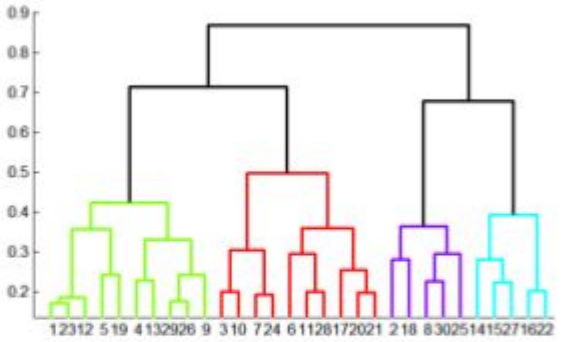
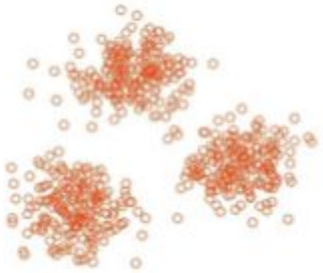
Clustering

Overview: Methods of Clustering

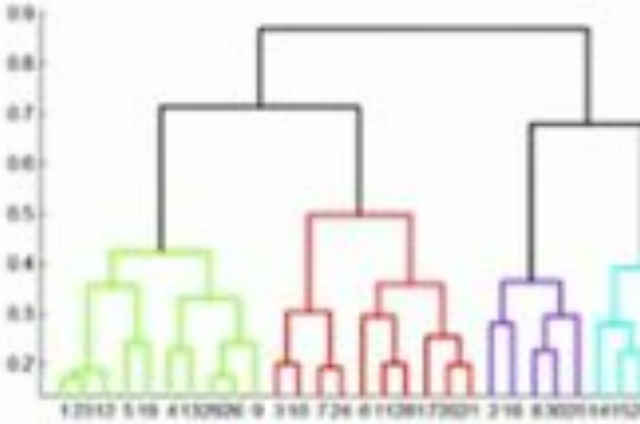
- **Hierarchical:**
  - **Agglomerative** (bottom up):
    - Initially, each point is a cluster
    - Repeatedly combine the two "nearest" clusters into one
  - **Divisive** (top down):
    - Start with one cluster and recursively split it



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



Hierarchical Clustering



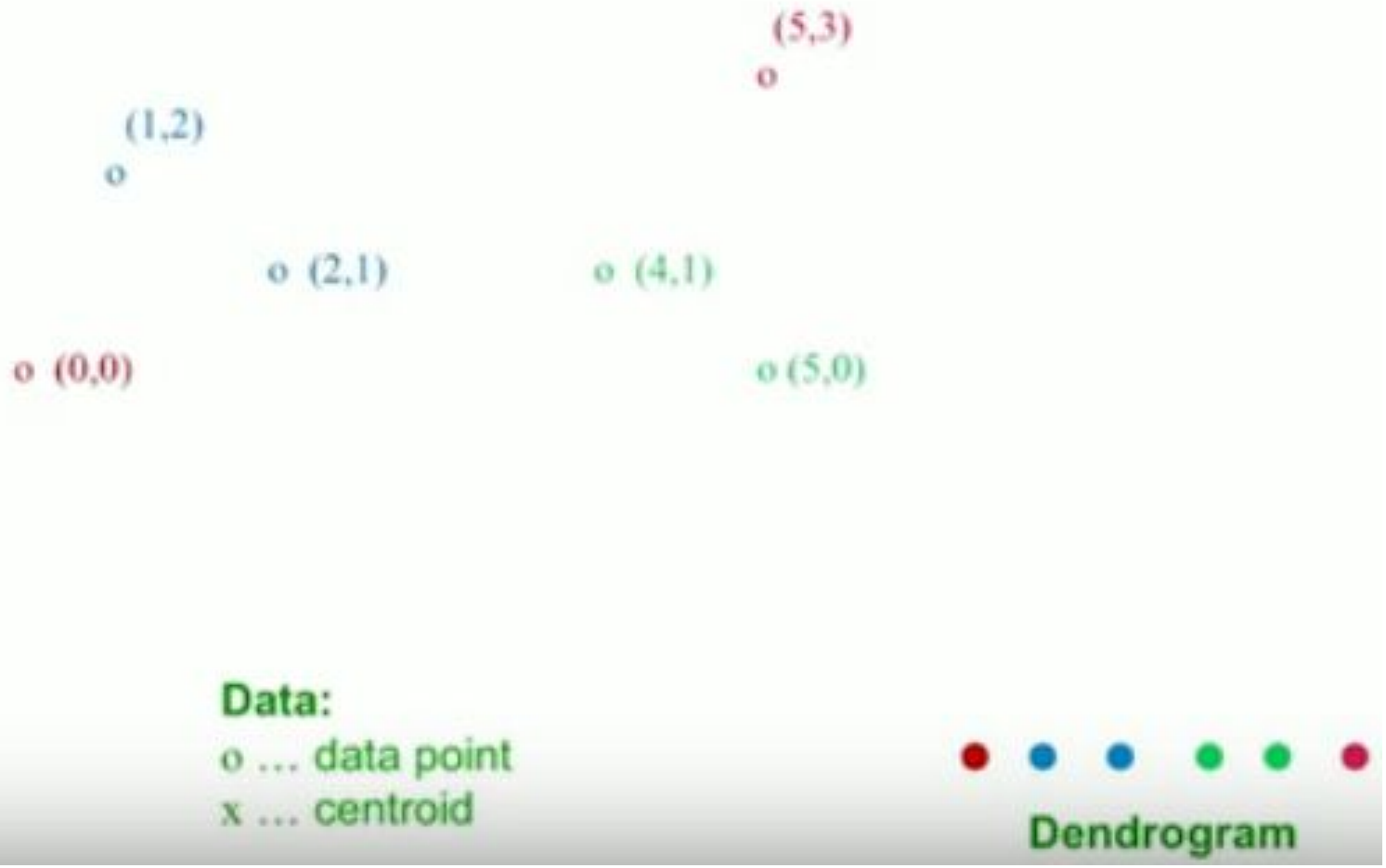
- **Key operation:**
  Repeatedly combine two nearest clusters

- **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?

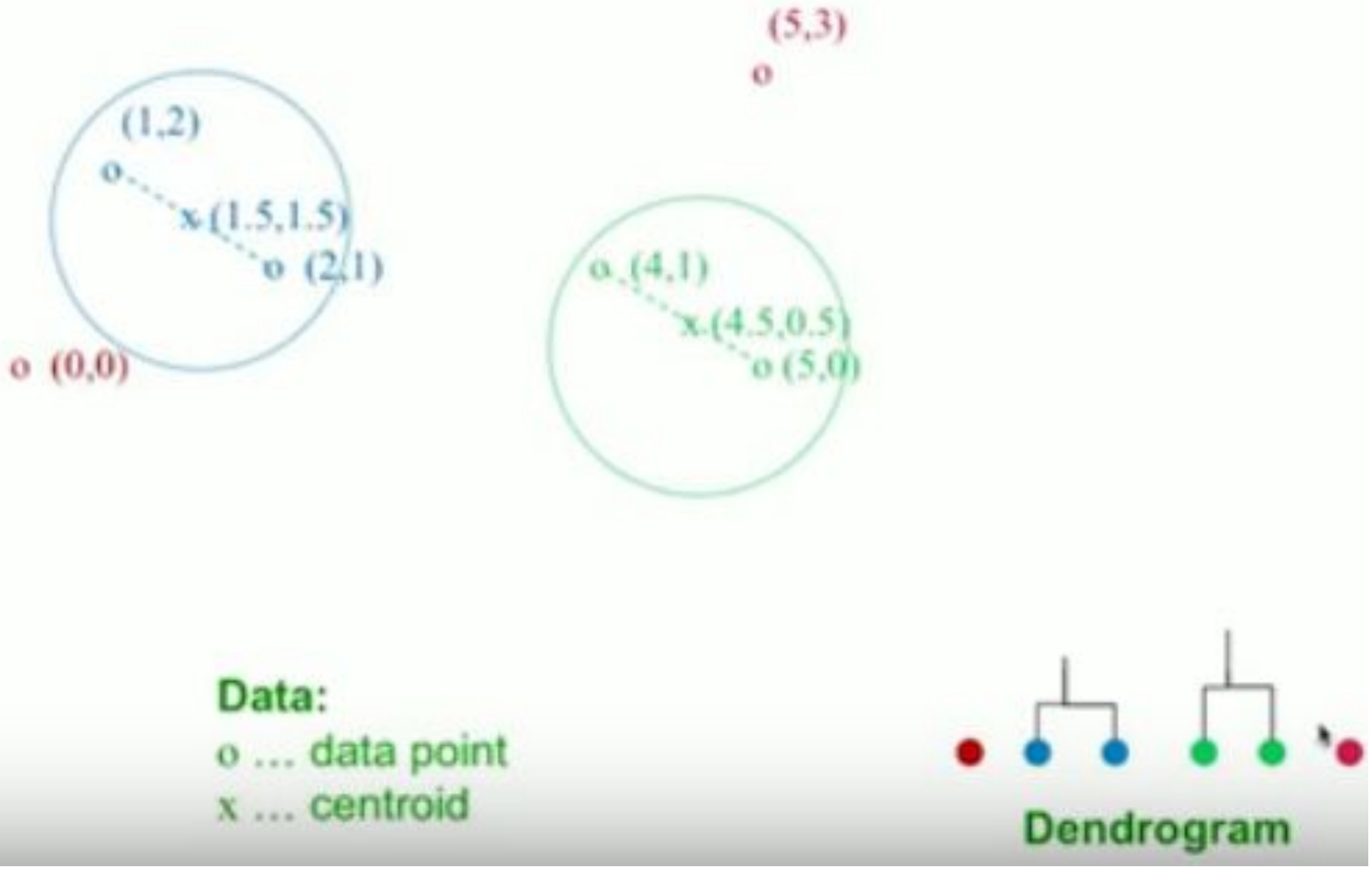
Euclidean Space

- **(1) How to represent a cluster of many points?**
  - How do you represent the location of each cluster, to tell which pair of clusters is closest?
  - Represent each cluster by its ***centroid*** = average of its points
- **(2) How to determine "nearness" of clusters?**
  - Measure cluster distances by distances of centroids

