



# Lecture 2: Introduction to Segmentation

Jonathan Krause

# Goal

- Goal: Identify groups of pixels that go together

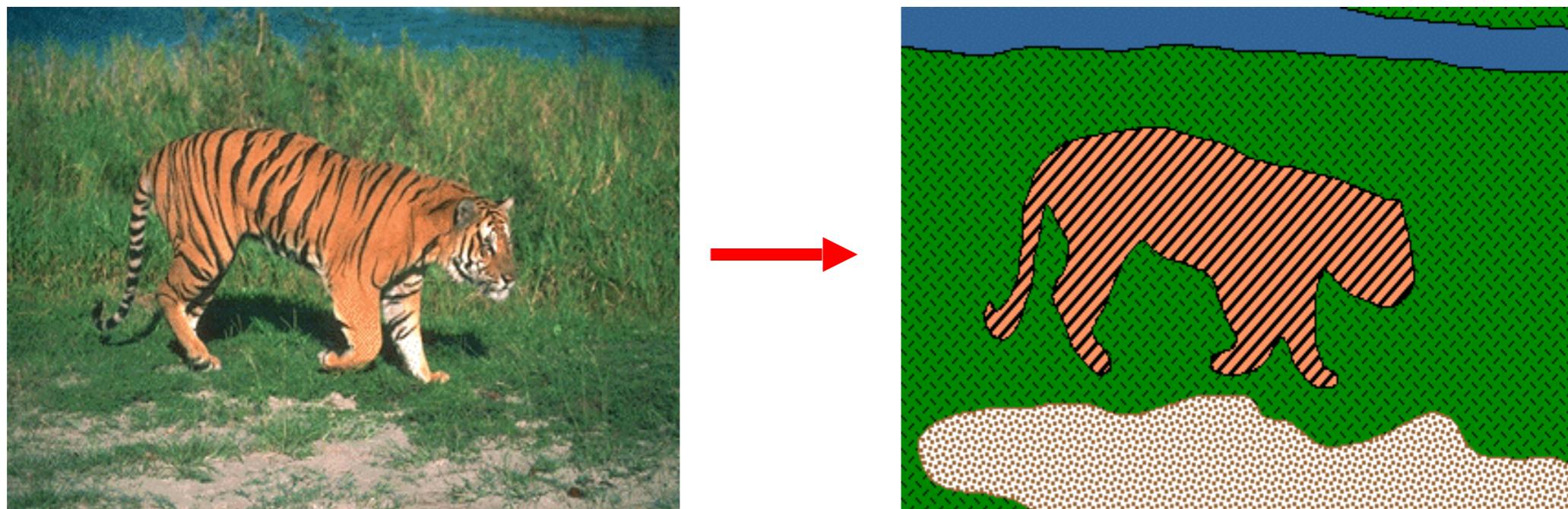


image credit: Steve Seitz, Kristen Grauman

# Types of Segmentation

- Semantic Segmentation: Assign labels

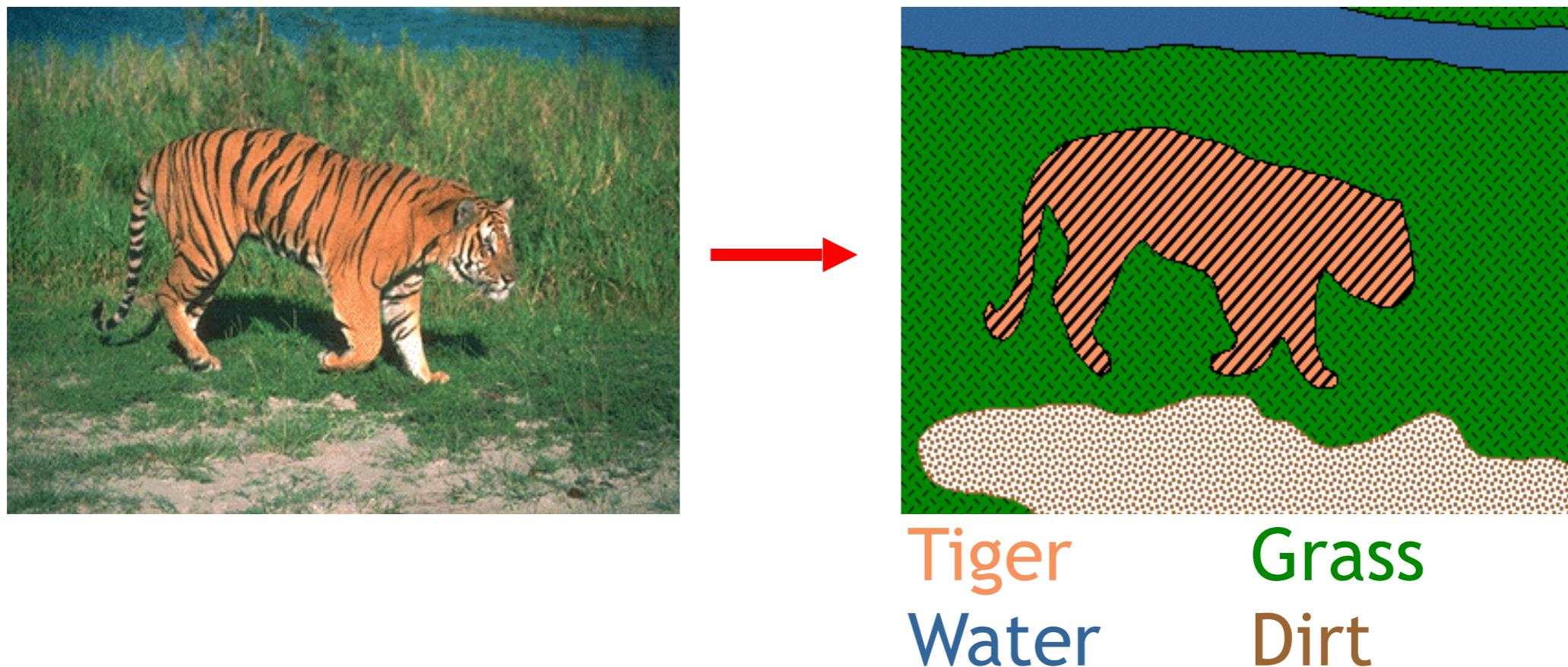


image credit: Steve Seitz, Kristen Grauman

# Types of Segmentation

- Figure-ground segmentation: Foreground/background



image credit: Carsten Rother

# Types of Segmentation

- Co-segmentation: Segment common object



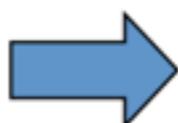
image credit: Armand Joulin

# Application: As a result

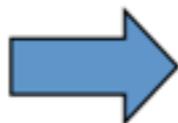


Rother et al. 2004

# Application: Speed up Recognition



[Felzenszwalb and Huttenlocher 2004]



[Hoiem et al. 2005, Mori 2005]

[Shi and Malik 2001]

Slide: Derek Hoiem

# Application: Better Classification



Angelova and Zhu, 2013

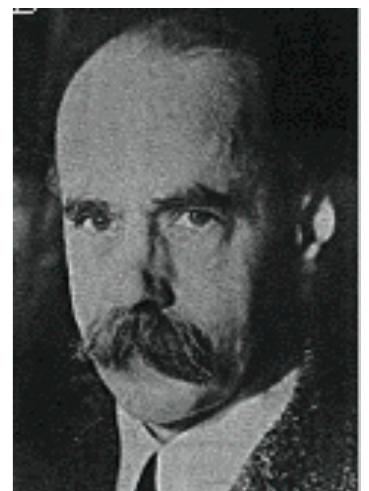
# History: Before Computer Vision

# Gestalt Theory

- Gestalt: whole or group
  - Whole is greater than sum of its parts
  - Relationships among parts can yield new properties/features
- Psychologists identified series of factors that predispose set of elements to be grouped (by human visual system)

*“I stand at the window and see a house, trees, sky.  
Theoretically I might say there were 327 brightnesses  
and nuances of colour. Do I have “327”? No. I have sky, house,  
and trees.”*

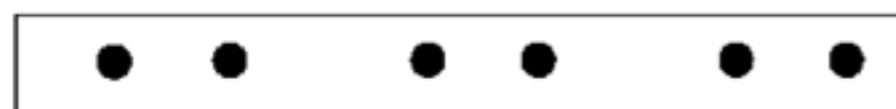
Max Wertheimer  
(1880-1943)



# Gestalt Factors



Not grouped



Proximity



Similarity



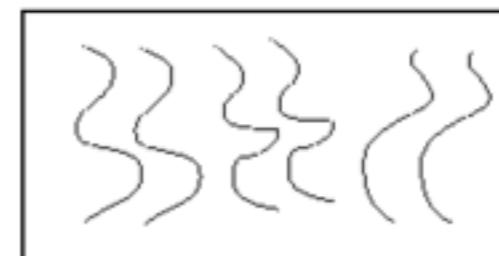
Similarity



Common Fate



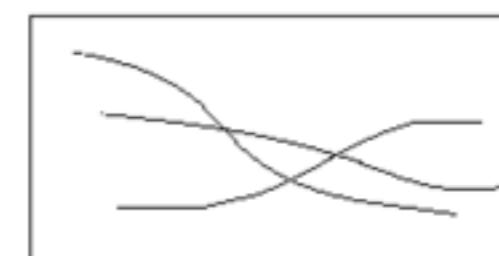
Common Region



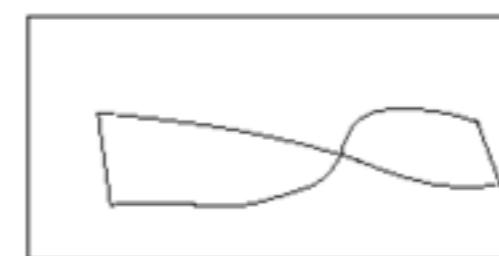
Parallelism



Symmetry



Continuity

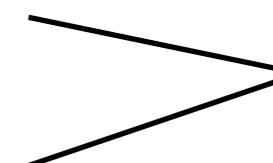


Closure

- These factors make intuitive sense, but are very difficult to translate into algorithms.

Image source: Forsyth & Ponce

# Outline

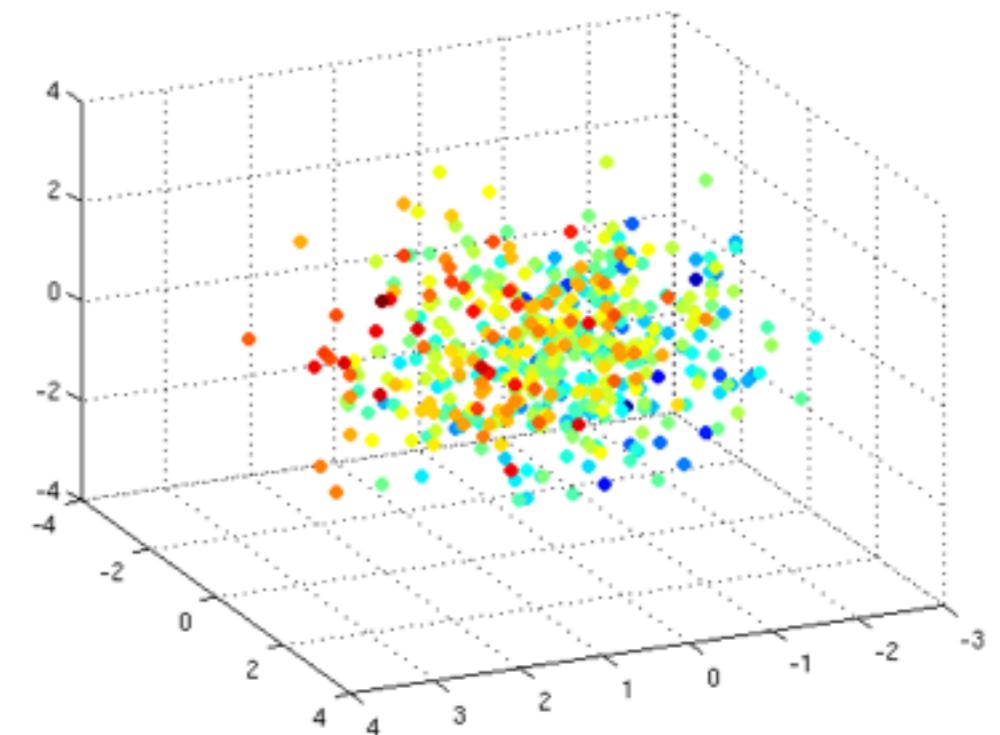
1. Segmentation as clustering
  2. Graph-based segmentation
  3. Segmentation as energy minimization — new stuff
- 
- CS 131 review

# Outline

1. Segmentation as clustering
  1. K-Means
  2. GMMs and EM
  3. Mean Shift
2. Graph-based segmentation
3. Segmentation as energy minimization

# Segmentation as Clustering

- Pixels are points in a high-dimensional space
  - color: 3d
  - color + location: 5d
- Cluster pixels into segments



