point to a cluster of its own. So start with n clusters for n objects.
2. Using a distance measure, at each step identify two clusters to be merged.
   - ✓ Single link approach → Find two clusters that have smallest distance between them (find nearest neighbor).
   - ✓ Complete link approach → Find two clusters that have largest distance between them (find farthest neighbor).
   - ✓ There are other approaches too……

Agglomerative algorithm

Single link approach

3. Two clusters are merged if

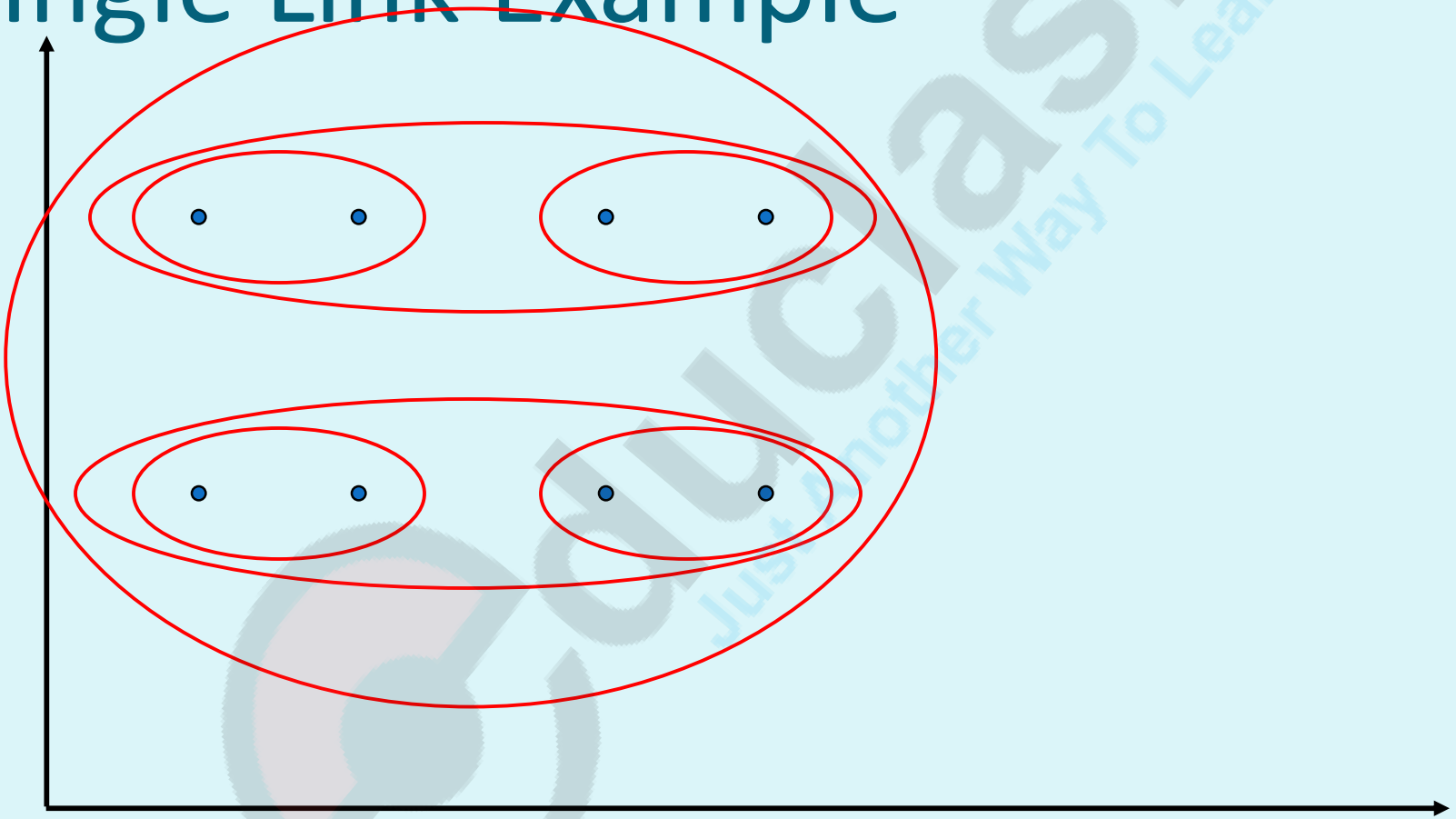
   min dist between any two points <= threshold dist

$$d(c_i, c_j) = \min_{x \in c_i, y \in c_j} d(x, y)$$

4. Identify the pair of objects and merge them.
5. Compute all distances from the new cluster and update the distance matrix after merger.
6. Continue merging process until only one cluster is left or stop merging once the distance between clusters is above the threshold.

- Single-linkage tends to produce stringy or elongated clusters that is clusters are chained together by only single objects that happen to be close together.

