k 8 1 d

Analyzing Massive Data Sets

Summer Semester 2019

Prof. Dr. Peter Fischer

Institut für Informatik

Lehrstuhl für Datenbanken und Informationssysteme

Chapter 5: Clustering

High-Dimensional Data and Similarity

- First *conceptual* and *algorithmic* part of the lecture
- Two core concepts:
  - **High-Dimensional Data**: Data items represented by many data points (hundreds, thousands, … possibly out of a much large space)
  - Analyzing a *single* or *few* dimensions insufficient to understand items
  - **Similarity/Distance**: Expressing pair-wise similarity over all features

- Applications:
  - **Finding Similar Items**: pairwise (this chapter)
  - **Clustering**: Identify structure / groups using similarity
  - **Retrieval**: Similarity between search expression and data set

- Strategies for massive volumes:
  - **Model of clustering** has significant impact on **complexity** (and **semantics**)
  - Getting from main memory to disk requires additional tweaking

High Dimensional Data

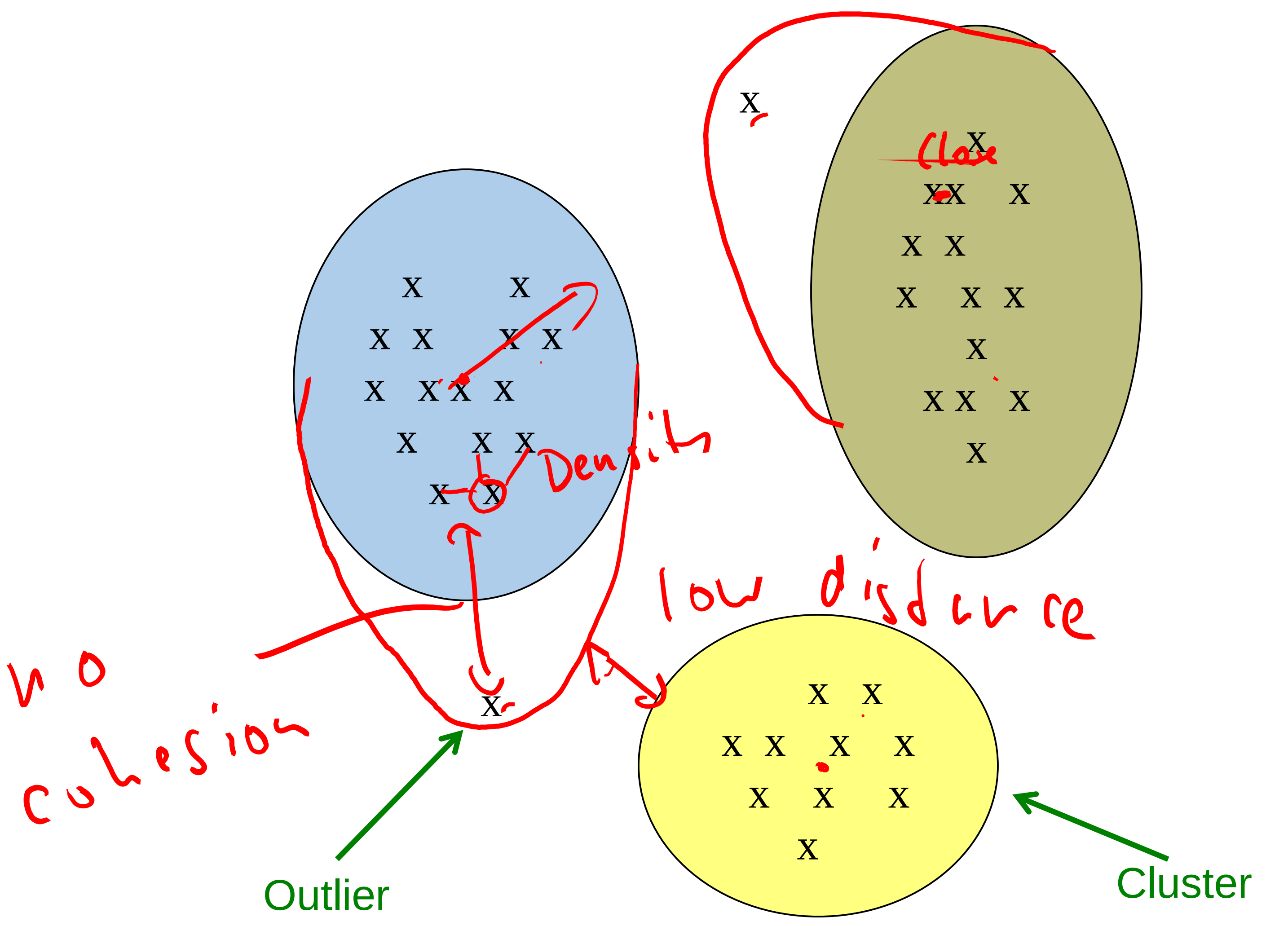
- **Given a cloud of data points we want to understand its structure**

The Problem of Clustering

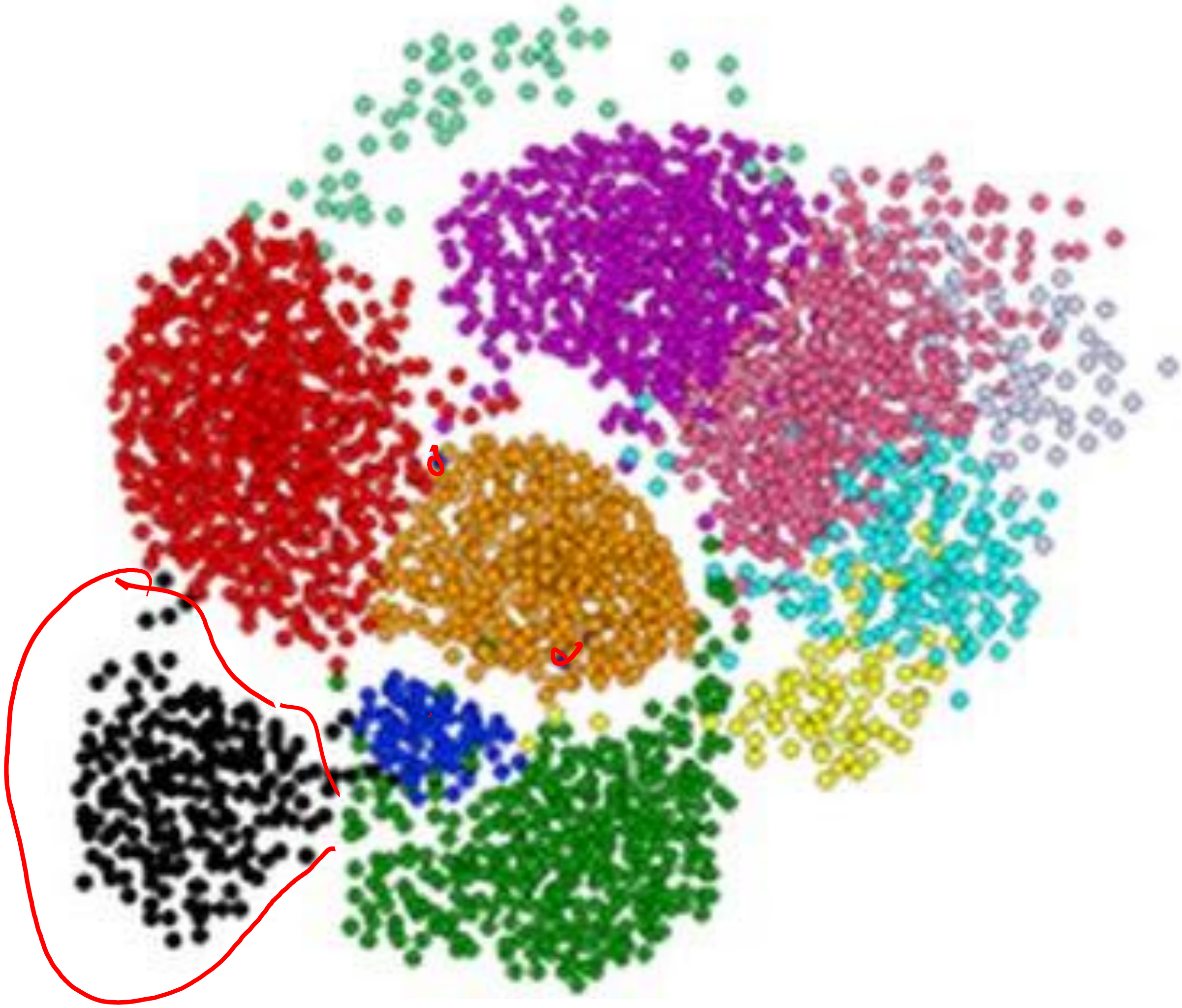
- Given a **set of points**, with a notion of **distance** between points, **group the points** into some number of ***clusters***, so that
  - Members of a cluster are close/similar to each other
  - Members of different clusters are dissimilar
- **Usually:**
  - Points are in a high-dimensional space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

Example: Clusters & Outliers

2D space



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

0 1

- Many applications involve not 2, but 10 or 10,000 dimensions
- **High-dimensional spaces look different:** Almost all pairs of points are at about the same distance (unless there is a very strong imbalance)
- Intuition for Euclidean Distance (similar for others): $\sqrt{\sum |x_i - x_j|^2}$
  - Assume a value space in each dimension from 0 to 1
  - Single dimension: points can be between 0 and 1 apart, average is 1/3
  - With many dimensions d, at least one dimension will be close to 1 – stets
  - Therefore, the minimum value is also at least 1 all 1
  - The maximum value is $\sqrt{d}$, but few dimensions actually close to maximum
  - Most values will be around the average

Very high likelihood

Clustering Problem: Music CDs

- **Intuitively: Music divides into categories, and customers prefer a few categories**
  - But what are categories really? (content-based recommendation)