# Clustering: K-Means

## Algorithm:

1. Randomly initialize the cluster centers,  $c_1, \dots, c_k$
  2. Given cluster centers, determine points in each cluster
    - For each point  $p$ , find the closest  $c_i$ . Put  $p$  into cluster  $i$
  3. Given points in each cluster, solve for  $c_i$ 
    - Set  $c_i$  to be the mean of points in cluster  $i$
  4. If  $c_i$  have changed, repeat Step 2
- 
- Properties
    - Will always converge to some solution
    - Can be a “local minimum”
      - Does not always find the global minimum of objective function:

$$\sum_{i=1}^k \sum_{x \in \mathcal{C}_i} \|x - c_i\|^2$$

slide credit: Steve Seitz

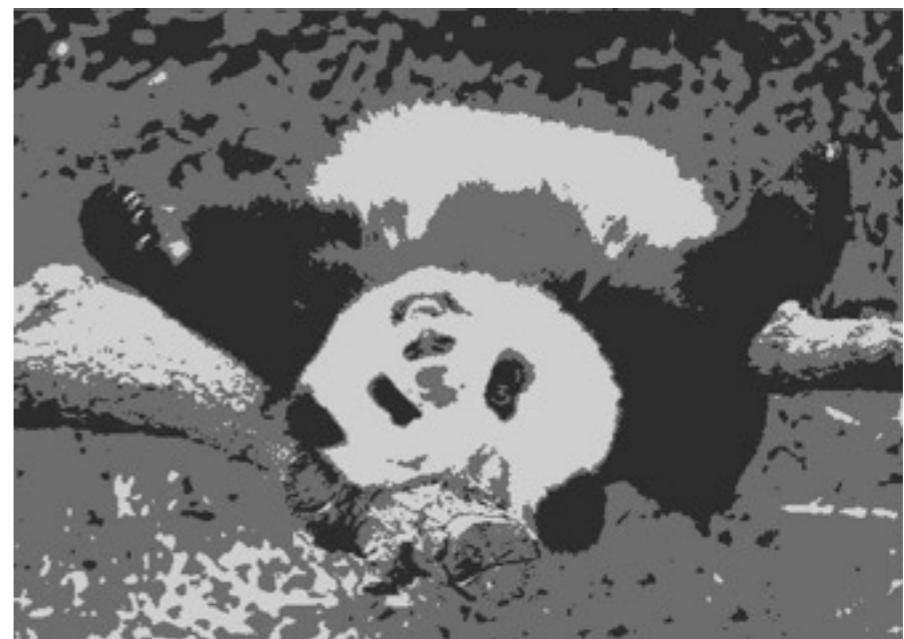
# Clustering: K-Means



$k=2$



$k=3$



slide credit: Kristen Grauman

# Clustering: K-Means



Note: Visualize segment with average color

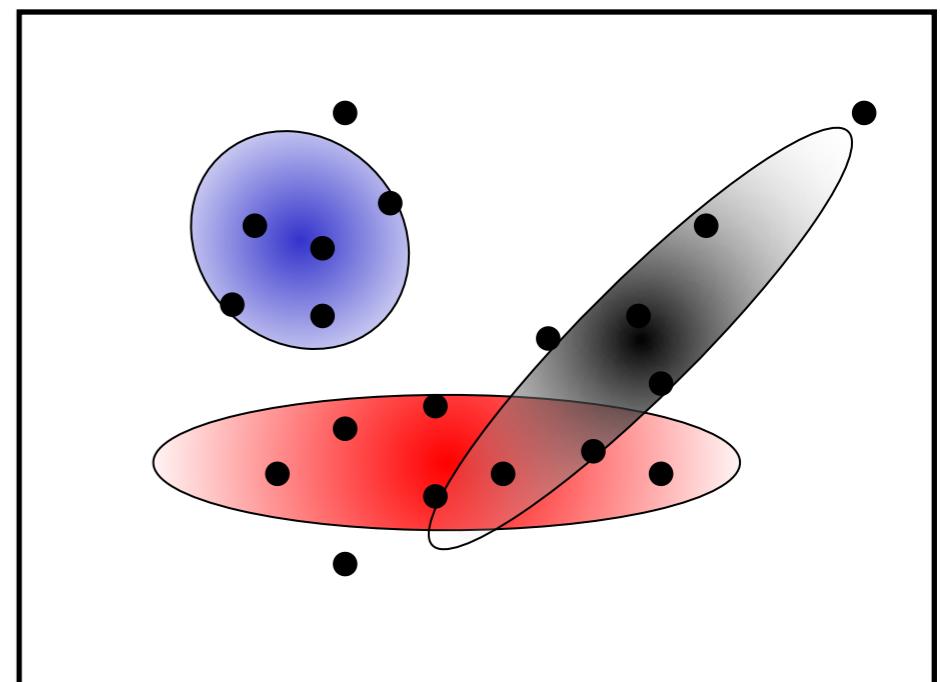
# K-Means

Pro:

- Extremely simple
- Efficient

Con:

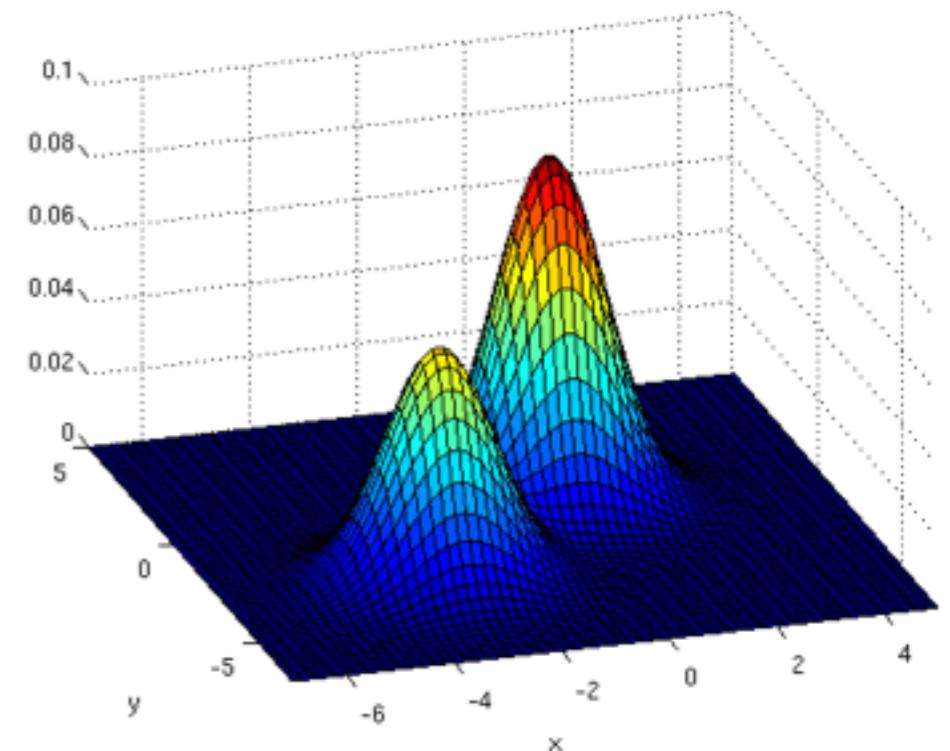
- Hard quantization in clusters
- Can't handle non-spherical clusters



# Gaussian Mixture Model

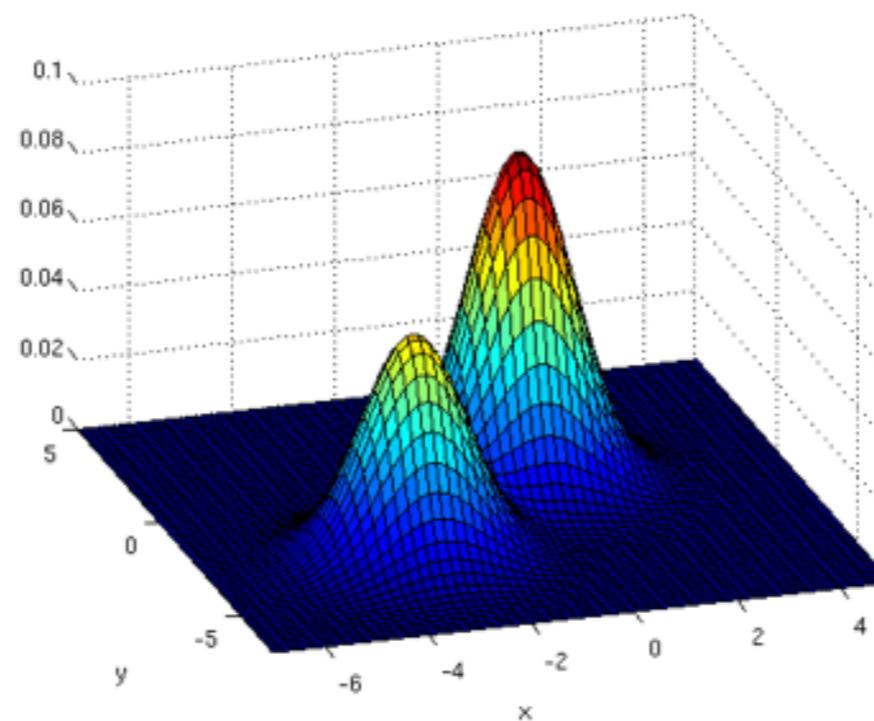
- Represent data distribution as mixture of multivariate Gaussians.

$$P(x) = \sum_{i=1}^K \pi_i \cdot pdf(x; \mu_i, \Sigma_i)$$



How do we actually fit this distribution?

# Expectation Maximization (EM)



- Goal
  - Find parameters  $\theta$  (for GMMs:  $\pi_i, \mu_i, \Sigma_i$ ) that maximize the likelihood function:
$$P(\text{data}; \theta) = \prod_{i=1}^N P(x_i; \theta)$$
- Approach:
  1. E-step: given current parameters, compute ownership of each point
  2. M-step: given ownership probabilities, update parameters to maximize likelihood function
  3. Repeat until convergence

See CS229 material if this is unfamiliar!

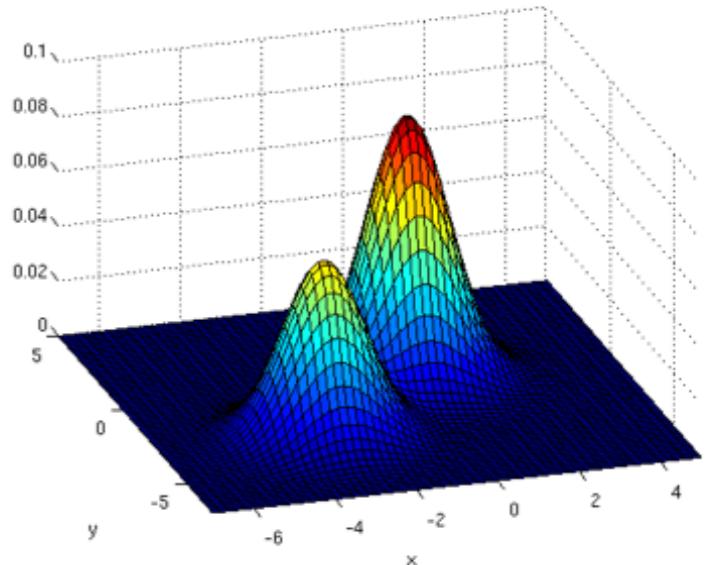
# Clustering: Expectation Maximization (EM)



# GMMs

Pro:

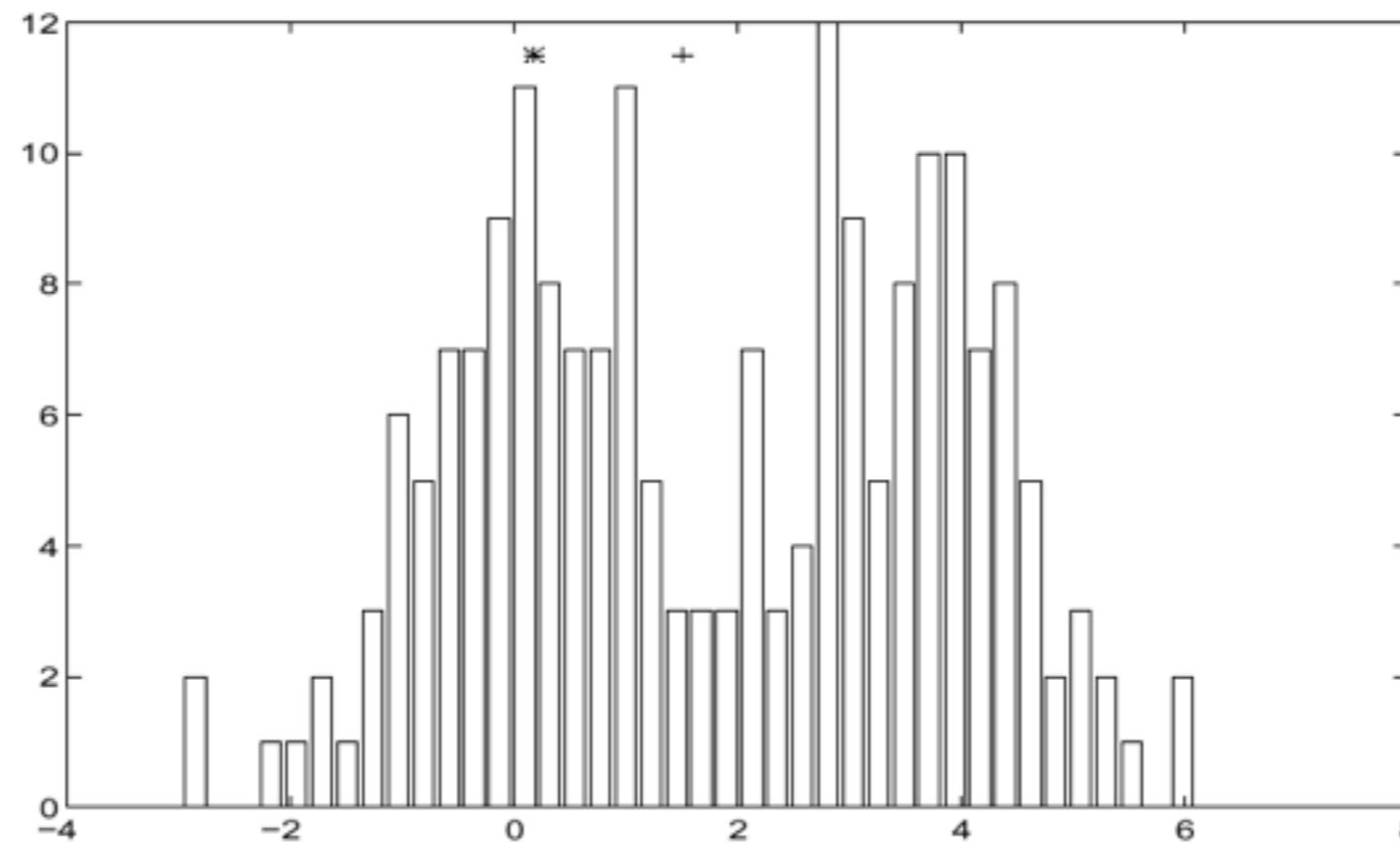
- Still fairly simple and efficient
- Model more complex distributions



Con:

- Need to know number of components in advance – hard to know unless you're looking at the data yourself!

# Clustering: Mean-shift



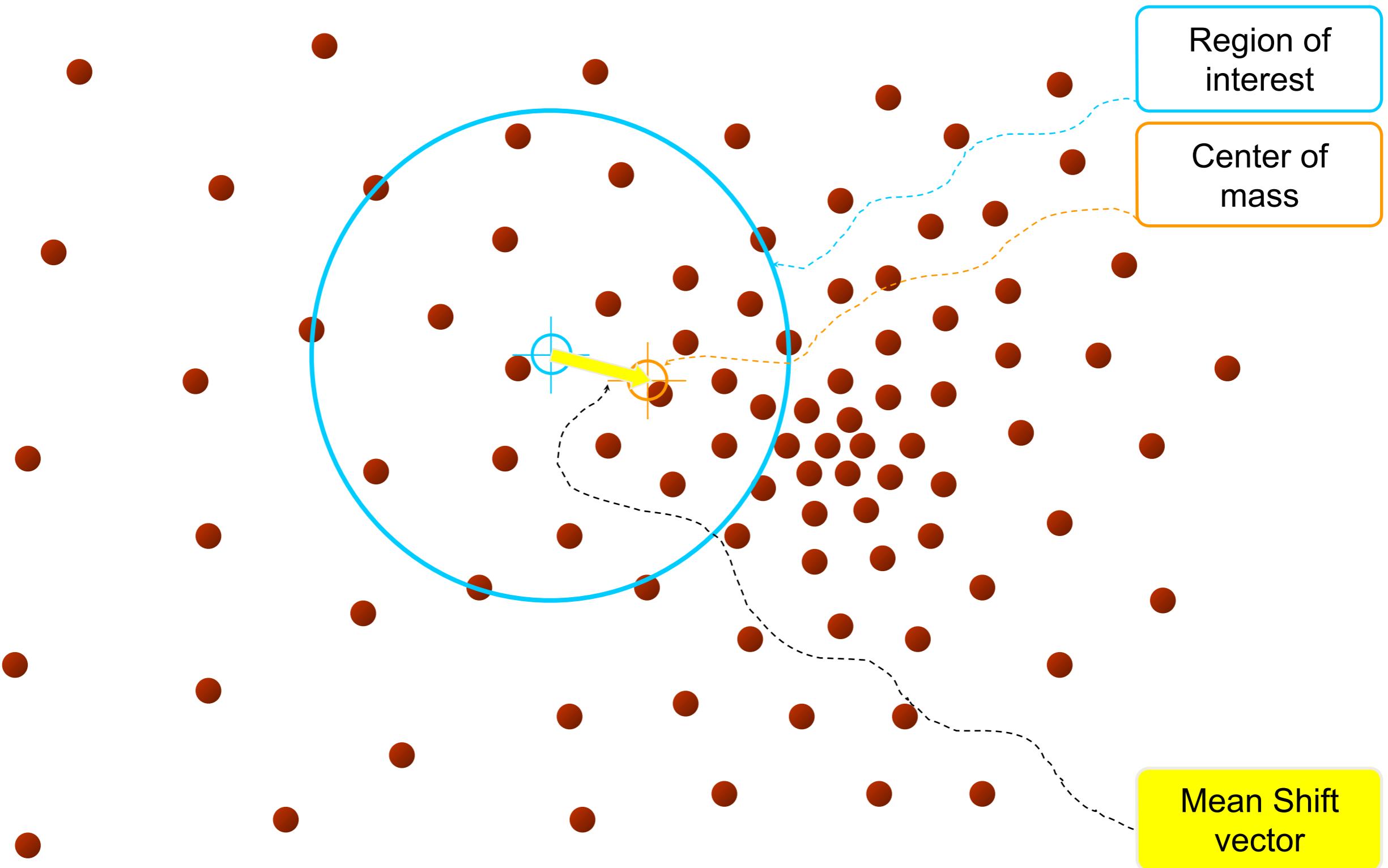
1. Initialize random seed, and window  $W$
2. Calculate center of gravity (the “mean”) of  $W$ :
  - Can generalize to arbitrary windows/kernels
3. Shift the search window to the mean
4. Repeat Step 2 until convergence

