CS5344

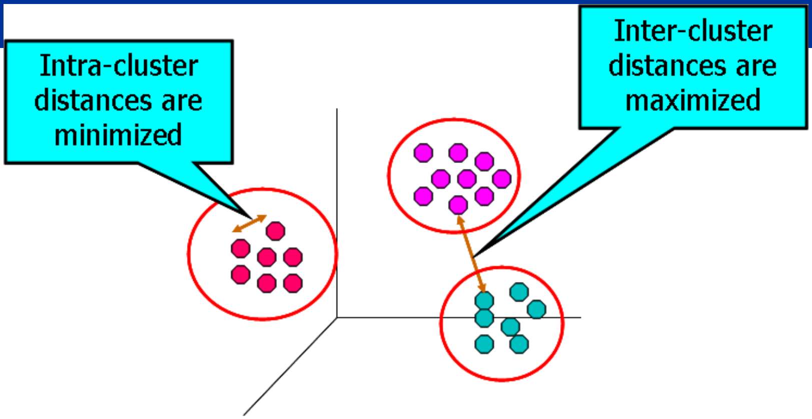
Clustering High Dimensional Data

Acknowledgement: Slides adapted from http://www.mmds.org

High Dimensional Data

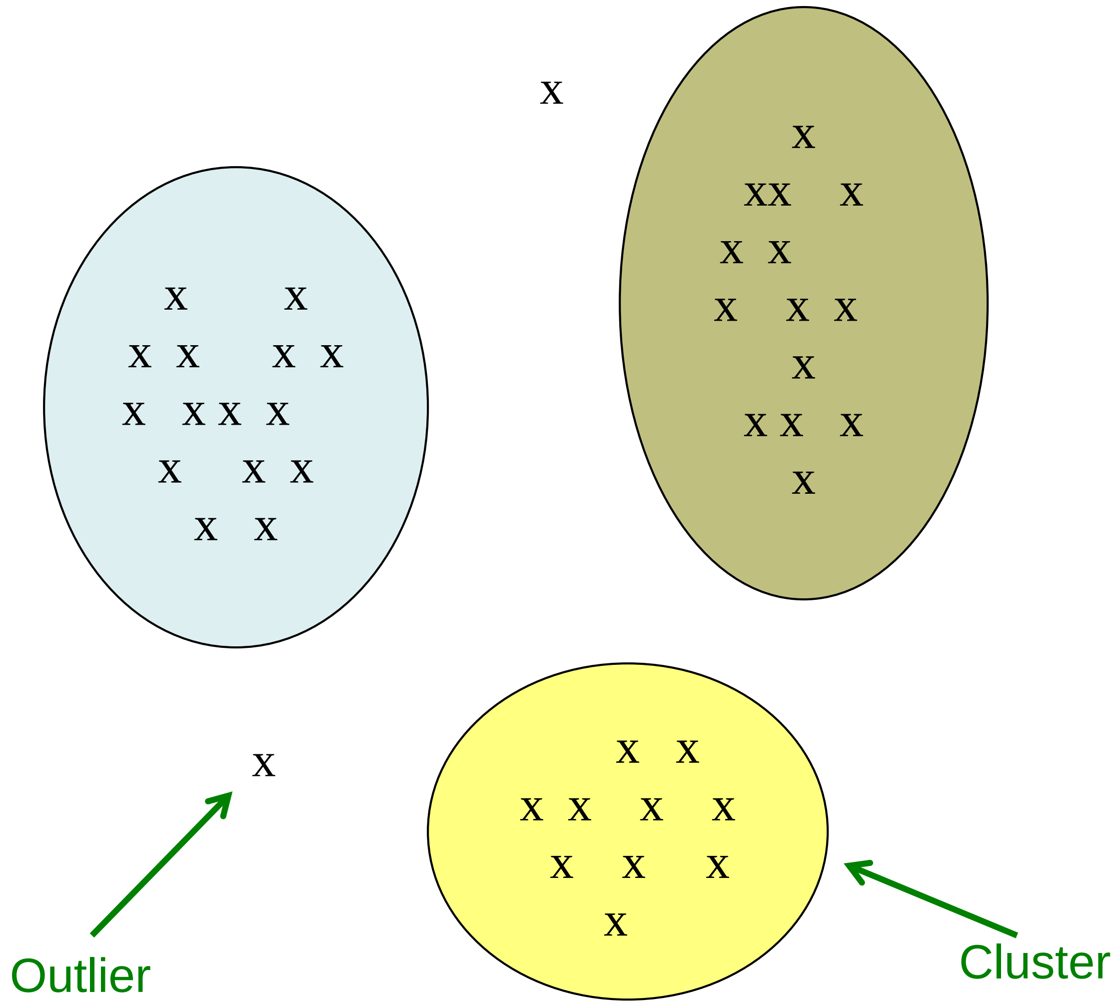
Given a cloud of data points we want to understand its structure

Problem of Clustering

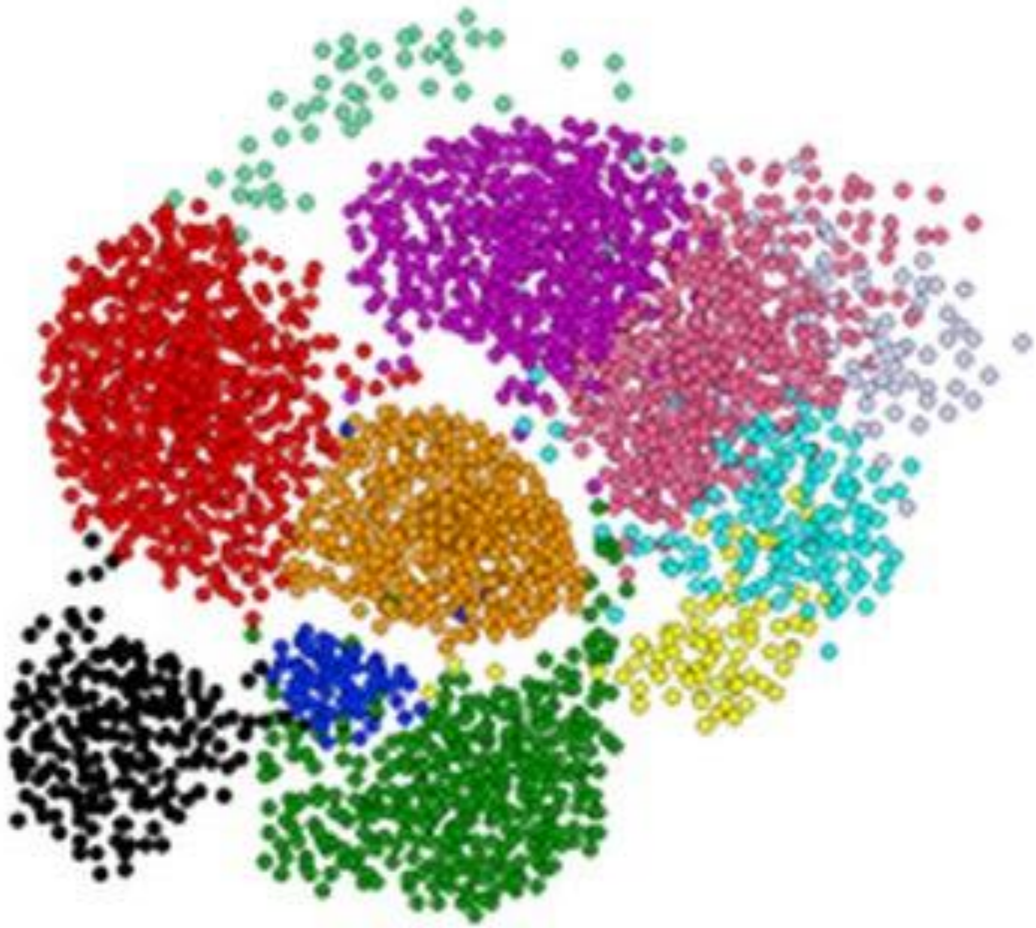


- **Given a <u>set of points</u>, with a notion of <u>distance</u> between points, <u>group the points</u> into some number of *clusters* such that**
  - Members of a cluster are similar to each other
  - Members of different clusters are dissimilar
- **Points are in a high-dimensional space**
  - Each dimension corresponds to a feature/attribute
- **Similarity is defined using a distance measure**
  - Euclidean, Cosine, Jaccard, Edit distance, …

Example: Clusters & Outliers



Clustering is a Hard Problem!



Why is Clustering Hard?

