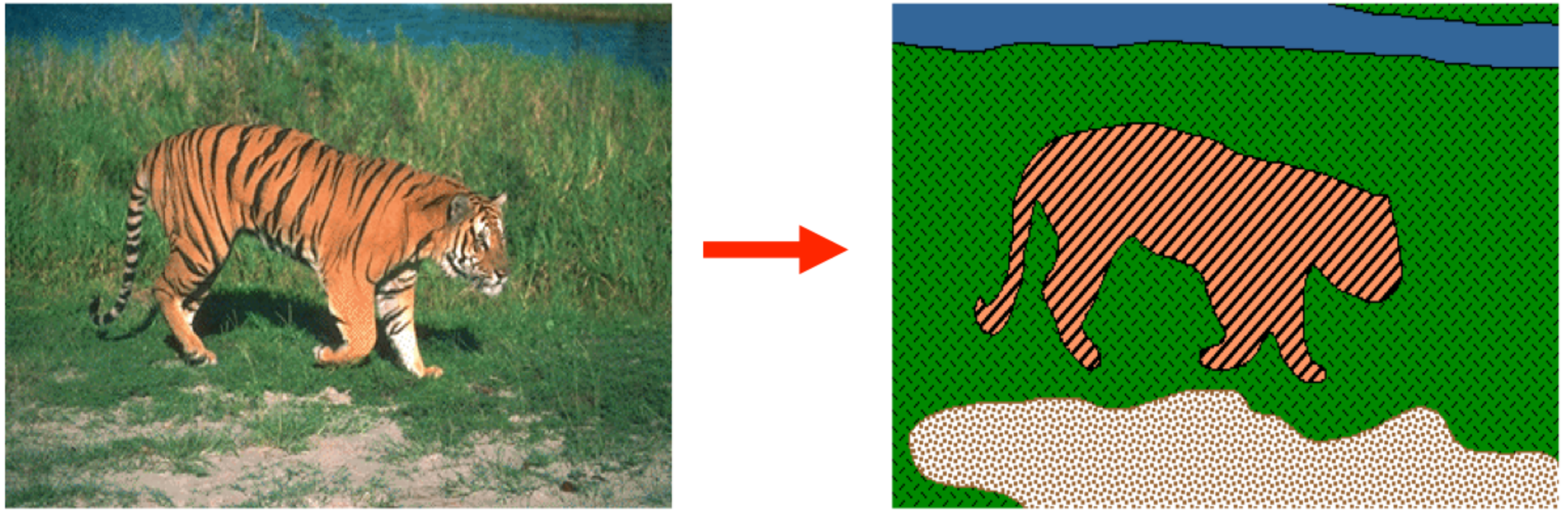
COMP0026: Image Processing

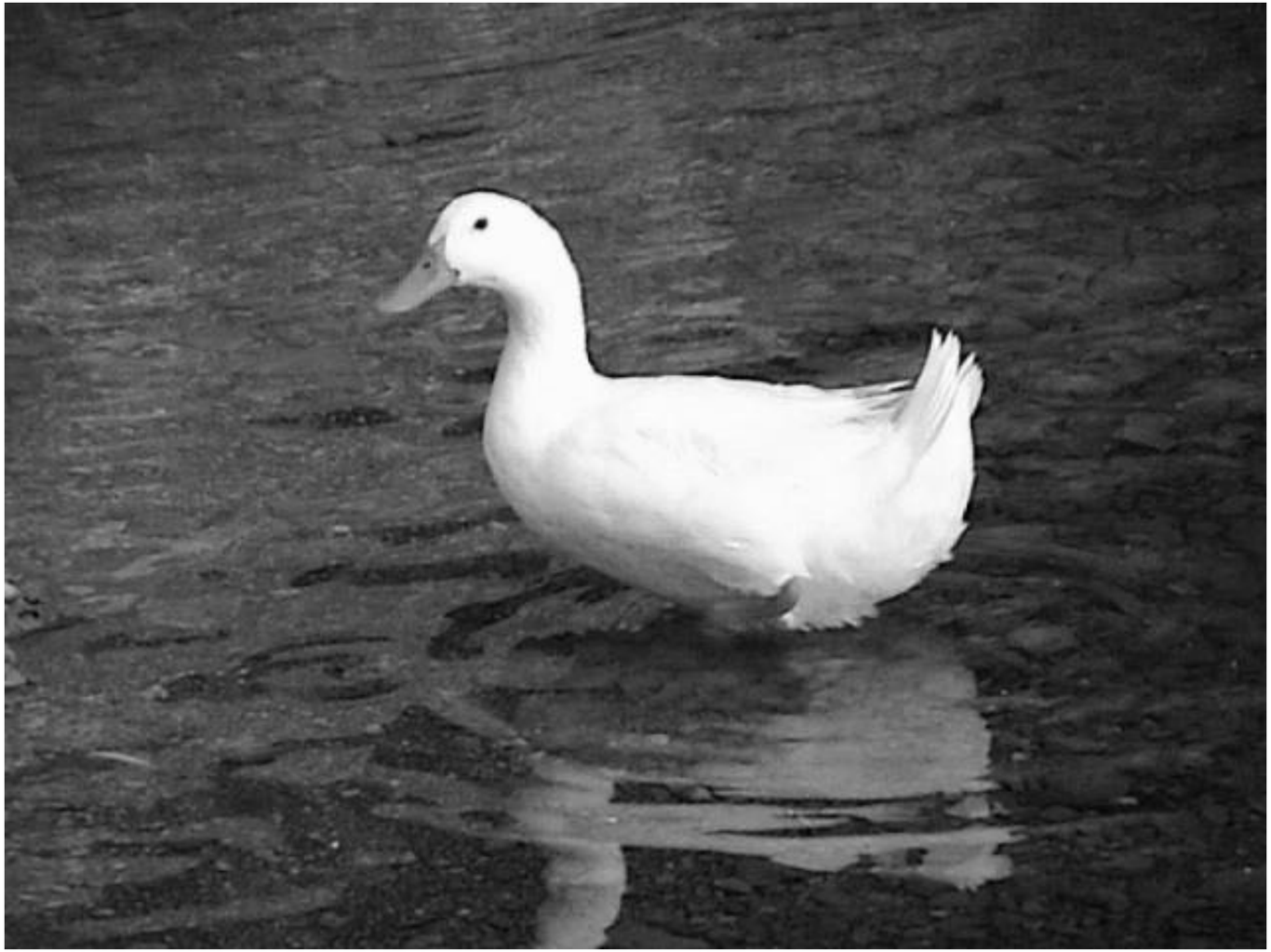
Image Segmentation



UCL

Lectures will be Recorded

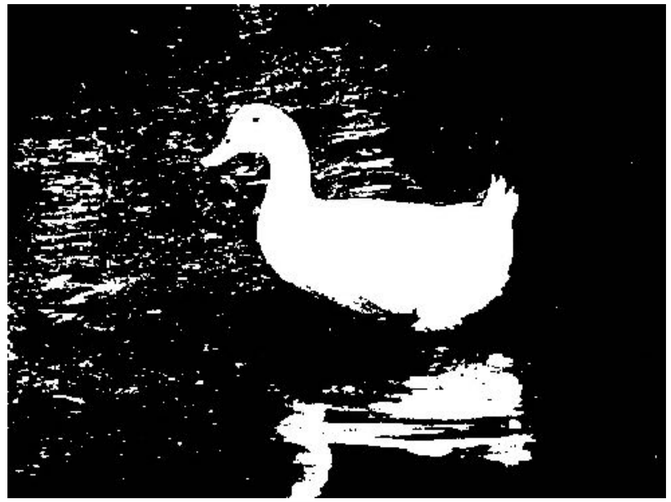
Example



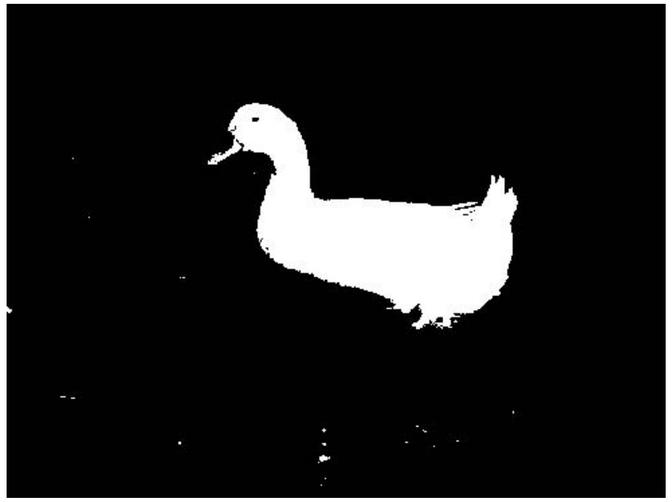
Different Choices for T

T=50

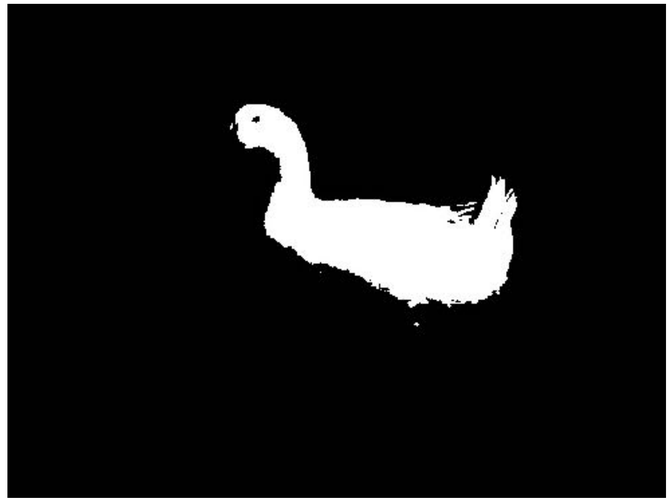
T=100

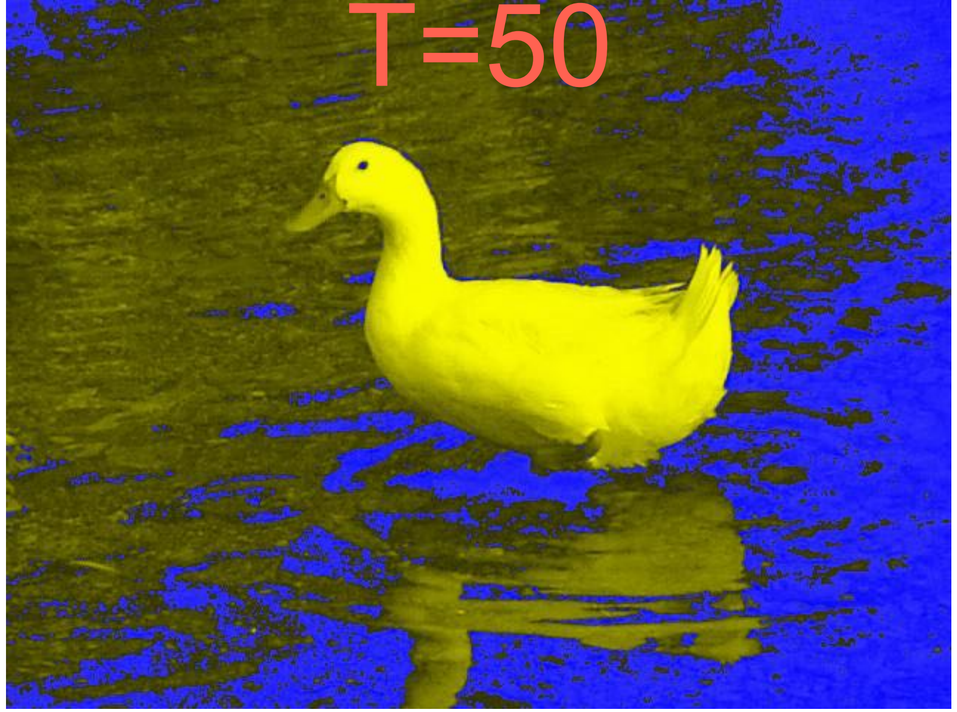


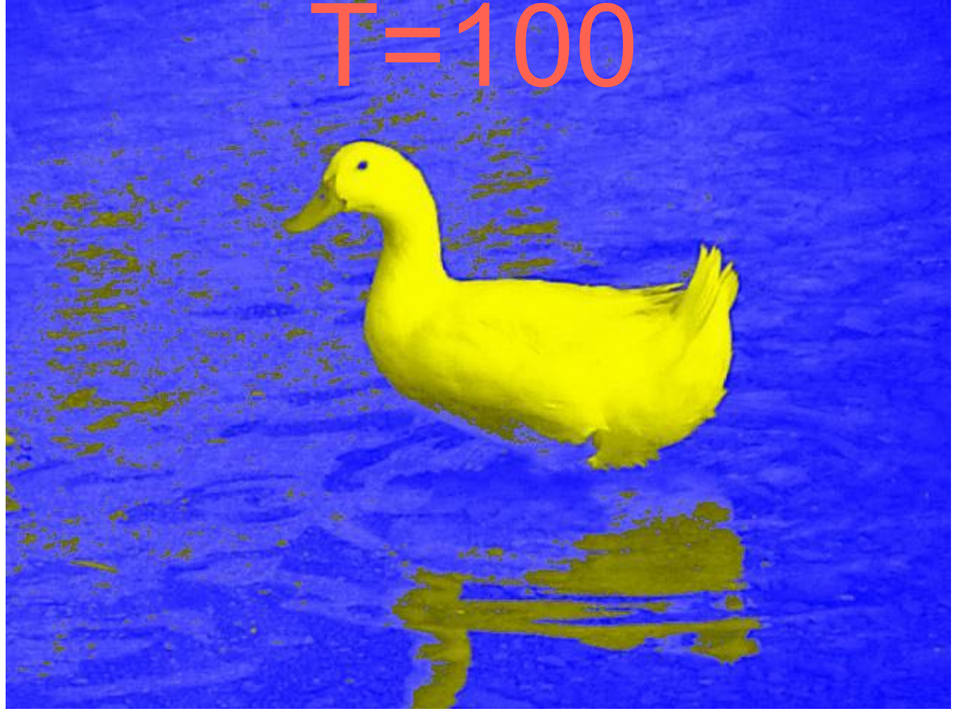
T=150

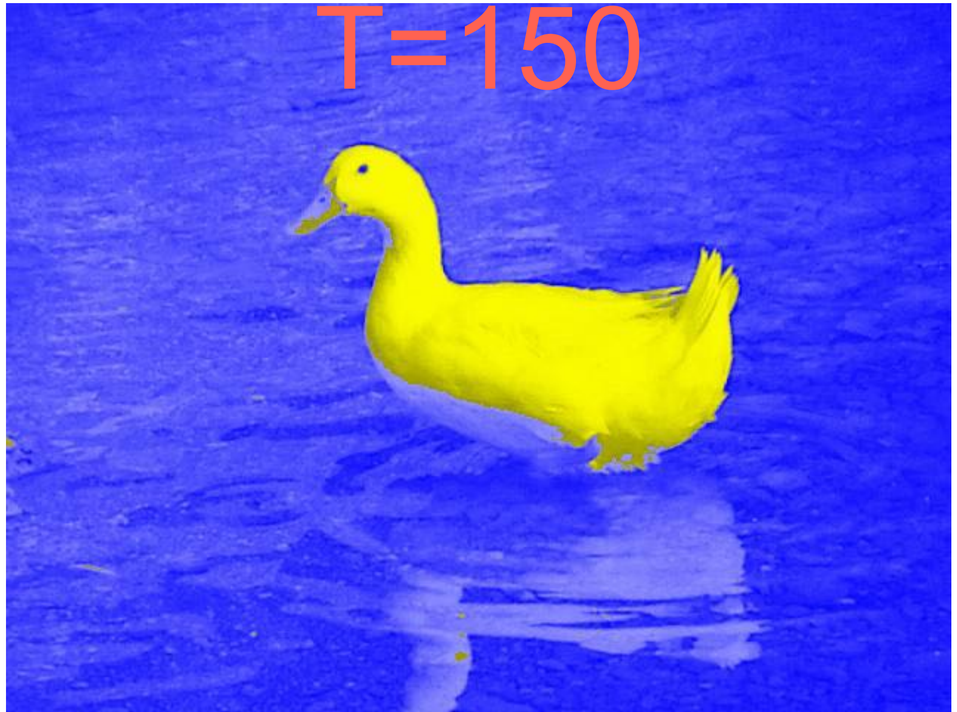


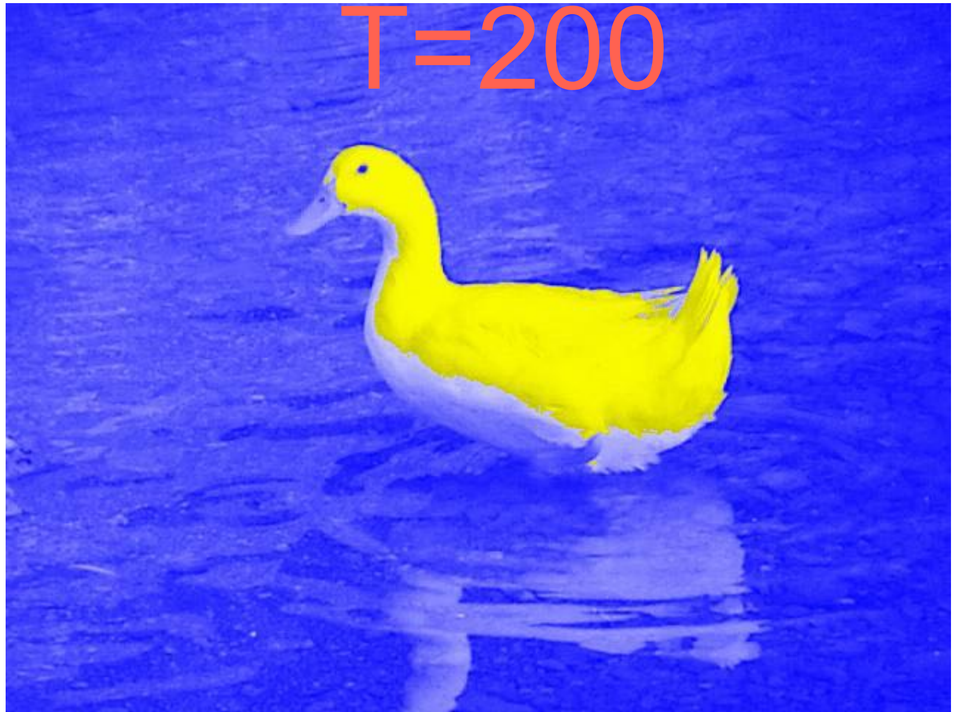
T=200




T=50


T=100

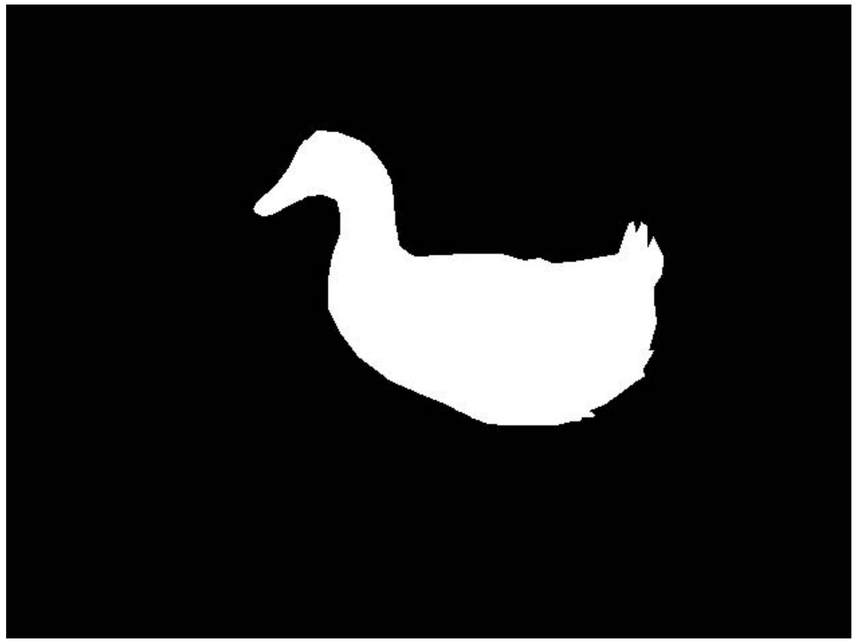

T=150


T=200

Segmentation Performance

- To analyze performance,
  we need to know the true classification of each test.
- We need to do the segmentation by hand on some example images.

ground truth



ROC Analysis

ROC Analysis

- An ROC (receiver operating characteristic) curve characterizes the performance of a binary classifier.

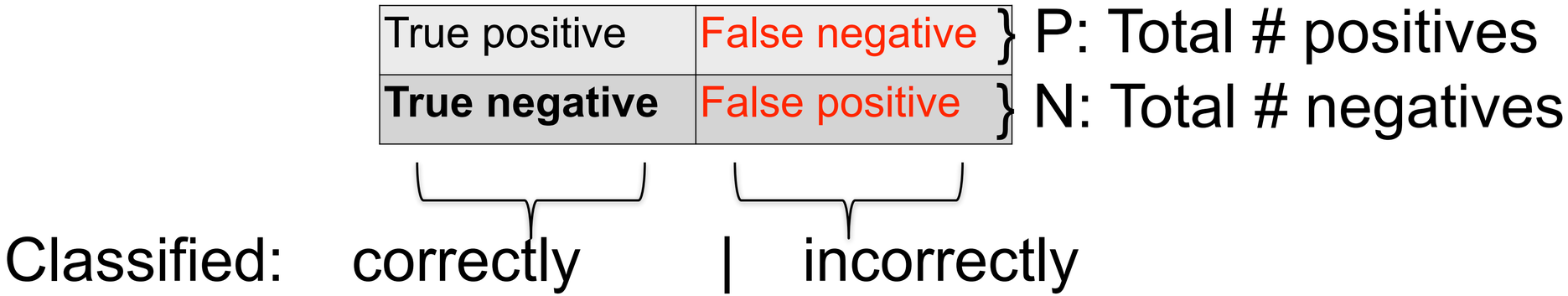
ROC Analysis

- An ROC (receiver operating characteristic) curve characterizes the performance of a binary classifier.

- A binary classifier distinguishes between two different types of thing, e.g.,
  - Healthy/afflicted patients – cancer screening
  - Pregnancy tests
  - Foreground/background image pixels
  - Object detection

Classification Error

- **Binary classifiers make errors.**
- **Two types of input to a binary classifier:**
  - **Positives**
  - **Negatives**
- **Four possible outcomes in any test:**



The ROC Curve

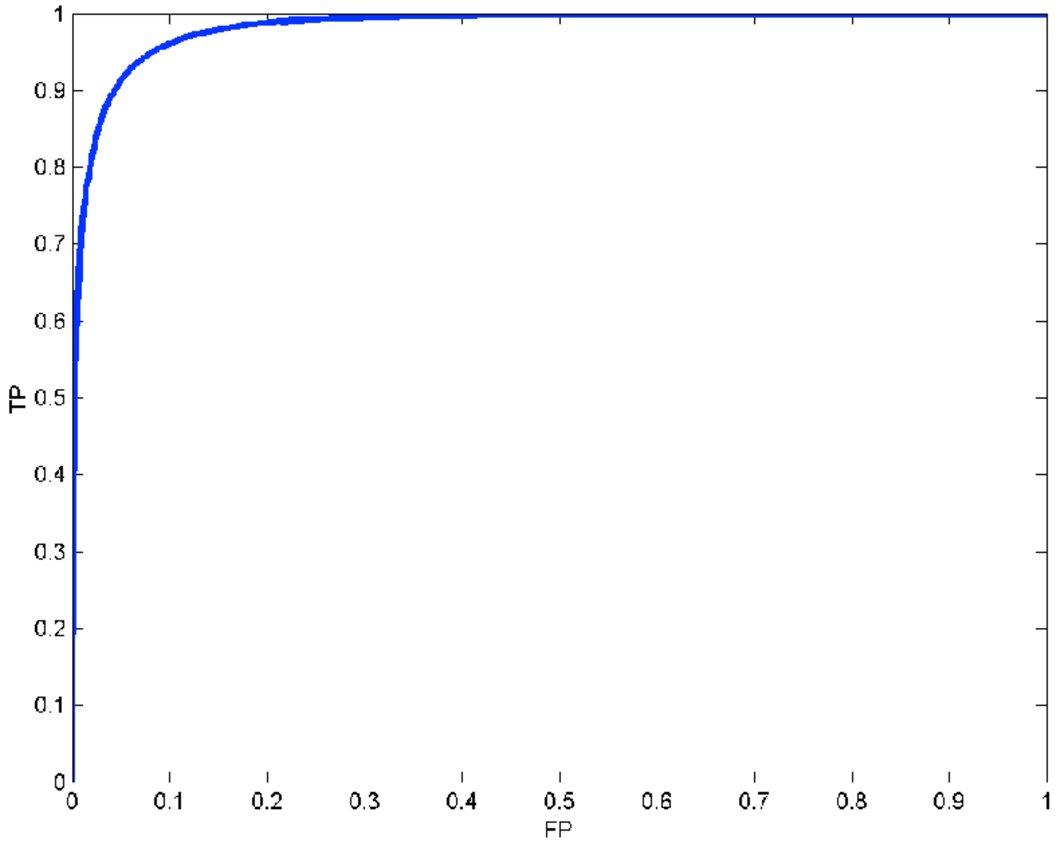
- **Characterizes the error trade-off in binary classification tasks.**
- **It plots the TP fraction against FP fraction.**

- **TP fraction (*sensitivity*) is** $\frac{\text{True positive count}}{P}$

- **FP fraction (1-*specificity*) is** $\frac{\text{False positive count}}{N}$

The ROC Curve

$$\left[\frac{(\text{True positive count})}{P = TP + FN}\right]$$



$$\left[\frac{(\text{False positive count})}{N = FP + TN}\right]$$

Properties of ROC curves

- An ROC curve always passes through (0,0) and (1,1).
- What is the ROC curve of a perfect system?
- What if the ROC curve is a straight line from (0,0) to (1,1)?

Area under the ROC curve

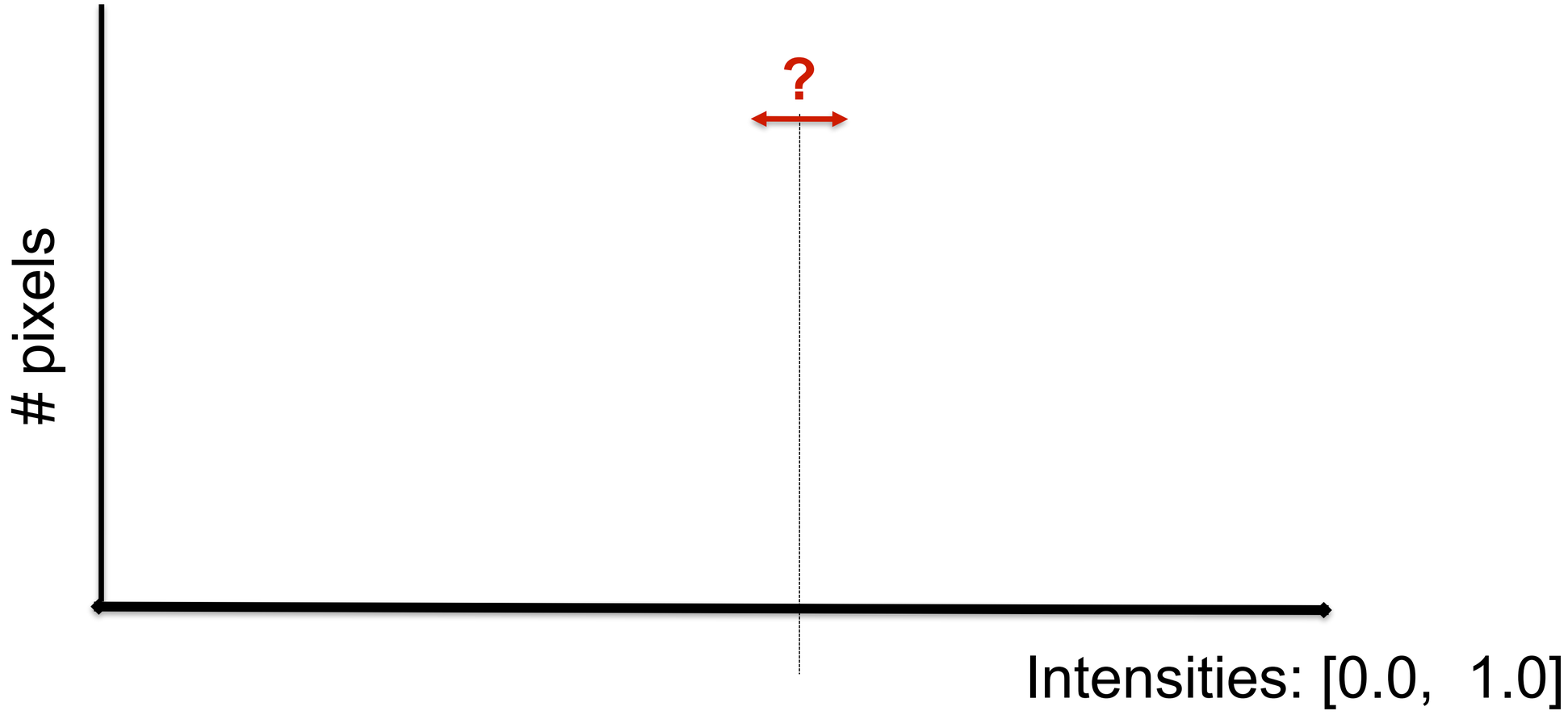
- The area *A* under the curve measures overall classification performance.
- If the distributions of measurements on positive and negative cases can be modeled as Normal distributions (Gaussians) $\mathcal{N}$ with $\mathcal{N}(\mu_P, \sigma_P)$ and $\mathcal{N}(\mu_N, \sigma_N)$, respectively, then

$$A = Z\left(\frac{|\mu_P - \mu_N|}{\sqrt{\mu_P^2 + \mu_N^2}}\right)$$

- *Z* is the cumulative normal distribution function

$$Z(x) = \int_{-\infty}^{x} z(t)dt$$

How to select a Threshold?



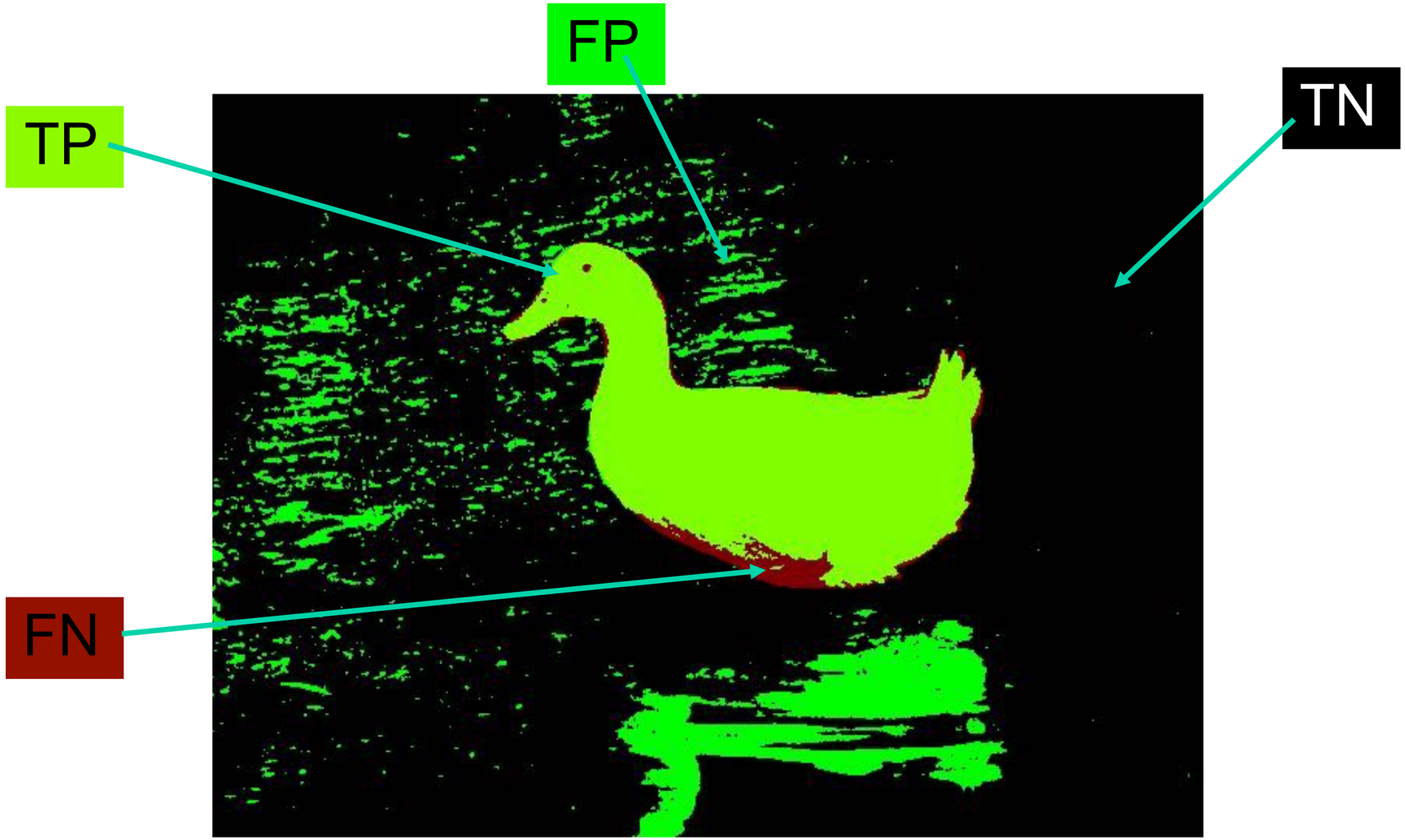
Operating points

- Choose an *operating point* by assigning relative costs and values to each outcome:
  - V_{TN} – value of true negative
  - V_{TP} – value of true positive
  - C_{FN} – cost of false negative
  - C_{FP} – cost of false positive
- Choose the point on the ROC curve with gradient

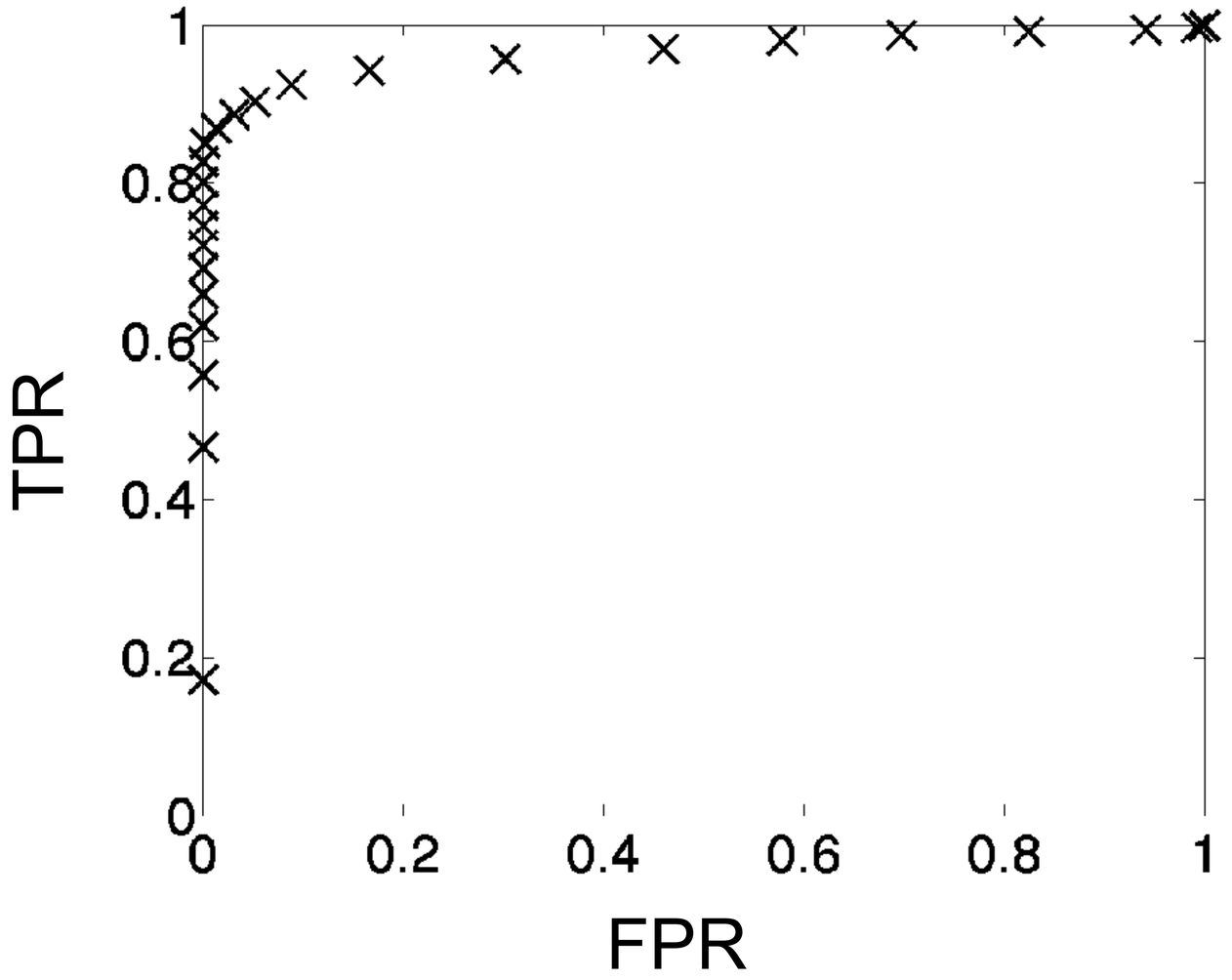
$$\beta = \frac{N}{P}\frac{V_{TN} + C_{FP}}{V_{TP} + C_{FN}}$$

- For simplicity, we often set $V_{TN} = V_{TP} = 0$.

Classification Outcomes



ROC Curve



Limitations of Thresholding

- **Why can we segment images much better by eye than through thresholding processes?**

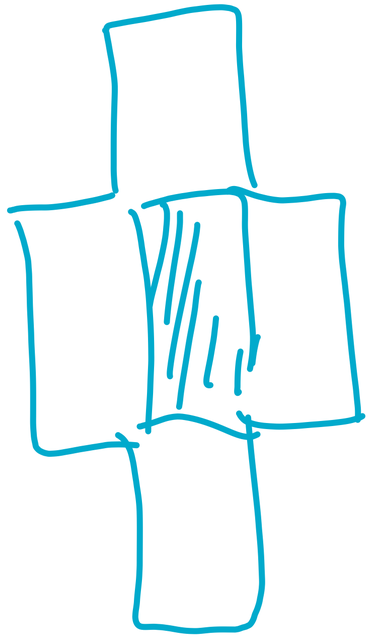
- **We might improve results by considering image context:**
  Local Coherence

Pixel Connectivity

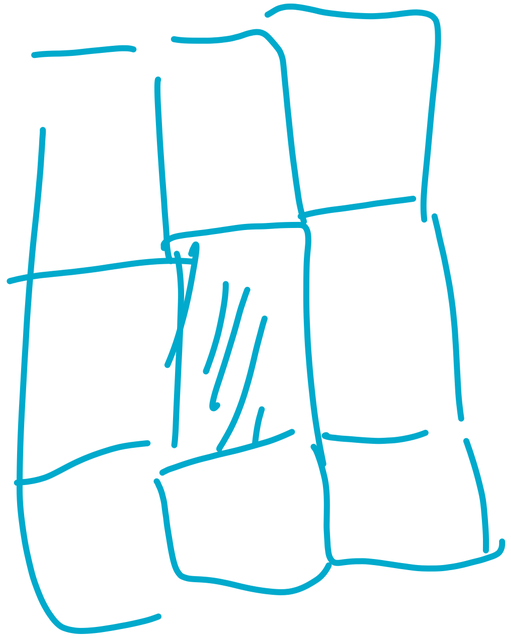
- We need to define which pixels are neighbors.

- Are the dark pixels in this array connected?

Pixel Neighborhoods



4-neighbourhood



8-neighbourhood

Pixel Paths

- A 4-connected path between pixels p_1 and p_n is a set of pixels $\{p_1, p_2, \ldots, p_n\}$ such that p_i is a 4-neighbor of p_{i+1}, $i=1,\ldots,n-1$.

- In an 8-connected path, p_i is an 8-neighbor of p_{i+1}.

Connected Regions

- A region is 4-connected if it contains a 4-connected path between every pair of its pixels.

- A region is 8-connected if it contains an 8-connected path between every pair of its pixels.

Connected Regions

- Now what can we say about the dark pixels in this array?

- What about the light pixels?

Connected Components Labelling

- Labels each connected component of a binary image with a separate number.

