

Lecture 13

K-means and Mean Shift and Graph Cuts

Administrative

A3 is out

- Due Nov 12

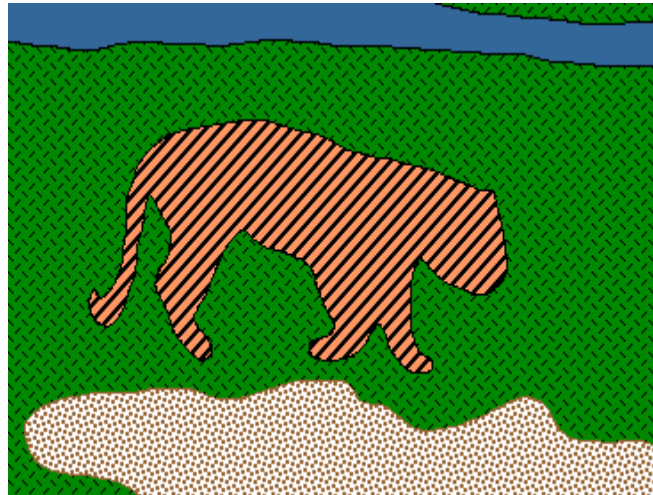
Administrative

Recitation

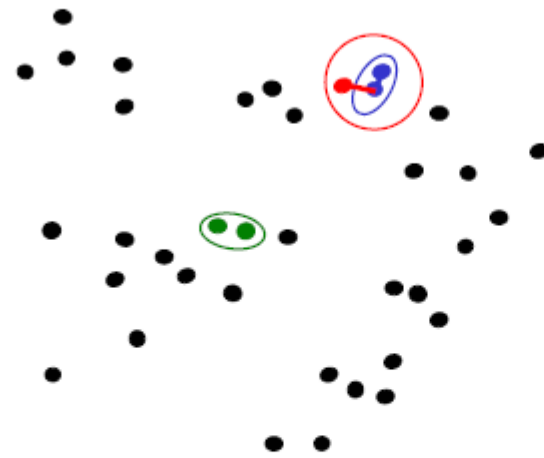
- Ontologies

So far: Segmentation and clustering

- Goal: identify groups of pixels that go together



So far: Agglomerative clustering



1. Say "Every point is its own cluster"
2. Find "most similar" pair of clusters
3. Merge it into a parent cluster
4. Repeat



Today's agenda

- K-means clustering
- Mean-shift clustering
- Normalized cuts

Reading:

Szeliski, 2nd edition, Chapter 7.5

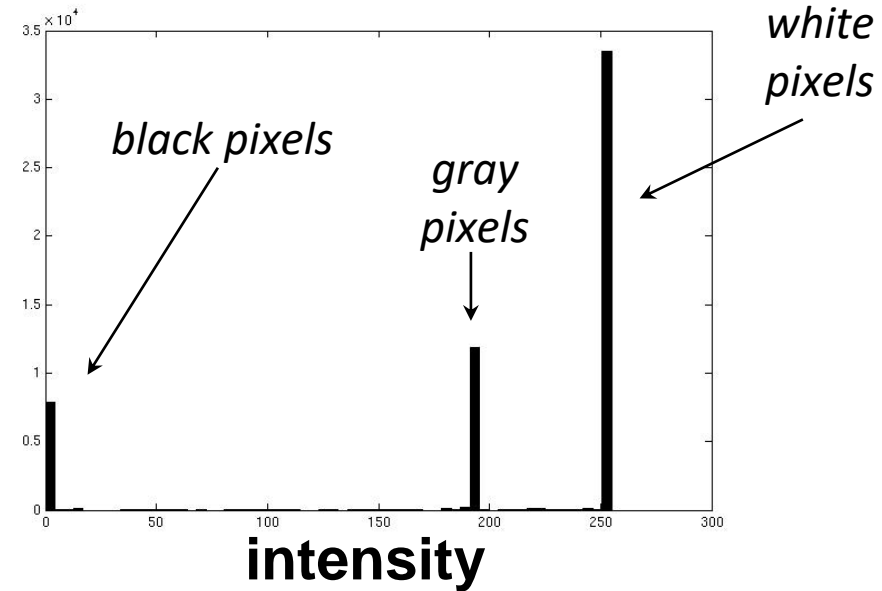
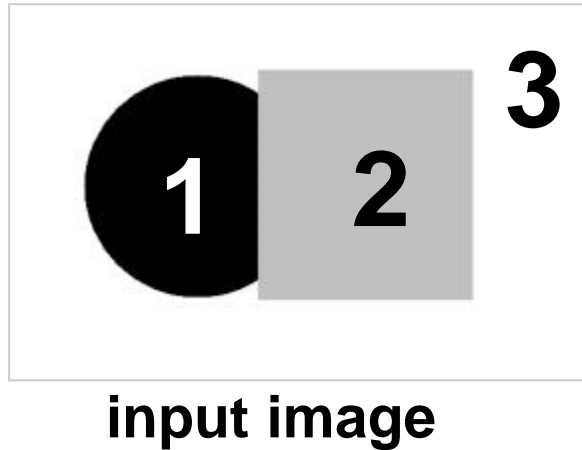
Today's agenda

- K-means clustering
- Mean-shift clustering
- Normalized cuts

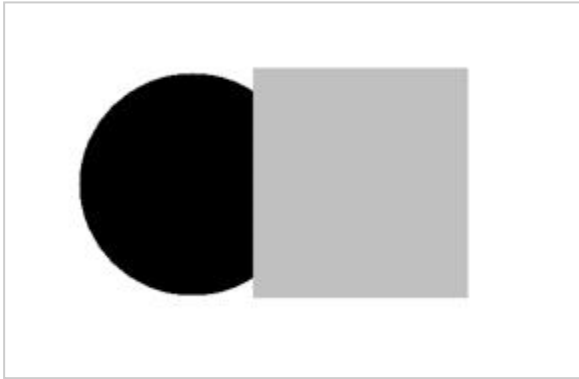
Reading: Szeliski Chapters: [5.2.2](#), 7.5.2

D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, PAMI 2002.

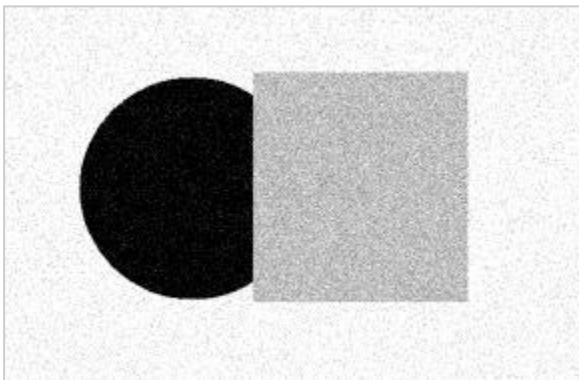
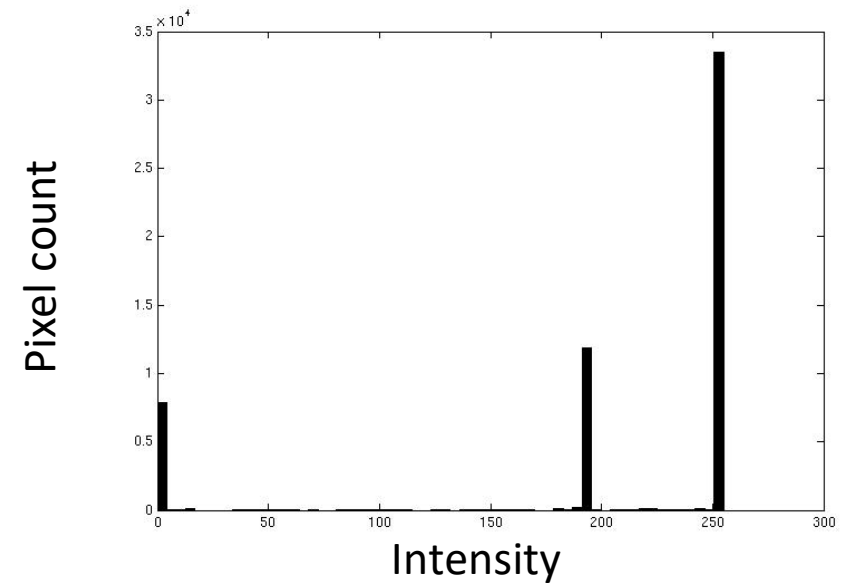
Image Segmentation: Binary image Example



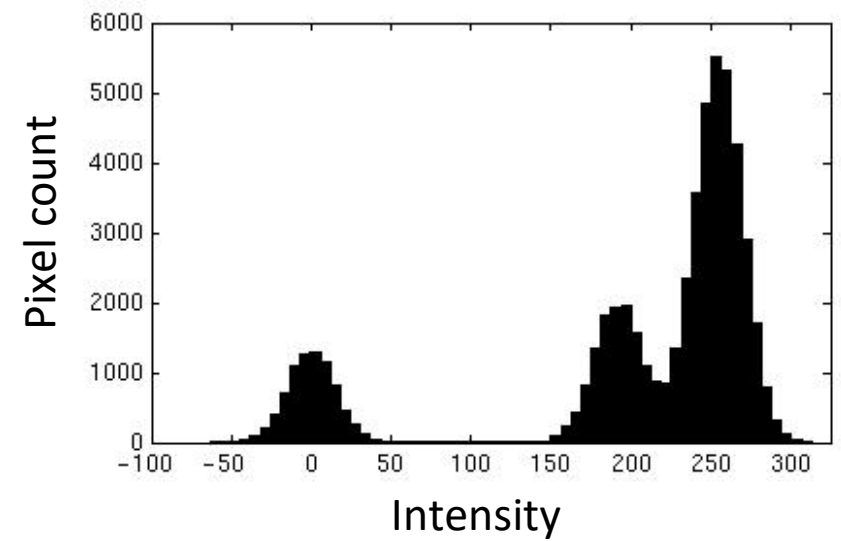
- These pixel values show that there are three things in the image.
- We could label every pixel in the image according to which of these primary intensities it is.
 - i.e., segment the image based on the intensity feature.
- What if the image isn't quite so simple?

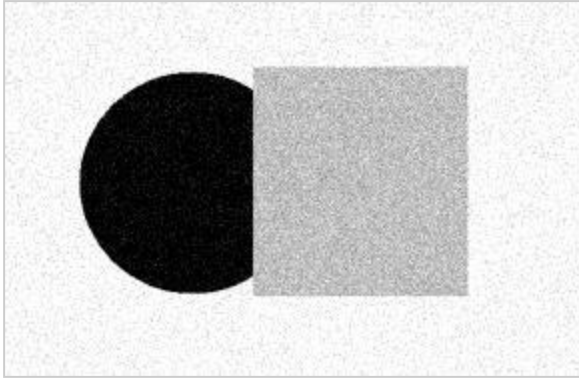


Input image

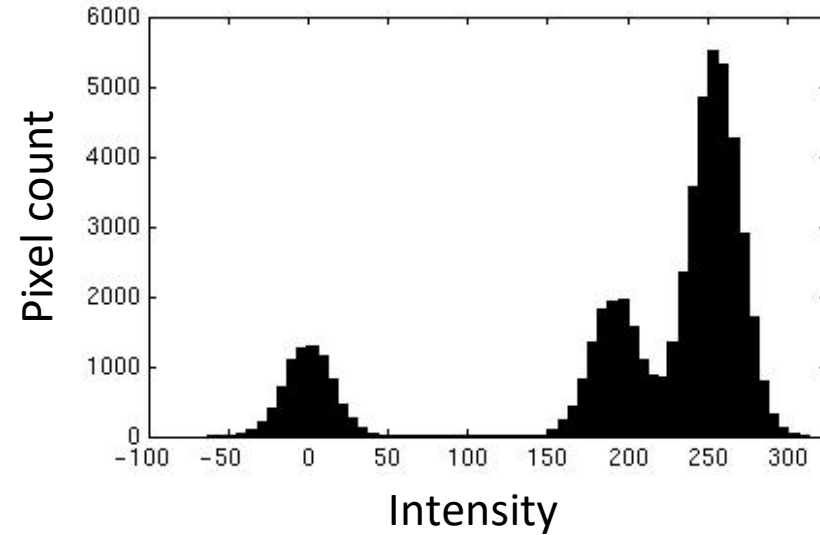


Input image

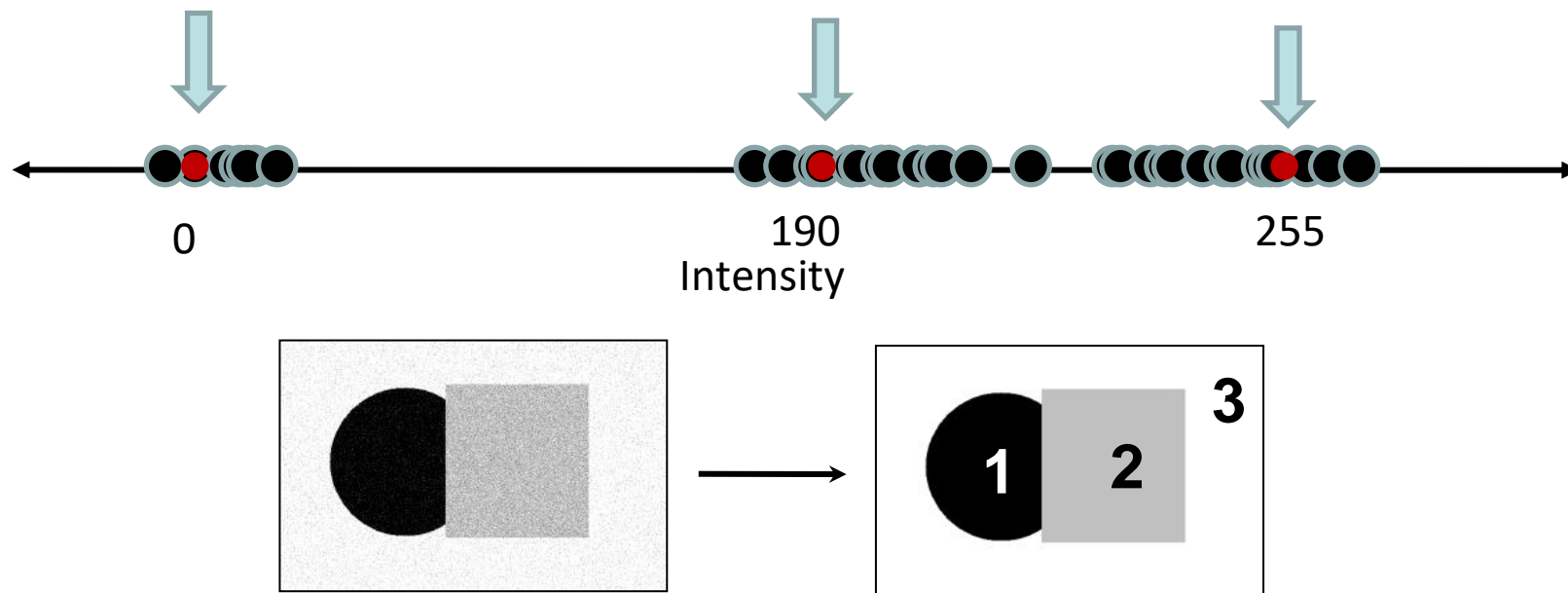




Input image



- How do we determine the three main intensities that define our groups?
- Each cluster has a cluster center
 - A mean cluster value.

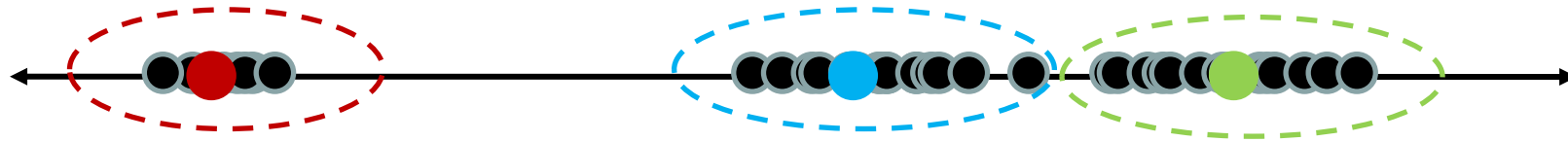


- Goal: choose three “**centers**” as the representative intensities and label every pixel according to which of these centers it is nearest to.
- **Best cluster centers** are those that minimize **Sum of Square Distance** (SSD) between all points and their nearest cluster center c_i :

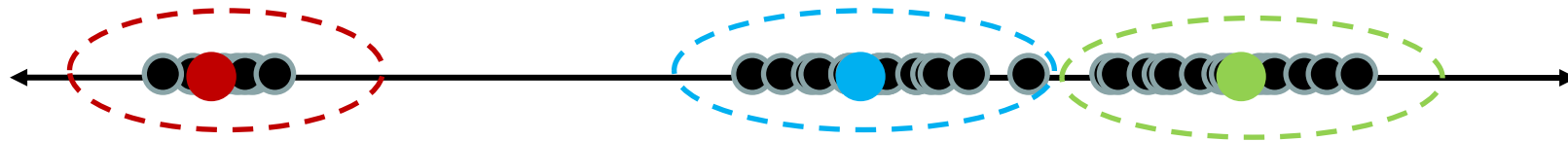
$$SSD = \sum_C \sum_{v \in C} (v - c_i)^2$$

Clustering

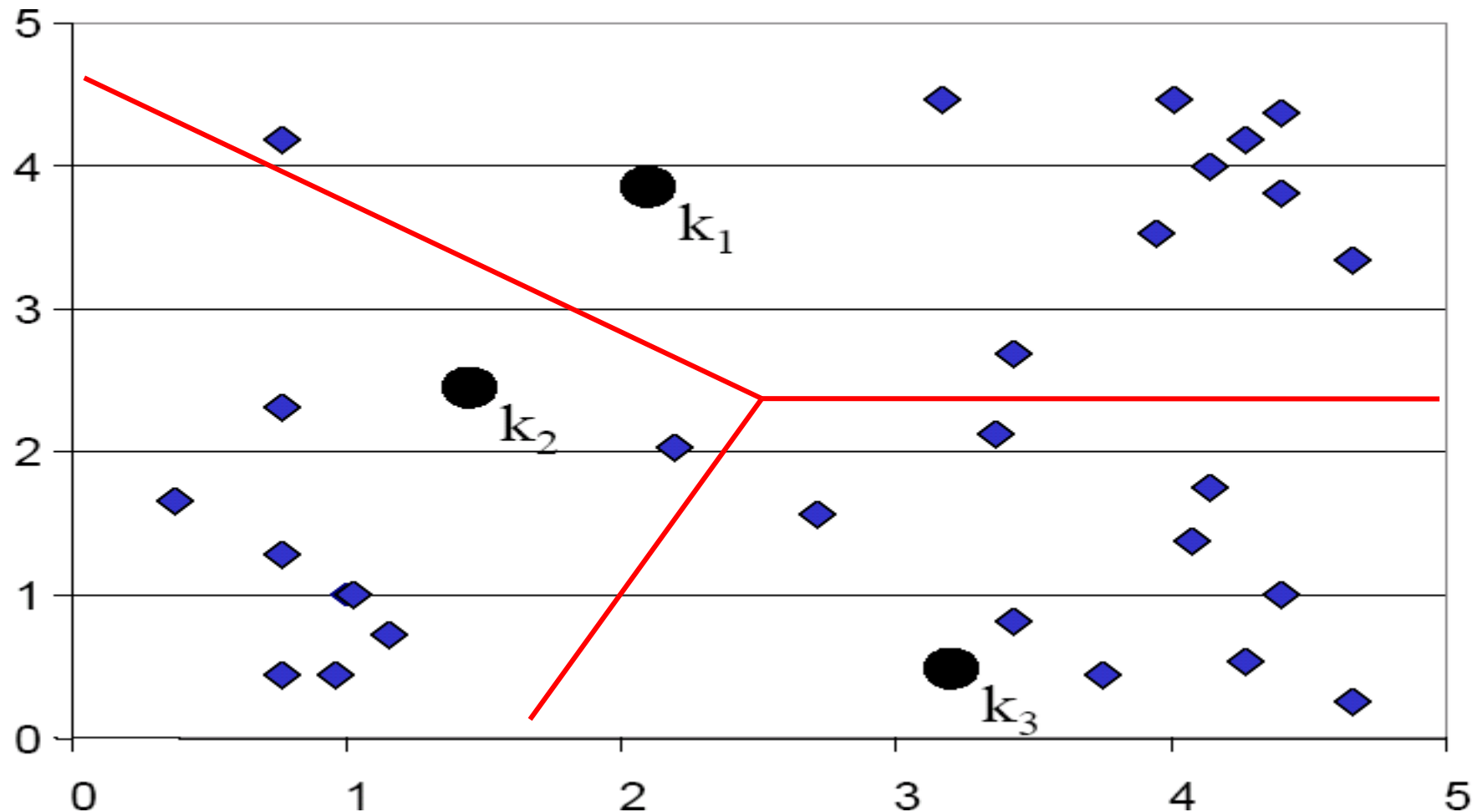
- With this objective, it is a “chicken and egg” problem:
 - If we knew the *cluster centers*, we could allocate points to groups by assigning each to its closest center.



- If we knew the *group memberships*, we could get the centers by computing the mean per group.