$$\frac{1}{|W|} \sum_{x \in W} x$$

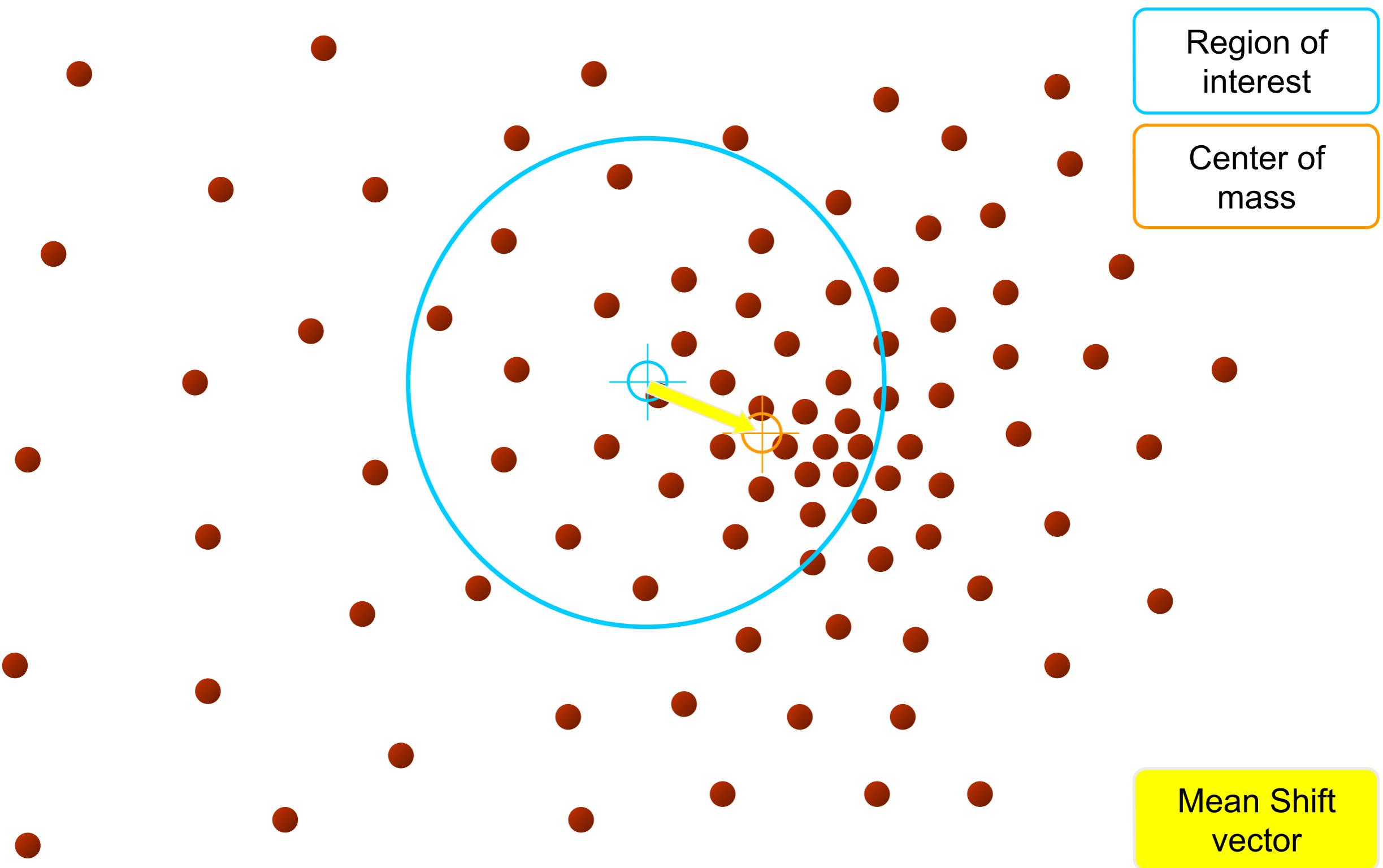
Only parameter: window size

slide credit: Steve Seitz

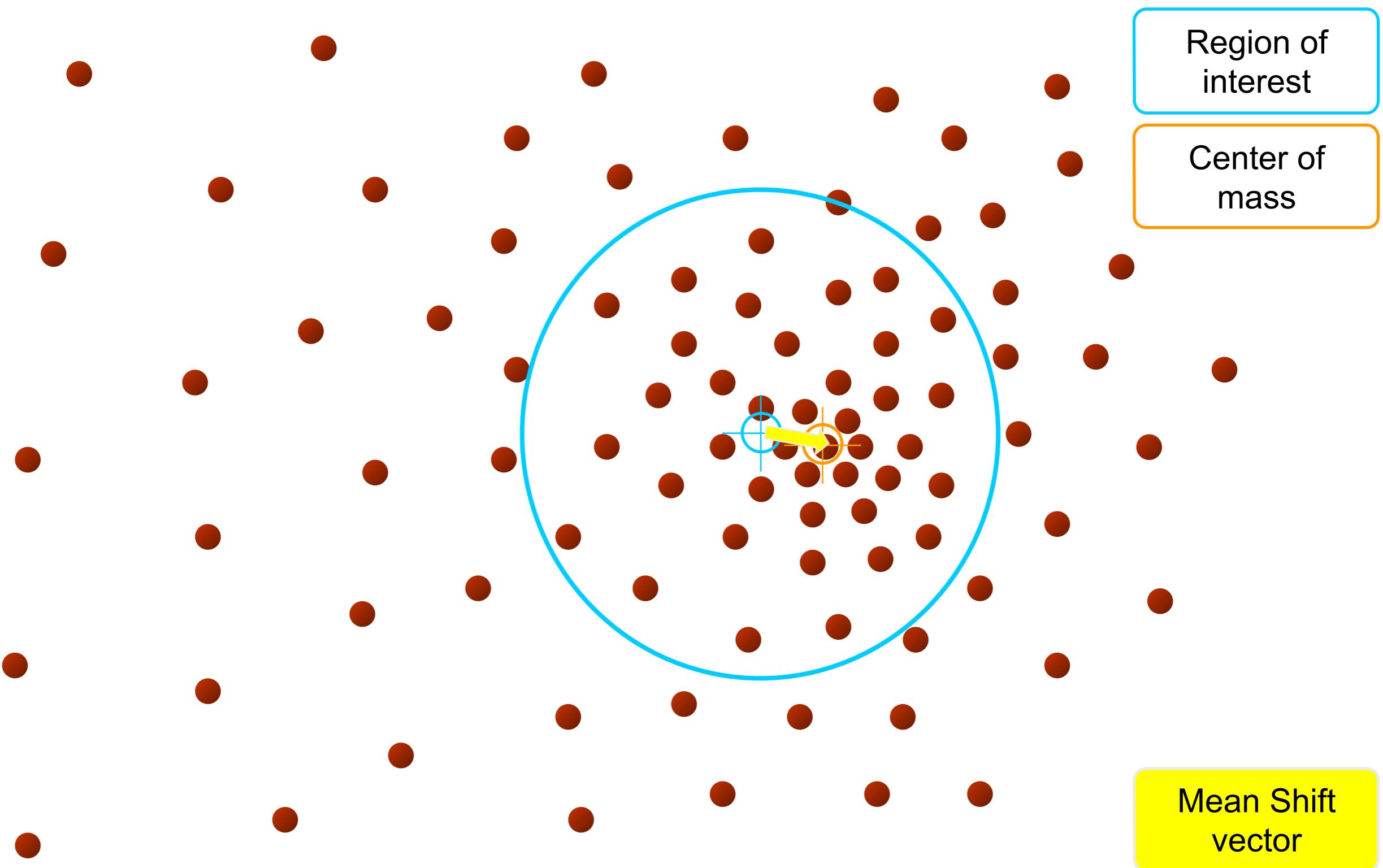
# Mean-Shift



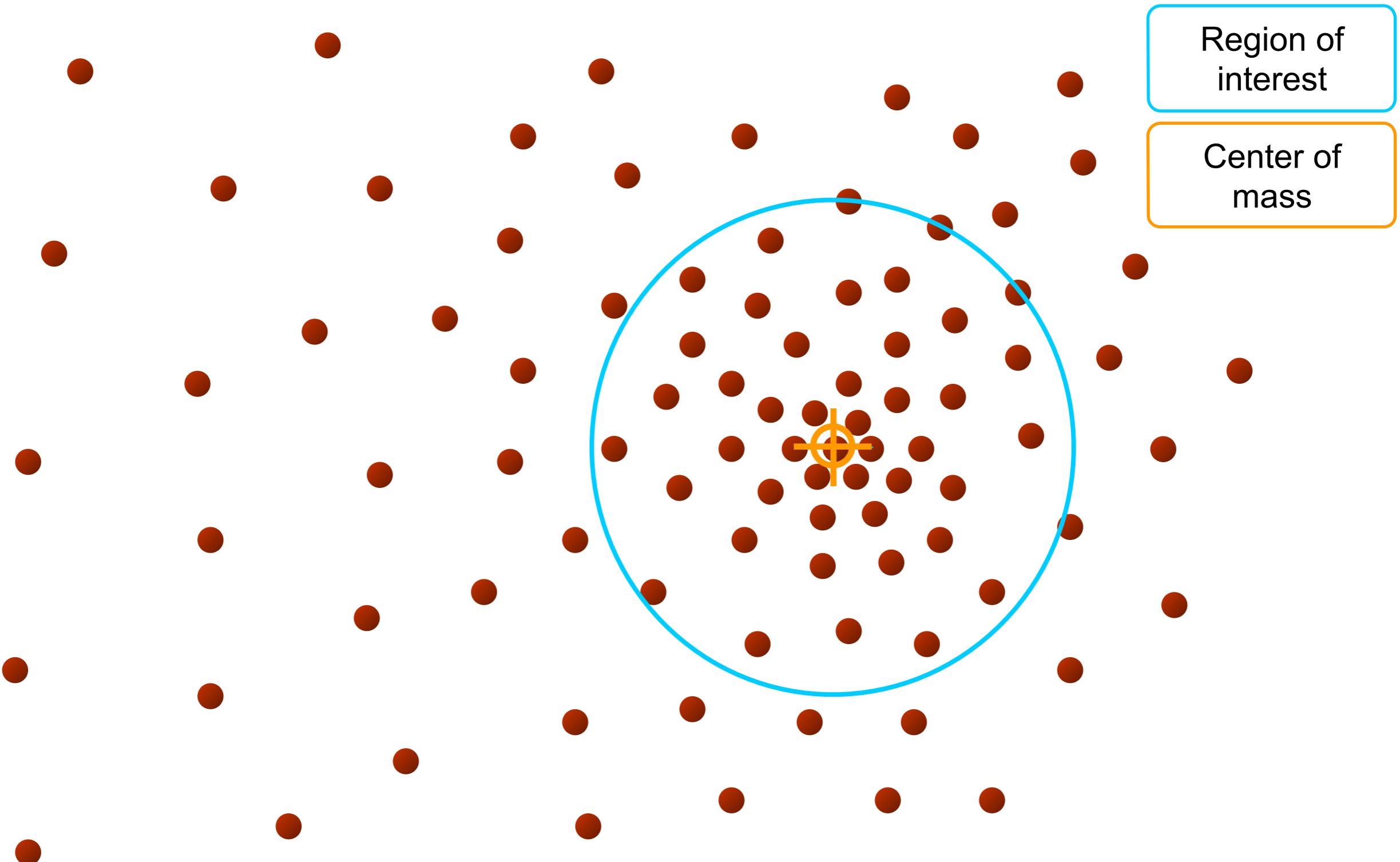
# Mean-Shift



# Mean-Shift

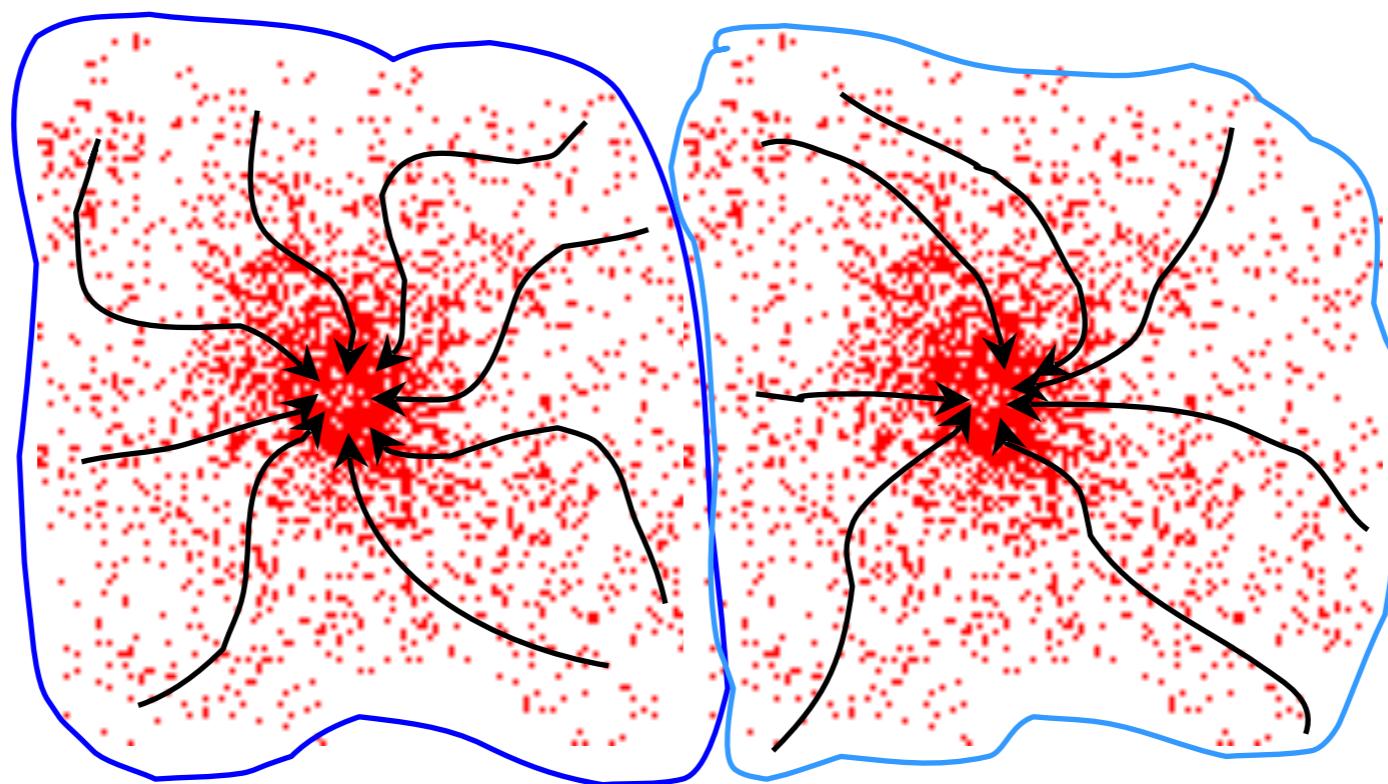


# Mean-Shift



# Clustering: Mean-shift

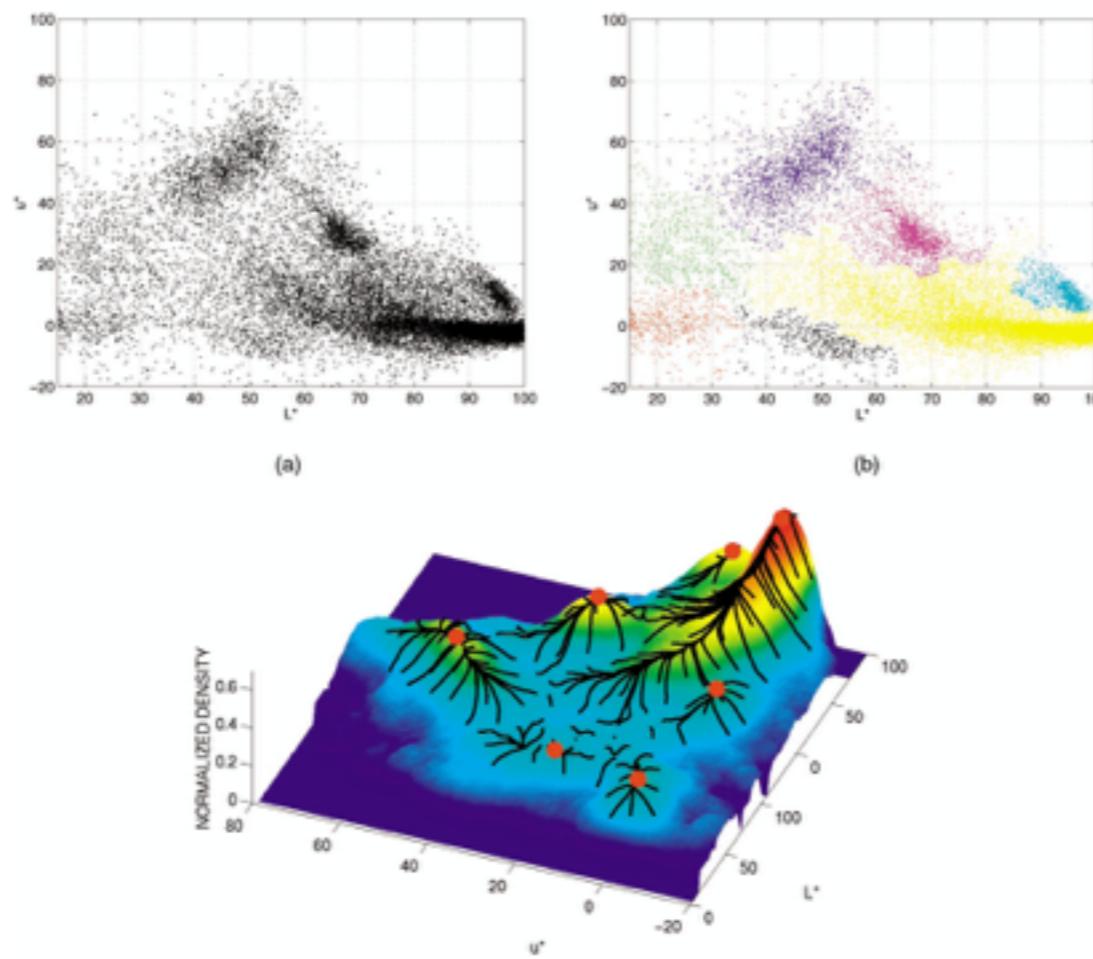
- Cluster: all data points in the attraction basin of a mode
- Attraction basin: the region for which all trajectories lead to the same mode



slide credit: Y. Ukrainitz & B. Sarel

# Mean-shift for segmentation

- Find features (color, gradients, texture, etc)
- Initialize windows at individual pixel locations
- Perform mean shift for each window until convergence
- Merge windows that end up near the same “peak” or mode



