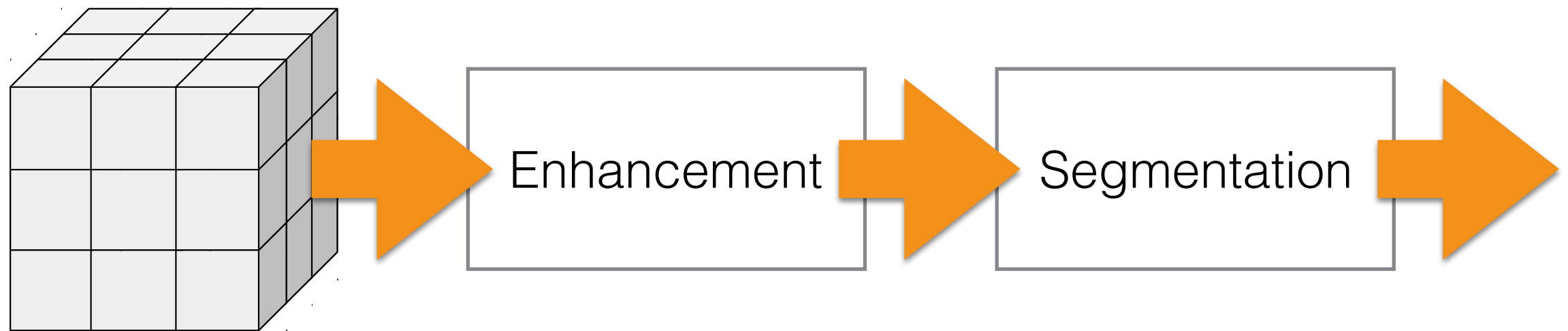


3D from Volume: Part II

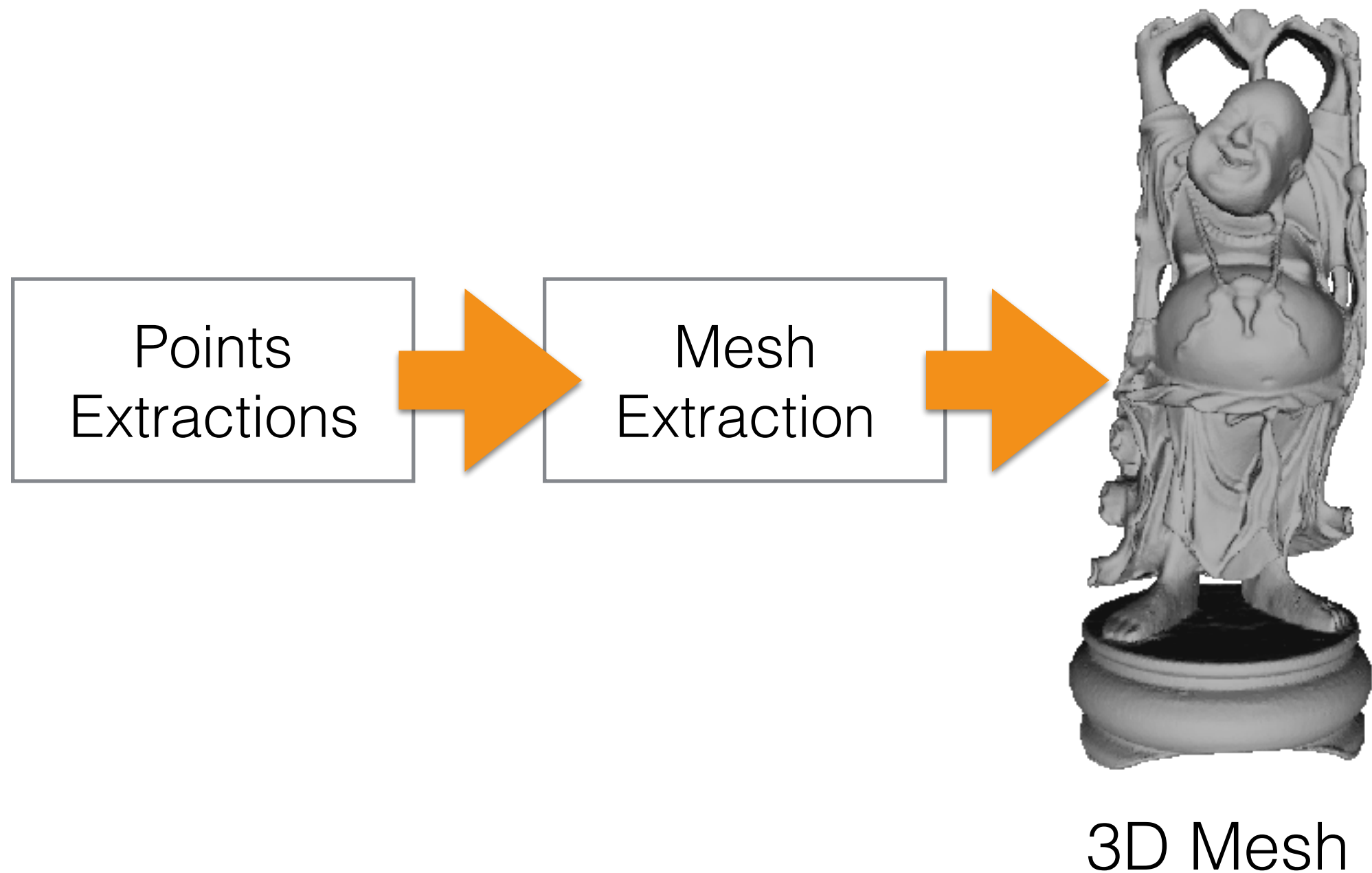
Dr. Francesco Banterle,
francesco.banterle@isti.cnr.it
banterle.com/francesco

The Processing Pipeline

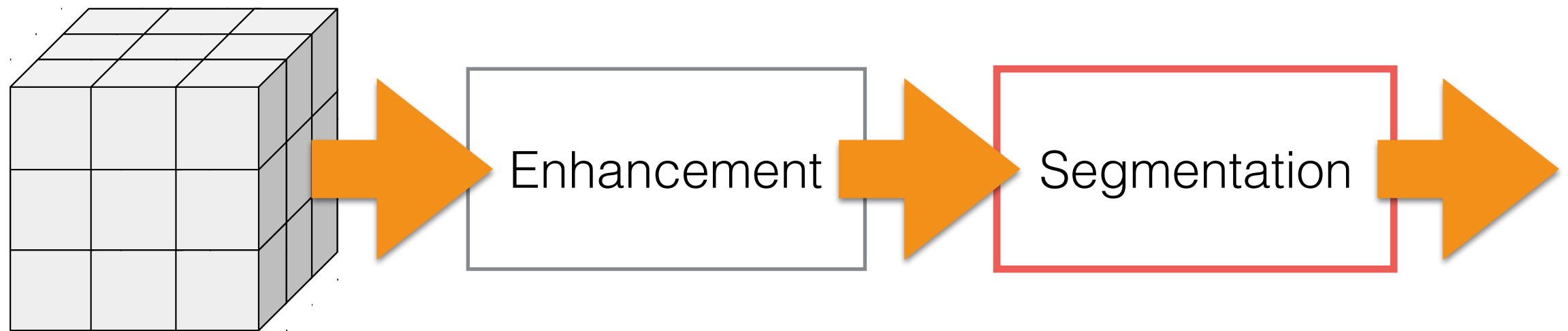


RAW Volume

The Processing Pipeline



The Processing Pipeline



RAW Volume

2D/3D Segmentation

Segmentation

- Segmentation is a process after which we obtain a mask of a structure in an/a image/voxel.
- 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!

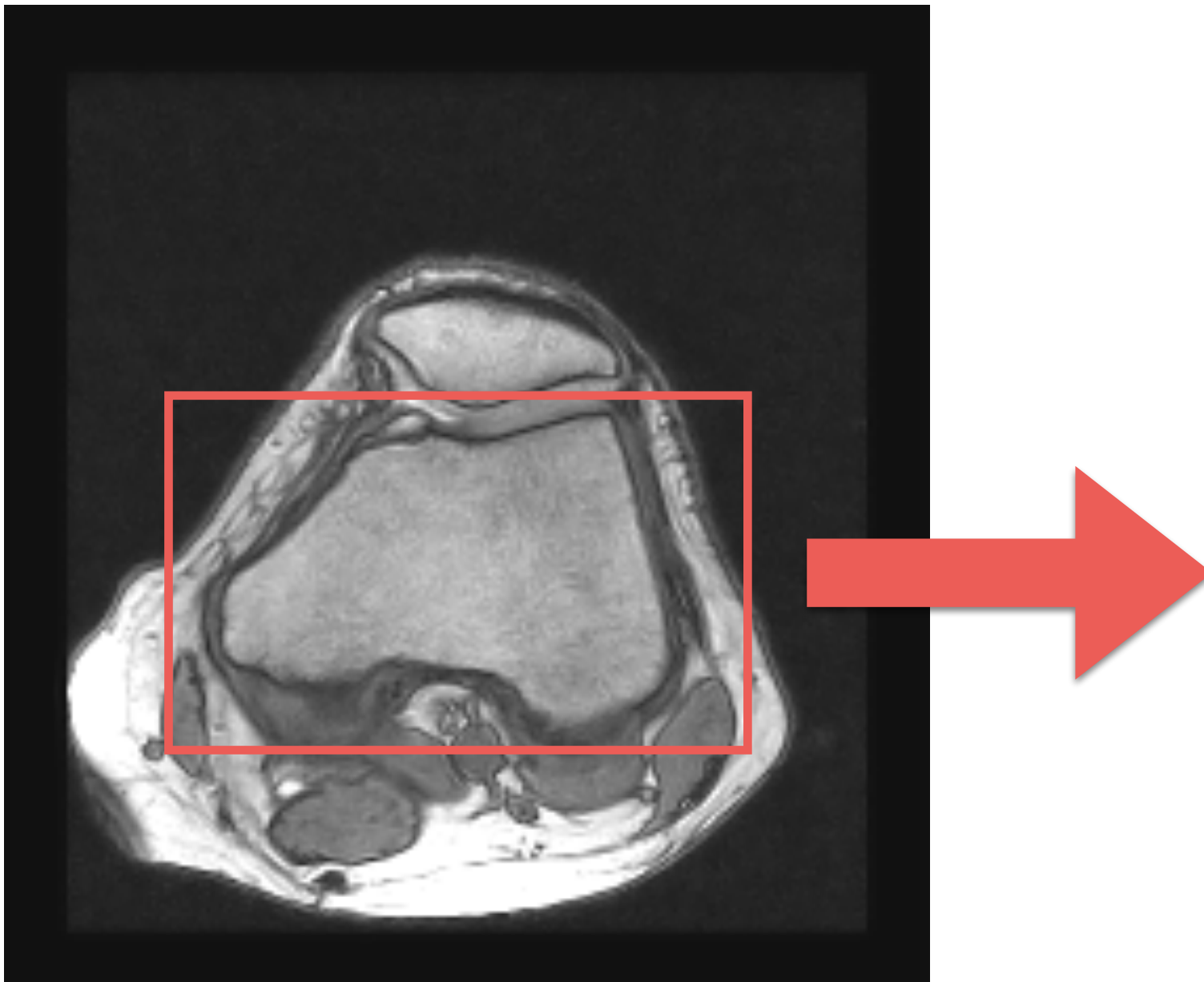
Segmentation Example



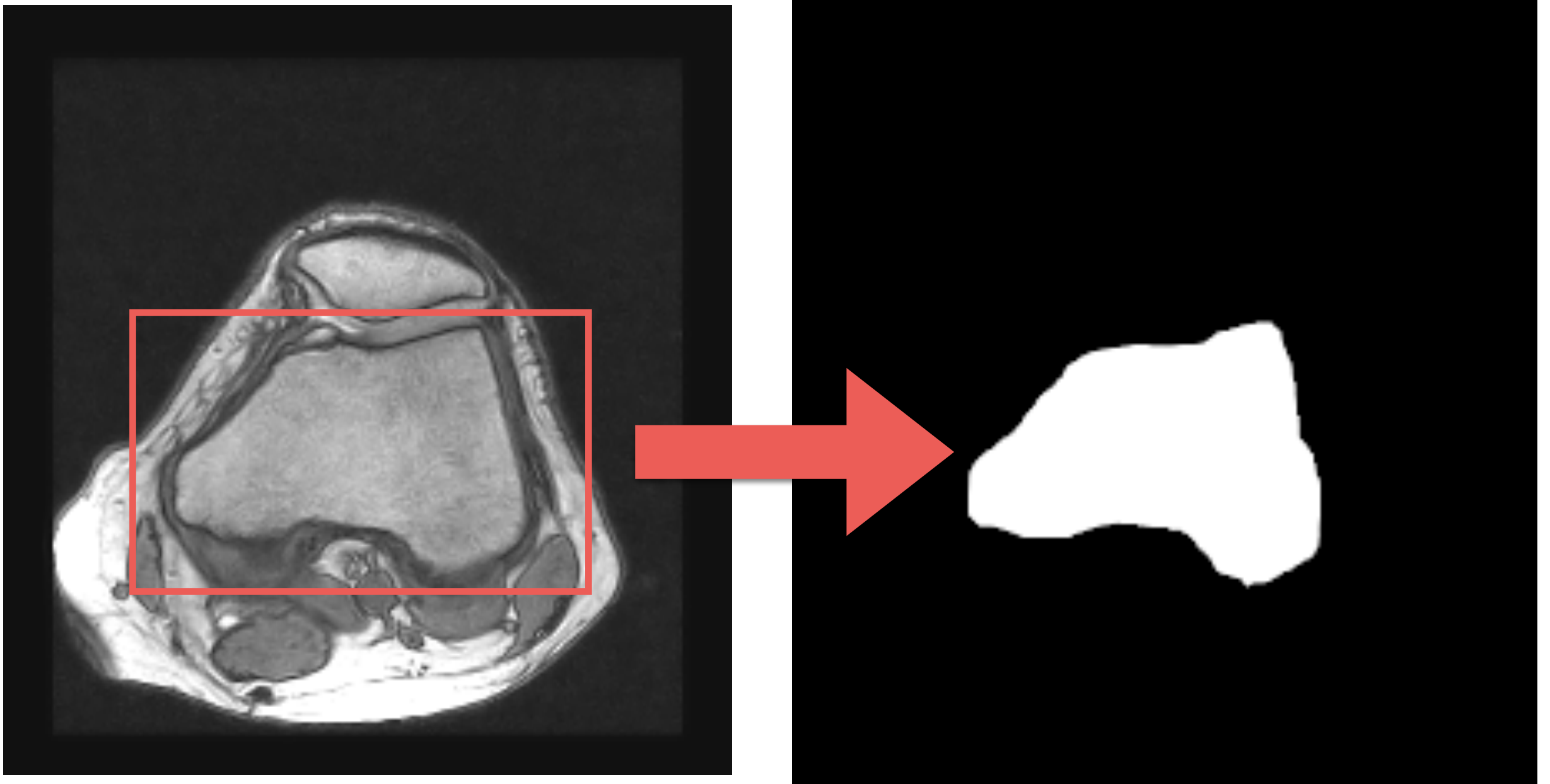
Segmentation Example



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 is labeled a number in $[1, k]$.
Background is always 0!

3D Segmentation

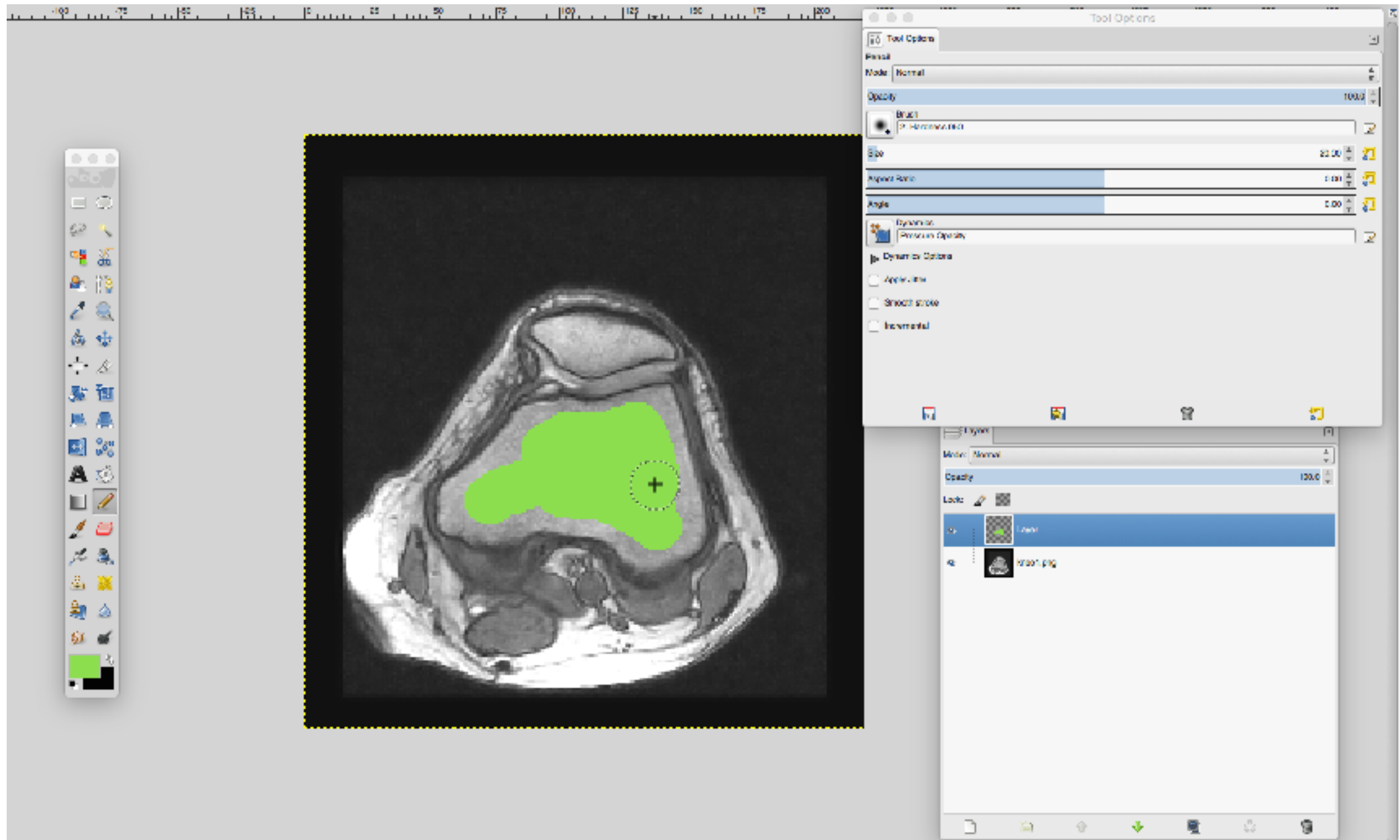
- There are typically two approaches:
 - 2D segmentation for each slice
 - 2D segmentation of a slice and propagation of the segmentation

Manual Segmentation

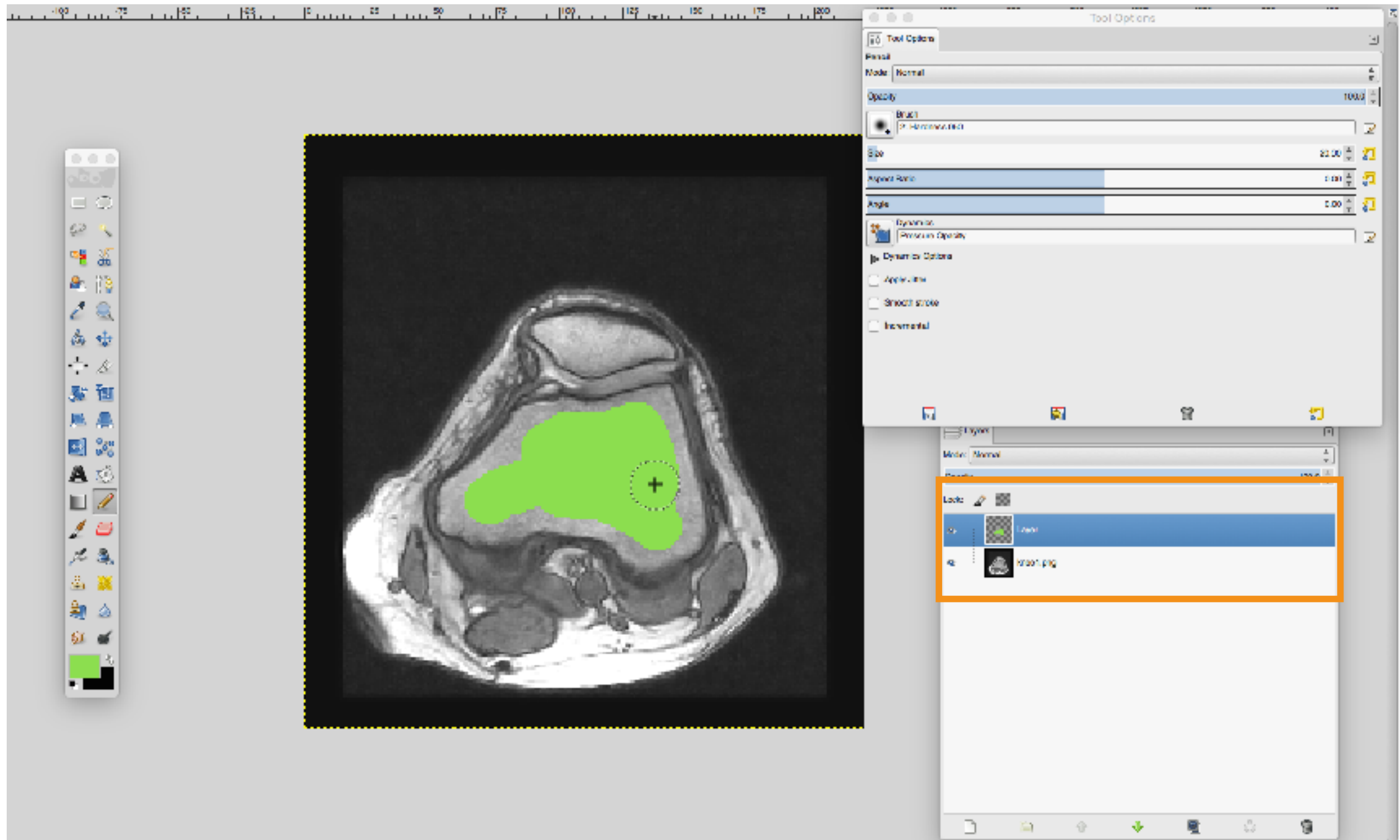
Manual Segmentation: Painting Approach

- 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



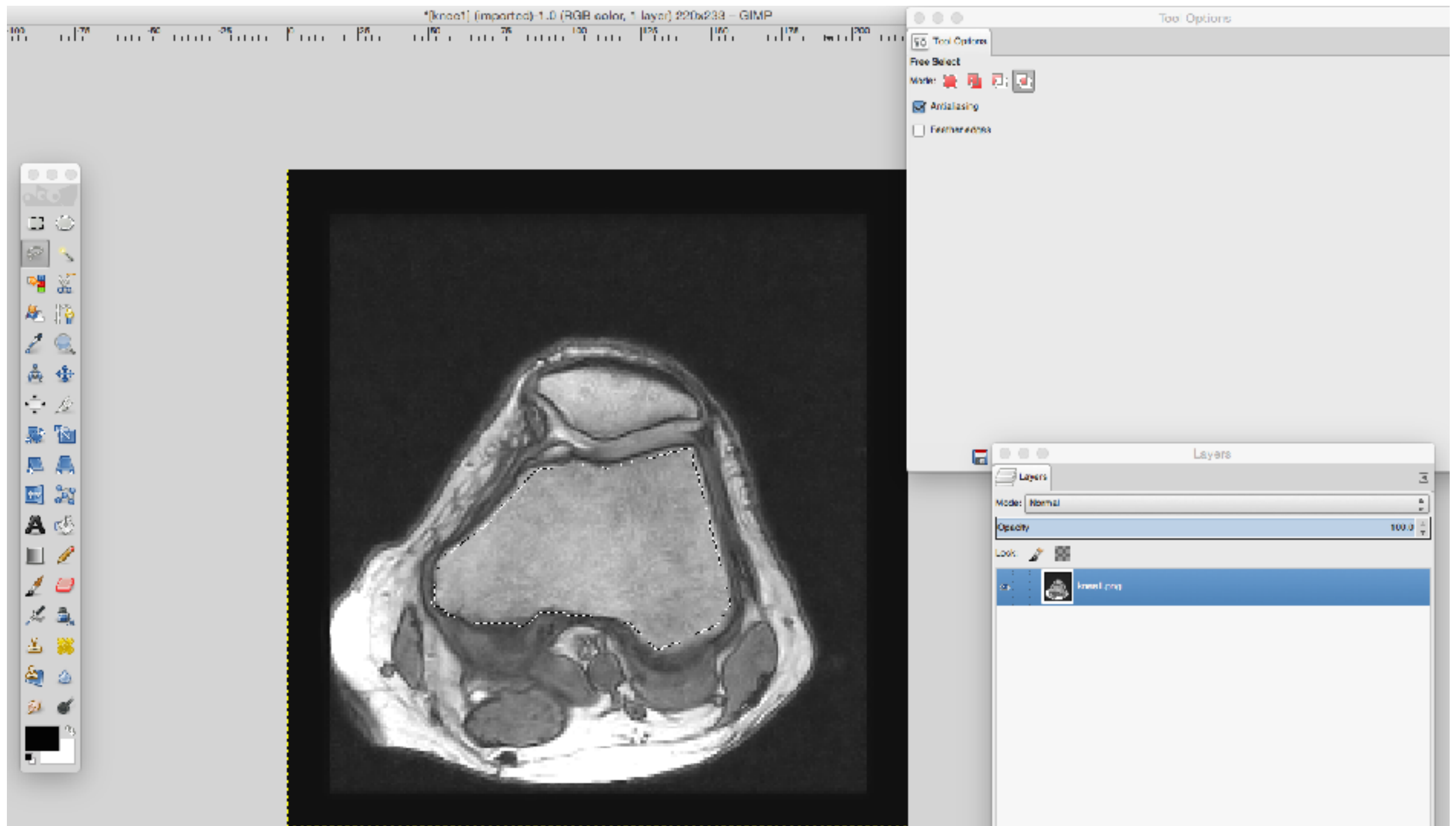
Manual Segmentation: Painting Approach



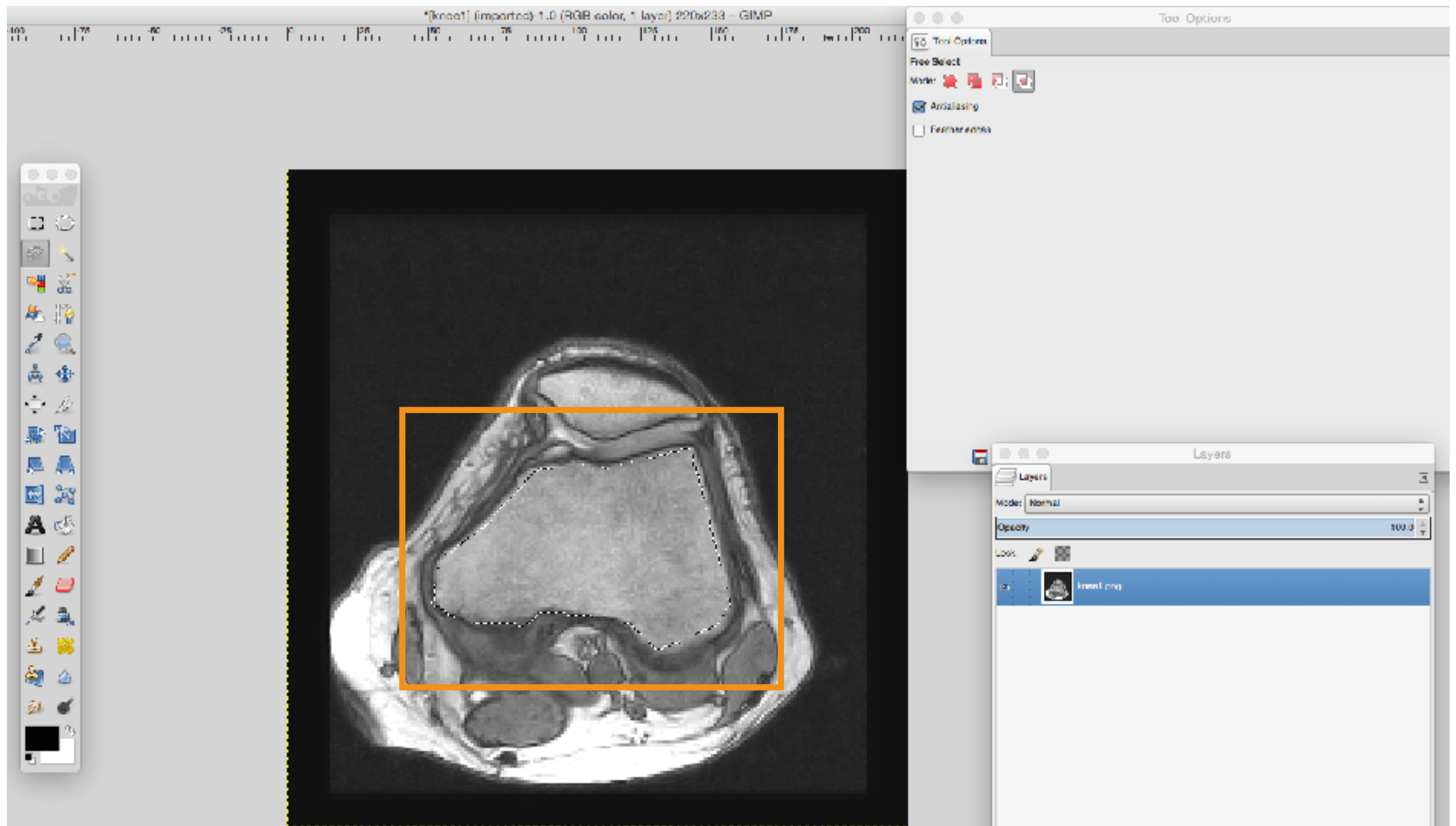
Manual Segmentation: Boundary Definition

- We manually define the boundary of the mask using a GUI.
- 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



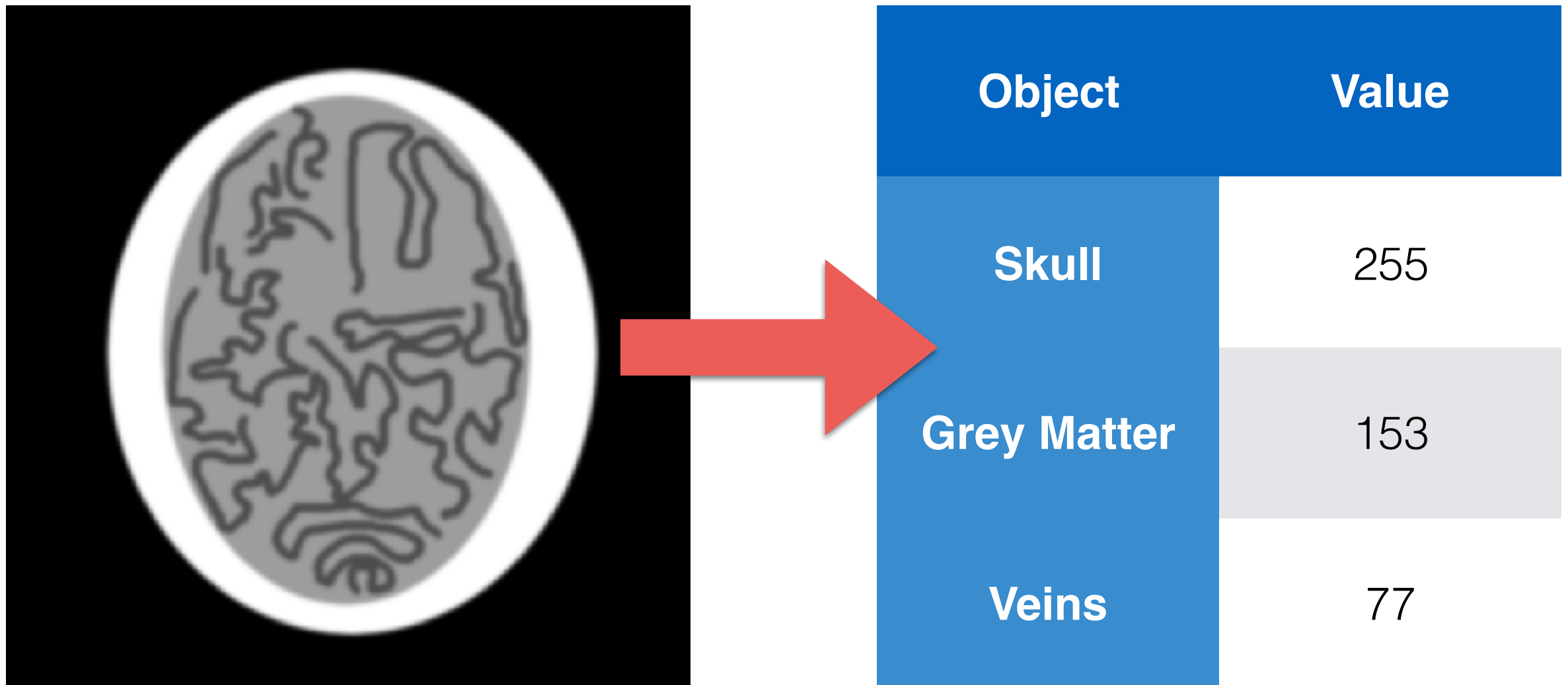
Manual Segmentation: Boundary Definition



Thresholding

Thresholding Example

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



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:

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

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

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

Thresholding

- This means:

Reference Value

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

- We can have different distance functions:

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

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

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

Thresholding

- This means:

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

Reference Value Threshold

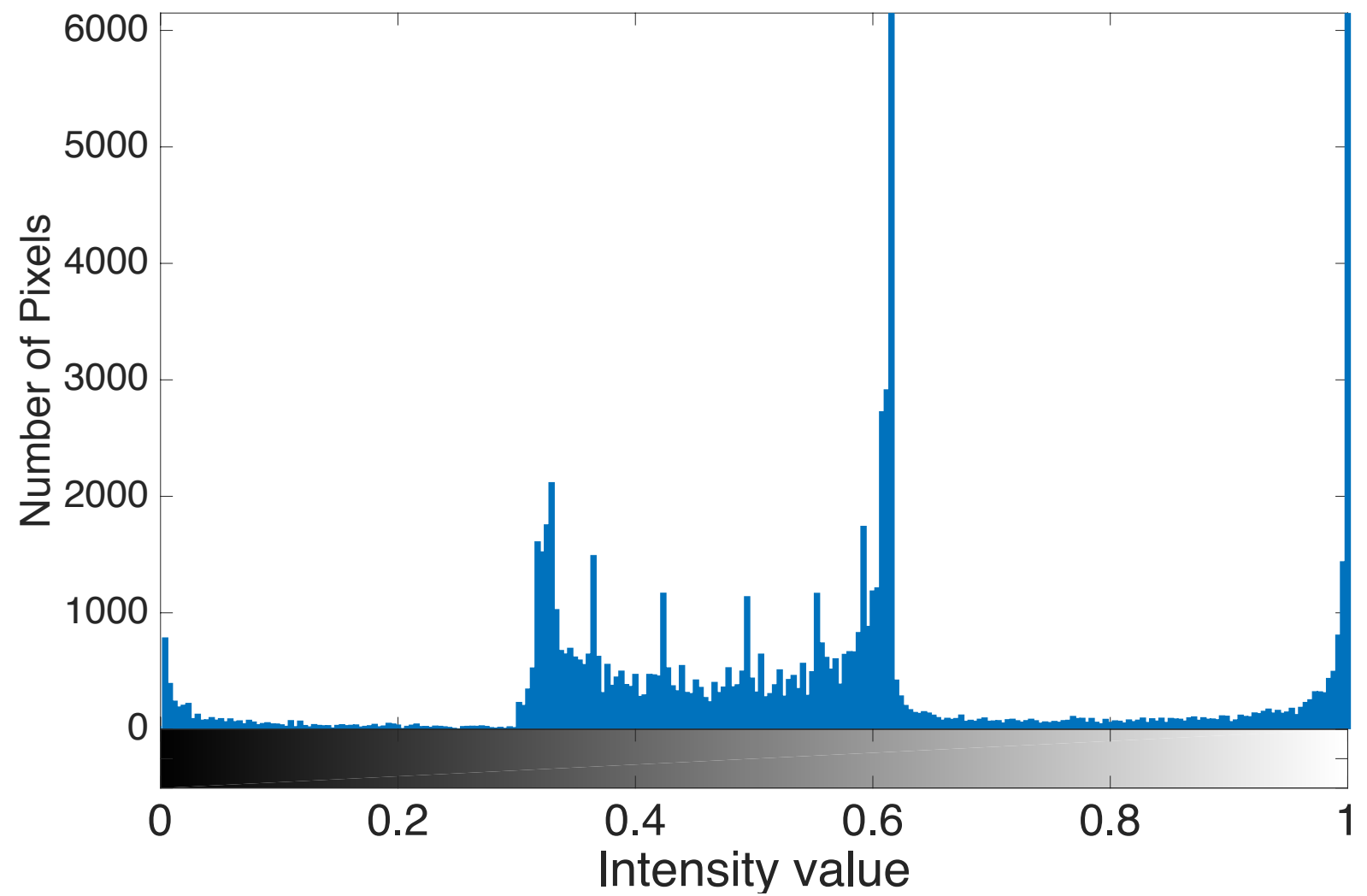
- We can have different distance functions:

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

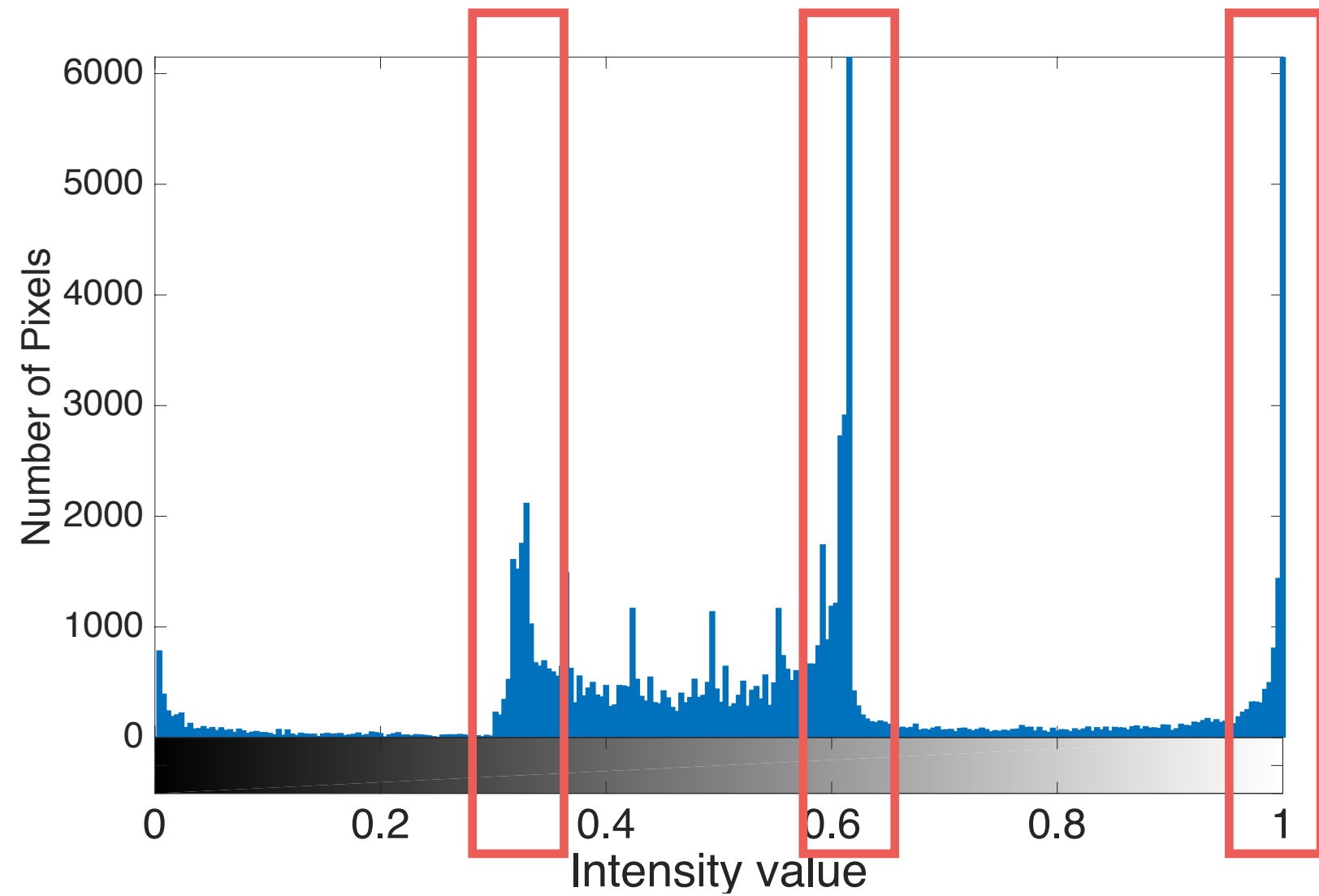
$$d(x; y) = (x - y)^2$$

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

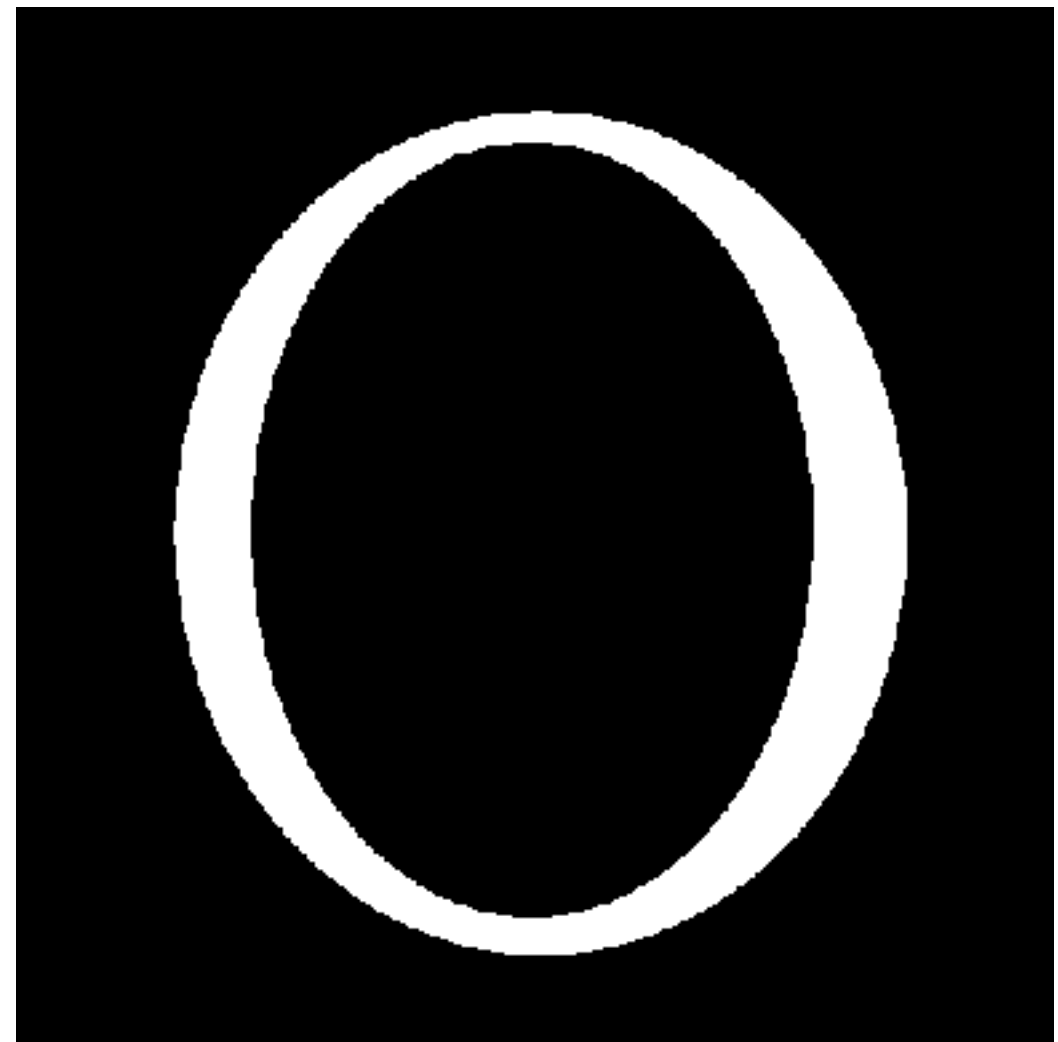
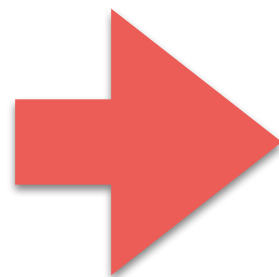
Thresholding



Thresholding

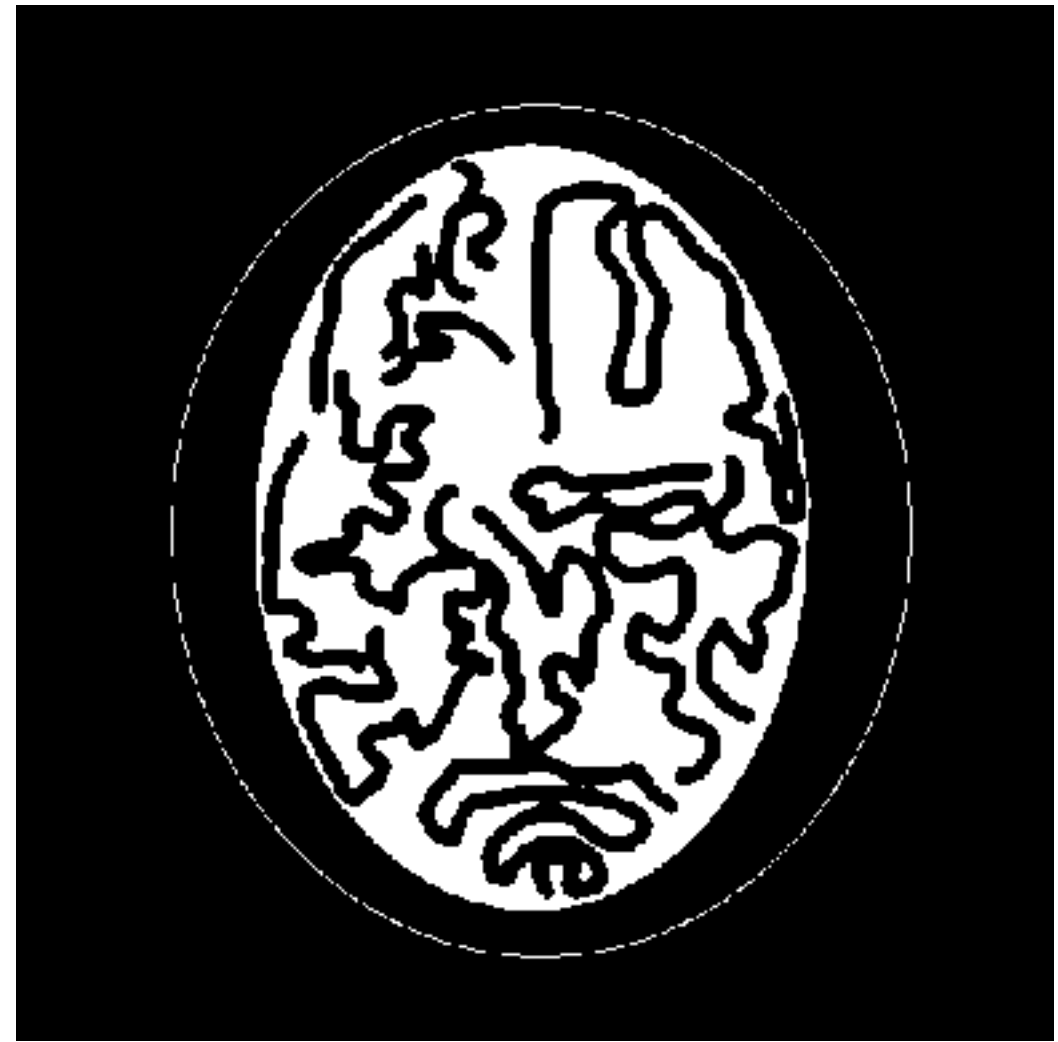
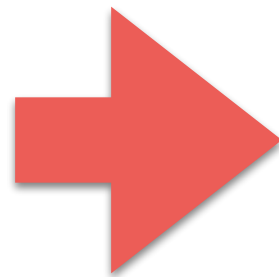


Thresholding Example



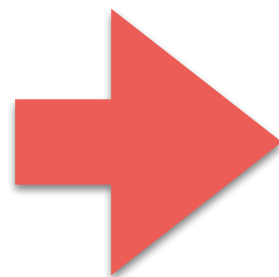
$$I_t = 1 \quad t = 0.1$$

Thresholding Example



$$I_t = 0.6 \quad t = 0.1$$

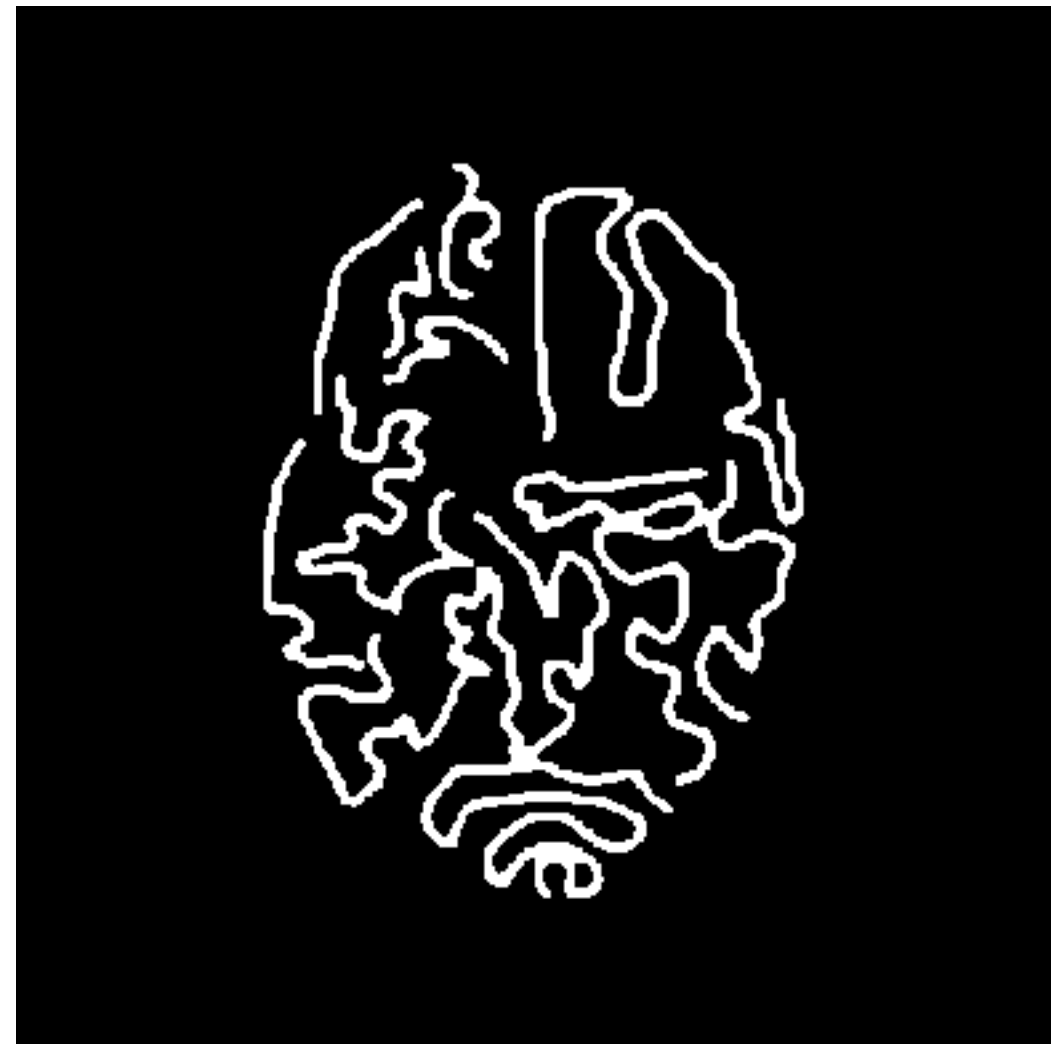
Thresholding Example



$$I_t = 0.3 \quad t = 0.1$$

Thresholding: Connected Components

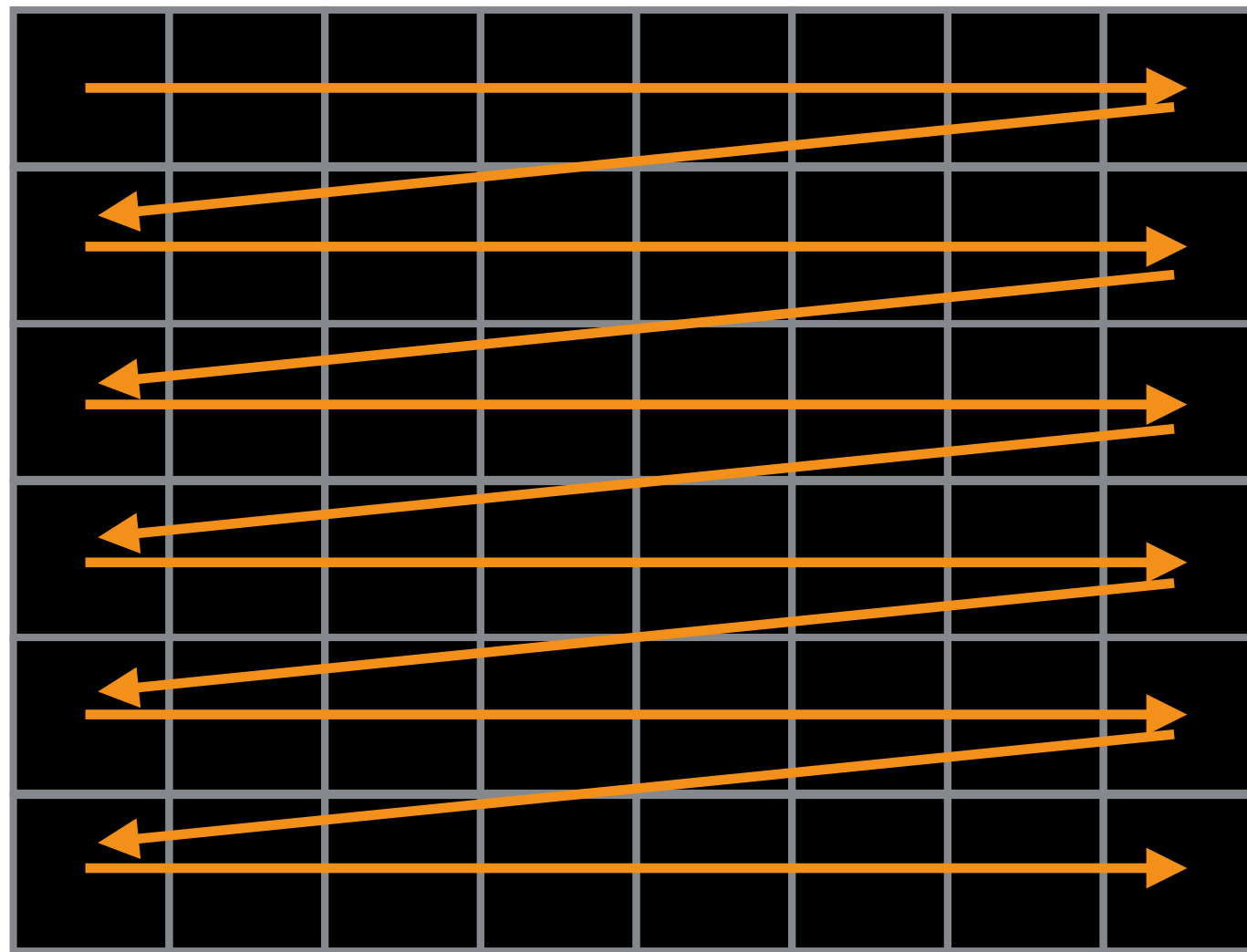
- After segmentation we may end up with different pieces that are not connected.