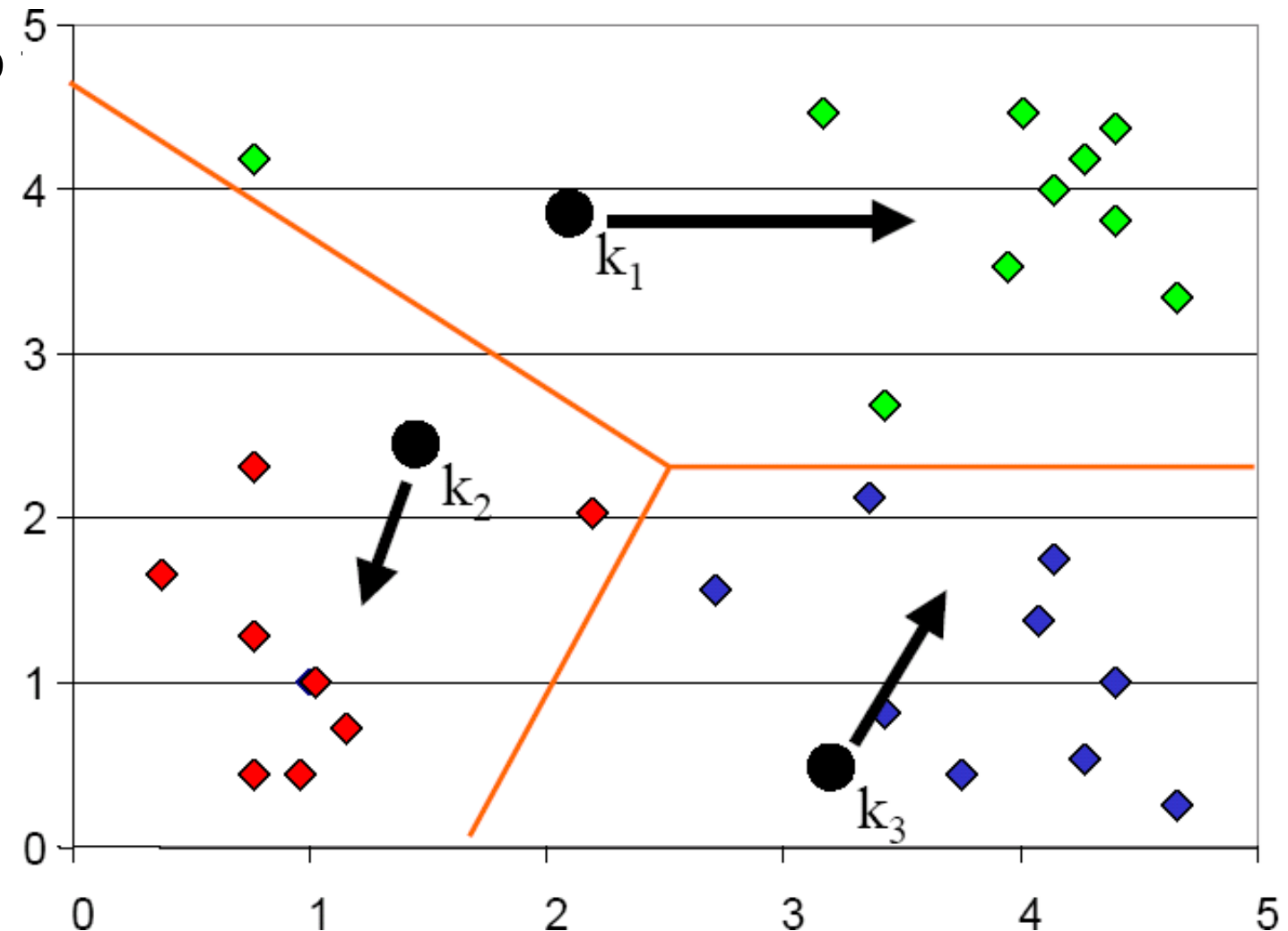
Given, a set of points, randomly select $k=3$ of them to be the cluster centers



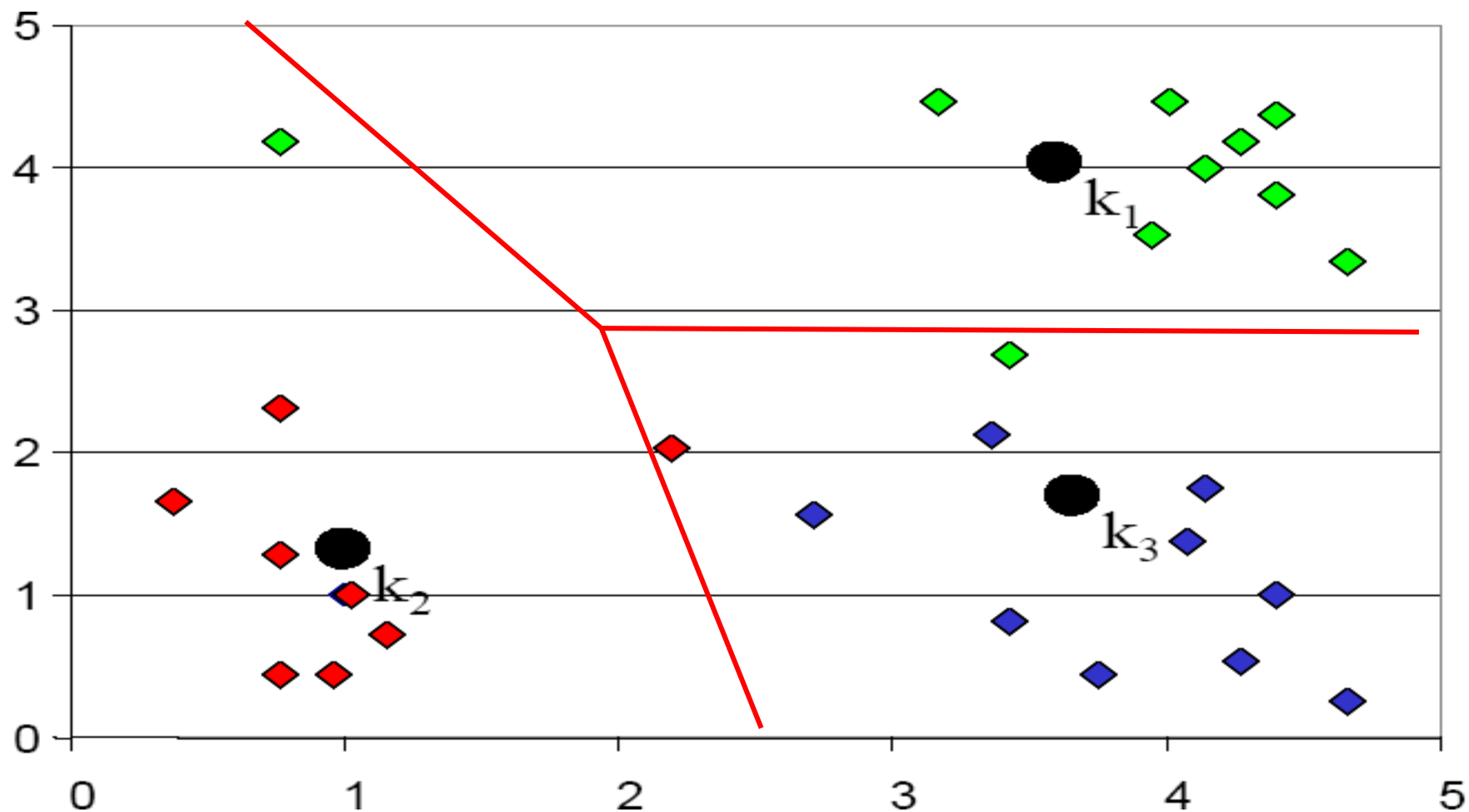
**Voronoi
diagram**

Categorize each point into a cluster defined by its closest center.

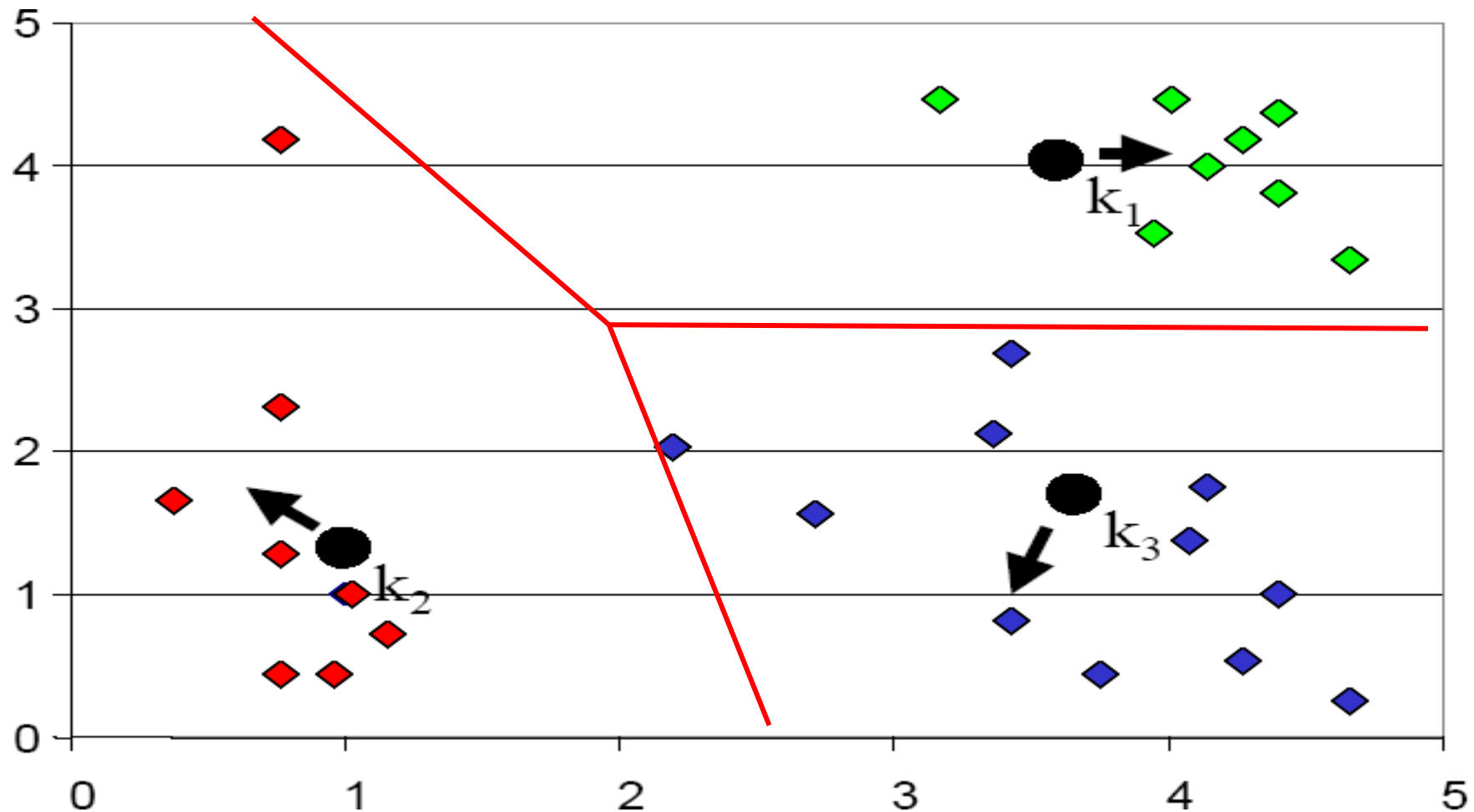
Next, move the cluster centers to location amongst its cluster



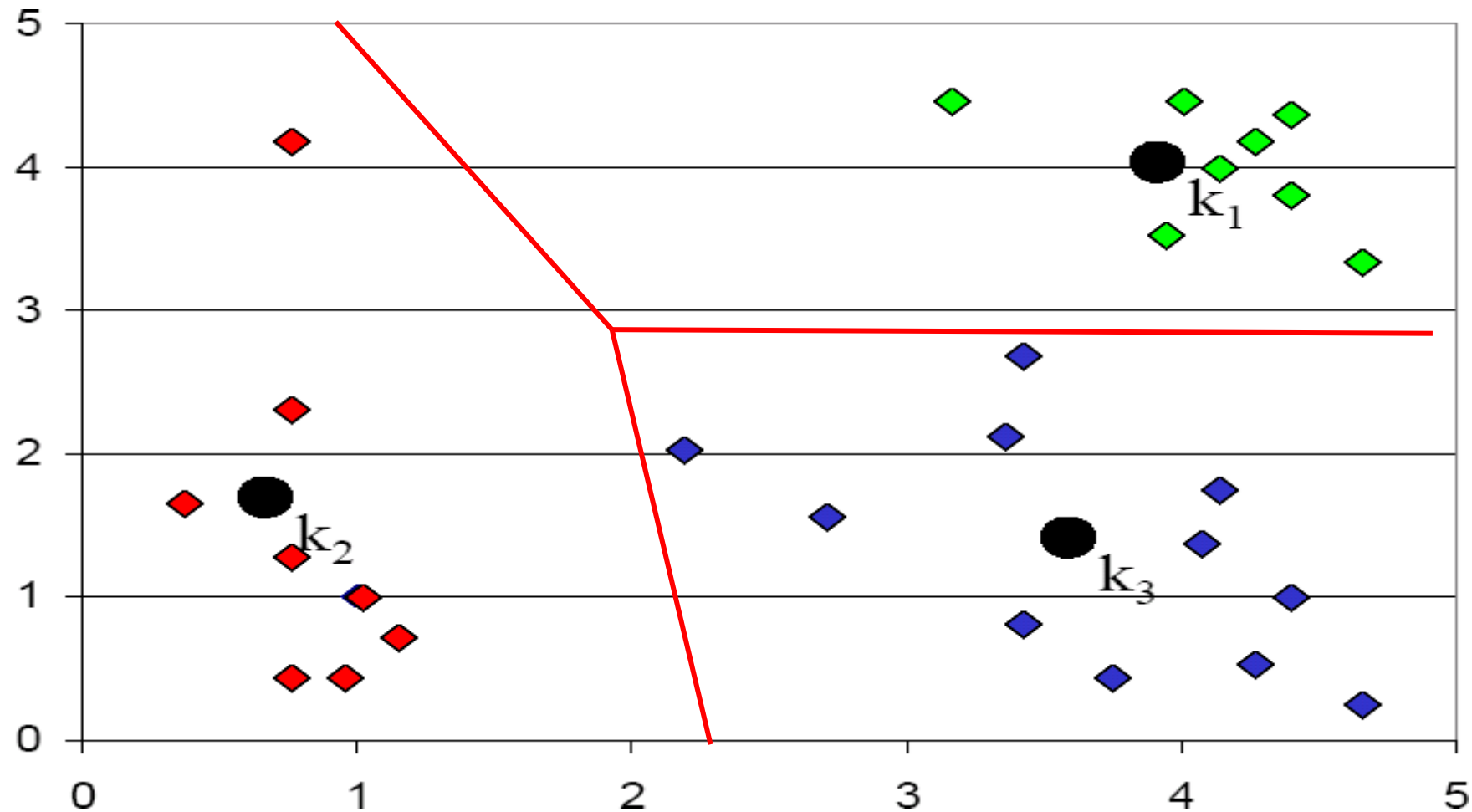
Repeat with new cluster center locations



Categorize into new clusters.
Move center to the mean



Repeat with new cluster centers



Computational Complexity

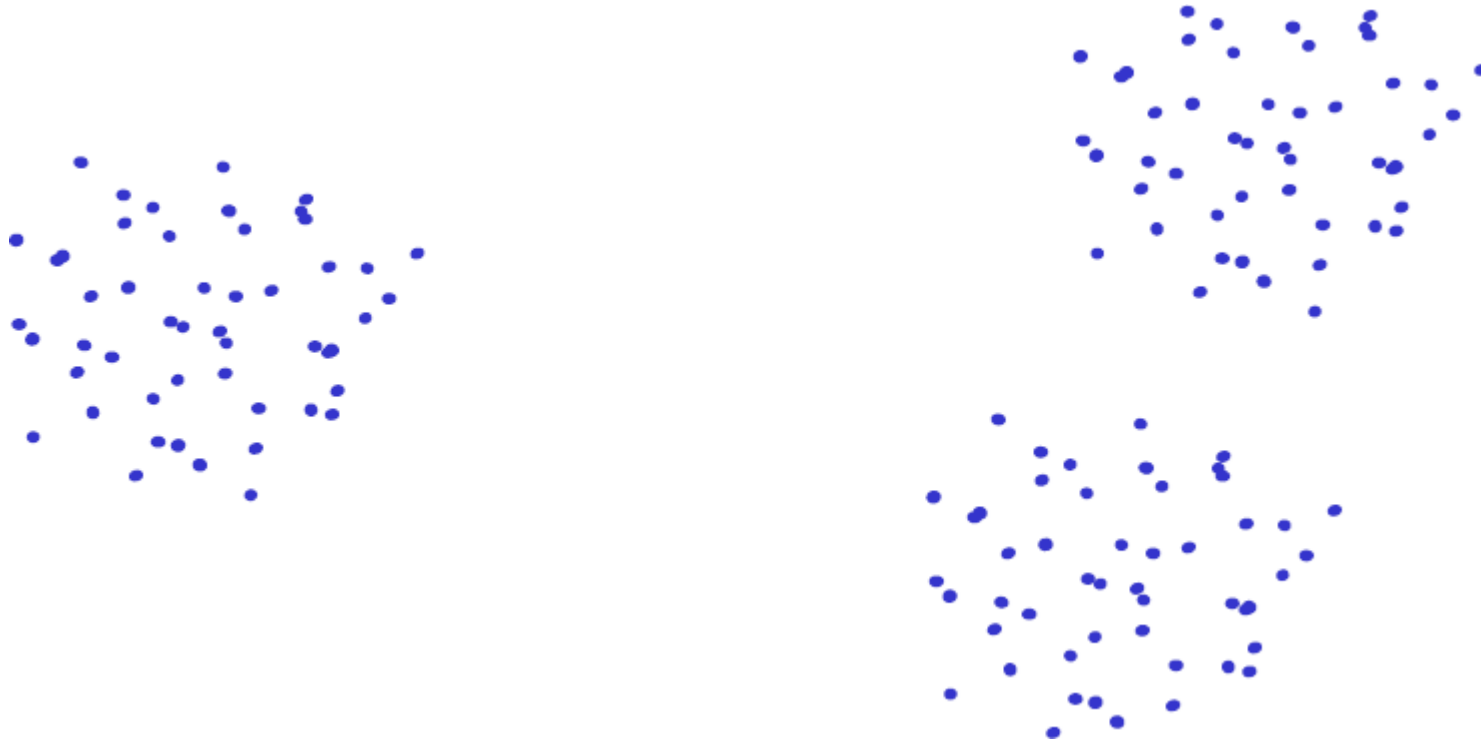
At each iteration,

- Computing distance between each of the n objects and the K cluster centers is $O(Kn)$.
- Computing cluster centers: Each object gets added once to some cluster: $O(n)$.

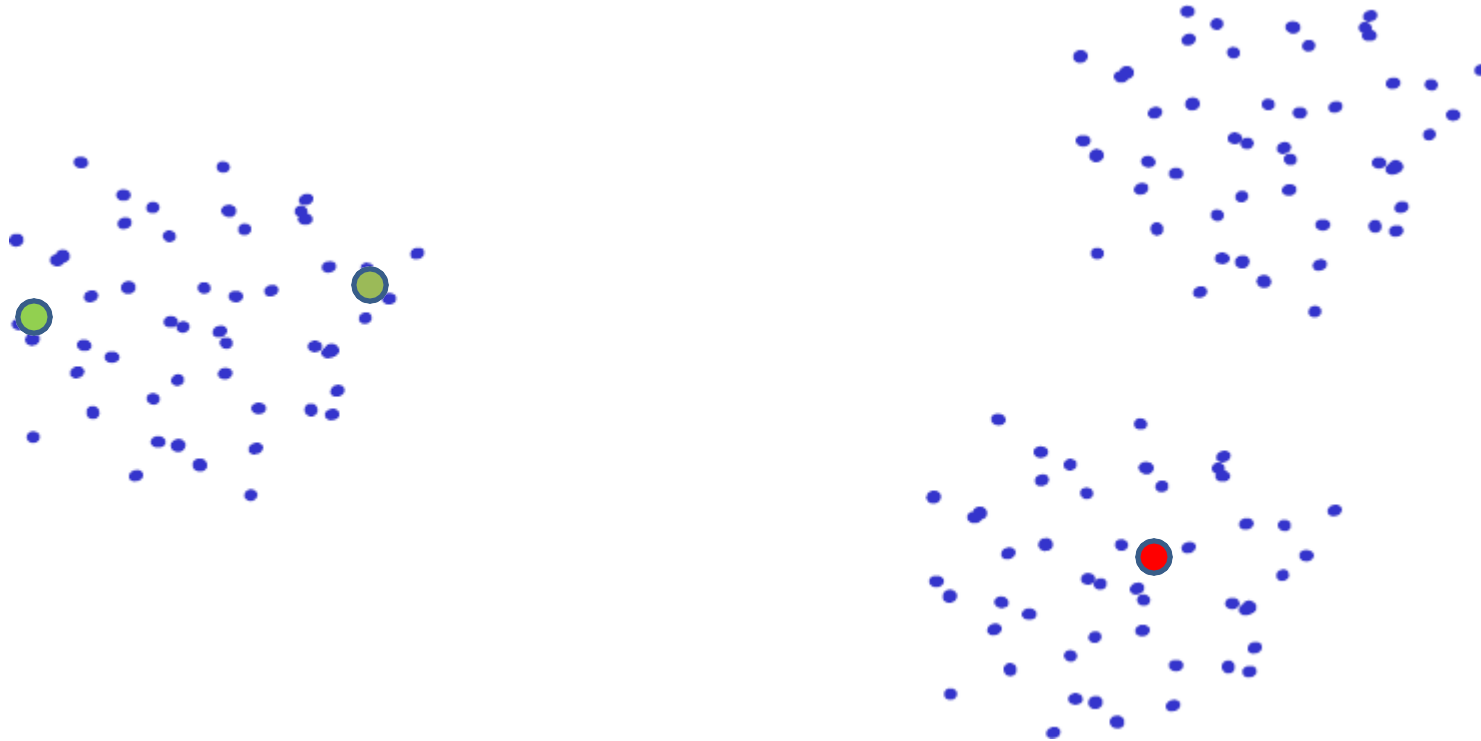
Assume these two steps are each done once for I iterations: $O(IKn)$.

