# Clustering

CS246: Mining Massive Datasets

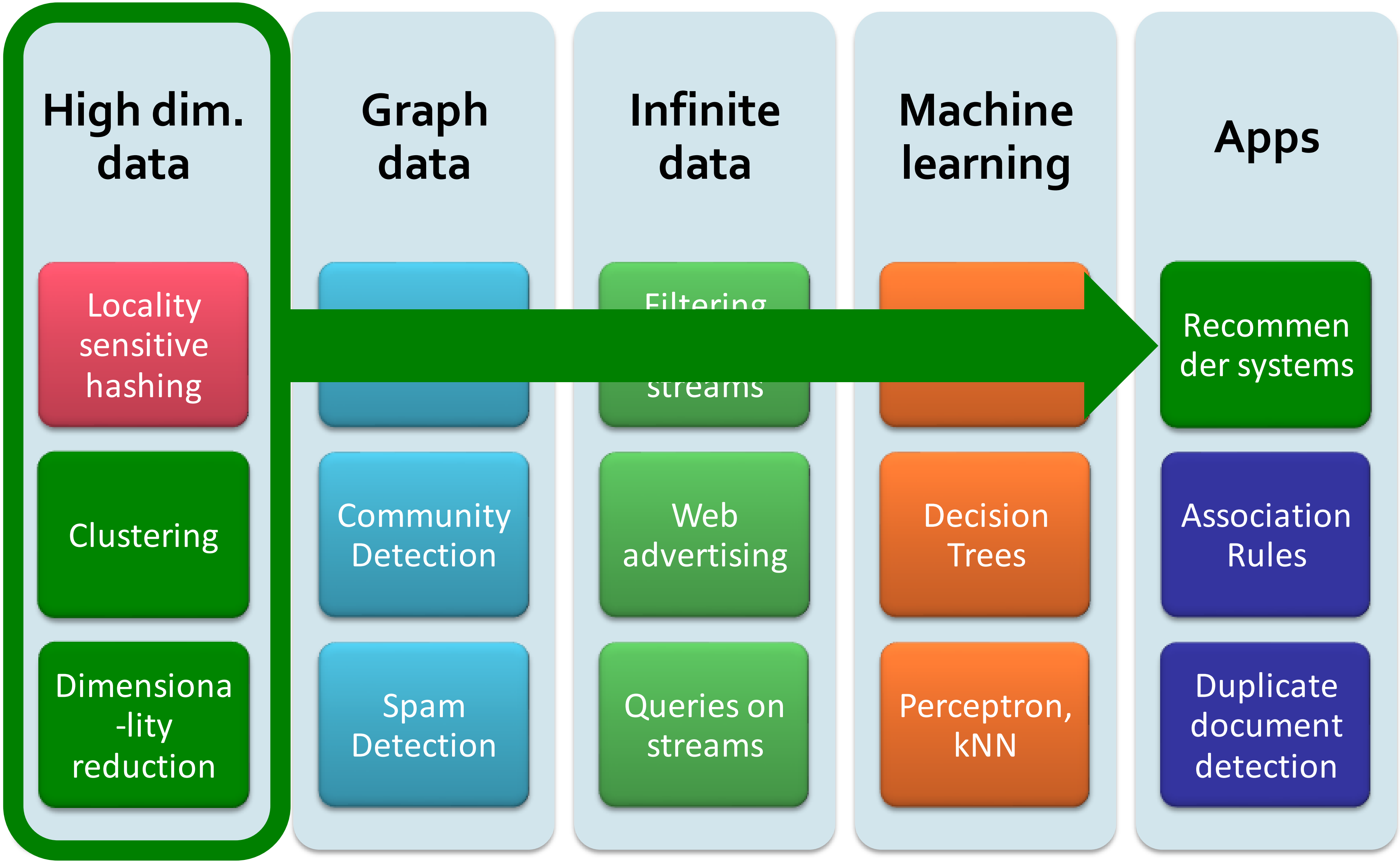
Jure Leskovec, Stanford University
Mina Ghashami, Stanford University

http://cs246.stanford.edu

# Announcements

- Colab 0 & 1 grades released.
  - Solutions to be released soon.
- Colab 2 and Homework 1 due this Thursday.

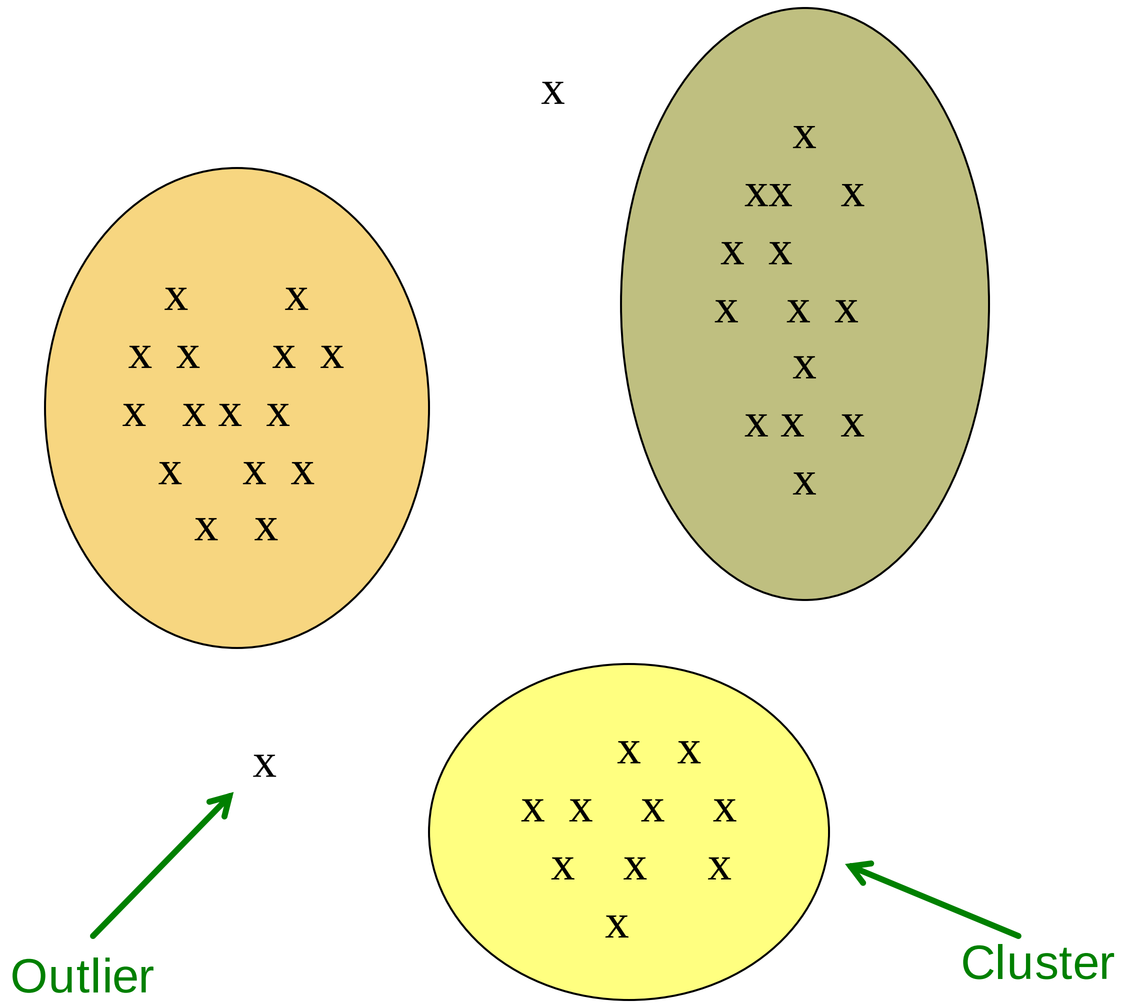
# High Dimensional Data



# The Problem of Clustering

- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, so that
  - Members of the same cluster are close/similar to each other
  - Members of different clusters are dissimilar
- **Usually:**
  - Points are in a high-dimensional space
  - Similarity is defined using a distance measure
    - Euclidean, Cosine, Jaccard, edit distance, …

# Example: Clusters & Outliers



# Clustering Problem: Galaxies

- A catalog of 2 billion "sky objects" represents objects by their radiation in 7 dimensions (frequency bands)
- **Problem:** Cluster similar objects, e.g., galaxies, nearby stars, quasars, etc.
- Sloan Digital Sky Survey

# Clustering Problem: Music CDs

- **Intuitively: Music can be divided into categories, and customers prefer a few genres**
  - But what are categories really?
- Represent a CD by a set of customers who bought it
- Similar CDs have similar sets of customers, and vice-versa

# Clustering Problem: Music CDs

**Space of all CDs:**

- Think of a space with one dim. for each customer
  - Values in a dimension may be 0 or 1 only
  - A CD is a "point" in this space $(x_1, x_2,\ldots, x_d)$, where $x_i = 1$ iff the $i^{th}$ customer bought the CD
- For Amazon, the dimension is tens of millions
- **Task:** Find clusters of similar CDs

# Clustering Problem: Documents

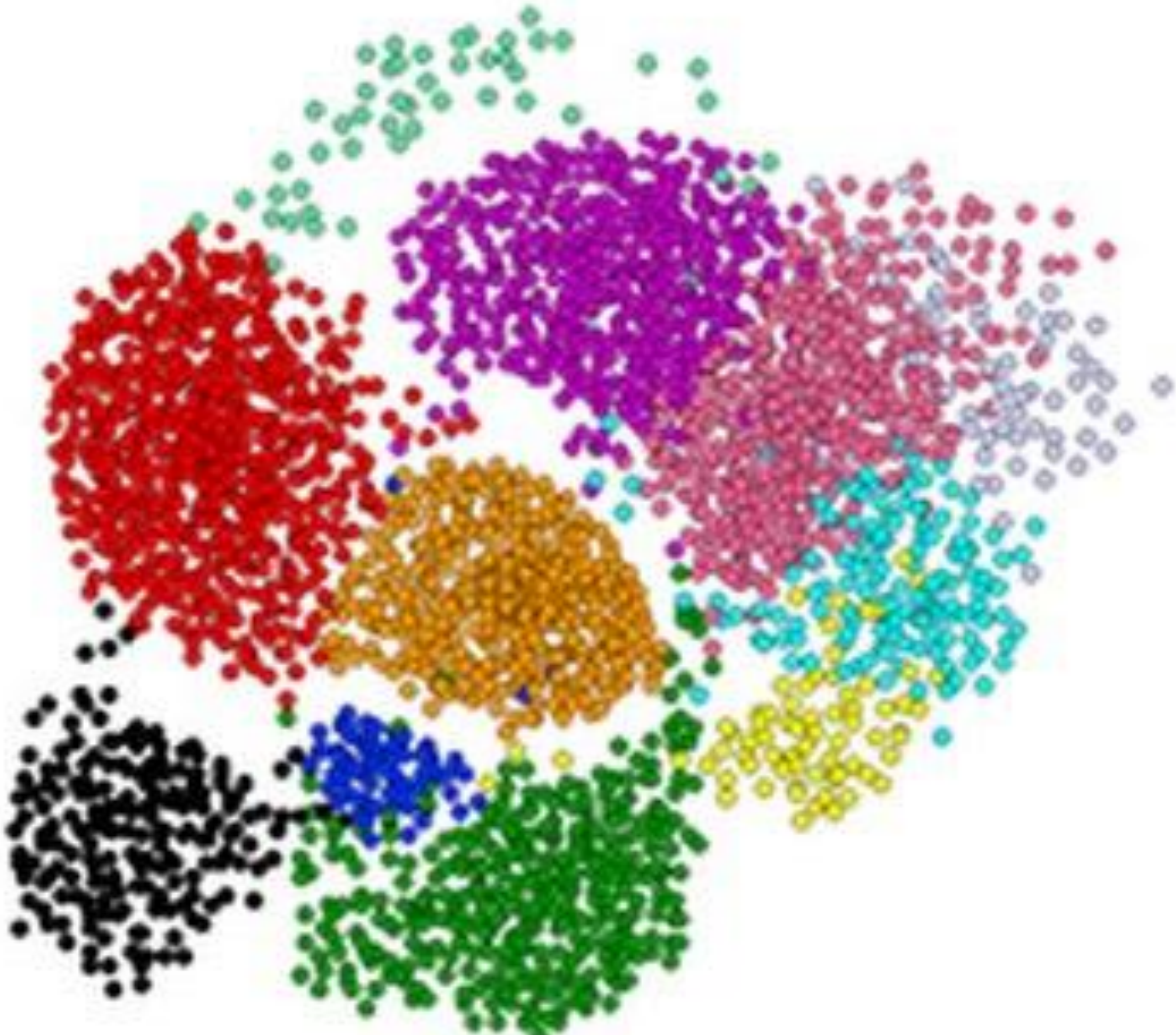
**Finding topics:**

- Represent a document by a vector $(x_1, x_2,\dots, x_k)$, where $x_i = 1$ iff the $i$ th word (in some order) appears in the document
  - It actually doesn't matter if $k$ is infinite; i.e., we don't limit the set of words
- **Documents with similar sets of words may be about the same topic**