- Represent a CD by a **set of customers** who bought it: **Similar CDs** have **similar sets of customers**, and vice-versa (collaborative recommendation)
- 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 $(x_1, x_2,\ldots, x_k)$, where $x_i = 1$ iff the i^{th} customer bought the CD

- For Amazon, the dimension is hundreds of millions

- **Task:** Find clusters of similar CDs

Clustering Problem: Documents

Finding topics:

- Represent a document by a vector ($x_1, x_2, \ldots, x_k$), 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**

Huge Design Space for Clustering

- Cluster Models:
  - *Connectivity: nearest fitting object*
  - *Centroid: common center point*
  - *Distributions: typically around "centers"*
  - *Density: require substantial amount neighbors before linking*
  - *Graphs: Cliques, high connectedness*
  - *Subspace: consider only certain dimensions*
  - *Neural networks: self-organizing maps*

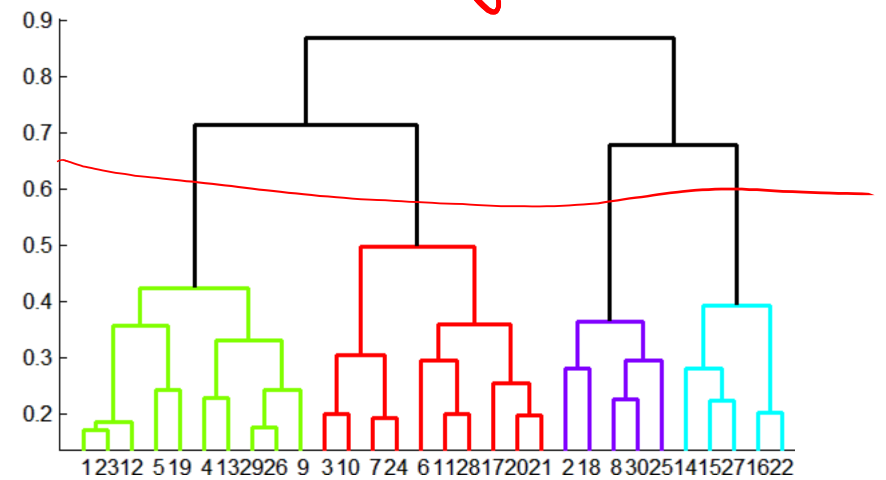
- *Assignment:*
  - *Hard clustering: at most in one cluster*
  - *Soft/fuzzy clustering: probability to belong to one or more clusters*
- *Refined Assigment:*
  - *Strict partitioning (with outliers)*
  - *Overlapping: at same level, often "hard"*
  - *Hierarchical: contained in parent set*

Focus of the lecture

dendrogram

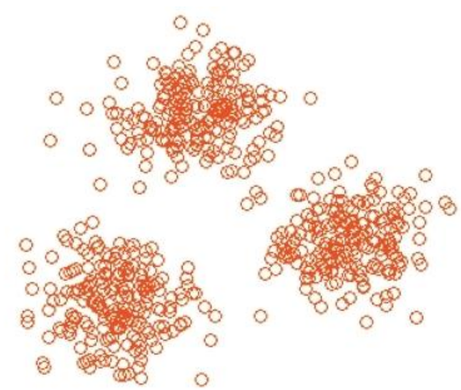
1) Hierarchical (Linkage):

- **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



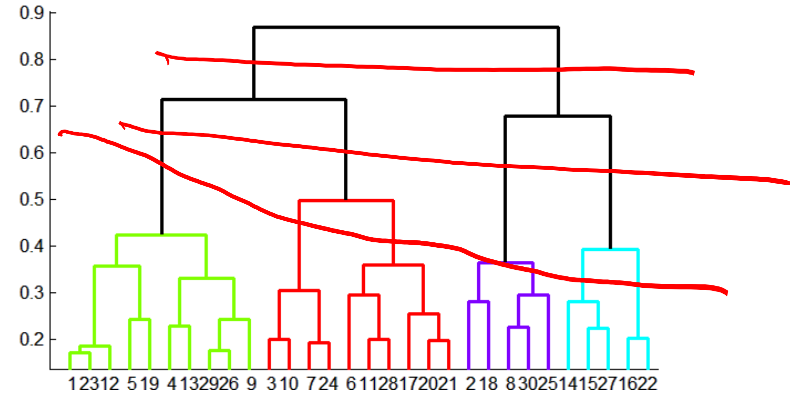
2) Point assignment (partitioning):

- Maintain a set of clusters
- Points belong to “nearest” cluster



Hierarchical Clustering

- **Key operation:** (agglomerative)
  Repeatedly combine two nearest clusters
- **In case of ties, break randomly**



- **Three important questions:**
  1) How do you represent a cluster of more than one point?
  2) How do you determine the “nearness” of clusters?
  3) When to stop combining clusters?

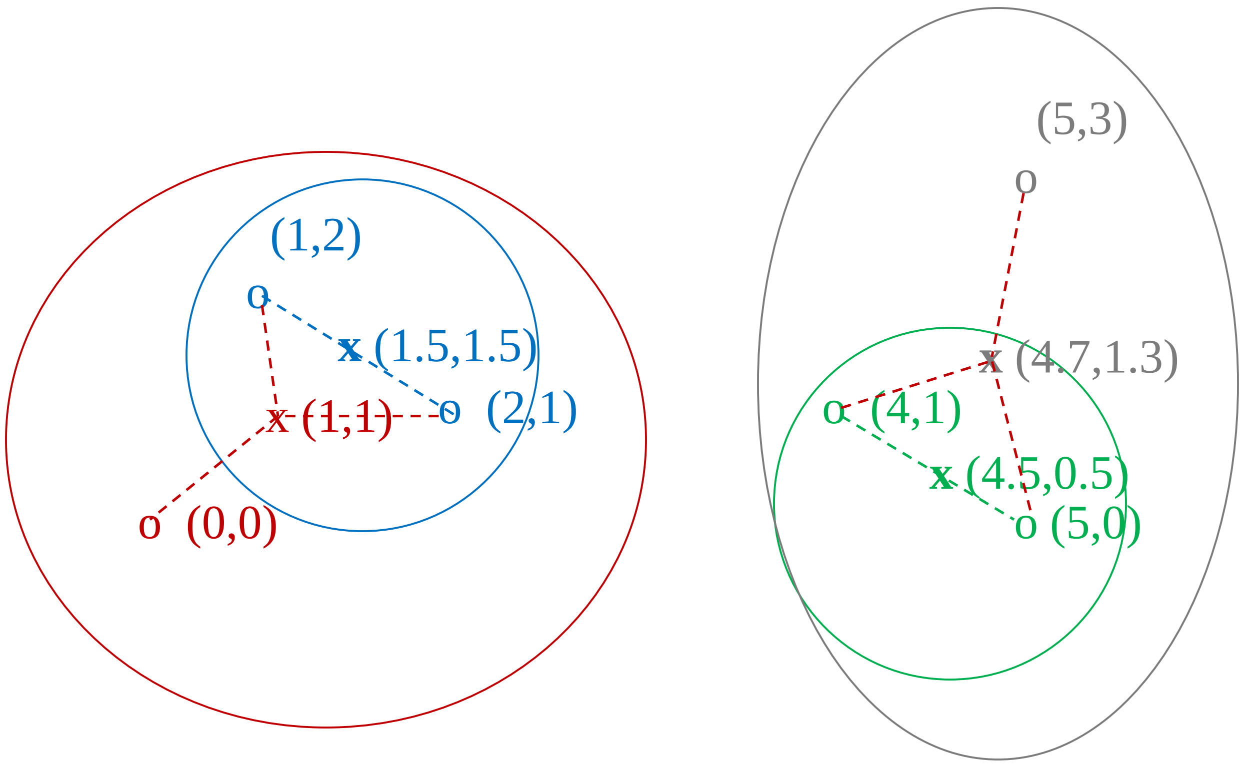
- Multiple options for 3, mostly orthogonal to 1 and 2:
  - Number of clusters known beforehand and reached
  - Single "root" cluster: utilize hierarchy
  - Quality of clusters too low or no longer increasing

Hierarchical Clustering

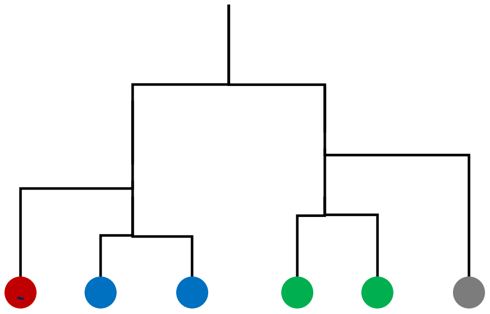
- **Key operation: Repeatedly combine two nearest clusters**
- **(1) How to represent a cluster of many points?**
  - **Key problem:** As you merge clusters,
    how do you represent the "location" of each cluster,
    to tell which pair of clusters is closest?
- **Euclidean case:** each cluster has a
  centroid = average of its (data)points

- **(2) How to determine "nearness" of clusters?**
  - Measure cluster distances by distances of centroids

Example: Hierarchical clustering



Data:
o … data point
x … centroid

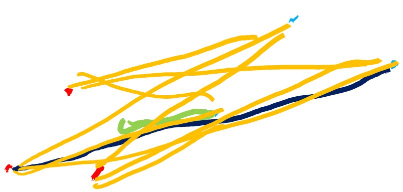


Dendrogram

And in the Non-Euclidean Case?

What about the Non-Euclidean case?