# Mean-shift for segmentation



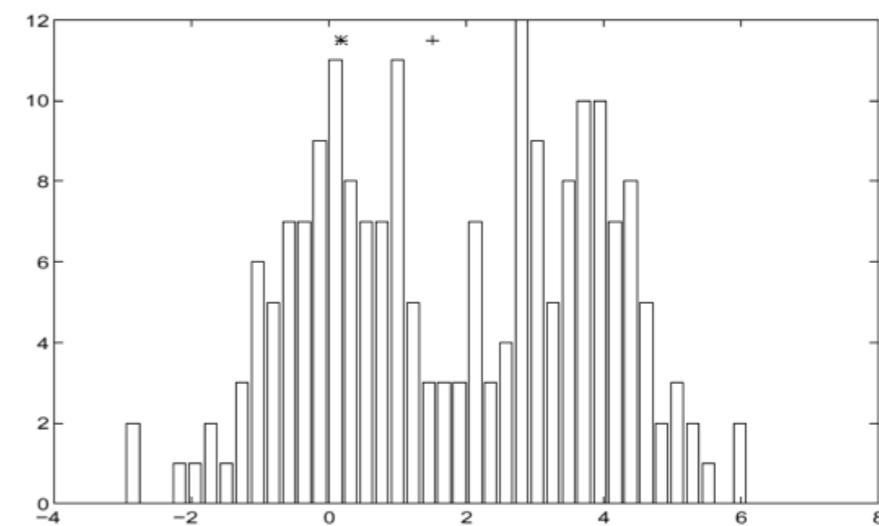
# Mean Shift

Pro:

- No number of clusters assumption
- Handle unusual distributions
- Simple

Con:

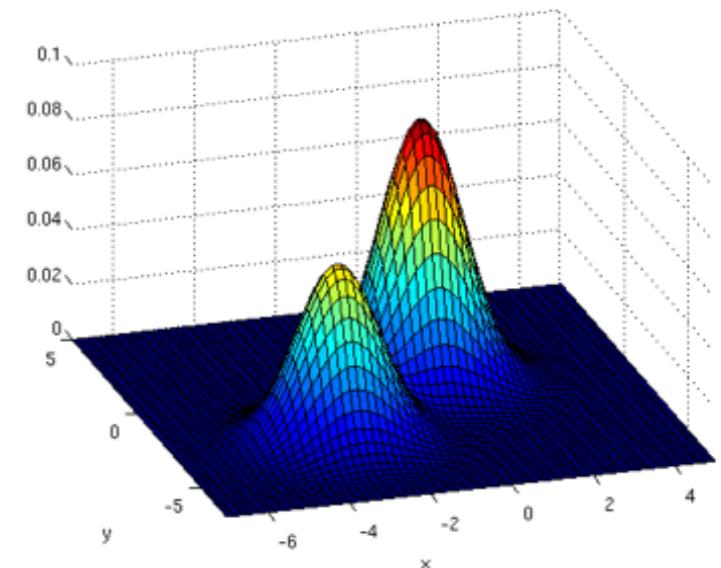
- Choice of window size
- Can be somewhat expensive



# Clustering

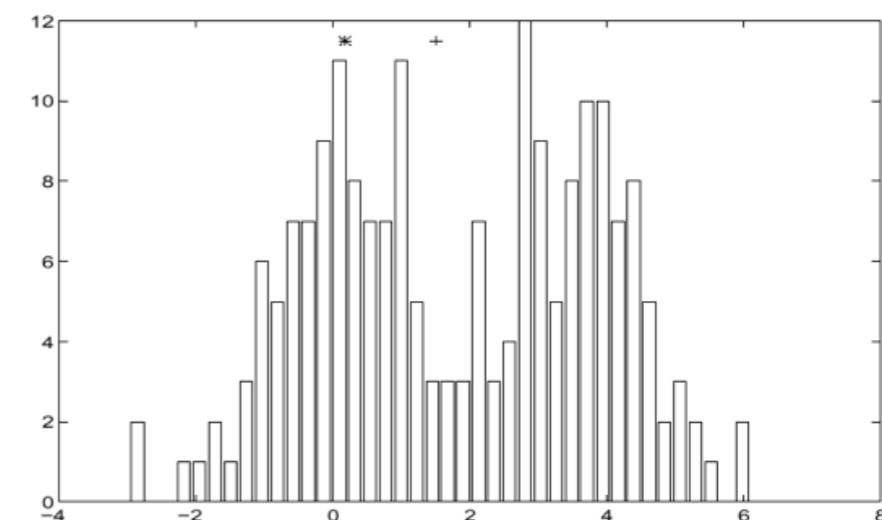
Pro:

- Generally simple
- Can handle most data distributions with sufficient effort.



Con:

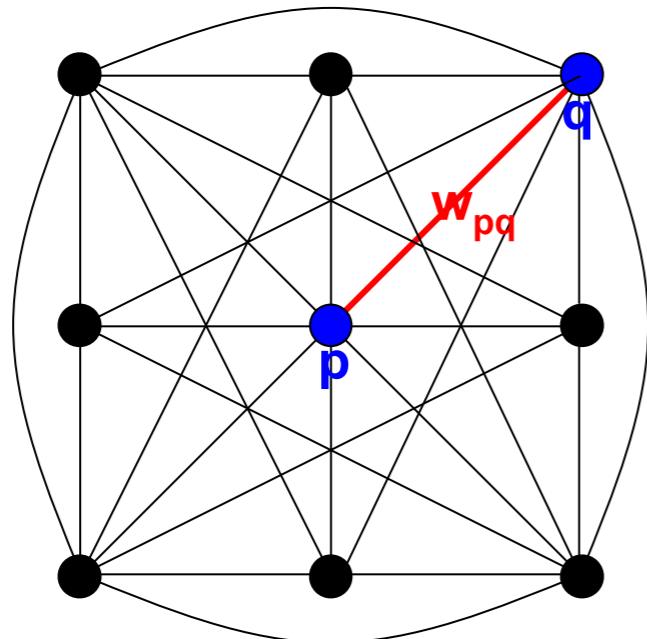
- Hard to capture global structure
- Performance is limited by simplicity



# Outline

1. Segmentation as clustering
2. Graph-based segmentation
  1. General Properties
  2. Spectral Clustering
  3. Min Cuts
  4. Normalized Cuts
3. Segmentation as energy minimization

# 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...)

slide credit: Steve Seitz

# 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

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

slide credit: Forsyth & Ponce

# Measuring Affinity

Distance:

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



slide credit: Forsyth & Ponce

# Measuring Affinity

Intensity:

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



slide credit: Forsyth & Ponce

# Measuring Affinity

Color:

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



slide credit: Forsyth & Ponce

# Measuring Affinity

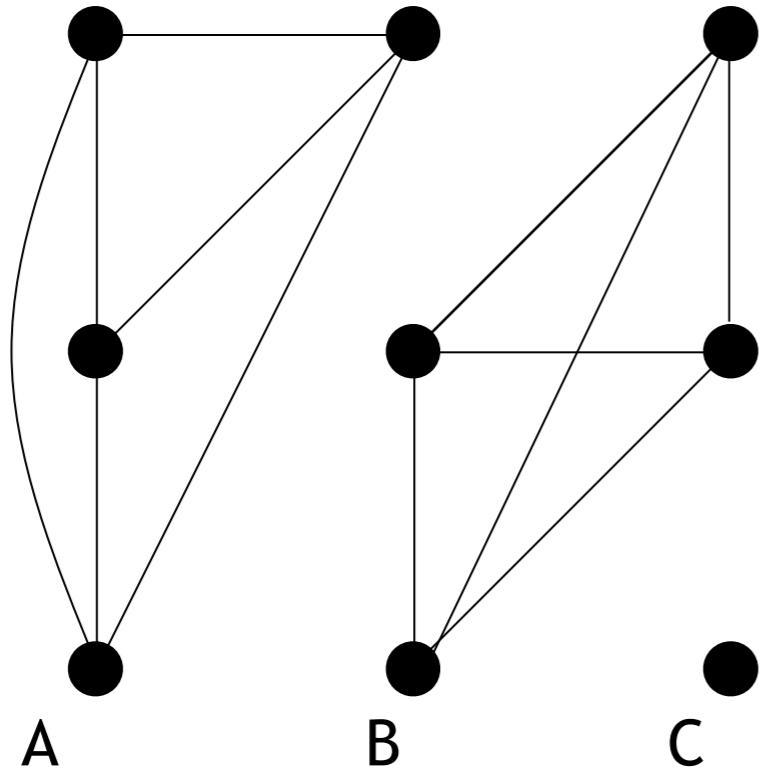
Texture:

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



slide credit: Forsyth & Ponce

# 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

slide credit: Steve Seitz

# Graph Cut with Eigenvalues

- Given: Affinity matrix  $W$
- Goal: Extract a single good cluster  $v$ 
  - $v(i)$ : score for point  $i$  for cluster  $v$

$$\begin{aligned} & \max_v v^T W v \\ & \text{s.t. } v^T v = 1 \end{aligned}$$

# Optimizing

$$\begin{array}{ll} \max_v & v^T W v \\ \text{s.t.} & v^T v = 1 \end{array} \quad \longleftrightarrow \quad \begin{array}{ll} \min_v & -\frac{1}{2} v^T W v \\ \text{s.t.} & v^T v = 1 \end{array}$$

Lagrangian:  $-\frac{1}{2}v^T W v + \lambda(v^T v - 1)$

$$-Wv + \lambda v = 0$$

$$Wv = \lambda v$$

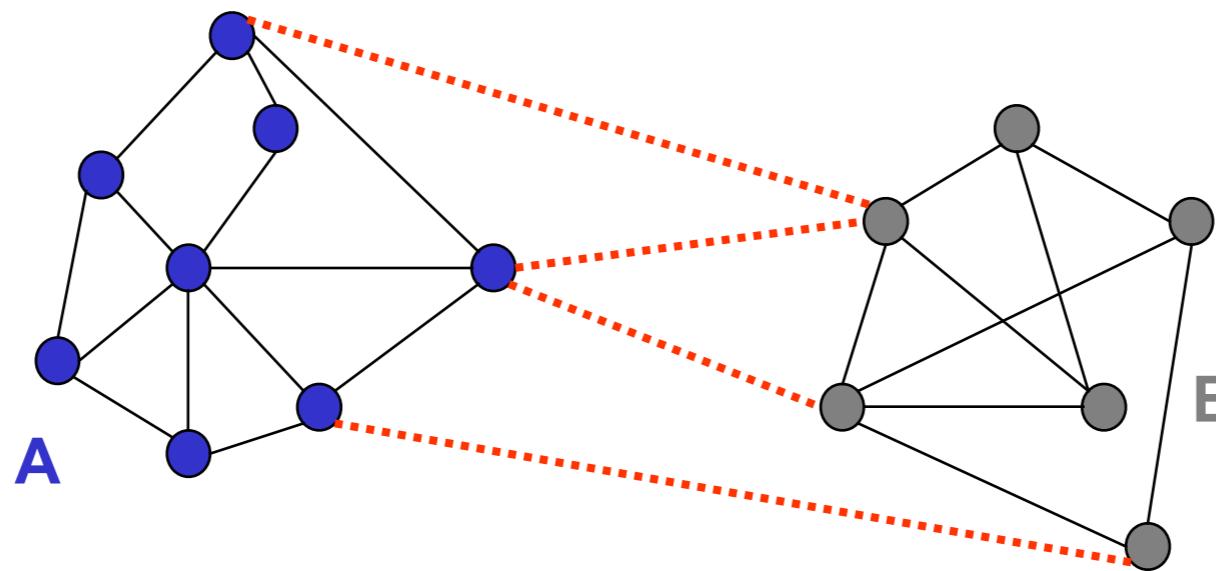
$v$  is an eigenvector of  $W$

# Clustering via Eigenvalues

1. Construct affinity matrix  $W$
2. Compute eigenvalues and vectors of  $W$
3. Until done
  1. Take eigenvector of largest unprocessed eigenvalue
  2. Zero all components of elements that have already been clustered
  3. Threshold remaining components to determine cluster membership

Note: This is an example of a *spectral clustering* algorithm

# Graph Cuts - Another Look



- Set of edges whose removal makes a graph disconnected
- Cost of a cut
  - Sum of weights of cut edges:  $cut(A, B) = \sum_{p \in A, q \in B} w_{pq}$
- A graph cut gives us a segmentation
  - What is a “good” graph cut and how do we find one?

slide credit: Steve Seitz

# Formulation: Min Cut

- We can do segmentation by finding the *minimum cut*
  - either smallest number of elements (unweighted) or smallest sum of weights (weighted)
  - efficient algorithms exist
- Drawback
  - Weight of cut proportional to number of edges
  - Biased towards cutting small, isolated components

