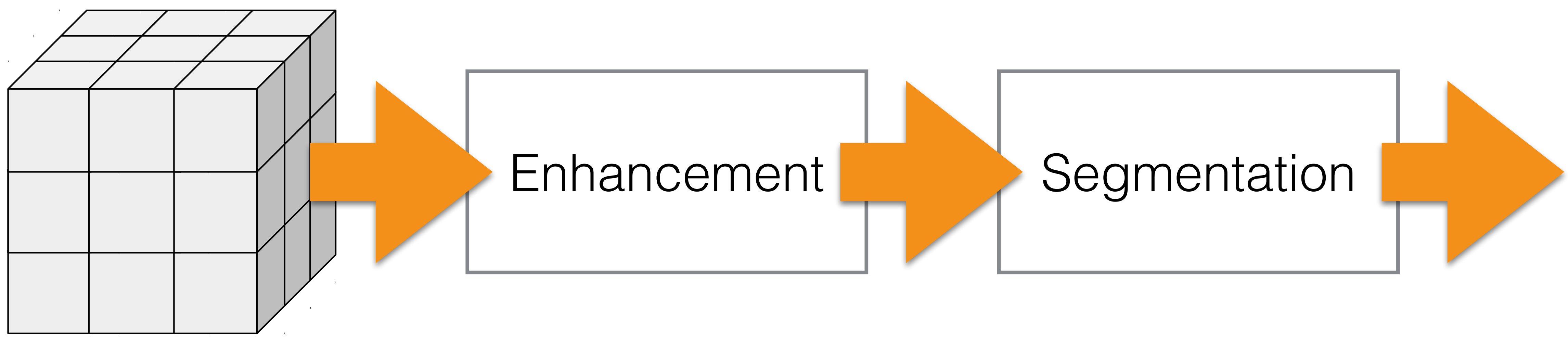
3D from Volume: Part II

Francesco Banterle, Ph.D.
francesco.banterle@isti.cnr.it

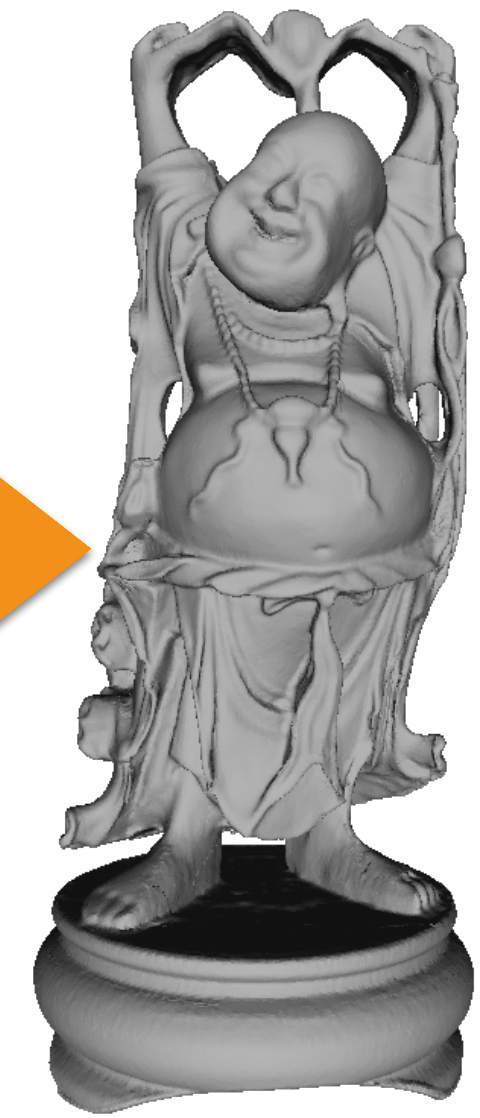
The Processing Pipeline



RAW Volume

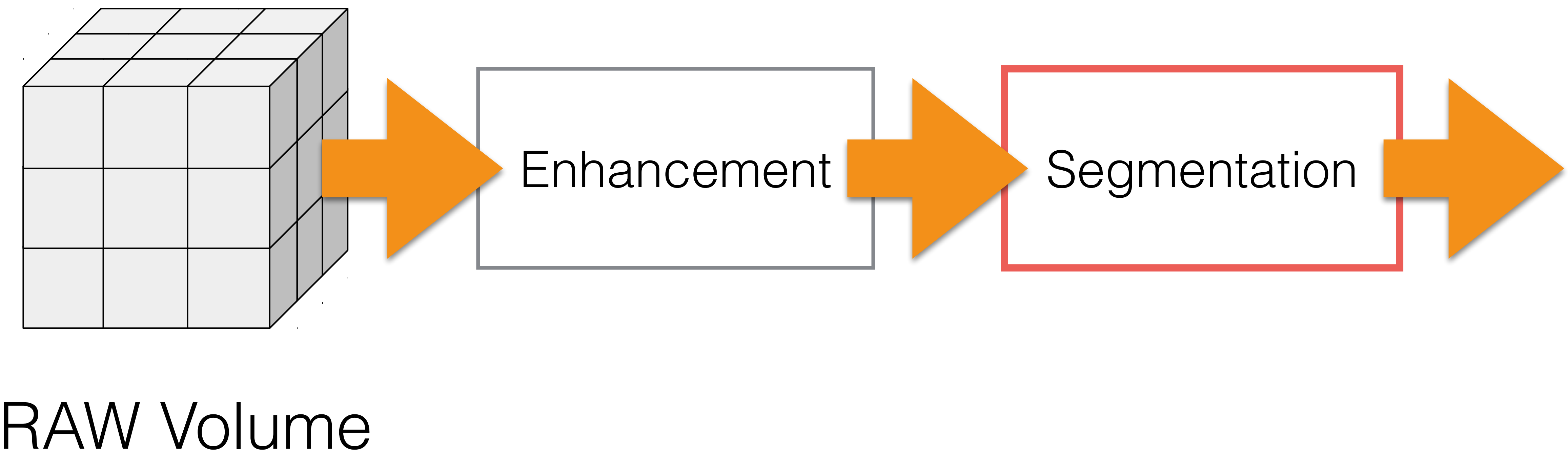
The Processing Pipeline

Points Extractions → Mesh Extraction →



3D Mesh

The Processing Pipeline

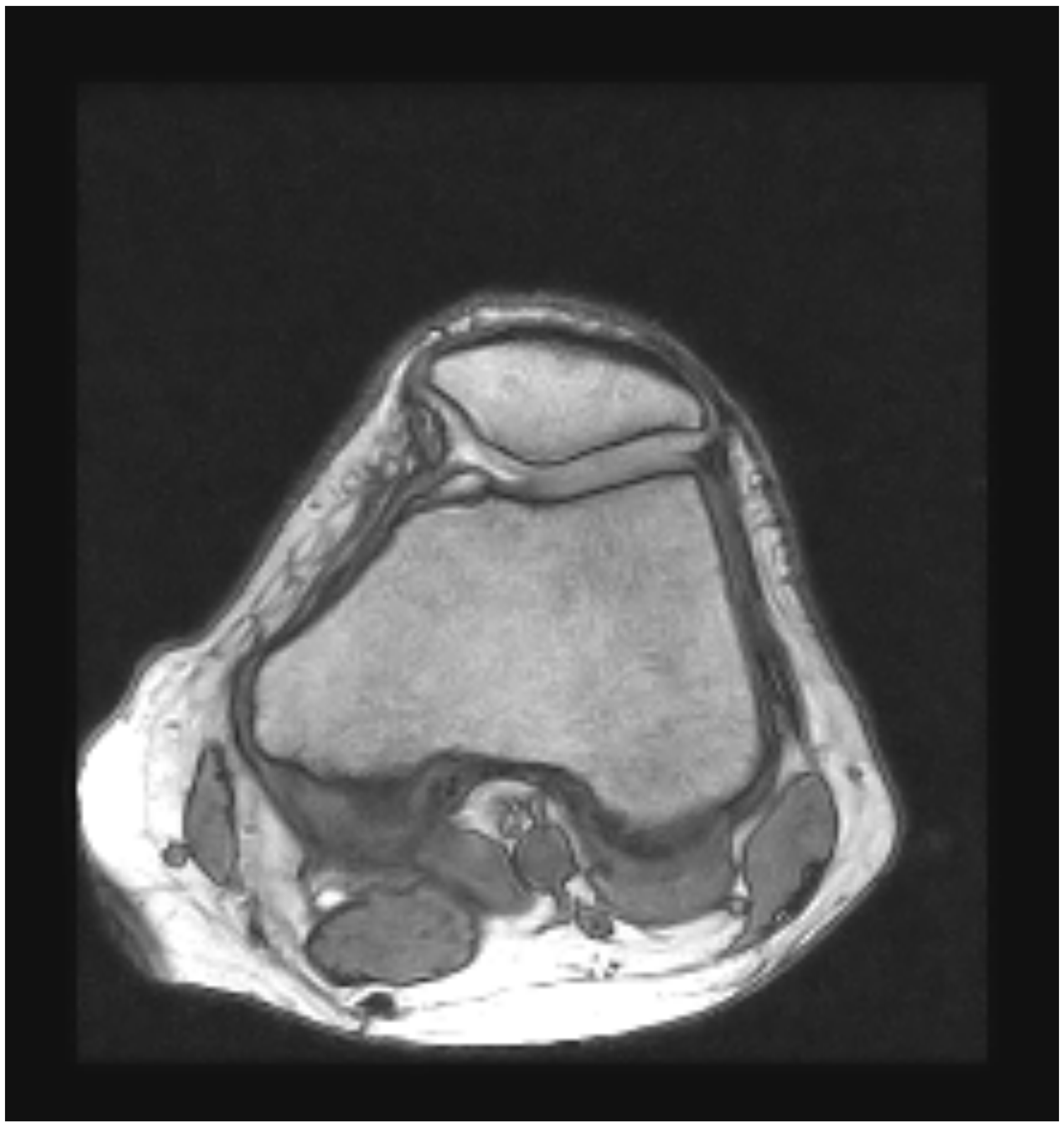


2D/3D Segmentation

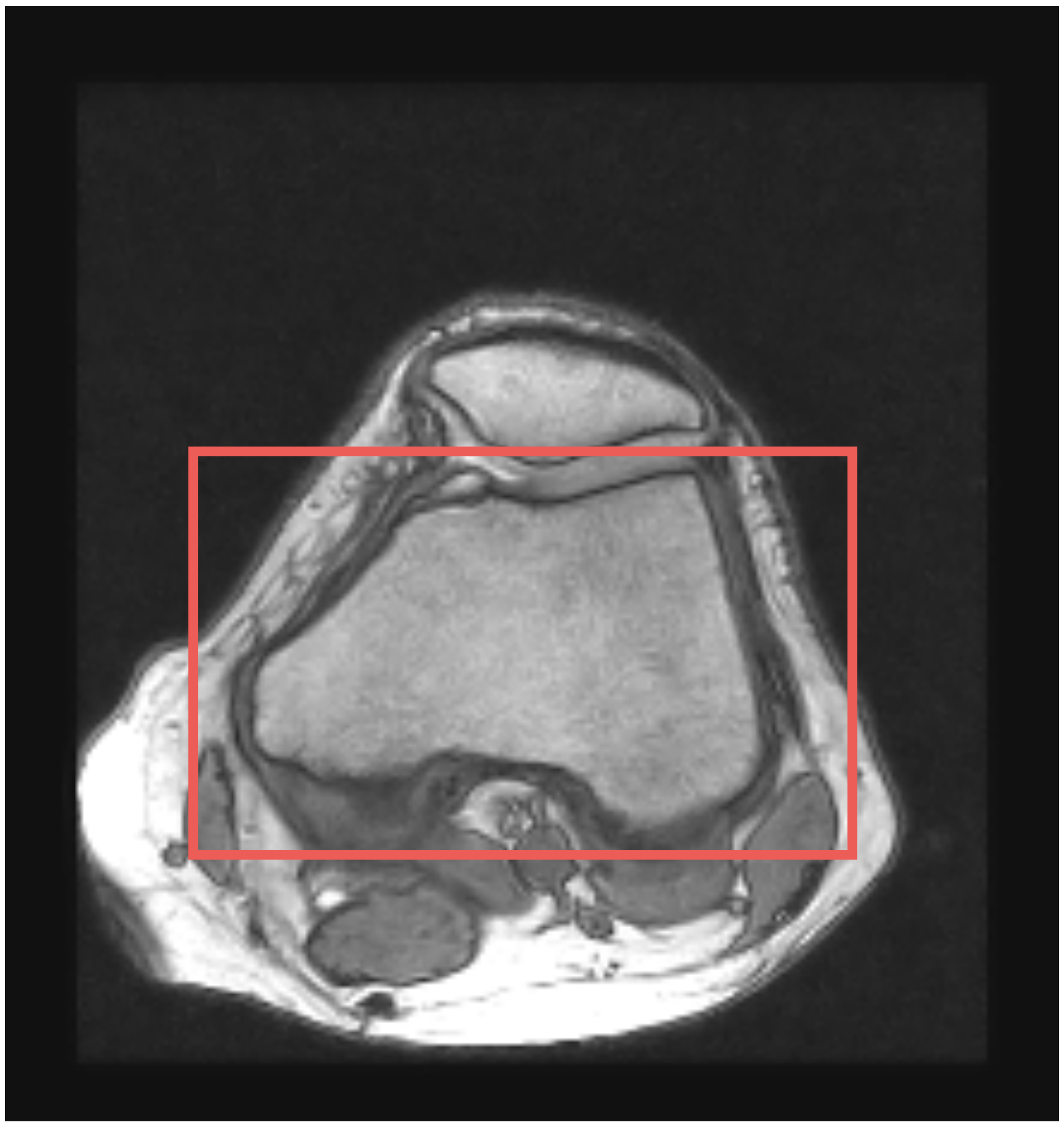
Segmentation

- Segmentation is a process after which we obtain a mask of a structure in an/a image/volume.
- A mask is binary image/volume; i.e., its values can be only either 0 or 1.
- 1 —> the pixel/voxel belongs to a structure of our interest
- 0 —> the pixel/voxel does not!

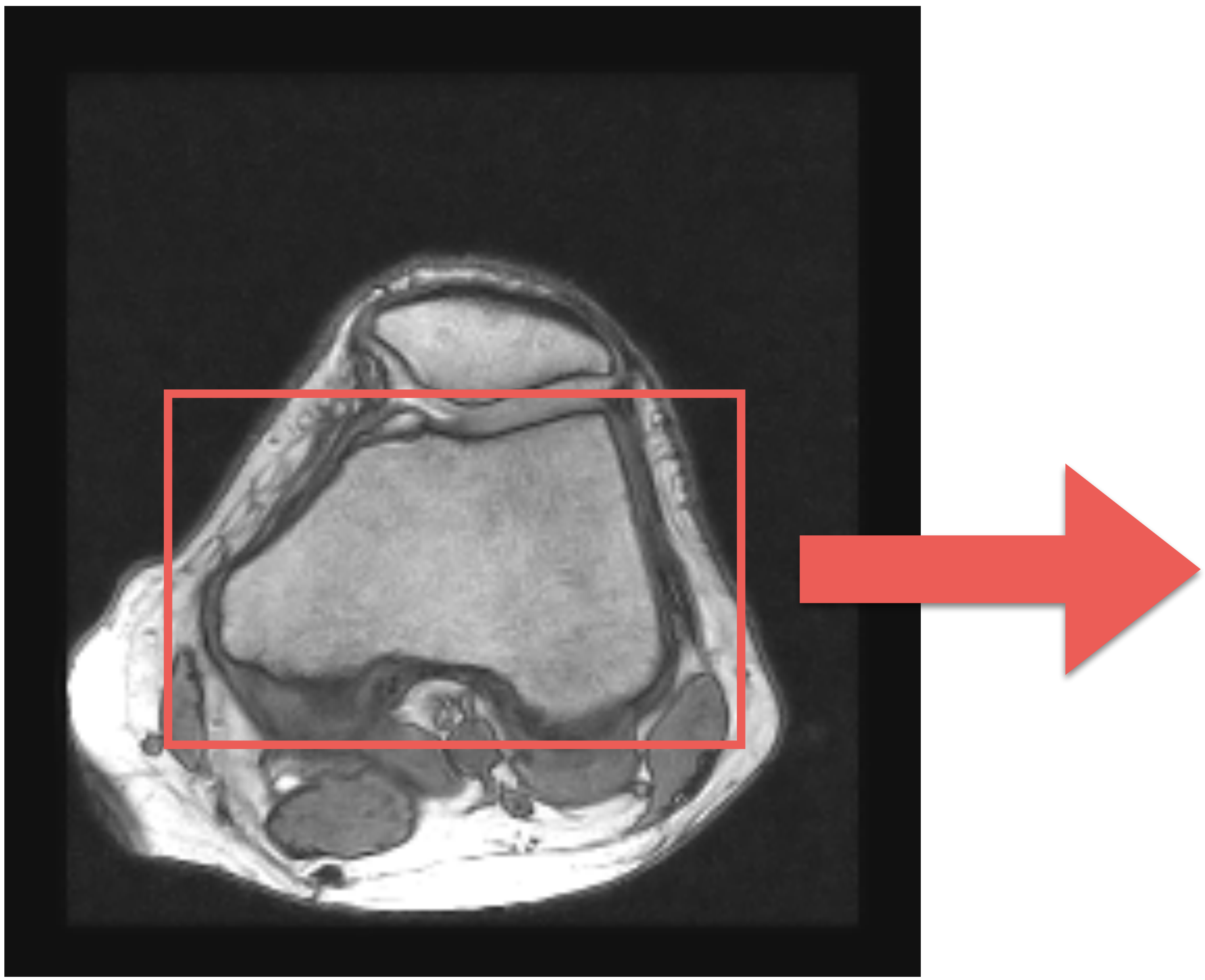
Segmentation Example



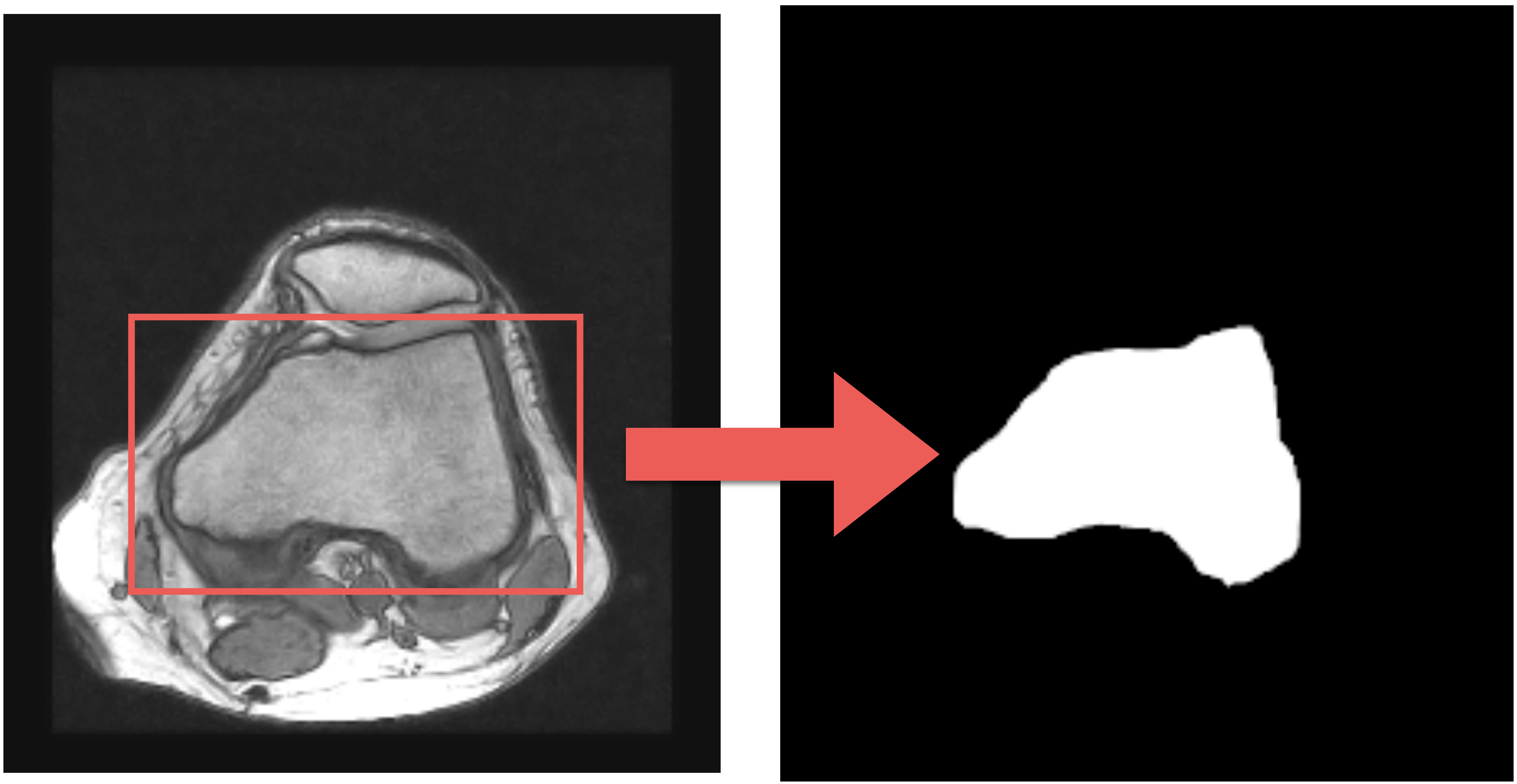
Segmentation Example



Segmentation Example



Segmentation Example



Segmentation

- Obviously, if we need to segment k objects in the image/volume we have two ways to proceed:

  1. We create k-masks, one for each object.

  2. We create an **unsigned integer** mask in which each object as label a number in $[1, k]$. Background is always 0!

3D Segmentation

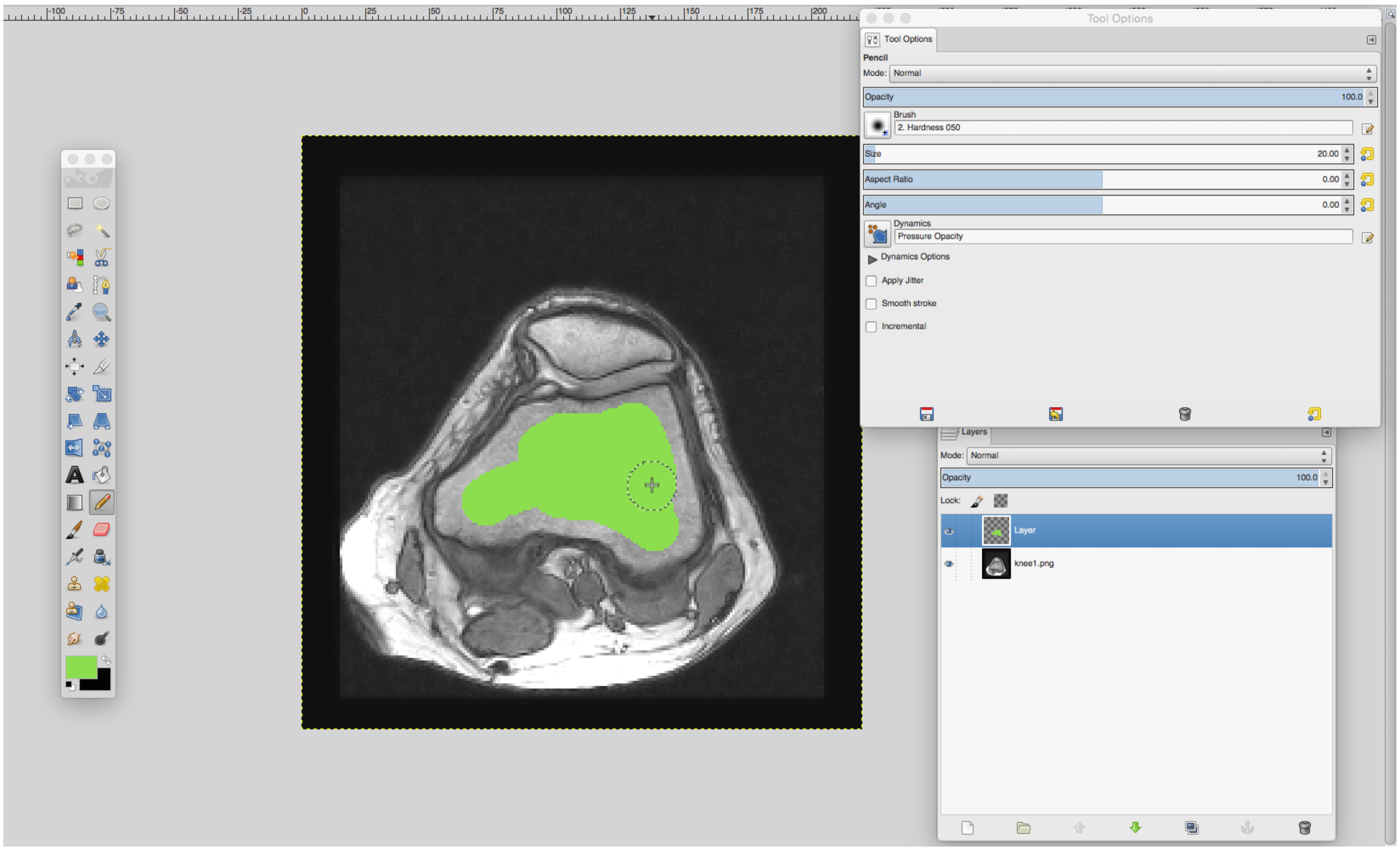
- There are typically two approaches:
  - 2D segmentation for each slice
  - 2D segmentation of a slice and propagation of the segmentation

Manual Segmentation

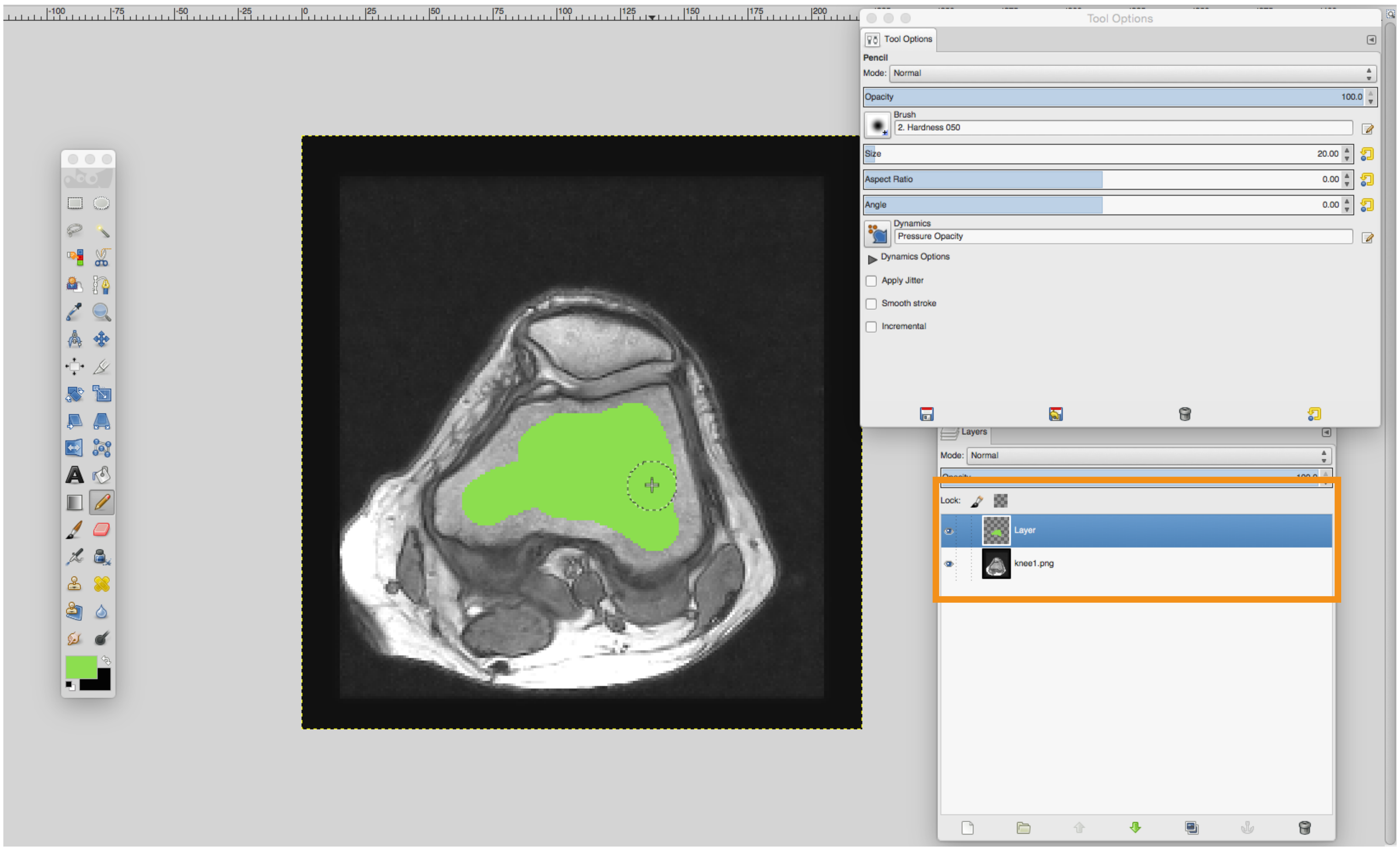
Manual Segmentation: Painting Approach

- We manually paint the mask using a GUI.
- Obviously, the segmentation mask is created in a different layer and not on the input image!

Manual Segmentation: Painting Approach



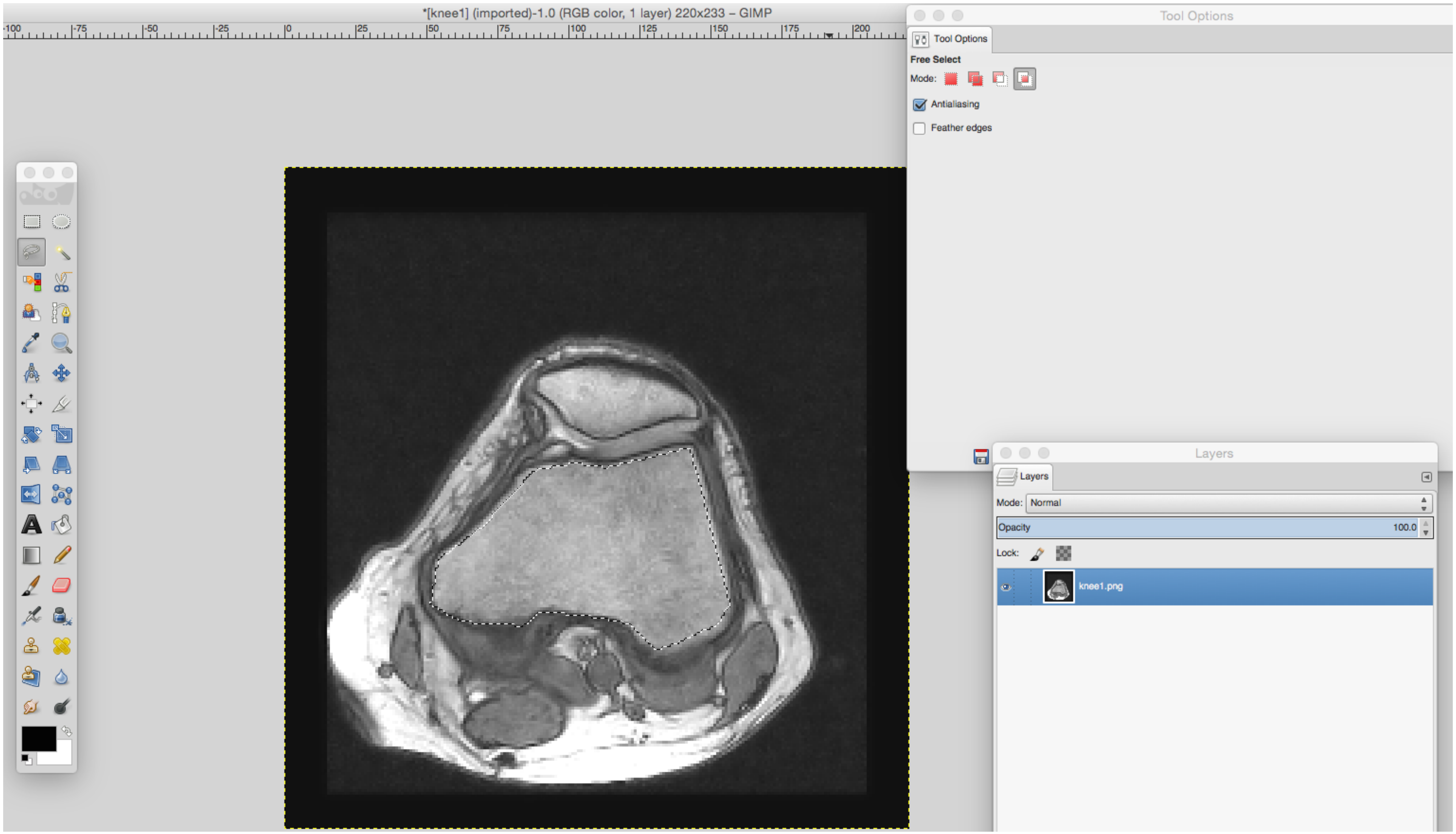
Manual Segmentation: Painting Approach



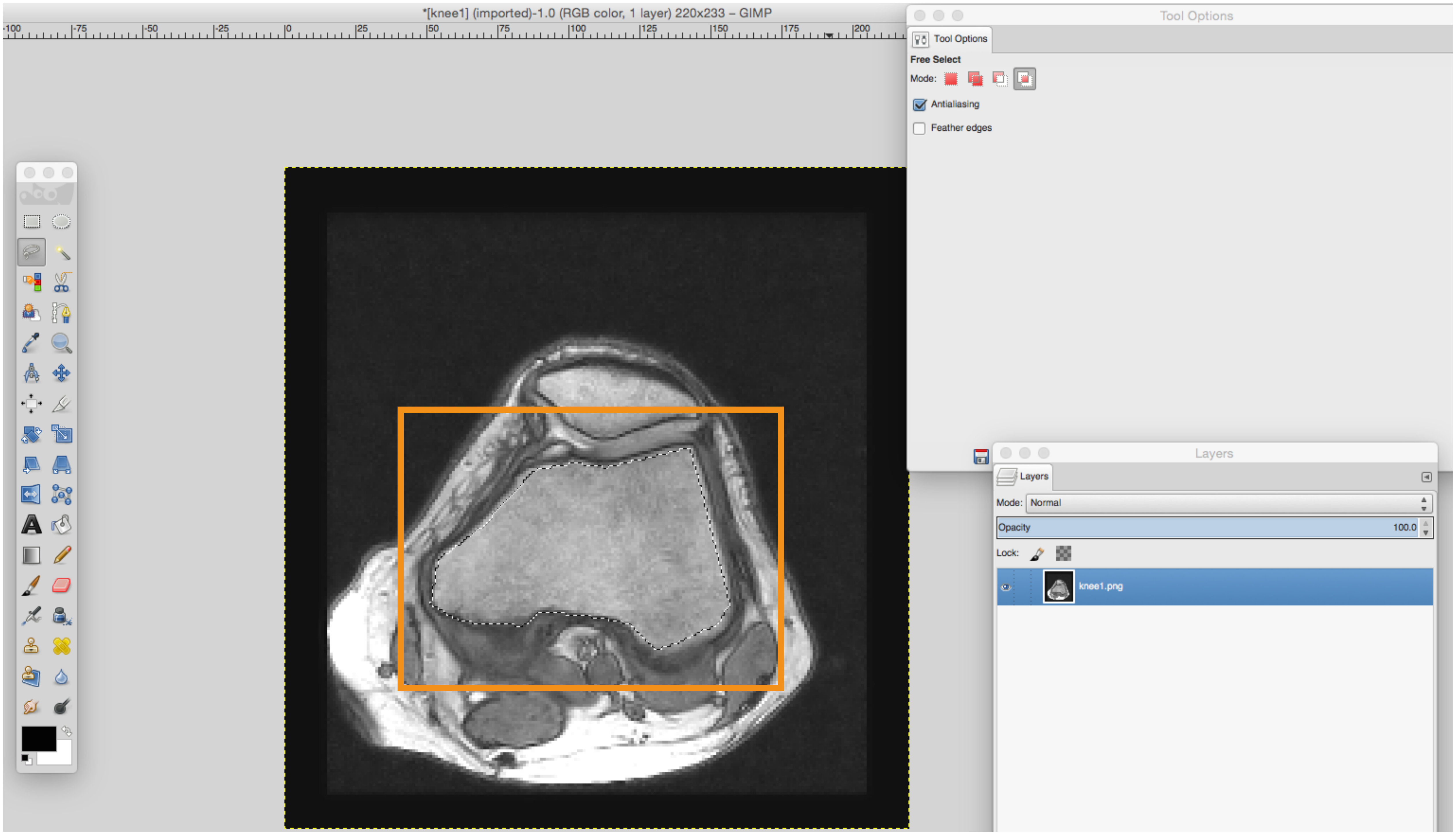
Manual Segmentation: Boundary Definition

- We manually define the mask boundary using a GUI (e.g., GIMP, Adobe PhotoShop, etc.).
- We either define it using polygons or free-hand.
- We can use image gradients and Laplacian to stick polygons to our object of interest.

Manual Segmentation: Boundary Definition



Manual Segmentation: Boundary Definition



Thresholding

Thresholding Example

- We assume that each object in an image/volume has a unique intensity value

Object	Value
Skull	255
Grey Matter	153
Veins	77

Thresholding

- This means:

$$M(i,j) = \begin{cases} 1 & \text{if } d(I(i,j), I_t) < t \\ 0 & \text{otherwise} \end{cases}$$

- We can have different distance functions:

$$d(x,y) = |x - y|$$

$$d(x,y) = (x-y)^2$$

$$d(x,y,\sigma) = \exp\left(-\frac{(x-y)^2}{2\sigma^2}\right)^2$$

Thresholding

- This means:

Reference Value

$$M(i,j) = \begin{cases} 1 & \text{if } d(I(i,j), \boxed{I_t}) < t \\ 0 & \text{otherwise} \end{cases}$$

- We can have different distance functions:

$$d(x,y) = |x - y|$$

$$d(x,y) = (x-y)^2$$

$$d(x,y,\sigma) = \exp\left(-\frac{(x-y)^2}{2\sigma^2}\right)^2$$

Thresholding

- This means:

Reference Value

$$M(i,j) = \begin{cases} 1 & \text{if } d(I(i,j), \boxed{I_t}) < \boxed{t} \\ 0 & \text{otherwise} \end{cases}$$

Threshold

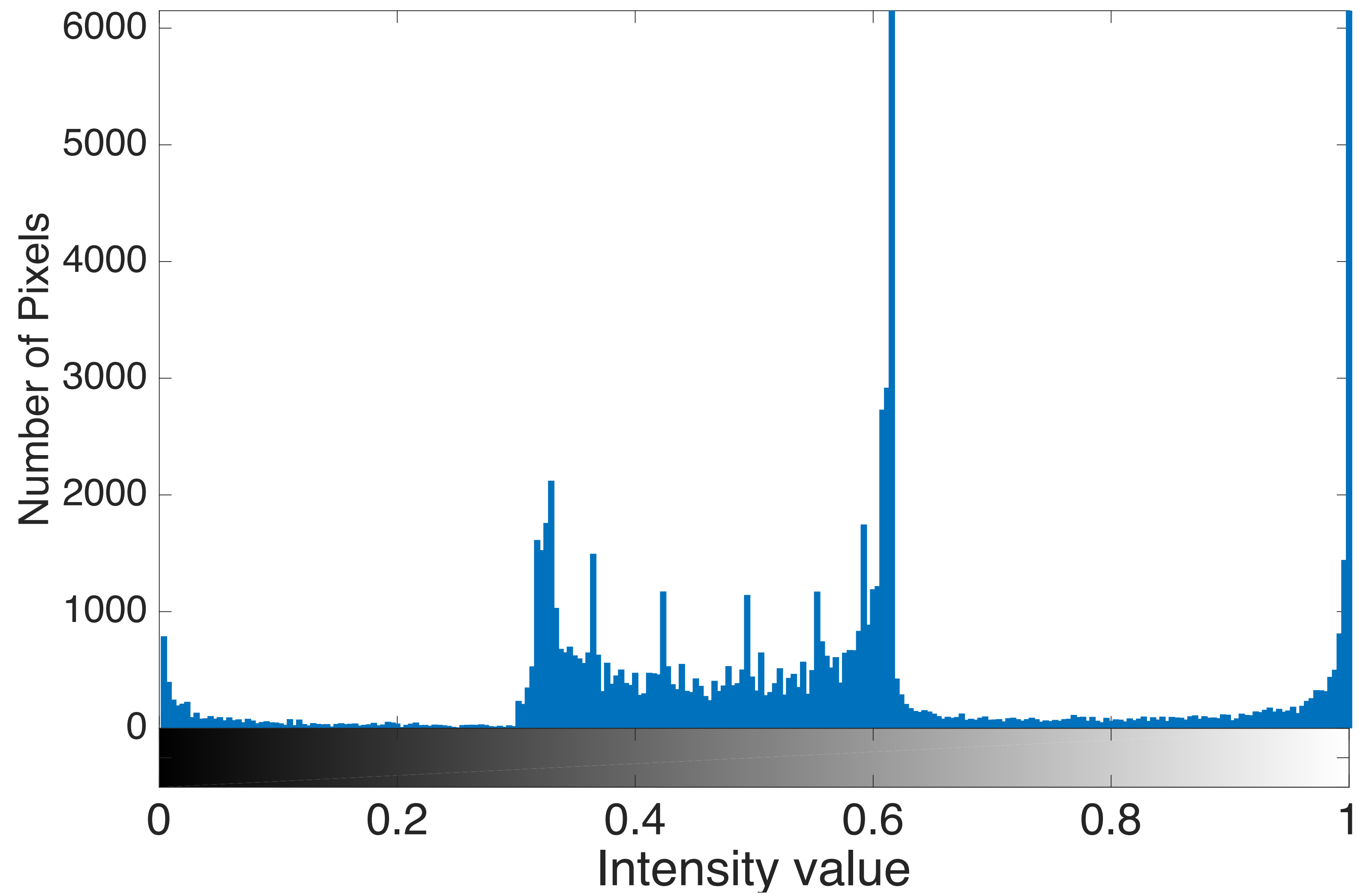
- We can have different distance functions:

$$d(x,y) = |x - y|$$

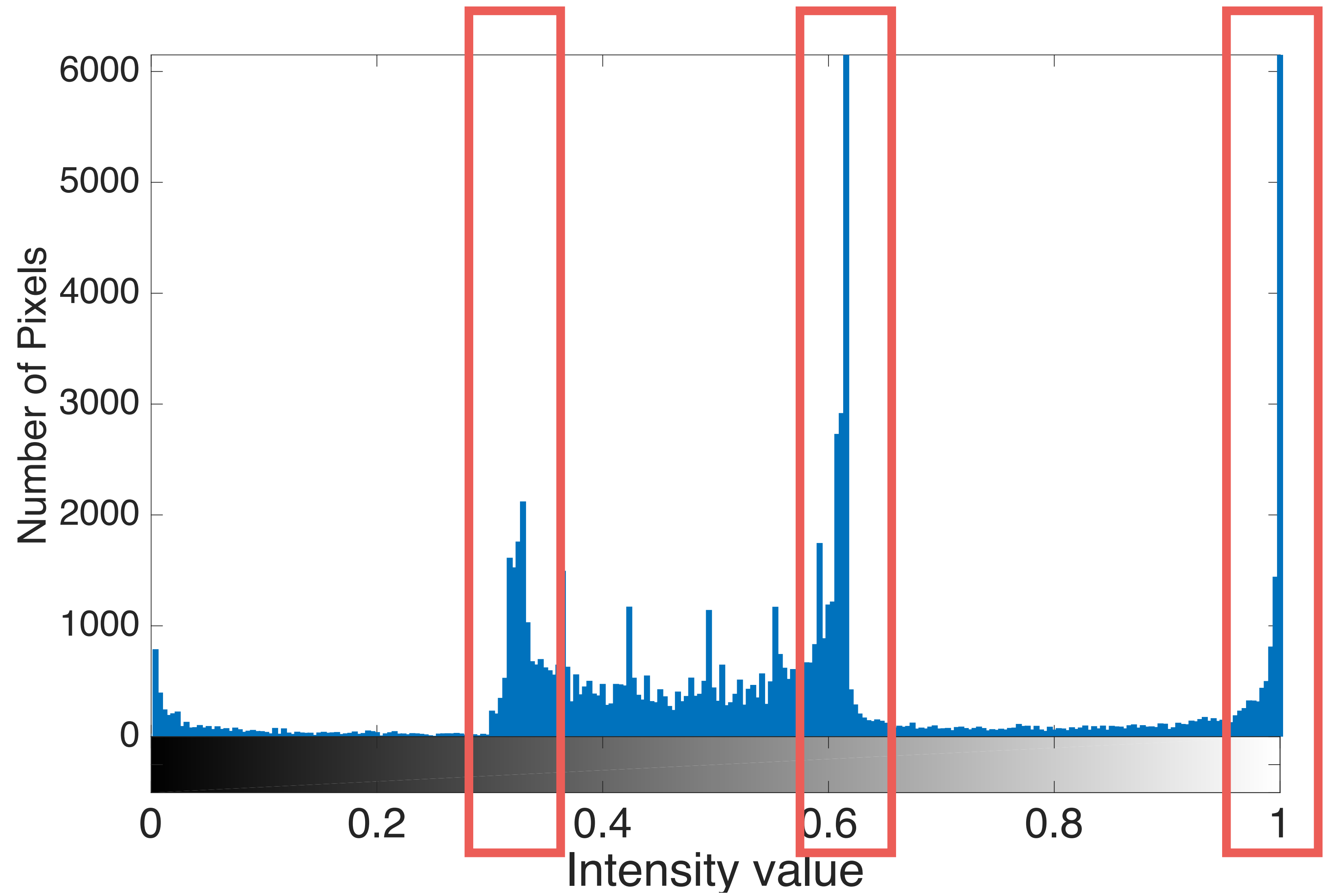
$$d(x,y) = (x - y)^2$$

$$d(x,y,\sigma) = \exp\left(-\frac{(x-y)^2}{2\sigma^2}\right)^2$$

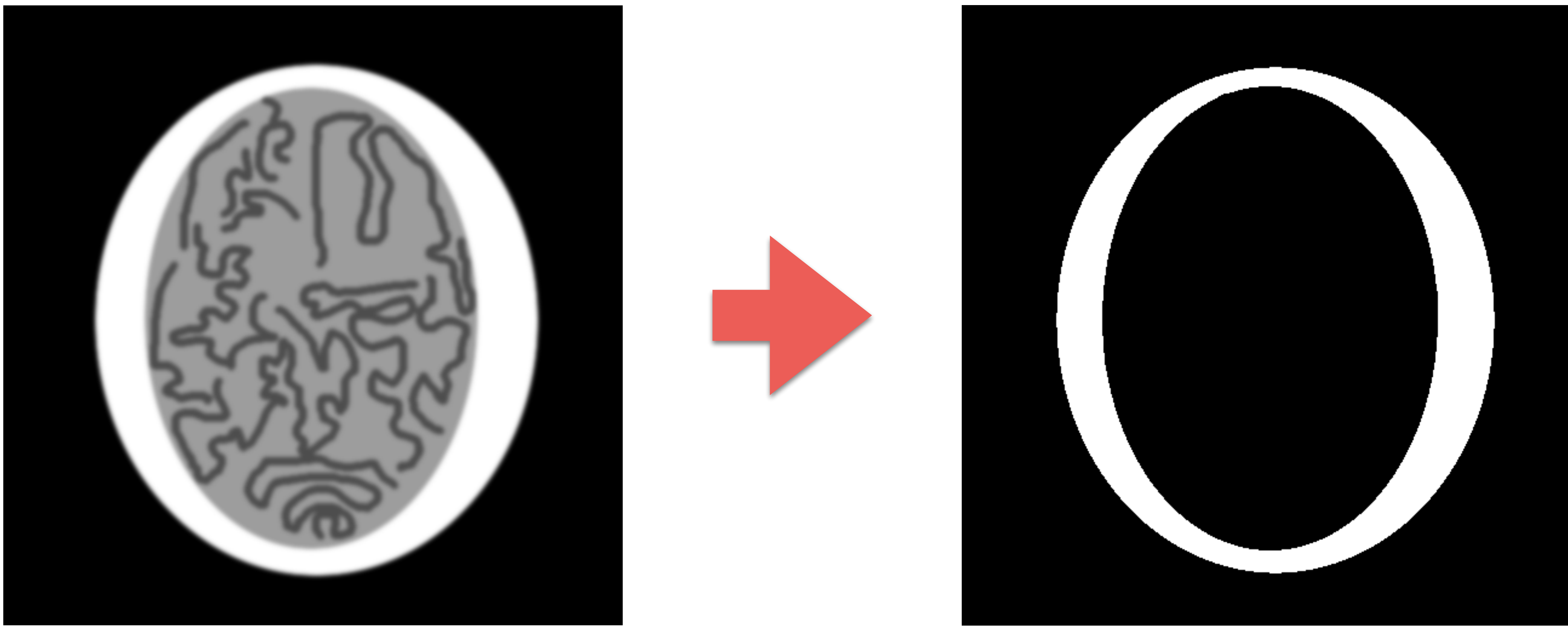
Thresholding



Thresholding

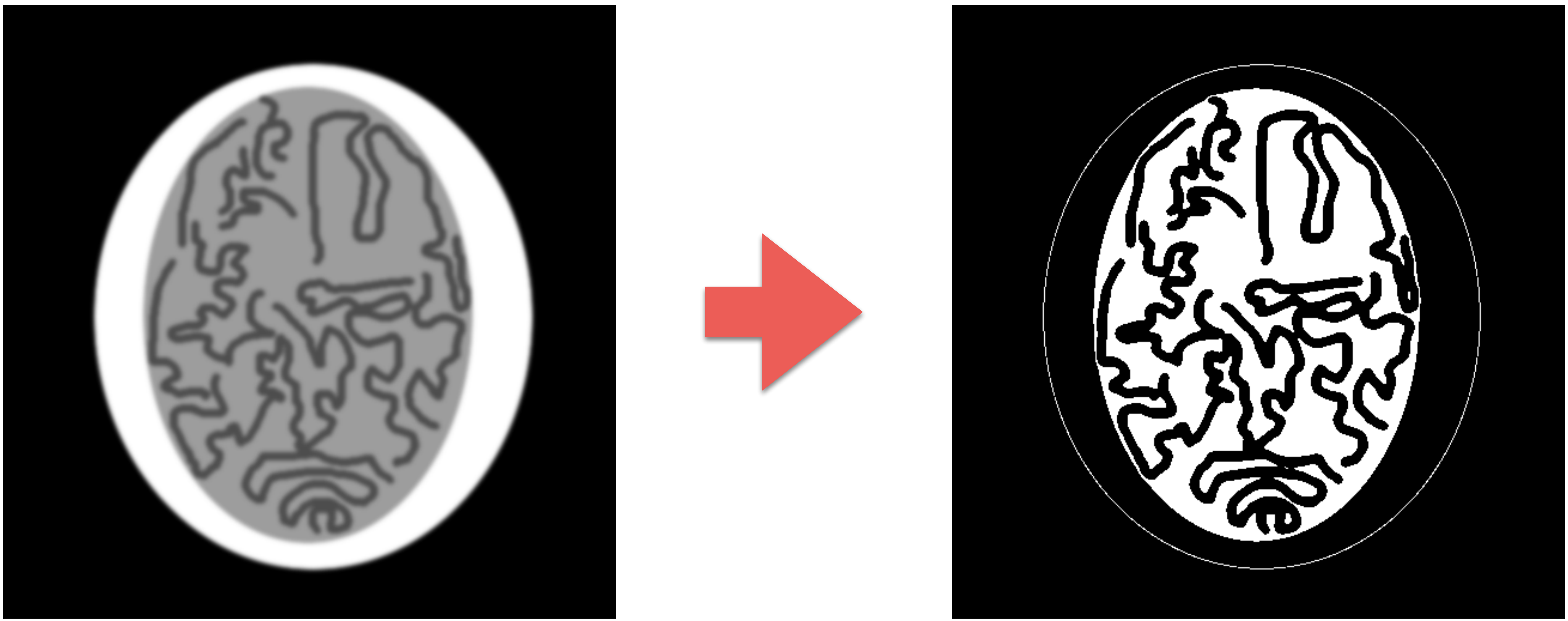


Thresholding Example



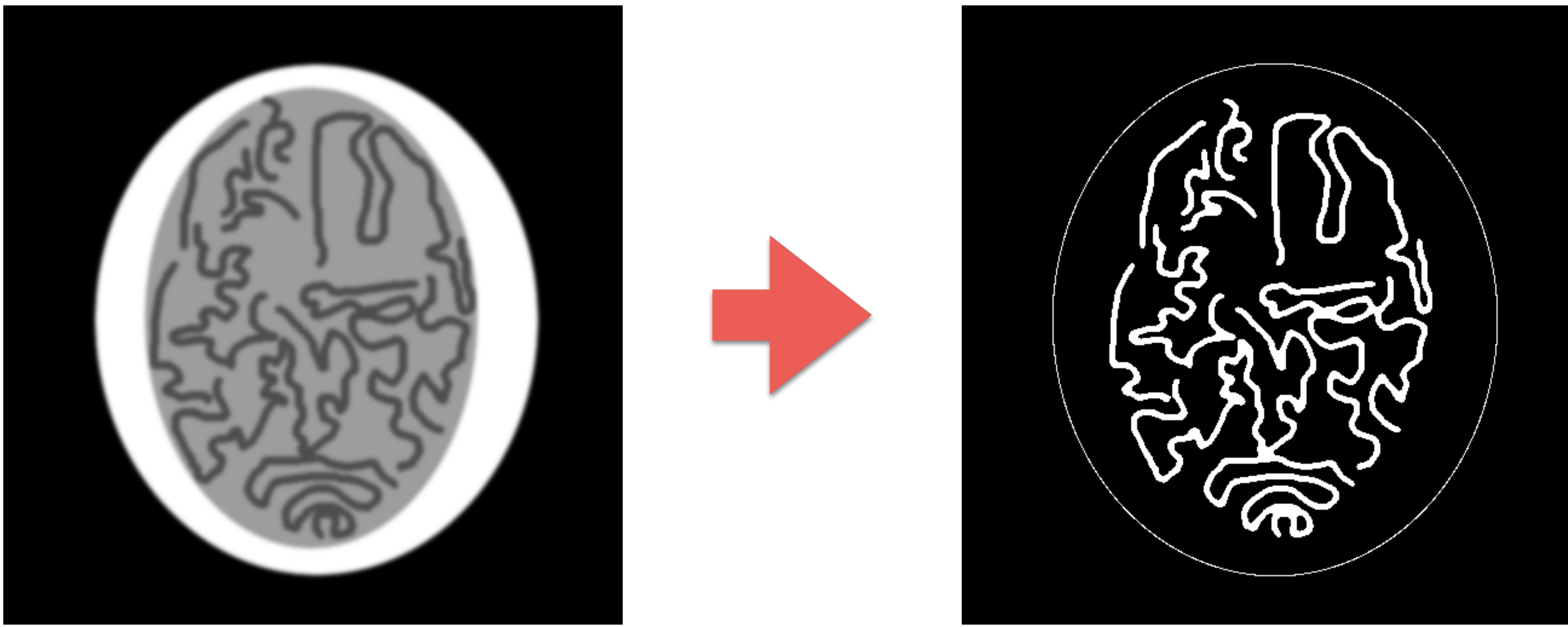
$$I_t = 1 \qquad t = 0.1$$

Thresholding Example



$I_t = 0.6 \qquad t = 0.1$

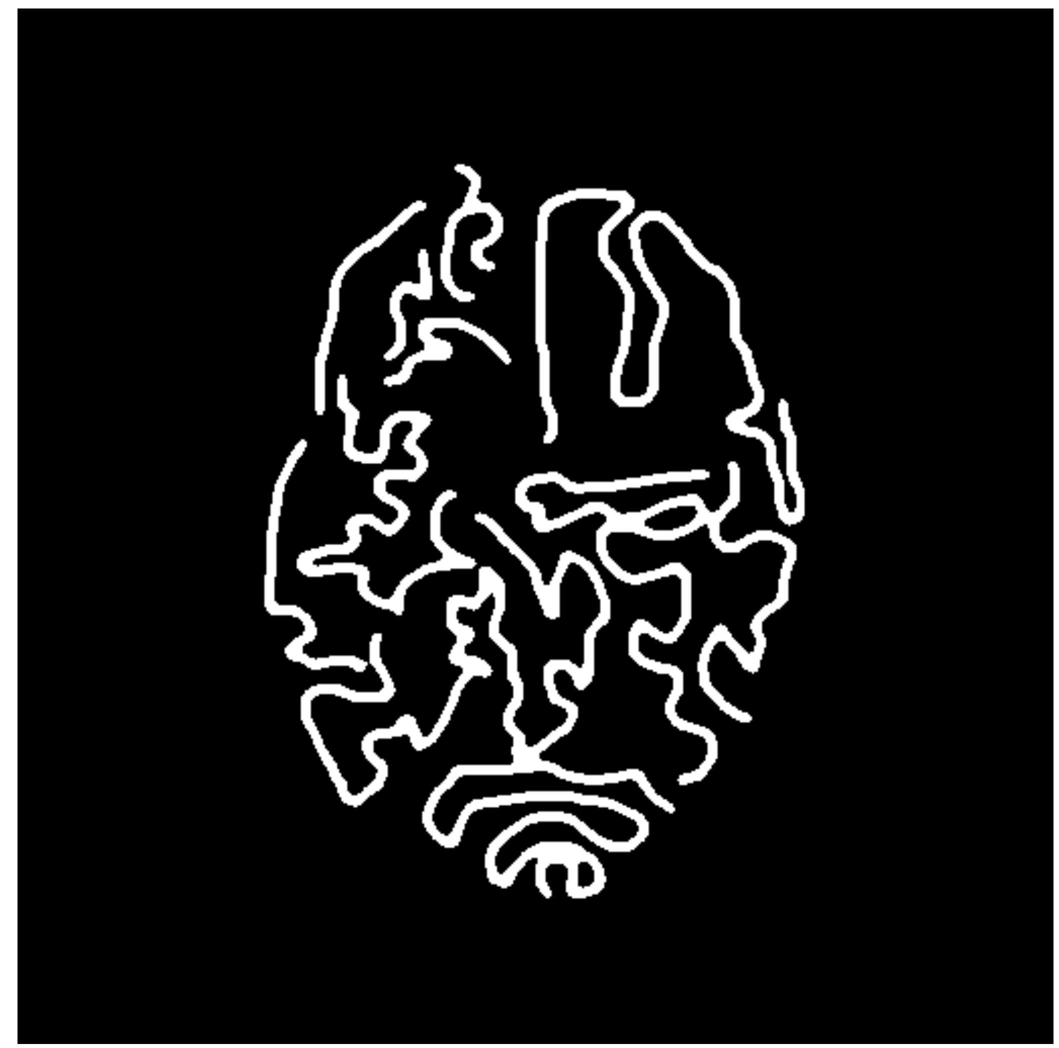
Thresholding Example



$I_t = 0.6 \qquad t = 0.1$

Thresholding: Connected Components

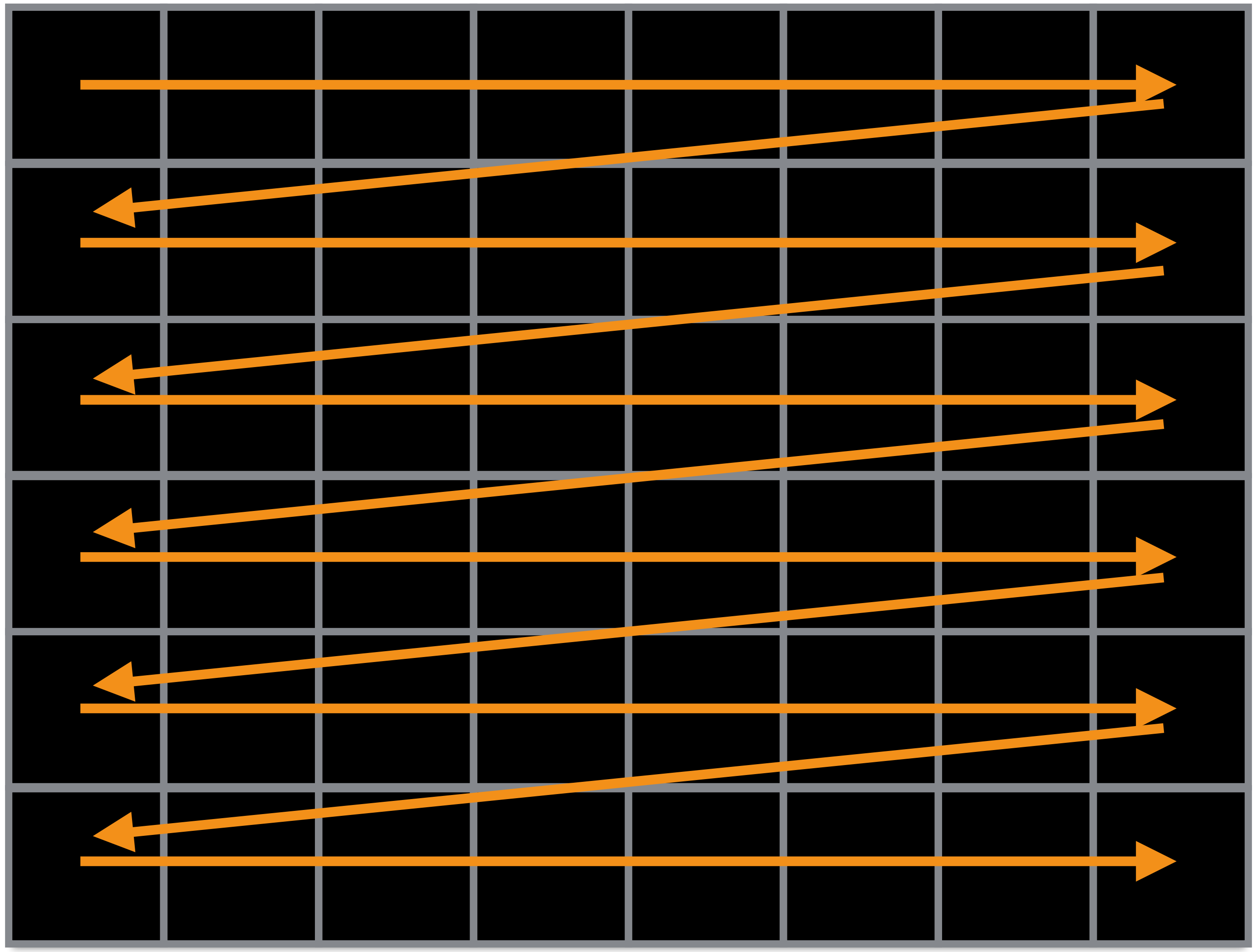
- After segmentation we may end up with different pieces that are not connected.