- The only "locations" we can talk about are the points themselves
  - i.e., there is no "average" of two points
  - Think of two sets or two strings!
- **Approach 1:**
  - **(1) How to represent a cluster of many points?**
    No explicit representative
  - **(2) How do you determine the "nearness" of clusters?**
    - **1.1 Minimum distance of any two nodes in either cluster (single linkage)**
    - **1.2. Maximum distance of any two nodes in either cluster (complete linkage)**
    - **1.3. Average distance among all pairs of nodes in each cluster/union …**



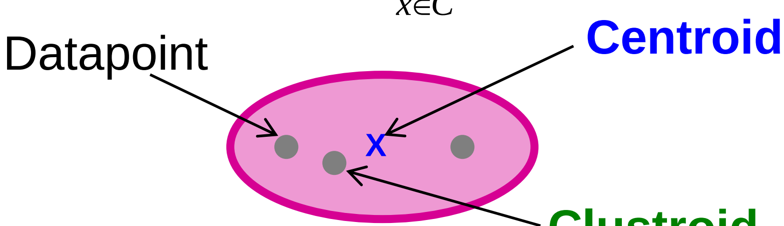
- **Approach 2:**
  - **(1) How to represent a cluster of many points?**
    clustroid = (data)point "***closest***" to other points
  - **(2) How do you determine the "nearness" of clusters?**
    Treat clustroid as if it were centroid, when computing inter-cluster distances

~ median instead mean

“Closest” Point?

- **(1) How to represent a cluster of many points?**
  clustroid = 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 ↪ Euclidean
    - 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.

Defining "Nearness" of Clusters

- **(2) How do you determine the "nearness" of clusters?**
  - **Approach 3:**
    Intercluster distance = minimum of the distances between any two points, one from each cluster
  - **Approach 4:**
    Pick a notion of "**cohesion**" (intracluster quality) of clusters, *e.g.*, maximum distance from the clustroid
    - Merge clusters whose *union* is most cohesive
    - **Approach 4.1:** Use the **diameter** of the merged cluster = maximum distance between points in the cluster
    - **Approach 4.2:** Use the **average distance** between points in the cluster
    - **Approach 4.3:** Use a **density-based approach**
      - Take the diameter or avg. distance, e.g., and divide by the number of points in the cluster

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**

- Constrained/naïve linkage models allow $O(N^2)$
  - **But often create degraded clusters**

k-means clustering

k–means Algorithm(s)

- Assumes Euclidean space/distance
- Minimizes
  - Squared distances to cluster centers
  - Variance among cluster members

- Start by picking ***k***, the number of clusters
  (most difficult problem in practice)

- Initialize clusters by picking one point per cluster
- Result quality and runtime often depend on correct initialization
- Wide range of approaches
  - **Naïve:** Pick all k points at random
  - **Simple improvement:** Pick one point at random, then ***k*-1** other points, each as far away as possible from the previous points.

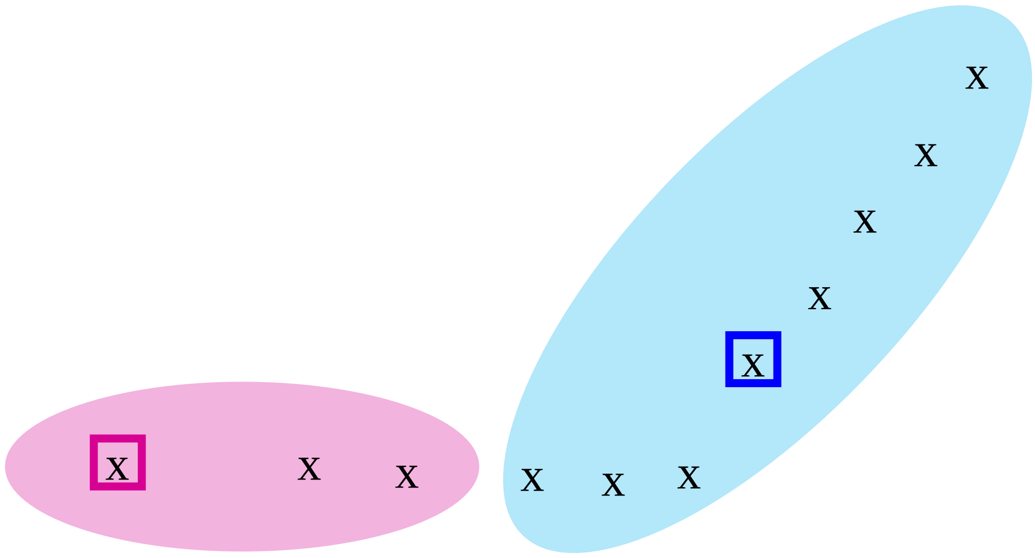
    Drawback: outliers
  - **K-Means++:** Middle ground, draw from distribution

Populating Clusters

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
- Note: Generalization for different models
  GMM – EM (statistical assignment)
  - Same steps
  - More general meaning: assign to distribution, adapt distribution