Q. Is K-means guaranteed to converge to a global optimum?

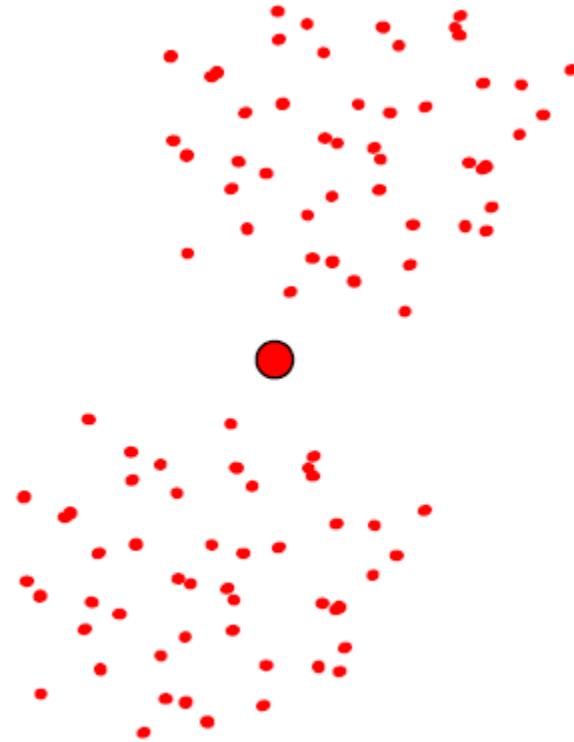
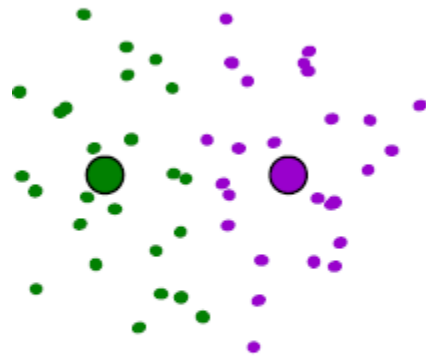
Results are quite sensitive to seed selection.



Results are quite sensitive to seed selection.



Results are quite sensitive to seed selection.



Results are quite sensitive to seed selection.

- Some seeds can result in poor convergence rate, or convergence to sub-optimal clustering.
- **So it may converge to a local minimum**
- Select good seeds using a heuristic (e.g., object least similar to any existing mean)
- **You can pick seeds from an image histogram**
- Try out multiple starting points (very important!!!)
- Initialize with the results of another method.

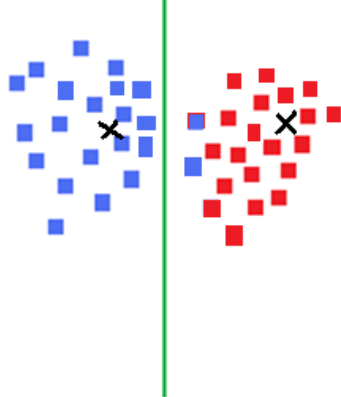
Other issues with k-means

Shape of clusters

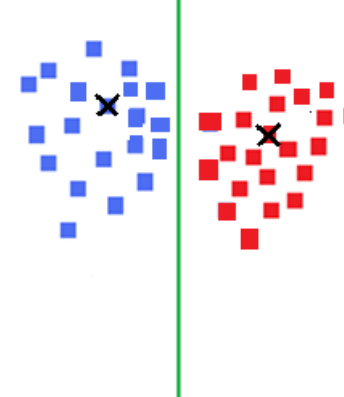
- Assumes convex clusters

Sensitive to Outliers

Outlier causes
misclassifications

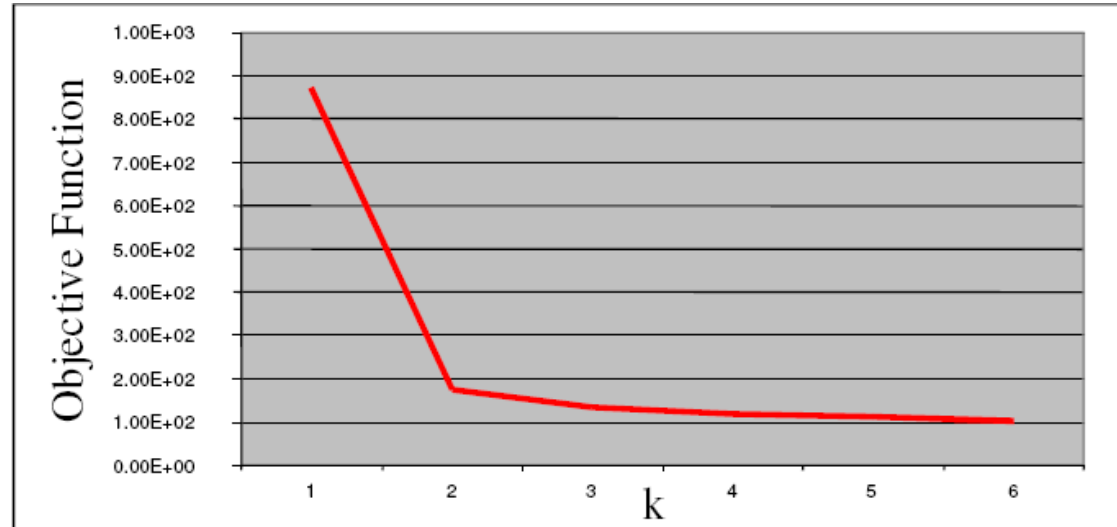


Ideal decision boundary



How to choose the value of k

- Number of clusters K
 - Objective function
 - Look for “Knee” in objective function



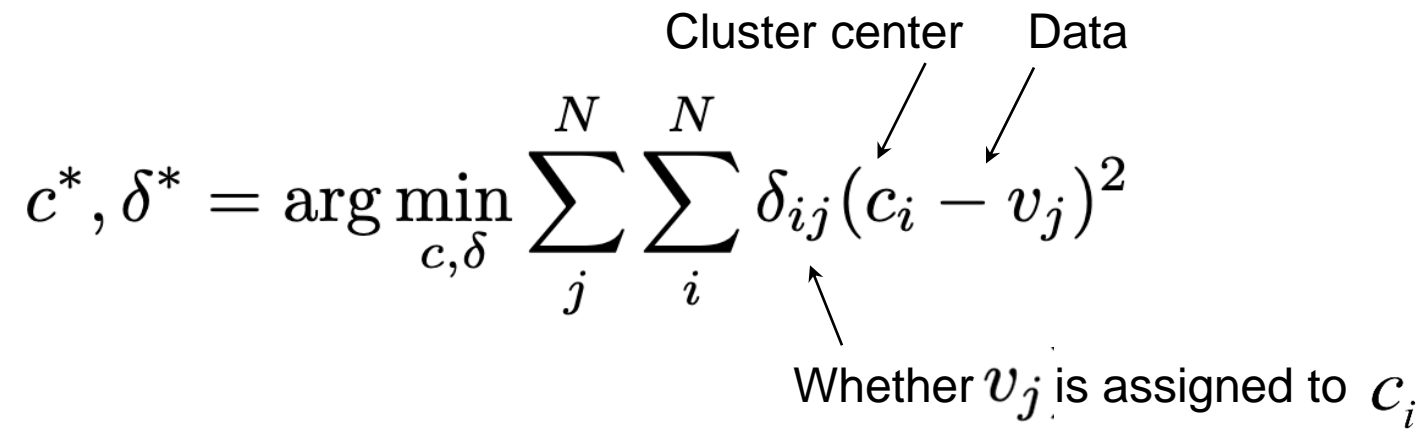
Clustering

Goal: cluster to minimize distance of pixels to their cluster centers

$$c^*, \delta^* = \arg \min_{c, \delta} \sum_j^N \sum_i^N \delta_{ij} (c_i - v_j)^2$$

Cluster center Data

Whether v_j is assigned to c_i



K-means clustering

1. Initialize ($t = 0$): cluster centers c_1, \dots, c_K

K-means clustering

1. Initialize ($t = 0$): cluster centers c_1, \dots, c_K
2. Compute δ^t : assign each point to the closest center
 - o δ^t denotes the set of assignment for each v_j to cluster c_i at iteration t

$$\delta^t = \arg \min_{\delta} \frac{1}{N} \sum_j^N \sum_i^K \delta_{ij}^{t-1} (c_i^{t-1} - v_j)^2$$

K-means clustering

1. Initialize ($t = 0$): cluster centers c_1, \dots, c_K
2. Compute δ^t : assign each point to the closest center
 - o δ^t denotes the set of assignment for each v_j to cluster c_i at iteration t

$$\delta^t = \arg \min_{\delta} \frac{1}{N} \sum_j \sum_i^K \delta_{ij}^{t-1} (c_i^{t-1} - v_j)^2$$

3. Computer c^t : update cluster centers as the mean of the points

$$c_i^t = 1/N \sum_j \delta_{ij}^t v_j$$

K-means clustering

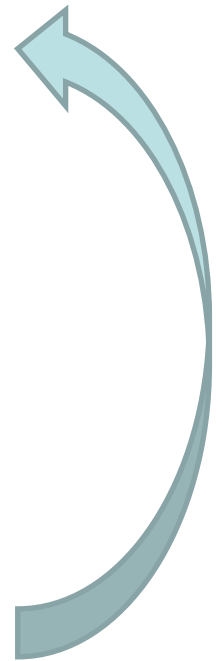
1. Initialize ($t = 0$): cluster centers c_1, \dots, c_K
2. Compute δ^t : assign each point to the closest center
 - o δ^t denotes the set of assignment for each v_j to cluster c_i at iteration t

$$\delta^t = \arg \min_{\delta} \frac{1}{N} \sum_j \sum_i^K \delta_{ij}^{t-1} (c_i^{t-1} - v_j)^2$$

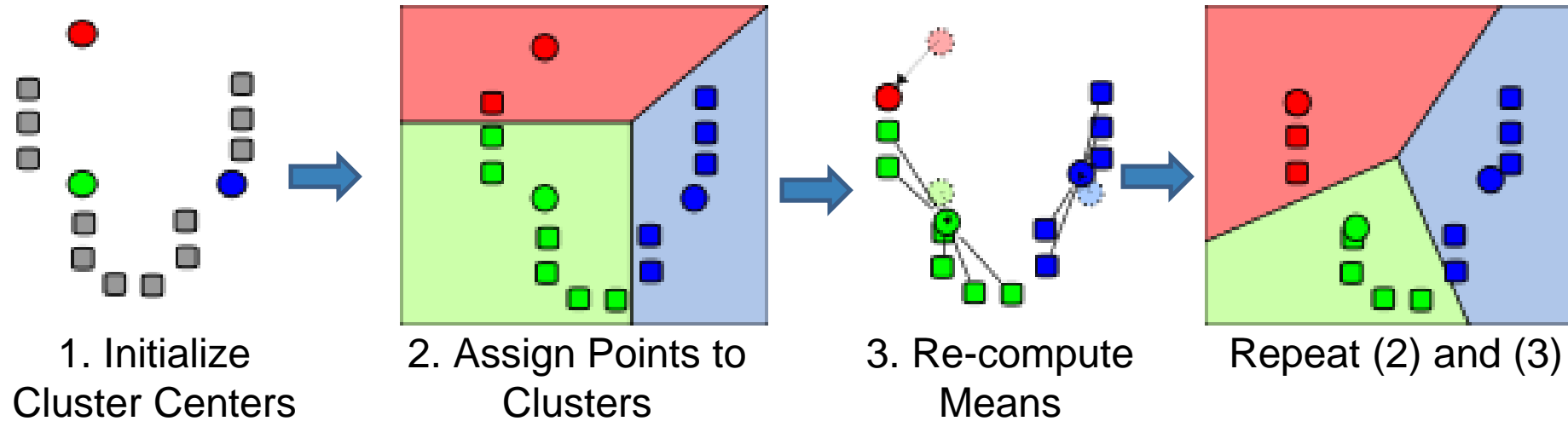
3. Computer C^t : update cluster centers as the mean of the points

$$c_i^t = 1/N \sum_j \delta_{ij}^t v_j$$

Update $t = t + 1$, Repeat Step 2-3 till stopped



K-means clustering



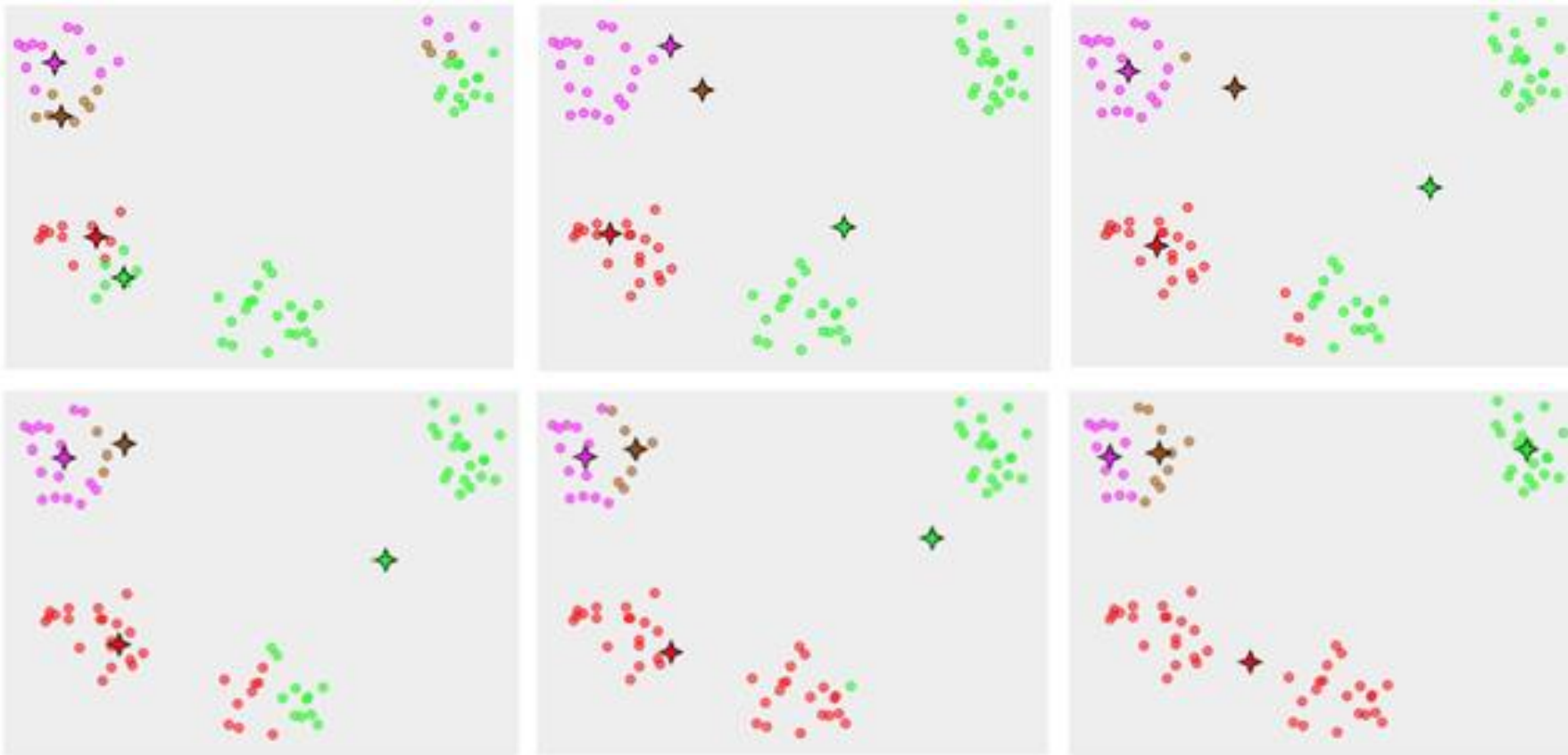
K-means clustering

Initial cluster centers are randomly initialized

- Can lead to bad initializations
- Can cause bad clusters

Another example of how K-means Converges to a local minimum solution

Initialize multiple runs!



K-Means++

Tries to prevent arbitrarily bad local minima?

1. Randomly choose first center.
2. Pick new center with prob. proportional to $(c_i - v_j)^2$
 - a. Basically we want to find as good of an initialization as possible
3. Repeat until K centers.

K-means clustering

Initial cluster centers are randomly initialized

- Can lead to bad initializations
- Can cause bad clusters

Different distance measures can change K-Means clusters

- Euclidean distance of cosine distance.

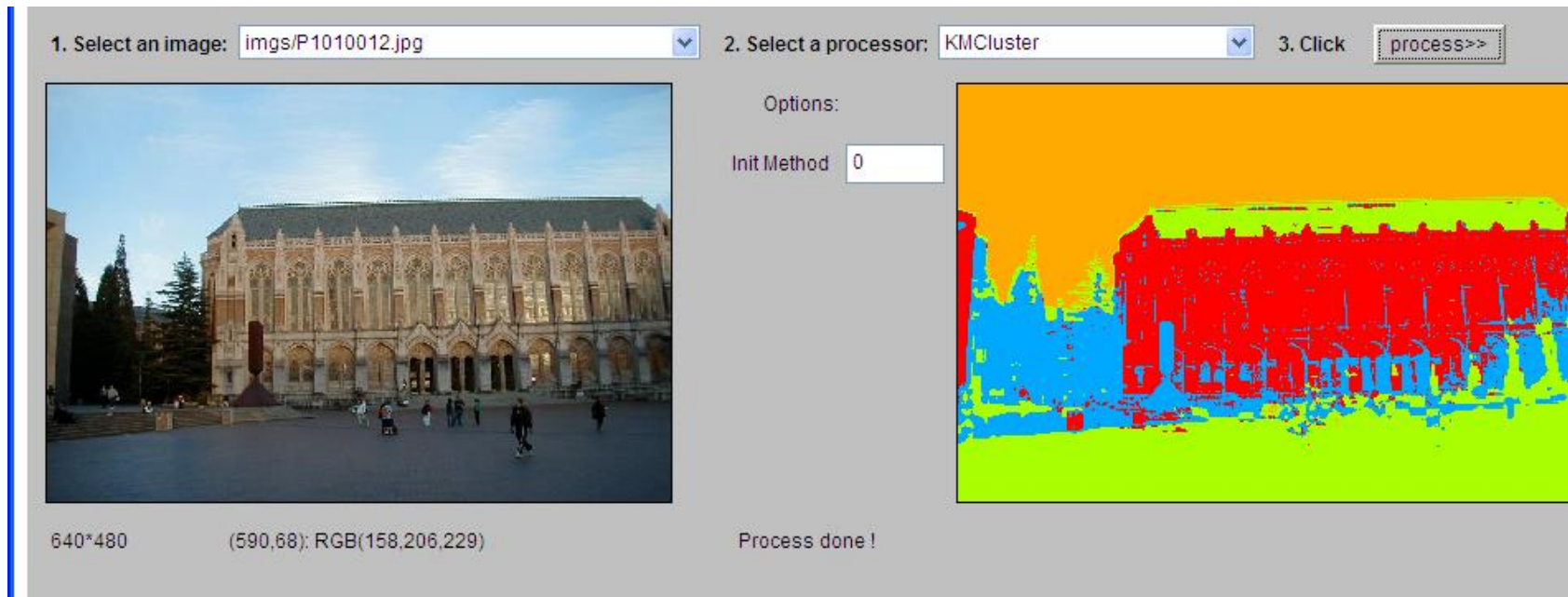
Different feature space can lead to different cluster

Example

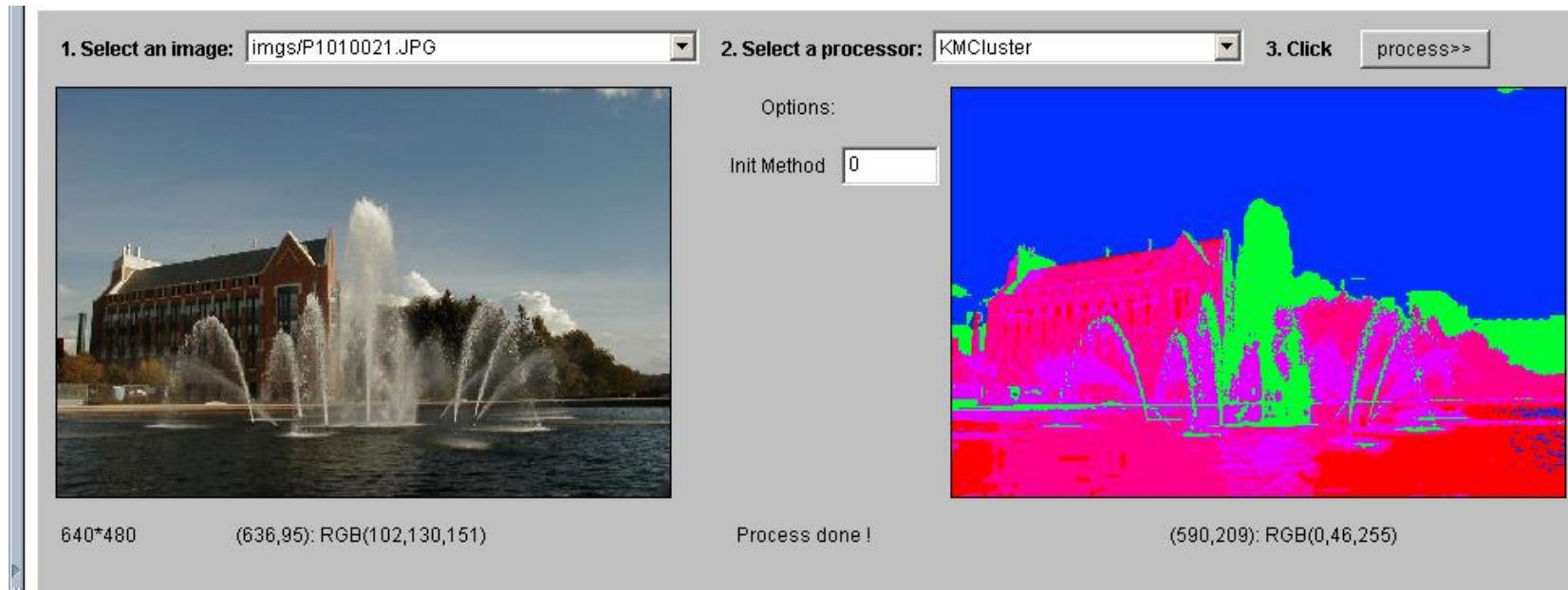


Figure 10.4: Football image (left) and $K=6$ clusters resulting from a K-means clustering procedure (right) shown as distinct gray tones. The six clusters correspond to the six main colors in the original image: dark green, medium green, dark blue, white, silver, and black.