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.

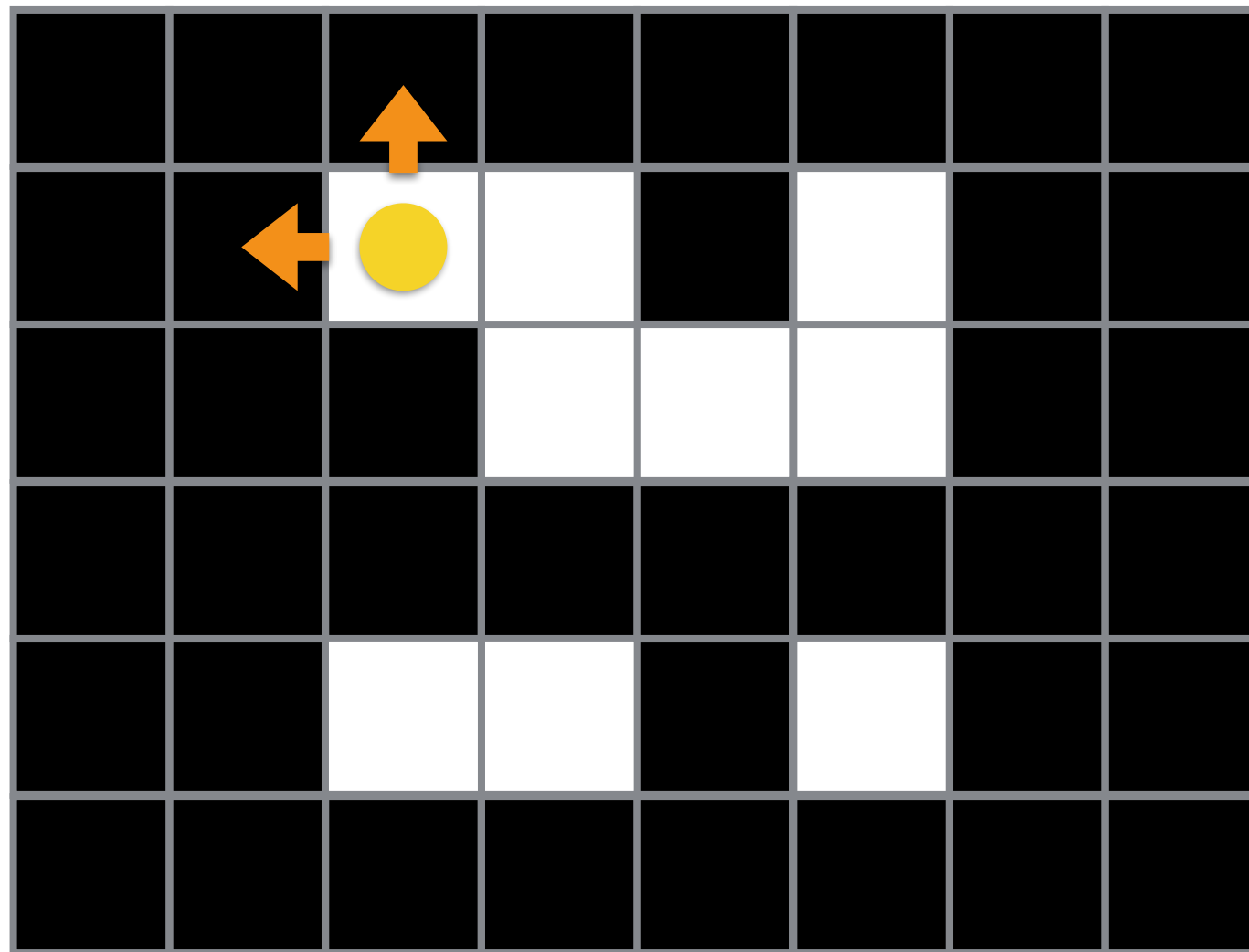
Thresholding: Connected Components



Scan order

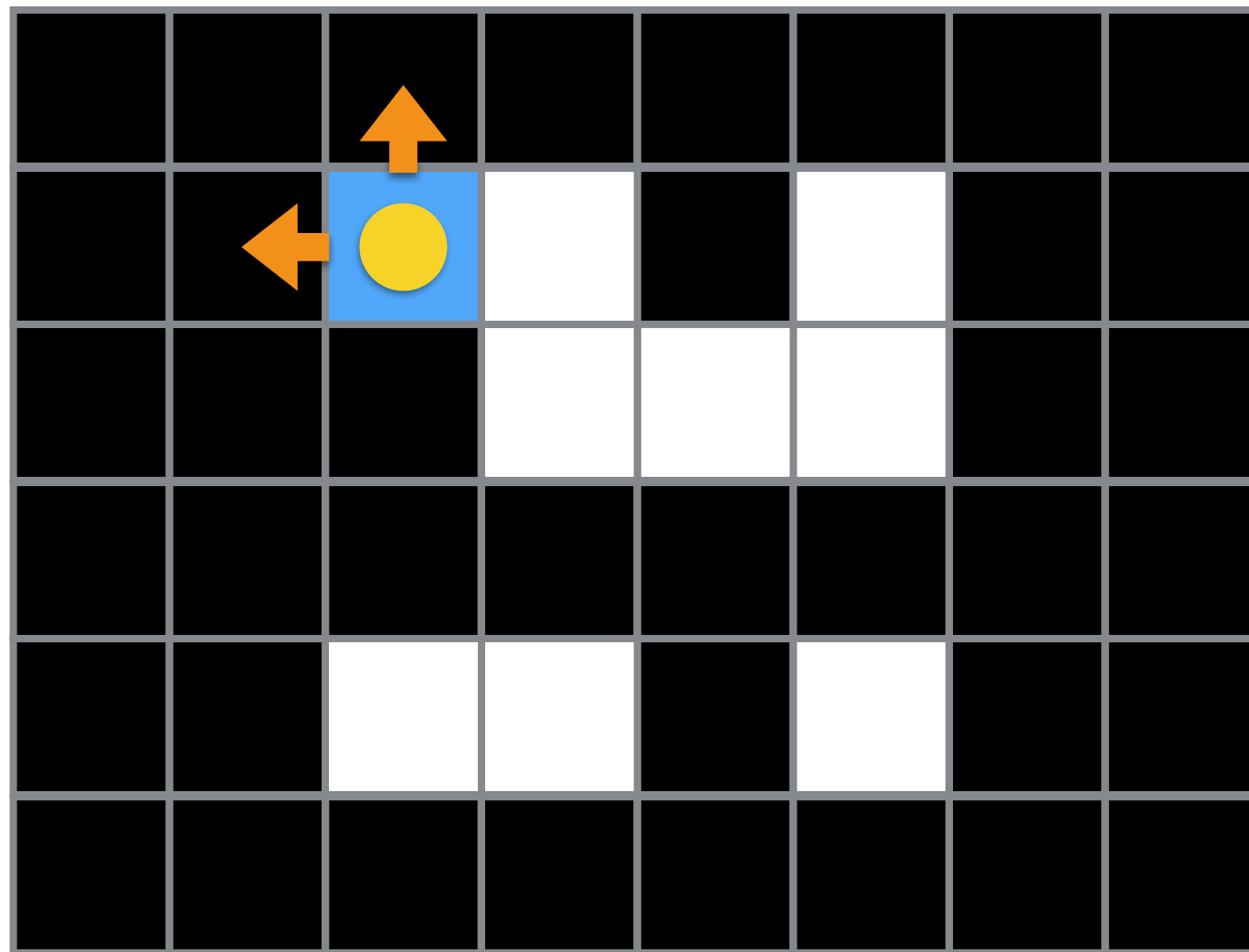
First Pass

Thresholding: Connected Components



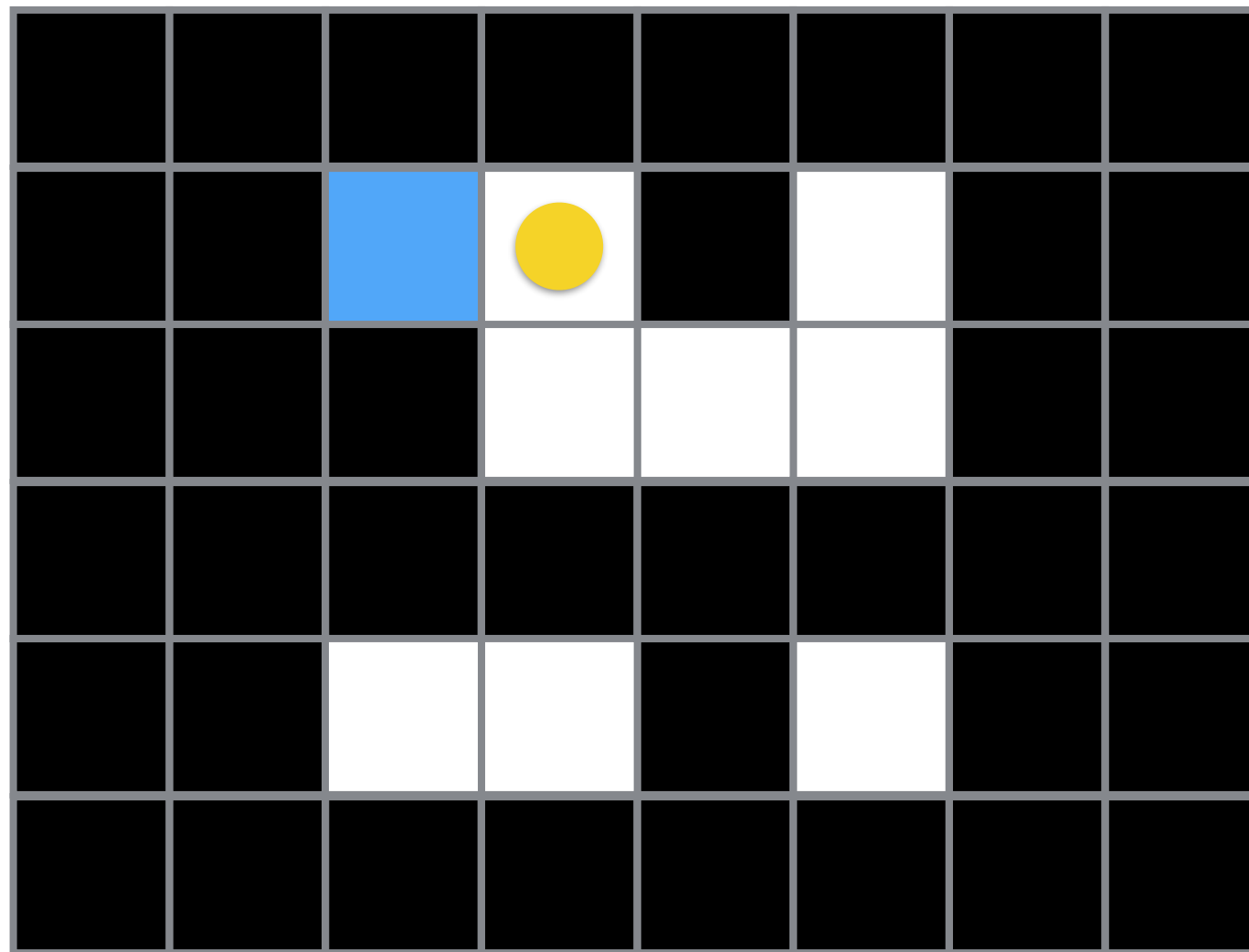
We check up and left neighbors to see if they have a label.

Thresholding: Connected Components



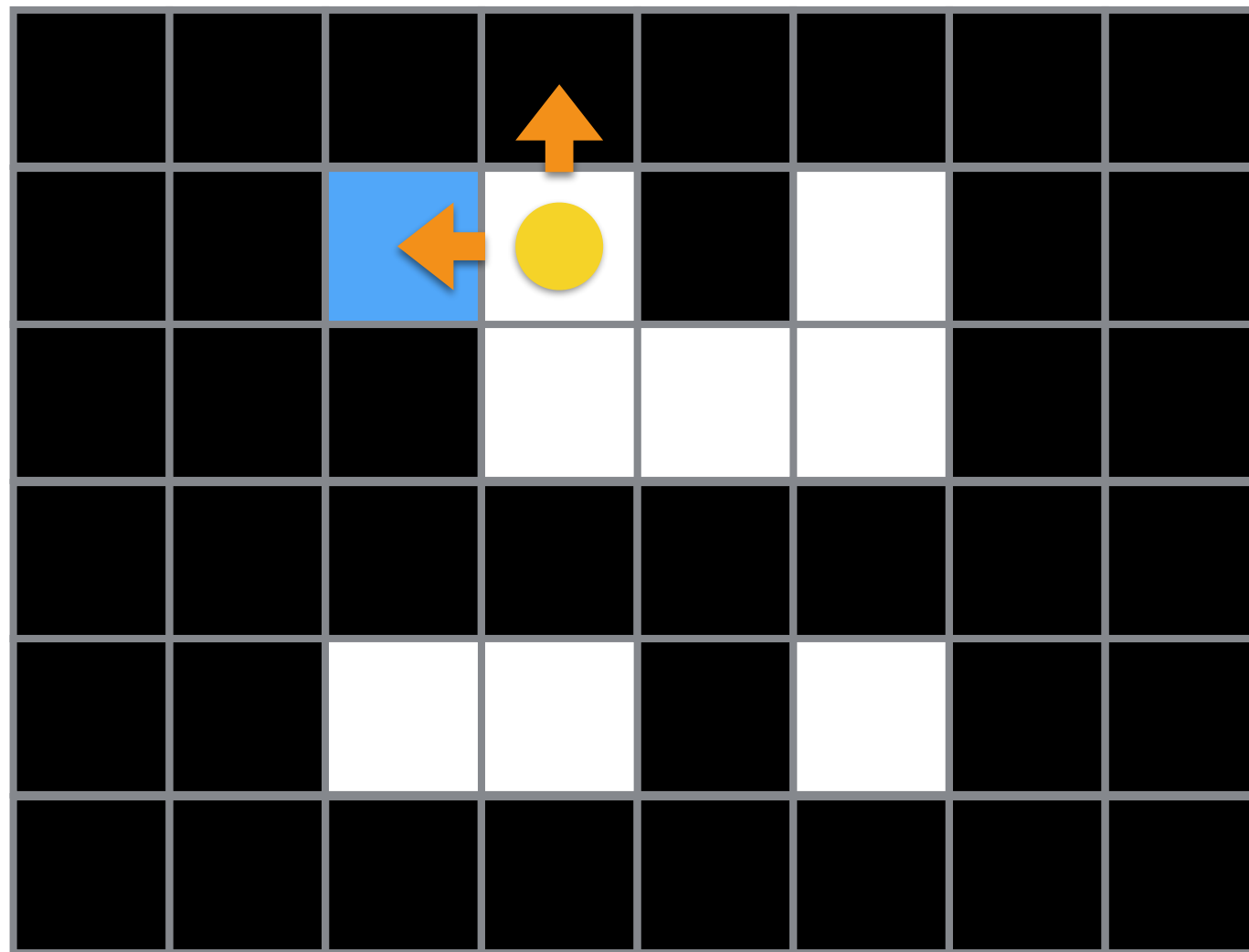
If not we create a
new one.

Thresholding: Connected Components



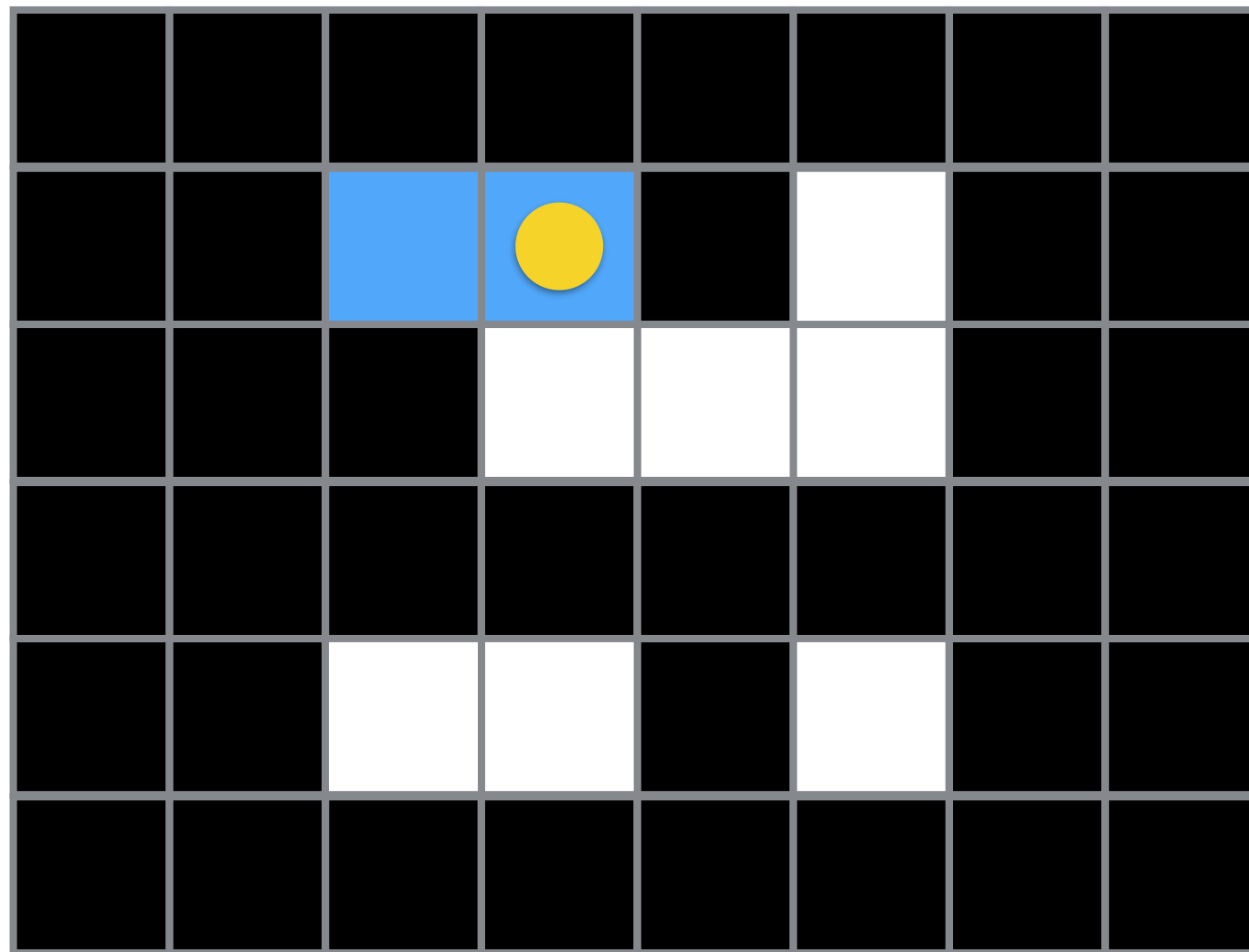
Then, we move
right, and we
repeat the
process.

Thresholding: Connected Components



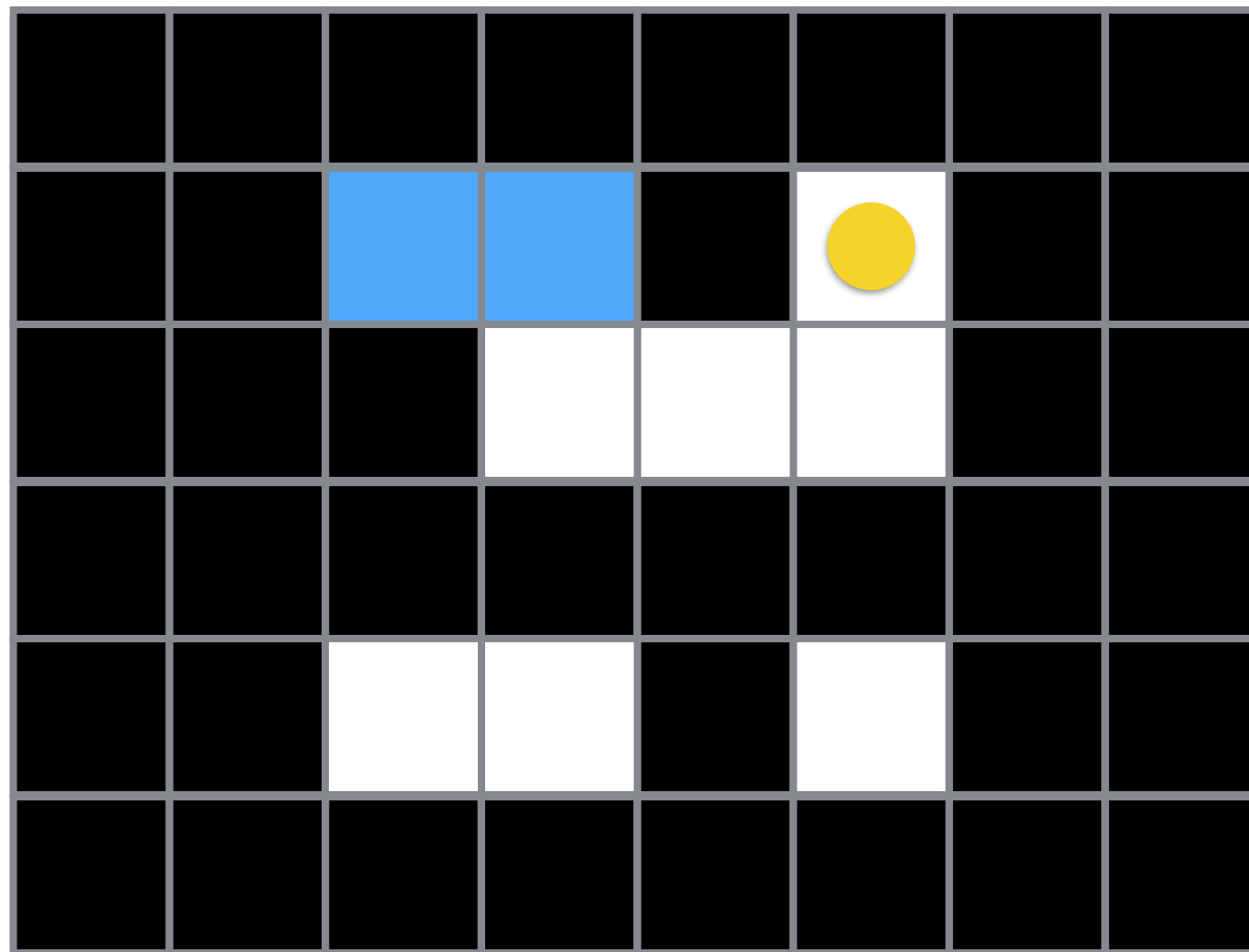
In this case, the left neighbor has a label, so we reuse it.

Thresholding: Connected Components

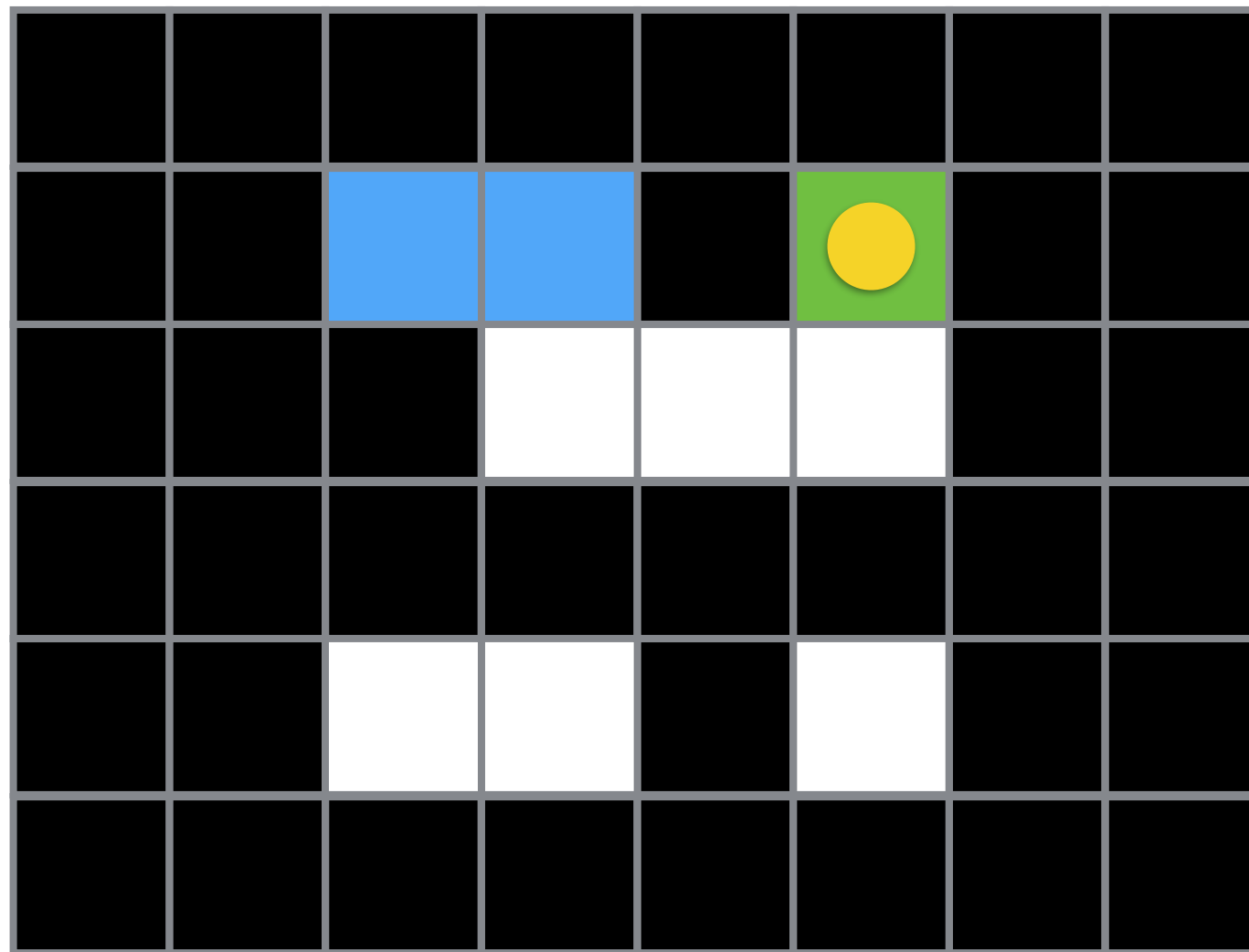


In this case, the left neighbor has a label, so we reuse it.