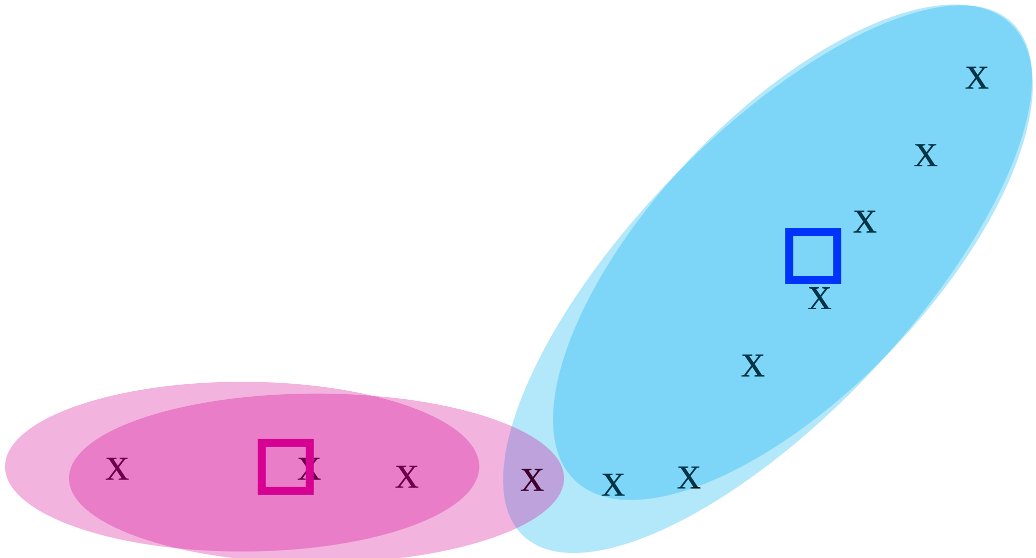
Example: Assigning Clusters



x … data point
… centroid

Clusters after round 1

Example: Assigning Clusters



x … data point
… centroid

Clusters after round 2

Example: Assigning Clusters

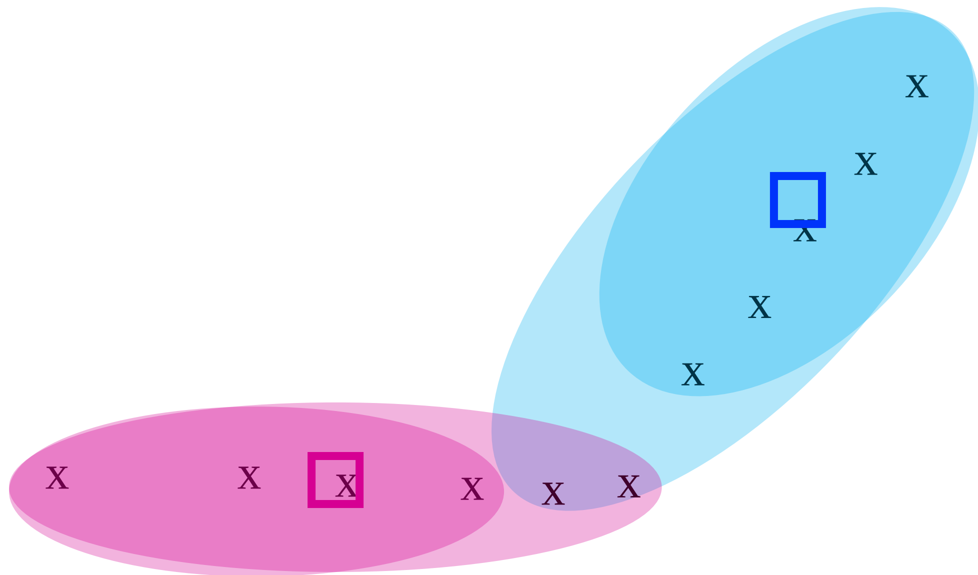
Complexity

→ 10s → 100s

N · k · i ← Iterations

Points

Centroids



x … data point

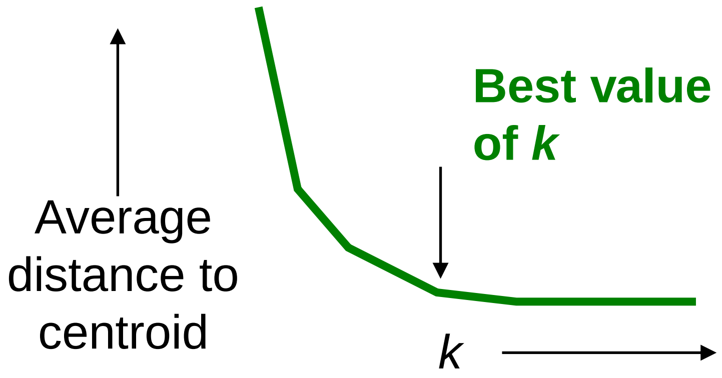
… centroid

Clusters at the end

Getting the *k* right

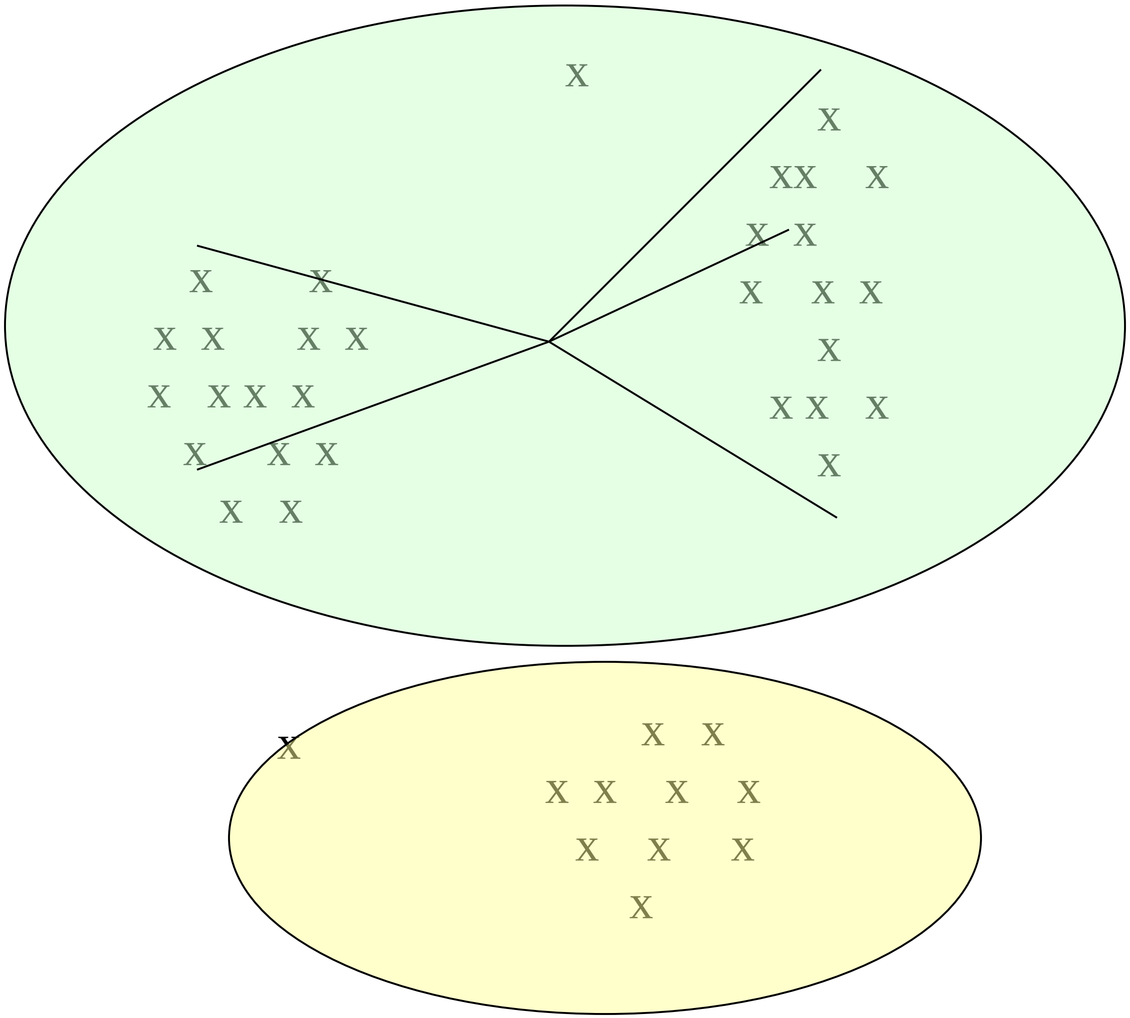
How to select *k*?

- 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



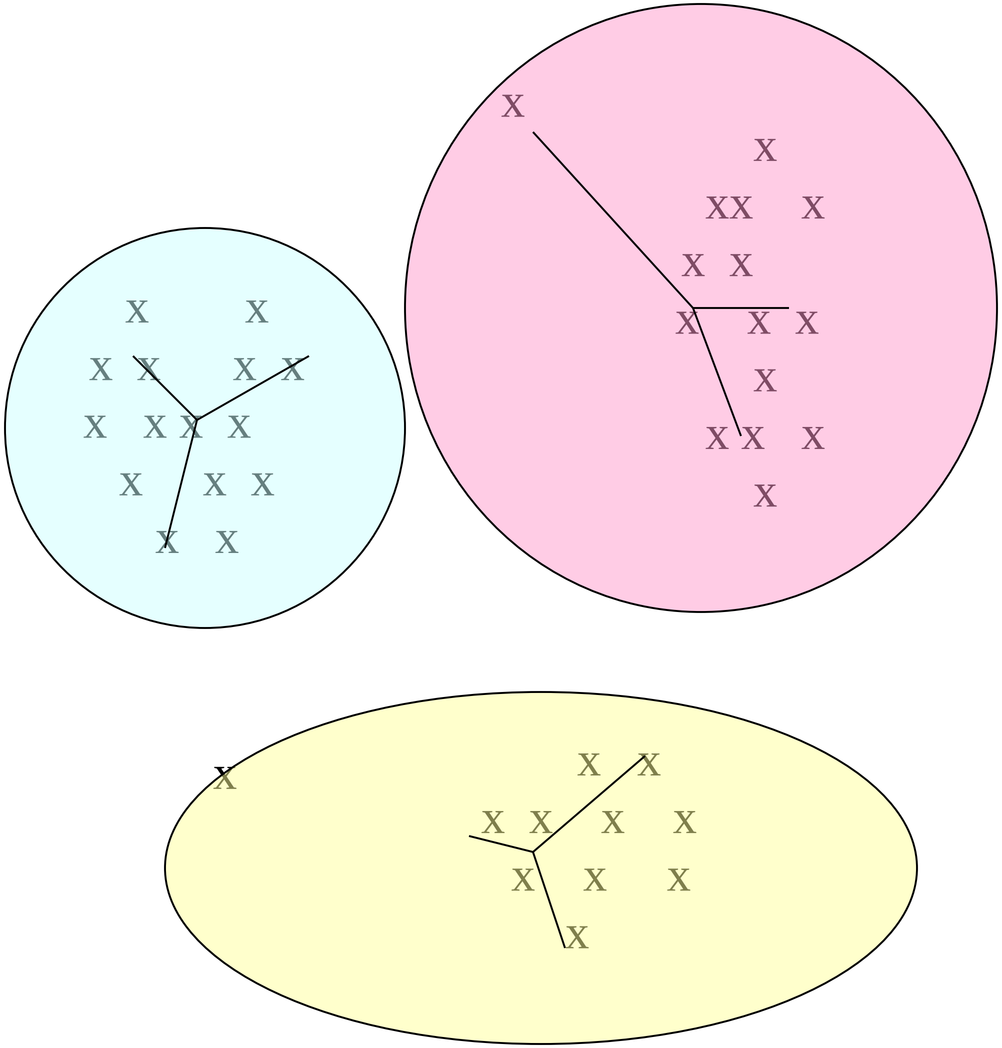
Example: Picking *k*

Too few;
many long
distances
to centroid.



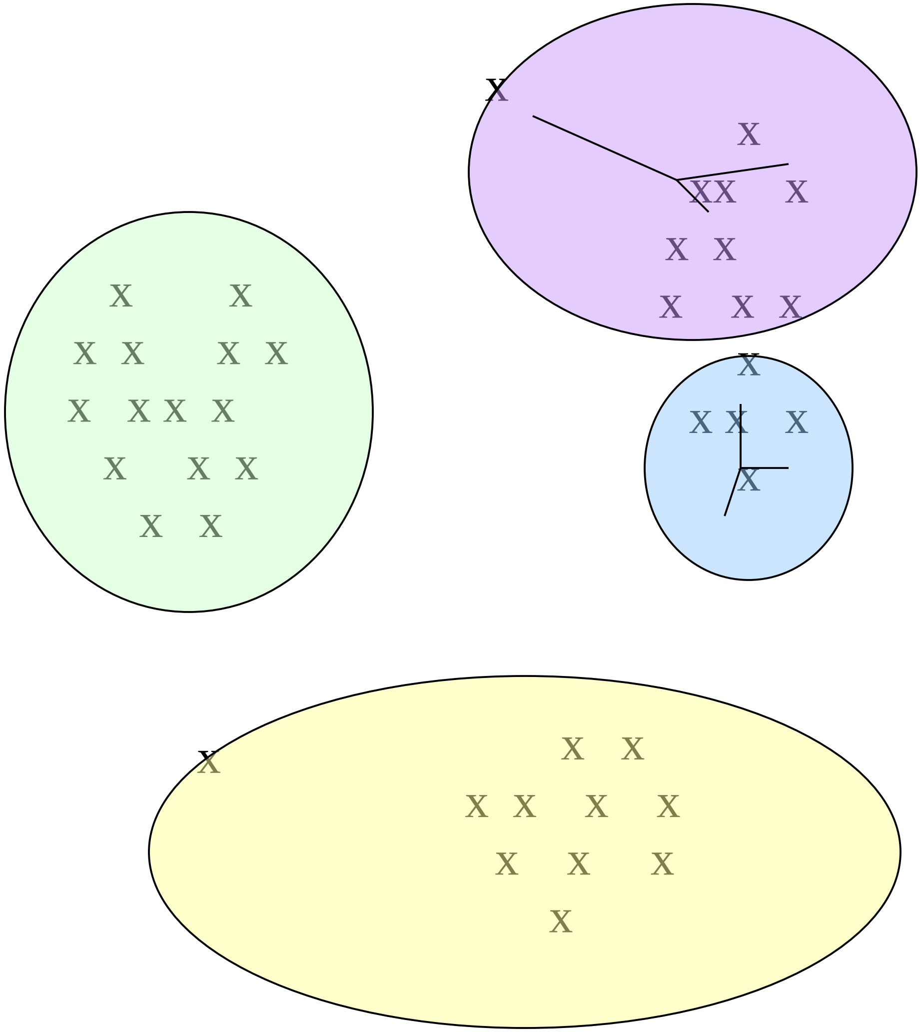
Example: Picking *k*

Just right;
distances
rather short.