- **Clustering in two dimensions looks easy**
- **Clustering small amounts of data looks easy**
- **BUT**
  - Number of clusters is typically not known
  - Clusters may be of arbitrary shapes and sizes
  - Quality of clustering result
    - Depends on similarity measure and method
    - Measured by its ability to find some or all hidden patterns
- **Many applications operate in high dimensional space (not 2, but 10 or 10000 dimensions)**
  - Almost all pairs of points are at about the same distance!

Application: Galaxies

- Catalog of 2 billion “sky objects” represents objects by their radiation in 7 dimensions (frequency bands)
- Task: Cluster into similar objects, e.g., galaxies, nearby stars, quasars, etc.
- Sloan Digital Sky Survey

Application: Music CDs

- **Music divided into categories, and customers prefer a few categories. What are categories actually?**
- **Represent a CD by a set of customers who bought it**
  - Similar CDs have similar sets of customers
- **Task: Find clusters of similar CDs**
  - Space of all CDs, one dimension for each customer
  - Values in a dimension may be 0 or 1
  - A CD is a point in this space $(x_1, x_2, \ldots, x_k)$ where $x_i = 1$ iff the i^{th} customer bought the CD
- **For Amazon, the dimension is tens of millions**

Application: Cluster Documents

- Task: Find topics
- Represent each document as a vector $(x_1, x_2, \ldots, x_k)$ where $x_i = 1$ iff the word v_i appears in the document
- Documents with similar sets of words may be about the same topic

Overview

- **Distance Measures**
- **Clustering Algorithms**
  - Hierarchical or agglomerative
  - Point assignment or *k*-means
- **Scaling Up Clustering Algorithms**
  - BFR
  - CURE

Distance Measure

- **Each clustering problem is based on some notion of distance between objects or points**
- **Euclidean Distance**
  - Based on <u>locations</u> of points in a *d*-dimensional space
  - Points are vectors of real numbers
  - Length of vector is *d*
- **Non-Euclidean Distance**
  - Based on the <u>properties</u> of points

Distance Measure

- **Distance measure is a function d(x,y) between two points x and y which satisfies the following axioms**

1. Non-negativity: $d(x, y) \geq 0$
2. Identity: $d(x, y) = 0$ if and only if $x = y$
3. Symmetry: $d(x, y) = d(y, x)$
4. Triangle Inequality: $d(x, y) \leq d(x, z) + d(z, y)$

Examples of Distance Measure

- **L_p-norm**

$$d(\boldsymbol{x}, \boldsymbol{y}) = \left(\sum_i (x_i - y_i)^p\right)^{\frac{1}{p}}$$

- **L_2-norm = Euclidean Distance**
  - Square the distance in each dim, sum the squares and take sqrt
  - E.g. two points in n-dimensional space
  - $d(x,y) = \text{sqrt}\ ((x_1 – y_1)^2 + (x_2 – y_2)^2 + \ldots + (x_n – y_n)^2)$

- **L_1-norm = Manhattan Distance**
  - Sum of the magnitude of differences in each dimension
  - $d(x,y) = |x_1 – y_1| + |x_2 – y_2| + \ldots + |x_n – y_n|$
  - Distance to travel if constrained along grid lines

Examples of Distance Measure

- **Cosine Distance**
  - Points are vectors with integer or boolean values

$$cosine(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

Examples of Distance Measure

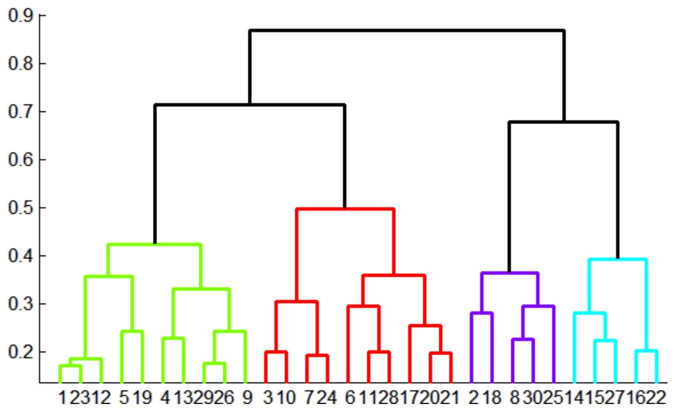
- **Jaccard Distance**
  - If A and B are two sets, then

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

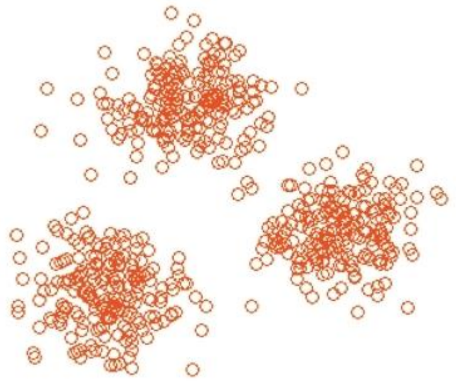
- **Edit Distance**
  - Points are strings
  - Miniumum number of inserts and deletes of characters needed to convert one string into another
  - E.g. if *x* = abcde and *y* = acfdeg, d(x,y) = 3

Clustering Strategies

- **Hierarchical or Agglomerative**
  - Each point is a cluster
  - Repeatedly combine the two "nearest" clusters into one

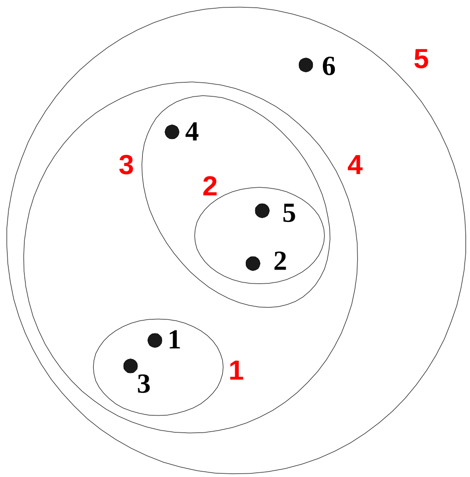


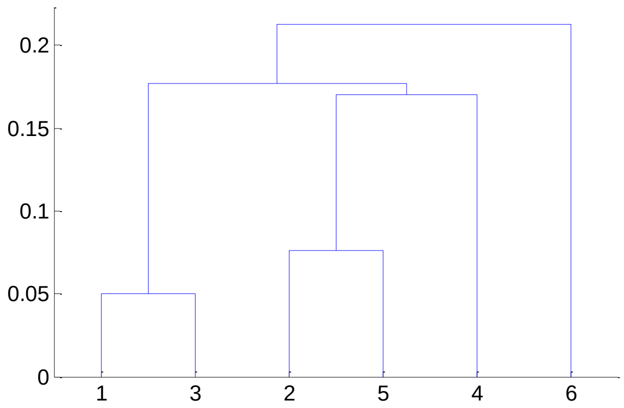
- **Point Assignment**
  - Maintain a set of clusters
  - Points belong to "nearest" cluster



Hierarchical Clustering

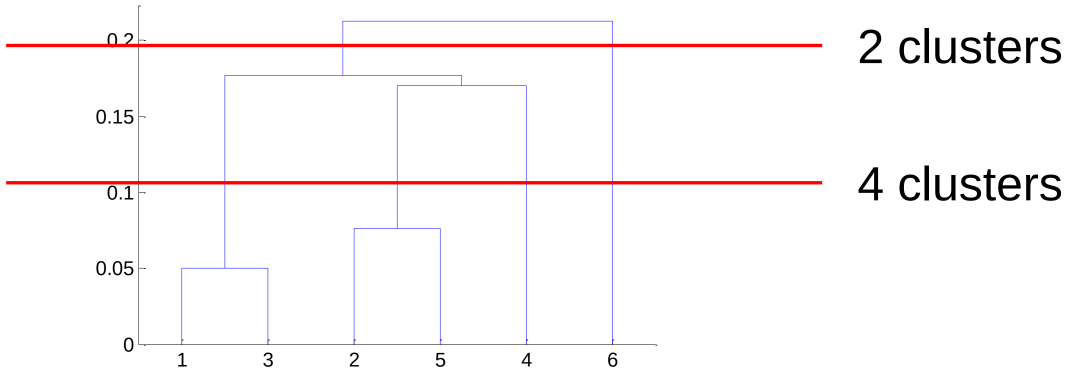
- Key operation: Repeatedly combine two nearest clusters
- Produce a set of nested clusters as a hierarchical tree
- Visualized as a dendrogram (tree-like diagram records the sequence of merges)





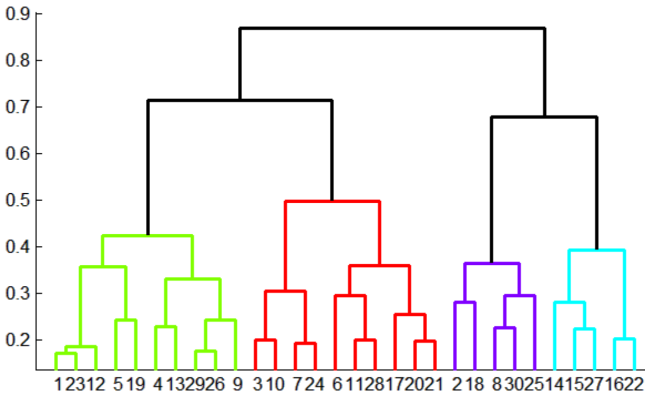
Hierarchical Clustering

- **Does not assume any number of clusters**
- **Any desired number of clusters can be obtained by 'cutting' the dendrogram at the proper level**
- **May correspond to meaningful taxonomies**
  - e.g., animal kingdom, phylogeny reconstruction in biological sciences



Hierarchical Clustering

- **Key operation: Repeatedly combine two nearest clusters**
- **Three important questions:**
  1. How to represent a cluster of more than one point?
  2. How to determine the “nearness” of clusters?
  3. When to stop combining clusters?