1	1	1	1	1	1	1	1	1
1	1	2	2	1	3	1	1	1
1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	1	1	1
1	4	1	1	5	5	1	1	1
1	1	1	1	5	5	5	1	1
6	6	1	1	1	1	5	5	1
7	6	1	1	8	8	1	1	1
7	6	1	1	8	8	1	1	1

Foreground Labelling

- Only extract the connected components of the foreground

1	1	1	1	1	1	1	1	1
1	1	0	0	1	0	1	1	1
1	1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1	1
1	0	1	1	0	0	1	1	1
1	1	1	1	0	0	0	1	1
0	0	1	1	1	1	0	0	1
2	0	1	1	0	0	1	1	1
2	0	1	1	0	0	1	1	1

Connected Components

```
% B is the binary image input. L is the labelled
% image output.
function L = ConnectedComponents(B)
[X,Y] = size(B);
L = zeros(X,Y);
n=1;
For each (x,y) in B
   if(B(x,y) & L(x,y)==0)
     label(x,y,n,B,L);
      n=n+1
   end
end
```

is foreground

Connected Components

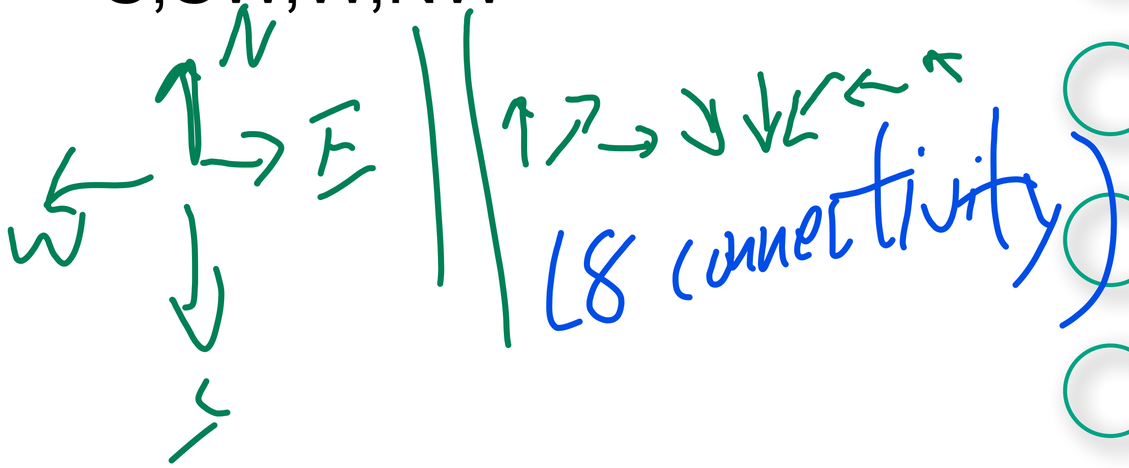
```
function label(x_start, y_start, n, B, L)
% Recursively give label n to this pixel
% and all its foreground neighbours.
L(x_start,y_start)=n;
For each (x,y) in N(x_start, y_start)
   if(L(x,y)==0 & B(x, y))
      label(x,y,n,B,L);
   end
end
```

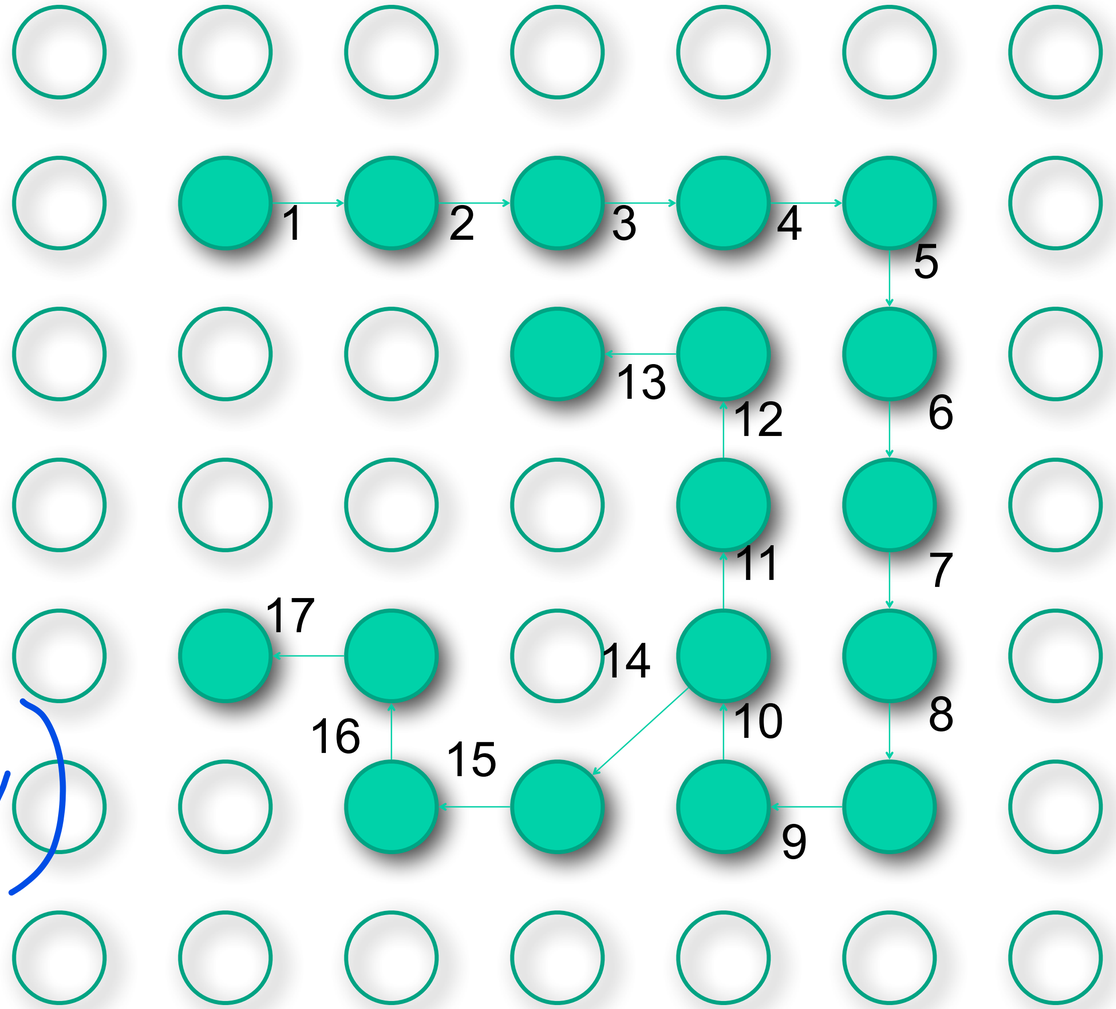
look its neighbour
↳ if neighbour && hasn't labeled

DFS

Processing Order

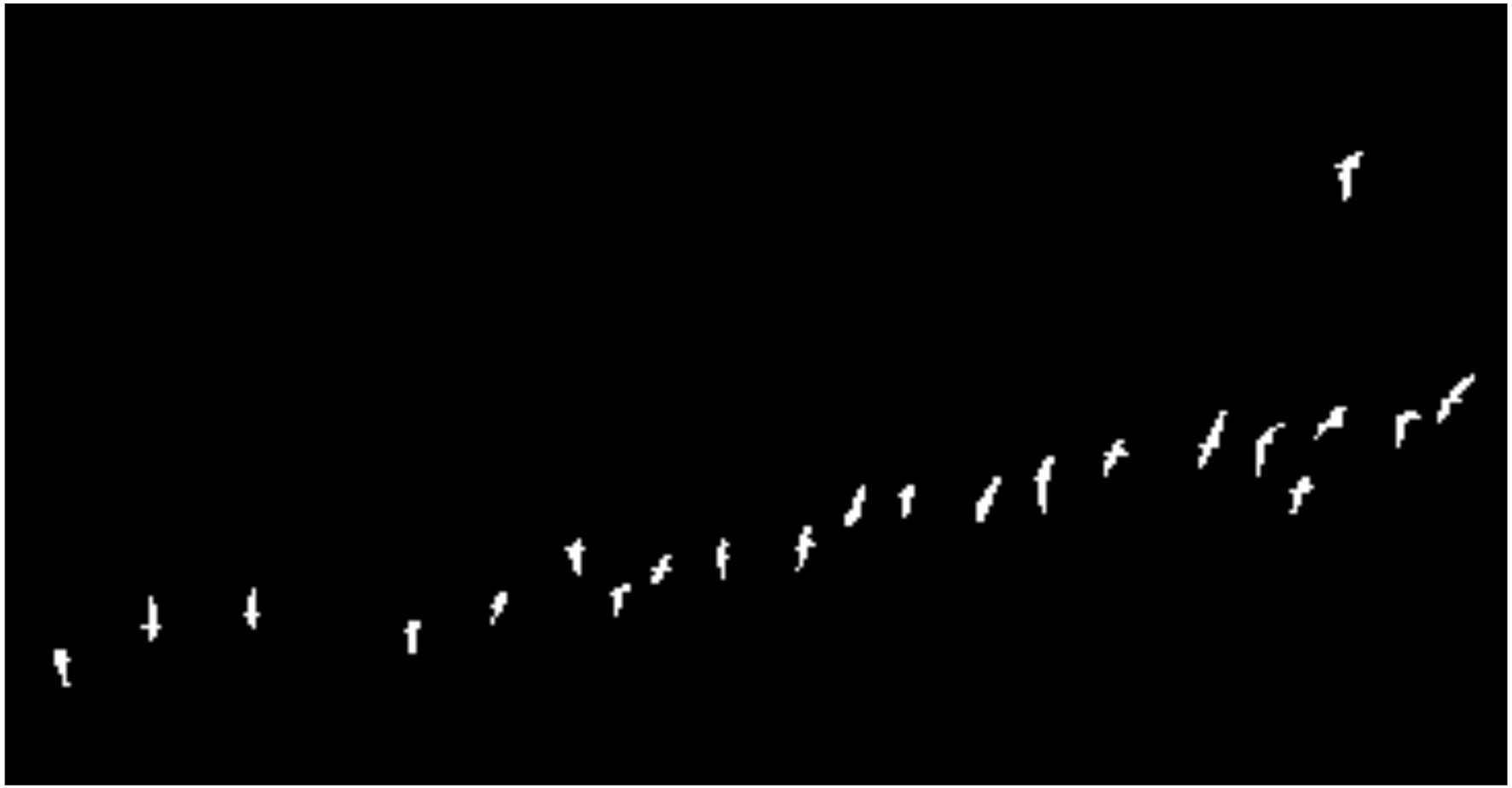
Order of processing nodes with recursive algorithm (**depth first** search). Start in top-left corner and search in order N,NE,E,SE, S,SW,W,NW



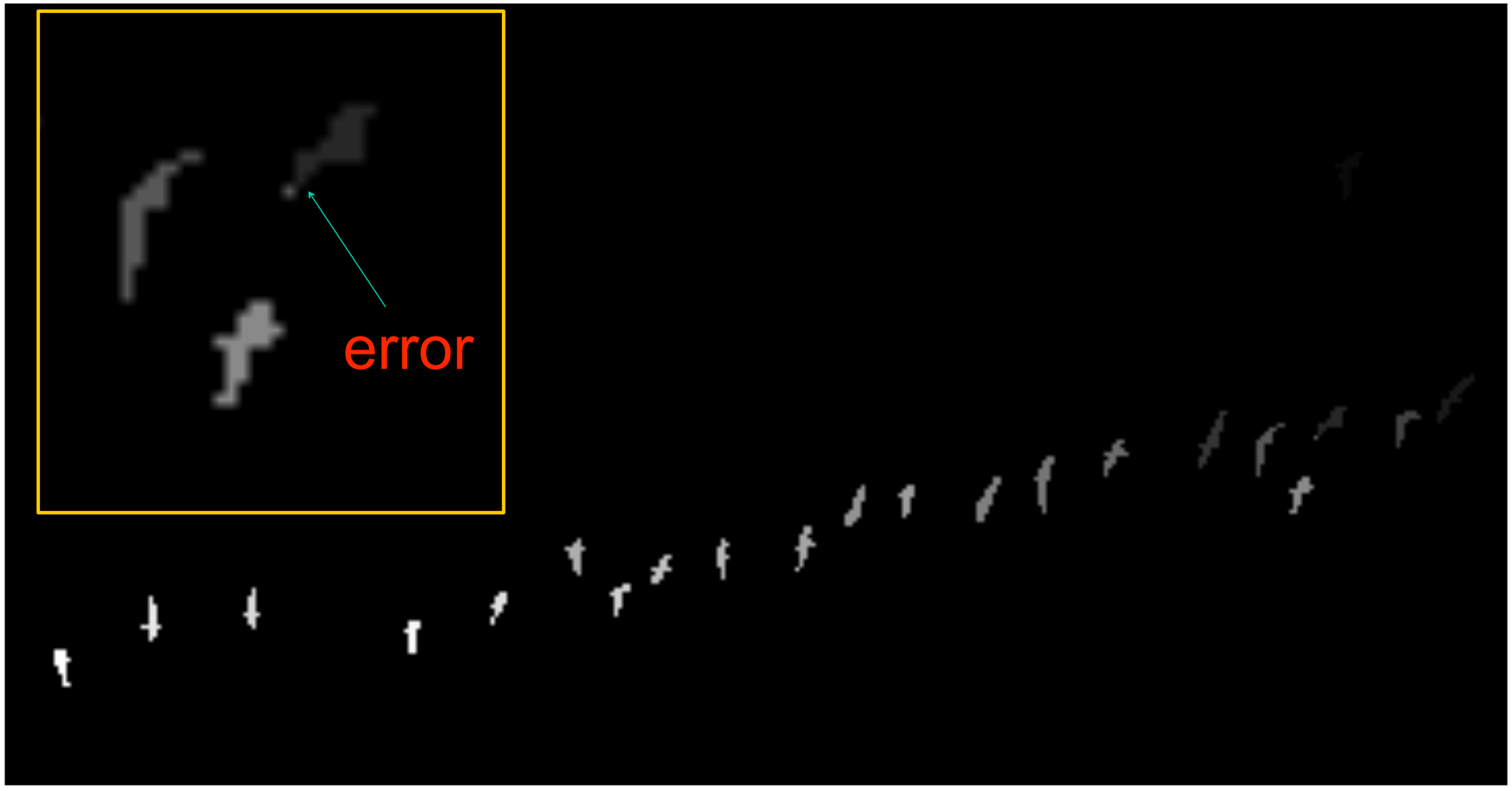


Goose Detector

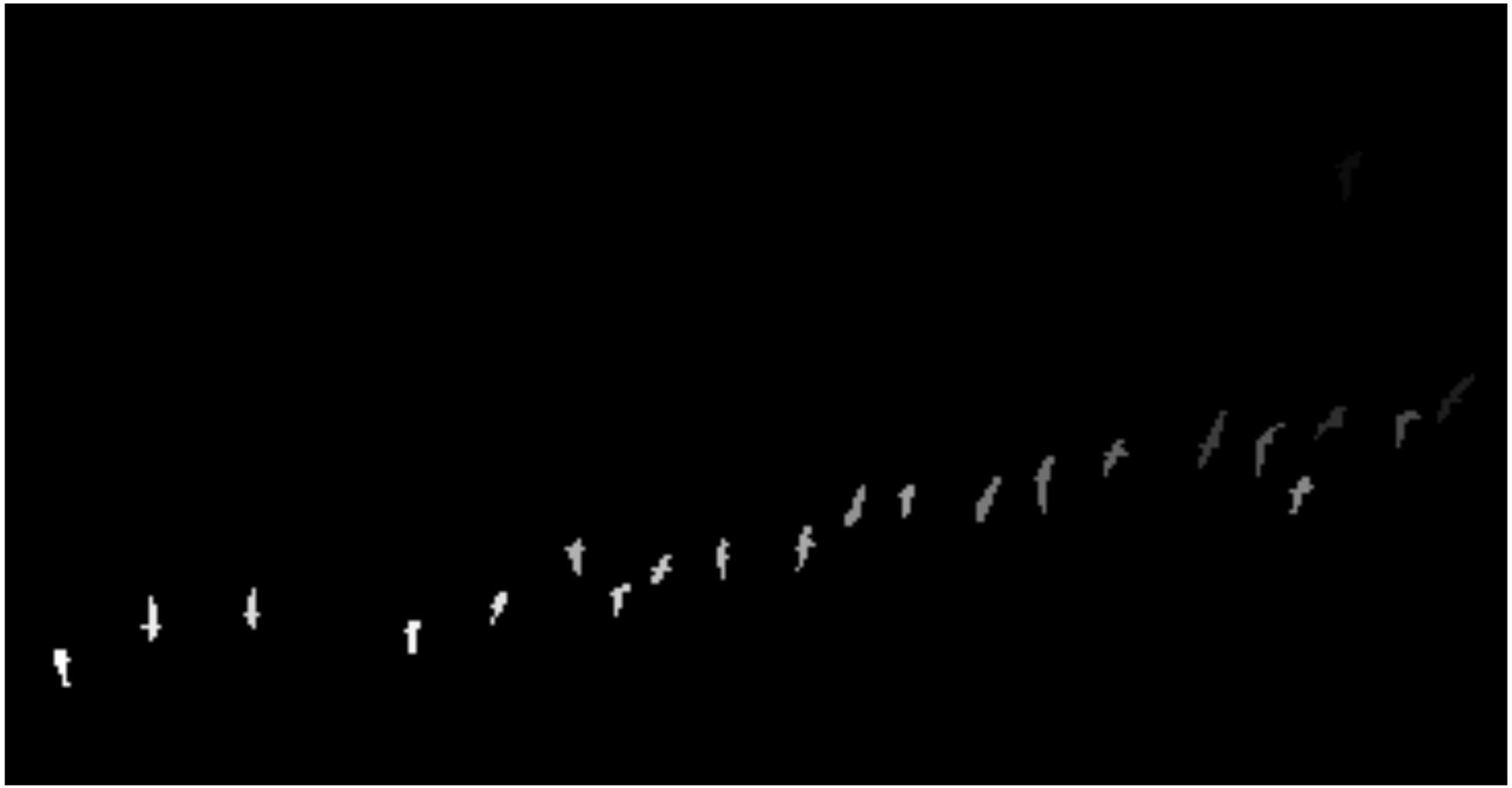
Goose Detector



Goose 4-components (26)



Goose 8-components (22)

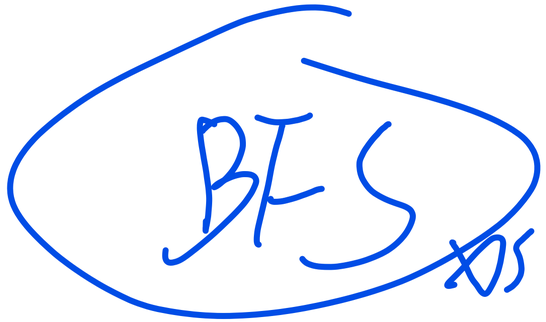


Connected Components

- **What happens if we use the connected components algorithm on this image?**

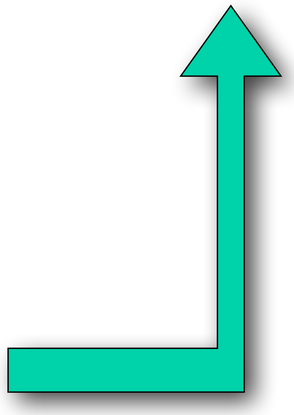
Region Growing

- Start from a seed point or region.
- Add neighboring pixels that satisfy the criteria defining a region.
- Repeat until we can include no more pixels.



Region Growing

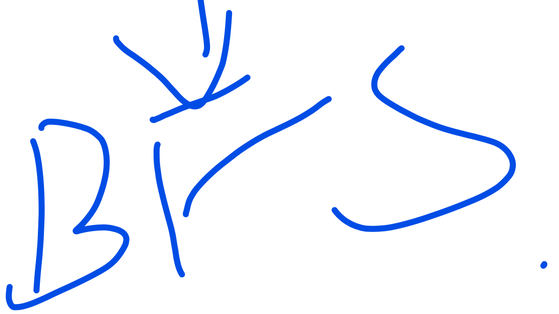
```
function B = RegionGrow(I, seed)

    [X,Y] = size(I);
    visited = zeros(X,Y);
    visited(seed) = 1;
    boundary = emptyQ;
    boundary.enQ(seed);
```

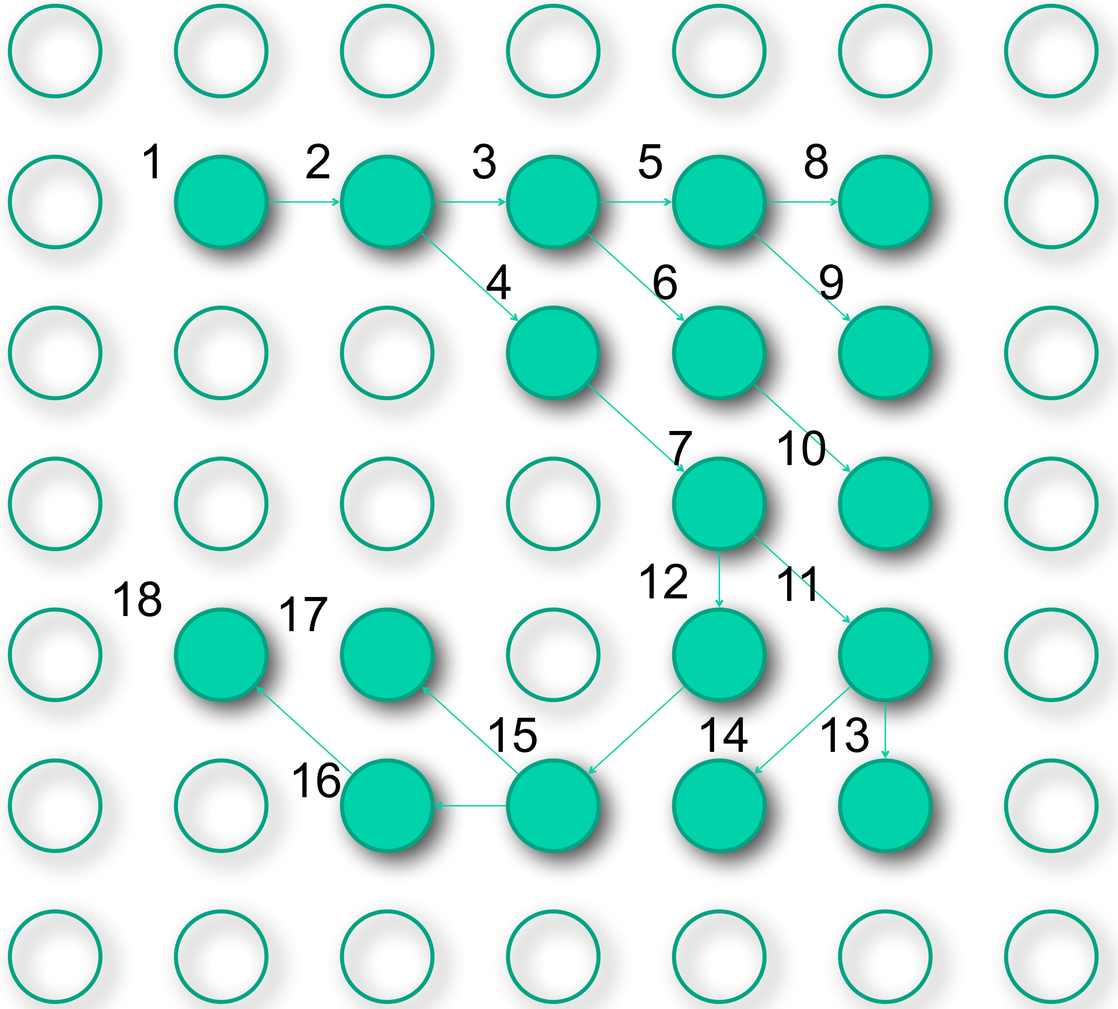


```
while(~boundary.empty())
    nextPoint = boundary.deQ();
    if(include(nextPoint))
        visited(nextPoint) = 2;
        Foreach (x,y) in N(nextPoint)
            if(visited(x,y) == 0)
                boundary.enQ(x,y);
                visited(x,y) = 1;
            end
        end
    end
end
```

Connected Components

Order of processing nodes with **breadth first** search. Start in top-left corner and search in order N,NE,E,SE, S,SW,W,NW

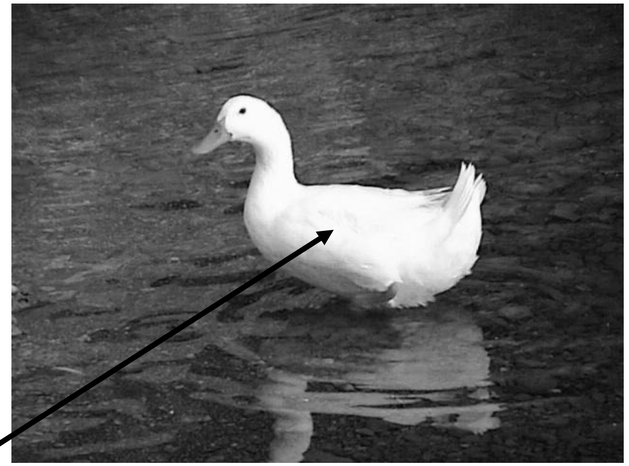




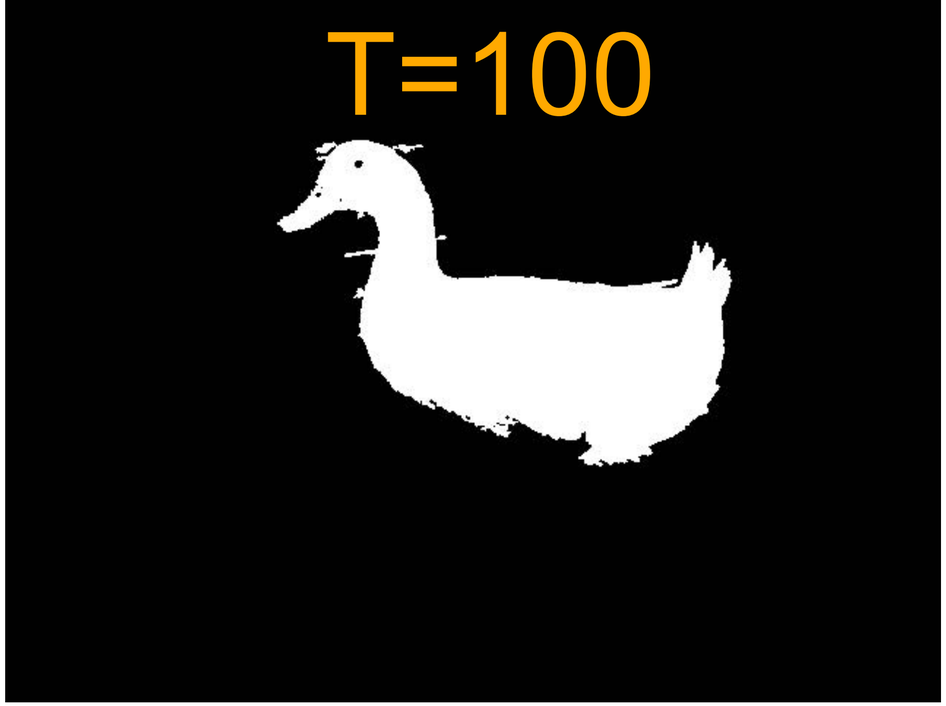
Region Growing Example

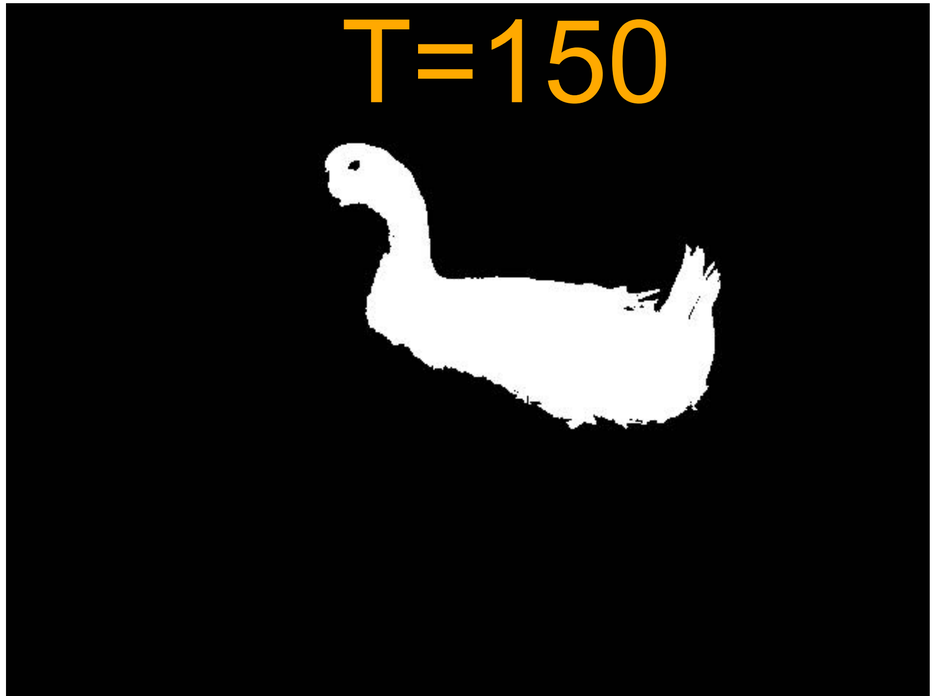
- **Pick a single seed pixel.**
- **Inclusion test is a simple grey level threshold:**

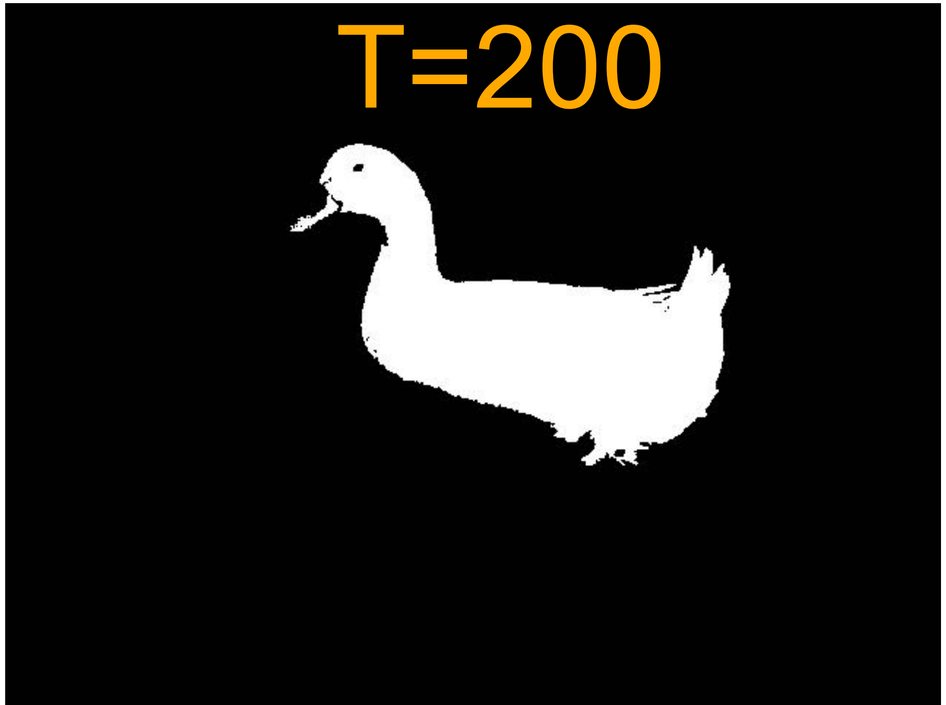
```
function test = include(p)
    test = (p>=T);
```

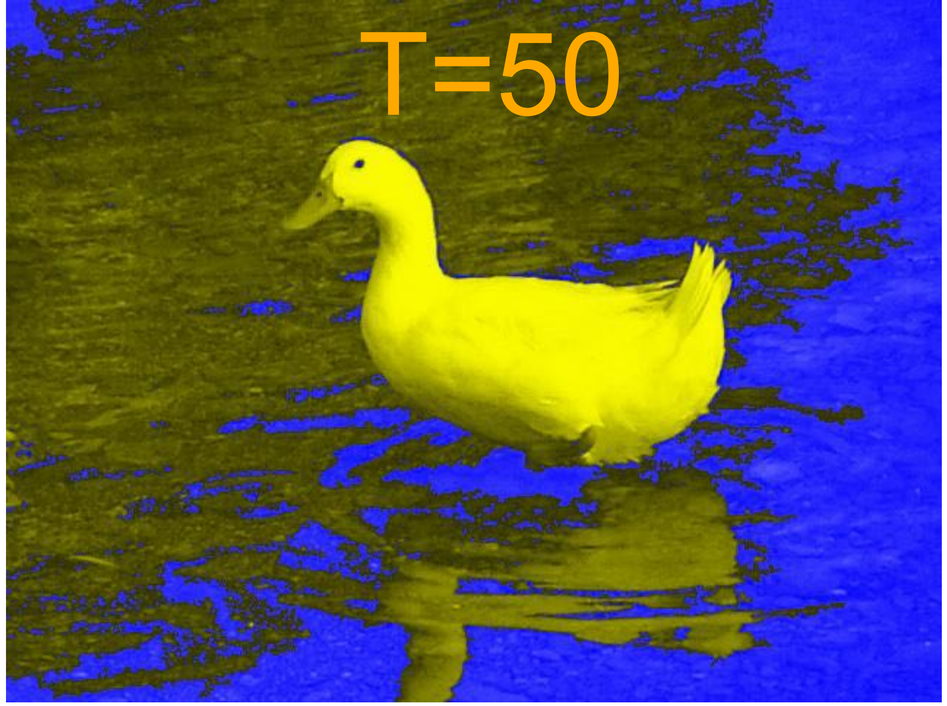


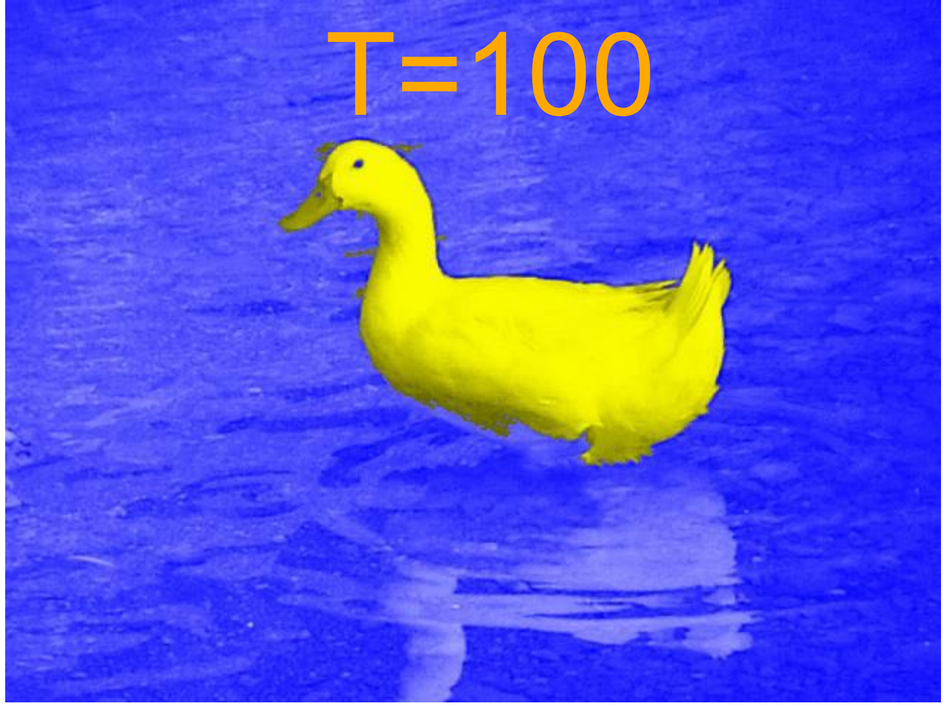
Seed pixel

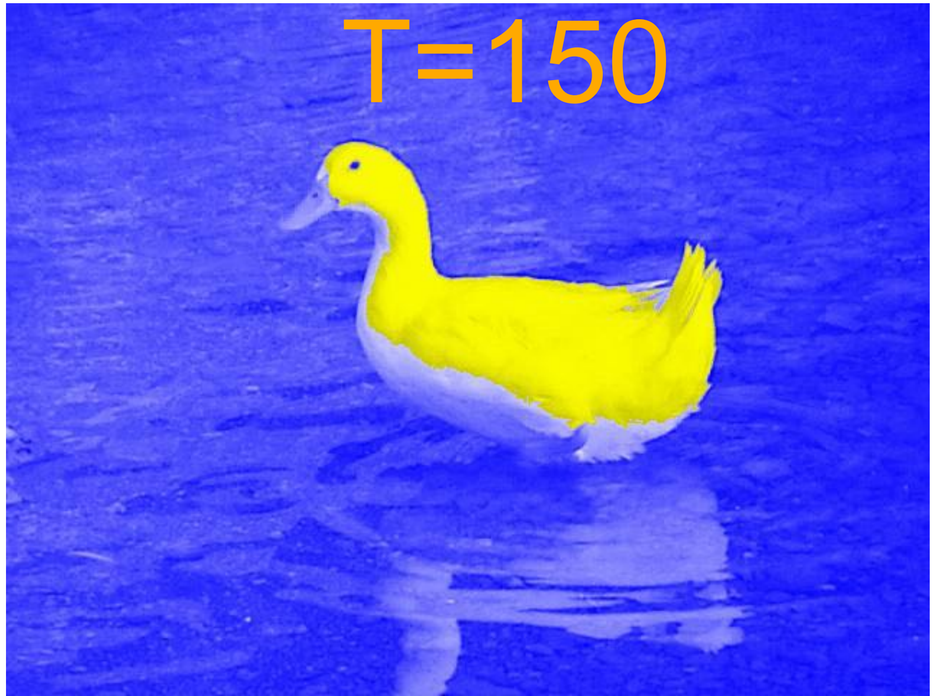

T=100

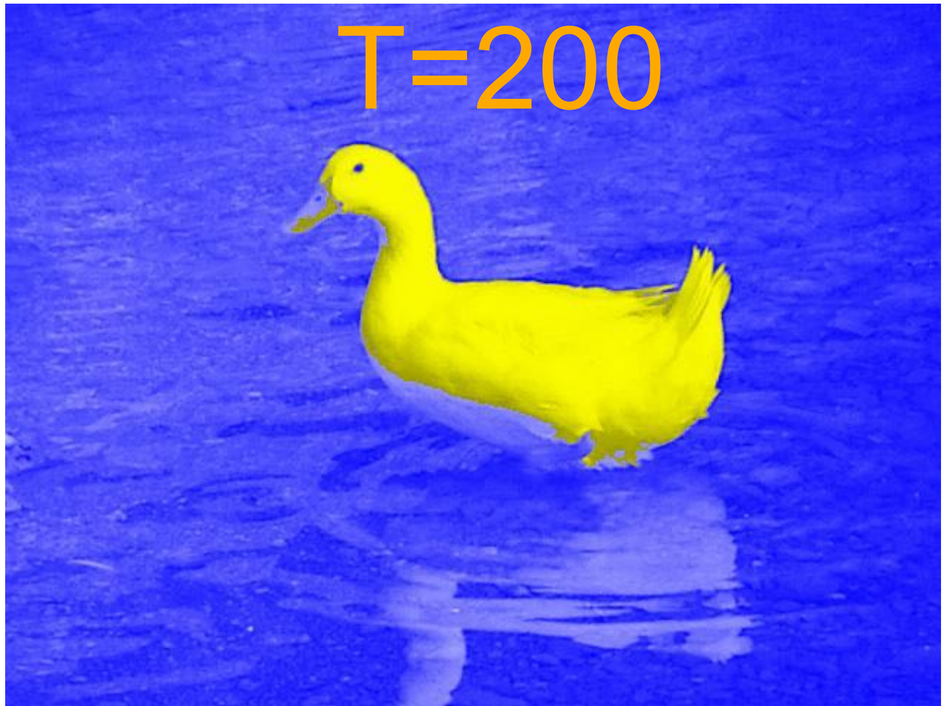

T=150


T=200

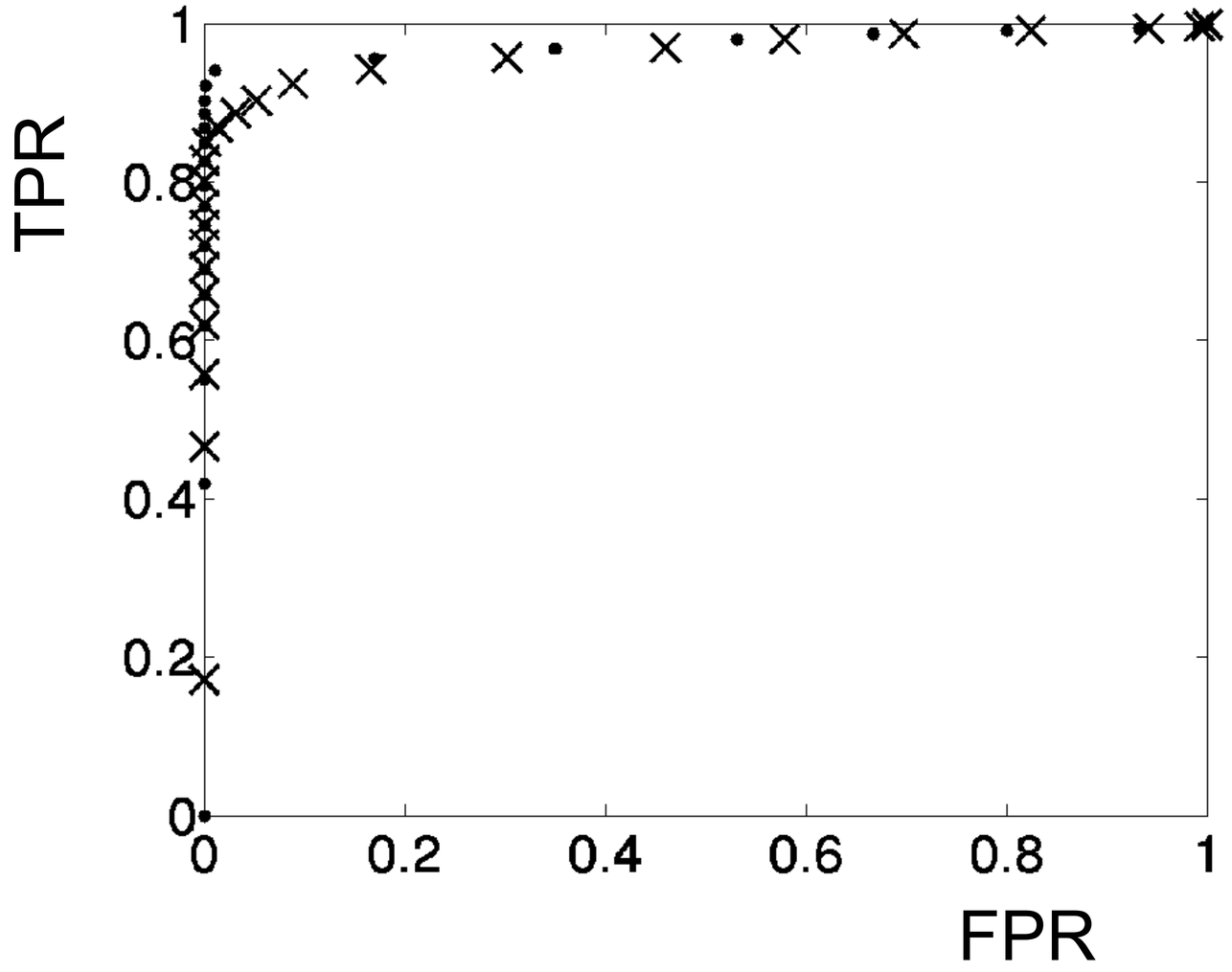

T=50


T=100


T=150


T=200

ROC Curve



- Region growing
- × Thresholding

Implementation

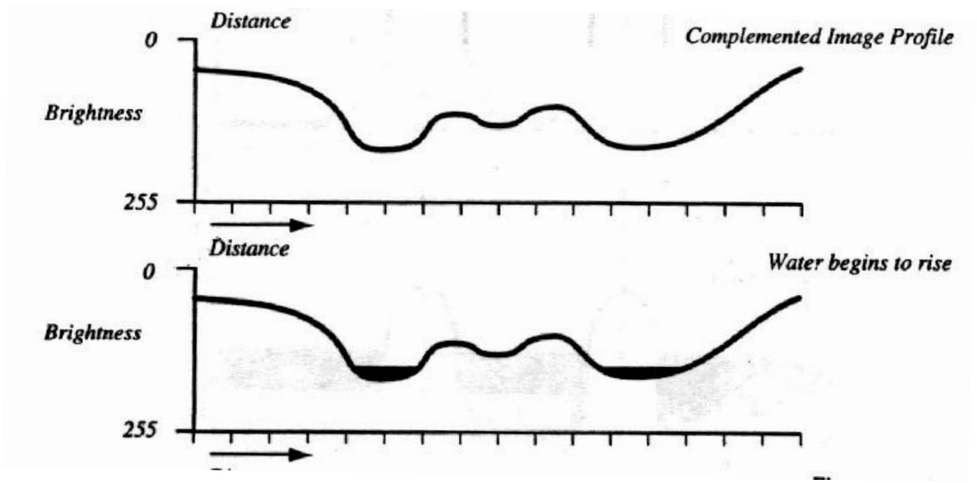
- The region-growing algorithm above uses a *breadth-first search*.
- The connected-components algorithm above uses a *depth-first search*.
- Both algorithms can use either search procedure.

Watershed Segmentation Algorithm

- The objective is to find watershed lines.
- Suppose that a hole is punched in each regional minimum and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate.
- When rising water in distinct catchment basins is about to merge, a dam is built to prevent merging. These dam boundaries correspond to the watershed lines.

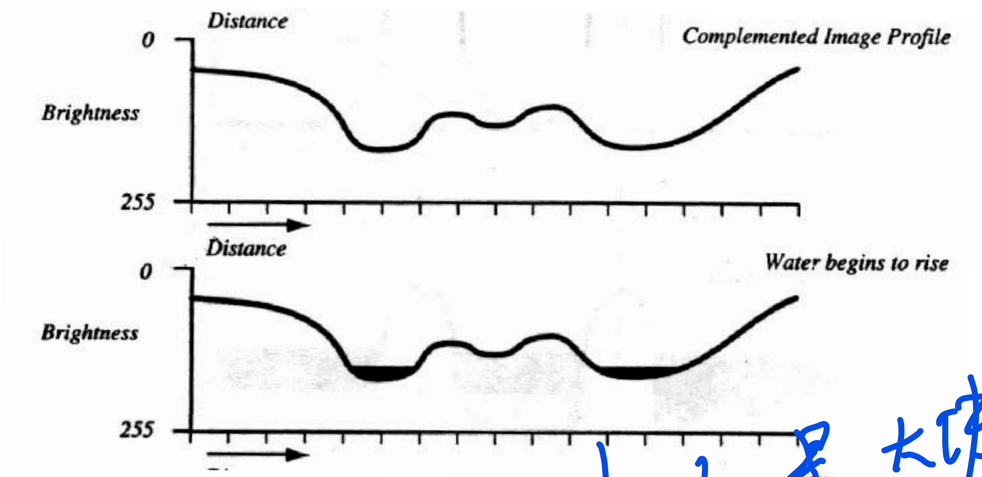
Watershed Segmentation Algorithm

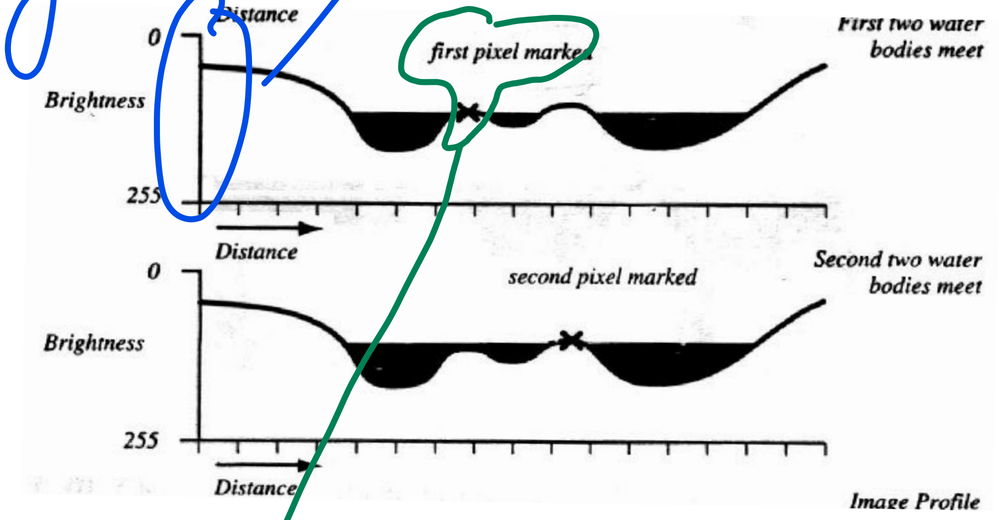
- The objective is to find watershed lines.
- Suppose that a hole is punched in each regional minimum and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate.
- When rising water in distinct catchment basins is about to merge, a dam is built to prevent merging. These dam boundaries correspond to the watershed lines.