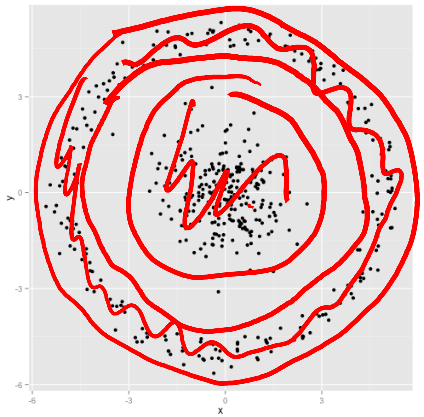
Example: Picking *k*

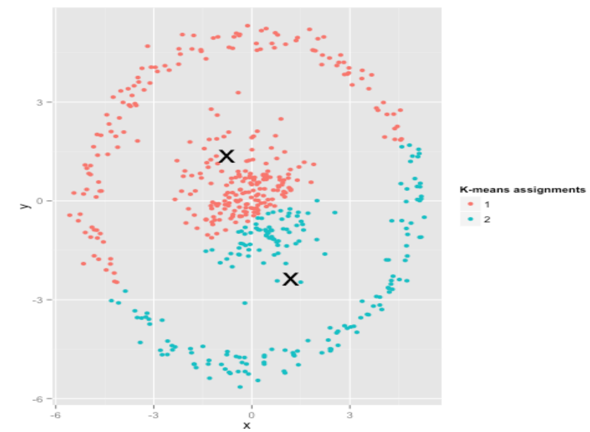
Too many;
little improvement
in average
distance.



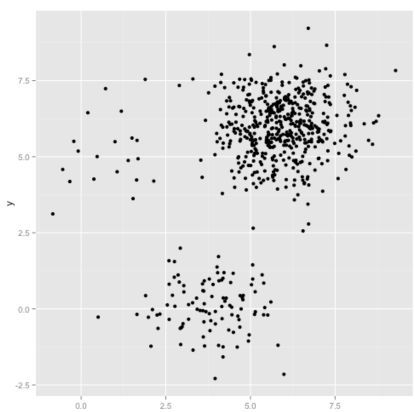
Limitations of k-means

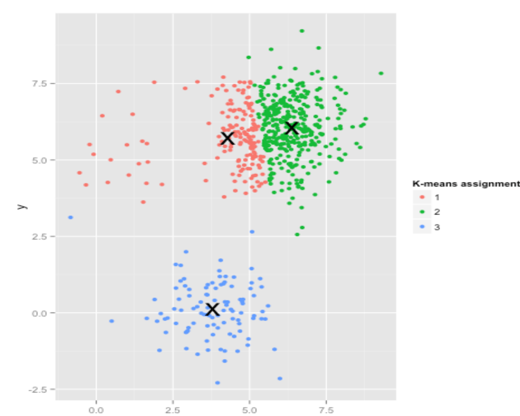
- Assumes "spherical data"





- Assumes evenly sized clusters

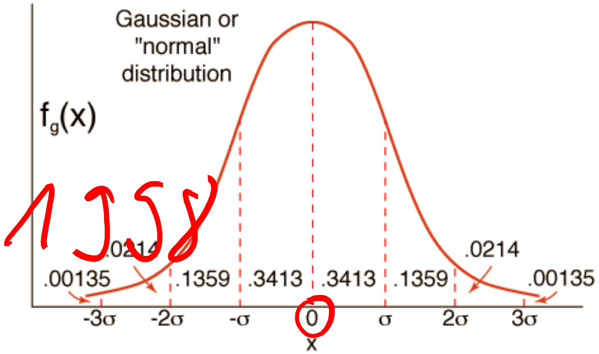




The BFR Algorithm

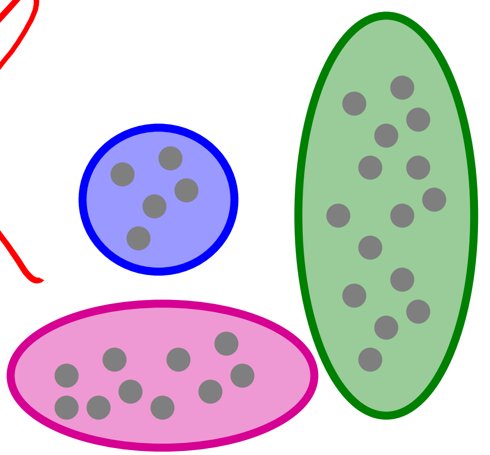
Extension of *k*-means to large data

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
- **Efficient way to summarize clusters** (want memory required O(clusters) and not O(data))

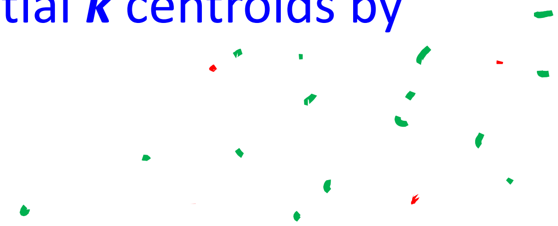


BFR Algorithm - Overview

- Standard k-means needs to access **every data element at every iteration** (to check / adapt points with cluster centers)
- In BFR, points are read from disk one main-memory-full at a time → 100s of speed up
  - Build initial information by sampling
  - Load data block (~ RAM size) and process it
  - Single pass over full data
  - Incremental results possible: Return initial, not fully precise results, improve with additional data

BFR Summaries and Initialization

- **Most points from previous memory loads are summarized by simple statistics**
- To begin, 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



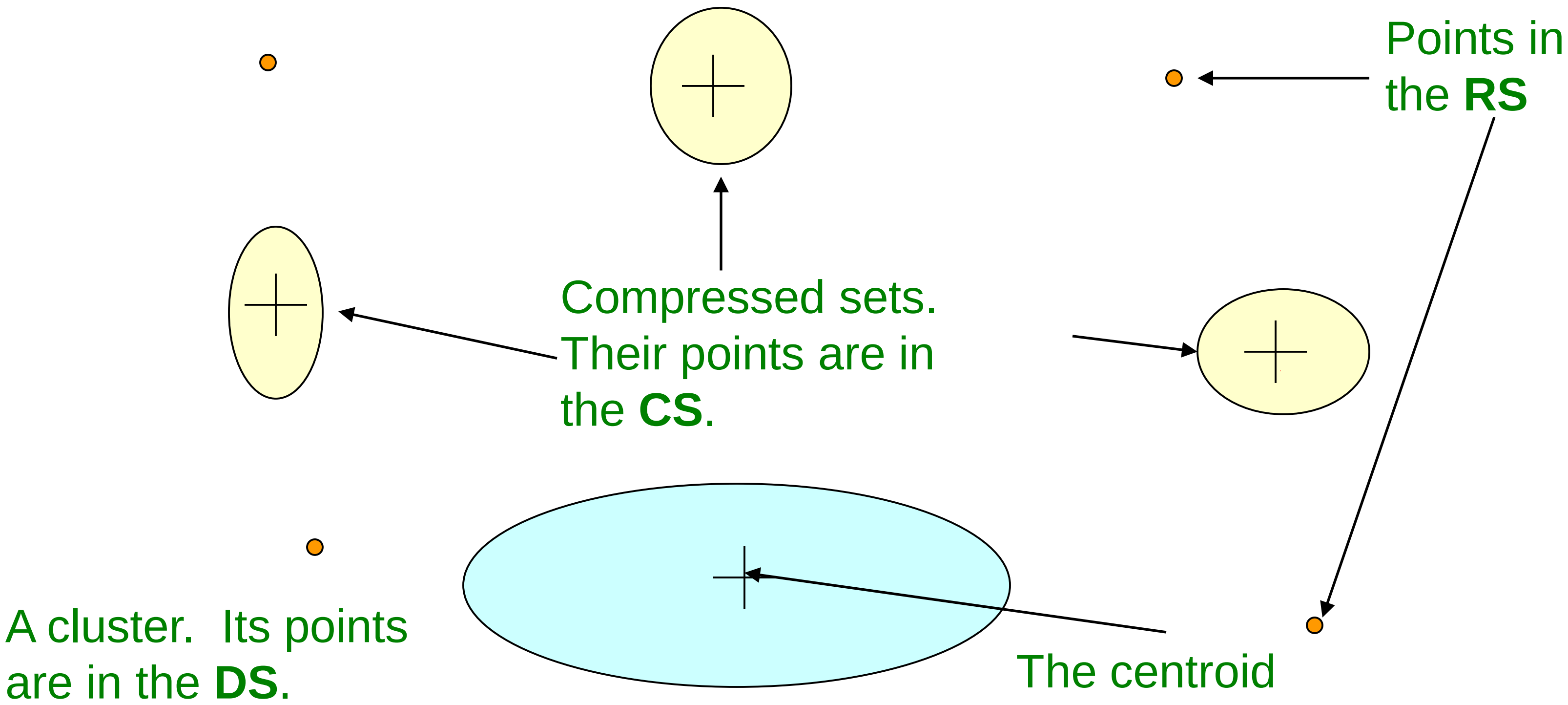
  - Take a sample; pick a random point, and then expensive possible
    k–1 more points, each as far from the previously selected points as possible

Summary: Three Classes of Points

3 sets of points which we keep track of:

- **Discard set (DS):** → works well → belong strongly to cluster
  - Points close enough to a centroid to be summarized → k
- **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 → more than k "clusters"
- **Retained set (RS):** keep as is
  - Isolated points waiting to be assigned to a compression set