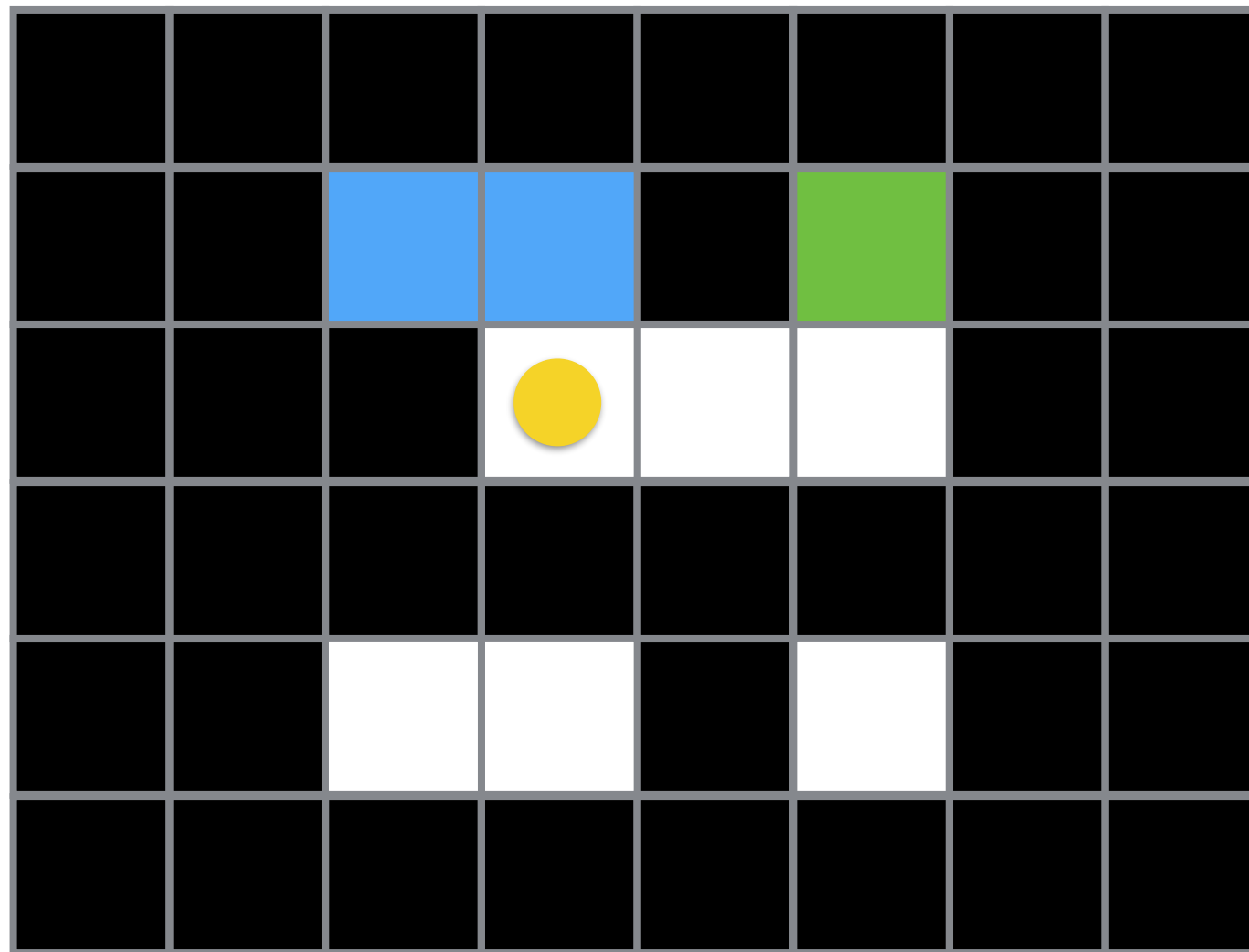
Thresholding: Connected Components



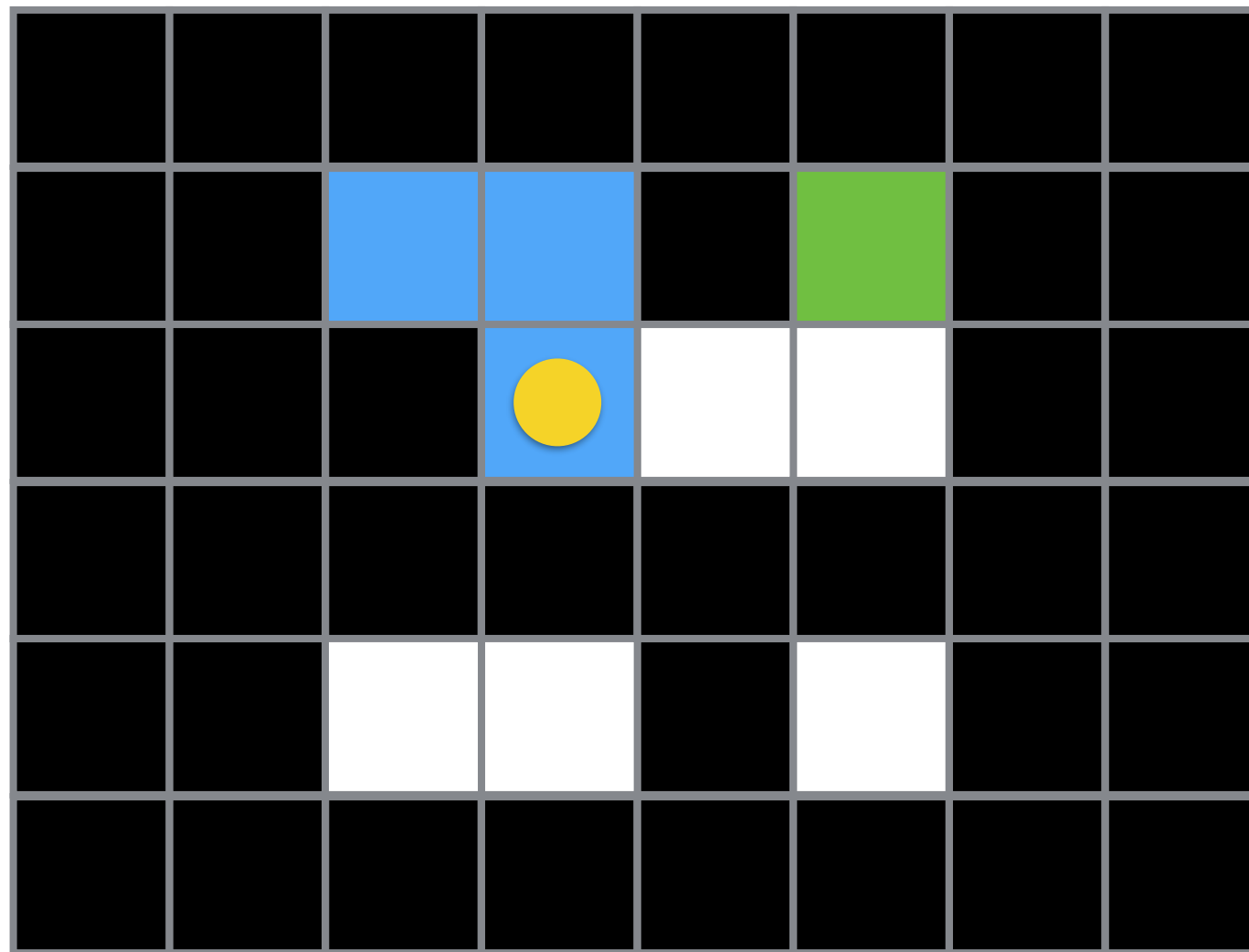
Thresholding: Connected Components



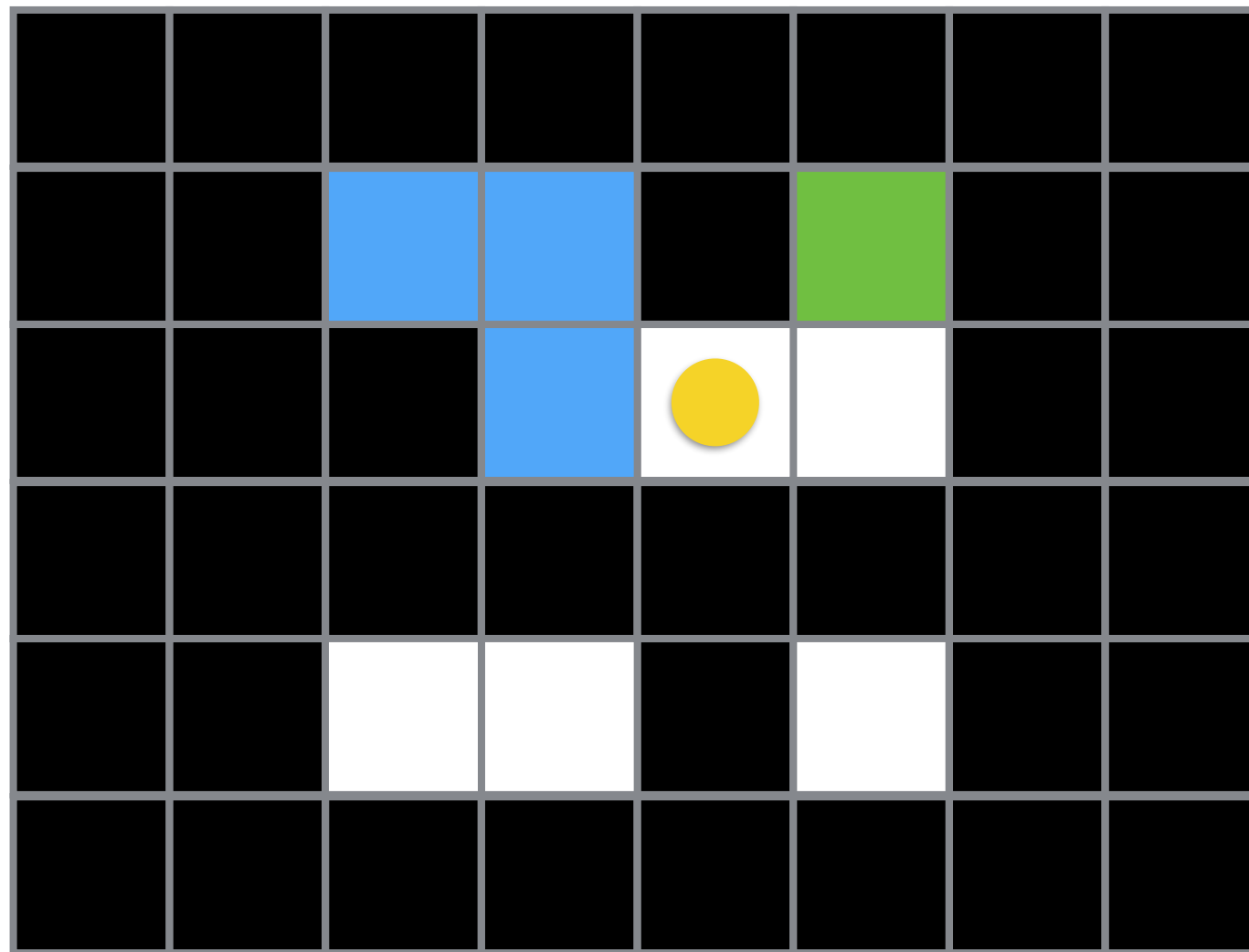
Thresholding: Connected Components



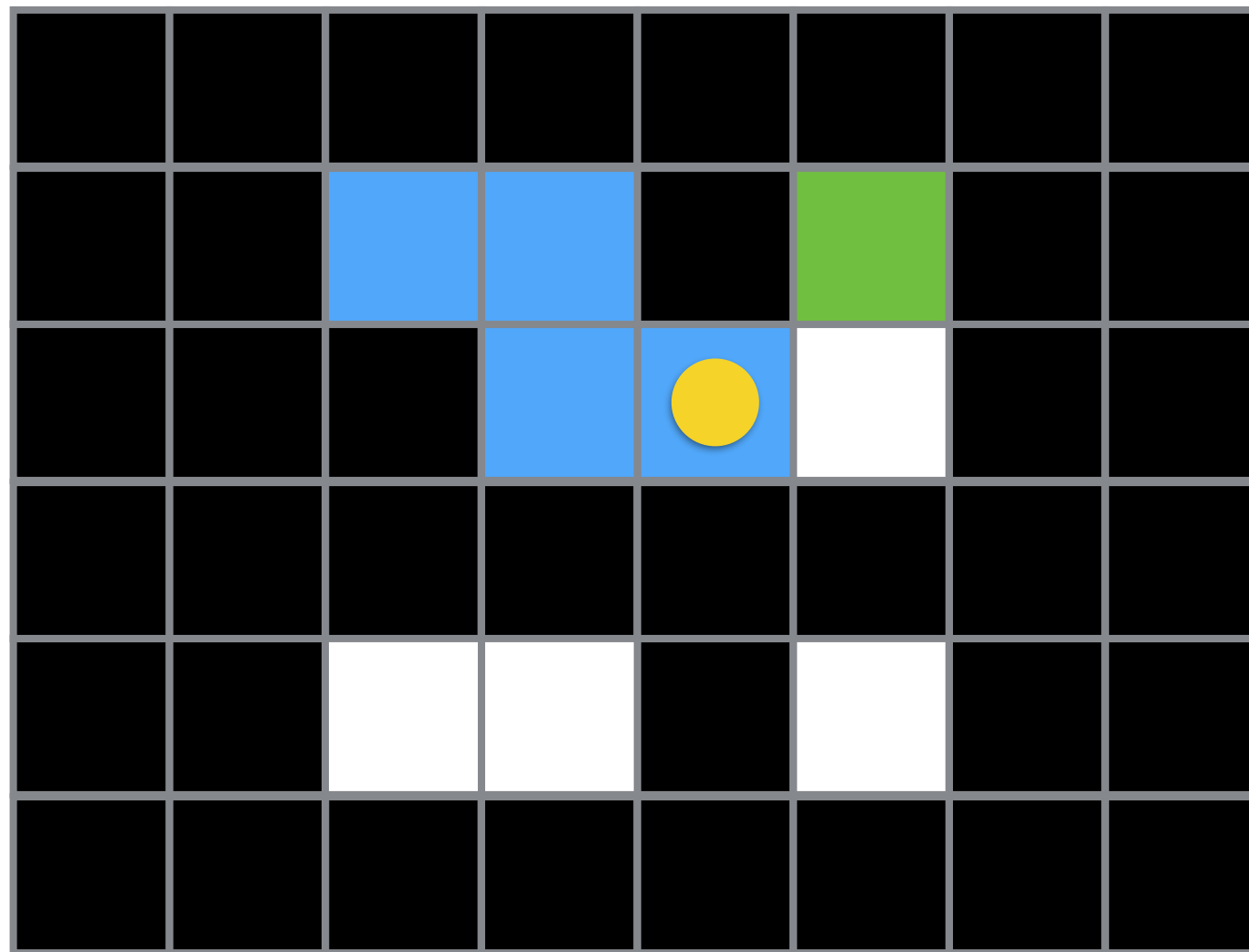
Thresholding: Connected Components



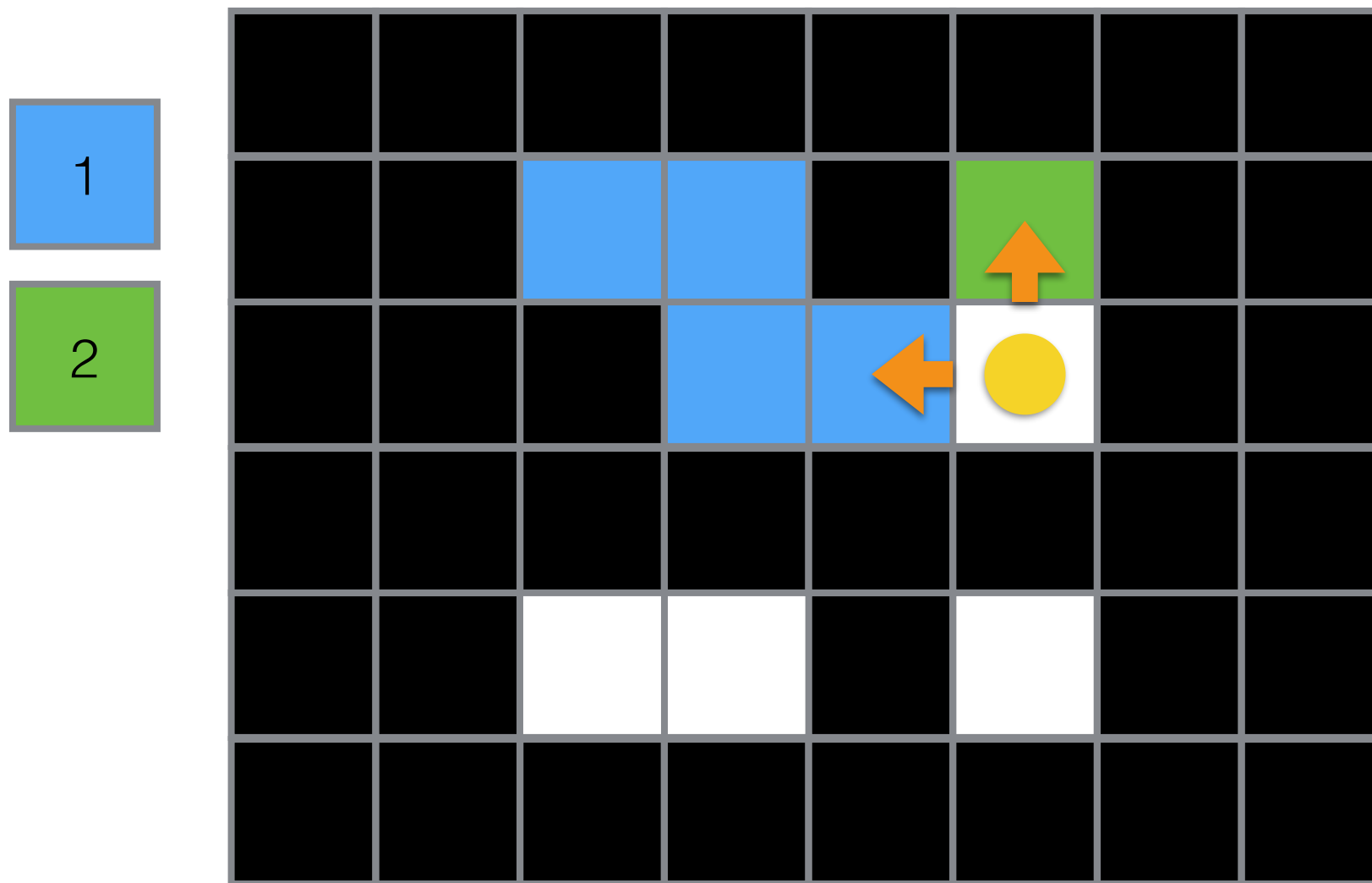
Thresholding: Connected Components



Thresholding: Connected Components

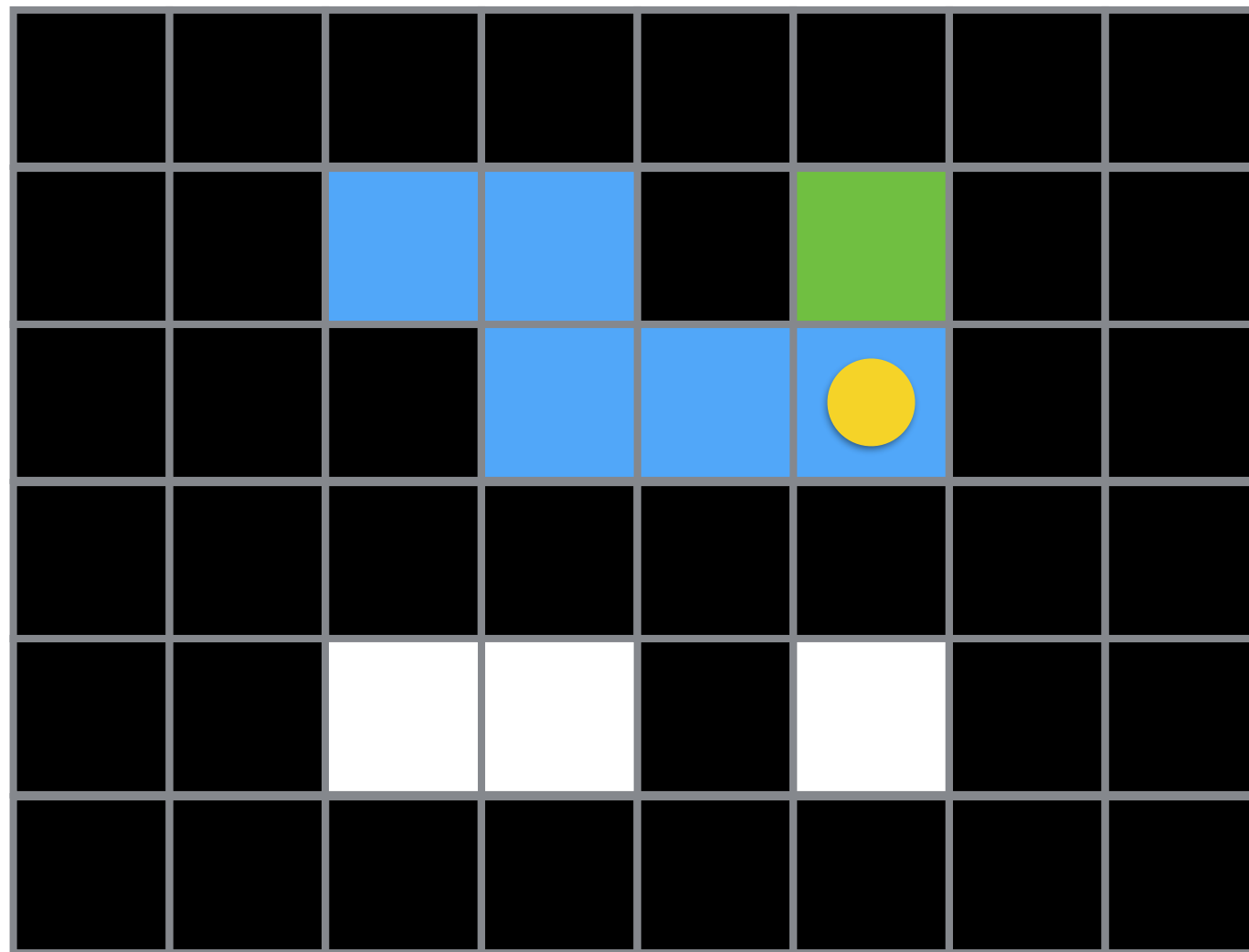


Thresholding: Connected Components

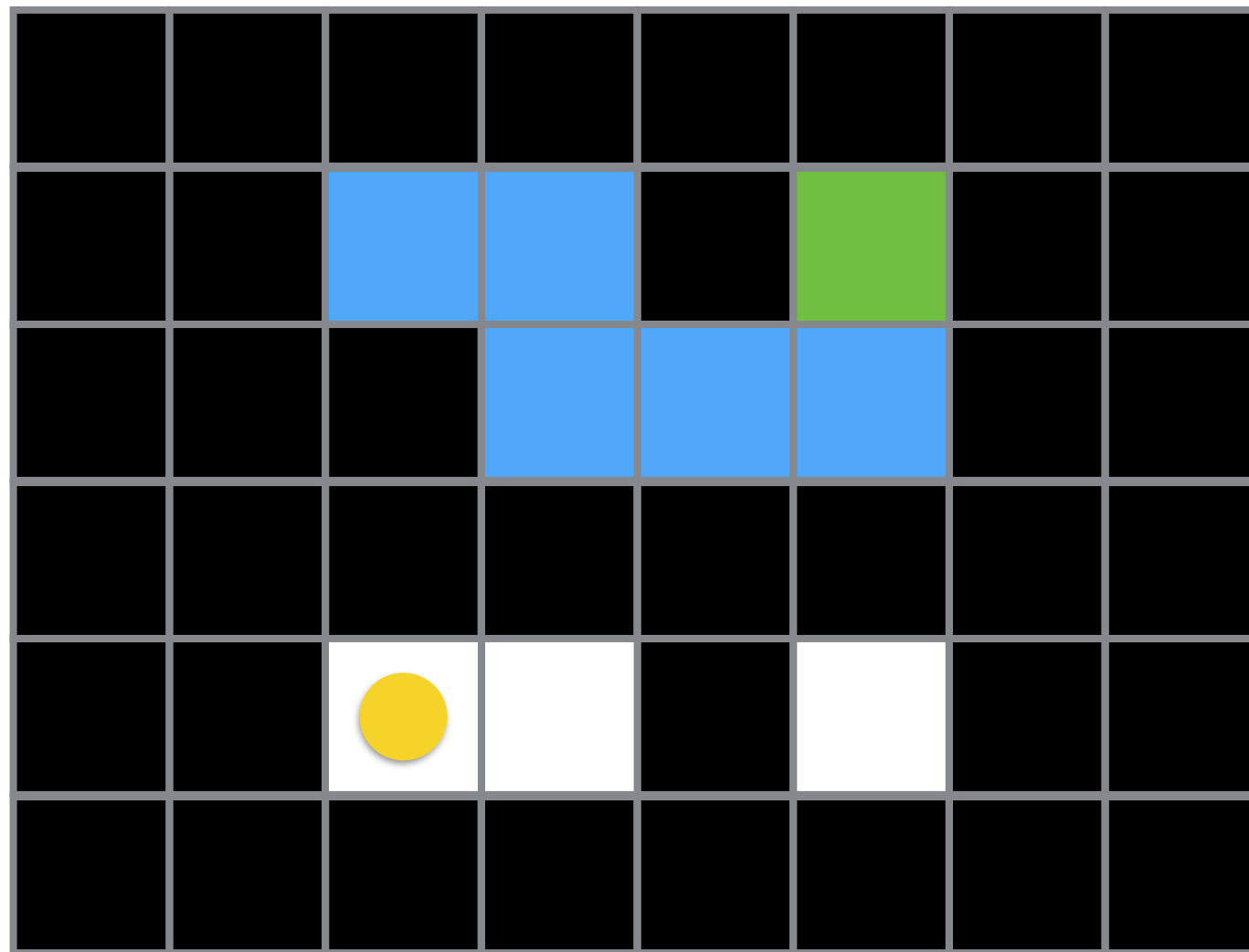


In this case, we choose the lowest label, and we store that 1 is equivalent to 2

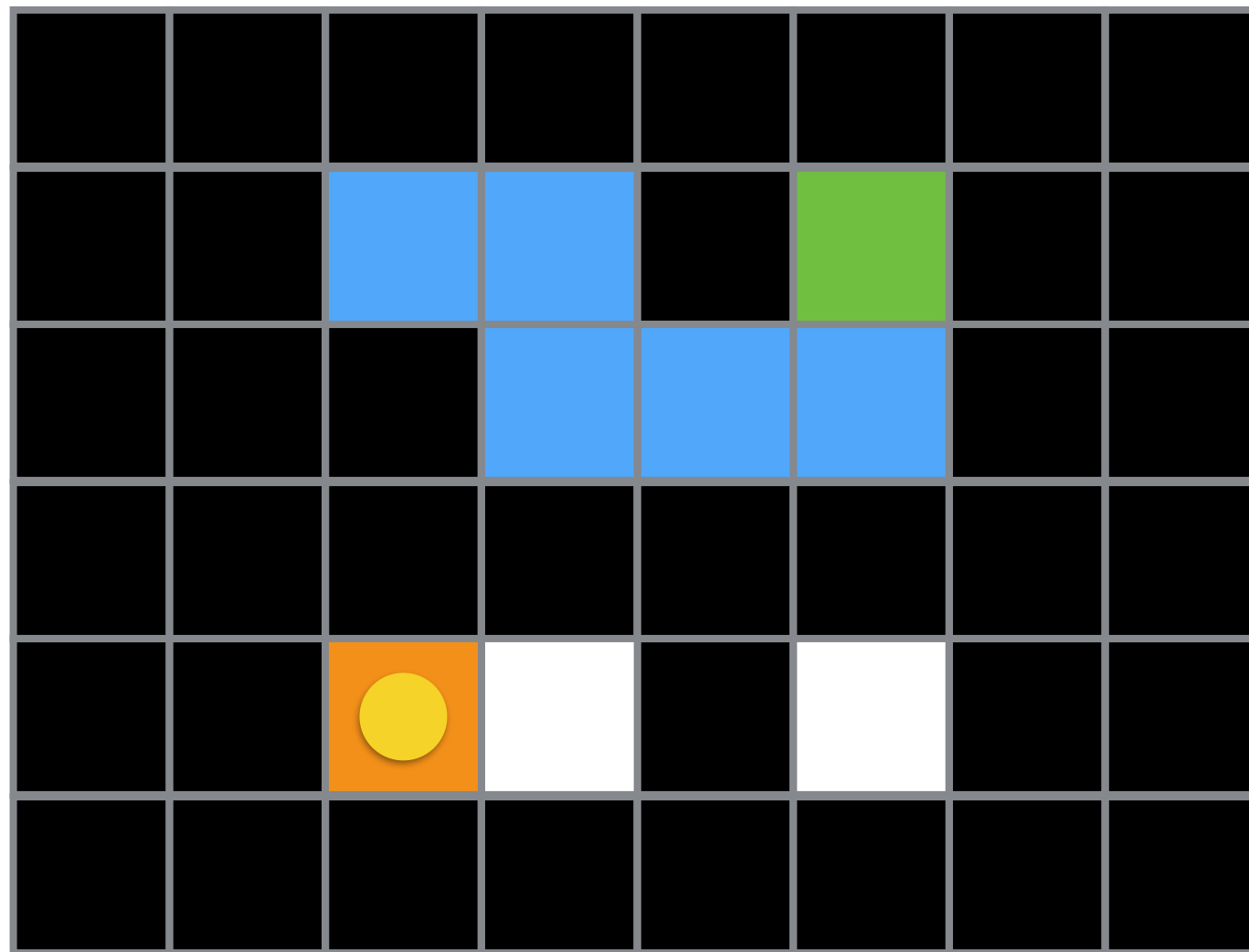
Thresholding: Connected Components



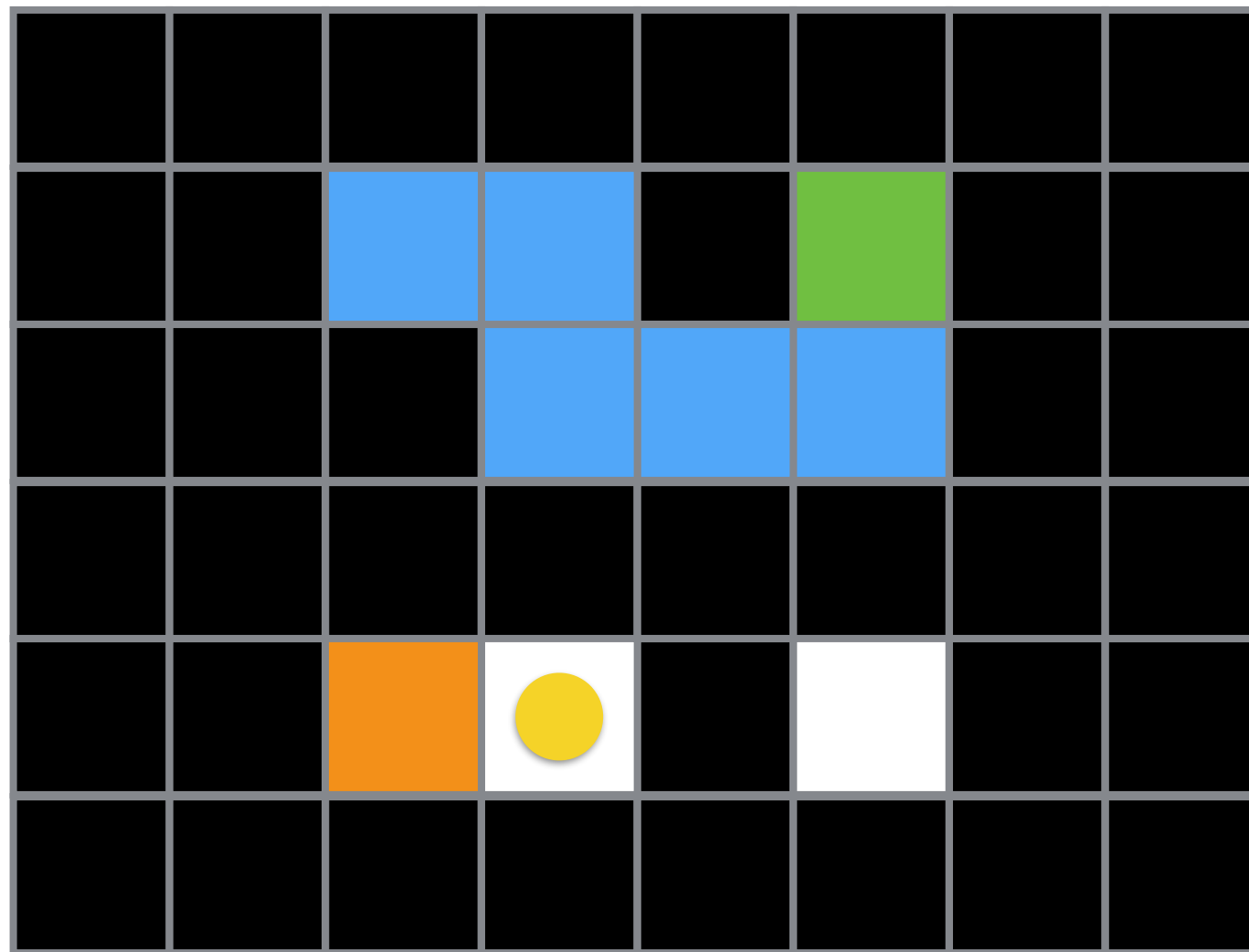
Thresholding: Connected Components



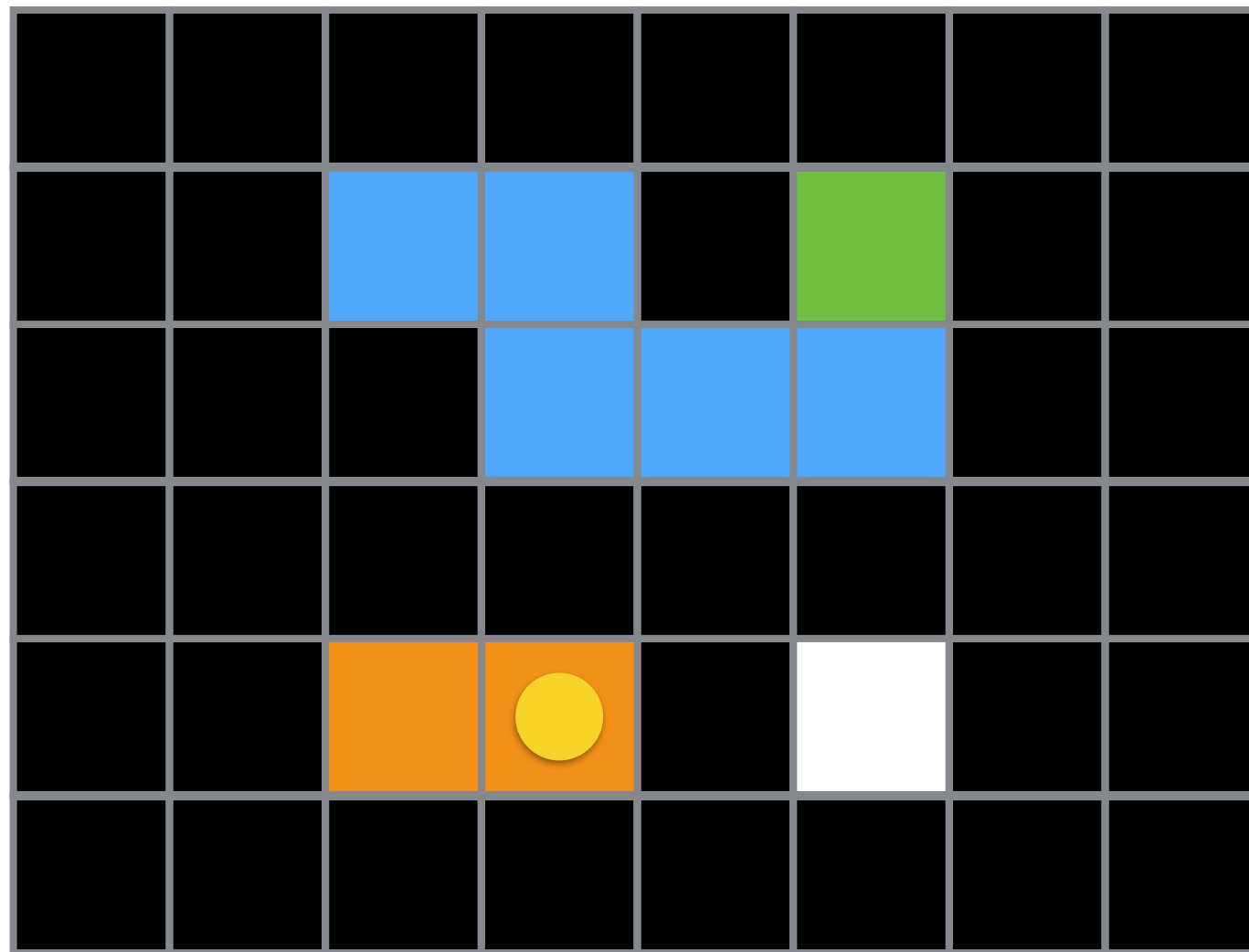
Thresholding: Connected Components



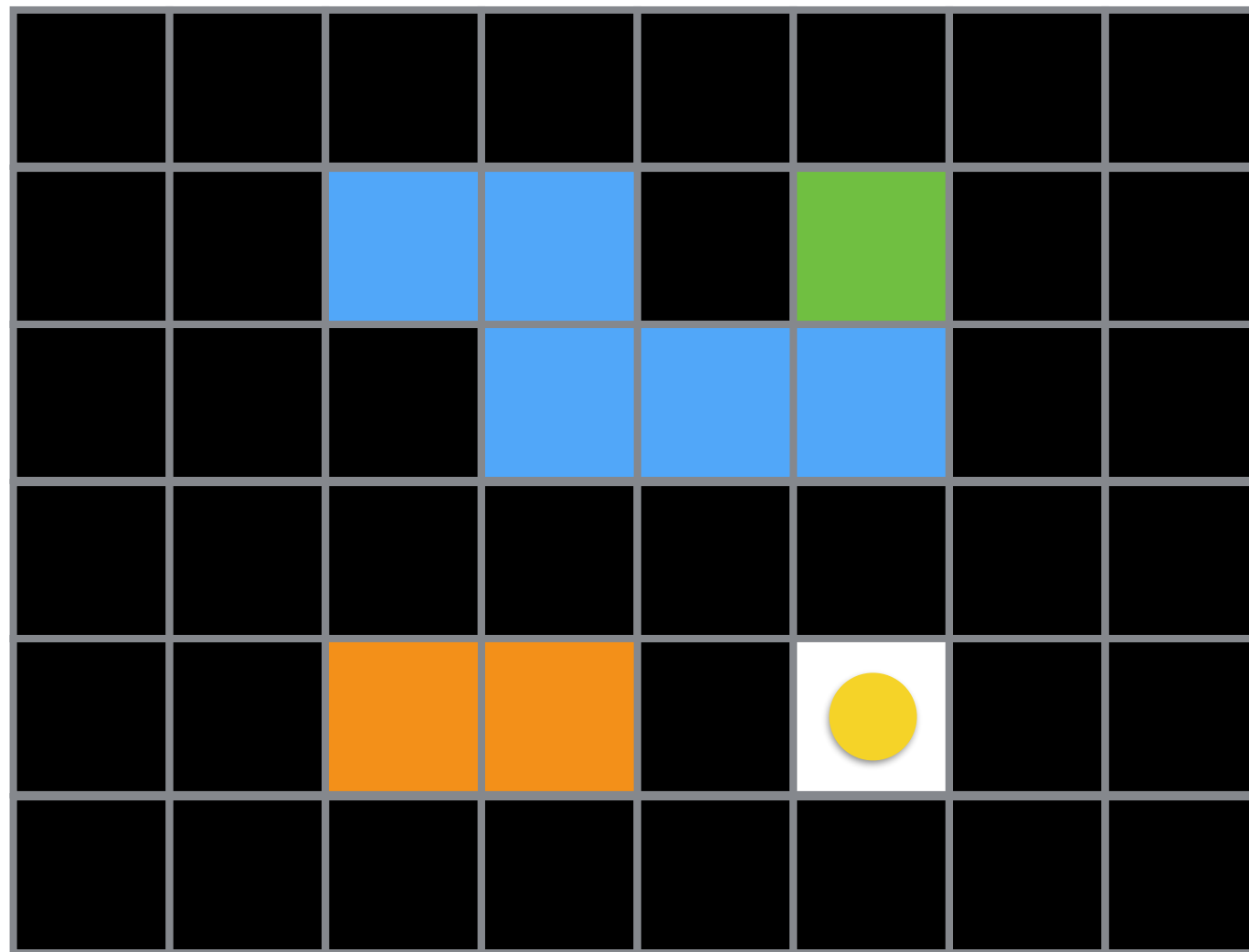
Thresholding: Connected Components



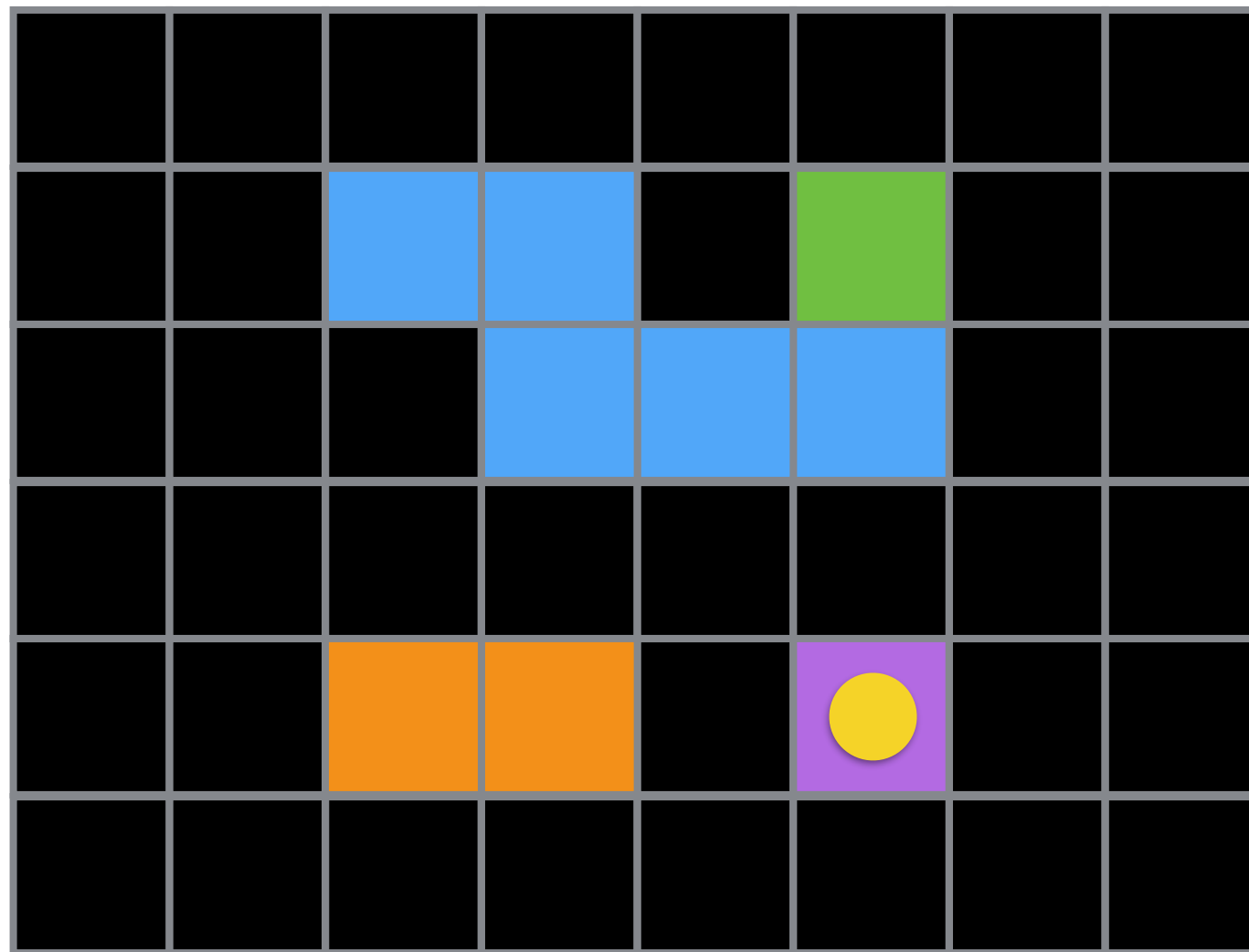
Thresholding: Connected Components



Thresholding: Connected Components

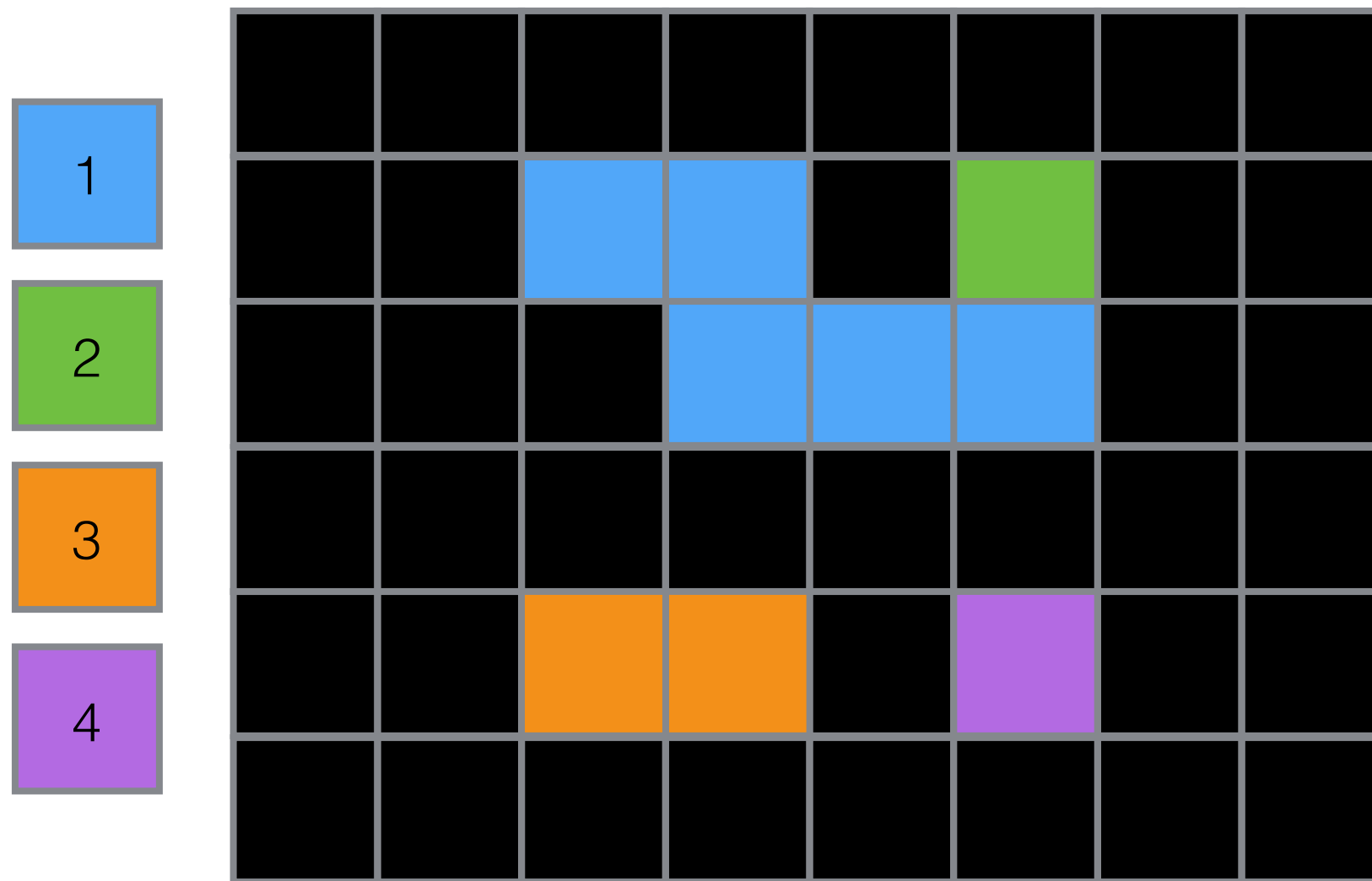


Thresholding: Connected Components



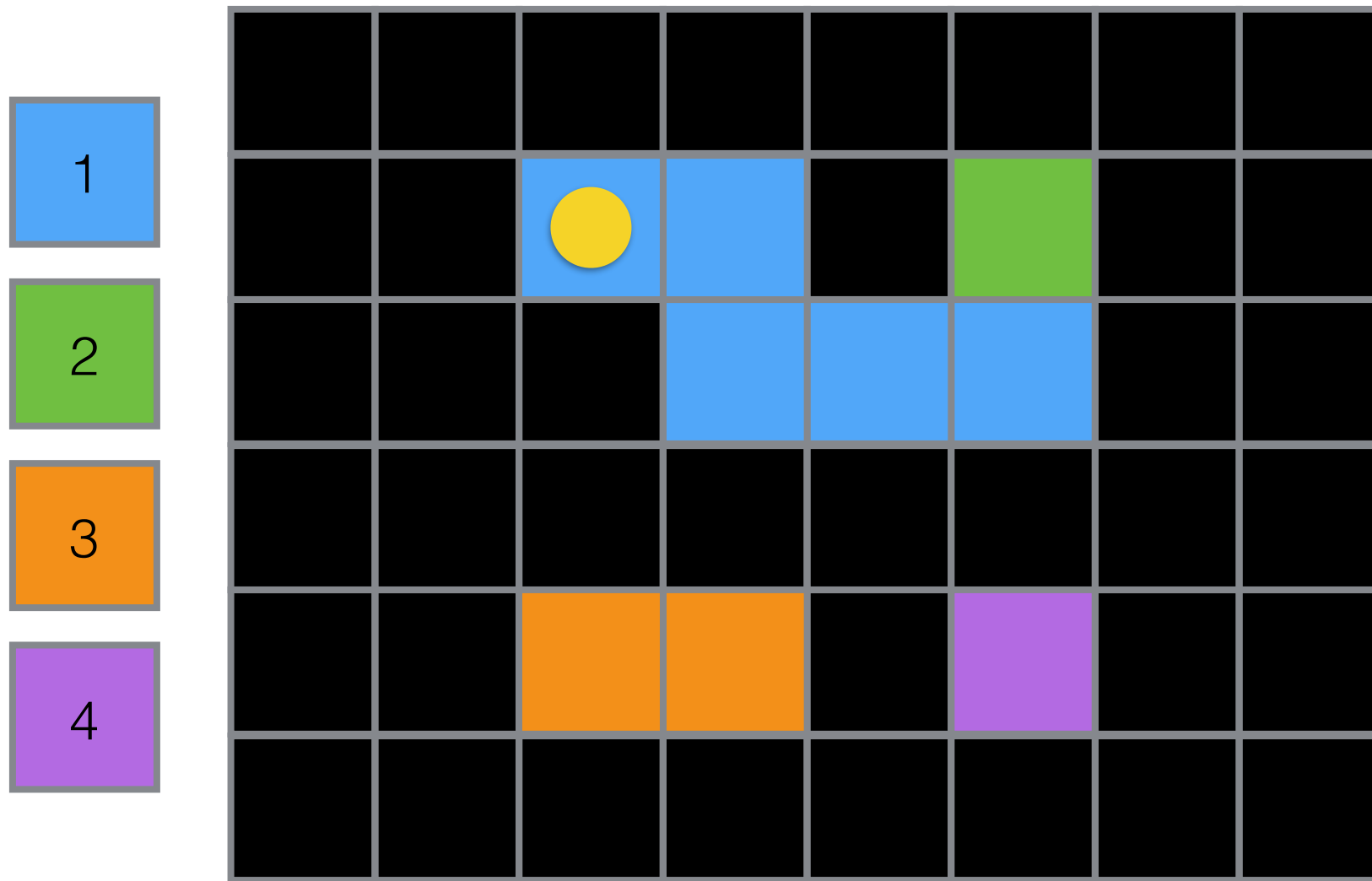
Second Pass

Thresholding: Connected Components

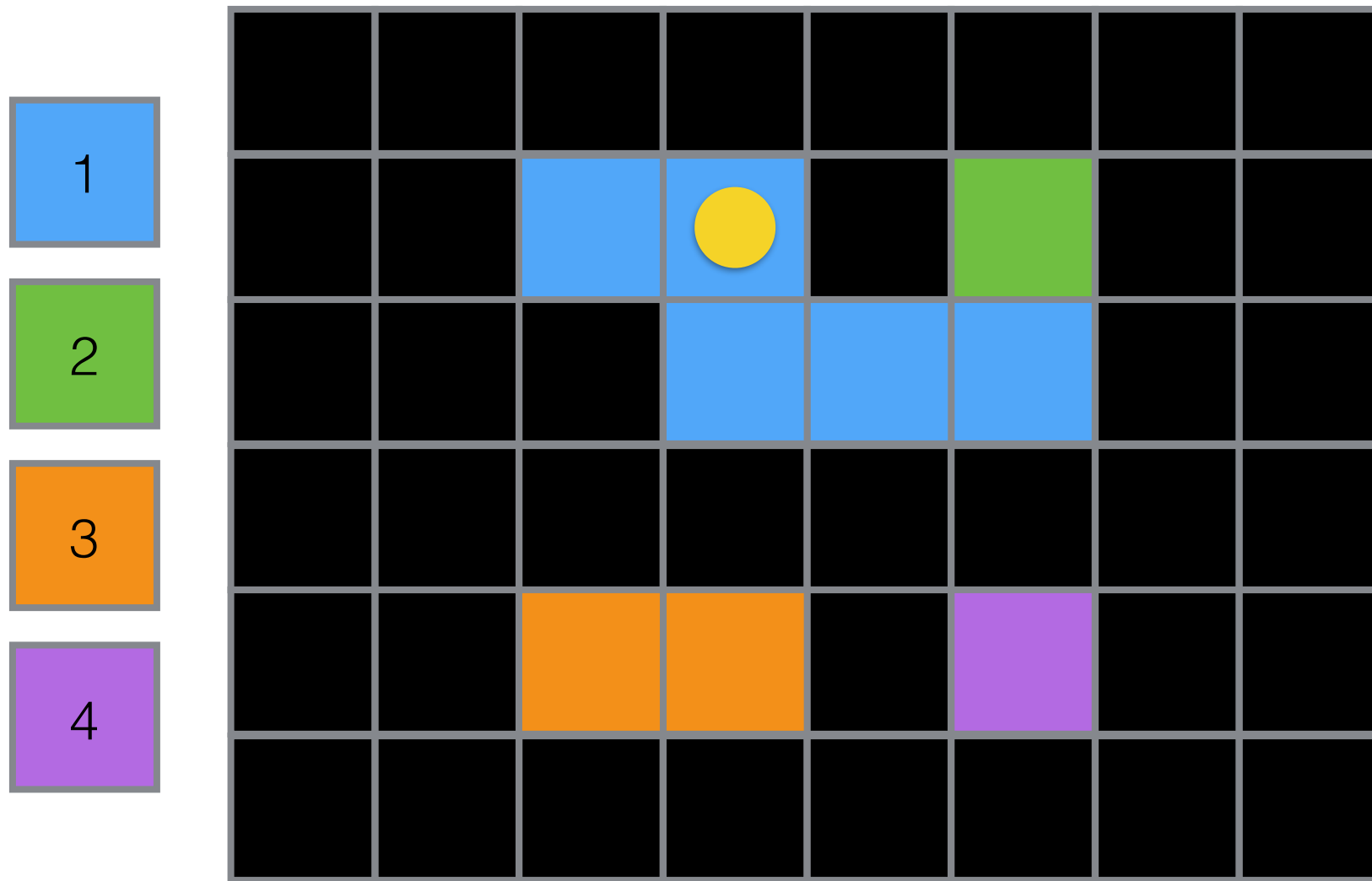


We go through all pixels. For each pixel we set the value of lowest equivalent.

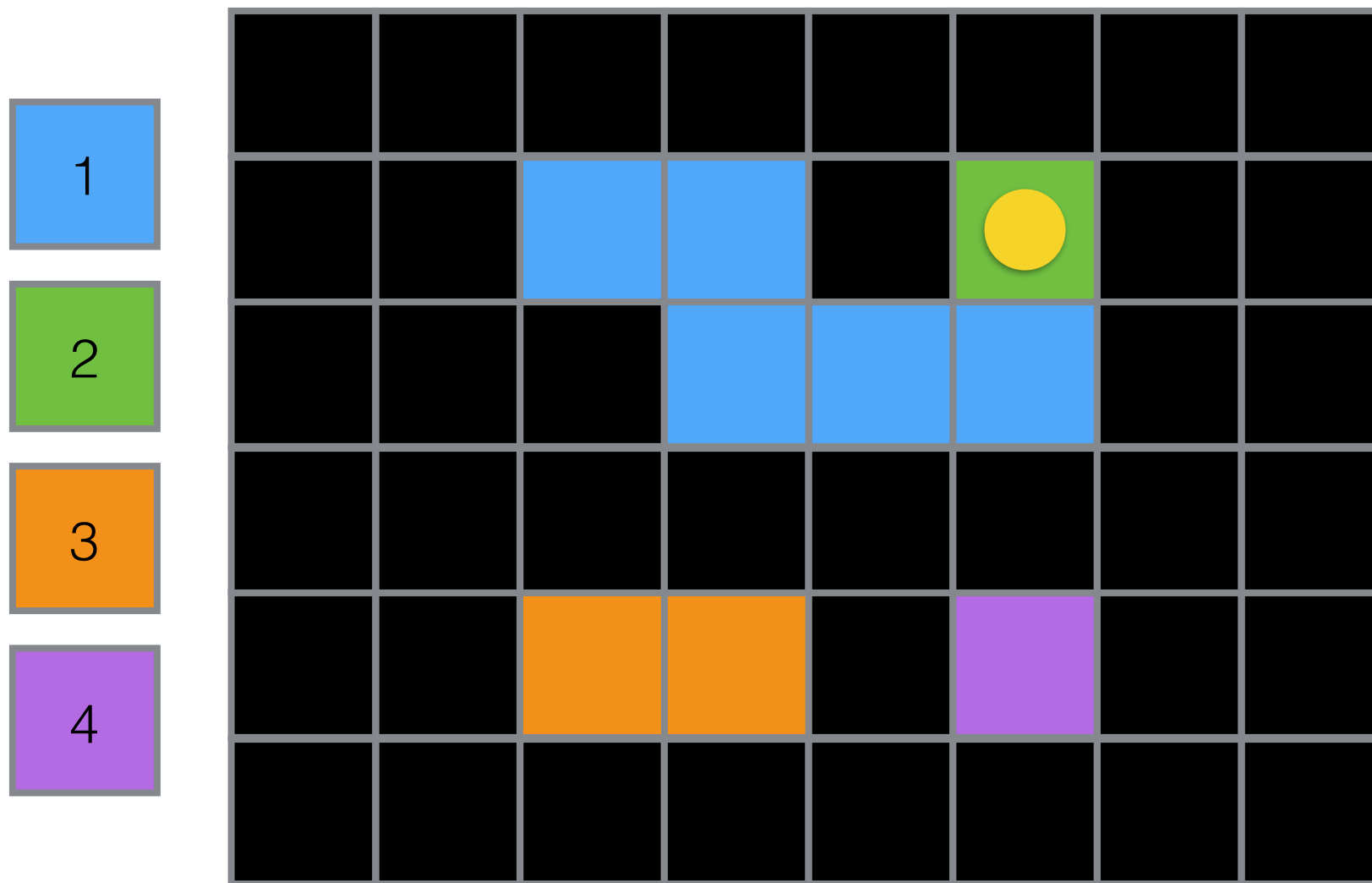
Thresholding: Connected Components



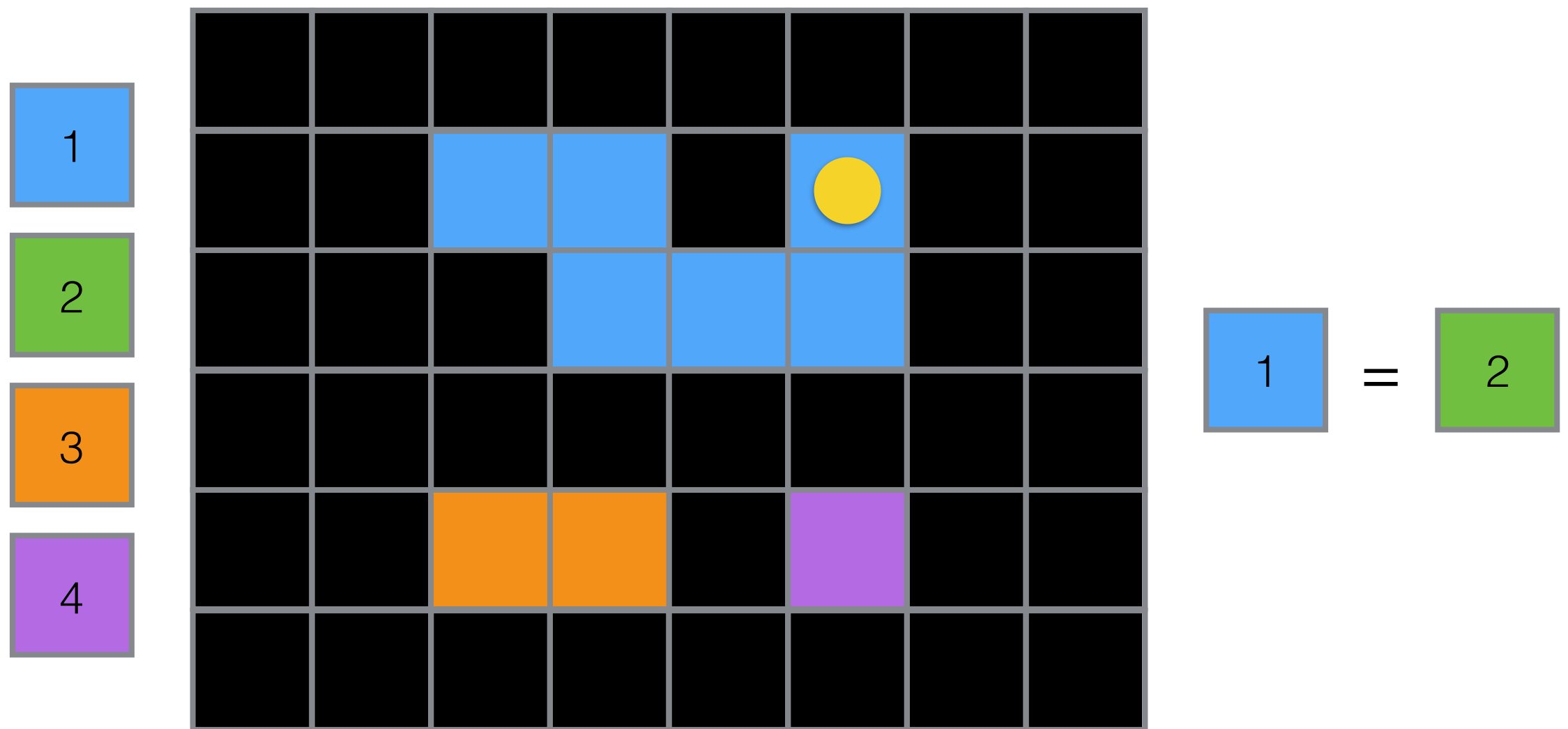
Thresholding: Connected Components



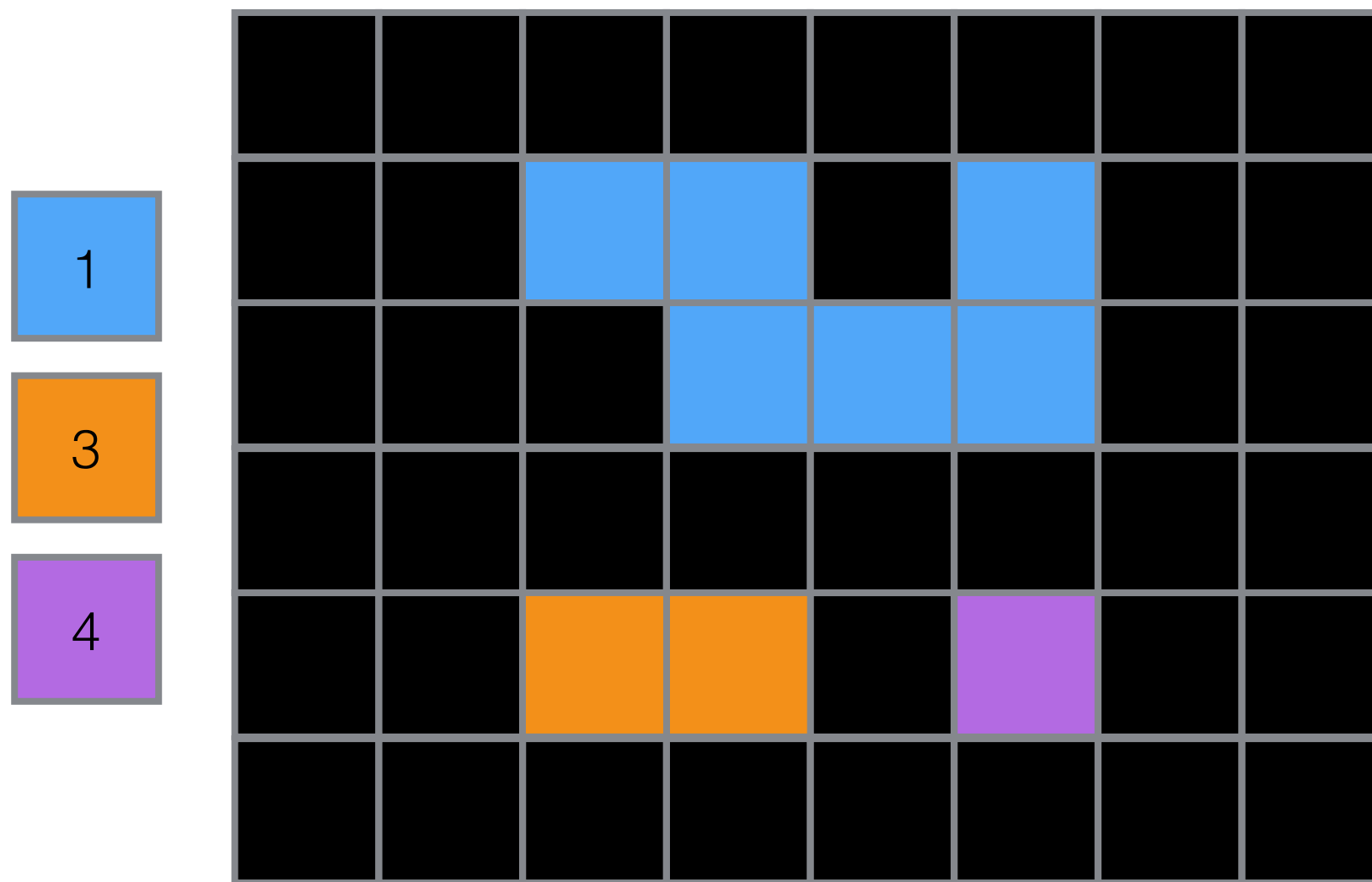
Thresholding: Connected Components



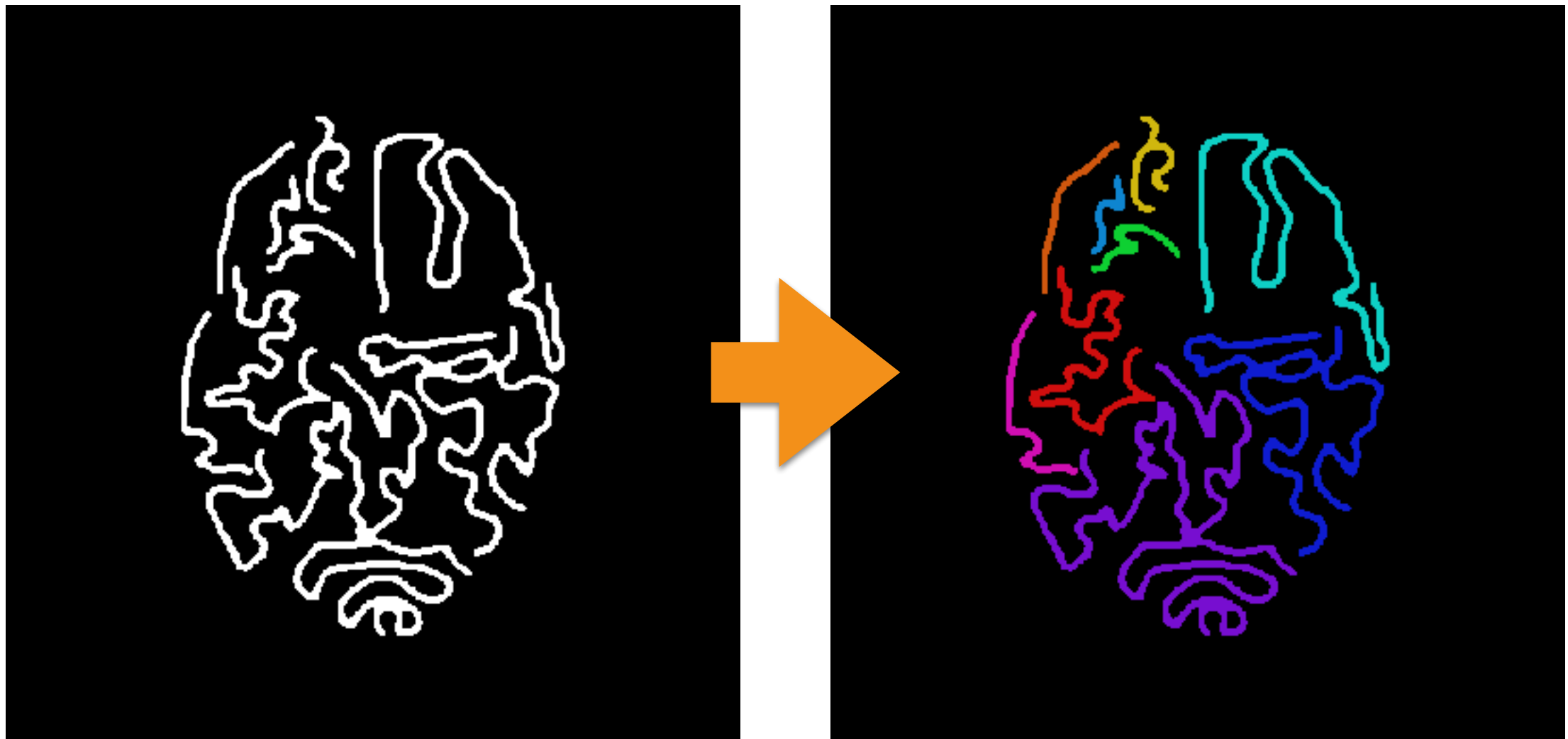
Thresholding: Connected Components



Thresholding: Connected Components



Thresholding: Connected Components Example



Thresholding

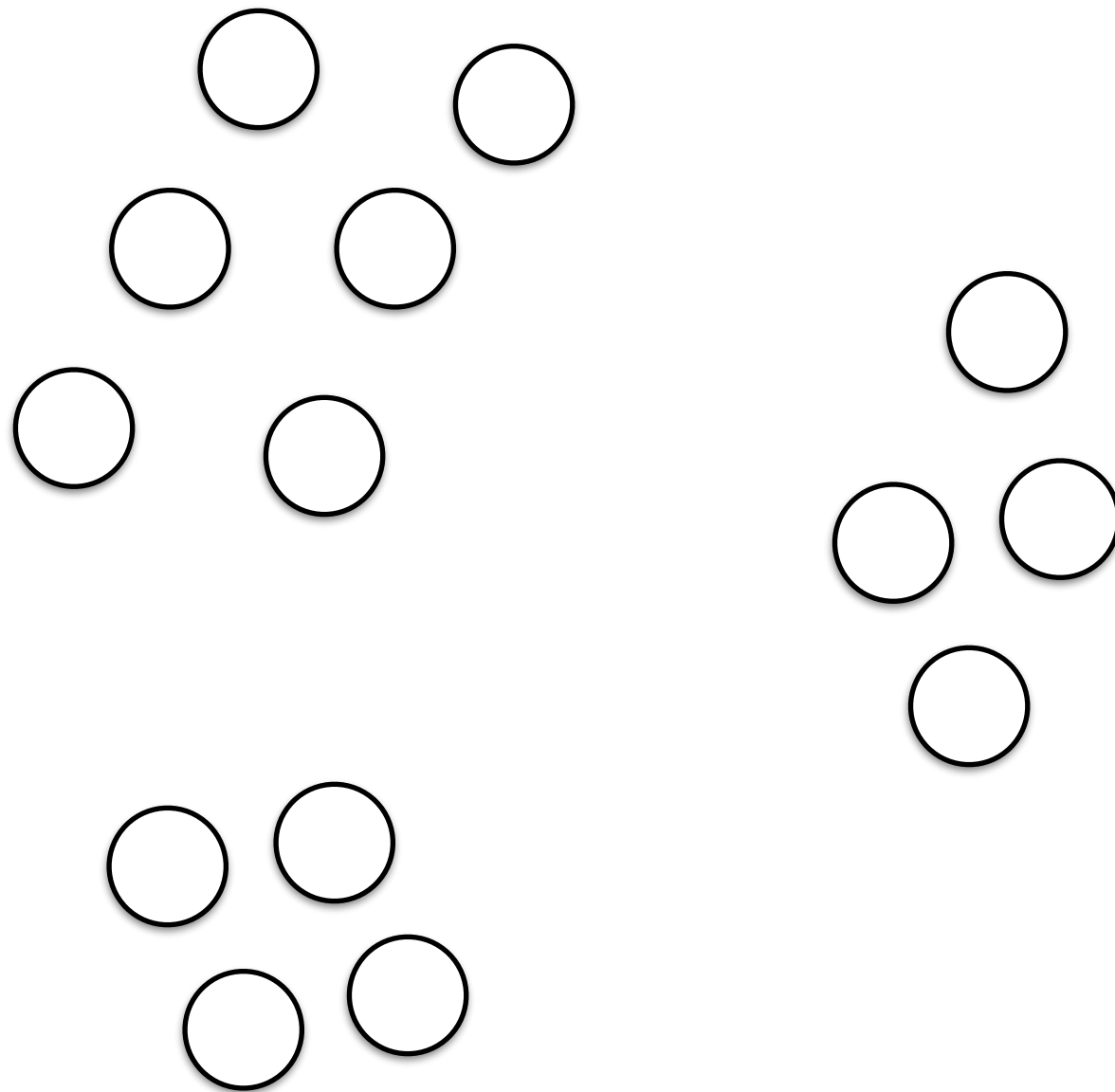
- It works if each object has a unique intensity value/color; this is a very limiting constraint!
 - However, it could be used as a starting point for other algorithms.
- The user needs to set the threshold!
 - The I_t value for each class may be inferred by analyzing the histogram of the input image.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