Watershed Segmentation Algorithm

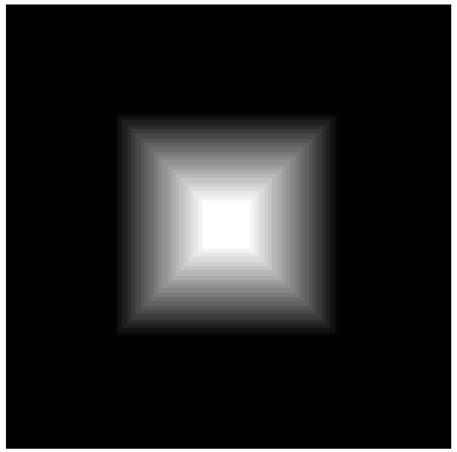
- The objective is to find watershed lines.
- Suppose that a hole is punched in each regional minimum and that the entire topography is flooded from below by letting water rise through the holes at a uniform rate.
- When rising water in distinct catchment basins is about to merge, a dam is built to prevent merging. These dam boundaries correspond to the watershed lines.

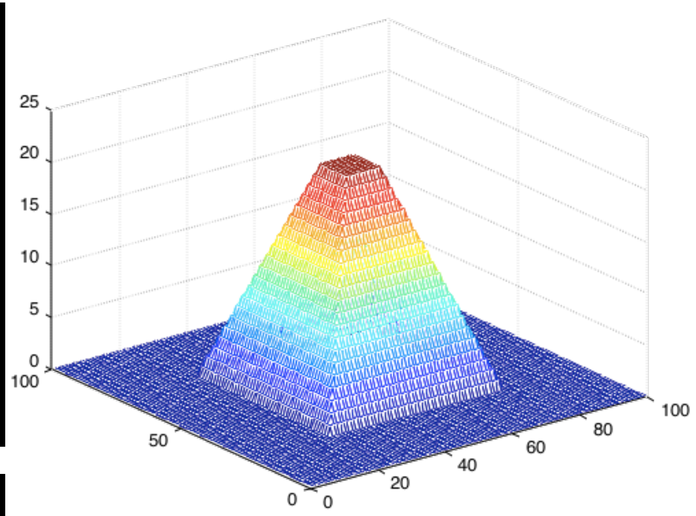


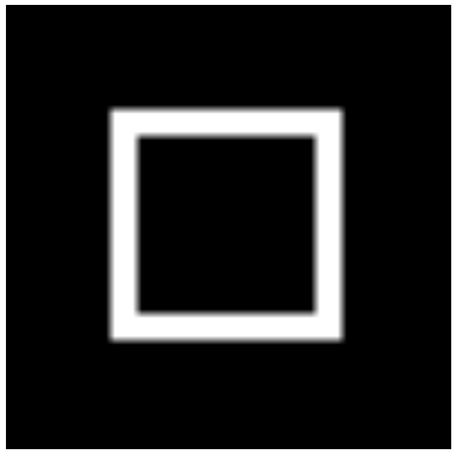


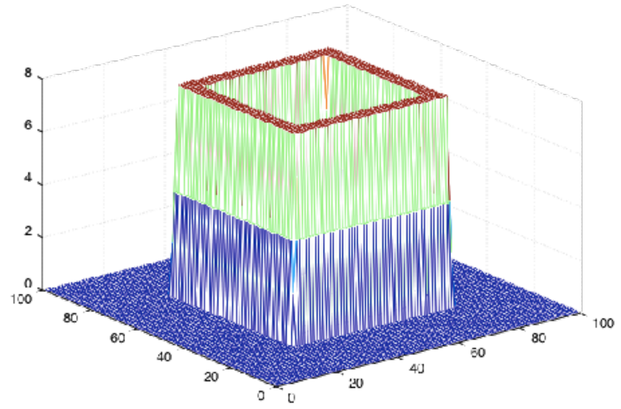
Watershed Segmentation

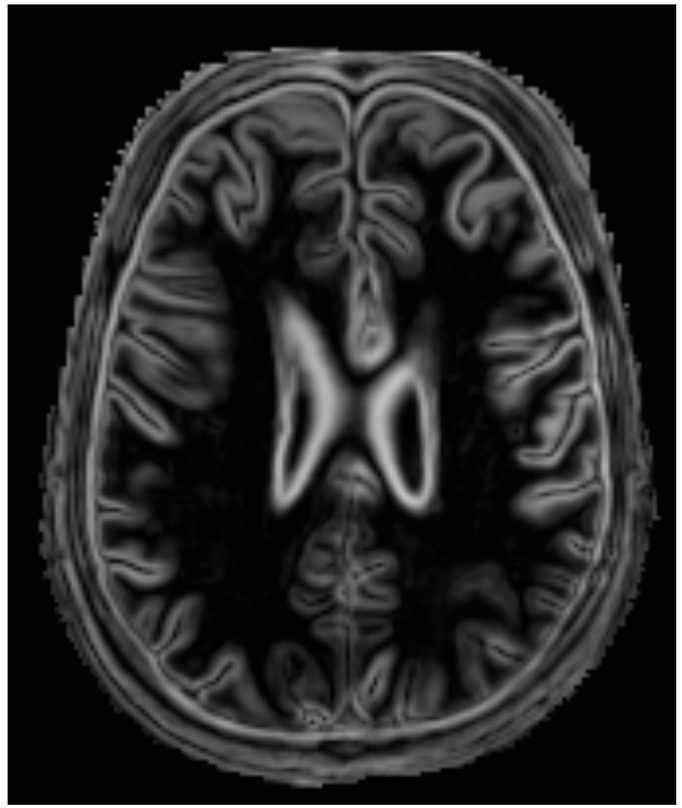
In this case, each object is distinguished from the background by its raised edges.









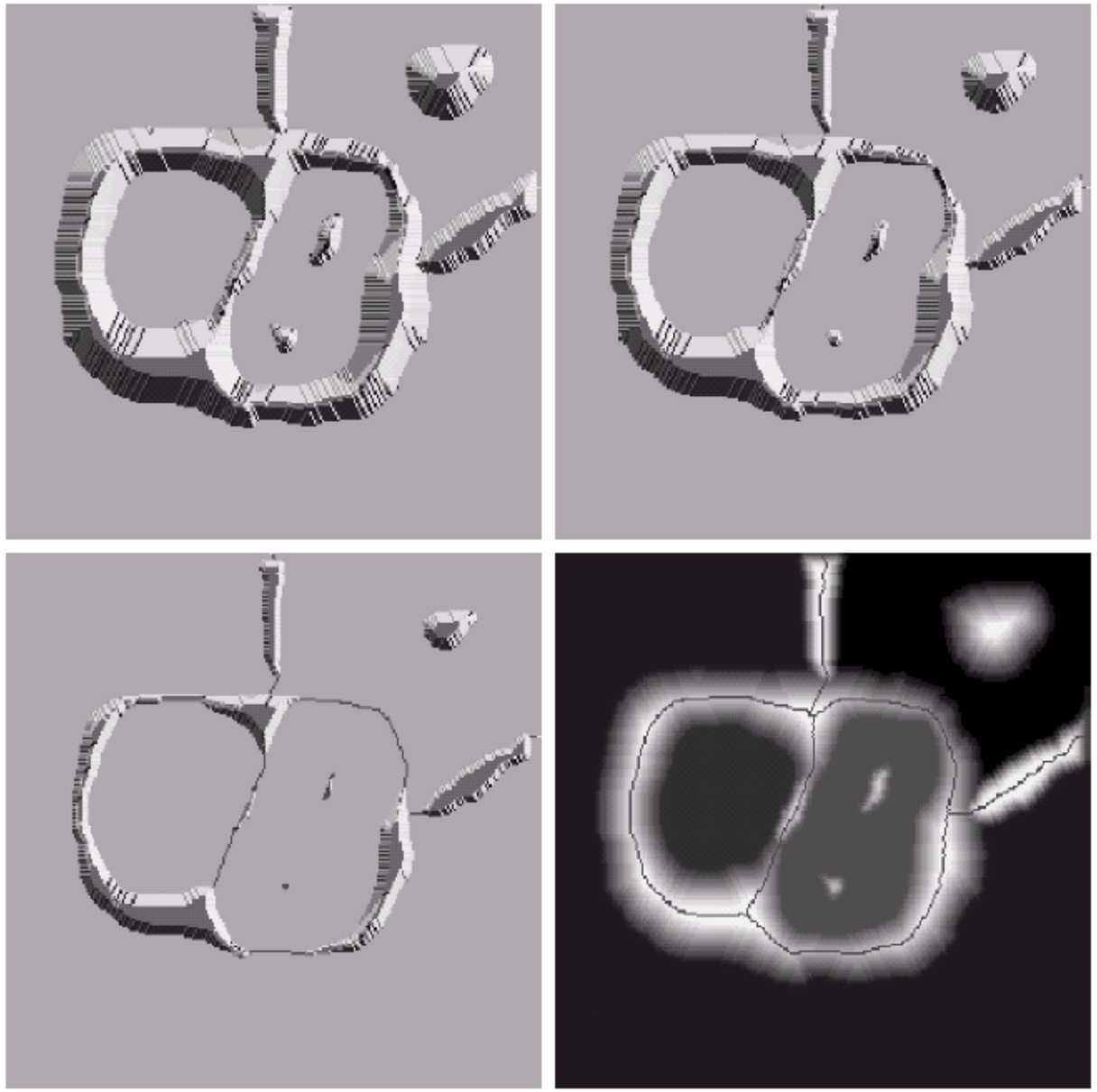


Basic Steps

1. Piercing holes in each regional minimum of I

2. The 3D topography is flooded from below gradually

3. When the rising water in distinct catchment basins is about to merge, a dam is built to prevent the merging



e f
g h

FIGURE 10.44
(Continued)
(e) Result of further flooding. (f) Beginning of merging of water from two catchment basins (a short dam was built between them). (g) Longer dams. (h) Final watershed (segmentation) lines. (Courtesy of Dr. S. Beucher, CMM/Ecole des Mines de Paris.)

4. The dam boundaries correspond to the watershed lines to be extracted by a watershed segmentation algorithm

Watershed Algorithm

priority-queue

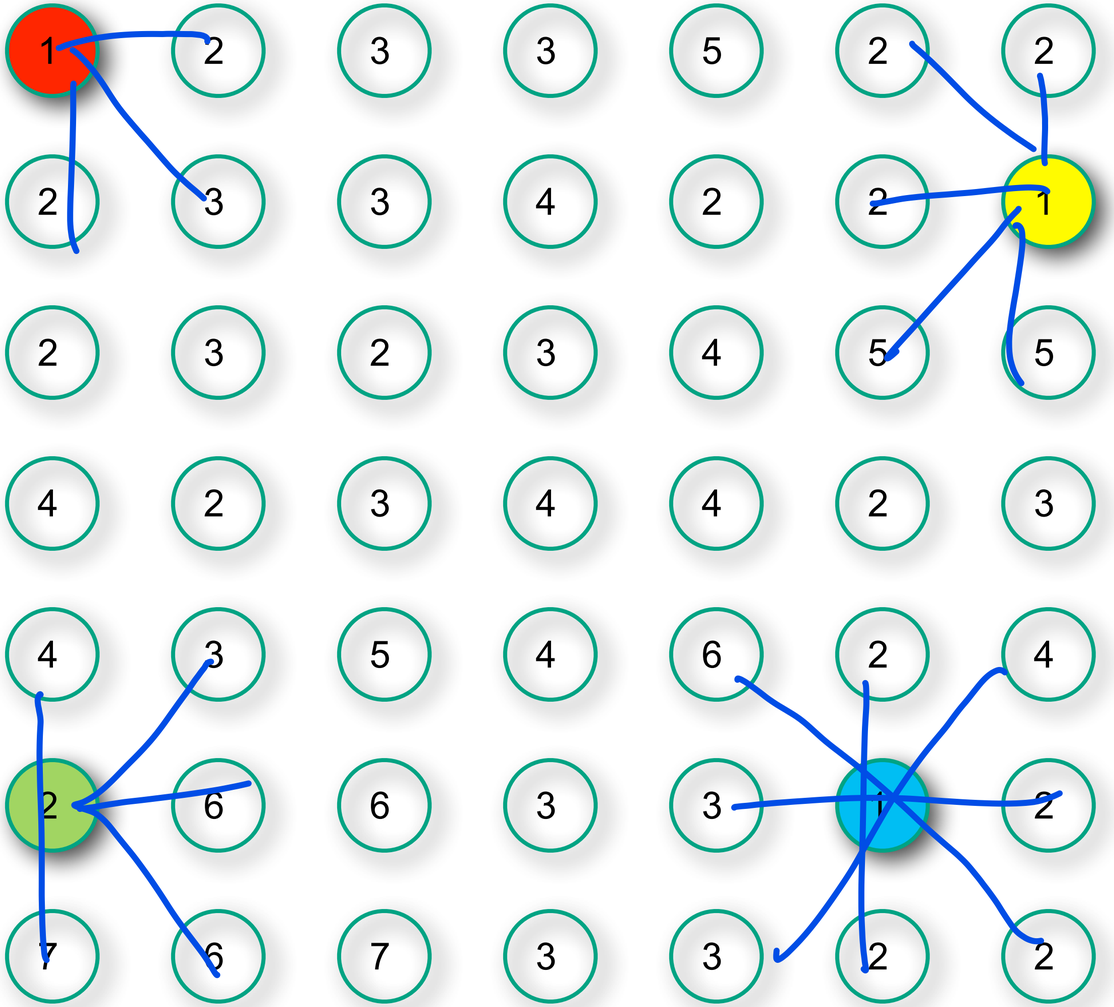
```
Set label at every pixel to -1

For each seed point u
    For each neighbour v of u
        Label[v] = label [u]
        Add v to priority queue: priority by gray level

while(elements in queue)
    u = popMinimum(Queue);
    For each neighbour v of u
    If (label[v] ==-1)
        Label[v] = label [u]
        Add v to priority queue: priority by gray level
```

Example

Watershed algorithm. Search simultaneously from seeds in order determined by gray level.

Direction Order:
N,NE,E,SE,
S,SW,W,NW

start at low intensity

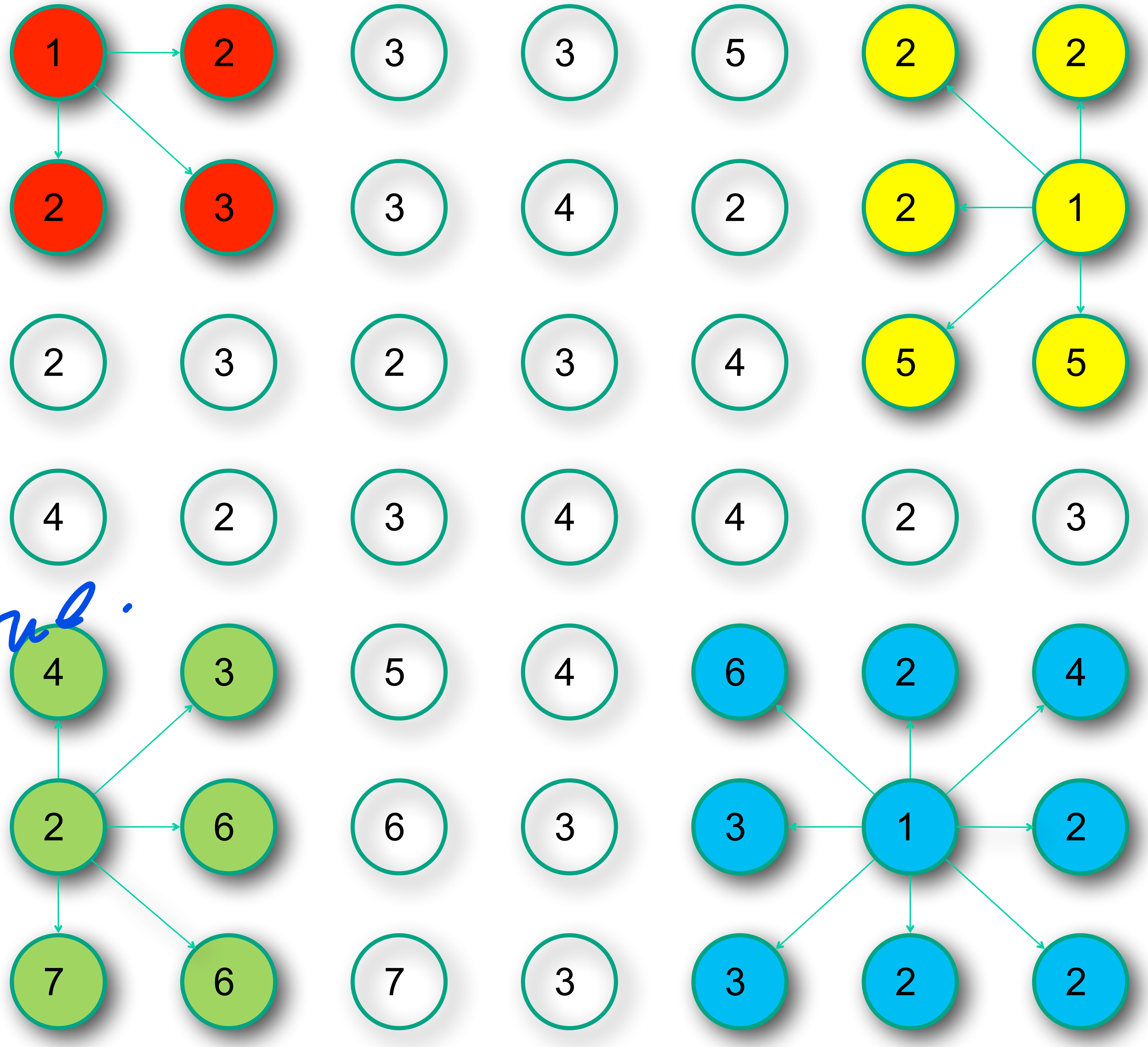


Example

Watershed algorithm. Search simultaneously from seeds in order determined by gray level.

Direction Order:
N,NE,E,SE,
S,SW,W,NW

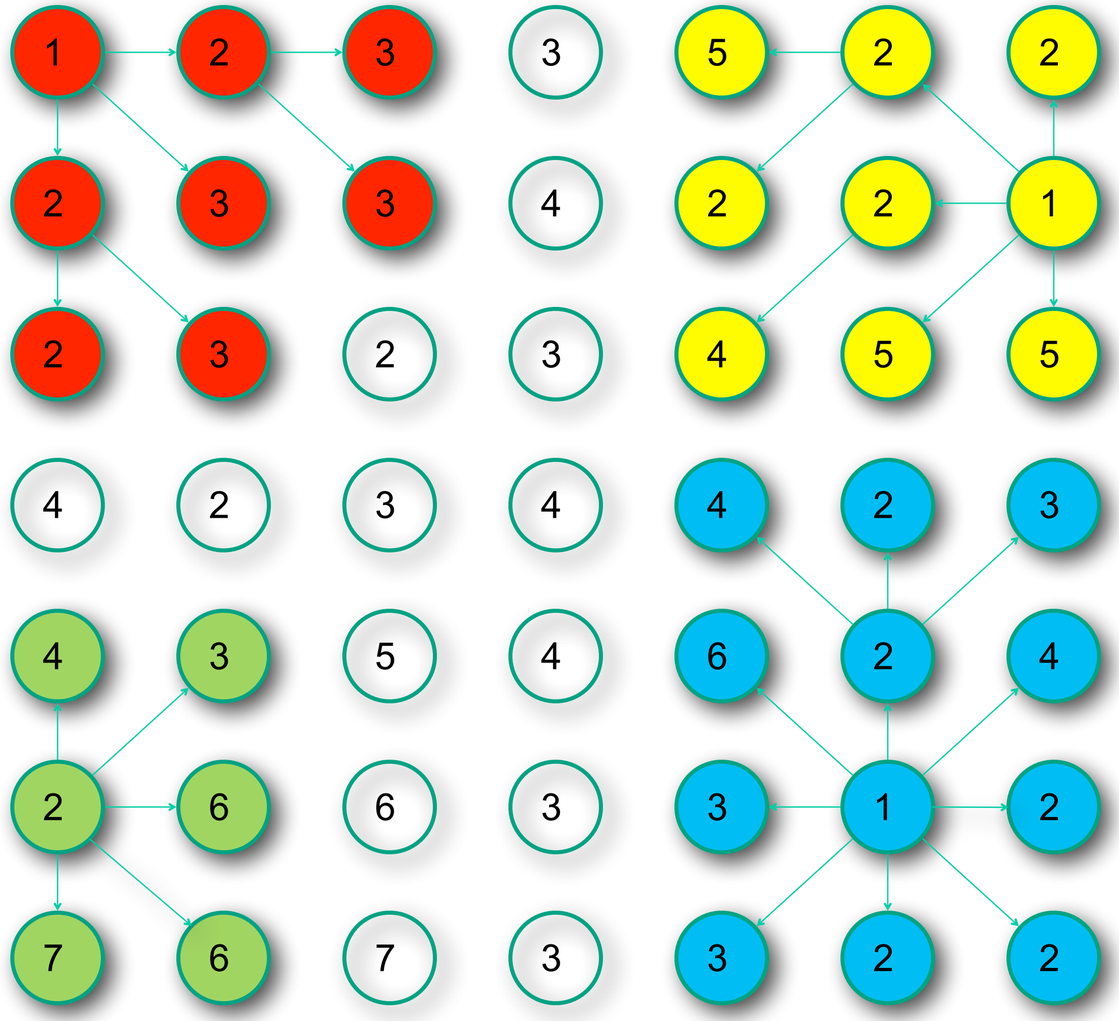
establish priority-queue.
把intensity小的排前面



Example

Watershed algorithm. Search simultaneously from seeds in order determined by gray level.

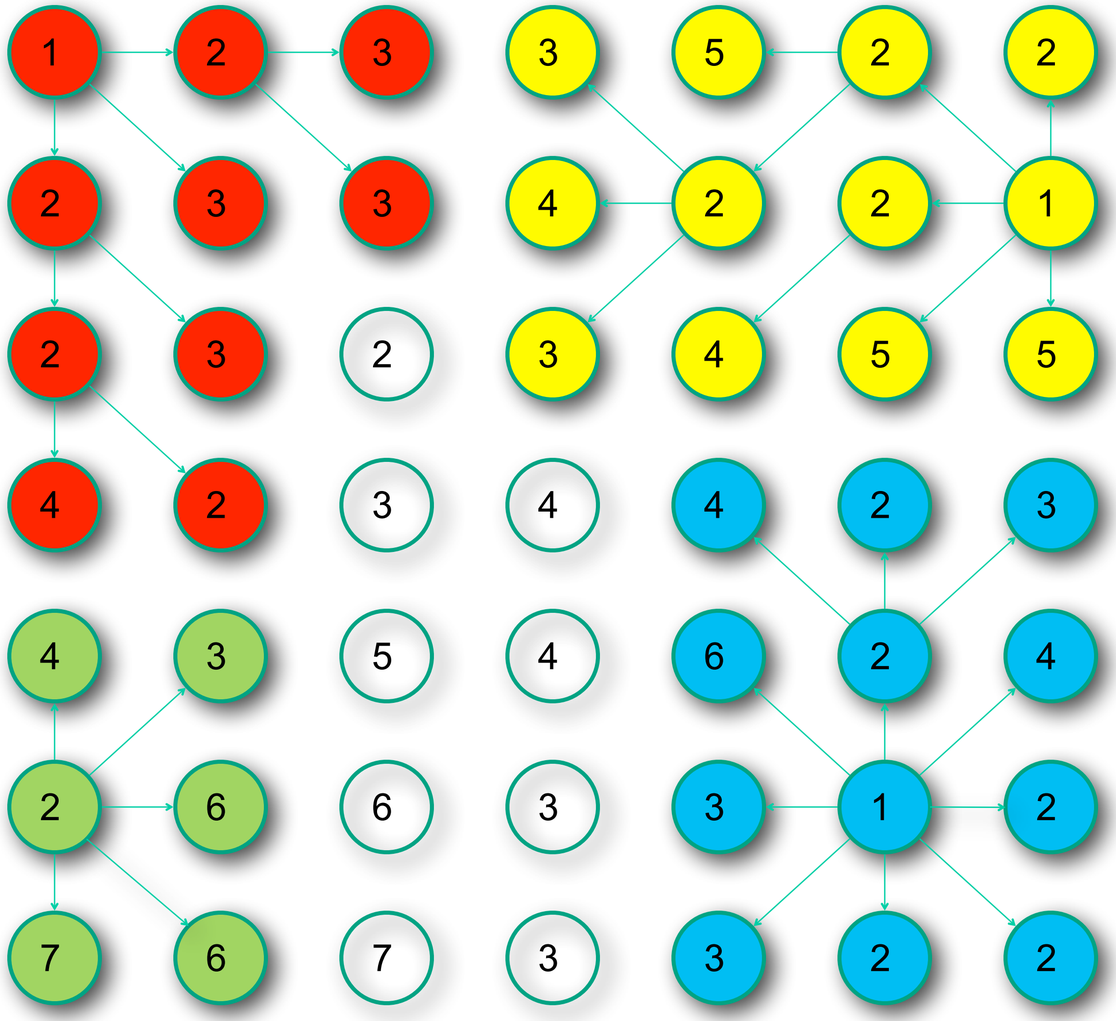
Direction Order:
N,NE,E,SE,
S,SW,W,NW



Example

Watershed algorithm. Search simultaneously from seeds in order determined by gray level.

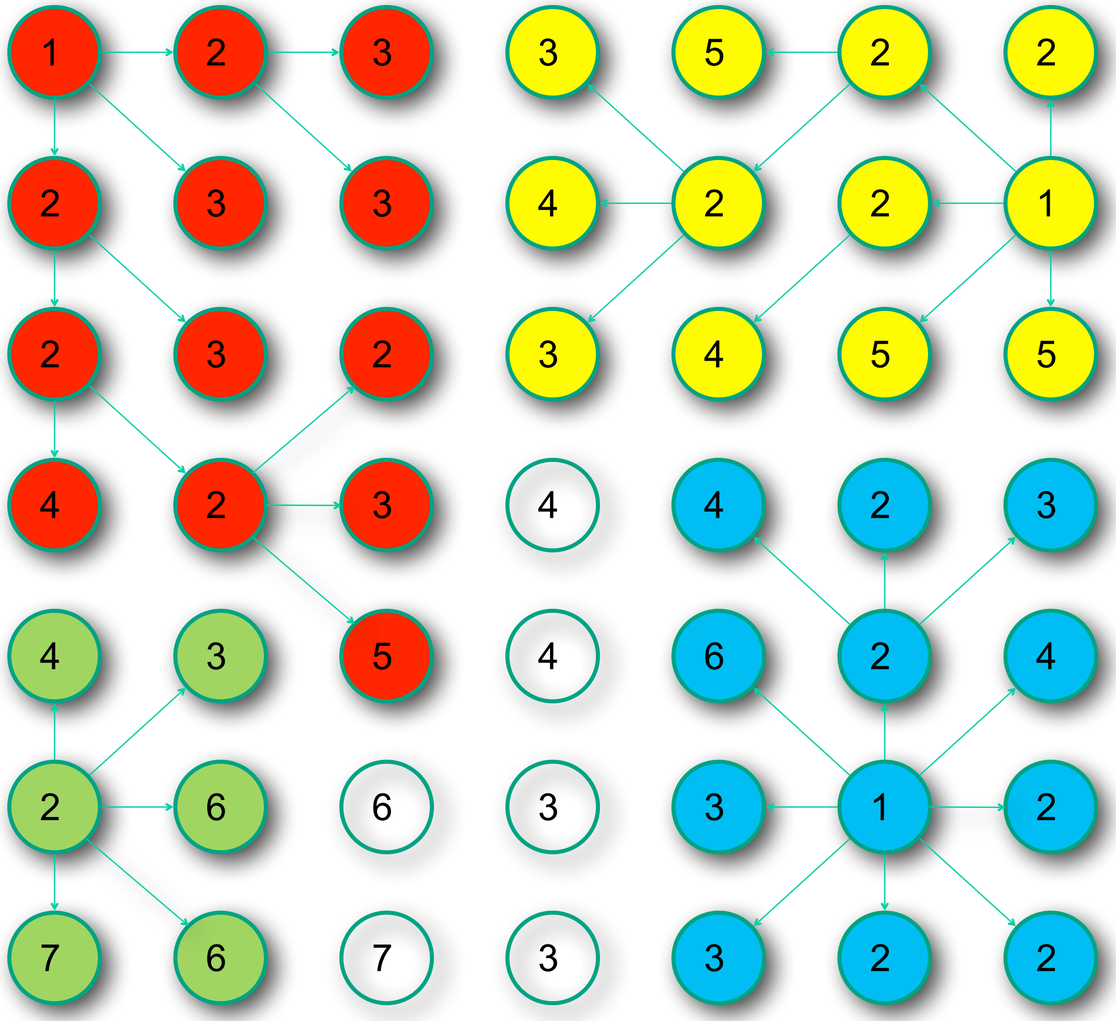
Direction Order:
N,NE,E,SE,
S,SW,W,NW



Example

Watershed algorithm. Search simultaneously from seeds in order determined by gray level.

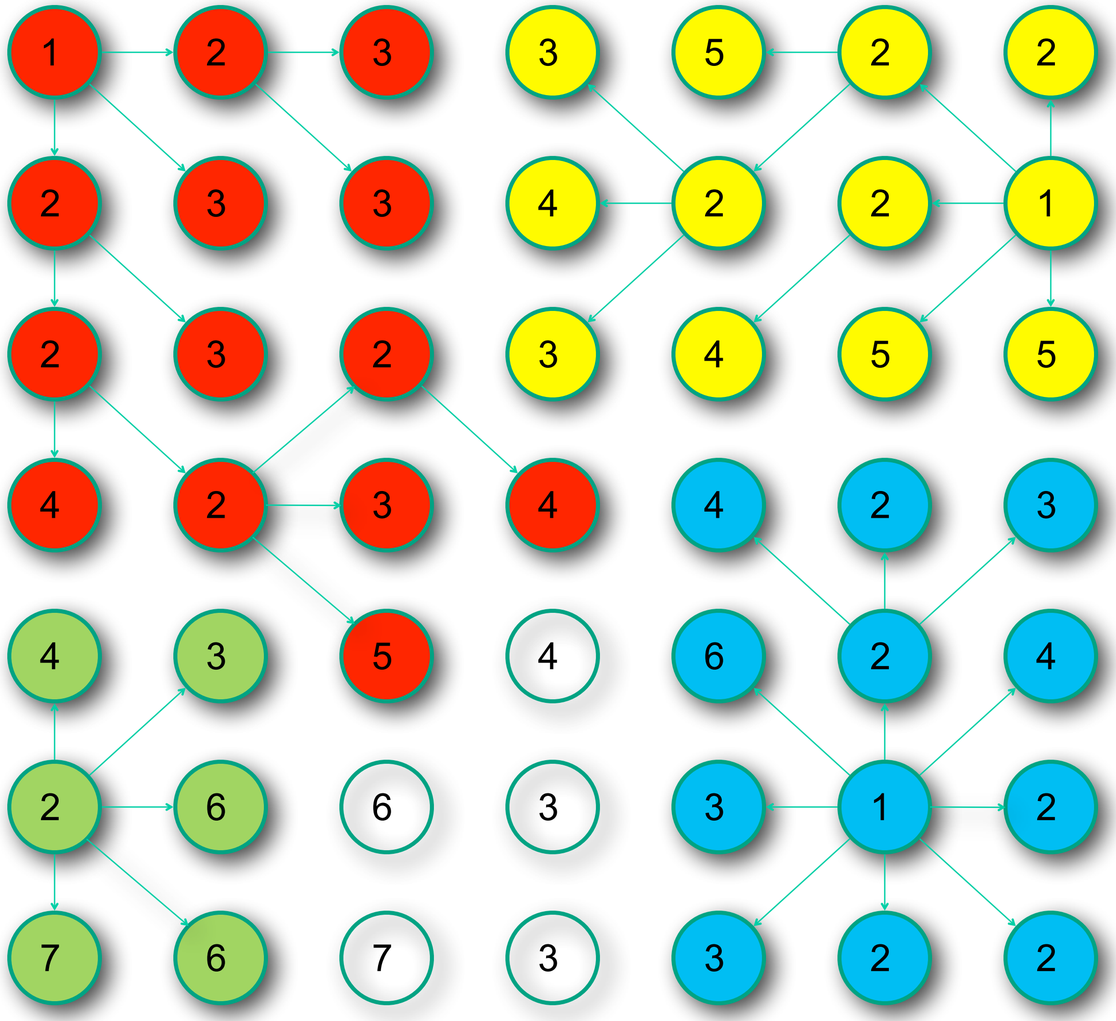
Direction Order:
N,NE,E,SE,
S,SW,W,NW



Example

Watershed algorithm. Search simultaneously from seeds in order determined by gray level.

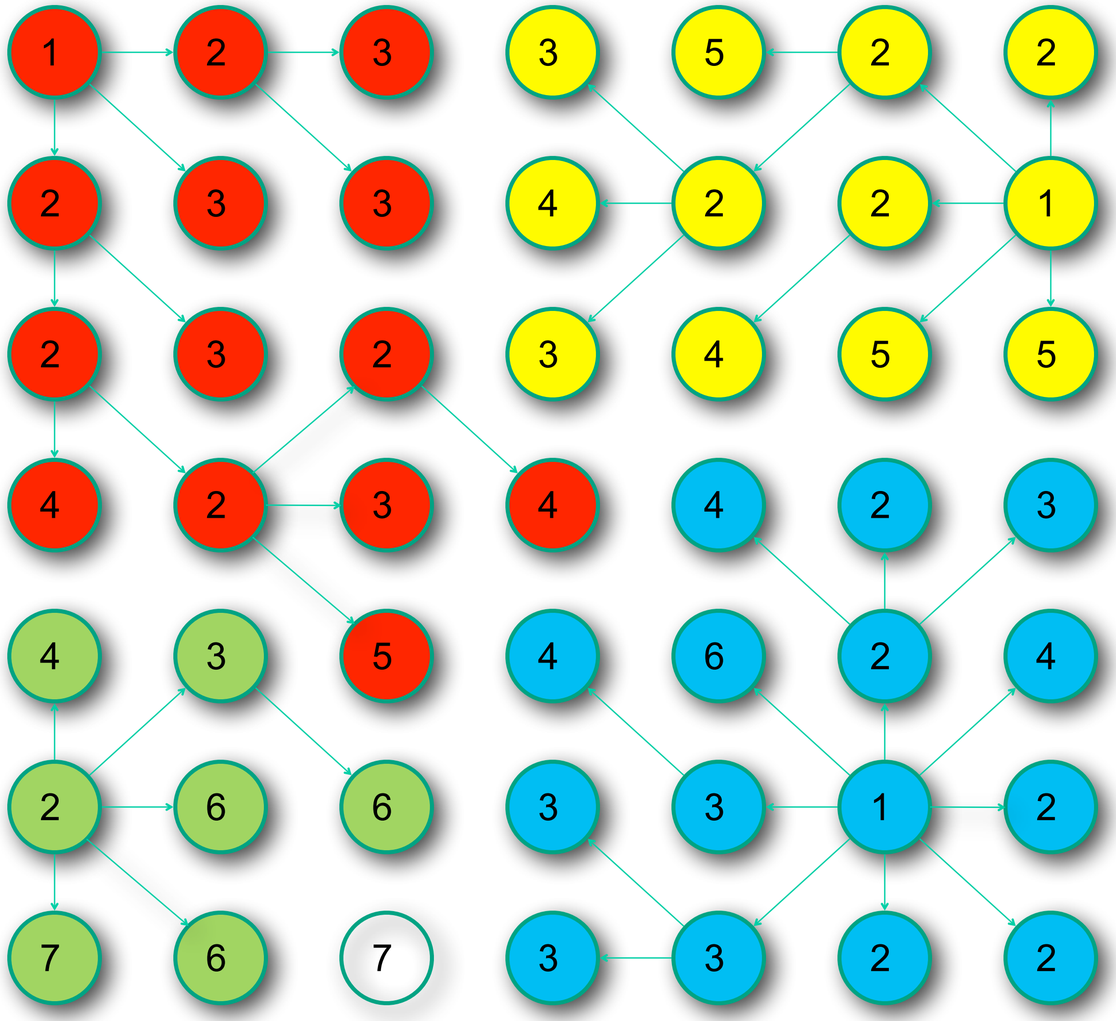
Direction Order:
N,NE,E,SE,
S,SW,W,NW



Example

Watershed algorithm. Search simultaneously from seeds in order determined by gray level.

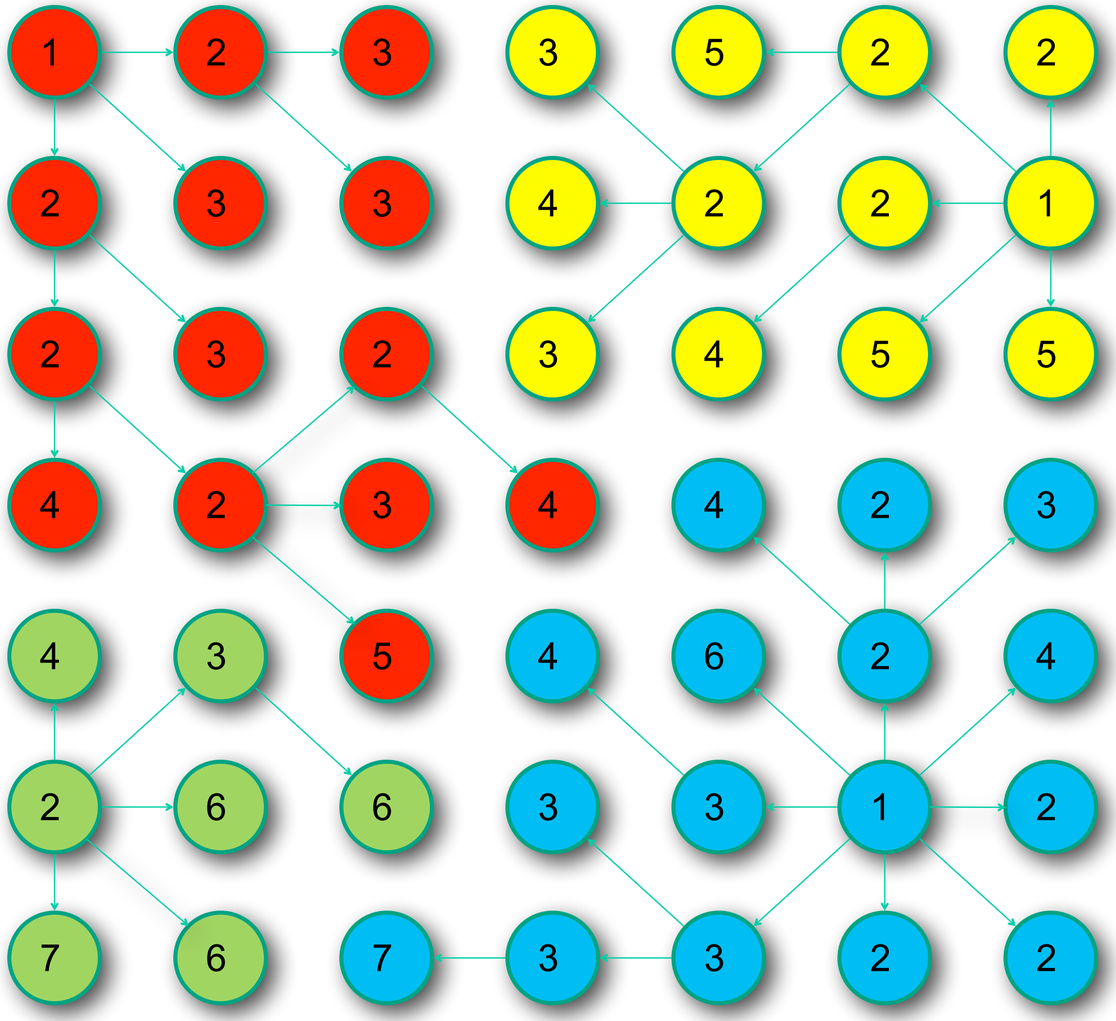
Direction Order:
N,NE,E,SE,
S,SW,W,NW

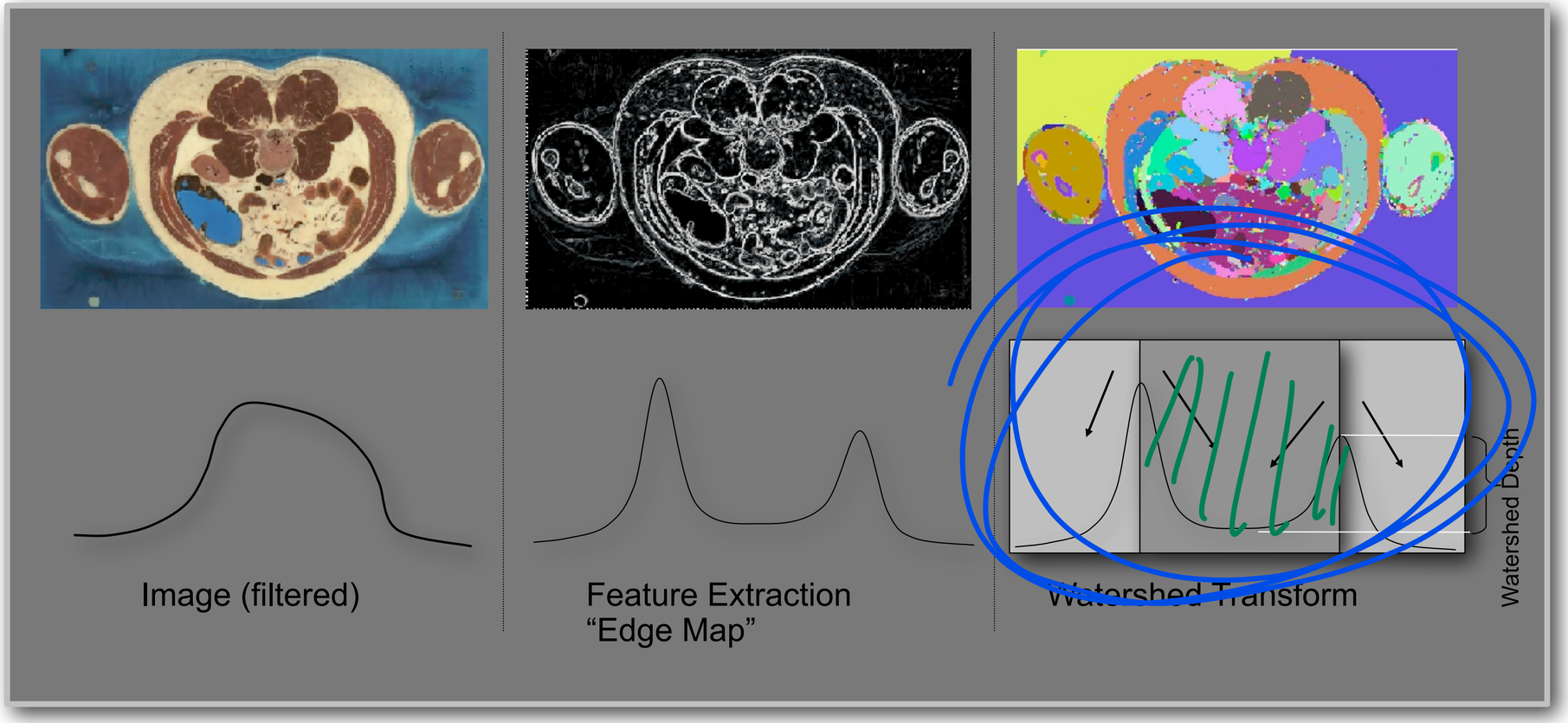


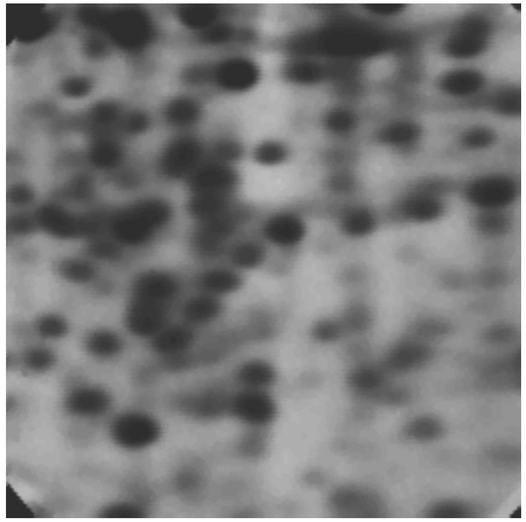
Example

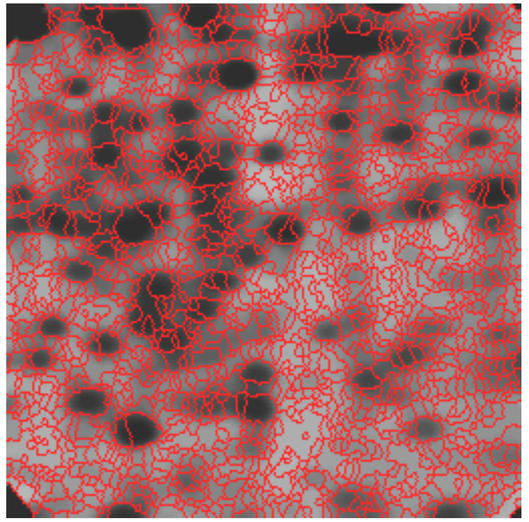
Watershed algorithm. Search simultaneously from seeds in order determined by gray level.