BFR: "Galaxies" Picture



Intuition:
- Due to the data distribution, most points are close to a center
- Summaries are mostly precise and cover many points in little space

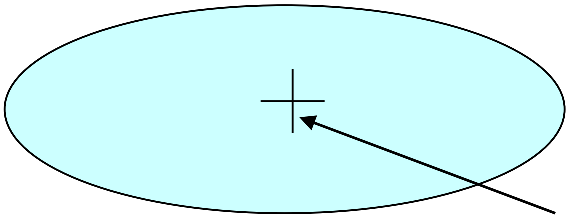
Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

Summarizing Sets of Points

For each cluster, the discard set (DS) is summarized by:

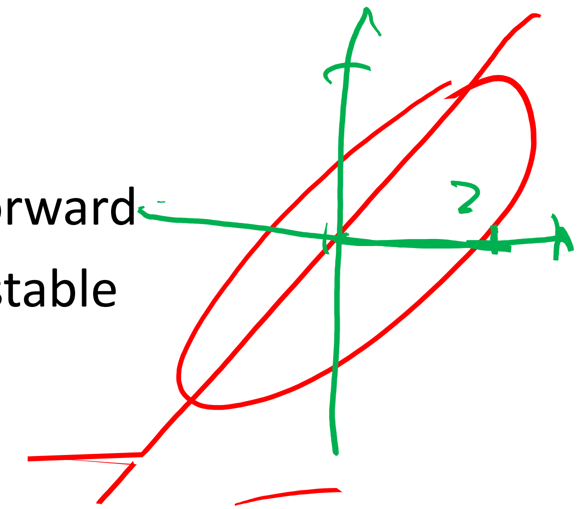
- The number of points, ***N***
- 1 The vector ***SUM***, whose i^{th} component is the sum of the coordinates of the points in the i^{th} dimension
- 2 The vector ***SUMSQ***: i^{th} component = sum of squares of coordinates in i^{th} dimension

- Space requirements (***d*** = number of dimensions)
  - **2*d* + 1** values represent any size cluster
  - Without summary: **d * n**



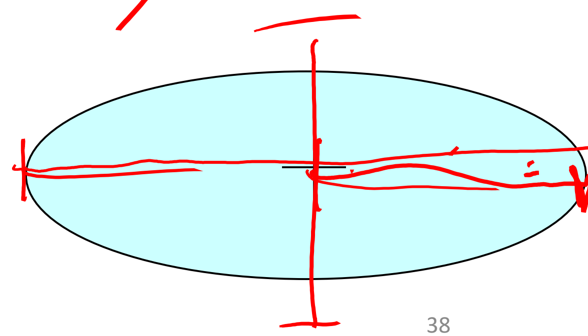
Summarizing Points: Comments

- 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

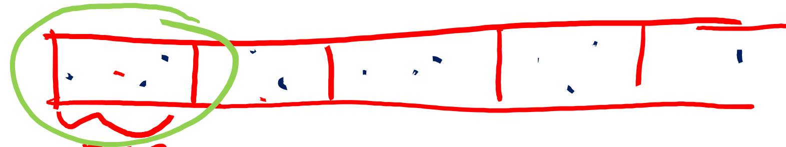


- Adding points or full clusters to cluster are straightforward
- Unless we run into an overflow, the computation is stable
- **Next step: Actual clustering**

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



RAM size

Processing the “Memory-Load” of points (1):

- **1)** 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
- **2)** 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

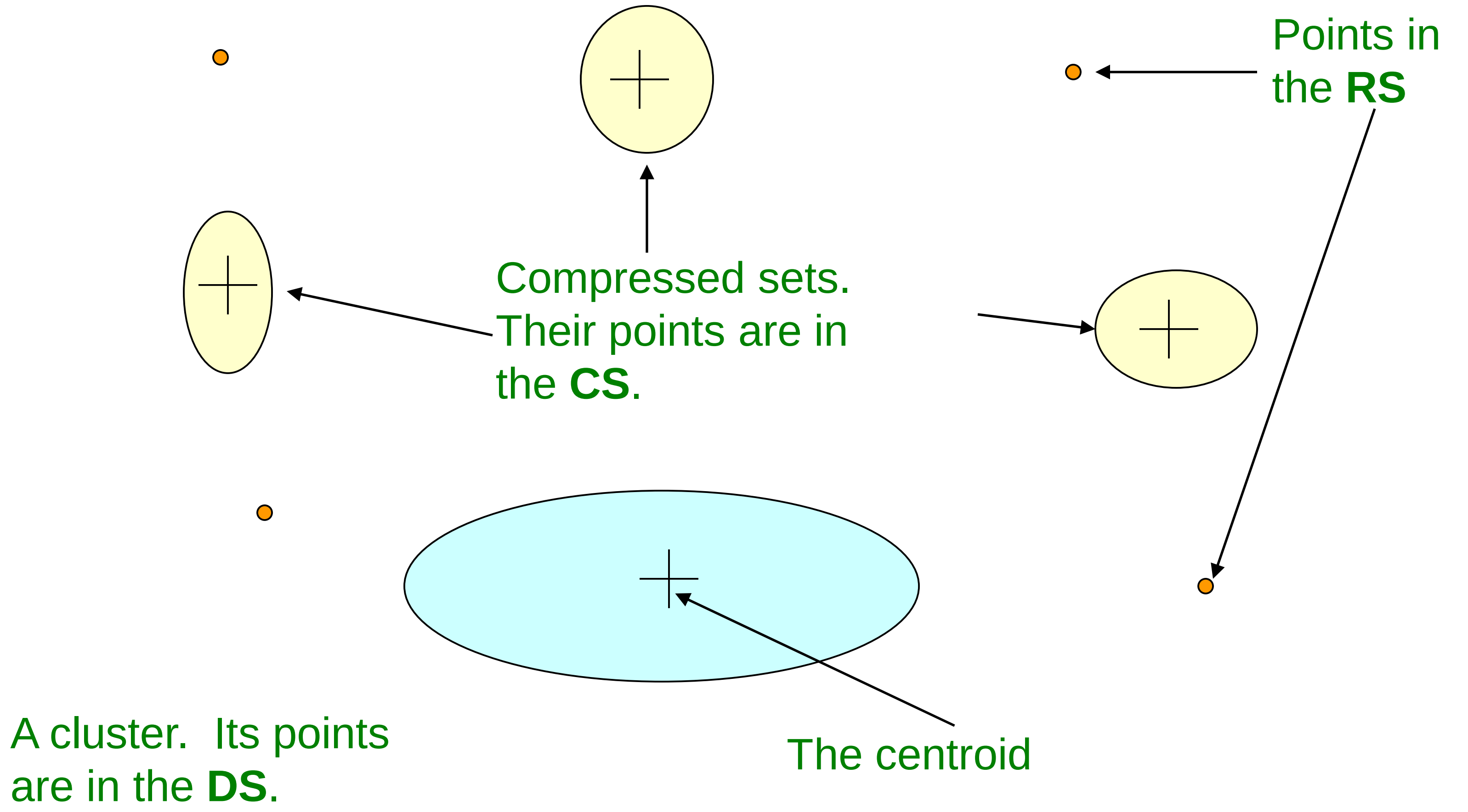
The "Memory-Load" of Points

Processing the "Memory-Load" of points (2):

- **3) DS set:** Adjust statistics of the clusters to account for the new points
  - Add ***N***s, ***SUM***s, ***SUMSQ***s
- **4)** Consider merging compressed sets in the **CS**
- **5)** 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

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

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?**

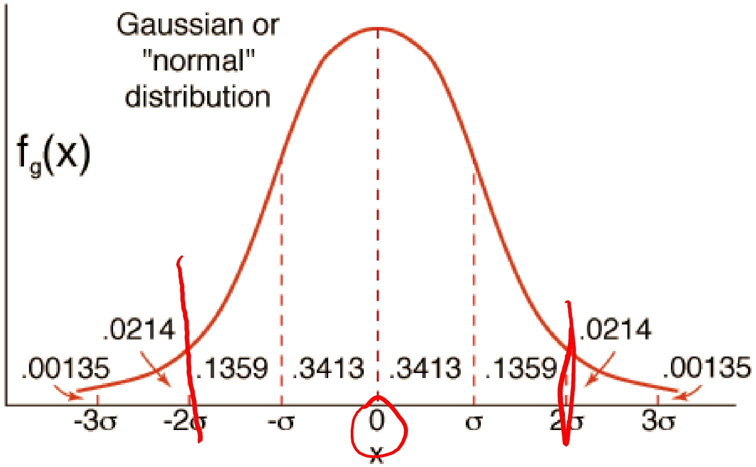
- **Q2) How do we decide whether two compressed sets (CS) deserve to be combined into one?**

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:**
  1. High likelihood of the point belonging to currently nearest centroid
     - Many assumptions on current and future distribution
  2. **The Mahalanobis (Normalized Euclidean) distance is less than a threshold**
     - Normalize in each dimension: $y_i = (x_i - c_i) / \sigma_i$ ***(Standard Deviation)***

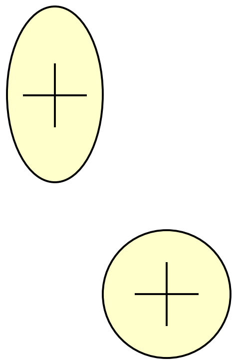
- If clusters are normally distributed in d dimensions, then after transformation, one standard deviation = $\sqrt{d}$
  - i.e., 68% of the points of the cluster will have a Mahalanobis distance $< \sqrt{d}$
  - Typical threshold
    < 2 standard deviations



Should 2 CS clusters be combined?