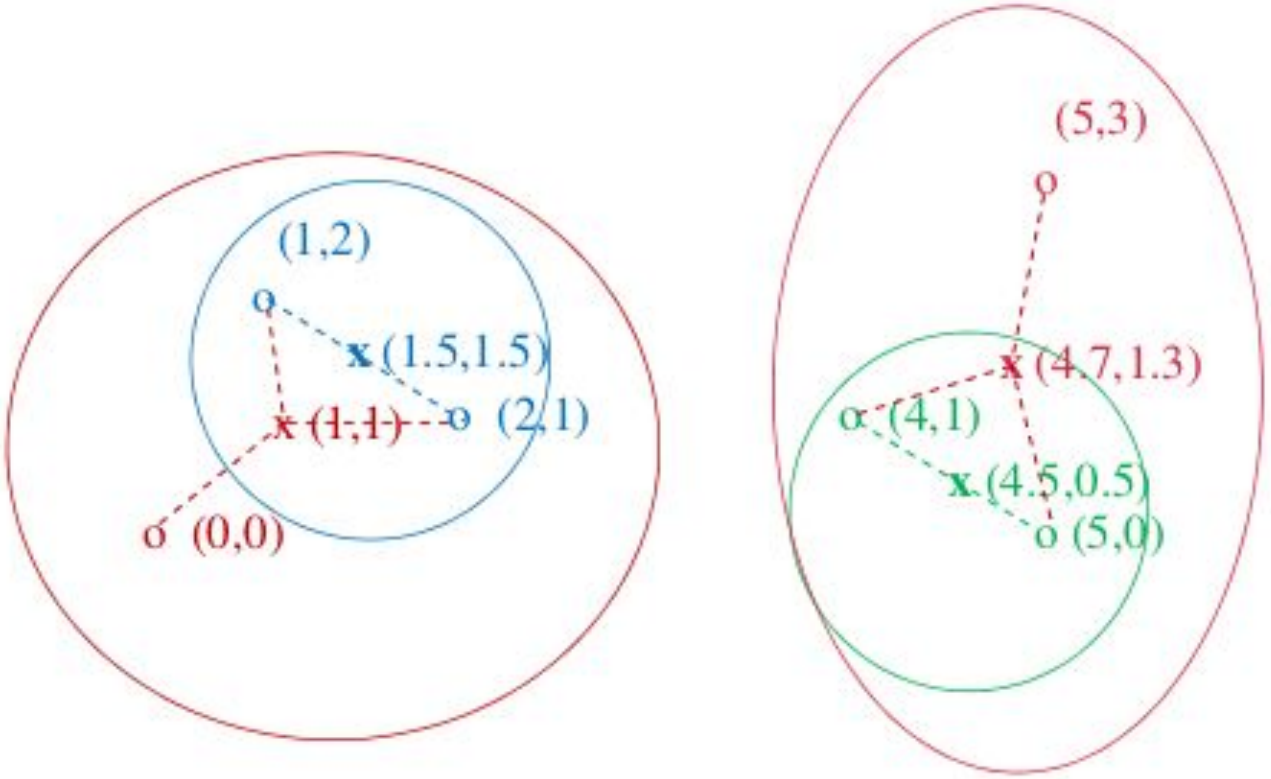
Example: Hierarchical clustering



Example: Hierarchical clustering



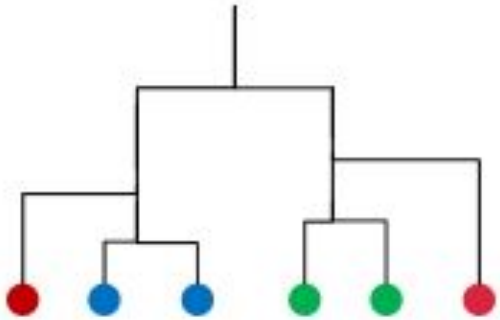
Example: Hierarchical clustering



Data:

o … data point

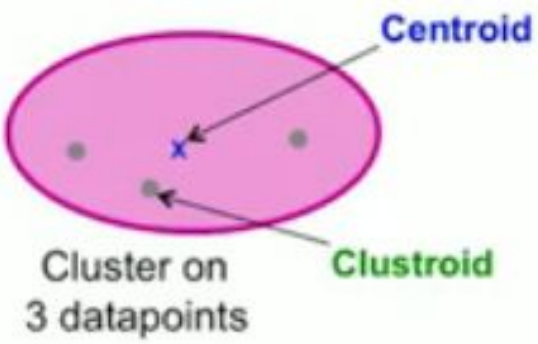
x … centroid



Dendrogram

Non-Euclidean Case

Clustroid



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.

And in the Non-Euclidean Case?

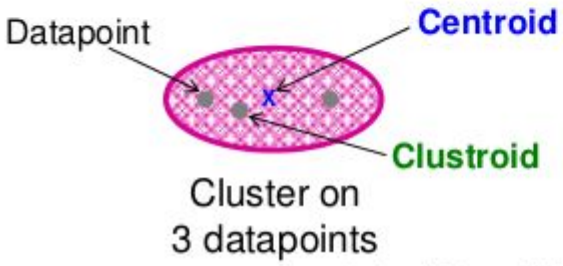
What about the Non-Euclidean case?

- The only "locations" we can talk about are the points themselves
  - i.e., there is no "average" of two points

- **Approach 1:**
  - **(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

"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
    - For distance metric ***d*** clustroid ***c*** of cluster ***C*** is: $\min_{c} \sum_{x \in C} d(x,c)^2$



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.

Termination condition

- (3) When do you stop combining clusters?
- Approach 1: Pick a number *k* upfront, and stop when we have *k* clusters
  - Makes sense when we know that the data naturally falls into *k* classes
- Approach 2: Stop when the next merge would create a cluster with low "cohesion"
  - i.e, a "bad" cluster

Merge clusters whose *union* is most cohesive

Cohesion

- **Approach 3.1:** Use the **diameter** of the merged cluster = maximum distance between points in the cluster
- **Approach 3.2:** Use the **average distance** between points in the cluster
- **Approach 3.3:** Use a **density-based approach**
  - Take the diameter or avg. distance, e.g., and divide by the number of points in the cluster

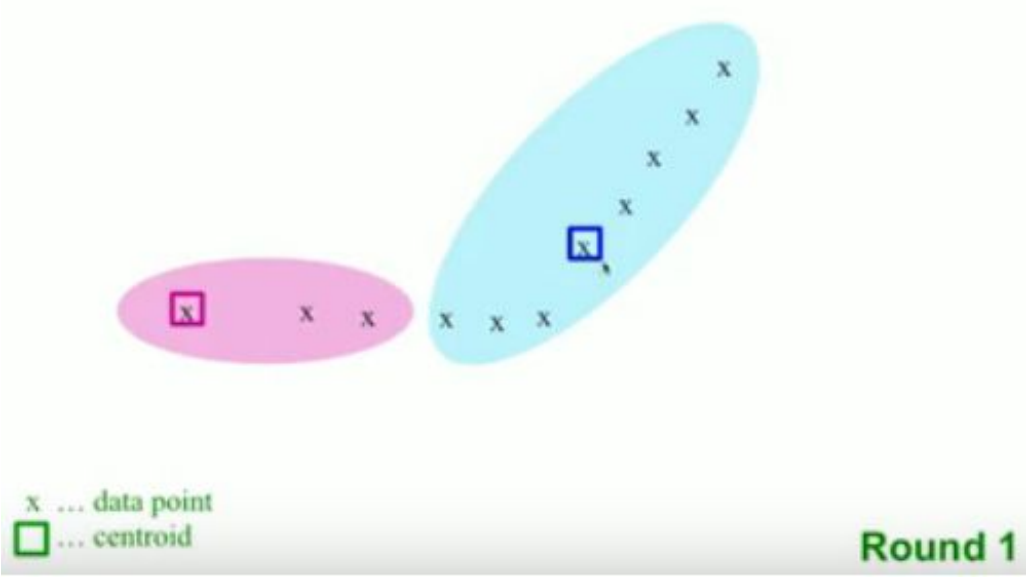
Implementation

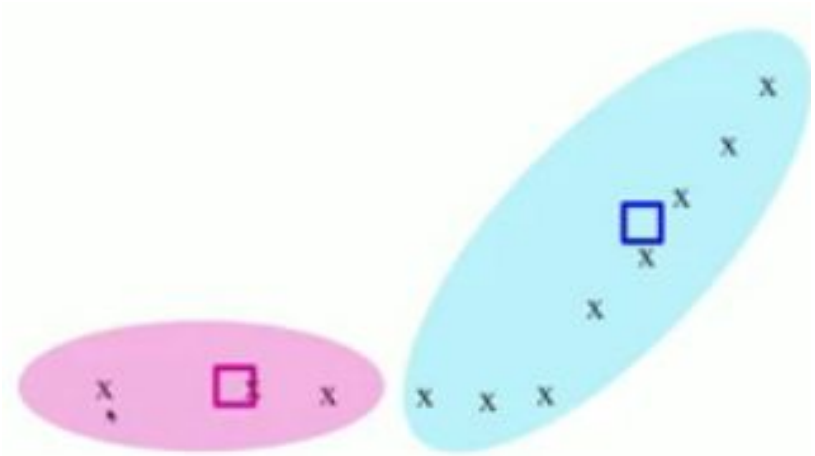
- **Naïve implementation of hierarchical clustering:**
  - At each step, compute pairwise distances between all pairs of clusters, then merge
  - $O(N^3)$
  - **too expensive for really big datasets that do not fit in memory**

k–means Algorithm(s)

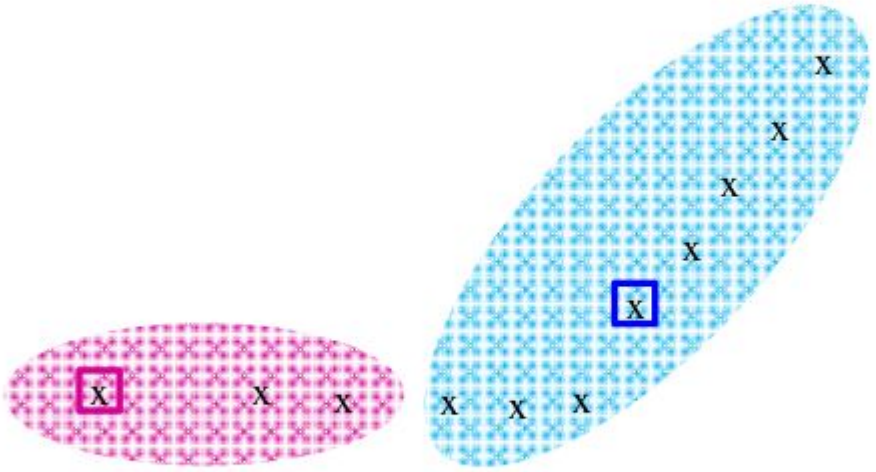
- Assumes Euclidean space/distance
- Start by picking ***k***, the number of clusters
- Initialize clusters by picking one point per cluster
- **1)** For each point, place it in the cluster whose current centroid it is nearest
- **2)** After all points are assigned, update the locations of centroids 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: $k = 2$