Thresholding: Connected Components

- A two-pass algorithm that works in scan order (from left to right and from top to bottom).
- 1-Pass: it creates labels to groups of pixel.
- 2-Pass: it merges groups that are connected.

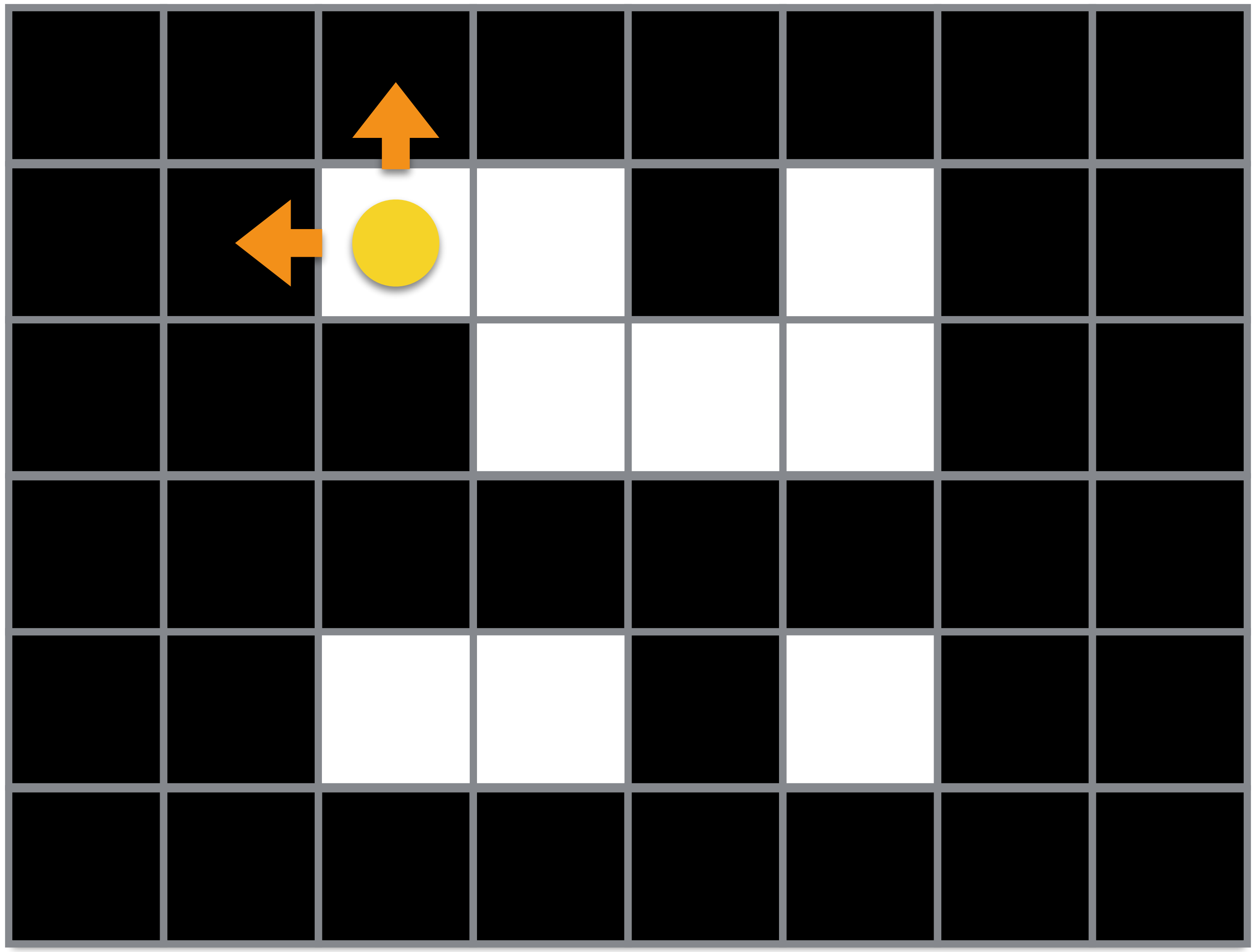
Thresholding: Connected Components



Scan order

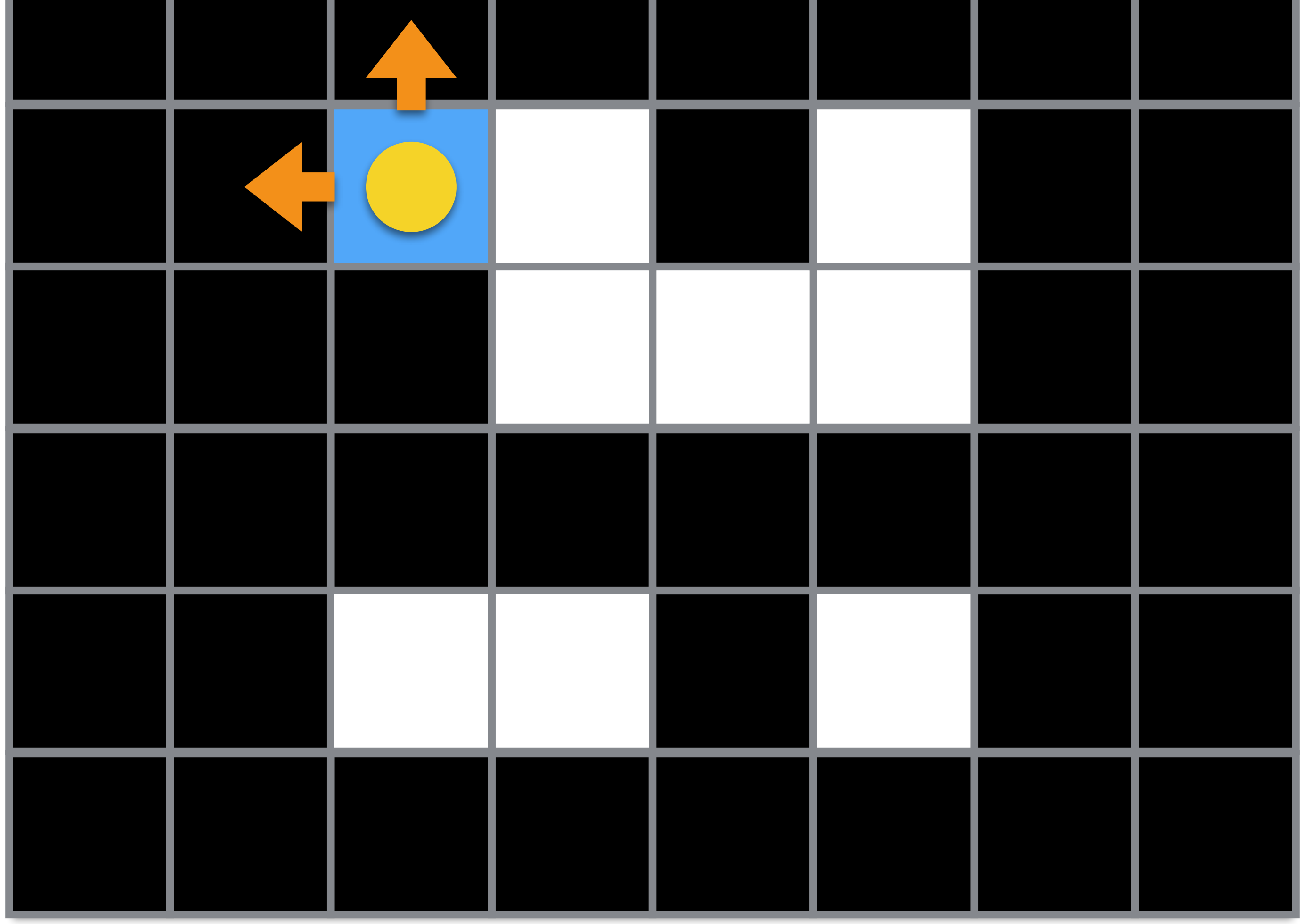
First Pass

Thresholding: Connected Components



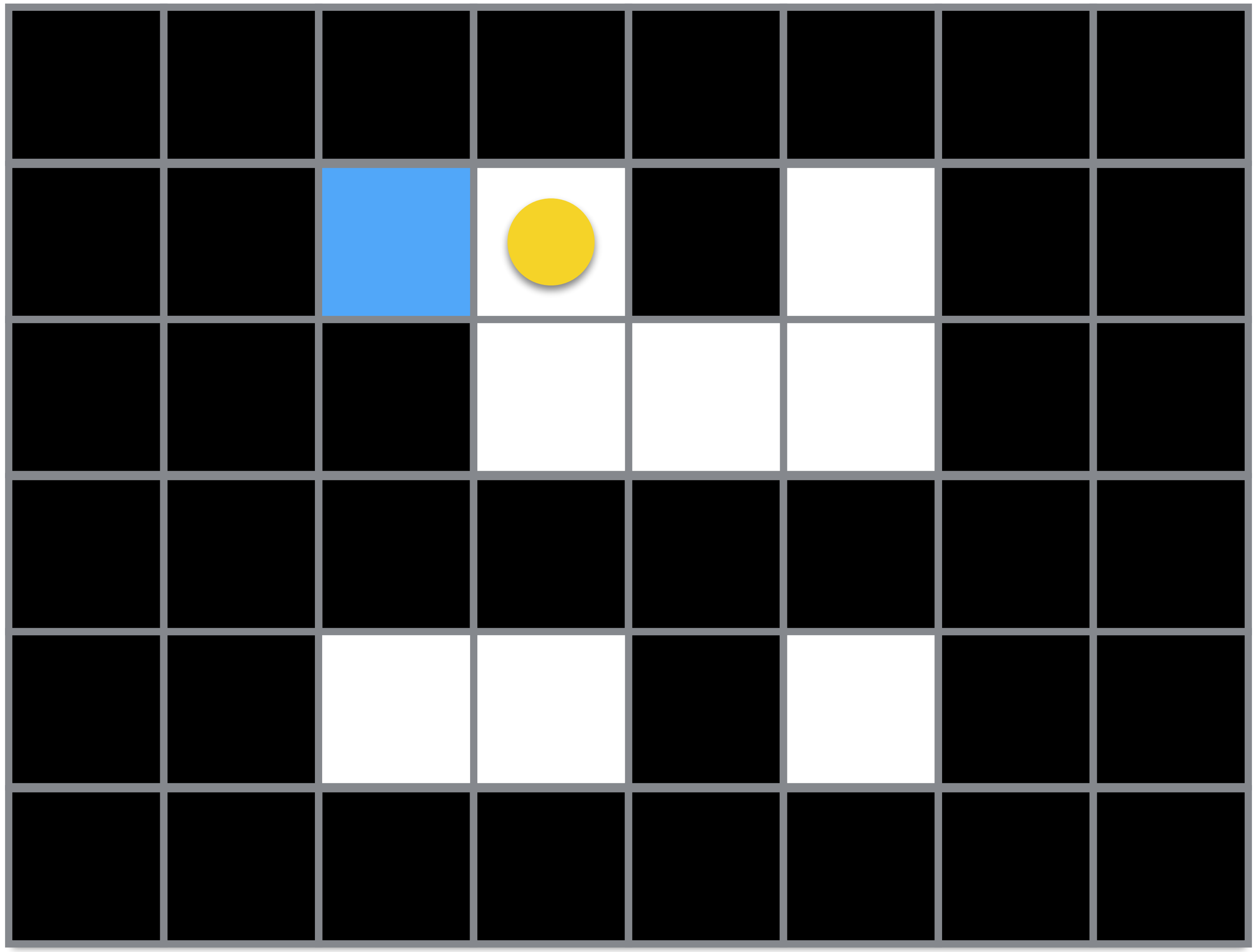
We check up and left neighbors to see if they have a label.

Thresholding: Connected Components



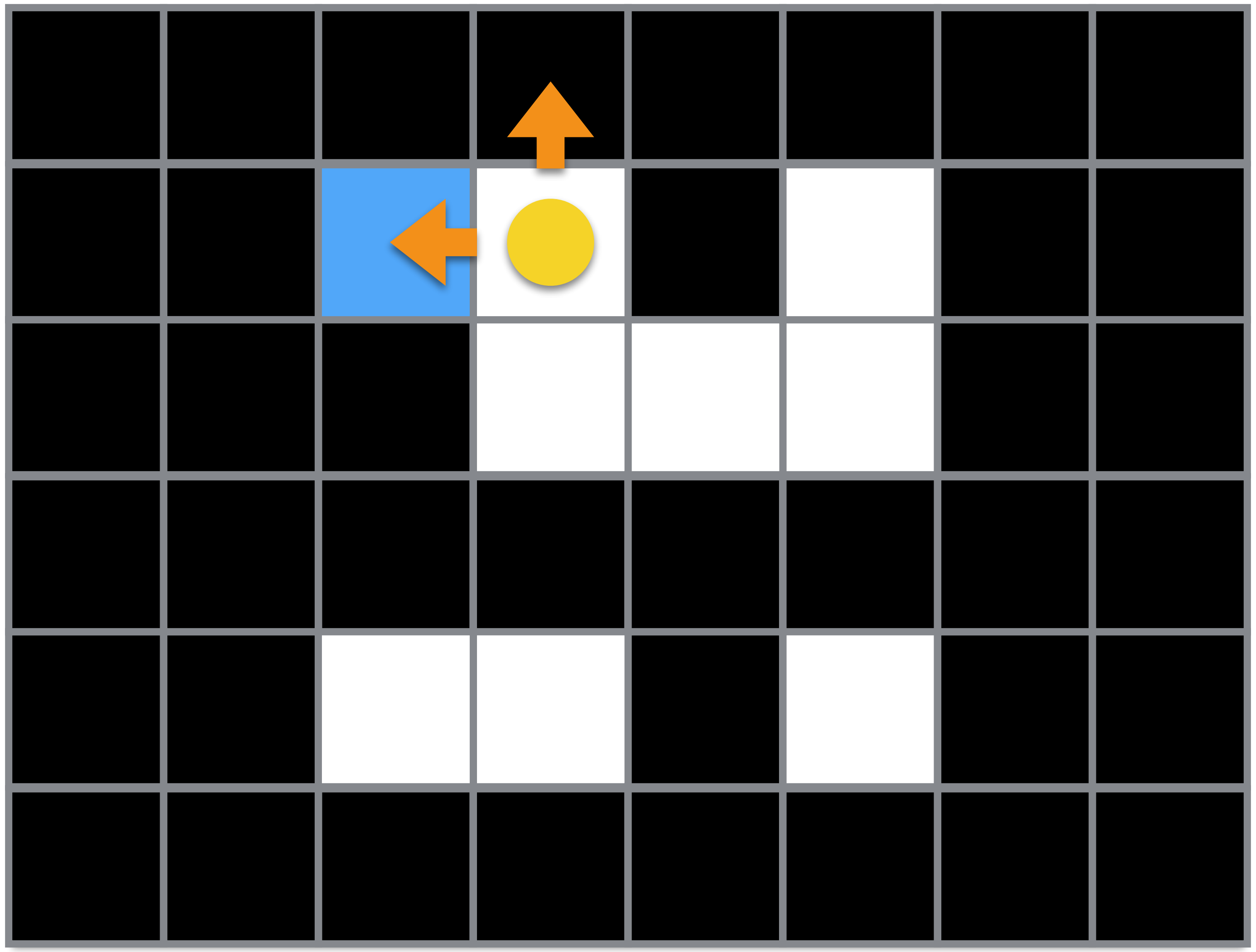
If not we create a new one.

Thresholding: Connected Components



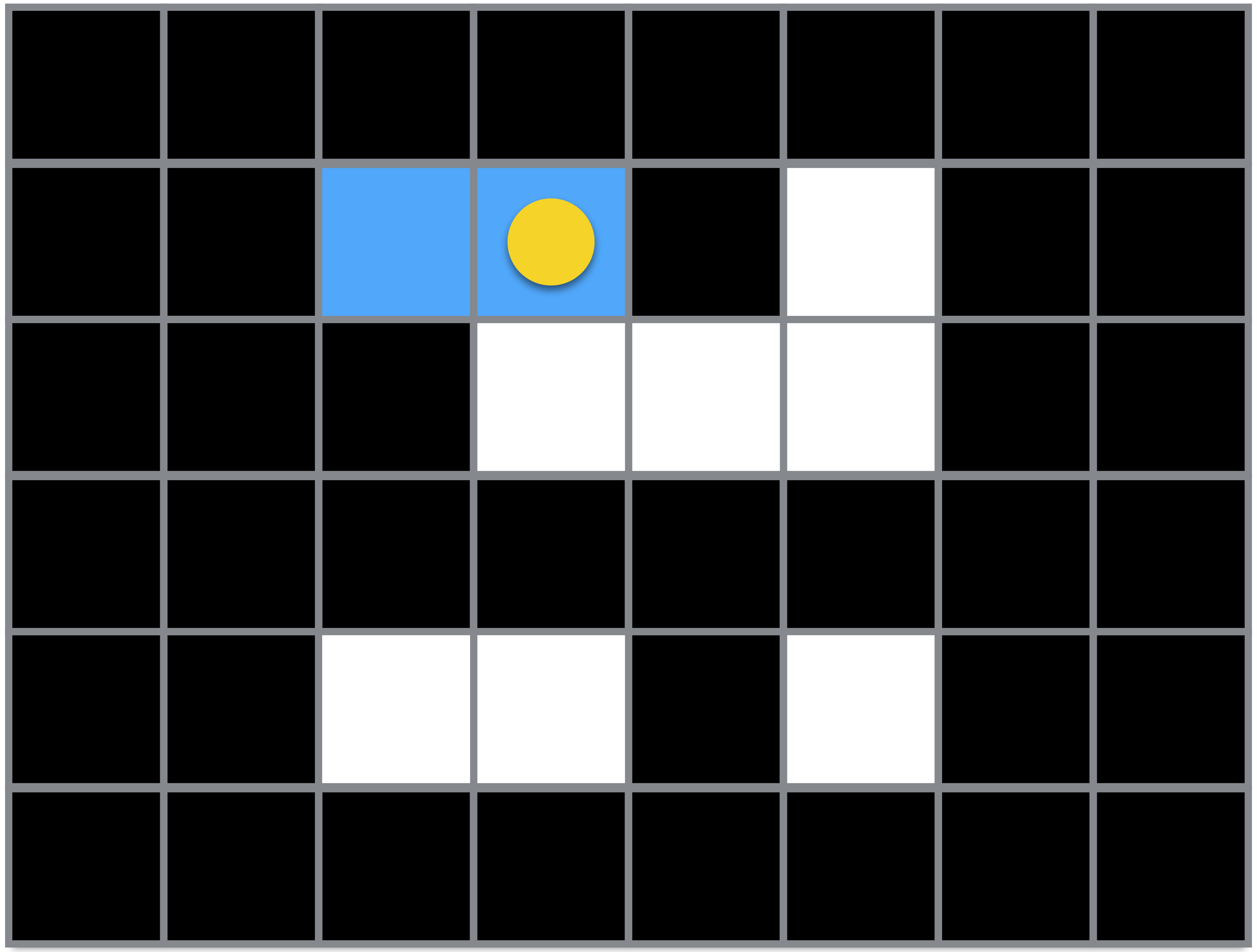
Then, we move right, and we repeat the process.

Thresholding: Connected Components



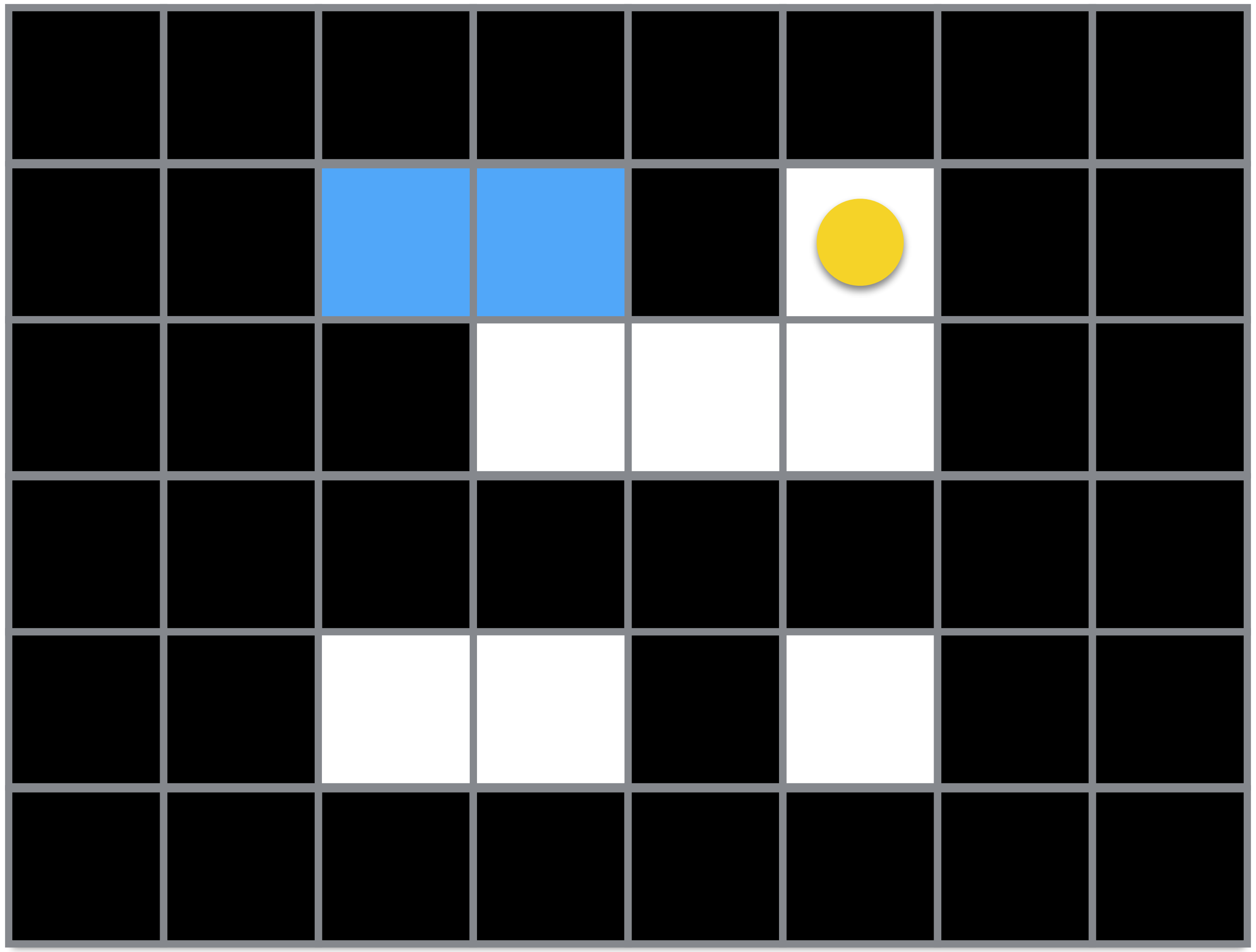
In this case, the left neighbor has a label, so we reuse it.

Thresholding: Connected Components

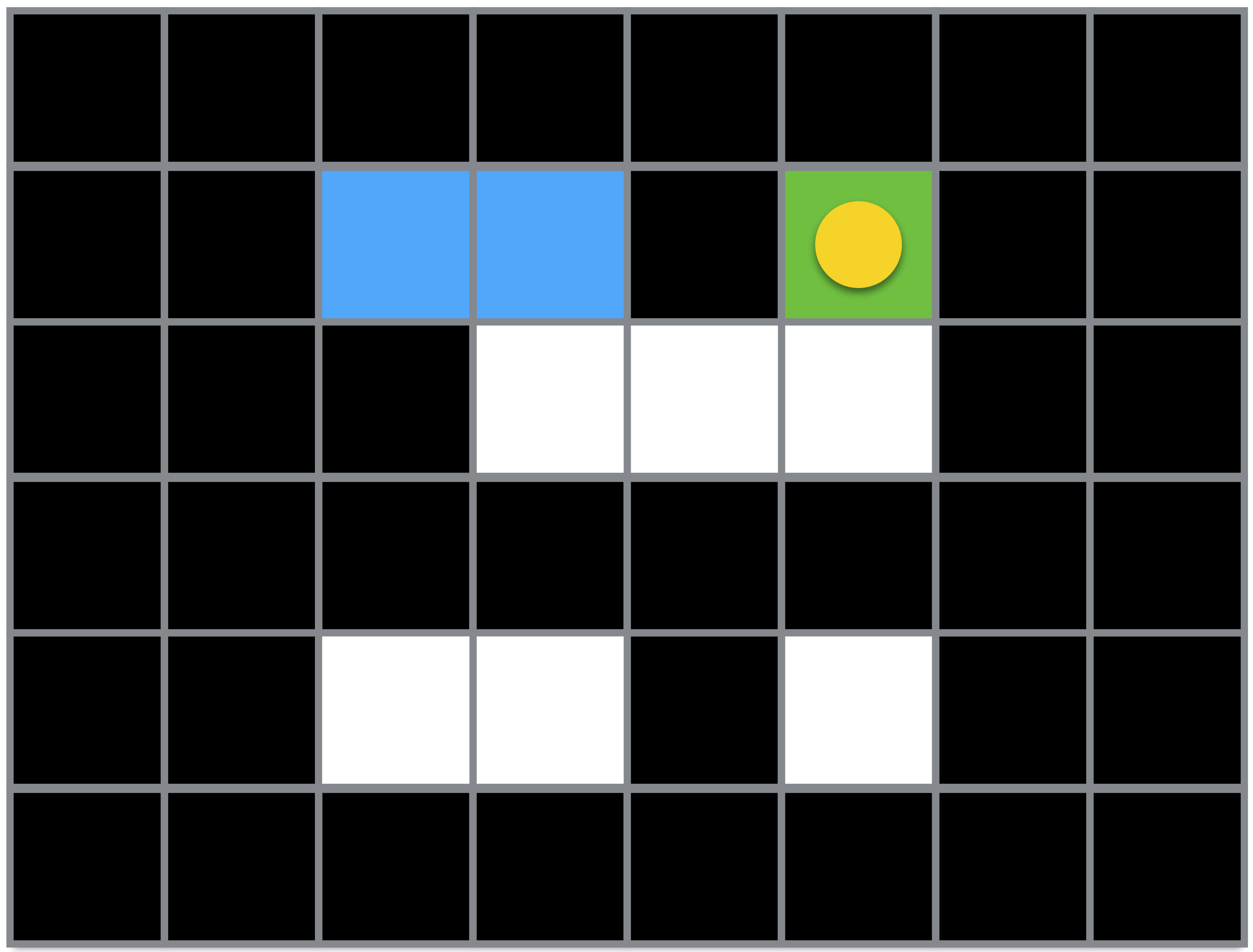


In this case, the left neighbor has a label, so we reuse it.