Single Link Example



Agglomerative algorithm

Complete link approach

3. Two clusters are merged if

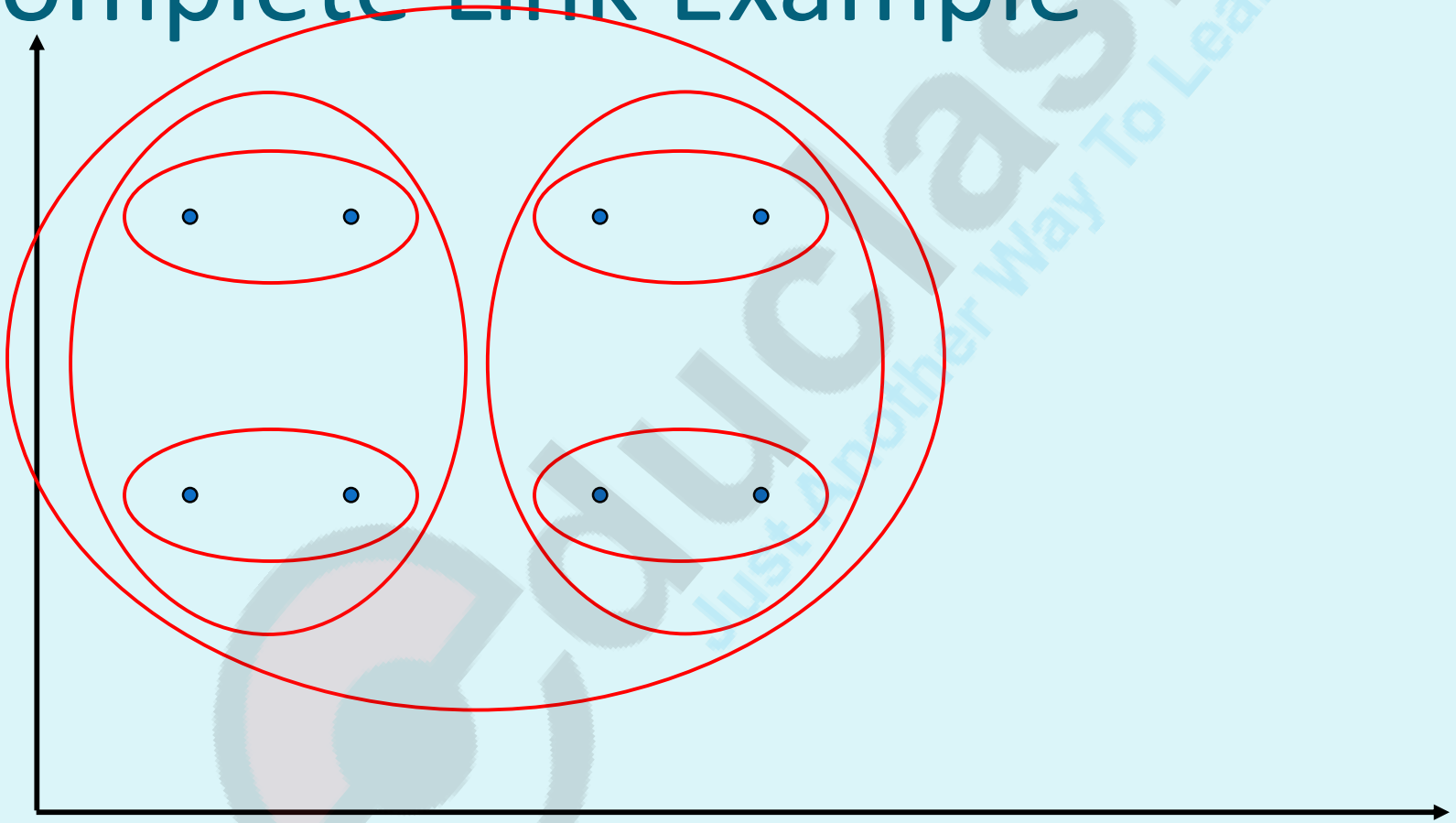
   max dist between any two points <= threshold dist

$$d(c_i, c_j) = \max_{x \in c_i, y \in c_j} d(x, y)$$

4. Identify the pair of objects and merge them.
5. Compute all distances from the new cluster and update the distance matrix after merger.
6. Continue merging process until only one cluster is left or stop merging once the distance between clusters is above the threshold.

- Clusters tend to be compact and roughly equal in diameter.

Complete Link Example

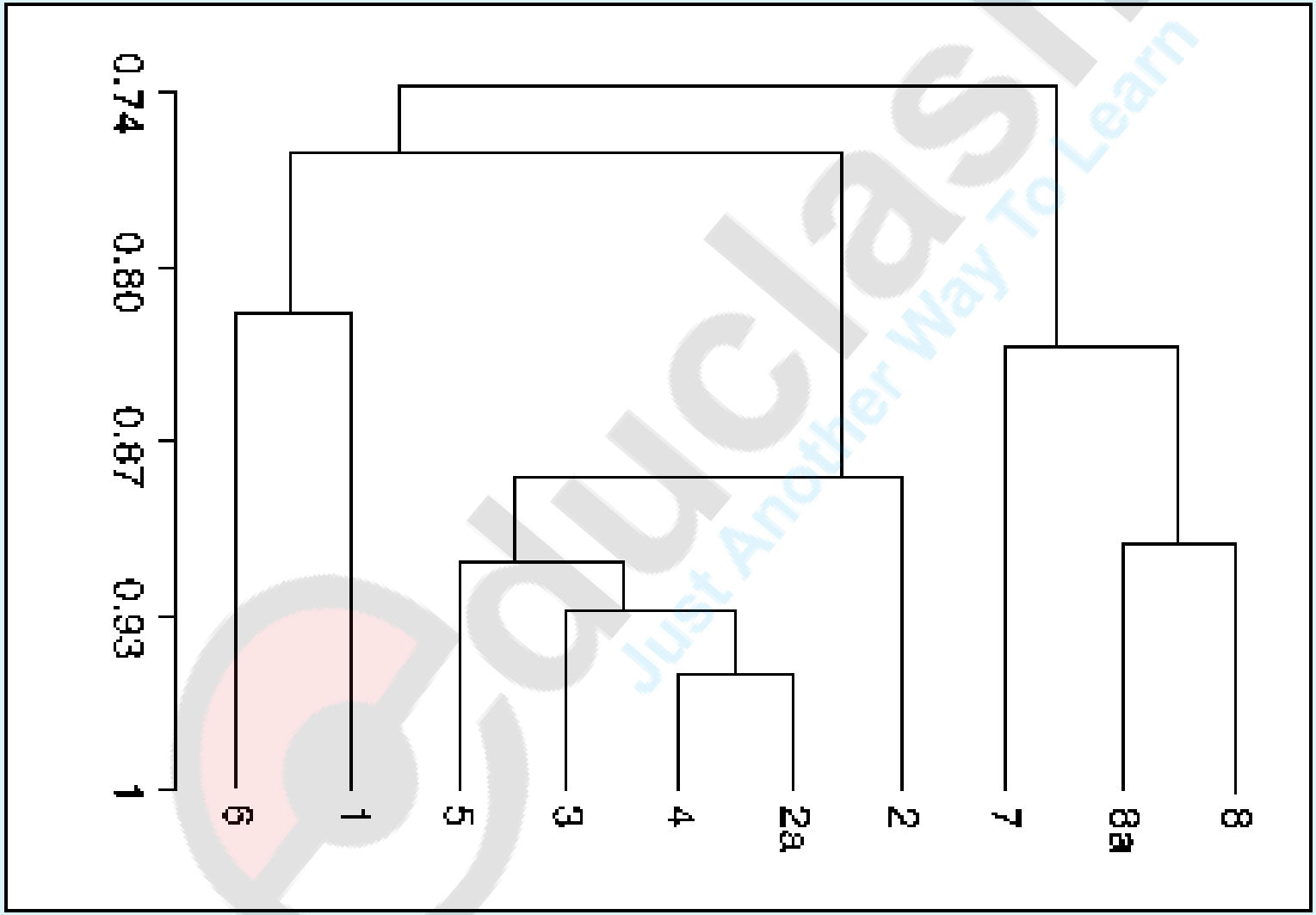


Agglomerative algorithm

Average link approach

$$d(c_i, c_j) = \sum_{x \in c_i, x' \in cj} d(x, x') / |c_i| . |c_j|$$

- Stop merging once the average distance between clusters is above the threshold.
- Somewhere between single-linkage and complete-linkage.
- and a million other ways you can think of …