# Cosine, Jaccard, and Euclidean

- **We have a choice when we think of documents as sets of words or shingles:**
  - **Sets as vectors:** Measure similarity by the **cosine distance**
  - **Sets as sets:** Measure similarity by the **Jaccard distance**
  - **Sets as points:** Measure similarity by **Euclidean distance**

# Clustering is a hard problem!



# Why is it hard?

- Clustering in two dimensions looks easy
- Clustering small amounts of data looks easy
- And in most cases, looks are not deceiving

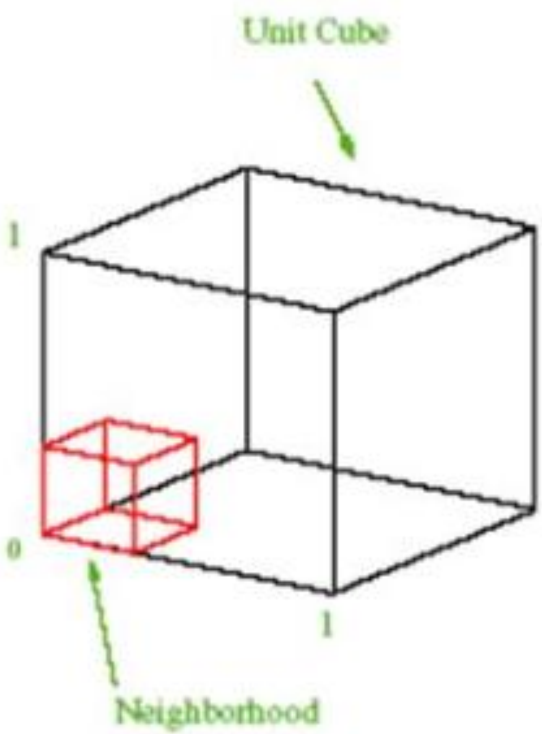
- Many applications involve not 2, but 10 or 10,000 dimensions
- **High-dimensional spaces look different:** Almost all pairs of points are very far from each other --> **The Curse of Dimensionality!**

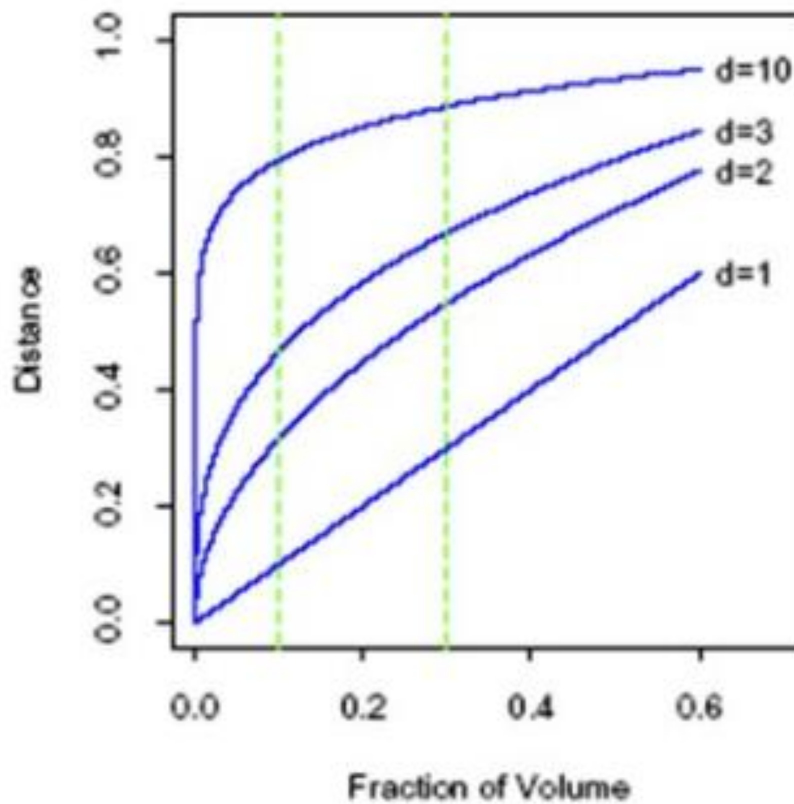
# Example: Curse of Dimensionality

- **Take 10,000 uniform random points on [0,1] line. Assume query point is at the origin**
- **What fraction of "space" do we need to cover to get 0.1% of data** (10 nearest neighbors)
- In 1-dim to get 10 neighbors we must go to distance 10/10,000=0.001 on the average
- In 2-dim we must go $\sqrt{0.001}$=0.032 to get a square that contains 0.001 volume
- In general, in d-dim we must go $(0.001)^{\frac{1}{d}}$
- So, in 10-dim to capture 0.1% of the data we need 50% of the range.

# Example: Curse of Dimensionality

**Curse of Dimensionality:** All points are very far from each other

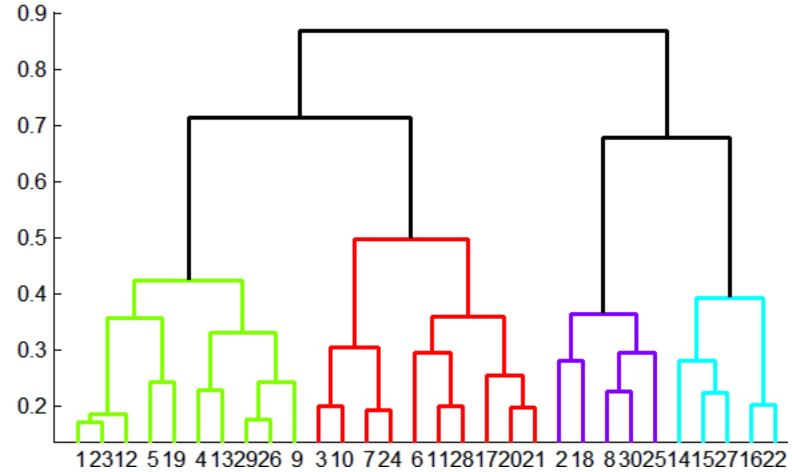


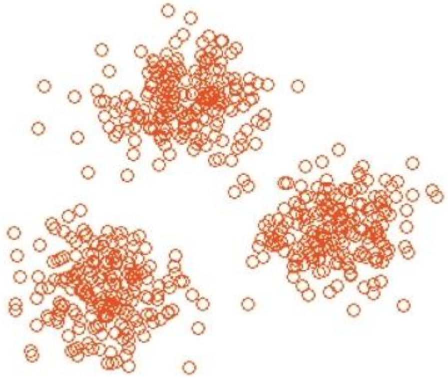
# Overview: Clustering Strategies (1)

Two group of methods:

- **Hierarchical:**
  - **Agglomerative** (bottom up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one
  - **Divisive** (top down):
    - Start with one cluster and recursively split it




- **Point assignment:**
  - Maintain a set of clusters
  - Points belong to the "nearest" cluster



# Overview: Clustering Strategies (2)

- Is the space Euclidean or non-Euclidean?
- In Euclidean:
  - Points are vectors of real numbers, i.e. coordinates
  - It is possible to summarize a collection of points as their average. We call it centroid.
  - Distance measure: L2 norm, L1 norm
- In non-Euclidean:
  - There is no notion of location, and centroid
  - We summarize a collection of points differently
  - Distance measures: Jaccard, Hamming, cosine

# Overview: Clustering Strategies (3)

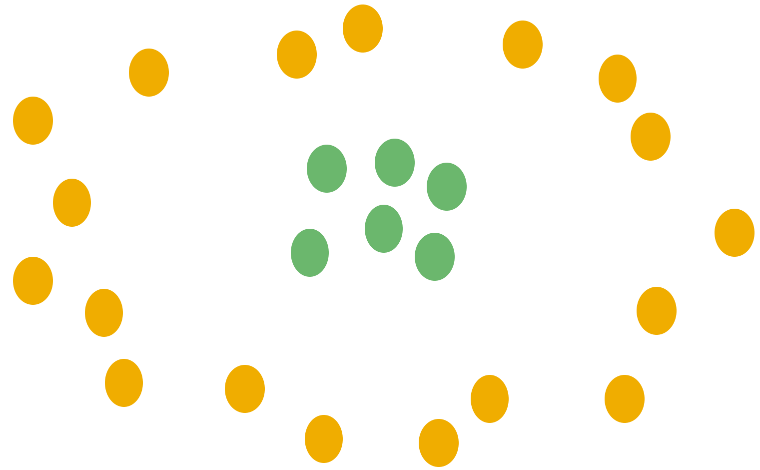
Does the data fit in memory or it resides on disk?

- In-memory clustering is more straightforward
  - Example: K-means
- Large-data clustering requires loading one batch of data at a time, cluster them in memory and keep summaries of clusters
  - Example: BFR, CURE

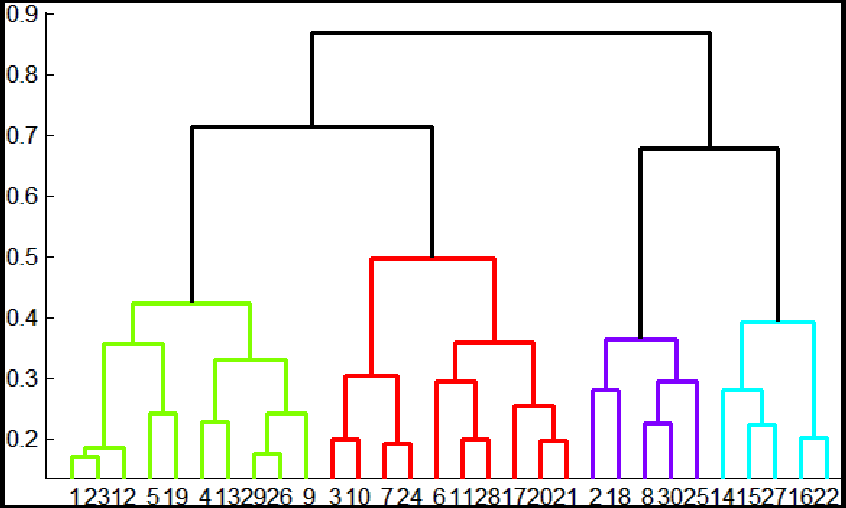
# Hierarchical vs point-assignment

- **Point assignment good when clusters are nice, convex shapes:**

- **Hierarchical can win when shapes are weird:**
  - Note both clusters have essentially the same centroid.