k-Means

k-Means

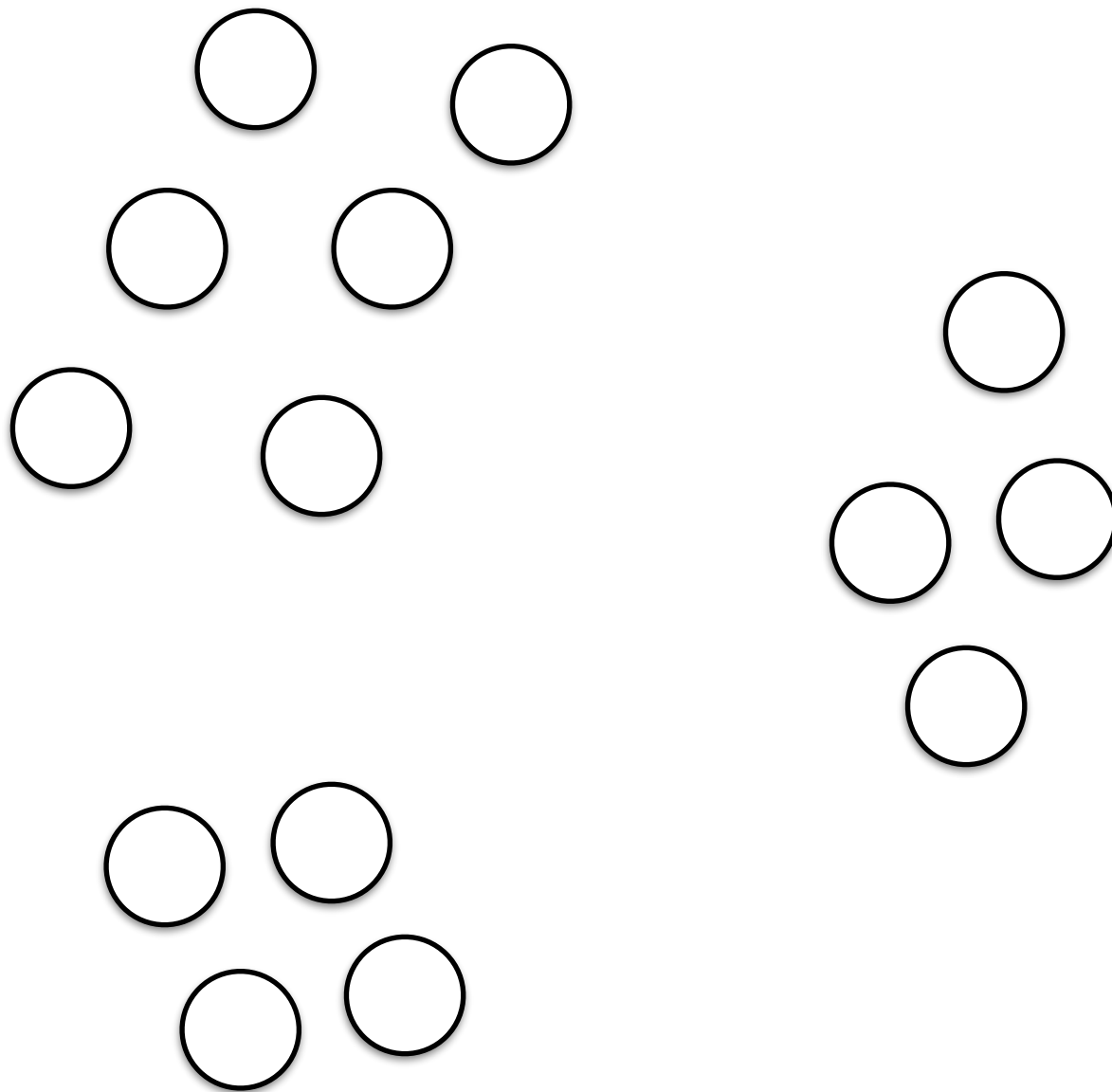
- k-means is a clustering algorithm for clustering n-D vectors/points in an n-D space:
 - A pixel with position (x, y) and intensity I is a 3D vector: $\langle x, y, I \rangle$
 - A voxel with position (x, y, z) and intensity I is a 4D vector: $\langle x, y, z, I \rangle$
- Let's assume we have **k**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

k-Means: How it Works



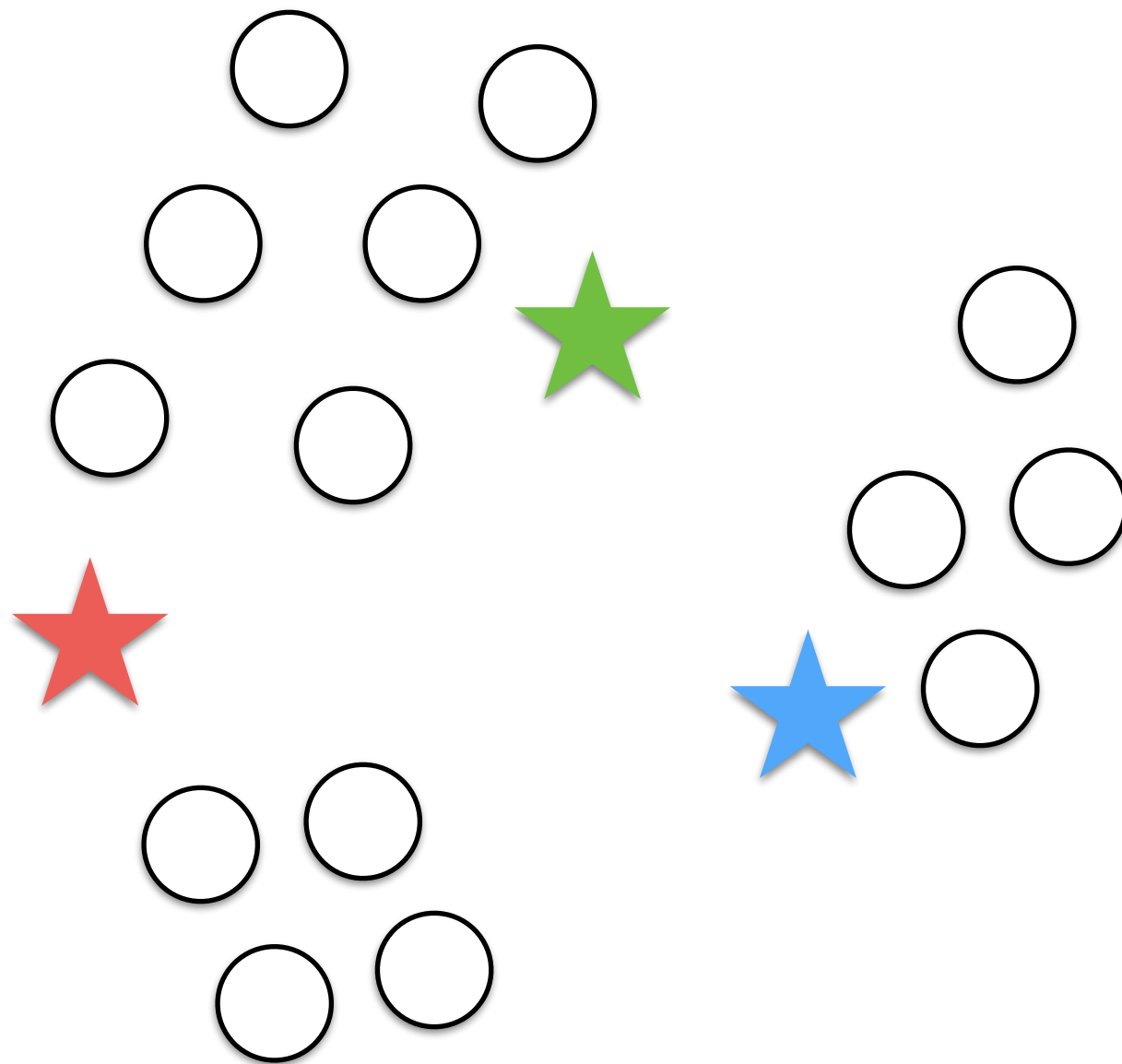
- Let's explain k-Means with 2D points

k-Means: Initialization



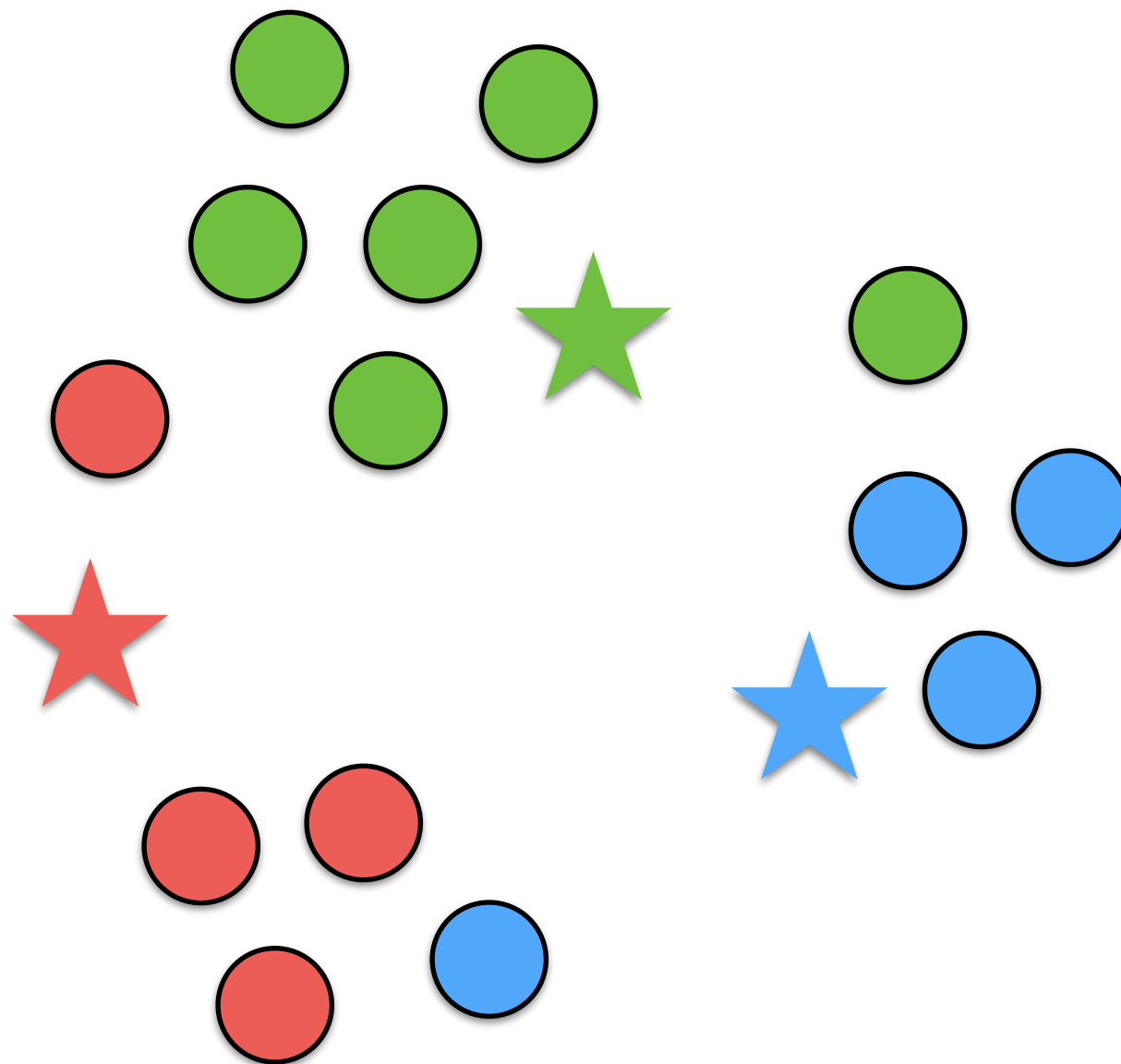
- Let's assume $k = 3$.
- We make a random guess on the k -centroids (the stars).

k-Means: Initialization



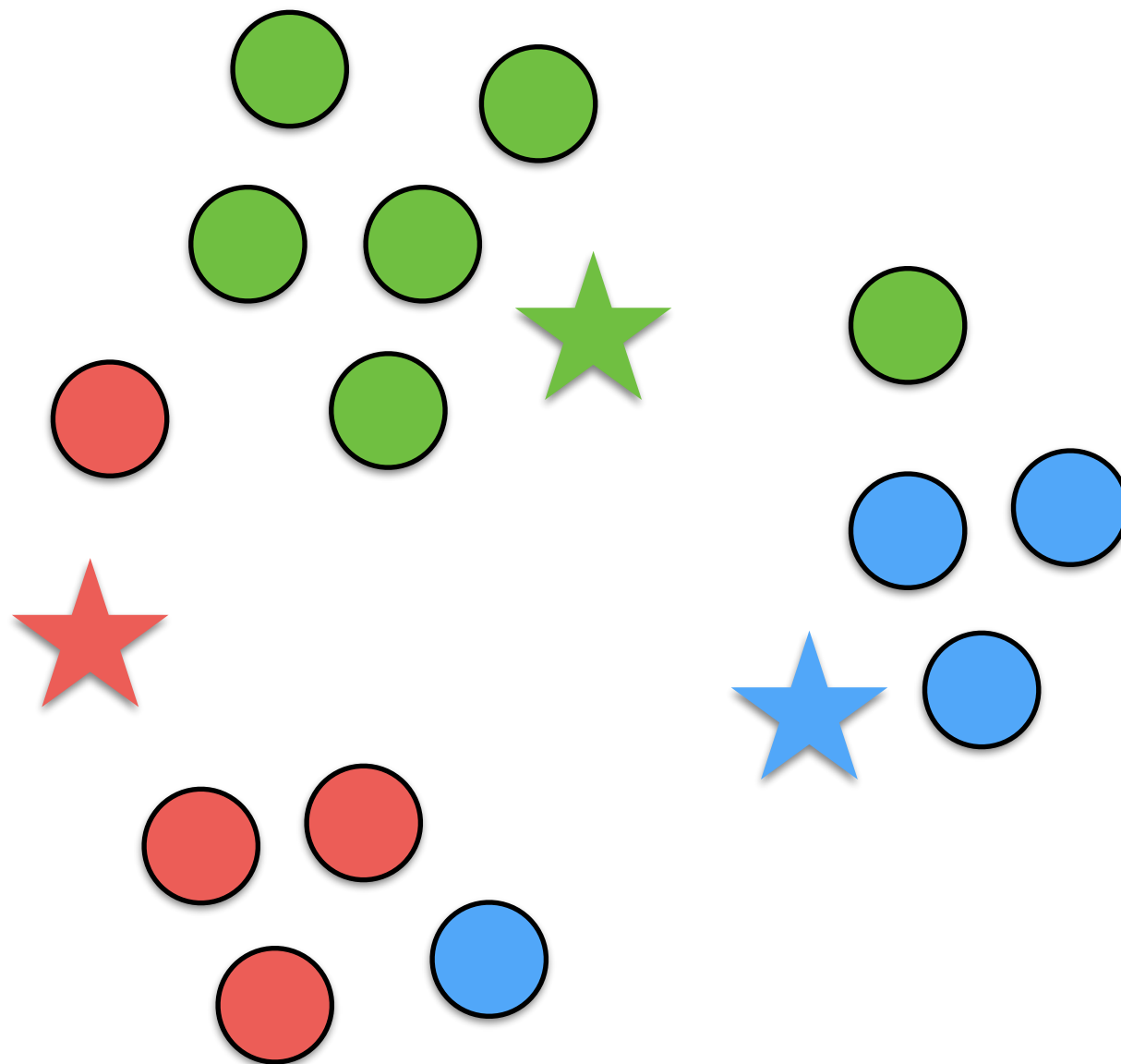
- Let's assume $k = 3$.
- We make a random guess on the k -centroids (the stars).

k-Means: Iteration



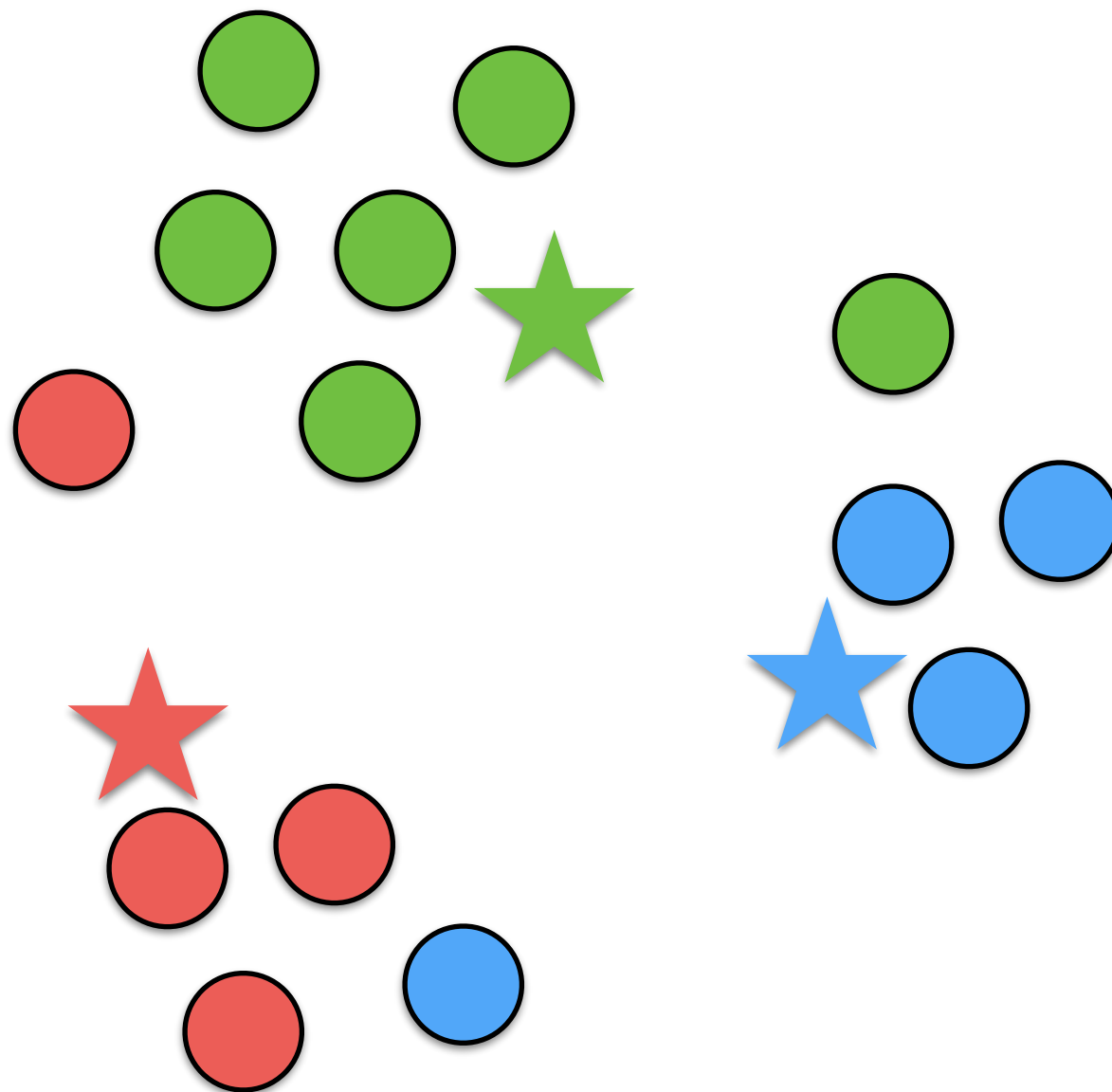
- We now assign a sample to a cluster if the distance (L1, L2, etc.), between a centroid is the minimum.

k-Means: Iteration



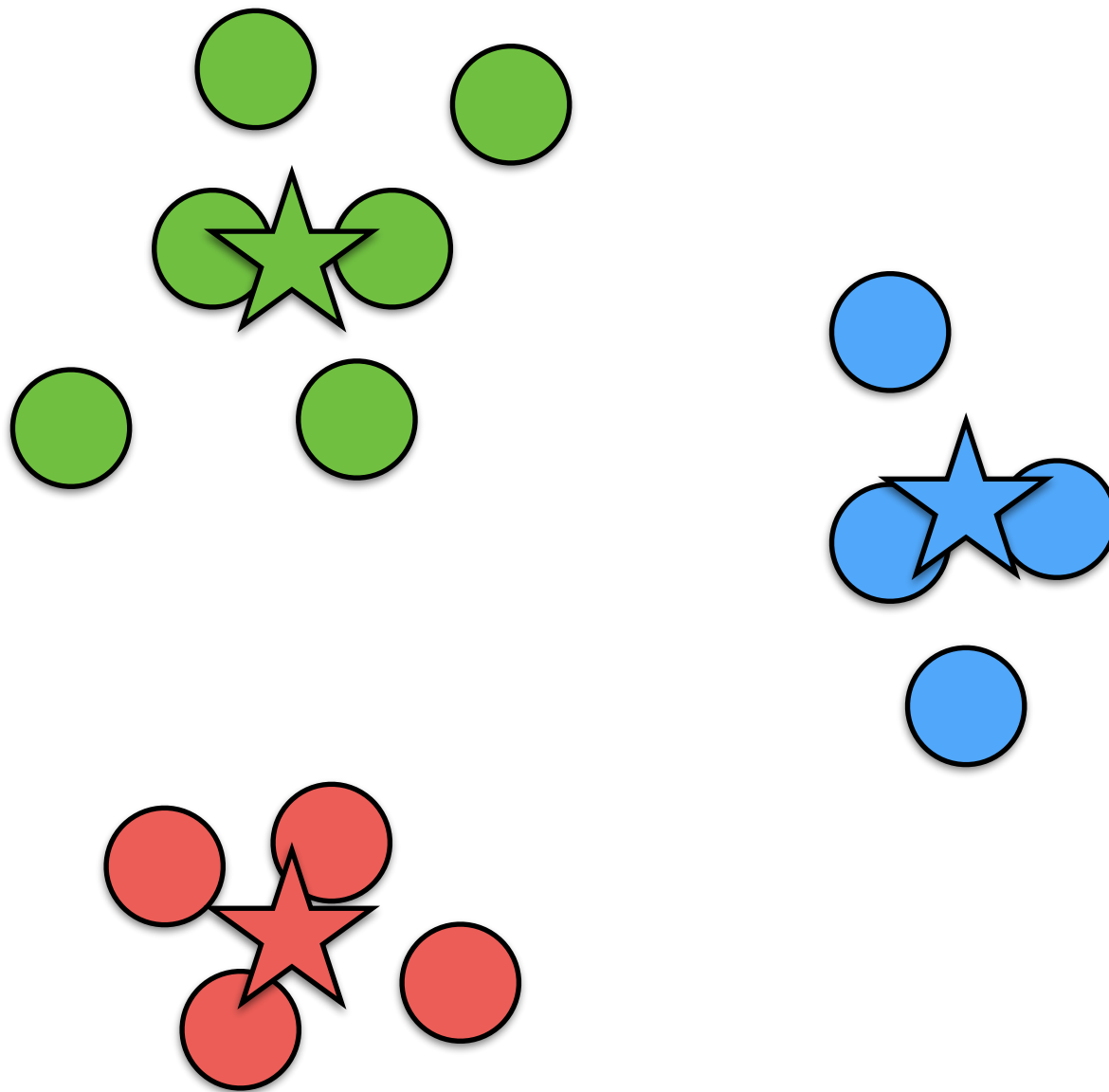
- We re-compute the centroid as the mean of samples of a cluster.

k-Means: Iteration



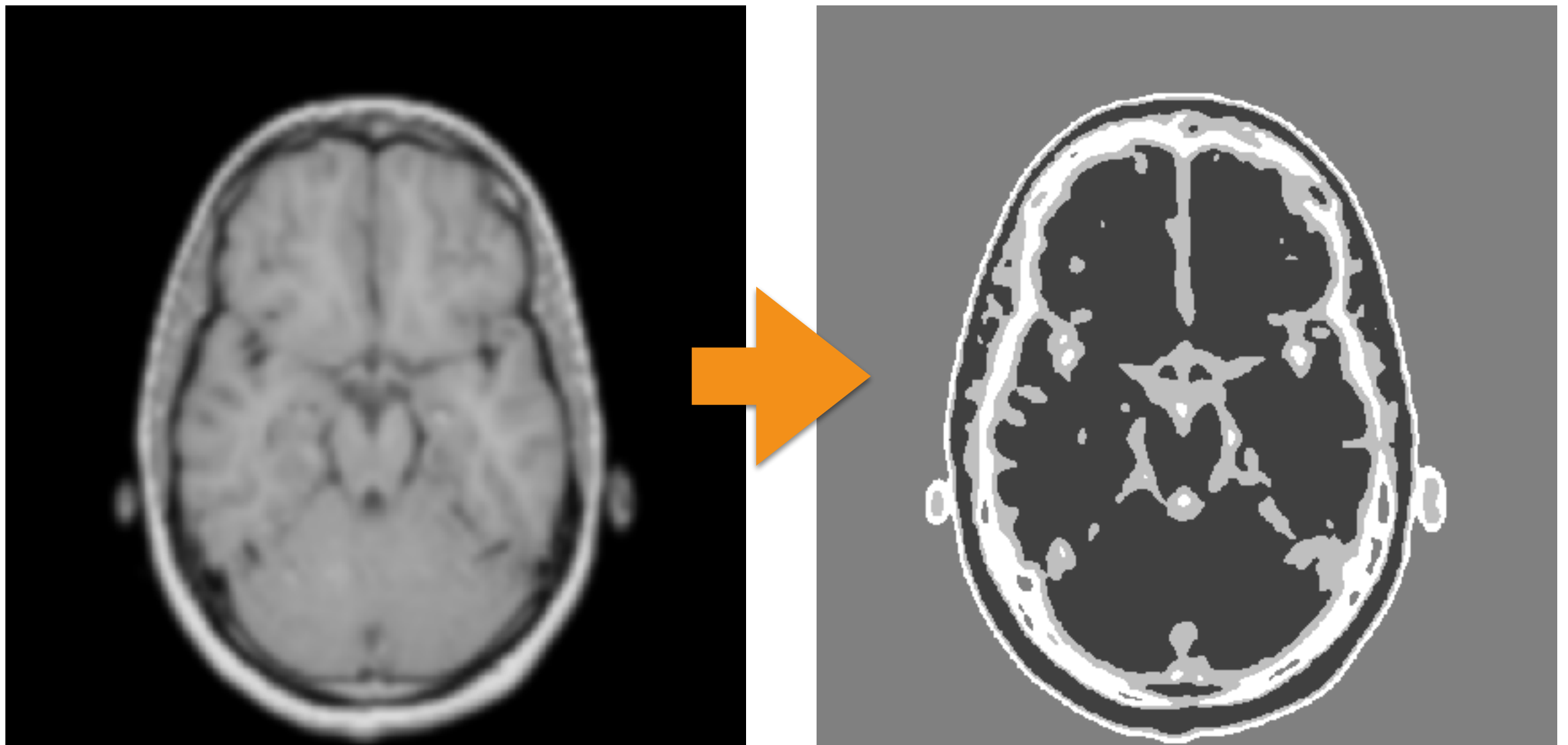
- We repeat the process until convergence (no more changes) or after m iterations.

k-Means: Iteration



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

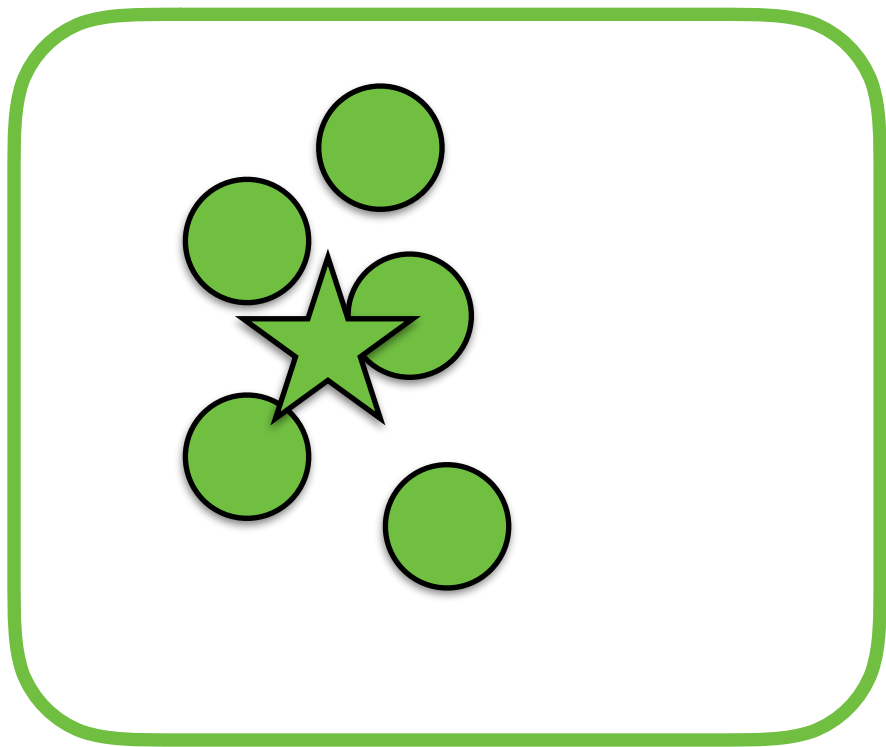
k-Means Example



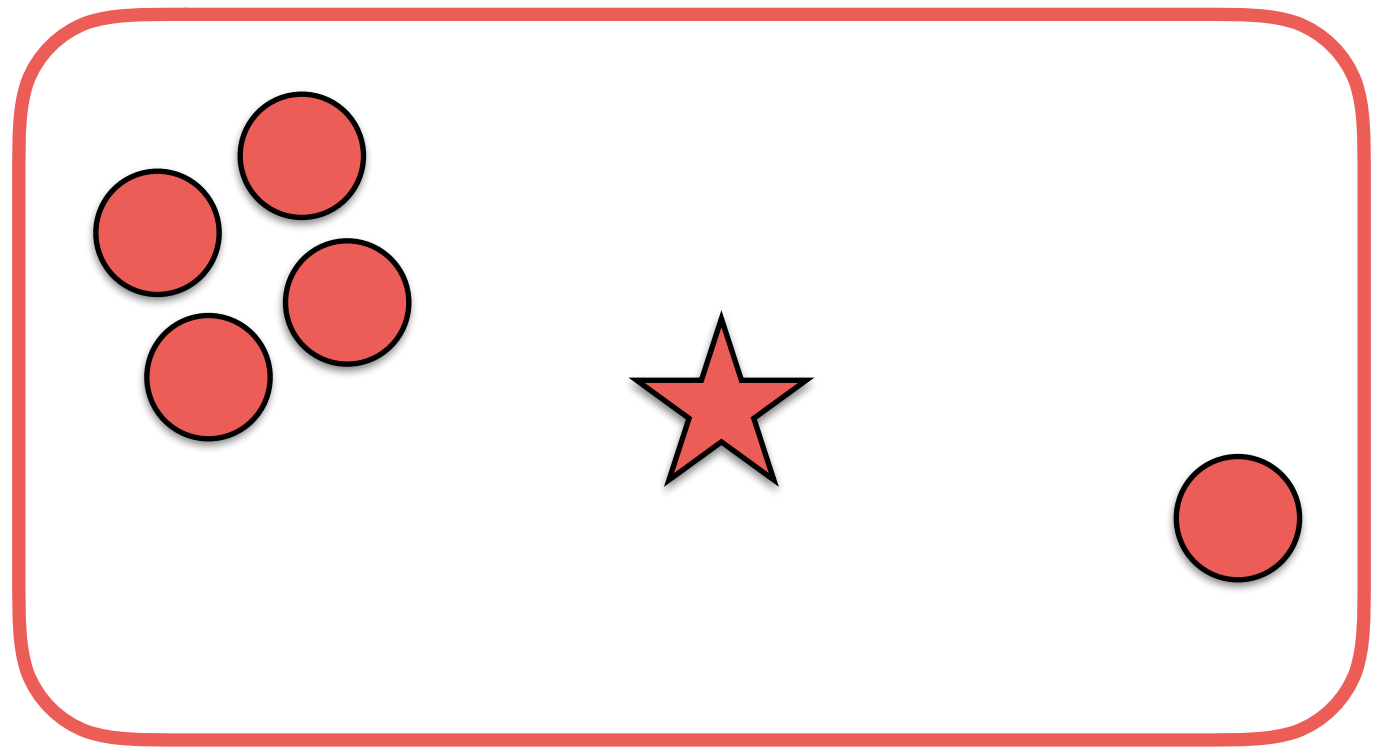
k-Means

- The method is fully automatic, we do not need to set threshold!
- Disadvantages:
 - we need to know how many objects (including the background) are in the image
 - Outliers
 - the method **may not** converge!
 - we need to set a **maximum** number of iterations

k-Means: Outliers

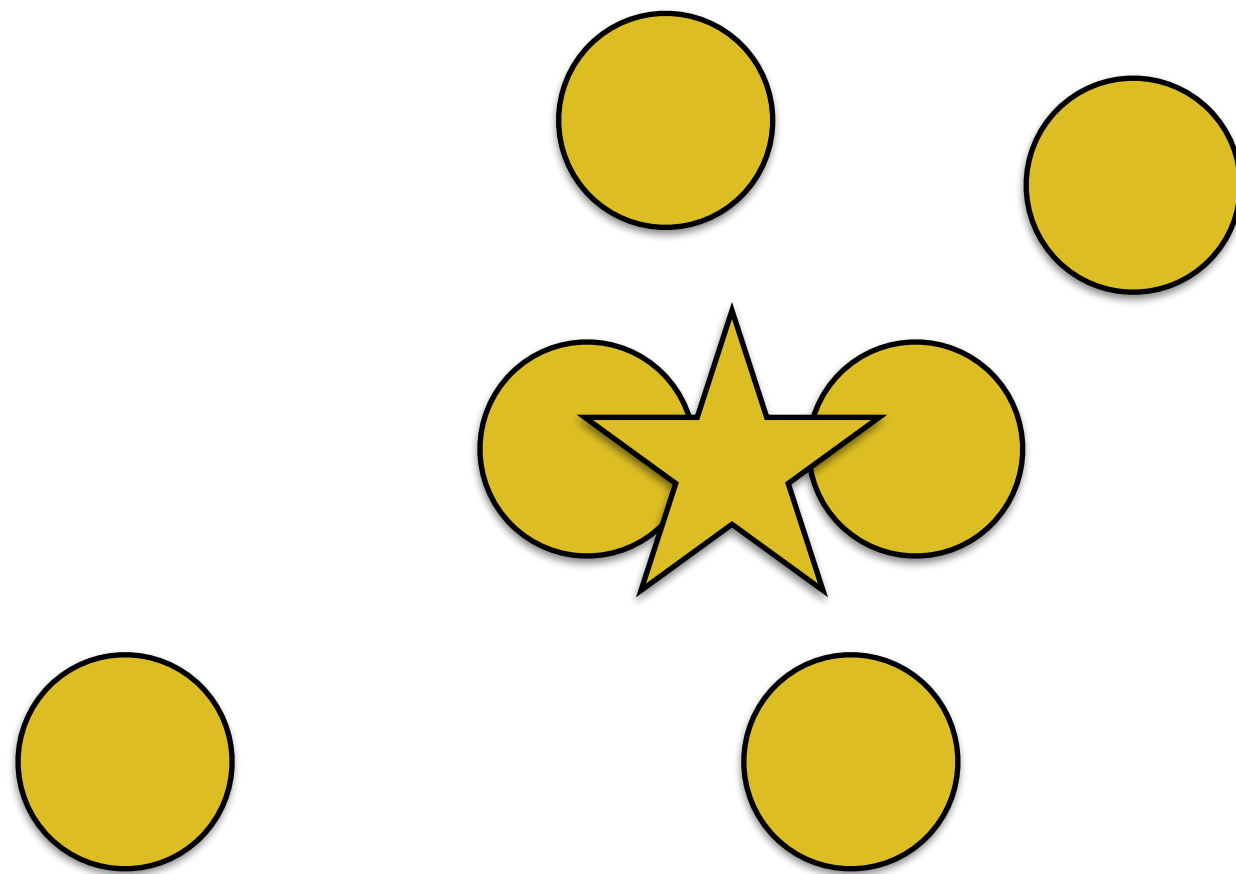


Cluster A



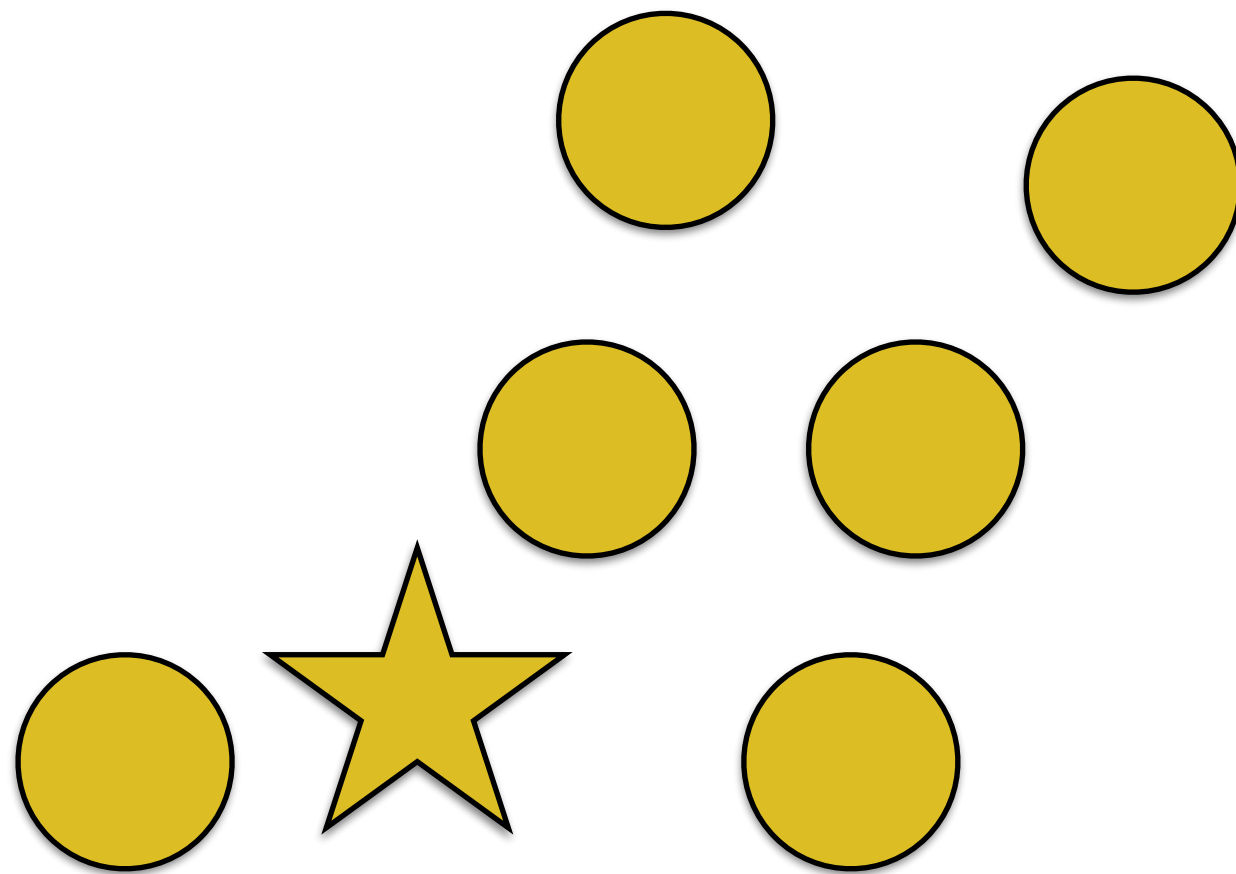
Cluster B

k-Means: Oscillation



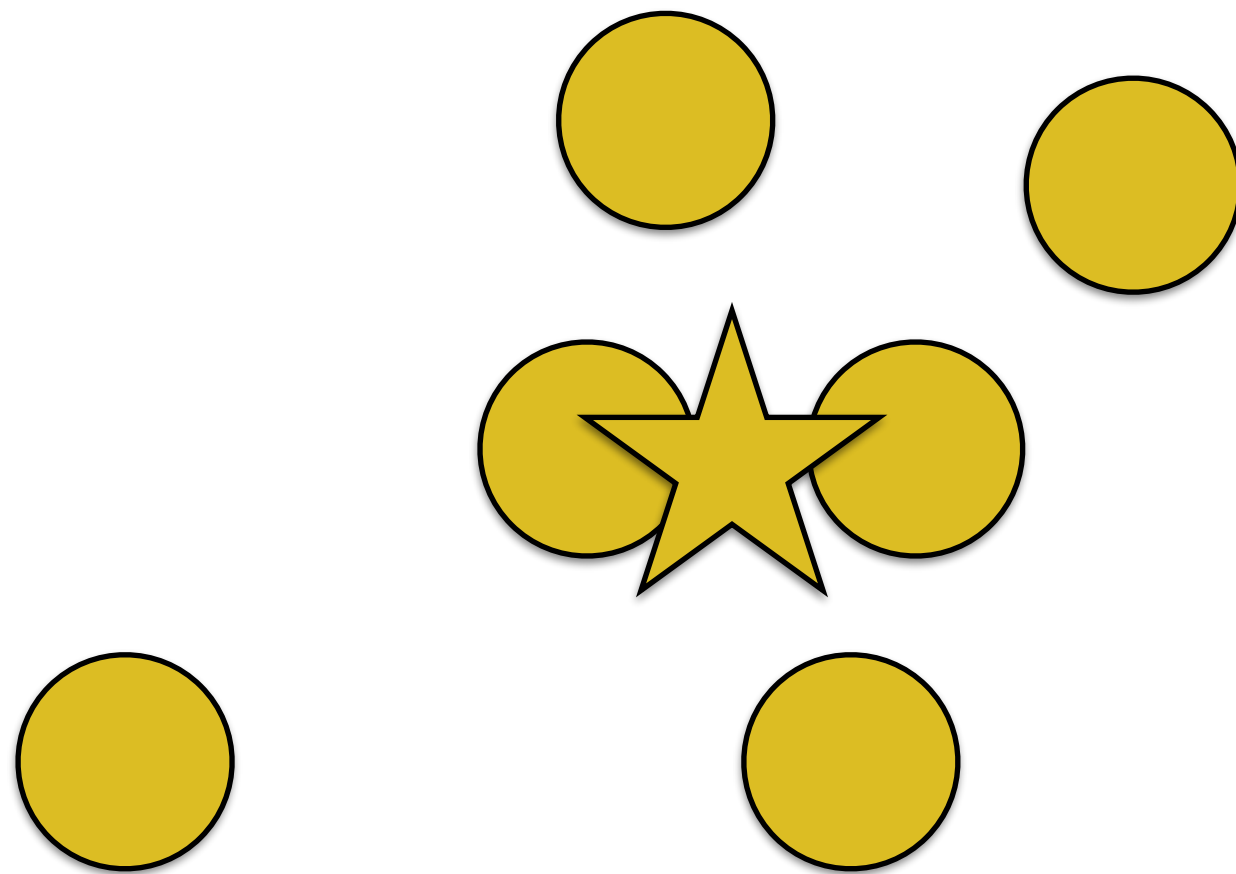
Even Iteration

k-Means: Oscillation



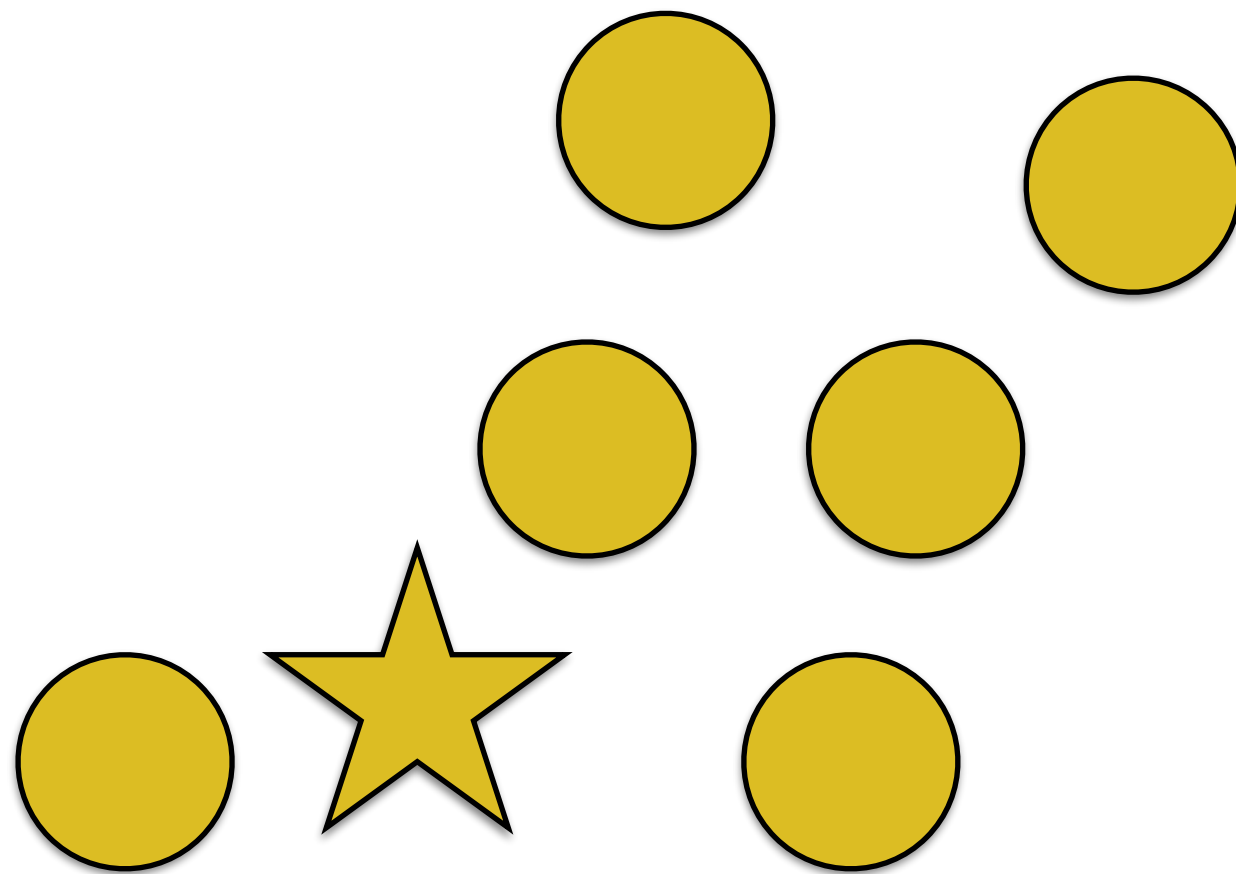
Odd Iteration

k-Means: Oscillation



Even Iteration

k-Means: Oscillation



Odd Iteration

Active Contour Model aka Snakes

Snakes

- A snake is a parametric curve:

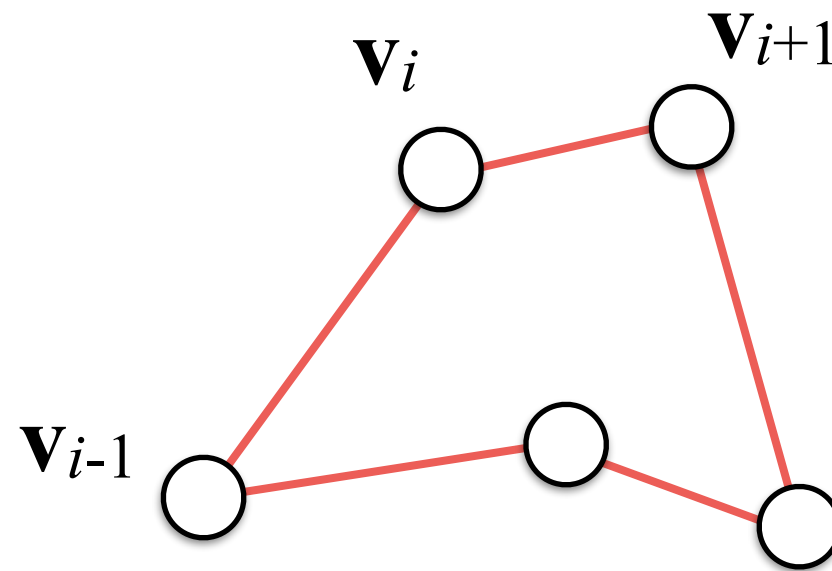
$$\mathbf{v}(t) = (x(t); y(t)) \quad t \in [0, 1]$$

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

Snakes

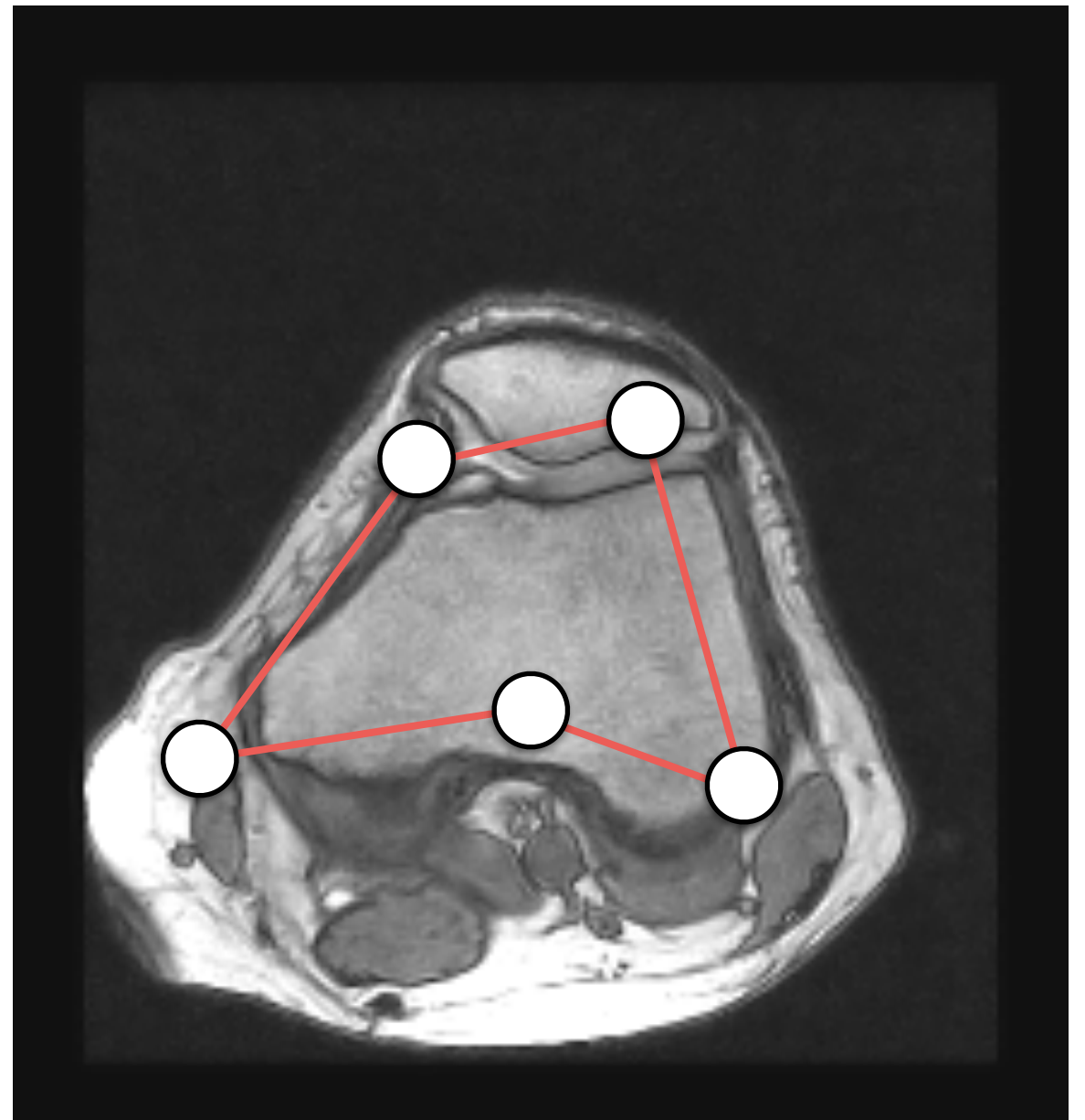
- The snake curve is defined by a set of control point that is defined as:

$$\mathbf{C} = \{\mathbf{v}_i | i \in [1, n]\} \text{ where } \mathbf{v}_i = (x_i, y_i)$$



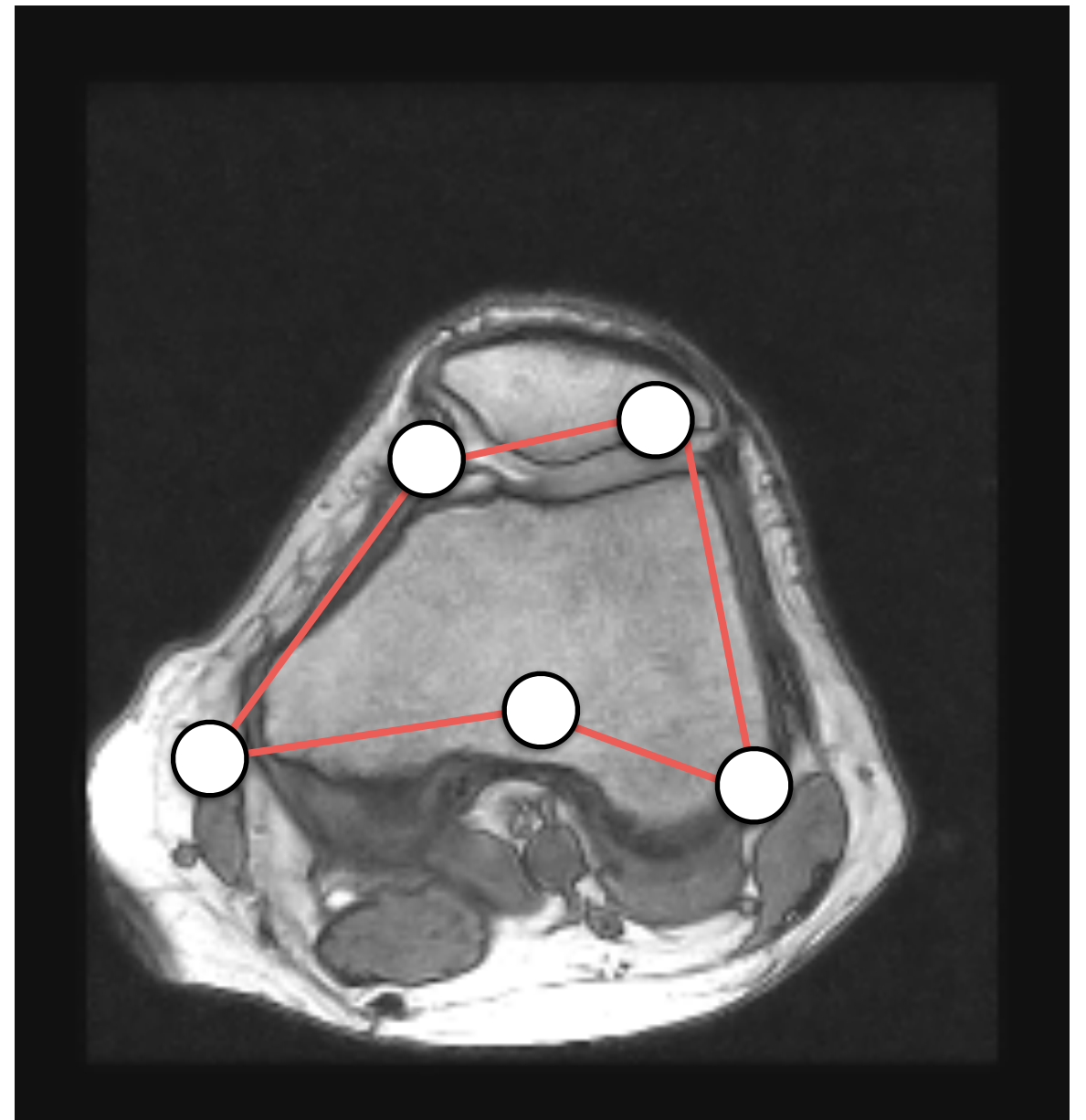
Snakes

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



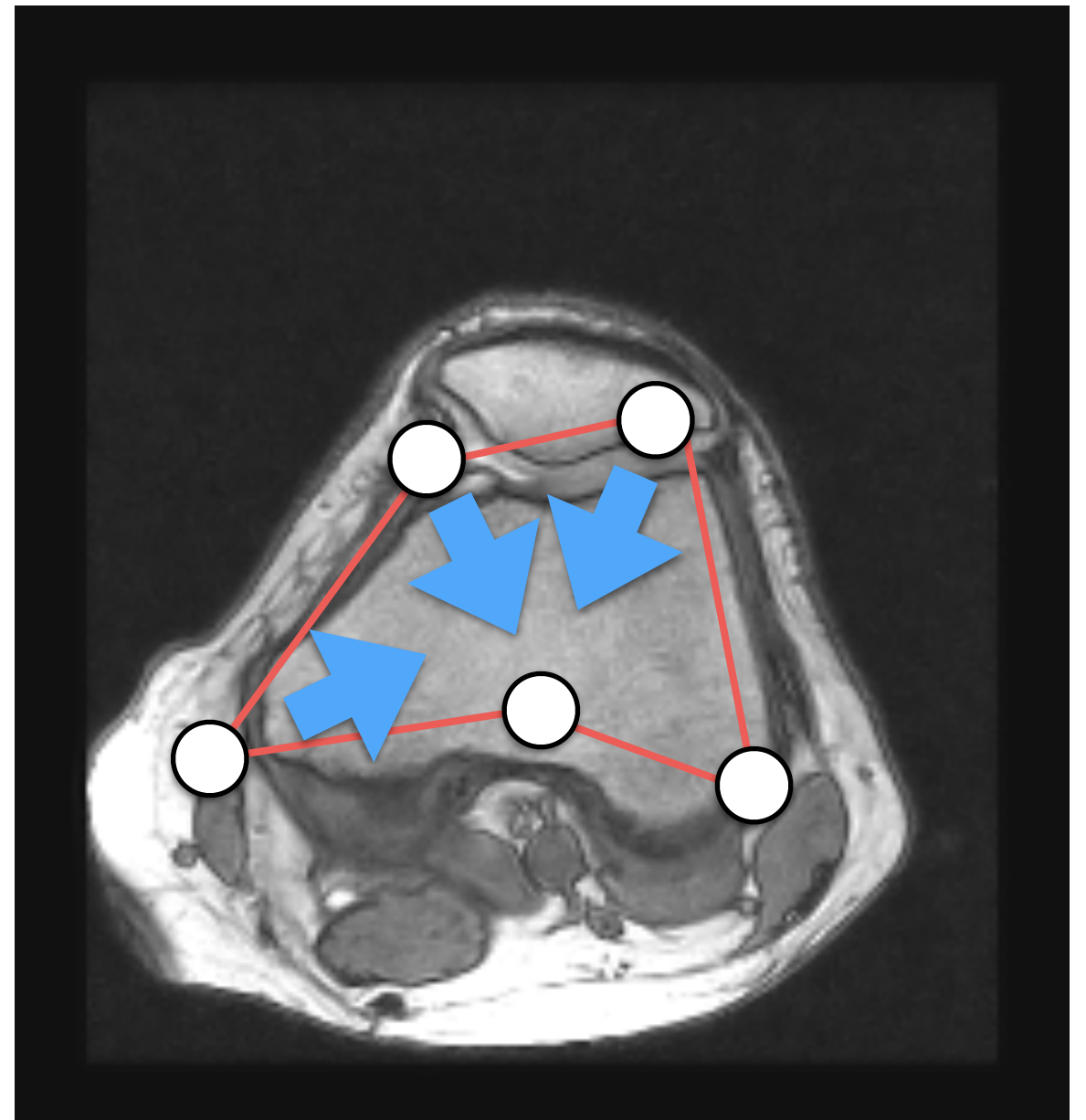
Snakes

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



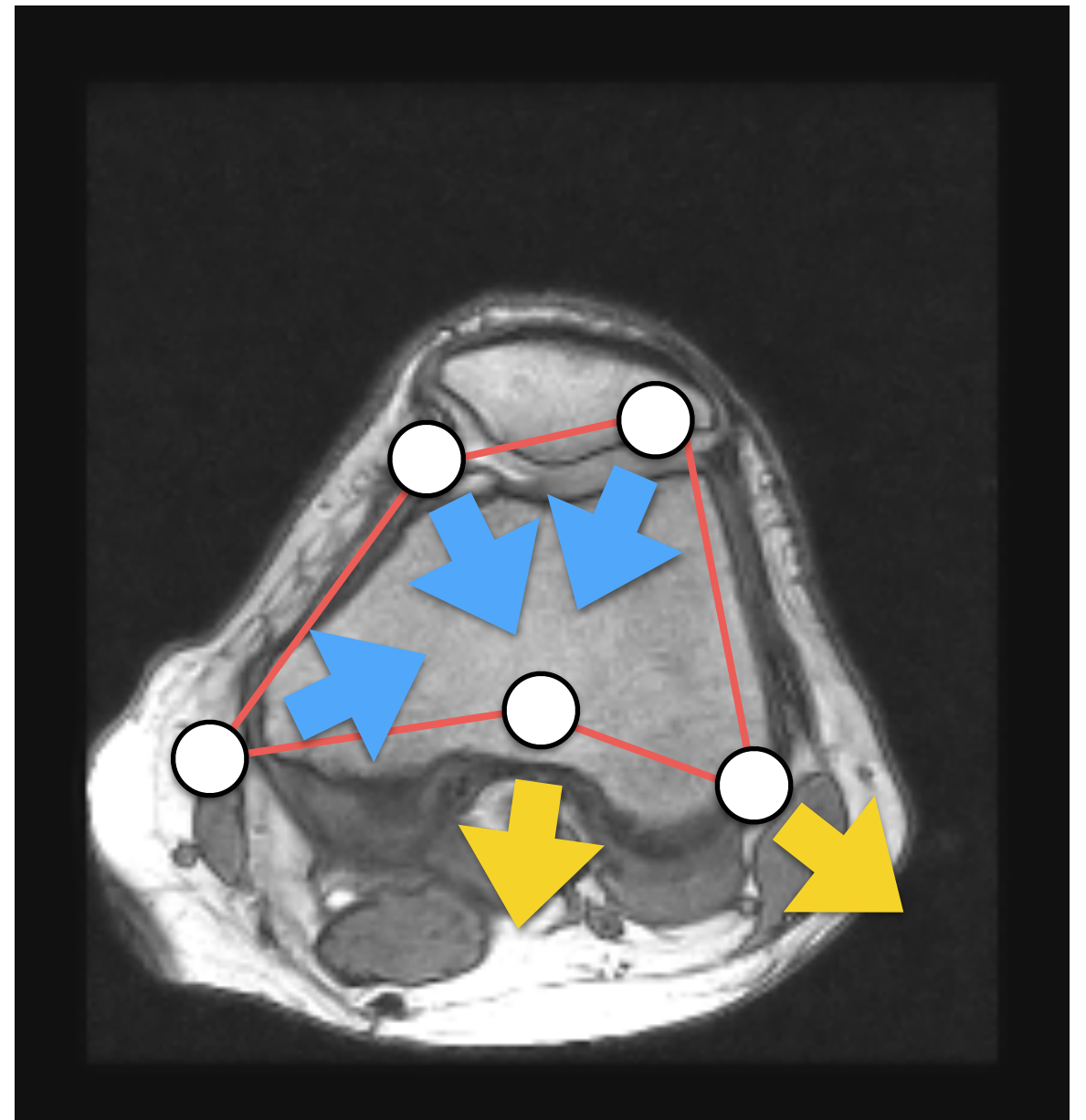
Snakes

- 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_v is associated with the curve.
- We deform control points by minimizing E_v ; i.e., we solve an optimization problem.

Snakes

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

$$E_{\mathbf{v}} = E_{\text{internal}} + E_{\text{external}} + E_{\text{constraint}}$$

Snakes: Internal Energy

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

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

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 curve due to bending. It is defined per point as

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

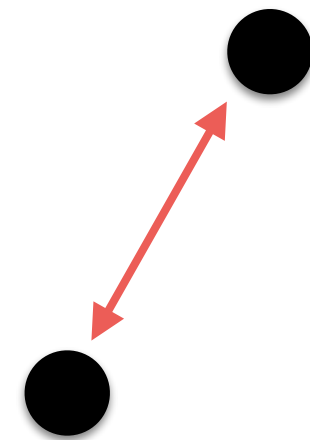
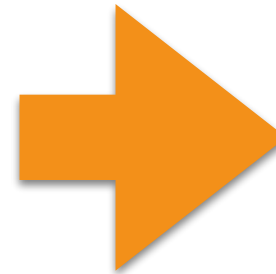
- The total energy is defined as

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

Snakes: Internal Energy

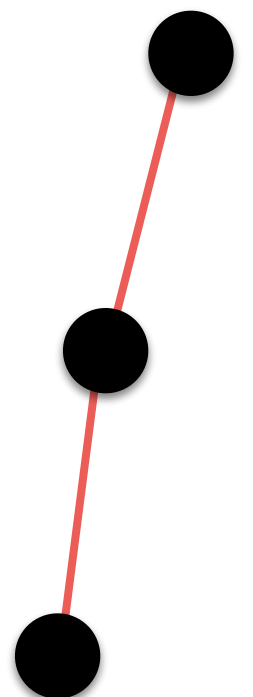
- The first term is an elastic energy:

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



- The second term is a bending energy:

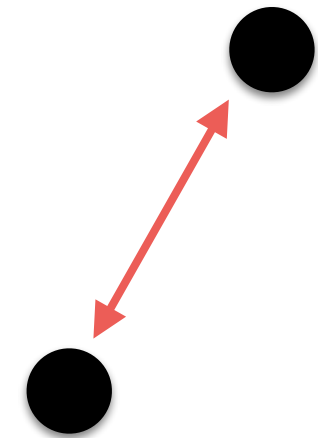
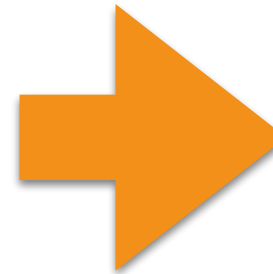
$$\frac{d^2\mathbf{v}(t)}{d^2t} \approx \mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}$$



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:

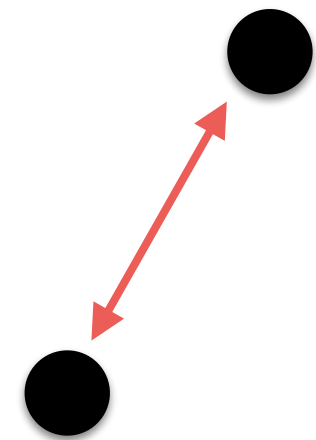
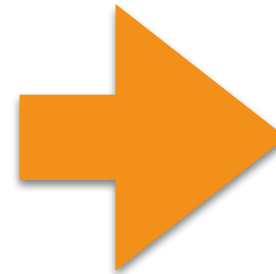
$$\frac{d^2\mathbf{v}(t)}{d^2t} \approx \mathbf{v}_{i+1} - 2\mathbf{v}_i + \mathbf{v}_{i-1}$$



Snakes: Internal Energy

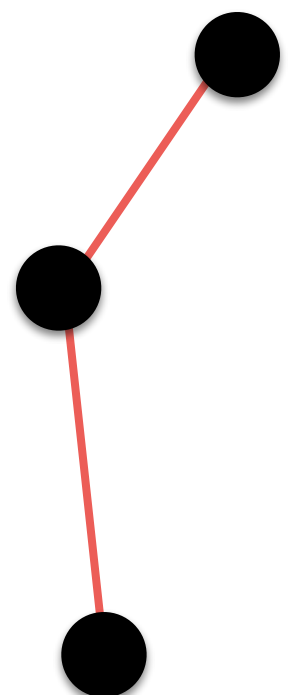
- The first term is an elastic energy:

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



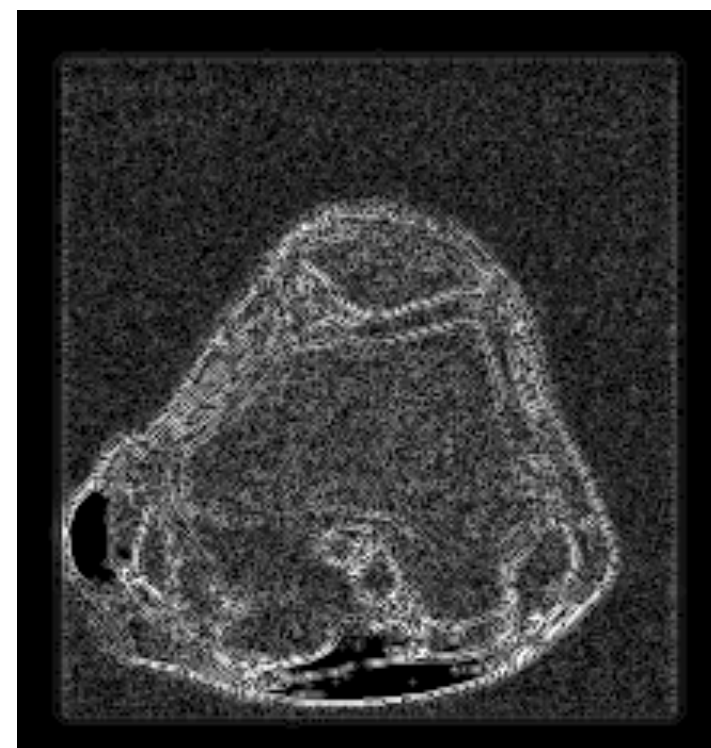
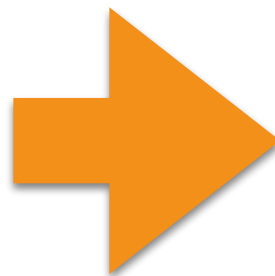
- The second term is a bending energy:

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



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

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

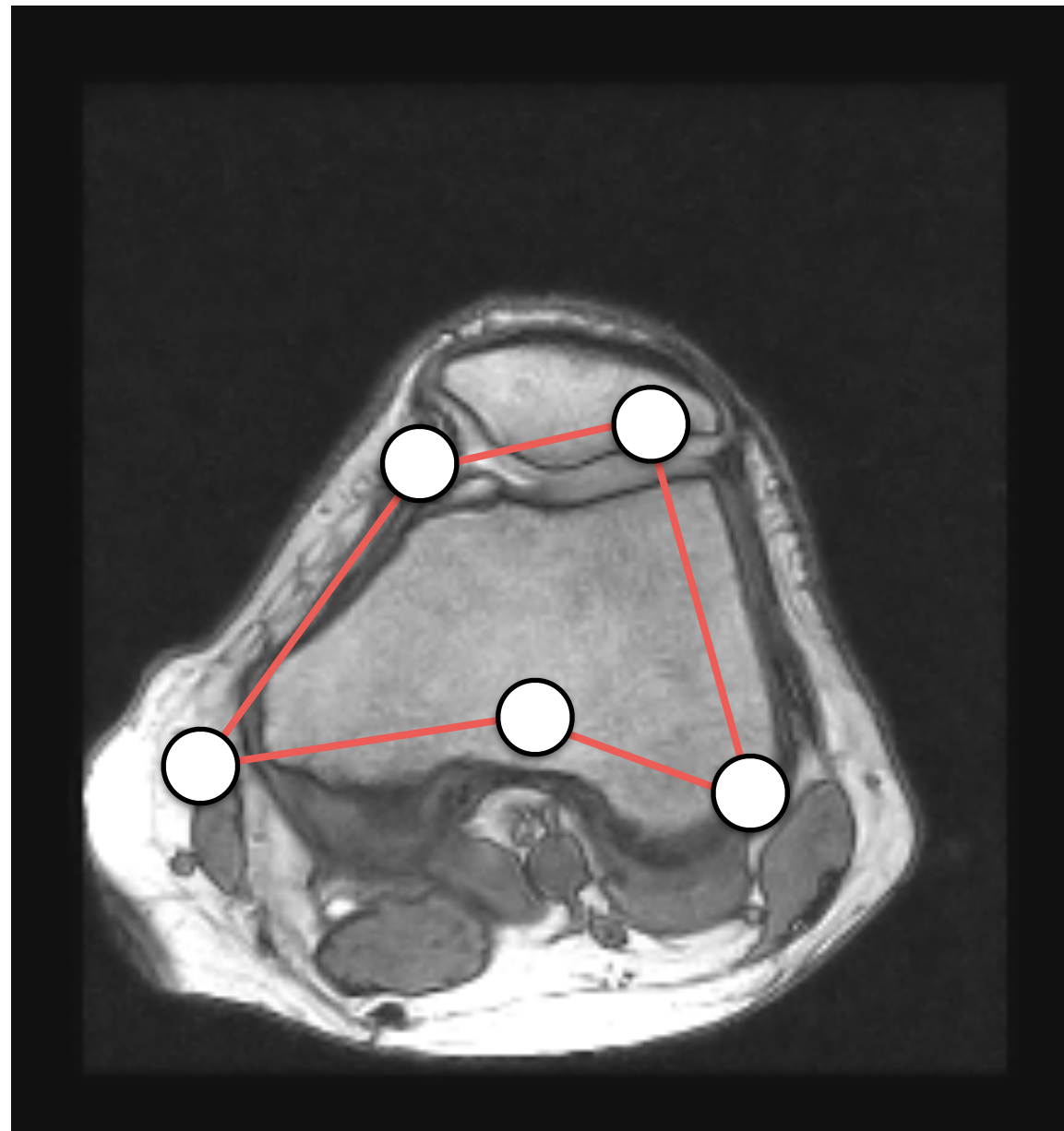
- The total energy is defined as

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

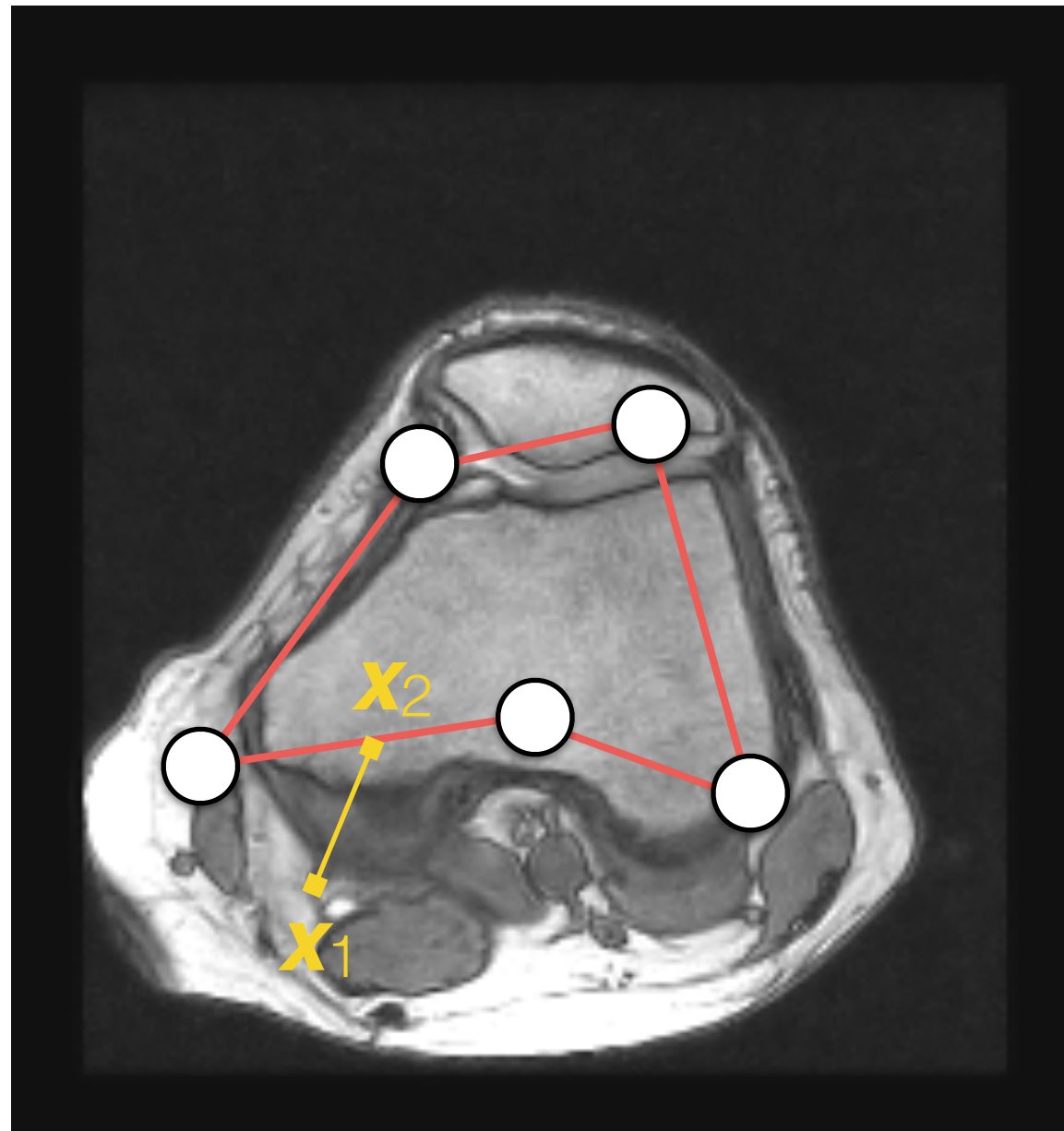
Snakes: Constraint Energy

- 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”: $\frac{1}{r^2}$
 - Spring forces: $-k(\mathbf{x}_1 - \mathbf{x}_2)^2$

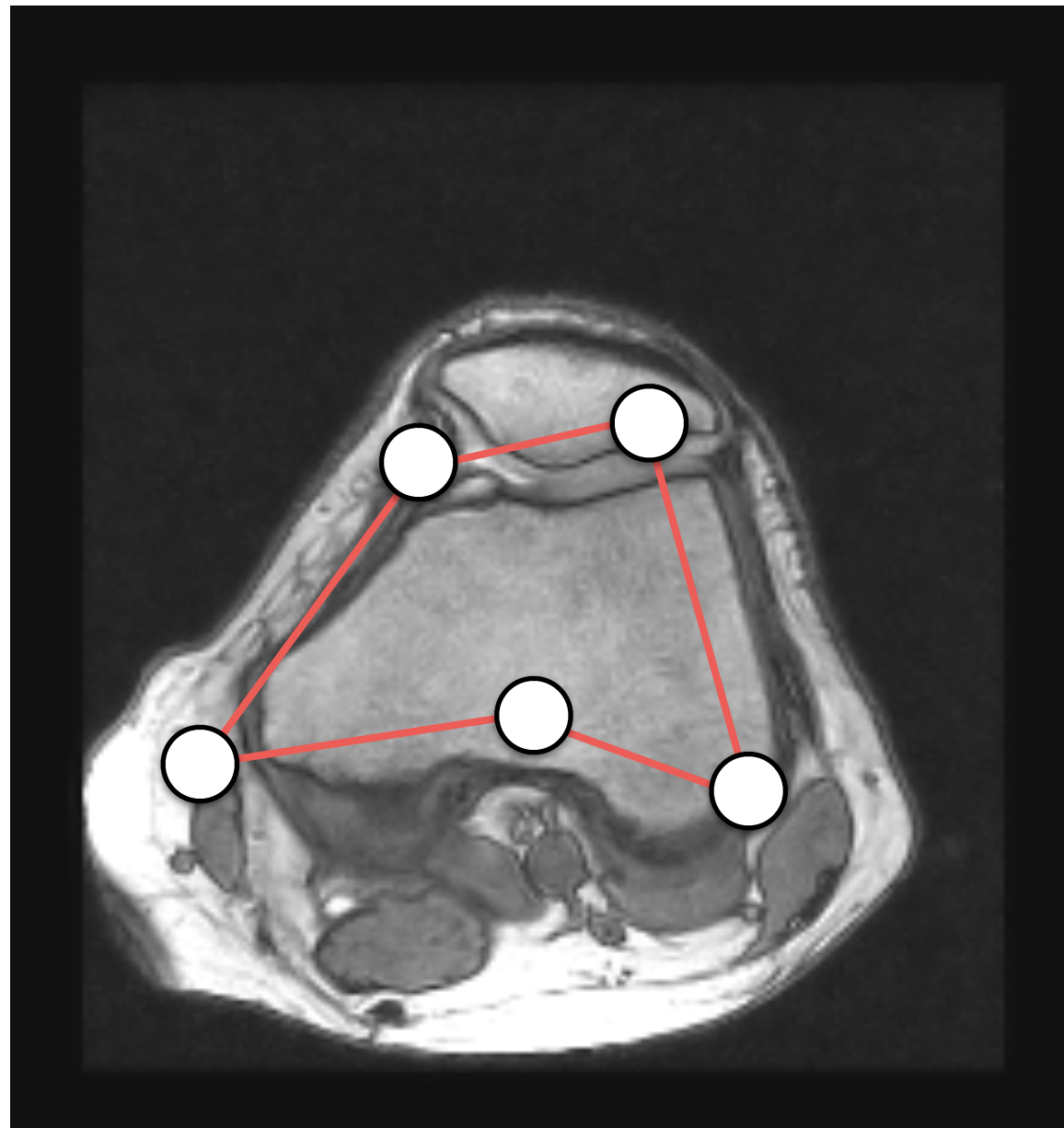
Snakes: Constraint Energy



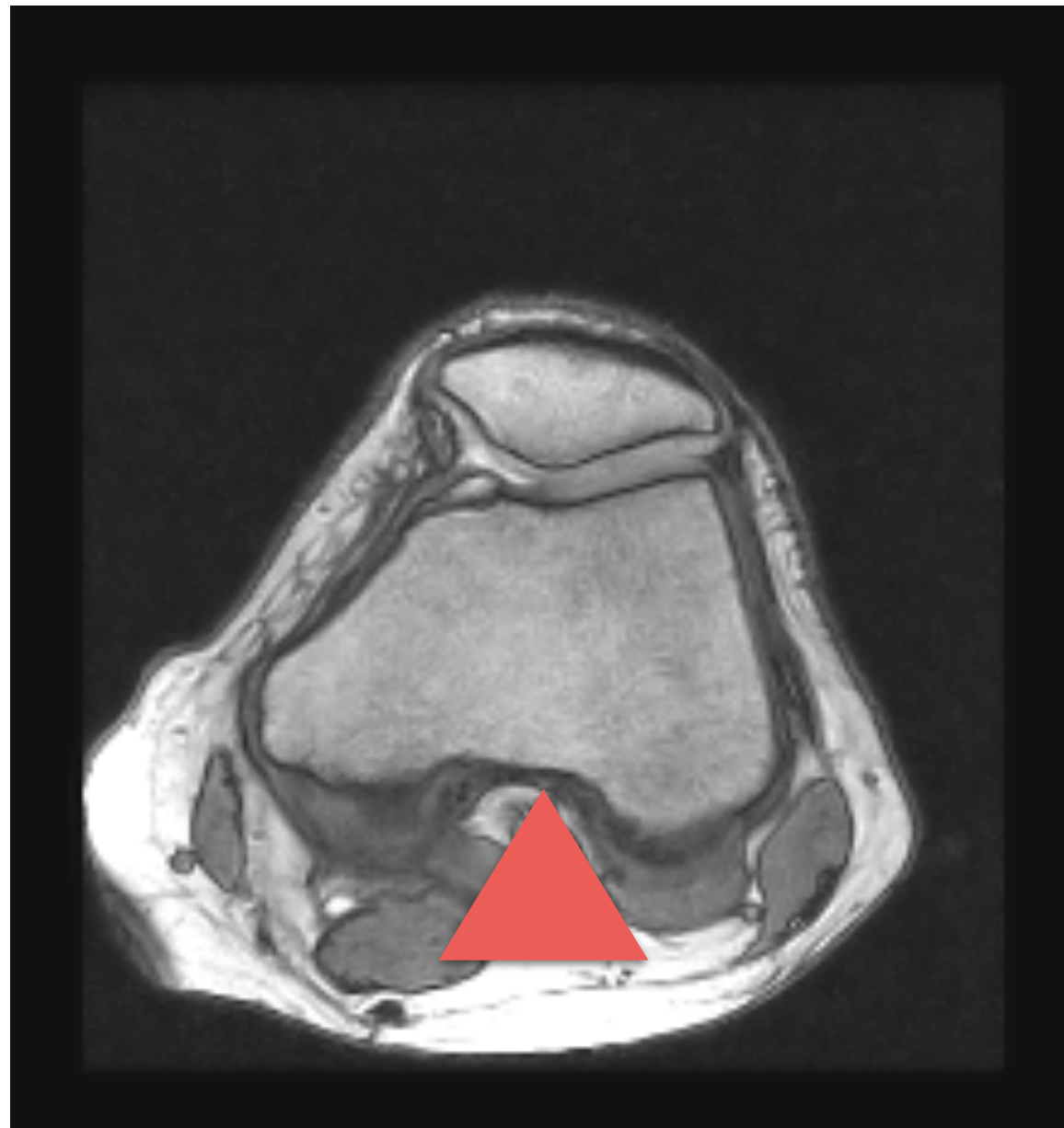
Snakes: Constraint Energy



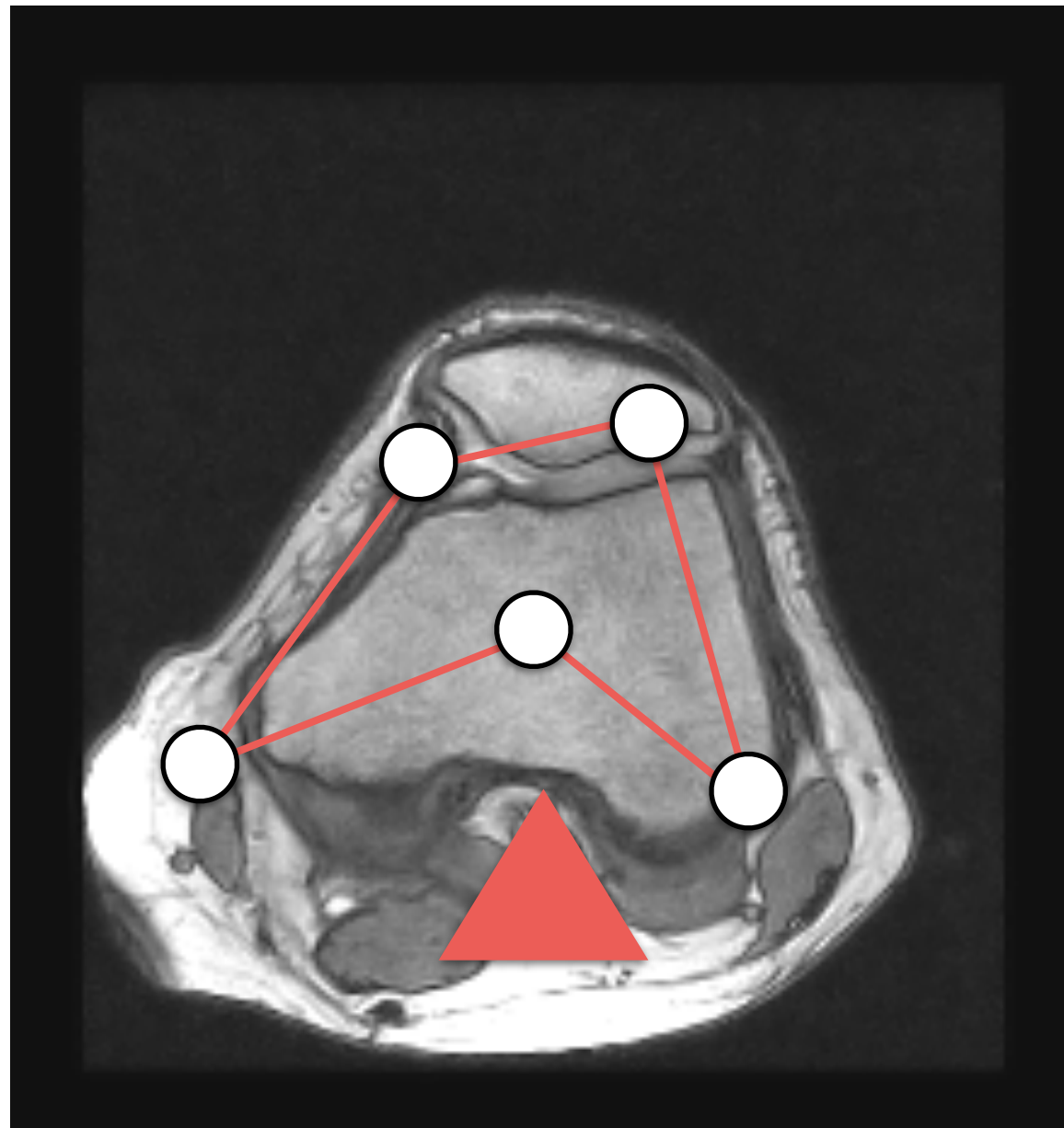
Snakes: Constraint Energy



Snakes: Constraint Energy



Snakes: Constraint Energy



How do we solve it?