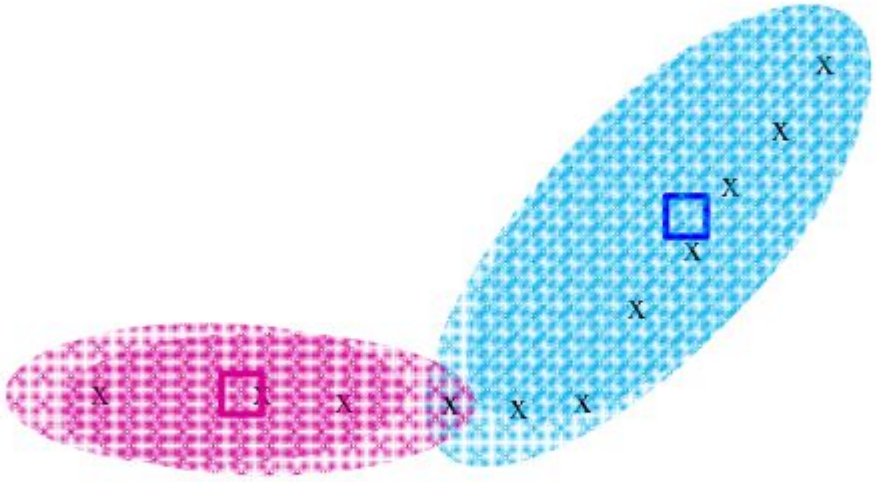
Example: Assigning Clusters



x … data point
□ … centroid

Clusters after round 1

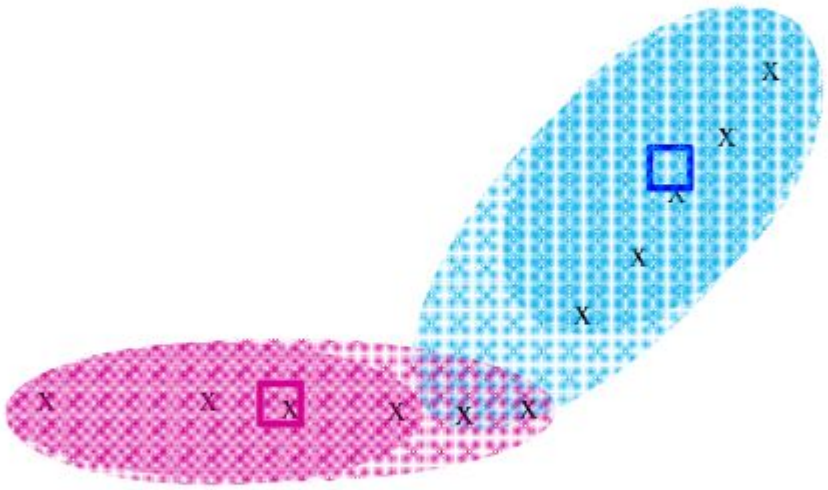
Example: Assigning Clusters



x … data point
□ … centroid

Clusters after round 2

Example: Assigning Clusters



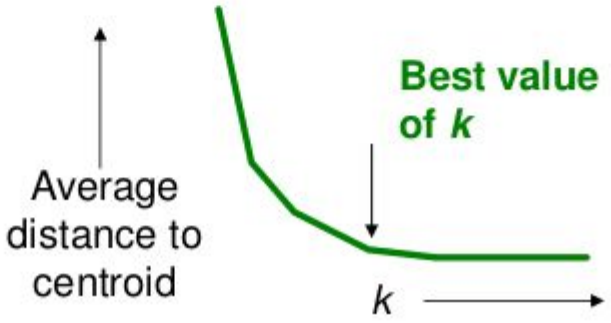
x … data point
□ … centroid

Clusters at the end

Getting the *k* right

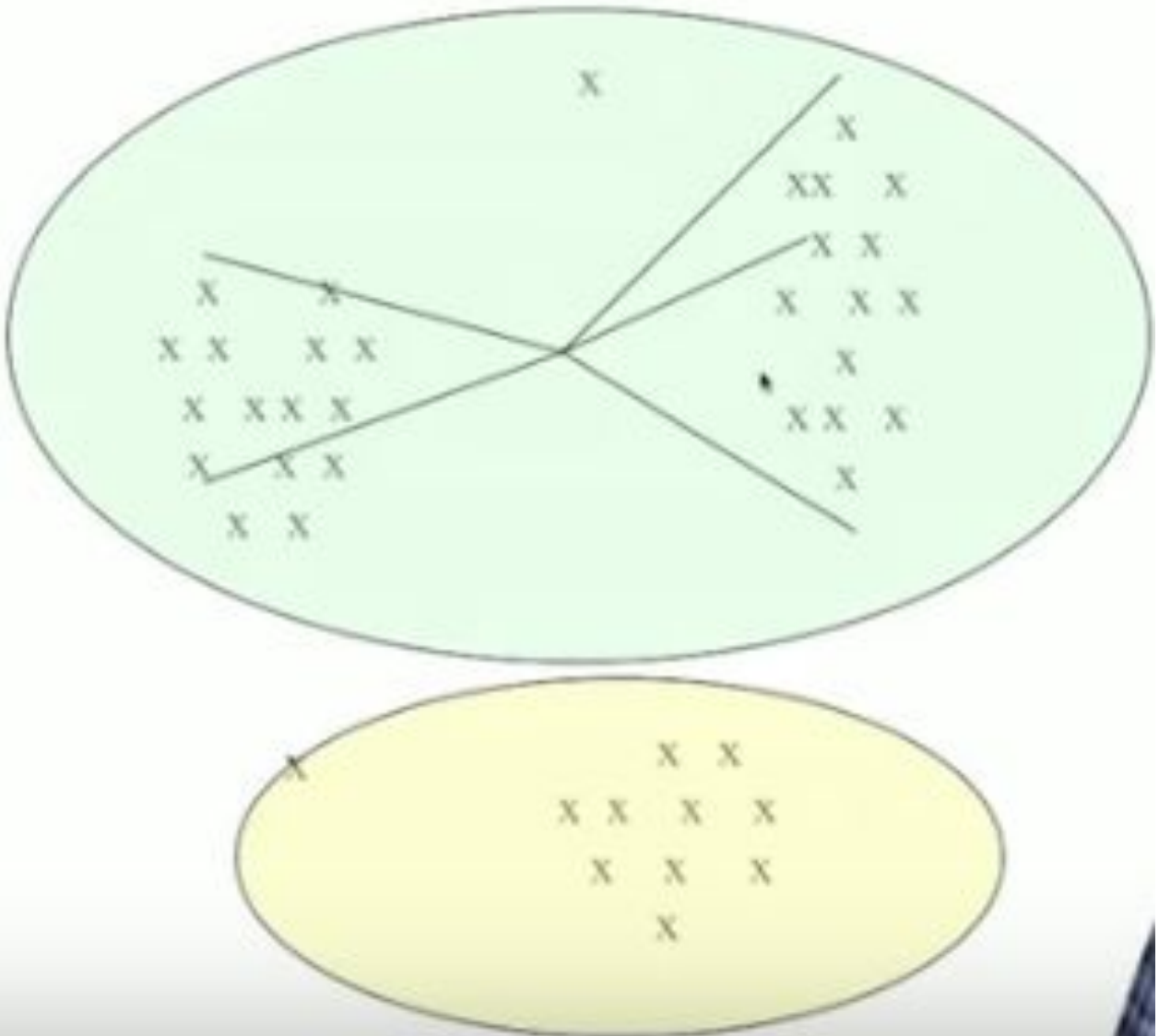
How to select *k*?

- 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



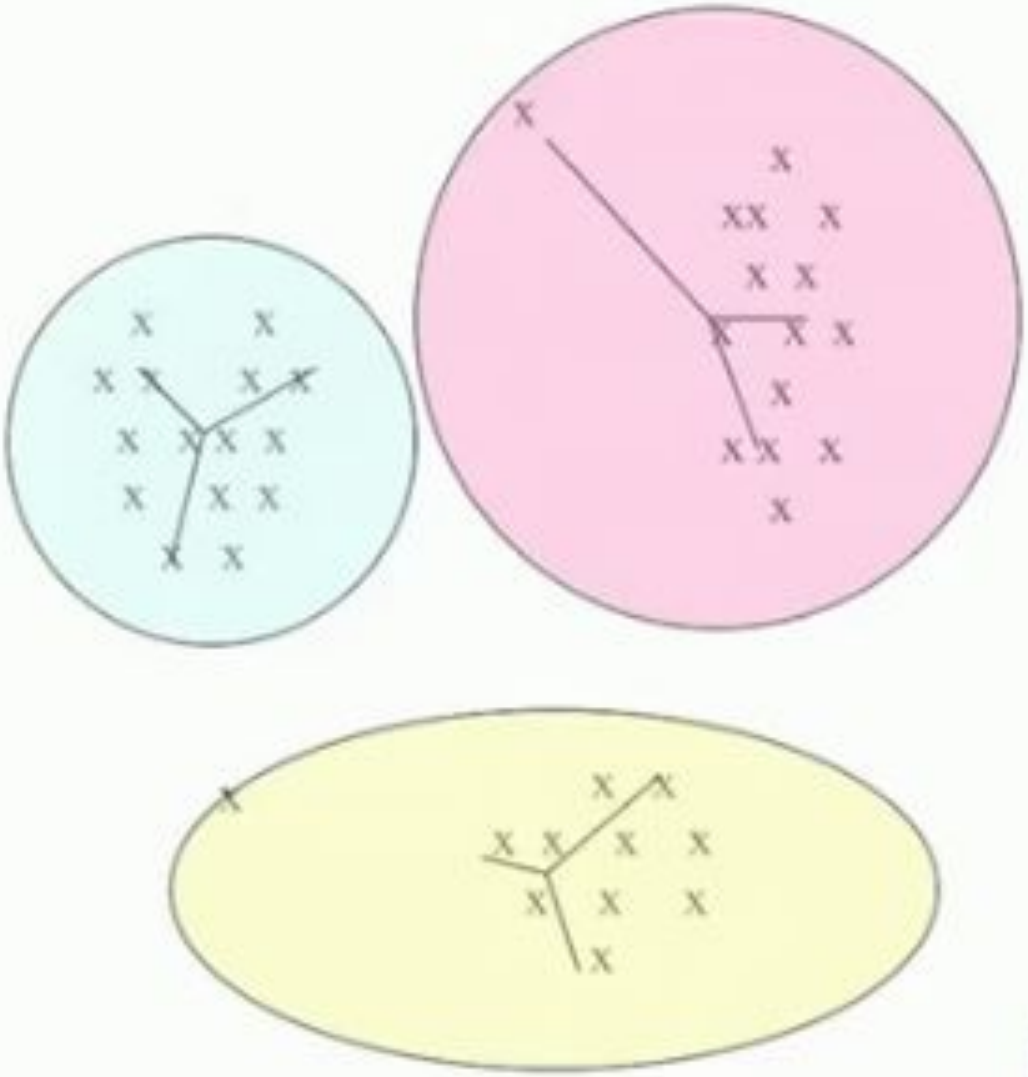
Example: Picking *k*

Too few;
many long
distances
to centroid.