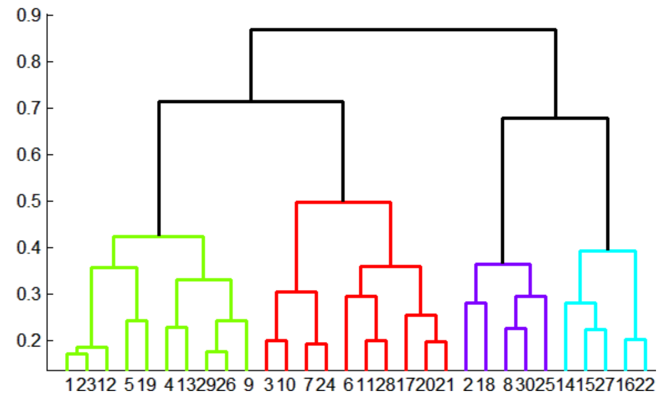
**Aside:** if you realized you had concentric clusters, you could map points based on distance from center, and turn the problem into a simple, one-dimensional case.




# Hierarchical Clustering

# Hierarchical Clustering

- **Key operation:** **Repeatedly merge two "*nearest*" clusters**




- **Three important questions:**
  - **1)** How to represent a cluster?
  - **2)** How to determine the nearness of clusters?
  - **3)** When to stop merging clusters?

# Hierarchical Clustering: Euclidean

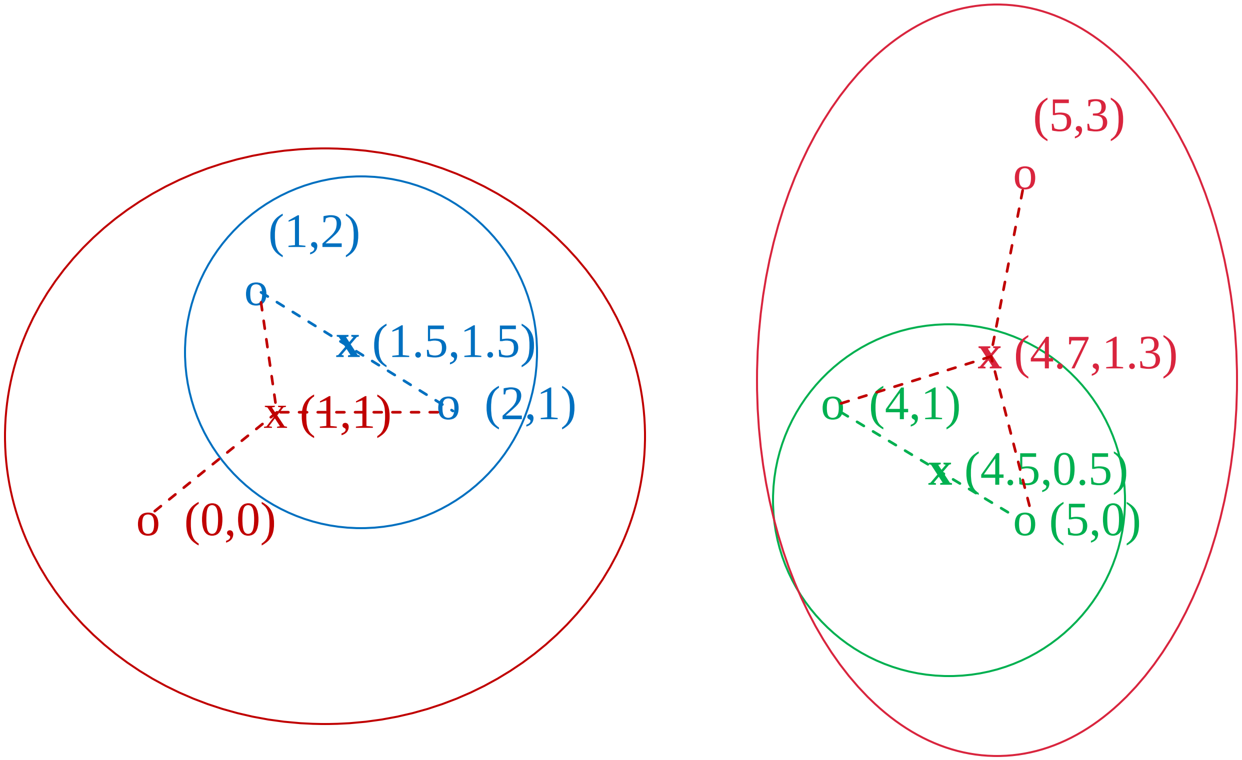
**In Euclidean case:**

- **(1) How to represent a cluster of many points?**
  - As we merge clusters, we represent the "location" of each cluster by its

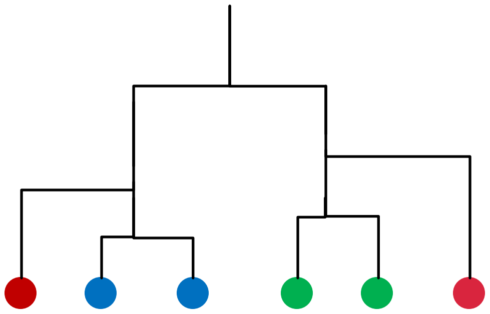
    ***centroid*** = average of its (data)points

- **(2) How to determine the nearness of clusters?**
  - Measure cluster distances by distances of centroids
  - Merge two clusters with the shortest distance

# Example: Hierarchical clustering




**Data:**
o … data point
x … centroid



**Dendrogram**

# Hierarchical Clustering: Non-Euclidean

**In non-Euclidean case:**

- The only "locations" we can talk about are the points themselves. There are three main approaches:

**(1) How to represent a cluster of many points?**

1. pick a ***clustroid*** = point "***closest***" to other points
2. As the collection of points it is.
3. As the collection of points it is.

**(2) How to determine the nearness of clusters?**

1. Treat clustroid as if it were centroid
2. Various distance measures between points of two clusters
3. Various cohesion measures of the union of two clusters

# Hierarchical Clustering: Non-Euclidean

## Approach 1:

### (1) How to represent a cluster:

- pick a ***clustroid*** = (data)point "***closest***" to other points

  **Possible meanings of "closest":**

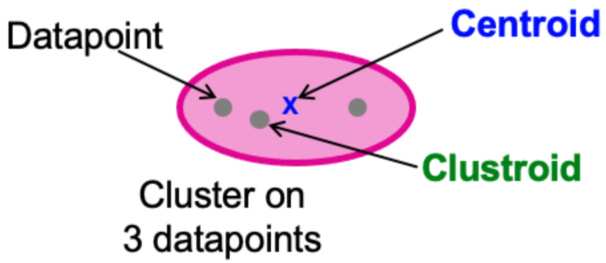
  - Smallest maximum distance to other points
  - Smallest average distance to other points
  - Smallest sum of squares of distances to other points
    - For distance metric ***d*** clustroid ***c*** of cluster ***C*** is $\arg\min_{c} \sum_{x \in C} d(x,c)^2$

### (2) How to determine the nearness of clusters?

- Treat clustroid as if it were centroid

# Centroid vs Clusteroid

- **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.



# Hierarchical Clustering: Non-Euclidean

**Approach 2:**

**(1) How to represent a cluster?** As the collection of points

**(2) How to determine the nearness of clusters?**

Define *inter-cluster distance*:

- Minimum of the distances between any two points, one from each cluster
- Average distance of all pairs of points, one from each cluster

# Hierarchical Clustering: Non-Euclidean

## Approach 3:

(1) **How to represent a cluster?** As the collection of points

**(2) How to determine the nearness of clusters?**
Define a notion of *cohesion,* and merge clusters whose *union* is most cohesive
Possible notions of cohesion (the smaller, the more cohesive):

- **diameter** of the merged cluster = maximum distance between points in the cluster
- **average distance** between points in the cluster
- **Density** of the merged cluster = divide by the number of points in the cluster by diameter or avg. distance

# When to Stop?

**When do we stop merging clusters?**

- When some number $k$ of clusters are found (assumes we know the number of clusters)
- When stopping criterion is met
  - Stop if diameter exceeds threshold
  - Stop if density is below some threshold
  - Stop if merging clusters yields a bad cluster
    - E.g., diameter suddenly jumps
- Keep merging until there is only 1 cluster left

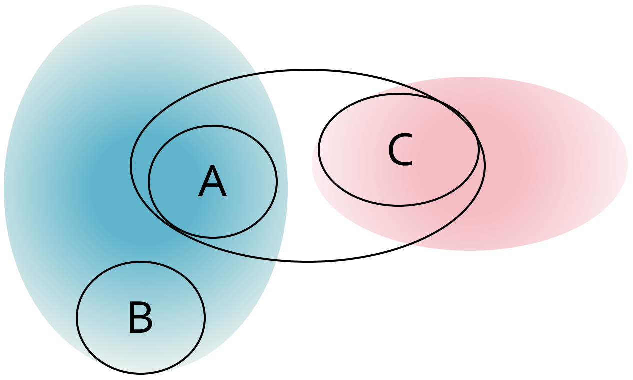
# Which design choice is the best?

- It really depends on the shape of clusters.
  - Which you may not know in advance.

- **Example:** we'll compare two approaches:
  1. Merge clusters with smallest distance between centroids (or clustroids for non-Euclidean)
  2. Merge clusters with the smallest distance between two points, one from each cluster

# Case 1: Convex Clusters

- Centroid-based merging works well.
- But merger based on closest members might accidentally merge incorrectly.



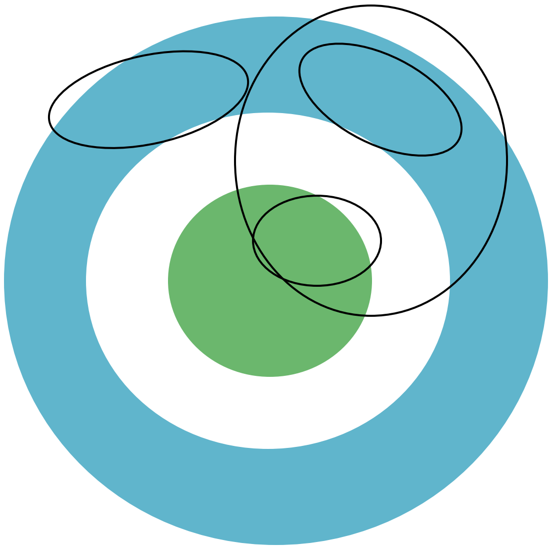

A and B have closer centroids than A and C, but closest points are from A and C.

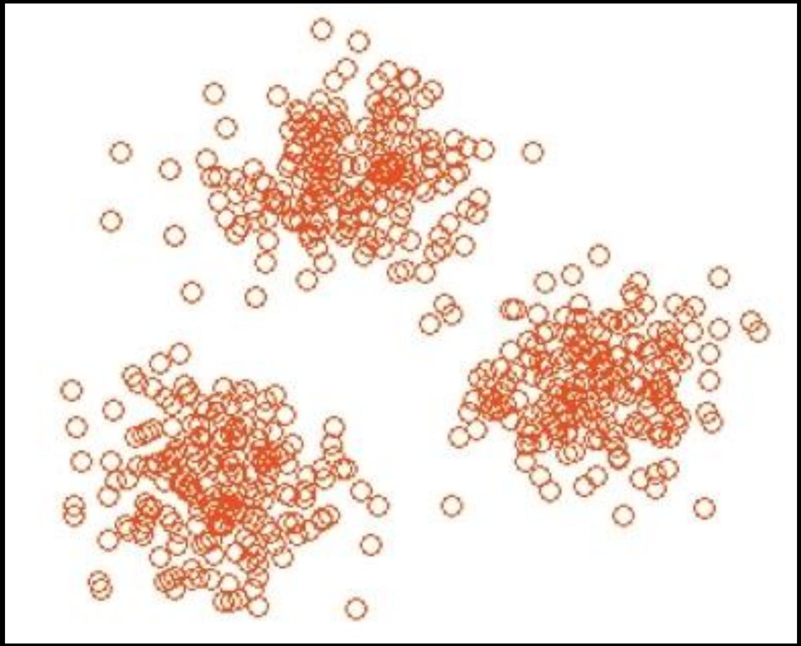


Data density

# Case 2: Concentric Clusters

- Linking based on closest members works well
- But Centroid-based linking might cause errors





# *k*-means Clustering

# *k*–means Algorithm(s)

- It is a problem formulation, not an algorithm.