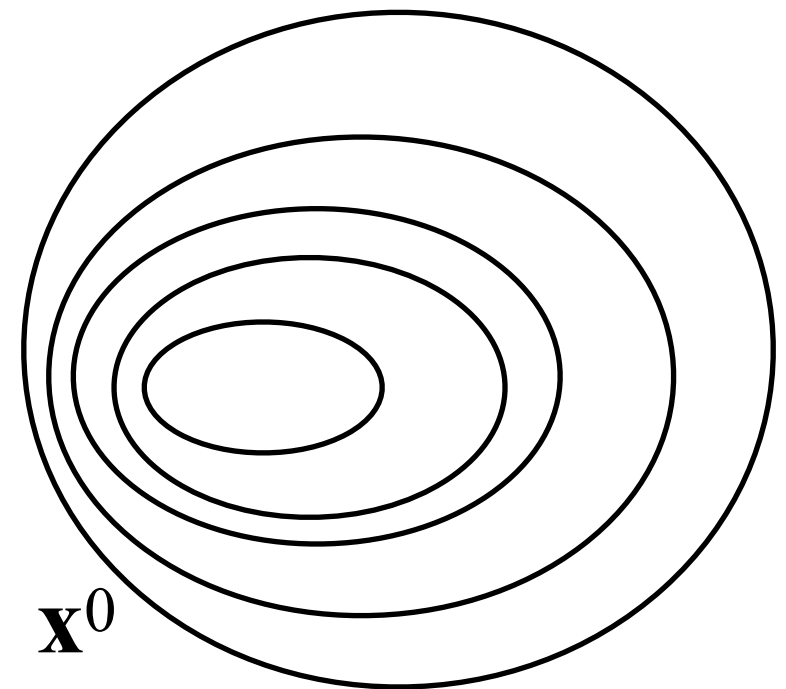
$$E_{\mathbf{v}} = E_{\text{internal}} + E_{\text{external}} + E_{\text{constraint}}$$

Gradient Descent

- It is a first-order iterative optimization method:

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

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

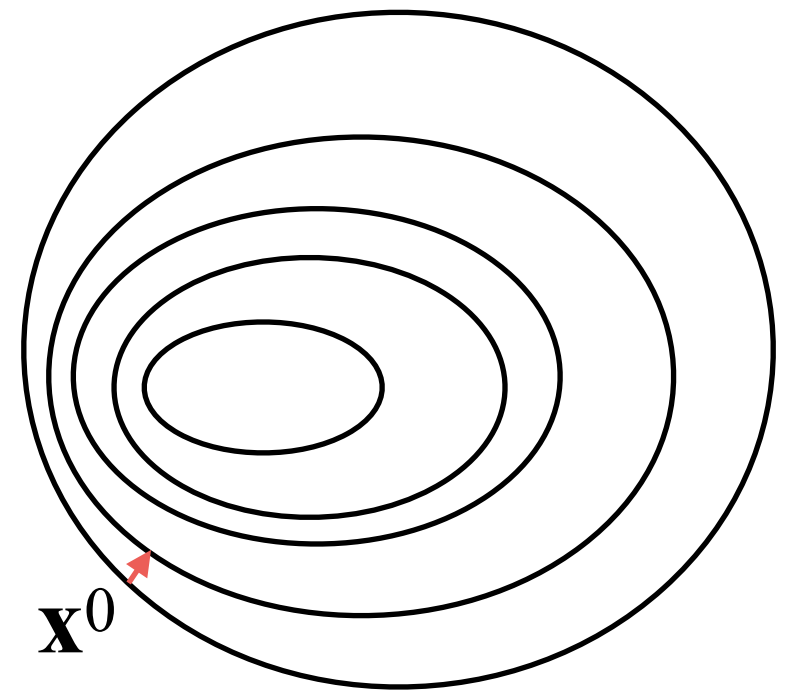


Gradient Descent

- It is a first-order iterative optimization method:

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

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

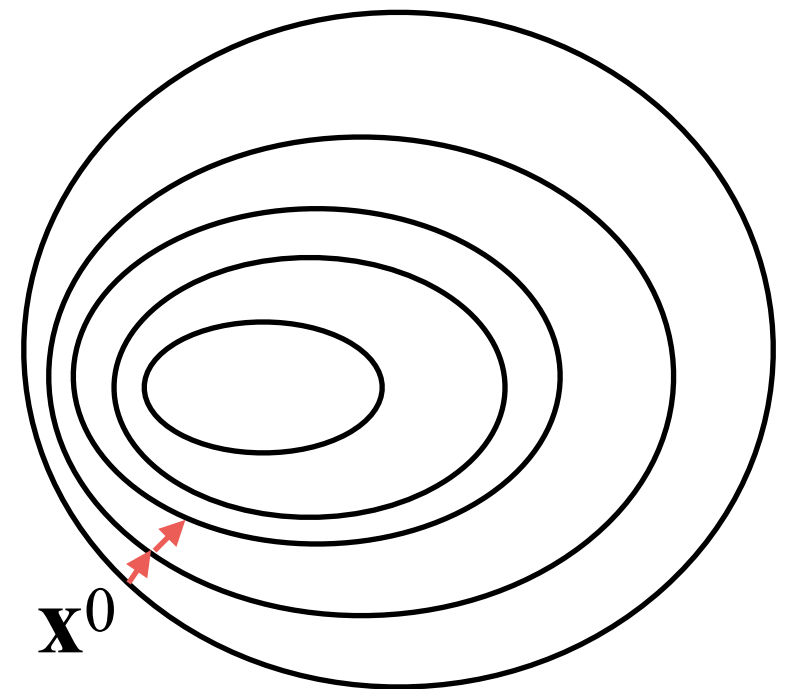


Gradient Descent

- It is a first-order iterative optimization method:

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

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

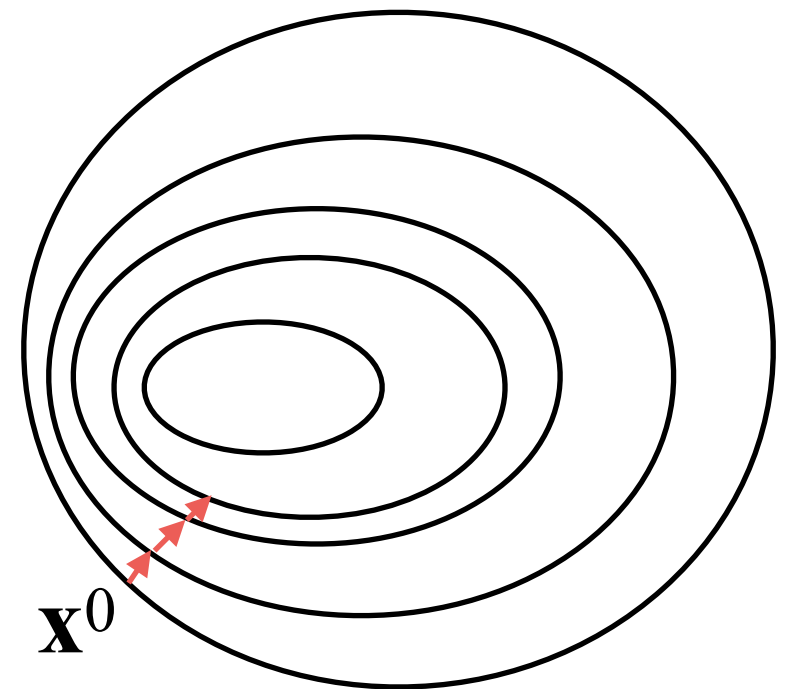


Gradient Descent

- It is a first-order iterative optimization method:

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

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

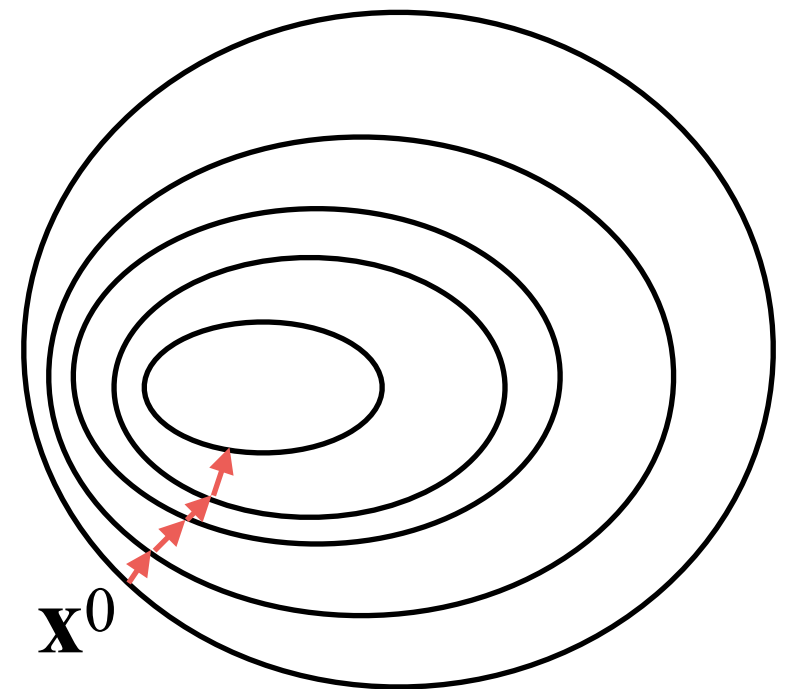


Gradient Descent

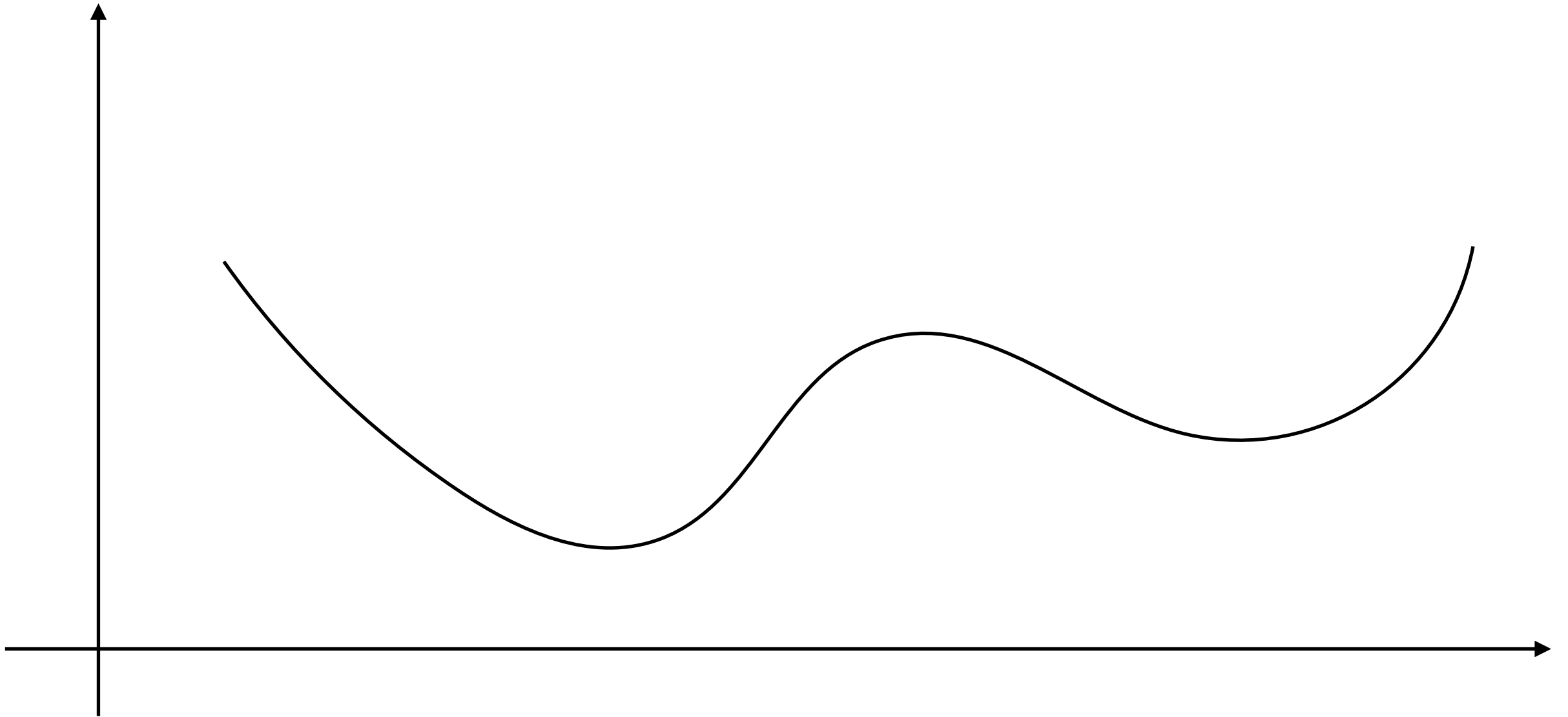
- It is a first-order iterative optimization method:

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

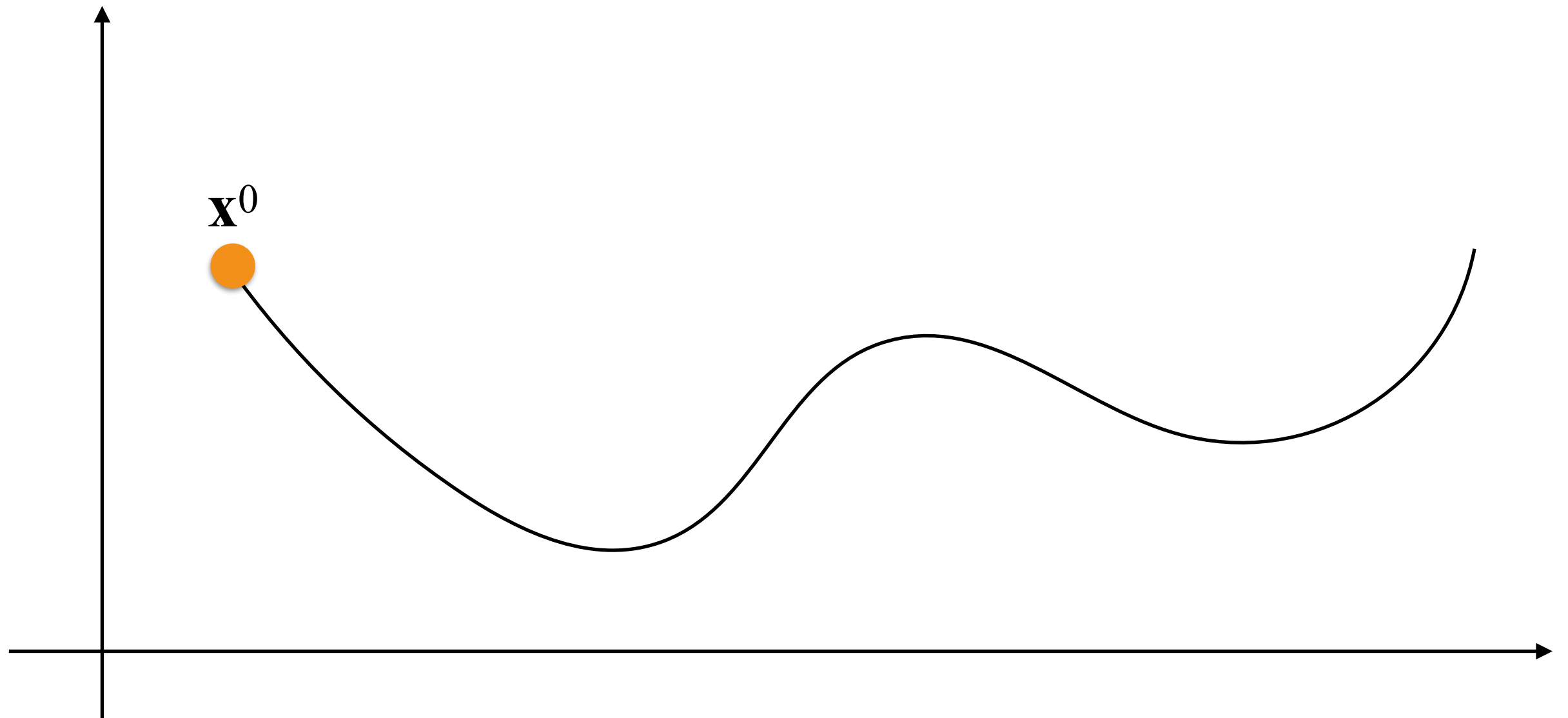
- We need to start with a g
- It will find a **local minimum**!
- f has to be differentiable.
- $\mathbf{x}^0$ is a “good” guess.



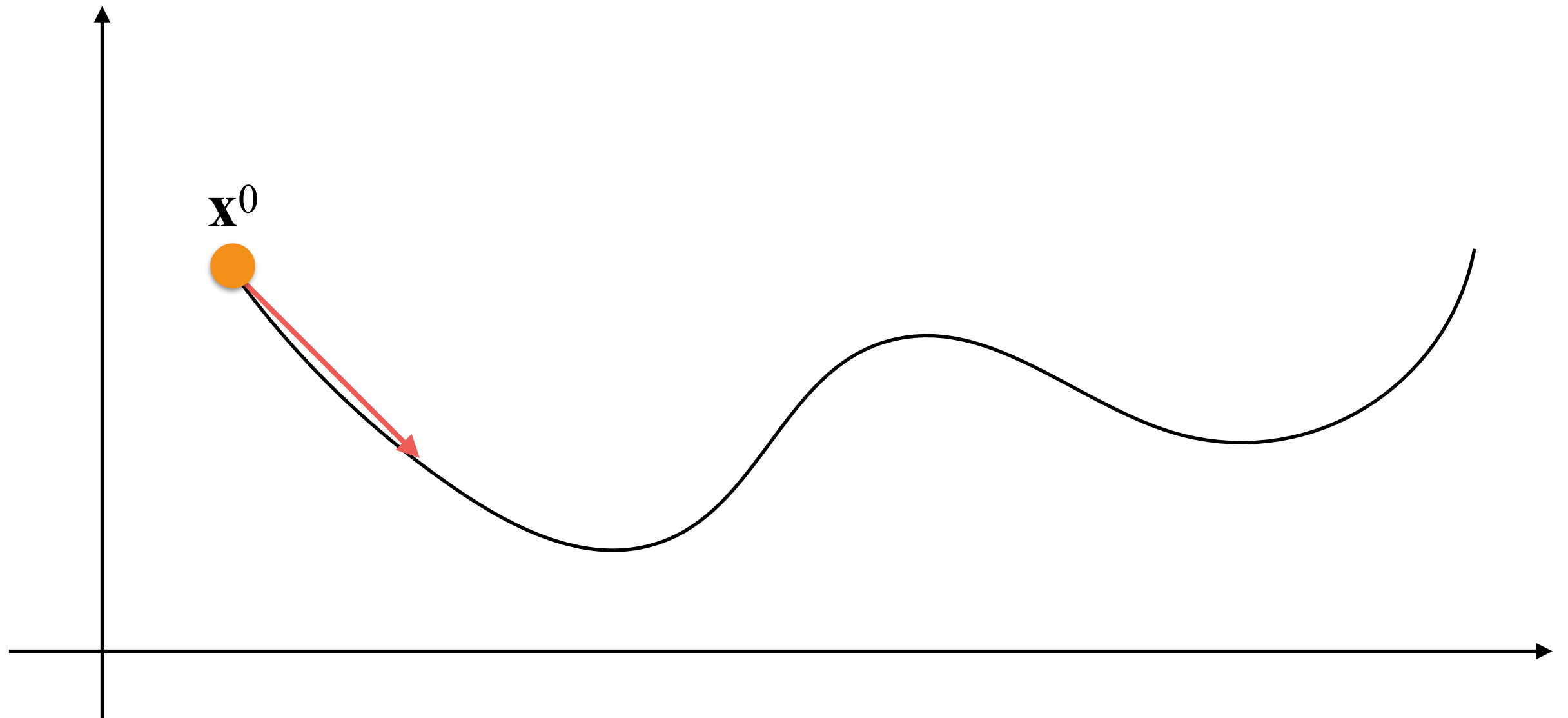
Gradient Descent



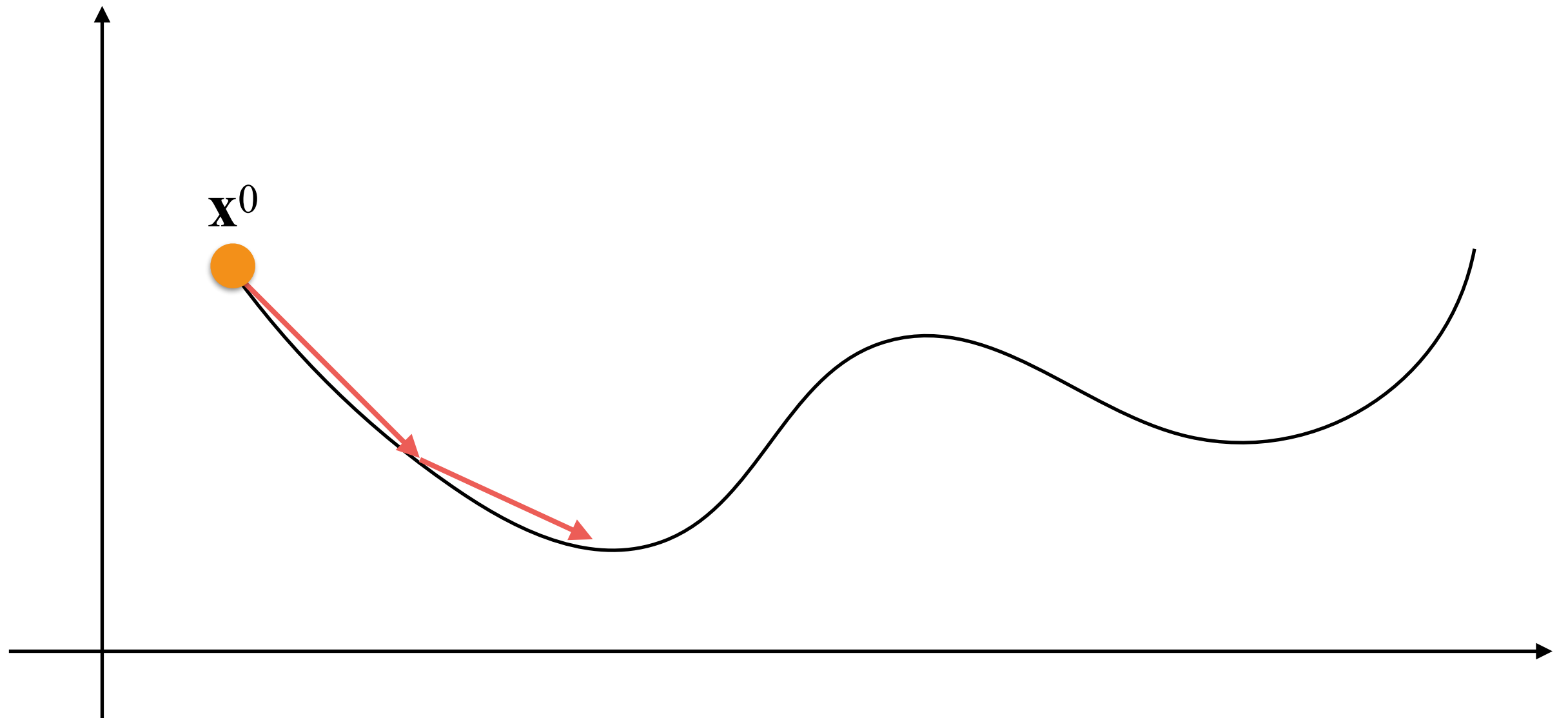
Gradient Descent



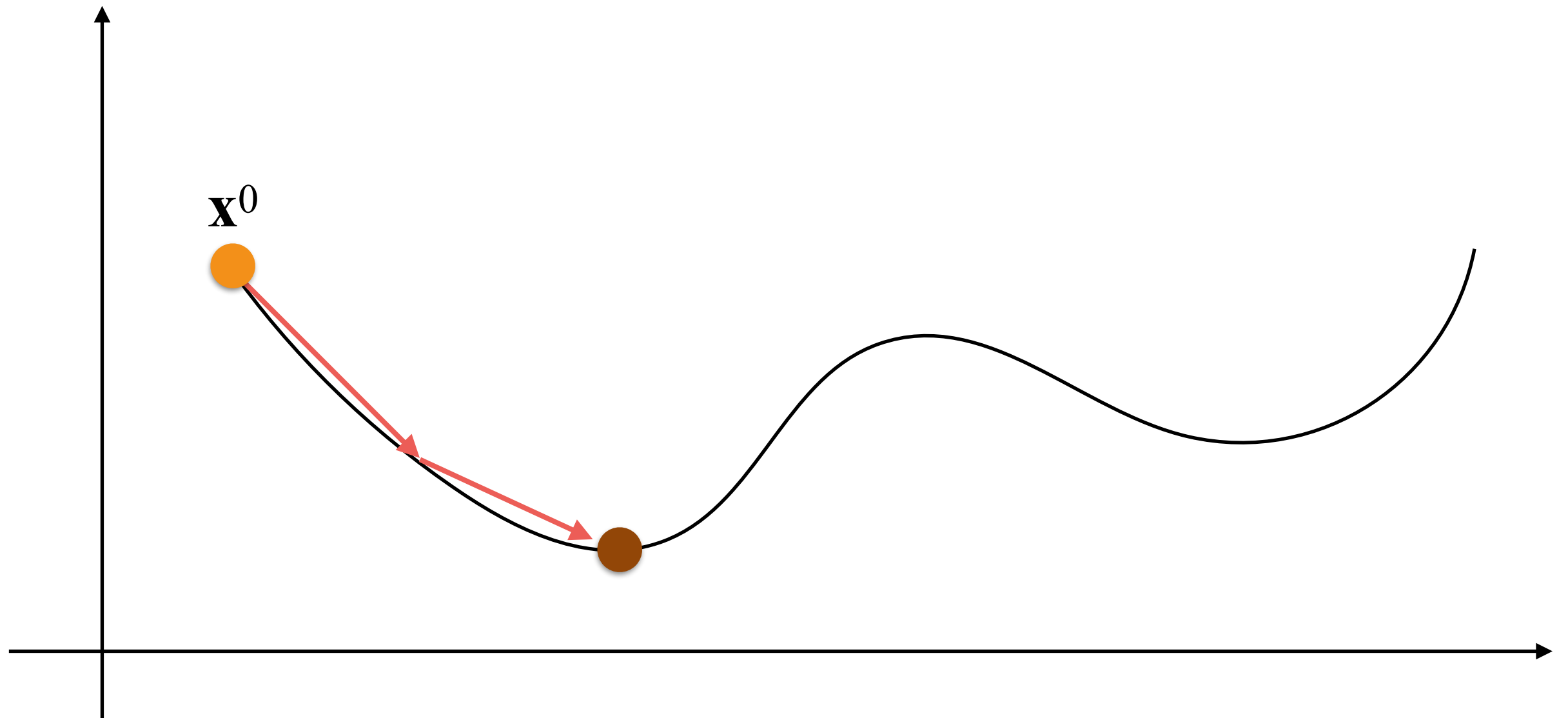
Gradient Descent



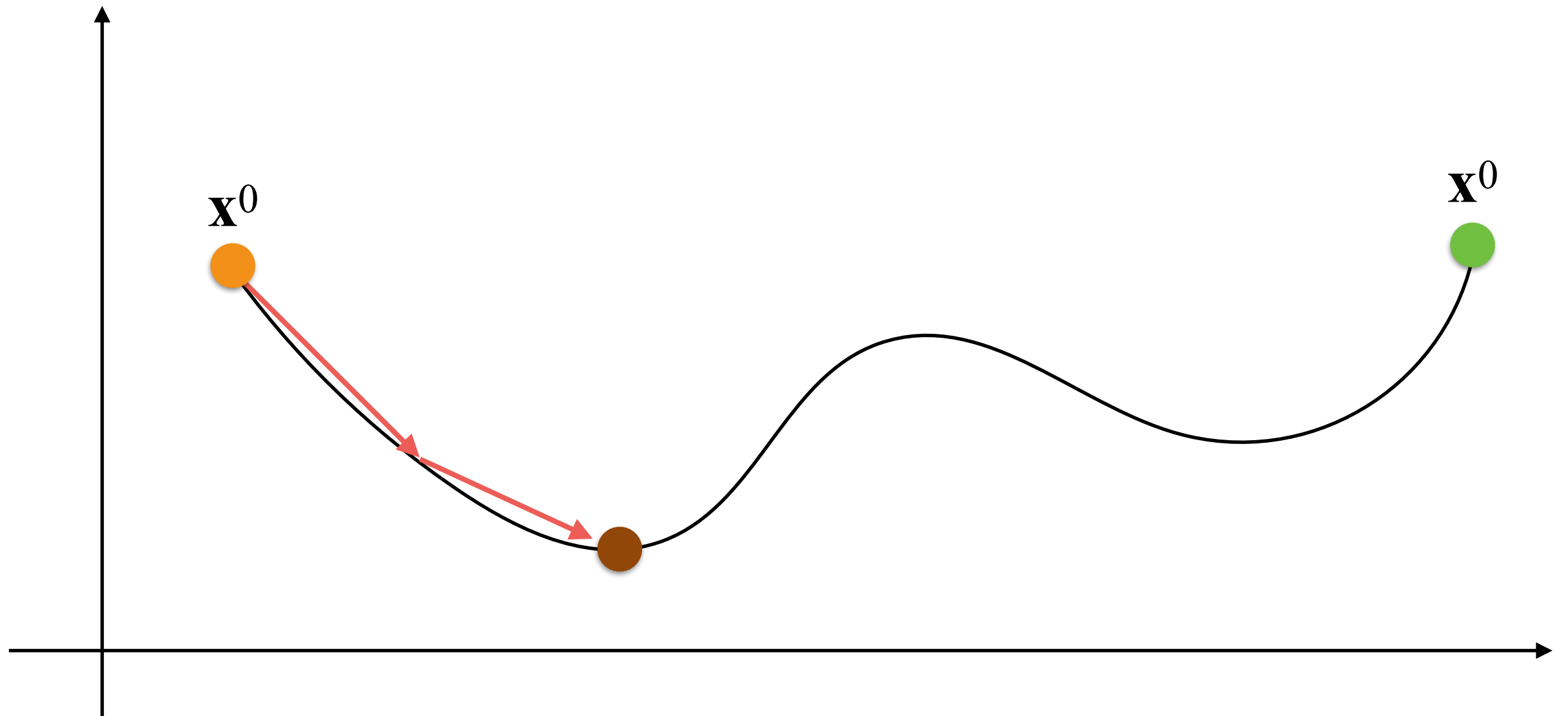
Gradient Descent



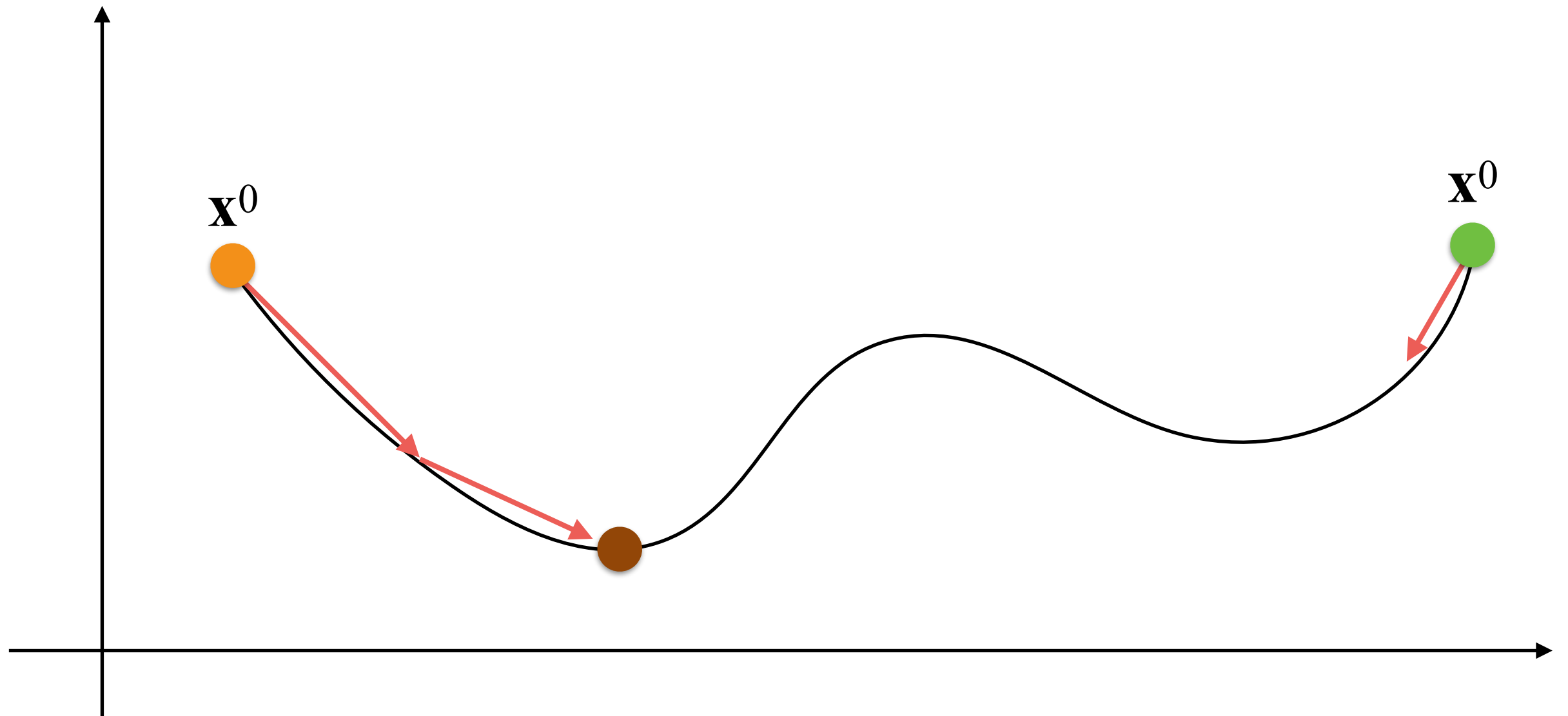
Gradient Descent



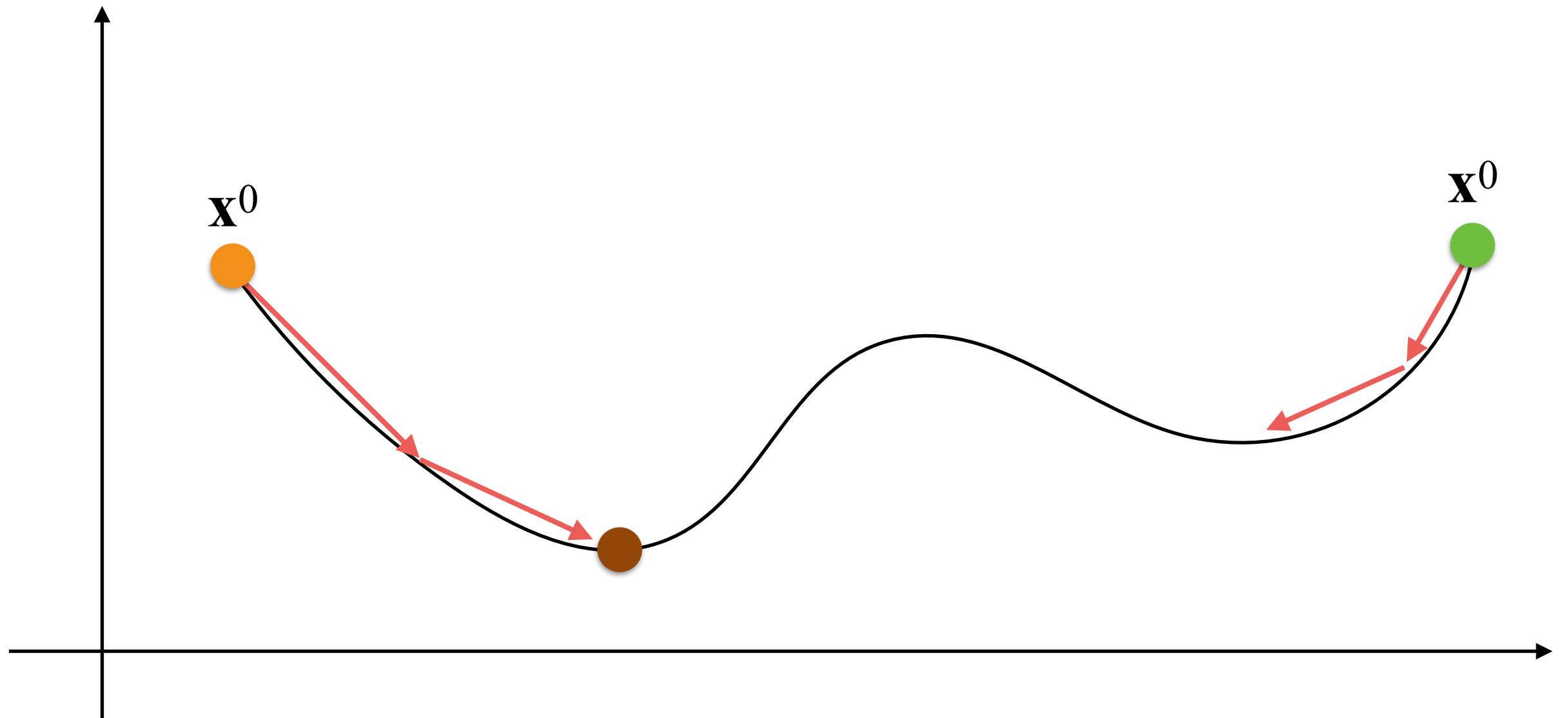
Gradient Descent



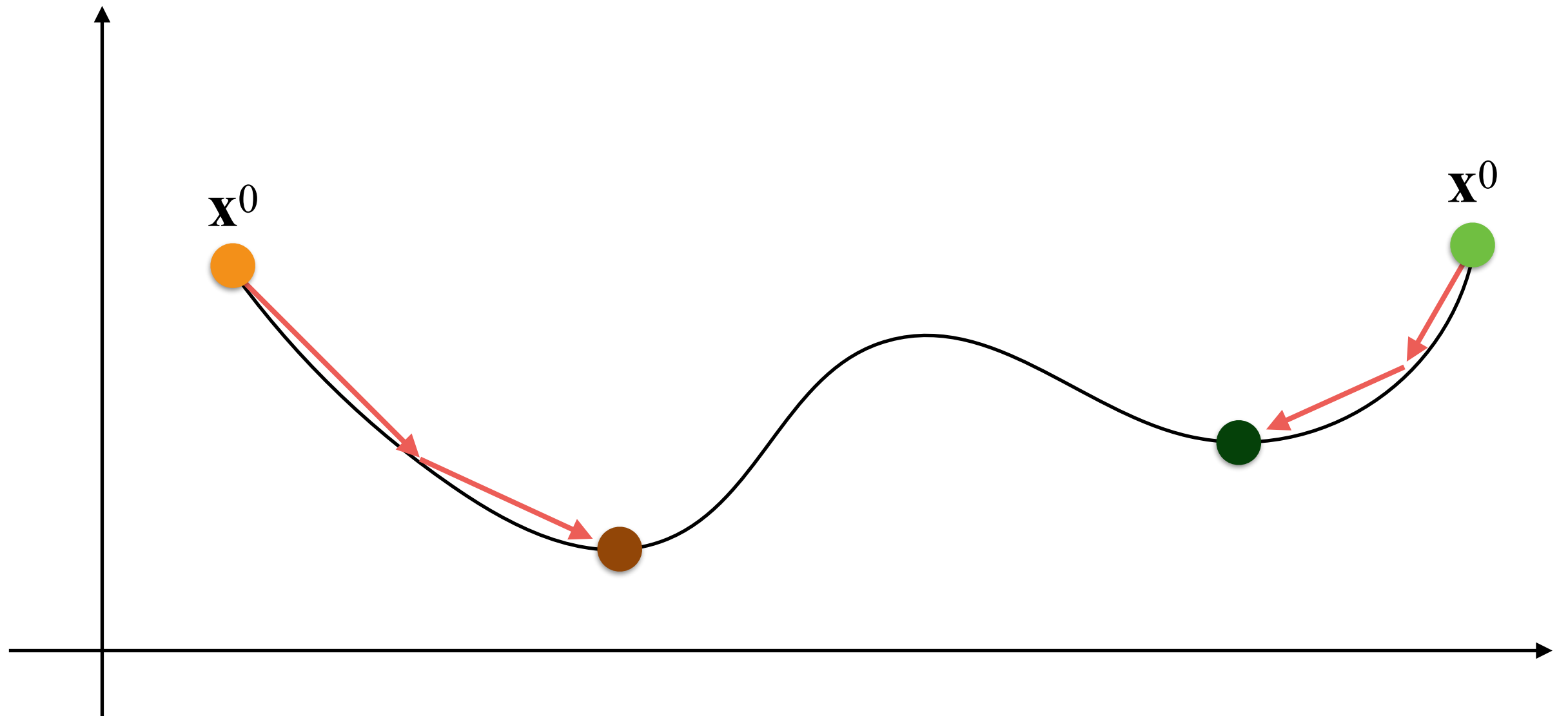
Gradient Descent



Gradient Descent



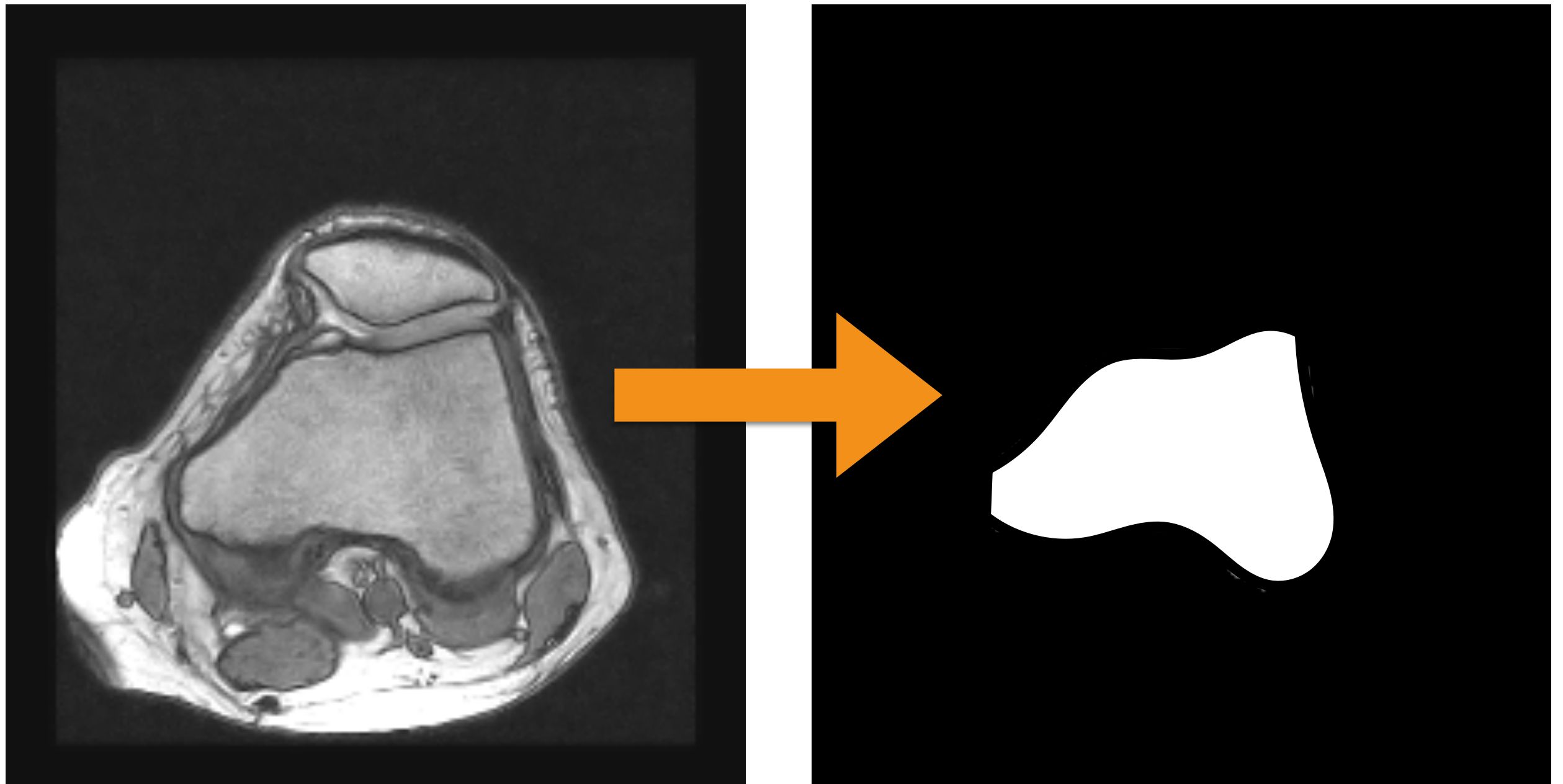
Gradient Descent



Snakes: Gradient Descent

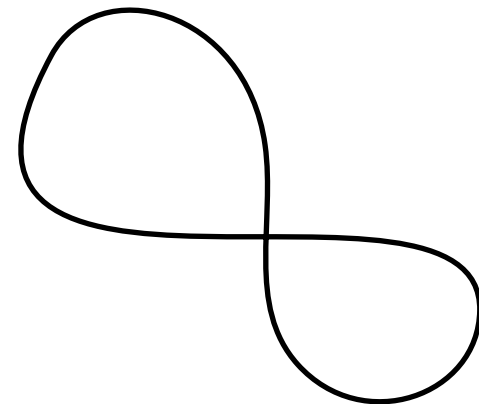
- What is our $\mathbf{x}^0$ in the snake minimization?
- We need to click a few points in the image around our object of interest!

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!

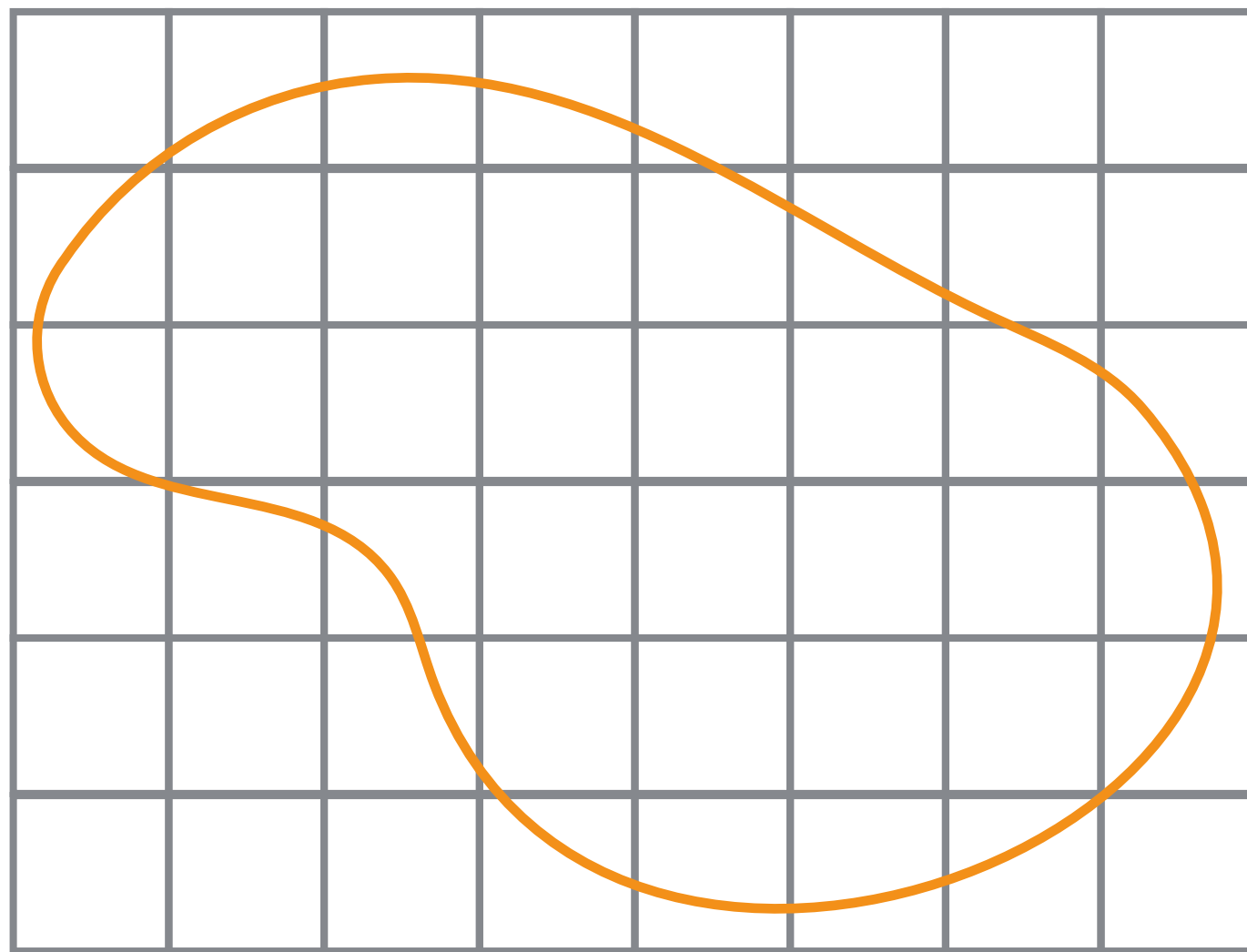


Region Growing

Region Growing

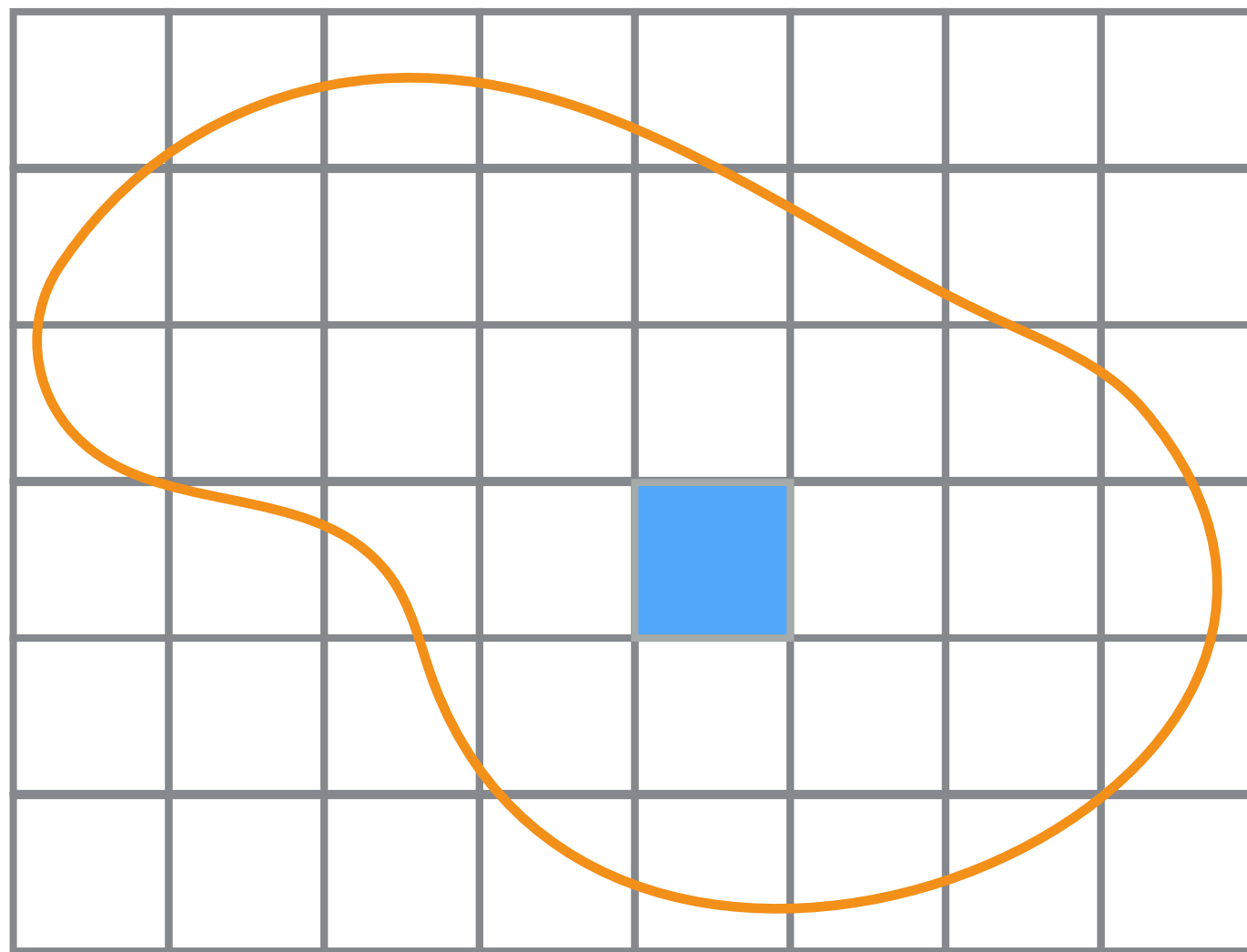
- This algorithm expands a painted initial mask until it reaches strong edges
- Therefore, we need to compute edges first!