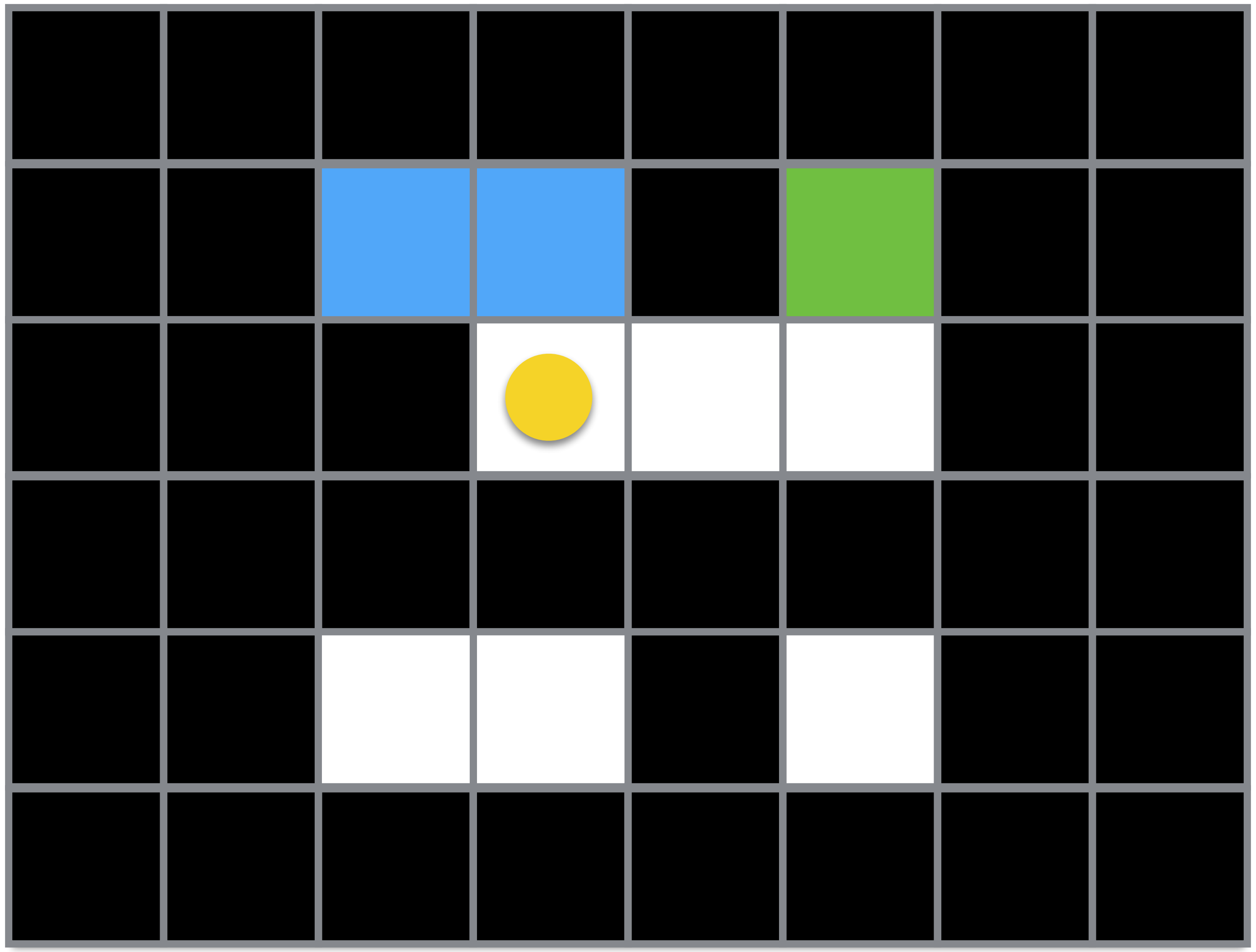
Thresholding: Connected Components



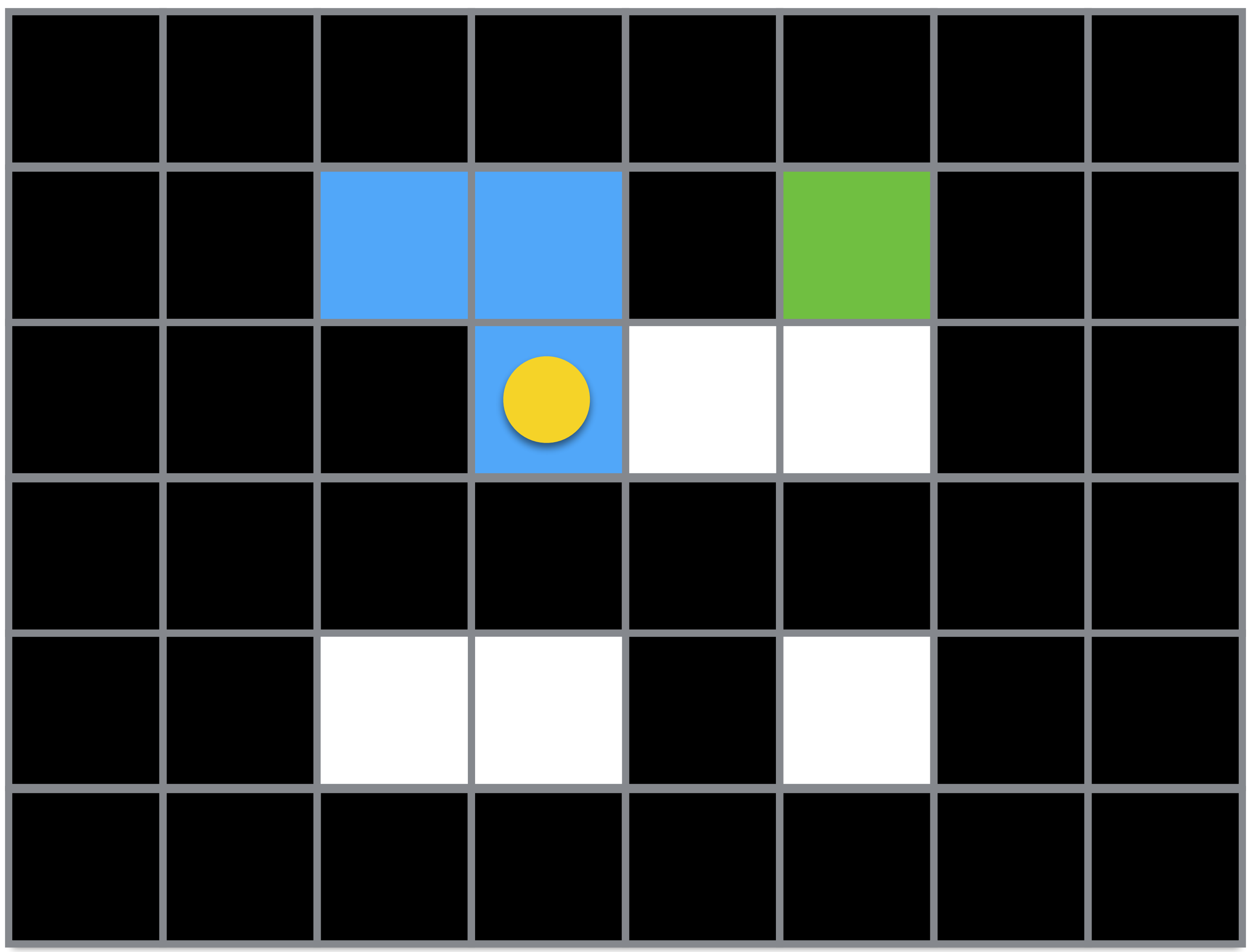
Thresholding:
Connected Components



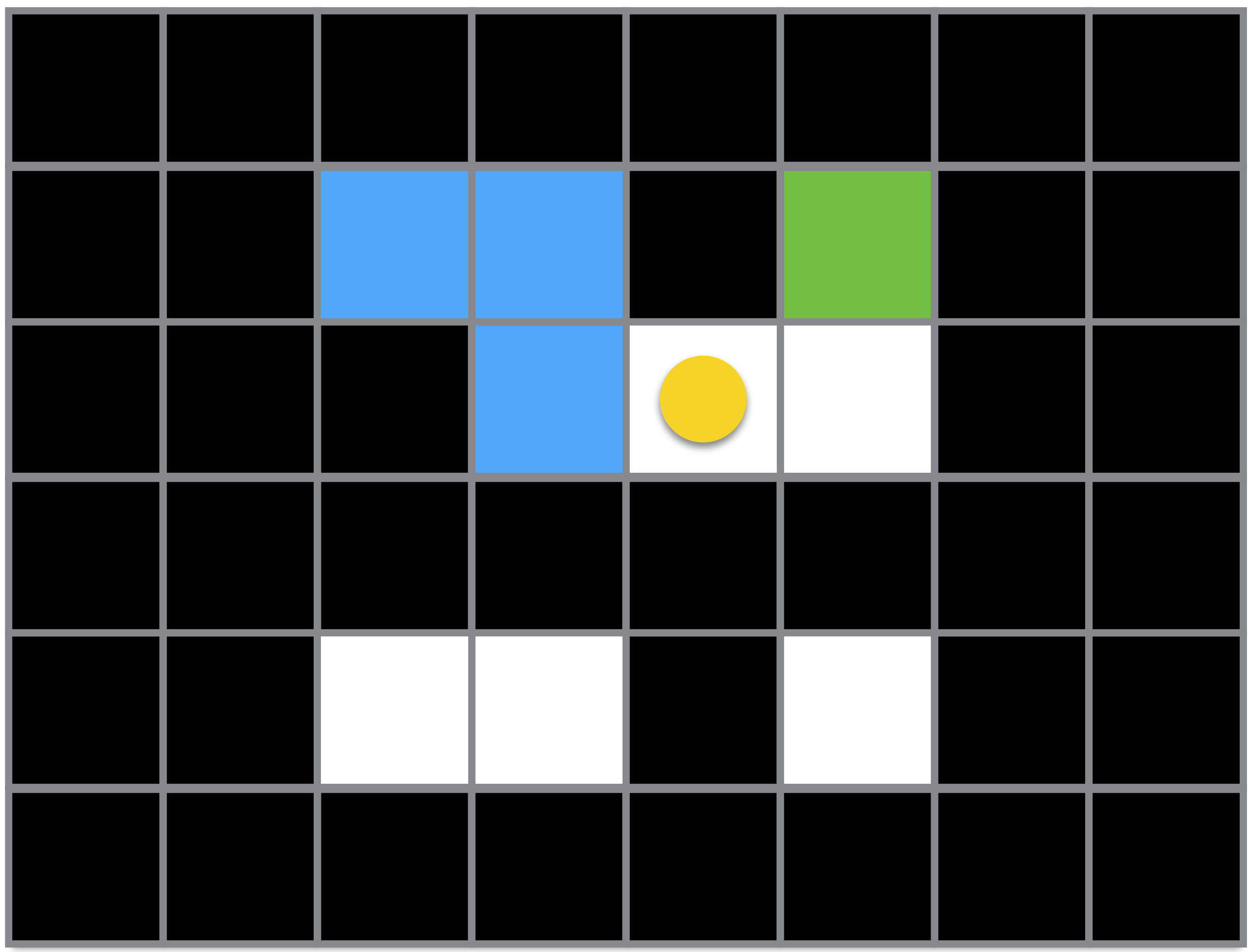
Thresholding:
Connected Components



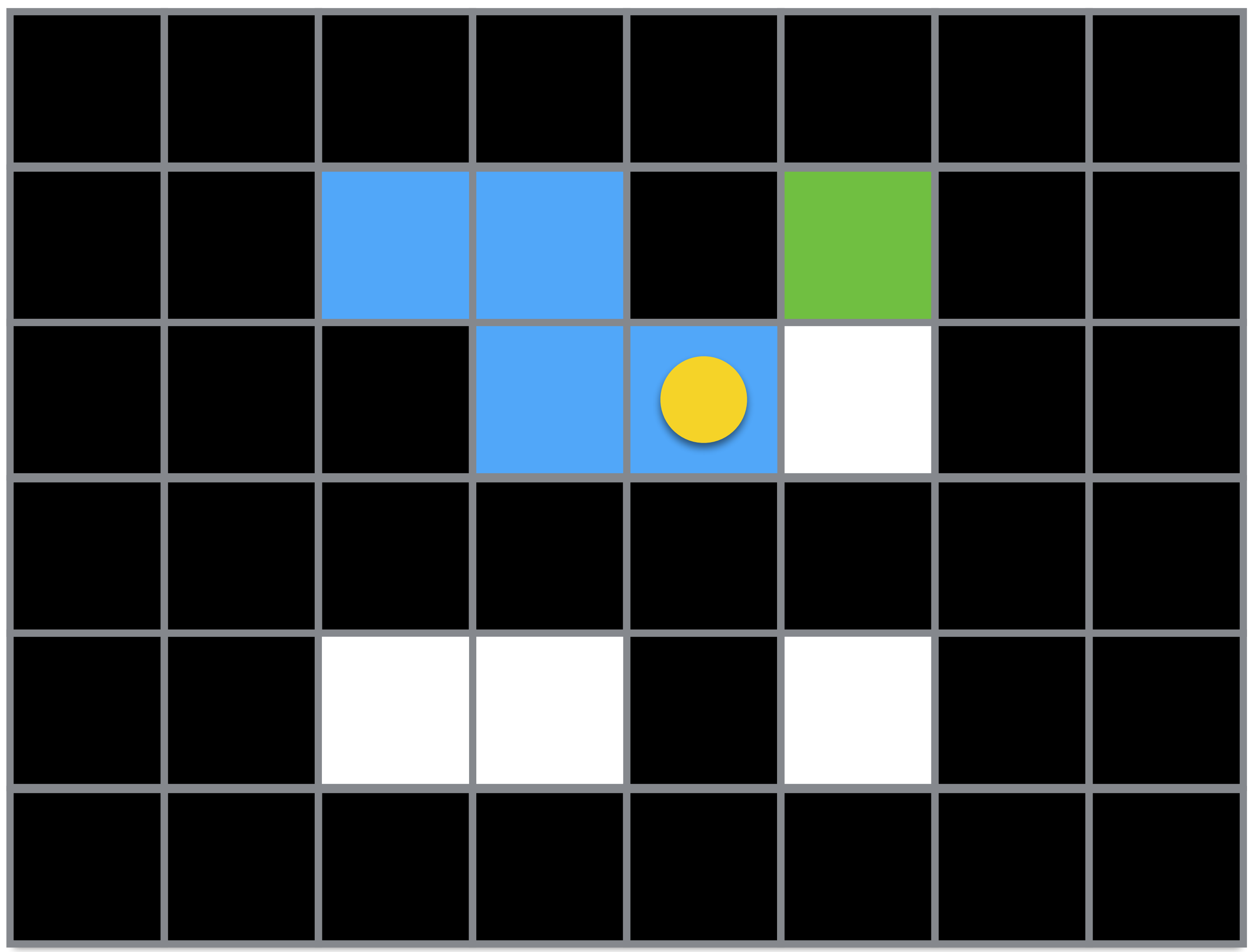
Thresholding: Connected Components



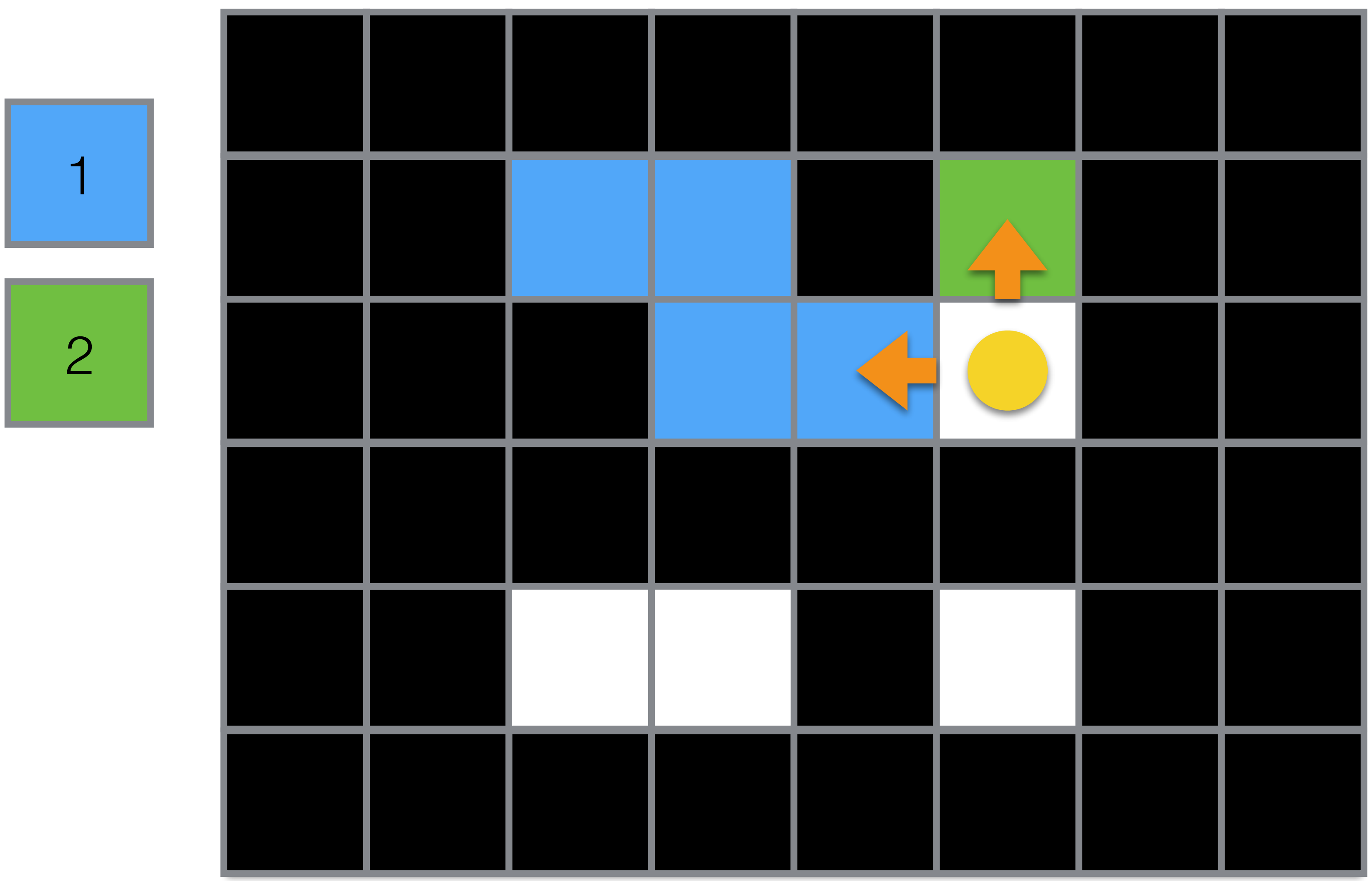
Thresholding: Connected Components



Thresholding: Connected Components

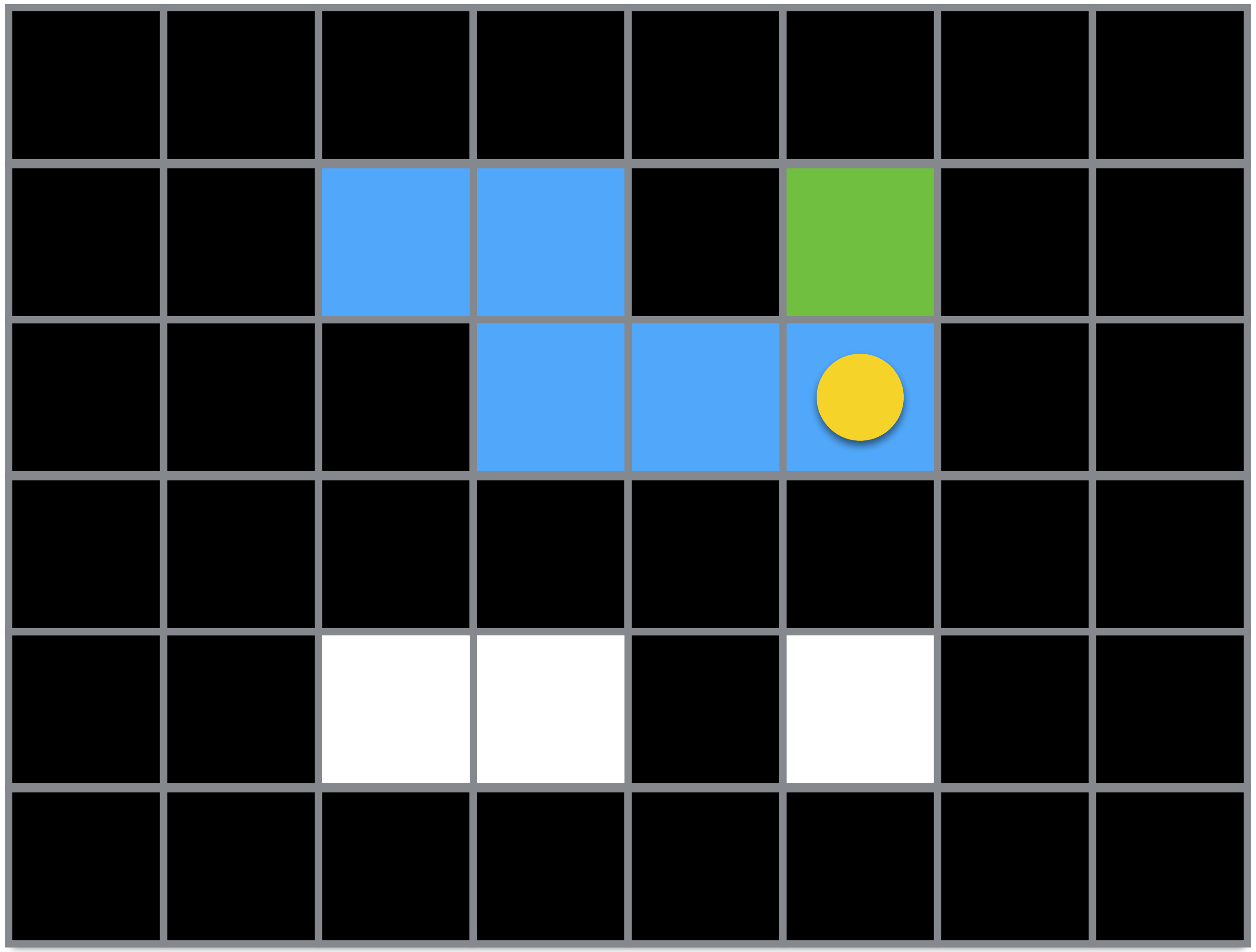


Thresholding: Connected Components

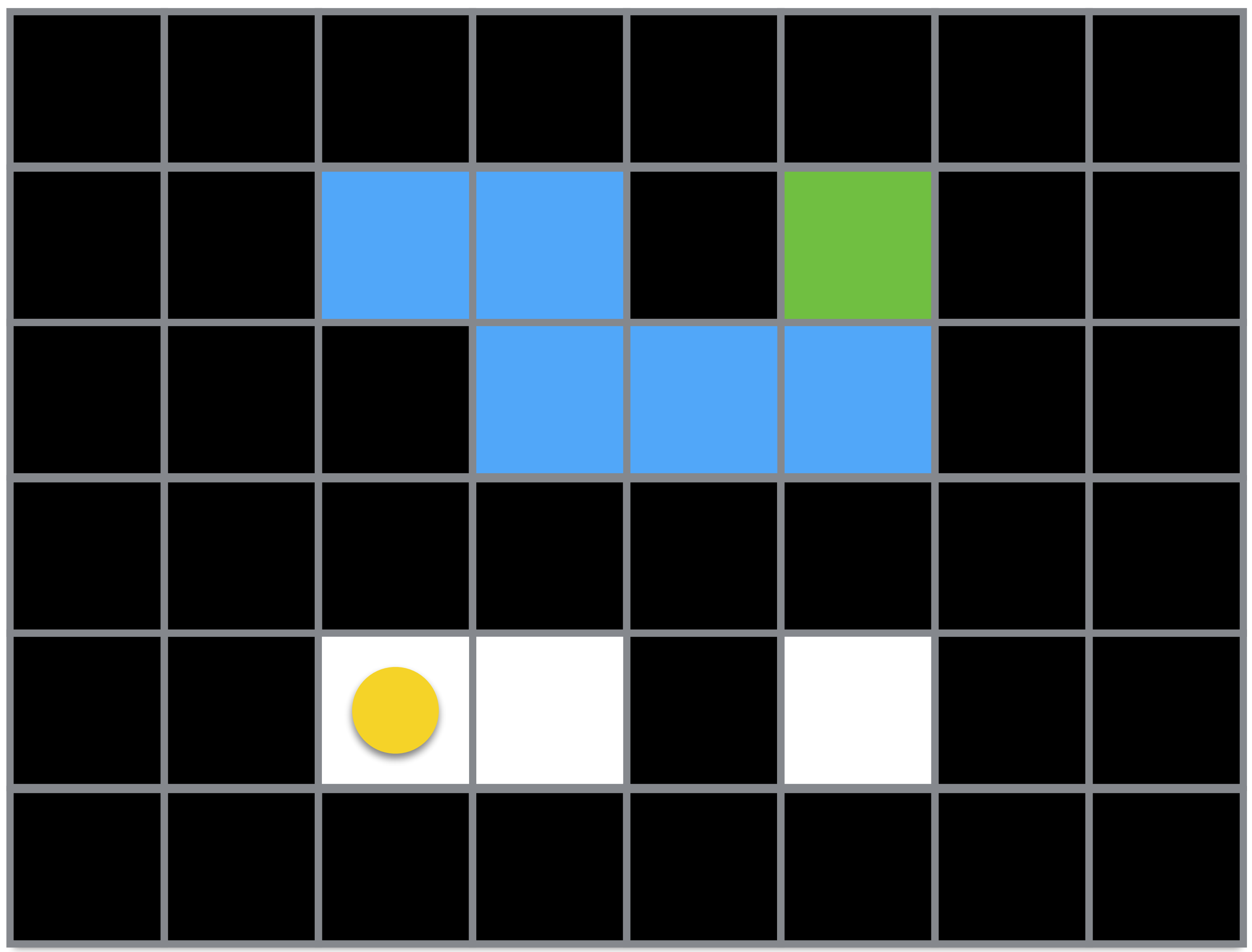


In this case, we choose the lowest label, and we store that 1 is equivalent to 2

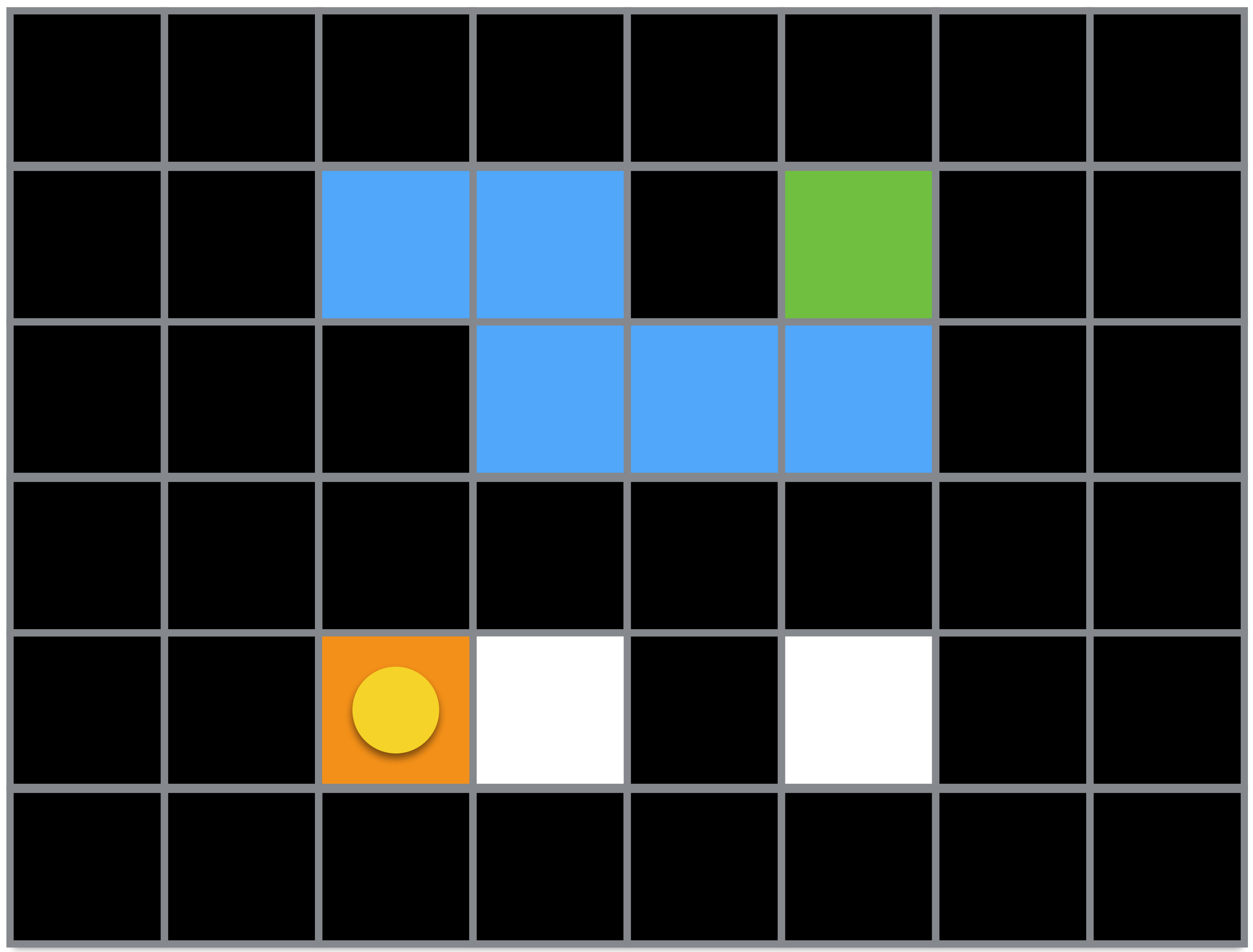
Thresholding: Connected Components



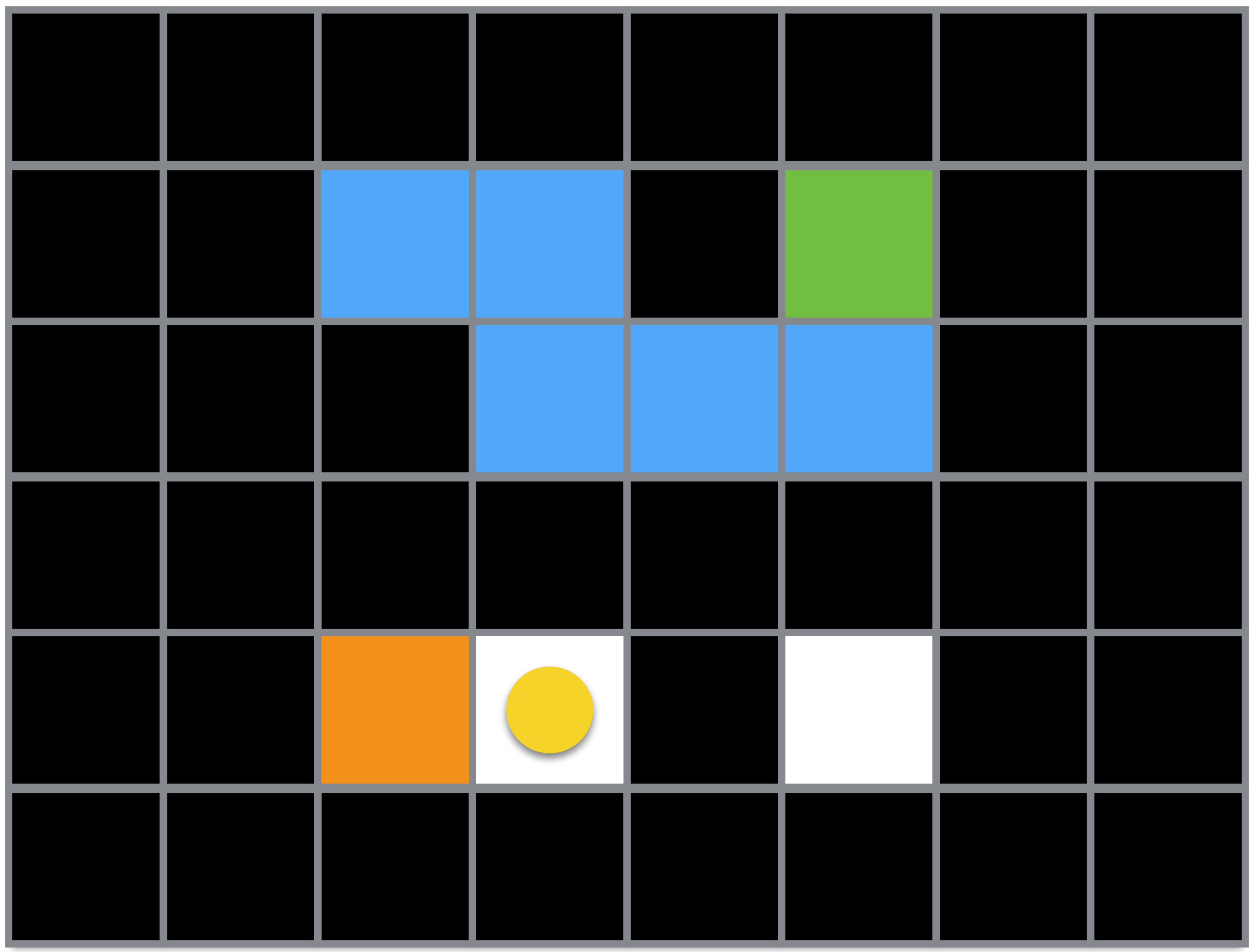
Thresholding: Connected Components



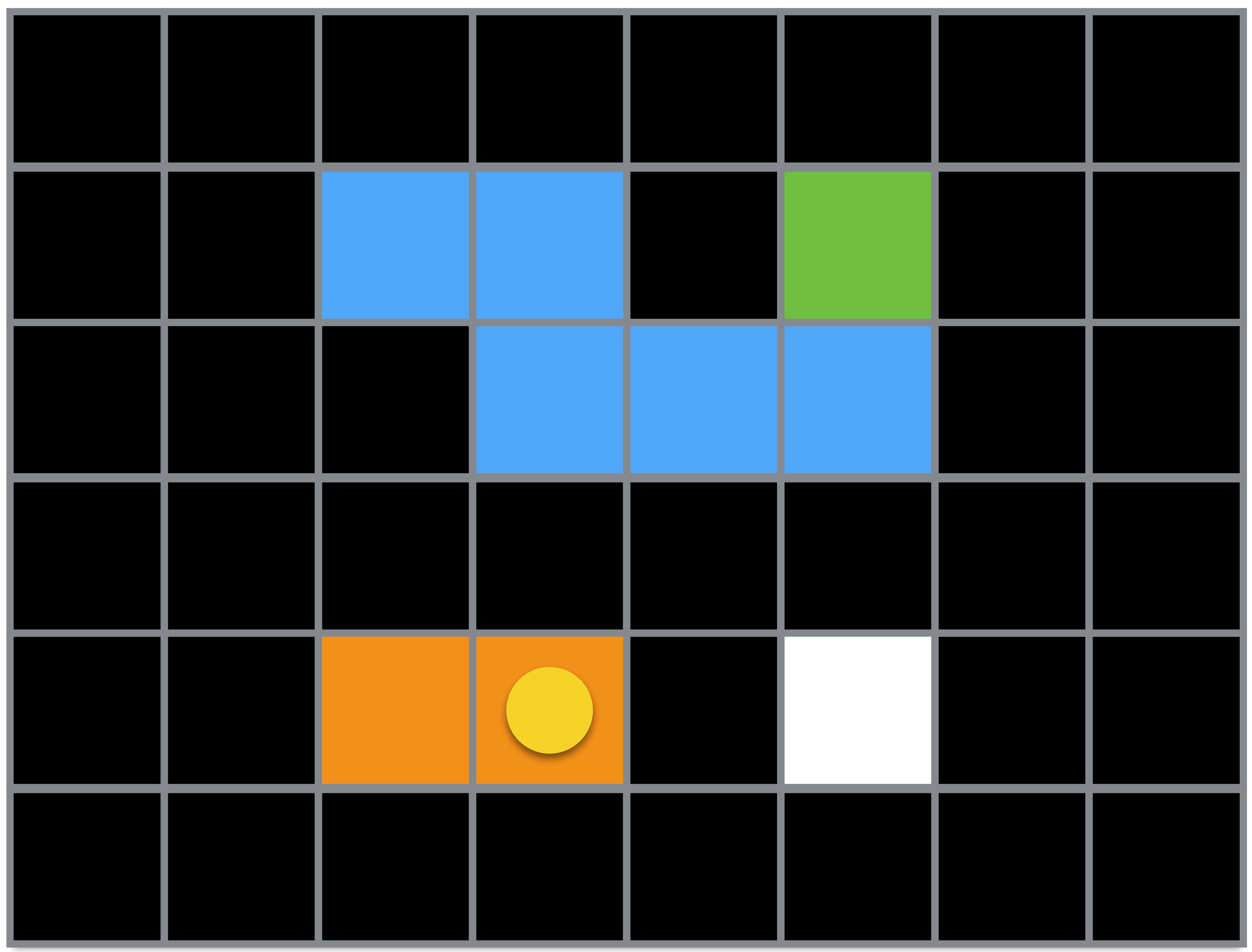
Thresholding: Connected Components



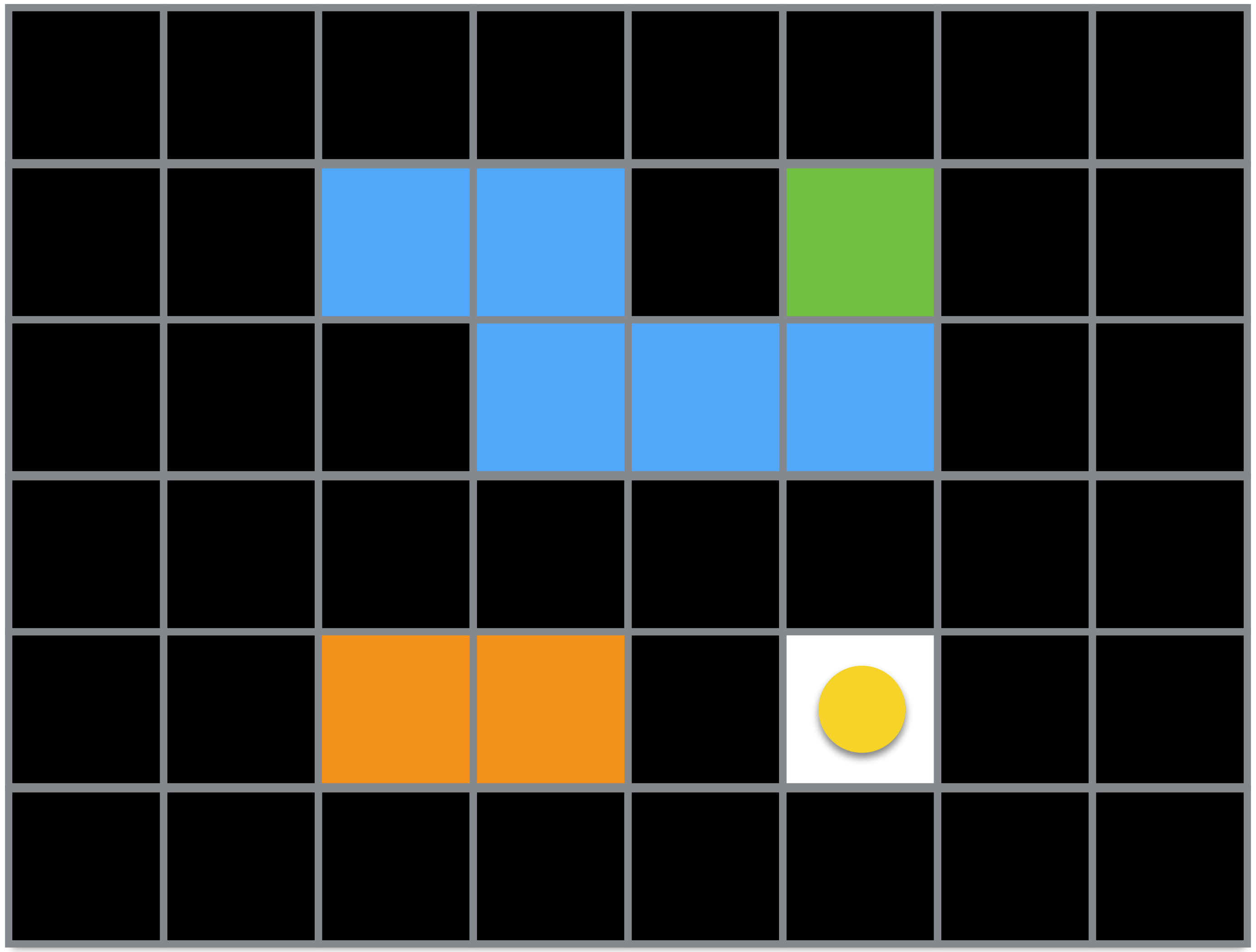
Thresholding: Connected Components



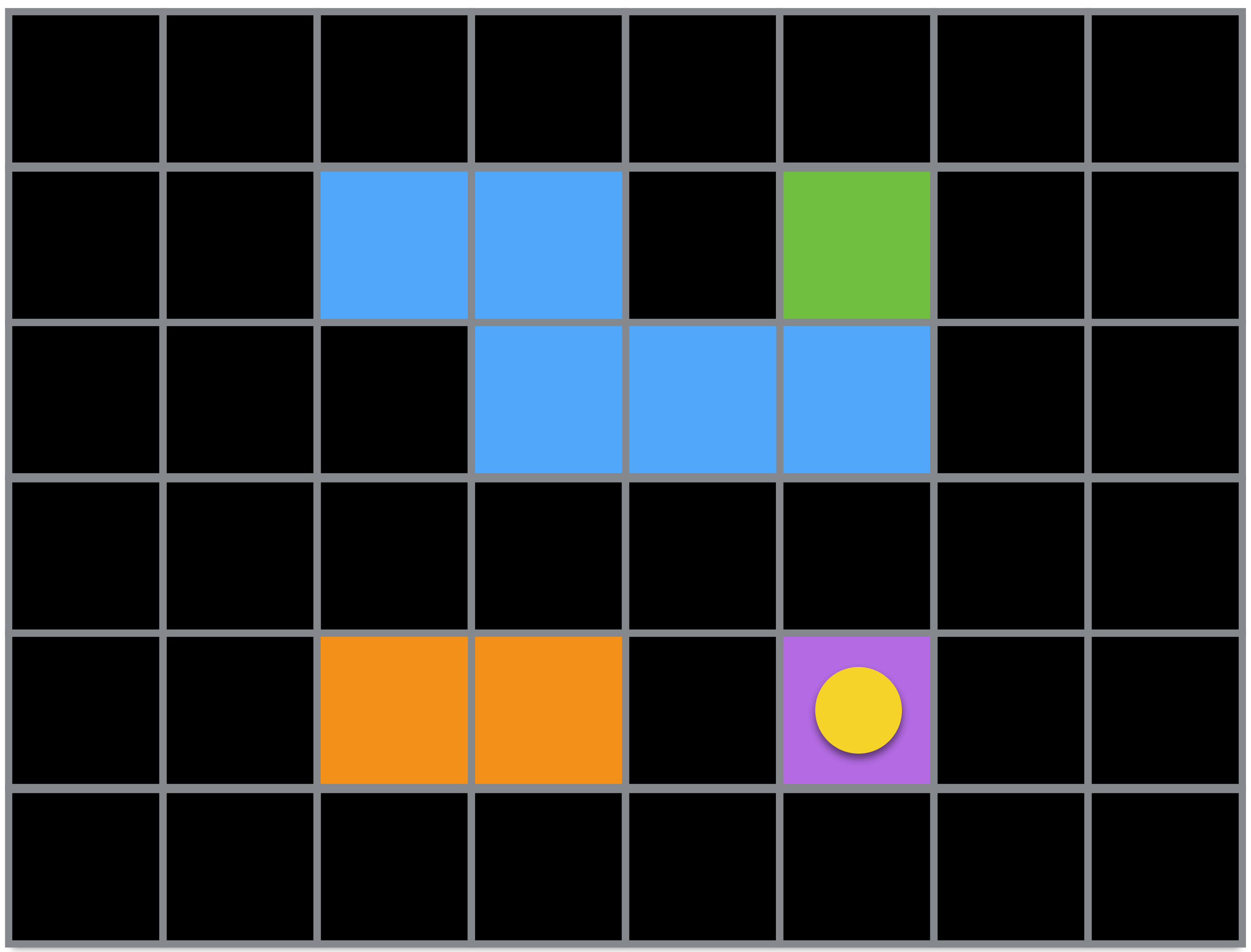
Thresholding:
Connected Components



Thresholding: Connected Components

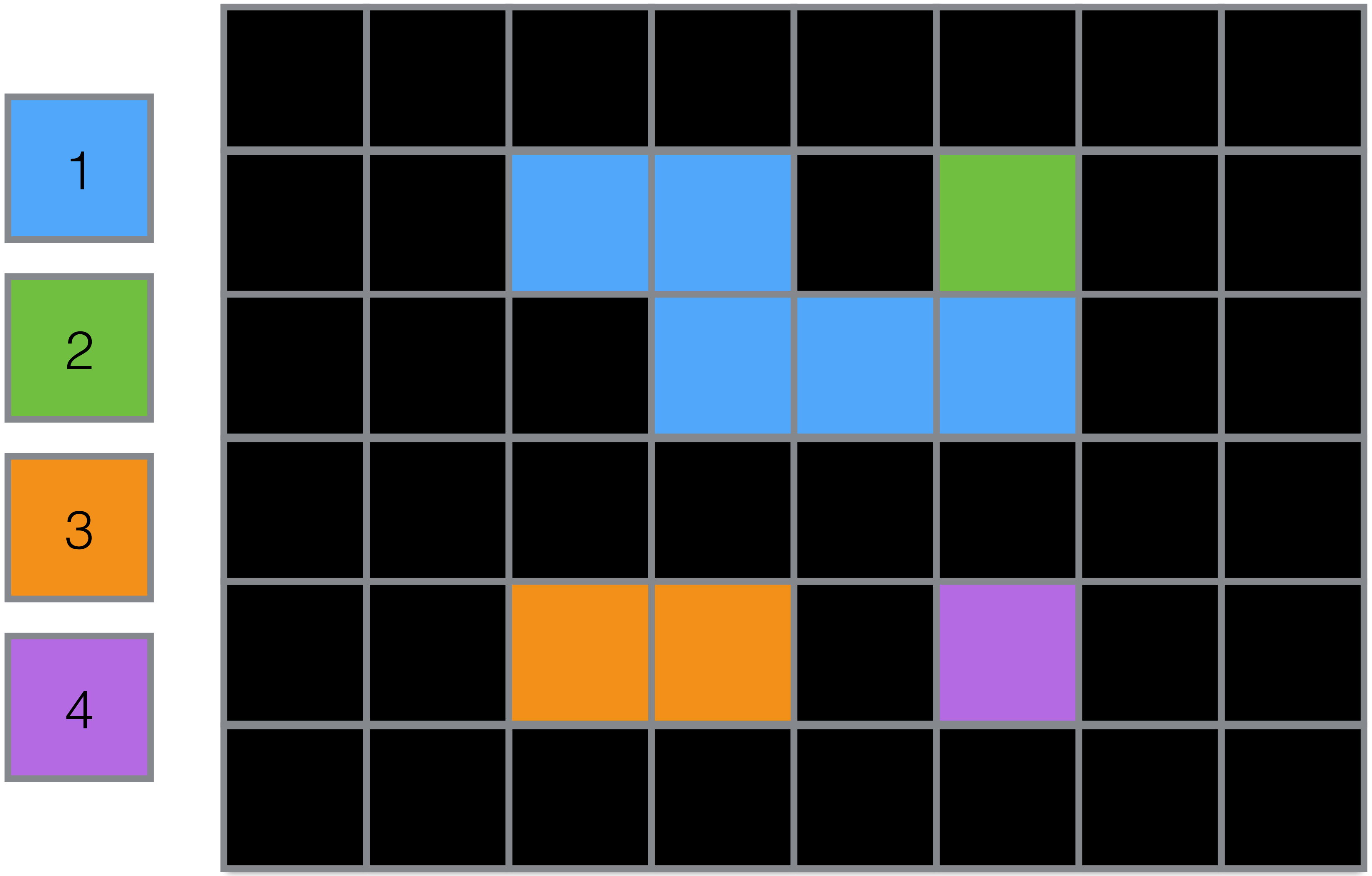


Thresholding: Connected Components



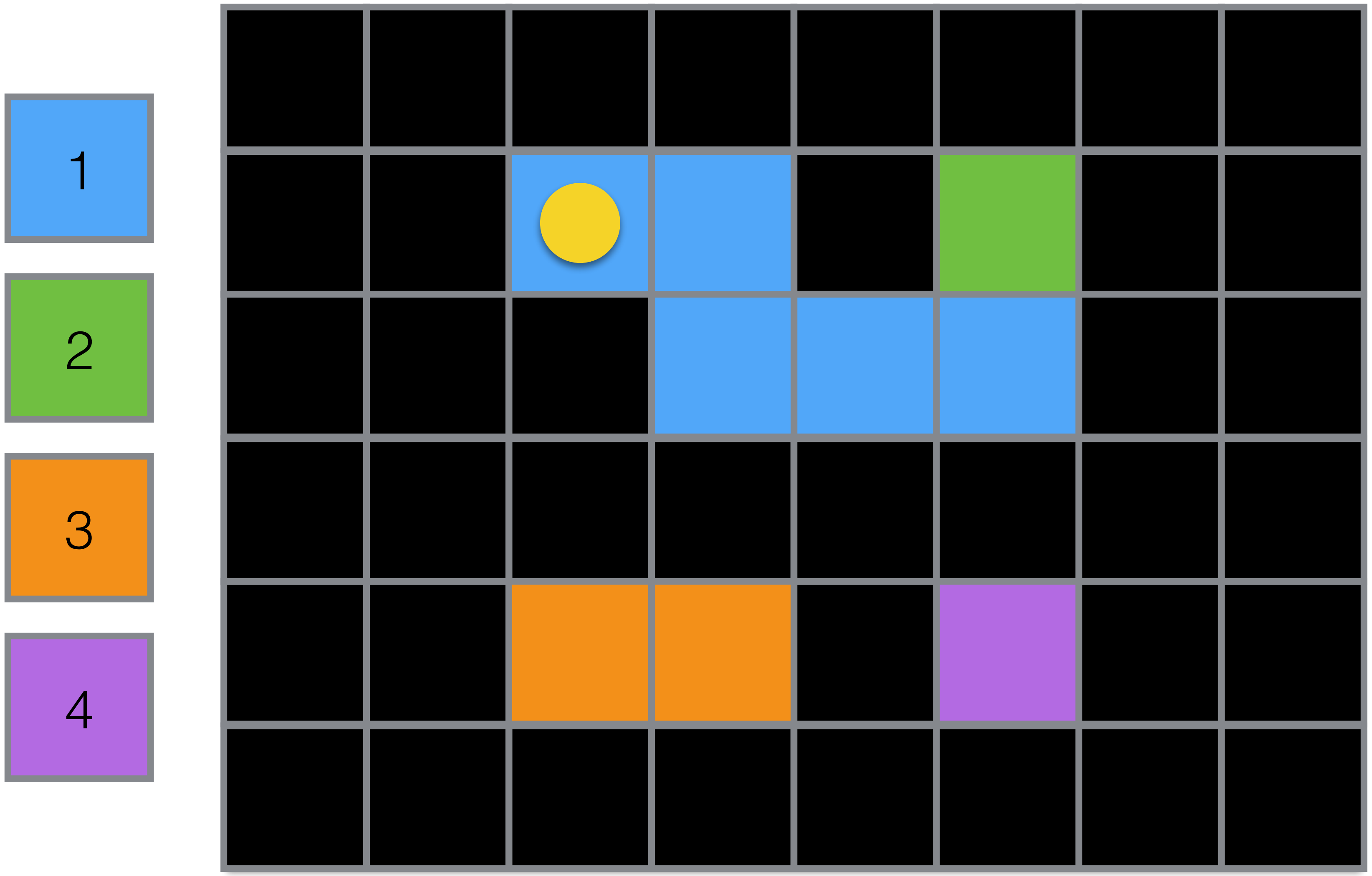
Second Pass

Thresholding: Connected Components

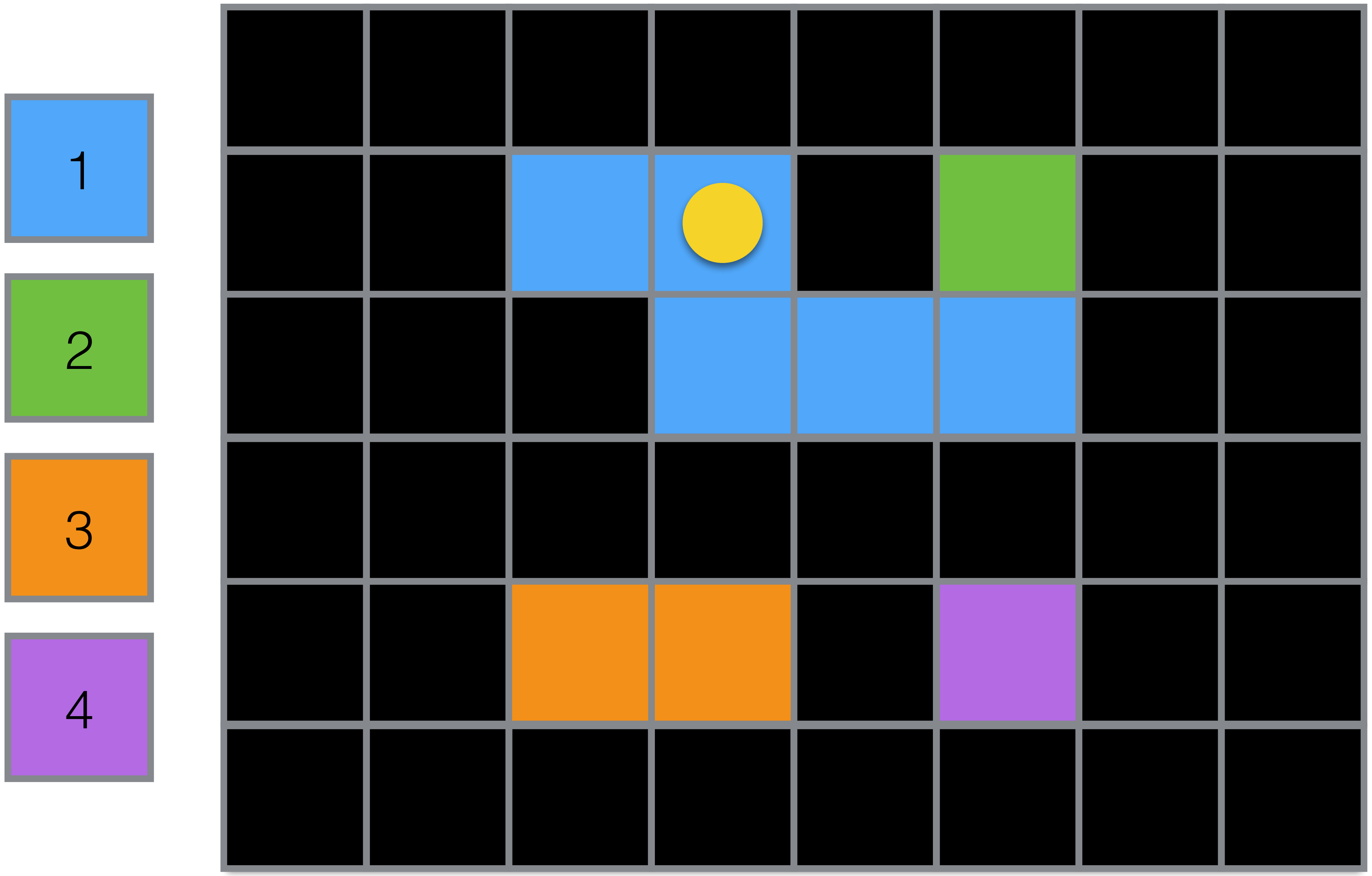


We go through all pixels. For each pixel we set the value of lowest equivalent.

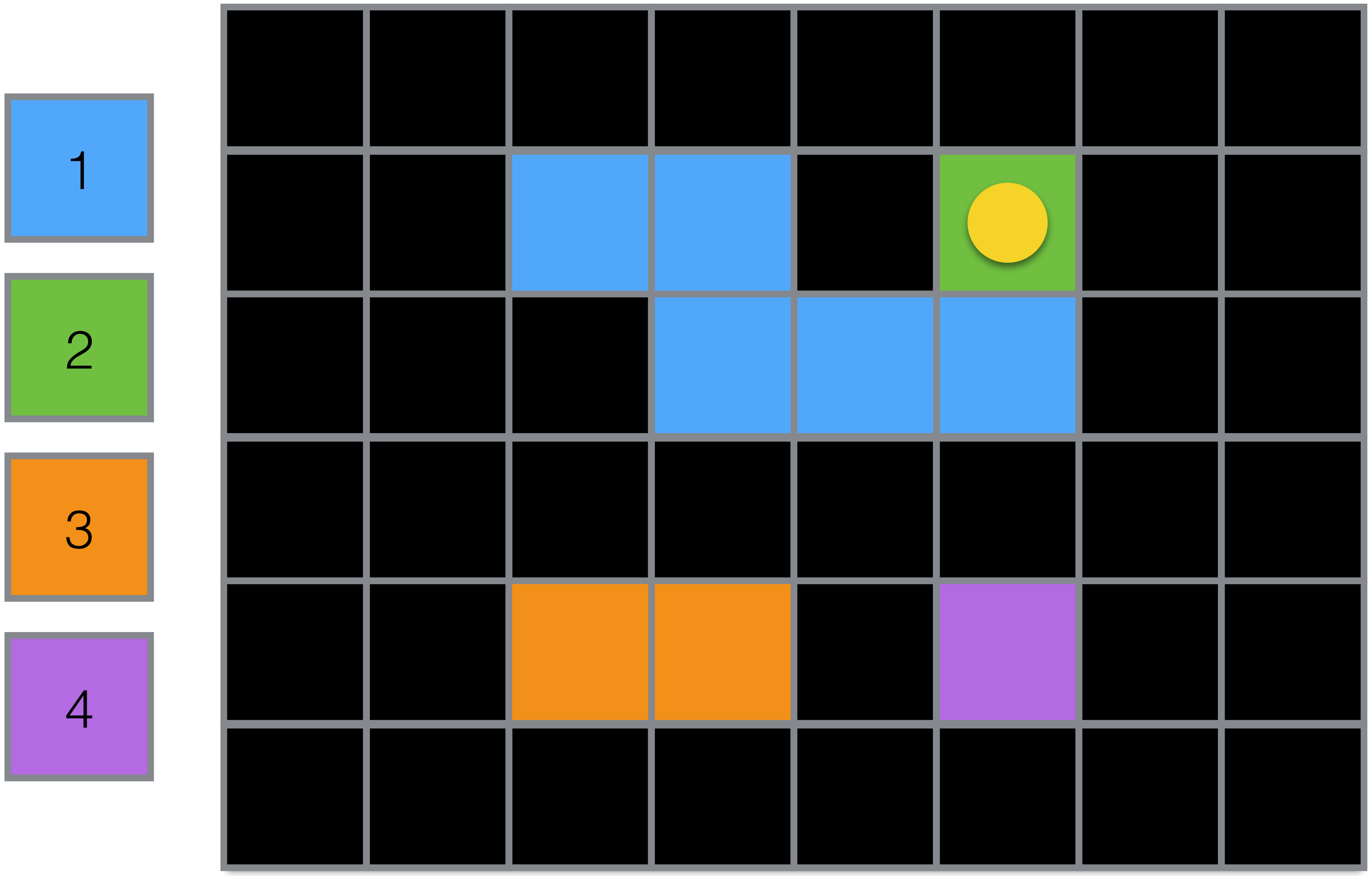
Thresholding: Connected Components



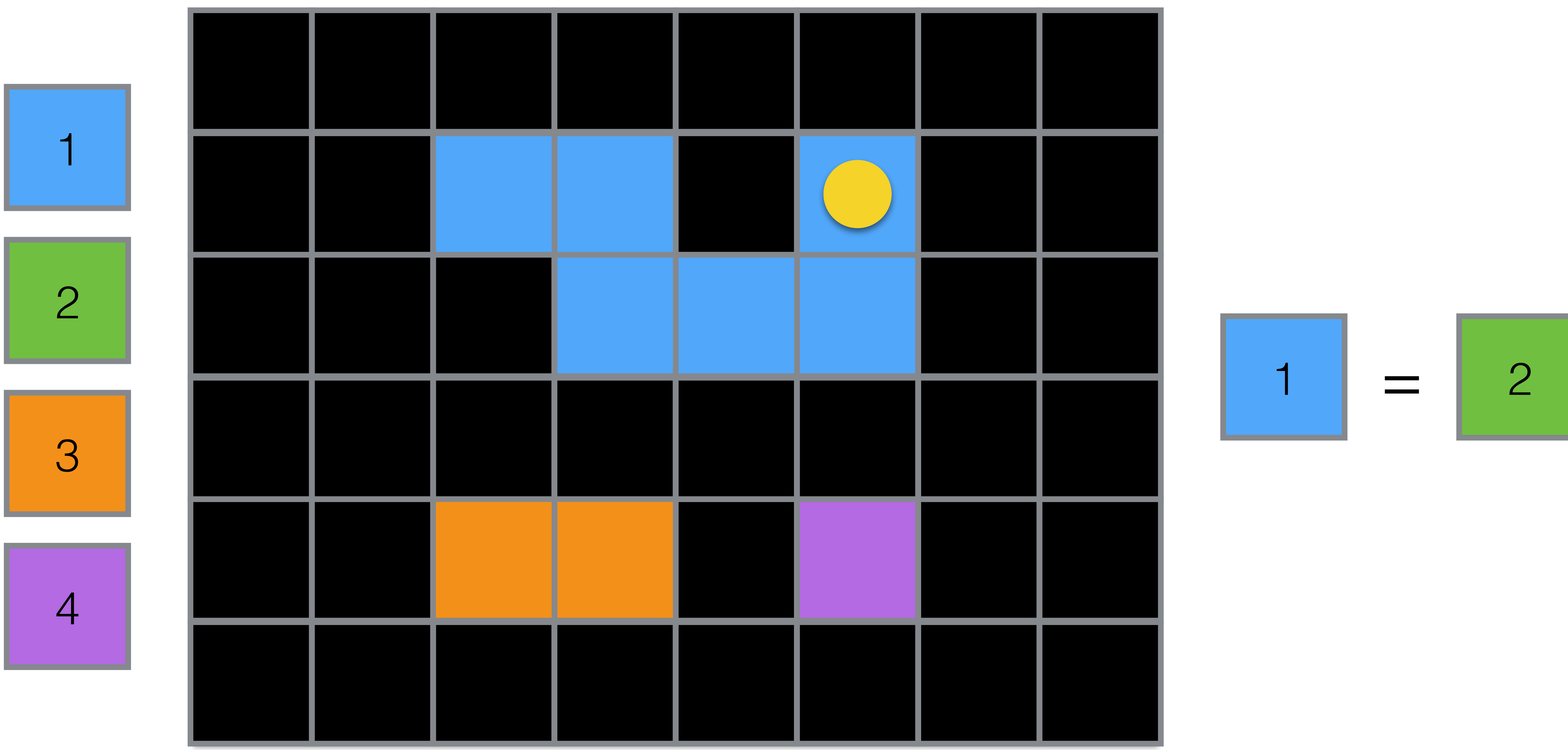
Thresholding: Connected Components



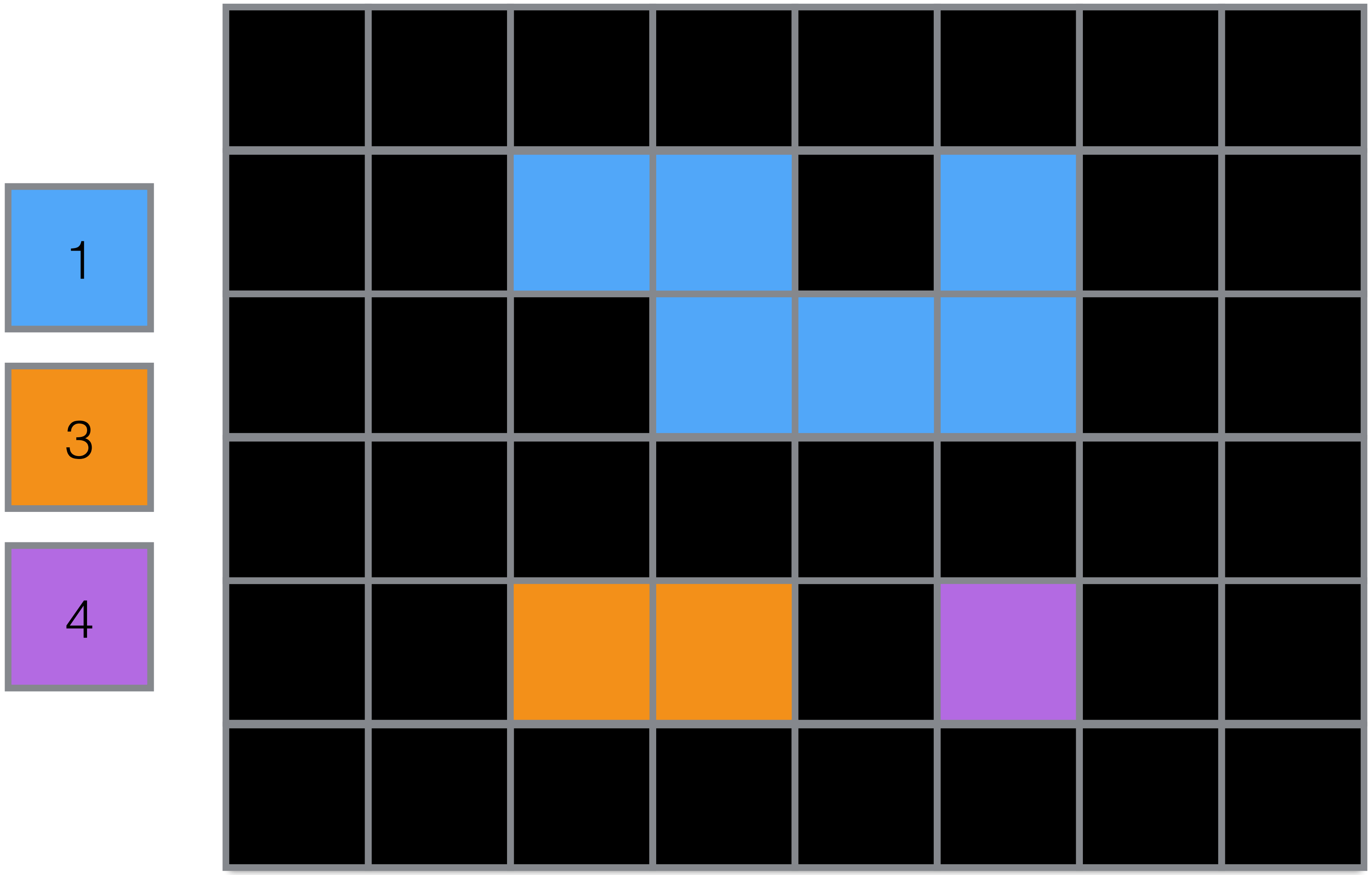
Thresholding: Connected Components



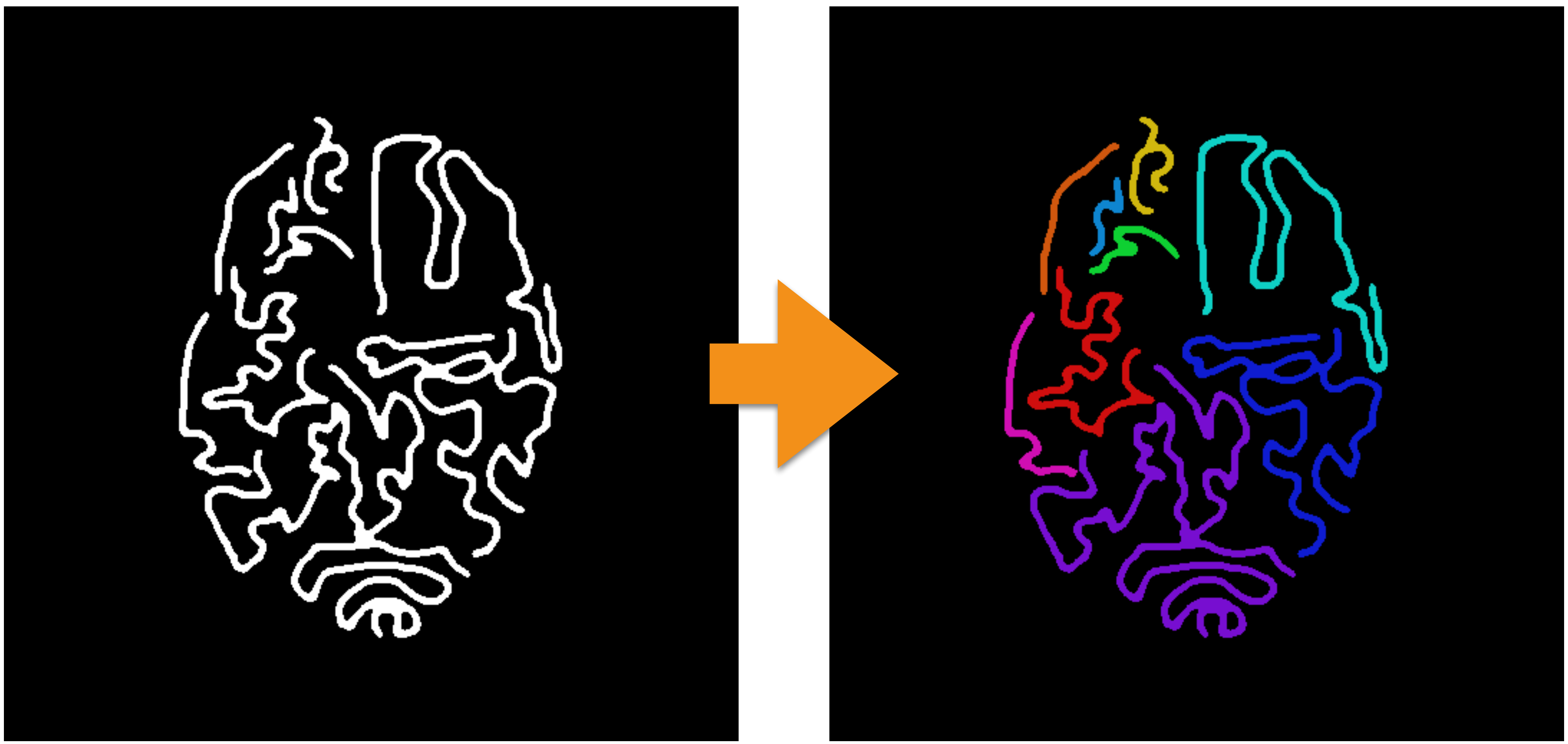
Thresholding: Connected Components



Thresholding: Connected Components



Thresholding:
Connected Components Example



Thresholding

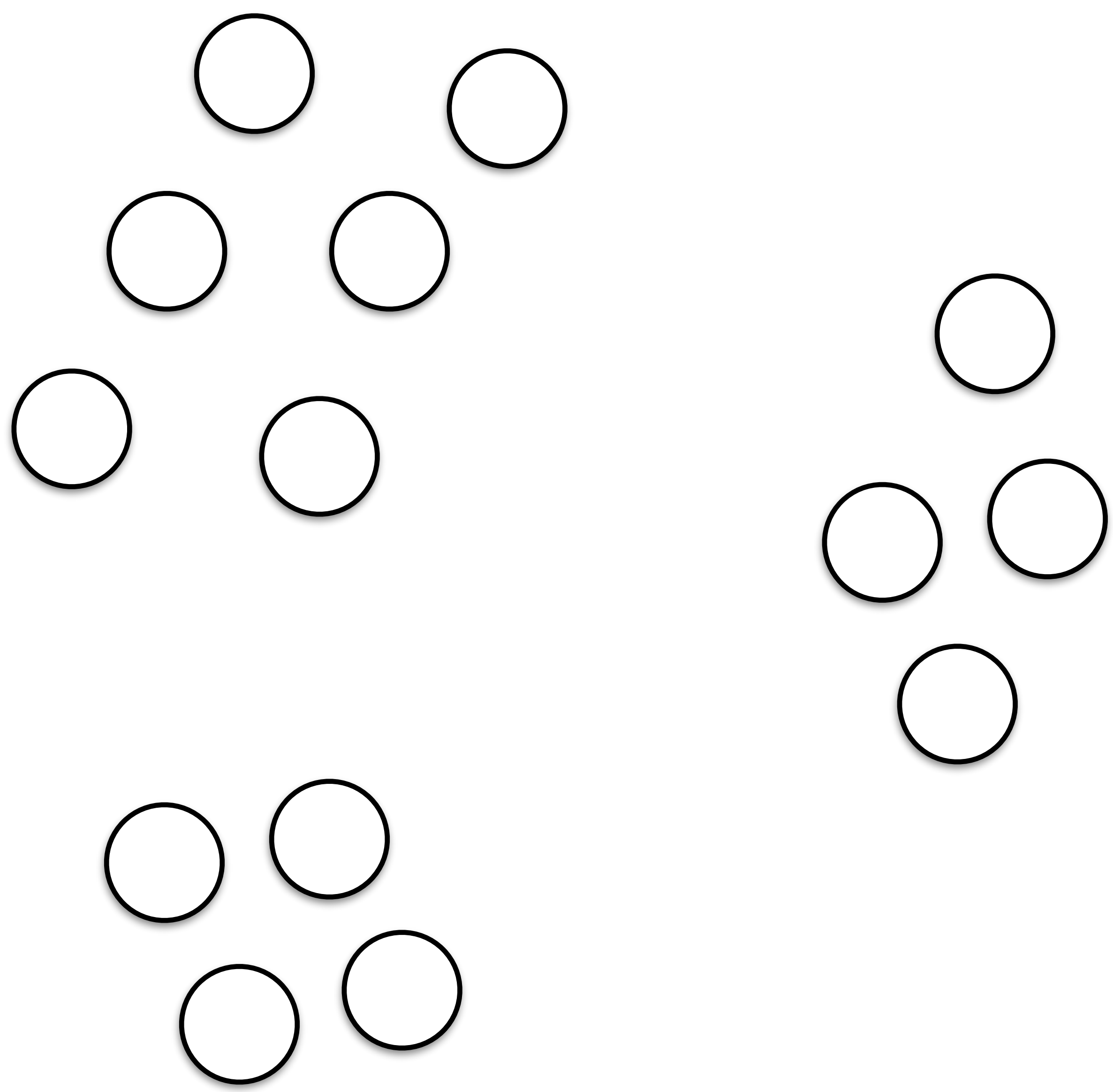
- It works if each object has a unique intensity value/ color; this is a very limiting constraint!
  - However, it could be used as a starting point for other algorithms.
- The user needs to set the threshold!
  - The I_t value for each class may be inferred by analyzing the histogram of the input image.
- Its 3D extension is trivial!

k-Means

k-Means

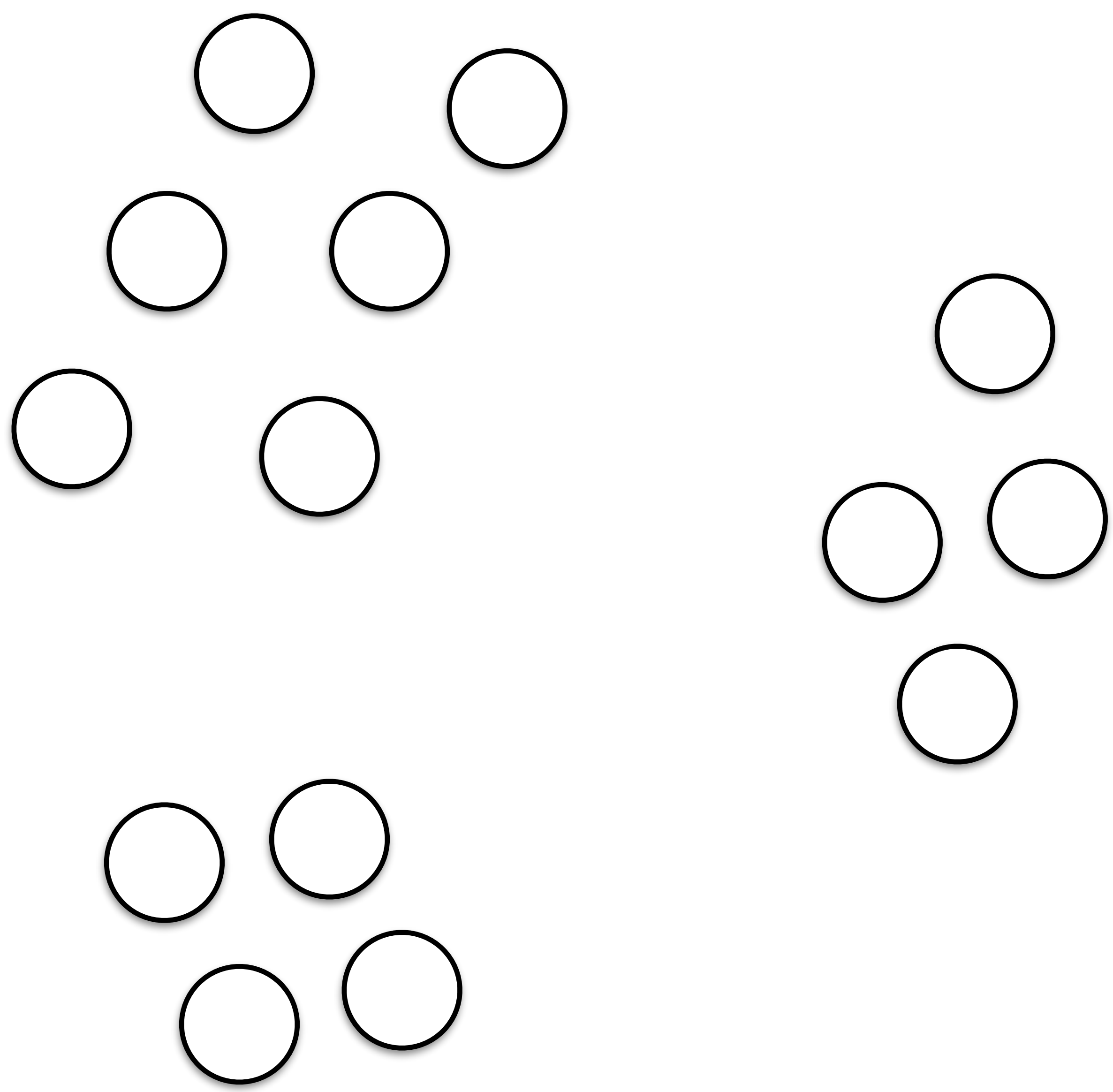
- k-means is a clustering algorithm for clustering n-D vectors/points in an n-D space:
  - A pixel with position (x, y) and intensity I is a 3D vector: $< x, y, I >$
  - A voxel with position (x, y, z) and intensity I is a 4D vector: $< x, y, z, I >$
- Let's assume we have k objects in the image.
- So we have to determine k-clusters.