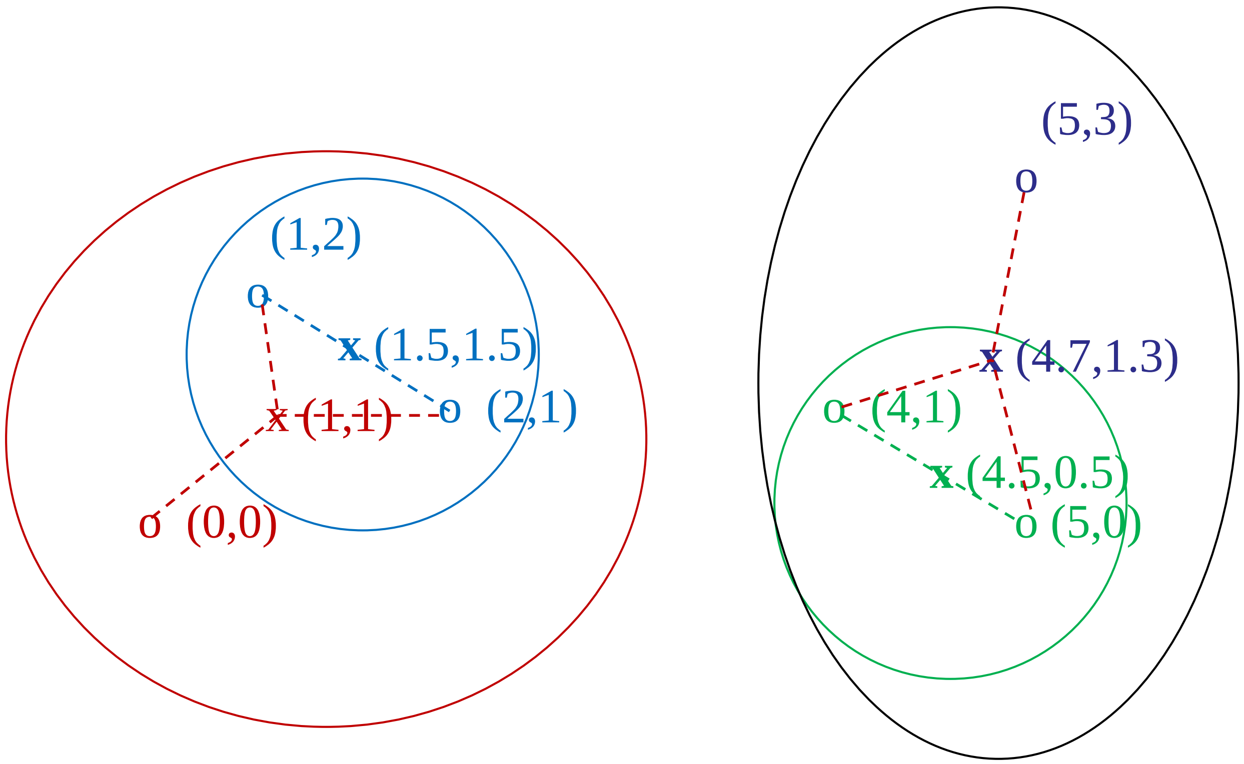


How to Represent Cluster?

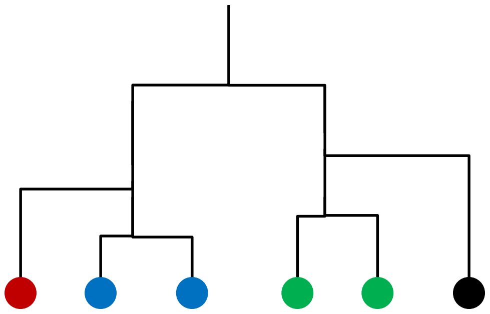
- **Each point is a cluster initially**
- **As we merge clusters, how to represent the “location” of each cluster?**
  - Need this information to know which pair of clusters is closest
- **Euclidean space**
  - Each cluster has a **centroid** = average of its (data) points

Which 2 Clusters to Merge?

- **Euclidean space**
  - Measure cluster distances by distances of centroids



o … data point
x … centroid



Dendrogram

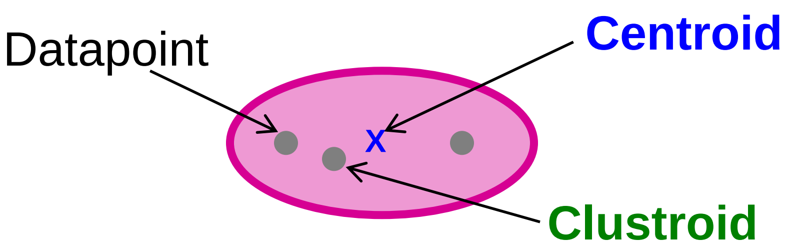
Non-Euclidean Space

- **There is no "average" of two points**
- **The only "locations" are the points themselves**
- **Clustroid = (data) point "closest" to other points**
  - Treat clustroid as if it were centroid, when computing inter-cluster distances
- **Possible meanings of "closest"**
  - Smallest maximum distance to other points
  - Smallest average distance to other points
  - Smallest sum of squares of distances to other points

Non-Euclidean Space

- **For distance metric *d*, clustroid *c* of cluster *C* is**

$$\min_{c} \sum_{x \in C} d(x,c)^2$$



Cluster on 3 datapoints

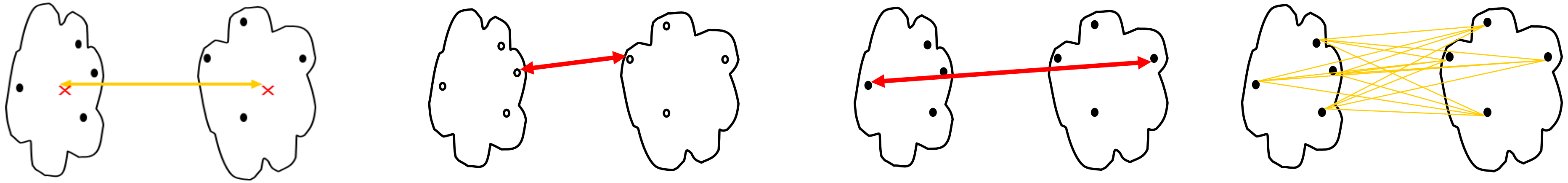
Centroid is the avg. of all (data)points in the cluster. This means centroid is an "artificial" point.

Clustroid is an **existing** (data)point that is "closest" to all other points in the cluster.

“Nearness” of Clusters

Different measures

1. **Based on distance between clustroids/centroids**
2. **Intercluster distance = minimum of the distances between any two points, one from each cluster**
3. **Pick a notion of “cohesion” of clusters and merge cluster whose union is most cohesive**
   - **Diameter** of merged cluster = maximum distance between points in the cluster
   - **Average distance** between points in the cluster
   - **Density-based** (take diameter or avg. distance, and divide by number of points in cluster)



When to Terminate?