Region Growing



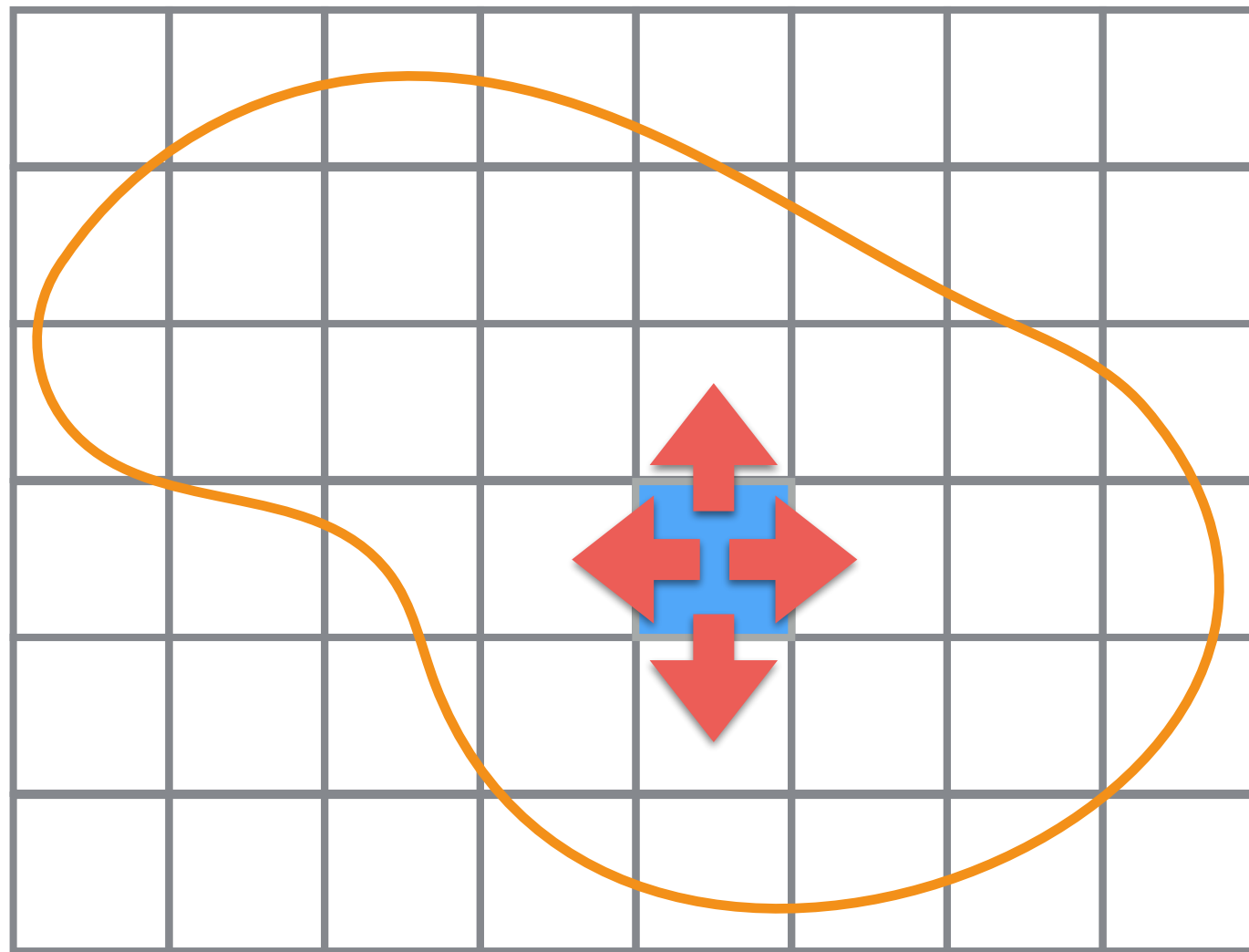
— Image
Edge

Region Growing

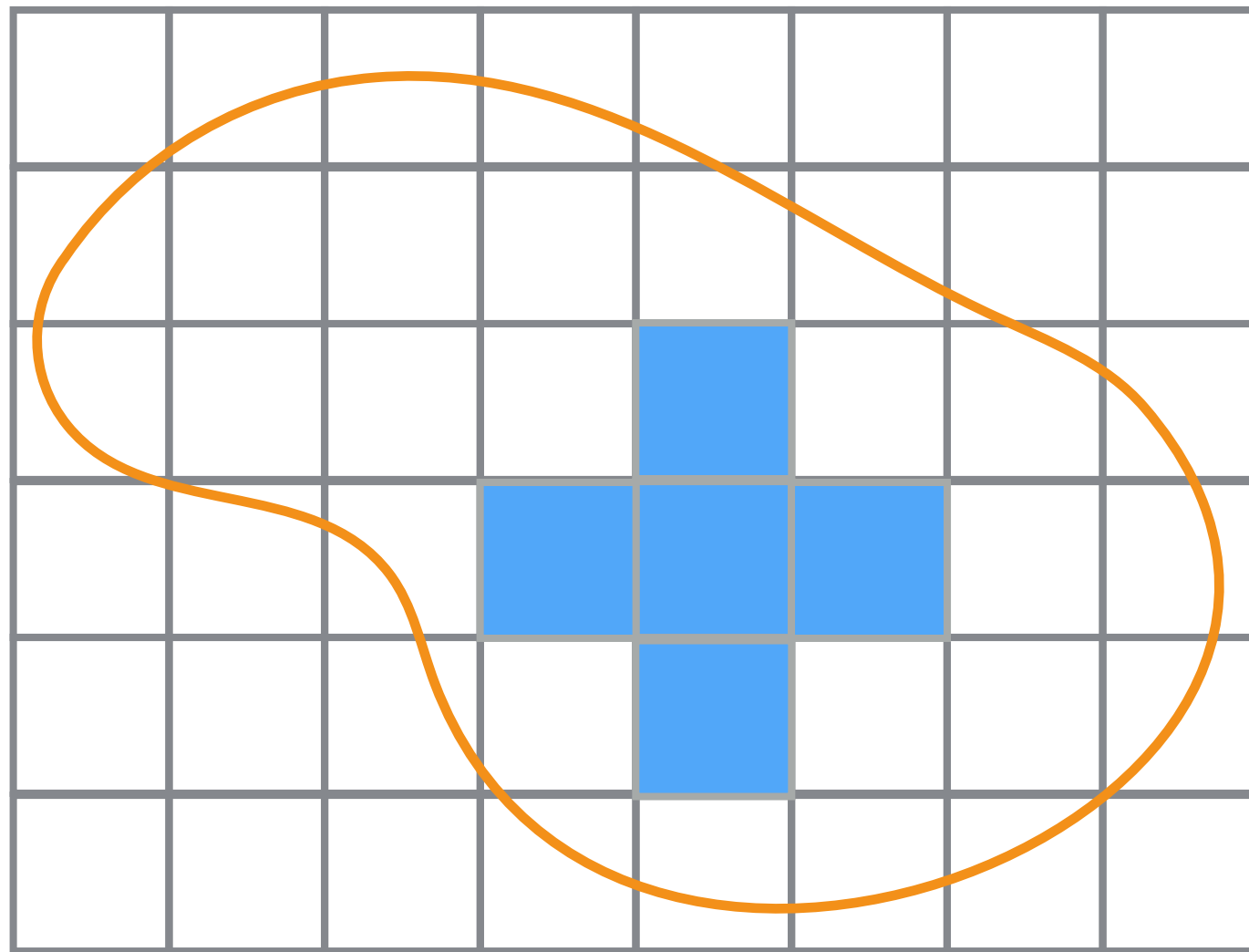


Seed

Region Growing

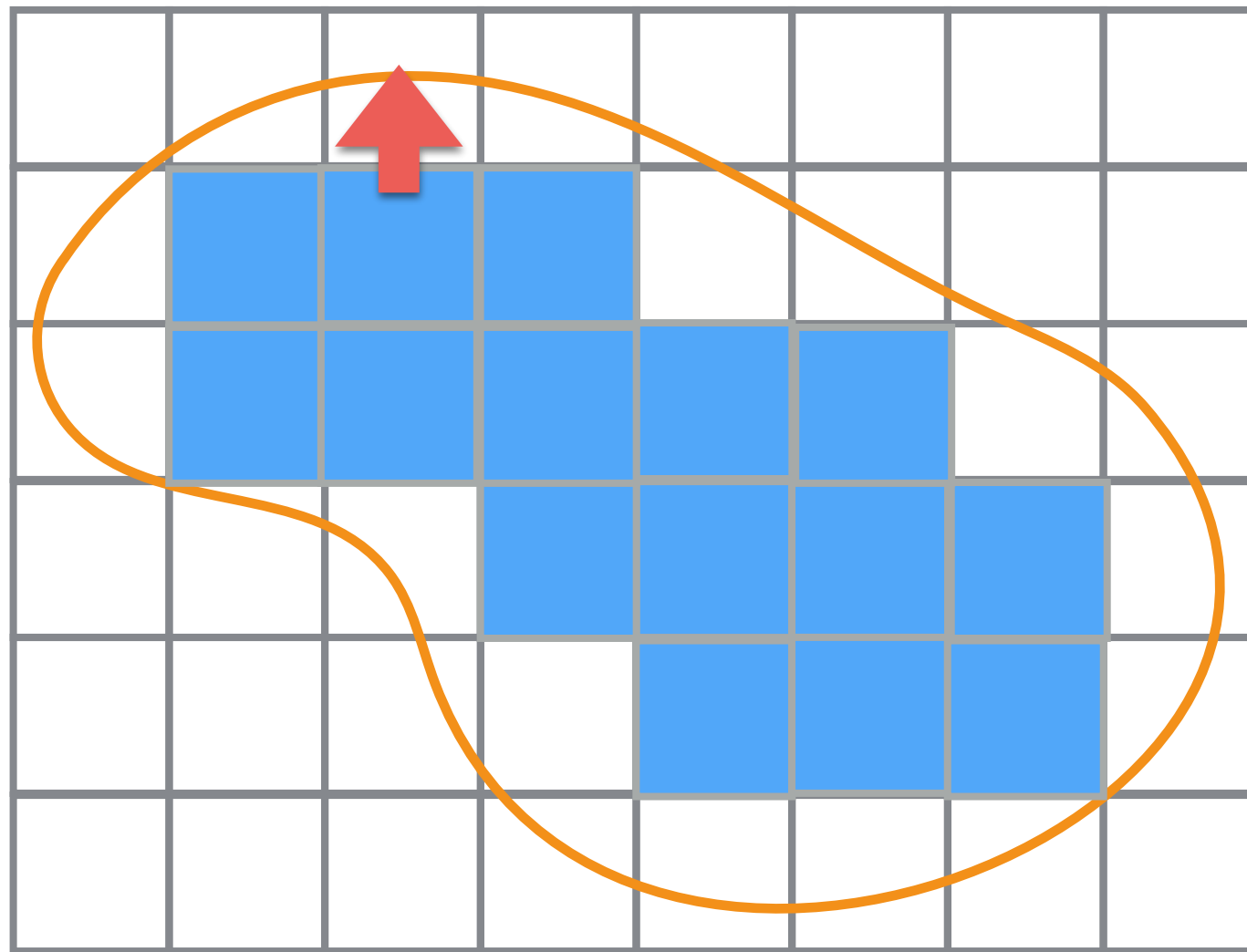


Region Growing

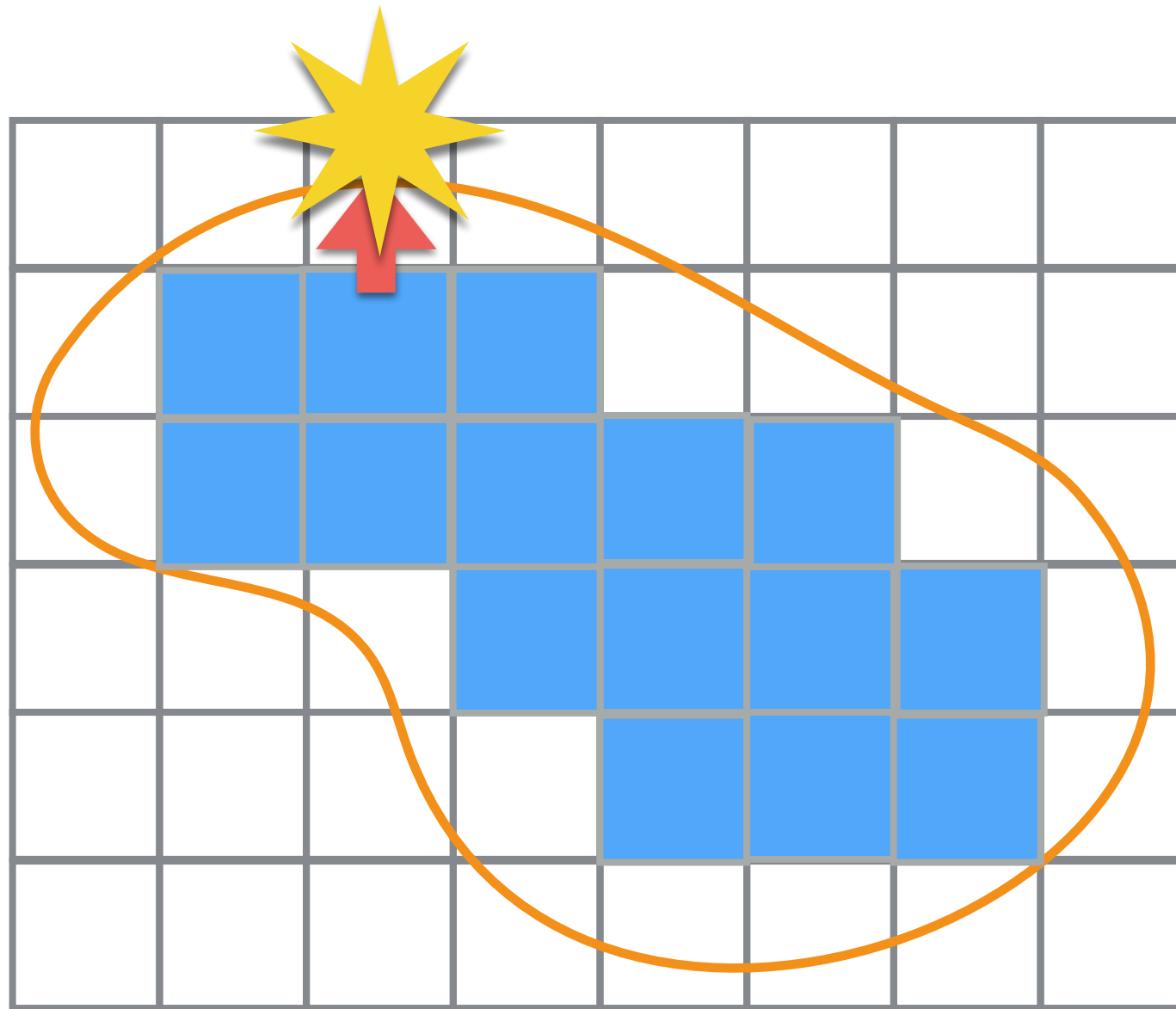


after a while...

Region Growing



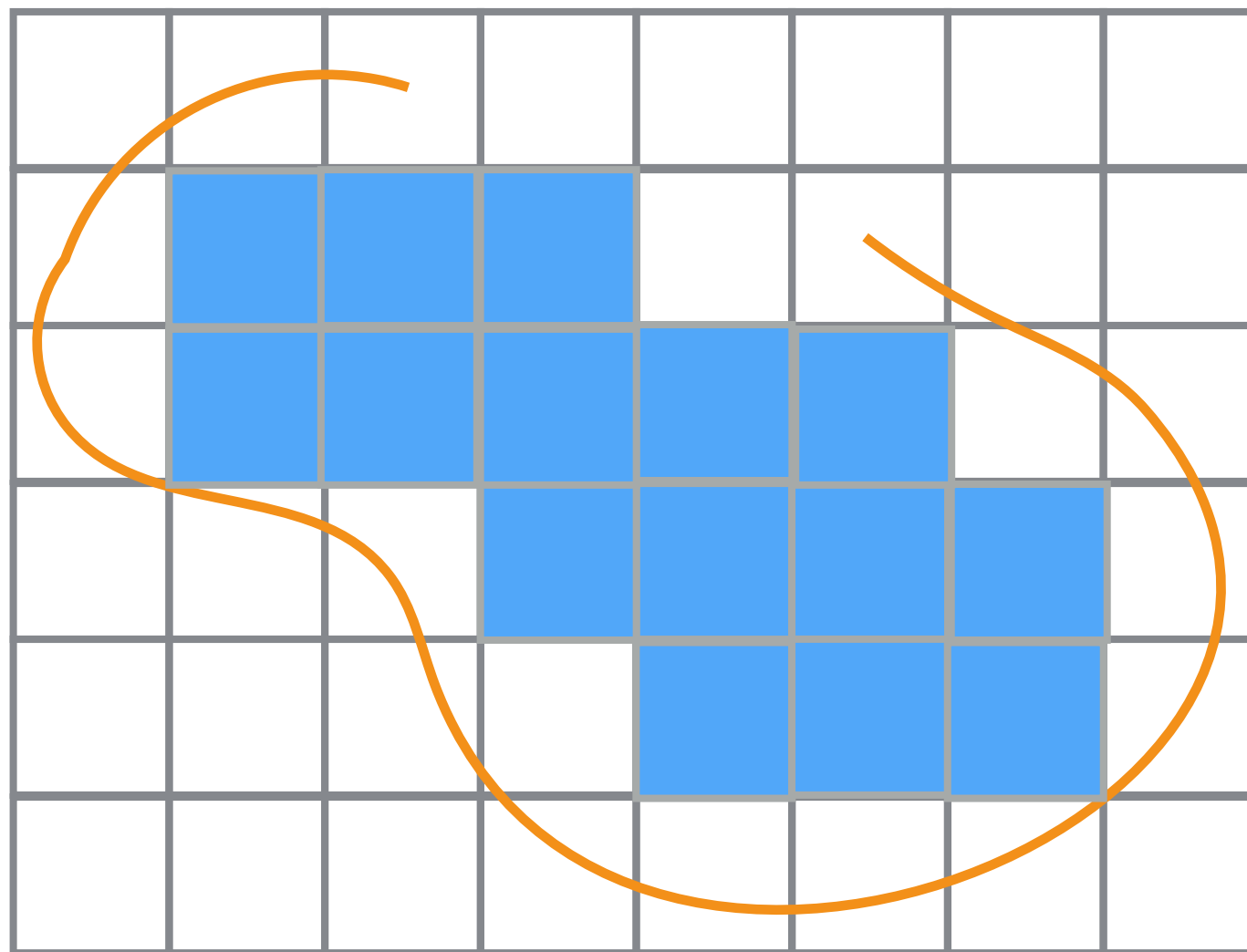
Region Growing



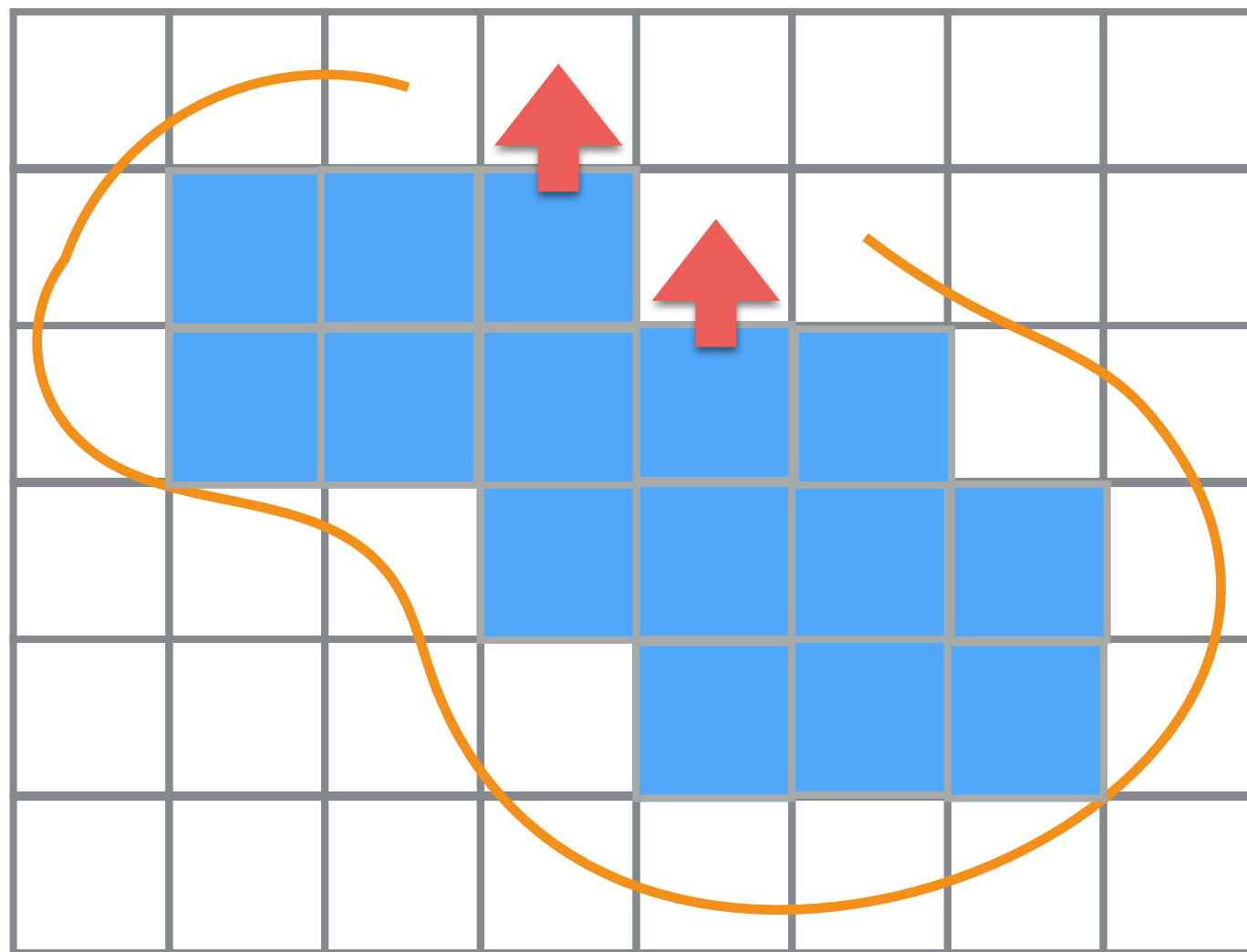
Region Growing

- 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.

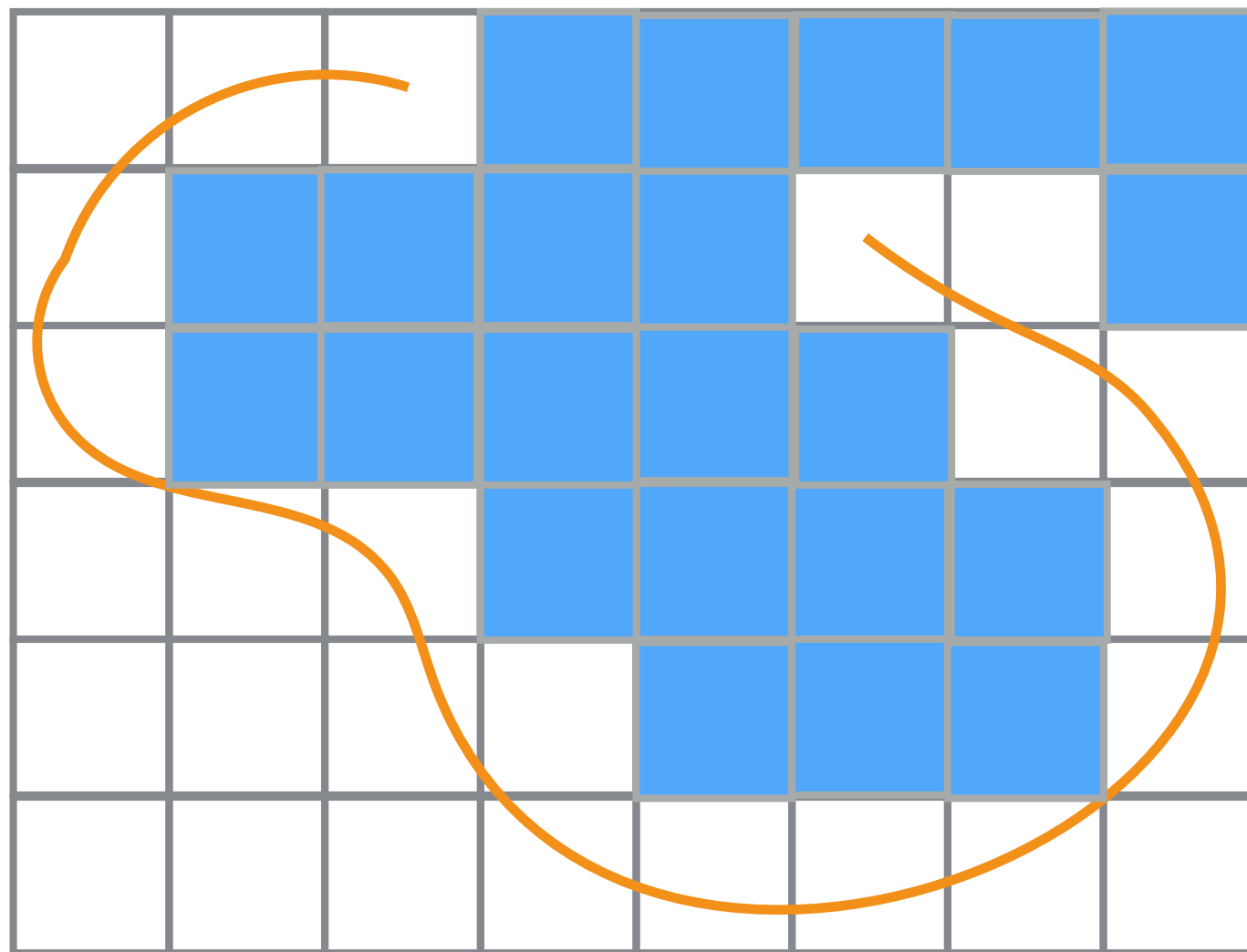
Region Growing: Epic Fail



Region Growing: Epic Fail



Region Growing: Epic Fail



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.

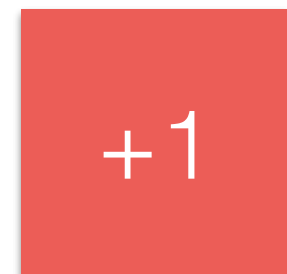
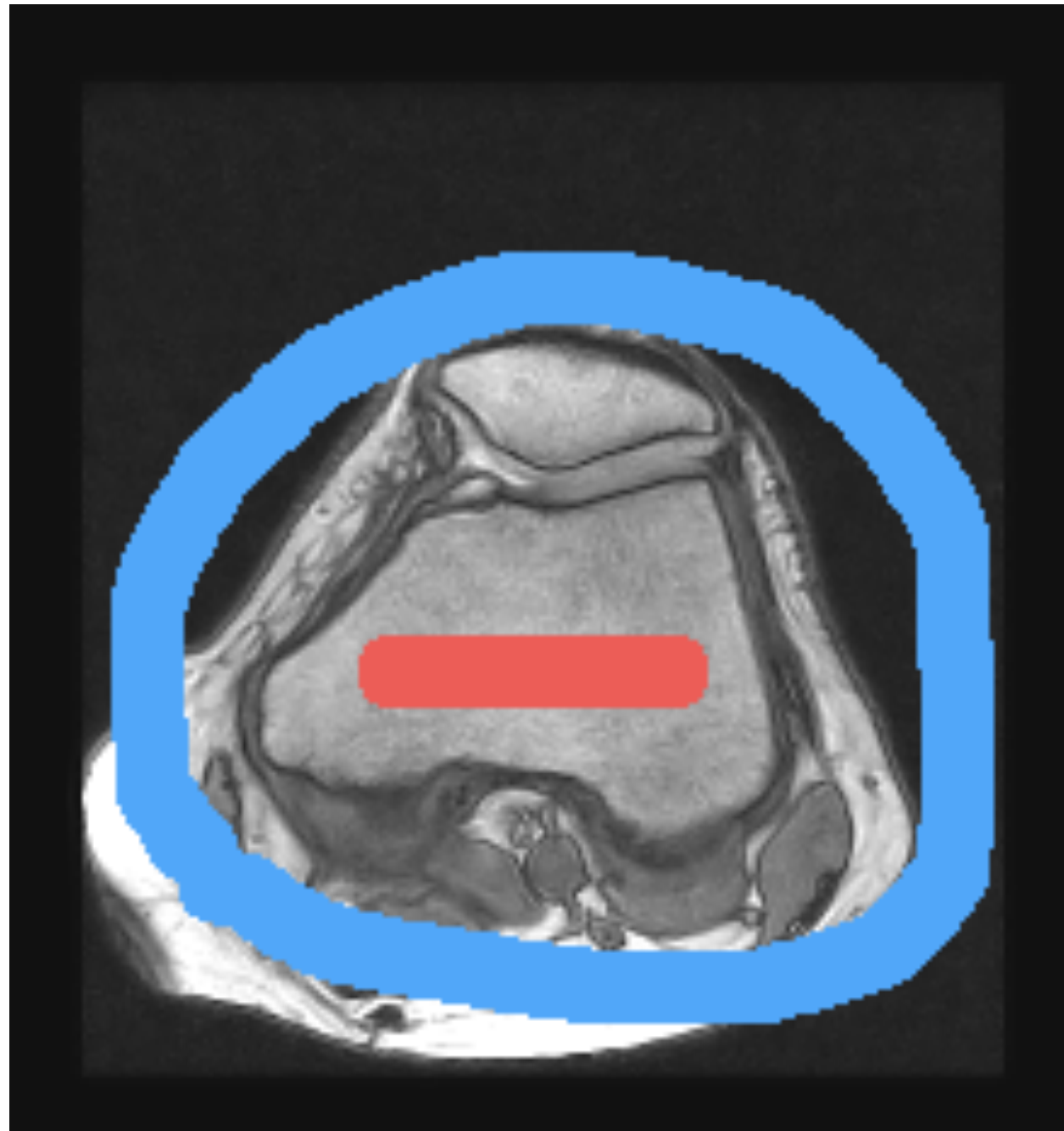
Stroke-Based



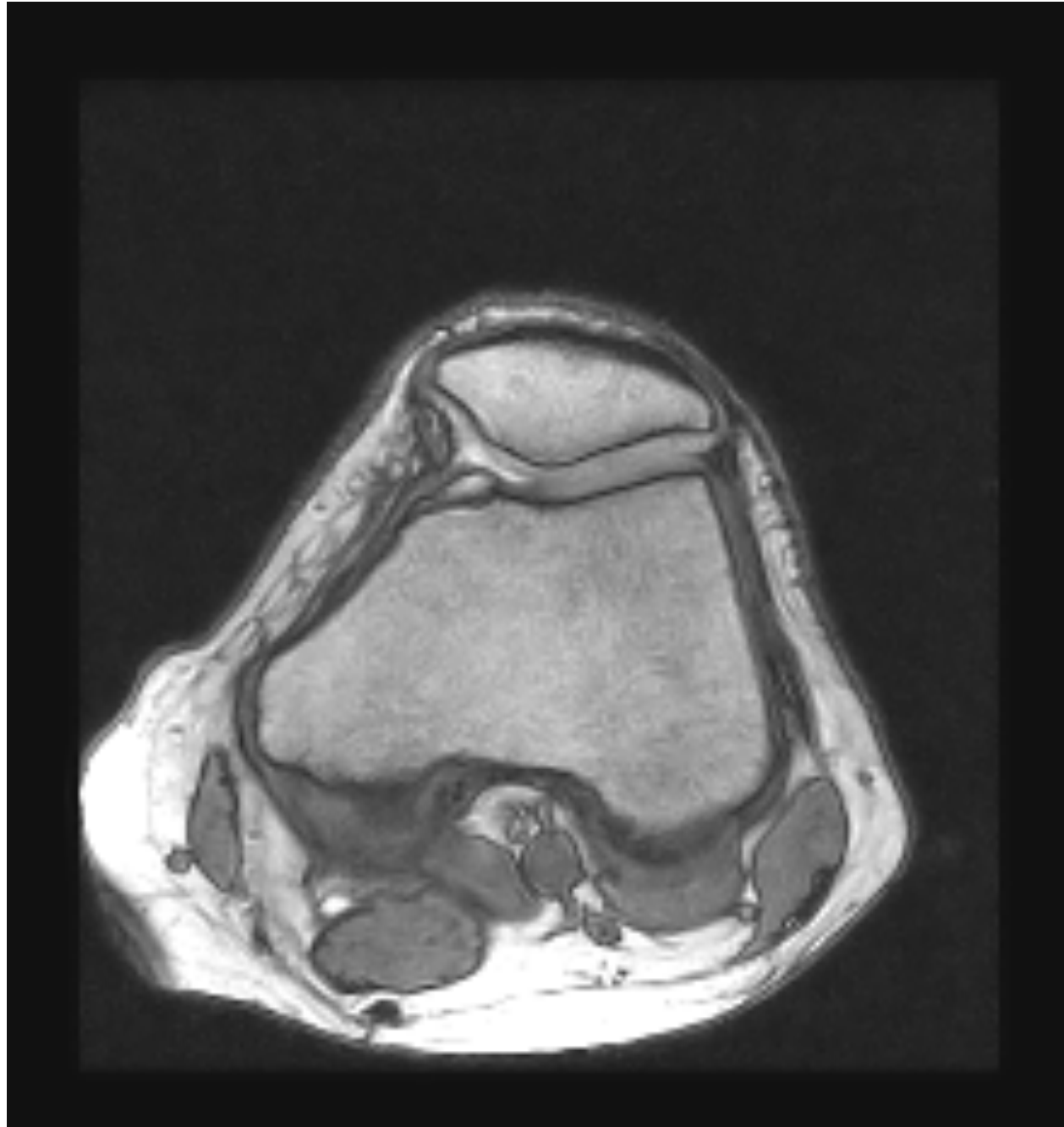
+1

-1

Stroke-Based



Stroke-Based



I



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):

$$l_i = 0; \quad \theta_i = 0; \quad C_i = I(x_i, y_i) \quad \forall i \quad s(x_i, y_i) = 0$$

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

$$l_i = s(x_i, y_i); \quad \theta_i = 1; \quad C_i = I(x_i, y_i) \quad \forall i \quad s(x_i, y_i) \neq 0$$

Stroke-Based: Grow-Cut

- For each pixel, we have:

$$\langle \overset{\text{Strength}}{\underset{\text{Label Intensity}}{l_i; \theta_i; C_i}} \rangle$$

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

$$l_i = 0; \quad \theta_i = 0; \quad C_i = I(x_i, y_i) \quad \forall i \quad s(x_i, y_i) = 0$$

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

$$l_i = s(x_i, y_i); \quad \theta_i = 1; \quad C_i = I(x_i, y_i) \quad \forall i \quad s(x_i, y_i) \neq 0$$

Stroke-Based: Grow-Cut

- For each pixel i :
 - We copy the previous status:

$$\langle l_i^{t+1}, \theta_i^{t+1}; I_i^{t+1} \rangle = \langle l_i^t, \theta_i^t; I_i^t \rangle$$

- For each neighbors j of i :
 - if $g(\|C_i^t - C_j^t\|_2)\theta_j^t > \theta_i^t$ then

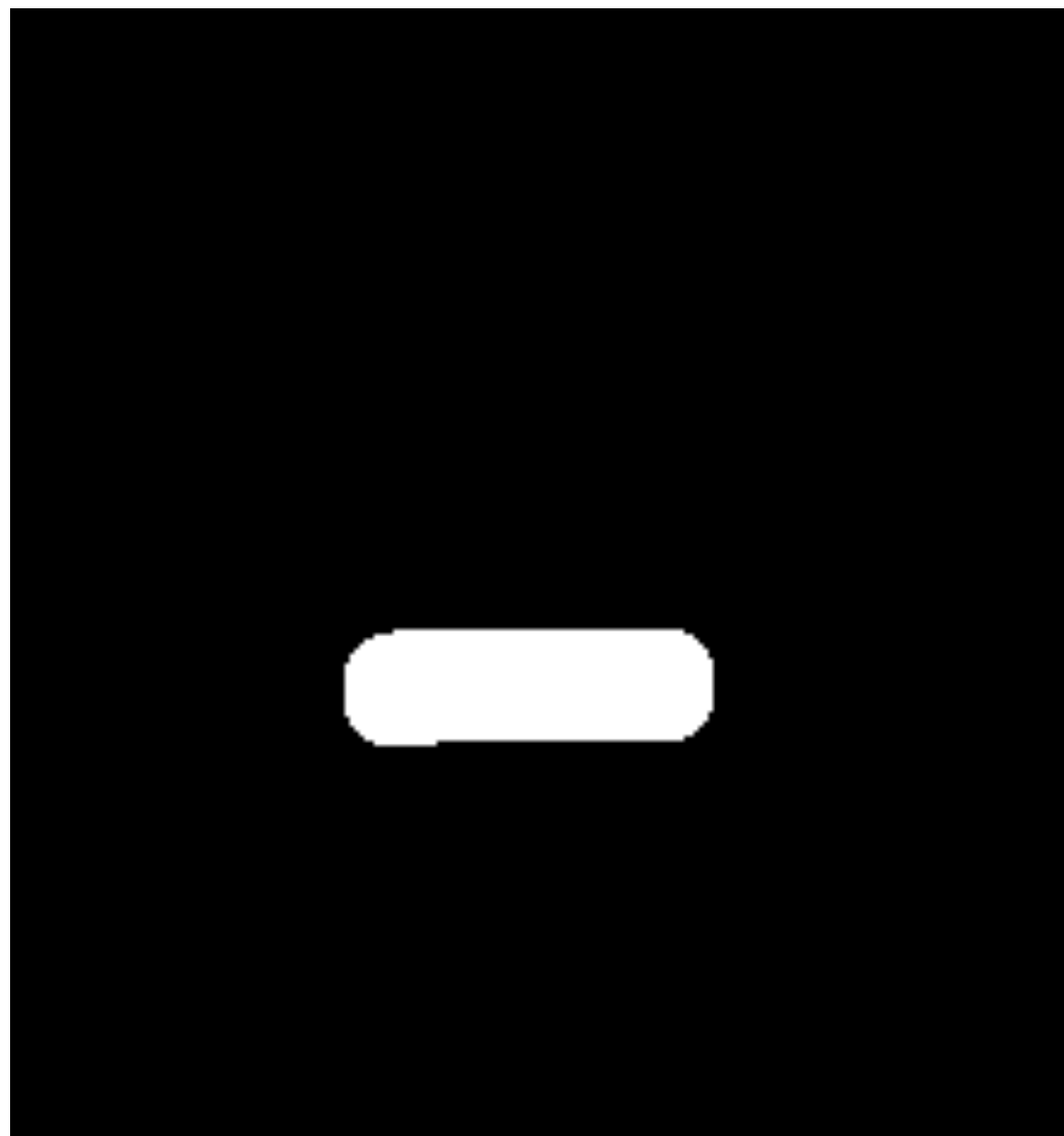
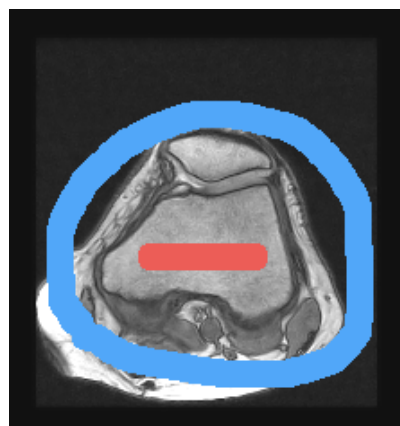
$$l_i^{t+1} = l_j^t$$

$$\theta_i^{t+1} = g(\|C_i^t - C_j^t\|_2) \cdot \theta_j^t$$

Stroke-Based: Grow-Cut

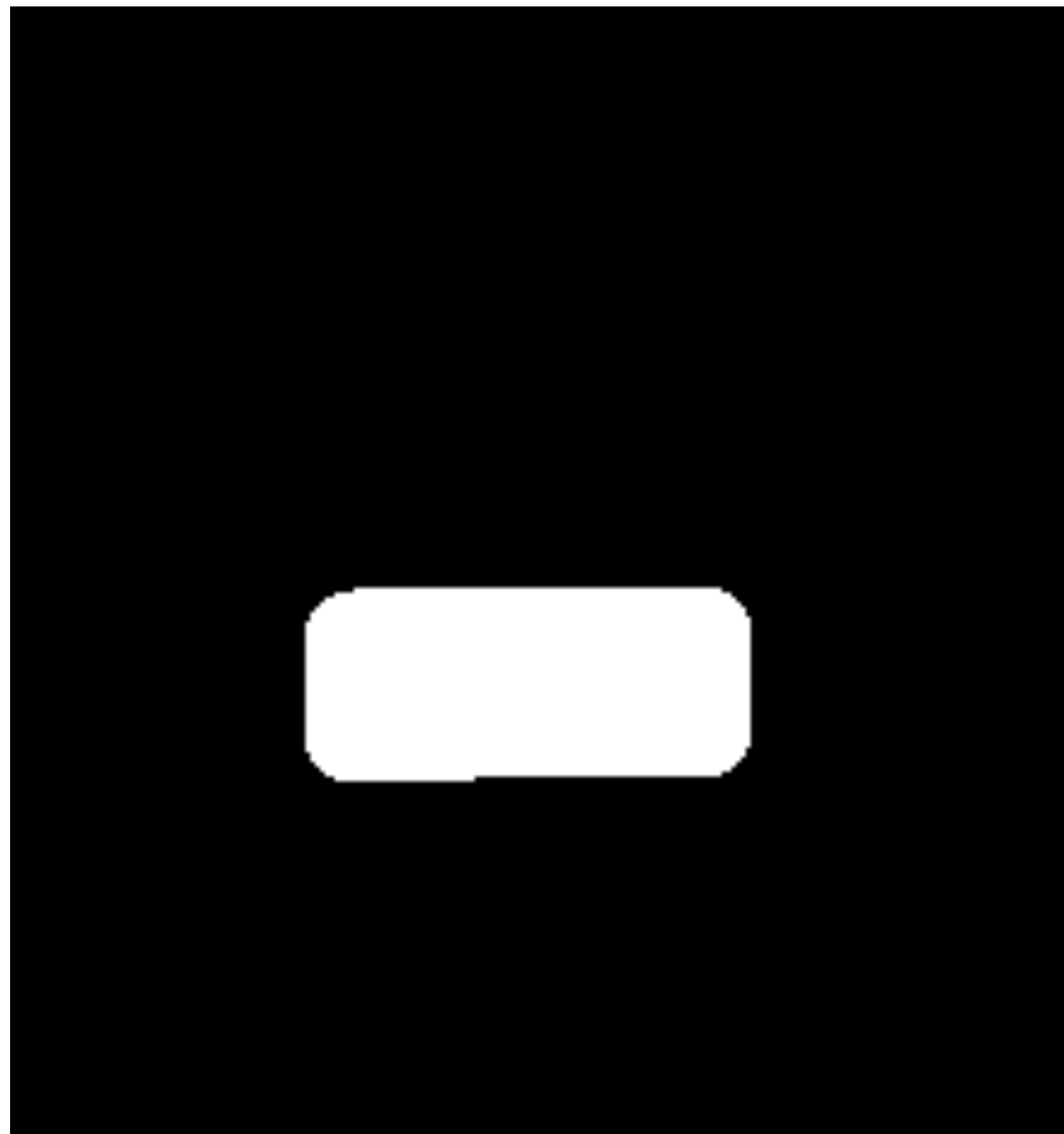
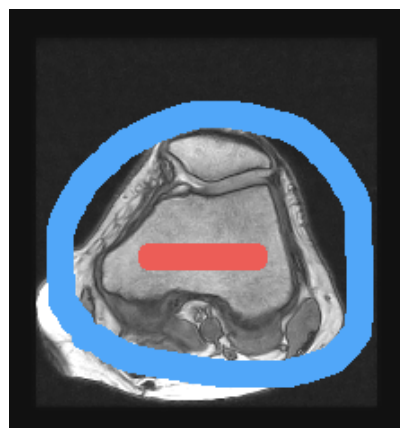
- Stopping criteria:
 - This process is iterated until either convergence; i.e., no changes in state
 - labels have been propagated enough (e.g., number of pixels of the diagonal).

Stroke-Based: Grow-Cut Example



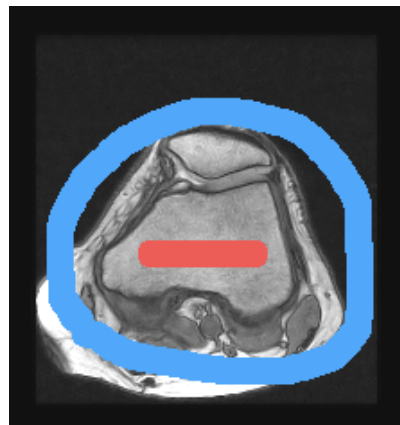
Iteration = 1

Stroke-Based: Grow-Cut Example



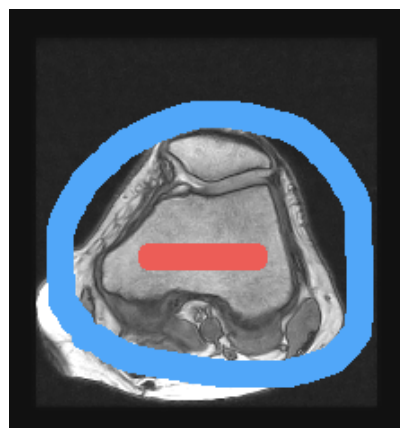
Iteration = 10

Stroke-Based: Grow-Cut Example



Iteration = 40

Stroke-Based: Grow-Cut Example



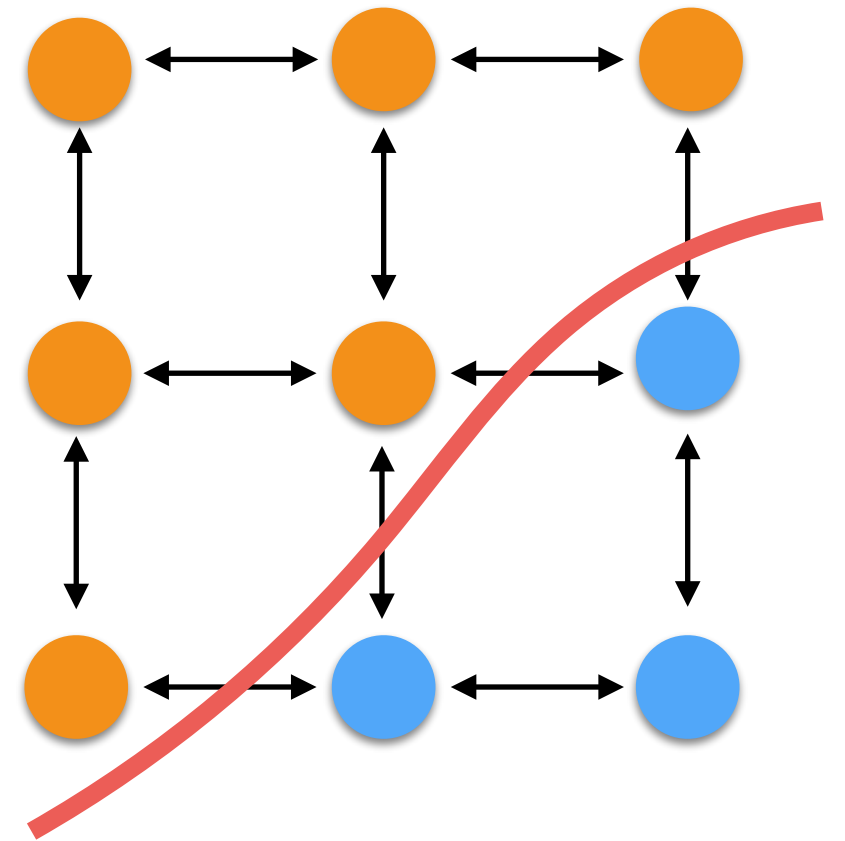
Iteration = 321

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!

Stroke-based

- **Graph-cut:** assuming the image as a 4-connected graph, we look for the minimum cut in the graph.



Machine Learning

Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!

Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!



Training
Set

Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!



Training
Set



Model

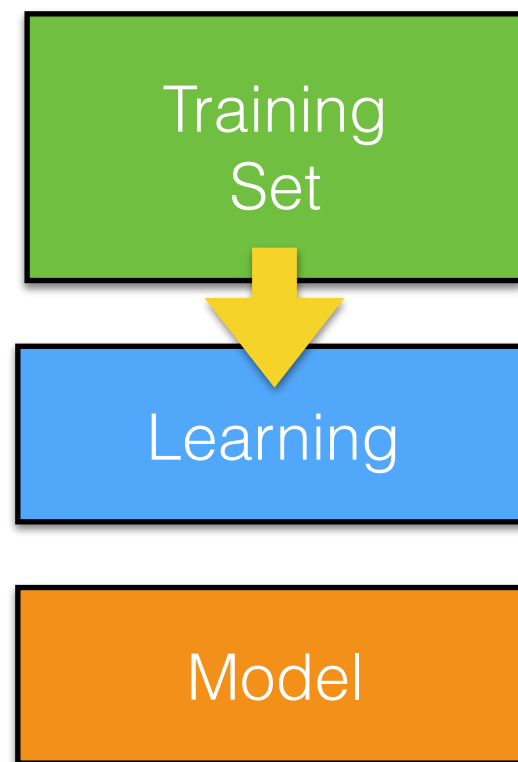
Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!



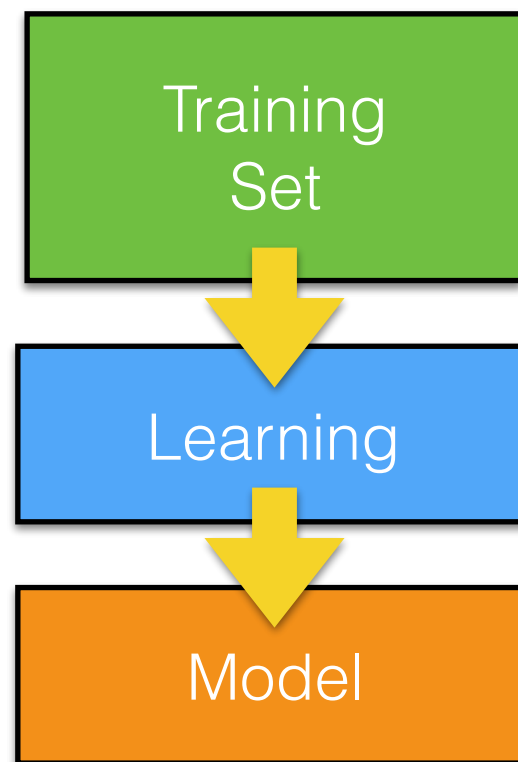
Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!



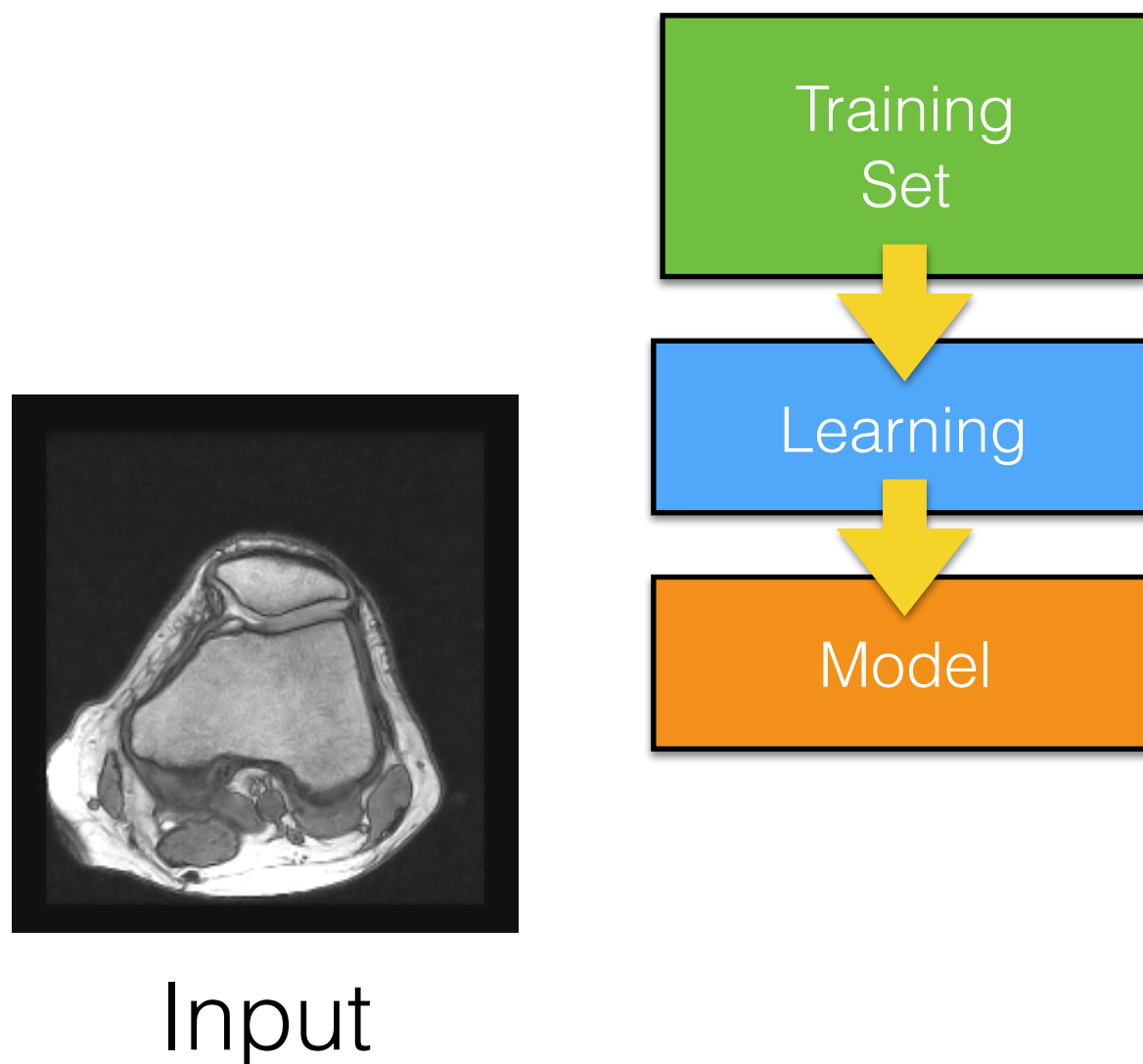
Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!