k-Means: How it Works



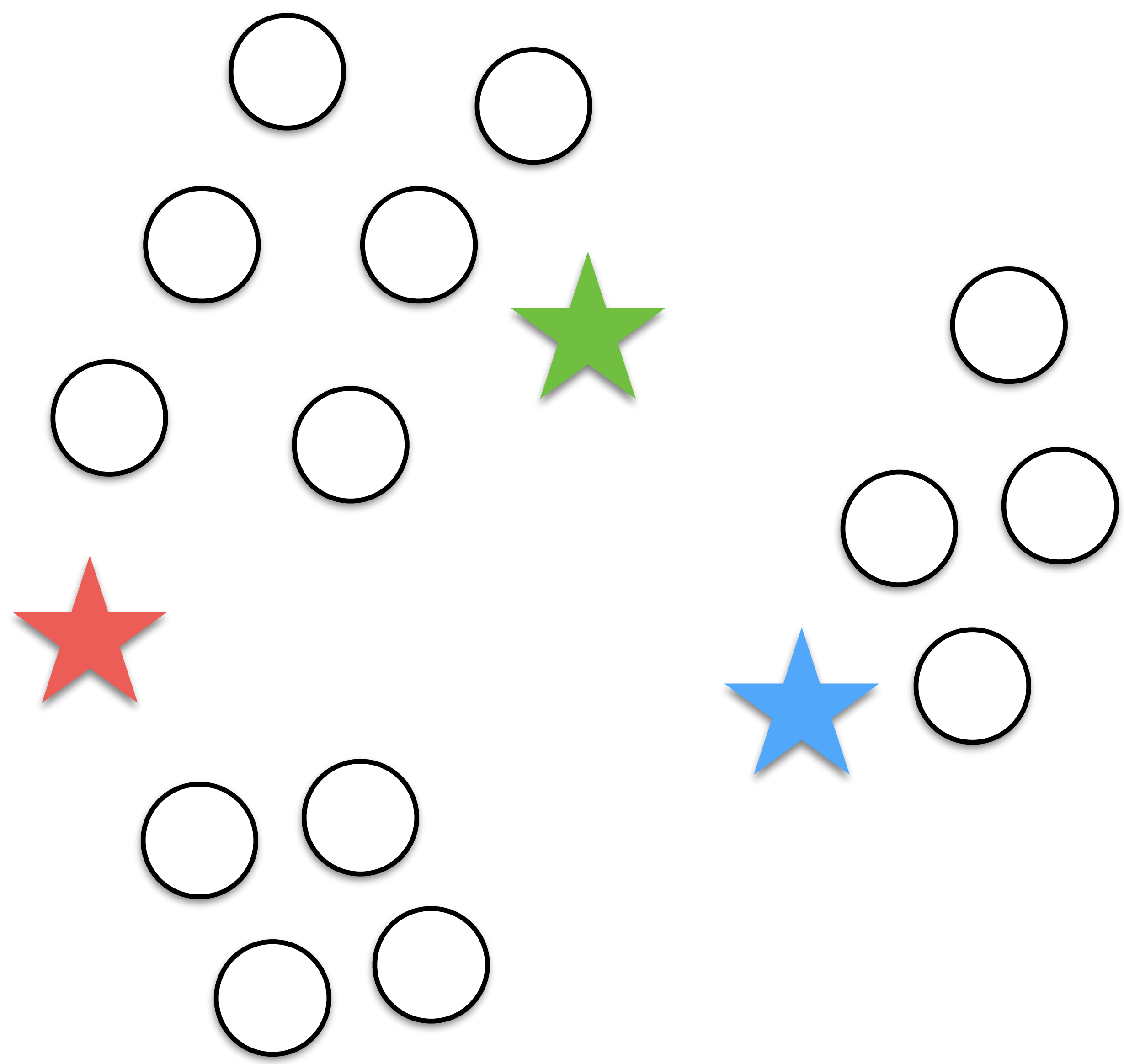
- A 2D example

k-Means: Initialization



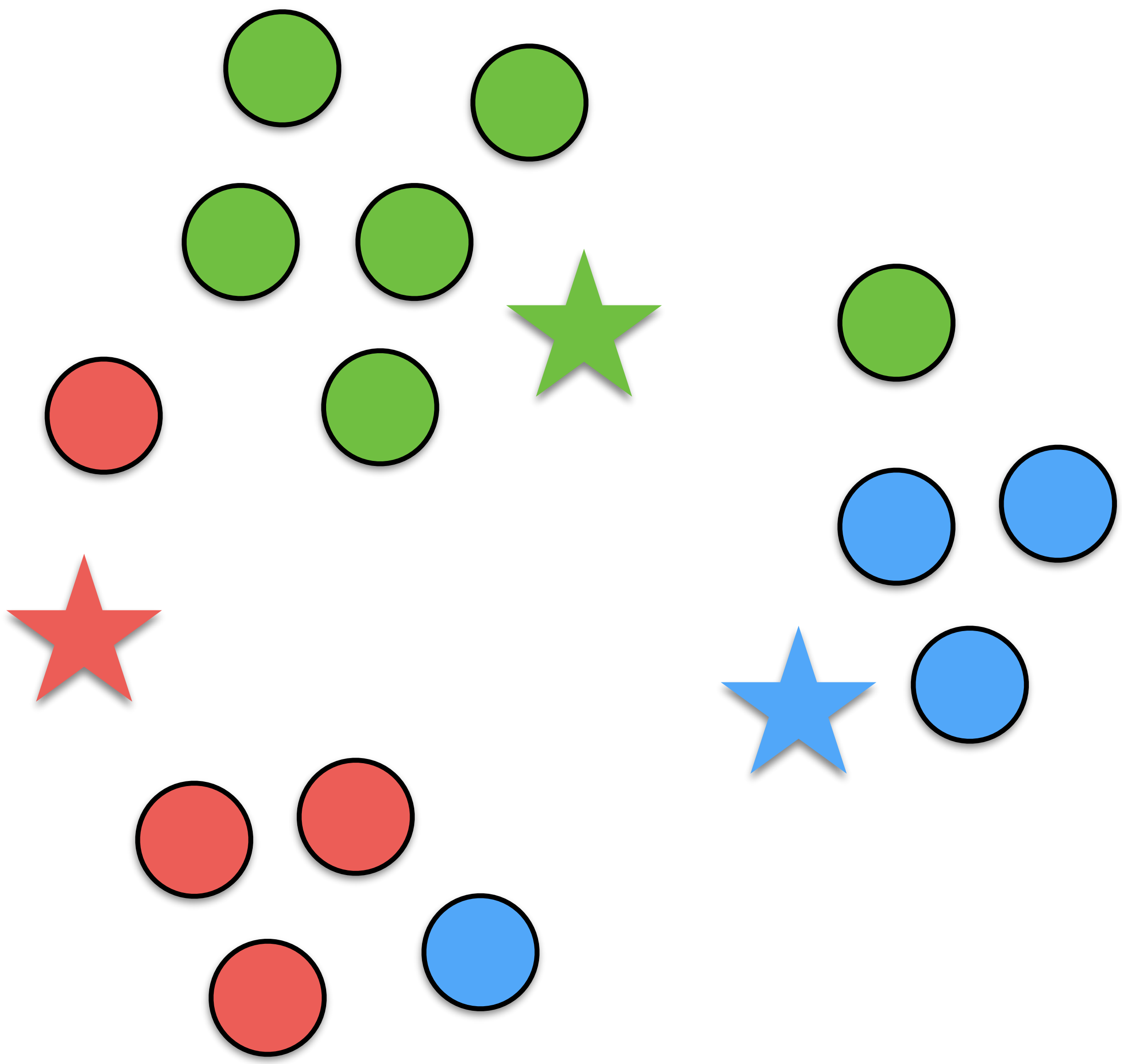
- Let's assume $k = 3$
- We make a random guess on the k-centroids; i.e., the stars.

k-Means: Initialization



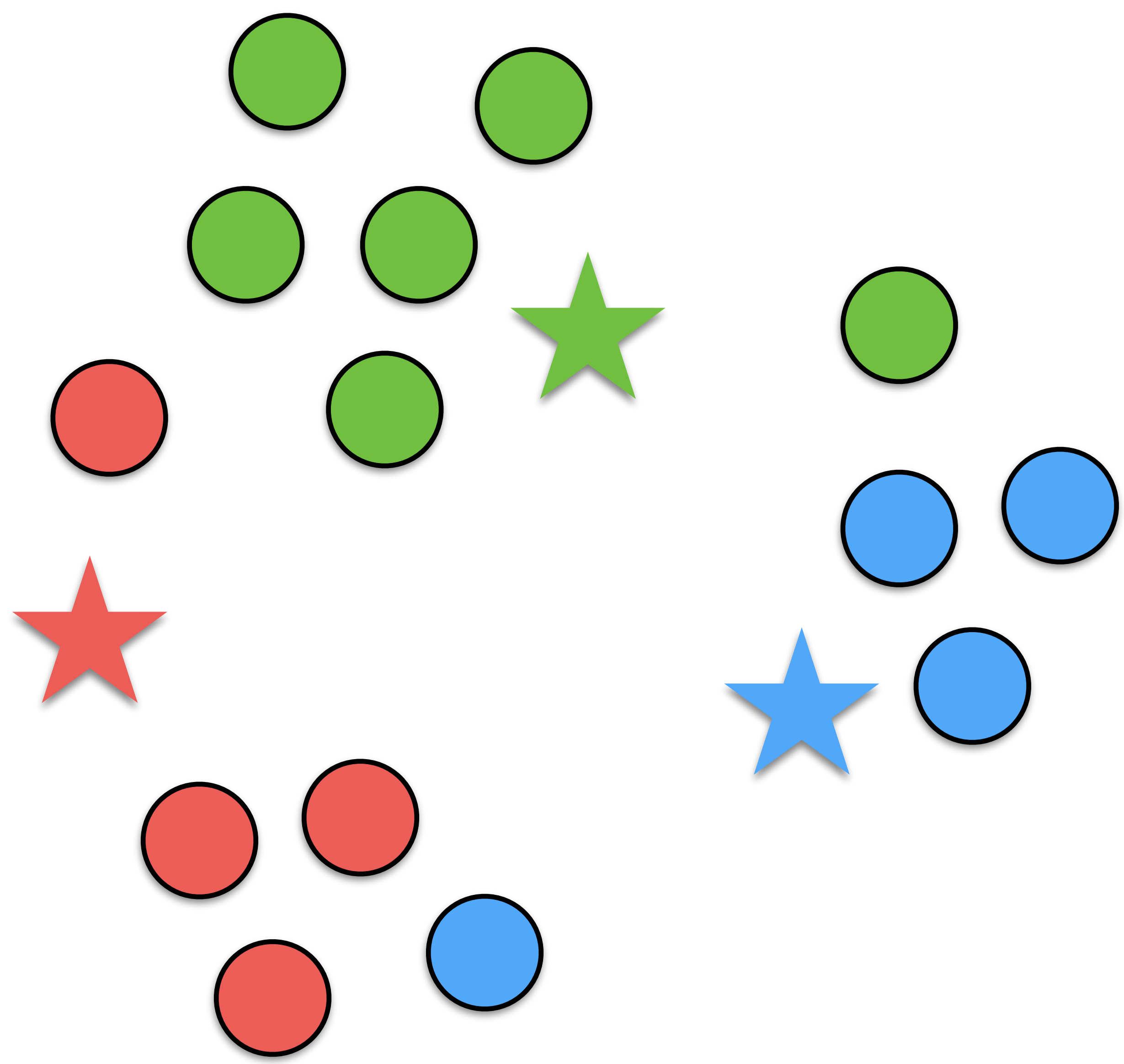
- Let's assume $k = 3$
- We make a random guess on the k-centroids; i.e., the stars.

k-Means: Iteration



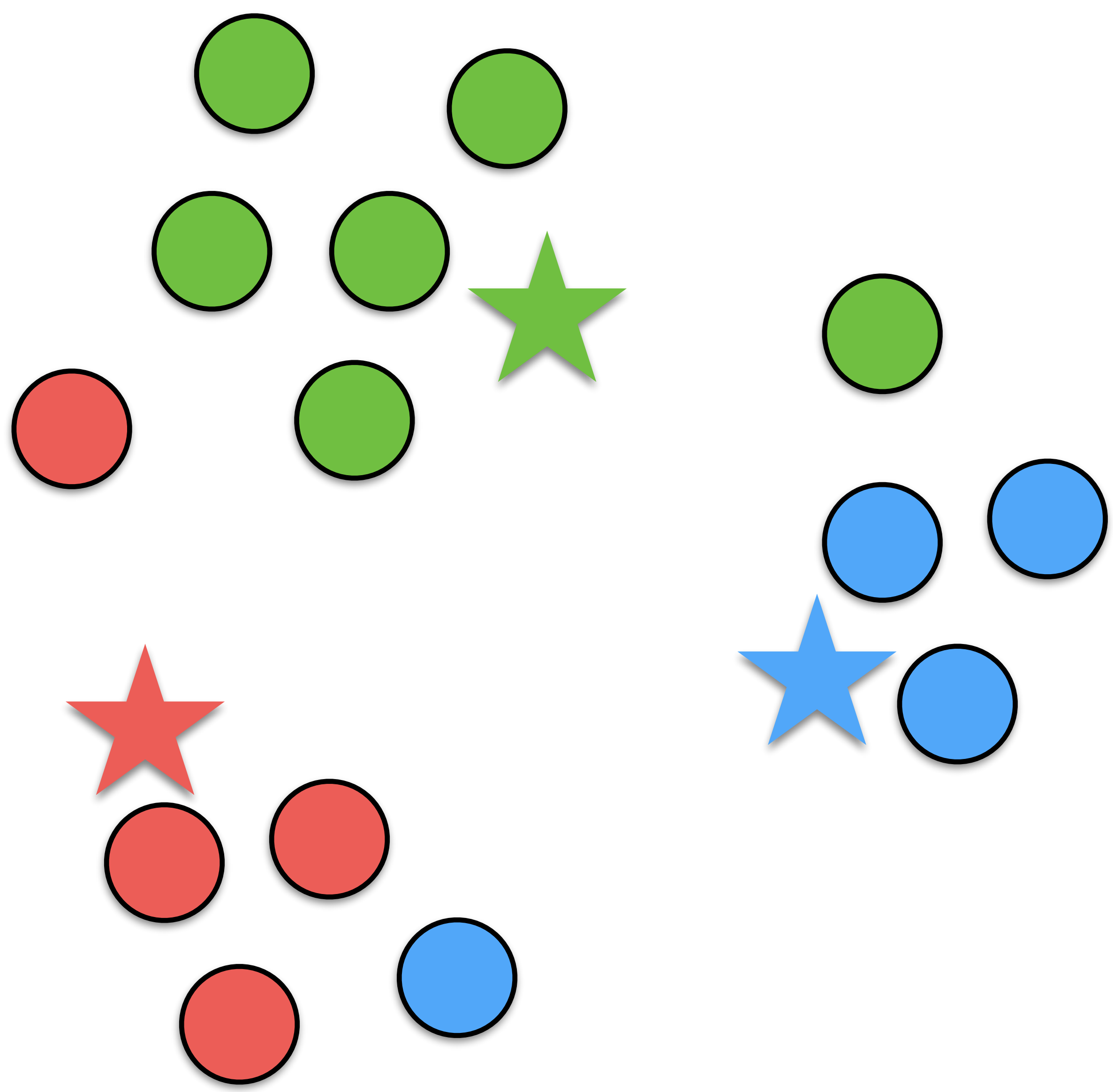
- We now assign a sample to a cluster if the distance (L1, L2, etc.), between a centroid is the minimum.

k-Means: Iteration



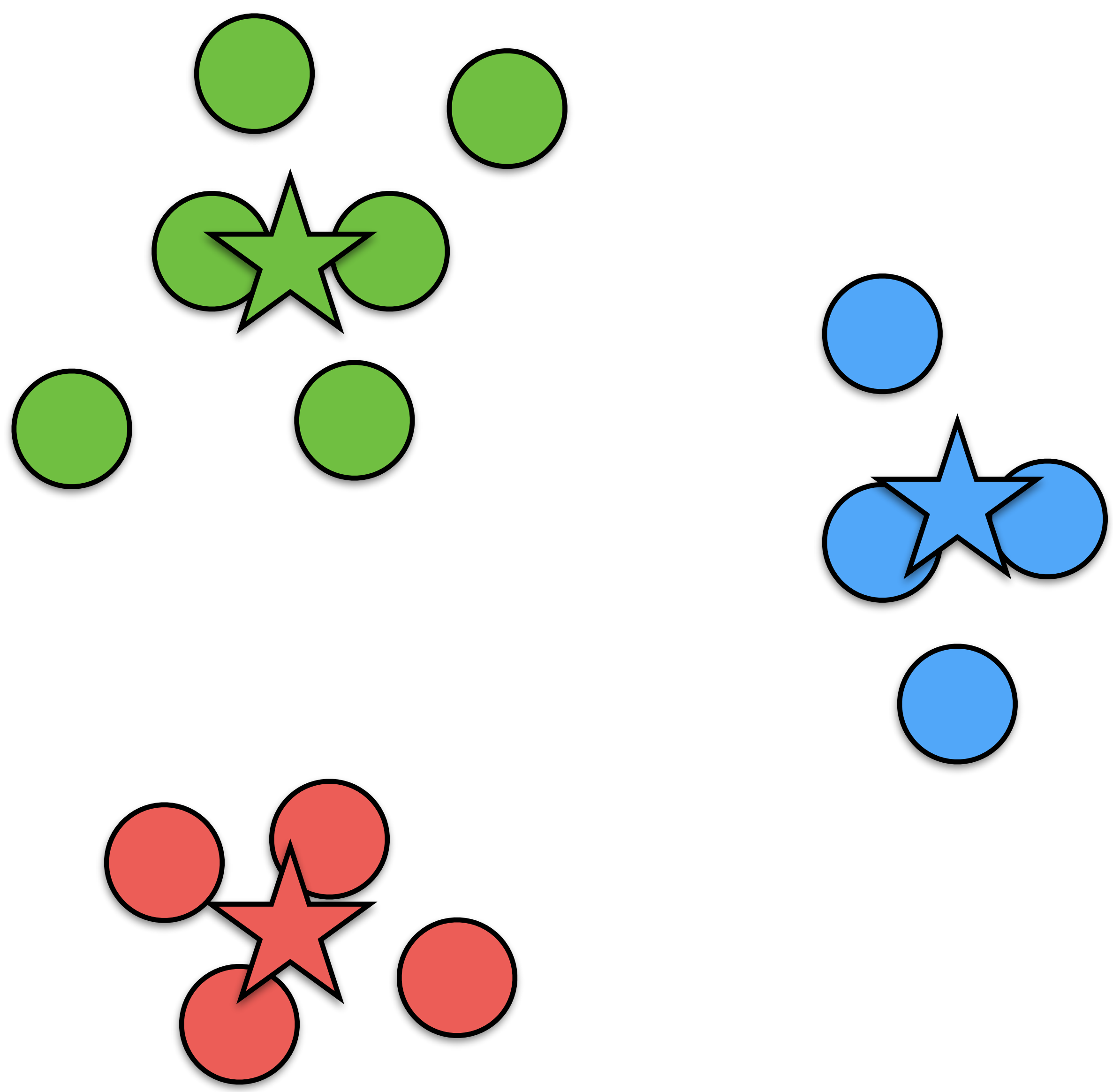
- We re-compute the centroid as the mean of samples of a cluster.

k-Means: Iteration



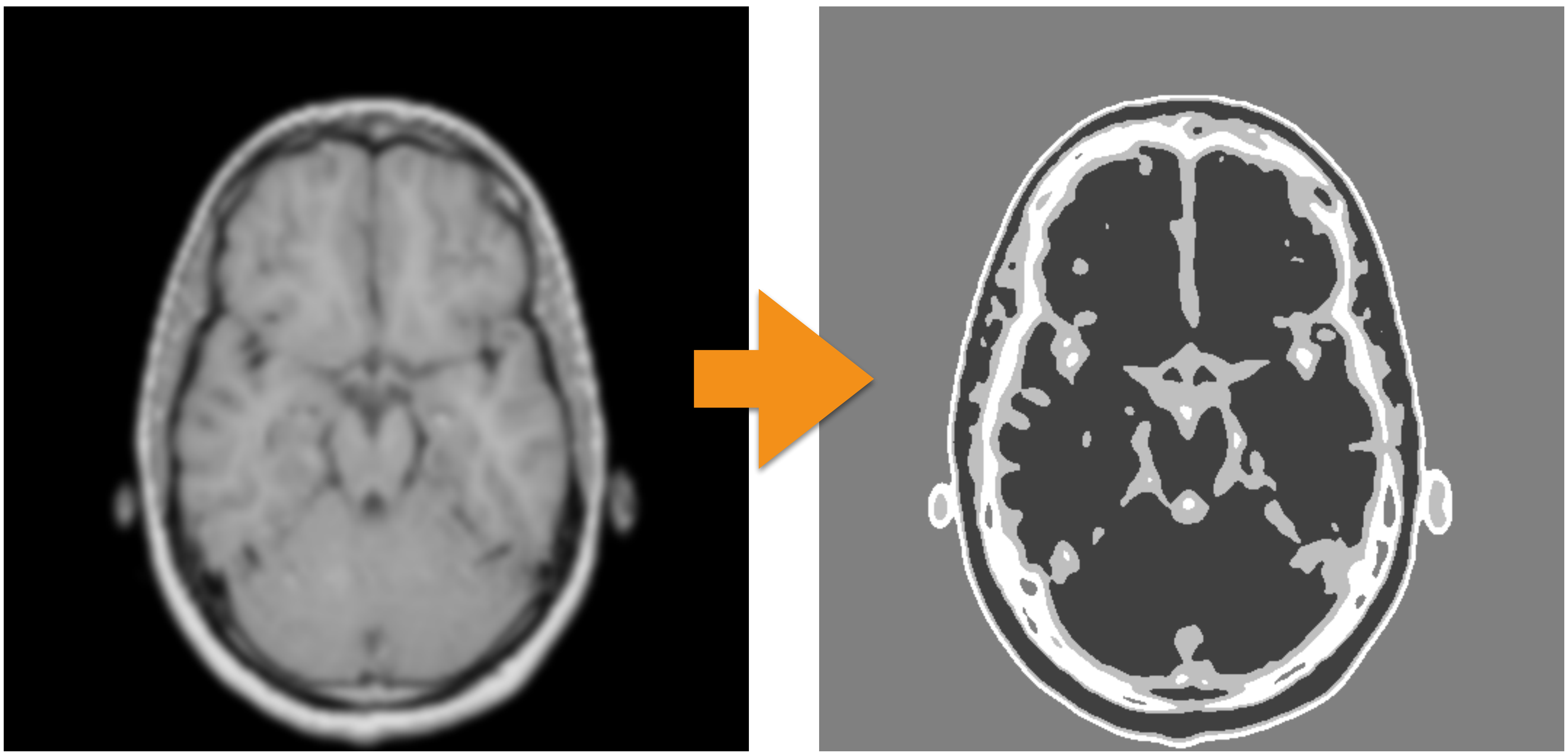
- We repeat the process until convergence (no more changes) or after m iterations.

k-Means: Iteration

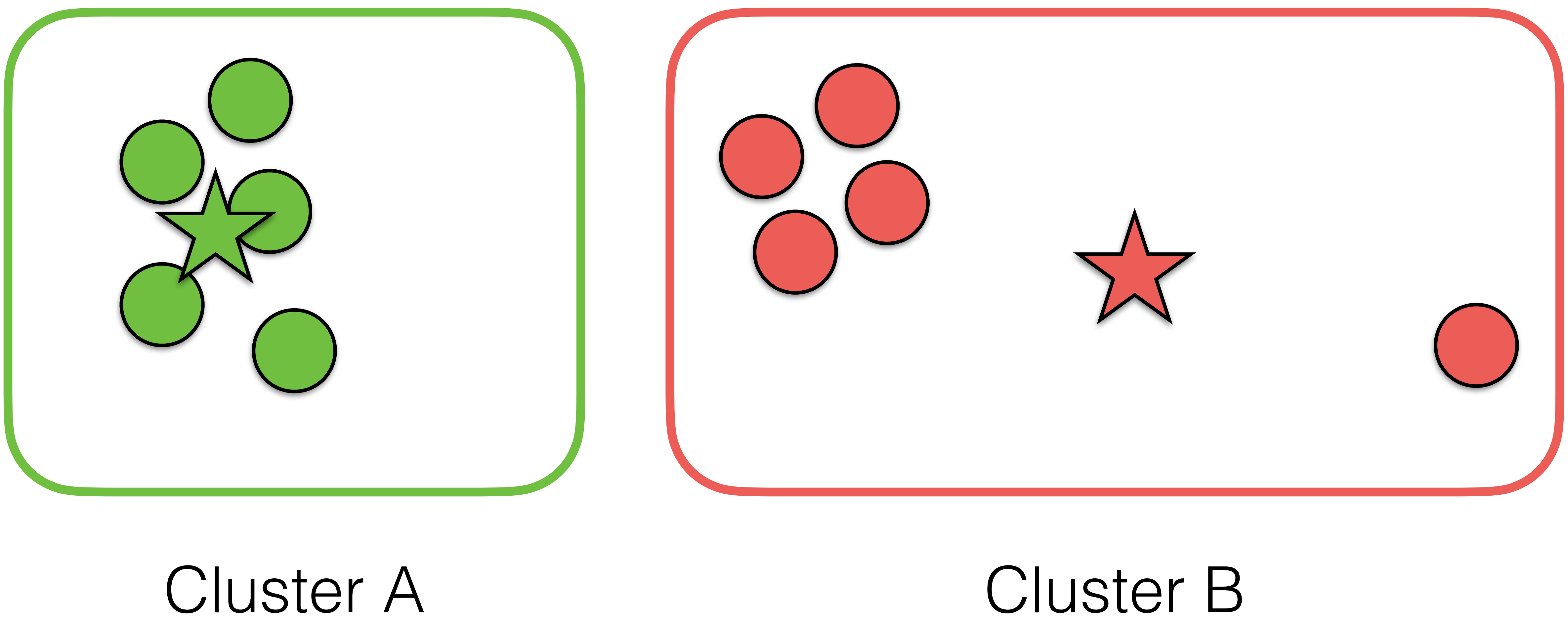


- We repeat the process until convergence (no more changes) or after m iterations.

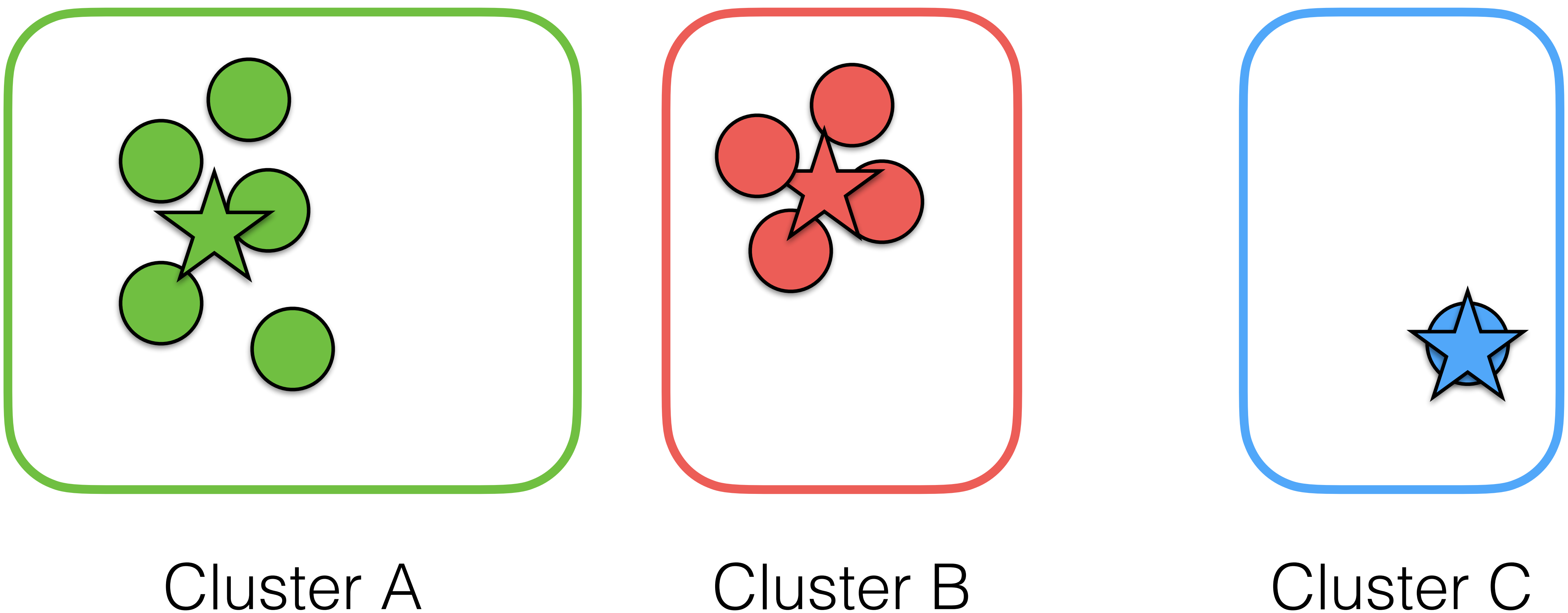
k-Means Example



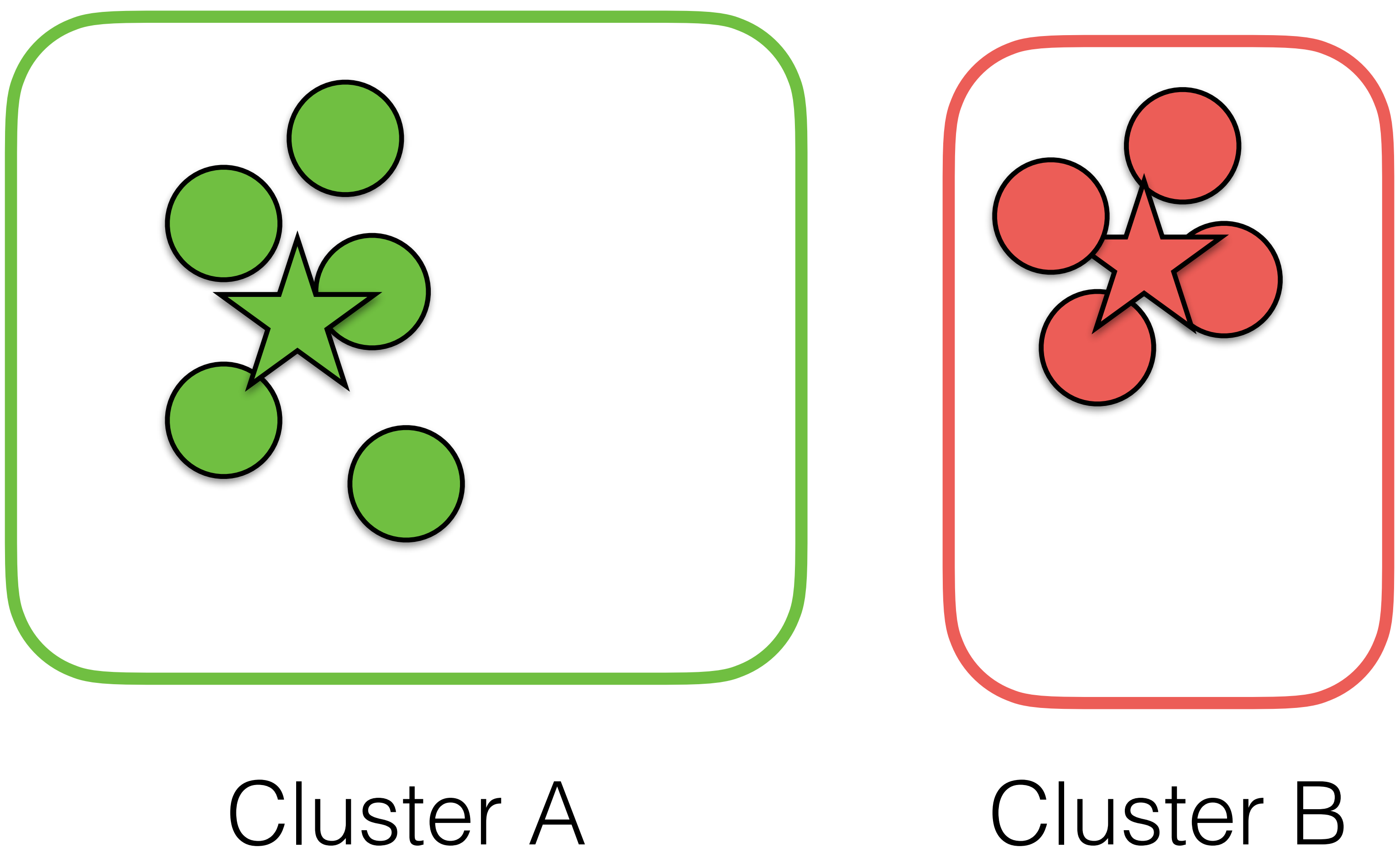
k-Means: Outliers



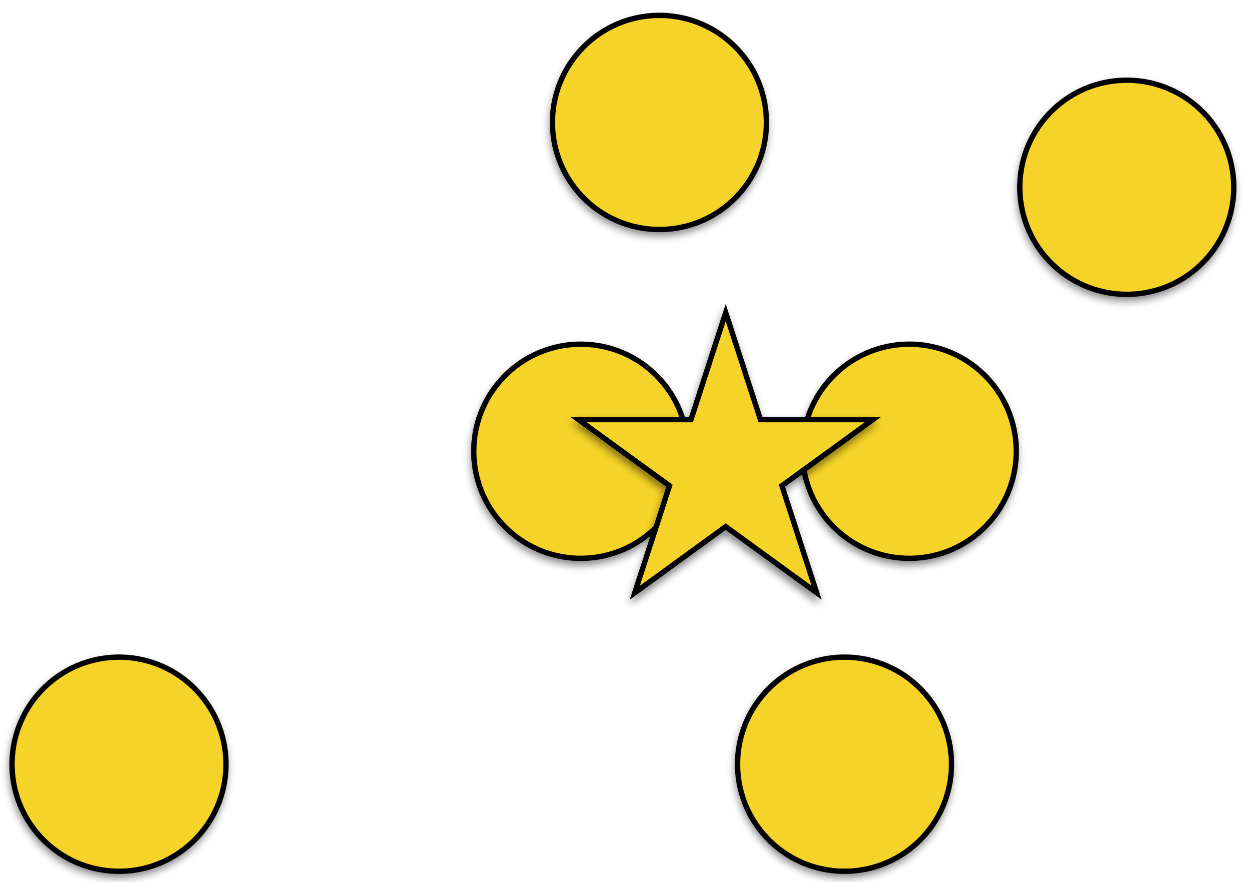
k-Means: Outliers
Solution 1



k-Means: Outliers
Solution 2

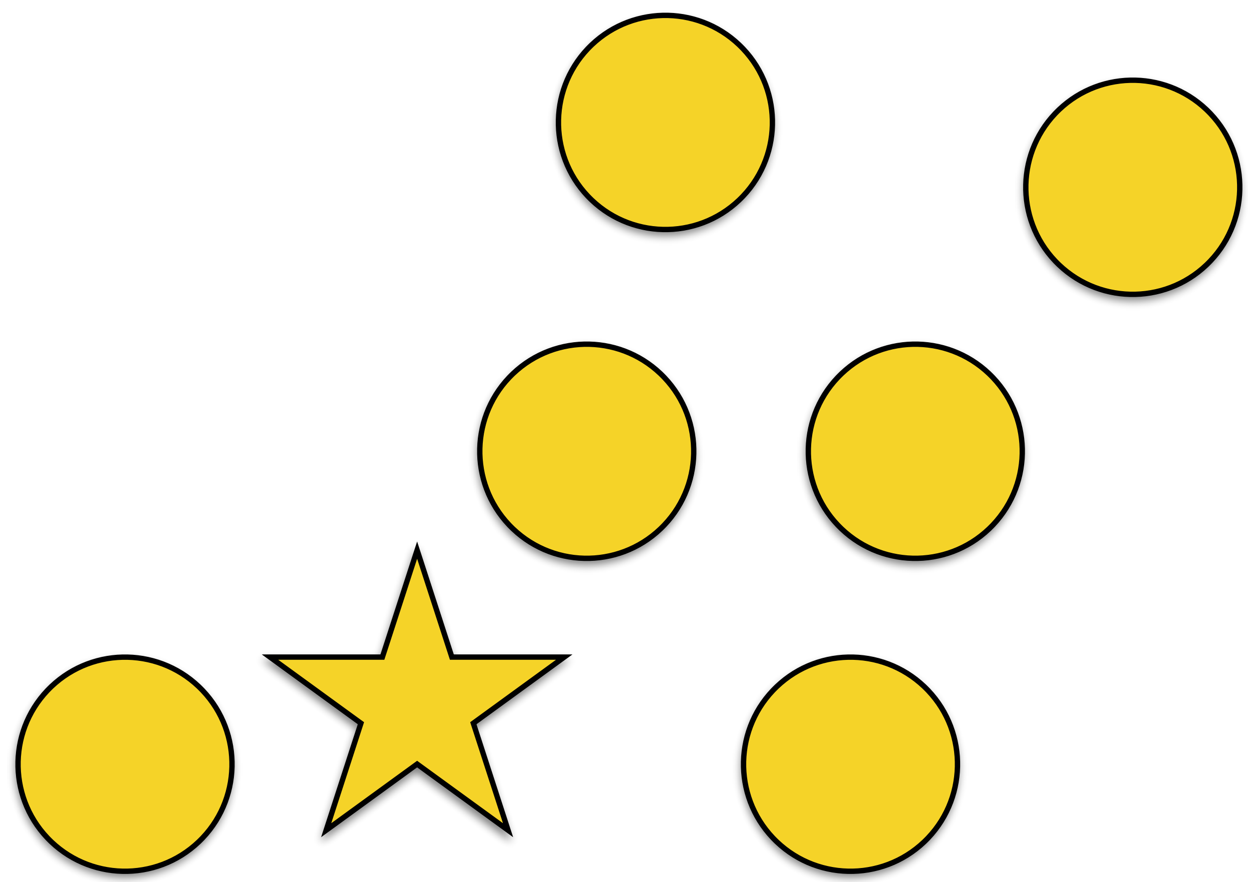


k-Means: Oscillation



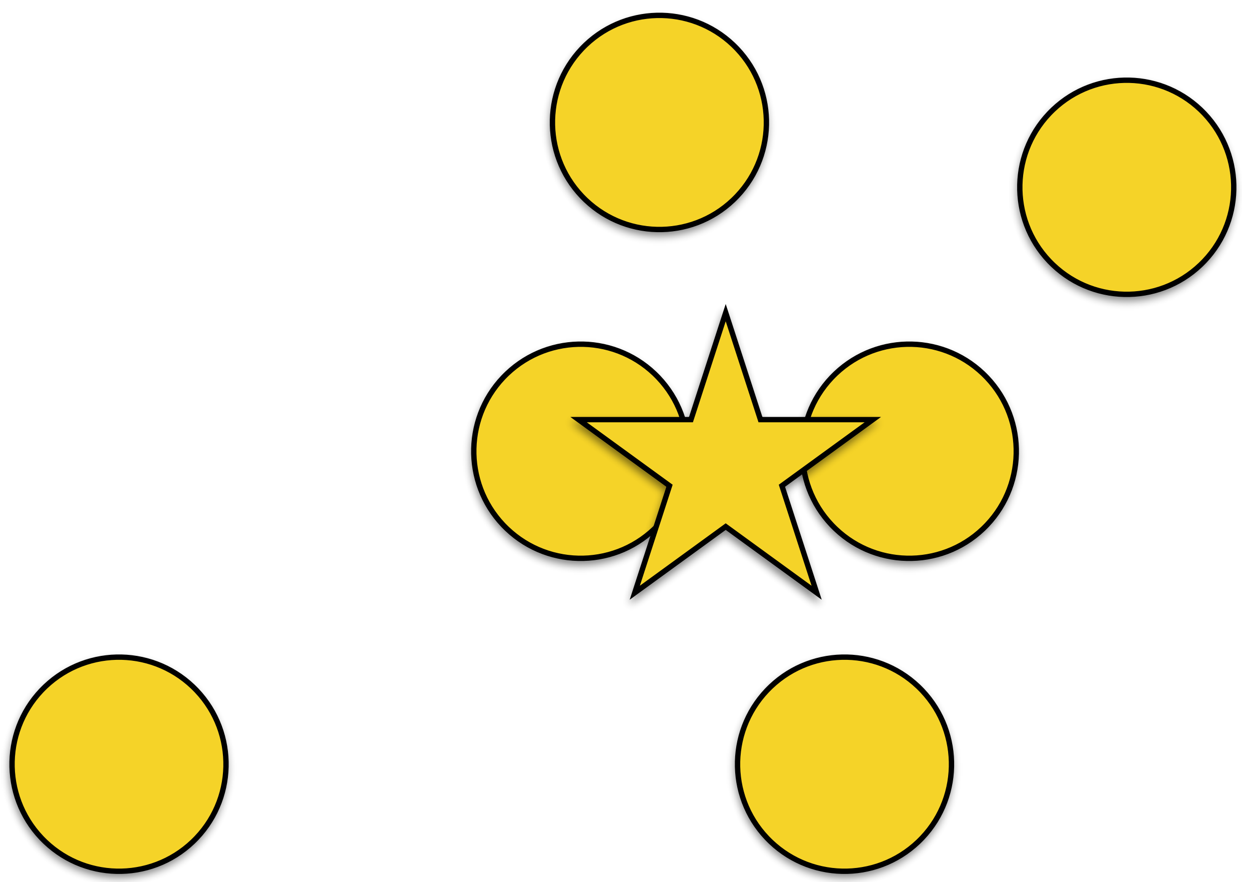
Even Iteration

k-Means: Oscillation



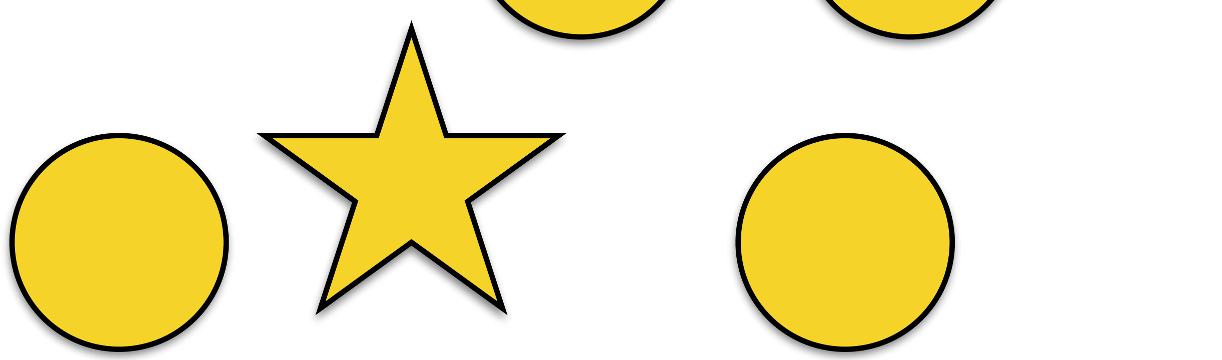
Odd Iteration

k-Means: Oscillation



Even Iteration

k-Means: Oscillation



Odd Iteration

k-Means: Advantages

- The method is fully automatic.
- This works for 2D and 3D volumes.
- This can “understand” neighbors in an implicit way.

k-Means: Disadvantages

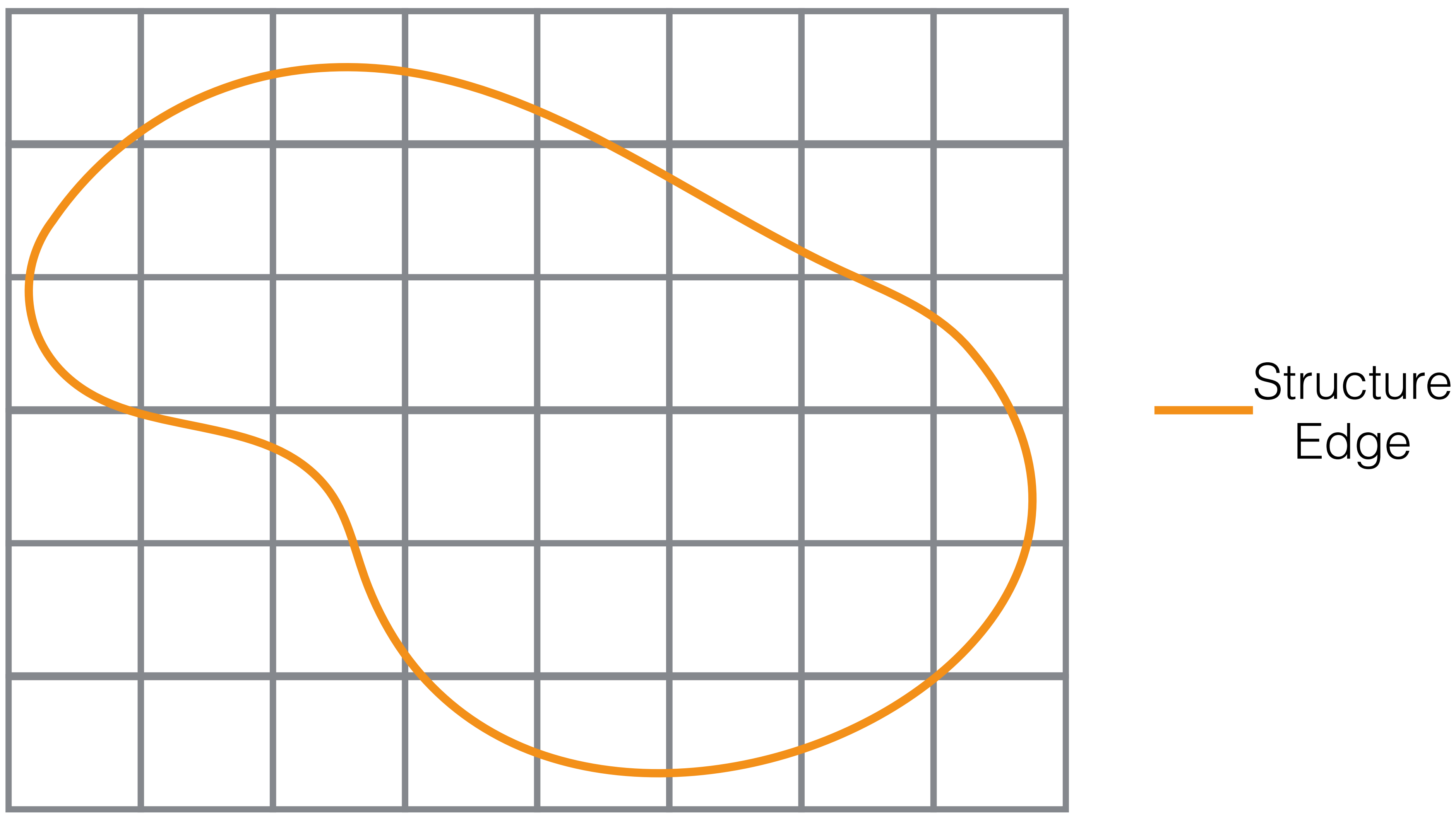
- We need to know how many objects (including the background) are in the image:
  - We may run k-means multiple times until a certain criterion is met; e.g., reaching the 80% of percentage of explained variance.
- Outliers:
  - better initialization (sampling).
- The method **may not** converge.
- We need to set a **maximum** number of iterations.

Region Growing

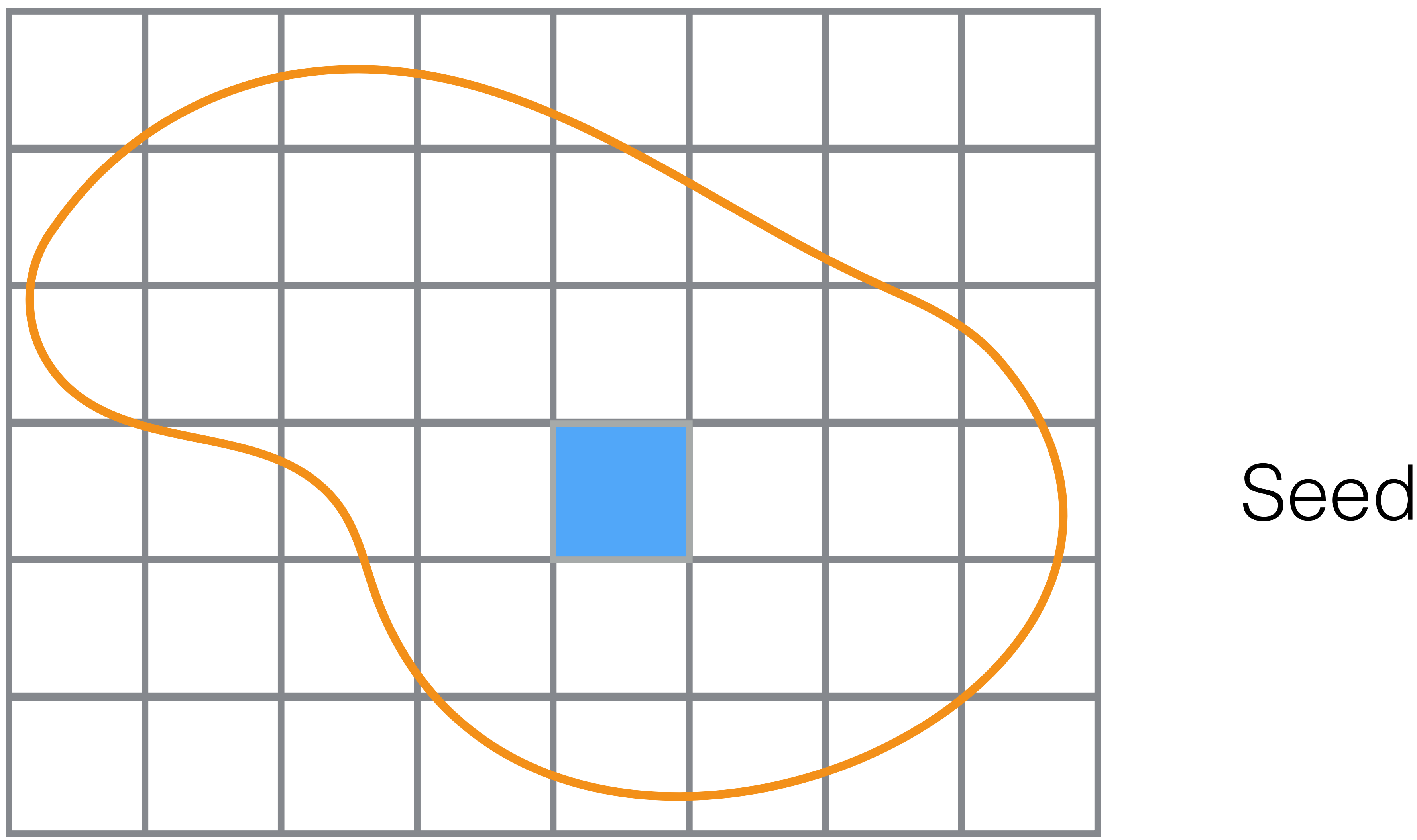
Region Growing

- This algorithms expands a painted initial mask until it reaches strong edges
- Therefore, we need to compute edges first!