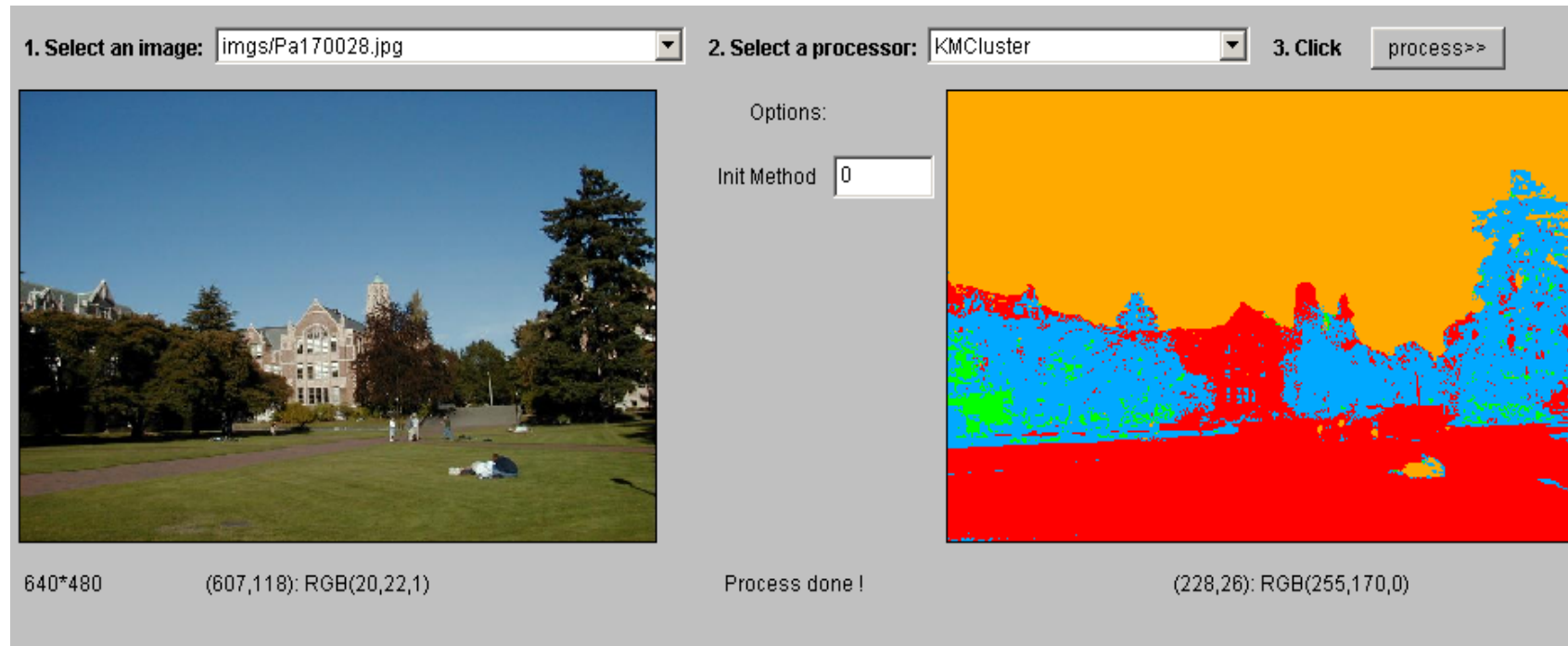
● K-Means Example 1



● K-Means Example 2



● K-Means Example 3



Segmentation as Clustering



Original image



2 clusters



3 clusters

Feature Space: pixel value

- Feature space: what measurements do we include in x_i ?
- Depending on what we choose as the *feature space*, we can group pixels in different ways.

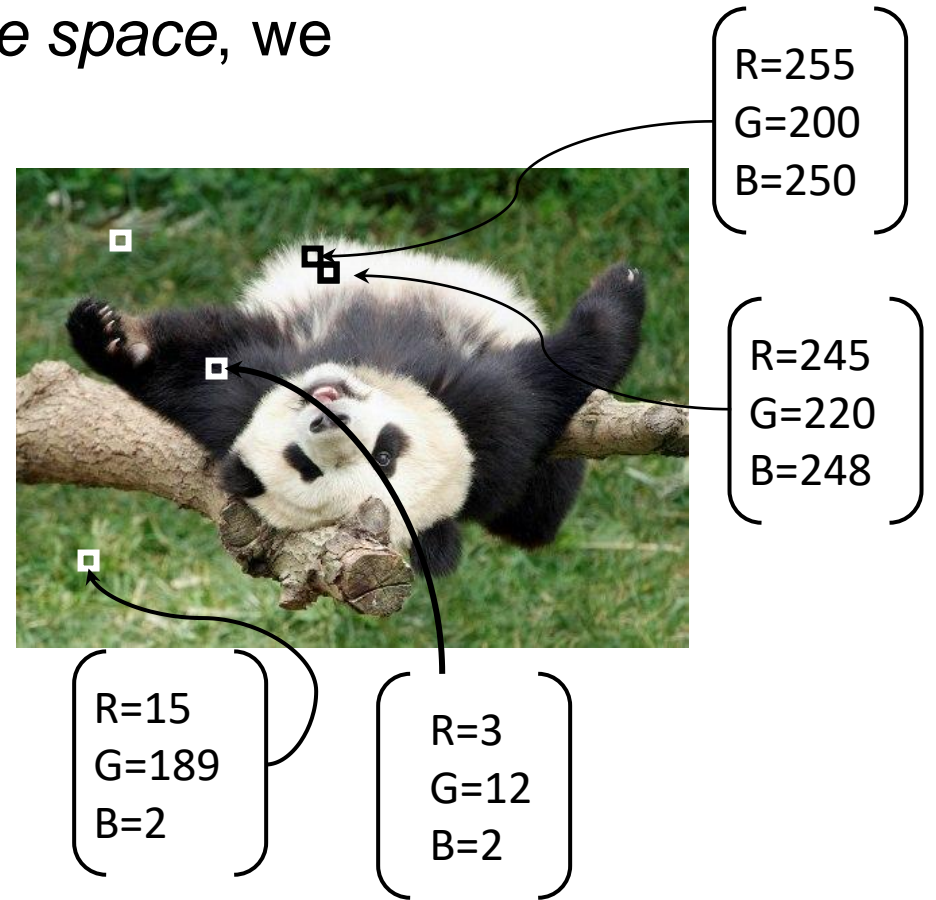
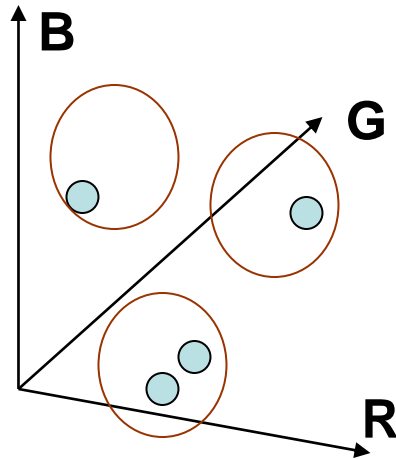
- Grouping pixels based on **intensity** similarity



- Feature space: intensity value (1D)

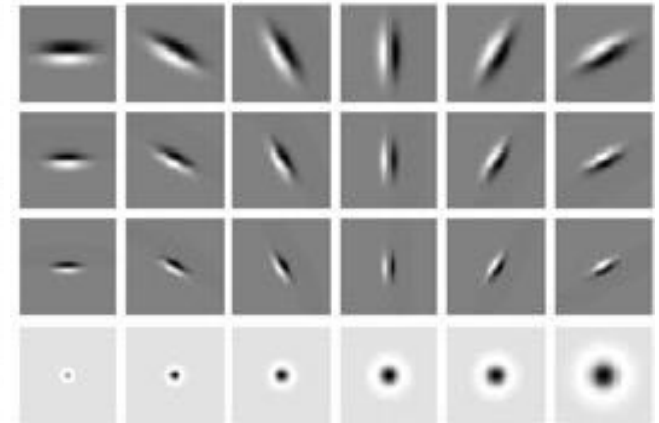
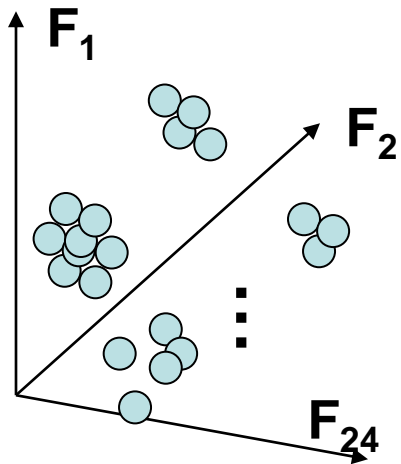
Feature Space: RGB

- Depending on what we choose as the *feature space*, we can group pixels in different ways.
- Grouping pixels based on **color** similarity
- Feature space: color value (3-dim)



Feature Space: edges and blobs

- Depending on what we choose as the *feature space*, we can group pixels in different ways.
- Grouping pixels based on **oriented gradient** similarity

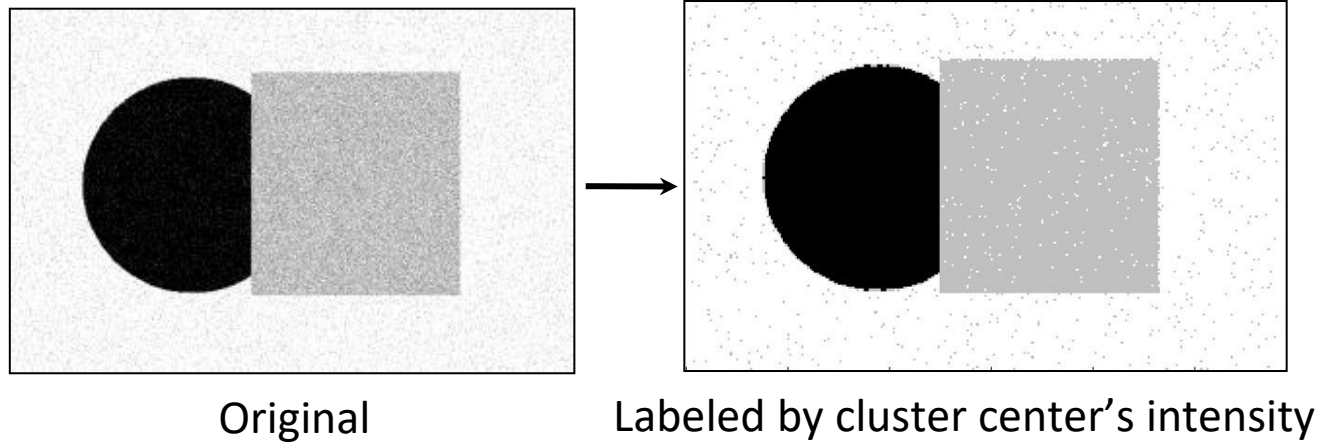


24 edge & blob filters

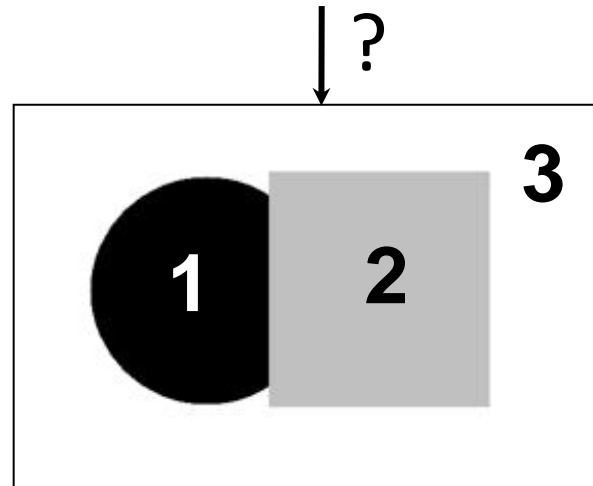
- Feature space: filter bank responses (e.g., 24D)

Smoothing Out Cluster Assignments

- Assigning a cluster label per pixel may yield outliers:

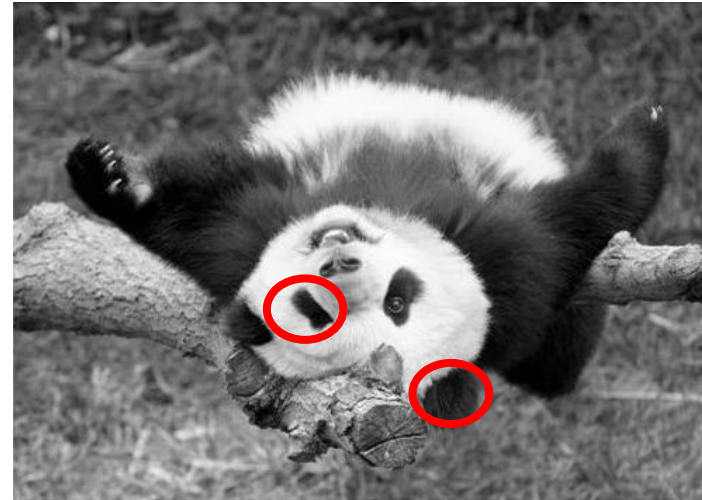
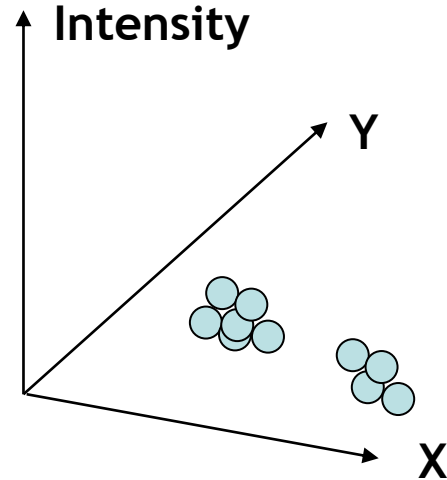


- How can we ensure they are spatially smooth?



Feature Space: RGB + XY location

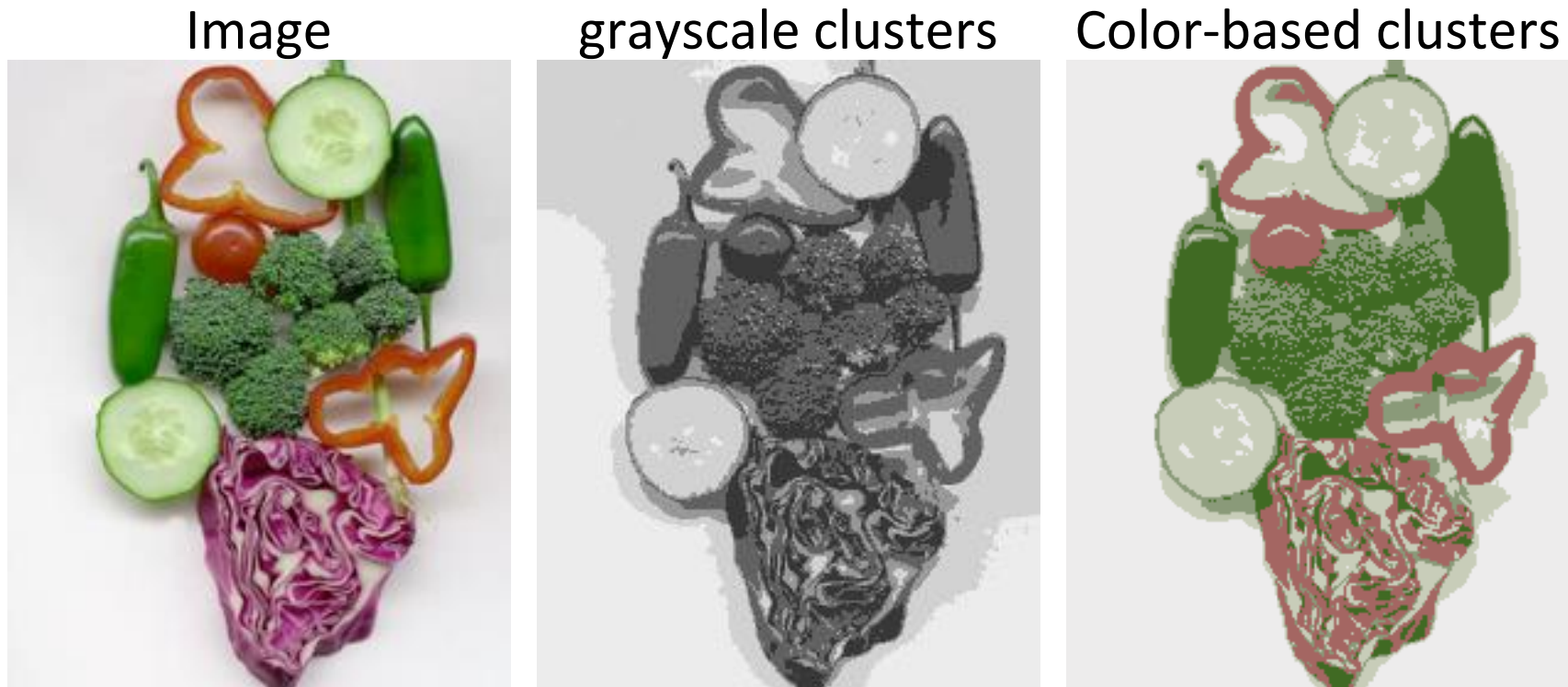
- Depending on what we choose as the *feature space*, we can group pixels in different ways.
- Grouping pixels based on *intensity+position* similarity



⇒ Way to encode both *similarity* and *proximity*.

K-Means Clustering Results

- Clusters don't have to be spatially coherent



K-Means Clustering Results

- Clustering based on (r,g,b,x,y) values enforces more spatial coherence



How to evaluate clusters?

- **Generative**

- How well are points reconstructed from the clusters?

- **Discriminative**

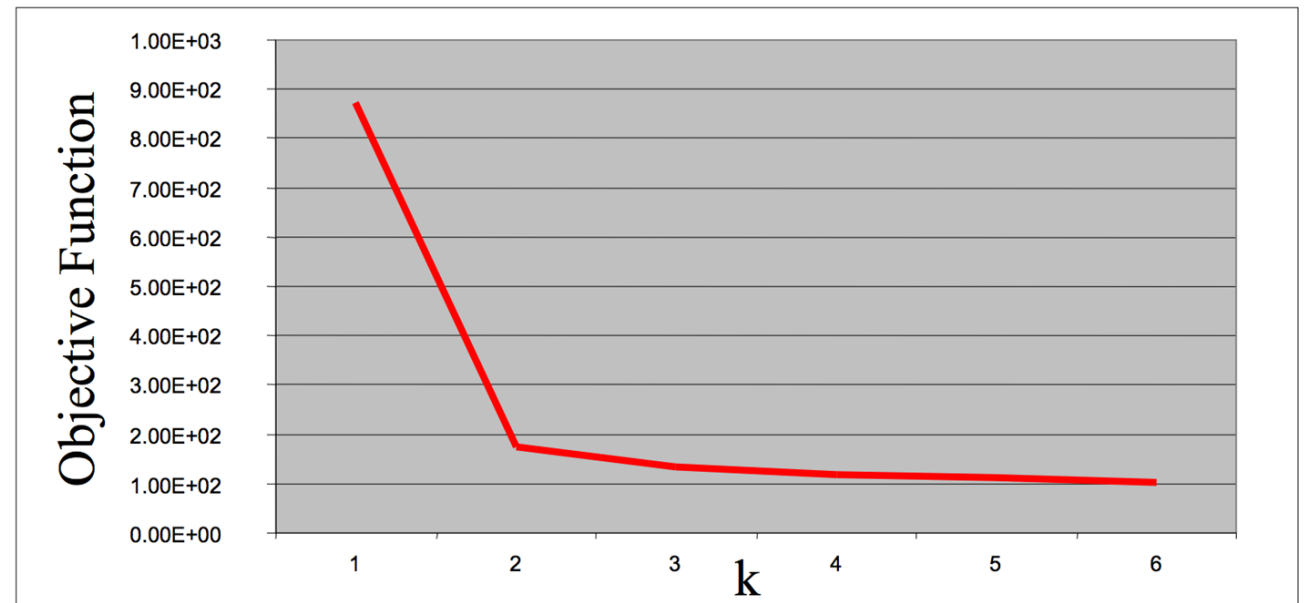
- How well do the clusters correspond to labels?
 - Can we correctly classify which pixels belong to the panda?
- Note: unsupervised clustering does not aim to be discriminative as we don't have the labels.

How to choose the number of clusters?

Try different numbers of clusters in a validation set and look at performance.