- **Problem**: Given Euclidean space/distance and ***k* =** number of clusters, find cluster centers that minimizes sum of squared distances from each point to its cluster center

- Finding an exact solution is NP-hard.
- The approximate solution is Lloyd’s algorithm or the k-means algorithm.

# *k*–means Algorithm(s)

- Initialize clusters by picking k centers

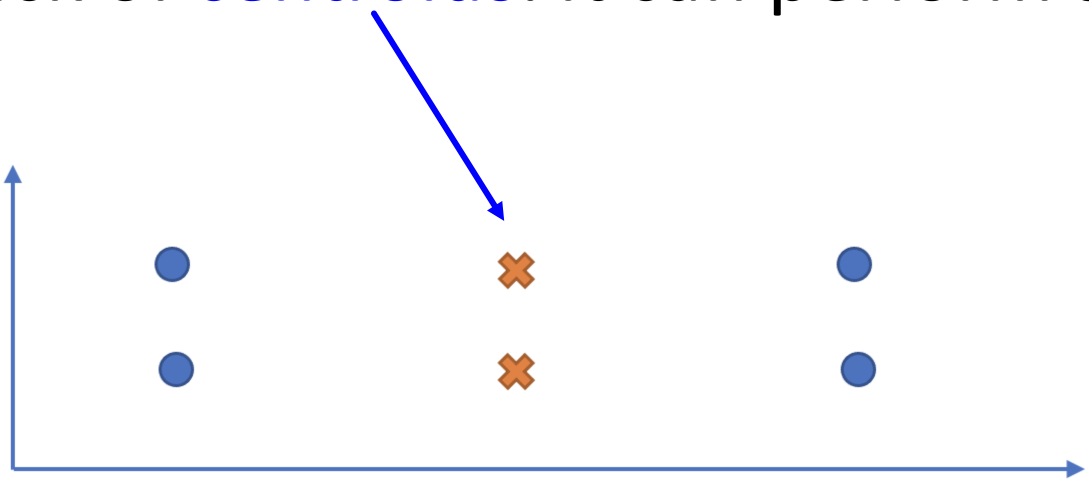
**Until convergence:**

- **1)** For each point, assign it to the cluster whose current centroid is the closest
- **2)** After all points are assigned, update the centroids of the ***k*** clusters as average of datapoints within each cluster

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

# Shortcoming of k-means

Convergence of $k$-means heavily depends on the initial pick of centroids. It can perform arbitrarily badly:



Different strategies for picking $k$ centers:

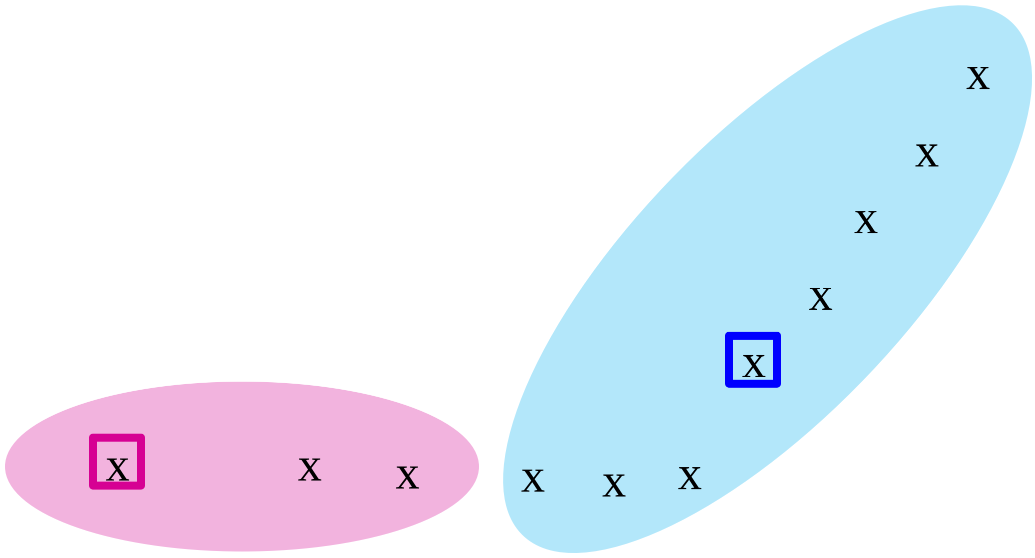
- Pick $k$ datapoints at random
- $k$-means ++

# k-means++

- **Basic idea:** Pick a small sample of points $S$, cluster them by any algorithm, and use the centroids as a seed
- In **k-means++,** sample size $|S|$ = $k$ times a factor that is logarithmic in the total number of points

- **How to pick sample points:** Visit points in random order, but the probability of adding a point $p$ to the sample is proportional to $D(p)^2$.
  - $D(p)$ = distance between $p$ and the nearest already picked point.

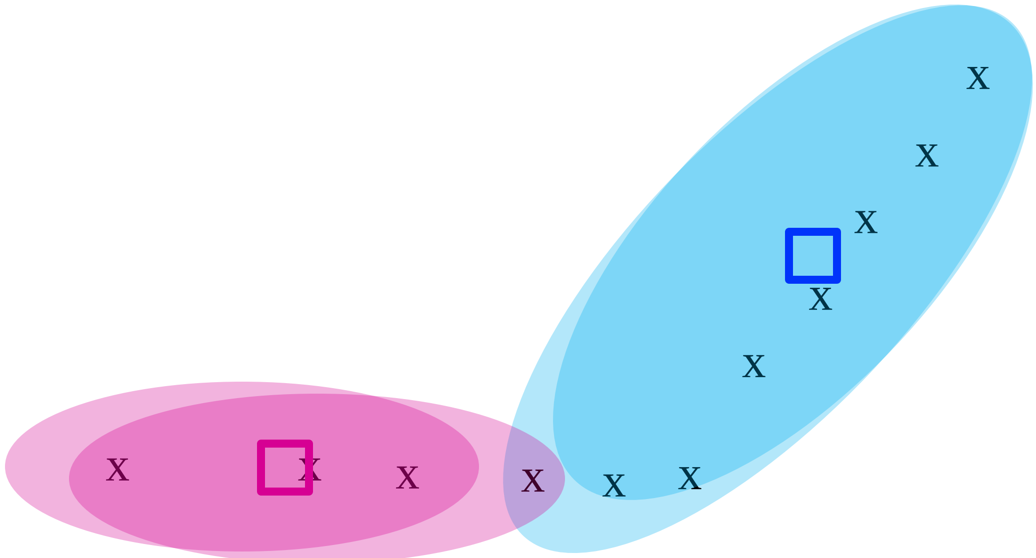
# Example: Assigning Clusters



x … data point
… centroid

**Clusters after round 1**

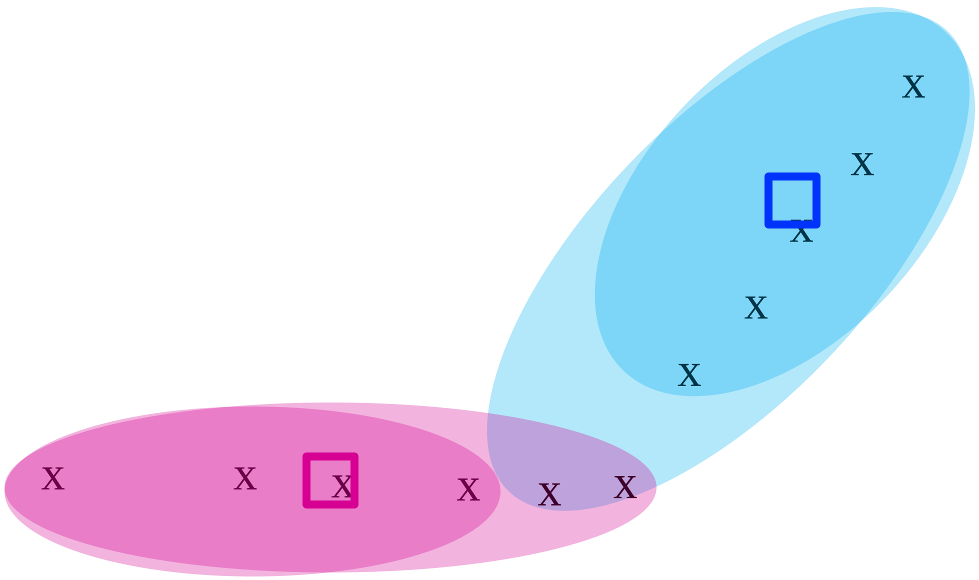
# Example: Assigning Clusters




x … data point
… centroid

**Clusters after round 2**

# Example: Assigning Clusters



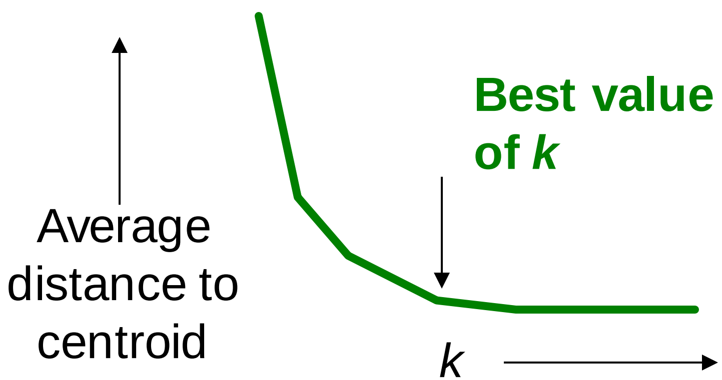

x … data point
… centroid

**Clusters at the end**

# Getting the *k* right

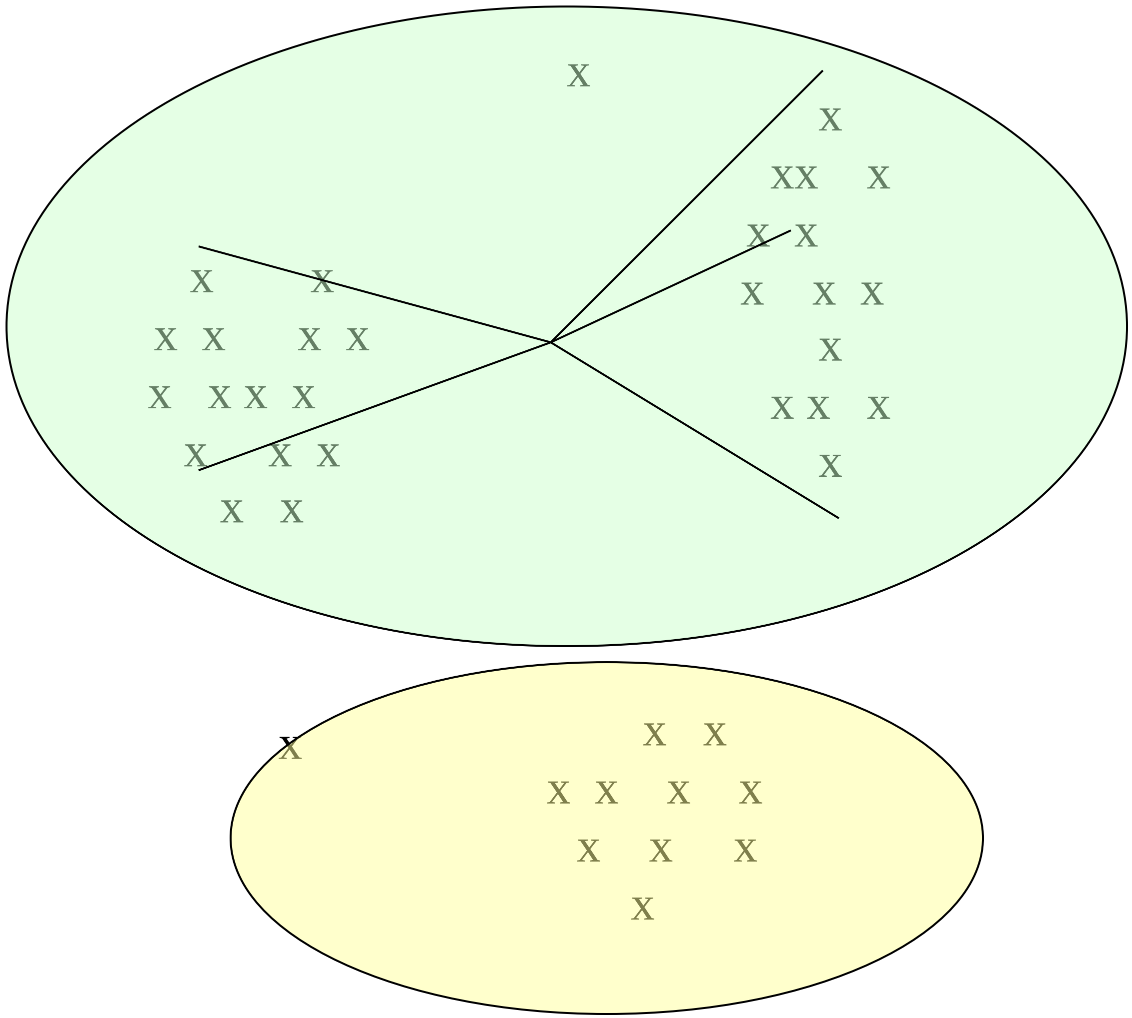
## How to select *k*?