- **Pre-determined *number of clusters***
- **When merging two clusters leads to a “bad” cluster**
  - Diameter of merged cluster exceeds some threshold
  - Diameter exceeds average diameter by a wide margin
  - Density of cluster falls below some threshold

Implementation

- **Naïve implementation of hierarchical clustering**
  - At each step, compute pairwise distances between all pairs of clusters, then merge
  - O(n^3)
- **Careful implementation using Priority Queue can reduce time to O(n^2 log n)**
  - **Still too expensive for really big datasets that do not fit in memory**

K–Means Algorithm

- **Assumes Euclidean space/distance**
- **Assumes number of clusters *k* is given**
- **Initialize clusters by picking one point per cluster**
  - Pick one point at random, then *k*-1 other points, each **as far away as possible** from the previous points
  - Make these points the centroid of their clusters

K–Means Algorithm

1. **For each point, place it in the cluster whose current centroid it is nearest**

2. **After all points are assigned, update the locations of centroids of the *k* clusters**

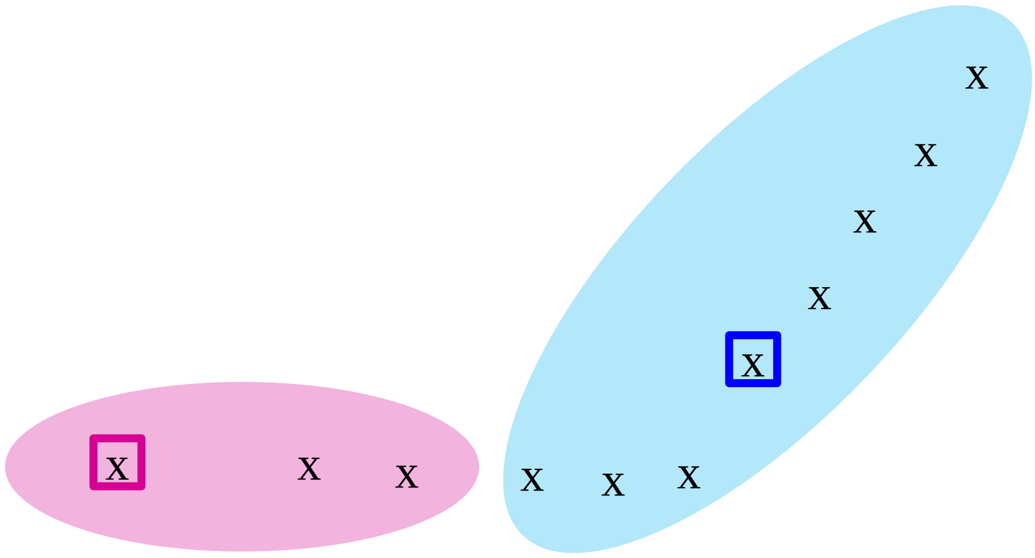
3. **Reassign all points to their closest centroid**
   - Sometimes moves points between clusters