Plot of SSD versus values of k

abrupt change at $k=2$ is suggestive of two clusters in the data



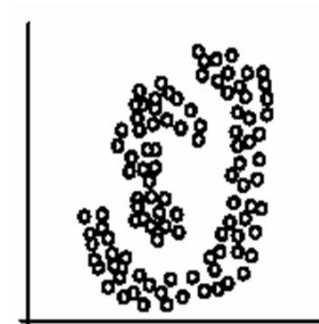
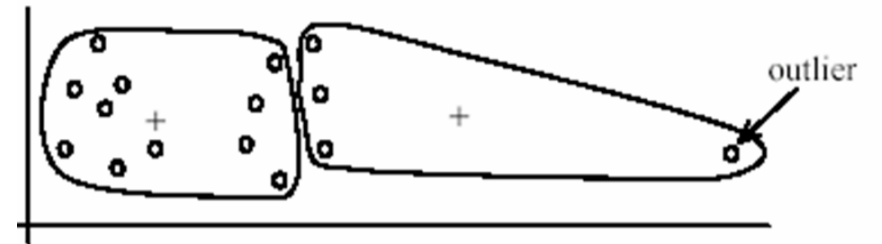
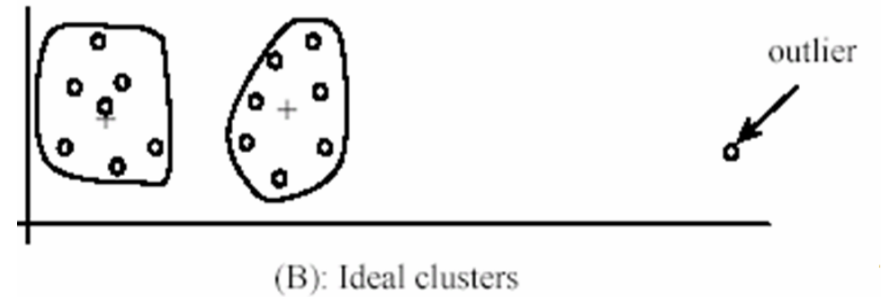
K-Means pros and cons

- **Pros**

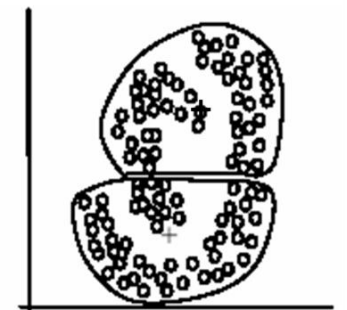
- Good representation of data
- Simple and fast, Easy to implement

- **Cons**

- Need to choose K
- Sensitive to outliers
- Prone to local minima
- All clusters have the same parameters (e.g., distance measure is non-adaptive)
- Can still be slow: each iteration is $O(KNd)$ for N d-dimensional pixels



(A): Two natural clusters



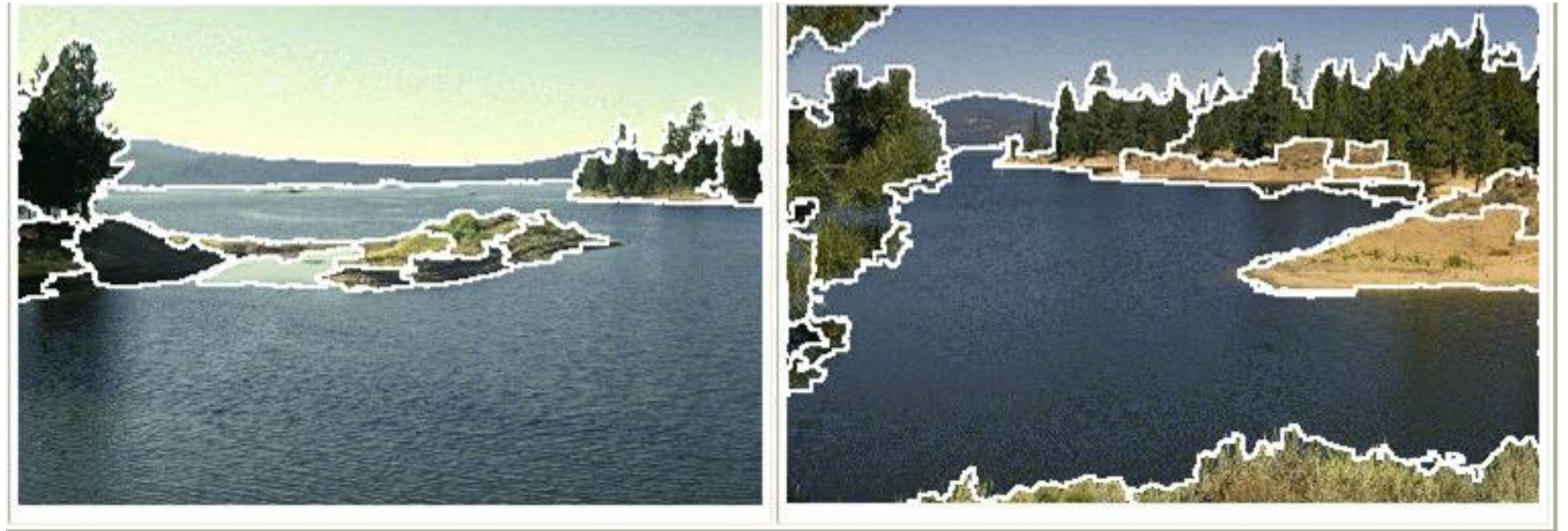
(B): k -means clusters

What will we learn today?

- K-means clustering
- Mean-shift clustering
- Normalized cuts

Mean-Shift Segmentation

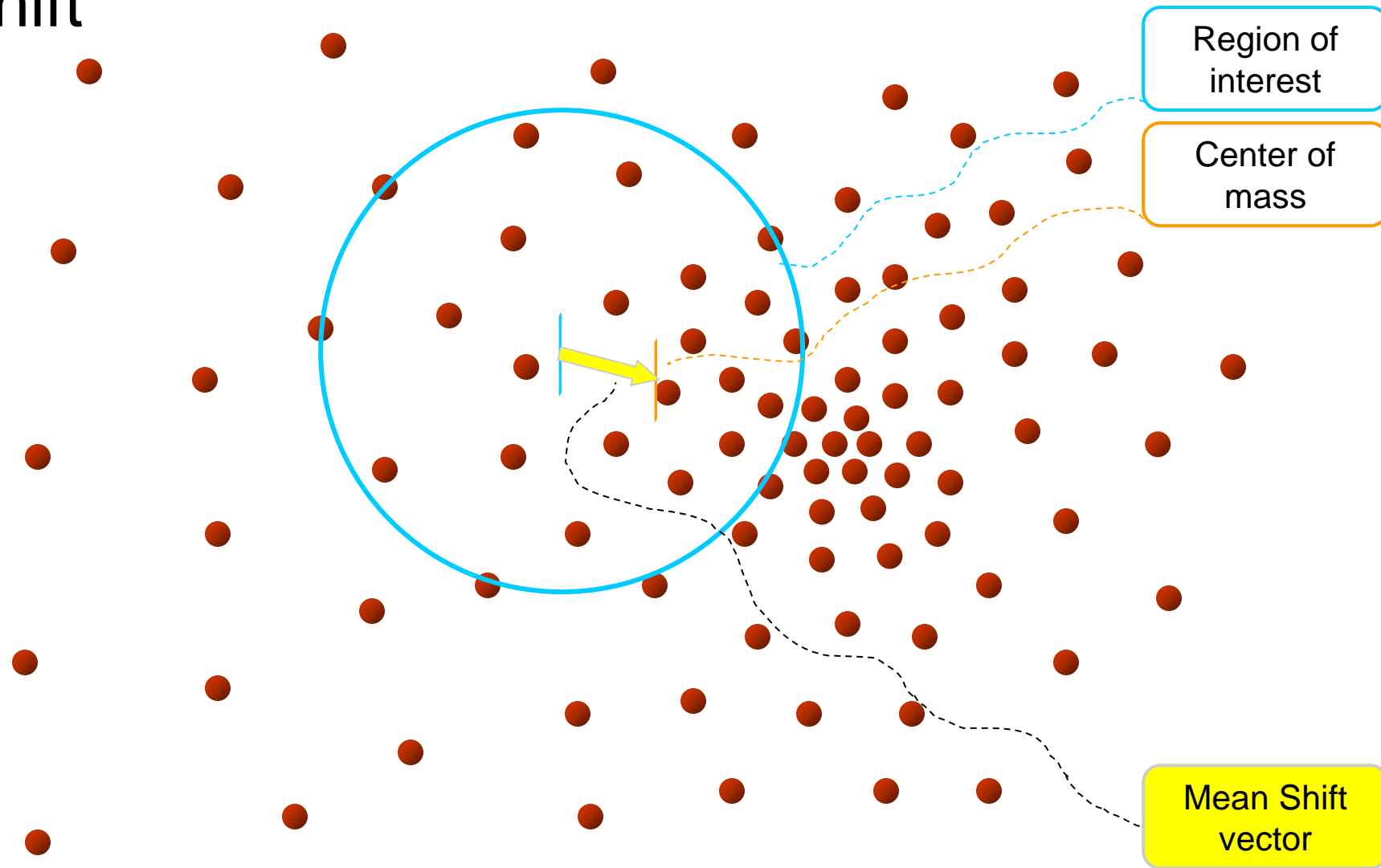
- An advanced and versatile technique for clustering-based segmentation



<http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html>

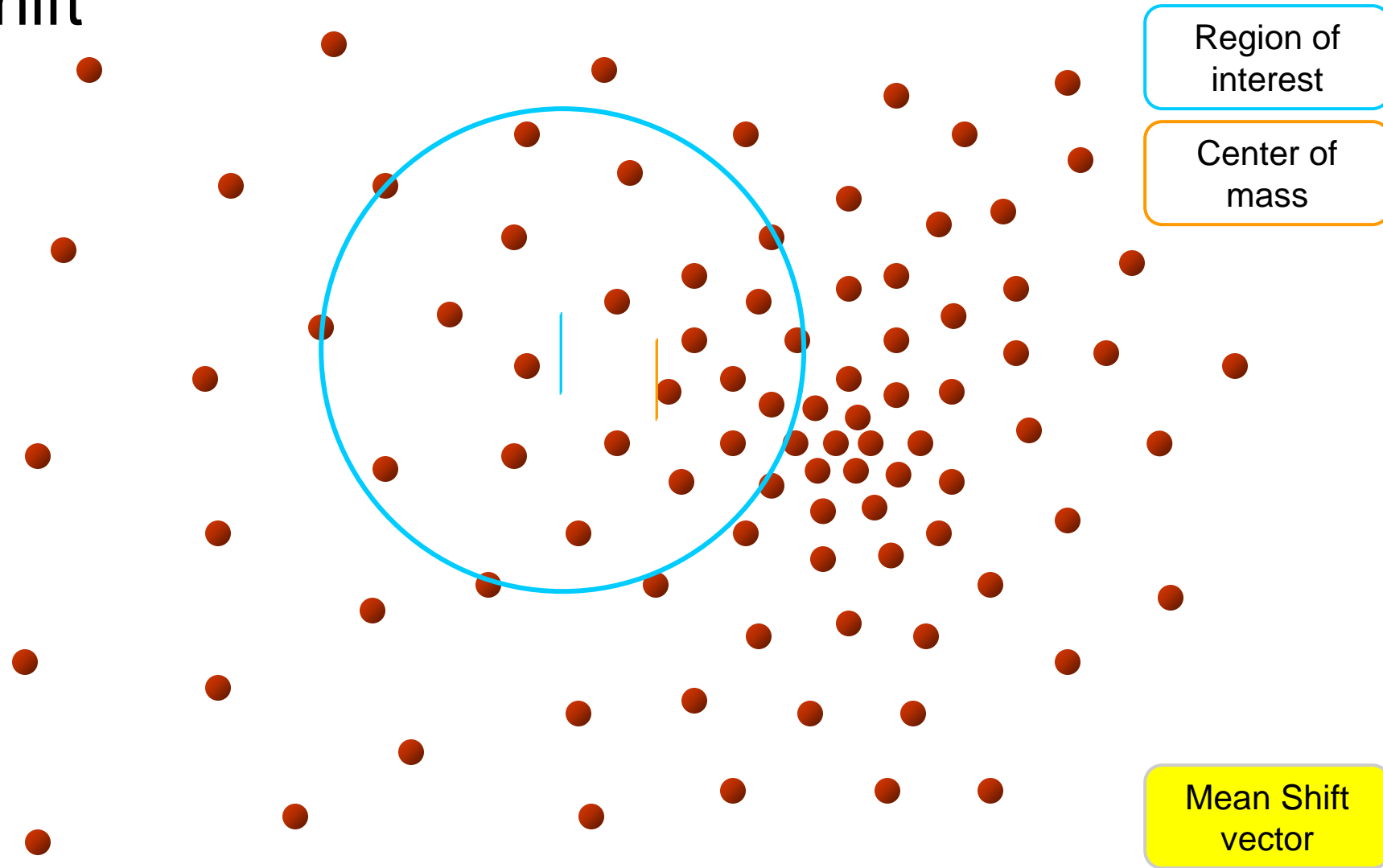
D. Comaniciu and P. Meer, Mean Shift: A Robust Approach toward Feature Space Analysis, PAMI 2002.

Mean-Shift



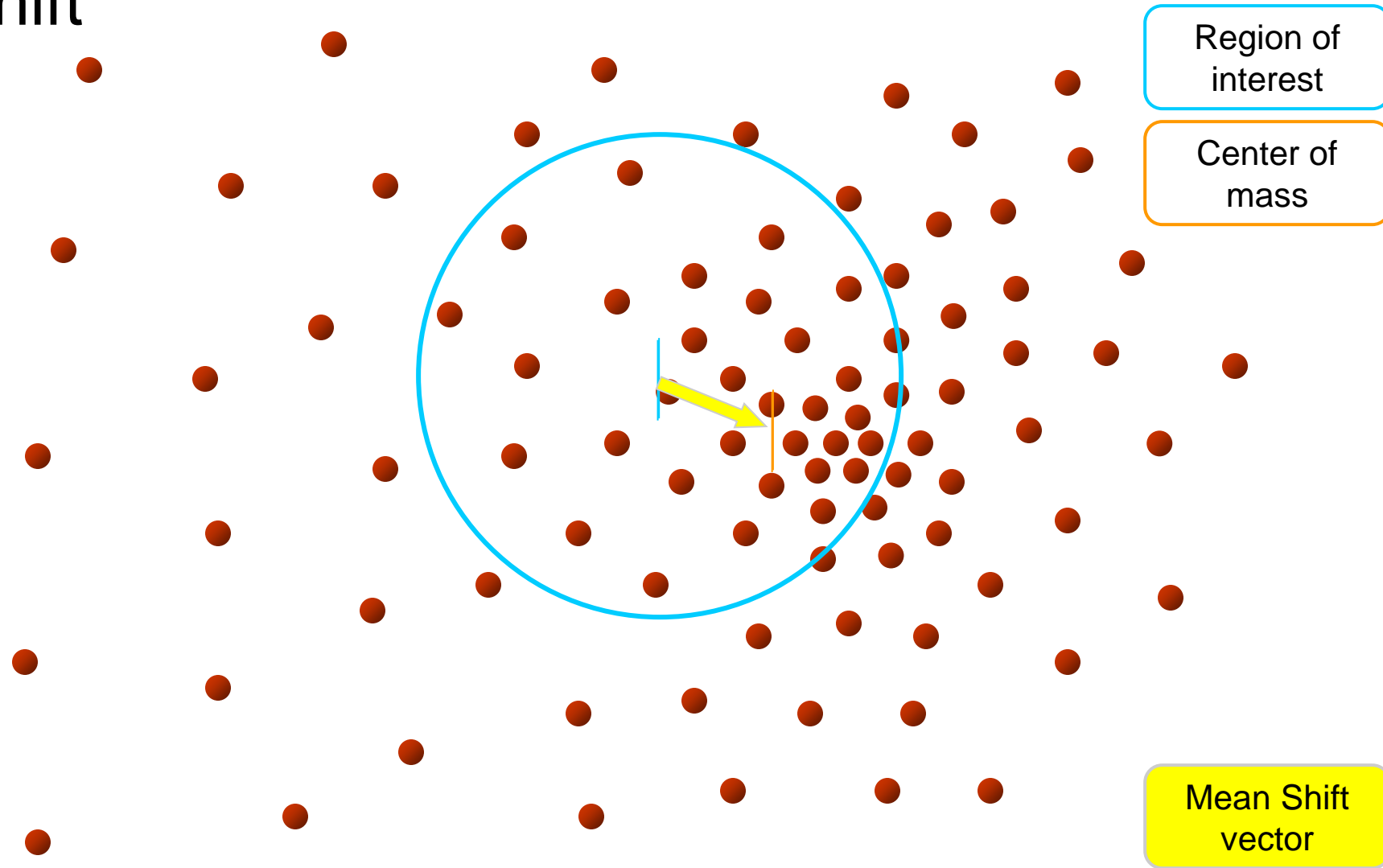
Slide by Y. Ukrainitz & B.

Mean-Shift



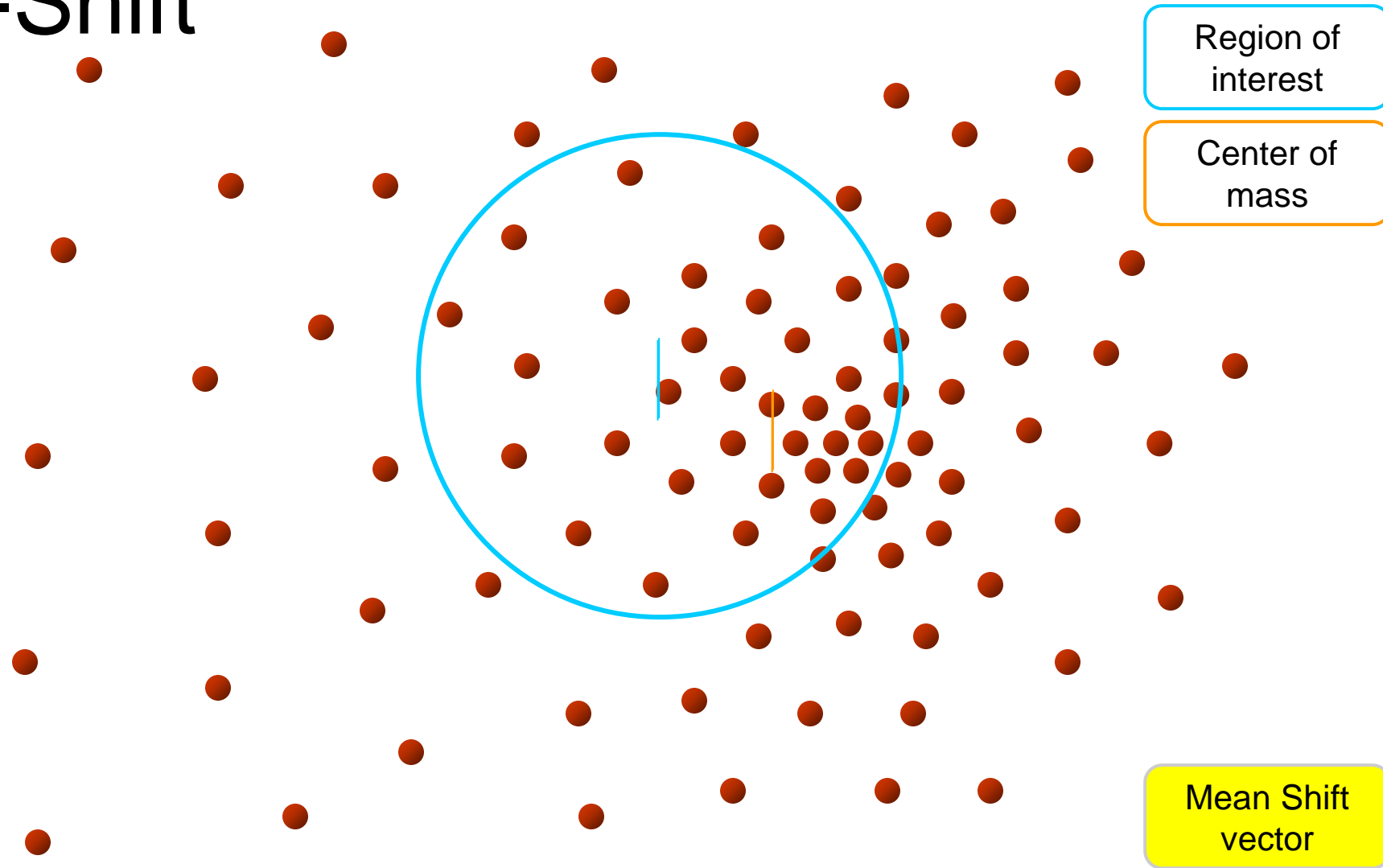
Slide by Y. Ukrainitz & B.

Mean-Shift



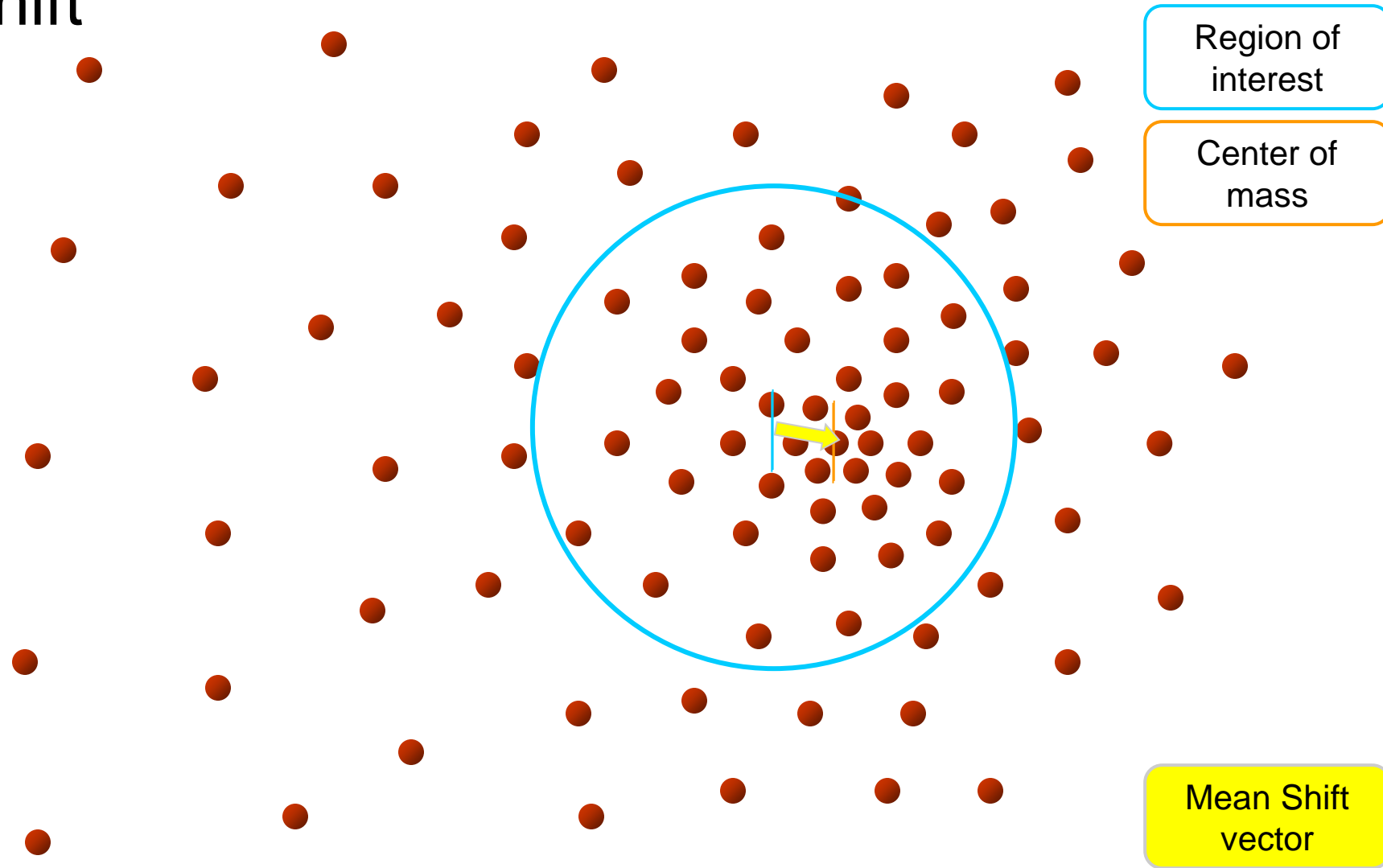
Slide by Y. Ukrainitz & B.