Direction Order:
N,NE,E,SE,
S,SW,W,NW



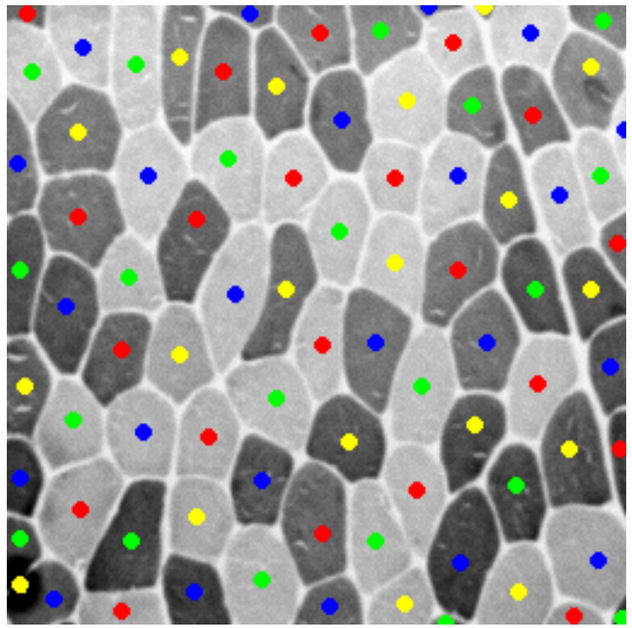




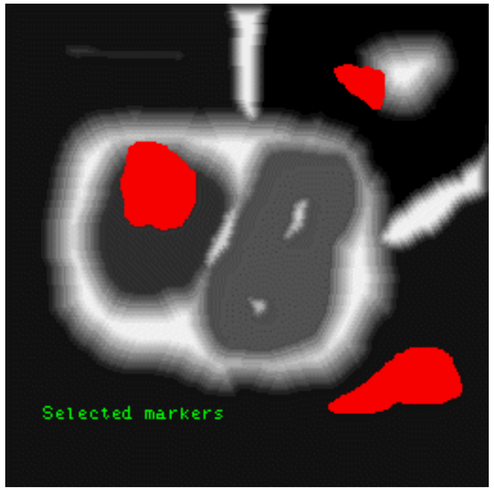


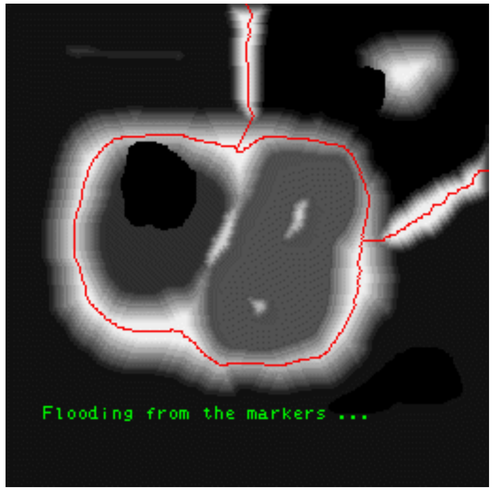
Problem:
Watershed
algorithm typically
over-segments

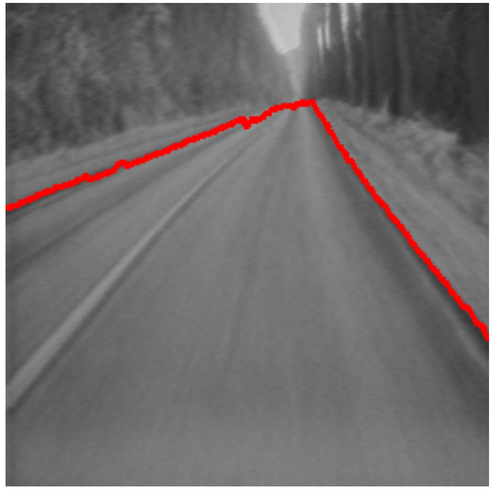
Initialization



IDEA: Initialize regions using other information (cell nuclei) or by hand-marking. Pre-defines number of segments







Analysis

Analysis

Thresholding

- Relies on a single global threshold
- Produces separate regions (may prefer connected components)

Analysis

Thresholding

- Relies on a single global threshold
- Produces separate regions (may prefer connected components)

Region growing

- Supervised, requires 1+ seed point
- Relies on a single threshold
- Produces one region

Analysis

Thresholding

- Relies on a single global threshold
- Produces separate regions (may prefer connected components)

Region growing

- Supervised, requires 1+ seed point
- Relies on a single threshold
- Produces one region

Watershed

- May be supervised or unsupervised
- No need for threshold parameter
- Partitions image into many regions
- To get good results best to specify number and positions of seeds

Overview

What is image segmentation?

Types of segmentation algorithms

1. **Thresholding, region labelling and growing algorithms**
   - **(connected components, region growing, watershed)**
2. **Statistical Segmentation**
   - **(k-means, mean shift)**
3. **Graph based methods**
   - **(Merging algorithms, splitting algorithms, split/merge)**
4. **Edge based methods**
   - **(Intelligent Scissors, Snakes)**

Overview

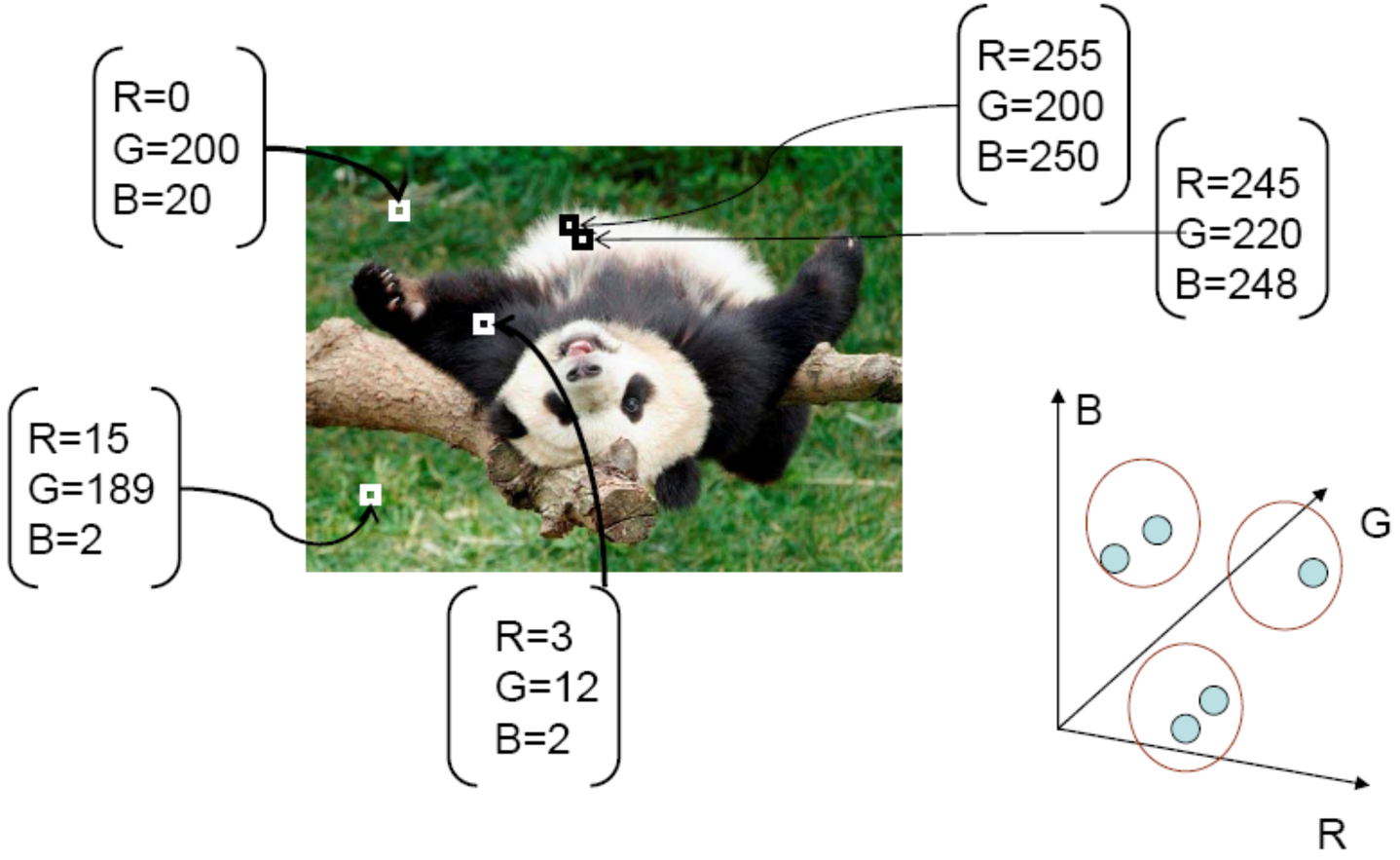
What is image segmentation?

Types of segmentation algorithms

1. **Thresholding, region labelling and growing algorithms**
   - **(connected components, region growing, watershed)**
2. **Statistical Segmentation**
   - **(k-means, mean shift)**
3. **Graph based methods**
   - **(Merging algorithms, splitting algorithms, split/merge)**
4. **Edge based methods**
   - **(Intelligent Scissors, Snakes)**

Segmentation as Clustering

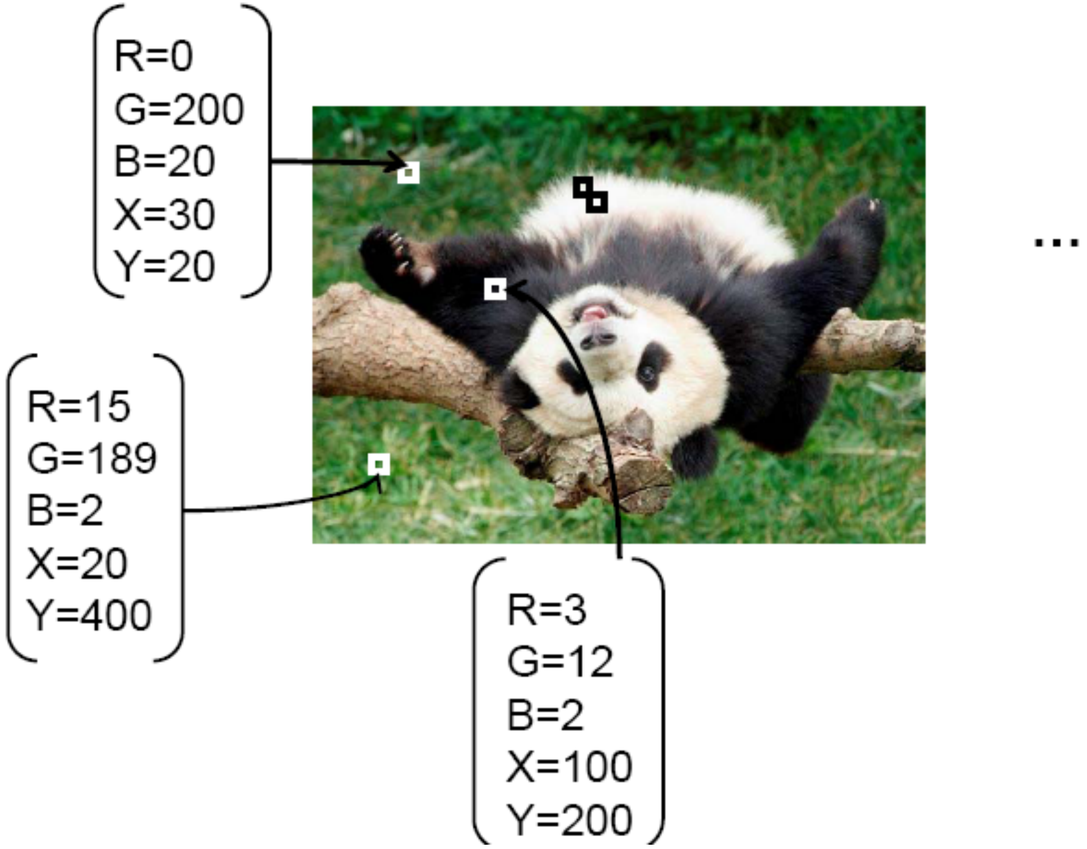
- Cluster similar pixels (features) together



Source: K. Grauman

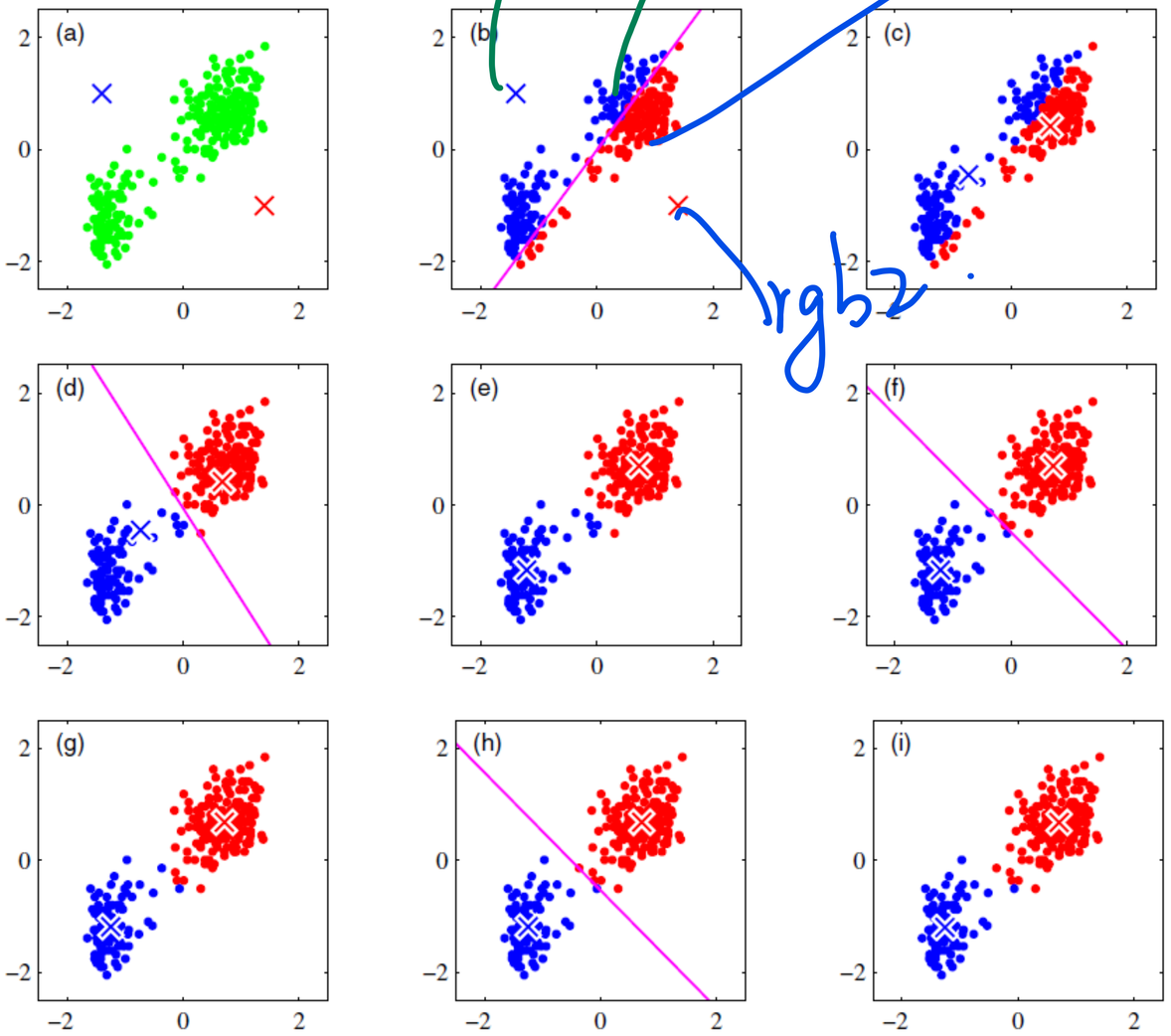
Segmentation as Clustering

- Cluster similar pixels (features) together



Source: K. Grauman

k-means Clustering in 2D



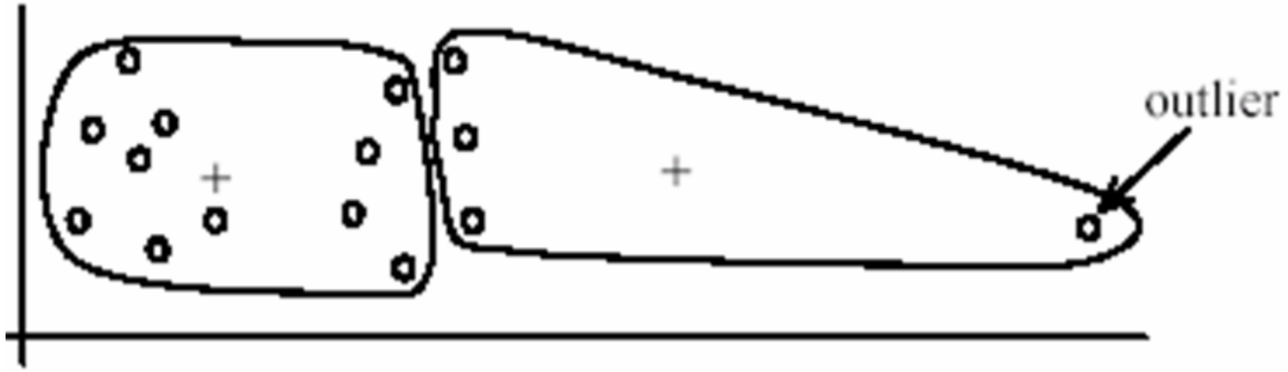
From Bishop (2006)

k-means as Image Compressor

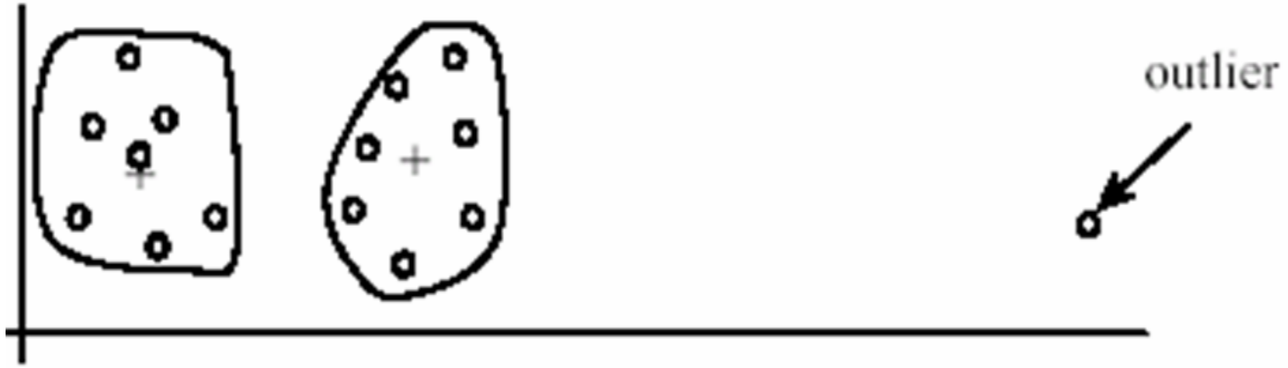
From Bishop (2006)

K-Mean

- **Pros**
  - Simple and fast
  - Converges to a local minimum of the distance function
- **Cons**
  - Need to pick K
  - Sensitive to initialization
  - Sensitive to outliers
  - Only finds “spherical” clusters



(A): Undesirable clusters



(B): Ideal clusters

Comparison

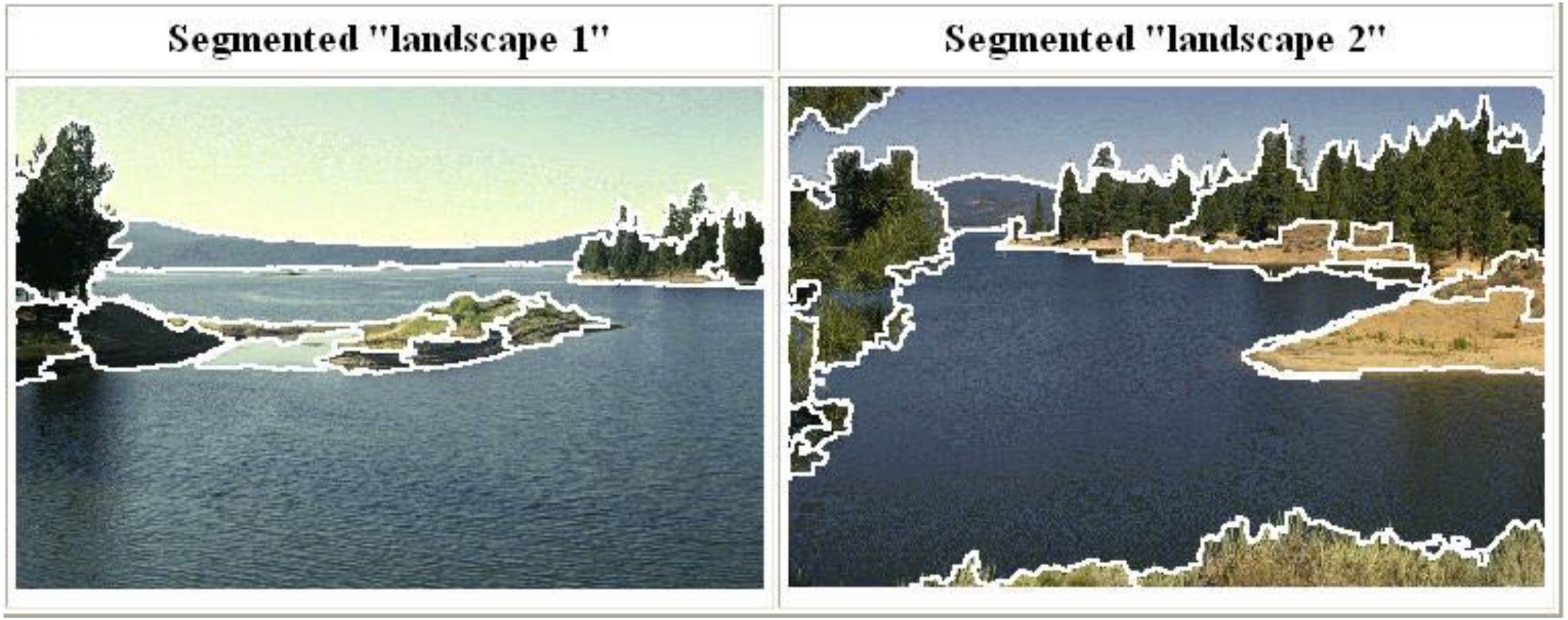
Watershed

- May be supervised or unsupervised
- No need for threshold parameter
- Partitions image into many regions
- To get good results best to specify number and positions of seeds

K-Means

- Unsupervised
- Works in high-dimensions
- May produce disjoint segments
- Must specify number of clusters but not position

Meanshift Segmentation

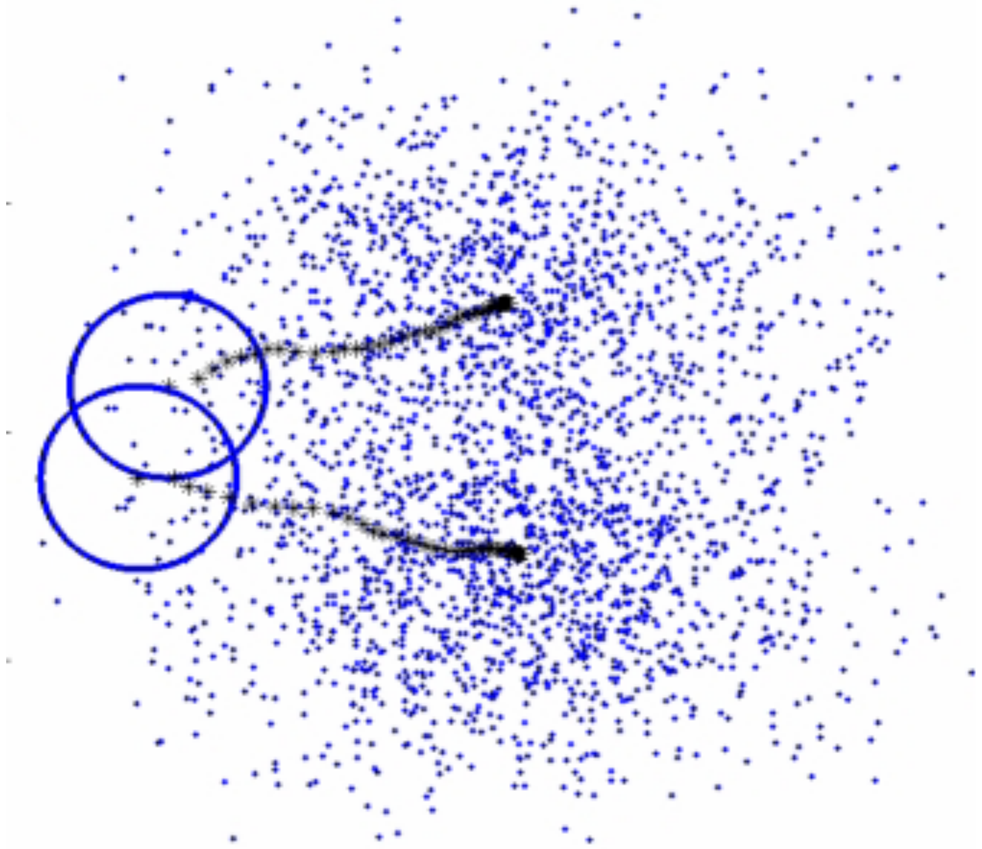


Mean Shift: A Robust Approach toward Feature Space Analysis, PAMI 2002.
D. Comaniciu and P. Meer

Mean Shift Algorithm

The mean shift algorithm seeks a *mode* or local maximum of density of a given distribution

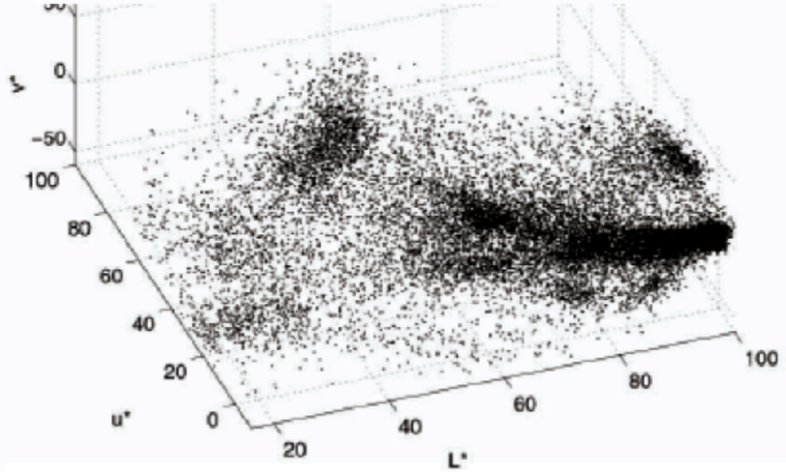
1. **Choose a search window (width and location)**
2. **Compute the mean of the data in the search window**
3. **Center the search window at the new mean location**
4. **Repeat until convergence**

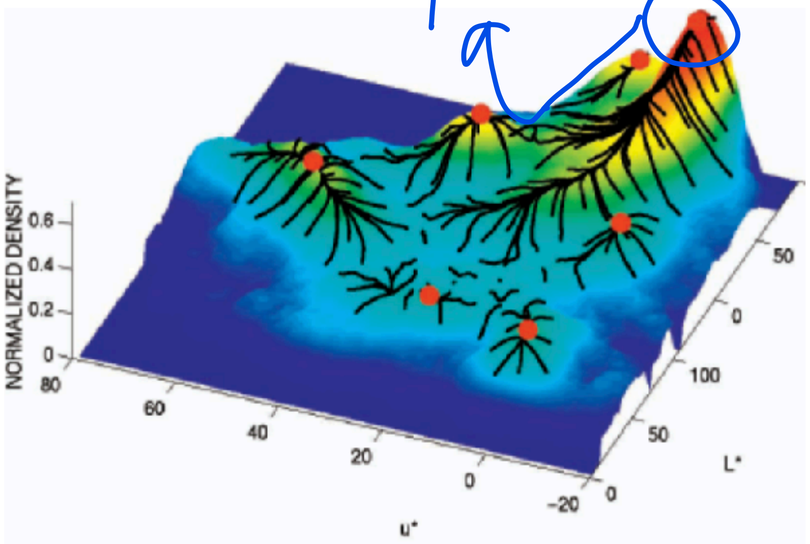


From Rob Fergus/ CV Online

Mean shift Segmentation

density of pixels with that specific colour





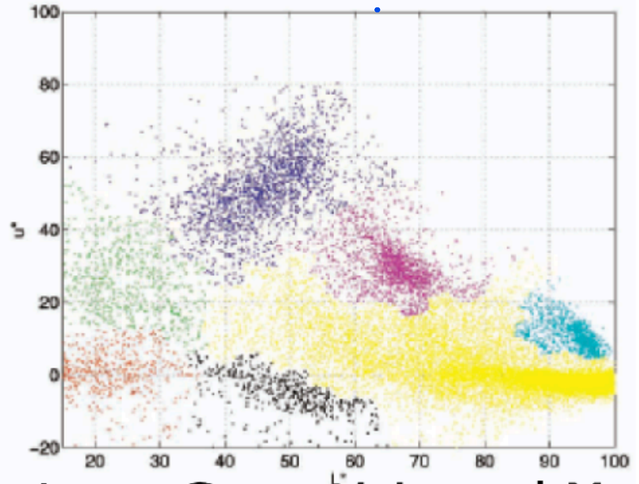


Image courtesy: Comaniciu and Meer [2.]

Mean shift Segmentation

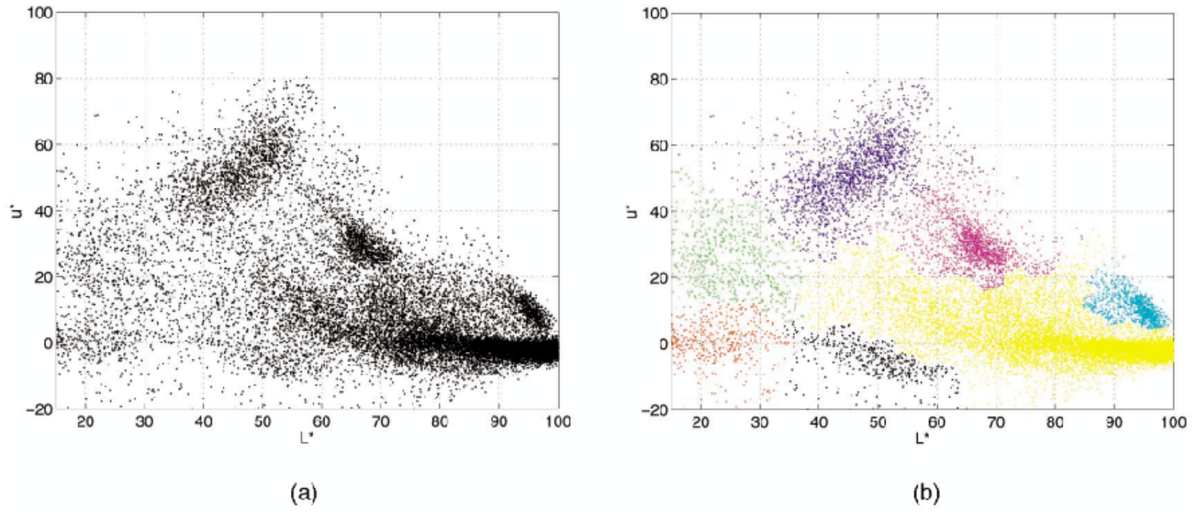
Idea: Estimating the PDF and Finding the Maxima

- Non-parametric approach to density estimation ("how many data points are in a certain region?")
- Find the local modes of this density
- All points that "belong" or "lead" to the same mode form a cluster

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

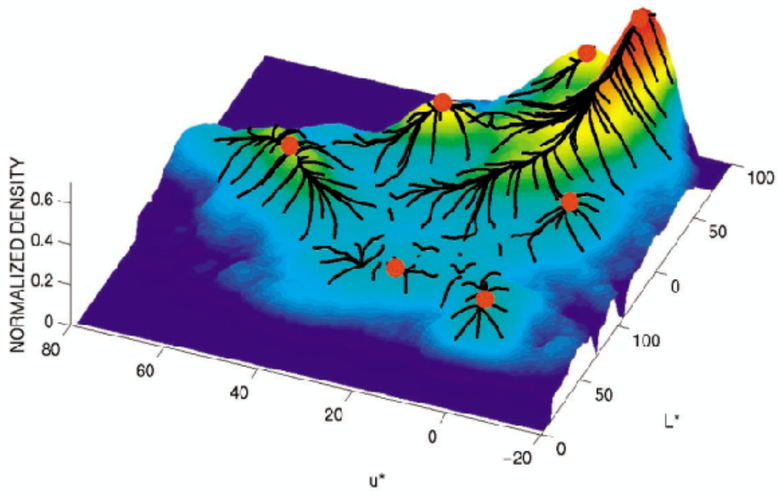
Mean shift Clustering/Segmentation

1. Find features (color, gradients, texture, etc)
2. Initialise windows at different locations
3. Perform mean shift for each window until convergence
4. Merge windows that end up near the same “peak” or mode



(a)

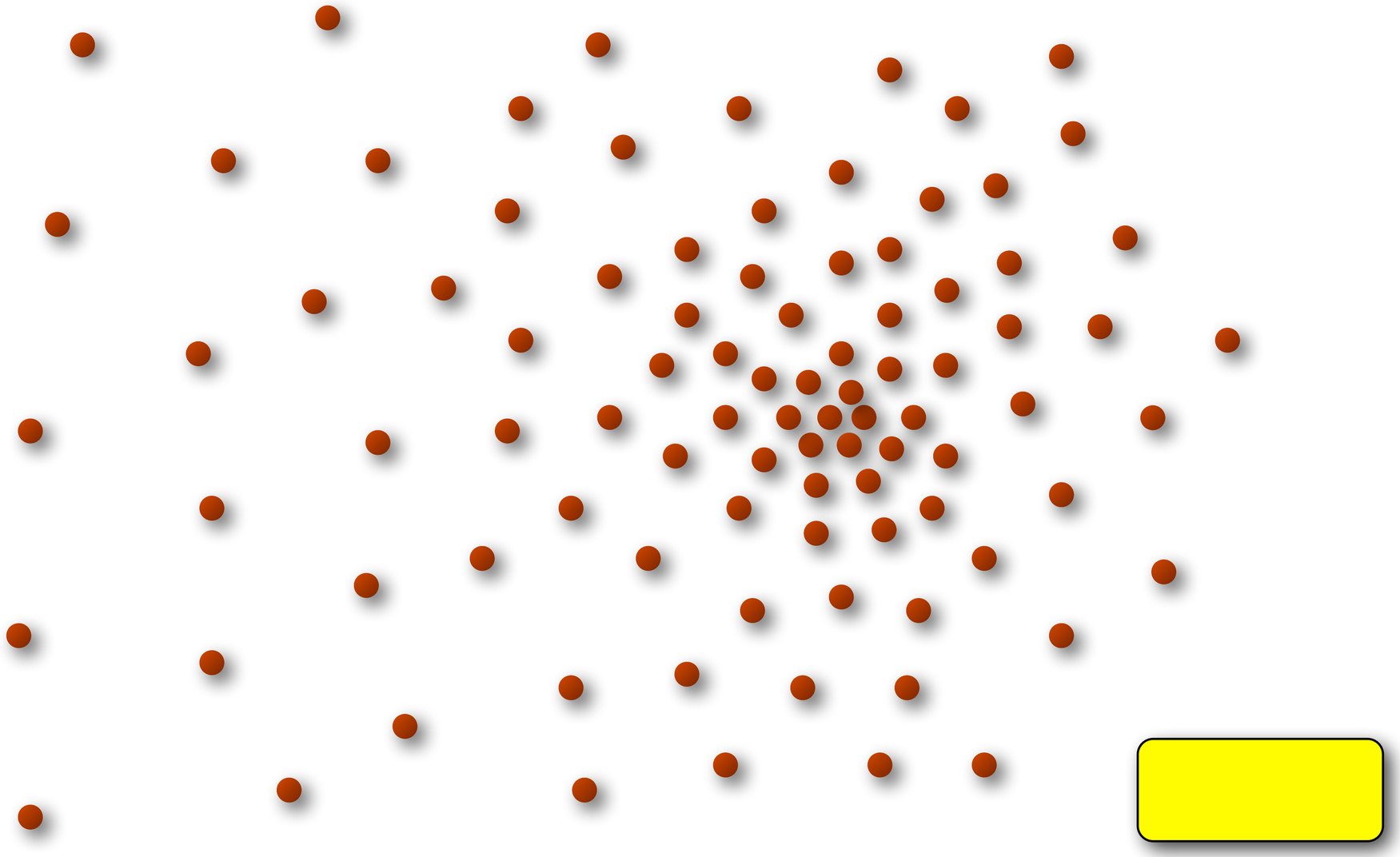
(b)



(c)