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



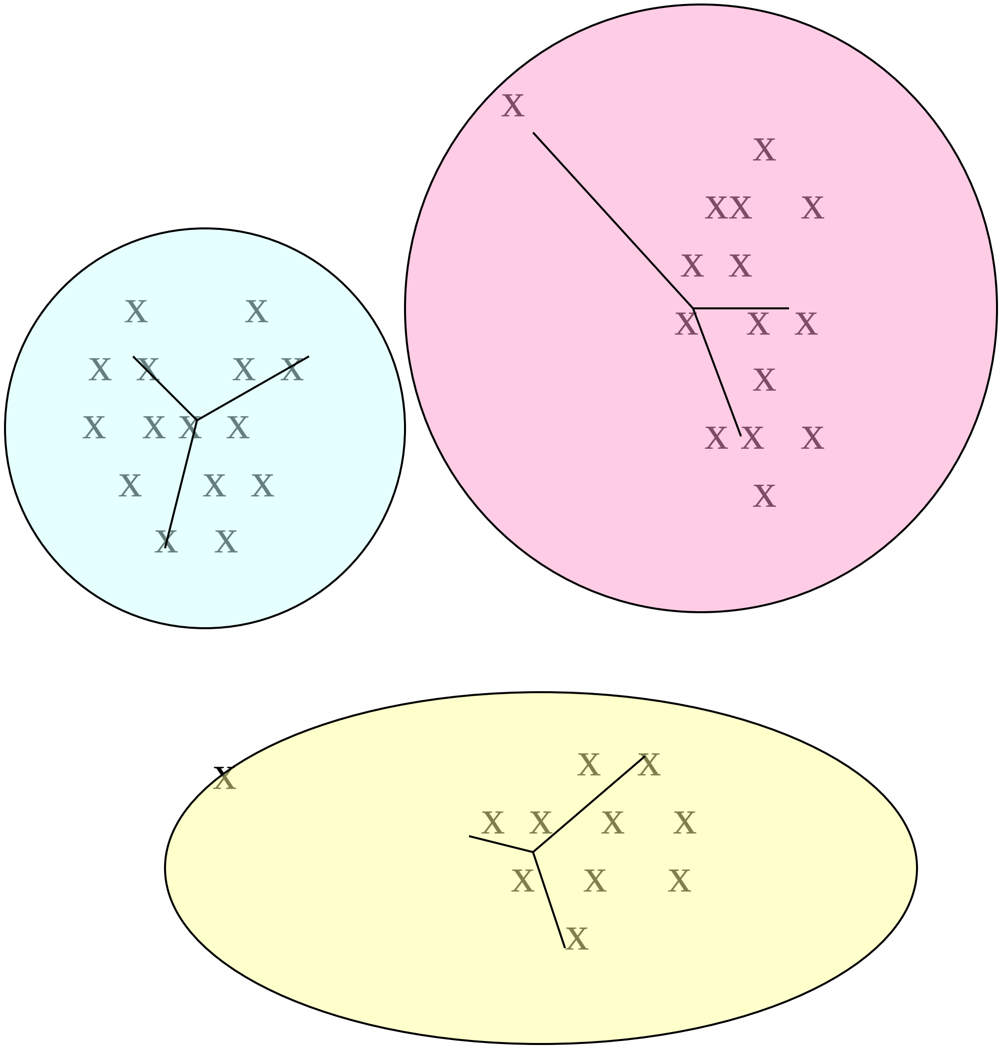
# Example: Picking *k*

**Too few;**
many long distances to centroid



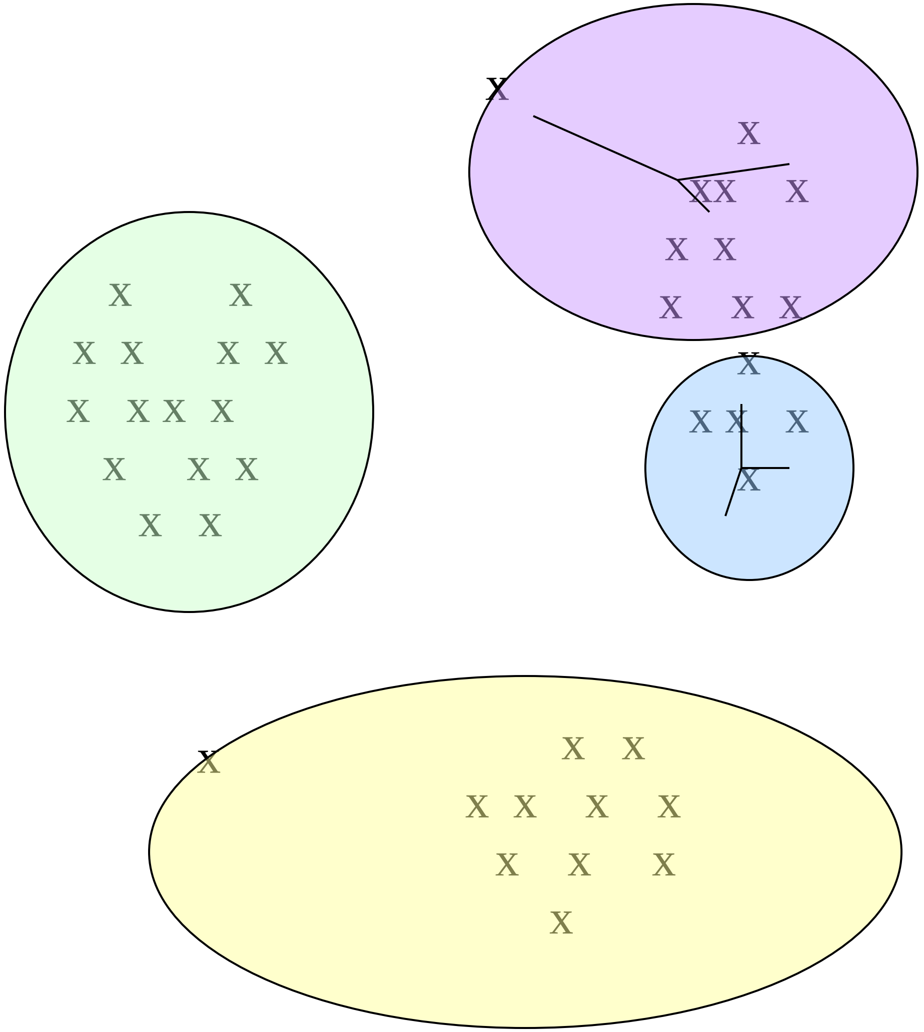
# Example: Picking *k*

**Just right;**
distances
rather short



# Example: Picking *k*

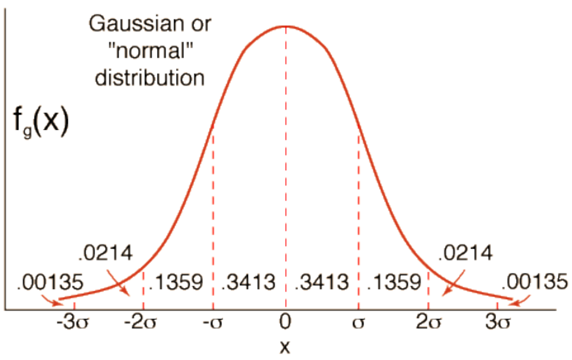
**Too many;**
little improvement in average distance



# The BFR Algorithm

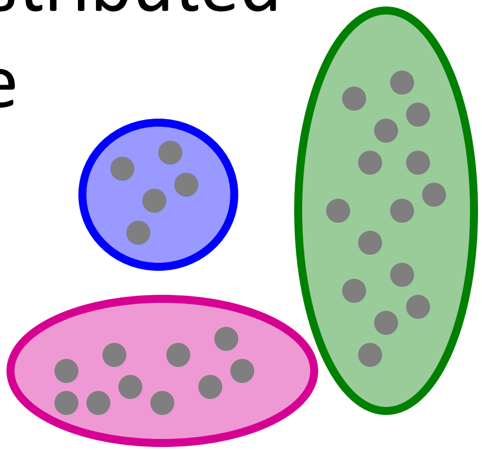
**Extension of *k*-means to large data**

# BFR Algorithm




- **BFR** [Bradley-Fayyad-Reina] is a variant of *k*-means designed to handle **very large** (disk-resident) data sets

- **Assumes** that clusters are normally distributed around a centroid in a Euclidean space
  - Standard deviations in different dimensions may vary
    - Clusters are axis-aligned ellipses



- Goal is to find cluster centroids; point assignment can be done in a second pass through the data.