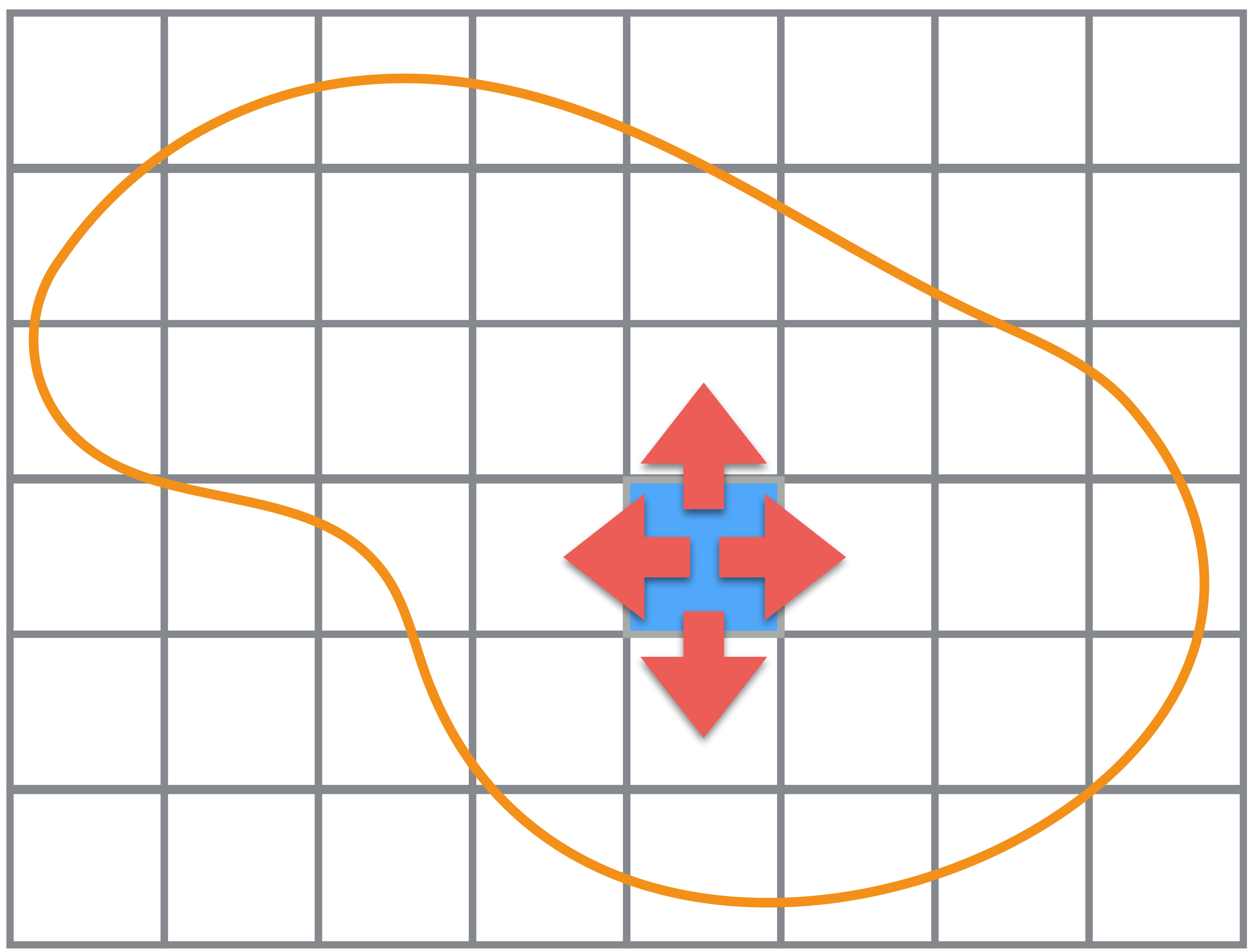
Region Growing



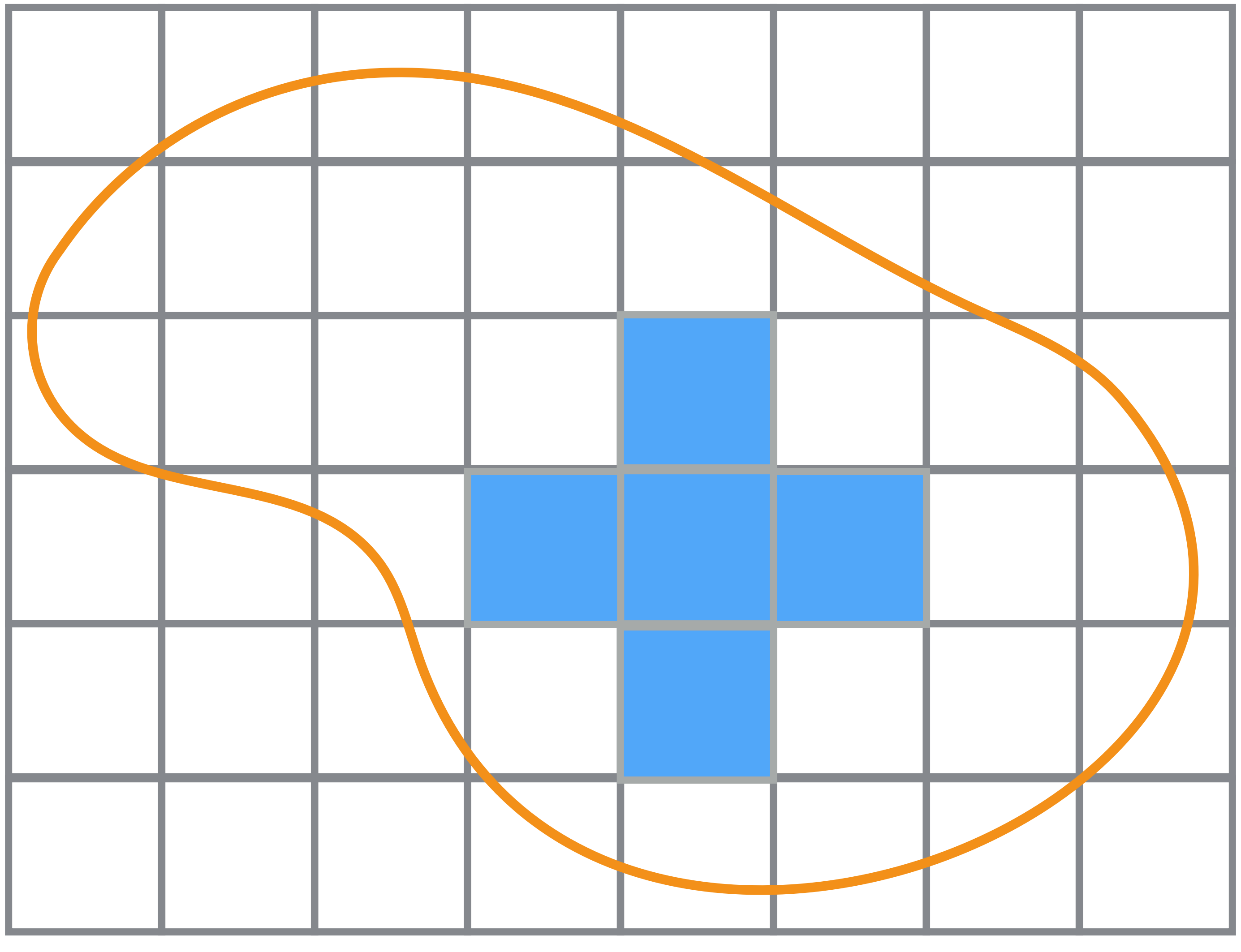
Region Growing



Region Growing

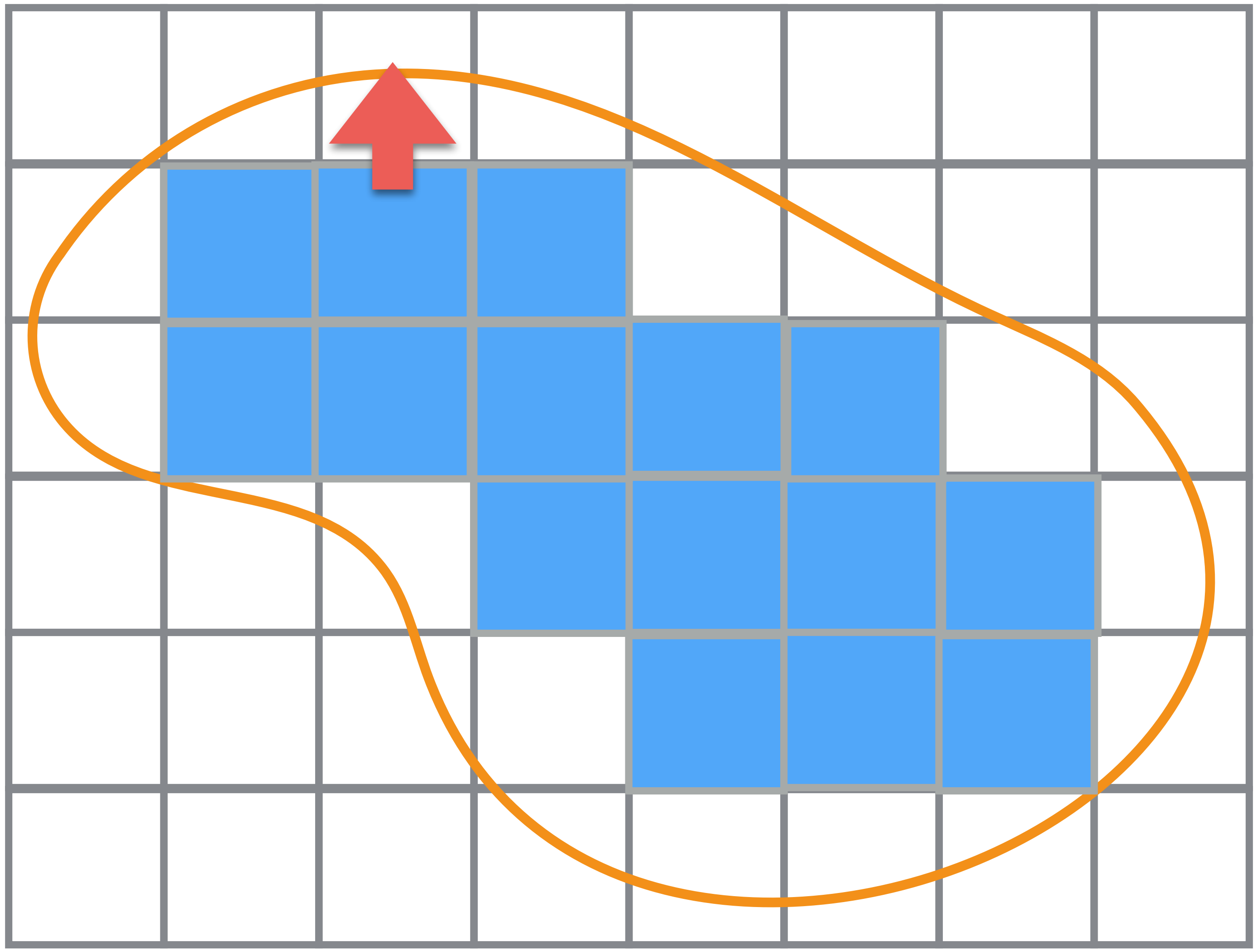


Region Growing

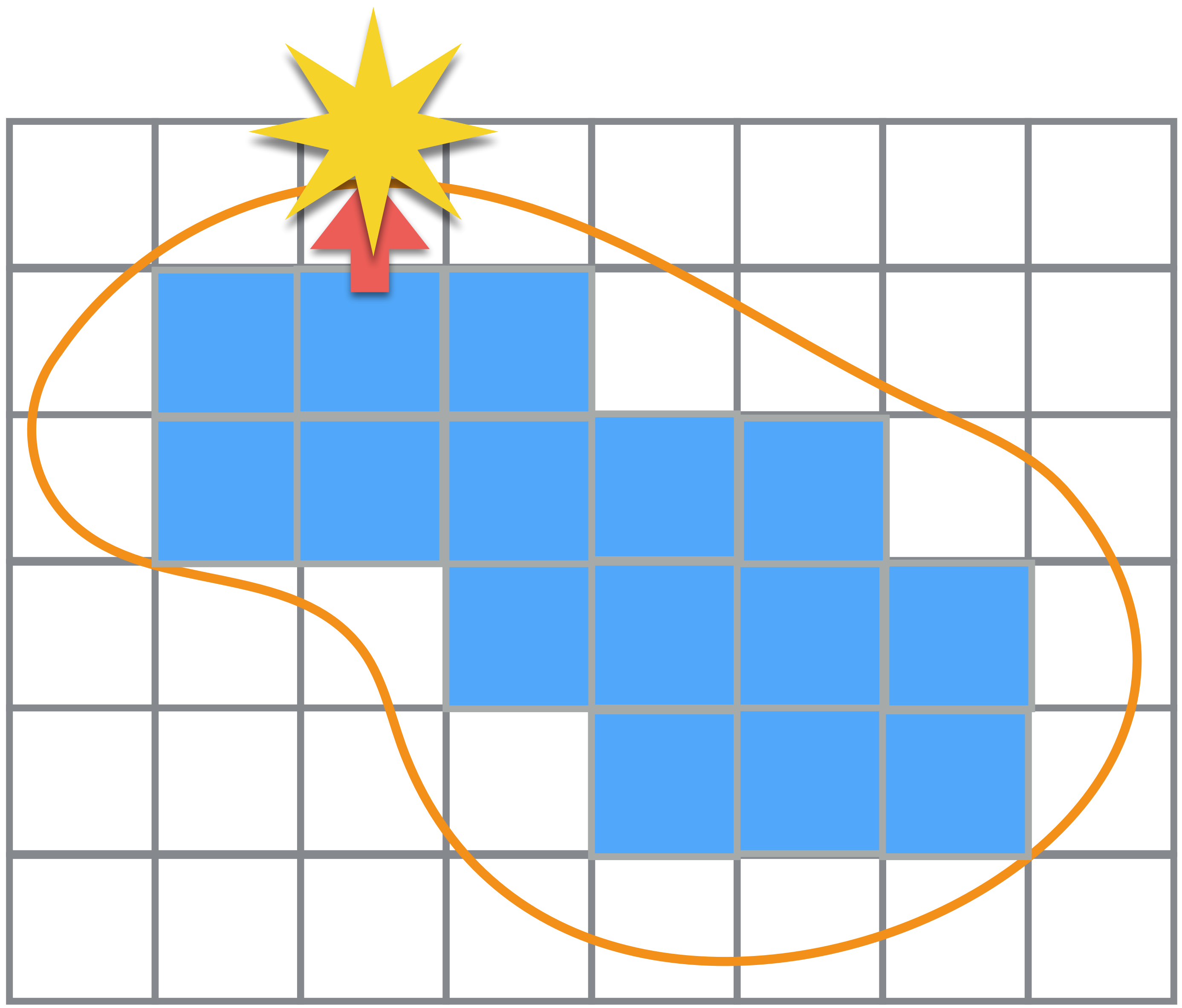


after a while…

Region Growing



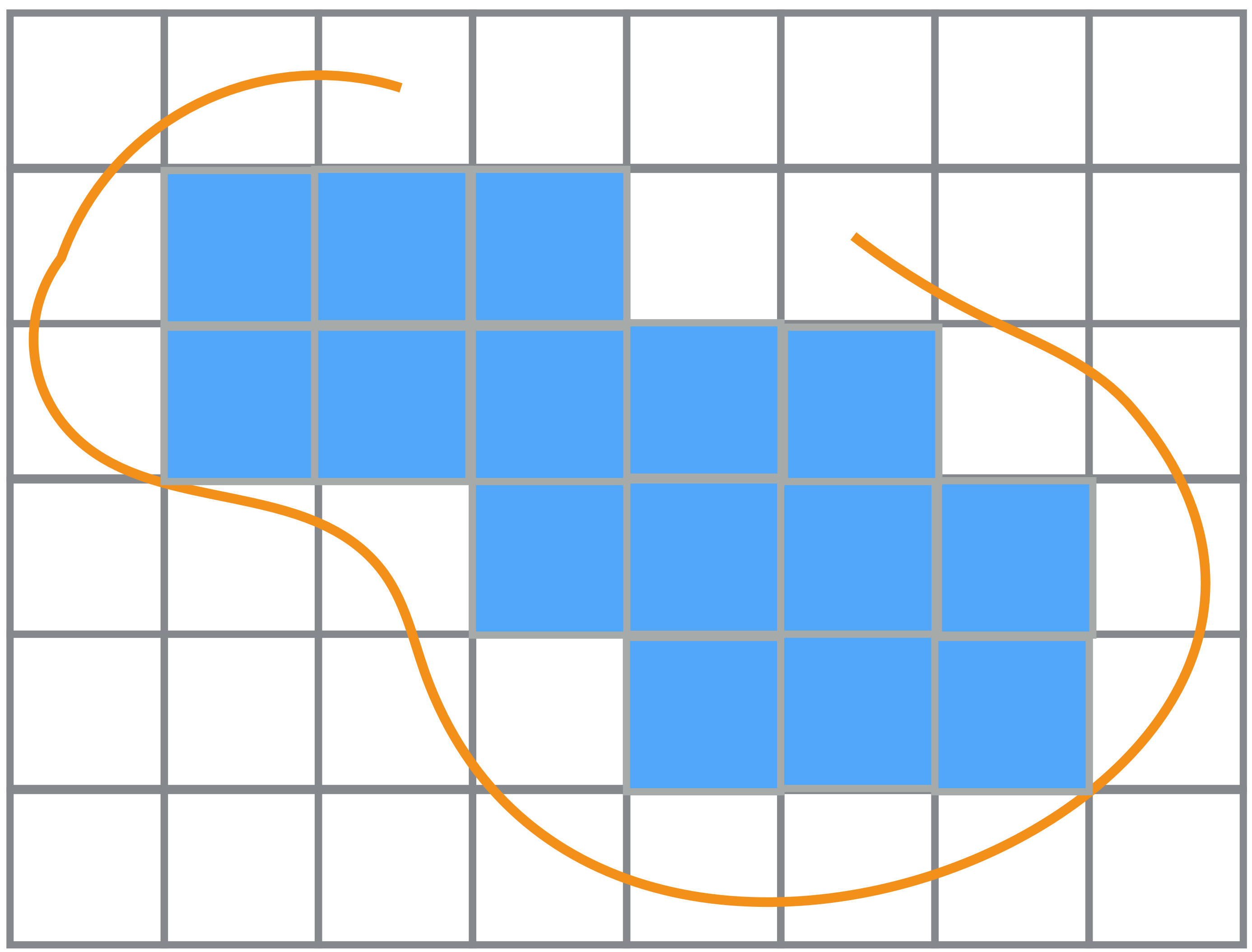
Region Growing



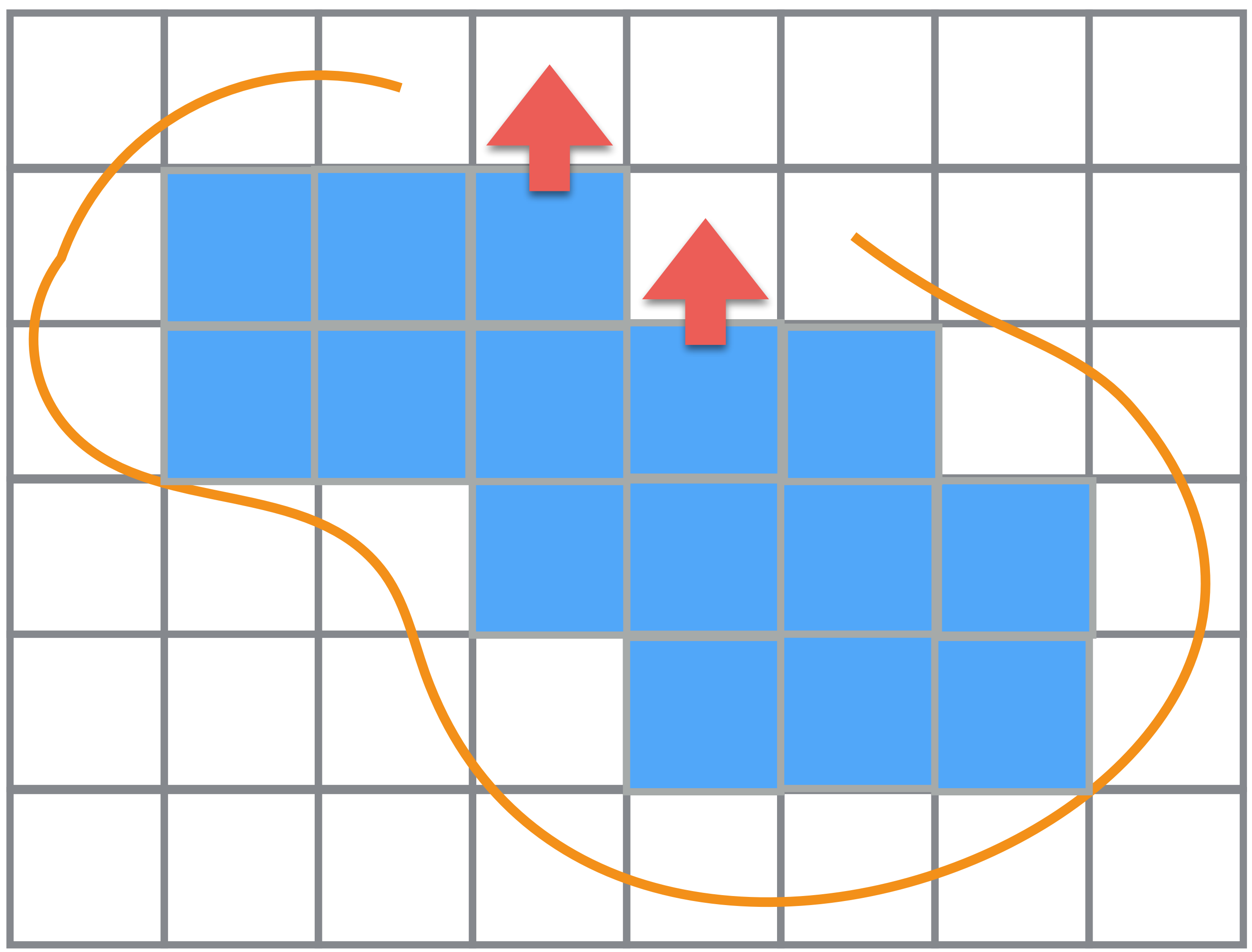
Region Growing

- It is straightforward to extend to 3D!
- This algorithm depends on:
  - The threshold of edge detection.
- It may be slow:
  - From an initial seed, the growing region needs to reach the farthest edge pixel/voxel.
    - Computational complexity is a function of the area/volume of the object we want to segment.

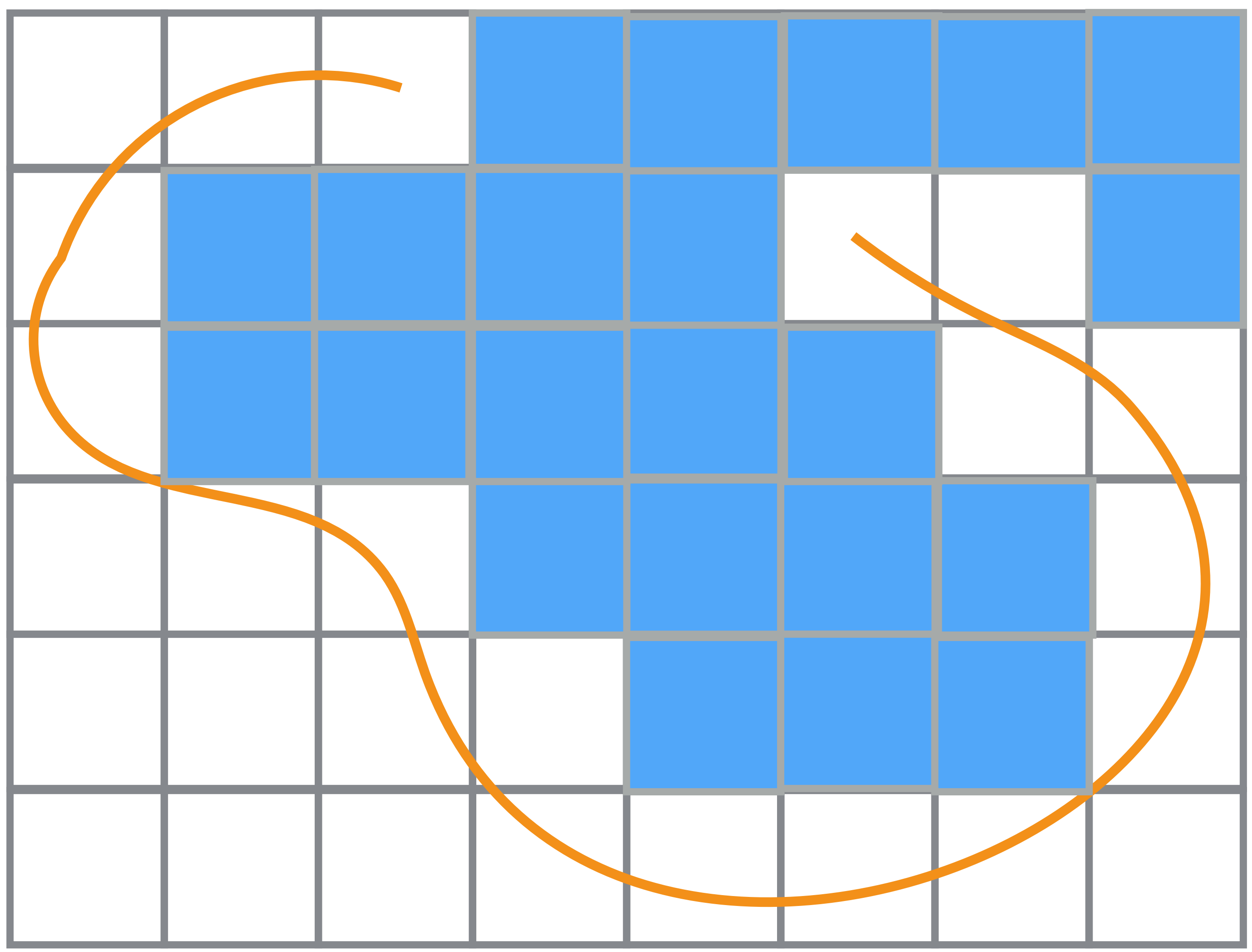
Region Growing: Epic Fail



Region Growing: Epic Fail



Region Growing: Epic Fail



Active Contour Model
aka Snakes

Snakes

- A snake is a parametric curve:

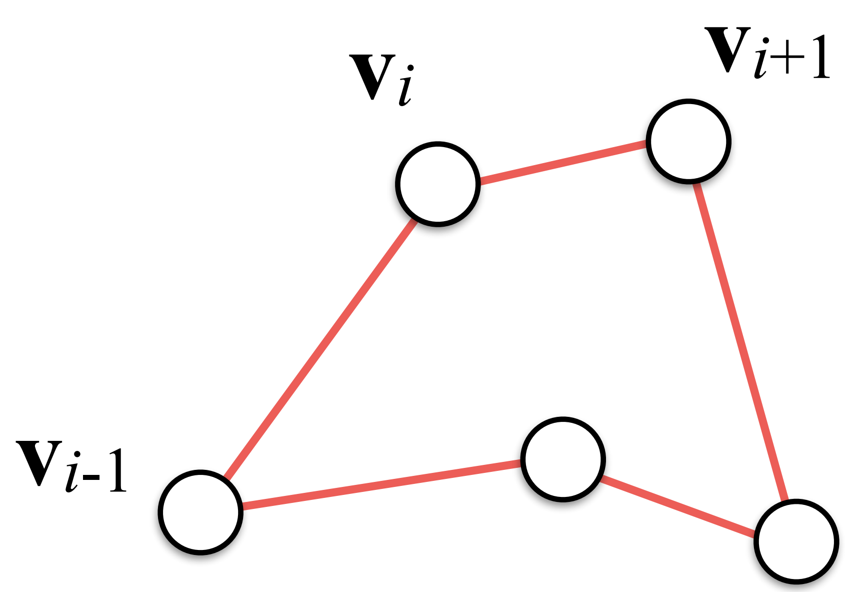
$$\mathbf{v}(t) = [x(t); y(t)] \qquad t \in [0,1]$$

- Typically, it is a spline (original paper), but for sake of simplicity let's assume a piecewise linear curve.

Snakes

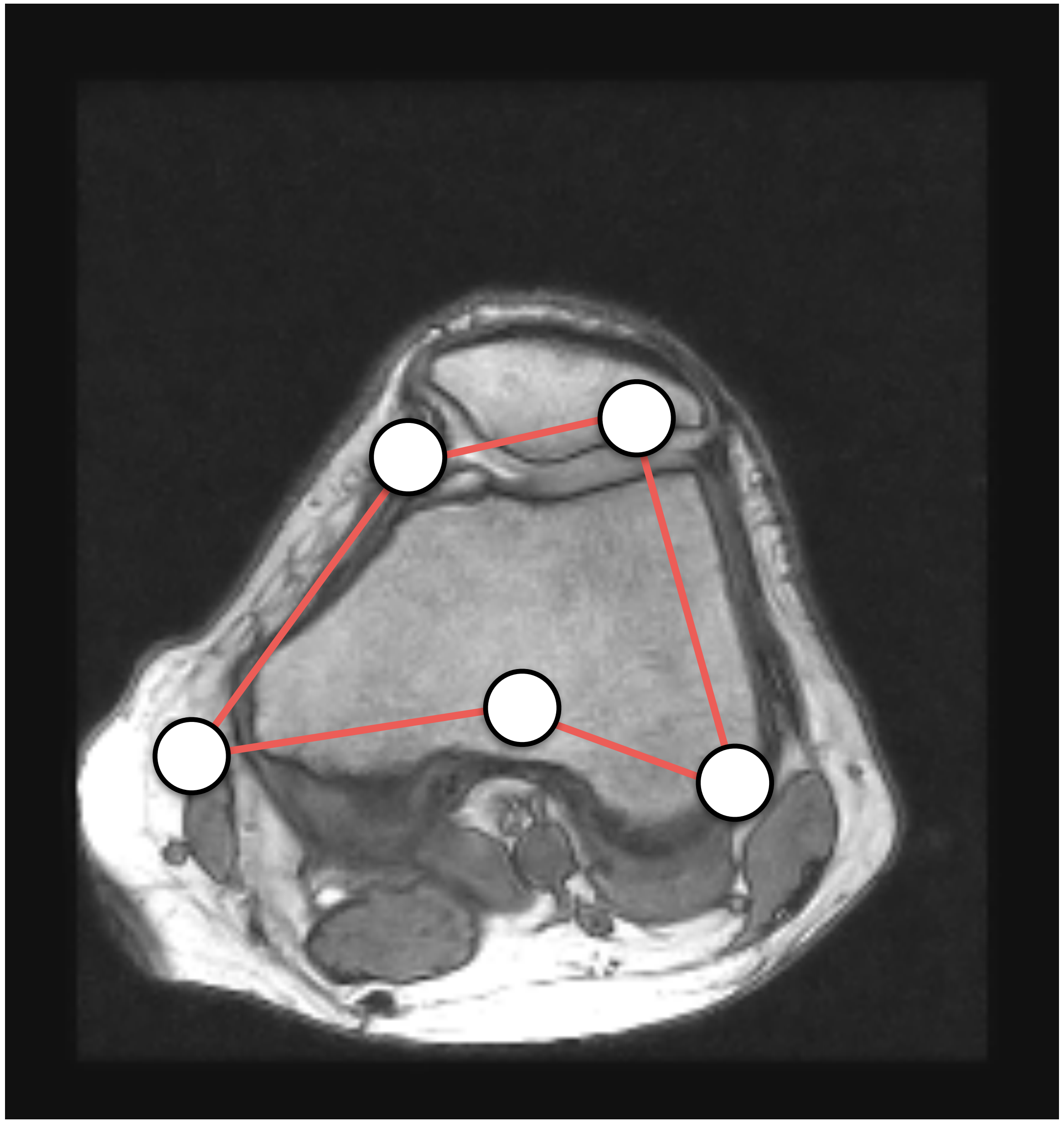
- The snake curve is defined by a set of control point that is defined as:

$$\mathbf{C} = \{\mathbf{v}_i \mid i \in [1,n]\} \qquad \mathbf{v}_i = [x_i, y_i]$$



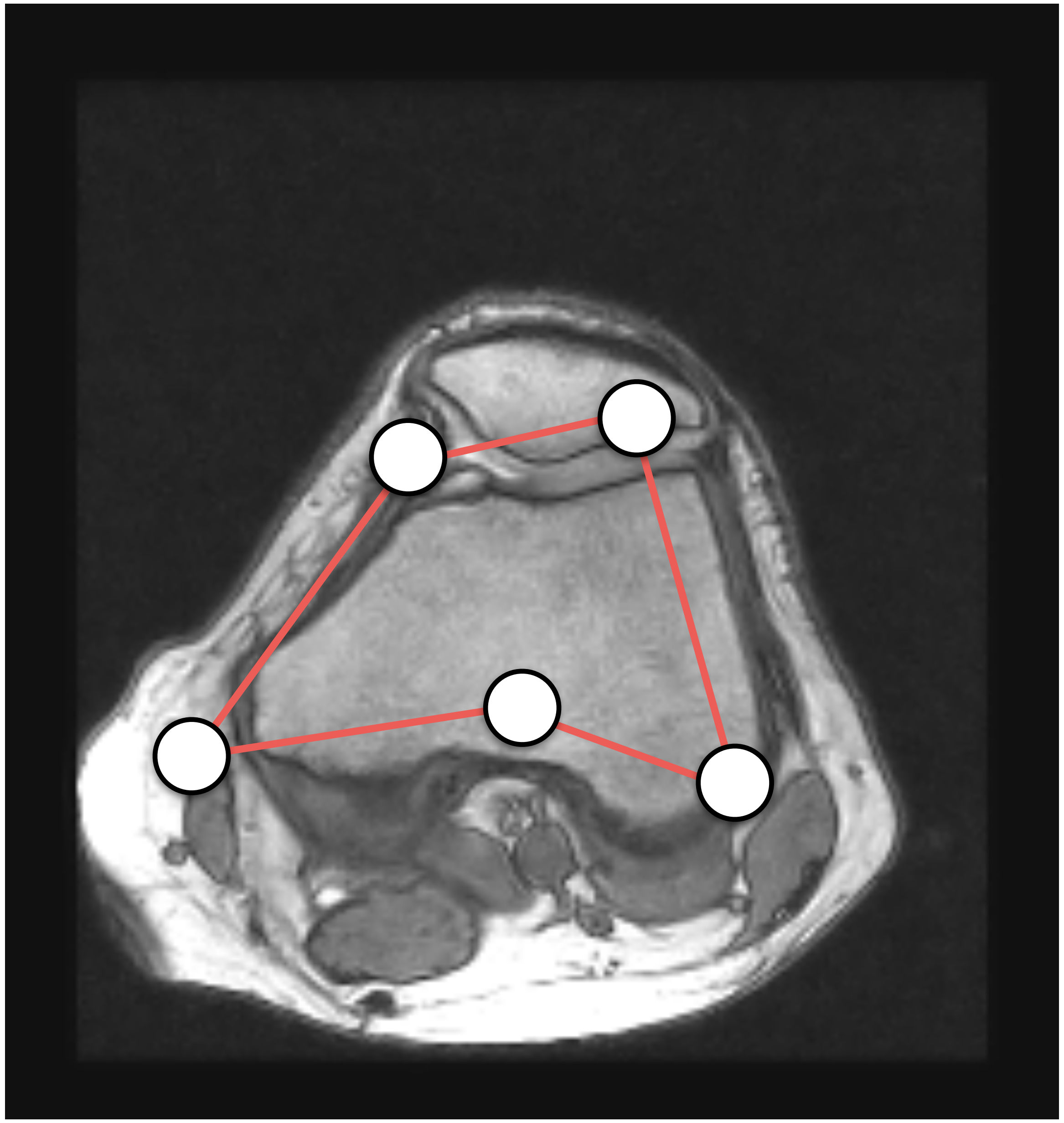
Snakes

- A first step, we draw a snake close to the boundary of the object we want to segment.



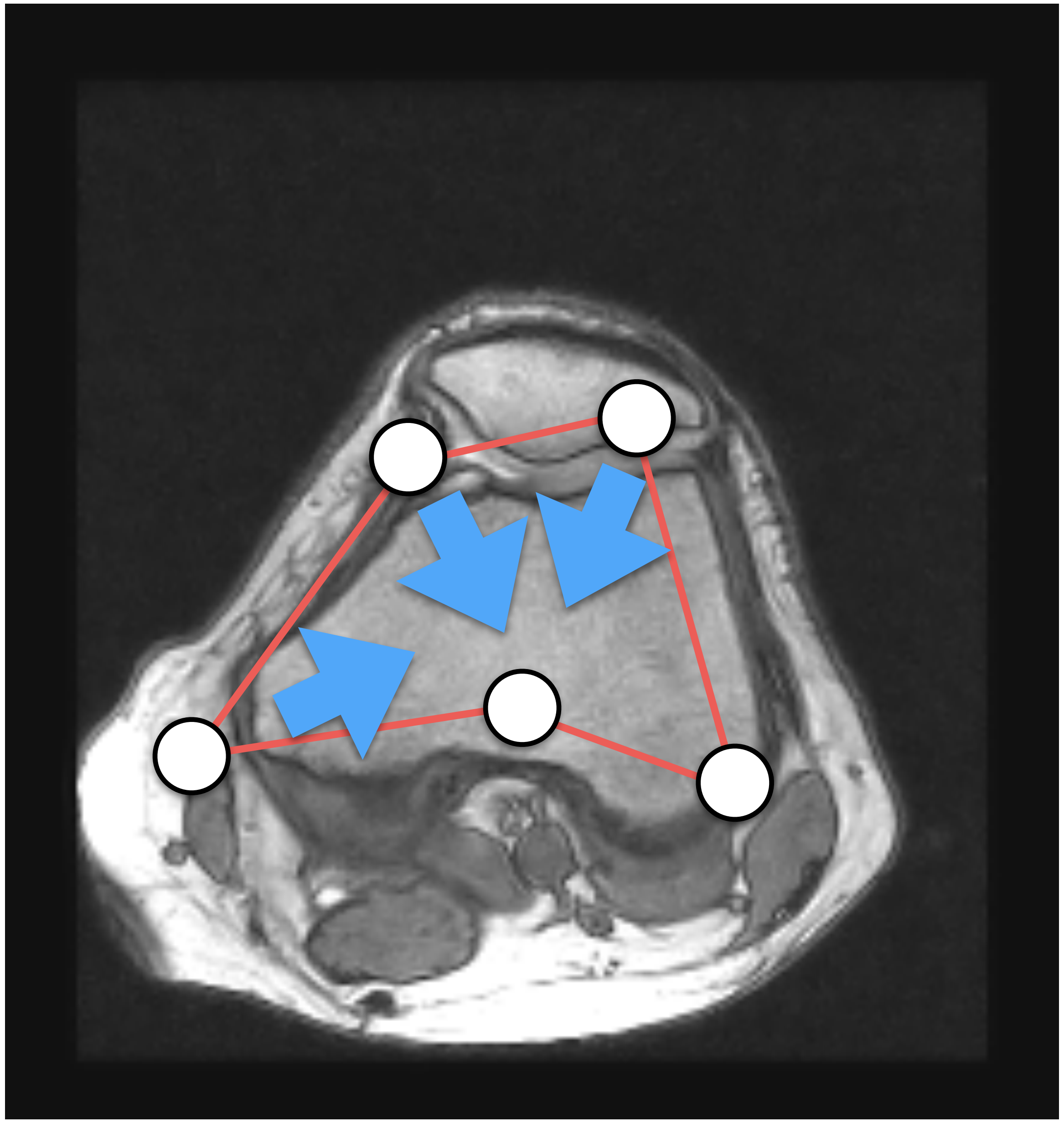
Snakes

- Then, we ***deform*** its control points in order to move them towards the object's boundary.



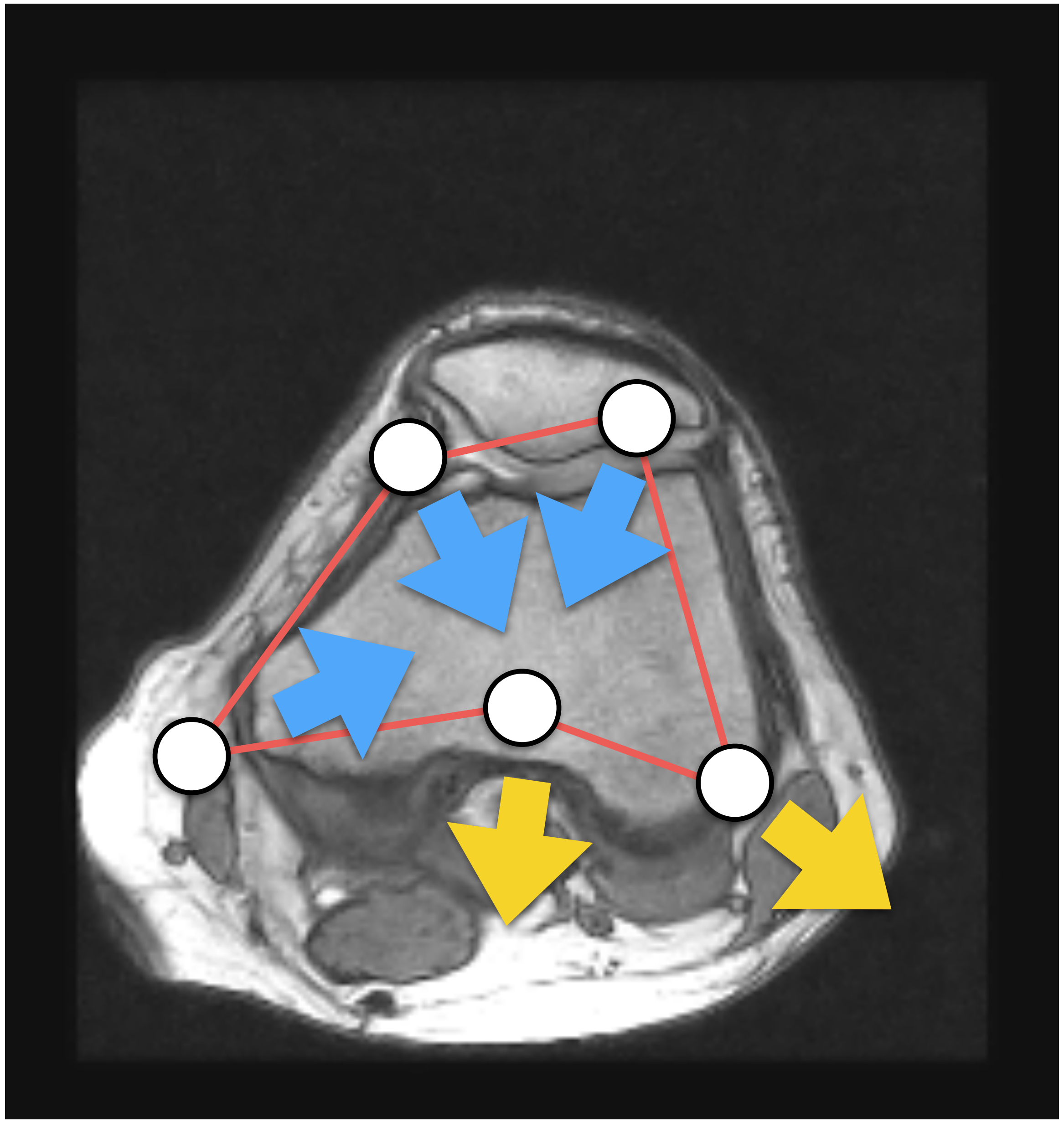
Snakes

- Then, we ***deform*** its control points in order to move them towards the object's boundary.



Snakes

- Then, we ***deform*** its control points in order to move them towards the object's boundary.



Snakes

- How do we deform the control points?
- An energy function E is associated with the curve.
- We deform control points by minimizing E; i.e., we solve an optimization problem.

Snakes

- How do we define the energy function?
- The energy of a snake has three components:

$$E = E_{\text{internal}} + E_{\text{external}} + E_{\text{constraint}}$$

Snakes: Internal Energy

- This energy represents the internal energy of the cure due to bending. It is defined per point as

$$E_{\text{internal}}(\mathbf{v}(t)) = \frac{1}{2}\left(\alpha(t)\left|\frac{d\mathbf{v}(t)}{dt}\right|^2 + \beta(t)\left|\frac{d^2\mathbf{v}(t)}{d^2t}\right|^2\right)$$

- The total energy is defined as

$$E_{\text{internal}} = \int_0^1 E_{\text{internal}}(\mathbf{v}(t))dt$$

Snakes: Internal Energy

- This energy represents the internal energy of the cure due to bending. It is defined per point as

$$E_{\text{internal}}(\mathbf{v}(t)) = \frac{1}{2}\left(\boxed{\alpha(t)\left|\frac{d\mathbf{v}(t)}{dt}\right|^2} + \beta(t)\left|\frac{d^2\mathbf{v}(t)}{d^2t}\right|^2\right)$$

Elasticity

- The total energy is defined as

$$E_{\text{internal}} = \int_0^1 E_{\text{internal}}(\mathbf{v}(t))dt$$

Snakes: Internal Energy

- This energy represents the internal energy of the cure due to bending. It is defined per point as

$$E_{\text{internal}}(\mathbf{v}(t)) = \frac{1}{2}\left(\underbrace{\boxed{\alpha(t)\left|\frac{d\mathbf{v}(t)}{dt}\right|^2}}_{\text{Elasticity}} + \underbrace{\boxed{\beta(t)\left|\frac{d^2\mathbf{v}(t)}{d^2t}\right|^2}}_{\text{Stiffness}} \right)$$

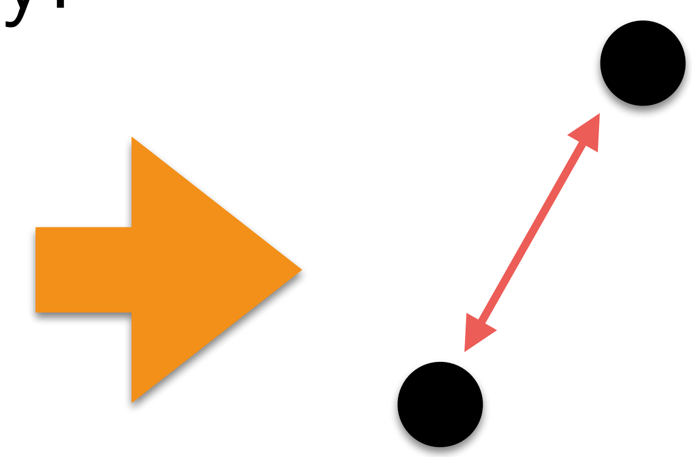
- The total energy is defined as

$$E_{\text{internal}} = \int_0^1 E_{\text{internal}}(\mathbf{v}(t))dt$$

Snakes: Internal Energy

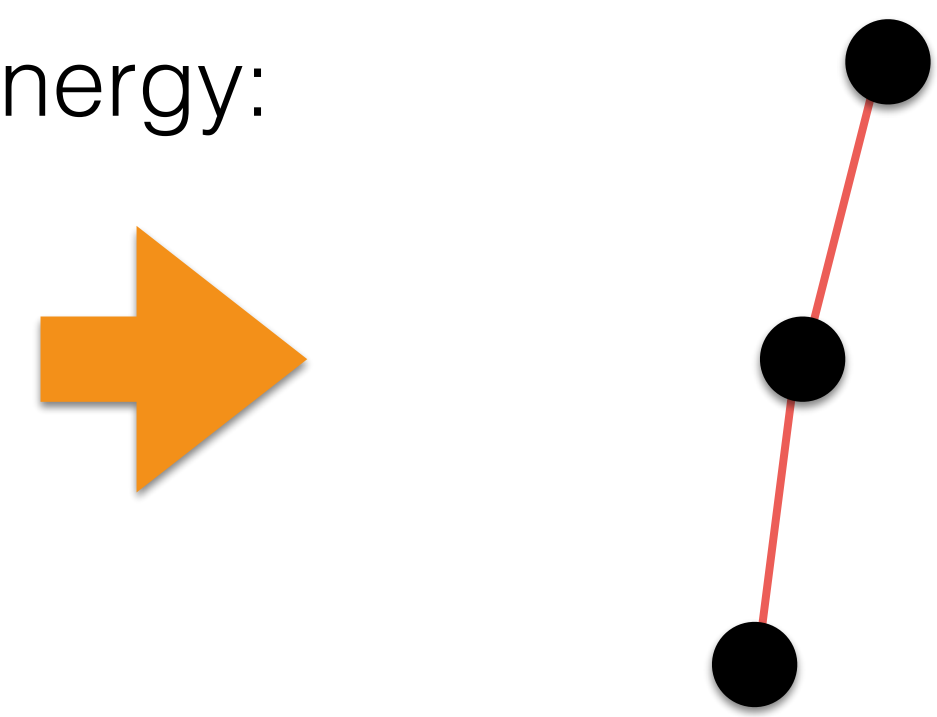
- The first term is an elastic energy:

$$\frac{d\mathbf{v}(t)}{dt} \approx \mathbf{v}_{i+1} - \mathbf{v}_i$$



- The second term is a bending energy:

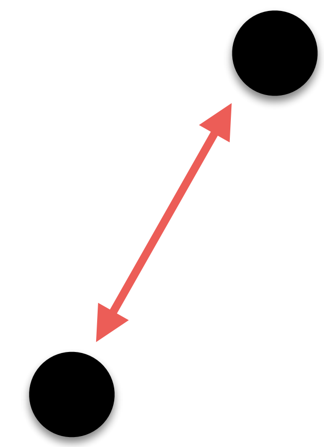
$$\frac{d^2\mathbf{v}(t)}{d^2t} \approx \mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}$$



Snakes: Internal Energy

- The first term is an elastic energy:

$$\frac{d\mathbf{v}(t)}{dt} \approx \mathbf{v}_{i+1} - \mathbf{v}_i$$



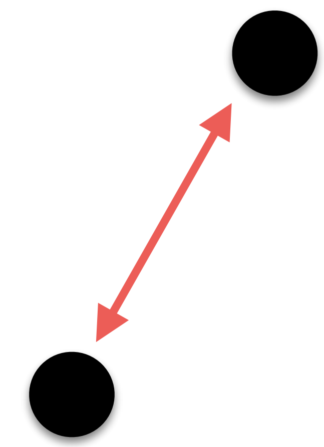
- The second term is a bending energy:

$$\frac{d^2\mathbf{v}(t)}{d^2t} \approx \mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}$$

Snakes: Internal Energy

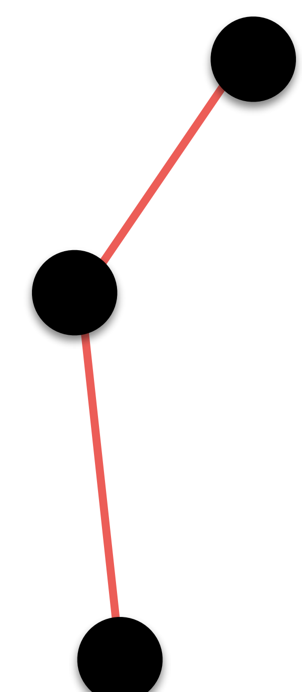
- The first term is an elastic energy:

$$\frac{d\mathbf{v}(t)}{dt} \approx \mathbf{v}_{i+1} - \mathbf{v}_i$$



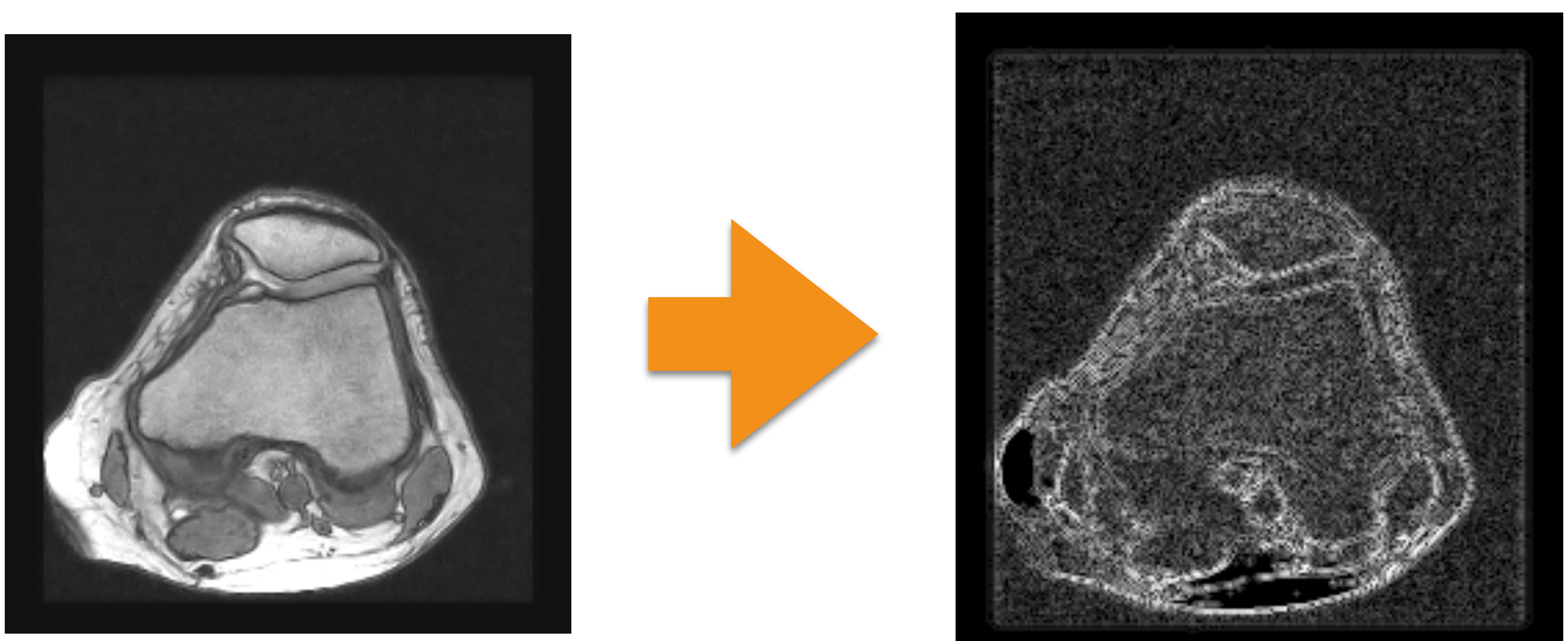
- The second term is a bending energy:

$$\frac{d^2\mathbf{v}(t)}{d^2t} \approx \mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}$$



Snakes: External Energy

- This energy determines how well the snake matches with the image locally!
- How can we achieve this?
  - Gradients magnitude



Snakes: External Energy

- It is defined per point as

$$E_{\text{external}}(\mathbf{v}(t)) = -\|\nabla I(\mathbf{v}(t))\|^2$$

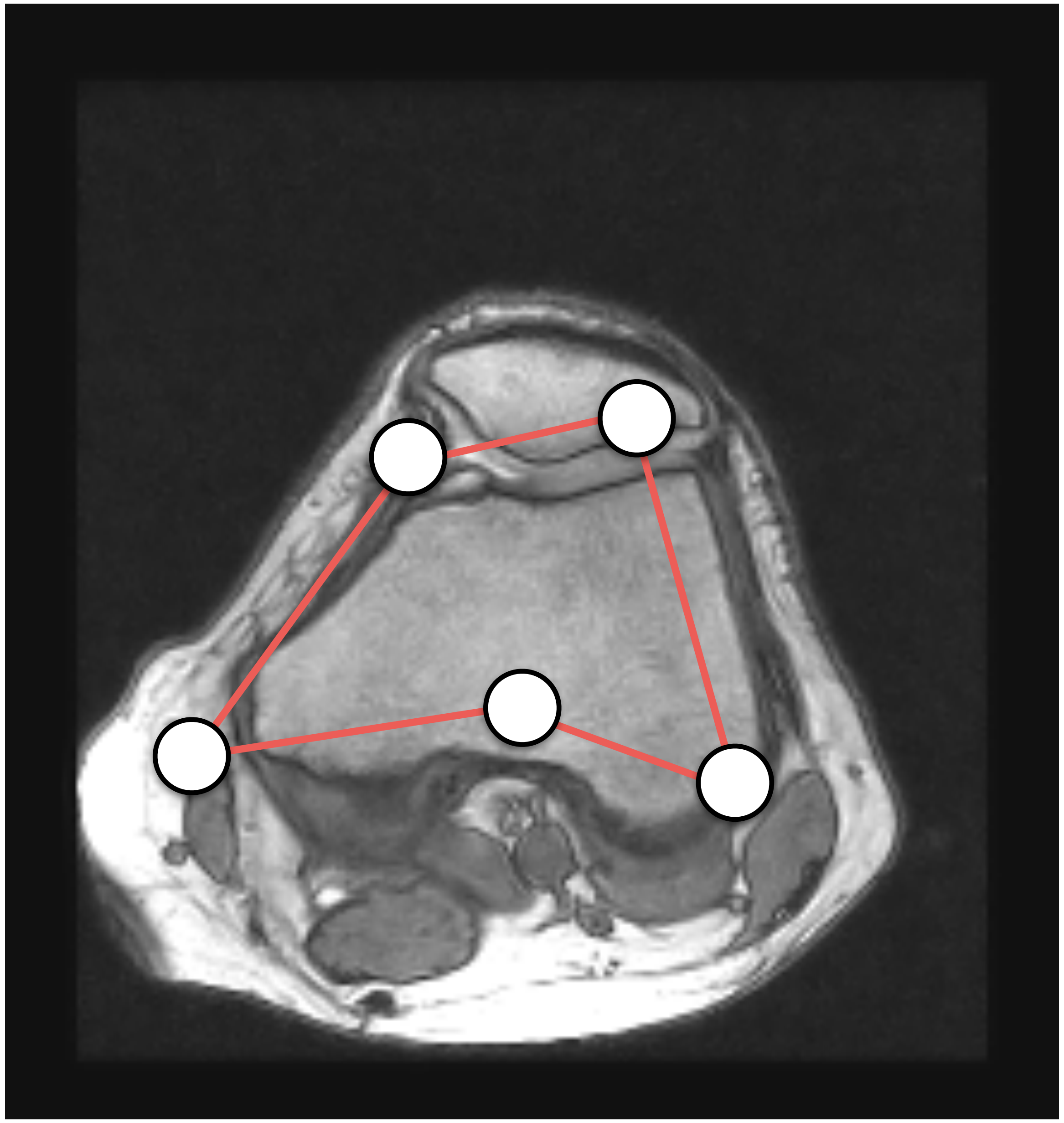
- The total energy is defined as

$$E_{\text{external}} = \int_0^1 E_{\text{external}}(\mathbf{v}(t))dt$$

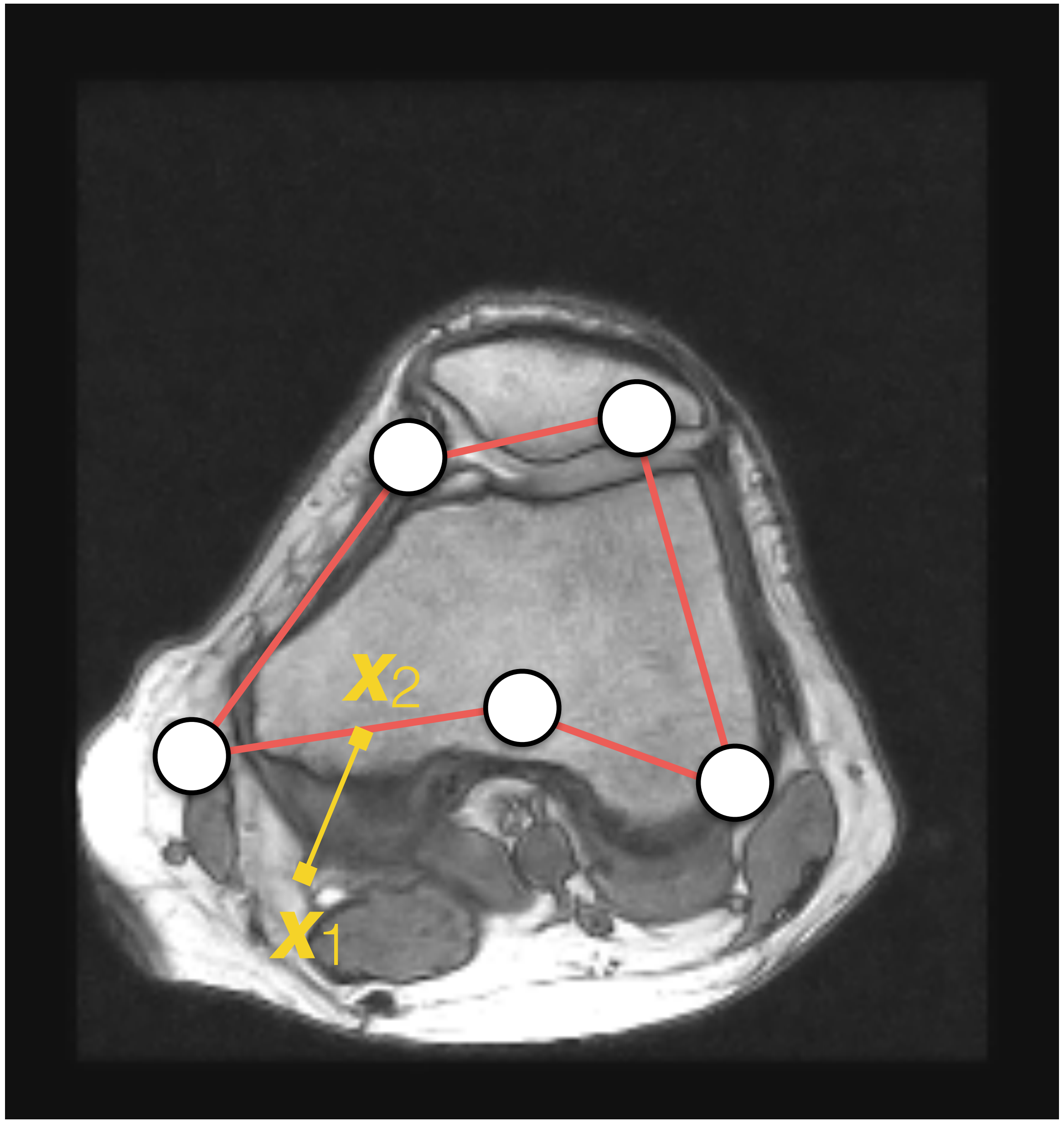
Snakes: Constraint Energy

- This energy is meant for interactive systems.
- The user interactively monitors the minimization, and she/he can push/pull vertices using the mouse cursor's position:
    - Repulsion forces or “vulcano”: $\dfrac{1}{r^2}$
    - Spring forces: $-k\|\mathbf{x}_1 - \mathbf{x}_2\|^2$