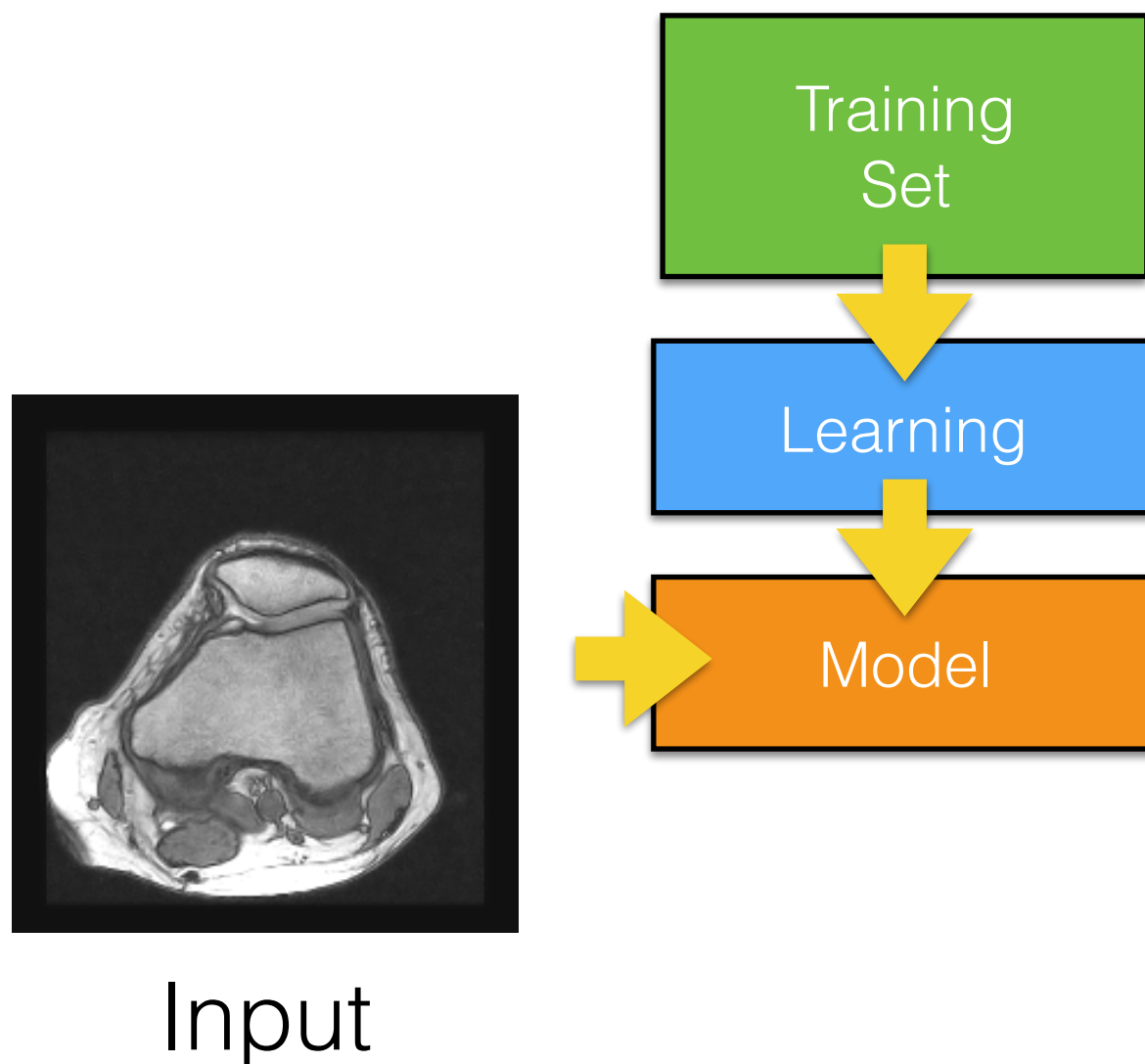
Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!



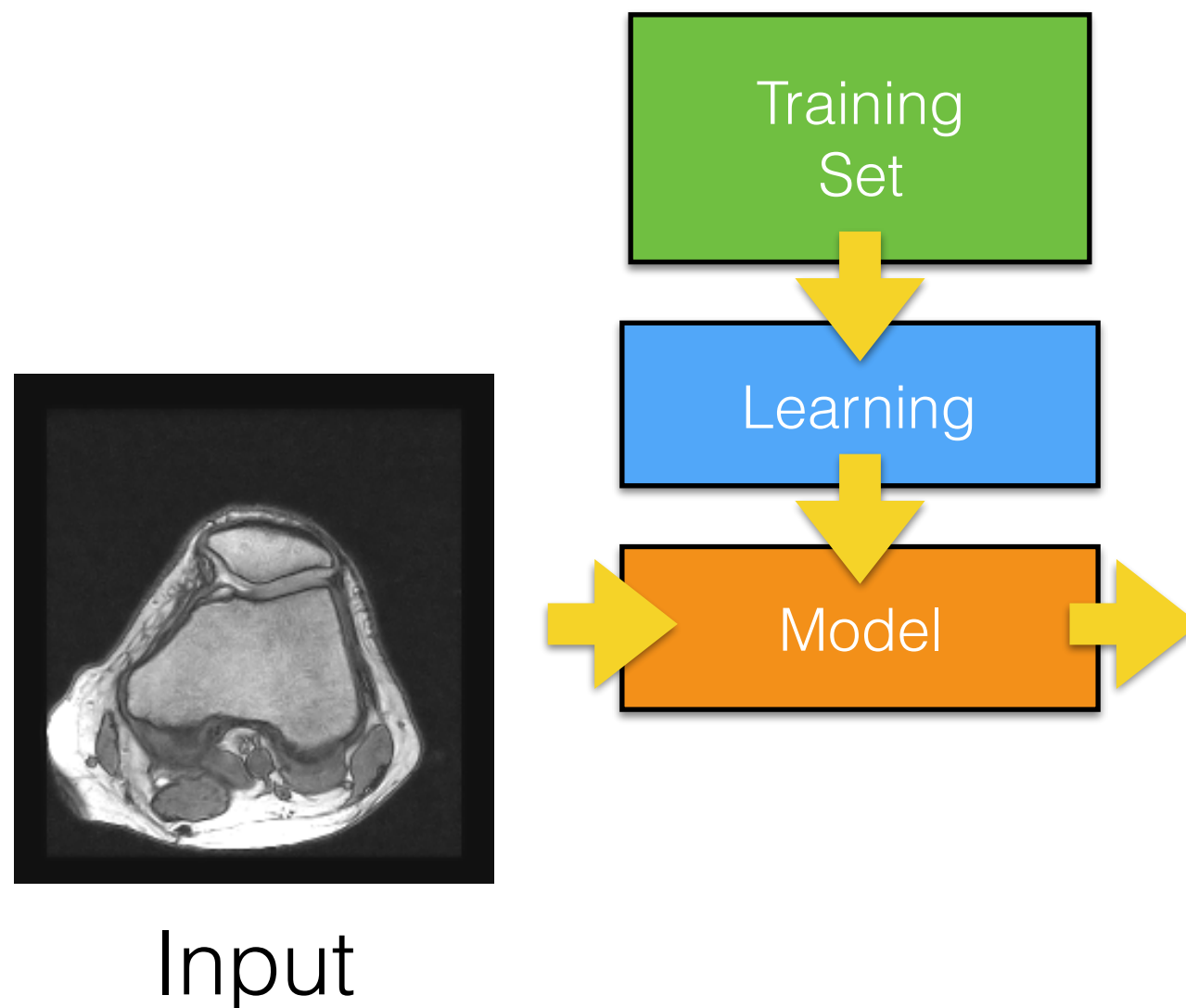
Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!



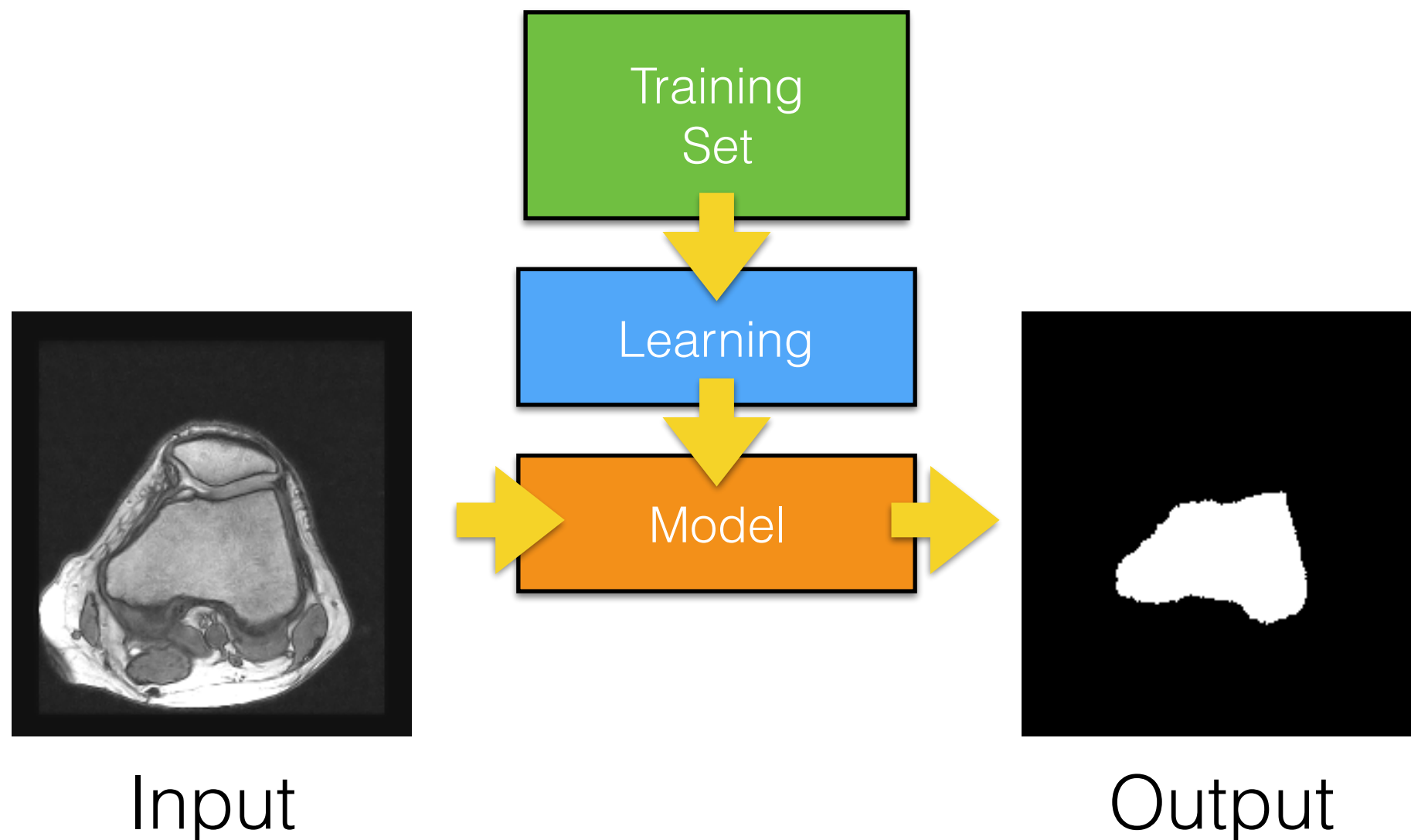
Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!



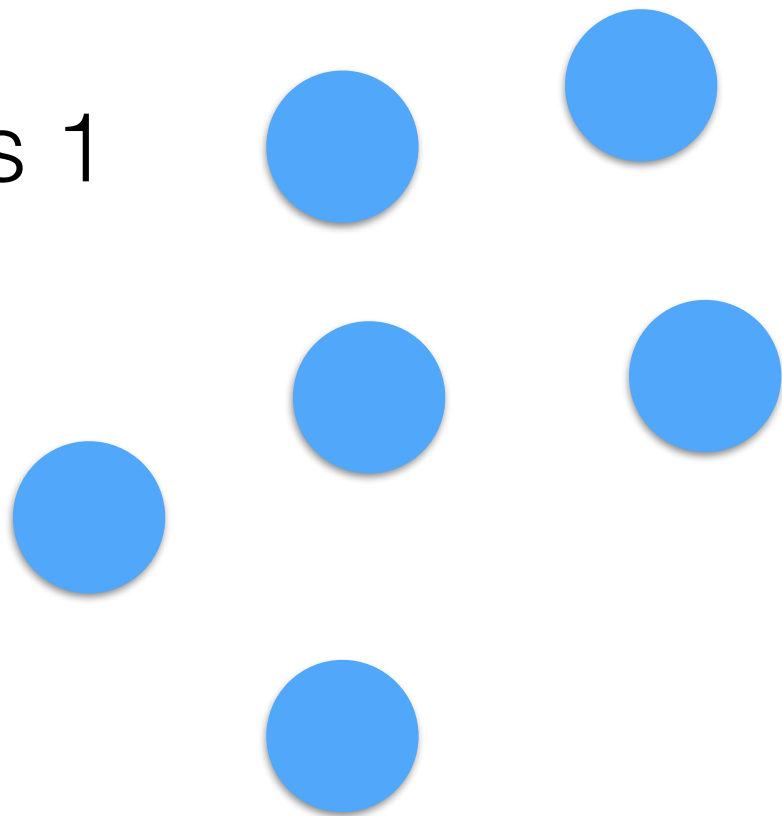
Machine Learning

- Machine learning algorithms work very well for classification. Therefore, for segmentation too!

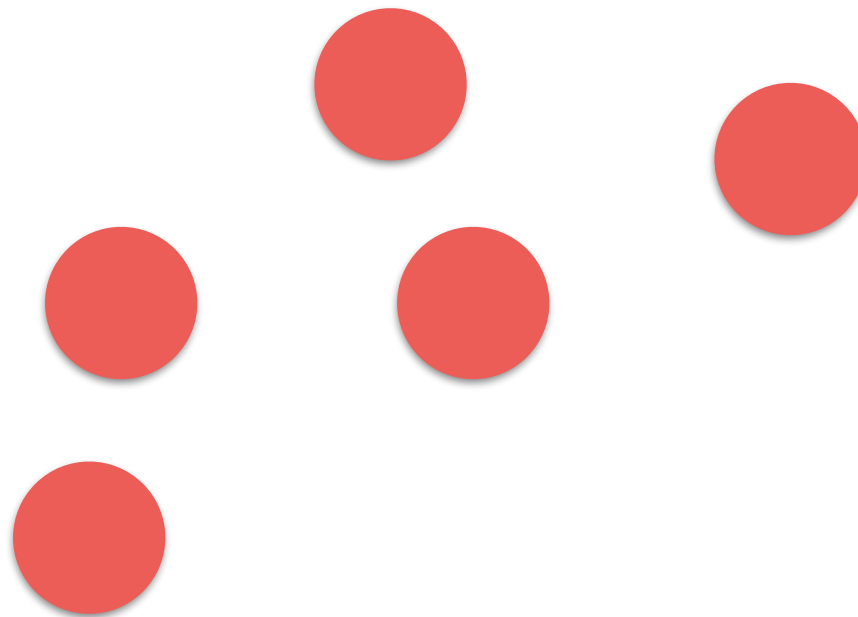


Machine Learning

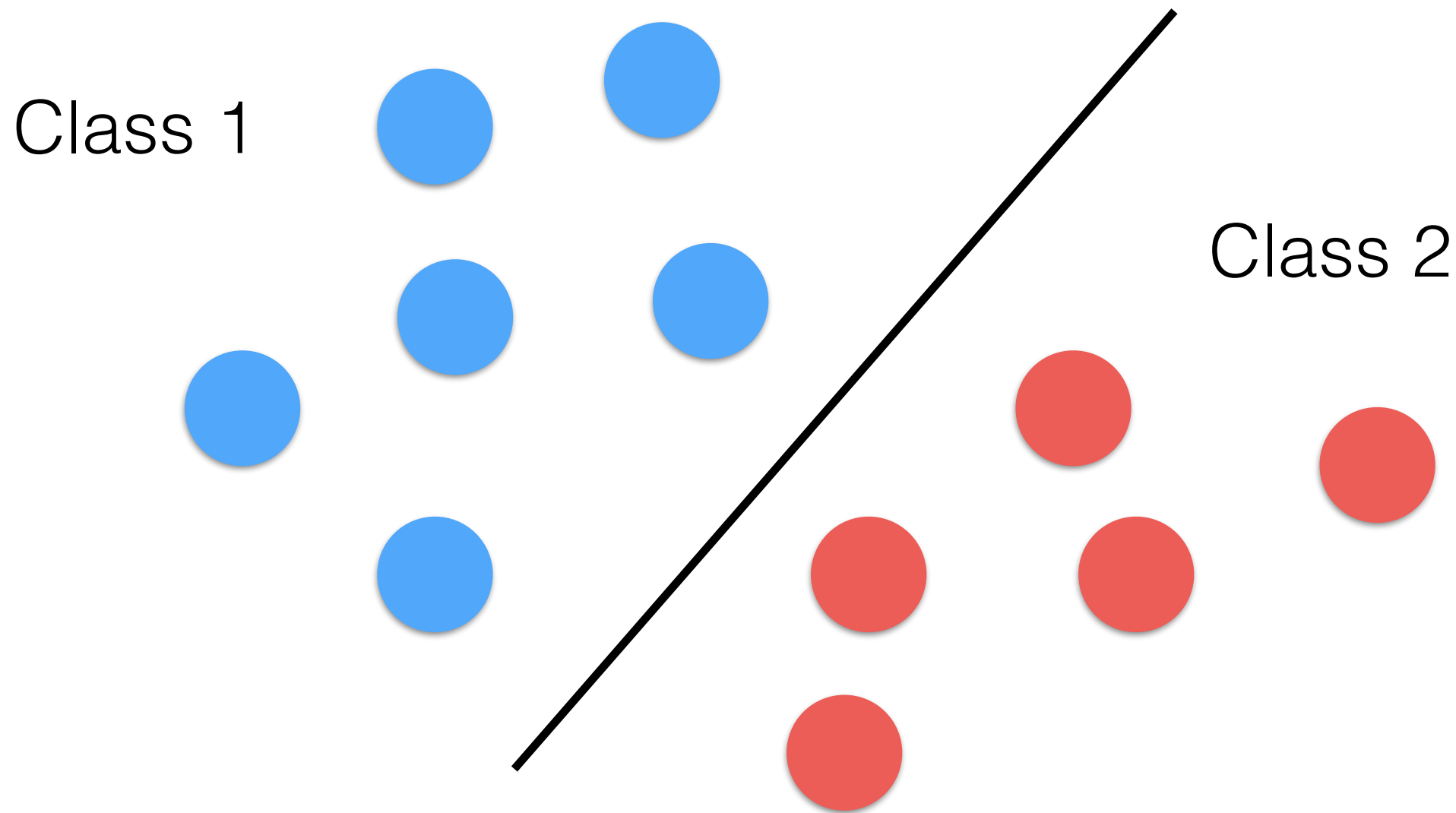
Class 1



Class 2



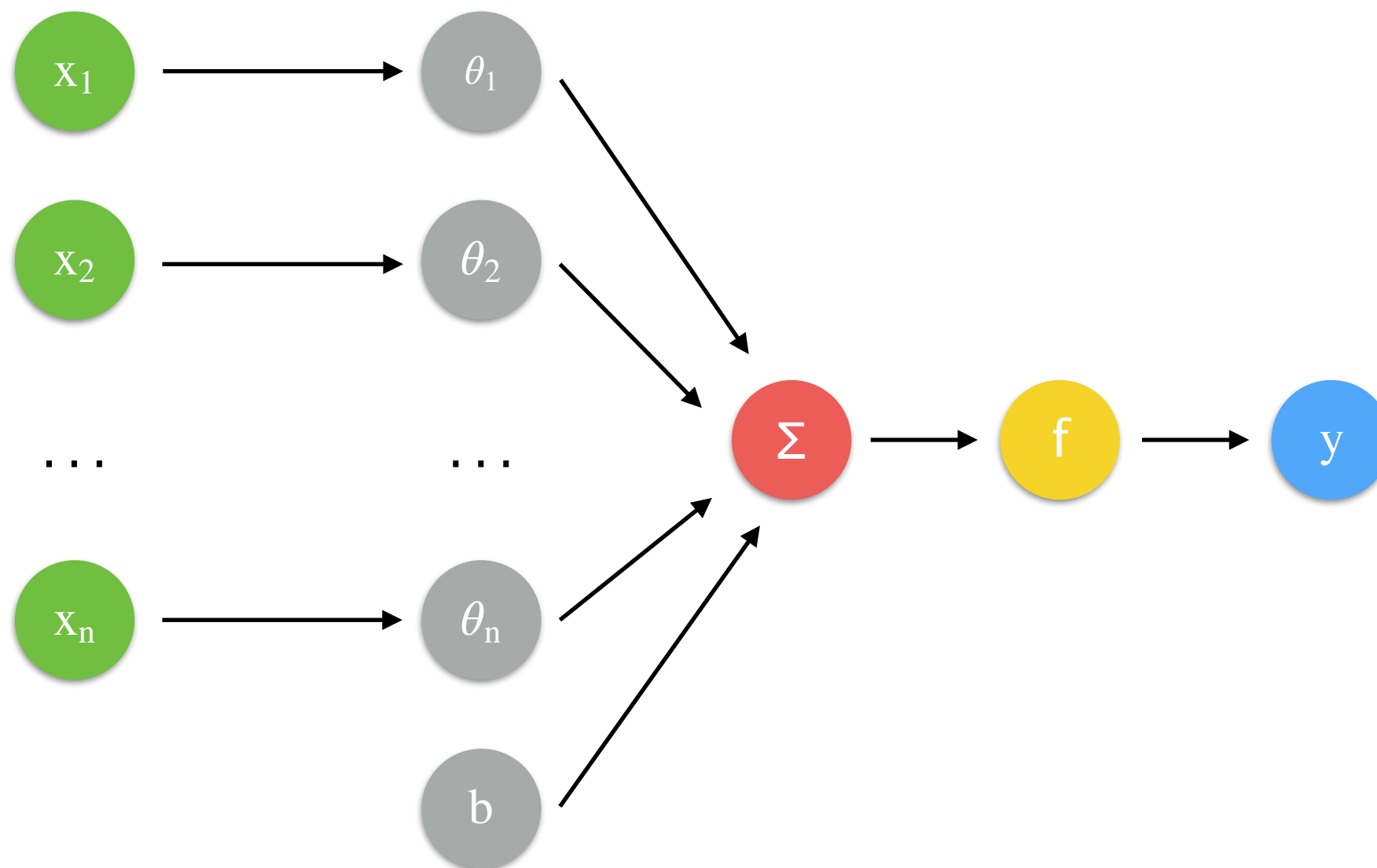
Machine Learning



Machine Learning: Neural Networks

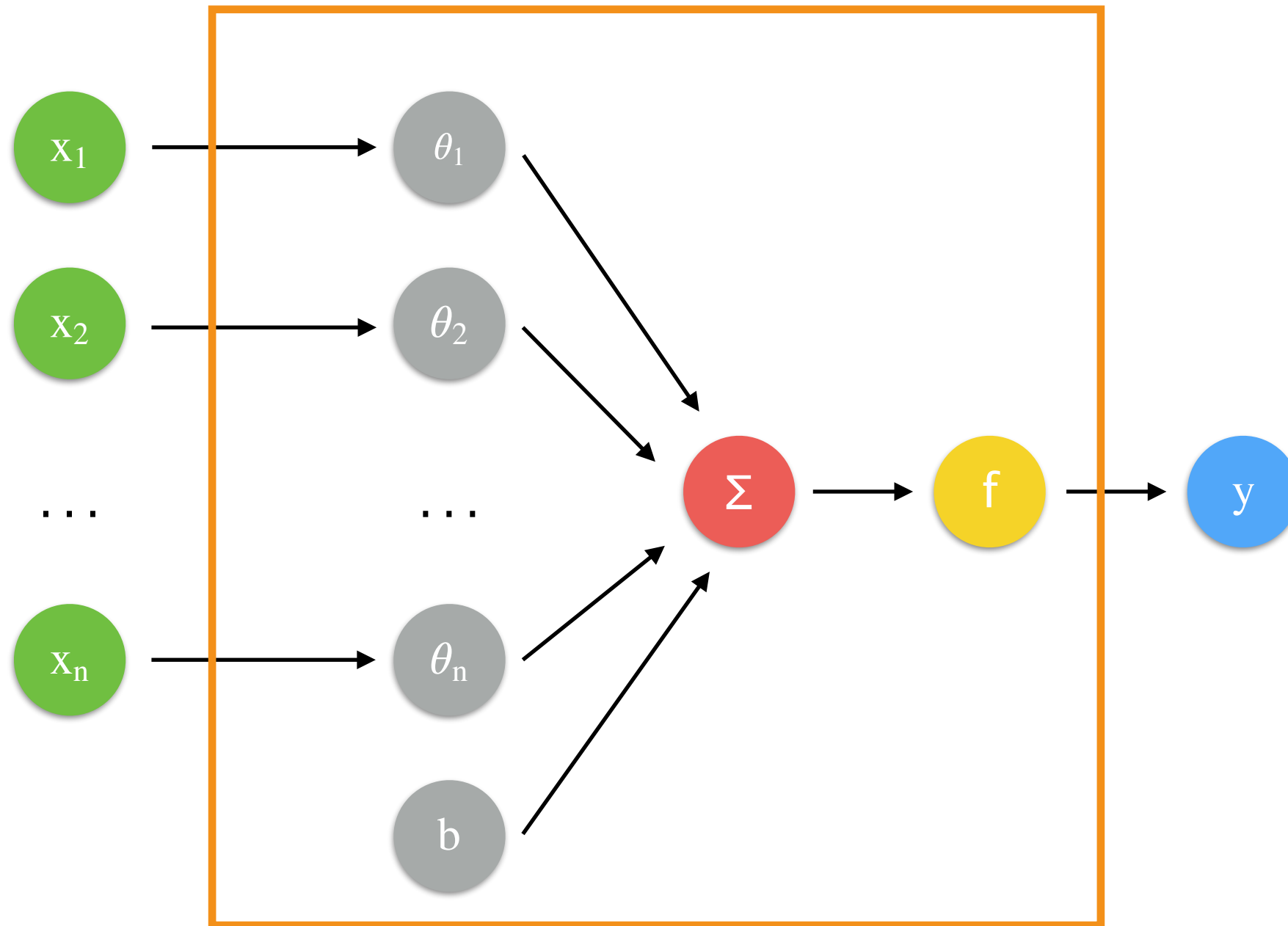
- The idea is to “mimic the neurons” in our brain.
- They work very well for binary classification, our case!

Neural Networks: Logistic Regression



$$y = f(\Theta \cdot \mathbf{x}^\top + b)$$

Neural Networks: Logistic Regression



$$y = f(\Theta \cdot \mathbf{x}^\top + b)$$

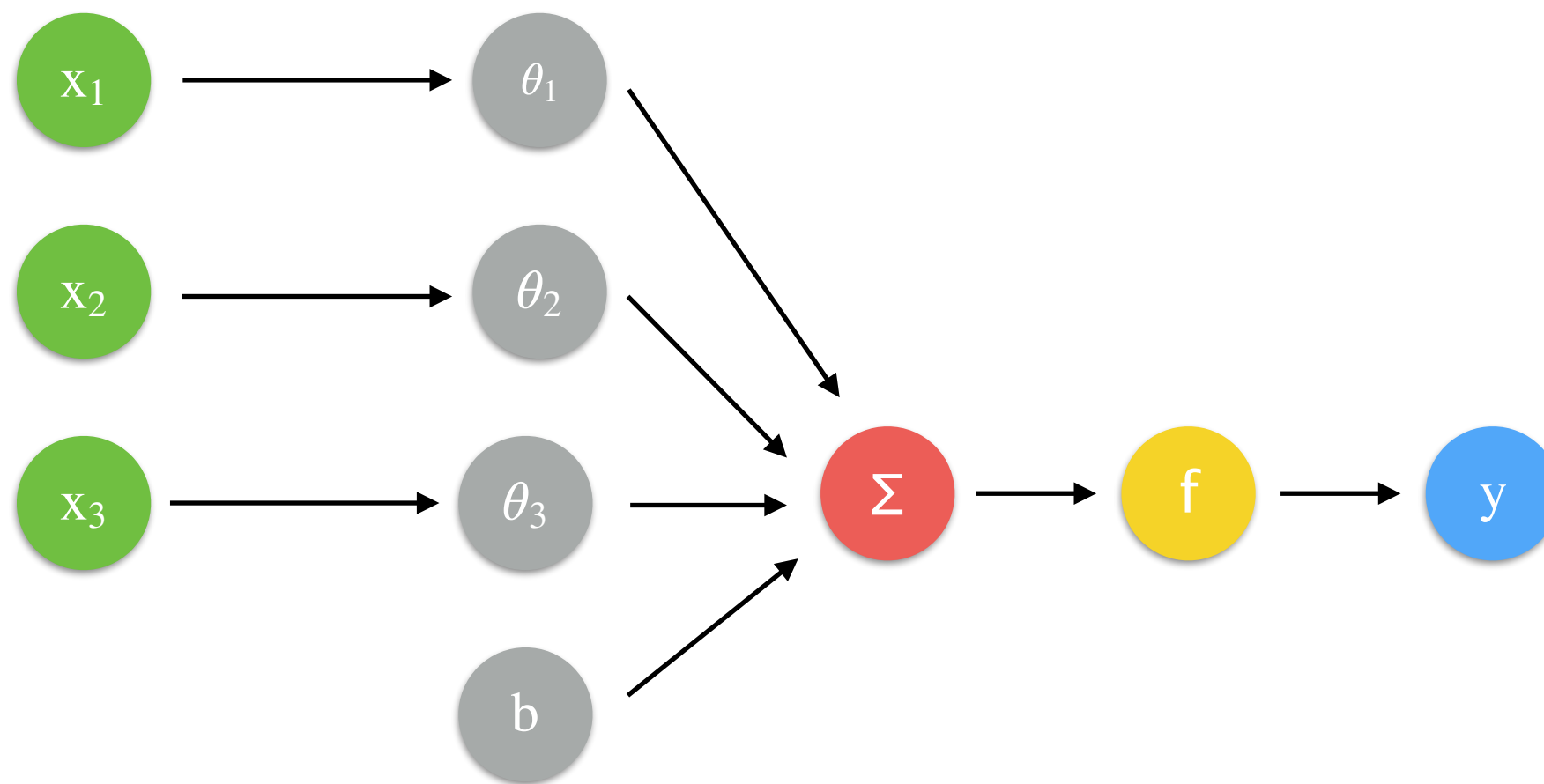
Neural Networks: Logistic Regression

- f is called the activation function.
- It can be defined in many ways. For example:

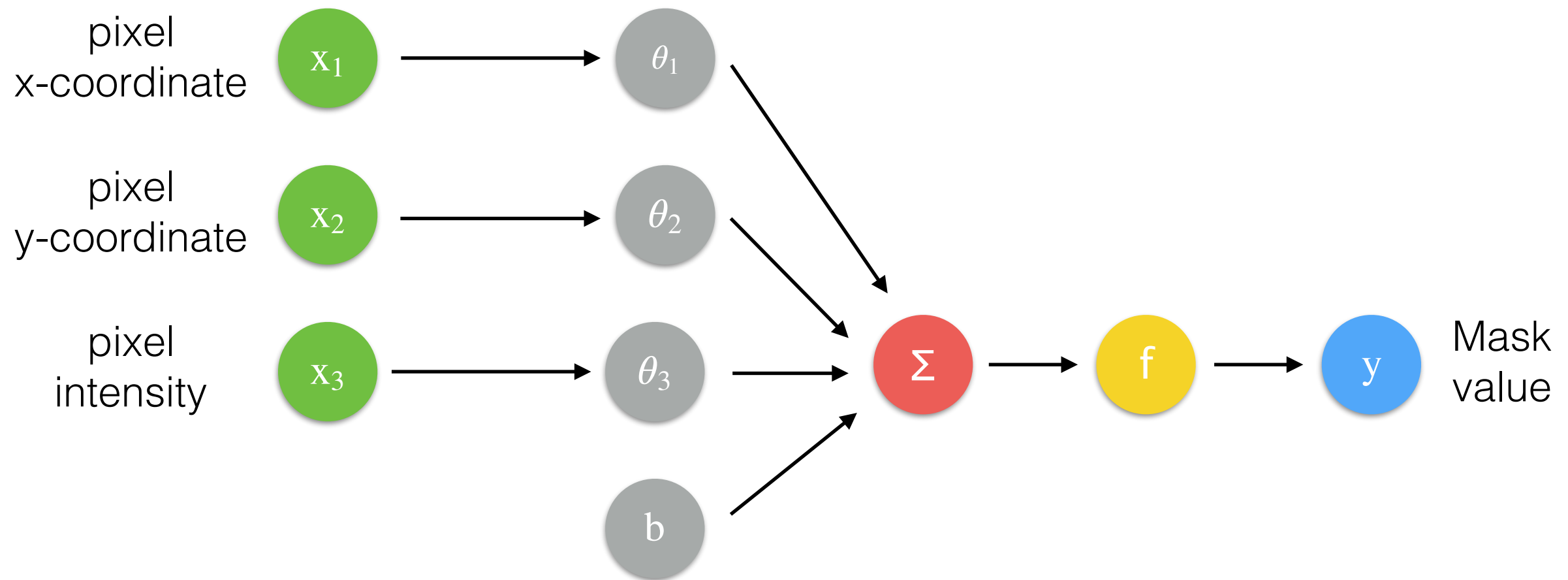
$$f(z) = \frac{1}{1 + e^{-z}} \quad f(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ 0 & \text{otherwise.} \end{cases}$$

- This is because the result has to be either belonging or not to a class (i.e., our area of interest).

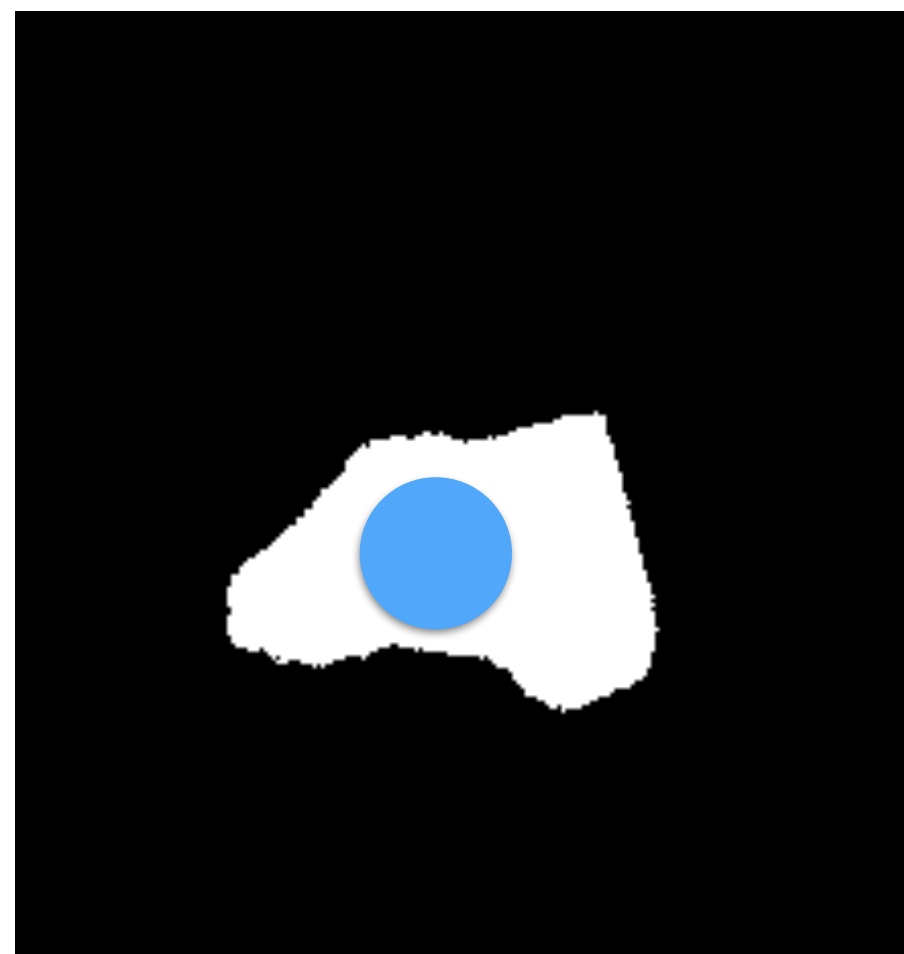
Neural Networks: A Toy Example



Neural Networks: A Toy Example



Neural Networks: Training Sample Example

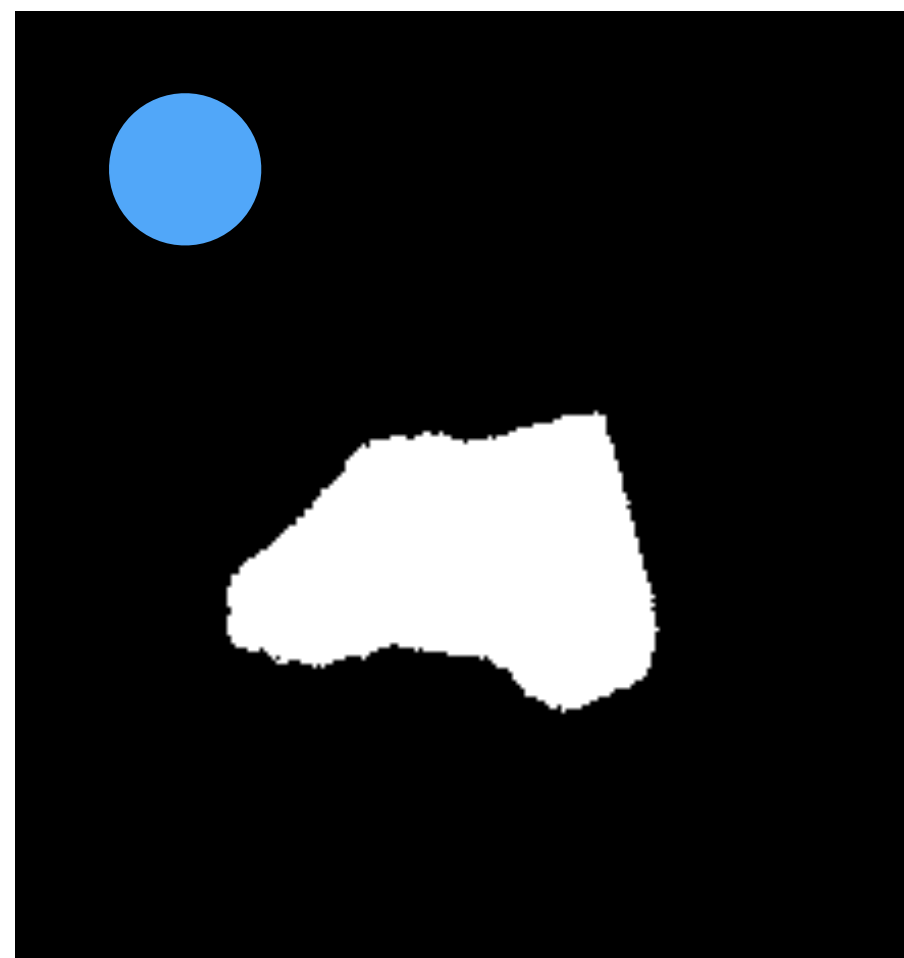


$$\mathbf{x} = \{100, 100, 0.78\}$$

$$y = 1$$

Assuming $n = 3$

Neural Networks: Training Sample Example



$$\mathbf{x} = \{20, 20, 0.039\}$$

$$y = 0$$

Assuming $n = 3$

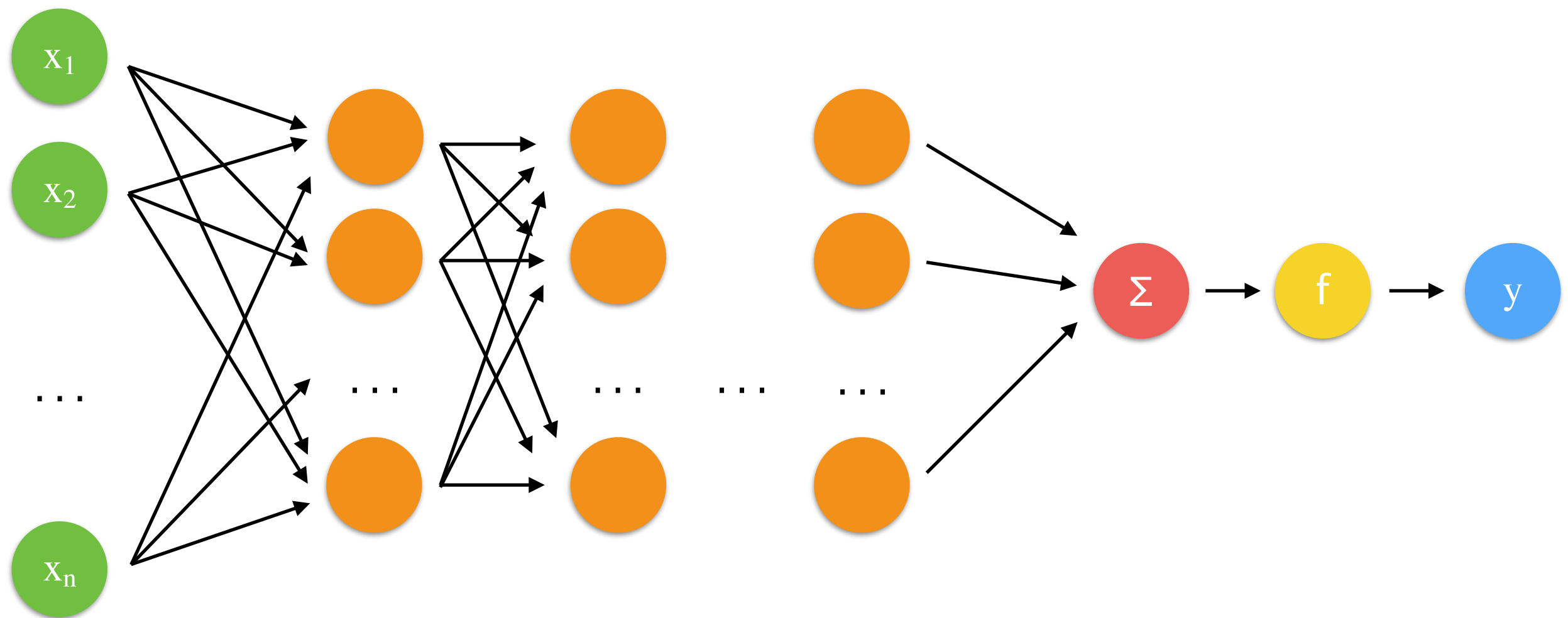
Neural Networks: Learning

- We need to collect m samples.
- We need to minimize an error function J :

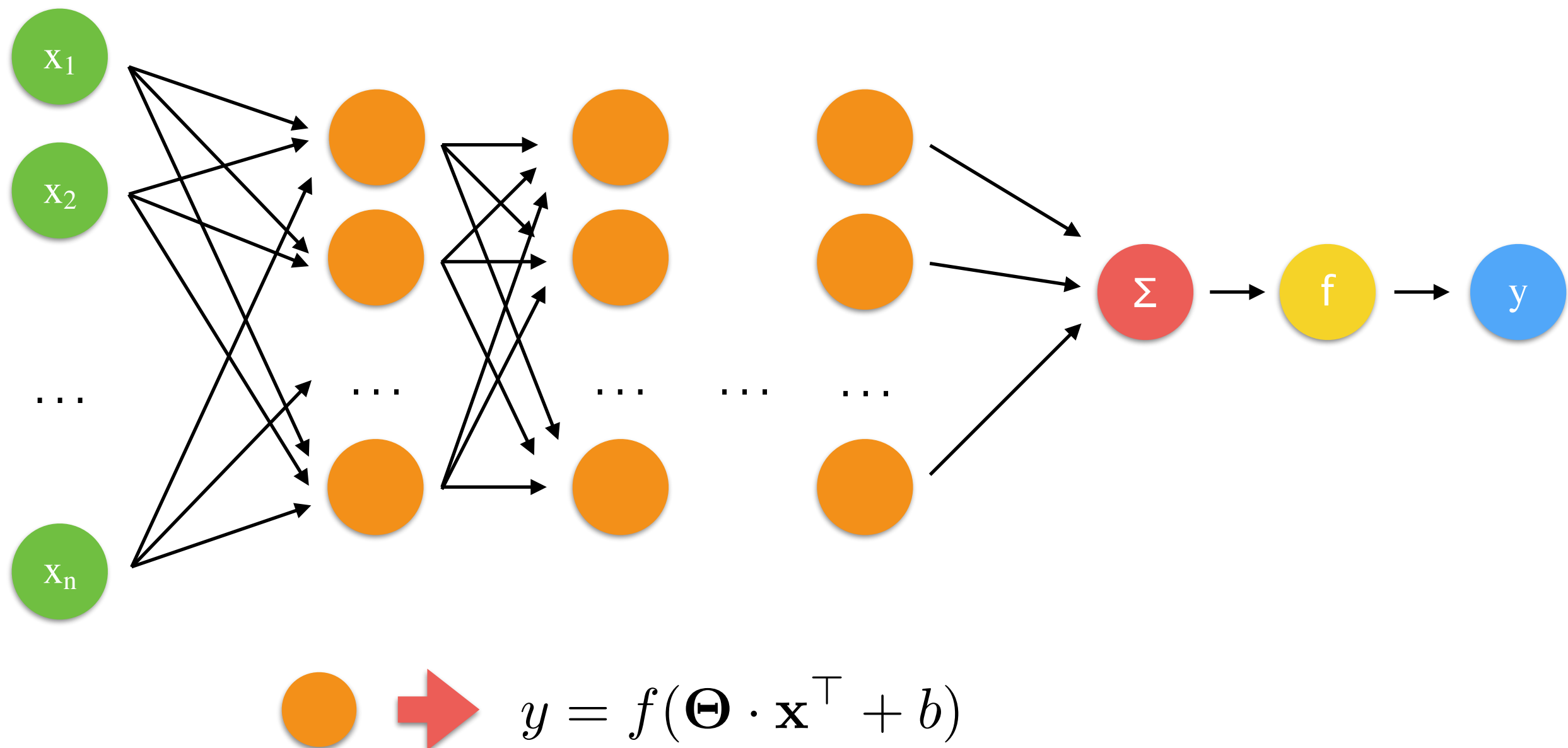
$$J(\Theta) = \frac{1}{2} \sum_{i=1}^m \left(f(\mathbf{x}^i \cdot \Theta^\top + b) - y^i \right)^2 \text{ with } f(x) = x$$

- How do we minimize it?
 - Gradient descent
 - Starting solution for theta? Random values in $[0,1]$!

Neural Networks: Bigger Networks

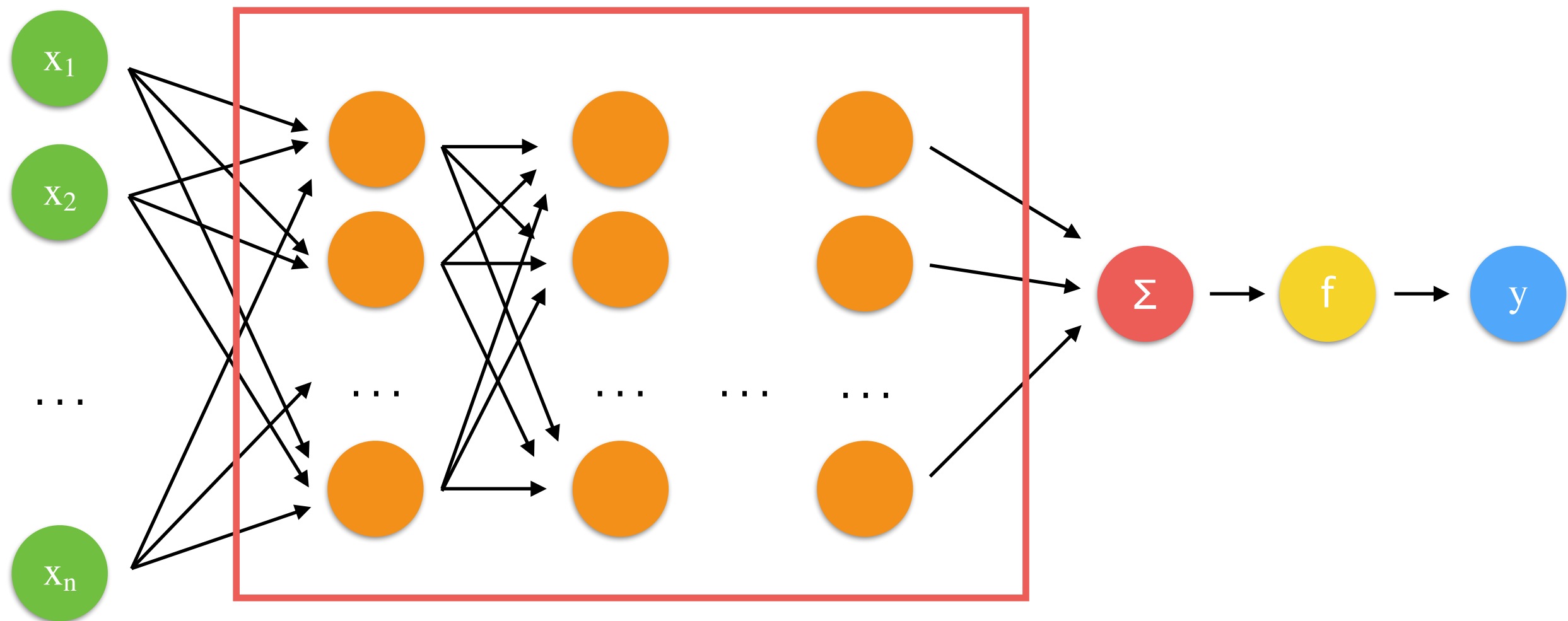


Neural Networks: Bigger Networks



Neural Networks: Bigger Networks

Hidden Layers



  $y = f(\Theta \cdot \mathbf{x}^\top + b)$

Neural Networks

- Advantages:
 - fully automatic!
 - computationally fast to evaluate (not the learning though); especially using GPUs.
- Disadvantages:
 - they required many many examples: more than 1,000 to get some decent result; better $>10,000$ training example!

that's all folks!