# BFR Overview

- **Efficient way to summarize clusters:** Want memory required O(clusters) and not O(data)
- **IDEA: Rather than keeping points, BFR keeps summary statistics of groups of points**
  - **3 sets: Discard set, Compressed set, Retained set**
- **Overview of the algorithm:**
  - **1.** Initialize *K* clusters/centroids
  - **2.** Load in a bag of points from disk
  - **3.** Assign new points to one of the *K* original clusters, if they are within some distance threshold of the cluster
  - **4.** Cluster the remaining points, and create new clusters
  - **5.** Try to merge new clusters from step 4 with any of the existing clusters
  - **6.** Repeat steps 2-5 until all points are examined

# BFR Algorithm

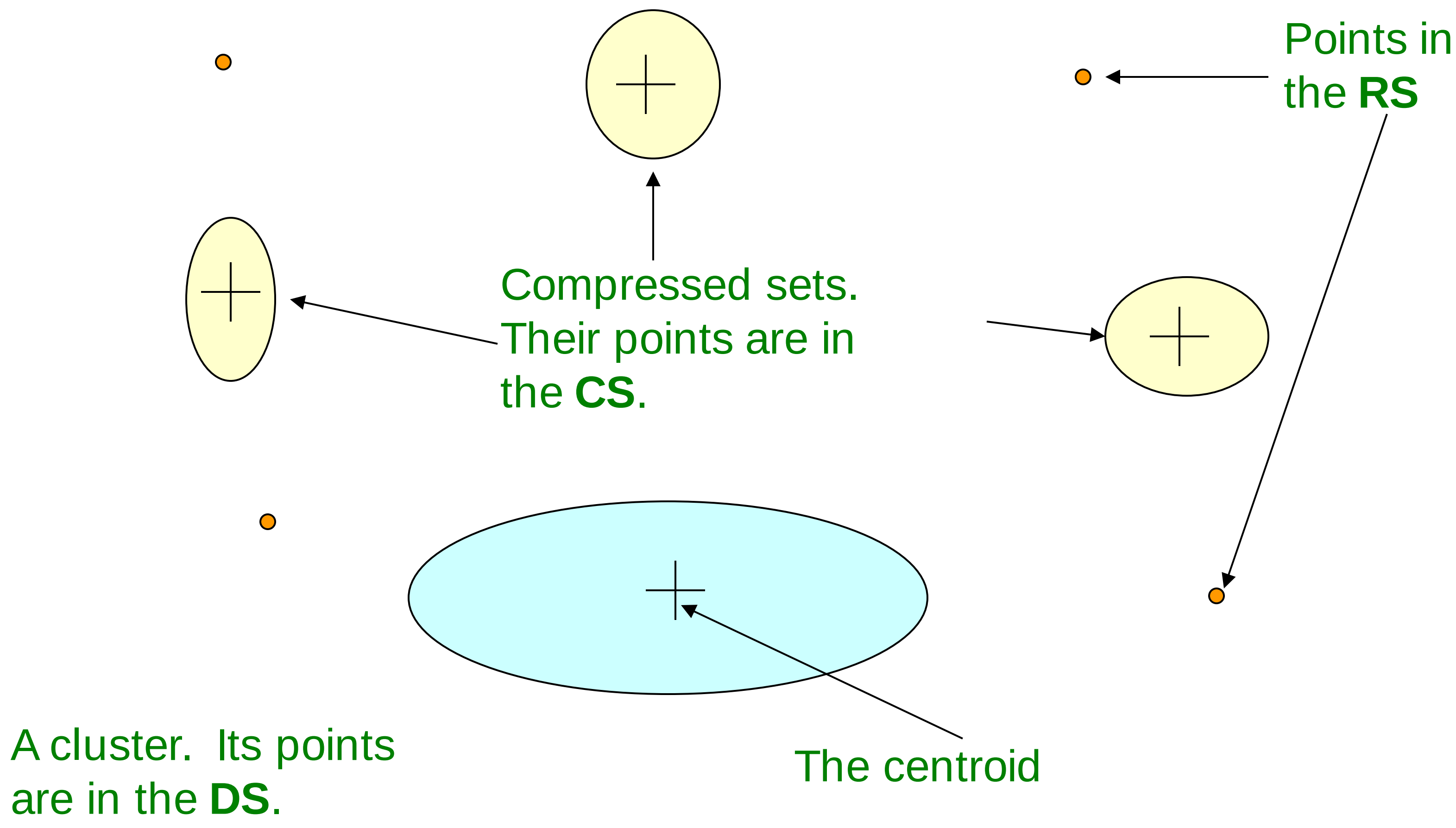
- **Points are read from disk one main-memory-full at a time**
- **Most points from previous memory loads are summarized by simple statistics**
- **Step 1)** From the initial load we select the initial ***k*** centroids by some sensible approach:
  - Take ***k*** random points
  - Take a small random sample and cluster optimally
  - Take a sample; pick a random point, and then ***k–1*** more points, each as far from the previously selected points as possible

# Three Classes of Points

**3 sets of points which we keep track of:**

- **Discard set (DS):**
  - Points close enough to a centroid to be summarized
- **Compressed 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 waiting to be assigned to a compression set

# BFR: "Galaxies" Picture

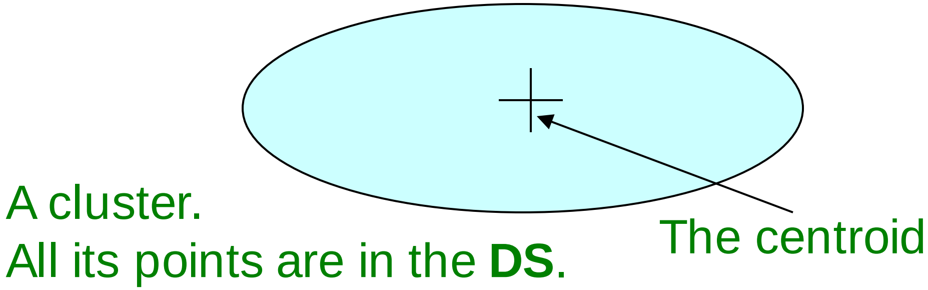



**Discard set (DS):** Close enough to a centroid to be summarized
**Compression set (CS):** Summarized, but not assigned to a cluster
**Retained set (RS):** Isolated points, we store them as they are

# Summarizing Sets of Points

**For each cluster, the discard set (DS) is <u>summarized</u> by:**

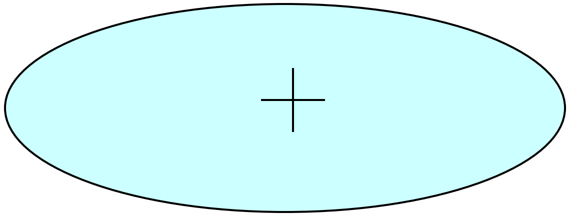
- The number of points, ***N***
- The vector ***SUM***, whose $i^{th}$ component is the sum of the coordinates of the points in the $i^{th}$ dimension
- The vector ***SUMSQ***: $i^{th}$ component = sum of squares of coordinates in $i^{th}$ dimension



# Summarizing Points: Comments

- **2*d* + 1** values represent any size cluster
  - ***d*** = number of dimensions
- Average in **each dimension** (**the centroid**) can be calculated as $\mathbf{SUM}_i / N$
  - $\mathbf{SUM}_i = i^{th}$ component of SUM
- Variance of a cluster's discard set in dimension *i* is: $(\mathbf{SUMSQ}_i / N) - (\mathbf{SUM}_i / N)^2$
  - And standard deviation is the square root of that
- **Next step: Actual clustering**

**Note:** Dropping the "axis-aligned" clusters assumption would require storing full covariance matrix to summarize the cluster. So, instead of **SUMSQ** being a ***d***-dim vector, it would be a ***d x d*** matrix, which is too big!



# The "Memory-Load" of Points

**Steps 3-5) Processing "Memory-Load" of points:**

- **Step 3)** 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
- **Step 4)** Use any in-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

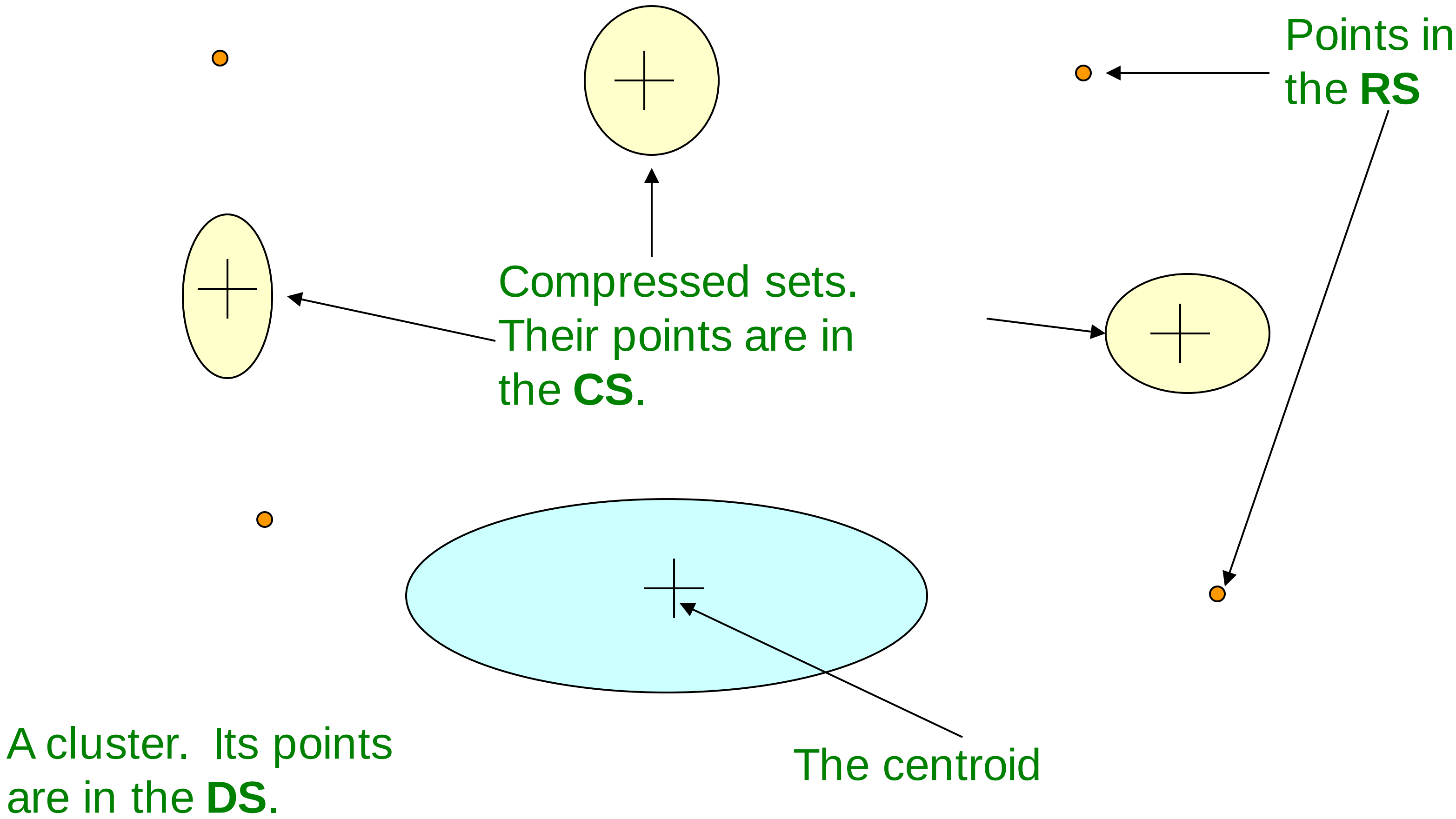
# The "Memory-Load" of Points

**Steps 3-5) Processing "Memory-Load" of points:**

- **Step 5) 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 **DS**
- **If this is the 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

# Summary: BFR




**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

# A Few Details…

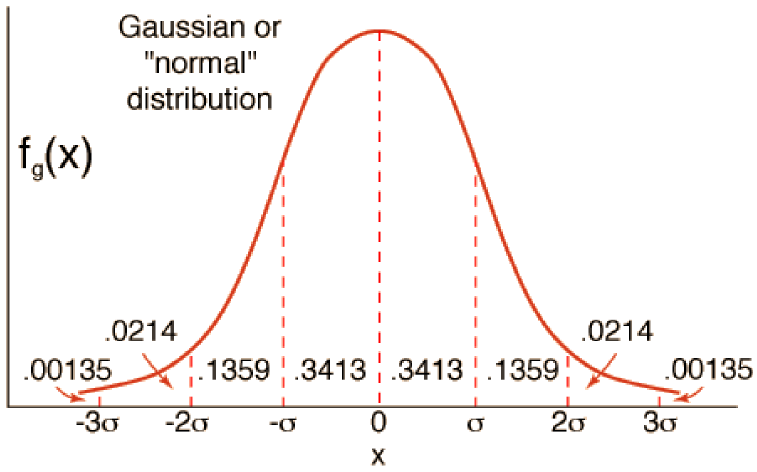
- **Q1) How do we decide if a point is "close enough" to a cluster that we will add the point to that cluster?**

- **Q2) How do we decide whether two compressed sets (CS) deserve to be combined into one?**

# How Close is Close Enough?

- **Q1) We need a way to decide whether to put a new point into a cluster (and discard)**

- **BFR suggests two ways:**
  - The **Mahalanobis distance** is less than a threshold
  - **High likelihood of the point belonging to currently nearest centroid**