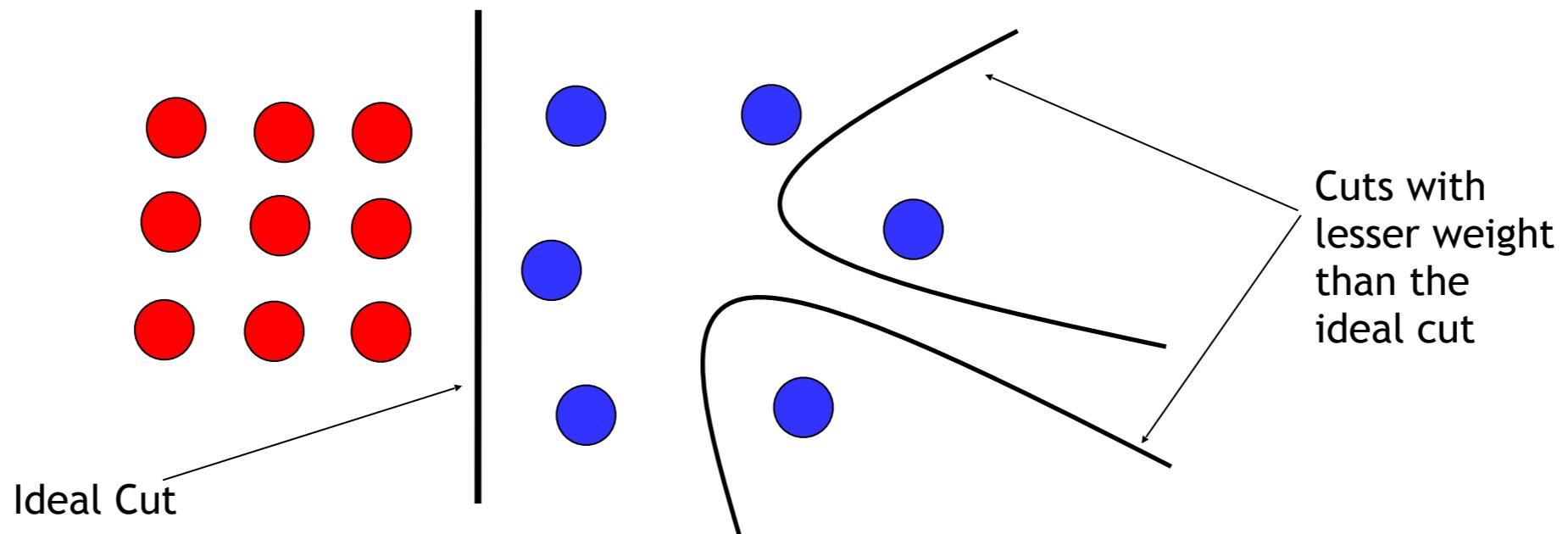


image credit: Khurran Hassan-Shafique

# Formulation: Normalized Cuts

- Key idea: normalize segment size
  - Fixes min cut's bias
- Formulation:

$$\begin{aligned} Ncut(A, B) &= \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \\ &= cut(A, B) \left[ \frac{1}{\sum_{p \in A} w_{p,q}} + \frac{1}{\sum_{q \in B} w_{p,q}} \right] \end{aligned}$$

$assoc(A, V)$  = sum of weights of edges in  $V$  that touch  $A$

- NP-hard, but can approximate

J. Shi and J. Malik. [Normalized cuts and image segmentation.](#) PAMI 2000

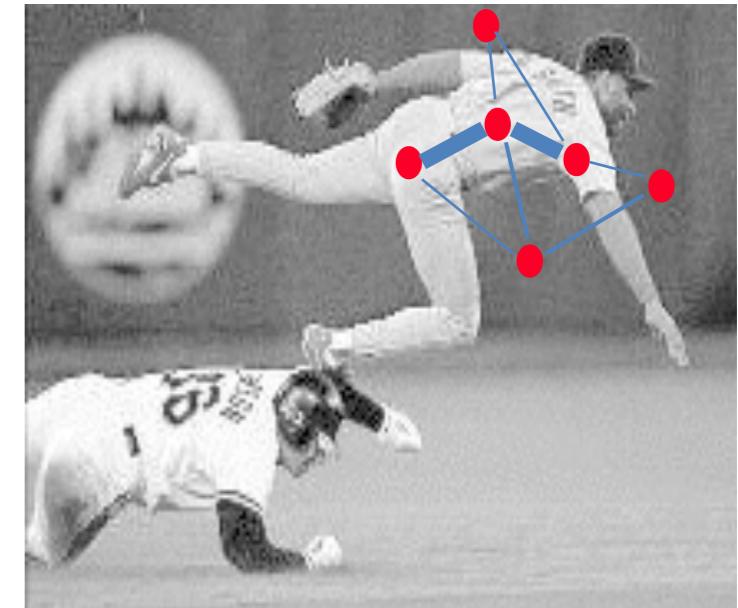
# NCuts as Generalized Eigenvector Problem

Definitions:

$W$  : **affinity matrix**

$D$  : **diagonal matrix**  $D(i, i) = \sum_j w_{i,j}$

$z$  : **vector** in  $\{-1, 1\}^N$ ,  $z_i = 1 \Leftrightarrow i \in A$



In matrix form:

$$\begin{aligned} NCut(A, B) &= \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)} \\ &= \frac{(1+z)^T(D-W)(1+z)}{k1^T D 1} + \frac{(1-z)(D-W)(1-z)}{(1-k)1^T D 1}; \quad k = \frac{\sum_{z_i>0} D(i, i)}{\sum_i D(i, i)} \\ &= \dots \end{aligned}$$

Slide credit: Jitendra Malik

# After a lot of math...

- After simplification, we get

$$NCut(A, B) = \frac{y^T(D - W)y}{y^T D y}, \quad y_i \in \{1, -b\}, \quad y^T D 1 = 0$$

This is hard,  
 $y$  is discrete!

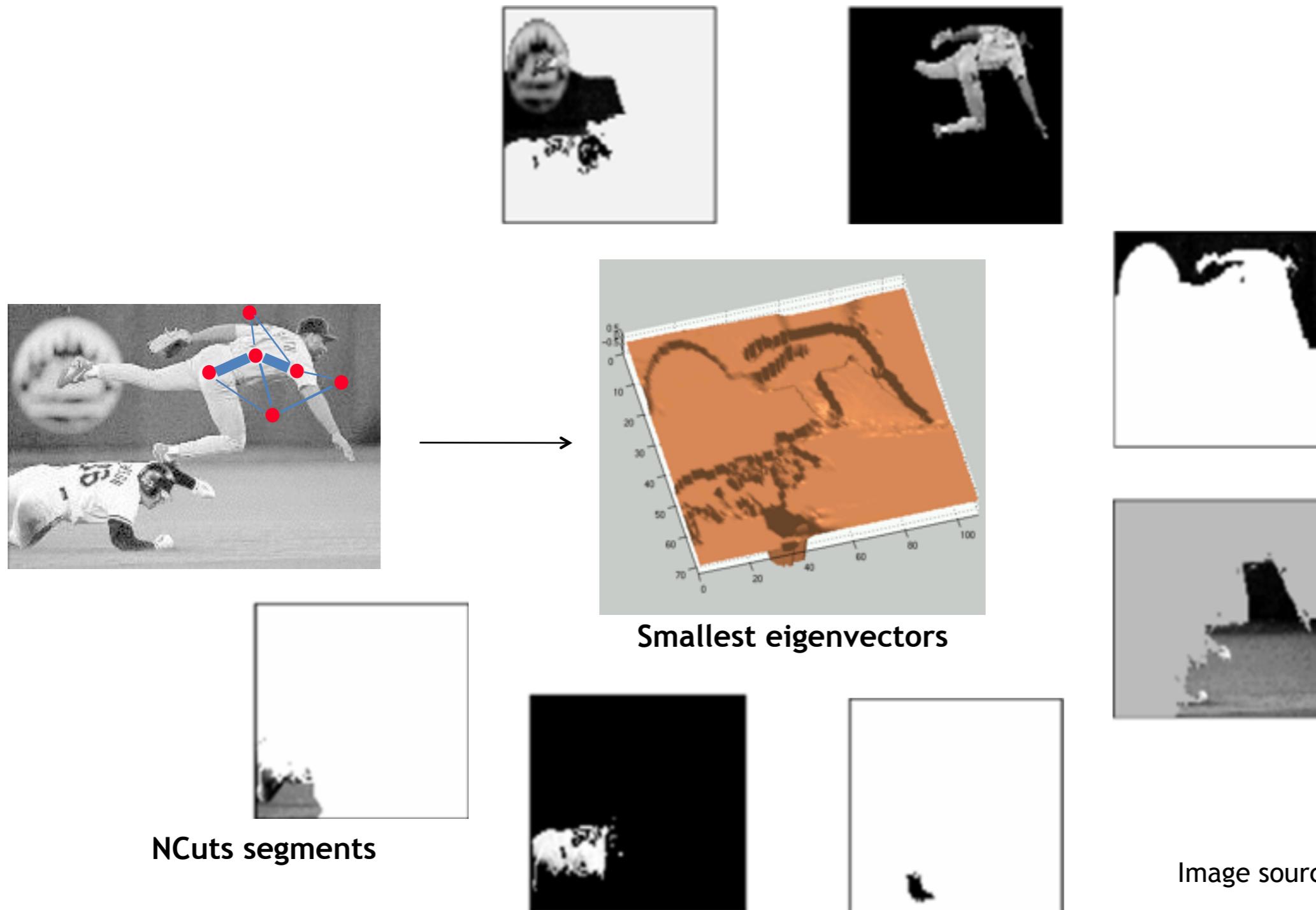
- This is a Rayleigh Quotient
  - Solution given by the “generalized” eigenvalue problem

$$(D - W)y = \lambda D y$$

Relaxation:  
continuous  $y$

- Subtleties
  - Optimal solution is second smallest eigenvector
  - Gives continuous result—must convert into discrete values of  $y$

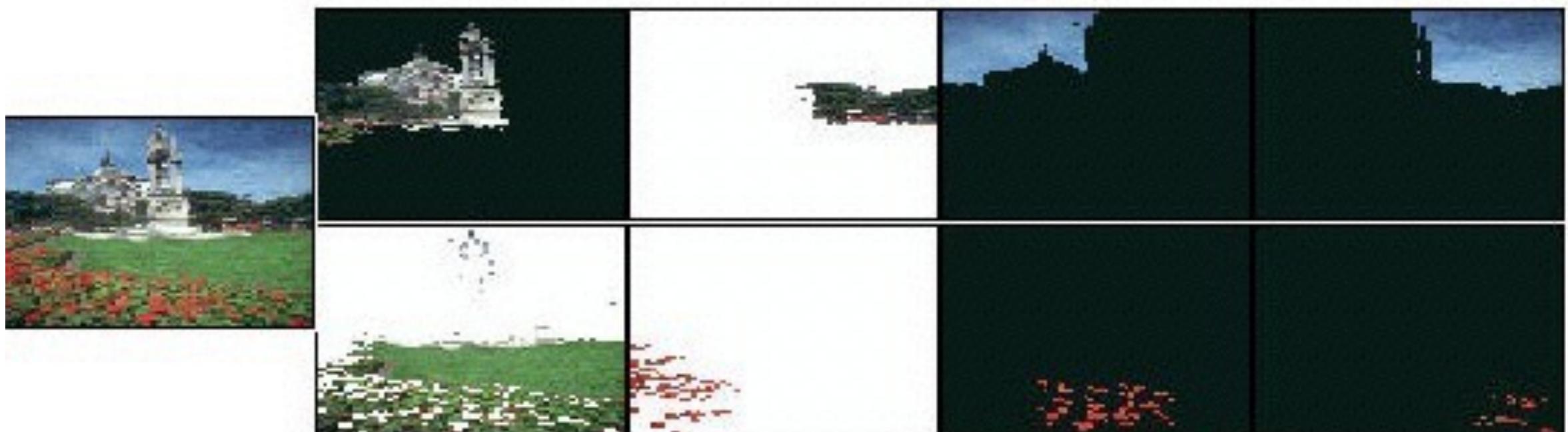
# NCuts example



# NCuts: Algorithm Summary

1. Construct weighted graph  $G = (V, E)$
2. Construct affinity matrix  $W$
3. Solve  $(D - W)y = \lambda Dy$  for smallest few eigenvectors.
  - This is a continuous solution
4. Threshold eigenvectors to get a discrete cut
  - This is the approximation
  - As before, several heuristics for doing this
5. Recursively subdivide as desired.