0.74
0.80
0.87
0.93
1
6
1
5
3
4
10
2
9
7
11
8

```
Input:
    D = {t1, t2, ..., tn}    //Set of elements
    A      //Adjacency matrix showing distance between elements
Output:
    DE     // Dendrogram represented as a set of ordered triples
Agglomerative algorithm:
    d = 0;
    k = n;
    K = {{t1}, ..., {tn}};
    DE = {(d, k, K)}; // Initially dendrogram contains each element
                      in its own cluster.

    repeat
        oldk = k;
        d = d + 1;

        Merge the clusters

        if oldk ≠ k then
            DE = DE ∪ (d, k, K); // New set of clusters added to dendrogram.
            until k = 1
```

Agglomerative algorithm

NOTE:

Start with:

- Initial no of clusters=n.
- Initial value of threshold dist, d = 0.
- Increment d in each iteration.
- Check condition (min dist<=d) for merging.
- Continue until you get one cluster.

- Use single and complete link agglomerative clustering to group the data described by the following distance matrix. Show the dendrograms.

	A	B	C	D
A	0	1	4	5
B		0	2	6
C			0	3
D				0

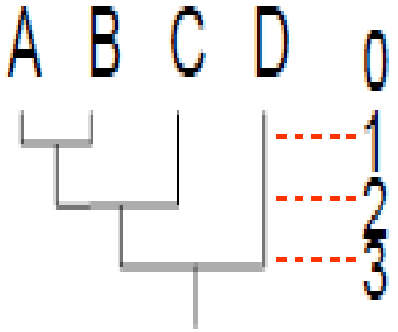
Solution:

Agglomerative ➔ initially every po
one unique cluster containing all poi

	A	B	C	D
A	0	1	4	5
B		0	2	6
C			0	3
D				0

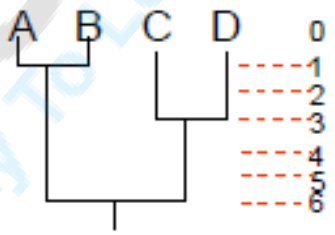
a) single link: distance between two
two clusters.

d	k	K	Comments
0	4	{A}, {B}, {C}, {D}	We start with each point = cluster
1	3	{A, B}, {C}, {D}	Merge {A} and {B} since A & B are the closest: d(A, B)=1
2	2	{A, B, C}, {D}	Merge {A, B} and {C} since B & C are the closest: d(B, C)=2
3	1	{A, B, C, D}	Merge D



complete link: distance between two clusters is the longest distance between a pair of elements from the two clusters

d	k	K	Comments
0	4	{A}, {B}, {C}, {D}	We start with each point = cluster
1	3	{A, B}, {C}, {D}	d(A,B)=1<=1 ➔ merge {A} and {B}
2	3	{A, B}, {C}, {D}	d(A,C)=4>2 so we can't merge C with {A,B} d(A,D)=5>2 and d(B,D)=6>2 so we can't merge D with {A, B} d(C,D)=3>2 so we can't merge C and D
3	2	{A, B}, {C, D}	- d(A,C)=4>3 so we can't merge C with {A,B} - d(A,D)=5>3 and d(B,D)=6>3 so we can't merge D with {A, B} - d(C,D)=3 <=3 so merge C and D
4	2	{A, B}, {C, D}	{C,D} cannot be merged with {A, B} as d(A,D)= 5 >4 (and also d(B,D)= 6 >4) although d(A,C)= 4 <= 4, d(B,C)= 2<=4)
5	2	{A, B}, {C, D}	{C,D} cannot be merged with {A, B} as d(B,D)= 6 > 5
6	1	{A, B, C, D}	{C, D} can be merged with {A, B} since d(B,D)= 6 <= 6, d(A,D)= 5 <= 6, d(A,C)= 4 <= 6, d(B,C)= 2 <= 6



	A	B	C	D
A	0	1	4	5
B		0	2	6
C			0	3
D				0

The K-Means algorithm

1. Select the no of clusters. Let this no be *k*.
2. Pick *k* seeds as centroids of *k* clusters. The seeds may be picked randomly unless the user has some insight into the data
3. Compute the Euclidean distance of each object in the dataset from each of the centroids.
4. Allocate each object to the cluster it is nearest to based on the distances computed in the previous step.
5. Compute the centroids of the clusters by computing the means of the attribute values of the objects in each cluster.
6. Check if the stopping criterion has been met(e.g. cluster membership is unchanged). If not, go to step 3.

Input:

$D = \{t_1, t_2, ..., t_n\}$ // Set of elements

A // Adjacency matrix showing distance between elements.

k // Number of desired clusters.

Output:

K // Set of clusters.

K-Means Algorithm:

assign initial values for means $m_1, m_2, ..., m_k$;

repeat

assign each item t_i to the cluster which has the closest mean ;

calculate new mean for each cluster;

until convergence criteria is met;

K-Means Example

- Given: {2,4,10,12,3,20,30,11,25}, k=2
- Randomly assign means: $m_1=3, m_2=4$
- $K_1=\{2,3\}$, $K_2=\{4,10,12,20,30,11,25\}$, $m_1=2.5, m_2=16$
- $K_1=\{2,3,4\}, K_2=\{10,12,20,30,11,25\}$, $m_1=3, m_2=18$
- $K_1=\{2,3,4,10\}, K_2=\{12,20,30,11,25\}$, $m_1=4.75, m_2=19.6$
- $K_1=\{2,3,4,10,11,12\}, K_2=\{20,30,25\}$, $m_1=7, m_2=25$
- Stop as the clusters with these means are the same.