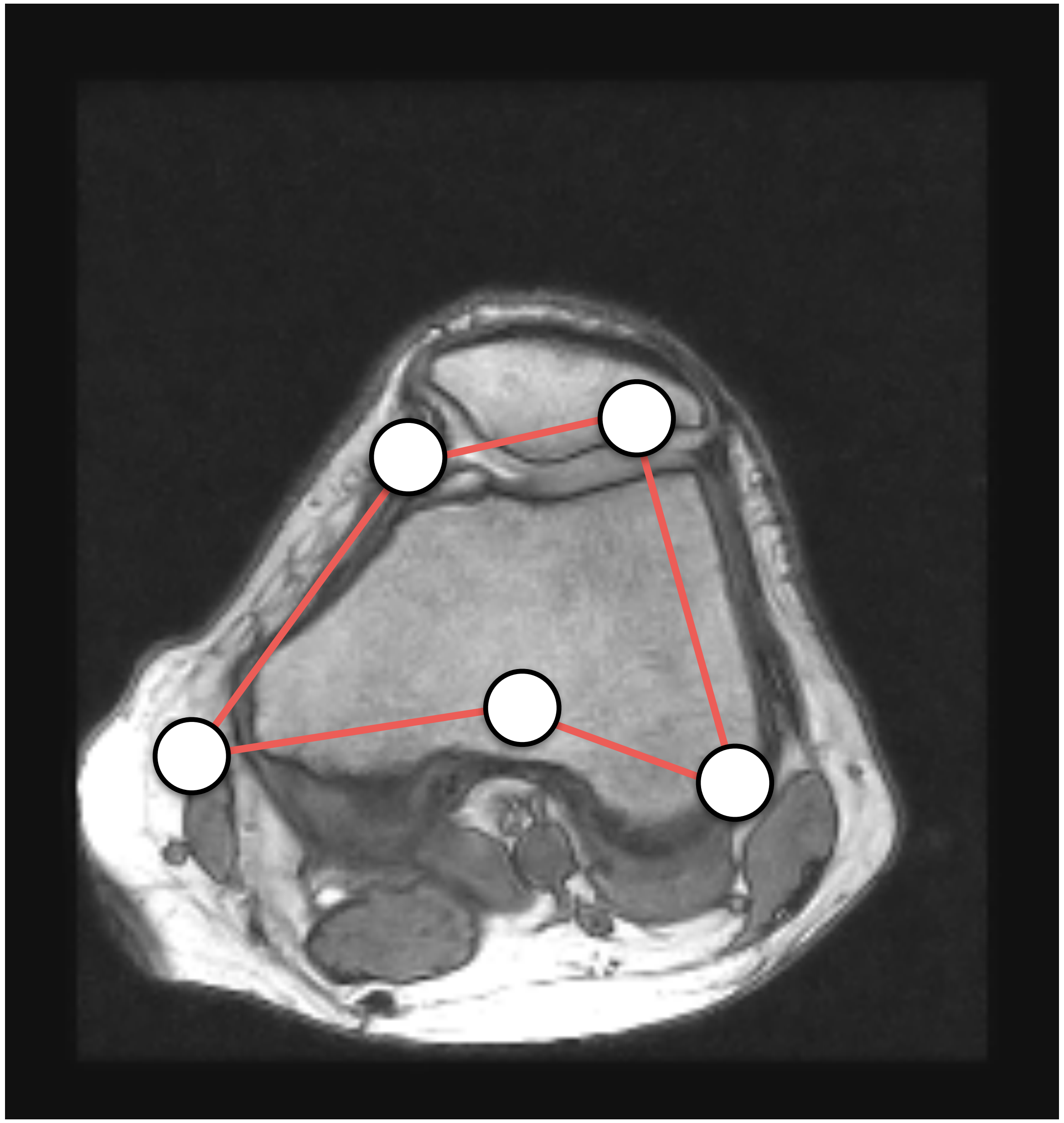
Snakes: Constraint Energy



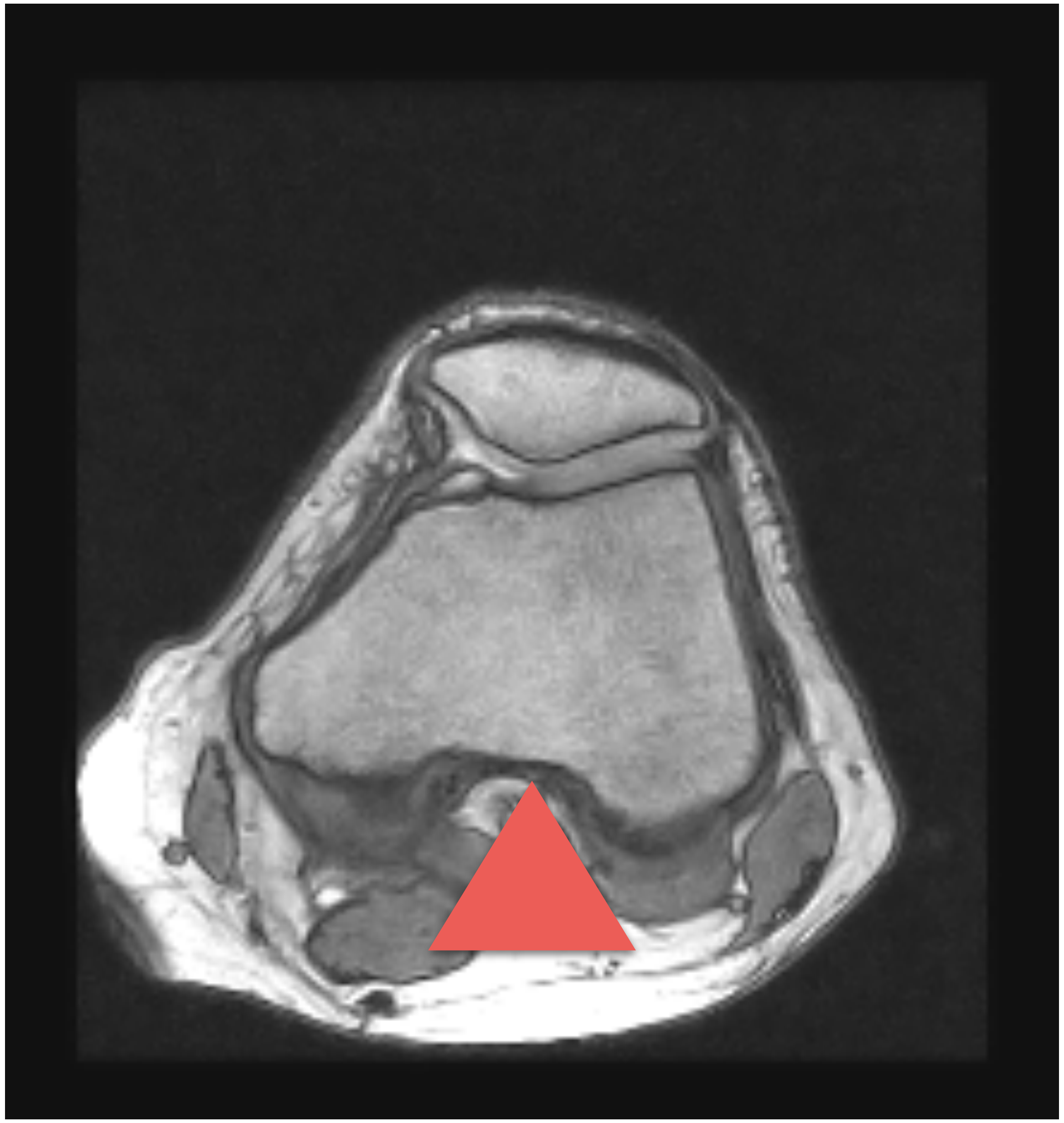
Snakes: Constraint Energy



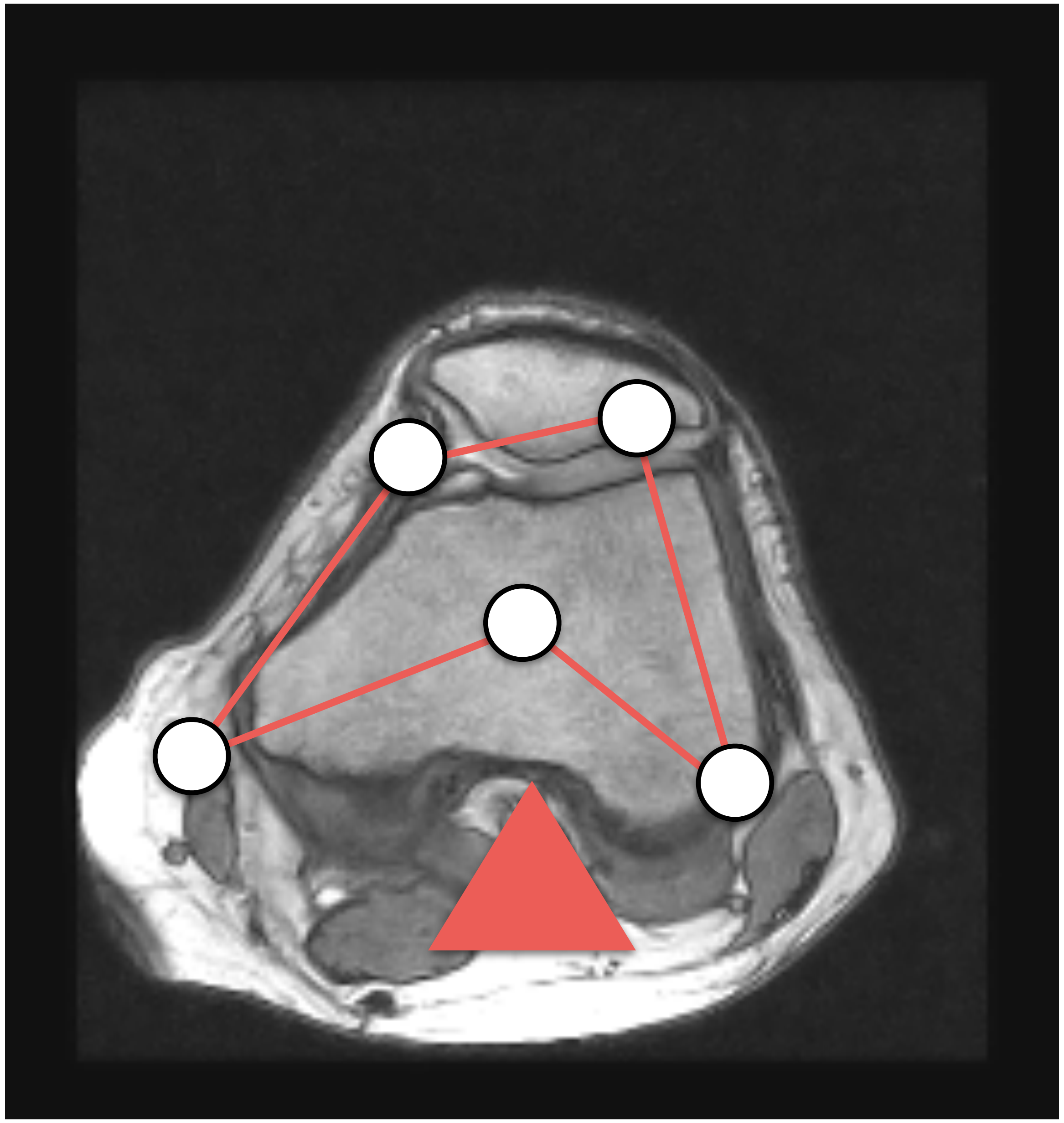
Snakes: Constraint Energy



Snakes: Constraint Energy



Snakes: Constraint Energy



How do we solve E?

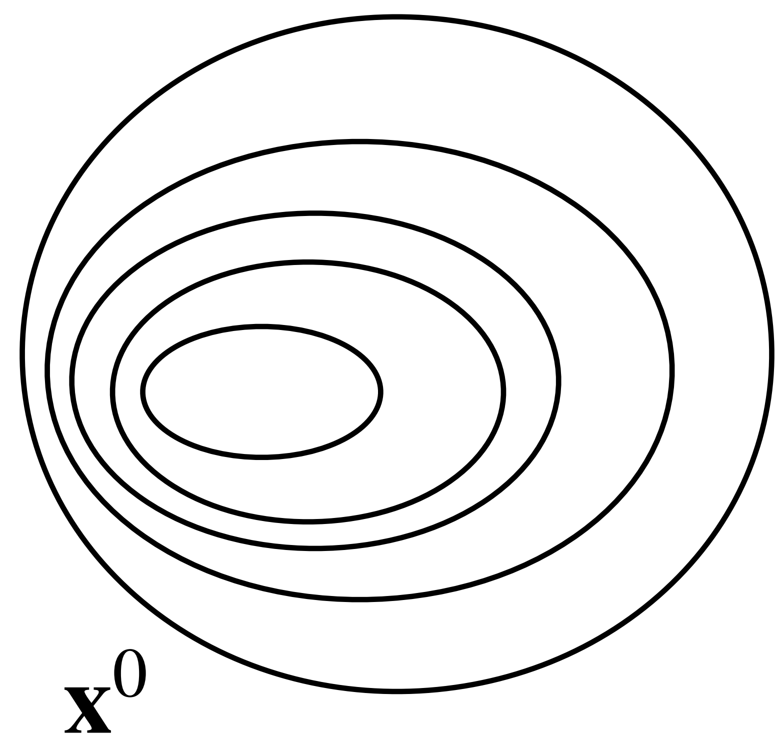
$$E = E_{\text{internal}} + E_{\text{external}} + E_{\text{constraint}}$$

Gradient Descent

- It is a first-order iterative optimization method:

$$\mathbf{x}_j^{i+1} = \mathbf{x}_j^i - \alpha \frac{\partial}{\partial \mathbf{x}_j} f(\mathbf{x}^i)$$

- $\mathbf{x}^0$ is a "**good**" guess.
- It will find a **local minimum**!
- f has to be differentiable.

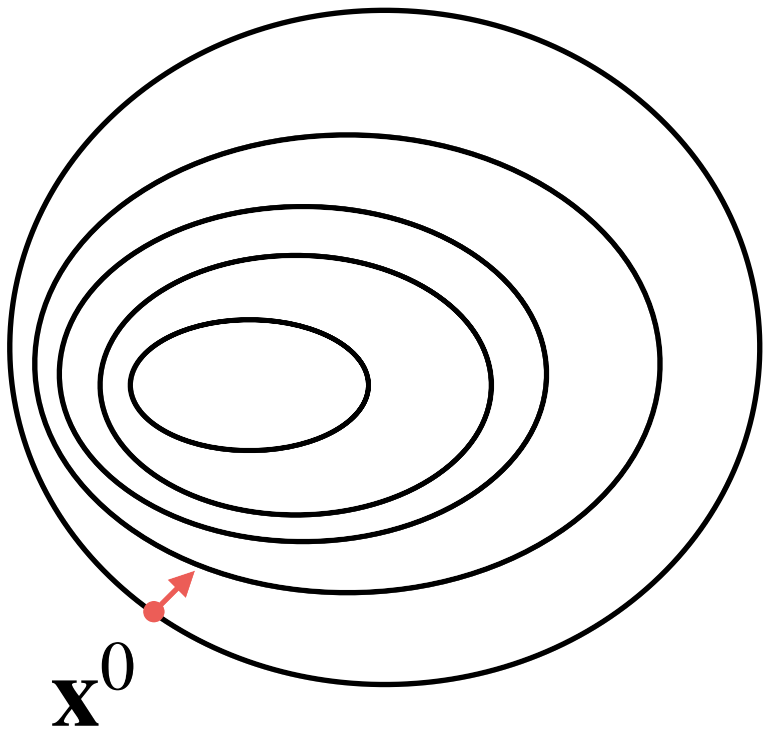


Gradient Descent

- It is a first-order iterative optimization method:

$$\mathbf{x}_j^{i+1} = \mathbf{x}_j^i - \alpha \frac{\partial}{\partial \mathbf{x}_j} f(\mathbf{x}^i)$$

- $\mathbf{x}^0$ is a "**good**" guess.
- It will find a **local minimum**!
- f has to be differentiable.

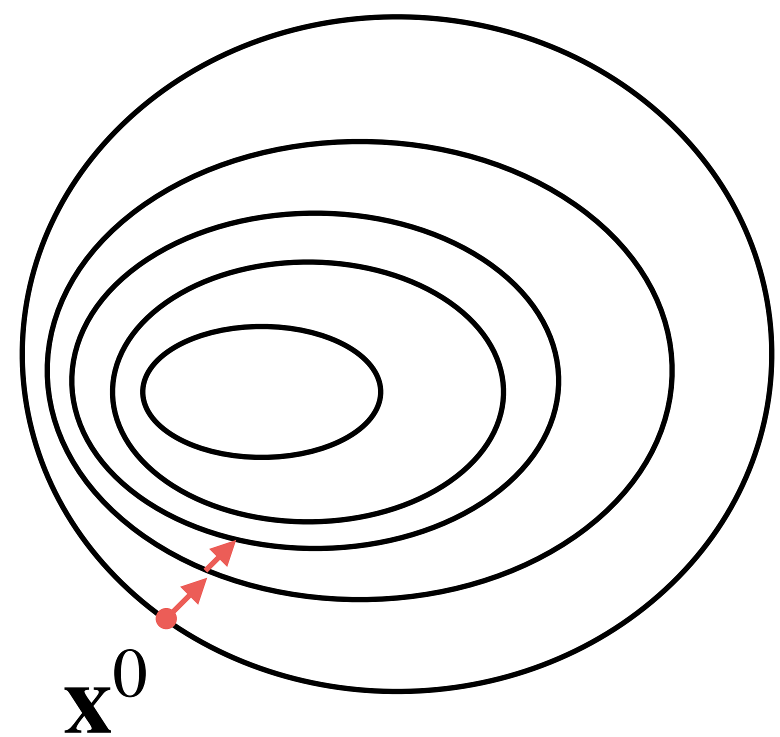


Gradient Descent

- It is a first-order iterative optimization method:

$$\mathbf{x}_j^{i+1} = \mathbf{x}_j^i - \alpha \frac{\partial}{\partial \mathbf{x}_j} f(\mathbf{x}^i)$$

- $\mathbf{x}^0$ is a "**good**" guess.
- It will find a **local minimum**!
- f has to be differentiable.

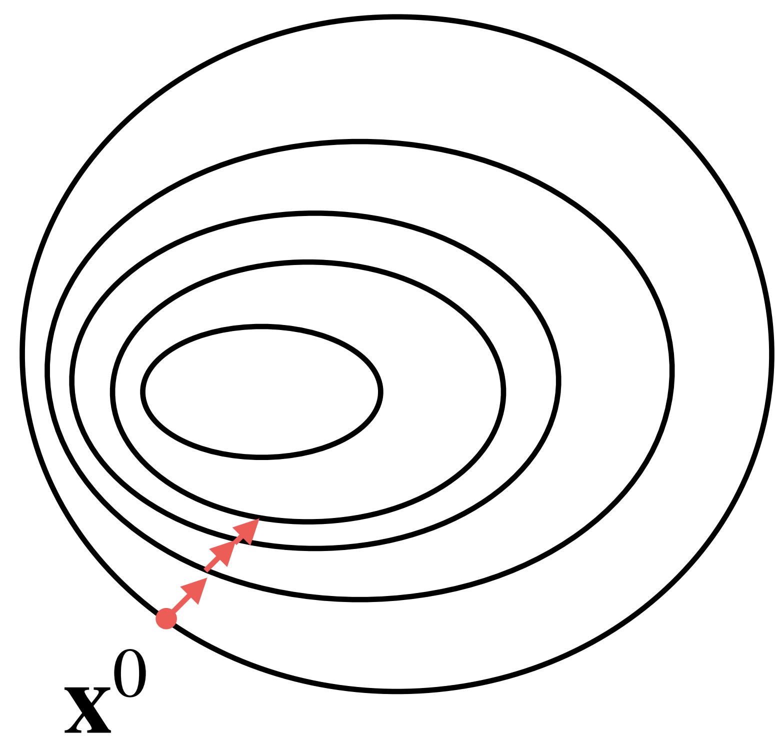


Gradient Descent

- It is a first-order iterative optimization method:

$$\mathbf{x}_j^{i+1} = \mathbf{x}_j^{i} - \alpha \frac{\partial}{\partial \mathbf{x}_j} f(\mathbf{x}^{i})$$

- $\mathbf{x}^0$ is a "**good**" guess.
- It will find a **local minimum**!
- f has to be differentiable.

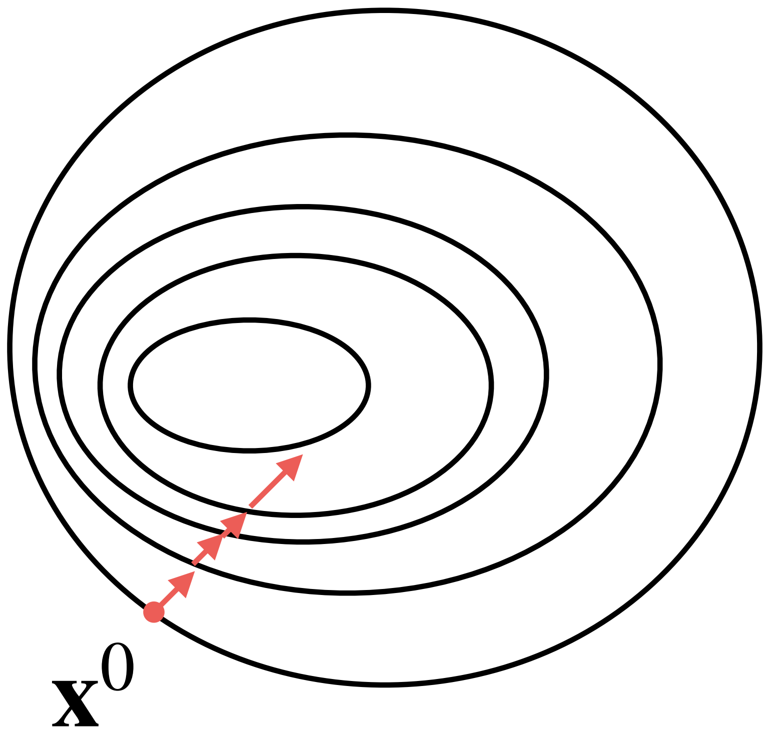


Gradient Descent

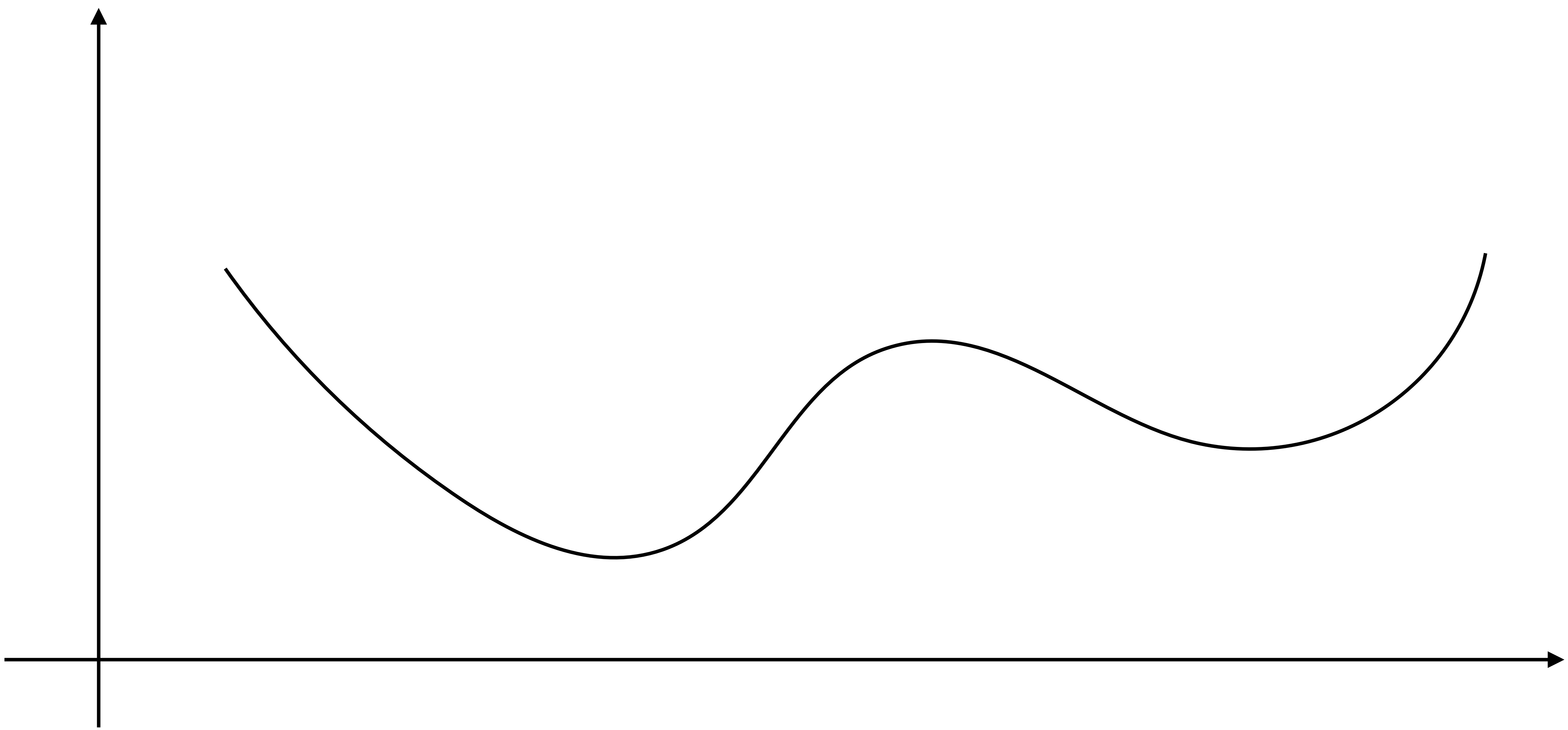
- It is a first-order iterative optimization method:

$$\mathbf{x}_j^{i+1} = \mathbf{x}_j^i - \alpha \frac{\partial}{\partial \mathbf{x}_j} f(\mathbf{x}^i)$$

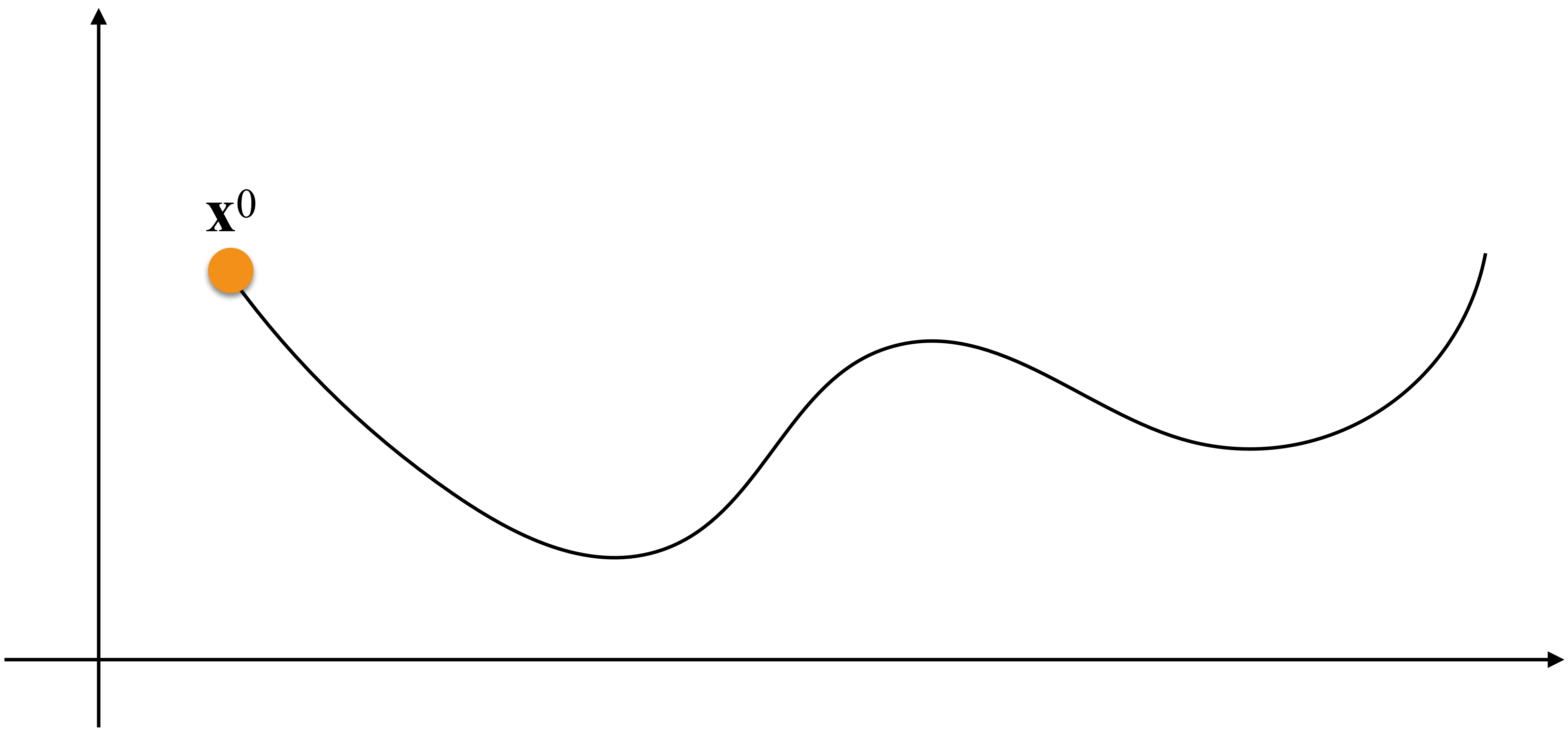
- $\mathbf{x}^0$ is a “**good**” guess.
- It will find a **local minimum**!
- f has to be differentiable.



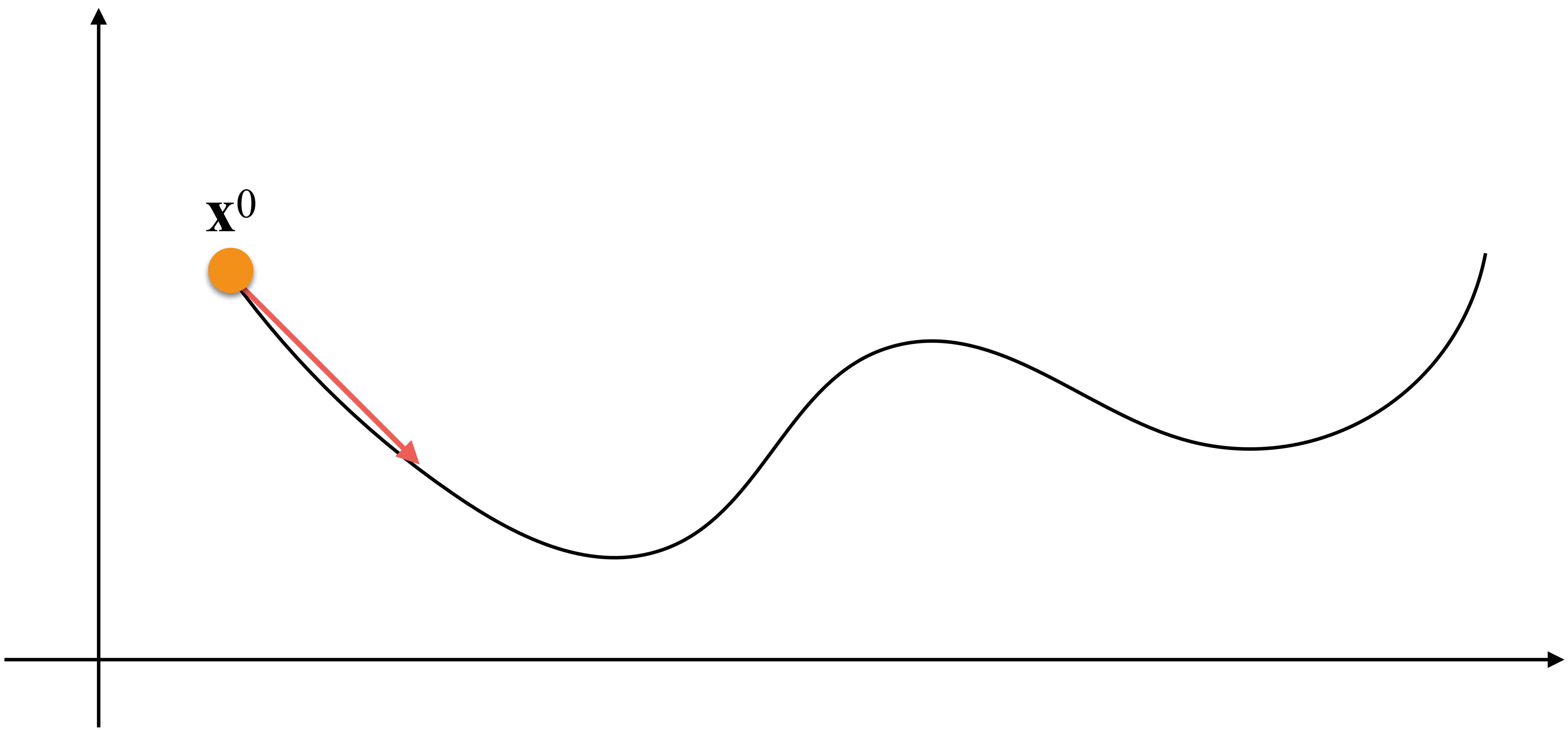
Gradient Descent



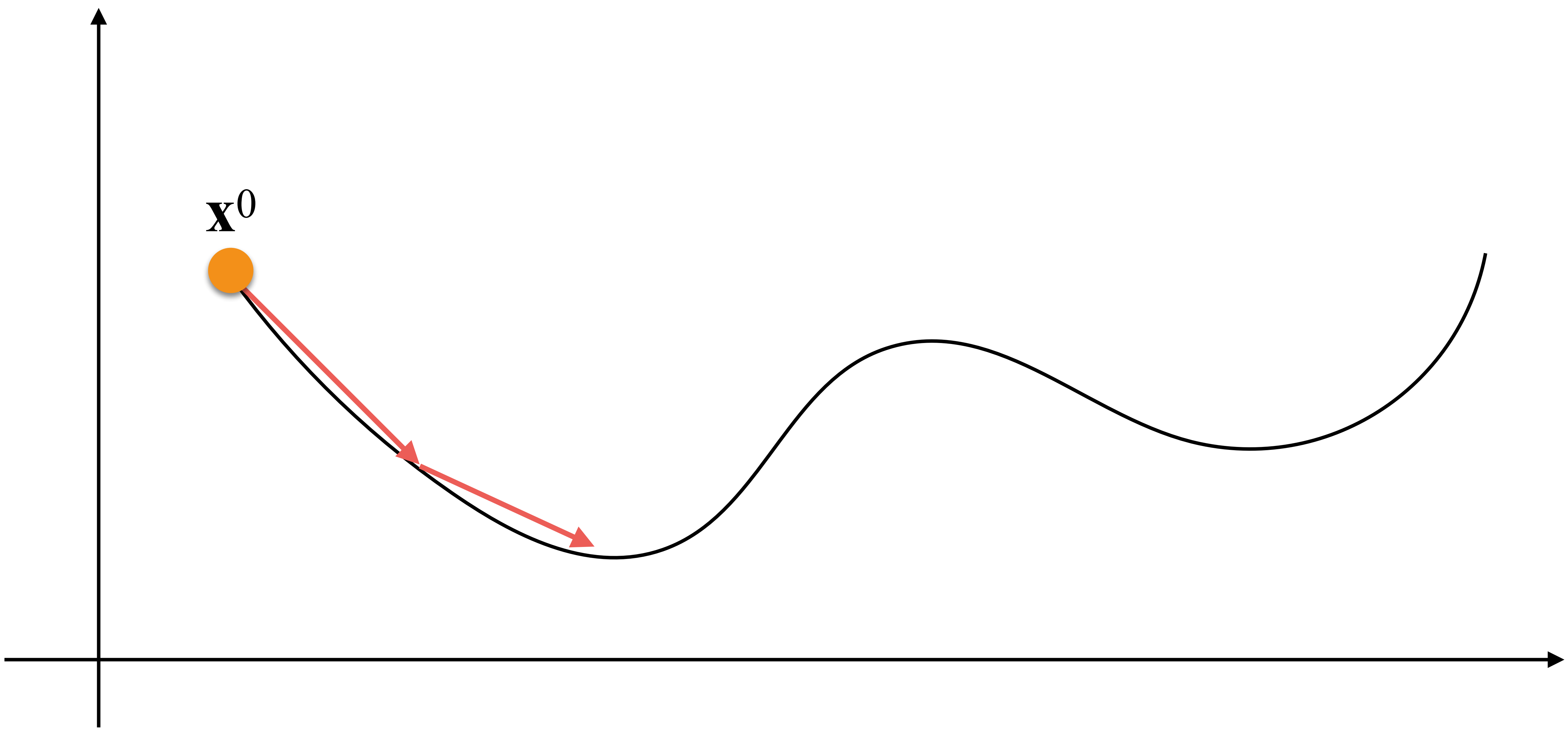
Gradient Descent



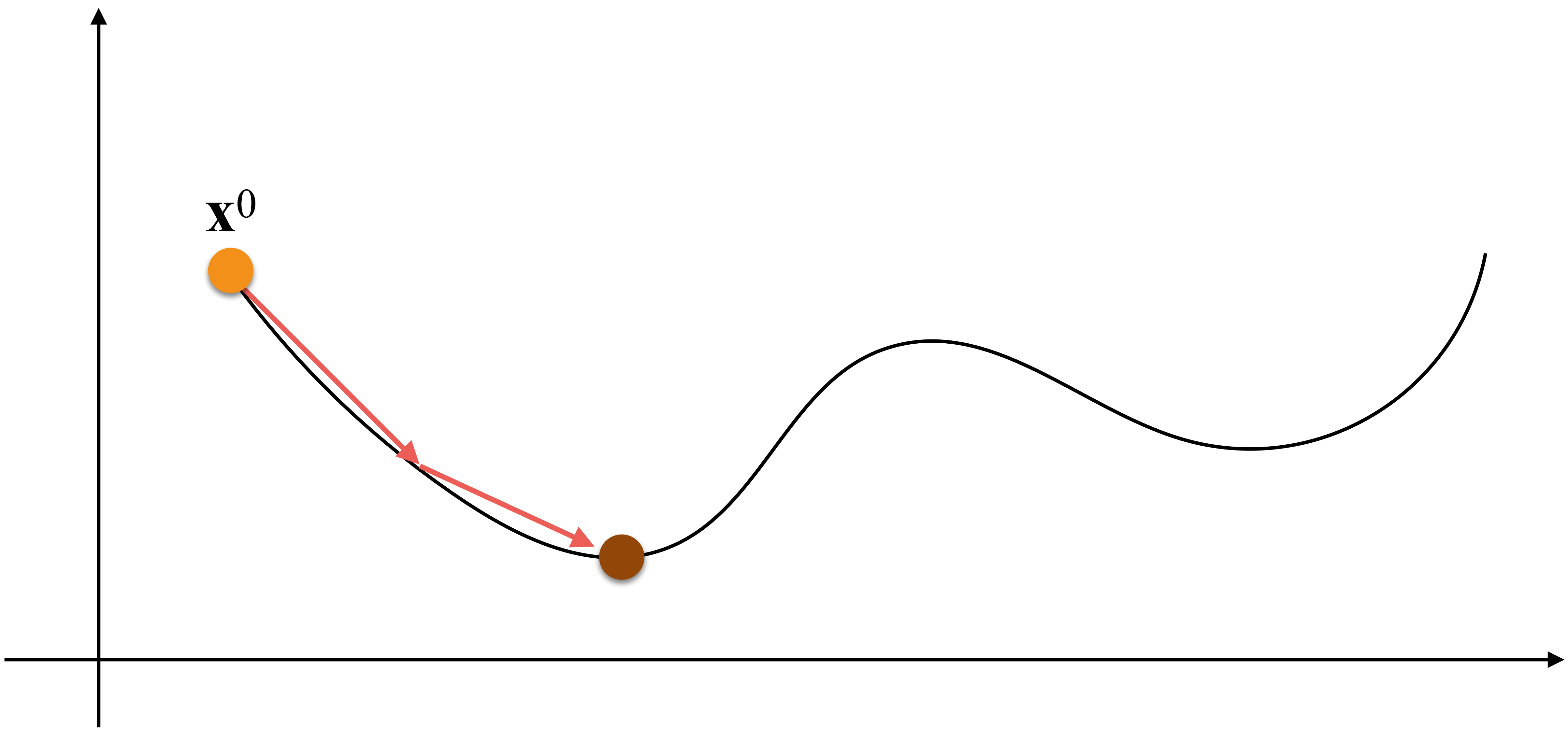
Gradient Descent



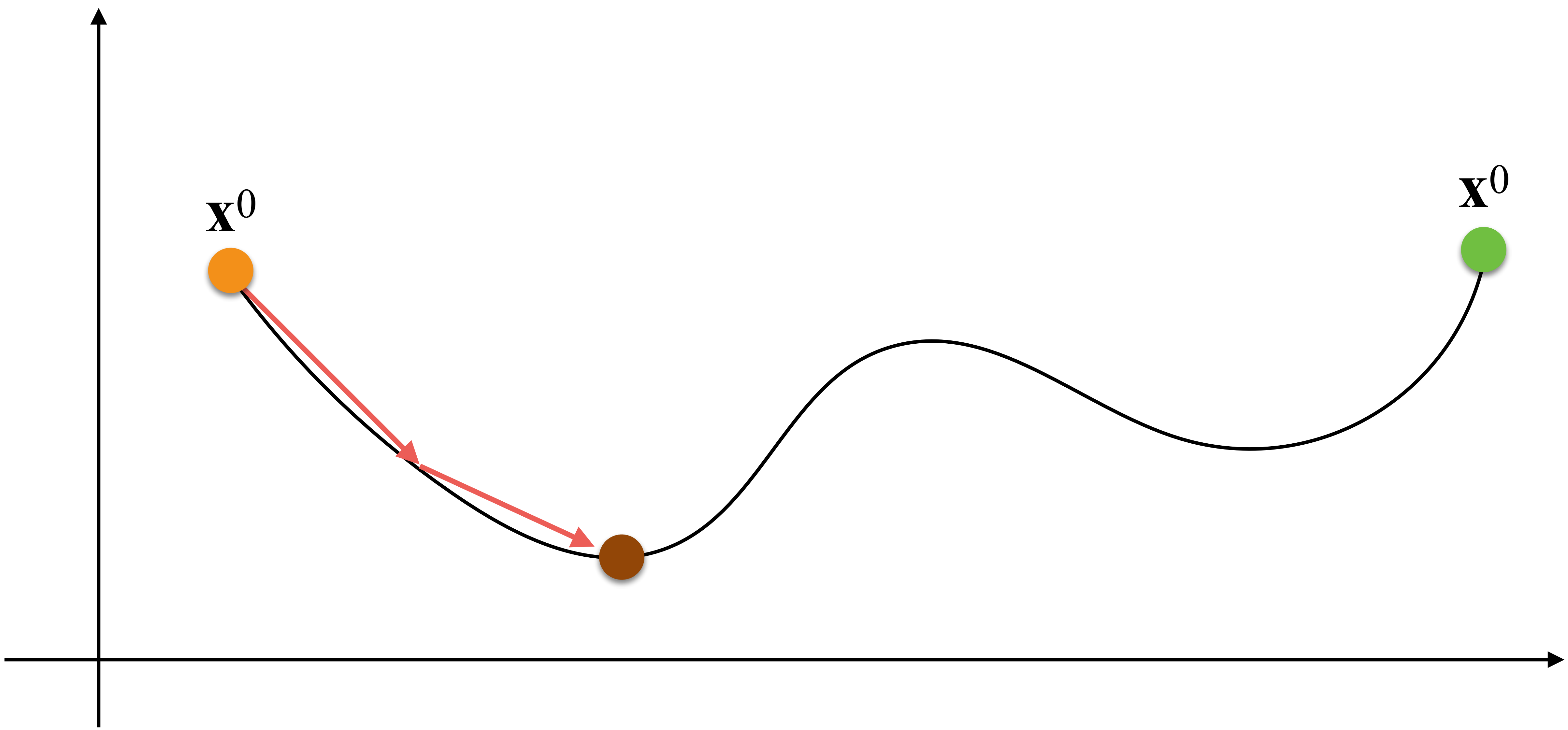
Gradient Descent



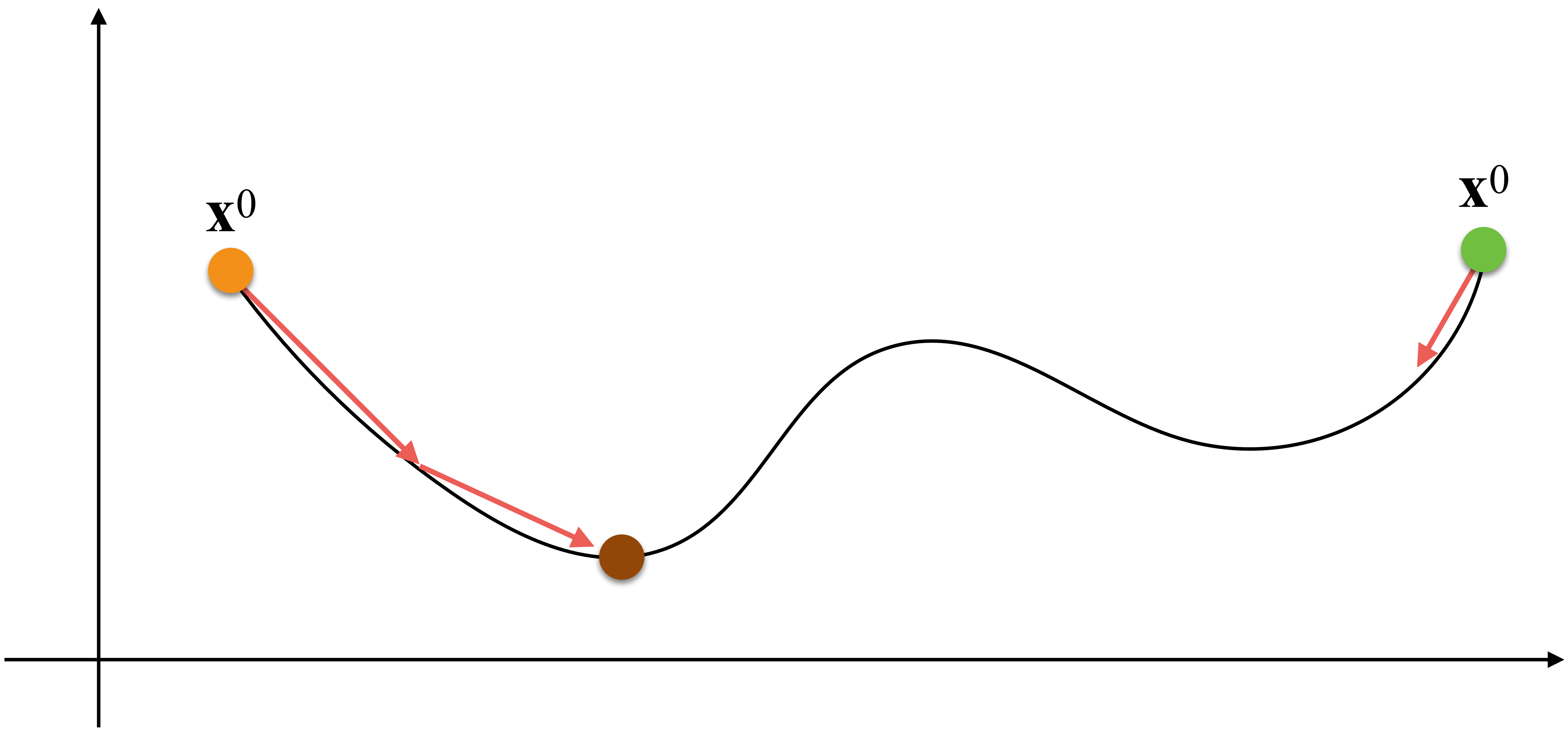
Gradient Descent



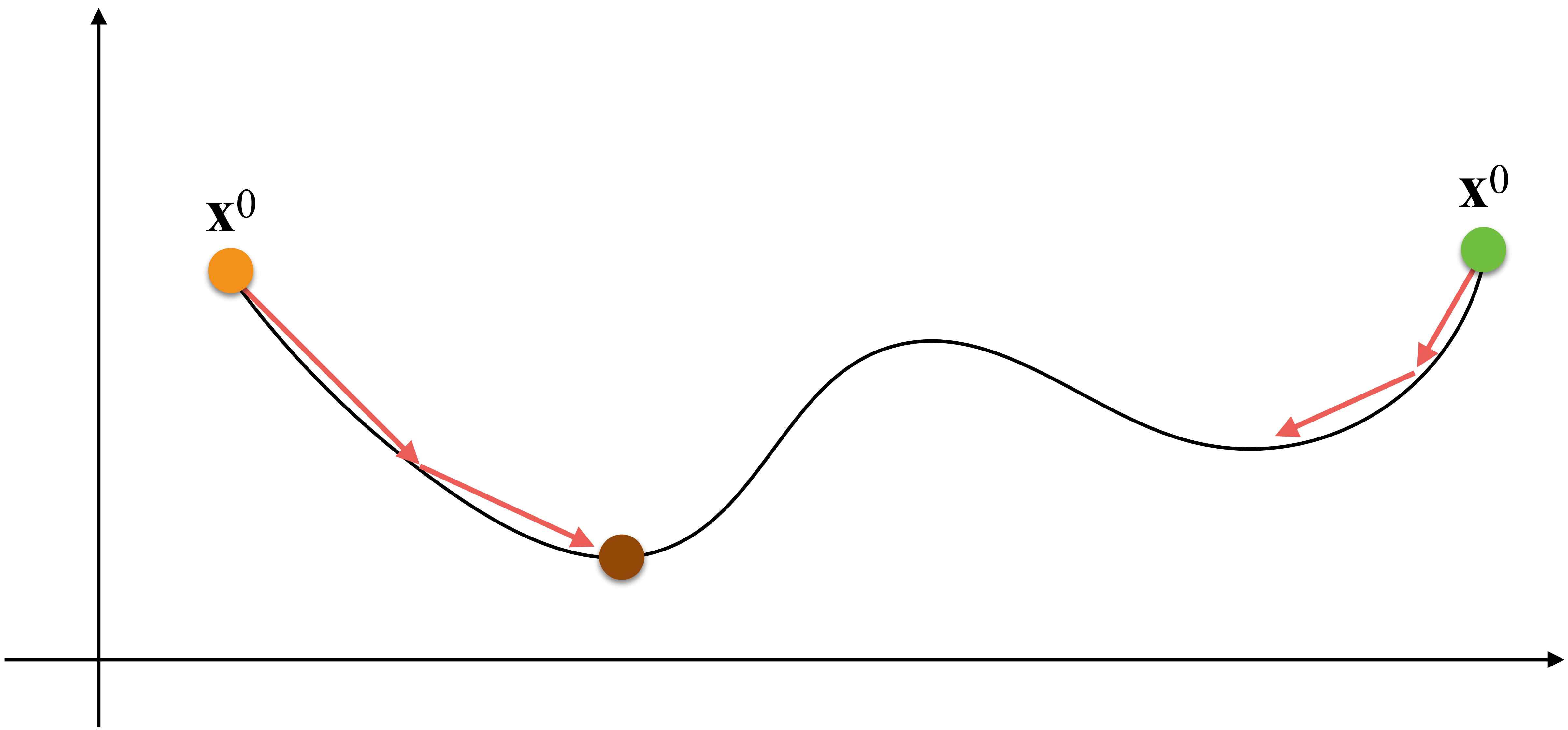
Gradient Descent



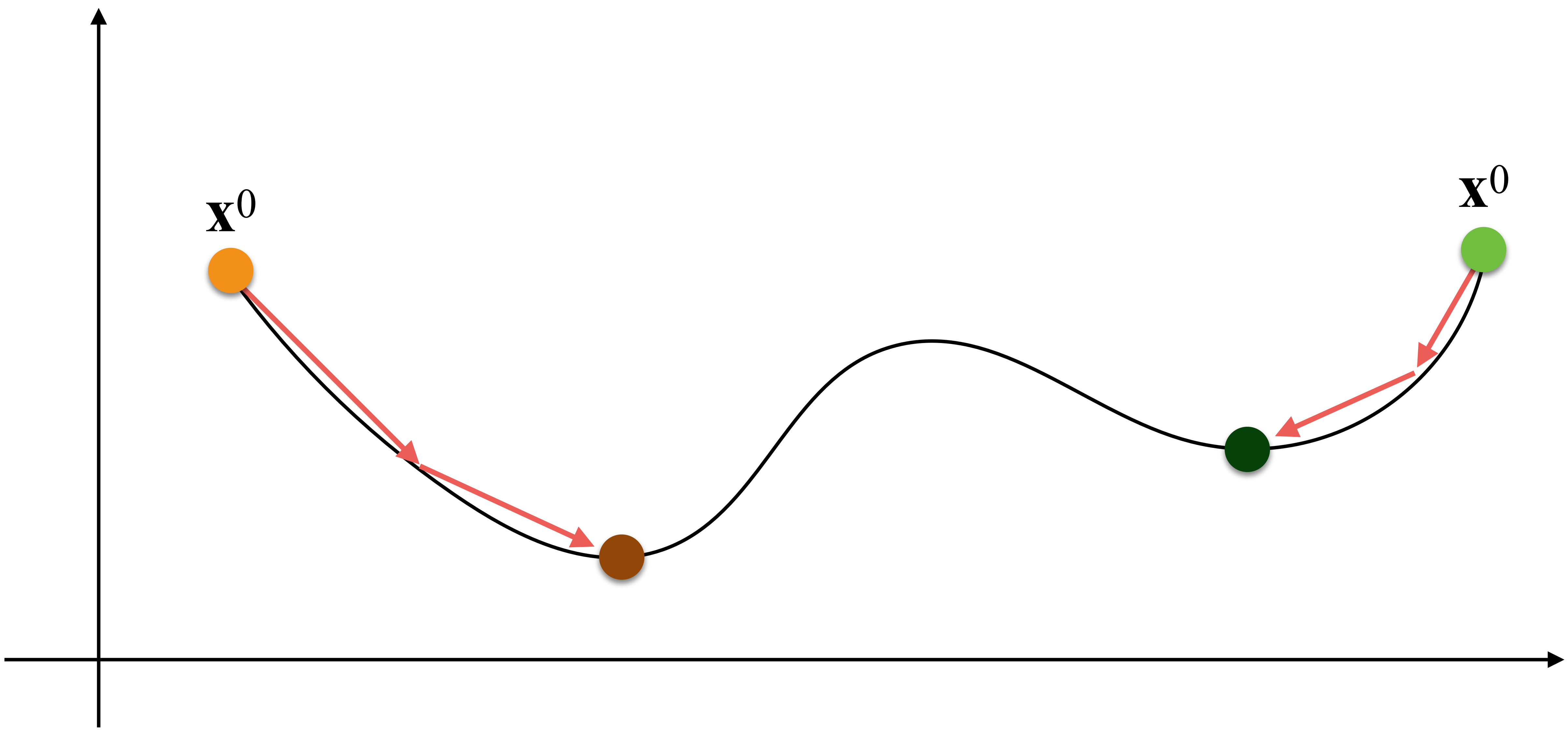
Gradient Descent



Gradient Descent



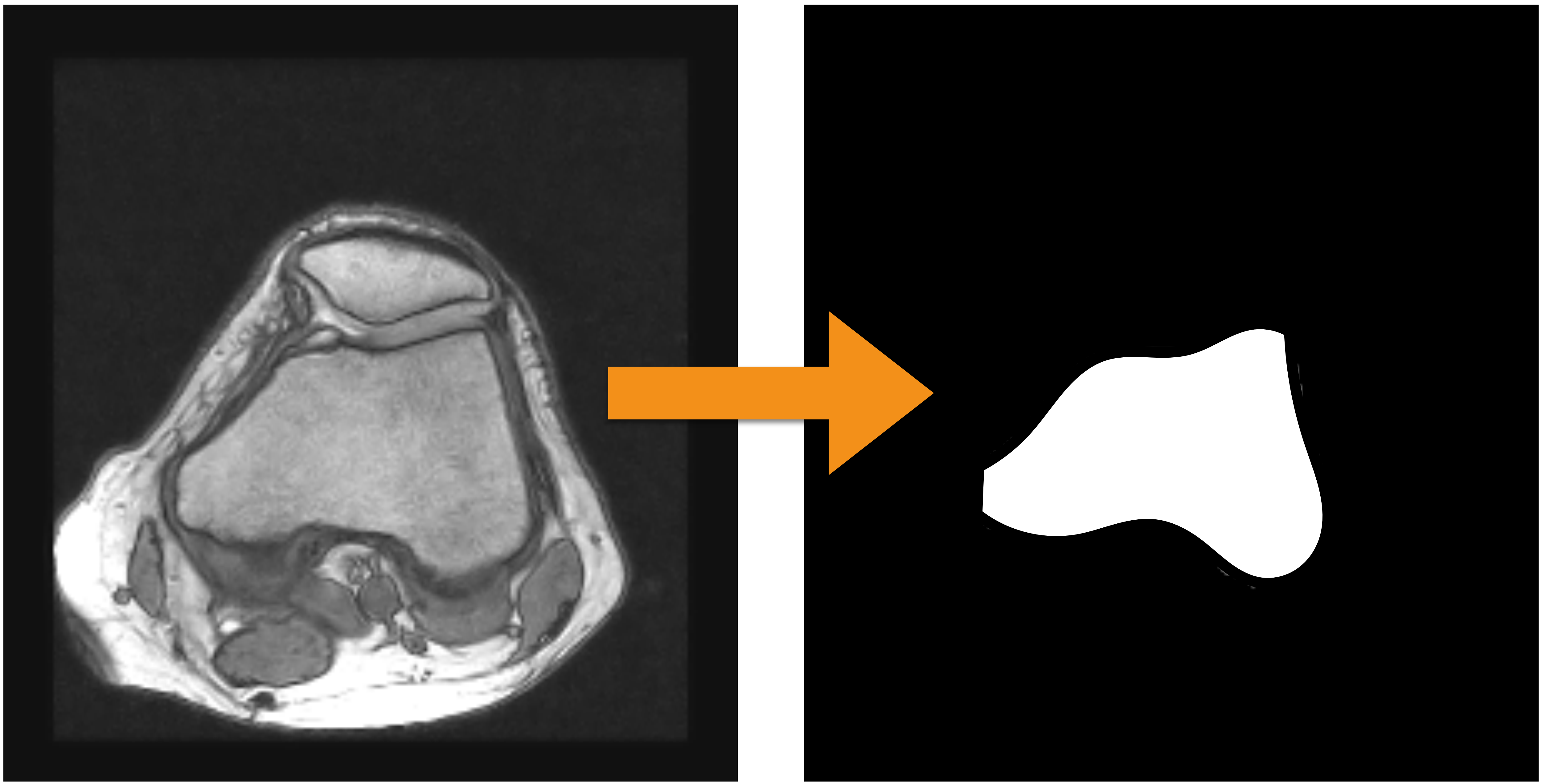
Gradient Descent



Snakes: Gradient Descent

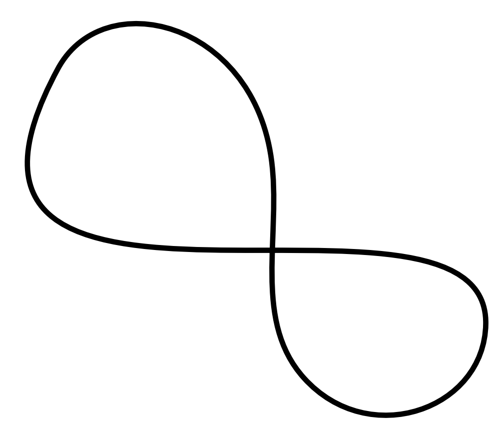
- What is our $\mathbf{x}^0$ in the snake minimization?
- We need to click a few points in the image around our object of interest!