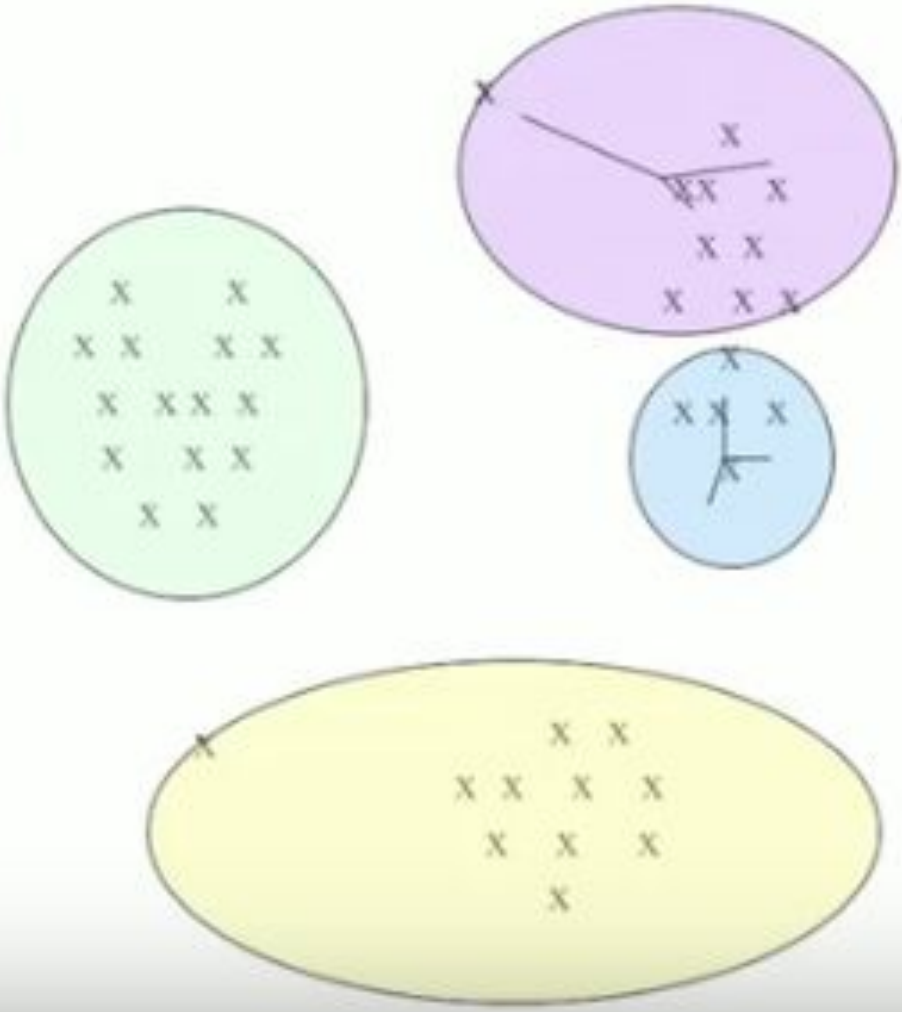
Example: Picking *k*

Just right;
distances
rather short.



Example: Picking *k*

Too many;
little improvement
in average
distance.



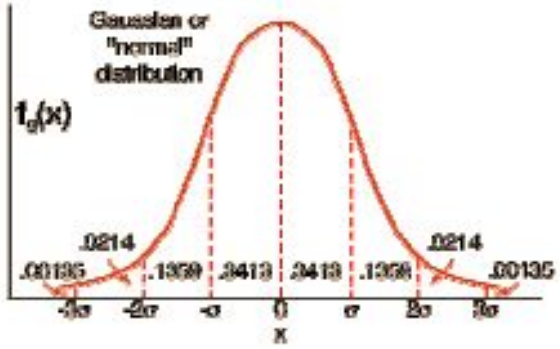
Picking the initial *k* points

- Approach 1: Sampling
  - Cluster a sample of the data using hierarchical clustering, to obtain k clusters
  - Pick a point from each cluster (e.g. point closest to centroid)
  - Sample fits in main memory
- Approach 2: Pick "dispersed" set of points
  - Pick first point at random
  - Pick the next point to be the one whose minimum distance from the selected points is as large as possible
  - Repeat until we have *k* points

The BFR Algorithm

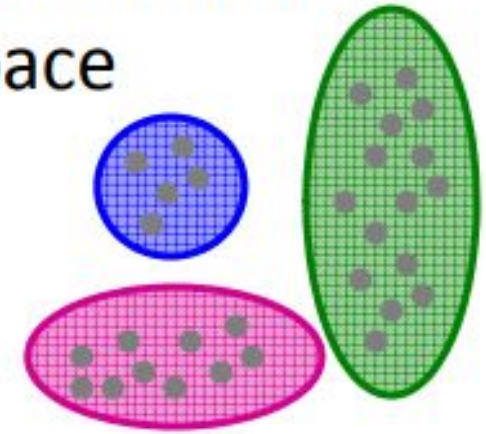
Extension of *k*-means to large data

BFR Algorithm

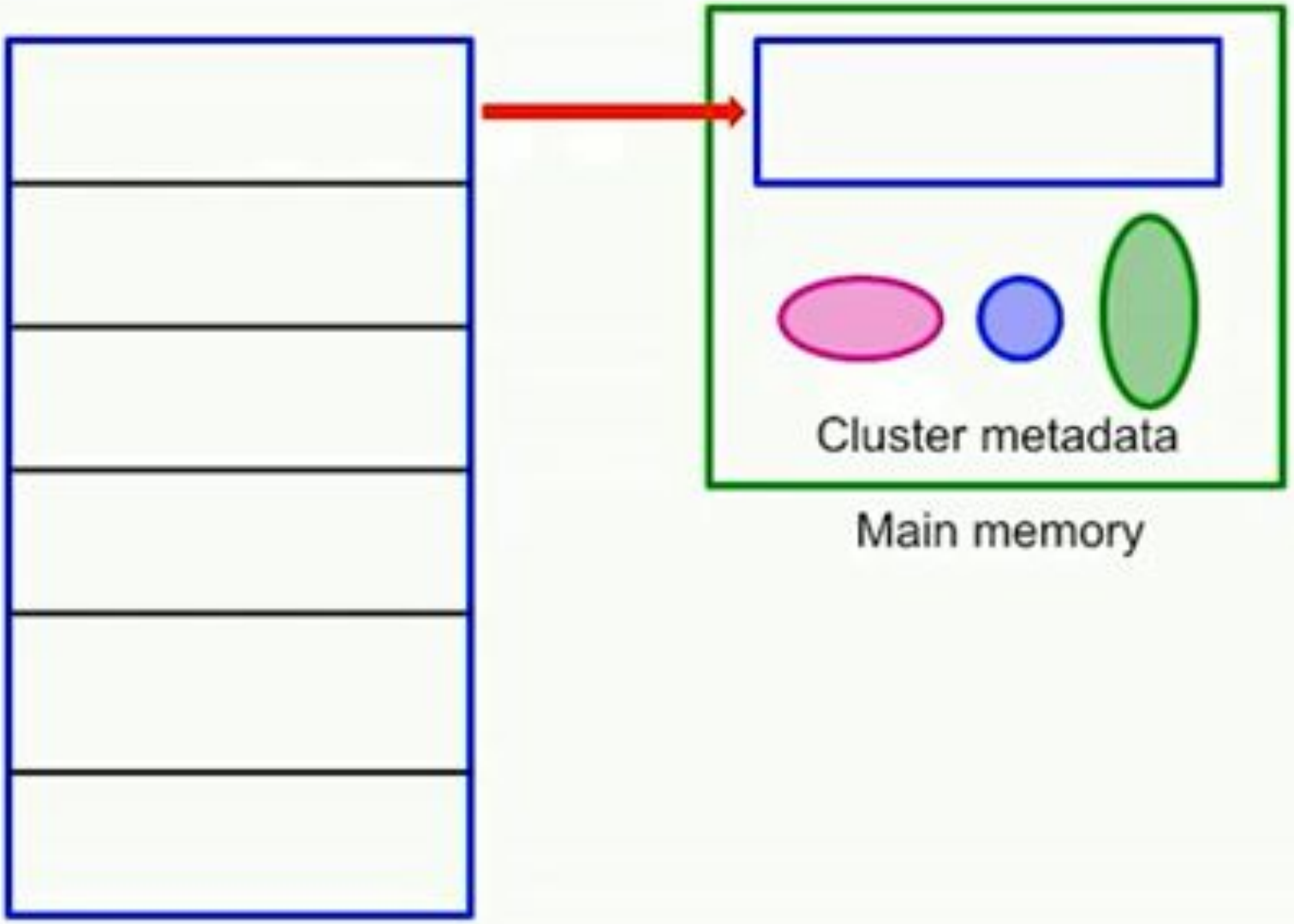


- **BFR** [Bradley-Fayyad-Reina] is a variant of *k*-means designed to handle **very large** (disk-resident) data sets

- **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



BFR Algorithm: Overview



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**
- To begin, from the initial load we select the initial ***k*** centroids by some sensible approach
  - Using one of the techniques from the k-Means