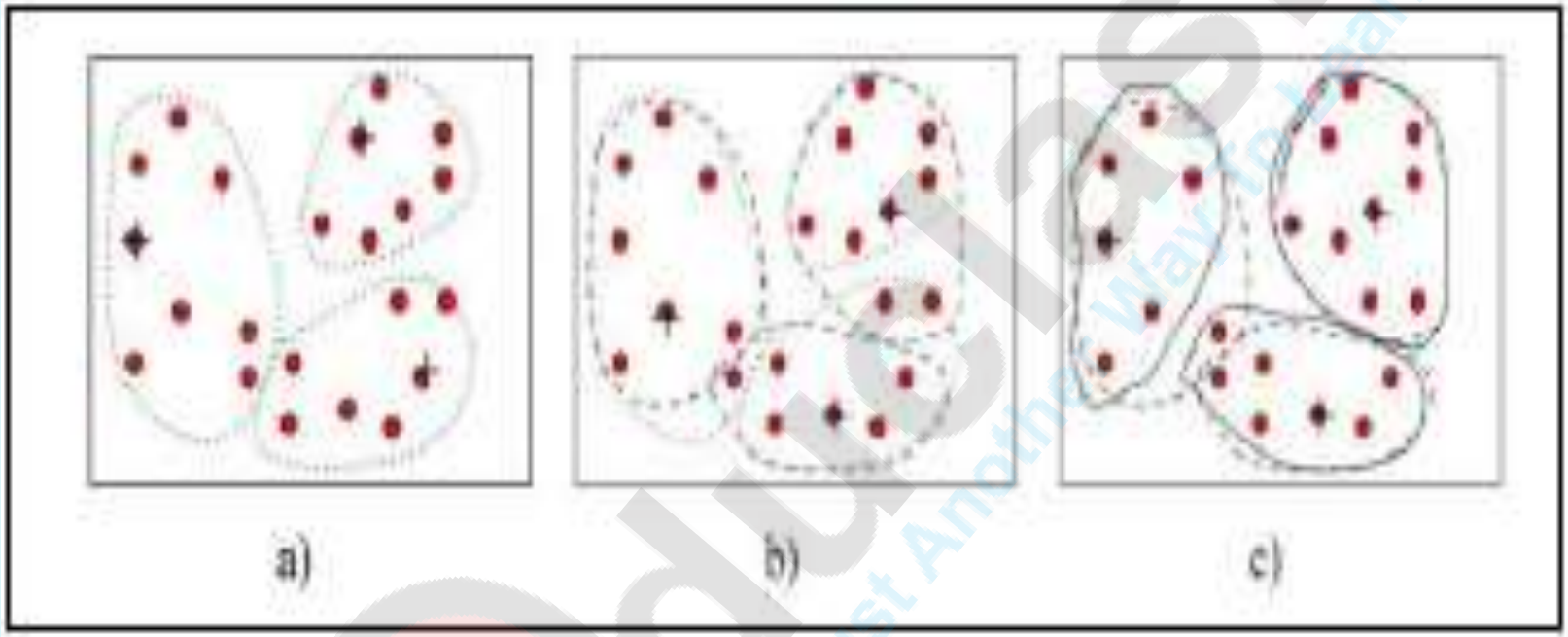


Figure 3. Clustering of a set of objects based on k-means method

Nearest Neighbor algorithm

1. Similar to single link technique.
2. Partitional clustering algorithm.
3. Randomly pick one seed and make a cluster with single item.
4. Items are iteratively merged into the existing clusters that are closest.
5. In this algorithm a threshold, t, is used to determine if items will be added to existing clusters or if a new cluster is created.

Input:

$D = \{t_1, t_2, \ldots, t_n\}$ //Set of elements

A //Adjacency matrix showing distance between elements

Output:

K //Set of clusters

Nearest neighbor algorithm:

$K_1 = \{t_1\};$

$K = \{K_1\};$

$k = 1;$

for $i = 2$ to n **do**

find the t_m in some cluster K_m in K such that dis(t_i, t_m) is the smallest;

if dis$(t_i, t_m), \leq t$ **then**

$K_m = K_m \cup t_i$

else

$k = k + 1;$

$K_k = \{t_i\};$

What Birch algorithm tries to solve?

- Most of the existing algorithms DO NOT consider the case that **datasets can be too large to fit in main memory**
- They DO NOT concentrate on **minimizing the number of scans of** the dataset
- **I/O costs are very high**

The complexity of BIRCH is O(n) where n is the number of objects to be clustered.

Introduction to BIRCH

- Balanced Iterative Reducing and Clustering Using Hierarchies
- Designed for very large data sets
- Time and memory are limited
- Incremental and dynamic clustering of incoming objects
- Only one scan of data is necessary

Two key phases:

- Scans the database to build an in-memory tree
- Applies clustering algorithm to cluster the leaf nodes

Introduction to BIRCH

Basic Idea:

- A tree is built that captures the needed info to perform clustering.
- Clustering is performed on the tree itself, where labeling of nodes in tree contain the needed info to calculate distance value.
- Major characteristics is to use **clustering feature(CF)**
- **CF** is a triple which provides summary of information about one cluster.
- So, BIRCH applies to numeric data.

Clustering Feature

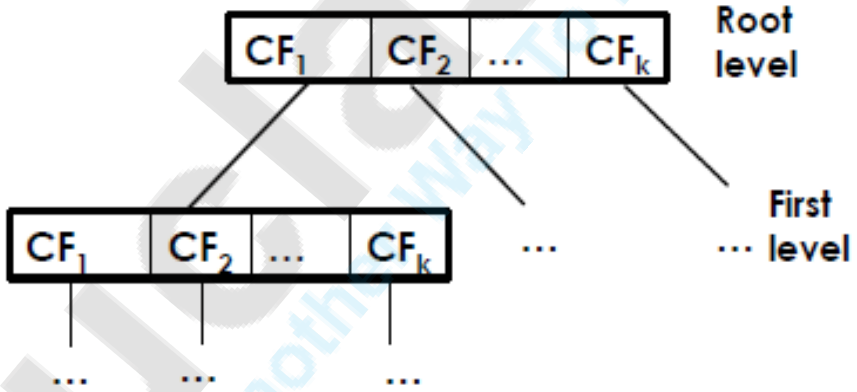
- CF is a triple (N, LS, SS), where N is the number of objects in the cluster and LS, SS are defined in the following.

$$LS = \sum_{P_i \in N} \vec{P}_i$$

$$SS = \sum_{P_i \in N} \left| \vec{P}_i \right|^2$$

CF Tree

- **B** = Branching Factor, maximum children in a non-leaf node
- **T** = Threshold for diameter or radius of the cluster in a leaf
- **L** = number of entries in a leaf