Q2) Should 2 CS subclusters 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
  - …
- Similar tradeoffs as for hierarchical clustering

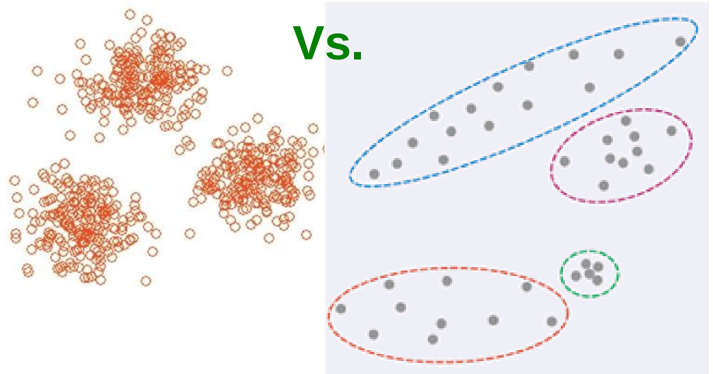
Discussion of BFR

- Old algorithm (1998), not much practical use any more
- Introduces a number of techniques and strategies
  - Incremental evaluation in memory-sized chunks
  - Online algorithm for preliminary results
  - Microclusters with count/sum/squared sum summaries
- Works well if assumptions are held
- Assumptions not always realistic, without
  - Axis alignment of clusters: summaries don't work
  - Normal distribution: too many outliers (RS), high memory consumption
  - Stable distribution: suboptimal clustering — too much shift -> reassignment -> not possible to remove
- Initialization/initial sampling weak
- Shape/structure assumption: CURE
- Unstable distributions/drift: Stream clustering (maybe later)

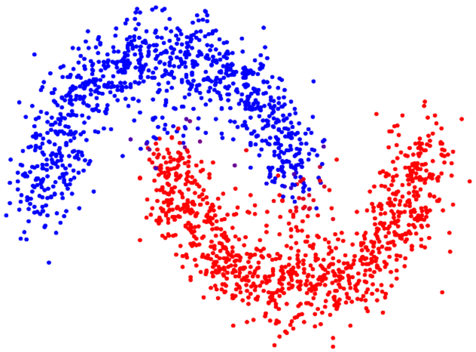
The CURE Algorithm

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

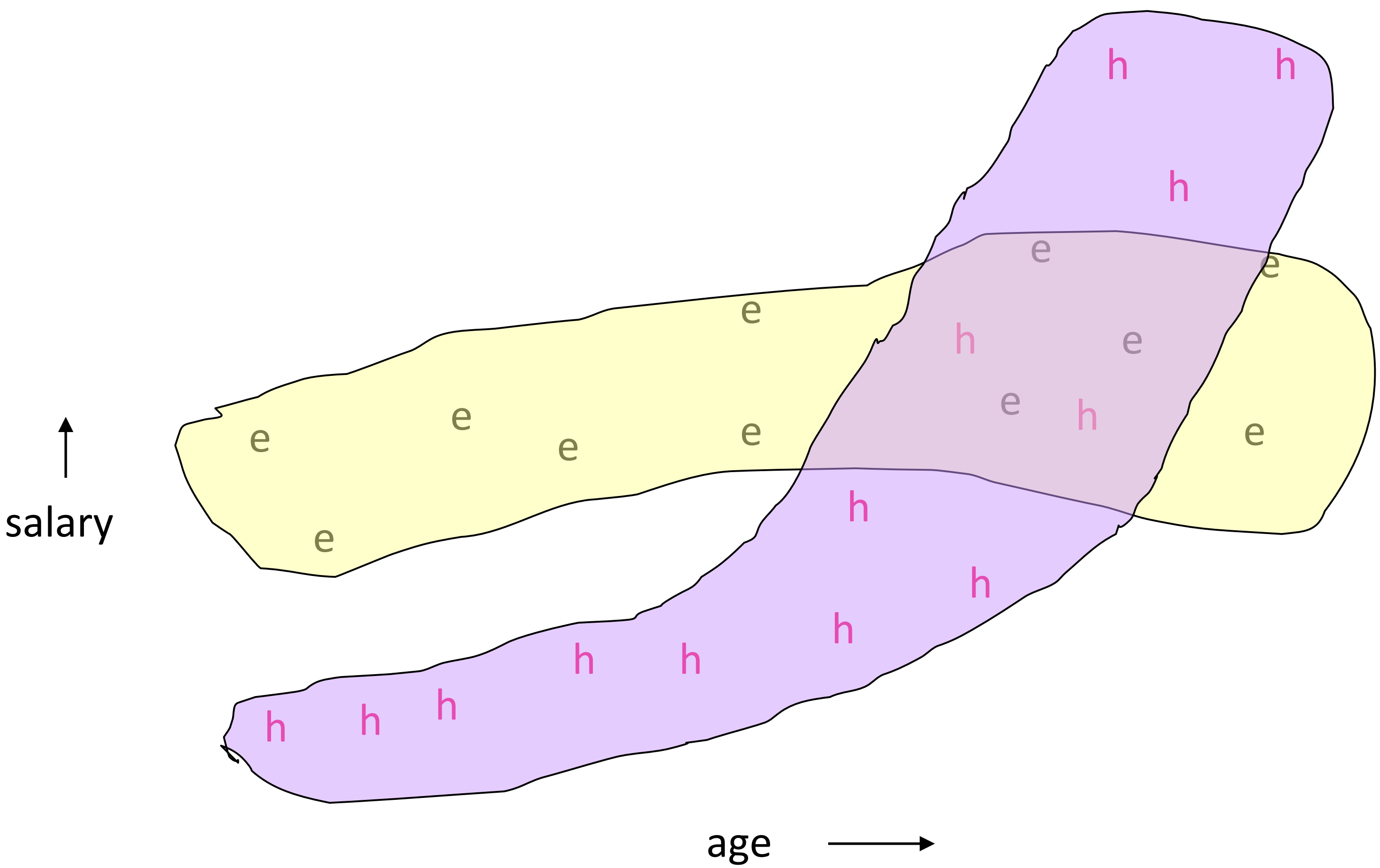
The CURE Algorithm



- **Problem with BFR/*k*-means:**
  - Assumes clusters are normally distributed in each dimension
  - And axes are fixed – ellipses at an angle are ***not OK***
- **CURE (Clustering Using REpresentatives):**
  - Assumes a Euclidean distance
  - Allows clusters to assume any shape
  - **Uses a collection of representative points to represent clusters instead of**



Example: Stanford Salaries



Starting CURE

2 Pass algorithm. Pass 1:

0) Pick a random sample of points that fit in main memory

1) Initial clusters:

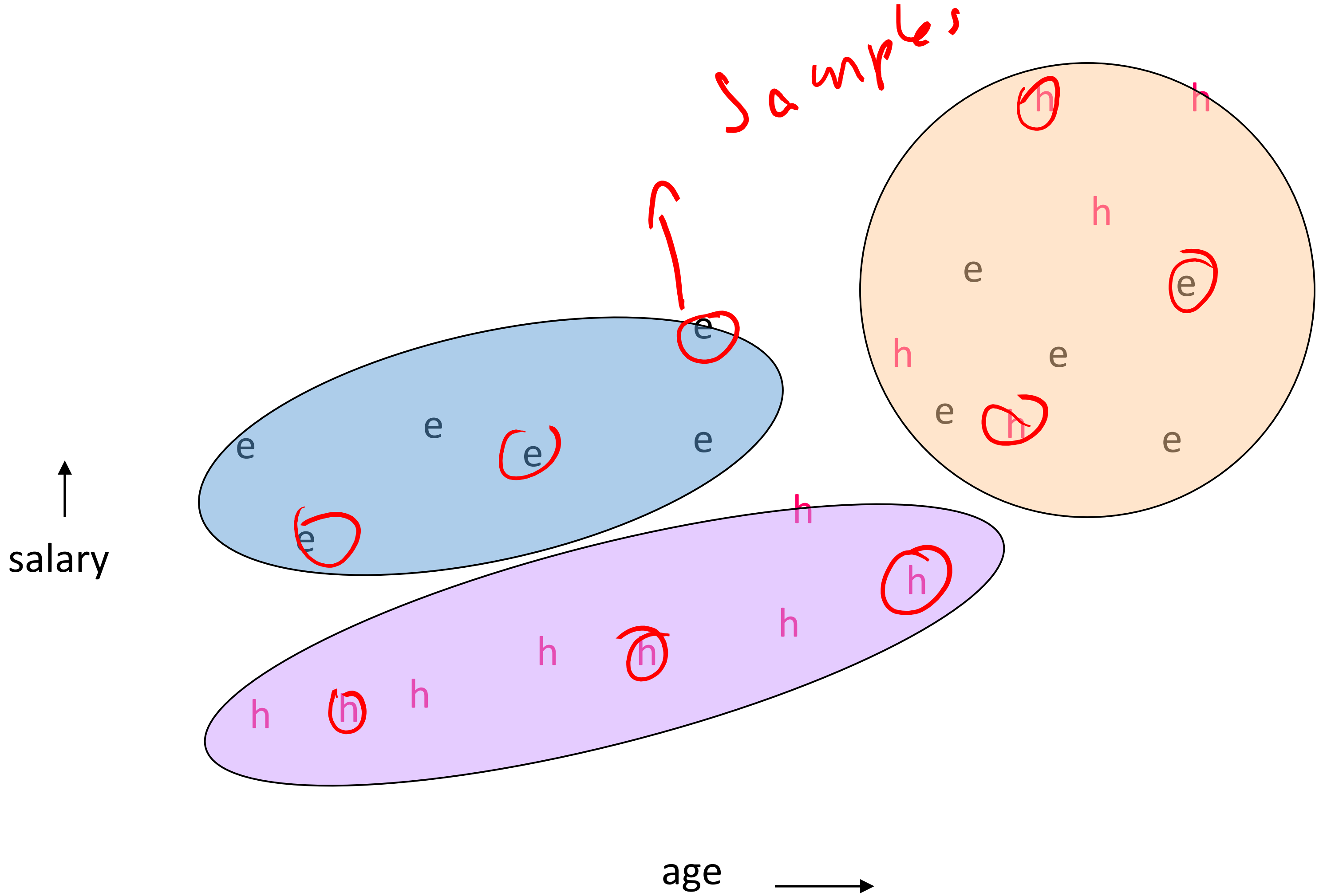
- Cluster these points hierarchically – group nearest points/clusters