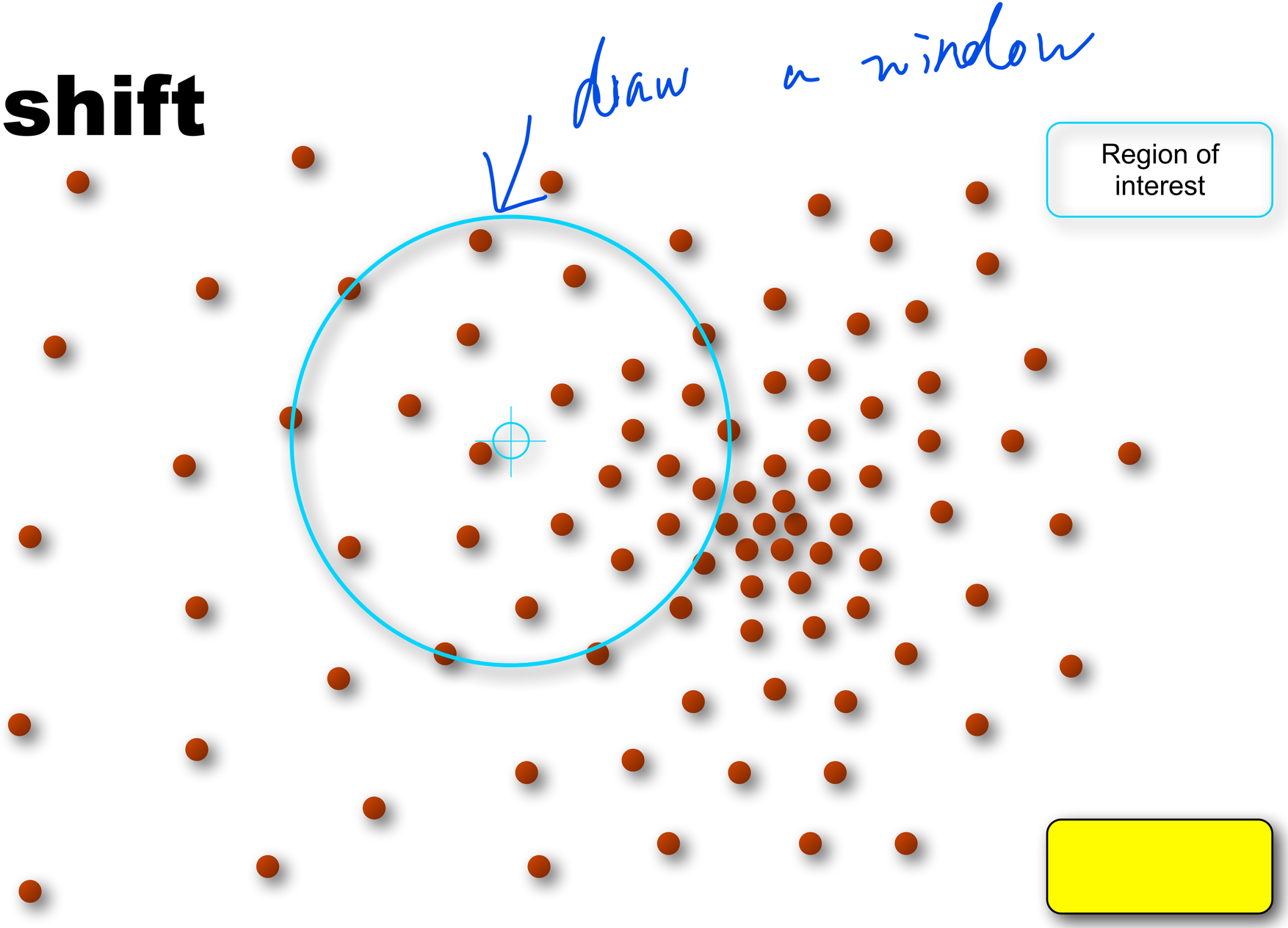
From Rob Fergus

Mean shift



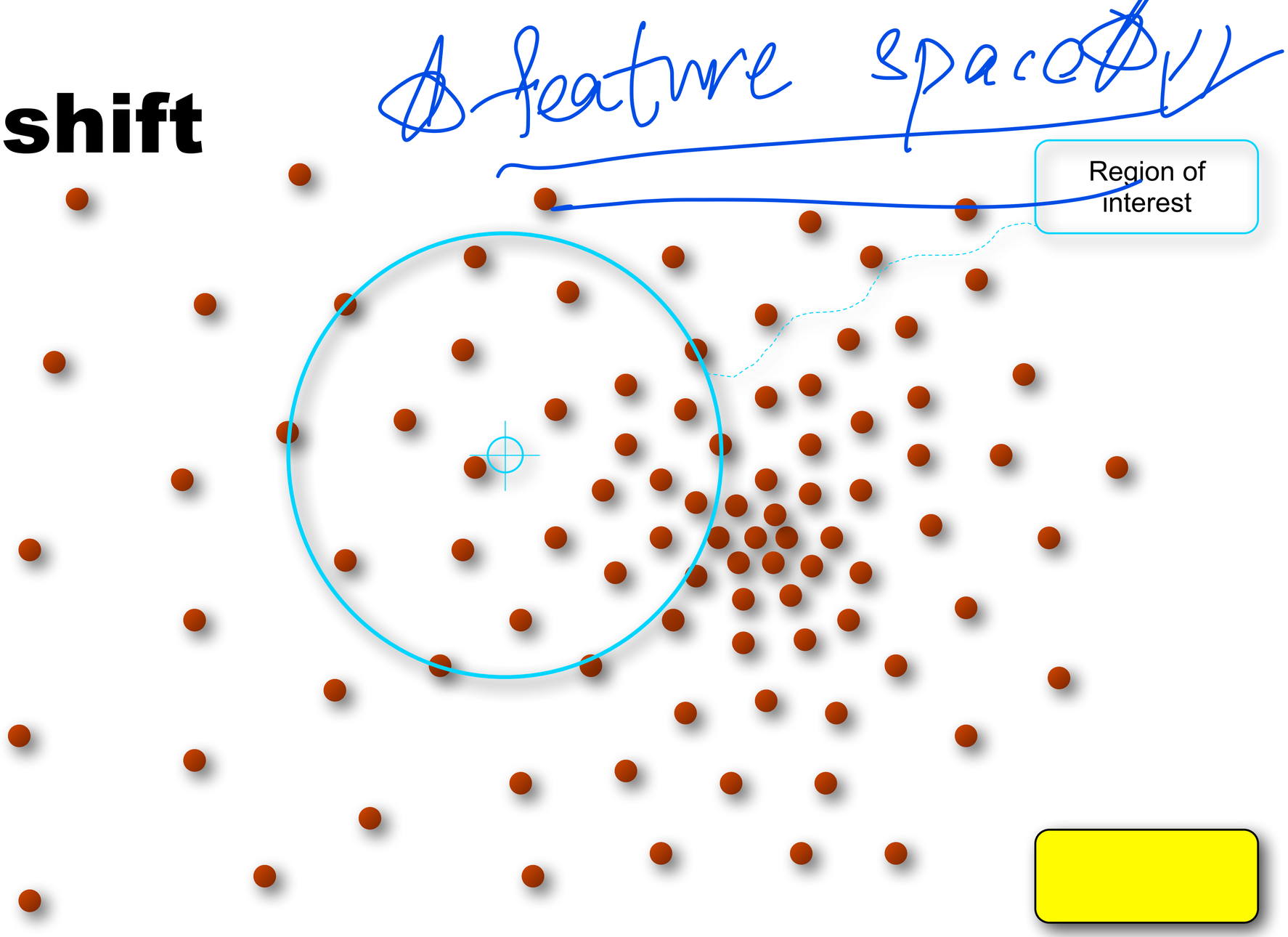
Slide by Y. Ukrainitz & B. Sarel

Mean shift



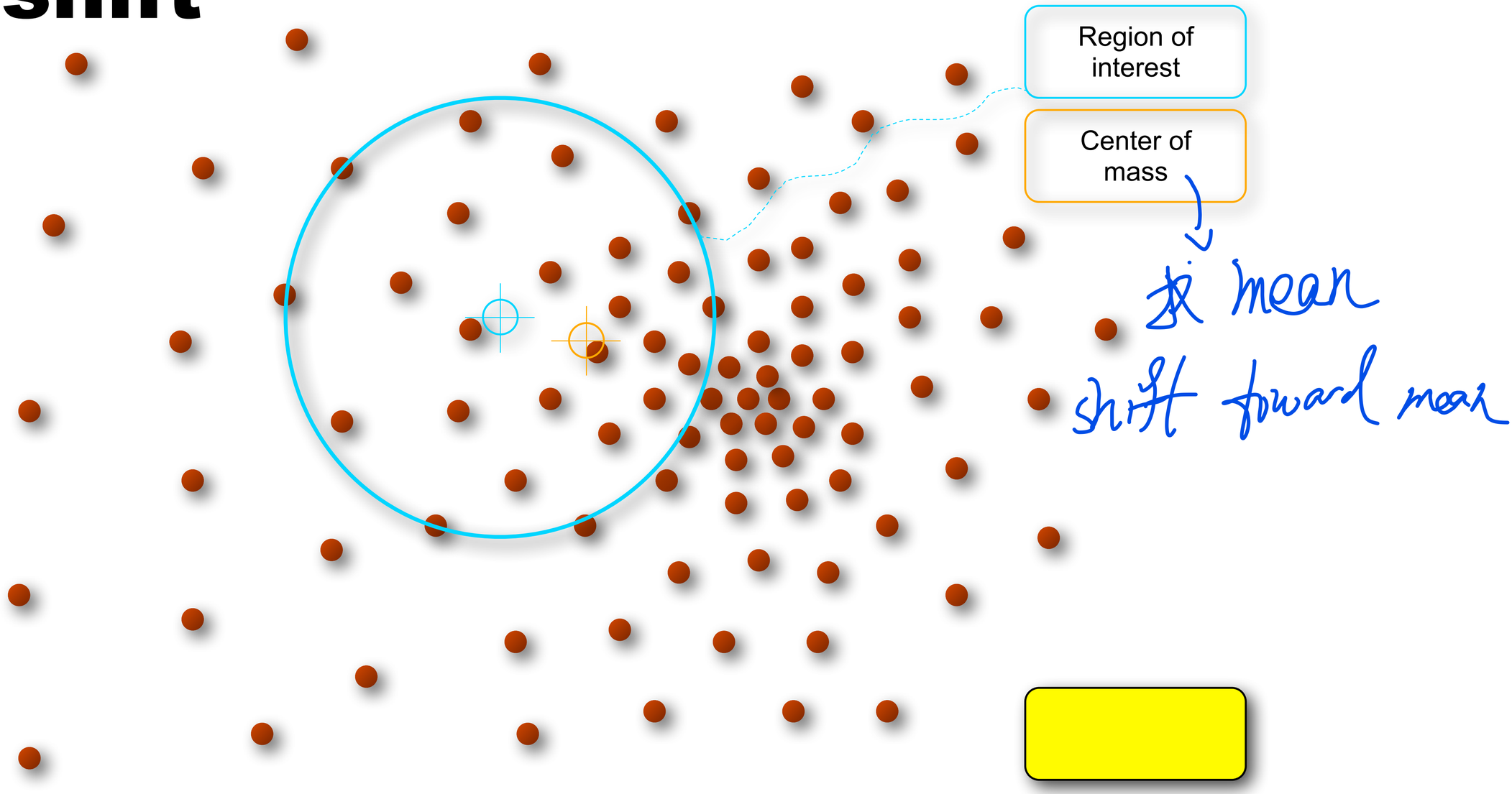
Slide by Y. Ukrainitz & B. Sarel

Mean shift

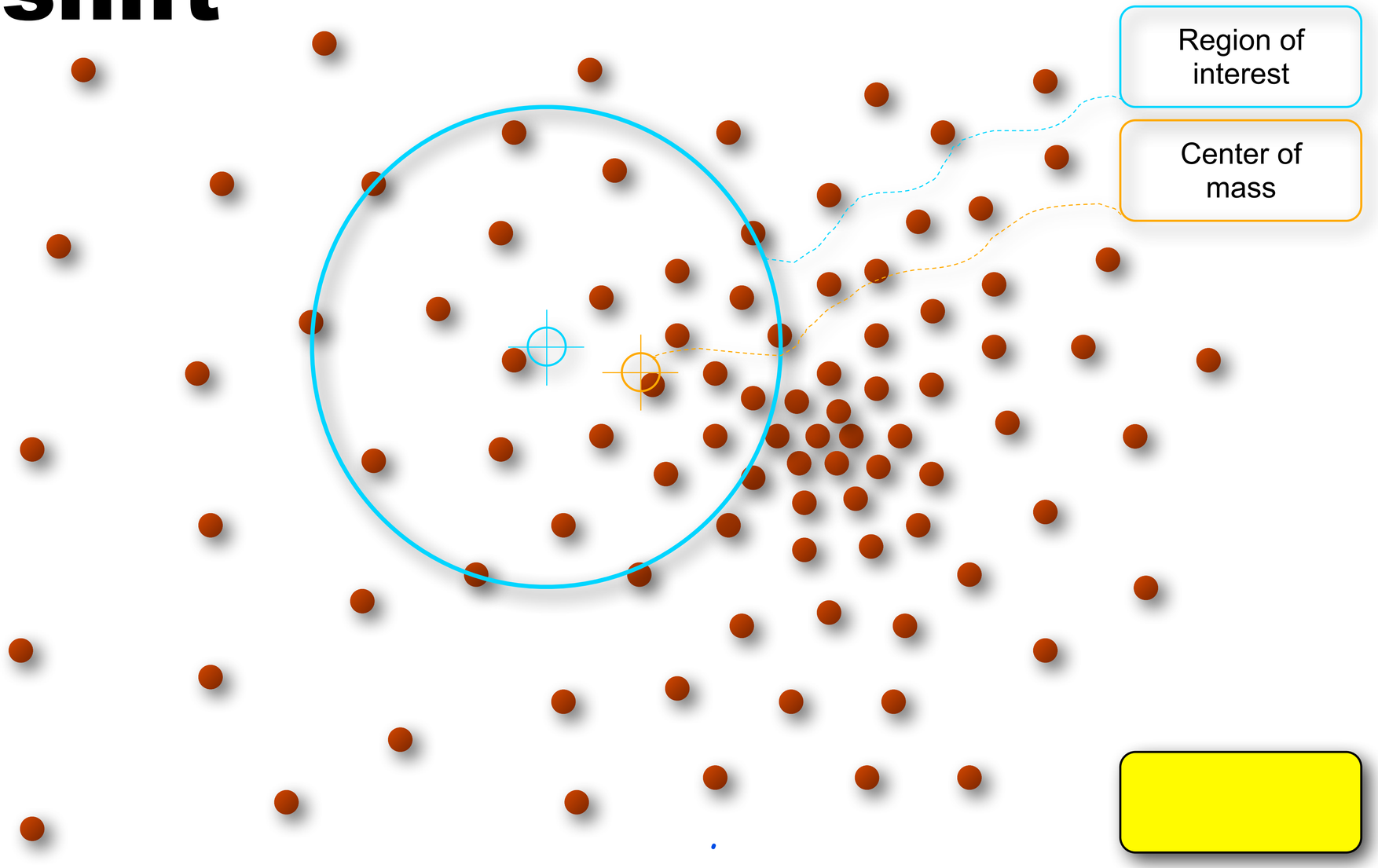


Slide by Y. Ukrainitz & B. Sarel

Mean shift

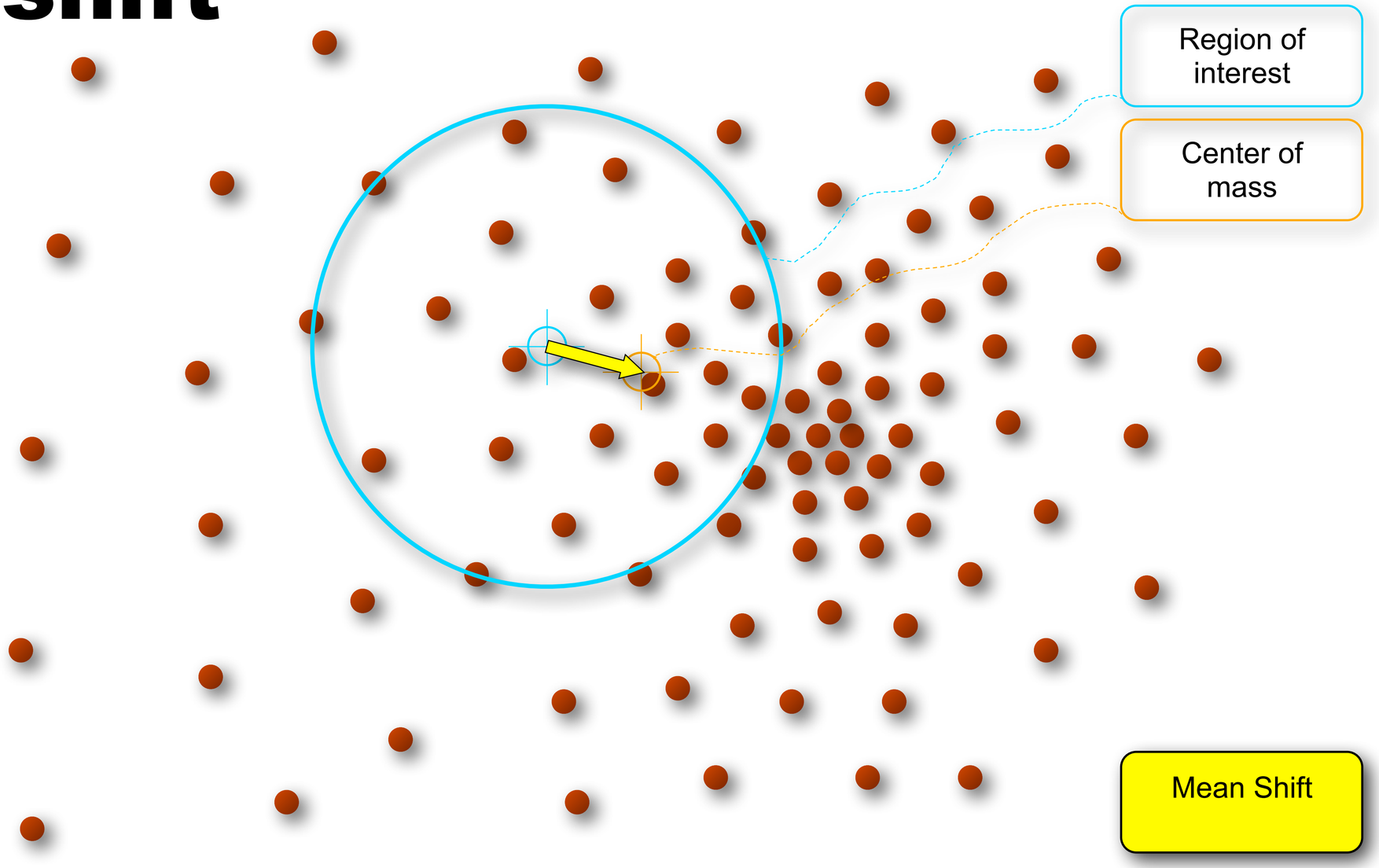


Mean shift

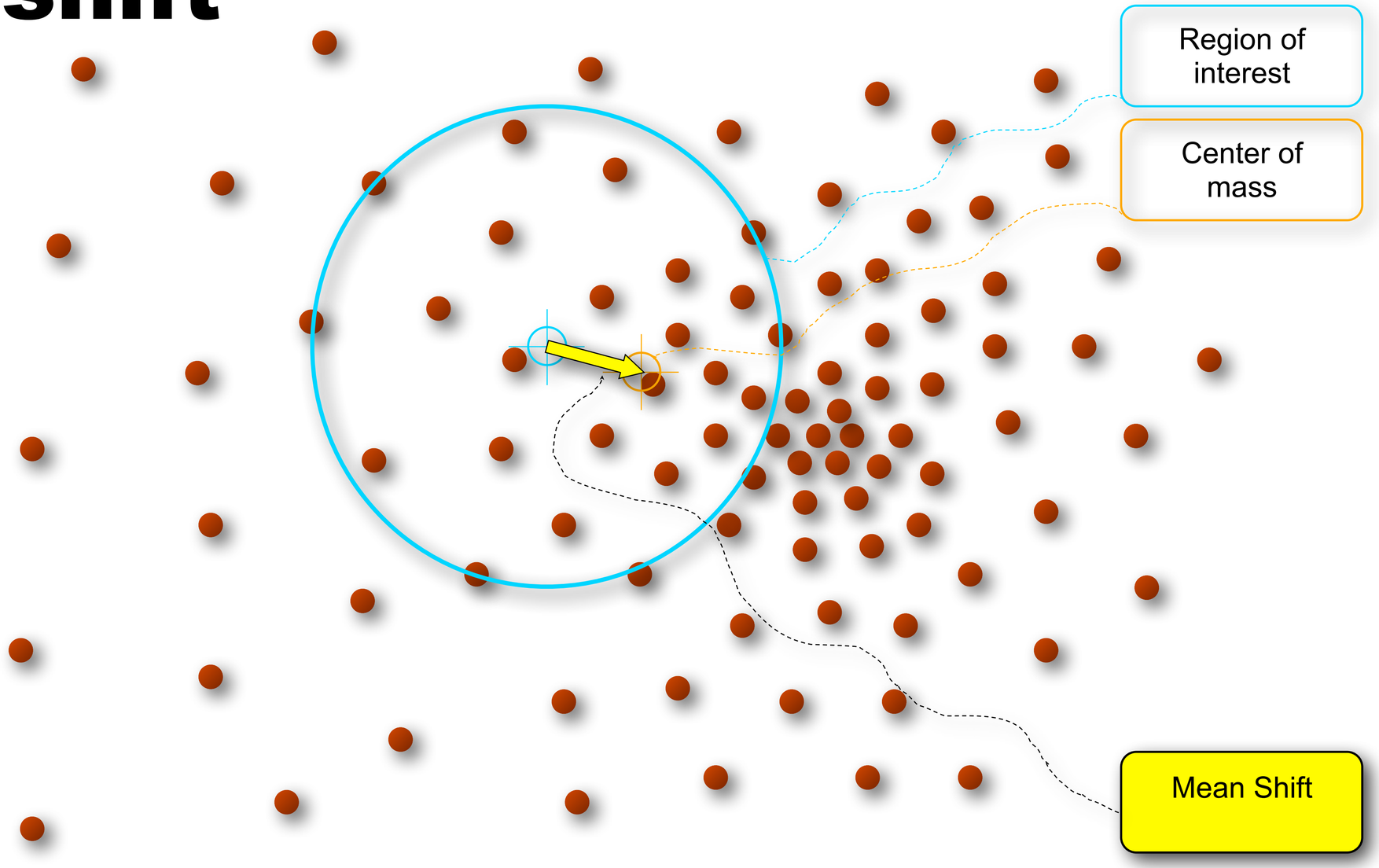


Slide by Y. Ukrainitz & B. Sarel

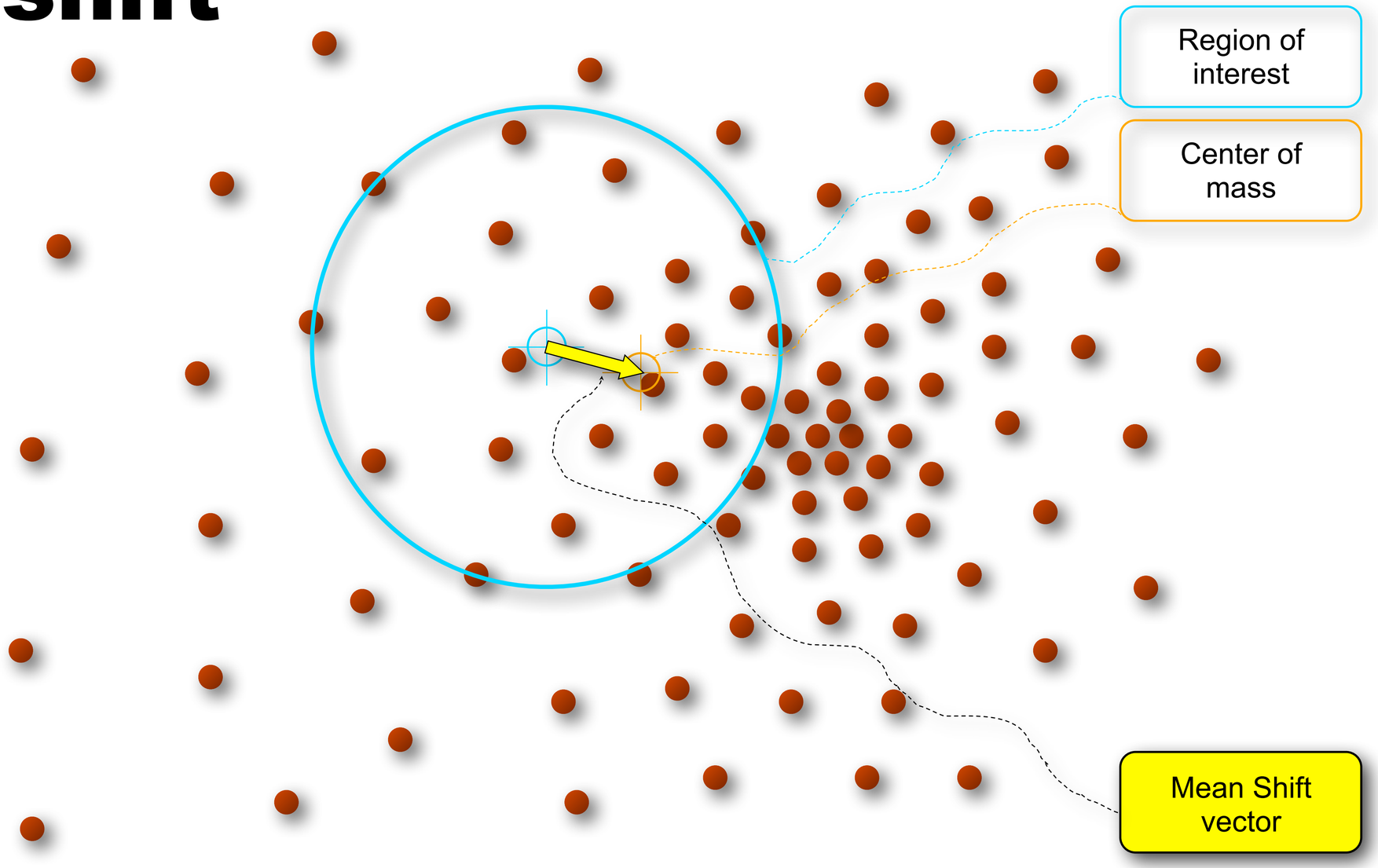
Mean shift



Mean shift

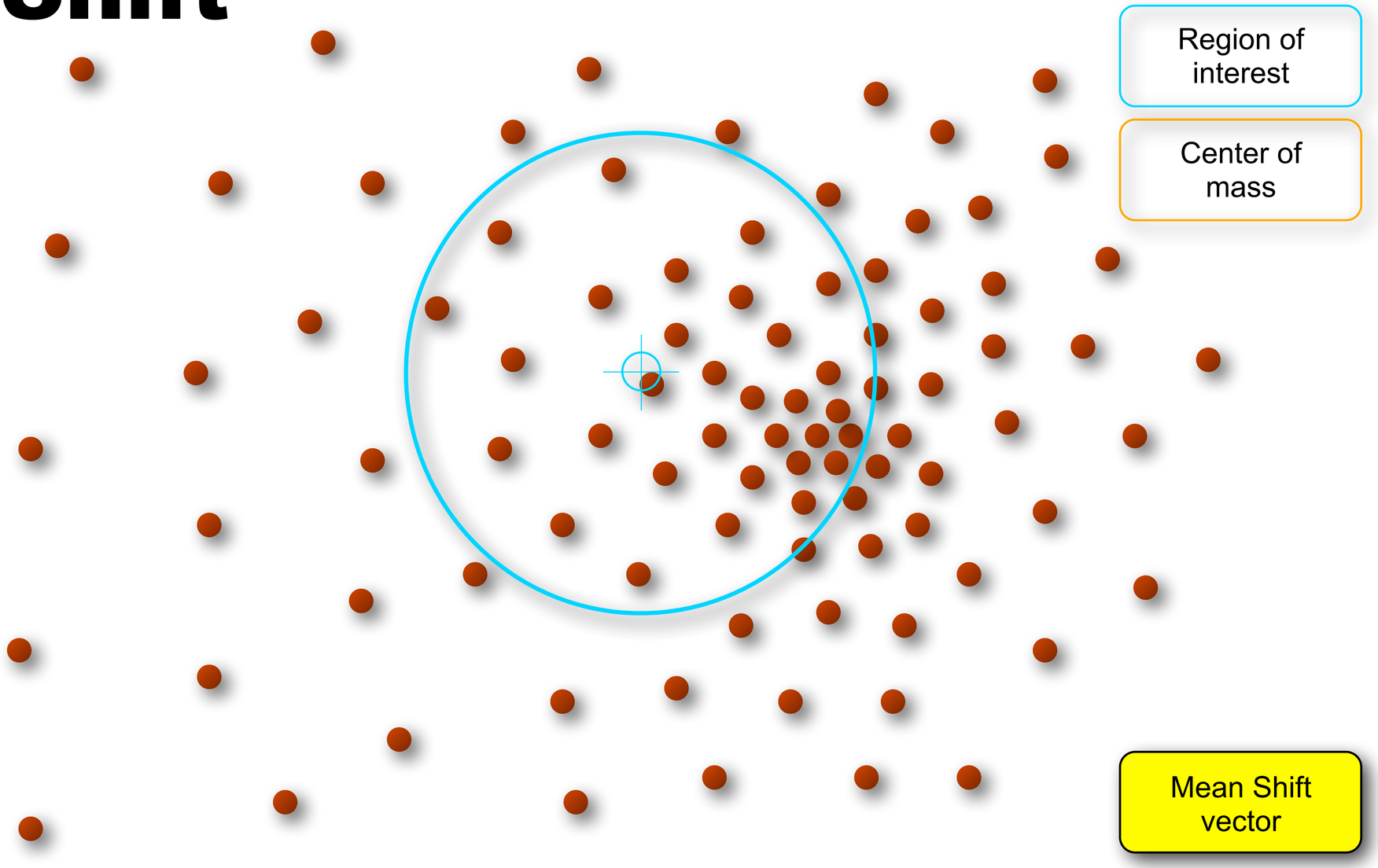


Mean shift

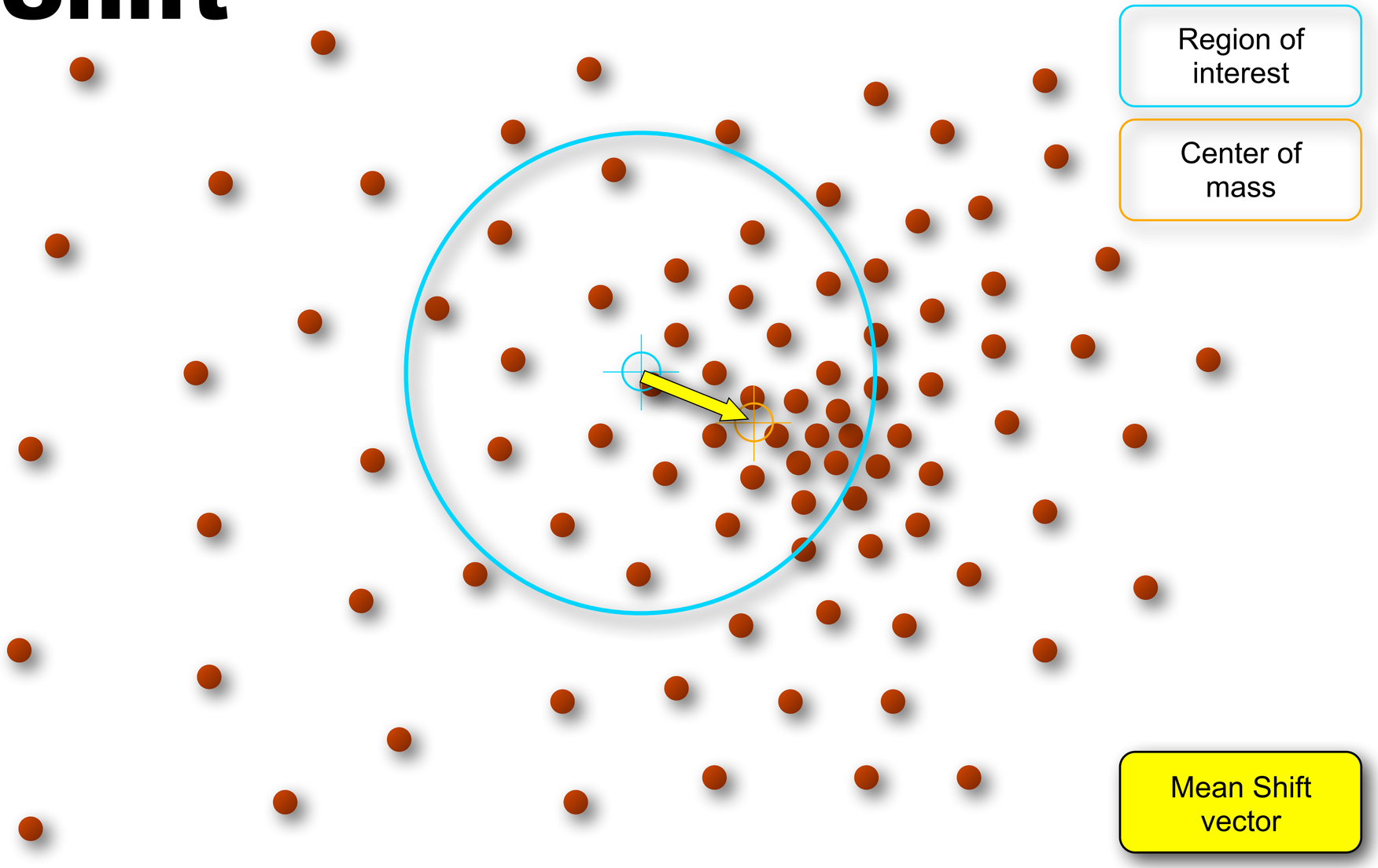


Slide by Y. Ukrainitz & B. Sarel

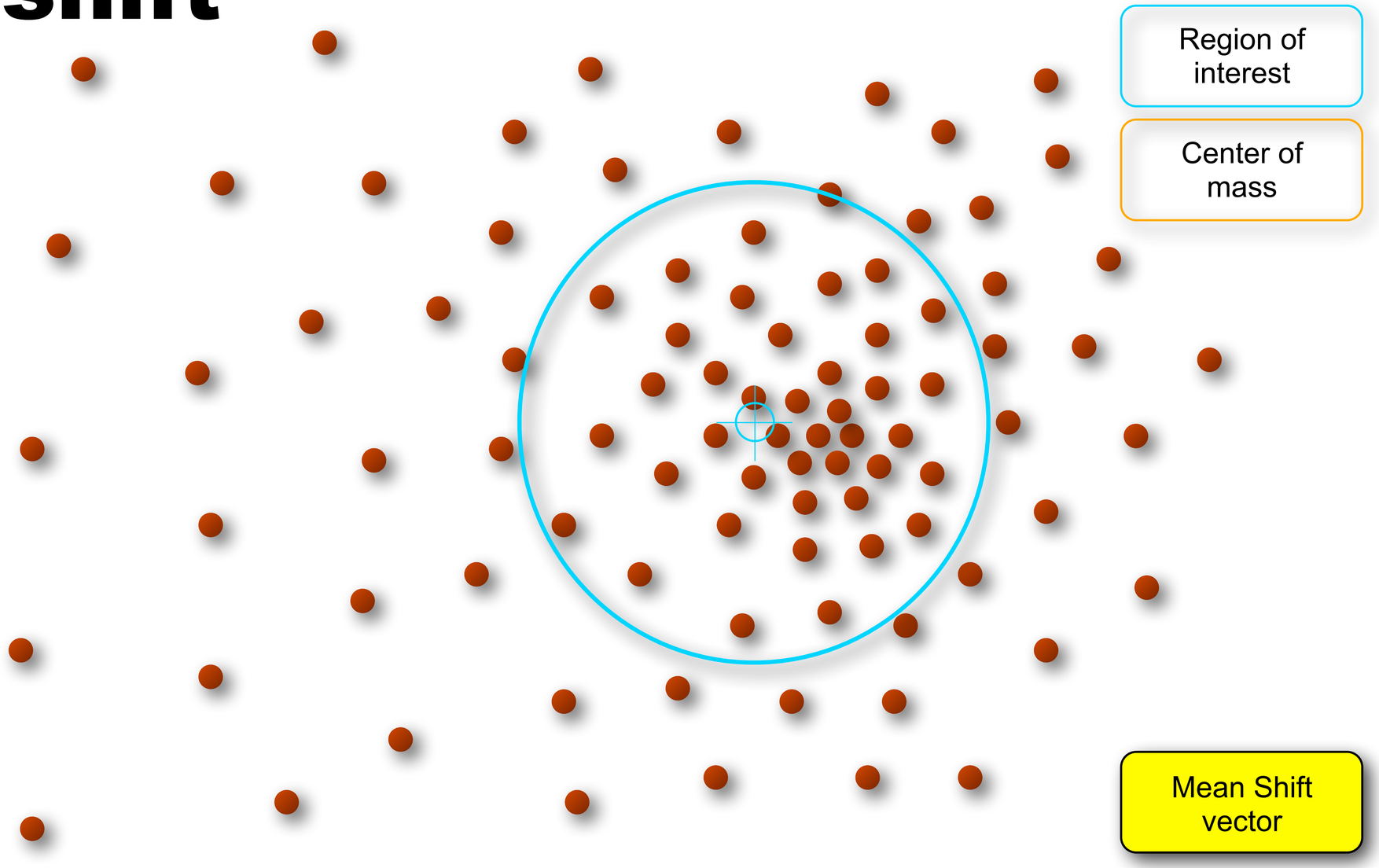
Mean Shift



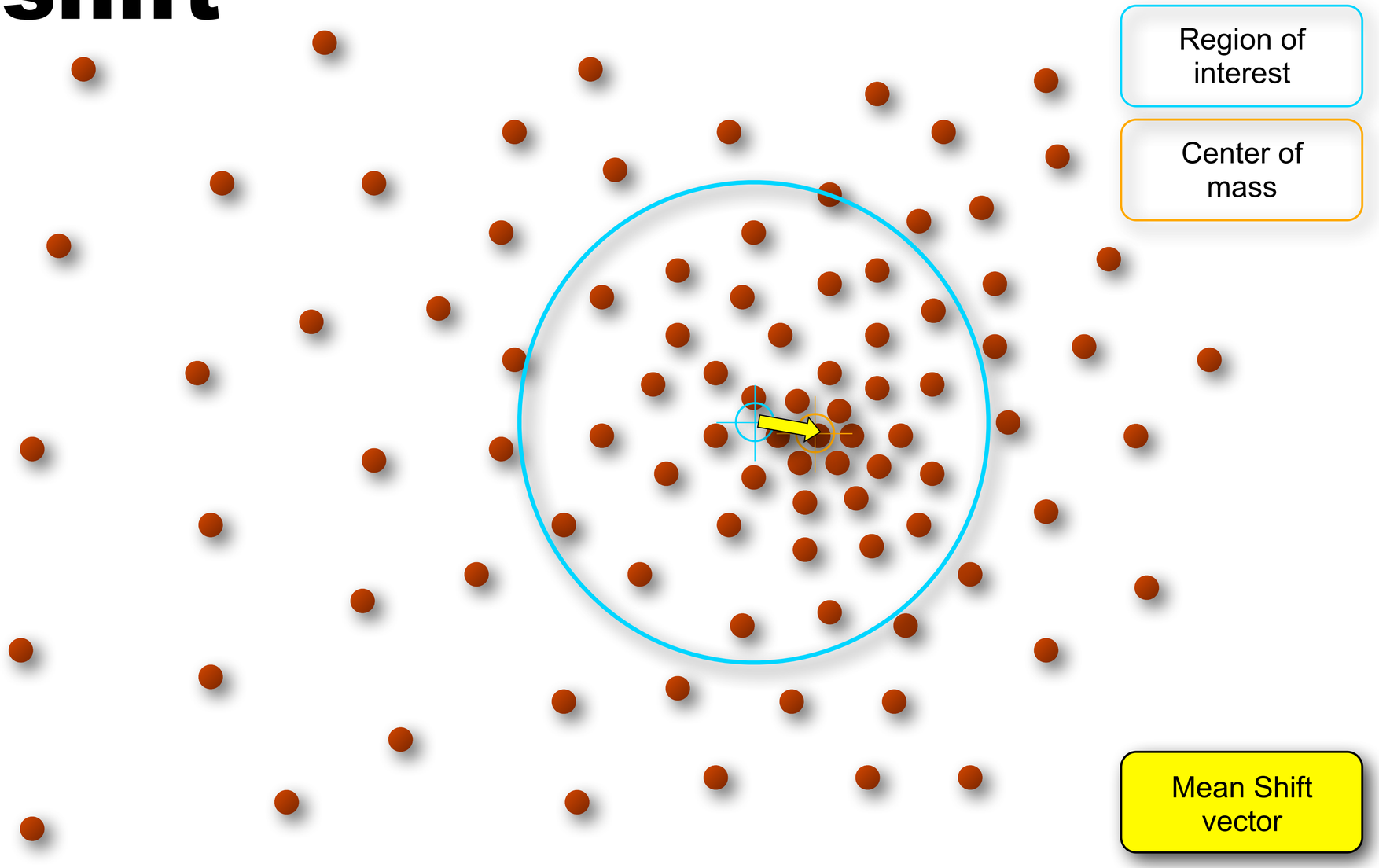
Mean Shift



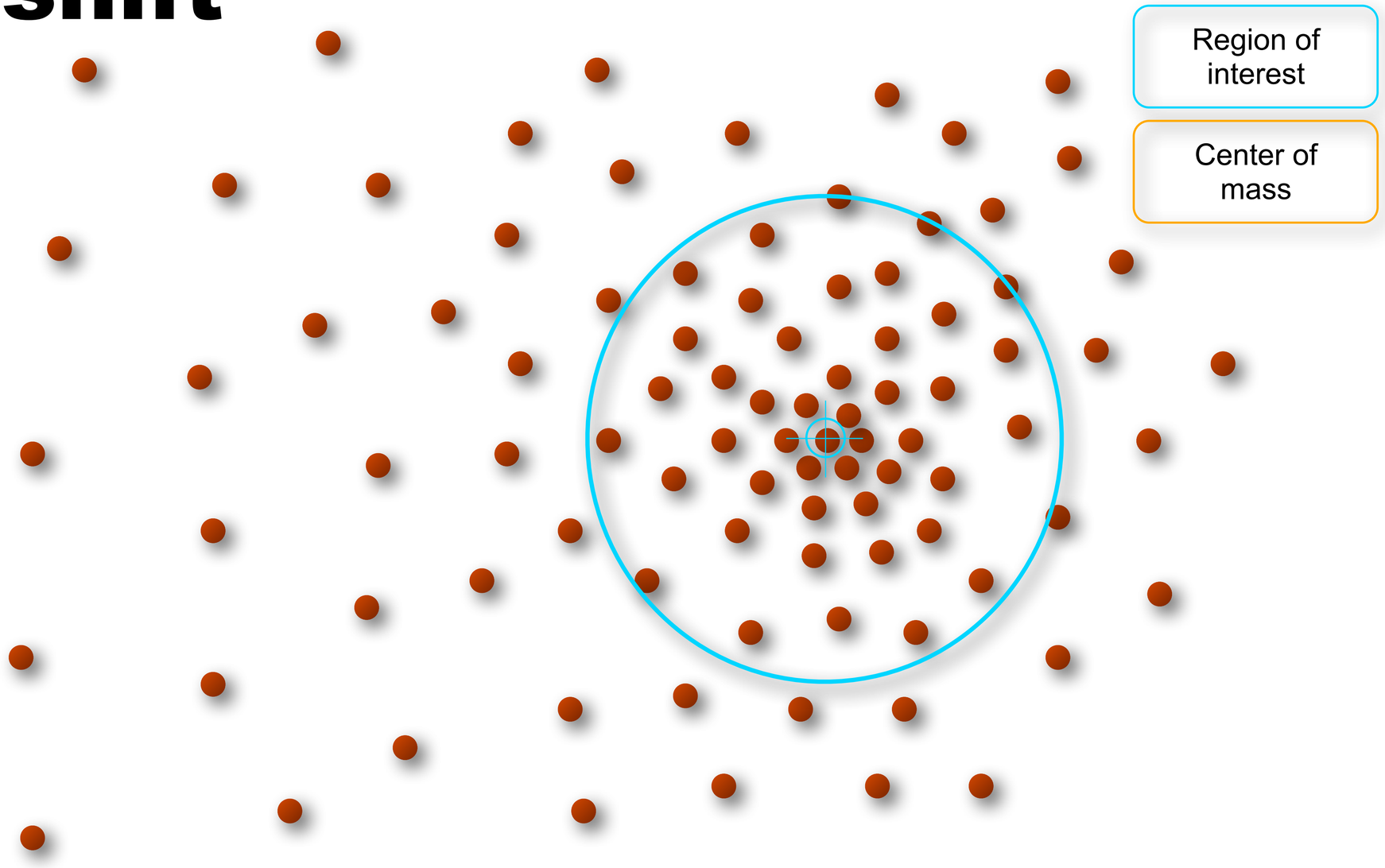
Mean shift



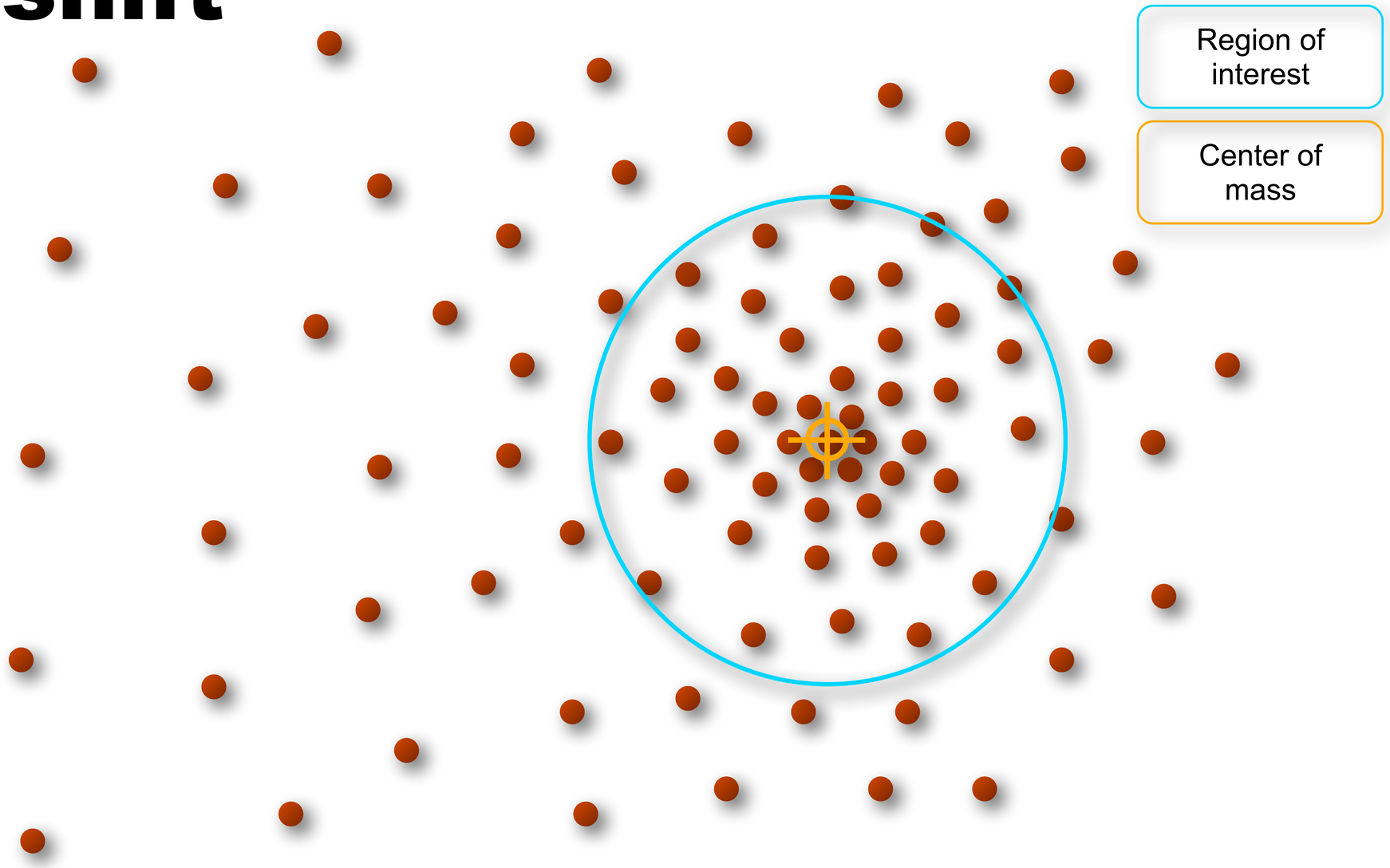
Mean shift



Mean shift



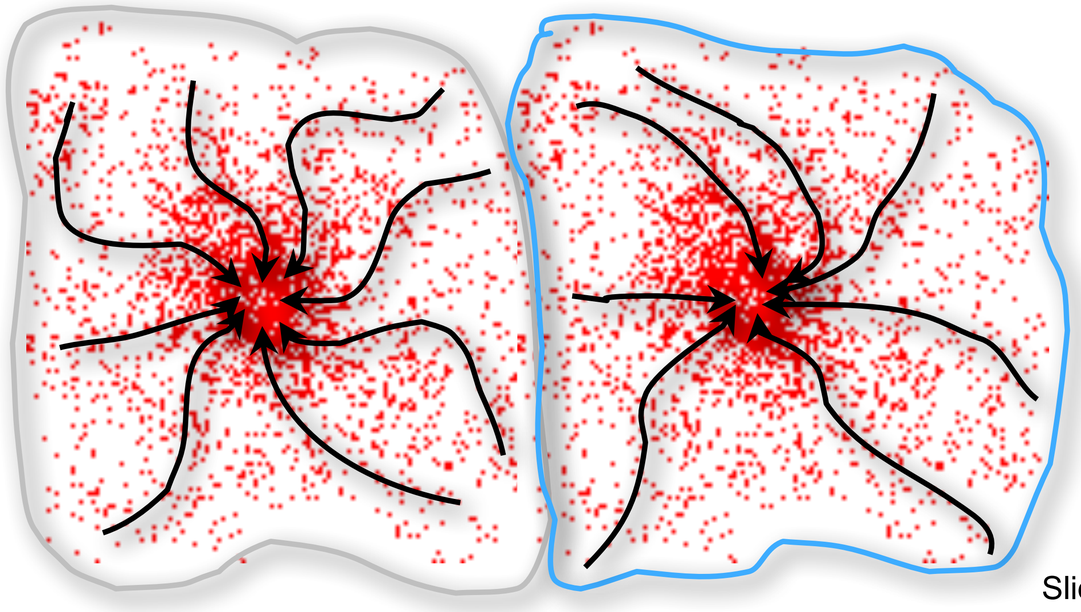
Mean shift



Slide by Y. Ukrainitz & B. Sarel

Mean shift Clustering

- **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 by Y. Ukrainitz & B. Sarel

Mean shift Segmentation Results

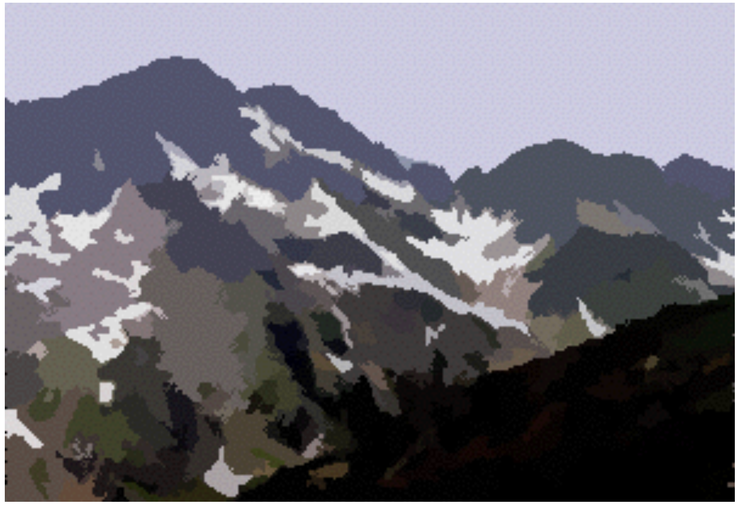
http://www.caip.rutgers.edu/~comanici/MSPAMI/msPamiResults.html

Mean shift Segmentation Results

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

Mean shift Segmentation Results



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

More Results

More Results

Pros/Cons

- Pros
  - Does not assume spherical clusters
  - Just a single parameter (window size)
  - Finds variable number of modes
  - Robust to outliers
- Cons
  - Output depends on window size
  - Computationally expensive
  - Does not scale well with dimension of feature space

Comparison

Watershed

- May be supervised or unsupervised
- No need for threshold parameter
- Partitions image into many regions
- To get good results best to specify number and positions of seeds

K-Means

- Unsupervised
- Works in high-dimensions
- May produce disjoint segments
- Must specify number of clusters but not position

Mean-Shift

- Unsupervised
- Works in high-dimensions (kind of)
- May produce disjoint segments
- “Only” specify windows size

Overview

What is image segmentation?