Mean-Shift



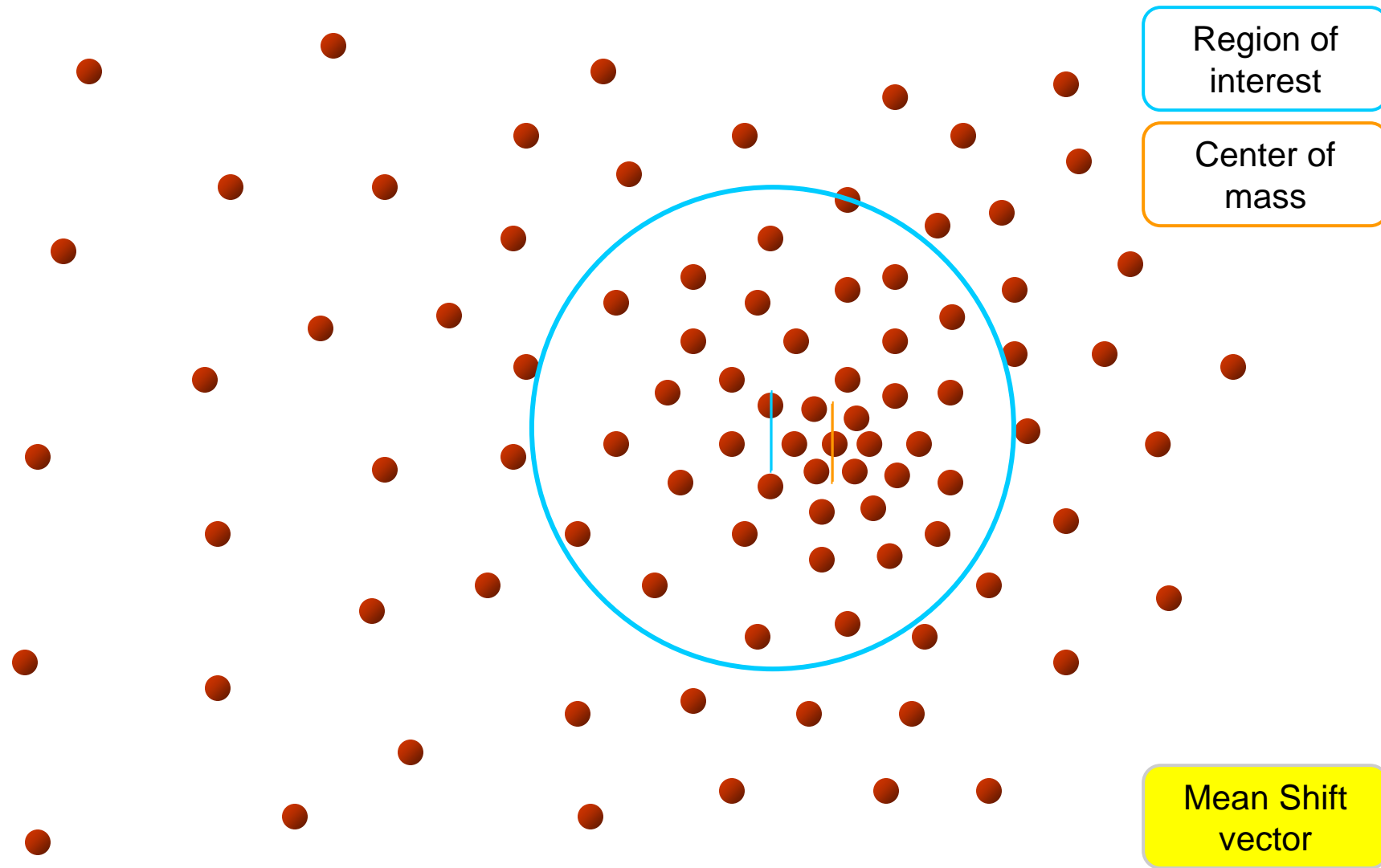
Slide by Y. Ukrainitz & B.

Mean-Shift



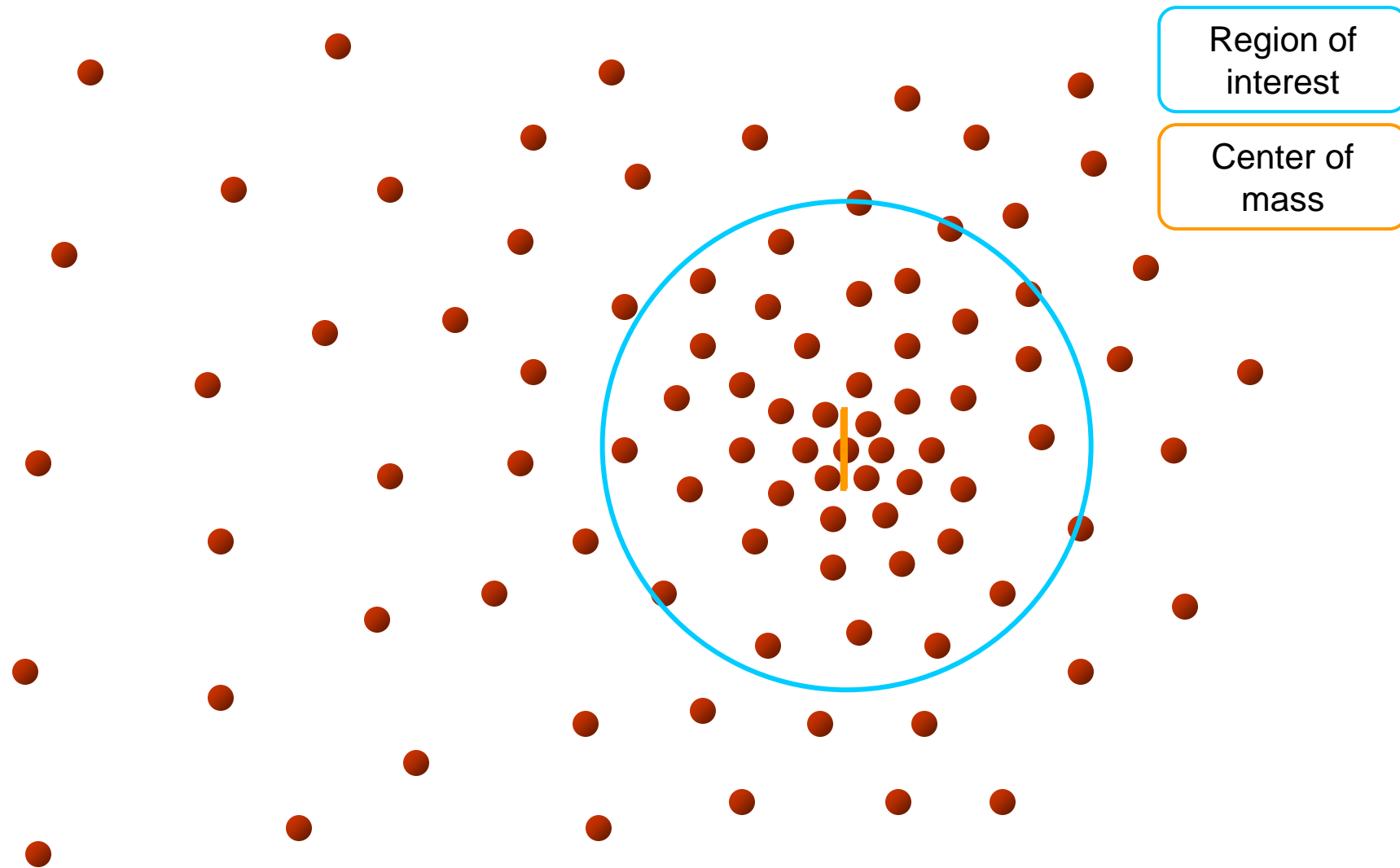
Slide by Y. Ukrainitz & B.

Mean-Shift

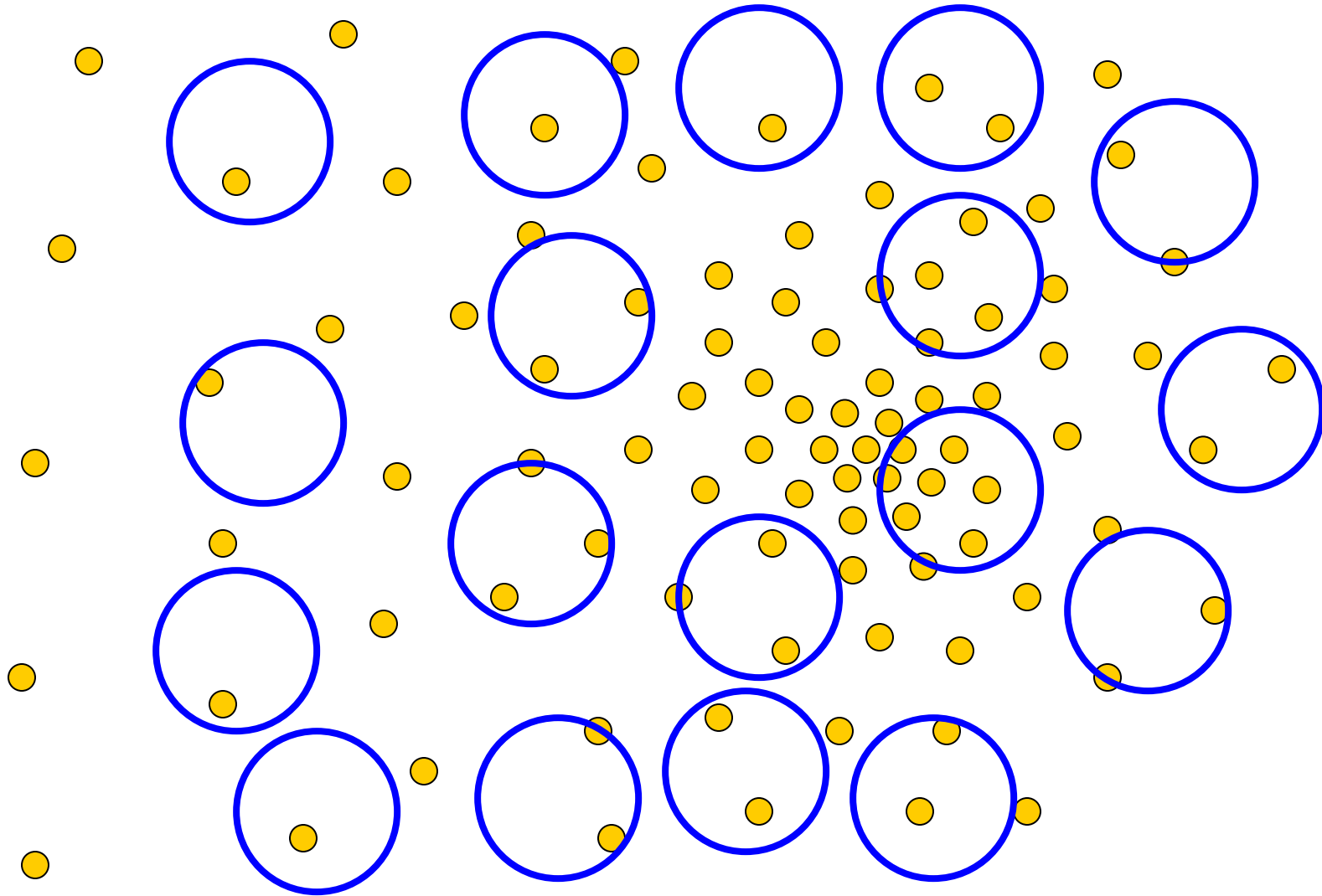


Slide by Y. Ukrainitz & B.

Mean-Shift



Real Modality Analysis

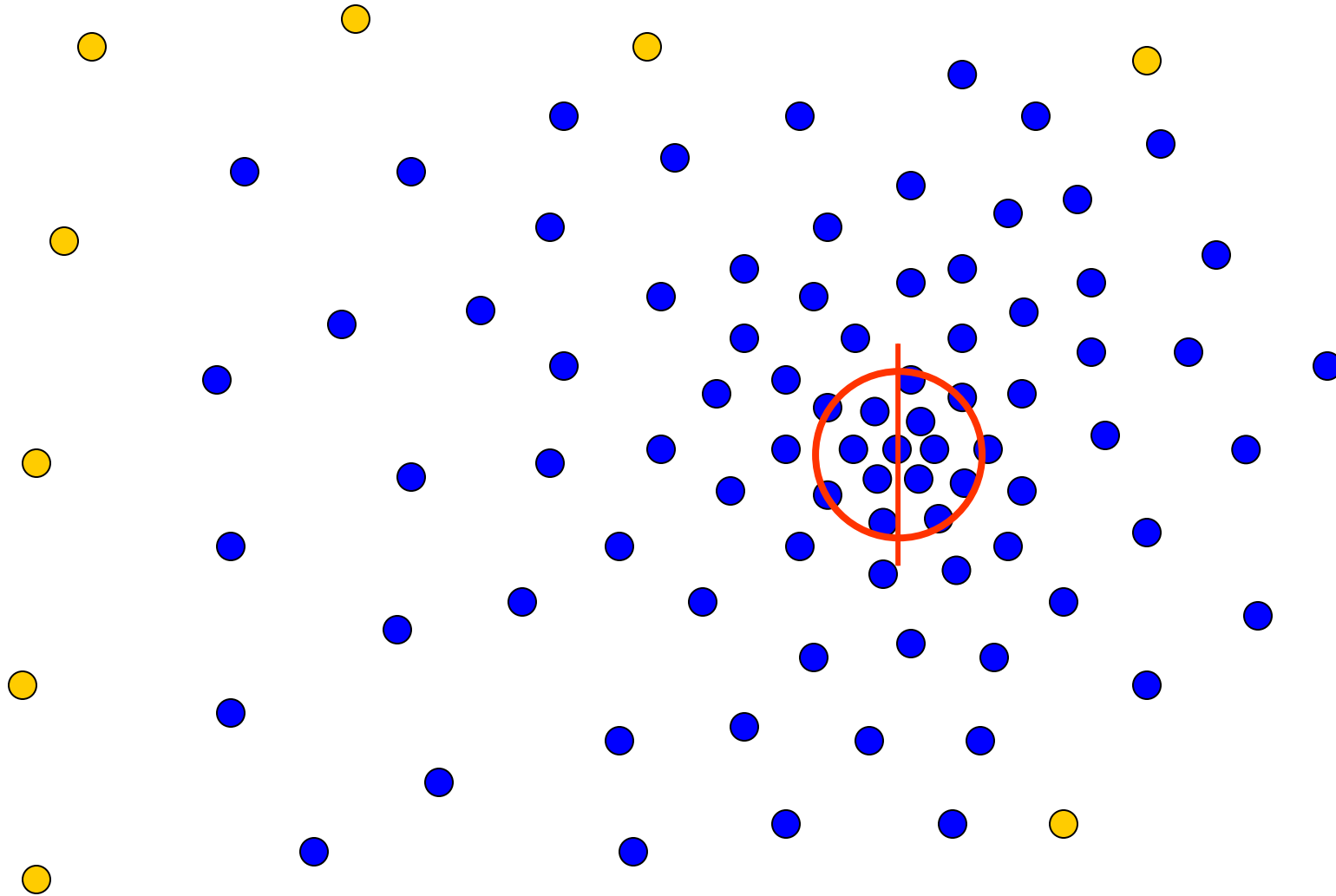


Tessellate the space with windows

Run the procedure in parallel

Slide by Y. Ukrainitz & B.
Sarel

Real Modality Analysis



The **blue** data points were traversed by the windows towards the mode.

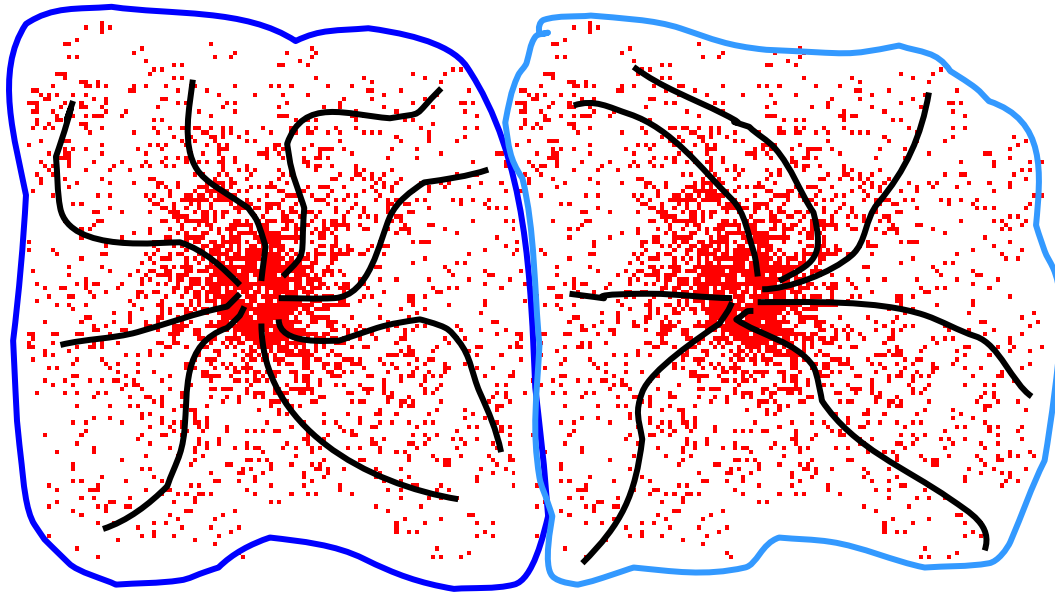
Slide by Y. Ukrainitz & B.
Sarel

Mean-Shift Algorithm

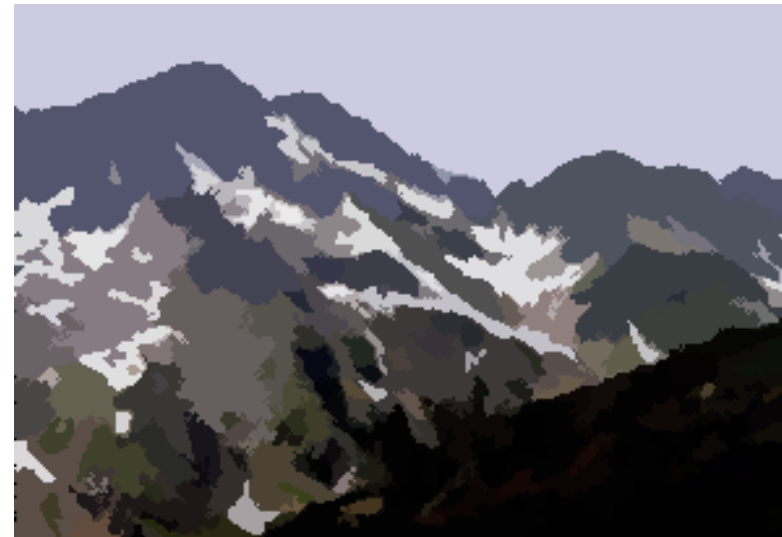
1. Represent each pixel i using some feature vector v_i
2. Generate a window \mathbf{W} as a random pixel feature v_w
3. Identify all the pixels within a radius r of v_w
4. Calculate the mean (“center of gravity”) amongst the neighbors of \mathbf{W}
5. Translate the window \mathbf{W} to the mean feature location
6. Repeat Step 2 until convergence

Mean-Shift Clustering

- Initialize not just 1 window but a multiple windows at random
- All pixels that end up in the same location belong to the same **cluster**
- **Attraction basin**: the feature region for which all windows end up in the same location