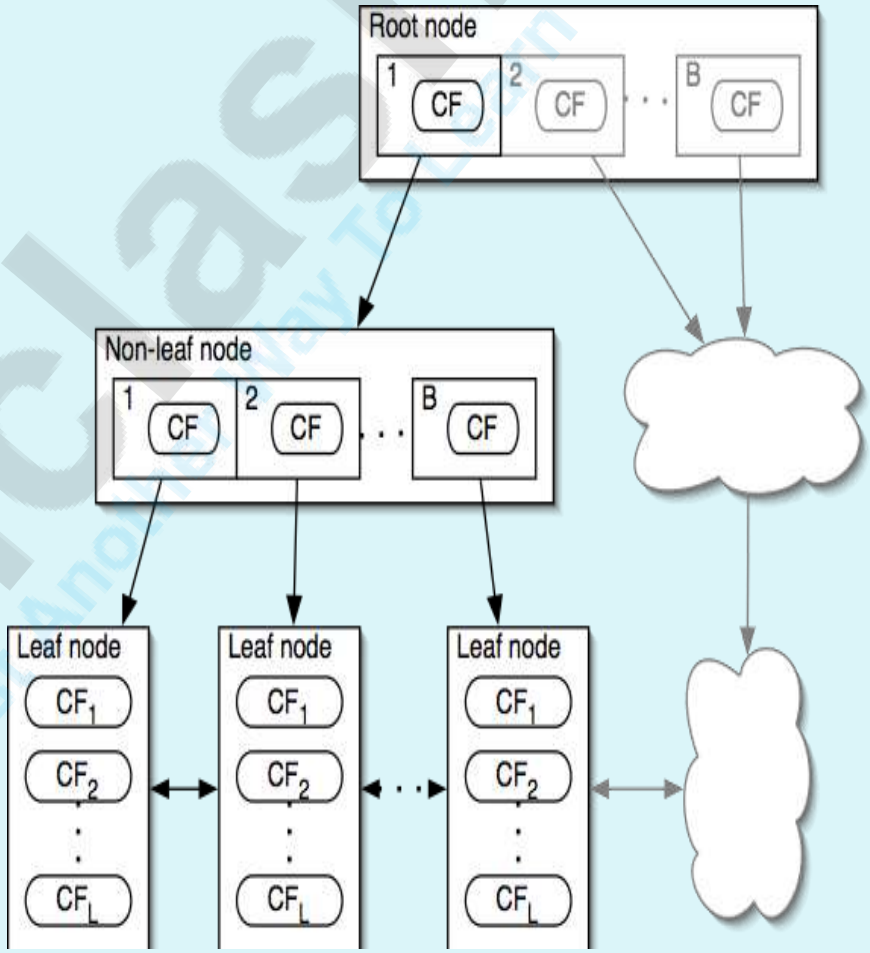


- CF entry in parent = sum of CF entries of a child of that entry

- In-memory, height-balanced tree

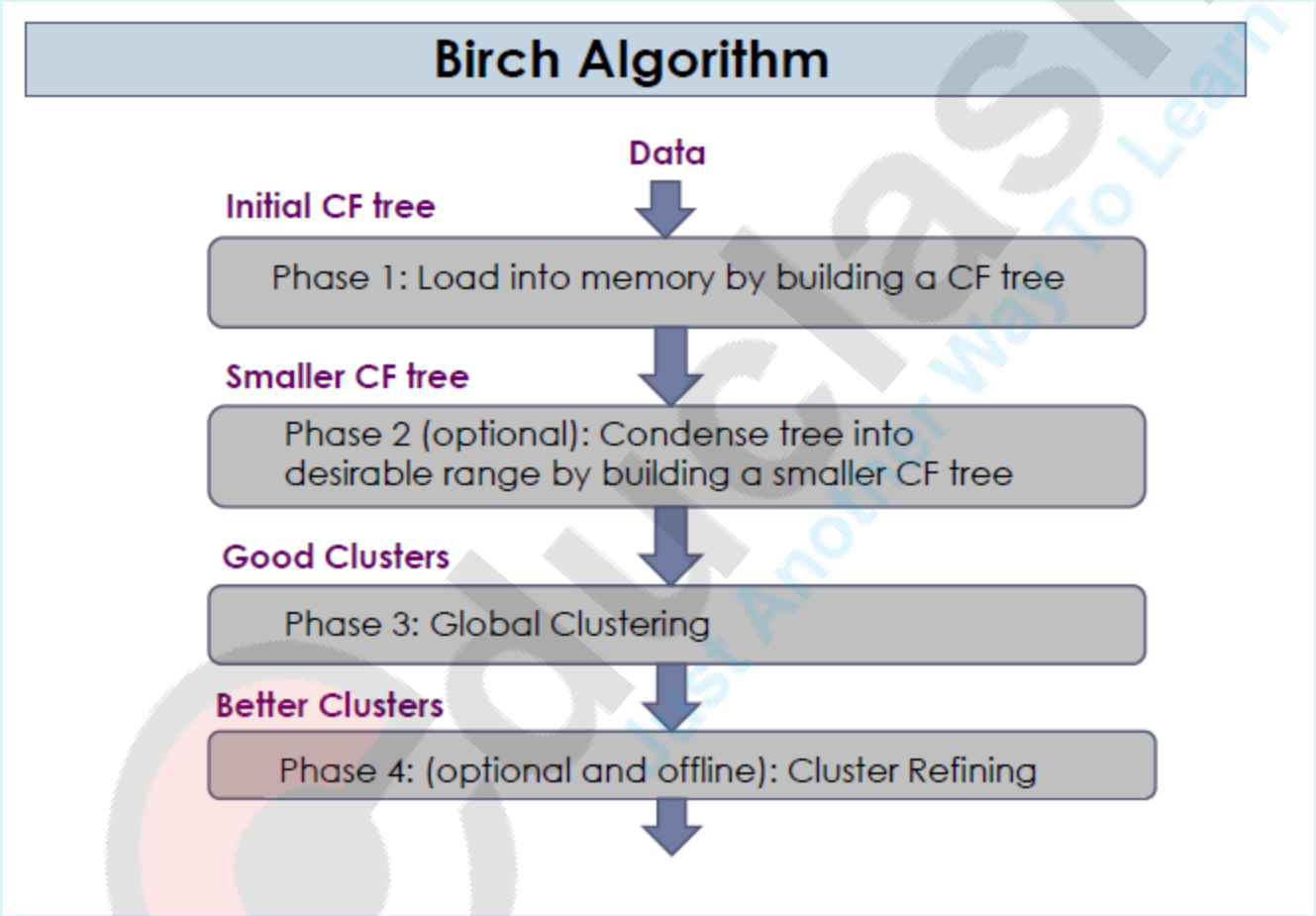
- Each internal node contains a CF triple for each of its children.
- Each leaf node represents a cluster and contain CF entry for each subcluster in it.
- A subcluster is a leaf node must have a diameter not greater than given threshold T.