Types of segmentation algorithms

1. **Thresholding, region labelling and growing algorithms**
   - **(connected components, region growing, watershed)**
2. **Statistical Segmentation**
   - **(k-means, mean shift)**
3. **Graph based methods**
   - **(Merging algorithms, splitting algorithms, split/merge)**
4. **Edge based methods**
   - **(Intelligent Scissors, Snakes)**

Overview

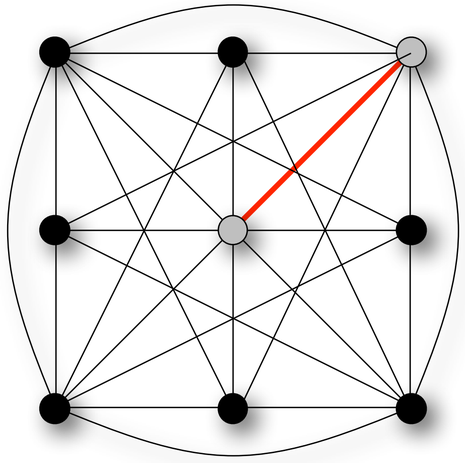
What is image segmentation?

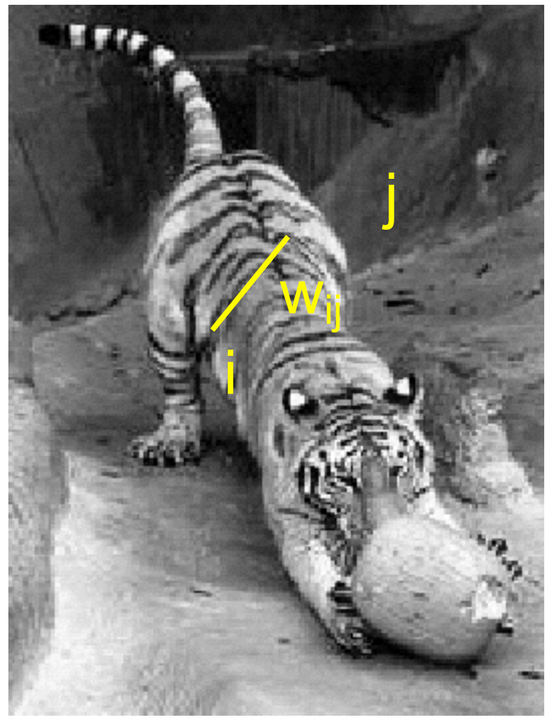
Types of segmentation algorithms

1. **Thresholding, region labelling and growing algorithms**
   - **(connected components, region growing, watershed)**
2. **Statistical Segmentation**
   - **(k-means, mean shift)**
3. **Graph based methods**
   - **(Merging algorithms, splitting algorithms, split/merge)**
4. **Edge based methods**
   - **(Intelligent Scissors, Snakes)**

Images as Graphs

- Node for every pixel
- Edge between every pair of pixels
  (or every pair of “sufficiently close” pixels)
- Each edge is weighted by the *affinity* or similarity of the two nodes





Source: S. Seitz

Greedy Merging Algorithm

Greedy Merging Algorithm

1. Initialize each pixel to a separate segment

Greedy Merging Algorithm

1. Initialize each pixel to a separate segment

2. Assign a cost c_{ij} to each edge (possibly based on pixel difference)

Greedy Merging Algorithm

1. Initialize each pixel to a separate segment

2. Assign a cost c_{ij} to each edge (possibly based on pixel difference)

3. Sort all edges by edge strength

Greedy Merging Algorithm

1. Initialize each pixel to a separate segment

2. Assign a cost c_{ij} to each edge (possibly based on pixel difference)

3. Sort all edges by edge strength

4. Connect edges in order of strength, merging segments as we go

Greedy Merging Algorithm

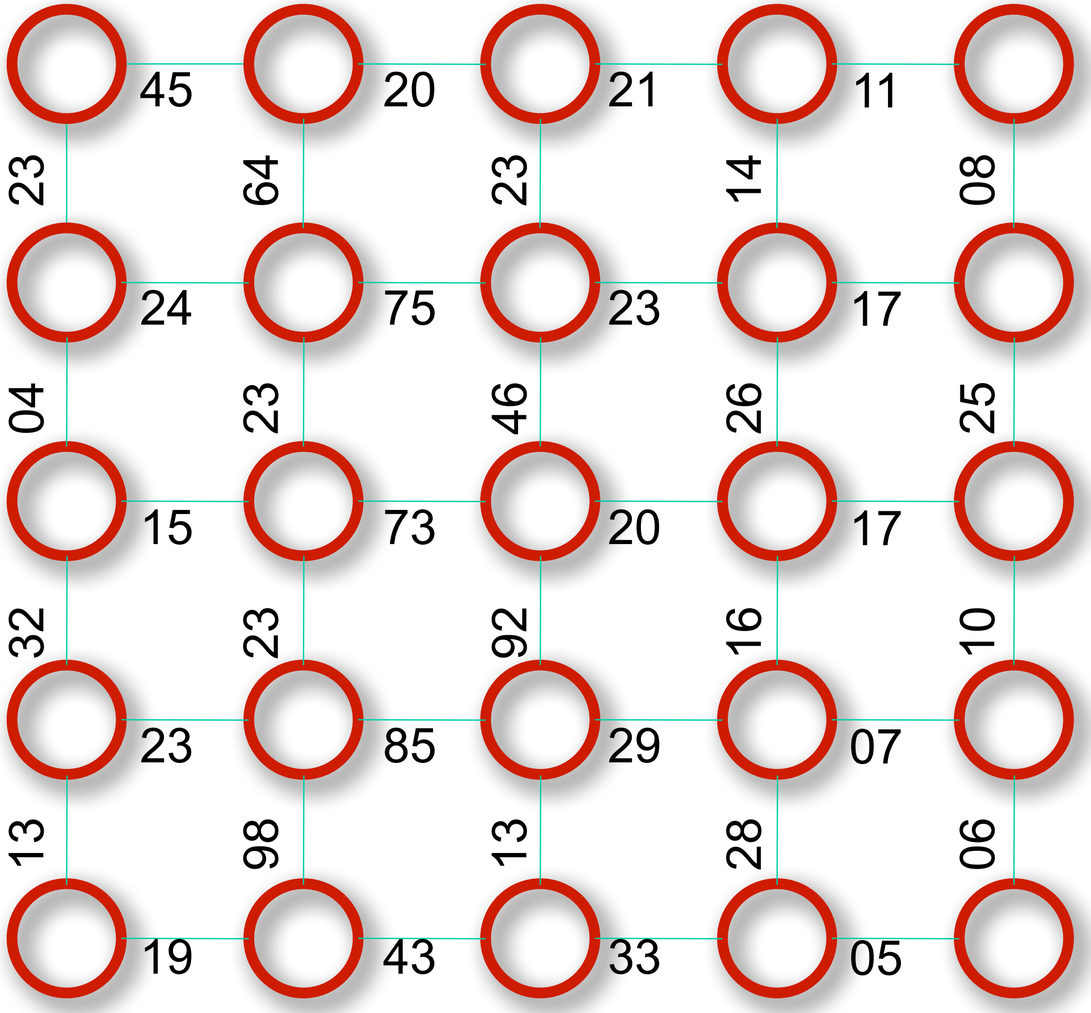
1. Initialize each pixel to a separate segment

2. Assign a cost c_{ij} to each edge (possibly based on pixel difference)

3. Sort all edges by edge strength

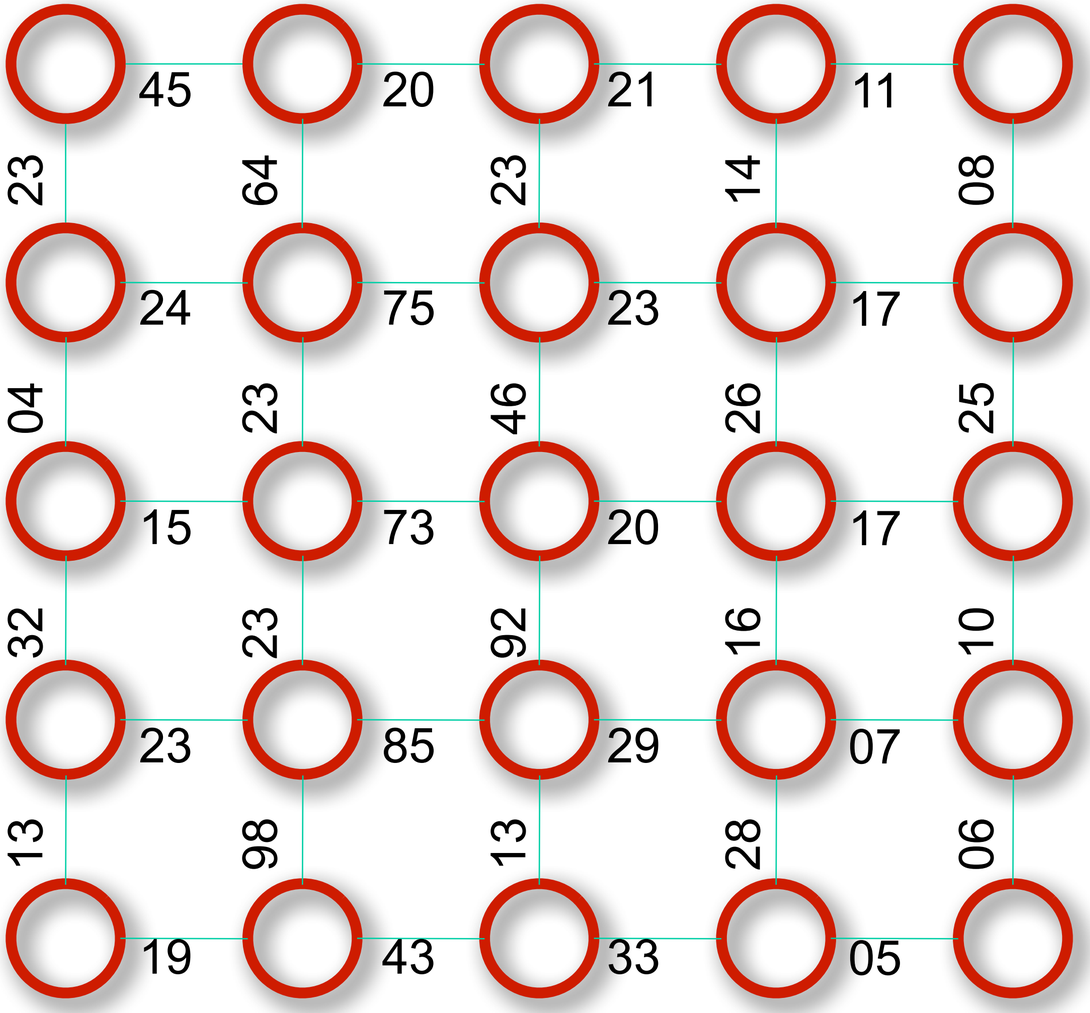
4. Connect edges in order of strength, merging segments as we go

5. Stop when we have reached some threshold edge strength



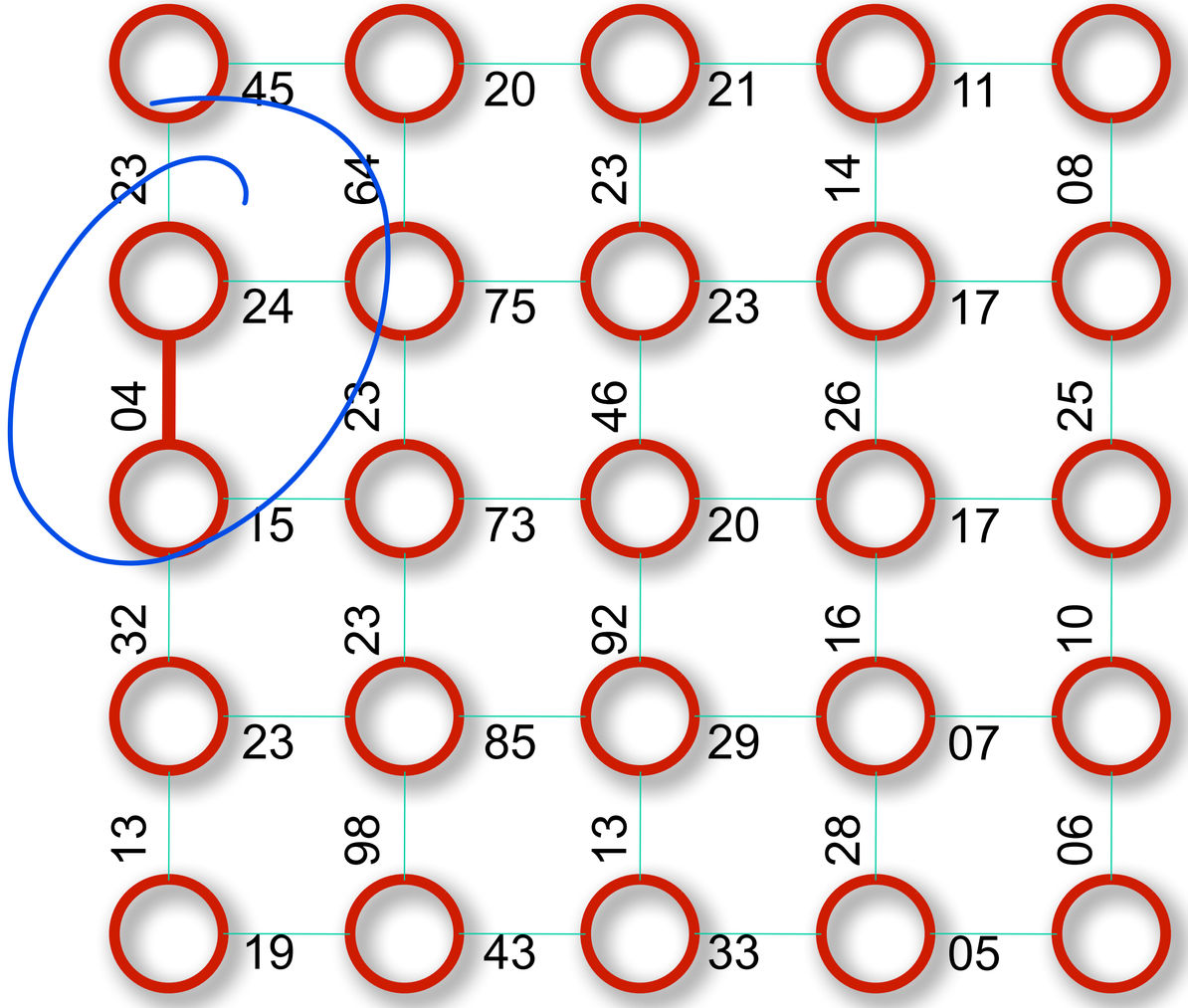
Extract all edge strengths

45, 20, 21, 11 ,24, 75, 23, 17, 15, 73, 20 ,17, 23, 85, 29, 07, 19,43,33, 05, 13, 22,
04,23, 98, 23, 23, 64, 13, 92, 46, 23, 28, 16, 26, 14, 06, 10, 25, 08



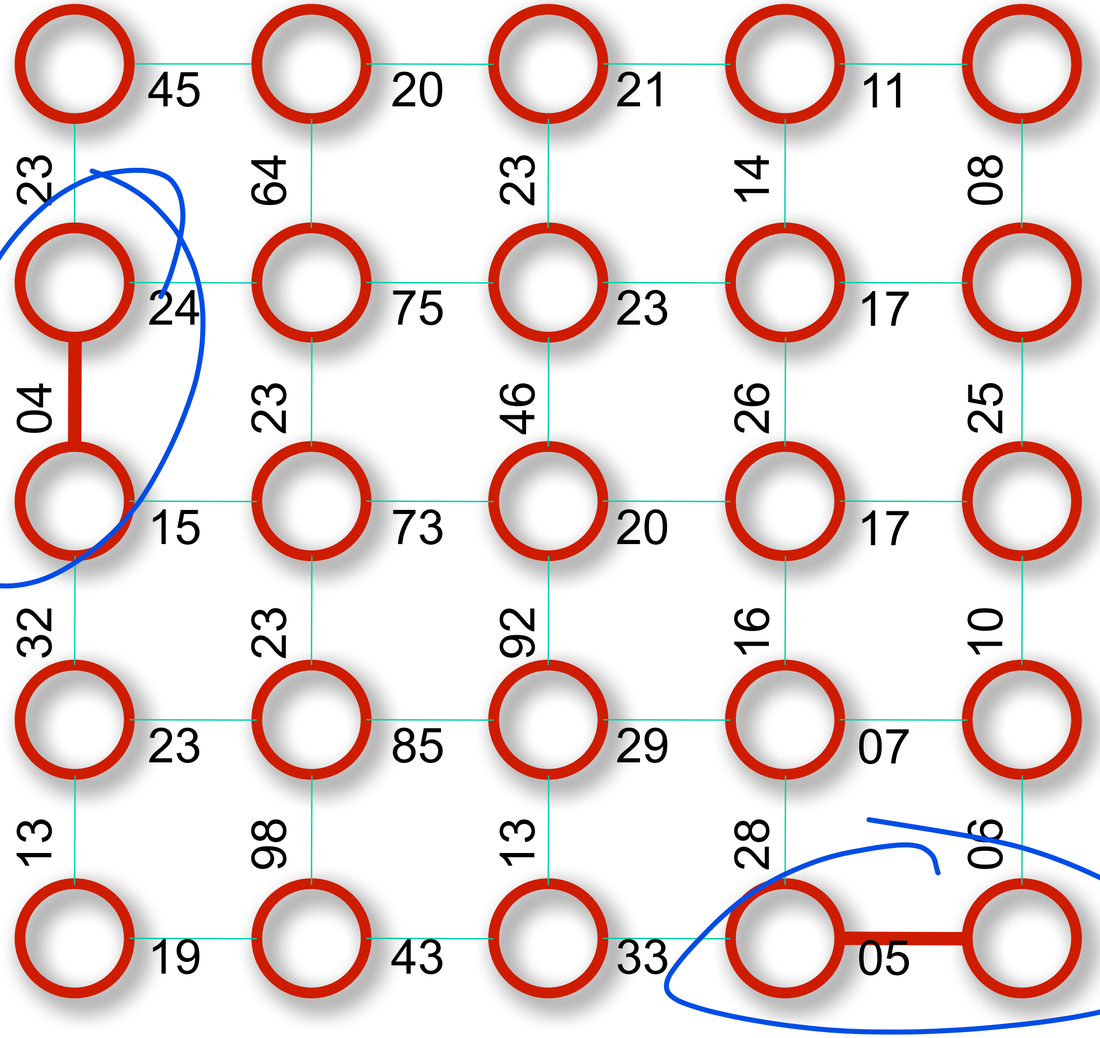
Sort edge strengths (remembering where they came from)

04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23,
23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98



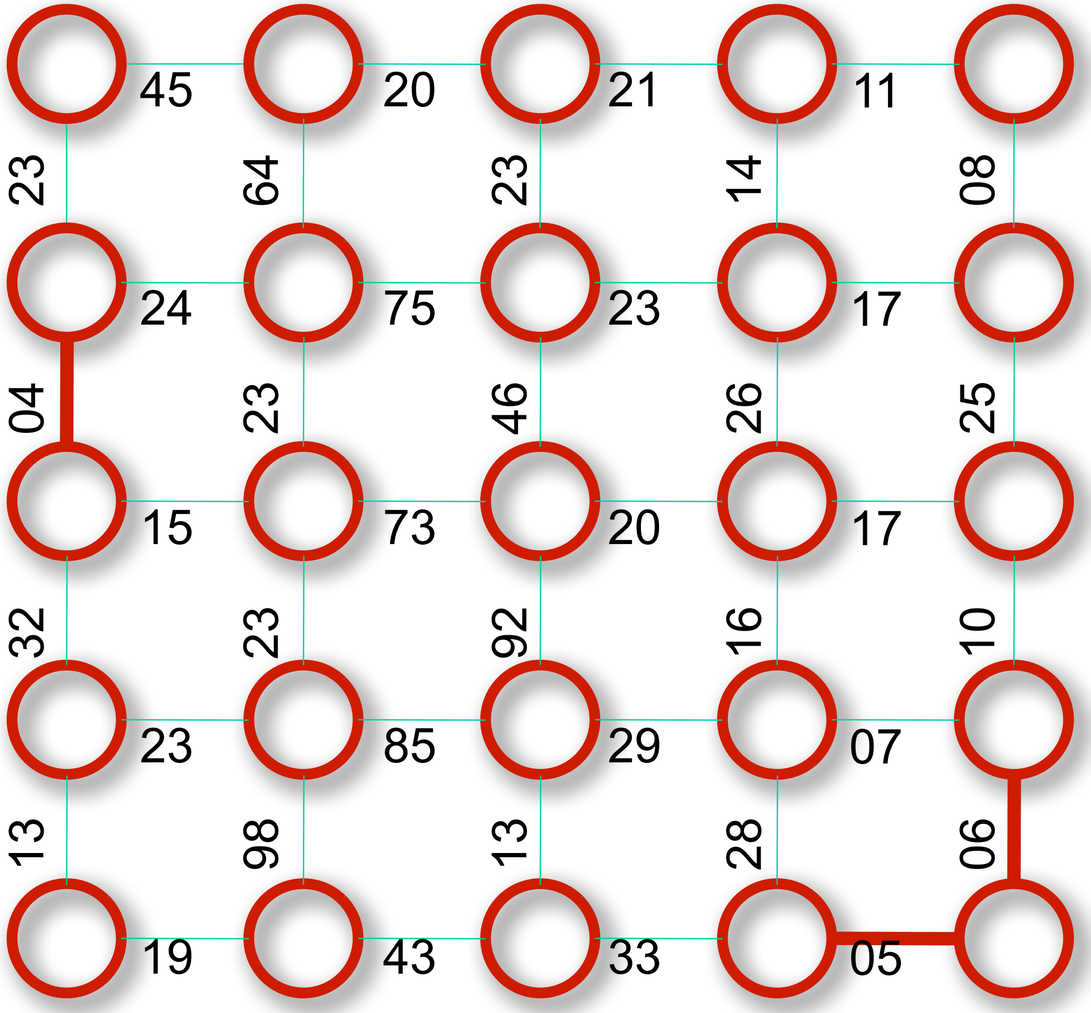
Connect together pixels, joining weakest first (until reach some threshold)

04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23, 23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98



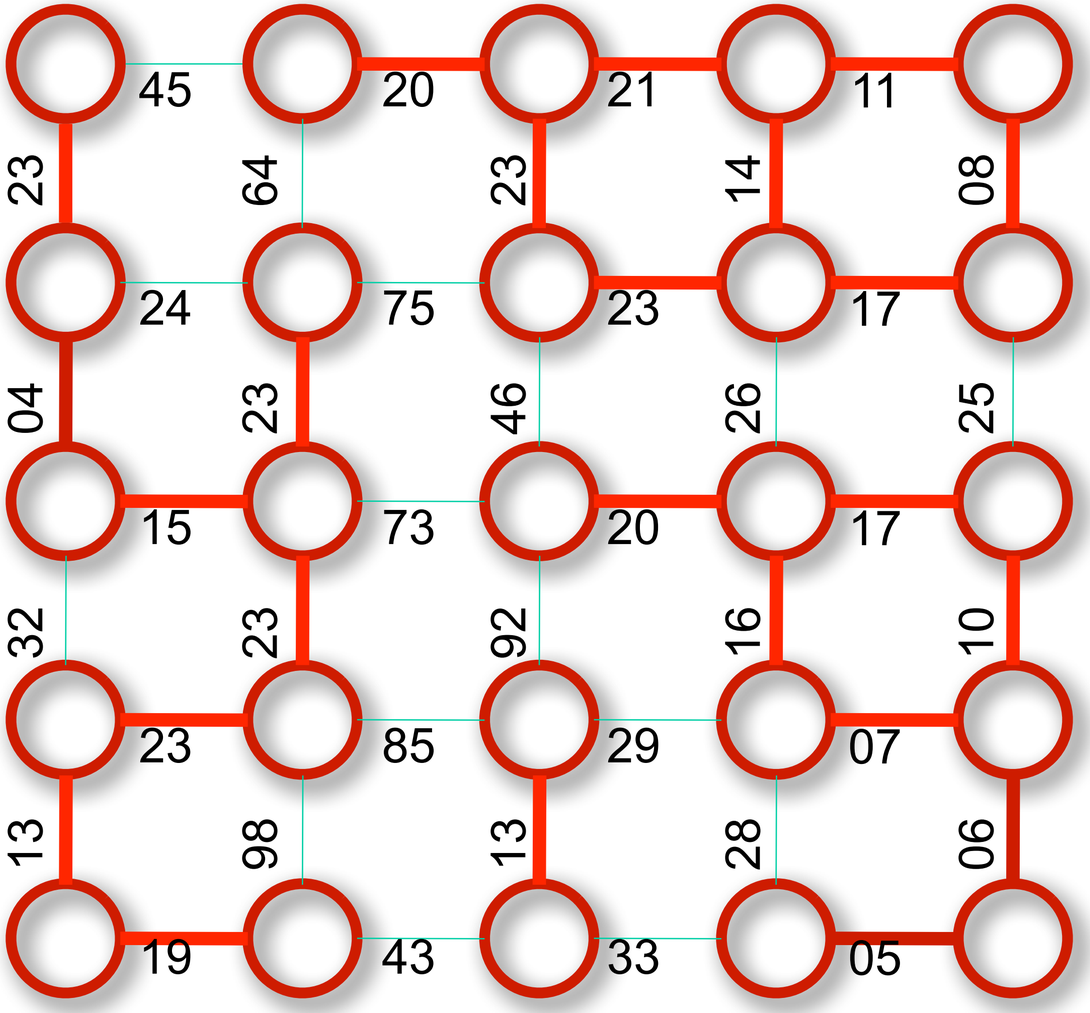
Connect together pixels, joining weakest first (until reach some threshold)

04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23, 23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98



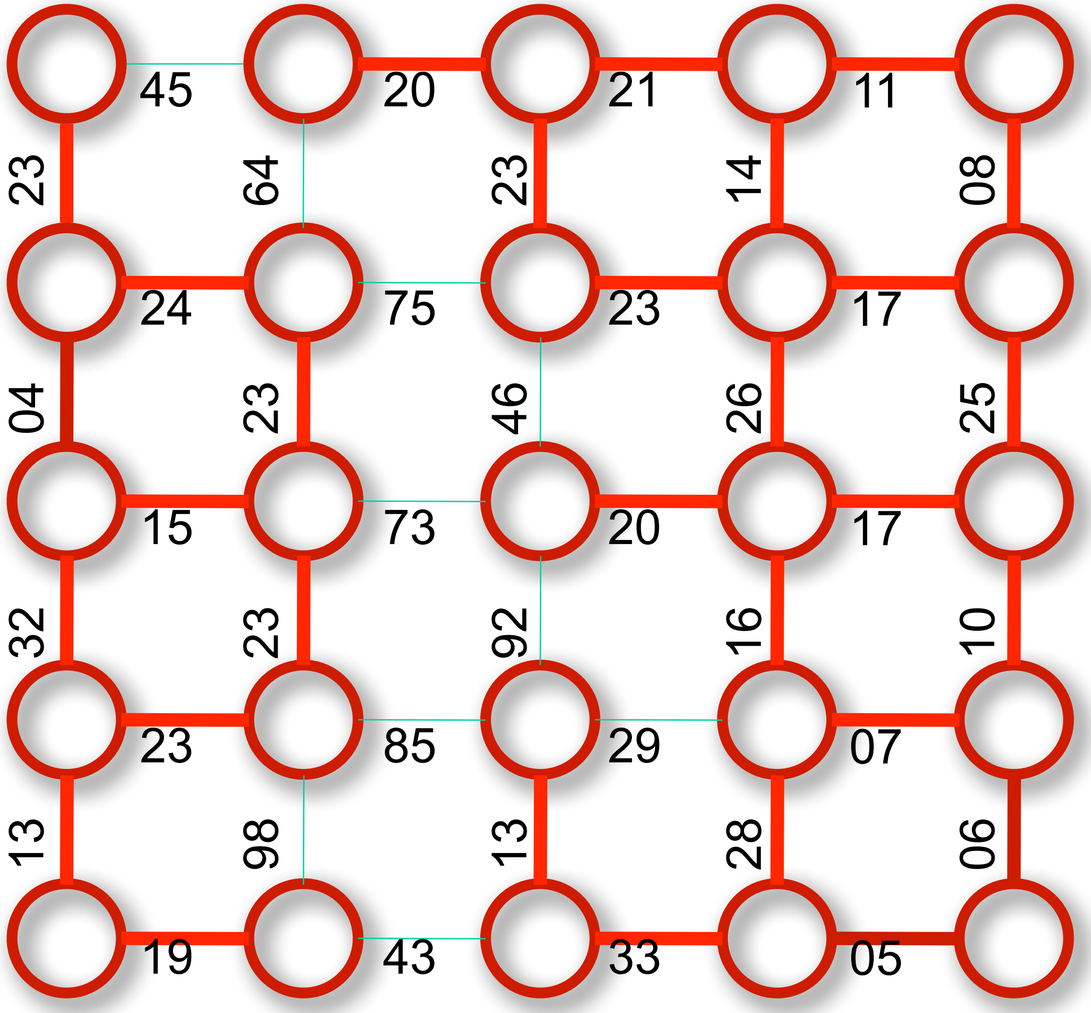
Connect together pixels, joining weakest first (until reach some threshold)

04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23, 23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98



Connect together pixels, joining weakest first (until reach some threshold)

04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23,
23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98



Connect together pixels, joining weakest first (until reach some threshold)

04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23,
23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98

A Better Merging Algorithm **Efficient Graph-Based Image Segmentation (Felzenswalb, Huttenlocher [IJCV 2004])**

- Assign weight $w(e_{ij})$ to each edge e_{ij}

- Sort edges by weight and run through

- Define Int(C) to be the maximum cost weight in the minimum spanning tree of cluster C

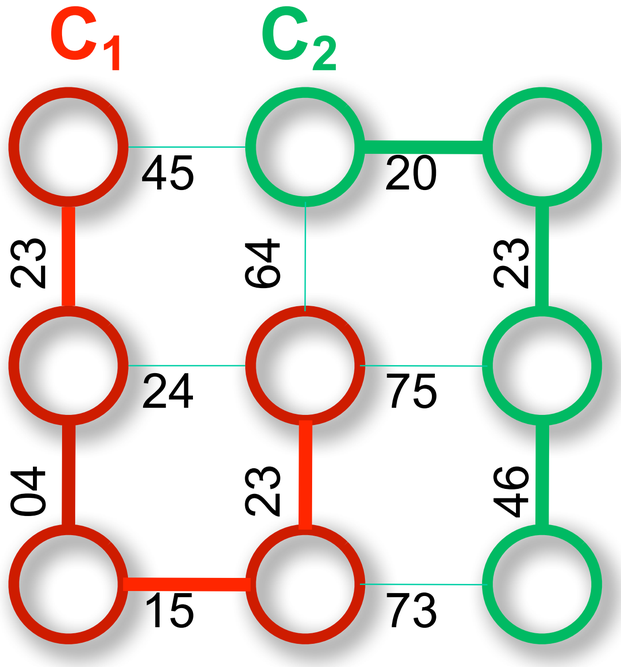
- Define Dif(C1,C2) to be lowest cost link joining components

- Merge components if

$$\mathrm{Dif}(C_1, C_2) < \min(\mathrm{Int}(C_1) + \tau(C_1), \mathrm{Int}(C_2) + \tau(C_2))$$

where

$$\tau(C) = k / |C|$$



A Better Merging Algorithm **Efficient Graph-Based Image Segmentation (Felzenswalb, Huttenlocher [IJCV 2004])**

- Assign weight $w(e_{ij})$ to each edge e_{ij}

- Sort edges by weight and run through

- Define Int(C) to be the maximum cost weight in the minimum spanning tree of cluster C

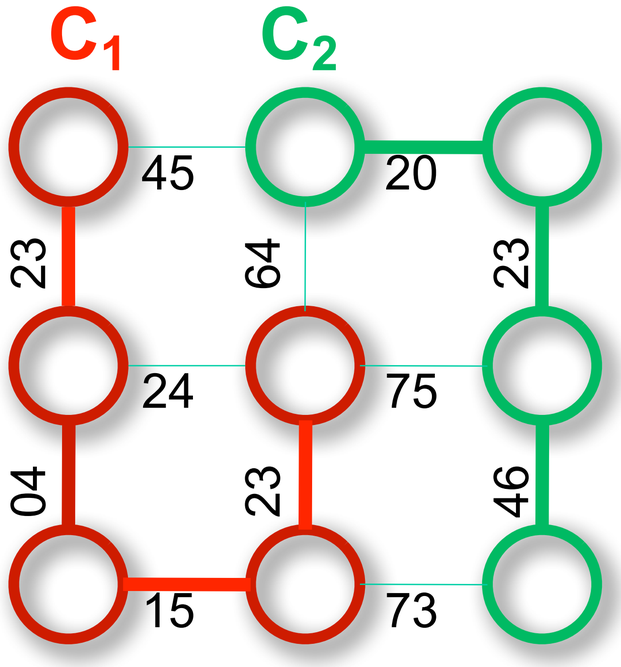
- Define Dif(C1,C2) to be lowest cost link joining components

- Merge components if

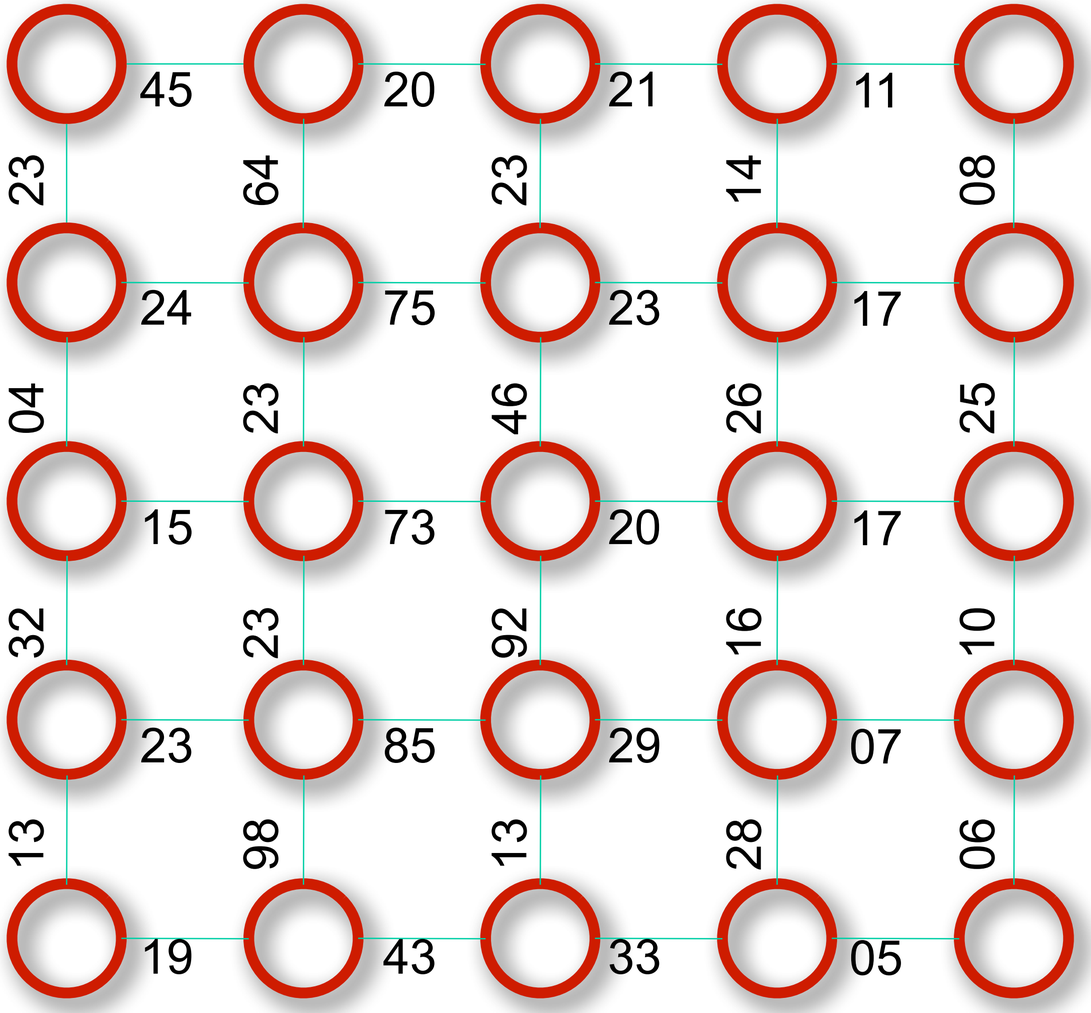
$$\mathrm{Dif}(C_1, C_2) < \min(\mathrm{Int}(C_1) + \tau(C_1), \mathrm{Int}(C_2) + \tau(C_2))$$

where

$$\tau(C) = k / |C|$$



Here: Int(C1) = 23
Int(C2) = 46
Dif(C1,C2) = min(45,64,73,75)=45



Sort edge strengths (remembering where they came from)

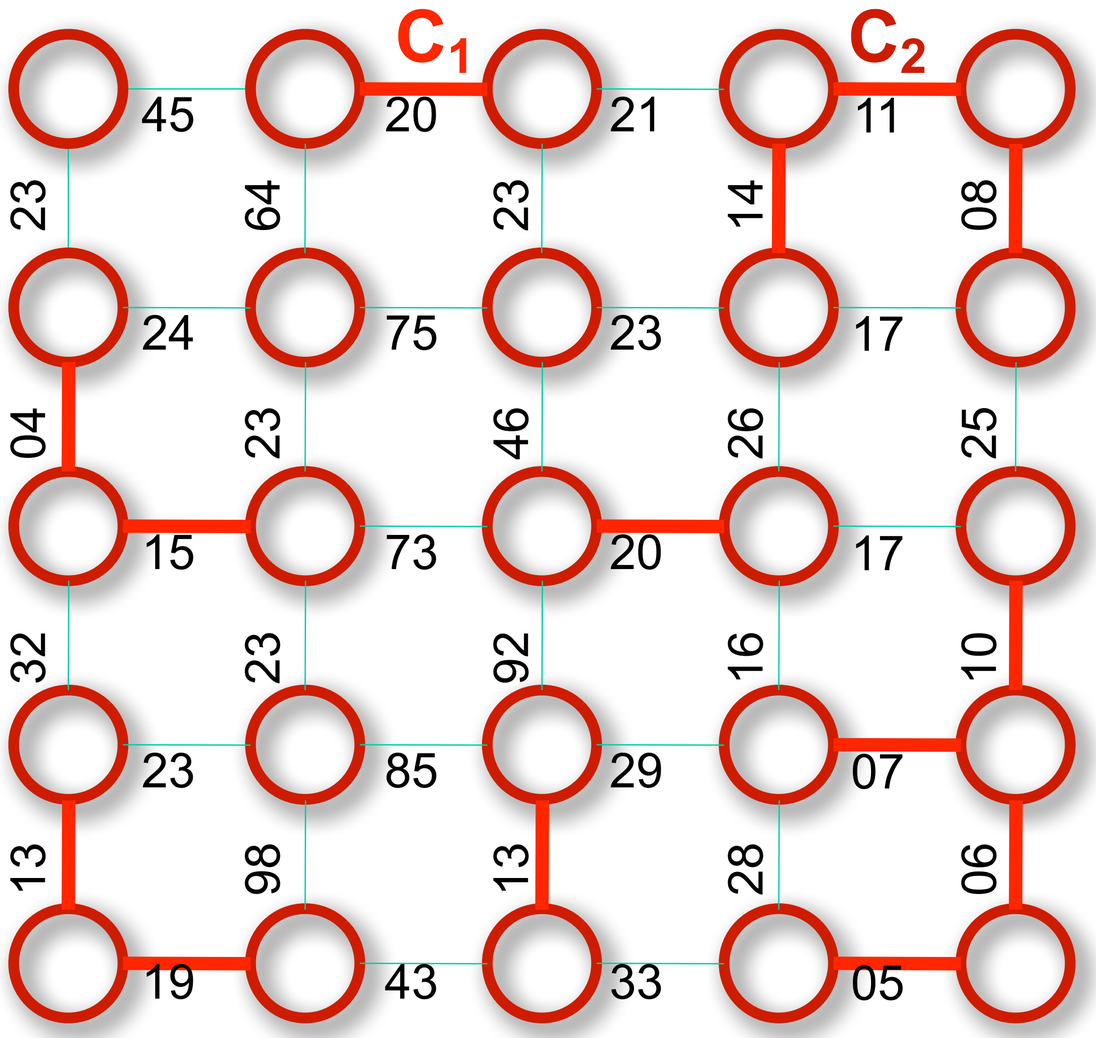
04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23,
23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98