Snakes An Example



Snakes

- Extension to the 3D case:
  - Instead of a curve we have a parametric surface; e.g., we can start using a sphere.
- Disadvantages:
  - We may have an over-smooth boundaries when using splines
  - How many *n* control points?
  - Not trivial to avoid self-intersection!



Stroke-Based

Stroke-Based

- Stroke-based algorithms are based on the idea to define with a stroke what is foreground (i.e., our object of interest) and what is background.

- These strokes are roughly painted.

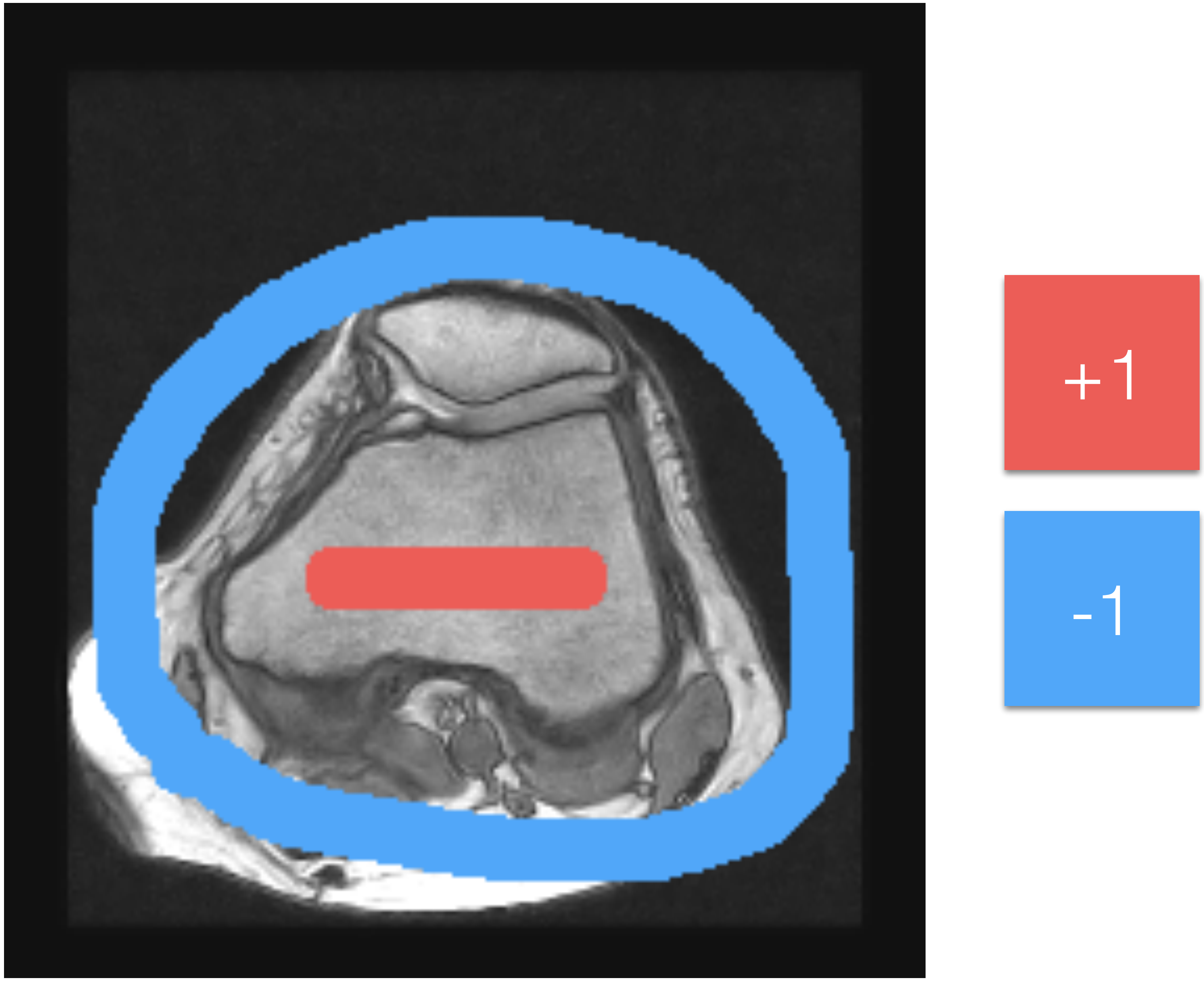
    - However, they have to be placed in areas where we are 100% sure how to classify the image.

Stroke-Based

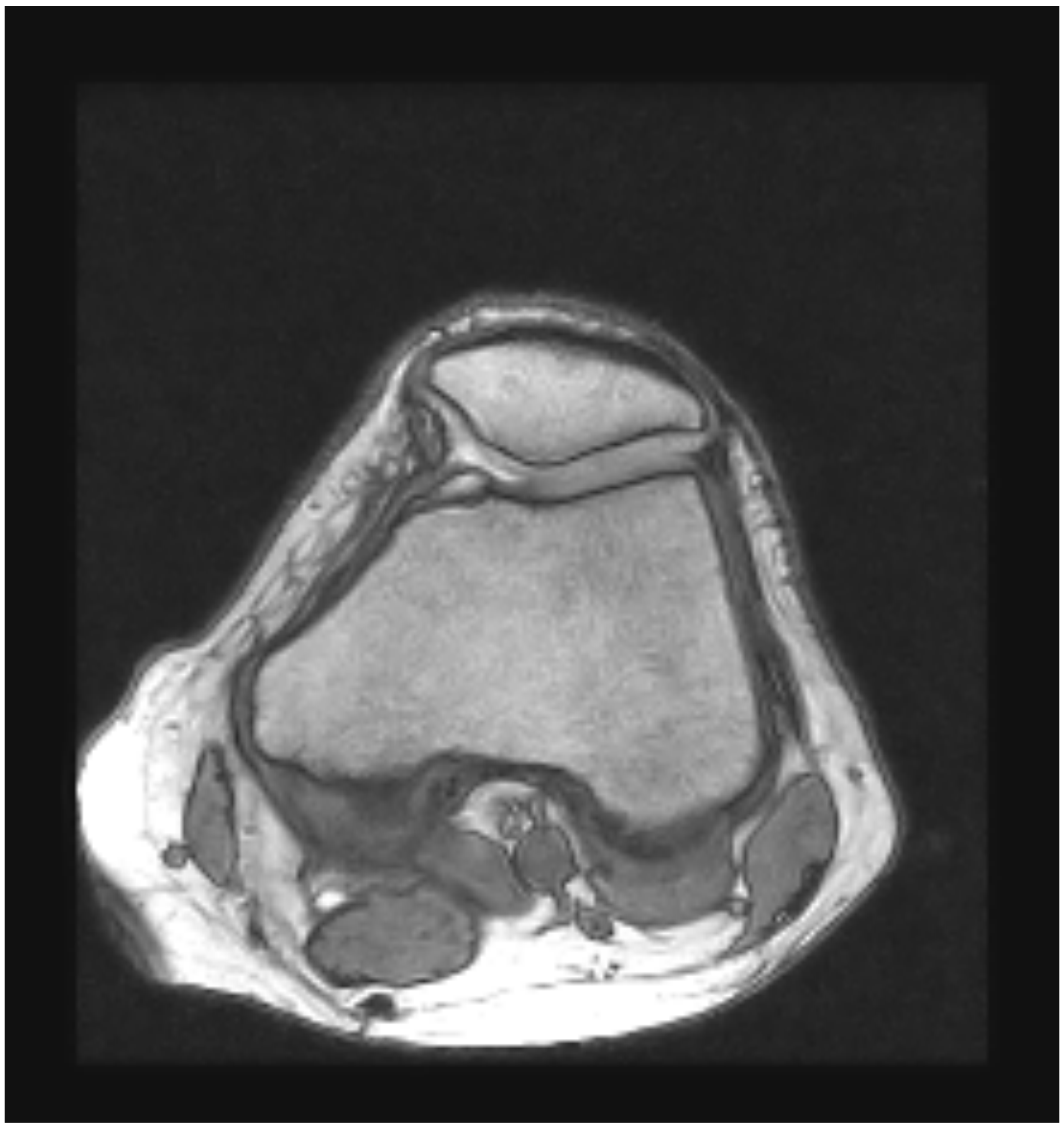
+1

-1

Stroke-Based



Stroke-Based



C

s

Stroke-Based: Grow-Cut

- Grow-cut is a stroke-based method.
- The idea is to propagate the label of the current pixels if its neighbors are “similar”.

Stroke-Based: Grow-Cut

- For each pixel, we have:

$$< l_i ; \theta_i ; C_i >$$

- Initialization for pixels **not covered** by a stroke (s):

$$< l_i = 0; \theta_i = 0; C_i = I(x_i, y_i) > \qquad \forall_i s(x_i, y_i) = 0$$

- Initialization for pixels **covered** by a stroke (s):

$$< l_i = s(x_i, y_i); \theta_i = 1; C_i = I(x_i, y_i) > \qquad \forall_i s(x_i, y_i) \neq 0$$

Stroke-Based: Grow-Cut

- For each pixel, we have:

$$\text{Label} < \boxed{l_i;} \overset{\text{Strength}}{\boxed{\theta_i;}} \boxed{C_i} > \text{Intensity}$$

- Initialization for pixels **not covered** by a stroke (s):

$$< l_i = 0; \theta_i = 0; C_i = I(x_i, y_i) > \qquad \forall_i s(x_i, y_i) = 0$$

- Initialization for pixels **covered** by a stroke (s):

$$< l_i = s(x_i, y_i); \theta_i = 1; C_i = I(x_i, y_i) > \qquad \forall_i s(x_i, y_i) \neq 0$$

Stroke-Based: A Single Grow-Cut Pass

- For each pixel I in the image:
  - We copy the previous status:

$$< l_i^{t+1}, \theta_i^{t+1}, C_i^{t+1} > = < l_i^t, \theta_i^t, C_i^t >$$

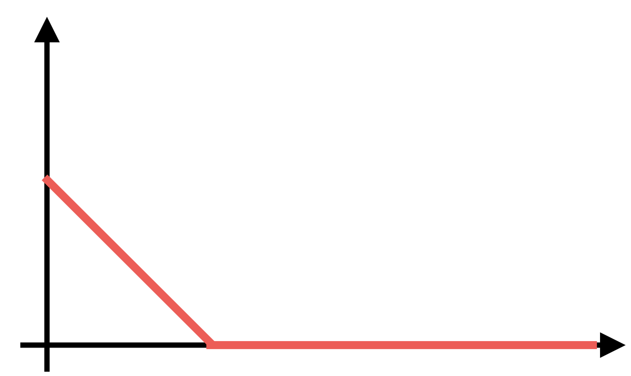
  - For each neighbor j of i:
    - if $g(C_i^t - C_j^t) \cdot \theta_j^t > \theta_i^t$ then

$$\begin{aligned} l_i^{t+1} &= l_j^t \\ \theta_i^{t+1} &= g(\|C_i^t - C_j^t\|_2) \cdot \theta_j^t \end{aligned}$$

Stroke-Based:
A Single Grow-Cut Pass

- Note that g is a decreasing function. For example:

$$g(x) = \begin{cases} 1 - |x| & \text{if } |x| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$



- This means that if the two pixels, which we compare, are close in intensity/color values they should have the same label l.

- They should also share the same label the neighbors have a higher strength!

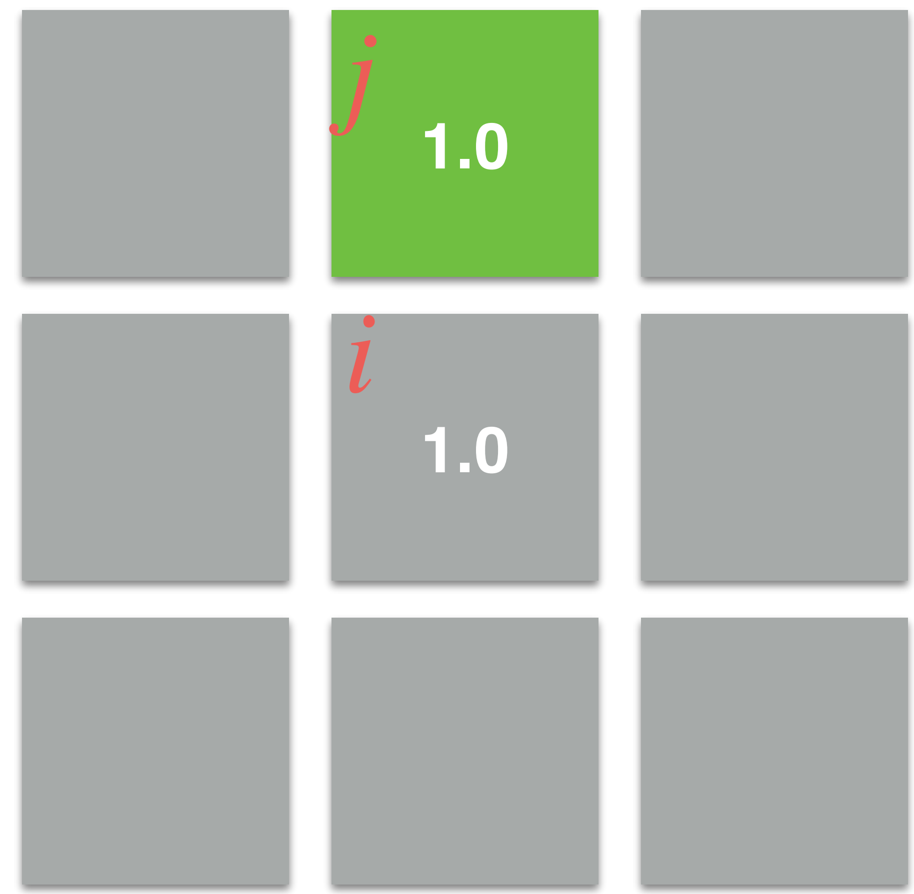
Example 1 Flat Area: Switching Labels

Stroke-Based: Example 1 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



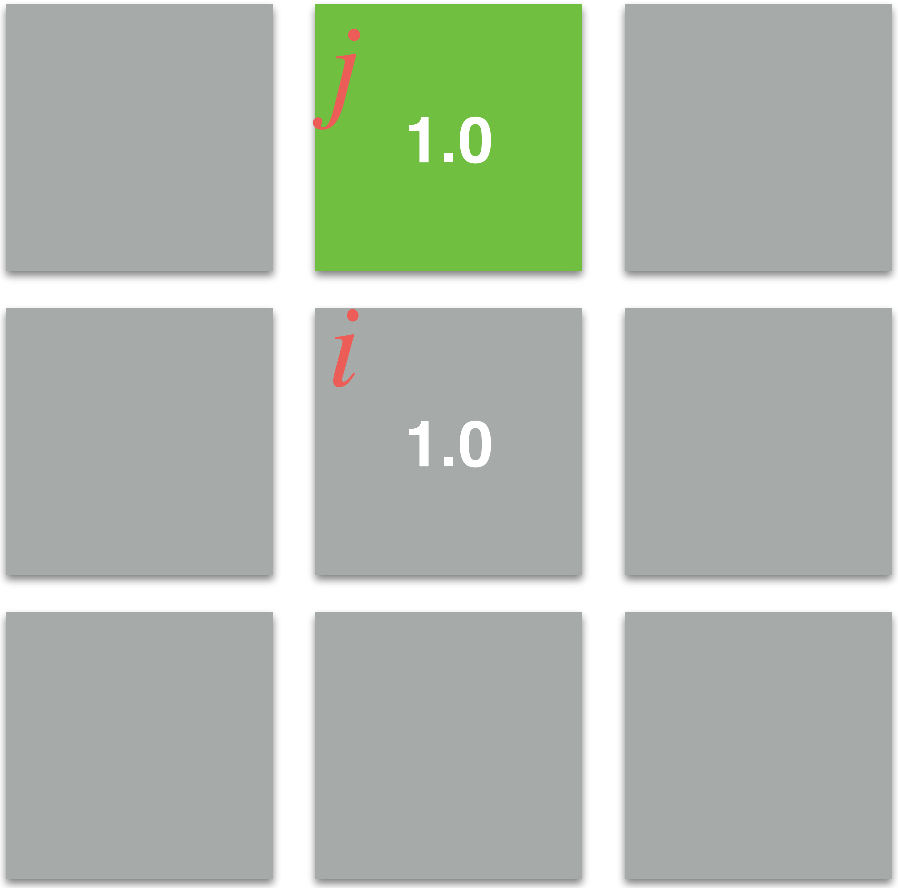
$$g(C_i^t - C_j^t) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 1 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0{,}0.9{,}1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1{,}0.8{,}1.0 >$$



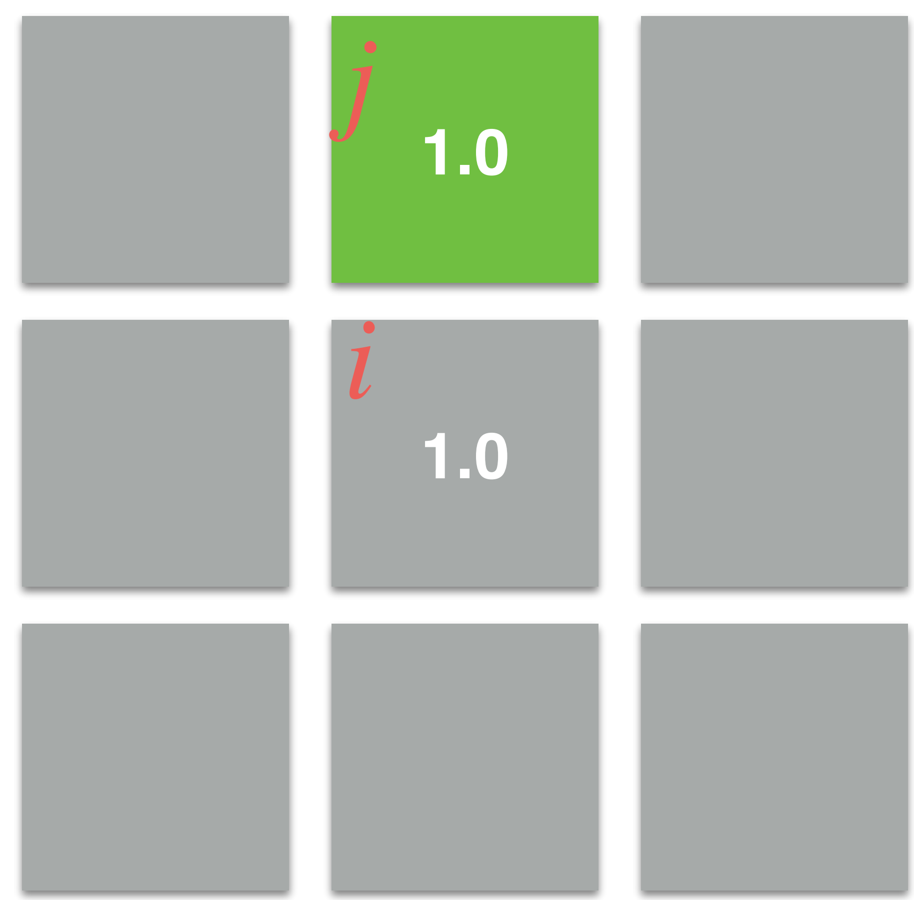
$$g(1.0 - 1.0) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based:
Example 1 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



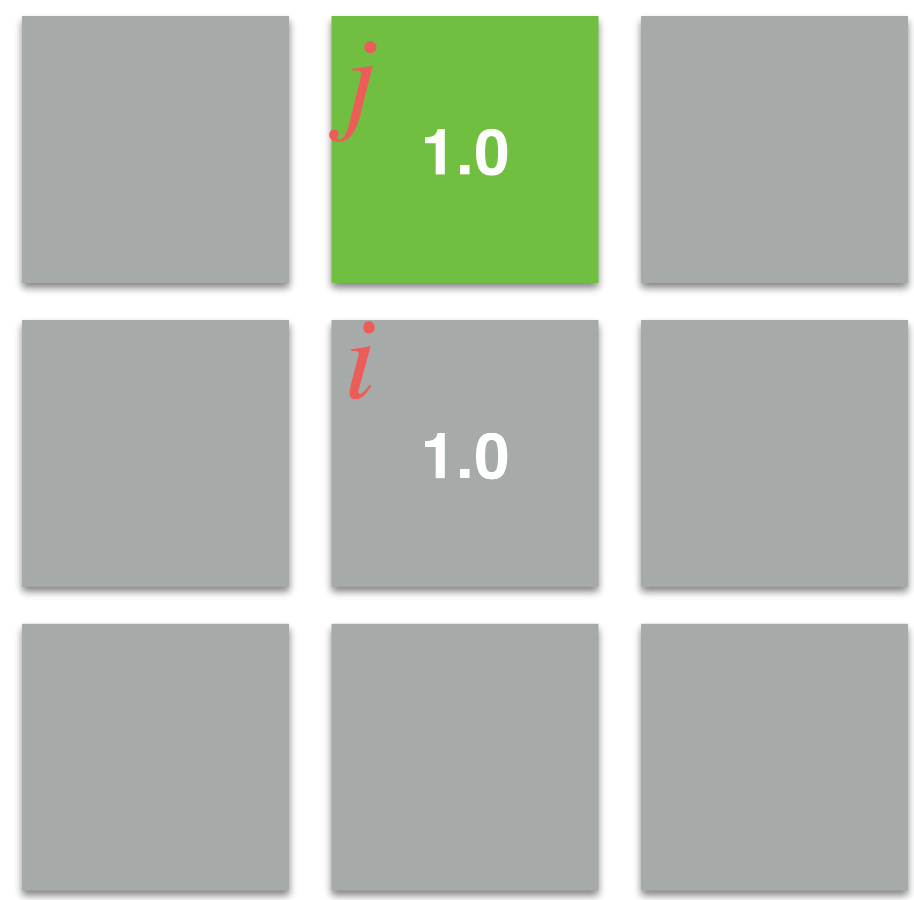
$$g(0.0) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 1 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



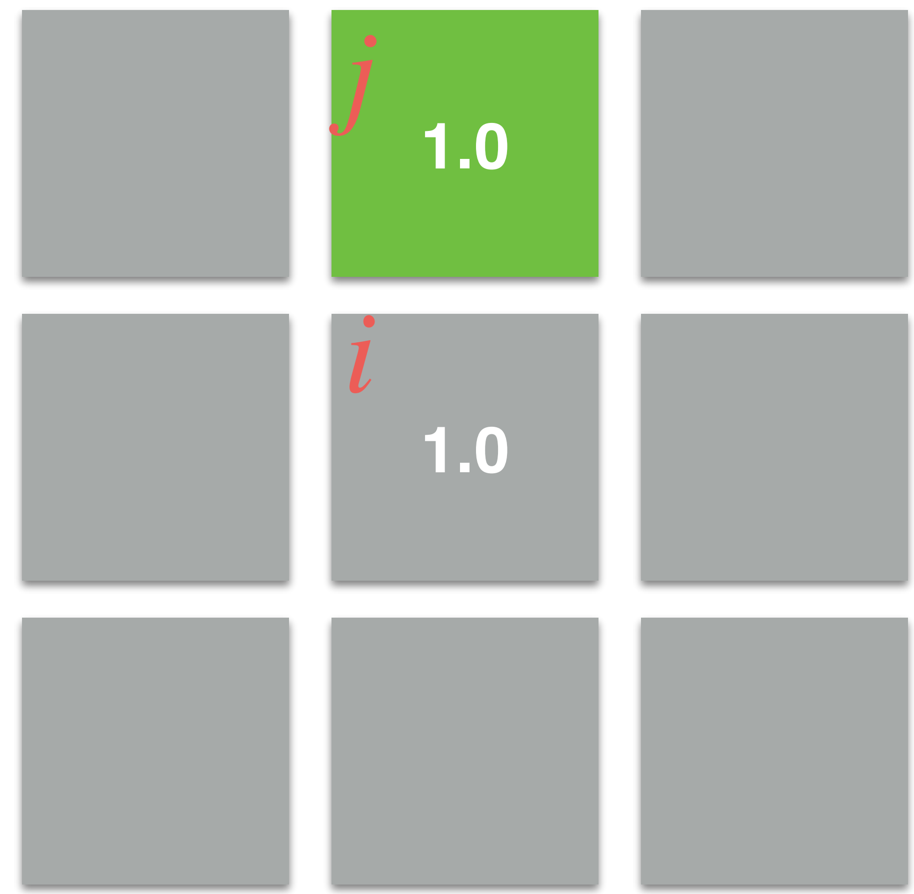
$$1.0 \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 1 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0{,}0.9{,}1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1{,}0.8{,}1.0 >$$



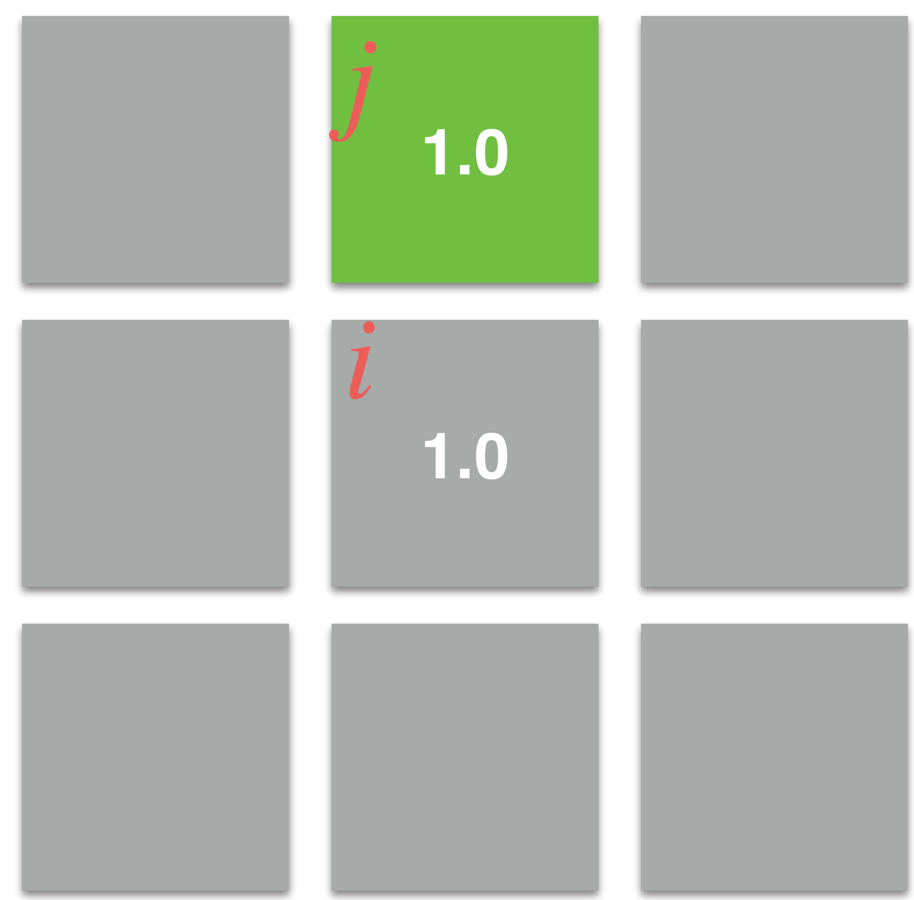
$0.9 > 0.8$

Stroke-Based: Example 1 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



$$0.9 > 0.8$$

0.9 is greater than 0.8. So, so we assign the value l_j^t to l_i^{t+1}.

$$l_i^{t+1} = l_j^t = 0!$$

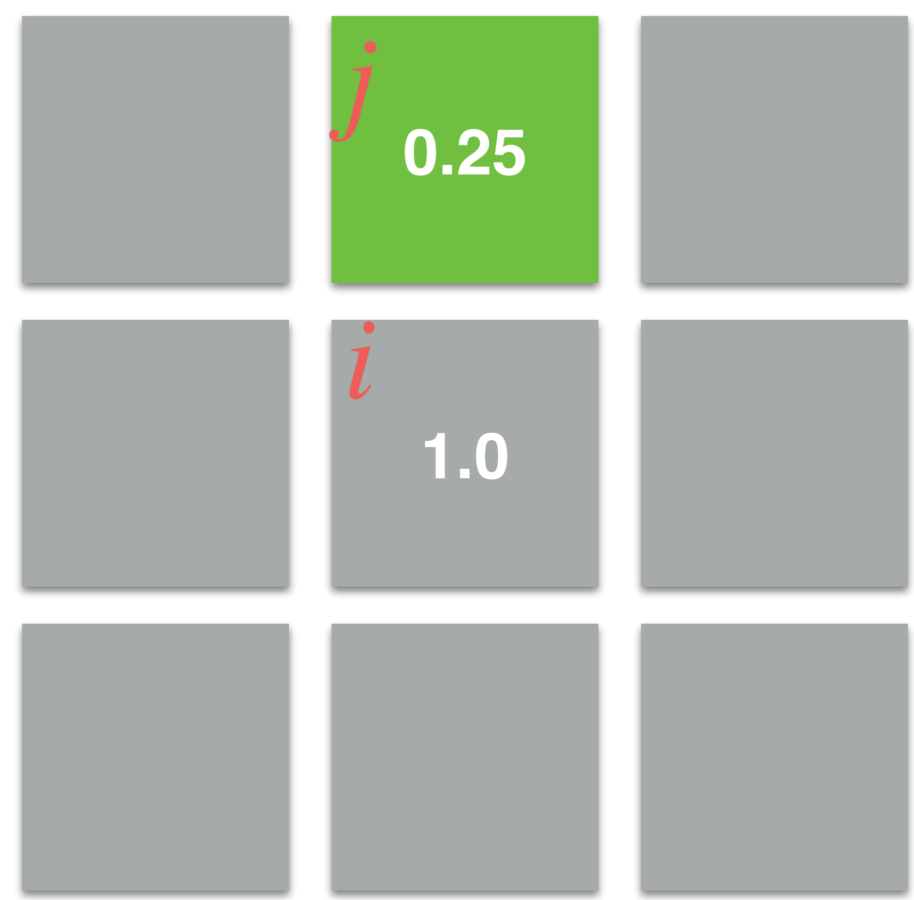
Example 2 Edge: Maintaining Labels

Stroke-Based: Example 2 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0{,}0.9{,}0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1{,}0.8{,}1.0 >$$



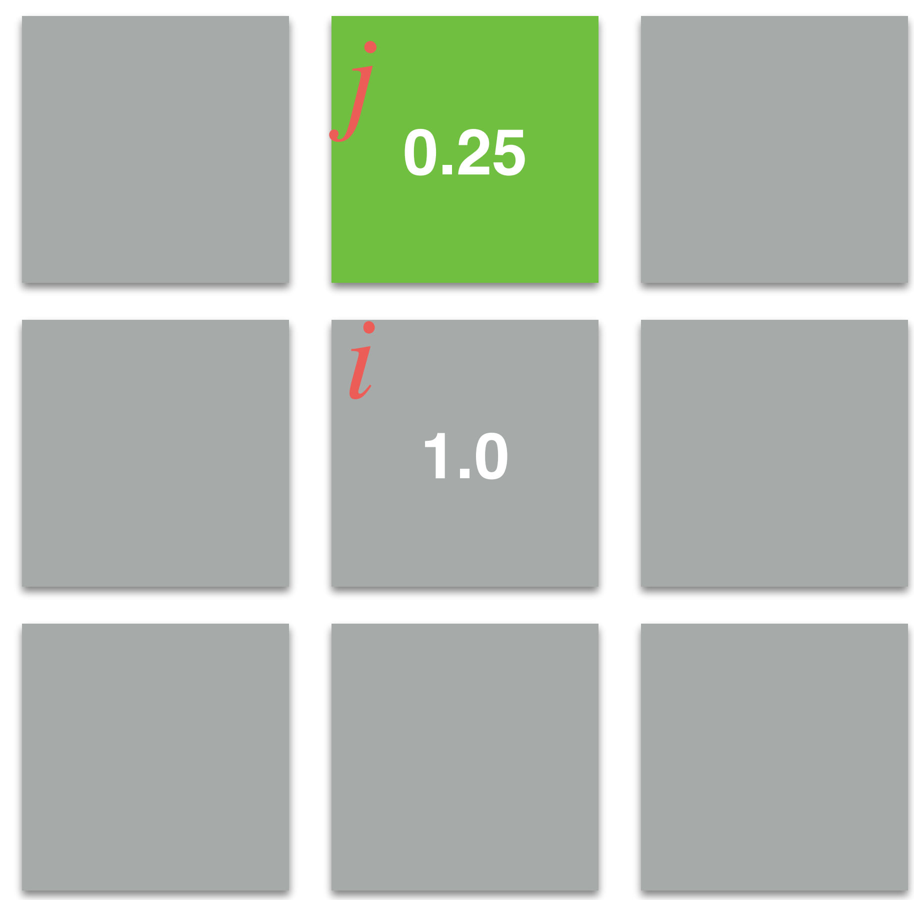
$$g(C_i^t - C_j^t) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based:
Example 2 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



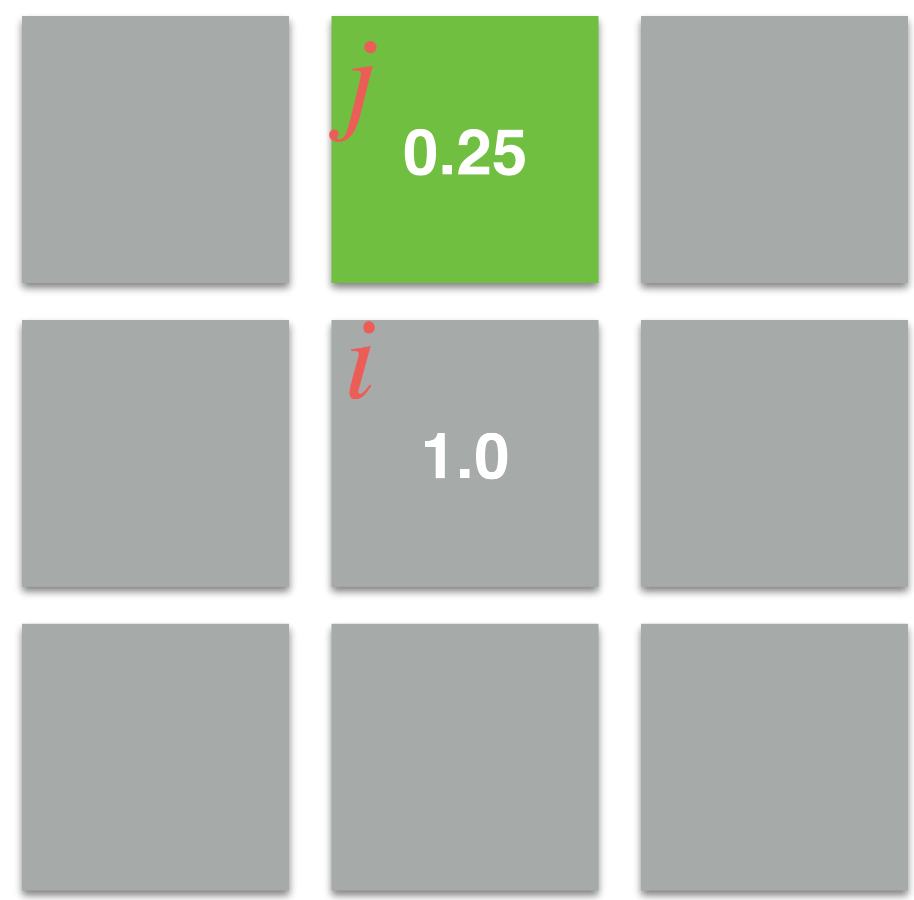
$$g(1.0 - 0.25) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 2 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0{,}0.9{,}0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1{,}0.8{,}1.0 >$$



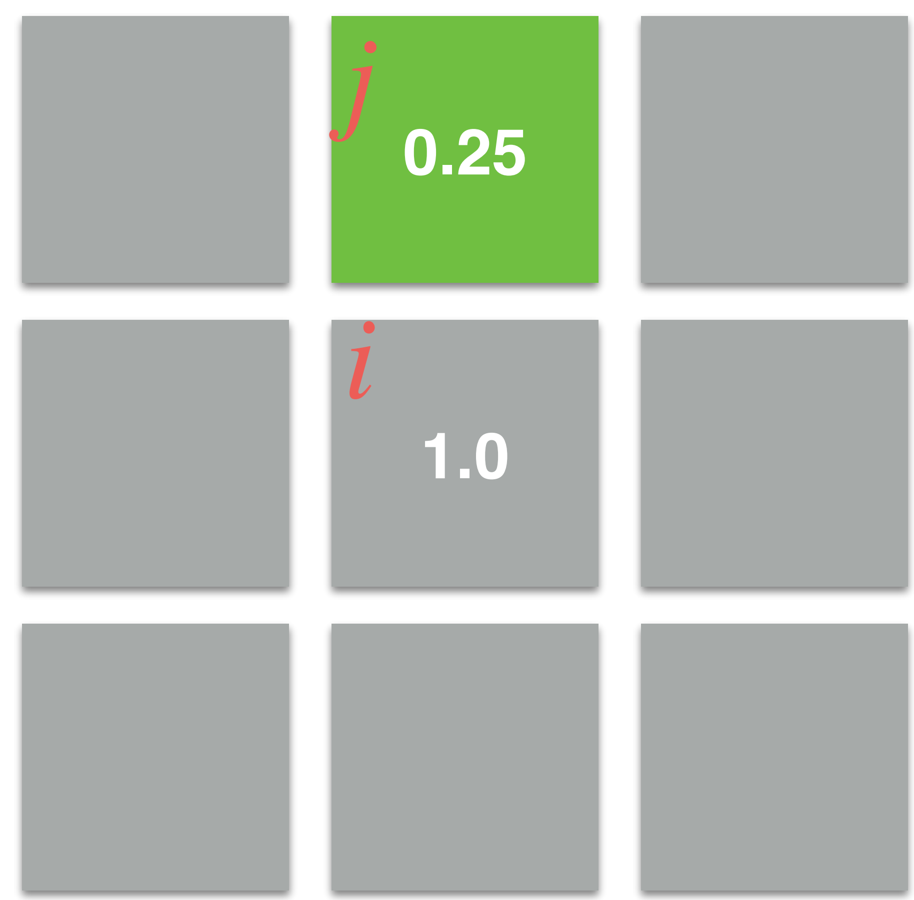
$$g(0.75) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 2 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



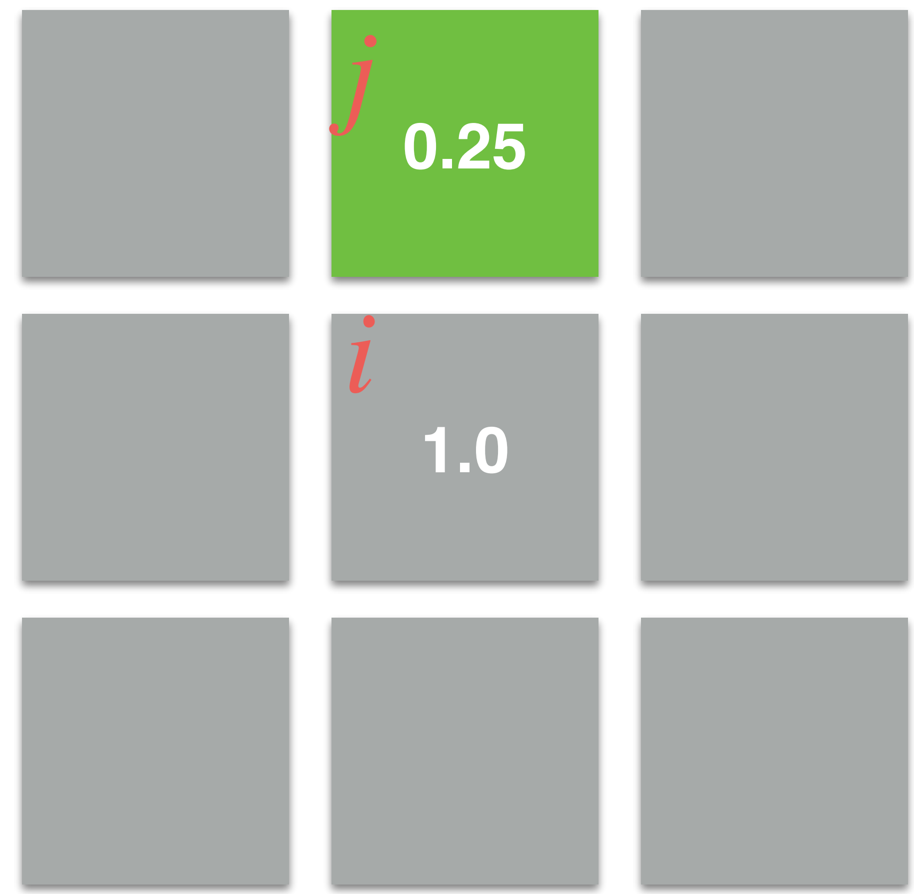
$$0.25 \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 2 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



$$0.25 \cdot 0.9 > 0.8$$

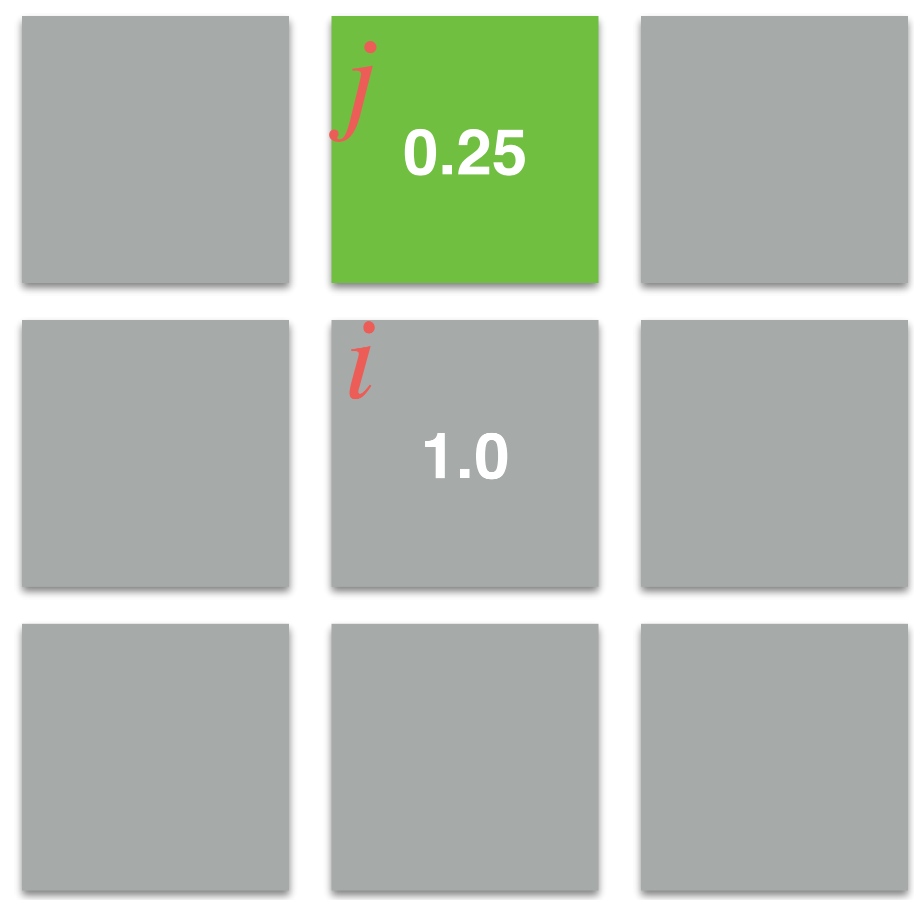
$$0.225 > 0.8$$

Stroke-Based: Example 2 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



$$0.25 \cdot 0.9 > 0.8$$

$$0.225 > 0.8$$

0.225 is not greater than 0.8. So, l_i^{t+1} **for this neighbor** remains the previous value!

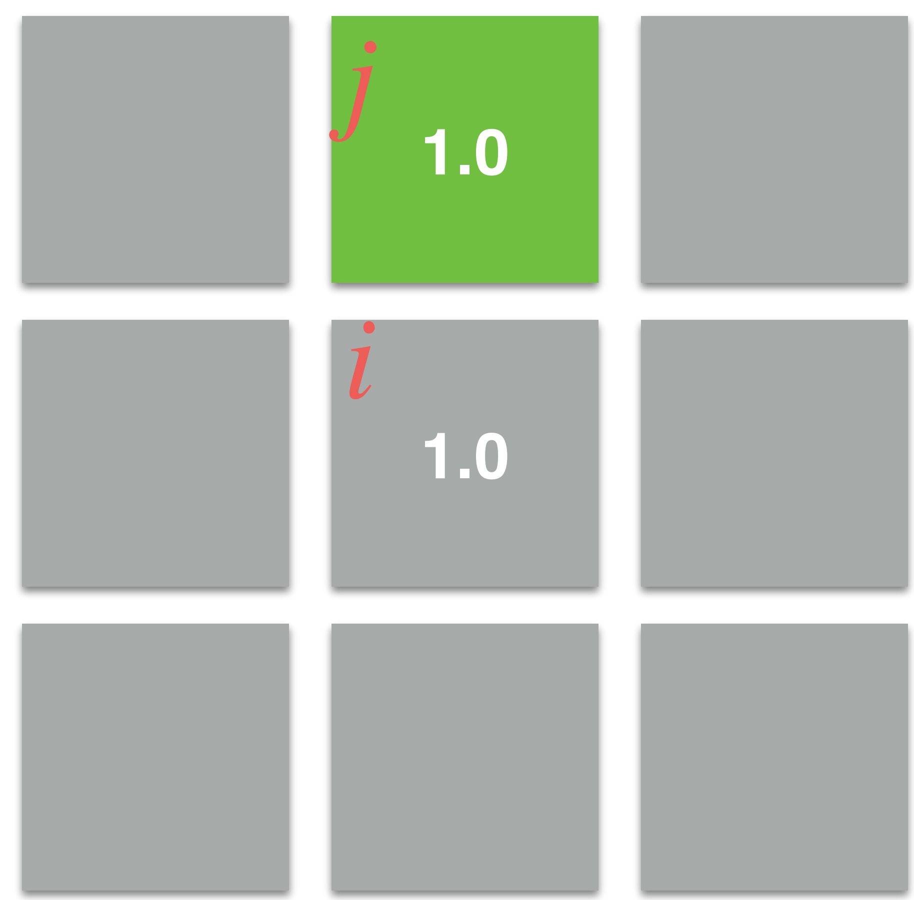
Example 3 Flat Area: Maintaining Labels

Stroke-Based:
Example 3 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.7,1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



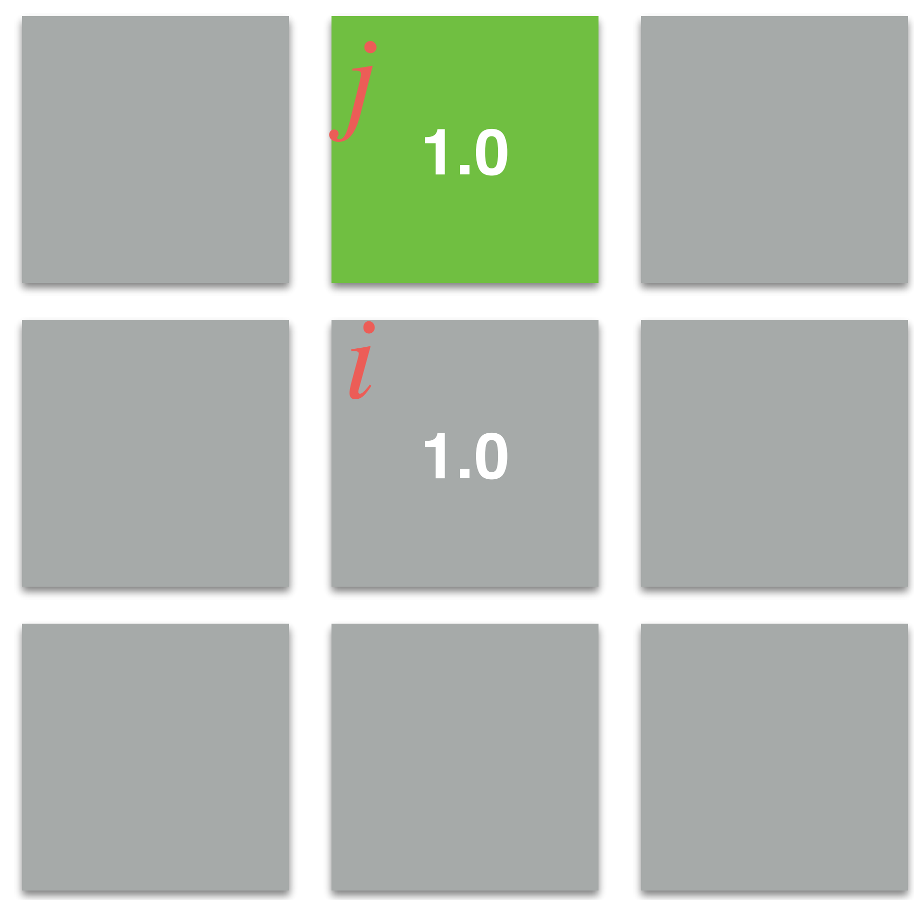
$$g(C_i^t - C_j^t) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 3 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.7,1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



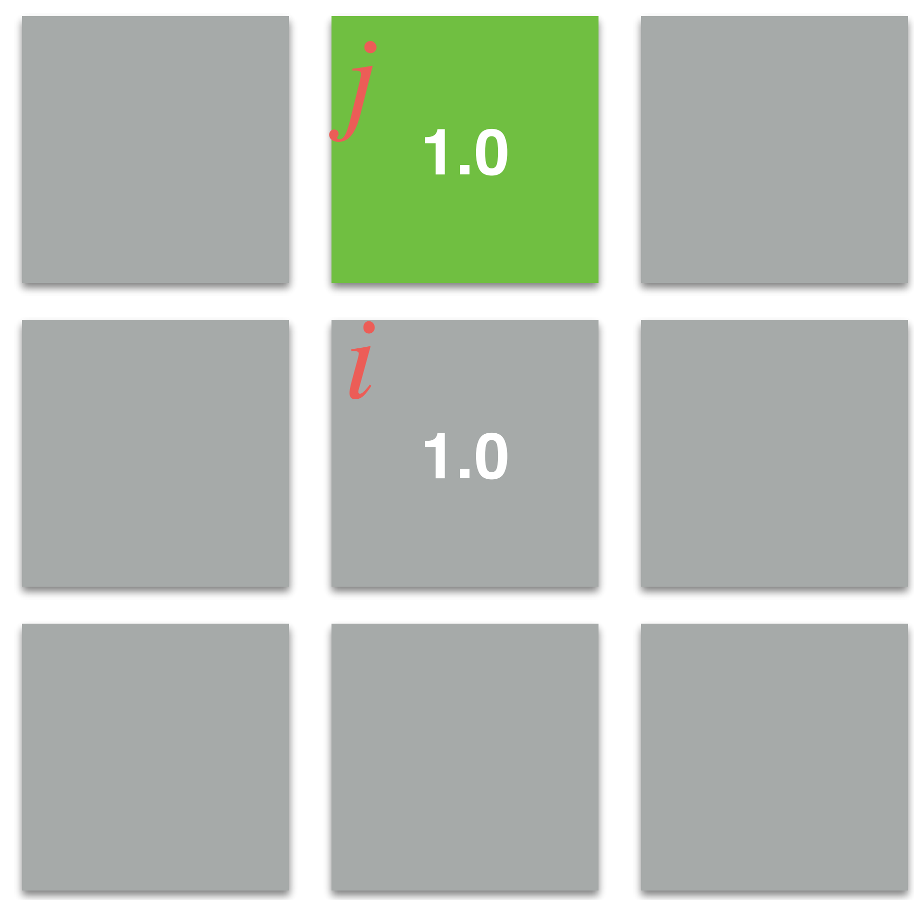
$$g(1.0 - 1.0) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 3 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.7,1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



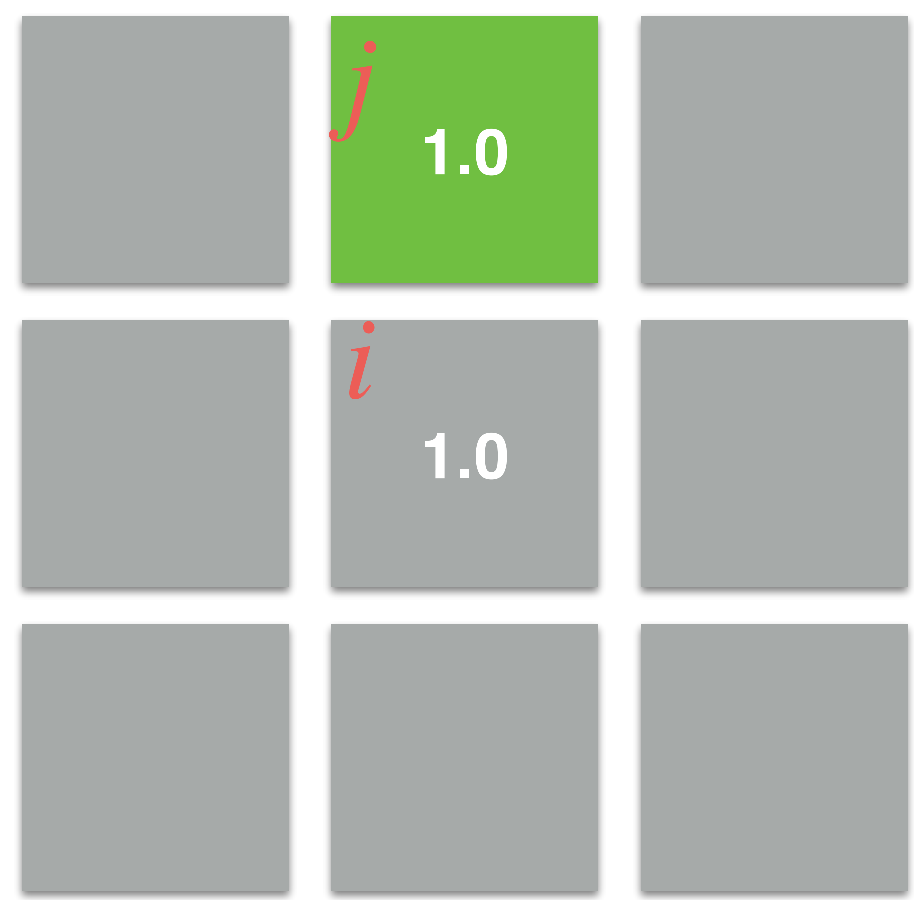
$$g(0.0) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 3 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.7,1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.8,1.0 >$$



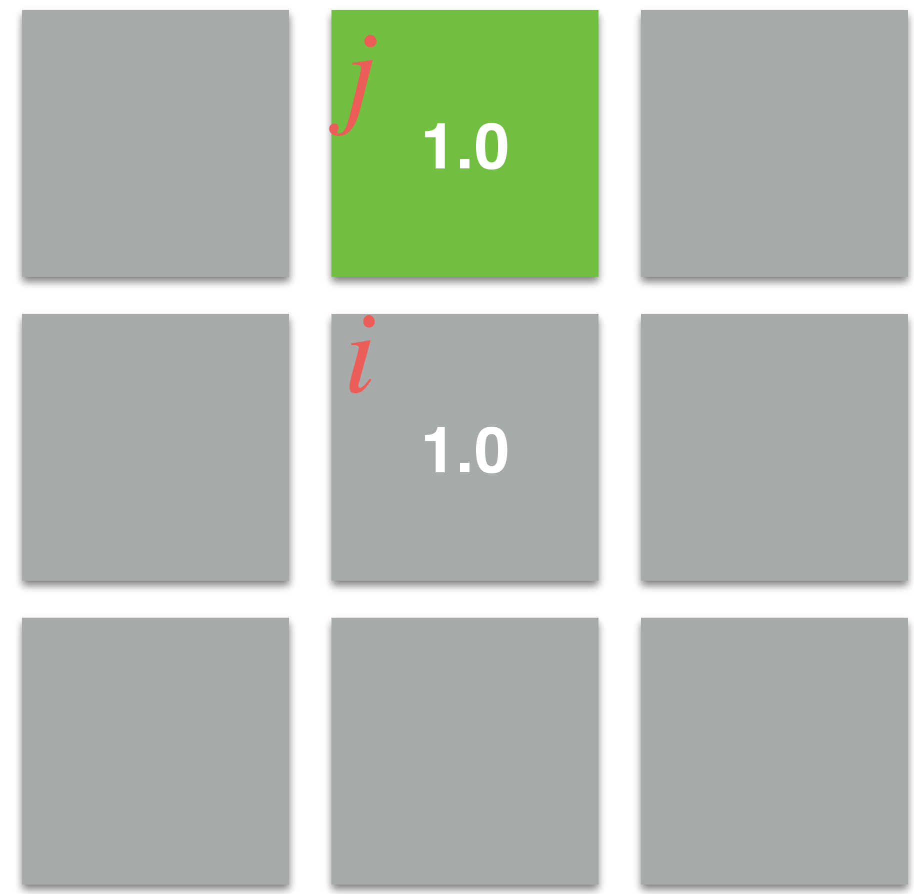
$$1.0 \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 3 - Flat Areas

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0, 0.7, 1.0 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1, 0.8, 1.0 >$$



$$1.0 \cdot 0.7 > 0.8$$

$$0.7 > 0.8$$

0.7 is not greater than 0.8. So, l_i^{t+1} **for this neighbor** remains the previous value!