# NCuts examples



# NCuts examples



# NCuts Pro and Con

- 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



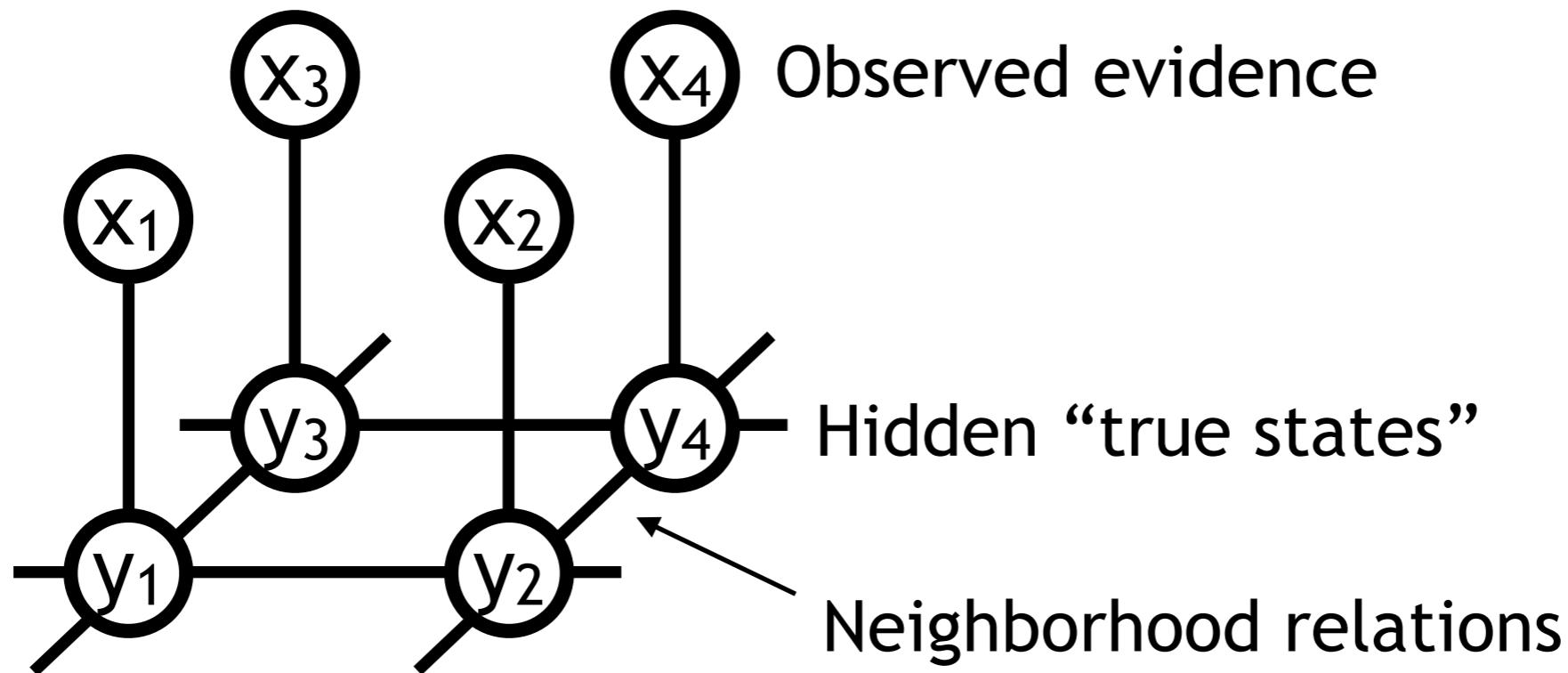
Slide source: Kristen Grauman

# Outline

1. Segmentation as clustering
2. Graph-based segmentation
3. Segmentation as energy minimization
  1. MRFs + CRFs
  2. Segmentation with CRFs
  3. GrabCut

# Conditional Random Fields (CRFs)

- Rich probabilistic model for images
- Built in local, modular way
  - Get global effects from only learning/modeling local ones
- After conditioning, get a Markov Random Field (MRF)



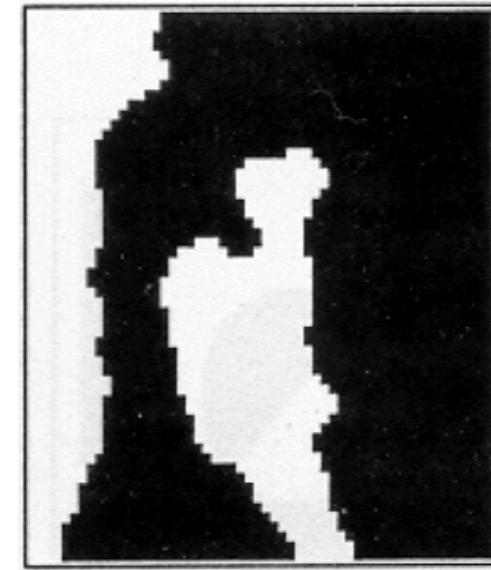
# Pixels as CRF Nodes



Original image



Degraded image



Reconstruction  
from MRF modeling  
pixel neighborhood  
statistics

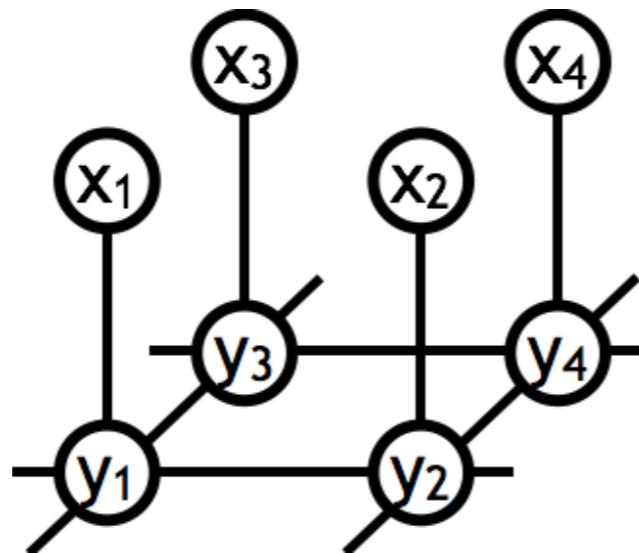


Image source: Bastian Liebe

# CRF Probability

$$P(x, y) = \frac{1}{Z} \prod_i \Phi(x_i, y_i) \prod_{i,j} \Psi(y_i, y_j)$$

↑  
Scene  
Image

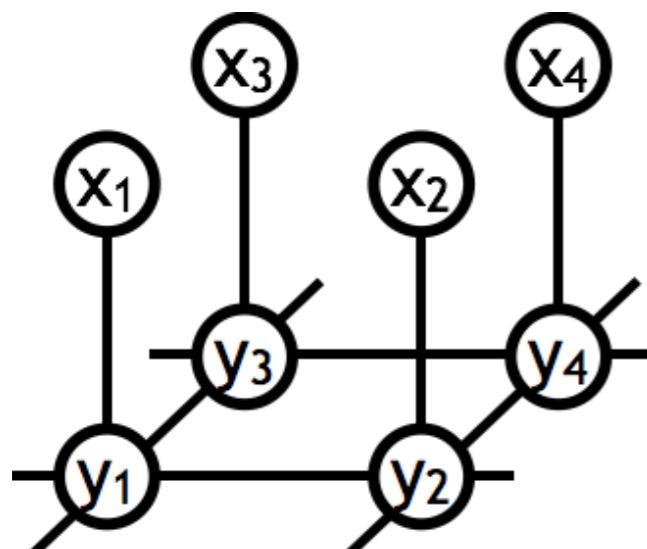
↑  
 $i$   
Image-scene compatibility function

↑  
Partition Function

↑  
Local observations

↑  
 $i, j$   
Scene-scene compatibility function

↑  
Neighboring scene nodes



# Energy Formulation

$$P(x, y) = \frac{1}{Z} \prod_i \Phi(x_i, y_i) \prod_{i,j} \Psi(y_i, y_j)$$

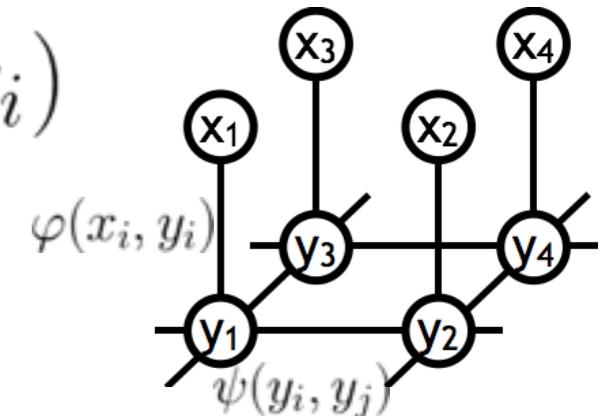
take logs, drop  $Z$

$$E(x, y) = \sum_i \varphi(x_i, y_i) + \sum_{i,j} \psi(y_i, y_j)$$

- We call  $E$  an *energy function*
  - named from free-energy problems in statistical mechanics
- Individual terms are *potentials*
- *Note: Derived this way, it's energy maximization. Be careful and check each formulation individually.*

# Energy Formulation

$$E(x, y) = \sum_i \varphi(x_i, y_i) + \sum_{i,j} \psi(y_i, y_j)$$



- Unary potentials  $\varphi$ 
  - Local information about each pixel
  - e.g. how likely a pixel/patch belongs to a certain class
- Pairwise potentials  $\psi$ 
  - Neighborhood information, enforces consistency
  - e.g. how different a pixel is from its neighbor in appearance

slide credit: Bastian Liebe

# CRF segmentation example

- Boykov and Jolly (2001)

$$E(x, y) = \sum_i \varphi(x_i, y_i) + \sum_{i,j} \psi(y_i, y_j)$$

- Variables

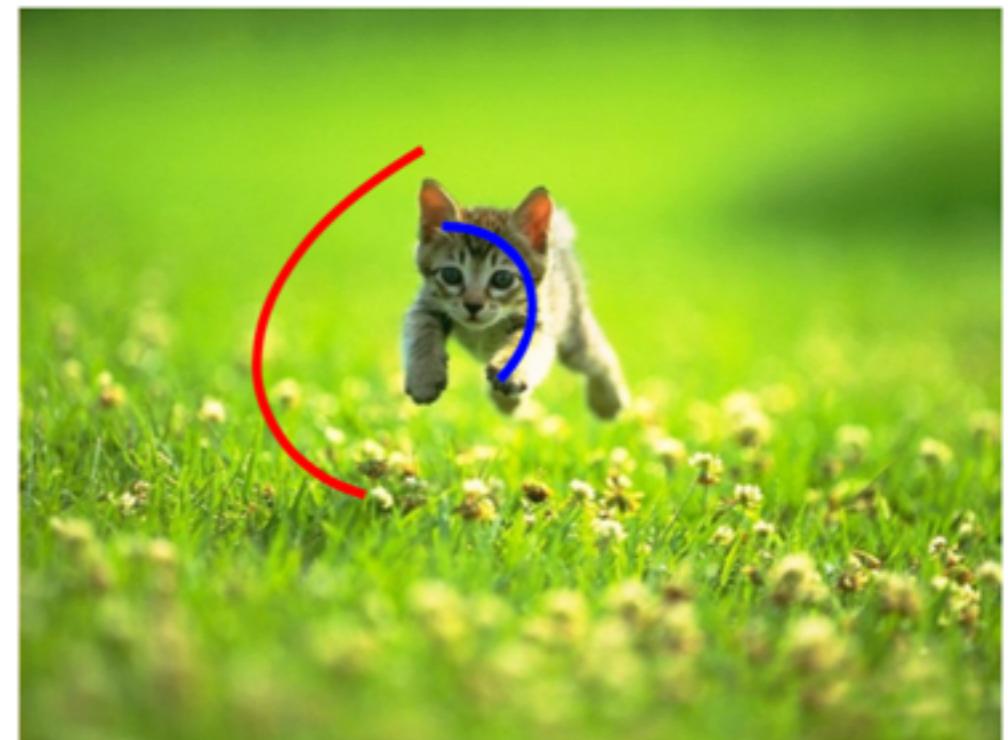
- $x_i$  : Annotation (Input)
  - Foreground/background/empty
- $y_i$  : Binary variable
  - Foreground/background