Let k = 50

Int(C1) = 20
Int(C2) = 14
Dif(C1,C2) = 21

Mint = min(20+50/2,14+50/4)
= 26.5

Dif(C1,C2)<Mint
so merge



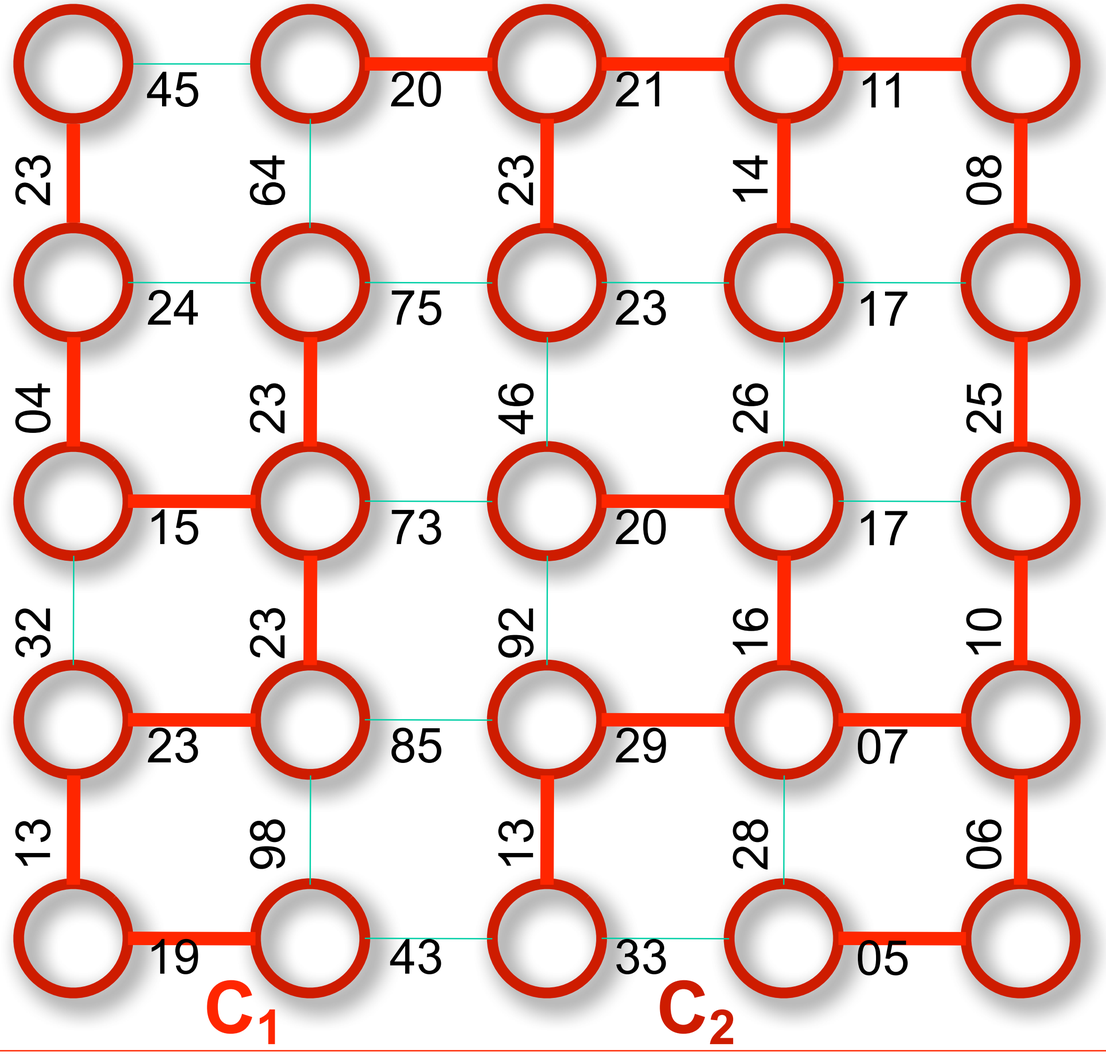
Sorted Edge Strengths: 04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23, 23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98

Let k = 50

Int(C1) = 23
Int(C2) = 29
Dif(C1,C2) = 43

Mint = min(23+50/9,19+50/16)
= 28.566

Dif(C1,C2)>Mint
so don't merge



Sorted Edge Strengths: 04, 05, 06, 07, 08, 10, 11, 13, 13, 14, 15, 16, 17, 17, 19, 20, 20, 21, 22, 23, 23, 23, 23, 23, 23, 24, 25, 26, 28, 29, 33, 43, 45, 46, 64, 73, 75, 85, 92, 98

Merging Example

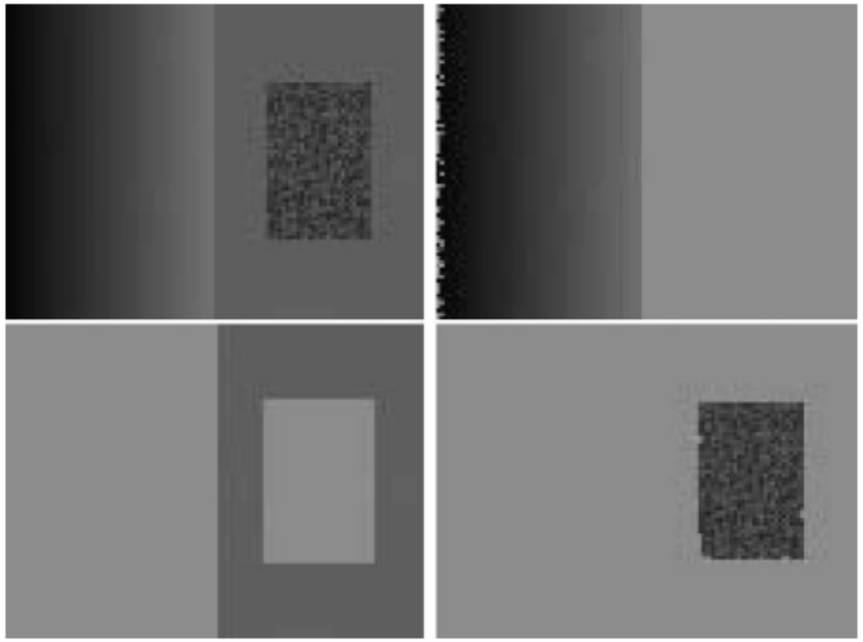


Figure 1. A synthetic image with three perceptually distinct regions, and the three largest regions found by our segmentation method (image 320×240 pixels; algorithm parameters $\sigma = 0.8$, $k = 300$, see Section 5 for an explanation of the parameters).

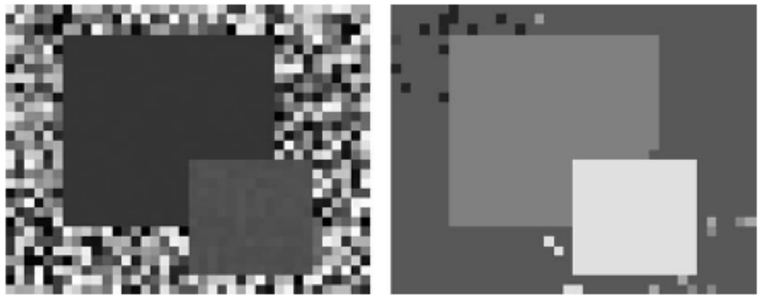


Figure 6. A synthetic image (40×32 grey image) and the segmentation using the nearest neighbor graph ($\sigma = 0$, $k = 150$).

Example Results

Example Results

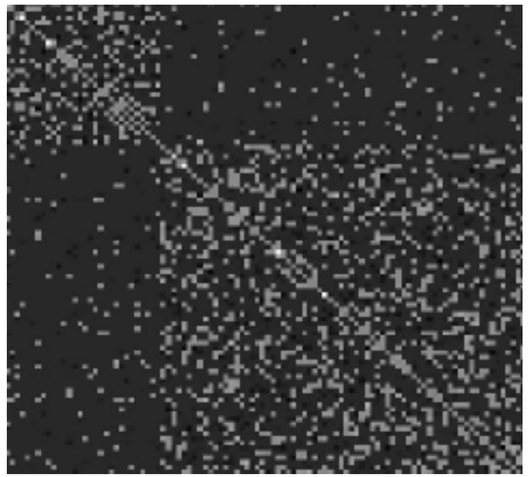
Example Results

Example Results

- Very fast
- But sensitive to noise
- Greedily chooses (too) large regions

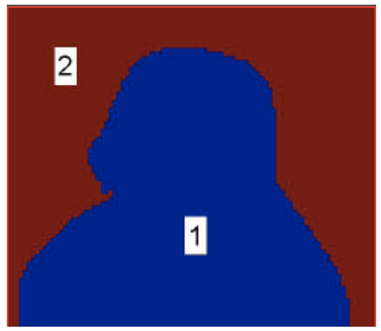
Splitting Algorithm **Normalized Cuts, Shi & Malik (2000)**

Image (I)



Graph Affinities
(W)

Eigenvector
X(W)



Discretization

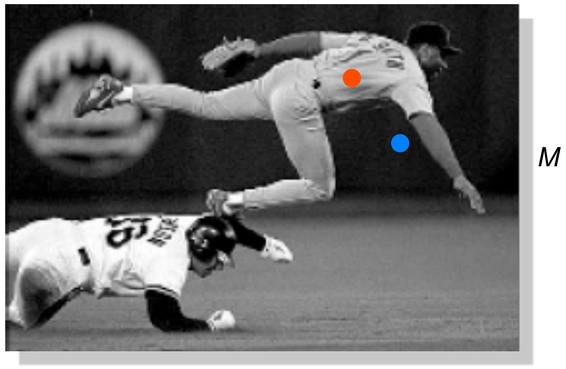
$$Ncut(A,B) := cut(A,B)\left(\frac{1}{vol(A)} + \frac{1}{vol(B)}\right)$$

$$(D - W)X = \lambda DX$$

$$X_A(i) = \begin{cases} 1 & if \quad i \in A \\ 0 & if \quad i \notin A \end{cases}$$

Affinity Matrix

N pixels

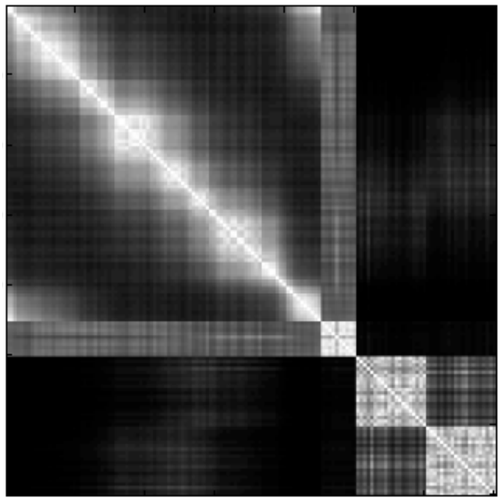


M pixels

Affinity Matrix

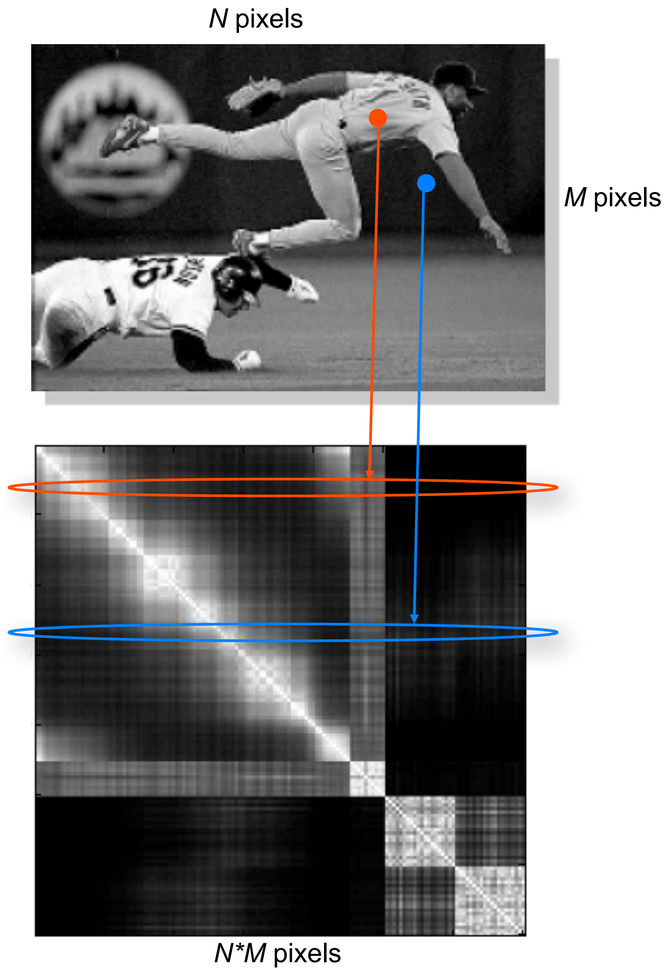
N pixels

M pixels

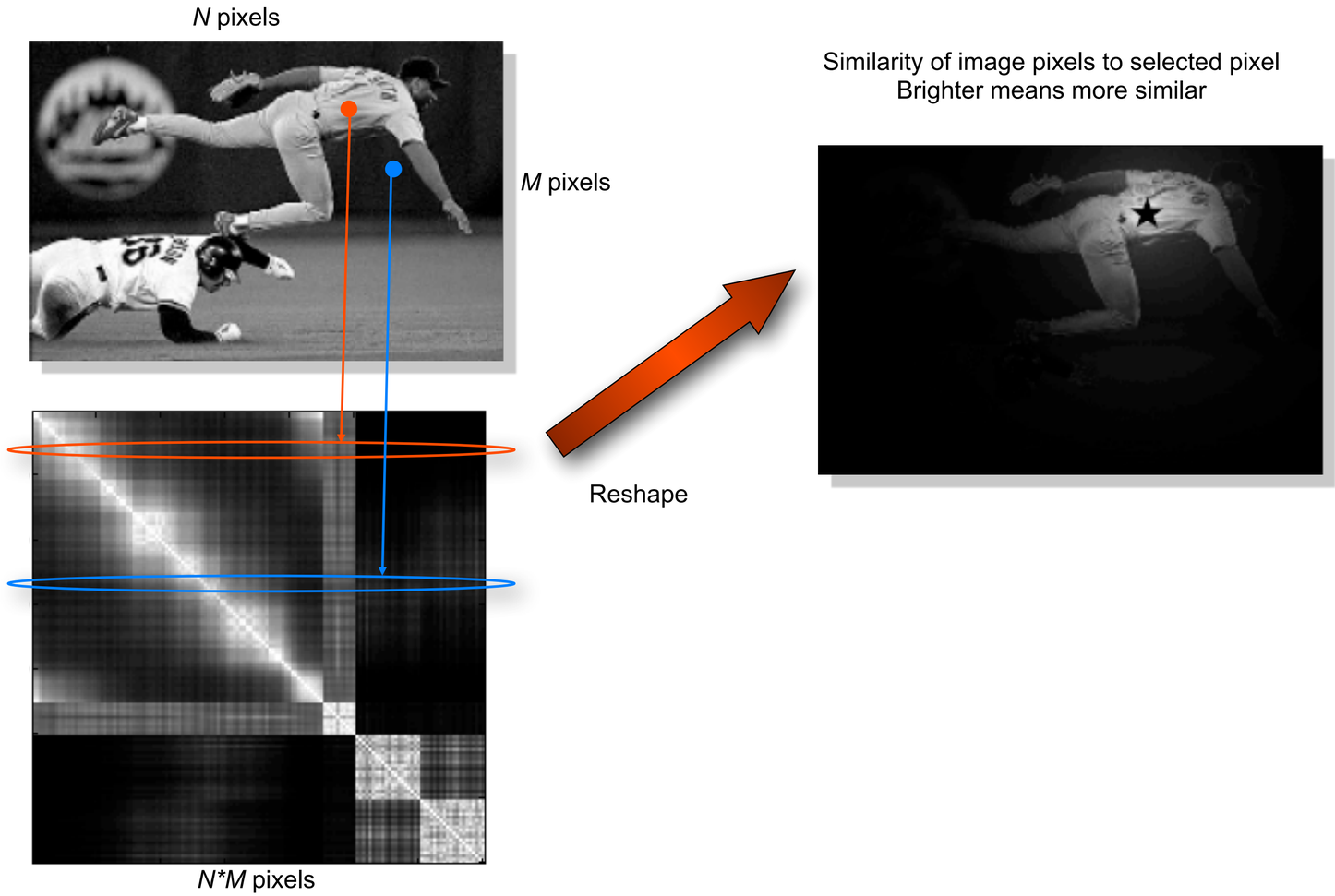


*N*M* pixels

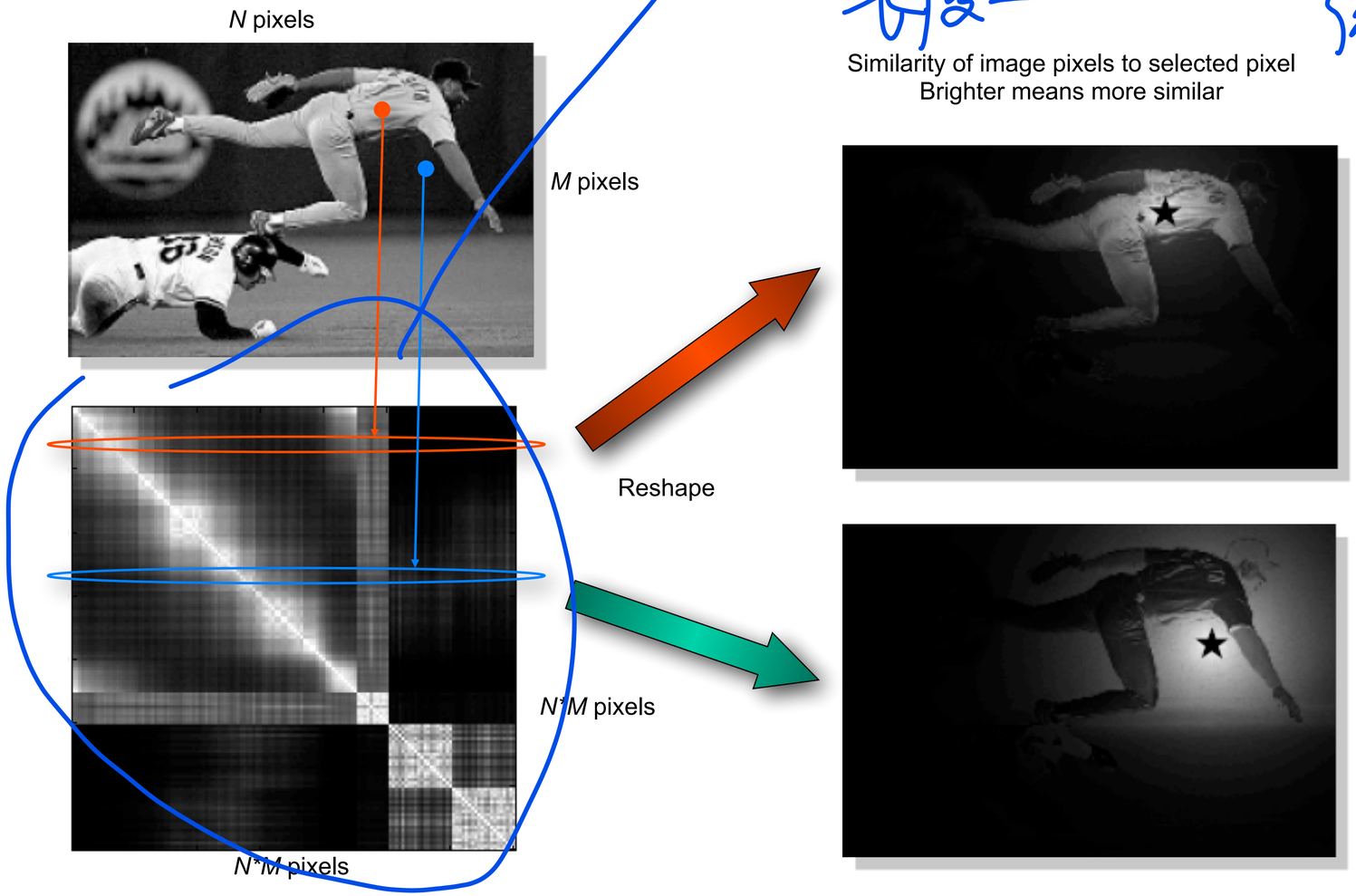
Affinity Matrix



Affinity Matrix



Affinity Matrix



Pixel Similarity Functions

Intensity

$$W(i,j) = e^{\frac{-\left\|I_{(i)} - I_{(j)}\right\|_2^2}{\sigma_I^2}}$$

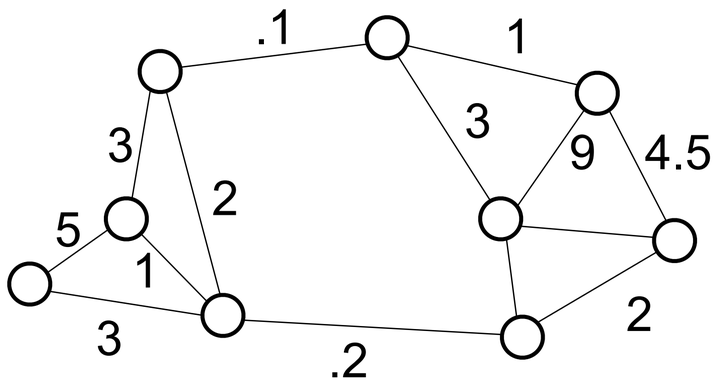
Distance

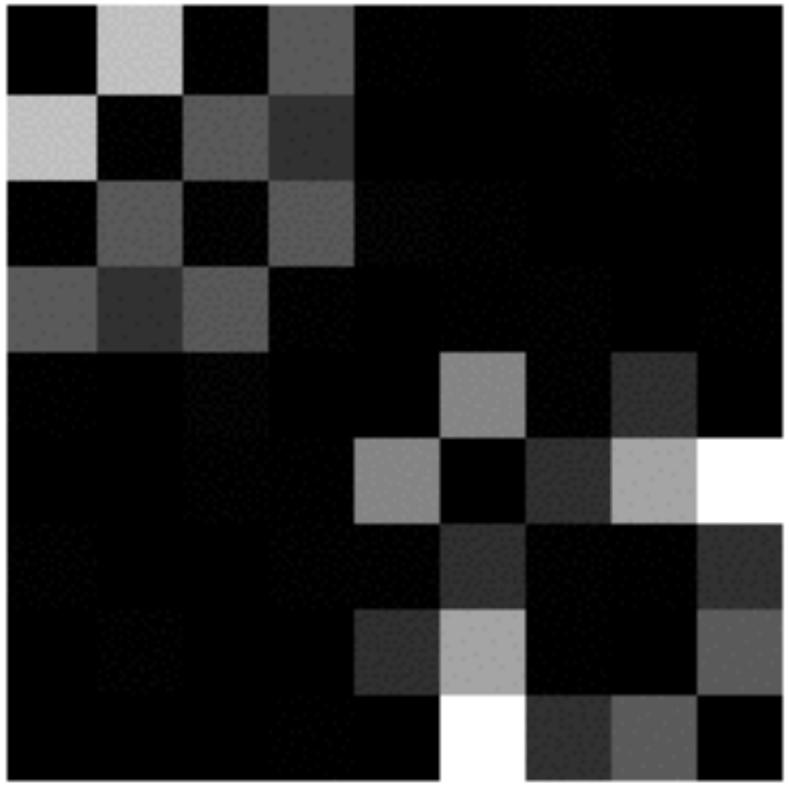
$$W(i,j) = e^{\frac{-\left\|X_{(i)} - X_{(j)}\right\|_2^2}{\sigma_X^2}}$$

Texture

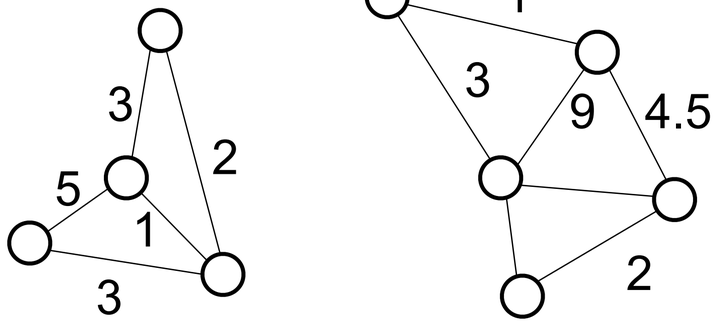
$$W(i,j) = e^{\frac{-\left\|c_{(i)} - c_{(j)}\right\|_2^2}{\sigma_c^2}}$$

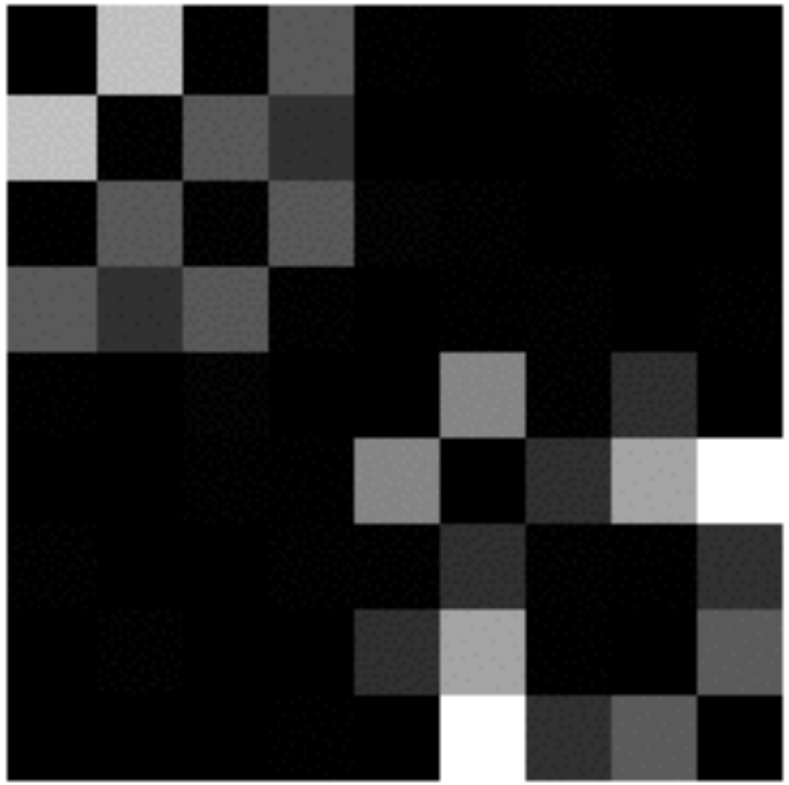
Minimum Cut and Clustering





Minimum Cut and Clustering



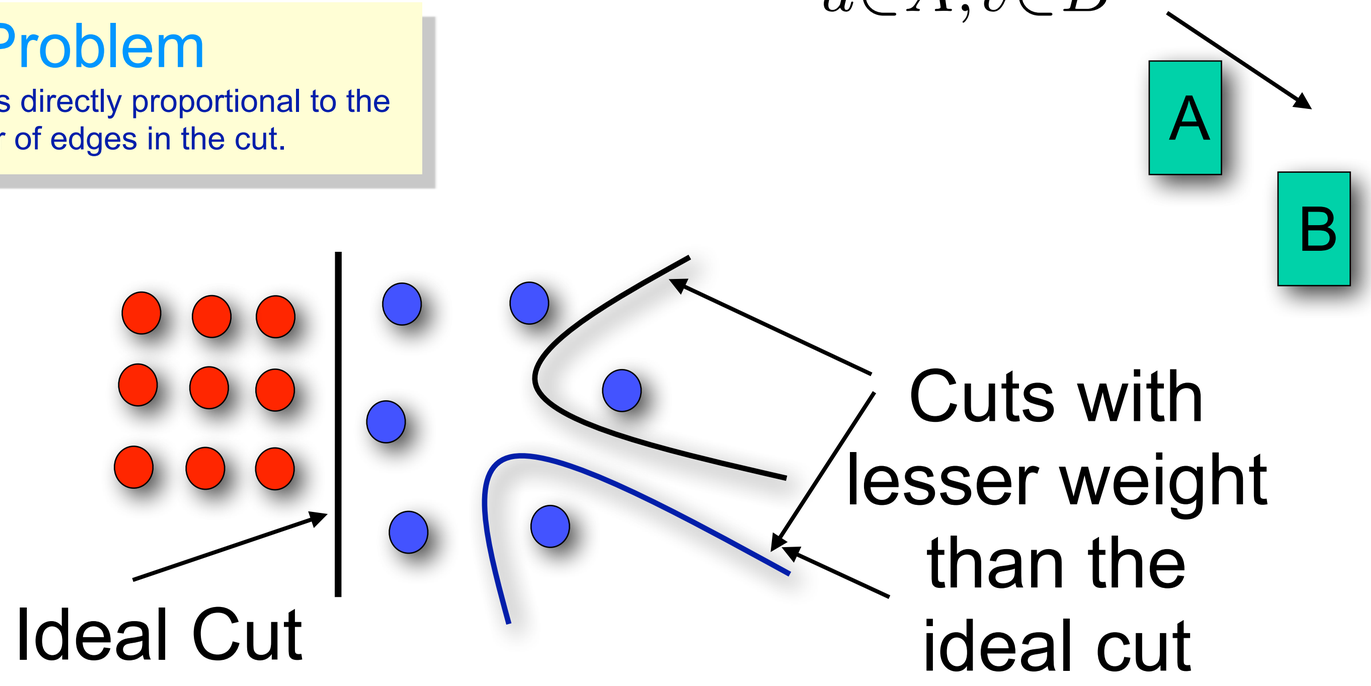


Minimum Cut

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

Problem

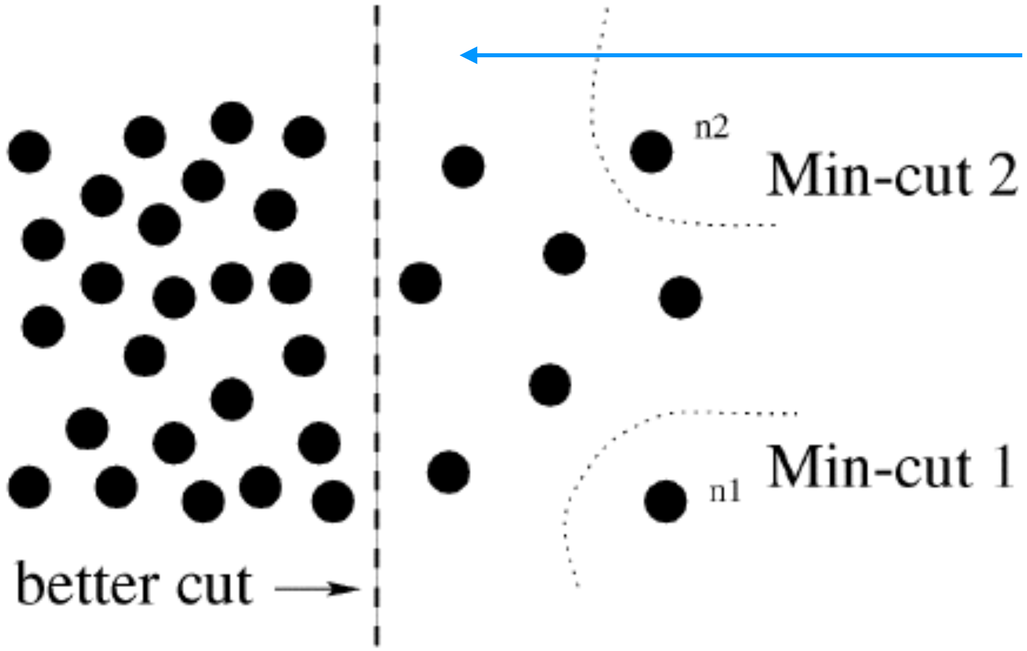
Weight of cut is directly proportional to the number of edges in the cut.



First proposed by Wu and Leahy

Normalized Cut

$$Ncut(A,B) := cut(A,B)\left(\frac{1}{vol(A)} + \frac{1}{vol(B)}\right)$$

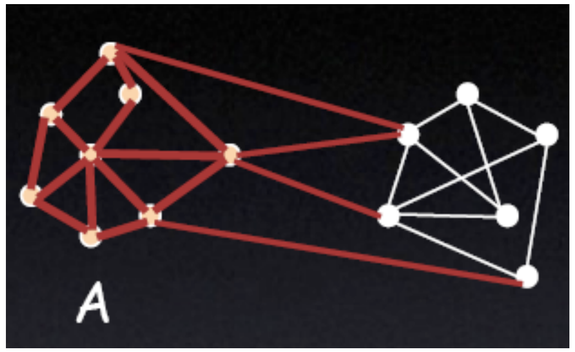


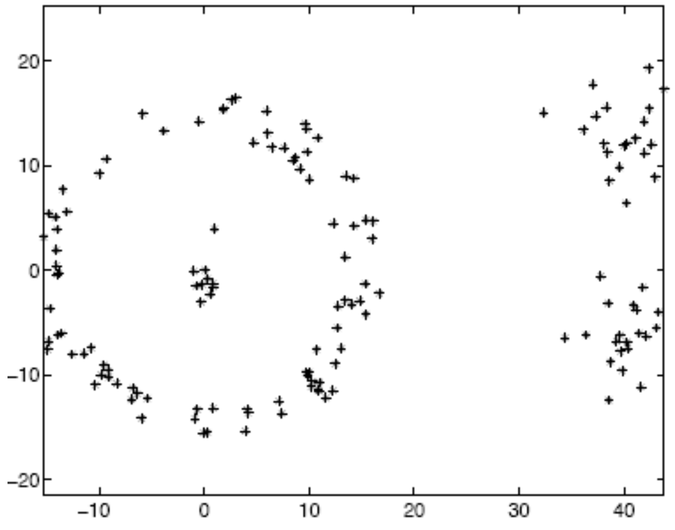
finds better cut

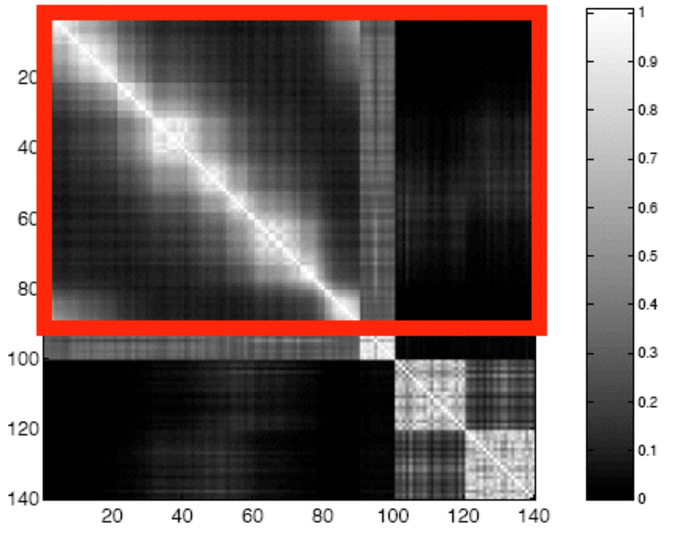
Graph Terminology

Volume of set:

$$vol(A) = \sum_{i \in A} d_i, A \subset V$$





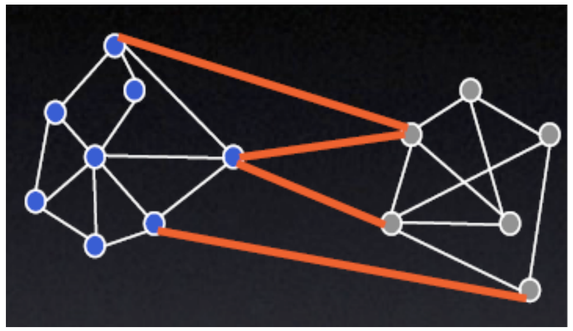


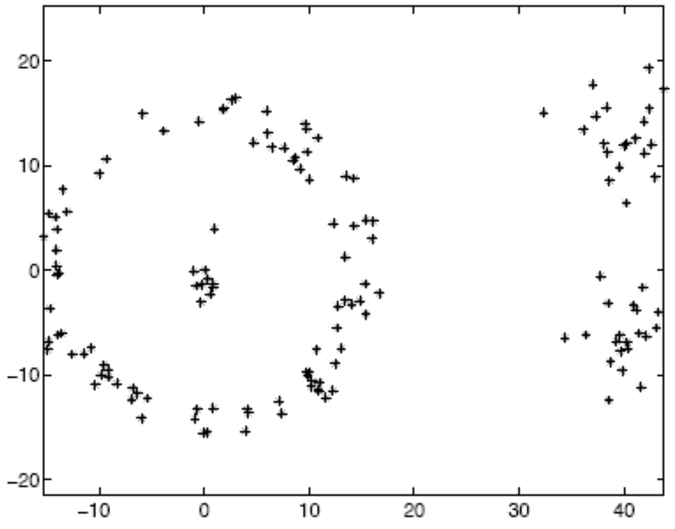
Slides from Jianbo Shi

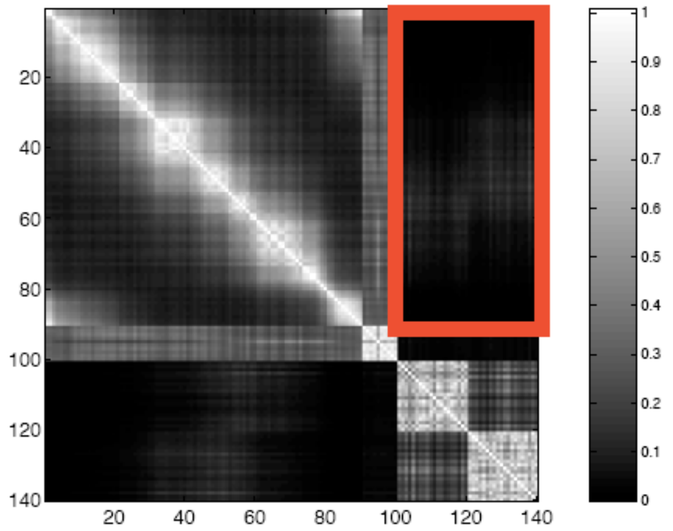
Graph Terminology

Cuts in graphs:

$$cut(A, \bar{A}) = \sum_{i \in A, j \in \bar{A}} w_{i,j}$$





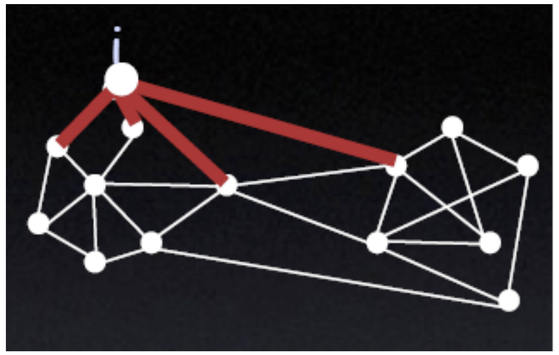


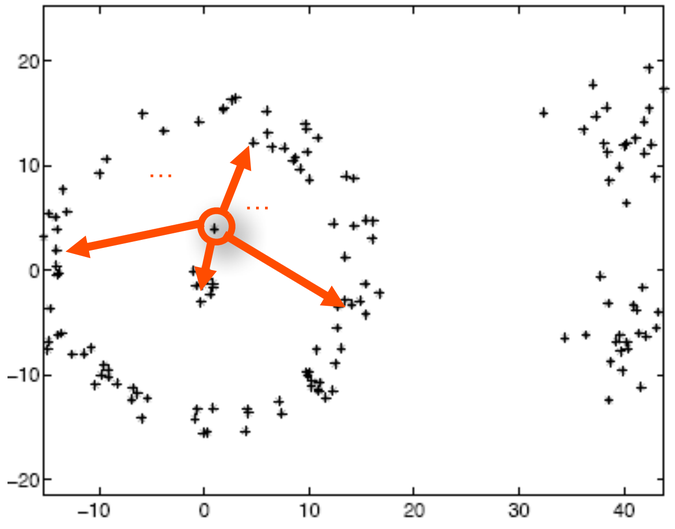
Slides from Jianbo Shi

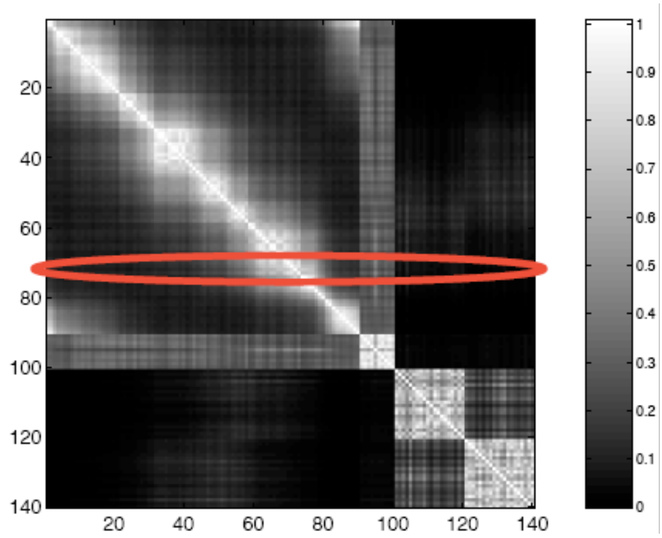
Graph Terminology

- **Degree of node:**

$$d_i = \sum_j w_{i,j}$$







Slides from Jianbo Shi

Finding Minimum Normalized-Cut

- **It can be shown that**

$$\min N_{cut} = \min_{\mathbf{y}} \frac{\mathbf{y}^T(D-W)\mathbf{y}}{\mathbf{y}^T D\mathbf{y}}$$

such that

$$y(i) \in \{1, -b\}, 0 < b \leq 1 \text{ ,and } y^T D1 = 0$$

- **If *y* is allowed to take real values then the minimization can be done by solving the generalized eigenvalue system**

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

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

NCut Algorithm

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

NCut Algorithm

- **Compute matrices W and D**

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

NCut Algorithm

- **Compute matrices W and D**
- **Solve $(D - W)\mathbf{y} = \lambda D\mathbf{y}$ for eigenvectors with the smallest eigenvalues**

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

NCut Algorithm

- Compute matrices W and D
- Solve $(D - W)\mathbf{y} = \lambda D\mathbf{y}$ for eigenvectors with the smallest eigenvalues
- Use the eigenvector with second smallest eigenvalue to bipartition the graph

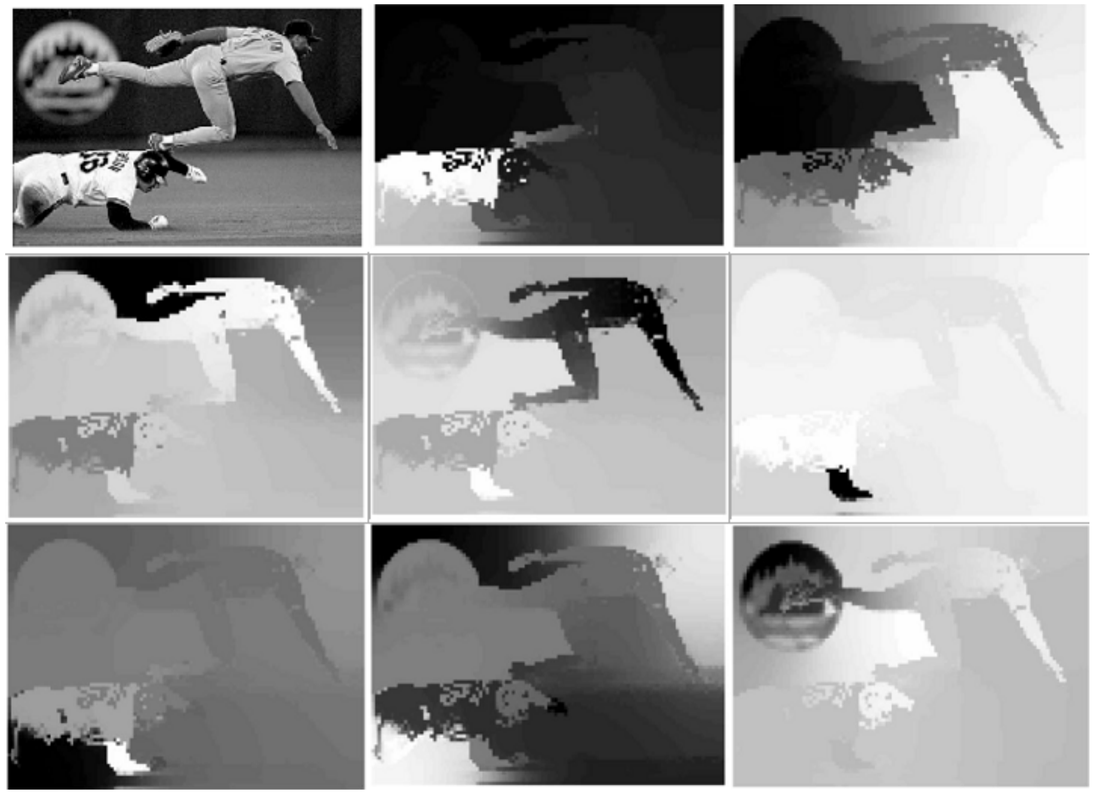
* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