Repeat 2 and 3 until convergence

- **Convergence:** Points don't move between clusters and centroids stabilize

Example

Clusters after Round 1

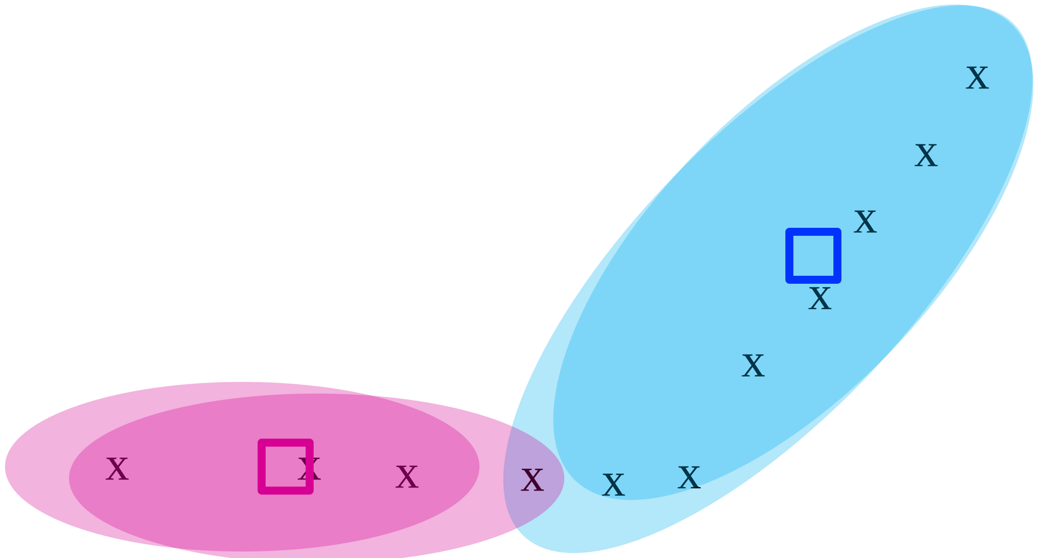


x … data point

□ … centroid

Example

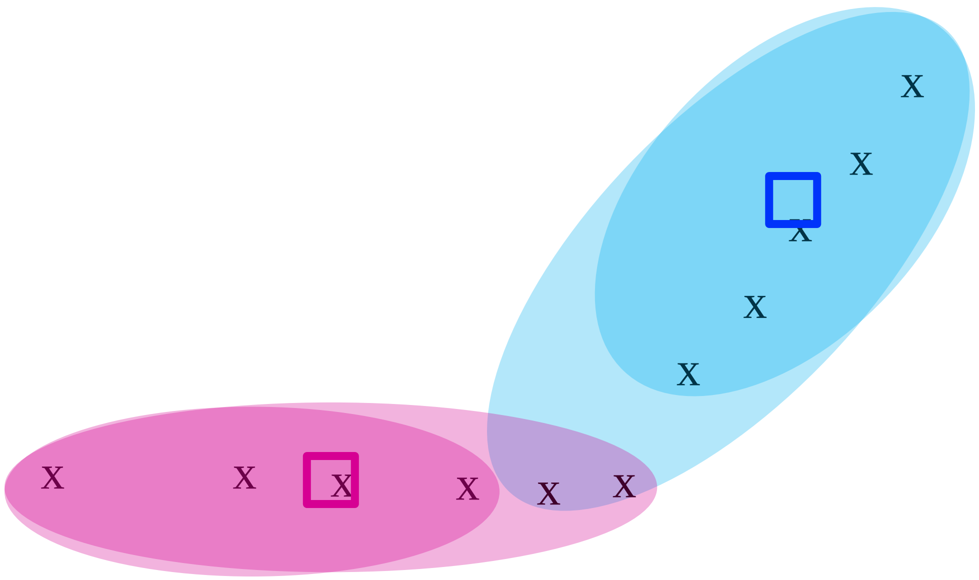
Clusters after Round 2



x … data point
… centroid

Example

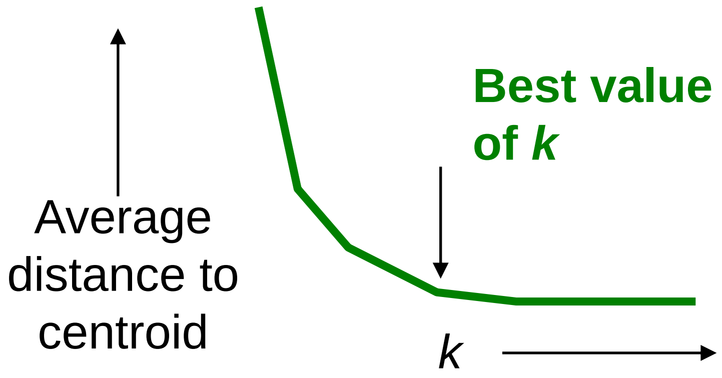
Clusters at the end



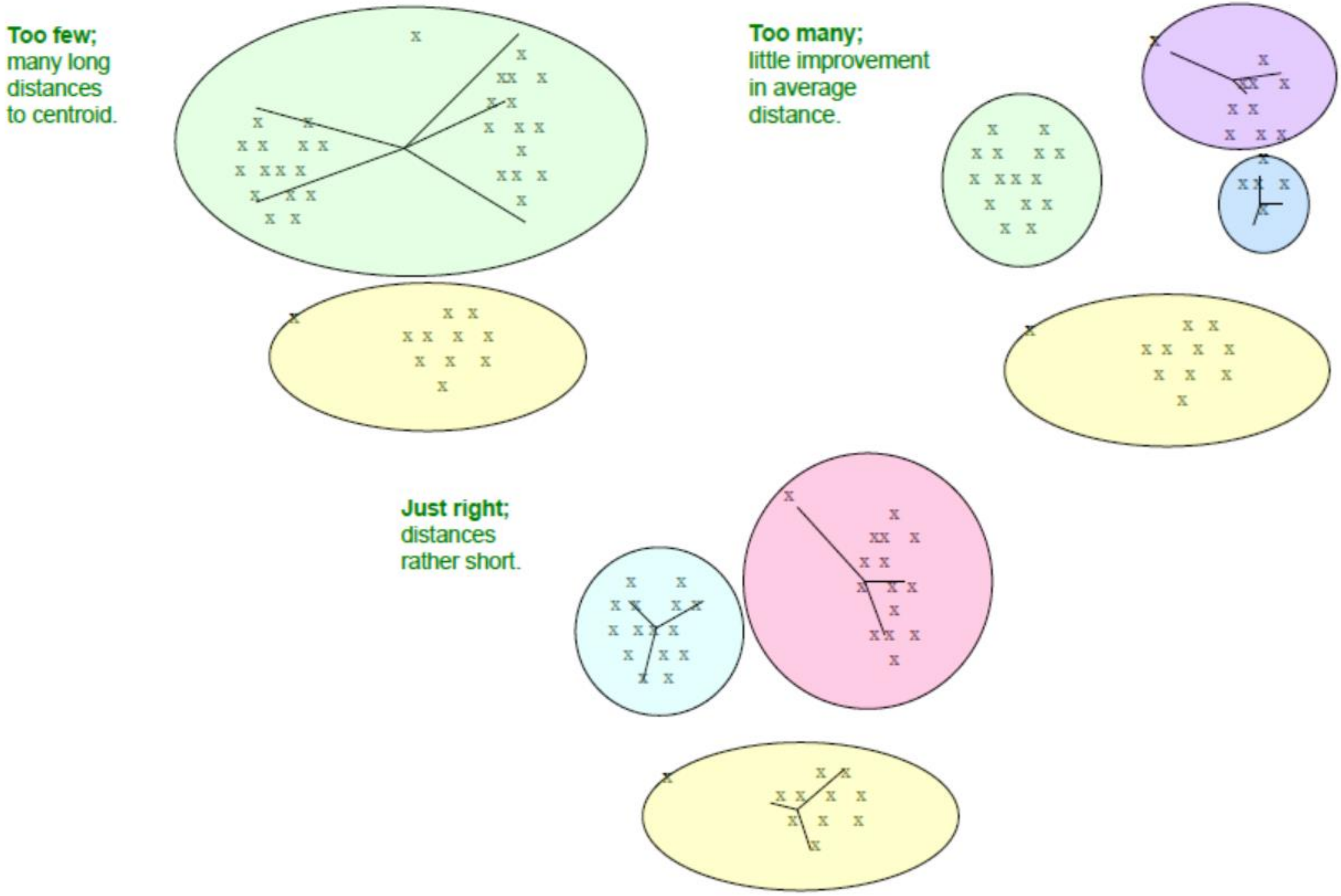
x … data point
□ … centroid

How to Select *k*?

- **Try different values of *k***
- **Look at the change in the average distance to centroid as *k* increases**
- **Average falls rapidly until right *k*, then changes little**



Example


Too few;
many long
distances
to centroid.
Too many;
little improvement
in average
distance.
Just right;
distances
rather short.

Does k-means Converge?

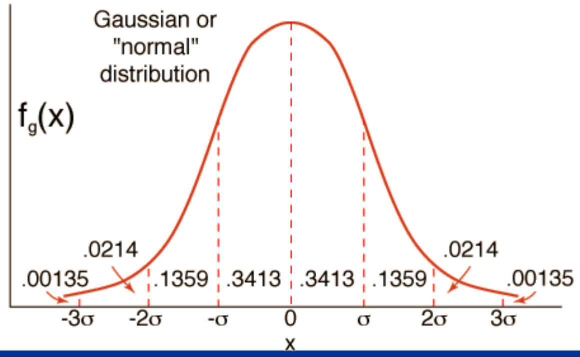
- **Most of the time, convergence happens in the first few iterations**
- **Stopping criterion is often changed to "until relatively few points change clusters"**
- **Complexity is O(n*k*m)**
  - n = number of points
  - k = number of clusters
  - m = number of iterations

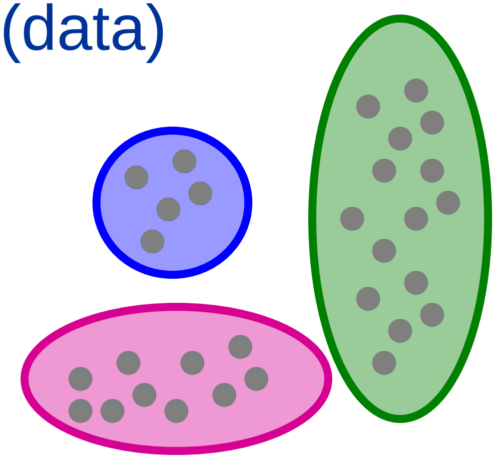
BFR Algorithm
(Bradley-Fayyad-Reina)

Extension of *k*-means to large data

BFR Algorithm

- **Variant of *k*-means designed to handle very large (disk-resident) data sets in high dimensions**
- **Assumes points in clusters are normally distributed around a centroid in a Euclidean space**
  - Standard deviations in different dimensions may vary
  - Clusters are axis-aligned ellipses
- **Efficient way to summarize clusters**
  - Memory required O(clusters) and not O(data)





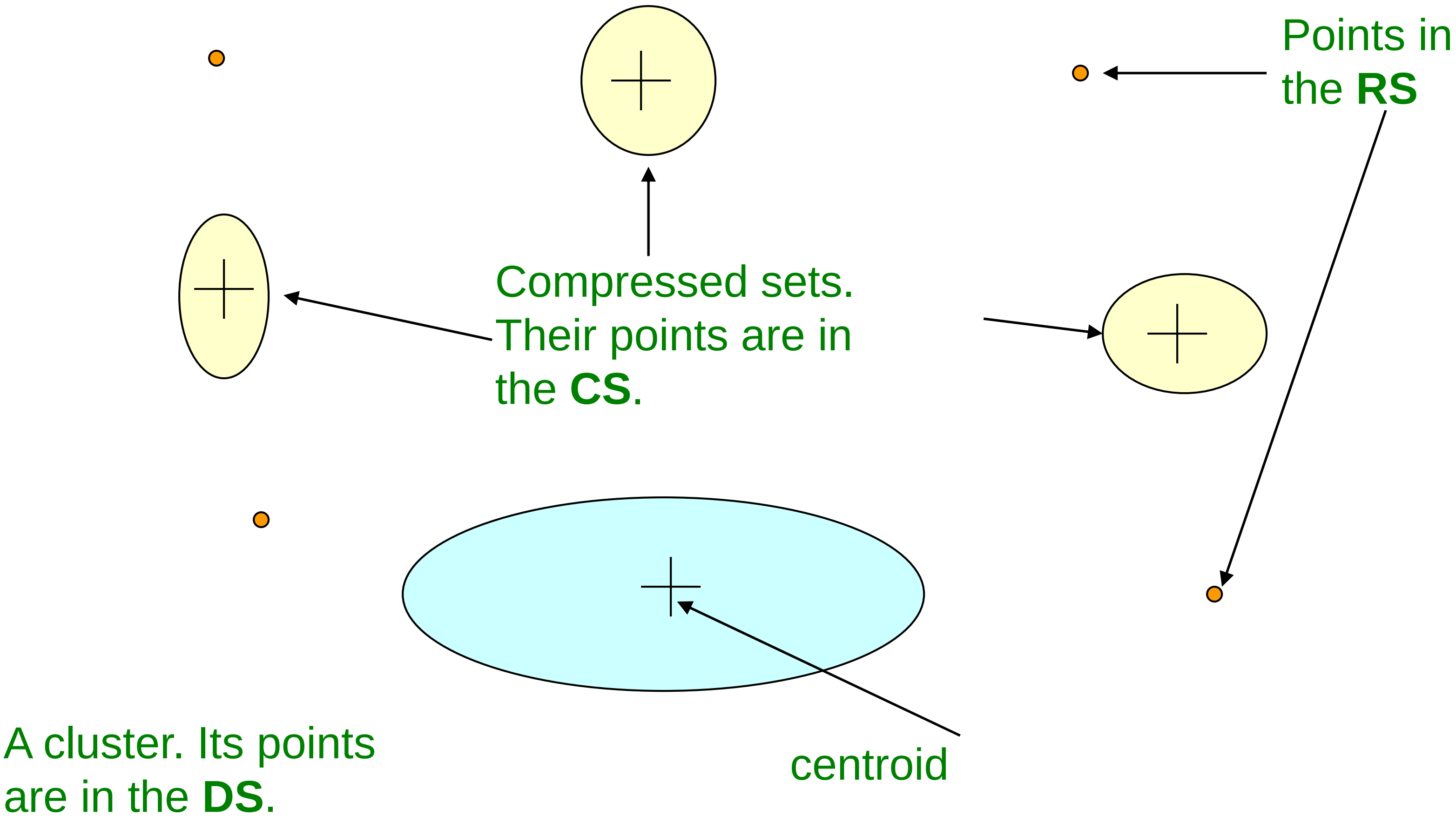
BFR Algorithm

- **Points are read from disk to memory in chunks**
- **Points from previous memory loads are summarized by simple statistics**
- **Select initial *k* centroids from the first chunk**
  - Take ***k*** random points; or
  - Take a small random sample and cluster optimally; or
  - Take a sample; pick a random point, and then ***k–1*** more points, each as far from the previously selected points as possible