2) Pick representative points:

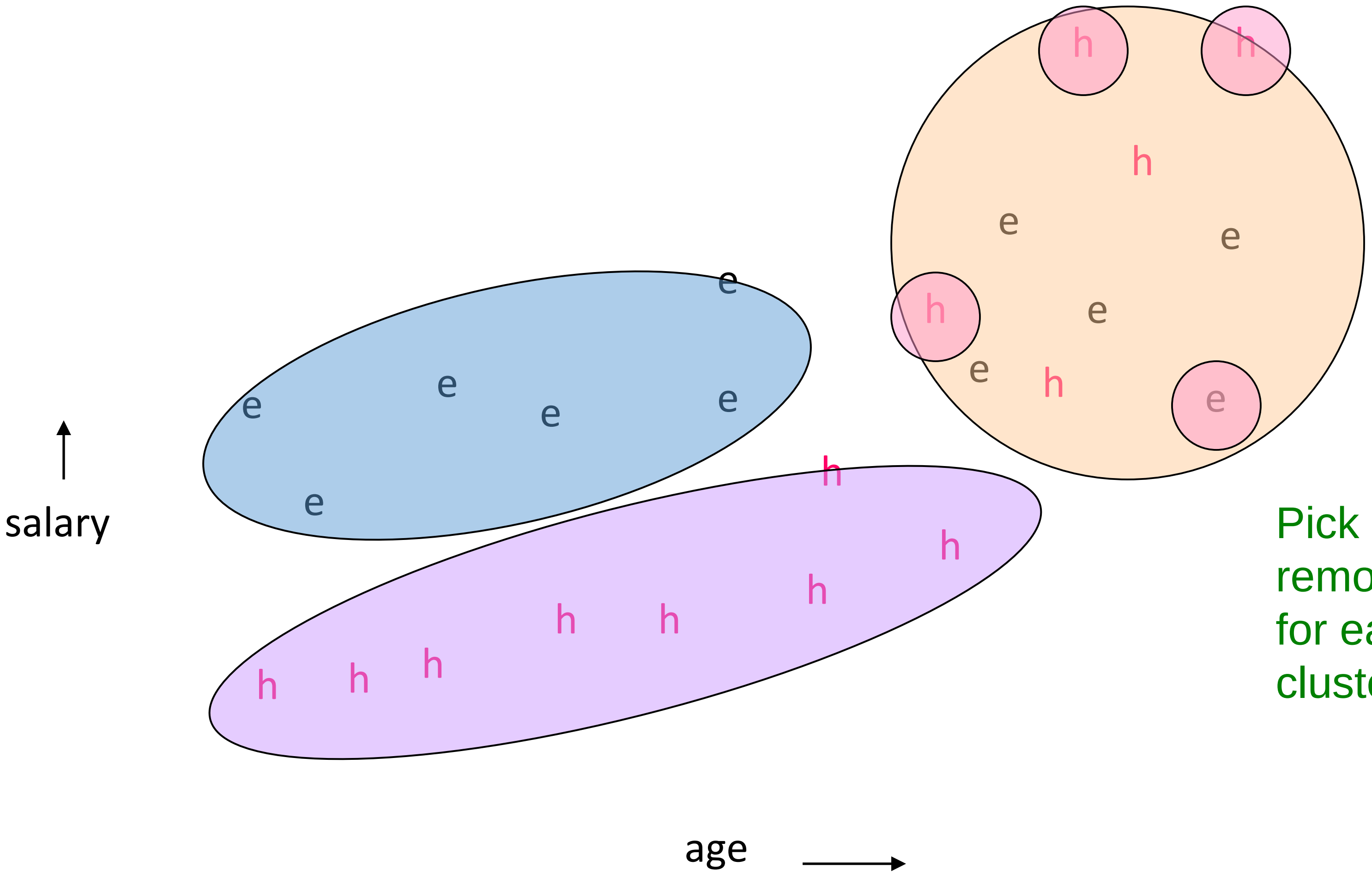
- For each cluster, pick a sample of points, as dispersed as possible → Euclidean-distance
- From the sample, pick representatives by moving them (say) 20% toward the centroid of the cluster → deal with outliers

3) Merge cluster if representatives sufficiently close

Example: Initial Clusters

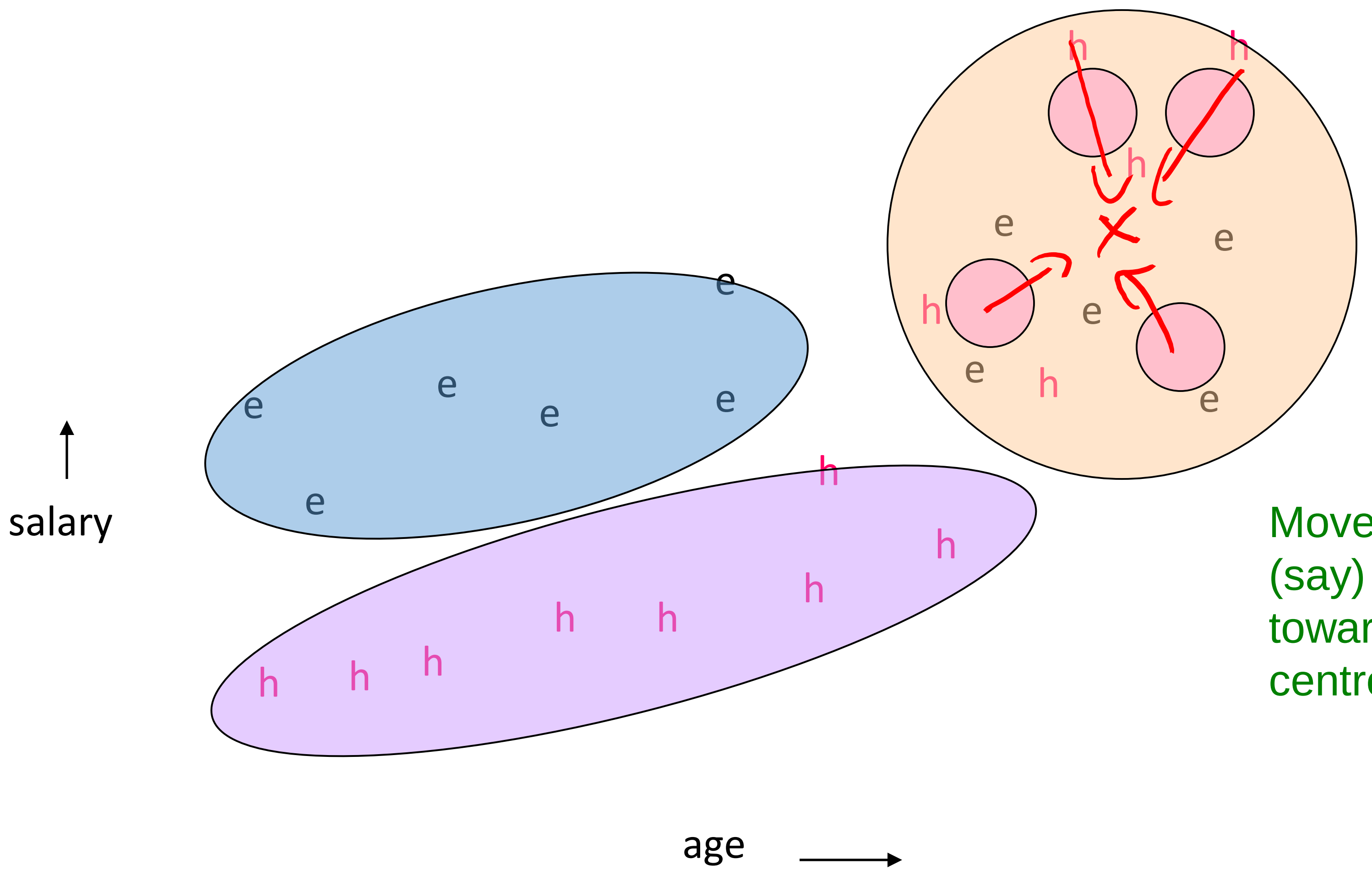


Example: Pick Dispersed Points



Pick (say) 4 remote points for each cluster.

Example: Pick Dispersed Points

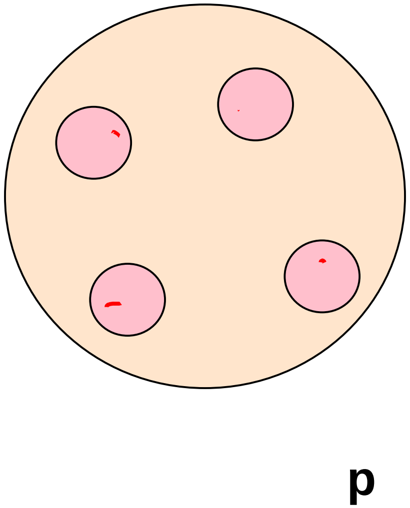


Move points (say) 20% toward the centroid.

Finishing CURE

Pass 2:

- 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 to ***p*** and assign it to representative's 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 + group distance metrics
  - ***k*-means:**
    - Initialization, picking *k*
  - **BFR:**
    - Scaling to very large datasets by summarizing
  - **CURE**
    - Supporting irregular shapes by representatives
- **Beyond the lecture:**
  - **EM/GMM:** Iteratively fit a model like normal distribution (~algorithms similar to k-means)
  - **DBScan:** Density-based clustering with index support
  - …