Mean-Shift Segmentation Results

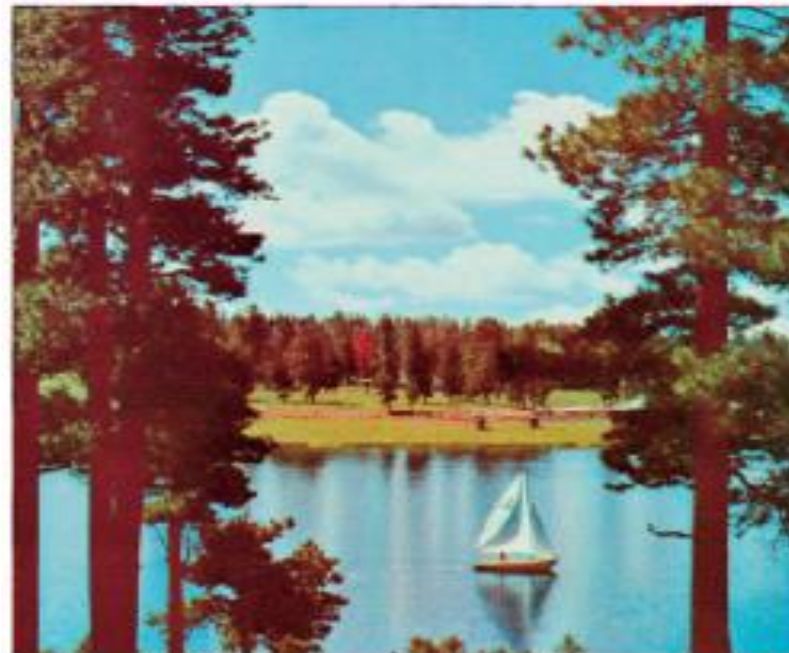


<http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html>

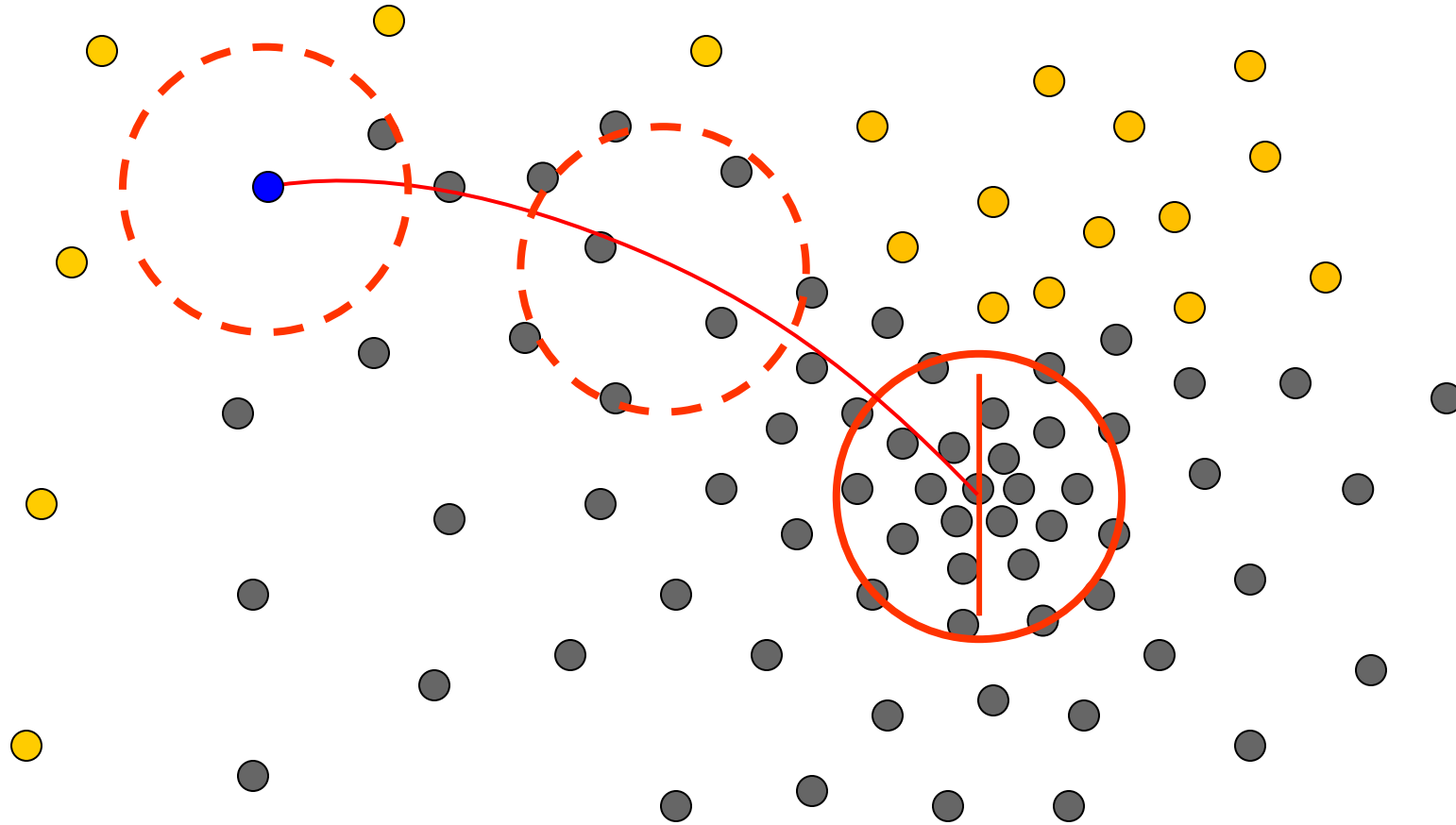
More Results



More Results

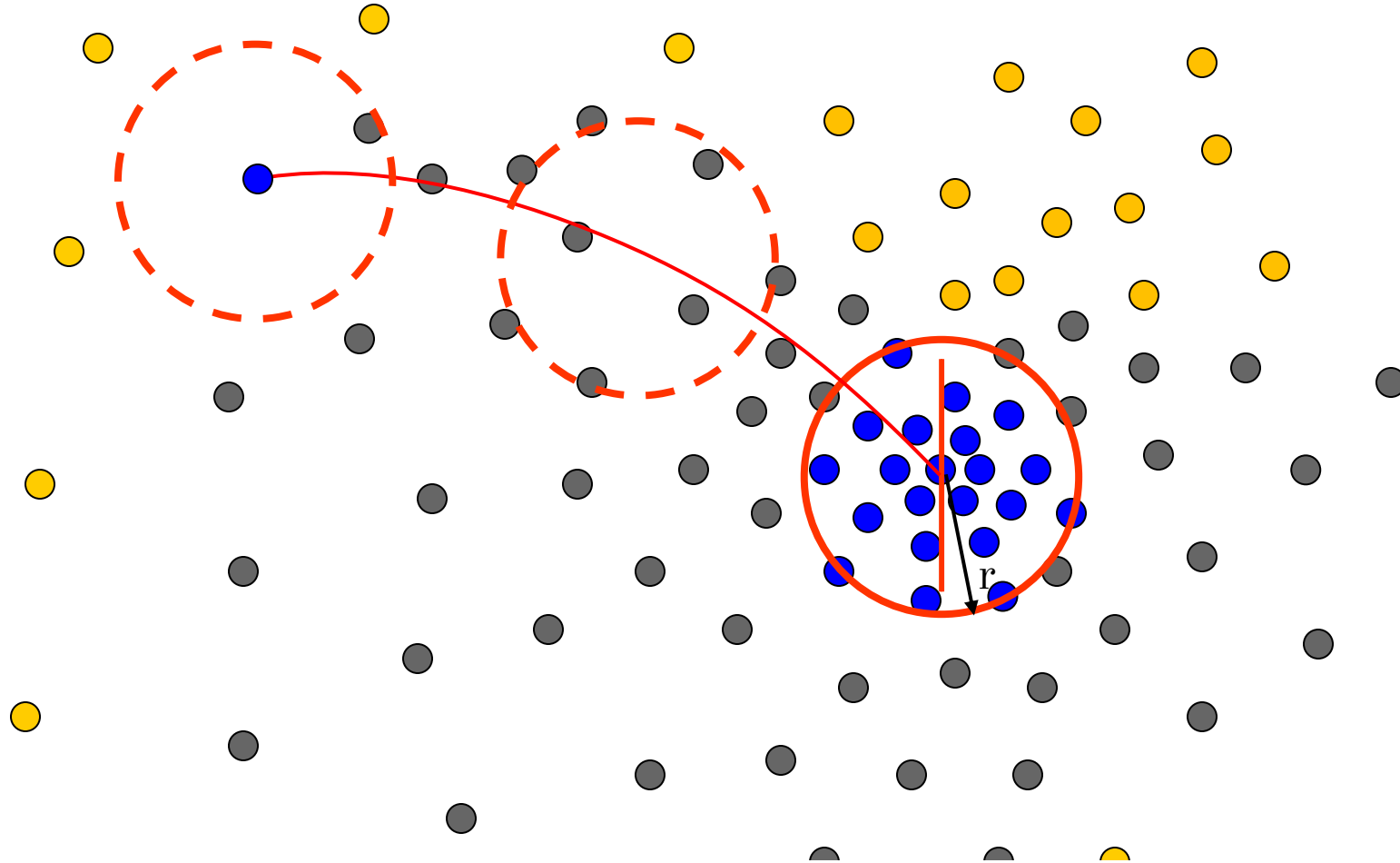


Problem: Computational Complexity



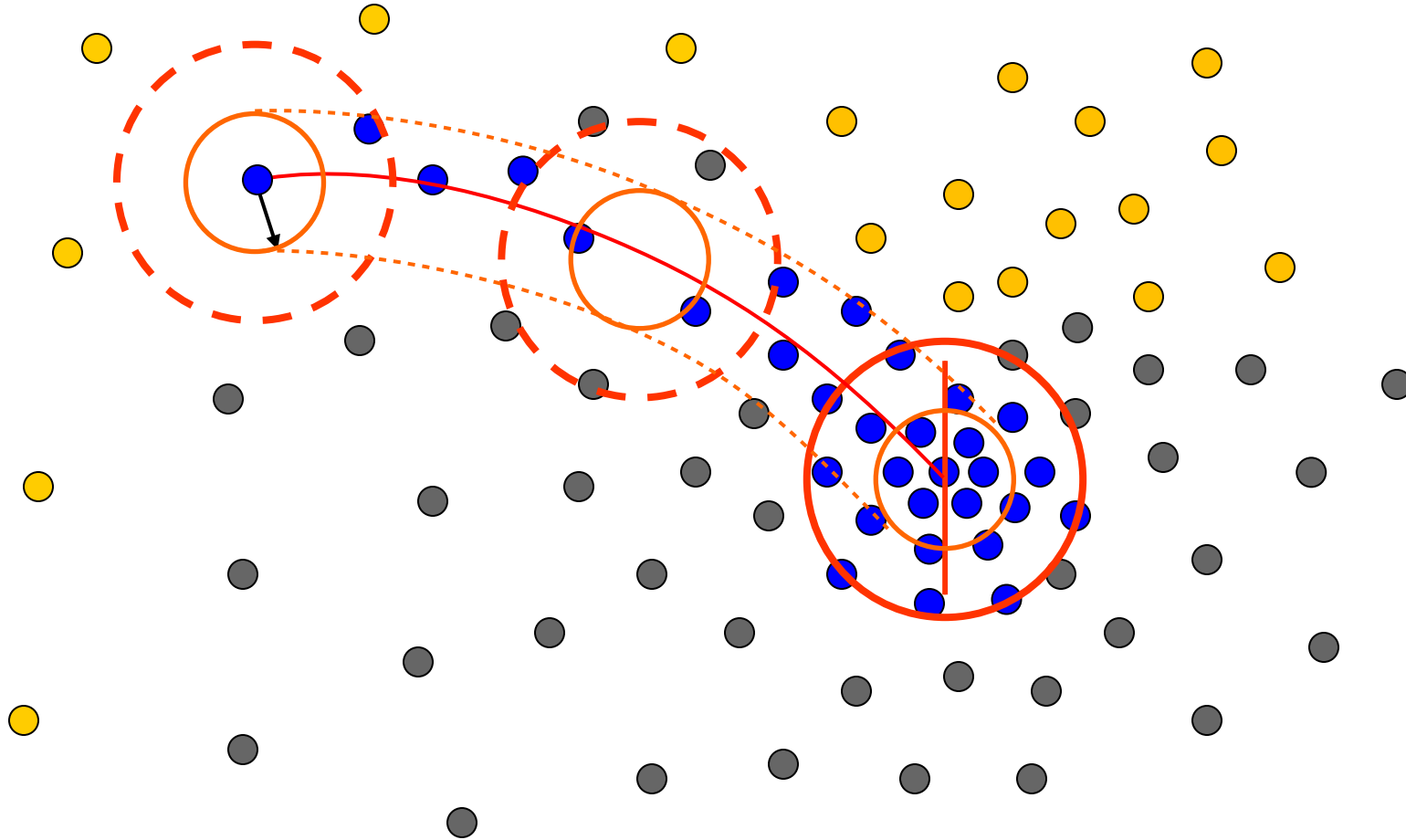
- Need to shift one window for every pixel
- Many computations will be redundant.

Speedups: Basin of Attraction



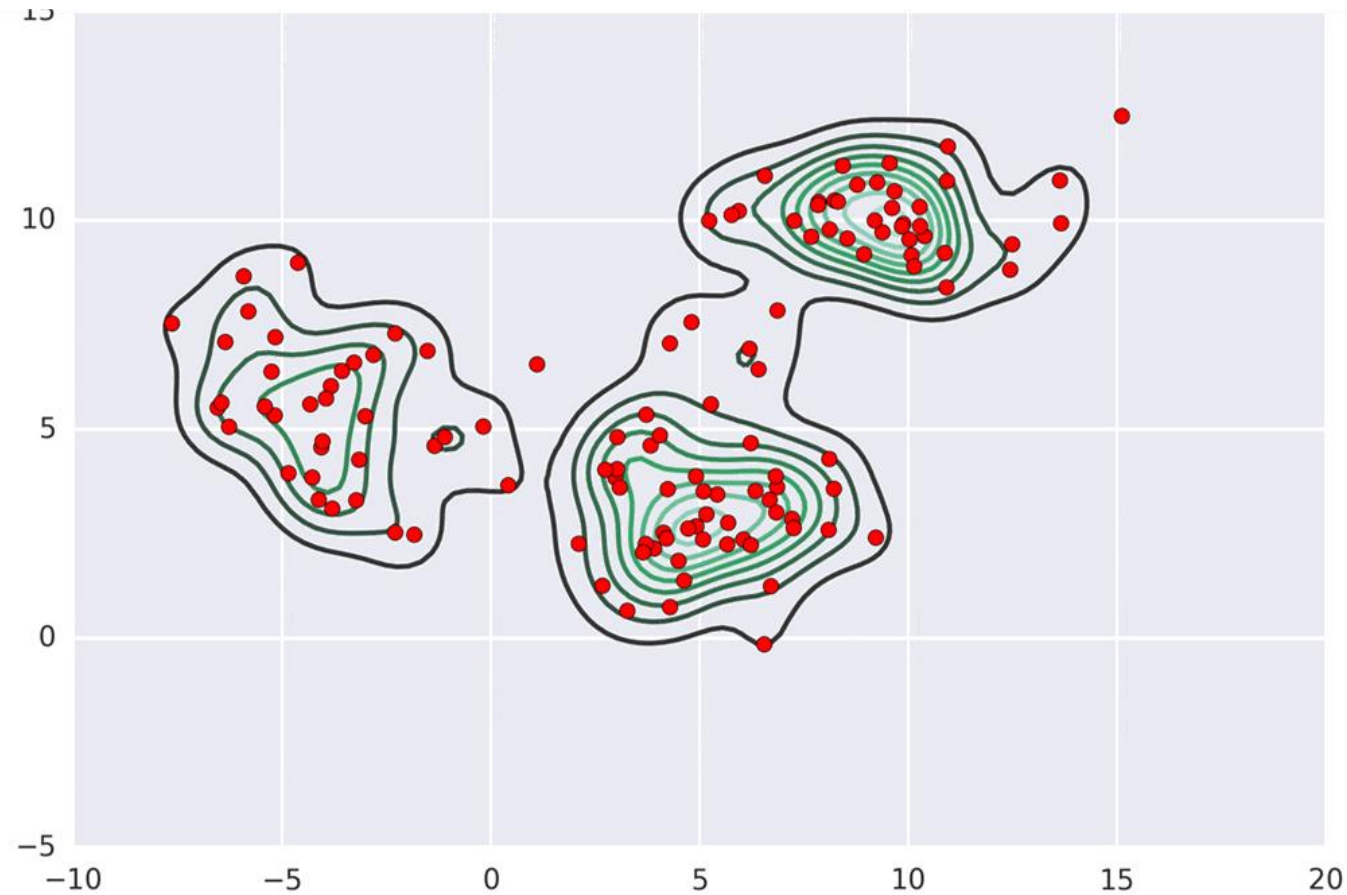
1. Assign all points within radius r of end point to the mode.

Speedups

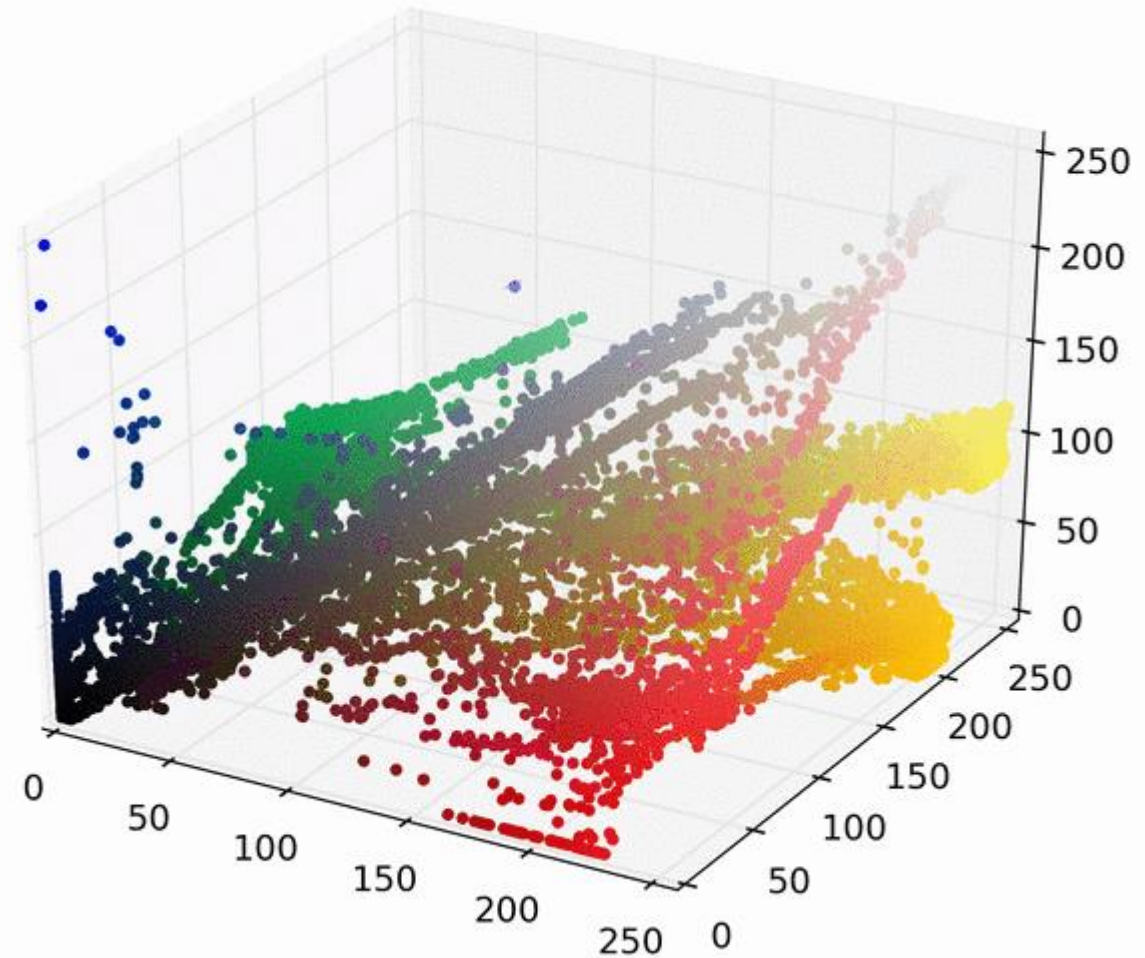


2. Assign all points within radius r/c of the search path to the mode \rightarrow reduce the number of data points to search.

Example of what running mean shift looks like

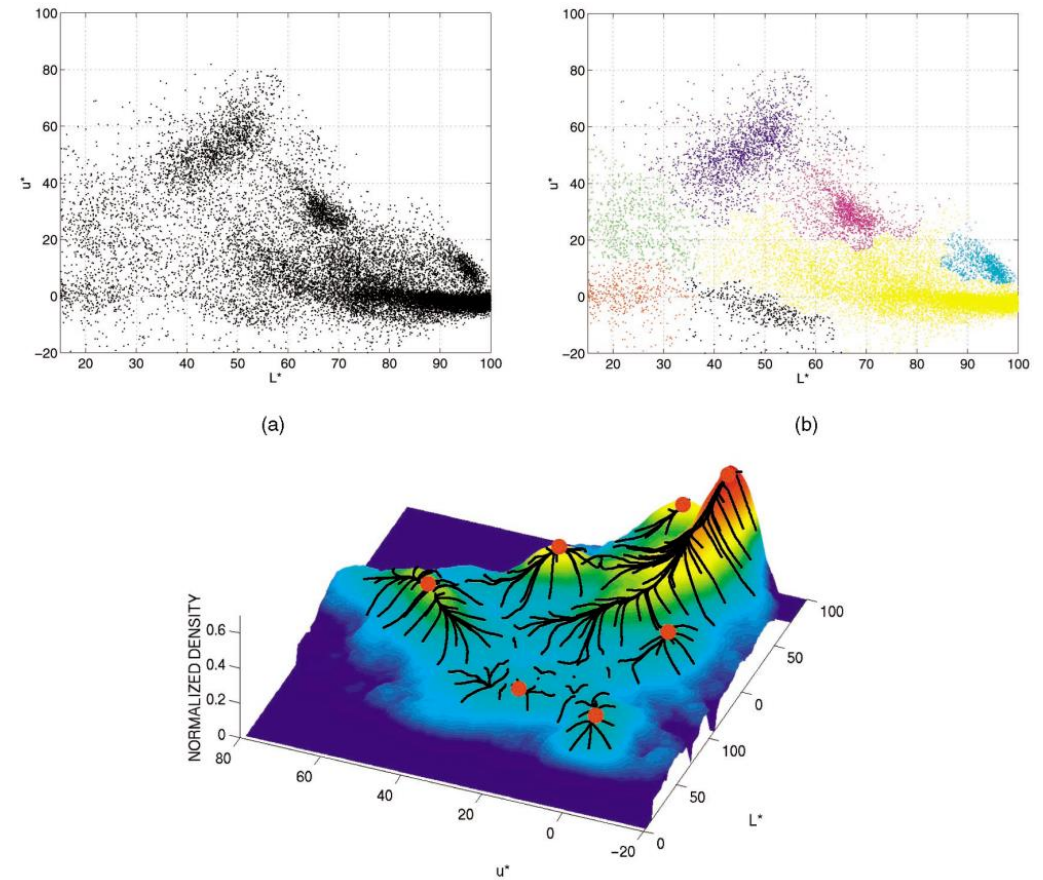


Another example



Mean-Shift Clustering

- Find features (color, gradients, texture, etc)
- Initialize windows at individual pixel locations
- Perform mean shift for each window until convergence
- At every step, merge windows that have high overlap to reduce computation



Mean-Shift pros and cons

- **Pros**

- General, application-independent algorithm
- Model-free, does not assume any prior shape (spherical, elliptical, etc.) of data clusters
- Just a single parameter (window size r)
 - r has a physical meaning (unlike k-means)
- Finds variable number of modes
- Robust to outliers

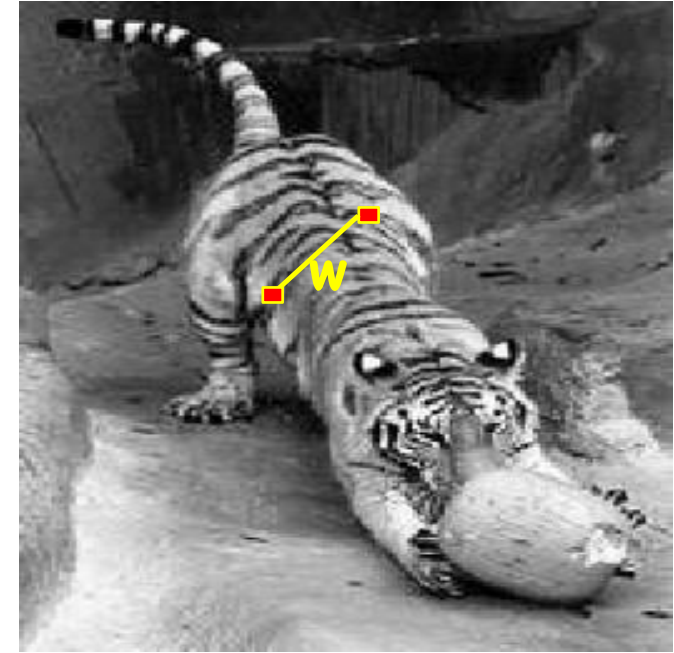
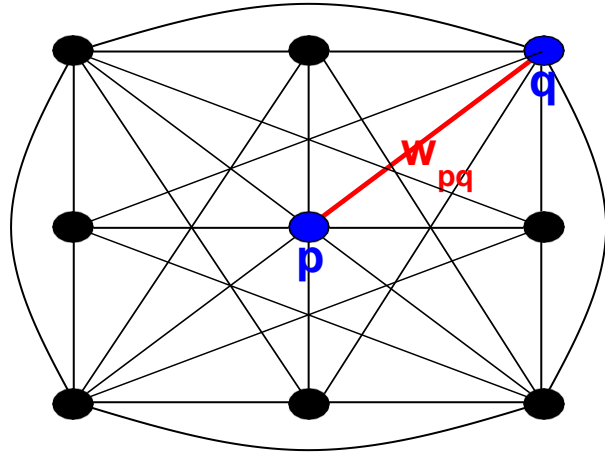
- **Cons**

- Output depends on window size
- Window size (bandwidth) selection is not easy
- Computationally (relatively) expensive (~2s/image)
- Does not scale well with dimension of feature space

Today's agenda

- K-means clustering
- Mean-shift clustering
- Normalized cuts

Images as Graphs



- Node (vertex) for every pixel
- Edge between pairs of pixels, (p,q)
- Affinity weight w_{pq} for each edge
 - w_{pq} measures similarity
 - Similarity is inversely proportional to difference (in color and position...)