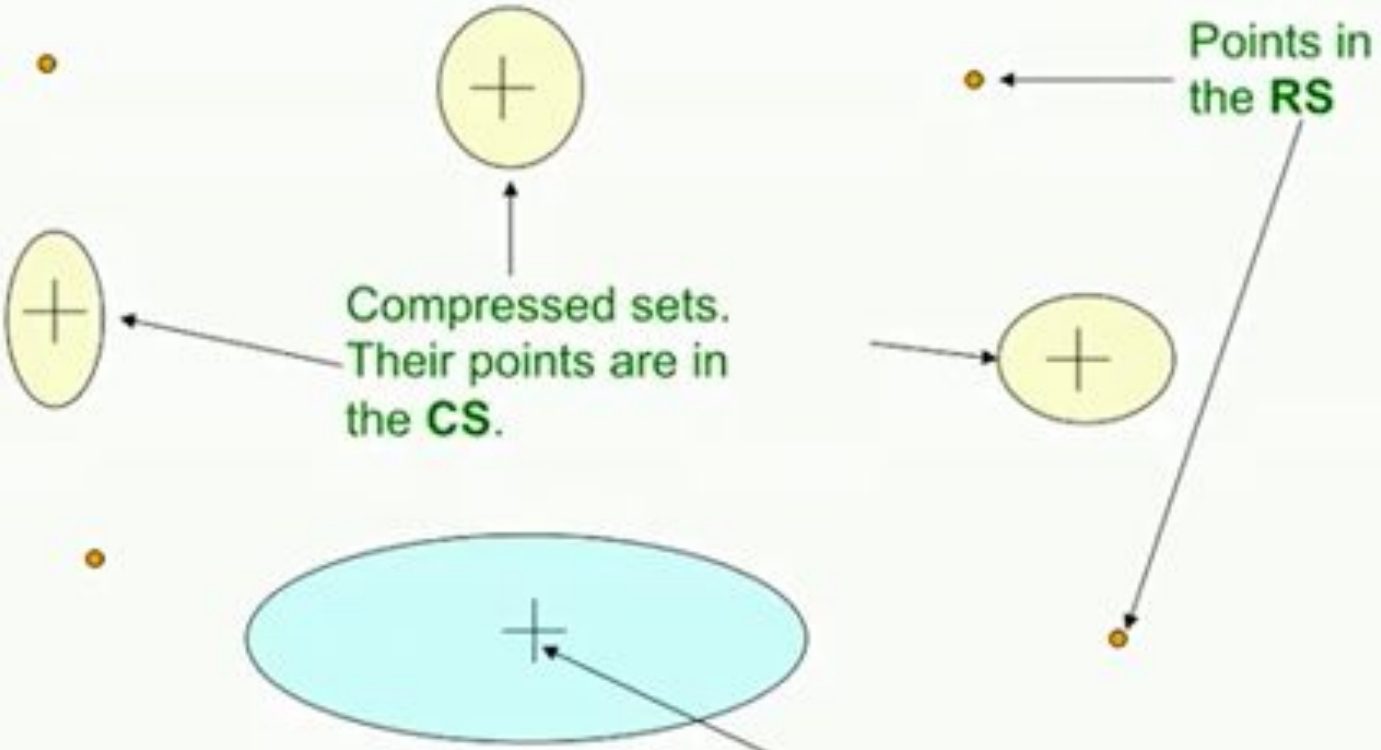
Three Classes of Points

3 sets of points which we keep track of:

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

Both DS and CS points can be discarded only points in RS needs to be stored in main memory. When a point belongs to DS or CS it's metadata is updated and point is discarded.

BFR: "Galaxies" Picture



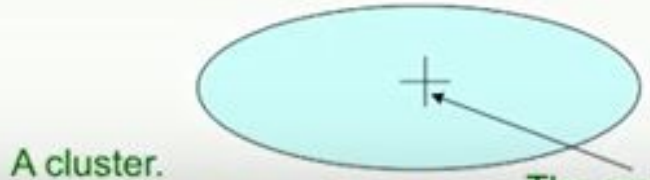
Discard set (DS): Close enough to a centroid to be summarized
Compression set (CS): Summarized, but not assigned to a cluster
Retained set (RS): Isolated points

Meta data

Summarizing Sets of Points

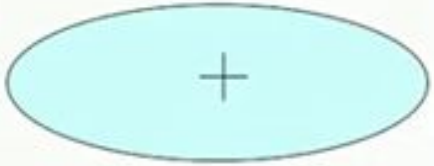
For each cluster, the discard set (DS) is summarized by:

- The number of points, ***N***
- The vector ***SUM***, whose i^{th} component = 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**



SUM and SUMSQ are needed for calculating centroid and deviation

Processing a chunk of points (1)

- Find those points that are "**sufficiently close**" to a cluster centroid
- Add those points to that cluster and the **DS**
  - Then discard the point
- **DS set:** Adjust statistics of each cluster to account for newly added points
  - Add ***N***s, ***SUM***s, ***SUMSQ***s

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 a chunk of points (2)

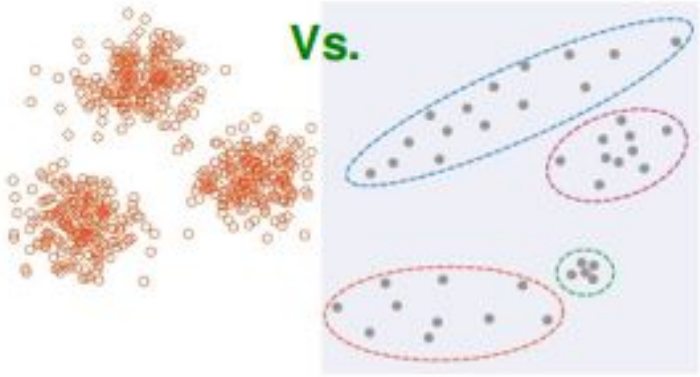
- The remaining points are not close to any cluster
- Use any main-memory clustering algorithm to cluster these points and the old **RS**
  - Clusters go to the **CS**; outlying points to the **RS**
- Consider merging compressed sets in the **CS**
- If this is the last round, merge all compressed sets in the **CS** and all **RS** points into their nearest cluster

CURE algorithm

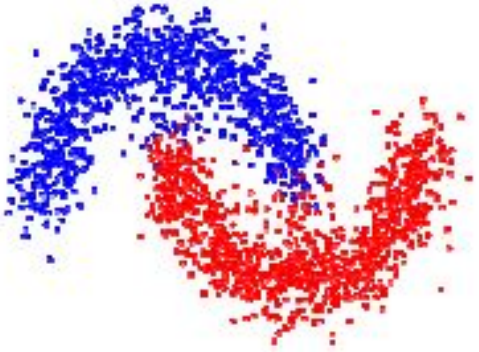
Clustering using REpresentative

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**



Starting CURE

2 Pass algorithm. Pass 1:

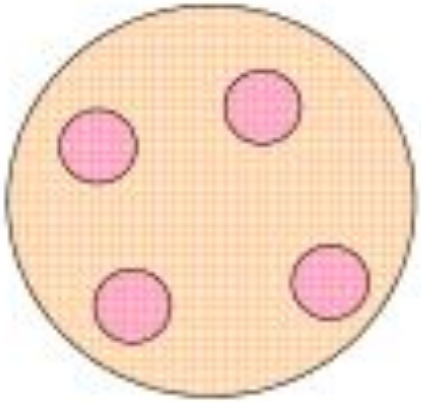
- **0) Pick a random sample of points that fit in main memory**
- **1) Initial clusters:**
  - Cluster these points hierarchically – group nearest points/clusters
- **2) Pick representative points:**
  - For each cluster, pick a sample of points, as dispersed as possible
  - From the sample, pick representatives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

- For each cluster, c well scattered points within the cluster are chosen, and then shrinking them toward the mean of the cluster by a fraction α
- The distance between two clusters is then the distance between the closest pair of representative points from each cluster.
- The c representative points attempt to capture the physical shape and geometry of the cluster.
- Shrinking the scattered points toward the mean gets rid of surface abnormalities and decrease the effects of outliers.

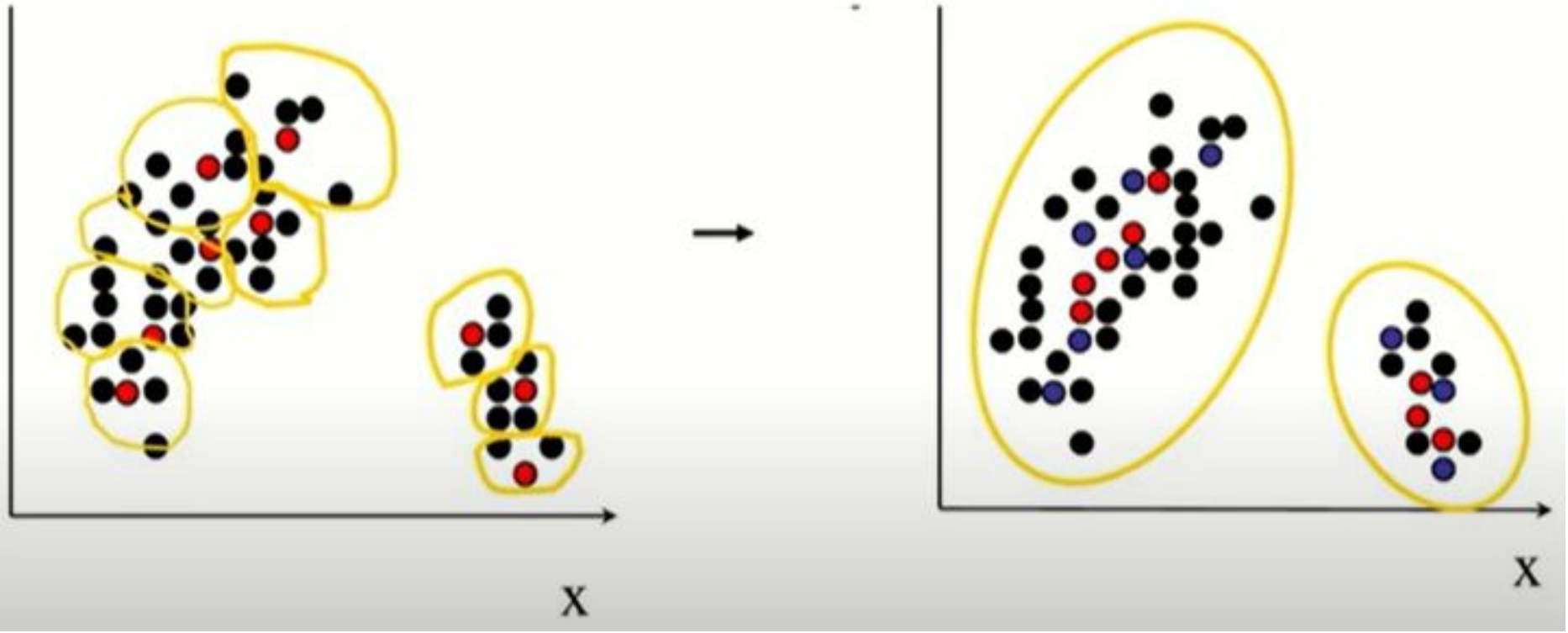
Finishing CURE

Pass 2:

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

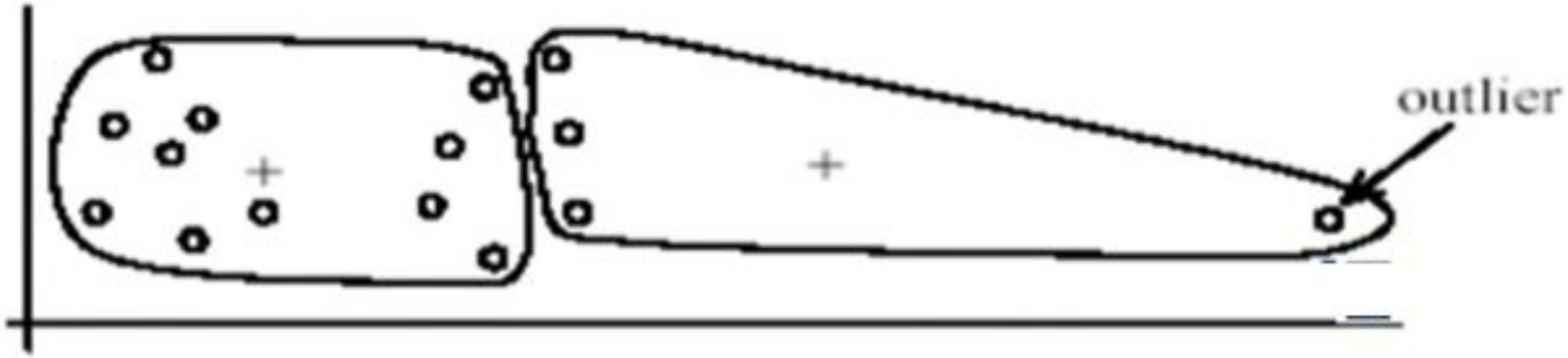


- **Place it in the "closest cluster"**
  - Normal definition of "closest":
    Find the closest representative to ***p*** and assign it to representative's cluster