BFR Algorithm

- **Keep track of 3 sets of points in memory**
  - **Discard set (DS):**
    - Points close enough to a centroid to be summarized
  - **Compression set (CS):**
    - Groups of points that are close together but not close to any existing centroid
    - These points are summarized, but not assigned to a cluster
  - **Retained set (RS):**
    - Isolated points to be assigned to a compression set

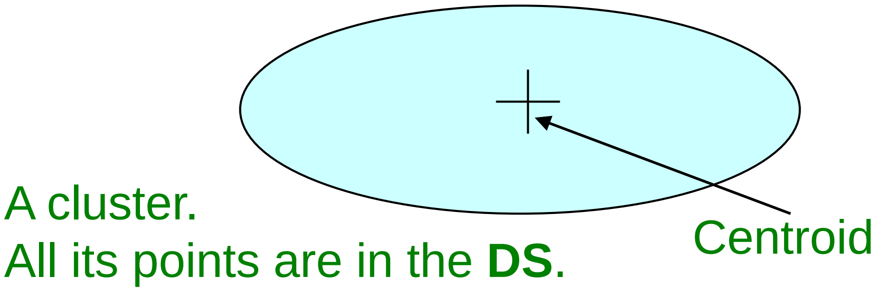
BFR: “Galaxies” Picture



Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

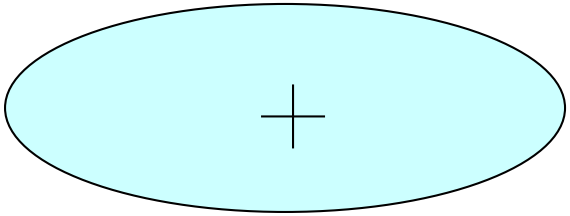
Summarizing Points

- **For each cluster, discard set (DS) is summarized by**
  - Number of points, ***N***
  - Vector ***SUM*** whose i^{th} component is the sum of the coordinates of the points in the i^{th} dimension
  - Vector ***SUMSQ*** where i^{th} component = sum of squares of coordinates in i^{th} dimension



Summarizing Points

- **2*d* + 1 values represent any size cluster**
  - ***d*** = number of dimensions
- **Centroid (Average in each dimension) is given by $\mathbf{SUM}_i\,/\,N$**
  - $\mathbf{SUM}_i$ = i^{th} component of SUM
- **Variance of a cluster's DS in dimension *i* is given by $(\mathbf{SUMSQ}_i\,/\,N) - (\mathbf{SUM}_i\,/\,N)^2$**
  - Standard deviation is the square root of variance
- **Next step: Actual clustering**



Processing "Memory-Load" of Points

- **Find those points that are "sufficiently close" to a cluster centroid and add those points to that cluster and the DS**
  - These points are so close to the centroid that they can be summarized and then discarded
- **Use any main-memory clustering algorithm to cluster the remaining points and the old RS**
  - Clusters go to the **CS**; outlying points to the **RS**

Discard set (DS): Close enough to a centroid to be summarized.
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

Processing “Memory-Load” of Points

- **DS set: Adjust statistics of the clusters to account for the new points**
  - Add ***N***s, ***SUM***s, ***SUMSQ***s
- **Consider merging compressed sets in the CS**
  - Add corresponding values of ***N***s, ***SUM***s, ***SUMSQ***s
- **Last round, merge all compressed sets in the CS and all RS points into their nearest cluster**

Discard set (DS): Close enough to a centroid to be summarized.
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

BFR Algorithm

Two Questions:

1. **How do we decide if a point is "close enough" to a cluster that we will add the point to that cluster?**
2. **How do we decide whether two compressed sets (CS) deserve to be combined into one?**

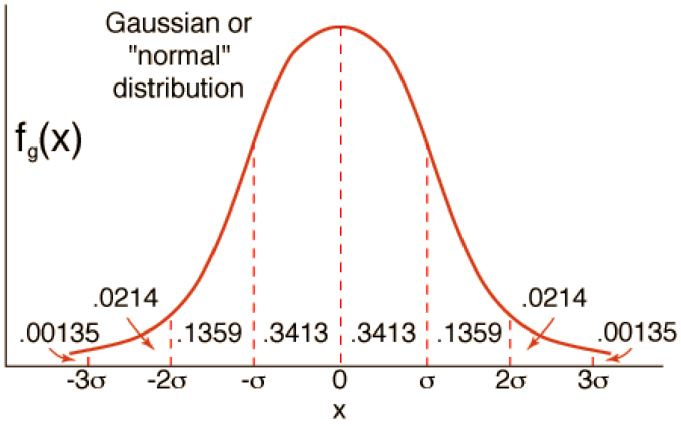
Mahalanobis Distance

- **Measure of the distance between a point P and a distribution D**
  - How many standard deviations is P away from mean of D?
- **For a point *(x_1, …, x_n)* and centroid *(c_1, …, c_n)***
  - $d(x,c) = \sqrt{\sum_{i=1}^{n}\left(\frac{x_i - c_i}{\sigma_i}\right)^2}$

    Normalized Euclidean Distance from Centroid
  - Normalize in each dimension: $y_i = (x_i - c_i) / \sigma_i$
  - Take sum of the squares of the y_i
  - Take the square root
  - σ_i is the standard deviation of points in the cluster in the i^{th} dimension

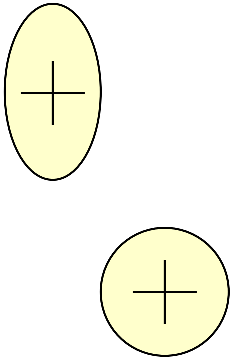
Mahalanobis Distance

- **Compute MD between point and each cluster centroid**
- **Choose cluster whose centroid has the least MD**
- **Add point to cluster if MD < threshold**
  - e.g. threshold = **2** (standard deviation)



Combining Two CS clusters

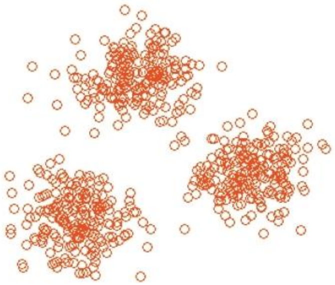
- **Compute the variance of combined cluster**
- ***N*, *SUM*, and *SUMSQ* allow us to make that calculation quickly**
- **Combine if the combined variance is below some threshold**