Images as Graphs

Which edges to include?

Fully connected:

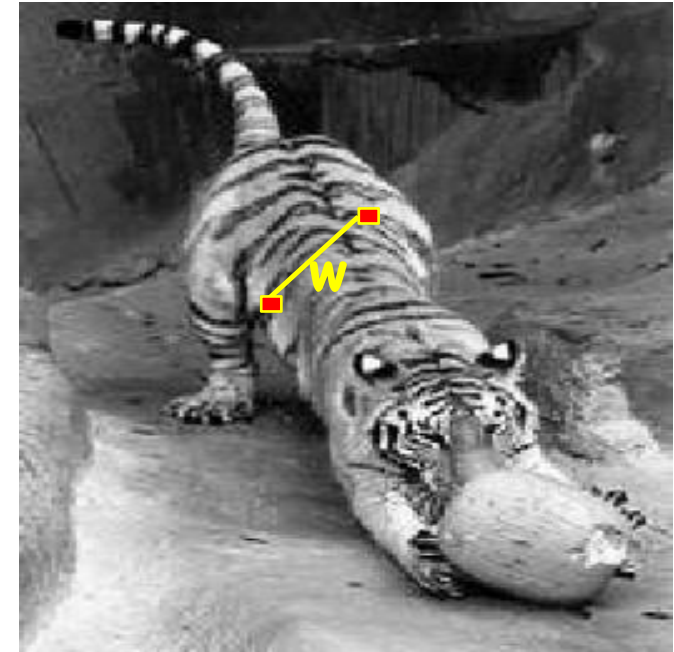
- Captures all pairwise similarities
- Infeasible for most images

Neighboring pixels:

- Very fast to compute
- Only captures very local interactions

Local neighborhood:

- Reasonably fast, graph still very sparse
- Good tradeoff



Measuring Affinity

- Distance: $aff(x, y) = \exp \left(-\frac{1}{2\sigma_d^2} \|f(x) - f(y)\|^2 \right)$
- Examples:
 - Distance: $f(x) = location(x)$
 - Distance: $f(x) = intensity(x)$
 - Intensity: $f(x) = color(x)$
 - Color: $f(x) = filterbank(x)$
 - Texture:

Measuring Affinity

Distance:

$$f(x) = \text{location}(x)$$



Measuring Affinity

Intensity:

$$f(x) = \textit{intensity}(x)$$



Measuring Affinity

Color:

$$f(x) = color(x)$$



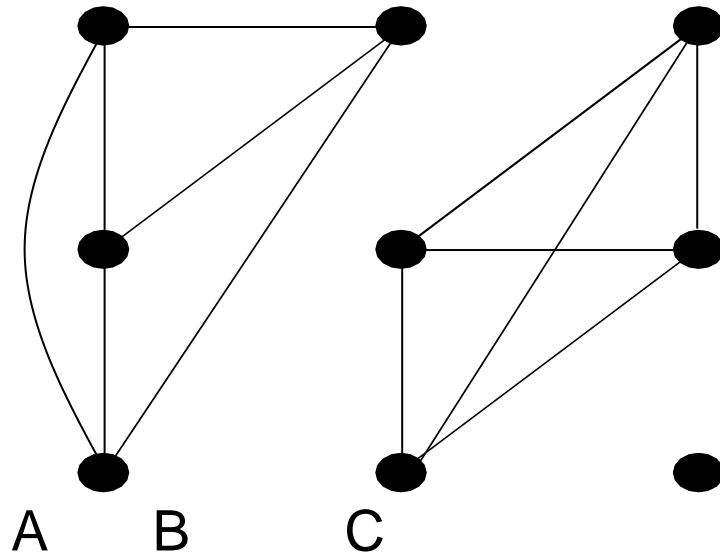
Measuring Affinity

Texture:

$$f(x) = \text{filterbank}(x)$$



Segmentation as Graph Cuts

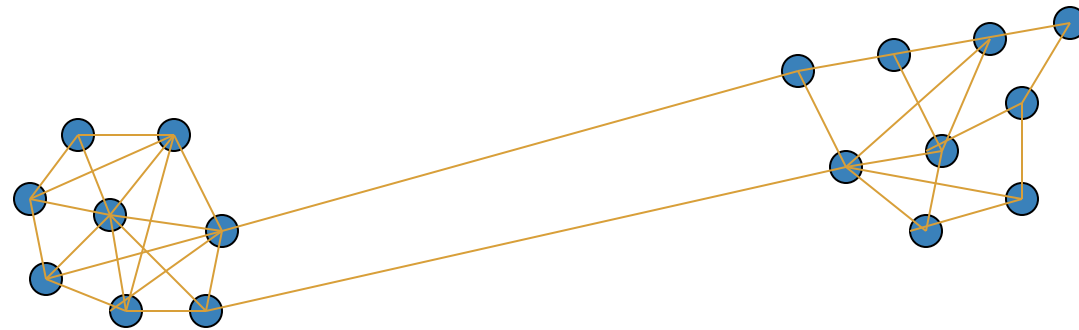


Break Graph into Segments

- Delete links that cross between segments
- Easiest to break links that have low similarity (low weight)
 - Similar pixels should be in the same segments
 - Dissimilar pixels should be in different segments

● Jianbo Shi's Graph-Partitioning

- An image is represented by a graph whose nodes are pixels or small groups of pixels.
- The goal is to partition the vertices into disjoint sets so that the similarity within each set is high and across different sets is low.

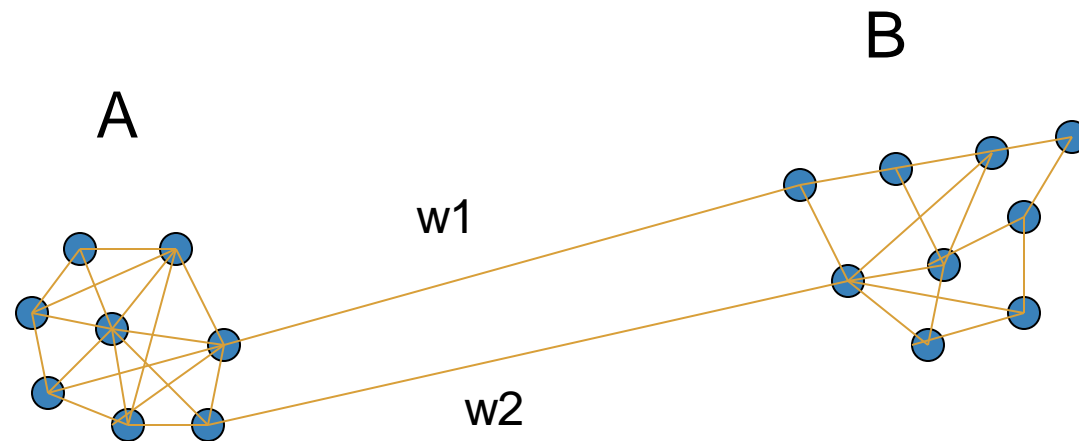


● Minimal Cuts

- Let $G = (V, E)$ be a graph. Each edge (u, v) has a weight $w(u, v)$ that represents the similarity between u and v .
- Graph G can be broken into 2 disjoint graphs with node sets A and B by removing edges that connect these sets.
- Let $\text{cut}(A, B) = \sum_{u \in A, v \in B} w(u, v)$.
- One way to segment G is to find the minimal cut.

- Cut(A,B)

$$\text{cut}(A,B) = \sum_{u \in A, v \in B} w(u,v)$$



● Normalized Cut

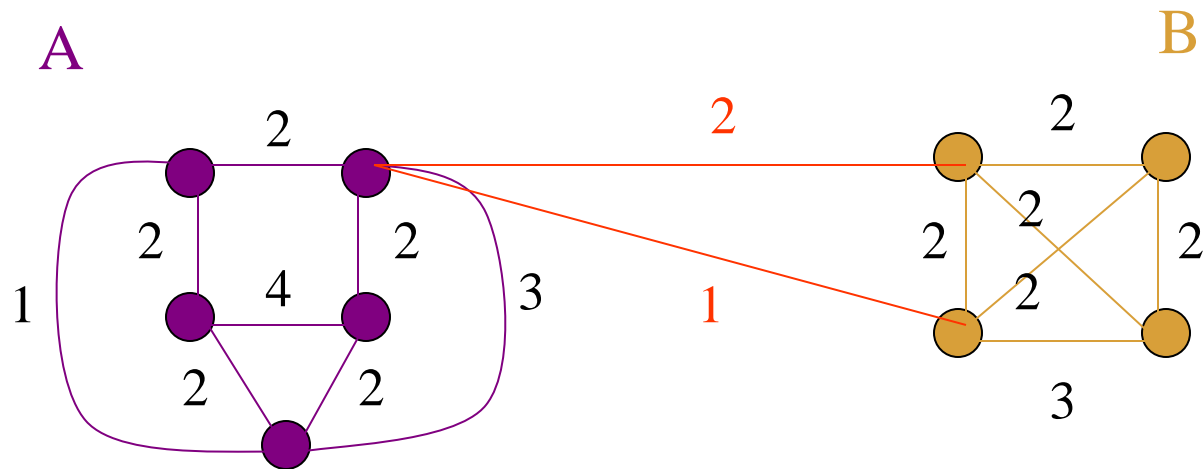
Minimal cut favors cutting off small node groups,
so Shi proposed the **normalized cut**.

$$Ncut(A,B) = \frac{cut(A, B)}{asso(A,V)} + \frac{cut(A,B)}{asso(B,V)} \quad \text{normalized cut}$$

$$asso(A,V) = \sum_{u \in A, t \in V} w(u,t)$$

How much is A connected
to the graph as a whole.

● Example Normalized Cut



$$\text{Ncut(A,B)} = \frac{3}{21} + \frac{3}{16}$$

● Shi turned graph cuts into an eigenvector/eigenvalue problem.

- Set up a weighted graph $G=(V,E)$
 - V is the set of (N) pixels
 - E is a set of weighted edges (weight w_{ij} gives the similarity between nodes i and j)
 - Length N vector d : d_i is the sum of the weights from node i to all other nodes
 - $N \times N$ matrix D : D is a diagonal matrix with d on its diagonal
 - $N \times N$ symmetric matrix W : $W_{ij} = w_{ij}$

Let \mathbf{x} be a characteristic vector of a set A of nodes

$x_i = 1$ if node i is in a set A

$x_i = -1$ otherwise

Let \mathbf{y} be a continuous approximation to \mathbf{x}

Solve the system of equations

$$(D - W) \mathbf{y} = \lambda D \mathbf{y}$$

for the eigenvectors \mathbf{y} and eigenvalues λ

Use the eigenvector \mathbf{y} with second smallest eigenvalue to bipartition the graph ($\mathbf{y} \Rightarrow \mathbf{x} \Rightarrow A$)

If further subdivision is merited, repeat recursively

- How Shi used the procedure

Shi defined the edge weights $w(i,j)$ by

$$w(i,j) = e^{-\|F(i)-F(j)\|_2 / \sigma_I} * \begin{cases} e^{-\|X(i)-X(j)\|_2 / \sigma_X} & \text{if } \|X(i)-X(j)\|_2 < r \\ 0 & \text{otherwise} \end{cases}$$

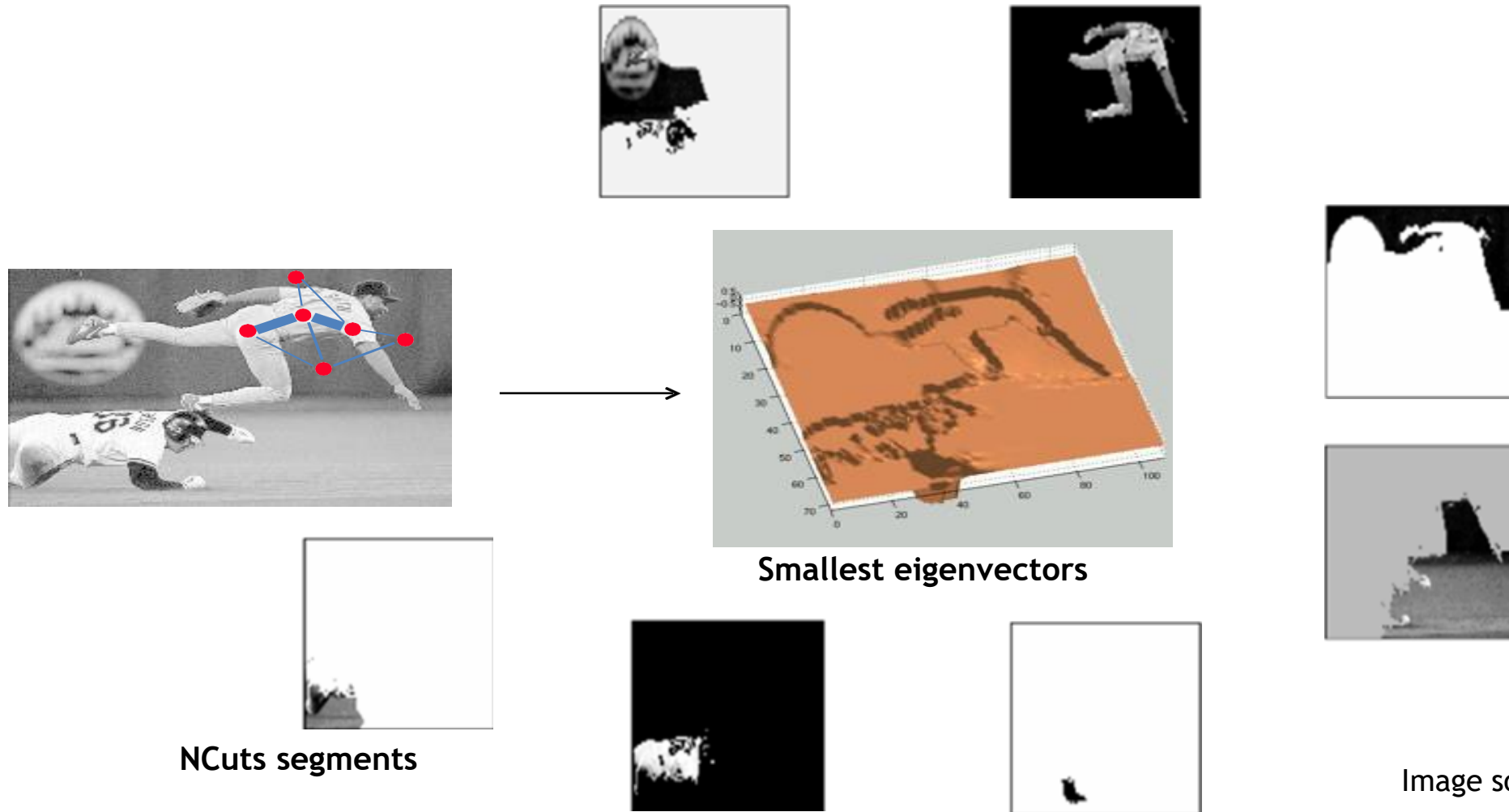
where $X(i)$ is the spatial location of node i

$F(i)$ is the feature vector for node i

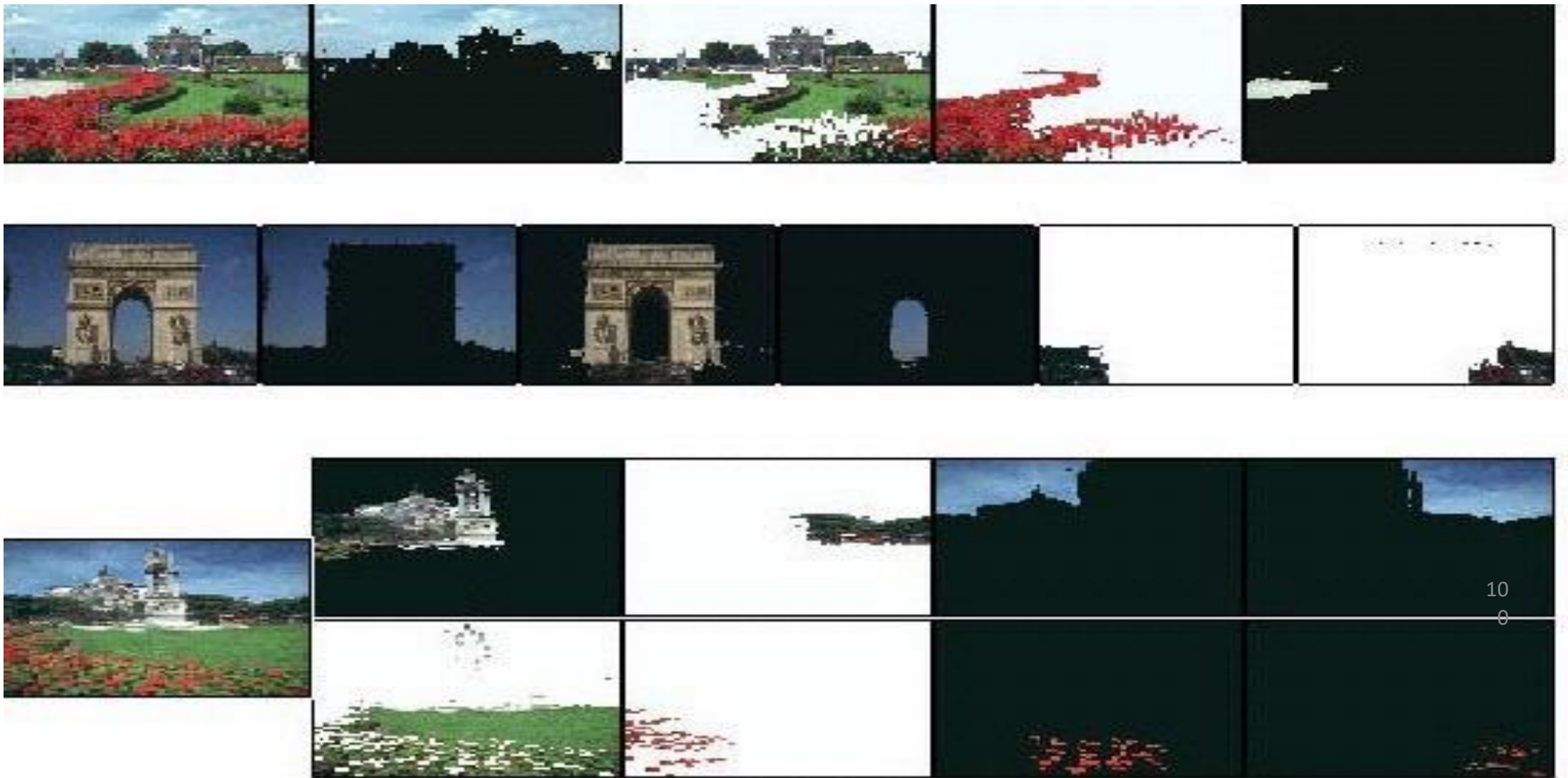
which can be intensity, color, texture, motion...

The formula is set up so that $w(i,j)$ is 0 for nodes that are too far apart.

Normalized Cuts example



Normalized Cuts example



Normalized Cuts example



Normalized Cuts summary

- Pro
 - Flexible to choice of affinity matrix
 - Generally works better than other methods we've seen so far
- Con
 - Can be expensive, especially with many cuts.
 - Bias toward balanced partitions
 - Constrained by affinity matrix model



Today's agenda

- K-means clustering
- Mean-shift clustering
- Normalized cuts

Next time

Recognition