X
X

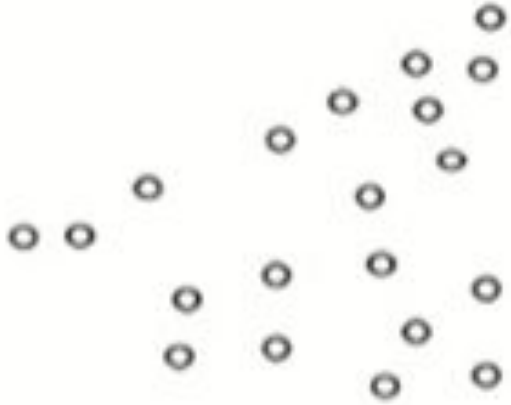
Handlining Outlier

Handling Outlier

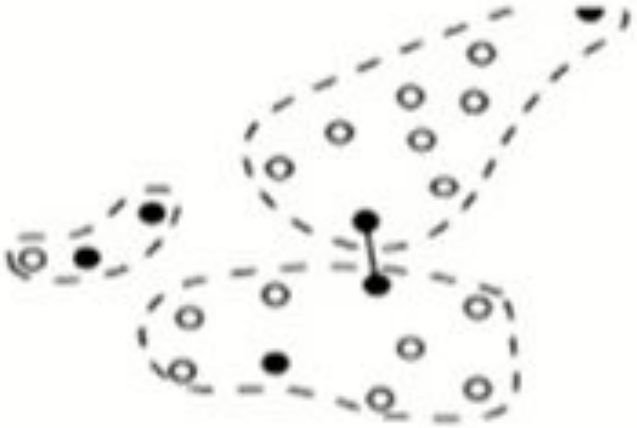


(A): Undesirable clusters

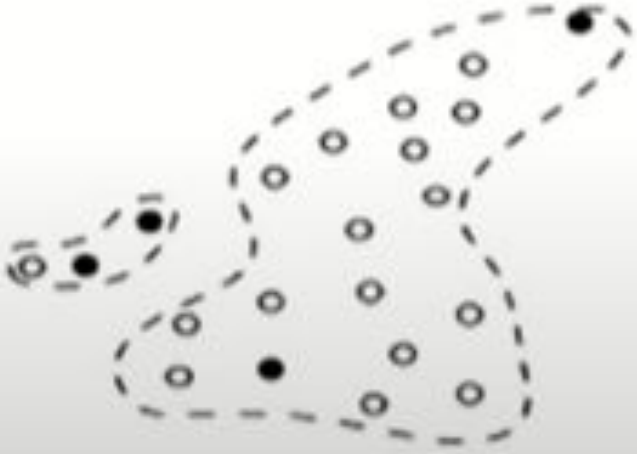
(B): Ideal clusters



a) Sample of Data



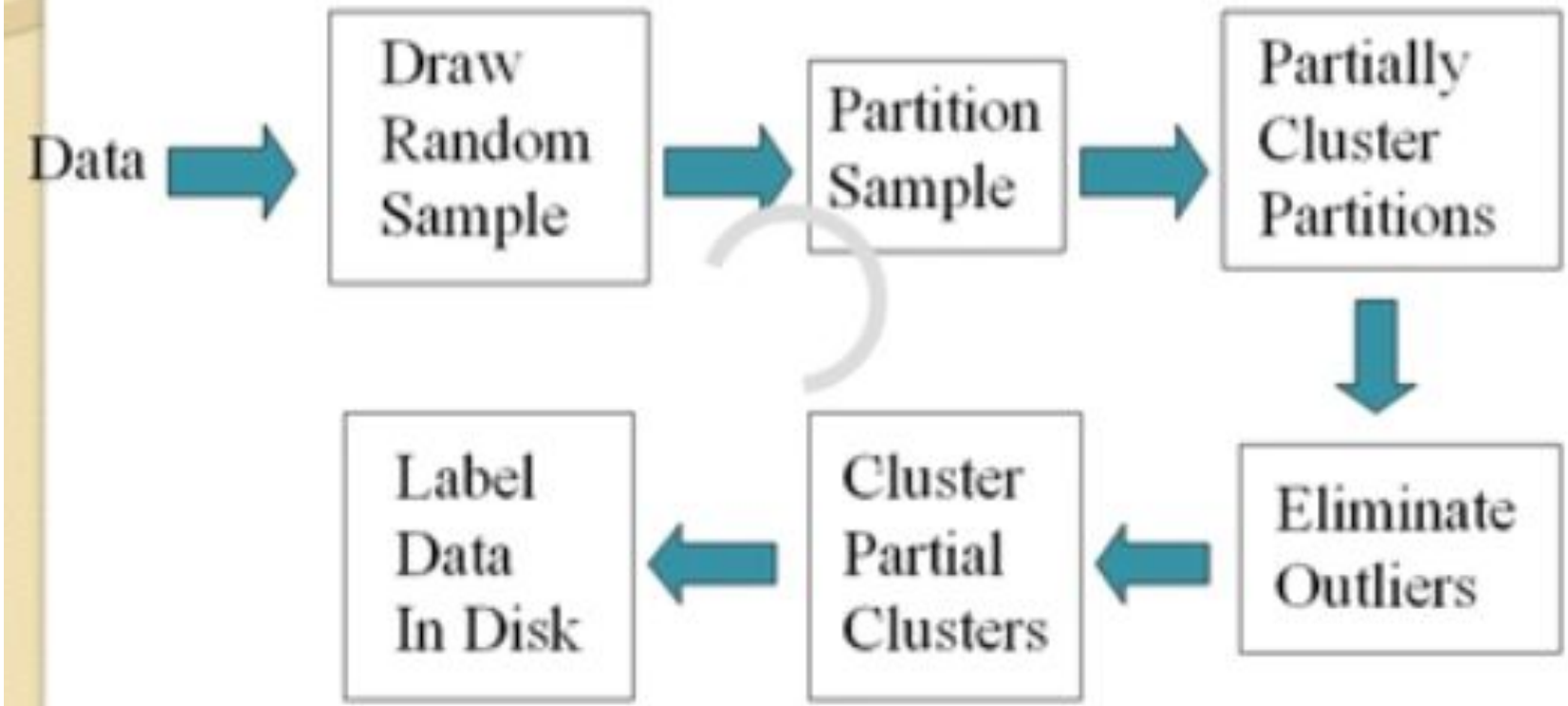
b) Three Clusters with Representative Points



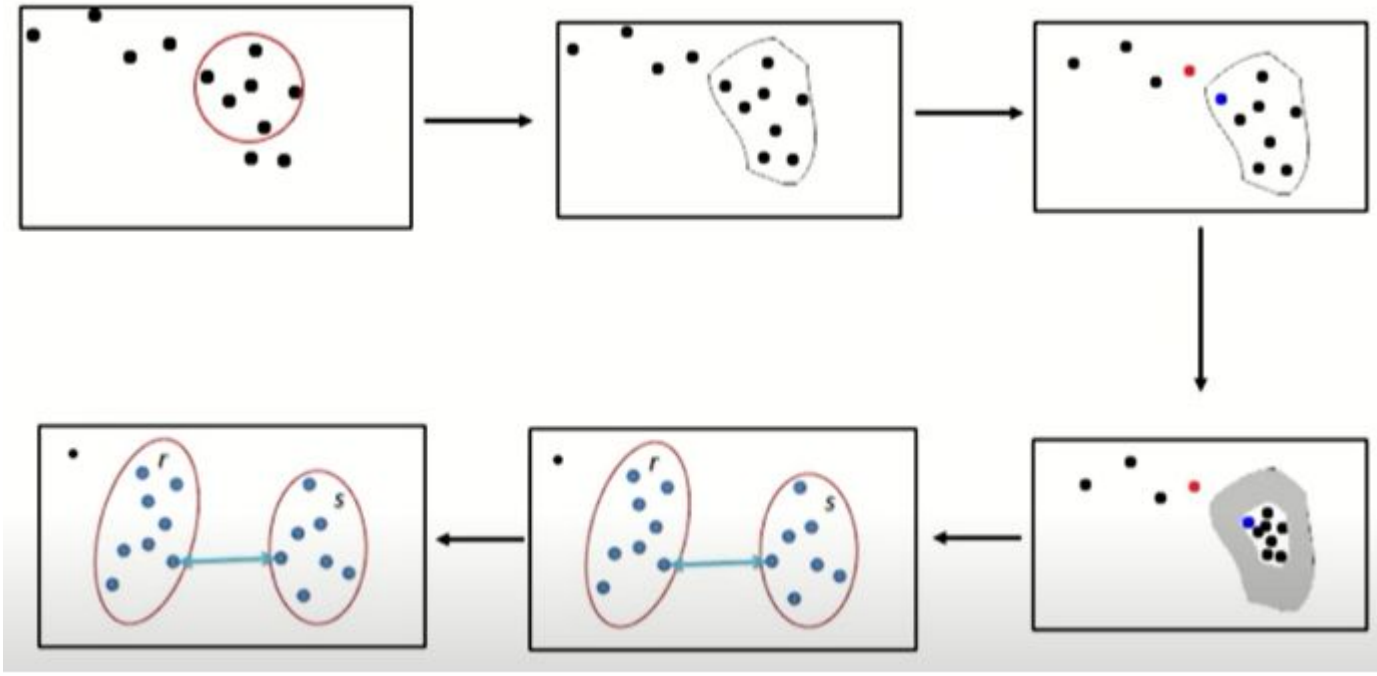
Summary

- **CURE** can detect clusters of non spherical shape, with variation of size with the representative points for each cluster.
- **G**ood execution time with large database and sets using random sampling and partition methods.
- **W**orks well with outliers, which are detected and merged or eliminated.