NCut Algorithm

- Compute matrices W and D
- Solve $(D - W)\mathbf{y} = \lambda D\mathbf{y}$ for eigenvectors with the smallest eigenvalues
- Use the eigenvector with second smallest eigenvalue to bipartition the graph
- Recursively partition the segmented parts if necessary

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

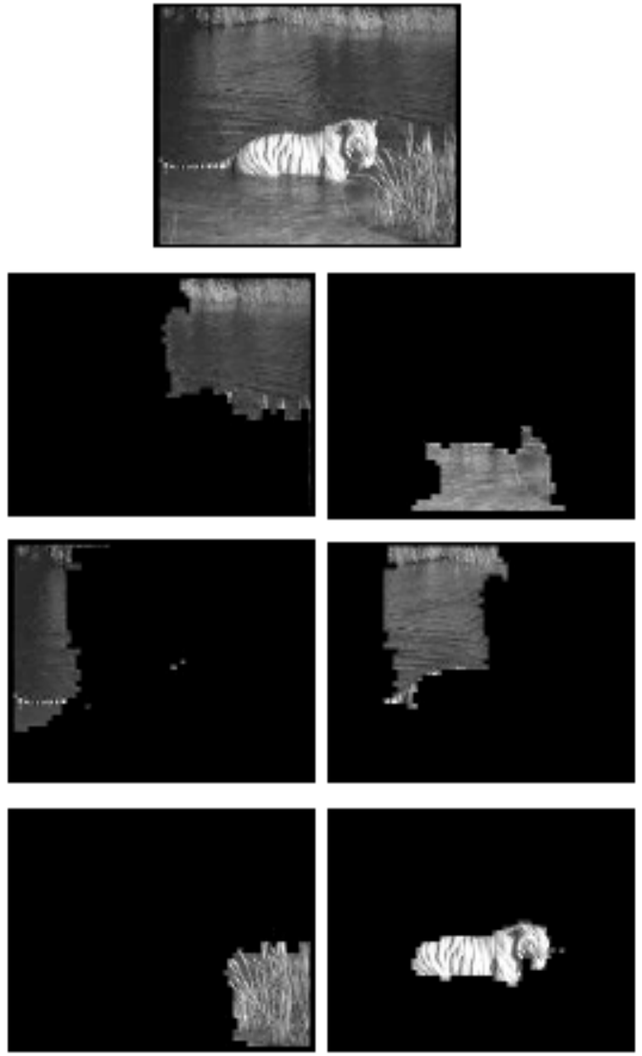
Results



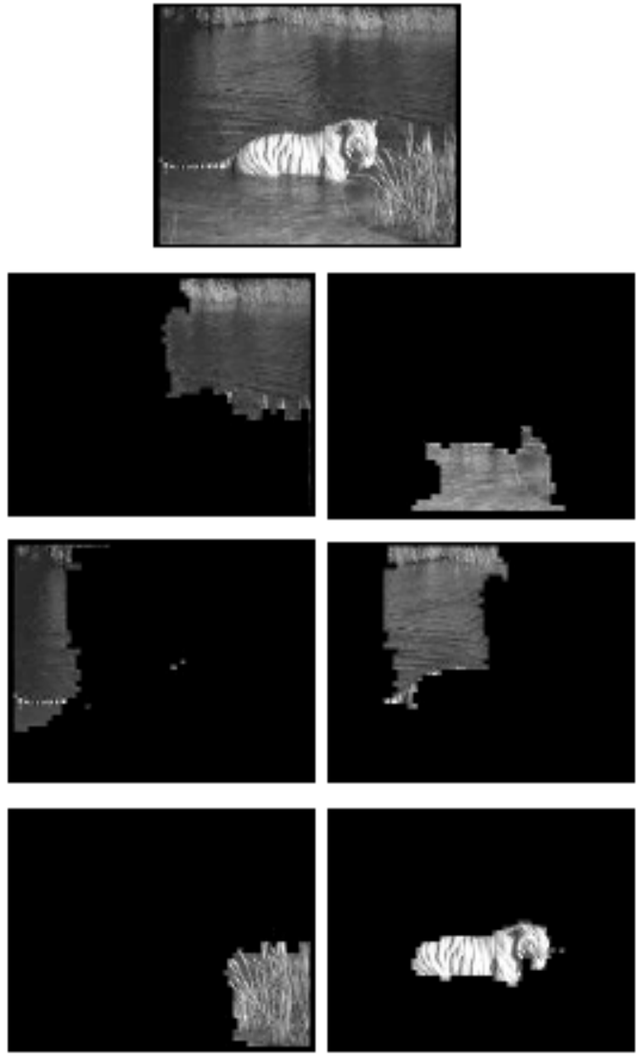
Eigenvectors

Segments

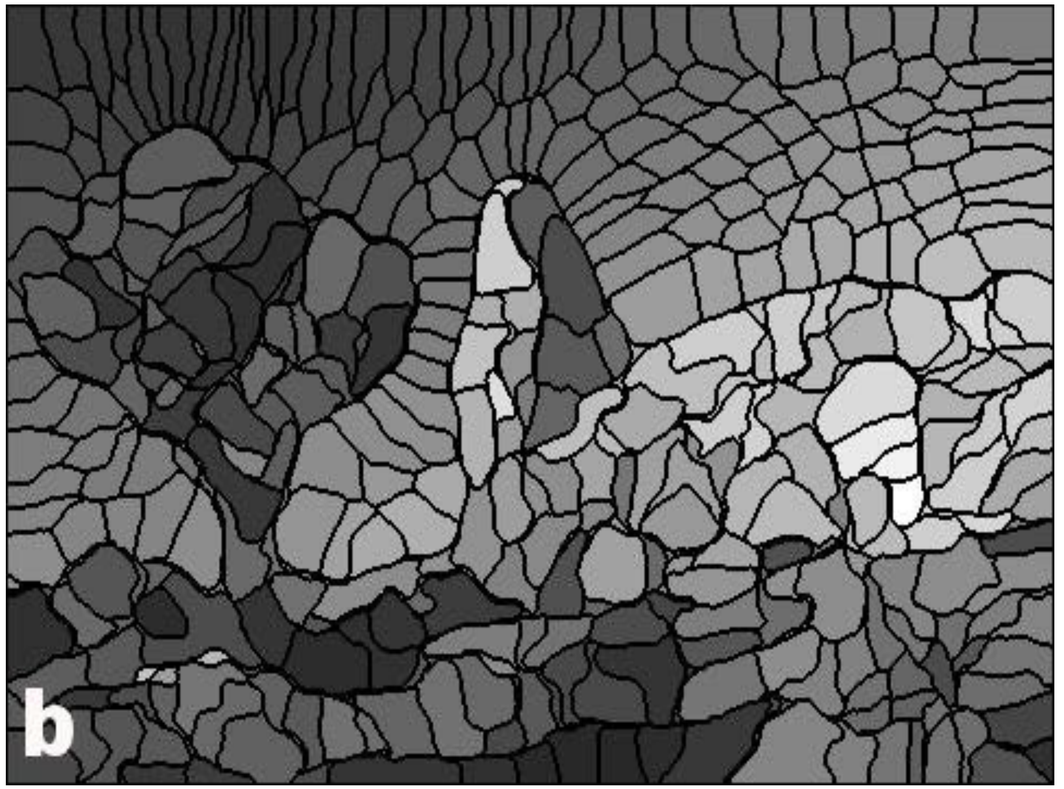
Sample Result



Sample Result



Over-segmentation Results



Create superpixels by recursively performing normalized cuts

Analyzing Normalized Cuts

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Analyzing Normalized Cuts

- State of the art results

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Analyzing Normalized Cuts

- State of the art results
- Huge storage requirement and time complexity

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Analyzing Normalized Cuts

- State of the art results
- Huge storage requirement and time complexity
- Bias towards partitioning into equal segments

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Analyzing Normalized Cuts

- State of the art results
- Huge storage requirement and time complexity
- Bias towards partitioning into equal segments
- Has problems with textured backgrounds

* Slide from Khurram Hassan-Shafique CAP5415 Computer Vision 2003

Split and Merge Algorithms

- **Recursively divide the image into homogeneous regions**
- **Merge adjacent regions that combine to a homogeneous region.**

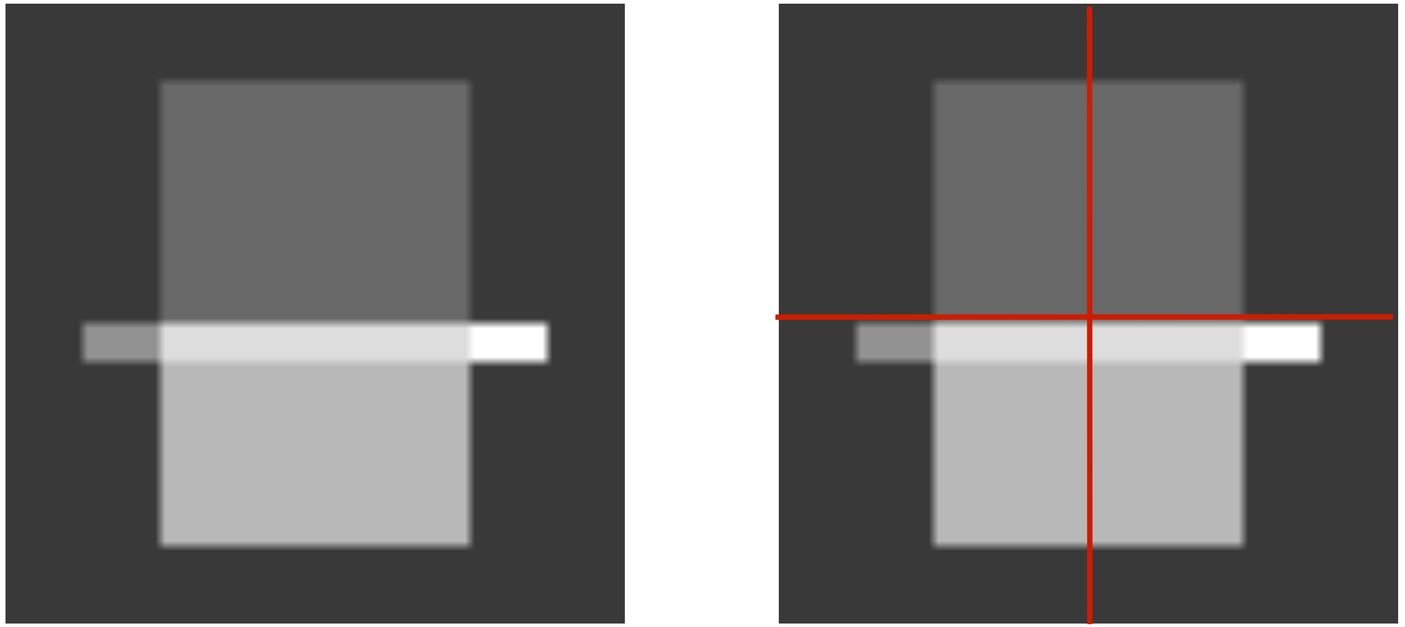
$$\frac{1}{N-1} \sum_{(r,c) \in Region} [I(r,c) - \overline{I}]^2$$

Splitting

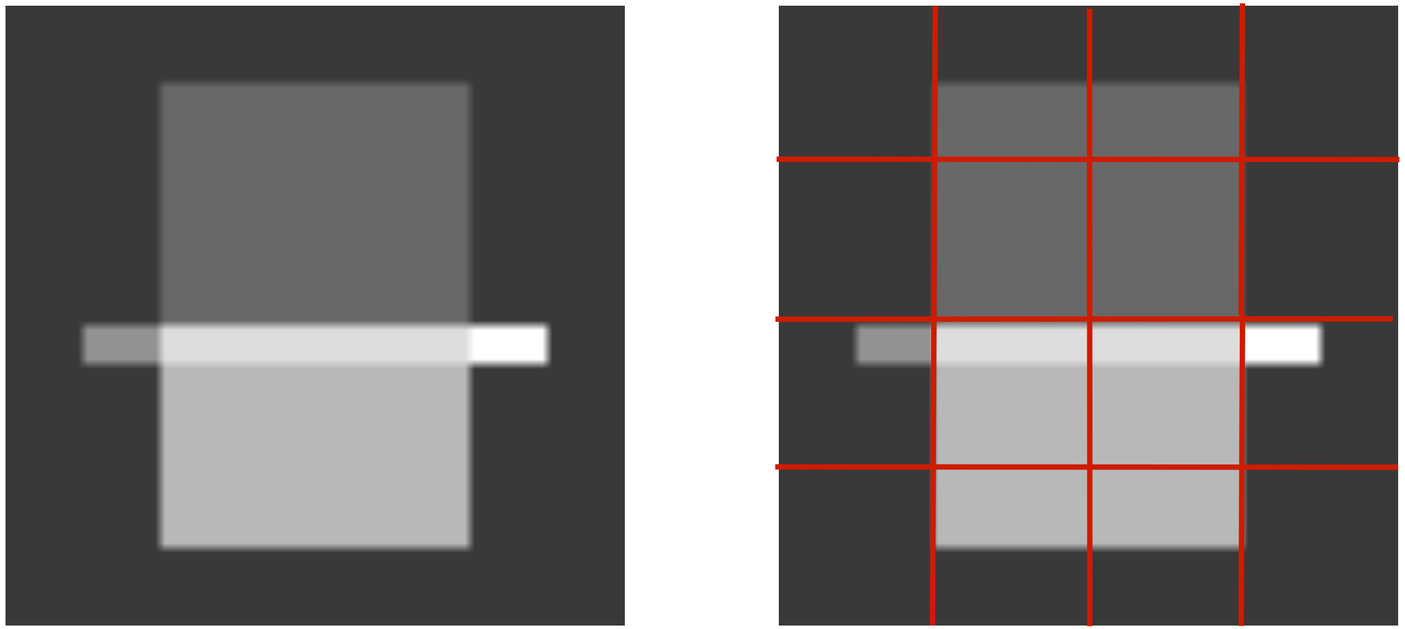
```
% Splits an image I into homogeneous regions
% labelled in L.
function L = ImageSplit(I)
% Initialize.
[X,Y] = size(I);
L = ones(X,Y);
n = 2;
% Call recursive splitting subroutine
L = split(I, L, 1, X, 1, Y, n)
```

```
function L = split(I, L, xmin, xmax, ymin, ymax, n)
if(~homogeneous(I(xmin:xmax, ymin:ymax)))
    xSplit = (xmin+xmax)/2;  ySplit = (ymin+ymax)/2;
    % Top left
    L = split(I, L, xmin, xSplit, ymin, ySplit, n);
    % Top right
    L((xSplit+1):xmax, ymin:ySplit) = n;
    n = n + 1;
    L = split(I, L, (xSplit+1), xmax, ymin, ySplit, n);
    % Bottom left
    L(xmin:xSplit, (ySplit+1):ymax) = n;
    n = n + 1;
    L = split(I, L, xmin, xSplit, (ySplit+1), ymax, n);
    % Bottom right… similar…
end
```

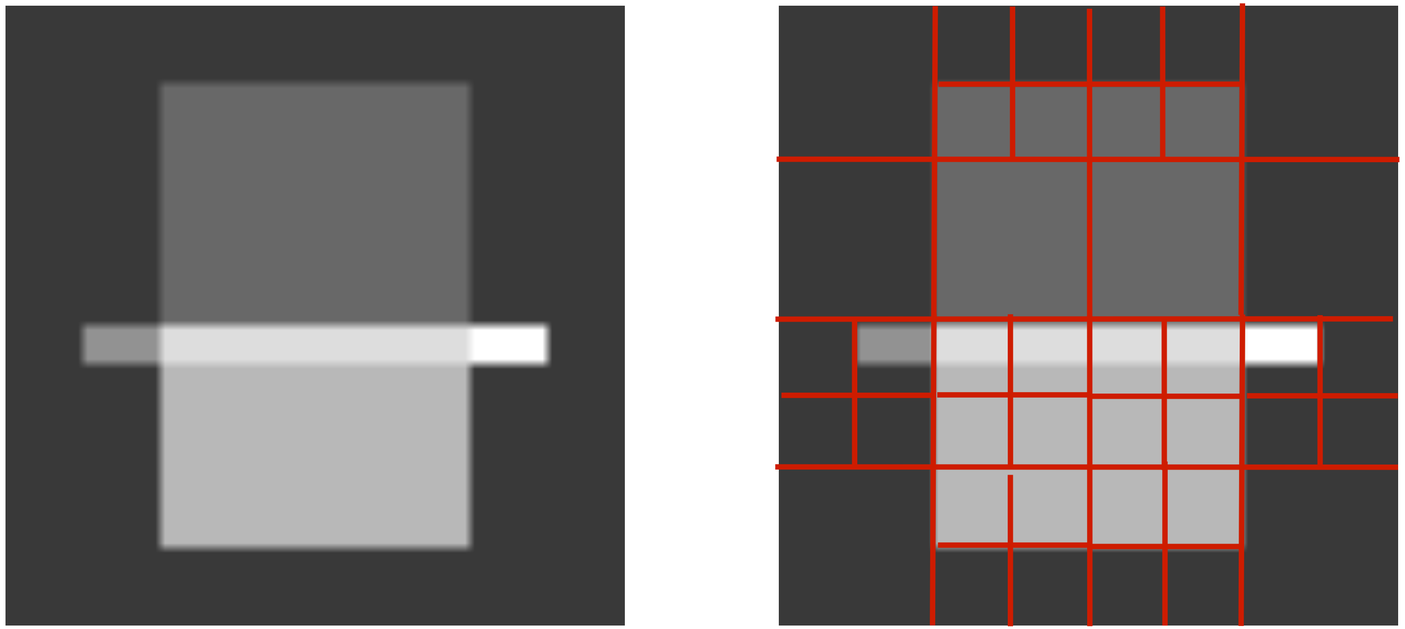
Splitting



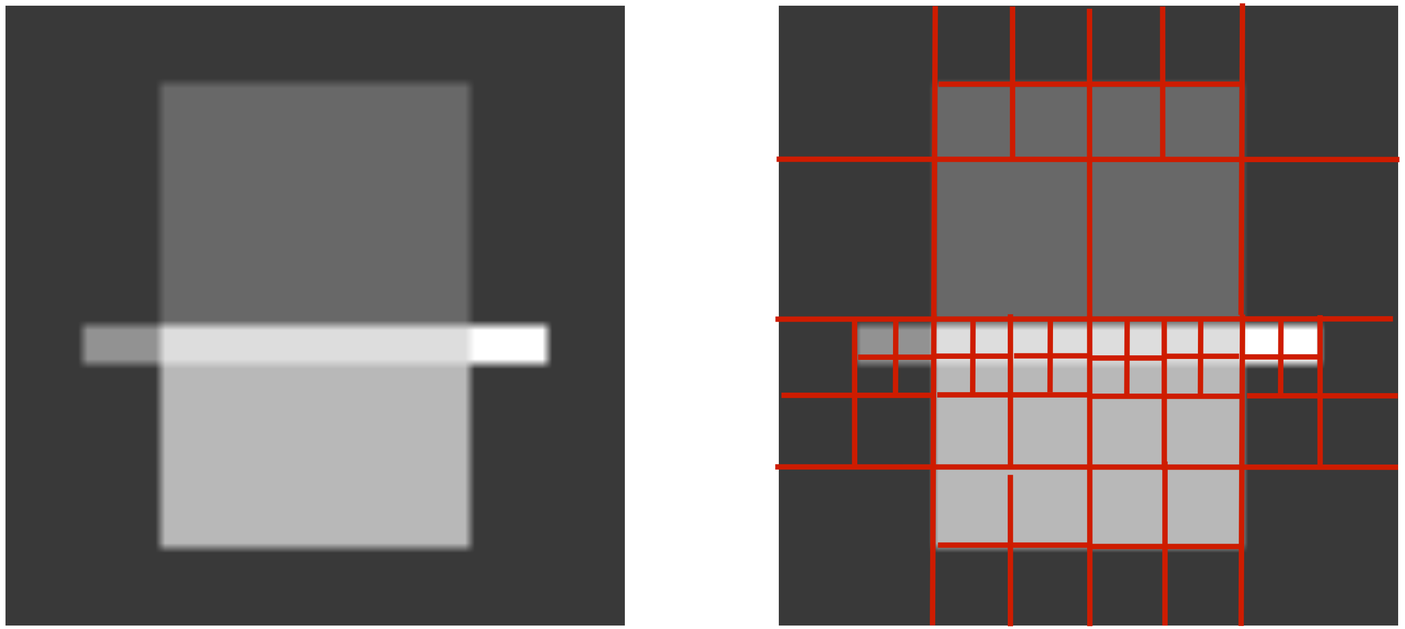
Splitting



Splitting



Splitting



Merging

Merging

- Build the *Region Adjacency Graph* (RAG)
  - Each labelled region of the image is a graph node
  - The nodes for adjacent regions have an edge between them

Merging

- **Build the *Region Adjacency Graph* (RAG)**
  - **Each labelled region of the image is a graph node**
  - **The nodes for adjacent regions have an edge between them**

- **Merge adjacent regions that are homogeneous**

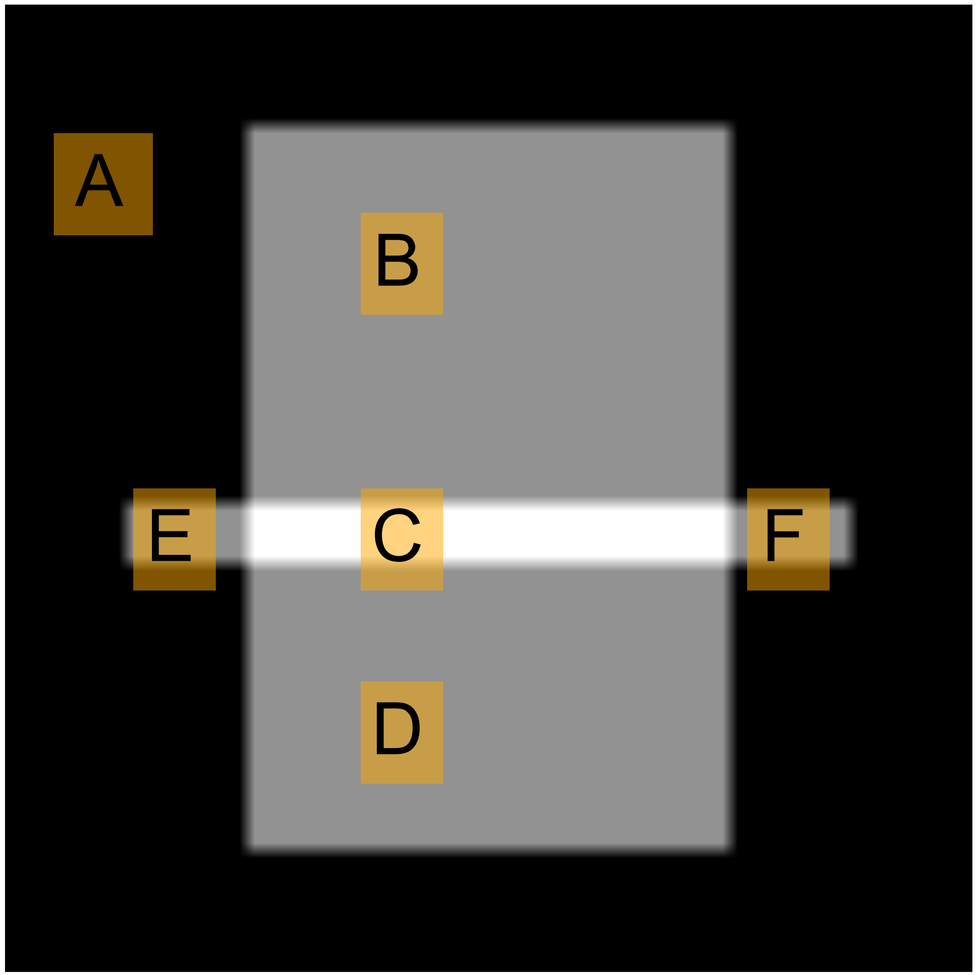
Merging

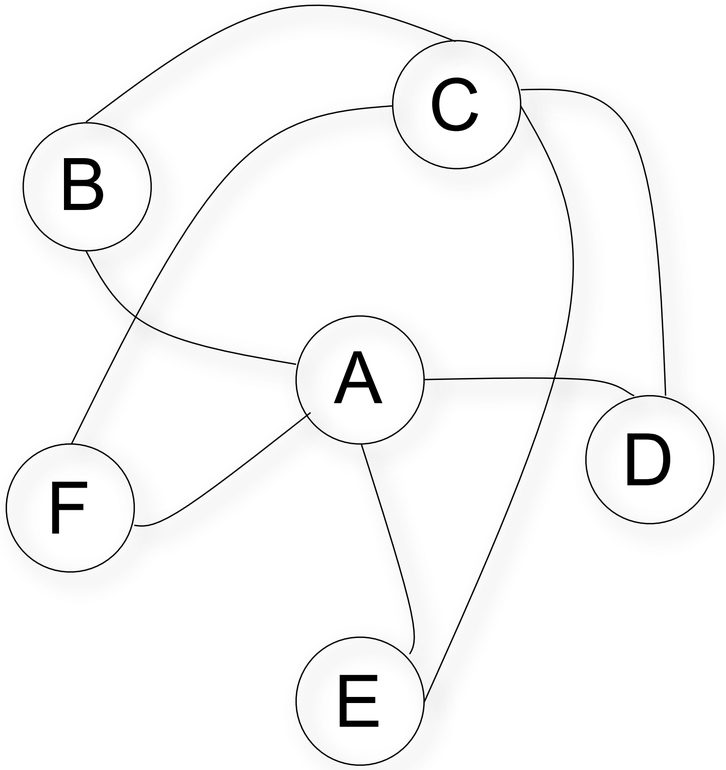
- Build the *Region Adjacency Graph* (RAG)
  - Each labelled region of the image is a graph node
  - The nodes for adjacent regions have an edge between them
- Merge adjacent regions that are homogeneous
- Update RAG

Merging

- Build the *Region Adjacency Graph* (RAG)
  - Each labelled region of the image is a graph node
  - The nodes for adjacent regions have an edge between them
- Merge adjacent regions that are homogeneous
- Update RAG
- Repeat to convergence

RAG Example

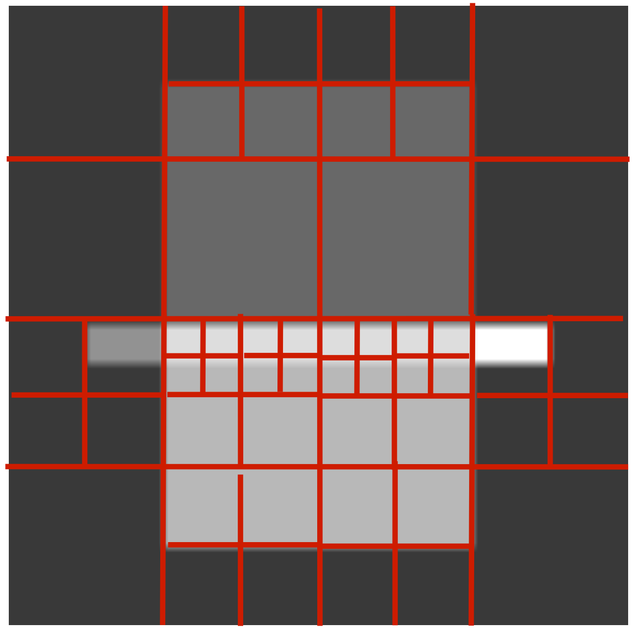




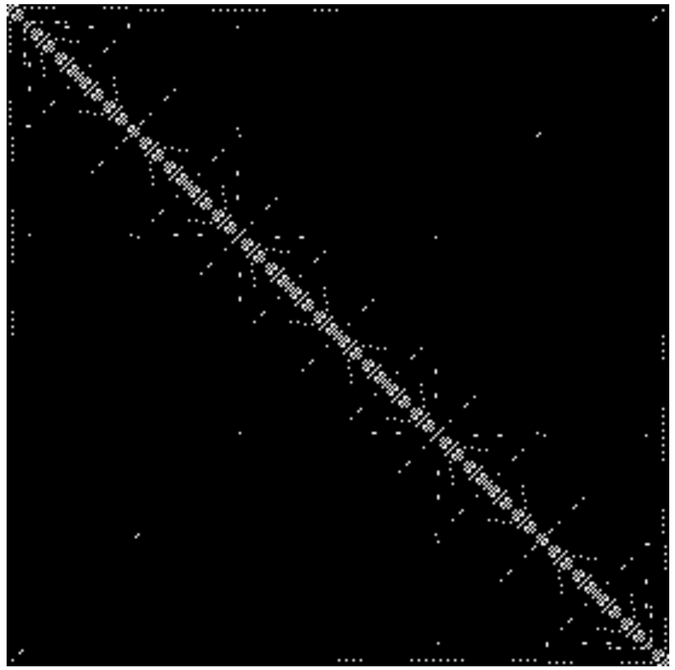
Computing the RAG

```
function rag = RegionAdjacencyGraph(L)
numRegions = max(max(L));  [X,Y] = size(L);
rag = zeros(numRegions, numRegions);
For each (x1, y1) in L
    r1 = L(x1, y1);
    For each (x2, y2) in N(x1, y1)
        r2 = L(x2, y2);
        if(r1 ~= r2)
            rag(r1, r2) = rag(r2, r1) = 1;
        end
    end
end
```

RAG Example



274 regions



Region Merging

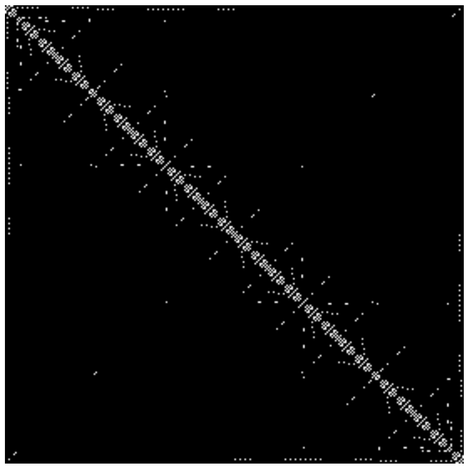
```
numRegions = max(max(L));
done = 0;
while(~done)
    done = 1;
    for i=1:(numRegions-1)
        for j=(i+1):numRegions
            if(rag(i,j))
                combinedR = I(find(L == i | L == j));
                if(homogeneous(combinedR))
                    [L, rag] = merge(i, j, rag, L);
                    done = 0;
                end
… end
```

Region Merging

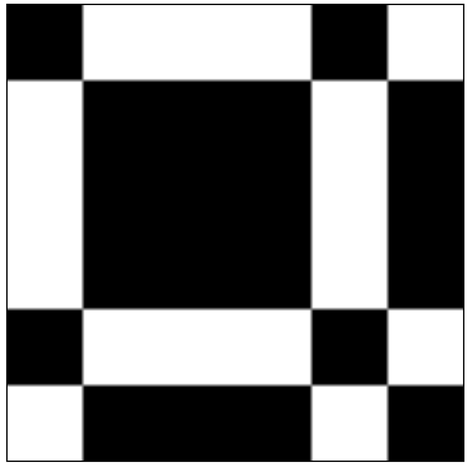
```
function [L, rag] = merge(label1, label2, rag, L)
% Merge the regions in the labelled image.
L(find(L == label2)) = label1;
% Combine the adjacency of the two regions.
rag(label1,:) = rag(label1,:) | rag(label2,:);
rag(:,label1) = rag(:,label1) | rag(:,label2);
% Make sure the combined region1 is not self-adjacent.
rag(label1, label1) = 0;
% Remove all adjacency to region2
rag(label2,:) = 0;
rag(:,label2) = 0;
```

Merging Example - Before/After

Image



RAG before merging



RAG after merging

Split and Merge Example

Split and merge example

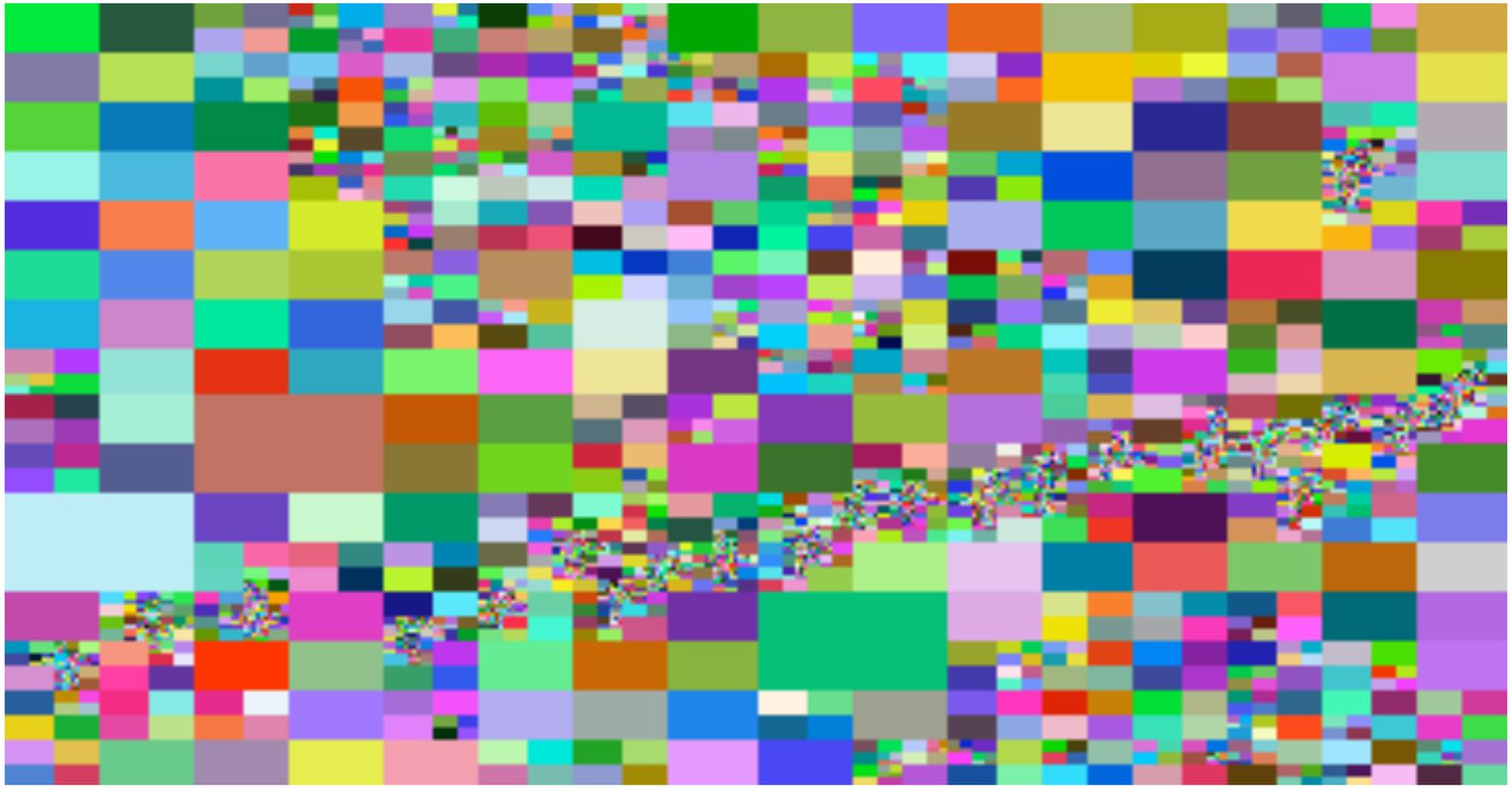
- Regions are homogeneous if

  Max. greylevel – min. greylevel < T.

- T = 20 for split.

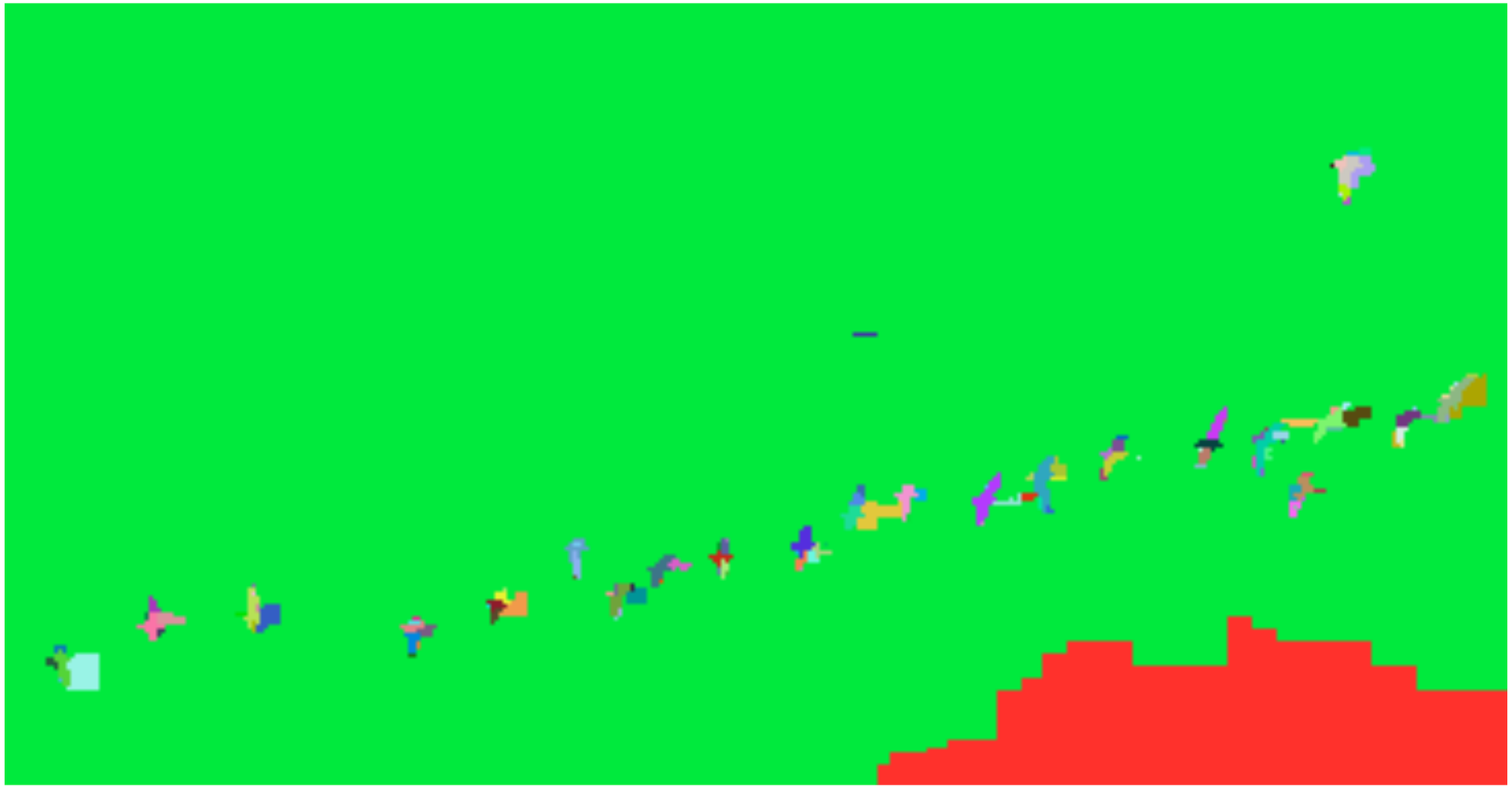
- T = 100 for merge.

Split and merge example



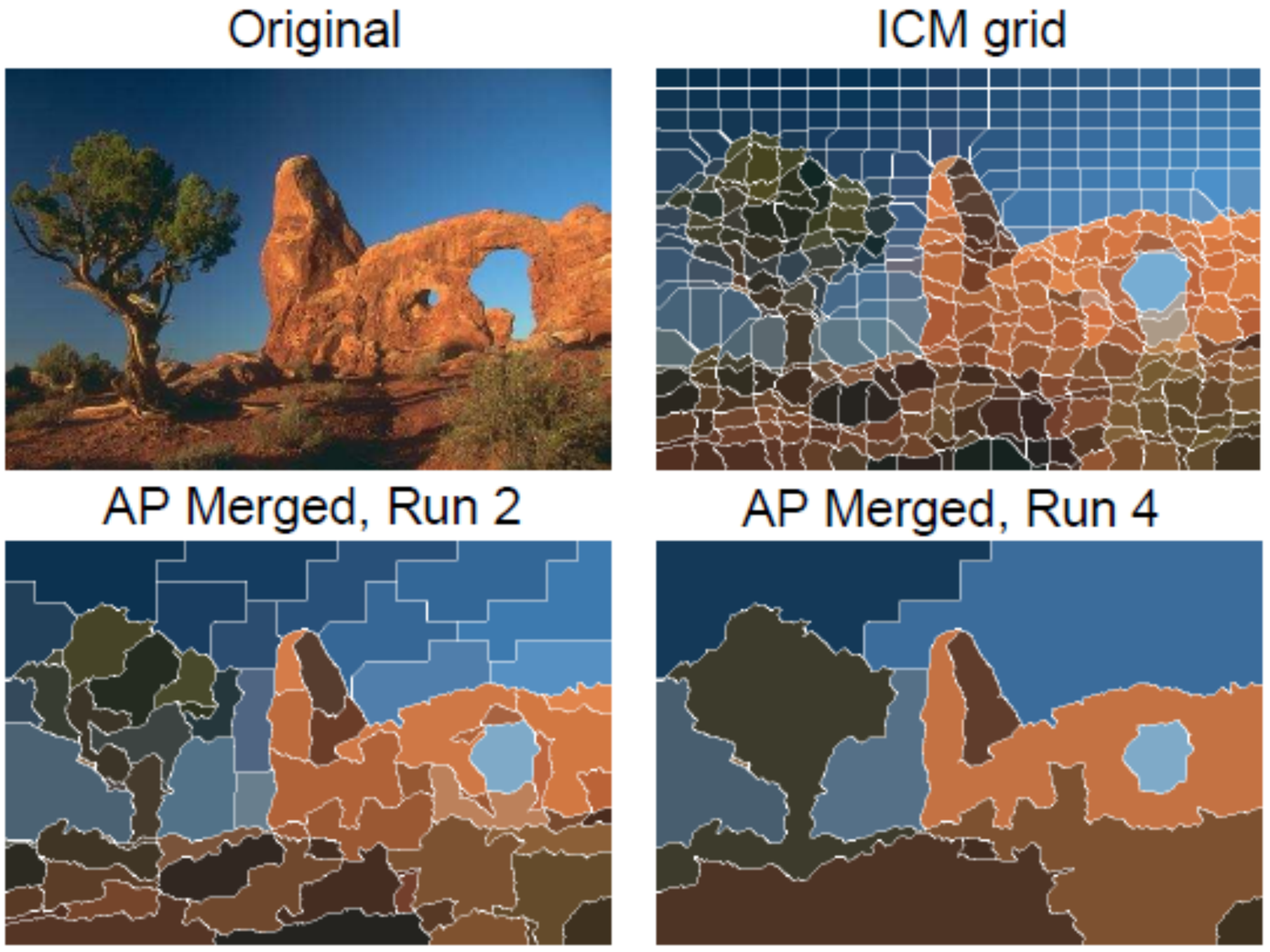
4087 regions

Split and merge example



135 regions

Split/merge Algorithm Superpixel Lattices (Moore et al. 2008)



Superpixel Lattices (Moore et al. 2008)