- *Eg:*
- *Algo:*



CF Tree Insertion

- Start with the root
- Find the CF entry in the root closest to the data point, move to that child and repeat the process until a closest leaf entry is found.
- At the leaf
  - → If the point can be accommodated in the cluster, update the entry
  - → If this addition violates the threshold T, split the entry, if this violates the limit imposed by L, split the leaf. If its parent node is full, split that and so on
- Update the CF entries from the leaf to the root to accommodate this point


Birch Algorithm
Data
Initial CF tree
Phase 1: Load into memory by building a CF tree
Smaller CF tree
Phase 2 (optional): Condense tree into desirable range by building a smaller CF tree
Good Clusters
Phase 3: Global Clustering
Better Clusters
Phase 4: (optional and offline): Cluster Refining

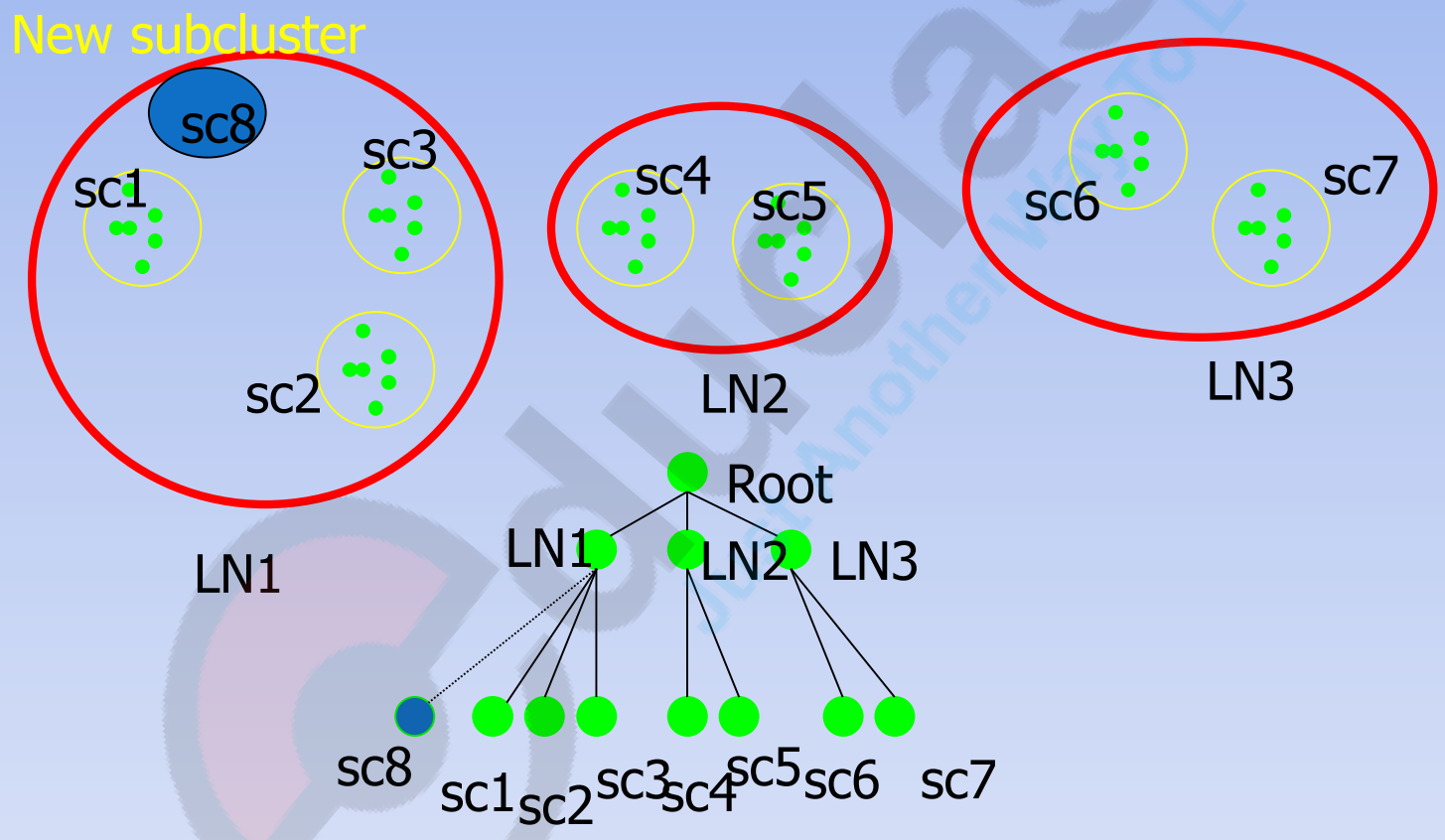
Birch Algorithm: Phase 1

- Choose an initial value for threshold, start inserting the data points one by one into the tree as per the insertion algorithm
- If, in the middle of the above step, the size of the CF tree exceeds the size of the available memory, increase the value of threshold
- Convert the partially built tree into a new tree
- Repeat the above steps until the entire dataset is scanned and a full tree is built
- Outlier Handling