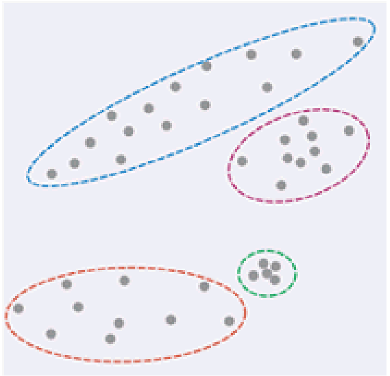
CURE Algorithm

Extension of *k*-means to clusters of arbitrary shapes

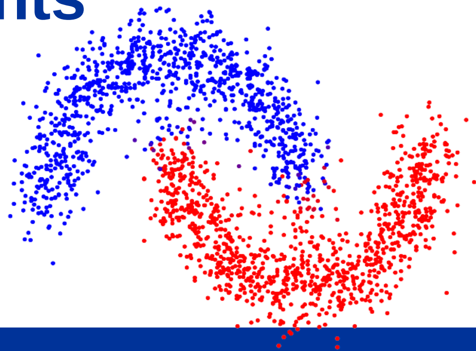
CURE Algorithm



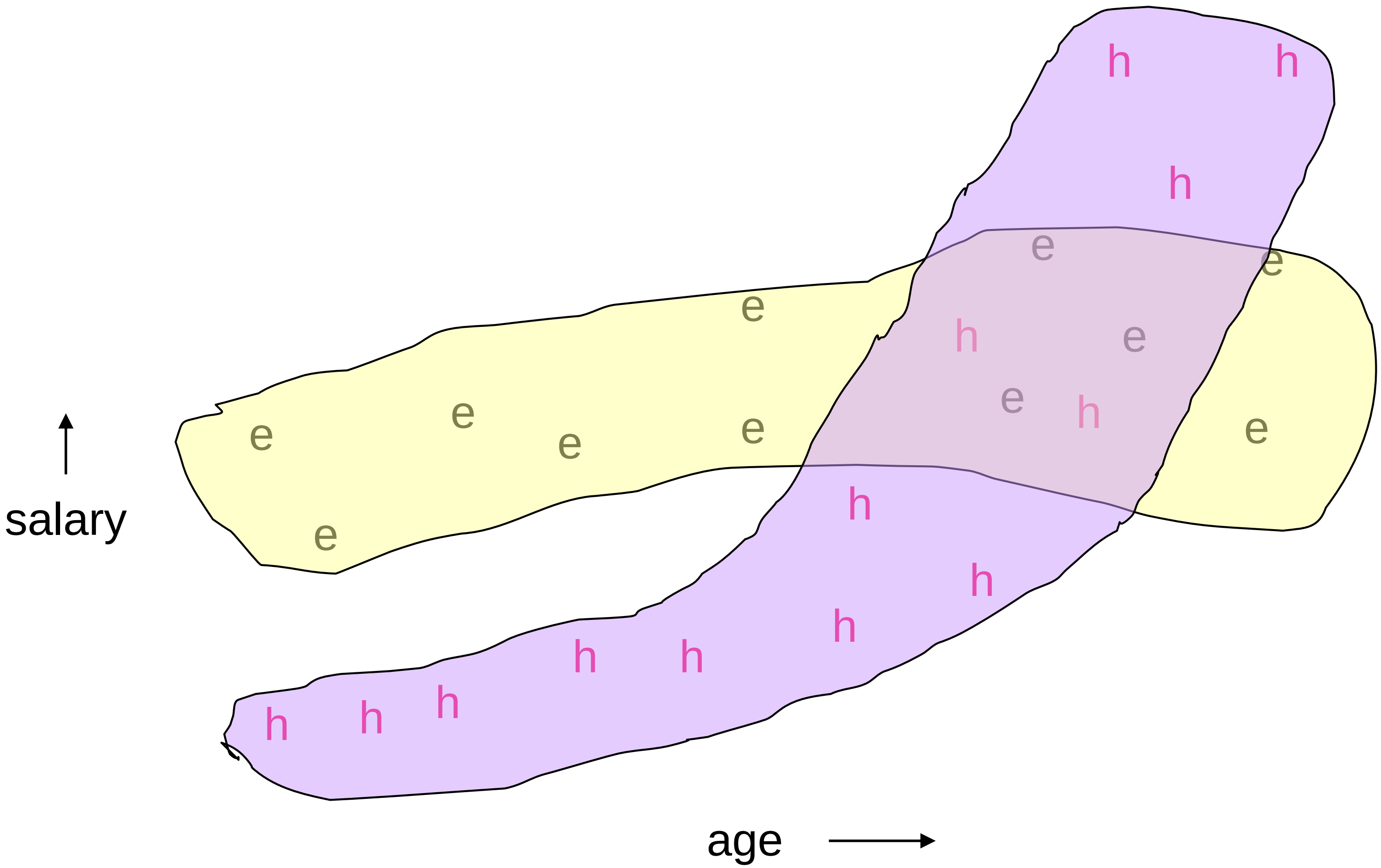
- **BFR and *k*-means assume clusters are normally distributed in each dimension**
  - Axes are fixed
  - Ellipses at an angle are ***not OK***



- **CURE (Clustering Using Representatives)**
  - Assumes Euclidean distance
  - Allows clusters of any shape
  - **Uses a collection of representative points to represent clusters**