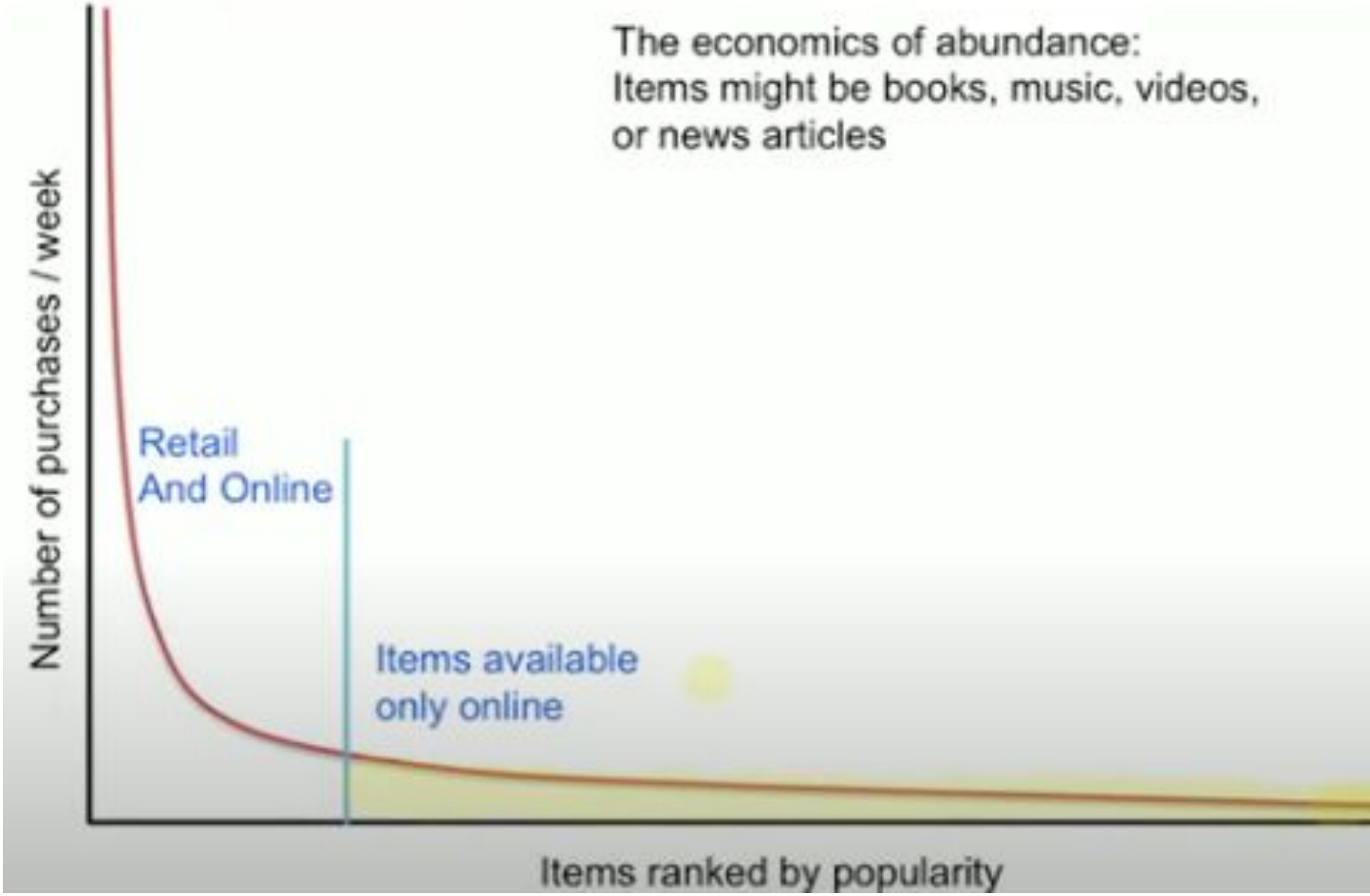
Six Steps in CURE Algorithm



Six Steps in CURE Algorithm



Recommendation Systems


The economics of abundance:
Items might be books, music, videos,
or news articles
Number of purchases / week
Retail
And Online
Items available
only online
Items ranked by popularity

Formal Model

- ***X*** = set of **Customers**
- ***S*** = set of **Items**

- **Utility function** ***u***: $X \times S \rightarrow R$
  - ***R*** = set of ratings
  - ***R*** is a totally ordered set
  - e.g., **0-5** stars, real number in **[0,1]**

Key Problems

- **(1) Gathering "known" ratings for matrix**
  - How to collect the data in the utility matrix
- **(2) Extrapolate unknown ratings from the known ones**
  - Mainly interested in high unknown ratings
    - We are not interested in knowing what you don't like but what you like
- **(3) Evaluating extrapolation methods**
  - How to measure success/performance of recommendation methods

(1) Gathering Ratings

- **Explicit**
  - Ask people to rate items
  - Doesn't work well in practice – people can't be bothered
- **Implicit**
  - Learn ratings from user actions
    - E.g., purchase implies high rating
  - What about low ratings?

(2) Extrapolating Utilities

- **Key problem:** Utility matrix ***U*** is **sparse**
  - Most people have not rated most items
  - **Cold start:**
    - New items have no ratings
    - New users have no history
- **Three approaches to recommender systems:**
  - **1)** Content-based
  - **2)** Collaborative
  - **3)** Latent factor based

} **Today!**

Content-based Recommender Systems

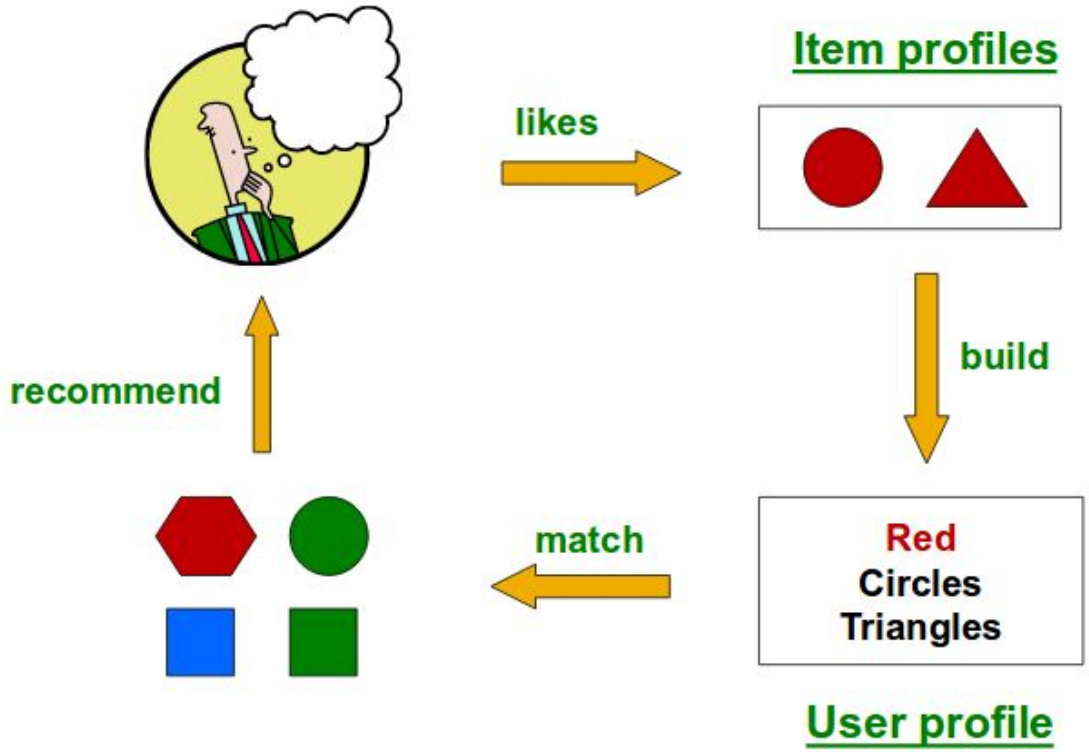
Content-based Recommendations

- **Main idea:** Recommend items to customer ***x*** similar to previous items rated highly by ***x***

Example:

- **Movie recommendations**
  - Recommend movies with same actor(s), director, genre, …
- **Websites, blogs, news**
  - Recommend other sites with "similar" content

Plan of Action



Item Profiles

- For each item, create an **item profile**
- Profile is a set of features
  - **Movies:** author, title, actor, director,…
  - **Images, videos:** metadata and tags
  - **People:** Set of friends
- Convenient to think of the item profile as a vector
  - One entry per feature (e.g., each actor, director,…)
  - Vector might be boolean or real-valued

Item Profile

Text features

- Profile = set of "important" words in item (document)
- How to pick important words?
  - Usual heuristic from text mining is **TF-IDF** (Term frequency * Inverse Doc Frequency)

Item Profile

Sidenote: TF-IDF

f_{ij} = frequency of term (feature) i in doc (item) j

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Note: we normalize TF to discount for "longer" documents

n_i = number of docs that mention term i

N = total number of docs

$$IDF_i = \log \frac{N}{n_i}$$

TF-IDF score: $w_{ij} = TF_{ij} \times IDF_i$

Doc profile = set of words with highest **TF-IDF** scores, together with their scores

User profiles

Example 1: Boolean Utility Matrix

- Items are movies, only feature is "Actor"
  - Item profile: vector with 0 or 1 for each Actor
- Suppose user x has watched 5 movies
  - 2 movies featuring actor A
  - 3 movies featuring actor B
- User profile = mean of item profiles
  - Feature A's weight = 2/5 = 0.4
  - Feature B's weight = 3/5 = 0.6

User profiles

Normalize ratings remove bias.

Example 2: Star Ratings

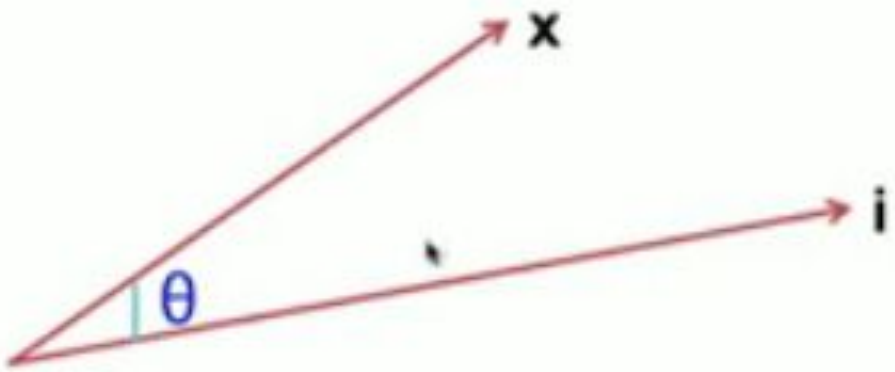
- Same example, 1-5 star ratings
  - Actor A's movies rated 3 and 5
  - Actor B's movies rated 1, 2 and 4
- Useful step: Normalize ratings by subtracting user's mean rating (3)
  - Actor A's normalized ratings = 0, +2
    - Profile weight = (0 + 2)/2 = 1
  - Actor B's normalized ratings = -2, -1, +1
    - Profile weight = -2/3

- Now we have user profile and item profile ready.
- The next task is to recommend some items to the user.
- The key step in this is to take a pair of user profile and item profile.
- Figure out what the rating for that user and item pair is likely to be.