Birch Algorithm: Phase 2,3, and 4

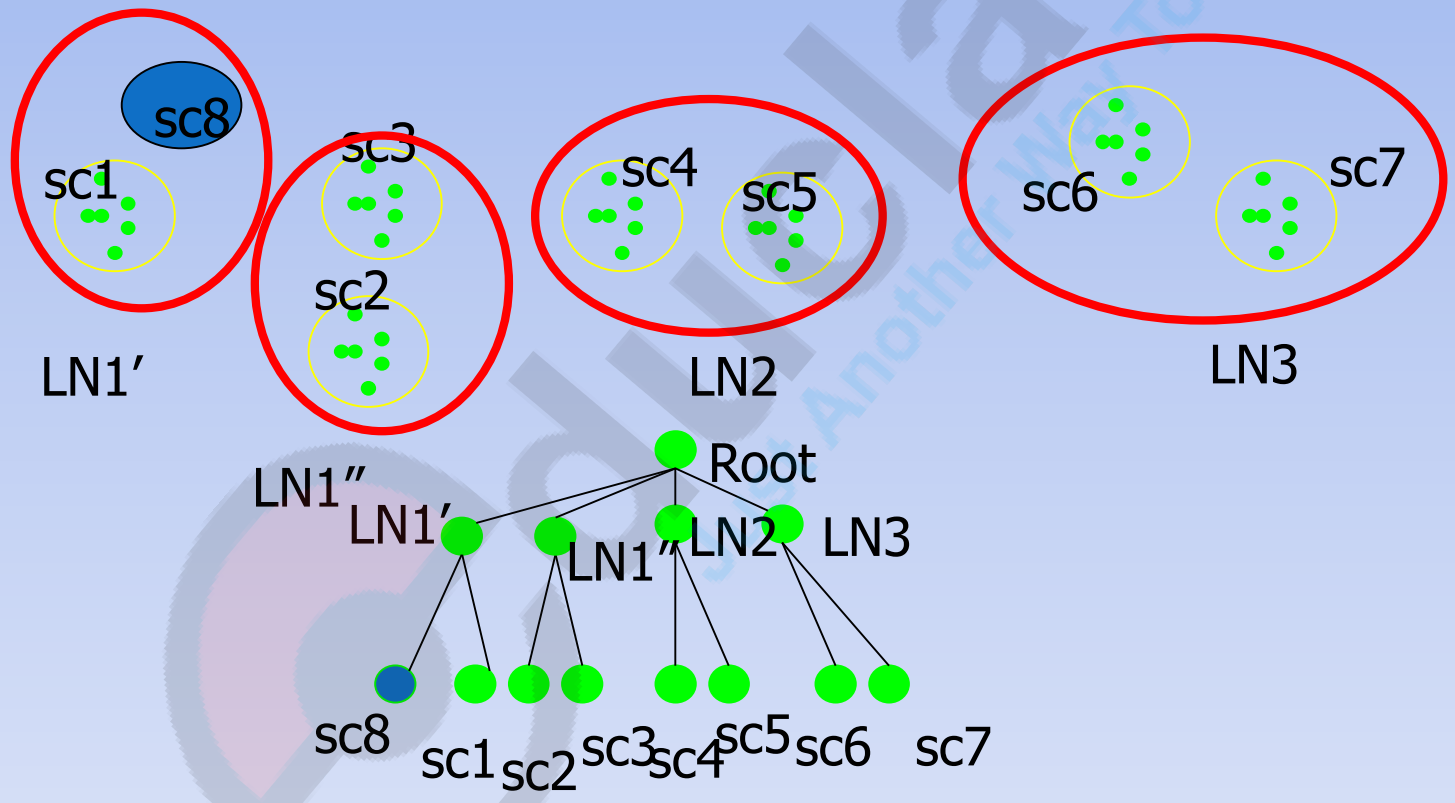
- **Phase 2**
  - → A bridge between phase 1 and phase 3
  - → Builds a smaller CF tree by increasing the threshold
- **Phase 3**
  - → Apply global clustering algorithm to the sub-clusters given by leaf entries of the CF tree
  - → Improves clustering quality
- **Phase 4**
  - → Scan the entire dataset to label the data points
  - → Outlier handling