Example Stanford Salaries

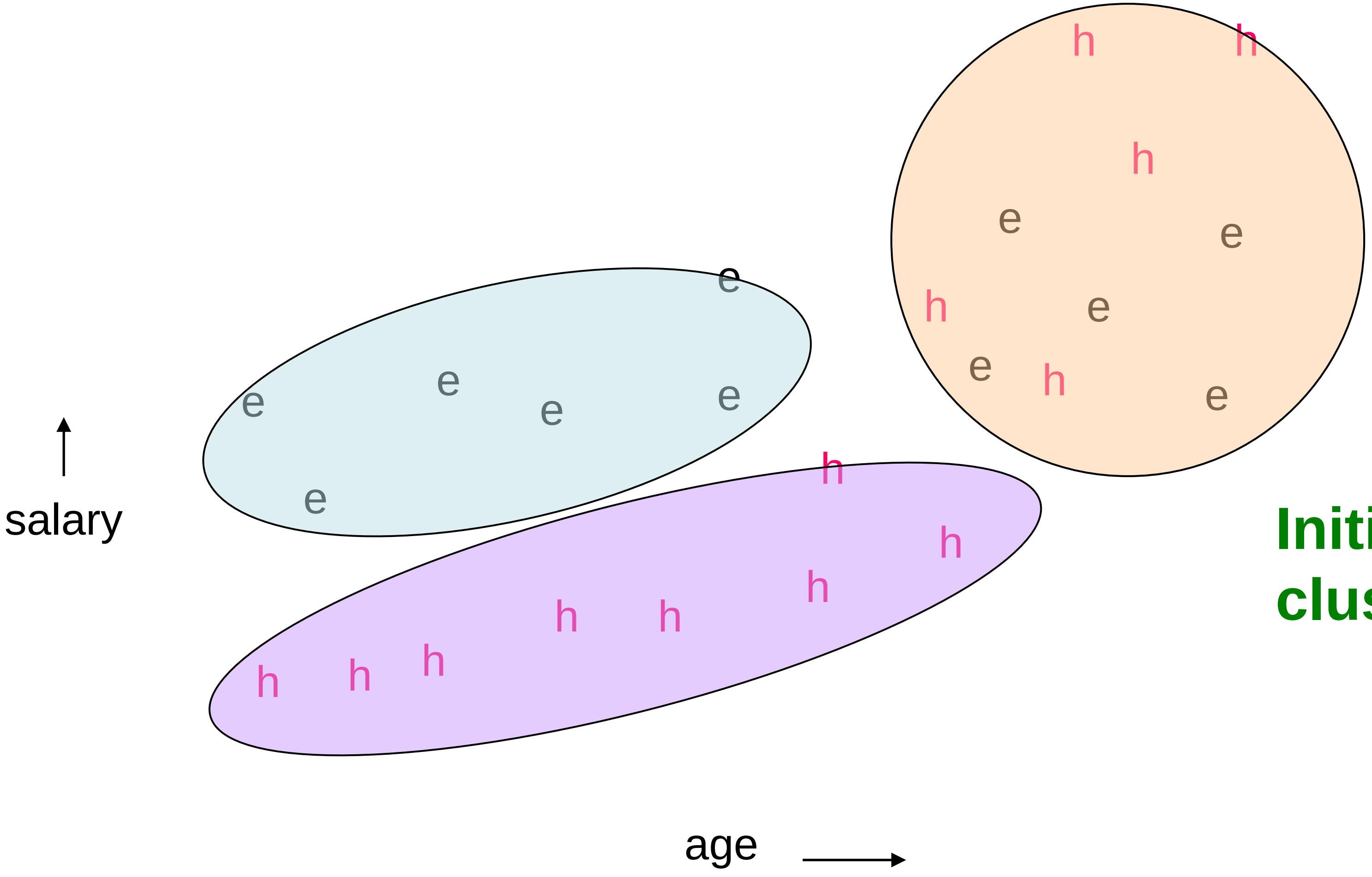


CURE Algorithm

Pass 1

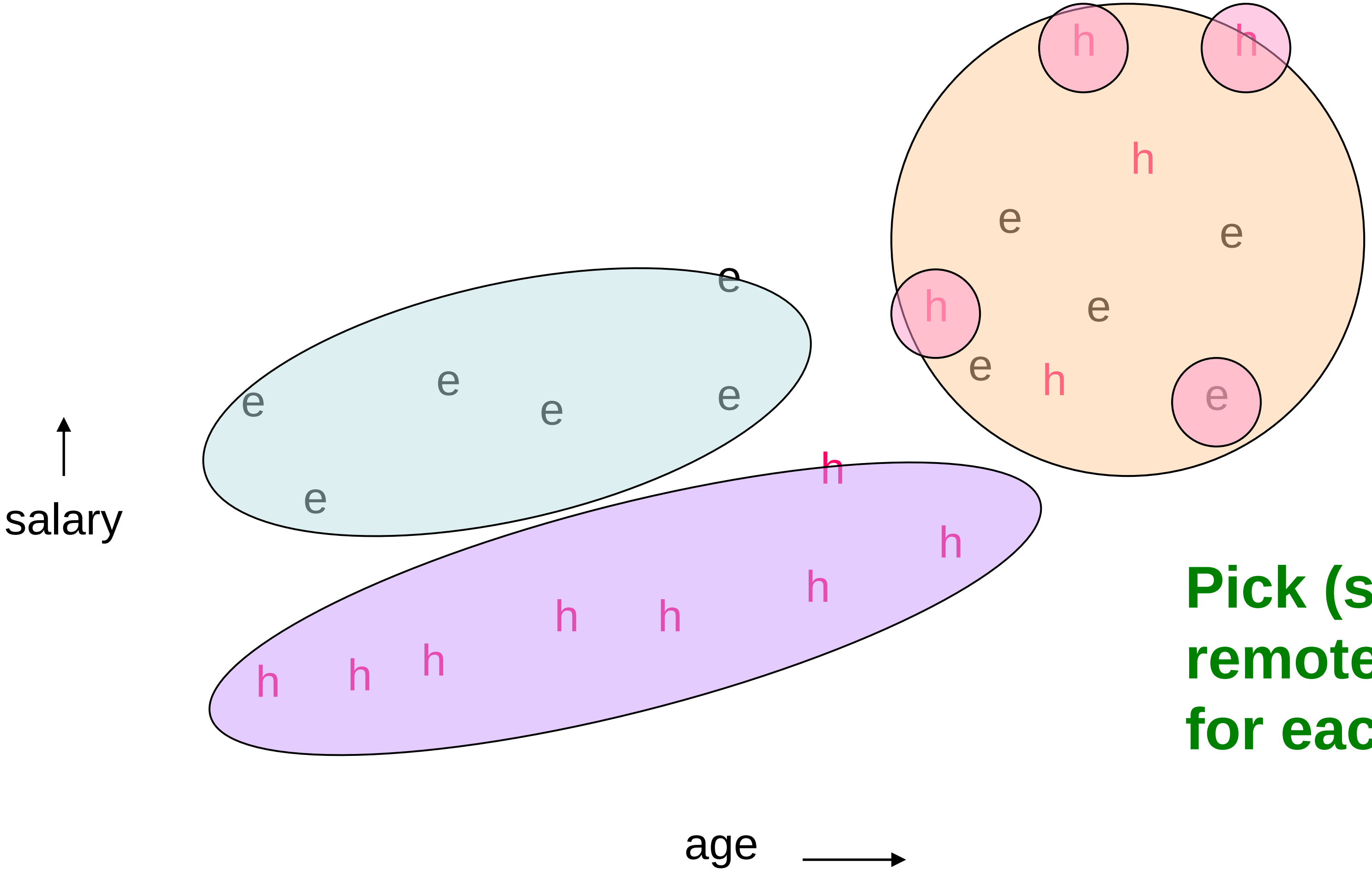
- **Pick a random sample of points that fit in main memory**
- **Cluster these points hierarchically – group nearest points/clusters**
- **Pick representative points**
  - For each cluster, pick a sample of points, as dispersed as possible
  - From the sample, pick representatives by moving them (say) 20% toward the centroid of the cluster

Example



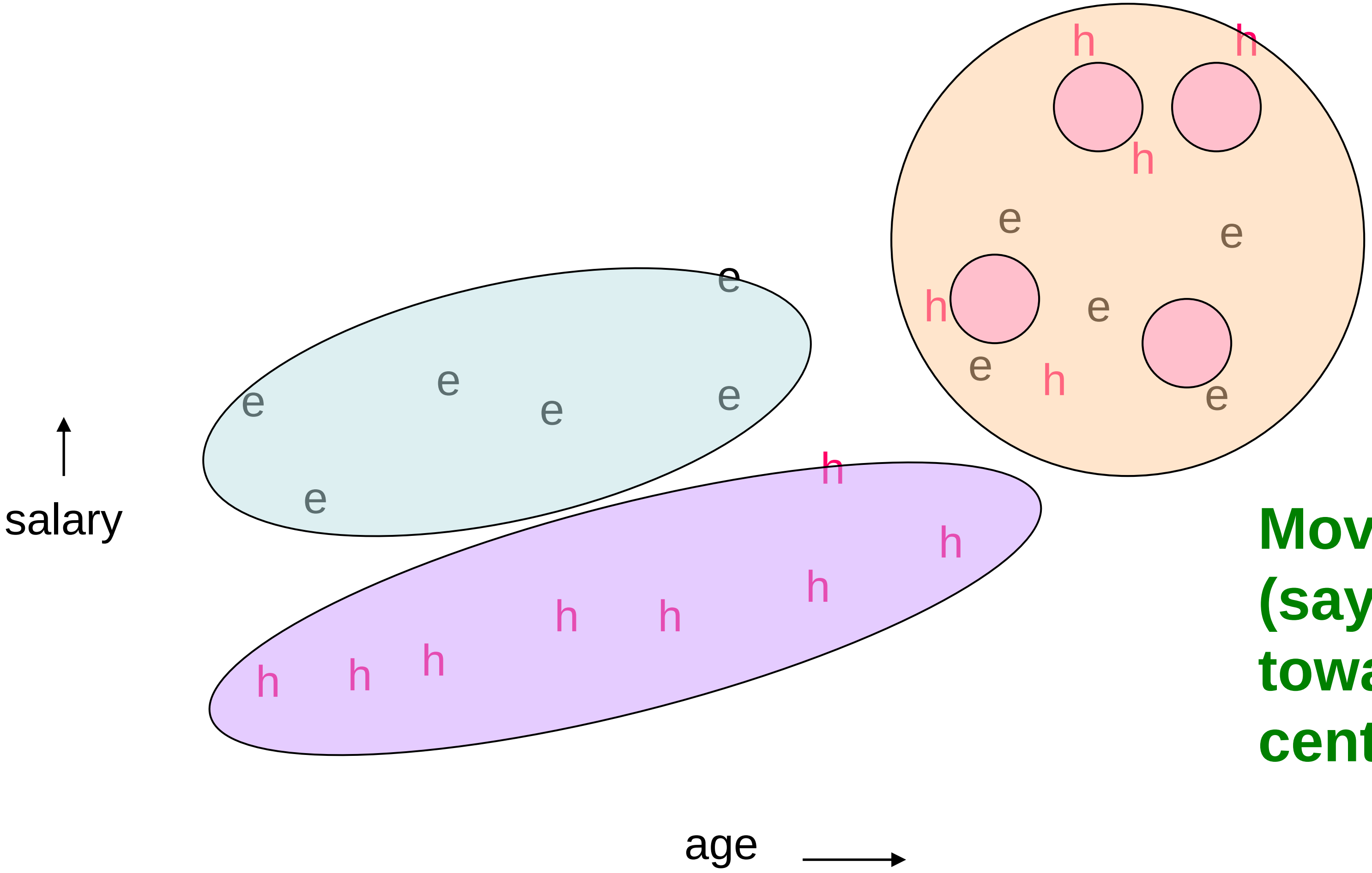
Initials clusters

Example



Pick (say) 4 remote points for each cluster

Example

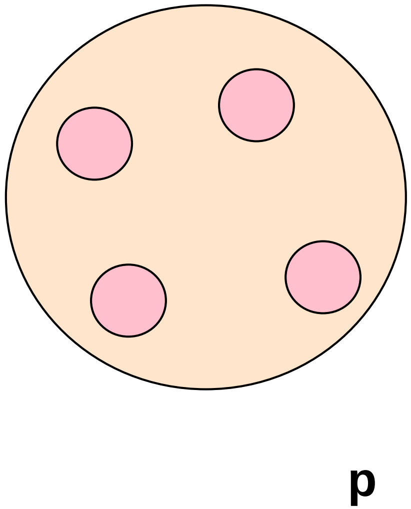


Move points (say) 20% toward the centroid.

CURE Algorithm

Pass 2

- **Rescan the whole dataset and visit each point *p* in the data set**
- **Place it in the "closest cluster"**
  - Find the closest representative to ***p*** and assign it to representative's cluster



Summary

- Given a set of points, with a notion of distance between points, group the points into some number of *clusters*
- Centroid in Euclidean space and clustroid in non-Euclidean space
- Agglomerative hierarchical clustering
- *k*-means, BFR (*k*-means extended for large data sets), CURE (*k*-means extended for arbitrary clusters**)**

- Self-Study: Clustering in non-Euclidean space
  - GRGPF