# Mahalanobis Distance

- **Normalized Euclidean distance from centroid**

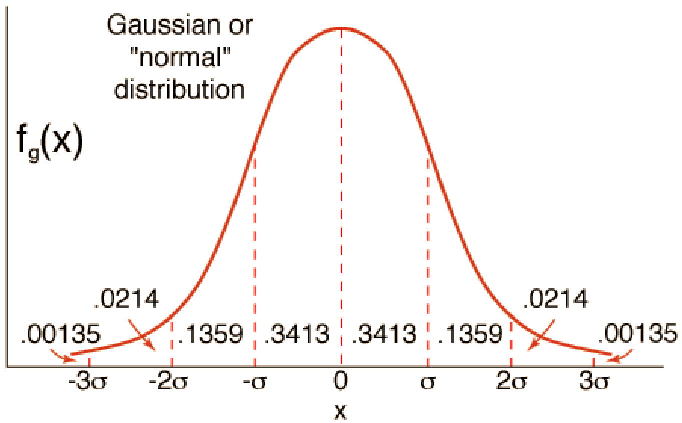
- For a given point $(x_1, \ldots, x_d)$ and a given centroid $(c_1, \ldots, c_d)$
  1. Normalize in each dimension: $y_i = (x_i - c_i) / \sigma_i$
  2. Take sum of the squares of the $y_i$
  3. Take the square root

$$d(x, c) = \sqrt{\sum_{i=1}^{d} \left( \frac{x_i - c_i}{\sigma_i} \right)^2}$$

$\sigma_i$ … standard deviation of points in the cluster in the $i^{th}$ dimension

# Mahalanobis Distance

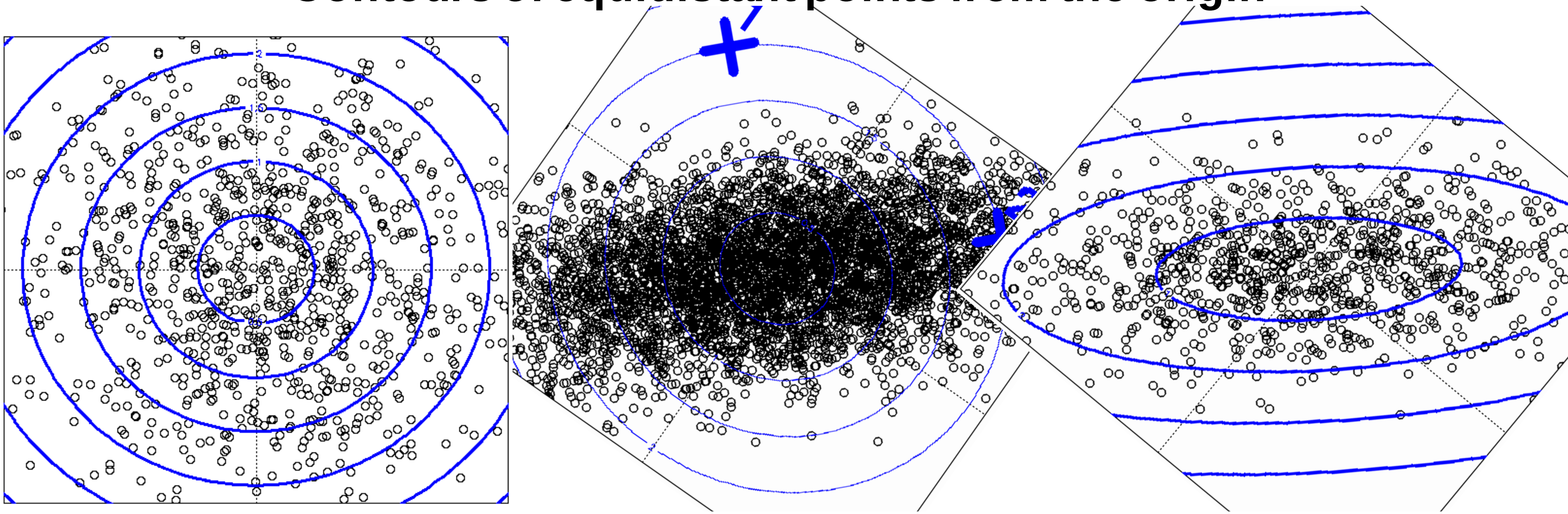
- If clusters are normally distributed in $d$ dimensions, then after transformation, one standard deviation = $\sqrt{d}$
  - i.e., 68% of the points of the cluster will have a Mahalanobis distance $< \sqrt{d}$
- Accept a point for a cluster if its M.D. is **<** some threshold, e.g. **2** standard deviations



# Picture: Equal M.D. Regions

- **Euclidean vs. Mahalanobis distance**

**Contours of equidistant points from the origin**



**Uniformly distributed points, Euclidean distance**

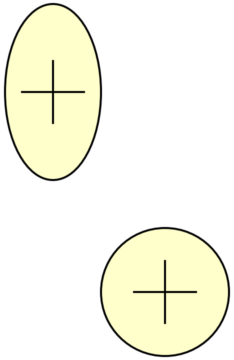
**Normally distributed points, Euclidean distance**

**Normally distributed points, Mahalanobis distance**

# Should 2 CS clusters be combined?

**Q2) Should 2 CS clusters be combined?**

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



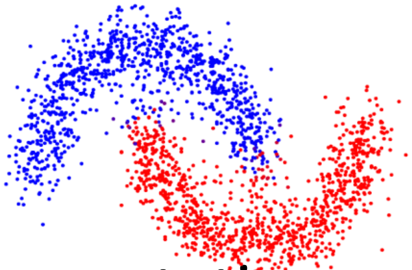
- **Many alternatives:** Treat dimensions differently, consider density

# The CURE Algorithm

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

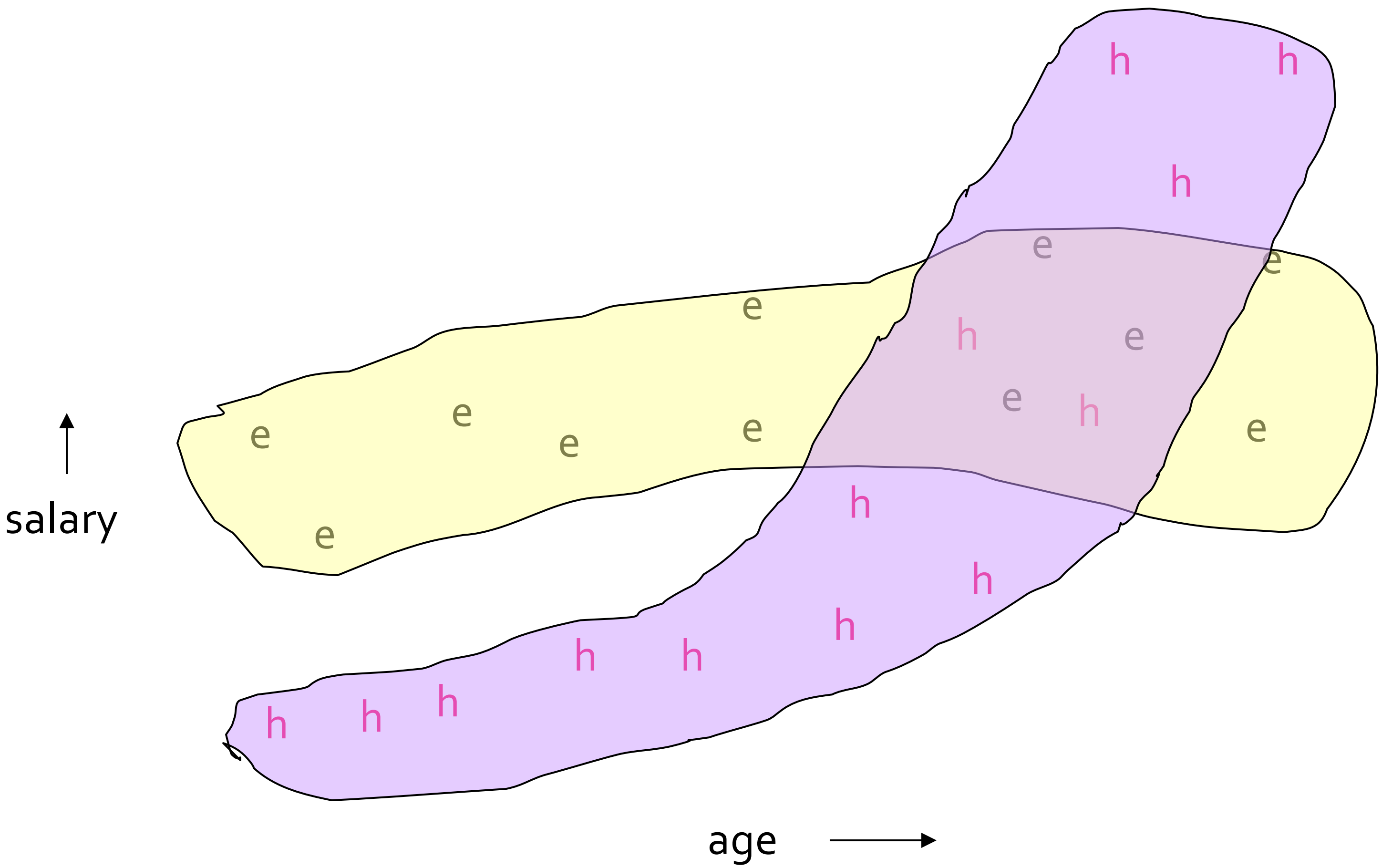
# CURE Algorithm

- **CURE (Clustering Using REpresentatives):**
  - Assumes a Euclidean distance
  - No assumption about shape of clusters
    - No need to be normally distributed in each dim
    - No need to have fixed axis
  - Instead of centroid, uses a collection of representative points to represent clusters
  - Assumes k=number of clusters is given



- In contrast, BFR and k-means assume:
  - clusters are normally distributed in each dimension
  - Axes are fixed – ellipses at an angle are not OK