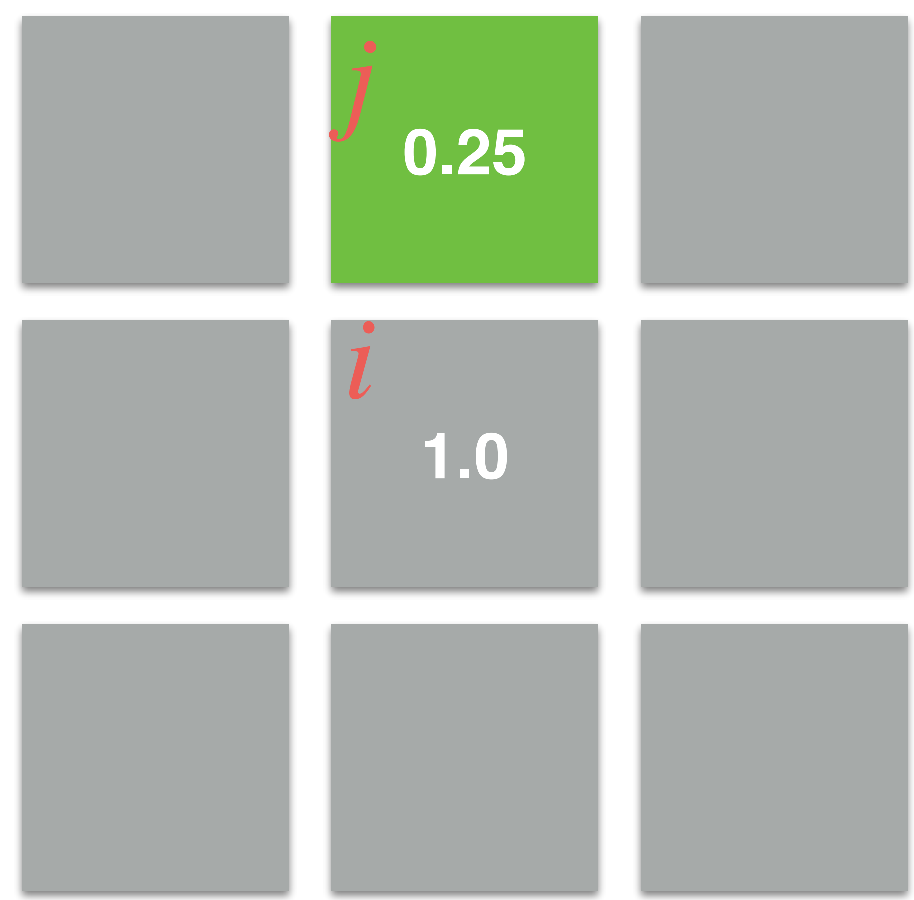
Example 4 Edges: Switching Labels

Stroke-Based: Example 4 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.1,1.0 >$$



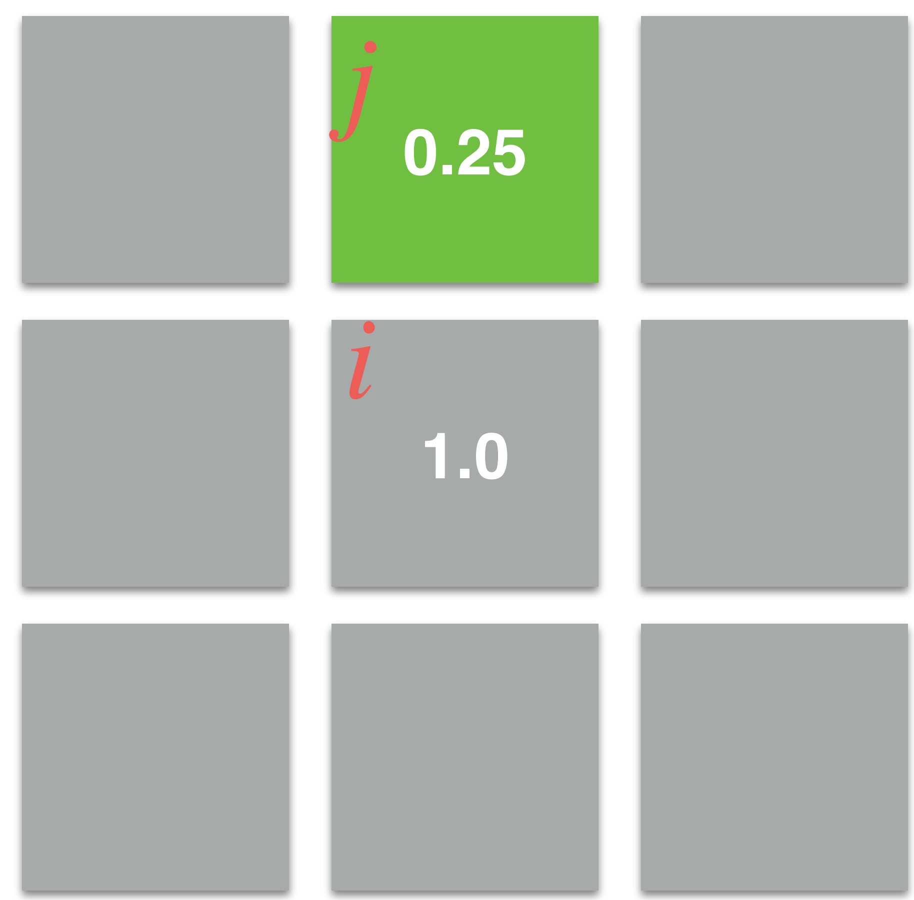
$$g(C_i^t - C_j^t) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 4 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.1,1.0 >$$



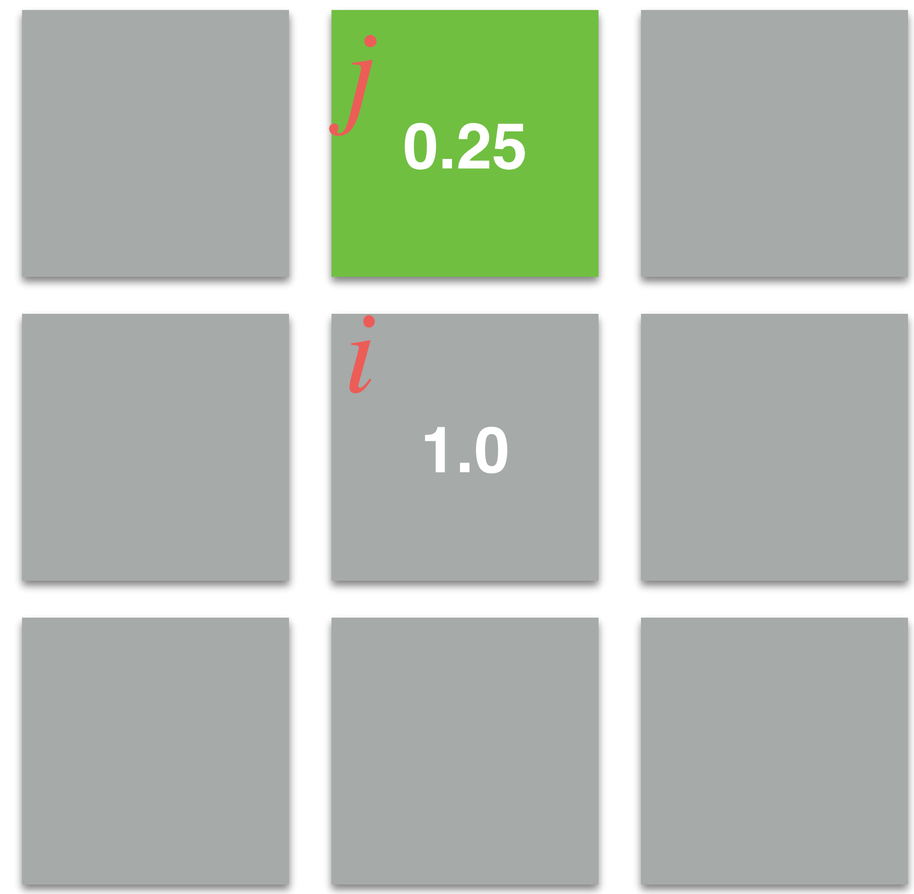
$$g(1.0 - 0.25) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 4 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.1,1.0 >$$



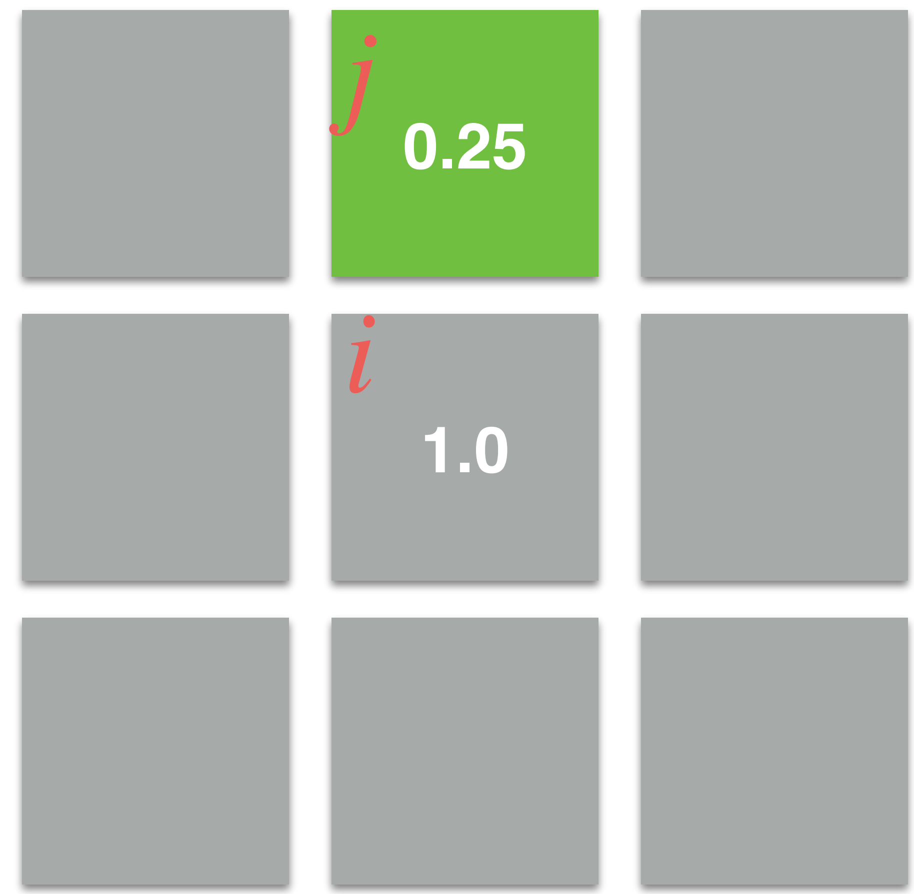
$$g(0.75) \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 4 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.1,1.0 >$$



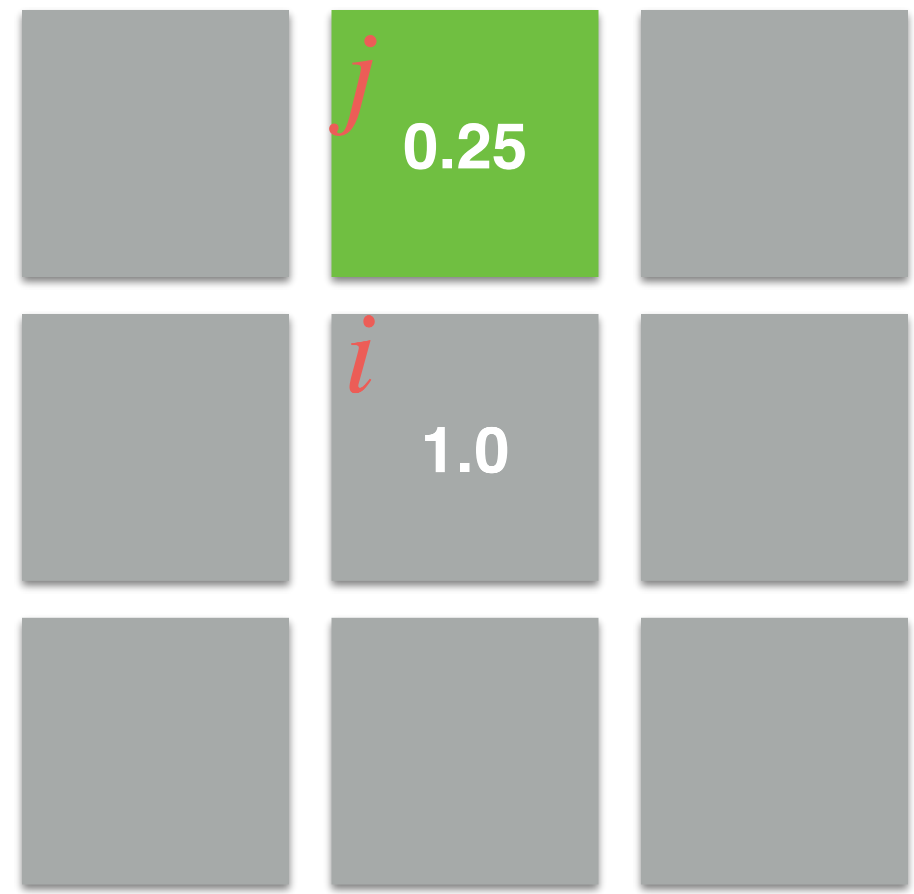
$$0.25 \cdot \theta_j^t > \theta_i^t$$

Stroke-Based: Example 4 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.1,1.0 >$$



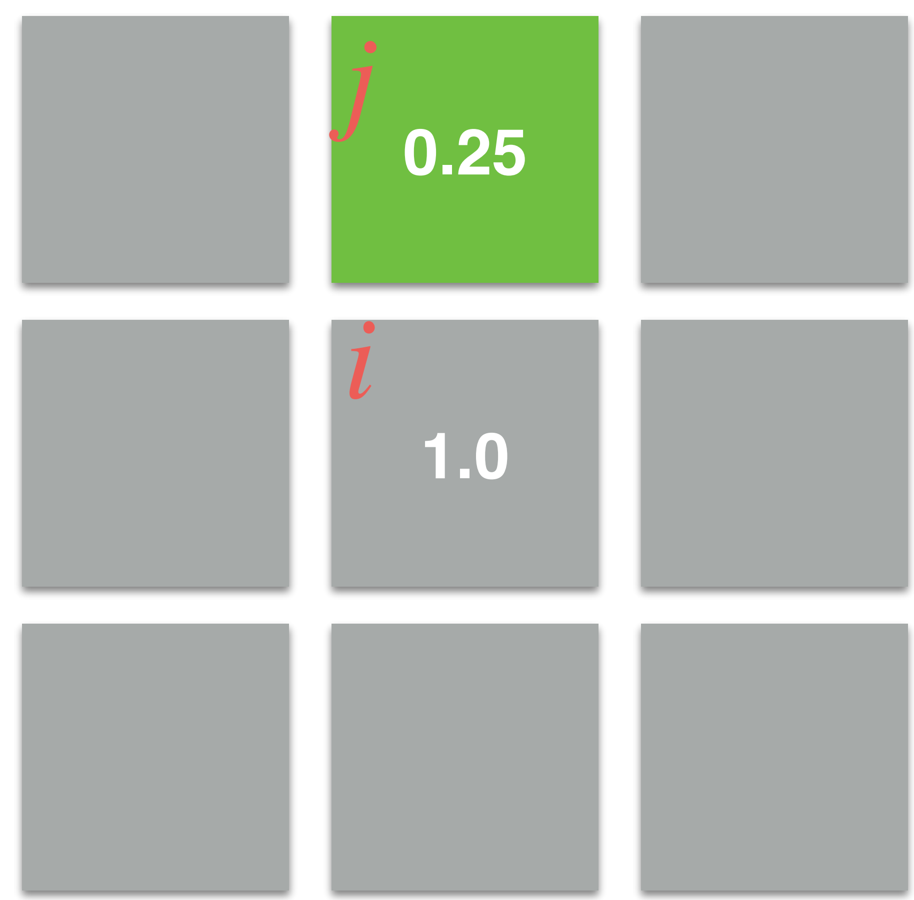
$$0.25 \cdot 0.9 > 0.1$$

Stroke-Based: Example 4 - Edges

Current neighbor

$$< l_j^t, \theta_j^t, C_j^t > = < 0,0.9,0.25 >$$

$$< l_i^t, \theta_i^t, C_i^t > = < 1,0.1,1.0 >$$



0.225 is greater than 0.1. So, so we assign the value l_j^t to l_i^{t+1}.

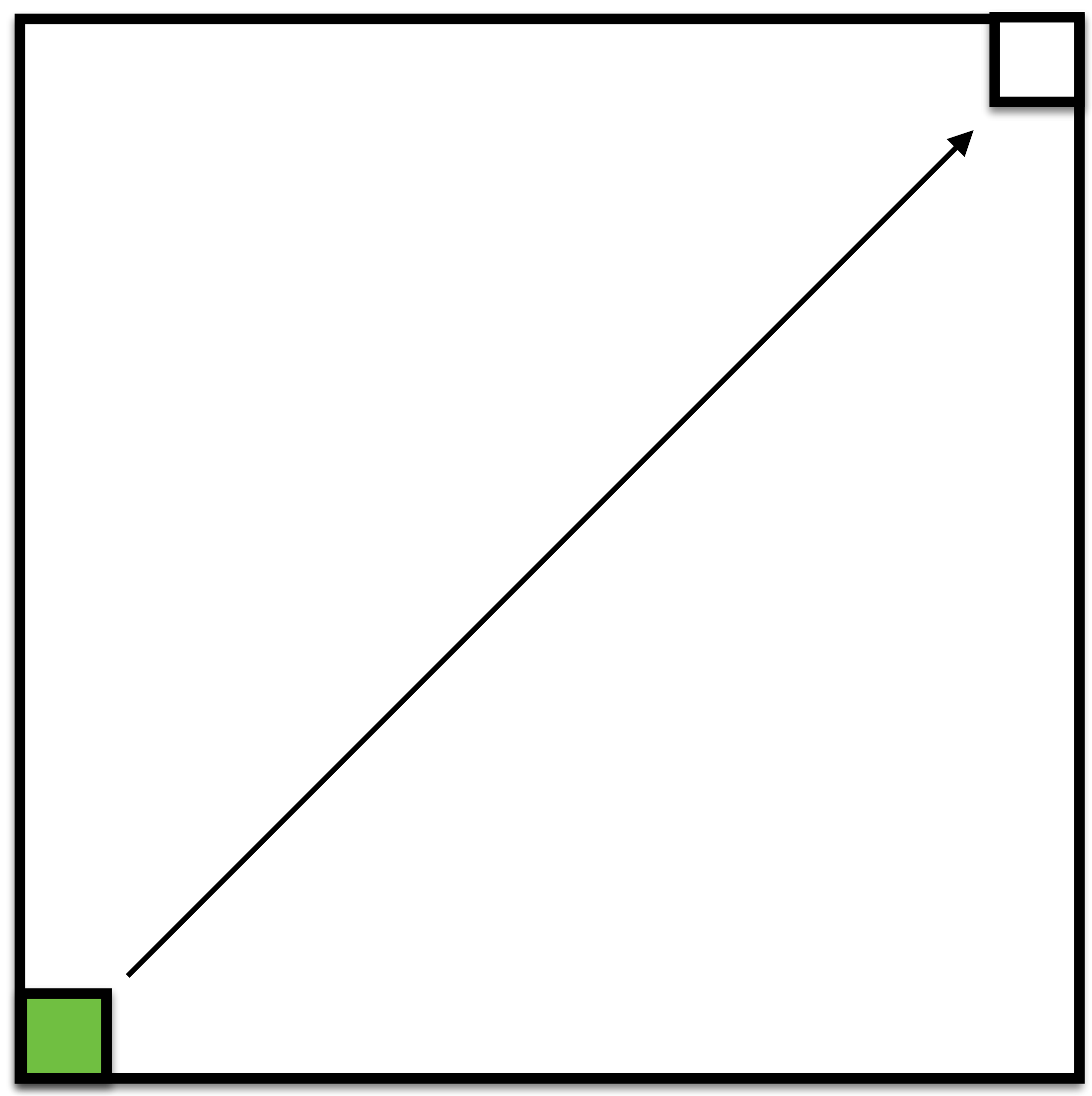
$$0.225 > 0.1$$

$$l_i^{t+1} = l_j^t = 0!$$

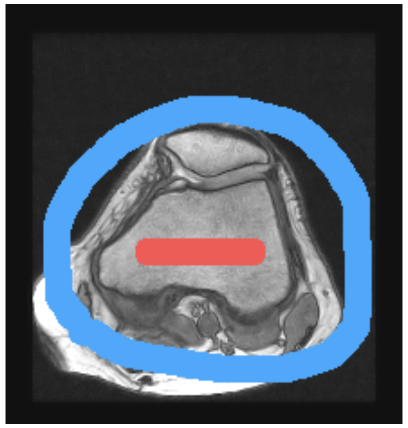
Stroke-Based: Grow-Cut

- Stopping criteria:
    - This process is iterated until either convergence; i.e., ***no changes in the labels***!
    - Labels have been propagated for enough iterations; e.g., the *number of pixels of the diagonal*. This trick is helpful for reducing the total computational time.

Stroke-Based: Grow-Cut

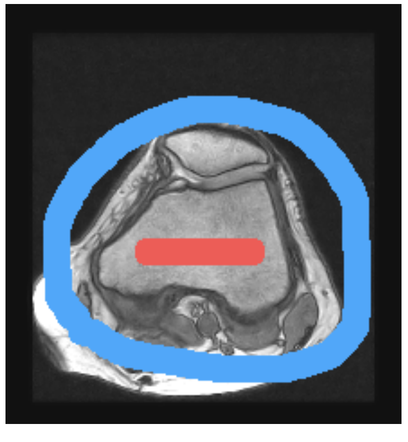


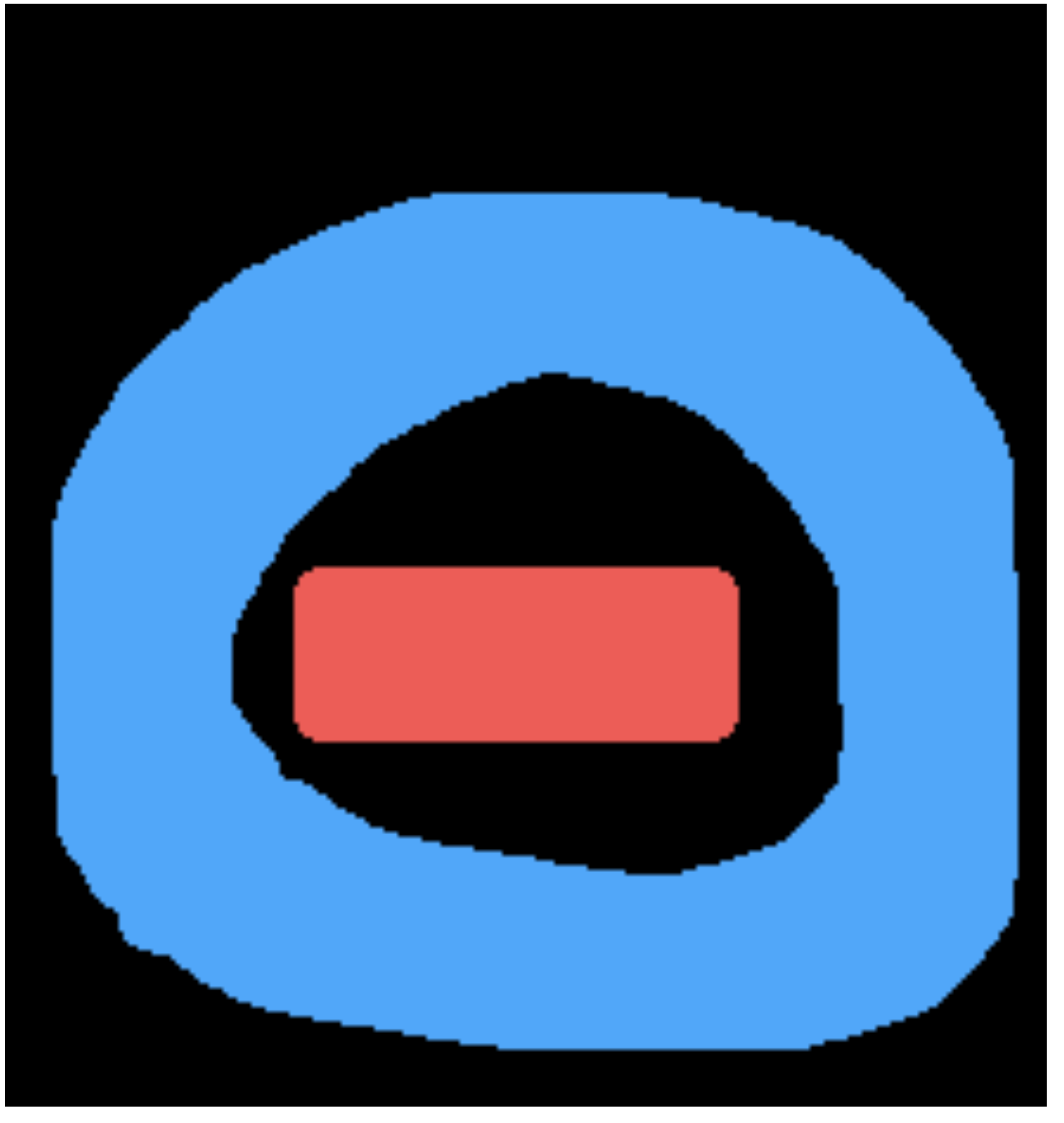
Stroke-Based: Grow-Cut Example



Iteration = 0

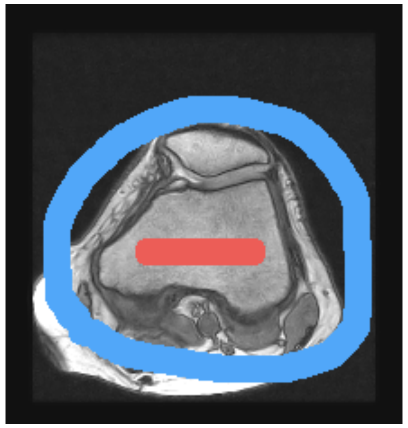
Stroke-Based: Grow-Cut Example

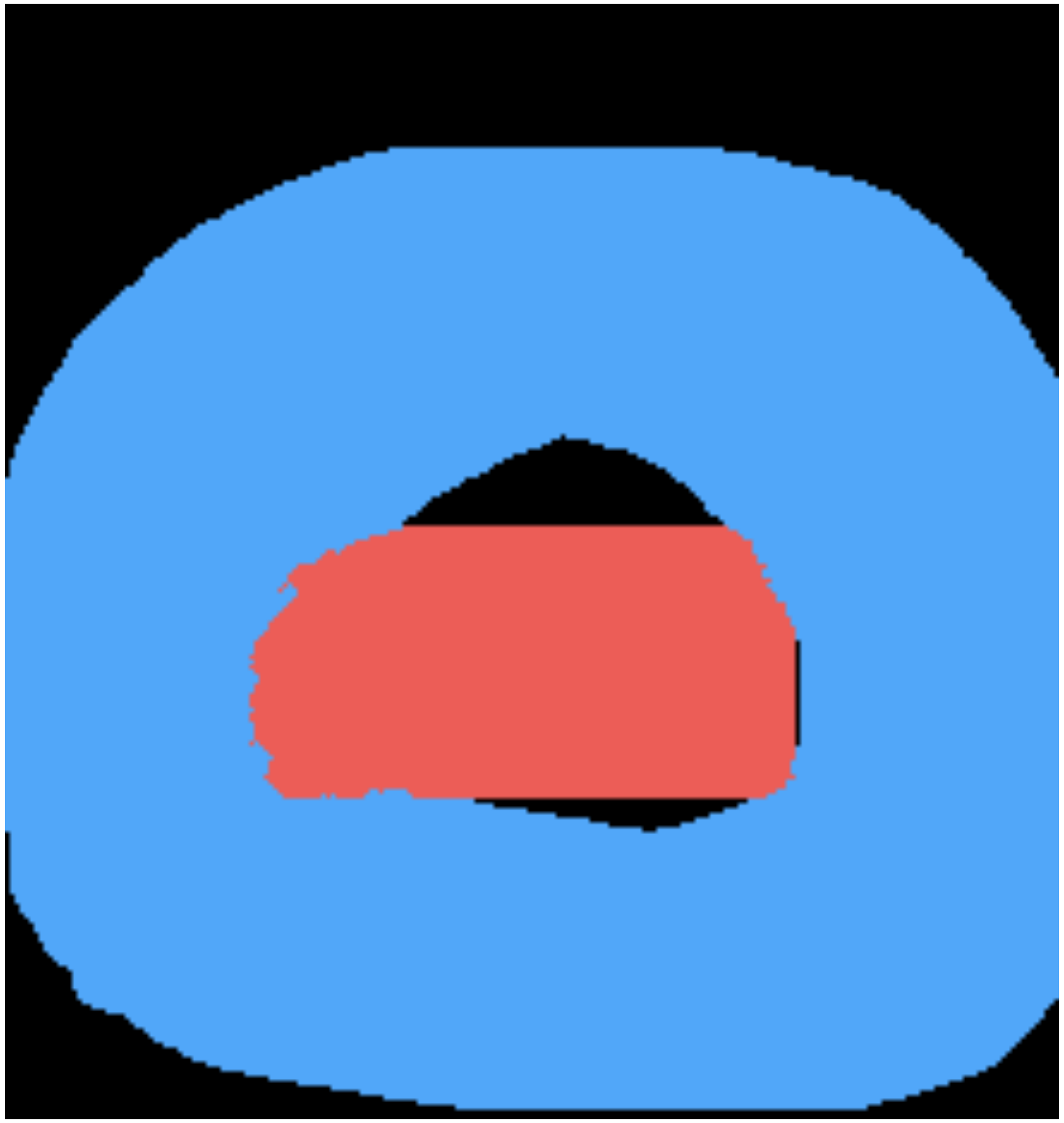




Iteration = 10

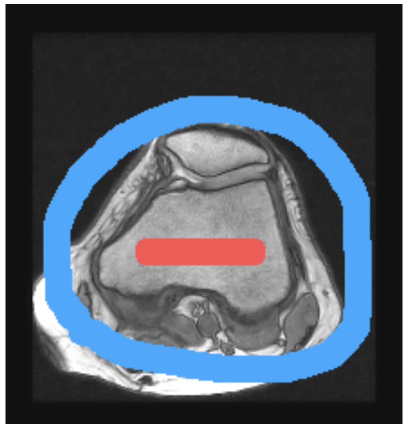
Stroke-Based: Grow-Cut Example

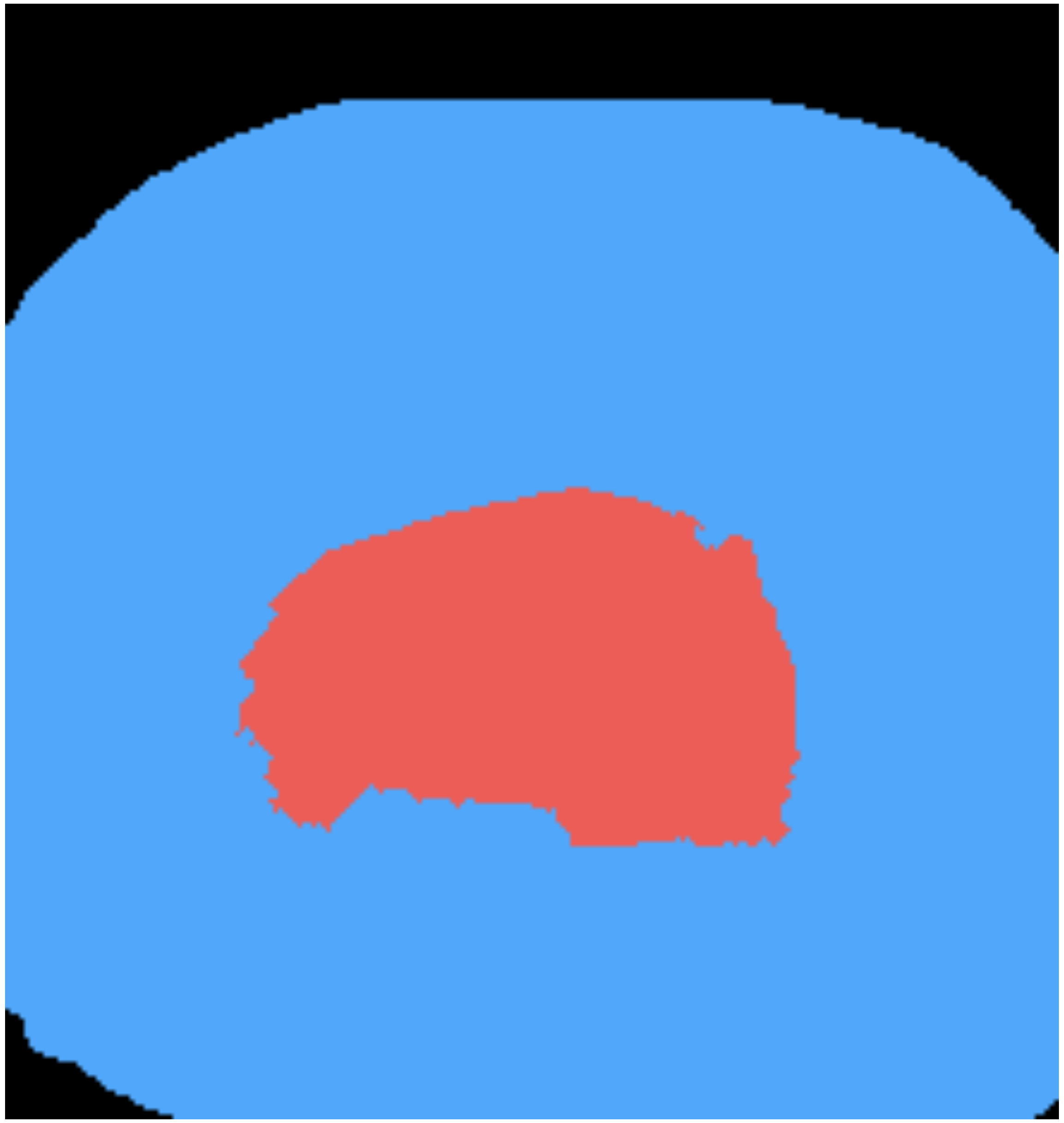




Iteration = 20

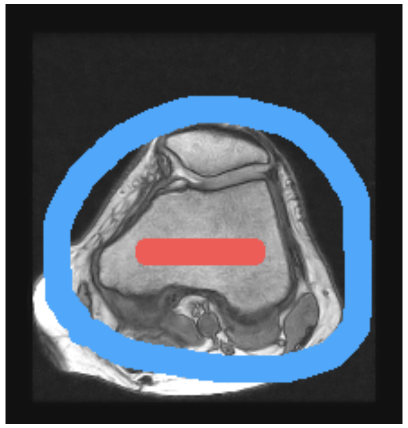
Stroke-Based: Grow-Cut Example

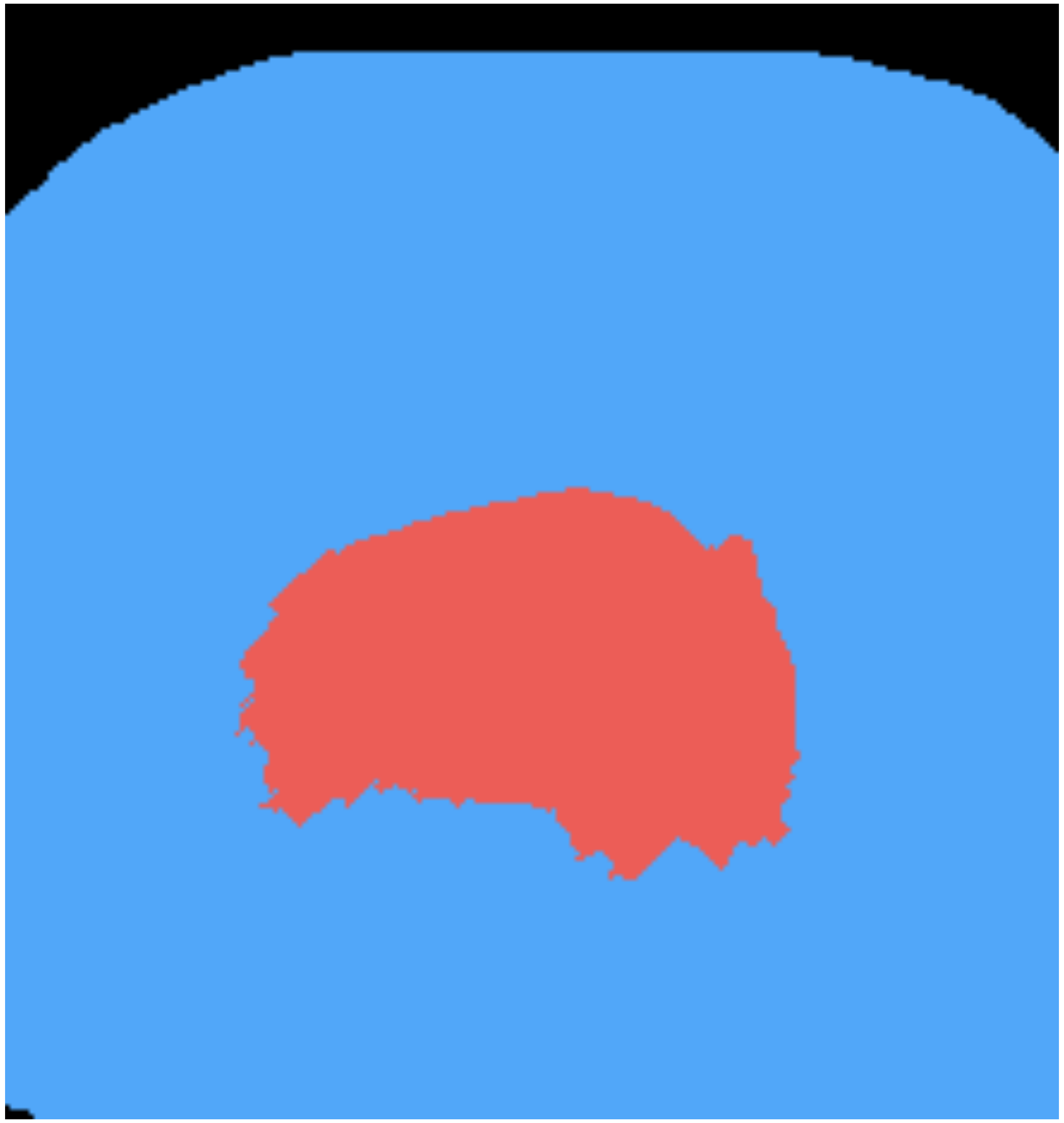




Iteration = 30

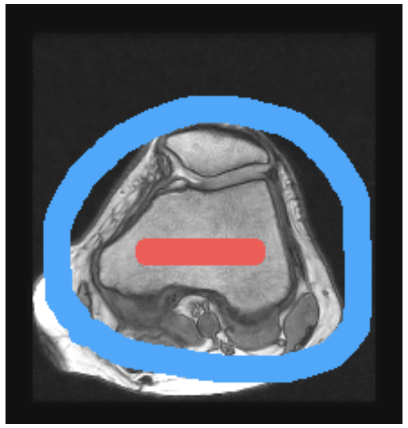
Stroke-Based: Grow-Cut Example

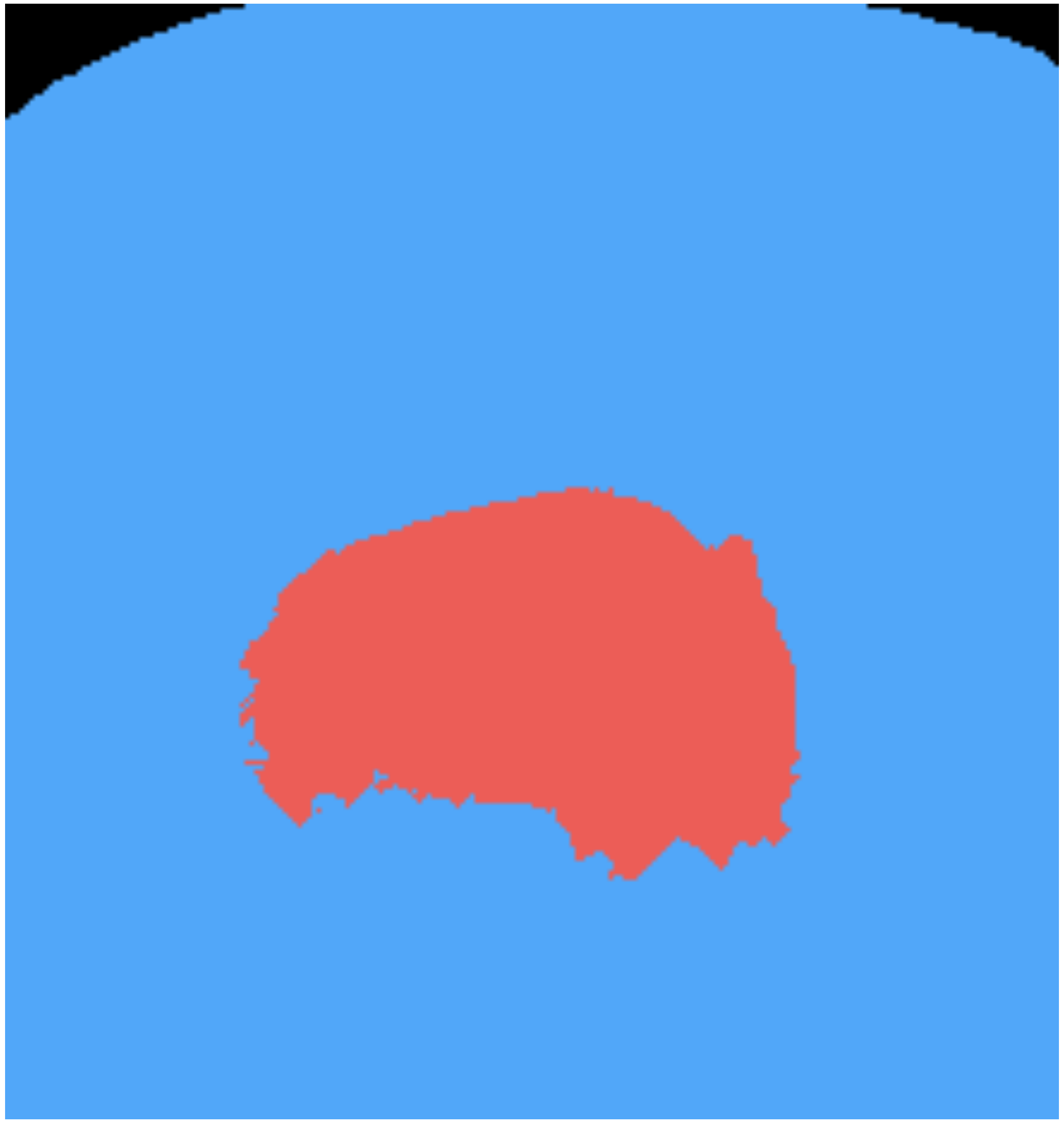




Iteration = 40

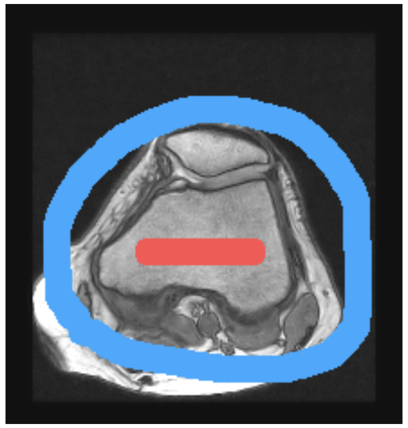
Stroke-Based: Grow-Cut Example





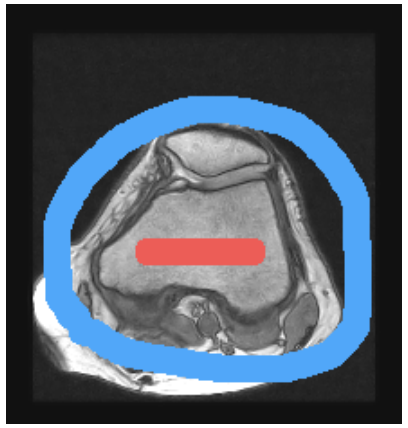
Iteration = 50

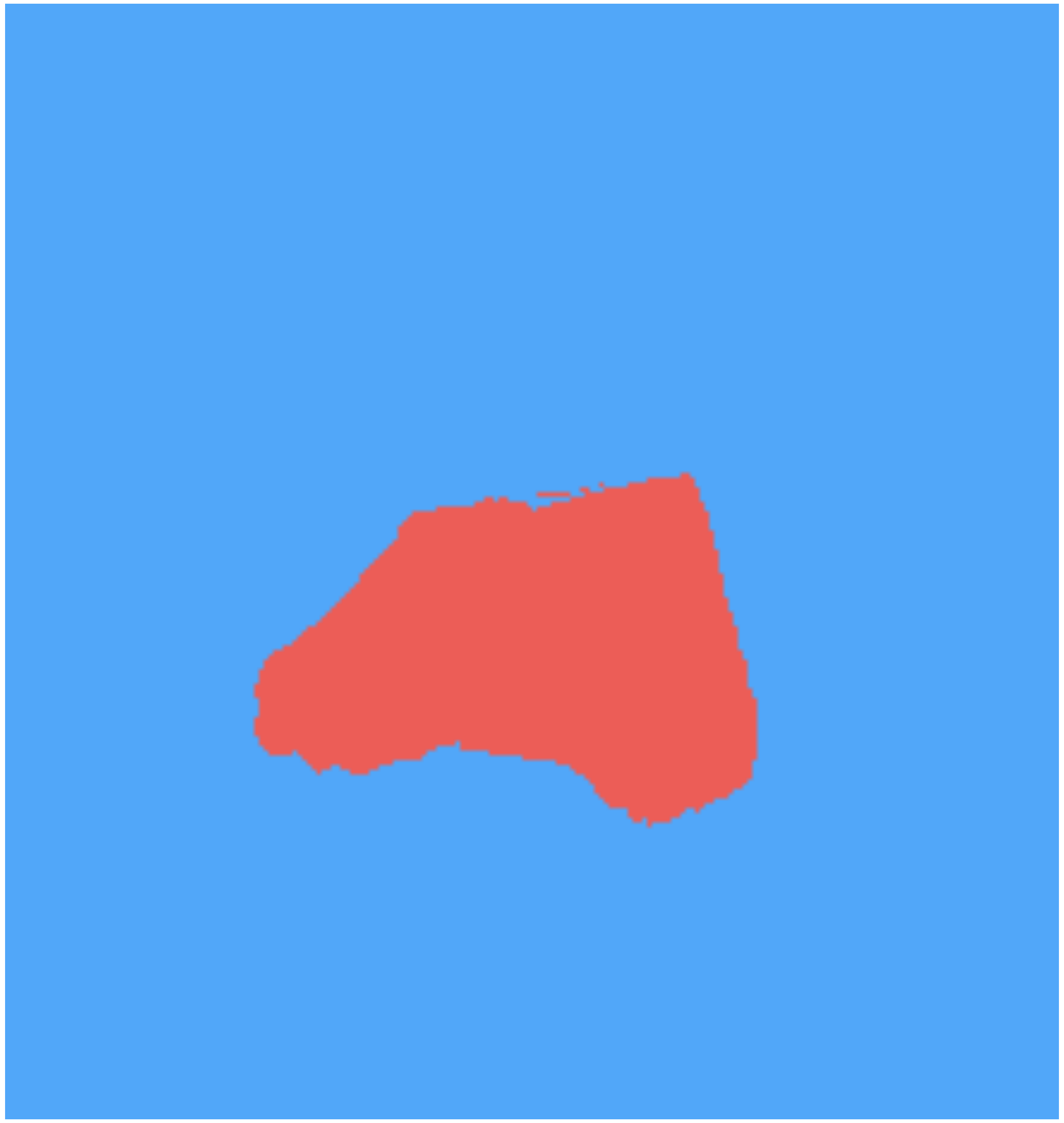
Stroke-Based: Grow-Cut Example



Iteration = 100

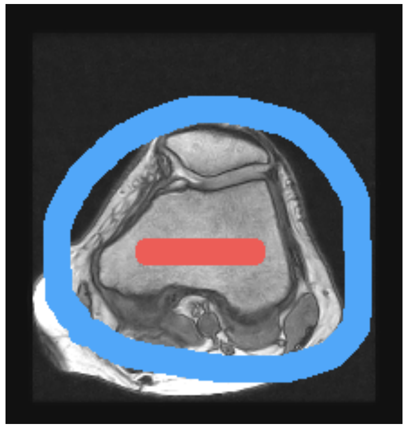
Stroke-Based: Grow-Cut Example

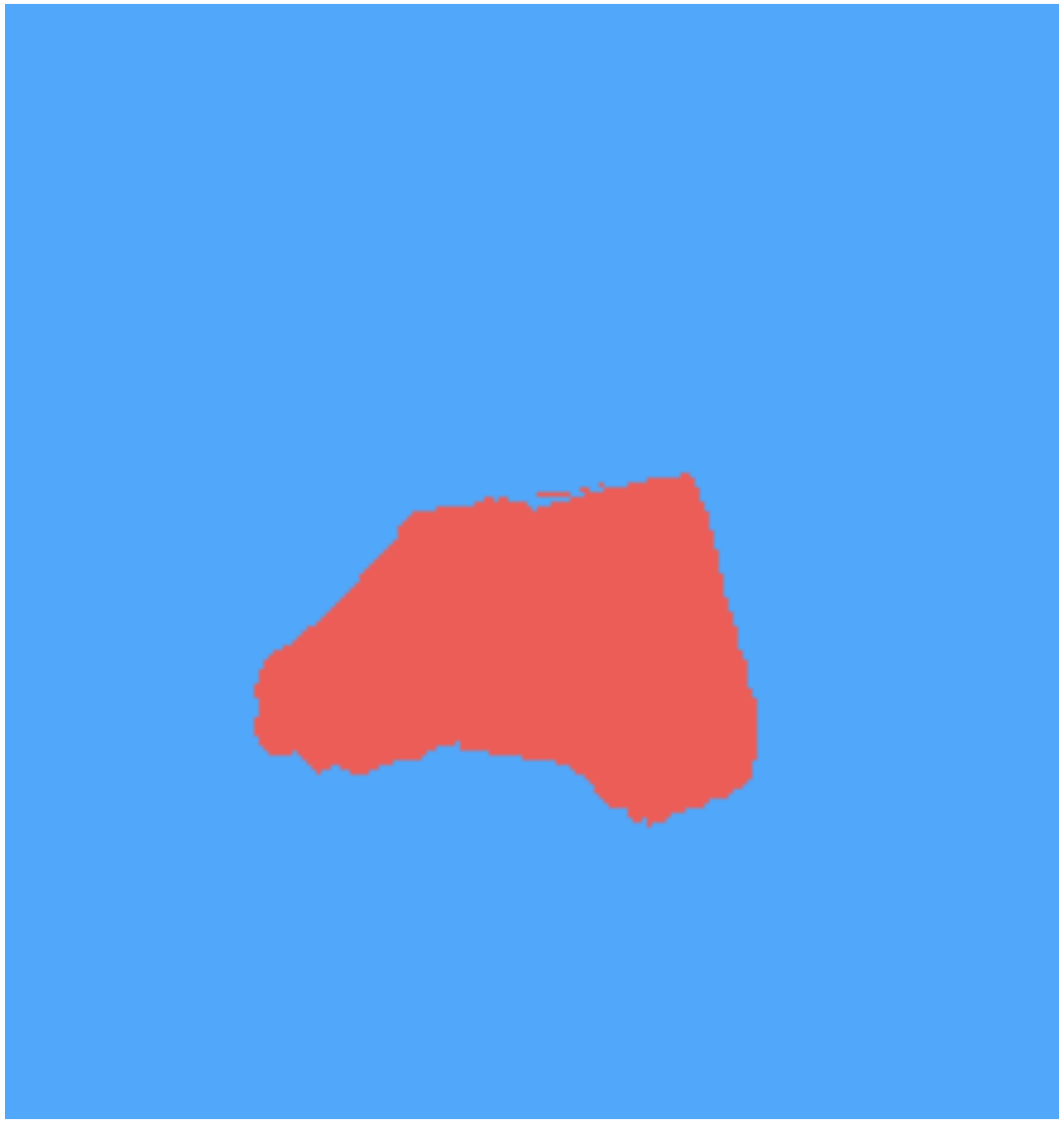




Iteration = 200

Stroke-Based: Grow-Cut Example





Iteration = 318

Stroke-Based: Grow-Cut

- This algorithm can be extended to 3D in a straightforward way, and it can be parallelized on the GPU.
- Disadvantages:
  - It is computationally slow!

that’s all folks!