- Unary term

- $\varphi(x_i, y_i) = K[x_i \neq y_i]$
- Penalty for disregarding annotation

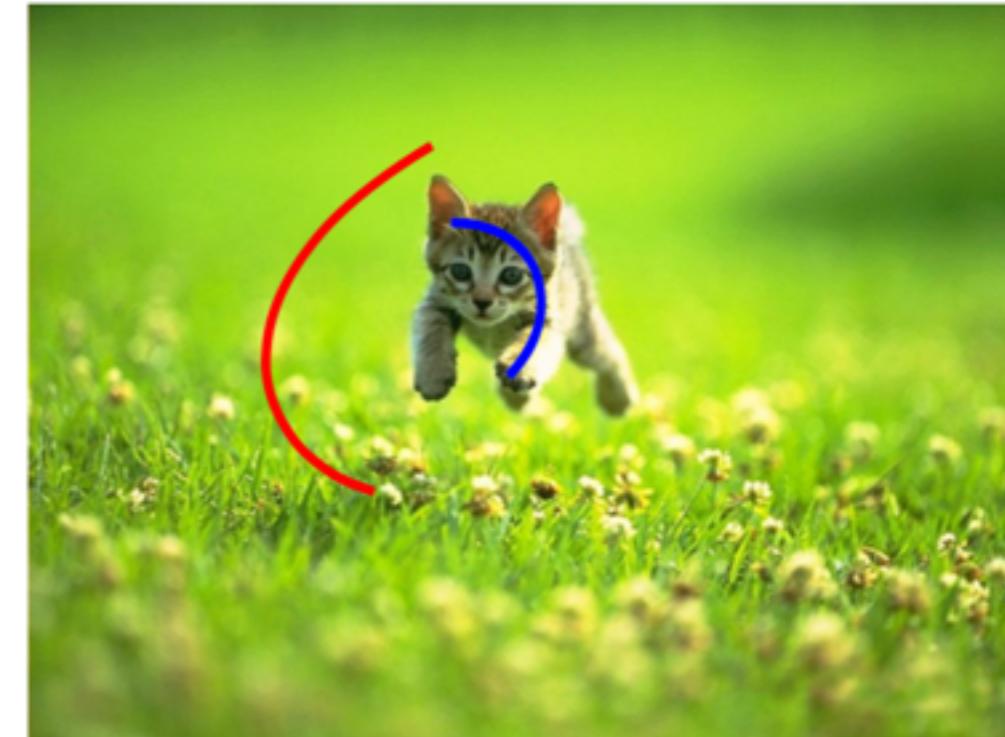
- Pairwise term

- $\psi(y_i, y_j) = [y_i \neq y_j]w_{ij}$
- Encourage smooth annotations
- $w_{ij}$  is affinity between pixels  $i$  and  $j$



# Solving Efficiently

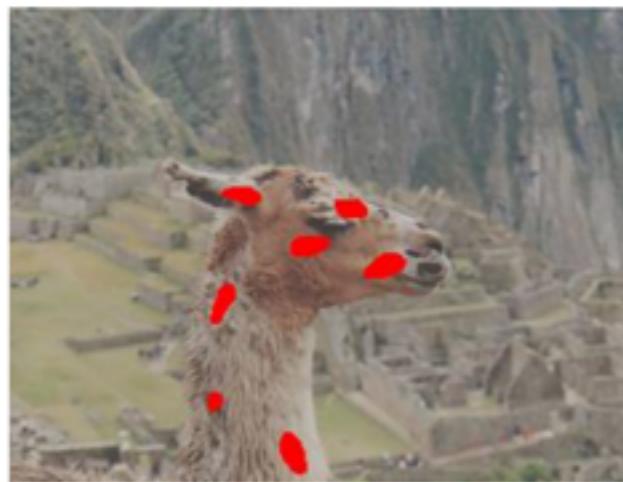
- Grid structured random fields
  - Maxflow/mincut
  - Optimal for binary labeling
    - submodular energy functions
  - Boykov & Kolmogorov, “An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision”, PAMI 2004
- Fully connected models
  - Efficient solution with convolution mean-field
  - Krähenbühl and Koltun, “Efficient Inference in Fully-Connected CRFs with Gaussian Edge Potentials”, NIPS 2011



# GrabCut: Interactive Foreground Extraction

User  
Input

**Magic Wand**  
(Adobe, 2002)



Result



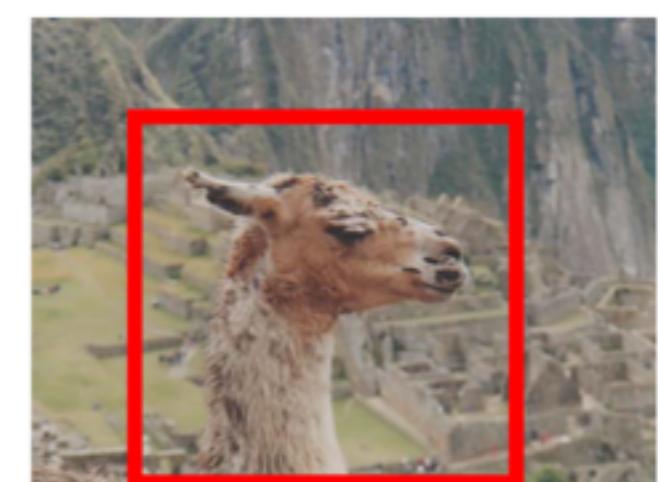
Regions

**Intelligent Scissors**  
Mortensen and Barrett (1995)



Boundary

**GrabCut**



Regions & Boundary

# GrabCut formulation

$$E(x, y, \theta, k) = \sum_i \varphi(x_i, y_i, \theta, k_i) + \sum_{ij} \psi(y_i, y_j, x_i, x_j)$$

- Variables
  - $x_i$ : pixel
  - $y_i \in \{0, 1\}$ : foreground/background label  $\{0, 1\}$
  - $k_i \in \{0, \dots, K-1\}$ : GMM mixture component
  - $\theta$ : GMM model parameters
  - $I = \{z_1, \dots, z_m\}$ : RGB image
- Unary Term  $\varphi(x_i, y_i, \theta, k_i)$ 
  - -log of GMM probability
- Pairwise Term

$$\psi(y_i, y_j, x_i, x_j) = \gamma[y_i \neq y_j] \exp(-\beta \|x_i - x_j\|^2)$$

# GrabCut - Iterative Optimization

1. Initialize Mixture Models based on user annotation
2. Assign GMM components

$$k_i = \arg \min \varphi(x_i, y_i, \theta, k_i)$$

3. Learn GMM parameters

$$\theta = \arg \min \sum_i \varphi(x_i, y_i, \theta, k_i)$$

4. Estimate segmentations (mincut)

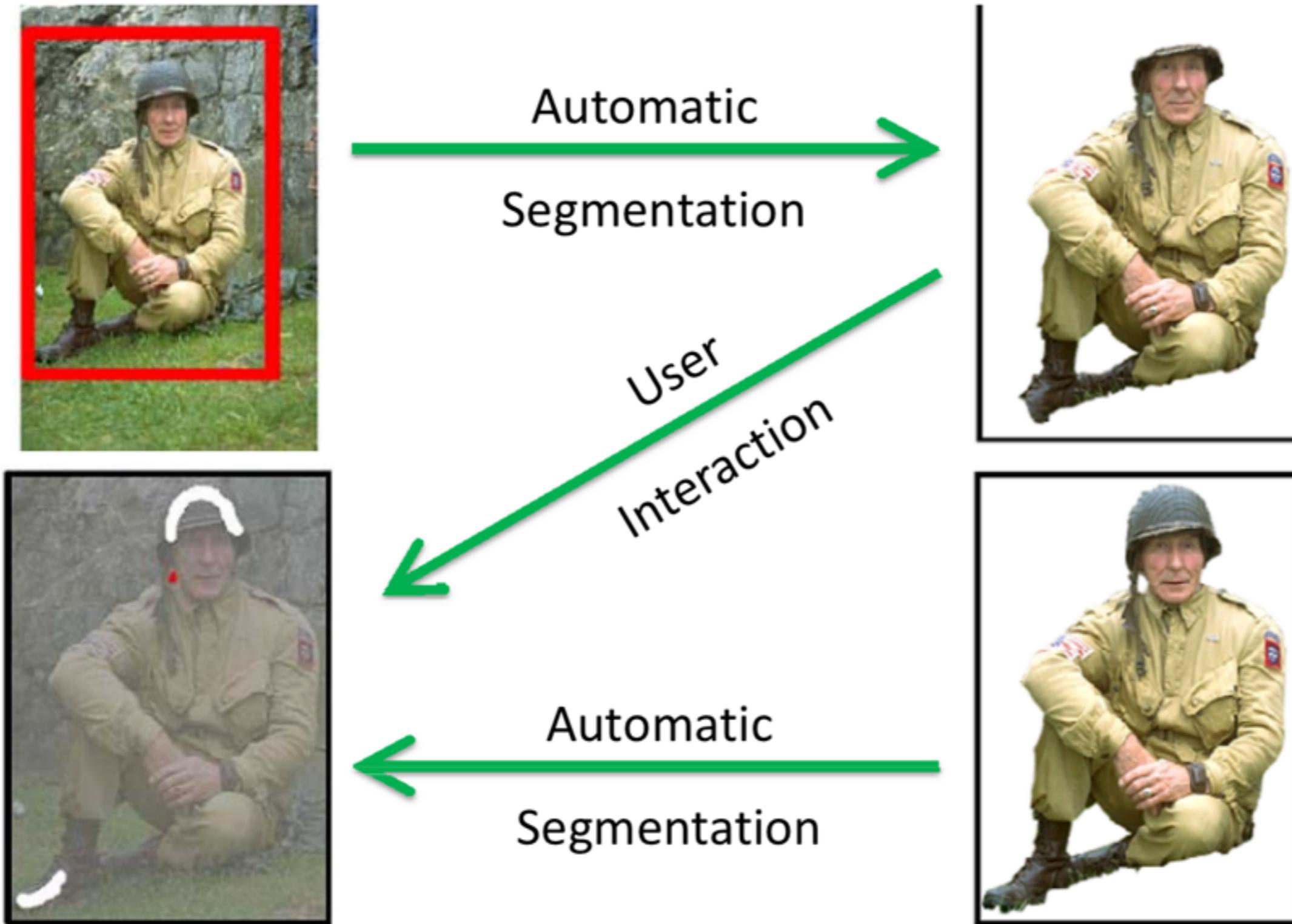
$$y = \arg \min E(x, y, \theta, k)$$

5. Repeat 2-4 until convergence

# GrabCut results



# Further editing with GrabCut

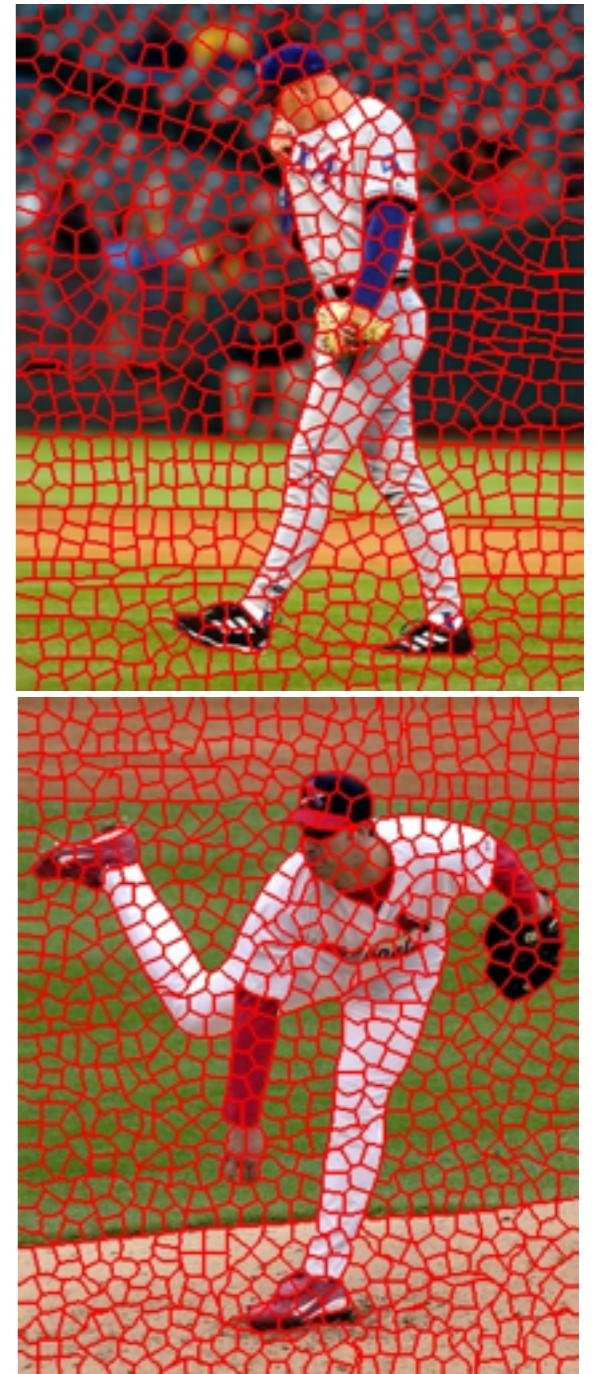


# Summary: Graph Cuts with CRFs

- Pros
  - Very powerful, get global results by defining local interactions
  - Very general
  - Rather efficient
  - Becoming more or less standard for many segmentation problems (GrabCut was 2004!)
- Cons
  - Only works for sub modular energy functions (binary)
  - Only approximate algorithms work for multi-label case

# Extra: Improving Segmentation Efficiency

- Images contain a lot of pixels
  - Even efficient methods can be slow
- Efficiency trick: Superpixels
  - Group together similar pixels
  - Cheap and local over segmentation
  - Must be high precision!
  - Many methods exist, try several.
- Another trick: Resize the image
  - Do segmentation in lower-res version, then scale back up to high-res.



# Additional Reading

- CS 131/CS231a slides (all)
- CS 229 (clustering, EM)
- CS 228/228T (MRFs, CRFs, energy minimization)
- EE 364 (optimization)