Example of BIRCH

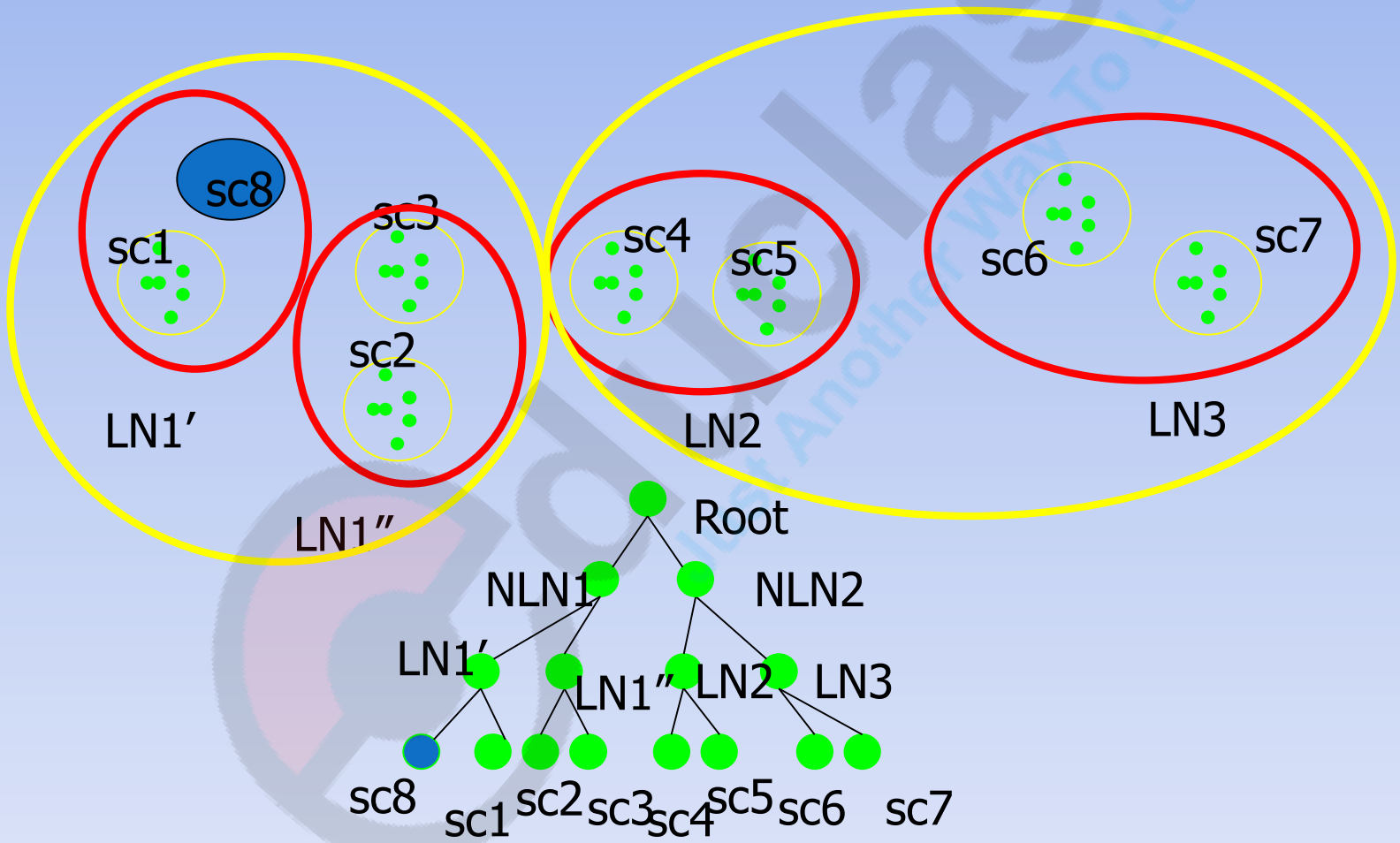


Insertion Operation in BIRCH

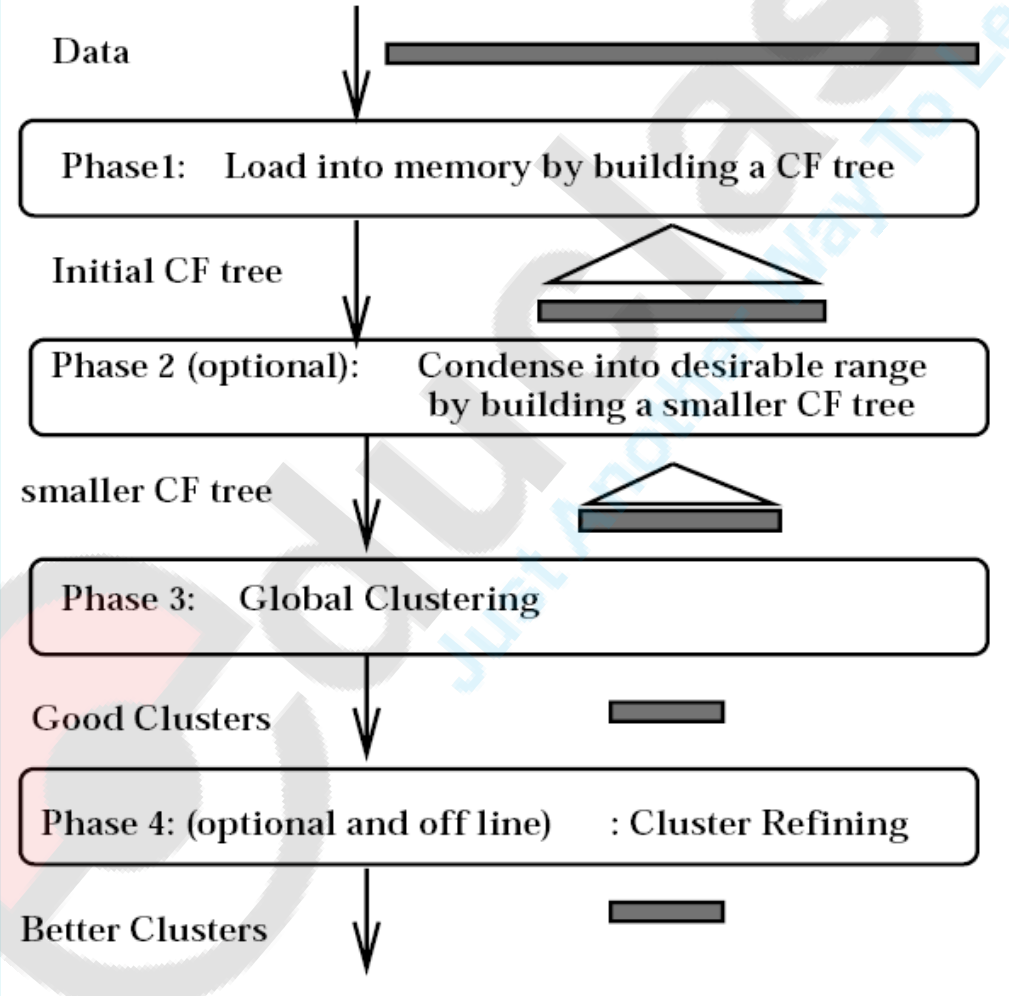
If the branching factor of a leaf node can not exceed 3, then LN1 is split.



If the branching factor of a non-leaf node can not exceed 3, then the root is split and the height of the CF Tree increases by one.



BIRCH Overview



Experimental Results

- Input parameters:
  - Memory (M): 5% of data set
  - Disk space (R): 20% of M
  - Distance equation: D2
  - Quality equation: weighted average diameter (D)
  - Initial threshold (T): 0.0
  - Page size (P): 1024 bytes

Experimental Results

KMEANS clustering

DS	Time	D	# Scan	DS	Time	D	# Scan
1	***43.9***	***2.09***	***289***	*1o*	*33.8*	*1.97*	*197*
2	*13.2*	*4.43*	*51*	*2o*	*12.7*	*4.20*	*29*
3	*32.9*	*3.66*	*187*	*3o*	*36.0*	*4.35*	*241*

BIRCH clustering

DS	Time	D	# Scan	DS	Time	D	# Scan
1	***11.5***	***1.87***	***2***	*1o*	*13.6*	*1.87*	*2*
2	*10.7*	*1.99*	*2*	*2o*	*12.1*	*1.99*	*2*
3	*11.4*	*3.95*	*2*	*3o*	*12.2*	*3.99*	*2*

Conclusions

- A CF tree is a height-balanced tree that stores the clustering features for a hierarchical clustering.
- Given a limited amount of main memory, BIRCH can minimize the time required for I/O.
- BIRCH is a scalable clustering algorithm with respect to the number of objects, and good quality of clustering of the data.

Exam Questions

- **What is the main limitation of BIRCH?**
  - Since each node in a CF tree can hold only a limited number of entries due to the size, a CF tree node doesn't always correspond to what a user may consider a nature cluster. Moreover, if the clusters are not spherical in shape, it doesn't perform well because it uses the notion of radius or diameter to control the boundary of a cluster.

Exam Questions

- **Name the two algorithms in BIRCH clustering:**
  - CF-Tree Insertion
  - CF-Tree Rebuilding
- **What is the purpose of phase 4 in BIRCH?**
  - Do additional passes over the dataset and reassign data points to the closest centroid .