# Example: Stanford Salaries

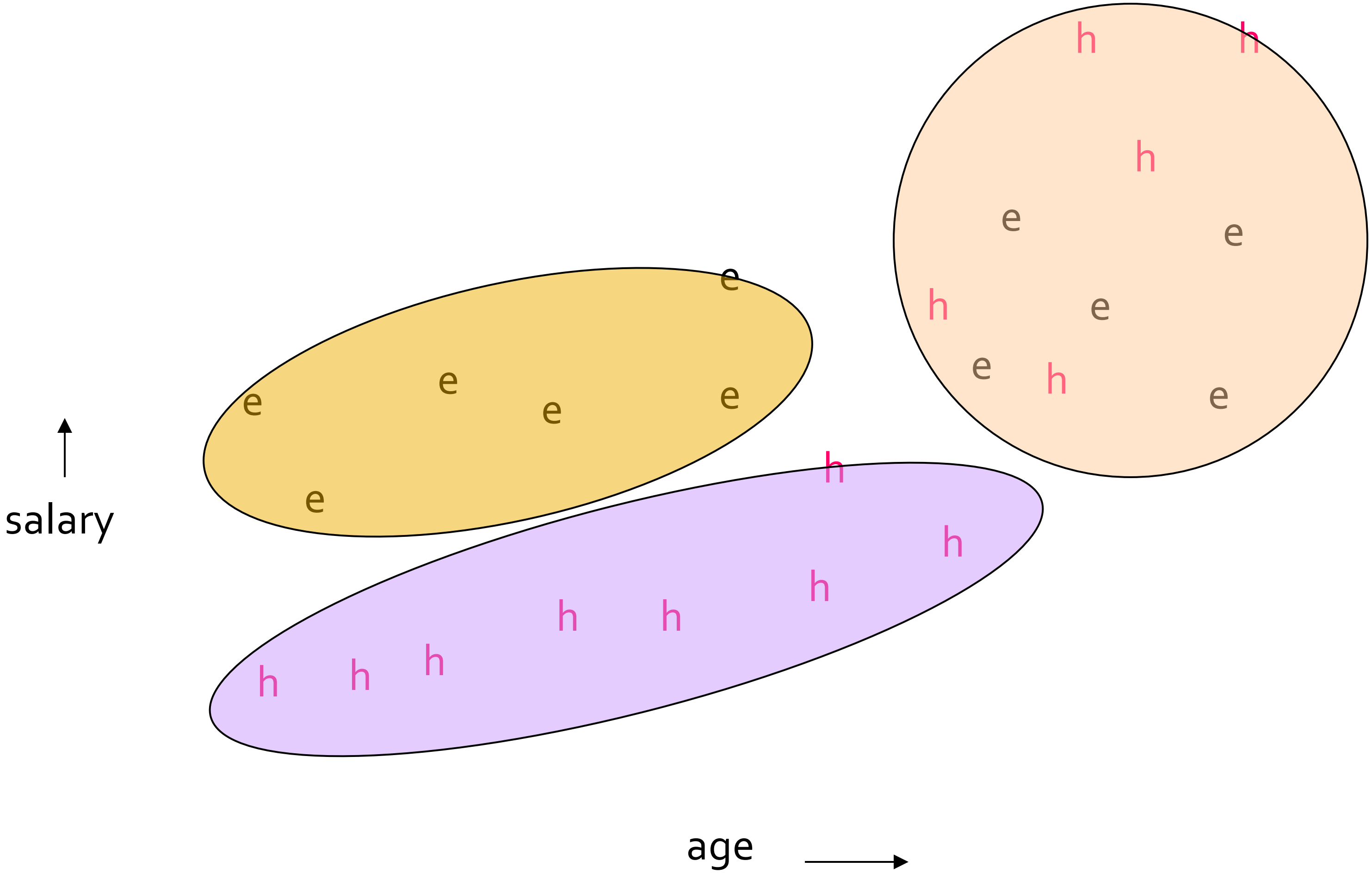


# Starting CURE

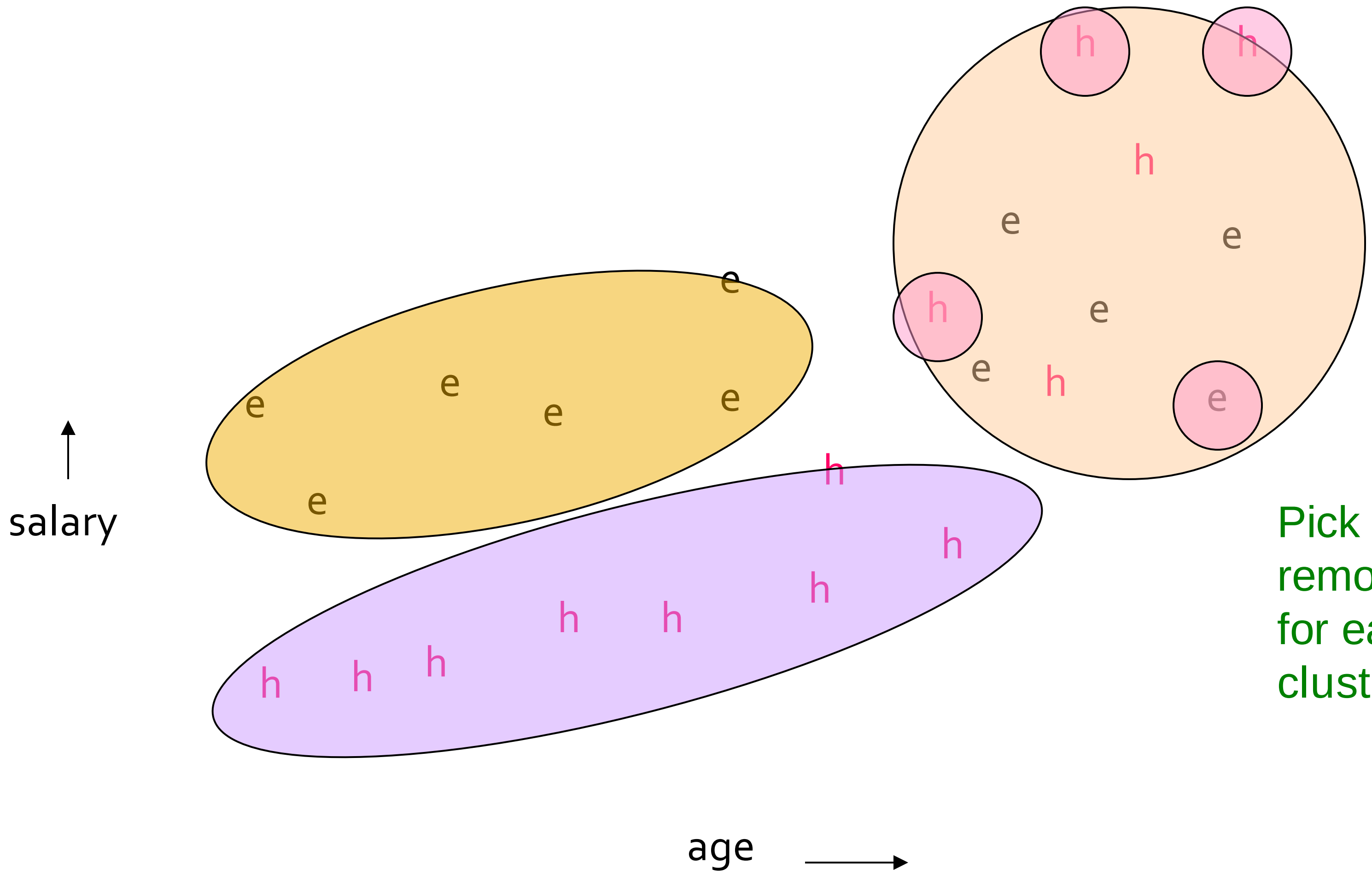
## 2 Pass algorithm. Pass 1:

- Pick a random sample of data and cluster them in main memory using hierarchical clustering
  - merge two clusters when they have close pair of points
- Pick representative points from each cluster
  - For each cluster, pick a sample of points, as dispersed as possible
  - move representatives a fraction of distance e.g. 20% toward the centroid of the cluster
  - Merge clusters with the closest pair of representatives

# Example: Initial Clusters

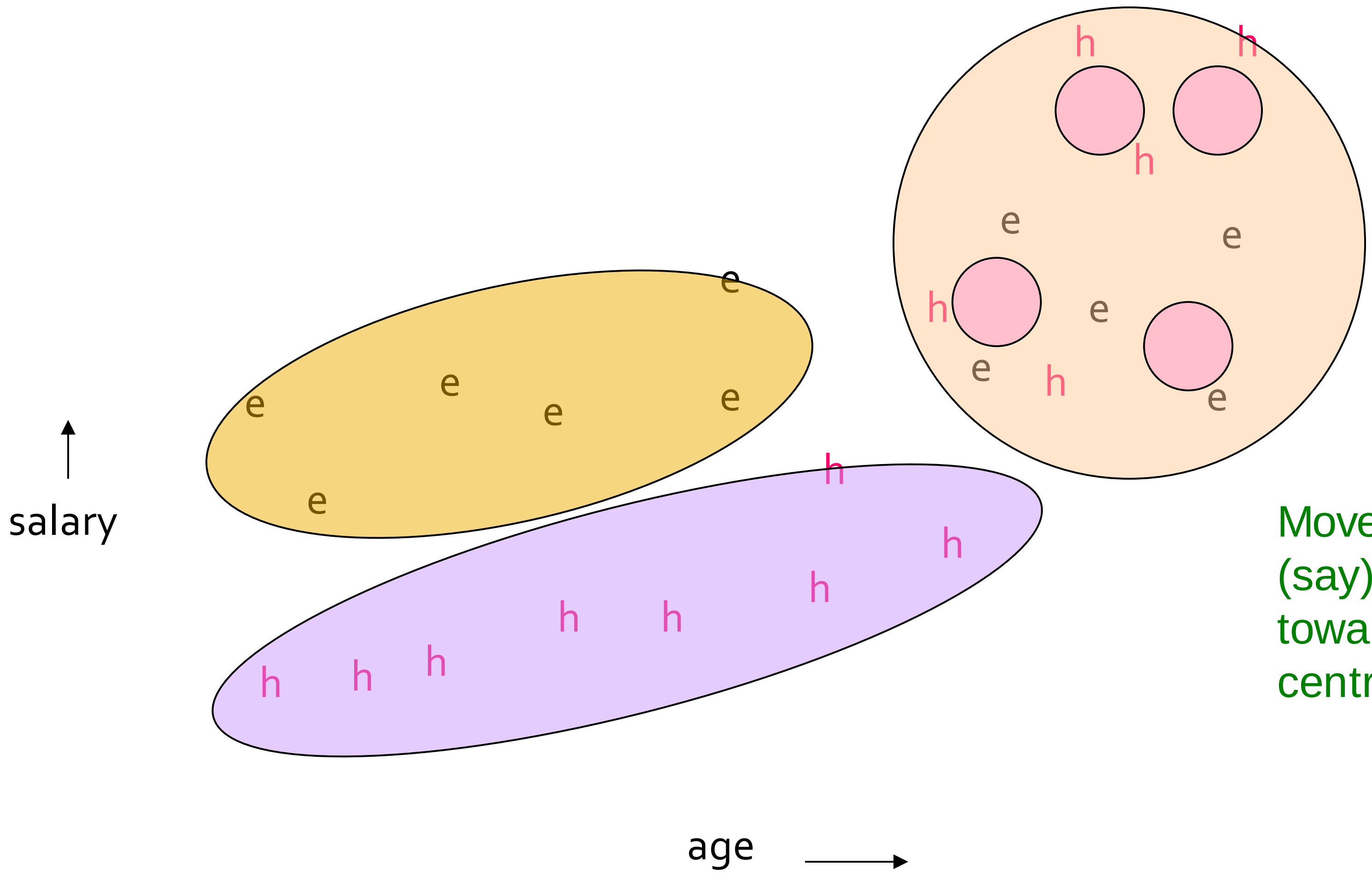


# Example: Pick Dispersed Points




Pick (say) 4 remote points for each cluster.

# Example: Pick Dispersed Points

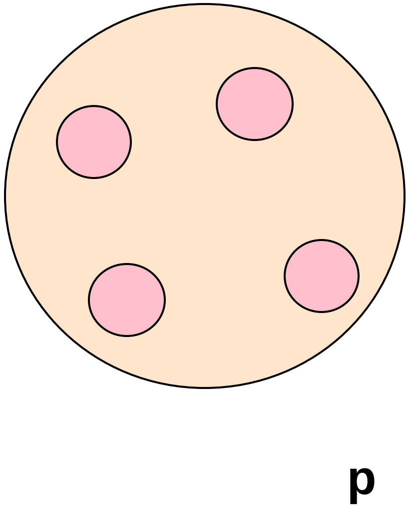



Move points (say) 20% toward the centroid.

# Finishing CURE

<u>**Pass 2:**</u>

- Now, rescan the whole dataset and visit each point ***p*** in the data set




- **Place it in the "closest cluster"**
  - Normal definition of "closest":
    Find the closest representative point to ***p*** and assign it to representative's cluster

# Why the 20% Move Inward?

## Intuition:

- If initial sample is *large* enough, some of the representatives will be on the *boundary* of clusters
  - Moving them towards centroid, move them inside

- A large, dispersed cluster will have larger moves as opposed to a small, dense cluster
  - Favors a small, dense cluster that is near a larger dispersed cluster

# Summary

- **Clustering:** Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***
- **Algorithms:**
  - Agglomerative **hierarchical clustering**:
    - Centroid and clustroid
  - ***k*-means:**
    - Initialization, picking *k*
  - **BFR**
  - **CURE**