- Both user profile and item profiles are vectors in high dimensional space.
- In higher dimensional space a good distance metric between the pairs of vector is angle between two vectors.
- Angle can estimated using cosine formula.

Making predictions

- User profile **x**, Item profile **i**
- Estimate $U(\mathbf{x},\mathbf{i}) = \cos(\theta) = (\mathbf{x} \cdot \mathbf{i})/(|\mathbf{x}|\,|\mathbf{i}|)$



So the way we make predictions are as follows. Given the user X, we compute the cosine similarity between that user and all the items in that catalog. And then we pick the items with highest cosine similarity and recommend those to the user. That's the theory of content based recommendation.

Pros: Content-based Approach

- No need for data on other users
- Able to recommend new & unpopular items
  - No first-rater problem
- Explanations for recommended items
  - Content features that caused an item to be recommended

Cons: Content-based Approach

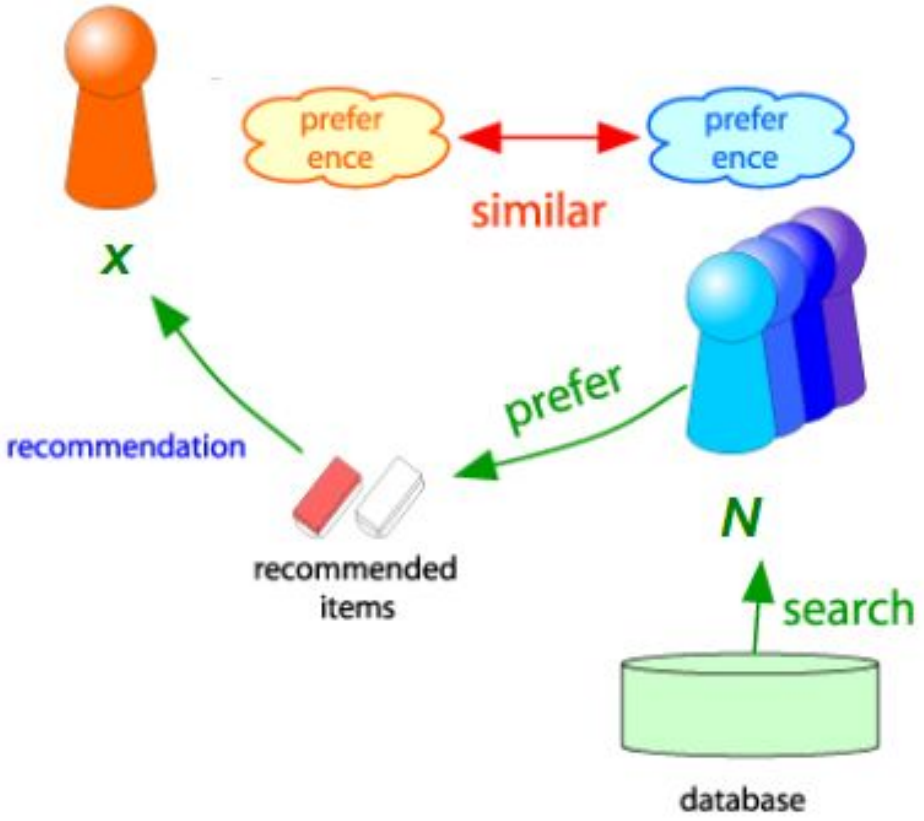
- Finding the appropriate features is hard
  - E.g., images, movies, music
- Overspecialization
  - Never recommends items outside user's content profile
  - People might have multiple interests
  - Unable to exploit quality judgments of other users
- Cold-start problem for new users
  - How to build a user profile?

Collaborative Filtering

Harnessing quality judgments of other users

Collaborative Filtering

- Consider user ***x***
- Find set ***N*** of other users whose ratings are "**similar**" to ***x***'s ratings
- Estimate ***x***'s ratings based on ratings of users in ***N***



Similar Users (1)

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- Consider users ***x*** and ***y*** with rating vectors r_x and r_y
- We need a similarity metric sim(***x***, ***y***)
- Capture intuition that sim(A,B) > sim(A,C)

Option 1: Jaccard Similarity

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- $sim(A,B) = | r_A \cap r_B | / | r_A \cup r_B |$
- sim(A,B) = 1/5; sim(A,C) = 2/4
  - sim(A,B) < sim(A,C)
- Problem: Ignores rating values!

Option 2: Cosine similarity

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

- sim(A,B) = cos(r_A, r_B)
- sim(A,B) = 0.38, sim(A,C) = 0.32
  - sim(A,B) > sim(A,C), but not by much
- Problem: treats missing ratings as negative

Option 3: Centered cosine

- Normalize ratings by subtracting row mean

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	2/3			5/3	−7/3		
B	1/3	1/3	−2/3				
C				−5/3	1/3	4/3	
D		0					0

- sim(A,B) = cos(r_A, r_B) = 0.09; sim(A,C) = -0.56
  - sim(A,B) > sim(A,C)
- Captures intuition better
  - Missing ratings treated as "average"
  - Handles "tough raters" and "easy raters"

Rating Predictions

- Let r_x be the vector of user x's ratings
- Let N be the set of k users most similar to x who have also rated item i
- Prediction for user x and item i

- Option 1: $r_{xi} = 1/k \sum_{y \in N} r_{yi}$
- Option 2: $r_{xi} = \sum_{y \in N} s_{xy} r_{yi} / \sum_{y \in N} s_{xy}$

where $s_{xy} = \text{sim}(x,y)$

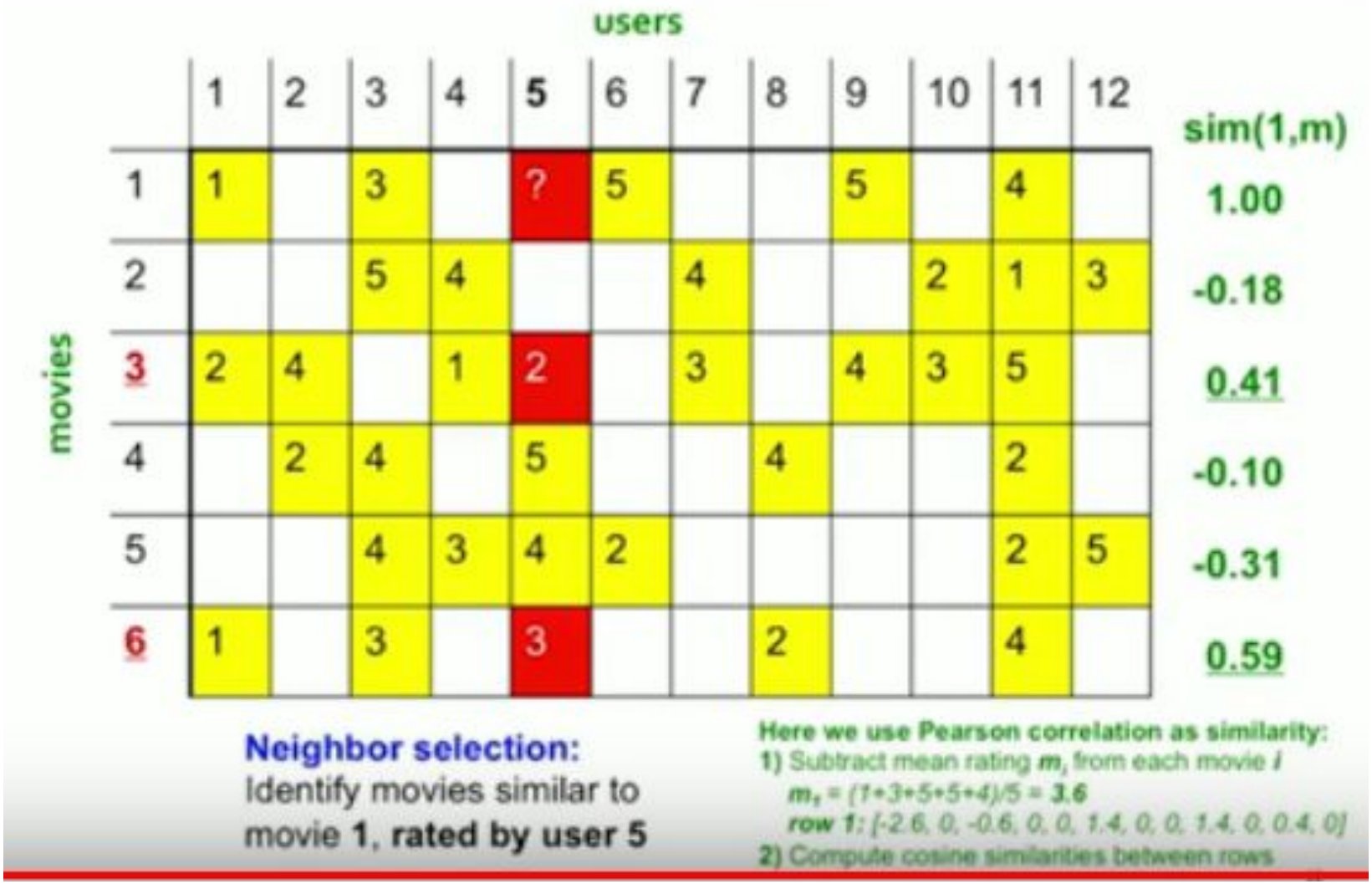
Item-Item Collaborative Filtering

- So far: User-user collaborative filtering
- Another view: Item-item
  - For item *i*, find other similar items
  - Estimate rating for item *i* based on ratings for similar items
  - Can use same similarity metrics and prediction functions as in user-user model

$$r_{xi} = \frac{\sum_{j \in N(i;x)} s_{ij} \cdot r_{xj}}{\sum_{j \in N(i;x)} s_{ij}}$$

s_{ij}... similarity of items *i* and *j*
r_{xj}...rating of user *x* on item *j*
$N(i;x)$... set items rated by *x* similar to *i*

Item-Item CF (|N|=2)



Predict by taking weighted average:

$r_{15} = (0.41*2 + 0.59*3) / (0.41+0.59) = 2.6$

Item-Item v. User-User

- In theory, user-user and item-item are dual approaches
- In practice, item-item outperforms user-user in many use cases
- Items are "simpler" than users
  - Items belong to a small set of `diversification` , users have varied tastes
  - Item Similarity is more meaningful than User Similarity

Pros/Cons of Collaborative Filtering

- **+ Works for any kind of item**
  - No feature selection needed
- **- Cold Start:**
  - Need enough users in the system to find a match
- **- Sparsity:**
  - The user/ratings matrix is sparse
  - Hard to find users that have rated the same items
- **- First rater:**
  - Cannot recommend an item that has not been previously rated
  - New items, Esoteric items
- **- Popularity bias:**
  - Cannot recommend items to someone with unique taste
  - Tends to recommend popular items